

Introduction to Edge Computing



Dr. Rajiv Misra

Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

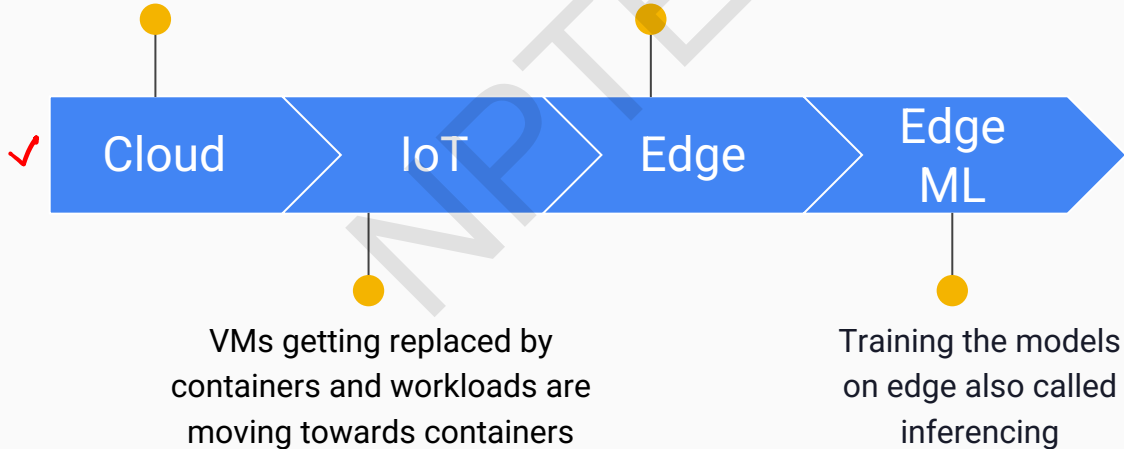
After this lecture you will be knowing following things:

- Introduction to Edge Computing
- Edge Computing Architecture & building blocks
- Edge Computing for IOT
- Advantages of Edge Computing

Recapitulate: Evolution of Cloud

Virtual machines running in a remote data center or storage that was offered in a remote data center

Data processed locally and compute comes much closer to the devices or the sources of data



Introduction to Edge Computing

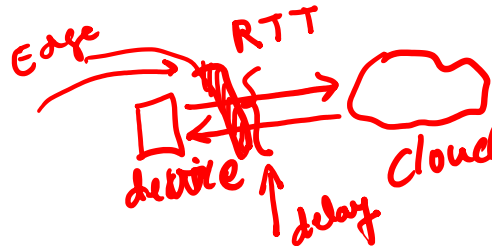
Edge computing allows the cloud to be genuinely distributed. ✓

Don't need to rely on the cloud for all the processing and data aggregation collection processing and querying. ✓

Mimics the public cloud platform capabilities. ✓

✓ Reduces the latency by avoiding the round-trip and brings in the data sovereignty by keeping data where it actually belongs. ✓

✓ Delivers local storage, compute, and network services.



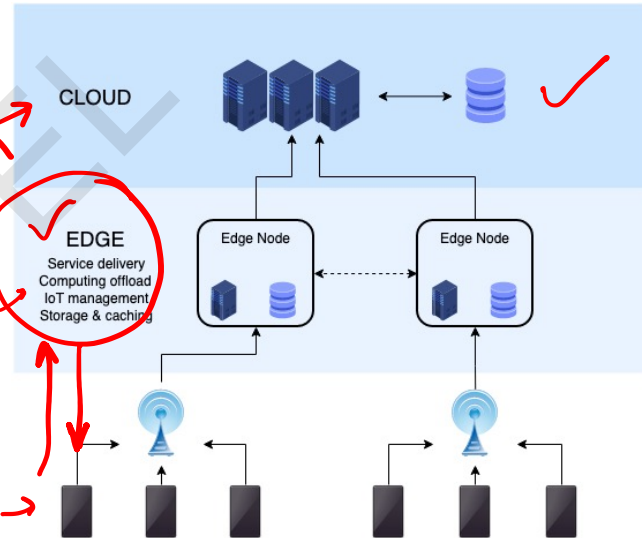
Edge Computing: makes distributed cloud ✓

● Edge computing makes the cloud truly distributed. The current cloud or rather the previous generation of cloud was almost like a mainframe or like a client-server architecture where very little processing was done on the client side but all the heavy lifting was done by the cloud. //

● With all the innovations in the hardware chips and with the affordable electronics and silicon it makes more sense to bring compute down to the last mile and actually keep the compute closer to the devices.

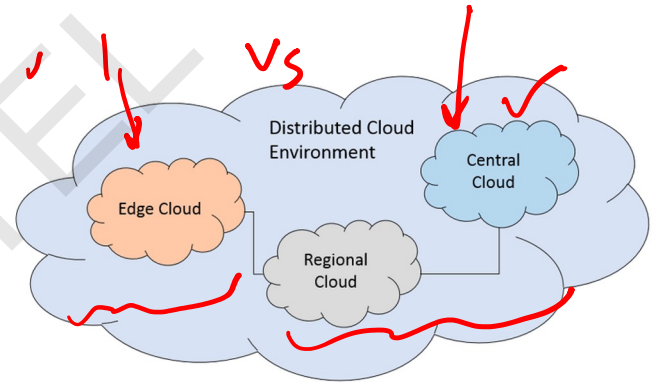
● So that's when edge computing becomes more and more viable where you don't need to rely on the cloud for all the processing and data aggregation, collection, processing and querying instead you could actually run a computing layer that is very close to the devices.

→ edge layer
devices



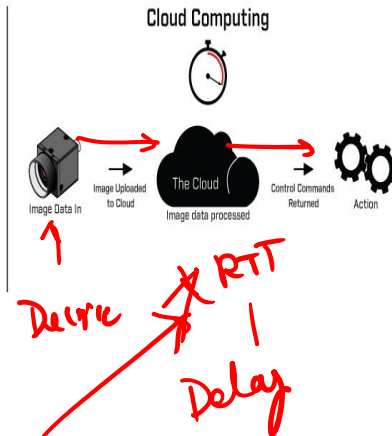
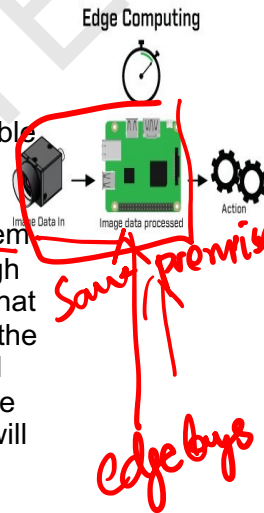
Edge Computing: Mimics the public cloud platform capabilities and Move cloud service closer to data-source

- The edge computing mimics the public cloud platform capabilities
- For example when you dissect an edge computing platform you would notice that it almost has all the capabilities of a typical public cloud
- IOT pass: it has device management, it has data ingestion, it has stream analytics and it can run machine learning models and it can run server less functions so all of those are capabilities that are predominantly available on the public cloud
- But with edge computing they all come to the last mile delivery point and run very close to the source of the data which is sensors actuators and devices



Edge Computing: Reduces the latency by avoiding the round-trip and brings in the data sovereignty

- The biggest advantage of deploying an edge computing layer is that it reduces the latency by avoiding the round-trip.
- It also brings in the data sovereignty by keeping data where it actually belongs to.
- For example in a healthcare scenario it may not be viable or it may not be compliant to actually stream sensitive patient data to the cloud where it is getting stored and processed instead the patient data should remain on-Prem within the hospital premises but it still needs to go through lot of processing and find out very useful insights so in that case the edge computing layer is going to stay close to the healthcare equipment with connectivity back to the cloud and the architects and the customer engineers will decide what data will stay within the edge boundary and what will actually cross that and move to the cloud may be anonymized data.



Edge Computing Building Blocks

Data Ingestion ✓

M2M Brokers ✓

Object Storage

Function as a Service

NoSQL/Time-Series Database

Stream Processing

ML Models

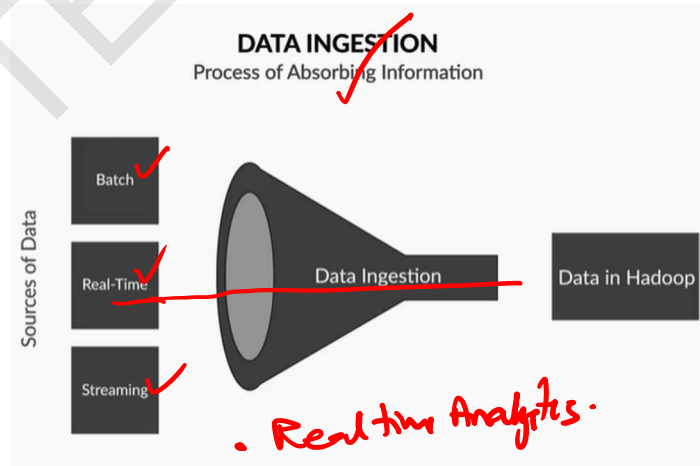
(minimize the
Public Cloud
Dependency)

Edge Computing Building Blocks: Data Ingestion

Data Ingestion:

This is the high velocity, high throughput data endpoint like the Kafka endpoint that is going to ingest the data.

It is the process of obtaining and importing data for immediate use or storage in a database. To ingest something is to take something in or absorb something. Data can be streamed in real time or ingested in batches. In real-time data ingestion, each data item is imported as the source emits it.

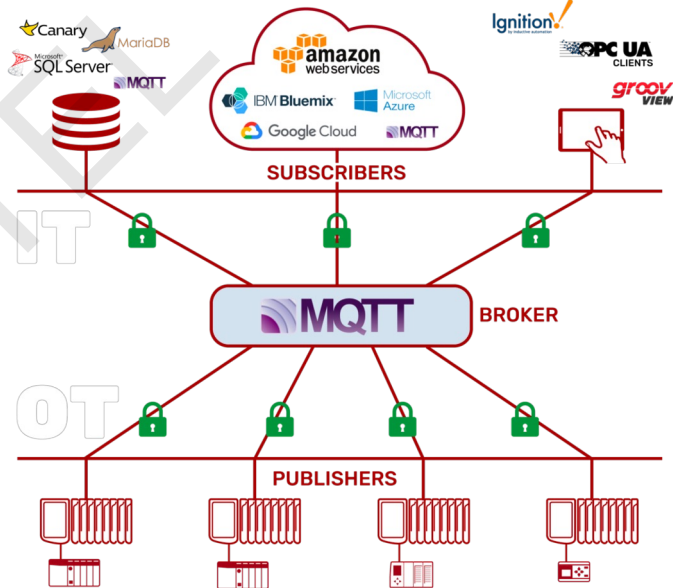


Edge Computing Building Blocks: M2M Brokers

M2M Brokers:

Edge will also run message brokers that will orchestrate machine to machine communication.

For example device one talks to device two via the M2M broker.



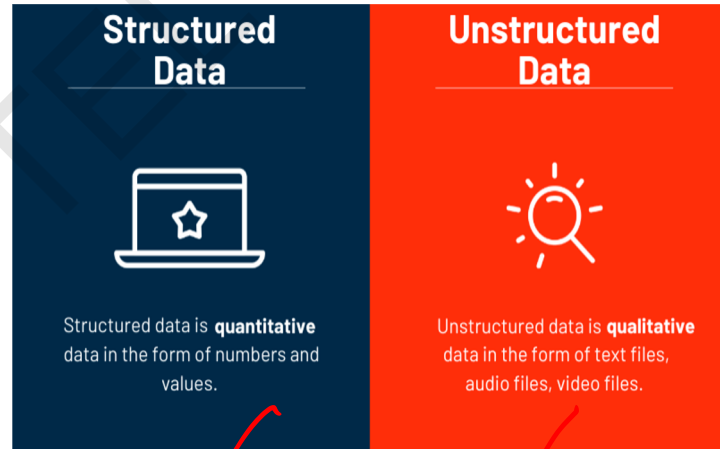
Edge Computing Building Blocks: Storage ✓

Object Storage: ✓

there may be unstructured storage particularly to store the feed from video cameras and mics and anything that is unstructured will go into object storage.

NoSQL/Time-Series Database: ✓

More structured data goes into time series data base and no sequel database ✓

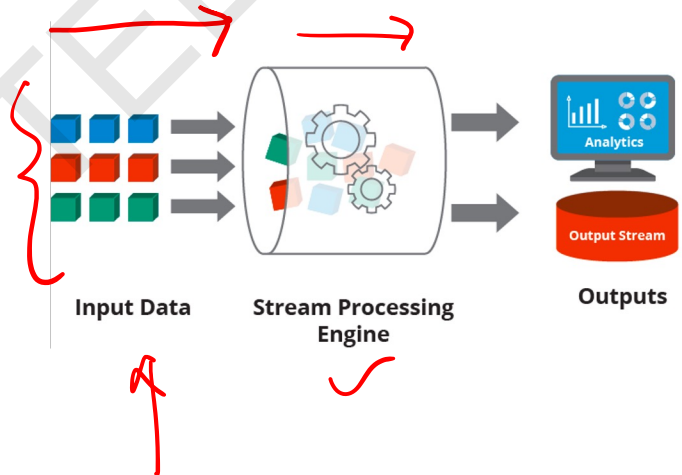


Edge Computing Building Blocks: Stream Processing

Stream Processing:

It is a complex event processing engine that is enabling you to perform real-time queries and process the data as it comes. ✓

For example for every data point you want to convert Fahrenheit to Celsius or you want to convert the timestamp from one format to another, you could do it either in stream processing.



Edge Computing Building Blocks: Function as a Service

Function as a service:

To add additional business logic there is a functions as a service which is actually responsible for running lightweight compute.

If you need to do more sophisticated code you could actually move that to **functions as a service**.



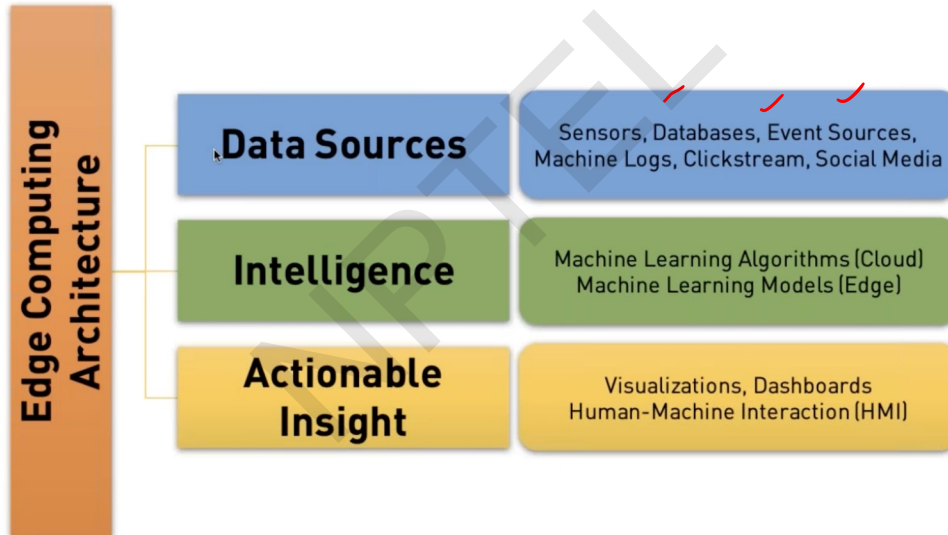
Edge Computing Building Blocks: ML Models

Machine Learning models:

Lastly, there is an ML runtime for example most of the computing platforms are capable of running tensorflow light, cafe models and pitorch models, so you can actually process the data that comes in more intelligently and take preventive measures and perform predictive analytics.



Edge Computing Architecture



Edge Computing: Three-tier Architecture

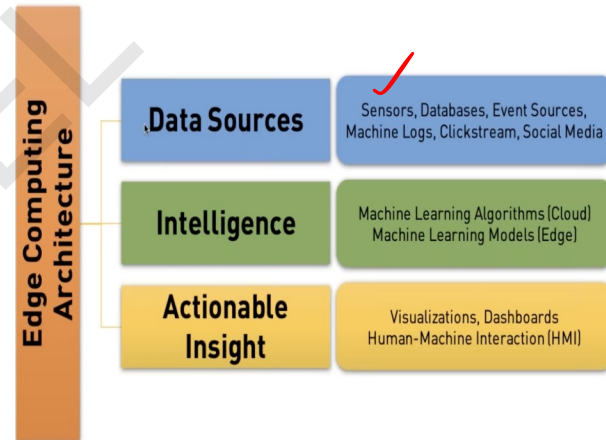
Now let's look at this from a different dimension. ✓

There are data sources and by the way edge computing is not confined just to IOT, it could be even for non IOT use cases. Anything that generates data can be fed into an IOT like cameras, clickstream analysis, gaming, etc.

A lot of use cases are relevant for the edge deployments so it's basically like a **three-tier architecture**. ✓

But this three-tier architecture is **not** the traditional three-tier that we are familiar of.

There is **no app server, no database, no middle layer, and there is no front end**, so this is not a traditional three tier architecture.

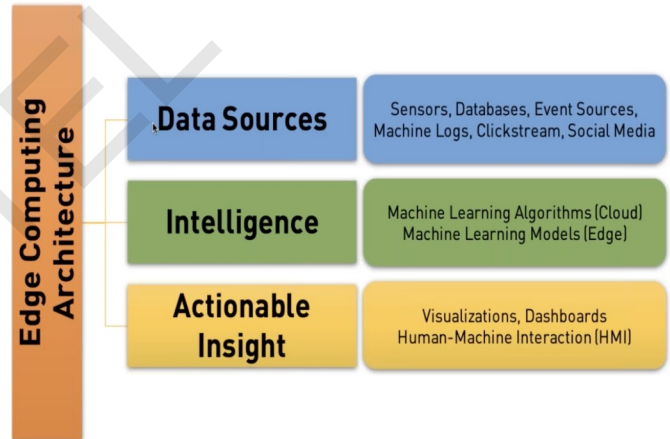


Edge Computing Architecture: Data Source Tier

The first tier is the **Data Source Tier**: ✓

In industrial IOT environment, this could be a set of devices that are generating the data.

These are nothing but original endpoint, from where the data is acquired or the origin of the data.

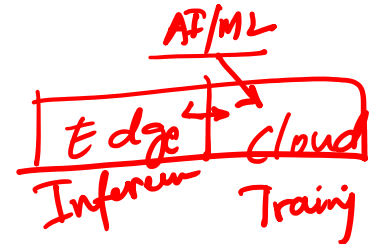
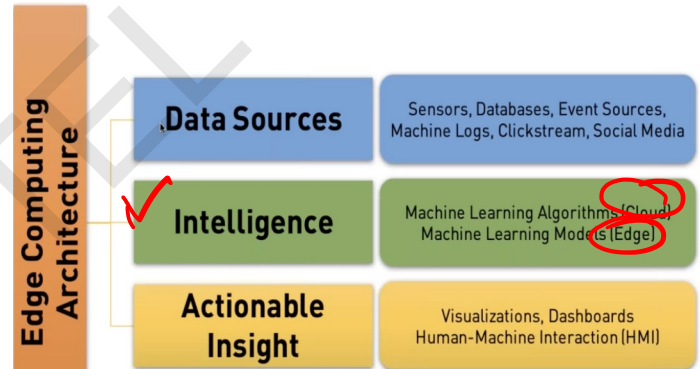


Edge Computing Architecture: Intelligence Tier

Then there is **an intelligent tier:** ✓

Responsible for running the machine learning models.

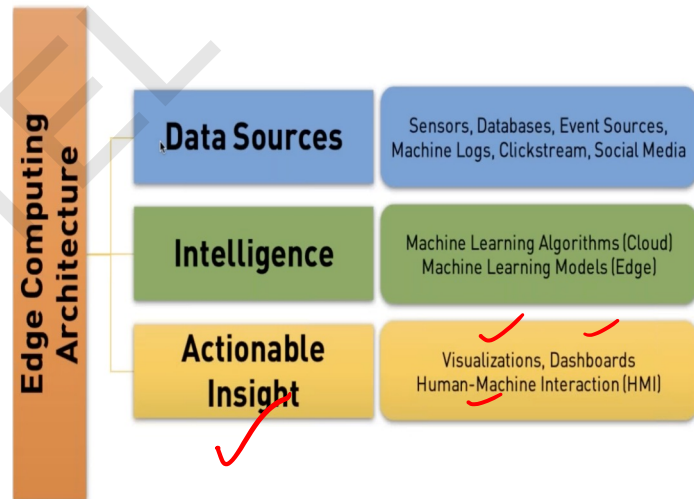
This intelligent tier cuts across the cloud and the edge so there is a very **well-defined boundary between edge and cloud** where the training takes place on the cloud ✓ and the inferencing is run on the edge. But collectively, this overlap between the cloud and the edge is this intelligence layer.



Edge Computing Architecture: Actionable Insight Tier

Then there is an **actionable insight layer**:

Responsible for sending an alert to the relevant stakeholders or populating the dashboards and showing some visualizations or even the edge taking an action to immediately shut down a faulty machine or controlling an actuator and again the actionable insight takes place on the edge so this is not the physical boundary.

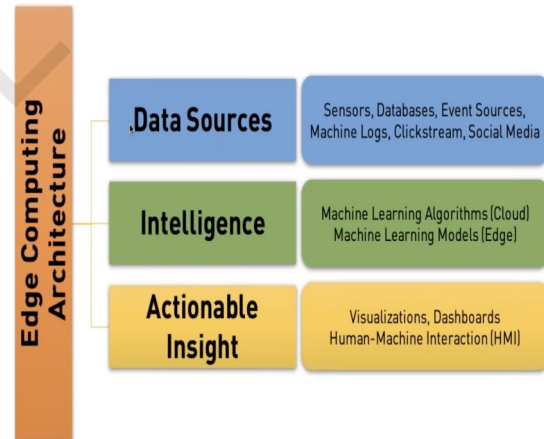


Edge Computing Architecture: Summary

In **Summary**, you logically look at the whole architecture so there is a data source which is the original endpoint from where the data is acquired or the origin of the data.

Then there is an intelligence layer where the constant training and inferencing takes place.

Then there is an insight layer where you actually visualize the outcome from the intelligence and also perform actions based on those insights so that is one way of visualizing edge computing.



Lecture Summary

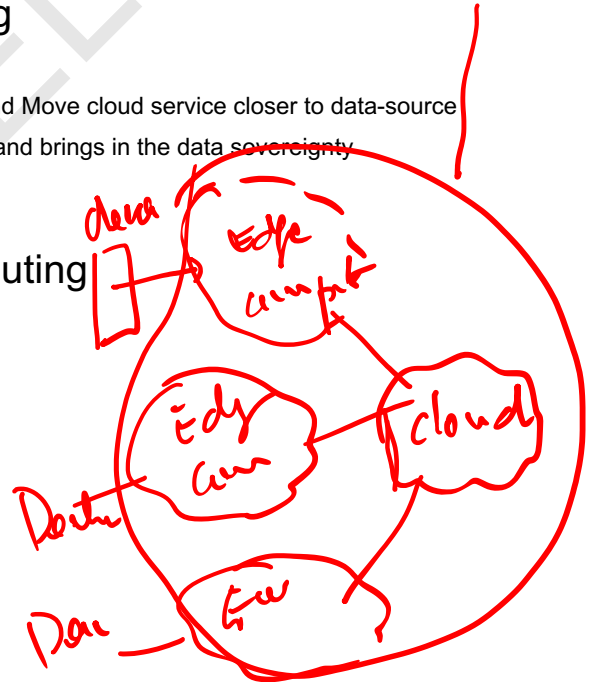
- In depth concepts of Edge Computing

- Edge makes distributed cloud
- Edge mimics the public cloud platform capabilities and Move cloud service closer to data-source
- Edge reduces the latency by avoiding the round-trip and brings in the data sovereignty

- Building Blocks of Edge Computing

- Three tier architecture of Edge Computing

- Data Source
- Intelligence
- Actionable Insight





THANK YOU!

Introduction to Cloud



Dr. Rajiv Misra

Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

On completion of this Lecture you will get to know about the following:

- Understanding of today's cloud scenario
- Different objectives of cloud
- Current limitations of traditional cloud
- Why there is a need of Edge Computing?

Current State of Today's Cloud

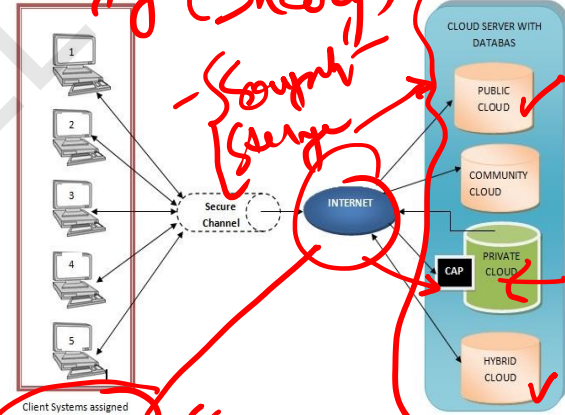
- Highly centralised set of resources
- Compute is going beyond VMs
- Storage is complemented by CDN
- Network stack is programmable
- The Web and Software-as-a-Service
- Infrastructure-as-a-Service
- High-Availability cloud



Current State of Today's Cloud: Highly Centralized in Client-Server Architecture

- ✓ Cloud computing started as all about virtual machines that were running in a remote data center (or storage).
- Highly centralized architecture closely resembles 90s client-server computing.
- For example cloud (the remote data center or the remote infrastructure) exposed by Amazon, Microsoft, Google, IBM and others is the server and the machine from which you are connecting to it and consuming the cloud resources is the client.

Virtualization Technology (Sharing)



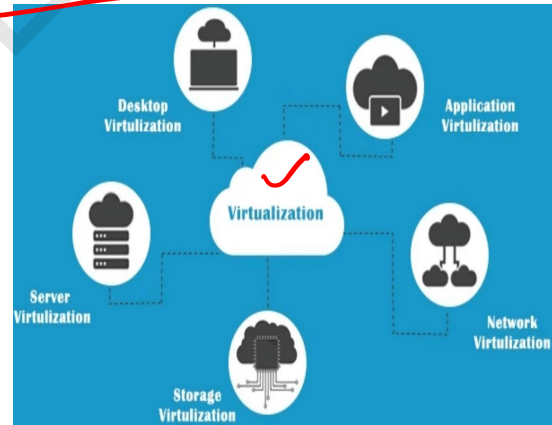
Summarize
Virtualization
- Hypervisor

Private Cloud
Ex - IIT Kanpur
(DC), IITS

✓ Server
(Cloud Data Center)
- Capacity Sharing

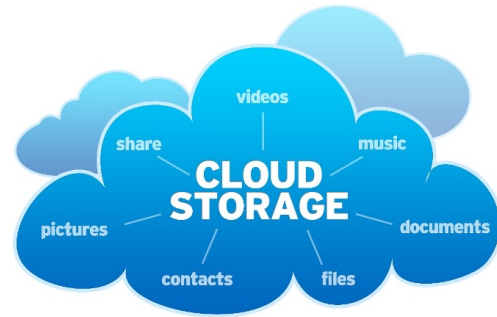
Current State of Today's Cloud: Compute is going beyond VMs IaaS / Containers

- ✓ Although cloud resembles the 90s client-server computing but at the same time compute has gone beyond VMs the first generation of cloud was all about VM virtual machines.
- ✓ Where you could programmatically launch a VM and you could SSH into it and take control of the Virtual Machine and install the software.
- ✓ But there is a dramatic shift in the compute where VMs are slowly getting replaced by containers.
- ✓ More and more workloads are moving towards containers.



Current State of Today's Cloud: Storage is complemented by CDN — *Content delivery Network.*

- Another important trend almost all the public cloud are in storage offerings.
- Object storage is complemented by a content delivery network today.
- Whenever you put an object in a bucket or a container of the public cloud storage you can click a check box to basically replicate and cache the data across multiple edge locations but this edge is not the edge that we are talking about this is the content delivery network where it caches the frequently accessed content in a set of pop or edge locations.

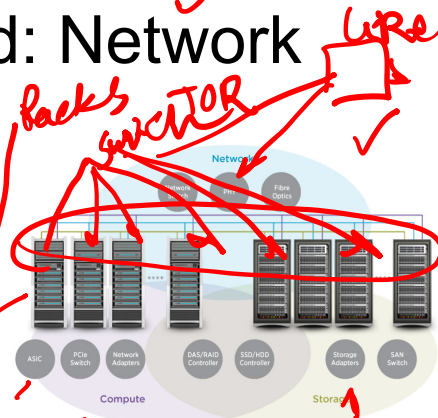


edge location of CDN means caching (NOT to be understood as edge computing)

Current State of Today's Cloud: Network stack is programmable

- Finally network has become extremely programmable today.
- If you look at the hybrid cloud, multi-cloud scenarios and how network traffic is getting routed and how load balancers firewalls and a variety of network components are configured it is through api's and programmability.
- The same capability of SDN is enabling hybrid scenarios particularly when we look at the combination of software-defined network with some of the emerging networking technologies.
- These mesh they are opening up additional avenues some of the very recent trends like Google's Anthos, IBM cloud private and some of the other container based hybrid cloud platforms are heavily relying on the programmable Network stack and also a combination of SDN with service mesh.
- This is the current state of the cloud and these trends represent how the cloud is currently being consumed or how it is delivered to customers but cloud is going through a huge transformation.

Shared { Tree-like hvin
→ network



SDN ← Control Plane
Data plane (DP)

Multiple waves of innovation in Cloud: Pass to IOT (Internet of Things) ✓

- Initially cloud was all about compute storage and network resources globally available highly centralized set of resources because cloud made compute and storage extremely cheap and affordable lot of industrial customers and enterprises started connecting devices to the cloud. ✓
- The data that was not persisted or aggregated or acquired is now streamed to the cloud because it is extremely cheap to store data in the cloud.
- So a lot of companies and lot of industrial environments started to take advantage of the cloud by streaming the data coming from a variety of sensors and devices.
- Also use the cheaper compute power to process those data streams and make sense out of the raw data generated these sensors and devices and that was the next big shift in the cloud this was IOT pass.

Summary

Devices → Sensors
→ Actuators

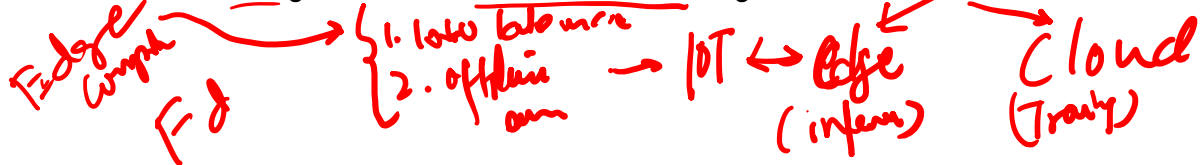
Challenges for IOT-Pass

- If you look at azure IOT, Google Cloud IOT, AWS IOT core all of them essentially give you a mechanism a platform to connect devices and store data and process it in the cloud but it was not sufficient or it was not enough to address a lot of scenarios while cloud enabled capabilities like Big Data and IOT.
- Lot of customers were not ready to move the data to the cloud that is one challenge.
- The second one is the round trip from the devices to the cloud and back to the devices was too long and it was increasing the latency in a lot of mission-critical industrial IOT scenarios.
- Sending the data to the cloud and waiting for the cloud to process it and send the results back was just not feasible so there had to be a mechanism where data could be processed locally and compute comes much closer to the devices or the sources of data so that's how IOT led to edge computing and today almost every mainstream enterprise IOT platform has a complimentary edge offering and associated edge offering and more recently there has been a lot of focus on artificial intelligence.



Cloud for AI-ML

- Today's cloud has become the logical destination for training and running artificial intelligence and machine learning models.
- Due to accelerators like GPUs GPUs FPGAs it has become extremely cheap and also powerful to train very complex very sophisticated ML models and AI models
- But in most of the scenarios a model that is restrained in the cloud is going to be run in an offline environment.
- For example, you might have trained an artificial intelligence model that can identify the make and model of a car and automatically charge the toll fee for that vehicle when it passes through the toll gate now since the toll gates are on highways and freeways with very little connectivity and almost with no network access you need to run this model in offline scenario.
- So edge computing became the boundary for running these cloud trained AI models but running in an offline mode within the edge so that is how we are basically looking at the evolution of cloud and on the waves of innovation.
- So cloud are distributed or rather decentralized platform for aggregating storing and processing data with high performance computing IOT brought in all the devices to the cloud with IOT data at edge made cloud decentralized by bringing compute closer to the data source and now it is AI that is actually driving the next wave where cloud is becoming the de facto platform for training the models and edge is becoming the de facto platform for running the models so one is called the training the other one is called inferencing.



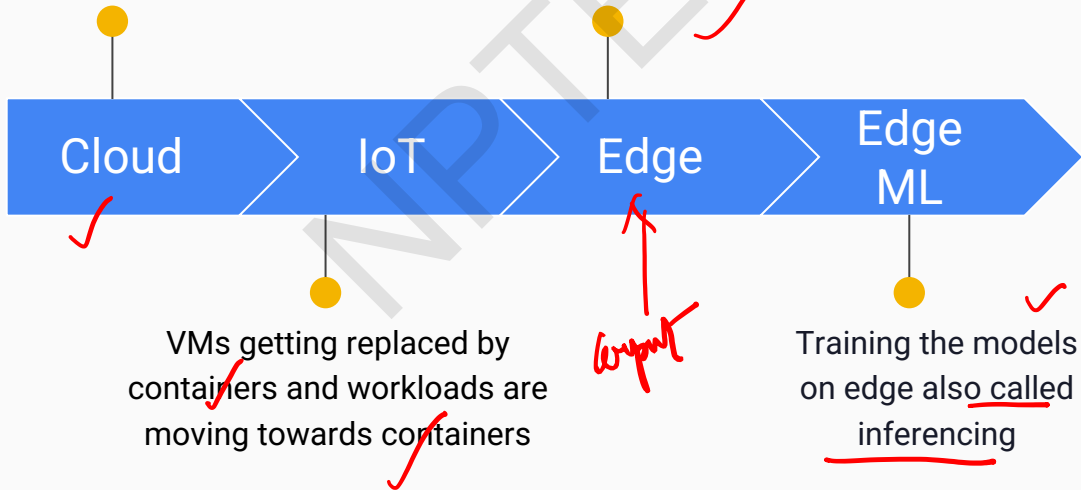
Limitations of current cloud system

- AI use cases need real-time responses from the devices they are monitoring.
- Cloud-based inference cannot provide this real-time response due to inherent issues with latency.
- If edge devices have connectivity issues or no internet connection it can not perform well.
- Sufficient bandwidth required to transfer the relevant amount of data in a proper time frame can also be an issue.


Evolution of Cloud

Virtual machines running in a remote data center or storage that was offered in a remote data center

Data processed locally and compute comes much closer to the devices or the sources of data



Summary of this lecture

- Today's cloud is highly Centralized in Client-Server Architecture
 - Compute is going beyond VMs → IOT-Containers
 - Storage is complemented by CDN → ✓
 - Network stack is programmable ✓ - SDN -
 - Multiple waves of innovation in Cloud
 - Challenges for IOT-Pass ✓ →
 - Evolution of Cloud towards Edge Computing
- 



THANK YOU!

Introduction to IoT Platform



Dr. Rajiv Misra

Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

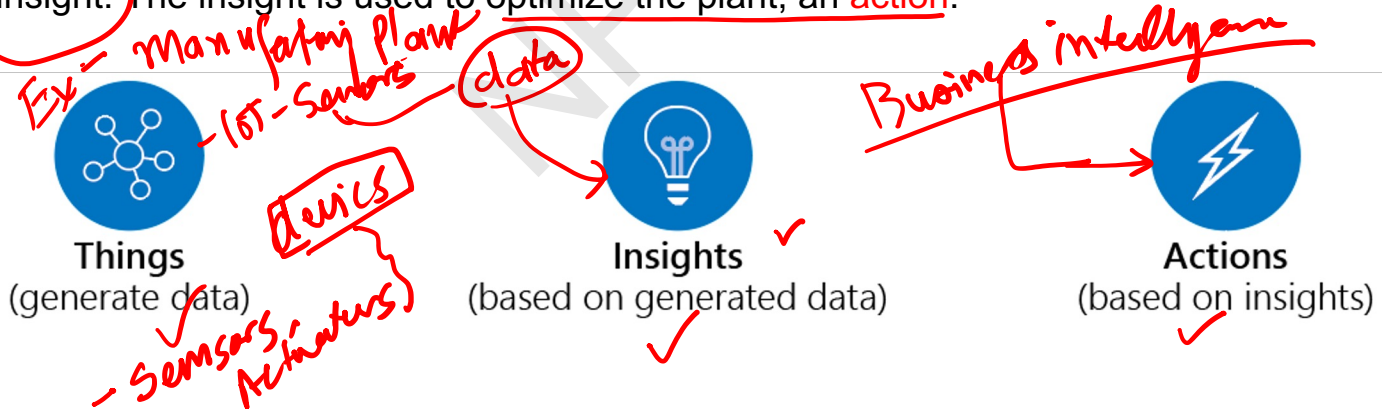
After completion of this lecture you will know the following:

- Different components of IoT platforms
- IoT platforms building blocks which are provided by different cloud providers such as microsoft, amazon, google, etc

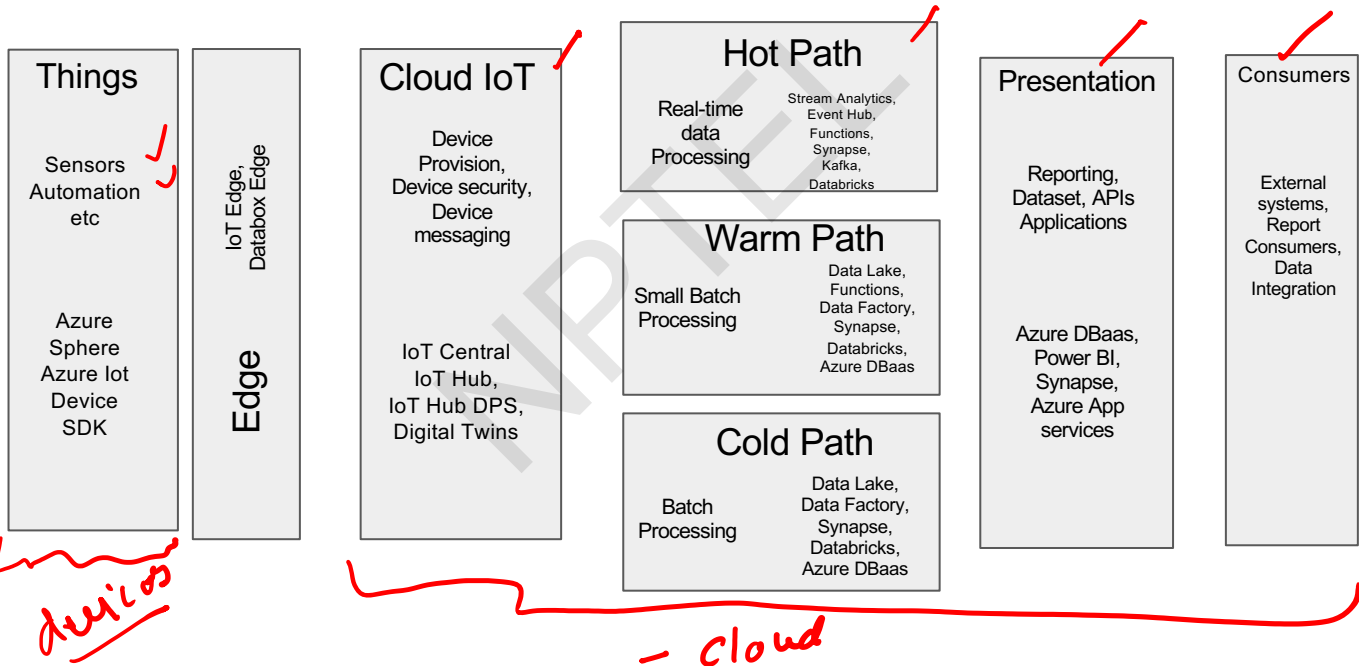
Architectural approach for IoT platform

IoT applications have three components. Things or devices send data or events that are used to generate **insights**. Insights are used to generate actions to help improve a business or process.

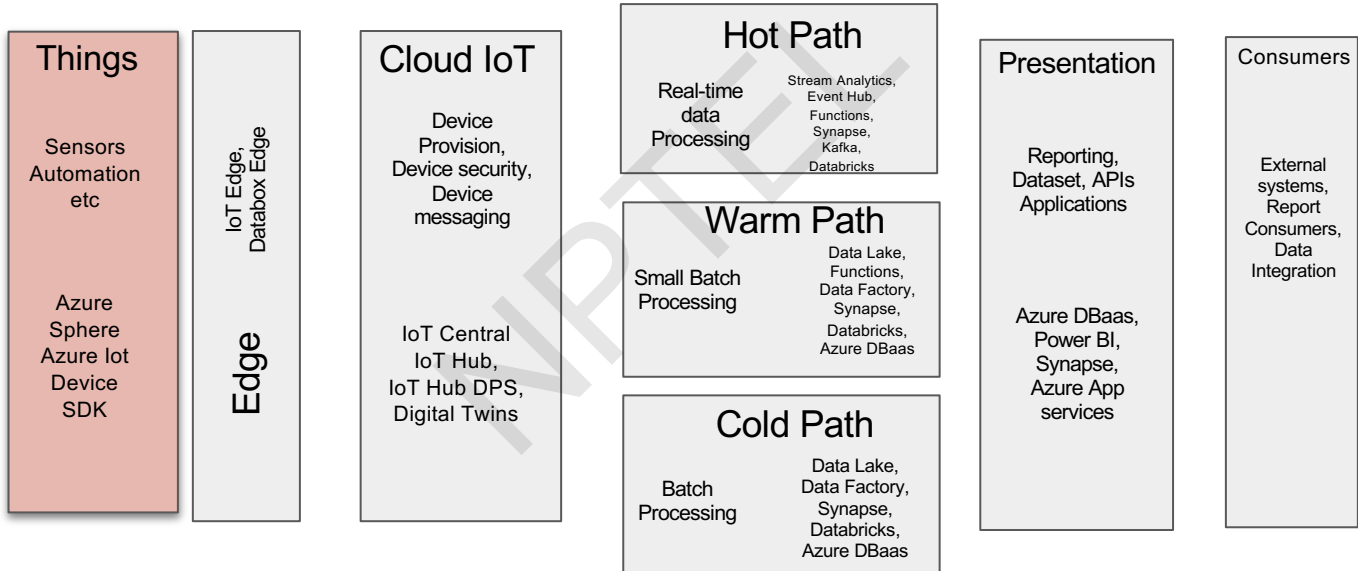
The **equipment or things** in a manufacturing plant send various types of data as they operate. An example is a milling machine sending feed rate and temperature data. This data is used to evaluate whether the machine is running or not, an insight. The insight is used to optimize the plant, an **action**.



Introduction to IoT platform



Introduction to IoT platform: Things



Introduction to IoT platform: Thing



Everything in the iot space starts with the things side of the internet of things.

When you talk to people about iot, people probably think about nest doorbells, simplisafe appliances, different kinds of things that you can use in your house that make your house smart.

All of these things are part of an IoT network so that's very familiar to most people and that is true, it is the sensors that goes into making a device work.

On azure there's a couple of things that you can use to create these things.

One is azure sphere which is like a lightweight operating system that you can put on a device and you can use this as an embedded system that will allow you to create devices and also have the connected back up to azure and also secure the device using that particular specialized operating system.

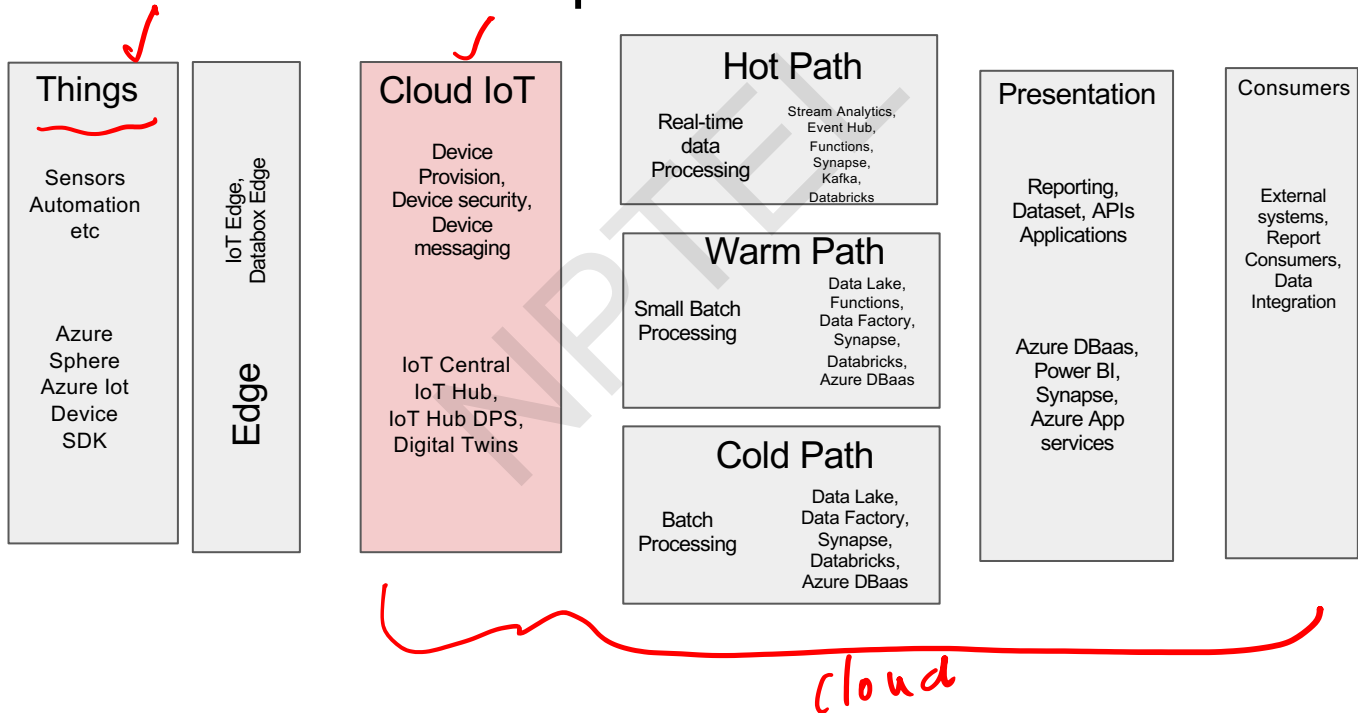
There's also the azure IoT SDK which is a specialized sdk for interacting with some of these other services. But it can be embedded on many different systems and supports a lot of different kinds of languages as well.

Things

Sensors ✓
Automation ✓
etc

Azure Sphere ✓
Azure IoT Device
SDK

Introduction to IoT platform: Cloud IoT



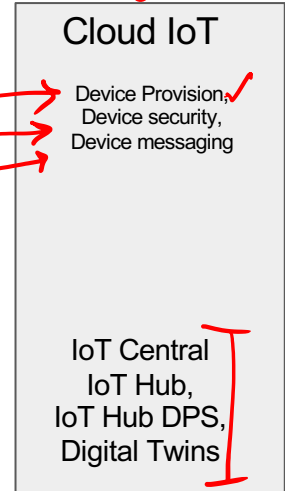
Introduction to IoT platform: Cloud IoT

Internet of things you also have the iot stack that typically exists on many different iot deployments.

Basically with the iot stack you're going to be managing devices and also brokering messages between devices and the cloud. (MQTT)

This is a management suite that allows you to scale devices and it provides a lot of services for that so you can provision devices you can take devices offline to provide security for devices it also provides a messaging infrastructure so that you can send commands to devices and also receive telemetry back from devices.

All of these endpoints and all of this management infrastructure is encapsulated in a couple of different services.



Smart-city

M2M Pub/Sub

MQTT
device
broker
Device Sensors, Actuators
cloud

MQTT- MB Telemetry
Transport
- Message Queue

Introduction to IoT platform: Cloud IoT

On azure iot central which is more a software-as-a-service offering that encapsulates a lot of the functionality for ability to create applications in the context of an iot central and that allows to have multi-tenancy with different devices to scale not only the devices but also the downstream components of things integrating with those devices are serving up.

IOT hub is a general purpose tool on azure for managing devices so it has device provisioning services that need for scaling up devices, for putting certificates on devices, generating those certificates for messaging from device for messaging to a device ie low level and more functionally oriented.

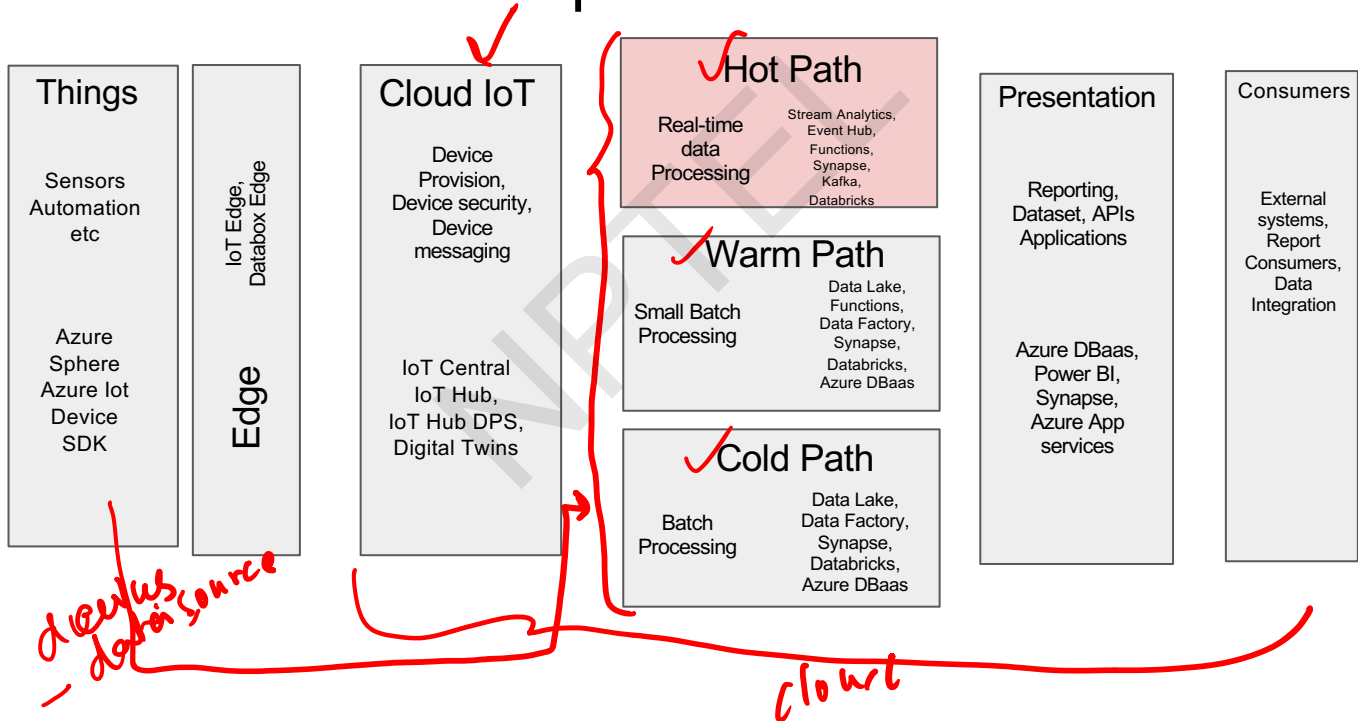
Azure digital twins, digital twinning is the ability to manage device configuration in a suite of software to integrate with azure IOT hub maintains some kind of state information about devices in the cloud.

Cloud IoT ✓

Device Provision, ✓
Device security,
Device messaging

IoT Central ✓
IoT Hub, ✓
IoT Hub DPS, ✓
Digital Twins ✓

Introduction to IoT platform: Hot Path



Introduction to IoT platform: Hot Path

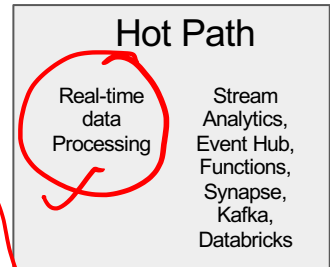
The data is routed to one of the three different paths.
ie the hot path or the cold path or the warm path

Hot path data is data that is processed in real time so as it comes off of the iot hub It gets processed within seconds of that happening so the message hits the hot path it's processed and then it's presented to something in the consumption layer.

The consumption layer is able to consume tt data immediately once it's been processed in the hot path.

You could write the output from a hotpath to a cold storage system that is consumed by something like an api. The data is written in real time but the api might be querying that data that was written an hour ago.

Processing data in real time such as a dashboard that is constantly monitoring things in their present state as comes off of the hot path and into the consumption layer.



Introduction to IoT platform: Hot Path

There are several offerings on azure for hotpath data is going to be event hubs the messaging platform

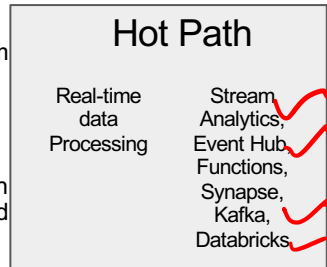
Event hubs can also write messages to a cold storage that can be consumed by cold pass or warm path but whatever you get out of event hubs can be wired up to all these other other kinds of processors such as stream analytics which is a platform as a service offering that uses sql to transform data aggregate data enrich it

Then you have functions which can be triggered by event hubs . Then there's azure synapse which as synapse allows you to have a full suite of tools at your disposal that do all kinds of things related to data processing that is streams.

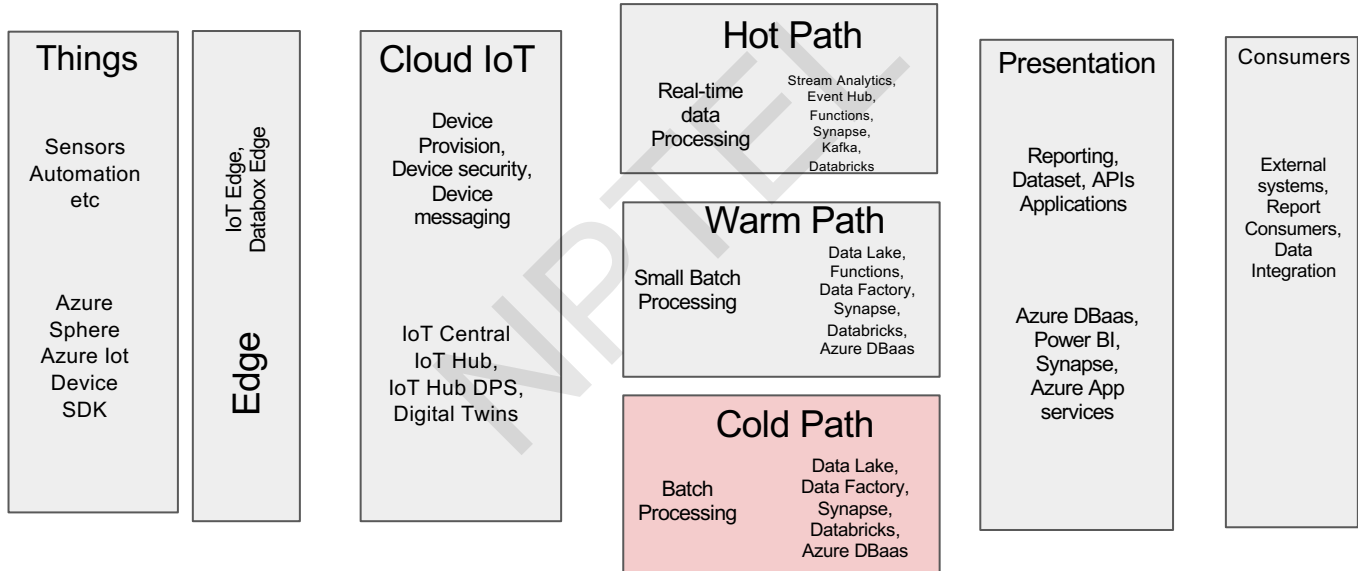
You can also use kafka which is out of the apache space which is similar to stream analytics in that you do real-time data processing but it's more specific in its implementation but it wires up directly to event hubs.

Databricks is typically used for more batch style oriented workloads but you can use it for streaming ✓

Combining any number of these can do a lot of different kinds of hotpath aggregations transformations queries filters whatever it might be they're all different tools that all do it very similar functionality within the azure context.



Introduction to IoT platform: Cold Path



Introduction to IoT platform: Cold Path ✓

Coldpath is more batch-oriented, hotpath will process the message as it hits the system while coldpath really processes the messages as they accumulate on the system and rather being triggered by the message itself what it allows for is data to be accumulated over a period of time and then typically on a trigger that is timer based it will then take whatever data has been accumulated and process that data in batch.

Then it will write the data back to some kind of cold storage whatever the processing on that data might look like ✓

This typically works as opposed to hot path where you have something like event hubs that deliver a message to a processor what you typically do in this case is you land the message that as it comes off of iot hub into some kind of what we call cold storage so that's typically some kind of database or some kind of data retention system.

Now that could be something like a data lake which would be basically built on top of blob storage, you can also do it with blob storage as well but fundamentally data lake is built on top of blob storage in any case.

Cold Path

Batch
Processing

Data Lake, ✓
Data Factory,
Synapse,
Databricks,
Azure DBaaS

Introduction to IoT platform: Cold Path

Azure database as a service offering, use sql databases, use cosmos databases, use postgres or mysql putting into some kind of data storage platform.

Then from there once it's accumulated in that cold storage then the trigger fires and it's going to launch whatever processing capability is going to be a part of that and that's where something like data data factory or azure synapse or or databricks

Data factory is software as a service or platform as a service gives the ability to visually build workflows inside of data factory that can then take data out of a data lake or database and process it in batches and then write the results back to some kind of output.

Now synapse has similar functionality but it is integrated with the synapse suite on azure

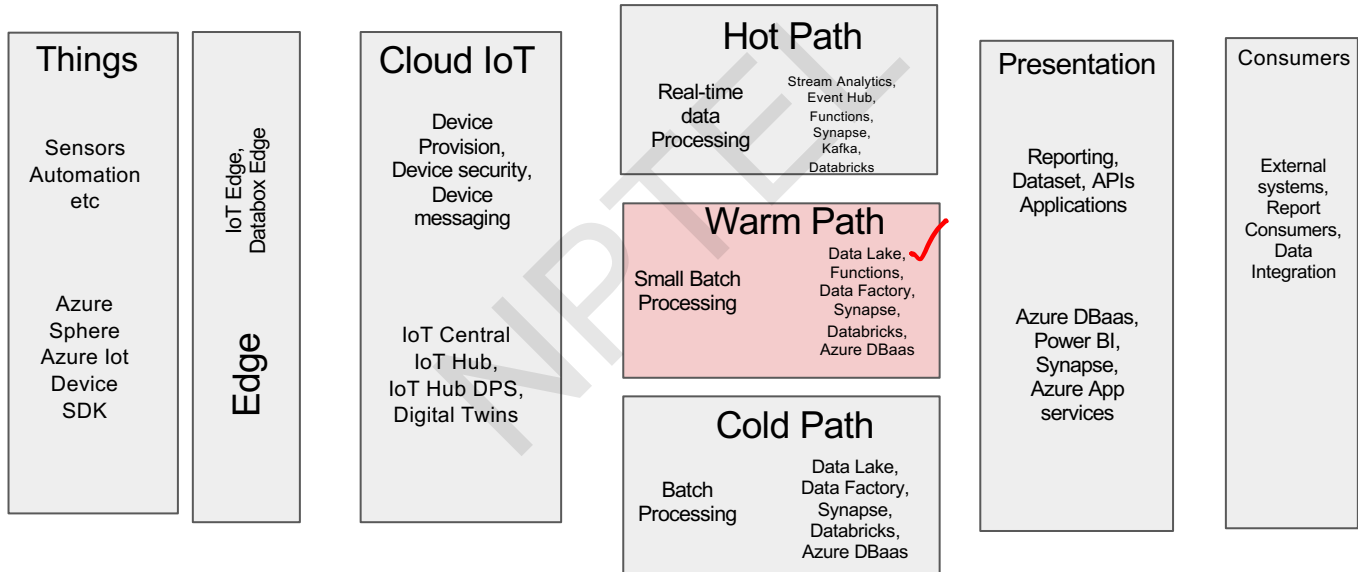
so databricks has the ability to scale and it also integrates with a lot of other different offerings on azure including the databases data lakes and many of these other similar things is more of a visual designer for building those kind of workflows.

Cold Path

Batch
Processing

Data Lake,
Data Factory,
Synapse,
Databricks
Azure DBaaS

Introduction to IoT platform: Warm Path



Introduction to IoT platform: Warm Path ✓

Between the hot path and the cold path is warm path. It has some kind of functionality that might seem similar to hot path and something that might seem to similar to cold path.

Tools that are more in line with warmpath such as data lake, factory data, factory synapse, databricks use azure functions.

Even use something like stream analytics or kafka for some smaller workloads ✓

The distinction between hot path, warm path and cold path really isn't clear.

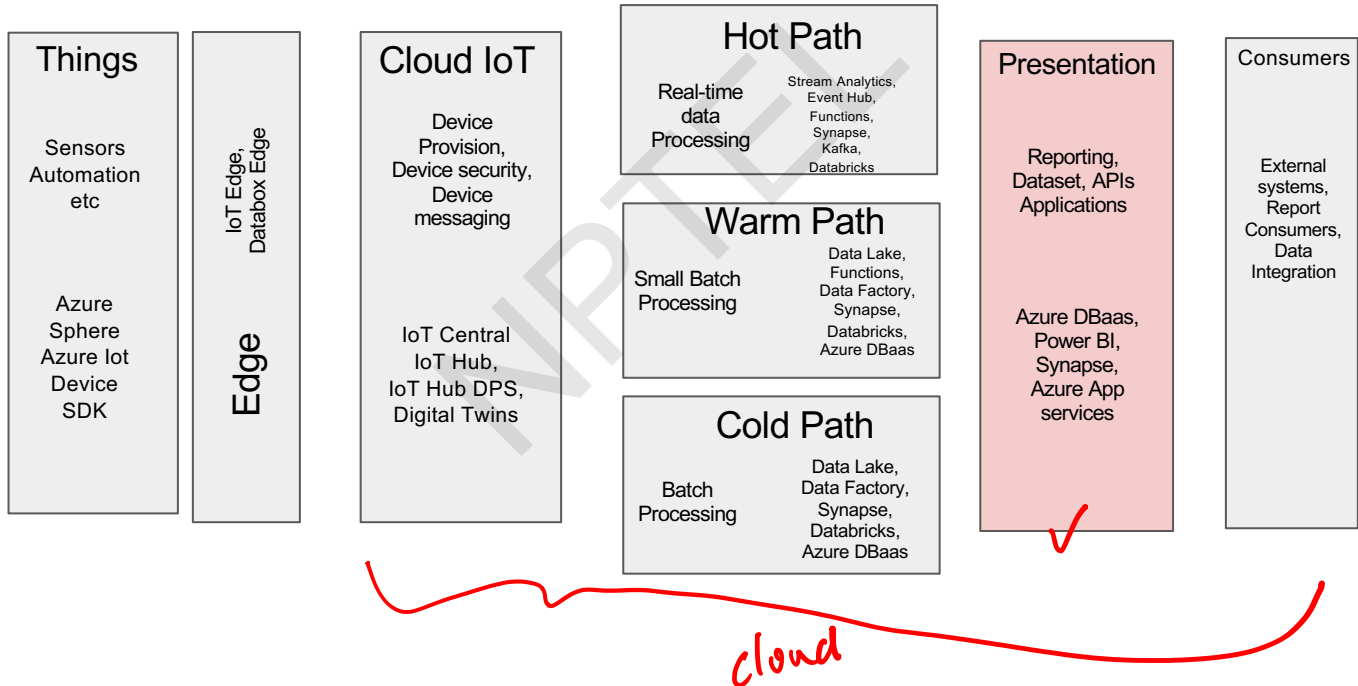
The takeaway from this is that hot path is real-time warm path is going to be more often smaller workloads that are going to be rating on smaller time scales like 5 minutes, 10 minutes, 15 minutes or an hour and cold path is going to be larger workloads that are going to be operating over long periods of time. It might be five minutes if there's a lot of data it could be an hour it could be a day could be a week.

Warm Path

Small Batch
Processing

Data Lake,
Functions,
Data Factory,
Synapse,
Databricks,
Azure DBaaS

Introduction to IoT platform: Presentation



Introduction to IoT platform: Presentation ✓

The data that was collected by way of things that originated in the iot layer i's going to be an aggregated plus some enhancement of that data and some filtering of that data. This is going to be things like api's that are going to be consumed by applications, it's going to be reports that people are going to be looking at.

That could be some kind of dashboard like report where you're looking at telemetry in real time or a query telemetry out of a data set or it could just be the raw data itself that you're going to be providing by way of some kind of data integration where you're taking some kind of export of the data and then taking that into another data system for consumption in that system. ✓

Regardless of whatever is the presentation of that data it's basically the output of the data pipelines that you're employing either as hotpath warm path or cold path and the presentation then can take that data and then just make it available.

Presentation

Reporting, ✓
Dataset, APIs
Applications

Azure DBaaS,
Power BI,
Synapse,
Azure App
services

Introduction to IoT platform: Presentation

So this is going to imply things like security, access controls and those kinds of things, also a database as a service offering, so anything that would store the data, that would be sql server, cosmos db, maria, azure data explorer there's a lot of different ways to present data.

Then you have the reporting services such as power guide, which is kind of the one tool that a lot of folks love to use for building dashboards in the microsoft context and it can hook up to all kinds of data sources and then it can import those and then use data sets that are manipulated inside of the rbi context itself.

You can use azure functions and azure app services for serving up api's, so azure functions gives you the ability to create http endpoints that can then query back into whatever database a source that you want to use or other data sources.

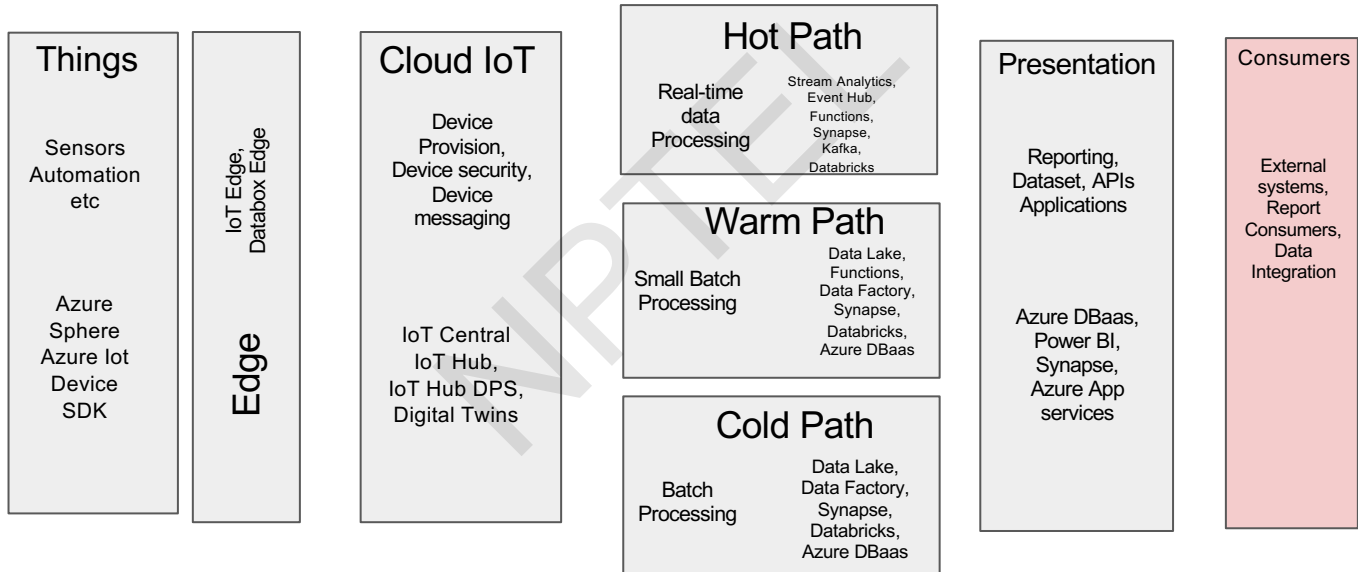
Then azure app services if you want to just write something like an mpd application that's going to be exposing some kind of data api that external applications then can consume from that data source.

Presentation

Reporting,
Dataset, APIs
Applications

Azure DBaaS,
Power BI,
Synapse,
Azure App
services

Introduction to IoT platform: Consumers



Introduction to IoT platform: Consumers

Now we have consumers, this is not so much an explicit part of the system as it is a more implicit part of the system.

Ultimately what ends up in the presentation layer is going to be determined by what the external consumers of this data are going to want to be in that presentation layer.

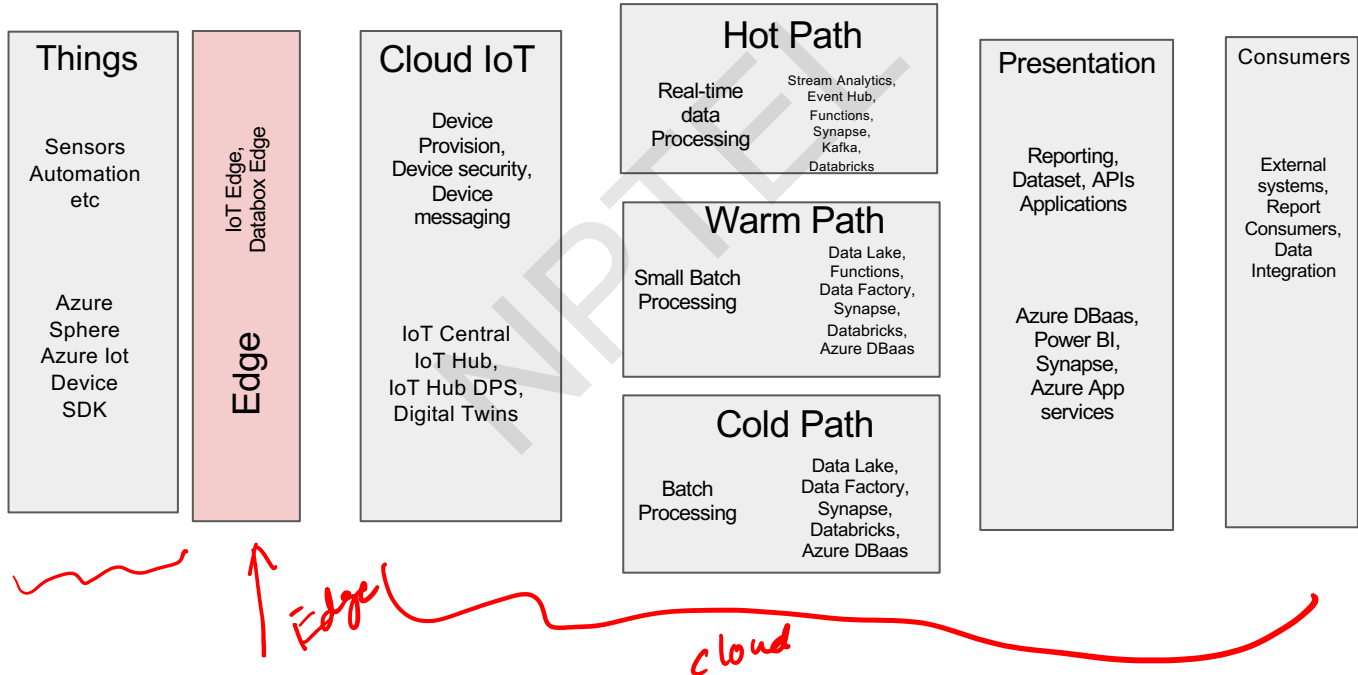
So whenever you're designing a system that is going to be presenting data, you start with the api in mind and you kind of work back from that to the source data and that's really why we have set it up this way.

The reason we included it is because you need to be cognizantly aware of how you want this data to show up in whatever is going to be integrated with it whether it be a report, whether it be an api or some kind of external data integration.

Consumers

External systems,
Report Consumers,
Data Integration

Introduction to IoT platform: Edge



Introduction to IoT platform: Edge

On the edge of a network, a local area network a bunch of devices that are emitting telemetry and events and doing all those kinds of things that they do and those are ultimately sent back to the cloud.

However, in some cases you might want to put some kind of preprocessor in place that will do some filtering and aggregation and some other enhancements on the data closer to where the devices are.

So in a sense the edge is almost a microcosm of everything that happens in the cloud.

You will have things like message buses, data pipelines and other kinds of data enhancement tools that exist in that context for the purpose of pre-processing that data before it goes over to the cloud side.

There are two services that are in this space on the edge, the first one is the IoT edge.

IoT edge is a platform that is more of an operating system that you can install on an appliance and it's based around Docker containers. You can do things like stream analytics in that context, it also gives you the ability to do message filtering and a number of other things that are a part of that ecosystem. Also the code that you want to make and install it by way of a Docker container on the IoT edge.

IoT Edge,
Databox Edge

Edge

Introduction to IoT platform: Edge

It also offers a message proxy for sending messages from devices to the cloud so that you can basically queue those messages up on the IoT edge.

In the event of an internet outage, you can then queue those messages up there and then when the internet is restored, it will then forward those onto the cloud so it mitigates against things like losses of message.

There is local response to events in that particular context as well, so you can build an ML and other kind of event management into the IoT edge. It can quickly respond to something like a fire, for instance if a device reports that there's a fire, you can have a command issued by the IoT edge to put that fire out for instance.

Databox is a similar service but it's not as purpose-built as IoT edge and it's basically bringing a lot more the ML type workloads that you get in something like ML workspaces.

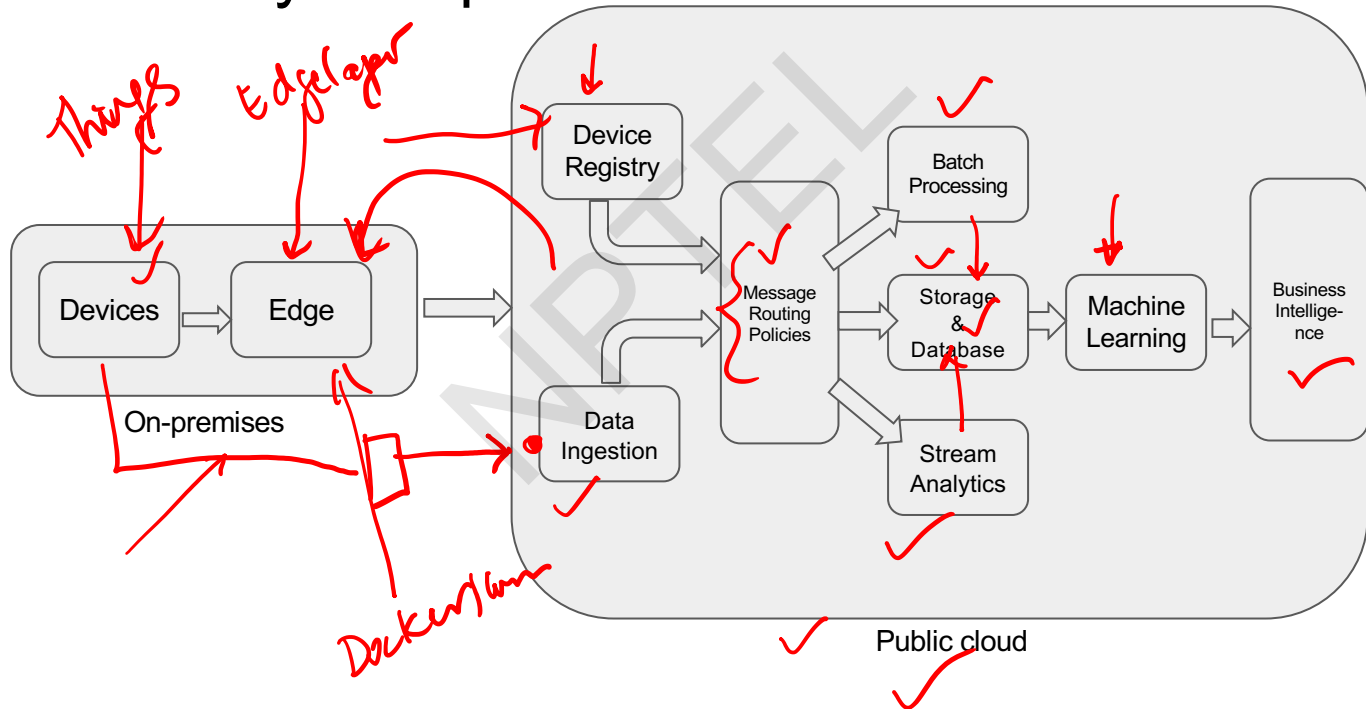
These kind of things are bringing to the edge as well, so it can do data ingestion and apply ML models.

In the context of an edge installation rather than having to ship all that data back up to the cloud you can do it more intelligently on the edge and do it more quickly, so that you don't have to rely on an internet connection and the latency that the cloud introduces.

IoT Edge,
Databox Edge

Edge

Summary: IoT platform



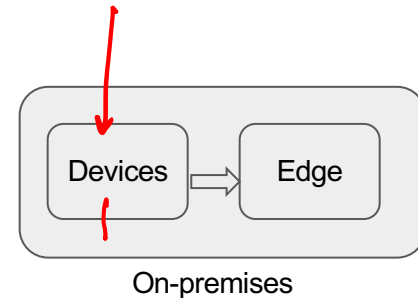
IoT platform: Things

On one end of the platform we have devices and these devices are either sensors or actuators.

Sensors generate data, for example a temperature sensor, a humidity sensor, pressure sensor and so on. The generated data is going to be acquired and ingested into the cloud.

Then there are actuators like switches and bulbs. These are the things that you could switch on and switch off that have electromechanical interface.

The devices are further connected to the edge and the edge acts as a gateway abstracting the devices that are at the lowest level of the spectrum and that actually connects to the public cloud.



IoT platform: Insight

Now on the cloud side we have two touch points for the edge or the devices.

One is the device registry that is primarily used for onboarding the devices and it is the repository of devices.

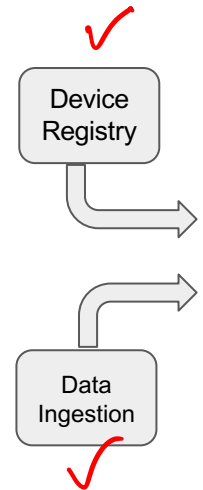
Every device that is connected to the IOT platform has an identity within the device registry.

The authentication authorization and the metadata of the devices is stored in the device registry.

Consider an enterprise corporate directory scenario where the device registry like an LDAP of devices, you can query to get a lot of metadata and useful information about every device connected to the platform.

The public cloud pass also called IOT pass exposes a data ingestion endpoint. This is the high velocity, high throughput endpoint where the sensor data gets streamed.

Typically could be Kafka if you are doing it yourself or it could be as your event hubs or Amazon kineses or Google cloud pub/sub so it is the pipe that basically acquires the data and passes on to the data processing pipeline




IoT platform: Insight ✓

Now both the device registry and data ingestion endpoints are connected to a message routing policy. ✓

A message routing policy which will define how this data is going to be split between real-time processing and batch processing and how the raw data is stored and how the processed data is going to be stored. ✓

This is the place where you actually create a rules engine or you basically create some kind of policy that is going to define how the data flows.

For example, some data needs to be batch process, where you first collect and then process, in some cases you need to perform real-time stream analytics.



Message
Routing
Policies ✓

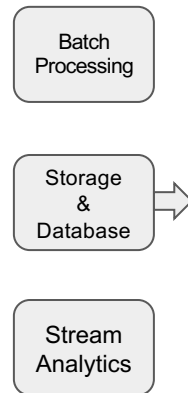
IoT platform: Insight

The batch processing layer which is also called as cold path analytics and stream analytics layer which is also called as **hot path analytics**. ✓

In other words, when you are performing queries on data as it comes that is called the hot path analytics and if you are storing and processing the data over a period of time it is called the **cold path analytics**. ✓

Now both the raw data which is going or about to go to either batch processing or stream analytics is first persisted in a time series database or an unstructured database and even the output from the batch processing and stream analytics gets persisted in the the same database.

Then we have storage and databases for persisting the raw sensor data and also the process data



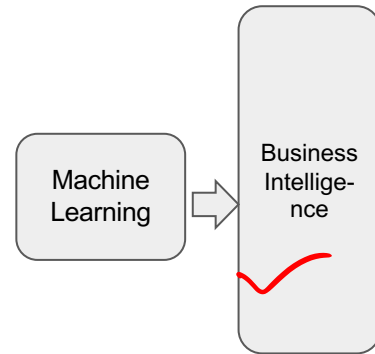
IoT platform: actions ✓

Now from the same data store we apply machine learning algorithms to basically find out anomaly detection and predictive analytics from the data that is coming in.

Finally, all of that is fed into an enterprise Business Intelligence Platform, where you can actually run dashboards and alerts and the entire visualization happens on the data warehousing or the business intelligence layer

These key building blocks of IOT platform you could actually map this to Azure or AWS or Google or G predicts, Bosch IOT, etc.

Every platform has a very similar architecture it's almost like a blueprint for any public cloud-based IOT platform.



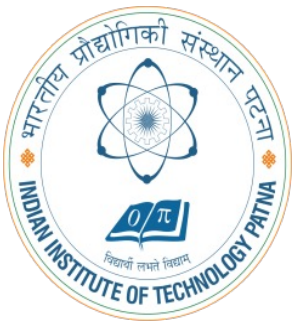
Lecture Summary

- Detail of components of IoT architecture.
- Concepts of IoT platform building blocks.



THANK YOU!

Time and Clock Synchronization in IoT



Dr. Rajiv Misra

Professor

Dept. of Computer Science & Engg.

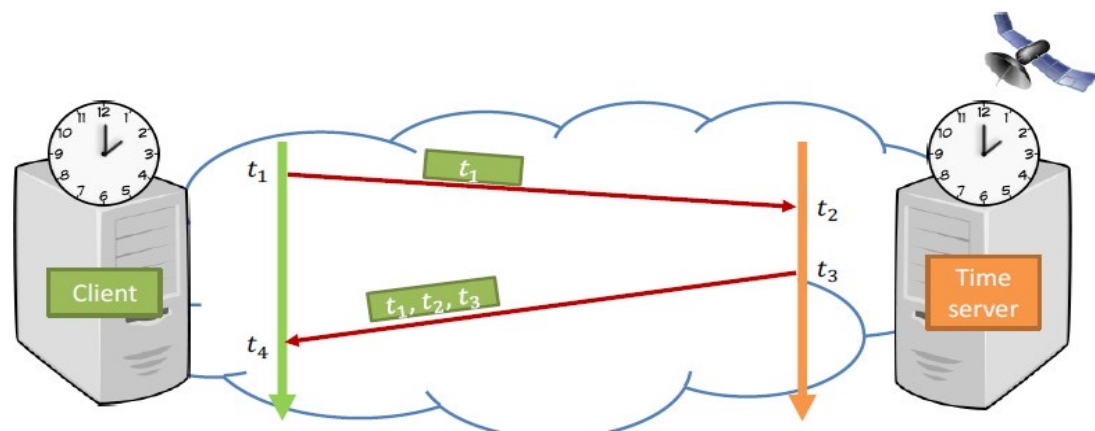
Indian Institute of Technology Patna

rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the fundamentals of clock synchronization in IoT and its different algorithms.
- To understand how clocks operate on IoT devices and how they can be synchronized in an accurate and efficient fashion.



Preface

Content of this Lecture:

- Internet of Things (IoT) devices that are wirelessly connected in mesh networks often need mutual clock time synchronization, to enable chronological ordering of sensor events, coordination of asynchronous processes across devices, or network-wide coordination of actuators. ✓
- We will also discuss the causality and a general framework of logical clocks and present two systems of logical time, namely, lamport and vector, timestamps to capture causality between distributed events of an Internet of things as a distributed system.

Need of Synchronization

Synchronizing clocks on Internet of Things (IoT) devices is important for applications such as monitoring and real time control.

✓
You want to catch a bus at 9.05 am, but your watch is off by 15 minutes

What if your watch is Late by 15 minutes?

- You'll miss the bus! ✓

What if your watch is Fast by 15 minutes?

- You'll end up unfairly waiting for a longer time than you intended ✓

Time and Synchronization

- **Time and Synchronization**

(“There’s is never enough time...”)

- **Distributed Time**

- The notion of time is well defined (and measurable) at each single location
- But the relationship between time at different locations is unclear

- **Time Synchronization is required for:**

- **Correctness** ✓
- **Fairness** ✓

Synchronization in an IoT

Example: Cloud based airline reservation system:

- Server X receives, a client request, to purchase the last ticket on a flight, say PQR 123.
- Server X timestamps the purchase using its local clock as **6h:25m:42.55s**. It then logs it. Replies ok to the client.
- That was the very last seat, Server X sends a message to Server Y saying the “flight is full”.
- Y enters, “Flight PQR 123 is full” + its own local clock value, (which happens to read **6h:20m:20.21s**).
- Server Z, queries X's and Y's logs. Is confused that a client purchased a ticket at X after the flight became full at Y.
- **This may lead to full incorrect actions at Z**

Key Challenges

- **End-hosts in Internet based systems (like clouds)**
 - Each have its own clock
 - Unlike processors (CPUs) within one server or workstation which share a system clock.
- **Processes in internet based systems follow an asynchronous model.**
 - No bounds on
 - Messages delays ✓
 - Processing delays ✓
 - Unlike multi-processor (or parallel) systems which follow a **synchronous** system model

Definitions

- An asynchronous distributed system consists of a number of **processes**.
- Each process has a state (**values of variables**).
- Each process takes **actions** to change its state, which may be an instruction or a communication action (**send, receive**).
- An **event** is the occurrence of an action.
- Each process has a large clock – events within a process can be assigned **timestamps**, and thus ordered linearly.
- But- in a IoT system, we also need to know the time order of events **across** different processes.

Space-time diagram

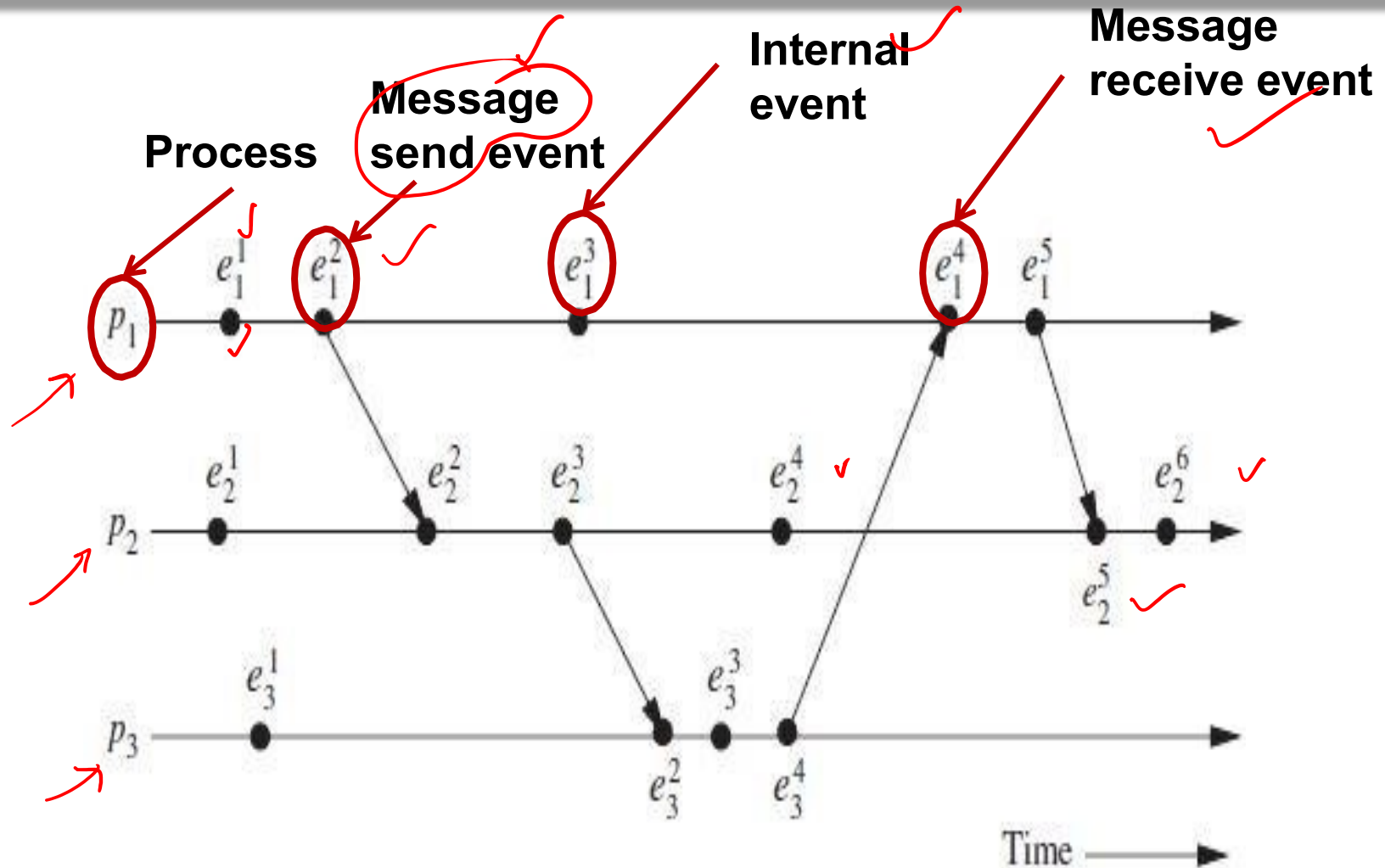


Figure : The space-time diagram of a distributed execution.

Clock Skew vs. Clock Drift

- Each process (running at some end host) has its own clock. ✓
- When comparing two clocks at two processes. ✓
 - **Clock Skew = Relative difference in clock values of two processes.**
 - ✓ • Like distance between two vehicles on road.
 - **Clock Drift = Relative difference in clock frequencies (rates) of two processes**
 - ✓ • Like difference in speeds of two vehicles on the road.
- **A non-zero clock skew implies clocks are not synchronized** ✓
- **A non-zero clock drift causes skew increases (eventually).** ✓
 - If faster vehicle is ahead, it will drift away.
 - If faster vehicle is behind, it will catch up and then drift away.

Clock Inaccuracies

- Clocks that must not only be synchronized with each other but also have to adhere to physical time are termed **physical clocks**. ✓
- Physical clocks are synchronized to an accurate real-time standard like **UTC (Universal Coordinated Time)**. ✓
- However, due to the clock inaccuracy, a **timer (clock)** is said to be working within its specification if (where constant ρ is the **maximum skew rate** specified by the manufacturer)

$$1 - \rho \leq \frac{dc}{dt} \leq 1 + \rho$$

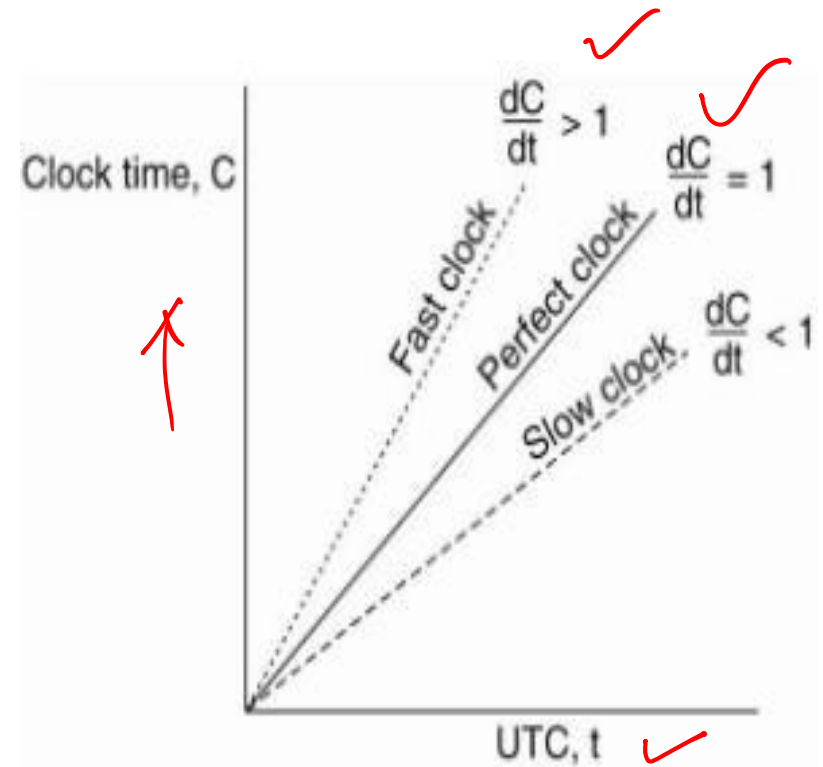


Figure: The behavior of fast, slow, and perfect clocks with respect to UTC.

How often to Synchronize

- **Maximum Drift rate (MDR) of a clock** ✓
- Absolute MDR is defined to relative coordinated universal Time (UTC). UTC is the correct time at any point of time.
 - MDR of any process depends on the environment.
- Maximum drift rate between two clocks with similar MDR is **$2 * \text{MDR}$** .
- Given a maximum acceptable skew M between any pair of clocks, need to synchronize at least once every: ✓
 $M / (2 * \text{MDR})$ time units. ✓
 - Since time = Distance / Speed. ✓

External vs Internal Synchronization

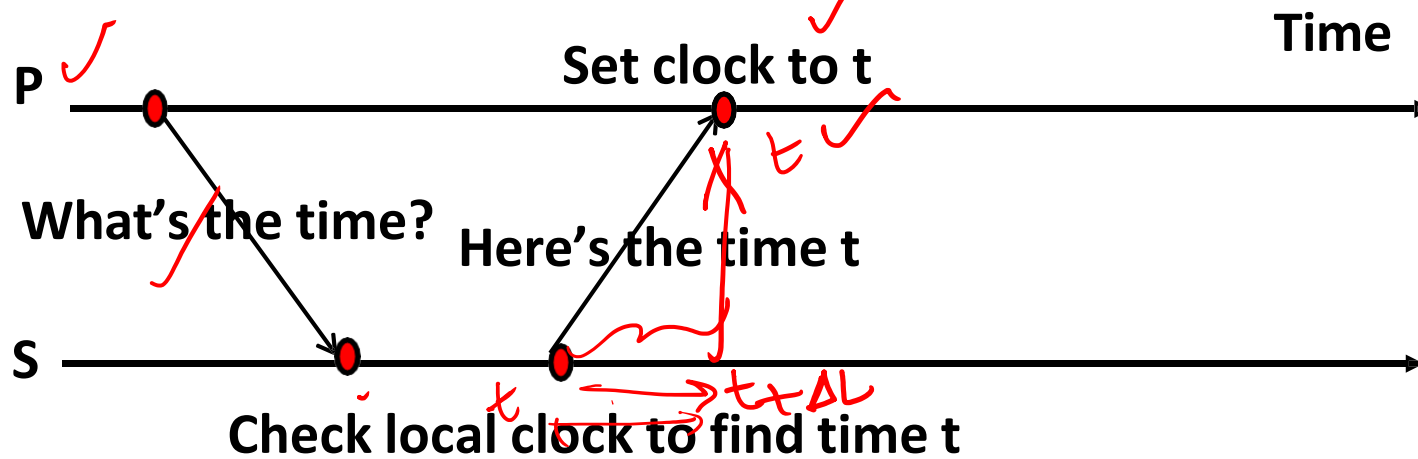
- Consider a group of processes ✓
- **External synchronization** ✓
 - Each process $C(i)$'s clock is within a bounded D of a well-known clock S external to the group ✓
 - $|C(i) - S| < D$ at all times. ✓
 - External clock may be connected to UTC (Universal Coordinated Time) or an atomic clock. ✓
 - **Example: Christian's algorithm, NTP**
- **Internal Synchronization**
 - Every pair of processes in group have clocks within bound D ✓
 - $|C(i) - C(j)| < D$ at all times and for all processes i, j . ✓
 - **Example: Berkley Algorithm, DTP** ✓

External vs Internal Synchronization

- External synchronization with $D \Rightarrow$ Internal synchronization with $2 * D$.
- Internal synchronization does not imply External Synchronization.
 - In fact, the entire system may drift away from the external clock S !

Basic Fundamentals

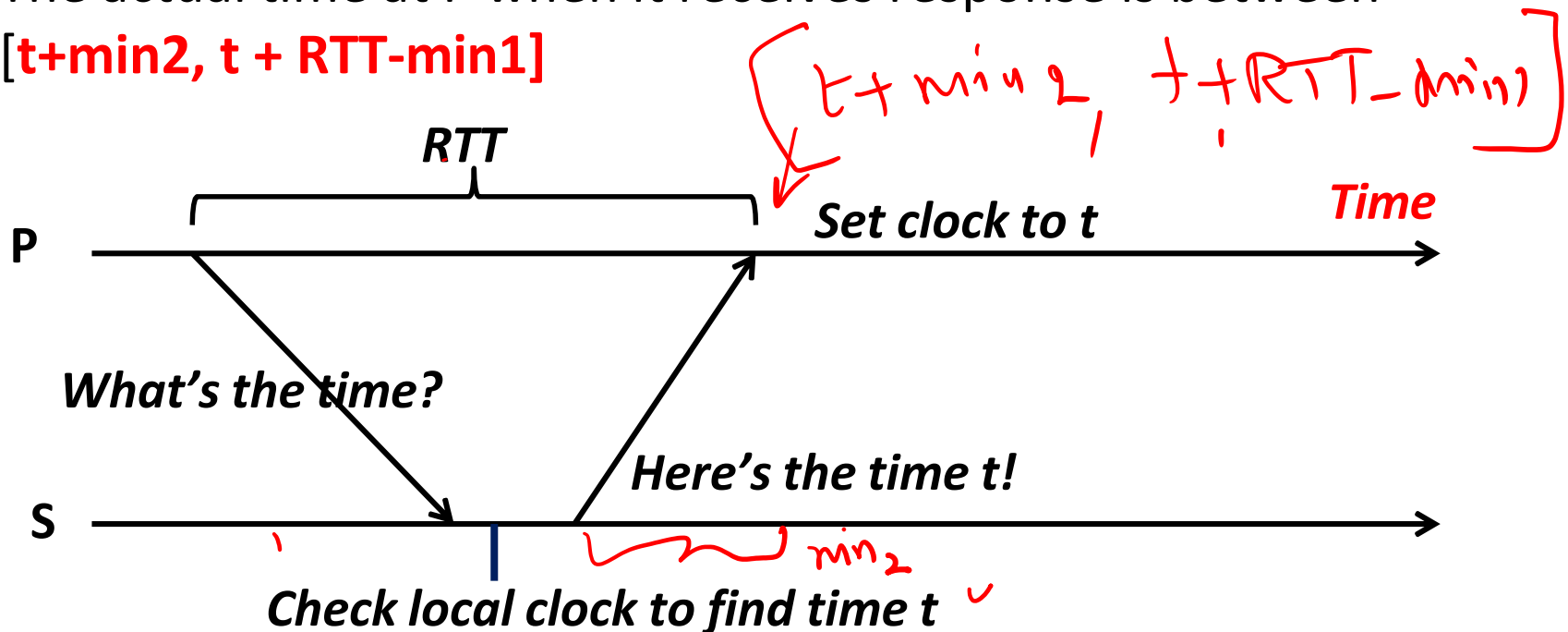
- External time synchronization
- All processes P synchronize with a time server S.



- **What's Wrong:**
 - By the time the message has received at P, time has moved on.
 - P's time set to t is inaccurate.
 - Inaccuracy a function of message latencies.
 - Since latencies unbounded in an asynchronous system, the inaccuracy cannot be bounded.

(i) Christians Algorithm

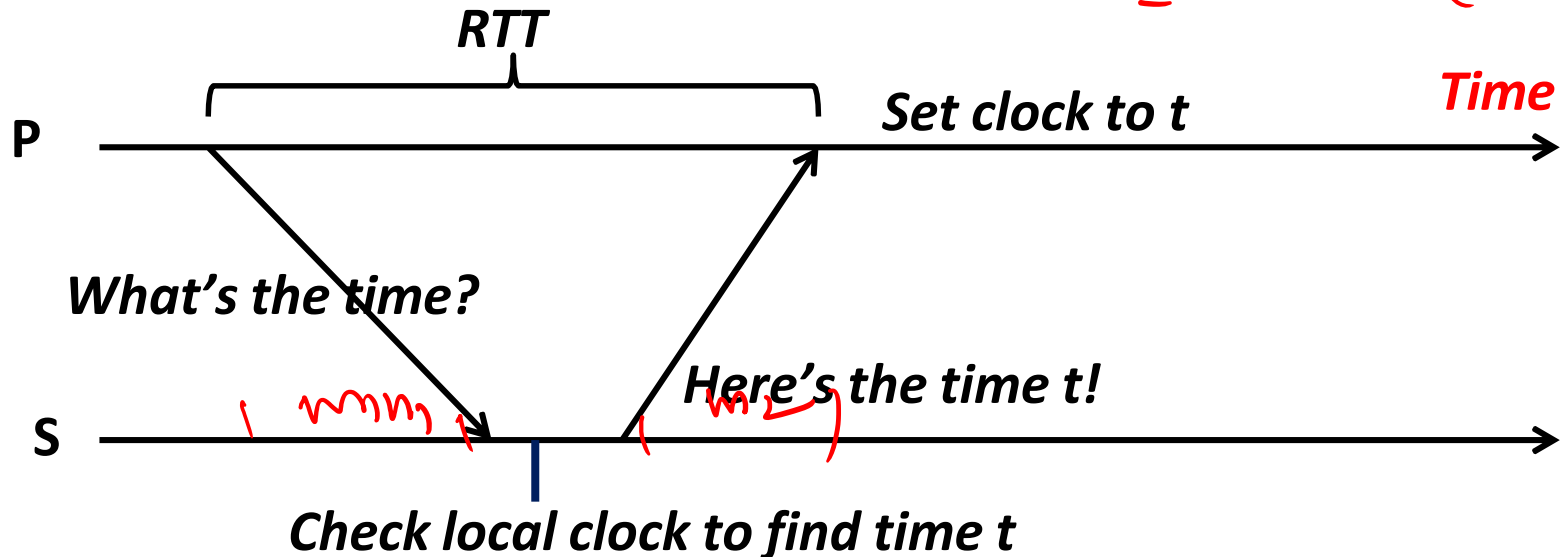
- P measures the round-trip-time RTT of message exchange
- Suppose we know the minimum $P \rightarrow S$ latency \min_1
- And the minimum $S \rightarrow P$ latency \min_2
 - \min_1 and \min_2 depends on the OS overhead to buffer messages, TCP time to queue messages, etc.
- The actual time at P when it receives response is between $[t + \min_2, t + \text{RTT} - \min_1]$



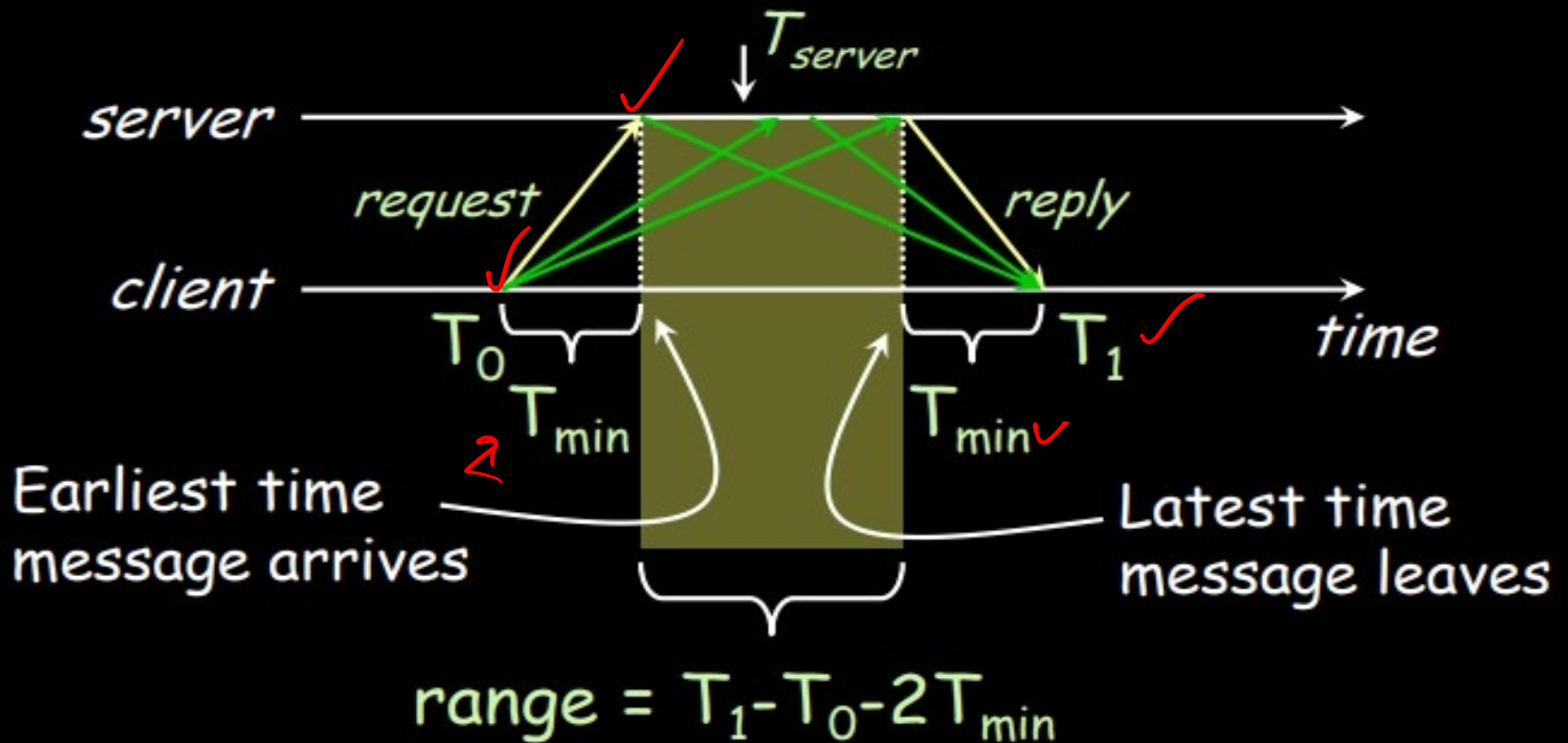
Christians Algorithm

- The actual time at P when it receives response is between $[t + \text{min2}, t + \text{RTT} - \text{min1}]$
- **P sets its time to halfway through this interval**
 - To: $t + (\text{RTT} + \text{min2} - \text{min1}) / 2$
- Error is at most $(\text{RTT} - \text{min2} - \text{min1}) / 2$
 - Bounded ✓

$$\text{Error} = \text{RTT} - (\text{min}_1 + \text{min}_2)$$



Error Bounds



$$\text{accuracy of result} = \pm \frac{T_1 - T_0}{2} - T_{min}$$

Error Bounds

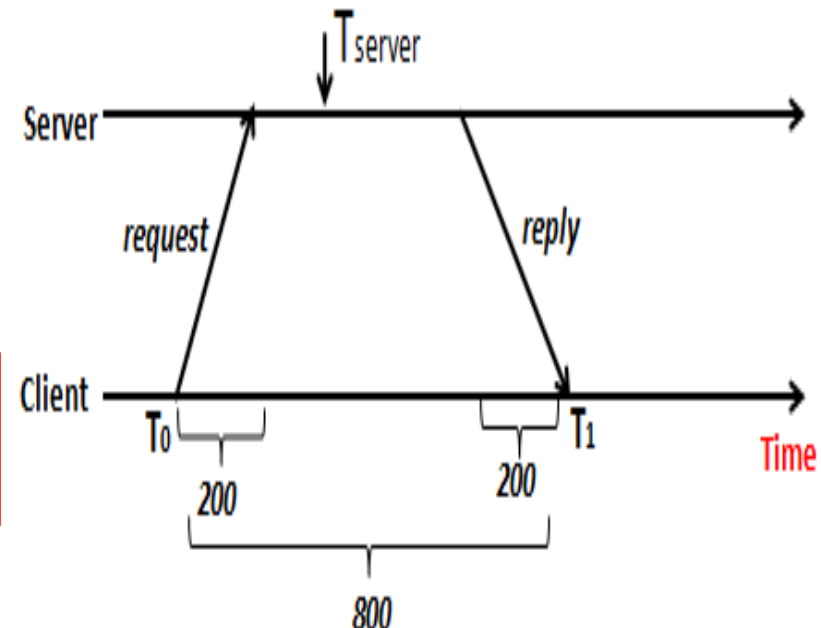
- **Allowed to increase clock value but should never decrease clock value**
 - May violate ordering of events within the same process.
- **Allowed to increase or decrease speed of clock**
- **If error is too high, take multiple readings and average them**

Christians Algorithm: Example

- Send request at 5:08:15.100 (T_0)
- Receive response at 5:08:15.900 (T_1)
 - Response contains 5:09:25.300 (T_{server})
- Elapsed time is $T_1 - T_0$
 - 5:08:15.900 - 5:08:15.100 = 800 msec
- Best guess: timestamp was generated
 - 400 msec ago
- Set time to $T_{\text{server}} +$ elapsed time
 - 5:09:25.300 + 400 = 5:09.25.700

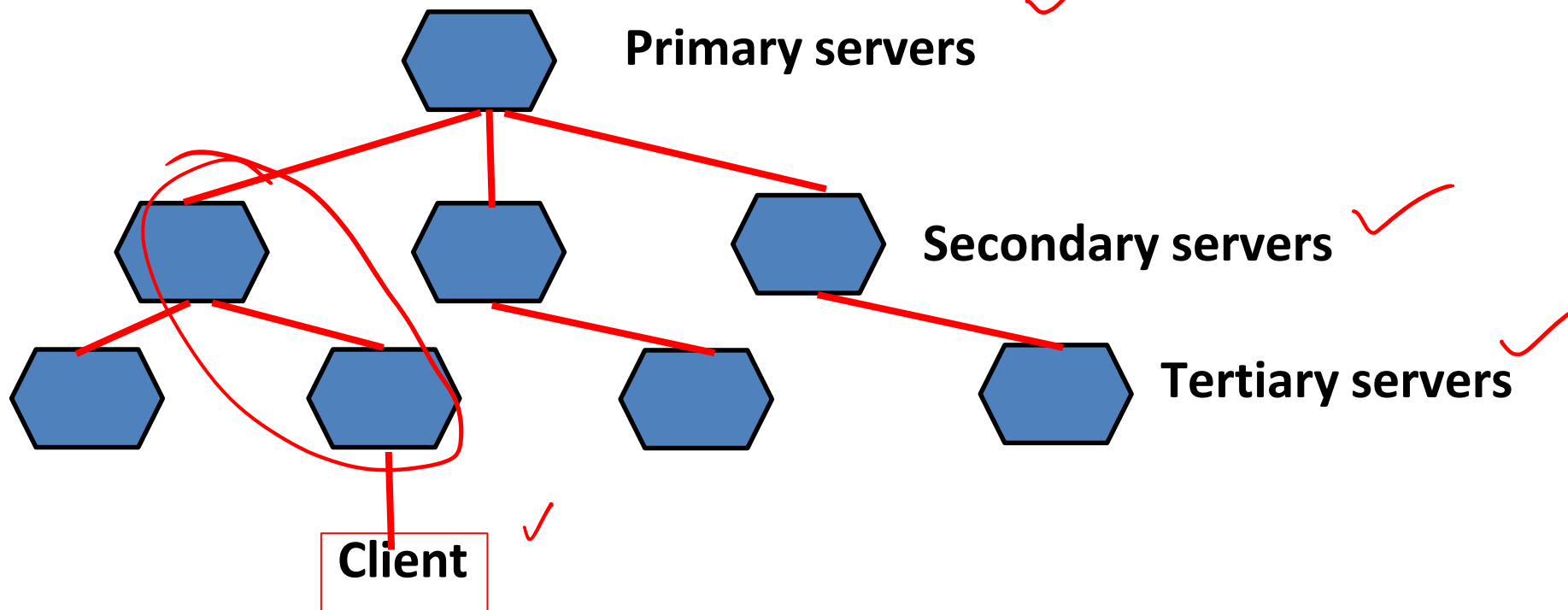
$$\text{Error} = \pm \frac{900 - 100}{2} - 200 = \pm \frac{800}{2} - 200 = \pm 200$$

- If best-case message time=200 msec
 $T_0 = 5:08:15.100$
 $T_1 = 5:08:15.900$
 $T_{\text{server}} = 5:09:25:300$
 $T_{\text{min}} = 200\text{msec}$

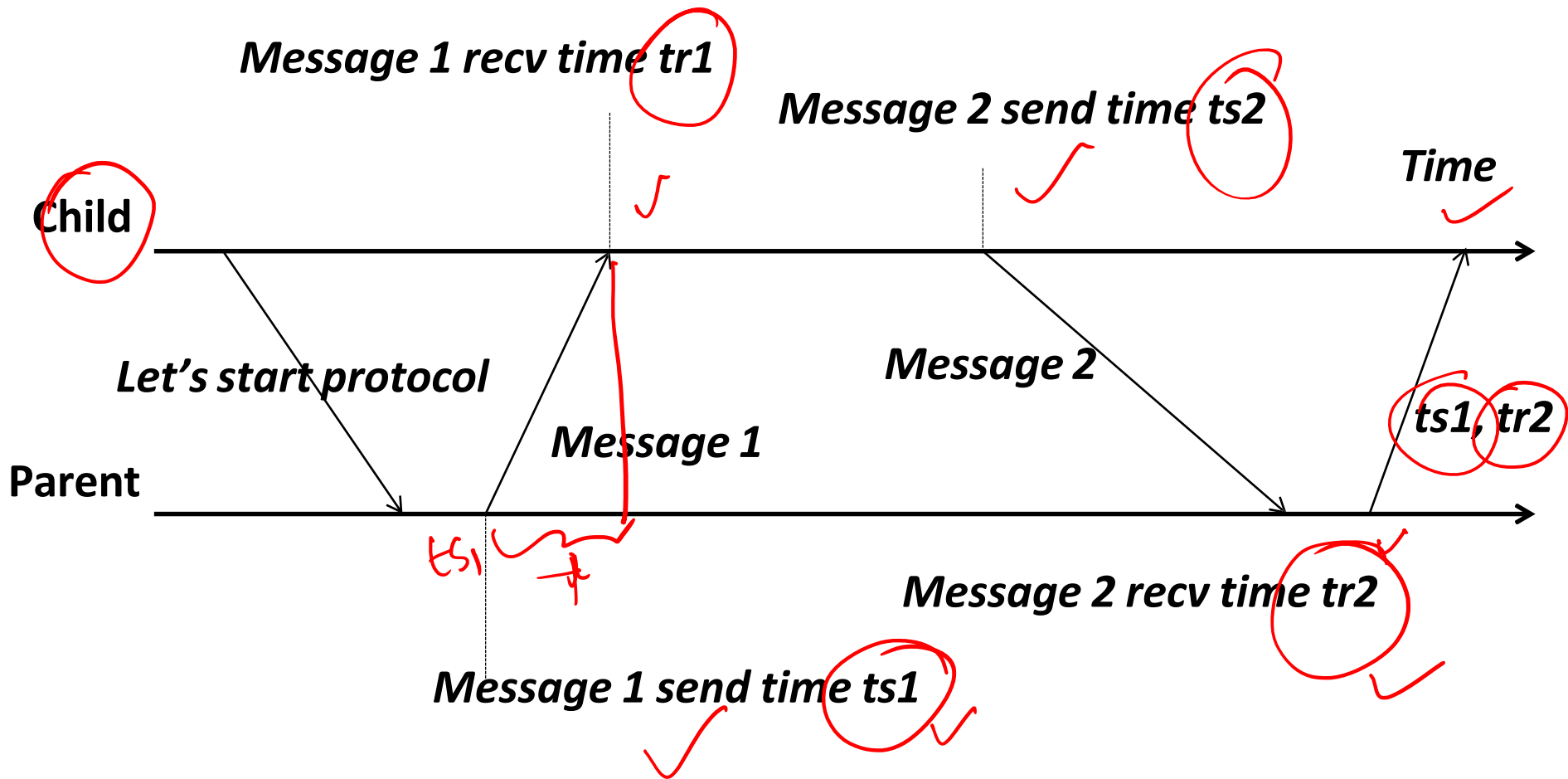


(ii) NTP: Network time protocol ✓

- (1991, 1992) Internet Standard, version 3: RFC 1305
- **NTP servers organized in a tree.** ✓
- Each client = a leaf of a tree.
- Each node synchronizes with its tree parent ✓



NTP Protocol



Why $o = (tr1 - tr2 + ts2 - ts1)/2$?

- Offset $o = (tr1 - tr2 + ts2 - ts1)/2$
- Let's calculate the error.
- Suppose real offset is **oreal**
 - Child is ahead of parent by oreal.
 - Parent is ahead of child by $-oreal$.
- Suppose one way latency of Message 1 is $L1$.
($L2$ for Message 2)
- No one knows $L1$ or $L2$!
- **Then**
 - $tr1 = ts1 + L1 + oreal$
 - $tr2 = ts2 + L2 - oreal$

Why $o = (tr1 - tr2 + ts2 - ts1)/2$?

- **Then**

- $tr1 = ts1 + L1 + o_{real}$.
- $tr2 = ts2 + L2 - o_{real}$.

- **Subtracting second equation from first**

- $o_{real} = (tr1 - tr2 + ts2 - ts1)/2 - (L2 - L1)/2$

- $\Rightarrow o_{real} = \overset{\checkmark}{o} + (L2 - L1)/2$

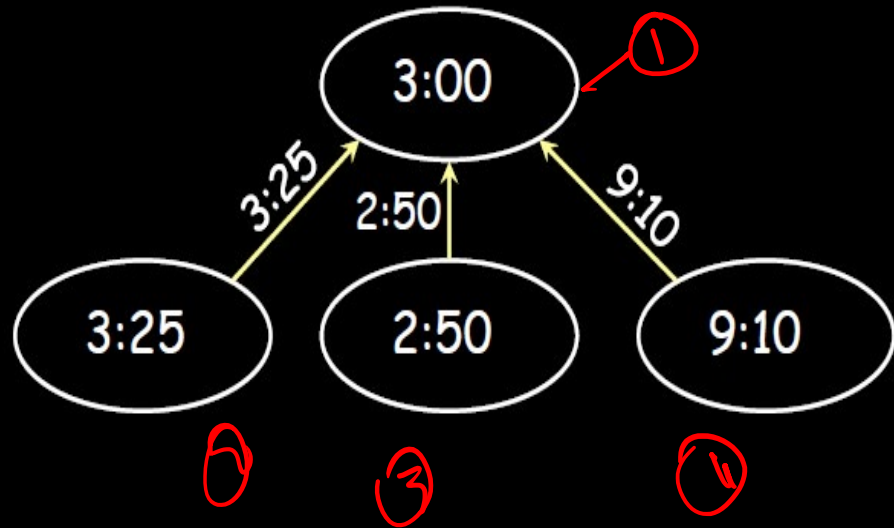
- $\Rightarrow |o_{real} - o| < |(L2 - L1)/2| < |(L2 + L1)/2|$

- Thus the error is bounded by the round trip time (RTT)

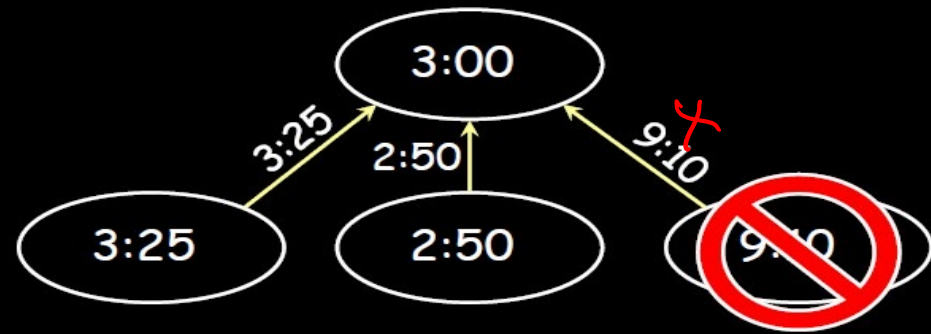
(iii) Berkley's Algorithm

- **Gusella & Zatti, 1989**
- Master poll's each machine periodically
 - Ask each machine for time
 - Can use Christian's algorithm✓ to compensate the network's latency.
- When results are in compute,
 - Including master's time.
- **Hope: average cancels out individual clock's tendency to run fast or slow**
- Send offset by which each clock needs adjustment to each slave
 - Avoids problems with network delays if we send a time-stamp.

Berkley's Algorithm : Example

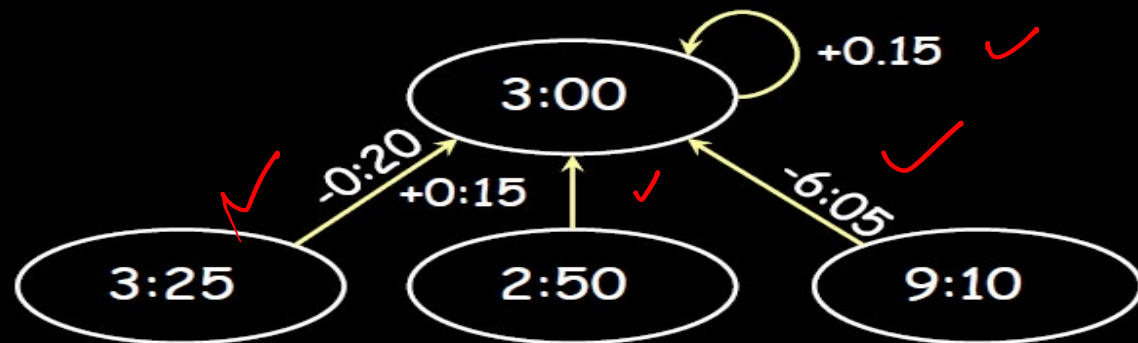


1. Request timestamps from all slaves



2. Compute fault-tolerant average: ✓

$$\checkmark \frac{3:25 + 2:50 + 3:00}{3} = 3:05 \checkmark$$



3. Send offset to each client

Internal clock system

(iv) DTP: Datacenter Time Protocol

Globally Synchronized Time via Datacenter Networks

Ki Suh Lee, Han Wang, Vishal Shrivastav, Hakim Weatherspoon
Computer Science Department
Cornell University
kslee,hwang,vishal,hweather@cs.cornell.edu

ACM SIGCOMM 2016

Application

Transport

Network

Data Link

Physical

- DTP uses the physical layer of network devices to implement a decentralized clock synchronization protocol.
- ***Highly Scalable with bounded precision!***
 - ~25ns (4 clock ticks) between peers
 - ~150ns for a datacenter with six hops
 - No Network Traffic
 - *Internal Clock Synchronization*
- End-to-End: ~200ns precision!

DTP: Phases

Application

Transport

Network

Data Link

Physical

- Runs in two phases between two peers
 - Init Phase: Measuring OWD (one-way delay)
 - Beacon Phase: Re-Synchronization



Physical

*local
delay*



Physical

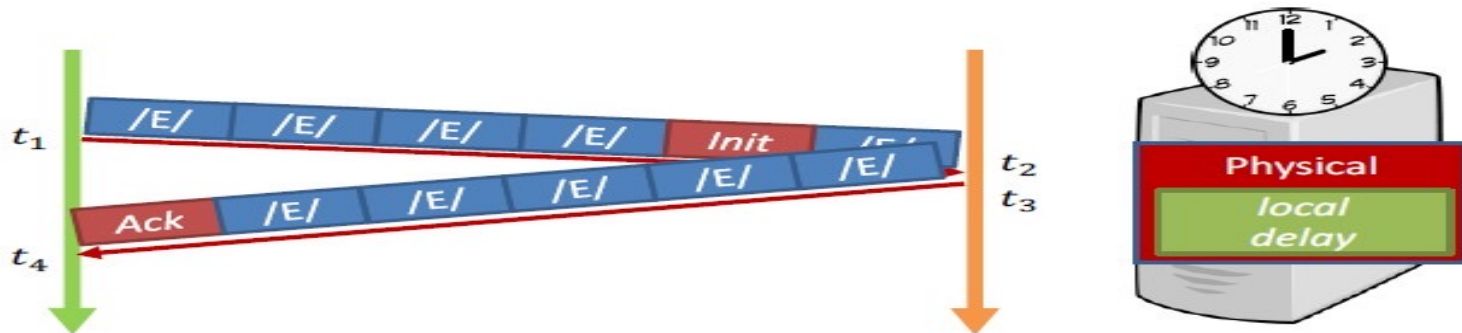
*local
delay*

DTP: (i) Init Phase

- **INIT phase:** The purpose of the INIT phase is to measure the one-way delay between two peers. The phase begins when two ports are physically connected and start communicating, i.e. when the link between them is established.
- Each peer measures the one-way delay by measuring the time between sending an INIT message and receiving an associated INIT-ACK message, i.e. measure RTT, then divide the measured RTT by two.



- $delay = (t_4 - t_1 - \alpha) / 2$
 - $\alpha=3$: Ensure *delay* is always less than actual delay
- Introduce 2 clock tick errors
 - Due to oscillator skew, timing and Sync FIFO

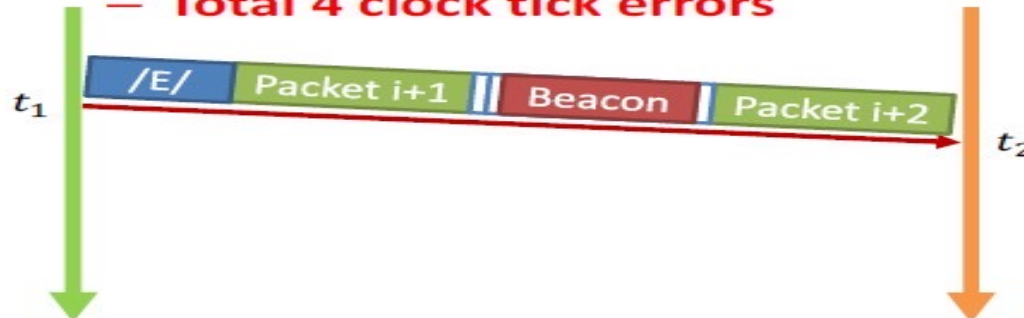


DTP: (ii) Beacon Phase

- BEACON phase:** During the BEACON phase, two ports periodically exchange their local counters for resynchronization. Due to oscillator skew, the offset between two local counters will increase over time. A port adjusts its local counter by selecting the maximum of the local and remote counters upon receiving a BEACON message from its peer. Since BEACON messages are exchanged frequently, hundreds of thousands of times a second (every few microseconds), the offset can be kept to a minimum.



- $local = \max(local, remote + delay)$
- Frequent messages
 - Every 1.2 us (200 clock ticks) with MTU packets
 - Every 7.2 us (1200 clock ticks) with Jumbo packets
- Introduces 2 clock tick errors
 - Total 4 clock tick errors**



DTP Switch

Application

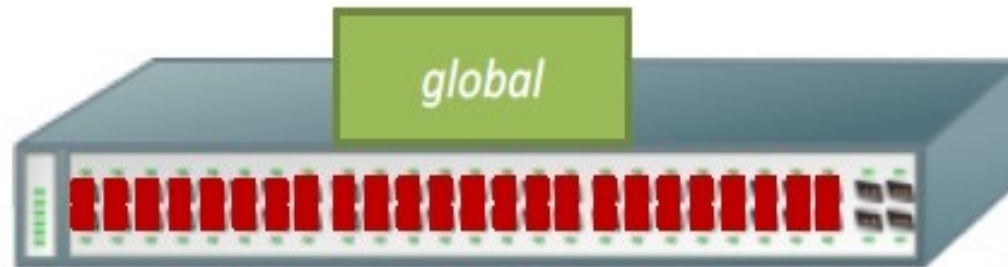
Transport

Network

Data Link

Physical

- $global = \max(local \text{ counters})$
- Propagates *global* via Beacon messages



max

Physical

*local
delay*

Physical

*local
delay*

Physical

*local
delay*

Physical

*local
delay*

Physical

*local
delay*

DTP Property

- DTP provides bounded **precision** and **scalability**
- **Bounded Precision in hardware**
 - Bounded by $4T$ ($=25.6\text{ns}$, $T=\text{oscillator tick is } 6.4\text{ns}$)
 - Network precision bounded by $4TD$
D is network diameter in hops
- **Requires NIC and switch modifications**

But Yet...

- We still have a non-zero error!
- We just can't seem to get rid of error
 - Can't as long as messages latencies are non-zero.
- Can we avoid synchronizing clocks altogether, and still be able to order events?

Physical clock
├── External
│ ├── UTC
│ └── NTP
└── Internal
 └── DTP

Ordering events in a distributed system

- To order events across processes, trying to synchronize clocks is an approach.
- What if we instead assigned timestamps to events that were not absolute time ?
- As long as those timestamps obey causality, that would work
 - If an event A causally happens before another event B, then $\text{timestamp}(A) < \text{timestamp}(B)$
 - Example: Humans use causality all the time
 - I enter the house only if I unlock it
 - You receive a letter only after I send it

Logical (or Lamport) ordering

- **Proposed by Leslie Lamport in the 1970s.**
- Used in almost all distributed systems since then
- Almost all cloud computing systems use some form of logical ordering of events.



- **Leslie B. Lamport** (born February 7, 1941) is an American computer scientist. Lamport is best known for his seminal work in distributed systems and as the initial developer of the document preparation system LaTeX. Leslie Lamport was the winner of the **2013 Turing Award** for imposing clear, well-defined coherence on the seemingly chaotic behavior of distributed computing systems, in which several autonomous computers communicate with each other by passing messages.

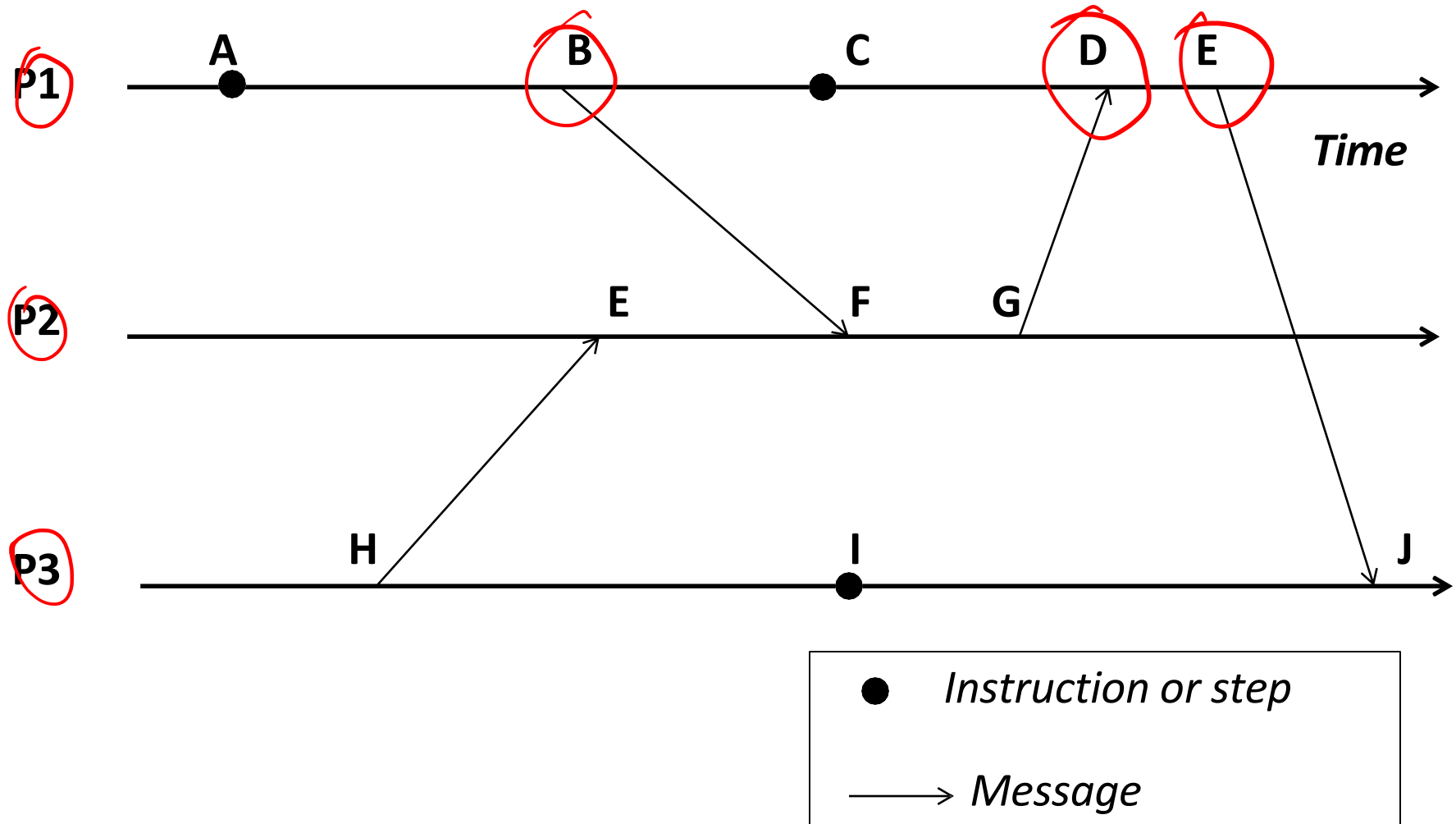
Lamport's research contributions

- **Lamport's research contributions have laid the foundations of the theory of distributed systems. Among his most notable papers are**
 - “Time, Clocks, and the Ordering of Events in a Distributed System”, which received the PODC Influential Paper Award in 2000,
 - “How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs”, which defined the notion of Sequential consistency,
 - “The Byzantine Generals' Problem”,
 - “Distributed Snapshots: Determining Global States of a Distributed System” and
 - “The Part-Time Parliament”.
- These papers relate to such concepts as logical clocks (and the *happened-before* relationship) and Byzantine failures. They are among the most cited papers in the field of computer science and describe algorithms to solve many fundamental problems in distributed systems, including:
 - the Paxos algorithm for consensus,
 - the bakery algorithm for mutual exclusion of multiple threads in a computer system that require the same resources at the same time,
 - the Chandy-Lamport algorithm for the determination of consistent global states (snapshot), and
 - the Lamport signature, one of the prototypes of the digital signature.

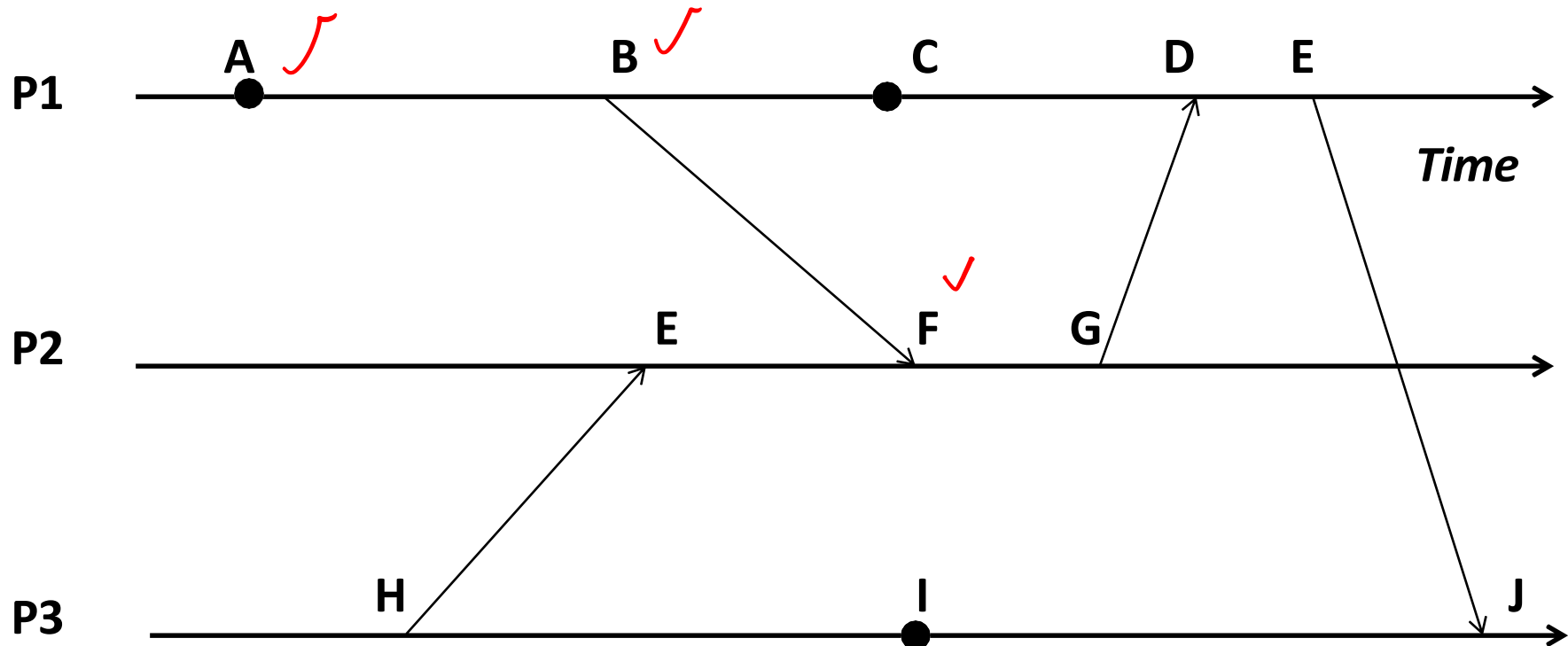
Logical (or Lamport) Ordering(2)

- Define a logical relation **Happens-Before** among pairs of events
- **Happens-Before** denoted as \rightarrow
- **Three rules:**
 1. On the same process: $a \rightarrow b$, if $time(a) < time(b)$ (using the local clock)
 2. If p1 sends m to p2: $send(m) \rightarrow receive(m)$
 3. **(Transitivity)** If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Creates a partial order among events
 - Not all events related to each other via \rightarrow

Example 1:



Example 1: Happens-Before

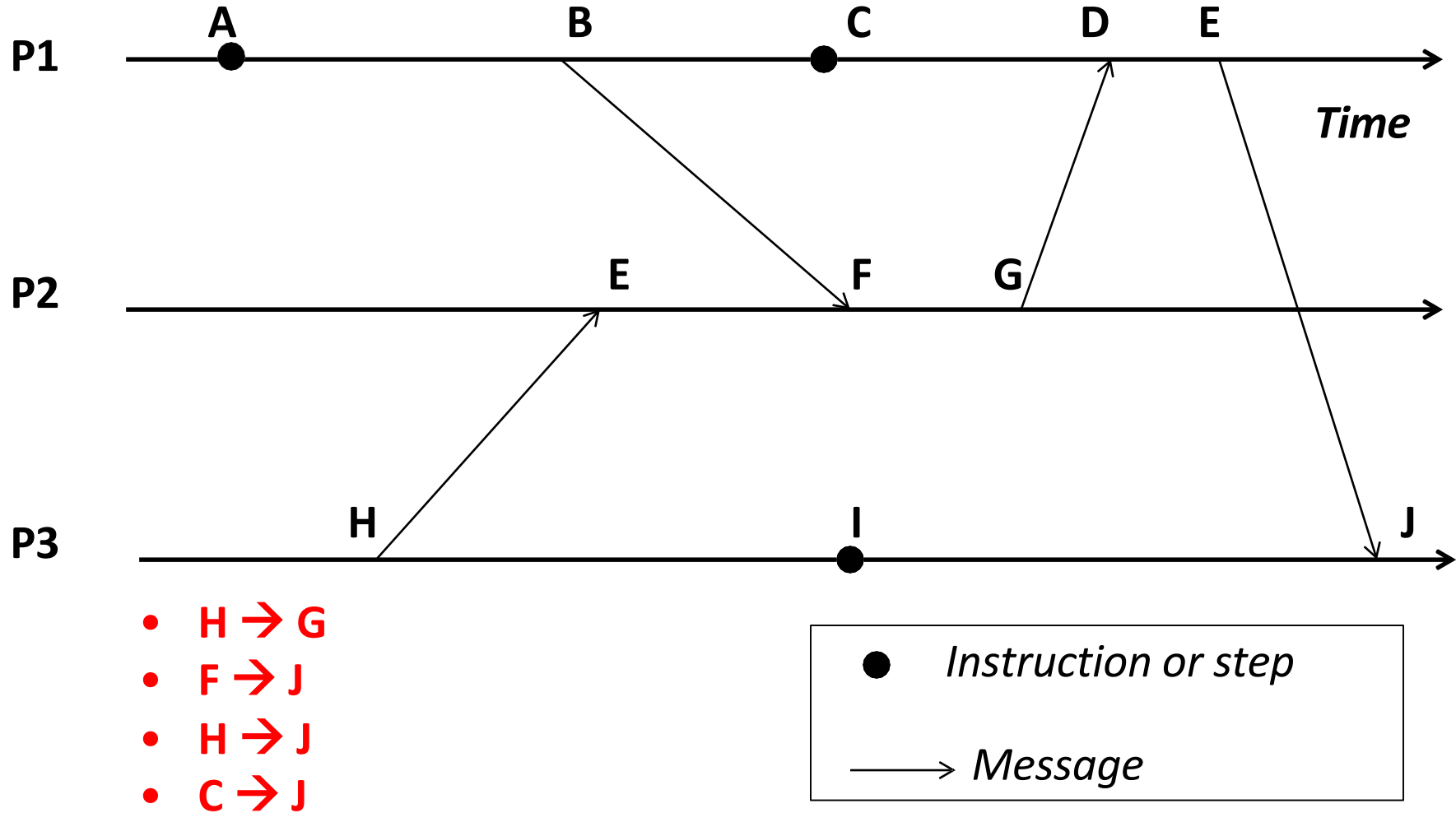


- $A \rightarrow B$
- $B \rightarrow F$
- $A \rightarrow F$ ✓

● *Instruction or step*

→ *Message*

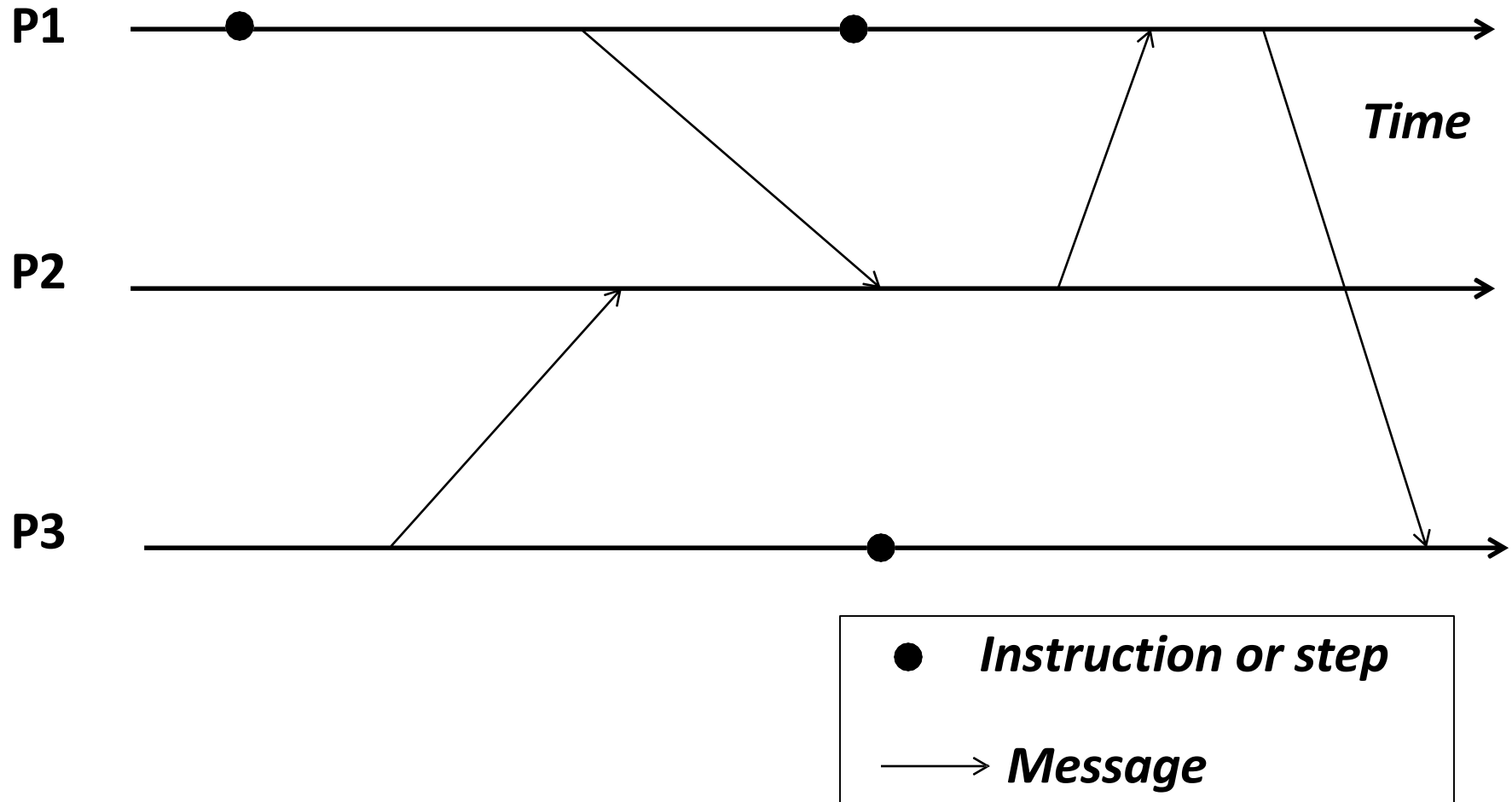
Example 2: Happens-Before



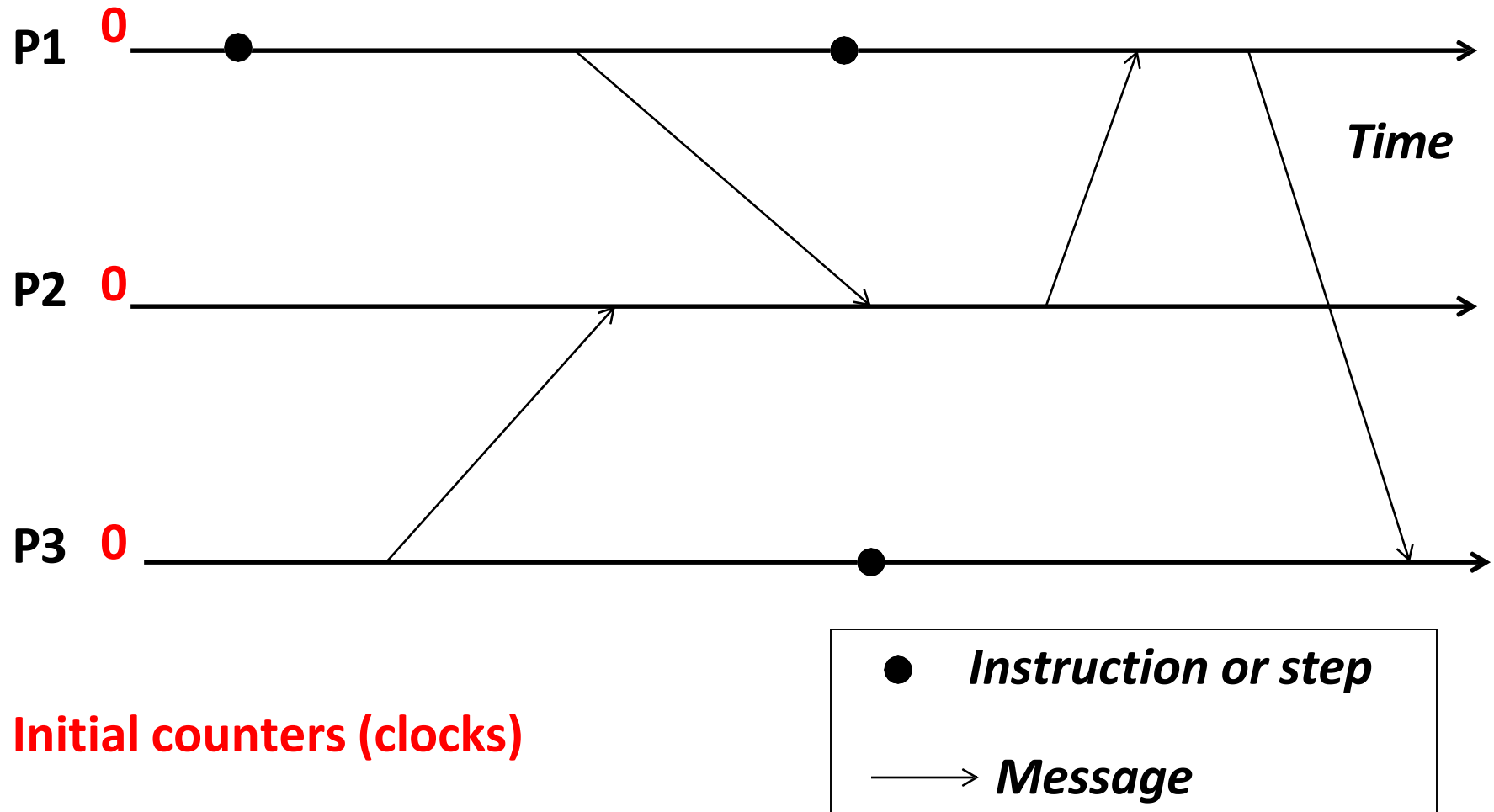
Lamport timestamps

- **Goal: Assign logical (Lamport) timestamp to each event**
- **Timestamps obey causality**
- **Rules**
 - Each process uses a local counter (clock) which is an integer
 - initial value of counter is zero
 - A process increments its counter when a **send** or an **instruction** happens at it. The counter is assigned to the event as its timestamp.
 - A **send (message)** event carries its timestamp
 - For a **receive (message)** event the counter is updated by
 - $\max(\text{local clock}, \text{message timestamp}) + 1$

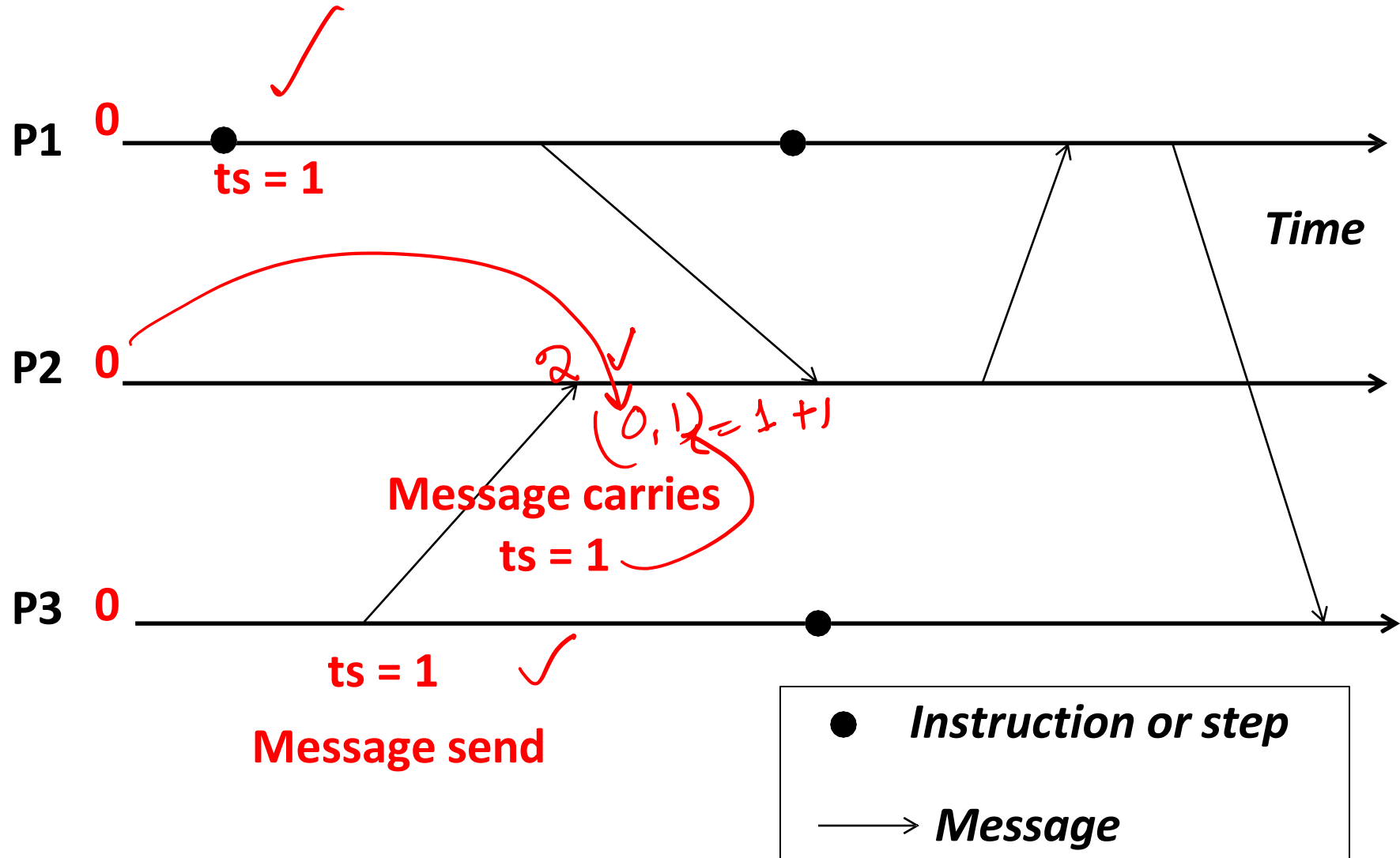
Example



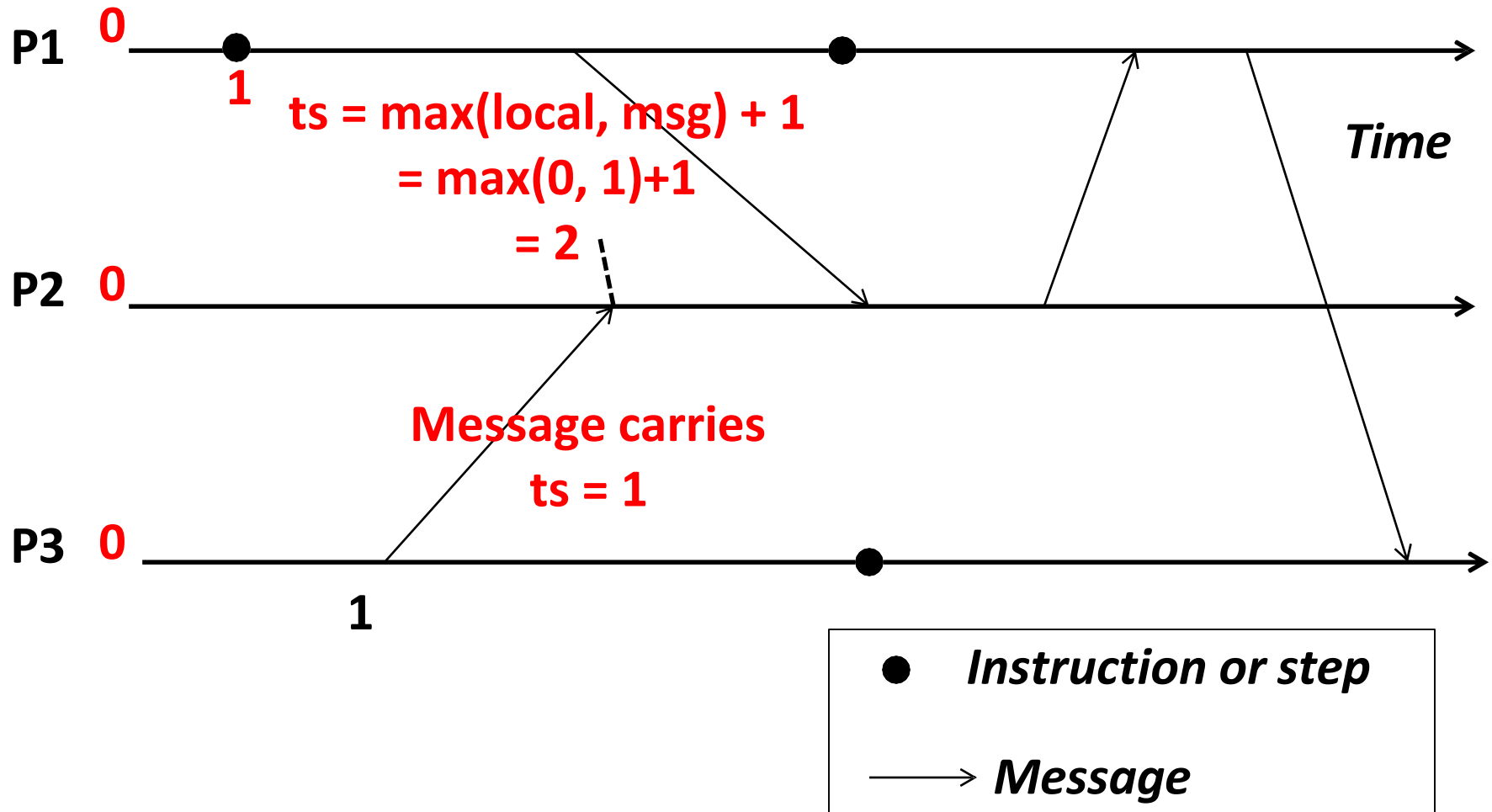
Lamport Timestamps



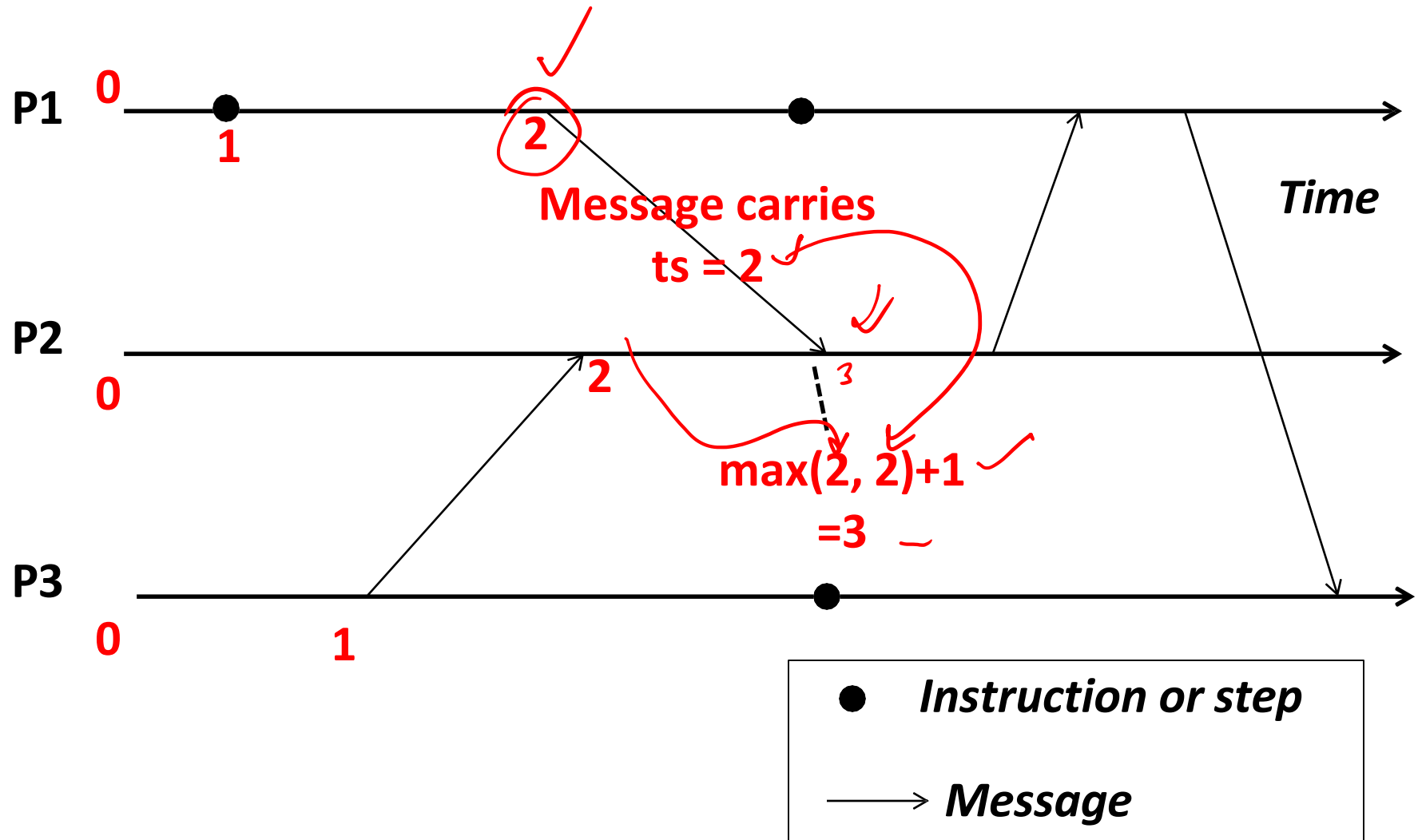
Lamport Timestamps



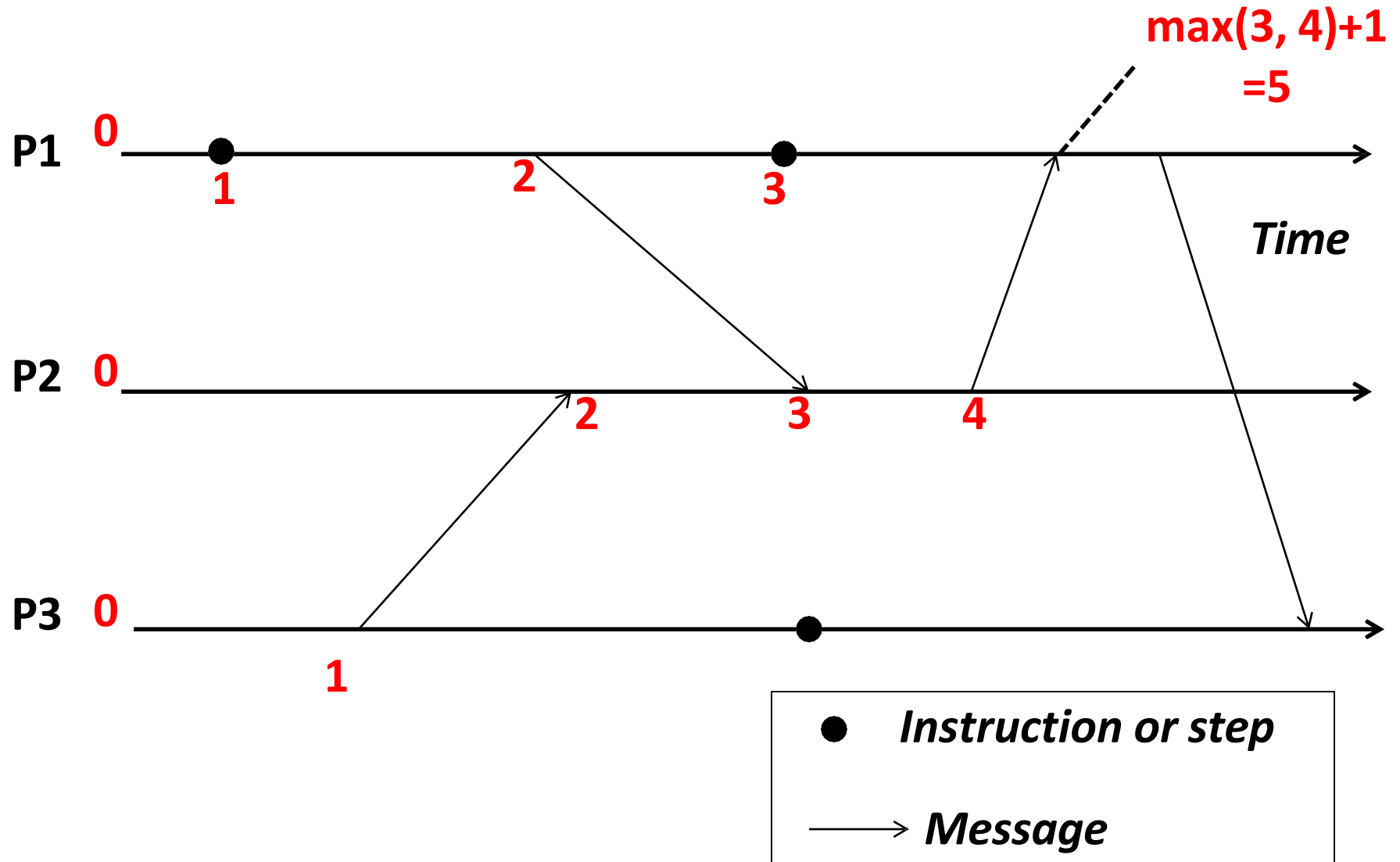
Lamport Timestamps



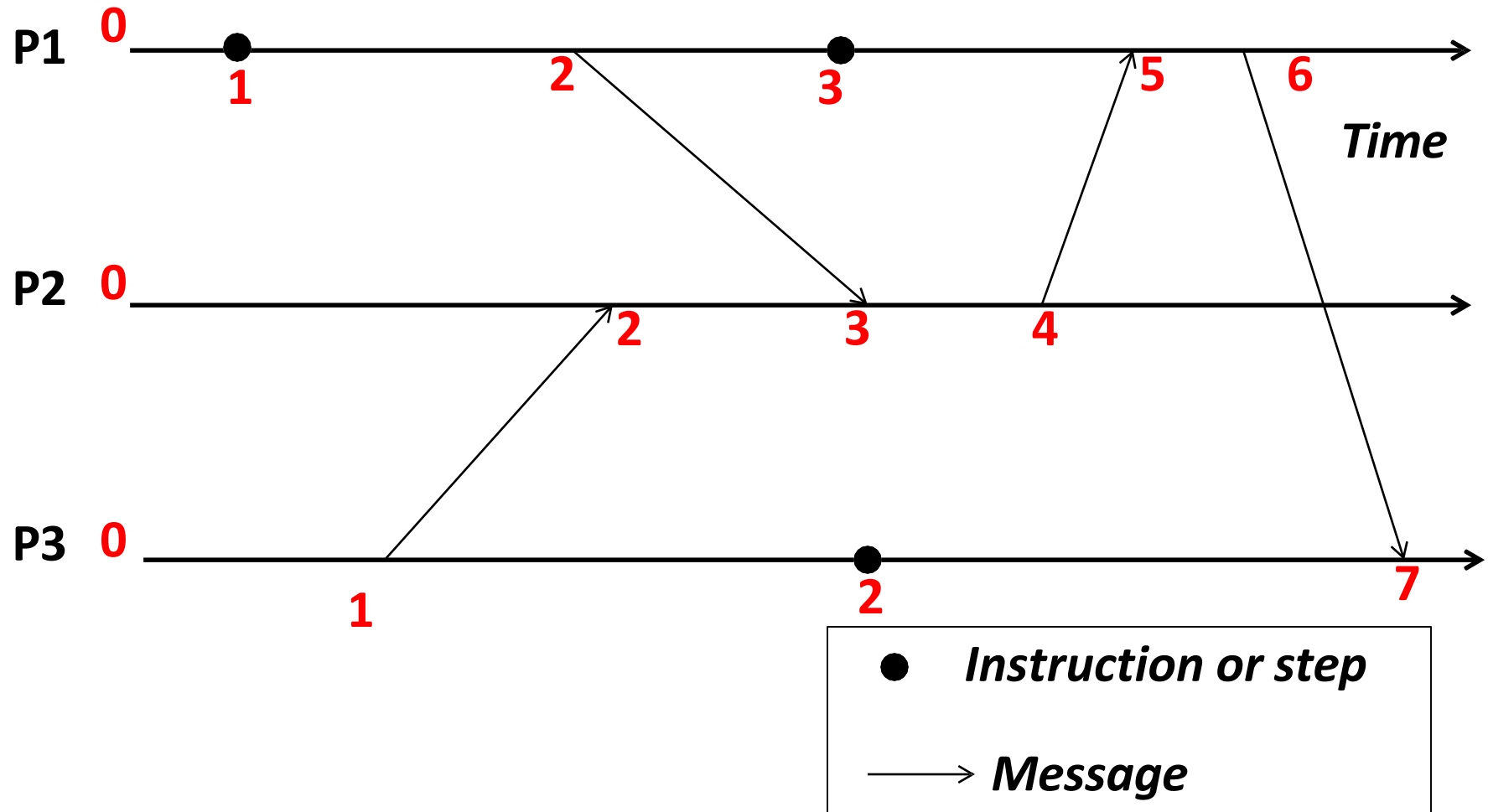
Lamport Timestamps



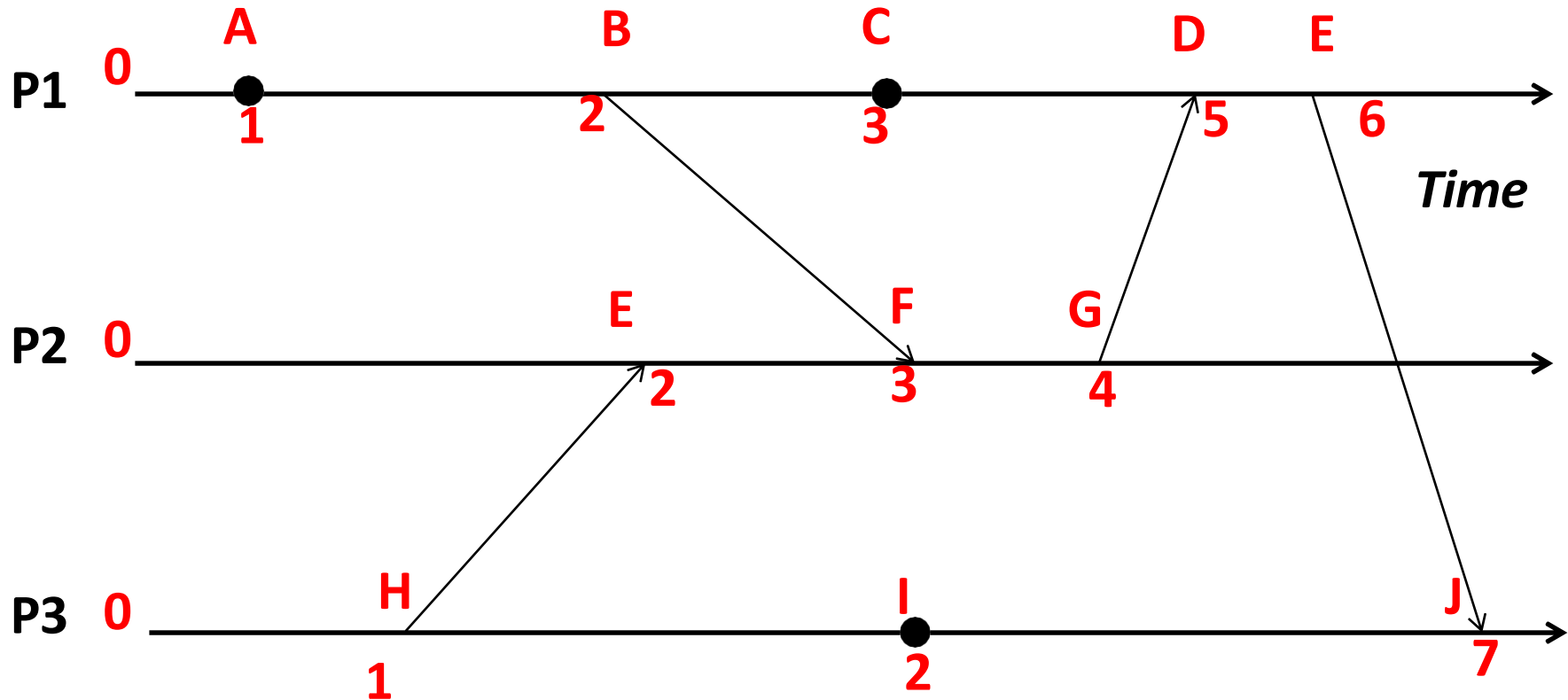
Lamport Timestamps



Lamport Timestamps



Obeying Causality

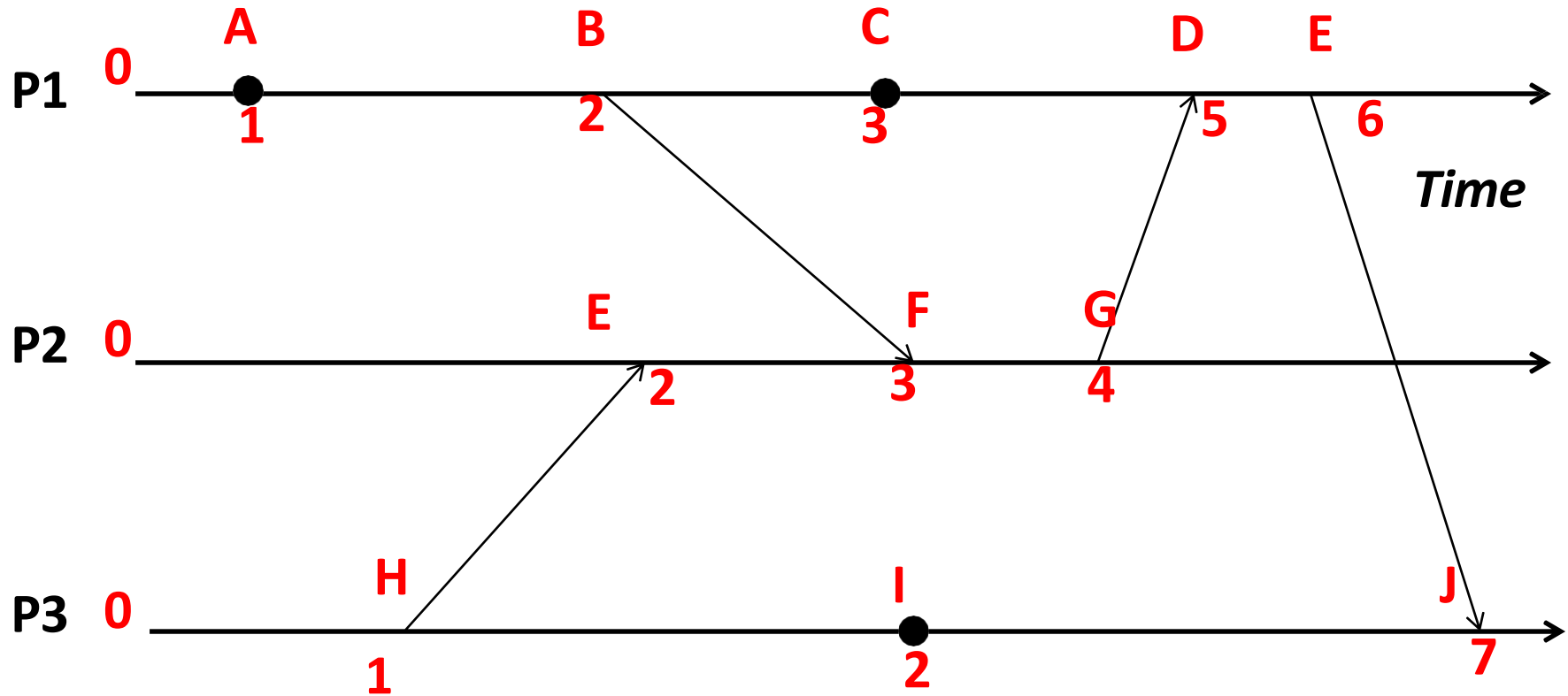


- $A \rightarrow B :: 1 < 2$
- $B \rightarrow F :: 2 < 3$
- $A \rightarrow F :: 1 < 3$

● *Instruction or step*

→ *Message*

Obeying Causality (2)



$H \rightarrow G :: 1 < 4$

$F \rightarrow J :: 3 < 7$

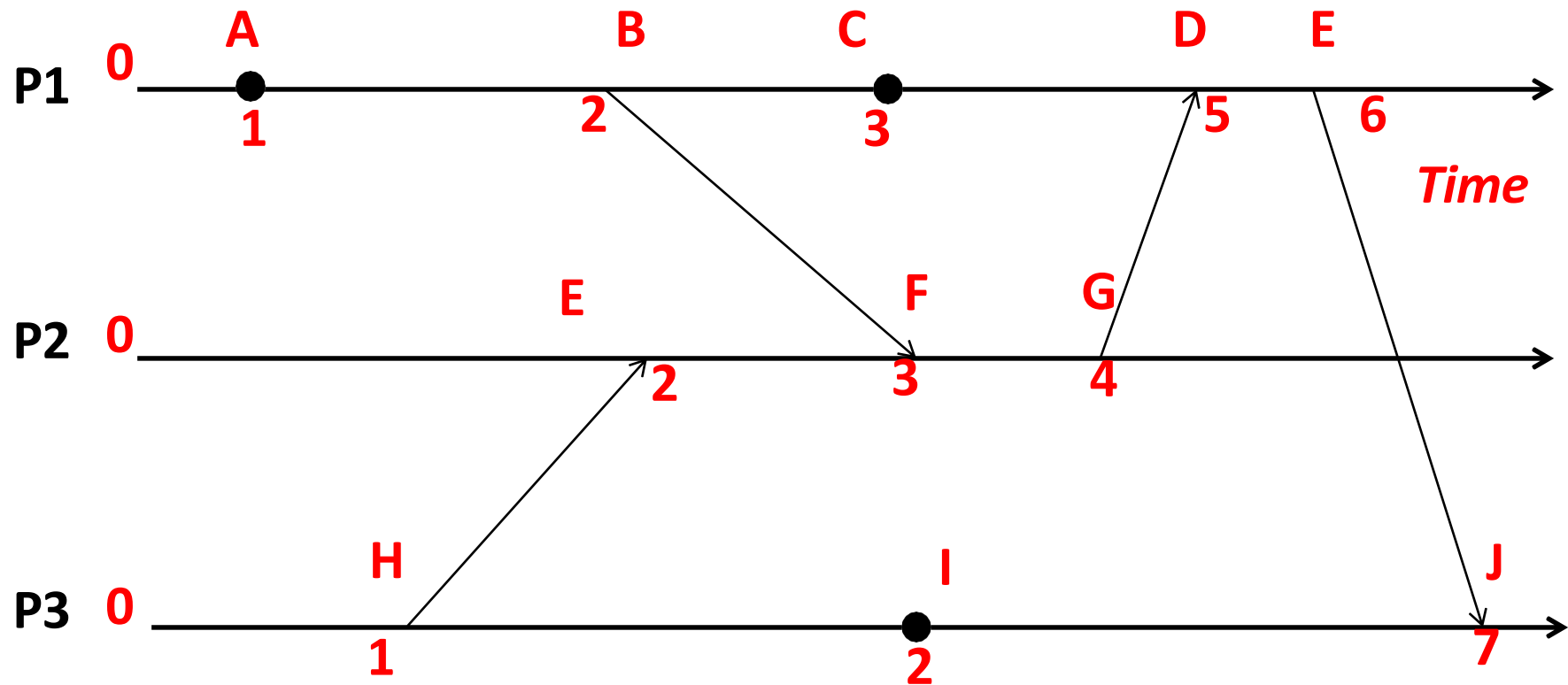
$H \rightarrow J :: 1 < 7$

$C \rightarrow J :: 3 < 7$

● *Instruction or step*

→ *Message*

Not always implying Causality



- $? C \rightarrow F ? :: 3 = 3$
- $? H \rightarrow C ? :: 1 < 3$
- (C, F) and (H, C) are pairs of concurrent events

● *Instruction or step*

→ *Message*

Concurrent Events

- A pair of concurrent events doesn't have a causal path from one event to another (either way, in the pair)
- Lamport timestamps not guaranteed to be ordered or unequal for concurrent events
- Ok, since concurrent events are not causality related!
- **Remember:**

$E1 \rightarrow E2 \Rightarrow \text{timestamp}(E1) < \text{timestamp}(E2),$ **BUT**

$\text{timestamp}(E1) < \text{timestamp}(E2) \Rightarrow$

$\{E1 \rightarrow E2\} \text{ OR } \{E1 \text{ and } E2 \text{ concurrent}\}$

Vector Timestamps

- Used in key-value stores like Riak ✓
- Each process uses a vector of integer clocks
- Suppose there are N processes in the group $1...N$
- Each vector has N elements ✓
- Process i maintains vector $V_i[1...N]$ ✓
- j th element of vector clock at process i , $V_i[j]$, is i 's knowledge of latest events at process j ✓
 $V_i[j]$

Assigning Vector Timestamps

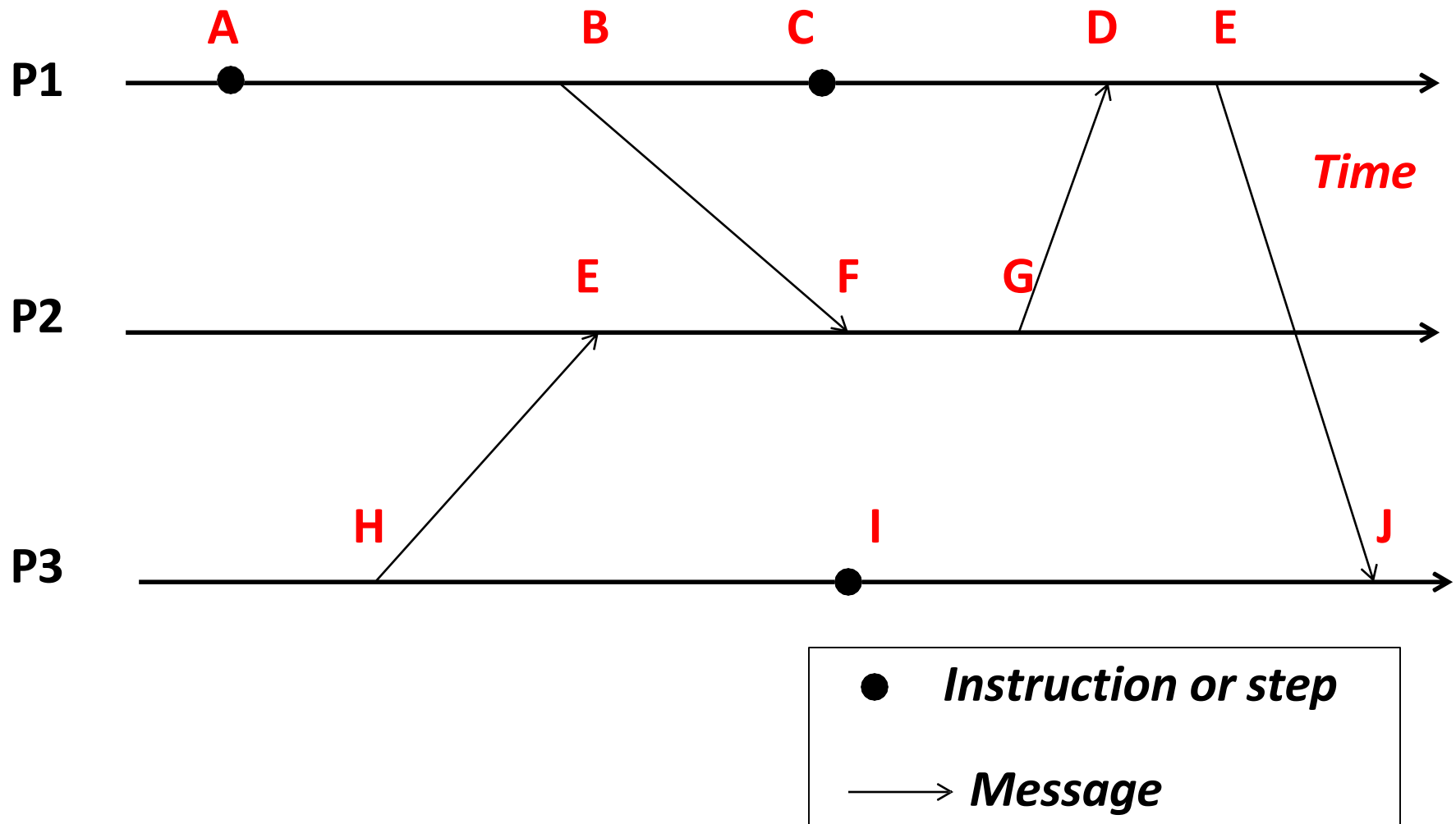
Incrementing vector clocks

1. On an instruction or send event at process i , it increments only its i th element of its vector clock ✓
2. Each message carries the send-event's vector timestamp
 $V_{\text{message}}[1...N]$ ✓
3. On receiving a message at process i :

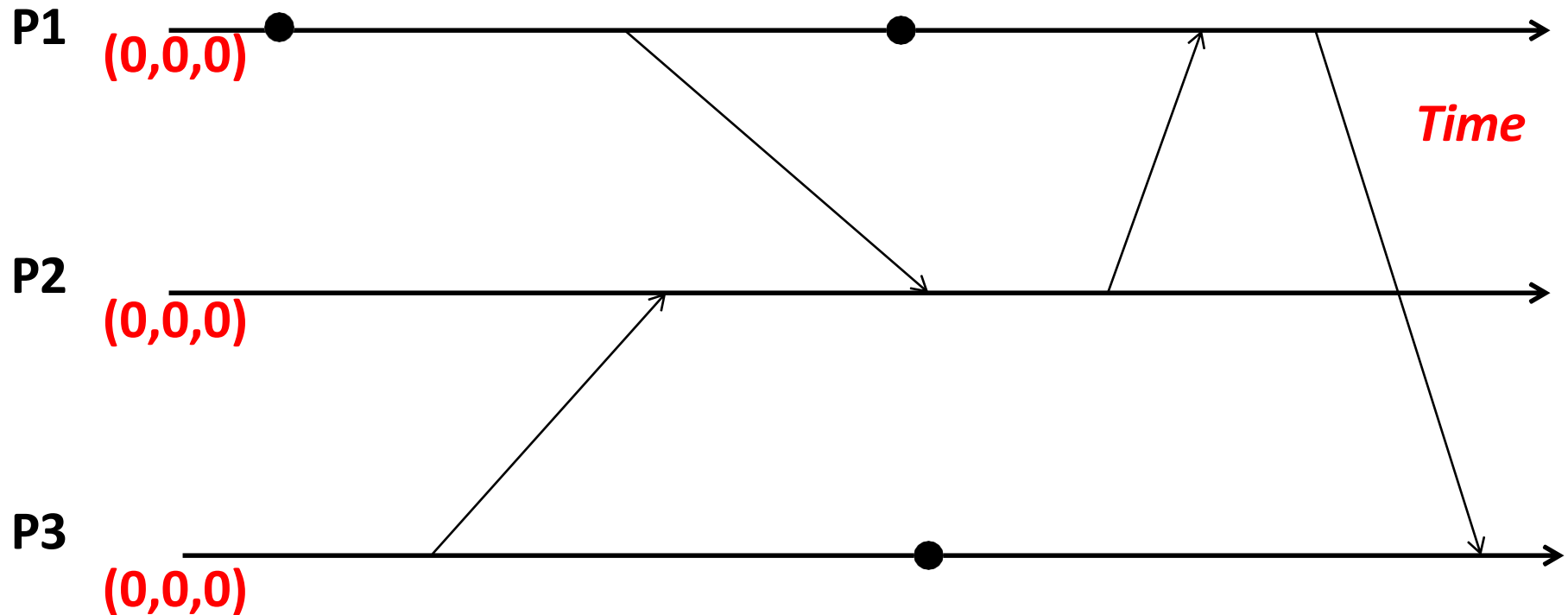
$$V_i[i] = V_i[i] + 1 \quad \checkmark$$

$$V_i[j] = \max(V_{\text{message}}[j], V_i[j]) \text{ for } \underline{j \neq i} \quad \checkmark$$

Example

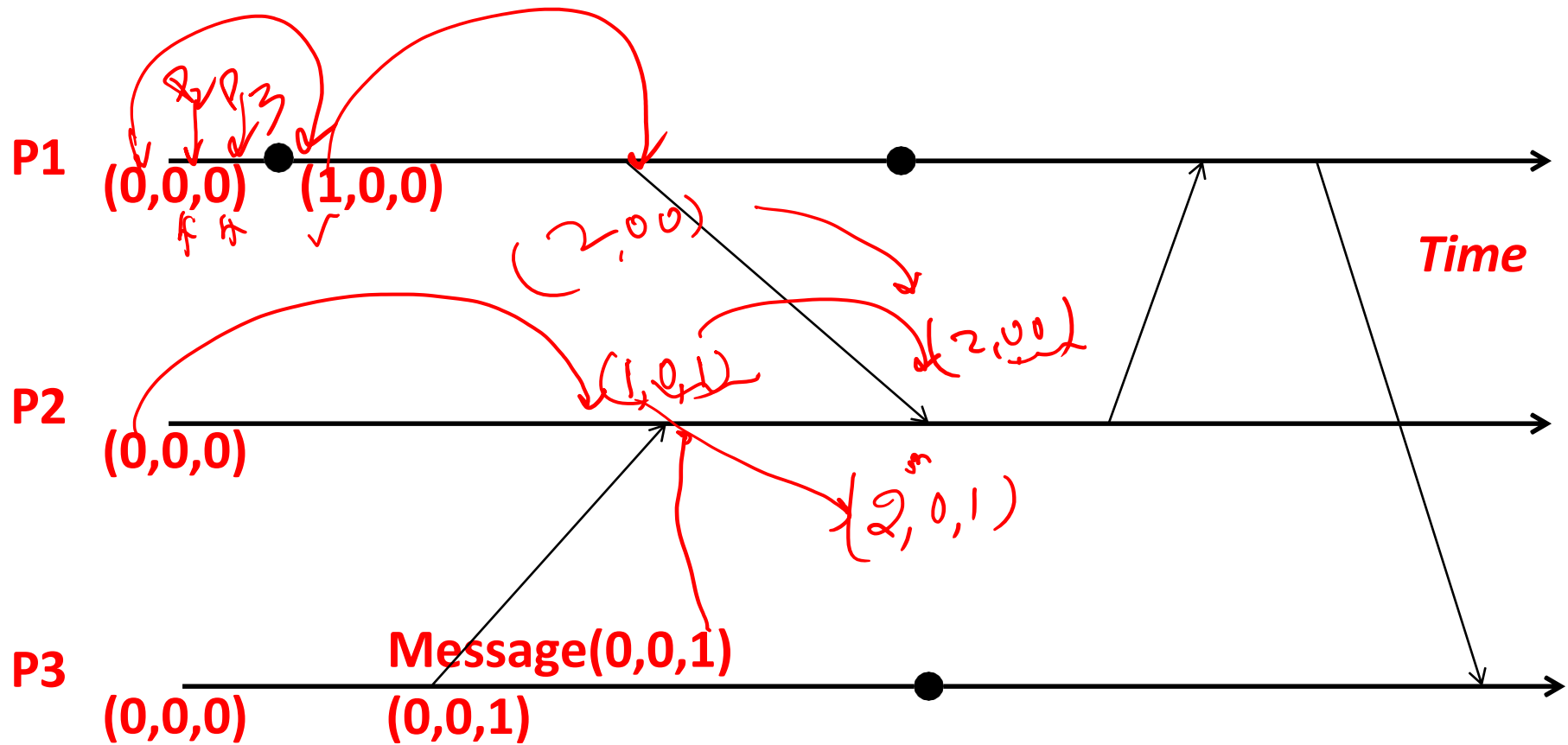


Vector Timestamps

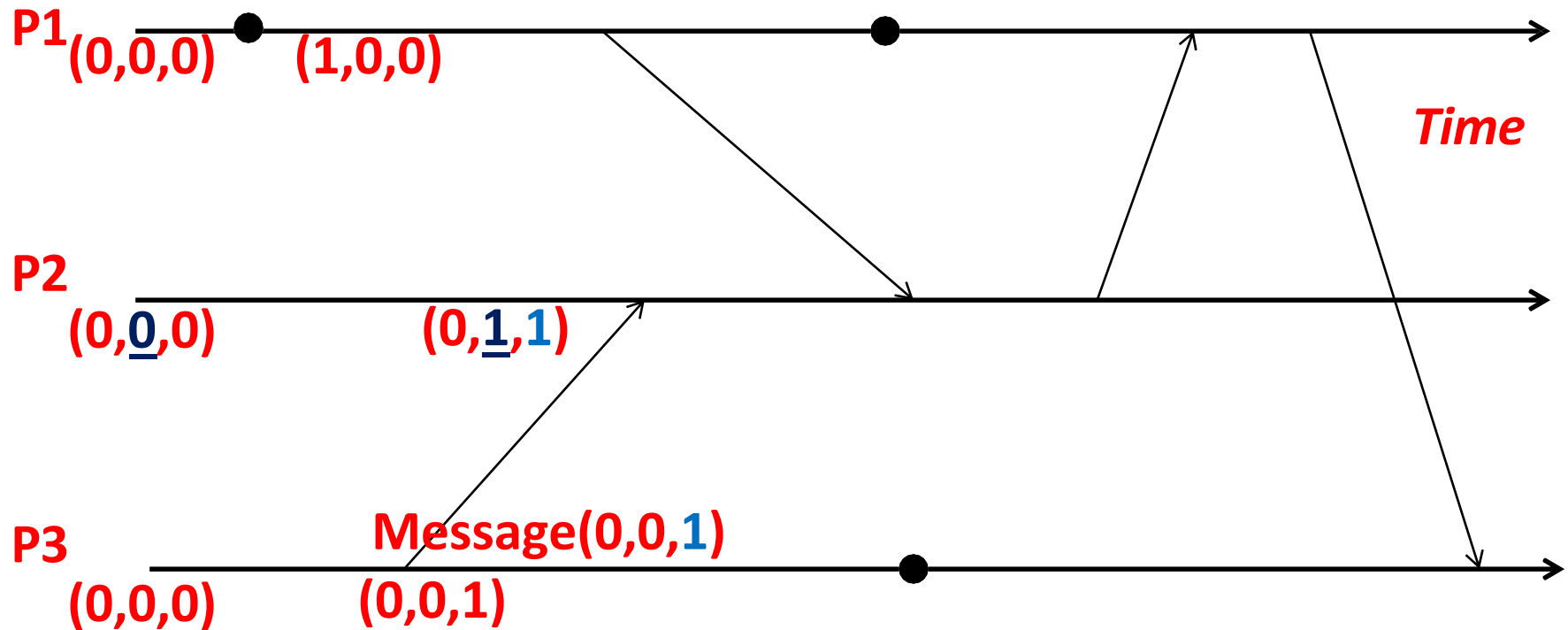


Initial counters (clocks)

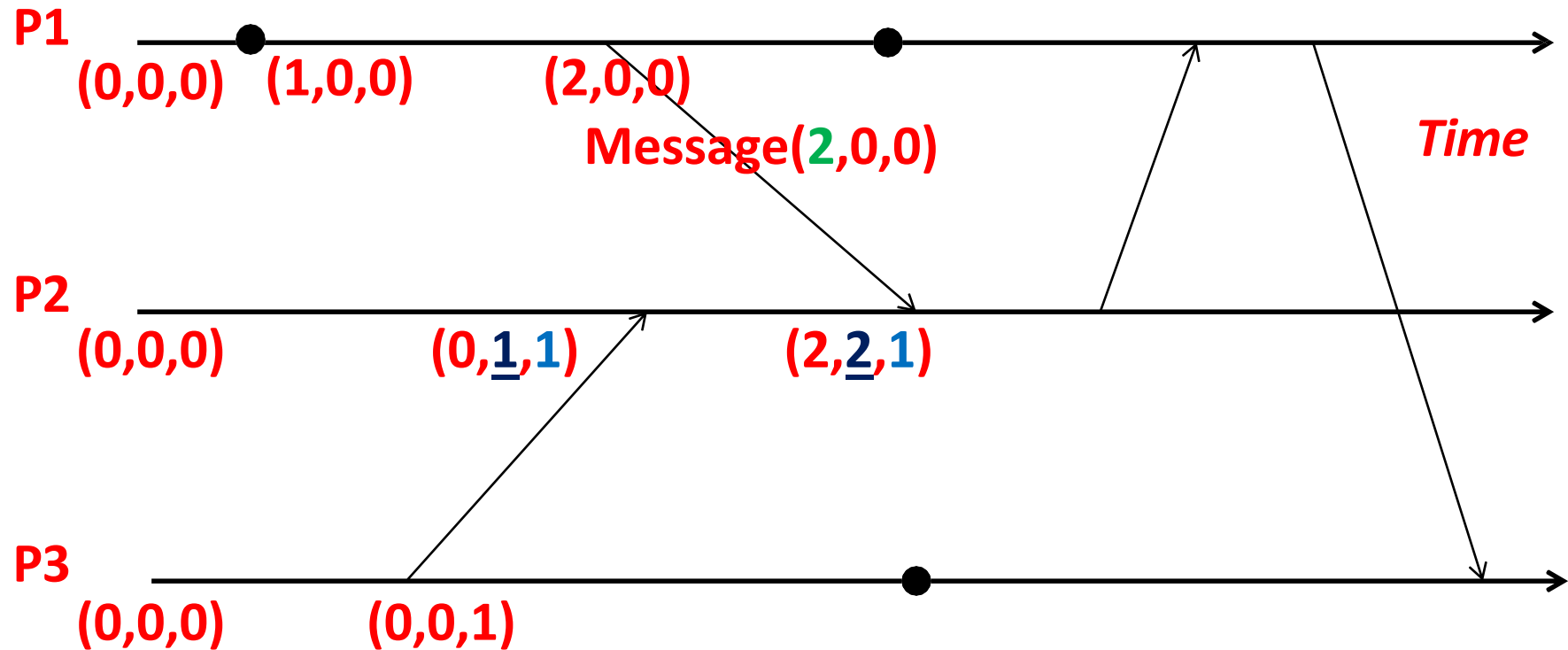
Vector Timestamps



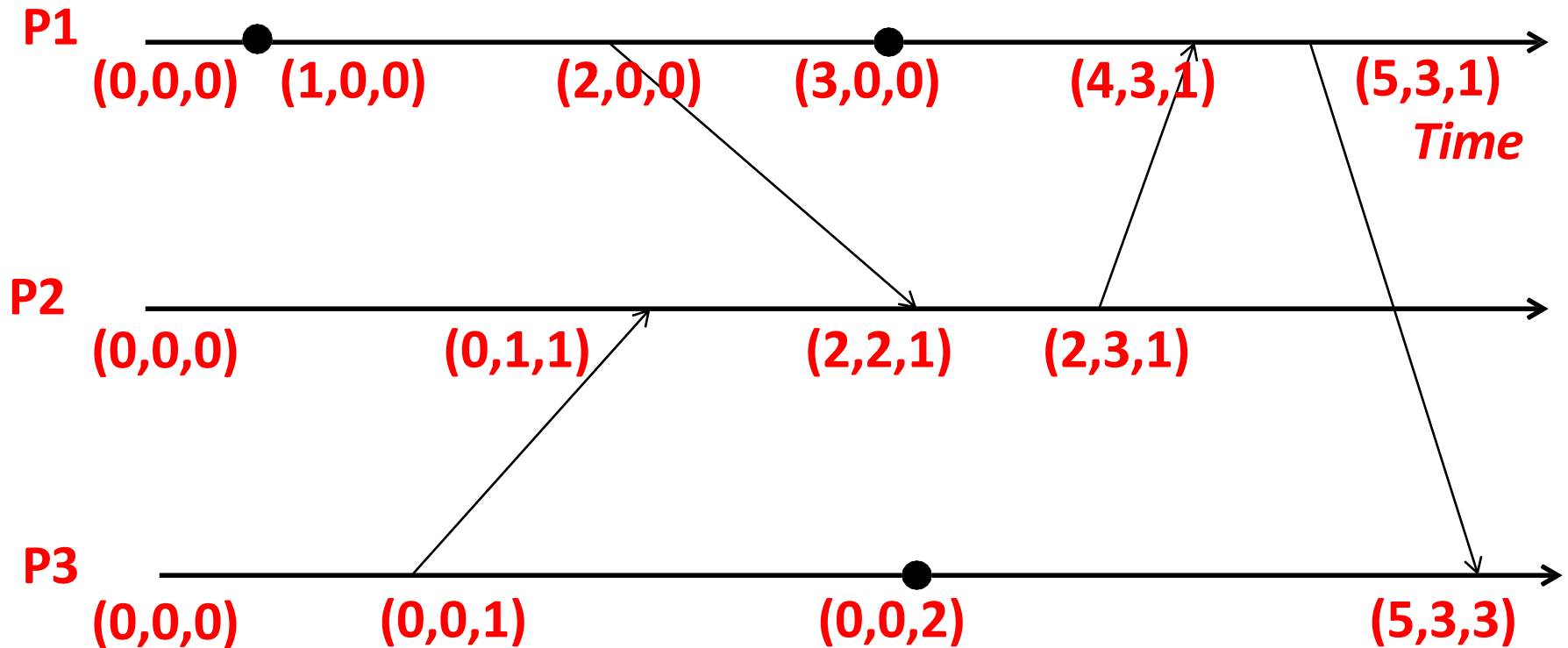
Vector Timestamps



Vector Timestamps



Vector Timestamps



Causally-Related

- $VT_1 = VT_2$,
iff (if and only if)
 $VT_1[i] = VT_2[i]$, for all $i = 1, \dots, N$
- $VT_1 \leq VT_2$,
iff $VT_1[i] \leq VT_2[i]$, for all $i = 1, \dots, N$
- **Two events are causally related *iff***
 $VT_1 < VT_2$, i.e.,
iff $VT_1 \leq VT_2$ &
there exists j such that
 $1 \leq j \leq N$ & $VT_1[j] < VT_2[j]$

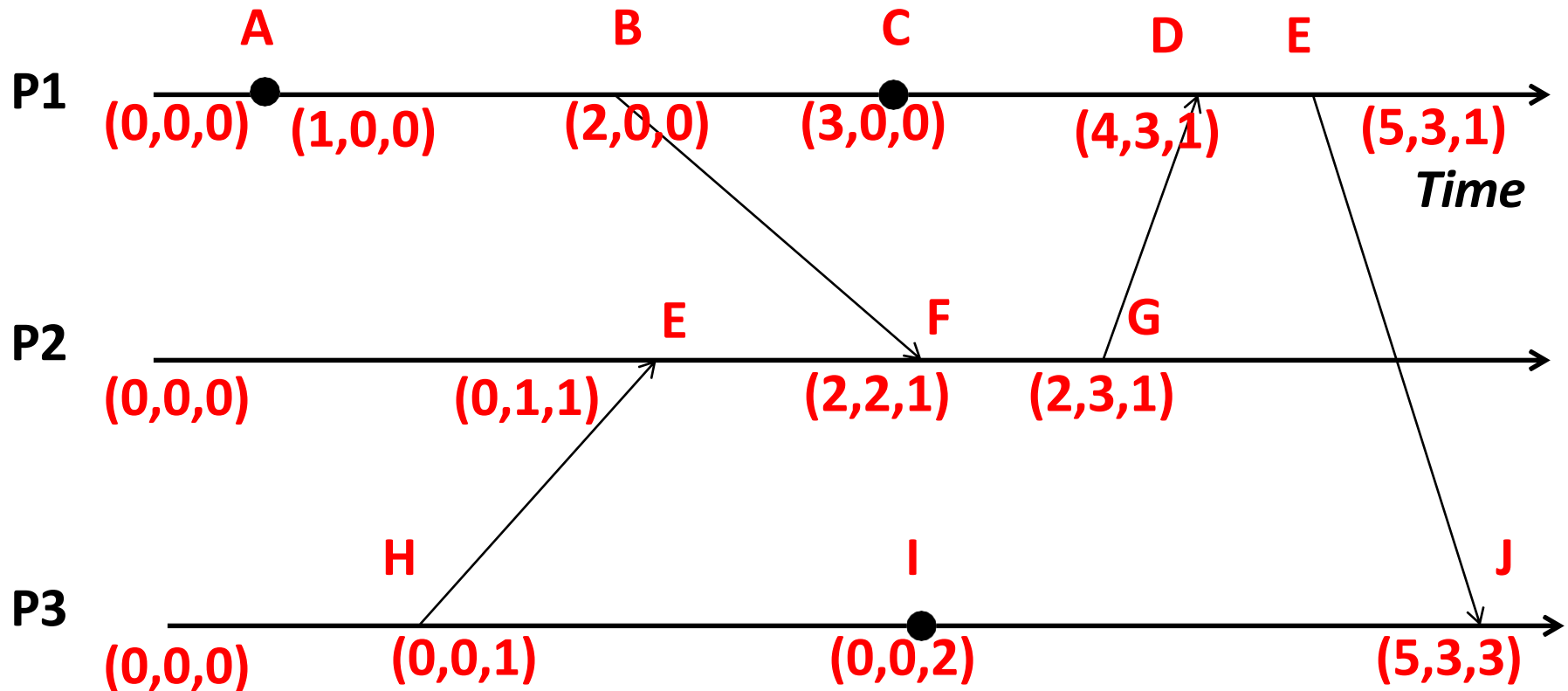
... or Not Causally-Related

- Two events VT_1 and VT_2 are **concurrent**
iff

$$\text{NOT } (VT_1 \leq VT_2) \text{ AND NOT } (VT_2 \leq VT_1)$$

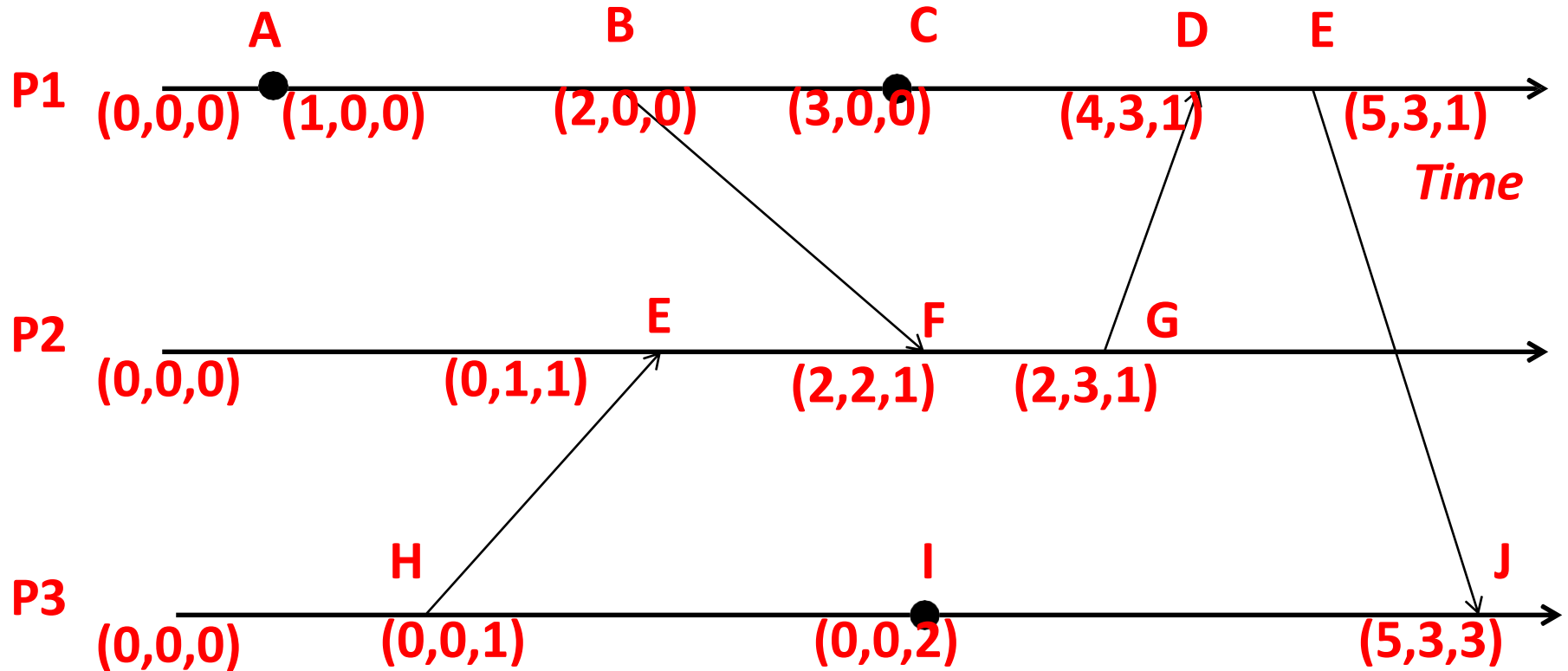
We'll denote this as $VT_2 ||| VT_1$

Obeying Causality



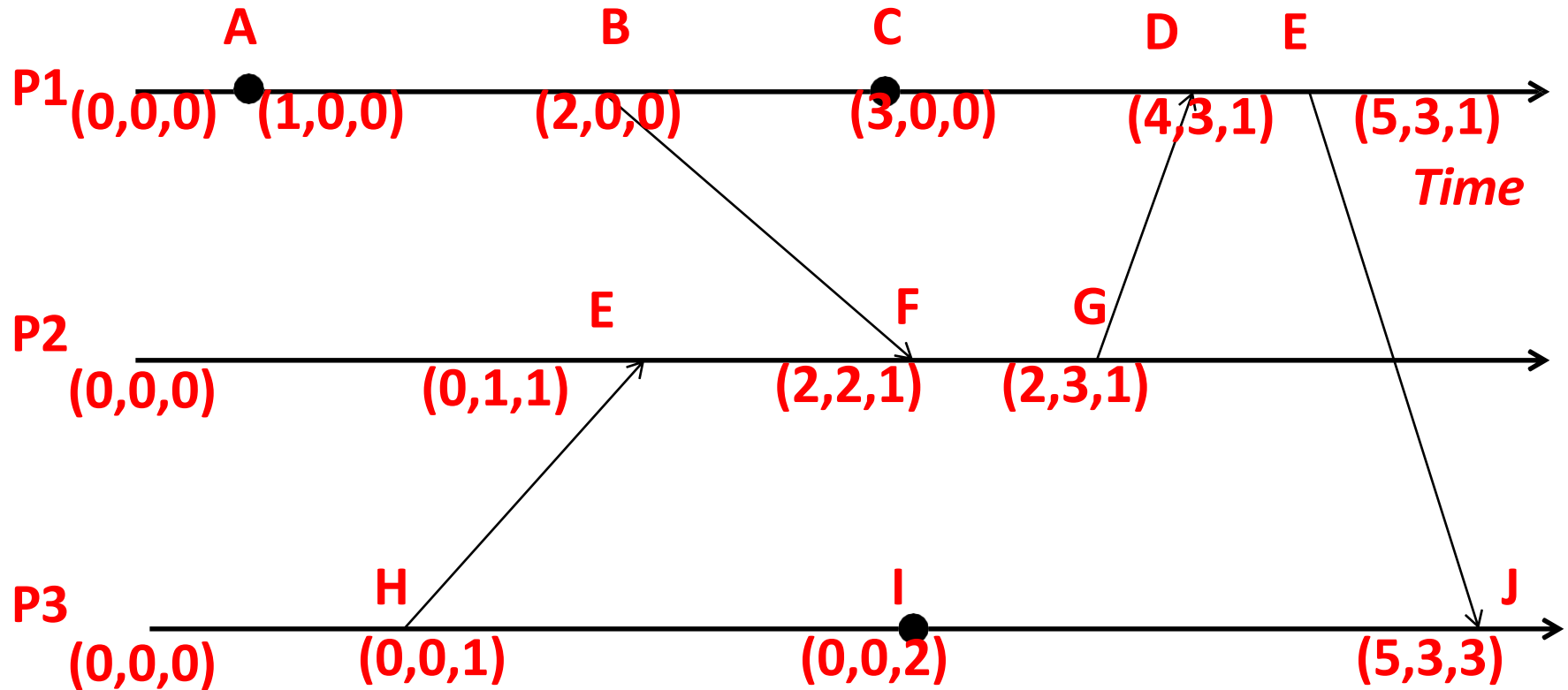
- $A \rightarrow B :: (1,0,0) < (2,0,0)$
- $B \rightarrow F :: (2,0,0) < (2,2,1)$
- $A \rightarrow F :: (1,0,0) < (2,2,1)$

Obeying Causality (2)



- $H \rightarrow G :: (0,0,1) < (2,3,1)$
- $F \rightarrow J :: (2,2,1) < (5,3,3)$
- $H \rightarrow J :: (0,0,1) < (5,3,3)$
- $C \rightarrow J :: (3,0,0) < (5,3,3)$

Identifying Concurrent Events



- C & F :: $(\underline{3},0,0) \parallel (2,2,\underline{1})$
- H & C :: $(0,0,\underline{1}) \parallel (\underline{3},0,0)$
- (C, F) and (H, C) are pairs of concurrent events

Conclusion

Internet of Things (IoT) devices that are wirelessly connected in mesh networks often need mutual clock time synchronization, to enable chronological ordering of sensor events, coordination of asynchronous processes across devices, or network-wide coordination of actuators. ✓

Time synchronization: ✓

- Christian's algorithm

- Berkeley algorithm

- NTP

- DTP

- But error a function of RTT

Can avoid time synchronization altogether by instead assigning logical timestamps to events

Enabling Intelligence at Edge layer for IOT



Dr. Rajiv Misra

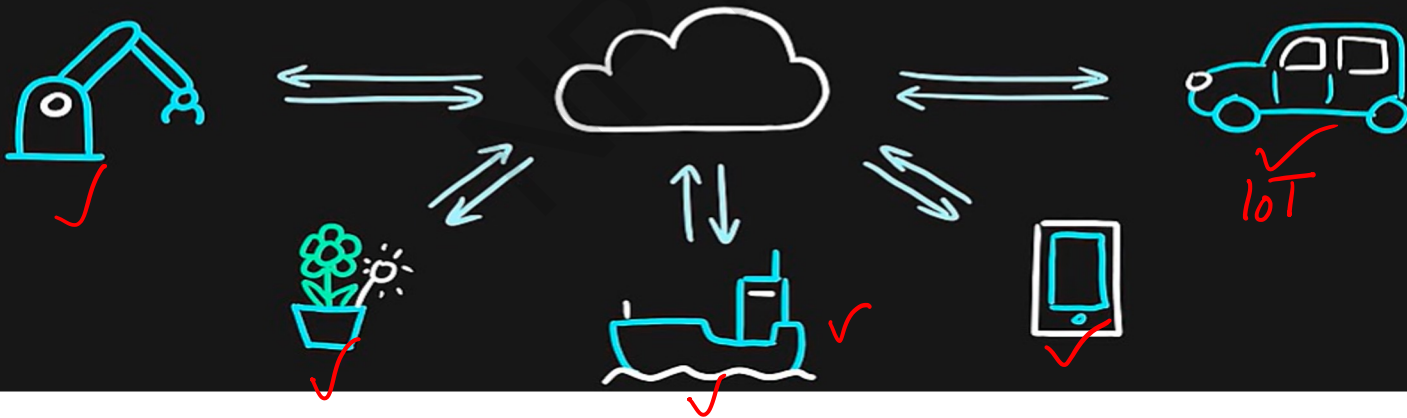
Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

After Completion of this lecture you will knowing the following:

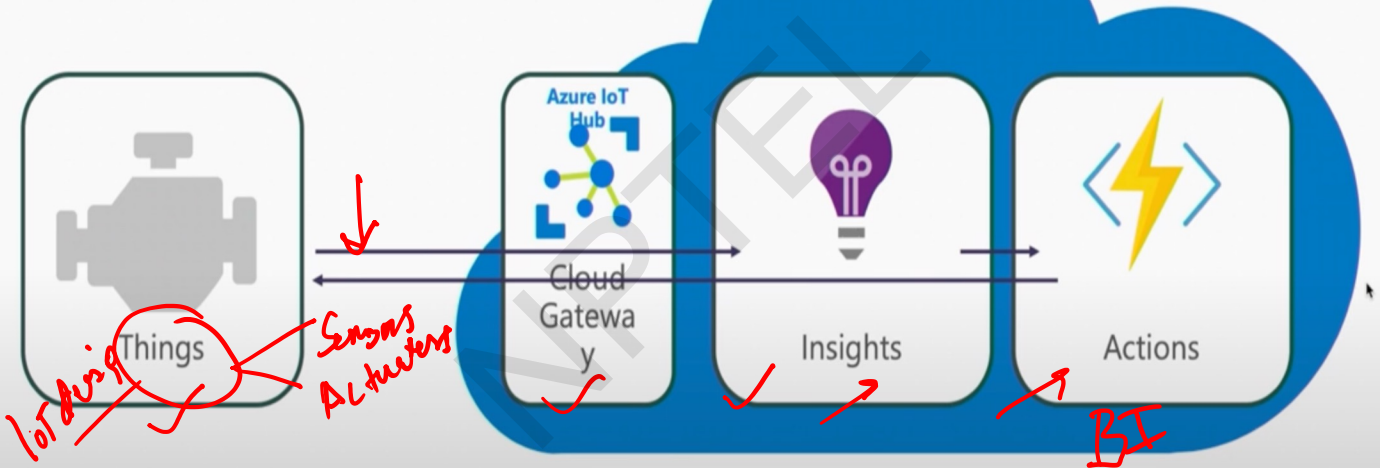
- Issues in traditional IoT platform
- How edge ML addresses the issues of IoT platform?
- Work flow of edge ML
- Advantages and applications of edge ML

Recapitulate: Internet of Things

Internet of Things (IoT) is a network of internet connected devices (IoT Devices) embedded in everyday objects enabling sending and receiving data such as settings and telemetry.



Recapitulate: Traditional IoT platform



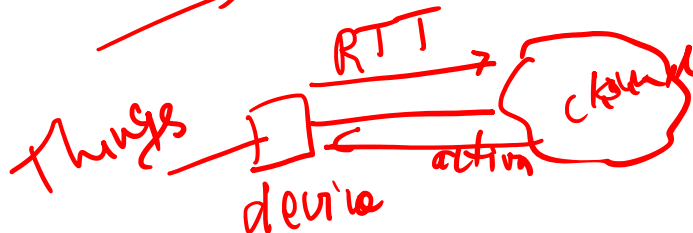
Once the edge device is connected to azure IoT hub service, different custom code modules are developed. One to capture incoming data and send that to the custom vision module and another one to manage, control and get the score out to display output of the model and the last one is a custom vision model which is used to provide the insight.

Recapitulate: Limitation of traditional IoT platform

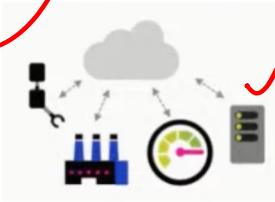
Poor internet connection, when the internet is down the system fails. For example, if a smart fire alarm system just detect fires when internet connection is up, then it fails in performing its task.

Data gravity, IoT devices create lots of data that demand more way to find the insights locally on a device than shipping all of the data to the cloud. For example, a smart doorbell, you don't want to stream video to the cloud 24-7 just to identify faces for the two minutes that someone is in front of your door, you would rather do that, locally on the smart doorbells.

Real time responses, as opposed to near real time responses that you cannot get by sending data to the cloud finding insights and then sending the actions back down.



ML on cloud



Remote monitoring and control

Merging remote data across multiple IoT devices

Near Infinite storage to train machine learning and other advanced ML models

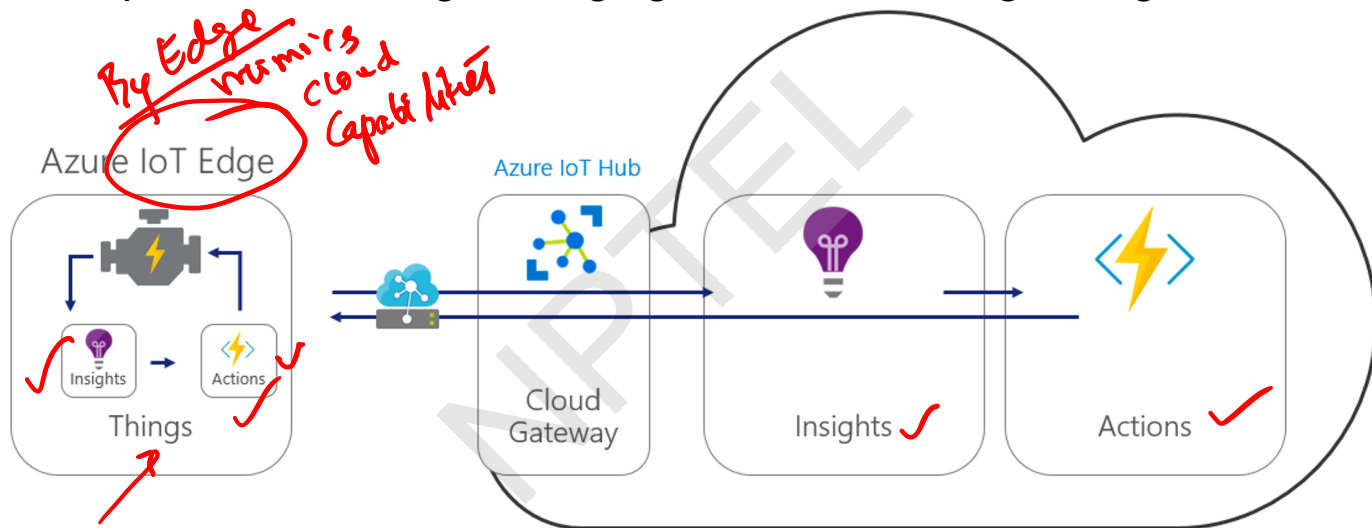
Vs ML on Edge



- Low latency tight control loops require near real-time response
- Pre-process data on premise
- Intelligence on edge
- Offline operations
- Data privacy and IP protection

IoT-Edge-Cloud

Recapitulate: IOT-Edge: Bringing machine learning to edge for IoT ✓



Once the IoT device fetches the workload description from cloud, then whenever the device receives his deployment manifest from the IoT hub service, it understands that it should go fetch those two containers i.e. action and things.

Enabling Intelligence at Edge layer for IOT

To manage the increasing amount of data that is generated by the devices, sensors, most of the business logic is now applied at the edge instead of the cloud to achieve low-latency and have faster response time for IOT devices using Machine learning at edge.

Edge layer is delivering three essential capabilities

1. local data processing,
2. filtered data transfer to the cloud and
3. faster decision-making

Enabling Intelligence at Edge layer for IOT

✓ Local data processing: ✓

- In order to deal with increasing amount of data generated by sensors, most of the business logic is now deployed at the edge layer instead of cloud to ensure low-latency and faster response time.
- Only a subset of the data generated by sensors is sent to the cloud after aggregating and filtering the data at the edge.

✓ Filtered data transfer to cloud:

- This Edge Computing approach significantly saves the bandwidth and cloud storage.

✓ Faster decision-making:

- AI has enabled new capabilities for edge computing. Since most of the decision-making is now taking advantage of artificial intelligence, the edge layer is becoming the perfect destination for deploying machine-learning models trained in the cloud.

Performance vs Cost trade-off in IOT-Edge Platforms

ML implementation on edge heavily depends on specialized processors that complement the CPU. There's no conventional CPU can increase the speed of training ML model.

To bridge the gap between the cloud and edge, innovations in chip designs offers purpose-built accelerator that speed up model inferencing significantly. Chip manufacturers such as Qualcomm, NVIDIA and ARM have launched specialized chips that speed up the execution of ML-enabled applications.

— Edge — GPU

These modern processors GPUs assist the CPU of the edge devices by taking over the complex mathematical calculations needed for running deep learning models, accelerate the inference process.

This result in faster prediction, detection and classification of data ingested to the edge layer.

The solutions like Microsoft Azure IoT Edge runtime and the Qualcomm Neural Processing SDK for ML makes it possible to take models trained in the cloud and run hardware-accelerated inference at the intelligent edge.

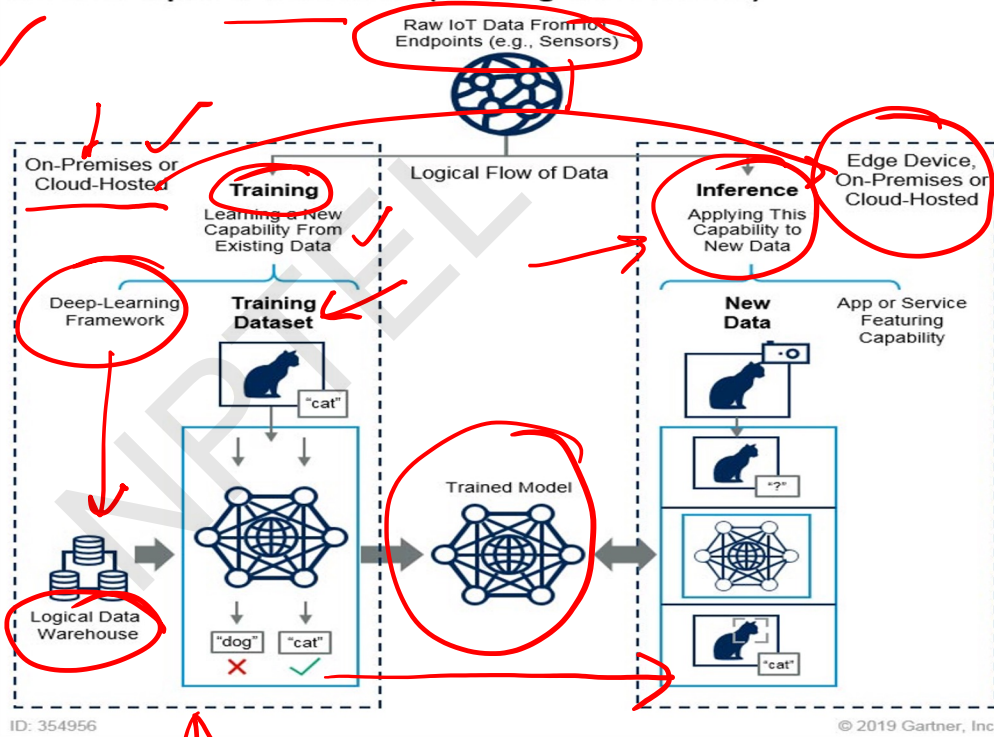
Following way to make Intelligent Edge Accelerations

- GPU are now available for edge layer
- AI models which are trained in cloud can do inference with help of accelerated GPU at edge

Training: Involves the use of a deep-learning framework (e.g., TensorFlow) and training dataset. IoT data provides a source of training data that data scientists and engineers can use to train machine learning models for a variety of use cases, from failure detection to consumer intelligence.

Inference: Inference refers to the process of using a trained machine learning algorithm to make a prediction. IoT data can be used as the input to a trained machine learning model, enabling predictions that can guide decision logic on the device, at the edge gateway or elsewhere in the IoT system.

IoT Data Input to ML Models (Training vs. Inference)

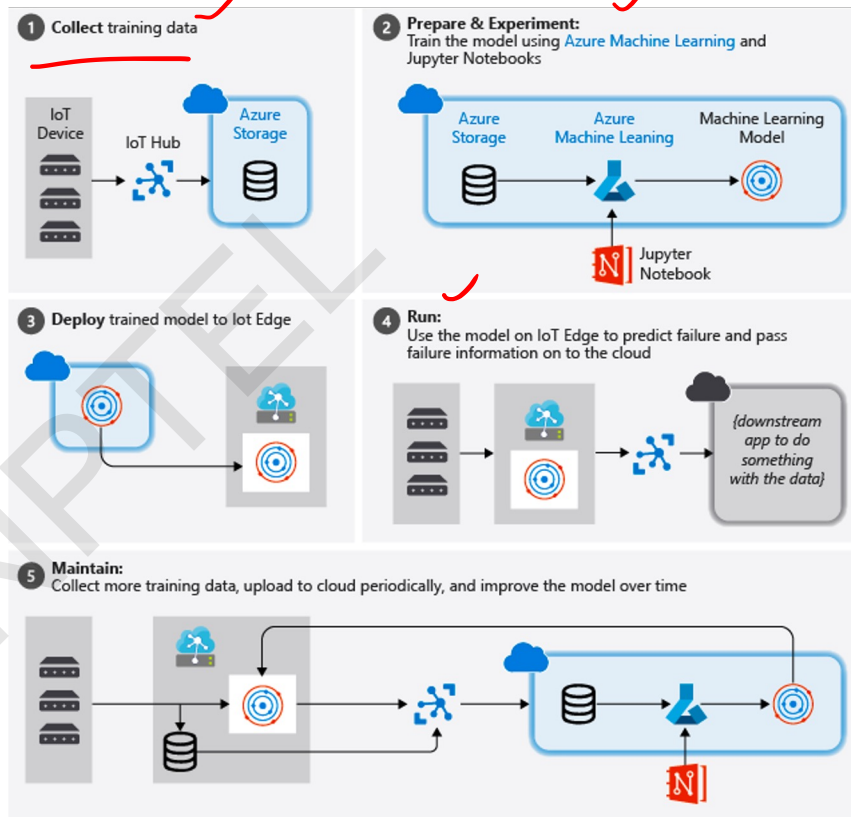


ML on edge IoT

While we may touch many aspects of a general machine learning workflow.

This lecture session is not intended as an in-depth introduction to machine learning.

We intend to illustrate the process of creating and using a viable model for IoT data processing.

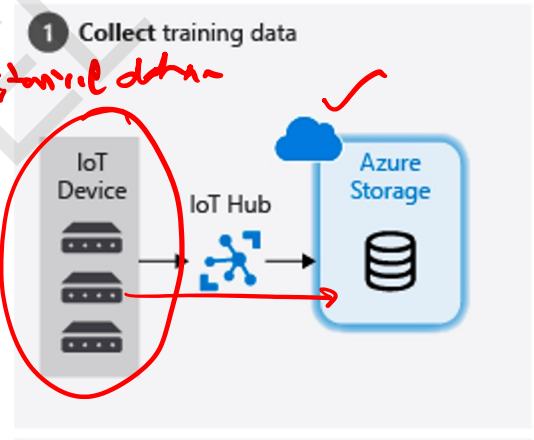


ML on edge IoT: Collect training data

The process begins by collecting training data.

In some cases, data has already been collected and is available in a database, or in form of data files.

In other cases, especially for IoT scenarios, the data needs to be collected from IoT devices and sensors and stored in the cloud.



ML on edge IoT: Prepare data & Experiment

In most cases, the raw data as collected from devices and sensors will require preparation for machine learning.

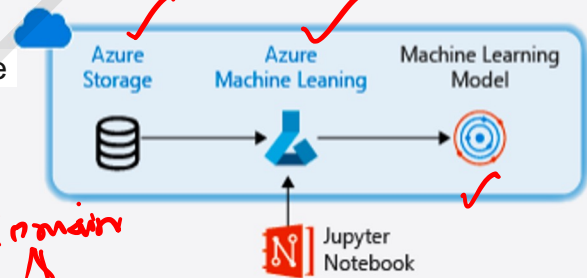
This step may involve data clean up, data reformatting, or preprocessing to inject additional information machine learning can key off.

Data preparation involves calculating explicit label for every data point in the sample based on the actual observations on the data.

This information allows the machine learning algorithm to find correlations between actual sensor data patterns and the expected results. This step is highly domain-specific.

2 Prepare & Experiment:

Train the model using [Azure Machine Learning](#) and Jupyter Notebooks



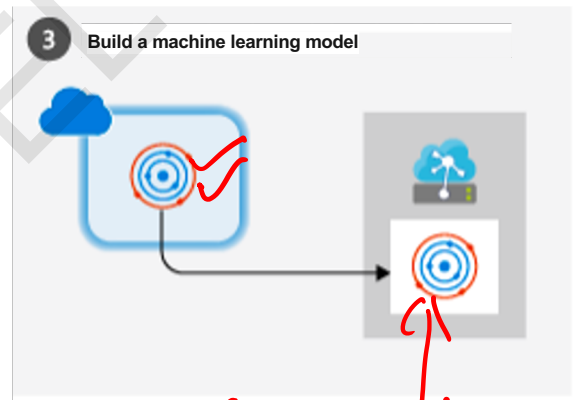
data scientists

ML on edge IoT: Build a machine learning model ✓

Based on the prepared data, we can now experiment with different machine learning algorithms and parameterizations to train models and compare the results to one another. ✓

In this case, for testing we compare the predicted outcome computed by the model with the real outcome observed for a IOT Application. ✓

In Azure Machine Learning, this can be done in the different iterations of models that is created in a model registry. ✓



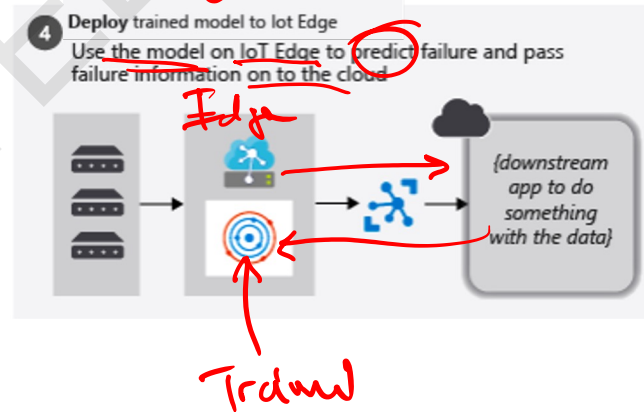
Trained machine learning model

ML on edge IoT: Deploy the machine learning model

Based on the prepared data, we can now experiment with different machine learning algorithms and parameterizations to train models and compare the results to one another.

In this case, for testing we compare the predicted outcome computed by the model with the real outcome observed for a IOT Application.

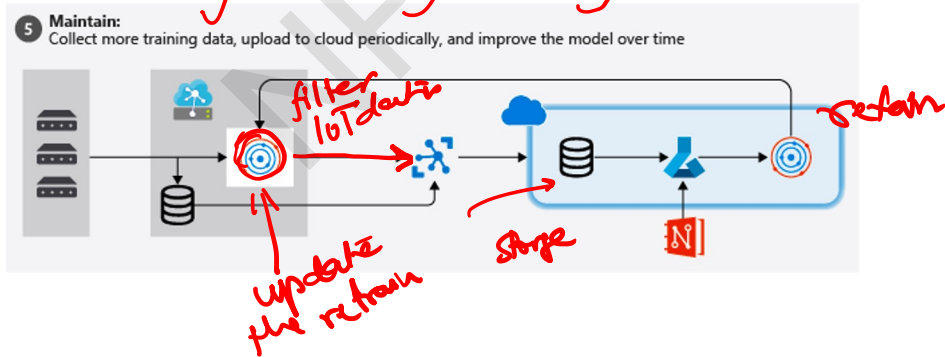
In Azure Machine Learning, this can be done in the different iterations of models that is created in a model registry.



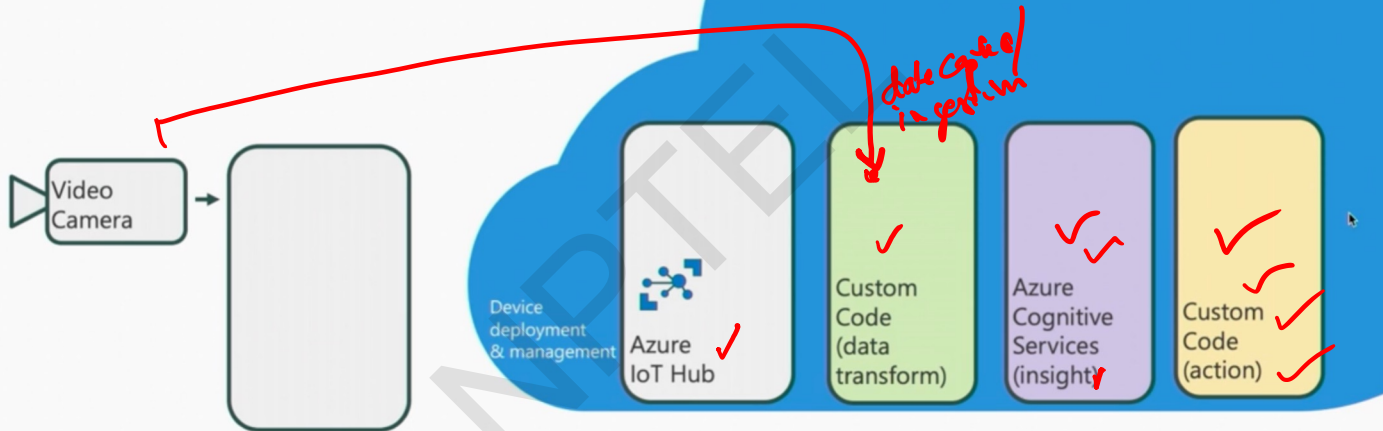
ML on edge IoT: Maintain and refine the model

Our work is not done once the model is deployed. In many cases, we want to continue collecting data and periodically upload that data to the cloud.

We can then use this data to **retrain** and **refine** our model, which we then can redeploy to IoT Edge.



Edge ML Platform (SaaS)



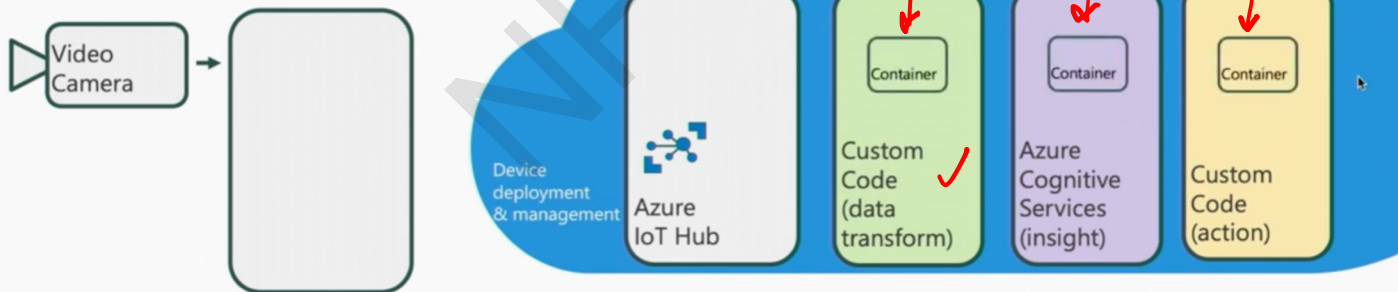
Once the edge device is connected to azure IoT hub service, two custom code modules are developed.

One to capture incoming data and send that to the custom vision module and another one to manage, control and get the score out to display output of the model and the last one is a custom vision model which is used to provide the insight.

Edge ML Platform: Insight-Container ✓

Step 1: Package the data transform, insight and action into containers. ✓

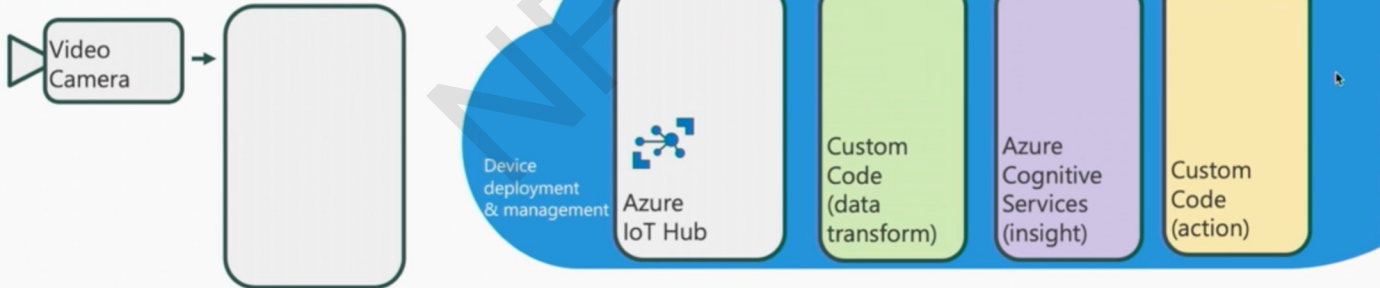
Now, write those three modules and package them as docker containers ✓



Edge ML Platform: Docker

Step 2: Put the containers to container registry.

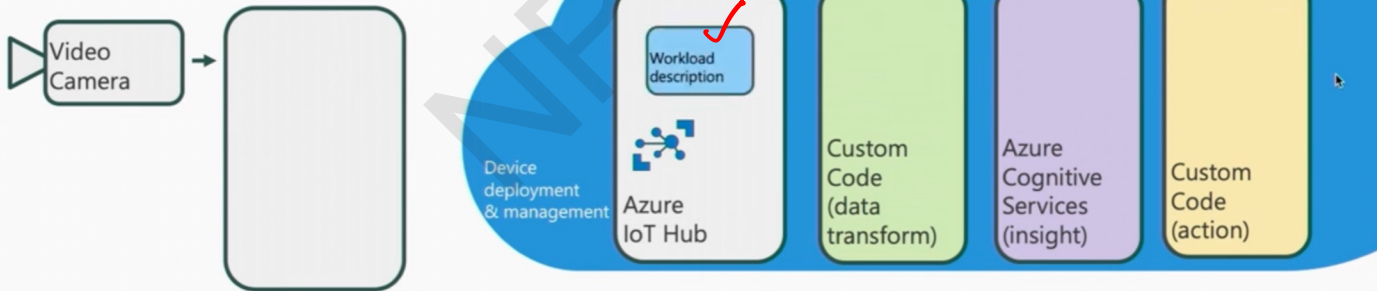
Push all those docker containers into container registry



Edge ML Platform: Cloud IOT-HUB

Step 3: Define a workload description in the cloud.

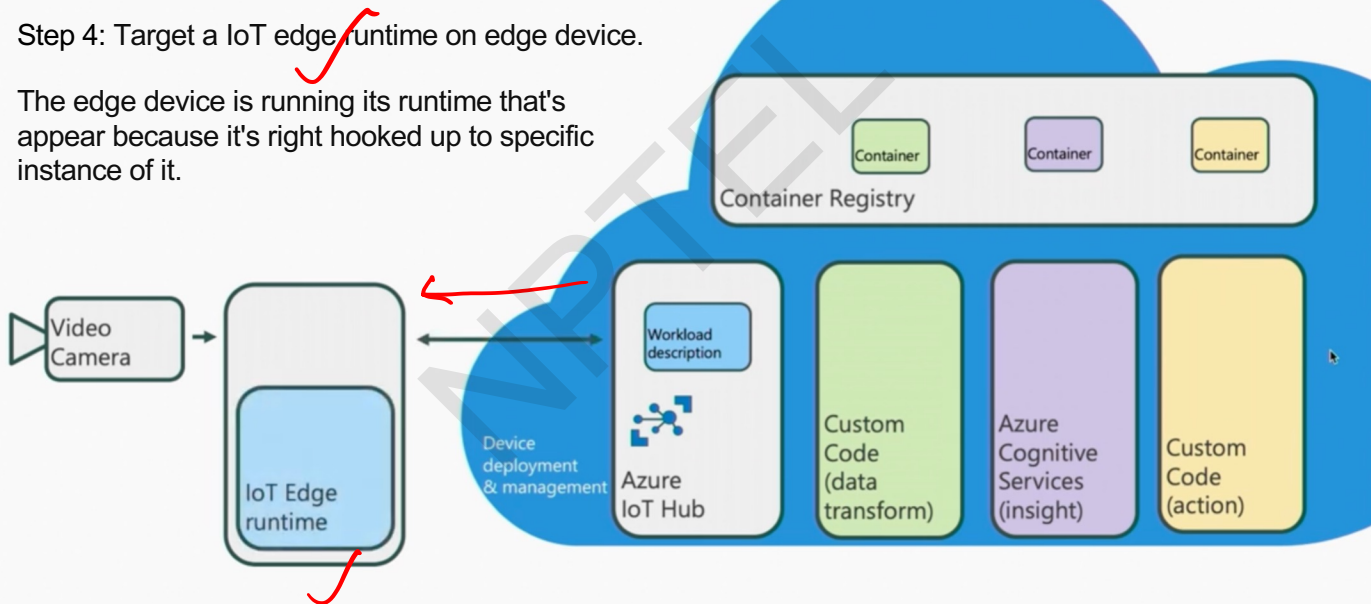
Then, write a deployment manifest which is also called as the workload description that deploy those three modules.



Edge ML Platform: Edge Run-time Manifestation

Step 4: Target a IoT edge runtime on edge device.

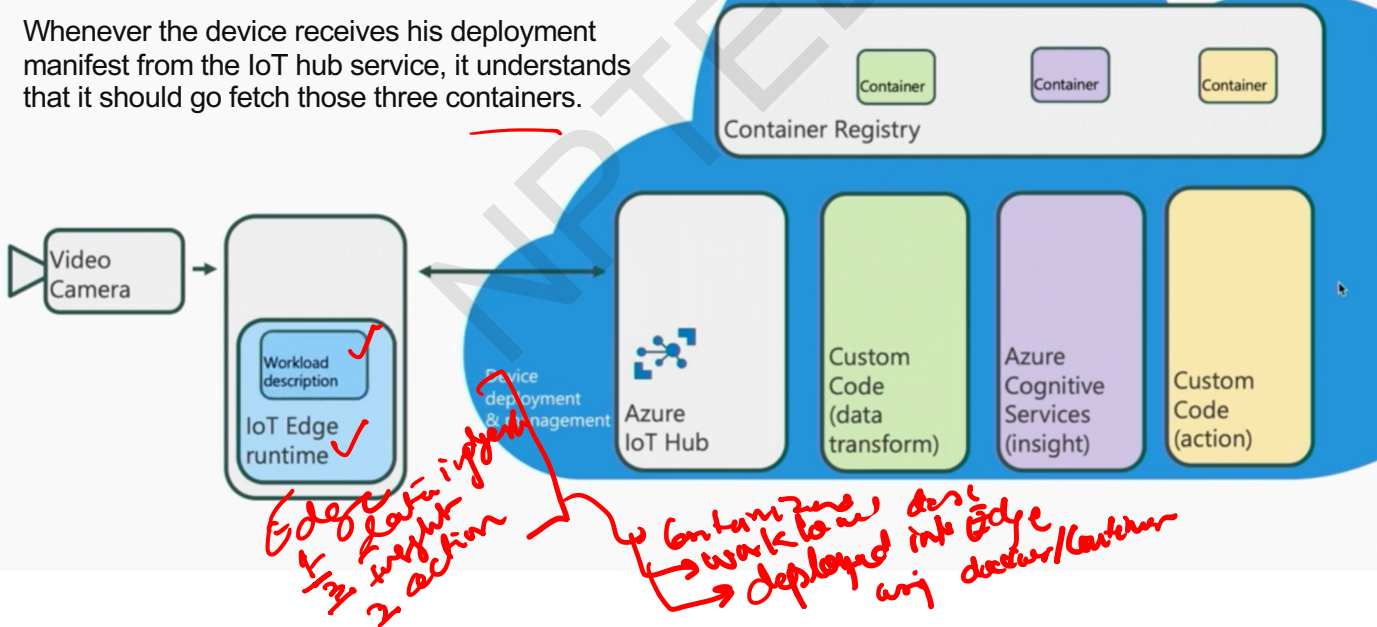
The edge device is running its runtime that's appear because it's right hooked up to specific instance of it.



Edge ML Platform: Migrating Workload

Step 5: Shift the workload description to the target IoT edge runtime on edge device.

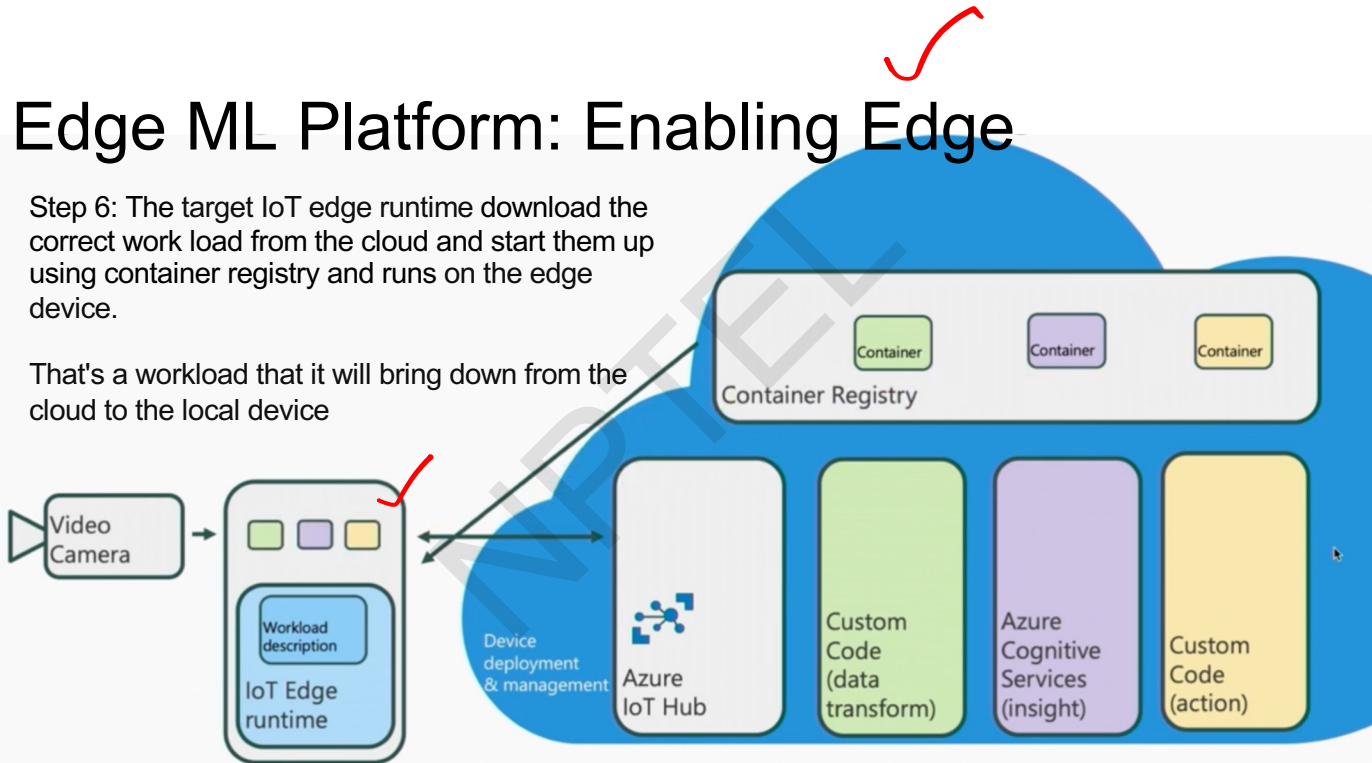
Whenever the device receives his deployment manifest from the IoT hub service, it understands that it should go fetch those three containers.



Edge ML Platform: Enabling Edge

Step 6: The target IoT edge runtime download the correct work load from the cloud and start them up using container registry and runs on the edge device.

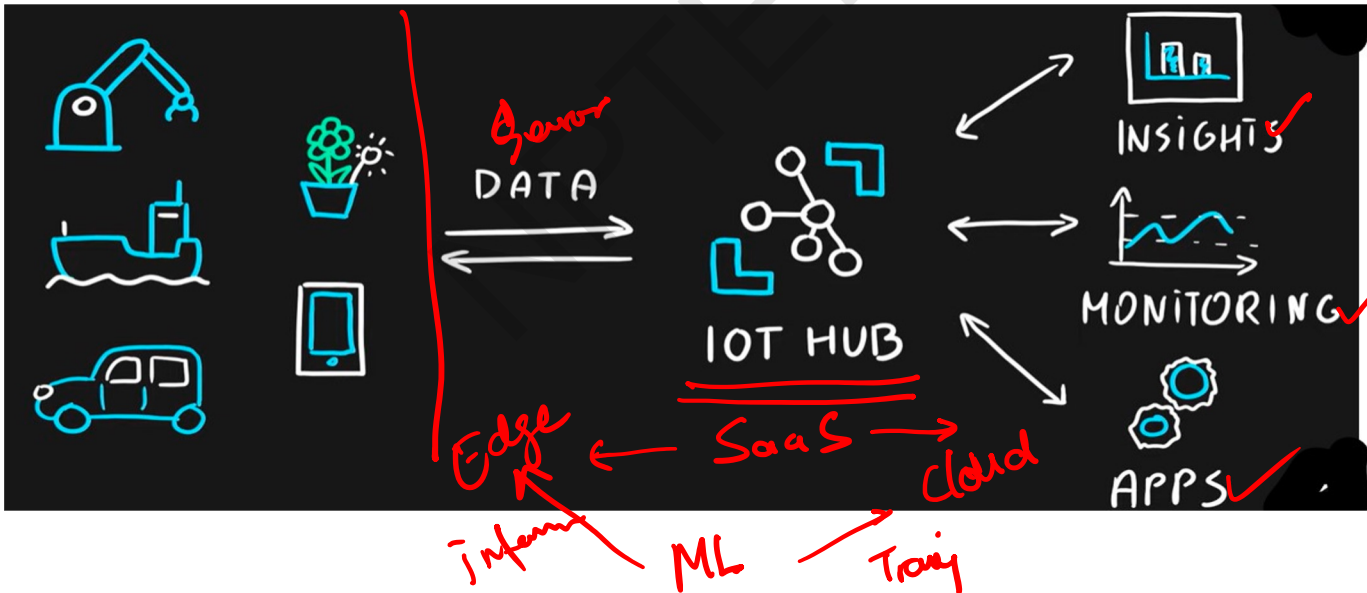
That's a workload that it will bring down from the cloud to the local device



Azure IoT Hub

Azure IoT Hub allows for bi-directional communication between the cloud and IoT devices.

Also, allows developers to take advantage of this information to provide insights, monitoring, and develop custom solutions for their IoT platform.



Azure IoT Hub: key characteristics



Manages service for bi-directional communication: it is a managed service for bi-directional communication between the cloud and iot devices.

Platform as a service (Paas): it's a platform as a service offering in azure for iot development.

Highly secure, scalable and reliable: it's a highly secure, scalable and reliable service for iot devices.

Integrates with lots of azure services: perfectly integrates with a lot of azure services.

Programmable SDK for popular languages: you do not need to learn any new language to take advantage of iot hub for their development purposes.

Multiple protocols: it support for multiple common standards on the market when it comes to communication protocols

MQTT, HTTP etc
↑
1

IoT-Edge: key characteristics

The **Camera Capture Module** handles scanning items using a camera. It then calls the Image Classification module to identify the item, a call is then made to the “Text to Speech” module to convert item label to speech, and the name of the item scanned is played on the attached speaker.

The **Image Classification Module** runs a Tensorflow machine learning model that has been trained with images of fruit. It handles classifying the scanned items.

The **Text to Speech Module** converts the name of the item scanned from text to speech using Azure Speech Services.

A USB Camera is used to capture images of items to be bought.

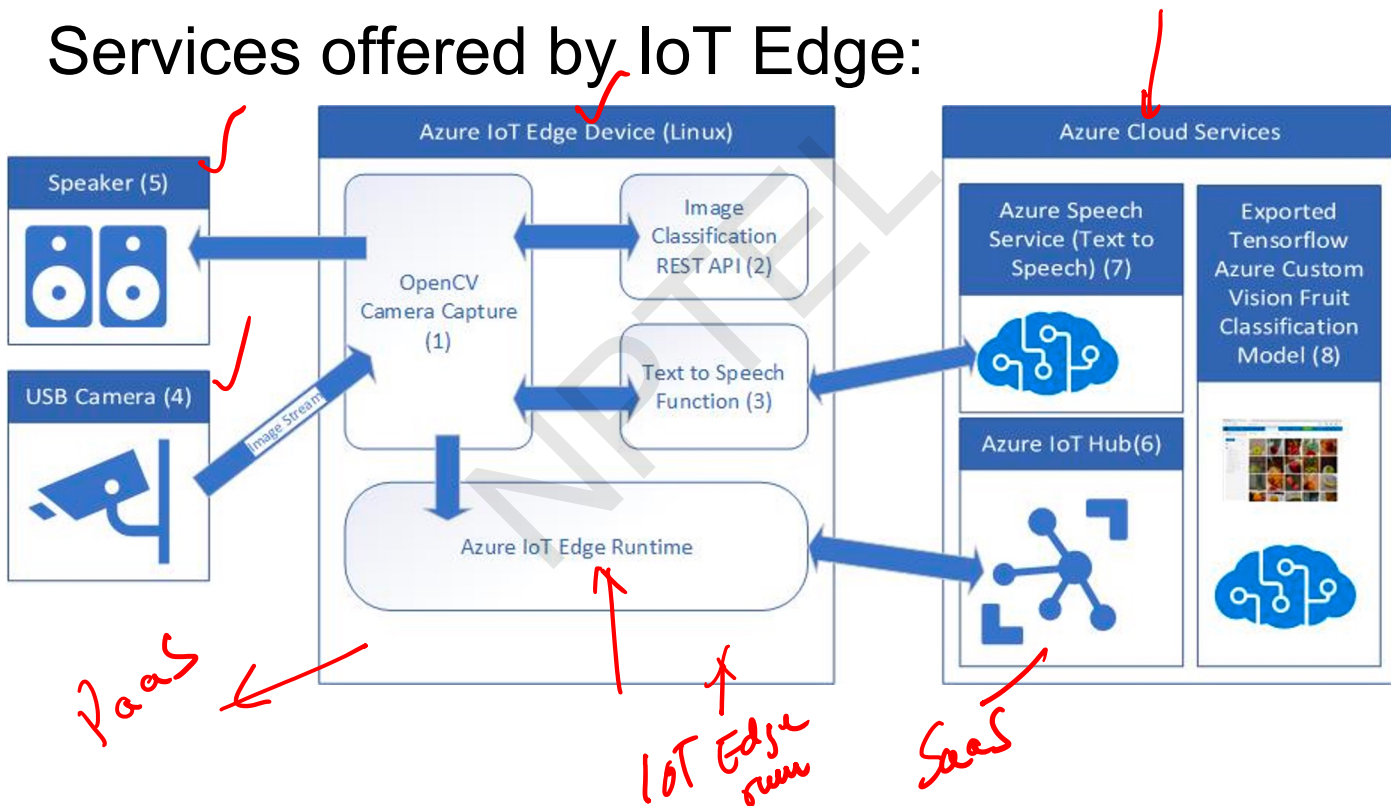
A Speaker for text to speech playback.

Azure IoT Hub (Free tier) is used for managing, deploying, and reporting Azure IoT Edge devices running the solution.

Azure Speech Services (free tier) is used to generate very natural speech telling the shopper what they have just scanned.

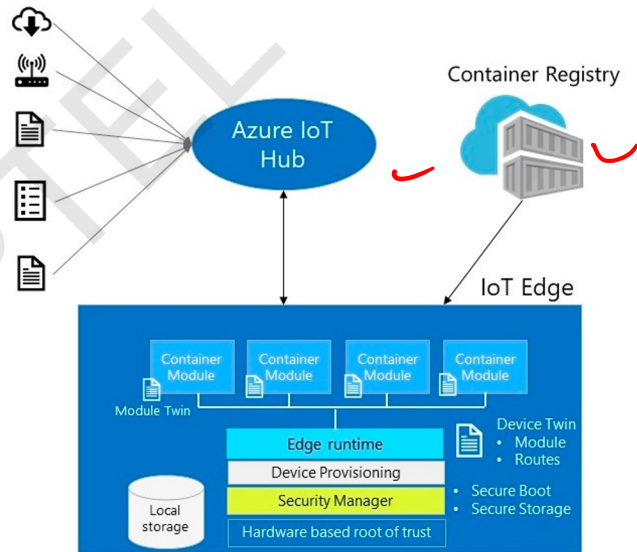
Azure Custom Vision service was used to build the fruit model used for image classification.

Services offered by IoT Edge:



Azure IoT Edge in Action

1. Develop your Container Workloads ✓
2. Upload Modules to a Container Registry
3. Provision IoT Edge Device Agents
4. Select Device/Deployment Set
5. Define Modules ✓
6. Define Message Routes ✓
7. Define Module Parameters ✓



Azure IoT edge: Functionalities

- Target workload at the correct type of device
 - Once the workload description sent down to the edge, the run time will download the correct workload from the cloud and start them up and running.
- Create workload which can include high value ML
 - This results in the custom code, machine learning model, and business logic all running locally independent of cloud connection and also all of those values of edge analytics.
- Run those workload locally, in disconnected manner
 - The runtime is smart enough to detect if the workload is trying to send messages to the cloud while it doesn't have internet connection, the runtime will catch those messages and sync them with the cloud once the internet is up.
- Monitor the health of the workloads
 - Azure IoT edge ensures that the work loads continue to run and report status sent back to the cloud. Reporting the status back to the cloud allows to understand if there is any issues issues in the deployment and take preventive actions.

Advantages of Edge ML

Reduced latency: Transfer of data back & forth from the cloud takes time. Edge ML reduces latency by processing data locally (at the device level).

Real-time analytics: Real-time analytics is a major advantage of Edge Computing. Edge ML brings high-performance computing capabilities to the edge, where sensors and IoT devices are located.

Higher speeds: Data is processed locally which significantly improves processing speed as compared to cloud computing

Reduced bandwidth requirement: Edge ML processes the data locally on the device itself, reducing the cost of internet bandwidth and cloud storage.

Improved data security: Edge ML systems perform the majority of data processing locally i.e. on the edge device itself. This greatly reduces the amount of data that is sent to the cloud and other external locations.

Advantages of Edge ML

Scalability: Edge ML typically processes large amounts of data. If you have to process video image data from many different sources simultaneously, transferring the data to a cloud service is not required.

Improved reliability: Higher levels of security combined with greater speed produce greater the reliability of Edge ML System.

Reduced cost: ML processing is working on the edge of the device so it is highly cost-cost efficient because only processed, data required or valuable data is sent to the cloud.

Reduced power: Edge ML processes data at the device level so it saves energy costs

Applications of Edge ML ✓

Manufacturing: rapid collection and analysis of data produced by edge-based devices and sensors.

Energy (Oil and Gas): real-time analytics with information processing in remote locations.

Industrial IoT: Inspection of devices/machines is done via ML algorithms instead of human beings performing manual inspections can save time & money. ✓

Autonomous Vehicles: fast data processing that could take milliseconds to perform which could reduce collision. ✓

Healthcare: to process all patient monitoring device data locally like glucose monitors, cardiac trackers, blood pressure sensors, etc. ✓

Smart Homes: data movement time can be reduced and also the sensitive information can be processed only on edge.

Self driving car ↑

Lecture Summary

- Limitations of IoT platform
- How edge ML addresses the issues of IoT platform?
- Work flow of edge ML
- Advantages and applications of edge ML



THANK YOU!

ML-based Image Classifier at IoT-Edge



Dr. Rajiv Misra

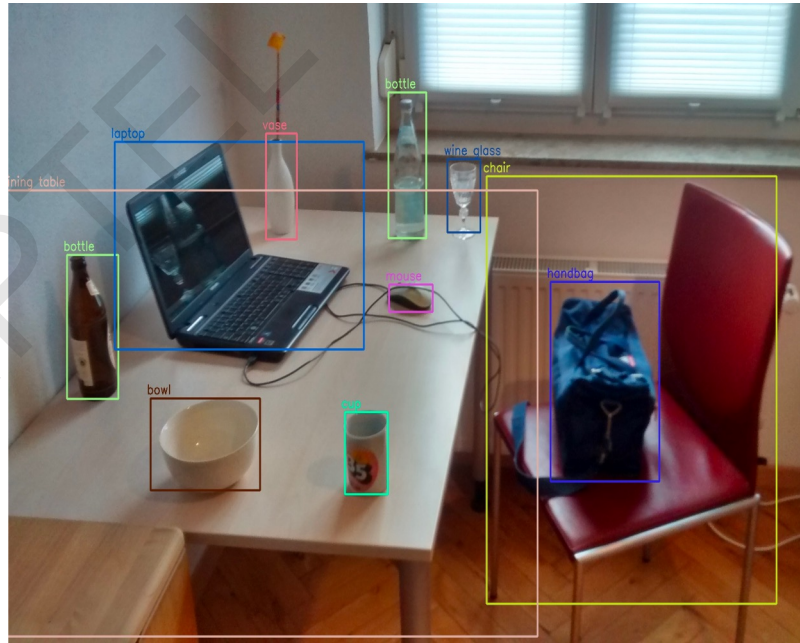
Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

After Completion of this lecture you will knowing the following:

- Basics of computer vision in ML
- Different techniques of computer vision like image classification, detection, segmentation, etc
- Object detection models like RCNN, Fast RCNN, Faster RCNN, SDD, YOLO
- Azure compute vision as SaaS

Computer Vision: Introduction

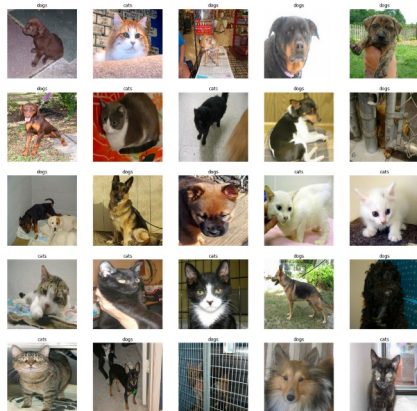
Computer vision is a sub branch of machine learning which deals with giving computers the ability to **see** and **interpret** and **extract information** from images and videos, videos can be seen as collection of images.



How Computer Vision Works?

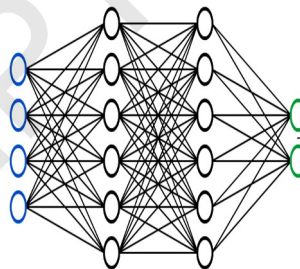
To train a computer vision model, you essentially feed some thousands of images of cats and it's going to do some complex mathematics and feature extraction etc in the background.

Based on that it learns some key understanding or properties that define cats.

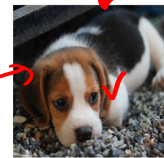


Thousands of cats images

Input



Feature extraction



Dog:	96%
Cat:	29%
Duck:	2%
Bird:	0%



Dog:	36%
Cat:	94%
Duck:	2%
Bird:	1%

Output

Trained model

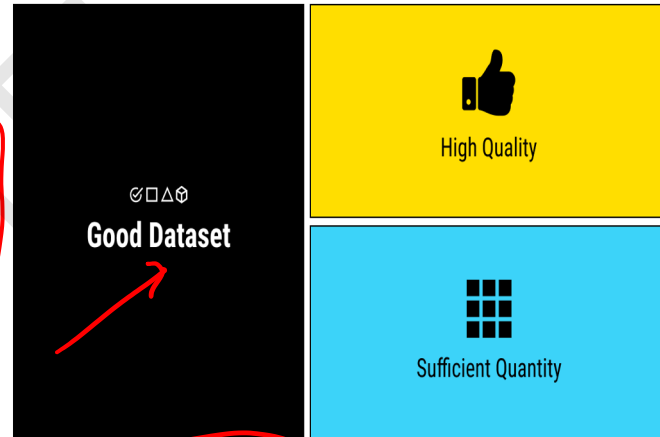
Computer Vision: Data Analytics View

With every machine learning model, the model is not only the important part.

The fundamental fact that's going to determine how good your model is the data you feed it.

Today, this is another point that we want to focus on is the fact that your model is only as good as your data.

So one of the key things that we're going to focus on today's lecture is how to make sure that our data is good when we're building computer vision models.

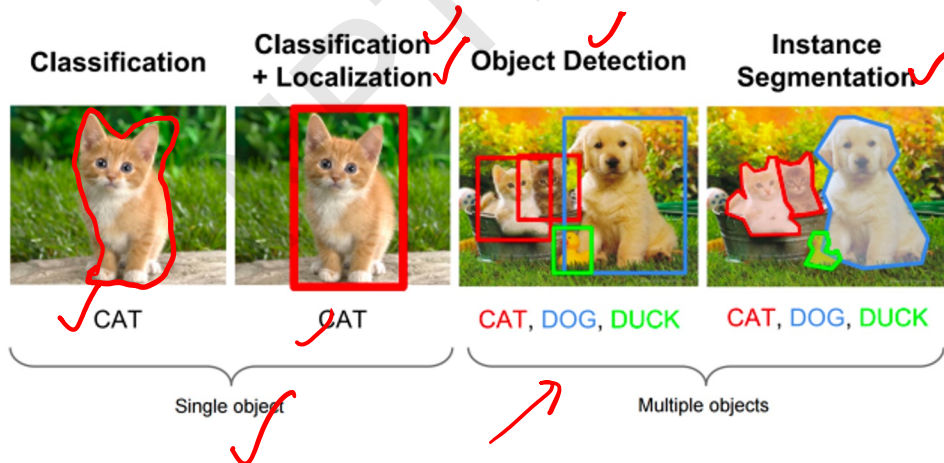


*Data Scientist
IoT*

*Data Science
Prepare the data-set*

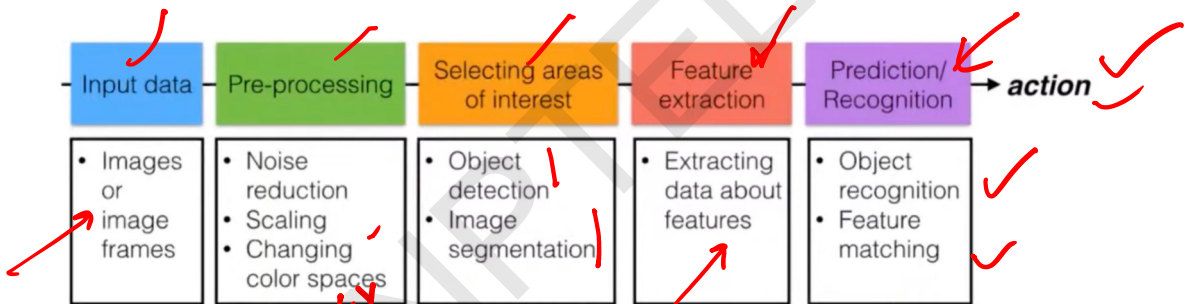
Computer Vision: Techniques

Computer vision deals with all the problems related to images and videos. There's a lot of techniques, fundamentals or problems that can be tackled with computer vision. A few of them includes image classification, image detection, image segmentation, pattern detection and object localisation. So object detection and image classification are the two things that we're going to talk about today and we're going to go into some of the predictive model build for an object detection model.



Computer Vision: Architecture

A typical end-to-end pipeline of computer vision architecture is shown below.



Input data, the images that you want to train your model on, but those images might be coming from a lot of different sources.

IoT -

Computer Vision Architecture: Pre-processing

The second and important step is pre-processing of input data.

Machine learning depends on standardization, means you need to pre-process input images to make sure that they're all of the same size.

There might be some noise in the images, all of that needs to be dealt with before the image fed into the model.

If it's not done correctly, the model might learn noise or other features that are not good features or it might learn from those that actually mean that your model is going to be fundamentally flawed.

Therefore, pre-processing is essentially it's very important that need to be.

Pre-processing

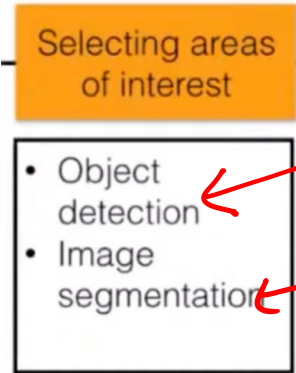
- Noise reduction
- Scaling
- Changing color spaces

Computer Vision Architecture: Data labeling

The third step is labeling your data.

For an object detection problem, you have images with different objects, let's say if you have an image with a cat and a dog, you would need to label that specific part of the image where there's a cat and a dog.

This label or tags to that specific area where there's a dog or cat, so this is essentially labeling and this needs to be done as well.



ML
→ Supervised Learning

— Data Scientist —

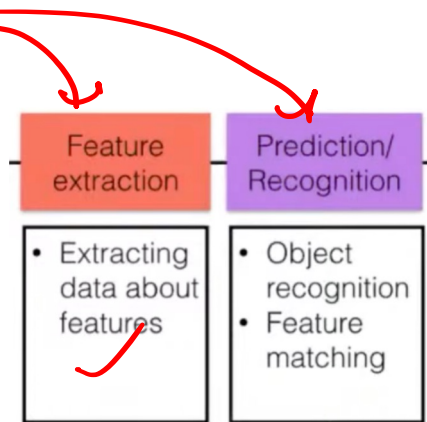


Computer Vision Architecture: Feature extraction and prediction

Then feature extraction and prediction part performed by a machine learning model.

These are part of the model training, the model learns about what features are present, it extracts the features.

And if features are relevant from the images, then those features are learned along with the patterns and later the model uses it to build a sort of rules for itself and these rules are used to predict the output.



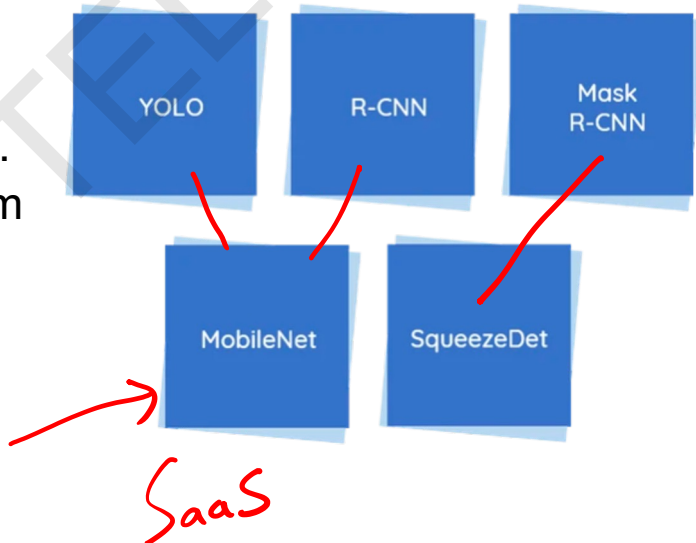
Model Training
- Supervised learning

Inference

Computer Vision: Object Detection Models

The field of object detection is not as new as it may seem. In fact, object detection has evolved over the past 20 years. Popular deep learning algorithm that achieved remarkable results in this domain are:

- RCNN ✓
- Fast RCNN
- Faster RCNN
- YOLO
- SSD (Single Shot Detector)

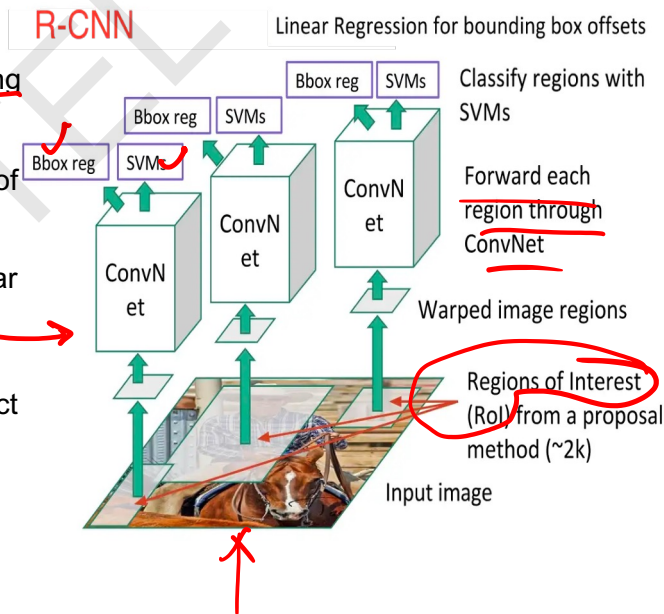


Object Detection Model: RCNN ✓

R-CNN, or Region-based Convolutional Neural Network, consisted of 3 simple steps:

1. Scan the input image for possible objects using an algorithm called Selective Search, generating ~2000 region proposals
2. Run a convolutional neural net (CNN) on top of each of these region proposals
3. Take the output of each CNN and feed it into a) an SVM to classify the region and b) a linear regressor to tighten the bounding box of the object, if such an object exists.

In other words, we first propose regions, then extract features, and then classify those regions based on their features. In essence, we have turned object detection into an image classification problem. R-CNN was very intuitive, but very slow.

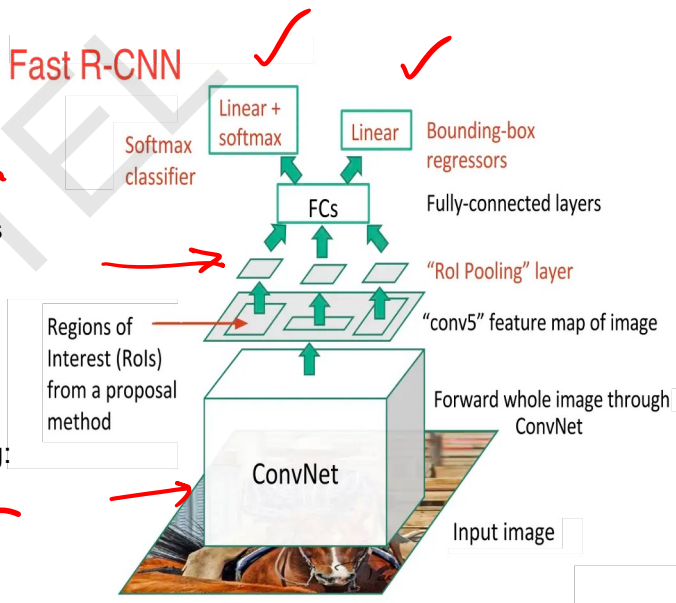


Object Detection Model: Fast RCNN

As we can see from the image, we are now generating region proposals based on the last feature map of the network, not from the original image itself. As a result, we can train just one CNN for the entire image.

In addition, instead of training many different SVM's to classify each object class, there is a single softmax layer that outputs the class probabilities directly. Now we only have one neural net to train, as opposed to one neural net and many SVM's.

Fast R-CNN performed much better in terms of speed. There was just one big bottleneck remaining: the selective search algorithm for generating region proposals.



Object Detection Model: Faster RCNN

RPN

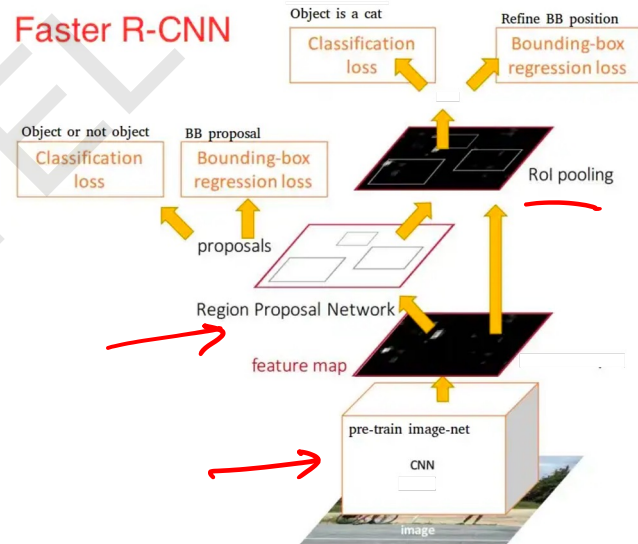
The main insight of Faster R-CNN was to replace the slow selective search algorithm with a fast neural net. Specifically, it introduced the region proposal network (RPN).

Here's how the RPN worked:

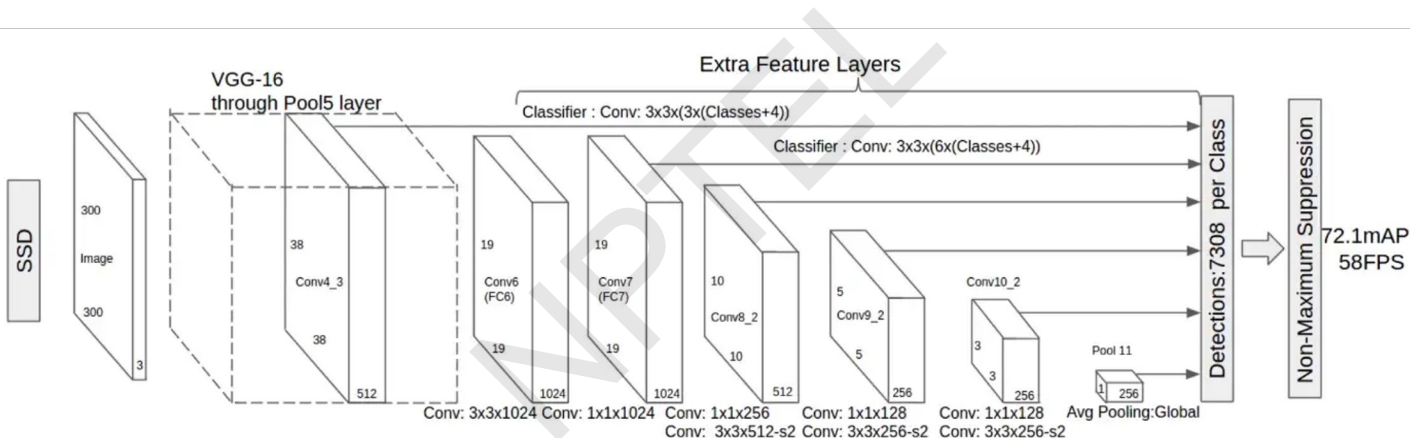
- At the last layer of an initial CNN, a 3x3 sliding window moves across the feature map and maps it to a lower dimension (e.g. 256-d)
- For each sliding-window location, it generates multiple possible regions based on k fixed-ratio anchor boxes (default bounding boxes)
- Each region proposal consists of a) an “objectness” score for that region and b) 4 coordinates representing the bounding box of the region

In other words, we look at each location in our last feature map and consider k different boxes centered around it: a tall box, a wide box, a large box, etc. For each of those boxes, we output whether or not we think it contains an object, and what the coordinates for that box are.

Faster R-CNN



Object Detection Model: SSD



Object Detection Model: SSD

SSD stands for Single-Shot Detector. Like R-FCN, it provides enormous speed gains over Faster R-CNN, but does so in a markedly different manner.

Our first two models performed region proposals and region classifications in two separate steps. First, they used a region proposal network to generate regions of interest; next, they used either fully-connected layers or position-sensitive convolutional layers to classify those regions. SSD does the two in a “single shot,” simultaneously predicting the bounding box and the class as it processes the image.

Given an input image and a set of ground truth labels, SSD does the following:

- Pass the image through a series of convolutional layers, yielding several sets of feature maps at different scales (e.g. 10x10, then 6x6, then 3x3, etc.)
- For each location in each of these feature maps, use a 3x3 convolutional filter to evaluate a small set of default bounding boxes. These default bounding boxes are essentially equivalent to Faster R-CNN's anchor boxes.
- For each box, simultaneously predict a) the bounding box offset and b) the class probabilities
- During training, match the ground truth box with these predicted boxes based on IoU. The best predicted box will be labeled a “positive,” along with all other boxes that have an IoU with the truth > 0.5 .

Intersection over Union

Object Detection Model: YOLOv3

You Only Look Once or more popularly known as YOLO is one of the fastest real-time object detection algorithm (45 frames per second) as compared to the R-CNN family (R-CNN, Fast R-CNN, Faster R-CNN, etc.)

45 FPS

The R-CNN family of algorithms uses regions to localise the objects in images which means the model is applied to multiple regions and high scoring regions of the image are considered as object detected.

Instead of selecting some regions, YOLO approaches the object detection problem in a completely different way.



It forwards the entire image to predict bounding boxes and their probabilities only once through the neural network.

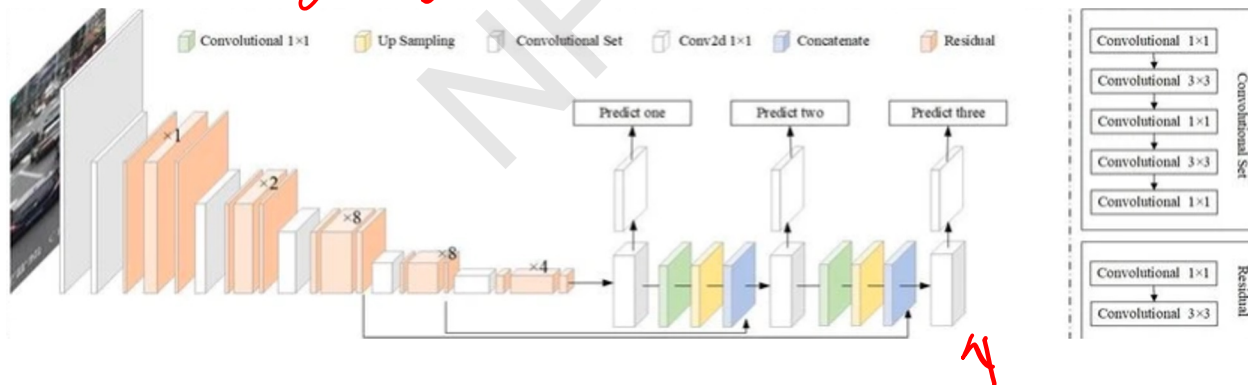
The authors have also improved the network by making it bigger and taking it towards residual networks by adding shortcut connections.

Object Detection Model: YOLOv3

First, it divides the image into a 13×13 grid of cells. The size of these 169 cells varies depending on the input size. For a 416×416 input size, the cell size was 32×32 . Each cell is responsible for predicting the number of boxes in image.

For each bounding box, the network also predicts the confidence that the bounding box actually encloses an object, and the probability of the enclosed object being a particular class.

Most of these bounding boxes are eliminated because their confidence is low or because they are enclosing the same object as another bounding box with a very high confidence score. This technique is called non-maximum suppression.

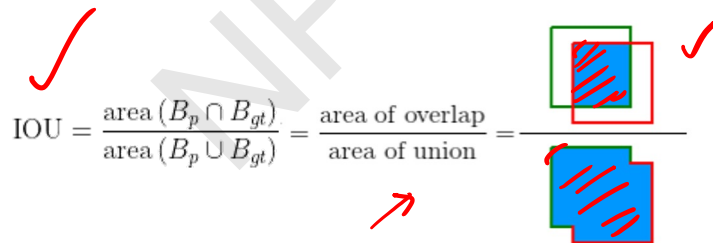


Object Detection Models: Performance Metric

An overview of the most popular metrics used to compare performance of different deep learning models:

Intersection Over Union (IOU)

Intersection Over Union (IOU) is a measure based on Jaccard Index that evaluates the overlap between two bounding boxes. IOU is given by the overlapping area between the predicted bounding box and the ground truth bounding box divided by the area of union between them:

$$\text{IOU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of overlap}}{\text{area of predicted box} + \text{area of ground truth box} - \text{area of overlap}}$$


Object Detection Models: Performance Metric

Precision:

Precision is the ability of a model to identify only the relevant objects. It is the percentage of correct positive predictions and is given by:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}}$$

Recall:

Recall is the ability of a model to find all the relevant cases (all ground truth bounding boxes). It is the percentage of true positive detected among all relevant ground truths and is given by:

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}$$

True Positive (TP): A correct detection. Detection with $\text{IOU} \geq \text{threshold}$

False Positive (FP): A wrong detection. Detection with $\text{IOU} < \text{threshold}$

False Negative (FN): A ground truth not detected

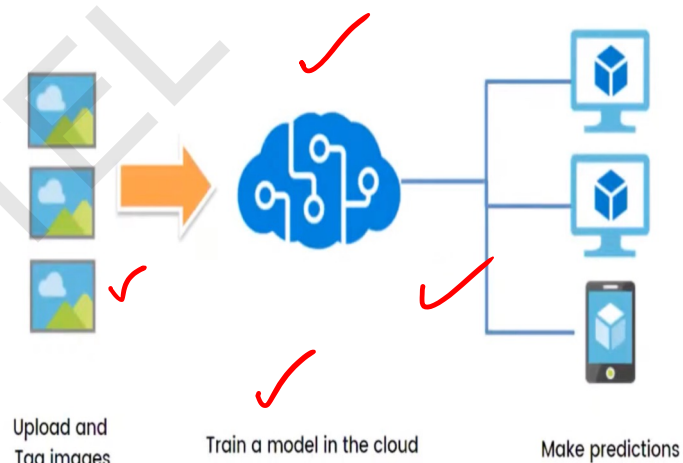
Computer Vision: SaaS Architecture

A computer vision architecture can easily be taken up by a cloud service that is running a computer vision model in the cloud.

A SaaS is a software as a service that is offered by all providers like azure, amazon aws, google cloud. All of them offers some variation of these for computer vision service.

In that architecture, all you need to do is you need to have your images and upload them and tag them. Tagging is vital because you as the domain expert know what information is present in the images.

Once you've uploaded them in the cloud, the model training and everything that is completely dependent on the cloud provider and fully managed by the cloud service provider.



Data-Science

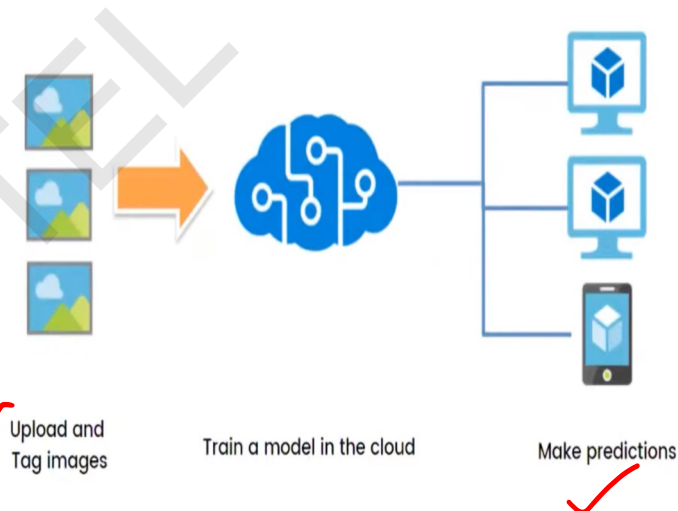
Computer Vision: SaaS Architecture

It's extremely easy to scale up your dataset and allow you to download the models that you've built that can later be used offline.

Once you've trained and download the model, simply use the rest API to query that model and get the predictions which is extremely reliable and simple.

The SaaS architecture provided by different cloud provider offers similar services.

There might be fundamental differences in the ui or how you are uploading images or the api or how you're calling the services but under the hood they're doing the same thing.



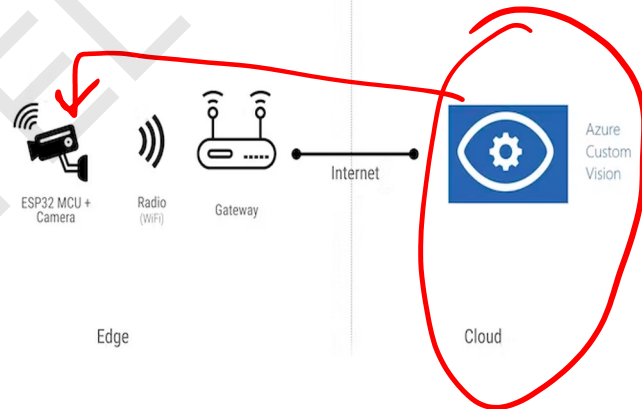
SaaS: Azure Custom Vision

Azure Custom Vision is a cloud service used to build and deploy computer vision models.

Custom Vision uses a pretty interesting neural network technique called **transfer learning**, which applies knowledge gained from solving one problem to a different, but related situation. This can substantially decrease the time needed for creating the models.

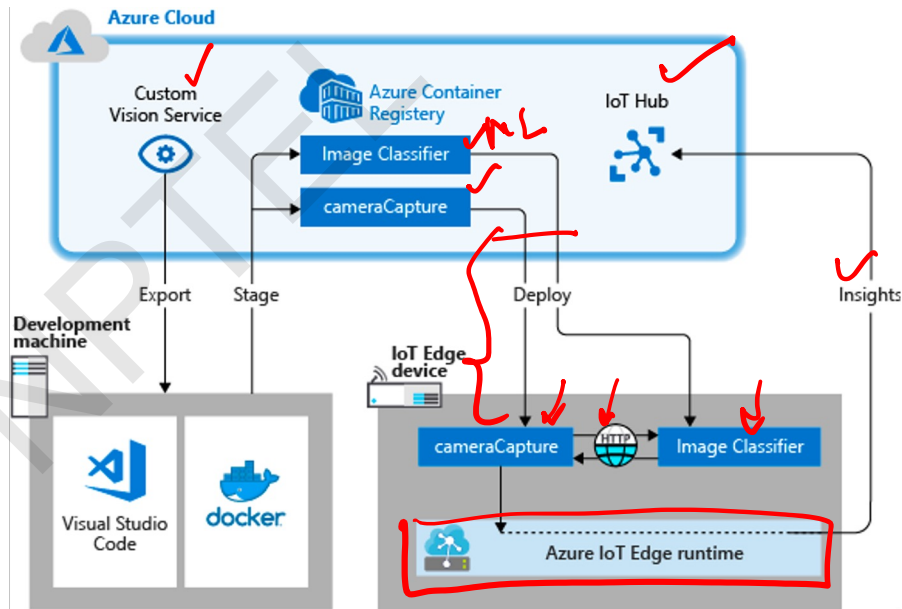
Features provided by Azure Custom Vision service:

- Train a computer vision model by simply uploading and labeling few images.
- Build image classifier model using code-free and code-first approach.
- Deploy the model in the cloud on-premise, or on edge devices.



Azure Custom Vision on an IoT Edge device

- Build an image classifier with Custom Vision.
- Develop an IoT Edge module that queries the Custom Vision web server on device.
- Send the results of the image classifier to IoT Hub.



Use Case: Creating an image recognition solution with Azure IoT Edge and Azure Cognitive Services

Although there are lots of applications for image recognition but we had chosen this application which is a solution for vision impaired people scanning fruit and vegetables at a self-service checkout.

Required Components

Raspberry Pi 3B or better, USB Camera, and a Speaker.

Note, the solution will run on a Raspberry Pi 3A+, it has enough processing power, but the device is limited to 512MB RAM. A Raspberry Pi 3B+ has 1GB of RAM and is faster than the older 3B model. Azure IoT Edge requires an ARM32v7 or better processor. It will not run on the ARM32v6 processor found in the Raspberry Pi Zero.

Desktop Linux - such as Ubuntu 18.0

This solution requires USB camera pass through into a Docker container as well as Azure IoT Edge support. So for now, that is Linux.

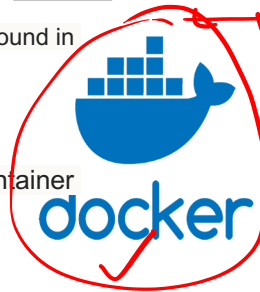
Desktop IoT USB camera

IoT Edge

Cloud



Raspberry Pi



+



Guide for installing Raspberry Pi

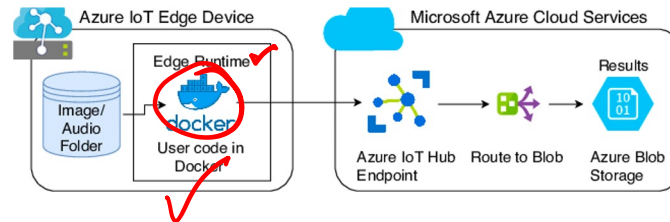
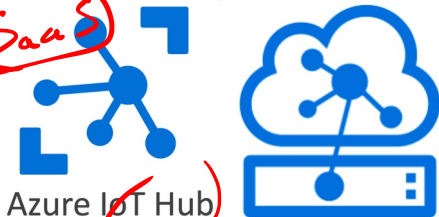
Set up **Raspbian Stretch Lite** on Raspberry Pi: Be sure to configure the correct Country Code in your `wpa_supplicant.conf` file.

Azure subscription: If you don't already have an Azure account then sign up for a free Azure account. If you are a student then sign up for an Azure for Students account, no credit card required.

Create an **Azure IoT Hub**, and an **Azure IoT Edge device**: Install Azure IoT Edge runtime on Raspberry Pi and download the deployment configuration file that describes the Azure IoT Edge Modules and Routes for this solution. Open the `deployment.arm32v7.json` link and save the `deployment.arm32v7.json` in a known location on your computer.

Install **Azure CLI** and **Azure CLI command line tools**: With CLI open a command line console/terminal and change directory to the location where you saved the `deployment.arm32v7.json` file.

Deploy edge to device: The modules will now start to deploy to your Raspberry Pi, the Raspberry Pi green activity LED will flicker until the deployment completes. Approximately 1.5 GB of Docker modules will be downloaded and decompressed on the Raspberry Pi. This is a one off operation.



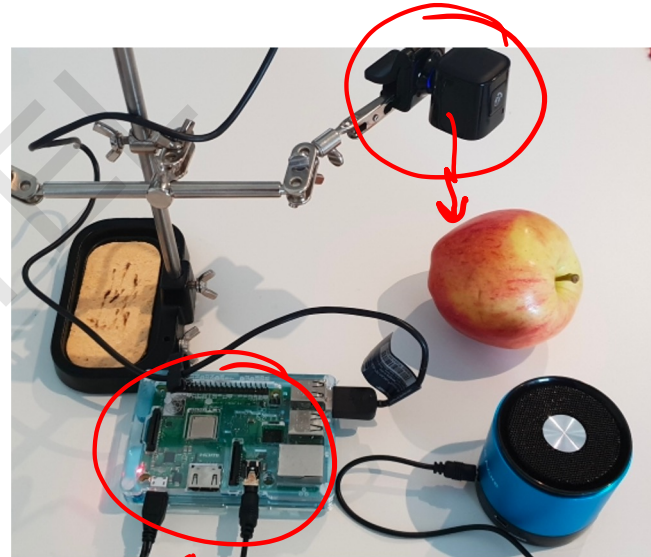
Considerations and constraints for the solution

The solution should scale from a Raspberry Pi (running Raspbian Linux) on ARM32v7, to my desktop development environment, to an industrial capable IoT Edge device such as those found in the Certified IoT Edge Catalog.

The solution needs camera input, uses a USB Webcam for image capture as it was supported across all target devices.

The camera capture module needed Docker USB device pass-through (not supported by Docker on Windows) so that plus targeting Raspberry Pi meant that need to target Azure IoT Edge on Linux.

To mirror the devices plus targeting, it requires Docker support for the USB webcam, so develop the solution on Ubuntu 18.04 developer desktop.



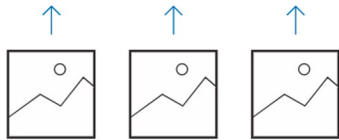
IoT Runtime
→ IoT Edge

Create Classification model using Azure Custom Vision

The **Azure Custom Vision** service is a simple way to create an image classification machine learning model without having to be a data science or machine learning expert.

You simply upload multiple collections of labelled images. For example, you could upload a collection of banana images and label them as 'banana'.

It is important to have a good variety of labelled images so be sure to improve your classifier.



Upload Images

Bring your own labeled images, or use Custom Vision to quickly add tags to any unlabeled images.

data source



Train

Use your labeled images to teach Custom Vision the concepts you care about.

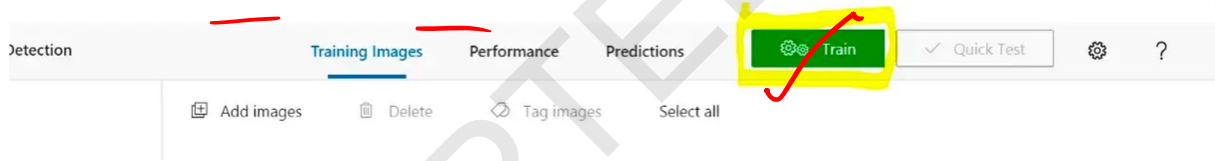


Evaluate

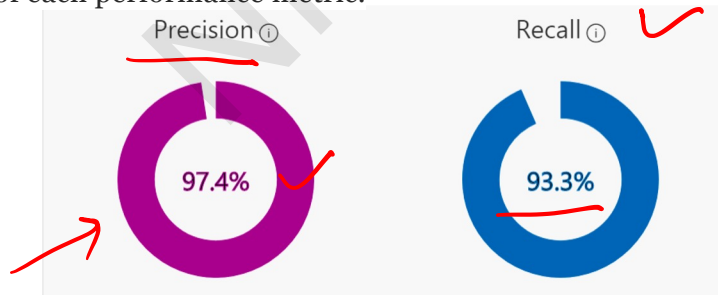
Use simple REST API calls to quickly tag images with your new custom computer vision model.

Create Custom Vision Classification model

1. Create a project in custom vision service mentioning the project type, classification type and domains.
2. Gather initial data (images) and separate them in different folders.
3. Once data is uploaded, train your model by clicking “Train” button on the navigation bar.



1. When the training is ended, the performance metrics will be shown. Click on the “i” bubble to see the meaning of each performance metric.

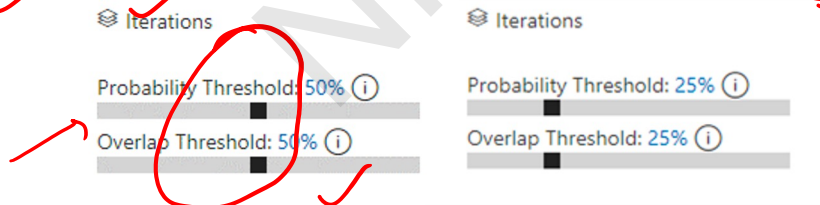


Create Custom Vision Classification model

5. Custom Vision offers fluent prediction thresholds adjustment to improve model performance. In our case we prefer higher Recall over high Precision. It is important not to lower those thresholds too much as the model performance will suffer significantly. E.g., having low probability threshold will lead to increased number of false positives. If the model is supposed to be deployed in a production setting, we can't be stopping the production line for every false positive detection produced by the model.

For the problem that we are working with right now we decided to set our KPIs as follows:


- The main metric to optimize for is mAP – it cannot be any lower than 85%
- The Recall and Precision are equally important, and both should stay above 80%







Export Custom Vision Classification model

Step 1: From the Performance tab of your Custom Vision project click Export.



The screenshot displays the Azure Custom Vision web interface. At the top, there are three tabs: 'Training Images', 'Performance' (which is selected and highlighted with a red box), and 'Predictions'. To the right of these tabs is a green 'Train' button. Below the tabs, there is a horizontal menu with several options: 'Prediction URL', 'Make default' (with a checkmark), 'Delete' (with a trash icon), and 'Export' (with a download icon and a red box around it). The main content area shows 'Iteration 8' and the text 'Finished training on 9/19/2018, 6:04:03 PM using General (compact) domain'. Below this, it states 'Classification type: Multiclass (Single tag per image)'. At the bottom, there are two gauge charts: 'Precision' (with a purple arc) and 'Recall' (with a blue arc). A large, light gray 'DRAFT' watermark is visible across the center of the image.



Training Images **Performance** Predictions 

 Prediction URL  Make default  Delete  **Export**

Iteration 8

Finished training on 9/19/2018, 6:04:03 PM using **General (compact)** domain
Classification type: **Multiclass (Single tag per image)**

Precision  Recall 

Export Custom Vision Classification model

Step 2: Select Dockerfile from the list of available options



Dockerfile

Azure IoT Edge, Azure Functions,
AzureML

Export Custom Vision Classification model

Step 3: Then select the Linux version of the Dockerfile.

Choose your platform



DF

Dockerfile

Linux

Download



Step 4: Download the docker file and unzip and you have a ready-made Docker solution with a Python Flask REST API. This was how the Azure IoT Edge Image Classification module is created in this solution.

Installing the solution

Step 1: Clone the repository for creating an image recognition solution with Azure IoT Edge and Azure Cognitive Services.

Step 2: Install the Azure IoT Edge runtime on your Linux desktop or device (eg Raspberry Pi).

Step 3: Install the following software development tools.

- Visual Studio Code

- Plus, the following Visual Studio Code Extensions

 - Azure IoT Edge

 - JSON Tools useful for changing the “Create Options” for a module.

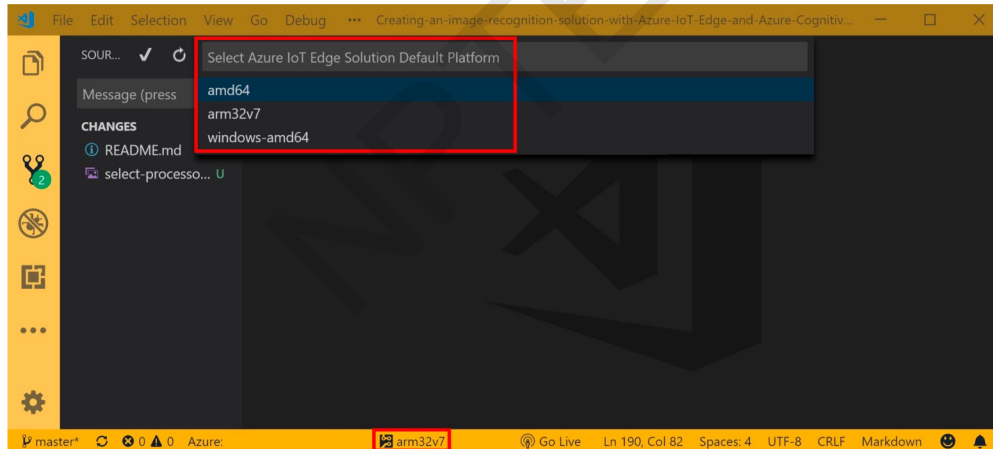
- Docker Community Edition on your development machine

Step 4: With Visual Studio Code, open the IoT Edge solution you cloned to your developer desktop.

Building the Solution

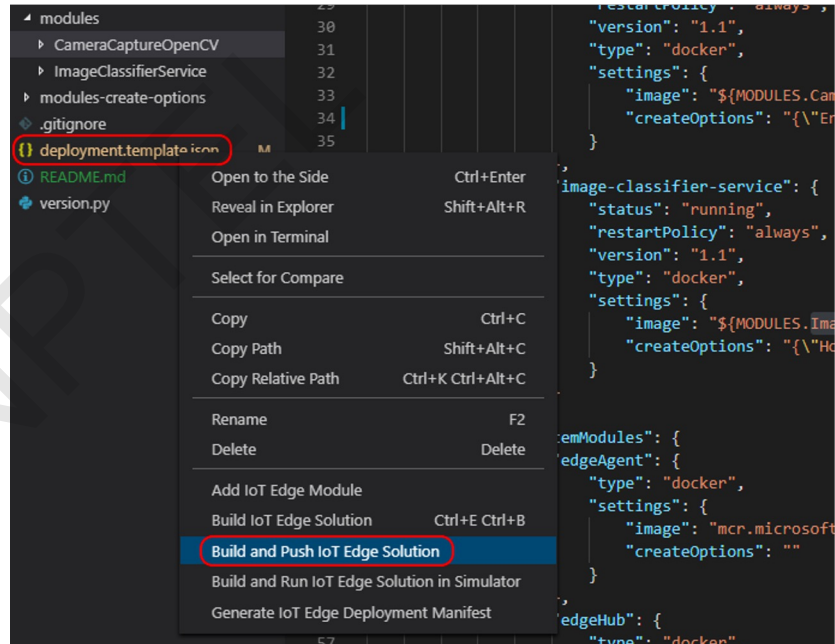
Step 1: Pushing the image to a local Docker repository with specifying the localhost.

Step 2: Confirm processor architecture using the Visual Studio Code



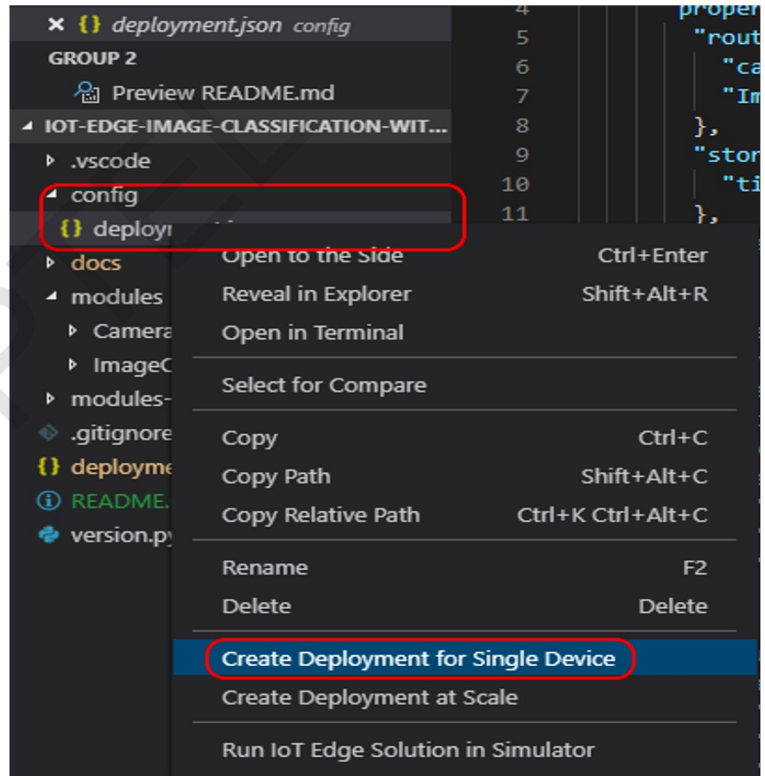
Building the Solution

Step 3: Build and Push the solution to Docker by right mouse clicking the deployment.template.json file and select "Build and Push IoT Edge Solution".



Deploying the Solution

When the Docker Build and Push process has completed select the Azure IoT Hub device you want to deploy the solution to. Right mouse click the deployment.json file found in the config folder and select the target device from the drop-down list.



Monitoring the Solution on the IoT Edge Device

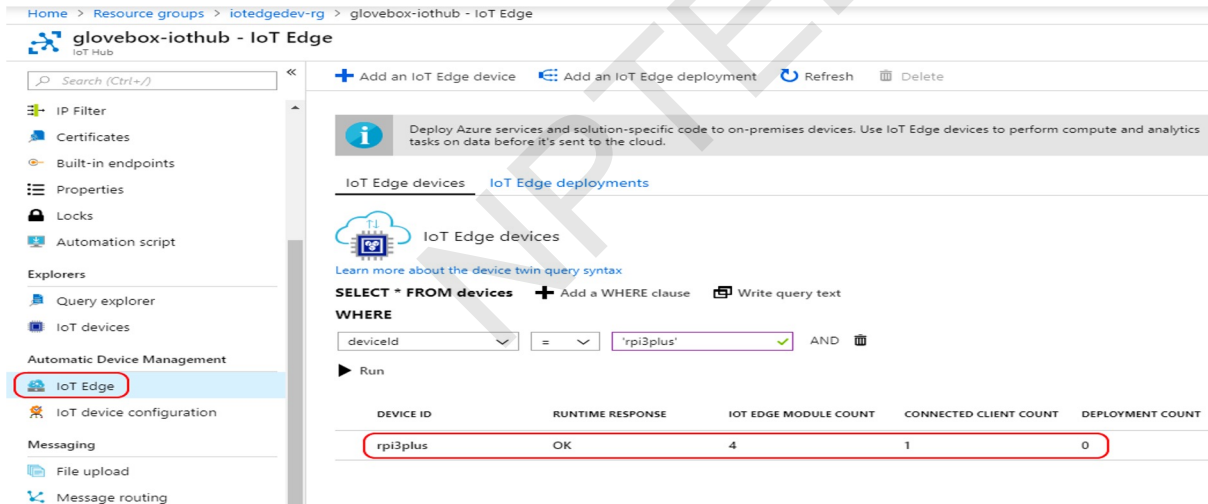
Once the solution has been deployed you can monitor it on the IoT Edge device itself using the *iotedge list* command.

```
File Edit View Search Terminal Help
Every 2.0s: iotedge list rpi3plus: Mon Oct 22 17:21:19 2018
```

NAME	STATUS	DESCRIPTION	CONFIG
image-classifier-service	running	Up 2 minutes	glovebox/image-classifier-service:0.1.9578-arm32v7
edgeHub	running	Up 2 minutes	mcr.microsoft.com/azureiotedge-hub:1.0.2
camera-capture	running	Up 2 minutes	glovebox/camera-capture-opencv:0.1.1990-arm32v7
edgeAgent	running	Up 2 minutes	mcr.microsoft.com/azureiotedge-agent:1.0.2

Monitoring the Solution on the IoT Edge Device

You can also monitor the state of the Azure IoT Edge module from the Azure IoT Hub blade on the Azure Portal.



The screenshot shows the Azure IoT Hub blade for 'glovebox-iothub - IoT Edge'. The left sidebar contains navigation options: IP Filter, Certificates, Built-in endpoints, Properties, Locks, Automation script, Explorers (Query explorer, IoT devices), Automatic Device Management (IoT Edge, IoT device configuration), Messaging (File upload, Message routing), and a search bar. The main content area displays 'IoT Edge devices' with a query editor and a table of device status.

Deploy Azure services and solution-specific code to on-premises devices. Use IoT Edge devices to perform compute and analytics tasks on data before it's sent to the cloud.

IoT Edge devices

Learn more about the device twin query syntax

SELECT * FROM devices + Add a WHERE clause Write query text

WHERE

deviceid = 'rpi3plus' AND

Run

DEVICE ID	RUNTIME RESPONSE	IOT EDGE MODULE COUNT	CONNECTED CLIENT COUNT	DEPLOYMENT COUNT
rpi3plus	OK	4	1	0

Monitoring the Solution on the IoT Edge Device

Click on the device from the Azure IoT Edge blade to view more details about the modules running on the device.

Home > Resource groups > iotedgedev-rg > glovebox-iothub - IoT Edge > Device details

Device details

rpi3plus

Save Set modules Manage child devices (Preview) Device twin Regenerate keys Refresh

Connection string (primary key) ⓘ
HostName=glovebox-iothub.azure-devices.net;DeviceId=rpi3plus;SharedAccessKey=[redacted]-3e0l6VNITC[redacted];QCTFppkhjUK4=[redacted]

Connection string (secondary key) ⓘ
HostName=glovebox-iothub.azure-devices.net;DeviceId=rpi3plus;SharedAccessKey=[redacted]3a2F1G+2R[redacted];Re69B6/vsM=[redacted]

Connect this device to an IoT hub ⓘ

Edge runtime response ⓘ
N/A

Modules IoT Edge hub connections Deployments

Verify that your modules are included in the deployment, and whether your modules have been reported by the device. Click Set modules to change the modules that appear. Each device can host a maximum of 20 modules.

NAME	TYPE	SPECIFIED IN DEPLOYMENT	REPORTED BY DEVICE	RUNTIME STATUS	EXIT CODE
SedgeAgent	IoT Edge System module	✓ Yes	✓ Yes	running	-
SedgeHub	IoT Edge System module	✓ Yes	✓ Yes	running	-
image-classifier-service	IoT Edge Custom module	✓ Yes	✓ Yes	running	--
camera-capture	IoT Edge Custom module	✓ Yes	✓ Yes	running	--

Lecture Summary

- Computer Vision
 - Introduction ✓
 - How it works?
 - Techniques
 - Architecture
- Objection detection models
 - RCNN
 - Fast-RCNN
 - Faster-RCNN ✓
 - SSD
 - YOLO
- Azure compute vision as SaaS ✓
- Usecase



THANK YOU!

Introduction to Docker Containers and Kubernetes



Dr. Rajiv Misra

Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

After completion of this lecture you will be knowing the following:

- Introduction to Kubernetes
 - Containers
 - Orchestration
- Concepts of Dockers
- Power of kubernetes to deploy software on edge devices

Introduction to Kubernetes

Kubernetes is the greek word for helmsman or captain of a ship.

Kubernetes also known as k-8 was built by Google based on their experience running containers in production

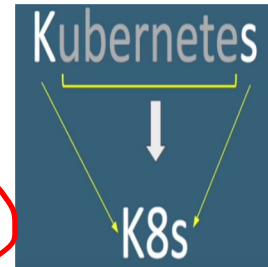
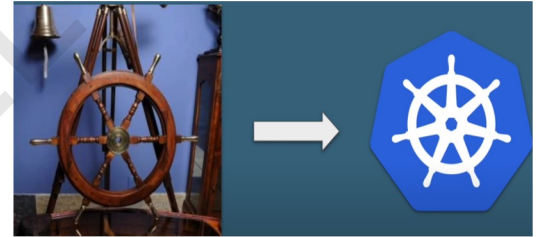
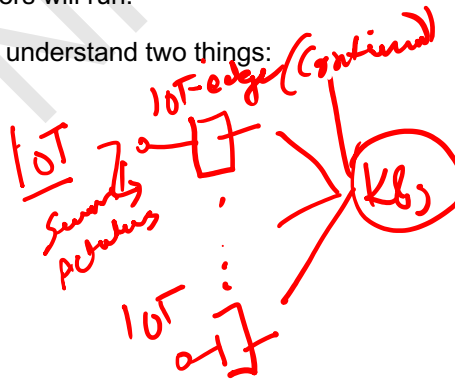
It is now an open source project and is one of the best and most popular container orchestration technologies out there. ✓

As applications grow to span multiple containers deployed across multiple servers, operating them becomes more complex.

To manage this complexity, Kubernetes provides an open source API that controls how and where those containers will run.

To understand kubernetes first we need to understand two things:

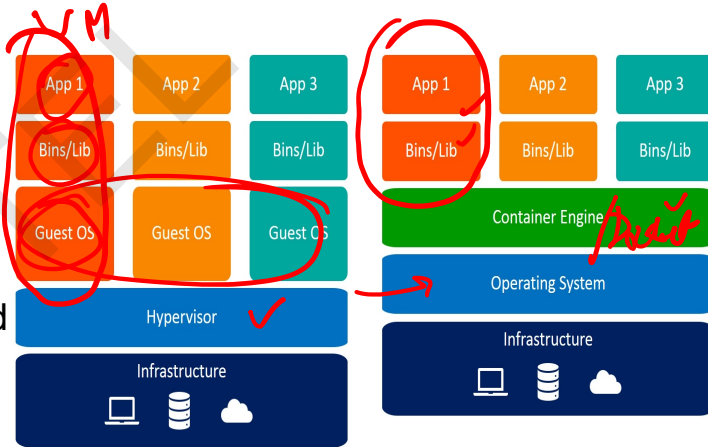
- Container and ✓
- Orchestration ✓



Introduction to Kubernetes

✓ **Containers** are isolated environments, have their own processes, services, networking interfaces, mounts similar to virtual machines except the fact that they all share the same operating system kernel.

✓ **Orchestration** consists of a set of tools and scripts that can help host containers in a production environment. An orchestration consists of multiple container hosts that can host containers, if one fails the application is ~~still~~ accessible through the others.



Virtual Machines

Containers

light weight

Introduction to Kubernetes

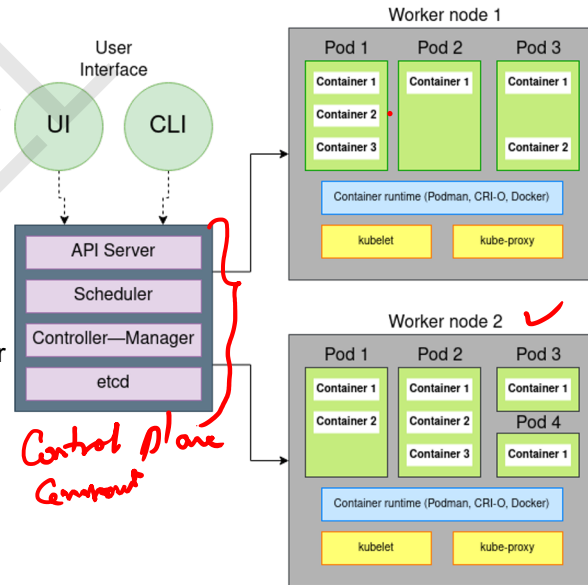
Kubernetes consists of one computer that gets designated as the control plane, and lots of other computers that get designated as worker nodes. Each of these has a complex but robust stack making orchestration possible,

Kubernetes **orchestrates** clusters of virtual machines and schedules containers to run on those virtual machines based on their available compute resources and the resource requirements of each container.

Kubernetes also automatically manages service discovery, incorporates load balancing, tracks resource allocation and scales based on compute utilisation. And, it checks the health of individual resources and enables apps to self-heal by automatically restarting or replicating containers.

Now get familiar with each of the kubernetes components:

- Control plane component
- Worker node component



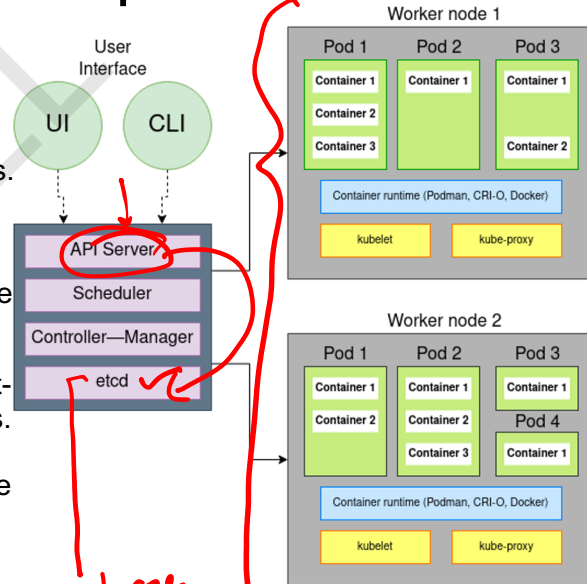
Kubernetes: Control plane Components

Etcd:

Etcd is a fast, distributed, and consistent **key-value store** used as a backing store for persistently storing Kubernetes object data such as pods, replication controllers, secrets, and services.

Etcd is the only place where Kubernetes stores cluster state and metadata. The only component that talks to etcd directly is the Kubernetes API server. All other components read and write data to etcd indirectly through the API server.

Etcd also implements a watch feature, which provides an event-based interface for asynchronously monitoring changes to keys. Once you change a key, its watchers get notified. The API server component heavily relies on this to get notified and move the current state of etcd towards the desired state.



*data base of all cluster states & worker nodes
Kubernetes cluster*

Kubernetes: Control plane Components

API Server:

The API server is the only component in Kubernetes that directly interacts with etcd. All other components in Kubernetes must go through the API server to work with the cluster state, including the clients (kubectl). The API server has the following functions:

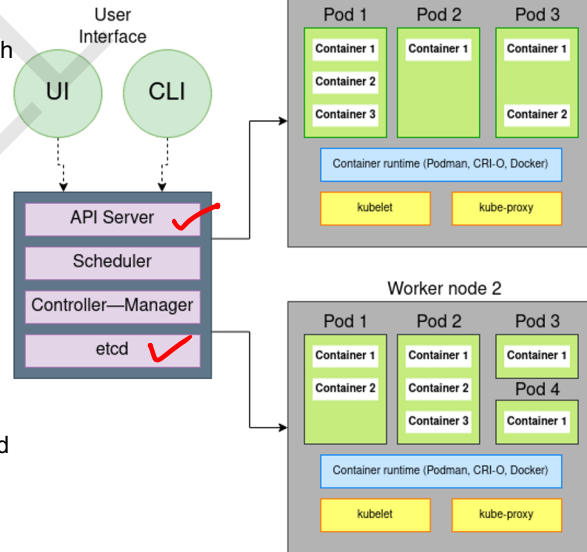
Provides a consistent way of storing objects in etcd.

Performs validation of those objects so clients can't store improperly configured objects.

Provides a RESTful API to create, update, modify, or delete a resource.

Performs authentication and authorization of a request that the client sends.

Responsible for admission control if the request is trying to create, modify, or delete a resource. For example, AlwaysPullImages, DefaultStorageClass, and ResourceQuota.



Kubernetes: Control plane Components

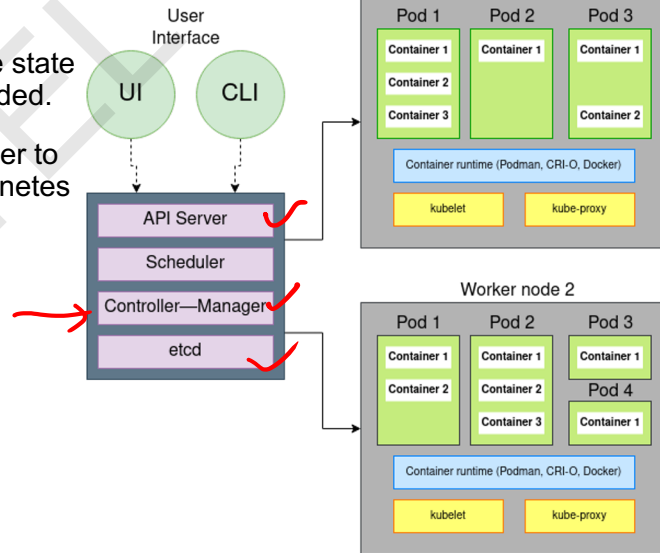
Controller Manager:

In Kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed.

Each controller tries to move the current cluster state closer to the desired state. The controller tracks at least one Kubernetes resource type, and these objects have a spec field that represents the desired state.

Controller examples:

- Node controller
- Service controller
- Endpoints controller
- Namespace controller
- Deployment controller
- StatefulSet controller



Kubernetes: Control plane Components

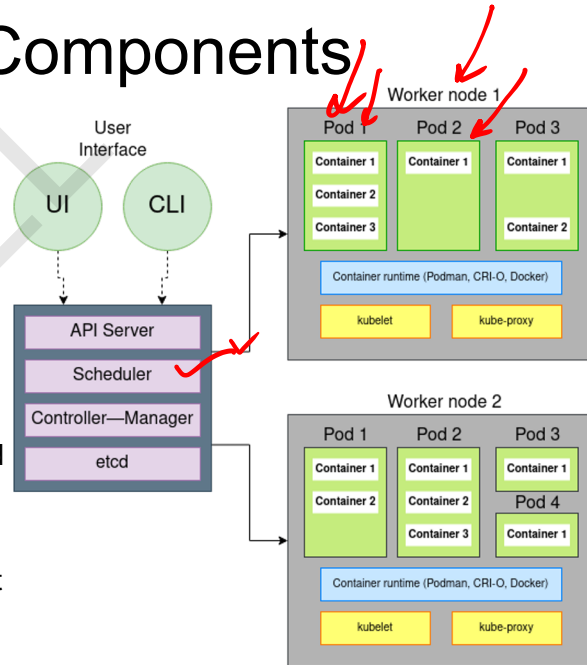
Scheduler:

The Scheduler is a control plane process that assigns Pods to nodes. It watches for newly created pods that have no nodes assigned.

For every pod that the Scheduler discovers, the Scheduler becomes responsible for finding the best node for that pod to run on.

Nodes that meet the scheduling requirements for a pod get called feasible nodes. If none of the nodes are suitable, the pod remains unscheduled until the Scheduler can place it.

Once it finds a feasible node, it runs a set of functions to score the nodes, and the node with the highest score gets selected. It then notifies the API server about the selected node. They call this process binding.



Kubernetes: Worker node components

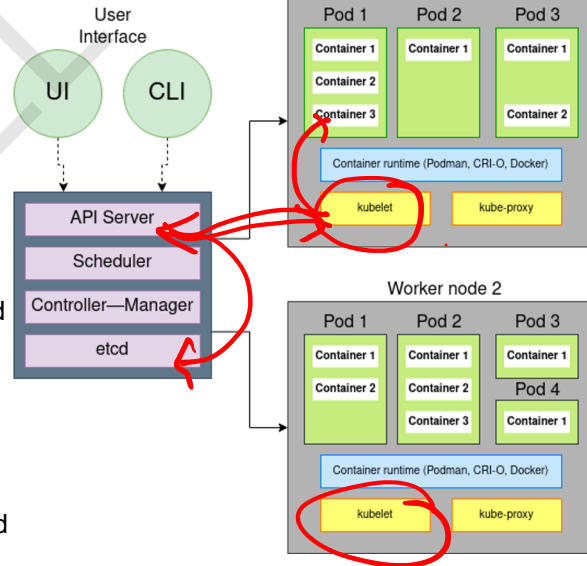
Kubelet:

Kubelet is an agent that runs on each node in the cluster and is responsible for everything running on a worker node.

It ensures that the containers run in the pod.

The main functions of kubelet service are:

- Register the node it's running on by creating a node resource in the API server.
- Continuously monitor the API server for pods that got scheduled to the node.
- Start the pod's containers by using the configured container runtime.
- Continuously monitor running containers and report their status, events, and resource consumption to the API server.
- Run the container liveness probes, restart containers when the probes fail and terminate containers when their pod gets deleted from the API server (notifying the server about the pod termination).



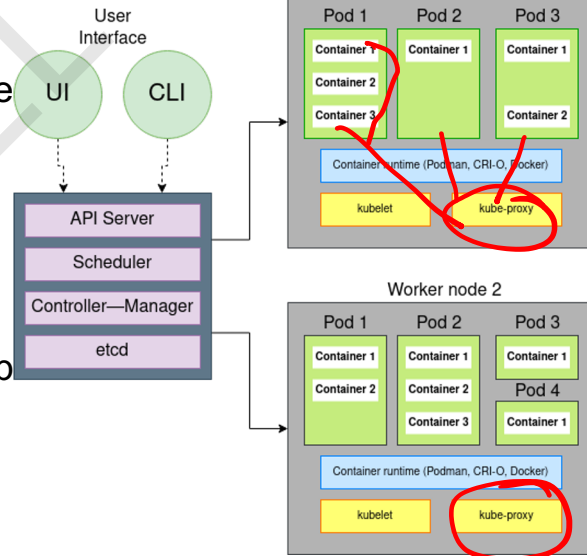
Kubernetes: Worker node components

Service proxy (kube-proxy) :

The service proxy (kube-proxy) runs on each node and ensures that one pod can talk to another pod, one node can talk to another node, and one container can talk to another container.

It is responsible for watching the API server for changes on services and pod definitions to maintain that the entire network configuration is up to date.

When a service gets backed by more than one pod, the proxy performs load balancing across those pods.



Kubernetes: Worker node components

Container runtime:

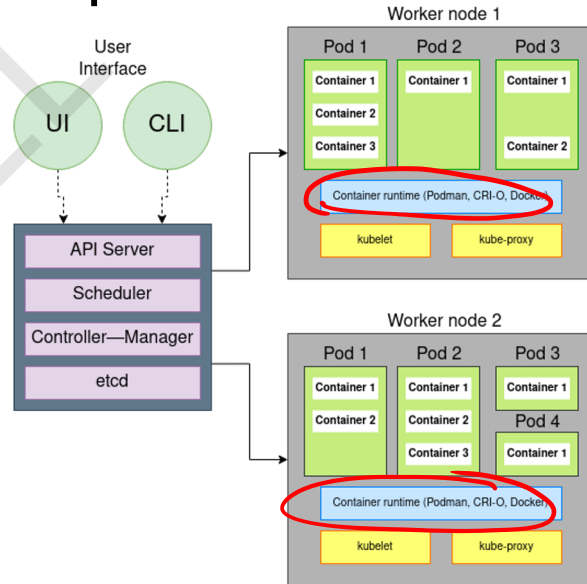
There are two categories of container runtimes:

Lower-level container runtimes: These focus on running containers and setting up the namespace and cgroups for containers.

Higher-level container runtimes (container engine): These focus on formats, unpacking, management, sharing of images, and providing APIs for developers.

Container runtime takes care of:

- Pulls the required container image from an image registry if it's not available locally.
- Prepares a container mount point.
- Alerts the kernel to assign some resource limits like CPU or memory limits.
- Pass system call (syscall) to the kernel to start the container.



Introduction to Dockers

The most popular container technology out in the market is Docker container.

Docker is an open platform for developing, shipping, and running applications.

Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

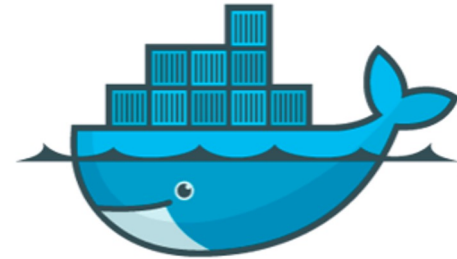
With Docker, you can manage your infrastructure in the same ways you manage your applications.

By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

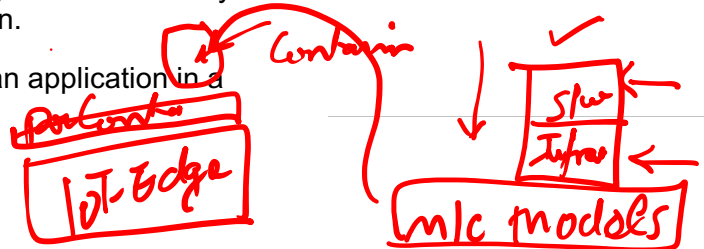
Docker provides the ability to package and run an application in a loosely isolated environment called a container.

Applications are
Containerized

~~Docker~~



docker



Docker Architecture

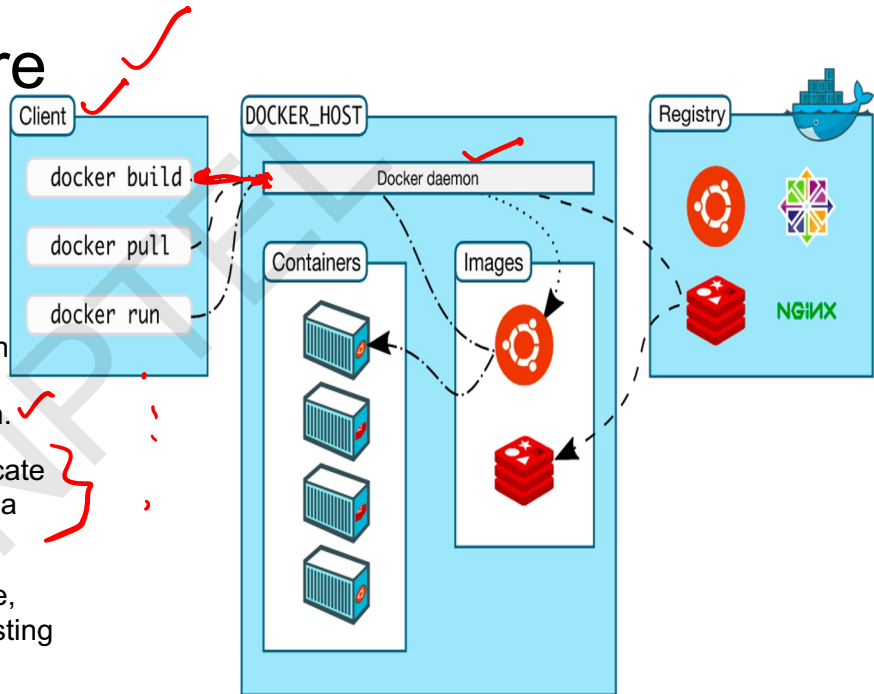
Docker uses a client-server architecture.

The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.

The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.

The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



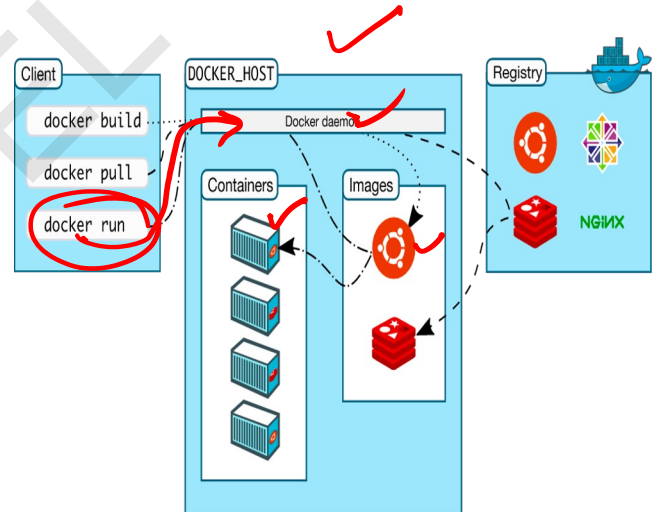
Docker Architecture: Components

The Docker daemon:

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client:

The Docker client is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.



Docker Architecture: Components

Docker registries:

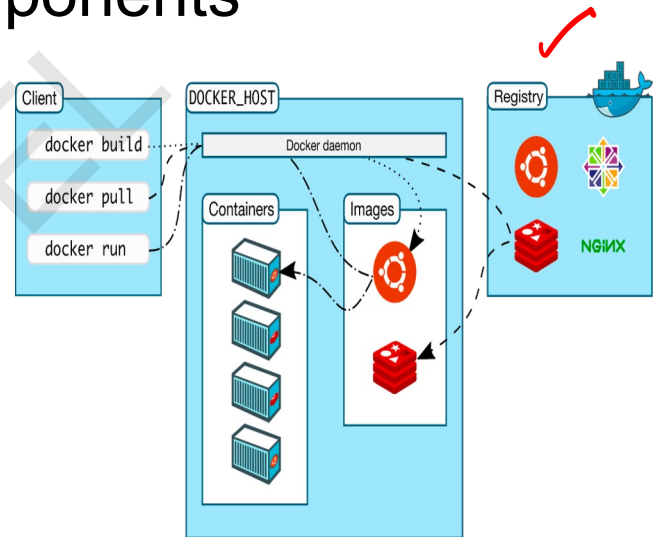
A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

Docker objects:

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects.

Docker Desktop:

Docker Desktop includes the Docker daemon, the Docker client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

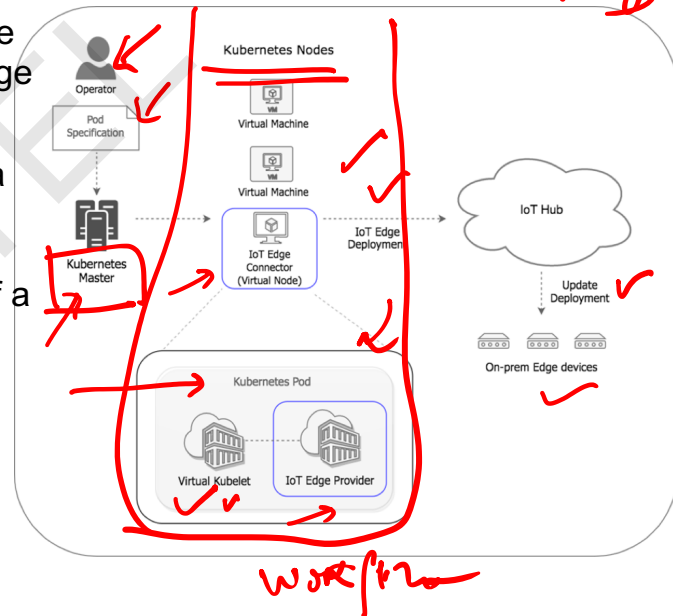


Power of Kubernetes to deploy software on edge devices (1 of 2)

Architecture diagram shows works flow from the cloud through the virtual kubelet through the edge provider down to all of your edge devices

First, the virtual kubelet project lets you create a virtual node in your kubernetes cluster, a virtual node is not a VM like most other nodes in the kubernetes cluster instead it is an abstraction of a kubernetes node that is provided by the virtual kubelet.

Backing it, is an IOT hub, it can schedule workloads to it and treat it like any other kubernetes node.



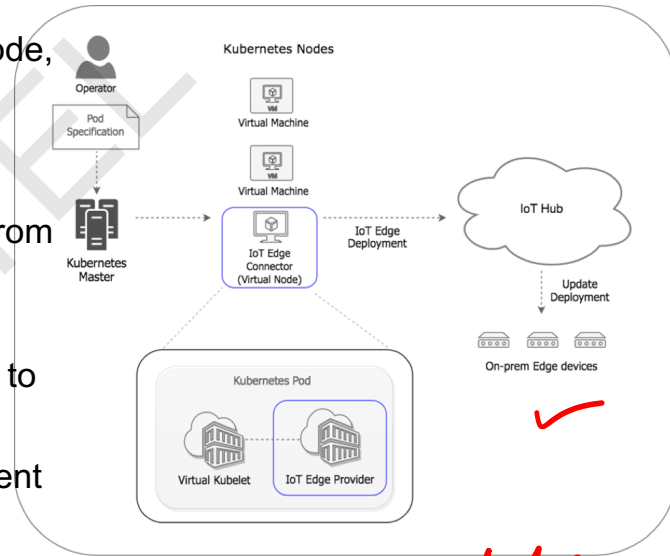
Power of Kubernetes to deploy software on edge devices

When workloads are scheduled to this virtual node, edge provider comes in and that's depicted.

The edge connector or the edge provider which are working in tandem with the virtual kubelet it takes the workload specification that comes in from kubernetes and converts it into an IOT edge deployment.

Then the IOT edge deployment is shipped back to the backing IOT hub for this virtual node.

Lastly, the IOT hub in turn pushes this deployment down to all the targeted devices.



Kubernetes as a solution for Azure IoT hub clusters (IoT-Devices) using Pods

Lecture Summary

- Understanding of Kubernetes including *or, orchestration engine*
 - Containers ✓
 - Orchestration ✓
- Concepts of Dockers ✓
- Power of kubernetes to deploy software on edge devices ✓

IoT



THANK YOU!

ML based Predictive Maintenance at IoT Edge



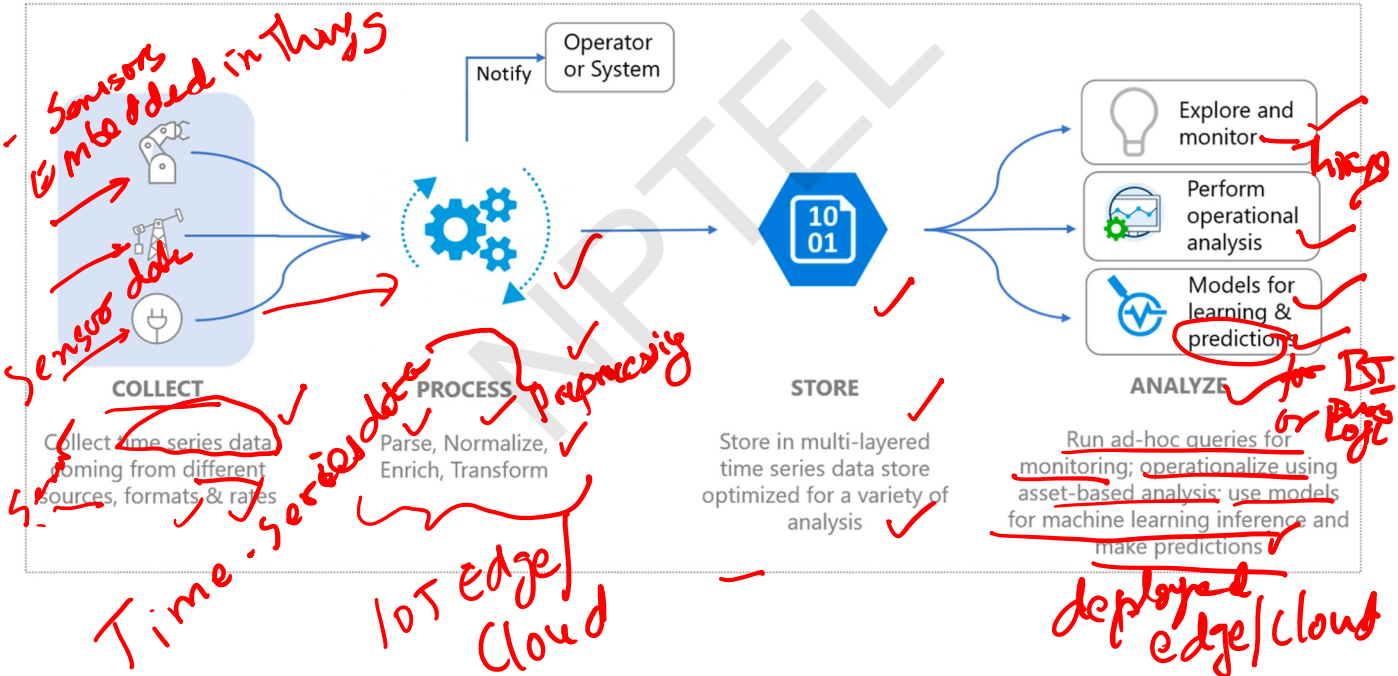
Dr. Rajiv Misra

Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

Lecture Overview

- In this lecture, we combine the Machine Learning (ML) and IoT together.
- The primary objective of this lecture is to introduce the processing of IoT data with machine learning, specifically on the edge.
- While we touch many aspects of a general machine learning workflow, this lecture is not intended as an in-depth introduction to machine learning
- We do not attempt to create a highly optimized model for the use case, it just illustrates the process of creating and using a viable model for IoT data processing.

ML Development at IoT Edge



Machine Learning: Background

✓ Artificial intelligence (A.I.) is defined as the property of machines that mimic human intelligence as characterized by behaviours such as cognitive ability, memory, learning, and decision making.

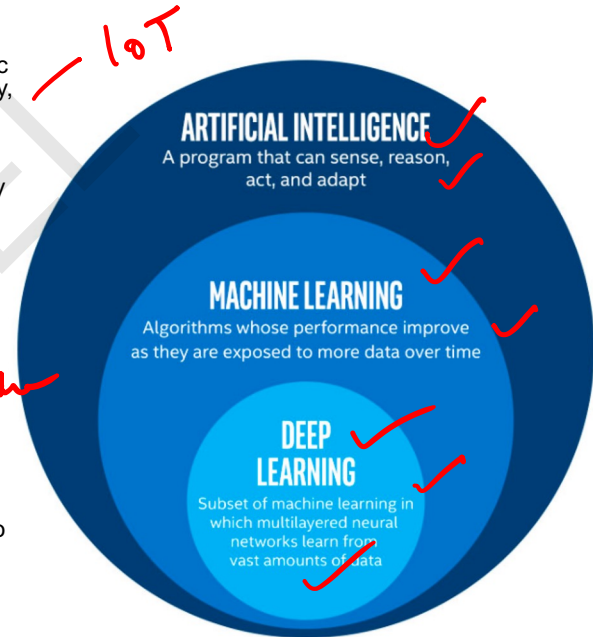
✓ Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

✓ "Deep" machine learning can use labeled datasets, also known as supervised learning, to inform its algorithm, but it doesn't necessarily require a labeled dataset.

Deep learning can ingest unstructured data in its raw form (e.g., text or images), and it can automatically determine the set of features which distinguish different categories of data from one another.

The "deep" in deep learning is just referring to the number of layers in a neural network.

"Non-deep", machine learning is more dependent on human intervention to learn. Human experts determine the set of features to understand the differences between data inputs, usually requiring more structured data to learn.



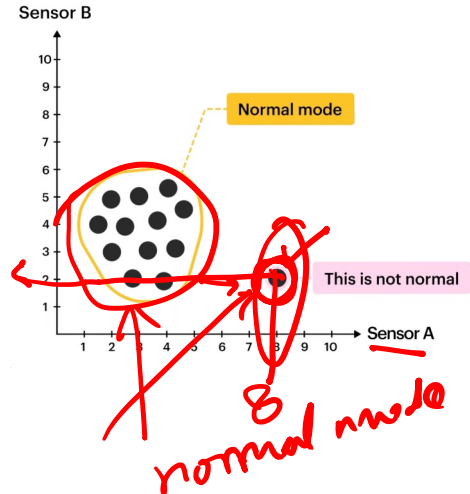
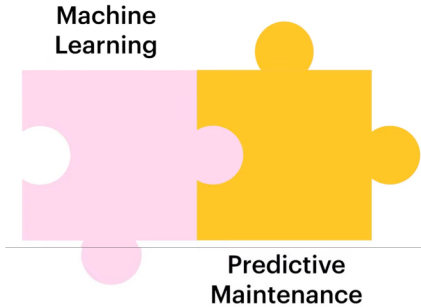
ML for Predictive Maintenance: Example

Using simple machine learning techniques we can create a simple model of a machine with normal operating conditions for any application and determine the values that fall outside of that normal area.

Example: Train a model for the motor vibration with two sensors namely A and B, in normal operating conditions. That means, using normal data points, model has good understanding of what the motor vibration value could be approximately when the motor is operating in normal mode and without any problems.

Now, let's say, one day at a random point in time, model observes that the value of sensor A is 8, and at the same time, the value for sensor B is 2. This is clearly an unusual value. The trained model can easily say that this new value is not normal and can indicate that there might be something wrong with the motor. This is how machine learning works to detect the unusual behavior of a machine.

anomaly detection - yes



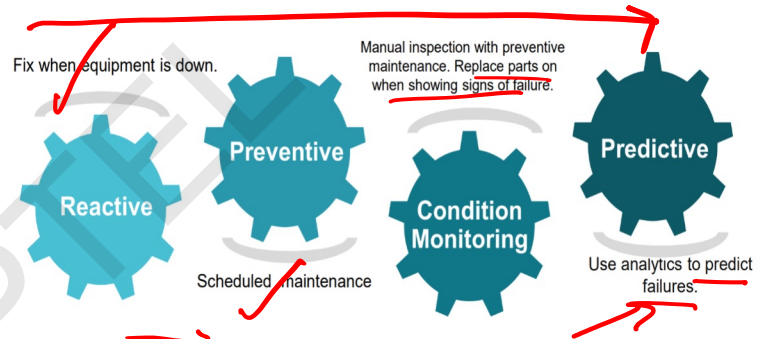
Predictive Maintenance: Introduction

In the past, companies have used **reactive maintenance**, which focused on preparing an asset once failures had occurred.

Then they moved to **preventive maintenance**, also known as the schedule-based or planned maintenance. This refers to performing periodic maintenance based on manufacturers' recommendation. The focus was on reducing the failures by replacing parts based on worst case lifetimes for critical pieces of manufacturing tooling.

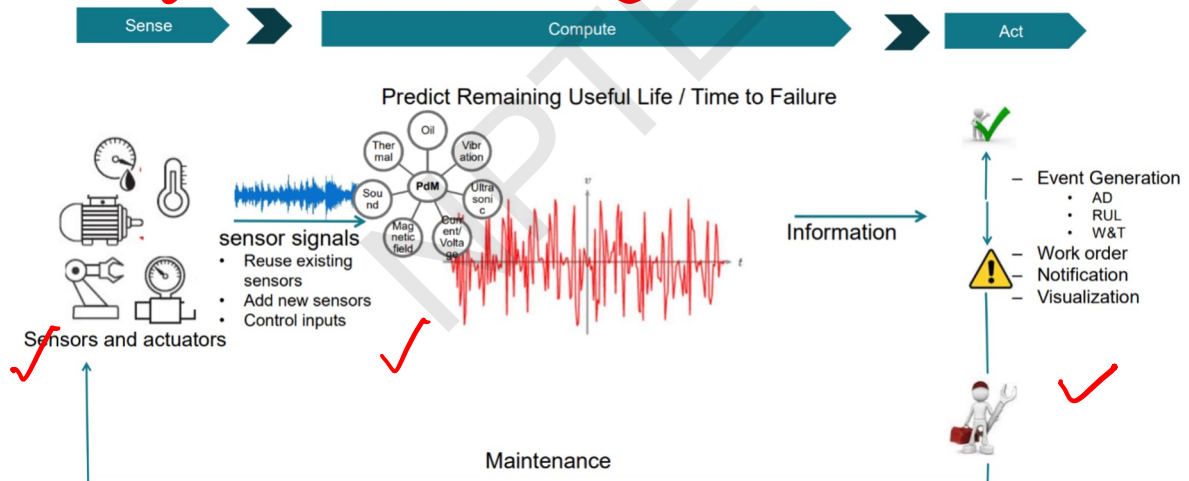
Next came **condition-based maintenance** methods, which repairs or replaces equipment when they begin to show signs of failure. However, this condition-based method requires an experienced maintenance team to inspect the equipment at regular intervals.

With the explosion of computers and sensors, companies are now engaging in **machine-led condition-based maintenance** to reduce costs while improving the uptime of factories. **Predictive maintenance** takes condition-based maintenance a step further. In this methodology, machine learning analytics are used to predict a machine's failure early by examining the real-time sensor data and detecting changes in machine health status.



Predictive Maintenance: Introduction

Predictive maintenance employs advanced analytics, on the machine data collected from end sensor nodes to draw meaningful insights that more accurately predict machine failures. It is comprised of three steps, sense, compute, and act.



Predictive Maintenance: Introduction

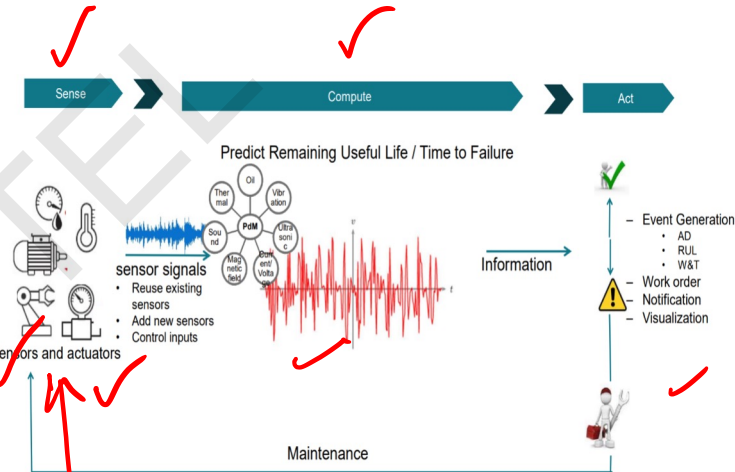
Data is collected from sensors that are already available in machines or by adding new sensors, or by using control inputs.

Depending upon the machine types and the required failure analysis, different sensor signals, such as temperature, sound, magnetic field, current, voltage, ultrasonic, vibration are analyzed to predict the failure.

The predicted information from sensor data analysis is used to generate an event, work order, and notification.

The sensor data is also used to visualize the machine's overall operating condition.

An action is taken when the event reports an anomaly, a machine that is nearing the end of its useful life, or when wear and tear is detected in machine parts.



IoT Sensors embedded in Things - data source

Predictive Maintenance: Problems

Will this equipment fail in a given period of time? ✓

What is the remaining useful life or the time to failure? ✓

How to quantify wear and tear of expandable components. ✓

- This is a subset of remaining useful life and focuses on shorter living subsystems. ✓

For detecting anomalies in equipment behavior.

- With further analysis, it can provide failure classification.

To optimize equipment settings. ✓



Machine Learning Workflow: Predictive Maintenance

A six-step process:

Define the problem and the outcome

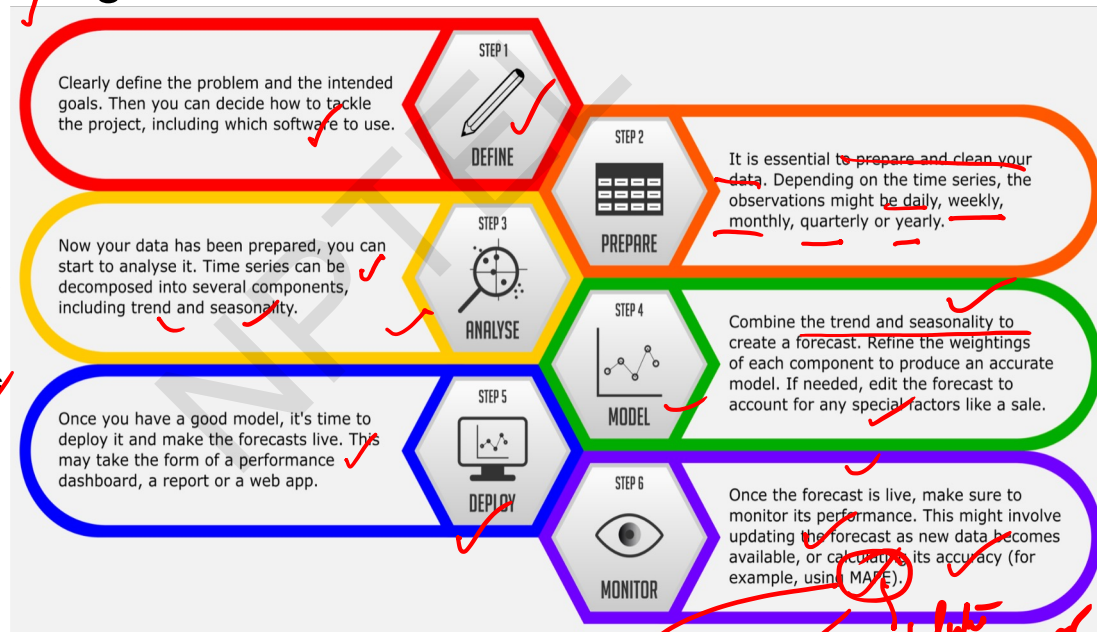
Prepare the data

Analyse the time series

Model the predict using insight from the analysis

Deploy the predictive model

Monitor the predictive performance



mean absolute percentage error

Machine Learning Workflow: Define ✓

As in any successful project, the first step is to clearly define the problem.

This includes the motivation behind creating a predictive and the intended goals and outcomes.

After this, you can decide how to tackle the task at hand, including which software to use at each stage.

For example, we might use Excel for data preparation, R for the analysis and modelling and Power BI for deployment.

Preprocessing data ✓

Machine Learning Workflow: Prepare

It is essential to properly prepare and clean the data that will be used to create the prediction. To train the model with good accuracy

Data cleansing might involve removing duplicated or inaccurate records, or dealing with missing data points or outliers.

In the case of a predictive maintenance project, the data will take the form of a time series.

Depending on what is being predicted, the observations might be daily, weekly, monthly, quarterly or yearly.

Machine Learning Workflow: Analyse

Once the data has been prepared, the next step is to analyse it.

For a time series, this involves decomposing the series into its constituent parts. These include trend and seasonal effects.

The **trend** is the long-term overall pattern of the data and is not necessarily linear.

Seasonality is a recurring pattern of a fixed length which is caused by seasonal factors.

Training ML

Machine Learning Workflow: Modelelling

Predictions are created by combining the trend and seasonality components.

There are functions that can do this for you in Excel, or it can be done by hand in a statistical package like R.

If modelling manually, refine the weightings of each component to produce a more accurate model.

The model can be edited to account for any special factors that need to be included.

However, be careful to avoid introducing bias into the prediction and making it less accurate.

Whether using Excel or R, your model will include prediction intervals (or confidence intervals). These show the level of uncertainty in the prediction at each future point.

Machine Learning Workflow: Deploy

Once you are happy with your model, it's time to deploy it and make the predictions live.

This means that decision makers within the business or organisation can utilise and benefit from your predictions.

Deployment may take the form of a visualisation, a performance dashboard, a graphic or table in a report, or a web application.

You may wish to include with the prediction intervals calculated in the previous step.

These show the user the limits within which each future value can be expected to fall between if your model is correct.

Machine Learning Workflow: Monitor

After the prediction goes live, it is important to monitor its performance.

A common way of doing this is to calculate the accuracy using an error measurement statistic.

Popular measures include the *mean absolute percentage error (MAPE)* and the *mean absolute deviation (MAD)*.

Depending on what is being predicted, it may be possible to update your model as new data becomes available.

This should also lead to a more accurate prediction of future values.

Machine Learning Methods: Predictive Maintenance

Problem definition: Classification and Regression approach

– Classification: Will it fail?

- Multi-class classification: Will it fail for reason X?

– Regression: After how long will it fail?

• Methods:

– Traditional machine learning:

- Decision trees, Random forests, gradient boosting trees, isolation forest
- SVM (Support Vector Machines)

– Deep learning approach:

- CNN (Convolution Neural Network)/Multilayer Perceptrons (MLPs)
- RNN (Recurrent Neural Network)/LSTM (Long Short Term Memory)/GRU (Gated Recurrent Unit)

– Hybrid of deep learning and Physics-Based Modeling (PBM):

- Use PBM to generate training data where lacking
- Use PBM to reduce the problem space (feature engineering)
- Use PBM to inform and validate DL models (e.g., to identify catastrophic failures, most notably in scenarios with low amounts of training data and a high degree of mission criticality)

Deep Learning Methods

Deep learning has proven to show superior performance in certain domains such as object recognition and image classification.

It has also gained popularity in domains such as finance where time-series data plays an important role.

Predictive Maintenance is also a domain where data is collected over time to monitor the state of an asset with the goal of finding patterns to predict failures which can also benefit from certain deep learning algorithms.

Among the deep learning methods, Long Short Term Memory (LSTM) networks are especially appealing to the predictive maintenance domain due to the fact that they are very good at learning from sequences.

This fact lends itself to their applications using time series data by making it possible to look back for longer periods of time to detect failure patterns.

Deep Learning Methods: Multilayer Perceptrons (MLPs) ✓

Generally, neural networks like Multilayer Perceptrons or MLPs provide capabilities that are offered by few algorithms, such as:

- **Robust to Noise.** ✓ Neural networks are robust to noise in input data and in the mapping function and can even support learning and prediction in the presence of missing values.
- **Nonlinear.** Neural networks do not make strong assumptions about the mapping function and readily learn linear and nonlinear relationships.
- **Multivariate Inputs.** An arbitrary number of input features can be specified, providing direct support for multivariate forecasting.
- **Multi-step Forecasts.** An arbitrary number of output values can be specified, providing direct support for multi-step and even multivariate forecasting.

For these capabilities alone, feedforward neural networks may be useful for time series forecasting.

Deep Learning Methods: Convolutional Neural Networks (CNNs)

Convolutional Neural Networks or CNNs are a type of neural network that was designed to efficiently handle image data.

The ability of CNNs to learn and automatically extract features from raw input data can be applied to time series forecasting problems. A sequence of observations can be treated like a one-dimensional image that a CNN model can read and distill into the most salient elements.

- **Feature Learning.** Automatic identification, extraction and distillation of salient features from raw input data that pertain directly to the prediction problem that is being modeled.

CNNs get the benefits of Multilayer Perceptrons for time series forecasting, namely support for multivariate input, multivariate output and learning arbitrary but complex functional relationships, but do not require that the model learn directly from lag observations. Instead, the model can learn a representation from a large input sequence that is most relevant for the prediction problem.

Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by different researchers.

LSTM add the explicit handling of order between observations when learning a mapping function from inputs to outputs, not offered by MLPs or CNNs. They are a type of neural network that adds native support for input data comprised of sequences of observations.

- **Native Support for Sequences.** Recurrent neural networks directly add support for input sequence data.

This capability of LSTMs has been used to great effect in complex natural language processing problems such as neural machine translation where the model must learn the complex interrelationships between words both within a given language and across languages in translating from one language to another.

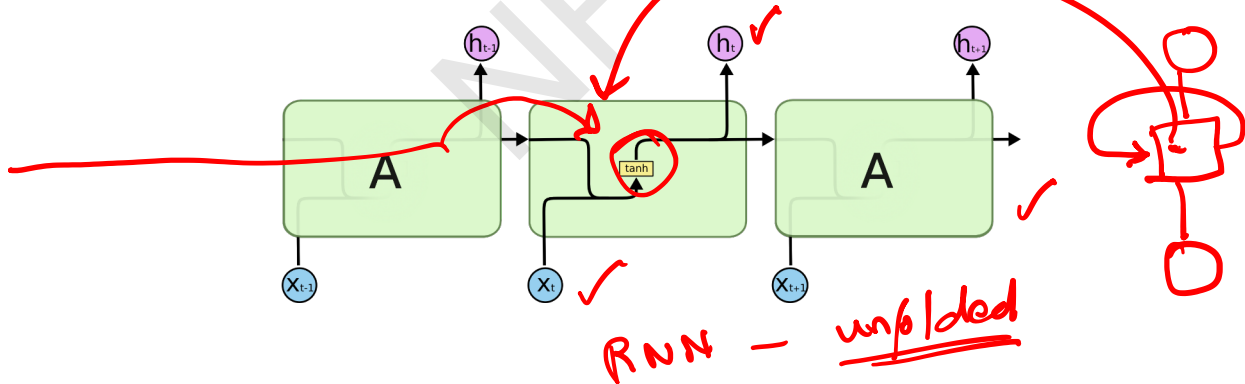
- **Learned Temporal Dependence.** The most relevant context of input observations to the expected output is learned and can change dynamically.

Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

The model both learns a mapping from inputs to outputs and learns what context from the input sequence is useful for the mapping, and can dynamically change this context as needed.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. LSTM

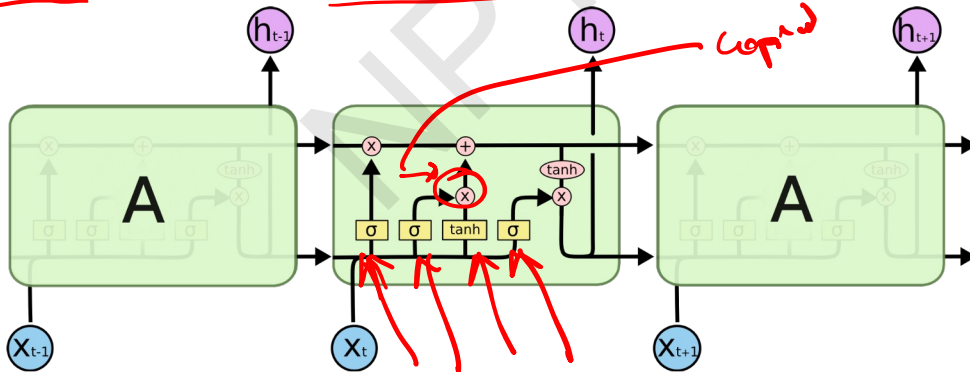
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

In the below diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

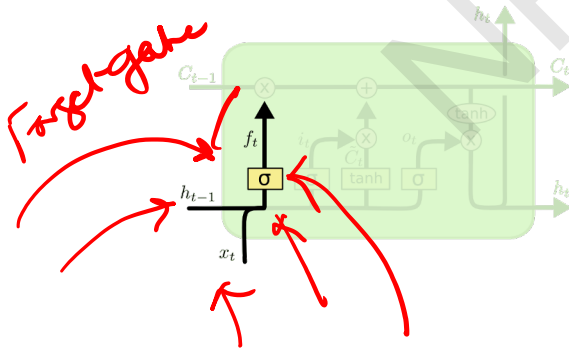


Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

An LSTM has three of gates, to protect and control the cell state. The first part is called **Forget gate**, the second part is known as the **Input gate** and the last one is the **Output gate**.

Forget Gate: The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1} .

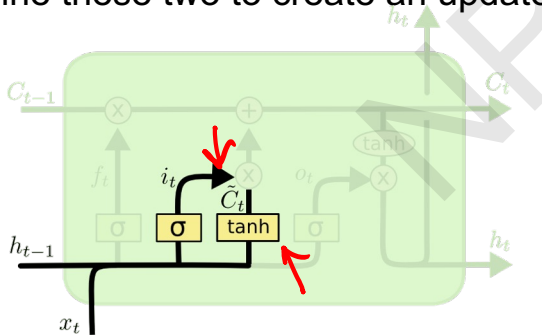
A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

Input Gate: The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a *tanh* layer creates a vector of new candidate values, \tilde{C}_t , that could be added to the state. In the next step, we'll combine these two to create an update to the state.

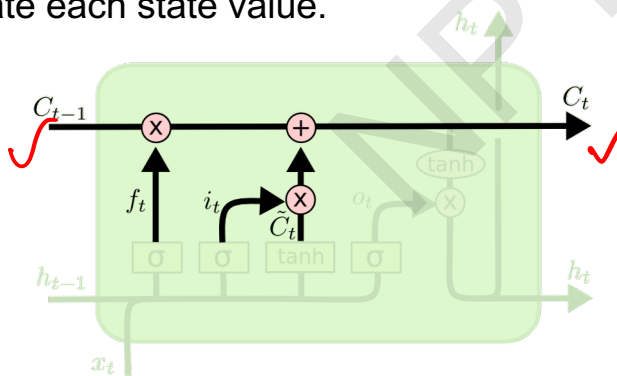


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

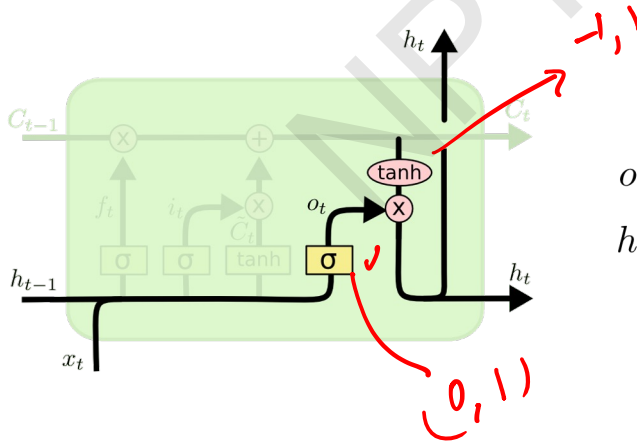
It's now time to update the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to actually do it. We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Deep Learning Methods: Long Short-Term Memory Networks (LSTMs)

Output gate: Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Performance Metric: R-squared

The stationary R-squared is used in time series forecasting as a measure that compares the stationary part of the model to a simple mean model. It is defined as,

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Where SS_{res} denotes the sum of squared residuals from expected values and SS_{tot} denotes the sum of squared deviations from the dependent variable's sample mean. It denotes the proportion of the dependent variable's variance that may be explained by the independent variable's variance. A high R^2 value shows that the model's variance is similar to that of the true values, whereas a low R^2 value suggests that the two values are not strongly related.

Performance Metric: Mean Absolute Error (MAE) ✓

The MAE is defined as the average of the absolute difference between forecasted and true values. Where y_i is the expected value and x_i is the actual value (shown below formula). The letter n represents the total number of values in the test set.

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

The MAE shows us how much inaccuracy we should expect from the forecast on average. MAE = 0 means that the anticipated values are correct, and the error statistics are in the original units of the forecasted values.

The lower the MAE value, the better the model; a value of zero indicates that the forecast is error-free. In other words, the model with the lowest MAE is deemed superior when comparing many models.

Performance Metric: Mean Absolute Percentage Error (MAPE)

MAPE is the proportion of the average absolute difference between projected and true values divided by the true value. The anticipated value is F_t , and the true value is A_t . The number n refers to the total number of values in the test set.

$$\text{MAPE} = \frac{100}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right|$$

It works better with data that is free of zeros and extreme values because of the in-denominator. The MAPE value also takes an extreme value if this value is exceedingly tiny or huge.

Performance Metric: Mean Squared Error (MSE)

MSE is defined as the average of the error squares. It is also known as the metric that evaluates the quality of a forecasting model or predictor. MSE also takes into account variance (the difference between anticipated values) and bias (the distance of predicted value from its true value).

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Where \hat{y}_i denotes the predicted value and y_i denotes the actual value. The number n refers to the total number of values in the test set. MSE is almost always positive, and lower values are preferable. This measure penalizes large errors or outliers more than minor errors due to the square term (as seen in the formula above).

Performance Metric: Root Mean Squared Error(RMSE)

This measure is defined as the square root of mean square error and is an extension of MSE. Where \hat{y}_i denotes the predicted value and y_i denotes the actual value. The number n refers to the total number of values in the test set. This statistic, like MSE, penalizes greater errors more.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2}$$

This statistic is likewise always positive, with lower values indicating higher performance. The RMSE number is in the same unit as the projected value, which is an advantage of this technique. In comparison to MSE, this makes it easier to comprehend.

Use Case: Prognostics and Health Management

The objective of this use case is to build an LSTM model that can predict the number of remaining operational cycles before failure in the test set, i.e., the number of operational cycles after the last cycle that the engine will continue to operate. Also provided a vector of true Remaining Useful Life (RUL) values for the test data.

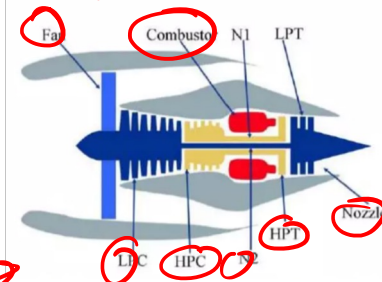
The data was generated using C-MAPSS, the commercial version of MAPSS (Modular Aero-Propulsion System Simulation) software. This software provides a flexible turbofan engine simulation environment to conveniently simulate the health, control, and engine parameters.

The simulated aircraft sensor values is used to predict two scenarios, so that maintenance can be planned in advance:

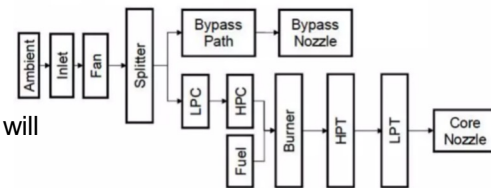
* **Regression models:** The question to ask is "Given these aircraft engine operation and failure events history, can we predict when an in-service engine will fail?"

* **Binary classification:** We re-formulate this question "Is this engine going to fail within w1 cycles?"

Aircraft



Simplified diagram of engine simulation



A layout showing various modules and their connections as modeled in the simulation

LSTM model: Dataset

Dataset consists of multiple multivariate time series, such data set is divided into training and test subsets. Each time series is from a different engine. The engine is operating normally at the start of each time series and develops a fault at some point during the series.

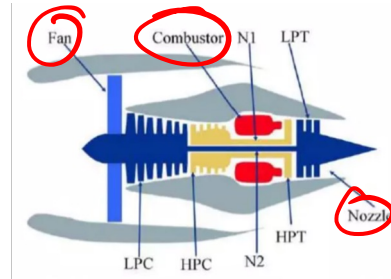
In the training set, the fault grows in magnitude until system failure. In the test set, the time series ends some time prior to system failure.

Public dataset (Nasa Turbo fan)

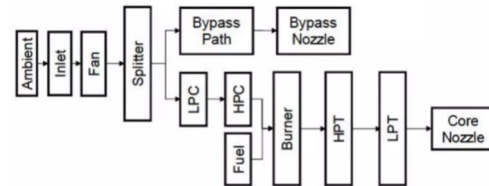
- Damage propagation for aircraft engine
- Run to failure simulation

Aircraft gas turbine. Dataset contains time series (cycles) for all measurements of 100 different engines.

The data used in this use-case is taken from the <https://www.nasa.gov/intelligent-systems-division>



Simplified diagram of engine simulation



A layout showing various modules and their connections as modeled in the simulation

LSTM model: Data Ingestion

We ingest the training, test and ground truth datasets.

The training data consists of multiple multivariate time series with "cycle" as the time unit, together with 21 sensor readings for each cycle.

Each time series can be assumed as being generated from a different engine of the same type.

The testing data has the same data schema as the training data. The only difference is that the data does not indicate when the failure occurs.

Finally, the ground truth data provides the number of remaining working cycles for the engines in the testing data.

```
train_df = train_df.sort_values(['id','cycle'])
train_df.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

5 rows x 26 columns

LSTM model: Data Preparation and Feature Engineering

First step is to generate labels for the training data which are Remaining Useful Life (RUL), label1 and label2.

Each row can be used as a model training sample where the s_k columns are the features and the RUL is the model target. The rows are treated as independent observations and the measurement trends from the previous cycles are ignored. The features are normalized to $\mu = 0$, $\sigma = 1$ and PCA is applied.

For the LSTM model, opt for more advanced feature engineering and chose to incorporate the trends from the previous cycles. In this case, each training sample consists of measurements at cycle i as well as $i-5$, $i-10$, $i-20$, $i-30$, $i-40$. The model input is a 3D tensor with shape $(n, 6, 24)$ where n is the number of training samples, 6 is the number of cycles (timesteps), and 24 is the number of principal components (features).

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s16	s17	s18	s19	s20	s21	cycle_norm	RUL	label1	label2
0	1	1	0.632184	0.750000	0.0	0.0	0.545181	0.310661	0.269413	0.0	...	0.0	0.333333	0.0	0.0	0.558140	0.661834	0.00000	142	0	0
1	1	2	0.344828	0.250000	0.0	0.0	0.150602	0.379551	0.222316	0.0	...	0.0	0.416667	0.0	0.0	0.682171	0.686827	0.0027	141	0	0
2	1	3	0.517241	0.583333	0.0	0.0	0.376506	0.346632	0.322248	0.0	...	0.0	0.416667	0.0	0.0	0.728682	0.721348	0.00554	140	0	0
3	1	4	0.741379	0.500000	0.0	0.0	0.370482	0.285154	0.408001	0.0	...	0.0	0.250000	0.0	0.0	0.666667	0.662110	0.0083	139	0	0
4	1	5	0.580460	0.500000	0.0	0.0	0.391566	0.352082	0.332039	0.0	...	0.0	0.166667	0.0	0.0	0.658915	0.716377	0.0110	138	0	0

5 rows x 30 columns

LSTM model: Modelling

When using LSTMs in the time-series domain, one important parameter to pick is the sequence length which is the window for LSTMs to look back.

This may be viewed as similar to picking `window_size = 5` cycles for calculating the rolling features which are rolling mean and rolling standard deviation for 21 sensor values.

The idea of using LSTMs is to let the model extract abstract features out of the sequence of sensor values in the window rather than engineering those manually. The expectation is that if there is a pattern in these sensor values within the window prior to failure, the pattern should be encoded by the LSTM.

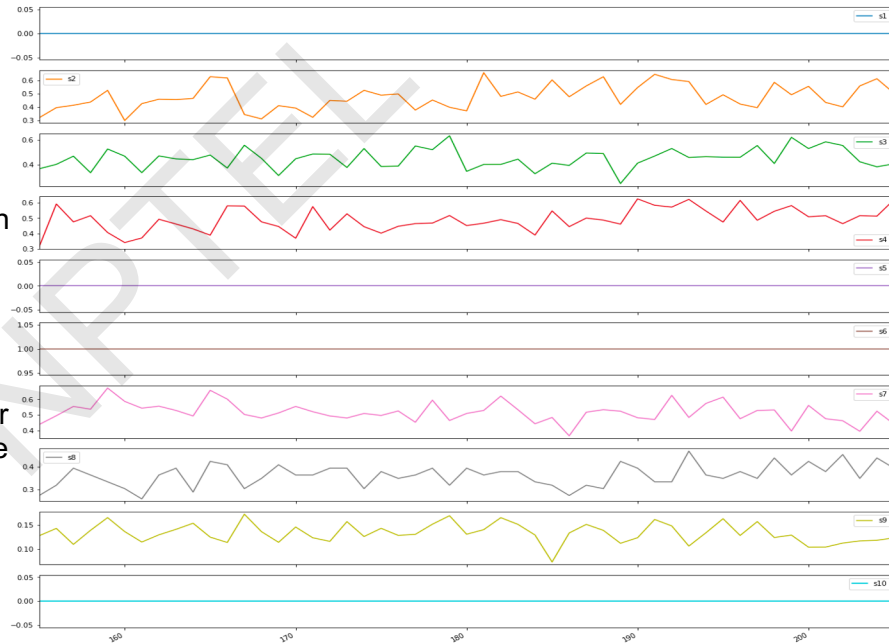
One critical advantage of LSTMs is their ability to remember from long-term sequences (window sizes) which is hard to achieve by traditional feature engineering. For example, computing rolling averages over a window size of 50 cycles may lead to loss of information due to smoothing and abstracting of values over such a long period, instead, using all 50 values as input may provide better results. While feature engineering over large window sizes may not make sense, LSTMs are able to use larger window sizes and use all the information in the window as input.

LSTM model: Modelling

Let's first look at an example of the sensor values 50 cycles prior to the failure for engine id 3. ✓

We will be feeding LSTM network this type of data for each time step for each engine id.

LSTM layers expect an input in the shape of a numpy array of 3 dimensions (samples, time steps, features) where samples is the number of training sequences, time steps is the look back window or sequence length and features is the number of features of each sequence at each time step.



LSTM model: Network Configuration

The first layer is an LSTM layer with 100 units followed by another LSTM layer with 50 units.

Dropout is also applied after each LSTM layer to control overfitting.

Final layer is a Dense output layer with single unit with sigmoid activation for the binary classification problem and linear activation for the regression problem.

Network for binary classification problem

Layer (type)	Output Shape	Param #
lstm_11 (LSTM)	(None, 50, 100)	50400
dropout_11 (Dropout)	(None, 50, 100)	0
lstm_12 (LSTM)	(None, 50)	30200
dropout_12 (Dropout)	(None, 50)	0
dense_6 (Dense)	(None, 1)	51

Total params: 80,651

Trainable params: 80,651

Non-trainable params: 0

Network for regression problem

Layer (type)	Output Shape	Param #
lstm_13 (LSTM)	(None, 50, 100)	50400
dropout_13 (Dropout)	(None, 50, 100)	0
lstm_14 (LSTM)	(None, 50)	30200
dropout_14 (Dropout)	(None, 50)	0
dense_7 (Dense)	(None, 1)	51
activation_3 (Activation)	(None, 1)	0

Total params: 80,651

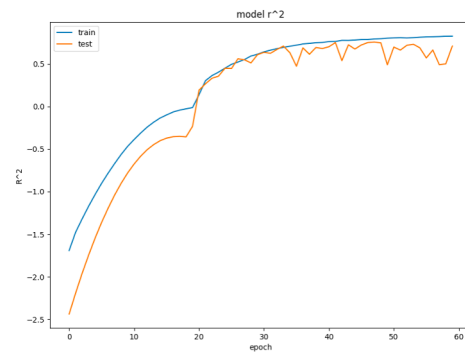
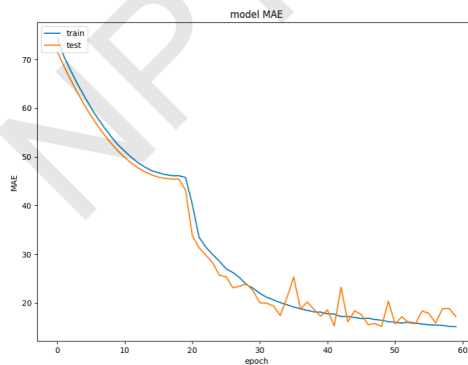
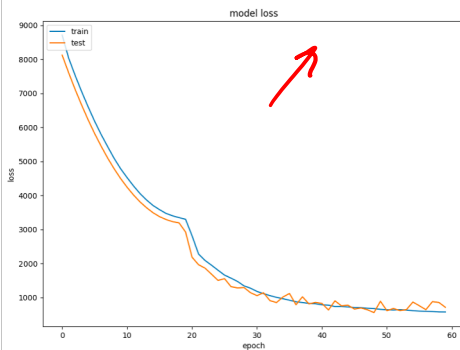
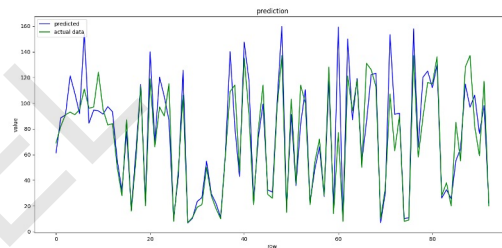
Trainable params: 80,651

Non-trainable params: 0

LSTM model: Model Evaluation

Results of Regression problem:

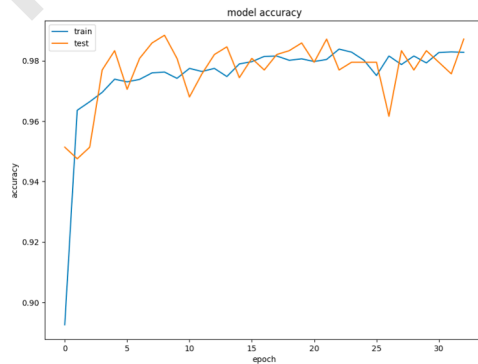
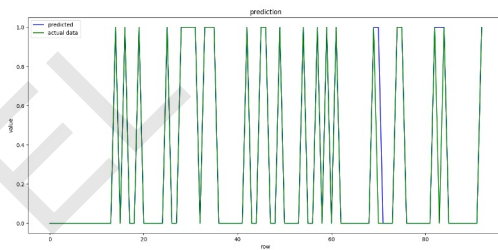
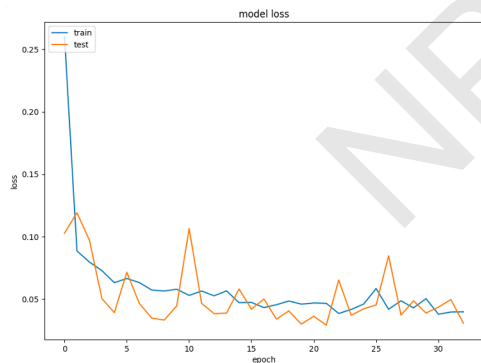
Mean Absolute Error	Coefficient of Determination (R^2)
12	0.7965



LSTM model: Model Evaluation

Results of Binary Classification problem:

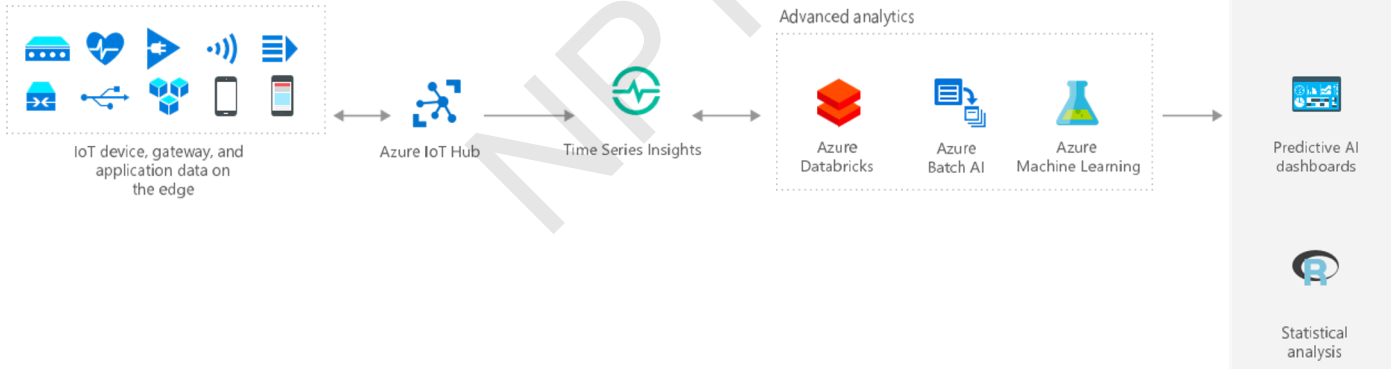
Accuracy	Precision	Recall	F-Score
0.97	0.92	1.0	0.96



Azure Time Series Insights (PaaS): Predictive Maintenance

Azure Time Series Insights (TSI) is a cloud-based service offered by Azure that can be used to ingest, model, query and visualize fast-moving time-series data generated by IoT devices.

It is a fully managed Platform as a Service(PaaS) solution built in for IoT.



Azure Time Series Insights (PaaS): Predictive Maintenance

Real-time data in the form of a time-series can be generated by various devices like mobile devices, sensors, satellites, medical devices etc.

Data from these devices can be fetched to the Azure environment using Azure IoT Hub. Azure IoT hub acts as a data integration pipeline to connect to the source devices and then fetch data and deliver it to the TSI platform.

Once the data is in the TSI, it can then be used for visualization purposes, and can be queried and aggregated accordingly. Additionally, customers can also leverage existing analytics and machine learning capabilities on top of the data available in TSI.

Data from TSI can be further processed using Azure Databricks and machine learning models can be applied based on pre-trained models that will offer predictions in real-time. This is how an overall architecture of Azure Time Series Insights can be enabled.



Azure Time Series Insights (PaaS): Components

Azure TSI provides the following four components using which users can consume data from varied data sources as follows.

- **Integration** – TSI provides an easy integration from data generated using IoT devices by allowing connection between the cloud data gateways such as Azure IoT Hub and Azure Event Hubs. Data from these sources can be easily consumed in JSON structures, cleaned and then stored in a columnar store
- **Storage** – Azure TSI also takes care of the data that is to be retained in the system for querying and visualizing the data. By default, data is stored on SSDs for fast retrieval and has a data retention policy of 400 days. This supports querying historic data for up to a period of 400 days
- **Data Visualization** – Once the data is fetched from the data sources and stored in the columnar stores, it can be visualized in the form of line charts or heat maps. The visuals are provided out of the box by Azure TSI and can be leveraged for easy visual analysis
- **Query Service** – Although, visualizing the data will answer many questions, however, TSI also provides a query service using which you can integrate TSI into your custom applications.

Usually, a time series data is indexed by timestamps. Therefore, you can build your applications by using TSI as a backend service for integrating and storing the data and using the client SDK for Azure TSI for building the frontend and display visuals like line charts and heat maps.

Predictive Maintenance: Steps

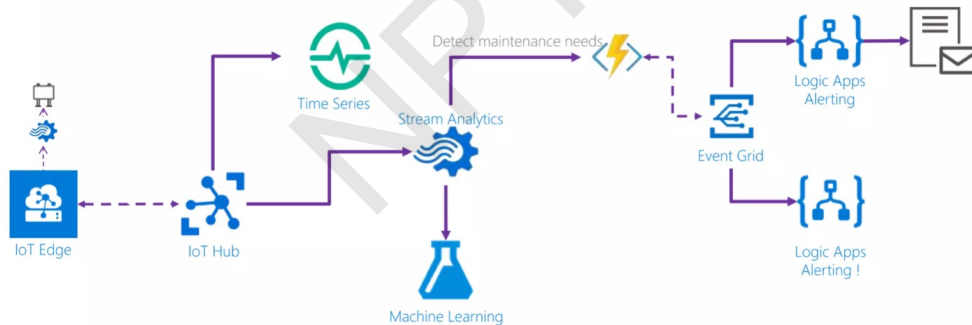
Step 1: Sensor data are collected from edge devices and are forwarded to Azure IoT Hub.

Step 2: Azure IoT hub then drives these gathered data to the TSI platform and Stream Analytics.

Step 3: At TSI, data can be visualised, queried and aggregated with other services.

Step 4: Azure machine learning service provides the training of ML model or using a pretrained model on top of the data available in TSI.

Step 5: Once the training is completed, inference is provided using Azure IoT Hub and IoT Edge service.



Lecture Summary

- Understanding of predictive maintenance
- Machine learning models for predictive maintenance
- Use case of predictive maintenance using LSTM model
- Azure Time Series Insights



THANK YOU!

Deep Reinforcement Learning for Cloud-Edge



Dr. Rajiv Misra

Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Preface

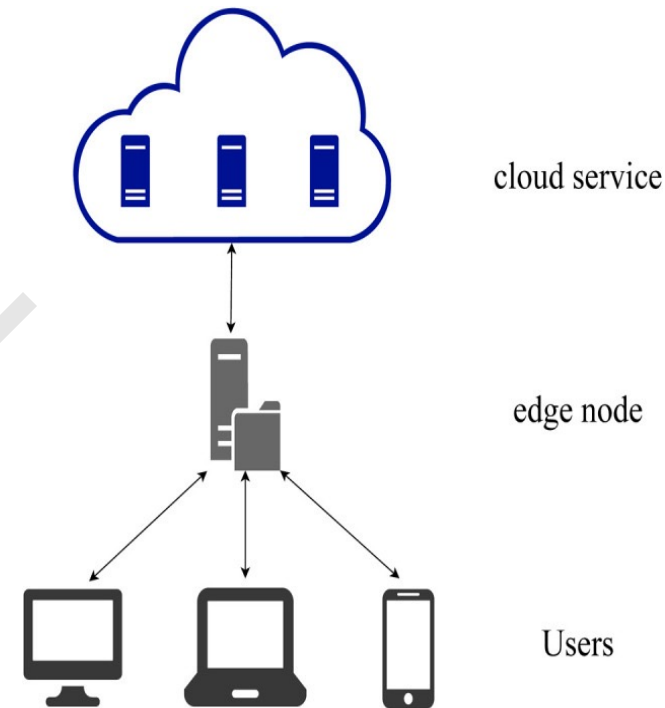
Content of this Lecture:

- In this lecture, we will discuss how Collaborative cloud-edge approaches can provide better performance and efficiency than traditional cloud or edge approaches.
- To understand how resource allocation strategies can be tailored to specific use cases and can evolve over time based on user demand and network conditions.

The Collaborative Cloud-Edge Environment

Introduction:

- The "user-edge-cloud" model refers to a distributed computing environment where resources are allocated across user devices, edge nodes, and cloud servers.
- Resource allocation is important for optimizing system performance while ensuring efficient use of resources.
- Collaborative cloud-edge approaches can be more effective than traditional approaches that focus solely on cloud or edge resources.



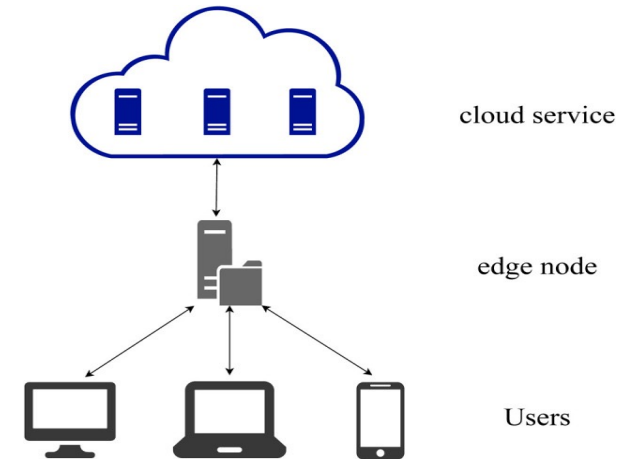
Cloud Services:

- Cloud services can be divided into private and public cloud.
- Private cloud is dedicated to a single organization and provides greater control and security.
- Public cloud is shared by multiple organizations and provides more flexibility and scalability.

The Collaborative Cloud-Edge Environment

Edge Nodes:

- Edge nodes are local computing resources that are closer to the user than the cloud node.
- Edge nodes can provide low-latency, high-bandwidth services to users and can offload some processing from the cloud.



Resource Allocation Strategies:

- Cloud Resource allocation strategies can be based on various factors, such as user demand, network conditions, and available resources.
- Collaborative cloud-edge approaches can use machine learning algorithms to optimize resource allocation over time.
- Load balancing, task offloading, and caching are some common resource allocation techniques that can be applied to both cloud and edge resources.

Multi-Edge-Node Scenario:

- Cloud In a multi-edge-node scenario, resource allocation becomes more complex as the cloud and edge nodes must coordinate with each other to allocate resources effectively.
- Collaborative cloud-edge approaches can use communication protocols and data sharing to enable effective coordination.

Public vs Private Cloud

Public Cloud Environment:

- In a public cloud environment, the cloud provider offers different pricing modes for cloud services based on demand characteristics.
- Pricing modes have different cost structures that affect resource allocation strategies.
- Cloud service providers like Amazon, Microsoft, and Alicloud provide three different pricing modes, each with different cost structures.
- The edge node must select the appropriate pricing mode and allocate user demands to rented VMs or its own VMs.

Private Cloud Environment:

- In a private cloud environment, the edge node has its own virtual machines (VMs) to process user demands.
- If the number of VMs requested exceeds the edge node's capacity, the edge node can rent VMs from the cloud node to scale up.
- The cost of private cloud changes dynamically according to its physical computing cost, so the edge node needs to allocate resources dynamically at each time slot according to its policy.
- After allocating resources, the computing cost of the edge node and private cloud in this time slot can be calculated and used to receive new computing tasks in the next time slot.

User Settings

The time is discretized into T time slots.

We assume that in each time slot t , the demand submitted by the user can be defined as the following:

$$\mathbf{D}_t = (d_t, l_t)$$

\mathbf{D}_t is a pair of d_t and l_t , where d_t is the number of VMs requested of D_t , and l_t is the computing time duration of D_t .

Computing Resources and Cost of Edge Nodes:

- The total computing resources owned by the edge node are represented by E .
- As the resource is allocated to users, we use e_t to represent the number of remaining VMs of edge node in time slot t .
- The number of VMs provided by the edge node is expressed as d_t^e .
- The number of VMs provided by the cloud node is expressed as d_t^c .
- It should be noted that if the edge node exhibits no available resources, it will hand over all the arriving computing tasks to the cloud service for processing. So, no. of VM provided by edge node in time t is given as:

$$d_t^e = \begin{cases} d_t - d_t^c, & e_t \geq 0 \\ 0, & e_t = 0 \end{cases}$$

Computing Resources and Cost of Edge Nodes:

- When the resource allocation is successfully performed on the edge node, each demand processed by the edge node will generate an allocation record.

$$h_t = (d_t^e, l_t)$$

- When a new demand arrives and resource allocation is completed, an allocation record will be generated and added to an allocation record list:

$$H = \langle h_1, h_2, \dots, h_m \rangle$$

At the end of each time slot, the following actions are taken:

- The edge node traverses the allocation record list and subtracts one from the remaining computing time of each record.
- If a record's remaining computing time reaches 0, it means that the demand has been completed. The edge node releases the corresponding VMs and deletes the allocation record from the list.
- The number of VMs waiting to be released at the end of time slot t is denoted as η_t .

$$\eta_t = \sum_{i=1}^m d_i^c$$
$$s. t. l_i = 0, h_i \in H$$

Computing Resources and Cost of Edge Nodes:

- The number of remaining VMs at the next time slot $t+1$ is calculated based on the number of remaining VMs at the beginning of time slot t , the quantity allocated in the end of time slot t , and the quantity released due to completion of the computing task in time slot t . Then, the number of remaining VMs of the edge node at the time slot $t + 1$ is

$$e_{t+1} = e_t - d_t^e + \eta_t$$

- The cost of the edge node in time slot t is calculated as the sum of standby cost ($e_t p_e$) and computing cost $((E - e_t)p_f)$.

$$C_t^e = e_t p_e + (E - e_t)p_f$$

Cost of Collaborative Cloud-Side Computing

Cost in Private Cloud:

- In time slot t , the cost of collaborative cloud-edge in private cloud environment is the following:

$$C_t^{pri} = d_t^c p_c + C_t^e$$

Where,

d_t^c : number of VMs provided by cloud node

p_c : unit cost of VMs in private cloud

C_t^e : cost of the edge node

Cost in Public Cloud:

- In time slot t , the cost of collaborative cloud-edge in public cloud environment includes the computing cost of cloud nodes and the cost of edge node, which is the following:

$$C_t^{pub} = X_1 p_{od} d_t^c + X_2 p_{upfront} + X_3 p_{re} d_t^c + X_4 p_t d_t^c + C_t^e$$

$$X_i = \begin{cases} 1, & \text{The service is used} \\ 0, & \text{The service is not used} \end{cases}$$

Where,

$X_1 p_{od} d_t^c$: cost of on-demand instance

$X_2 p_{upfront} + X_3 p_{re} d_t^c$: cost of reserved instance

$X_4 p_t d_t^c$: cost of spot instance

Goal

- The time is divided into T time slots, and at the beginning of each time slot t , the user submits its demand to the edge node.
- The edge node allocates the demands to either cloud VMs or its own VMs based on its resource allocation strategy.
- In a public cloud environment, the edge node determines the type of cloud service to be used based on the allocation and the price of the corresponding cloud service set by the cloud service provider.
- The cost of the current time slot t , denoted as C_t , is calculated based on the allocation and the price of the corresponding cloud service set by the cloud service provider.
- The long-term cost of the system is minimized over the T time slots by minimizing the sum of the costs over all time slots i.e.

$$\sum_{t=1}^T C_t$$

Resource Allocation Algorithms: 1. Markov Decision Process

- The **resource allocation problem is a sequential decision-making problem**
- It can be modeled as a Markov decision process.
- Markov decision process is a tuple (S, A, P, r, γ) , where S is the finite set of states, A the finite set of actions, P is the probability of state transition, r and γ are the immediate reward and discount factor, respectively.
- $\mathbf{s}_t = (\mathbf{e}_t, \boldsymbol{\eta}_{t-1}, \mathbf{D}_t, p_t) \in S$, is used to describe the state of the edge node at the beginning of each time slot, where
 - \mathbf{e}_t : number of remaining VMs of the edge node in \mathbf{t} ,
 - $\boldsymbol{\eta}_{t-1}$: number of VMs returned in the previous time slot
 - \mathbf{D}_t : user's demand information in \mathbf{t}
 - p_t : unit cost of VMs in private cloud in \mathbf{t} .
- $\mathbf{a}_t = (\mathbf{x}_e, \mathbf{x}_k) \in A$, where
 - \mathbf{x}_e : ratio of the number of VMs provided by the edge node to the total number of VMs.
 - \mathbf{x}_k : ratio of the number of VMs provided by the cloud node to the total number of VMs.
- $\mathbf{r}_t = -\mathbf{C}_t^{pri}$ is the reward in each time slot

Note :

We want to reduce the long-term operation cost $R = \sum_{i=1}^T r(s_i, a_i)$ therefore, the reward function is set as a negative value of the cost.

2. Parameterized Action Markov Decision Process

- In the public cloud environment, first, the edge node needs to select the pricing mode of cloud service to be used and then determine the resource segmentation between the edge node and the cloud node in each time slot t .
- The resource allocation action can be described by parametric action.
- In order to describe this parameterized action sequential decision, parameterized action Markov decision process (PAMDP) is used.
- Similar to Markov decision process, PAMDP is a tuple (S, A, P, r, γ) .
- The difference with the Markov decision process is that A is the finite set of parameterized actions.
- The specific modeling is as follows.
- $s_t = (et, \eta_{t-1}, D_v, p_v, \xi_t) \in S$, where p_t is the unit cost of spot instance in t , and ξ_t is the remaining usage time of reserved instance. When the edge node does not use this type of cloud service or it expires, this value is 0.
- $a_t = (x_e, (k, x_k)) \in A$, where $K = \{k_1, k_2, k_3\}$ is the set of all discrete actions, k_1 is the on-demand instance, k_2 is the reserved instance, and k_3 is the spot instance.
- $r_t = -C_t^{pri}$ is the reward in each time slot.

3.Resource Allocation Based on Deep Deterministic Policy Gradient

- The DDPG algorithm is the classical algorithm of the ActorCritic algorithm
- Actor generates actions based on policies and interacts with the environment
- Critic evaluates Actor's performance through a value function that guides Actor's next action
- This improves its convergence and performance.

DDPG introduces the idea of DQN and contains four networks, where the main Actor network selects the appropriate action a , according to the current state, s and interacts with the environment:

$$a = \pi_{\theta}(S) + \mathcal{N}$$

where \mathcal{N} is the added noise

For the Critic master network, the loss function is,

$$\nabla J(\omega) = \frac{1}{m} \sum_{j=1}^m \left(y_j - Q(\phi(s_j), a_j, \omega) \right)^2 \quad (1)$$

Where y_j is target Q value , calculated as ,

$$y_j = r_j + \gamma Q' \left(\phi(s'_j), \pi_{\theta} \left(\phi(s'_j) \right), (\omega') \right) \quad (2)$$

3.Resource Allocation Based on Deep Deterministic Policy Gradient

For the Actor master network, the loss function is:

$$\nabla J(\theta) = \frac{1}{m} \sum_{j=1}^m \nabla_a Q(s_i, a_i, \omega)|_{s=s_i, a=\pi_\theta(s)} \nabla_a \pi_\theta(s)|_{s=s_i} \quad (3)$$

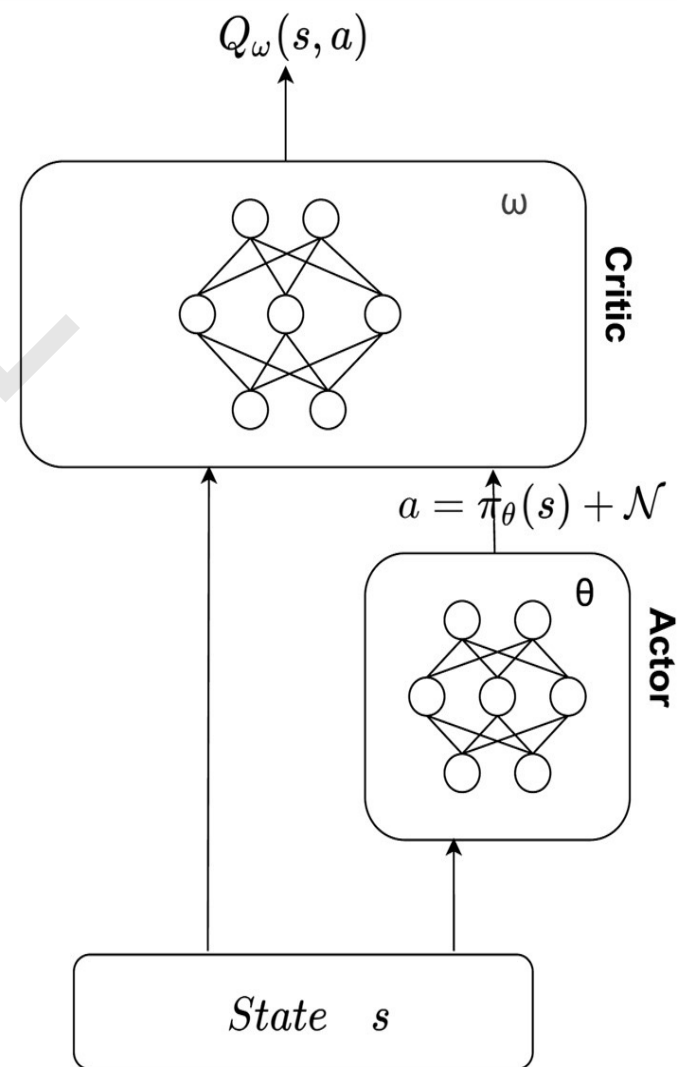
The parameters ω of the Actor target network and the parameters θ of the Critic target network are updated using a soft update:

$$\begin{aligned} \omega &\leftarrow \tau \omega + (1 - \tau) \omega' \\ \theta &\leftarrow \tau \theta + (1 - \tau) \theta' \end{aligned} \quad (4)$$

Resource Allocation Algorithms

3. Resource Allocation Based on Deep Deterministic Policy Gradient

- DDPG structure is shown in figure
- Input of the algorithm contains information about the user requests demands \mathbf{D}_t and the unit cost of VMs in private cloud
- At beginning of each iteration, the edge node first obtains state \mathbf{s}_t of the collaborative cloud-edge environment
- It then pass the state as the input of the neural network into the main Actor network to obtain the action \mathbf{a}_t .
- After the edge node gets the action, the number of demands to be processed by the edge node and the number of demands to be processed by the private cloud will be calculated by the action value, i.e., d_t^e and d_t^c , respectively.
- Then, interaction with the environment based on d_t^e and d_t^c , to get the next state, reward, and termination flag.
- Storing this round of experience to the experience replay pool
- CERA will sample from the experience replay pool and calculate the loss functions of Actor and Critic to update the parameters of the master and target networks.
- After one round of iterative, the training will be continued to the maximum number of training rounds set to ensure the convergence of the resource allocation policy.



CERAI(Cost efficient resource allocation with private cloud) Algorithm

1. **Initialize** Actor main network and target network parameters θ, θ' Critic main network and target network parameters ω, ω' , . soft update coefficient τ . number of samples for batch gradient descent m , maximum number of iterations M , random noise \mathcal{N} and experience replay pool K
2. For $i = 1$ to M do
3. Receive user task information and obtain the status s of collaborative cloud-edge computing environment;
4. Actor main network selects actions according to s : $a = \pi_{\theta}(S) + \mathcal{N}$;
5. The edge node performs action a and obtains the next status s' , reward r and termination flag $isend$
6. The edge node generates an allocation record h_i according to the allocation operation. Add it to the allocation record H ;
7. Add the state transition tuple $(s, a, r, s', isend)$ in the experience replay pool K ;
8. Update status: $s = s'$;
9. Sample m samples from experience replay pool P calculate the target Q value y according to the eq 2;
10. Calculate the loss function according to (1) and update the parameters of the Critic main network;
11. Calculate the loss function according to (3) and update the parameters of the Actor main network;
12. update the parameters of the Critic and Actor target network according to (4)
13. Update allocation record H and release computing resources for completed tasks;
14. If s' is terminated, complete the current round of iteration, otherwise goto step 3;
15. end.

4. Resource Allocation Based on P-DQN

The basic idea of P-DQN is as follows.

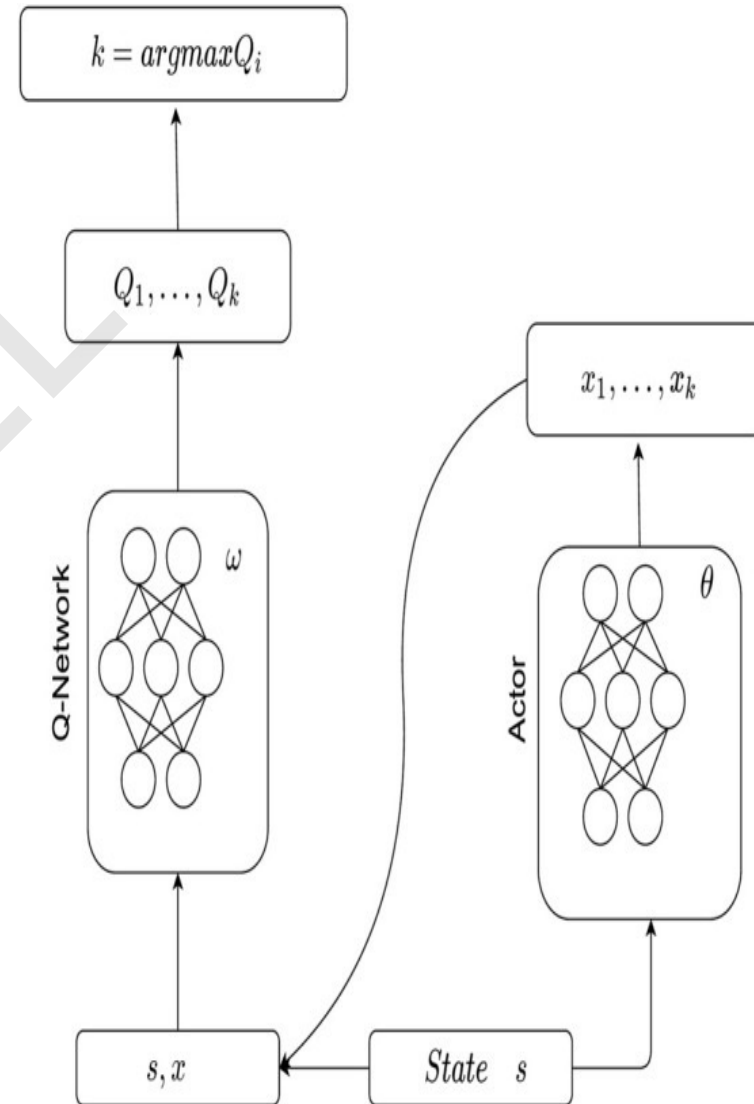
- For each action $a \in A$ in the parametric action space, because of $\mathbf{x}_e + \mathbf{x}_k = \mathbf{1}$, we can only consider k and \mathbf{x}_k in the action value function, that is $Q(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, k, \mathbf{x}_k)$, where $\mathbf{s} \in S$, $k \in K$ is the discrete action selected in the time slot t , and $\mathbf{x}_k \in \mathbf{X}_k$ is the parameter value corresponding to k .
- Similar to DQN, deep neural network $Q(\mathbf{s}, k, \mathbf{x}_k; \omega)$ is used in P-DQN to estimate $Q(\mathbf{s}, k, \mathbf{x}_k)$, where ω is the neural network parameter.
- For $Q(\mathbf{s}, k, \mathbf{x}_k; \omega)$, P-DQN uses the determined policy network $\mathbf{x}_k(\cdot; \theta): S \rightarrow \mathbf{X}_k$ to estimate the parameter value $\mathbf{x}_k^Q(\mathbf{s})$, where θ is used to represent the policy network. That means the goal of P-DQN is to find the corresponding parameters θ , when ω is fixed. It can be written as the following

$$Q(\mathbf{s})^k, \mathbf{x}_k(\mathbf{s}; \theta); \omega \approx Q(\mathbf{s}, k, \mathbf{x}_k; \omega) \quad (5)$$

- Similar to DQN, the value of ω can be obtained by minimizing the mean square error by gradient descent.
- In particular, step t , ω_t and θ_t are the parameters of value network and deterministic policy network, respectively.
- y_t can be written as :

$$y = r + \max_{k \in [k]} Q(\mathbf{s}; k, \mathbf{x}_k(\mathbf{s}', \theta_t); \omega_t) \quad (6)$$

where \mathbf{s}' is the next state after taking the mixed action $\mathbf{a} = (k, \mathbf{x}_k)$.



4. Resource Allocation Based on P-DQN

The loss function of value network can be written as the following:

$$l^Q(\omega) = \frac{1}{2} [Q(s, k, x_k; \omega) - y]^2 \quad (7)$$

loss function of a policy network can be written as

$$l^\theta(\theta) = - \sum_{k=1}^K \mu(s, k, x_k(s; \theta); \omega) \quad (8)$$

- P-DQN structure is shown in Figure .
- Cost Efficient Resource Allocation with public cloud (CERAU) is a resource allocation algorithm based on P-DQN,. The input of the algorithm contains information about the user requests demands D_t and the unit cost of spot instance in public cloud in time slot t p_t .
- At the beginning of each iteration of the algorithm, the edge node first needs to obtain the state s_t of the collaborative cloud-edge environment
- Then pass the state as the input of the neural network into the strategy network to obtain the parameter values of each discrete action.
- After the edge node gets the action, it will select the appropriate public cloud instance type based on the discrete values in the action and determine the number of public cloud instances to be used based on the parameter values.
- Then, interaction with the environment occurs, to get the next state, reward, and termination flag.
- Storing this round of experience to the experience replay pool, CERAU will sample from the experience replay pool and calculate the gradient of the value network and the policy network.
- Then, it will update the parameters of the corresponding networks.
- After one round of iterative, to ensure the convergence of the resource allocation policy, the training will be continued to the maximum number of training rounds set.

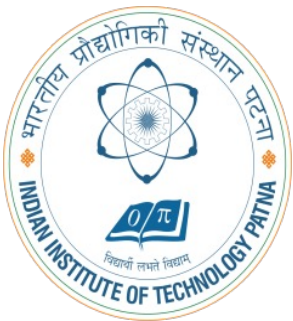
CERAU Algorithm

Algorithm: Cost efficient resource allocation with public cloud (CERAU)

1. Initialize exploration parameters ϵ , soft update coefficient τ_1 and τ_2 , number of samples for batch gradient descent m , maximum number of iterations M , random noise \mathcal{N} and experience replay pool P ;
2. for $i = 1$ to M do
3. Receive user task information and obtain the status s of collaborative cloud-edge computing environment;
4. Calculate the parameter value of each instance type in the cloud service; $x_k \leftarrow x_k(s_t, \theta_t) + \mathcal{N}$;
5. Selects discrete actions according to ϵ –greedy strategy:
$$a = \begin{cases} \text{random discrete action, } rnd > \epsilon \\ (k, x_k), k = \operatorname{argmax}_{k \in [K]} Q(s, k, x_k; \omega), rnd \geq \epsilon \end{cases}$$
6. The edge node performs action and obtains the next status s' , reward r and termination flag $isend$;
7. The edge node generates an allocation record h_i according to the allocation operation. Add it to the allocation record list H ;
8. Add the state transition tuple $(s, a, r, s', isend)$ in the experience replay pool P ;
9. Sample m samples from experience replay pool P , calculate the target Q value y according to (6);
10. Update status: $s = s'$;
11. Calculate gradient $\nabla_{\omega} l^Q(\omega)$ and $\nabla_{\theta} l^{\theta}(\theta)$ according to (7) and (8);
12. Update network parameters: $\omega' \leftarrow \omega - \tau_1 \nabla_{\omega} l^Q(\omega), \theta' \leftarrow \theta - \tau_2 \nabla_{\theta} l^{\theta}(\theta)$
13. Update allocation record H and release computing resources for completed tasks;
14. If s' is terminated, complete the current round of iteration. otherwise go to step 3;
15. end

Thank You

Deep Reinforcement Learning for Cloud-Edge: Example



Dr. Rajiv Misra

Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss how Collaborative cloud-edge approaches can provide better performance and efficiency than traditional cloud or edge approaches with the help of some examples

Example

Consider the resource allocation problem where a client submits the following demands in three consecutive time slots:

Time-Slot (t)	Demand $D_t = (d_t, l_t)$
1	(30, 2)
2	(10, 1)
3	(20, 2)

where (d_t) represents the number of VMs requested and (l_t) represents the duration of service request. Assume that time slot (1) is the starting slot such that no VMs have been allocated a priori. There are 80 VMs available at the edge node.

Example

(a) Resource allocation using private cloud: Suppose that we have our own private cloud and a policy has been deployed to allocate VMs as per client demands which outputs the following actions at each timeslot:

Time-Slot (t)	Policy Action (x_t^k)
1	0.4
2	0.7
3	0.8

The action ($x_t^k \in [0, 1]$) represents the ratio of VMs allocated from the private cloud to the total VMs requested by client at time slot t . The remaining VMs ($1 - x_t^k$) are allocated from the edge node.

Calculate the cost of collaborative cloud side computing (C_t^{pri}) in the given private cloud setting at each of the three time slots. Also, find out the number of VMs that will be available at the edge node at the beginning of fourth time slot

Given Constants:

Constant	Value
Stand-by cost of a VM at the edge node (p_e)	0.03
Computing cost of a VM at the edge node (p_f)	0.20
Computing cost of a private cloud (p_c)	3.00

Example

(b) Resource allocation using public cloud: Assume that we have replaced the private cloud with a public cloud setting with a new policy that outputs the following actions at each timeslot:

Time-Slot (t)	Policy Action (k_t, x_t^k)
1	(1,0.4)
2	(0,0.7)
3	(2,0.8)

where ($k_t \in \{0=\text{on_demand}, 1=\text{reserved}, 2=\text{spot}\}$) represents the type of public cloud instance that was allocated. Calculate the cost of collaborative cloud side computing (C_t^{pub}) in the given public cloud setting at each of the three time slot. Assume that the same demands were made by client as in part (a) and that no customization is performed on reserved instances.

Additional Constants:

Constant	Value
Unit price of on-demand instance in public cloud (p_{od})	3.0
Unit price of reserved instance in public cloud (p_{re})	1.5
Customization price of reserved instance ($p_{upfront}$)	800
Unit price of spot instance in public cloud (p_t)	1.0

Example : Solution

Let (e_t) represent the number of VMs available at the edge node after allocation at time slot (t) .

Assume $e_1 = E = 80$

At time slot $(t = 1)$:

Demand: $D_1 = (d_1, l_1) = (30, 2)$

Action: $x_1^k = 0.4$

No of VMs allocated from cloud: $d_1^c = x_1^k * d_1 = 0.4 * 30 = 12$

No of VMs allocated from edge node: $d_1^e = d_1 - d_1^c = 30 - 12 = 18$

No of VMs remaining at the edge node: $e_1 = e_1 - d_1^e = 80 - 18 = 62$

Resources can be successfully allocated from edge node; hence, allocation record will be generated:

Allocation record: $h_1 = (d_1^e, l_1) = (18, 2)$

Allocation Record List H : $\langle h_1 \rangle : \langle (18, 2) \rangle$

Updated Allocation Record List H : $\langle h_1 \rangle : \langle (18, 1) \rangle$

Number of VMs waiting to be released: $n_1 = 0$

Number of VMs available at next time slot: $e_2 = e_1 + n_1 = 62 + 0 = 62$

Cost at the edge node: $C_1^e = e_1 p_e + (E - e_1) p_f = 62 * 0.03 + (80 - 62) * 0.2 = 1.86 + 3.6 = 5.46$

Cost at the private cloud: $C_1^{pri} = d_1^c p_c + C_1^e = 12 * 3.0 + 5.46 = 41.46$

Cost at the public cloud: $C_1^{pub} = d_1^c p_{re} + C_1^e = 12 * 1.5 + 5.46 = 23.46$

Example : Solution

At time slot ($t = 2$):

$$\text{Demand: } D_2 = (d_2, l_2) = (10, 1)$$

$$\text{Action: } x_2^k = 0.7$$

$$\text{No of VMs allocated from cloud: } d_2^c = x_2^k * d_2 = 0.7 * 10 = 7$$

$$\text{No of VMs allocated from edge node: } d_2^e = d_2 - d_2^c = 10 - 7 = 3$$

$$\text{No of VMs remaining at the edge node: } e_2 = e_2 - d_2^e = 62 - 3 = 59$$

Resources can be successfully allocated from edge node; hence, allocation record will be generated:

$$\text{Allocation record: } h_2 = (d_2^e, l_2) = (3, 1)$$

$$\text{Allocation Record List } H: < h_1, h_2 > : < (18, 1), (3, 1) >$$

$$\text{Updated Allocation Record List } H: < h_1, h_2 > : < (18, 0), (3, 0) >$$

$$\text{Number of VMs waiting to be released: } n_2 = 18 + 3 = 21$$

$$\text{Number of VMs available at next time slot: } e_3 = e_2 + n_2 = 59 + 21 = 80$$

$$\text{Cost at the edge node: } C_2^e = e_2 p_e + (E - e_2) p_f = 59 * 0.03 + (80 - 59) * 0.2 = 1.77 + 4.2 = 5.97$$

$$\text{Cost at the private cloud: } C_2^{pri} = d_2^c p_c + C_2^e = 7 * 3.0 + 5.97 = 26.97$$

$$\text{Cost at the public cloud: } C_2^{pub} = d_2^c p_{od} + C_2^e = 7 * 3.0 + 5.97 = 26.97$$

Example : Solution

At time slot ($t = 3$):

$$\text{Demand: } D_3 = (d_3, l_3) = (20, 2)$$

$$\text{Action: } x_3^k = 0.8$$

$$\text{No of VMs allocated from cloud: } d_3^c = x_3^k * d_3 = 0.8 * 20 = 16$$

$$\text{No of VMs allocated from edge node: } d_3^e = d_3 - d_3^c = 20 - 16 = 4$$

$$\text{No of VMs remaining at the edge node: } e_3 = e_3 - d_3^e = 80 - 4 = 76$$

Resources can be successfully allocated from edge node; hence, allocation record will be generated:

$$\text{Allocation record: } h_2 = (d_3^e, l_3) = (4, 2)$$

$$\text{Allocation Record List } H: < h_3 > : < (4, 2) >$$

$$\text{Updated Allocation Record List } H: < h_3 > : < (4, 1) >$$

$$\text{Number of VMs waiting to be released: } n_3 = 0$$

$$\text{Number of VMs available at next time slot: } e_4 = e_3 + n_2 = 76 + 0 = 76$$

$$\text{Cost at the edge node: } C_3^e = e_3 p_e + (E - e_3) p_f = 76 * 0.03 + (80 - 76) * 0.2 = 2.28 + 0.8 = 3.08$$

$$\text{Cost at the private cloud: } C_3^{pri} = d_3^c p_c + C_3^e = 16 * 3.0 + 3.08 = 51.08$$

$$\text{Cost at the public cloud: } C_3^{pri} = d_3^c p_t + C_3^e = 16 * 1.0 + 3.08 = 19.08$$

Thank You

Public Cloud Services: Case Study of AWS Services



Dr. Rajiv Misra

Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

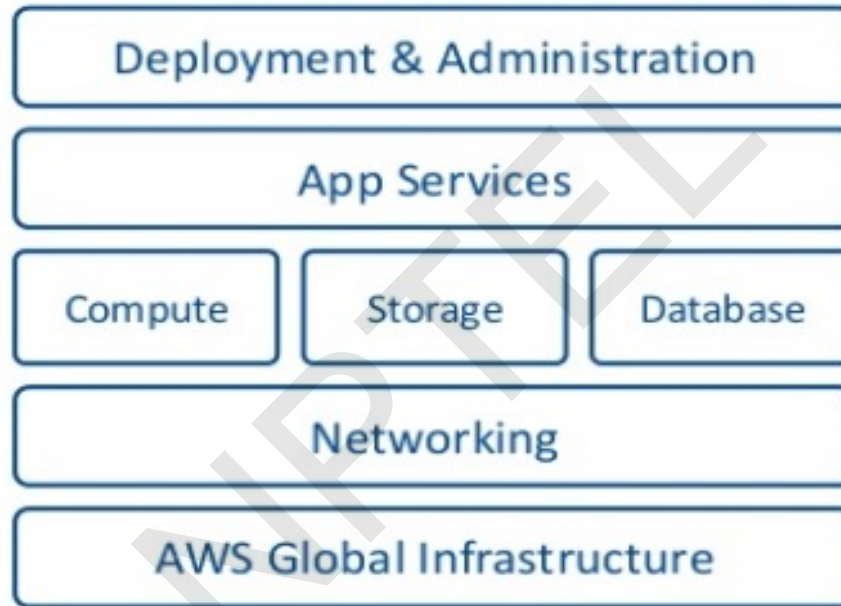
Contents of lecture

In this lecture, we will cover a Public Cloud Services,
a case study of AWS services

NPTEL

Reference Model

We will use a reference model to explain AWS services systematically as 5-layered model



Global Infrastructure



Regions

An independent collection of AWS resources in a defined geography

A solid foundation for meeting location-dependent privacy and compliance requirements

Availability Zones

Designed as independent failure zones

Physically separated within a typical metropolitan region

Edge Locations

To deliver content to end users with lower latency

A global network of edge locations

Supports global DNS infrastructure (Route53) and Cloud Front CDN

Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Global Footprint

AWS is available today in the U.S., Brazil, Europe, Japan, Singapore, Australia, and China. Additional regions in the UK, Canada, China, and Ohio are expected to come online over the next 12 – 18 months.

Over 1 million **active** customers per month across 190 countries

2,300 government agencies

7,000 educational institutions

13 regions

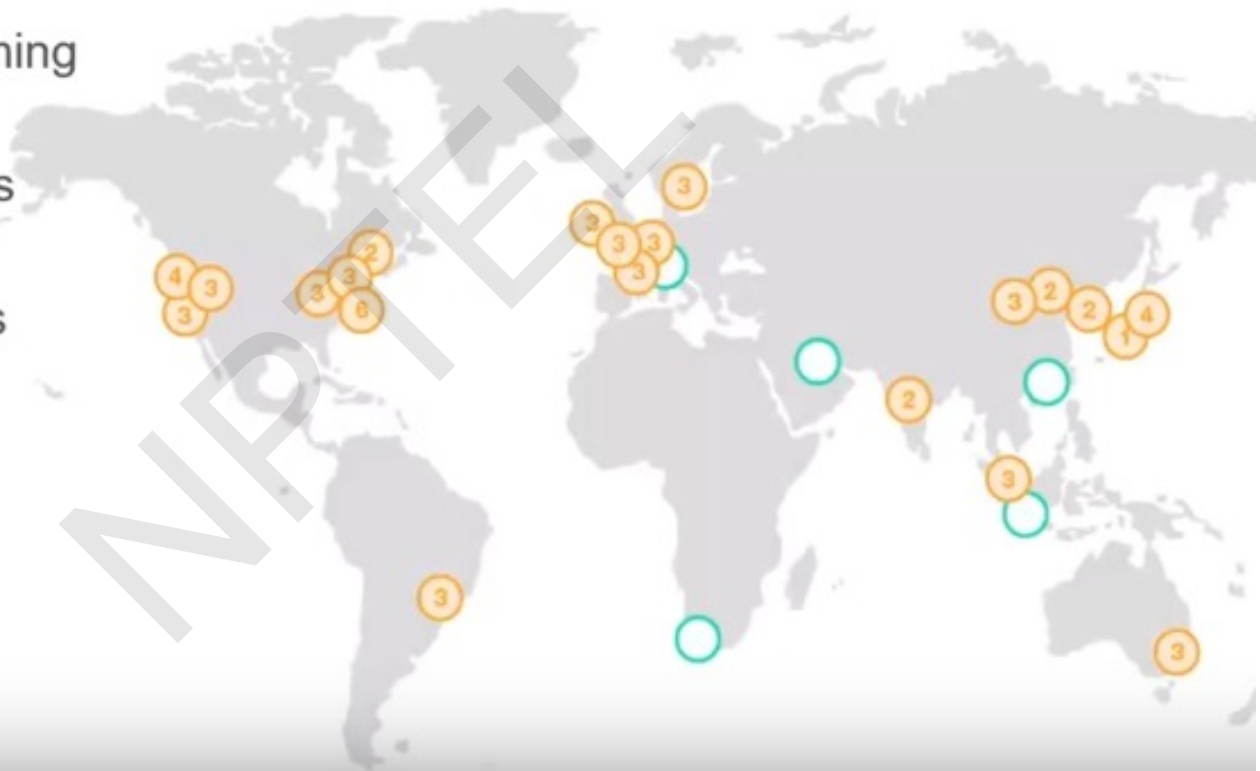
35 availability zones +
4 more coming soon

56 edge locations



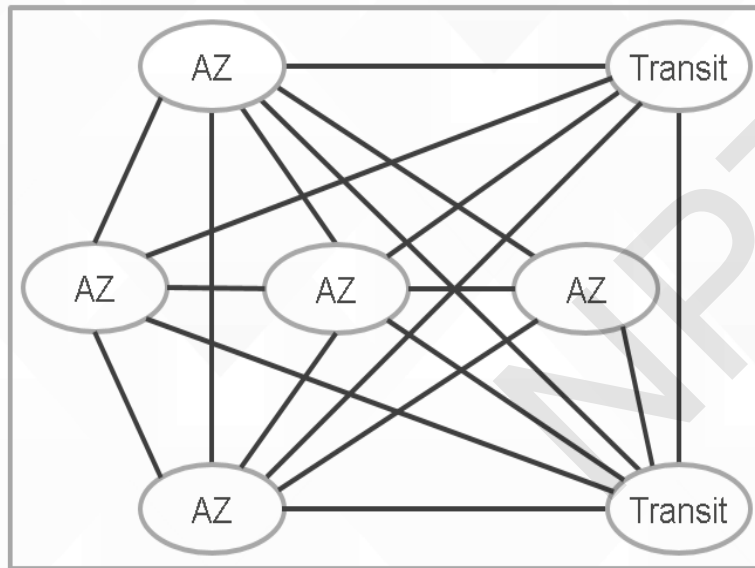
AWS Global Datacenter


- 20 Regions + 5 Coming soon
- 61 Availability Zones
- 158 Edge locations
- 11 Regional Caches
- 130 + Services



Region

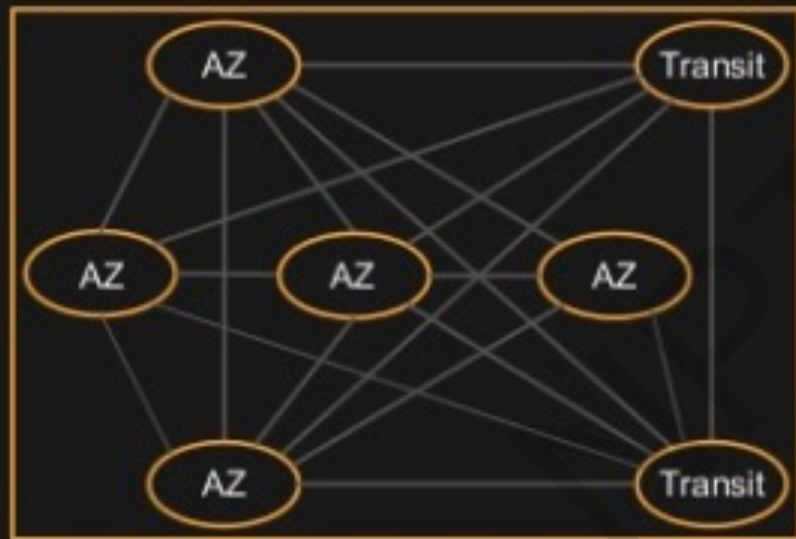
Region



- 
- Geographic area where AWS services are available
 - Customers choose region(s) for their AWS resources
 - Eleven regions worldwide

AWS Region

Example AWS Region



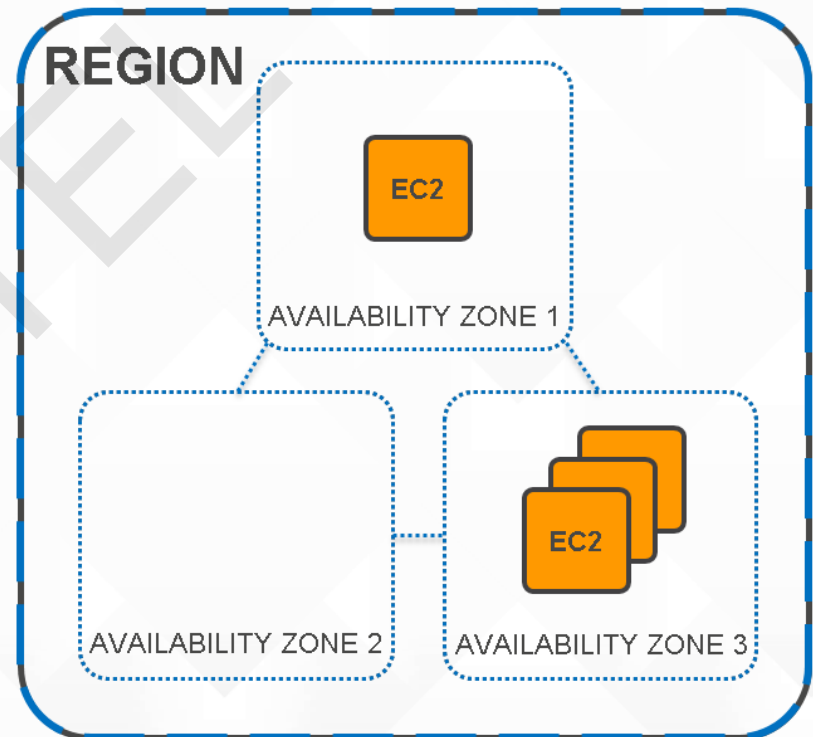
- 1 of 13 AWS worldwide AWS regions
- Redundant paths to transit centers
- Transit centers connect to:
 - Private links to other AWS Regions
 - Private links to AWS Direct Connect customers
 - Internet through peering & paid transit
- Metro-area DWDM links between AZs
- 82,864 fiber strands in region
- AZs <2 ms apart & usually <1 ms
- 25 Tbps peak inter-AZs traffic



Availability Zone(AZ)

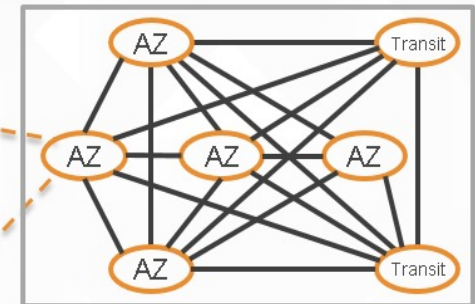
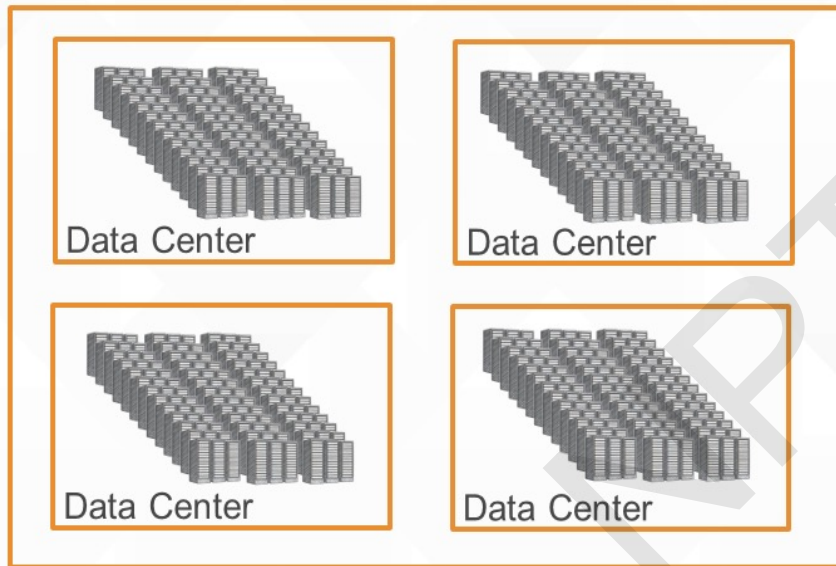
Availability Zone (AZ)

- Each region has multiple, isolated locations known as Availability Zones
- Low-latency links between AZs in a region $<2\text{ms}$, usually $<1\text{ms}$
- When launching an EC2 instance, a customer chooses an AZ
- Private AWS fiber links interconnect all major regions



Availability Zone(AZ): Example

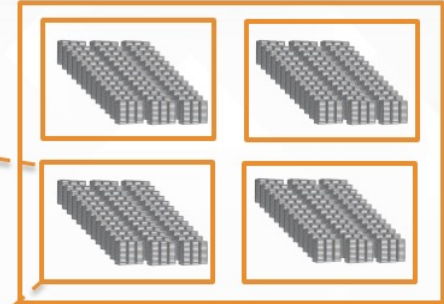
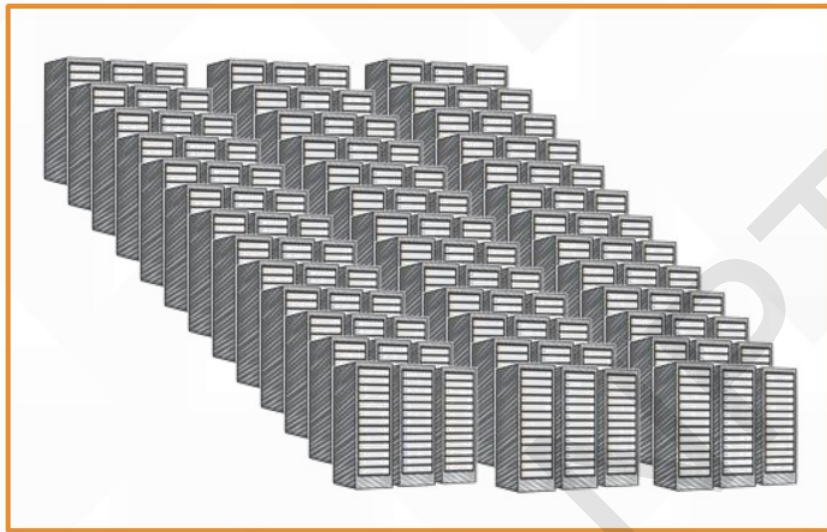
Example AWS Availability Zone



- 1 of 28 AZs world-wide
- All regions have 2 or more AZs
- Each AZ is 1 or more DC
 - No data center is in two AZs
 - Some AZs have as many as 6 DCs
- DCs in AZ less than $\frac{1}{4}$ ms apart

AWS Data Center: Example

Example AWS Data Center

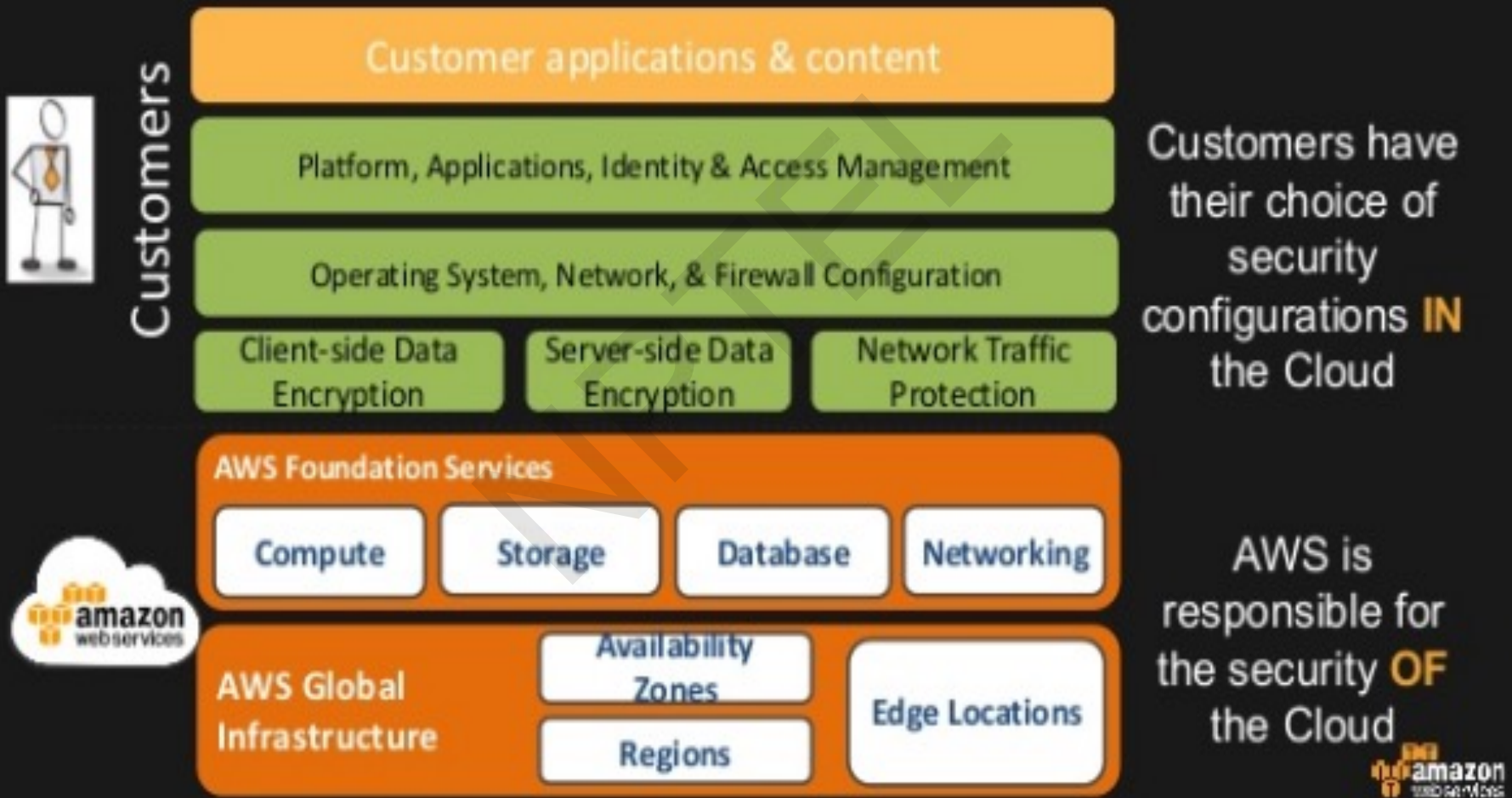


- Single DC typically over 50,000 servers & often over 80,000
 - Larger DCs undesirable (blast radius)
- Up to 102Tbps provisioned to a single DC
- AWS custom network equipment:
 - Multi-ODM sourced
 - Amazon custom network protocol stack



AWS Security

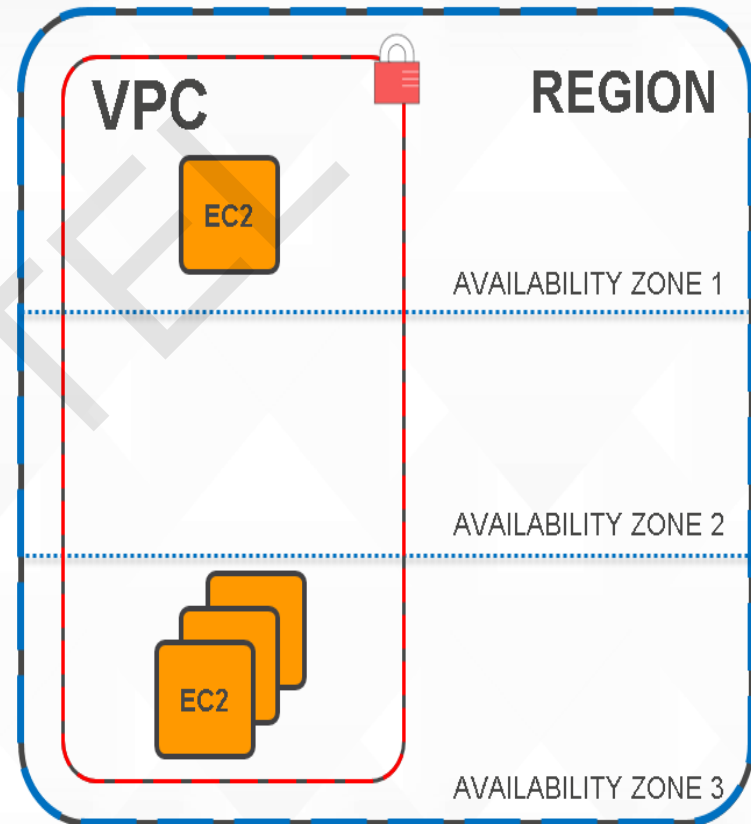
Security is **shared** between AWS and its customers



Virtual Private Cloud(VPC)

Virtual Private Cloud (VPC)

- Logically isolated section of the AWS cloud, virtual network defined by the customer
- When launching instances and other resources, customers place them in a VPC
- All new customers have a default VPC



Networking

Networking



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Direct Connect

Dedicated connection to AWS

VPN Connection

Secure internet connection to AWS

Virtual Private Cloud

Private, isolated section of the AWS Cloud

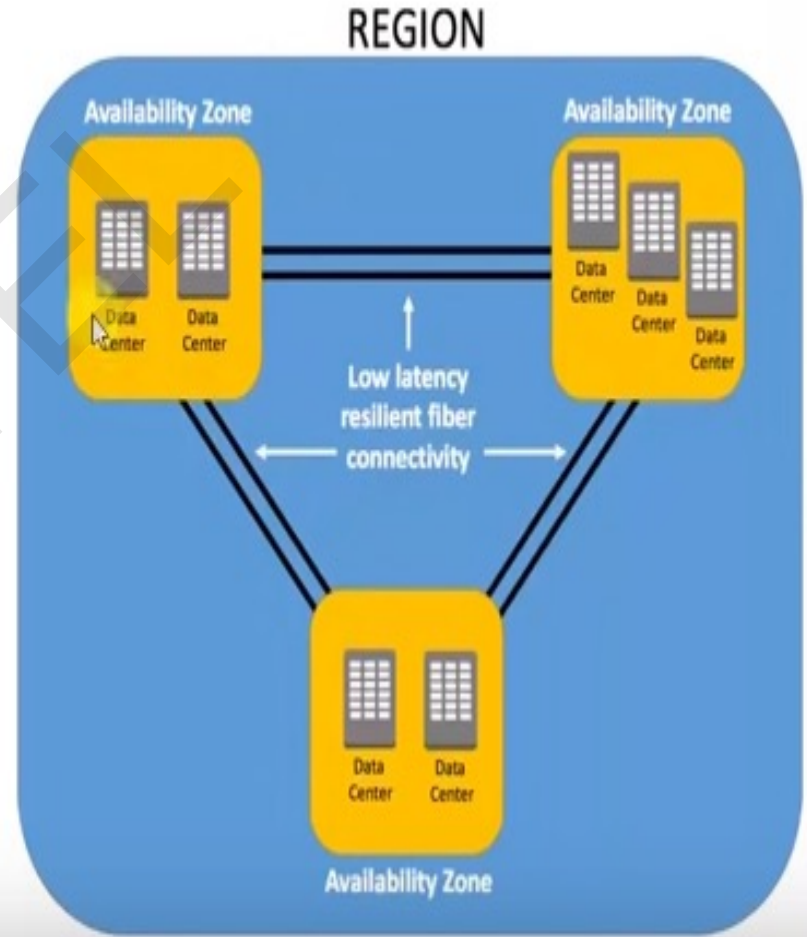
Route 53

Highly available and scalable Domain Name System

Region and Availability Zones

Region = Independent Geographic Area

Availability Zone = Multiple isolated locations / data centers within a region



Why Availability Zones

Why Availability Zones?

Challenges with traditional asynchronous replication between distant data centers

- Committing to an SSD order 1 to 2 ms
- But Sydney to Melbourne is 20 ms
- You can't wait 20 ms to commit a transaction

Traditional failure, difficult decision:

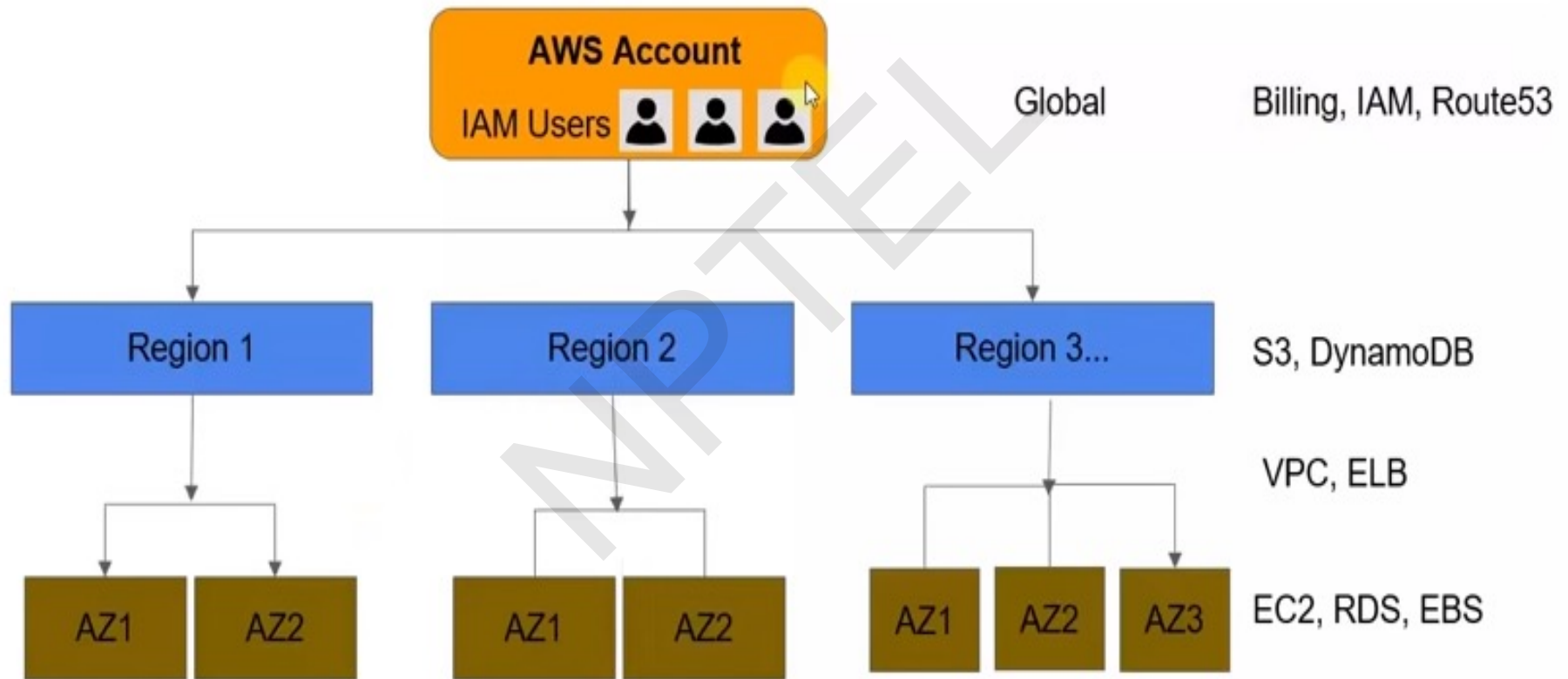
- Failover & lose transactions, or
- Or don't failover & lose availability
- Difficult choice

AZs for no-admin failover

- Sync works when < 2 ms
- Combine with regional replication for very high availability (VHA)



AWS Account, Users and Service scope



AWS Compute and Analytics Services



EC2
Elastic Compute Cloud



Auto Scaling
EC2 Horizontal scaling



Lambda
Serverless Computing



ELB
Elastic Load Balancer



ECS
Elastic Container Service



EMR
Elastic Mapreduce



Kinesis
Real time data/video streaming



Athena
Interactive Query Engine



QuickSight
Business Intelligence



Glue
Elastic Data Integration

Compute Services

Clip slide

Compute



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Elastic Compute Cloud (EC2)

Basic unit of compute capacity

Range of CPU, memory & local disk options

13 instance types available, from micro to cluster compute

Security Group

Auto-scaling

Automatic re-sizing of compute clusters based upon demand

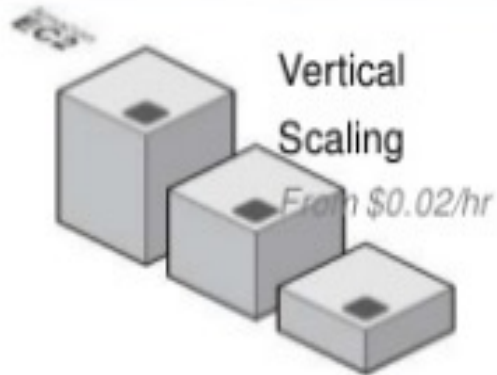
Elastic Load Balancing

Create highly scalable applications

Distribute load across EC2 instances in multiple availability zones

Elastic Compute Cloud (EC2)

Compute



Elastic Compute Cloud (EC2)

Basic unit of compute capacity

Range of CPU, memory & local disk options

13 Instance types available, from micro to cluster compute



Feature	Details
Flexible	Run windows or linux distributions
Scalable	Wide range of instance types from micro to cluster compute
Machine Images	Configurations can be saved as machine images (AMIs) from which new instances can be created
Full control	Full root or administrator rights
Secure	Full firewall control via Security Groups
Monitoring	Publishes metrics to Cloud Watch
Inexpensive	On-demand, Reserved and Spot instance types
VM Import/Export	Import and export VM images to transfer configurations in and out of EC2

Compute : Auto-scaling

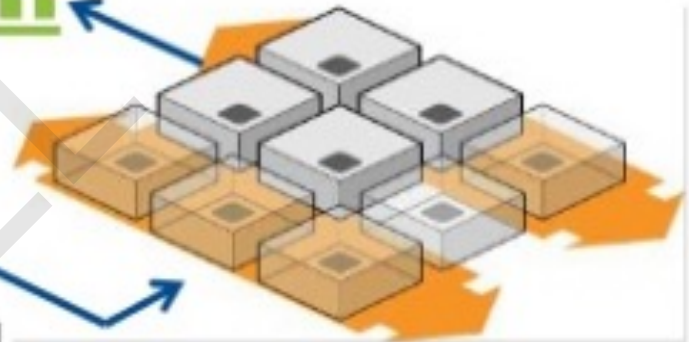
Compute

Clip slide



Trigger auto-scaling policy

```
as-create-auto-scaling-group MyGroup
--launch-configuration MyConfig
--availability-zones eu-west-1a
--min-size 4
--max-size 200
```



Auto-scaling

Automatic re-sizing of compute clusters based upon demand

Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Feature

Details

Control

Define minimum and maximum instance pool sizes and when scaling and cool down occurs

Integrated to CloudWatch

Use metrics gathered by CloudWatch to drive scaling

Instance types

Run auto scaling for on-demand instances and spot. Compatible with VPC

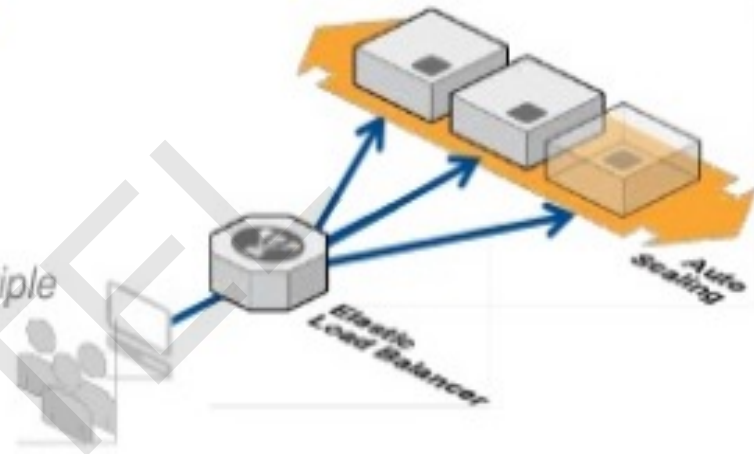
Compute : Elastic Load Balancing

Compute

Elastic Load Balancing

Create highly scalable applications

Distribute load across EC2 instances in multiple availability zones



Feature	Details
Auto-scaling	Automatically scales to handle request volume
Available	Load balance across instances in multiple availability zones
Health checks	Automatically checks health of instances and takes them in or out of service
Session stickiness	Route requests to the same instance
Secure sockets layer	Supports SSL offload from web and application servers with flexible cipher support
Monitoring	Publishes metrics to Cloud Watch

Storage

Storage



S3 - Durable storage, any object

99.999999999% durability of objects

Unlimited storage of objects of any type

Up to 5TB size per object

Elastic Block Store

High performance block storage device

1GB to 1TB in size

Mount as drives to instances

Deployment & Administration

App Services

Compute

Storage

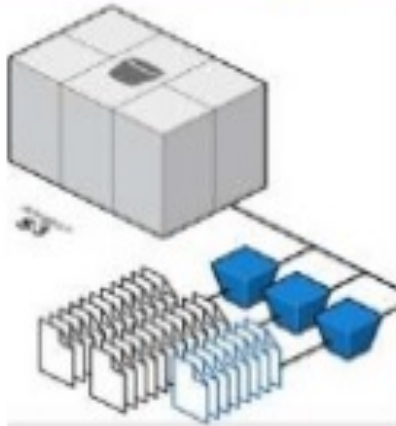
Database

Networking

AWS Global Infrastructure

Storage

Storage



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

S3 - Durable storage, any object

99.999999999% durability of objects

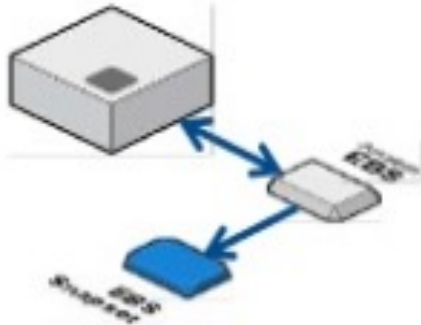
Unlimited storage of objects of any type

Up to 5TB size per object

Feature	Details
Flexible object store	Buckets act like drives, folder structures within
Access control	Granular control over object permissions
Server-side encryption	256bit AES encryption of objects
Multi-part uploads	Improved throughput & control
Object versioning	Archive old objects and version new ones
Object expiry	Automatically remove old objects
Access logging	Full audit log of bucket/object actions
Web content hosting	Serve content as web site with built in page handling
Notifications	Receive notifications on key events
Import/Export	Physical device import/export service

Storage

Storage



Elastic Block Store

High performance block storage device

1GB to 1TB in size

Mount as drives to instances



Feature	Details
High performance file system	Mount EBS as drives and format as required
Flexible size	Volumes from 1GB to 1TB in size
Secure	Private to your instances
Available	Replicated within an Availability Zone
Backups	Volumes can be snapshotted for point in time restore
Monitoring	Detailed metrics captured via Cloud Watch

Database

Database



Relational Database Service

Database-as-a-Service

No need to install or manage database instances

Scalable and fault tolerant configurations



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

DynamoDB

Provisioned throughput NoSQL database

Fast, predictable performance

Fully distributed, fault tolerant architecture

Application Services

Application services

Clip slide



Deployment & Administration

App Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Amazon SQS

Reliable, highly scalable, queue service for storing messages as they travel between instances

Simple Workflow

*Reliably coordinate processing steps across applications
Integrate AWS and non-AWS resources
Manage distributed state in complex systems*

Amazon SES

Amazon-grade Web Service for Sending High-Quality High-Volume (HQHV) email

CloudFront

*World-wide content distribution network
Easily distribute content to end users with low latency, high data transfer speeds, and no commitments.*

What is Amazon SES ?



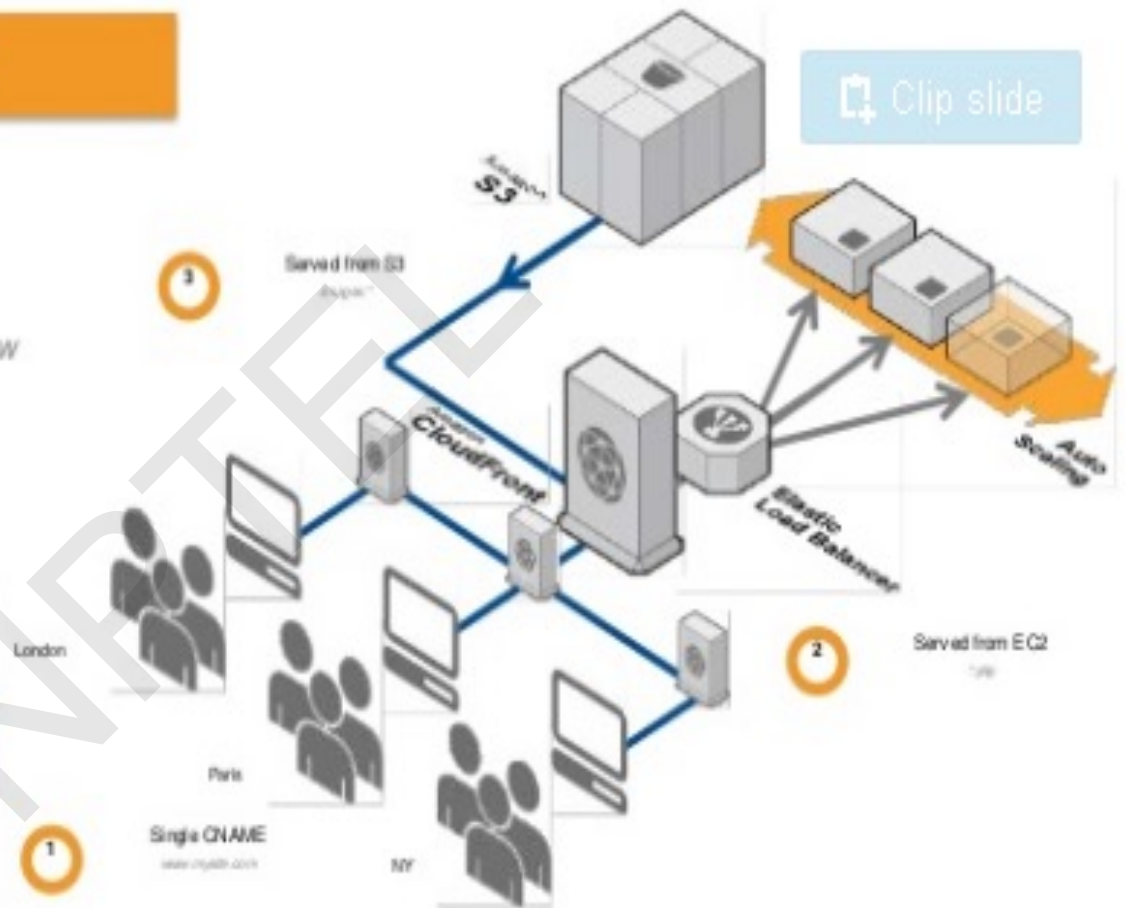
- Amazon-grade Web Service for Sending *High-Quality High-Volume (HQHV)* email
- Our users are free to concentrate on email Strategy and Quality; SES manages the complexity of IP Reputation and Deliverability
- Robust, Hardened API-Driven infrastructure = minimal downtime

Application Services

Application Services

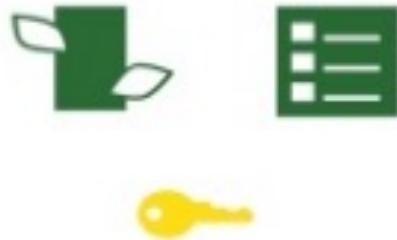
CloudFront

World-wide content distribution network
Easily distribute content to end users with low latency, high data transfer speeds, and no commitments.



Deployment and Administration

Deployment & admin



Elastic Beanstalk

One-click deployment from Eclipse, Visual Studio and Git
Rapid deployment of applications
All AWS resources automatically created

Cloud Formation

Automate creation of 'stacks' in a repeatable way
Scripting framework for AWS resource creation

Identity & Access Management

Granular control of user rights with AWS
Automated granting of EC2 service rights

Software Developer Kits

Comprehensive support of programming models for using AWS services



AWS Storage and Database Services



EBS
Elastic Block Storage



S3
Simple Storage Service



EFS
Elastic File System



RDS
Relational Database Service



DynamoDB
AWS NoSQL Database



Redshift
Data Warehousing



Elasticache
Fast and Flexible caching

AWS Network and Management Services



VPC
Virtual Private Cloud



Route53
AWS DNS Service



Direct Connect
Dedicated Network



CloudFront
Content Delivery Network



CloudWatch
Application & Infrastructure monitoring



CloudFormation
Provision Infrastructure as a Code



Elastic Beanstalk
Application Orchestration Service



Opsworks
Infrastructure Configuration Management

AWS Application and Development Services



API Gateway

Managed REST and Websocket APIs



SQS

Simple Queue Service



SNS

Simple Notification Service



SES

Simple Email Service



CodeCommit

Hosted GIT Repository by AWS



CodeBuild

Continuous Integration Service



CodeDeploy

Automated Deployments

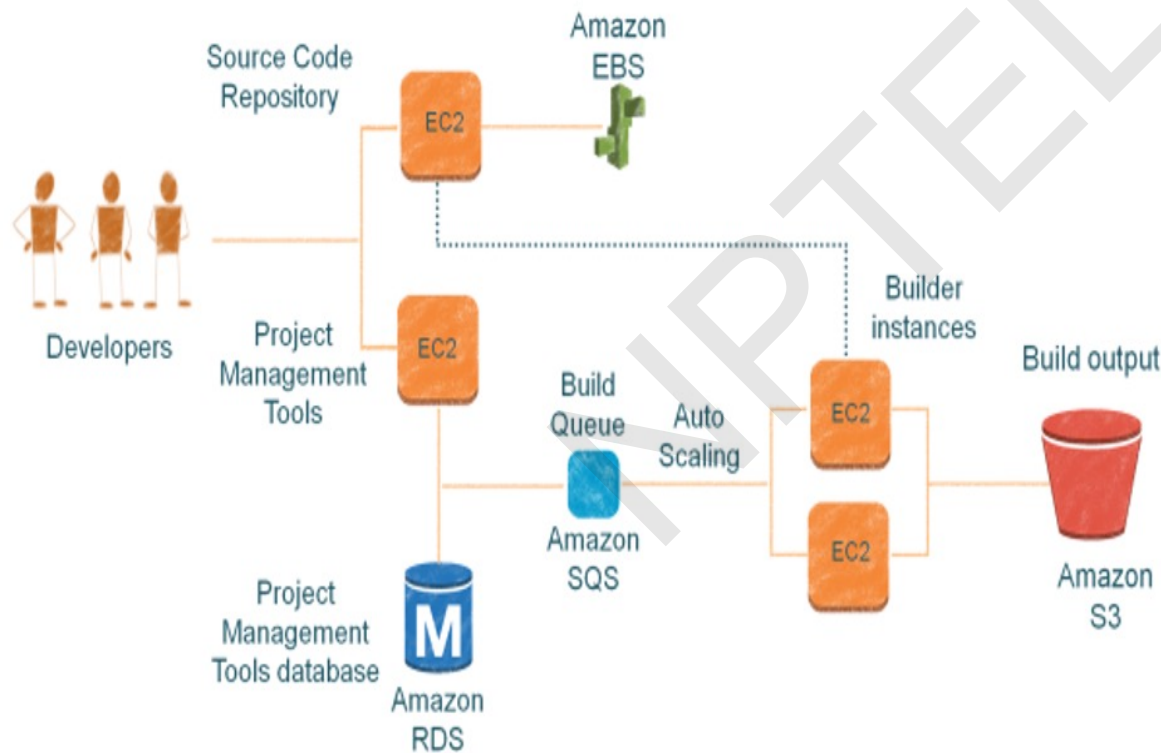


Code Pipeline

Continuous Delivery Service

Development and Test Environments

Development and Test Environments



AWS Dev./Test Scenarios

On-demand Dev. environments
On-Demand Builds
Automated Test environments
Load Testing

Big Data

Big Data

Amazon EMR (Elastic MapReduce)

Managed Hadoop Service

Easily launch, customize, and resize your managed Hadoop cluster



Amazon Redshift

Petabyte-scale data warehouse service



AWS Data Pipeline

More data among AWS Services and on-premises data sources

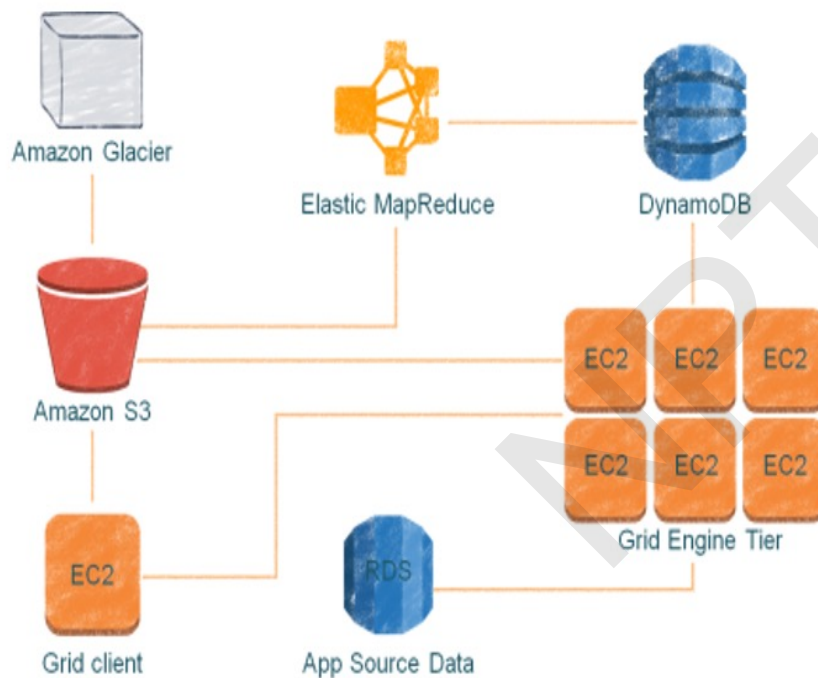


Amazon Kinesis - Fully-managed service for real time processing of streaming data, at any scale.

Plug and play with a simple, pre-built client library
Deploy Amazon Kinesis-enabled applications to Amazon EC2
Integrates with Amazon Redshift, Amazon DynamoDB, Amazon EMR, and Amazon

High Performance Computing (HPC)

High Performance Computing (HPC)



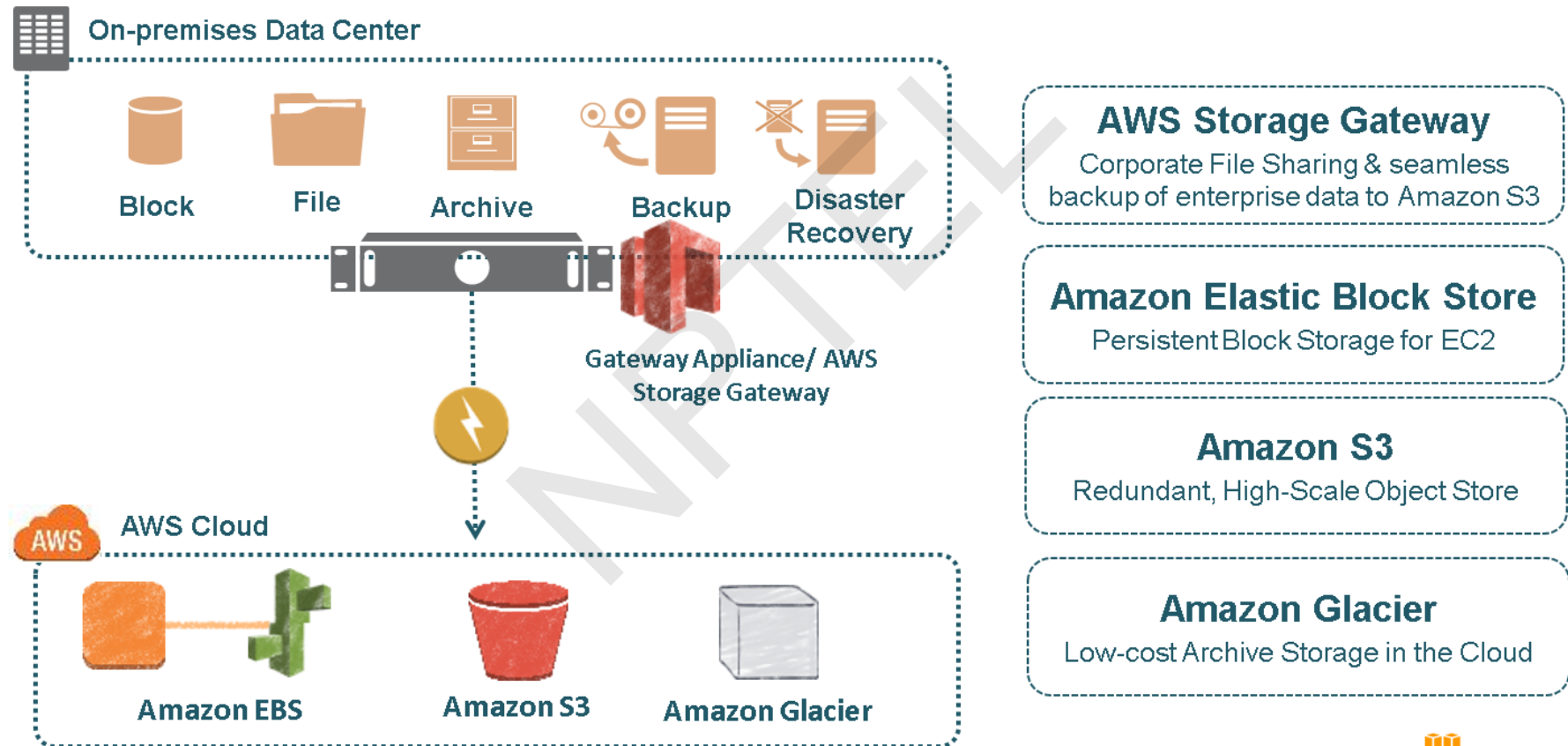
C3 instances

Instance Name	vCPU Count	Total ECU	RAM	Local Storage
c3.large	2	7	3.75 GiB	2 x 16 GB SSD
c3.xlarge	4	14	7 GB	2 x 40 GB SSD
c3.2xlarge	8	28	15 GiB	2 x 80 GB SSD
c3.4xlarge	16	55	30 GiB	2 x 160 GB SSD
c3.8xlarge	32	108	60 GiB	2 x 320 GB SSD

Improved Network Performance
SSD based Platform

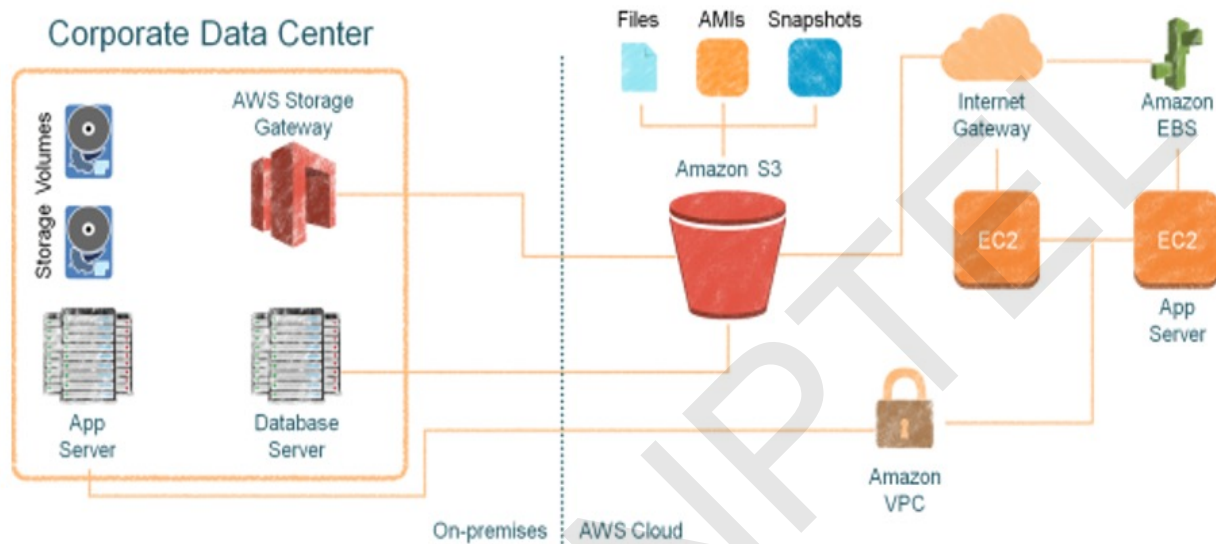
Storage, Backup, and Archival

Storage, Backup, and Archival



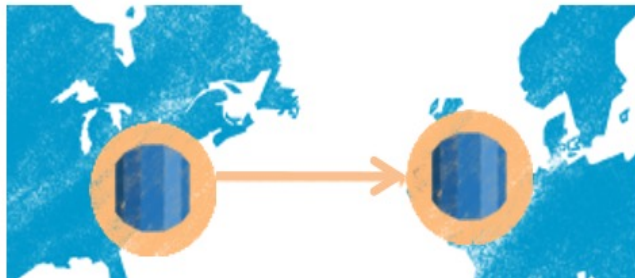
Disaster Recovery

Disaster Recovery



AWS DR Scenarios

Backup and Restore
Pilot Light for Simple
Recovery into AWS
Warm Standby Solution
Multi-site Solution

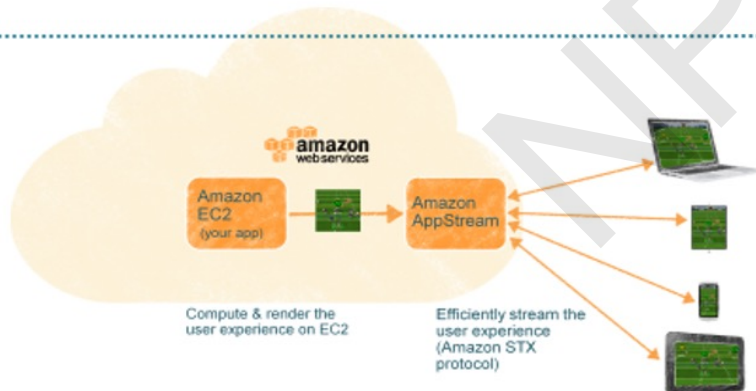
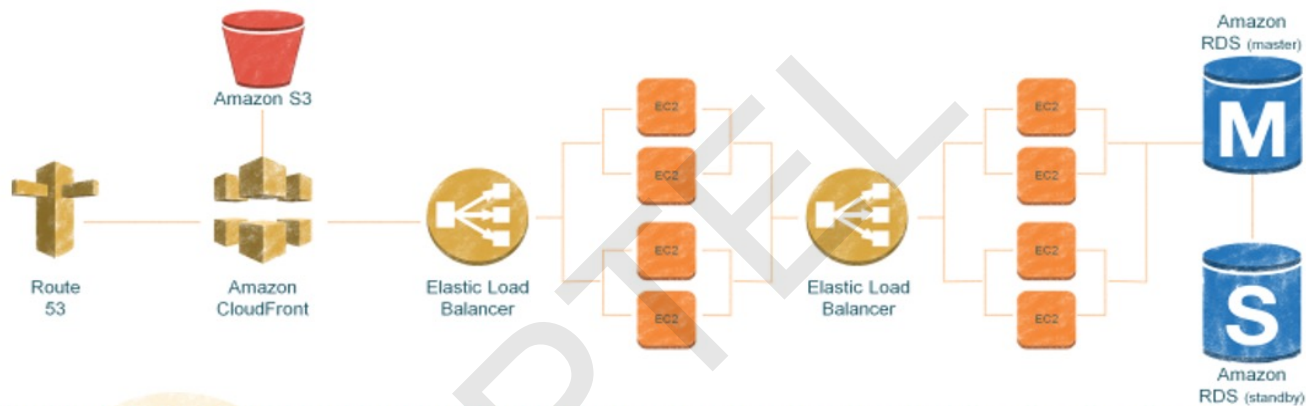


Amazon RDS – Cross Region Read Replicas

Improved disaster recovery operations.
Readable copies for cross-region applications.
Easy migration between regions

Web, Mobile, and Social Apps

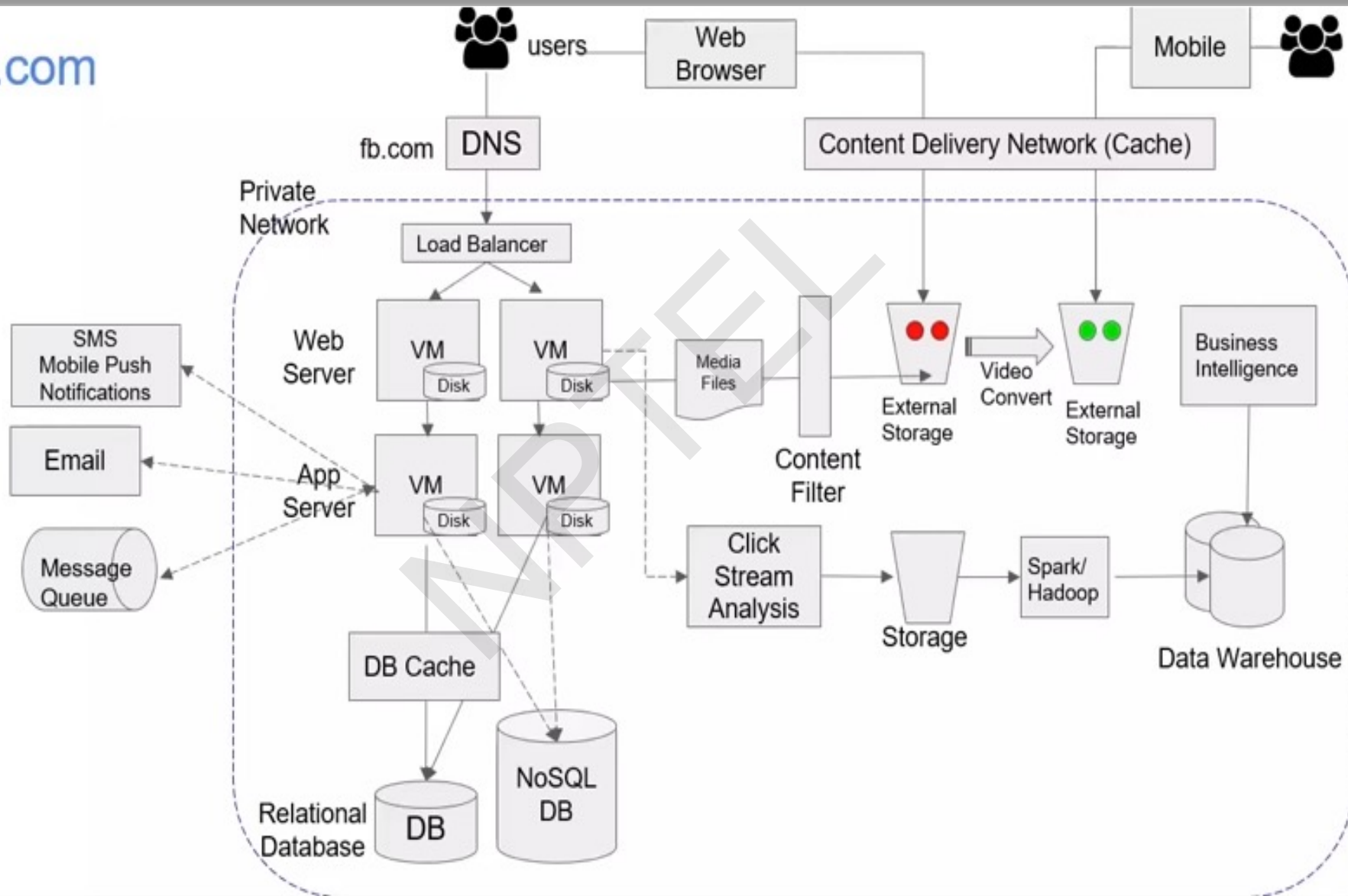
Web, Mobile, and Social Apps



Amazon AppStream

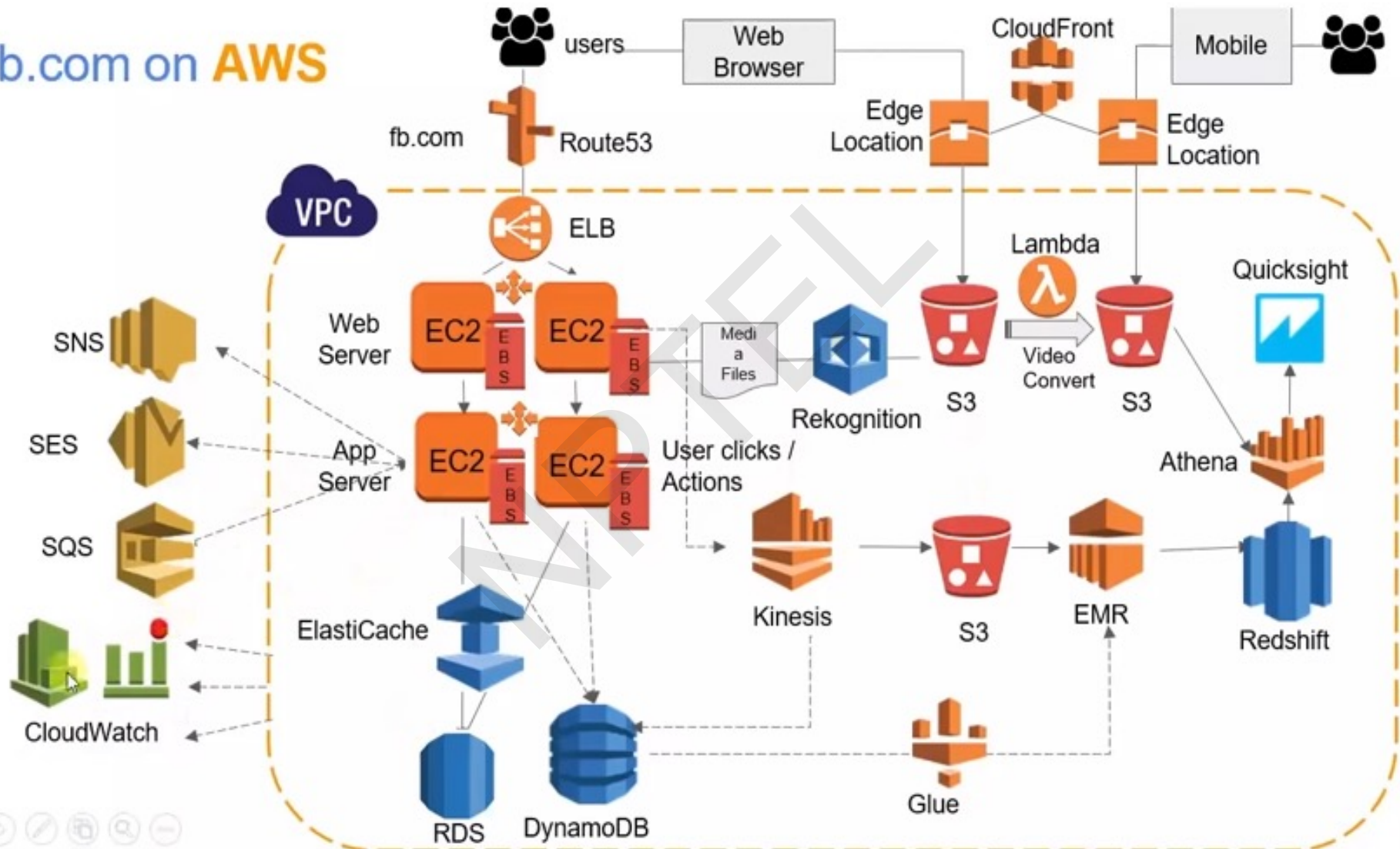
HD Video Quality Application Streaming
Captures user input to send back to the cloud. Responsive and consistent experience across devices





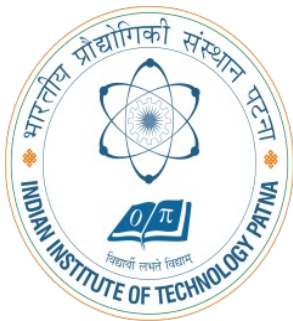
fb.com on AWS

fb.com on AWS



Thank You

Mathematical formulations for task-offloading in Edge-Cloud



Dr. Rajiv Misra
Professor

Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- An Edge-Cloud system architecture that includes the required components to support scheduling offloading tasks of IoT applications.
- An Edge-Cloud latency models that show the impact of different tasks' offloading scenarios/schemes for time-sensitive applications in terms of end-to-end service times.
- Evaluation of the offloading latency models that consider computation and communication as key parameters with respect to offloading to the local edge node, other edge nodes or the cloud..

Introduction

- Internet of Things (IoT) technology has quickly evolved in recent years, where the number of devices that are connected to the internet (IoT) has increased massively.
- More than 50 billion devices will be connected to the internet , which will produce a new set of applications such as Autonomous Vehicles, Augmented Reality (AR), online video games and Smart CCTV.
- Thus, Edge Computing has been proposed to deal with the huge change in the area of the distributed system.

Motivation

- For enhancing customer experience and accelerating job execution, IoT task offloading enables mobile end devices to release heavy computation and storage to the resource-rich nodes in collaborative Edges or Clouds.
- Nevertheless, resource management at the Edge-Cloud environment is challenging because it deals with several complex factors (e.g., different characteristics of IoT applications and heterogeneity of resources).
- Different service architecture and offloading strategies quantitatively impact the end-to-end service time performance of IoT applications .
- Consequently, the latency depends on the scheduling policy of applications offloading tasks as well as where the jobs will be placed in order to meet the requirements of latency-sensitive applications.

System Architecture

Recently, the **number of Internet of Things (IoT) devices connected to the Internet** has increased dramatically as well as the data produced by these devices.

This would require **offloading IoT tasks to release heavy computation and storage to the resource-rich nodes such as Edge Computing and Cloud Computing.**

Different service architecture and offloading strategies have a different impact on the service time performance of IoT applications.

An Edge-Cloud system architecture that supports scheduling offloading tasks of IoT applications in order to minimize the enormous amount of transmitting data in the network.

Also, it introduces the offloading latency models to investigate the delay of different offloading scenarios/schemes and explores the effect of computational and communication demand on each one.

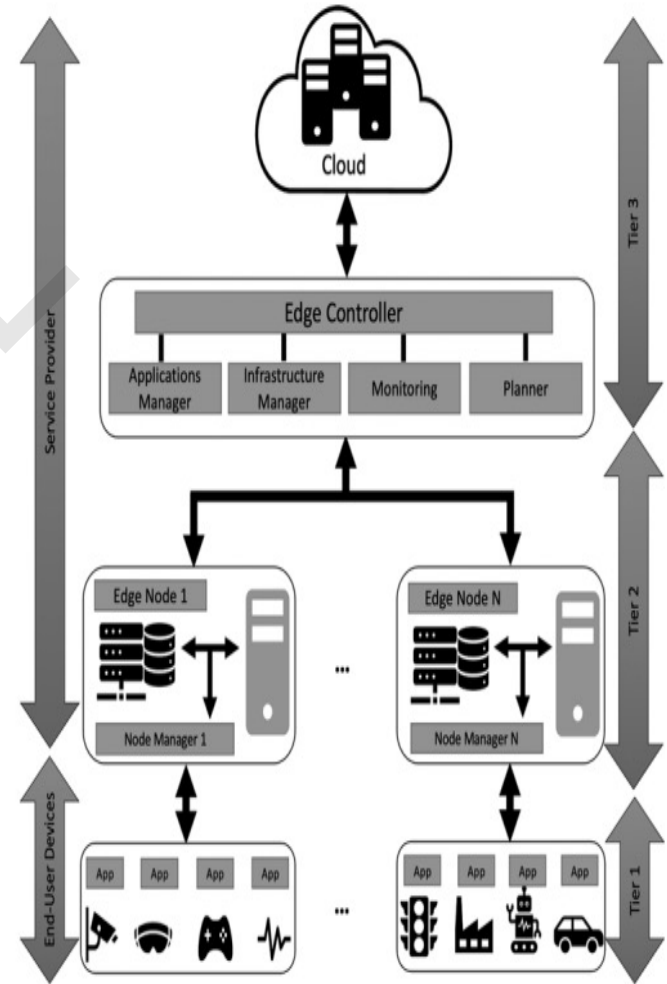


Figure 1: An overview of edge-cloud system

System Architecture

- The Edge-Cloud system from bottom to the top consists of three layers/tiers: IoT devices (end-user devices), multiple Edge Computing nodes and the Cloud (service provider).
- The IoT level is composed of a group of connected devices (e.g., smartphones, self driving cars, smart CCTV);
- These devices have different applications where each application has several tasks
- Difference in the given architecture is the introduced layer between the edge nodes and the cloud. This layer responsible for managing and assign offloading tasks to the edge nodes.

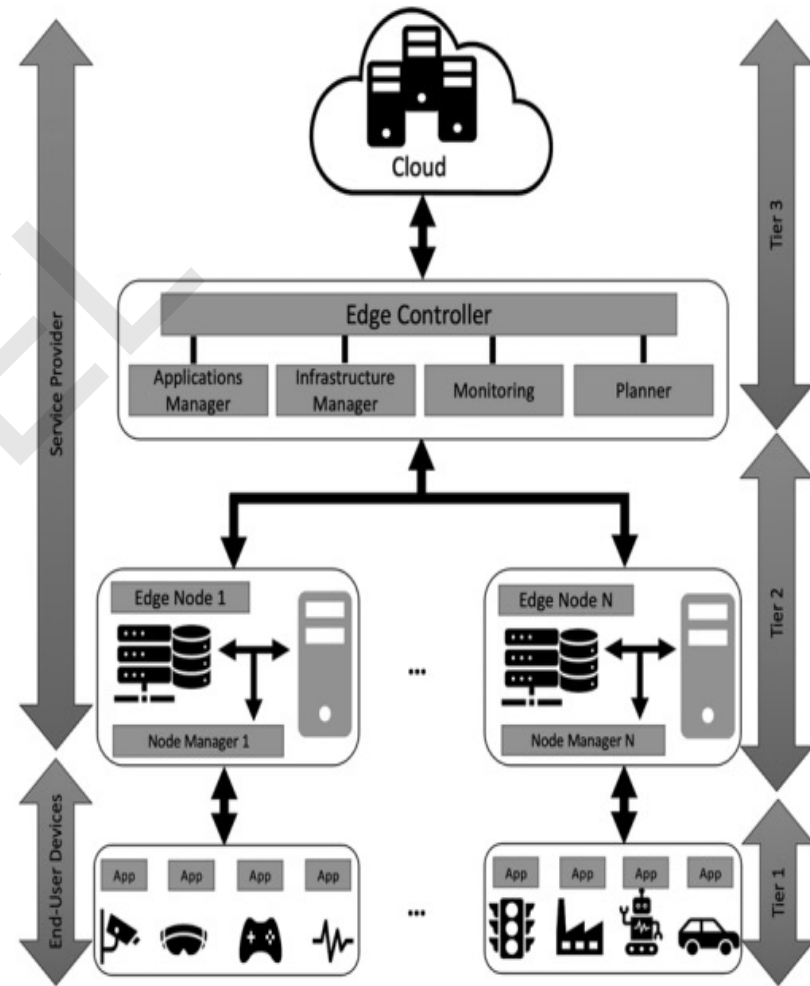


Figure 1: An overview of edge-cloud system

System Architecture: Edge Controller

- Edge Controller (EC) is also called *Edge Orchestrator*, which is a centralized component responsible for planning, deploying and managing application services in the Edge-Cloud system.
- EC communicates with other components in the architecture to know the status of resources in the system (e.g., available and used), the number of IoT devices, their applications' tasks and where IoT tasks have been allocated (e.g., Edge or Cloud).
- EC consists of the following components: Application Manager, Infrastructure Manager, Monitoring and Planner.
- The location of the Edge Controller can be deployed in any layer between Edge and Cloud.
- **For example**, EC act as an independent entity in the edge layer that manages all the edge nodes in its control. It is also responsible for scheduling the offloading tasks in order to satisfy applications' users and Edge-Cloud System requirements. The EC is synchronizing its data with the centralized Cloud because if there is any failure, other edge nodes can take EC responsibility from the cloud .

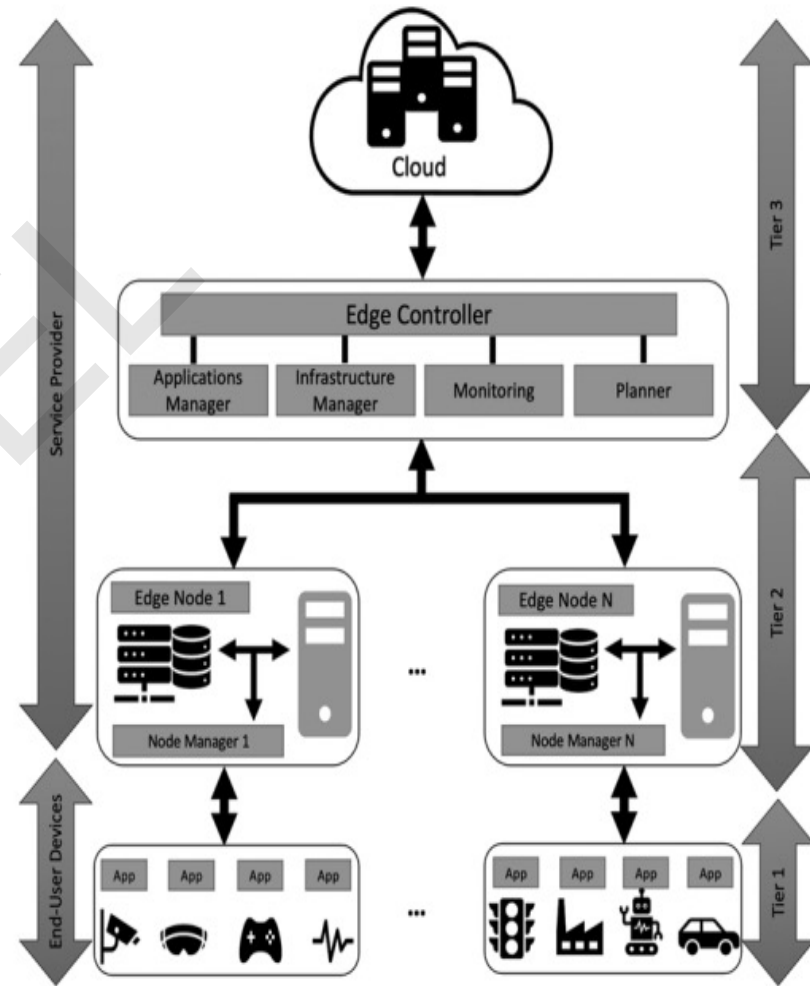


Figure 1: An overview of edge-cloud system

System Architecture: Application Manager

- **The application manager:**

It is responsible for managing applications running in the Edge-Cloud system. This includes requirements of application tasks, such as the amount of data to be transferred, the amount of computational requirement (e.g., required CPU) and the latency constraints. Besides, the number of application users for each edge node

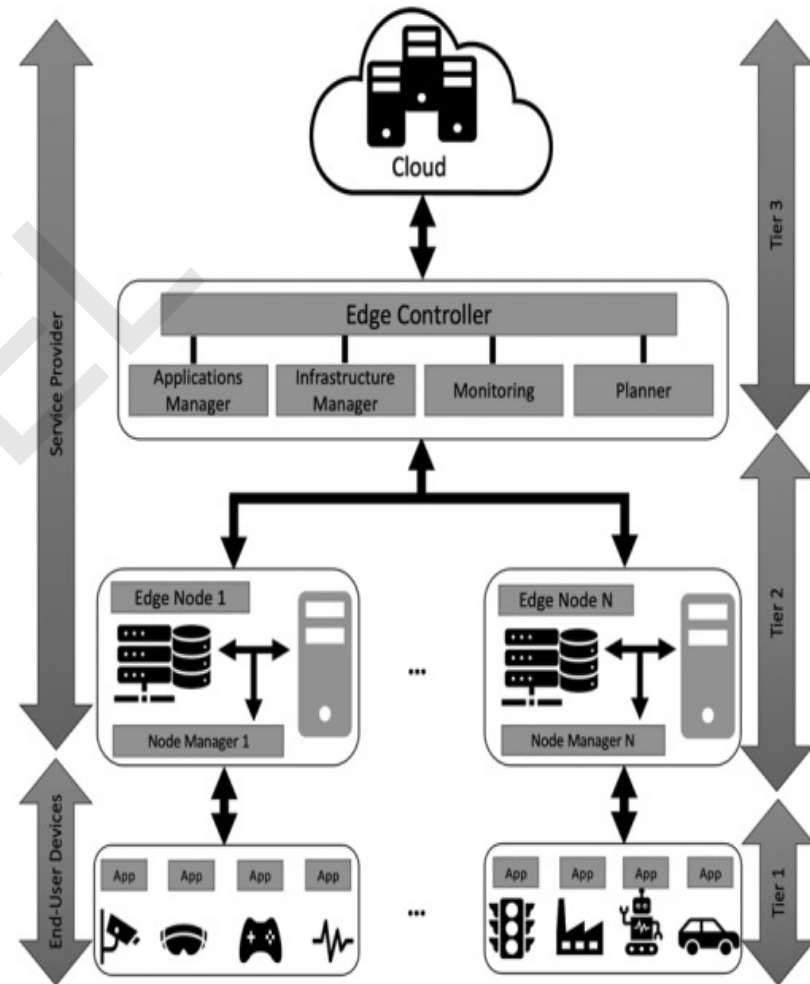


Figure 1: An overview of edge-cloud system

System Architecture: Infrastructure Manager

- **The Infrastructure Manager :**

The role of the infrastructure manager is to be in charge of the physical resources in the entire Edge-Cloud system. For instance, processors, networking and the connected IoT devices for all edge nodes.

Edge-Cloud is a virtualized environment; thus, this component responsible for the VMs as well. In this context, this component provides the EC with the utilization level of the VMs.

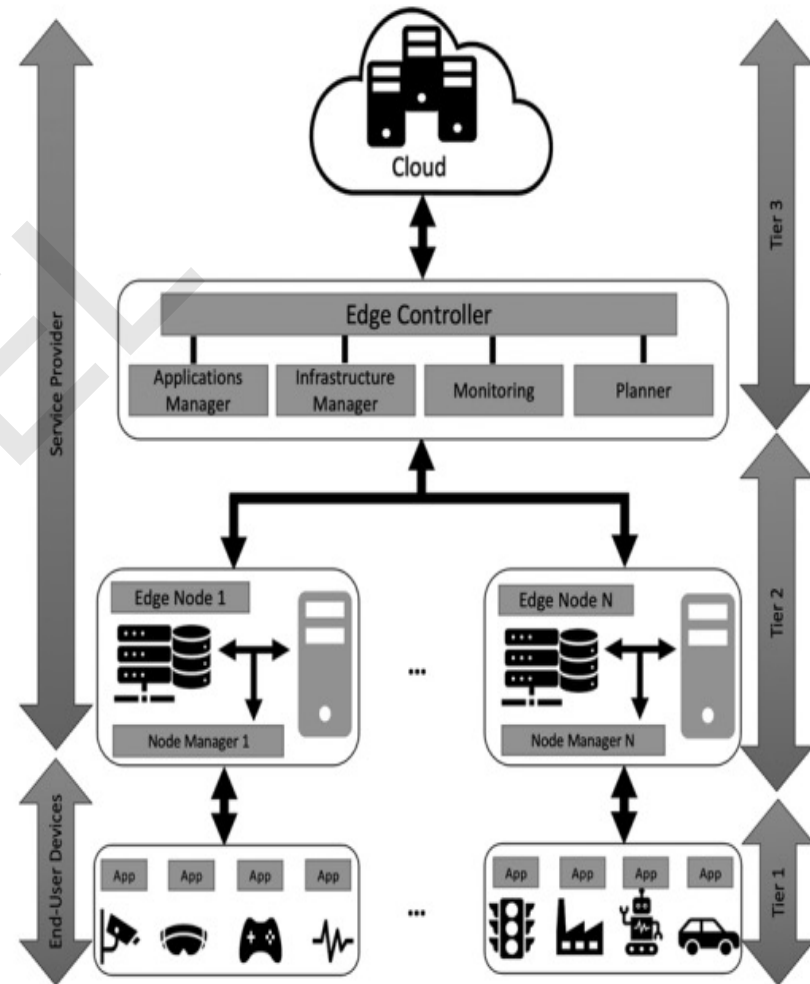


Figure 1: An overview of edge-cloud system

System Architecture: Monitoring and Planner

- **The Infrastructure Manager :**

The main responsibility of this component is to monitoring application tasks (e.g., computational delay and communication delay) and computational resources (e.g., CPU utilization) during the execution of applications' tasks in the Edge-Cloud system. Furthermore, detecting the tasks' failures due to network issues or the shortage of computational resources.

Planner:

The main role of this component is to propose the scheduling policy of the offloading tasks in the Edge-Cloud system and the location where they will be placed (e.g., local edge, other edges or the cloud). This offloading tasks works on this component and passes its results to EC for execution.

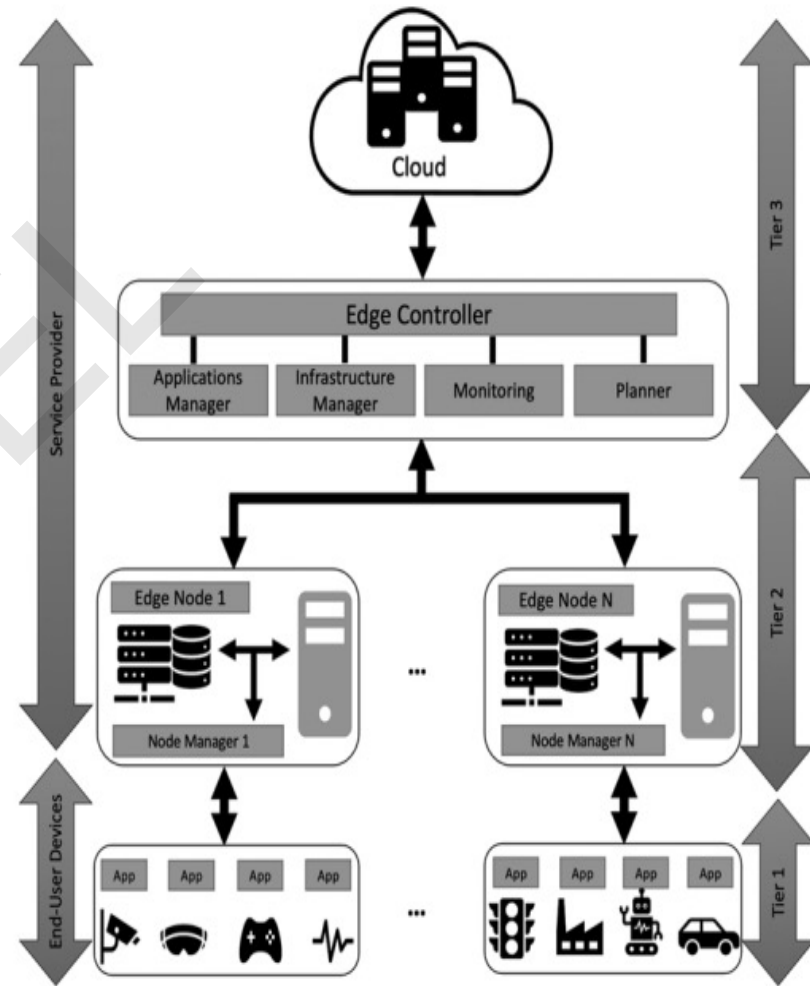
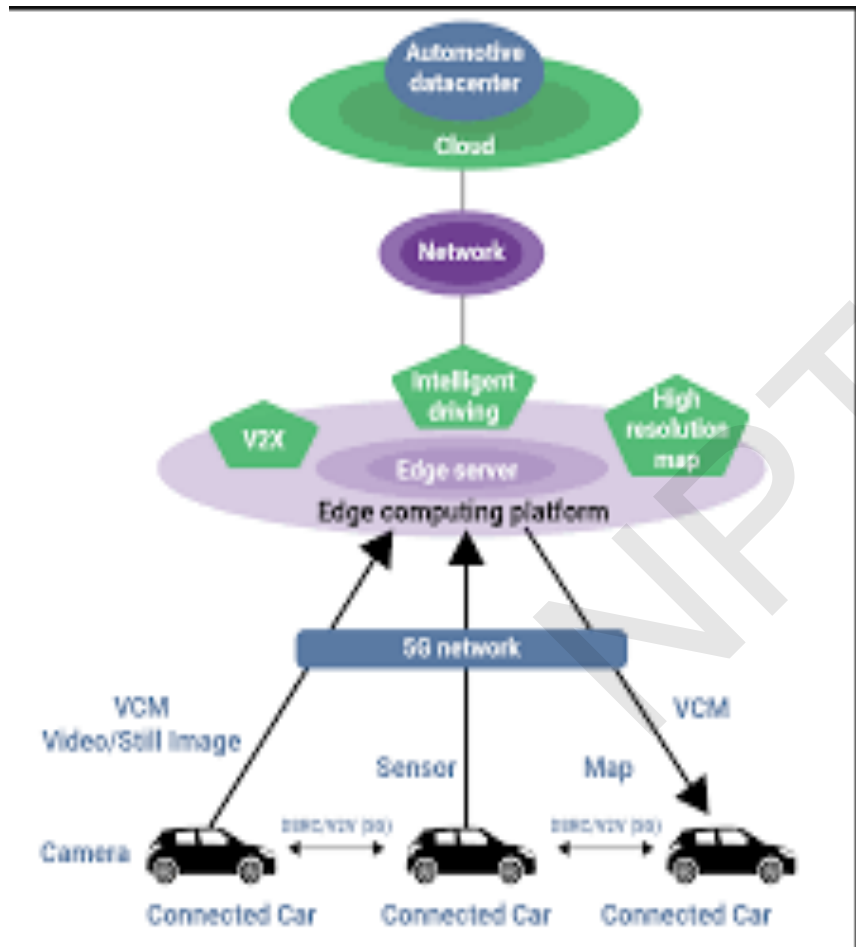


Figure 1: An overview of edge-cloud system

Latency Sensitive Applications



Latency Sensitive Applications

- Latency-sensitive applications have high sensitivity to any delays accrue in communication or computation during the interaction with the Edge-Cloud system.
- For instance, the IoT device sends data to the point that processing is complete at the edge node or the cloud in the back end of the network, and the subsequent communications are produced by the network in response to receive the results.
- For example, self-driving cars consist of several services, classified these services in categories based on their latency-sensitivity, quality constraints and workload profile (required communication and computation).
- First, critical applications, which must be processed in the car's computational resources, for instance, autonomous driving and road safety applications.
- Second, high-priority applications, which can be offloaded but with minimum latency, such as image aided navigation, parking navigation system and traffic control.
- Third, low-priority applications, which can be offloaded and not vital as high-priority applications (e.g., infotainment, multimedia, and speech processing).

Latency Sensitive Applications

Latency-sensitive applications

Industry	Applications
Industrial automation	Industrial Control Robot Control Process Control
Healthcare Industry	Remote Diagnosis Emergency Response Remote Surgery
Entertainment Industry	Immersive Entertainment Online Gaming
Transport Industry	Driver Assistance Applications Autonomous Driving Traffic Management
Manufacturing Industry	Motion Control Remote Control AR and VR Applications

Latency Models

- Modelling the various offloading decisions for IoT tasks that can increase the Quality of Service (QoS).
- With the increasing number of IoT devices, the amount of produced data, the need for an autonomous system that requires a real-time interaction as well as the lack of support from the central Cloud due to network issues; service time has been considered as one of the most important factors to be handled in Edge Computing.
- One of the main characteristics of Edge Computing is to reduce the latency level.
- Additionally, using Edge Computing will enhance application performance in terms of overall service time comparing to the traditional Cloud system.
- However, different offloading decisions within the Edge-Cloud system can lead to various service time due to the computational resources and communications types. The current real-world applications measure the latency between the telecommunication service provider and the cloud services.
- Compare the latency between offloading to the edge or the cloud, latency between multiple edge nodes that work collectively to process the offloading tasks. Investigating the latency of the Edge-Cloud system is an essential step towards developing an effective scheduling policy.
- Firstly, task allocation in the Edge-Cloud system is not only two choices (e.g., either at IoT device or in the cloud), but could be on any edge nodes. Moreover, edge nodes connected in a loosely coupled way on heterogeneous wireless networks (i.e., WLAN, MAN and WAN), making the process of resource management and the offloading decision more sophisticated.
- Secondly, given that task processing is allocated among multiple edge nodes working collectively and the cloud, it is challenging to make an optimal offloading decision. The latency models to investigate the delay of different offloading scenarios/schemes.

Latency Models: Latency to Local Edge

This is known as a one-level offloading system, which is basically offloading to “Cloudlet” or “Local Edge”.

It aims to provide a micro-data center that supports IoT devices within a specific area such as a coffee shop, mall center and airport .

Thus, IoT devices can offload their tasks to be processed on the edge or cloud, as an example.

This offloading scenario/scheme provides ultra-low latency due to the avoidance of network backhaul delays. The end-to-end service time composed of two delays, network delay and computational delay.

The network delay consists of the time of sending the data to the edge and the time to receive the output from the edge to the IoT device.

The computation time is the time from arriving the task to the edge node until the processing has completed. Therefore, the end-to-end service time latency is the sum of communication delay and computational delay, which can be calculated as follows:

- $L_{Local_edge} = t_{te_up} + t_{ce} + t_{te_down}$

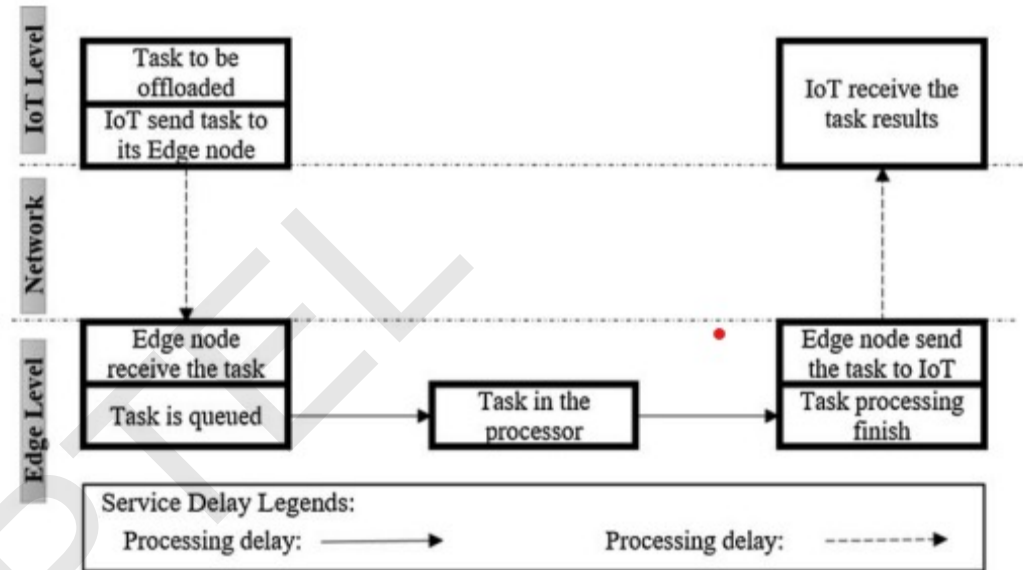


Figure 2: Latency to local edge

To clarify, IoT devices send their offloading tasks through the wireless network, and then the tasks will be processed by the edge node and finally send the results to IoT devices,

Latency Models: Latency to Local Edge with Cloud

In this offloading scenario/scheme, rather than relying on only one Edge node, the IoT tasks can be processed collaboratively between the connected Edge node and the cloud servers.

This will combine the benefits of both Cloud and Edge Computing, where the cloud has a massive amount of computation resources, and the edge has lower communication time.

In this scenario/scheme, the edge can do part of the processing such as pre-processing, and the rest of the tasks will be processed in the cloud.

IoT sends the computation tasks to the connected edge and then part of these tasks forwarded to the cloud.

Once the cloud finishes the computation, it will send the result to the edge, and the edge will send it to the IoT devices.

This scenario/scheme consists of communication time (e.g., the time between the IoT device to the edge node and the time between edge nodes to the cloud) and computation time (e.g., processing time in the edge and processing time in the cloud). Thus, the end-to-end service time can be calculated as follows:

$$L_{L_C} = t_{te_up} + t_{ce} + t_{tc_up} + t_{cc} + t_{tc_down} + t_{te_down}$$

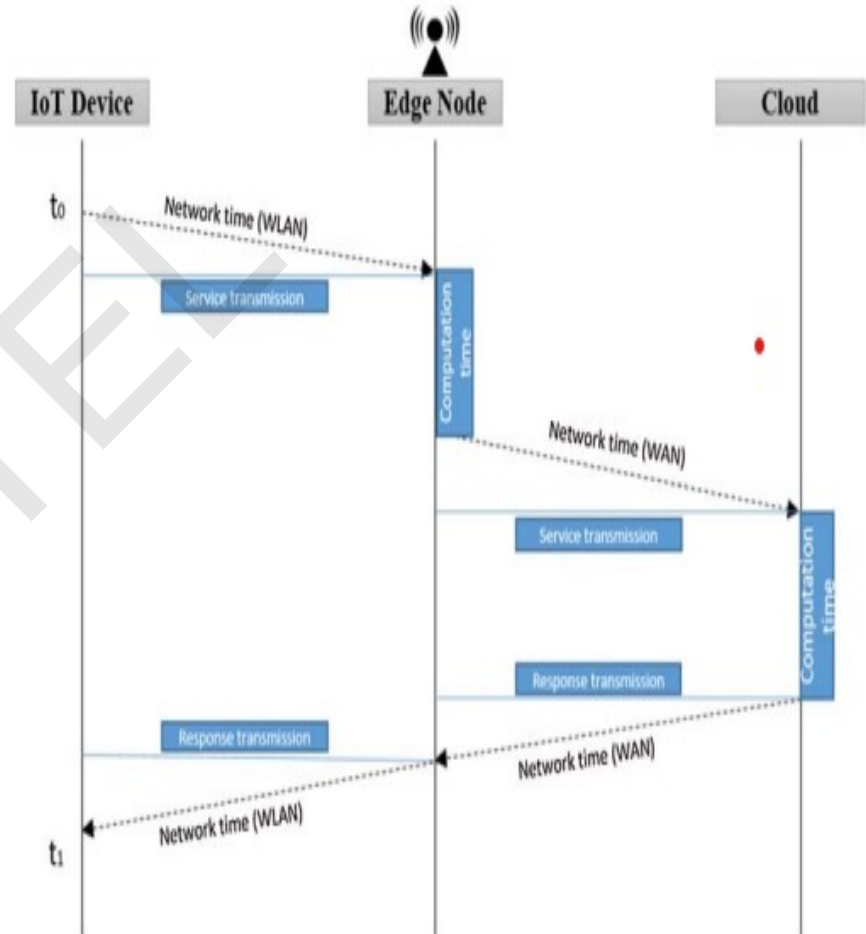


Figure 3: Latency to local edge with the cloud

Latency Models: Latency to Multiple Edge Nodes with Cloud

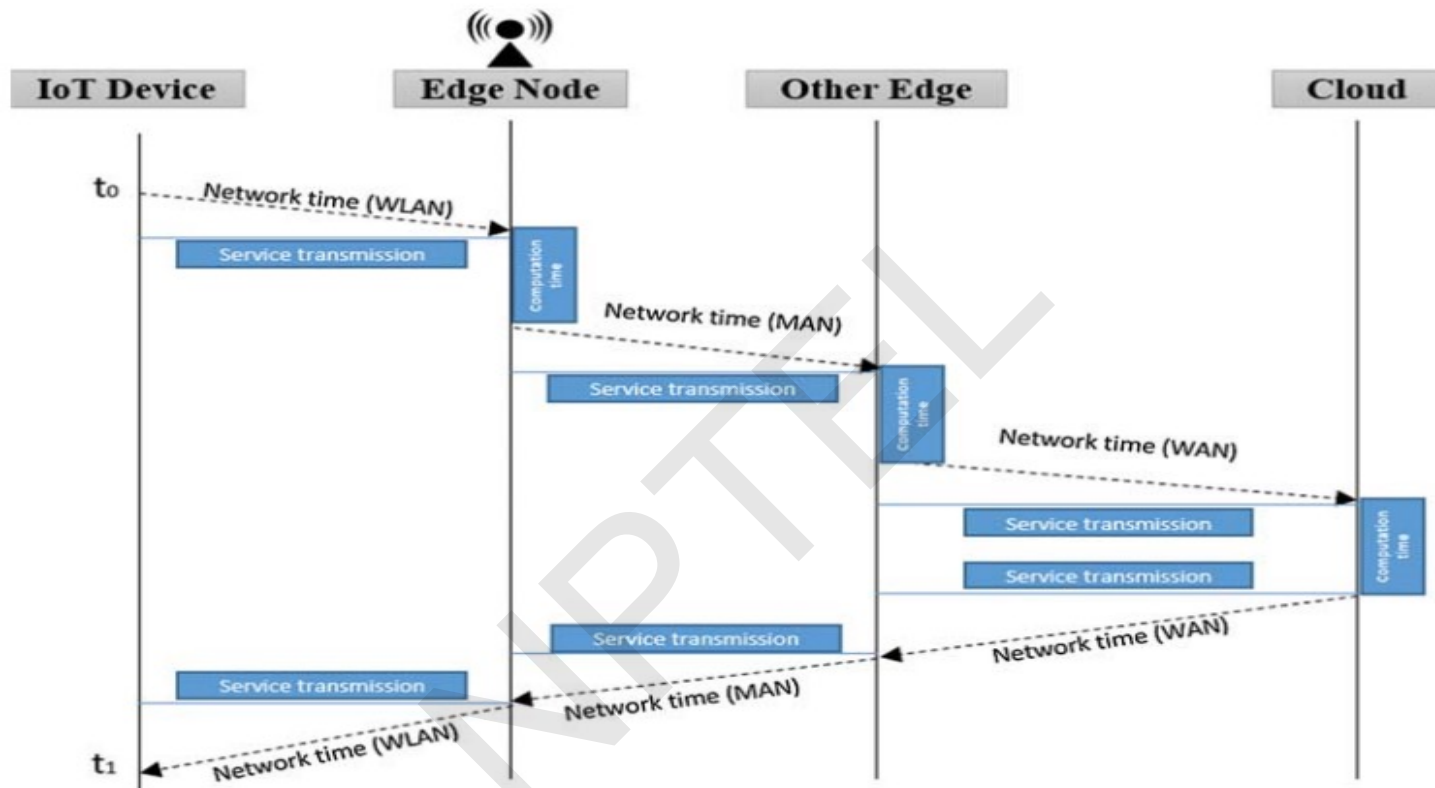


Figure 4: Latency to multiple edge nodes with the cloud

IoT sends the computation tasks to the connected edge and then part of these tasks transferred to other available resources in the edge level through the edge controller and the rest to the cloud.

Latency Models: Latency to Multiple Edge Nodes with Cloud

- This is known as a three-level offloading scenario/scheme that aims to utilize more resources at the edge layer and support the IoT devices in order to reduce the overall service time.
- It adds another level by considering other available computation resources in the edge layer.
- Basically, it distributes IoT tasks over three levels: connected edge, other available edge nodes and the cloud.
- The edge controller (edge orchestrator) controllers all edge servers by Wireless Local Area Network (WLAN) or Metropolitan Area Network (MAN), which have low latency compared to Wild Area Network (WAN).
- This will help to decrease the dependency of cloud processing as well as increase the utilization of computing resources at the edge.
- This scenario/scheme consists of communication time (e.g., the time between the IoT device to the edge node, the time between edge node to other collaborative edge node and the time between edge nodes to the cloud) and computation time (e.g., processing time in the edge, processing time in other collaborative edge node and processing time in the cloud). Thus, the end-to-end service time can be calculated as follows:
- $L_{three_off} = t_{te_up} + t_{ce} + t_{ceo} + t_{tc_up} + t_{cc} + t_{tc_down} + t_{teo_down} + t_{te_down}$

Experiment

Assumptions:

- We have three edge nodes connected to the cloud.
- Each edge node has two servers, and each of them has four VMs with a similar configuration.
- The cloud contains an unlimited number of computational resources

Key parameters of the simulation environment :

Key parameters: Values

Simulation Time :30 min

Warm-up Period :3 min

Number of Iterations: 5

Number of IoT Devices: 100–1000

Number of Edge Nodes :3

Number of VM per Edge Server: 8

Number of VM in the Cloud :not limited

Average Data Size for Upload/Download (KB) :500/500

Results

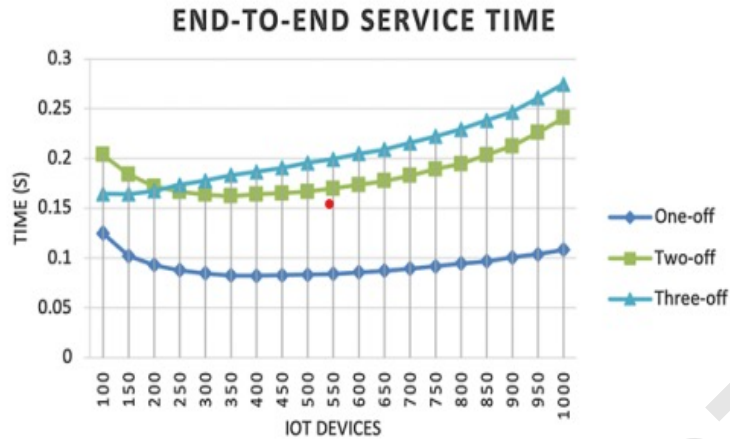


Figure 5: End-to-end service time for three offloading scenarios/schemes

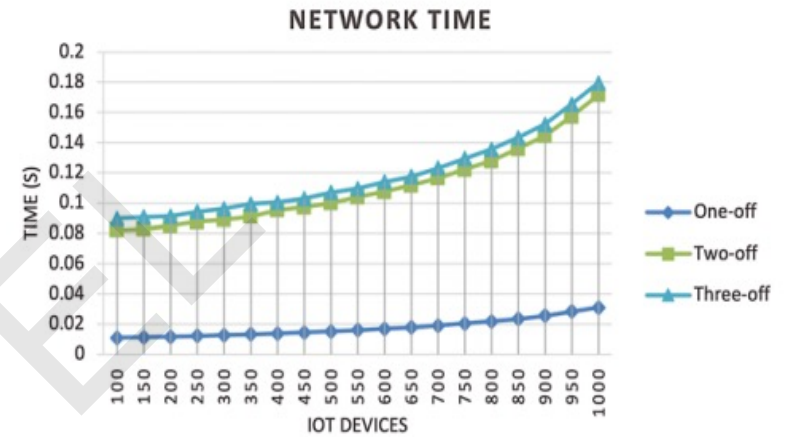


Figure 6: Network time for three offloading scenarios/schemes

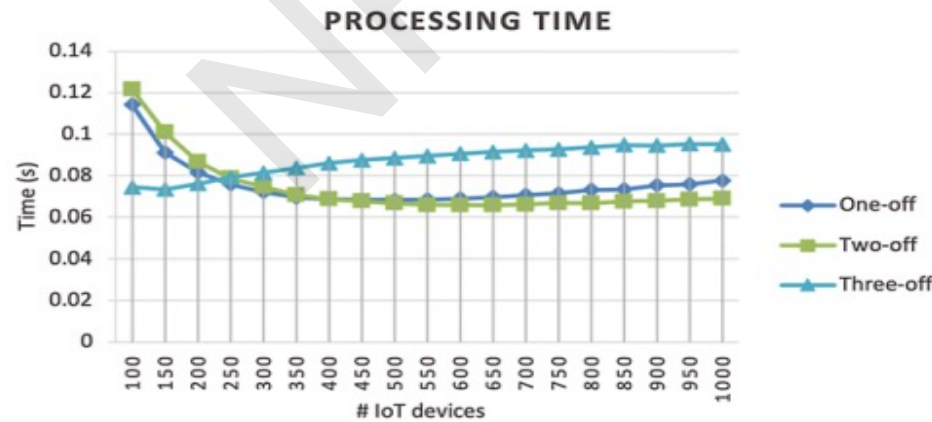


Figure 7: Processing time for three offloading scenarios/schemes

Summary

- We presents an Edge-Cloud system architecture that enables the offloading of tasks for IoT applications.
- The architecture includes several components that interact with each other to support task offloading, such as IoT devices, edge nodes, and cloud servers.
- Offloading latency models were explained that consider computation and communication as key parameters for offloading tasks to different destinations, including local edge nodes, other edge nodes, and the cloud.
- The experiments conducted on EdgeCloudSim to evaluate the latency models for three different offloading scenarios .

Thank You

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning



Dr. Rajiv Misra
Professor

Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- A joint decision-making problem for cost minimization in IoT edge computing is modeled, taking into account processing latency, energy consumption, and task throw rate.
- The Online Predictive Offloading (OPO) algorithm is proposed based on Deep Reinforcement Learning and Long Short-Term Memory.
- The algorithm predicts the edge server's load in real-time and allocates resources in advance, improving the convergence accuracy and speed of the offloading process.

Introduction

- Modeling the problem of computing offloading in a multi-edge, multi-device computing scenario as a nonlinear optimization problem.
- Moreover, the goal of task offloading is minimizing long-term costs in terms of latency and energy consumption.
- By predicting the characteristics of tasks and edge server loads, tasks are dynamically offloaded to the optimal edge server
- . In the decision model, the prediction is combined with task decision to dynamically allocate resources for different tasks to further reduce latency and improve service quality.

Motivation

- Task offloading can result in **additional transmission delays and energy consumption**.
- Task offloading problem is modelled as a joint decision-making problem for cost minimization, considering processing latency, energy consumption, and task throw rate.
- The Online Predictive Offloading (OPO) algorithm based **on Deep Reinforcement Learning (DRL) and Long Short-Term Memory (LSTM) networks** is used to solve the task offloading problem.
- In the training phase, the OPO algorithm predicts the load of the edge server in real-time with the LSTM algorithm, improving the convergence accuracy and speed of the DRL algorithm in the offloading process.
- In the testing phase, the **LSTM network predicts the characteristics of the next task**, and the DRL decision model allocates computational resources for the task in advance, reducing the response delay and improving offloading performance.

System Model

- The model is built on a multi-terminal, multi-edge network scenario
- Here the set of terminal layer devices are denoted by $\mathcal{M} = \{1, 2, \dots, M\}$.
- On each **MD (Mobile device)**, there exists a task queue and a computation queue
- the task queue stores the tasks to be decided for offloading
- the computation queue processes the tasks that are executed locally.
- the set of edge layer servers are denoted by $\mathcal{N} = \{1, 2, \dots, N\}$.
- Multiple computation queues are included in each edge server for parallel computation of transmission queue offload tasks.
- Figure shows an illustration of EC system with a mobile device and an edge node.

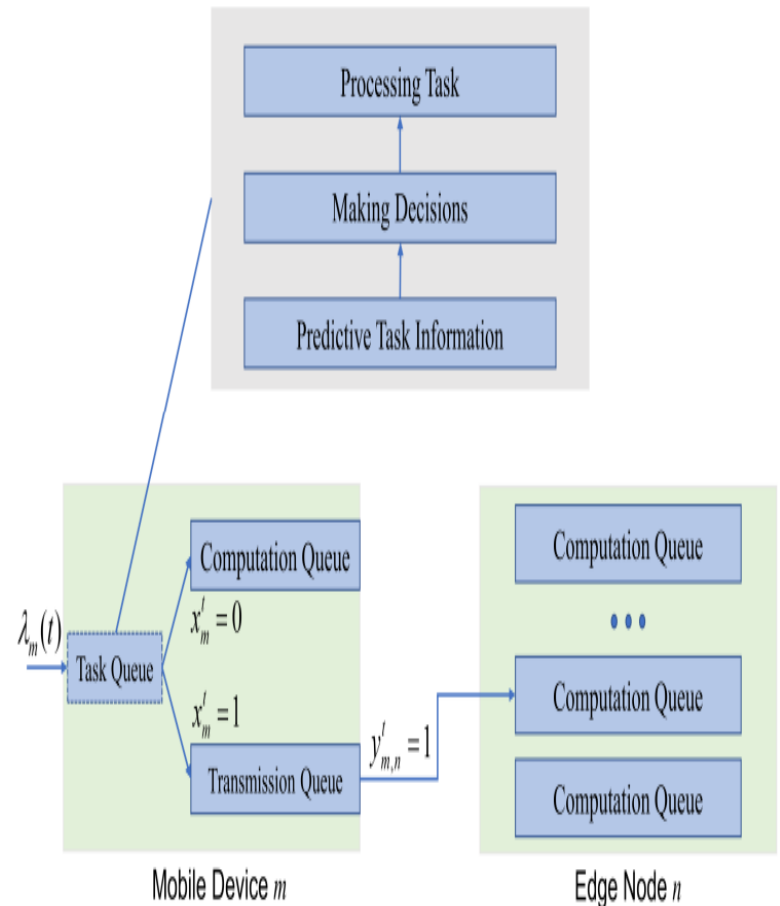


Figure 1. An illustration of EC system with a mobile device and an edge node.

Task Model

- For any MD, the tasks generated in different time slots are identified by $\mathcal{T} = \{1, 2, \dots, T\}$.
- Each arriving task is first stored in corresponding MD task cache queue, and then the decision model gives where the task will be offloaded to be executed.
- For $t \in \mathcal{T}$, new task generated by the terminal device $m \in \mathcal{M}$ is denoted as

$$\lambda_m(t) = (D_m^t, \rho_m^t, \tau_{m,max}^t).$$

Where ,

D_m^t : size of the task data

ρ_m^t : computational resources required per bit

$\tau_{m,max}^t$: maximum tolerated delay of the task

Decision Model

- When the terminal device m has a new task $\lambda_m(t)$ in time slot t the decision model has to give the offloading scheme.
- $x_m^t \in \{0, 1\}$, indicates whether the current task is offloaded
- $x_m^t = 0$ indicates that the task is executed on the MD,
- $x_m^t = 1$ indicates that the task will be offloaded to an edge server for execution
- $y_{m,n}^t \in \{0, 1\}$ represents the edge server to which the task is offloaded for execution
- $y_{m,n}^t = 1$, the task is offloaded to the edge server $n \in \mathcal{N}$ for execution
- The tasks in this model are atomic level tasks
- Each offloaded task can be executed in only one edge server, and the tasks offloaded to the edge server for execution are constrained by

$$\sum_{n \in \mathcal{N}} y_{m,n}^t = 1, m \in \mathcal{M}, t \in \mathcal{T}, x_m^t = 1$$

Computation model: Terminal Layer Computing Model

- The task generated by the t time slot must wait until the computation queue is free to execute the computation. Then waiting delay is:

- $$\tau_{m,wait}^t = \left[\max_{t' \in \{0,1,\dots,t-1\}} l_m^{comp}(t') - t + 1 \right]^+ \quad (1)$$

Where,

$l_m^{comp}(t)$: completion time slot of the task

processing delay in the computational queue is

- $$\tau_{m,exe}^t = \frac{D_m^t \rho_m^t}{f_m^{device}} \quad (2)$$

Where,

f_m^{device} : processing capacity (bits/s) of the MD

By 1 and 2,

$$l_m^{comp}(t) = \min\{t + \tau_{m,wait}^t, \tau_{m,exe}^t, \tau_{m,max}^t\}$$

energy consumption E_m^{device} required for the task to be executed locally

$$E_m^{device} = P_m^{exe} \tau_{m,exe}^t + P_m^{wait} \tau_{m,wait}^t$$

Computation model: Edge Layer Computing Model

the processing latency of the task at the edge layer

$$\tau_{m,exe}^t = \frac{D_m^t \rho_m^t}{f_m^{device}}$$

completion time for edge layer tasks

$$l_{m,n}^{comp}(t) = \min\{t + \tau_{m,wait}^t + \tau_{m,n}^{tran} + \tau_{m,n,exe}^t, t + \tau_{m,max}^t\}$$

Total delay of task on edge server n,

$$\tau_{m,nEC}^t = \min\{\tau_{m,wait}^t, \tau_{m,n}^{tran}, \tau_{m,n,exe}^t, \tau_{m,max}^t\}$$

Where ,

$\tau_{m,wait}^t$: waiting delay in the local model

$\tau_{m,n}^{tran}$: transmission delay

$\tau_{m,wait}^t + \tau_{m,n}^{tran} + \tau_{m,n,exe}^t$: time slot required for a task to be offloaded from the endpoint to the edge server and executed to completion

$\tau_{m,max}^t$: maximum tolerated delay

energy consumption incurred when tasks are offloaded to the edge server

$$E_{m,n}^{edge} = P_m^{wait} \tau_{m,wait}^t + P_{m,n}^{tran} \tau_{m,n}^{tran} + P_{m,n}^{exe} \tau_{m,n,exe}^t$$

Where, $P_m^{wait} \tau_{m,wait}^t$, $P_{m,n}^{tran} \tau_{m,n}^{tran}$, $P_{m,n}^{exe} \tau_{m,n,exe}^t$ denote waiting energy consumption, transmission consumption, and edge node computation consumption of the task, respectively.

Goal

- The overall model of the system is a trade-off between the time delay and energy consumption of the task computation to create a minimization cost problem
- The solution goal is to minimize the total cost of the tasks generated in the system over time.

Prediction Model: Task Prediction Model

- A decision process is required after task generation, and there will be a certain time delay from task generation to give a decision.
- Although task generation is a dynamic and random process, considering the long-term nature of the task, it will have a strong correlation with time.
- Therefore, based on the history of user devices, we can predict the tasks that will be generated in the next network time slot
- As shown in Figure, we can predict the information of the future task by the prediction model, and determine the decision and allocate computing resources for the task.
- If the error between the real task and the predicted task is within the allowed threshold, the task is directly offloaded and computed according to the assigned decision information.
- Otherwise, the offloading decision is given using the decision model and the information of the new task is added to the historical data as training samples.
- By training the LSTM network, the weights and biases of each gate in the network are updated to improve the accuracy of the prediction model.

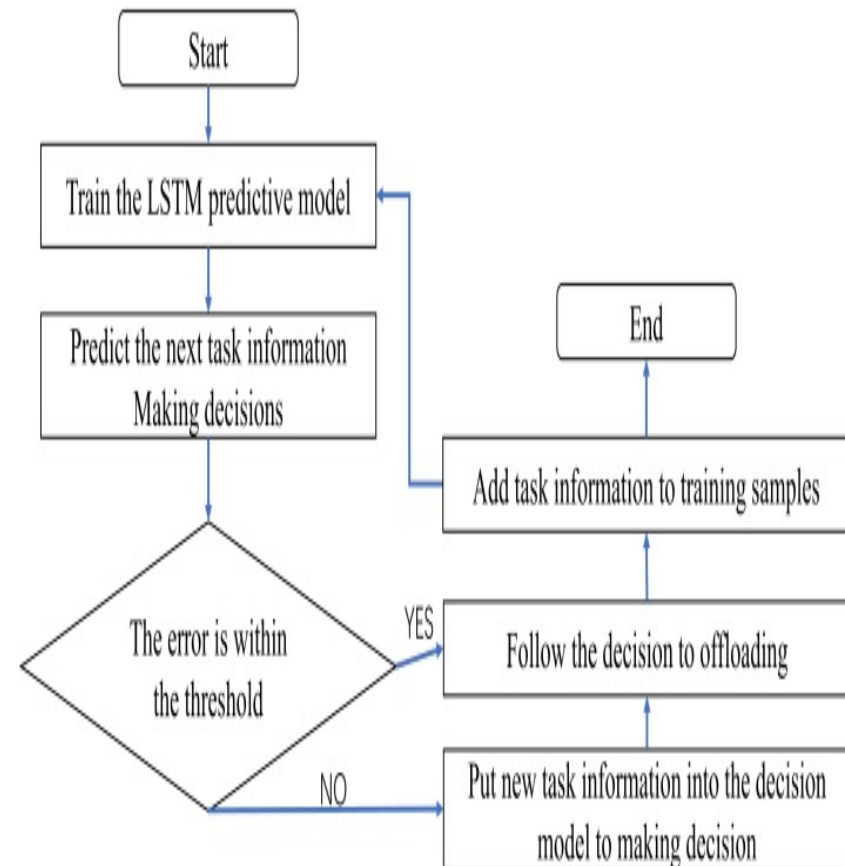


Figure 2. Flow chart of task prediction.

Prediction Model: Load Prediction Model

- Historical load sequence data is logged and used to train an LSTM load prediction model.
- The predicted idle server (H_t) is obtained from the trained model using historical load sequence data as input.
- The predicted idle server is used as the offload computing node when training the DRL.
- The DRL training process involves selecting actions with a certain probability (ϵ).
- When a random action is selected, the size comparison between a random value σ and the probability ϵ is used to determine whether it is a Random Action or a Prediction Action.
- Using Prediction Action with the pre-selected idle server can reduce the number of explorations by the agent and improve convergence speed of the algorithm.

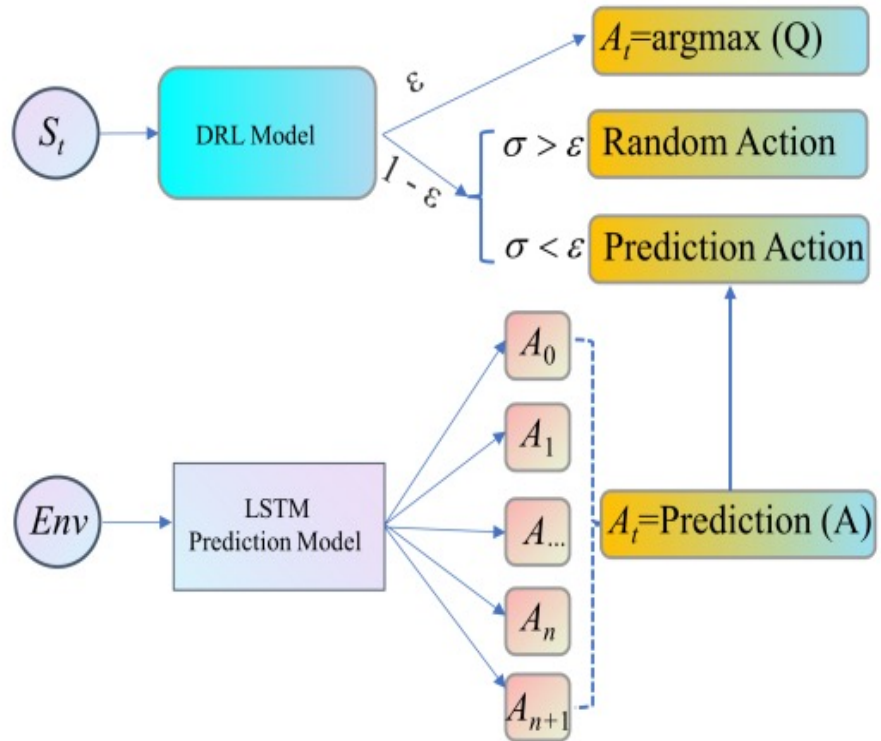


Figure 3. Illustration of offloading decision.

Model Training

- The goal of DRL is to maximize the total reward by making optimal actions.
- DRL typically uses ϵ -greedy strategies for exploration and exploitation.
- Exploration involves random selection of any action with probability in expectation of a higher reward, while exploitation selects the action with the largest action estimate.
- The stochastic strategy fully explores the environment state, but requires extensive exploration and low data utilization.
- In the model, action selection is the offloading decision of the task, with the action space known whether to execute locally or offload to an edge server.
- During stochastic exploration, LSTM is used to predict the load of the edge server and give an optimal action.
- The optimal server at the next time slot is predicted based on historical load situation to obtain a higher reward and avoid edge server load imbalance.

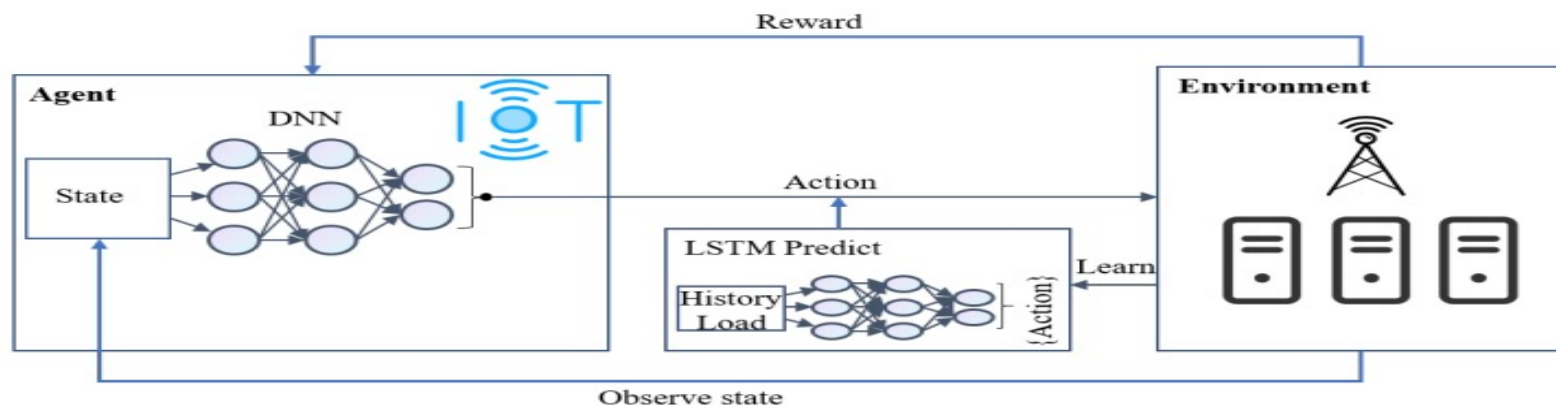


Figure 4. Illustration of model training phase.

Offloading Decision

- Each MD generates different types of tasks at different time slots.
- There is a system response delay to the task's decision request and a waiting delay in the queue between the generation of a task and giving a decision.
- The edge system processes data from MD and stores processed records.
- Based on historical records, feature information of the next arriving task can be predicted by LSTM.
- The predicted information is given to the reinforcement learning decision model to make an offloading scheme for the predicted task.
- When the real task arrives, the offloading decision is given directly if the error between the real task and predicted task is within the allowed range.
- If the error is not within the allowed range, the decision is made according to the real task using the decision model.
- Predicting the task's information can reduce the task's response and waiting delay in the system.

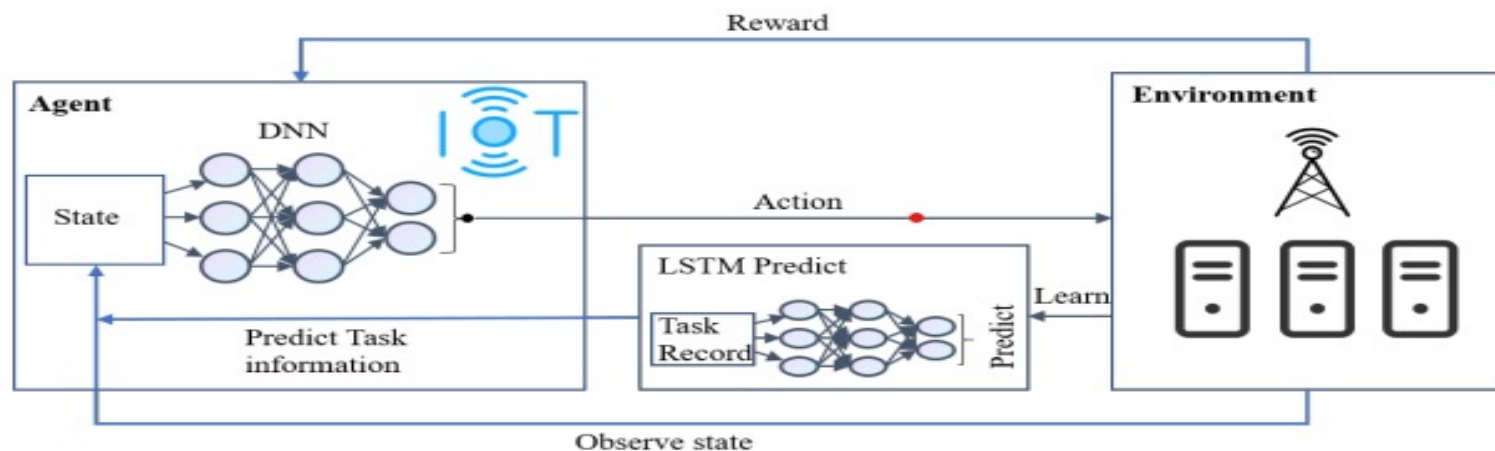


Figure 5. Illustration of offloading decision phase.

Algorithm Design: DQN(Deep Q Network)

- A typical DQN model is composed of agent, state, action, and reward
- the policy is generated as a mapping $\pi : S \rightarrow A$ of states to actions to obtain a reward $R, r_t(s_t, a_t)$, denotes the reward that can be obtained by choosing action a_t in state s_t
- $R_0^\gamma = \sum_{(t=0)}^T \gamma^t r_t(s_t, a_t)$, is the long-term reward
- when the state space and action space dimensions are large, it is difficult to put all state-action pairs into Q-table.
- To solve this problem, the DQN model in DRL combines deep neural networks and Q-learning algorithms, and it transforms the Q-table tables into the Q-networks and uses neural networks to fit the optimal Q-functions.
- There are two neural networks with the same structure but different parameters in DQN, i.e., the target network and the main network.
- When iteratively updating the network, the algorithm first uses the target network to generate the target Q-value as the label $f(t)$, and uses the loss function $\text{Loss}(\theta)$ to update the parameters of the main network.
- After the introduction of the target network, the target Q value generated by the target network remains constant in time j , which can reduce the correlation between the current Q value and the target Q value and improve the stability of the algorithm.

Algorithm Design: Replay Memory

- In order to break the correlation within the data, DQN uses the experience replay method to solve this problem.
- After interacting with the environment, the agent is stored in the replay buffer in the form of (s_t, a_t, r_t, s_{t+1}) .
- When executing valuation updates, the agent randomly selects a small set of experience tuples (s_t, a_t, r_t, s_{t+1}) from the replay buffer at each time step
- Then the algorithm updates the network parameters by optimizing the loss function
- Using experience replay can not only make training more efficient, but also reduce the problem overfitting that generated by the training process

Algorithm Design: Double DQN

- Double DQN is proposed to solve the overestimation problem.
- DQN takes the maximum value with max each time, and the difference between this maximum value and the weighted average value introduces an error,
- this will lead to overestimation after a long time accumulation.
- The Double DQN is composed of two networks, QA and QB ,
- it utilizes these two networks to proceed the state valuation and the action output alternatively.
- That is, one network is used to select out the action, and the other network is used to update the Q value according to the selected action.
- The Double DQN makes the learning process more stable and reliable by separating the two steps of selecting the action corresponding to the Q value and evaluating the Q value corresponding to the action,
- this eliminates the overestimation brought by the greedy algorithm and obtains a more accurate Q estimation.
- Instead of finding the label value of parameter update directly from the target network, Double DQN finds the action corresponding to the maximum Q value in QA and then uses this selected action to compute the target value of parameter update in QB .

Algorithm Design: Dueling DQN

- Compared with DQN, Dueling DQN considers the Q network into two parts

the first part is only related to the state S , and the specific action A to be adopted has nothing to do with this part is called the **value function** part, noted as $V^\pi(s)$,

second part is related to both the state S and action A , this part is called the **action advantage function**, noted as $A^\pi(s, a)$, the final value function can be expressed as

$$Q^\pi(s, a) = A^\pi(s, a) + V^\pi(s)$$

Algorithm Design: Decision Model Elements

- Agent:

1. Each MD is considered as an agent that selects the next action according to the current state of the environment and improves the ability of the agent to make decisions by continuously interacting with the environment.
2. The goal of the agent is to make the optimal action in any state, thus minimizing the total cost in the edge computing system.

- State:

1. At the beginning of each time slot, each agent observes the state of the environment
2. It includes the properties of the MD task, the waiting queue state, the transmission queue state, bandwidth information, and the real-time load of the edge nodes, all the states are closely related to the action to be selected by the agent.

- Action:

1. Based on the current state, the agent first decides whether the newly generated task needs to be offloaded for computation.
2. If it needs to be offloaded, it chooses which server to offload
3. It also chooses the appropriate transmission power when offloading the transmission

Algorithm Design: Decision Model Elements

- Reward:
 1. After observing the state at time slot t , the agent takes an action according to the policy and then receives a reward at time slot $t + 1$ while updating the scheduling policy network to make an optimal decision in the next time slot.
 2. The goal of each agent is to maximize its long-term discounted reward by optimizing the mapping from states to actions so that the agent tends to make optimal decisions in its continuous interaction with the environment.
 3. The reward function is shown below,

$$\mathbb{E} \left[\sum_{t=0}^T \gamma^t r_t(s_t, a_t) \right]$$

Algorithm: Online Predictive Offloading Algorithm

```
1: Input: input different tasks in each time slots
2: Output: Optimal offloading decision and total cost
3: Initialize  $Q^A$ ,  $Q^B$  and  $s$ 
4: Initialize replay memory  $D$  to capacity  $N$ ;
5: for episode = 1,  $M$  do
6:     Initialize sequence  $s$ , and preprocessed sequence
7:     for  $t = 1, T$  do
8:         With probability  $1 - \epsilon$  select a random action or LSTM predict action
9:         Generate another random number  $\sigma$ 
10:        if  $\sigma > \epsilon$  then
11:             $a_t = \text{Random Action Selection}(s_t)$ 
12:        end if
13:        if  $\sigma < \epsilon$  then
14:             $a_t = \text{Prediction Action Selection}(s_t)$ 
15:        end if
16:        Otherwise select  $a$  by  $a^* = \operatorname{argmax}_a Q^A(s, a)$  or  $b^* = \operatorname{argmax}_a Q^A(s, a)$ 
17:        Execute action  $a_t$  and receive  $r_t$  and  $s_{t+1}$ 
18:        Store  $(s_t, a_t, r_t, s_{t+1})$  into  $D$ 
```

Algorithm: Online Predictive Offloading Algorithm

```
19:      Randomly sample a mini-batch of experience from D
20:      Perform a gradient descent step on Loss( $\theta$ ) with respect to the
network parameters
21:      Choose a, based on  $Q^A(s, \bullet)$  and  $Q^B(s, \bullet)$ , observe r,s'
22:      if UPDATE(A) then
23:           $Q^A((s, a) \leftarrow Q^A((s, a) + \rho[r + \gamma \max_{a'} Q^B(s', a^*) - Q^A(s, a)$ 
24:      else if UPDATE(B) then
25:           $Q^B(s, a) \leftarrow Q^B(s, a) + \rho[r + \gamma \max_{a'} Q^A(s', b^*) - Q^B(s, a)$ 
26:      end if
27:  end for
28: end for
29: Repeat
```


Experiment

Assume,

- We use a dataset from Google Cluster, which includes information about the arrival time, data size, processing time, and deadline of the tasks.
- Each type of task processing density, task processing time and the size of data volume are related
- preprocess the raw data according to the characteristics of the data and make the data size compatible with the established model by normalization and denormalization
- Considering following Simulation parameters.

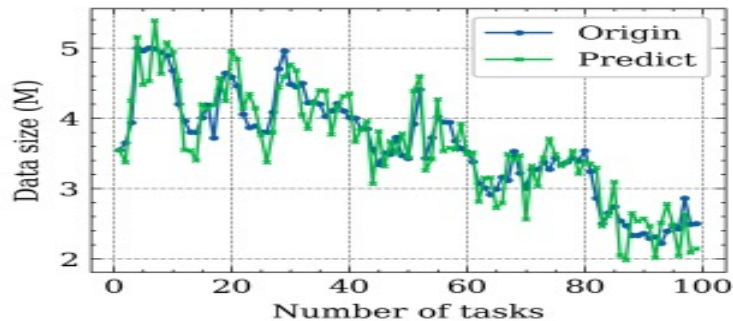
Parameter	Value
f_m^{device}	2.5 GHz
f_m^{EC}	41.8 GHz
B_m^t	10 MHz
$p_{m, \text{exe}}$	5 Watt
$p_{m, \text{wait}}$	0.2 Watt
$p_{m, n, \text{tran}}$	2 Watt
$p_{m, n, \text{exe}}$	10 Watt

Experiment: Task Prediction

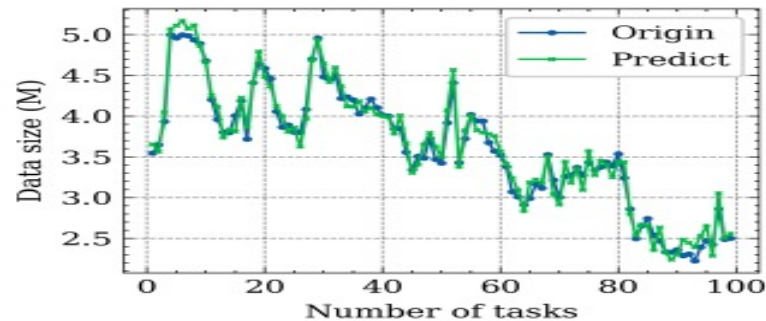
- A T times history window is used to predict the task at $T + 1$ times.
- Set the history window to 50
- Set different thresholds for the optimization target $|D_t - \tilde{D}_t|$.
- The experimental results are shown in Figure .

Observations:

- When the threshold value is set small, the LSTM prediction model describes the historical data volume with higher accuracy and can fully explore the changing pattern of the data volume
- However, it will introduce a larger prediction overhead, such as will increase the training time of the LSTM model.



(a)



(b)

Effect of threshold size on LSTM prediction task features. (a) Threshold size = 0.5 M; (b) Threshold size = 0.1 M.

Experiment: Training Process of LSTM & DRL

- When performing training on the DRL offload decision model, it takes a longer time to explore and select the better result due to the initial random selection action of the agent.
- We predict the server load based on the edge server record history data
- Based on the prediction results, the server predicted to be non-idle is selected with a certain probability as the offload choice for the next moment
- This solution allows the agent to effectively avoid selecting servers with high loads, thus reducing task processing latency and task dropping rates.
- We use LSTM for load prediction and compare the impact of decisions with load prediction (LSTM & DRL) and without load prediction (DRL) on offloading performance.
- As result, DRL is significantly slower than the LSTM & DRL for load prediction in the early stages of training decision making
- after certain training, the average delay, energy consumption, and the number of task throw volumes is reduced rapidly by using LSTM for load prediction

Experiment: Results

Impact of the Tasks Number

- We use different time slots to verify the impact of the number of tasks on the system cost, average task delay, and task discard rate.
- Set the time slots in the dataset to $T = 100, 200, 500, 1000$, and compare the performance of DQN, double DQN, dueling DQN, and OPO under different time slots
- As the running time of the system increases (i.e., the number of tasks increases), OPO reduces at least 6.25% of the average latency, 25.6% of the offloading cost, and 31.7% of the task drop rate compared to other algorithms in terms of cost, average latency, and task dropped rate.

Impact of the Learning Rate

- We study the convergence of the algorithm at different learning rates (denoted as lr)
- when $lr = 0.001$, the algorithm is able to achieve a relatively fast convergence rate and a small convergence cost
- As the learning rate decreases (i.e., below 0.0001), the convergence is slower and takes longer to reach a better value.
- When the learning rate is larger, the convergence cost increases and may even be higher than that of the stochastic strategy.

LSTM Prediction: Numerical

Suppose you are managing a data center that provides cloud computing services to customers. You want to use an LSTM model to forecast the hourly CPU utilization of the data center for the next 24 hours in order to optimize resource allocation and minimize energy consumption.

You have a dataset with hourly CPU utilization data for the past year, which contains 8,760 data points. You decide to use the first 7,000 data points for training and the remaining 1,760 data points for validation. You set the batch size to 64 and the number of epochs to 50.

Assuming the model takes 5 seconds to process one batch of data on a GPU, how long will it take to train the model?

Note: This question assumes that the data has already been preprocessed and formatted for input into the LSTM model.

LSTM Prediction: Numerical

Solution:

The time it will take to train the model can be calculated as follows:

- Batch size = 64
- Number of training data points = 7,000
- Number of epochs = 50
- Number of iterations per epoch = Number of training data points / Batch size = $7,000 / 64$
= 109.375 = ~109 (rounded down to nearest integer)

Total number of iterations = Number of epochs x Number of iterations per epoch = 50×109
= 5,450

Time taken to process one batch of data on a GPU = 5 seconds

Total time taken to train the model = Time taken per iteration x Total number of iterations =
(5 seconds x Batch size) x Total number of iterations
= (5 seconds x 64) x 5,450 = 1,760,000 seconds = ~20.4 days (rounded to 1 decimal place)

Therefore, it will take approximately 20.4 days to train the LSTM model using the given dataset, batch size, and number of epochs.

Summary

- The lecture considers the computational offloading problem in edge computing.
- The optimization objective is to minimize long-term cost by jointly optimizing task latency, energy consumption, and discard rate.
- The model combines the prediction method of LSTM networks and the decision method of DQN.
- An OPO algorithm based on deep reinforcement learning is proposed, combining the advantages of Double DQN and Dueling DQN.
- The training speed and accuracy of the DRL model are improved using LSTM's prediction capability.
- The proposed algorithm reduces the offloading decision delay of tasks in the actual inference process.
- Future research will migrate the method to experimental testing and combine the latest algorithms and techniques to improve its performance in real IoT application scenarios.

Thank You

Vertical and Horizontal Offloading for Cloud-Edge



Dr. Rajiv Misra

Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss a generic architecture of cloud-edge computing with the aim of providing both vertical and horizontal offloading between service nodes.
- An approximation algorithm which applies a branch-and-bound method to obtain optimal solutions iteratively.

Cloud-Edge Computing Environment

Introduction:

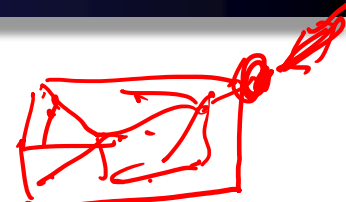
- Edge computing is a paradigm that enables virtualized computational and communication resources to be deployed near the source of service workloads instead of relying on massive data centers.
- This allows for a reduction in end-to-end delay for accessing these resources, and makes it more suitable for real-time or delay-sensitive services.
- Additionally, edge computing enables virtualized resources to be geographically distributed which can address the requirements of mobility and geo-distribution of mobile and IoT services.

Cloud- Edge Computing: ✓

- Cloud-edge computing can efficiently accommodate different types of services, with end devices and network edges suitable for real-time and delay-sensitive services and central offices and data centers able to handle services which require a large amount of computing capacity.
- Integration of cloud and edge computing is proposed to take advantage of the benefits both technologies offer.

Cloud-Edge Computing Environment

Example:



- A smart home system that utilizes edge computing could provide a more secure, efficient, and cost-effective solution for controlling and monitoring devices such as lights, thermostats, cameras, and door locks.
- The system would have a gateway device, such as a router, that would provide a local connection for each device.
- The gateway would run a virtualized instance of a cloud application, allowing for local processing of data and commands.
- This would reduce the latency for any commands sent to the devices, providing a more responsive system.
- Additionally, all data would be stored on the local gateway, providing a more secure solution than if the data were stored in a cloud.

Cloud-Edge Computing Environment

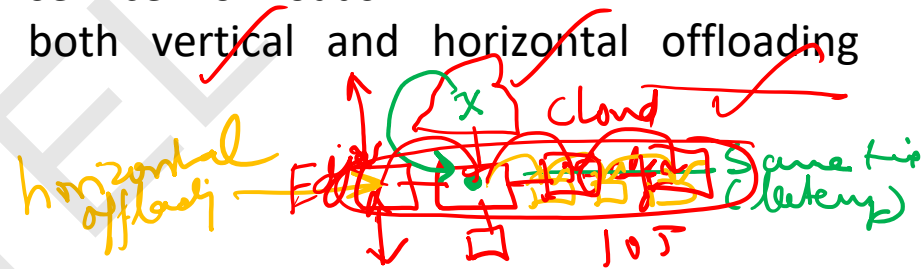
- The concept of cloud-edge computing is an effective way to manage and guarantee the quality of services while efficiently managing capital and operating expenses. Research has been conducted to address the requirements of cloud-edge computing in order to meet the increasing demand for service workloads.
- Cloud-edge computing should consider both vertical and horizontal offloading between service nodes.

① Vertical Offloading :

- Vertical offloading refers to the process of transferring tasks or services from cloud or datacenters to edge nodes in order to reduce latency or increase efficiency. It is also known as cloud-edge computing and is used to reduce the burden on the cloud.

② Horizontal Offloading :

- Horizontal offloading, on the other hand, is the process of transferring tasks or services between edge nodes in order to reduce latency or increase efficiency. It is used to improve the capacity of edge nodes and can also be used to reduce the load on the cloud.

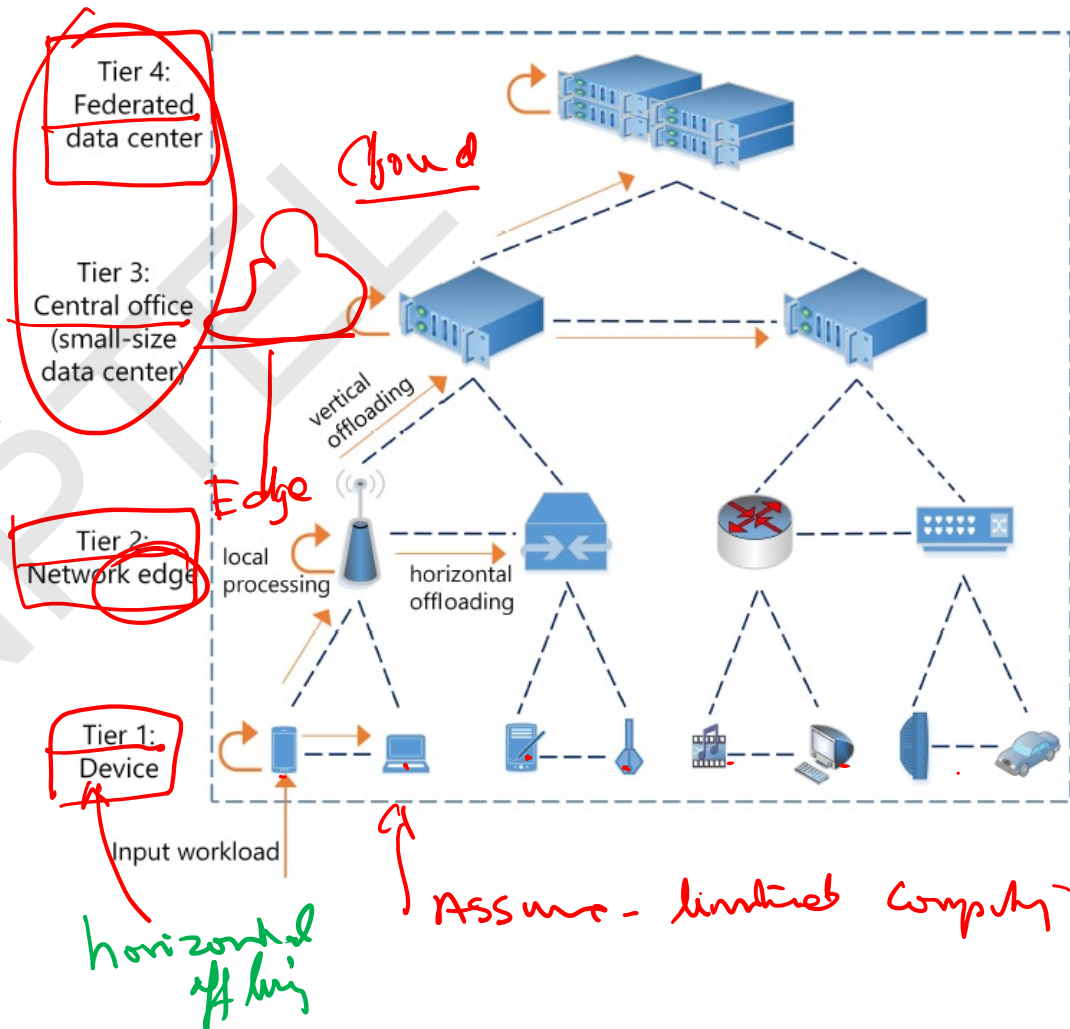


Architecture of Collaborative Cloud-Edge Computing

The novel aspect of the design is that it deploys virtualized communication and computation services to four different hierarchical tiers.

1. First Tier:

- The first tier of the hierarchy is composed of end devices, such as smartphones, IP cameras and IoT sensors, which directly receive service workloads from their sources.
- A device can by itself locally process a fraction of the input workloads or horizontally offload some of the other workloads to neighboring devices, using various short-range wireless transmission techniques such as LTE D2D, Wi-Fi Direct, ZigBee, and Bluetooth.



Architecture of Collaborative Cloud-Edge Computing

This generic architecture is designed to provide a framework for building and deploying different types of services.

For example:

In the case of a vehicle congestion avoidance service in a smart city:

- IP cameras are used to monitor traffic and detect abnormal behavior that might indicate an emergency event.
- The data captured by the cameras is then sent to an edge server for further analysis and processing.
- The server can then send the refined data to drivers or news outlets throughout the city.
- If there is a lack of computational power, the data can be redirected to other edge servers or even to a remote data center.

The proposed architecture is designed to be flexible and customizable, allowing service nodes to be merged or removed as needed. This flexibility allows for specific architectures to be built and deployed, such as Edge server, Coordinate device, and Device cloud. These architectures are designed to accommodate different types of cloud-edge services and applications.

Architecture of Collaborative Cloud-Edge Computing

1) Workload Model: (hand-edge)

Let $f \in F$ denote an offered service of a cloud-edge computing system. Each service f has a computation size Z_f^S which is the number of mega CPU cycles required to process a request for service f . Also, communication size Z_f^N indicates the data size of the request in megabytes.

Let I^α , I^β , I^γ , and I^δ be the sets of devices, network edges, central offices and data centers of the system, respectively. A service node $i \in I$ could process a set of services $F_i \subseteq F$, where I is the set of all service nodes of the system, i.e., $I = I^\alpha \cup I^\beta \cup I^\gamma \cup I^\delta$.

a) Local processing:

Let p_i^f denote the workload (in requests per second) of a service f which is locally processed by a node i . We have

$$p_i^f = \begin{cases} \geq 0, & \text{if } f \in F_i, \forall i \in I \\ = 0, & \text{if } f \notin F_i, \forall i \in I \end{cases}$$

Architecture of Collaborative Cloud-Edge Computing

b) Sibling node and horizontal offloading:

The set of siblings H_i of a node $i \in I$ consists of service nodes which are located in the same tier as i , and to which i can horizontally offload its workloads. Also, let $x_{i,j}^f$ be the workload of a service f which is horizontally offloaded from i to a service node $j \in H_i$.

Similarly, let $u_{j,i}^f$ be the workload of a service f which is horizontally offloaded from $j \in H_i$ to i . Here, we assume that a service node i can offload the workload of a service f to a sibling node j on condition that j is able to process

f , i.e., $f \in F_j$. In addition, to prevent loop situations, a node cannot receive the workloads of a service f from its siblings if it already horizontally offloads this type of workload.

Thus, we have

$$x_{i,j}^f = \begin{cases} \geq 0, & \text{if } f \in F_j, \forall j \in H_i, \forall i \in I, \\ = 0, & \text{if } f \notin F_j, \forall j \in H_i, \forall i \in I, \end{cases}$$

$$u_{j,i}^f = \begin{cases} \geq 0, & \text{if } f \in F_j, \forall j \in H_i, \forall i \in I, \\ = 0, & \text{if } f \notin F_j, \forall j \in H_i, \forall i \in I, \end{cases}$$

Architecture of Collaborative Cloud-Edge Computing

c) Parent/child node and vertical offloading:

The set of parents V_i of a service node $i \in I$ consists of the nodes located in the next tier up with i , and to which i can vertically offload its workloads. Let $y_{i,j}^f$ be the workload of a service f which is vertically offloaded from i to a node $j \in V_i$.

The set of children K_i of i consists of the nodes which are located in the right lower tier with i , and from which i receives incoming workloads. Let $v_{j,i}^f$ denote the workload of a service f which is vertically offloaded from $j \in K_i$ to i . Since a device $i \in I^\alpha$ directly receives service workloads from external sources, it has no child nodes, i.e., $K_i = \emptyset, \forall i \in I^\alpha$.

Similarly, a data center $i \in I^\delta$ is in the most-top tier of the system, and hence has no parent nodes, i.e., $V_i = \emptyset, \forall i \in I^\delta$.

Opposed to horizontal offloading, a service node can carry out vertical offloading for all services $f \in F$. In other words, it can dispatch all types of workloads to its parents. Thus, we have

$$y_{i,j}^f \geq 0, \forall f \in F, \forall j \in V_i, \forall i \in I^\alpha \cup I^\beta \cup I^\gamma$$

$$v_{j,i}^f \geq 0, \forall f \in F, \forall j \in K_i, \forall i \in I^\beta \cup I^\gamma \cup I^\delta.$$

Let λ_i^f denote the submitted workload of a service f from external sources to a device $i \in I^\alpha$. We have

$$\lambda_i^f \geq 0, \forall f \in F, \forall i \in I^\alpha.$$

Architecture of Collaborative Cloud-Edge Computing

2) Computation and Communication Delay:

Computation and Communication Delay consists of:

- a) Computation delay of device and edge nodes
- b) Computation delay of central office and data center Nodes
- c) Communication delay of network connections
- d) Computation and communication delay of the cloud-edge computing system

3) System total cost:

The total system cost C of a cloud-edge computing is defined as

$$C = C^S + C^N$$


Where C^S is Computation cost of service nodes and C^N is Communication cost of network connections.

Since we aim to minimize the total cost of the cloud-edge computing system while guaranteeing its delay constraints, we hence have an optimization problem.

Algorithm: Branch-and-Bound With Parallel Multi-Start Search Points

- We try to solve a problem (P) which has variables that are integers and nonlinear delay constraints.
- This type of problem is usually very hard to solve, so we are using the Branch-and-bound algorithm.
- We search the tree looking for solutions with integers and when we find one, we use it as an upper bound for the original problem.
- We keep searching until all the nodes of the tree have been solved or the search conditions have been met.

ALGORITHM DESIGN - BRANCH-AND-BOUND WITH PARALLEL MULTI-START SEARCH POINTS

1. Attempt to find an initial solution by applying a Feasibility Pump relaxation heuristic
2. If a feasible solution $C^*(N^*, O^*)$ is reached, set it to the current optimal solution $C(N, O)$
3. Add an NLP sub-problem SP, generated by removing the integrality conditions of variables n_i of the problem P, to the tree data structure T
4. Start the branch-and-bound procedure  iteratively solve the sub-problem SP using Interior/Direct algorithm with parallel multiple initial searching points
5. If a feasible solution $C^*(N^*, O^*)$ is smaller than the current optimal solution $C(N, O)$ and N^* are integers, set $C^*(N^*, O^*)$ to the current optimal solution and prune the node SP, removing it and its sub-nodes from T
6. If N^* is not an integer, perform a branching operation on a variable $n_i \in N^*$ creating two new sub-problems SSP1 and SPP2 of SP, added to T using the Pseudo-cost branching method
7. If $C^*(N^*, O^*)_i \geq C(N, O)$, or there is not a feasible solution, prune the node SP
8. Repeat the branch-and-bound procedure until all nodes of T have been resolved

ALGORITHM DESIGN - BRANCH-AND-BOUND WITH PARALLEL MULTI-START SEARCH POINTS

```
1  $\mathbb{C}(\mathcal{N}, \mathcal{O}) \leftarrow \infty, \mathcal{T} \leftarrow \emptyset;$   
   /* Attempt to find an initial solution */  
2 Solve  $\mathcal{P}$  by Feasibility Pump heuristic;  
3 if find a feasible solution  $\mathbb{C}^*(\mathcal{N}^*, \mathcal{O}^*)$  then  
4   |  $\mathbb{C}(\mathcal{N}, \mathcal{O}) \leftarrow \mathbb{C}^*(\mathcal{N}^*, \mathcal{O}^*);$   
5 end  
   /* Begin branch-and-bound procedure */  
6  $SP \leftarrow$  Relax integrality constraints of  $\mathcal{P}$ ;  
7  $Add(\mathcal{T}, SP);$   
8 while  $\exists SP \in \mathcal{T}$  has not been reached or pruned do  
9   | Select a subproblem  $SP \in \mathcal{T}$  by depth-first strategy;  
10  | Solve  $SP$  by Parallel Multi-start Interior/Direct  
   | algorithm;  
11  | if find a feasible solution  $\mathbb{C}^*(\mathcal{N}^*, \mathcal{O}^*) < \mathbb{C}(\mathcal{N}, \mathcal{O})$  then  
12  |   | if  $\mathcal{N}^* \in \mathbb{N}$  then  
13  |   |   |  $\mathbb{C}(\mathcal{N}, \mathcal{O}) \leftarrow \mathbb{C}^*(\mathcal{N}^*, \mathcal{O}^*);$   
14  |   |   |  $Prune(\mathcal{T}, SP);$   
15  |   | else  
16  |   |   | Create 2 subproblem nodes  $SSP1, SSP2$  of  
   |   |   |  $SP$  by Pseudo-cost branching on a  $n_i \in \mathcal{N}^*$ ;  
17  |   | end  
18  | else  
19  |   |  $Prune(\mathcal{T}, SP);$   
20  | end  
21 end
```

Experiment:

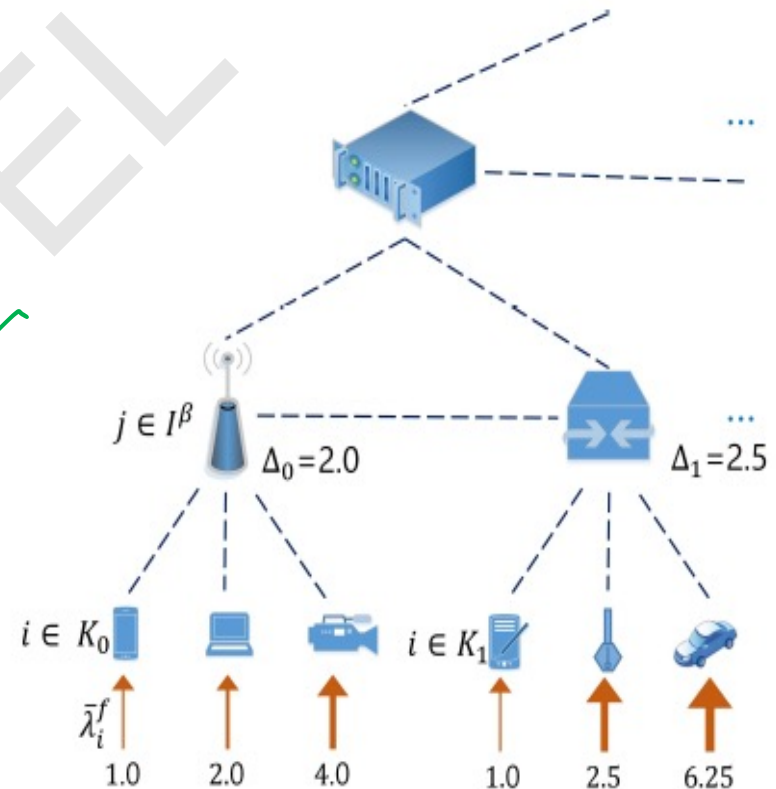
1. Summarize the cloud-edge computing system and its parameters.
2. Compare the cloud-edge computing system with a traditional design (NH) which does not support horizontal offloading.
3. Adjust the arrival rate to generate workloads whose total demanded computation capacity is 10%, 50%, and 100% of the maximum capacity of all service nodes.
4. Optimize the system to minimize the total system cost C which consists of the computation cost of service nodes and the communication cost of network connections.
5. Present results of other metrics such as computation capacity allocation, workload allocation, and horizontal offloading workloads.

Analysis of the result:

1. Evaluate performance of cloud-edge computing architecture design and traditional design in unbalanced and balanced workload scenarios.

Unbalanced Workload:

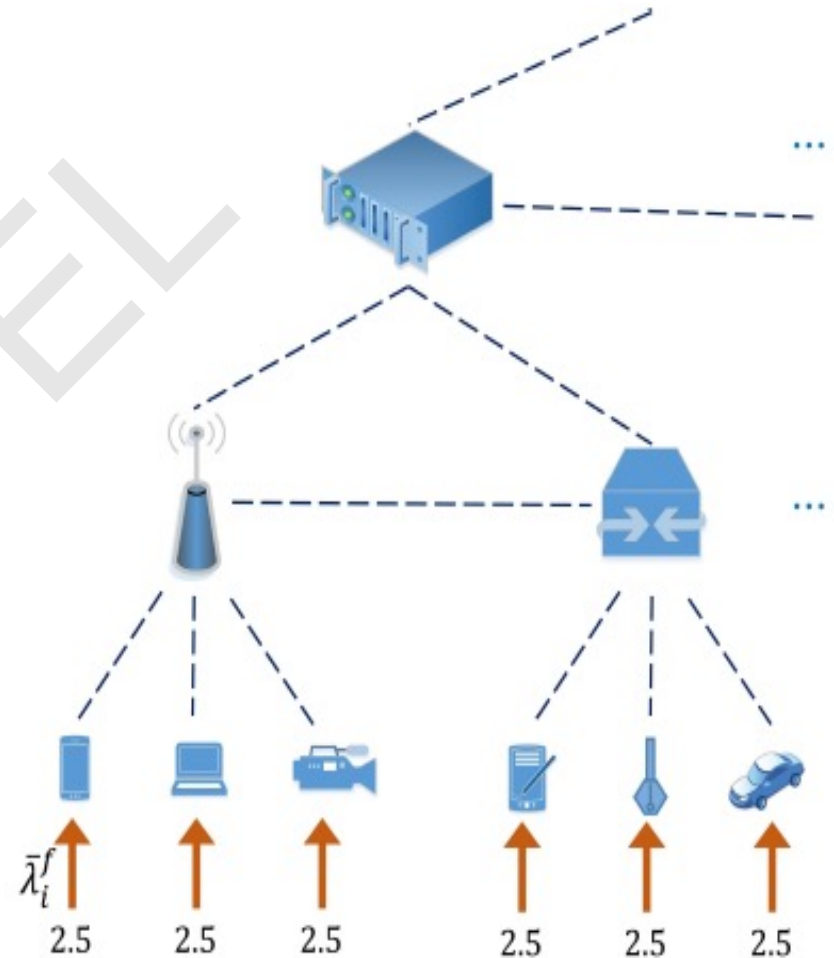
- Unbalanced input workload scenarios refer to scenarios where incoming workloads are not evenly distributed across cloud computing and edge computing resources.
- This could occur due to a sudden spike in requests from one geographical location or due to a particular type of workload that is more suited to being processed locally at the edge.
- In such cases, the cloud resources may be overloaded, leading to degraded performance, while the edge resources may be underutilized.



Analysis of the result:

Balanced Workload:

- Balanced input workload scenarios, on the other hand, refer to scenarios where incoming workloads are evenly distributed across cloud computing and edge computing resources.
- This can be achieved through careful planning, careful monitoring of incoming workloads and the use of intelligent algorithms to route the workloads to the most appropriate resources.
- This ensures that both cloud and edge resources are being utilized efficiently, leading to improved performance and cost savings.



Analysis of the result:

✓
2. Test two service allocation strategies: homogeneous and heterogeneous.

A. Homogeneous Service Allocation Scenario: ✓

- In a homogeneous service allocation scenario, services are allocated to the same type of cloud-edge computing environment and resources.
- This means that the same type of hardware and software is used across all the cloud-edge sites.
- This type of scenario is useful when the same types of applications are running across multiple sites or when the same types of services need to be provided.
- For example, if the same type of virtual machine is allocated to different tasks on the cloud and edge, then it would be a homogeneous service allocation scenario. ✓

Analysis of the result:

B. Heterogeneous Service Allocation Scenario:

- In a heterogeneous service allocation scenario, services are allocated to different types of cloud-edge computing environments and resources.
- This means that different types of hardware and software are used across different cloud-edge sites.
- This type of scenario is useful when different types of applications are running across multiple sites or when different types of services need to be provided.
- This type of scenario also allows for more flexibility in the types of resources that can be used, allowing for a more customized experience for each site.
- For example, if different types of virtual machines are allocated to different tasks on the cloud and edge, then it would be a heterogeneous service allocation scenario.

Analysis of the result:

3. Observe impact of different computation capacity costs on cloud-edge computing architecture design and traditional design:
 - The impact of different computation capacity costs on cloud-edge computing architecture design and traditional design is largely based on the cost efficiency of the solution.
 - Cloud-edge computing architectures typically provide more cost-efficient solutions than traditional designs, as they leverage the cost-effectiveness of the cloud while providing more localized processing power.
 - For example, if computation capacity costs are high, cloud-edge computing architectures can be more cost-effective by utilizing the cloud for its cost-effectiveness and leveraging localized processing power for more efficiency.
 - This allows for cost savings in both cloud and edge compute costs, as cloud capacity is leveraged for less expensive compute and edge compute resources can be used as needed to meet performance and latency requirements.

Thank You

Global State and Snapshot Recording Algorithms



Dr. Rajiv Misra, Professor

**Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in**

Preface

Content of this Lecture:

- In this lecture, we will discuss about the Global states (i.e. consistent, inconsistent), Models of communication and Snapshot algorithm *i.e.* Chandy-Lamport algorithm to record the global snapshot.

Snapshots

Here's Snapshot: Collect at a place



Distributed Snapshot

How do you calculate a “global snapshot” in this distributed system?

What does a “global snapshot” even mean?



→ 105

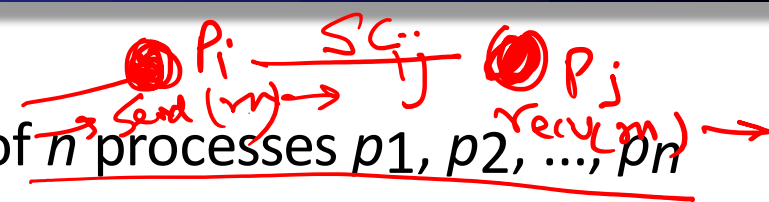
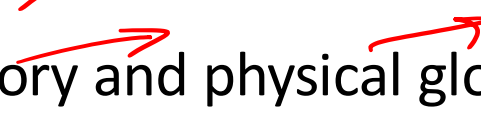
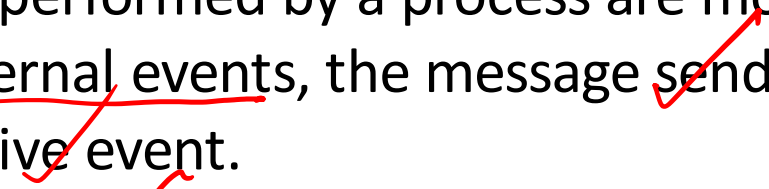
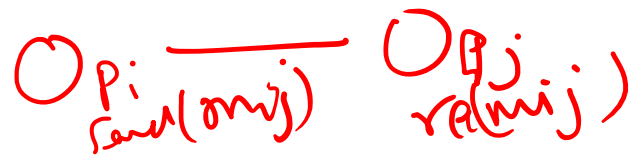
In the Cloud: Global Snapshot

- In a cloud each application or service is running on multiple servers
- Servers handling concurrent events and interacting with each other
- The ability to obtain a “global photograph” or “Global Snapshot” of the system is important
- Some uses of having a global picture of the system
 - **Checkpointing**: can restart distributed application on failure
 - **Garbage collection of objects**: objects at servers that don't have any other objects (at any servers) with pointers to them
 - **Deadlock detection**: Useful in database transaction systems
 - **Termination of computation**: Useful in batch computing systems

Global State: Introduction

- **Recording the global state** of a distributed system on-the-fly is an important paradigm.
- The **lack of globally shared memory, global clock and unpredictable message delays** in a distributed system make this problem non-trivial.
- This lecture first defines consistent global states and discusses issues to be addressed to compute consistent distributed snapshots.
- Then the algorithm to determine on-the-fly such snapshots is presented.

System Model

- The system consists of a collection of n processes p_1, p_2, \dots, p_n that are connected by channels. 
- There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels. 
- C_{ij} denotes the channel from process p_i to process p_j and its state is denoted by SC_{ij} .
- The actions performed by a process are modeled as three types of events: Internal events, the message send event and the message receive event. 
- For a message m_{ij} that is sent by process p_i to process p_j , let $send(m_{ij})$ and $rec(m_{ij})$ denote its send and receive events. 

System Model

P_i \downarrow time LS_i

- At any instant, the state of process p_i , denoted by LS_i , is a result of the sequence of all the events executed by p_i till that instant.
- For an event e and a process state LS_i , $e \in LS_i$ iff e belongs to the sequence of events that have taken process p_i to state LS_i .
- For an event e and a process state LS_i , $e \notin LS_i$ iff e does not belong to the sequence of events that have taken process p_i to state LS_i .
- For a channel C_{ij} , the following set of messages can be defined based on the local states of the processes p_i and p_j

Transit: $transit(LS_i, LS_j) = \{m_{ij} \mid send(m_{ij}) \in LS_i \wedge rec(m_{ij}) \notin LS_j\}$

Consistent Global State

- The global state of a distributed system is a collection of the local states of the processes and the channels.

- Notationally, global state GS is defined as,

$$GS = \{U_i LS_i, U_{i,j} SC_{ij}\}$$

- A global state GS is a **consistent global state** iff it satisfies the following two conditions:

C1: $\text{send}(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus \text{rec}(m_{ij}) \in LS_j$
(\oplus is Ex-OR operator)

C2: $\text{send}(m_{ij}) \notin LS_i \Rightarrow m_{ij} \notin SC_{ij} \wedge \text{rec}(m_{ij}) \notin LS_j$

Global State of a Distributed System



- In the distributed execution of Figure 6.2:
- A global state GS_1 consisting of local states $\{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ is **inconsistent** because the state of p_2 has recorded the receipt of message m_{12} , however, the state of p_1 has not recorded its send.
- On the contrary, a global state GS_2 consisting of local states $\{LS_1^2, LS_2^4, LS_3^4, LS_4^2\}$ is **consistent**; all the channels are empty except c_{21} that contains message m_{21} .

Global State of a Distributed System

- A global state $GS = \{U_i LS_i^{xi}, U_{j,k} SC_{jk}^{yj,zk}\}$ is transitless iff
$$\forall i, \forall j : 1 \leq i, j \leq n :: SC_{jk}^{yj,zk} = \emptyset$$
- Thus, all channels are recorded as empty in a transitless global state. A global state is **strongly consistent** iff it is transitless as well as consistent. Note that in figure 6.2, the global state of local states $\{LS_1^2, LS_2^3, LS_3^4, LS_4^2\}$ is **strongly consistent**.
- Recording the global state of a distributed system is an important paradigm when one is interested in analyzing, monitoring, testing, or verifying properties of distributed applications, systems, and algorithms. Design of efficient methods for recording the global state of a distributed system is an important problem.

Example:

Time

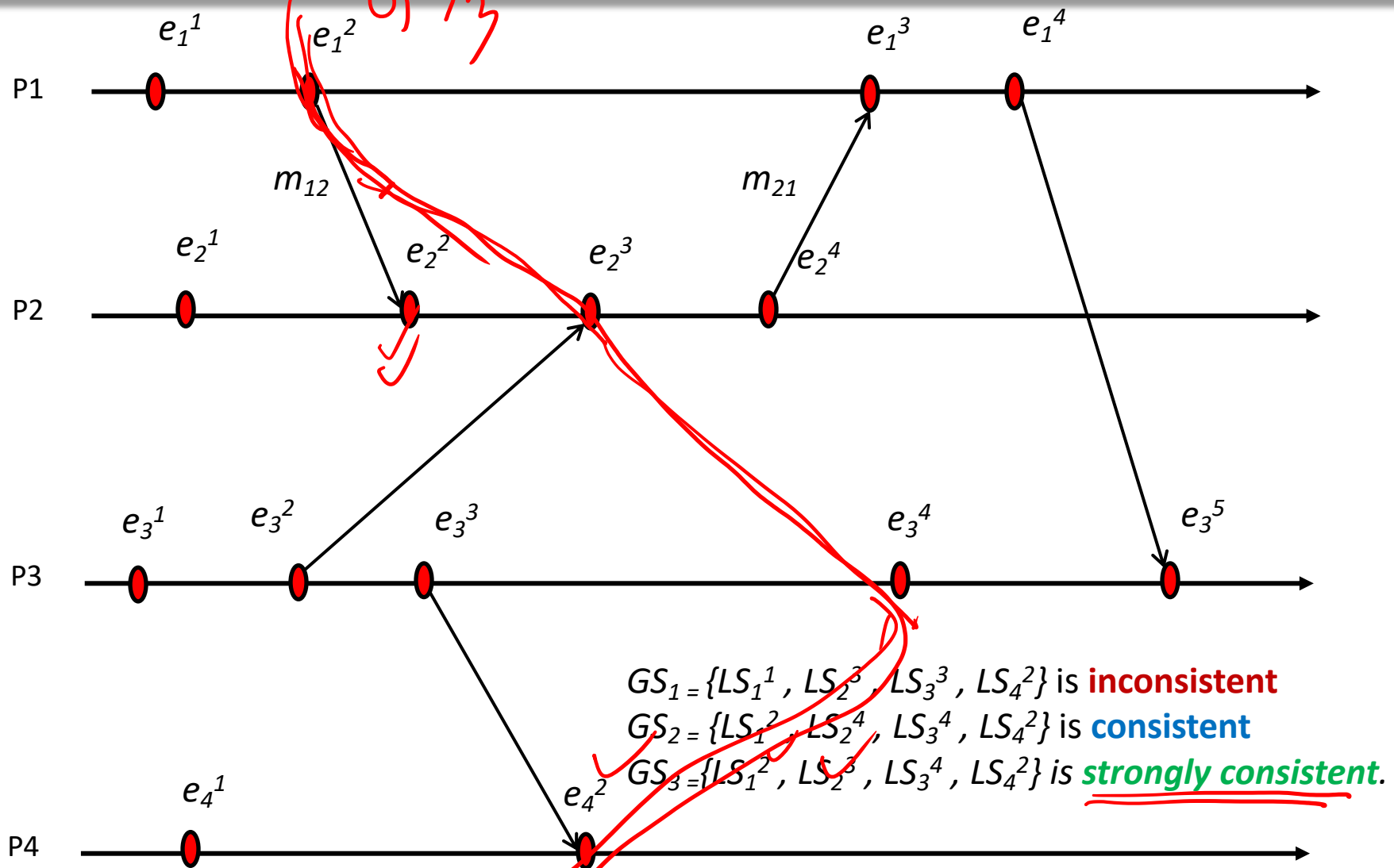


Figure 6.2: The space-time diagram of a distributed execution.

Issues in Recording a Global State

- The following two issues need to be addressed:

I1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.

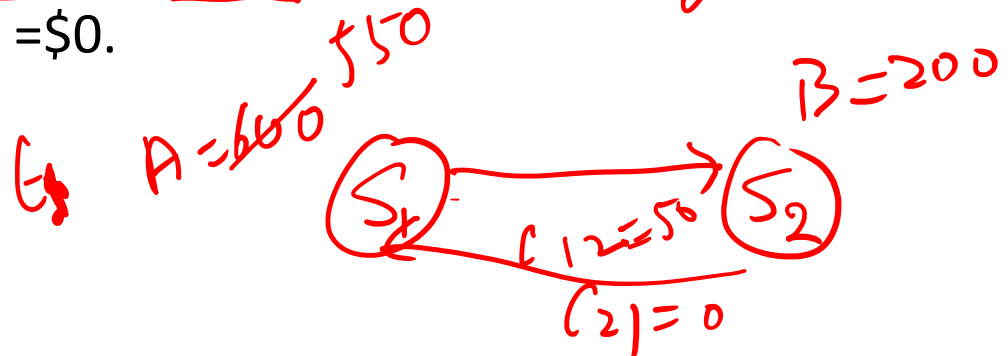
- -Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from **C1**).
- -Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from **C2**).

I2: How to determine the instant when a process takes its snapshot.

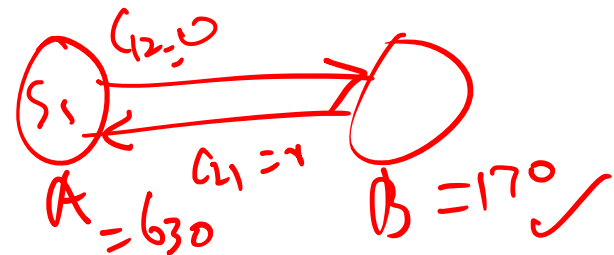
- A process p_j must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its snapshot.

Example of Money Transfer

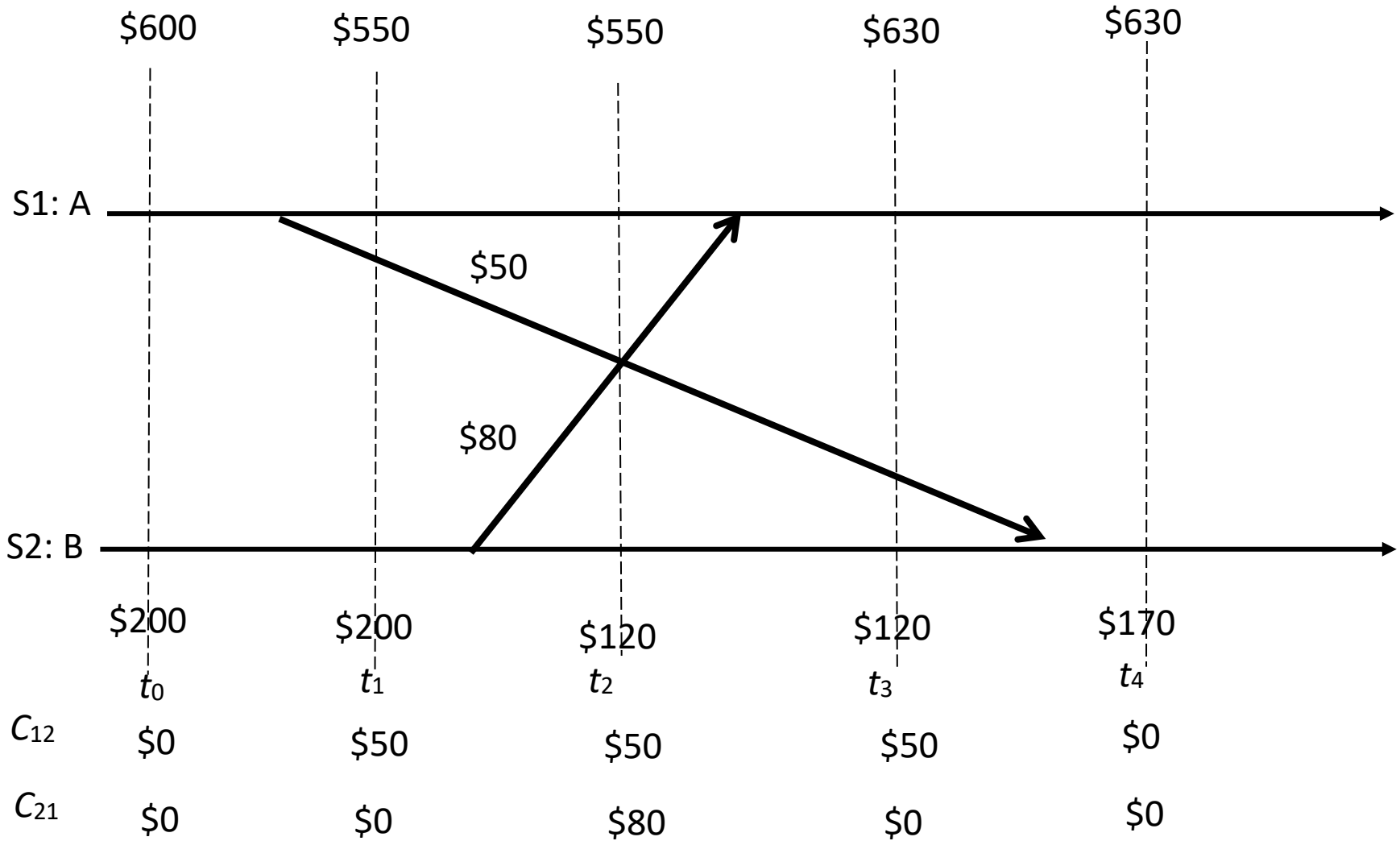
- Let S_1 and S_2 be two distinct sites of a distributed system which maintain bank accounts A and B, respectively. A site refers to a process in this example. Let the communication channels from site S_1 to site S_2 and from site S_2 to site S_1 be denoted by C_{12} and C_{21} , respectively. Consider the following sequence of actions, which are also illustrated in the timing diagram of Figure 6.3:
- Time t_0 : Initially, Account A=\$600, Account B=\$200, $C_{12} = \$0$, $C_{21} = \$0$.
- Time t_1 : Site S_1 initiates a transfer of \$50 from Account A to Account B.
- Account A is decremented by \$50 to \$550 and a request for \$50 credit to Account B is sent on Channel C_{12} to site S_2 . Account A=\$550, Account B=\$200, $C_{12} = \$50$, $C_{21} = \$0$.



- Time t_2 : Site S2 initiates a transfer of \$80 from Account B to Account A.
- Account B is decremented by \$80 to \$120 and a request for \$80 credit to Account A is sent on Channel C_{21} to site S1. Account A=\$550, Account B=\$120, $C_{12} = \$50$, $C_{21} = \$80$.
- Time t_3 : Site S1 receives the message for a \$80 credit to Account A and updates Account A. Account A=\$630, Account B=\$120, $C_{12} = \$50$, $C_{21} = \$0$.
- Time t_4 : Site S2 receives the message for a \$50 credit to Account B and updates Account B. Account A=\$630, Account B=\$170, $C_{12} = \$0$, $C_{21} = \$0$.



**T_3 : Site S1 receives the message for a
\$80 credit to Account A and updates**



**T_4 : Site S2 receives the message for a \$50
credit to Account B and updates Account B**

- Suppose the local state of Account A is recorded at time t_0 to show \$600 and the local state of Account B and channels C_{12} and C_{21} are recorded at time t_2 to show \$120, \$50, and \$80, respectively. Then the recorded global state shows \$850 in the system. An extra \$50 appears in the system.
- **The reason for the inconsistency** is that Account A's state was recorded before the \$50 transfer to Account B using channel C_{12} was initiated, whereas channel C_{12} 's state was recorded after the \$50 transfer was initiated.
- This simple example shows that recording a consistent global state of a distributed system is not a trivial task. Recording activities of individual components must be coordinated appropriately.

Model of Communication

- Recall, there are three models of communication: FIFO, non-FIFO, and Co.

In **FIFO model**, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.

In **non-FIFO model**, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.

- A system that supports **causal delivery** of messages satisfies the following property: “For any two messages m_{ij} and m_{kj} ,

if $send(m_{ij}) \rightarrow send(m_{kj})$, then $rec(m_{ij}) \rightarrow rec(m_{kj})$ ”



Snapshot algorithm for FIFO channels

Chandy-Lamport algorithm:

- The **Chandy-Lamport** algorithm uses a **control message**, called a **marker** whose role in a **FIFO system** is to separate messages in the channels.
- After a site has recorded its snapshot, it sends a **marker**, along all of its outgoing channels before sending out any more messages.
- A marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot.
- A process must record its snapshot no later than when it receives a marker on any of its incoming channels.

Chandy-Lamport Algorithm



- The algorithm can be initiated by any process by executing the “**Marker Sending Rule**” by which it records its local state and sends a marker on each outgoing channel.
- A process executes the “**Marker Receiving Rule**” on receiving a marker. If the process has not yet recorded its local state, it records the state of the channel on which the marker is received as empty and executes the “Marker Sending Rule” to record its local state.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

Chandy-Lamport Algorithm

✓ **Marker Sending Rule** for process i

- ✓ 1) Process i records its state.
- ✓ 2) For each outgoing channel C on which a marker has not been sent, i sends a marker along C before i sends further messages along C .

✓ **Marker Receiving Rule** for process j

On receiving a marker along channel C :

if j has not recorded its state **then**

Record the state of C as the empty set ✓

Follow the "Marker Sending Rule" ✓

else

- ✓ Record the state of C as the set of messages received along C after j 's state was recorded and before j received the marker along C

Properties of the recorded global state

- The recorded global state may not correspond to any of the global states that occurred during the computation.
- Consider two possible executions of the snapshot algorithm (shown in Figure 6.4) for the previous money transfer example .

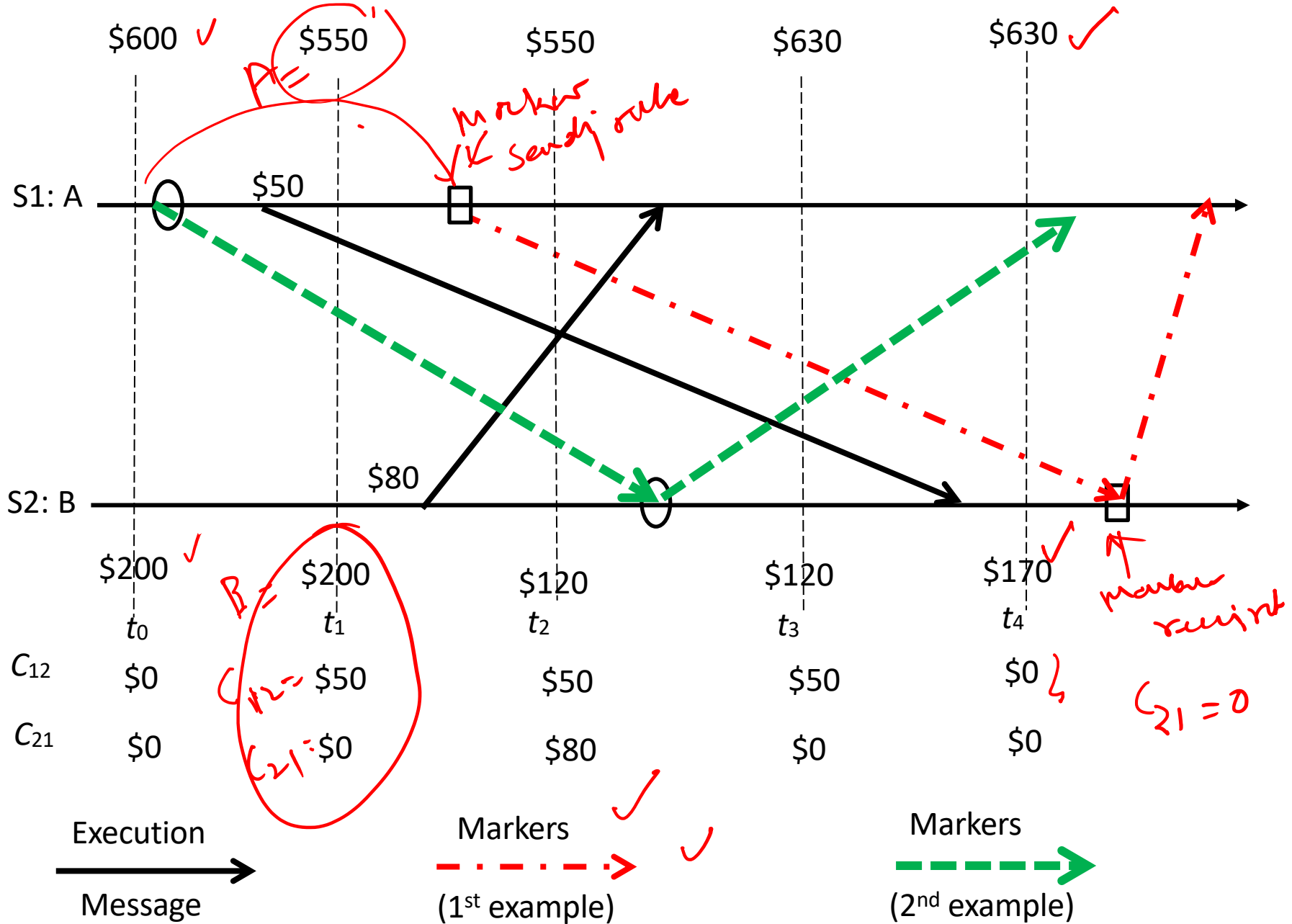


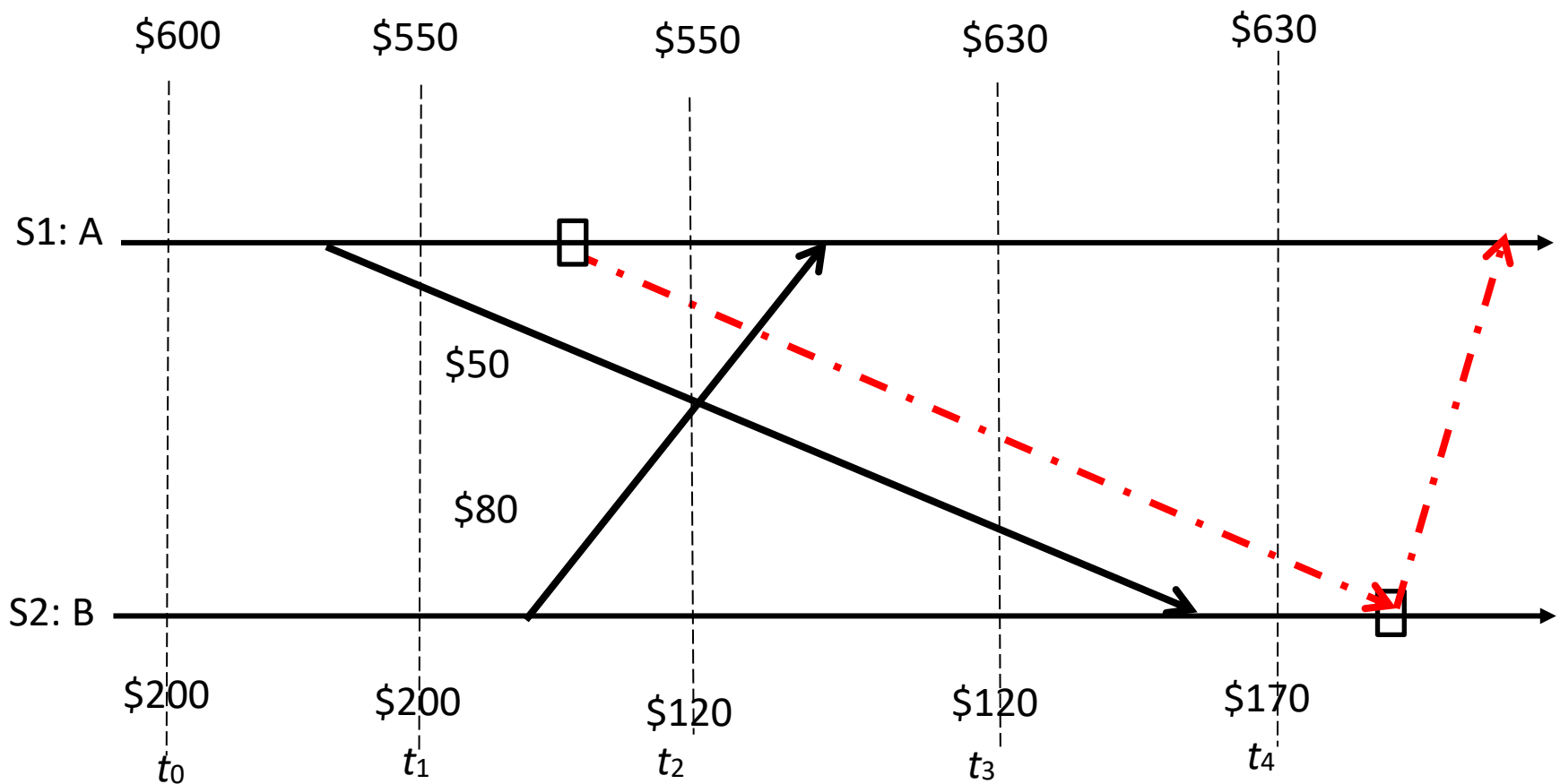
Figure 6.4: Timing diagram of two possible executions of the banking example

Properties of the recorded global state

1. (Markers shown using red dashed-and-dotted arrows.)

Let site S1 initiate the algorithm just after t_1 . Site S1 records its local state (account A=\$550) and sends a marker to site S2. The marker is received by site S2 after t_4 . When site S2 receives the marker, it records its local state (account B=\$170), the state of channel C_{12} as \$0, and sends a marker along channel C_{21} . When site S1 receives this marker, it records the state of channel C_{21} as \$80. The \$800 amount in the system is conserved in the recorded global state,

$$A = \$550, B = \$170, C_{12} = \$0, C_{21} = \$80$$



A = \$550

B = \$170

$C_{12} = \$0$

$C_{21} = \$80$

The \$800 amount in the system is conserved in the recorded global state

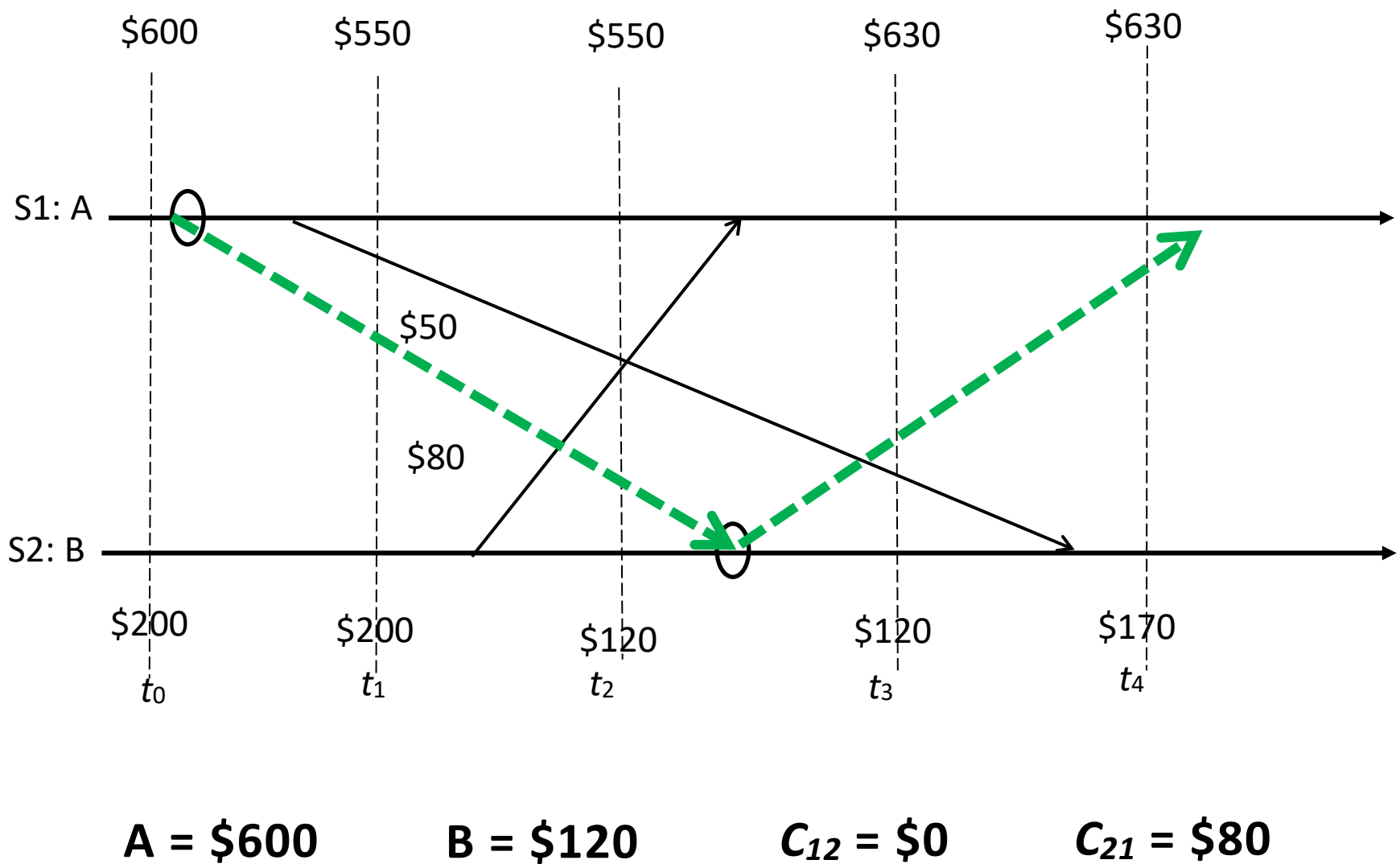
Figure 6.4: Timing diagram of two possible executions of the banking example

Properties of the recorded global state

2. (Markers shown using green dotted arrows.)

Let site S1 initiate the algorithm just after t_0 and before sending the \$50 for S2. Site S1 records its local state (account A = \$600) and sends a marker to site S2. The marker is received by site S2 between t_2 and t_3 . When site S2 receives the marker, it records its local state (account B = \$120), the state of channel C_{12} as \$0, and sends a marker along channel C_{21} . When site S1 receives this marker, it records the state of channel C_{21} as \$80. The \$800 amount in the system is conserved in the recorded global state,

$$A = \$600, B = \$120, C_{12} = \$0, C_{21} = \$80$$



The \$800 amount in the system is conserved in the recorded global state

Figure 6.4: Timing diagram of two possible executions of the banking example

Properties of the recorded global state

- In both these possible runs of the algorithm, the recorded global states never occurred in the execution.
- This happens because a process can change its state asynchronously before the markers it sent are received by other sites and the other sites record their states.
 - But the system could have passed through the recorded global states in some equivalent executions.
 - The recorded global state is a valid state in an equivalent execution and if a stable property (i.e., a property that persists) holds in the system before the snapshot algorithm begins, it holds in the recorded global snapshot.
- Therefore, a recorded global state is useful in detecting stable properties.

Conclusion

- Recording global state of a distributed system is an important paradigm in the design of the distributed systems and the design of efficient methods of recording the global state is an important issue.
- This lecture first discussed a formal definition of the **global state of a distributed system and issues** related to its capture; then we have discussed the **Chandy-Lamport Algorithm** to record a snapshot of a distributed system.

Hot Data Analytics for Real-Time Streaming in IoT Platform



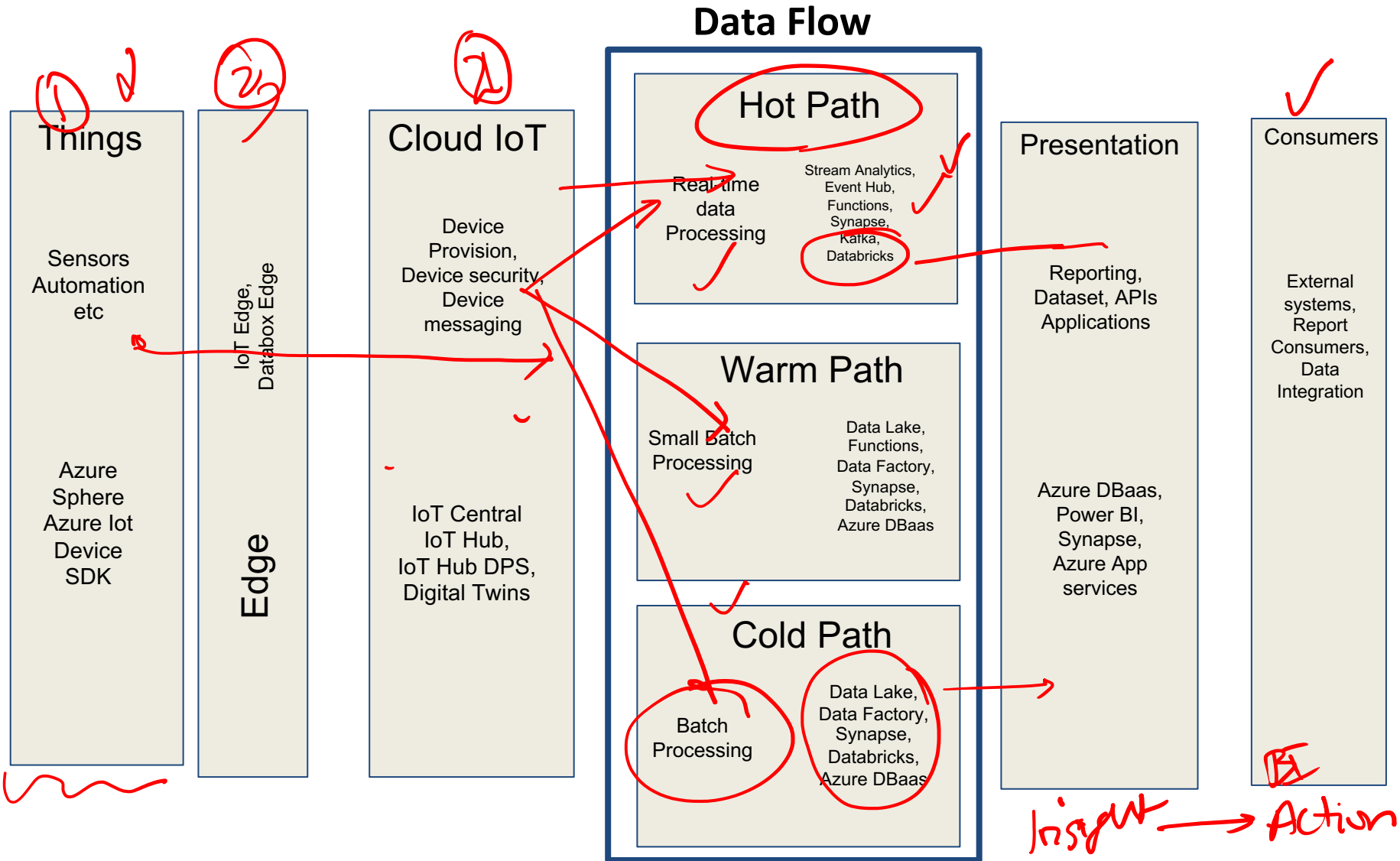
Dr. Rajiv Misra, Professor,
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss Real-time data processing in IoT edge platform with Spark Streaming and Sliding Window Analytics.
- We will also discuss a case study based on Twitter Sentiment Analysis using Streaming.

IoT platform: Overview



Hot Data Analytics

IoT platform: Data Flow

The data is routed to one of the three different paths i.e. the hot path or the cold path or the warm path.

Hot path data is the data that is processed in real time. It gets processed within seconds of that happening, so when the message hits the hot path it's processed and then it's presented to something in the consumption layer. The consumption layer consume that data immediately once it's been processed in the hot path.

The output from a hot path to a cold storage system can be written that is consumed by an api. The data is written in real time but the api might be querying that data that was written an hour ago.

The main thing about hotpath is that you're processing data in real time as it's happening however what's consuming that might be querying old data that was processed an hour ago. It could be something that's processing it and then presenting it in real time such as a dashboard that is constantly monitoring things in their present state as comes off of the hot path and into the consumption layer.

Data Flow

Hot Path

Real-time
data
Processing

Stream Analytics,
Event Hub,
Functions,
Synapse,
Kafka,
Databricks

Warm Path

Small Batch
Processing

Data Lake,
Functions,
Data Factory,
Synapse,
Databricks,
Azure DBaaS

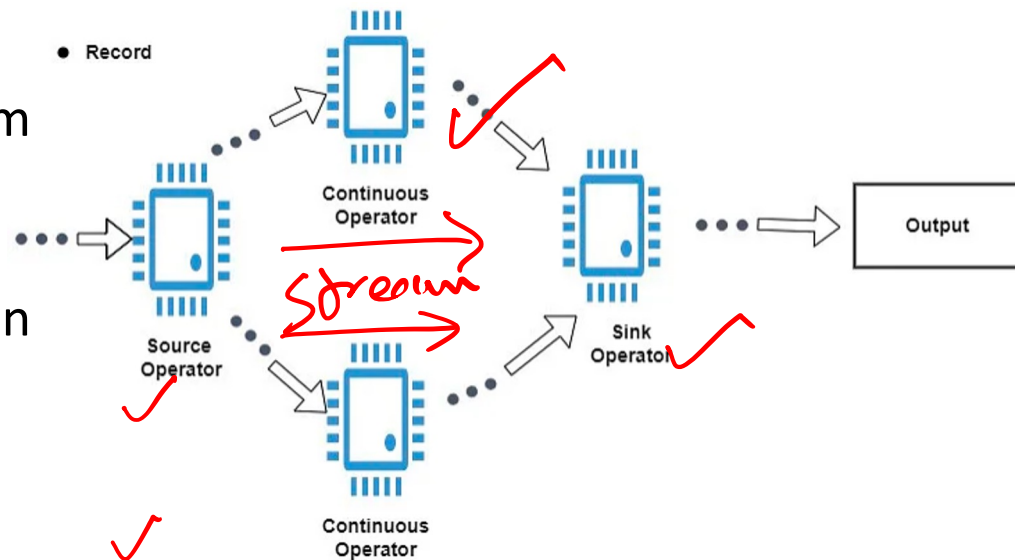
Cold Path

Batch
Processing

Data Lake,
Data Factory,
Synapse,
Databricks,
Azure DBaaS

IoT platform: Traditional Stream Processing

- Streaming data is received from data sources (e.g. live logs, system telemetry data, IoT device data, etc.) into some data ingestion system like Apache Kafka, Amazon Kinesis, etc.
- The data is then processed in parallel on a cluster.
- Results are given to downstream systems like HBase, Cassandra, Kafka, etc.
- There is a set of worker nodes, each of which runs one or more continuous operators. Each continuous operator processes the streaming data one record at a time and forwards the records to other operators in the pipeline.
- Data is received from ingestion systems via Source operators and given as output to downstream systems via sink operators.
- Continuous operators are a simple and natural model. However, this traditional architecture has also met some challenges with today's trend towards larger scale and more complex real-time analytics



Traditional Stream Processing: Limitations

- **Fast Failure and Straggler Recovery** In real time, the system must be able to fastly and automatically recover from failures and stragglers to provide results which is challenging in traditional systems due to the static allocation of continuous operators to worker nodes.
- **Load Balancing** In a continuous operator system, uneven allocation of the processing load between the workers can cause bottlenecks. The system needs to be able to dynamically adapt the resource allocation based on the workload.
- **Unification of Streaming, Batch and Interactive Workloads** In many use cases, it is also attractive to query the streaming data interactively, or to combine it with static datasets (e.g. pre-computed models). This is hard in continuous operator systems which does not designed to new operators for ad-hoc queries. This requires a single engine that can combine batch, streaming and interactive queries.
- **Advanced Analytics with Machine learning and SQL Queries** Complex workloads require continuously learning and updating data models, or even querying the streaming data with SQL queries. Having a common abstraction across these analytic tasks makes the developer's job much easier.

Big Streaming Data Processing

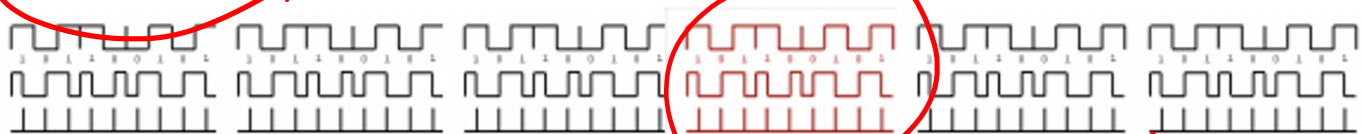
Fraud detection in bank transactions ✓



Stream data

Hot path anomaly

Anomalies in sensor data ✓



Stream data

data streams

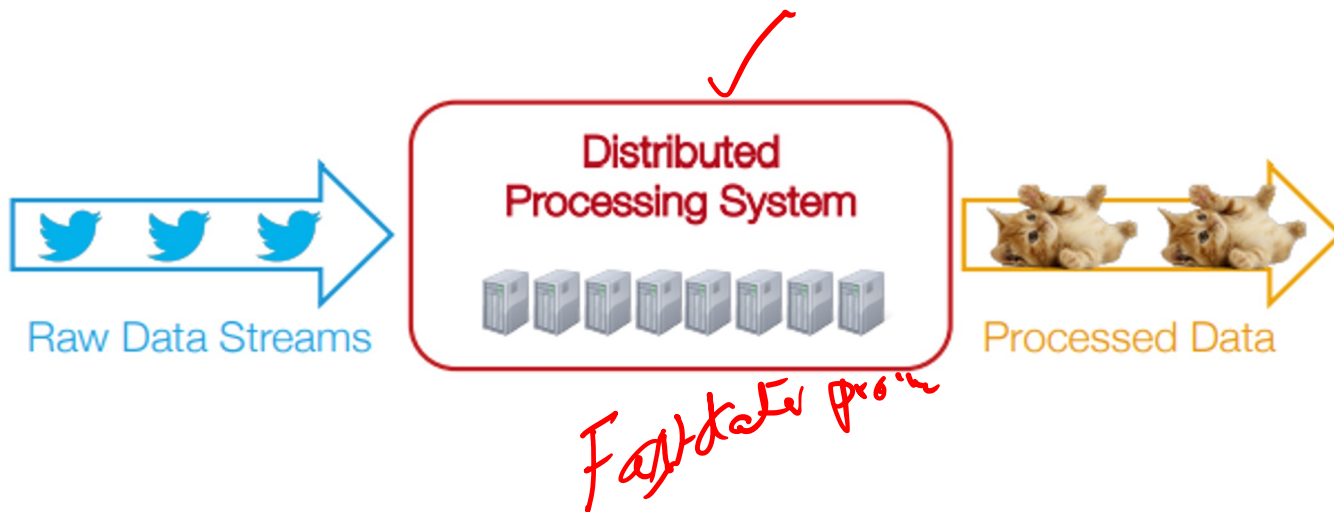
Cat videos in tweets ✓



tweet data stream

How to Process Big Streaming Data

- Scales to hundreds of nodes
 - Achieves low latency
 - Efficiently recover from failures
 - Integrates with batch and interactive processing (stream better with analytics)
- with time window before next batch if store news arrives*



What people have been doing?

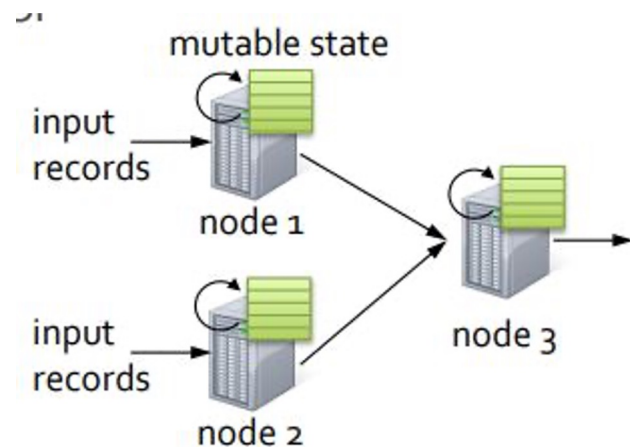
- Build two stacks – one for batch, one for streaming
 - Often both process same data
- Existing frameworks cannot do both
 - Either, stream processing of 100s of MB/s with low latency
 - Or, batch processing of TBs of data with high latency

What people have been doing?

- Extremely painful to maintain two different stacks
 - Different programming models
 - Doubles implementation effort
 - Doubles operational effort

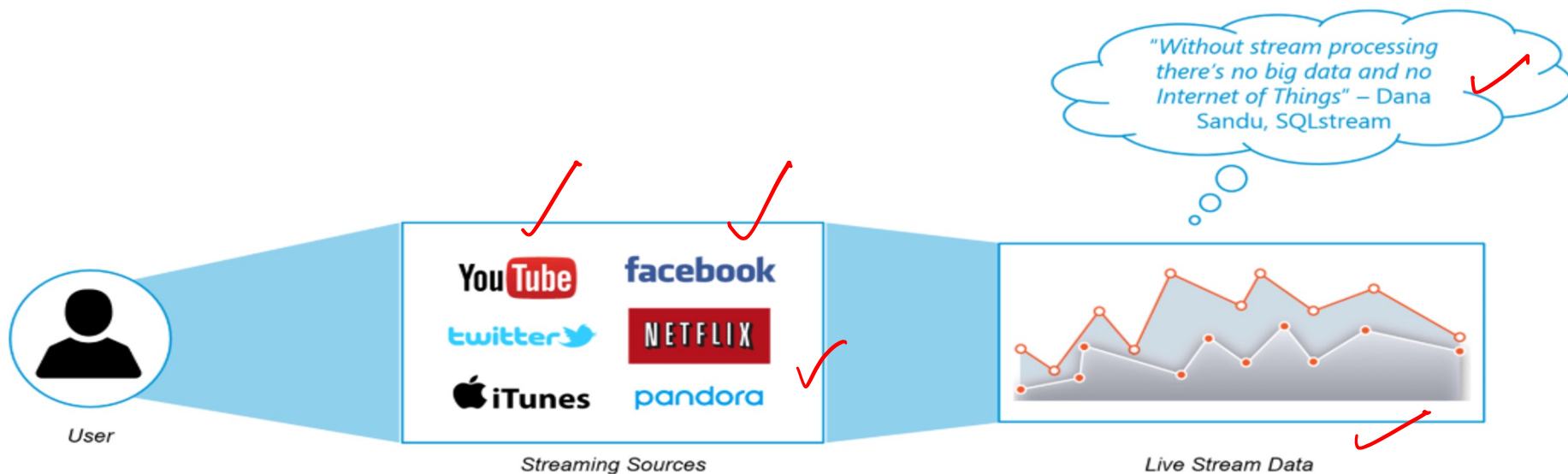
Fault-tolerant Stream Processing

- Traditional processing model
 - Pipeline of nodes
 - Each node maintains mutable state
 - Each input record updates the state and new records are sent out
- Mutable state is lost if node fails
- Making stateful stream processing fault-tolerant is challenging!

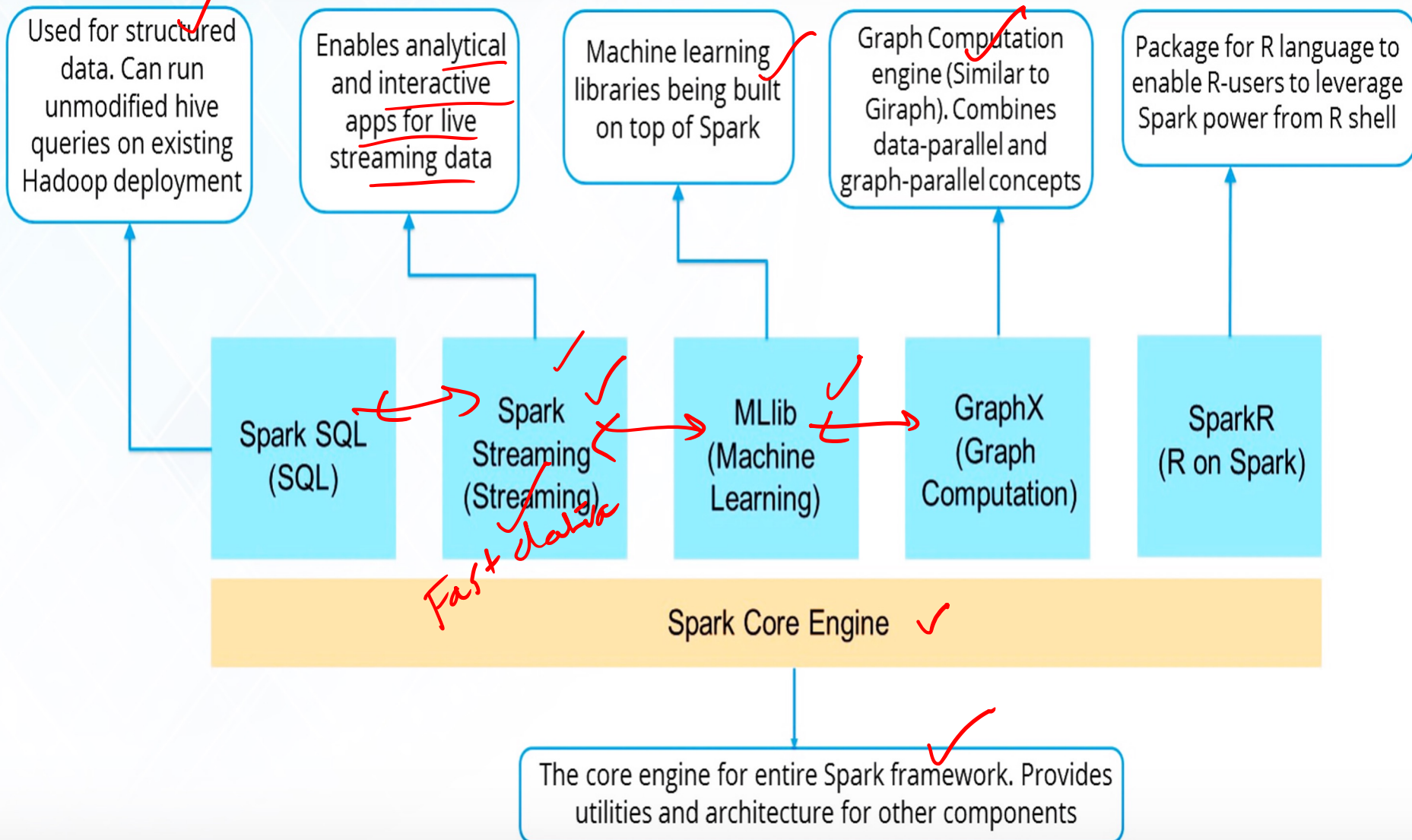


What is Streaming?

- Data Streaming is a technique for transferring data so that it can be processed as a steady and continuous stream.
- Streaming technologies are becoming increasingly important with the growth of the Internet.

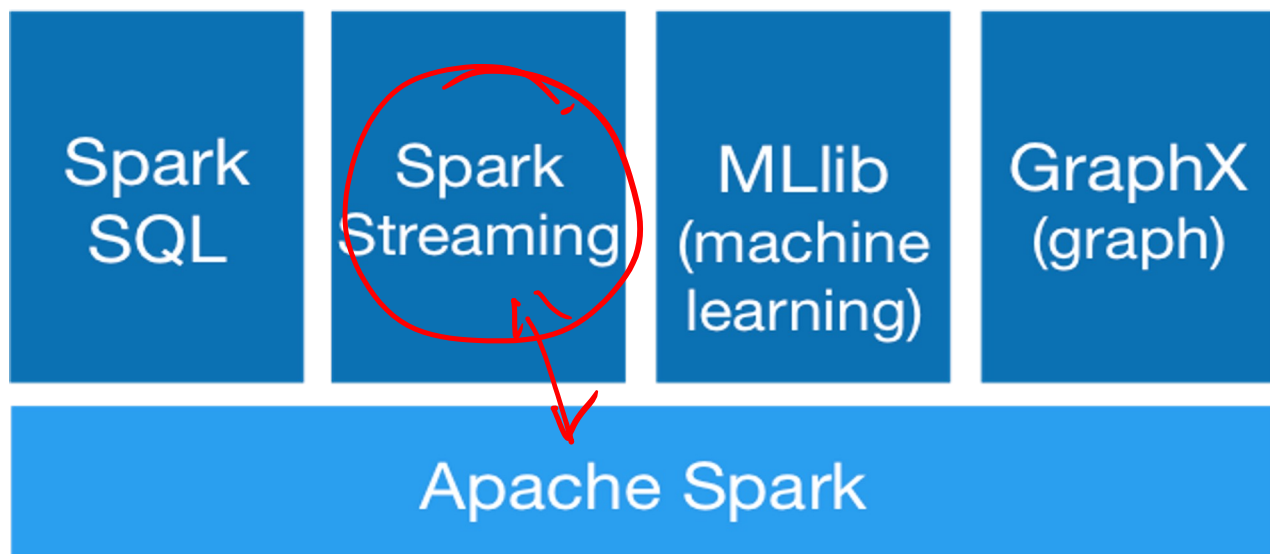


Spark Ecosystem



What is Spark Streaming?

- Extends Spark for doing big data stream processing
- Project started in early 2012, alpha released in Spring 2017 with Spark 0.7
- Moving out of alpha in Spark 0.9
- Spark Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets.
- In Spark 2.x, a separate technology based on Datasets, called Structured Streaming, that has a higher-level interface is also provided to support streaming.



What is Spark Streaming?

- Framework for large scale stream processing
- Scales to 100s of nodes ✓
- Can achieve second scale latencies ✓
- Integrates with Spark's batch and interactive processing ✓
- Provides a simple batch-like API for implementing complex algorithm ✓
- Can absorb live data streams from Kafka, Flume, ZeroMQ, etc. ✓

What is Spark Streaming?

- Receive data streams from input sources, process them in a cluster, push out to databases/ dashboards
- Scalable, fault-tolerant, second-scale latencies



Why Spark Streaming ?

- Many big-data applications need to process ~~large~~ data streams in realtime (10T) ✓

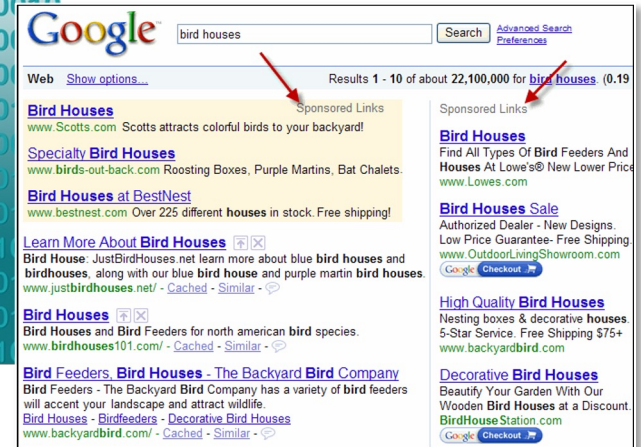
Website monitoring



Fraud detection



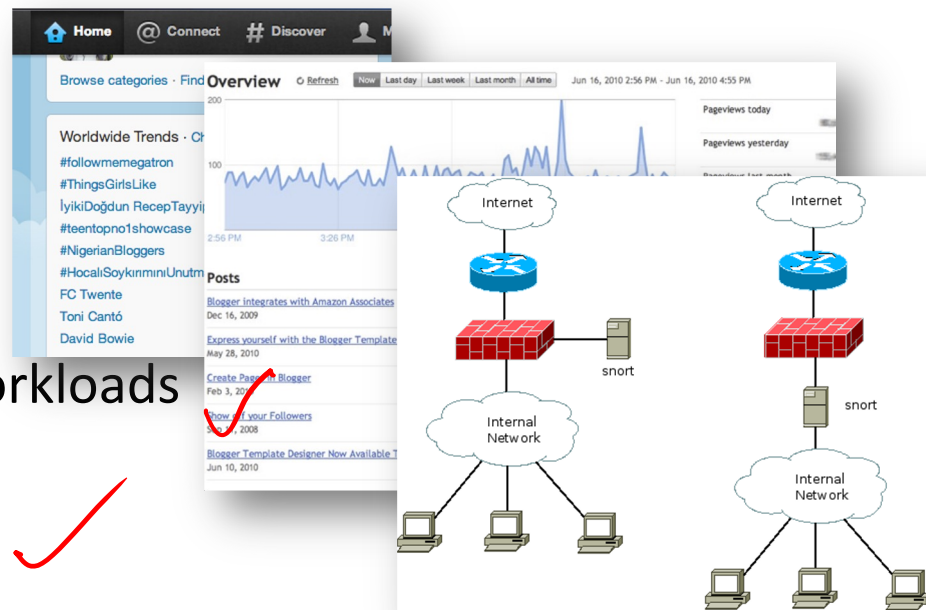
Ad monetization



Hot Data Analytics

Why Spark Streaming ?

- Many important applications must process large streams of live data and provide results in near-real-time
 - Social network trends
 - Website statistics
 - Intrusion detection systems
 - etc.
- Require large clusters to handle workloads
- Require latencies of few seconds



Why Spark Streaming ?

- We can use Spark Streaming to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.



Spark Streaming is used to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.

Why Spark Streaming?

Need a framework for big data
stream processing that

Website monitoring



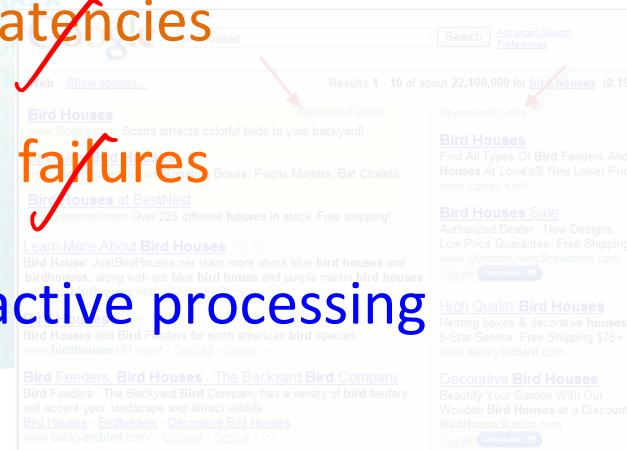
Scales to hundreds of nodes

Achieves second-scale latencies

Efficiently recover from failures

Integrates with batch and interactive processing

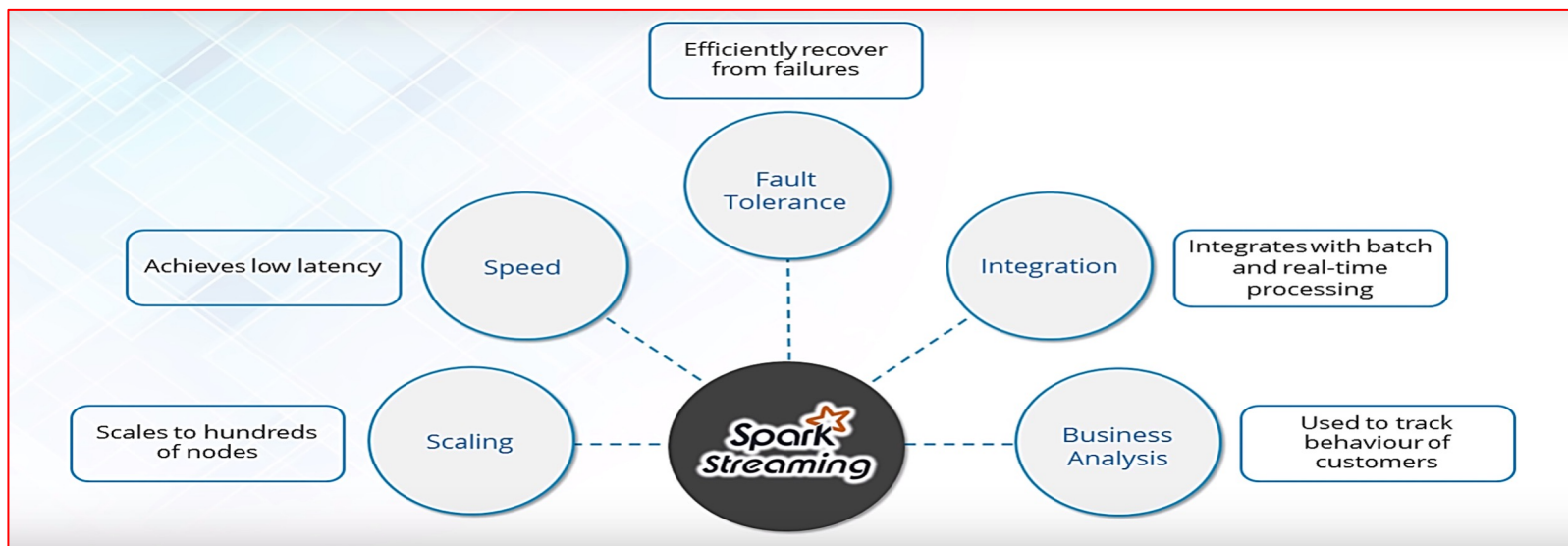
Ad monetization



Hot Data Analytics

Spark Streaming Features

- **Scaling:** Spark Streaming can easily scale to hundreds of nodes.
- **Speed:** It achieves low latency.
- **Fault Tolerance:** Spark has the ability to efficiently recover from failures.
- **Integration:** Spark integrates with batch and real-time processing.
- **Business Analysis:** Spark Streaming is used to track the behavior of customers which can be used in business analysis




Requirements

- **Scalable** to large clusters
- **Second-scale** latencies
- **Simple** programming model
- **Integrated** with batch & interactive processing
- **Efficient fault-tolerance** in stateful computations

Batch vs Stream Processing

Batch Processing

- Ability to process and analyze data at-rest (stored data) 
- Request-based, bulk evaluation and short-lived processing
- Enabler for **Retrospective, Reactive and On-demand Analytics**

Stream Processing

- Ability to ingest, process and analyze data in-motion in real- or near-real-time
- Event or micro-batch driven, continuous evaluation and long-lived processing
- Enabler for **real-time Prospective, Proactive and Predictive Analytics** for Next Best Action

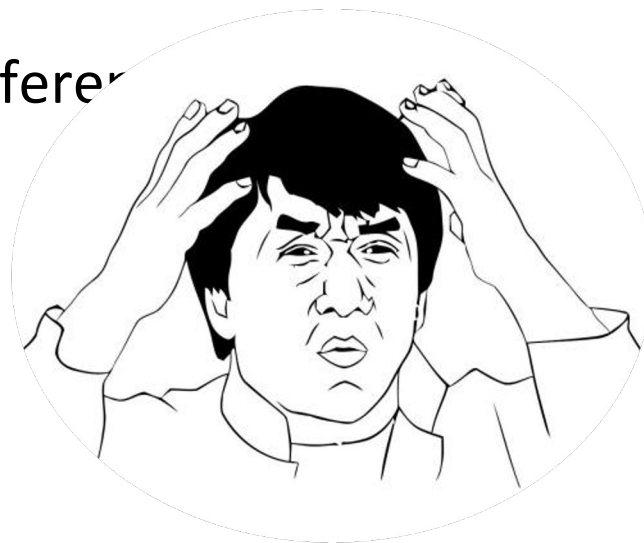
Stream Processing + Batch Processing = All Data Analytics

real-time (now)

historical (past)

Integration with Batch Processing

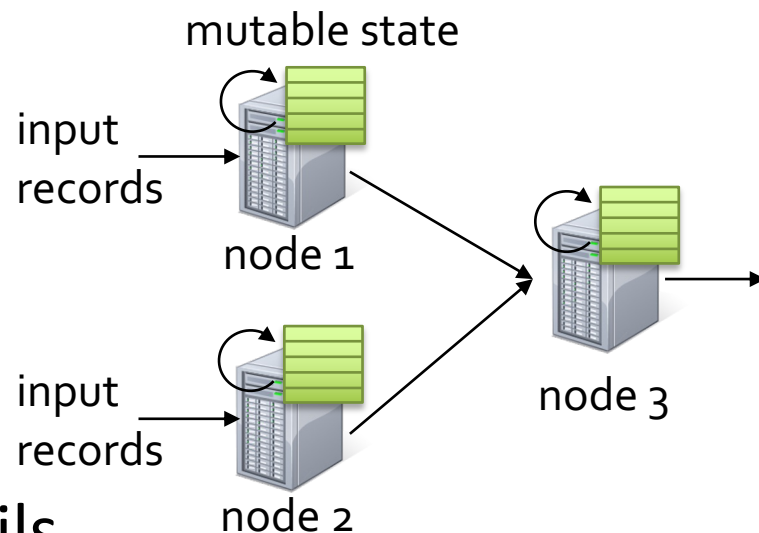
- Many environments require processing same data in live streaming as well as batch post-processing
- Existing frameworks cannot do both
 - Either, stream processing of 100s of MB/s with low latency
 - Or, batch processing of TBs of data with high latency
- Extremely painful to maintain two different
 - Different programming models
 - Double implementation effort



Stateful Stream Processing

- Traditional model

- Processing pipeline of nodes
- Each node maintains mutable state
- Each input record updates the state and new records are sent out

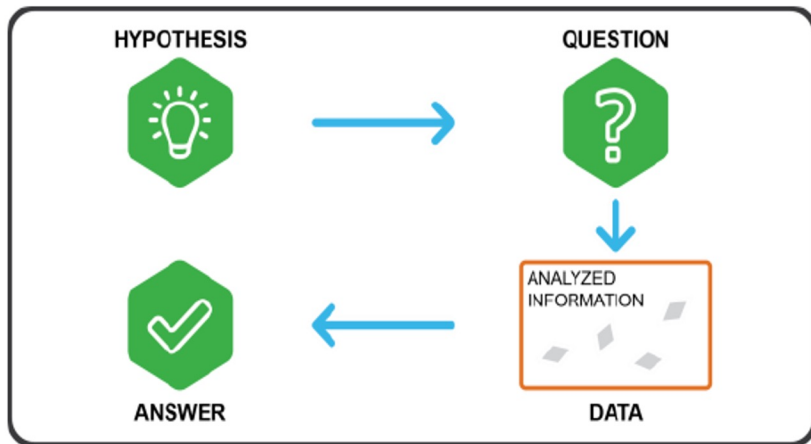


- Mutable state is lost if node fails
- Making stateful stream processing fault tolerant is challenging!

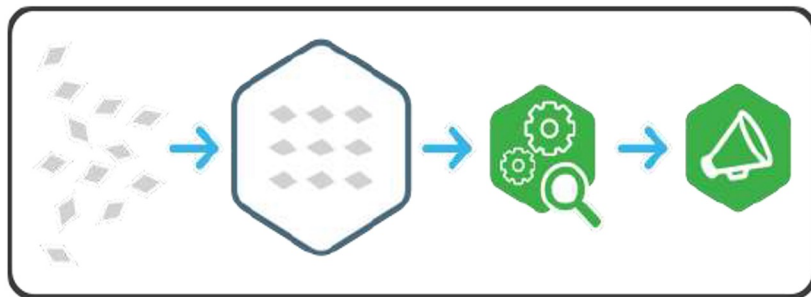
Modern Data Applications approach to Insights

Traditional Analytics

Structured & Repeatable
Structure built to store data



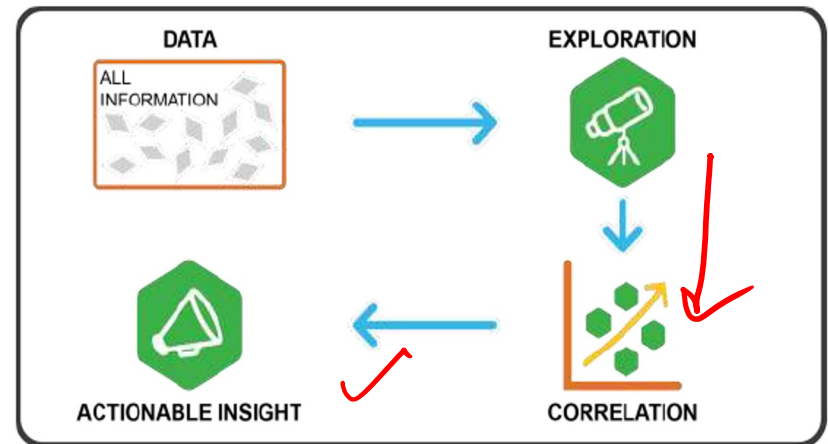
Start with hypothesis
Test against selected data



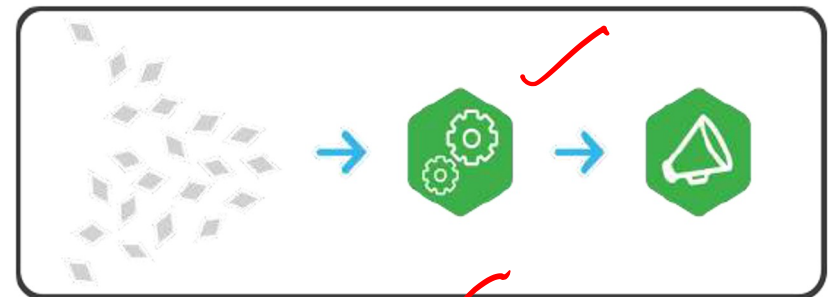
Analyze after landing...

Next Generation Analytics

Iterative & Exploratory
Data is the structure



Data leads the way
Explore all data, identify correlations



Analyze in motion...

Hot Data Analytics

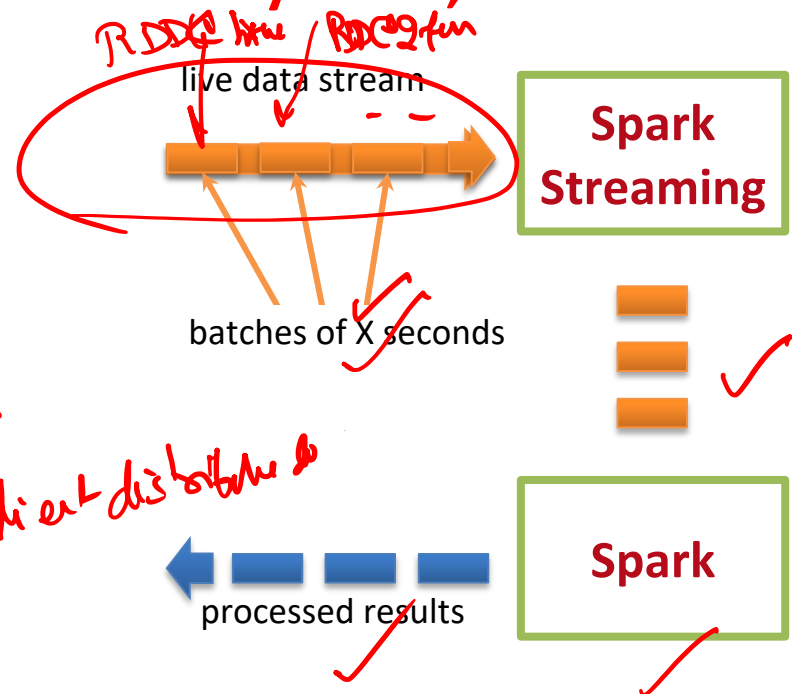
Existing Streaming Systems

- Storm ✓
 - Replays record if not processed by a node
 - Processes each record at least once
 - May update mutable state twice!
 - Mutable state can be lost due to failure!
- Trident – ✓
 - Use transactions to update state
 - Processes each record exactly once
 - Per-state transaction to external database is slow

How does Spark Streaming work?

Run a streaming computation as a **series of very small, deterministic batch jobs**

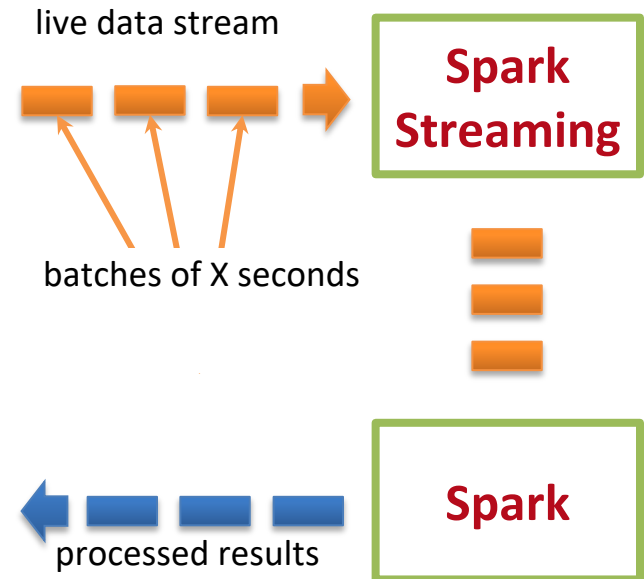
- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as **RDDs** and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



How does Spark Streaming work?

Run a streaming computation as a **series of very small, deterministic batch jobs**

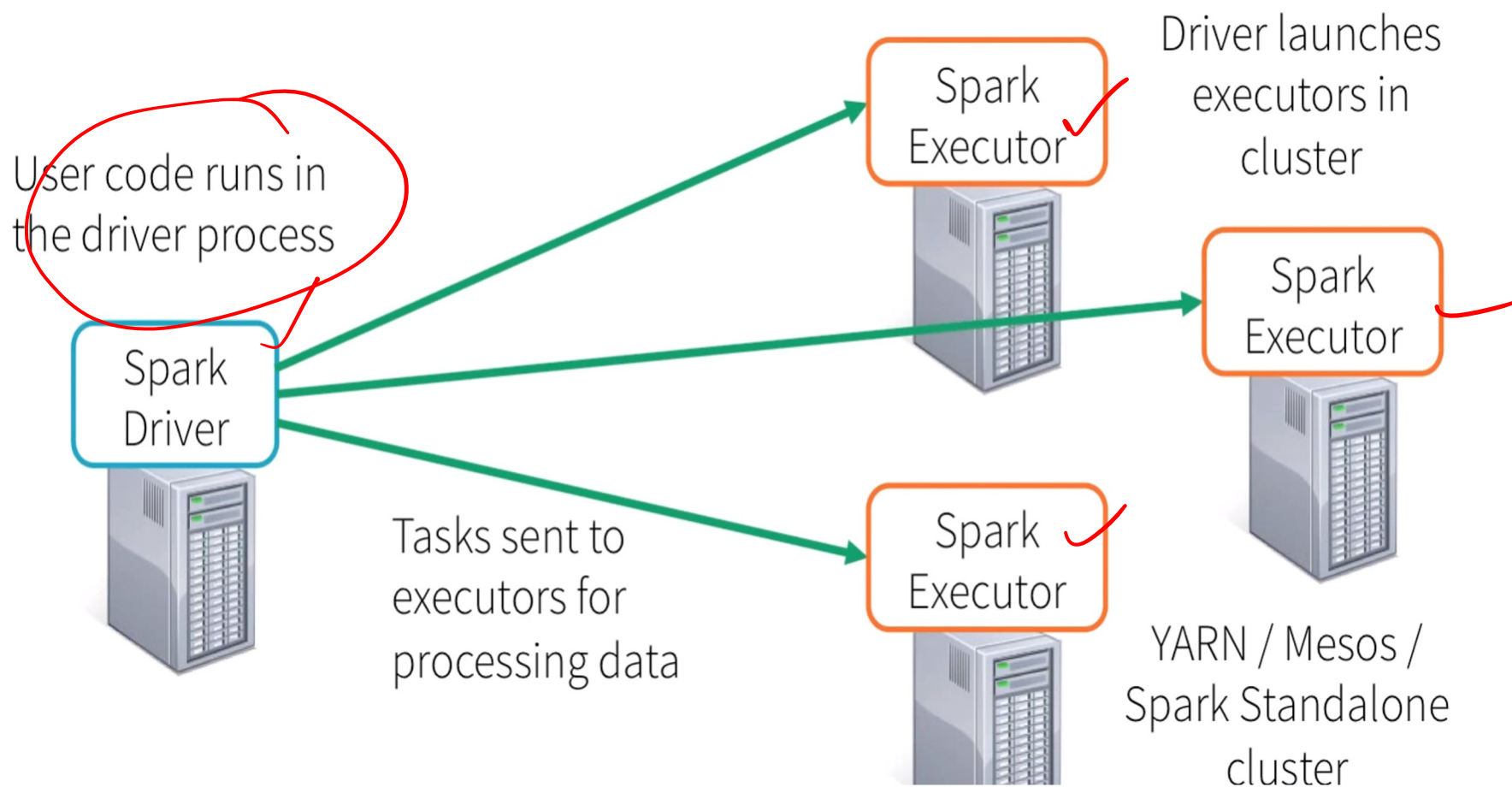
- Batch sizes as low as ½ second, latency of about 1 second
- Potential for combining batch processing and streaming processing in the same system



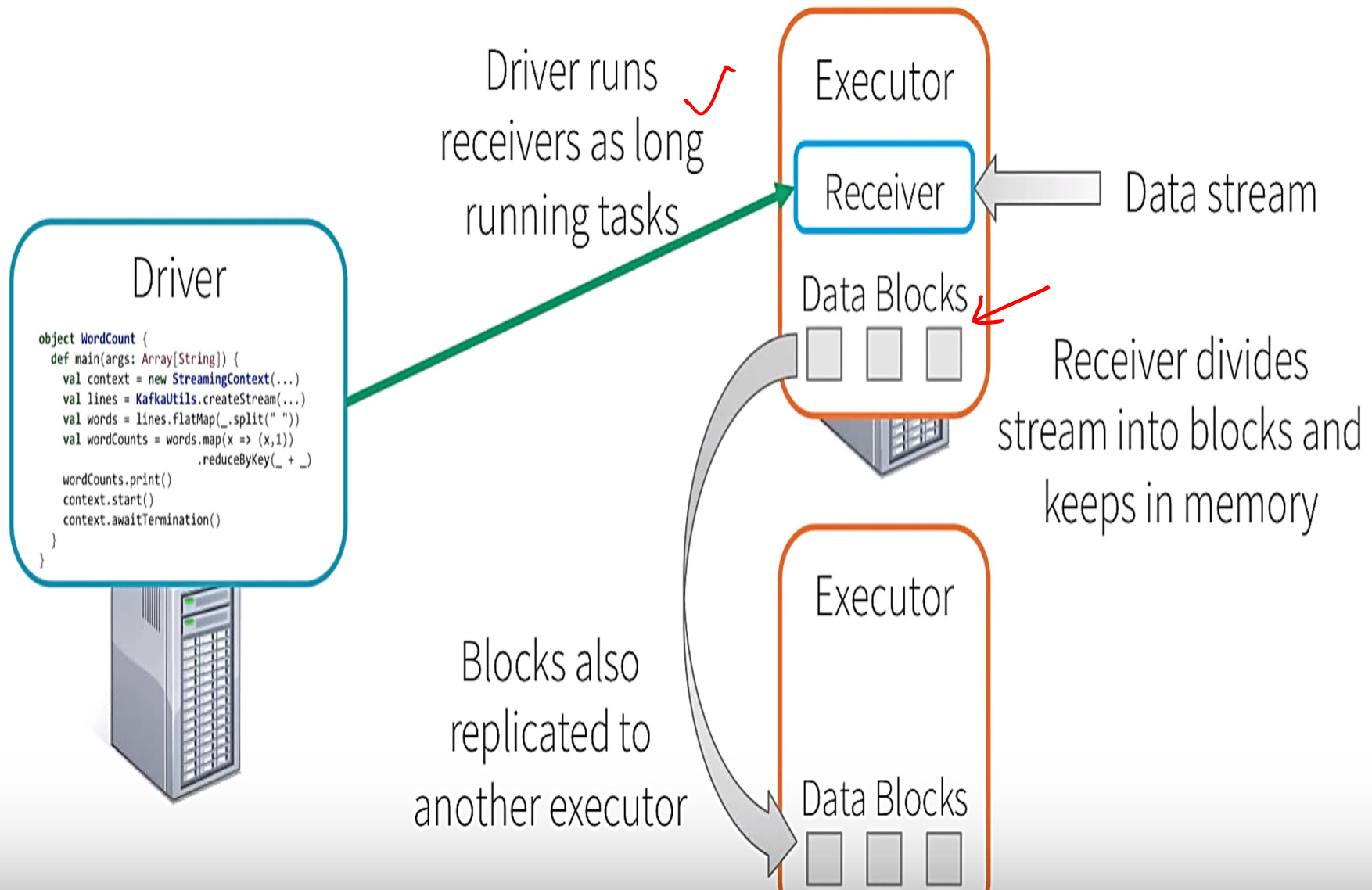
Word Count with Kafka

```
object WordCount {  
  def main(args: Array[String]) {  
    val context = new StreamingContext(new SparkConf(), Seconds(1))  
    val lines = KafkaUtils.createStream(context, ...)  
    val words = lines.flatMap(_.split(" "))  
    val wordCounts = words.map(x => (x,1)).reduceByKey(_ + _)  
    wordCounts.print()  
    context.start()  
    context.awaitTermination()  
  }  
}
```

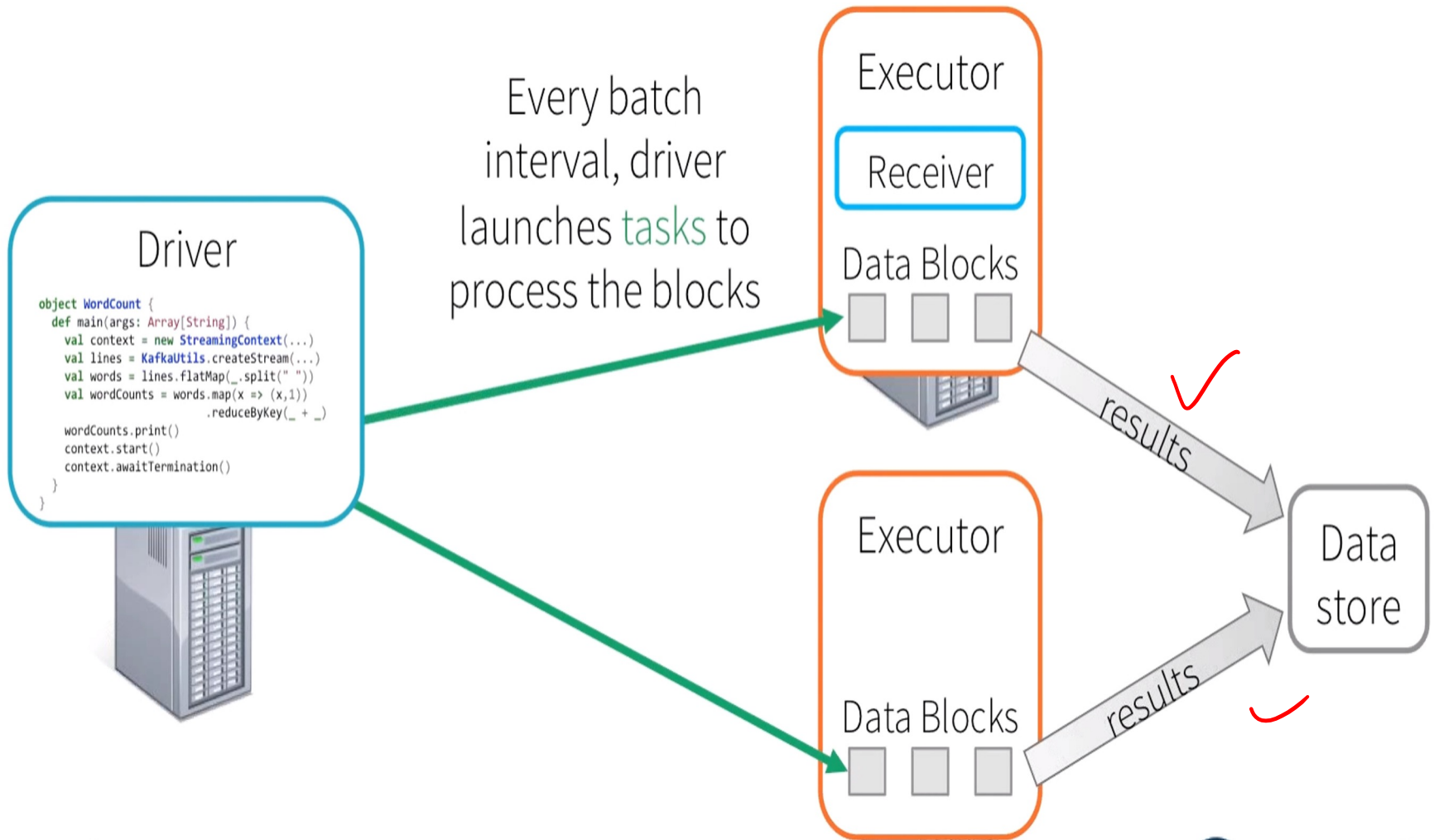

Any Spark Application



Spark Streaming Application: Receive data

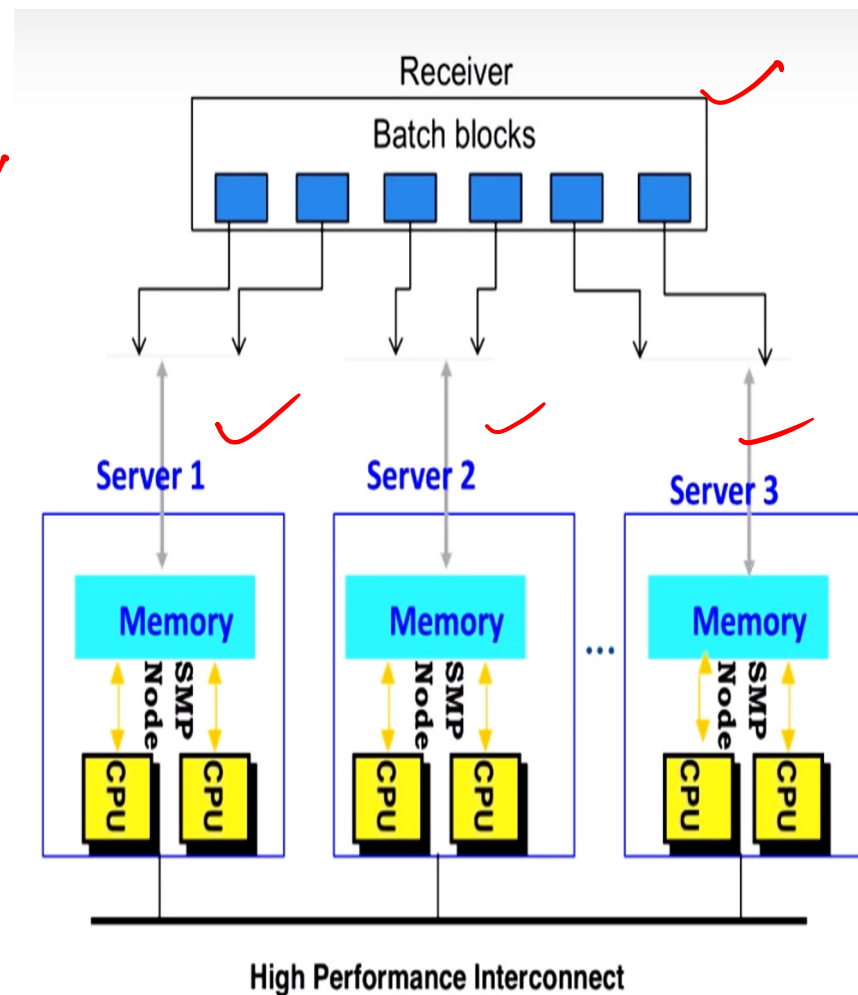


Spark Streaming Application: Process data



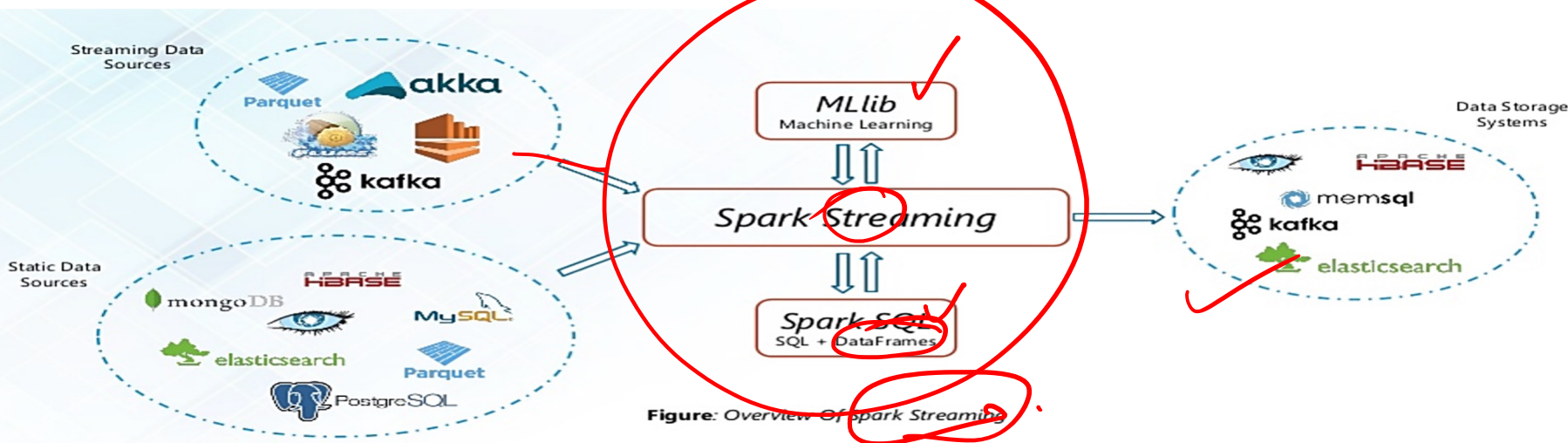
Spark Streaming Architecture

- Micro batch architecture. ✓
- Operates on interval of time ✓
- New batches are created at regular time intervals. ✓
- Divides received time batch into blocks for parallelism ✓
- Each batch is a graph that translates into multiple jobs ✓
- Has the ability to create larger size batch window as it processes over time.



Spark Streaming Workflow

- Spark Streaming workflow has four high-level stages. The first is to stream data from various sources. These sources can be streaming data sources like Akka, Kafka, Flume, AWS or Parquet for real-time streaming. The second type of sources includes HBase, MySQL, PostgreSQL, Elastic Search, Mongo DB and Cassandra for static/batch streaming.
- Once this happens, Spark can be used to perform Machine Learning on the data through its MLlib API. Further, Spark SQL is used to perform further operations on this data. Finally, the streaming output can be stored into various data storage systems like HBase, Cassandra, MemSQL, Kafka, Elastic Search, HDFS and local file system.



Spark Streaming Workflow



Figure: Data from a variety of sources to various storage systems

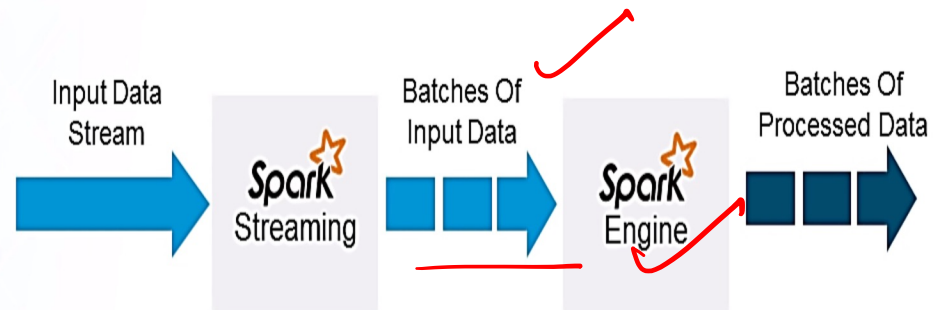


Figure: Incoming streams of data divided into batches



Figure: Input data stream divided into discrete chunks of data

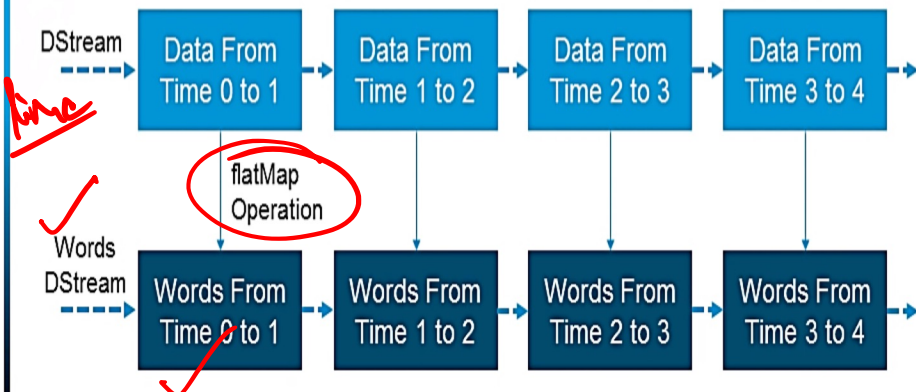


Figure: Extracting words from an InputStream

Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()
```

DStream: a sequence of RDDs representing a stream of data

Twitter Streaming API

batch @ t

batch @ t+1

batch @ t+2



tweets DStream



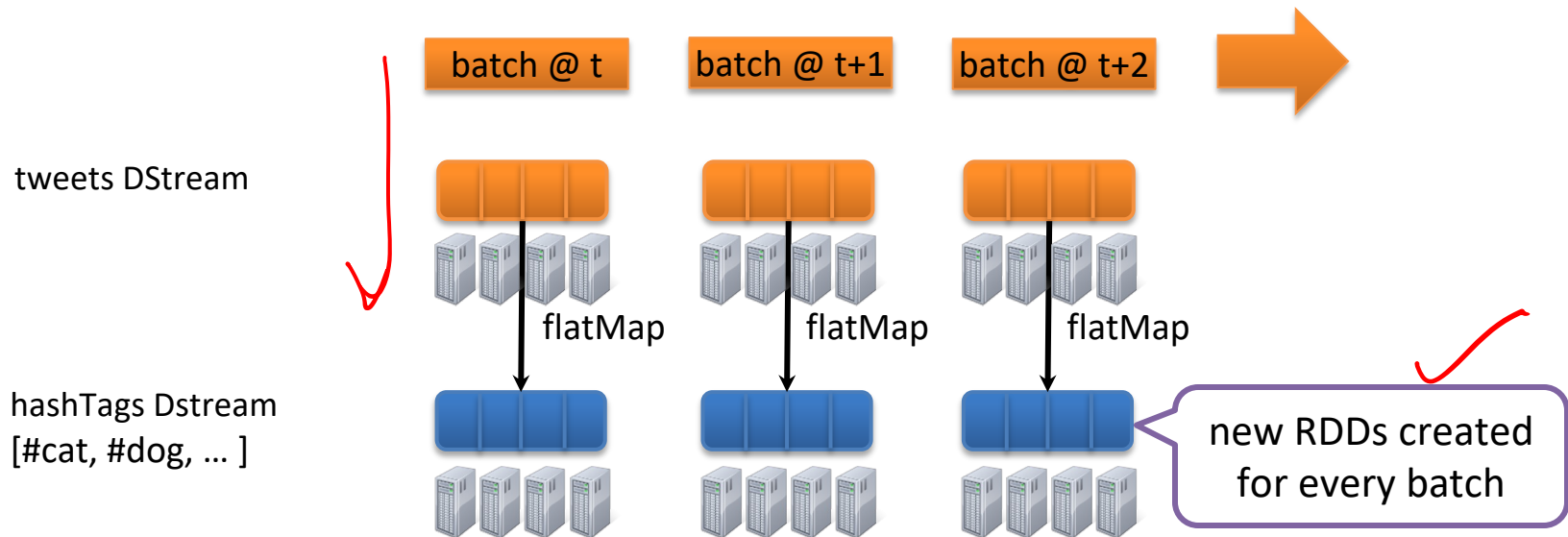
stored in memory as an RDD
(immutable, distributed)

Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()  
val hashTags = tweets.flatMap(status => getTags(status))
```

new DStream

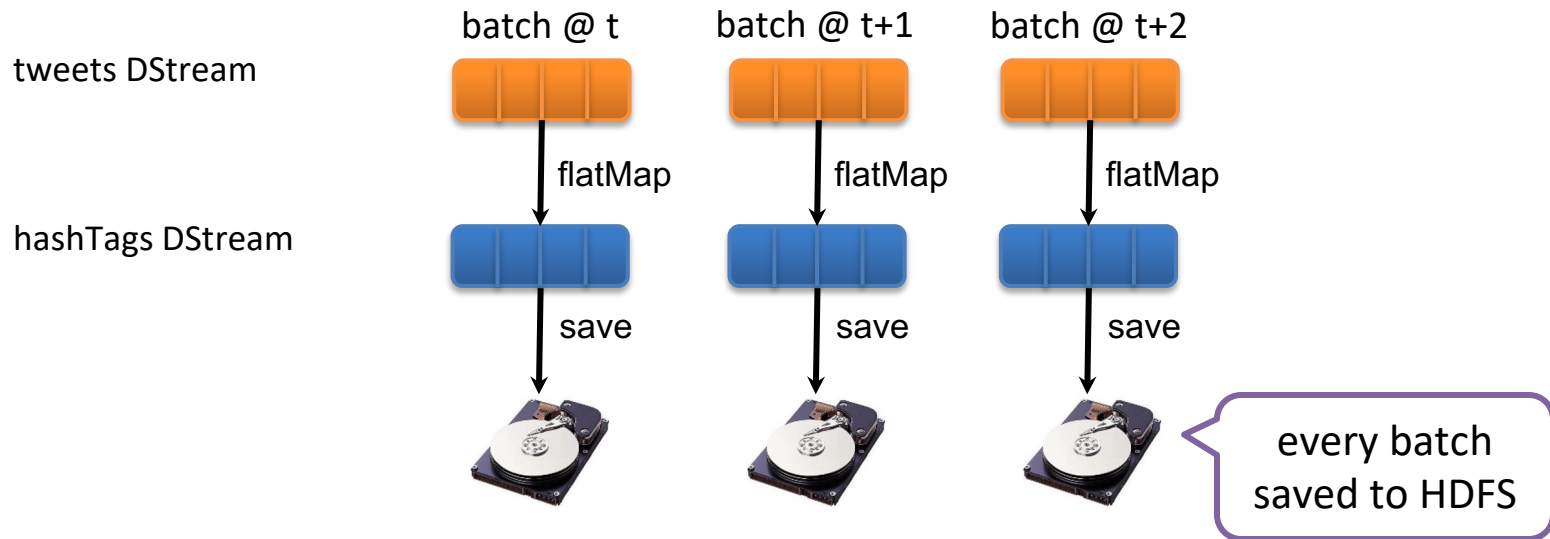
transformation: modify data in one DStream to create another DStream



Example 1– Get hashtags from Twitter

```
val tweets = ssc.twitterStream()  
val hashTags = tweets.flatMap(status => getTags(status))  
hashTags.saveAsHadoopFiles("hdfs://...")
```

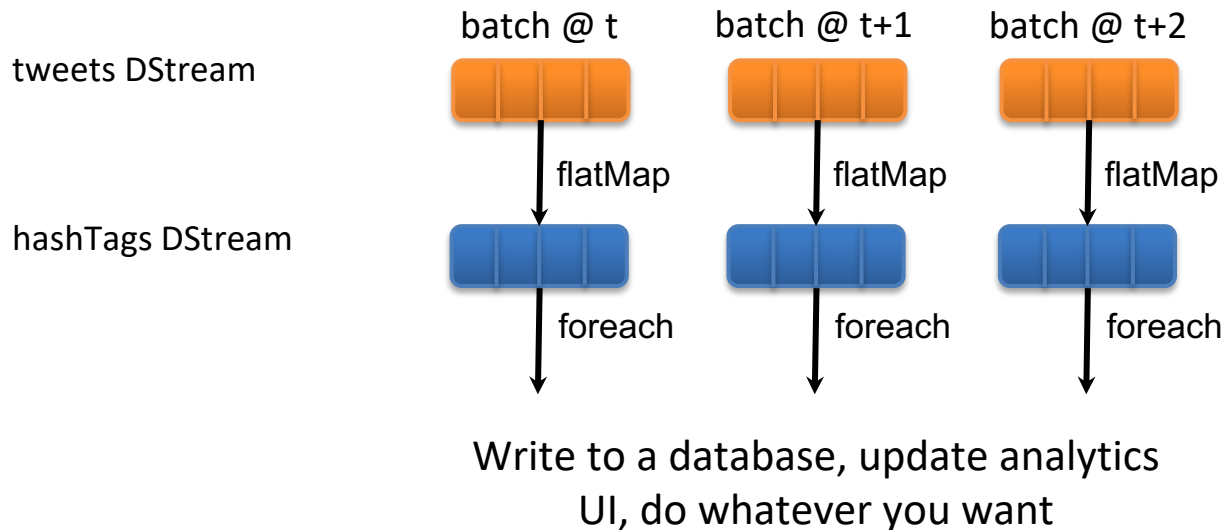
output operation: to push data to external storage



Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()  
val hashTags = tweets.flatMap(status => getTags(status))  
hashTags.foreach(hashTagRDD => { ... })
```

foreach: do whatever you want with the processed data



Java Example

Scala

```
val tweets = ssc.twitterStream()  
val hashTags = tweets.flatMap(status => getTags(status))  
hashTags.saveAsHadoopFiles("hdfs://...")
```

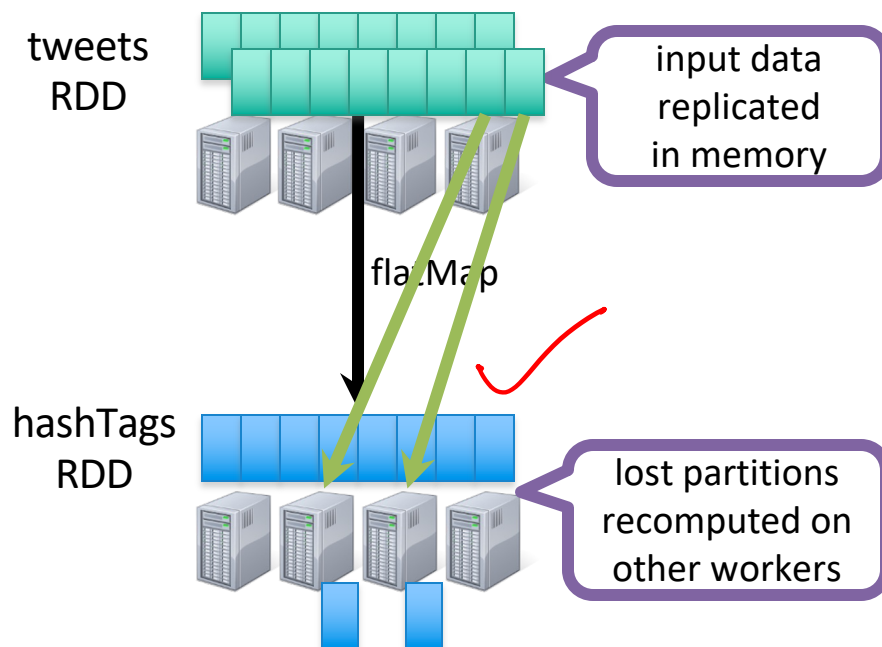
Java

```
JavaDStream<Status> tweets = ssc.twitterStream()  
JavaDStream<String> hashTags = tweets.flatMap(new Function<...> { })  
hashTags.saveAsHadoopFiles("hdfs://...")
```

Function object

Fault-tolerance

- RDDs remember the sequence of operations that created it from the original fault-tolerant input data
- Batches of input data are replicated in memory of multiple worker nodes, therefore fault-tolerant
- Data lost due to worker failure, can be recomputed from input data

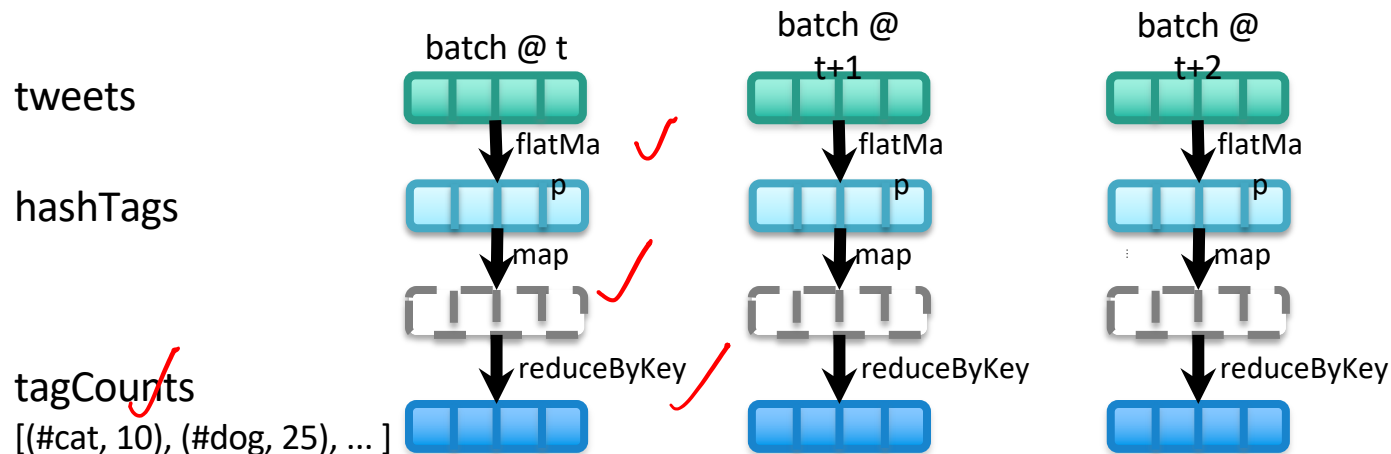


Key concepts

- **DStream** – sequence of RDDs representing a stream of data
 - Twitter, HDFS, Kafka, Flume, ZeroMQ, Akka Actor, TCP sockets
- **Transformations** – modify data from on DStream to another
 - Standard RDD operations – map, countByValue, reduce, join, ...
 - Stateful operations – window, countByValueAndWindow, ...
- **Output Operations – send data to external entity**
 - saveAsHadoopFiles – saves to HDFS
 - foreach – do anything with each batch of results

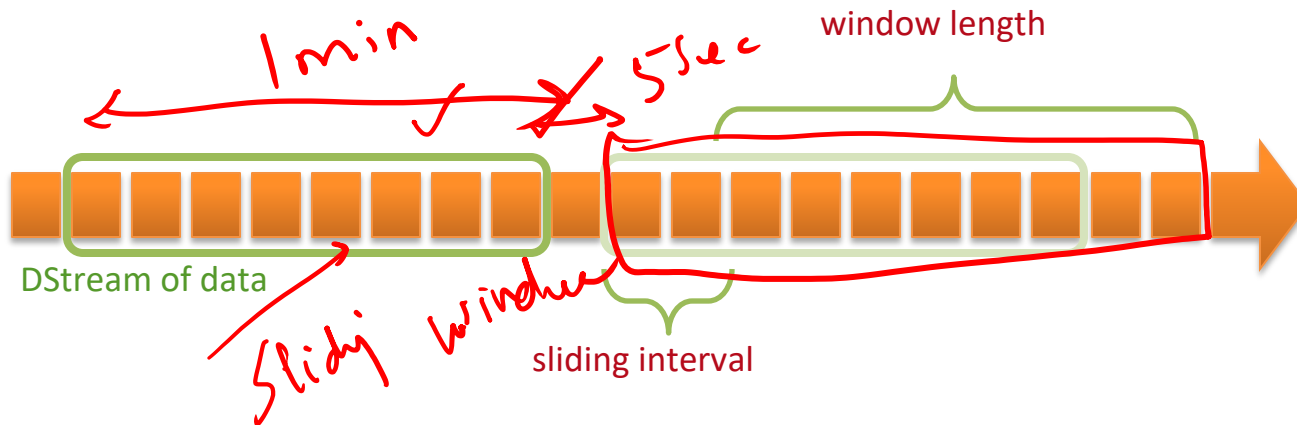
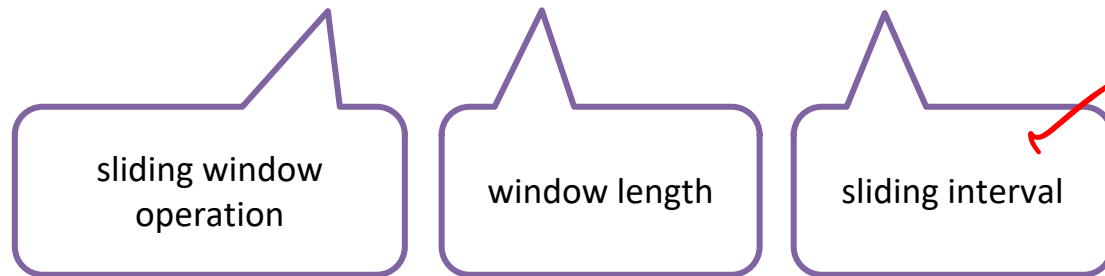
Example 2 – Count the hashtags

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)  
val hashTags = tweets.flatMap(status => getTags(status))  
val tagCounts = hashTags.countByValue()
```



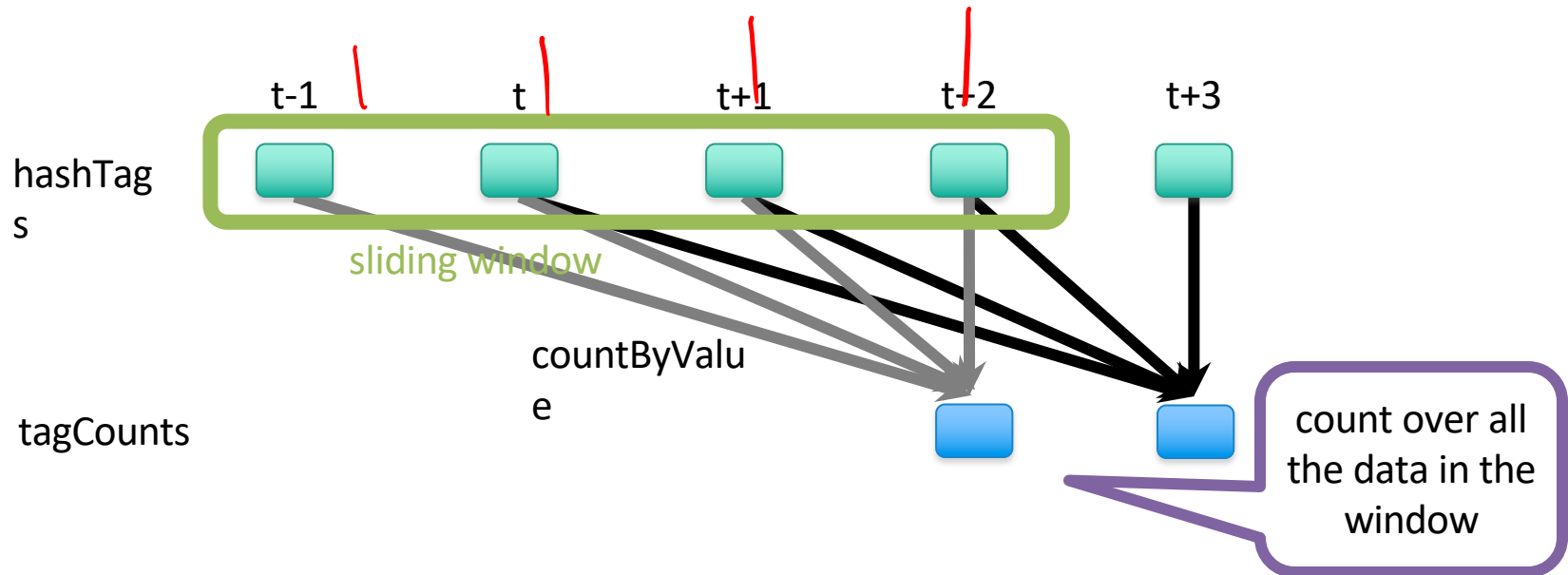
Example 3 – Count the hashtags over last 10 mins

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1),  
Seconds(5)).countByValue()
```



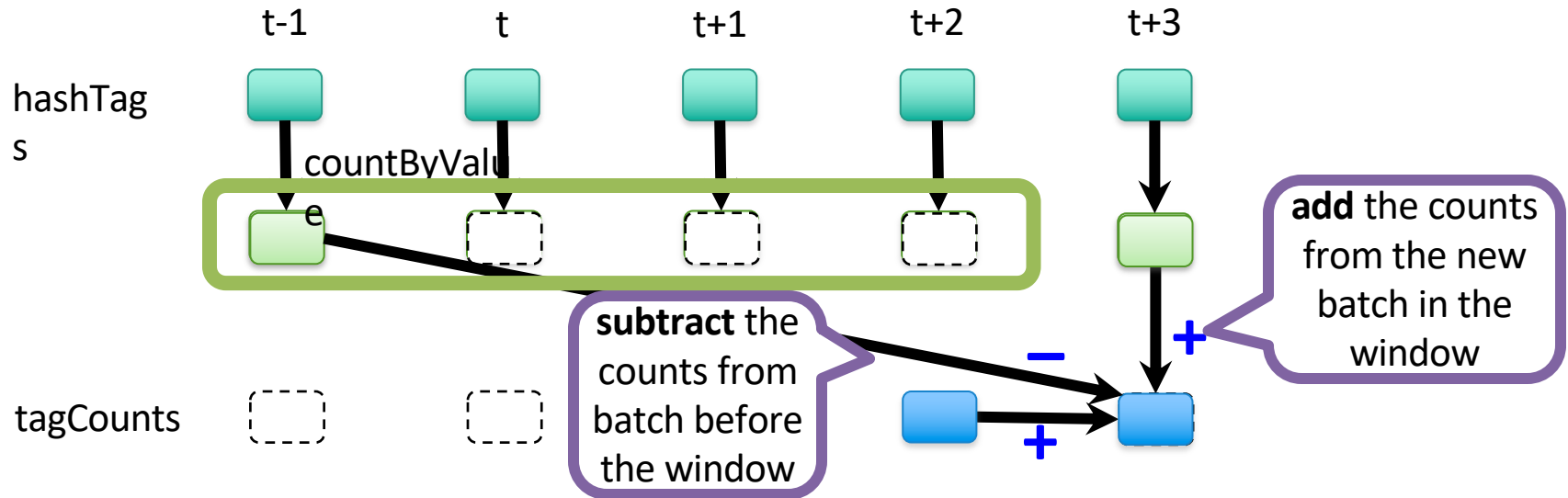
Example 3 – Counting the hashtags over last 10 mins

```
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```



Smart window-based countByValue

```
val tagCounts = hashtags.countByValueAndWindow(Minutes(10), Seconds(1))
```



Smart window-based *reduce*

- Technique to incrementally compute count generalizes to many reduce operations
 - Need a function to “inverse reduce” (“subtract” for counting)
- Could have implemented counting as:
`hashTags.reduceByKeyAndWindow(_ + _, _ - _,`
`Minutes(1), ...)`

Arbitrary Stateful Computations

Specify function to generate new state based on previous state and new data

- Example: Maintain per-user mood as state, and update it with their tweets

```
def updateMood(newTweets, lastMood) => newMood
```

```
moods = tweetsByUser.updateStateByKey(updateMood _)
```




Arbitrary Combinations of Batch and Streaming Computations

Inter-mix RDD and DStream operations!

- Example: Join incoming tweets with a spam HDFS file to filter out bad tweets

```
tweets.transform(tweetsRDD => {  
    tweetsRDD.join(spamHDFSFile).filter(...)  
})
```



Spark Streaming-Dstreams, Batches and RDDs



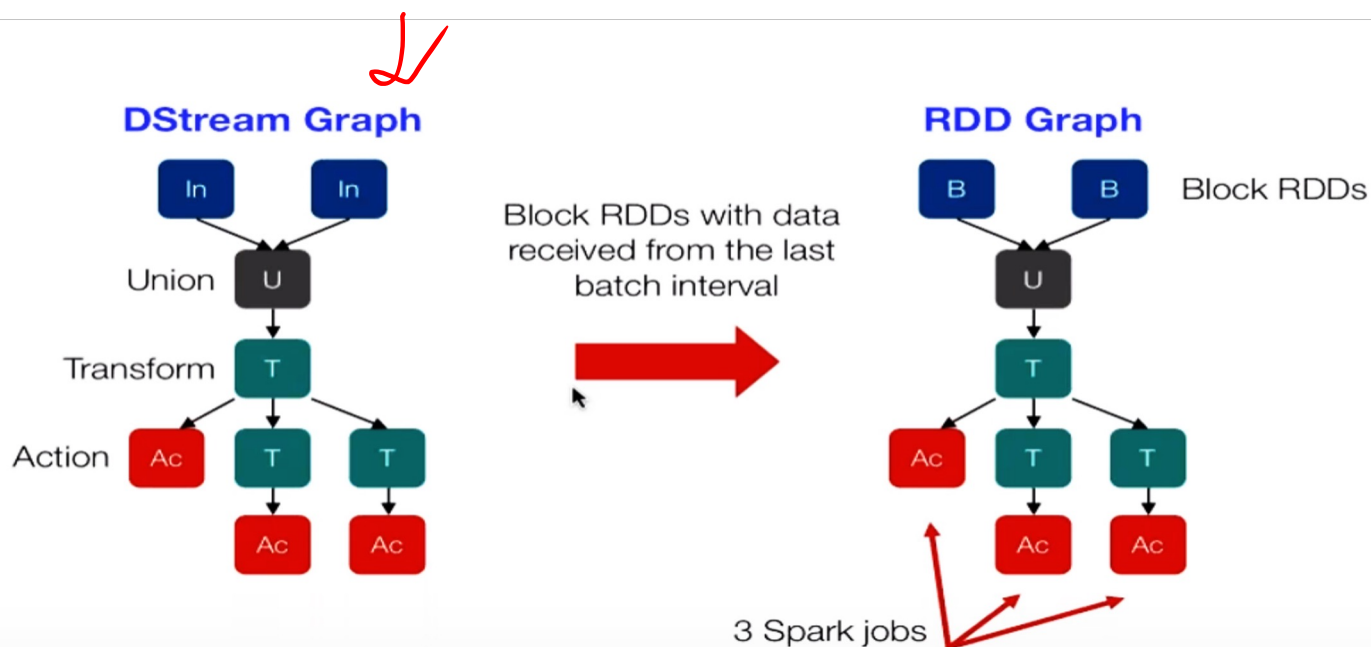
- These steps repeat for each batch.. Continuously ✓
- Because we are dealing with Streaming data. Spark Streaming has the ability to “remember” the previous RDDs...to some extent.

DStreams + RDDs = Power

- Online machine learning
 - Continuously learn and update data models (*updateStateByKey* and *transform*)
- Combine live data streams with historical data
 - Generate historical data models with Spark, etc.
 - Use data models to process live data stream (*transform*)
- CEP-style processing
 - window-based operations (*reduceByWindow*, etc.)

From DStreams to Spark Jobs

- Every interval, an RDD graph is computed from the DStream graph
- For each output operation, a Spark action is created
- For each action, a Spark job is created to compute it



Input Sources

- Out of the box, we provide
 - Kafka, HDFS, Flume, Akka Actors, Raw TCP sockets, etc.
- Very easy to write a *receiver* for your own data source
- Also, generate your own RDDs from Spark, etc. and push them in as a “stream”

Current Spark Streaming I/O

- Input Sources ✓

- Kafka, Flume, Twitter, ZeroMQ, MQTT, TCP sockets
- Basic sources: sockets, files, Akka actors
- Other sources require receiver threads ✓














- Output operations

- Print(), saveAsTextFiles(), saveAsObjectFiles(), saveAsHadoopFiles(), foreachRDD()
- foreachRDD can be used for message queues, DB operations and more ✓



Dstream Classes

- Different classes for different languages (Scala, Java) ✓
- Dstream has 36 value members ✓
- Multiple types of Dstreams ✓
- Separate Python API

org.apache.spark.input		hide	focus
	PortableDataStream		
org.apache.spark.serializer		hide	focus
	DeserializationStream		
org.apache.spark.streaming.api.java		hide	focus
	 JavaDStream		
	 JavaDStreamLike		
	 JavaInputDStream		
	 JavaPairDStream		
	 JavaPairInputDStream		
	 JavaPairReceiverInputDStream		
	 JavaReceiverInputDStream		
org.apache.spark.streaming.dstream		hide	focus
	 ConstantInputDStream		
	 DStream		
	 InputDStream		
	 PairDStreamFunctions		
	 ReceiverInputDStream		

Spark Streaming Operations

- All the Spark **RDD** operations
 - Some available through the transform() operation ✓

map/flatmap ✓	filter	repartition	union
count	reduce	countByValue	reduceByKey
join	cogroup	transform	updateStateByKey

- Spark Streaming **window** operations ✓

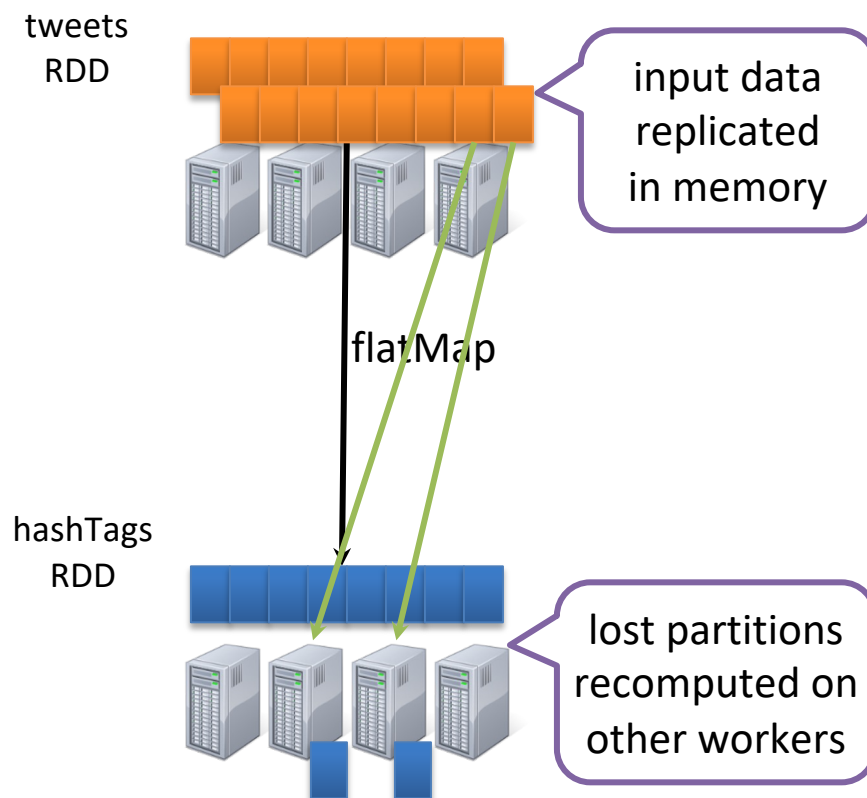
window	countByWindow ✓	reduceByWindow
reduceByKeyAndWindow	countByValueAndWindow	

- Spark Streaming **output** operations

print	saveAsTextFiles	saveAsObjectFiles
saveAsHadoopFiles	foreachRDD	

Fault-tolerance

- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data



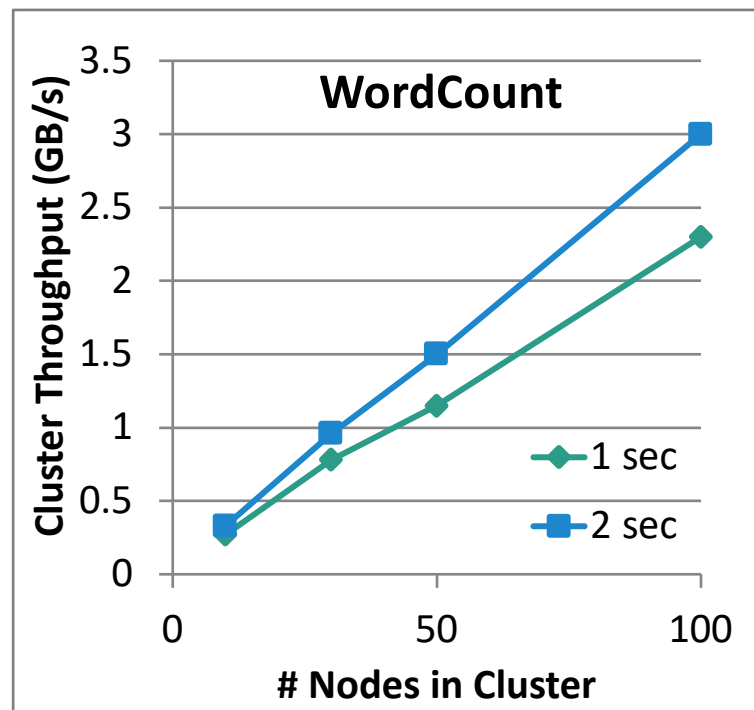
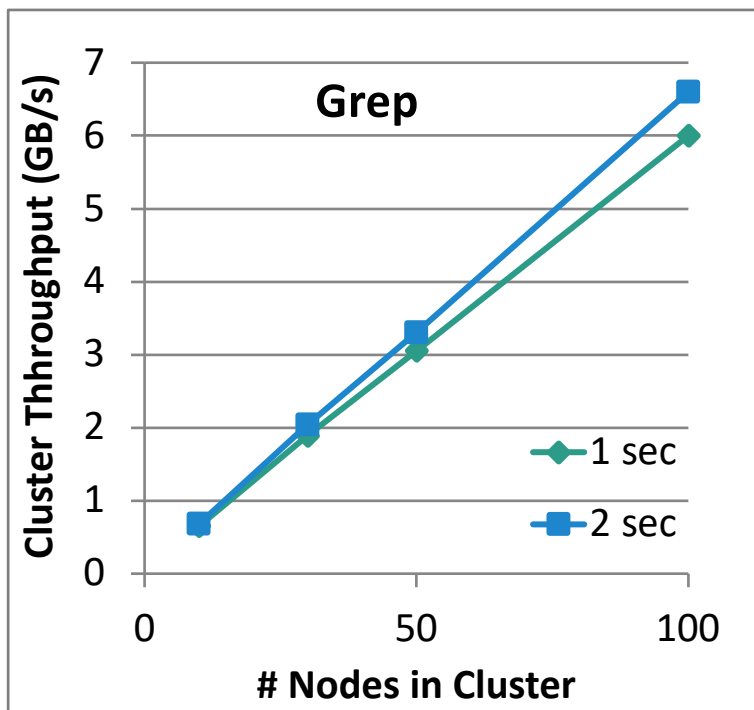
- All transformations are fault-tolerant, and *exactly-once* transformations

Fault-tolerance

- Received data is **replicated** among multiple Spark executors
 - Default factor: 2
- **Checkpointing**
 - Saves state on regular basis, typically every 5-10 batches of data
 - A failure would have to replay the 5-10 previous batches to recreate the appropriate RDDs
 - Checkpoint done to HDFS or equivalent
- Must protect the **driver program**
 - If the driver node running the Spark Streaming application fails
 - Driver must be restarted on another node.
 - Requires a checkpoint directory in the StreamingContext
- **Streaming Backpressure**
 - `spark.streaming.backpressure.enabled`
 - `spark.streaming.receiver.maxRate`

Performance

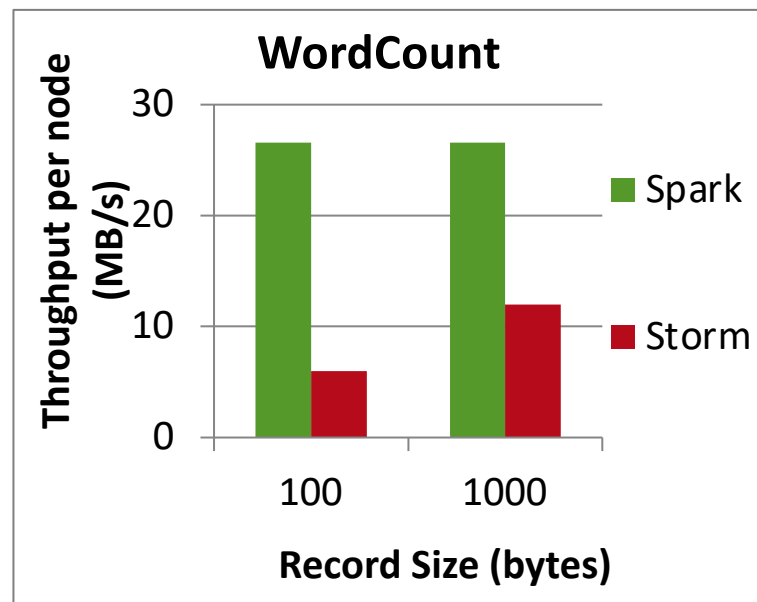
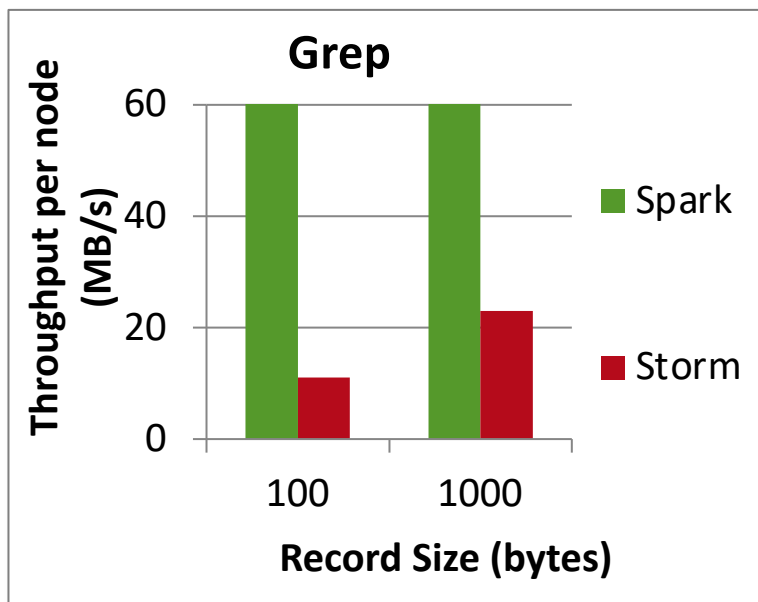
Can process **60M records/sec (6 GB/sec)** on
100 nodes at **sub-second** latency



Comparison with other systems

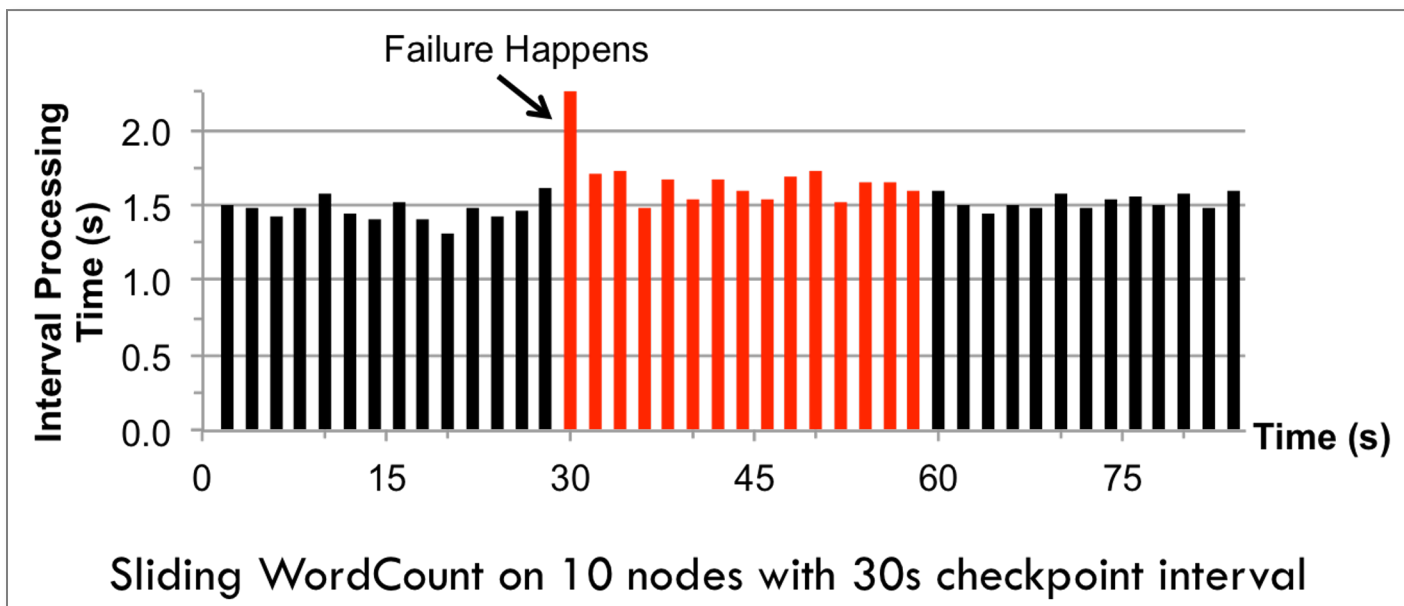
Higher throughput than Storm

- Spark Streaming: **670k** records/sec/node
- Storm: **115k** records/sec/node
- Commercial systems: **100-500k** records/sec/node



Fast Fault Recovery

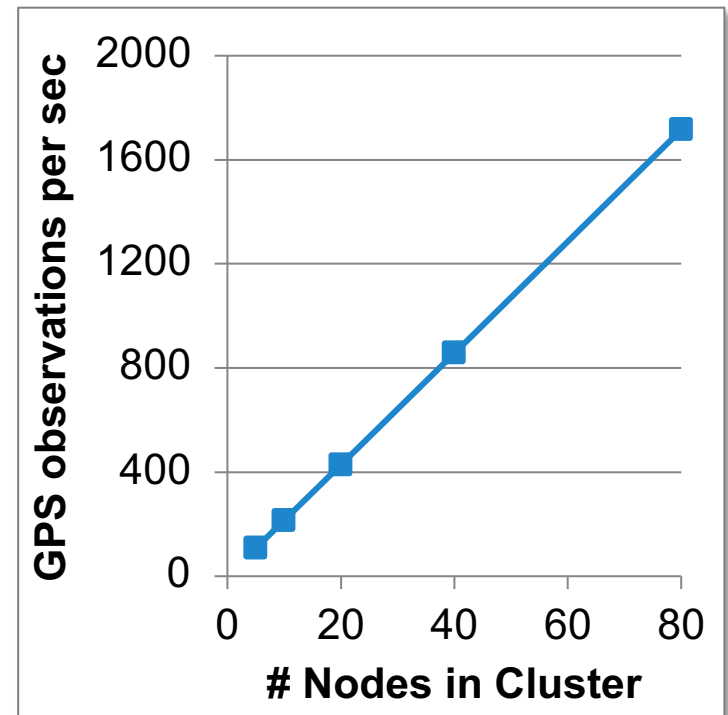
Recovers from faults/stragglers within **1 sec**



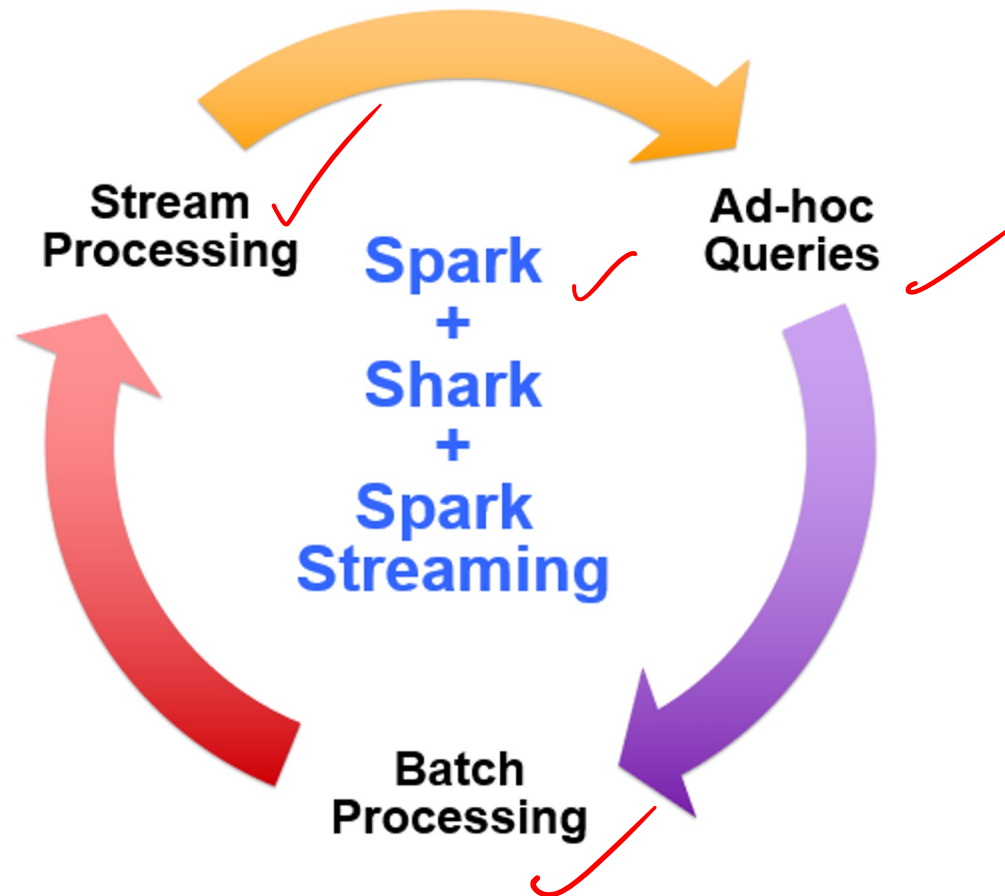
Real time application: Mobile Millennium Project

Traffic transit time estimation using online machine learning on GPS observations

- Markov-chain Monte Carlo simulations on GPS observations
- Very CPU intensive, requires dozens of machines for useful computation
- Scales linearly with cluster size



Vision - *one stack to rule them all*



Spark program vs Spark Streaming program

Spark Streaming program on Twitter stream

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)  
val hashTags = tweets.flatMap(status => getTags(status))  
hashTags.saveAsHadoopFiles("hdfs://...")
```

Spark program on Twitter log file

```
val tweets = sc.hadoopFile("hdfs://...")  
val hashTags = tweets.flatMap(status => getTags(status))  
hashTags.saveAsHadoopFile("hdfs://...")
```

Advantage of an unified stack

- Explore data interactively to identify problems
- Use same code in Spark for processing large logs
- Use similar code in Spark Streaming for realtime processing

```
$ ./spark-shell
scala> val file = sc.hadoopFile("smallLogs")
...
scala> val filtered = file.filter(_.contains("ERROR"))
...
scala> val mapped = filtered.map(...)
```

```
• object ProcessProductionData {
  def main(args: Array[String]) {
    val sc = new SparkContext(...)
    val file = sc.hadoopFile("productionLogs")
    val filtered = file.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

```
} object ProcessLiveStream {
  def main(args: Array[String]) {
    val sc = new StreamingContext(...)
    val stream = sc.kafkaStream(...)
    val filtered = stream.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

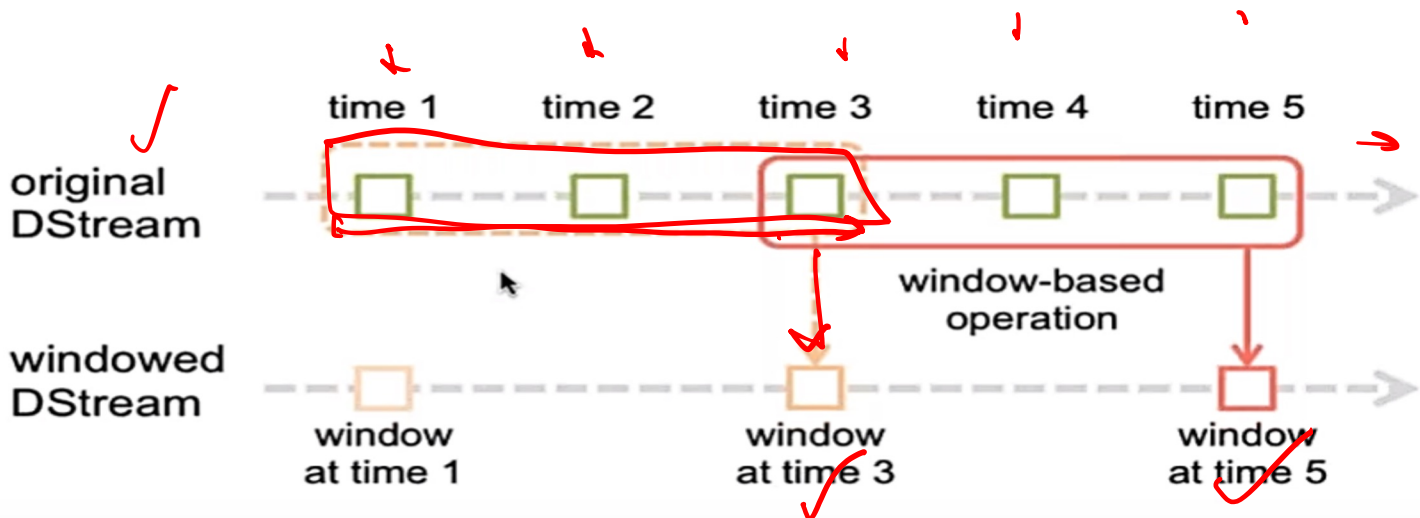
Roadmap

- Spark 0.8.1
 - Marked alpha, but has been quite stable
 - Master fault tolerance – manual recovery
 - Restart computation from a checkpoint file saved to HDFS
- Spark 0.9 in Jan 2014 – out of alpha!
 - Automated master fault recovery
 - Performance optimizations
 - Web UI, and better monitoring capabilities
- Spark v2.4.0 released in November 2, 2018

Sliding Window Analytics

Spark Streaming Windowing Capabilities

- **Parameters**
 - **Window length:** duration of the window
 - **Sliding interval:** interval at which the window operation is performed
 - Both the parameters must be a multiple of the batch interval
- A window creates a new DStream with a larger batch size



Spark Window Functions

Spark Window Functions for DataFrames and SQL

Introduced in Spark 1.4, Spark window functions improved the expressiveness of Spark DataFrames and Spark SQL. With window functions, you can easily calculate a moving average or cumulative sum, or reference a value in a previous row of a table. Window functions allow you to do many common calculations with DataFrames, without having to resort to RDD manipulation.

Aggregates, UDFs vs. Window functions

Window functions are complementary to existing DataFrame operations: aggregates, such as sum and avg, and UDFs. To review, aggregates calculate one result, a sum or average, for each group of rows, whereas UDFs calculate one result for each row based on only data in that row. In contrast, window functions calculate one result for each row based on a window of rows. For example, in a moving average, you calculate for each row the average of the rows surrounding the current row; this can be done with window functions.

Moving Average Example

- Let us dive right into the moving average example. In this example dataset, there are two customers who have spent different amounts of money each day.
- // Building the customer DataFrame. All examples are written in Scala with Spark 1.6.1, but the same can be done in Python or SQL.

```
val customers = sc.parallelize(List("Alice", "2016-05-01", 50.00),  
    ("Alice", "2016-05-03", 45.00),  
    ("Alice", "2016-05-04", 55.00),  
    ("Bob", "2016-05-01", 25.00),  
    ("Bob", "2016-05-04", 29.00),  
    ("Bob", "2016-05-06", 27.00))).  
toDF("name", "date", "amountSpent")
```

Moving Average Example

// Import the window functions.

```
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._
```

// Create a window spec.

```
val wSpec1 =  
Window.partitionBy("name").orderBy("date").rowsBetween(-1, 1)
```



- In this window spec, the data is partitioned by customer. Each customer's data is ordered by date. And, the window frame is defined as starting from -1 (one row before the current row) and ending at 1 (one row after the current row), for a total of 3 rows in the sliding window.

Moving Average Example

// Calculate the moving average

```
customers.withColumn( "movingAvg",  
                      avg(customers("amountSpent")).over(wSpec1) ).show()
```

This code adds a new column, “movingAvg”, by applying the avg function on the sliding window defined in the window spec:

name	date	amountSpent	movingAvg
Alice	5/1/2016	50	47.5
Alice	5/3/2016	45	50
Alice	5/4/2016	55	50
Bob	5/1/2016	25	27
Bob	5/4/2016	29	27
Bob	5/6/2016	27	28

Window function and Window Spec definition

- As shown in the above example, there are two parts to applying a window function: (1) specifying the window function, such as avg in the example, and (2) specifying the window spec, or wSpec1 in the example. For (1), you can find a full list of the window functions here:
- <https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions>
- You can use functions listed under “Aggregate Functions” and “Window Functions”.
- For (2) specifying a window spec, there are three components: partition by, order by, and frame.
 1. “Partition by” defines how the data is grouped; in the above example, it was by customer. You have to specify a reasonable grouping because all data within a group will be collected to the same machine. Ideally, the DataFrame has already been partitioned by the desired grouping.
 2. “Order by” defines how rows are ordered within a group; in the above example, it was by date.
 3. “Frame” defines the boundaries of the window with respect to the current row; in the above example, the window ranged between the previous row and the next row.

Cumulative Sum

Next, let us calculate the cumulative sum of the amount spent per customer.

// Window spec: the frame ranges from the beginning (Long.MinValue) to the current row (0).

```
val wSpec2 =
```

```
Window.partitionBy("name").orderBy("date").rowsBetween(Long.MinValue, 0)
```

// Create a new column which calculates the sum over the defined window frame.

```
customers.withColumn( "cumSum",  
    sum(customers("amountSpent")).over(wSpec2) ).show()
```

name	date	amountSpent	cumSum
Alice	5/1/2016	50	50
Alice	5/3/2016	45	95
Alice	5/4/2016	55	150
Bob	5/1/2016	25	25
Bob	5/4/2016	29	54
Bob	5/6/2016	27	81

Data from previous row

In the next example, we want to see the amount spent by the customer in their previous visit.

// Window spec. No need to specify a frame in this case.

```
val wSpec3 = Window.partitionBy("name").orderBy("date")
```

// Use the lag function to look backwards by one row.

```
customers.withColumn("prevAmountSpent",  
  lag(customers("amountSpent"), 1).over(wSpec3) ).show()
```

name	date	amountSpent	prevAmountSpent
Alice	5/1/2016	50	null
Alice	5/3/2016	45	50
Alice	5/4/2016	55	45
Bob	5/1/2016	25	null
Bob	5/4/2016	29	25
Bob	5/6/2016	27	29

Rank

- In this example, we want to know the order of a customer's visit (whether this is their first, second, or third visit).

// The rank function returns what we want.

```
customers.withColumn( "rank", rank().over(wSpec3) ).show()
```

name	date	amountSpent	rank
Alice	5/1/2016	50	1
Alice	5/3/2016	45	2
Alice	5/4/2016	55	3
Bob	5/1/2016	25	1
Bob	5/4/2016	29	2
Bob	5/6/2016	27	3

Case Study: Twitter Sentiment Analysis with Spark Streaming

Case Study: Twitter Sentiment Analysis

- Trending Topics can be used to create campaigns and attract larger audience. Sentiment Analytics helps in crisis management, service adjusting and target marketing.
- Sentiment refers to the emotion behind a social media mention online.
- Sentiment Analysis is categorising the tweets related to particular topic and performing data mining using Sentiment Automation Analytics Tools.
- We will be performing Twitter Sentiment Analysis as an Use Case or Spark Streaming.



Figure: Facebook And Twitter Trending Topics

Problem Statement

- To design a Twitter Sentiment Analysis System where we populate real-time sentiments for crisis management, service adjusting and target marketing.

Sentiment Analysis is used to:

- Predict the success of a movie ✓
- Predict political campaign success ✓
- Decide whether to invest in a certain company
- Targeted advertising
- Review products and services

Importing Packages

//Import the necessary packages into the Spark Program

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkContext._
import org.apache.spark.streaming.twitter._
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._
import org.apache.spark.rdd._
import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext._
import org.apache.spark.sql
import org.apache.spark.storage.StorageLevel
import scala.io.Source
import scala.collection.mutable.HashMap
import java.io.File
```

Twitter Token Authorization

```
object mapr {  
  
  def main(args: Array[String]) {  
    if (args.length < 4) {  
      System.err.println("Usage: TwitterPopularTags <consumer key>  
<consumer secret> " +  
        "<access token> <access token secret> [<filters>]")  
      System.exit(1)  
    }  
  
    StreamingExamples.setStreamingLogLevels()  
    //Passing our Twitter keys and tokens as arguments for authorization  
    val Array(consumerKey, consumerSecret, accessToken,  
      accessTokenSecret) = args.take(4)  
    val filters = args.takeRight(args.length - 4)
```

DStream Transformation

```
// Set the system properties so that Twitter4j library used by twitter stream
// Use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
System.setProperty("twitter4j.oauth.accessToken", accessToken)
System.setProperty("twitter4j.oauth.accessTokenSecret",
accessTokenSecret)

val sparkConf = new
SparkConf().setAppName("Sentiments").setMaster("local[2]")
val ssc = new StreamingContext(sparkConf, Seconds(5))
val stream = TwitterUtils.createStream(ssc, None, filters)

//Input DStream transformation using flatMap
val tags = stream.flatMap { status =>
status.getHashtagEntities.map(_.getText) }
```

Results

```
Markers Properties Servers Data Source Explorer Snippets Console Scala Interpreter (TwitterStreaming)
<terminated> mapr$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 11:56:26 AM)
debug: weighted: 1.0
-----
Time: 1486621640000 ms
-----
(東芝、半導体新棟を着工=メモリー製造、18年夏完成へ https://t.co/DU5goZAp25 #不動産 #投資 #マネー #株 #市況 #拡散, NEGATIVE, [Ljava.lang.String;@1a25ec3)
(RT @bts_bighit: [투표] Q. 투표하라는 말 지겹다?
아미: 좋아요~ 짜릿해! ♪ #방탄소년단 투표하는 게 최고야 ♪

#가온차트어워드 https://t.co/OHDWR2smt4
#ShortyAwards https://t.co/..., NEGATIVE, [Ljava.lang.String;@121986a)
(RT @MukePL: Jeżeli na tym zdjęciu widzisz swój świat to daj RT. ♡ #oneDbestfans & #5S0Sbestfans ♡ https://t.co/rn2EmNvjFp, NEGATIVE, [Ljava.lang.String;@1c3681d)
(RT @Horocasts: #Cancer most enduring quality is an unexpected silly sense of humor. POSITIVE, [Ljava.lang.String;@174e1a2)
(I'm listening to "A Song For Mama" by @BoyzIIMen on @PandoraMusic. #pandora https://t.co/71n5Rw3CY0, NEUTRAL, [Ljava.lang.String;@95f6d4)
('Greenwashing' Costing Walmart $1 Million https://t.co/D8X02RZMnP #Biodegradability #Compostability #biobased, NEGATIVE, [Ljava.lang.String;@1511e25)
(RT @camilasxdinah: Serayah representando a las camilizers cuando un hombre se le acerca a Camila #CamilaBestFans https://t.co/8IggLo3RGn, NEGATIVE, [Ljava.lang.String;@78c835)
(RT @CamiIaVoteStats: #CamilaBestFans https://t.co/qsLxPQpD1n, NEUTRAL, [Ljava.lang.String;@16e7255)
(@tos 六甲道駅 https://t.co/0rKl8rlSb3 #TFB, NEGATIVE, [Ljava.lang.String;@1a3fe)
(Ilmar pro Marcos: "Vai dormir puta.. Bebe e fica aí com o cu quente." KKKKKKKKKKKKKKKKKKKKKKKKKKKKK #BBB17, NEGATIVE, [Ljava.lang.String;@1516ece)
...

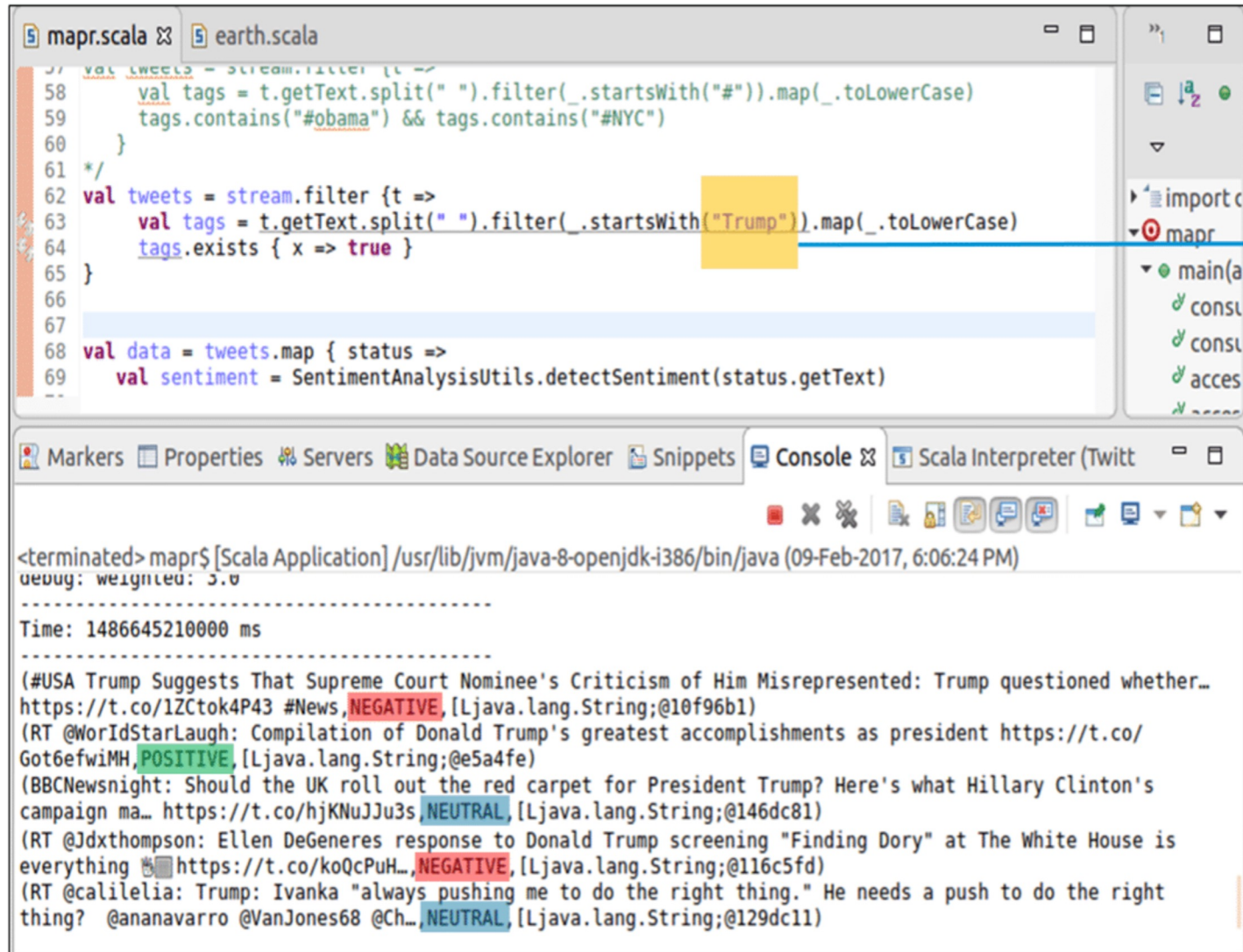
Adding annotator tokenize
```

Positive

Neutral

Negative

Sentiment for Trump



```
mapr.scala | earth.scala
57 val tweets = stream.filter { t =>
58   val tags = t.getText.split(" ").filter(_.startsWith("#")).map(_.toLowerCase)
59   tags.contains("#obama") && tags.contains("#NYC")
60 }
61 */
62 val tweets = stream.filter { t =>
63   val tags = t.getText.split(" ").filter(_.startsWith("Trump")).map(_.toLowerCase)
64   tags.exists { x => true }
65 }
66
67
68 val data = tweets.map { status =>
69   val sentiment = SentimentAnalysisUtils.detectSentiment(status.getText)
70 }
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console | Scala Interpreter (Twitt)

<terminated> mapr\$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 6:06:24 PM)
debug: weighted: 5.0

Time: 1486645210000 ms

(#USA Trump Suggests That Supreme Court Nominee's Criticism of Him Misrepresented: Trump questioned whether...
https://t.co/1ZCtok4P43 #News, **NEGATIVE**, [Ljava.lang.String;@10f96b1)
(RT @WorIdStarLaugh: Compilation of Donald Trump's greatest accomplishments as president https://t.co/
Got6efwiMH, **POSITIVE**, [Ljava.lang.String;@e5a4fe)
(BBCNewsnight: Should the UK roll out the red carpet for President Trump? Here's what Hillary Clinton's
campaign ma... https://t.co/hjKNuJJU3s, **NEUTRAL**, [Ljava.lang.String;@146dc81)
(RT @Jdxthompson: Ellen DeGeneres response to Donald Trump screening "Finding Dory" at The White House is
everything 🐼 https://t.co/koQcPuH..., **NEGATIVE**, [Ljava.lang.String;@116c5fd)
(RT @calilelia: Trump: Ivanka "always pushing me to do the right thing." He needs a push to do the right
thing? @ananavarro @VanJones68 @Ch..., **NEUTRAL**, [Ljava.lang.String;@129dc11)

'Trump'
Keyword

Positive
Neutral
Negative

Applying Sentiment Analysis

- As we have seen from our Sentiment Analysis demonstration, we can extract sentiments of particular topics just like we did for 'Trump'. Similarly, Sentiment Analytics can be used in crisis management, service adjusting and target marketing by companies around the world.
- Companies using Spark Streaming for Sentiment Analysis have applied the same approach to achieve the following:
 1. Enhancing the customer experience
 2. Gaining competitive advantage
 3. Gaining Business Intelligence
 4. Revitalizing a losing brand

References

- <https://spark.apache.org/streaming/>
- Streaming programming guide –
spark.incubator.apache.org/docs/latest/streaming-programming-guide.html
- <https://databricks.com/speaker/tathagata-das>

Conclusion

- Stream processing framework that is ...
 - Scalable to large clusters ✓
 - Achieves second-scale latencies
 - Has simple programming model
 - Integrates with batch & interactive workloads
 - Ensures efficient fault-tolerance in stateful computations
- Handwritten notes:*
- hot data path only
 - real time streaming and fast data

Introduction to MQTT and Kafka in IoT Platform



Dr. Rajiv Misra, Professor

**Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in**

Preface

Content of this Lecture:

- Define MQTT and Kafka
- Integration of MQTT and Kafka
- Describe the Kafka data model
- Describe Kafka architecture
- List the types of messaging systems
- Explain the importance of brokers

Introduction: Internet of Things with MQTT

MQTT (Message Queuing Telemetry Transport): ✓

AWS IoT
Azure IoT Hub

MQTT is a widely used ISO standard (ISO/IEC PRF 20922) client-server messaging protocol.

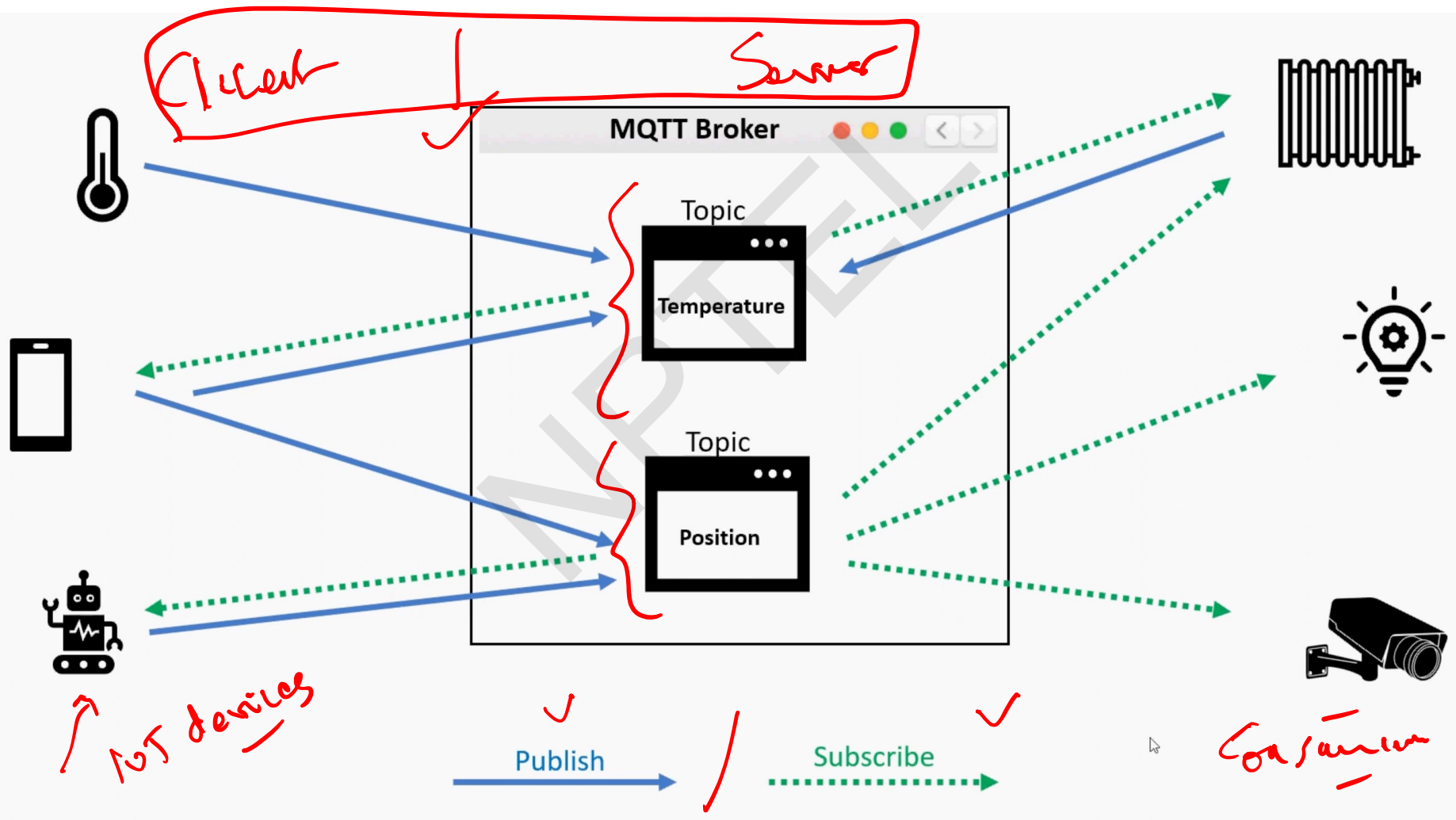
The protocol is lightweight and implements a publish/subscribe communication pattern.

MQTT is stable in unreliable environments of high latency and low network bandwidth which makes it a perfect match for Internet of Things scenarios like connected cars or smart homes.

MQTT has many implementations of client libraries and brokers like Mosquitto, HiveMQ, JoramMQ, etc and its primary purpose is to connect millions of devices — especially in the IoT context.

Introduction: Internet of Things with MQTT

MQTT (Message Queuing Telemetry Transport):



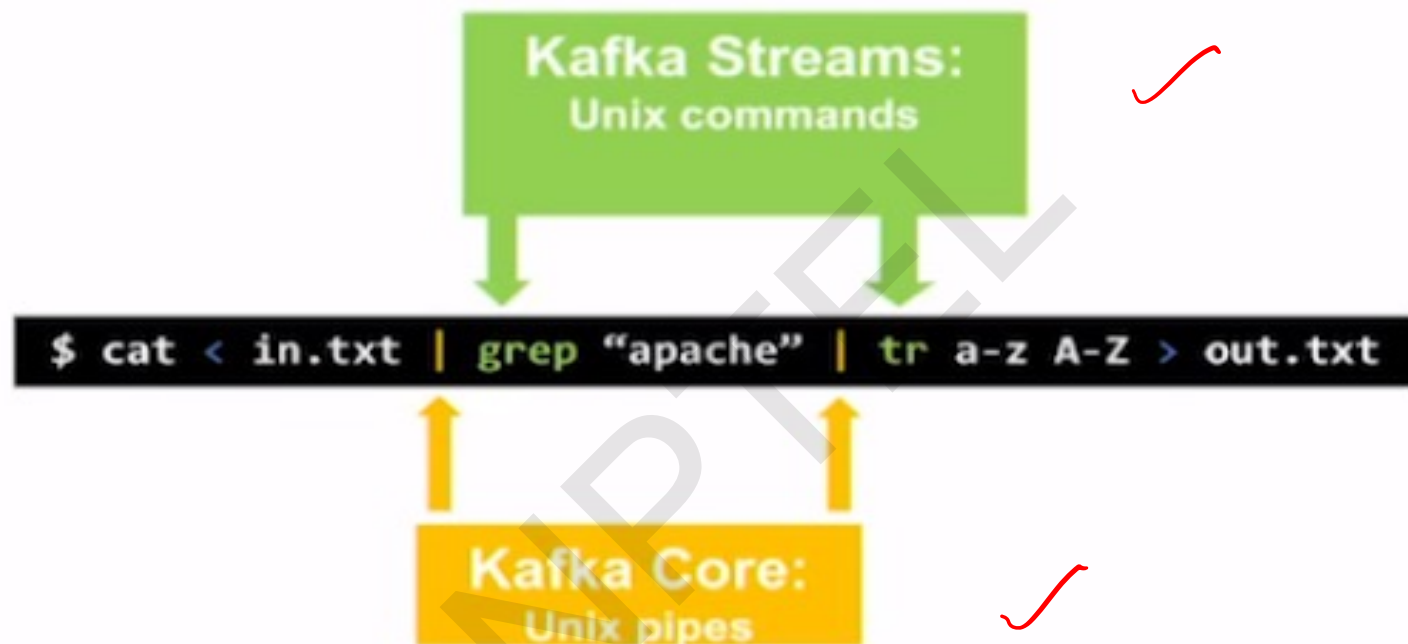
Introduction: Internet of Things Streaming using Kafka

- Kafka was initially created by LinkedIn and later distributed on the Apache License. Kafka is still open-source. Additionally, a company named Confluent is offering Confluent Platform with Kafka as a managed service providing several additional features around like Schema Registry, REST & MQTT Proxies, and specific connectors.
- Kafka implements an own protocol under the hood, following a publish/subscribe pattern which structures communication into topics — similar to MQTT. However, that's the only thing both have in common.

Introduction: Internet of Things/Streaming using Kafka

- Kafka is designed to be deployable as a cluster of multiple nodes which makes it excellent for scaling. Additionally, it offers persistent storage of messages and integration to business on-premise or cloud data centers and applications. *hot data on disk*
- Its main use cases are distributed event streaming and storage/consumption of massive amounts of data as messages.
- It makes Kafka a perfect match for scenarios that require high-performance, scalable data pipelines, or data integration across multiple systems.

Introduction: Internet of Things Streaming using Kafka



- **Kafka Core:** is the **distributed, durable equivalent** of Unix pipes. Use it to connect and compose your large-scale data applications.
- **Kafka Streams** are the commands of your Unix pipelines. Use it to transform data stored in Kafka.
- **Kafka Connect** is the I/O redirection in your Unix pipelines. Use it to get your data into and out of Kafka.

Introduction: Internet of Things Streaming using Kafka

- **Kafka is a high-performance, real-time messaging system. It is an open source tool and is a part of Apache projects.**
- The characteristics of Kafka are:
 1. It is a distributed and partitioned messaging system.
 2. It is highly fault-tolerant
 3. It is highly scalable.
 4. It can process and send millions of messages per second to several receivers.

Kafka Use Cases

- Kafka can be used for various purposes in an organization, such as:

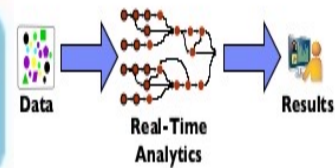
Messaging service

Millions of messages can be sent and received in real-time, using Kafka.



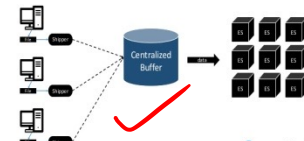
Real-time stream processing

Kafka can be used to process a continuous stream of information in real-time and pass it to stream processing systems such as Storm.



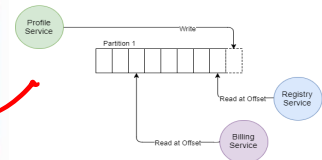
Log aggregation

Kafka can be used to collect physical log files from multiple systems and store them in a central location such as HDFS.



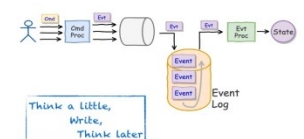
Commit log service

Kafka can be used as an external commit log for distributed systems.



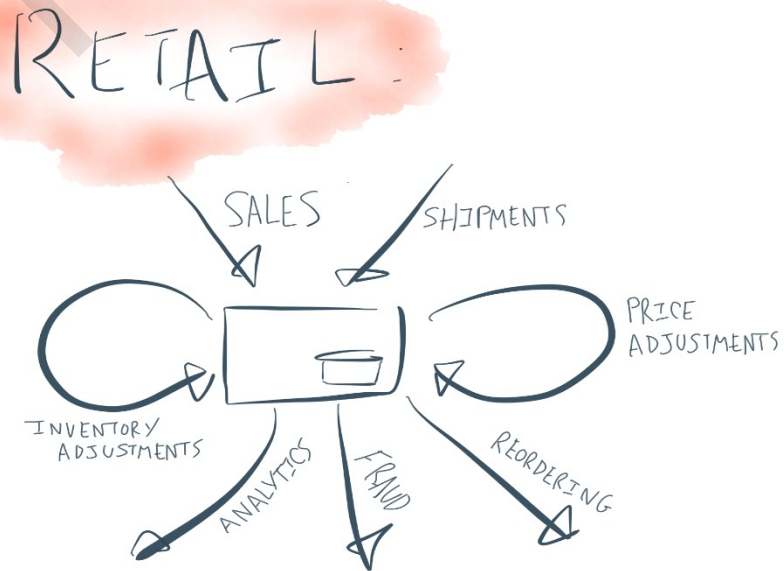
Event sourcing

A time ordered sequence of events can be maintained through Kafka.



Apache Kafka: a Streaming Data Platform

- Most of **what a business does** can be thought as **event streams**. They are in a
- **Retail system**: orders, shipments, returns, ...
 - **Financial system**: stock ticks, orders, ...
 - **Web site**: page views, clicks, searches, ...
 - **IoT**: sensor readings, ...
- and so on.



Why using both MQTT and Kafka?

If you need to build performant data pipelines, store massive amounts of messages, or integrate different business applications or data centers in real-time — use Kafka.

If you have lots of small applications or devices, running in unwired or unstable environments, exchanging messages in real-time on numerous different channels/topics — use MQTT.

There are two things that make it quite obvious to combine the two technologies:

- the communication structure in topics and
- the publish/subscribe message exchange pattern.

But in which scenarios would you use both Kafka and MQTT together?
Let's see in further slides.

Use Case: Why using both MQTT and Kafka?

The most popular use case is probably the integration of MQTT devices with backend applications for monitoring, control, or analytics running in the companies' data centers or the cloud.


Imagine you want to send data from different IoT devices to a backend application for machine learning based pattern recognition or analytics. At the same time, the backend application should send back messages to control the IoT device based on the central insights (e.g. send control messages to avoid a device from overheating, ...).

Consequently, MQTT and Kafka are a perfect combination for end-to-end IoT integration from the edge to the business applications' and data centers.

The IoT/edge devices can connect to the MQTT broker via MQTT protocol (with all the advantages it has in these environments).

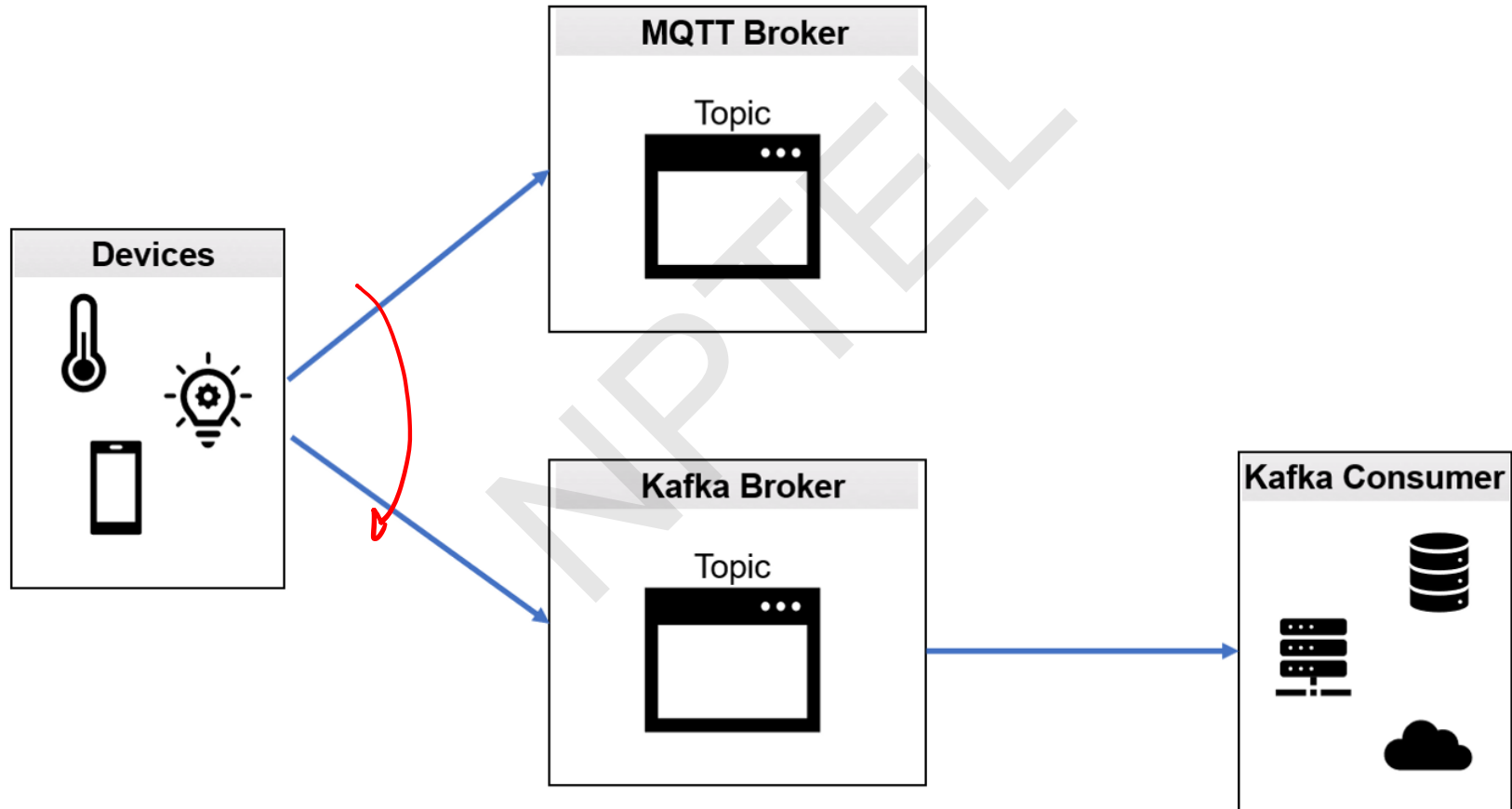
The messages are then forwarded to Kafka to distribute them into the subscribing business applications and the other way around.

Integration between MQTT and Kafka

- 
1. IoT device to publish two messages
 2. Build an MQTT Bridge to Kafka
 3. Connect to Kafka via MQTT proxy
 4. Connect MQTT Broker to Kafka via Kafka Connect
 5. Connect MQTT Broker to Kafka via MQTT Broker extension

Integration between MQTT and Kafka

IoT device to publish two messages:



Integration between MQTT and Kafka

IoT device to publish two messages:

The IoT device can publish two messages — one to the topic of the MQTT broker and a second one to the topic of the Kafka broker. This has several drawbacks:

- The IoT device needs to check the delivery guarantees of both protocols and it must be ensured that the message is received by both or not at all. A lot of investment in error handling must be done.
- Additionally, most IoT devices are lightweight. Sending two messages with two different protocols is a huge overhead. Most IoT devices might have not even the possibility to connect to Kafka natively.
- Kafka is not designed to handle a massive amount of different topics with millions of different devices. A full-blown IoT scenario with this integration option could lead to issues on the Kafka broker side.

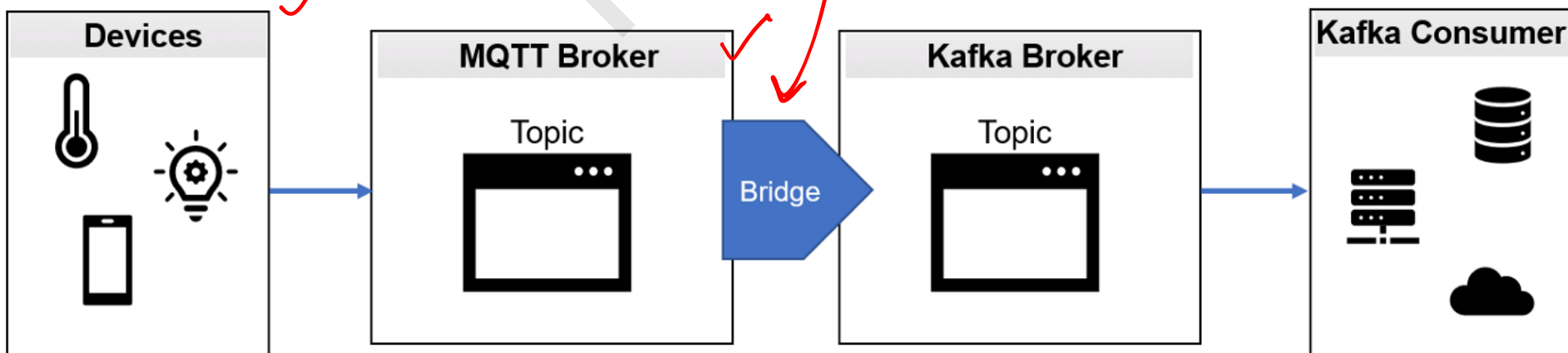
Integration between MQTT and Kafka

Scenario 1: Build an MQTT Bridge to Kafka

End-to-end / Delivery

Another alternative of connecting Kafka and MQTT is building a custom application as a bridge between the MQTT and Kafka broker. This application needs to use an MQTT client library to connect to the MQTT broker and a Kafka client library to connect to the Kafka broker and consequently subscribe to the relevant topics and publish the messages in the desired direction.

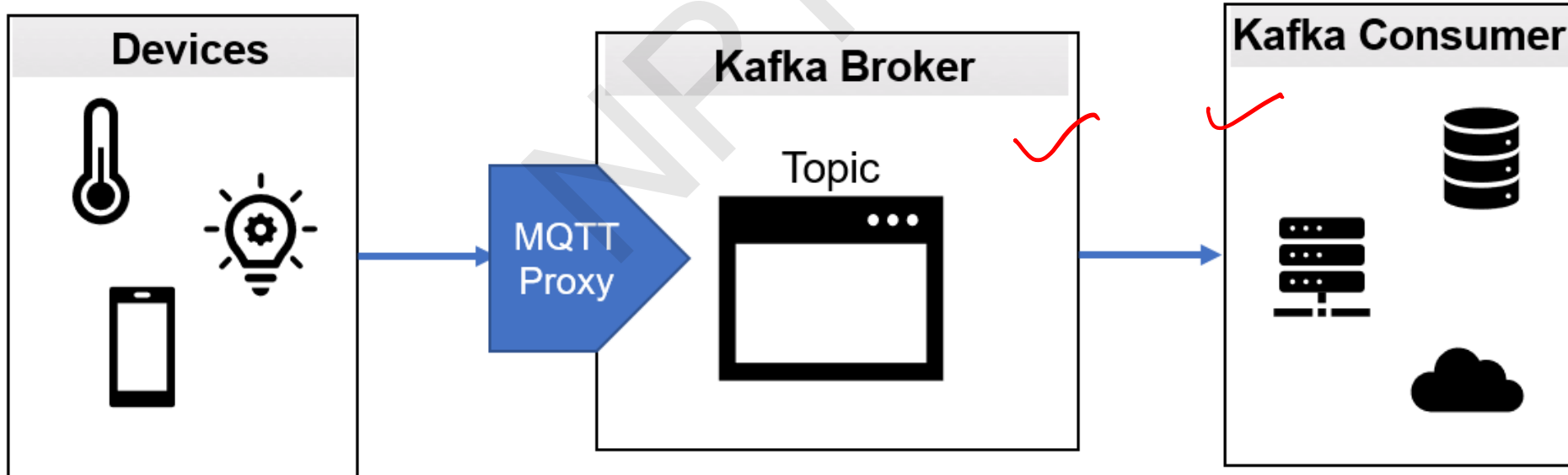
In this context resilience and fault tolerance are very important, but hard to reach, especially if an end-to-end guaranty of at least once or exactly once message delivery is required. The custom bridge application can only acknowledge the MQTT receipt if it successfully forwarded the message to the Kafka broker or need to buffer the messages in case something goes wrong. A significant development effort in error handling and functionality similar to technology already found in Kafka an/or MQTT broker is necessary.



Integration between MQTT and Kafka

Scenario 2: Connect to Kafka via MQTT proxy

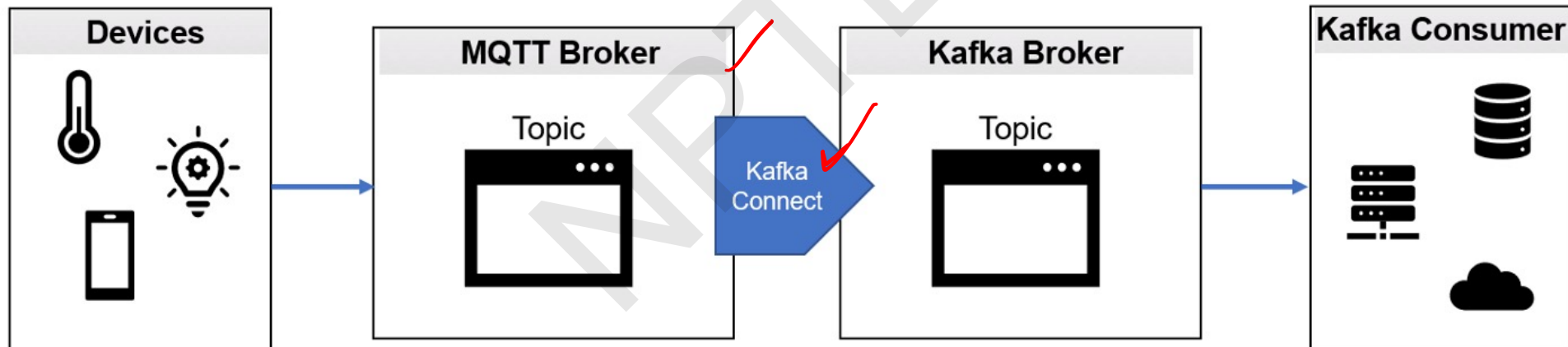
If the only requirement is to persist MQTT messages or integrate them with legacy systems, this option could be a good fit. In this case, the Confluent Kafka MQTT proxy can be used by the IoT devices to directly publish the messages to Kafka. An MQTT broker would be additional overhead and would be simply removed from the picture.



Integration between MQTT and Kafka

Scenario 3: Connect MQTT Broker to Kafka via Kafka Connect

Kafka Connect is an extension framework providing different connectors for data ingestion to or data query from Kafka for multiple technologies or software vendors. Kafka Connect provides an MQTT connector out of the box which represents an MQTT client that can subscribe to the MQTT brokers topics.

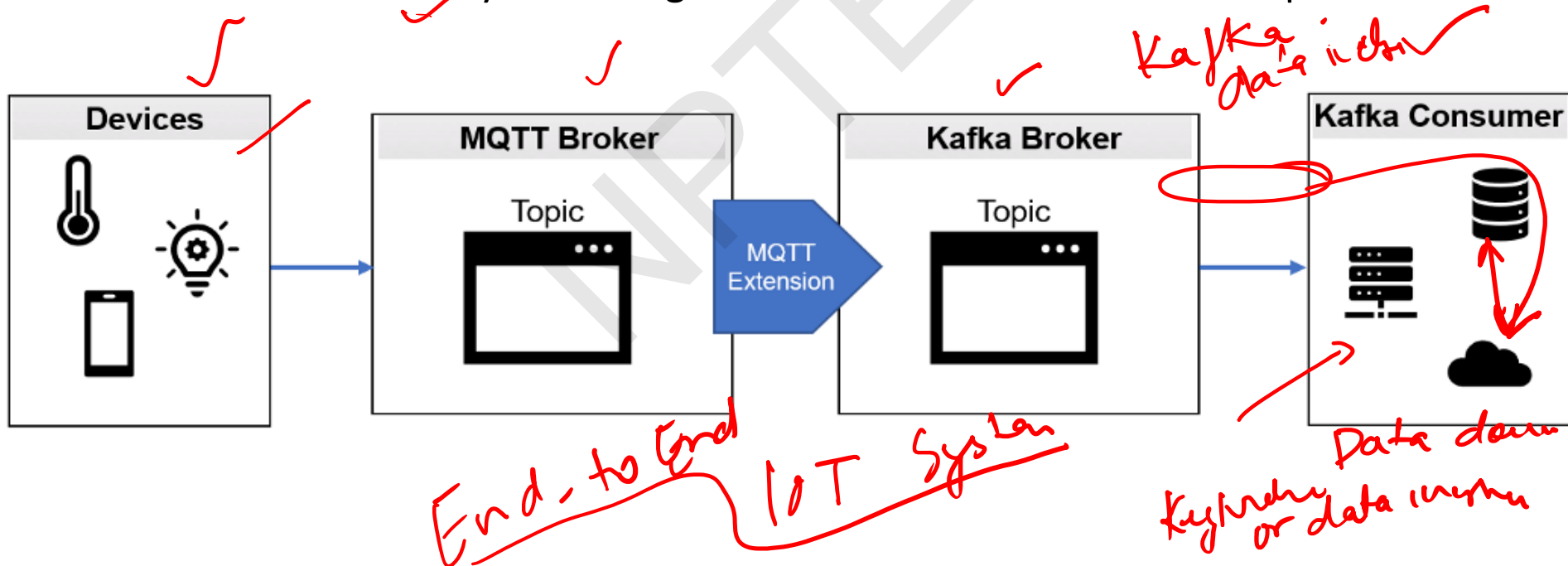


Integration between MQTT and Kafka

Scenario 4 : Connect MQTT Broker to Kafka via MQTT Broker extension

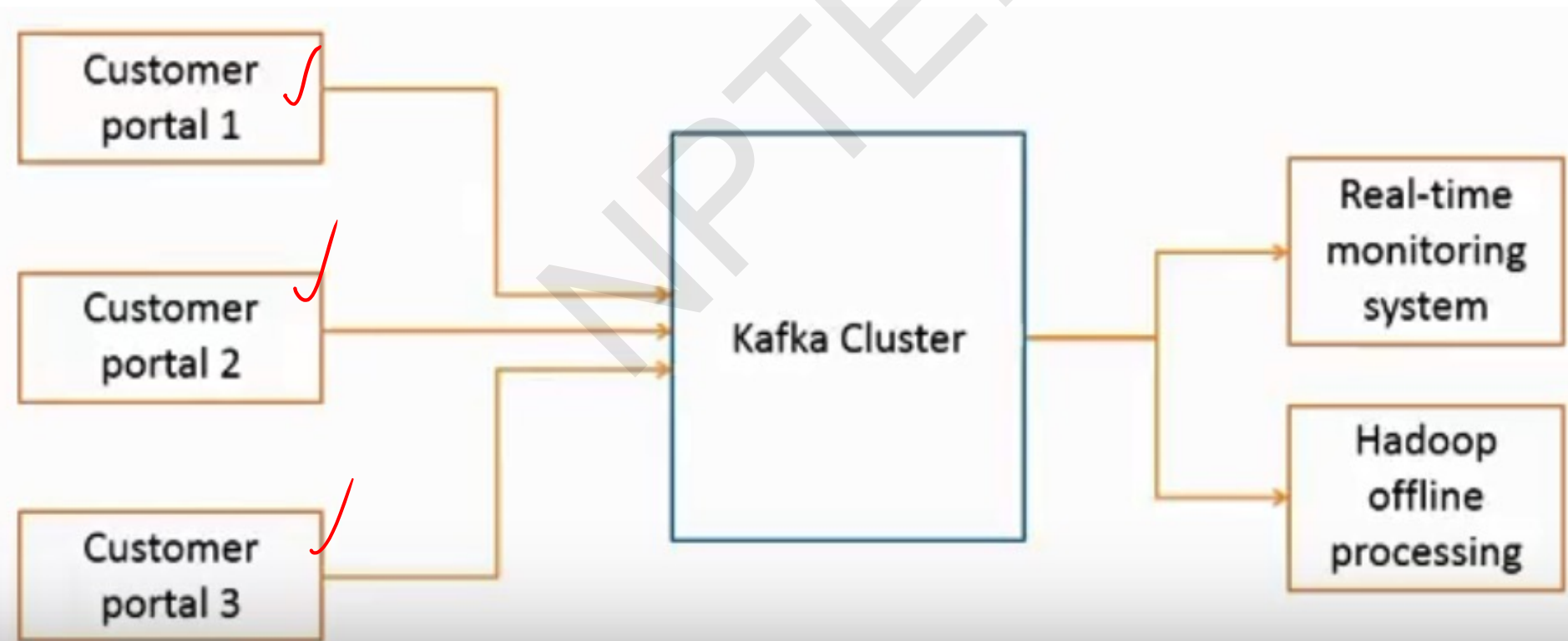
Another approach is to implement a Kafka client as an extension on the MQTT broker. This allows the MQTT broker to ingest the IoT device messages to the Kafka broker/cluster.

Some MQTT providers like EMQ or HiveMQ have already implemented the bridging of MQTT broker and Kafka by extending their brokers with a native Kafka protocol.



Aggregating User Activity Using Kafka-Example

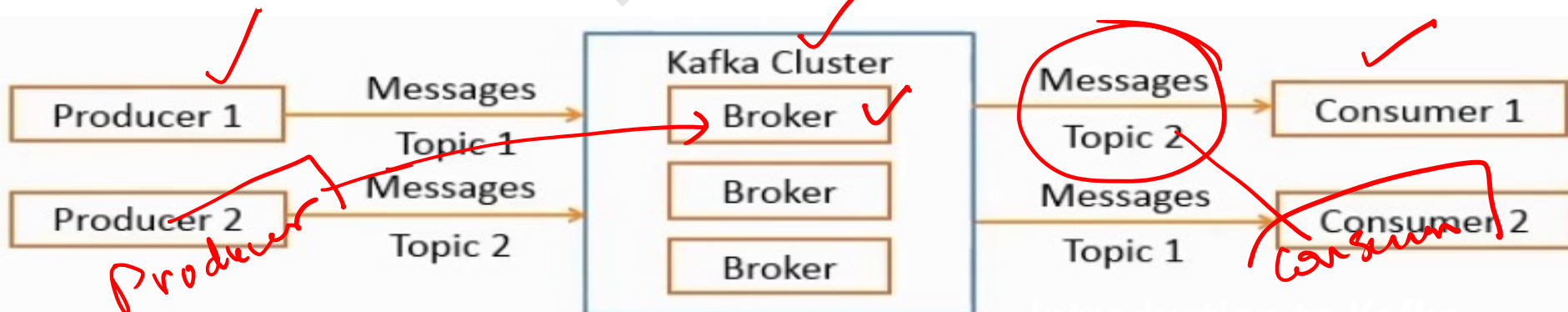
- Kafka can be used to aggregate user activity data such as clicks, navigation, and searches from different websites of an organization; such user activities can be sent to a real-time monitoring system and hadoop system for offline processing.



Kafka Data Model

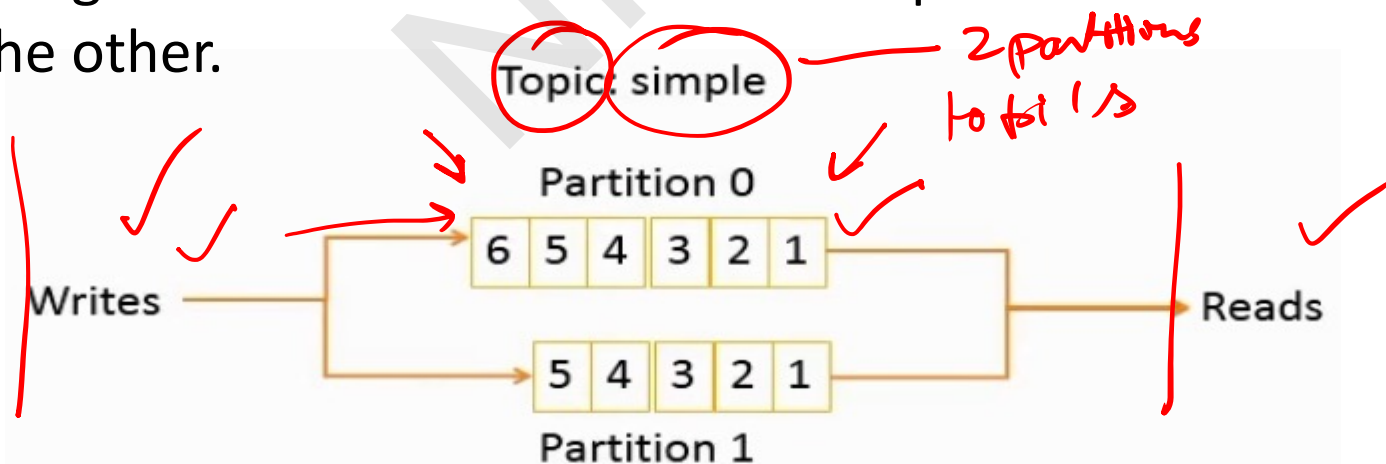
The Kafka data model consists of messages and topics.

- Messages represent information such as, lines in a log file, a row of stock market data, or an error message from a system.
- Messages are grouped into categories called topics. Example: LogMessage and Stock Message.
- The processes that publish messages into a topic in Kafka are known as producers.
- The processes that receive the messages from a topic in Kafka are known as consumers.
- The processes or servers within Kafka that process the messages are known as brokers.
- A Kafka cluster consists of a set of brokers that process the messages.



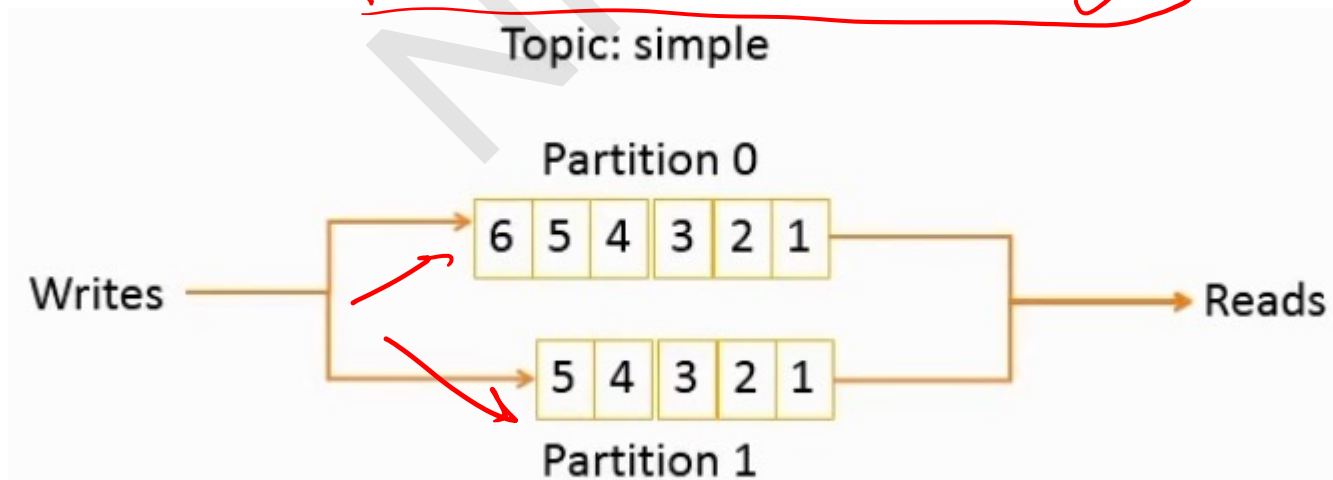
Topics ✓

- A topic is a category of messages in Kafka.
- The producers publish the messages into topics. ✓
- The consumers read the messages from topics. ✓
- A topic is divided into one or more partitions. ✓
- A partition is also known as a commit log. ✓
- Each partition contains an ordered set of messages. ✓
- Each message is identified by its offset in the partition. ✓
- Messages are added at one end of the partition and consumed at the other. ✓



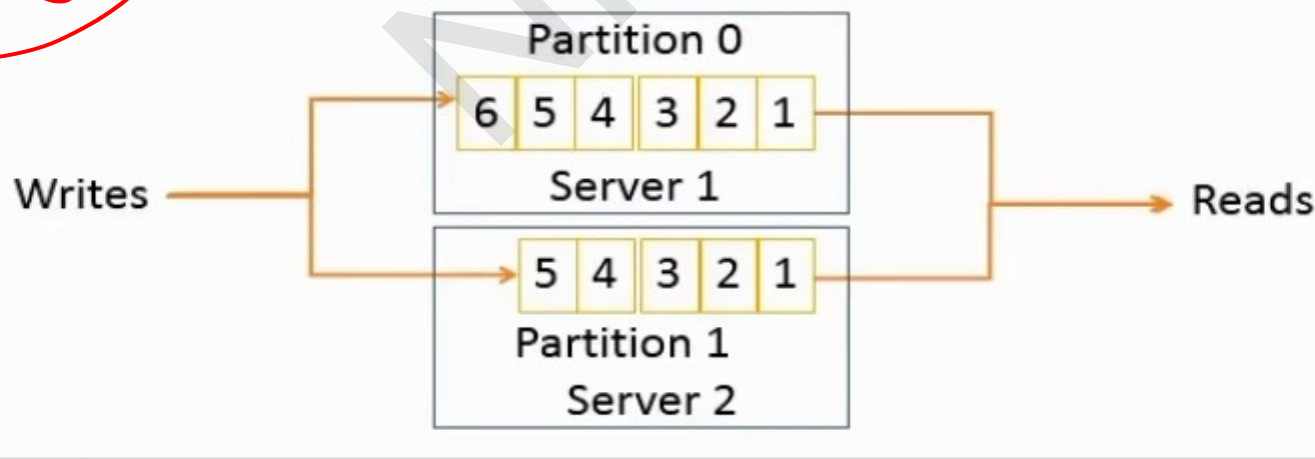
Partitions ✓

- Topics are divided into partitions, which are the unit of parallelism in Kafka. ✓
- Partitions allow messages in a topic to be distributed to multiple servers. ✓
- A topic can have any number of partitions. ✓
- Each partition should fit in a single Kafka server. ✓
- The number of partitions decide the parallelism of the topic. ✓



Partition Distribution

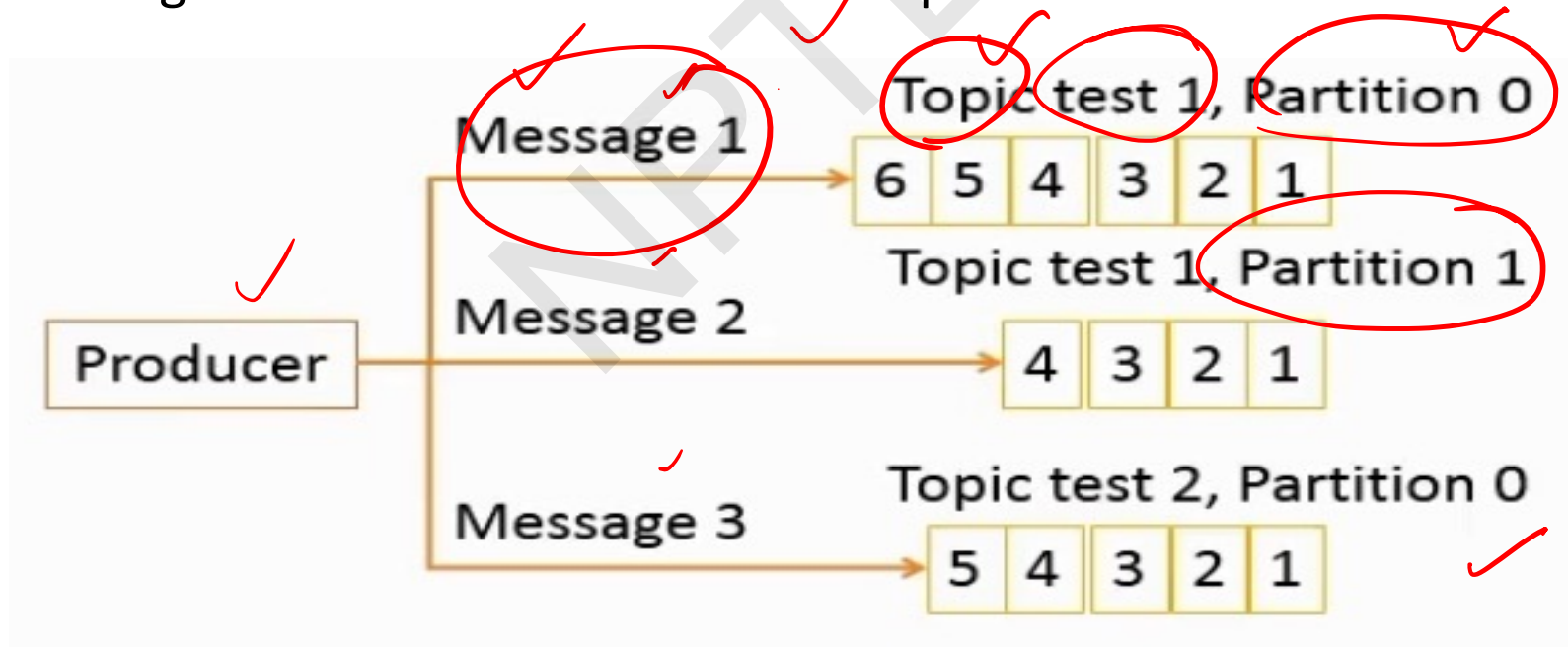
- Partitions can be distributed across the Kafka cluster.
- Each Kafka server may handle one or more partitions.
- A partition can be replicated across several servers for fault-tolerance.
- One server is marked as a leader for the partition and the others are marked as followers.
- The leader controls the read and write for the partition, whereas, the followers replicate the data.
- If a leader fails, one of the followers automatically become the leader.
- Zookeeper is used for the leader selection.



Producers

The producer is the creator of the message in Kafka.

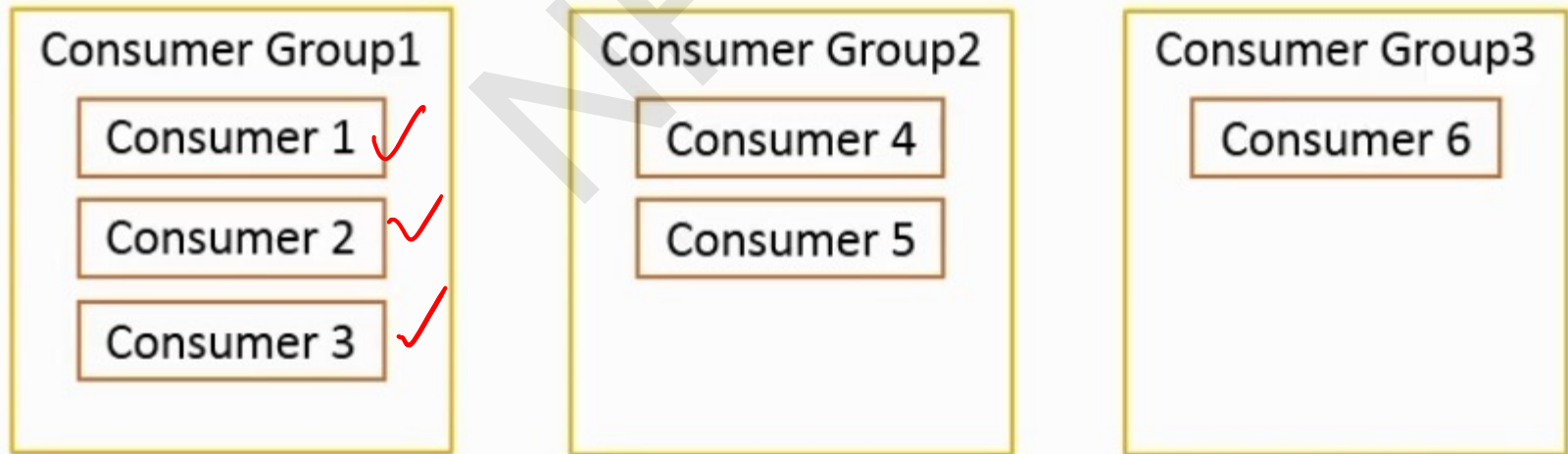
- The producers place the message to a particular topic.
- The producers also decide which partition to place the message into.
- Topics should already exist before a message is placed by the producer.
- Messages are added at one end of the partition.



Consumers

The consumer is the receiver of the message in Kafka.

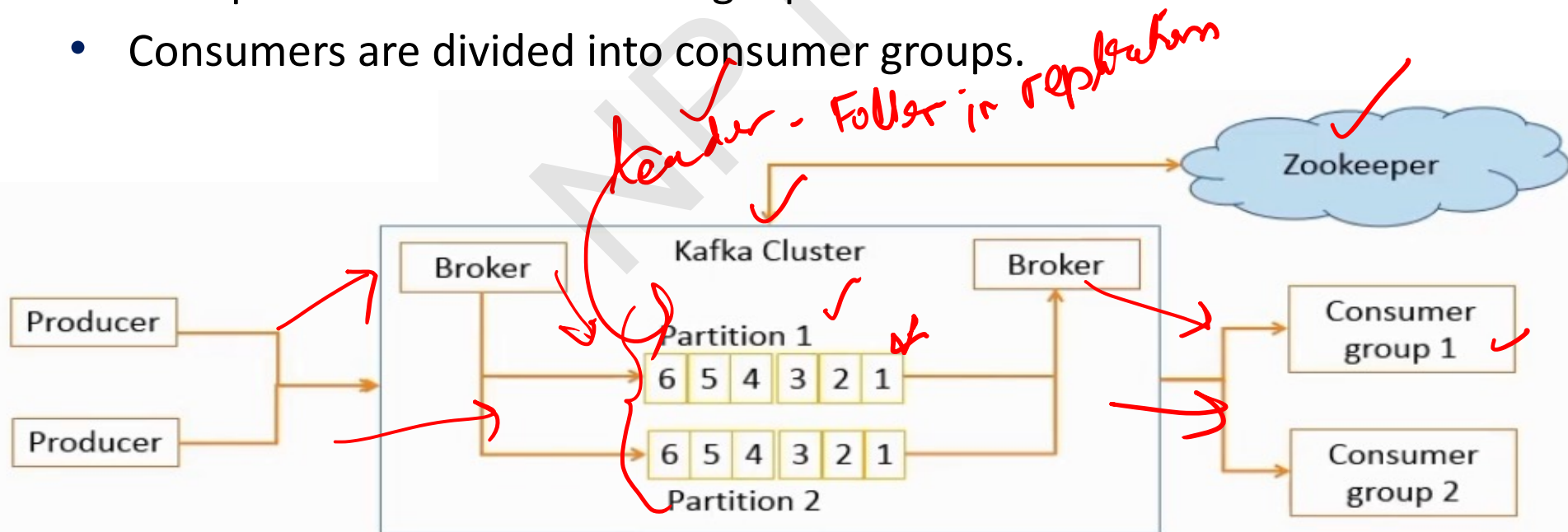
- Each consumer belongs to a consumer group.
- A consumer group may have one or more consumers.
- The consumers specify what topics they want to listen to.
- A message is sent to all the consumers in a consumer group.
- The consumer groups are used to control the messaging system.



Kafka Architecture

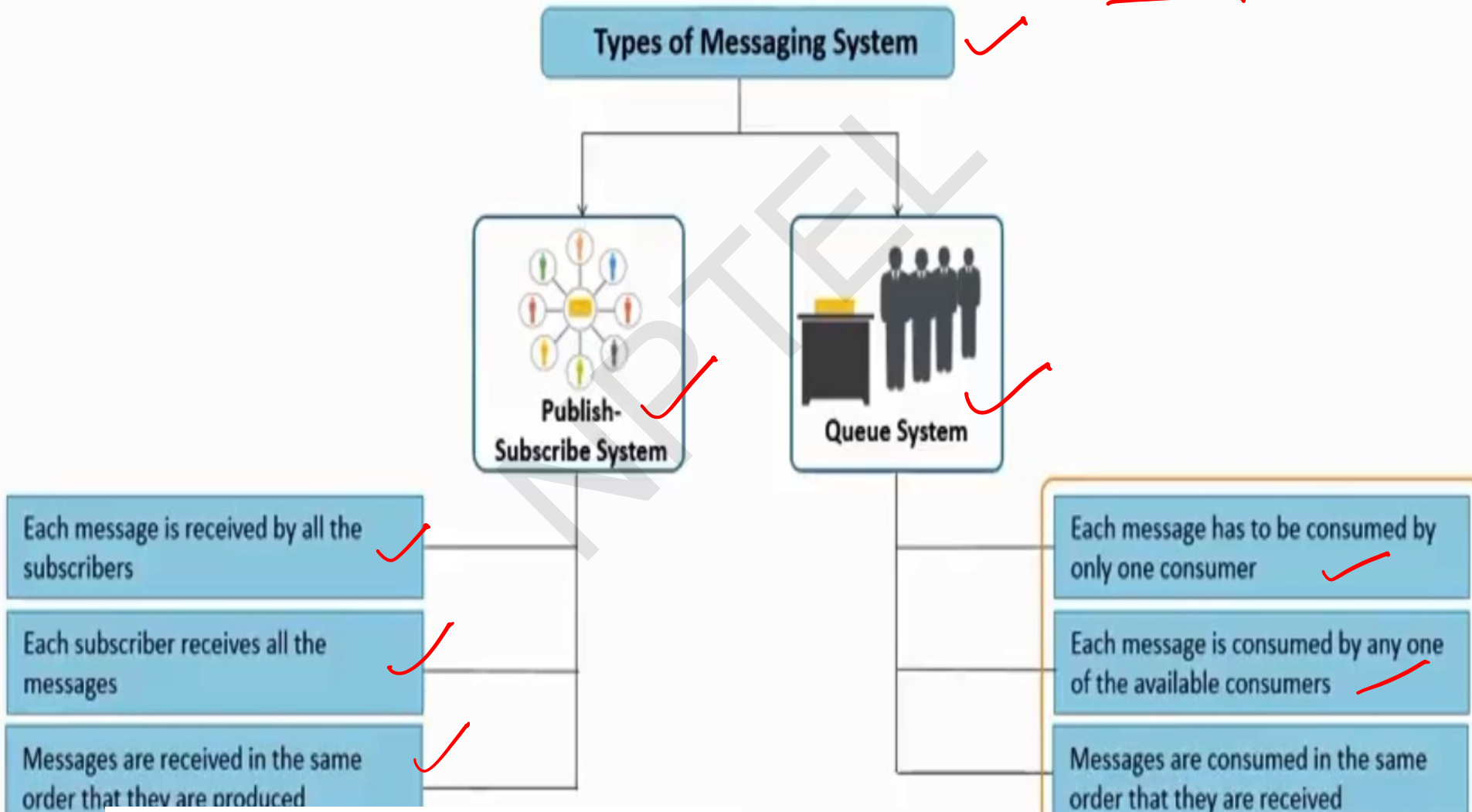
Kafka architecture consists of brokers that take messages from the producers and add to a partition of a topic. Brokers provide the messages to the consumers from the partitions.

- A topic is divided into multiple partitions. */ Shared / Parallelism*
- The messages are added to the partitions at one end and consumed in the same order.
- Each partition acts as a message queue.
- Consumers are divided into consumer groups.

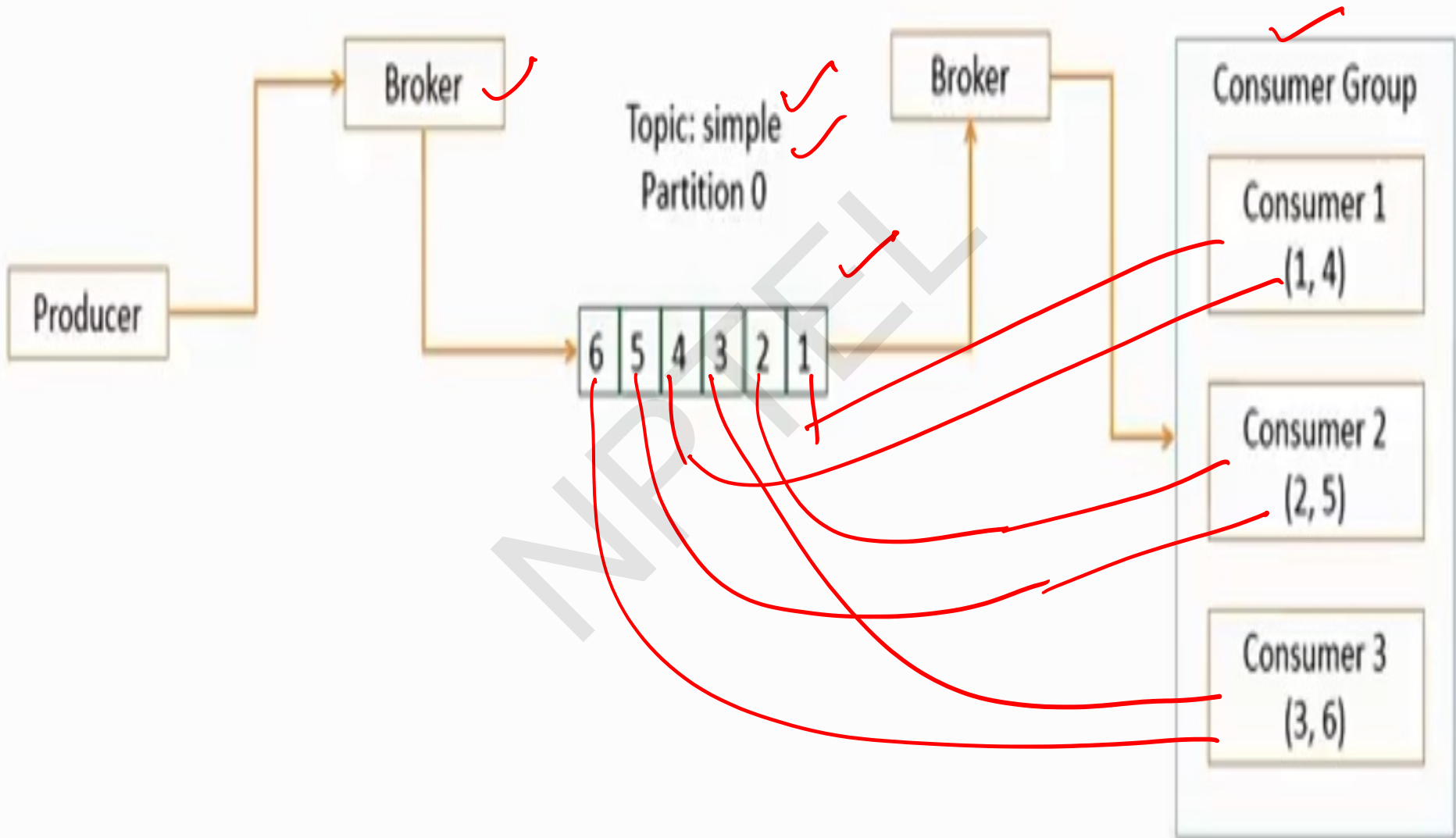


Types of Messaging Systems

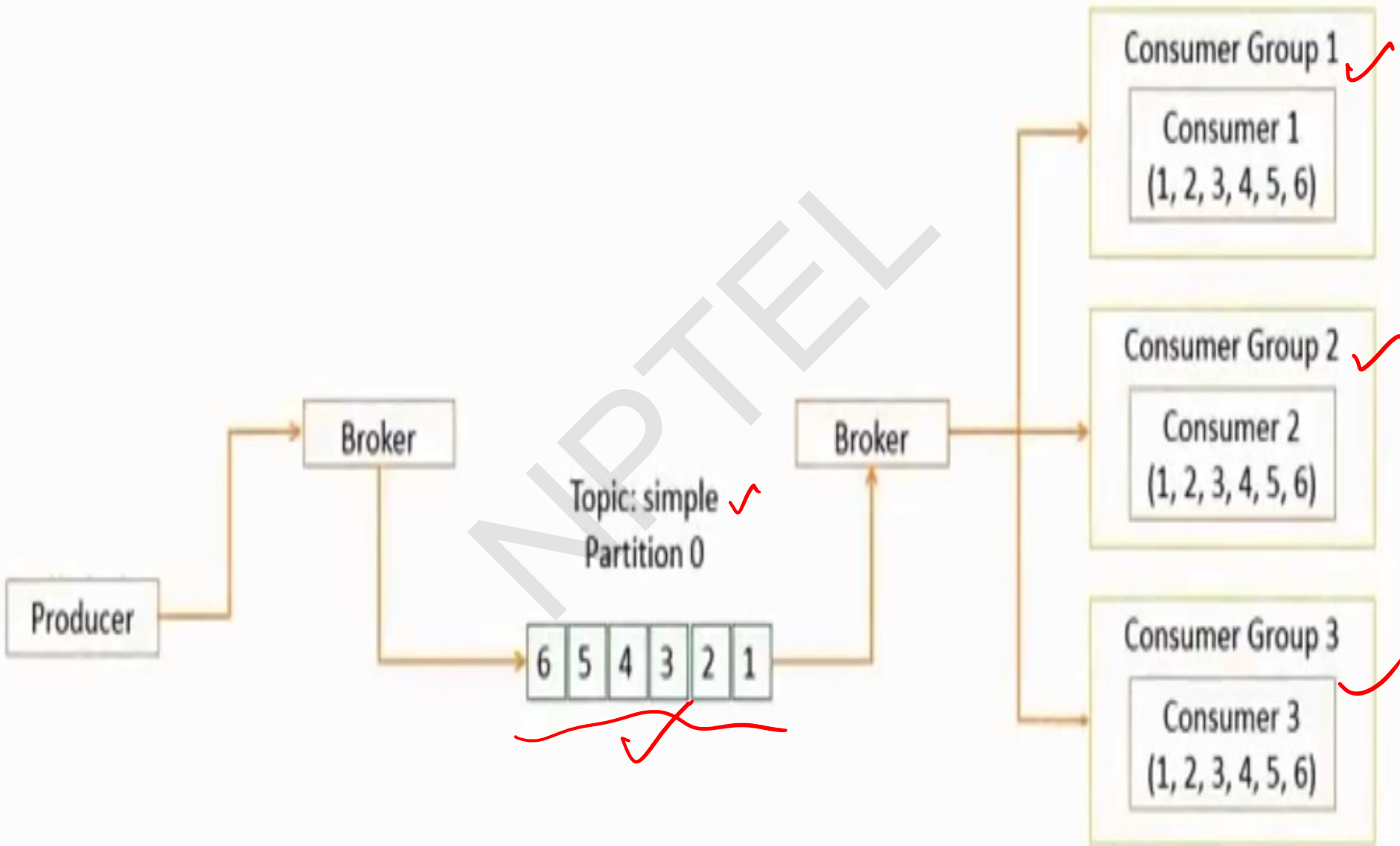
Kafka architecture supports the publish-subscribe and queue system.



Example: Queue System



Example: Publish-Subscribe System



Brokers

Brokers are the Kafka processes that process the messages in Kafka.

- Each machine in the cluster can run one broker.
- They coordinate among each other using Zookeeper.
- One broker acts as a leader for a partition and handles the delivery and persistence, where as, the others act as followers.

Kafka Guarantees

- Kafka guarantees the following:
 1. **Messages sent by a producer to a topic and a partition are appended in the same order** ✓
 2. **A consumer instance gets the messages in the same order as they are produced.** ✓
 3. **A topic with replication factor N , tolerates upto $N-1$ server failures.** ✓

Replication in Kafka

Kafka uses the primary-backup method of replication.

- One machine (one replica) is called a leader and is chosen as the primary; the remaining machines (replicas) are chosen as the followers and act as backups.
- The leader propagates the writes to the followers.
- The leader waits until the writes are completed on all the replicas.
- If a replica is down, it is skipped for the write until it comes back.
- If the leader fails, one of the followers will be chosen as the new leader; this mechanism can tolerate $n-1$ failures if the replication factor is 'n'

Persistence in Kafka

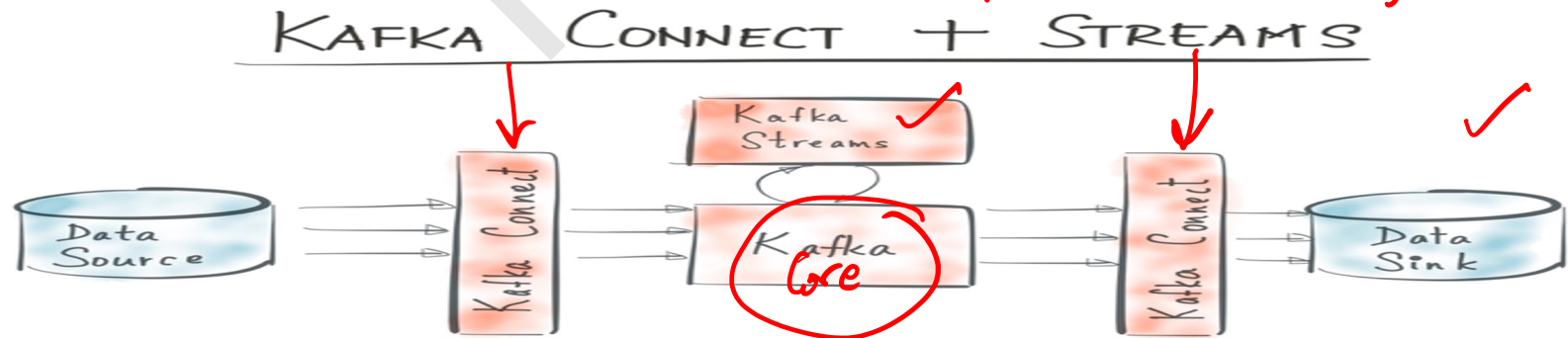
Kafka uses the Linux file system for persistence of messages

- Persistence ensures no messages are lost.
- Kafka relies on the file system page cache for fast reads and writes.
- All the data is immediately written to a file in file system.
- Messages are grouped as message sets for more efficient writes.
- Message sets can be compressed to reduce network bandwidth.
- A standardized binary message format is used among producers, brokers, and consumers to minimize data modification.

Apache Kafka: a Streaming Data Platform

➤ **Apache Kafka** is an open source streaming data platform (a new category of software!) with **3 major components**: —

1. **Kafka Core**: A **central hub** to **transport** and **store** event streams in real-time. *Hot data Analytics*
2. **Kafka Connect**: A **framework** to **import** event streams from other source data systems into Kafka and **export** event streams from **Kafka** to destination data systems.
3. **Kafka Streams**: A **Java library** to **process** event streams live as they occur. *Data Streams*



Further Learning

- **Kafka Streams code examples**
 - Apache Kafka <https://github.com/apache/kafka/tree/trunk/streams/examples/src/main/java/org/apache/kafka/streams/examples>
 - Confluent <https://github.com/confluentinc/examples/tree/master/kafka-streams>
- **Source Code** <https://github.com/apache/kafka/tree/trunk/streams>
- **Kafka Streams Java docs**
<http://docs.confluent.io/current/streams/javadocs/index.html>
- **First book on Kafka Streams (MEAP)**
 - Kafka Streams in Action <https://www.manning.com/books/kafka-streams-in-action>
- **Kafka Streams download**
 - Apache Kafka <https://kafka.apache.org/downloads>
 - Confluent Platform <http://www.confluent.io/download>

Conclusion

- Kafka is a high-performance, real-time messaging system. *data indeshum Kafka 105 - hot data Analysis*
- Kafka can be used as an external commit log for distributed systems.
- Kafka data model consists of messages and topics.
- Kafka architecture consists of brokers that take messages from the producers and add to a partition of a topics.
- Kafka architecture supports two types of messaging system called publish-subscribe and queue system.
- Brokers are the Kafka processes that process the messages in Kafka.

Introduction to Edge Data Center for IoT platform



Dr. Rajiv Misra, Professor

**Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in**

Preface

Content of this Lecture:

- Current demand of Data centers
- Why to move Data centers to Edge?
- In this lecture, we will discuss a brief introduction to Cloud Computing and also focus on the aspects *i.e.* Why Clouds, What is a Cloud, Whats new in todays Clouds and also distinguish Cloud Computing from the previous generation of distributed systems

Waves of Innovation: Cloud IoT Edge ML

Cloud

(the waves of innovation started with cloud)
Globally available, unlimited compute resources

IoT

(IoT-as-SaaS platform is key drivers of public cloud)
Harnessing signals from sensors and devices, managed centrally by the cloud

Edge

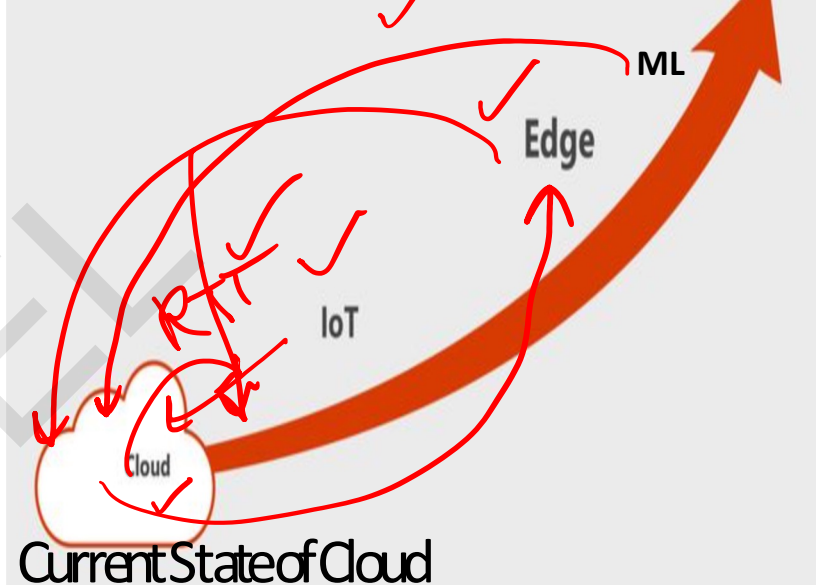
(IoT realize not everything needs to be in the cloud)
Intelligence offloaded from the cloud to IoT devices

ML

(rise of AI, ML models are trained in cloud are deployed at the edge to make inferencing for predictive analytics)

Breakthrough intelligence capabilities, in the cloud and on the edge

Waves of Innovation



- Highly centralized set of resources,
- Resembles Client/Server computing
- Compute is going beyond VMs as Containers becoming mainstream
- Storage is complemented by CDN is replicated and cached at edge locations
- Network stack is programmable SDN enabling hybrid scenarios

Edge Computing

- Edge computing makes the cloud truly distributed ✓
- Moves core cloud services closer to the origin of data ✓
- Edge Mimics public cloud platform capabilities ✓
- Delivers storage, compute, and network services locally.
- Reduces the latency by avoiding the roundtrip to the cloud
- Brings in **data sovereignty** by keeping data where it actually belongs, **savings on cloud and bandwidth** usages

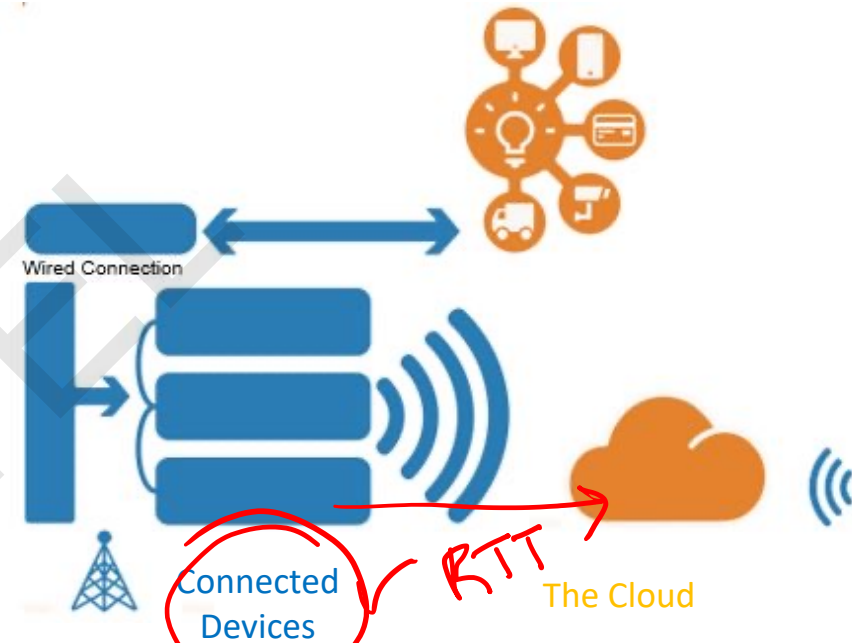
Functionality of Edge Computing for IOT

- Data Ingestion and M2M Brokers
- Object Storage ✓
- Functions as a Service ✓
- Containers ✓
- Distributed Computing ✓
- NoSQL/Time-Series Database
- Stream Processing ✓
- ML Models ✓

— Hot data analytics
— Cloud - Training/learning of ML models
— Edge - Inference

Cloud Data Center: Current Demand

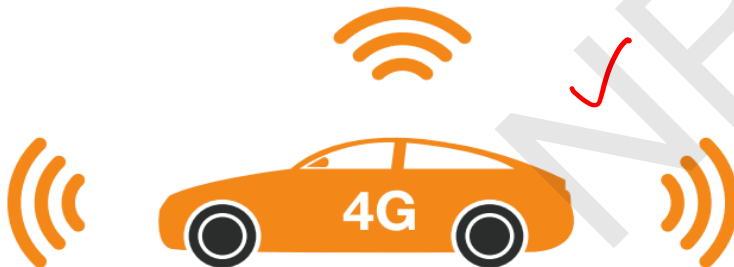
- In the next decade, we will continue to see skyrocketing growth in the number of IP-connected mobile and machine-to-machine (M2M) devices, which will handle significant amounts of IP traffic.
- Tomorrow's consumers will demand faster Wi-Fi service and application delivery from online providers. Also, some M2M devices, such as autonomous vehicles, will require real-time communications with local processing resources to guarantee safety.
- Today's IP networks cannot handle the high-speed data transmissions that tomorrow's connected devices will require. In a traditional IP architecture, data must often travel hundreds of miles over a network between end users or devices and cloud resources. This results in latency, or slow delivery of time-sensitive data.



Cloud Data Center: Current Demand

CURRENT: 4G

Only a few large centralized data centers

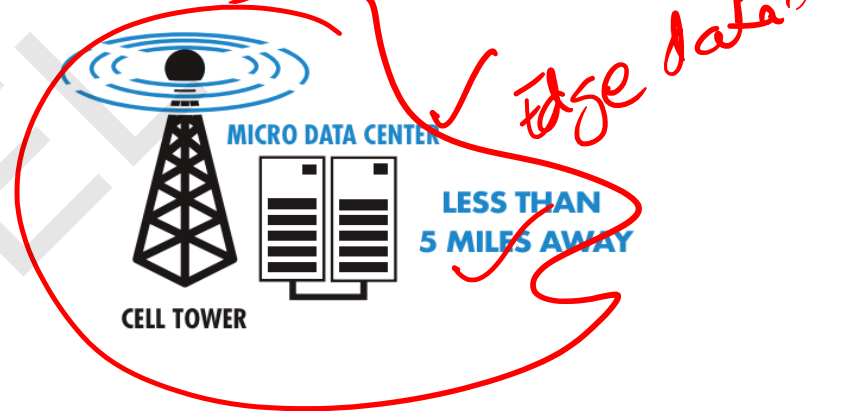


> 80 ms Latency

The vehicle moved over four feet by the time it received a response due to the large distance from the data center.

UPCOMING: 5G

Thousands of new micro data centers under cell towers



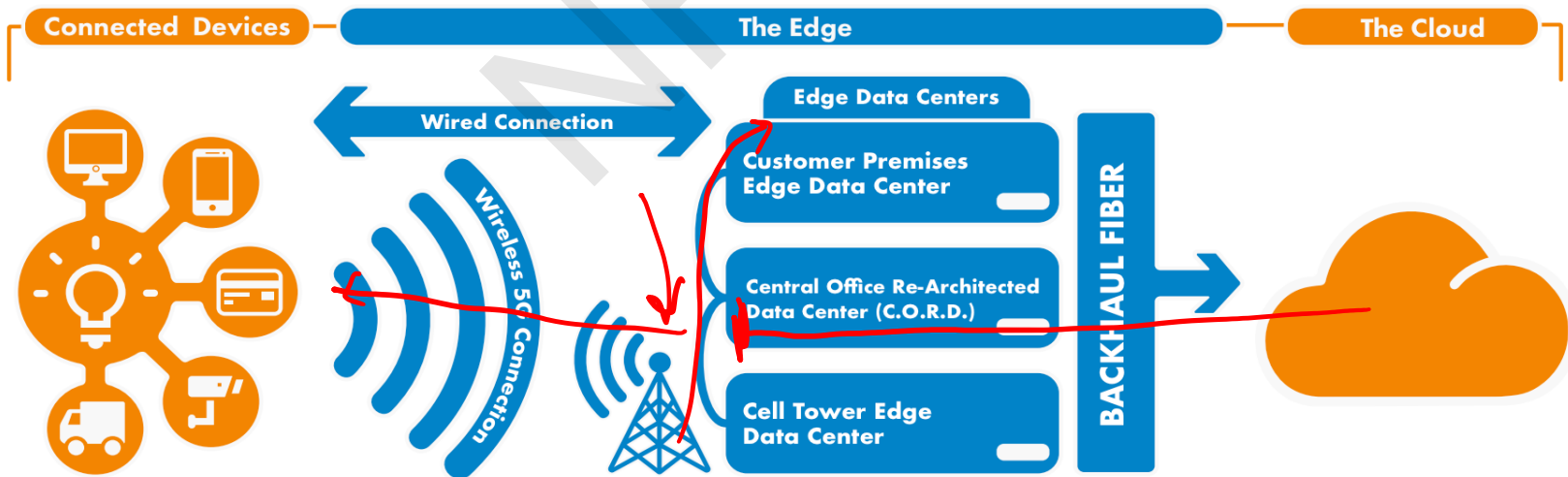
< 5 ms Latency

The vehicle moved less than four inches by the time it received a response, thanks to the close distance to the micro data center.

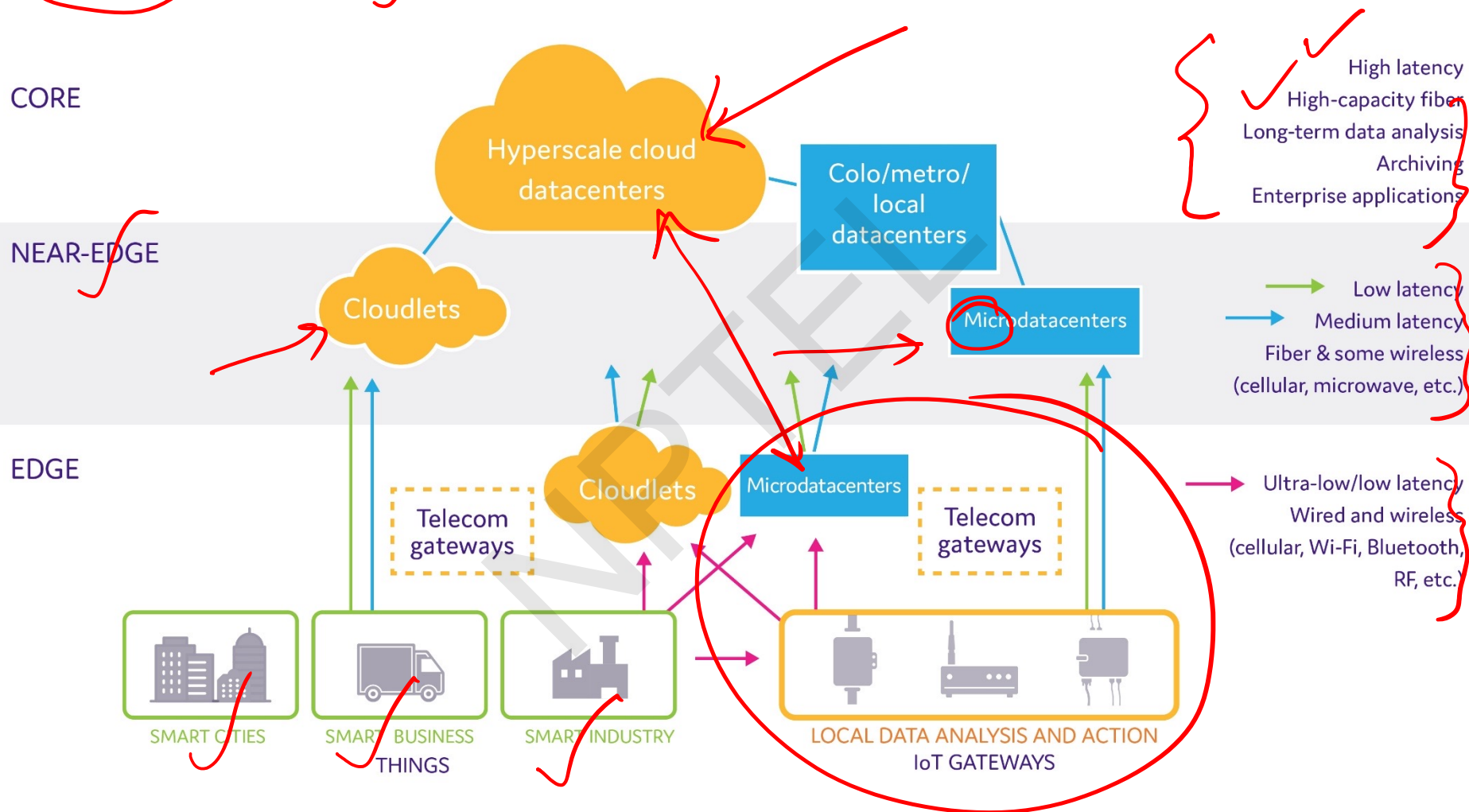
Edge Data Center: Solution

The solution to reducing latency lies in edge computing. By establishing IT deployments for cloud-based services in edge data centers in localized areas, we effectively bring IT resources closer to end users and devices. This helps us achieve efficient, high-speed delivery of applications and data. Edge data centers are typically located on the edge of a network, with connections back to a centralized cloud core.

Instead of bringing the users and devices to the data center, we bring the power of the data center to the users and devices. Edge computing relies on a distributed data center architecture, in which IT cloud servers housed in edge data centers are deployed on the outer edges of a network. By bringing IT resources closer to the end users and/or devices they serve, we can achieve high-speed, low-latency processing of applications and data.



Edge Data Center: Solution



Why Move Data Centers to the Edge?

There are four main benefits of moving data centers to the edge, which involve improvements to latency, bandwidth, operating costs, and security:

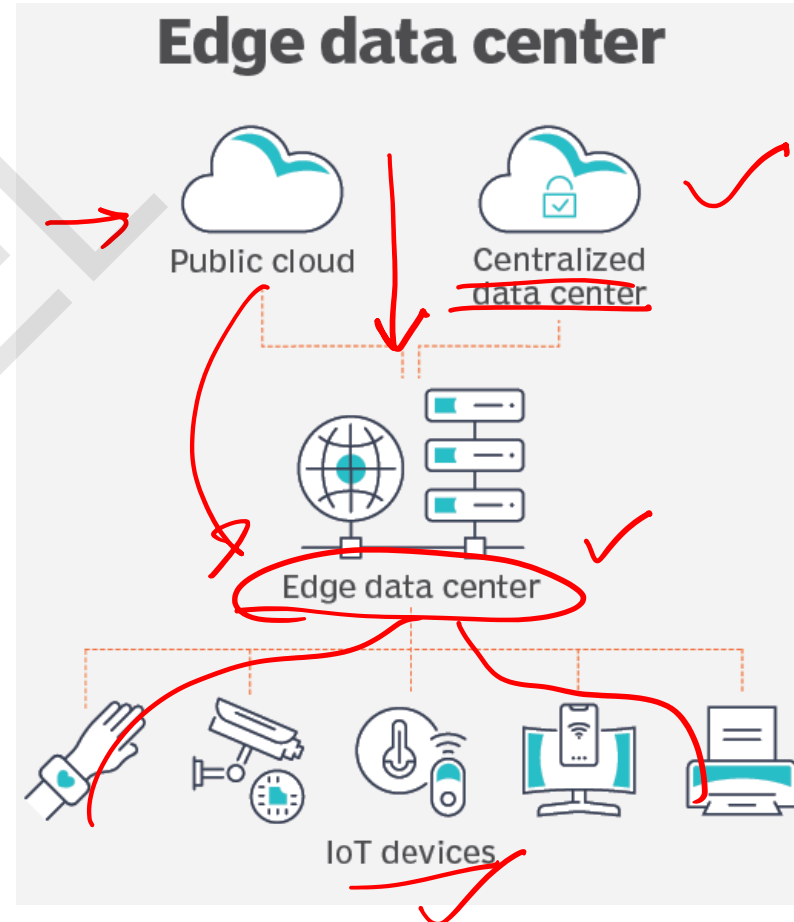
1. **Latency:** edge data centers facilitate lower latency, meaning much faster response times. Locating compute and storage functions closer to end users reduces the physical distance that data packets need to traverse, as well as the number of network "hops" involved, which lowers the probability of hitting a transmission path where data flow is impaired
2. **Bandwidth:** edge data centers process data locally, reducing the volume of traffic flowing to and from central servers. In turn, greater bandwidth across the user's broader network becomes available, which improves overall network performance
3. **Operating Cost:** because edge data centers reduce the volume of traffic flowing to and from central servers, they inherently reduce the cost of data transmission and routing, which is important for high-bandwidth applications. More specifically, edge data centers lessen the number of necessary high-cost circuits and interconnection hubs leading back to regional or cloud data centers, by moving compute and storage closer to end users
4. **Security:** edge data centers enhance security by: i) reducing the amount of sensitive data transmitted, ii) limiting the amount of data stored in any individual location, given their decentralized architecture, and iii) decreasing broader network vulnerabilities, because breaches can be ring-fenced to the portion of the network that they compromise

Edge Data Center: Introduction

Edge data centers are small data centers that are located close to the edge of a network. They provide the same devices found in traditional data centers, but are contained in a smaller footprint, closer to end users and devices.

Edge data centers can deliver cached content and cloud computing resources to these devices. The concept works off edge computing, which is a distributed IT architecture where client data is processed as close to the originating source as possible. Because the smaller data centers are positioned close to the end users, they are used to deliver fast services with minimal latency.

In an edge computing architecture, time-sensitive data may be processed at the point of origin by an intermediary server that is located in close geographical proximity to the client. The point is to provide the quickest content delivery to an end device that may need it, with as little latency as possible. Data that is less time-sensitive can be sent to a larger data center for historical analysis, big data analytics and long-term storage. Edge data centers work off of the same concept, except instead of just having one intermediary server in close geographical proximity to the client, it's a small data center -- that can be as small as a box. Even though it is not a new concept, edge data center is still a relatively new term.

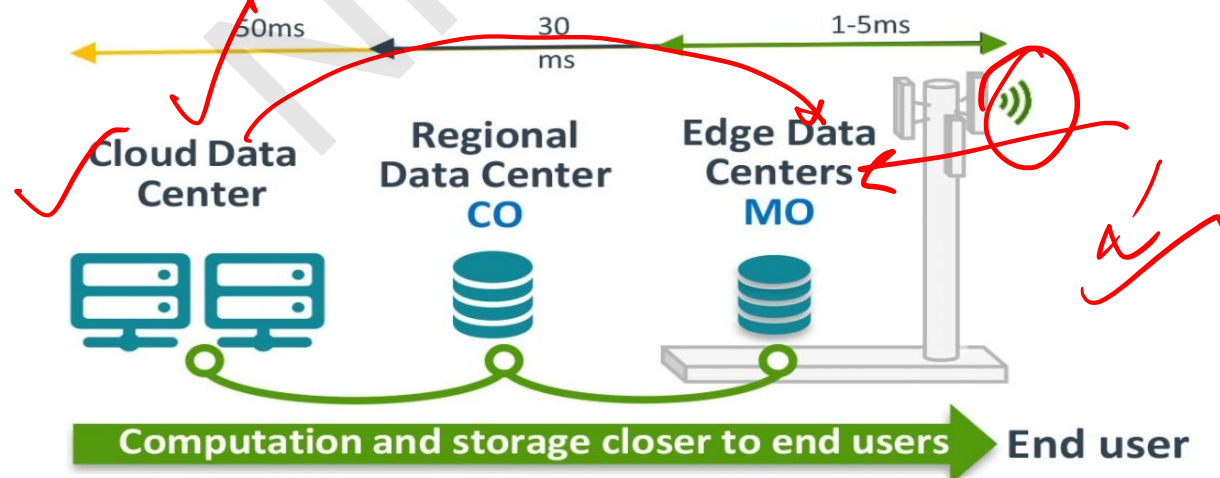


Edge Data Center: Introduction

The major benefit of an edge data center is the quick delivery of services with minimal latency, thanks to the use of edge caching. Latency may be a big issue for organizations that have to work with the internet of things (IoT), big data, cloud and streaming services.

Edge data centers can be used to provide high performance with low levels of latency to end users, making for a better user experience. Typically, edge data centers will connect to a larger, central data center or multiple other edge data centers.

Data is processed as close to the end user as possible, while less integral or time-centric data can be sent to a central data center for processing. This allows an organization to reduce latency.



Edge Data Center: Use Cases

1. **5G**: Where a decentralized cell network made of edge data centers can help provide low latency for 5G in use cases with high device density.
Telecommunications companies. With cell-tower edge data centers, telecom companies can get better proximity to end users by connecting mobile phones and wireless sensors.
2. **IoT**: Edge data centers can be used for data generated by IoT devices. An edge data center would be used if data generated by devices needs more processing but is also too time-sensitive to be sent to a centralized server.
3. **Healthcare**: Some medical equipment, such as those used for robotic surgeries, would require extremely low latency and network consistency, of which, edge data centers can provide.
4. **Autonomous vehicles**: Edge data centers can be used to help collect, process and share data between vehicles and other networks, which also relies on low latency. A network of edge data centers can be used to collect data for auto manufacturers and emergency response services.
5. **Smart factories**: Edge data centers can be used for machine Predictive maintenance, as well as predictive quality management. It can also be used for efficiency regarding robotics used within inventory management.

Scalable Computing at network edge

- Evolutionary changes that have occurred in **distributed edge and cloud computing** over the past 30 years, **driven by applications with variable workloads, low-latency usecase and large data sets**.
- Evolutionary changes in machine architecture, operating system platform, network connectivity, and application workload.
- **Edge computing** uses multiple computers at network edge to solve large-scale problems locally and over the Internet. Thus, **distributed edge computing becomes data-intensive and network-centric**.
- **The emergence of distributed edge computing clouds** instead demands high-throughput computing (HTC) systems built with distributed computing technologies.
- **High-throughput computing (HTC)** appearing as computer clusters, service-oriented, computational grids, peer-to-peer networks, Internet clouds and edge, and the future Internet of Things.

The Hype of Cloud: Forecasting

- Gartner in 2009 – Cloud computing revenue will soar faster than expected and will **exceed \$150 billion** by 2013. It will represent 19% of IT spending by 2015.
- IDC in 2009: “Spending on IT cloud services will triple in the next 5 years, reaching **\$42 billion**.”
- Forrester in 2010 – Cloud computing will go from **\$40.7 billion** in 2010 to **\$241 billion** in 2020.
- Companies and even federal/state governments using cloud computing now: **fbo.gov**

Many Cloud Providers

- AWS: Amazon Web Services
 - EC2: Elastic Compute Cloud
 - S3: Simple Storage Service
 - EBS: Elastic Block Storage
- Microsoft Azure
- Google Compute Engine/AppEngine
- Rightscale, Salesforce, EMC, Gigaspaces, 10gen, Datastax, Oracle, VMWare, Yahoo, Cloudera
- And 100s more...



service cloud

ORACLE



VirtualBox

10gen



Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- **Private clouds** are accessible only to company employees
- **Public clouds** provide service to any paying customer:
 - **Amazon S3 (Simple Storage Service):** store arbitrary datasets, pay per GB-month stored
 - **Amazon EC2 (Elastic Compute Cloud):** upload and run arbitrary OS images, pay per CPU hour used
 - **Google App Engine/Compute Engine:** develop applications within their App Engine framework, upload data that will be imported into their format, and run

Customers Save: Time and Money

- “With AWS, a new server can be up and running in **three minutes** compared to **seven and a half weeks** to deploy a server internally and a **64-node Linux cluster** can be online in five minutes (compared with three months internally.”
- “With Online Services, reduce the IT **operational costs** by roughly **30%** of spending”
- “A private cloud of virtual servers inside its datacenter has saved nearly **crores of rupees annually**, because the company can share computing power and storage resources across servers.”
- 100s of startups can harness large computing resources without buying their own machines.

What is a Cloud?

- Advances in virtualization make it possible to see the growth of Internet clouds **as a new computing paradigm**.
- i.e. dramatic differences between developing software for millions to use **as a service** versus distributing software to run on their PCs."

History:

- In 1984, John Gage Sun Microsystems gave the slogan, **"The network is the computer."**
- In 2008, David Patterson UC Berkeley said, **"The data center is the computer."**
- Recently, Rajkumar Buyya of Melbourne University simply said: **"The cloud is the computer."**
- Some people view **clouds as grids** or **clusters** with changes through virtualization, since clouds are anticipated to process huge data sets generated by the traditional Internet, social networks, and the future IoT.

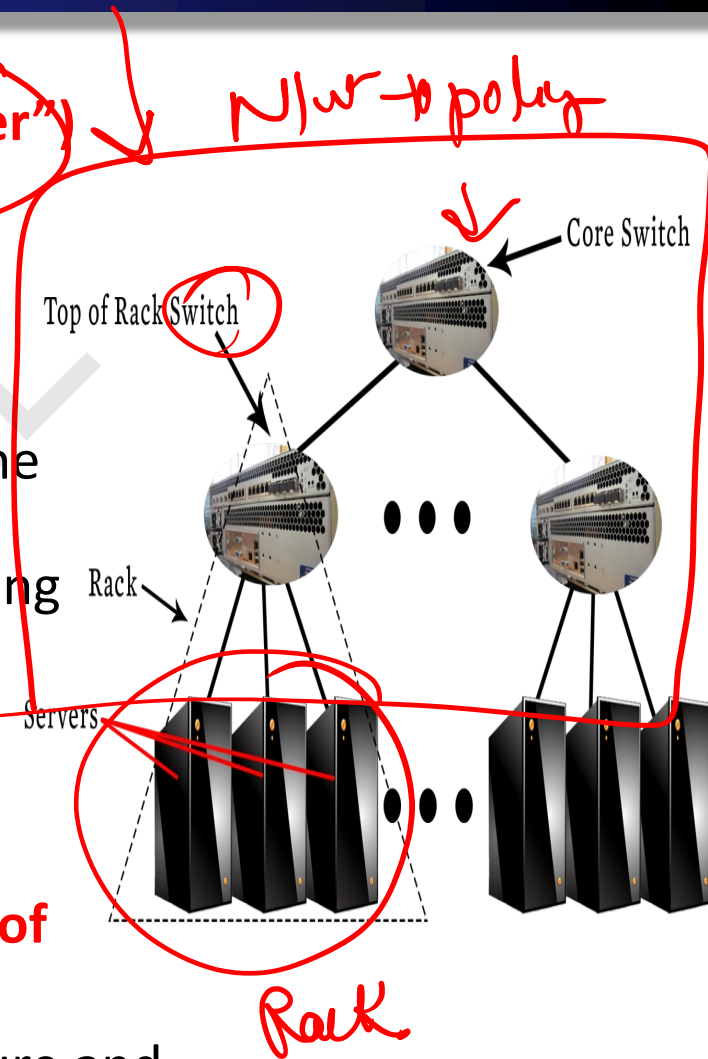
What is a Cloud?

- **A single-site cloud (as known as "Datacenter")** consists of

- Compute nodes (grouped into racks)
- Switches, connecting the racks
- A network topology, e.g., hierarchical
- Storage (backend) nodes connected to the network
- Front-end for submitting jobs and receiving client requests
- (Often called "three-tier architecture")
- Software Services

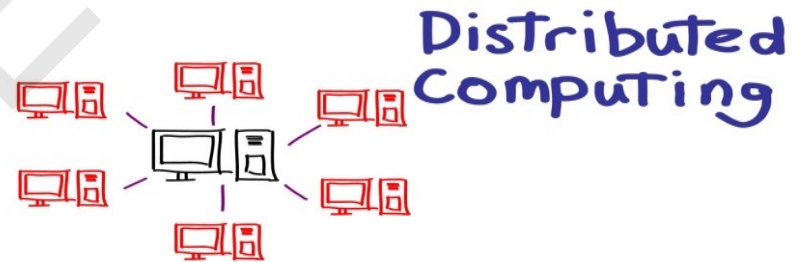
- **A geographically distributed cloud consists of**

- Multiple such sites
- Each site perhaps with a different structure and services

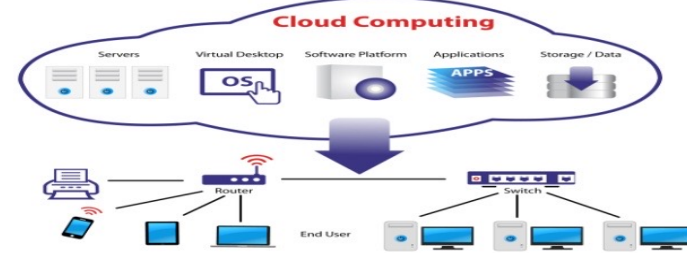


Computing Paradigm Distinctions

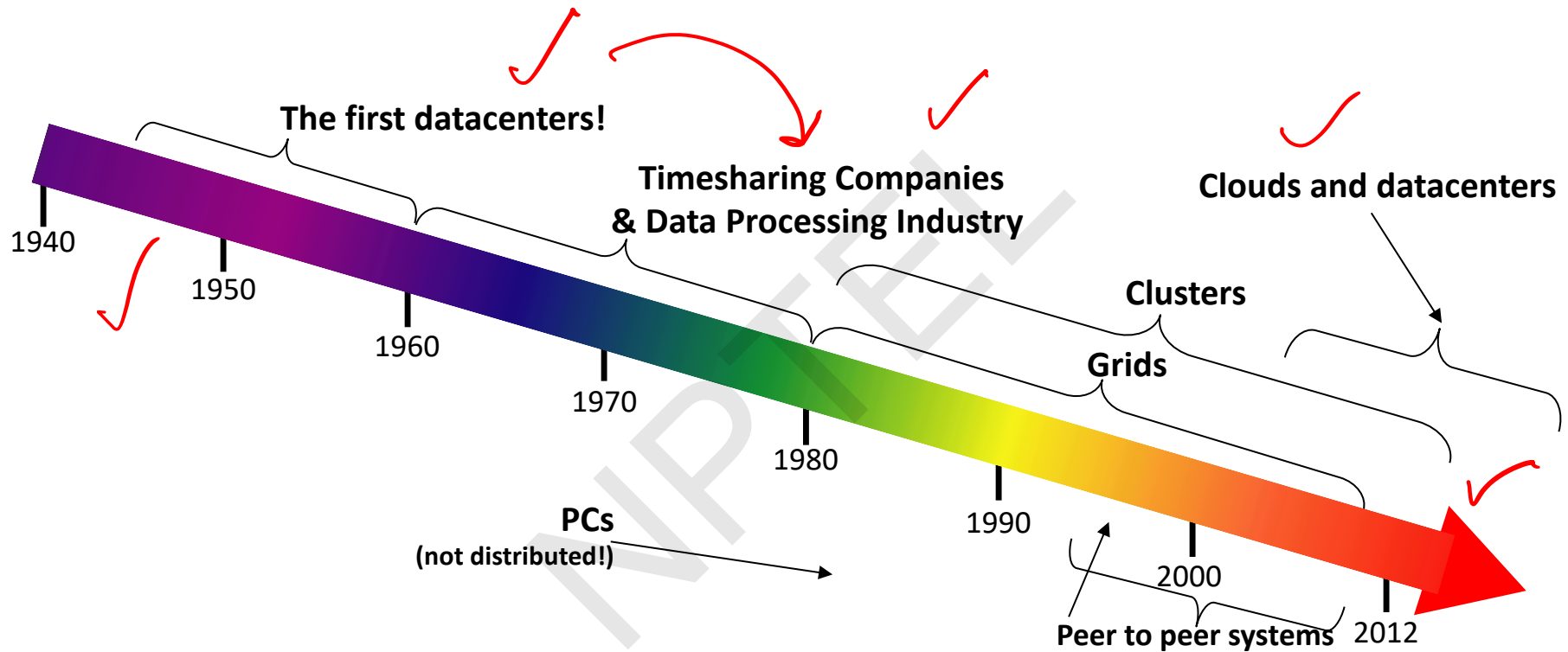
- Cloud computing overlaps with distributed computing.
- Distributed computing:** A *distributed system* consists of multiple autonomous computers, having its own memory, communicating through *message passing*.



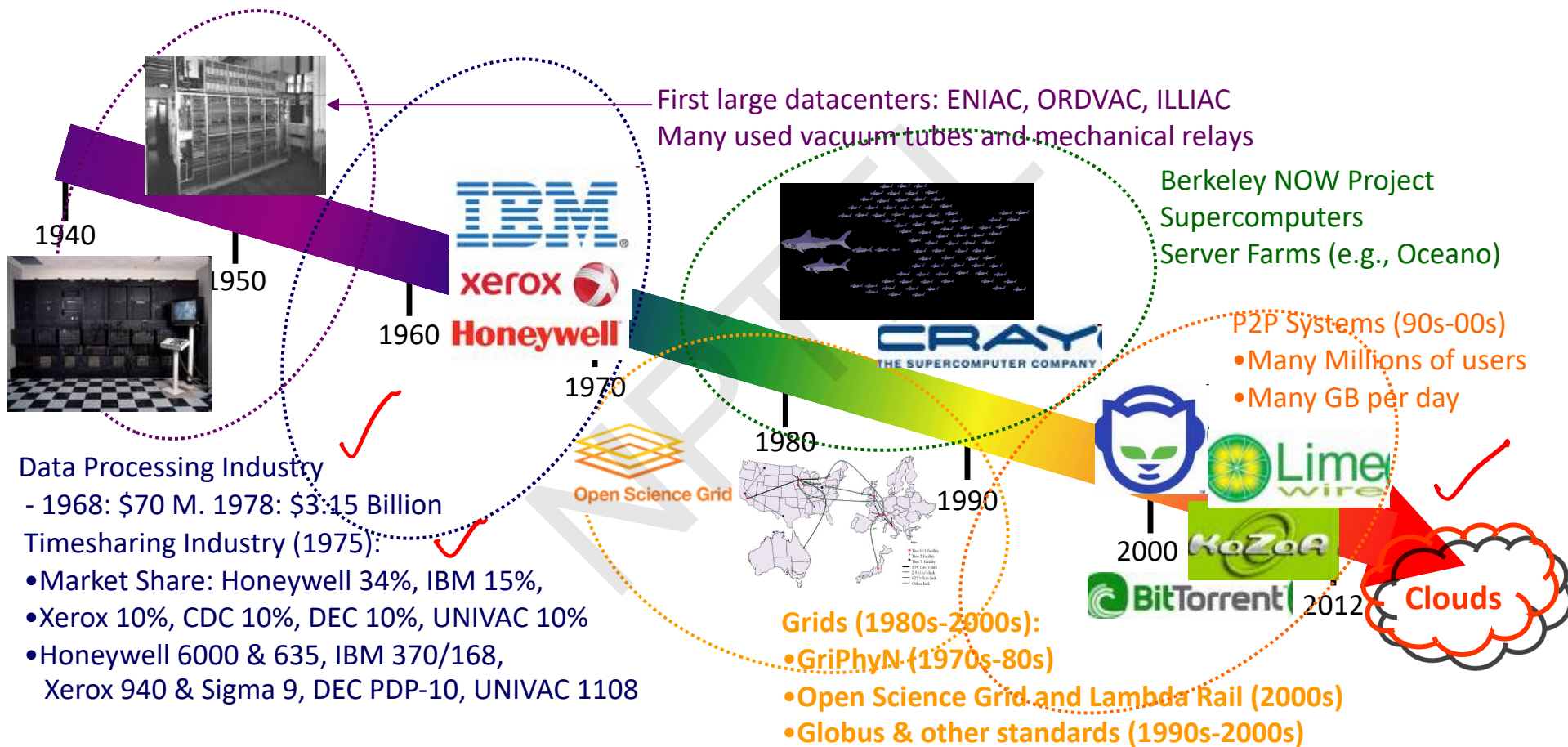
- Cloud computing:** Clouds can be built with physical or virtualized resources over large data centers that are distributed systems. Cloud computing is also considered to be a form of *utility computing or service computing*.



“A Cloudy History of Time”



“A Cloudy History of Time”



Scalable Computing Trends: Technology

- **Doubling Periods** – storage: 12 months, bandwidth: 9 months, and CPU compute capacity: 18 months (what law is this?)
- **Moore's law** indicates that processor speed doubles every 18 months.
- **Gilder's law** indicates that network bandwidth has doubled each year in the past.
- Then and Now
 - Bandwidth
 - 1985: mostly 56Kbps links nationwide
 - 2015: Tbps links widespread
 - Disk capacity
 - Today's PCs have TBs, far more than a 1990 supercomputer

The Trend toward Utility Computing

- Aiming towards autonomic operations that can be self-organized to support dynamic discovery. Major computing paradigms are composable with QoS and SLAs (service-level agreements).
- In 1965, MIT's Fernando Corbató of the Multics operating system envisioned a computer facility operating “like a power company or water company”.
- **Plug** your thin client into the computing Utility **and Play** Intensive Compute & Communicate Application
- **Utility computing** focuses on a business model in which customers receive computing resources from a paid service provider.
- All **grid/cloud platforms are regarded as utility service providers**.

Features of Today's Clouds

- I. **Massive scale:** Very large data centers, contain tens of thousands sometimes hundreds of thousands of servers and you can run your computation across as many servers as you want and as many servers as your application will scale.
- II. **On-demand access:** Pay-as-you-go, no upfront commitment.
 - And anyone can access it
- III. **Data-intensive Nature:** What was MBs has now become TBs, PBs and XBs.
 - Daily logs, forensics, Web data, etc.
- IV. **New Cloud Programming Paradigms:** MapReduce/Hadoop, NoSQL/Cassandra/MongoDB and many others.
 - Combination of one or more of these gives rise to novel and unsolved distributed computing problems in cloud computing.

I. Massive Scale

- **Facebook [GigaOm, 2012]**

- 30K in 2009 -> 60K in 2010 -> 180K in 2012



- **Microsoft [NYTimes, 2008]**

- 150K machines
- Growth rate of 10K per month
- 80K total running Bing
- In 2013, Microsoft Cosmos had 110K machines (4 sites)



- **Yahoo! [2009]:**

- 100K
- Split into clusters of 4000



- **AWS EC2 [Randy Bias, 2009]**

- 40K machines
- 8 cores/machine



- **eBay [2012]: 50K machines**



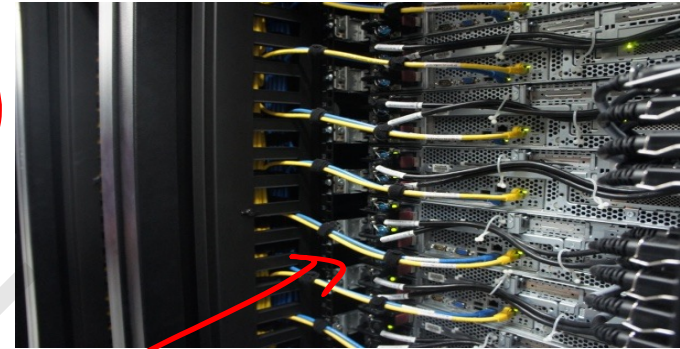
- **HP [2012]: 380K in 180 DCs**



- **Google: A lot**



What does a datacenter look like from inside?



Lots of Servers



Power and Energy



- $WUE = \frac{\text{Annual Water Usage}}{\text{IT Equipment Energy (L/kWh)}}$
– low is good
- $PUE = \frac{\text{Total facility Power}}{\text{IT Equipment Power}}$
– low is good (e.g., Google~1.11)

Off-site

On-site

Power usage efficiency

$$PUE = \frac{\text{Total facility power}}{\text{IT equipment power}} \approx 1.11 \text{ (low)}$$

Cooling



- Air sucked in
- Combined with purified water
- Moves cool air through system



II. On-demand access: *AAS Classification

- **On-demand:** renting vs. buying one. E.g.:
 - AWS Elastic Compute Cloud (EC2): a few cents to a few \$ per CPU hour
 - AWS Simple Storage Service (S3): a few cents per GB-month
- **HaaS: Hardware as a Service**
 - Get access to barebones hardware machines, do whatever you want with them, Ex: Your own cluster
 - Not always a good idea because of security risks
- **IaaS: Infrastructure as a Service**
 - Get access to flexible computing and storage infrastructure. **Virtualization** is one way of achieving this. subsume HaaS.
 - Ex: Amazon Web Services (AWS: EC2 and S3), OpenStack, Eucalyptus, Rightscale, Microsoft Azure, Google Cloud.

II. On-demand access: *AAS Classification

● **PaaS: Platform as a Service**

- Get access to flexible computing and storage infrastructure, coupled with a software platform (often tightly coupled)
- Ex: Google's AppEngine (Python, Java, Go)

● **SaaS: Software as a Service**

- Get access to software services, when you need them. subsume SOA (Service Oriented Architectures).
- Ex: Google docs, MS Office on demand

III. Data-intensive Computing





- **Computation-Intensive Computing**

- Example areas: MPI-based, High-performance computing, Grids
- Typically run on supercomputers (e.g., NCSA Blue Waters)

- **Data-Intensive**

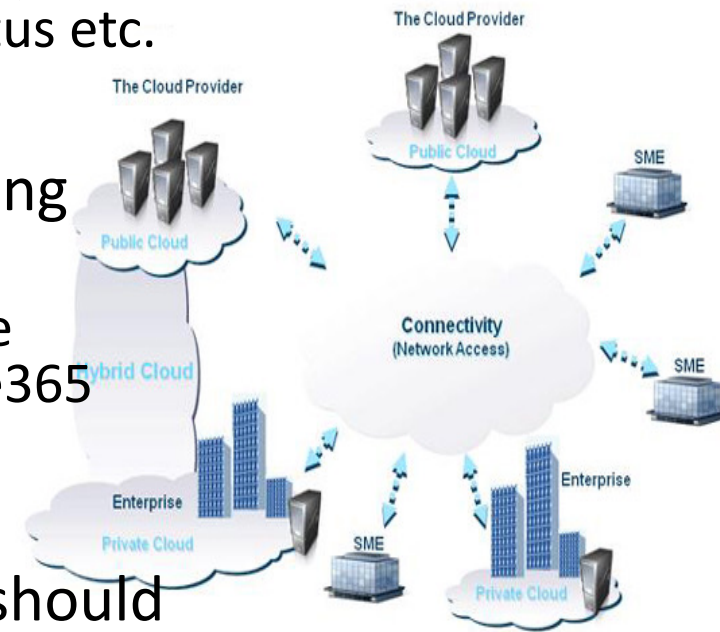
- Typically store data at datacenters
 - Use compute nodes nearby
 - Compute nodes run computation services
- In data-intensive computing, the **focus shifts from computation to the data:**
 - CPU utilization no longer the most important resource metric, instead I/O is (disk and/or network)

IV. New Cloud Programming Paradigms

- Easy to write and run highly parallel programs in new cloud programming paradigms:
 - **Google:** MapReduce and Sawzall 
 - **Amazon:** Elastic MapReduce service (pay-as-you-go) 
 - Google (MapReduce)
 - Indexing: a chain of 24 MapReduce jobs
 - ~200K jobs processing 50PB/month (in 2006)
 - **Yahoo!** (Hadoop + Pig) 
 - WebMap: a chain of several MapReduce jobs
 - 300 TB of data, 10K cores, many tens of hours (~2008)
 - **Facebook** (Hadoop + Hive) 
 - ~300TB total, adding 2TB/day (in 2008)
 - 3K jobs processing 55TB/day
 - NoSQL: MySQL is an industry standard, but Cassandra is 2400 times faster

Two Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- **Private clouds** are accessible only to company employees
- Example of popular vendors for creating private clouds are VMware, Microsoft Azure, Eucalyptus etc.
- **Public clouds** provide service to any paying customer
- Examples of large public cloud services include Amazon EC2, Google AppEngine, Gmail, Office365 and Dropbox etc.
- You're starting a new service/company: should you use a public cloud or purchase your own private cloud?



Single site Cloud: to Outsource or Own?

- Medium-sized organization: wishes to run a service for M months
 - Service requires 128 servers (1024 cores) and 524 TB
- **Outsource** (e.g., via AWS): *monthly* cost
 - S3 costs: \$0.12 per GB month. EC2 costs: \$0.10 per CPU hour (costs from 2009) Storage = $\$0.12 \times 524 \times 1000 \sim \62 K
 - Total = Storage + CPUs = $\$62 \text{ K} + \$0.10 \times 1024 \times 24 \times 30 \sim \136 K

- **Own**: *monthly* cost
 - Storage $\sim \$349 \text{ K} / M$ Total $\sim \$1555 \text{ K} / M + 7.5 \text{ K}$ (includes 1 sysadmin / 100 nodes)
 - using 0.45:0.4:0.15 split for hardware:power: network and 3 year lifetime of hardware

○ Breakeven analysis: **more preferable to own if:**

- $\$349 \text{ K} / M < \62 K (storage)
- $\$1555 \text{ K} / M + 7.5 \text{ K} < \136 K (overall)

|| Breakeven points

- $M > 5.55$ months (storage)
- $M > 12$ months (overall)

-Startups use clouds a lot
-Cloud providers benefit monetarily most from storage

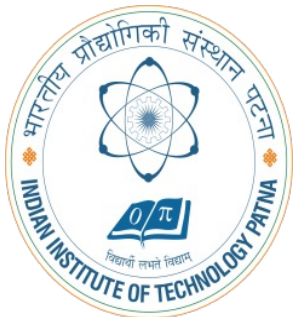
Conclusion

- Limitations of current cloud data center.
- Understanding the concept of edge data center.
- Clouds build on many previous generations of distributed systems
- Characteristics of cloud computing problem
 - **Scale, On-demand access, data-intensive, new programming**



THANK YOU!

Design of Key-Value Stores for IoT Edge Storage



Dr. Rajiv Misra

Associate Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

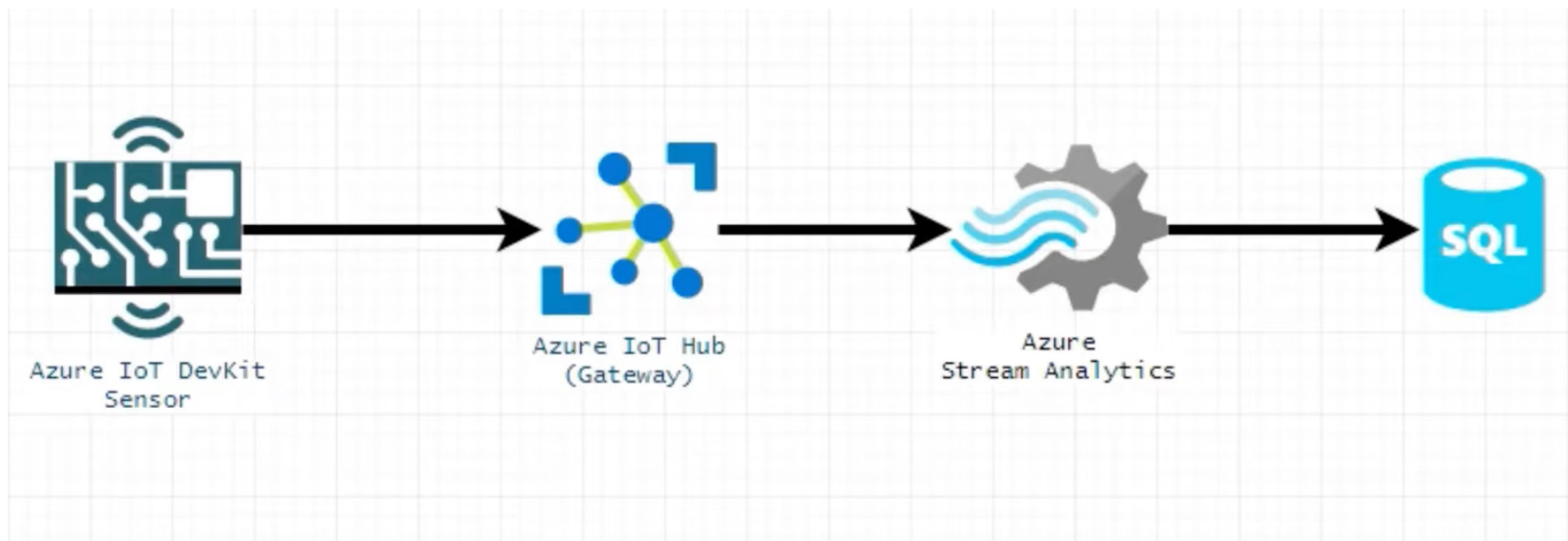
Preface

Content of this Lecture:

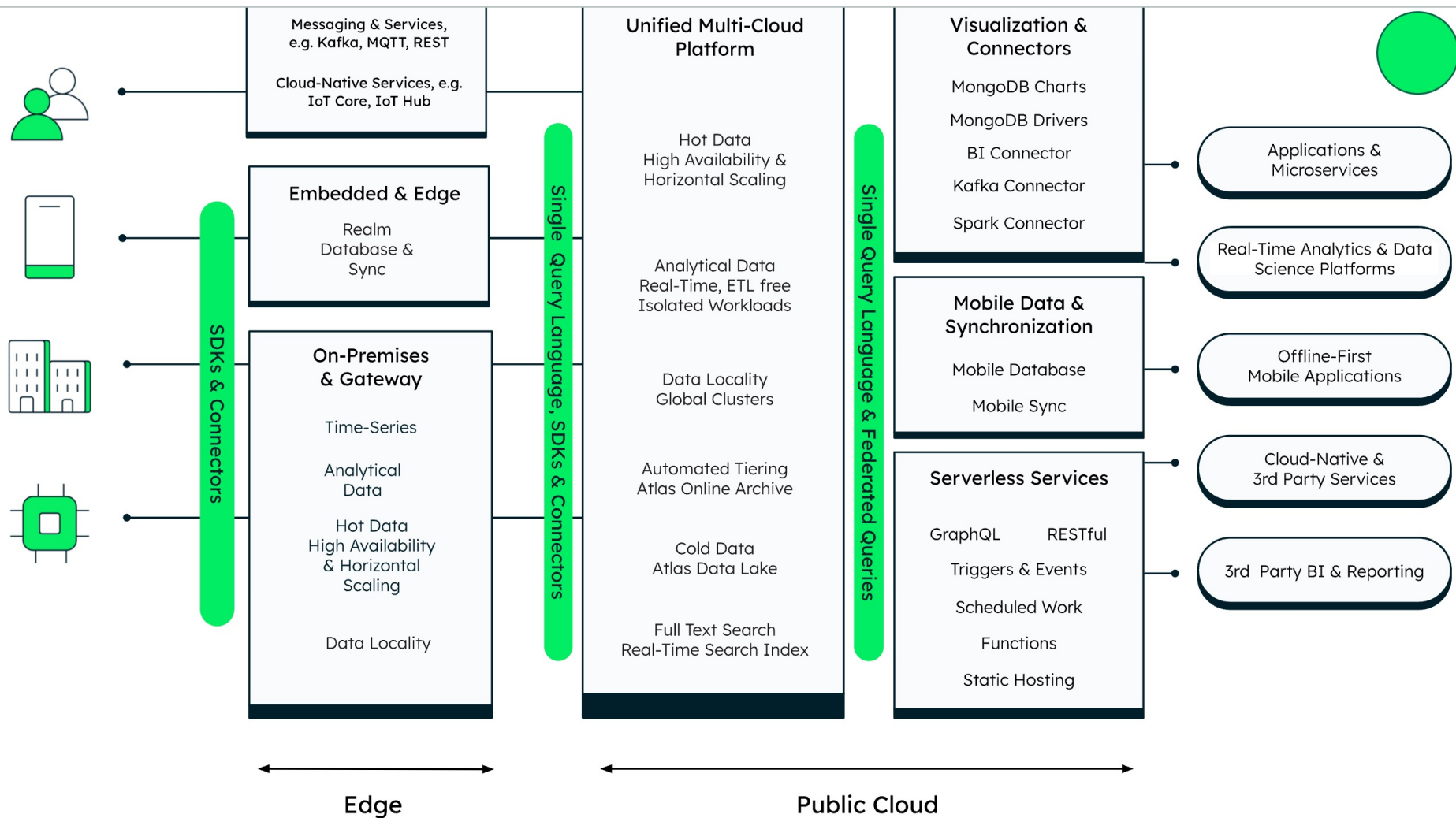
- Design of datastore for IoT applications.
- In this lecture, we will discuss the design and insight of **Key-value/NoSQL stores** for today's Edge storage systems.
- We will also discuss one of the most popular cloud storage system i.e. **Apache Cassandra** and different consistency solutions.

IoT Edge: Data Flow

Initially the data collected from the azure IOT sensor is passed through the IOT-Edge gateway and then we need to pull the data from IOT-Edge using stream analytics and then stream that data then to from the IOT Edge to the data-store/database



IoT Edge: Data Flow



IoT Edge: Databases

The most popular databases for IoT apps are InfluxDB, CrateDB, Riak TS, MongoDB, RethinkDB, SQLite, **Apache Cassandra**.

To select the right storage for Time Series and IoT domain use case, it depends upon the data-access methods, you may require the following database:

- **Hot database:**

These are typically used for data that is frequently being queried or updated. They are often a good choice for storing data as they provide read and write capabilities with little latency at the lowest cost. When choosing a hot database you can consider the following features — flexibility in data formats, querying abilities, messaging/ queueing capability, and tiered memory models.

- **Cold Database:**

They store information in their original state with little to no changes made thereafter. In contrast with real-time data collection, storing huge volumes

IoT Edge: Databases

NoSQL with Built-In Sorting

BigTable, HBase, Cassandra, DynamoDB, Accumulo are often used to store time-series data.

Strong Sides: *Extremely well scaled for writes. Performing the basic level of analytics extremely efficient.*

Weak Sides: *All other kinds of analytics are not supported and not efficient*

NoSQL Purpose-Built Time Series DB

There are engines that have been designed from the ground up as Time Series databases. In the majority of cases, they are NoSQL.

NewSQL In-Memory Databases

The in-memory nature of SQL databases increases their ability to handle fast data ingestion. SQL interface enriched by the time buckets normalization support

Strong Sides: *Provide the reach analytics capabilities.*

Weak Sides: *The scalability for writes and reads are usually limited or is very expensive*

Cloud Time-Series Platforms

Azure and AWS released recently their time series data services/platforms:

Azure Time Series Insights

Amazon Timestream

The platforms cover many aspects of the time series data storing, visualizing, and really reach capabilities in querying. They have built-in separation of data between hot, warm, and cold storage to make the data storing and retrieval well balances from the cost of ownership perspective.

IoT Edge Database: Example

As a continuation of the series of lecture about IoT Data Analytics, let's use the Fitness Tracker use case which represents well a typical IoT use case. A dataset (as it is also described here and here) consists of a set of observation, and each observation contains:

- A *metric name* generating by a sensor/edge, i.e.: heart rate, elevation, steps
- A *metric value* generated by the sensor bound to the point in time, i.e.: (2020-11-12 17:14:07, 71bpm), (2020-11-12 17:14:32, 93bpm), etc
- *Tags or Context description* in which a given sensor is generating data, i.e.: device model, geography location, user, activity type, etc.

IoT Edge Database: Functional Requirement

Basic Level: Simple Data Retrieval

- **Random data access:** for the particular point in time return the proper metric value
- **Small range scans:** for the particular time range (reasonably small, within minutes or hours depending on the frequency of data generation) return the sequential metric values (i.e.: to draw a standard chart on it)

IoT Edge Database: Functional Requirement

Middle Level: Time Window Normalization

The measurement events usually supposed to be triggered on a predefined recurrence basis, but there are always deviations in data points timing. That is why it is highly desirable to have capabilities around building predefined time windows to normalize the time series data.

To the mid-level capacities it is worth to add more sophisticated diagnostic analytics/ad-hoc queries:

- **Flexible Filtering:** filter data points based on predicate on tags/context attributes, *i.e.: filtering data points by some region, user, or activity type*
- **Flexible Aggregations:** grouping and aggregations on tags/context attributes or their combinations, *i.e.: max hearth rate by region by activity type.*

IoT Edge Database: Functional Requirement

Advance Level: Sequential Row Pattern Matching

The most advanced level would include checking if the **sequence of events matches** the particular **pattern** to perform introspection and advanced diagnosis:

- Did similar patterns of measurements precede specific events?;
- What measurements might indicate the cause of some event, such as a failure?

IoT Edge Database: Non Functional Requirement

Besides the functional requirements, it's really crucial to consider non-functional requirements which often are the main drivers for the selection:

- **Scalable storage:** ability to handle big data volumes
- **Scalable writes:** the ability to handle a big amount of simultaneous writes. This is closely related to the real-time data access — the ability to have the minimum possible lag between when the data point is generated and when it's available for reading.
- **Scalable reads:** the ability to handle a big amount of simultaneous reads
- **High Maturity:** presence on the market and community support.

The Key-value Abstraction

- (Business) Key → Value
- (flipkart.com) item number → information about it
- (easemytrip.com) Flight number → information about flight, e.g., availability
- (twitter.com) tweet id → information about tweet
- (mybank.com) Account number → information about it

The Key-value Abstraction (2)

- **It's a dictionary datastructure.**
 - Insert, lookup, and delete by key
 - Example: hash table, binary tree
- But distributed.
- Seems familiar? Remember **Distributed Hash tables (DHT) in P2P systems?**
- Key-value stores reuse many techniques from DHTs.

Is it a kind of database ?

- Yes, kind of
- **Relational Database Management Systems (RDBMSs)** have been around for ages
- **MySQL** is the most popular among them
- Data stored in tables
- Schema-based, i.e., structured tables
- Each row (data item) in a table has a primary key that is unique within that table
- Queried using **SQL (Structured Query Language)**
- Supports joins

Relational Database Example

users table

user_id	name	zipcode	blog_url	blog_id
110	Smith	98765	smith.com	11
331	Antony	54321	antony.in	12
767	John	75676	john.net	13

Example SQL queries

1. **SELECT** zipcode
FROM users
WHERE name = "John"
2. **SELECT** url
FROM blog
WHERE id = 11
3. **SELECT** users.zipcode,
blog.num_posts
FROM users **JOIN** blog
ON users.blog_url =
blog.url



Primary keys



blog table

Id	url	last_updated	num_posts
11	smith.com	9/7/17	991
13	john.net	4/2/18	57
12	antony.in	15/6/16	1090



Foreign keys

Mismatch with today's workloads

- **Data: Large and unstructured:** Difficult to come out with schemas where the data can fit
- **Lots of random reads and writes:** Coming from millions of clients.
- **Sometimes write-heavy:** Lot more writes compare to read
- **Foreign keys rarely needed**
- **Joins infrequent**

Needs of Today's Workloads

- Speed
- Avoid Single point of Failure (SPoF)
- Low TCO (Total cost of operation and Total cost of ownership)
- Fewer system administrators
- Incremental Scalability
- Scale out, not scale up

Scale out, not Scale up

- **Scale up = grow your cluster capacity by replacing with more powerful machines**
 - Traditional approach
 - Not cost-effective, as you're buying above the sweet spot on the price curve
 - And you need to replace machines often
- **Scale out = incrementally grow your cluster capacity by adding more COTS machines (Components Off the Shelf)**
 - Cheaper
 - Over a long duration, phase in a few newer (faster) machines as you phase out a few older machines
 - Used by most companies who run datacenters and clouds today

Key-value/NoSQL Data Model

- NoSQL = “Not Only SQL”
- Necessary API operations: **get(key)** and **put(key, value)**
 - And some extended operations, e.g., “CQL” in Cassandra key-value store
- **Tables**
 - “Column families” in Cassandra, “Table” in HBase, “Collection” in MongoDB
 - Like RDBMS tables, but ...
 - May be unstructured: May not have schemas
 - Some columns may be missing from some rows
 - Don’t always support joins or have foreign keys
 - Can have index tables, just like RDBMSs

Key-value/NoSQL Data Model

- Unstructured
- No schema imposed
- Columns missing from some Rows
- No foreign keys, joins may not be supported

Key ↓

Value

users table

user_id	name	zipcode	blog_url
110	Smith	98765	smith.com
331	Antony		antony.in
767		75676	john.net

Value

blog table

Id	url	last_updated	num_posts
11	smith.com	9/7/17	991
13	john.net		57
12	antony.in	15/6/16	

Column-Oriented Storage

NoSQL systems often use column-oriented storage

- RDBMSs store an entire row together (on disk or at a server)
- NoSQL systems typically store a column together (or a group of columns).
 - Entries within a column are indexed and easy to locate, given a key (and vice-versa)
- **Why useful?**
 - Range searches within a column are fast since you don't need to fetch the entire database
 - E.g., Get me all the blog_ids from the blog table that were updated within the past month
 - Search in the the last_updated column, fetch corresponding blog_id column
 - Don't need to fetch the other columns

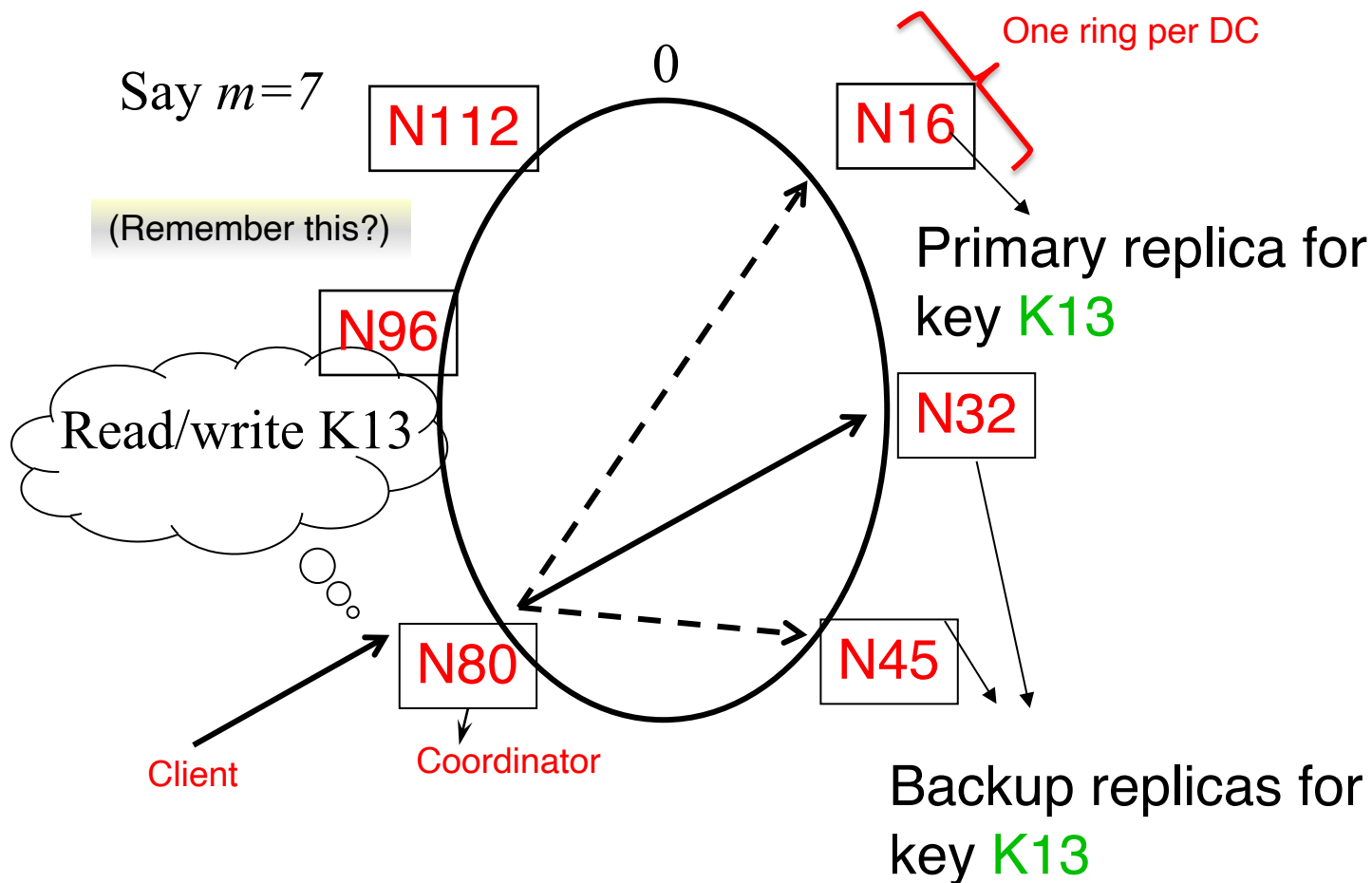
Design of Apache Cassandra

Cassandra

- **A distributed key-value store**
- Intended to run in a datacenter (and also across DCs)
- Originally designed at Facebook
- Open-sourced later, today an Apache project
- Some of the companies that use Cassandra in their production clusters
 - Blue chip companies: IBM, Adobe, HP, eBay, Ericsson
 - Newer companies: Twitter
 - Nonprofit companies: PBS Kids
 - Netflix: uses Cassandra to keep track of positions in the video.

Inside Cassandra: Key -> Server Mapping

- How do you decide which server(s) a key-value resides on?



Cassandra uses a Ring-based DHT but without
finger tables or routing
Key \rightarrow server mapping is the "Partitioner"

Data Placement Strategies

- **Replication Strategy:**

1. *SimpleStrategy*
2. *NetworkTopologyStrategy*

1. SimpleStrategy: uses the Partitioner, of which there are two kinds

1. ***RandomPartitioner***: Chord-like hash partitioning
2. ***ByteOrderedPartitioner***: Assigns ranges of keys to servers.

- Easier for ***range queries*** (e.g., Get me all twitter users starting with [a-b])

2. NetworkTopologyStrategy: for multi-DC deployments

- Two replicas per DC
- Three replicas per DC
- Per DC
 - First replica placed according to Partitioner
 - Then go clockwise around ring until you hit a different rack

Snitches

- **Maps:** IPs to racks and DCs. Configured in `cassandra.yaml` config file
- **Some options:**
 - **SimpleSnitch:** Unaware of Topology (Rack-unaware)
 - **RackInferring:** Assumes topology of network by octet of server's IP address
 - `101.102.103.104 = x.<DC octet>.<rack octet>.<node octet>`
 - **PropertyFileSnitch:** uses a config file
 - **EC2Snitch:** uses EC2.
 - EC2 Region = DC
 - Availability zone = rack
- Other snitch options available

Writes

- Need to be lock-free and fast (no reads or disk seeks)
- Client sends write to one coordinator node in Cassandra cluster
 - Coordinator may be per-key, or per-client, or per-query
 - Per-key Coordinator ensures writes for the key are serialized
- Coordinator uses Partitioner to send query to all replica nodes responsible for key
- When X replicas respond, coordinator returns an acknowledgement to the client
 - X?

Writes (2)

- **Always writable: Hinted Handoff mechanism**
 - If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
 - When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours).
- **One ring per datacenter**
 - Per-DC coordinator elected to coordinate with other DCs
 - Election done via Zookeeper, which runs a Paxos (consensus) variant

Writes at a replica node

On receiving a write

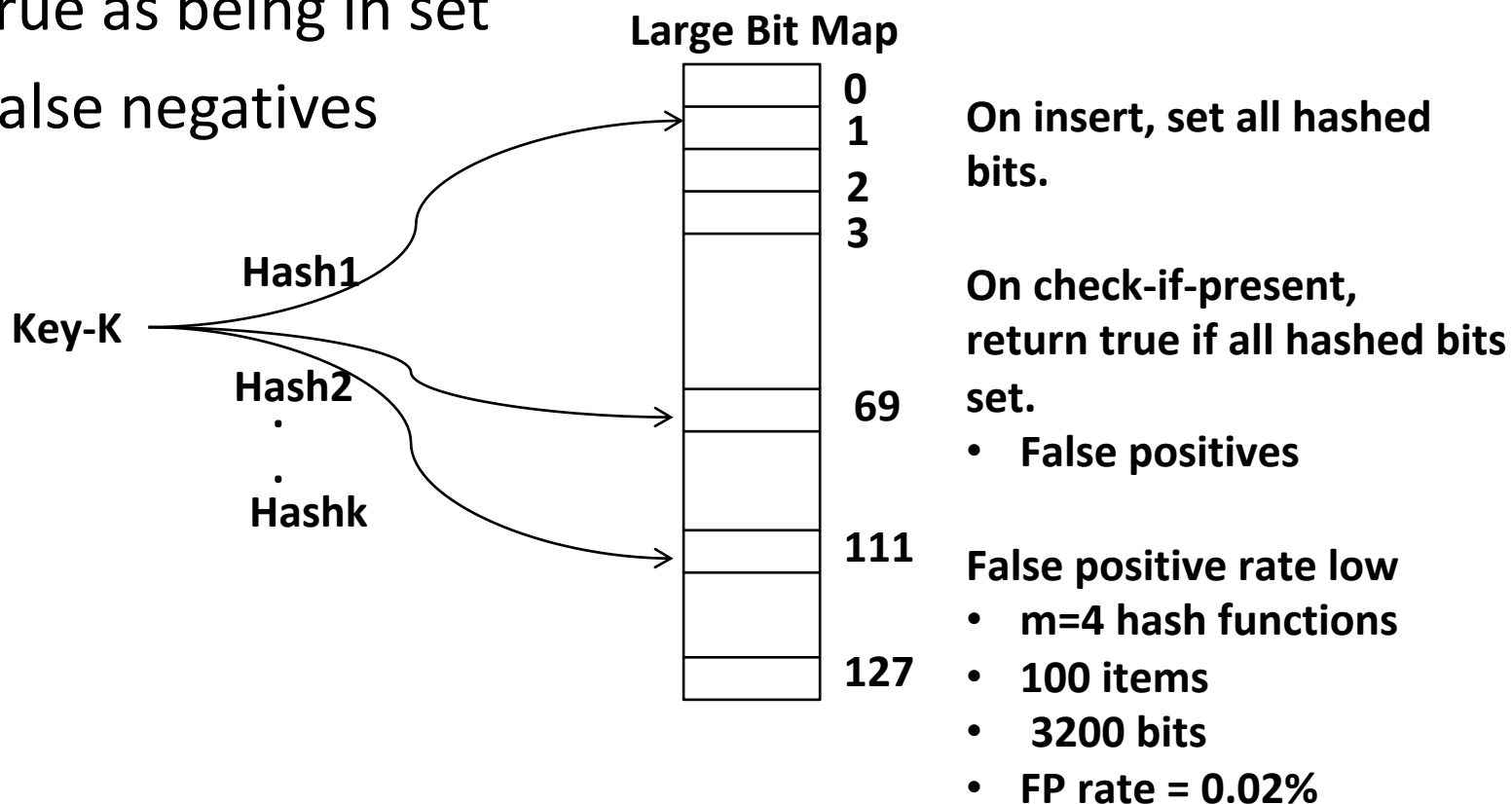
1. Log it in disk commit log (for failure recovery)
2. Make changes to appropriate memtables
 - **Memtable** = In-memory representation of multiple key-value pairs
 - *Typically append-only datastructure (fast)*
 - Cache that can be searched by key
 - Write-back as opposed to write-through

Later, when memtable is full or old, flush to disk

- Data File: An **SSTable** (Sorted String Table) – list of key-value pairs, sorted by key
- *SSTables are immutable (once created, they don't change)*
- Index file: An SSTable of (key, position in data sstable) pairs
- And a Bloom filter (for efficient search)

Bloom Filter

- Compact way of representing a set of items
- Checking for existence in set is cheap
- Some probability of false positives: an item not in set may check true as being in set
- Never false negatives



Compaction

Data updates accumulate over time and SSTables and logs need to be compacted

- The process of compaction merges SSTables, i.e., by merging updates for a key
- Run periodically and locally at each server

Deletes

Delete: don't delete item right away

- Add a **tombstone** to the log
- Eventually, when compaction encounters tombstone it will delete item

Reads

Read: Similar to writes, except

- **Coordinator can contact X replicas (e.g., in same rack)**
 - Coordinator sends read to replicas that have responded quickest in past
 - When X replicas respond, coordinator returns the latest-timestamped value from among those X
 - (X? We will check it later.)
- **Coordinator also fetches value from other replicas**
 - Checks consistency in the background, initiating a **read repair** if any two values are different
 - This mechanism seeks to eventually bring all replicas up to date
- **At a replica**
 - A row may be split across multiple SSTables => reads need to touch multiple SSTables => reads slower than writes (but still fast)

Membership

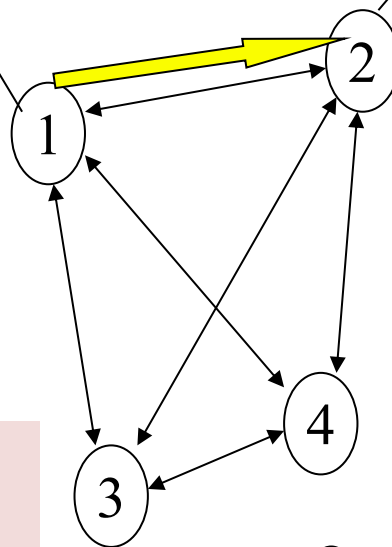
- Any server in cluster could be the coordinator
- So every server needs to maintain a list of all the other servers that are currently in the server
- List needs to be updated automatically as servers join, leave, and fail

Cluster Membership – Gossip-Style

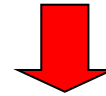
Cassandra uses gossip-based cluster membership

1	10120	66
2	10103	62
3	10098	63
4	10111	65

Address Time (local)
Heartbeat Counter



1	10118	64
2	10110	64
3	10090	58
4	10111	65



1	10120	70
2	10110	64
3	10098	70
4	10111	65

Current time : 70 at node 2
(asynchronous clocks)

Protocol:

- Nodes periodically gossip their membership list
- On receipt, the local membership list is updated, as shown
- If any heartbeat older than T_{fail} , node is marked as failed

(Remember this?)

Suspicion Mechanisms in Cassandra

- Suspicion mechanisms to adaptively set the timeout based on underlying network and failure behavior
- **Accrual detector:** Failure Detector outputs a value (PHI) representing suspicion
- Applications set an appropriate threshold
- **PHI calculation for a member**
 - Inter-arrival times for gossip messages
 - $\text{PHI}(t) =$
 - $\log(\text{CDF or Probability}(t_{\text{now}} - t_{\text{last}})) / \log 10$
 - PHI basically determines the detection timeout, but takes into account historical inter-arrival time variations for gossiped heartbeats
- In practice, $\text{PHI} = 5 \Rightarrow 10\text{-}15$ sec detection time

Cassandra Vs. RDBMS

- MySQL is one of the most popular (and has been for a while)
- On > 50 GB data
- **MySQL**
 - Writes 300 ms avg
 - Reads 350 ms avg
- **Cassandra**
 - Writes 0.12 ms avg
 - Reads 15 ms avg
- Orders of magnitude faster
- What's the catch? What did we lose?

CAP Theorem

CAP Theorem

- **Proposed by Eric Brewer (Berkeley)**
- Subsequently proved by Gilbert and Lynch (NUS and MIT)
- In a distributed system you can satisfy atmost 2 out of the 3 guarantees:
 - 1. Consistency:** all nodes see same data at any time, or reads return latest written value by any client
 - 2. Availability:** the system allows operations all the time, and operations return quickly
 - 3. Partition-tolerance:** the system continues to work in spite of network partitions

Why is Availability Important?

- **Availability** = Reads/writes complete reliably and quickly.
- Measurements have shown that a 500 ms increase in latency for operations at Amazon.com or at Google.com can cause a 20% drop in revenue.
- At Amazon, each added millisecond of latency implies a \$6M yearly loss.
- **User cognitive drift:** If more than a second elapses between clicking and material appearing, the user's mind is already somewhere else
- SLAs (Service Level Agreements) written by providers predominantly deal with latencies faced by clients.

Why is Consistency Important?

- **Consistency** = all nodes see same data at any time, or reads return latest written value by any client.
- When you access your bank or investment account via multiple clients (laptop, workstation, phone, tablet), you want the updates done from one client to be visible to other clients.
- When thousands of customers are looking to book a flight, all updates from any client (e.g., book a flight) should be accessible by other clients.

Why is Partition-Tolerance Important?

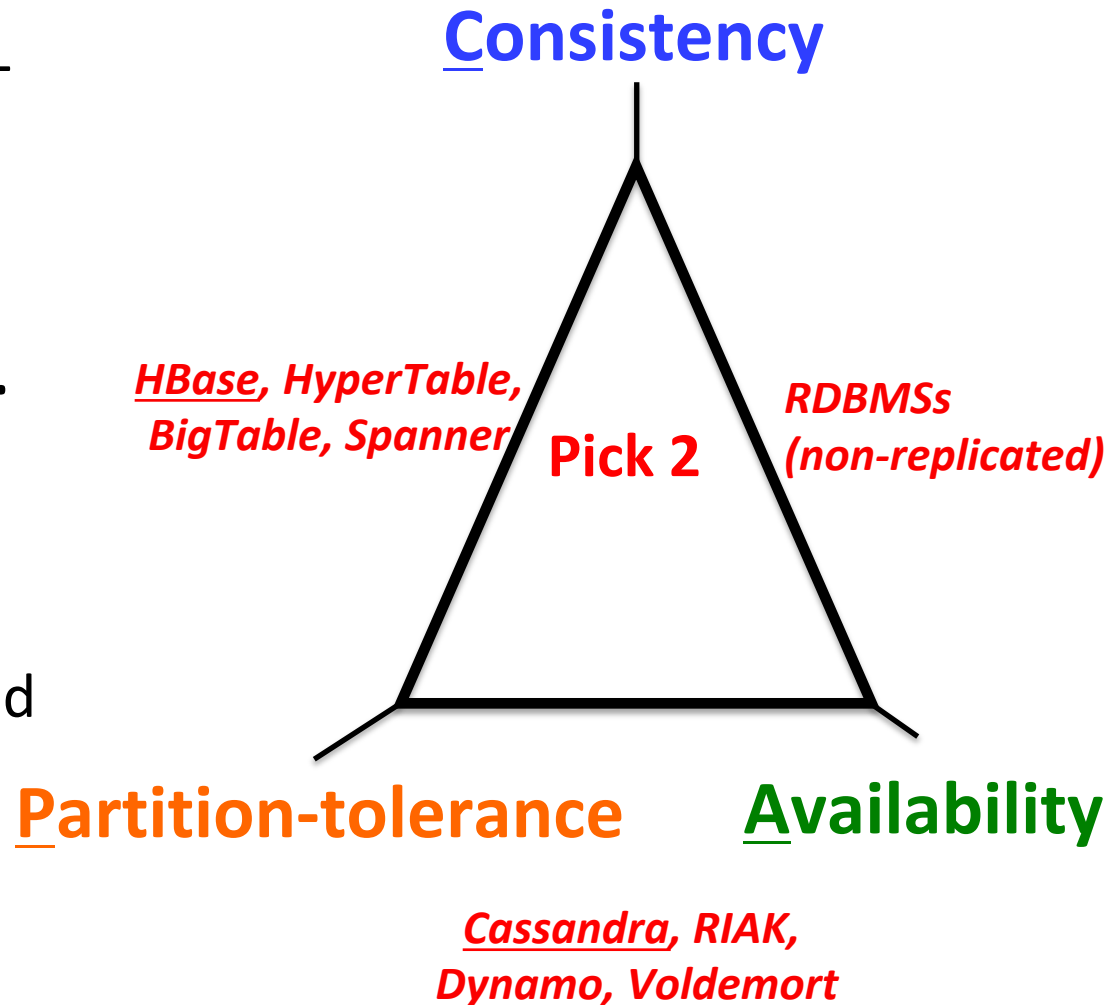
- **Partitions** can happen across datacenters when the Internet gets disconnected
 - Internet router outages
 - Under-sea cables cut
 - DNS not working
- Partitions can also occur within a datacenter, e.g., a rack switch outage
- Still desire system to continue functioning normally under this scenario

CAP Theorem Fallout

- Since partition-tolerance is essential in today's cloud computing systems, CAP theorem implies that a system has to choose between consistency and availability
- **Cassandra**
 - Eventual (weak) consistency, Availability, Partition-tolerance
- **Traditional RDBMSs**
 - Strong consistency over availability under a partition

CAP Tradeoff

- Starting point for NoSQL Revolution
- A distributed storage system can achieve **at most two of C, A, and P.**
- When partition-tolerance is important, you have to choose between consistency and availability



Eventual Consistency

- If all writes stop (to a key), then all its values (replicas) will converge eventually.
- If writes continue, then system always tries to keep converging.
 - **Moving “wave” of updated values lagging behind the latest values sent by clients, but always trying to catch up.**
- May still return stale values to clients (e.g., if many back-to-back writes).
- But works well when there a few periods of low writes – system converges quickly.

RDBMS vs. Key-value stores

- While RDBMS provide **ACID**
 - **A**tomicity
 - **C**onsistency
 - **I**solation
 - **D**urability
- Key-value stores like Cassandra provide **BASE**
 - **B**asically **A**vailable **S**oft-state **E**ventual Consistency
 - Prefers Availability over Consistency

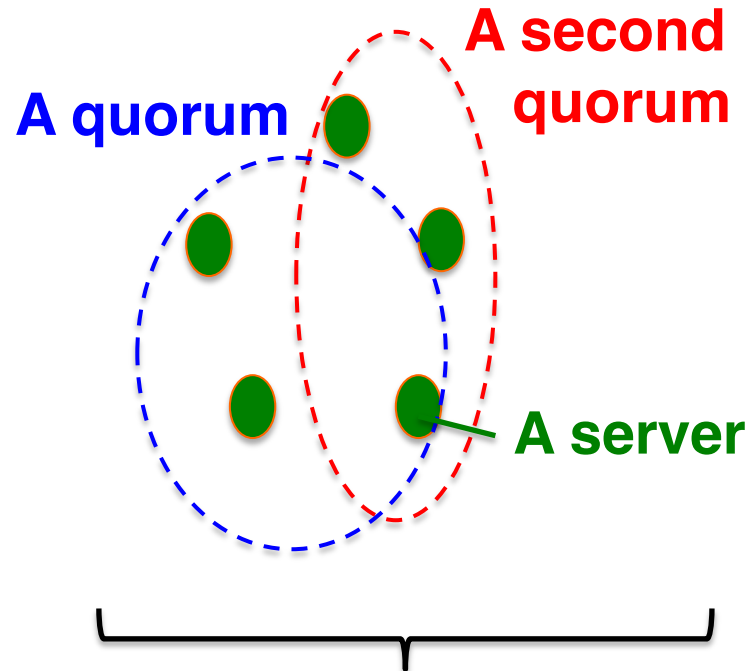
Consistency in Cassandra

- Cassandra has **consistency levels**
- Client is allowed to choose a consistency level for each operation (read/write)
 - **ANY:** any server (may not be replica)
 - Fastest: coordinator caches write and replies quickly to client
 - **ALL:** all replicas
 - Ensures strong consistency, but slowest
 - **ONE:** at least one replica
 - Faster than ALL, but cannot tolerate a failure
 - **QUORUM:** quorum across all replicas in all datacenters (DCs)
 - What?

Quorums for Consistency

In a nutshell:

- Quorum = majority
 - $> 50\%$
- Any two quorums intersect
 - Client 1 does a write in red quorum
 - Then client 2 does read in blue quorum
- At least one server in blue quorum returns latest write
- Quorums faster than ALL, but still ensure strong consistency



Five replicas of a key-value pair

Quorums in Detail

- Several key-value/NoSQL stores (e.g., Riak and Cassandra) use quorums.
- **Reads**
 - Client specifies value of **R** ($\leq N$ = total number of replicas of that key).
 - R = read consistency level.
 - Coordinator waits for R replicas to respond before sending result to client.
 - In background, coordinator checks for consistency of remaining $(N-R)$ replicas, and initiates read repair if needed.

Quorums in Detail (Contd..)

- Writes come in two flavors
 - Client specifies W ($\leq N$)
 - W = write consistency level.
 - Client writes new value to W replicas and returns. Two flavors:
 - Coordinator blocks until quorum is reached.
 - Asynchronous: Just write and return.

Quorums in Detail (Contd.)

- R = read replica count, W = write replica count
- Two necessary conditions:
 1. $W+R > N$
 2. $W > N/2$
- Select values based on application
 - $(W=1, R=1)$: very few writes and reads
 - $(W=N, R=1)$: great for read-heavy workloads
 - $(W=N/2+1, R=N/2+1)$: great for write-heavy workloads
 - $(W=1, R=N)$: great for write-heavy workloads with mostly one client writing per key

Cassandra Consistency Levels (Contd.)

- Client is allowed to choose a consistency level for each operation (read/write)
 - ANY: any server (may not be replica)
 - Fastest: coordinator may cache write and reply quickly to client
 - ALL: all replicas
 - Slowest, but ensures strong consistency
 - ONE: at least one replica
 - Faster than ALL, and ensures durability without failures
 - **QUORUM**: quorum across all replicas in all datacenters (DCs)
 - Global consistency, but still fast
 - **LOCAL_QUORUM**: quorum in coordinator's DC
 - Faster: only waits for quorum in first DC client contacts
 - **EACH_QUORUM**: quorum in every DC
 - Lets each DC do its own quorum: supports hierarchical replies

Types of Consistency

- Cassandra offers **Eventual Consistency**
- Are there other types of weak consistency models?

Consistency Solutions

Consistency Solutions



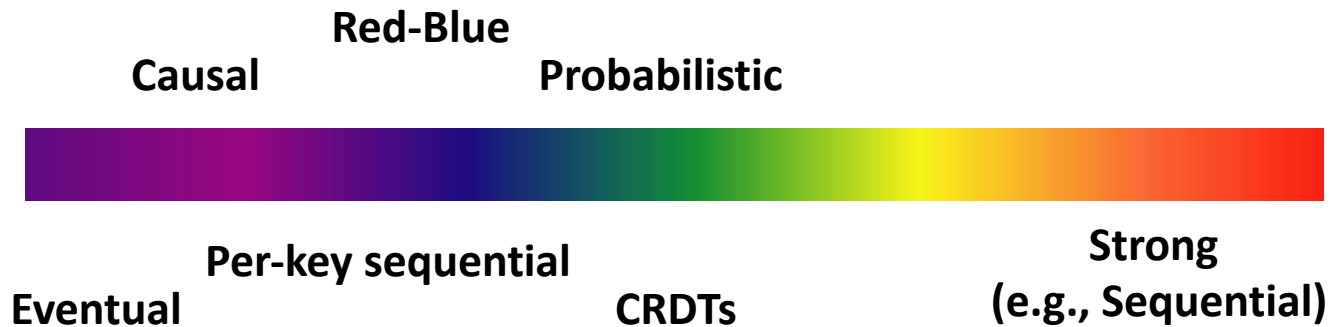
Eventual Consistency

- Cassandra offers **Eventual Consistency**
 - If writes to a key stop, all replicas of key will converge
 - Originally from Amazon's Dynamo and LinkedIn's Voldemort systems



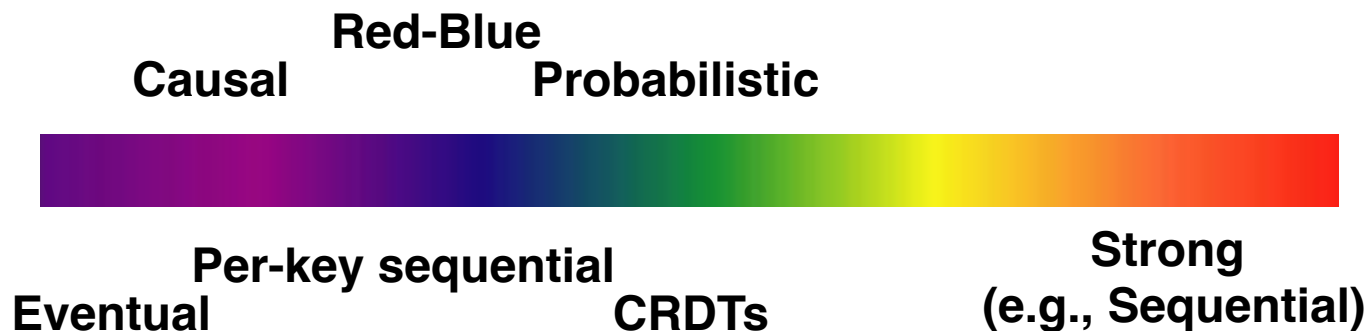
Newer Consistency Models

- Striving towards strong consistency
- While still trying to maintain high availability and partition-tolerance



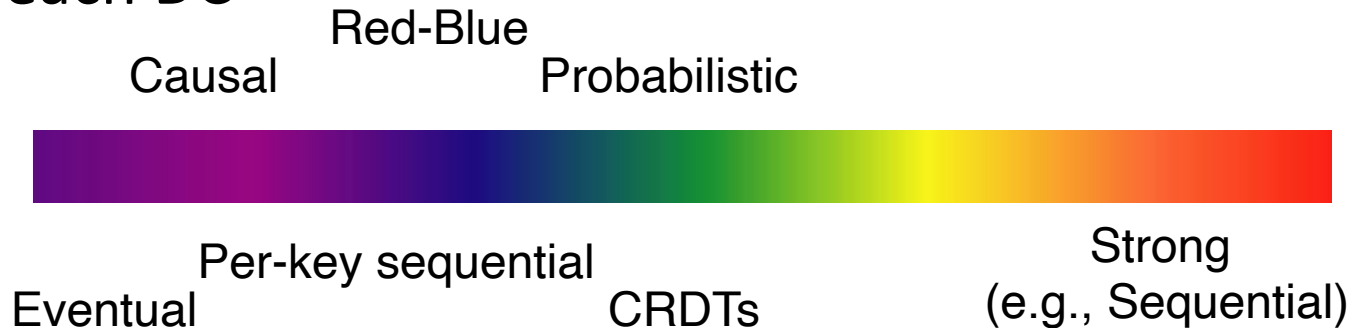
Newer Consistency Models (Contd.)

- **Per-key sequential:** Per key, all operations have a global order
- **CRDTs** (Commutative Replicated Data Types): Data structures for which commutated writes give same result [INRIA, France]
 - E.g., value == int, and only op allowed is +1
 - Effectively, servers don't need to worry about consistency



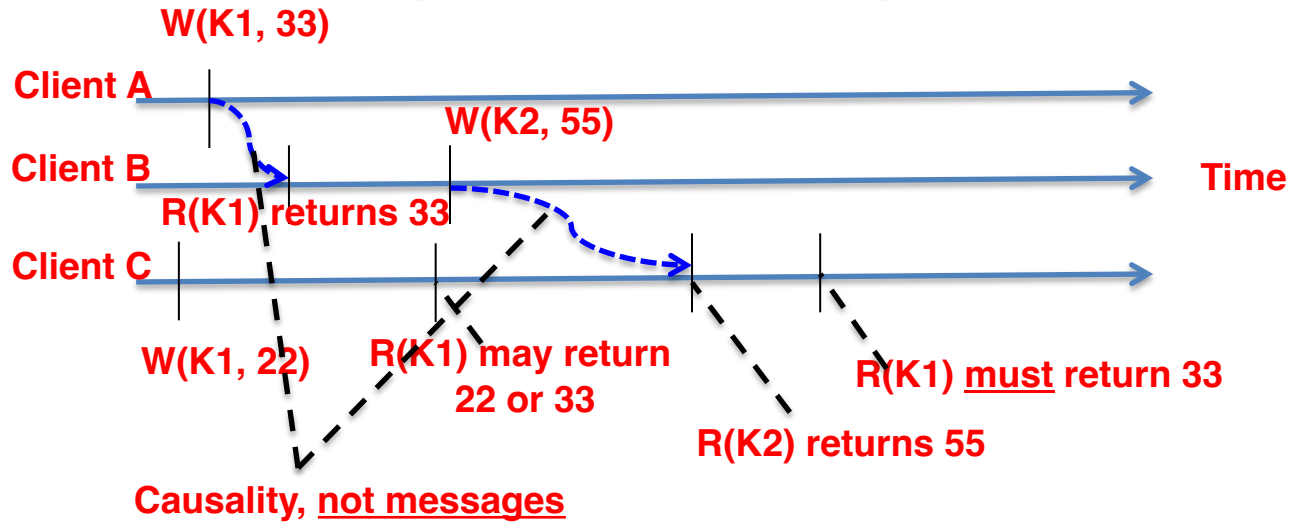
Newer Consistency Models (Contd.)

- **Red-blue Consistency:** Rewrite client transactions to separate operations into red operations vs. blue operations [MPI-SWS Germany]
 - Blue operations can be executed (commutated) in any order across DCs
 - Red operations need to be executed in the same order at each DC



Newer Consistency Models (Contd.)

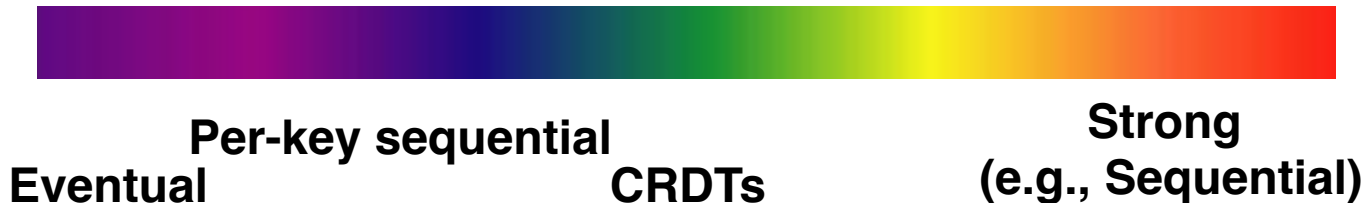
Causal Consistency: Reads must respect partial order based on information flow [Princeton, CMU]



Red-Blue

Causal

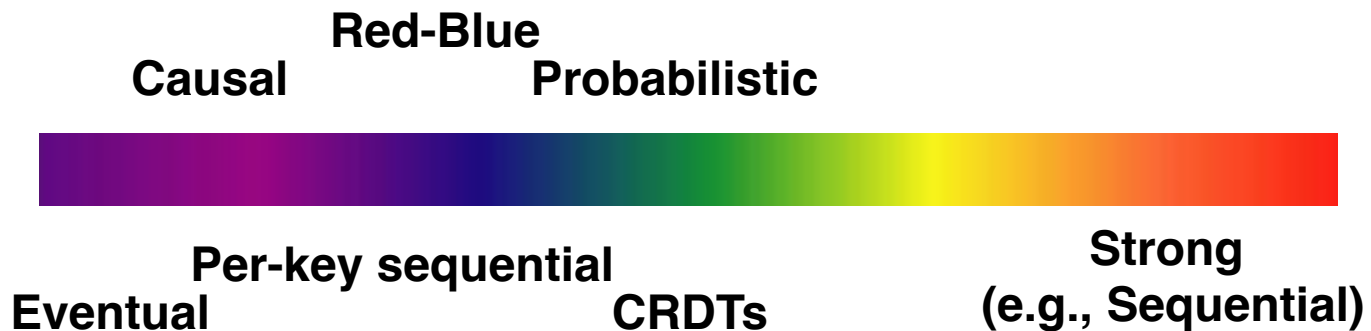
Probabilistic



Consistency Solutions

Which Consistency Model should you use?

- Use the lowest consistency (to the left) consistency model that is “correct” for your application
 - Gets you fastest availability



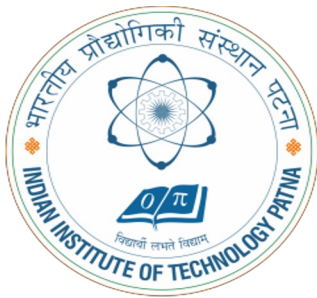
Strong Consistency Models

- **Linearizability:** Each operation by a client is visible (or available) instantaneously to all other clients
 - Instantaneously in real time
- **Sequential Consistency** [Lamport]:
 - *"... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."*
 - After the fact, find a “reasonable” ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.
- Transaction ACID properties, example: newer key-value/NoSQL stores (sometimes called **“NewSQL”**)
 - Hyperdex [Cornell]
 - Spanner [Google]
 - Transaction chains [Microsoft Research]

Conclusion

- Traditional Databases (RDBMSs) work with strong consistency, and offer ACID
- Modern workloads don't need such strong guarantees, but do need fast response times (availability)
- Unfortunately, CAP theorem
- **Key-value/NoSQL systems offer BASE**
[Basically Available Soft-state Eventual Consistency]
 - Eventual consistency, and a variety of other consistency models striving towards strong consistency
- We have also discussed the design of Cassandra and different consistency solutions.

Introduction to Edge ML with AWS IoT platform



Dr. Rajiv Misra

Professor, Dept. of Computer
Science & Engg. Indian Institute of
Technology Patna rajivm@iitp.ac.in

After Completion of this lecture you will knowing the following:

- Introduction to AWS IoT platform
- Layered architecture of AWS IoT
- Concepts of AWS IoT Core
- Understanding of AWS greengrass
- Event-Driven architecture with sensor data in AWS IoT

Recapitulate: Traditional IoT platform

Cloud

Globally available, unlimited compute resources

IoT

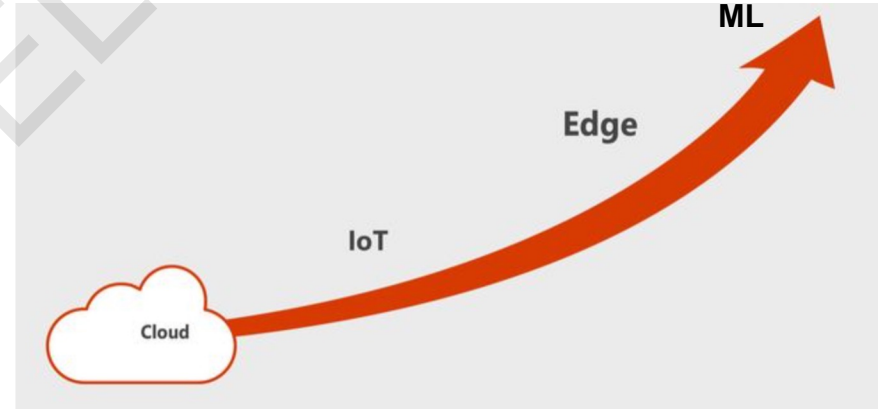
Harnessing signals from sensors and devices, managed centrally by the cloud

Edge

Intelligence offloaded from the cloud to IOT devices

ML

Breakthrough intelligence capabilities, in the cloud and on the edge



AWS IoT: Introduction

AWS IoT started in 2015 with Amazon acquiring a company called telemetry.

It started with several cloud services with a very simple IoT device management and M2M.

Now it has been expanded significantly.

AWS IoT architecture consists of three different layers:

- Things
- Cloud
- Intelligence



Devices
Sense & Act

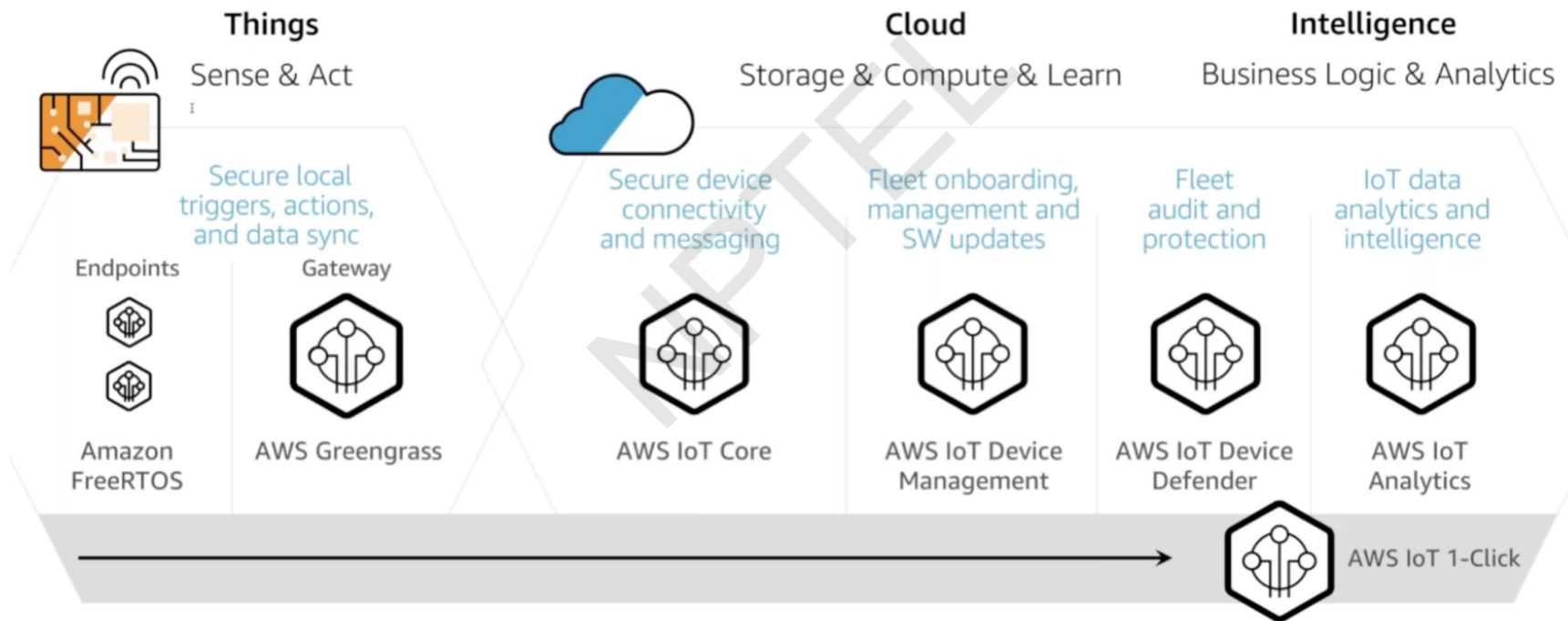


Cloud
Storage & Compute



Intelligence
Insights & Logic → Action

AWS IoT Architecture: Services Suite



AWS IoT Architecture: Things

- Things comprises of components which are on premises at the field and on the devices side, which actually sense data and act.
- Amazon offers a couple of products for this layer.
- First one is Amazon FreeRTOS which is a real-time operating system that can run on top of a microcontroller with 64 KB of memory or more
- Then AWS greengrass which is the edge computing software act as a interfacing with the local devices running either Amazon FreeRTOS or the AWS IoT devices SDK



AWS IoT Architecture: Cloud

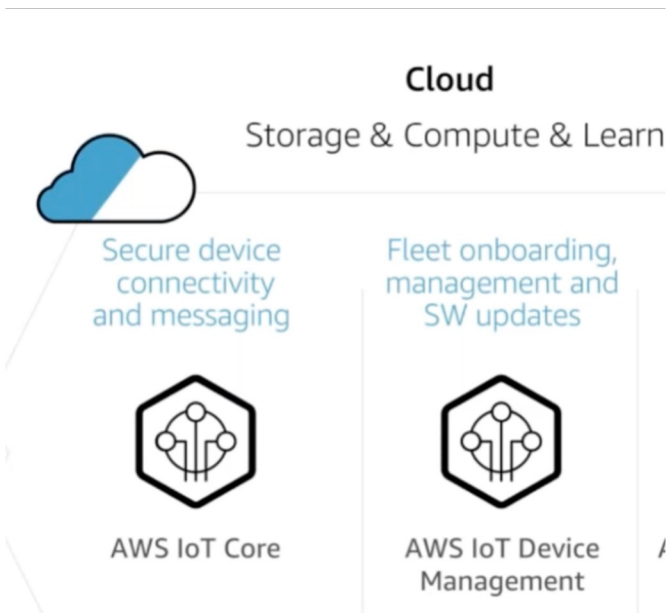
When it comes to cloud there are two important aspects:

The first one is **AWS IoT core** and as the name suggests it is the core building block of the AWS IoT platform and is responsible for registering the devices, so it acts as the device registry.

It also exposes endpoints for MQTT WebSockets and HTTP for the devices to talk to each other and to talk to the cloud and it is also the touch point for applications that want to control the devices running in the field.

AWS IoT core acts as an interface between the applications for example a mobile app that is talking to a device and similarly a device that is sending sensor data to the cloud.

Second one is **IoT device management** which supports bulk onboarding of devices because registering one device at a time in industrial use cases is not feasible, so it supports bulk onboarding and also has properties like over-the-air software updates, maintenance and performing bulk jobs, operations and so on.

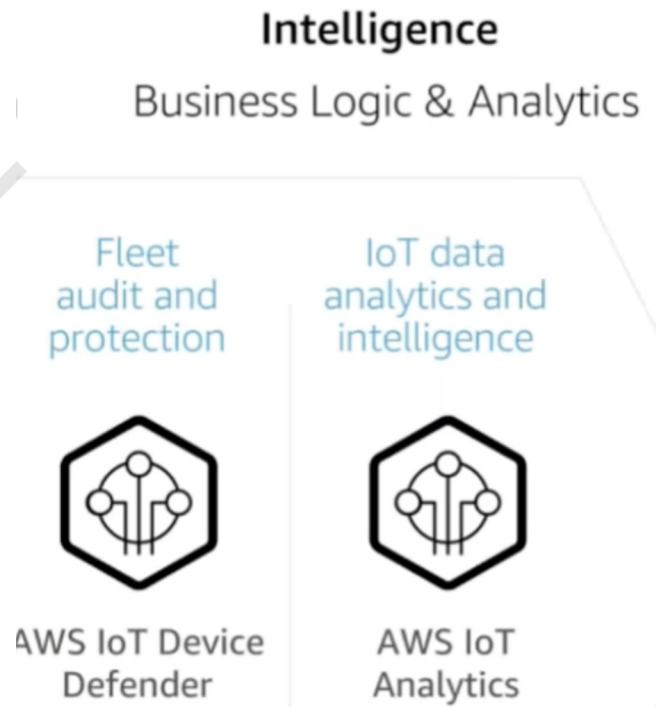


AWS IoT Architecture: Intelligence

AWS IOT device defender is all about security and if there is a drift between the preferred configuration and the policy and what is currently running and it automatically raises alerts.

It also maintains a highly secure footprint of all the devices and if there is any anomaly it raise an alert so that is the fleet audit or protection service.

Finally, AWS IoT analytics which is an analytic solution and this service is responsible for analyzing the trends, visualizing and from there feeding to more powerful systems like quick site or redshift and so on.



AWS IoT Core: Building Blocks



AWS IoT Core

Connect Devices at Scale

AWS IoT core is all about connecting devices to the cloud, the moment you bring in your first device that is going to become available you need to talk to AWS IoT core.

The workflow is very straightforward you need to register your device with AWS IoT core and that is going to act as the digital identity of your device.

AWS IoT Core is a managed cloud that lets connected devices easily and securely interact with cloud applications and other devices.



Custom Authentication
& Credentials Provider



Device Gateway



Message Broker



Rules Engine



Device Shadow



Device Registry

The moment you register a device you receive a set of credentials for the device and you're going to embed those credentials in the device and once the device has those credentials and it connects to the cloud it gets authenticated, authorized and it shows up in the device registry.

The device could be running a microcontroller, a single board computer, a slightly more powerful machine that can talk to an Modbus or canvas internally or, even an automobile device like a car.

After that it can send messages to the cloud and it can receive commands from the cloud.

AWS IoT Core: Building Blocks



AWS IoT Core

Connect Devices at Scale

When you zoom into AWS IoT core, the first one is all about authentication and authorization and the second one is device gateway which is the cloud endpoint for talking to the IoT core.

AWS IoT Core is a managed cloud that lets connected devices easily and securely interact with cloud applications and other devices.

Message broker which is based on MQTT WebSockets and HTTP for publishing and subscribing messages or feeding data from the device to the cloud but it is predominantly uses it for a communication between devices at the cloud to send some metadata or telemetry and to receive some settings or commands.



Custom Authentication
& Credentials Provider



Device Gateway



Message Broker



Rules Engine



Device Shadow



Device Registry

There is a rules engine which decides how the messages will flow into rest of the system and the rules engine is ANSI SQL compliant that writes simple select statements that will filter the messages and apply a rule and the outcome of this rule can be hooked to a lambda function to take further action.

The device shadow is the digital twin or digital identity of the physical device and all the changes that are made to the device will first get applied to the device shadow and then it gets propagated all the way to the device. When the device state changes it automatically gets synchronized with the device shadow. It acts as the buffer between the desired state and the current state.

The job of the AWS IoT core is to make sure that the desired configuration is matching with the current configuration or not.

Device registry is a huge database repository but meant for the devices or things that you connect to AWS IoT.

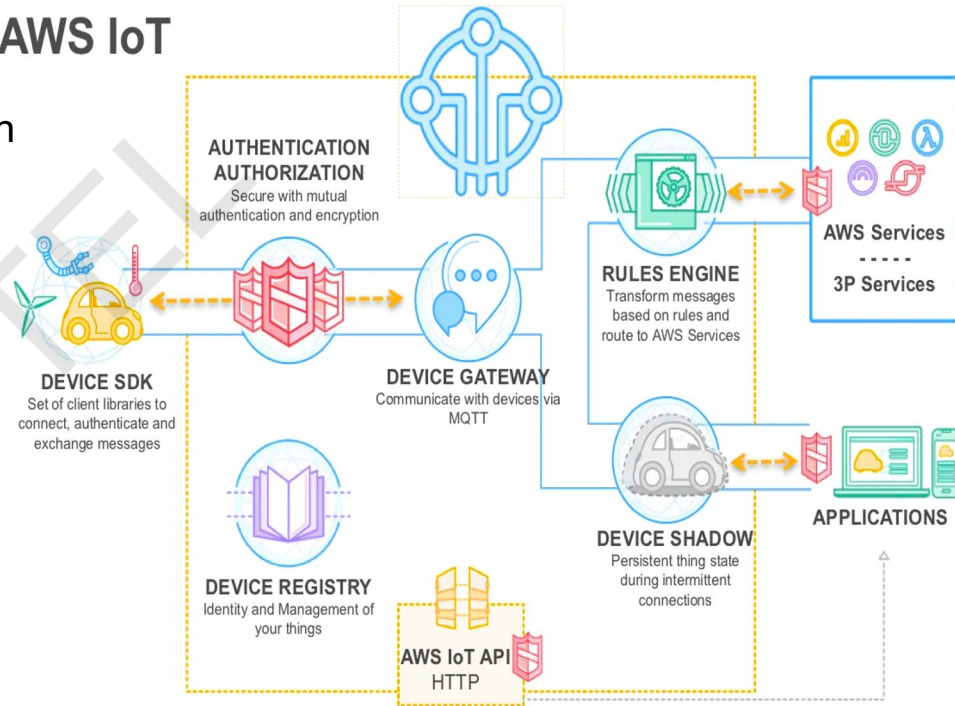
AWS IoT Core: Summary

To put things in perspective, for using multiple building blocks of AWS IoT, the device SDK which is supported in variety of languages like node.js, Python, C, Java where SDK is used to connect your device to the cloud.

The first touch point is authentication and authorization and then the device gateway for communication and further it goes to a rules engine and device shadow which maintains a replica of this state

The rules engine is responsible for extending the IoT platform to rest of AWS services like dynamo DB, Neptune, redshift, AWS sage maker and to third-party services.

AWS IoT



AWS Greengrass: Building Blocks

AWS Greengrass extends AWS IoT to your devices so they can act locally and the data that they generate or filter is filtered before it is sent to the cloud.

Like AWS IoT core there is a message broker built into green grass so devices can continue to talk to each other, there is a compute layer which is based on lambda to write functions that are running locally and triggered when a specific condition is met and these triggers will actually fire lambda functions that perform an action.

Greengrass also have the data and state synchronized with the cloud with the help of local device shadows and the cloud device shadow. If something updated locally, it first gets written to the device shadow running on the edge and then it eventually gets synchronized with the cloud.

Greengrass provides local resource access. For example, you want to talk to a local database which already has some metadata or material asset tracking information you can you can query that directly, talk to the file system, databases or anything that is accessible within the network.



AWS Greengrass

Extend AWS IoT to the edge

AWS Greengrass extends AWS IoT onto your devices, so that they can act locally on the data they generate, while still taking advantage of the cloud



Local actions



Local triggers



Data and state sync.
Over the air updates



Security.
Protocol adapter for OPC-UA



Local resource access



Local ML inferencing
New!

AWS Greengrass: Building Blocks

The most recent feature of greengrass is the ability to run machine learning inferencing on the edge and this is one of the key drivers because there are three aspects when it comes to IOT.

First one is the learning part which is happening in the cloud, where you train machine learning models then you have decision-making that takes place at the edge and where fully trained machine learning models are used and they make decisions on behalf of the cloud and the action phase that is directly done by the devices.

For example, a machine learning model trained in the cloud to find an anomaly is deployed on the edge and because an anomaly is found with a very critical device, the machine learning model decides that one of the other equipments need to be shut down and that decision will result in an action where an actuator or a relay or another interface physically shuts down a malicious or a vulnerable machine to avoid any eventuality or any fatalities.

Thus the learn, decide and act cycle that happens with the cloud, edge and devices and performing the decision part run locally by ML inferencing.



AWS Greengrass

Extend AWS IoT to the edge

AWS Greengrass extends AWS IoT onto your devices, so that they can act locally on the data they generate, while still taking advantage of the cloud



Local actions



Local triggers



Data and state sync.
Over the air updates



Security.
Protocol adapter for OPC-UA



Local resource access



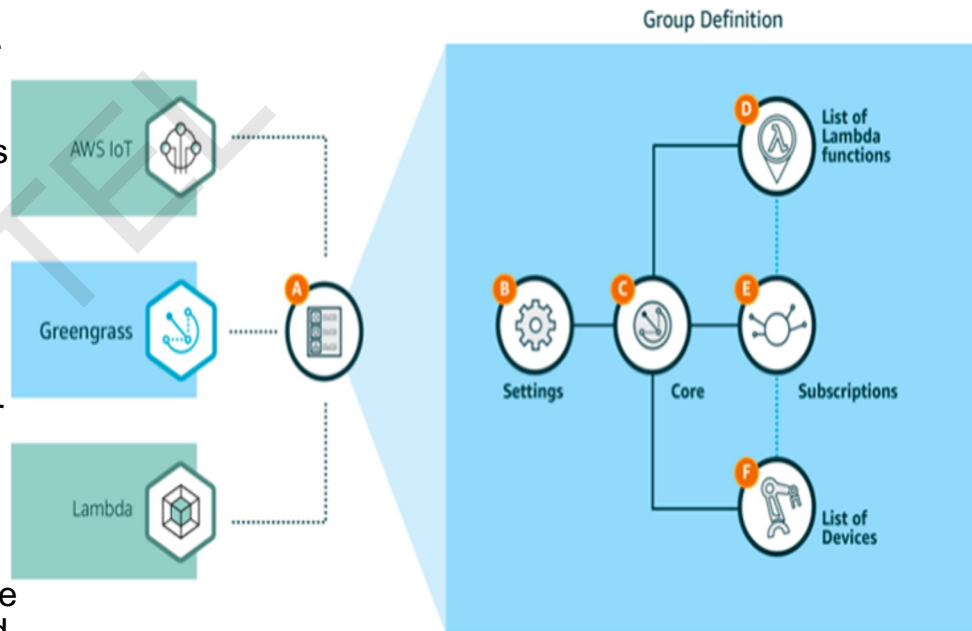
Local ML inferencing
New!

AWS Greengrass Group: Cloud Capabilities to the Edge

AWS IoT Greengrass Group: An AWS IoT Greengrass group is a collection of settings and components, such as an AWS IoT Greengrass core, devices, and subscriptions. Groups are used to define a scope of interaction. For example, a group might represent one floor of a building, one truck, or an entire mining site. Since the group acts as the logical boundary for all the devices, it enforces consistent configuration and policies to all the entities.

AWS IoT Greengrass Core: This is just a device in AWS IoT Core registry that doubles up as an edge device. It is an x86 and ARM computing device running the Greengrass runtime. Local devices talk to the Core similar to the way they interact with AWS IoT Core.

AWS IoT Devices: These are the devices that are a part of the Greengrass group. Once devices become a part of the group, they automatically discover the Core to continue the communication. Each device has a unique identity and runs AWS IoT Device SDK. Existing devices can be added to a Greengrass Group.



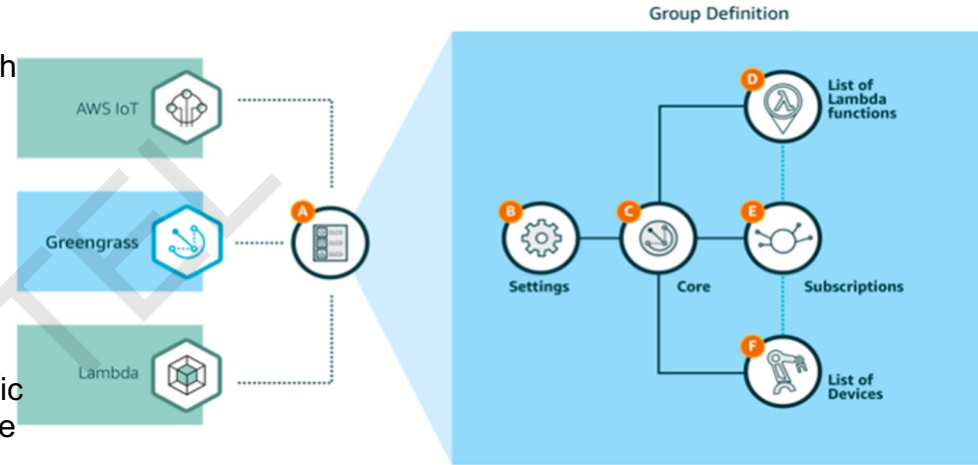
AWS Greengrass Group: Cloud Capabilities to the Edge

Lambda Functions: As discussed earlier, Lambda provides the local compute capabilities for AWS IoT Greengrass. Each function running within the Core uses Greengrass SDK to interact with the resources and devices. Lambda functions can be customized to run within the Greengrass sandbox container or directly as a process within the device OS.

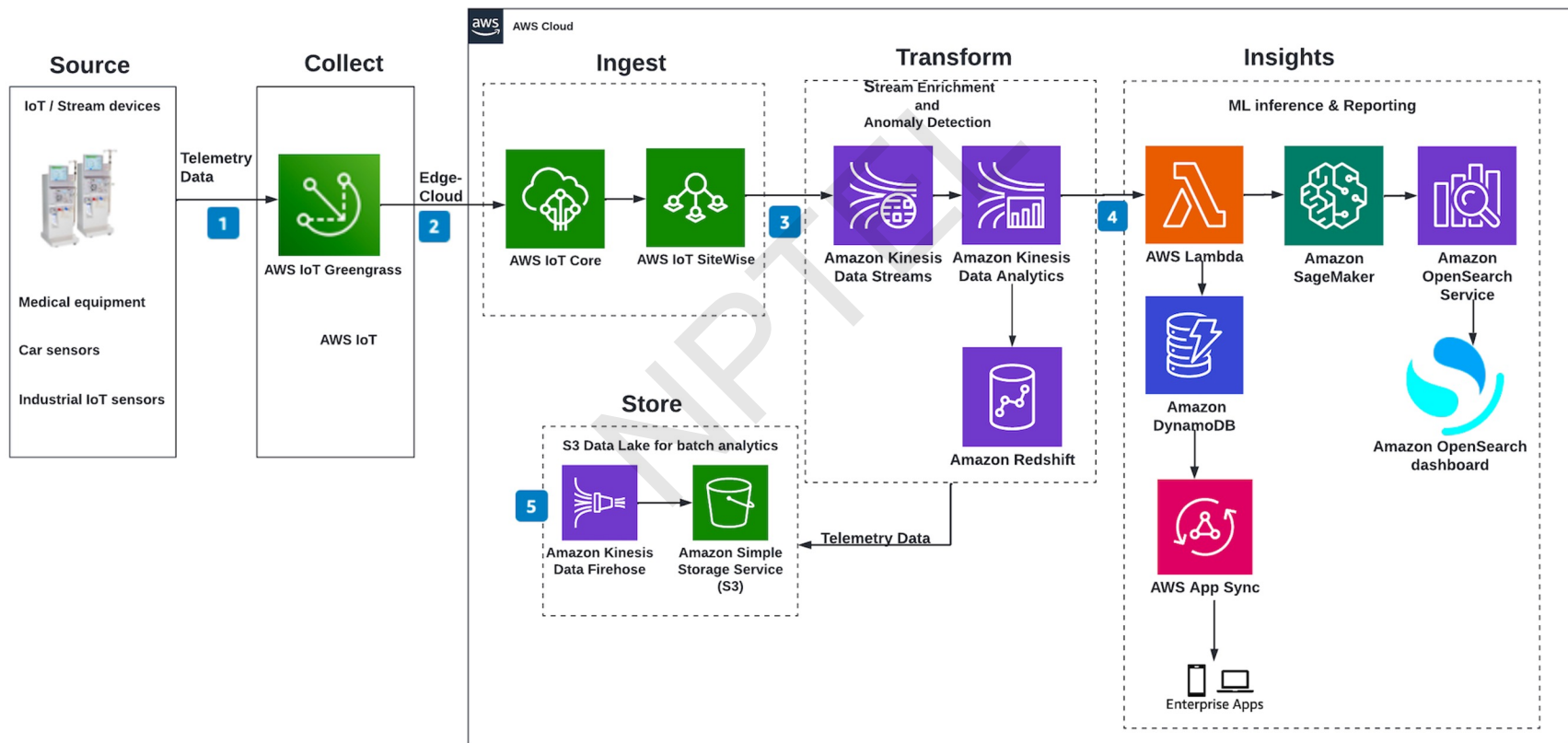
Subscriptions: AWS IoT Greengrass subscriptions connect the resources declaratively. It maintains a list of publishers and subscribers that exchange messages. For another scenario, a Lambda function may publish messages to a topic to which the device is subscribed. Subscriptions eliminate the strong dependency between publishers and consumers by effectively decoupling them.

Connectors: AWS IoT Greengrass Connectors allows developers to easily build complex workflows on AWS IoT Greengrass without having to worry about understanding device protocols, managing credentials, or interacting with external APIs. Based on a declarative mechanism, Connectors extend the edge computing scenarios to 3rd party environments and services. Connectors rely on Secrets for maintaining the API keys, passwords, and credentials needed by external services.

ML Inferencing: This is one of the recent additions to AWS IoT Greengrass. The trained model is first uploaded to an Amazon S3 bucket that gets downloaded locally. A Lambda function responsible for inferencing inbound data stream publishes the predictions to a MQTT topic after loading the local model. Since Python is a first-class citizen in Lambda, many existing modules and libraries can be used to perform ML inferencing at the edge.



AWS IoT: Event-driven architecture with sensor data



AWS IoT: Event-driven architecture with sensor data

Phase 1:

- Data originates in IoT devices such as medical devices, car sensors, industrial IoT sensors.
- This telemetry data is collected using **AWS IoT Greengrass**, an open-source IoT edge runtime and cloud service that helps your devices collect and analyze data closer to where the data is generated.
- When an event arrives, AWS IoT Greengrass reacts autonomously to local events, filters and aggregates device data, then communicates securely with the cloud and other local devices in your network to send the data.

Phase 2:

- Event data is ingested into the cloud using edge-to-cloud interface services such as AWS IoT Core, a managed cloud platform that connects, manages, and scales devices easily and securely.
- AWS IoT Core interacts with cloud applications and other devices.
- You can also use AWS IoT SiteWise, a managed service that helps you collect, model, analyze, and visualize data from industrial equipment at scale.

AWS IoT: Event-driven architecture with sensor data

Phase 3:

- AWS IoT Core can directly stream ingested data into Amazon Kinesis Data Streams.
- The ingested data gets transformed and analyzed in near real time using Amazon Kinesis Data Analytics with Apache Flink and Apache Beam frameworks.
- Stream data can further be enriched using lookup data hosted in a data warehouse such as Amazon Redshift.

Phase 4:

- Amazon Kinesis Data Analytics can persist SQL results to Amazon Redshift after the customer's integration and stream aggregation (for example, one minute or five minutes).
- The results in Amazon Redshift can be used for further downstream business intelligence (BI) reporting services, such as Amazon QuickSight.
- Amazon Kinesis Data Analytics can also write to an AWS Lambda function, which can invoke Amazon SageMaker models.
- Amazon SageMaker is a the most complete, end-to-end service for machine learning.

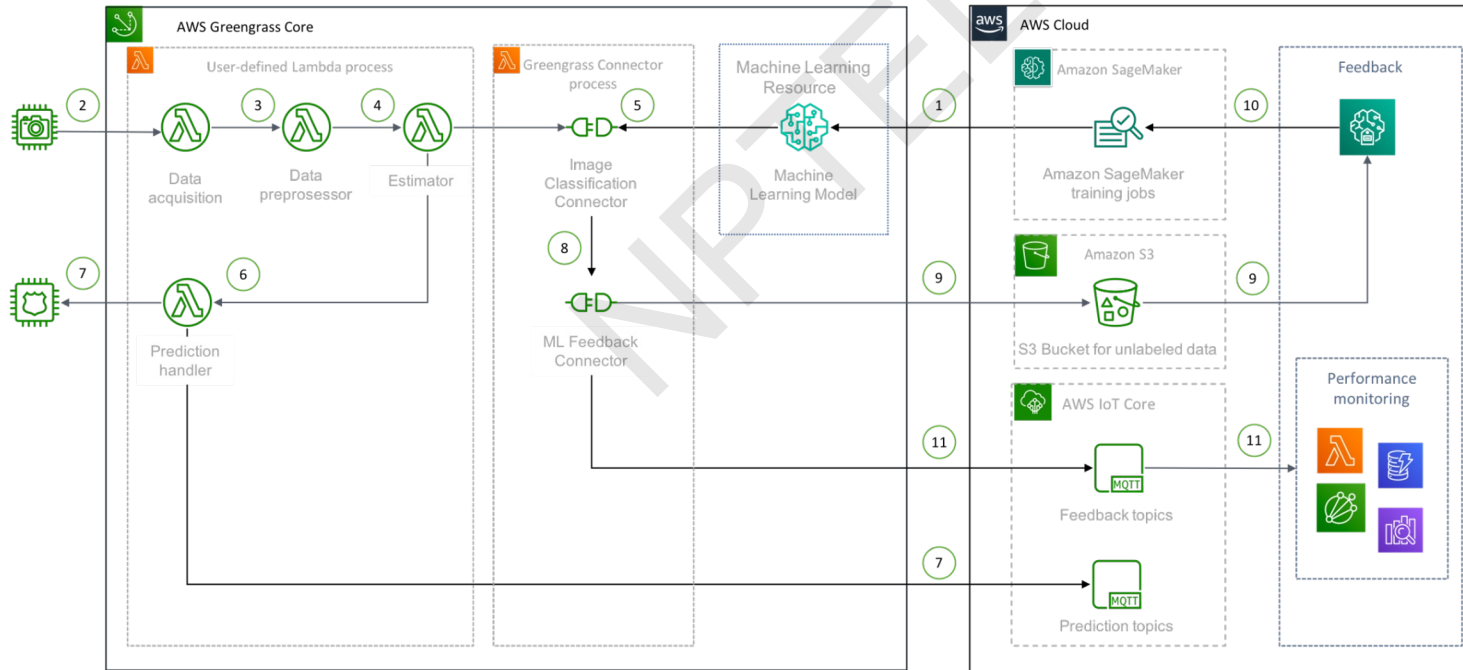
AWS IoT: Event-driven architecture with sensor data

Phase 5:

- Once the ML model is trained and deployed in SageMaker, inferences are invoked in a micro batch using AWS Lambda.
- Inferred data is sent to Amazon OpenSearch Service to create personalized monitoring dashboards using Amazon OpenSearch Service dashboards.
- The transformed IoT sensor data can be stored in Amazon DynamoDB.
- Customers can use AWS AppSync to provide near real-time data queries to API services for downstream applications.
- These enterprise applications can be mobile apps or business applications to track and monitor the IoT sensor data in near real-time.
- Amazon Kinesis Data Analytics can write to an Amazon Kinesis Data Firehose stream, which is a fully managed service for delivering near real-time streaming data to destinations like Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service, Splunk, and any custom HTTP endpoints or endpoints owned by supported third-party service providers, including Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, and Sumo Logic.

Use Case: Greengrass Machine Learning Inference

This use case describes the steps in setting up Greengrass Machine Learning Inference, using Greengrass Image Classification ML Connector with model trained with Amazon SageMaker, and Greengrass ML Feedback connector to send data back to AWS for model retraining or prediction performance analysis.



Use Case: Greengrass Machine Learning Inference

The common design patterns of using Greengrass Connectors:

1. Creates a Amazon SageMaker training job to create the model. When the Greengrass configuration is being deployed, the Greengrass Core will download the model from the *Amazon SageMaker* training job as a local machine learning resource.
2. Data acquisition - This function periodically acquire the raw data inputs from a image source. In this example, we are using static images to simulate image sources.
3. Data preprocessor - This function pre-process the image by resize to the images used to train the model.
4. Estimator - This function predict the data input with the connector via IPC
5. Greengrass ML Image Classification Connector - The Connector loads the model from local Greengrass resource and invoke the model.
6. The process will handle the prediction result, with object detected and confidence level.
7. The result can be used to trigger an action, or send it back to the cloud for further processing.
8. Greengrass ML Feedback Connector - Greengrass ML Feedback Connector sends field data back to AWS according to the sampling strategy configured
9. Greengrass ML Feedback Connector sends unlabeled data to AWS
10. Unlabeled data can be labeled using Amazon Ground Truth, and the labeled data can be used to retrain the model
11. Greengrass ML Feedback Connector sends prediction performance which can be used for realtime performance analysis.

Use Case Greengrass ML Inference: Deployment

The main steps for deployment are:

1. *Prerequisites.* Ensure there is an AWS IoT certificate and private key created and accessible locally for use.
2. *Train the ML model.* We will use an example notebook from Amazon SageMaker to train the model with the Image Classification Algorithm provided by Amazon SageMaker.
3. *Generate and launch the CloudFormation stack.* This will create the Lambda functions, the Greengrass resources, and an AWS IoT thing to be used as the Greengrass Core. The certificate will be associated with the newly created Thing. At the end, a Greengrass deployment will be created and ready to be pushed to the Greengrass core hardware.
4. *Create the config.json file*, using the outputs from the CloudFormation. Then place all files into the `/greengrass/certs` and `/greengrass/config` directories.
5. *Deploy to Greengrass.* From the AWS Console, perform a Greengrass deployment that will push all resources to the Greengrass Core and start the MLI operations.

Use Case Greengrass ML Inference: Deployment

Prerequisites:

- **AWS Cloud.**

Ensure you have an AWS user account with permissions to manage `iot`, `greengrass`, `lambda`, `cloudwatch`, and other services during the deployment of the CloudFormation stack.

- **Local Environment**

Ensure a recent version of the AWS CLI is installed and a user profile with permissions mentioned above is available for use.

- **Greengrass Core AWS IoT**

Greengrass Core SDK Software which can be installed using pip command `sudo pip3.7 install greengrasssdk`

Use Case Greengrass ML Inference: Deployment

Train the model with Amazon SageMaker:

We will train the model using algorithm provided by Amazon SageMaker, [Amazon SageMaker Image Classification Algorithm](#) and [Caltech-256 dataset](#).

- Login to Amazon SageMaker Notebook Instances console <https://console.aws.amazon.com/sagemaker/home?#/notebook-instances>
- Select `Create notebook instance`
- Enter a name in Notebook instance name, such as `greengrass-connector-training`
- Use the default `ml.t2.medium` instance type
- Leave all default options and select `Create notebook instance`
- Wait for the instance status to be `InService`, and select `Open Jupyter`
- Select `SageMaker Example` tab, expand `Sagemaker Neo Compilation Jobs, Image-classification-fulltraining-highlevel-neo.ipynb`, select `Use`
- Keep default option for the file name and select `Create copy`

Use Case Greengrass ML Inference: Deployment

Train the model with Amazon SageMaker:

- We are to use transfer learning approach with `use_pretrained_model=1`. Locate the cell that configure the `hyper-parameters` and add the additional `use_pretrained_model=1`. Details of the hyperparameters can be found in [Amazon SageMaker Developer Guide - Image Classification Hyperparameters](#)
- We will also be setting the prefix for our training job so that the [Cloudformation Custom Resources](#) is able to get the latest training job. Configure a `base_job_name` in the `sagemaker.estimator`. Locate the cell that initialize the `sagemaker.estimator` and add the `base_job_name`, for example, using `greengrass-connector` as the prefix. You will need this name prefix when creating the stack.
- Add a cell below the cell that do the training `ic.fit()` and the command `ic.latest_training_job.name` in the empty cell. This will give you the name of the training job that you can verify to make sure the Cloudformation stack picks up the correct job.
- Select the `Cell` from the notebook menu and `Run All`

Use Case Greengrass ML Inference: Deployment

Launch the CloudFormation Stack:

Prior to launching the accelerator locally, a CloudFormation package needs to be created, and then the CloudFormation stack launched from the Template. Follow the steps below to create the package via the command line, and then launch the stack via the CLI or AWS Console.

The CloudFormation template does most of the heavy lifting. Prior to running, each *input* template needs to be processed to an *output* template that is actually used. The package process uploads the Lambda functions to the S3 bucket and creates the output template with unique references to the uploaded assets.

Use Case Greengrass ML Inference: Deployment

Configure the Greengrass Core:

With the stack deployed, we use one output from the CloudFormation stack, the *GreengrassConfig* value, along with the certificate and private key to complete the `config.json` so that Greengrass Core can connect and authenticate.

Starts the Greengrass Core:

With the Greengrass configuration `config.json` in place, start the Greengrass Core.

Use Case Greengrass ML Inference: Deployment

Deploy Cloud Configurations to the Greengrass Core:

From the AWS Console of AWS IoT Greengrass, navigate to the Greengrass Group you created with the Cloudformation, and perform *Actions->Deploy* to deploy to the Greengrass Core machine.

Use Case Greengrass ML Inference: Testing

To test out this accelerator without any hardware, you can install the Greengrass on an EC2 to simulate as a Greengrass Core

1. Create a EC2 running Greengrass, using the Cloudformation template in `cfn/greengrass_core_on_ec2-s3_models.cfn.yml`
2. Once the instance is created, copy the `greengrass-setup.zip` to the EC2
3. In the EC2, extract `greengrass-setup.zip` into `/greengrass` folder using command `sudo unzip -o greengrass-setup.zip -d /greengrass`
4. Restart the Greengrass daemon using the command `sudo systemctl restart greengrass`

Lecture Summary

- Introduction to AWS IoT platform
- Layered architecture of AWS IoT
- Concepts of AWS IoT Core
- Understanding of AWS greengrass
- Event-Driven architecture with sensor data in AWS IoT



THANK YOU!

Introduction to Federated Learning at IoT Edge



Dr. Rajiv Misra, Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

After Completion of this lecture you will knowing the following:

- Current IoT scenarios
- Why there is a need to shift from centralized ML training to decentralized ML training of data?
- Concepts of Federated Learning (ie Distributed ML)
- Several challenges of federated learning

Current IoT Scenario

Explosion of IoT Market

- McKinsey reported \$11.1 Trillion market value by 2025
- 14 billion connected devices - Bosch
- 5 billion connected devices - Cisco
- 309 billion IoT supplier revenue - Gartner
- 7.1 trillion IoT solutions revenue - IDC

A “deluge of data” is observed in 2020

1.5 GB of traffic per day from average internet user

3000 GB per day - Smart Hospitals

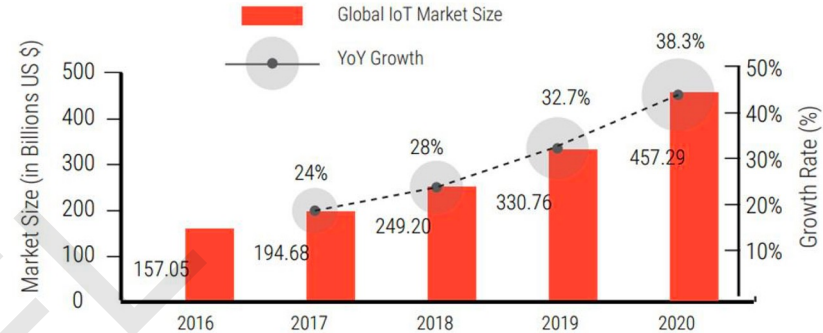
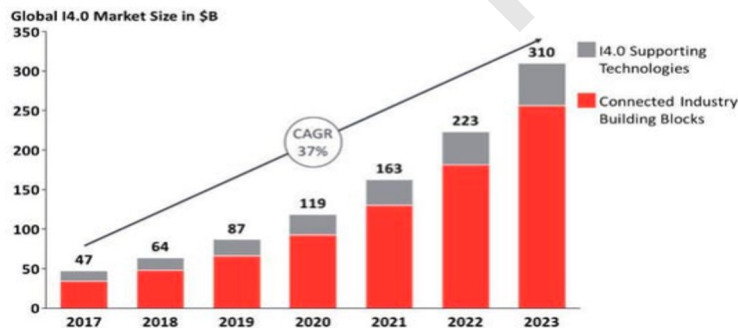
4000 GB data per day - self driving cars EACH

Radars ~ 10-100 kb per sec

40,000 GB per day - connected aircrafts

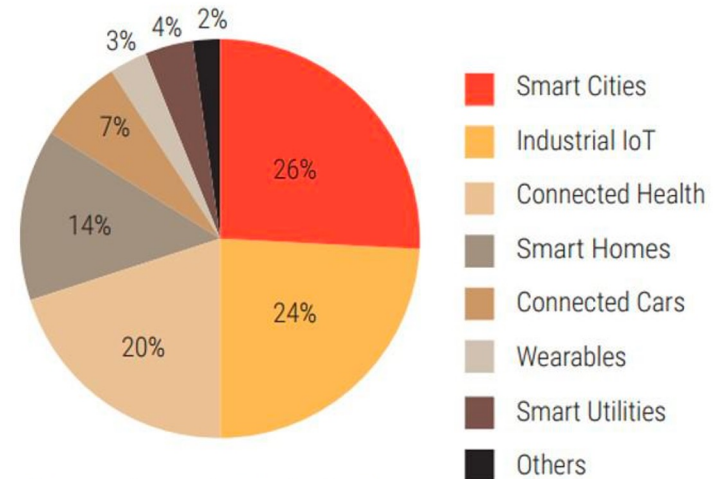
1,000,000 GB per day - connected factories

 Global Industry 4.0 Market Size 2017-2023



[Sources: GrowthEnabler Analysis/MarketsandMarkets]

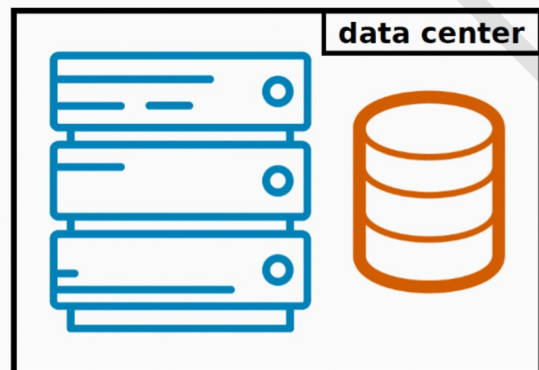
Global IoT Market Share by Sub-Sector



[Source: GrowthEnabler Analysis]

Shift from Centralized to Decentralized data

- The standard setting in Machine Learning (ML) considers a centralized dataset processed in a tightly integrated system
- But in the real world data is often decentralized across many IOT devices
- Sending the data to Cloud for centralized ML may be too costly
 - Self-driving cars are expected to generate several TBs of data a day
 - Some wireless devices have limited bandwidth/power
- Data may be considered too sensitive sometimes such as medical reports
 - We see a growing public awareness and regulations on data privacy
 - Keeping control of data can give a competitive advantage in business and research



≠



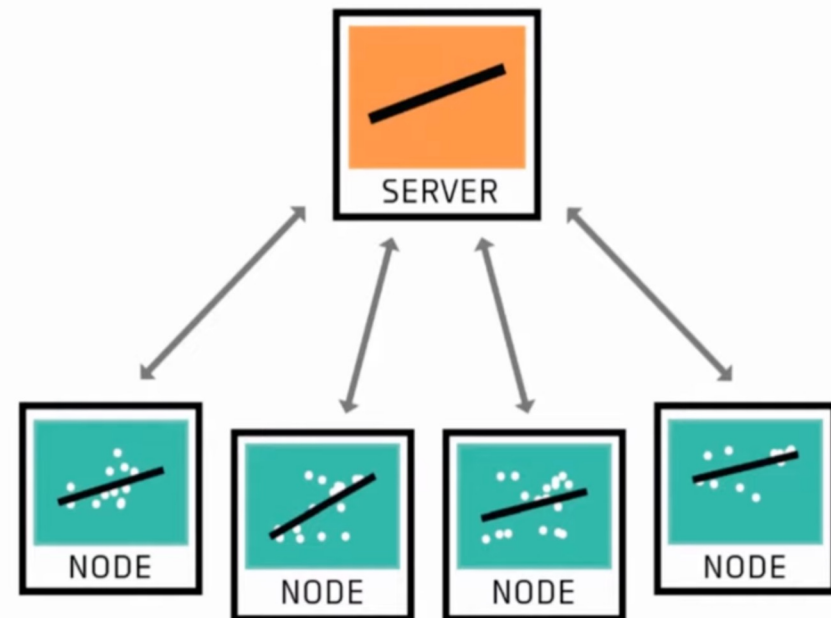
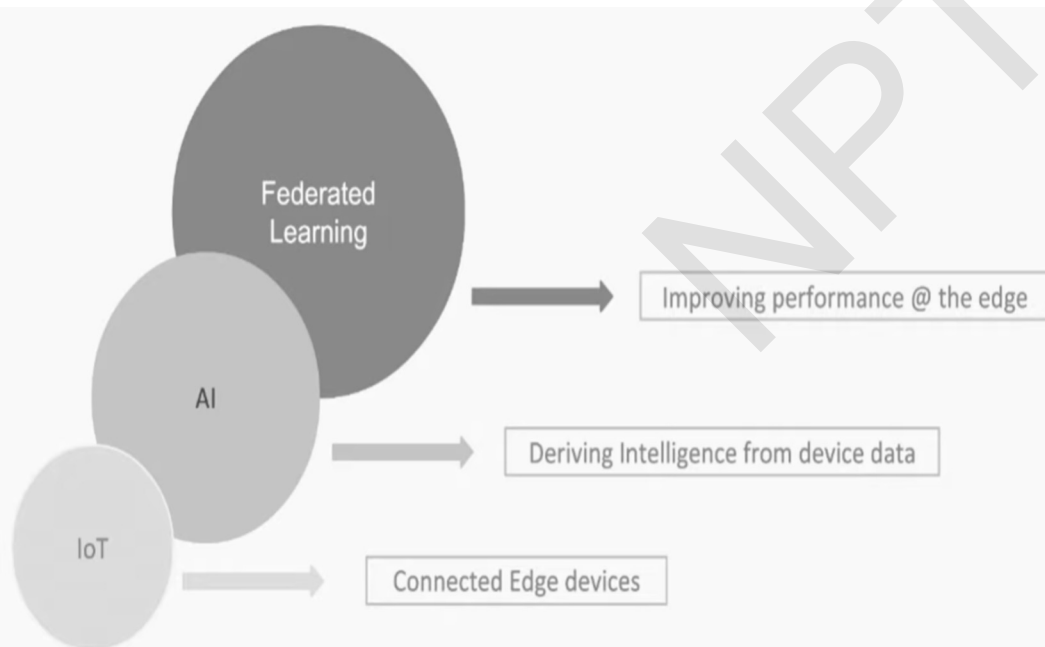
Federated Learning

Federated Learning: Distributed ML

- 2016: the term FL is first coined by Google researchers; 2020: more than 1,000 papers on FL in the first half of the year (compared to just 180 in 2018)¹
- We have already seen some real-world deployments by companies and researchers for large scale IOT devices
- Several open-source libraries are under development: PySyft, TensorFlow Federated, FATE, Flower, Substra...
- FL is highly multidisciplinary: it involves machine learning, numerical optimization, privacy & security, networks, systems, hardware...

Federated Learning: Decentralised data

- Federated Learning (FL) aims to collaboratively train a ML model while keeping the data decentralized
- Enabling devices to learn from each other (ML training is brought close
- A network of nodes and all nodes with their own central server but instead of sharing data with the central server, we share model we don't send data from node to server instead send our model to server



Federated Learning

Gradient Descent Procedure

The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.

$$\text{coefficient} = 0.0$$

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

$$\text{cost} = f(\text{coefficient}) \text{ or } \text{cost} = \text{evaluate}(f(\text{coefficient}))$$

We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

$$\text{delta} = \text{derivative}(\text{cost})$$

we can now update the coefficient values.

A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.

$$\text{coefficient} = \text{coefficient} - (\text{alpha} * \text{delta})$$

This process is repeated until the cost of the coefficients (cost) is 0.0 or close to 0. It does require you to know the gradient of your cost function or the function you are optimizing.

Gradient Descent Algorithm

Gradient Descent

- Gradient Descent is the most basic but most used optimization algorithm. It's used heavily in linear regression and classification algorithms. Backpropagation in neural networks, Federated Learning also uses a gradient descent algorithm.
- Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach a minima. Through backpropagation, the loss is transferred from one layer to another and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized.

$$\text{algorithm: } \vartheta = \vartheta - \alpha \cdot \nabla J(\vartheta)$$

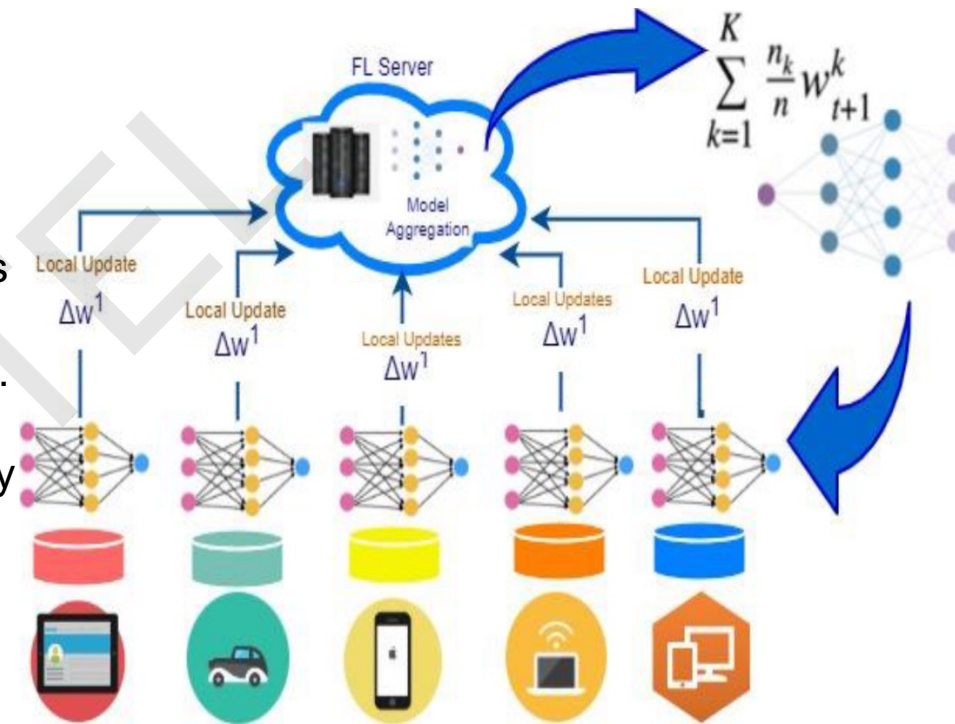
Advantages:

- Easy computation
- Easy to implement
- Easy to understand

The devices train the generic neural network model using the gradient descent algorithm, and the trained weights are sent back to the server. The server then takes the average of all such updates to return the final weights.

Edge Computing ML: FL

- FL is category of machine learning (ML) , which moves the processing over the edge nodes so that the clients' data can be maintained. This approach is not only a precise algorithm but also a design framework for edge computing.
- Federated learning is a method of ML that trains an ML algorithm with the local data samples distributed over multiple edge devices or servers without any exchange of data. This term was first introduced in 2016 by McMahan.
- Federated learning distributes deep learning by eliminating the necessity of pooling the data into a single place.
- In FL, the model is trained at different sites in numerous iterations. This method stands in contrary to other conventional techniques of ML, where the datasets are transferred to a single server and to more traditional decentralized techniques that undertake that local datasets



Edge Computing ML: FL

Finding the function: model training

- Given a training dataset containing n input-output pairs (x_i, y_i) , $i \in [1, n]$, the goal of deep learning model training is to find a set of parameters w , such that the average of $p(y_i)$ is maximized given x_i .

- That is,

$$\text{maximize} \quad \frac{1}{n} \sum_{i=1}^n p(y_i | x_i, w)$$

Which is equivalent to

$$\text{minimize} \quad \frac{1}{n} \sum_{i=1}^n -\log(p(y_i | x_i, w))$$

A basic component for loss function $l(x_i, y_i, w)$ given sample (x_i, y_i) :

Let $f_i(w) = l(x_i, y_i, w)$ denote the loss function.

Deep Learning model training

- Given one input sample pair (x_0, y_0) , the goal of deep learning model training is to find a set of parameters w , to maximize the probability of outputting y_0 given x_0 .

For a training dataset containing n samples (x_i, y_i) , $1 \leq i \leq n$, the training objective is:

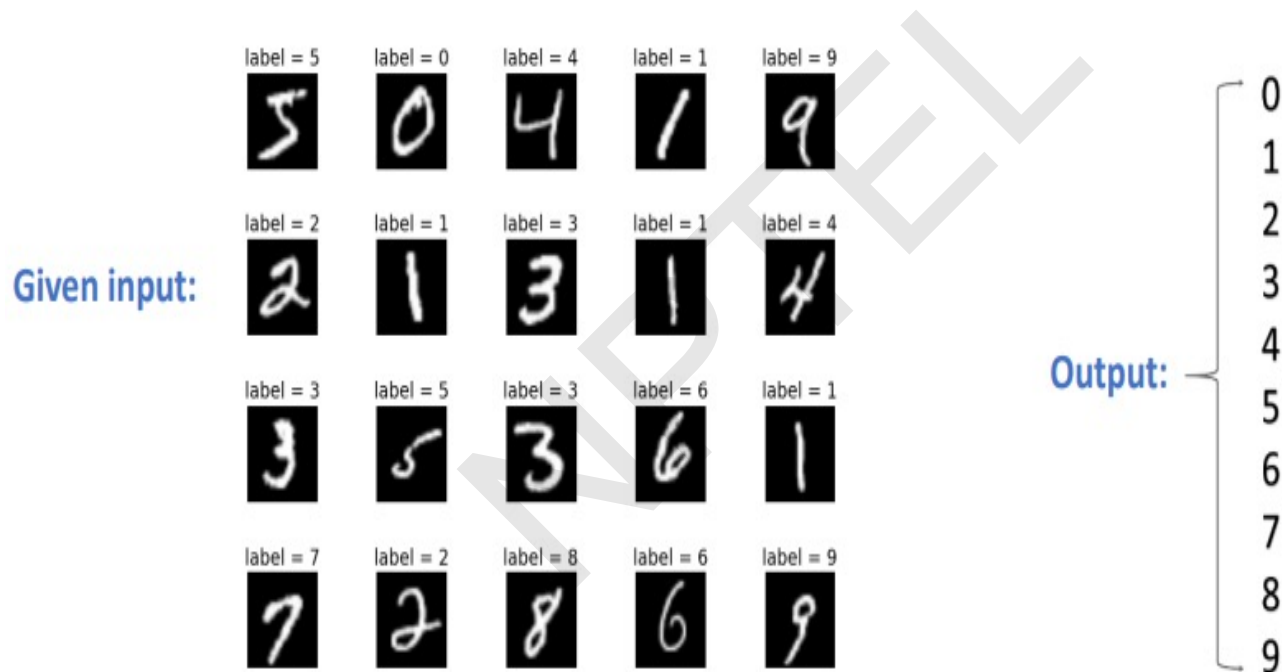
$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{where } f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n f_i(w)$$

$f_i(w) = l(x_i, y_i, w)$ is the loss of the prediction on example (x_i, y_i)

Edge Computing ML: FL

Finding the function: model training

- Given one input sample pair (x_0, y_0) , the goal of deep learning model training is to find a set of parameters w , to maximize the probability of outputting y_0 given x_0 .



How is this aggregation applied? FedAvg Algo

Federated learning – *FederatedAveraging (FedAvg)*

Algorithm 1 FederatedAveraging. The K clients are indexed by k ; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

initialize w_0

for each round $t = 1, 2, \dots$ do

$m \leftarrow \max(C \cdot K, 1)$

$S_t \leftarrow$ (random set of m clients)

for each client $k \in S_t$ in parallel do

$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

$w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): // Run on client k

$\mathcal{B} \leftarrow$ (split \mathcal{P}_k into batches of size B)

for each local epoch i from 1 to E do

for batch $b \in \mathcal{B}$ do

$w \leftarrow w - \eta \nabla \ell(w; b)$

return w to server

1. At first, a model is randomly initialized on the central server.
2. For each round t :
 - i. A random set of clients are chosen;
 - ii. Each client performs local gradient descent steps;
 - iii. The server aggregates model parameters submitted by the clients.

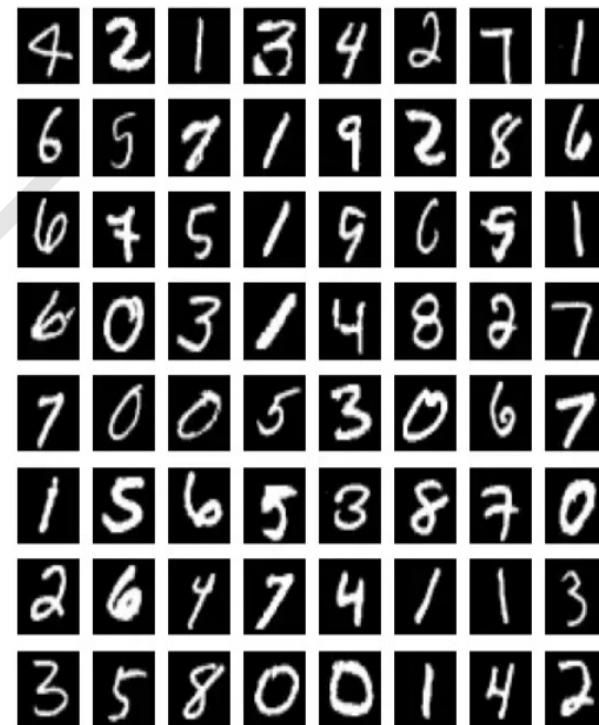
Example: FL with i.i.d.

In FL, each client trains its model decentral. In other words, the model training process is carried out separately for each client.

Only learned model parameters are sent to a trusted center to combine and feed the aggregated main model. Then the trusted center sent back the aggregated main model back to these clients, and this process is circulated.

A simple implementation with IID (independent and identically distributed) data to show how the parameters of hundreds of different models that are running on different nodes can be combined with the FedAvg method and whether this model will give a reasonable result.

This implementation was carried out on the MNIST Data set. The MNIST data set contains $28 * 28$ pixel grayscale images of numbers from 0 to 9.



Handwritten Digits from the MNIST dataset

Image Classifier using FedAvg

The MNIST data set does not contain each label equally. Therefore, to fulfill the IID requirement, the dataset was grouped, shuffled, and then distributed so that each node contains an equal number of each label.

A simple 2-layer model can be used for the classification process used FedAvg. Since the parameters of the main model and parameters of all local models in the nodes are randomly initialized, all these parameters will be different from each other, so the main model sends its parameters to the nodes before the training of local models in the nodes begins.

Nodes start to train their local models over their own data by using these parameters. Each node updates its parameters while training its own model. After the training process is completed, each node sends its parameters to the main model.

The main model takes the average of these parameters and sets them as its new weight parameters and passes them back to the nodes for the next iteration.

The above flow is for one iteration. This iteration can be repeated over and over to improve the performance of the main model.

The accuracy of the centralized model was calculated as approximately 98%. The accuracy of the main model obtained by FedAvg method started from 85% and improved to 94%.

Apple personalizes Siri without hoovering up data

The tech giant is using privacy-preserving machine learning to improve its voice assistant while keeping your data on your phone.

It relies primarily on a technique called federated learning.

It allows Apple to train different copies of a speaker recognition model across all its users' devices, using only the audio data available locally.

It then sends just the updated models back to a central server to be combined into a master model.

In this way, raw audio of users' Siri requests never leaves their iPhones and iPads, but the assistant continuously gets better at identifying the right speaker. In addition to federated learning, Apple also uses something called differential privacy to add a further layer of protection. The technique injects a small amount of noise into any raw data before it is fed into a local machine-learning model. The additional step makes it exceedingly difficult for malicious actors to reverse-engineer the original audio files from the trained model.

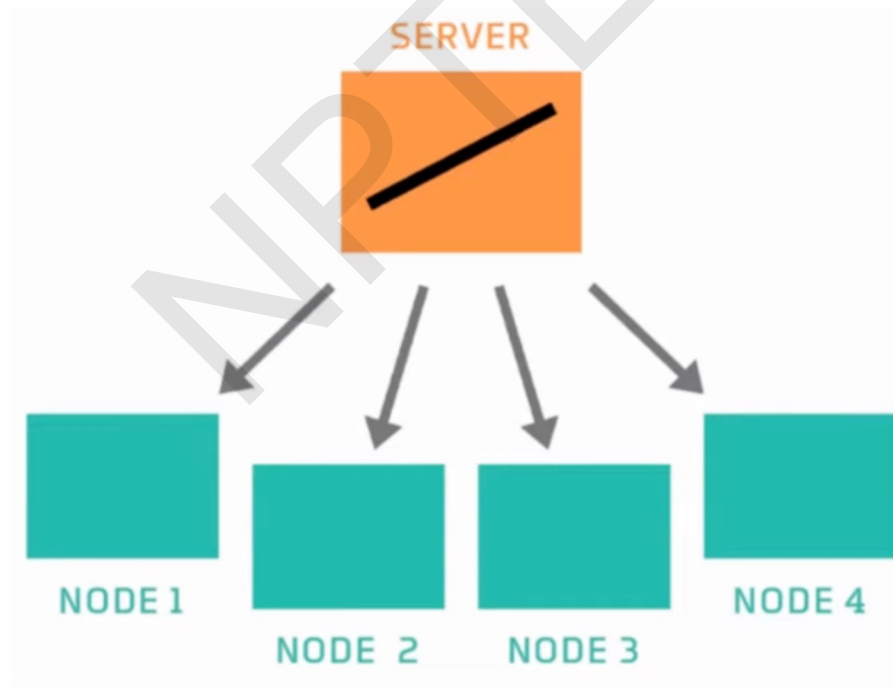


Siri is a perfect example of how Apple runs on AI. The voice-powered assistant is designed for continual, at-the-edge improvement

Federated Learning

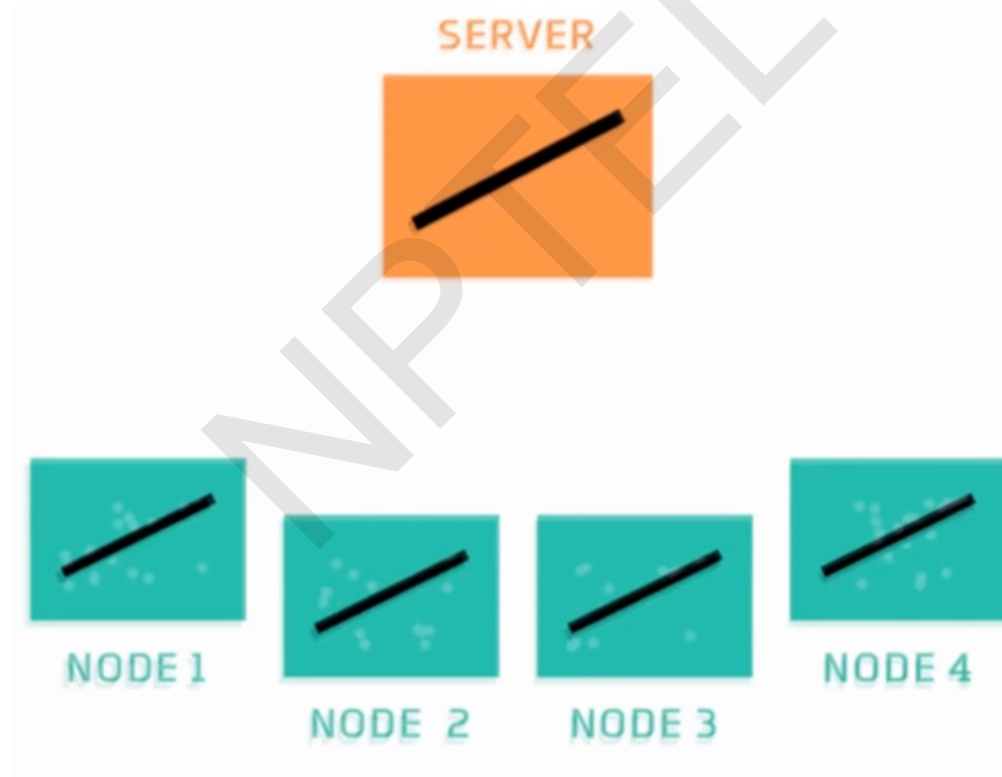
Federated Learning: Training

- There are connected devices let's say we have cluster of four IOT Devices from four of the IOT devices and there is one central server that has an untrained model.
- We will send a copy of the model to each of the node.
- Each node would receive a copy of that model.



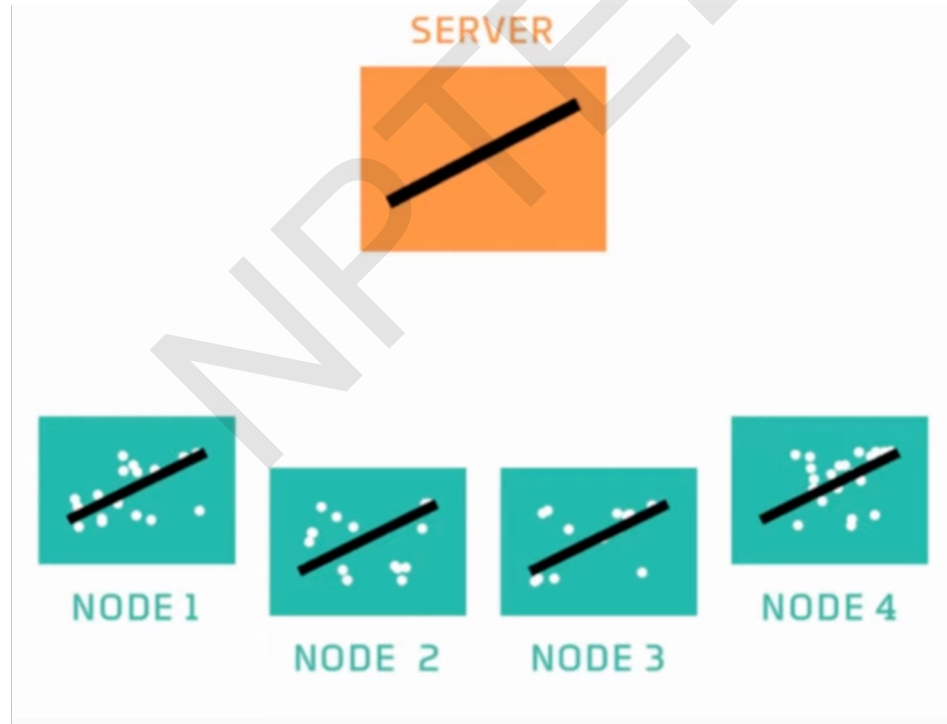
Federated Learning: Training

- Now all the nodes in the network has that untrained model that is received from the server.



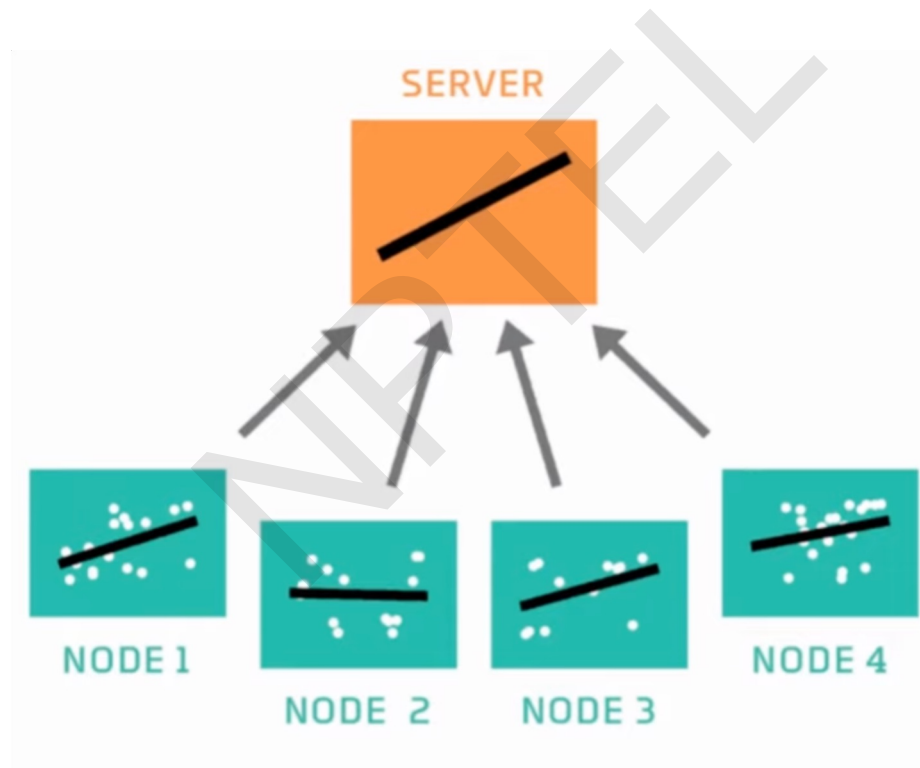
Federated Learning: Training

- In the next step, we are taking data from each node by taking data it doesn't mean that we are sharing data.
- Every node has its own data based on which it is going to train a model.



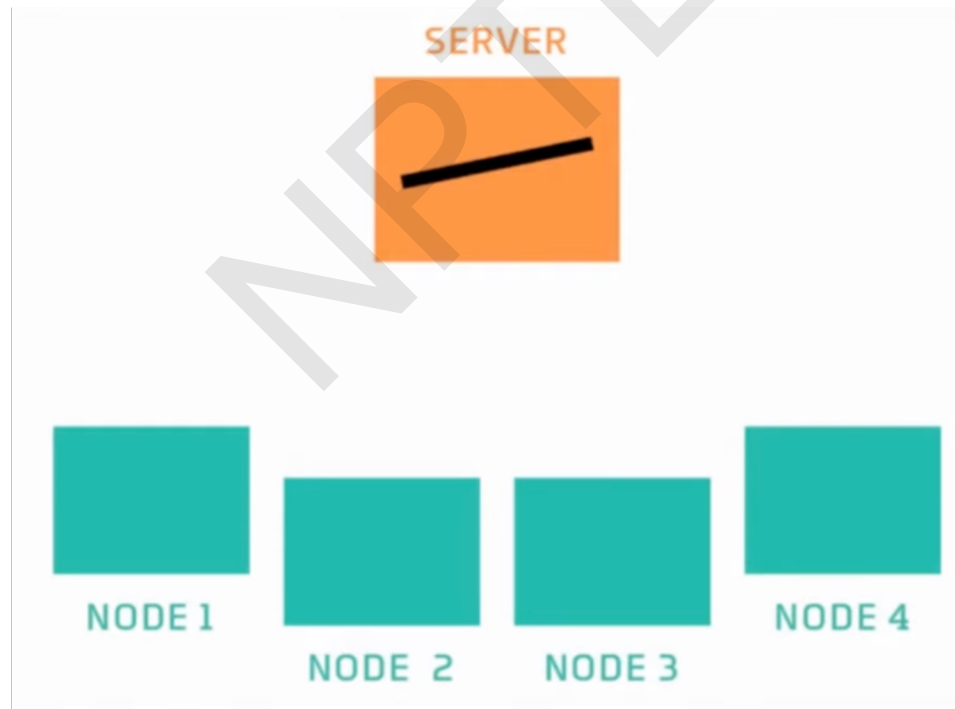
Federated Learning: Training

- Each node is training the model to fit the data that they have and it will train the model accordingly to its data.



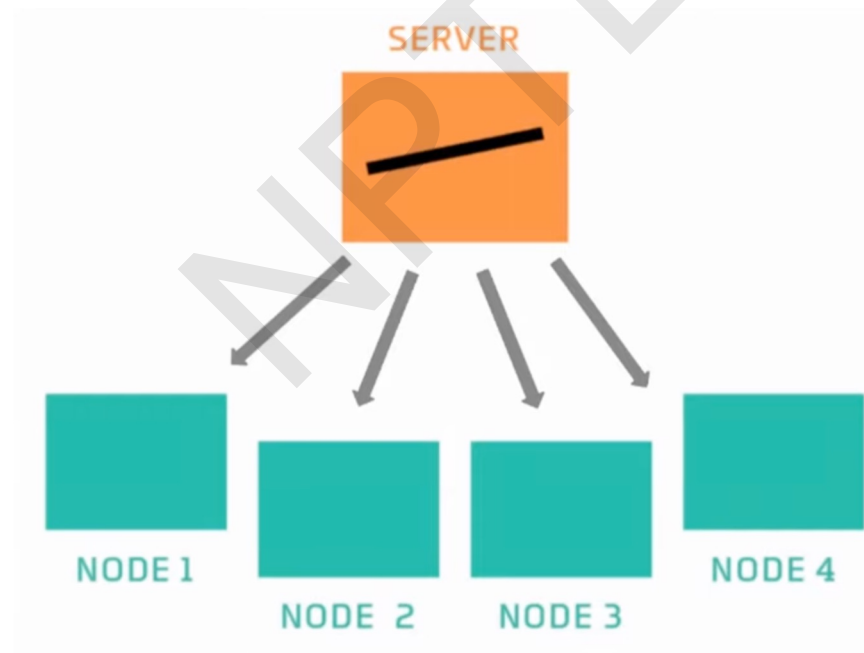
Federated Learning: Training

- Now the server would combine all these model received from each node by taking an average or it will aggregate all the models received from the nodes.
- Then the server will train that a central model, this model which is now trained by aggregating the models from each node. It captures the pattern in the training data on all the nodes it is an aggregated one



Federated Learning: Training

- Once the model is aggregated, the server will send the copy of the updated model back to the nodes.
- Everything is being achieved at the edge so no data sharing is done which means there is privacy preservation and also very less communication overhead.



Federated Learning: Challenges

Systems heterogeneity

- Size of data
- Computational power
- Network stability
- Local solver
- Learning rate

Expensive Communication

- Communication in the network can be slower than local computation by many order of magnitude.

Statistical Heterogeneity:

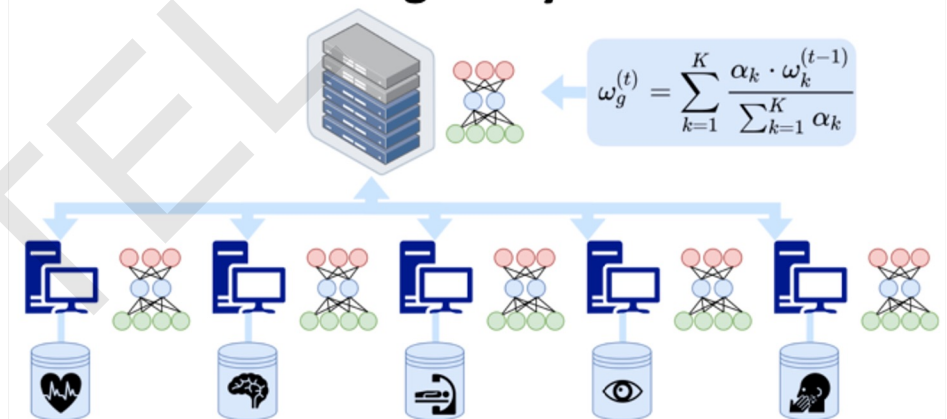


Fig. 1: Federated learning with non-iid data - The data has different distributions among clients.

Federated Learning: Challenges

- Dealing with Non-I.I.D. data** i.i.d (independent and identical distributed)
- Learning from non-i.i.d. data is difficult/slow because **each IOT device needs the model to go in a particular direction**
 - If data distributions are very different, learning a single model which performs well for all IOT devices may require a very large number of parameters
 - Another direction to deal with non-i.i.d. data is thus to **lift the requirement that the learned model should be the same for all IOT devices** (“one size fits all”)
 - Instead, we can allow each IOT k to learn a (potentially simpler) **personalized model θ_k** but **design the objective so as to enforce some kind of collaboration**
 - When local datasets are non-i.i.d., FedAvg suffers from **client drift**
 - To avoid this drift, one must use **fewer local updates and/or smaller learning rates**, which hurts convergence

Federated Learning: Challenges

Preserving Privacy

- ML models are susceptible to various attacks on data privacy
- **Membership inference attacks** try to infer the presence of a known individual in the training set, e.g., by exploiting the confidence in model predictions
- **Reconstruction attacks** try to infer some of the points used to train the model, e.g., by differencing attacks
- **Federated Learning offers an additional attack surface** because the server and/or other clients observe model updates (not only the final model)

Key differences with Distributed Learning

Data distribution

- In distributed learning, data is centrally stored (e.g., in a data center)
 - The main goal is just to train faster
 - We control how data is distributed across workers: usually, it is distributed uniformly at random across workers
- In FL, data is naturally distributed and generated locally
 - Data is not independent and identically distributed (non-i.i.d.), and it is imbalanced

Additional challenges that arise in FL

- Enforcing privacy constraints
- Dealing with the possibly limited reliability/availability of participants
- Achieving robustness against malicious parties

Federated Learning: Concerns

When to apply Federated Learning

- Data privacy needed
- Bandwidth and power consumptions are concerns
- High cost of data transfer

When NOT to apply Federated Learning

- When more data won't improve your model (construct a learning cure)
- When additional data is uncorrelated
- Performance is already at ceiling

Federated Learning: Applications

- **Predictive maintenance/industrial IOT**
- **Smartphones**
- **Healthcare (wearables, drug discovery, prognostics, etc.)**
- **Enterprise/corporate IT (chat, issue trackers, emails, etc.)**

Lecture Summary

- Market trend of IoT platform
- Why decentralized training is important?
- Understanding of Federated Learning
- Different issues with federated learning



THANK YOU!

ML for Autonomous Driving Car



Dr. Rajiv Misra, Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

After completion of this lecture you will know the following:

- Understanding of Autonomous Vehicles
- Role of Edge computing in Automotive Industry
- How ML is trained in Self-driving cars?
- Use Case of LSTM model for self-driving cars

Autonomous Vehicles: Introduction

Autonomous vehicles (AVs) have attracted a significant amount of interest in recent years. According to a report released by the US state Department of Transportation, “Self-Driving-Cars can reduce 90% of Traffic Deaths”.

A big chunk of major Automobile companies is trying to develop Self-Driving-Cars. Some big players are Tesla, Waymo, even Google is developing Self Driving Cars which has no presence in the automobile sector, have invested a huge amount of money, manpower and engineering capabilities in developing such systems.

Designing policies for an autonomous driving system is particularly challenging due to demanding performance requirements in terms of both making safe operational decisions and fast processing in real-time.



Edge Computing in Automotive

(AVs)

Historically, the adoption of computing (be it cloud or edge) and software in automotive has trailed the in-general adoption in other industries.

Cloud computing has been around for a while in many industries and many forms. But, vehicle telematics became one of the top use cases adopted in automotive somewhere in 2008.

Connected vehicles will continue to evolve at an exponential rate with V2V and V2X communication. This generates a large volume of data (every connected vehicle will generate data up to 4TB/day). How to handle, process, analyse the large amounts of data and make critical decisions quickly and efficiently?

Automobile makers are focused on leveraging edge computing to address these ever-evolving challenges. A group of cross-industry global players has formed the Automotive Edge Computing Consortium (AECC) to drive best practices for the convergence between the vehicle and computing ecosystem.

When driving a vehicle, milliseconds matter. Autonomous vehicles are no different, even though it may be your AI that drives them. $AI = data + compute$, and you want your compute to be as close to your data as possible. Enter edge computing.

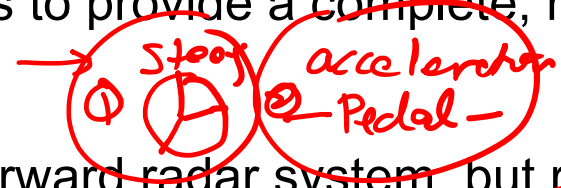
Edge Computing: Self-Driving Car Sensors

Given its real-time data processing capabilities, edge computing has naturally established itself as a pillar in autonomous vehicle technology. However, this data isn't generated by the computer but rather by the multitude of sensors that comprise an autonomous vehicle's peripheral "eyes" and "ears."

Sensor topology can vary widely amongst autonomous vehicles, even within the same sector.

Most self-driving sensors are fundamentally similar - they collect data about the world around them to help pilot the vehicle. For example, the Nuro vehicle contains cameras, radar, Lidar, and thermal cameras to provide a complete, multi-layered view of the vehicle's surroundings.

Currently, a Tesla utilize eight cameras, 12 , and a forward radar system, but rely much more heavily on camera visuals than Nuro vehicles. Google's Waymo Driver primarily relies on Lidar and uses cameras and radar sensors to help map the world around it.



Self-Driving Car: Requirements

Autonomous driving vehicles require two in-vehicle computing systems. One computer processes a large amount of sensed data and images collected by cameras and sensors. And a second computer to analyze processed image data and make intelligent and quick decisions for the vehicle.

- **Pre-processing collected data.** Autonomous vehicles have video cameras and a variety of sensors like ultrasonic, LiDAR, and radar to become aware of their surroundings and the internals of the vehicle. This data coming from different vehicle sources must be quickly processed through data aggregation and compression processes. An in-vehicle computer needs to have multiple I/O ports for receiving and sending data.
- **Secure network connectivity.** The in-vehicle computing solution must remain securely connected to the Internet to upload the pre-processed data to the cloud. In this case, having multiple wireless connections for redundancy and speed is crucial. High-speed connectivity is also vital for continuous deployments of vehicle updates or "push" updates like location, on-road conditions, and vehicle telematics.
- **High-performance computing.** Autonomous vehicles may generate approximately 1 GB of data every second. Gathering and sending a fraction of that data (for instance, 5 minutes of data) to a cloud-based server for analysis is impractical and quite challenging due to limited bandwidth and latency. Autonomous driving systems shouldn't always rely on network connectivity and cloud services for their data processing. Self-driving vehicles need real-time data processing to make crucial quick decisions according to their surroundings. In-vehicle edge computing is essential for reducing the need for network connectivity (offline decision-making) and for increasing decision-making accuracy.

How Machine Learning Trains AI in Self-Driving Cars

The value of the sensor data collected in all self-driving cars and vehicles depends on the compute methodologies downstream of the sensors themselves. In many ways, the most valuable intellectual property of companies like Tesla, Waymo, Aurora Innovations, and Nuro is the software and data infrastructure built to process and action the sensor data.

Today, all autonomous vehicles on the road utilize edge computing AI programs, which are often trained using data center machine learning models. Autonomous car machine learning models are only made possible by the incredible computing power of modern data centers capable of hundreds of petaflops.

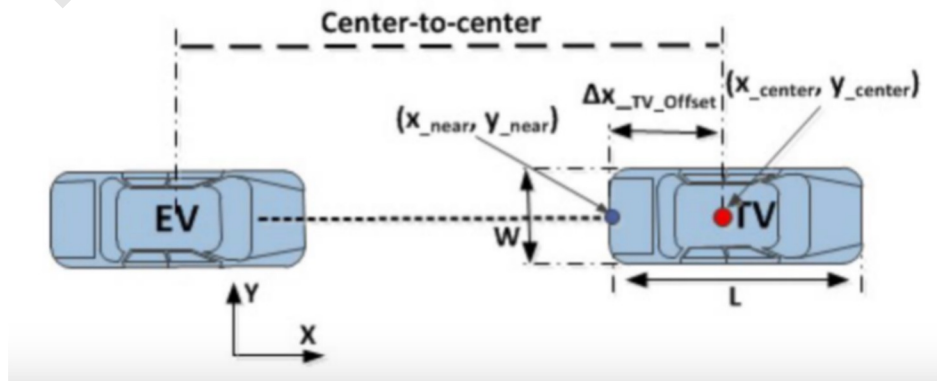
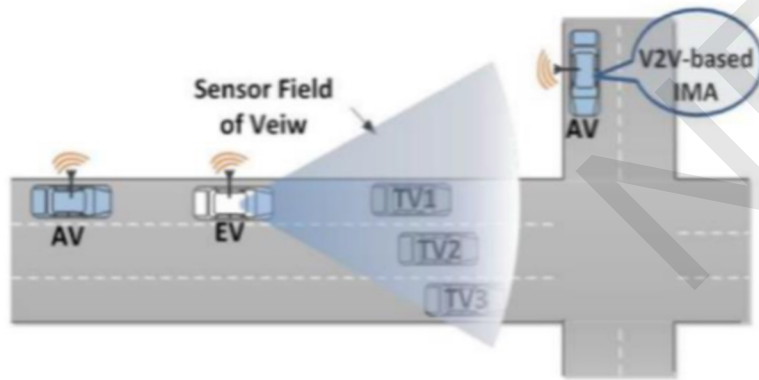
The computing requirements of these vast machine learning models well exceed the computing power of edge computers. Given this information, data centers are often used to form algorithms deployed for edge.

The problem of self-driving-car can be seen as a Regression Problem.

Training an AI algorithm is similar; it takes hundreds of compute hours on a high-power data center. Yet once that algorithm is learned, it can quickly and accurately utilize that algorithm using much less computing power.

Machine learning in autonomous driving

Kalman Filter, In real-life autonomous driving, the machine will deal with the same information from different sensors, such as Lidar, Radar, MEC signals and V2V Communications. This information will always have discrepancies with each other, and Kalman filter can help us to get a relatively reliable answer according to these two sets of information.



Machine learning in autonomous driving

Lidar, Radar, and Cameras ML is an important part of autonomous driving. A self-driving vehicle usually has multiple sensors, including cameras, lidar, and radar sensors. The machine learning module will tell the vehicle what to do with different information. For example, the car needs to stop when there were pedestrians, and the machine must be able to tell the difference between actual pedestrians and pictures of human. Additionally, camera sets cannot precisely measure distance or work at night. Lidar sensors usually emit high-frequency signals, and those high-frequency signals could be used for positioning and 3D modelling, being able to tell the difference between actual human and pictures of human. Radar is a low energy cost solution for positioning because the radio wave it emits is usually with low frequency. Low-frequency wave cannot depict the detailed 3D shape, but it is enough for positioning. However, cameras are still needed because neither lidar nor radar can identify colors.

Vehicle-to-Vehicle Communication. Communication (V2V) technology can increase the accuracy of autonomous driving prominently. When multiple cars are sharing their information, they can calibrate according to their relative positions.

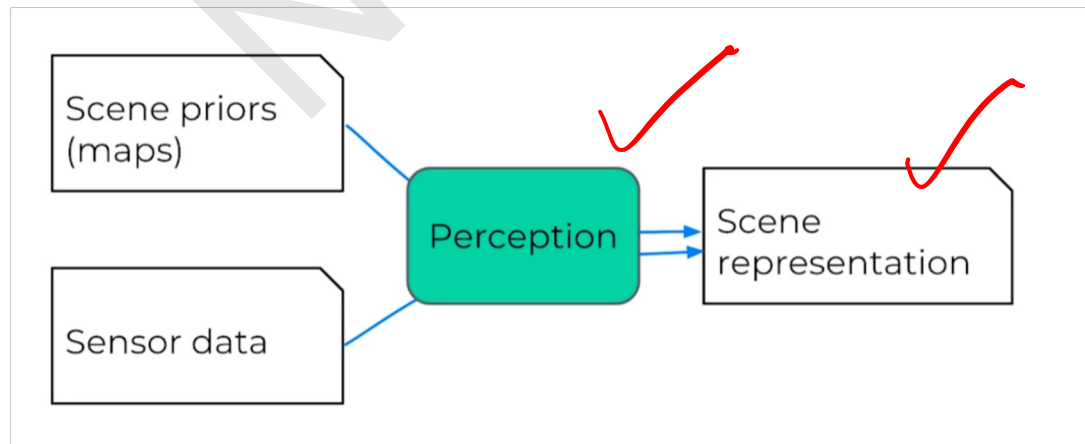
Key component of ML for self driving cars

Perception: a core element of what the self-driving car needs to build an understanding of the world around it using two major inputs:

- Scene Prior, and
- Sensor Data

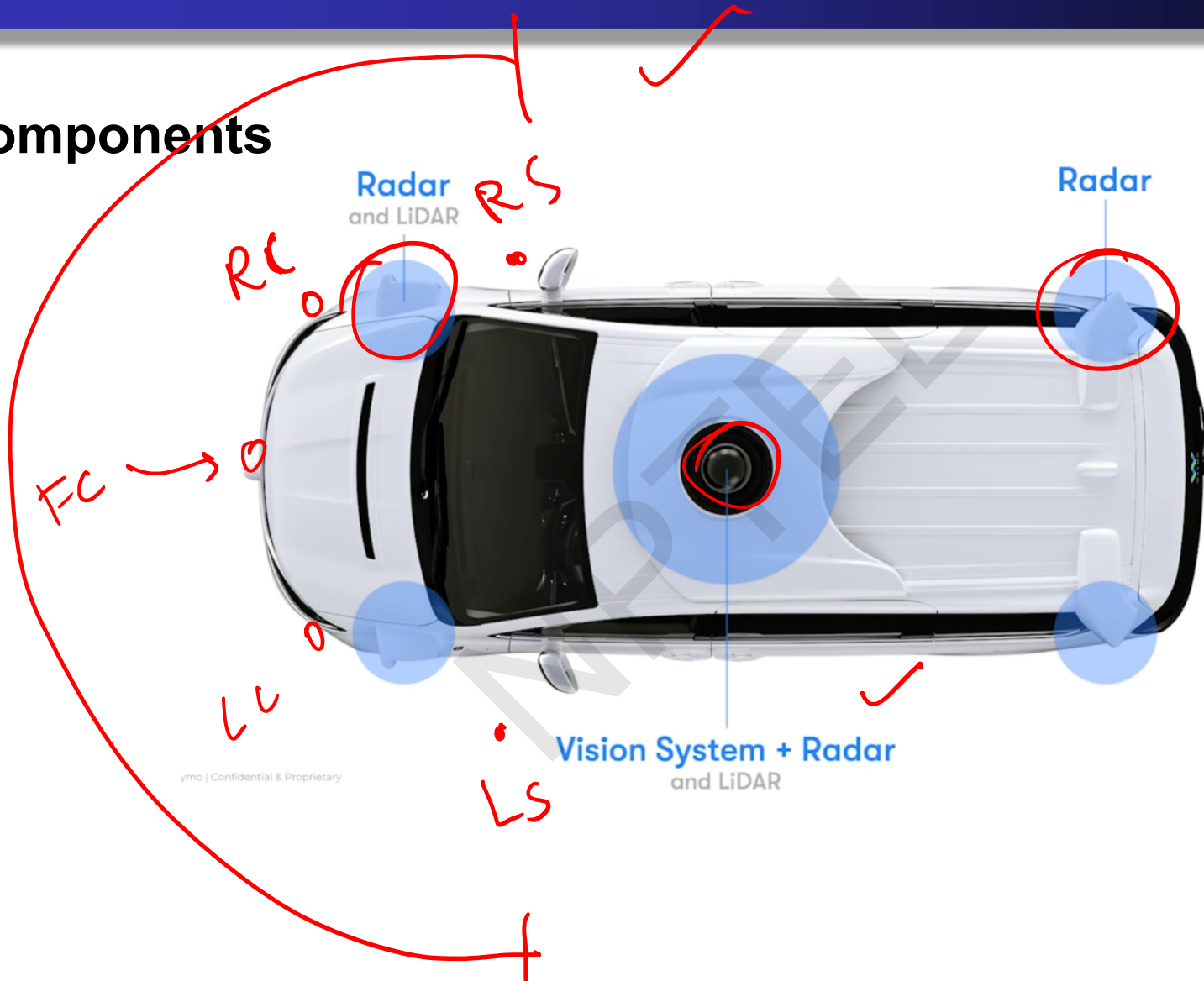
Scene Prior, is prior on the scene. For example it would be a little silly to recompute the actual location of the road, interconnectivity of the intersections of every intersection. Things you can pre-compute in advance and save your onboard computing for all the tasks that are more critical which is often referred to as the mapping exercise.

Sensor, the signal that's going to tell you what is not like, what you mapped and the things like traffic light right or green, where are the pedestrians and the cars what are you doing.



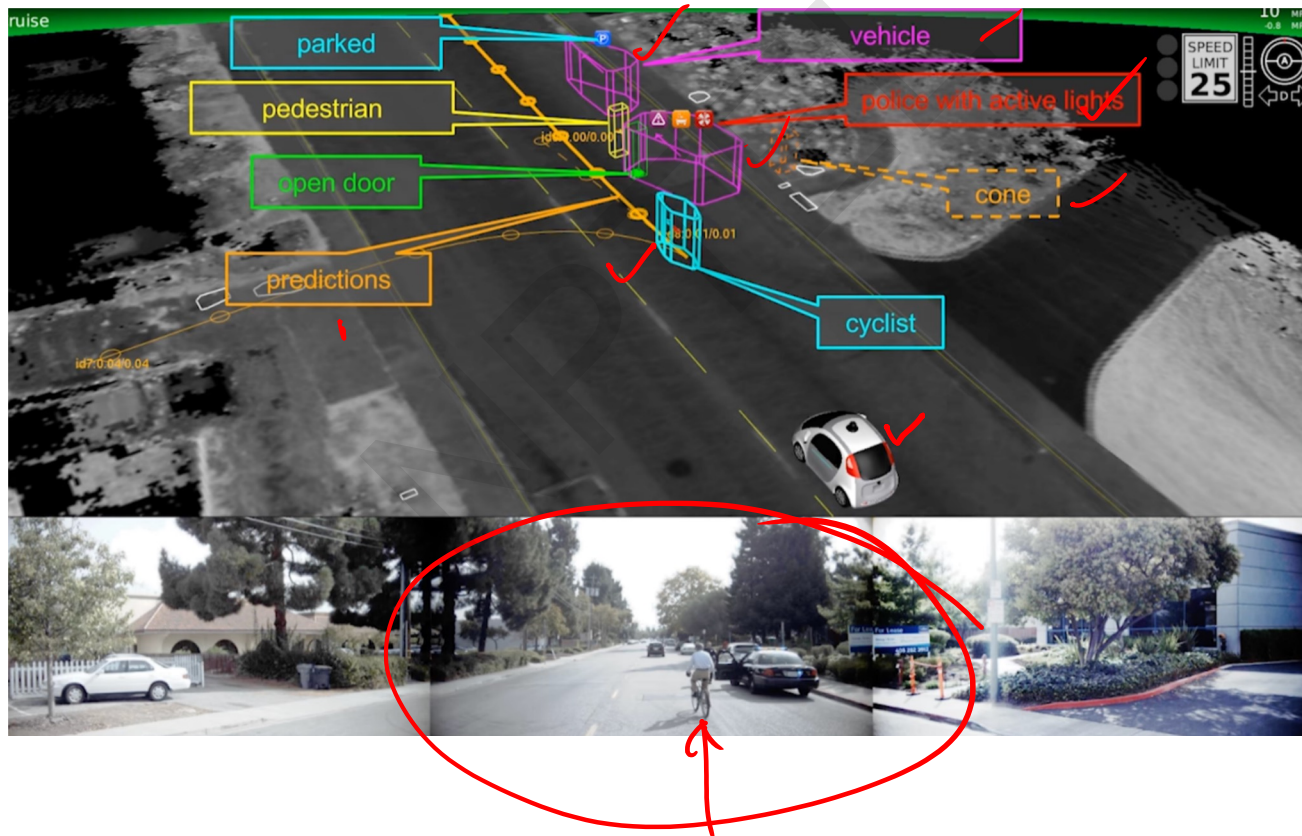
Key component of ML for self driving cars

Components



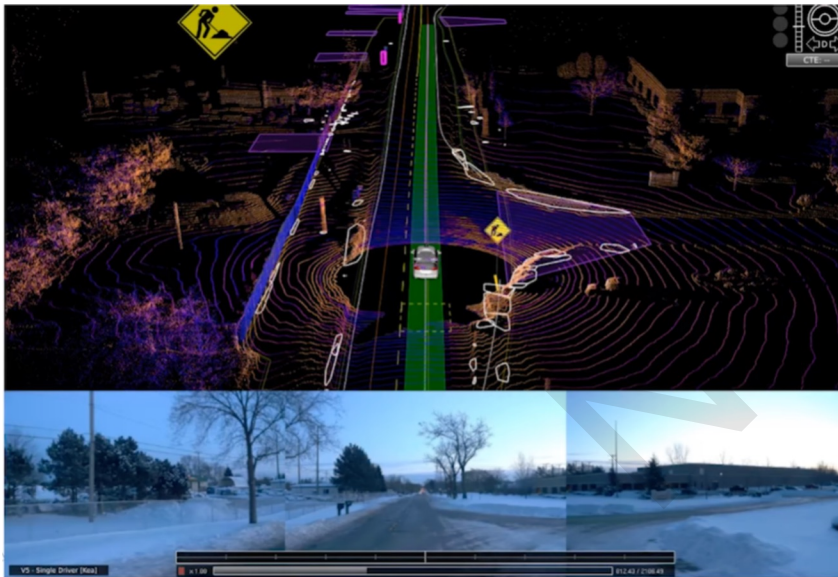
Key component of ML for self driving cars

Scene Representation



Key component of ML for self driving cars

Perform semantic ~~object~~ segmentation



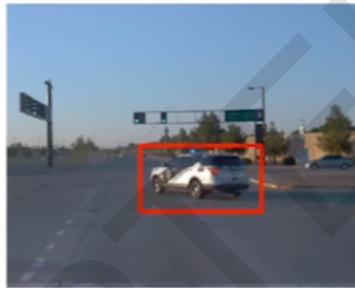
Key component of ML for self driving cars

Perform finer classification of objects

School Bus



Police Car



Fire Truck



Ambulance



Key component of ML for self driving cars

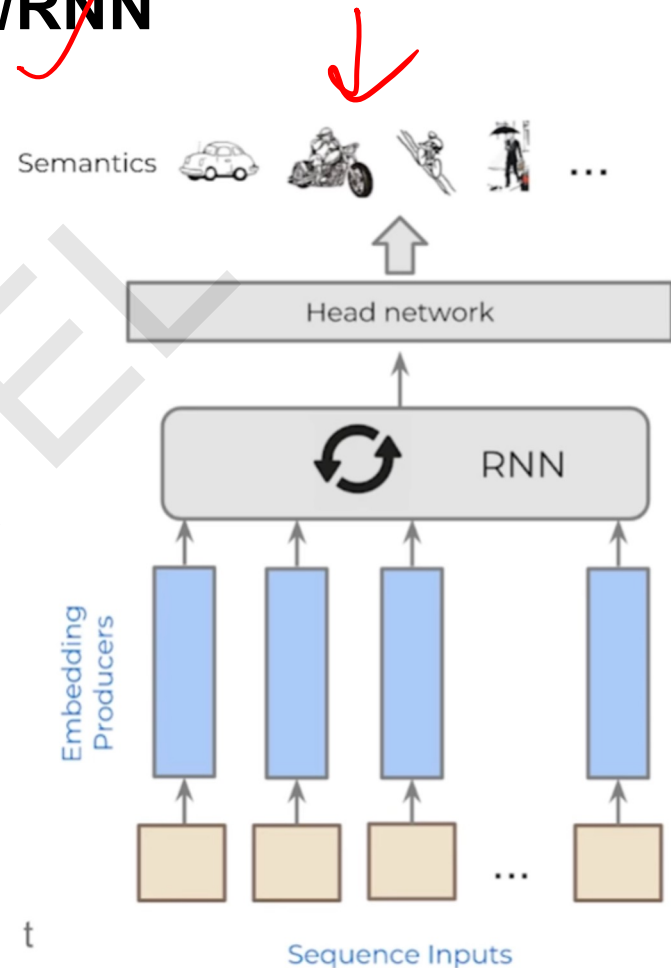
Time tracking using embeddings/RNN

Now the vector representations of different objects will be tracked over time.

A common technique that you can use is a recurrent neural networks that essentially are networks that will build a state that gets better and better as it gets more observation sequential observations of for the pattern.

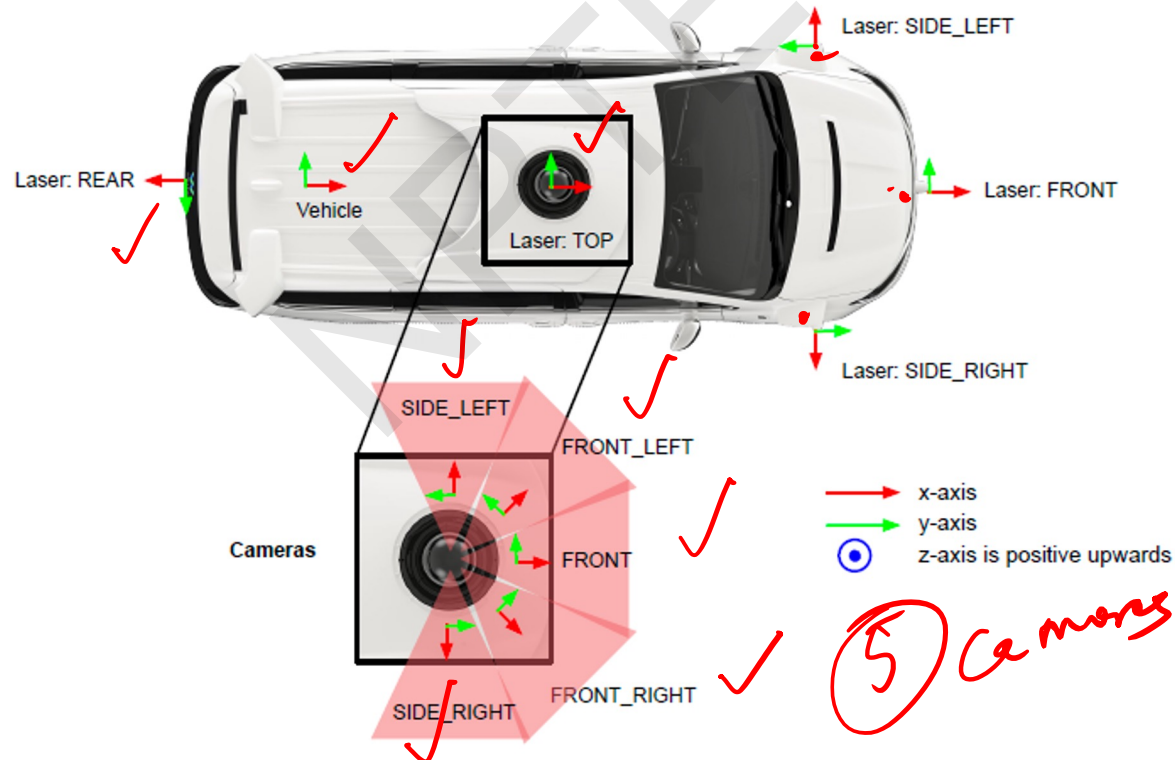
Once semantic representation and coding in an embedding for the pedestrian, the car under it and the model will track that over time and build a state of a good understanding of what's going on in the scene.

The vector representation combined with recurrent neural networks is a common technique to achieve this.



Data for training ML models in Self-Driving Cars

Waymo Open Dataset is the largest, richest and most diverse AV datasets ever published for academic research Sun et al. (2019). This dataset, collected from Waymo level-5 autonomous vehicles in various traffic conditions, comprise radar, lidar and camera data from 1000 20-second segments with labels. We will introduce details about the Waymo dataset, as well as how the data is preprocessed before being fed into several machine learning models.



Data for training ML models in Self-Driving Cars

Labels refer to kinematics and spatial parameters of objects, which are represented as bounding boxes. Specifically, one kind of labels, type, is classified into pedestrian, vehicle, unknown, sign and cyclist categories. Detailed information is provided for each label, among which we especially pay attention to the coordinates of the bounding boxes, velocities v , and accelerations a in the subsequent feature extraction step.

Coordinate Systems three coordinate systems are provided in this dataset: **global frame, vehicle frame, and sensor frame**. Some raw features are represented in unintended coordinate systems. In order to maintain consistency, it is crucial to transform data into the correct coordinate system. The dataset also provides vehicle pose VP , a 4×4 row matrix, to transform variables from one coordinate system to another.

Acceleration Computation Because one's instant acceleration of is not directly available in the dataset, the "ground truth" for training and evaluation needs to be computed by velocity differences.

Data for training ML models in Self-Driving Cars

Data Size: According to the data format, 1000 segments are packed into multiple compressed files (tars) with a size of 25 GB each. In our experiments, 32 training tars are used as the training set and 8 validation tars are used as the testing set. The total number of videos extracted from the segments is 45000.

Image embedding there are five cameras installed on the AV, facing towards front, front-left, front-right, side-left, and side-right respectively. These images reflect the time-series information of the moving vehicle with relatively smoother variation than numerical data, which helps to prevent spiky prediction between consecutive frames.



Use Case: LSTM model for self driving cars

Acceleration Prediction model

Basic Model with 12 Features, One of the straightforward ways to build the acceleration prediction model is to treat 12 basic features as the input of the model. The "encoder-decoder" architecture proposed for trajectory prediction in SS-LSTM is a suitable architecture for the acceleration prediction problem as the acceleration curve is a trajectory based on past experiences.

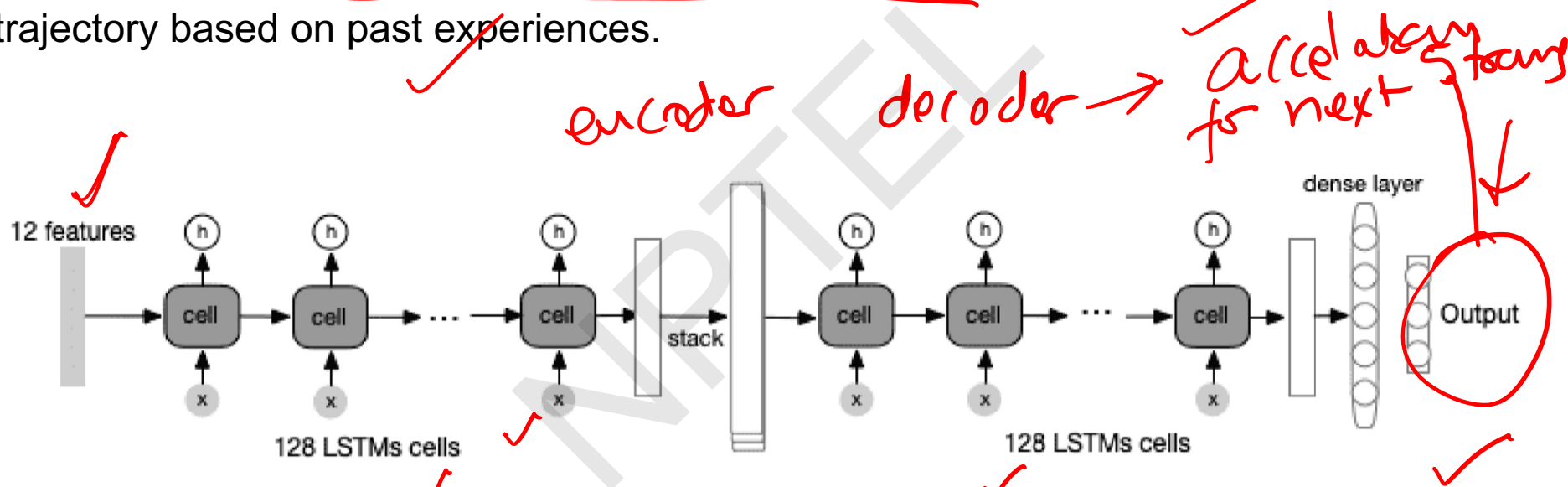


Figure 4: Given one video clip with a frame-length of 10, the input is the vector consists of 12 features from these 10 frames. The output is the acceleration for the next 5 frames starting from the end of the video clip. The "encoder" module contains 128 LSTM-cells and the "decoder" module contains 128 LSTM-cells

Use case: LSTM model for self driving cars

Advanced Model with Image Inputs, The architecture of such an advanced model is similar to the previous basic model. An "encoder-decoder" structure is maintained to learn the information hidden in the input features. The difference is that the front camera images are treated as additional inputs.

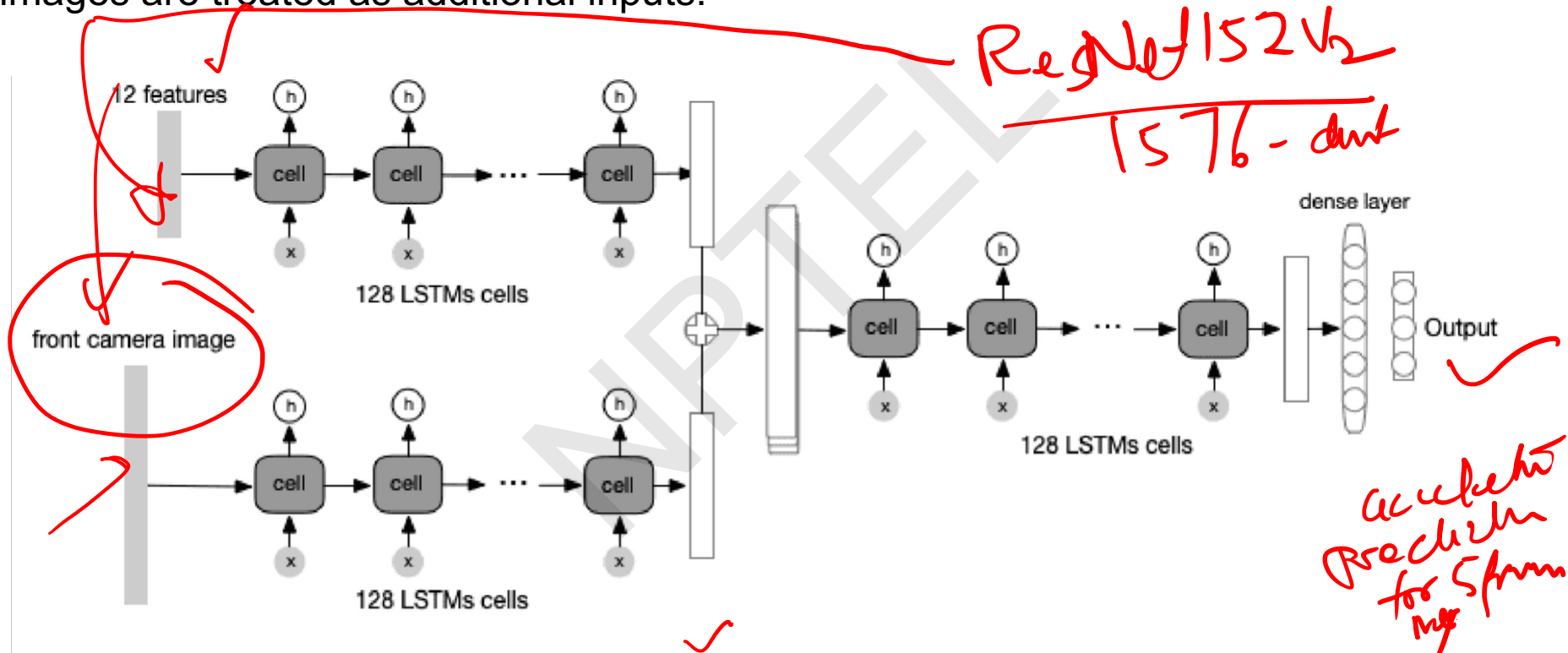


Figure 5: Noticed that the "image" input is actually a vector containing key image content. The first channel input is the 12 features for the observed 10 frames. The second channel input is the 1576-dimensional representation of front camera images from these 10 frames. Such representation is extracted from the second last output of a pre-trained Resnet152v2. The output is the acceleration for the future 5 frames. The "encoder" module contains 128 LSTM-cells and the "decoder" module contains 128 LSTM-cells

Use Case: LSTM model for self driving cars

Advanced Model with more Image Inputs

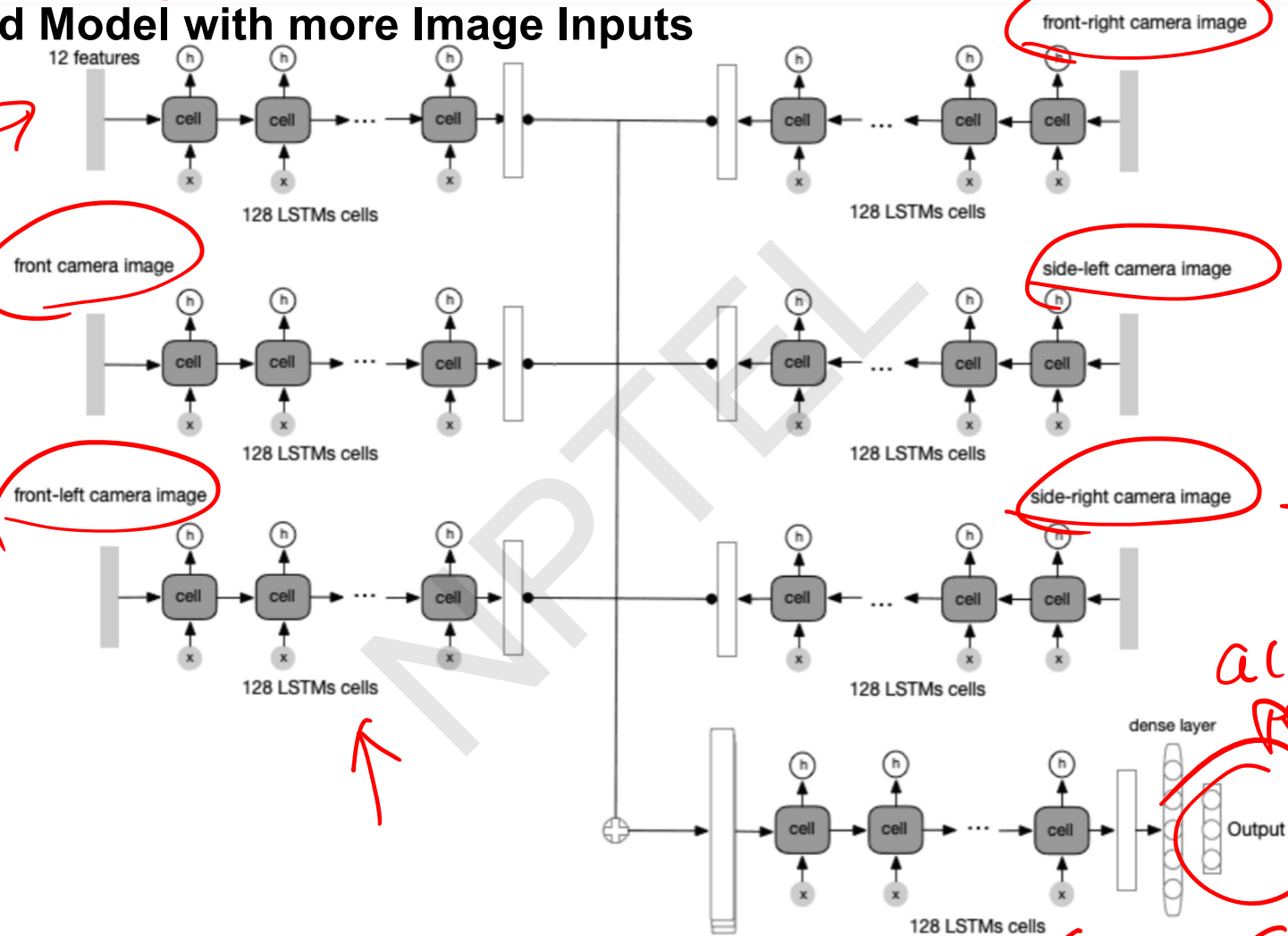


Figure 6: The first channel input is the 12 features for the observed 10 frames. The rest of the input is the 1576-dimensional representation of camera image from different views in the observed 10 frames. The output is the acceleration for the future 5 frames. All "encoder" modules contain 128 LSTM-cells and all "decoder" modules contain 128 LSTM-cells

Use Case: LSTM model for self driving cars

Comparison of results with other state-of-the-art methods

Models	MAE X	MAE Y
NN	0.4014	0.4312
CNN	0.3272	0.3123
NN+CNN	0.2985	0.2802
XGBoost	0.3502	0.3537
Light Gradient Boosting	0.3459	0.3444
Stacked Linear Regressor	0.3623	0.3222
LSTM with 12 features	0.3179	0.3088
LSTM with front camera	0.1379	0.1278
LSTM with all cameras	0.1327	0.1363

Future trend of autonomous driving

Like other intelligent industries of IIoT, autonomous driving is also reducing the total energy consumption. Gasoline has been the primary fuel for all kinds of vehicles, and natural gas storage only has about 52 years left, with current consumption levels. If the natural gas demands increased, natural gas could run out faster. So, the energy crisis is existing all the time.

First of all, the rise of autonomous driving cars can improve the energy efficiency of private-owned cars. Usually, an average family car can reach its maximum speed at about 200 to 250 km/h, but the city's usual speed limit is usually about 60km/h. That means the engine displacement of nowadays cars are mostly excessive. However, high engine displacement is necessary because faster cars are always safer because driver can overtake or change lane faster. If autonomous vehicles took the places of private-owned vehicles. In that case, it is pointless to use bigger and faster cars because autonomous driving cars are much more reliable than human drivers.

Future trend of autonomous driving

Secondly, auto-driving vehicles could reduce the natural gas dependency. As this paper mentioned before, smaller cars do not need potent energy resource, and electricity will be enough for most auto-driving vehicles. The popularization of auto-Driving cars is also an excellent opportunity to accept renewable energy over traditional energy sources, which will do good to the global climate as well.

Last but not least, when autonomous driving vehicles replaced private cars, parking issues will be solved, people will have bigger house and living areas because no garage is needed. There will be no traffic congestion as routes will be pre-scheduled to ensure efficiency. Long-distance deliverance will be more reliable because the auto-driving vehicle will never be tired.

Lecture Summary

- Different concepts of Autonomous Vehicles
- How Edge computing is important in Automotive Industry?
- How ML is trained in Self-driving cars?
- Use Case of LSTM model for self-driving cars



THANK YOU!