



## CLOUD COMPUTING

### Master-slave v/s p2p models

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

#### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

## Cloud Computing – Cloud Controller (recap)

---

- We have been studying Cloud computing as an ubiquitous, convenient, on demand network accessible shared pool of applications and configurable resources that can be rapidly provisioned and released with minimum management effort.
- These shared pool of applications and resources are supported by physical IT components, platforms and software which are housed in distributed data centers, accessible through the Internet.
- We studied different Application architectures, and different Service models like IaaS, PaaS, SaaS by which these applications and the resources hosted in a distributed infrastructure, are delivered to customers and users over the Internet as a service at different levels of abstraction.
- We also studied different deployment models like public, private or hybrid where the resources are deployed by service providers as a shared public platform or shared private platform or a combination of both.
- We also discussed technologies like virtualization supporting the abstraction, sharing and effective utilization of the resources, both compute and storage to provide reliable and available services as above to customers.

## Cloud Computing – Cloud Controller

---

- We also discussed that the physical resources are structured or organized and configured as distributed clustered systems in distributed Datacenters in different architectural models
- Most part of these discussions were focused on technologies, programming models, deployment models, application architectures and Data paths.
- Given that customer requests reaching the cloud physical resources will need to be managed, we will need to look at the control path for an application to run in the cloud environment in the next set of classes in Unit 4.
- We start with the context of the cloud infrastructure, structured or organized as distributed clustered systems e.g. as in master-slave, or peer-to-peer distributed system architectures *lecture 37*, challenges associated with these models in terms of unreliable communication *lecture 38*, different failures and their impact to reliability and availability, fault tolerance to be built to support availability *lecture 39*. There are various scenarios where the distributed transactions will need a consensus mechanism to ensure progress in the presence of certain errors which would be discussed in *lecture 40* as building consensus. There may be a need for handling scenarios if a leader node dies there is a need to electing a new leader, need for managing the provisioning and scheduling within the cloud environment, challenges of distributed Locking and Zoo keeper which will be discussed post these.

## Context of Cloud Control - Eucalyptus

Illustrative example of **Eucalyptus** and what is cloud control

**Eucalyptus** : Eucalyptus is an open-source software suite designed to build private/hybrid cloud computing environments compatible with Amazon Web Service's EC2 and S3

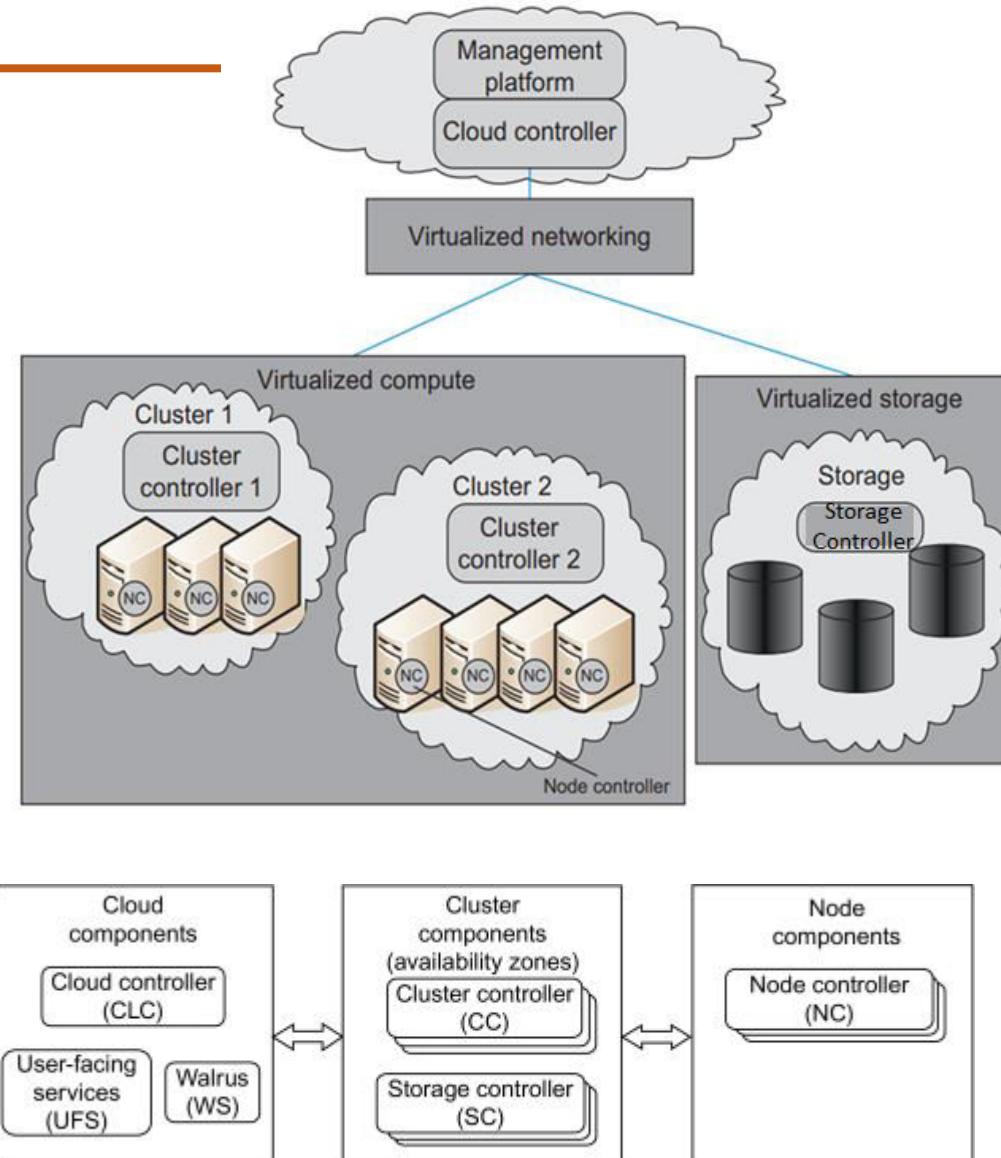
**UFS (User Facing Services)** Implements web service interfaces that handle the AWS-compatible APIs. This component accepts requests from command line clients or GUIs.

**Cloud Controller** : Provides high-level resource tracking and management. Only one CLC can exist in a Eucalyptus cloud

**Cluster Controller** : Manages node controllers and is responsible for deploying and managing VM instances

**Node Controller** : runs on machine that hosts VMs and interacts with both OS and hypervisors to maintain the lifecycle of instances running on each of the nodes

**Storage Controller** : Interfaces with various storage systems



## Context of Cloud Control – Eucalyptus (Cont.)

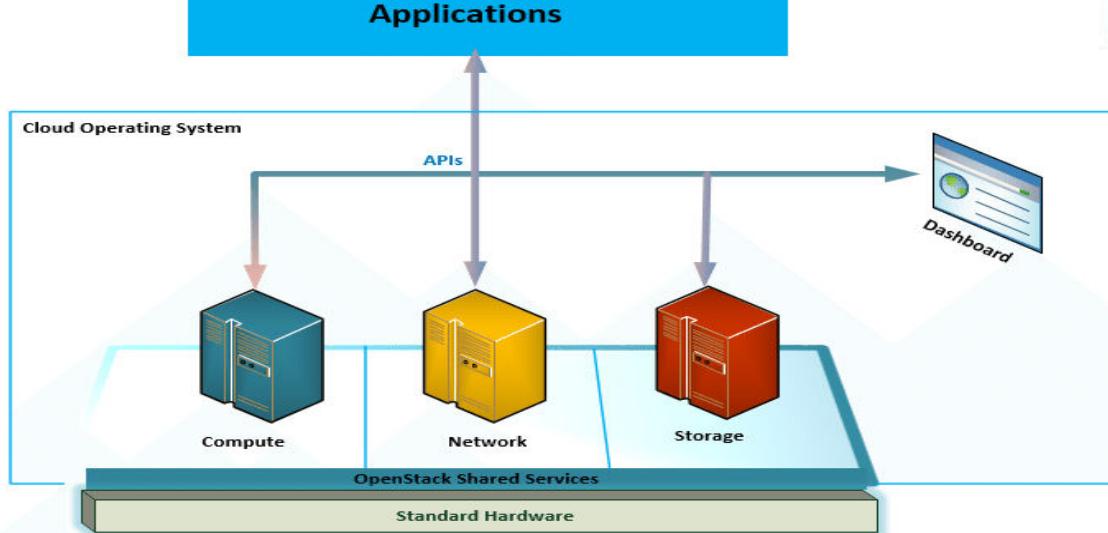
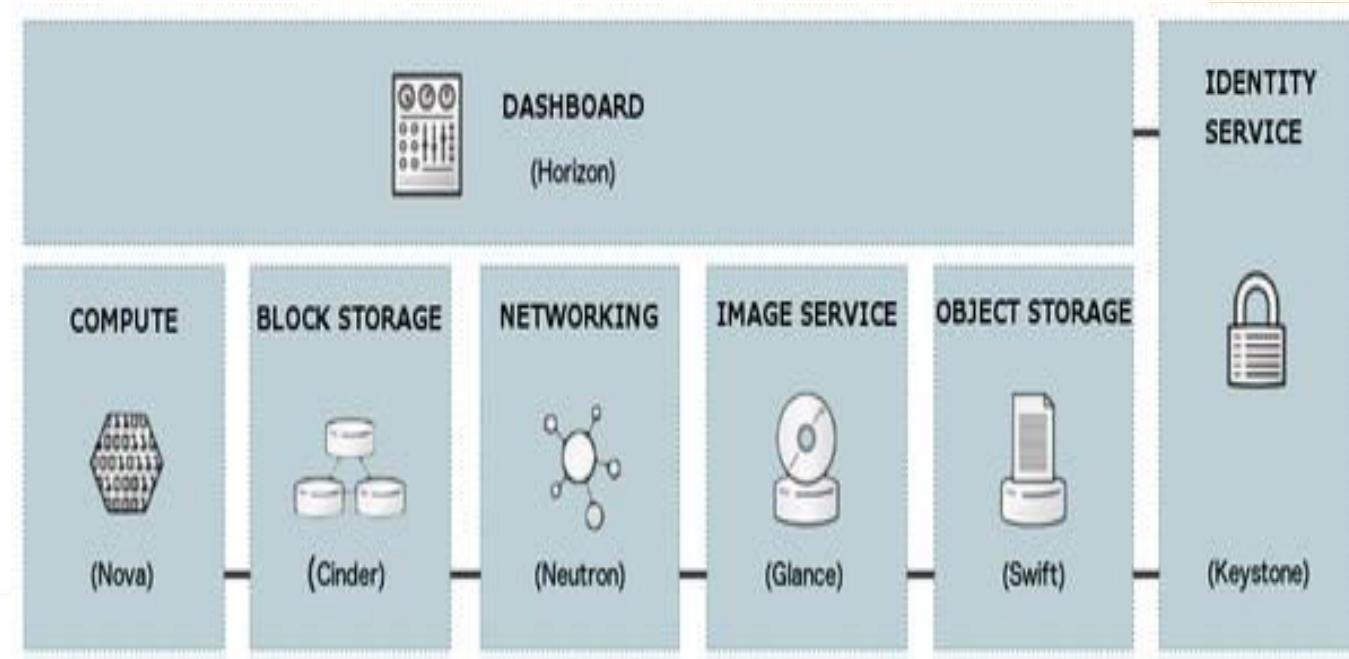
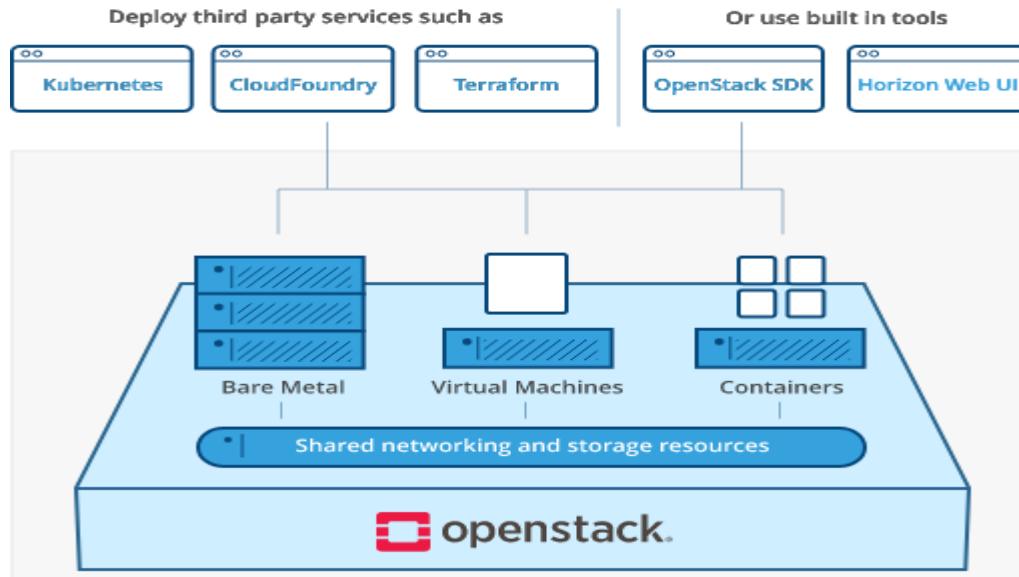
---

### Cloud Controller Module

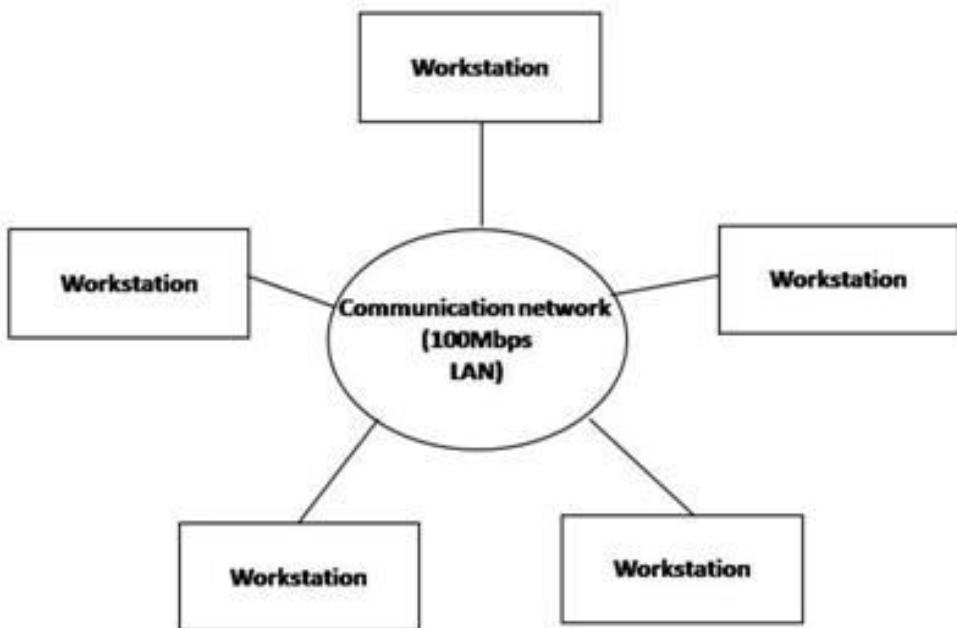
- The key interface for users and administrators is the Cloud Controller (CLC), which queries the different nodes and makes broad decisions about the virtualization setup, and executes those decisions through cluster controllers and node controllers.
- The Cloud Controller is a set of web services that provides (a) system-wide arbitration of resource allocation, (b) governing of persistent user and system data, and (c) handling authentication and protocol translation for user-visible interfaces.
- A System Resource State (SRS) is maintained with data coming in from CC (Cluster Controller) and NCs (Node Controller) of the system.
- When a user's request arrives, the SRS is used to make an admission control decision for the request based on service-level expectation.
- To create a VM, the resources are first reserved in the SRS and the request is sent downstream to CC and NC. Once the request is satisfied, the resource is committed in the SRS, else rolled back on failure.

# CLOUD COMPUTING

## Context of Cloud Control - OpenStack



A **distributed system**, also known as **distributed computing**, is a **system** with multiple components located on different machines and communicate and coordinate actions in order to appear as a single coherent **system** to the end-user



- Depending on the needs of the application, the system components which are part of the cloud infrastructure can be structured/organized and configured and classified into as three different Distributed System Models

### 1. Architectural Models

- System Architecture
  - This indicates how the components of a distributed system are placed across multiple machines
  - How the responsibilities are distributed across system components
    - E.g. P2P Model or Client Server Model
- Software architecture
  - This indicates the logical organization of software components and their interactions/independence
  - Focusses about the components
    - E.g. 3 Tier Architecture

### 2. Interaction Models

- How do we handle time? Are there time limits on process execution, message delivery, and clock drifts?
- Ex: Synchronous distributed systems, Asynchronous distributed systems

### 3. Fault Models

- What kind of faults can occur and what are their effects?
- Ex: Omission faults, Arbitrary faults, Timing faults

Two main architectures:

- Peer-to-Peer architecture

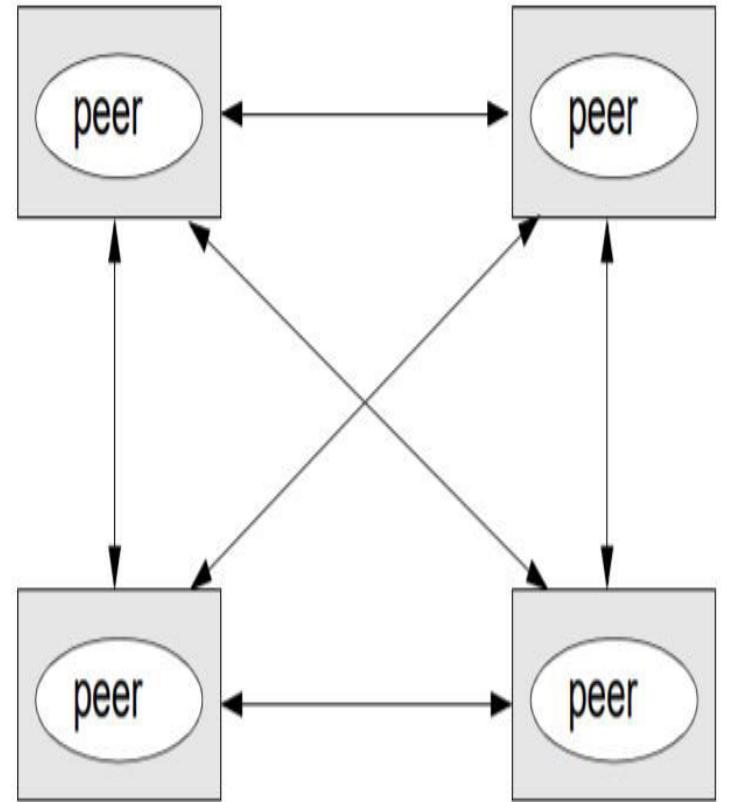
Roles of entities are *symmetric*

- Client/Server or Master-Slave architecture

Roles of entities are asymmetric

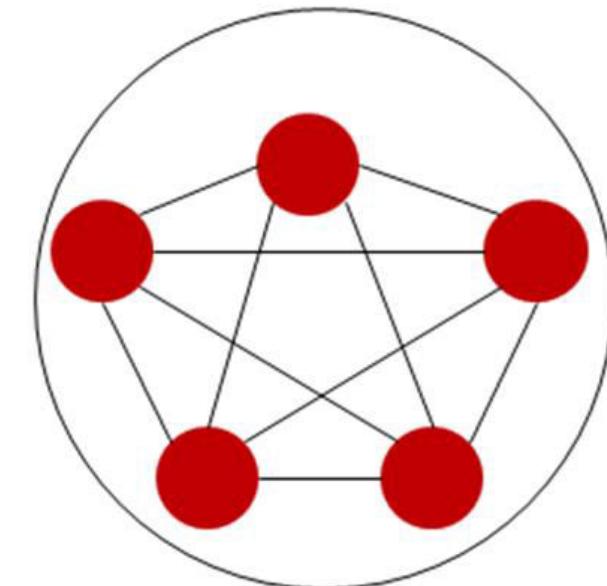
### What does a Peer-to-Peer System mean?

- In a P2P network, every node (peer) **acts as both a client and server**.
- Peers act autonomously to join or leave the network.
- The system is self-organizing nodes (recognize and create relatively stable connections to other nodes with similar interests/requirements/capabilities) with distributed control. In other words, no peer machine has a global view of the entire P2P system
- Processing and communication loads for access to objects are distributed across many computers and access links.
- This is the most general and flexible model but securing the overall system would be more challenging as each of the peer would have their own data



## What does a Peer-to-Peer System mean? (Cont.)

- A model of communication where every node in the network acts alike.
  - As opposed to the Client-Server model, where one node provides services and other nodes use the services.
- All nodes are equal (no hierarchy)
  - No Single-Point-of-Failure (SPOF)
- A central coordinator is not needed
  - But, decision making becomes harder
- The underlying system can scale out indefinitely
  - In principle, no performance bottleneck



- Peers can interact directly, forming groups and sharing contents (or offering services to each other)
  - At least one peer should share the data, and this peer should be accessible
  - Popular data will be highly available (it will be shared by many)
  - Unpopular data might eventually disappear and become unavailable (as more users/peers stop sharing them)
- Peers can form a virtual overlay network on top of a physical network topology
  - Logical paths do not usually match physical paths (i.e., higher latency)
  - Each peer plays a role in routing traffic through the overlay network

## P2P Model Advantages and Limitations

---

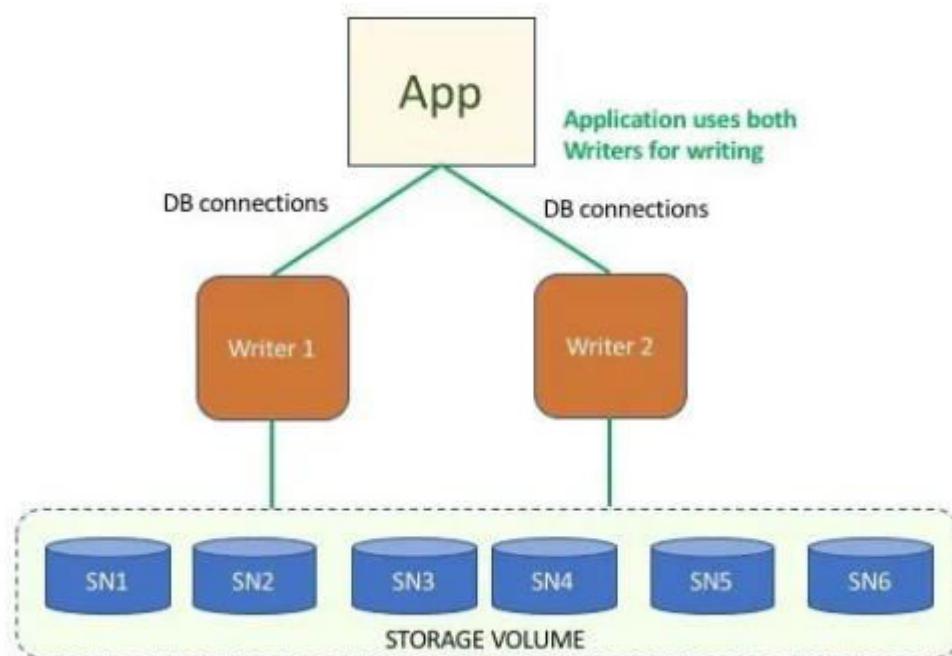
Although P2P model

- Is Scalable as it is possible to add more peers to the system and scale to larger networks.
- Does not need the individual nodes to be aware of the global state
- Is Resilient to faults and attacks, since few of their elements are critical for the delivery of service and the abundance of resources can support a high degree of replication.
- They are highly decentralized, with the individual systems not needing to be aware of the global state and thus limiting P2P systems in terms of the ability to be managed effectively and provide security as required by various applications

This leads to architectures where servers in the cloud are in a single administrative domain and have a centralized management as with Client Server architecture

## Where is P2P used in real world Cloud Systems – Amazon Aurora

- Amazon Aurora – RDMS build for cloud with SQL compatibility
- Multi-master clusters in Aurora consist of DB instances with both read/write capacity
- Multi master clusters are however unconventional since they require different availability characteristics and different procedures for monitoring

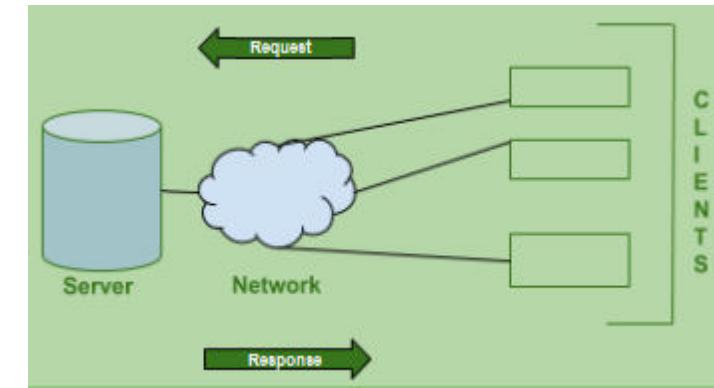
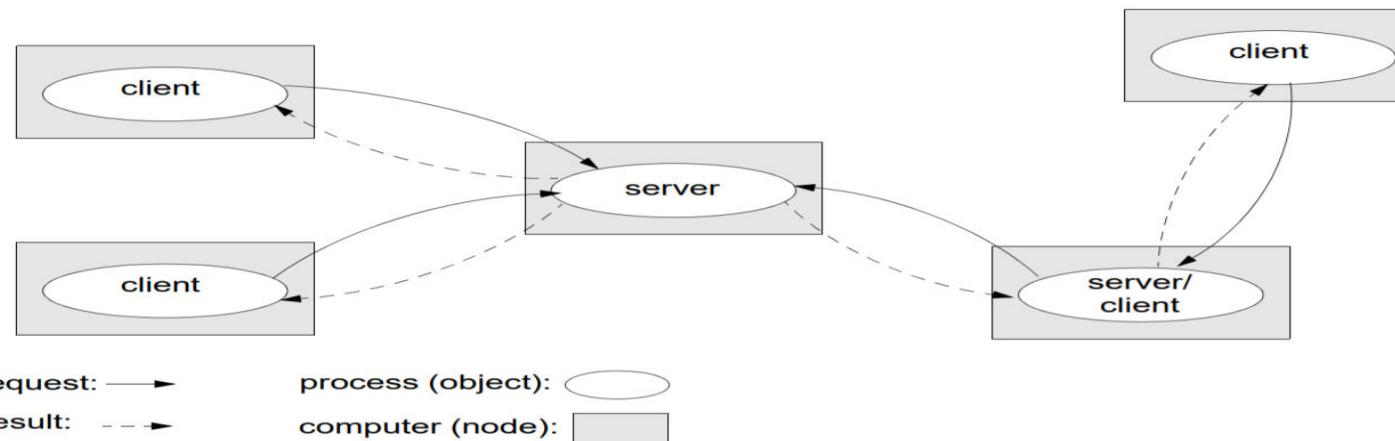


## Distributed System - Client-Server Model (Recap)

### What is the Client-Server model?

The system is structured where a set of machines called **servers** (which are performing some process), offer services to another set of machines called **clients** for their needs.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC).
- The client asks the server for a service, the server does the work and returns a result or an error code if the required work could not be done
- This organization by its structure distributes the functionality across different machines
- A Server can provide services for more than one Client and a client can request services from several servers on the network without regard to the location or the physical characteristics of the computer in which the Server process resides.



- A master-slave architecture can be characterized as follows:
  - 1) Nodes are *unequal* (there is a hierarchy)
    - Vulnerable to *Single-Point-of-Failure* (SPOF)
  - 2) The master acts as a *central coordinator*
    - Decision making becomes easy
  - 3) The underlying system *cannot scale out* indefinitely
    - The master can render a performance bottleneck as the number of workers is increased

## Where is master/slave architecture is used in the real world?

- Most predominantly undertaken architecture
- Due to ease of monitoring, to reduce consistency and linearizability issues
- Shortcoming: Compromise on availability due to single point of failure

### Example: Map Reduce

- Hadoop ecosystem
- Single Master node –
  - Assigns work to multiple worker nodes
  - Monitors jobs assigned to workers
  - Redistributions tasks in case of worker failure

- [An Introduction to Client Server Computing \(aagasc.edu.in\)](http://aagasc.edu.in)
- [https://eclass.uoa.gr/modules/document/file.php/D416/Cloud ComputingTheoryAndPractice.pdf\(P2P\)](https://eclass.uoa.gr/modules/document/file.php/D416/Cloud%20ComputingTheoryAndPractice.pdf)
- “Moving to the clouds: Developing Apps in the new world of cloud computing”, Dinkar Sitaram and Geetha Manjunath. Syngress, 2011.



# THANK YOU

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



## CLOUD COMPUTING

### Unreliable Communication

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

#### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

## Trouble with Distributed Systems

---

- In a cloud environment where Businesses are hosting their services in third-party provisioned distributed cloud infrastructure, its mandatory for the cloud systems to be reliable and robust.
- Thus the cloud controller will need to constantly monitor the infrastructure which would include the distributed systems, their communication mechanism and their storage to ensure that these resources are reliable and robust.
- These large-scale cloud environment has a lot of commodity hardware which would mean hardware failures and software bugs, can be expected to occur relatively frequently.
- These hardware failures can trigger other failures leading to an avalanche of failures that can lead to significant outages.
- Controlling these cloud resources thus would need understanding and using reliability as the metric, and monitoring the cloud components for different categories or types of failures or for unacceptable performances.
- Monitoring would also need that the challenges related to the communication between the nodes in the distributed environment have to be been factored in, along with considerations to different categories of failures and approaches to monitor or detect them and approaches to make the environment to be more reliable and robust.

## Trouble with Distributed Systems

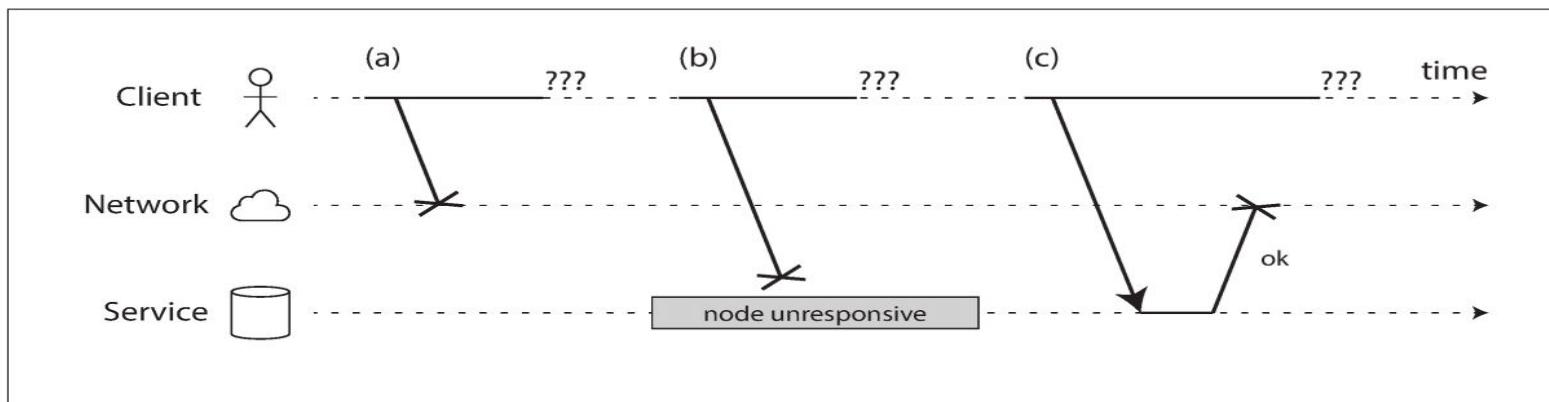
---

- This challenge is compounded within a distributed system where,
  - There may well be some parts of the system that are broken in some unpredictable way, even though other parts of the system are working fine. (known as a partial failures).
- Given that most distributed systems have multiple nodes in the network, it gets hard to predict failures and thus these partial failures get to be *non-deterministic*:
- You may not even *know* whether something succeeded or not, as the time it takes for a message to travel across a network is also non-deterministic!
- This **non-determinism and possibility of partial failures is what makes distributed systems hard to work with**
- This will mean having mechanisms and components for monitoring for the different kind of faults which can occur, build fault models to have ways of handling these partial or full failures, have mechanisms to tolerate some the faults to make these distributed systems to be ***reliable and robust***.

## Trouble with Distributed Systems

Consider that you are sending a request and expect a response, where many things could go wrong:

1. Your request may have been lost (perhaps someone unplugged a network cable).
2. Your request may be waiting in a queue and will be delivered later (perhaps the network or the recipient is overloaded).
3. The remote node may have failed (perhaps it crashed or it was powered down).
4. The remote node may have temporarily stopped responding, but it will start responding again later.
5. The remote node may have processed your request, but the response has been lost on the network (perhaps a network switch has been misconfigured).
6. The remote node may have processed your request, but the response has been delayed and will be delivered later



## What do we mean by Reliability?

---

- It's the probability that the system will meet certain performance standards and yield correct output for a specific time
- It's the ability for a system to remain available over a period of time to support the requirements of the applications under the slated conditions
- The application performs the function that the user expected ***without errors, disruptions, or significant reduction in performance*** under the expected load and data volume
- High reliability is extremely important in critical applications. There are many applications hosted on the cloud for whom failure or an outage of service can mean the loss of businesses and lives.
- Reliability can be used to understand how well the service will be available in context of different real-world conditions.

## Summarizing Failures and Fault (Recap)

A system or a component is said to “**fail**” when it *cannot meet* its promises or meet its requirements. A failure is brought about by the *existence* of “**errors**” in the system when the system as a whole stops providing the required service. The *cause* of an error is called a “**fault**”.

A **failure** of a system is brought about due to an **error** in the system caused by a **fault**.

There can be different kinds of faults

- **Transient Faults** : Appears once, then disappears
- **Intermittent Faults** : Occurs, Vanishes, reappears, but no real pattern (worst form of faults)
- **Permanent Faults** : Once it occurs, only the replacement/repair of the faulty component will allow the Distributed System to function normally

A common metric to measure reliability is the **Mean Time Between Failures (MTBF)** which is the average length of operating time between one failure and the next

$$\text{MTBF} = \text{Total uptime} / \# \text{ of Breakdowns}$$

For example, if the specified operating time is 24 hours and in that time 12 failures occur, then the MTBF is two hours.

It is impossible to reduce the probability of a fault to zero; therefore it is usually best to design fault-tolerance mechanisms that prevent faults from causing failures

# Types of Failure from a Designer perspective (Recap)

- Faults can occur both in *processes* and *communication channels*. The reason can be both software and hardware.
- Characterization of these different types of faults and fault models are needed to build systems with predictable behavior in case of faults (systems which are fault tolerant).

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
<i>Receive omission</i>	A server fails to receive incoming messages
<i>Send omission</i>	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	The server's response is incorrect
<i>Value failure</i>	The value of the response is wrong
<i>State transition failure</i>	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Crash: fail-stop, fail-safe (*detectable*), fail-silent (*seems to have crashed*)

## Issues which can lead to Faults in a distributed system : Timeouts and Unbounded Delays

### How long should the time-out be?

- Every packet is either delivered within some time  $d$ , or it is lost, but delivery never takes longer than  $d$
- If there is a Guarantee that a non-failed node always handles a request within some time  $r$  you could guarantee that every successful request receives a response within time  $2d + r$  and
  - If you don't receive a response within that time, you know that either the network or the remote node is not working
  - Unfortunately, most systems we work with have neither of those guarantees:

### Asynchronous networks have *unbounded delays*

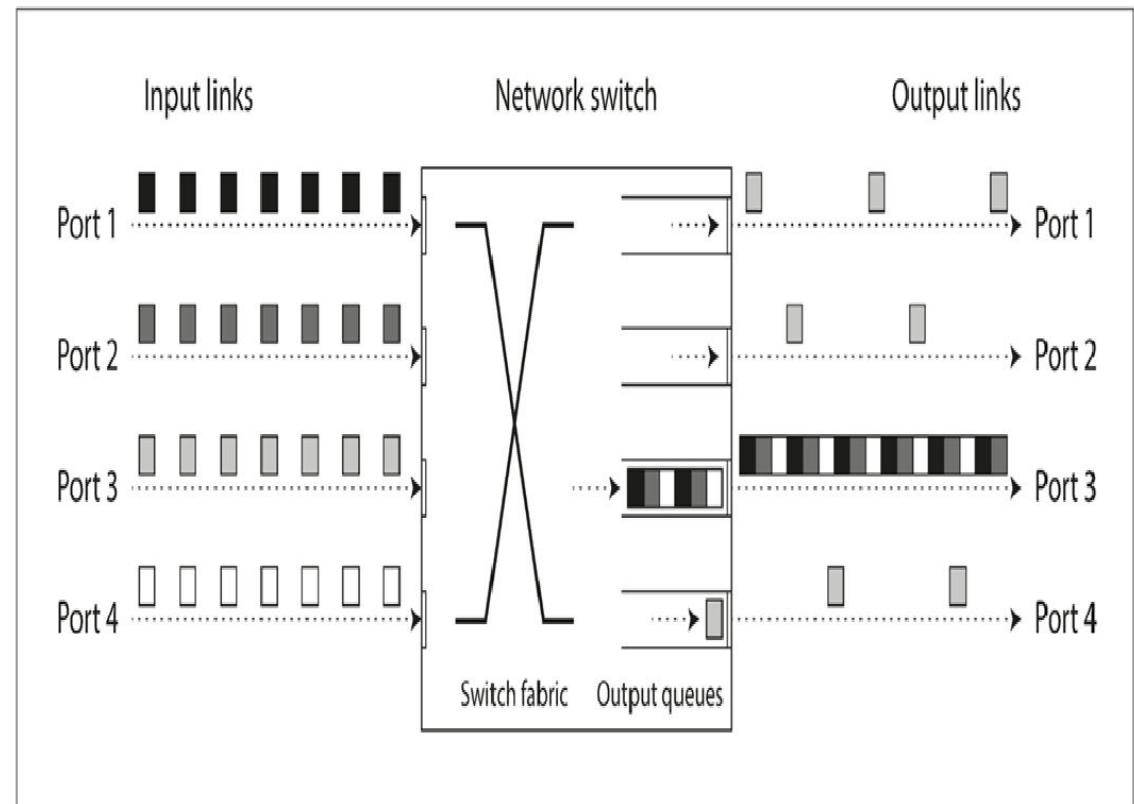
- Try to deliver packets as quickly as possible, but there is no upper limit on the time it may take for a packet to arrive), and most server implementations cannot guarantee that they can handle requests within some maximum time

### Choose timeouts experimentally as there's no "correct" value for timeouts:

- Measure the distribution of network round-trip times over an extended period, and over many machines, determine the expected variability of delays.
- Taking into account of the application's characteristics, determine an appropriate trade-off between failure detection delay and risk of premature timeouts.
- Even better, rather than using configured constant timeouts, systems can continually measure response times and their variability (*jitter*), and automatically adjust time-outs according to the observed response time distribution.

## Issues which can lead to Faults in a distributed system : Network Congestion and Queueing

- On a busy network link, a packet may have to wait a while, until it can get a slot (this is called **network congestion**).
- If there is so much incoming data that the switch queue fills up, and there will be a delay in the some of the packet being moved to its output links
- This variability of packet delays on computer networks is most often due to **queueing**
- This could also lead to some of the packets being dropped, so these need to be resent—even though the network is functioning fine.



*If several machines send network traffic to the same destination, its switch queue can fill up. Here, ports 1, 2, and 4 are all trying to send packets to port 3.*

## Issues which can lead to Faults in a distributed system : Network Congestion and Queueing

---

- In public clouds and multi-tenant datacenters, increased and effective utilization of resources are achieved using Virtualization and sharing of the resources among many customers (tenants)
  - This includes the network links and switches, and even each machine's network interface and CPUs (when running on virtual machines).
  - Virtualized multi-tenant resources have overheads which can add latencies towards consuming the network traffic and thus increase Queue lengths
  - This leads to variable delays, congestion and queueing based on the workload characteristics of the multiple tenants making it hard for having a consistently deterministic and optimized designs
- Moreover, TCP considers a packet to be lost if it is not acknowledged within some timeout and lost packets are automatically retransmitted.
  - Although the application does not see the packet loss and retransmission, it does see the resulting delay
- Batch workloads such as MapReduce can easily saturate network links.
- As there is no control over or insight into other customers' usage of the shared resources, network delays can be highly variable if someone near you (a noisy neighbor) is using a lot of resources

Fault Models have to be arrived at for building systems with **predictable behavior** for handling these kind of faults

Class of Faults	Description
<i>Crash-stop faults</i>	<ul style="list-style-type: none"><li>The node may suddenly stop responding at any moment, and thereafter that node is gone forever it never comes back.</li><li>There are scenarios where this could be detected by other processes/applications or this may not be detectable by other processes/applications or nodes</li></ul>
<i>Crash recovery faults</i>	<ul style="list-style-type: none"><li>Nodes may crash at any moment, and start responding again after some unknown time.</li><li>In the crash-recovery model, nodes are assumed to have stable storage (i.e., nonvolatile disk storage) that is preserved across crashes, while the in-memory state is assumed to be lost.</li></ul>
<i>Byzantine (arbitrary) faults</i>	<ul style="list-style-type: none"><li>Nodes exhibit arbitrary behavior and may do absolutely anything</li></ul>
• • •	

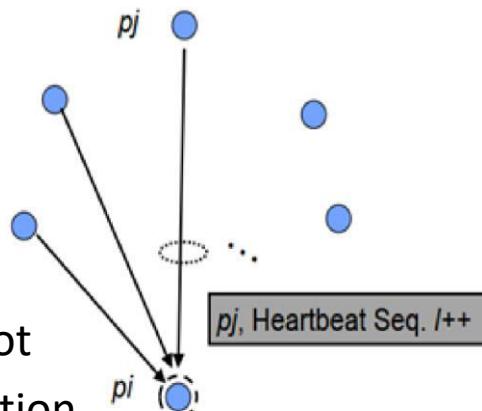
## Detection of Failures in a Distributed System (Recap)

### Failure Monitoring :

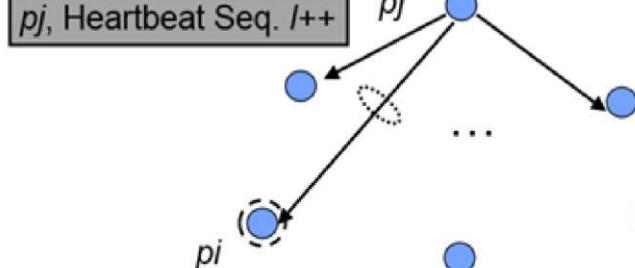
#### Centralized Heartbeat

#### Heartbeats

- Each application periodically sends a signal (called heartbeat)
- If the heartbeat is not received the application maybe declared as failed

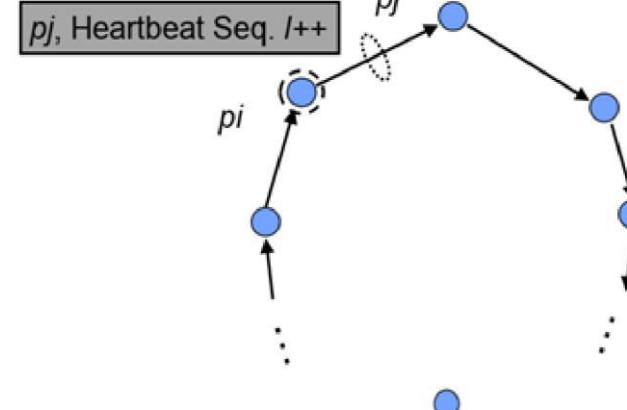


#### All-to-All Heartbeat



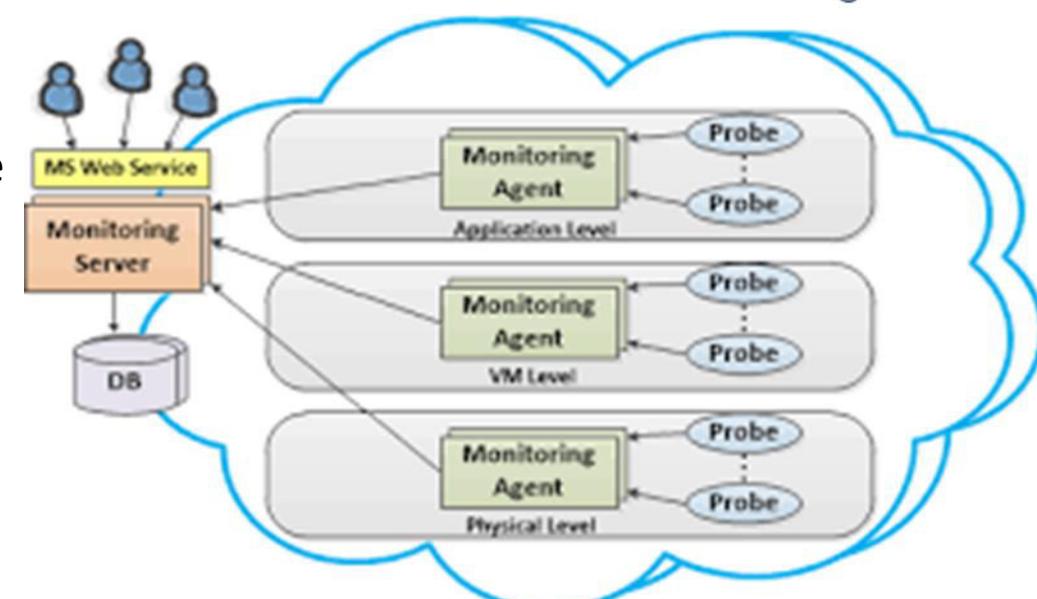
*Advantage: Everyone is able to keep track of everyone*

#### Ring Heartbeat



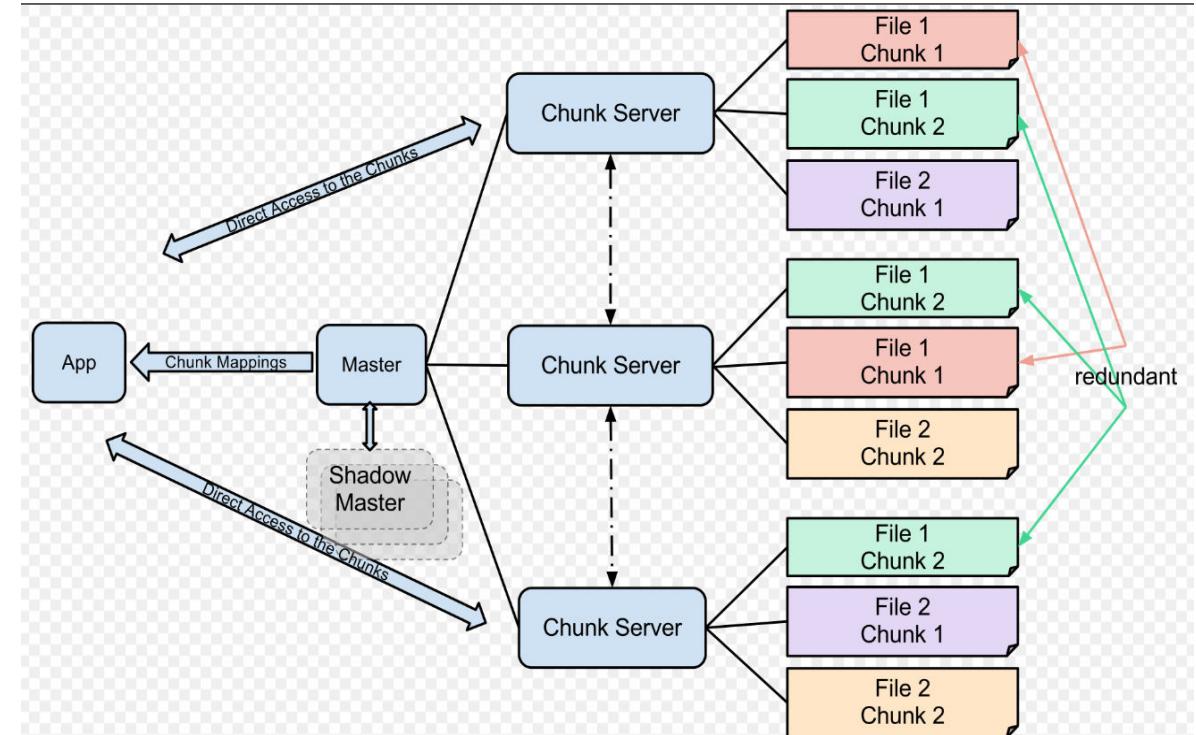
#### Probing

- Periodically sends a probe, which is a lightweight service request, to the application instance and decide based on the response.



## Detection of Failures in a Distributed System in Real World systems

- **GFS – Google File System**
- Centralised heartbeat mechanism
- **Chunkservers** – Hold the data of the file system
- Master/Leader node communicates with each chunkserver through Heartbeat messages
- Will know when a chunkserver is not reachable
- On chunkserver failure: Master begins replicating its contents to another chunkserver
- Heartbeat messages  
alternate use case: Additional configuration information sent through each heartbeat message at times



## Once a fault is detected

---

- Once a fault is detected, making a system tolerate it is not easy either: there is no global variable, no shared memory, no common knowledge or any other kind of shared state between the machines.
- Nodes can't even agree on what time it is, let alone on anything more profound. The only way information can flow from one node to another is by sending it over the unreliable network.
- Major decisions cannot be safely made by a single node, so we require protocols that enlist help from other nodes and try to get a quorum to agree.
- Some of the approaches for these scenarios on how to get to a consensus, electing a leader are leading towards solutions once a fault is detected
- A few other approaches for addressing partial failures are also discussed

## Once a fault is detected : Some strategies for dealing with them

---

### Strategies for dealing with partial failures

- **Using asynchronous communication across internal microservices** — Long chains of synchronous HTTP calls across the internal microservices can become the main cause of bad outages. Eventual consistency and event-driven architectures will help to minimize ripple effects. This approach enforces a higher level of microservice autonomy.
- **Using retries with exponential backoff** — This technique helps to avoid short and intermittent failures by performing call retries a certain number of times, in case the service was not available only for a short time. This might occur due to intermittent network issues or when a microservice/container is moved to a different node in a cluster.
- **Working around network timeouts** — In general, clients should be designed not to block indefinitely and to always use timeouts when waiting for a response. Using timeouts ensures that resources are never tied up indefinitely.

Note: [Exponential Backoff](#), in case you're not caught up with WT-II

<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/partial-failure-strategies>

## Once a fault is detected : Some strategies for dealing with them

---

### Strategies for dealing with partial failures

- **Use the Circuit Breaker pattern** — The client process tracks the number of failed requests. If the error rate exceeds a configured limit, a "circuit breaker" trips so that further attempts fail immediately. After a timeout period, the client should try again and, if the new requests are successful, close the circuit breaker.
- **Provide fallbacks** — In this approach, the client process performs fallback logic when a request fails, such as returning cached data or a default value. This is an approach suitable for queries, and is more complex for updates or commands.
- **Limit the number of queued requests** — Clients should impose an upper bound on the number of outstanding requests that a client microservice can send to a particular service. If the limit has been reached, those attempts should fail immediately. In terms of implementation, the [Polly Bulkhead Isolation policy](#) can be used to fulfill this requirement.

## Once a fault is detected : Some strategies for dealing with them

---

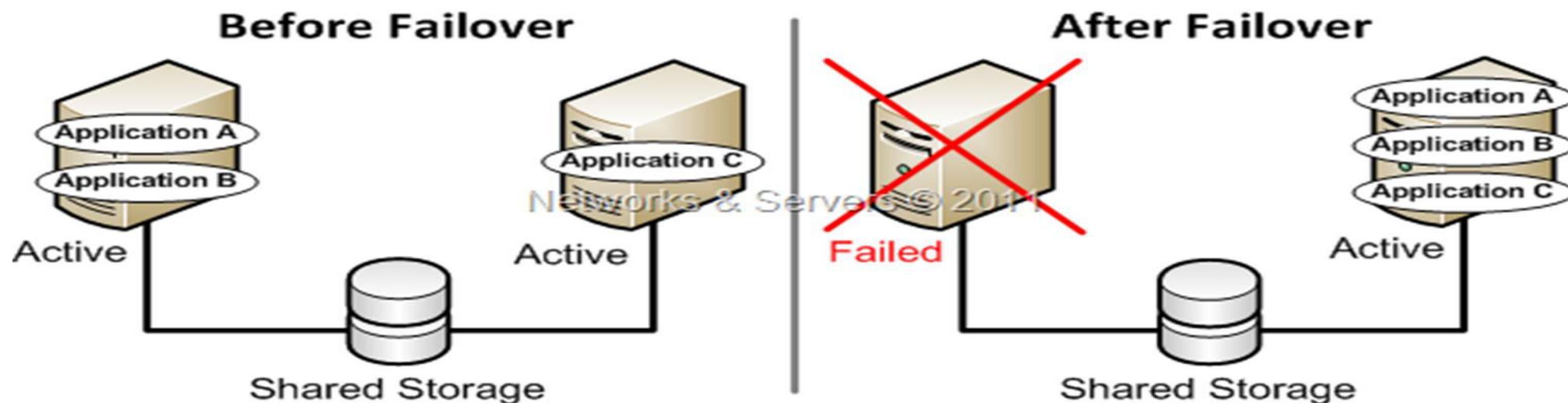
### Failover Strategy

Given that we have replication of data and computes which have been discussed till now, which are available, one of the options once a fault is found would be to failover

- Failover is switching to a replica (this could be a redundant or standby computer server, system, hardware component or network) upon the failure or abnormal termination of the previously active application, server, system, hardware component, or network.
- Failover can be performed through
  - An Active-active architecture
  - An Active-Passive or Active-Standby architecture

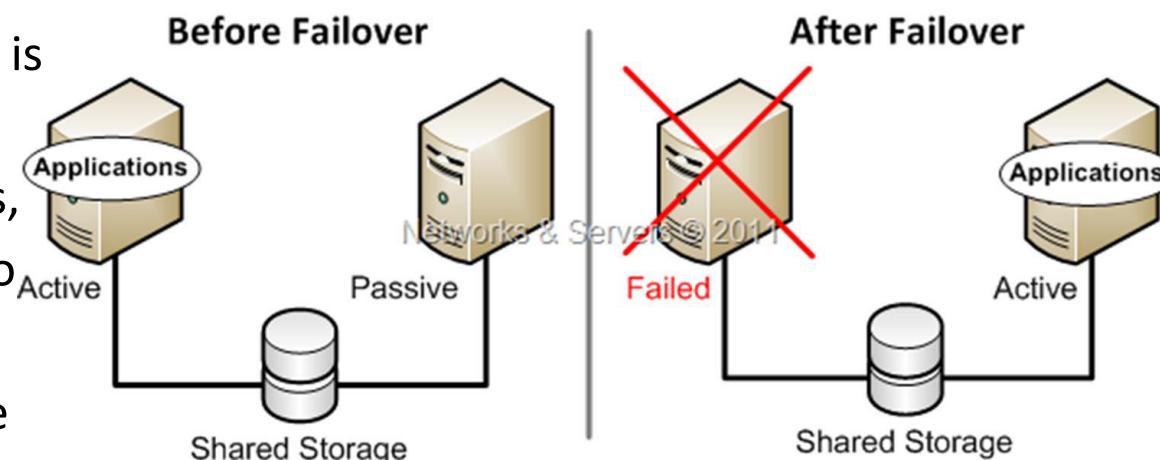
**Active-Active Failover Architecture** (sometimes referred to as symmetric)

- Each server is configured to run a specific application or service and provide redundancy for its peer.
- In this example, each server runs one application service group and in the event of a failure, the surviving server hosts both application groups.
- Since the databases are replicated, it mimics having only one instance of the application, allowing data to stay in sync.
- This scheme is called Continuous Availability because more servers are waiting to receive client connections to replicate the primary environment if a failover occurs.



### Active-Passive Failover Architecture (or asymmetric)

- applications run on a primary, or master, server.
- A dedicated redundant server is present to take over on any failure but apart from that it is not configured to perform any other functions.
- Thus, at any time, one of the nodes is active and the other is passive.
- The server runs on the primary node until a failover occurs, then the single primary server is restarted and relocated to the secondary node.
- The fallback to the primary node is not necessary since the environment is already running on the secondary node.
- Client connections must then reconnect to the active node and resubmit their transactions.

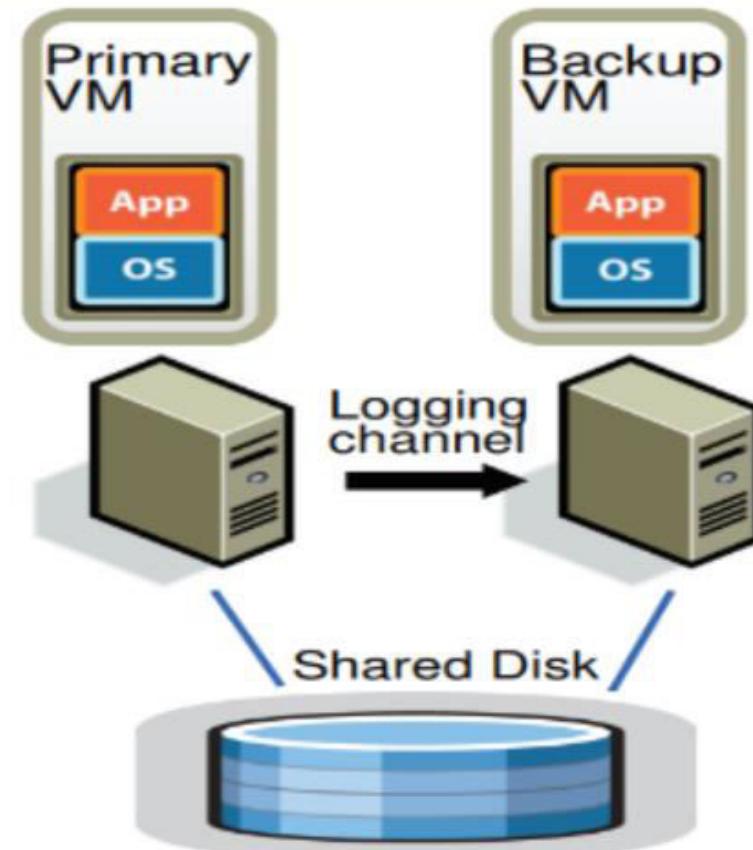


Note: Failback is the process of restoring operations to a primary machine or facility after they have been shifted to a secondary machine or facility during failover

## Once a fault is detected : Some strategies for dealing with them

### VMWare FT

- VMWare FT configuration involves Backup and Primary VMs
- Leverages the concept of virtual lock-step
- Both the VMs execute all operations
- Only one VM sends out outputs to clients
- When Primary VM fails, Backup VM takes over
- After Primary VM recovers, it acts as the new Backup VM



**Figure 1: Basic FT Configuration.**

# CLOUD COMPUTING

## Additional Resources

---

1. [https://www.youtube.com/watch?v=qrzhZY\\_IB7w](https://www.youtube.com/watch?v=qrzhZY_IB7w)



**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



# CLOUD COMPUTING

## Fault tolerance

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

## Distributed Computing System : Availability

Key aspect of designing a Distributed Computing System beyond Performance are

### 1. System availability

- Can be used to describe the period when a service is available and work as required, when required during the period when it needs to be used for a purpose.
- Often expressed as a percentage indicating how much time the particular system or component is available in a given period of time in consideration, where a value of 100% would indicate that the system never fails.
- It's ***oriented towards the users*** and is an assurance that the system or component which the users need for their job is available to them when they need it.

### 2. Application flexibility

- Flexibility to applications to use different computer platforms

### Uptime

- This is oriented towards the service provider and is the assurance that the system will be running or will be functionally operational for the time-period.
- It's typically computed in terms of a percentile e.g., 99.999 % or also called five 9s.

### Downtime

- It's Outage duration for which the system would be unavailable or non-operational.
- The unavailability of the system or network may be due to any system failures (unplanned event) which may occur from a software crash, hardware component crash or from communication failures (network outages).
- Maintenance routines (this are planned events) can also lead to unavailability of the system and thereby causing Downtime

### Slow or non-responsiveness

- Refers to system or a service e.g., a transaction taking unacceptably long periods of time for processing and returning the response to the user (depending on the transaction)

## Uptime and Downtime

- Helps us quantify the success value of these services or systems.
- Generally, every service level agreements and contracts include an Availability or uptime and downtime assuring the service availability to the users.

$$\text{Percentage Availability (or Uptime)} = \frac{\text{time agreed to be up} - \text{time for which down}}{\text{time agreed to be up}} \%$$

Let us consider a website monitored for 24 hours (= 86400 seconds).

Now in this timeframe say, the monitor was down for about 10 minutes (=600 seconds)

$$\text{Availability or Uptime \%} = \frac{86400 - 600}{86400} = 99.31\%$$

$$\text{Downtime \%} = \frac{\text{Time it was down}}{86400} = \frac{600}{86400} = 0.69\%$$

### Example

Let's say you monitored a website during 24 hours (which translates to 86,400 seconds), and in that timeframe the website went down for 10 minutes (600 seconds). To define the uptime and downtime percentages, the following calculation is performed:

Total time your website was down: 600 seconds

Total time your website was monitored: 86,400 seconds

Downtime percentage =  $600 \text{ seconds} / 86,400 \text{ seconds} = 0.0069 = 0.69\%$

Uptime percentage =  $100\% - 0.69\% = 99.31\%$

- We discussed on the kinds of ***faults*** which can occur in a distributed environment.
- If there is any failure that will pull down the operation of the entire system, then it's called a ***single point of failure***. The rule of thumb is to design a dependable computing system with no single point of failure.
- In general, as a distributed system increases in size, availability decreases due to a higher chance of failure and a difficulty in isolating the failures.
- ***High availability*** refers to the design of the system environment such that all single points of failure are removed through redundancy and faults are tolerated through automatic fail-over to the backup/replicas leading to the system having virtually no downtime
- Most clusters are designed to have High Availability with detection of the faults and failover capability with the techniques as discussed in the last session.

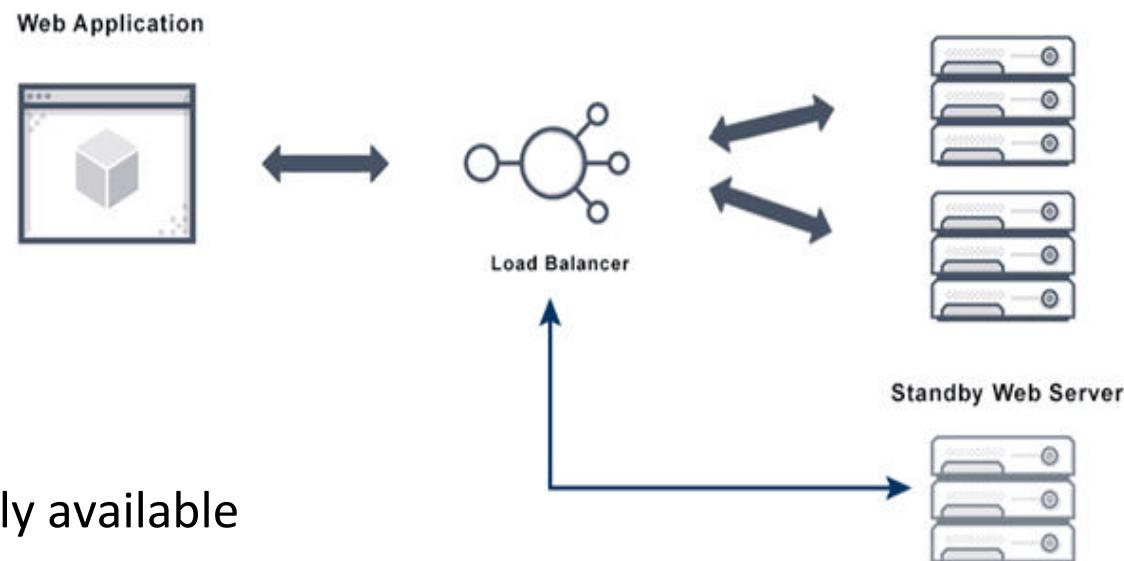
### Approaches used by some of the cloud service providers to measure availability

- Microsoft cloud services : time when the overall service is up, divided by total time.
- Gmail : percentage of successful interactive user requests for uptime error rate & consecutive minutes of downtime for downtime.
- Amazon Web Services : error rate - average percentage of requests that result in errors in a 5-minute interval.
- Recent published work also use approaches as “Windowed user-upptime” factoring in user perceived availability over many windows to distinguish between many short periods of unavailability and fewer longer periods of unavailability.

## What is Fault tolerance?

### Fault Tolerance

- Is the system's ability to continue operating un-interrupted despite the failure of one or more of its components.
- This fault tolerance can be achieved by hardware, software, or a combined solution e.g., leveraging load balancers or components as such
- Fault tolerance enables systems to be reliable and Highly available
- There are two ways to consider Fault-tolerance
  - **Fail-Safe tolerance** – Given safety state is preserved but performance may drop. E.g., Due to failure, no process can enter its critical section for an indefinite period. In a traffic crossing, failure changes the traffic in both directions to red.
  - **Graceful degradation** – Application continues, but in a degraded mode .. So depends on how much degradation is acceptable. E.g., Processes will enter their critical sections, but not in timestamp order



## Considerations for building Fault Tolerance

---

Important considerations when building fault tolerance into systems in an organizational setting include

### 1. Downtime

A fault-tolerant system is expected to work continuously with no acceptable service interruption.

### 2. Scope or Extent of fault-tolerance

- Needs to manage failures and minimize downtime.
- Considers backup components like power supply backups, as well as hardware or software that can detect failures and instantly switch to redundant components.

### 3. Cost

Needs additional cost as it requires the continuous operation and maintenance of additional, redundant components.

Building Fault tolerance, would need to have considerations in terms of the system's need for tolerance to service interruptions, the cost of such interruptions, existing SLA agreements with service providers and customers, as well as the cost and complexity of implementing full fault tolerance.

## Approaches for Building Fault Tolerance

---

- The goal would be to increase the system availability by making the systems to extend the uptime, reduce the downtime.
- Fault tolerance of the systems supports extension of the uptime and minimizes the possible downtime.

There are different approaches to support fault tolerance

### 1. Redundancy

- This looks to eliminate the single point of failures in the systems whether in the hardware components as part of manufactured systems E.g., Power supplies, Multiple processors, Segmented memory, Redundant disks etc. or could be software components which avoids single point of failures.

### 2. Reliability

- This looks at the dependability of components to function under the stated conditions.
- Its dependent on the process used to build the component and can be analyzed based on the failure's logs, frequency of failures etc.

### Reliability (Cont.)

- Typically measured using the metric **Mean Time Between Failure (MTBF)** which is the average time between **repairable failures** of the technology components of the System (Higher better). Considering the time where you want to compute it for

$$\text{MTBF} = \frac{\text{Total Operational Time in that Interval}}{\# \text{ of failures during Sampling Interval}}$$

- There are couple of other Metrics which are also relevant called **Mean Time To Failure (MTTF)** which is the average time between **non-repairable failures** of a technology product

$$\text{MTTF} = \frac{\text{Total time of correct opeeration (uptime)in a period}}{\# \text{ of tracked Items}}$$

E.g., If there are 3 identical systems starting from time 0 until all of them failed. Say the first system failed at 10 hours, the second failed at 12 hours and the third failed at 14 hours. The **MTTF** is the average of the three failure times = 12 Hours

### 3. Repairability

- It's a measure of how quickly and easily suppliers can fix or replace failing parts . **Mean Time To Recovery (MTTR)** is a metric used for measuring the time taken to do actual repair

$$\text{MTTR} = \frac{\text{Total hours of downtime caused by system}}{\# \text{ of failures}}$$

E.g., A system fails three times in a month, and the failures resulted in a total of six hours of downtime, then **MTTR** would be two hours.

### 4. Recoverability

- This refers to the ability to overcome a momentary failure in such a way that there is no impact on end-user availability.
- Recoverability also includes retries of attempted reads and writes out to disk or tape, as well as the retrying of transmissions down network lines.
- This could also include mechanisms of acknowledgements used to ensure missing or non-ordered delivery of sequence of data in protocols like TCP
- Building Error control coding (ECC) E.g., Parity Bit, Hamming Code, CRC much less redundancy than replication
- Checkpointing and Rollbacks, usually through logging, but may not complete tasks predictably in case of rollbacks

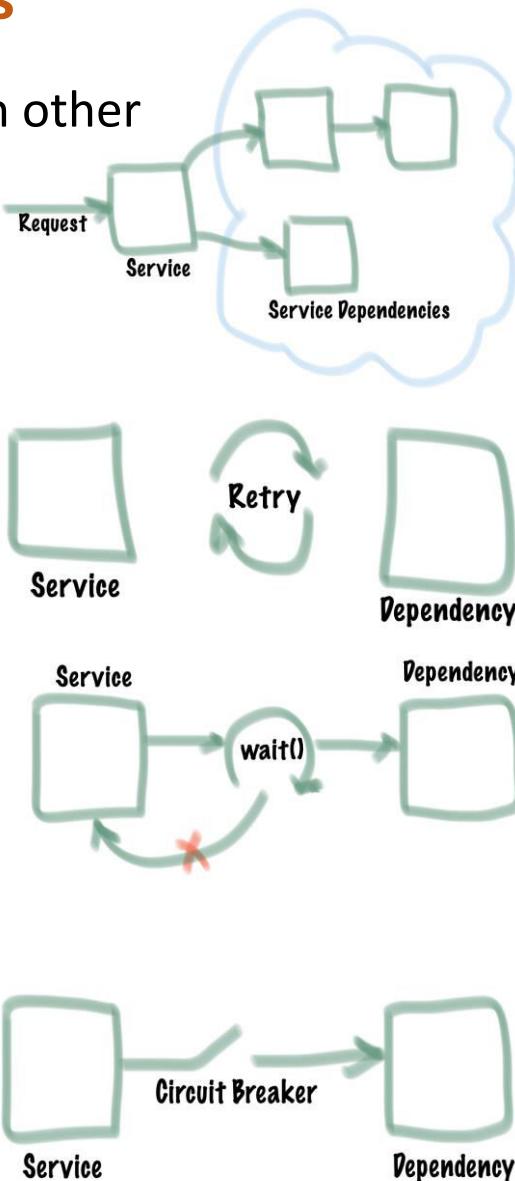
### 5. There are others which can be looked at too .. like Reputation, Robustness or Resilience



## A few additional techniques used for Fault tolerance in distributed systems

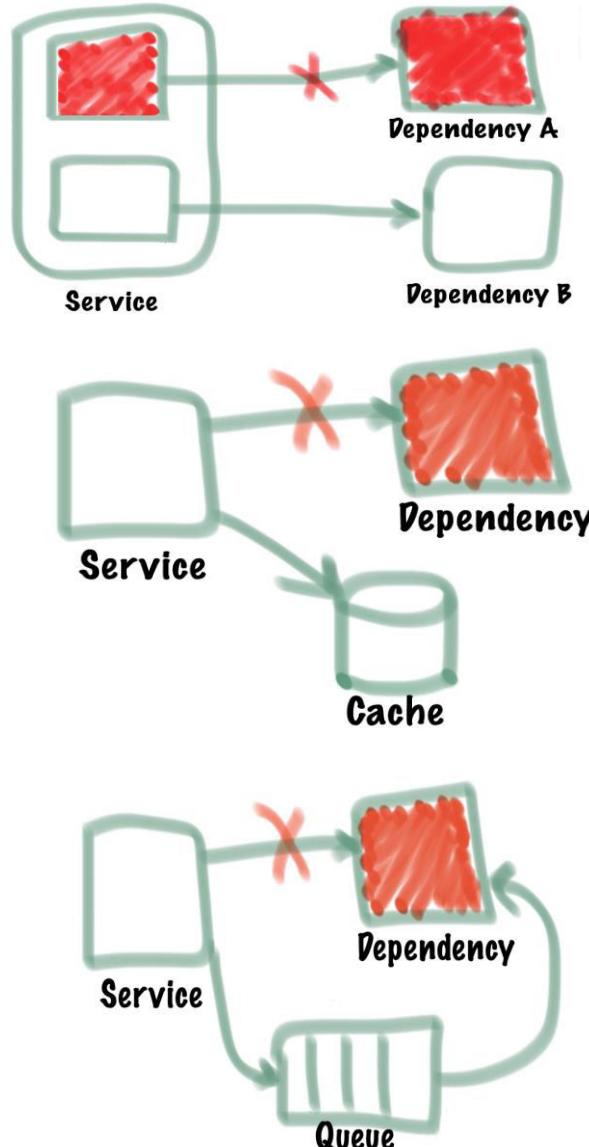
A distributed Cloud system with network of computers, which are communicating with each other by passing messages comes with the challenges of failures like failure or degradation of Network connections, Server failures, software has bugs, etc. Handling the interaction endpoints in terms of

- **Retries** - Trying the same request again causes the request to succeed. It happens because of partial or transient failures. This can be coupled with some kind of **exponential backoff**
- **Timeouts** - When a request is taking longer than usual, it might increase latency in the system (and fail eventually) so using some kind of **connection and request timeouts** will help. This can also help in avoiding impact on resources used for requests and avoid the server quickly running out of the resources (memory, threads, connections, etc.).
- **Use circuit breakers** – When there is an issue with the frequency of failures, stop calling for some time like a circuit breaker opening



## A few additional techniques used for Fault tolerance in distributed systems - 2

- **Isolate Failures (Bulkheads)** – Have low coupling and Separate processes/threads pools dedicated to different functions so that if one fails, the others will continue to function.
- **Cache** - Save the data that comes from remote services to a local or remote cache and reuse the cached data as a response during one of the service failure
- **Queue** - Setup a queue for the requests to a remote service to be persisted until the dependency is available.
- **Two Phase Commit** -



### A few additional techniques used for Fault tolerance in distributed systems - 3

Additionally, to the generic ones described earlier, there are other different best practices or patterns which exist for fault tolerance in micro services too.

- The usage of Asynchronous communication (for example, message-based communication) across internal microservices where It's advisable not to create long chains of synchronous HTTP calls across the internal microservices. Eventual consistency and event-driven architectures will help to minimize ripple effects. These approaches enforce a higher level of microservice autonomy and therefore prevent against the problems

- Percentage of time the system is up and running normally

$$\text{System Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

- A system is highly available if it has a long mean time to failure (MTTF) and a short mean time to repair (MTTR)
- High Availability is the quality of a system or component that assures a high level of operational performance for a given period. E.g. 99% of availability in a year leads to up to 3.65 days max of downtime (1%).
- One of the goals is to eliminate single points of failure in the infrastructure.
- When setting up robust production systems, minimizing downtime and service interruptions is often a high priority.

# CLOUD COMPUTING

## Additional Resources

---

1. <https://www.youtube.com/watch?v=ADHcBxEXvFA>
2. <https://www.youtube.com/watch?v=R2FT5edyKOg>
3. <https://hub.schoolofdevops.com/courses/docker-mastery/lectures/2915640>
4. <https://medium.com/the-cloud-architect/creating-your-own-chaos-monkey-with-aws-systems-manager-automation-6ad2b06acf20>
5. <https://sksonudas.medium.com/fault-tolerant-patterns-for-microservice-8d0c40f4f514>
6. <https://www.gremlin.com/chaos-monkey/chaos-monkey-tutorial/>
7. <https://www.youtube.com/watch?v=-smx0-qeurw>
8. [https://www.youtube.com/watch?v=AnSrpk\\_thDE](https://www.youtube.com/watch?v=AnSrpk_thDE)
9. <https://www.youtube.com/watch?v=0qbJV2CYbos>



**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



## CLOUD COMPUTING

### Cluster Co-Ordination Consensus

---

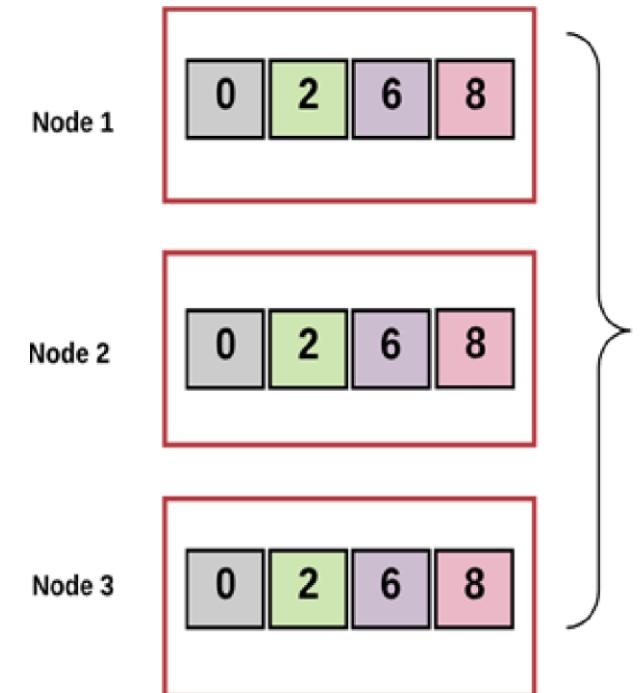
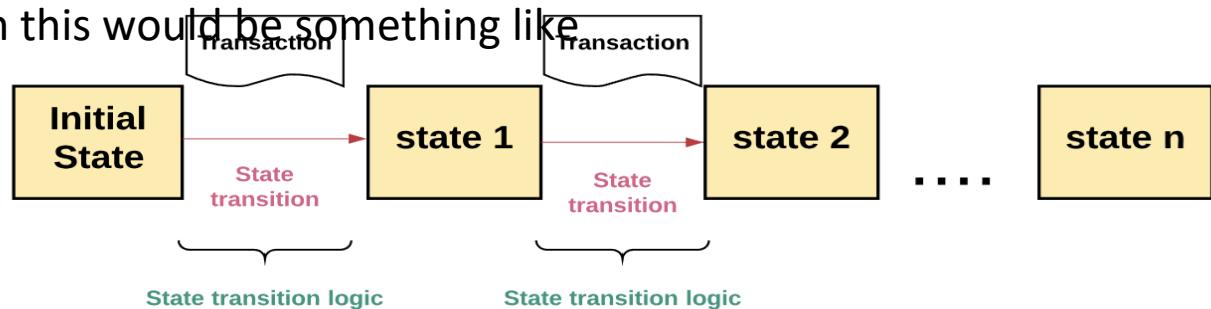
**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

#### **Acknowledgements:**

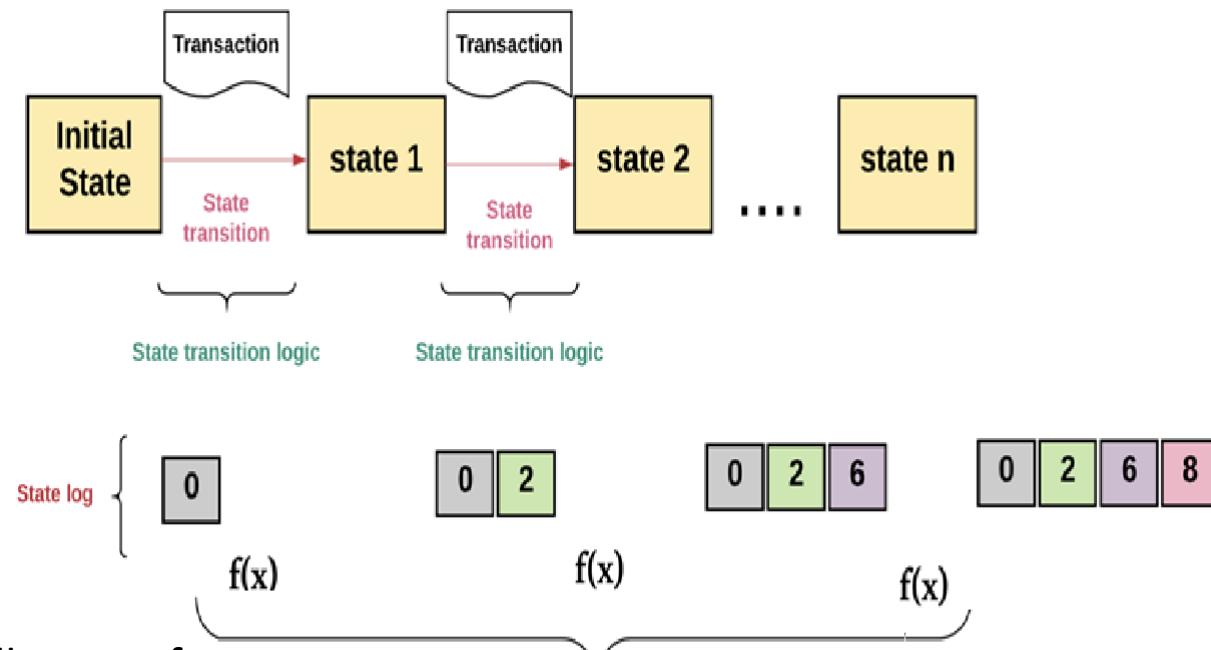
Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

- We discussed that Distributed System has parts of the system functioning on multiple nodes but functioning as a single system with a single state machine.
- These distributed systems are replicated across many computers but still function as a single state machine using distributed transactions
- If the transaction is valid in this replicated state machine, a set of inputs will cause the state of the distributed system to transition to the next state.
- Each of the distributed transaction is atomic and the system will all complete or never complete.
- The logic for transitioning from one valid state to the next is called the state transition logic
- For the distributed system this would be something like



## Distributed Systems and Consensus:

- In other words, a replicated state machine is a set of distributed computers that all start with the same initial value. For each transition, each of the processes decides on the next value. Consensus is all the computers collectively agreeing on the output of this value
- They would need to continue to have the same goal (through having a consistent transaction log across every computer in the system).
- The goal of the consensus algorithm would be to achieve that all the computers would be in the same state with say  $f(x)$  as seen in the figure
- This must happen as the distributed system will continue to receive new inputs and have new transactions with the challenges of
  - Some computers could be faulty
  - Network could be broken or slow
  - Not having a global clock to synchronize the order of the events



## The Consensus Problem, Defined

---

- Consider that there are N nodes
- Say each node (say) p has
  - Input variable  $x_p$  which is initially at either 0 or 1
  - Output variable  $y_p$  which was initially at b can only be changed once

An algorithm achieves consensus if it satisfies the following conditions:

- **Agreement:** All non-faulty nodes decide on the same output value (in this example either 1 or 0)
- **Termination:** All non-faulty nodes eventually decide on some output value (or do not decide differently).

Couple of other basic properties or constraints

- **Validity:** Any value decided is the value proposed
- **Integrity:** A node decides at most once.
- **Non-Triviality:** there is at least one initial system state that leads to each of the all-0's or all-1's outcomes

Generally, we can define a consensus algorithm by three steps:

### Step 1: Elect

- Processes elect a single process (i.e., a leader) to make decisions.
- The leader proposes the next valid output value.

### Step 2: Vote

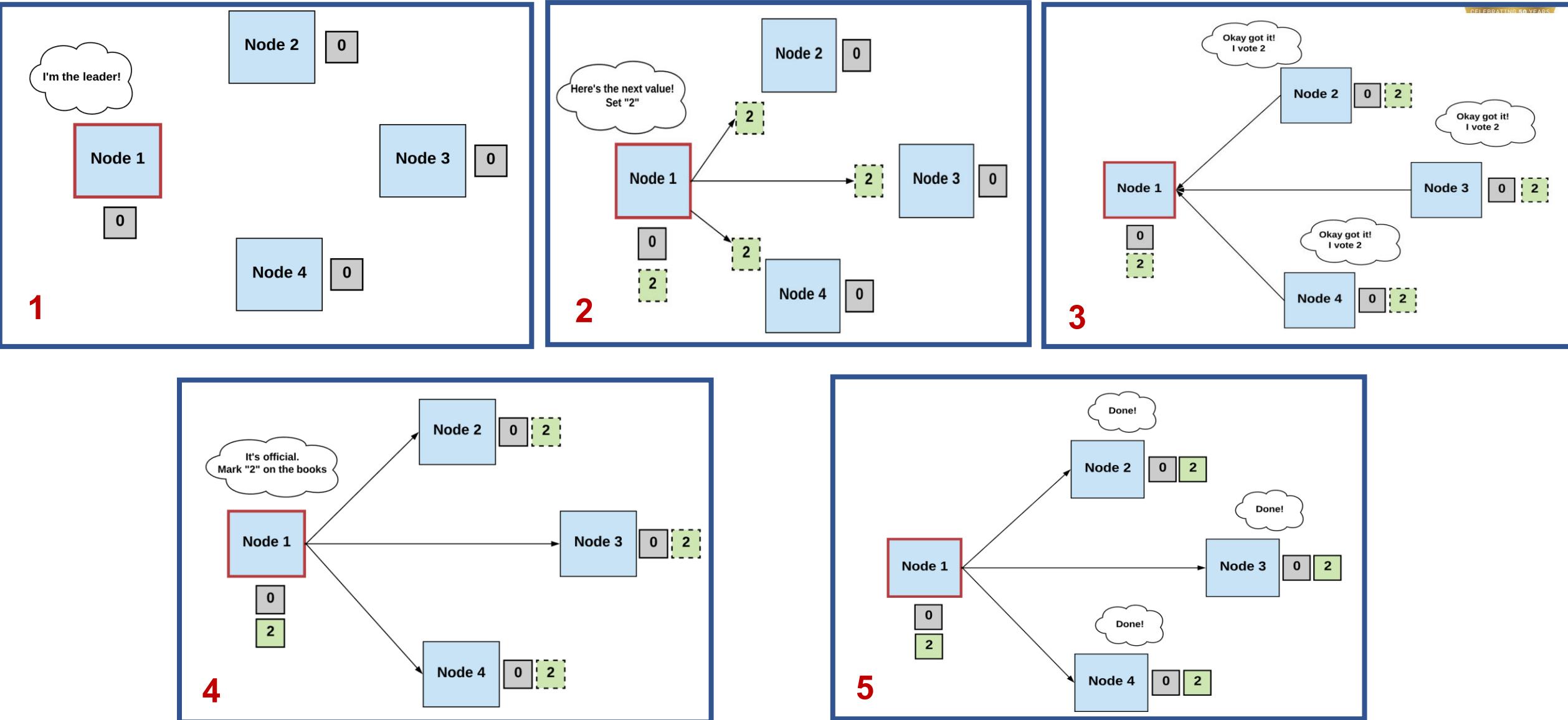
- The non-faulty processes listen to the value being proposed by the leader, validate it, and propose it as the next valid value.

### Step 3: Decide

- The non-faulty processes must come to a consensus on a single correct output value. If it receives a threshold number of identical votes which satisfy some criteria, then the processes will decide on that value.
- Otherwise, the steps start over.

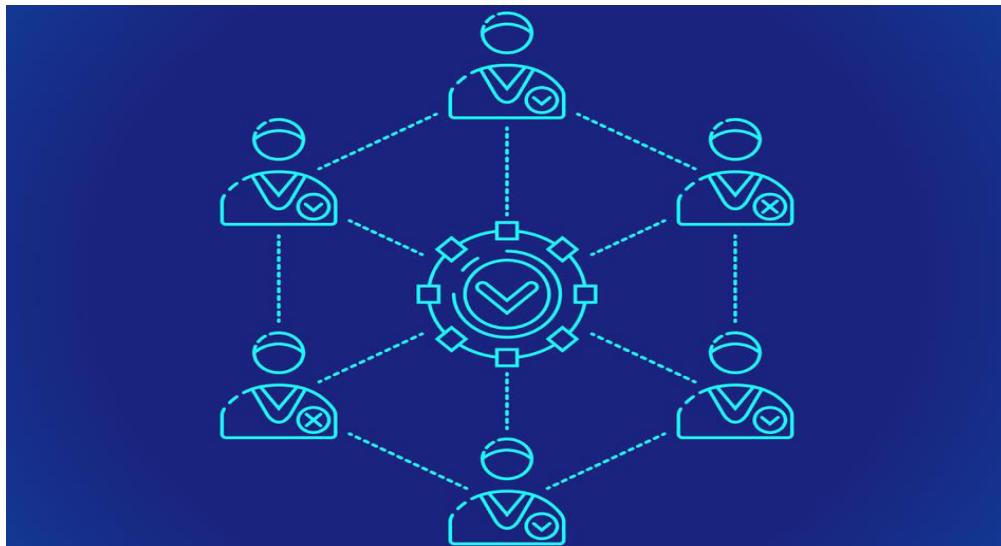
# CLOUD COMPUTING

## A Typical approach on how Consensus is arrived at



## Summarizing Consensus

- Achieving overall system reliability in the presence of several faulty processes is a fundamental problem in distributed systems, cloud computing, and multi-agent systems.
- During computation, the system reliability often requires processes to agree on some value. So, the consensus problem needs agreement among several processes for a single data value.
- Consensus is the task of all processes in a group to agree on some specific value based on voting. A group consists of  $N$  processes. All processes in that group set their output variable either to 0 or 1. This statement is called consensus.



## Common Challenges to be factored in while arriving at consensus

### 1. Reliable multicast:

- When a group of servers, attempt to solve a particular problem in spite of failure, they have to reach a consensus.
- The group of servers together, need to receive the same updates in the same order as sent which is known as a reliable multicast. Given the failures, this leads to the Reliable multicast problem and will need to be addressed by consensus.

### 2. Membership Failure detection:

A group membership failure occurs when one of the group members fail/crash. The problem would be for all of the rest of the group to be aware of the failure/crashing of one of the group members be aware of the current status (i.e., processes is living or failed) . This is typically called Membership failure detection, which is equivalent to the consensus.

## Common Challenges to be factored in while arriving at consensus

### 3. Leader election:

- A group of servers together attempt to elect a leader among them, and this particular decision of electing the leader is known to everyone in that particular group. This problem is termed as leader election.
- Leader election under this specific failure requires the solution of consensus. So, the consensus problem can be solved by anyone, thus leading a solution to the leader election.

### 4. Mutual exclusion:

- A group of servers together attempt to ensure mutually exclusive access to the critical resources such as files writing on a file in a distributed database.
- In case of failure, this problem also requires the consensus protocols for reliability aspect.

## Common issues in the consensus problem

---

In the consensus problem every process will contribute towards value, and the goal is to have all the processes decide on some value. Once the decision is finalized, it cannot be changed. There are some other constraints that will also be required to be enforced to solve a particular consensus problem, such as the following:

- 1. Validity:** Every process will propose the same value, then that is what is decided.
- 2. Integrity:** Some process must have offered the selected values.
- 3. Non-triviality:** There is at least one initial state that leads to each of all 0s or all 1s outcomes.

## Importance of the consensus problem

---

Many problems in distributed systems are either equivalent to or harder than the consensus problem. This implies that if the consensus problem can be solved, then those problems in the distributed systems can also be solved. Some of the problems in distributed systems that are either equivalent to or harder than the consensus problem are as follows:

1. In failure detection, if a failure can be detected, the consensus problem can also be solved, or if the consensus problem is solvable, then the faults are also discoverable.
2. In leader election, exactly one leader is elected, and every alive process is aware of the same. If the leader election problem is equivalent to the consensus problem, then solving the consensus problem is equivalent to solving the leader election problem.
3. Agreement on a decision in the distributed system is hard thus, the consensus is a fundamental problem and solving it will be beneficial.

### 1. Synchronous Distributed System

The synchronous model states that each message is received within a limited time and the drift of each process' local clock has a known bound. Each step in a process belongs in between an upper bound and a lower bound. For example, a collection of processors connected by a communication bus such as a Cray supercomputer and a multicore machine.

Reaching consensus in a synchronous environment is possible because we can make assumptions about the maximum time it takes for messages to get delivered. Thus, in this type of system, we can allow the different nodes in the system to take turns proposing new transactions, poll for a majority vote, and skip any node if it doesn't offer a proposal within the maximum time limit.

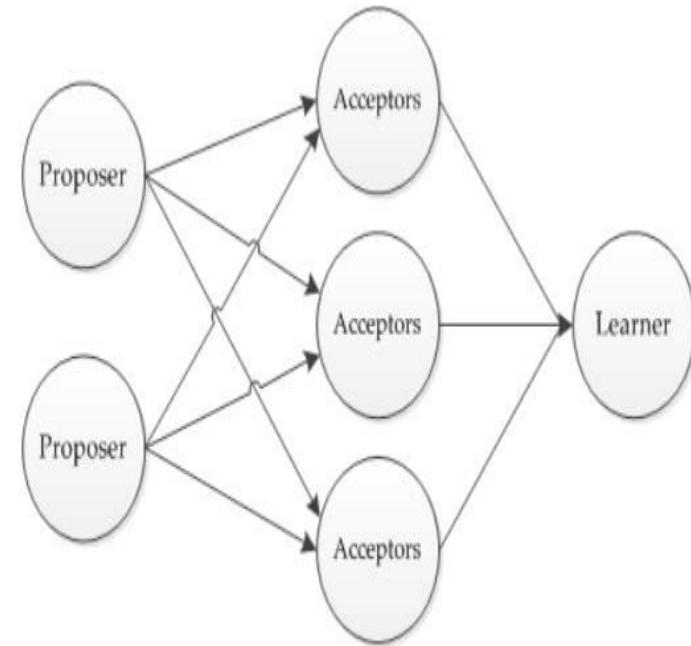
But assuming we are operating in synchronous environments is not practical outside of controlled environments where message latency is predictable, such as data centers which have synchronized atomic clocks.

### 2. Asynchronous Distributed System

- An asynchronous model does not have any specified bound on process execution. When the clocks drift, that bound is also arbitrary and the message transmission is delayed; that is, delay in message receiving is also not bounded. The Internet is a perfect example of an asynchronous distributed system model. Other examples include ad hoc networks and sensor networks.
- In the asynchronous system model, the consensus is impossible to solve. Whatever the protocols or algorithms, there is always a worst possible execution scenario with the failures and message delays, which will prevent the system from reaching the consensus. Therefore, although probabilistic solutions are used to solve the consensus problem, it might be impossible to solve; Fischer–Lynch–Paterson Impossibility says the same.

- Family of distributed algorithms used to reach consensus and has been used by companies like Google and Amazon
- Paxos define three roles in a distributed system:
  1. **Proposers**, often called leaders or coordinators.
  2. **Acceptors**, processes that listen to requests from proposers and respond with values.
  3. **Learners**, other processes in the system which learn the final values that are decided upon.
- Paxos nodes can take multiple roles
- Paxos nodes must know the number of majority acceptors
- A Paxos run aims at reaching a single consensus

- A *proposer* proposes a value that it wants agreement upon. It does this by sending a proposal containing a value to the set of all *acceptors*, which decide whether to accept the value.
- Each acceptor chooses a value independently
  - It may receive multiple proposals, each from a different *proposer* and it
  - Sends its decision to *learners*, which determine whether any value has been accepted.
- For a value to be accepted by *Paxos*, a majority of acceptors must choose the same value



I: Basic Paxos architecture. A number of proposers make proposals to acceptors. When an acceptor accepts a value it sends the result to learner nodes.

# CLOUD COMPUTING

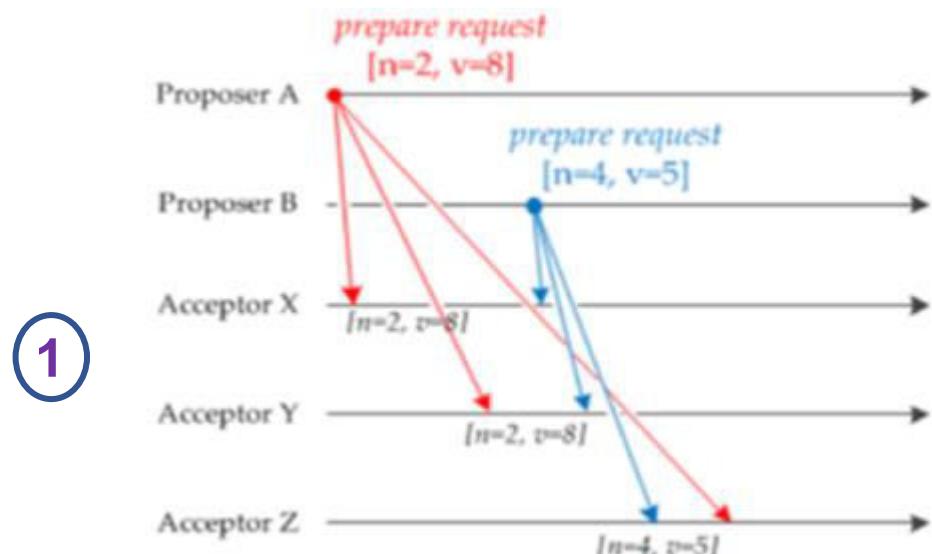
## Paxos Algorithm

n – proposal number  
v – proposed value  
 $n_v$  – number of the highest-numbered proposal accepted earlier  
 $v'$  – value of the highest-numbered proposal it has accepted

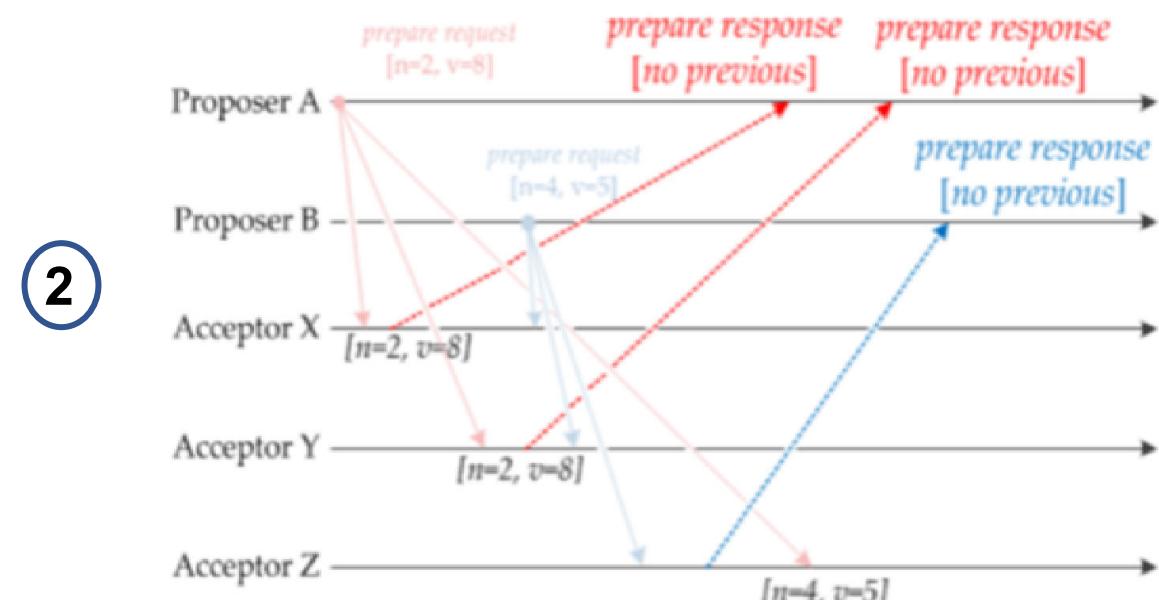
1. The proposer sends a message  $\text{prepare}(n, v)$  to all acceptors. (Sending only to majority of the acceptors is enough, assuming they will all respond.)
2. Each acceptor compares n to the highest-numbered proposal for which it has responded to a prepare message. If n is greater than  $n_v$  then it responds with ack  $(n_v, v')$  where  $v'$  is the value of the highest-numbered proposal it has accepted  $n_v$ , thus indicating its last received proposal number and value
3. The proposer waits (possibly forever) to receive ack from a majority of acceptors. It then sends accept  $(n, \max(v \text{ and } v' \text{ from all max acks received}))$  to all acceptors (or just a majority).
4. Upon receiving accept( $n, v$ ), an acceptor accepts v unless it has already received  $\text{prepare}(n')$  for some  $n' > n$ . If a majority of acceptors accept the value of a given proposal, that value becomes the decision value of the protocol.

## Example

- Proposers send two types of messages to acceptors:
  - **prepare** and **accept** requests.
- In the Prepare stage of this algorithm a proposer sends a **prepare request** to each acceptor containing a proposed value v, & a proposal number n.
- Each proposer's proposal number must be a positive, monotonically increasing, unique, natural number, with respect to other proposers' proposal numbers

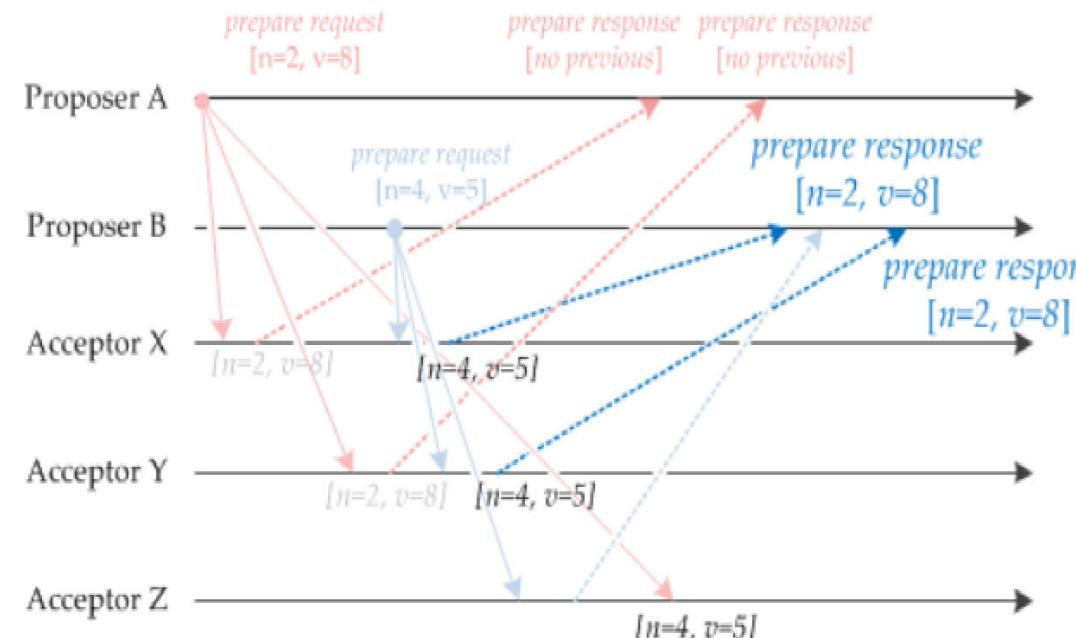


- Proposer A and B each send prepare requests to every acceptor. In this example proposer A's request reaches acceptors X and Y first, and proposer B's request reaches acceptor Z first

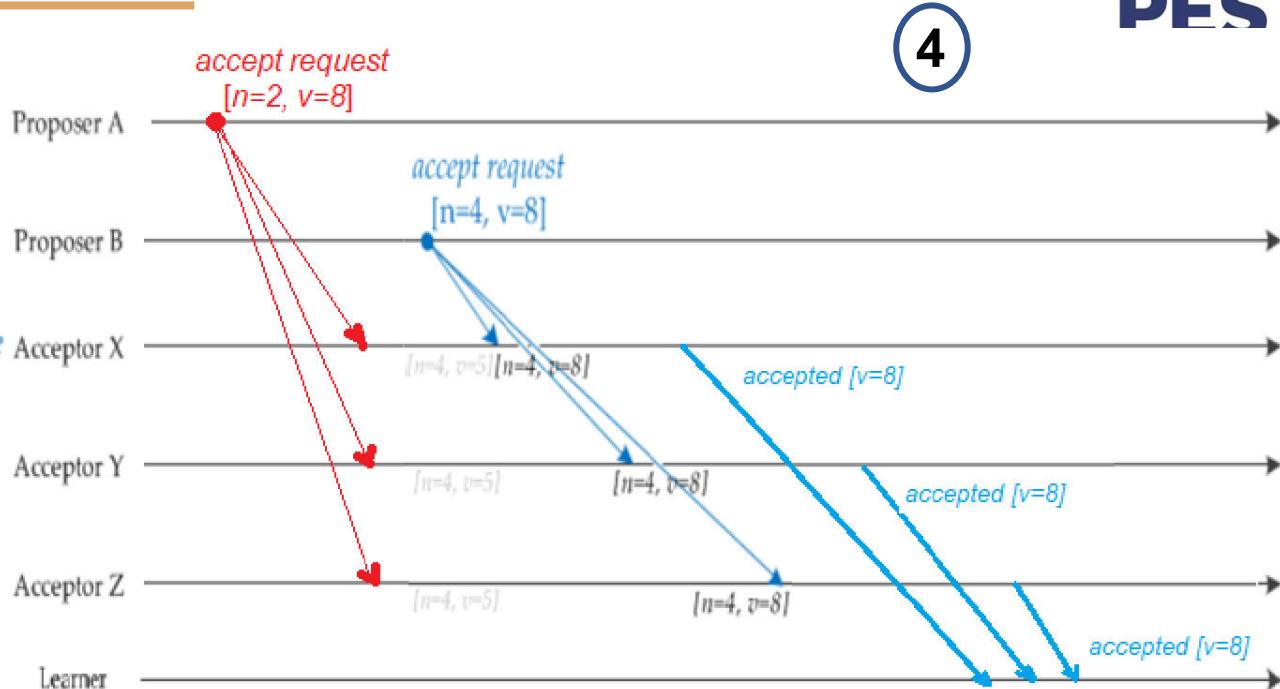


- Each acceptor responds to the first prepare request message that it receives
- If the acceptor receives a prepare request and has not seen another proposal, it responds with a prepare response which promises never to accept another proposal with a lower proposal number.

## Example



- Eventually, acceptor Z receives proposer A's request, and acceptors X and Y receive proposer B's request.
- Acceptor Z ignores proposer A's request because it has already seen a higher numbered proposal ( $4 > 2$ )
- Acceptors X and Y respond to B's request with the previous highest request that they acknowledged and a promise to ignore and lower numbered proposals



- Once a proposer has received prepare responses from a majority of acceptors it can issue an accept request
- Since proposer A only received responses indicating that there were no previous proposals, it sends an accept request to every acceptor with the same proposal number and value as its initial proposal ( $n=2, v=8$ ) which is rejected as the request number is below 4
- Proposer B sends an accept request to each acceptor, with its previous proposal number (4), and the value of the highest numbered proposal it has seen, which is not the earlier 5 but 8 from ( $n=2, v=8$ ). So it returns accept values of ( $n=4, v=8$ )
- A notification is send to every learner on the accepted value 8

## Further Reading

---

- [Paxos Consensus Algorithm](#)
- [Raft Consensus Algorithm](#)
- <https://www.cs.yale.edu/homes/aspnes/pinewiki/Paxos.html>
- <https://medium.com/@angusmacdonald/paxos-by-example-66d934e18522>
- [https://www.alibabacloud.com/blog/paxos-raft-epaxos-how-has-distributed-consensus-technology-evolved\\_597127](https://www.alibabacloud.com/blog/paxos-raft-epaxos-how-has-distributed-consensus-technology-evolved_597127)



**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**







## CLOUD COMPUTING

### Cluster Co-Ordination Leader Election

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

#### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

- Many applications require coordinator and/or leader to control data flow across nodes
- Leader-dependent algorithms depend on a leader
  - Leaders turn out to be the a single point of the failure in the entire network!
  - If the leader is down, the rest of the system will either go down or will be rendered faulty
  - To overcome this problem other nodes can take over the leader role when the current leader fails
  - A leader Selection Algorithm determines which node takes over in case of a leader failure!

## Leader Election in Distributed Systems

- Leader election is a process of designating a single node as the organizer, coordinator, initiator of tasks.
- Leader election is a process of determining which node will manage some of the tasks distributed across nodes.

### Reasons to elect a leader:

- A centralized control simplifies process synchronization, but is also a potential single point of failure,
- Leader election is intended to address this with a solution to choose a new controller (leader) upon failure of the existing controller/coordinator.
- There are many algorithms for leader election and one of them is chosen depending on the context

### When to elect a leader?

- During the system initiation
- When the existing leader fails.
  - If there's no response from the current leader for a predetermined time

### Goal of Election algorithm:

1. Elect one leader only among the non-faulty processes
2. All non-faulty processes agree on who is the leader.

### With the context that

- Any process can call for an election.
- A process can call for ***at most*** one election at a time.
- Multiple processes can call an election simultaneously.
  - All of them together must yield a single leader only
- The result of an election should not depend on which process calls for it.
- Messages are eventually delivered.

### Liveness and Safety condition in an Election

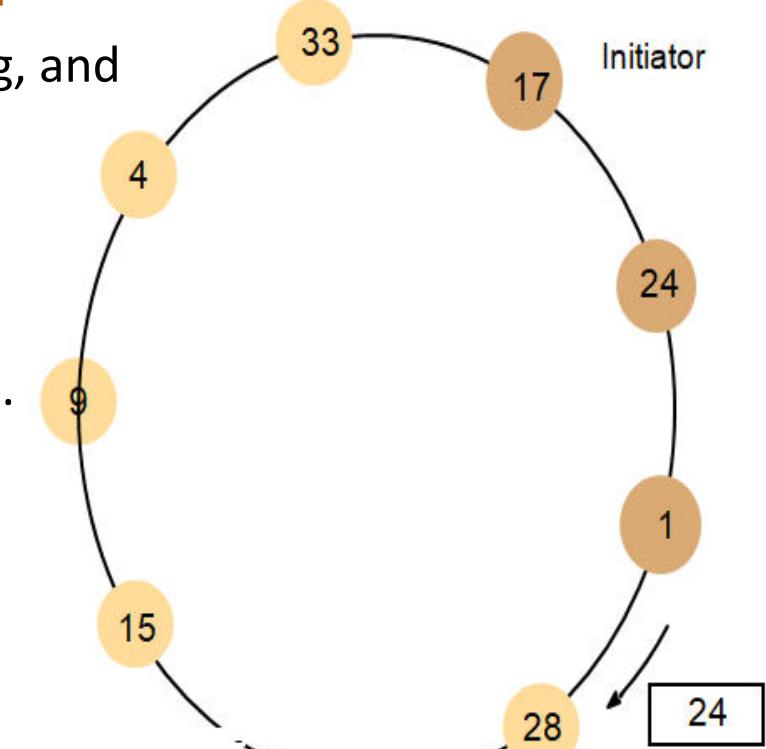
- Liveness condition expects every node will eventually enter a state that is elected or that is not elected.
- Safety condition expects only one node can enter the elected state and eventually become the leader of the distributed system.

### Liveness and Safety condition in an Election

- Liveness condition
  - Expects that Every node will eventually enter a state that is elected or that is not elected.
- Safety condition
  - Expects only one node can enter the elected state and eventually become the leader

- A run (execution) of the election algorithm must always guarantee at the end:
  - Safety: For all non-faulty process p: (p's elected = (q: a particular non-faulty process with the best attribute value) or Null)  
*"Within all the non-faulty processes (nodes) P, one non-faulty process Q with the best attribute value will be elected as leader, or the election will terminate with Null"*
  - Liveness: For all election: (election terminates) & For all p: non-faulty process, p's elected is not Null  
*"with all the non-faulty processes (nodes) entering an election, each one of them will eventually enter a state that is elected or that is not elected (Null)"*
- At the end, the non-faulty process with the best (highest) election *attribute value* is elected.
  - Attribute: Fastest cpu. Most disk space. Most number of files, priority etc.

1. Consider there are N Processes/Nodes which are organized in a logical ring, and
  - They communicate only with their logical neighbour and
  - All messages are sent clockwise around the ring.
2. Any process  $p_i$ , that discovers that the old coordinator has failed, will then initiate an “election” message that contains  $p_i$ ’s own id:attr (say in the E.g. it’s the Node/Process number). This is the *initiator* of the election.
3. When any process  $p_i$ , receives an *election* message, it compares the attr in the message with its own attr.
  - i) If the arrived attr is greater,  $p_i$  forwards the message.
  - ii) If the arrived attr is smaller and  $p_i$  has not yet forwarded an election message, it overwrites the message with its own id:attr, and forwards it.
  - iii) If the arrived id:attr matches that of  $p_i$ , then  $p_i$ ’s attr must be the greatest (why?), and it becomes the new coordinator.
  - iv) This process then sends an “elected” message to its neighbor with its id, announcing the election result.

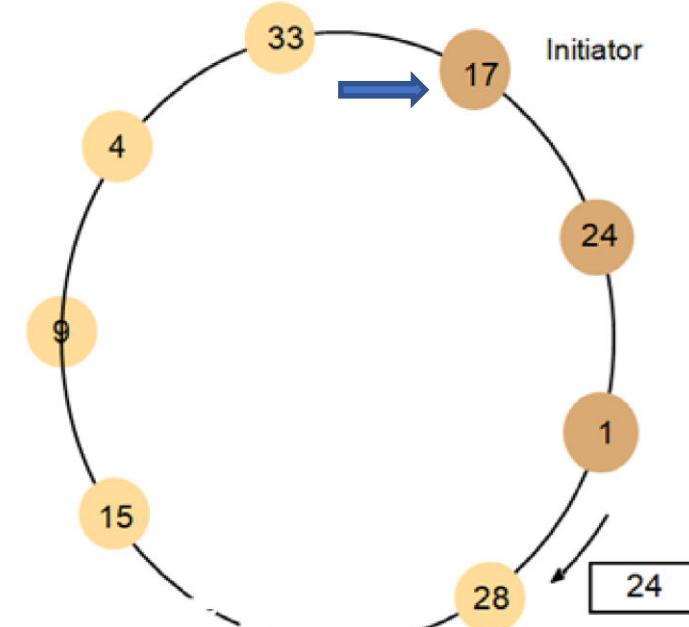


4. When a process  $p_i$  receives an elected message, it

- sets its variable  $elected_i$ , id of the message.
- forwards the message, unless it is the new coordinator.

In the example:

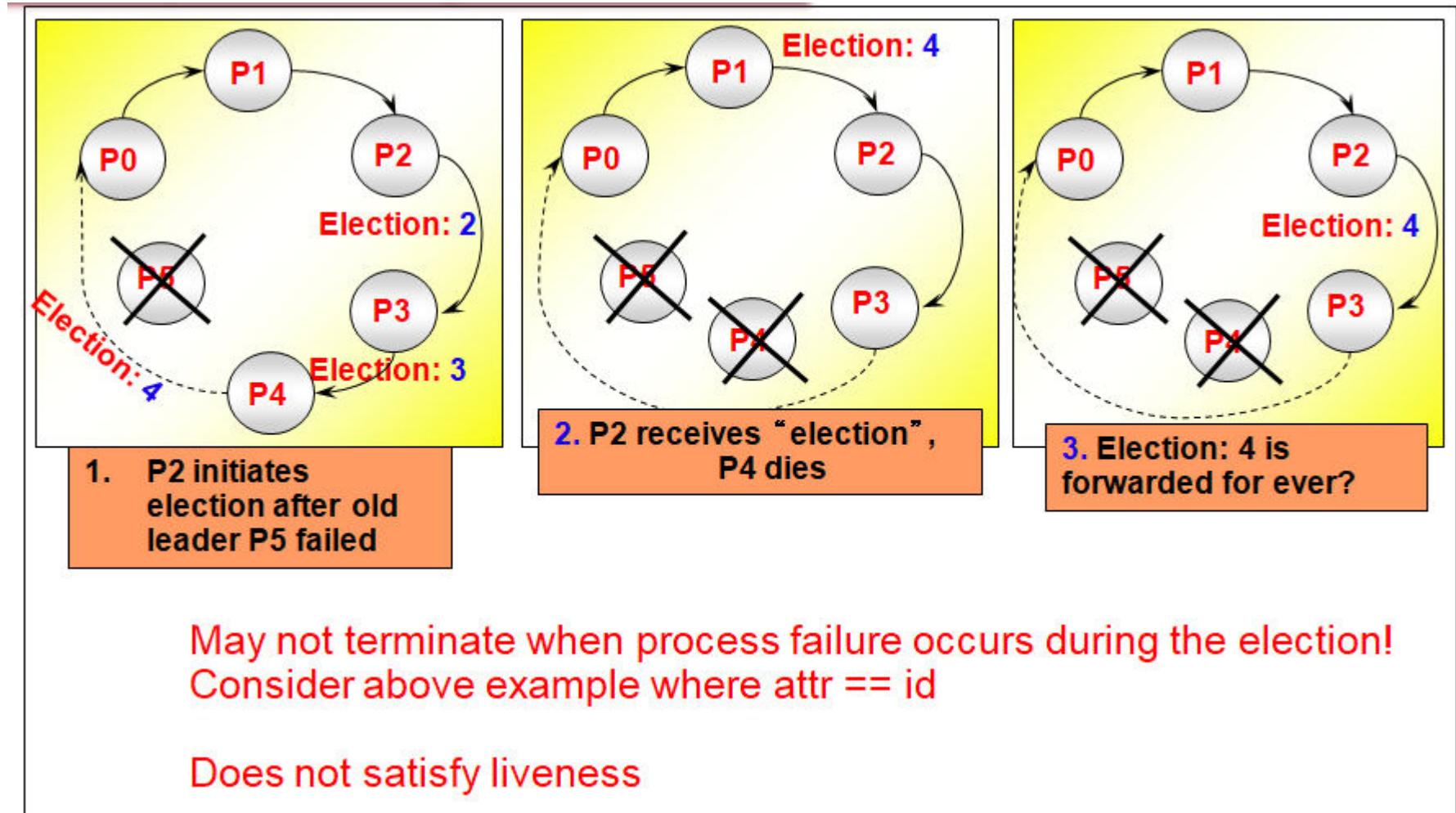
- The election was started by process 17. The highest process identifier encountered so far is 24.(final leader will be 33)
- The worst-case scenario occurs when the counter-clockwise neighbor (@ the initiator) has the highest attr.
  - In a ring of N processes, in the worst case:
  - A total of  $N-1$  messages are required to reach the new coordinator-to-be(election messages).
  - Another  $N$  messages are required until the new coordinator-to-be ensures it is the new coordinator (election messages – no changes).
  - Another  $N$  messages are required to circulate the elected messages.
  - Total Message Complexity =  $3(N-1)$**
  - Turnaround time =  $3(N-1)$**



In the previous example

- no failures happen during the run of the election algorithm.
- Safety and Liveness are satisfied.

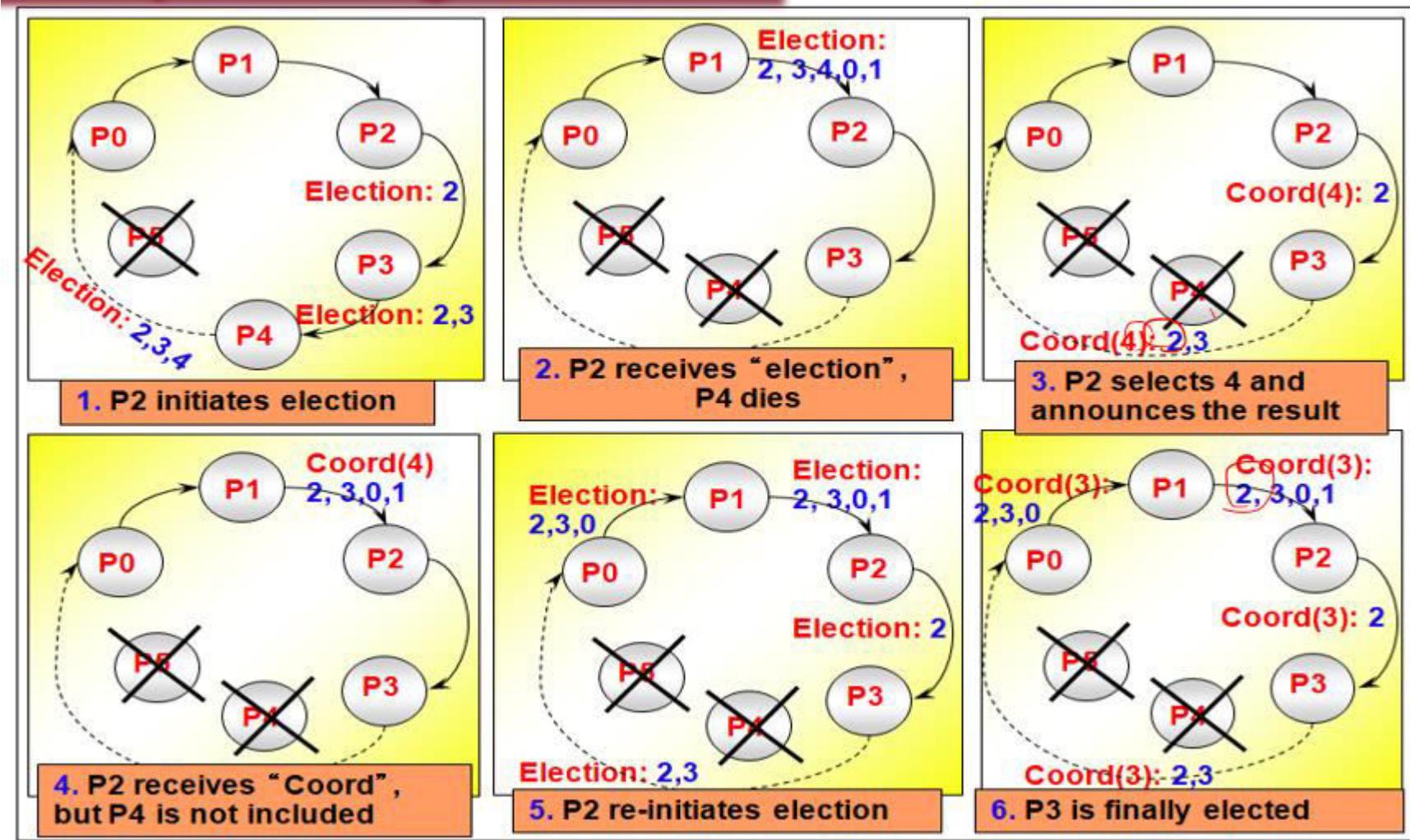
**What happens if there are failures during the election run?**



## Modified Ring Election

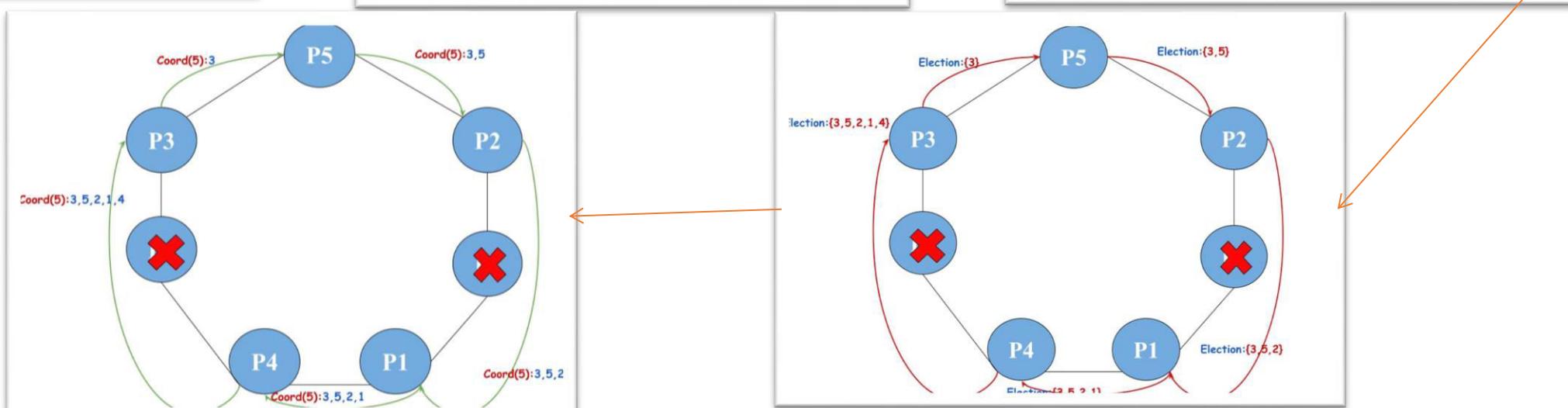
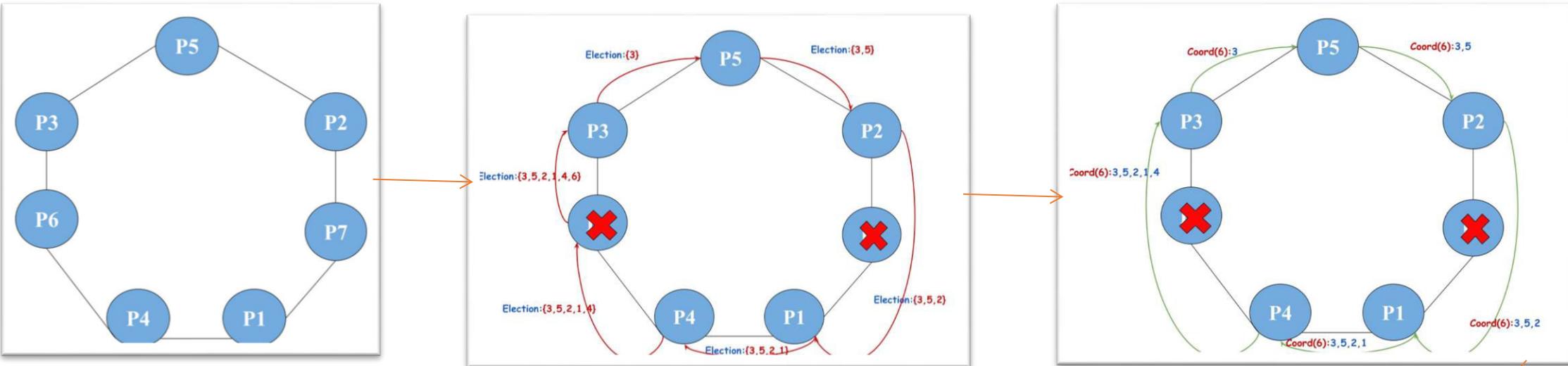
---

- Processes are organized in a logical ring.
- Any process that discovers the coordinator (leader) has failed, initiates an “election” message.
- The message is circulated around the ring, bypassing failed processes.
- Each process appends (adds) its id:attr to the message as it passes it to the next process (without overwriting what is already in the message)
- Once the message gets back to the initiator, it elects the process with the best election attribute value.
- It then sends a “coordinator” message with the id of the newly-elected coordinator. Again, each process adds its id to the end of the message and records the coordinator id locally.
- Once “coordinator” message gets back to initiator,
  - election is over if would-be-coordinator’s id is in id-list.
  - else the algorithm is repeated (handles election failure).



## Practice Problem on Ring Election Algorithm

**Problem:** In the given system, Process 7 fails and Process 3 initiates the election. After Process 3 announces Process 6 as the new Leader, Process 6 also fails. Show the flow of events that take place?





**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



# CLOUD COMPUTING

## Leader election: Bully Algorithm

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

- The Election problem is a ***consensus problem***
- Consensus is impossible to solve with 100% guarantee in an asynchronous system
  - Leader election in asynchronous model
- Problem associated with Modified Ring election
  - $P_i$  may just be very slow, but not faulty (yet it is not elected as leader!)
  - Slow initiator
  - Ring reorganization

- Application Context
  - A distributed System where every process can send messages to every other process in the system. All the processes know other processes ids.

Why is it called as Bully Algorithm?

- When a process with the next highest ID (after the current leader) detects the leader failure, it elects itself as the new leader.
- It sends a coordinator message to other processes with lower identifiers
- At this point, the election is completed. Therefore, this is called a bully algorithm.

- When a process wants to initiate an election
  - If it knows its id is the highest
    - It elects itself as coordinator, then sends a Coordinator message to all processes with lower identifiers. Election is completed.
  - Else
    - it initiates an election by sending an Election message
    - Sends it to only processes that have a higher id than itself.
    - if receives no answer within timeout, calls itself leader and sends Coordinator message to all lower id processes. Election completed.
    - if an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another timeout, start a new election run.
- A process that receives an Election message replies with disagree message, and starts its own leader election protocol (unless it has already done so)

## Bully Algorithm- Timeout Values

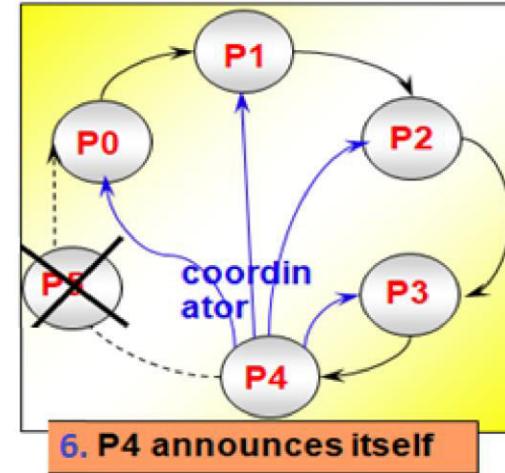
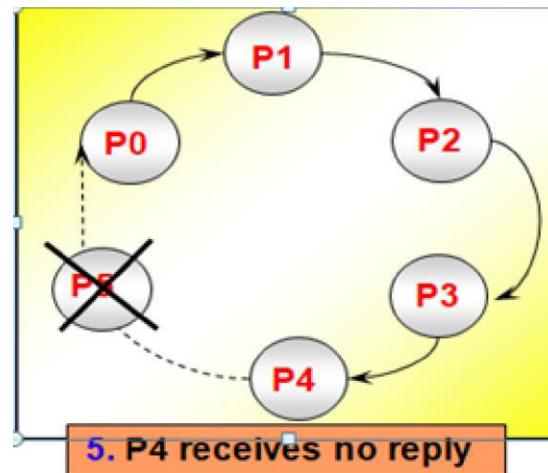
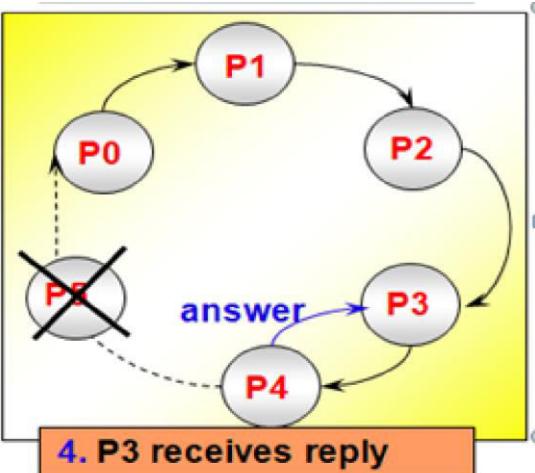
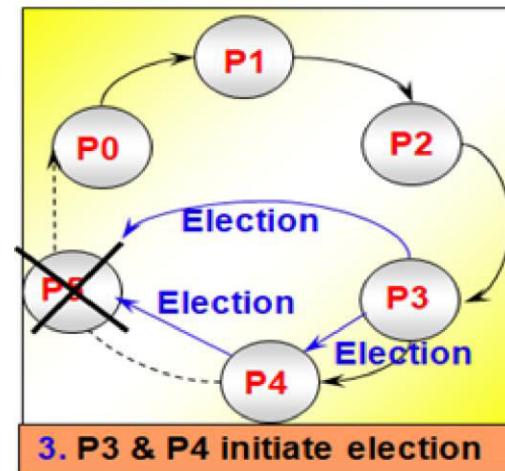
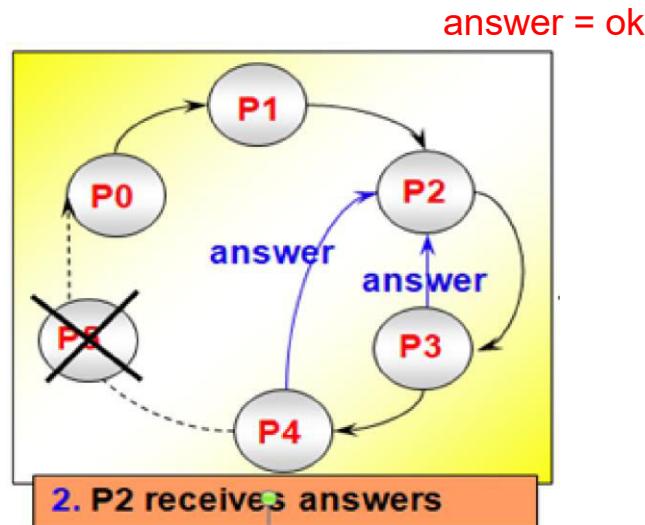
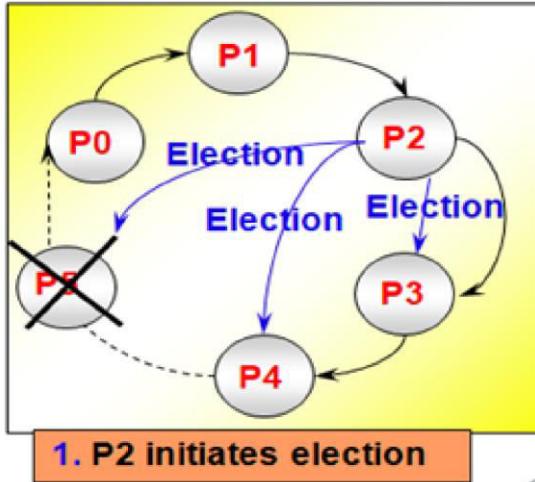
---

- Assume the one-way message transmission time (T) is known.
- First timeout value (when the process that has initiated election waits for the first response)
  - Must be set as accurately as possible.
    - If it is too small, a lower id process can declare itself to be the coordinator even when a higher id process is alive.
    - What should be the first timeout value be, given the above assumption?
      - $2T + (\text{processing time}) \approx 2T$
  - When the second timeout happens (after ‘disagree’ message), election is restarted.
    - A very small value will lead to extra “Election” messages.
    - A suitable option is to use the worst-case turnaround time.

### Assumptions:

- **Synchronous system**
  - All messages arrive within  $T_{trans}$  units of time.
  - A reply is dispatched within  $T_{process}$  units of time after the receipt of a message.
  - if no response is received in  $2T_{trans} + T_{process}$ , the process is assumed to be faulty (crashed).
- attr == id
- Each process knows all the other processes in the system (and thus their id's)

## Bully Algorithm Election Example



## Steps in Bully Algorithm:

---

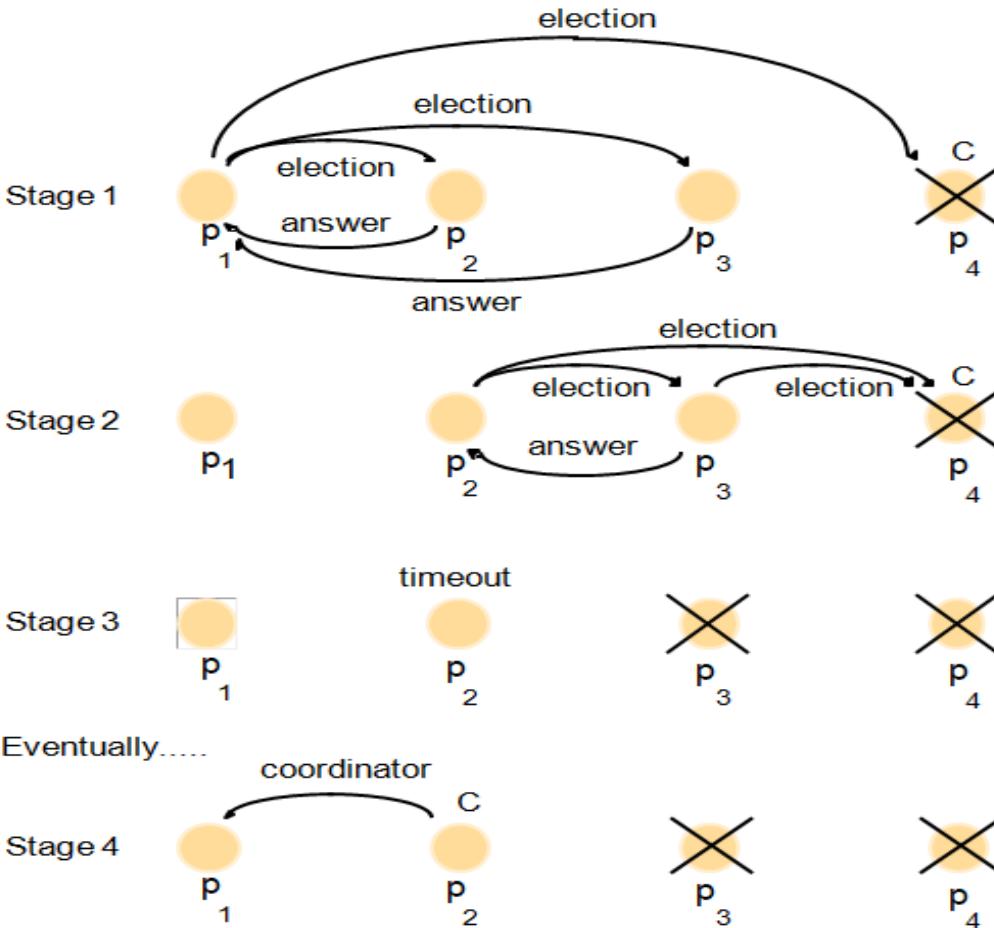
1. We start with 5 processes, which are connected to each other. Process 5 is the leader, as it has the highest number.
2. Process 5 fails.
3. Process 2 notices that Process 5 does not respond. Then it starts an election, notifying those processes with ids greater than 2.
4. Then Process 3 and Process 4 respond, telling Process 2 that they'll take over from here.
5. Process 3 sends election messages to the Process 4 and Process 5.
6. Only Process 4 answers to process 3 and takes over the election.
7. Process 4 sends out only one election message to Process 5.
8. When Process 5 does not respond Process 4 ,then it declares itself the winner.

### Failures During Election Run:

You can also have failures that happen during the election run, so P4 might fail after it has sent the coordinator, after it has sent the election message, but before it sends the coordinator message.

## *The Bully Algorithm with Failures*

The coordinator  $p_4$  fails and  $p_1$  detects this



## Analysis of Bully Algorithm

**Worst case scenario:** When the process with the lowest id in the system detects the failure.

- N-1 processes altogether begin elections, each sending messages to processes with higher ids.  
    i-th highest id process sends i-1 election messages
- The message overhead is  $O(N^2)$ .
- Turnaround time is approximately 5 message transmission times if there are no failures during the run:
  1. Election message from lowest id process
  2. Answer to lowest id process from 2<sup>nd</sup> highest id process
  3. Election from 2nd highest id process
  4. Timeout for answers @ 2nd highest id process
  5. Coordinator message from 2<sup>nd</sup> highest id process

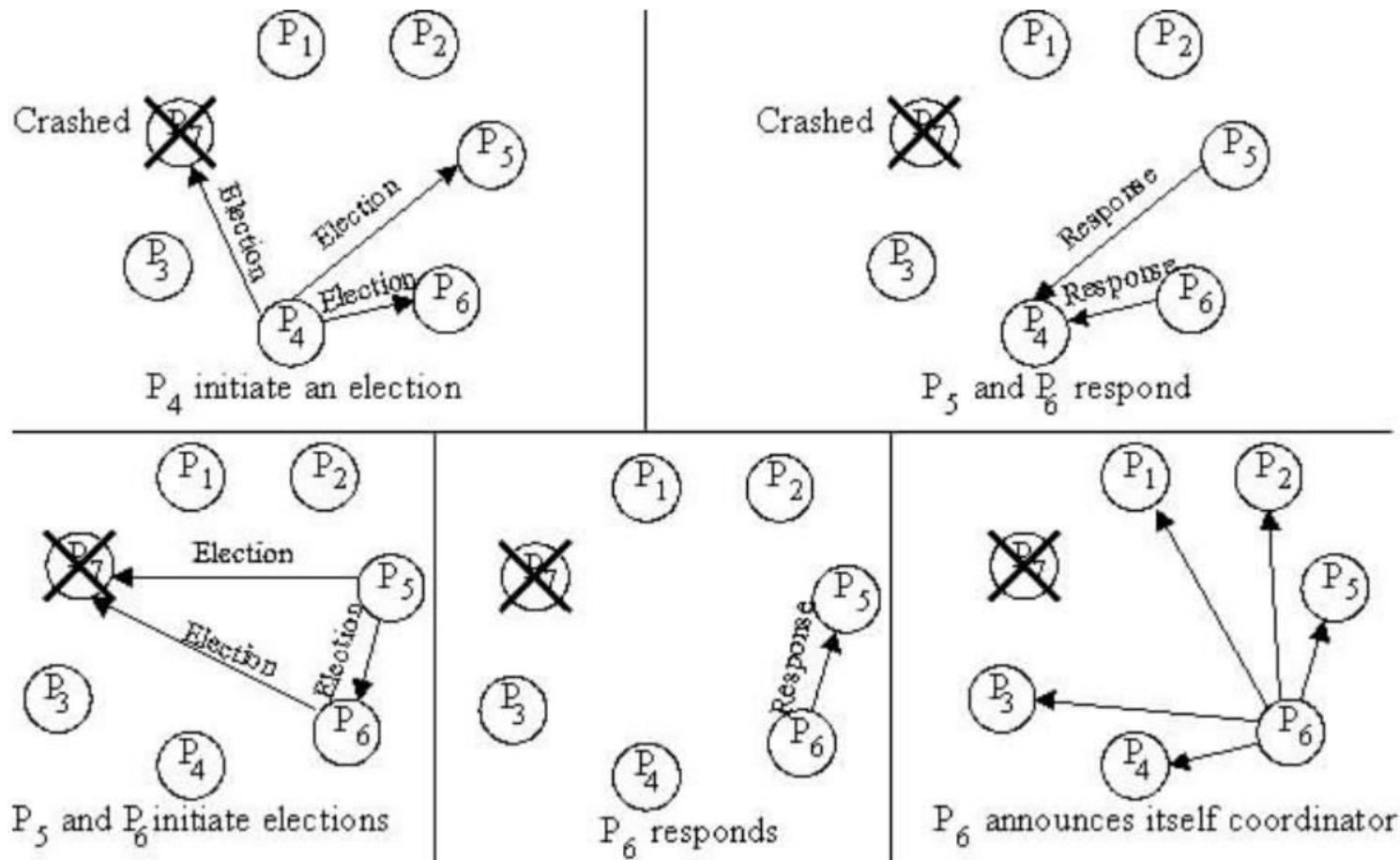
**Best case scenario:** The process with the second highest id notices the failure of the coordinator and elects itself.

- N-2 coordinator messages are sent.
- Turnaround time is one message transmission time.

## Practice Problem on Bully Algorithm

Out of 7 processes, P7 fails and P4 detects this first. How many election messages and how many response messages are sent if all other Processes function ideally and do not fail?

**Solution:** 6 Election Messages (3 by P4, 2 by P5 and 1 P6) and 3 Response Message (P5 & P6 to P4 and P6 to P5)



# CLOUD COMPUTING

## Summary

---

- Coordination in distributed systems requires a leader process
- Leader process might fail
- Need to (re-) elect leader process
- Three Algorithms
  - Ring algorithm
  - Modified Ring algorithm
  - Bully Algorithm

## Additional References

---

1. <https://courses.grainger.illinois.edu/ece428/sp2021//assets/slides/lect12-after.pdf>  
[https://onlinecourses.nptel.ac.in/noc21\\_cs15/unit?unit=31&lesson=32](https://onlinecourses.nptel.ac.in/noc21_cs15/unit?unit=31&lesson=32)
2. <https://www.coursera.org/learn/cloud-computing-2/lecture/K8QwJ/1-4-bully-algorithm>
3. <https://www.cs.colostate.edu/~cs551/CourseNotes/Synchronization/BullyExample.html>



# THANK YOU

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



# CLOUD COMPUTING

## Resource Management and Scheduling

---

**Dr. Prafullata Kiran Auradkar**

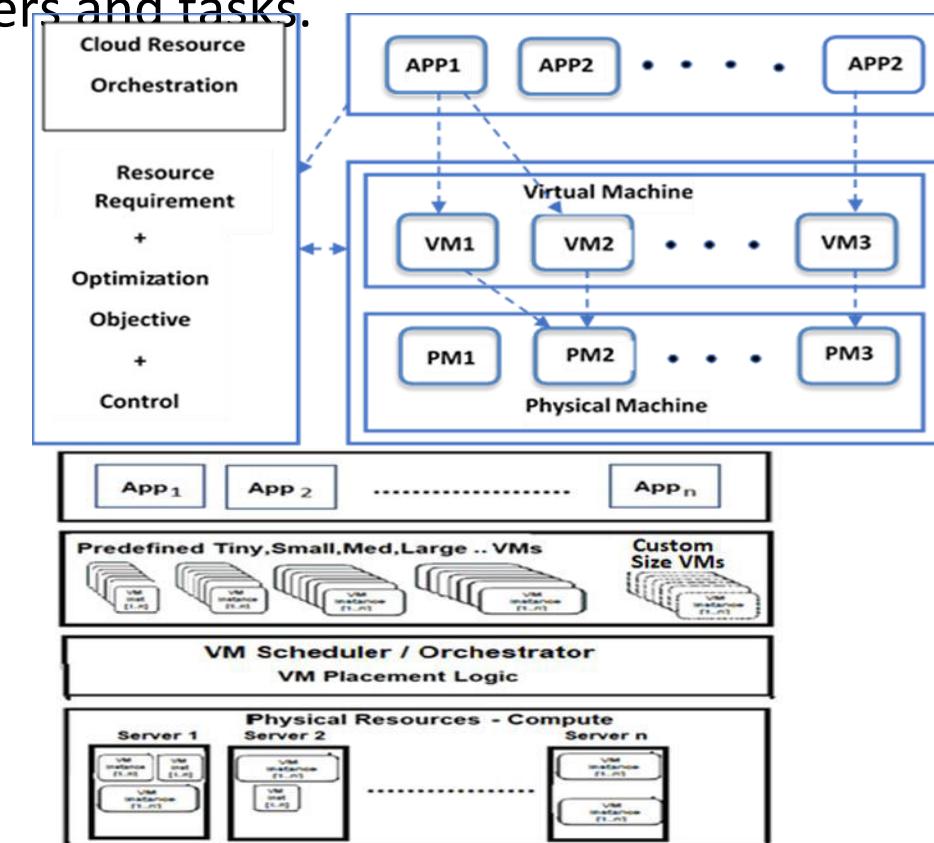
Department of Computer Science and Engineering

### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

## Introduction Resource Management and Scheduling

- Scheduling
  - Deciding how to allocate resources of a system, such as CPU cycles, memory, secondary storage space, I/O and network bandwidth, between users and tasks.
- Critical Function that affects criterion of
  - Functionality
  - Performance
  - Cost
- Involves policies and mechanisms for resource allocation.
  - Policy -> principles guiding decisions.
  - Mechanisms -> the means to implement policies



PM – Physical Machine VM – Virtual Machine

## Resource Management and Scheduling : Motivation

---

Cloud resource management is complex

- Multi-objective optimization - Policy and decisions are difficult
- It is impossible to have accurate global state information.
- System failures, attacks makes system unpredictable
- Cloud service providers are faced with large fluctuating loads which challenge the claim of cloud elasticity.
- Resource Management Strategies are different for different System (IaaS, PaaS, and SaaS)

### 1. Admission control :

- Prevent the system from accepting workload in violation of high-level system policies.

### 2. Capacity allocation :

- Allocate resources based on the need in terms of size - provisioning of different fixed, variable sizes

### 3. Optimization and Load balancing :

- Distribute the workload evenly among the servers for optimization.

### 4. Energy optimization :

- Minimization of energy consumption through Choice of components, Operating modes etc.

### 5. Quality of service (QoS) guarantees : ability to satisfy conditions specified by a Service Level Agreement.

### 6. Locale based : Location based legal compliance, performance etc.

### 7. Cost: Based on perceived value or auctions

## Mechanisms for the implementation of resource management policies

---

1. **Control theory** : These approaches use feedback in terms of the system utilization, latencies, time for scheduling etc. to control the allocation of resources to guarantee system stability and predict transient behavior.
2. **Utility-based** : This requires a performance model and a mechanism to correlate user-level performance with cost. Considering the SLA ( with rewards and specifies the rewards as well as penalties associated with specific performance metrics using the average response times, the allocations are done.
3. **Machine learning** : Uses historic data and predicts the resource requirements
4. **Market-oriented/economic Oriented** : Mechanisms such as Combinatorial auctions are used by the users to pay for the resources.

## Cloud Resource management involves Tradeoffs

---

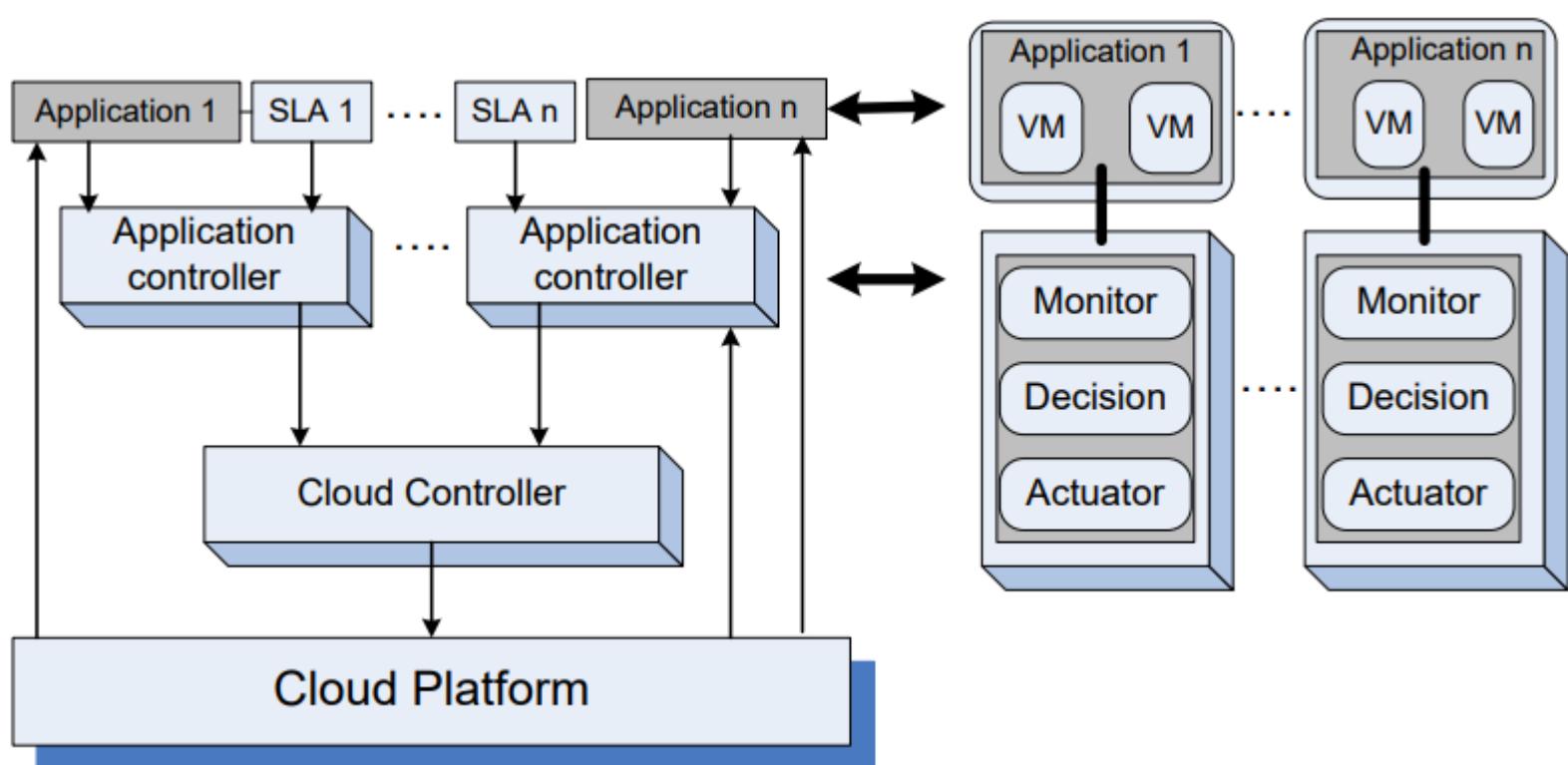
- Load may be concentrated on a few nodes instead of balancing the load across the cluster
- There is a large relationship between the frequency of the clock and the performance of the system. It decreases at a lower rate. Similarly, the case with Voltage and energy where the energy consumed reduces with the reduction in Voltage

CPU speed (GHz)	Normalized energy (%)	Normalized performance (%)
0.6	0.44	0.61
0.8	0.48	0.70
1.0	0.52	0.79
1.2	0.58	0.81
1.4	0.62	0.88
1.6	0.70	0.90
1.8	0.82	0.95
2.0	0.90	0.99
2.2	1.00	1.00

## Application of Control Theory for Cloud Resource Management - Illustration

The main components of a control system:

- The inputs -> workload, the policies, the capacity allocation, the load balancing, the energy optimization, and the QoS guarantees
- The control system components -> sensors used to estimate relevant measures of performance and controllers which implement various policies.
- The outputs -> the resource allocations to the individual applications.



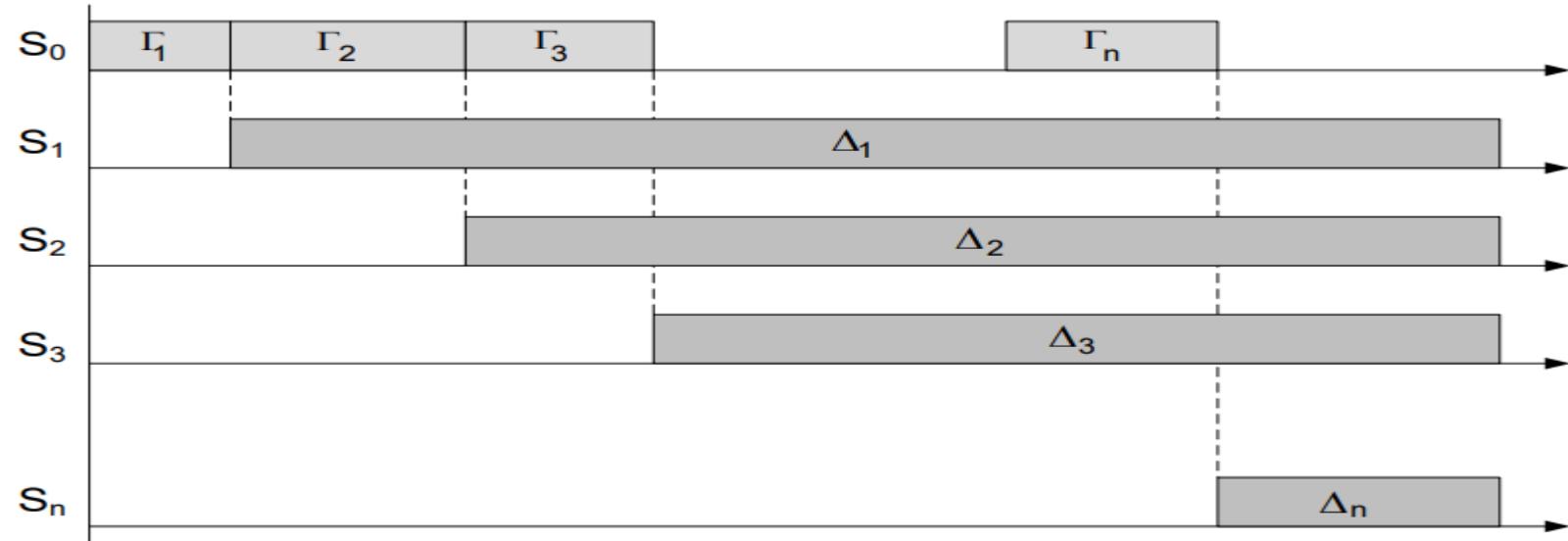
- Scheduling : responsible for resource sharing at several levels:
  - A server can be shared among several virtual machines.
  - A virtual machine could support several applications.
  - An application may consist of multiple threads.
- A scheduling algorithm should be efficient, fair, and starvation-free.
- The objectives of a scheduler:
  - Batch system : maximize throughput and minimize turnaround time.
  - Real-time system : meet the deadlines and be predictable.
- Best-effort: batch applications and analytics.
- Common algorithms for best effort applications:
  - Round-robin.
  - First-Come-First-Serve (FCFS).
  - Shortest-Job-First (SJF).
  - Priority algorithms.

## Cloud Scheduling subject to deadlines

---

- Hard deadlines :
  - If the task is not completed by the deadline, other tasks which depend on it may be affected and there are penalties;
  - a hard deadline is strict and expressed precisely as milliseconds, or possibly seconds
  - This would be for number of cores, memory, type of VM
- Soft deadlines :
  - More of a guideline and, in general, there are no penalties; soft deadlines can be missed by fractions of the units used to express them
    - e.g., minutes if the deadline is expressed in hours, or hours if the deadlines is expressed in days.
  - We consider only aperiodic tasks with arbitrarily divisible workloads
  - E.g. Energy efficiencies, affinity etc

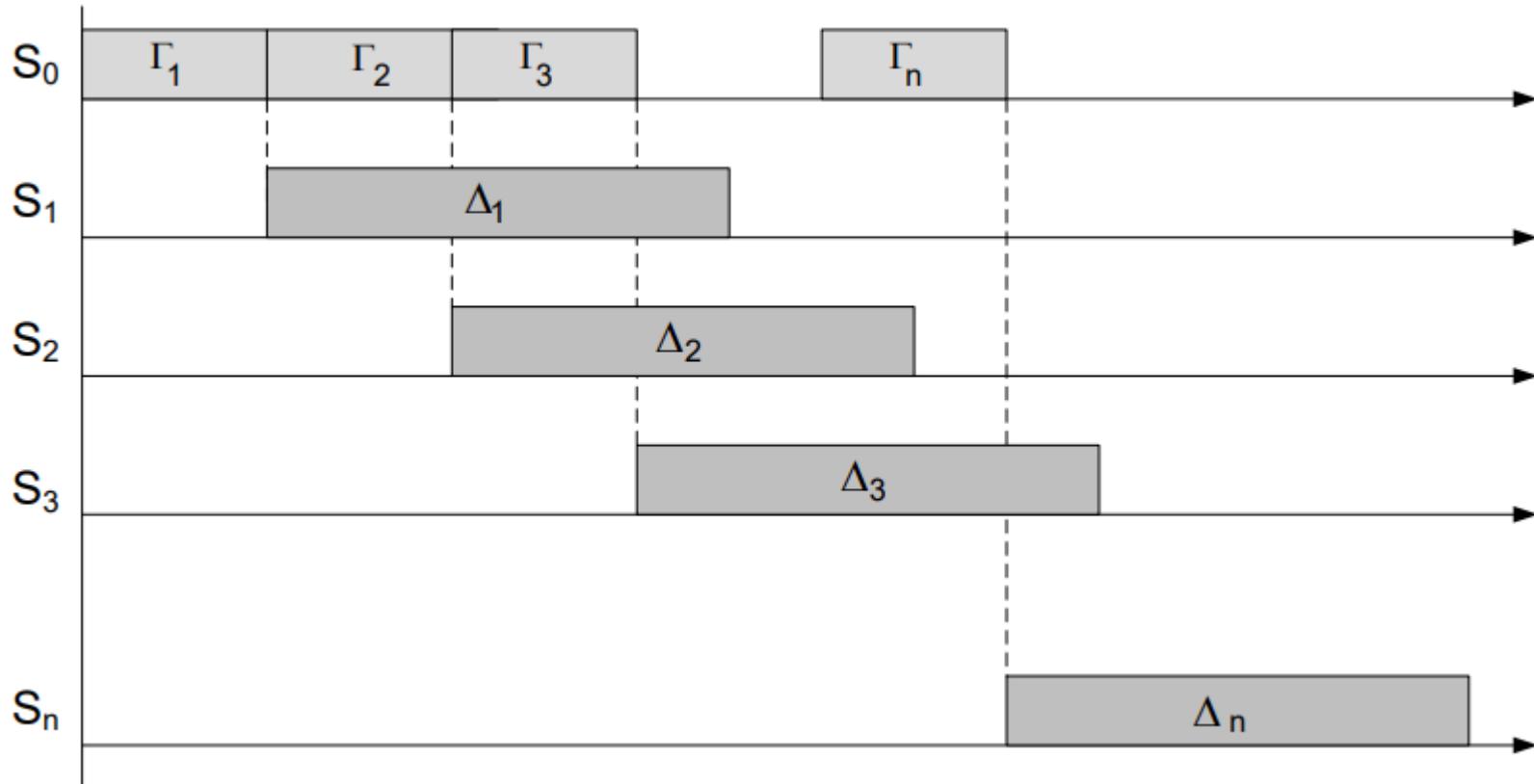
- Optimal Partitioning Rule (OPR) : the workload is partitioned to ensure the earliest possible completion time and all tasks are required to complete at the same time.
- Equal Partitioning Rule (EPR) : assigns an equal workload to individual worker nodes.



- The timing diagram for the Optimal Partitioning Rule; the algorithm requires worker nodes to complete execution at the same time. The head node,  $S_0$ , distributes sequentially the data to individual worker nodes.

## Workload Partition rules

- Equal Partitioning Rule (EPR) : assigns an equal workload to individual worker nodes.

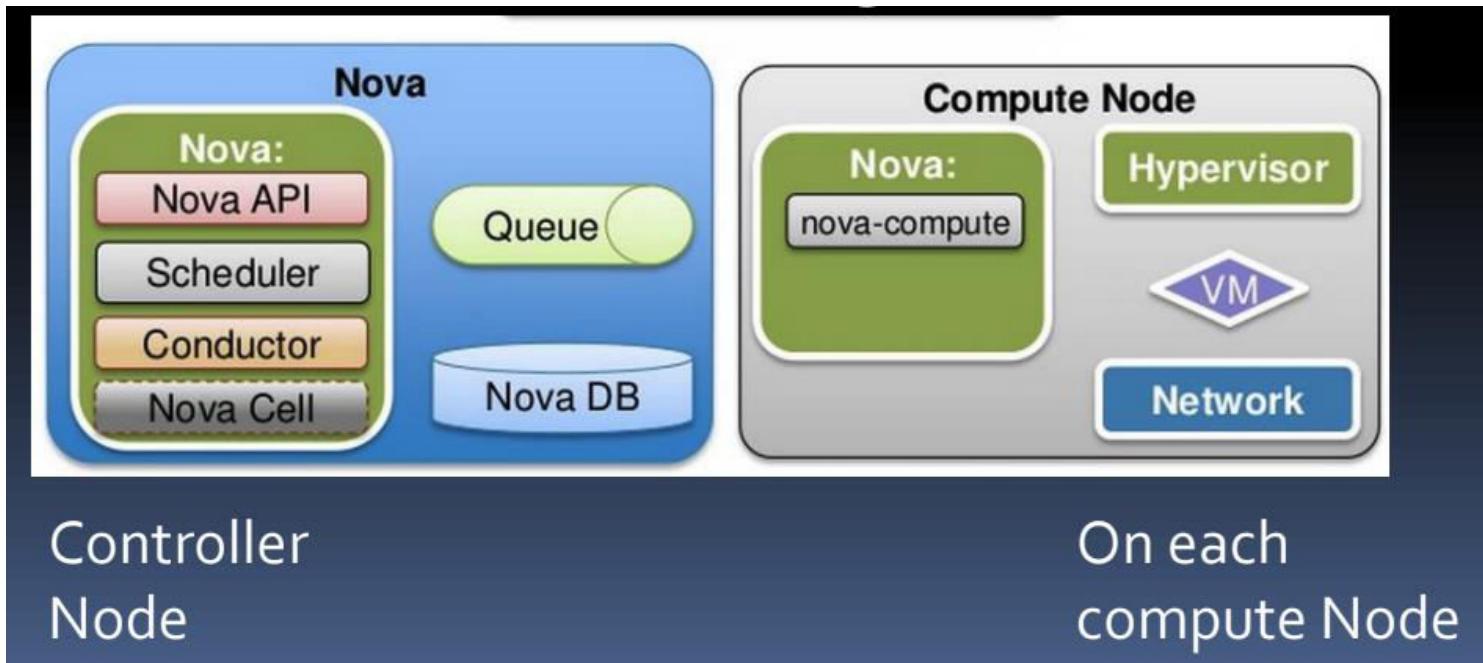


- The timing diagram for the Equal Partitioning Rule; the algorithm assigns an equal workload to individual worker nodes.

## Illustration - How does Nova allocate VMs?

When the user asks for a VM instance, how it is allocated?

Compute uses the **nova-scheduler** service to determine how to dispatch compute requests. For example, the **nova-scheduler** service determines on which host a VM should launch.



Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

1. Choose AMI	2. Choose Instance Type	3. Configure Instance	4. Add Storage	5. Add Tags	6. Configure Security Group														
<b>AMI Details</b> <span style="float: right;">Edit AMI</span> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">  <b>Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-08e0ca9924195beba</b>            Free tier eligible         </div> Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Gilbc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is a... Root Device Type: ebs Virtualization type: hvm																			
<b>Instance Type</b> <span style="float: right;">Edit instance type</span> <table border="1"> <thead> <tr> <th>Instance Type</th> <th>ECUs</th> <th>vCPUs</th> <th>Memory (GiB)</th> <th>Instance Storage (GB)</th> <th>EBS-Optimized Available</th> <th>Network Performance</th> </tr> </thead> <tbody> <tr> <td>t2.micro</td> <td>-</td> <td>1</td> <td>1</td> <td>EBS only</td> <td>-</td> <td>Low to Moderate</td> </tr> </tbody> </table>						Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	t2.micro	-	1	1	EBS only	-	Low to Moderate
Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance													
t2.micro	-	1	1	EBS only	-	Low to Moderate													
<b>Security Groups</b> <span style="float: right;">Edit security groups</span> Security group name: launch-wizard-1 Description: launch-wizard-1 created 2021-02-07T22:57:35.731+05:30																			
Type: SSH Protocol: TCP Port Range: 22 Source: 0.0.0.0/0																			
<b>Instance Details</b> <span style="float: right;">Edit instance details</span>																			
<b>Storage</b> <span style="float: right;">Edit storage</span>																			
<b>Tags</b> <span style="float: right;">Edit tags</span>																			

**Cancel** **Previous** **Launch**

# CLOUD COMPUTING

## Illustration - How does Nova allocate VMs?

1. Choose AMI   2. Choose Instance Type   3. Configure Instance   4. Add Storage   5. Add Tags   6. Configure Security Group

### Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

**AMI Details** [Edit AMI](#)

 **Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-08e0ca9924195beba**  
Free tier eligible  
Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is a...  
Root Device Type: ebs Virtualization type: hvm

**Instance Type** [Edit instance type](#)

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

**Security Groups** [Edit security groups](#)

**Security group name** launch-wizard-1  
**Description** launch-wizard-1 created 2021-02-07T22:57:35.731+05:30

Type <small>i</small>	Protocol <small>i</small>	Port Range <small>i</small>	Source <small>i</small>	Description <small>i</small>
SSH	TCP	22	0.0.0.0/0	

**Instance Details** [Edit instance details](#)

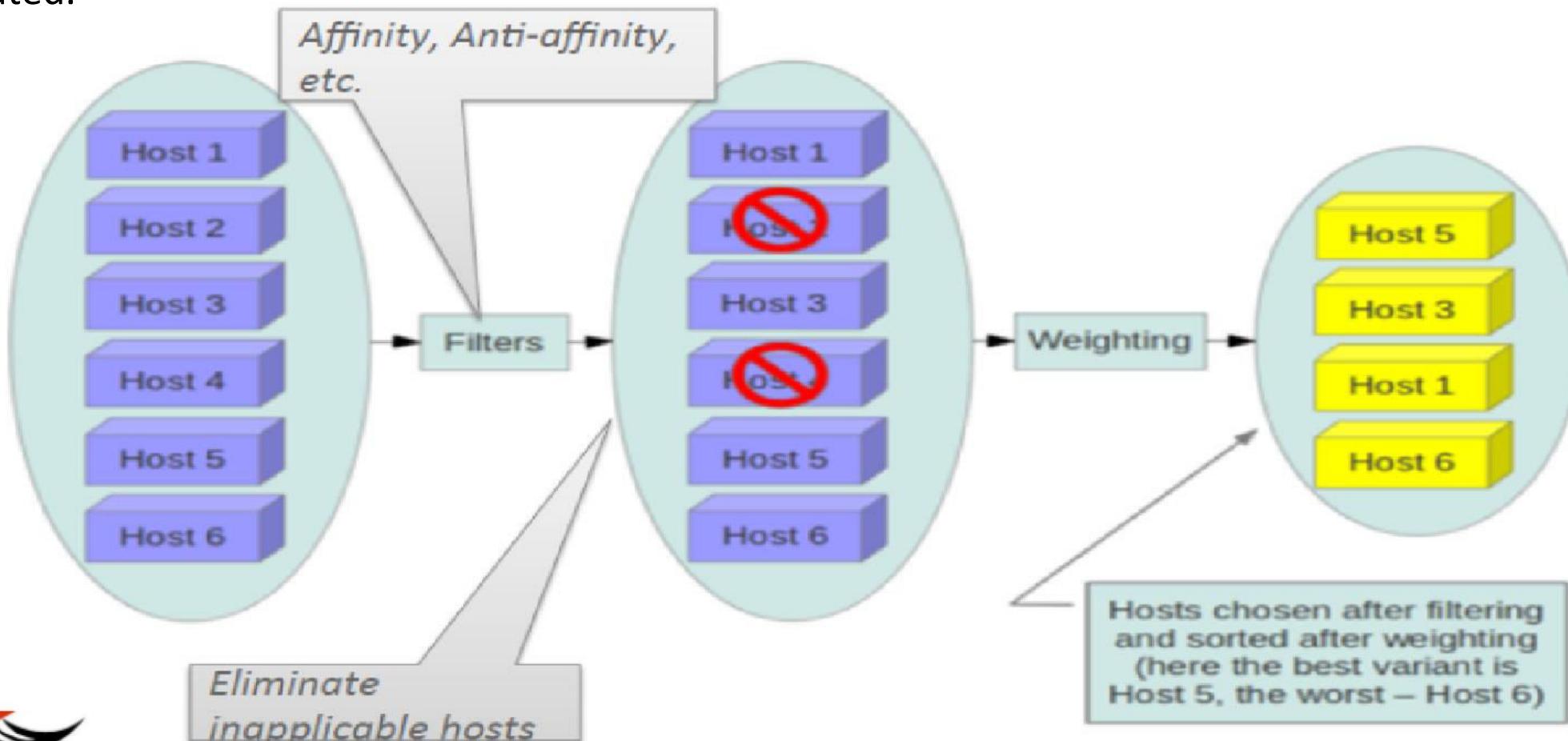
**Storage** [Edit storage](#)

**Tags** [Edit tags](#)

[Cancel](#) [Previous](#) [Launch](#)

## Illustration - How does Nova allocate VMs?

In the context of filters, the term host means a physical node that has a nova-compute service running on it. You can configure the scheduler through a variety of options. The scheduler chooses a new host when an instance is migrated.





**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



# CLOUD COMPUTING

## Distributed Locking

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

## Distributed System: Truth is defined by majority

---

- A node cannot trust its own judgment
- A distributed system cannot rely only on a single node
  - Nodes may fail anytime
- Distributed Systems **rely on the decisions of a quorum**
  - Decisions are based on the votes of minimum number of nodes
  - Thus reduces dependency on one node

## What is Distributed Locking

---

- Nodes must **acquire a lock** before trying to lock data
- Acquiring a lock gives exclusive access to the data.
- Once the lock is acquired, node ***performs the operations*** and ***release the lock***
- ***acquire, operate, release*** sequence is well understood in the context of **shared-memory among threads**
- In distributed locking
  - Different clients running on different machines must be taken into account, instead of multiple threads
  - One must be able to lock ***different clients running on different machines***
- Distributed locks are **mutual exclusion locks** that have global presence.
- Distributed locks on resources, prevents logical failures that may be caused by resource contention

## Advantages of Distributed Locking

---

“Locking often isn’t a good idea, and trying to lock something in a distributed environment may be more dangerous.”

Then why use locks?

1. Locking *ensures that only one node does the work at a time among the several that are trying participate*
  - Work might be to write some data to a shared storage system, to perform some computation, to call some external API, or such like.
2. **Efficiency: Locking can save software from performing useless work more times than it is really needed**
  - some expensive computation like triggering a timer twice
3. If the lock fails and two nodes end up doing the same piece of work, the result is a minor increase in cost (you end up paying ₹5 more to AWS than you otherwise would have) or a minor inconvenience (e.g. a user ends up getting the same email notification twice).
4. **Correctness : A lock can prevent the concurrent processes of the same data, avoiding data corruption, data loss, inconsistency etc.**
  - Locks avoid concurrency - No stepping on each others' toes, NO messing up the state of the system.
  - Failure of locks - Result in a corrupted file, data loss, permanent inconsistency,

- **Mutual Exclusion :**
  - Only one client (globally) can hold a lock at a given moment.
- **Deadlock free :**
  - Distributed locks use a **lease-based locking mechanism**.
    - If a client acquires a lock and encounters an exception, the lock is automatically released after a certain period.
- **Consistency :**
  - Failovers may be triggered by external errors or internal errors.
    - hardware failures and network exceptions (external)
    - Slow queries and system defects. (internal)
  - After failovers are triggered, a replica serves take over
  - The lock of the client must remain the same after the failovers.

## Types of distributed locks

---

- **Optimistic:**

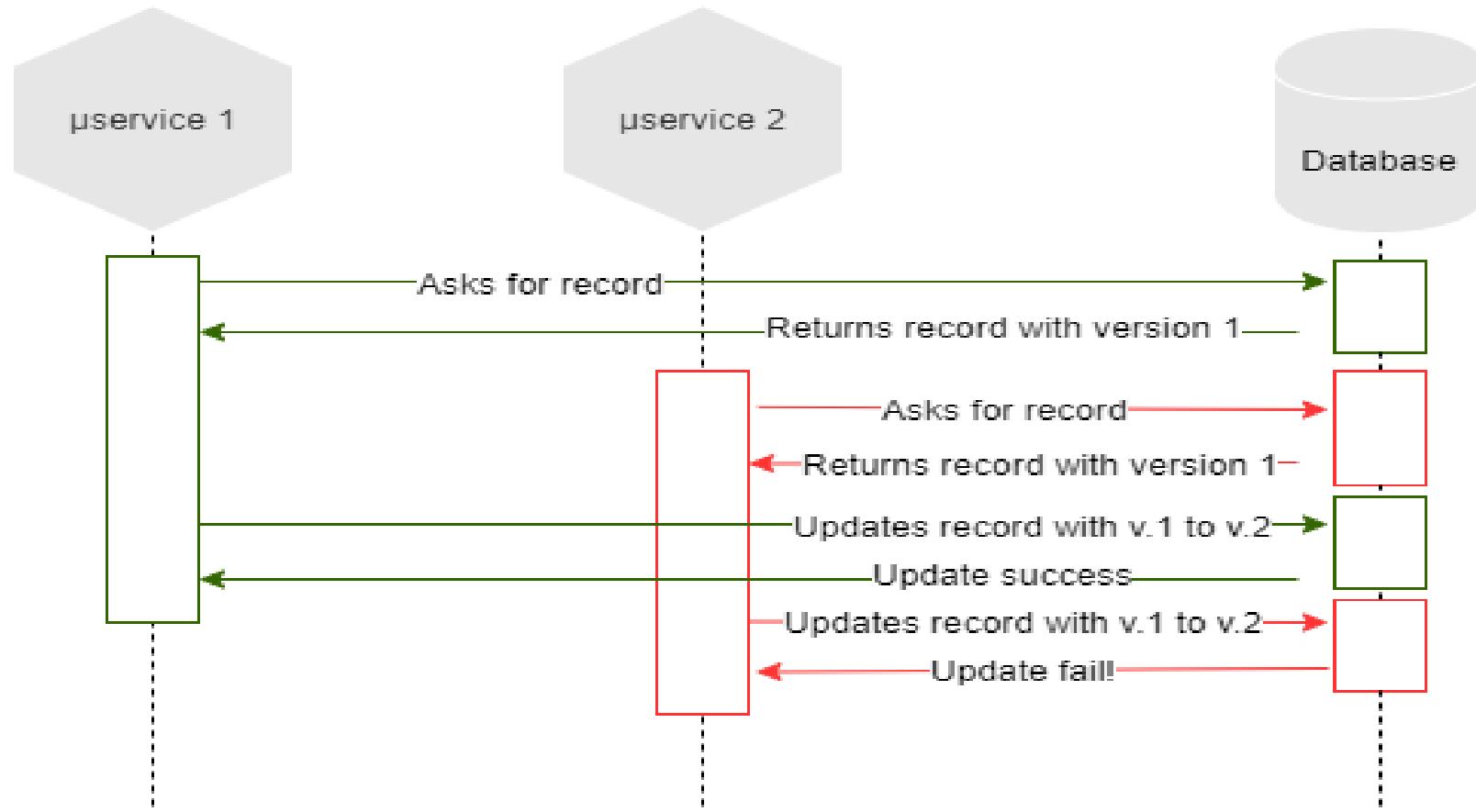
Instead of blocking something potentially dangerous happens, we continue anyway, in the hope that everything will be ok.
- **Pessimistic:**

Block access to the resource before operating on it, and then release the lock at the end.

## Types of Locks : Optimistic Locks

### Optimistic:

Instead of blocking something potentially dangerous happens, we continue anyway, in the hope that everything will be ok.



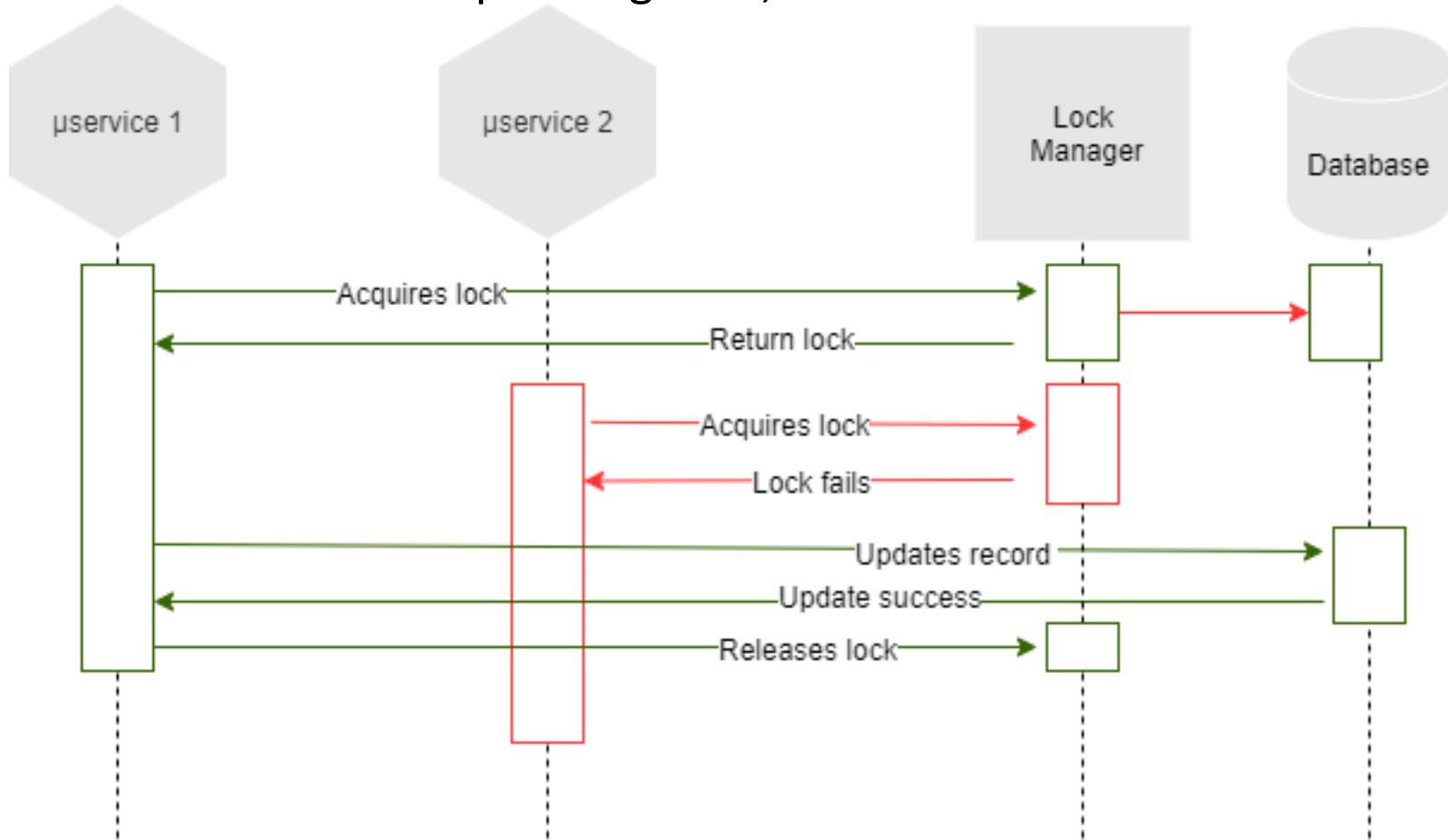
Check if the record was updated by someone else before the commit operation

Optimistic lock sequence diagram

## Types of Locks : Pessimistic Locks

**Pessimistic:**

Block access to the resource before operating on it, and then release the lock at the end.



Take an exclusive lock so that no one else can modify the record.

Pessimistic lock sequence diagram

- Distributed Lock is more complicated than the Mutex Lock in multi threaded system
  - Different Node / Network failures can occur independently
- Consider an application in which a client needs to update a file in shared storage (e.g. HDFS or S3).
- A client first acquires the lock, then reads the file, makes some changes, writes the modified file back, and finally releases the lock.
- The lock prevents two clients from performing this read-modify-write cycle concurrently, which would result in lost updates.
- A code that might be written to perform this is shown on the next slide.

## Implementing Distributed Locking

```
// THIS CODE IS BROKEN
function writeData(filename, data) {
    var lock = lockService.acquireLock(filename);
    if (!lock) {
        throw 'Failed to acquire lock';
    }

    try {
        var file = storage.readFile(filename);
        var updated = updateContents(file, data);
        storage.writeFile(filename, updated);
    } finally {
        lock.release();
    }
}
```

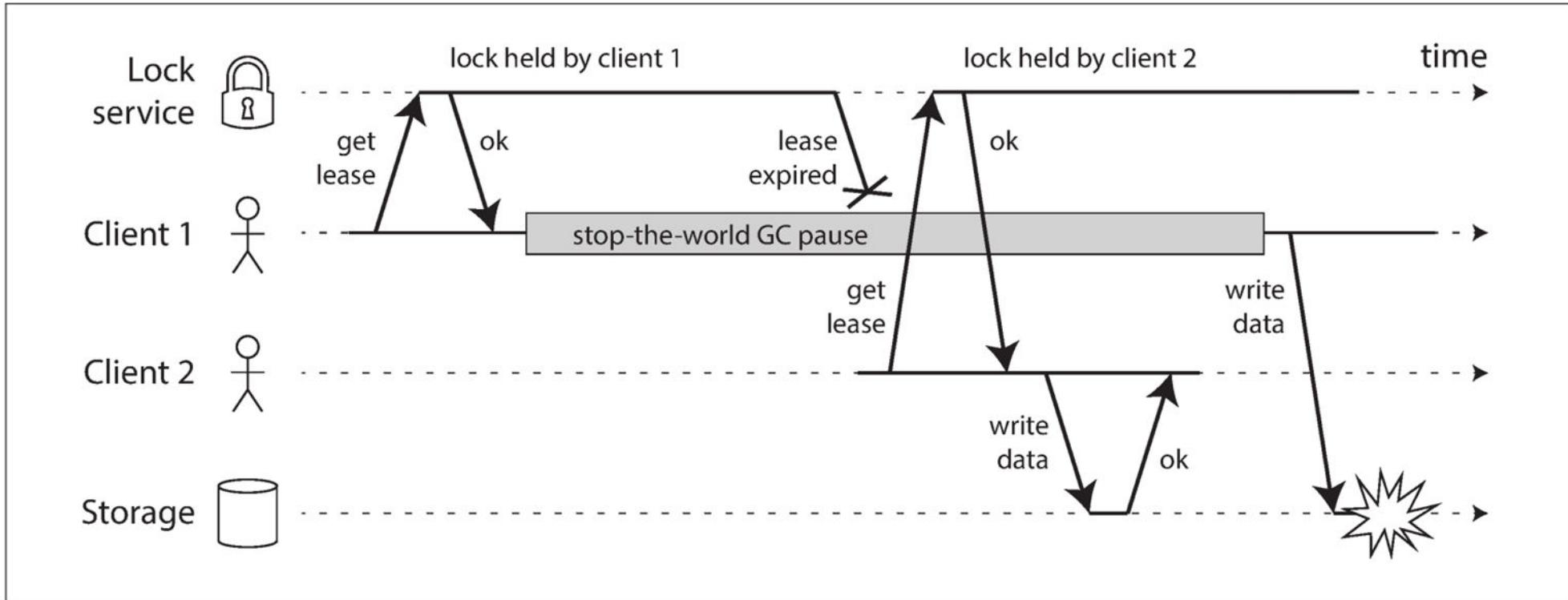


Figure 8-4. Incorrect implementation of a distributed lock: client 1 believes that it still has a valid lease, even though it has expired, and thus corrupts a file in storage.

NOTE: GC stands for Garbage Collector

## Implementing Distributed Locking

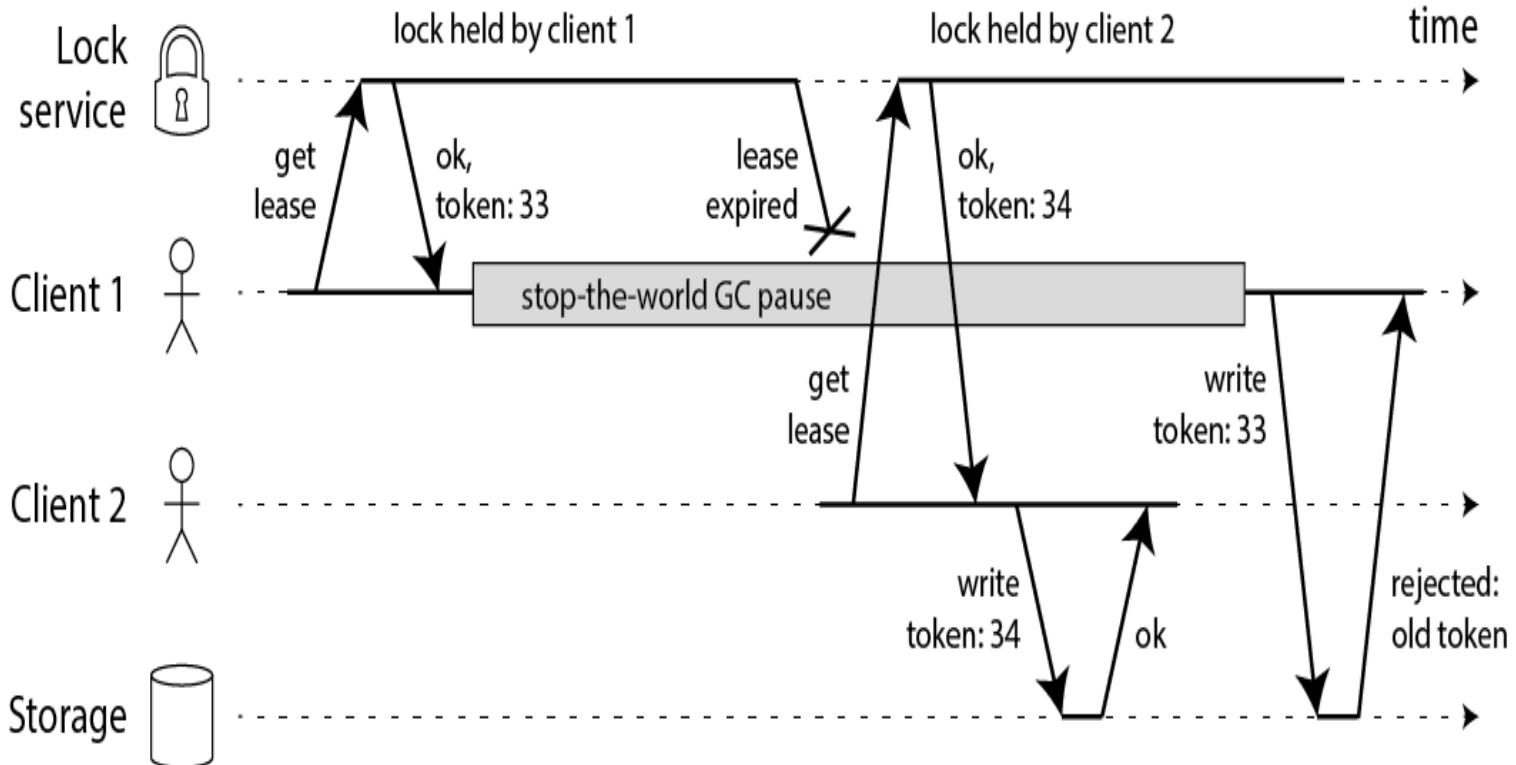
---

In this example,

- the client that acquired the lock is paused for an extended period of time while holding the lock for example because the garbage collector kicked in.
- The lock has a timeout (i.e. it is a lease), which is always a good idea (otherwise a crashed client could end up holding a lock forever and never releasing it).
- However, if the GC pause lasts longer than the lease expiry period, and the client doesn't realize that it has expired, it may go ahead and make some unsafe change.

NOTE: GC stands for Garbage Collector

## Implementing Distributed Locking with Fencing



### Fencing

When using a lock or lease to protect access to some resource, such as the file storage, ensure that a node under a false belief of being “the chosen one” cannot disrupt the rest of the system.

NOTE: GC stands for Garbage Collector

## Implementing Distributed Locking with Fencing

---

- Assume that every time the lock server grants a lock or lease, it also returns a *fencing token*, which is a number that increases every time a lock is granted
- Every time a client sends a write request to the storage service, it must include its current fencing token.
- Client 1 acquires the lease with a token of 33, but then it goes into a long pause and the lease expires.
- Client 2 acquires the lease with a token of 34 and then sends its write request to the storage service, including the token of 34.
- Later, client 1 comes back to life and sends its write to the storage service, including its token value 33.
- However, the storage server remembers that it has already processed a write with a higher token number (34), and so it rejects the request with token 33.

## Implementing Distributed Locking with Fencing

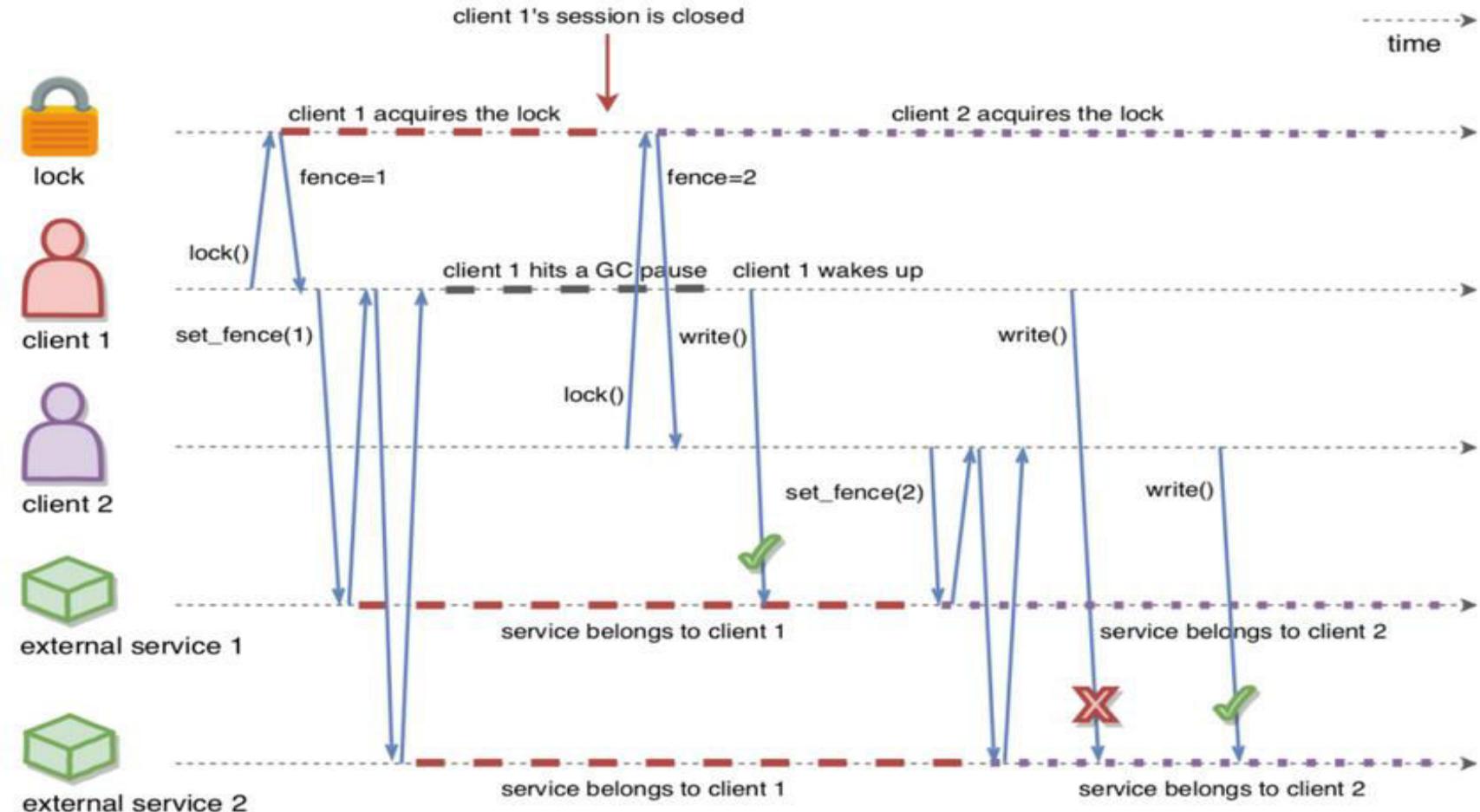


Figure 1: Using Fencing tokens to fence off stale lock holders

A distributed lock manager (DLM)

- Runs in every node with an identical copy of a **cluster-wide lock database**.
- DLM provides software applications a **means to synchronize their accesses to shared resources**.
- The DLM uses a generalized concept of a resource
  - Treats every resource as an entity, shared access to which must be controlled.
- This applies to a file, a record, an area of shared memory,
  - any resource that needs to shared

## Distributed Lock Manager

---

### DLM Implementation

- Google's Chubby,
  - a lock service for loosely coupled distributed systems.
- Apache ZooKeeper
  - open-source software used to perform distributed locks.
- Redis is an open source,
  - BSD licensed, advanced key-value cache and store.[9] Redis can be used to implement the Redlock Algorithm for distributed lock management.

## Further Reading

---

- [Distributed Locks are Dead; Long Live Distributed Locks!](#)
- [Distributed Lock using Zookeeper](#)
- [Distributed locks with Redis](#)



**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



Apache ZooKeeper™



**PES**  
UNIVERSITY

# CLOUD COMPUTING

## ZooKeeper

---

**Dr. Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

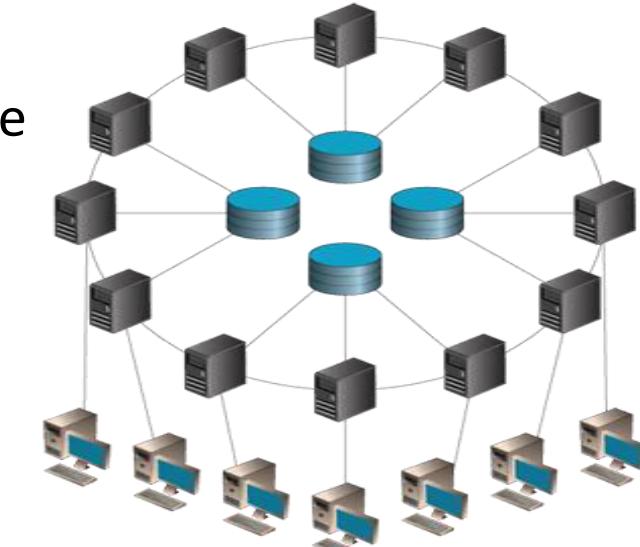
### Acknowledgements:

Significant information in the slide deck presented through the Unit 4 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

## ZooKeeper : Why is ZooKeeper needed?

In large cluster environments, like say If you had a Hadoop cluster spanning 500 or more commodity servers, you would need centralized management of the entire cluster in terms of name, group and synchronization services, configuration management, and more. If there are other projects using this cluster, then there is a need for cross-cluster services too. Embedding ZooKeeper means you don't have to build synchronization services from scratch. Interaction with ZooKeeper occurs by way of Java™ or C interface time.

- It addresses the need for a simple distributed co-ordination process, where message-based communication can be hard to use
- It supports use of locks correctly where its hard to use
- It provides reliability and availability by addressing single point of failures using an ensemble of nodes
- It ensures atomicity of the distributed transactions and ensures the applications run consistently by ensuring that the transactions either succeed or failed completely but without transaction is partial
- It helps to maintain a standard hierarchical namespace like files and directories



## Apache ZooKeeper: Objectives

---

- Simple, Robust, Good Performance
- High Availability, High throughput, Low Latency
- Tuned for Read dominant workloads
- Familiar models and interfaces
- Wait-Free: A slow/failed client will not interfere with the requests of a fast client

## Apache ZooKeeper – What is Apache ZooKeeper

- An open source, high-performance coordination service for distributed applications having many hosts.
- It automates this process of management of the hosts and allows developers to focus on building software features rather than worry about its distributed nature.
- It provides a **Centralized coordination service** that can be used by distributed applications to maintain configuration information, perform distributed locks & synchronization and enable group services.
- It uses a **shared hierarchical name space of data registers (called znode's)** to coordinate distributed processes.
- **Exposes common services in simple interface:**

- **Naming**
- **configuration management**
- **locks & synchronization**
- **group services**

*.. developers don't have to write them from scratch ..Build your own on it for specific needs.*

- For reliability, three copies of the ZooKeeper can be run so that it does not become a single point of failure.
- Apache Kafka uses ZooKeeper to manage configuration, Apache HBase uses ZooKeeper to track the status of distributed data.



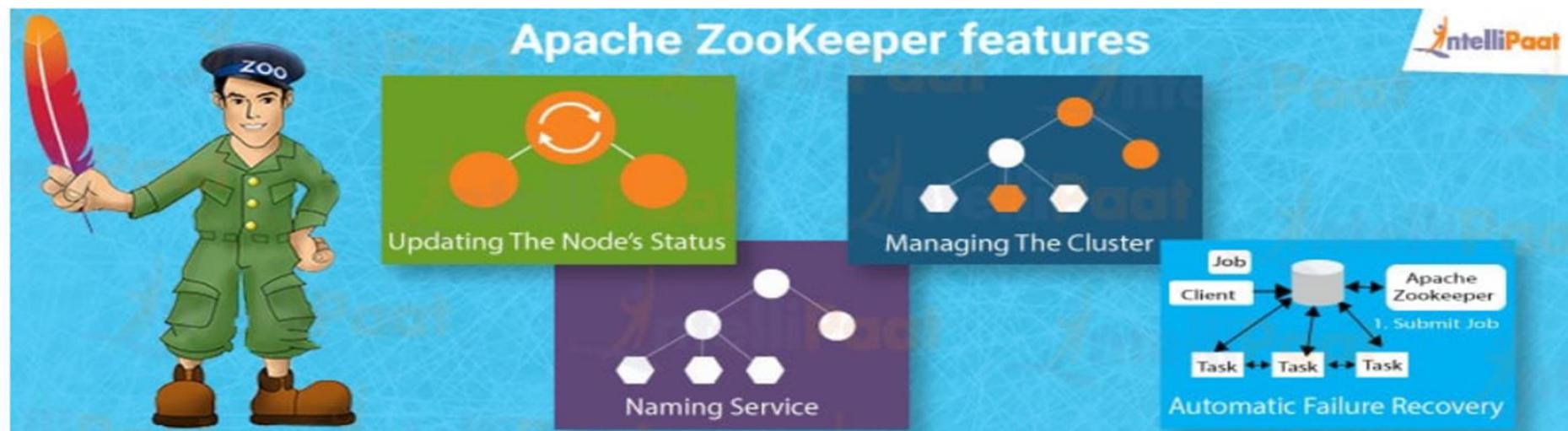
## Apache ZooKeeper Functioning

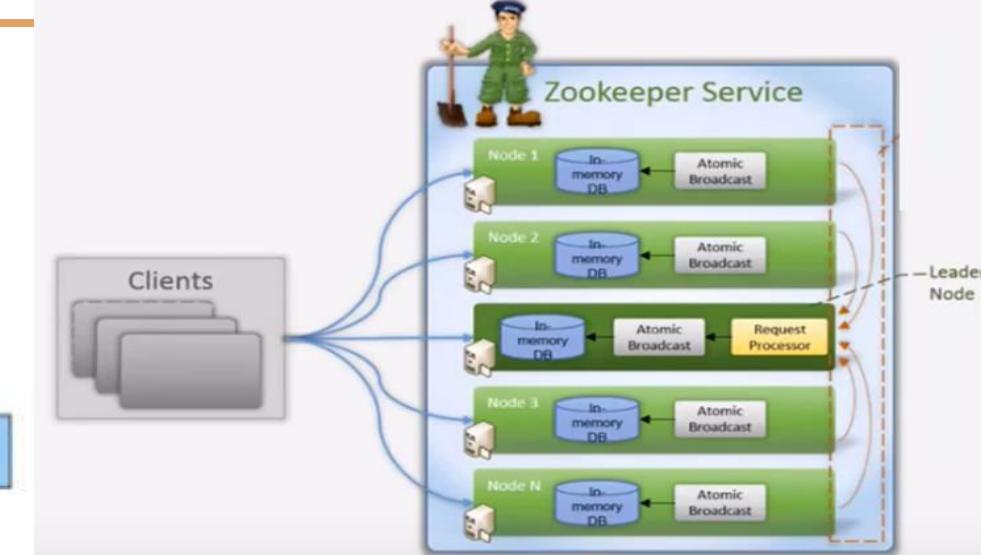
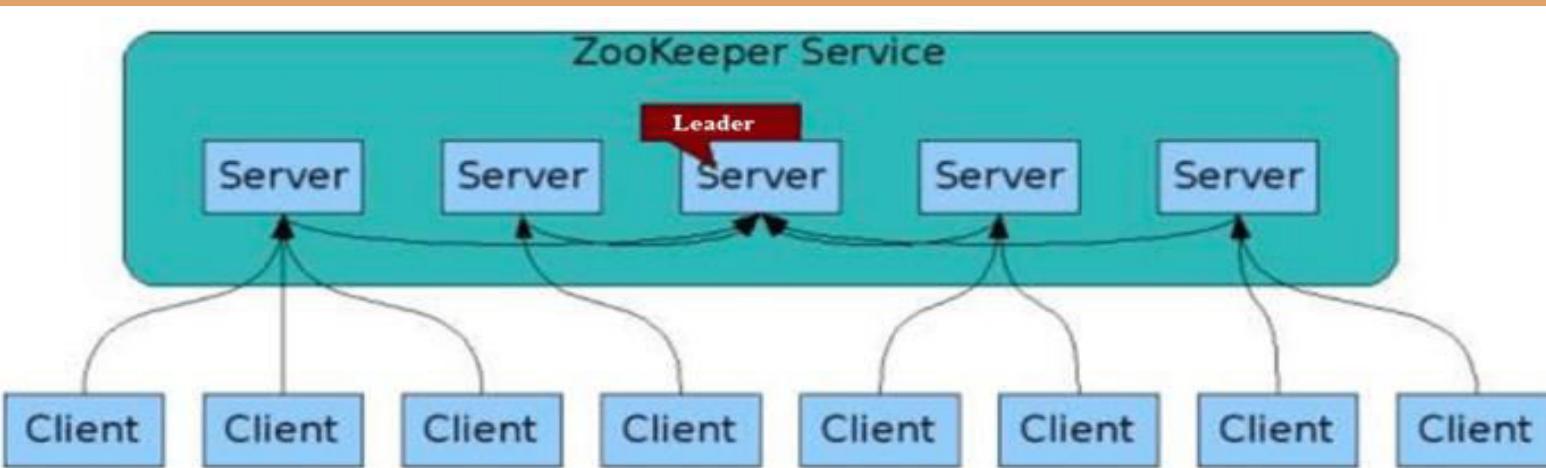
---

- For applications, ZooKeeper provides an infrastructure for cross-node synchronization by maintaining status type information in memory on ZooKeeper servers.
- An application can create what is called a znode, which is a file that persists in memory on the ZooKeeper servers. The znode can be updated by any node in the cluster, and any node in the cluster can register to be notified of changes to that znode.
- An application can synchronize its tasks across the distributed cluster by updating their status in a ZooKeeper znode.
- The znode then informs the rest of the cluster of a specific node's status change.
- A ZooKeeper server keeps a copy of the state of the entire system and persists this information in local log files. Large Hadoop clusters are supported by multiple ZooKeeper servers, with a master server synchronizing the top-level servers.
- This cluster-wide status centralization service is critical for management and serialization tasks across a large distributed set of servers.

## Apache ZooKeeper Features

- 1. Updating the Node's Status:** updates every node that allows it to store updated information about each node across the cluster.
- 2. Managing the Cluster:** manage the cluster in such a way that the status of each node is maintained in real time, leaving lesser chances for errors and ambiguity.
- 3. Naming Service:** attaches a unique identification to every node which is quite similar to the DNA that helps identify it.
- 4. Automatic Failure Recovery:** locks the data while modifying which helps the cluster recover it automatically if a failure occurs in the database.





**Server :** The server sends an acknowledgement when any client connects. A leader is elected at startup. All servers store a copy of the data tree (discussed later) in memory

**Client:** Client is one of the nodes in the distributed application cluster. It helps you to access information from the server. Every client sends a message to the server at regular intervals that helps the server to know that the client is alive.

In the case when there is no response from the connected server, the client automatically redirects the message to another server.

## Apache ZooKeeper: Architecture

**Leader:** One of the servers is designated a Leader. It gives all the information to the clients as well as an acknowledgment that the server is alive. It would perform automatic recovery if any of the connected nodes failed.

**Follower:** Server node which follows leader's instruction is called a follower.

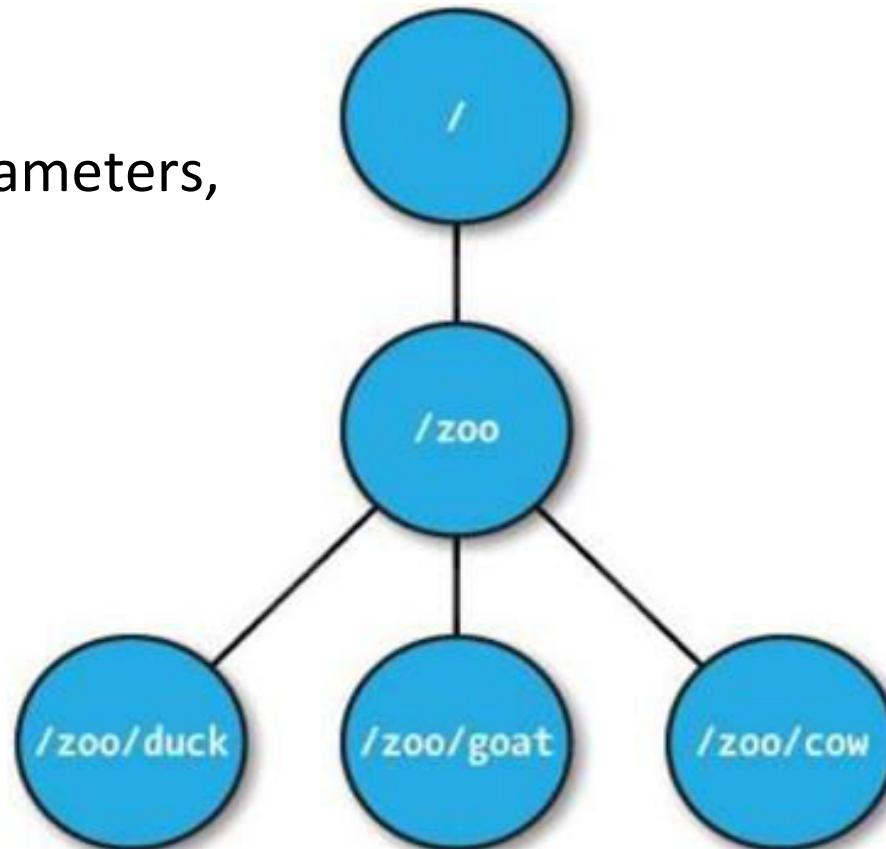
- Client read requests are handled by the correspondingly connected ZooKeeper server
- The client writes requests are handled by the ZooKeeper leader.

**Ensemble/Cluster:** Group of ZooKeeper servers which is called ensemble or a Cluster. You can use ZooKeeper infrastructure in the cluster mode to have the system at the optimal value when you are running the Apache.

**ZooKeeper WebUI:** If you want to work with ZooKeeper resource management, then you need to use WebUI. It allows working with ZooKeeper using the web user interface, instead of using the command line. It offers fast and effective communication with the ZooKeeper application.

- The data model follows a **Hierarchical namespace** (like a file system) as discussed earlier
- A **node** is a system where the cluster runs and each node is created with a “**znode**”
- Each **znode** has data (Configurations, Status, Counters, Parameters, Location information) and may or may-not have children
  - Keeps metadata information
  - Does not store big datasets
- ZNode paths:
  - Are Canonical, slash-separated and absolute
  - Do not use any relative references

Zookeeper Data Model



The entire ZooKeeper tree is kept in main memory

## Apache ZooKeeper: Types of Znodes

### 1. Persistent znodes

- Permanent and have to be deleted explicitly by the client.
- Will persist even after the session that created the znode is terminated.

### 2. Ephemeral znodes

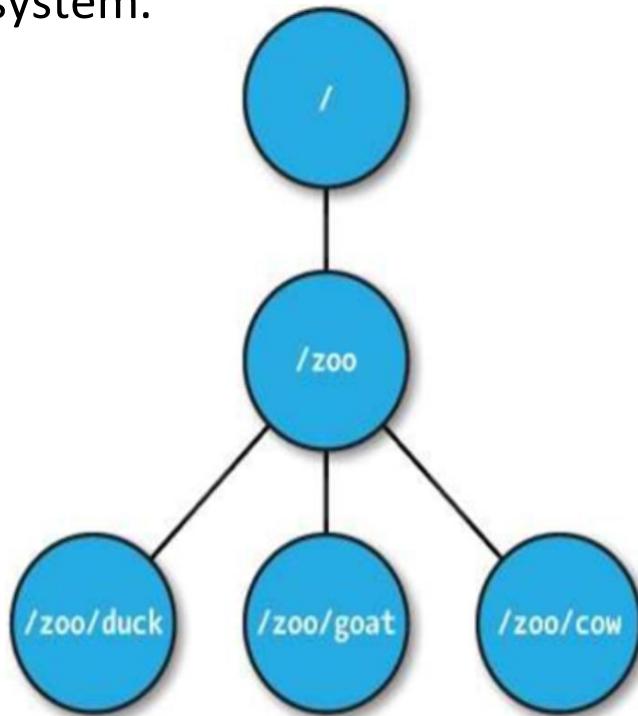
- Temporary and Exists as long as the session that created the znode is active.
- Deleted automatically when the client session creating them ends.
- Used to detect the termination of a client.
- Alerts, known as a watch, can be set up to detect the deletion of the znode.
- These nodes can't have children

### Sequence Nodes (Unique Naming)

- Append a monotonically increasing counter to the end of path
- Applies to both persistent & ephemeral nodes

***Note that when a znode is created, its type is specified and it cannot be changed later.***

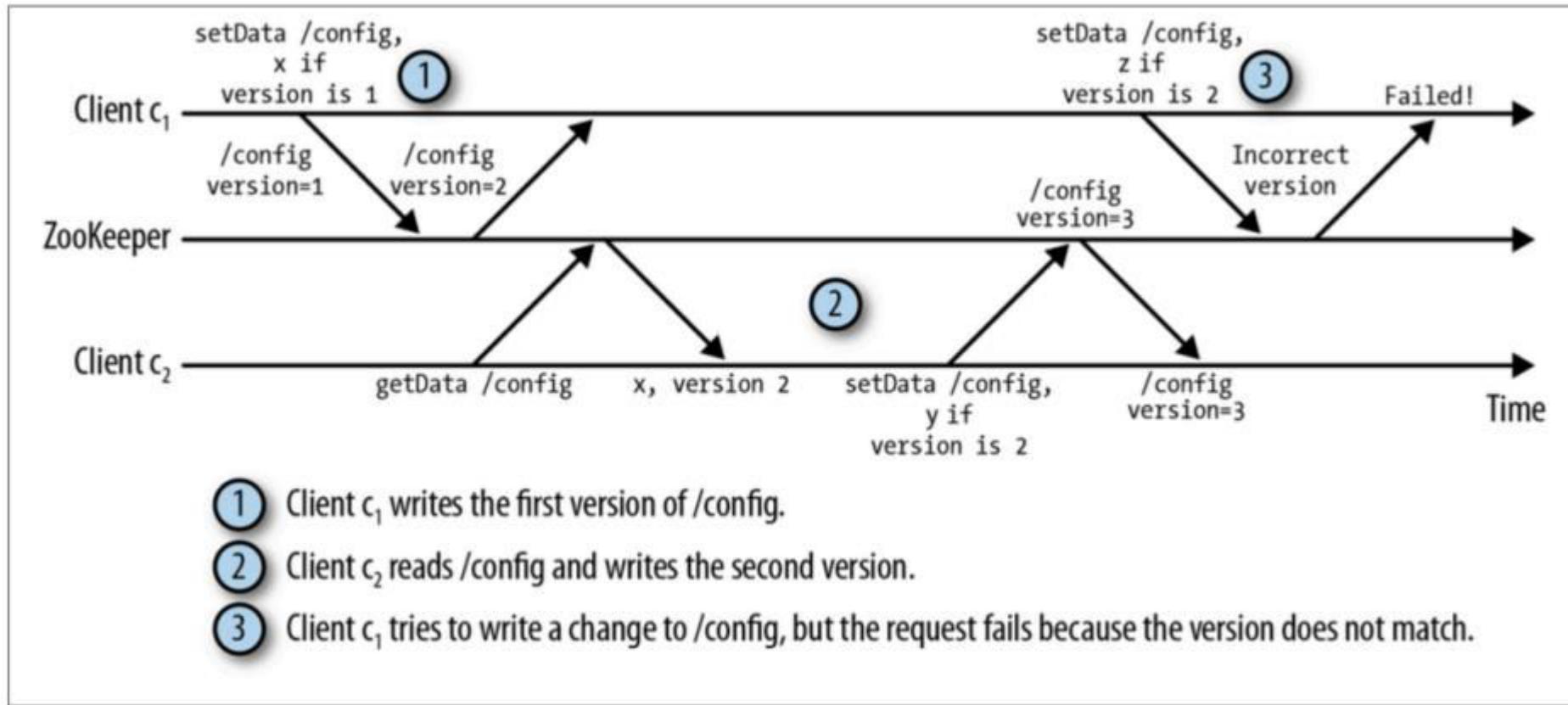
- ZooKeeper provides a name space very similar to a standard shared file system but are more like a distributed, consistent shared memory that is hierarchically organized like a file system.
- Every znode is identified by a name, which is a sequence of path elements separated by a slash ("/"), and every znode has a parent except the root ("/").
- A znode cannot be deleted if it has any children.
- Every znode can have data associated with it and is limited to the amount of data that it can have to kilobytes.
- This is because the ZooKeeper is designed to store just coordination data, such as status information, configuration, location information, and so on.
- The ZooKeeper service maintains an in-memory image of the data tree that is replicated on all the servers on which the ZooKeeper service executes.
- Only transaction logs and snapshots are stored in a persistent store, enabling high throughput.



- Only transaction logs and snapshots are stored in a persistent store, enabling high throughput. As changes are made to the znodes (during application execution), they are appended to the transaction logs.
- Each client connects only to a single ZooKeeper server and maintains a TCP connection, through which it sends requests, gets responses, gets watch events, and sends heart beats. If the TCP connection to the server breaks, the client will connect to an alternate server.

### Implementation

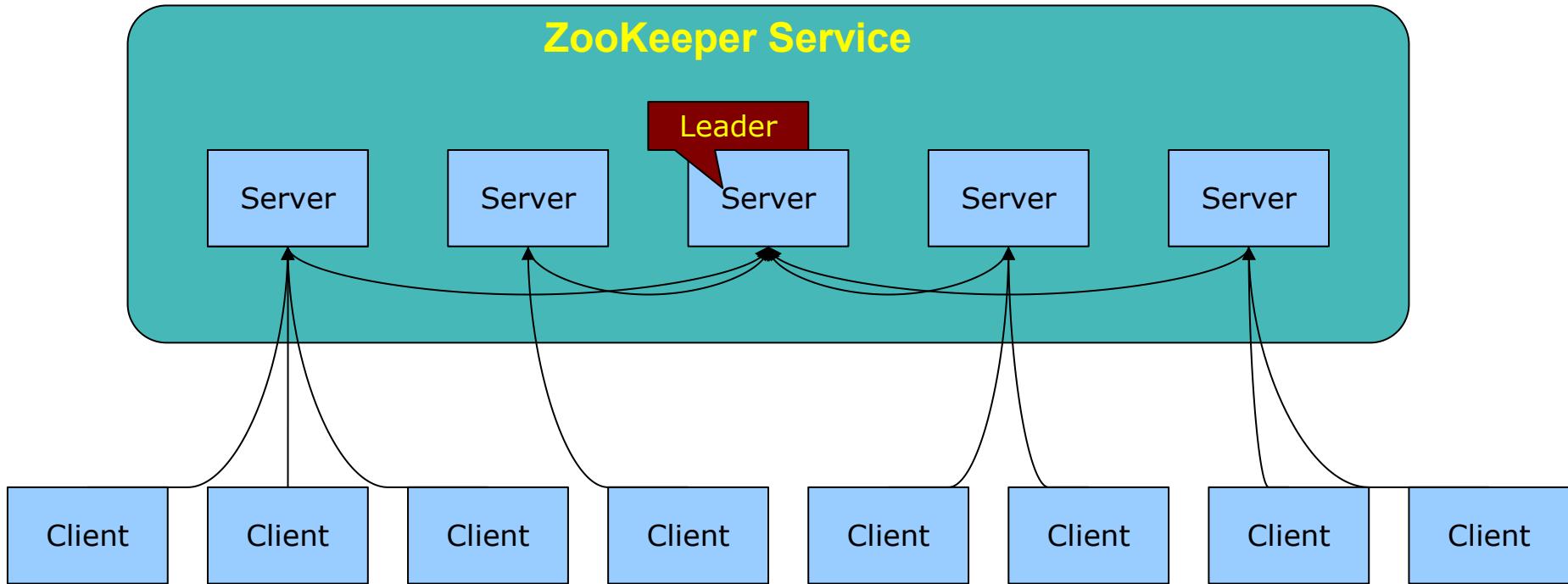
- Znode maintains a stat structure that includes version numbers for data changes, ACL changes and timestamps
- When a client performs an update or a delete, it must supply the version of the data of the znode it is changing.
- If the version it supplies doesn't match the actual version of the data, the update will fail
- All updates made by ZooKeeper are totally ordered. It stamps each update with a sequence called zxid (ZooKeeper Transaction Id). **Each update will have a unique zxid.**



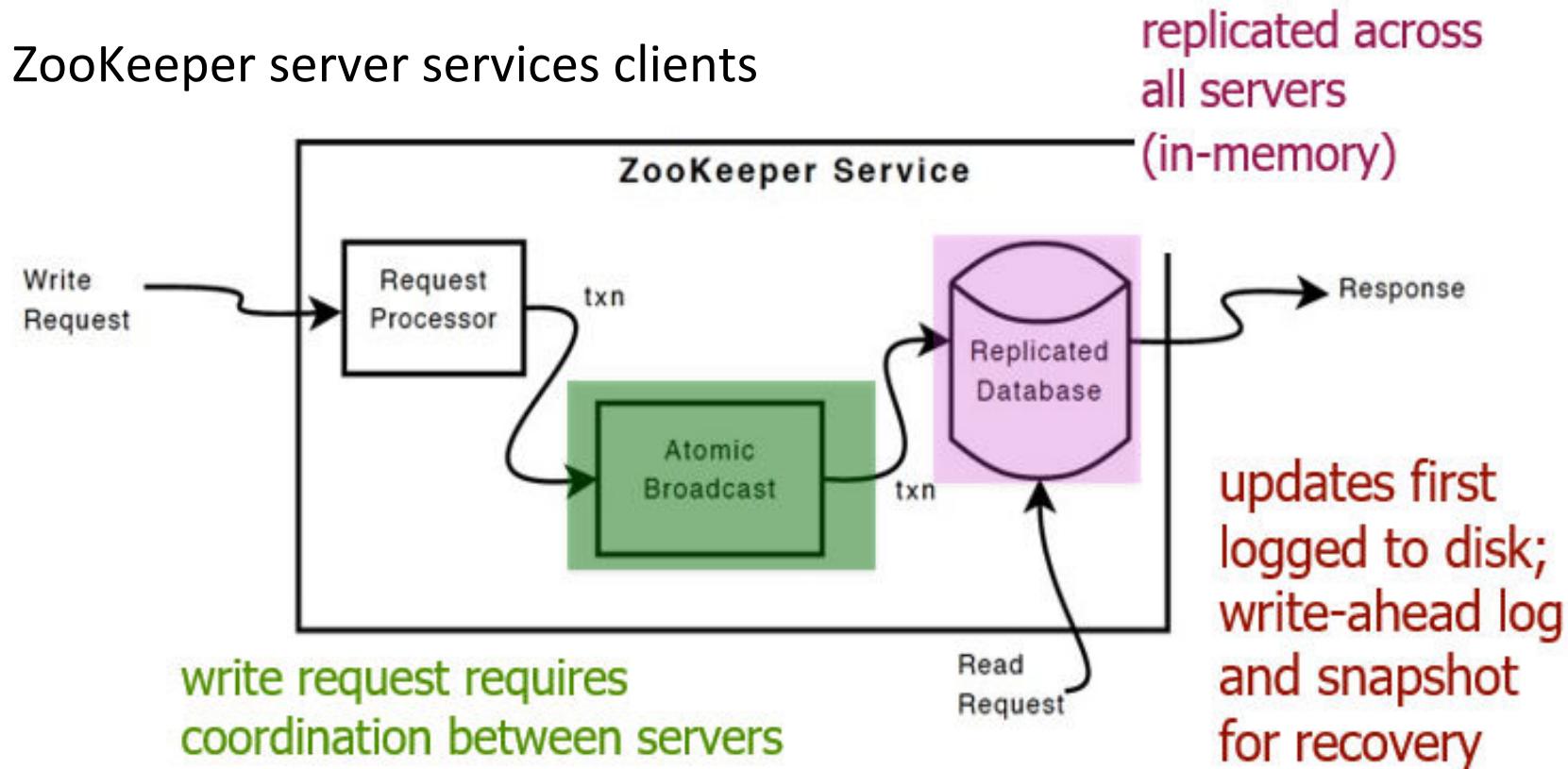
Each Znode has version number, is incremented every time its data changes

- `setData` and `delete` take version as input, operation succeeds only if client's version is equal to server's one

## ZooKeeper Reads/Writes/Watches



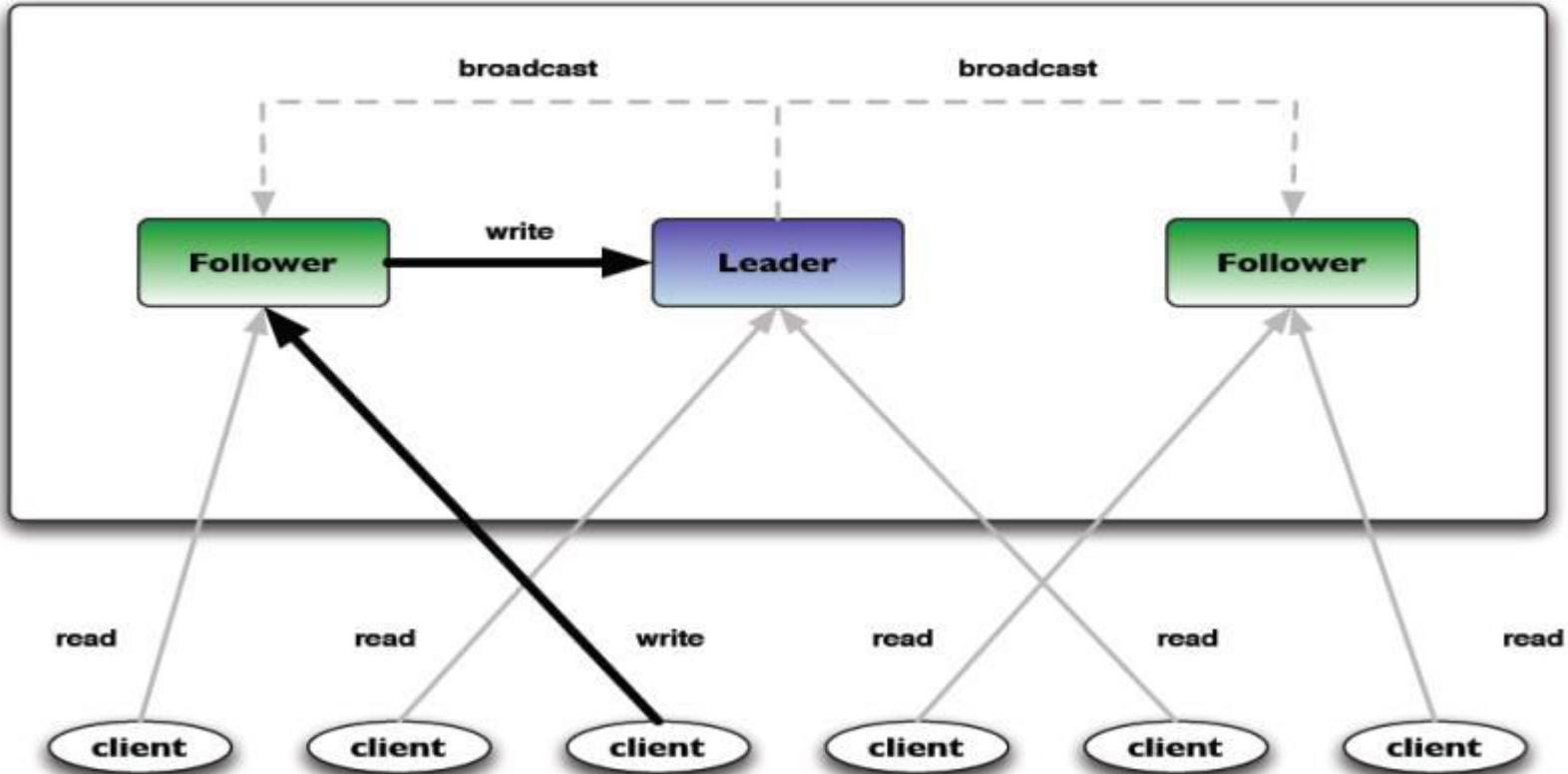
- **Reads:** between the client and single server
- **Writes:** sent between servers first, get consensus, then respond back
- **Watches:** between the client and single server
  - A watch on a znode basically means “monitoring it”



- Request Processor
- Atomic Broadcast
- Zab Protocol
- Replicated Database

Clients connect to exactly one server to submit requests

- read requests served from the local replica
- write requests are processed by an agreement protocol (an elected server leader initiates processing of the write request)



- Read requests are processed locally at the ZooKeeper server to which client is currently connected
- Write requests are forwarded to leader and go through majority consensus before a response is generated

One of the design goals of ZooKeeper is providing a very simple programming interface.

- *create* : creates a node at a location in the tree
- *delete* : deletes a node
- *exists* : tests if a node exists at a location
- *get data* : reads the data from a node
- *set data* : writes data to a node
- *get children* : retrieves a list of children of a node
- *sync* : waits for data to be propagated

Operation	Type
create	Write
delete	Write
exists	Read
getChildren	Read
getData	Read
setData	Write
getACL	Read
setACL	Write
sync	Read

## Advantages of ZooKeeper

---

- **Simple distributed coordination process**
- **Synchronization** – Mutual exclusion and co-operation between server processes. This process helps in Apache HBase for configuration management.
- **Ordered Messages**
- **Serialization** – Encode the data according to specific rules. Ensure your application runs consistently. This approach can be used in MapReduce to coordinate queue to execute running threads.
- **Reliability**
- **Atomicity** – Data transfer either succeeds or fails completely, no transaction is partial.

## Disadvantages of ZooKeeper

---

- Other solutions like consul have features like service discovery baked in, with health checks in the loop, while ZooKeeper does not.
- The ephemeral node lifecycle is tied to TCP connection; it can happen that TCP connection is up, but the process on the other side just went out of memory or otherwise not functioning properly
- It's Complex

## ZooKeeper Recipes : Why should I use ZooKeeper

- **Naming service** - A name service is a service that maps a name to some information associated with that name. E.g. if there is a need to track which servers or services are up and running and look up their status by name, Naming Service provides it with a simple Interface. Its extended to a **group membership** service by means of which you can obtain information pertaining to the group associated with the entity whose name is being looked up.
- **Distributed Locks** - To allow for serialized access to a shared resource in the distributed system, there may be a need for implementing distributed mutexes. ZooKeeper provides for an easy way for implementing them. Assuming there are N clients trying to acquire a lock, Clients creates an ephemeral, sequential znode under the path /Cluster/\_locknode. A client having the least ID in the list of children for the \_locknode will acquire the lock. Other clients watch on it and when the lock is released, and the watch event is generated, the next smallest ID will acquire the lock.
- **Data Synchronization** – Beyond the distributed mutexes above, Zookeeper addresses the need for synchronizing access to shared resources for ensuring consistency. Whether implementing a producer-consumer queue or a barrier, ZooKeeper provides for a simple interface to implement that continuously over time.

## ZooKeeper Recipes : Why should I use ZooKeeper (Cont.)

---

- **Cluster/Configuration management** – ZooKeeper centrally stores and manage the configuration of your distributed system. This means that any new nodes joining will pick up the up-to-date centralized configuration from ZooKeeper as soon as they join the system. This also allows for centralized changes to the state of the distributed system by changing the centralized configuration through one of the ZooKeeper clients.
- **Leader election** – It supports the problem of nodes going down in the distributed system, with an automatic fail-over strategy. ZooKeeper provides off-the-shelf support for doing so via leader election.
- **Message queue** - A znode will be designated to hold a queue instance. All queue items are stored as znodes under queue-znode. Producers add an item to queue by creating znode under queue-znode and Consumers retrieve items by getting and then deleting a child from queue-znode. Typically used for inter-node communication.

## Who uses ZooKeeper

---

Companies:

- Yahoo!
- Zynga
- Rackspace
- LinkedIn
- Netflix, and many more...

Projects:

- Apache Map/Reduce (Yarn)
- Apache HBase
- Apache Kafka
- Apache Storm
- Neo4j, and many more...



**THANK YOU**

---

**Prafullata Kiran Auradkar**

Department of Computer Science and Engineering

**[prafullatak@pes.edu](mailto:prafullatak@pes.edu)**



# CLOUD COMPUTING

## Reverse Proxies

---

**Prof.Jeny Jijo**

**Prof. Saritha**

Department of Computer Science & Engineering

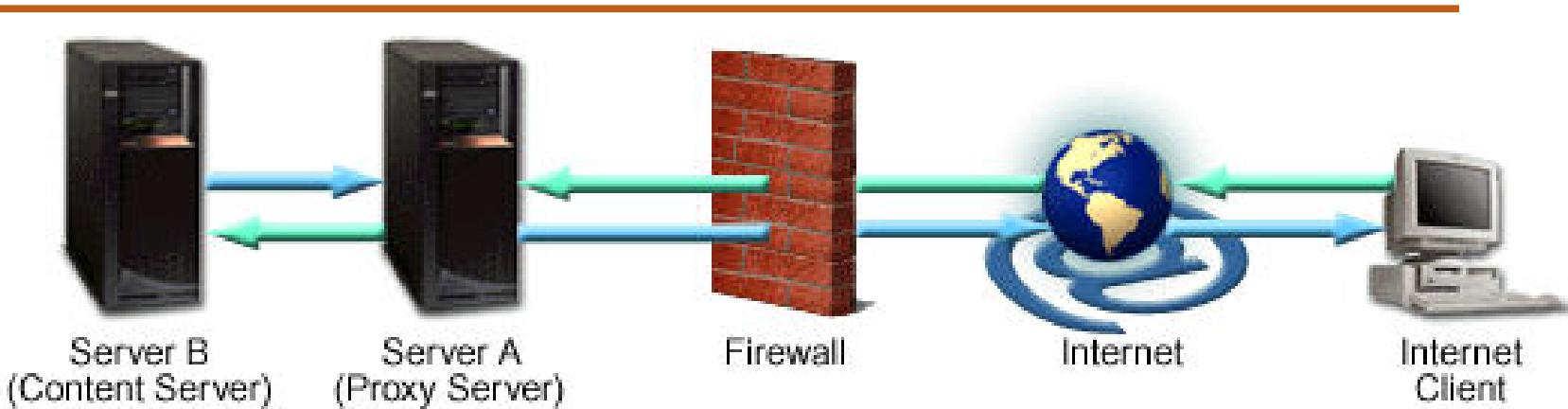
## What is a reverse proxy?

- A reverse proxy is a server that sits in front of web servers and forwards client (e.g. web browser) requests to those web servers.
- Reverse proxies help increase scalability, performance, resilience and security. The resources returned to the client appear as if they originated from the reverse proxy itself.
- Reverse proxies are typically owned or managed by the web service, and they are accessed by clients from the public internet
- Reverse proxy servers are implemented in popular open-source web servers such as Apache, Nginx and Caddy

### □ How it works

- A reverse proxy server will first check to make sure a request is valid. If a request is not valid, or not allowed (blocked by the proxy), it will not continue to process the request resulting in the client receiving an error or a redirect.
- If a request is valid, a reverse proxy may check if the requested information is cached. If it is, the reverse proxy serves the cached information.
- If it is not, the reverse proxy will request the information from the content server and serve it to the requesting client. It also caches the information for future requests.

## What is a reverse proxy?- Example



An Internet client initiates a request to Server A (Proxy Server) which, unknown to the client, is actually a reverse proxy server. The request is allowed to pass through the firewall and is valid but is not cached on Server A. The reverse proxy (Server A) requests the information from Server B (Content Server), which has the information the Internet client is requesting. The information is served to the reverse proxy, where it is cached, and relayed through the firewall to the client. Future requests for the same information will be fulfilled by the cache, lessening network traffic and load on the content server (proxy caching is optional and not necessary for proxy to function on your HTTP Server). In this example, all information originates from one content server (Server B).

### Benefits of reverse proxy

---

Security, load balancing, and ease of maintenance are the three most important benefits of using reverse proxy. Besides, they can also play a role in identity branding and optimization.

#### Improved online security

Reverse proxies play a key role in building a zero trust architecture for organizations – that secures sensitive business data and systems. They only forward requests that your organization wants to serve. Reverse proxies also make sure no information about your backend servers is visible outside your internal network, thus protecting them from being directly accessed by malicious clients to exploit any vulnerabilities. They safeguard your backend servers from distributed denial-of-service (DDoS) attacks – by rejecting or blacklisting traffic from particular client IP addresses, or limiting the number of connections accepted from

## Benefits of reverse proxy

---

### □ Increased scalability and flexibility

Increased scalability and flexibility, is generally most useful in a load balanced environment where the number of servers can be scaled up and down depending on the fluctuations in traffic volume. Because clients see only the reverse proxy's IP address, the configuration of your backend infrastructure can be changed freely. When excessive amounts of internet traffic slow down systems, the load balancing technique distributes traffic over one or multiple servers to improve the overall performance. It also ensures that applications no longer have a single point of failure. If and when one server goes down, its siblings can take over!

### □ Web acceleration

Reverse proxies can also help with 'web acceleration' - reducing the time taken to generate a response and return it to the client.

## Benefits of reverse proxy

---

### □ Identity branding

Most businesses host their website's content management system or shopping cart apps with an external service outside their own network. Instead of letting site visitors know that you're sending them to a different URL for payment, businesses can conceal that detail using a reverse proxy.

### □ Caching commonly-requested data

Businesses that serve a lot of static content like images and videos can set up a reverse proxy to cache some of that content. This kind of caching relieves pressure on the internal services, thus speeding up performance and improving user experience – especially for sites that feature dynamic content.

## What is a forward proxy

---

- ❑ A forward proxy, often called a **proxy, proxy server, or web proxy**.
- ❑ Forward proxy is a server that sits in front of a group of client machines. When those computers make requests to sites and services on the Internet, the proxy server intercepts those requests and then communicates with web servers on behalf of those clients, like a middleman.
- ❑ There are a few reasons one might want to use a forward proxy:
  1. To avoid state or institutional browsing restrictions
  2. To block access to certain content
  3. To protect their identity online

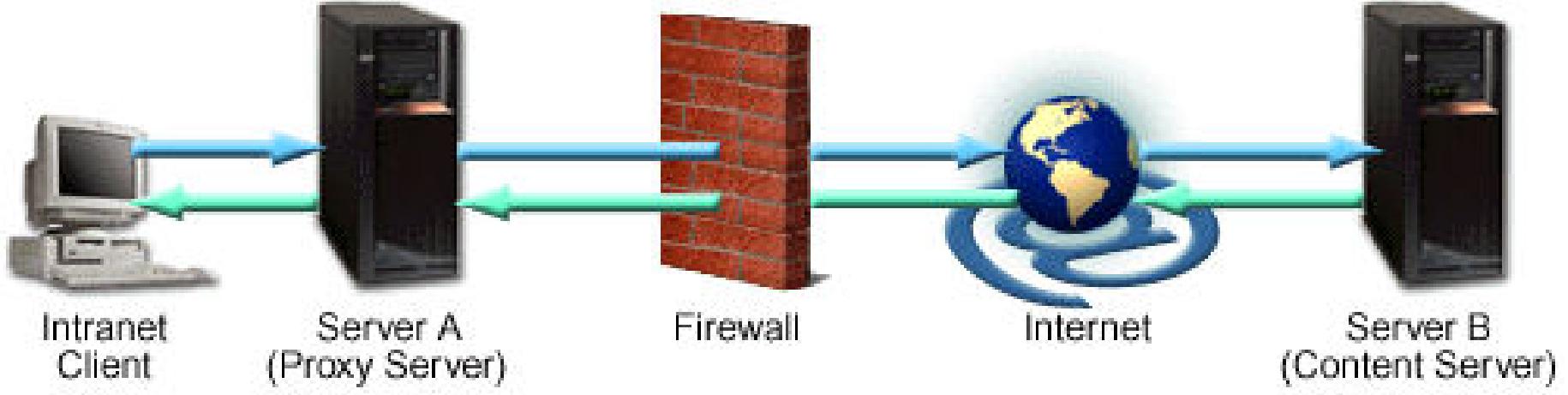
## What is a forward proxy

---

### □ How it works

- A forward proxy server will first check to make sure a request is valid. If a request is not valid, or not allowed (blocked by the proxy), it will reject the request resulting in the client receiving an error or a redirect.
- If a request is valid, a forward proxy may check if the requested information is cached. If it is, the forward proxy serves the cached information.
- If it is not, the request is sent through a firewall to an actual content server which serves the information to the forward proxy. The proxy, in turn, relays this information to the client and may also cache it, for future requests.

## What is a forward proxy?- Example



An intranet client initiates a request that is valid but is not cached on Server A (Proxy Server). The request is sent through the firewall to the Internet server, Server B (Content Server), which has the information the client is requesting. The information is sent back through the firewall where it is cached on Server A and served to the client. Future requests for the same information will be fulfilled by the cache, lessening network traffic (proxy caching is optional and not necessary for forward proxy to function on your HTTP Server).

### Benefits of forward proxy

---

There are a few reasons one might want to use a forward proxy:

- **To avoid state or institutional browsing restrictions** - Some governments, schools, and other organizations use firewalls to give their users access to a limited version of the Internet. A forward proxy can be used to get around these restrictions, as they let the user connect to the proxy rather than directly to the sites they are visiting.
- **To block access to certain content** - Conversely, proxies can also be set up to block a group of users from accessing certain sites. For example, a school network might be configured to connect to the web through a proxy which enables content filtering rules, refusing to forward responses from Facebook and other social media sites.

### Benefits of forward proxy

---

**To protect their identity online** - In some cases, regular Internet users simply desire increased anonymity online, but in other cases, Internet users live in places where the government can impose serious consequences to political dissidents. Criticizing the government in a web forum or on social media can lead to fines or imprisonment for these users. If one of these dissidents uses a forward proxy to connect to a website where they post politically sensitive comments, the IP address used to post the comments will be harder to trace back to the dissident. Only the IP address of the proxy server will be visible.

## Reverse Proxy vs Forward Proxy

S.No.	FORWARD PROXY	REVERSE PROXY
1	Forward proxy connection initiates from inside secured zone and destined to outside unsecured global network.	Reverse proxy connection comes from outside global network and destined to inside secured network.
2	Forward proxy are not used for Application Delivery.	Reverse proxy are built for Application Delivery.
3	Forward proxy are good for content filtering, natting, Email Security etc.	Reverse Proxy are used for Load Balancing (TCP Multiplexing), Content Switching, Authentication and application firewall.
4	Forward proxy restrict the internal user from accessing the user filtered/restricted site.	Reverse proxy restrict the outside user/client to have direct access to internal/private networks.

## Reverse Proxy vs Forward Proxy

---

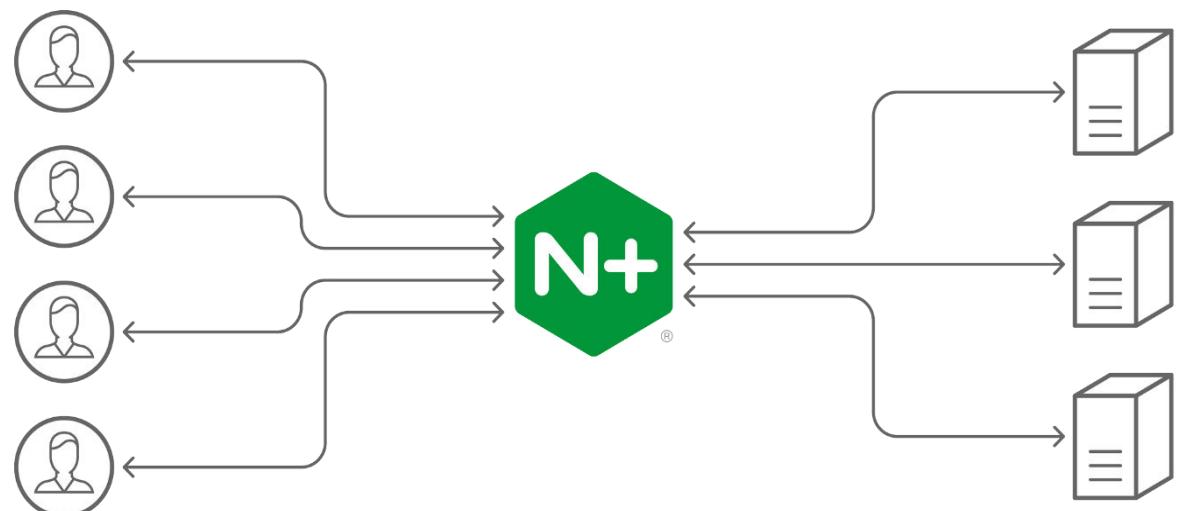
**Forward Proxies are good for:**

- ✓ Content Filtering
- ✓ eMail security
- ✓ NAT'ing
- ✓ Compliance Reporting

**Reverse Proxies are good for:**

- ✓ Application Delivery including:
- ✓ Load Balancing (TCP Multiplexing)
- ✓ SSL Offload/Acceleration (SSL Multiplexing)
- ✓ Caching
- ✓ Compression
- ✓ Content Switching/Redirection
- ✓ Application Firewall
- ✓ Authentication
- ✓ Single Sign On

- Nginx is open-source web server that provides capabilities like reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP/2, TCP, and UDP protocols.



# Nginx as High Requests Handling Architecture

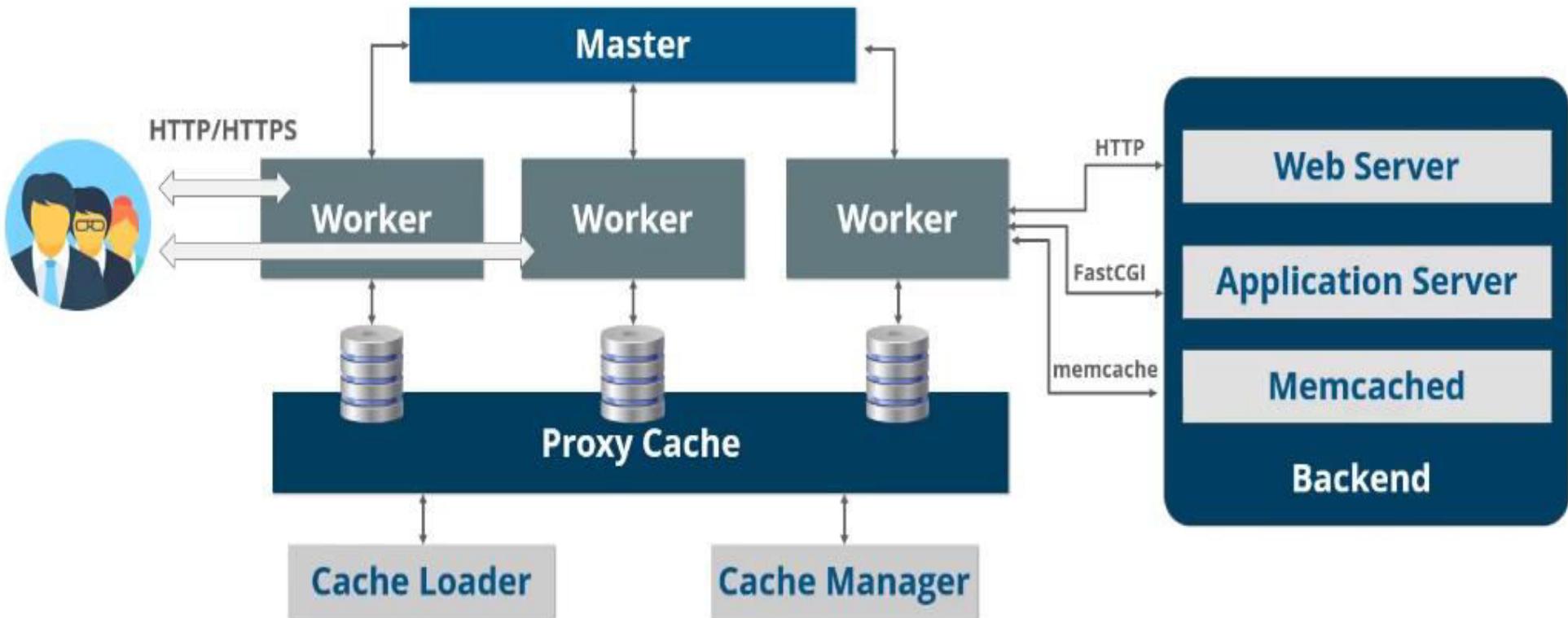
---

- Nginx utilizes an event-driven architecture and deals with the requests asynchronously. It was designed to use a non-blocking event-driven connection handling algorithm. Hence, its process can handle thousands of connections (requests) within 1 processing thread. Such connections process modules allow Nginx to work very fast and wide with limited resources.
- Also, Nginx can handle more than 10,000 simultaneous connections with low (CPU & Memory) resources under heavy request loads.

# CLOUD COMPUTING

## Nginx Architecture

NGINX follows the master-slave architecture by supporting event-driven, asynchronous and non-blocking model..



### MASTER:

The master allocate the jobs for the workers as per the request from the client. Once the job allocated to the workers, the master will look for the next request from the client that's it won't wait for the response from the workers. Once the response comes from workers master will send the response to the client.

### WORKERS:

Workers are the slaves in the NGINX architecture, will heed to the Master. Each worker can handle more than 1000 request at a time in a single-threaded manner. Once the process gets done, the response will be sent to the master. The single-threaded will saves the RAM and ROM size by working on the same memory space instead of different memory spaces. The multi-threaded will work on different memory spaces.

### Cache:

Nginx cache is used to render the page very fast by getting from cache memory instead of getting from the server. The pages are getting stored in cache memory on the first request for the page.

## Benefits of Nginix

---

- **Load balancing:** Nginx load balance the client request to multiple upstream servers evenly which improve performance and provide redundancy in case of server failure. This helps to keep the application up all the time to serve client requests and provide better SLA for application.
- **Security:** Nginx server provides security to backend servers that exist in the private network by hiding their identity. The backend servers are unknown to the client that are making requests. it also provides a single point of access to multiple backend servers regardless of the backend network topology.
- **Caching:** Nginx can serve static content like image, videos, etc directly and deliver better performance. It reduces the load on the backend server by serving static content directly instead of forwarding it to the backend server for processing.

## Benefits of Nginix

---

- **Logging:** Nginx provides centralized logging for backed server request and response passing through it and provides a single place to audit and log for troubleshooting issues.
- **TLS/SSL support:** Nginx allows secure communication between client and server using TLS/SSL connection. User data remains secure & encrypted while transferring over the wire using an HTTPS connection.
- **Protocol Support:** Nginx supports HTTP, HTTPS, HTTP/1.1, HTTP/2, gRPC - Hypertext Transport Protocol along with both IP4 & IP6 internet protocol.



**THANK YOU**

---

**Jeny Jijo  
Saritha**

Department of Computer Science & Engineering

**[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)  
[saritha.k@pes.edu](mailto:saritha.k@pes.edu)**



## CLOUD COMPUTING

### Scaling Computation- Hybrid Cloud and Cloud Bursting

---

**Prof.Jeny Jijo  
Prof. Saritha**

Department of Computer Science & Engineering

## Scalability

---

- ❑ **Cloud scalability** in cloud computing refers to the ability to increase or decrease IT resources as needed to meet changing demand.
- ❑ Data storage capacity, processing power and networking can all be scaled using existing cloud computing architecture.
- ❑ scaling can be done quickly and easily, typically with little to no disruption or down time.
- ❑ Third-party cloud providers have all the infrastructure already in place; in the past, when scaling with on-premises physical infrastructure, the process could take weeks or months and require tremendous expense.

## Cloud scalability versus cloud elasticity

- ❑ **Elasticity** refers to a system's ability to grow or shrink dynamically in response to changing workload demands, like a sudden spike in web traffic.
- ❑ An elastic system automatically adapts to match resources with demand as closely as possible, in real time. A system's scalability, refers to its ability to increase workload with existing hardware resources.
- ❑ A **scalable** solution enables stable, longer-term growth in a pre-planned manner, while an elastic solution addresses more immediate, variable shifts in demand.
- ❑ Elasticity and scalability in cloud computing are both important features for a system, but the priority of one over the other depends in part on whether your business has predictable or highly variable workloads.

## Benefits of cloud scalability

---

- **Cost savings:** Thanks to cloud scalability, businesses can avoid the upfront costs of purchasing expensive equipment that could become outdated in a few years. Through cloud providers, they pay for only what they use and minimize waste.
- **Disaster recovery:** With scalable cloud computing, you can reduce disaster recovery costs by eliminating the need for building and maintaining secondary data centers.
- **Convenience:** Often with just a few clicks, IT administrators can easily add more VMs that are available without delay—and customized to the exact needs of an organization. That saves precious time for IT staff. Instead of spending hours and days setting up physical hardware, teams can focus on other tasks.

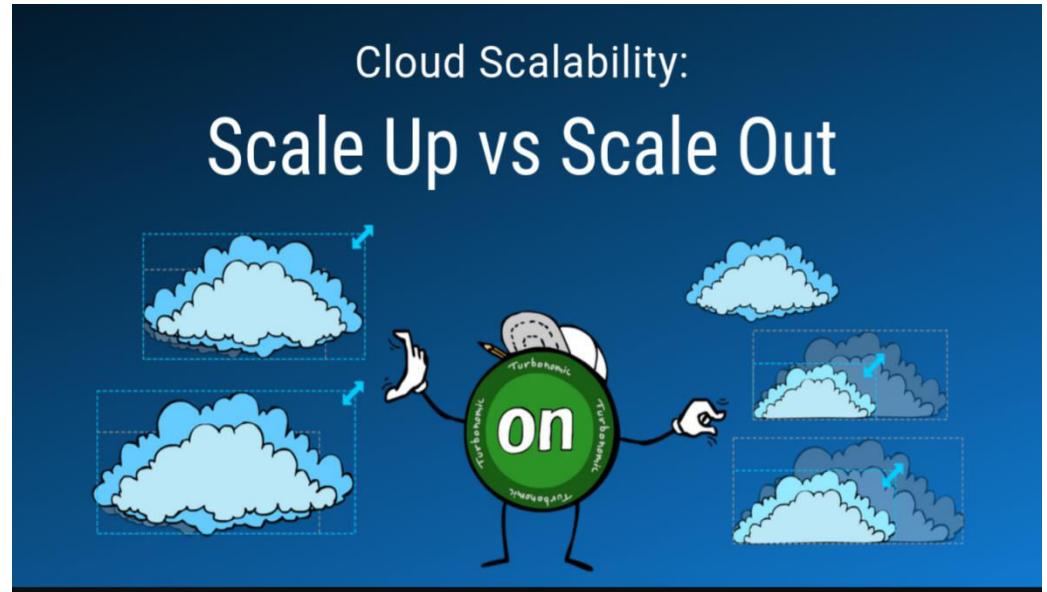
## Benefits of cloud scalability

---

**Flexibility and speed:** As business needs change and grow—including unexpected spikes in demand—cloud scalability allows IT to respond quickly. Today, even smaller businesses have access to high-powered resources that used to be cost prohibitive. No longer are companies tied down by obsolete equipment—they can update systems and increase power and storage with ease.

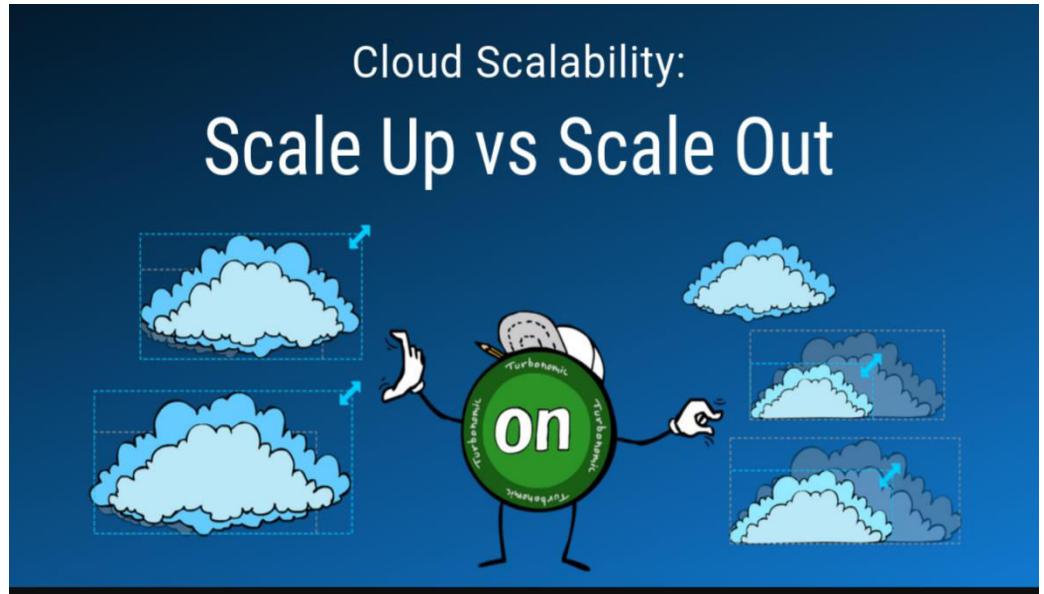
## Cloud Scaling Strategies

- ❑ **Cloud Vertical Scaling** refers to adding more CPU, memory, or I/O resources to an existing server, or replacing one server with a more powerful server.
- ❑ Amazon Web Services (AWS) vertical scaling and Microsoft Azure vertical scaling can be accomplished by changing instance sizes, or in a data center by purchasing a new, more powerful appliance and discarding the old one.
- ❑ AWS and Azure cloud services have many different instance sizes, so scaling vertically is possible for everything from EC2 instances to RDS databases.

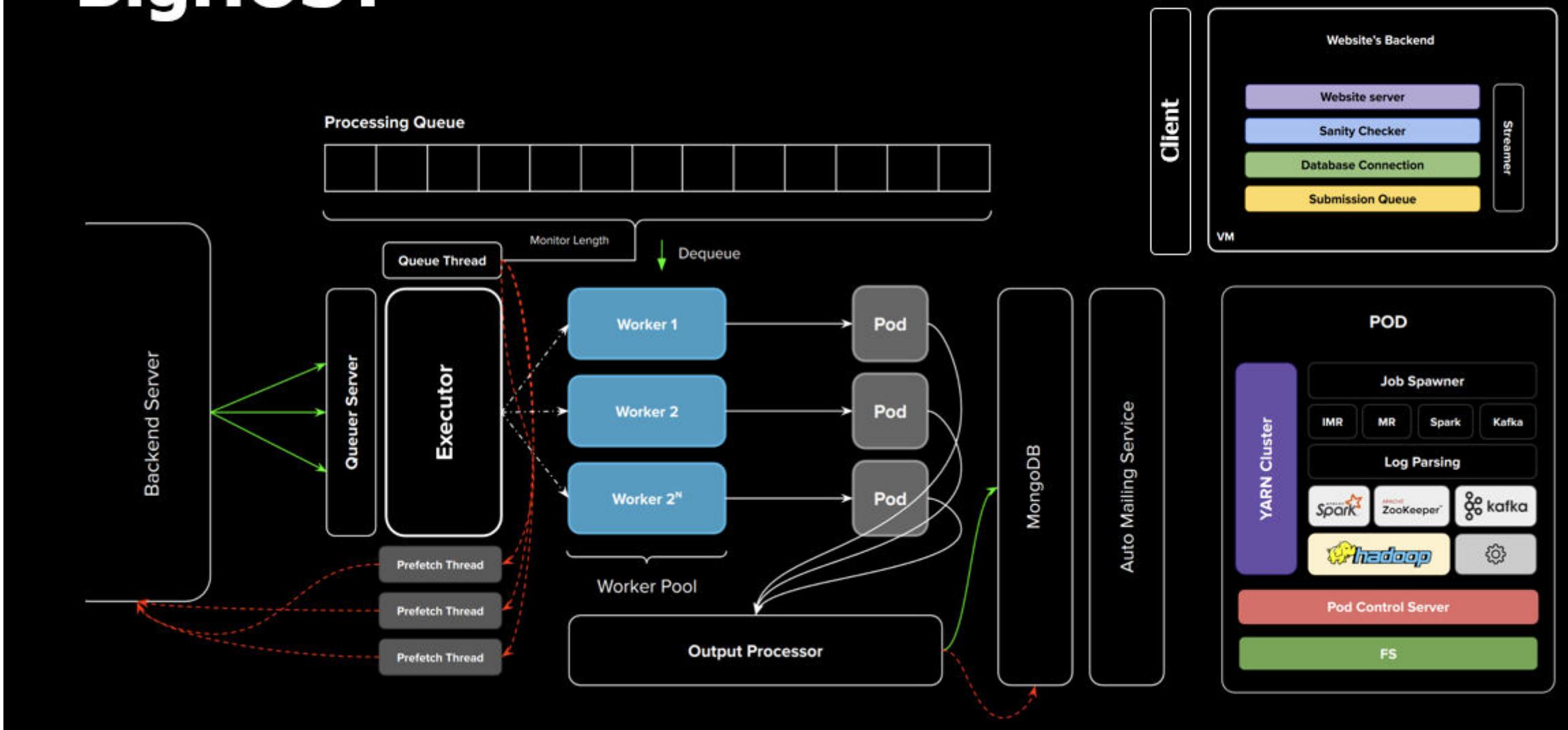


## Cloud Scaling Strategies

- ❑ **Cloud Horizontal Scaling** refers to provisioning additional servers to meet your needs, often splitting workloads between servers to limit the number of requests any individual server is getting.
- ❑ In a cloud-based environment, this would mean adding additional instances instead of moving to a larger instance size.

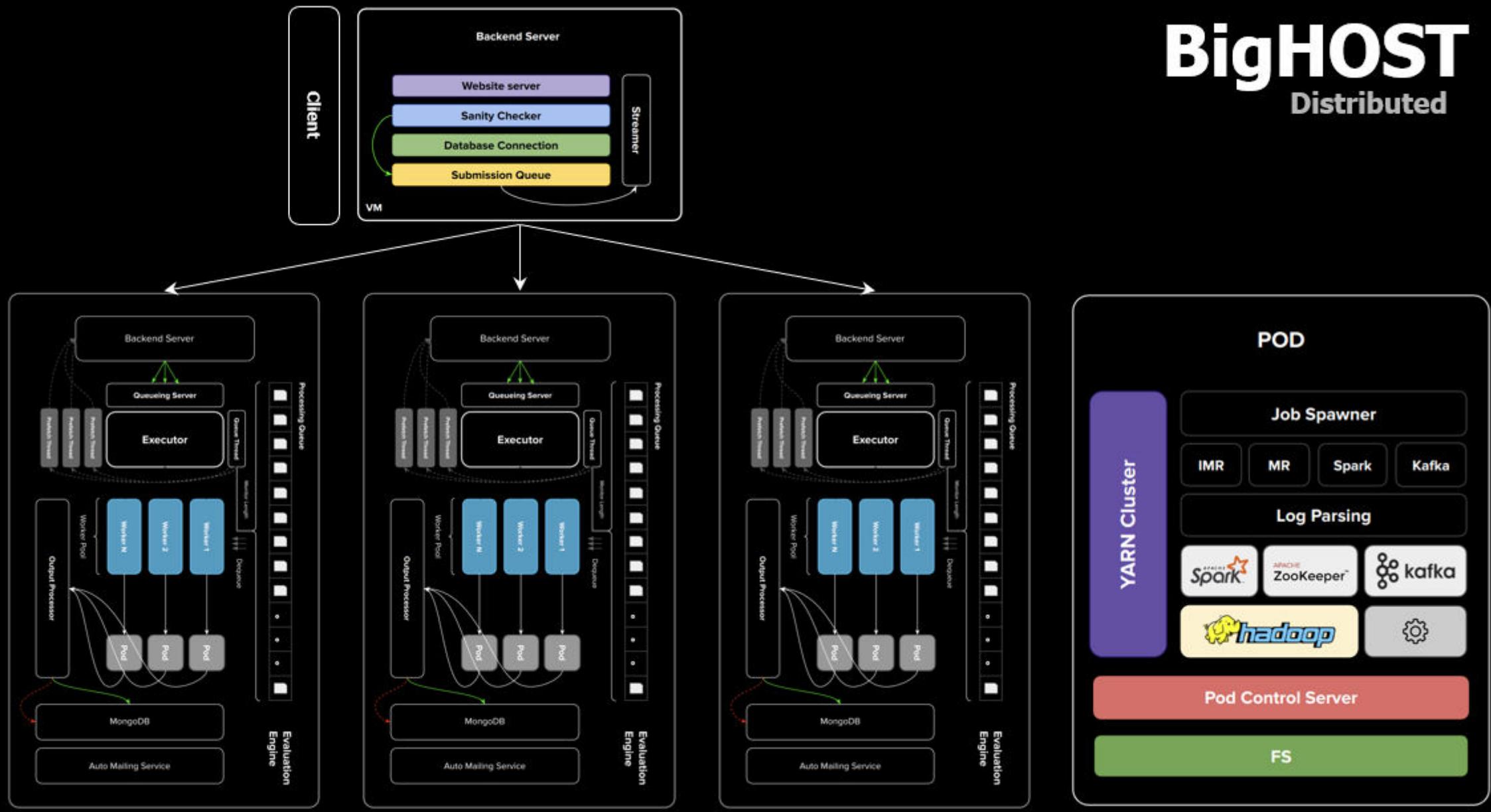


# BigHOST



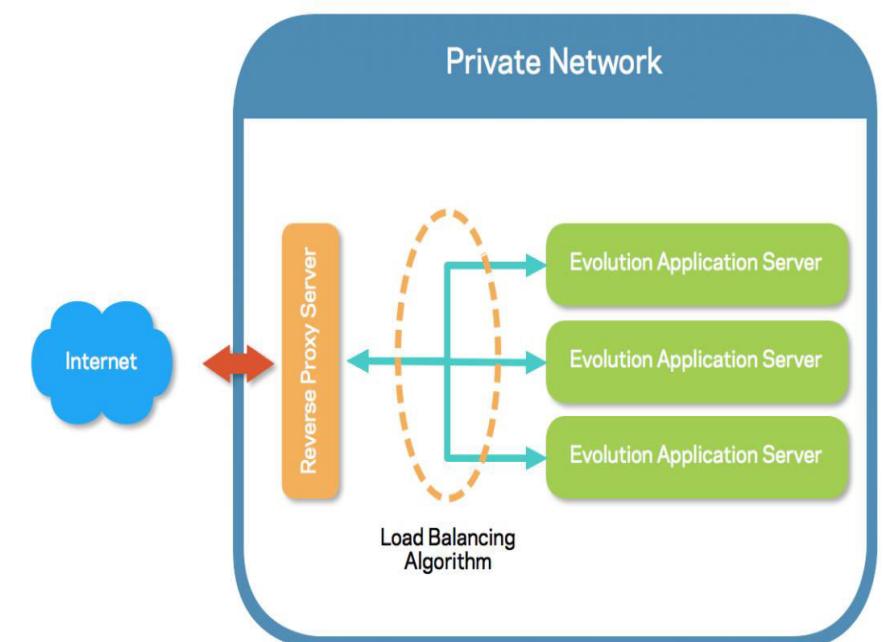
# BigHOST

Distributed



## Scalability through reverse proxy

- ❑ A reverse proxy server can handle many different roles depending on the size and complexity of your network.
- ❑ In general, a reverse proxy is a server that retrieves resources from other servers. In a cloud-based environment, this would mean adding additional instances instead of moving to a larger instance size.
- ❑ The reverse proxy might even be referred to as a load balancer.
- ❑ To the outside world there is just a single server to connect to, but this load balancer or proxy server takes each request and forwards it on to an application server on our private network.
- ❑ Load balancer decides which server should receive the request.



### Hybrid Cloud and Cloud Bursting

- ❑ Hybrid cloud is a combination of private and public clouds enabling expansion of local infrastructure to commercial infrastructure on a need basis
- ❑ Using a Hybrid cloud, an organization can leverage existing infrastructure available in-house and seamlessly include or supplement additional resources from public cloud based on demand.
- ❑ Hybrid cloud is most cost effective for the cloud user as it ensures good utilization of existing infrastructure.
- ❑ **The primary benefit of a hybrid cloud is agility.** The need to adapt and change direction quickly is a core principle of a digital business.

## Hybrid Cloud Benefits

---

- ✓ **Dynamic or frequently changing workloads.** Use an easily scalable public cloud for your dynamic workloads, while leaving less volatile, or more sensitive, workloads to a private cloud or on-premises data center.
- ✓ **Separating critical workloads from less-sensitive workloads.** You might store sensitive financial or customer information on your private cloud, and use a public cloud to run the rest of your enterprise applications.
- ✓ **Big data processing.** It's unlikely that you process big data continuously at a near-constant volume. Instead, you could run some of your big data analytics using highly scalable public cloud resources, while also using a private cloud to ensure data security and keep sensitive big data behind your firewall.
- ✓ **Moving to the cloud incrementally, at your own pace.** Put some of your workloads on a public cloud or on a small-scale private cloud. See what works for your enterprise, and continue expanding your cloud presence as needed—on public clouds, private clouds, or a mixture of the two.

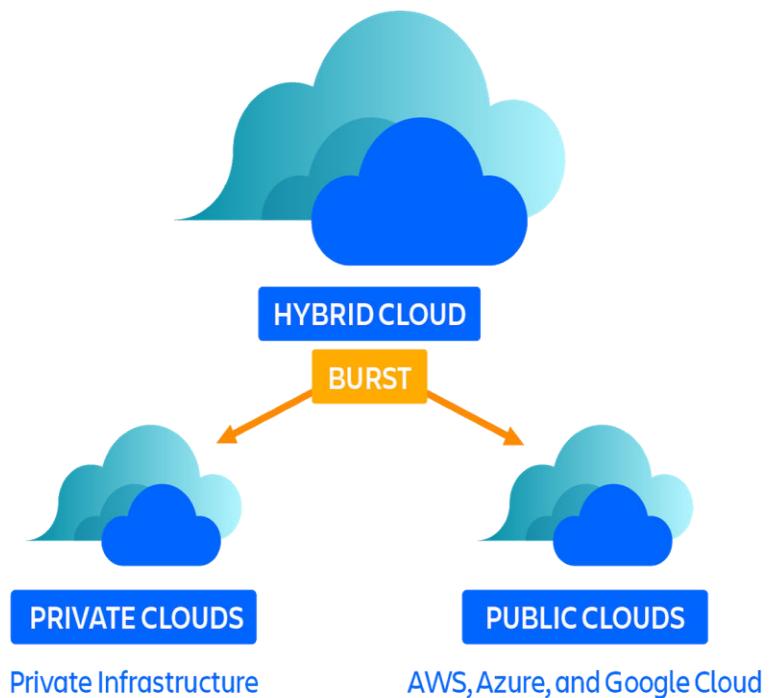
## Hybrid Cloud Scenarios

---

- ✓ **Temporary processing capacity needs.** A hybrid cloud lets you allocate public cloud resources for short-term projects, at a lower cost than if you used your own data center's IT infrastructure. That way, you don't overinvest in equipment you'll need only temporarily.
- ✓ **Flexibility for the future.** A hybrid cloud approach lets you match your actual data management requirements to the public cloud, private cloud, or on-premises resources that are best able to handle them.
- ✓ **Best of both worlds.** Unless you have clear-cut needs fulfilled by only a public cloud solution or only a private cloud solution, why limit your options? Choose a hybrid cloud approach, and you can tap the advantages of both worlds simultaneously.

## Cloud Bursting

- ❑ Cloud bursting is defined as a configuration that is set up between the public cloud and the private cloud to deal with the peaks in IT demands with an advantage of economical saving (i.e. a user need to pay for the resource if and only if there is demand for those resources) and if any organization uses a private cloud consumes 100 percent of the available resource then the overflow traffic is directed to the public cloud to avoid any interruption of the service.
- ❑ Hybrid cloud enables Cloud Bursting of the private cloud by allowing the addition of extra capacity to a private infrastructure by borrowing from a public cloud



### Need for Cloud Bursting

---

Cloud bursts are triggered by spikes in computational resource demand. These bursts could be an influx of user traffic or expensive one-off computational tasks. Some common cloud bursting situations include:

#### **Software development**

Software development and analytics are two of the most common use cases for cloud bursting. DevOps teams often use multiple virtual machines for testing purposes, which are only needed for a short amount of time. Plus, CI/CD tasks are good candidates for bursting into the public cloud, since CI/CD requires several one-off tasks that run for a short amount of time when developers push new commits.

## Need for Cloud Bursting

### Marketing campaigns

Marketing campaigns for new product launches can generate a huge influx of traffic that requires extended cloud resources. Imagine the marketing push for an anticipated Hollywood movie or video game release. These events generate a temporary stampede of traffic that subsides after the launch news passes, so it is a great time to utilize cloud bursting.

### **Big data modelling and queries**

Big data companies frequently need to execute one-time queries or generate models that will exceed the capacity of their private cloud. These tasks are well suited to cloud bursting. The company can burst into the public cloud for additional resources to expedite the task. Some granular examples of big data tasks include:

- ✓ High-fidelity 3D rendering
- ✓ AI and ML model training
- ✓ Autonomous vehicle simulation

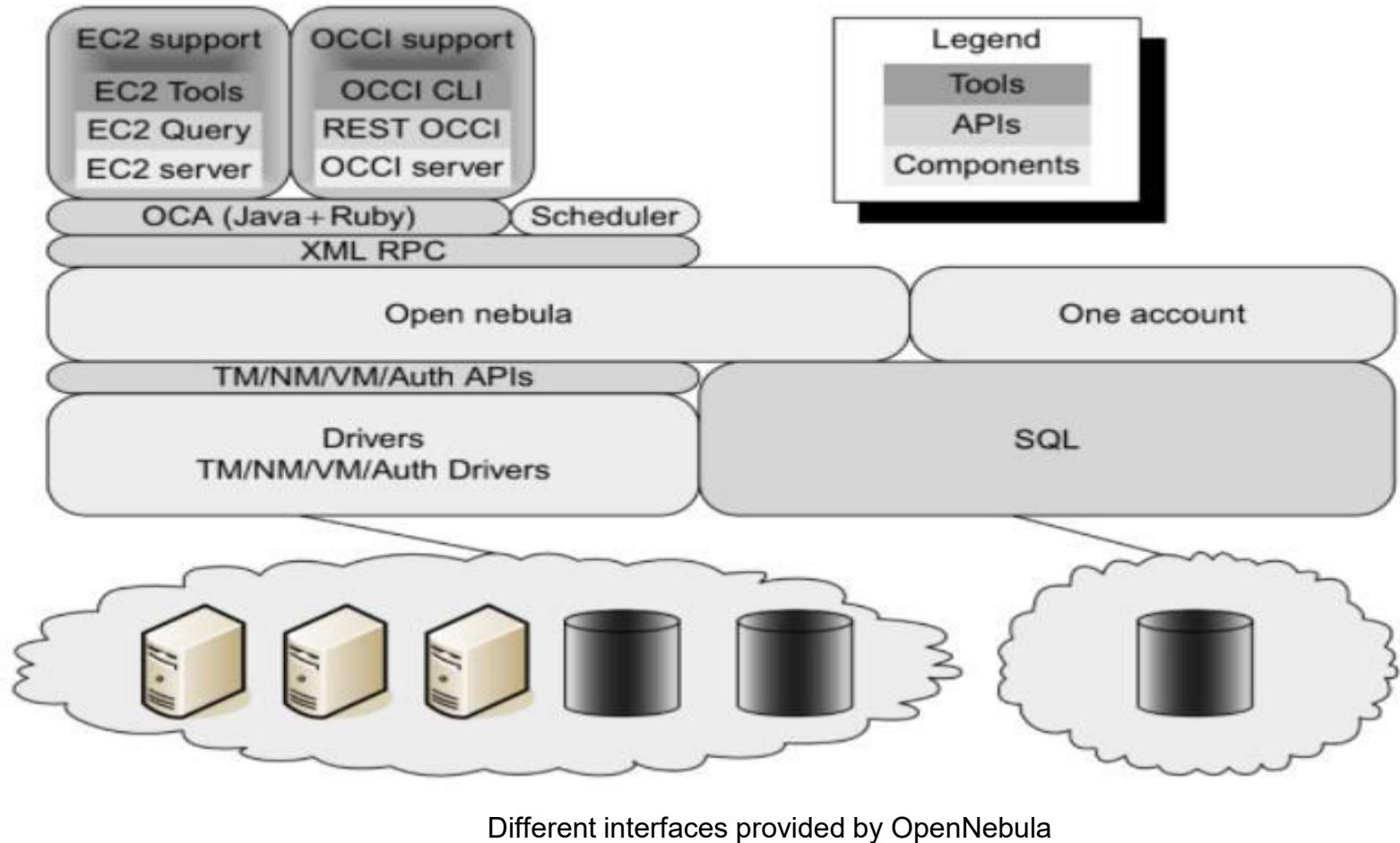
## Need for Cloud Bursting

---

### ❑ Seasonal businesses

Seasonal businesses need additional computational resources during known peak times. For example:

- ✓ Holiday rush shopping for an eCommerce or shipping site
- ✓ End-of-business-quarter financial processing
- ✓ Political election seasons (campaign fundraising and website traffic for education on a candidate's proposals)



### OpenNebula

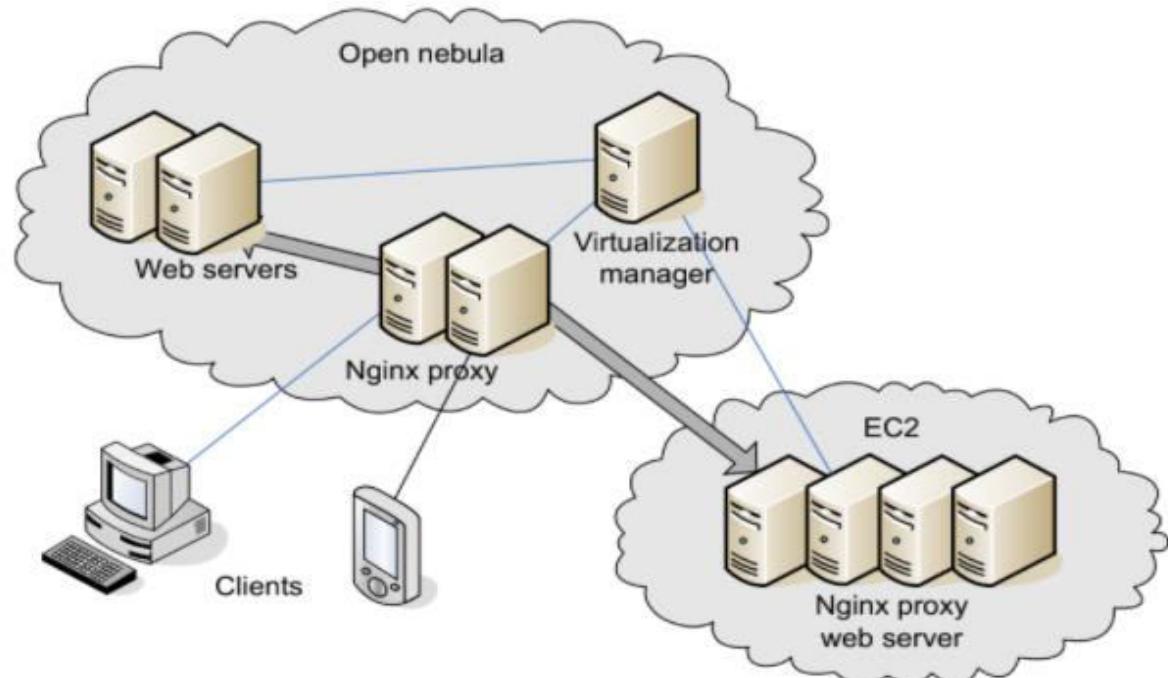
---

- ✓ OpenNebula is an open source cloud middleware solution that manages heterogeneous distributed data centre infrastructures. It is designed to be a simple but feature-rich, production-ready, customisable solution to build and manage enterprise clouds—simple to install, update and operate by the administrators; and simple to use by end users.
  
- ✓ OpenNebula combines existing virtualisation technologies with advanced features for multi-tenancy, automated provisioning and elasticity. A built-in virtual network manager maps virtual networks to physical networks.
  
- ✓ Distributions such as Ubuntu and Red Hat Enterprise Linux have already integrated OpenNebula.



## OpenNebula with reverse proxy for cloud bursting

OpenNebula can be used in conjunction with a reverse proxy to form a cloud bursting hybrid cloud architecture with load balancing and virtualization support provided by OpenNebula. The OpenNebula VM controls server allocation in both the EC2 cloud as well as the OpenNebula cloud, while the Nginx proxy to which the clients are connected distributes load over the web servers both in EC2 as well as the OpenNebula cloud. In addition to web servers, the EC2 cloud also has its own Nginx load balancer.





**THANK YOU**

---

**Jeny Jijo  
Saritha**

Department of Computer Science & Engineering

[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)  
[saritha.k@pes.edu](mailto:saritha.k@pes.edu)





# CLOUD COMPUTING

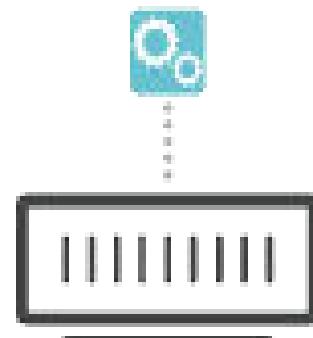
## Multitenancy, Multitenant Databases

---

**Prof . Jeny Jijo**

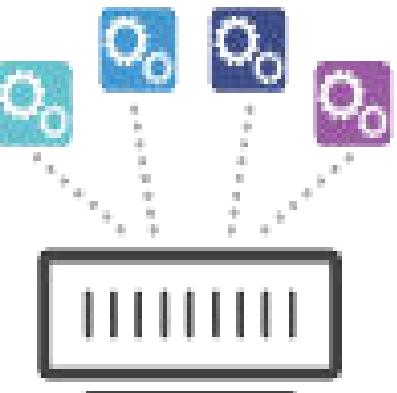
Department of Computer Science & Engineering

- Multitenancy means multiple customers of a cloud vendor are using the same computing resources.
- Multitenancy is an architecture in which a single instance of a software application serves multiple customers. Systems designed in such manner are often called *shared*. A *tenant* is a group of users who share a common access with specific privileges to the software instance.



2

Single Tenant



Multitenant

## Multitenancy - Benefits

---

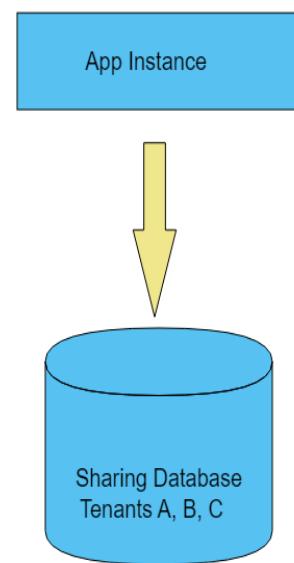
Many of the benefits of cloud computing are only possible because of multitenancy. Here are two crucial ways multitenancy improves cloud computing:

- ✓ **Better use of resources:** One machine reserved for one tenant isn't efficient, as that one tenant is not likely to use all of the machine's computing power. By sharing machines among multiple tenants, use of available resources is maximized.
- ✓ **Lower costs:** With multiple customers sharing resources, a cloud vendor can offer their services to many customers at a much lower cost than if each customer required their own dedicated infrastructure.

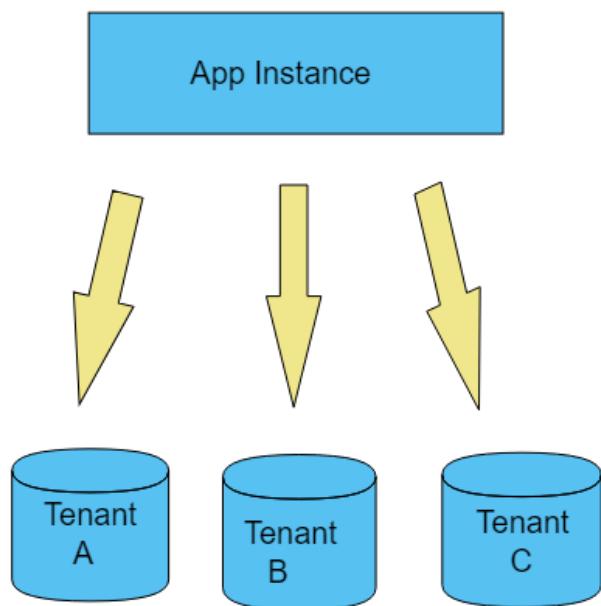
## Multi-Tenancy - Architecture Types

There are 3 main types of multi-tenant architecture:

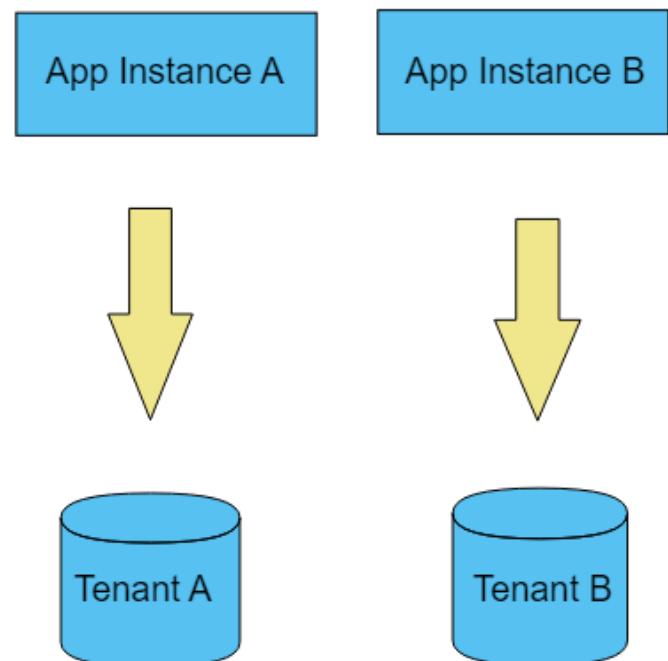
**(1) Multi-tenancy with a single multi-tenant database:** This is the **simplest form of multi-tenancy**. It uses single application instance and the single database instance to host the tenants and store/retrieve the data. This architecture is **highly scalable**, and when more tenants are added the database is easily scaled up with more data storage. This architecture is **low-cost** due to shared resources. Operational complexity is high, especially during application design and setup.



**(2) Multi-tenancy with one database per tenant:** This is another type of multi-tenancy. It uses a single application instance and an individual database for each tenant. The scalability of this architecture may be lower and the cost higher than multi-tenancy with a single multi-tenant database because each database adds overhead. We can increase the scalability of this architecture by adding more database nodes but it depends on the workload. Operational complexity is low due to usage of individual databases.



**(3) Standalone single-tenant app with single-tenant database:** In this architecture you install the **whole application separately for each tenant**. Each tenant has its own app instance as well as database instance. This architecture provides **highest level of data isolation**. The cost of this architecture is high due to the standalone applications and databases.



The finer the granularity of sharing, the greater the scalability of the system.

The increased efficiency in scaling brings with it additional security requirements - access control needs to be specified for each row of the table.

- *Fine grain resource sharing*

- *Security and isolation between customers*

- *Customization of tables*

## Multi-Tenancy - example use case

---

- In the shared machine method, each customer was given their own database process and tables on a shared machine.
- In the shared process method, each customer had their own database tables, but there was only one database process which executed instructions on behalf of all customers.
- In the shared table method, in addition to the customers sharing the database process, the data was stored in shared tables (each row being prefixed with the customer id to indicate the customer to which the row belonged).

**Measurements showed that under the shared machine approach, PostGreSQL used 55MB of main memory and 4 MB of disk memory for storing data for one customer.**

**However under the shared process approach, for 10,000 customers, PostGreSQL used only 80 MB of main memory and 4,488 MB of disk memory.**

**Clearly, the scalability of the system would be much better under the shared process method.**

## Multi-Tenancy - Levels

---

Implementing the highest degree of resource sharing for all resources may be expensive in development effort and complexity of the system. A balanced approach, where there is fine grained sharing of resources for important resources, may be the optimum approach. The four levels of multi-tenancy are :

1. **Ad Hoc/Custom instances:** This is the lowest level where each customer has their own custom version of the software. This represents the situation currently in most enterprise data centers where there are multiple instances and versions of the software. Each customer has their own instance of the software being supported. This makes management extremely difficult, since each customer would need their own management support.
2. **Configurable Instances:** all customers share the same version of the program. Customization is possible through configuration and other options. Customization includes the ability to put the customer's logo on the screen, tailoring of workflows to the customer's processes, and so on. In this level, there are significant manageability savings over the previous level, since there is only one copy of the software that needs to be maintained.

3. **Configurable, multi-tenant efficient instances:** Cloud systems at this level in addition to sharing the same version of the program also have only one instance of the program running which is shared among all the customers. This leads to additional efficiency since there is only one running instance of the program.

4. **Scalable, configurable, multi-tenant efficient instances:** In addition to the attributes of the previous level, the software is also hosted on a cluster of computers, allowing the capacity of the system to scale almost limitlessly. The number of customers can scale from a small number to a very large number, and the capacity used by each customer can range from being small to very large. Performance bottlenecks and capacity limitations that may have been present in the earlier level are eliminated.

The drawback of shared storage devices is that security requirements are greater.

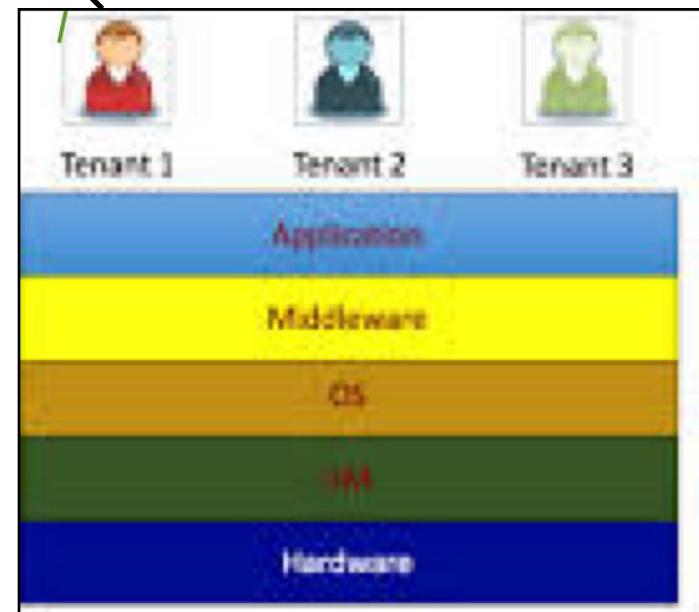
## Multi-Tenancy - Tenants and Users

The **customers** of a SaaS or PaaS service are referred to as **tenants**, regardless of whether they are businesses or users (in the case of a service like Gmail).

The term **user** continues to be used for the actual users of the service.

Users are people that stay in an apartment.

Think of tenant as a family that rents an apartment and its services like gym etc.



The key challenge in multi-tenancy is the *secure sharing of resources which can be enforced through authentication*. Two approaches can be used:

- ✓ a centralized authentication system or
- ✓ a decentralized authentication system

In the ***centralized system***, all authentication is performed using a centralized user data base. The cloud administrator gives the tenant's administrator rights to manage user accounts for that tenant. When the user signs in, they are authenticated against the centralized database.

In the ***decentralized system***, each tenant maintains their own user data base, and the tenant needs to deploy a federation service that interfaces between the tenant's authentication framework and the cloud system's authentication service.

## Multi-Tenancy - Implementing Resource Sharing

---

The key technology to ensure secure resource sharing in a multi-tenant service is **access control**. Two forms of access control can be provided in a cloud service provider –

- roles
- business rules

**Roles** consist of a set of permissions included in the role;

A storage administrator role may include the permissions to define storage devices.

The ability to specify roles for users, is itself a permission that only certain roles can possess.

***Business rules*** are policies that provide fine-grained access control than roles provide, since they may depend upon the parameters of the operation.

EG - In a banking application, it may be possible to specify a limit on the amount of money a particular role can transfer, or specify that transfers can occur only during business hours.

Business rules can be implemented using policy engines such as Drools Expert and Drools Guvnor.

There are two types of **access control models**.

**1. access control lists** (ACL) where each object is attached with a list of permissions for each role.

**2. capability-based access control**, which works just like a house-key. If a user holds a reference or capability to an object, he has access to the object. The key is the link to the object; the virtue of the user having this key grants him access to the object.

Two major resources that need to be shared are **storage** and **servers**.

**1. Sharing storage resources:** In a multi-tenant system, many tenants share the same storage system. Cloud applications use two kinds of storage systems:

- file systems and
- databases, - relational and NoSQL databases

File systems are shared to specified users via ACLs and other mechanisms.

There are two methods of *sharing data in a single database* –

- ✓ table sharing
- ✓ dedicated tables per tenant

In the ***dedicated table method***, each tenant stores their data in a separate set of tables different from other tenants restricting access to users.

Eg -

If the database is a relational database, the three garages can be registered as database users, and access rights can be set by the SQL statement SQL GRANT SELECT, ..., ON FriendlyTable TO FriendlyGarage **WITH GRANT OPTION**.

This statement gives access rights to the table FriendlyTable to the database user FriendlyGarage. The WITH GRANT OPTION clause allows the tenant to further give access rights to other database users.

Best Garage

Car License	Service	Cost

Friendly Garage

Car License	Service	Cost

Honest Garage

Car License	Service	Cost

- In the ***shared table*** approach, the data for all tenants is stored in the same table in different rows.
- The shared table method is space-efficient than the dedicated table method but multiples SELECT statements are required.
- An auxiliary table, called a metadata table, stores information about the tenants.

Data Table 1

Tenant Id	Car License	Repair	Cost
1			
2			
2			
1			
3			
2			

Metadata Table 1

Tenant Id	Data
1	Best Garage
2	Friendly Garage
3	Honest Garage

2. Sharing compute resources: In the *dedicated table* method, each tenant has their own set of files so, operating system features for security ensures that one tenant cannot read the tables of another tenant.

In the *shared table* case, the cloud system clearly relies upon the application to ensure security of the data

### 3. Customization

Cloud infrastructure should provide customization of stored data. Three methods are:

**1. Preallocated columns:** Space is reserved in the tables for **custom columns**, which can be used by tenants for defining new columns. In the pre-allocated columns technique is that there could be a lot of wasted space. Less columns users are constrained, more number of columns leads to space wastage. (Refer Figure in next slide)

**2. Name-Value pair:** This method, has an extra column which is a pointer to a table of name-value pairs. The name-value pair method is space efficient, but it is necessary to re-construct the data before it is used by *joins*.

**3. XML Method:** The final column is an XML Document where records of any arbitrary structure can be stored.

Data table 1

Tenant Id	Car license	Service	Cost	Custom1	Custom2
1					
2					
2					
1					
3					
2					

Metadata table 1

Tenant Id	Tenant name	Custom1 name	Custom1 type
1	Best garage	Service rating	int
2	Friendly garage	Service manager	string
3	Honest garage		

**FIGURE 6.12**

Pre-allocated columns.

Data table 1

Tenant Id	Car license	Service	Cost	Name-value pair rec
1				275
2				
2				
1				
3				
2				

Data table 2 (name-value pairs)

Name-value pair rec	NameID	Value
275	15	5.5

Pivot table

Metadata table 1

Name Id	Name	Type
15	Service rating	int
	Service manager	string

Metadata table 2

Tenant Id	Data
1	Best garage
2	Friendly garage
3	Honest garage

FIGURE 6.13

Name-value pairs.



THANK YOU

---

Jeny Jijo

Department of Computer Science & Engineering

[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)



## CLOUD COMPUTING

Failure detection – Checkpointing &  
Application recovery

---

Prof . Jeny Jijo

Department of Computer Science & Engineering

- Failure detection is valuable to distributed systems as it adds to their reliability and increases their usefulness.
- Hence, it is important for distributed systems to be able to detect and cater to failures when they occur efficiently and accurately
- It is proved that consensus cannot be solved in a totally asynchronous model.
- However by augmenting the asynchronous system model with failure detectors it would be possible to bypass the impossibility
- Failure detection should be a component of the operating system.
- All failure detection algorithms/schemes use **time as a means to identify failure**. There are different protocols/ways of using this tool. They vary on how this timeout issue should be addressed, when/where messages should be sent and how often, synchronous or asynchronous.
- For small-distributed networks, such as LANs etc., coordination and failure detection is simple and does not require much complexity.

An independent failure management system should be such that when it is investigating a service that is not responding, it will contact the Operating system that is running it to obtain further confidence. Their independent failure detector should have the following **three functional modules**:

1. A library that implements simple failure management functionality and provide the API to the complete service.
2. A service implementing per node failure management, combining fault management with other local nodes to exploit locality of communication and failure patterns.
3. An inquiry service closely coupled with the operating system which, upon request, provides information about the state of local participating processes.

*Two types of support can be built into the cloud infrastructure for high availability.*

- ✓ The first technique is **failure detection**, where the cloud infrastructure detects failed application instances, and avoids routing requests to such instances.
- ✓ The second technique is **application recovery**, where failed instances of application are restarted.

## Failure Detection - Introduction

---

- Cloud providers, such as Amazon Web Services' Elastic Beanstalk, detect when an application instance fails, and avoid sending new requests to the failed instance. To detect failures, one needs to monitor for failures.

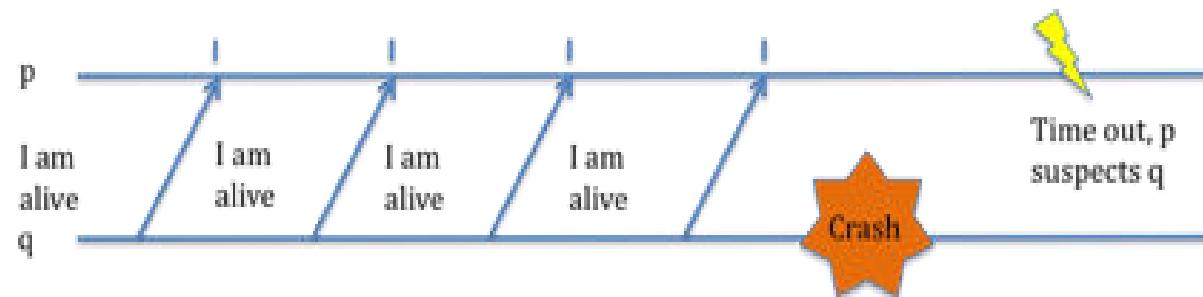
**1) Failure Monitoring:** There are two techniques of failure monitoring. The ***heartbeats***, is an application instance periodically sends a signal (called a heartbeat) to a monitoring service in the cloud. If the monitoring service does not receive a specified number of consecutive heartbeats, it may declare the application instance as failed.

***probes*** periodically sends a probe, which is a lightweight service request, to the application instance. If the instance does not respond to a specified number of probes, it may be considered failed.

The two most famous ones are the push and pull strategies.

### The Push Model

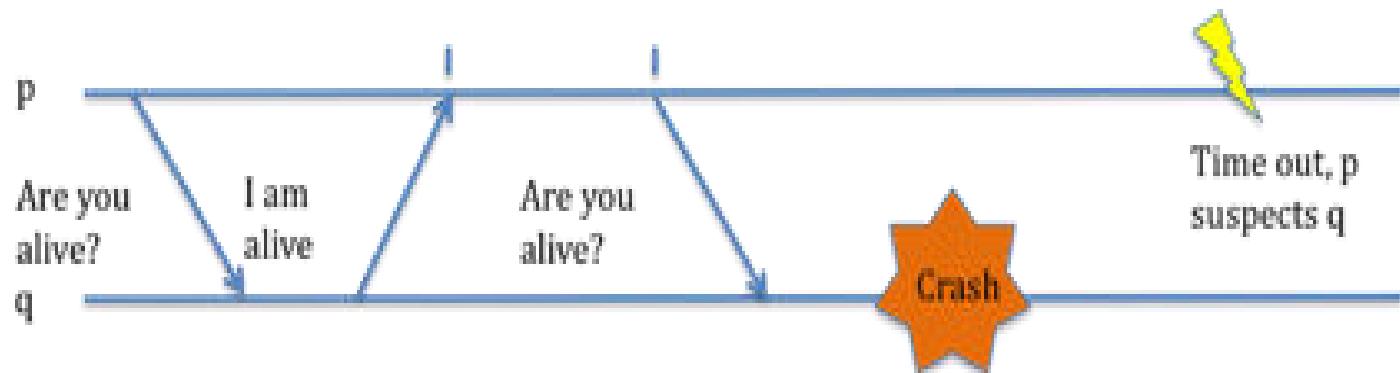
Assuming that we have two processes, p and q. With p being the process that is the monitor. Process q will be **sending heartbeat messages every t seconds**, hence process p will be expecting a “I am alive message” from q every t seconds. Heartbeat messages are messages that are sent on a timely bases to inform/get informed about a process. If after a timeout period T, p does not receive any messages from q, then it starts suspecting q.



process q is sending heartbeat messages periodically. At one point it crashes and it stops sending messages. Process p, waiting for heartbeat messages, at this point does not receive any more messages, and after a timeout period of T, suspects that q has failed.

## The Pull Model

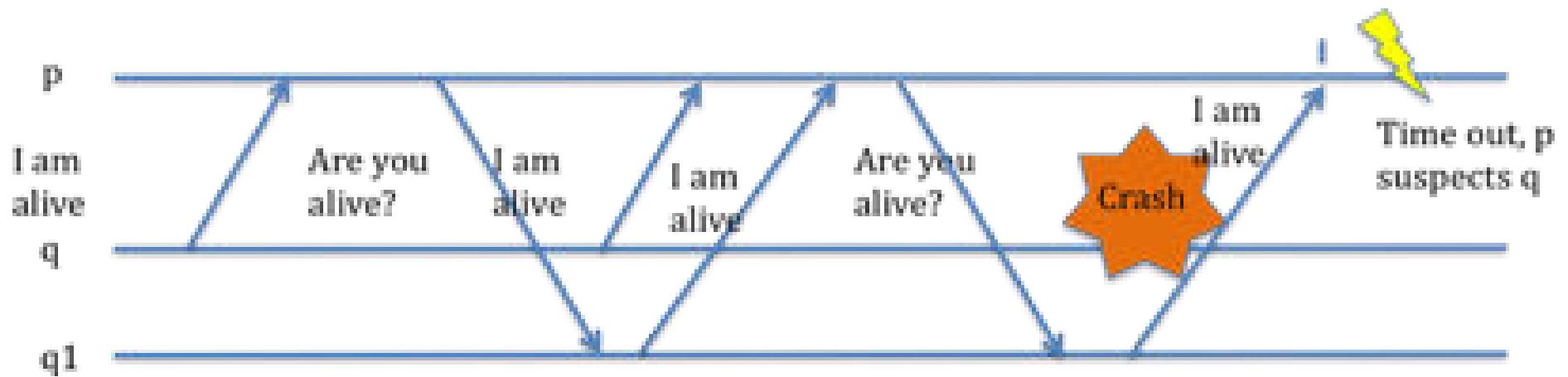
Assume that we have the same parameters as Push model . In this model, instead of the process q (the one that has to prove its alive) sending messages to p every few seconds, it sends 'are you alive?' messages to q every t seconds, and waits for q to respond 'yes I am alive].



In the above figure, p repeatedly checks if q is alive, and if q is alive it will respond with the 'I am alive' heartbeat message. Once q crashes, p does not receive any more responses from q and starts suspecting the failure of q after a timeout T time.

### The Dual Scheme

The pull model is somehow inefficient as there are potentially too many messages sent between the processes. The push model is a bit more efficient in this manner. Hence, a model is proposed that is a mix of the two. During the first message sending phase, any q like process that is being monitored by p, is assumed to use the push model, and hence send “ I am alive” or “liveness” messages to p. After some time, p assumes that any q like process that did not send a liveness message is using the pull model.



## Failure Detection.

---

There is a trade-off between *speed* and *accuracy* of detecting failures.

To detect failures rapidly, set a low value for the number of missed heartbeats or probes. This could lead to an increase in the number of false failures. An application may not respond due to a momentary overload or some other transient condition. Since the consequences of falsely declaring an application instance failed are severe, generally a *high threshold is set* for the number of missed heartbeats or probes.

### 2) Redirection

After identifying failed instances, it is necessary to avoid routing new requests to these instances.

A common mechanism used in HTTP-based protocols is HTTP-redirection.

The Web server returns a 3xx return together with a new URL to be visited.

For example, if a user types “[http:// www.pustakportal.com/](http://www.pustakportal.com/)” into their browser, the request may first be sent to a load-balancing service at Pustak Portal, which may return a return code of 302 with a URL “<http://pps5.pustakportal.com>”. [pp5.pustakportal.com](http://pps5.pustakportal.com) is the address of a server that is currently known to be up, the user is directed to a server that is up.

In addition to directing new requests to a server that is up, it is necessary to recover old requests.

**Check-point/restart**: The cloud infrastructure periodically saves the state of the application. If the application has failed, the most recent check- point can be activated, and the application can resume from that state.

In a **distributed checkpoint/restart**, all processes of distributed application instances are checkpointed, and all instances are restarted from a common checkpoint if any instance fails. This has obvious scalability limitations and also suffers from correctness issues if any inter process communication data is in-transit at the time of failure.



**THANK YOU**

---

**Jeny Jijo**

Department of Computer Science & Engineering

[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)



## CLOUD COMPUTING

### Cloud Security Requirements

---

**Jeny Jijo**

Department of Computer Science & Engineering

# CLOUD COMPUTING

---

## Cloud Security Requirements

**Jeny Jijo**

Assistant Professor, Department of Science and Humanities

Trust and security become even more demanding for web and cloud services, because leaving user applications completely to the cloud providers has faced strong resistance by most PC and server users. Cloud platforms become worrisome to some users for lack of privacy protection, security assurance, and copyright protection.

### What is Cloud Security?

Cloud security is a set of control-based safeguards and technology protection designed to protect resources stored online from leakage, theft, or data loss. Protection encompasses cloud infrastructure, applications, and data from threats.

### Securing Cloud Data

#### Data Loss

*BYOD*



*External threats*



*Internal threats*



#### Data Privacy

*Regulatory Compliance*



*Data Sovereignty*



*Data Remanence*



## Cloud Security Requirements- Physical and Virtual security

At a high level, the cloud infrastructure can be partitioned into a *physical infrastructure, and a virtual infrastructure*. The security requirements and best practices can also similarly be divided into the requirements for physical security and those for virtual security.

### *Basic objectives of cloud security*

- confidentiality,
- integrity
- availability of the cloud system.

### *additional objectives*

- Cost-effectiveness:
- Reliability & performance

### Physical Security

Physical security implies that the data center the cloud is hosted in should be *secure against physical threats*. This includes not only attempts at penetration by intruders, but also protection against natural hazards and disasters such as floods, and human error such as switching off the air conditioning. It is important to note that *security is only as strong as its weakest link*.

## Cloud Security Requirements

---

*To ensure physical security, a multi-layered system is required which includes*

- i. A central monitoring and control center with dedicated staff
- ii. Monitoring for each possible physical threat, such as intrusion, or natural hazards such as floods
- iii. Training of the staff in response to threat situations
- iv. Manual or automated back-up systems to help contain threats (e.g., pumps to help contain the damage from floods)
- v. Secure access to the facility. This requires that the various threats to the data center be identified, and appropriate procedures derived for handling these threats

### Virtual Security

The following best practices have been found to be very useful in ensuring cloud security.

#### ✓ *Cloud Time Service*

If all systems in *the datacenter are synchronized to the same clock*, this is helpful both to *ensure correct operation of the systems*, as well as to facilitate later analysis of system logs. It is particularly important in correlating events occurring across geographically distributed systems. A common way to do this is by use of the *Network Time Protocol (NTP)* which is a protocol that synchronizes the clock on a computer to a reference source on the Internet . To protect against false reference sources, the protocol messages can be encrypted.

## Cloud Security Requirements- Virtual

---

### ✓ *Identity Management*

Identity management is a foundation for achieving confidentiality, integrity and availability. Some of the requirements for identity management are that:

- i. It should scale to the number of users typically found in a cloud system
- ii. In heterogeneous cloud systems, a federated identity management system that allows establishing a single identity and single signon across multiple different types of systems may be needed.
- iii. The IMS should satisfy applicable legal and policy requirements (for example, allow deleting of users across the system within a specified time period)
- iv. Maintain historical records for possible future investigation.

## Cloud Security Requirements - Virtual

---

### ✓ *Access Management*

The core function of access management is to *allow accesses to cloud facilities only to authorized users*. However, additional requirements are to:

- i. Not allow unrestricted access to cloud management personnel
- ii. Allow implementation of multi-factor authentication (e.g., use of a password together with a digital key) for very sensitive operations.

*It is also good practice to:*

- i. Disallow shared accounts, such as admin
- ii. Implement white-listing of IP addresses for remote administrative actions.

### ✓ *Break-Glass Procedures*

It is desirable for the access management system to **allow alarmed break-glass procedures, which bypass normal security controls in emergency situations.** The analogy is with breaking the glass to set off a fire alarm. It is important to ensure that the break-glass procedure can be executed only in emergencies under controlled situations, and that the procedure triggers an alarm.

## Cloud Security Requirements- Virtual

---

### ✓ *Key Management*

In a cloud, with shared storage, encryption is a key technology to ensure isolation of access. The cloud infrastructure needs to provide secure facilities for the generation, assignment, revocation, and archiving of keys. It is also necessary to generate procedures for recovering from compromised keys.

### ✓ *Auditing*

The audit should capture all security-related events, together with data needed to analyze the event such as the time, system on which the event occurred, and userid that initiated the event. The audit log should be centrally maintained and secure and should be possible to sanitize or produce a stripped-down version of the audit log for sharing with cloud customers, in case their assistance is needed to analyze the logs.

### ✓ *Security Monitoring*

This includes an infrastructure to generate alerts when a critical security event has occurred, including a cloud-wide intrusion and anomaly detection system.

The intrusion detection systems may be installed both on the network as well as the host nodes. It may also be necessary to allow cloud users to implement their own intrusion and anomaly detection systems.

### ✓ *Security Testing*

It is important to test all software for security *before deployment* in an isolated test bed. Patches to software should also be tested in this environment before being released into production. Additionally, security testing should be carried out on an *ongoing basis to identify vulnerabilities in the cloud system*. Depending upon the risk assessment, some of these tests may be carried out by third parties. There should also be a remediation process to fix identified vulnerabilities.



THANK YOU

---

Jeny Jijo  
[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)



## CLOUD COMPUTING

### Risk Management & Security Design Patterns

---

Jeny Jijo

Department of Computer Science & Engineering

# CLOUD COMPUTING

---

## Risk Management & Security Design Patterns

**Jeny Jijo**

Assistant Professor, Department of Science and Humanities

## Why to use a risk approach for cloud selection?

---

- Many organizations are embracing cloud computing, it's a rage these days
- Data security risks-** Do you trust an external third party with your sensitive data?
- Prepared for cloud failure ([\*cloud outages at Microsoft and Amazon\*](#)) ??
- In March 2009, Microsoft Windows Azure was down for 22 hours
- In April 2011, a large scale outage hit Amazon, affecting Amazon's Web Services' Elastic Compute Cloud (EC2).
- These outages prevent users from accessing applications or data stored in the cloud and the financial cost of these outages can be quite high especially when mission critical- such as accounting information systems are outsourced

## Why to use a risk approach for cloud selection?



**ZDNet**

Hot Topics Newsletters Reviews Downloads White Papers

Log In | Join ZDNet

Edition Cloud Security IT Jobs CXO SaaS Storage Smartphones All India SMBs

Topic: Amazon

Follow via:

The best of ZDNet, delivered

## Amazon Web Services suffers outage, takes down Vine, Instagram, others with it

**Summary:** The cloud giant suffered an outage for about an hour on Sunday, showing once again the perils of an outsourced cloud service, as many AWS customers went down with it.

By Zack Whittaker for Between the Lines | August 26, 2013 — 13:22 GMT (18:52 IST)  
Follow @zackwhittaker

Comments 0 Share on Facebook

more +

North America South America Europe Asia Pacific

	Aug 25	Aug 24	Aug 23	Aug 22	Aug 21
Amazon CloudFront	✓	✓	✓	✓	✓
Amazon CloudSearch (N. California)	✓	✓	✓	✓	✓
Amazon CloudSearch (N. Virginia)	✓	✓	✓	✓	✓

## Why to use a risk approach for cloud selection?

---



25th August 2013

- ✓ Amazon Web Services (AWS), one of the world's largest cloud provider, stumbled **over on Sunday for 59 minutes**, due to issues with its U.S.-EAST datacenter.
- ✓ This led to "degraded experience," resulted in a "small number of EC2 instances unreachable due to packet loss in a single"
- ✓ The biggest casualty of the outage, however, was **Amazon.com itself**, which rejected customers from accessing its site in the U.S. and Canada.
- ✓ Other Amazon-owned websites also suffered, including Audible.com, while Netflix continued to power through the problems.

## What is Risk Management?

---

**Risk management** is the process for evaluating risks, deciding how they are to be controlled, and monitoring their operation. When managing risk, a **number of factors** need to be kept in mind.

- The suitable approaches in different domains (e.g., finance and healthcare) may be different.
- There should be a careful trade-off between the impact of the risks involved, and the cost of the security measure, measured both in terms of impact to usage of the system as well as the actual cost of security

## Risk Management concepts

---

A **security control** is a safeguard (process or system function) to prevent, detect or respond to a security risk. NIST divides security controls into three broad categories – *Technical, Operational, and Management*. The controls in each category are further sub-divided into 18 families.

Eg - Audit and Accountability is one of the families, and Response to Audit Processing Failures is one of the security controls in this family.

FIPS 200 defines the security requirement of the system as *low-impact, moderate impact, or high-impact* depending upon the impact of a security breach in the system.

The motivation behind this definition is that *high-impact systems have the greatest requirement in terms of security controls.*

A system is defined to be a *low-impact system* if the result of a security breach is that there is a *limited degradation in capability, but the system is able to perform its primary functions.*

*Moderate impact systems* are those where the system is able to perform its primary functions, but there is a *significant degradation in the functions*; while high impact systems are those where the system is incapable of performing some of its primary functions.

### 1) Information Resource Categorization:

The first step is to evaluate each information resource in the organization from the perspectives of:

- a. Criticality – the impact to the business of a security failure
- b. Sensitivity – the confidentiality of the resource

This evaluation will determine the level of security to be provided for each resource.

### 2) Select Security Controls

Security controls appropriate to the criticality and sensitivity of the information resources need to be selected.

Eg –Whitelisting IP addresses is a security control, and forces administrators to work only from within office premises, which may be an acceptable security measure. This control may not be appropriate for the user network. If the user network is also protected by whitelisting IP addresses, the user would have to register a new IP address for whitelisting each time they go to a new location and try to access the corporate network.

### 3) Risk Assessment:

After deciding upon the security controls to be implemented, it is necessary to determine if the controls provide adequate protection against the anticipated threats, and to augment the controls if more protection is needed.

### 4) Implement Security Controls:

Next, the security controls decided upon would need to be implemented. These security controls may be administrative, technical or physical.

### 5) Operational Monitoring:

Once the security controls are implemented, their effectiveness in operation should be continuously monitored.

### 6) Periodic Review:

The security controls in place should be periodically reviewed to determine if they continue to be effective. The need for review comes from the fact that:

- a. New threats may appear
- b. Operational changes (e.g., new software) may result in requiring a change in security design.

## Security Design Patterns

*A design pattern is a class or category of designs that can be customized to fit a particular situation.*

- ***Defense in Depth***

It states that defenses should be layered, so that an attacker has to overcome multiple defenses before gaining access to important resources. In a similar way, remote administrative access to a cloud system could be allowed only through a VPN. For further protection, this access could be allowed only from white-listed IP addresses. Furthermore, an administrator may be required to provide a one-time password for additional security.

- *Honeypots*

A honeypot is a *decoy computer system* that appears attractive to an attacker. While the attacker is attacking the honeypot under the impression that it is a worthwhile system to control, they can be observed by security personnel who can then attempt to trap and control the attack.

*Honeypots are widely used in network security.* In the cloud context, a honeypot virtual machine can be deployed, which would monitor any suspicious attempt to break into the virtual machine. *Honeypots can be deployed both by the cloud service provider as well as cloud customers.*

- **Sandboxes**

Sandboxing refers to a technique in which *software is executed in a restricted environment inside the operating system* in which it is running. Since the software is being executed in a restricted environment, an attacker who breaks into the software does not have unrestricted access to the facilities provided by the operating system. This limits the amount of damage an attacker who has gained control of the software can do; additionally, the attacker has to overcome the sandbox if they have to gain full control over the operating system. Sandboxes thus provide defense in depth as well.

- *Network Patterns*

### ✓ VM Isolation

New techniques have to be used to isolate traffic between VMs that share the same physical hardware, since this traffic does not enter the switching network.

The security features offered by the VM are:

- i. Encryption of traffic between VMs
- ii. Tightened security controls on VMs, such as the ports that will accept traffic.

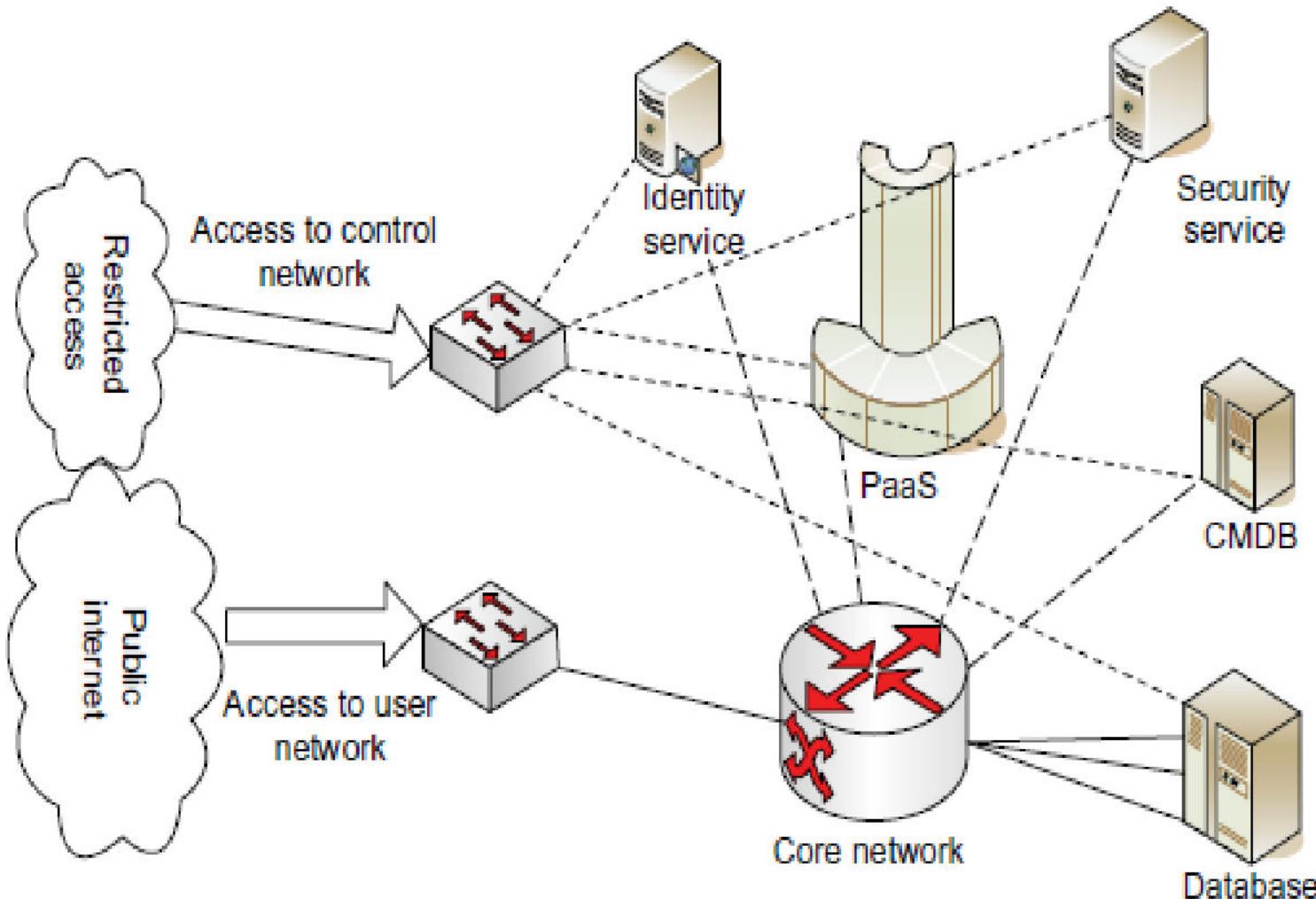
### ✓ Subnet Isolation

It is good practice to have physically separate traffic for administrative network traffic, customer network elements, and storage networks. Physically separate networks are preferred due to the possibility of mis-configuration in virtual LANs (VLAN) that are not physically separate. Routing between the networks can be handled by firewalls.

- *Common Management Database*

A Common Management Database (CMDB) is a database that contains information regarding the components of an IT system. The information includes an inventory of the components, as well as their present configuration and status. The presence of a CMDB simplifies implementation and management of an infrastructure, including security, since it ensures that all administrative components have a single consistent view of all the components.

## Example - Security Design for a PaaS System



A cloud service provider who wants to expose a DBMS via the cloud as a PaaS (and DaaS) offering. The design satisfies the cloud security requirements and leverages the design patterns presented.

## Example - Security Design for a PAAS System

---

### ✓ External Network Access

The figure shows two entry points into the cloud network. The first is the access to the control network for administration. The second is the interface used for public access to the cloud services. *Both are distinct, and lead to separate physical networks.*

To provide defense in depth, the control network access can be limited to whitelisted IP addresses.

*Multi-factor authentication* can be made mandatory for increasing secure access to administrative functions. *The access to the public network is via two switches, to increase availability via redundancy.*

## Example - Security Design for a PaaS System

---

### ✓ Internal Network Access

The management network is physically separated from the public (or user) network, in order to reduce the risk of an attacker accessing the cloud via the public network and attempting to access the management network.

The DBMS is connected to the public network via an aggregated set of links to provide increased bandwidth as well as availability. The DBMS server may be accessed from the internal PaaS service as well.

Similarly, the PaaS service may also be accessed from both the internal network as well as the external network. However, the security server, which performs audit and other security functions, need not be accessible from the external network.

## Example - Security Design for a PAAS System

---

### ✓ *Server Security*

The database server is managed by the cloud provider, and the database is offered as a service to the customers. Access to the cloud services can be managed via *the identity server shown together with access control*. Hence, the security of the database is managed by the cloud service provider. The database itself can be secured by disallowing access to unneeded ports in the server, and by implementing an intrusion detection system on the server hosting the database. *An additional security layer can be implemented by checking the ODBC connections to the database.*

## Example - Security Design for a PAAS System

---

### ✓ *Security Server*

The diagram also includes a security server to perform security services, including auditing, monitoring, hosting a security operations center, and security scanning of the cloud infrastructure.



THANK YOU

---

Jeny Jijo  
[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)



## CLOUD COMPUTING

### Security Architecture ,Legal & Regulatory Issues

---

**Jeny Jijo**

Department of Computer Science & Engineering

# CLOUD COMPUTING

---

## Security Architecture ,Legal & Regulatory Issues

**Jeny Jijo**

Assistant Professor, Department of Science and Humanities

- **SSE-CMM**

The System Security Engineering Capability Maturity Model is an adaption of the well-known Capability Maturity Model (CMM) for software engineering by Carnegie Mellon University. It is the basis of the ISO/IEC 21827 standard. Similarly to CMM, it defines five capability levels for any organization, and allows organizations to assess themselves, and put in place processes to improve their levels.

- ***ISO/IEC 27001-27006***

This is a set of related standards under the ISO/IEC 27000 family that provides *an Information Security Management System*. It specifies a set of requirements that organizations must satisfy (e.g., there should be a process to systematically evaluate information security risks). The ISO/IEC 27000 family of standards is not specific to cloud security.

- *European Network and Information Security Agency (ENISA)*

The Cloud Computing Information Assurance Framework from ENISA is a *set of assurance criteria designed to assess the risk of adopting cloud services, compare different Cloud Provider offers, obtain assurance from the selected cloud providers, and reduce the assurance burden on cloud providers*. It is based upon applying ISO/IEC 27001-27006 to cloud computing. In addition to assurance criteria, ENISA offers an assessment of the benefits and risks of cloud computing.

- ***ITIL Security Management***

ITIL (Information Technology Infrastructure Library) is a well-known and *comprehensive set of standards for IT service management*. The section on security management is based upon ISO/IEC 27002. An advantage of using ITIL Security Management is that ITIL itself is very common in many data centers, hence the use of ITIL-SM will have a smaller learning curve and result in an integrated security solution.

- **COBIT**

ISACA, an international organization devoted to *the development of IT governance standards*, has developed the Control Objectives for Information and Related Technology. This is a set of processes and best practices for *linking business goals to IT goals, together with metrics and a maturity model.* COBIT is broader in scope than ISO/IEC 27000, since it relates to IT governance.

- **NIST**

The US National Institute of Standards and Technology has a number of whitepapers and other resources in its Security Management & Assurance working group. These are targeted at U.S. federal agencies; however, many of the recommendations will apply to other organizations as well.



Cloud computing may involve additional legal and regulatory issues due to the possible involvement of a third party – the cloud service provider. Since local, national, and international laws may apply (*due to the geographical distribution of the cloud infrastructure*) it is important to consider the impact of these laws

The laws and regulations will typically specify *who is responsible for the accuracy and security of the data*. Any breach will lead to the imposition of penalties and fines by regulatory bodies, as well as prosecution of officials held responsible for complying with the laws.

- First, *covering the risks arising from the presence of a third party* (the cloud service provider) is considered.
- The second set of issues arises from the need for *ensuring data security*.
- The third set of issues deals with the *obligation of the cloud service provider during litigation*.

### 1) Third-party /Contractual Issues

#### ✓ *Due diligence*

The enterprise should define the scope of the cloud services needed and then specify the *regulations and compliance standards* that need to be adhered. The process of due diligence may rule out a cloud service provider (because they are not able to satisfy the applicable laws) or limit the scope of the cloud service that can be utilized.

Due diligence process should also *consider risks arising from the stability and reliability of the cloud service provider* (e.g., are they likely to exit the cloud service market) and the criticality of the business function being outsourced.

### ✓ *Contract negotiation*

The next phase after the due diligence process is to *negotiate a contract with the cloud service provider*. Unlike traditional outsourcing contracts, it is possible that *cloud service providers may have a standard online click-through agreement that is not customizable*. In many cases, where the risk is low, a standardized agreement may be acceptable. One way cloud service providers can avoid having to negotiate custom agreements with each customer is through external accreditation(Eg-Statement on Auditing Standards or SAS 70)

### ✓ *Implementation*

Next, the enterprise has to ensure that the *safeguards laid out in the contract are actually being followed*. For example, if sensitive data is to be handled differently, then it is important to check that the appropriate procedure is actually followed.

Additionally, it is important to *continuously re-evaluate the system to check for changed circumstances* (e.g., the sensitivity of the data being outsourced has increased, or an external accreditation has been revoked).

✓ *Termination*

Contract terminations (normal or otherwise) are the times when compliance is most at risk, since the service provider may be reluctant to take adequate security precautions. Therefore, it is important to identify an alternative service provider and ensure timely and secure transfer of the services. It is also extremely important to *ensure that sensitive data is deleted from the original service provider's system.*

### 2) Data Handling

#### ✓ *Data Privacy*

Organizations have to protect the privacy of the data that they have collected, and use the collected data only for the purpose for which it was collected. Organizational data cannot generally be sold to third parties. These restrictions have to be followed by subcontractors, including cloud service providers.

*Privacy laws often state that individuals can access their own data and modify or delete it.* It is necessary to ensure that a cloud service provider makes the same facilities available in a timely manner.

### ✓ *Data Location*

Laws on the handling of data differ from country to country. Therefore, transferring confidential data between countries may be problematic. In a cloud context, the location of the data centers and backups needs to be known in advance, to ensure that legal problems do not arise. This is one of the reasons that Amazon allows specification of a region for its storage services that governs the location of the data center where the data is to be stored. *The large difference between laws in different countries implies that if data is stored in multiple countries, then the enterprise has to abide by the most stringent set of data storage laws.*

✓ *Secondary Use of Data*

In addition to disallowing unauthorized access to data, enterprises need to *ensure that cloud service providers do not use the data for data mining or other secondary usage*. To ensure this, it is necessary to carefully read the service agreements exhibited by cloud service providers before clicking on the I Agree button.

### ✓ *Business Continuity Planning and Disaster Recovery*

Most organizations would have implemented Business Continuity Planning (BCP) to ensure continued operation in the face of any catastrophic and unforeseen disaster, such as a terrorist attack or earthquake. BCP typically involves identifying the possible catastrophes, carrying out Business Impact Analysis, and using the results of the analysis to formulate a recovery plan. Disaster Recovery (DR) is the part of the BCP that involves the recovery of IT operations. Since IT operations have become increasingly critical, DR is a very important part of a BCP.

When using a public cloud provider, it is important that the *BCP and DR be expanded to include catastrophes that impact the public cloud provider.* In the case of a natural disaster or other calamity, a cloud service provider's datacenters may become unavailable.

The disaster recovery plan should be formulated before deploying applications to the cloud, and implemented during deployment (e.g., by performing regular backups of data). Additionally, the cloud service provider's DR plans should be studied. It is important to use features that the cloud service provider may provide for DR, such as the use of multiple data locations.

### ✓ *Security Breaches*

In the eventuality of a security breach, it is necessary to be informed of the breach as quickly as possible so that corrective action can be taken. It is therefore necessary to understand the cloud service provider's disclosure policy, and understand how quickly they would notify their customers. To avoid ambiguity, the service agreement should specify the actions to be taken during a breach.

## Litigation Related Issues

---

Another set of issues arise from the obligations of the cloud service provider during litigation. *The litigation may involve either the business that is using the cloud or the cloud service provider itself.* If the business is involved in litigation, and is asked to make available certain data as part of the court proceedings, it is important to know if the cloud service provider can satisfactorily respond to the request. This is important since the courts will hold the business, and not the cloud service provider, responsible for responding to the request.

It is also possible that the cloud service provider may be asked directly to provide some data as part of litigation involving the business. In this case, it is important that the cloud service provider notify the business in a timely manner, to allow the business to contest the request if needed.



THANK YOU

---

Jeny Jijo  
[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)



## CLOUD COMPUTING

### Authentication in cloud- Keystone

---

**Jeny Jijo**

Department of Computer Science & Engineering

# CLOUD COMPUTING

---

## Authentication in cloud – Keystone



**Jeny Jijo**

Assistant Professor, Department of Science and Humanities

Keystone is an *OpenStack service* that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API.



### **Project**

In Keystone, a Project is an abstraction used by other OpenStack services to group and isolate resources (e.g., servers, images, etc.). The most fundamental purpose of Keystone is to be the registry of Projects and to be able to articulate who should have access to those Projects. Projects themselves don't own Users, but Users or User Groups are given access to a Project using the concept of Role Assignments. Having a Role assigned to a User or User Group on a Project denotes that the User or User Group has some kind of access to resources in the Project, and the specific Role selected determines the type of access and capabilities the User or User Group is entitled to have.

### **Domain**

In order for OpenStack clouds to more cleanly support multiple user organizations at the same time, Keystone added a new abstraction, called a *Domain*, that could provide the ability to *isolate the visibility of a set of Projects and Users (and User Groups) to a specific organization*. A Domain is formally defined as a collection of users, groups, and projects. Domains enable you to divide the resources in your cloud into silos that are for use by specific organizations. A domain can serve as a logical division between different portions of an enterprise, or each domain can represent completely separate enterprises.

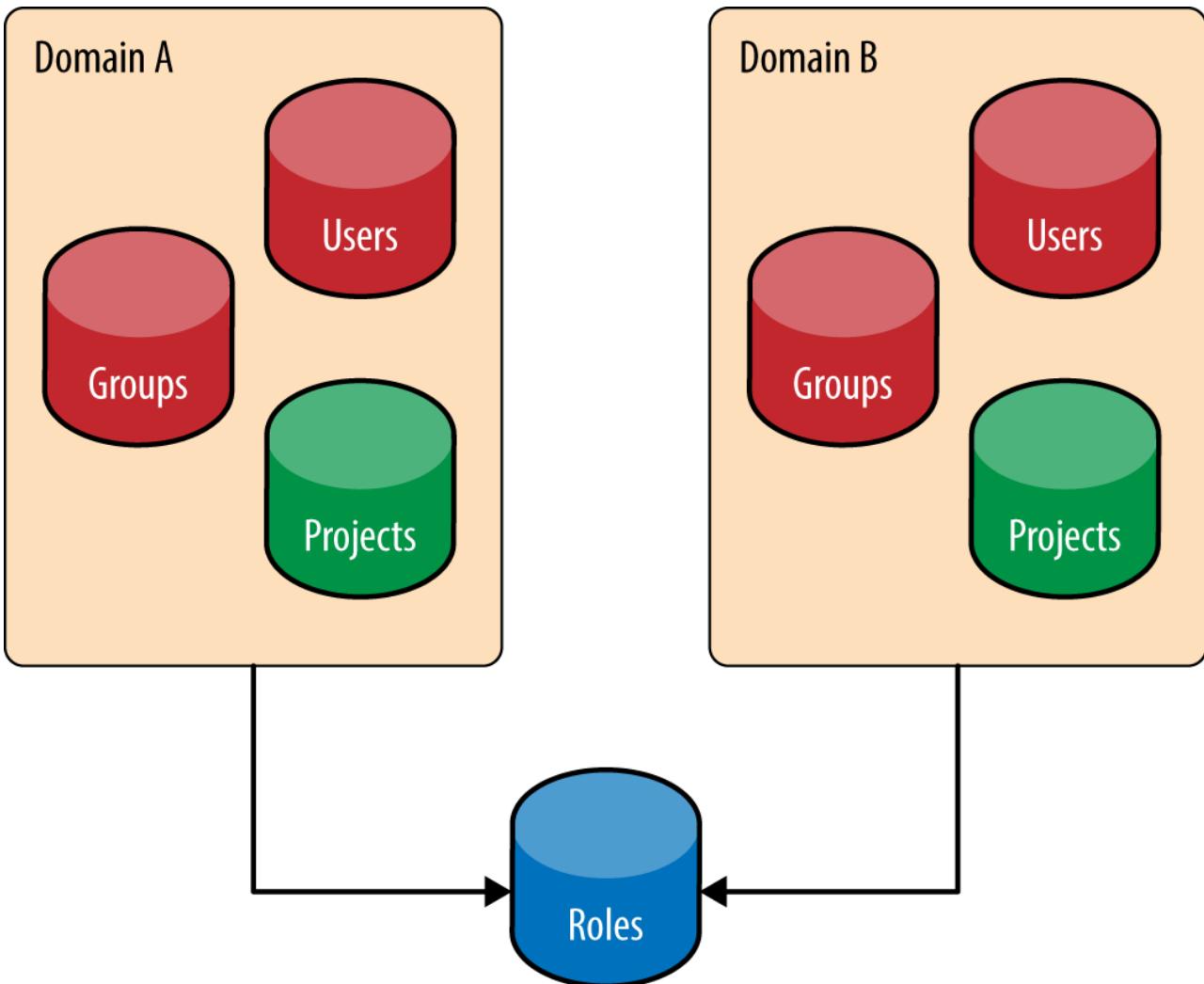
### *Users and User Groups(Actors)*

In the Keystone realm, Users and User Groups are the entities given access to resources that are isolated in Domains and Projects. Groups are a collection of Users. Users are individuals who will end up using your cloud. We refer to Users and Groups as *Actors* since, when assigning a role, these are the entities to which the role is “assigned to.”

# CLOUD COMPUTING

## Keystone Concepts

The relationship between *domains, projects, users, and groups*



*Domains are a collection of Users, Groups, and Projects. Roles are globally unique. Users may have membership in many Groups.*

### **Roles**

*Roles* are used in Keystone to convey a sense of Authorization. An actor may have numerous roles on a target. For example, the role of admin is “assigned to” the user “bob” and it is “assigned on” the project “development.”

### **Assignment**

A role *assignment* is a ternary (or triple): the combination of an actor, a target, and a role. Role assignments are granted and revoked, and may be inherited between groups and users and domains and projects.

### **Targets**

Projects and Domains are very similar in that both are entities where the role is “assigned on.” In other words, a User or User Group is given access to either a Project or Domain by assigning a particular Role value for that User or User Group for a specific Project or Domain. Because Projects and Domains have such similar characteristics, when we need to refer to both entities collectively we refer to them as *Targets*.

### □ **Tokens**

In order for a user to call any OpenStack API they need to prove who they are, and that they should be allowed to call the API in question. The way they achieve that is by passing an OpenStack token into the API call—and Keystone is the OpenStack service responsible for generating these tokens. A user receives this token upon successful authentication against Keystone. The token also carries with it authorization. It contains the authorization a user has on the cloud. A token has both an ID and a payload. The ID of a token is guaranteed to be unique per cloud, and the payload contains data about the user.

```
"token": {  
    "issued_at": "201406-10T20:55:16.806027Z",  
    "expires_at": "2014-06-10T2:55:16.806001Z",  
    "roles": [{  
        "id": "c703057be878458588961ce9a0ce686b",  
        "name": "admin"}  
    ],
```

the token payload contains information about when it was created, when it will expire, which user authenticated—and thus is allowed to use the token—which project the token is valid on etc..

```
{  
  "serviceCatalog": [  
    {  
      "endpoints": [  
        {  
          "adminURL": "http://swift.admin-nets.local: 8080/",  
          "region": "RegionOne",  
          "internalURL": "http://127.0.0.1: 8080/v1/AUTH_1",  
          "publicURL": "http://swift.publicinternets.com/v1/AUTH_1"  
        }  
      ],  
      "type": "object-store",  
      "name": "swift"  
    },  
    {  
      "adminURL": "http://identity-admin-nets.local: 5000/v3",  
      "region": "RegionOne",  
      "internalURL": "http://127.0.0.1: 5000/v3",  
      "publicURL": "http://identity-publicinternets.com/v3",  
      "type": "identity",  
      "name": "keystone"  
    }  
  ]  
}
```

### □ *Catalog*

The service catalog is essential for an OpenStack cloud. It contains the URLs and endpoints of the different Cloud services. Without the catalog, users and applications would not know where to route requests to create VMs or store objects. The service catalog is broken up into a list of endpoints, and each endpoint is broken down into an admin URL, internal URL, and public URL, which may be the same.

The **Identity Service in Keystone** provides the Actors. Identities in the Cloud may come from various locations, including but not limited to *SQL, LDAP, and Federated Identity Providers*.

### SQL

Keystone includes the option to store your actors (Users and Groups) in SQL; supported databases include MySQL, PostgreSQL, and DB2. Keystone will store information such as name, password, and description. The settings for the database must be specified in Keystone's configuration file. Essentially, Keystone is acting as an Identity Provider, which may not be the best case for everyone, and certainly not the best case for enterprise customers.

### Pros:

- Easy to set up
- Management of users and groups through OpenStack APIs

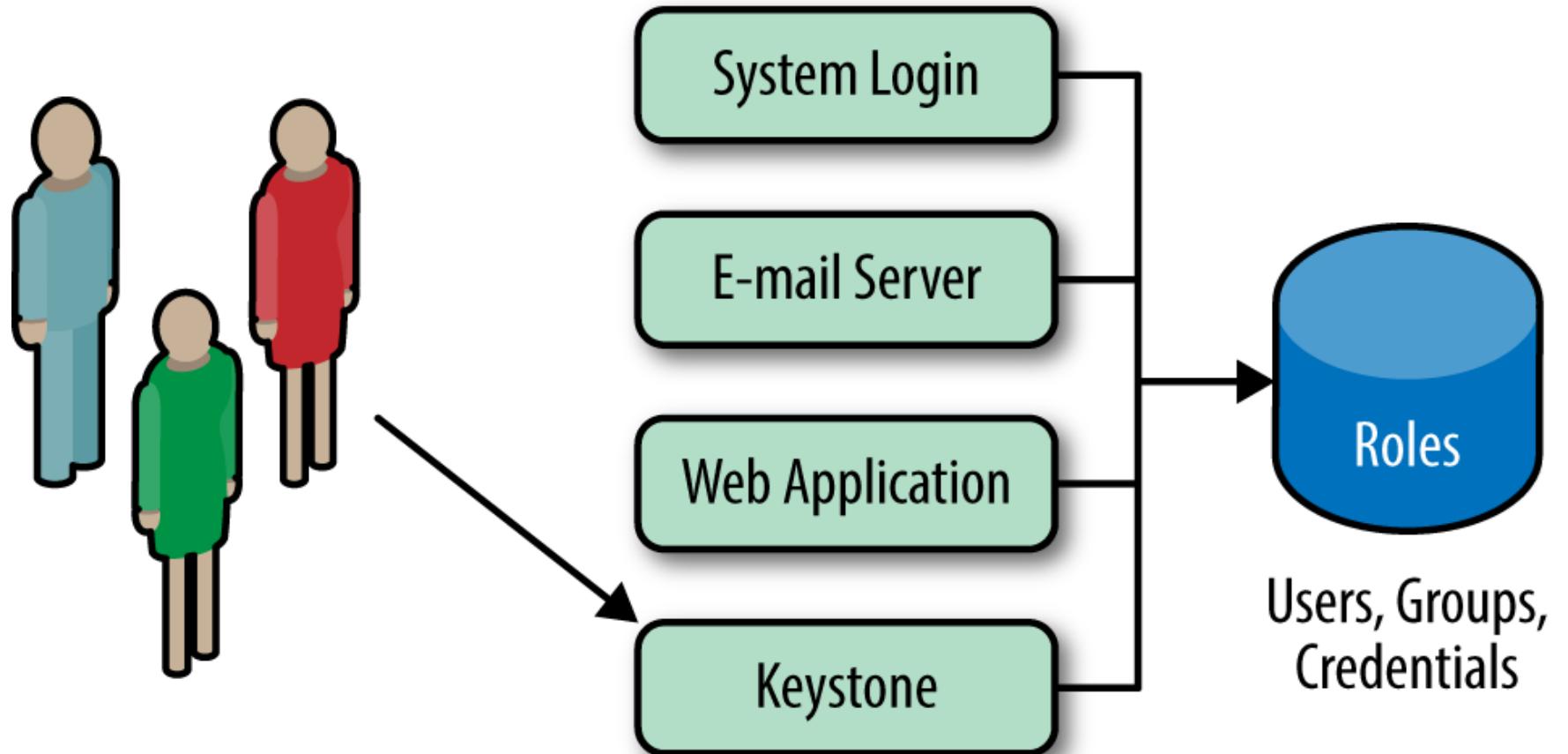
### Cons:

- Keystone should not be an Identity Provider
- Weak password support
  - ✓ No password rotation
  - ✓ No password recovery
- Most enterprises have an LDAP server they want to use
- Identity silo: yet another username and password users must remember

### □ LDAP

Keystone also has the option to retrieve and store your actors (Users and Groups) in **Lightweight Directory Access Protocol (LDAP)**. Keystone will access the LDAP just like any other application that uses the LDAP (System Login, Email, Web Application, etc.). The settings for connecting to the LDAP are specified in Keystone's configuration file.

Ideally, the LDAP should perform only read operations, such as user and group lookup (via search) and authentication (via bind). If using LDAP as a read-only Identity backend, Keystone should need a minimal amount of privilege to use the LDAP. For instance, it needs read access to the user and group attributes defined in `keystone.conf`, an unprivileged account (anonymous access is preferred), and it does *not* require access to password hashes.



*Keystone should use an internal LDAP just like any other application*

### Pros:

- No longer need to maintain copies of user accounts.
- Keystone does not act as an Identity Provider.

### Cons:

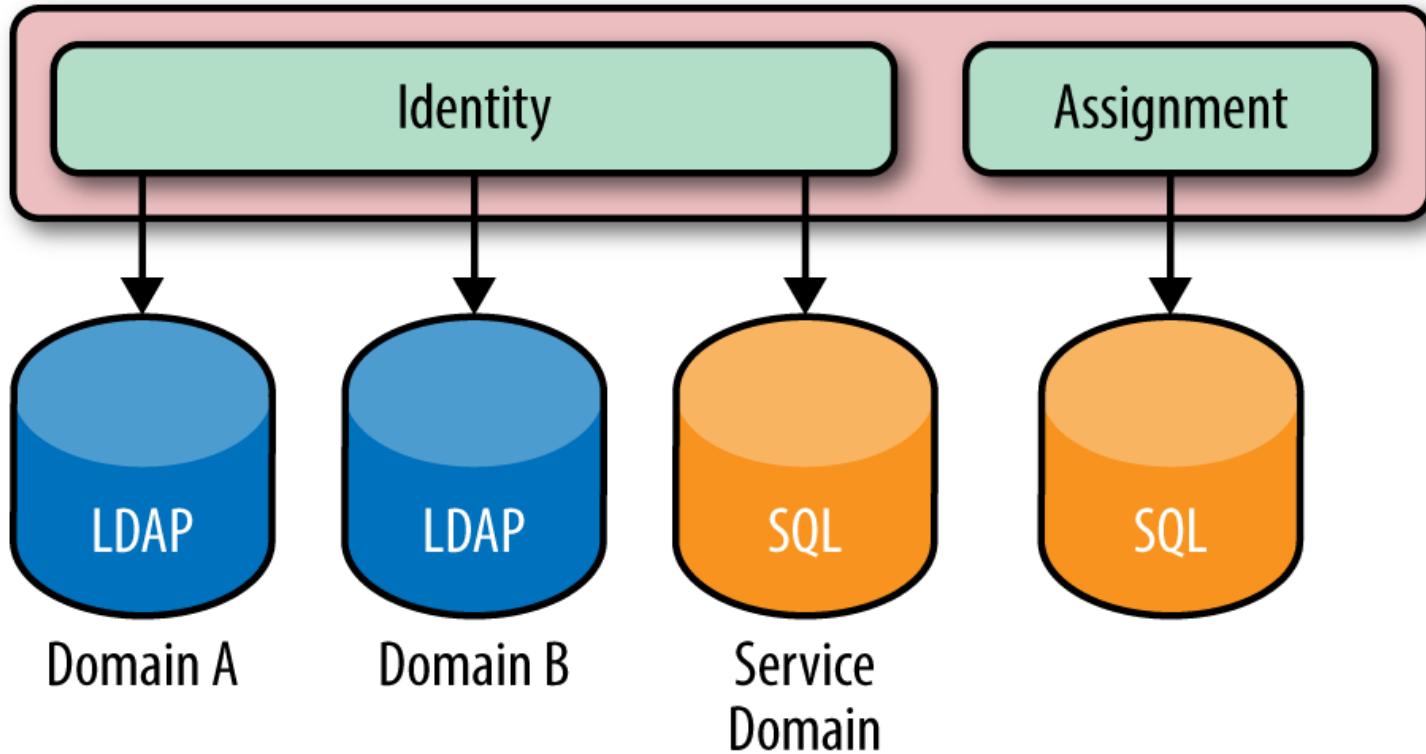
- Service accounts still need to be stored somewhere, and the LDAP admin may not want these accounts in LDAP.
- Keystone is still “seeing” user passwords, since the passwords are in the authentication request. Keystone simply forwards these requests, but ideally Keystone does not want to see a user’s password, ever!

## Identity- Multiple Backends

---

### **Multiple Backends**

The impact of Keystone supporting Multiple Backends is that a deployment may have one identity source (backend) per Keystone domain. The motivation for supporting multiple Identity backends is that in an enterprise setting, LDAP administrators may not be the same organization as the OpenStack deployment team, so creating service accounts in LDAP is highly unlikely. **LDAP is typically restricted to be used only for employee information.** Another benefit of a logical split between Identity backends and domains is that now multiple LDAPS can be used. So, in the case of a company merger or different departments having different LDAPS, the same enterprise can still be represented.



*The Identity service may have multiple backends per domain. LDAPs for both Domains A and B, and an SQL-based backend for the service accounts is usual. The Assignment service is also shown to remind readers that the assignment, resource, and identity service may all be backed by various stores.*

## Identity- Multiple Backends

---

### Pros:

- Able to simultaneously support multiple LDAPs for various user accounts and SQL backends for service accounts
- Leverage existing identity LDAP while not impacting it

### Cons:

- Slightly more complex to set up
- Authentication for user accounts must be domain scoped

## Identity- Identity Providers

---

### Identity Providers

Keystone is able to consume federated authentication via Apache modules for multiple trusted Identity Providers. These users are not stored in Keystone, and are treated as ephemeral. The federated users will have their attributes mapped into group-based role assignments. From a Keystone perspective, **an identity provider is a source for identities**; it may refer to software that is backed by various backends (LDAP, AD, MongoDB) or Social Logins (Google, Facebook, Twitter). Essentially, it is software that abstracts out the backend and translates user attributes to a standard federated identity protocol format (SAML, OpenID Connect).

## Identity- Identity Providers

---

### Pros:

- Able to leverage existing infrastructure and software to authenticate users and retrieve information about users
- Further separation between Keystone and handling identity information
- Opens the door for new possibilities in the federation realm, such as single sign-on and hybrid cloud
- Keystone does not see any user passwords
- Identity provider handles authentication completely, so whether it is password, certificate, or two-factor based is irrelevant to Keystone

### Cons:

- Most complex set up of the identity sources

There are various ways to authenticate with the Keystone service—by far the two most common are by *supplying a password or by using a token.*

### **Password**

The most common way for a user or service to authenticate is to supply a password.

Eg –

The payload shown below is a sample POST request to Keystone. It is helpful to show the entire payload so the reader recognizes the information that is necessary to authenticate.

# CLOUD COMPUTING

## Authentication

```
{  
  "auth": {  
    "identity": {  
      "methods": [  
        "password"  
      ],  
      "password": {  
        "user": {  
          "domain": {  
            "name": "example.com"  
          },  
          "name": "Joe",  
          "password": "secretsecret"  
        }  
      }  
    }  
  },  
  "scope": {  
    "project": {  
      "domain": {  
        "name": "example.com"  
      },  
      "name": "project-x"  
    }  
  }  
}
```

*An example of an authentication payload request that contains a scope.*

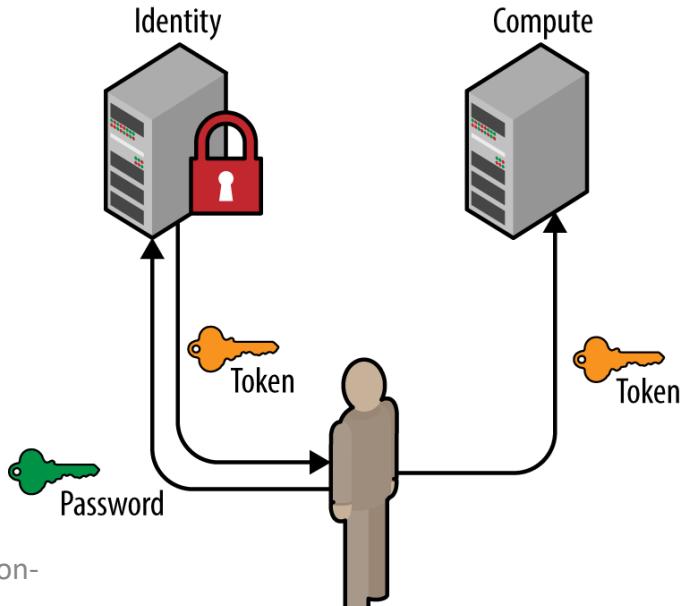
The *payload of the request* must contain enough information to find where the user exists, authenticate the user, and optionally, retrieve a service catalog based on the user's permissions on a scope (project).

The *user section* that identifies the incoming user should have domain information (either the domain name or ID), unless a user's globally unique ID is used, in which case that is sufficient to identify the user. This is because in a multi-domain deployment, there may be multiple users with the same name, so proper scoping is necessary to determine which user is authenticating.

# CLOUD COMPUTING

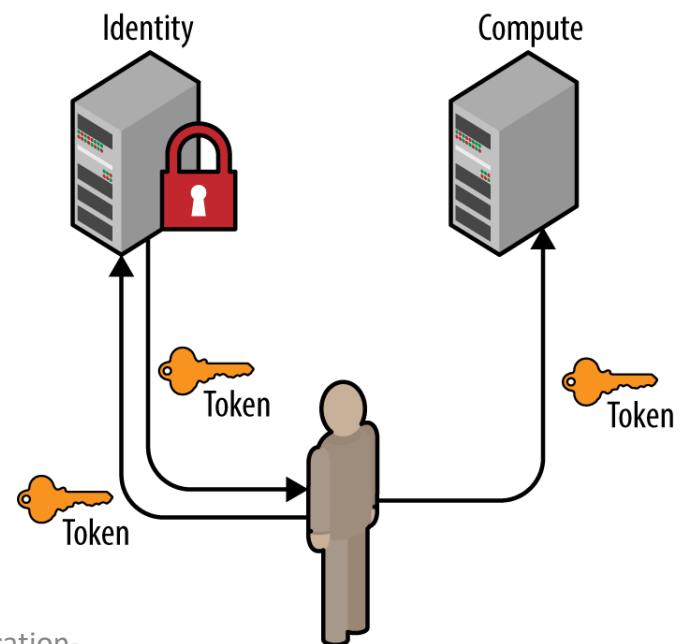
## Authentication

The *scope section* is optional but is often used, since without a scope a user will not be able to retrieve a service catalog. The scope is used to indicate which project the user wishes to work against. If a user does not have a role on that project, then the request will be rejected. Similar to the user section, the scope section must have enough information about the project to find it, so the owning domain must be specified.



Similar to password , a user may also request a new token by providing a current token. The payload of this POST request is significantly less code than its password counterpart. There are many reasons why a token will be used to retrieve another, such as refreshing a token that will soon expire or to change an unscoped token to a scoped token.

```
{  
  "auth": {  
    "identity": {  
      "methods": [  
        "token"  
      ],  
      "token": {  
        "id": "e80b74"  
      }  
    }  
  }  
}
```



## Access Management and Authorization

---

Managing access and authorizing what APIs a user can use is one of the key reasons Keystone is essential to OpenStack. Keystone's approach to this problem is to create a **Role-Based Access Control (RBAC)** policy that is enforced on each public API endpoint. These policies are stored in a file on disk, commonly named policy.json.

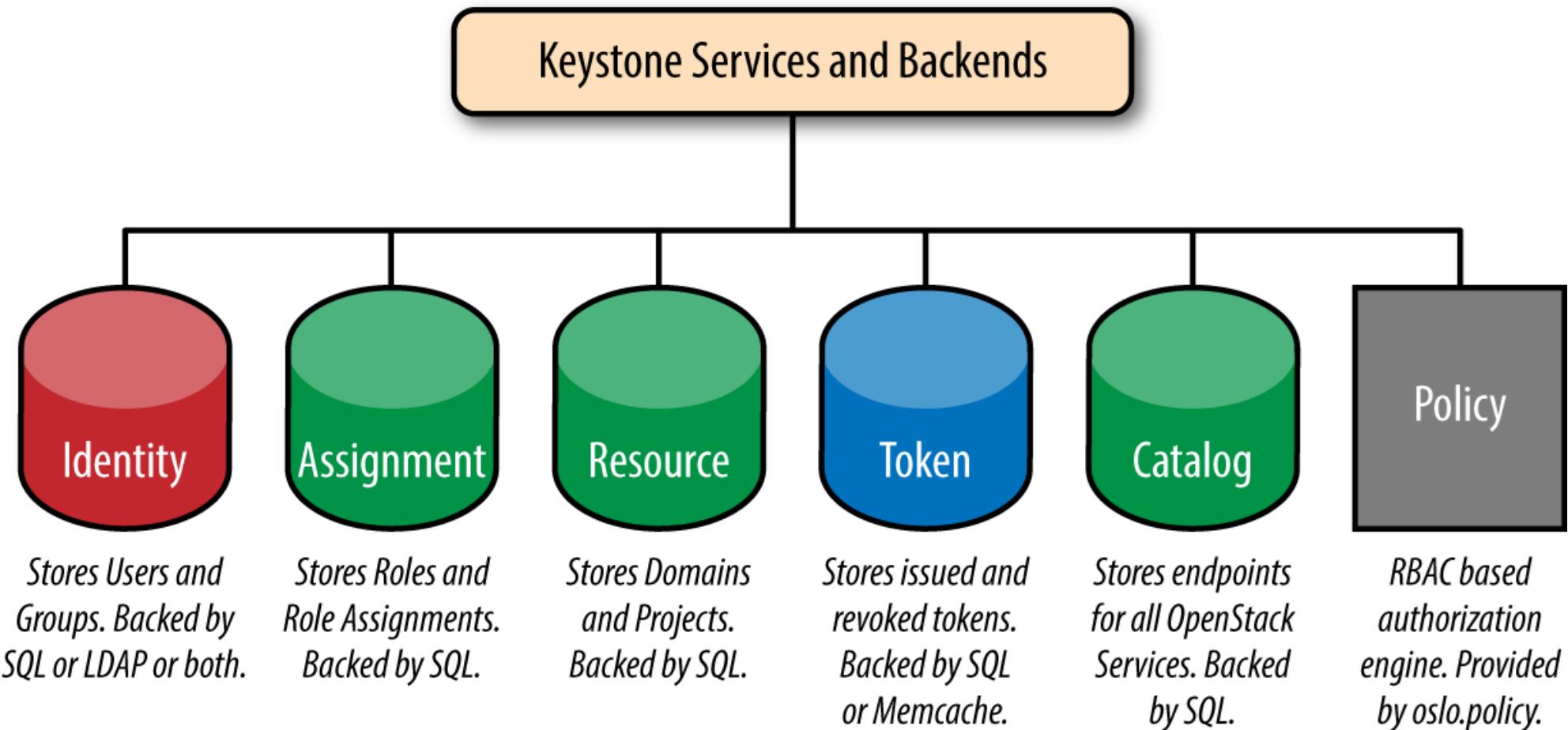
```
{  
    "admin_required": "role:admin or is_admin:1",  
    "owner" : "user_id:%(user_id)s",  
    "admin_or_owner": "rule:admin_required or rule:owner",  
  
    "identity:list_projects": "rule:admin_required",  
    "identity:create_project": "rule:admin_required",  
    "identity:delete_project": "rule:admin_required",  
  
    "identity:list_user_projects": "rule:admin_or_owner"  
}
```

Snippet of Keystone's  
policy.json file

Targets refer to the left-hand key, and rules refer to the right-hand value. At the top of the file, targets are established that can be used for evaluation of other targets. It is here that we define what it means to be an admin, an owner, or either.

# CLOUD COMPUTING

## Backends and Services





THANK YOU

---

Jeny Jijo  
[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)



## CLOUD COMPUTING

### Cloud Threats –Dos & EDoS

---

**Jeny Jijo**

Department of Computer Science & Engineering

# CLOUD COMPUTING

---

## Cloud Threats –Dos & EDoS

**Jeny Jijo**

Assistant Professor, Department of Computer Science & Engineering

Trust and security become even more demanding for web and cloud services, because leaving user applications completely to the cloud providers has faced strong resistance by most PC and server users. Cloud platforms become worrisome to some users for lack of privacy protection, security assurance, and copyright protection.

### What is Cloud Security?

Cloud security is a set of control-based safeguards and technology protection designed to protect resources stored online from leakage, theft, or data loss. Protection encompasses cloud infrastructure, applications, and data from threats.

### Securing Cloud Data

#### Data Loss

*BYOD*



*External threats*



*Internal threats*



#### Data Privacy

*Regulatory Compliance*



*Data Sovereignty*



*Data Remanence*

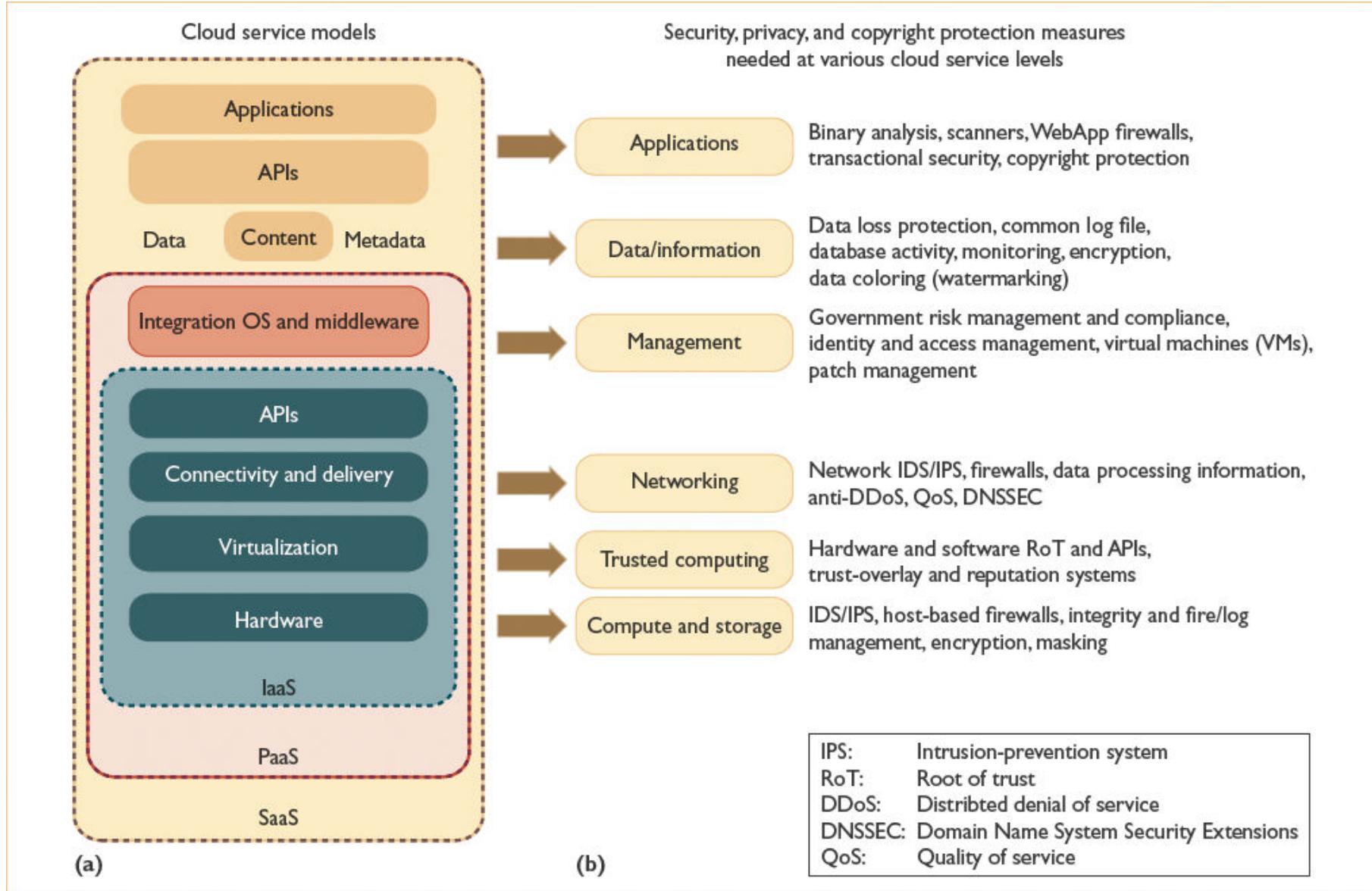


### Cloud Security Defense Strategies

A healthy cloud ecosystem is desired to free users from abuses, violence, cheating, hacking, viruses, rumours, pornography, spam, and privacy and copyright violations. The security demands of three cloud service models, IaaS, PaaS, and SaaS are based on various SLAs between providers and users.

#### i) *Basic Cloud Security*

Three basic cloud security enforcements are expected. First, **facility security** in data centers demands on-site security year round. Biometric readers, CCTV (close-circuit TV), motion detection, and man traps are often deployed. Also, **network security** demands fault-tolerant external firewalls, intrusion detection systems (IDSes), and third-party vulnerability assessment. Finally, **platform security** demands SSL and data decryption, strict password policies, and system trust certification.



A security-aware cloud architecture demands security enforcement such as

- *Protection of servers from malicious software attacks such as worms, viruses, and malware*
- *Protection of hypervisors or VM monitors from software-based attacks and vulnerabilities*
- *Protection of VMs and monitors from service disruption and DoS attacks*
- *Protection of data and information from theft, corruption, and natural disasters*
- *Providing authenticated and authorized access to critical data and services*

### *ii) Security Challenges in VMs*

In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle attack for VM migrations. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers.

Program shepherding can be applied to control and verify code execution. Other defense technologies include using the RIO dynamic optimization infrastructure, or VMware's vSafe and vShield tools, security compliance for hypervisors, and Intel vPro technology.

### *iii) Cloud Defense Methods*

Virtualization enhances cloud security. But VMs add an additional layer of software that could become a single point of failure. Virtualization provides each VM with better **security isolation** and each partition is protected from DoS attacks by other partitions. Security attacks in one VM are isolated and contained from affecting the other VMs. Trust negotiation is often done at the SLA level. Public Key Infrastructure (PKI) services could be augmented with data-center reputation systems. Worm and DDoS attacks must be contained.

### *iv) Defense with Virtualization*

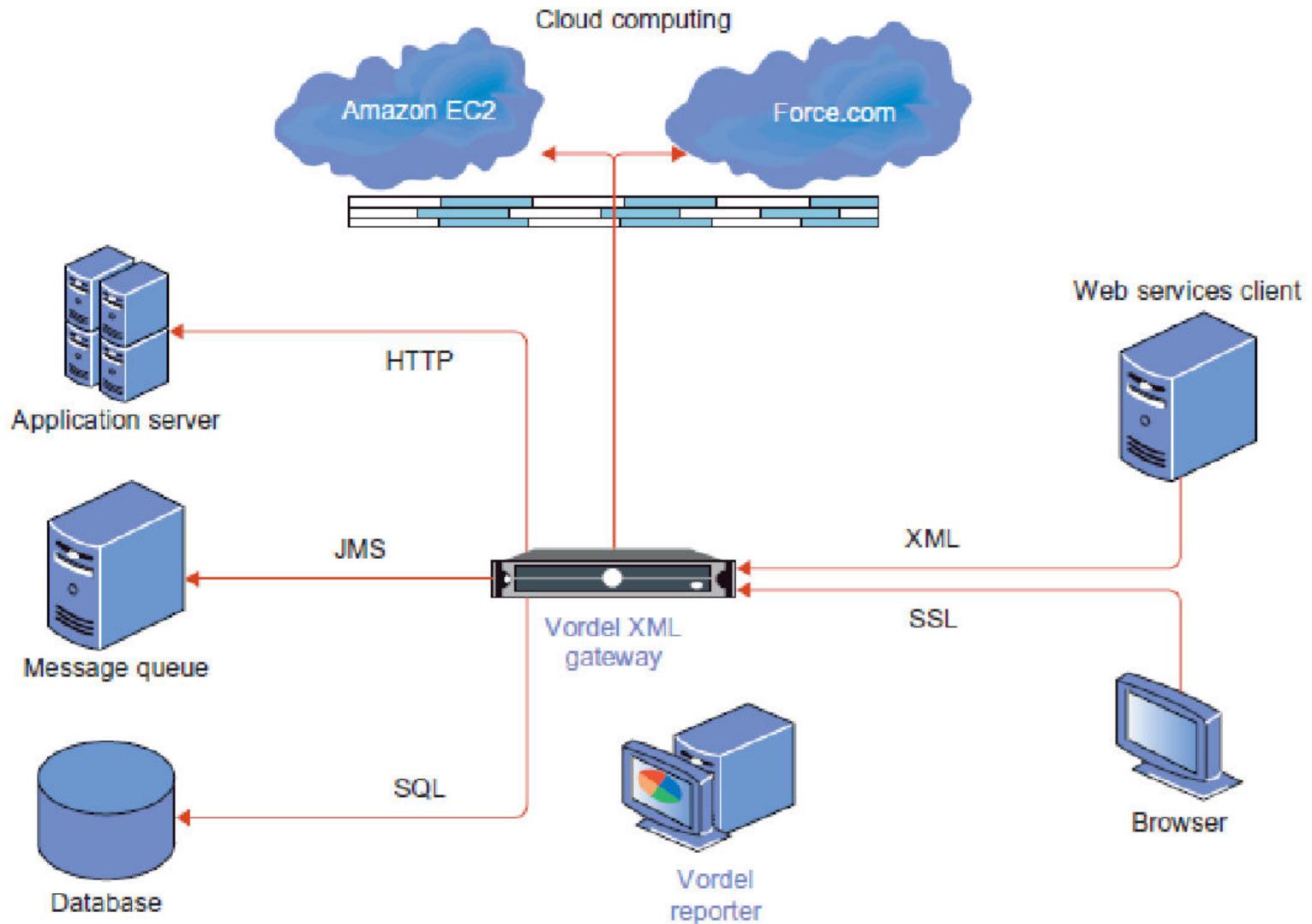
The VM is decoupled from the physical hardware and can be represented as a software component regarded as binary or digital data. The VM can be saved, cloned, encrypted, moved, or restored with ease. **VMs enable High Availability and faster disaster recovery.** Live migration of VMs was suggested for building distributed intrusion detection systems (DIDSes). Multiple IDS VMs can be deployed at various resource sites including data centers. DIDS design demands trust negation among PKI domains. Security policy conflicts must be resolved at design time and updated periodically.

### v) *Privacy and Copyright Protection*

With shared files and data sets, privacy, security, and copyright data could be compromised in a cloud computing environment. *Several security features desired in a secure cloud are :*

- Dynamic web services with full support from secure web technologies
- Trust between users and providers through SLAs and reputation systems
- Effective user identity management and data-access management
- Single sign-on and single sign-off to reduce security enforcement overhead
- Auditing and copyright compliance through proactive enforcement
- Control of data operation shifting from the client environment to cloud providers
- Protection of sensitive and regulated information in a shared environment

## Example –Security defense system for a private cloud environment



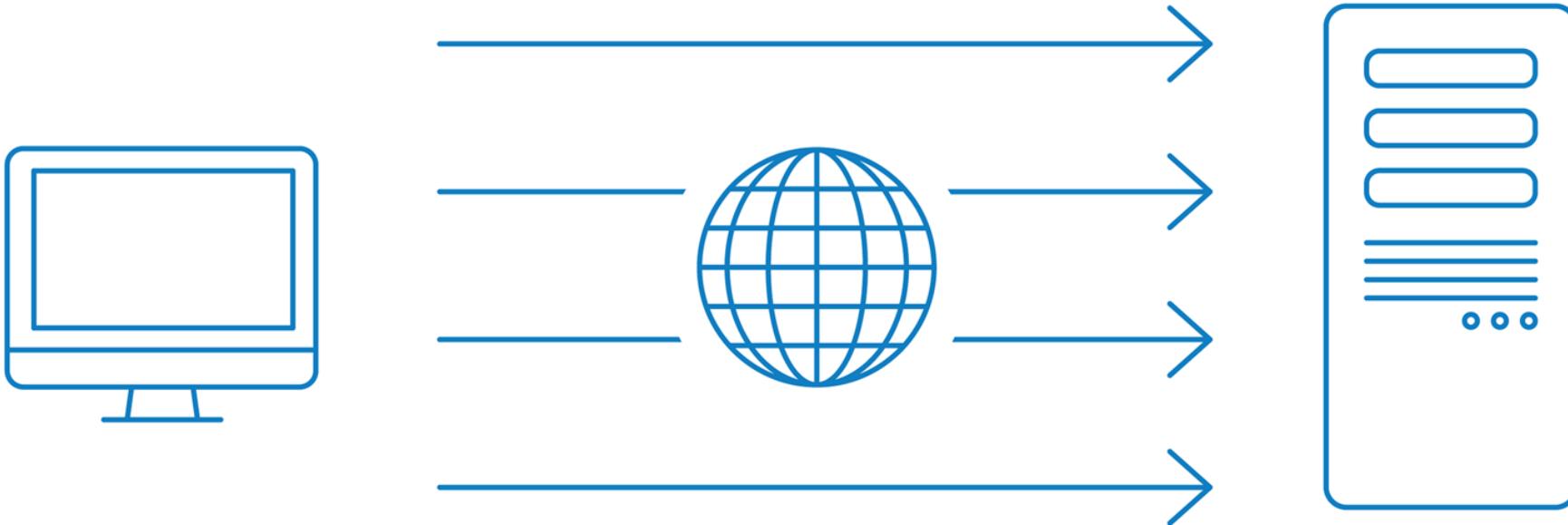
The typical security structure coordinated by a secured gateway plus external firewalls to safeguard the access of public or private clouds.

## Cloud Security-DoS

---

A **denial-of-service (DoS) attack** is a type of cyber attack in which a malicious actor aims to render a computer or other device unavailable to its intended users by interrupting the device's normal functioning. DoS attacks typically function by overwhelming or flooding a targeted machine with requests until normal traffic is unable to be processed, resulting in denial-of-service to additional users. A DoS attack is characterized by using a single computer to launch the attack.

The **primary focus of a DoS attack** is to oversaturate the capacity of a targeted machine, resulting in denial-of-service to additional requests. The multiple attack vectors of DoS attacks can be grouped by their similarities.



<https://developer.okta.com/books/api-security/dos/what/>

DoS attacks typically fall in 2 categories:

### Buffer overflow attacks

An attack type in which a memory buffer overflow can cause a machine to consume all available hard disk space, memory, or CPU time. This form of exploit often results in sluggish behavior, system crashes, or other deleterious server behaviors, resulting in denial-of-service.

### Flood attacks

By saturating a targeted server with an overwhelming amount of packets, a malicious actor is able to oversaturate server capacity, resulting in denial-of-service. In order for most DoS flood attacks to be successful, the malicious actor must have more available bandwidth than the target.

## Cloud Security-DoS

---

While it can be difficult to separate an attack from other network connectivity errors or heavy bandwidth consumption, some characteristics may indicate an attack is underway.

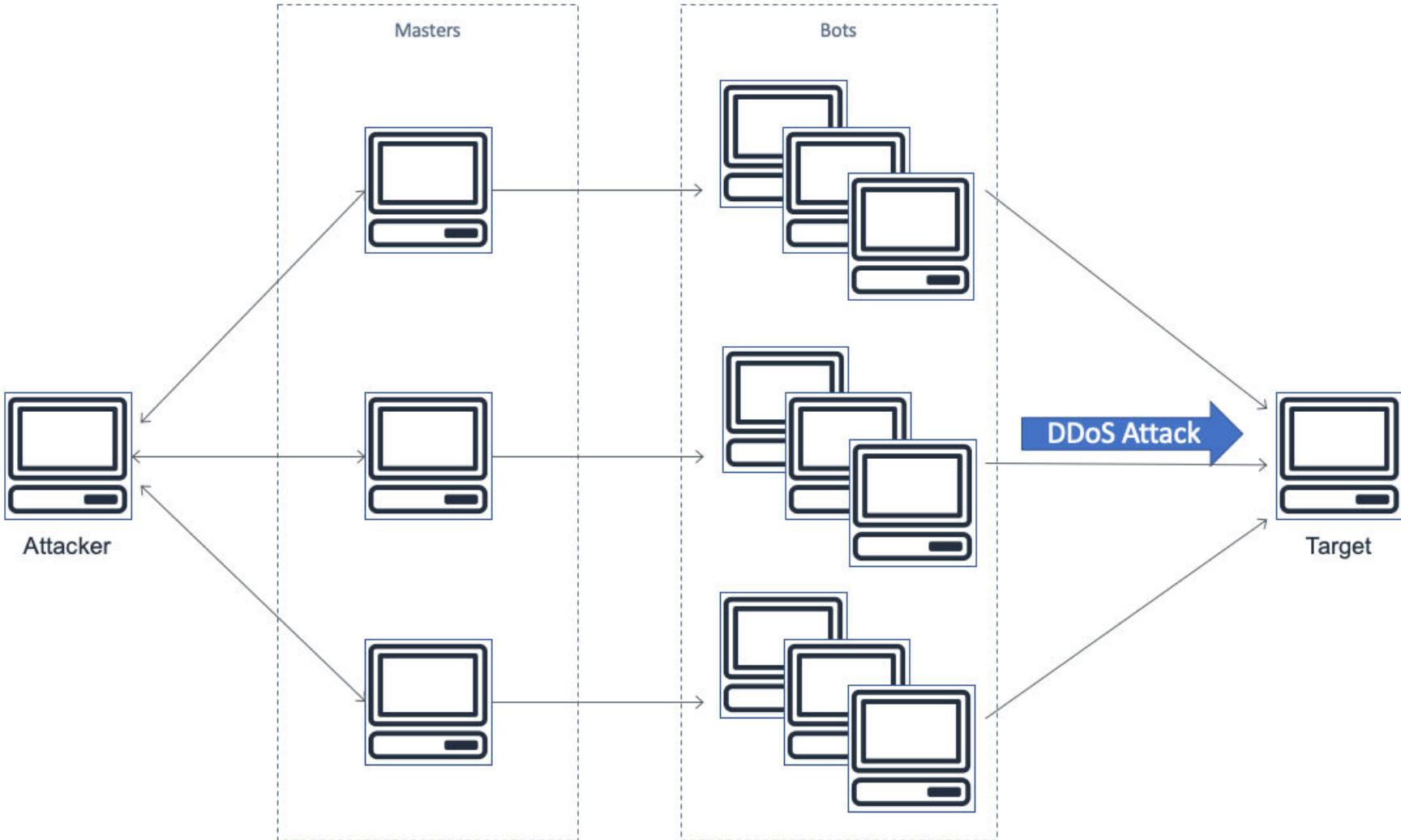
*Indicators of a DoS attack include:*

- ✓ A typically slow network performance such as long load times for files or websites
- ✓ The inability to load a particular website such as your web property
- ✓ A sudden loss of connectivity across devices on the same network

A **distributed denial-of-service (DDoS)** attack is a malicious attempt to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

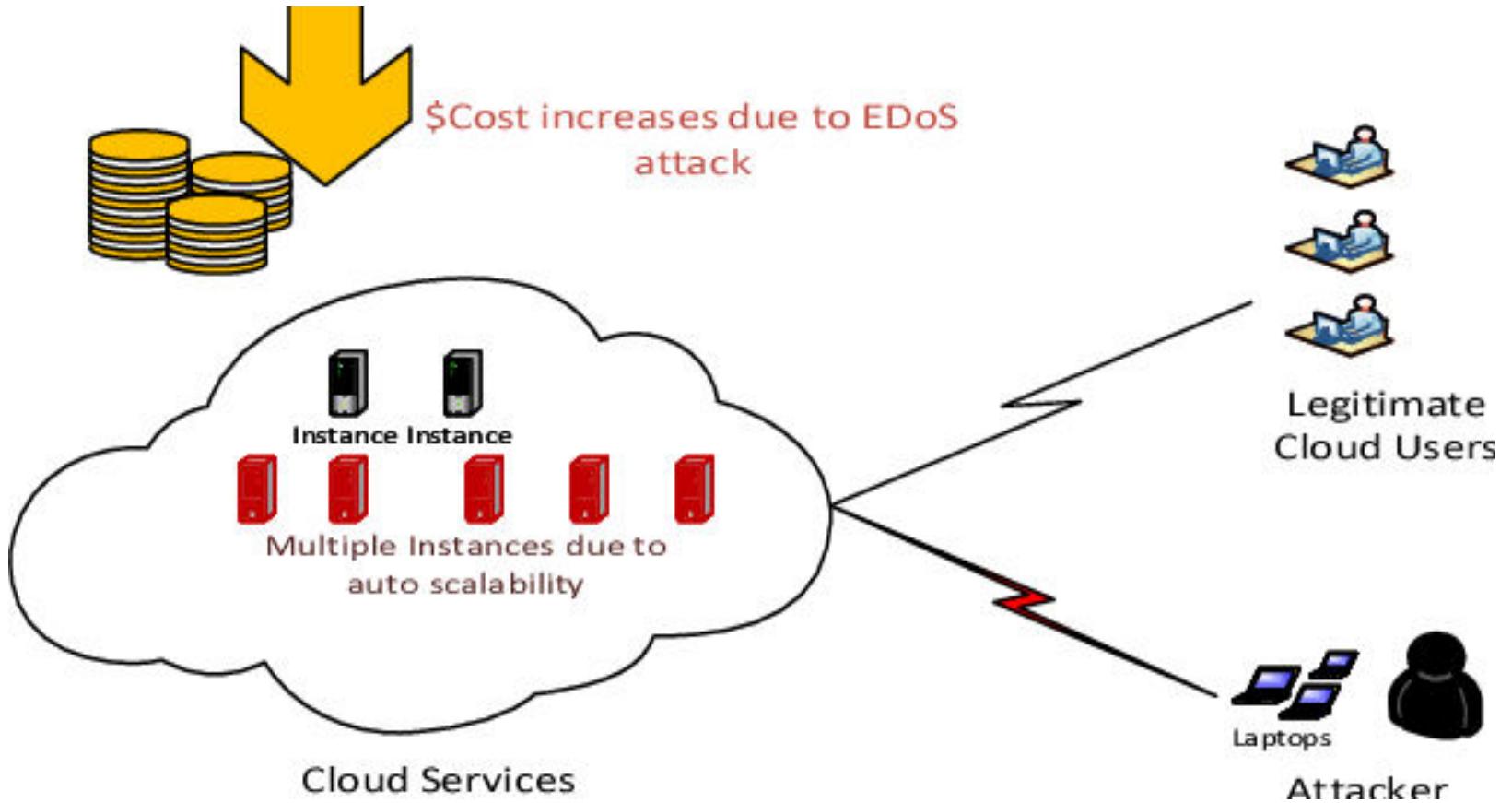
DDoS attacks achieve effectiveness by utilizing multiple compromised computer systems as sources of attack traffic. Exploited machines can include computers and other networked resources such as [IoT devices](#).

From a high level, a DDoS attack is like an unexpected traffic jam clogging up the highway, preventing regular traffic from arriving at its destination.



- DDoS attacks are carried out with networks of Internet-connected machines.
- These networks consist of computers and other devices (such as IoT devices) which have been infected with malware, allowing them to be controlled remotely by an attacker. These individual devices are referred to as bots (or zombies), and a group of bots is called a botnet.
- Once a botnet has been established, the attacker is able to direct an attack by sending remote instructions to each bot.
- When a victim's server or network is targeted by the botnet, each bot sends requests to the target's IP address, potentially causing the server or network to become overwhelmed, resulting in a denial-of-service to normal traffic.
- Because each bot is a legitimate Internet device, separating the attack traffic from normal traffic can be difficult.

- In EDoS attack, the attacker brilliantly and deliberately moves to cloud computing auto scaling feature. This leads to an increase in utilization of computing resource of a destination user and results in huge resource costs as well as the security price provided to the destination cloud user. This constant effort towards the utilization of cloud computing services by the target cloud user ends with a huge financial loss to the consumer.
- **DDoS** attack focuses to **cease all the services provided by the service provider**. Thus, in DDoS attack, the attacker interrupts maximum server resources in a short interval whereas **EDoS** attack is typically more intelligent and effective. It targets an individual by gradually push the illegal data towards the genuine user for a longer duration.



- EDoS concentrates on **maximizing the financial costs of cloud-based consumers through cloud-based resources**. The EDoS's traffic domain is somewhat different than the DDoS attack domain. The EDoS attack affects cloud computations in terms of performance as well as in terms of price model. EDoS may also lead to hike in cost of energy bill by consuming fraudulent resources

## Cloud Security-DDoS vs EDoS

DDoS attack	EDoS attack
Degrade/block cloud services	Makes cloud resource economically unsustainable
The attacker period is very short	The attacker period is long
The attacker occurs above the EDoS attack zone	The attacks appear between normal data traffic and DDoS attack region

Overall Objective of Cloud Security:

**Transparency**

- Secure cloud infrastructure- Physical & Virtual
- Delivering Secure Cloud Services
- Providers should implement current & future cloud **Standards & Certifications**
- Automation of auditing & security

# Transparency



# Confidence

[https://en.wikipedia.org/wiki/Denial-of-service\\_attack](https://en.wikipedia.org/wiki/Denial-of-service_attack)

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.4908&rep=rep1&type=pdf>

<https://www.youtube.com/watch?v=mJ8Z30unbo0>



THANK YOU

---

Jeny Jijo  
[jenyjijo@pes.edu](mailto:jenyjijo@pes.edu)