CLOUD COMPUTING

Introduction to Storage

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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

Storage

Storage : An Introduction

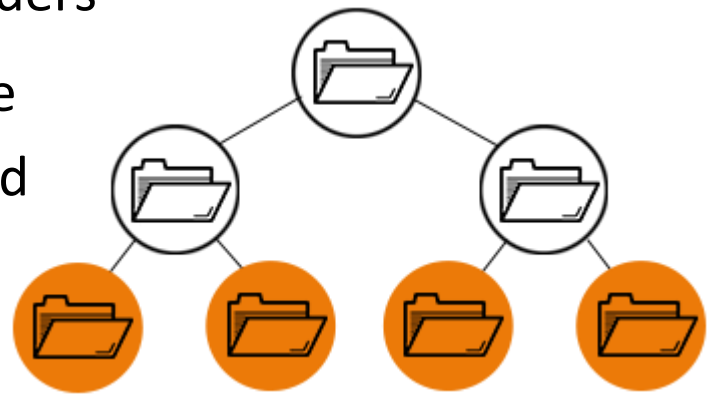
CLOUD COMPUTING

Storage : Introduction

- Data : A collection of raw facts
- Types of Data :
  - Structured Data (Organized in rows and Columns and typically stored using say DBMS)
  - Unstructured Data (Cannot be organized and stored in rows and columns and difficult to uniquely id and retrieve and typically stored in object stores)
- Data explosion in the last decade is leading to a prediction of 572 Zeta bytes (10^{21} bytes) by 2030 and expected to 50,000 Zeta bytes by 2050
- Storage systems (and we are looking at external storage systems) will need to support the same.
- These storage systems are characterized by their
  - Cost
  - Speed (access time) or performance
  - Reliability
  - Availability
  - Scalability
  - Management

Storage : Understanding Data Storage

- There are different types of storage based on the format in which the data is held, organized and presented. These are
- **File Storage :** Organizes and represents data as a hierarchy of files in folders
  - Data is stored as a single piece of information inside a folder, just like you'd organize pieces of paper inside a manila folder. When you need to access that piece of data, your computer needs to know the path to find it

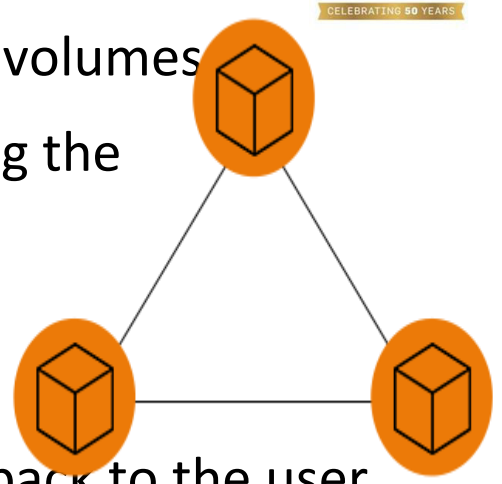


  - Data stored in files is organized and retrieved using a limited amount of metadata that tells the computer exactly where the file itself is kept.
  - Network Attached Storage (NAS) or the Direct Attached Storage (DAS) are examples of the same.
  - Scaling with file storage is typically scale out as there is a limit to addition of capacity to a system

https://www.redhat.com/en/topics/data-storage/file-block-object-storage

CLOUD COMPUTING

Storage : Understanding Data Storage (2)

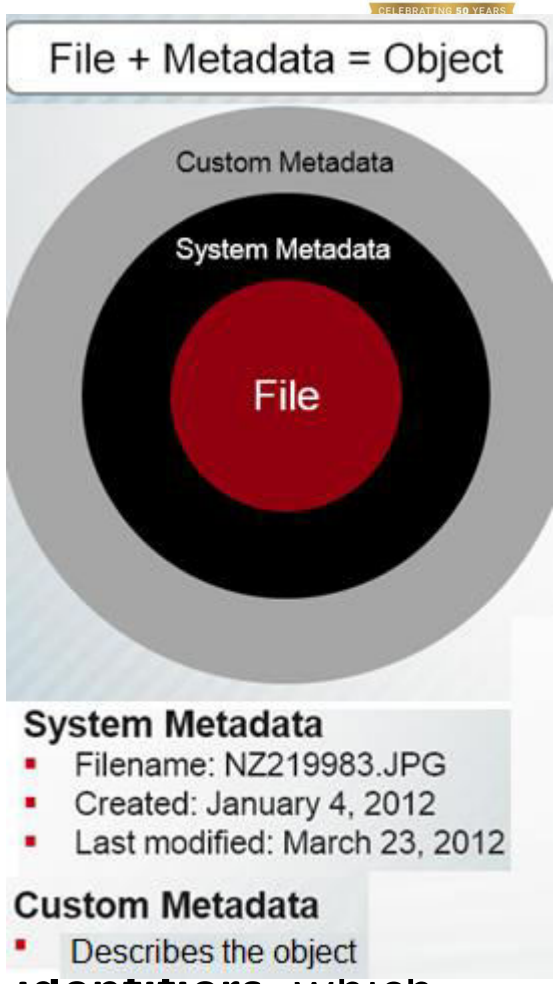


- **Block Storage :** block storage chunks data into arbitrarily organized, evenly sized volumes
  - Data is chopped into blocks with each block given an unique identifier allowing the storage system to place the data where-ever convenient
  - Block storage is often configured to decouple the data from the user's environment and when data is requested, the underlying storage software reassembles the blocks of data from these environments and presents them back to the user
  - Usually used with SAN environments
  - It can be retrieved quickly and can be accessed by any environment
  - It's an efficient and reliable way to store data and is easy to use and manage.
  - It works well with enterprises performing big transactions and those that deploy huge databases, meaning the more data you need to store, the better off you'll be with block storage
  - Block storage can be expensive and it has limited capability to handle metadata, which means it needs to be dealt with in the application or database level

CLOUD COMPUTING

Storage : Understanding Data Storage (3)

- **Object Storage :** manages data and links it to associated metadata
  - Object storage, **also known as object-based storage**, is a flat structure with the data broken into discrete units called objects **and is kept in a single repository** (instead of being kept as files in folders or as blocks)
  - Object storage **volumes work as modular units:**
  - Each is a **self-contained repository that owns the data, a unique identifier that allows the object to be found over a distributed system**, and the metadata that describes the data.
  - Metadata is important and includes details at two levels like age, privacies/securities, and access contingencies etc. and also have custom information about the object (data) itself information.
  - **To retrieve the data, the storage operating system uses the metadata and identifiers**, which distributes the load better and lets administrators apply policies that perform more robust searches.

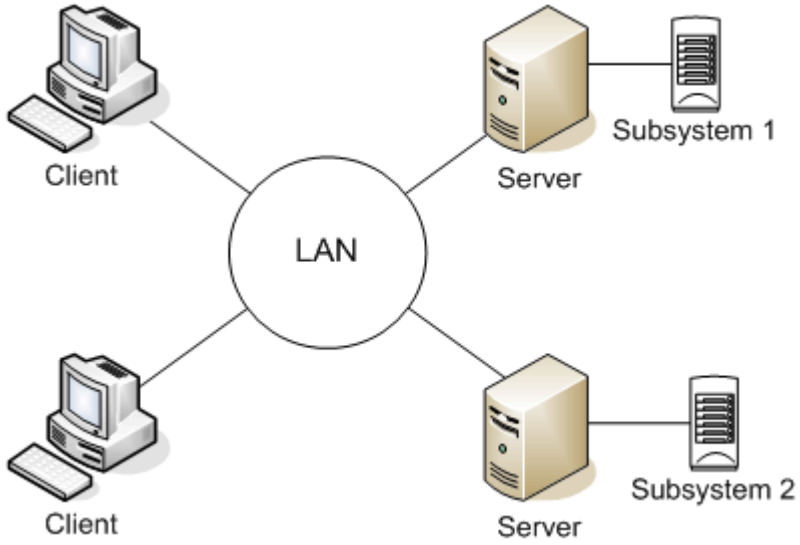


CLOUD COMPUTING

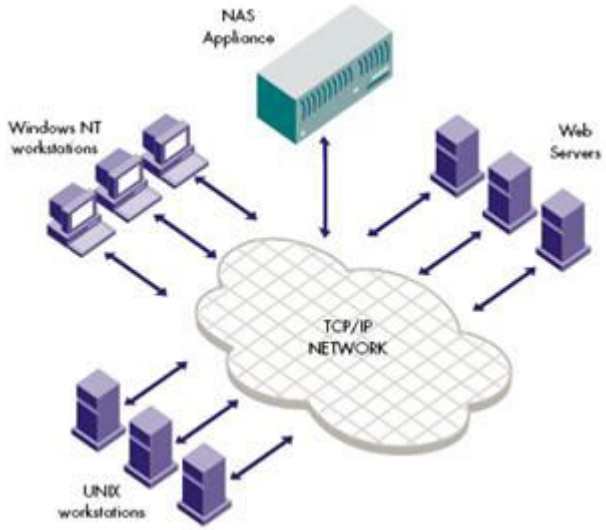
Storage Architectures to support these

Different ways of connecting storage devices and Servers in an network leads to different storage architecture

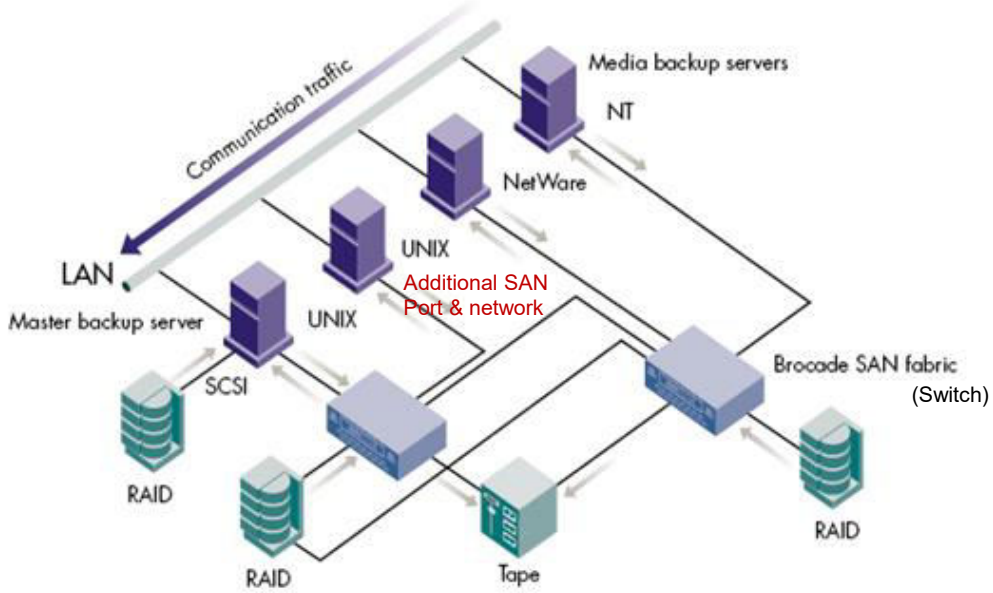
Directly Attached Storage (DAS)



Network Attached Storage (NAS)

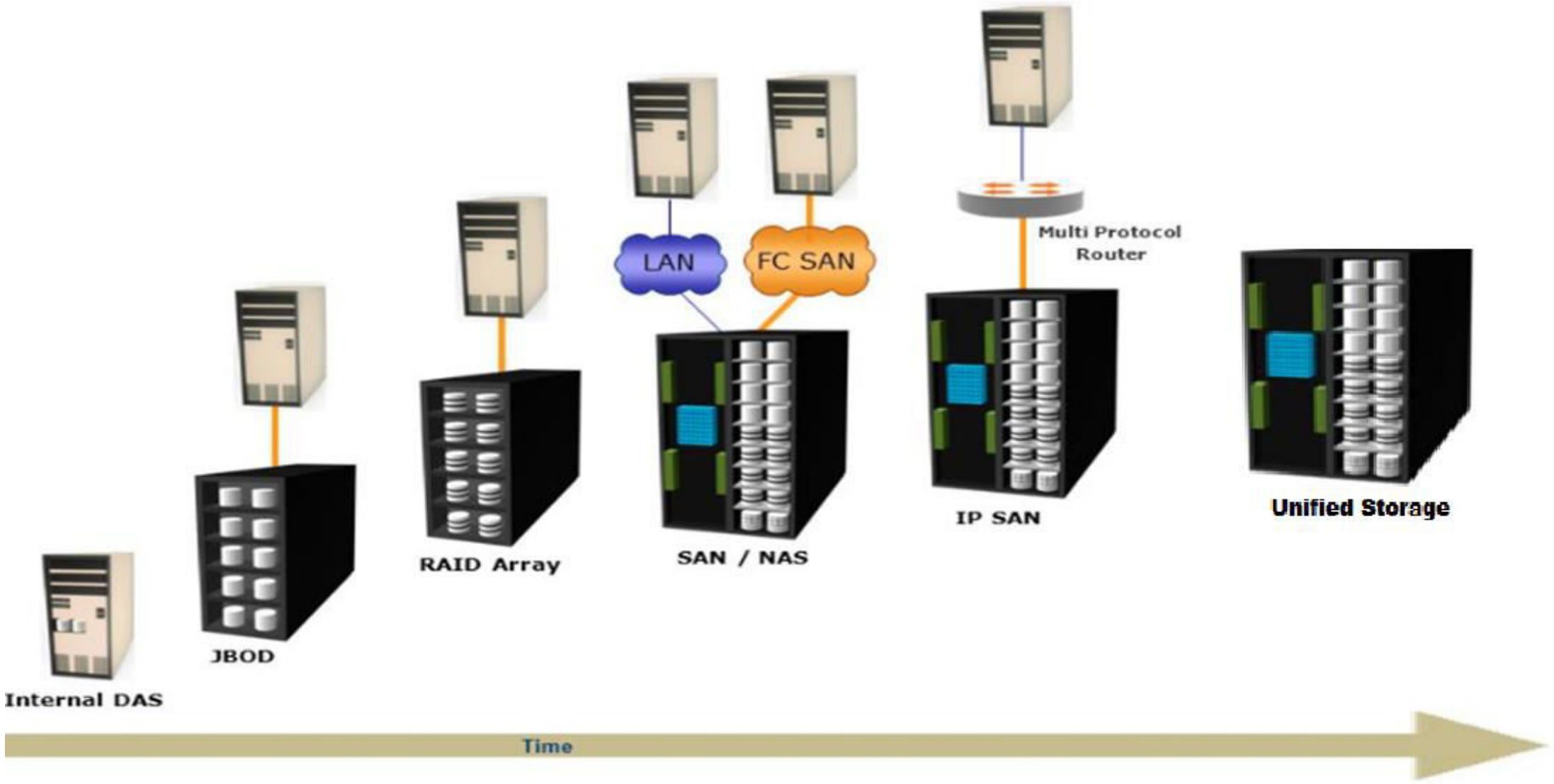


Storage Area Network (SAN)



CLOUD COMPUTING

Evolution of the Storage Systems influenced by Architecture

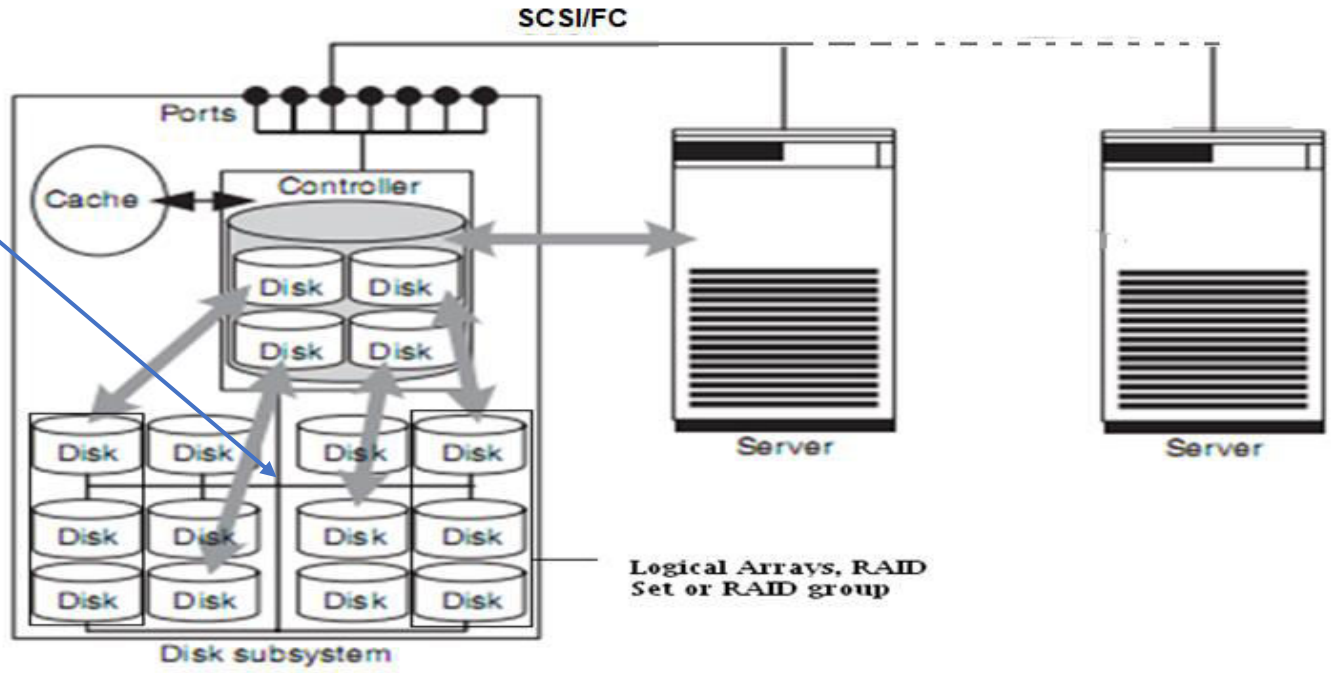


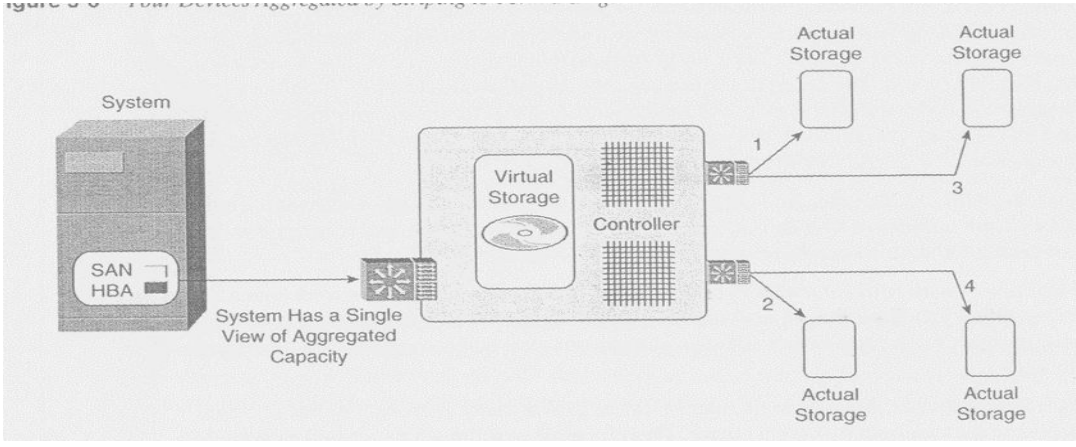
CLOUD COMPUTING

Typical Architecture of a Storage Device/Subsystem

PES UNIVERSITY
CELEBRATING 50 YEARS

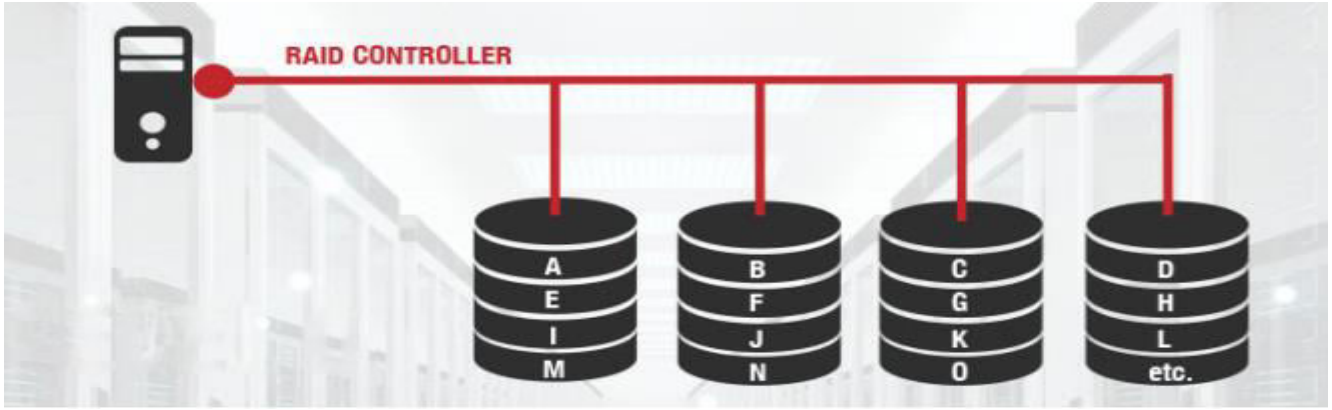
- Figure shows a controller between the disks and the ports
- External Ports are extended to disks through Internal IO Channels.
- Controller functions help with
  - **Increasing data availability and data access performance through RAID**
  - **Uses caches to speedup read and write access to the server**
- Most reasonable disk subsystem would contain
  - Redundant controllers
  - Significant Cache
  - **Storage Disks which can support petabytes of disks**
  - Could weigh over a ton with the size of a large wardrobe
  - **Consolidated disks which provide better utilization of disks**





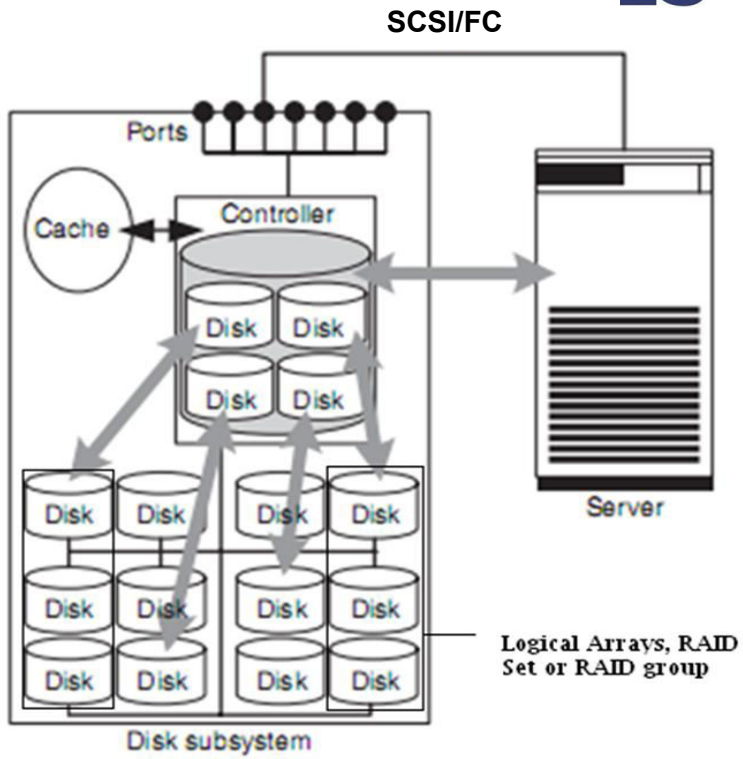
CLOUD COMPUTING

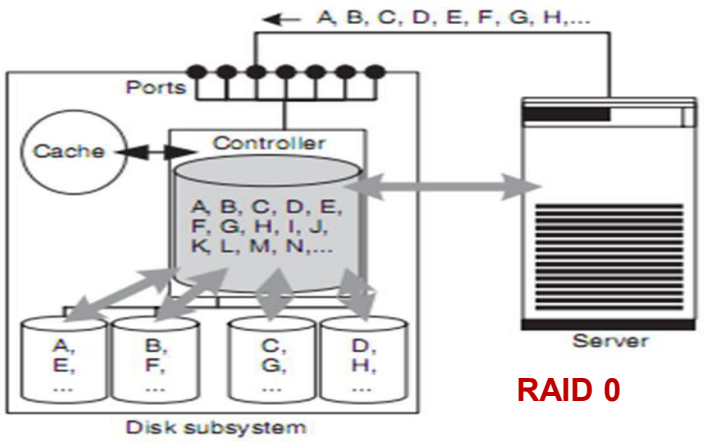
RAID – Redundant Array of Independent Disks



RAID is a **data storage virtualization technology** that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy and performance improvement.

Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the required level of redundancy and performance
Eg. RAID 0, RAID 1, RAID 4, ….

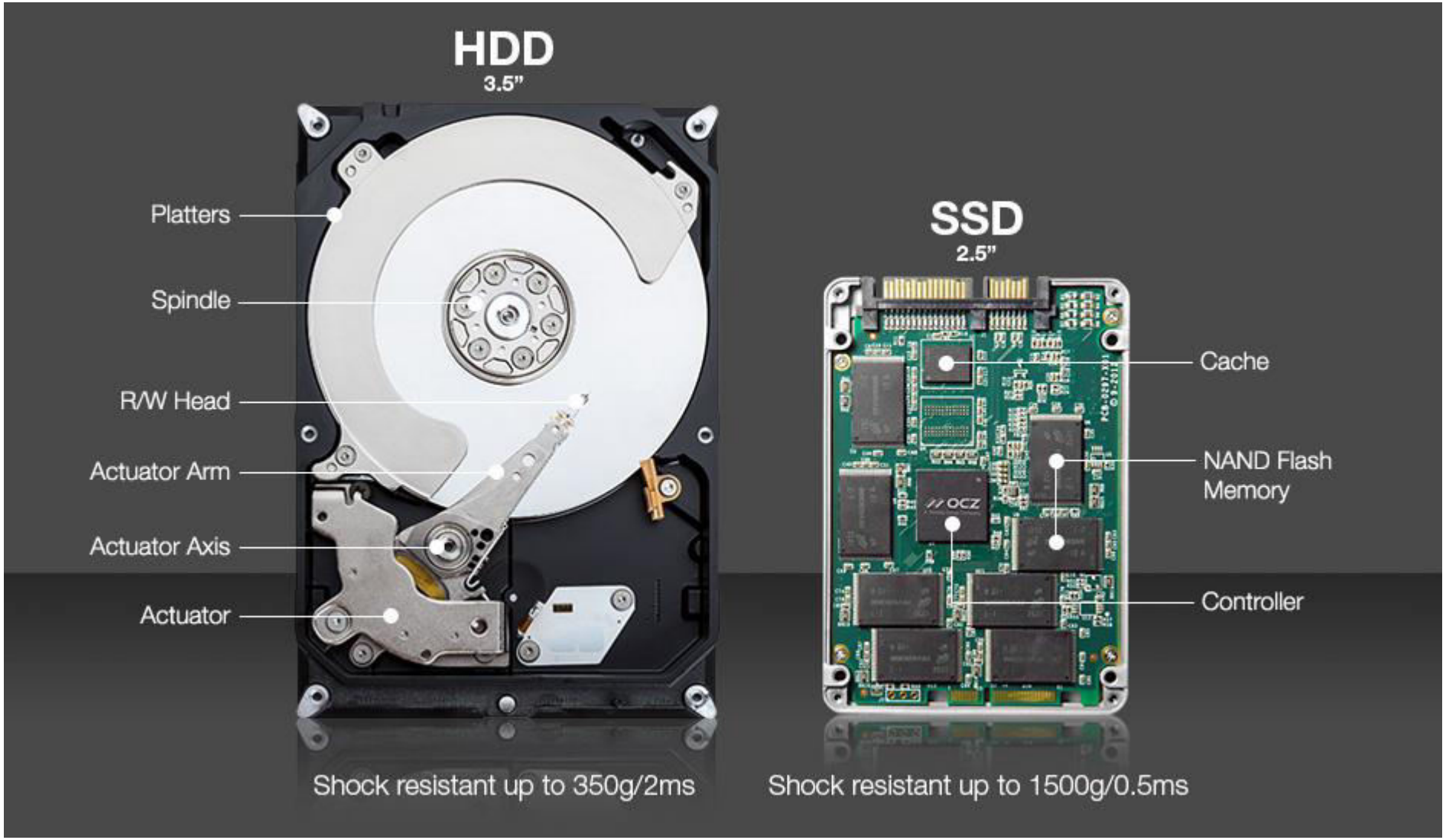




RAID 0

CLOUD COMPUTING

Typical Storage devices within this disk subsystems :



CLOUD COMPUTING

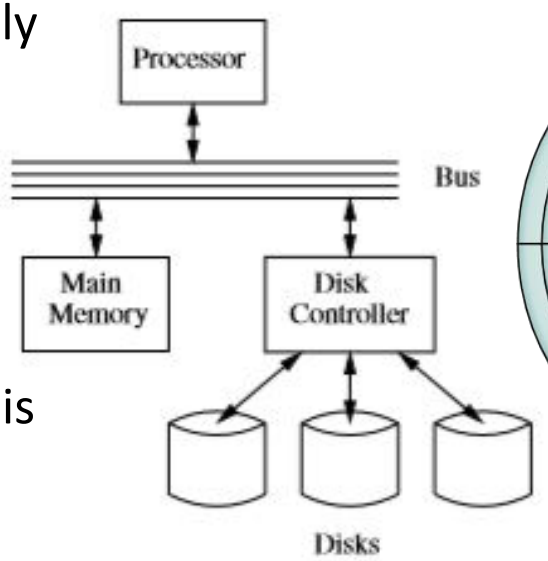
Magnetic Disk Structure and Disk Latency

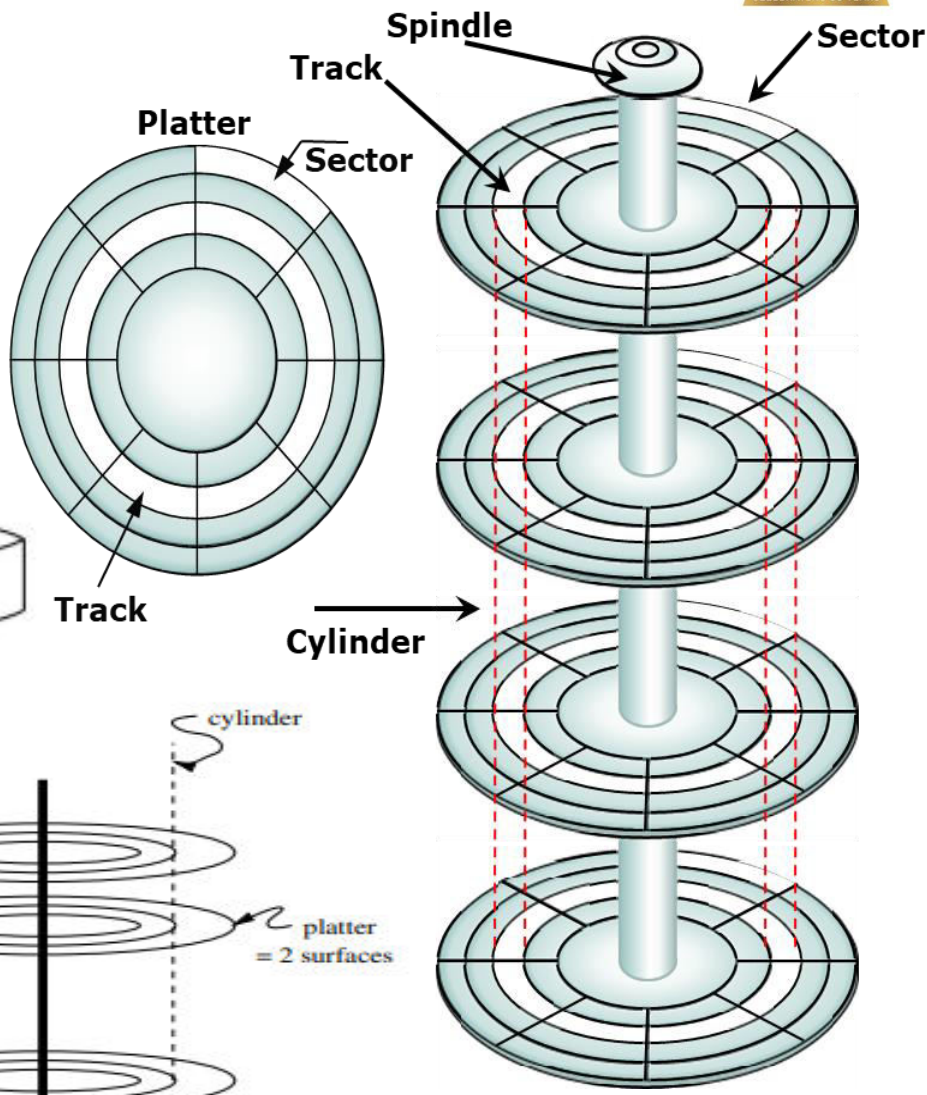
Reading or writing a block involves three steps

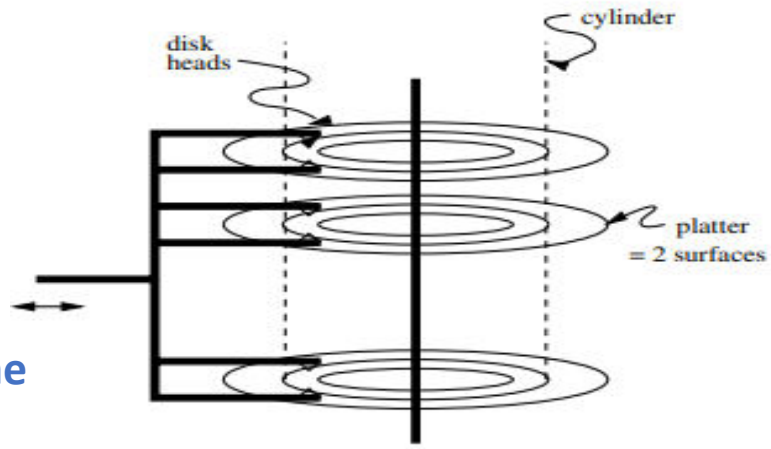
1. The disk controller positions the head assembly at the cylinder containing the track on which the block is located. The time to do so is the **seek time**

2. The disk controller waits while the first sector of the block moves under the head. This time is called the **rotational latency**.

3. All the sectors and the gaps between them pass under the head, while the disk controller reads or writes data in these sectors. This delay is called the **transfer time**







Disk Latency = Seek time + Rotational latency + Transfer time

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Cloud Storage and Enablers for Storage Virtualization

Dr. Prafullata Kiran Auradkar

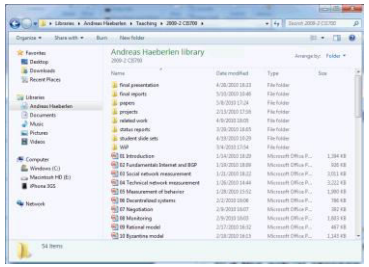
Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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.

What is Cloud Storage : Complex service, simple storage

Consider the following files which you see regularly



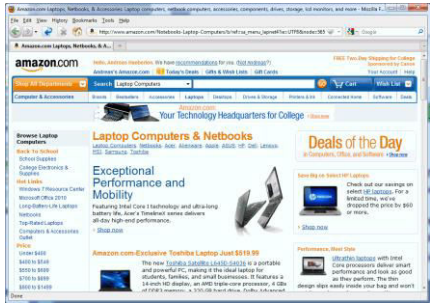
Variable-size files
- read, write, append
- move, rename
- lock, unlock
- ...

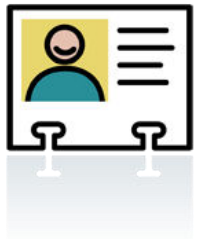
Operating system

Fixed-size blocks
- read
- write

- **PC users see a rich, powerful interface**
  - Hierarchical namespace (directories); can move, rename, append to, truncate, (de)compress, view, delete files, ...
- **But the actual storage device is very simple**
  - HDD only knows how to read and write fixed-size data blocks
- **Translation done by the operating system**

What is Cloud Storage : Analogy to cloud storage





Shopping carts
Friend lists
User accounts
Profiles
...

Web service

Key/value store
- read, write
- delete

- Many cloud services have a similar structure
  - Users see a rich interface (shopping carts, product categories, searchable index, recommendations, ...)
- But the actual storage service is very simple
  - Read/write 'blocks', similar to a giant hard disk
- Translation done by the web service

Cloud Storage

- We have seen the Applications can be thrown into the cloud, and we don't care what server the application lands on. This application needs to store/write or read data, there needs to be an ability for the application to access this seamlessly. This data should also be accessible in case the VM migrates too.
- Cloud Storage provides an ability to an application running some-where, to save data and files in an off-site location, and access the same either through the public internet or a dedicated private network connection from where-ever.
- Storage Virtualization helps supporting access, utilization, availability and other features which are needed by the applications.
- Terminologies like Private cloud, Public cloud, Hybrid Cloud, Internal cloud, external cloud can be applied to the storage based on the where the storage is located, but finally its just the storage systems having the capability of virtual storage pool and multi-tenancy.
- Cloud Storage infrastructure includes the hardware and software cloud components. Object based storage is the prominent approach and access to the infrastructure is via web services API

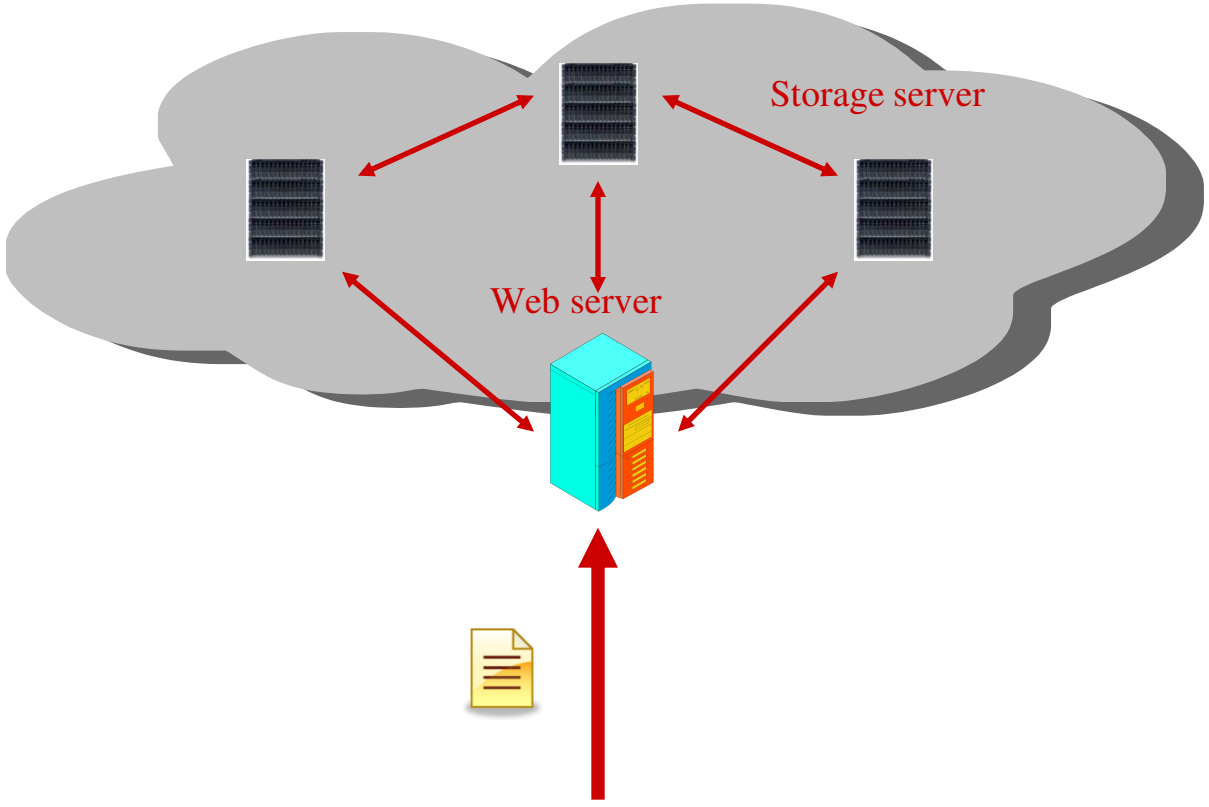
What is Cloud Storage : Cloud Storage

- **Cloud storage** is a data storage service model in which data is maintained, managed, and backed up remotely and made available to users over a network (typically the internet)
- These cloud storage providers are responsible for keeping the data available and accessible, and the physical environment, protected and running.
- People and organizations buy or lease storage capacity from the providers to store user, organization, or application data.
- Cloud Storage or cloud enabled storage can also be visualized as virtual storage pool
- Object storage services like Amazon S3 and Microsoft Azure Storage, object storage software like Openstack Swift, object storage systems like EMC Atmos, EMC ECS and Hitachi Content Platform, and distributed storage research projects like OceanStore and VISION Cloud are all examples of storage that can be hosted and deployed with cloud storage characteristics.

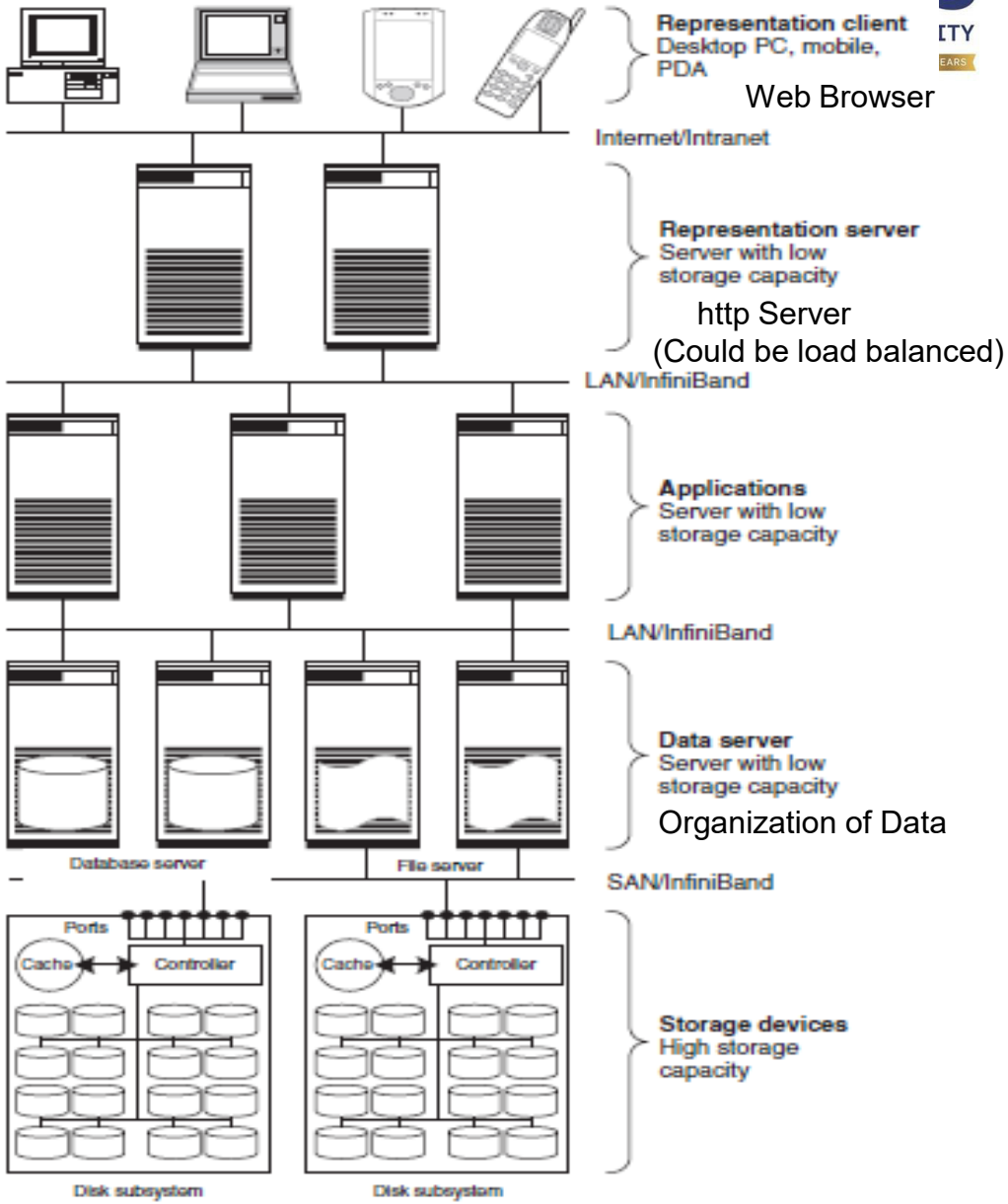
CLOUD COMPUTING

Storage in the Cloud – Cloud Storage Service

Cloud Storage Provider



Client



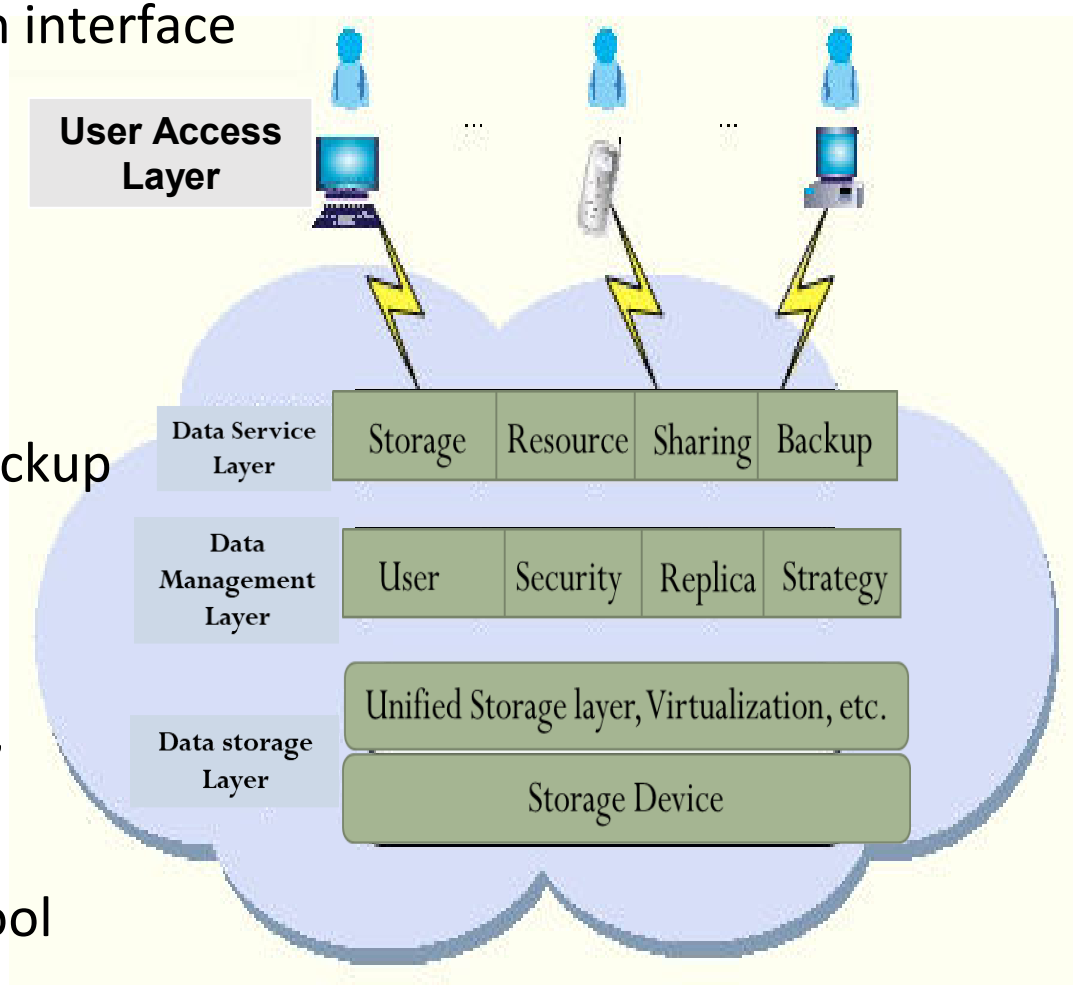
E.g. Architecture of a cloud storage platform

User access layer: an authorized user can log into the cloud storage platform from any location via a standard public application interface and access cloud storage

Data service layer: deals directly with users and depending on user demands, different application interfaces can be developed to provide services such as data storage, space leasing, public resource, multi-user data sharing, or data backup

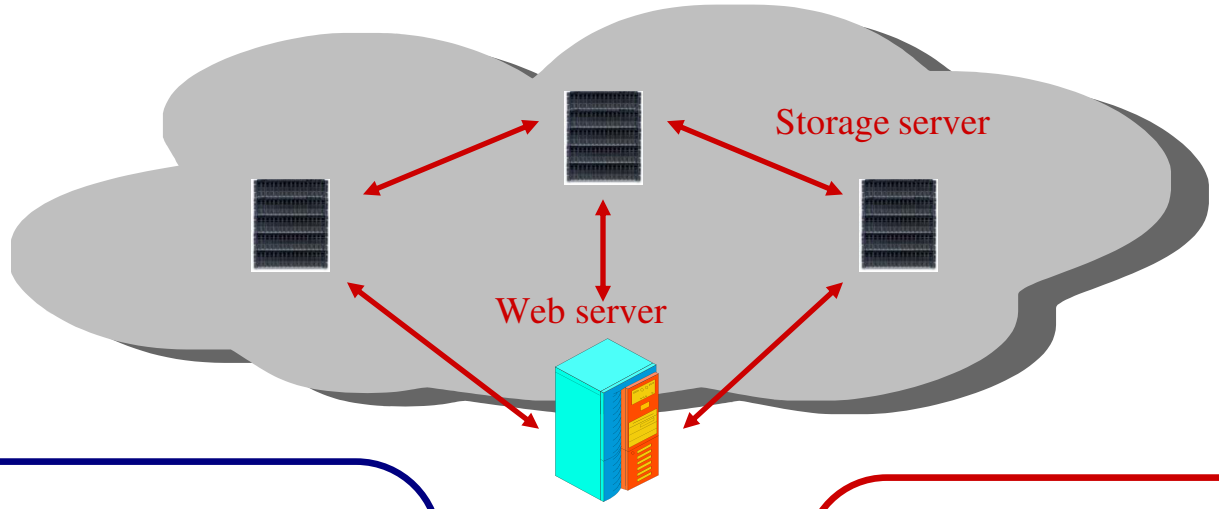
Data management: provides the upper layer with a unified public management interface for different services. With functions such as user management, security management, replica management, and strategy management

Data storage: data stored in the system forms a massive pool and needs to be organized



Cloud storage : Cloud Storage Service

Cloud Storage Provider



Pros:

- Lower cost
- Easier management
- Enables sharing and access from anywhere

Cons:

- Loss of control
- No guarantees of data availability
- Provider failures

Client

What Constitutes Cloud Storage

1. **Dramatic reduction in TCO**
   - Cuts storage cost by more than 10 times compared to block or file storage
2. **Unlimited scalability**
   - Since built using distributed technologies, has unlimited scalability
   - Seamlessly add or remove storage systems from the pool
3. **Elasticity**
   - Storage virtualization decouples and abstracts the storage pool from its physical implementation. So we can get an virtual elastic (grow and shrink as required) and unified storage pool
4. **On-Demand**
   - Uses a pay-as-you-go model, where you pay only for the data stored and the data accessed. For a private cloud, there is a minimal cluster to start with, beyond which it is on-demand.
   - This can result in huge cost savings for the storage user.

What Constitutes Cloud Storage

5. **Universal Access**
   - Traditional storage has limitations like for block storage, the server needing to be on the same SAN , but Cloud storage offers flexibility on the number of users and from where to access the same.
6. **Multitenancy**
   - Cloud Storage is typically multi-tenant and supports centralized management, higher storage utilization and lower costs
7. **Data durability and availability**
   - Runs on commodity hardware but still highly available even with partial failures of the storage system supported by software layer providing the availability
8. **Usability**
9. **Disaster Recovery**

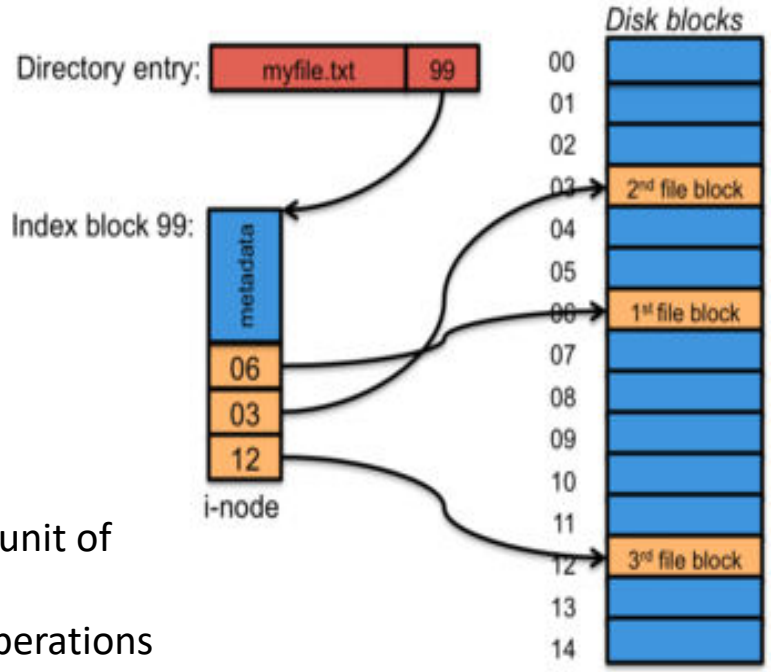
Cloud Storage Enablers

- Storage like Compute resources is enabled for the cloud by Virtualization
- Storage virtualization could be implemented in hardware or software.
- Storage virtualization could also be implemented in the Server, in the storage device and in the network carrying the data
- Techniques and components like File Systems, Volume Manager, Logical Volume Managers enable Storage virtualization in the Server
- Techniques like RAID and Logical Volume Management is also used with Storage virtualization in the storage device
- We will discuss a few of these enablers now and discuss the Storage Virtualization itself in the next session.

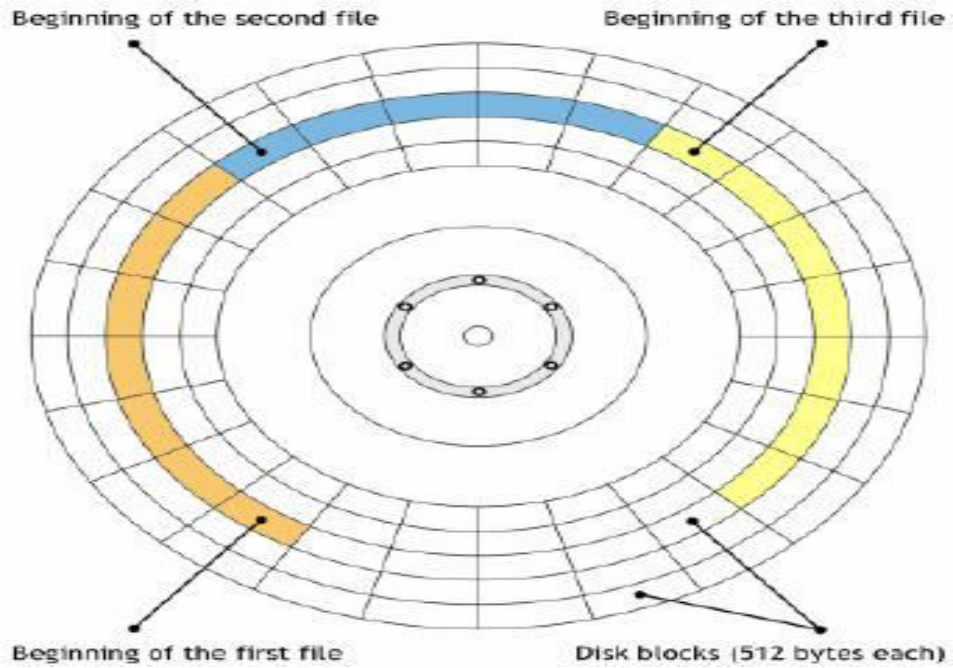
CLOUD COMPUTING

Enablers for Storage Virtualization – File Systems

- Data is separated and grouped into pieces and given a name called a **file**
- The ***structure and logic rules*** used ***to manage the groups of information (files) and their names*** is what forms a file system
- **File system** is used to control how data is stored and retrieved.
  Without a file system, information placed in a storage medium
  would be one large body of data with no way to tell where one
  piece of information stops and the next begins.

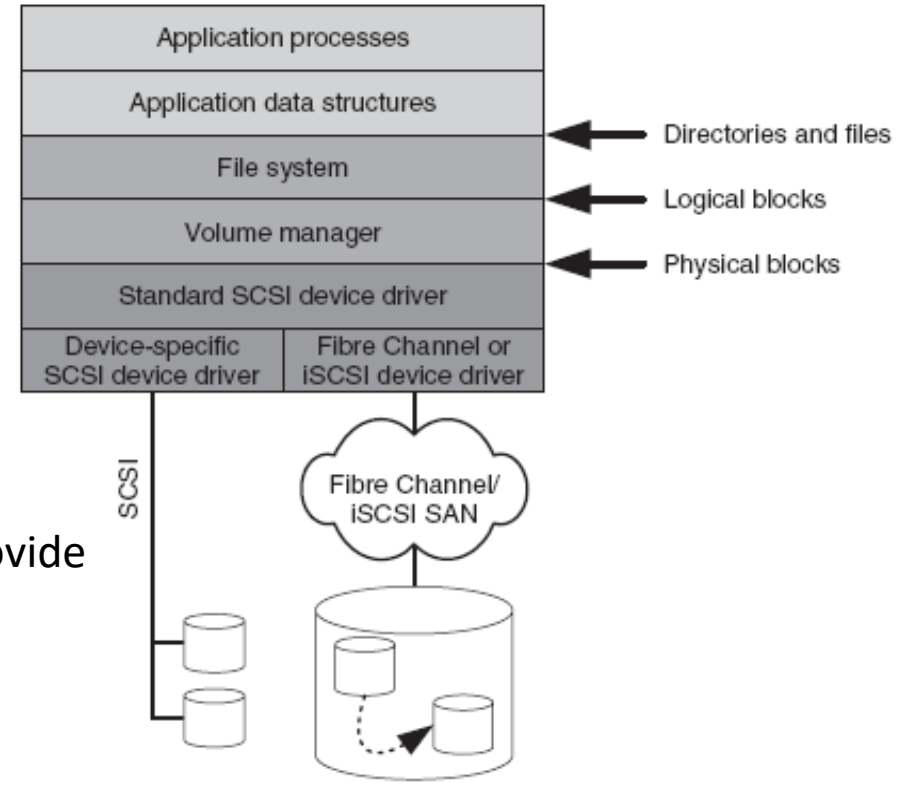


Block is a basic unit of storage for IO (Read/Write) operations

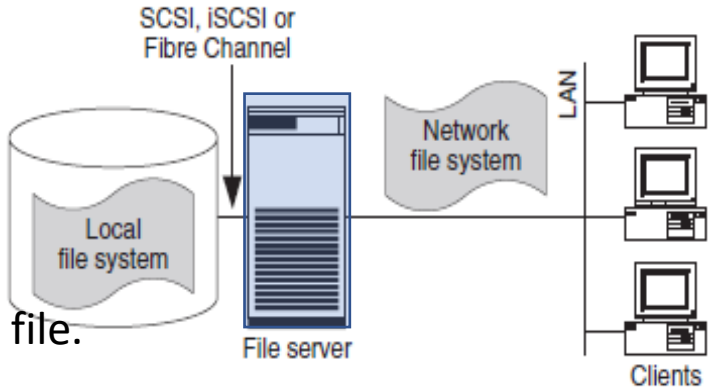


CLOUD COMPUTING

Enablers for Storage Virtualization – File Systems

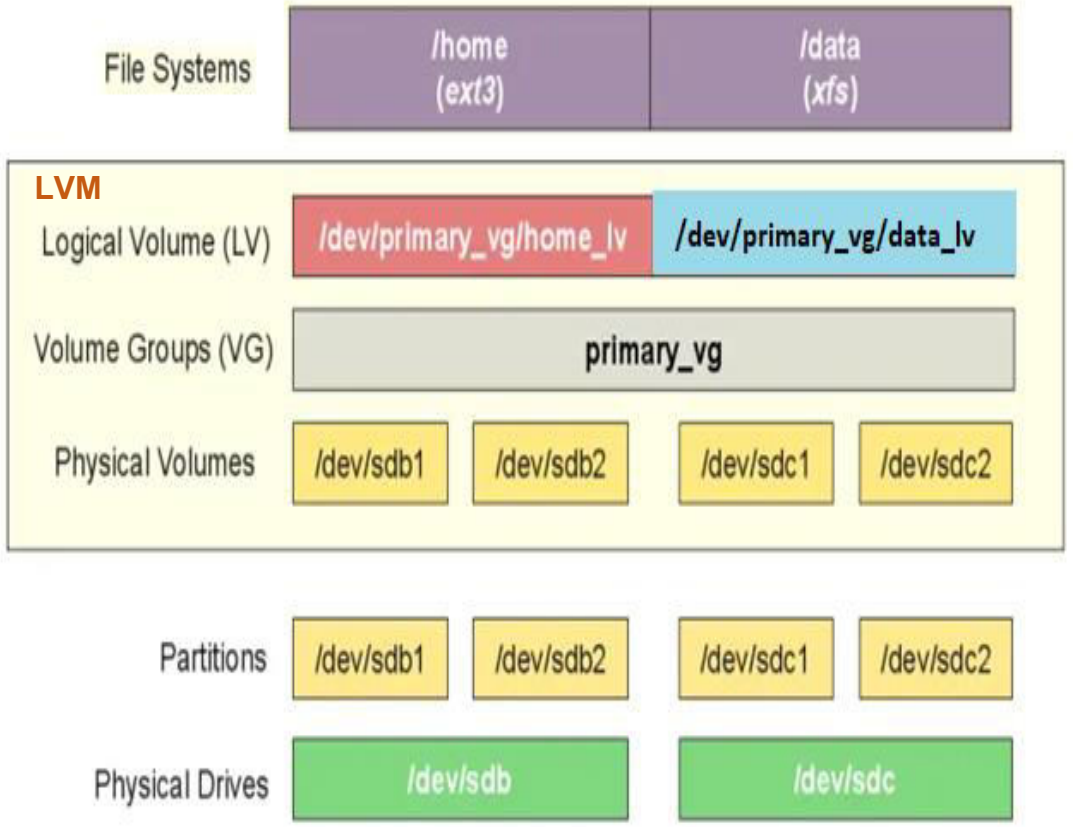


- Local file systems could be seen in this fashion
- File systems
  - Can be used on numerous different types of storage devices that uses different kinds of media
  - There are also many different kinds of file systems.
- Some filesystems are used on local data storage devices and others provide file access via a network interface and is responsible for arranging storage space;
- **Network file systems** make local files and directories available over the LAN. Several end users can thus work on common files
- Network File Systems supports applications to share and access files remotely from various computers
- NFS functionality (in a typical server dedicated to host files and directories and making them available to be accessed across the network), has a daemon process called nfsd. Server administrator exports the directories and advertises the same in a configuration file.
- NFS client requests to the exported directories with the mount command
- Once mounted, its transparent on where the data is being accessed and can be controlled by permissions



Enablers for Storage Virtualization– Logical Volume Manager (LVM)

- Independent layer between the File System and the disk drives
- You could create partitions on the physical disk and create physical volumes.
- These physical volumes could be grouped into a volume group
- This volume group could be broken up into logical volumes
- File system can be created on these Logical volumes and mounted

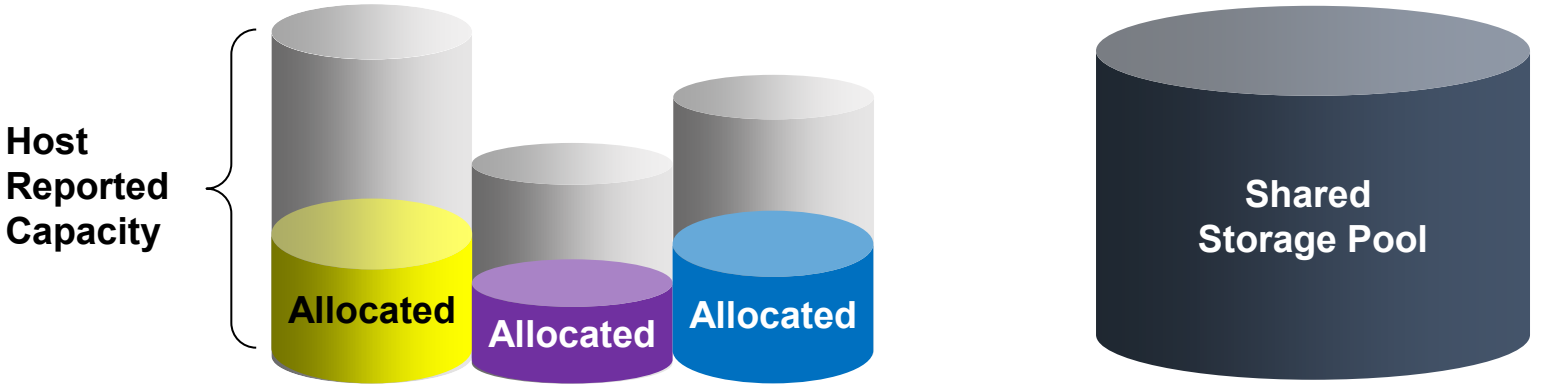


https://opensource.com/business/16/9/linux-users-guide-lvm

Enablers for Storage Virtualization

Thin Provisioning or Virtual Provisioning

- Capacity-on-demand from a shared storage pool
  - Logical units presented to hosts have more capacity than physically allocated (The physical resources are thinly or virtually provisioned)
  - Physical storage is allocated only when the host requires it
  - Provisioning decisions not bound by currently available storage



Storage perceived by the application is larger than physically allocated storage

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Storage Virtualization

Dr. Prafullata Kiran Auradkar

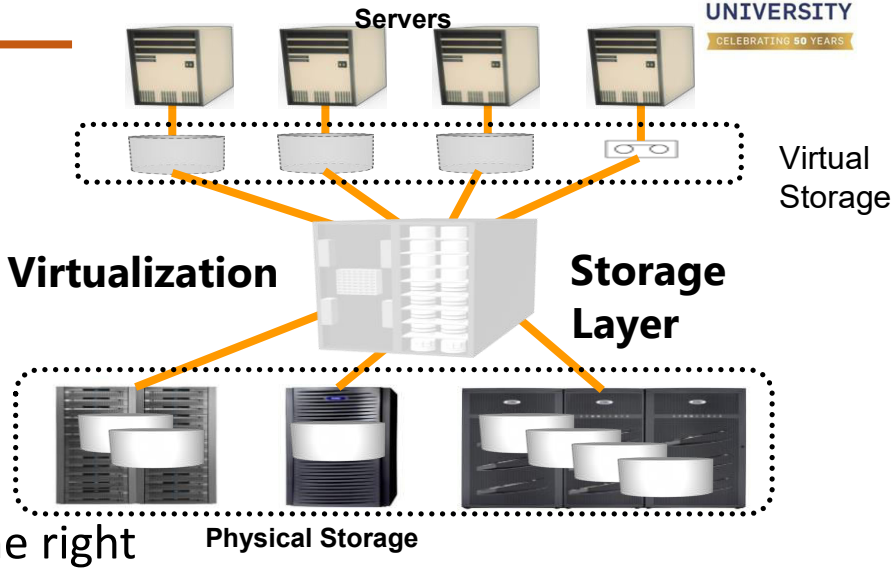
Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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

Storage – Virtualization



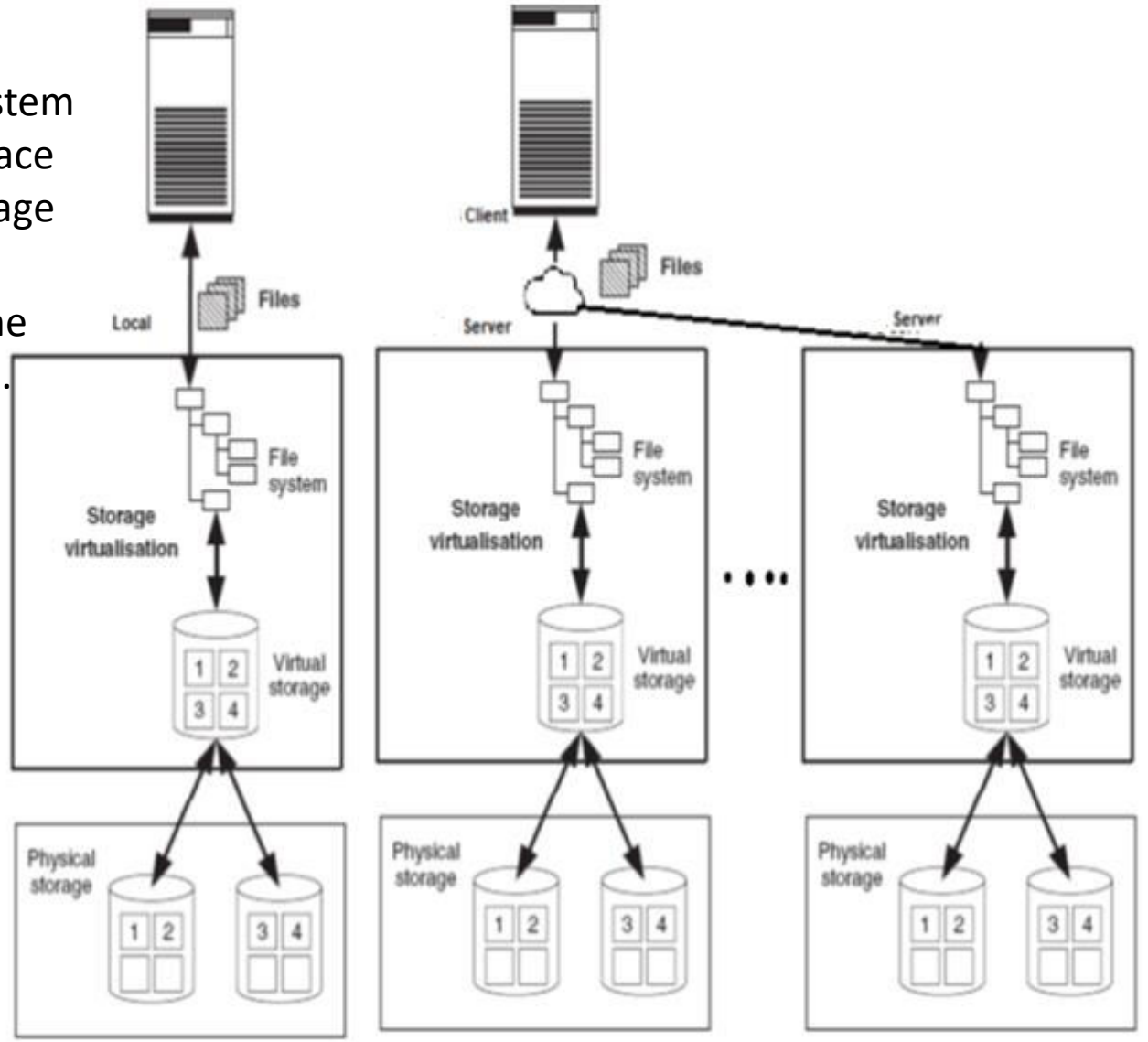
- Storage Virtualization is a means through which physical storage subsystems (disks, tapes ..) are abstracted from the user's application and presented as logical entities, hiding the underlying complexity of the storage subsystems and nature of access, network or changes to the physical devices.
- Its the process of aggregating the capacity of multiple storage devices into storage pools
- Aggregates multiple resources as one addressable entity (pool) or divides a resource to multiple addressable entities and enables easy provisioning of the right storage for performance or cost.
  - E.g. Single virtual large disks from multiple small disks or Many smaller virtual disks from a large disk
- Virtualization of storage helps achieve ***location independence*** by abstracting the physical location of the data. The virtualization system/layer presents to the user a logical space for data storage and handles the ***mapping*** to the actual physical location.
- Virtualization software is responsible for maintaining a consistent view of all of the mapping information and keeping it consistentt. This mapping information is often part of called **metadata**.
- Virtualization layer can be in H/W or in S/W

Storage – Categories of Storage Virtualization

1. File level Virtualization

- A file system virtualization provides an abstraction of a file system to the application (with a standard file-serving protocol interface such as NFS or CIFS) and manages changes to distributed storage hardware underneath the file system implementation.
- Eliminates the dependencies between the data accessed at the file level and the location where the files are physically stored.
- Virtualization layer manages files, directories or file systems across multiple servers and allows administrators to present users with a single logical file system.
- A typical implementation of a virtualized file system is as a network file system that supports sharing of files over a standard protocol with one or more file servers enabling access to individual files.
- **File-serving protocols** that are typically employed are NFS, CIFS and Web interfaces such as HTTP or WebDAV

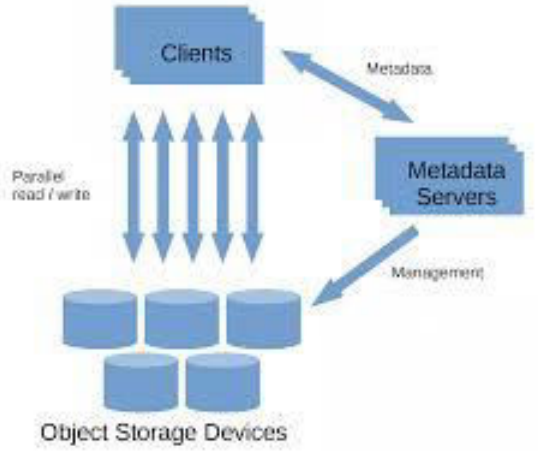
  Adv: This provides opportunities to optimize storage usage, server consolidation & non-disruptive file migrations.



Storage : 1. File Level Virtualization

Distributed File System

- A distributed file system (DFS) is a network file system wherein the file system is distributed across multiple servers.
- DFS enables location transparency and file directory replication as well as tolerance to faults.
- Some implementations may also cache recently accessed disk blocks for improved performance. Though distribution of file content increases performance considerably, efficient management of metadata is crucial for overall file system performance
- Two important techniques for managing metadata for highly scalable file virtualization:
  a. **Separate data from metadata** with a **centralized metadata** server (used in ***Lustre***)
  b. **Distribute data and metadata** on multiple servers (used in ***Gluster***)



Storage : 1. File Level Virtualization – Distributed File System

Distributed File Systems with Centralized Metadata

- A centralized metadata management scheme achieves scalable DFS with a dedicated metadata server to which all metadata operations performed by clients are directed.
- Lock-based synchronization is used in every read or write operation from the clients.
- In centralized metadata systems, the metadata server can become a bottleneck if there are too many metadata operations.
- For workloads with large files, centralized metadata systems perform and scale very well

Storage : 1. File Level Virtualization – Distributed File System

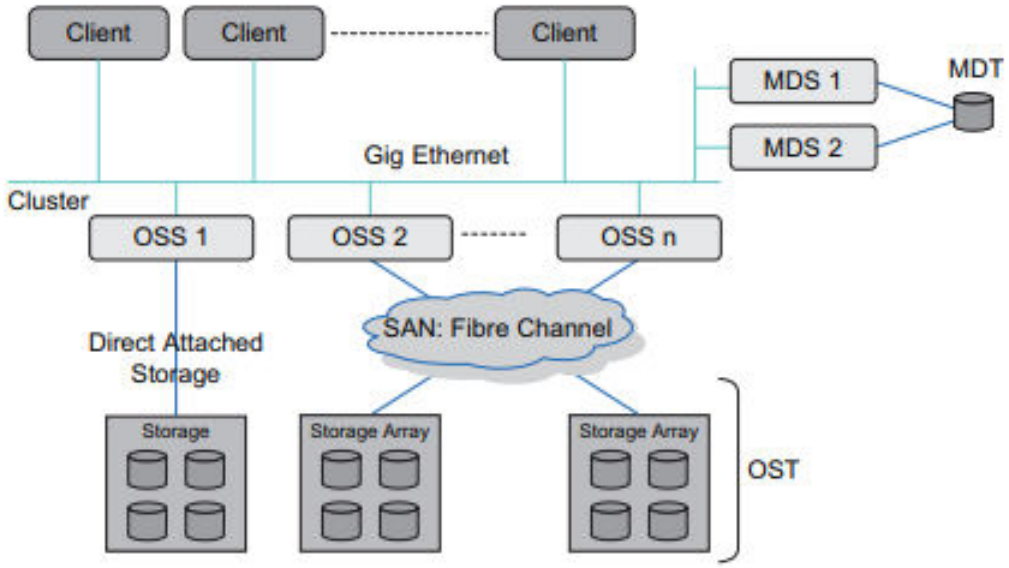
Lustre - Distributed File Systems with Centralized Metadata

- Lustre is a massively parallel, scalable distributed file system for Linux which employs a cluster-based architecture with centralized metadata.
- This is a software solution with an ability to scale over thousands of clients for a storage capacity of petabytes with high performance I/O throughput.
- The architecture of Lustre includes the following three main functional components, which can either be on the same nodes or distributed on separate nodes communicating over a network
    1. Object storage servers (OSSes), which store file data on object storage targets (OSTs).
    2. A single metadata target (MDT) that stores metadata on one or more Metadata servers (MDS)
    3. Lustre Clients that access the data over the network using a POSIX interface

Storage : 1. File Level Virtualization – Distributed File System

Lustre architecture (Functioning)

- When a client accesses a file, it does a filename lookup on a MDS.
- Then, MDS creates a metadata file on behalf of the client or returns the layout of an existing file.
- The client then passes the layout to a logical object volume (LOV) for read or write operations.
- The LOV maps the offset and size to one or more objects, each residing on a separate OST.
- The client then locks the file range being operated on and executes one or more parallel read or write operations directly to the OSTs.



Storage : 1. File Level Virtualization – Distributed File System

Distributed File Systems with Distributed Metadata

- In distributed metadata management, metadata is distributed across all nodes in the system, rather than using centralized metadata servers.
- Such systems have greater complexity than centralized metadata systems, since the metadata management is spread over all the nodes in the system.

Storage : 1. File Level Virtualization – Distributed File System

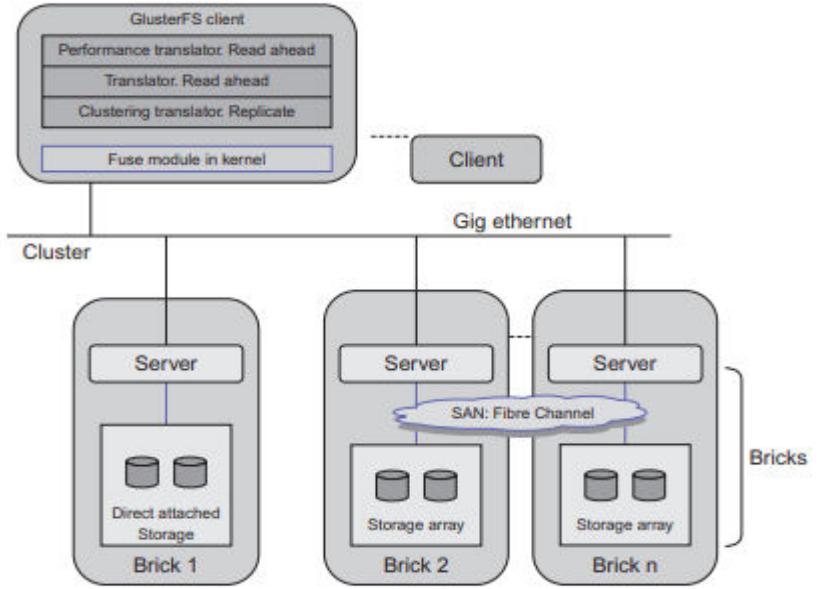
GlusterFS - Distributed File Systems with Distributed Metadata

- GlusterFS is an open-source, distributed cluster file system without a centralized metadata server.
- It is also capable of scaling to thousands of clients, Petabytes of capacity and is optimized for high performance.
- GlusterFS employs a modular architecture with a stackable user-space design. It aggregates multiple storage bricks on a network (over Infiniband RDMA or TCP/IP interconnects) and delivers as a network file system with a global name space.
- It consists of just two major components: a Client and a Server.
  - The Gluster server clusters all the physical storage servers and exports the combined diskspace of all servers as a Gluster File System.
  - The Gluster client is actually optional and can be used to implement highly available, massively parallel access to every storage node and handles failure of any single node transparently

Storage : 1. File Level Virtualization – Distributed File System

Gluster FS architecture

- GlusterFS uses the concept of a storage brick consisting of a server that is attached to storage directly (DAS) or through a SAN.
- Local file systems (ext3, ext4) are created on this storage.
- Gluster employs a mechanism called translators to implement the file system capabilities.
- Translators are programs (like filters) inserted between the actual content of a file and the user accessing the file as a basic file system interface
- Each translator implements a particular feature of GlusterFS.



- Translators can be loaded both in client and server side appropriately to improve or achieve new functionalities
- Gluster performs very good load balancing of operations using the I/O Scheduler translators
- GlusterFS also supports file replication with the Automatic File Replication (AFR) translator, which keeps identical copies of a file/directory on all its subvolumes

Storage : 2. Block Virtualization

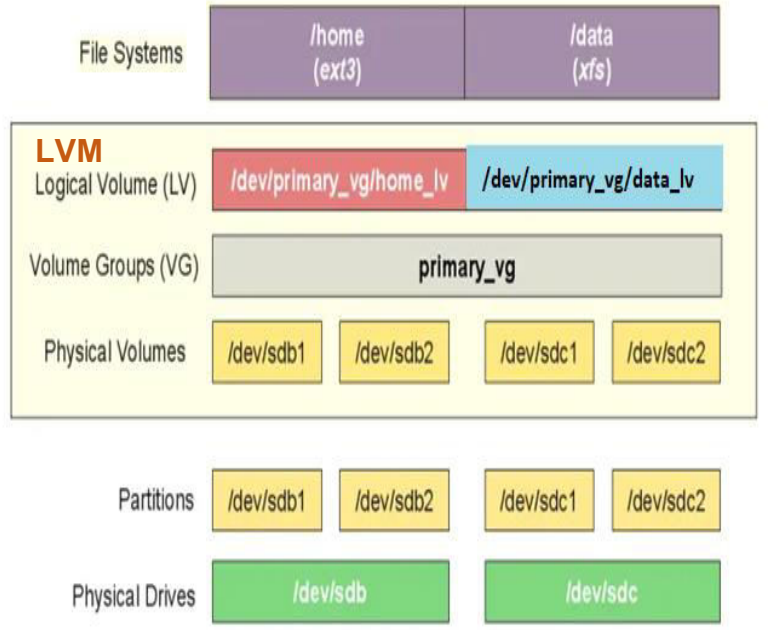
- Block level Virtualization Virtualizes multiple physical disks and presents the same as a single logical disk.
- The data blocks are mapped to one or more physical disks sub-systems.
- These block addresses may reside on multiple storage sub-systems, appearing however as a single storage (logical) storage device.
- Block level storage virtualization can be performed at three levels:
  a. Host-Based
  b. Storage Level
  c. Network level

CLOUD COMPUTING

Storage : 2. Block Virtualization

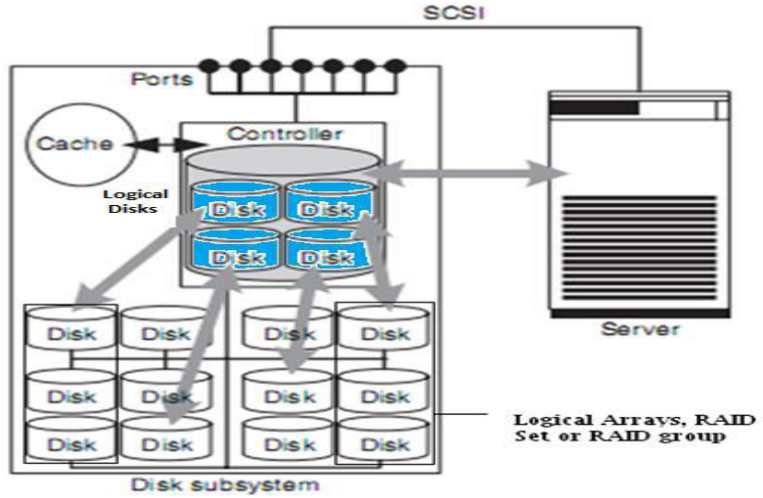
Host-Based block virtualization

- Uses a **Logical Volume Manager** (LVM), a virtualization layer that supports creation of a Storage pool by combining multiple disks (as in the picture) that is greater than the size of a physical disk from where the logical storage is created.
- This also allows transparent allocation and management of disk space for file systems or raw data with capabilities to dynamically shrink or increase physical volumes.



Storage-Device level virtualization

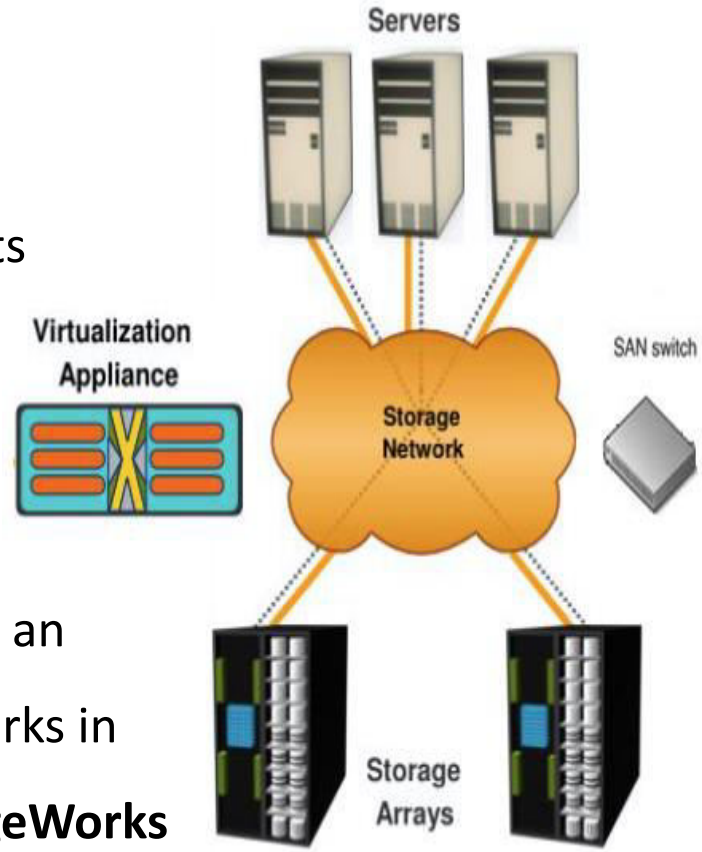
- Creates **Virtual Volumes** over the physical storage space of the specific storage subsystem.
- Storage disk arrays provide this form of virtualization using RAID techniques. Array controllers create **Logical UNits** (LUNs) spanning across multiple disks in the array in RAID Groups. Some disk arrays also virtualize third-party external storage devices attached to the array.
- This technique is generally host-agnostic and has low latency since the virtualization is a part of the storage device itself and in the firmware of the device.



Storage : 2. Block Virtualization

Network-Based block virtualization

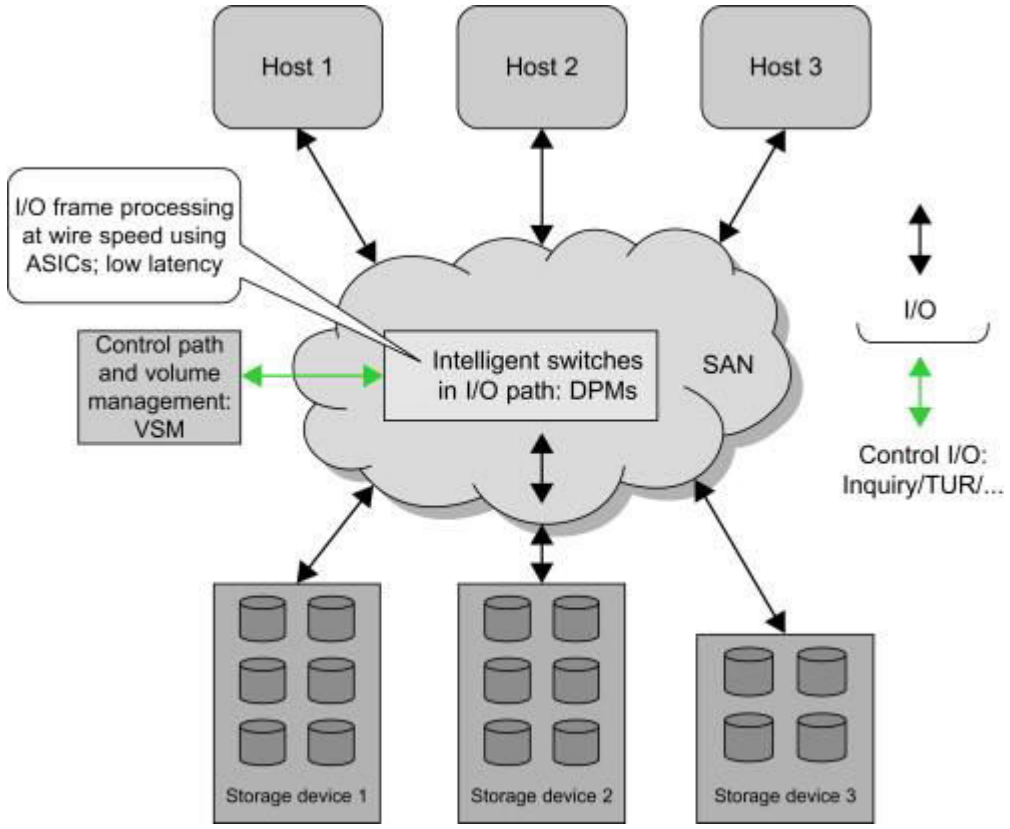
- This is the most commonly implemented form of scalable virtualization.
- Virtualization functionality is implemented within the network connecting hosts and storage, say a Fibre Channel Storage Area Network (SAN).
- There are broadly two categories **based on where the virtualization functions are implemented**: either in **switches** (routers) or in **appliances** (servers).
  - In a switch-based network virtualization, the actual virtualization occurs in an intelligent switch in the fabric and the functionality is achieved when it works in conjunction with a metadata manager in the network. Example**: HP StorageWorks SAN Virtualization Services Platform**
  - In an appliance-based approach, the I/O flows through an appliance that controls the virtualization layer. Example: **IBM SAN Volume Controller**



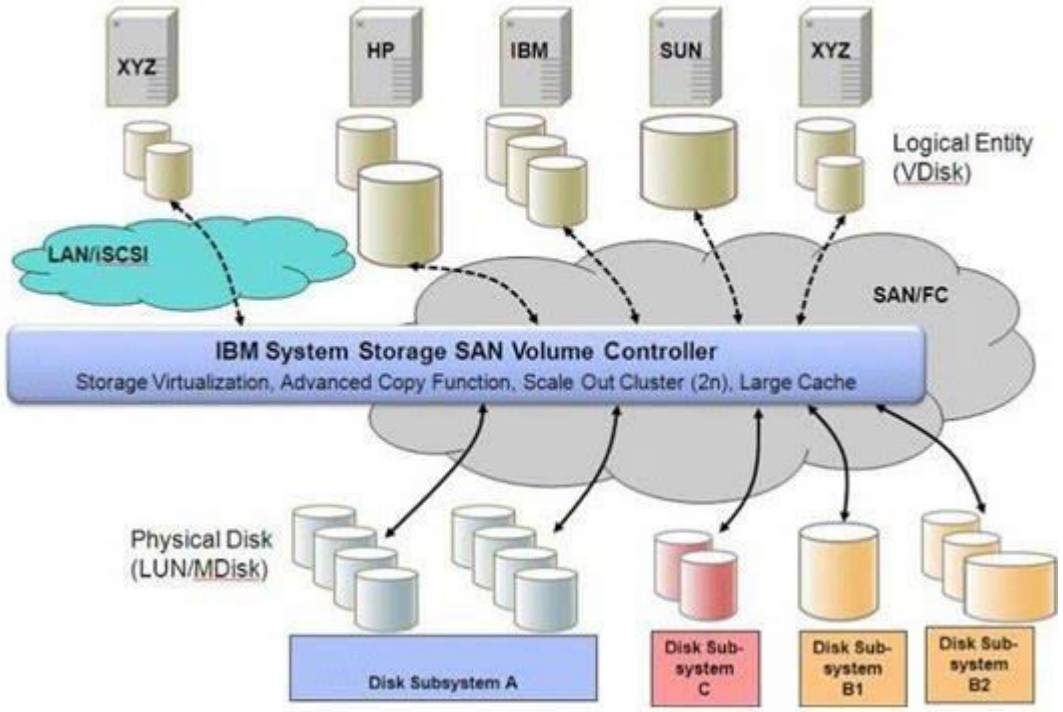
CLOUD COMPUTING

Storage : 2. Block Virtualization

Switch-based network virtualization



Appliance-based network virtualization



Storage : 2. Block Virtualization

Network-Based block virtualization

- There are broadly two variations of an appliance-based implementation.
- The appliance can either be **in-band** or **out-of-band**.
- In **in-band**, all I/O requests and their data pass through the virtualization device and the clients do not interact with the storage device at all. All I/O is performed by the appliance on behalf of the clients
- In **out-of-band** usage, the appliance only comes in between for metadata management (control path), while the data (I/O) path is directly from the client to each host (with agents on each host/client).

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Object Storage

Dr. Prafullata Kiran Auradkar

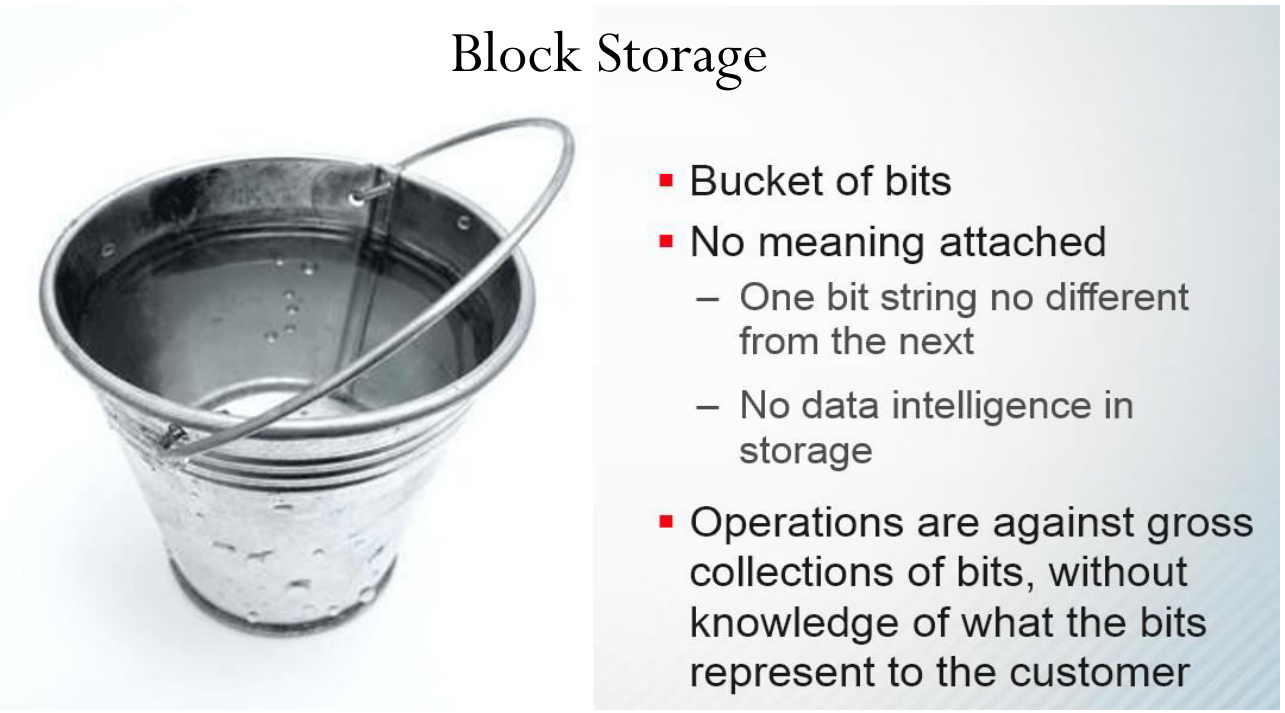
Department of Computer Science and Engineering

Acknowledgements:

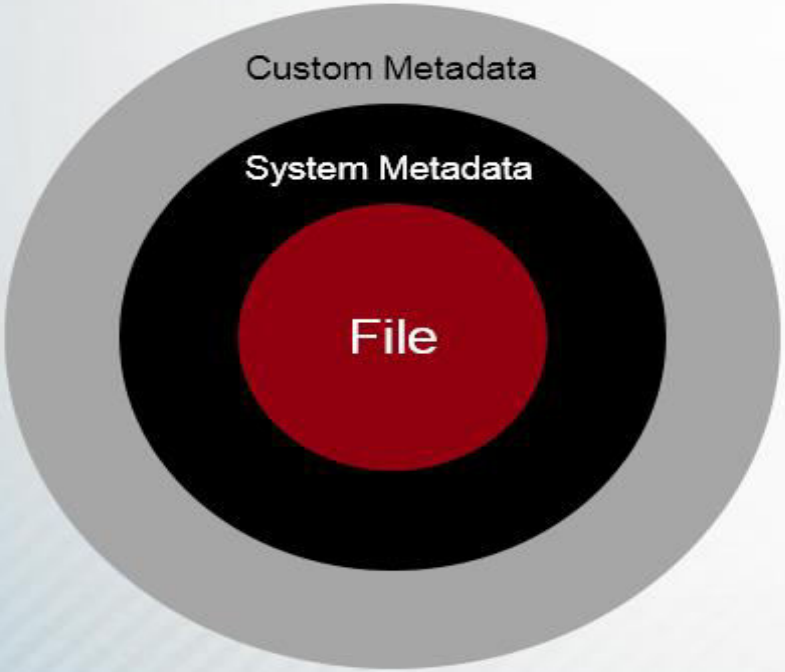
Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture 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.

Object Storage

- Object storage is prominently used approach while building cloud storage systems
- Object storage is different from block or file storage as it allows a user to store data in the form of objects (essentially files in a logical view) by virtualizing the physical implementation in a flat namespace eliminating name collisions using REST HTTP APIs.



Object Storage Example of the Object



File + Metadata = Object

File Class = Image

System Metadata

- Filename: NZ219983.JPG
- Created: January 4, 2012
- Last modified: March 23, 2012

Custom Metadata

- Subject: Castle Point Lighthouse
- Place Taken: New Zealand
- Category: Travel
- Allow sharing: Yes

Object Storage

- Object storage :
  - Data manipulated using GET, PUT, DELETE, UPDATE
  - Use REpresentational State Transfer (REST) APIs
  - A distillation of the way the Web already works
    - Resources are identified by uniform resource identifiers (URIs)
    - Resources are manipulated through their representations
    - Messages are self-descriptive and stateless (XML)
    - Multiple representations are accepted or sent
- Objects contain additional descriptive properties which can be used for better indexing or management.
- Object storage also allows the addressing and identification of individual objects by more than just file name and file path.

Object Storage

- Object storage is built using scale-out distributed systems where each node, most often, actually runs on a local filesystem.
- Object storage does not need specialized & expensive hardware and the architecture allows for the use of commodity hardware
- The most critical tasks of an object storage system are :
  - Data placement
  - Automating management tasks, including durability and availability
- Typically, a user sends their HTTP GET, PUT, POST, HEAD, or DELETE request to any one node out from a set of nodes, and the request is translated to physical nodes by the object storage software.
- The object storage software also takes care of the durability model by doing any one of the following:
  - creating multiple copies of the object and chunking it
  - creating erasure codes (*Erasure coding (EC) is a method of data protection in which data is broken into fragments, expanded and encoded with redundant data pieces and stored across a set of different locations or storage media*

  or a combination of these.
- Supports activities towards Management, such as periodic health checks, self-healing, and data migration.
- Management is also made easy by using a single flat namespace, which means that a storage administrator can manage the entire cluster as a single entity.

Object Storage : Illustration – Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3)

- Amazon S3 is a **highly reliable, highly available, scalable and fast storage in the cloud for storing and retrieving large amounts of data just through simple web services.**
- There are three ways of using S3. Most common operations can **be performed via the AWS console (GUI interface to AWS)**
- For use of S3 within applications, Amazon provides a **REST-ful API with familiar HTTP operations such as GET, PUT, DELETE, and HEAD.**
- There are libraries and SDKs for various languages that abstract these operations
- Since S3 is a storage service, several S3 browsers exist that allow users to explore their S3 account as if it were a directory (or a folder). There are also file system implementations that let users treat their S3 account as just another directory on their local disk.

Object Storage - Amazon S3 – Organizing Data in S3

Organizing Data In S3: Buckets, Objects and Keys

- Data is stored as **objects** in S3.
- These objects in S3 are stored in resources called **buckets**
- S3 objects can be up to 5 Terabytes in size and there are no limits on the number of objects that can be stored.
- Objects in S3 are replicated across multiple geographic locations to make it resilient to several types of failures.
- Objects are referred to with **keys** – basically an optional directory path name followed by the name of the object.
- If object versioning is enabled, recovery from inadvertent deletions and modifications is possible.

Organizing Data In S3: Buckets, Objects and Keys (Cont.)

- Buckets provide a way to keep related objects in one place and separate them from others. There can be up to 100 buckets per account and an unlimited number of objects in a bucket.
- Each object has a key, which can be used as the path to the resource in an HTTP URL.
- **Example**: if the bucket is named johndoe and the key to an object is resume.doc, then its HTTP URL is http://s3.amazonaws.com/johndoe/resume.doc or alternatively, http://johndoe.s3.amazonaws.com/resume.doc
- By convention, slash-separated keys are used to establish a directory-like naming scheme for convenient browsing in S3

Object Storage - Amazon Simple Storage Service (S3)

Security

- Users can ensure the security of their S3 data by two methods
  1. **Access control to objects**: Users can set permissions that allow others to access their objects. This is accomplished via the AWS Management Console.
  2. **Audit logs**: S3 allows users to turn on logging for a bucket, in which case it stores complete access logs for the bucket in a different bucket. This allows users to see which AWS account accessed the objects, the time of access, the IP address from which the accesses took place and the operations that were performed. Logging can be enabled from the AWS Management Console

Object Storage - Amazon Simple Storage Service (S3)

Data Protection

1. **Replication**:
   - By default, S3 replicates data across multiple storage devices, and is designed to survive two replica failures.
   - It is also possible to request Reduced Redundancy Storage(RRS) for noncritical data. RRS data is replicated twice, and is designed to survive one replica failure.
   - S3 provides strong consistency if used in that mode and guarantees consistency among the replicas.
2. **Regions**:
   - For performance, legal, Availability and other reasons, it may be desirable to have S3 data running in specific geographic locations.
   - This can be accomplished at the bucket level by selecting the region that the bucket is stored in during its creation.

Data Protection

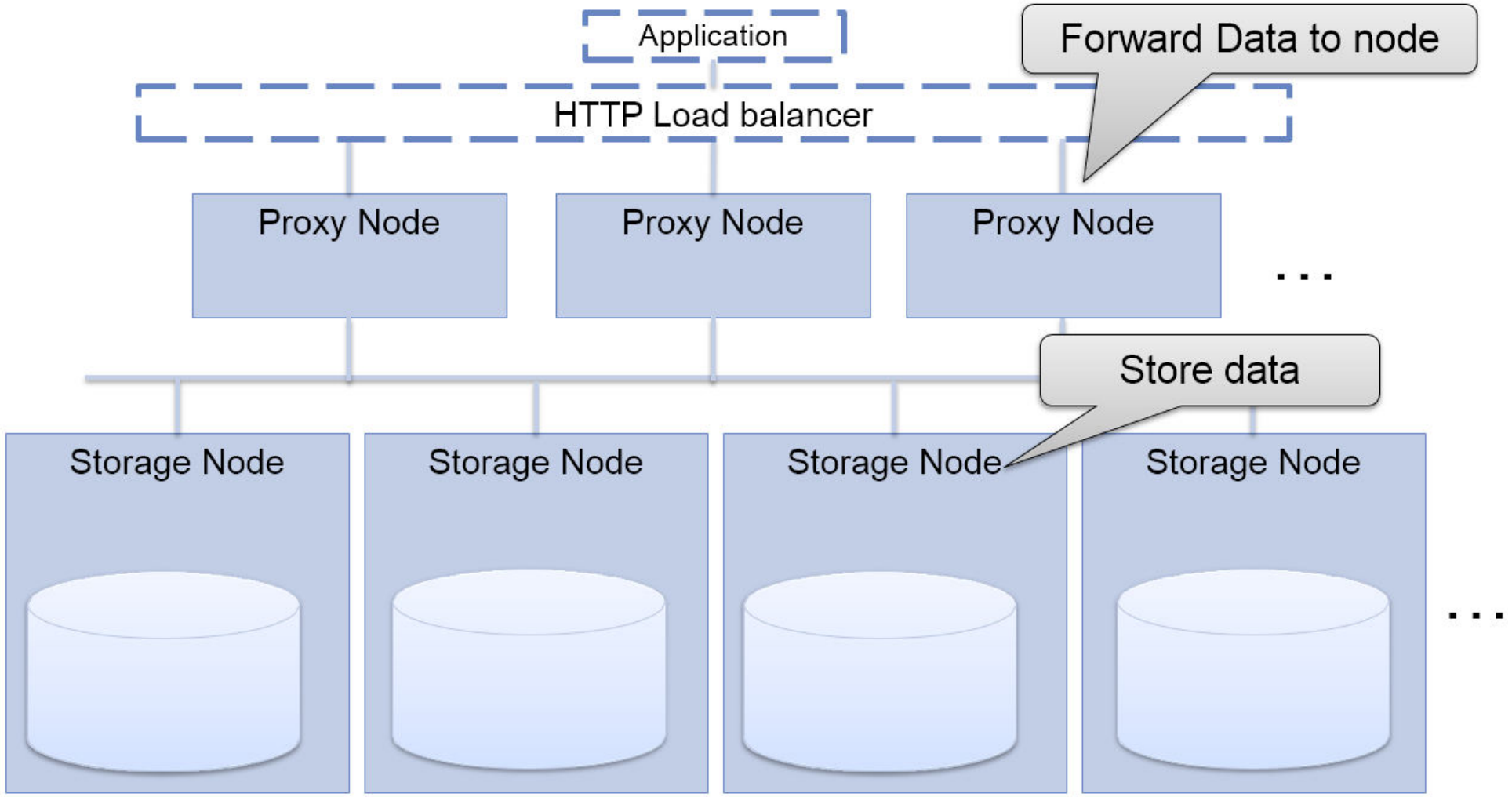
3. Versioning:

- If versioning is enabled on a bucket, then S3 automatically stores the full history of all objects in the bucket from that time onwards.
- The object can be restored to a prior version and even deletes can be undone. This guarantees that data is never inadvertently lost

Large Objects and Multi-part Uploads

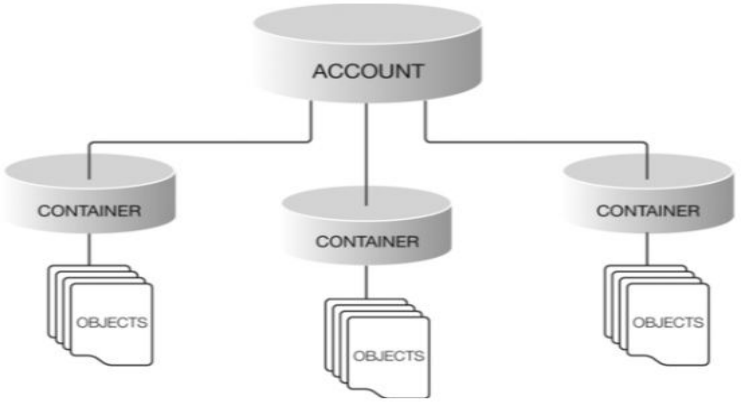
- S3 provides APIs that allow the developer to write a program that splits a large object into several parts and uploads each part independently.
- These uploads can be parallelized for greater speed to maximize the network utilization. If a part fails to upload, only that part needs to be re-tried

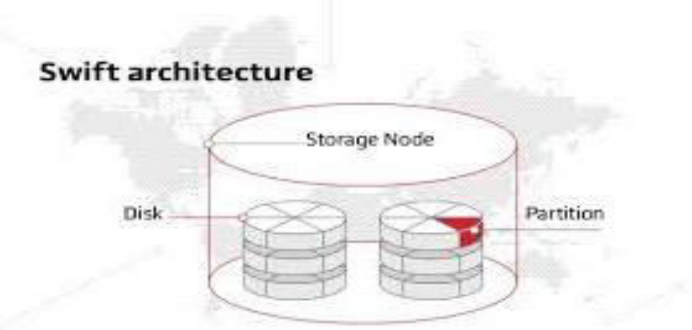
Open Stack Swift – Another Illustration of Object Storage

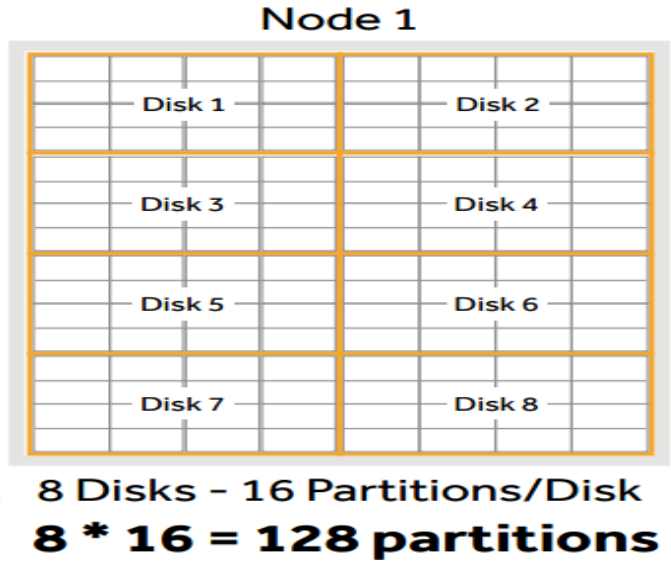


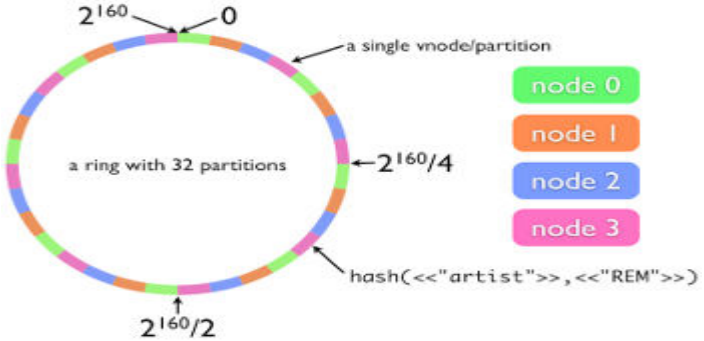
Swift

- **Swift Partitions** breaks the storage available into locations where data will be located including account databases, container databases or objects. It's the core of the replication system and distributed across all disks.
  Its like a moving bin moving in a warehouse. These are not linux partitions. Adding a node leads to reassigning of partitions
- **Account** is an user in the storage system- unlike volumes, swift creates accounts which enables multiple users and applications to access the storage system at the same time
- **Container:** Containers are where Accounts create and store data.
  Containers are name spaces used to group objects (conceptually like directori
- **Object :** Actual data is stored on the disk. This could be photos etc.
- **Ring** maps the partition space to physical locations on disk. Its like an encyclopedia, instead of letters swift used hash for searchin









NoSQL Database - DynamoDB

- DynamoDB is a NoSQL fully managed cloud-based document and a Key Value database available through Amazon Web Services (AWS)
- All data in DynamoDB is stored in tables that you have to create and define in advance, though tables have some flexible elements and can be modified later
- DynamoDB requires users to define only some aspects of tables, most importantly the structure of keys and local secondary indexes, while retaining a schema less flavor.
- DynamoDB enables users to query data based on secondary indexes too rather than solely on the basis of a primary key
- DynamoDB supports only item-level consistency, which is analogous to row-level consistency in RDBMSs
- If consistency across items is a necessity, DynamoDB is not the right choice
- DynamoDB has no concept of joins between tables. Table is the highest level at which data can be grouped and manipulated, and any join-style capabilities that you need will have to be implemented on the application side

CLOUD COMPUTING

DynamoDB

- In DynamoDB, tables, items, and attributes are the core components
- A table is a collection of items and each item is a collection of attributes
- DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility
- DynamoDB supports two different kinds of primary keys:
    - **Partition key** – A simple primary key, composed of one attribute known as the partition key. DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored.
    - **Partition key and sort key** – Referred to as a composite primary key, this type of key is composed of two attributes. The first attribute is the partition key and the second attribute is the sort key. All items with the same partition key value are stored together in sorted order by sort key value.

People

```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/dynamodb-dg.pdf#HowItWorks

DynamoDB

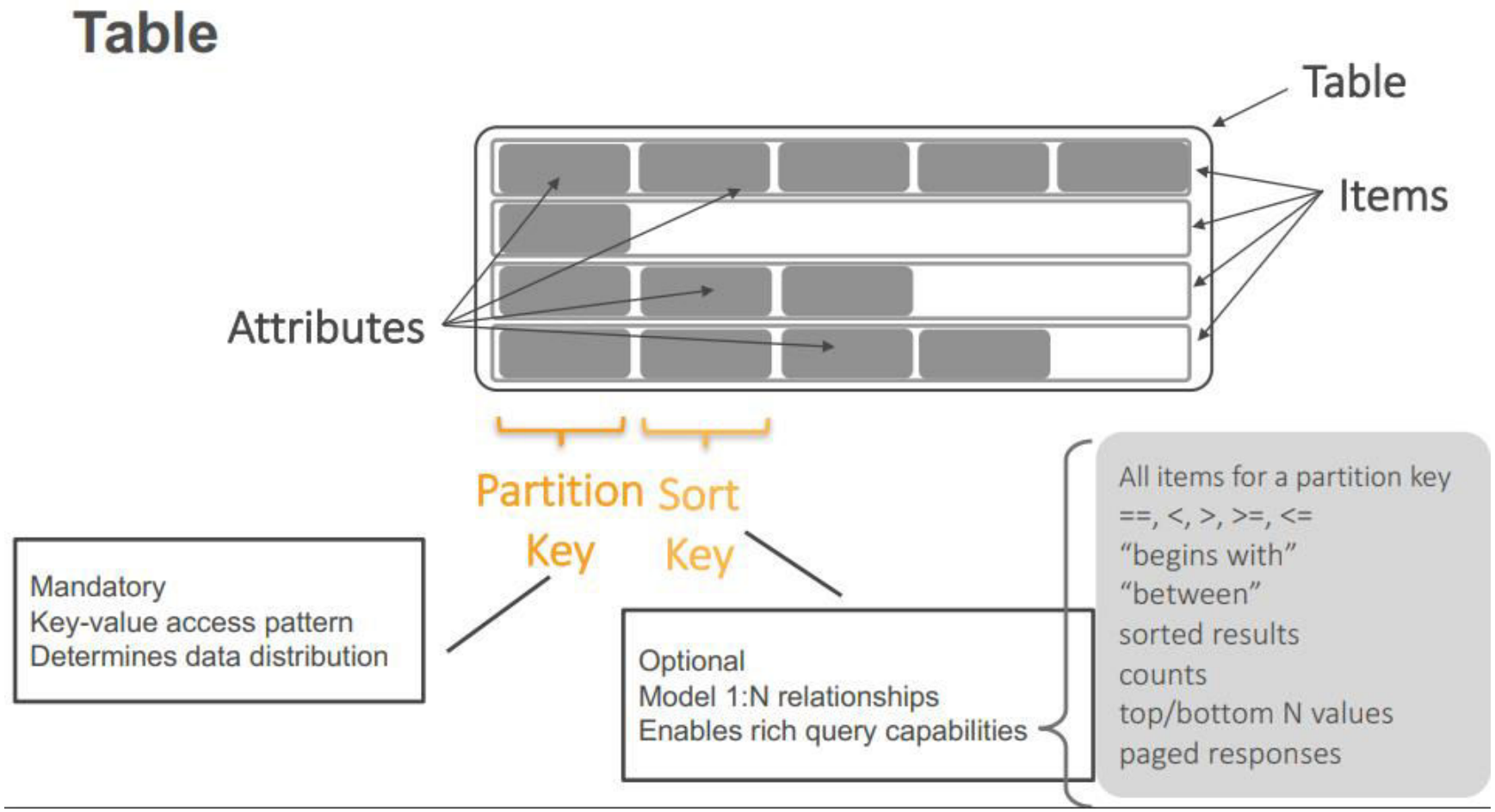
Secondary Indexes

- Users can create one or more secondary indexes on a table.
- A secondary index lets users query the data in the table using an alternate key, in addition to queries against the primary key.

Partitions and Data Distribution

- DynamoDB stores data in partitions.
- A partition is an allocation of storage for a table, backed by solid state drives (SSDs) and automatically replicated across multiple Availability Zones within an AWS Region.
- Partition management is handled entirely by DynamoDB

DynamoDB Architecture



THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Partitioning and Consistent Hashing

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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

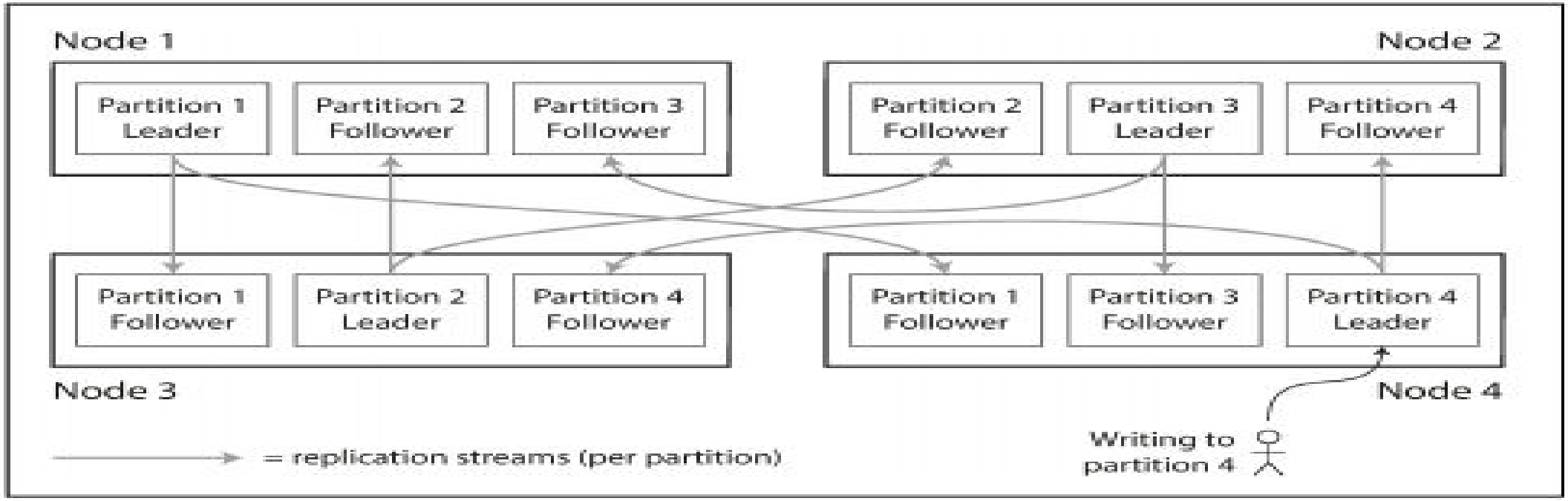
Partitioning

- Cloud Applications typically scale using elastic computing resources for addressing variations in the workload. This could lead to a bottleneck in the backend in-spite of scaling the hosts, if they continue to use the same data store.
- Partitioning is a way of intentionally breaking this large Volume of data into smaller partitions of data, where each of these partition can be placed on a single node or distributed across different nodes in a cluster, such that query or IO operations can be done across these multiple nodes supporting the performance and throughput expectations of applications.
- Partitions are defined in such a way that each piece of data (if its say a DB, then each record, row, or document) belongs to exactly one partition. There could be many operations which will touch partitions at the same time
- Each node can independently execute the queries or do IO and operate on these single (own) partition enabling scaling of throughput with the addition of more nodes.

  Eg. Partitioning a large piece of data to be written to disk into multiple partitions and distributing these partitions to multiple disks will lead to better total IO performance.

Partitioning

- Thus Large complex queries or large IO workload can potentially be parallelized across many nodes
- Partitioning is usually combined with replication so that copies of each partition are stored on multiple nodes. This means that, even though each record belongs to exactly one partition, it may still be stored on several different nodes for fault tolerance. (We will discuss about replication in subsequent sessions)
- A node may store more than one partition



Goal of Partitioning and some terminologies

- Goal of partitioning is to spread the data and the query load evenly across nodes.
- If some partitions have more data than others, we call it **skewed**
- The presence of skew makes partitioning much less effective. In an extreme case, all the load could end up on one partition.
- A partition with disproportionately high load is called a **hot spot**

CLOUD COMPUTING

Different Approaches of Partitioning

There are 4 different approaches of partitioning

1. Vertical Partitioning

- In this approach, data is partitioned vertically.
- Its also called column partitioning where set of columns are stored on one data store and other to a different data store (could be on a different node) and data is distributed accordingly.
- In this approach no two critical columns are stored together which improve the performance

First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Relational DB in a Block/File Storage

Object Storage

2. Workload Driven Partitioning

- Data access patterns generated from the application is analysed and partitions are formed according to that. This improves the scalability of transactions in terms of throughput and response time

Different Approaches of Partitioning

3. Partitioning by Random Assignment

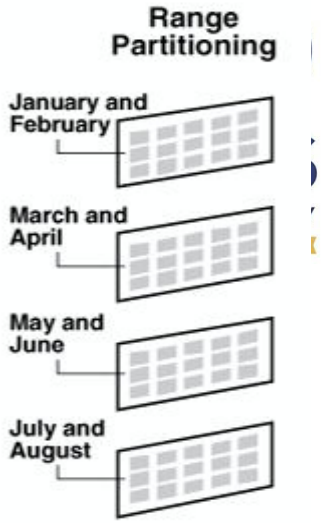
- Random assignment of of records to nodes would be the simplest approach for avoiding hot spots.
- This would distribute the data quite evenly across the nodes,
- The disadvantage of this would be, when trying to read a particular item, you have no way of knowing which node it is on, so you have to query all nodes in parallel.

4. Horizontal Partitioning

- This is a static approach of horizontally partitioning data to store it on different nodes
- Once its partitioned, this would not change.
- There are different techniques which are used for horizontal partitioning like
  1. ***Partitioning by Key Range***
  2. ***Partitioning using the Schema***
  3. ***Partitioning using Graph Partitioning***
  4. ***Partitioning using* Hashing**

CLOUD COMPUTING

4.1 Partitioning by Key -Range



- Range partitioning, is partitioning the cloud data by using range of keys by assigning a continuous range of keys to each partition (from some minimum to some maximum).
- The values of the keys should be adjacent but not overlapped
- Range of keys is decided on the basis of some conditions or operators.
- If we know the boundaries between the ranges, we can easily determine which partition contains a given key
- If we also know which partition is assigned to which node, then we can make the request directly to the appropriate node
- The ranges of keys are not necessarily evenly spaced, because data may not be evenly distributed Within each partition, we can keep keys in sorted order.
- **Example:**
  - Consider an application that stores data from a network of sensors, where the key is the timestamp of the measurement (year-month-day-hour-minute-second).
  - Range scans are very useful in the case where all the readings for a particular month need to be fetched.
- The disadvantage of key range partitioning is that certain access patterns can lead to hot spots
- If the key is a timestamp, all writes end up going to the same partition (the one for today), so that partition can be overloaded with writes while others are idle
- To avoid this problem, we can use some other attribute other than the timestamp as the first element of the key

4.2-4.3 : Schema Based and Graph Partitioning :

4.2 Schema Based Partitioning

- Schema Partitioning is basically designed to minimize the distributed transactions.
- In this database schema is partitioned in such a way that related rows are kept in the same partition instead of separating them in different partitions.

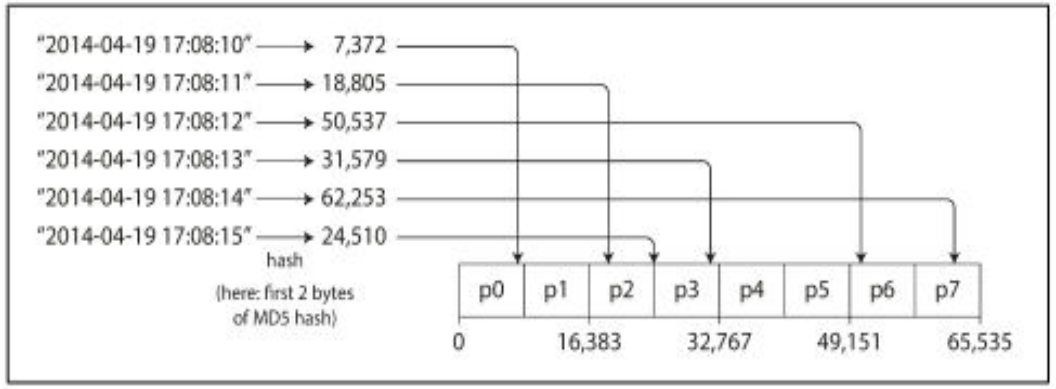
4.3 Graph Partitioning

- Graph partitioning is a workload- based static partition in which partitions are made by analysing the pattern of data access.
- Once partitioning is done, the workload is not observed for changes and there is no-repartitioning
- We could have a workload based dynamic partitioning strategy too .. Which we will discuss as part of Partition rebalancing

CLOUD COMPUTING

4.4 Partitioning by Hash of Key

- Hash partitioning maps data to partitions based on a hashing algorithm that is applied to the partitioning key identified.
- The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size and thus avoids the risk of skew and hot spots
- Hash partitioning is also an easy-to-use alternative to range-partitioning, especially when the data to be partitioned is not historical or has no obvious partitioning key
- Using a suitable hash function for keys, you can assign each partition a range of hashes (rather than a range of keys), and every key whose hash falls within a partition's range will be stored in that partition.



https://notes.shichao.io/dda/ch6/#:~:text=Document%2Dpartitioned%20indexes%20(local%20indexes,scatter%2Fgather%20across%20all%20partitions.

Different Approaches of Partitioning : Partitioning by Hash of Key (Cont.)

- By using the hash of the key for partitioning we lose the ability to do efficient range queries
- Some approaches to circumvent this would be to concatenated index (or a composite key) approach which can enable a one-to-many relationships.

  Example: On a social media site, one user may post many updates. If the primary key for updates is chosen to be (user_id, update_timestamp), then you can efficiently retrieve all updates made by a particular user within some time interval, sorted by timestamp.
- Different users may be stored on different partitions, but within each user, the updates are stored ordered by timestamp on a single partition.

Example :

```
CREATE PARTITION FUNCTION myRangePF1 (int)
   AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
CREATE PARTITION SCHEME myRangePS1
   AS PARTITION myRangePF1
   TO (test1fg, test2fg, test3fg, test4fg);
GO
CREATE TABLE PartitionTable (col1 int, col2 char(10))
   ON myRangePS1 (col1);
GO
```

- Partition test1fg will contain tuples with col1 values <= 1
- Partition test2fg will contain tuples with col1 values > 1 and <= 100
- Partition test3fg will contain tuples with col1 values > 100 and <= 1000
- Partition test4fg will contain tuples with col1 values > 1000

4.4 Distributed Hashing

- The hash table discussed earlier may need to be split into several parts and stored in different servers for working around the memory limitations of a single system to create and keep these large hash tables
- In these environments the data partitions and their keys (the hash table) are distributed among many servers (thus the name of distributed hashing)
- The mechanism for distribution of the keys onto different servers could be a simple hash modulo of the number of servers in the environment. Detailing, for the partition key under consideration, compute the hash and then by using modulo of the number of servers, determine which server the key and the partition will need to be stored in or read from.
- These setups also typically consist of a pool of caching servers that host many key/value pairs and are used to provide fast access to data. If the query for a key for an partition information is not available on the cache, then the data is retrieved from the distributed hashing table as a cache miss.
  So if a client needs to retrieve the partition with key say “Name”, then it looks it up in the cache and gets location of the partition. If the cache entry for “Name” does not exist, then it does a hash and the mod of the number of servers, for getting the information on which server the partition exists and then retrieves the partition information, and also populates it back into the cache for future.

CLOUD COMPUTING

4.4 Distributed Hashing – Rehashing Problem

- This Distributed hashing scheme is simple, intuitive, and works fine .. but may have issues in say the following two scenarios
  - What if we add a few more servers into the pool for scaling
  - What if one of the servers fail
- Keys need to be redistributed to account for the missing server or to account for new servers in the pool
- This is true for any distribution scheme, but the problem with our simple modulo distribution is that when the number of servers changes, most hashes modulo N will change, so most keys will need to be moved to a different server. So, even if a single server is removed or added, all keys will likely need to be rehashed into a different server. Eg. Lets say Server 2 (of 0,1 and 2) failed. The new keys are below. Note that all key locations changed, not only the ones from server C representing 2 in the HASH mod 3.

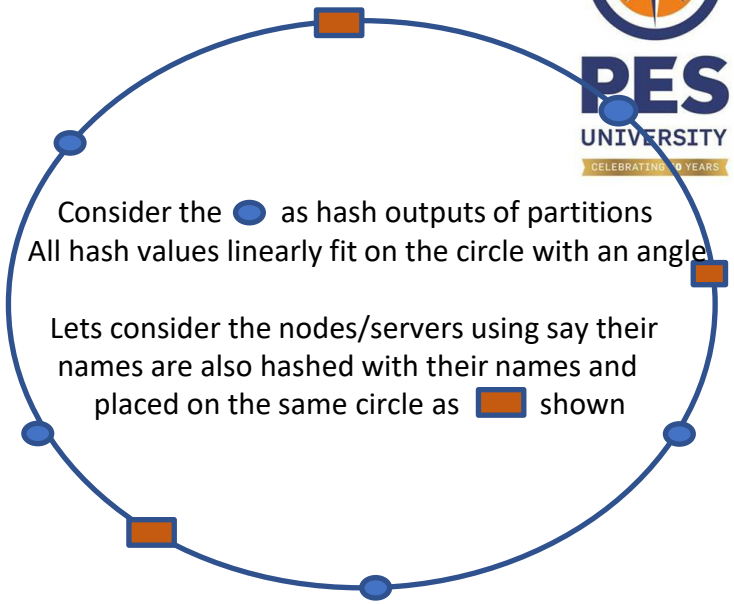
KEY	HASH	HASH mod 3
"john"	1633428562	2
"bill"	7594634739	0
"jane"	5000799124	1
"steve"	9787173343	0
"kate"	3421657995	2

KEY	HASH	HASH mod 2
"john"	1633428562	0
"bill"	7594634739	1
"jane"	5000799124	0
"steve"	9787173343	1
"kate"	3421657995	1

Typical use case discussed earlier with caching, this would mean that, all of a sudden, the keys won't be found because they won't yet be present at their new location.
So, most queries will result in misses, and the original data will likely need retrieving again from the source to be rehashed, thus placing a heavy load on the origin

CLOUD COMPUTING

Consistent Hashing



- Addressing this would need a distribution scheme that does not depend directly on the number of servers, so that, when adding or removing servers, the number of keys that need to be relocated is minimized.
- ***Consistent Hashing*** is a distributed hashing scheme that operates independently of the number of servers or objects in a distributed hash table by assigning them a position on an abstract circle, or hash ring. This allows servers and objects to scale without affecting the overall system.
- Imagine we mapped the hash output range onto the edge of a circle. That means that the minimum possible hash value, zero, would correspond to an angle of zero, the maximum possible value would correspond to an angle of 360 degrees, and all other hash values would linearly fit somewhere in between.
- So, we could take the keys, compute their hash, and place it on the circle's edge. If we now consider the servers too and using their names, compute a hash and place them also on the edge of the circle.
- Since we have the keys for both the objects and the servers on the same circle, we can define a simple rule to associate the former with the latter:
- Each object key will belong in the server whose key is closest, in a counterclockwise direction (or clockwise, depending on the conventions used). Thus to find out which server to ask for a given key, we need to locate the key on the circle and move in the ascending angle direction until we find a server.
- Now if a server is added and placed on the circle, then only those which are closest to it in clockwise direction will need to be moved and the rest of them will not need changes. If a server fails its similar only those behind will need to change.

Partitioning and Secondary Indexes

- We have seen till now that If records are only accessed via their primary key, we can determine the partition from that key, and use it to route read and write requests to the partition responsible for that key.
- A **secondary index** usually doesn't identify a record uniquely but rather is a way of searching for occurrences of a particular value

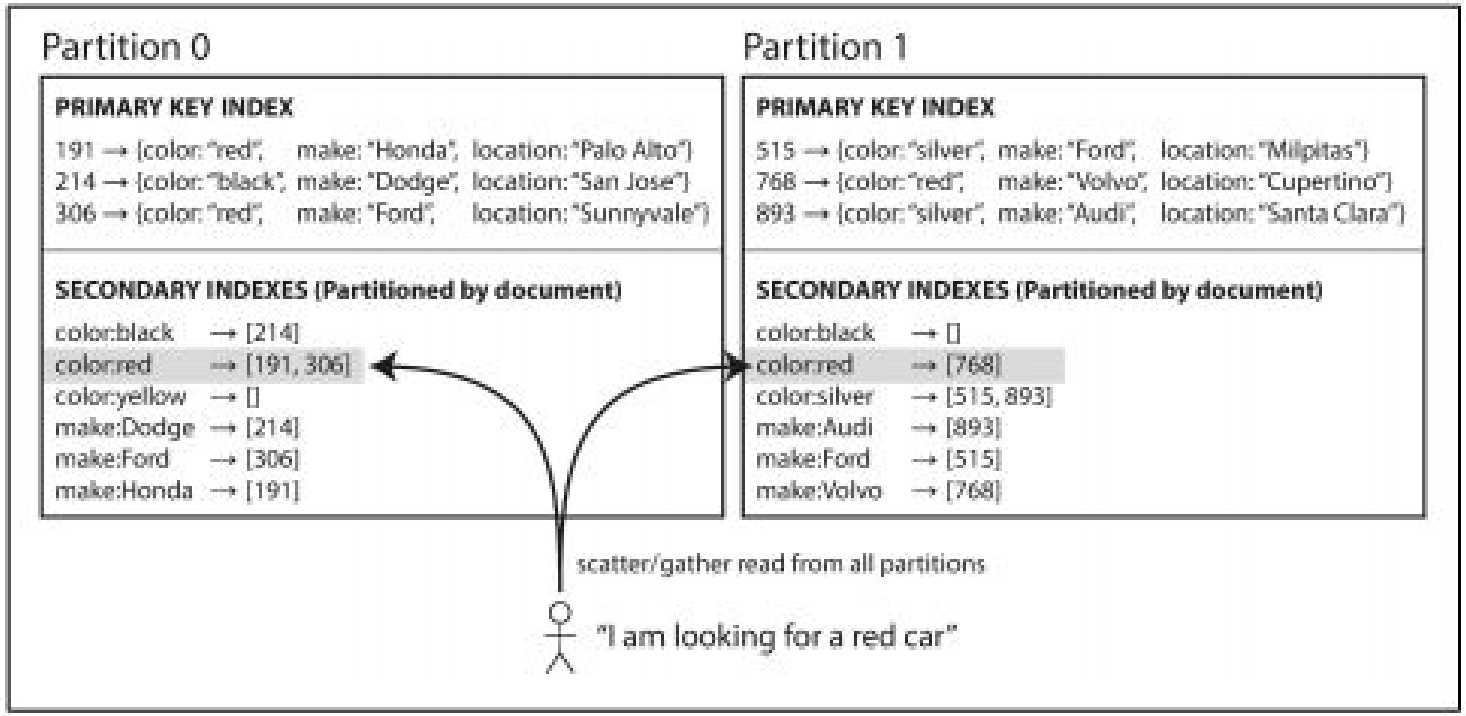
  E.g. find all actions by user 123, find all articles containing the word hogwash,
  find all cars whose color is red, and so on.
- Secondary indexes don't map neatly to partitions.
- Two main approaches to partitioning a database with secondary indexes:
  1. Document-based partitioning
  2. Term-based partitioning

Partitioning with secondary Indexes : Document based Partitioning

- In this indexing approach, each partition is completely separate
- Each partition maintains its own secondary indexes, covering only the documents in that partition. It doesn't care what data is stored in other partitions.
- Whenever you need to write to the database, you only need to deal with the partition that contains the document ID that you are writing. For that reason, a document-partitioned index is also known as a **local index**
- Reading from a document-partitioned index requires all partitions to be queried and combining all the results.
- This approach to querying a partitioned database is sometimes known as **scatter/ gather** and it can make read queries on secondary indexes quite expensive.

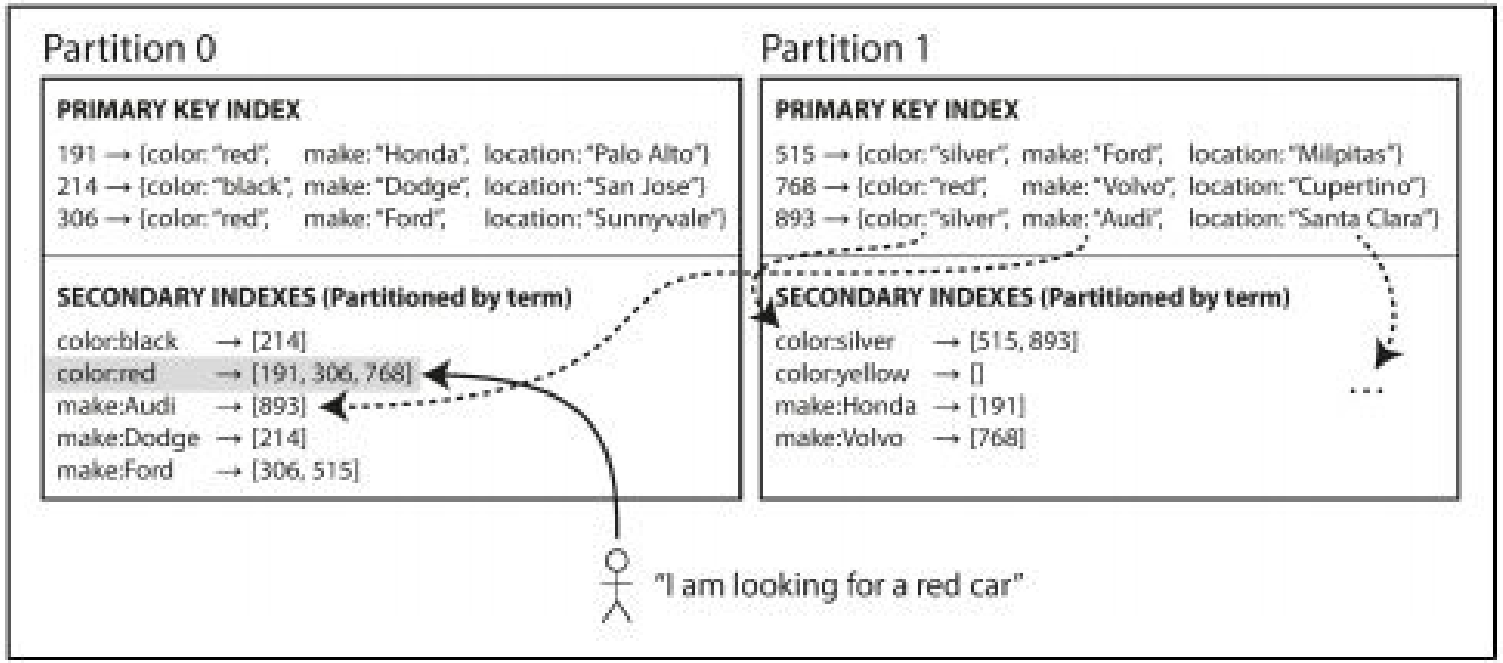
Partitioning with secondary Indexes : Document based Partitioning

- Even if the partitions are queried in parallel, scatter/gather is prone to tail latency amplification
- Most database vendors recommend that you structure your partitioning scheme so that secondary index queries can be served from a single partition, but that is not always possible, especially when we're using multiple secondary indexes in a single query



Partitioning with secondary Indexes : Term based Partitioning

- Construct a global index that covers data in all partitions
- A global index must also be partitioned, but it can be partitioned differently from the primary key index.
- This kind of index is called **term-partitioned**, because the term we're looking for determines the partition of the index.



Partitioning with secondary Indexes : Term based Partitioning

- We can partition the index by the term itself or using a hash of the term.
- Partitioning by the term itself can be useful for range scans, whereas partitioning on a hash of the term gives a more even distribution of load.
- The advantage of a global (term-partitioned) index over a document-partitioned index is that it can make reads more efficient: rather than doing scatter/gather over all partitions, a client only needs to make a request to the partition containing the term that it wants.
- The downside of a global index is that writes are slower and more complicated, because a write to a single document may now affect multiple partitions of the index (every term in the document might be on a different partition on a different node).
- The global index needs to be up to date so that every document written to the database would immediately be reflected in the index. That would require a distributed transaction across all partitions affected by a write, which is not supported in all databases. Typically these updates are often asynchronous
- Therefore, if you read the index shortly after a write, the change you just made may not yet be reflected in the index

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Rebalancing Partitions and Request Routing

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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.

Skewed Workloads and Relieving Hot Spots

- We saw that Hashing a key to determine its partition can help reduce hot spots. However, it can't avoid them entirely
- In the extreme case where all reads and writes are for the same key, all requests will be routed to the same partition.
- Most data systems may not be able to automatically compensate for such a highly skewed workload, and this may be handled by simple techniques like adding a random number to the beginning or end of the key in case of one of the key is known to be very hot or the applications may need to support this. This can address it in a limited fashion and can cause lower performances as for reads data will need to be read from all the keys and combined.
- Distributed hashing discussed earlier, will help in distributing and supporting the throughput, but there will always be scenarios where a workload may need to be moved across nodes.

CLOUD COMPUTING

Rebalancing Partitions

- Since over time, things change like
  - The query throughput in a Database increases, so you want to add more CPUs to handle the load.
  - The dataset size increases, so you want to add a new partition with appropriate capacity
  - A machine fails, and other machines need to take over the failed machine's responsibilities.
  - A Machine joins in .. A new consumer or an recovered machine
- All of these changes call for data and requests to be moved from one node to another. The process of moving load from one node in the cluster to another is called **rebalancing**.
- Rebalancing is expected to meet some minimum requirements regardless of the partitioning scheme:
  - After rebalancing, the load (data storage, read and write requests) should be shared fairly between the nodes in the cluster.
  - While rebalancing is in progress, the database should continue to accept reads and writes.
  - No more data than necessary should be moved between nodes, to make rebalancing fast and to minimize the network and disk I/O load.

https://notes.shichao.io/dda/ch6/#:~:text=Document%2Dpartitioned%20indexes%20(local%20indexes,scatter%2Fgather%20across%20all%20partitions

Strategies for Rebalancing Partitions: Hash based rebalancing : hash mod N

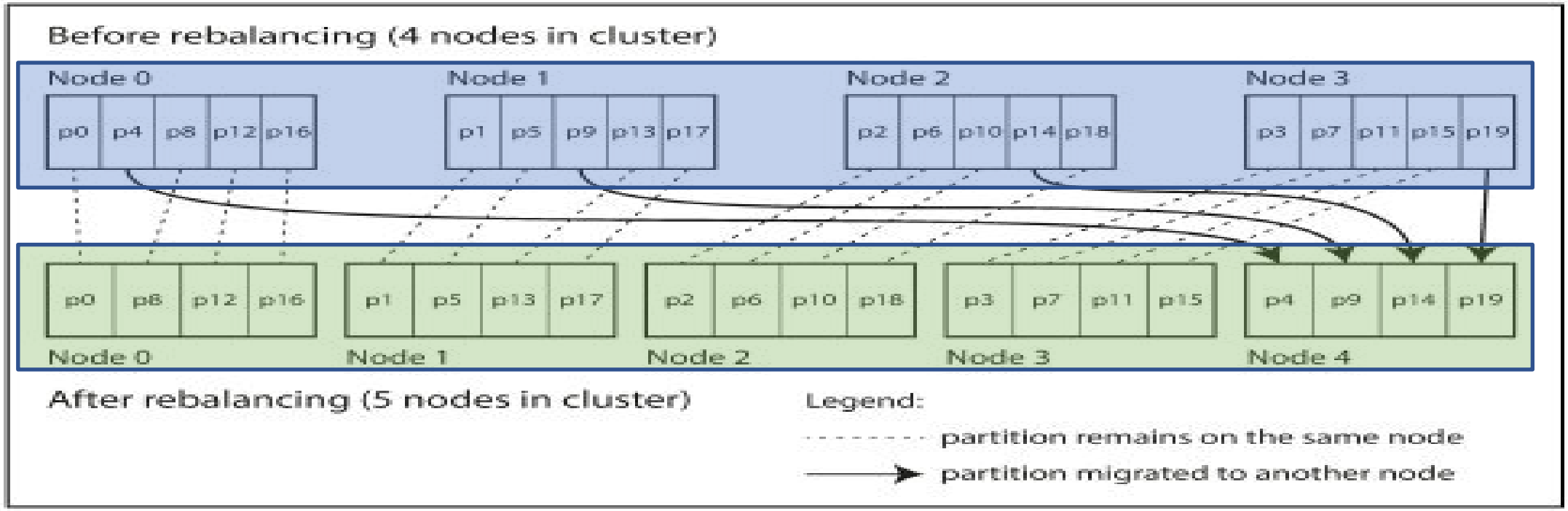
- We had discussed as part of distributed hashing, an approach of distribution of the keys onto different servers using a simple hash modulo of the number of servers in the environment
- So, we partition by the hash of a key, and divide the possible hashes into ranges and assign each range to a partition (e.g., assign key to partition 0 if $0 \leq hash(key) < b0$, to partition 1 if $b0 \leq hash(key) < b1$ etc.) and just use mod (the % operator in many programming languages) to associate a partition to a node.

  E.g. Say if we have 10 nodes (numbered from 0 to 9), hash(key) mod 10 would return a number between 0 and 9 (if we write the hash as a decimal number, the hash mod 10 would be the last digit) which seems like an easy way of assigning each key to a node.
- We also discussed that in this approach if the number of nodes N changes, most of the keys will need to be moved from one node to another.
- So this frequent moves make rebalancing using this approach to be excessively expensive.

Strategies for Rebalancing Partitions : Fixed number of partitions

- Create many more partitions than there are nodes and assign several partitions to each node
- If a node is added to the cluster, the new node can steal a few partitions from every existing node until partitions are fairly distributed once again

  Eg. If there are 04 nodes and 20 partitions. And we add an additional node.



Strategies for Rebalancing Partitions : Fixed number of partitions (Cont.)

- Only entire partitions are moved between nodes.
- The number of partitions does not change, nor does the assignment of keys to partitions. The only thing that changes is the assignment of partitions to nodes
- This change of assignment is not immediate. It takes some time to transfer a large amount of data over the network so the old assignment of partitions is used for any reads and writes that happen while the transfer is in progress.
- Choosing the right number of partitions is difficult if the total size of the dataset is highly variable
- The best performance is achieved when the size of partitions is “just right,” neither too big nor too small, which can be hard to achieve if the number of partitions is fixed but the dataset size varies.
- A fixed number of partitions is operationally simpler. So many fixed-partition databases choose not to implement partition splitting but use fixed number of partitions

Strategies for Rebalancing Partitions : Dynamic Partitioning

- For databases that use key range partitioning , a fixed number of partitions with fixed boundaries would be very inconvenient if the boundaries are wrong. All of the data could end up in one partition and all of the other partitions could remain empty
- Alternatively if the partitions can dynamically created, say when a partition grows to exceed a configured size , it is split into two partitions so that approximately half of the data ends up on each side of the split. One of its two halves can be transferred to another node in order to balance the load.

  If lots of data is deleted & a partition shrinks below some threshold, it can be merged with an adjacent partition
- Advantage of this dynamic partitioning is that the number of partitions adapts to the total data volume
- Each partition is assigned to one node and each node can handle multiple partitions
- In this approach if we start of with a single partition (say like an empty DB), all writes have to be processed by a single node while the other nodes sit idle.
- Dynamic partitioning is not only suitable for key range partitioned data, but can equally well be used with hash partitioned data

Strategies for Rebalancing Partitions : Partitioning proportionally to nodes

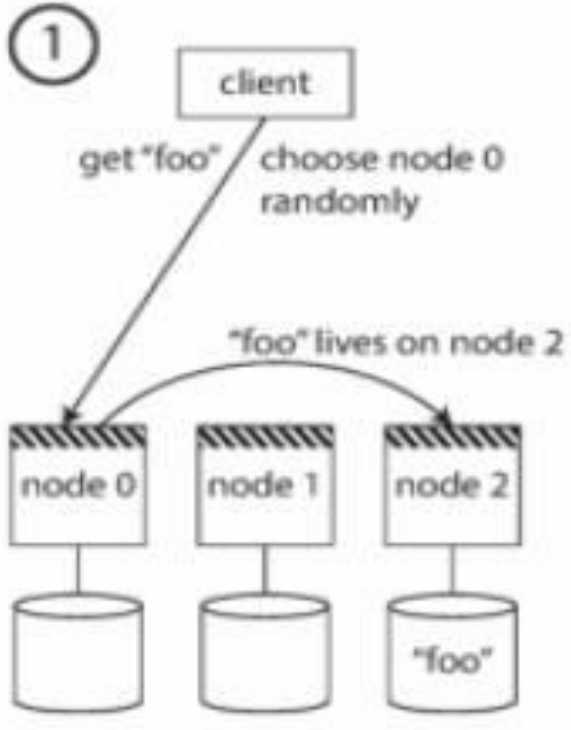
- With *dynamic partitioning, the number of partitions is proportional to the size of the dataset*, since the splitting and merging processes keep the size of each partition between some fixed minimum and maximum
- With a *fixed number of partitions*, the *size of each partition is proportional to the size of the dataset*.
- In both the above cases, the *number of partitions* is *independent of the number of nodes*
- Make the *number of partitions proportional to the number of nodes*—in other words, to have a *fixed number of partitions per node*
- The *size of each partition grows proportionally to the dataset size* while the *number of nodes remains unchanged*, but when you increase the number of nodes, the partitions become smaller again
- Since a larger data volume generally requires a larger number of nodes to store, this approach also keeps the size of each partition fairly stable.

Strategies for Rebalancing Partitions : Partitioning proportionally to nodes

- When a new node joins the cluster, it randomly chooses a fixed number of existing partitions to split, and then takes ownership of one half of each of those split partitions while leaving the other half of each partition in place
- The randomization can produce unfair splits, but when averaged over a larger number of partitions, the new node ends up taking a fair share of the load from the existing nodes.
- Picking partition boundaries randomly requires that hash-based partitioning is used

Request Routing

- Consider a partitioned data distributed across multiple nodes running on multiple machines
- If a client wants to make a request, how does it know which node to connect to?
- As partitions are rebalanced, the assignment of partitions to nodes change. So there is a need to stay on top of those changes in order to provide information like which IP address and port number to connect to.
- This is typically called the service discovery problem which has been addressed with approaches as below.



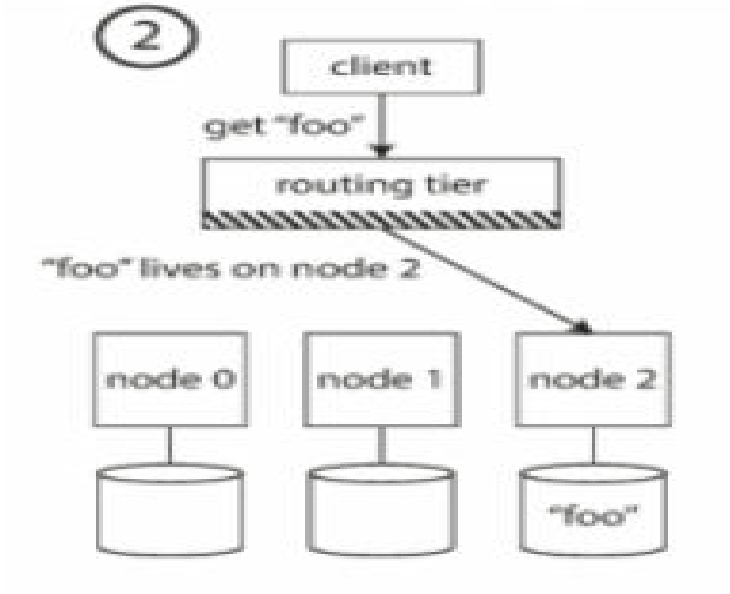
Approach 1

Allow clients to contact any node (e.g., via **a round-robin load balancer**). If that node coincidentally owns the partition to which the request applies, it can handle the request directly; otherwise, it forwards the request to the appropriate node, receives the reply and passes the reply along to the client.

Request Routing

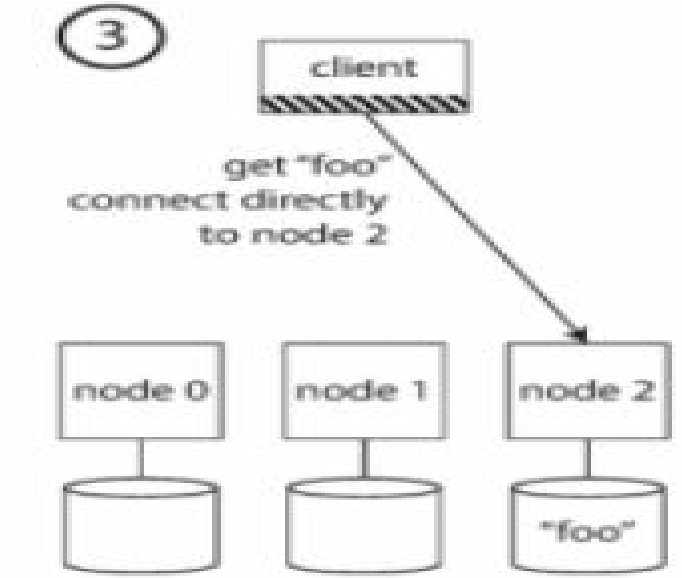
Approach 2

Send all requests from clients to a routing tier first, which determines the node that should handle each request and forwards it accordingly. This routing tier does not itself handle any requests; it only acts as a **partition-aware load balancer**.



Approach 3

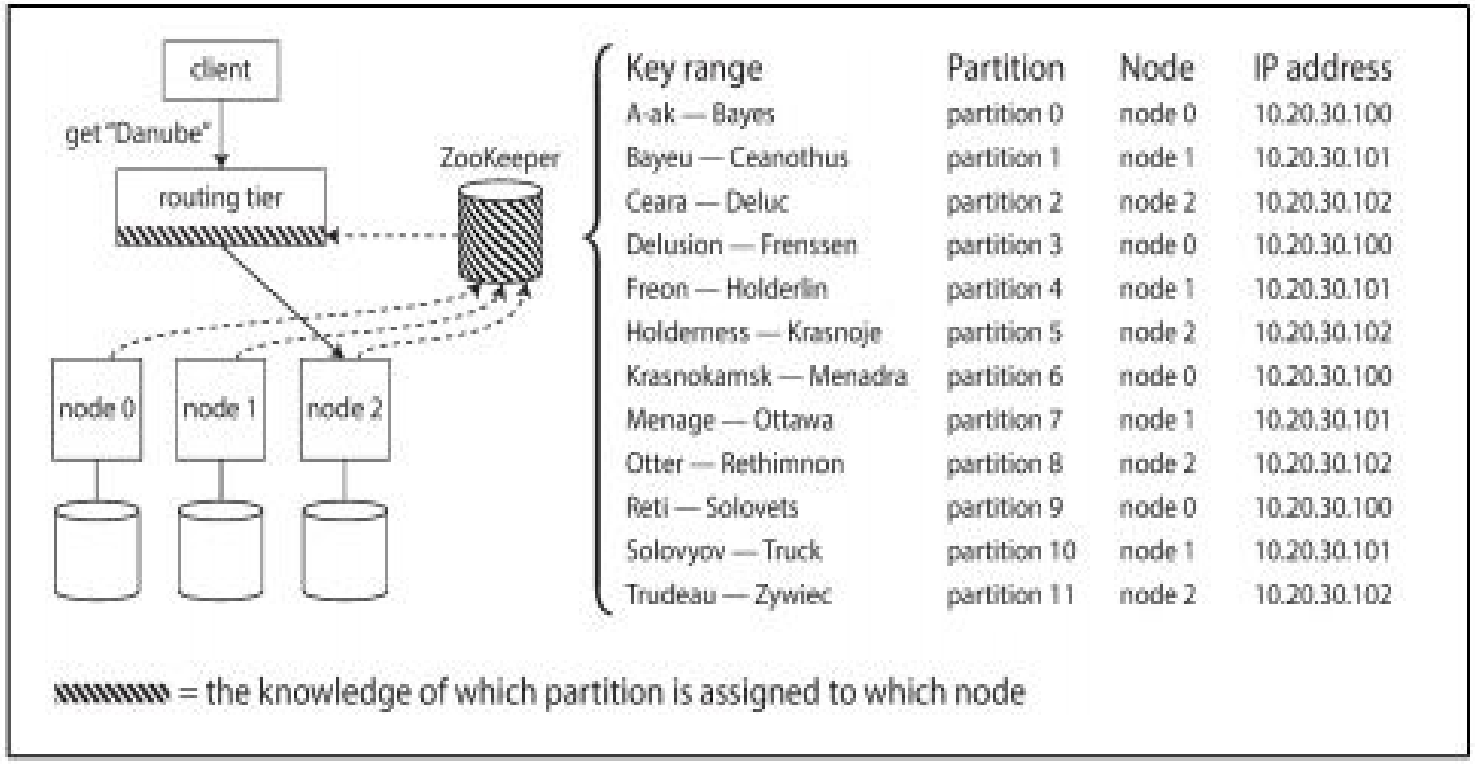
Require that clients be aware of the partitioning and the assignment of partitions to nodes. In this case, a client can connect directly to the appropriate node, without any intermediary



Request Routing

Approach 4 : ZooKeeper

- Many distributed data systems rely on a separate coordination service such as ZooKeeper to keep track of this cluster metadata
- Each node registers itself in ZooKeeper, which maintains the authoritative mapping of partitions to nodes.
- Other actors, such as the routing tier or the partitioning-aware client, can subscribe to this information in ZooKeeper.
- Whenever a partition changes ownership or a node is added or removed, ZooKeeper notifies the routing tier so that it can keep its routing information up to date.



Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

ZooKeeper - A Distributed Coordination Service for Distributed Applications

https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index

- HBase, SolrCloud and Kafka use ZooKeeper to track partition assignment
- LinkedIn's Espresso uses Helix for cluster management (which in turn relies on ZooKeeper), implementing a routing tier
- Cassandra and Riak use a gossip protocol among the nodes to disseminate any changes in cluster state. Requests can be sent to any node, and that node forwards them to the appropriate node for the requested partition

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Replication
Leader Based Replication and
Replication Lag

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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.

Replication

- We looked at partitioning as way of distributed data across different nodes in a cluster, such that query or IO operations can be done across these multiple nodes supporting the performance and throughput expectations of applications.
- **Replication** is a means keeping a copy of the same data on multiple machines that are connected via a network.
- The data and the metadata are replicated for reasons like:
  1. To keep data geographically close to the users (and thus reduce latency)
  2. To allow the system to continue working even if some of its parts have failed (and thus increase availability)
  3. To scale out the number of machines that can serve read queries (and thus increase read throughput)
- This is very beneficial from a performance perspective for read-only data.

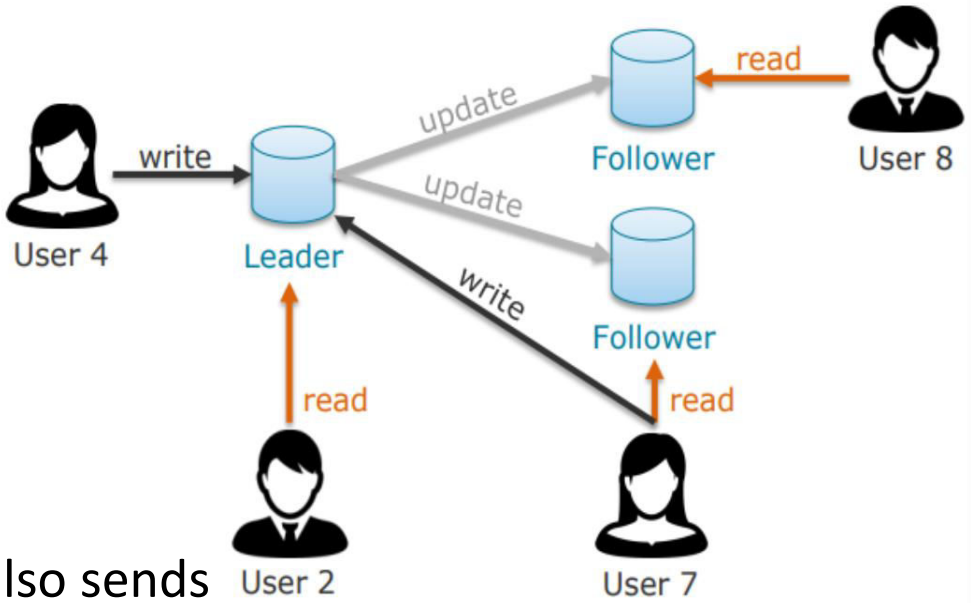
CLOUD COMPUTING

Replication - 2

- Each node that stores a copy of the dataset is called a ***replica***.
- Every write needs to be processed by every replica; otherwise, the nodes will not hold the same data.
- The challenge is how to handle data that changes in a replicated system:
  - Should there be a leader replica and if yes, how many?
  - Should one use a synchronous or asynchronous propagation of the updates among the replicas?
  - How to handle a failed replica if it is the follower? What if the leader failed? How does a resurrection work
- Three popular algorithms for replicating changes between nodes:
  - **Leader based or single-leader based replication**
    - **Synchronous Replication**
    - **Asynchronous Replication**
  - **Multi-leader**
  - **Leaderless** replication

Replication : Leader Based Replication

- Leader based replication is also known as Leader-Follower or master-slave replication
- How do we ensure that all the data is consistent across multiple replicas?
  - One of the replicas is designated the **Leader**.
  - When Users/clients want to write data, they must send their requests to the **leader**, which first writes the new data to its local storage.
  - The other replicas are known as **followers** (read replicas)
  - Whenever the leader writes new data to its local storage, it also sends the data change to all of its followers as part of a replication log
  - Each follower takes the log from the leader and updates its local copy of the data- base accordingly, by applying all writes in the same order as they were processed on the leader.
  - The client can read from anywhere (leader or the followers) but writes are accepted only by the leader
  - Leader-based replication is used in some DBs & distributed message brokers like Kafka and RabbitMQ.



Replication : Single Leader Replication (Rephrased)

Leader

- Dedicated compute node (usually also a replica) responsible for propagating changes
- Also known as master or primary
- Accepts read and write queries
- Sends changes as replication logs to followers

Follower

- General replica
- Also known as slave, secondary, or hot standby
- Accepts only read queries and responds with data from local storage/copy
- Receives changes from leader(s) and updates local copy accordingly:
- Apply all writes in the same order as applied on the leader

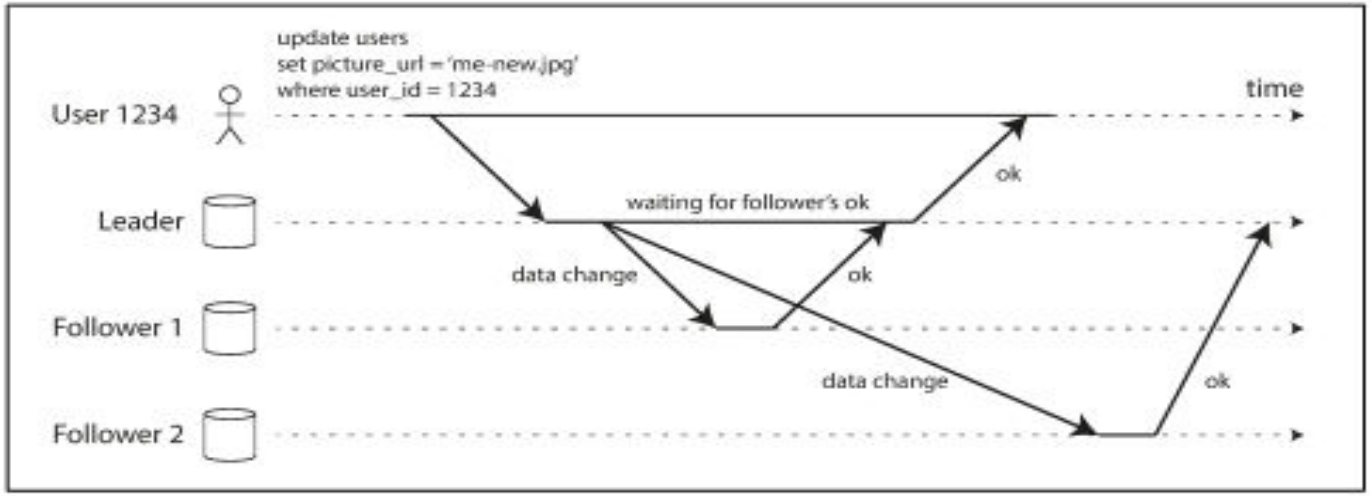
Replication : Leader Based Replication : Synchronous Versus Asynchronous Replication

- In **synchronous** replication, the leader waits until followers have confirmed that it received the write before reporting success to the user and before making the write visible to other clients.

  E.g. the replication to follower 1 is synchronous

- In **asynchronous** replication, the leader sends the message to its follower (s) but doesn't wait for a response from the followers before answering success to the User

  E.g. The replication to follower 2 is asynchronous



Leader based replication with one synchronous & one asynchronous follower

Replication : Leader Based Replication : Implementation of Replication Logs

1. **Statement based replication -** The leader logs every write request that it executes and sends that statement log to its followers.
2. **Write-ahead log (WAL) shipping -** The log is an append-only sequence of bytes containing all writes to the database. The leader writes the log to disk and sends it across the network to its followers. Then the leader updates the data (say the DB) and the followers processes also processes this log and make the changes to the database and thus they builds copy of the exact same data structures as found on the leader.
3. **Change data capture (CDC) based replication - Logical log replication –** Sequence of records that describe the write to database tables at the granularity of rows. Its based on the identification, capture and delivery of the changes made. Replicas can run on different versions or storage engines but use different log formats for different storage engines. Its also easier to parse for external applications. These logs are also called a **logical log**. A logical log for a relational database is usually a sequence of records describing writes to database tables at the granularity of a row.
4. **Trigger based replication (application layer) -** A trigger lets users register custom application code that is automatically executed when a data change (write transaction) occurs in a database system. The trigger has the opportunity to log this change which can be read by an external process. The external process can then apply any necessary application logic and replicate the data change to another system.

Follower Failure: Catch-up recovery

- On its local disk, each follower keeps a log of the data changes it has received from the leader.
- If a follower crashes and is restarted, or if the network between the leader and the follower is temporarily interrupted, the follower can recover quite easily from its log
- Follower knows the last transaction that was processed before the fault occurred. Thus, the follower can connect to the leader and request all the data changes that occurred during the time when the follower was disconnected.
- When the follower has applied these changes, it has caught up to the leader and can continue receiving a stream of data changes as before

Leader Failure - Failover

- One of the followers needs to be promoted to be the new leader
- Clients need to be reconfigured to send their writes to the new leader and the other followers need to start consuming data changes from the new leader.
- Failover can happen manually (an administrator is notified that the leader has failed and takes the necessary steps to make a new leader) or automatically
- Steps followed in an automatic failover process:
  1. Determining that the leader has failed.
  2. Choosing a new leader
  3. Reconfiguring the system to use the new leader

Replication : Leader Based Replication : Potential Issues and their Handling

Replication Lag

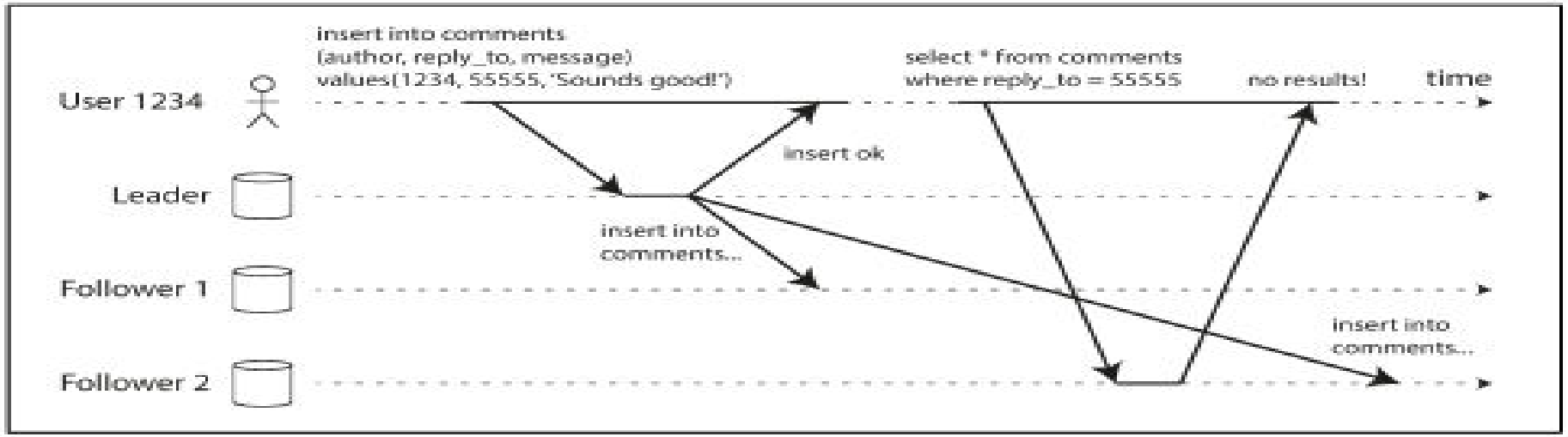
- In Leader-based replication all writes go to the leader, but read-only queries can go to any replica.
- This makes it attractive also for scalability and latency, in addition to fault-tolerance.
- For read-mainly workloads: have many followers and distribute the reads across those followers.
  - **Removes load from the leader, allows read requests to be served by nearby replicas**.
  - But, only realistic for asynchronous replication otherwise the system will not be available
- If an **application reads from an asynchronous follower, it may see outdated information if the follower has fallen behind.**
- This leads to apparent inconsistencies in the database: if you run the same query on the leader and a follower at the same time, you may get different results, because not all writes would have been reflected in the follower.

Replication Lag (Continued)

- This inconsistency is just temporary—**if there are no writes to the database in a while, the followers will eventually catch up and become consistent with the leader. This effect is known as eventual consistency**
- In normal operation, **the delay between a write happening on the leader and the same being reflected on a follower is known as the *replication lag*.** This may be only a fraction of a second and not noticeable in practice
- When the lag is large, the inconsistencies it introduces are not just a theoretical issue but a real problem for applications.

Identification of Inconsistencies due to Replication Lag

Reading Your Own Writes



- Reading your Own writes will help identify that if the user re-reads the data, they will always see any updates they submitted themselves.

There are different models for Consistency like the one above, Read-after-write consistency, different eventual consistency models which will bring in consistency.

Some possible solutions for Replication Lag:

- A simple rule: always read critical data from the leader and rest from a follower (negates the benefit of read scaling)
- Monitor the replication lag on followers and prevent queries on any follower with significant lag behind the leader.
- The client can remember the timestamp of its most recent write—then the system can ensure that the replica serving any reads for that user reflects updates at least until that timestamp
- Monotonic reads - make sure that each user always makes their reads from the same replica
- Consistent prefix reads - if a sequence of writes happen in a certain order, then anyone reading those writes should see them appear in the same order

Multi-Leader Replication

Recap: Replication *is a means keeping a copy of the same data on multiple machines that are connected via a network. We discussed that there were three popular algorithms for replicating changes between nodes: single-leader, multi-leader, and leaderless replication.*

- *We discussed* ***Leader Based Replication*** *earlier.*
- Leader-based replication has a single bottleneck in the leader
- All writes must go through it. If there is a network interruption between the user and the leader, then no writes are allowed.
- An alternate approach to consider is , what if we have more than one leader through whom you can do the writes.

Multi-Leader Replication

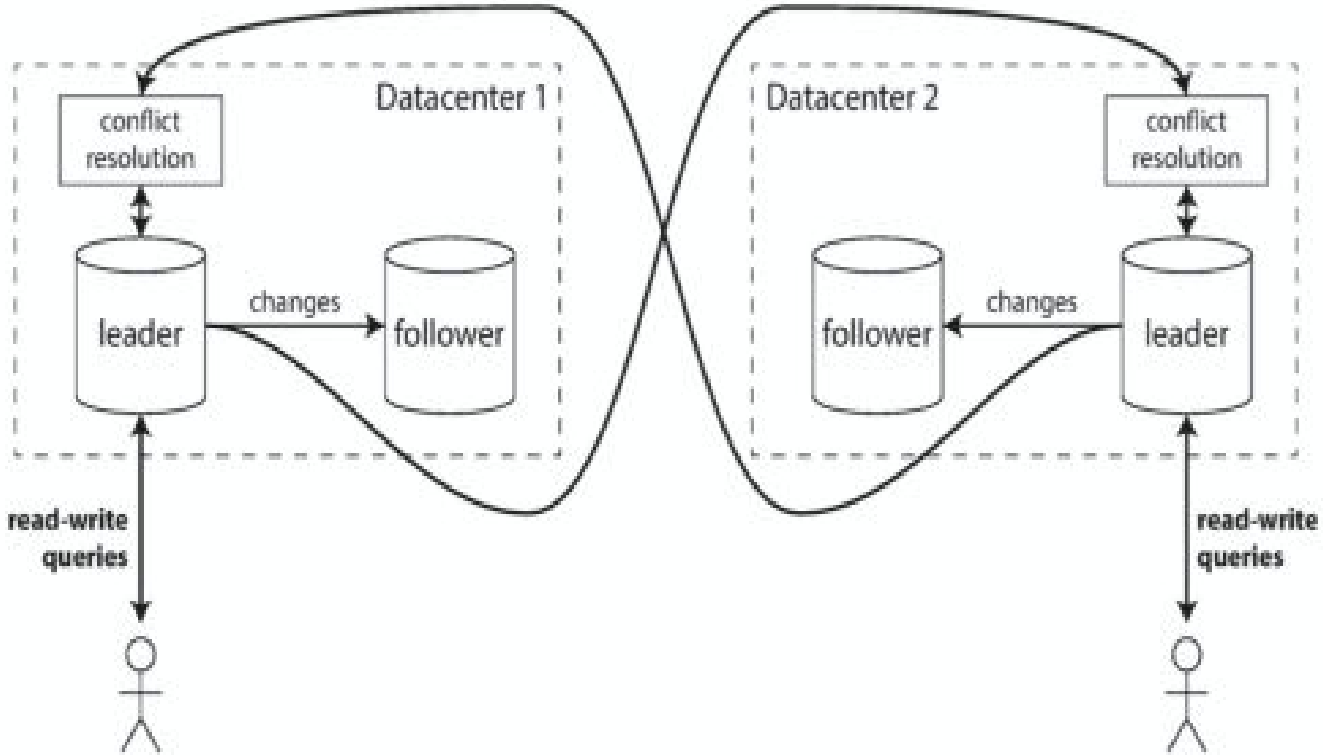
So with this natural extension of the leader-based replication model of allowing more than one node to accept writes will lead to

- Each node that processes a write must forward that data change to all the other nodes
- This approach is also known as master/master or active/active replication
- Each leader simultaneously acts as a follower to the other leaders.

This would have different Use cases like

- **Multi-Datacenter operation** can have a leader in each datacenter. Each datacenter has a regular leader– follower replication and between datacenters, each datacenter's leader replicates its changes to the leaders in other datacenters.
- **Clients with offline operation**

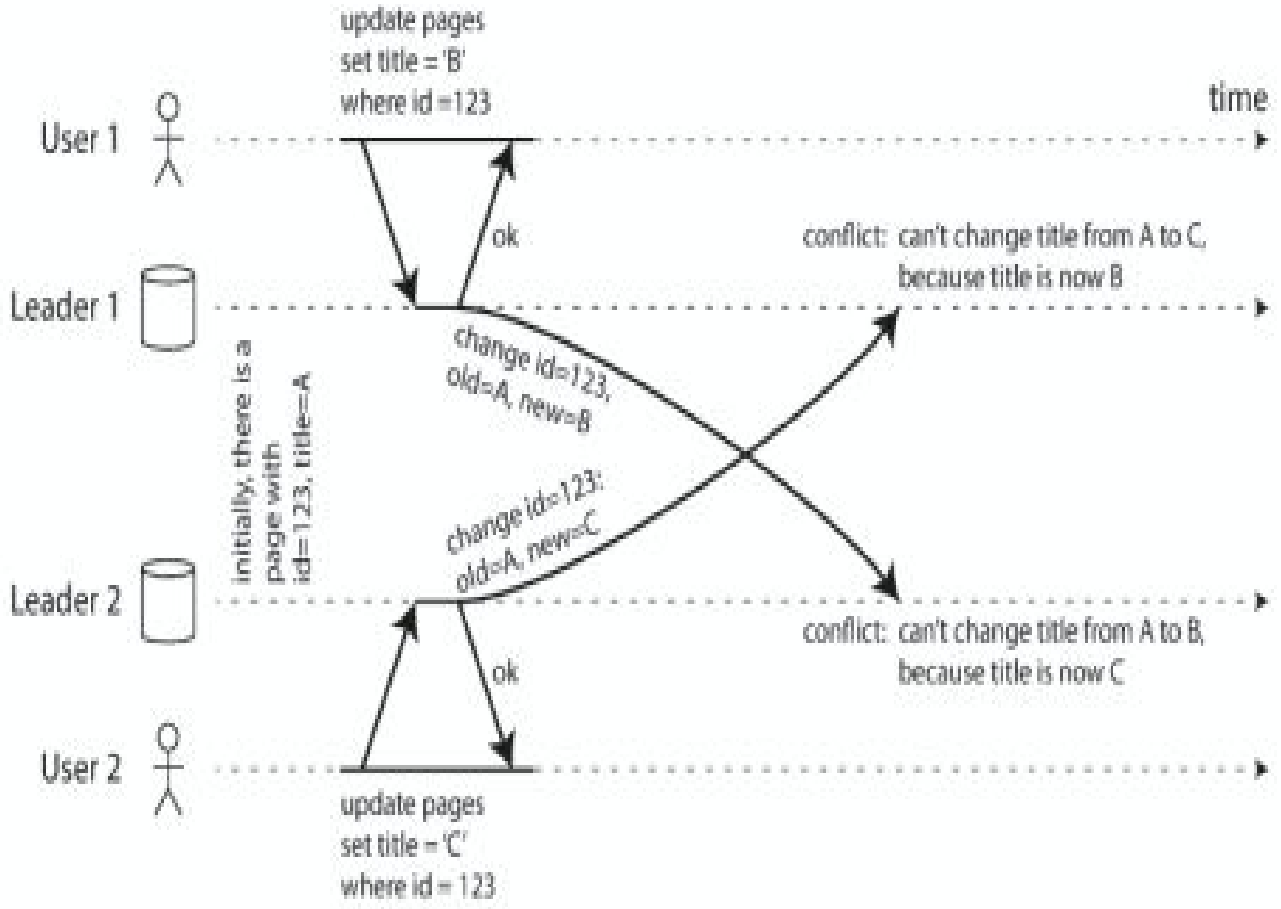
  Every device has a local database that acts as a leader
- **Collaborative editing**



Multi-Leader Replication Challenges

A problem with multi-leader replication is that it can lead to write conflicts

- **Conflict avoidance** - ensure that all writes for a particular record go through the same leader
- **Converging toward a consistent state** - Give each write a unique ID (e.g., a timestamp, a long random number, a UUID, or a hash of the key and value), pick the write with the highest ID as the winner and throw away the other writes.
- **Custom conflict resolution logic -** Write conflict resolution logic in application code. That code may be executed on write or on read



Single-Leader vs. Multi-Leader Replication

Performance

- In a single-leader configuration, every write must go over the internet to the datacenter with the leader. This can add significant latency to writes and might contravene the purpose of having multiple datacenters in the first place.
- In a multi-leader configuration, every write can be processed in the local datacenter and is replicated asynchronously to the other datacenters. Thus, the inter-datacenter network delay is hidden from users, which means the perceived performance may be better.

Tolerance of datacenter outages

- In a single-leader configuration, if the datacenter with the leader fails, failover can promote a follower in another datacenter to be leader.
- In a multi-leader configuration, each datacenter can continue operating independently of the others, and replication catches up when the failed datacenter comes back online.

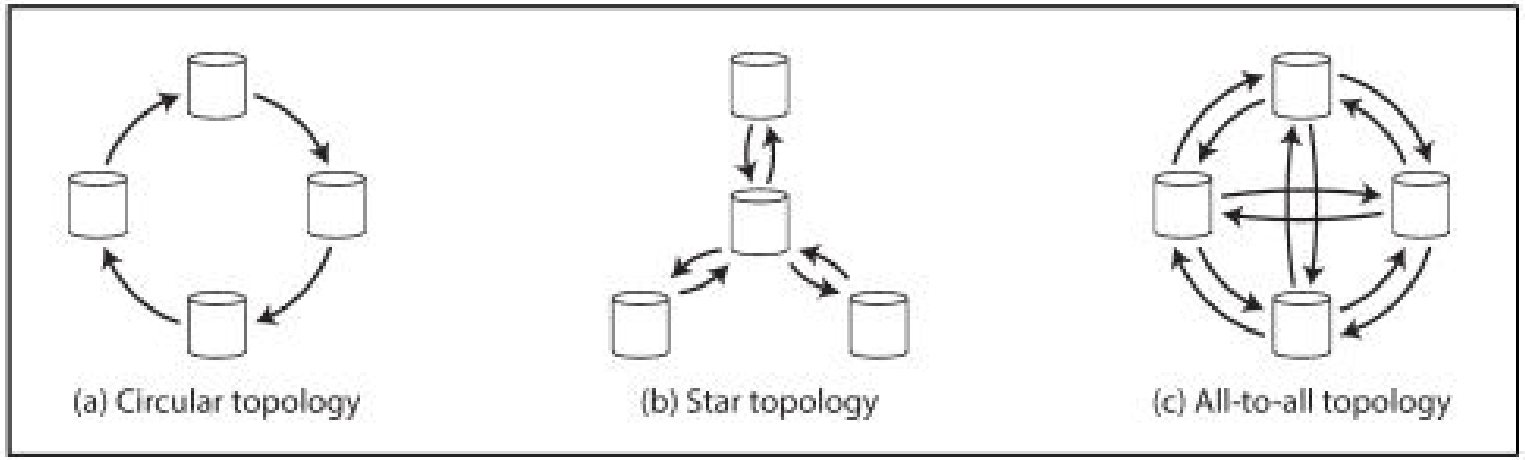
Single-Leader vs. Multi-Leader Replication

Tolerance of network problems

- Traffic between datacenters usually goes over the public internet, which may be less reliable than the local network within a datacenter.
- A single-leader configuration is very sensitive to problems in this inter-datacenter link, because writes are made synchronously over this link.
- A multi-leader configuration with asynchronous replication can usually tolerate network problems better: a temporary network interruption does not prevent writes being processed.

Multi-Leader Replication Topologies

- A replication topology describes the communication paths along which writes are propagated from one node to another.



- **Circular topology** - each node receives writes from one node and forwards those writes (plus any writes of its own) to one other node
- **Star topology** - one designated root node forwards writes to all of the other nodes.
- **All-to-all topology** - allows messages to travel along different paths, avoiding a single point of failure.

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Multi Leader Replication

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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.

Multi-Leader Replication

Recap: Replication *is a means keeping a copy of the same data on multiple machines that are connected via a network. We discussed that there were three popular algorithms for replicating changes between nodes: single-leader, multi-leader, and leaderless replication.*

- *We discussed* ***Leader Based Replication*** *earlier.*
- Leader-based replication has a single bottleneck in the leader
- All writes must go through it. **If there is a network interruption between the user and the leader, then no writes are allowed.**
- An alternate approach to consider is , what if we have more than one leader through whom you can do the writes.

CLOUD COMPUTING

Multi-Leader Replication

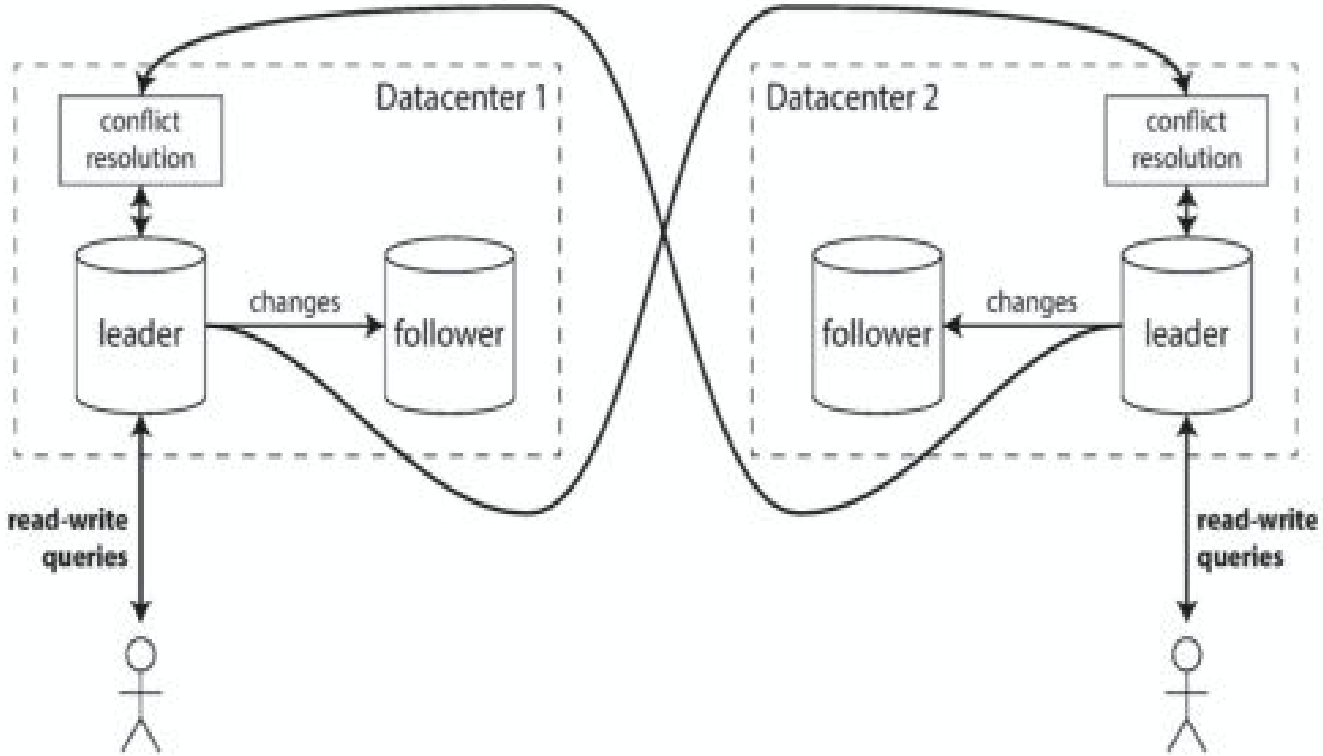
So with this natural extension of the leader-based replication model of allowing more than one node to accept writes will lead to

- **Each node that processes a write must forward that data change to all the other nodes**
- This approach is also known as master/master or active/active replication
- **Each leader simultaneously acts as a follower to the other leaders**.

This would have different Use cases like

- **Multi-Datacenter operation** can have a leader in each datacenter. Each datacenter has a regular leader– follower replication and between datacenters, each datacenter's leader replicates its changes to the leaders in other datacenters.
- **Clients with offline operation**

  Every device has a local database that acts as a leader
- **Collaborative editing**

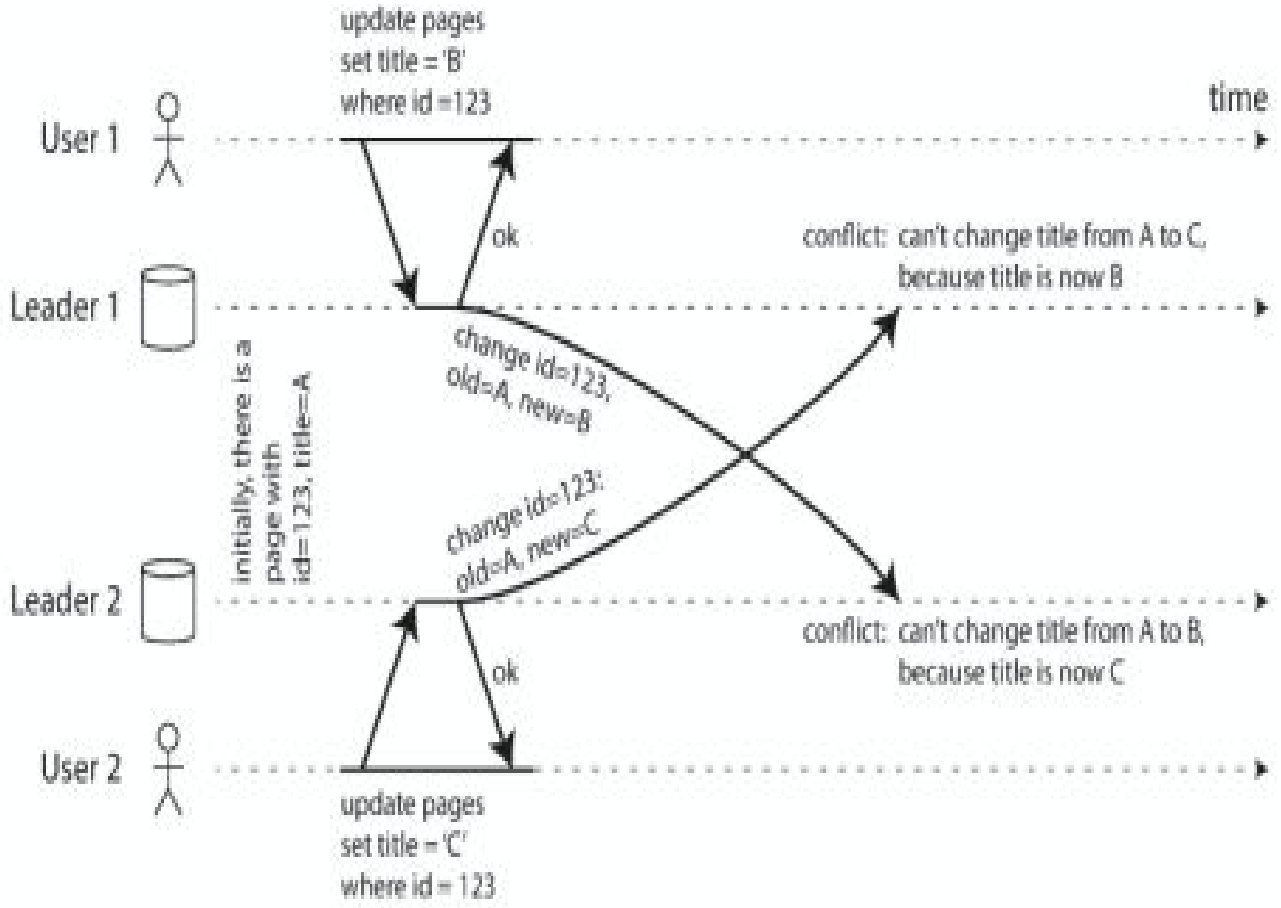


CLOUD COMPUTING

Multi-Leader Replication Challenges

A problem with multi-leader replication is that it can lead to write conflicts

- **Conflict avoidance** - ensure that all writes for a particular record go through the same leader
- **Converging toward a consistent state - Give each write a unique ID** (e.g., a timestamp, a long random number, a UUID, or a hash of the key and value), **pick the write with the highest ID as the winner and throw away the other writes**.
- **Custom conflict resolution logic** - Write conflict resolution logic in application code. That code may be executed on write or on read



Single-Leader vs. Multi-Leader Replication

Performance

- In a **single-leader configuration, every write must go over the internet to the datacenter with the leader**. This can add significant latency to writes and might contravene the purpose of having multiple datacenters in the first place.
- In a multi-leader configuration**, every write can be processed in the local datacenter and is replicated asynchronously to the other datacenters**. Thus, the inter-datacenter network delay is hidden from users, which means the perceived performance may be better.

Tolerance of datacenter outages

- In a single-leader configuration, if the datacenter with the leader fails, failover can promote a follower in another datacenter to be leader.
- In a multi-leader configuration, **each datacenter can continue operating independently of the others, and replication catches up when the failed datacenter comes back online**.

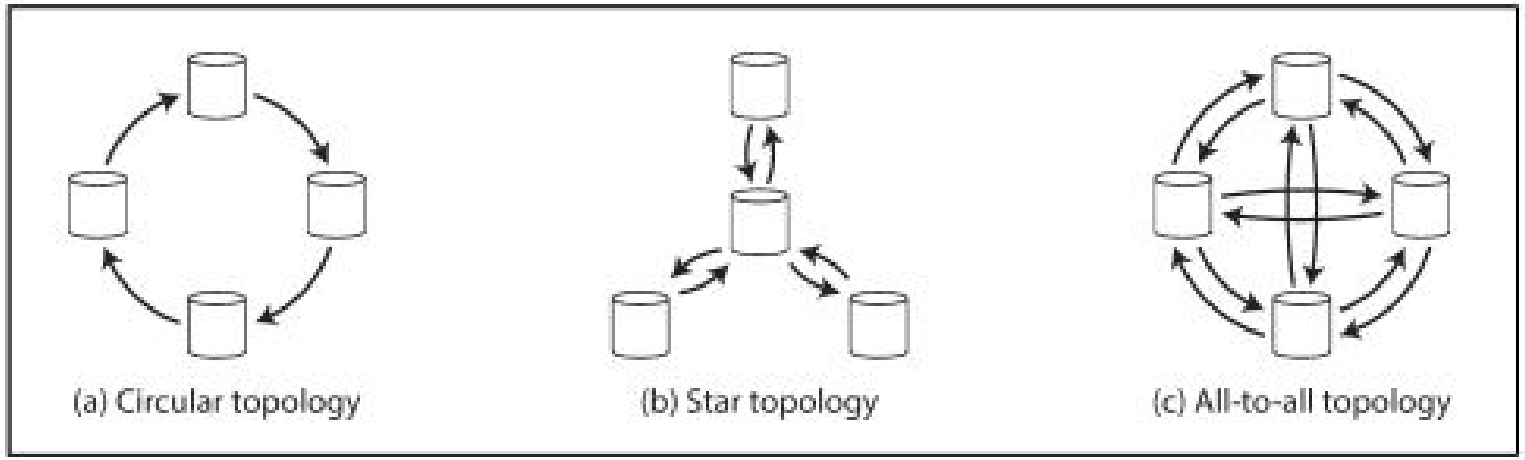
Single-Leader vs. Multi-Leader Replication

Tolerance of network problems

- Traffic between datacenters usually goes **over the public internet, which may be less reliable than the local network within a datacenter.**
- A **single-leader configuration is very sensitive to problems in this inter-datacenter link**, because writes are made synchronously over this link.
- A multi-leader configuration with asynchronous replication can usually tolerate network problems better: a temporary network interruption does not prevent writes being processed.

Multi-Leader Replication Topologies

- A replication topology describes the communication paths along which writes are propagated from one node to another.



- **Circular topology** - each node receives writes from one node and forwards those writes (plus any writes of its own) to one other node
- **Star topology** - one designated root node forwards writes to all of the other nodes.
- **All-to-all topology** - allows messages to travel along different paths, avoiding a single point of failure.

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Leaderless Replication

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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

Replication (Recap)

- We looked at **Replication** as a means of keeping a copy of the same data on multiple machines that are connected via a network.
- The data and the metadata are replicated for reasons like:
  1. To keep data geographically close to the users (and thus reduce latency)
  2. To allow the system to continue working even if some of its parts have failed (and thus increase availability)
  3. To scale out the number of machines that can serve read queries (and thus increase read throughput)
- We discussed that every write needs to be processed by every replica; otherwise, the nodes will not hold the same data.
- We discussed on the two of the three popular algorithms for replicating changes between nodes:
  - Leader based or single-leader based replication
    - Synchronous Replication
    - Asynchronous Replication
  - Multi-leader
  - Leaderless replication

CLOUD COMPUTING

Replication (Recap – 2)

- As part of the Leader based replication, we discussed one of the replicas would be designated as a leader and typically all Users/clients wanting to write data to storage, would send their requests to the leader, which first writes the new data to its local storage. The leader then sends the data change to all of the other replicas known as followers using some kind of replication log. These followers are also call read replicas.
- We also discussed that potential issues like the leader failure, follower failure or the **replication lag** which is the lag between the write happening on the leader and getting replicated on the follower or that eventually up-dation would happen leading to eventual concurrency across the nodes. We also discussed on simple techniques which were used to address the same.
- We also saw that **a single leader was a bottleneck for the environment and looked at having multiple leaders**. We saw the advantages it would provide in scenarios like data centers, we saw the challenges it would bring in like synchronization and looked at a few simple approaches which could be used to address the same.
- We also contrasted the single leader and multi-leader replications in terms of performance, tolerance to data center failures, network failures

CLOUD COMPUTING

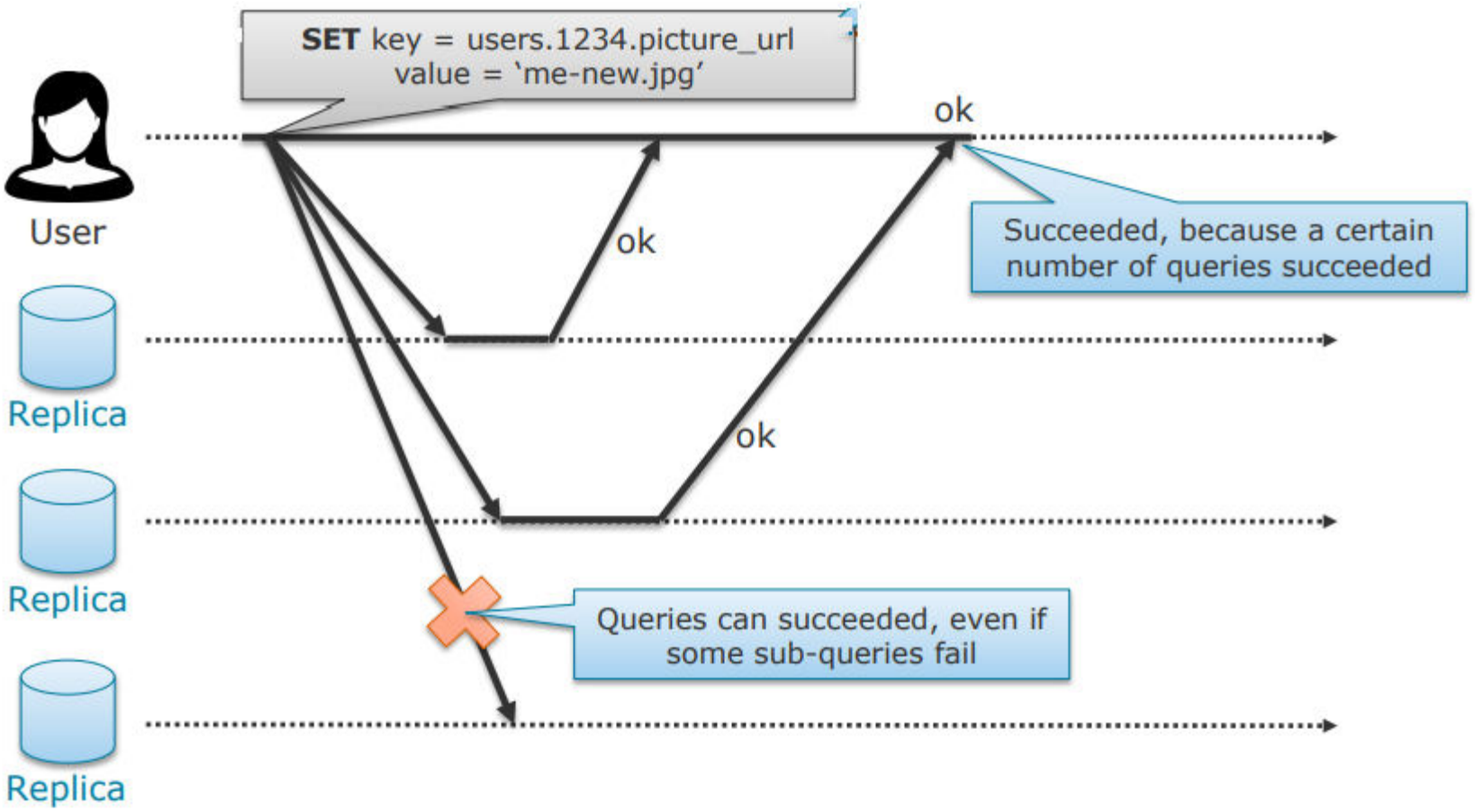
Leaderless Replication

- In contrast to the Single Leader and Multileader replication strategies, Leaderless replication adopts a philosophy that **Nodes in the leaderless setting are considered peers**
- All the nodes accept writes & reads from the client.
- Without the bottleneck of a leader that handles all write requests, leaderless replication offers better availability.
- A leaderless replication is an **architecture where every write must be sent to every replica.**
- A write is considered to be successful when the write is acknowledged by (a quorum of) at least k out of n replicas and the read is considered successful in reading a particular value, when (a quorum of) at least k out of n reads must agree on a value
- It has the advantage of parallel writes

Leaderless Replication

Writes with Leaderless Replication

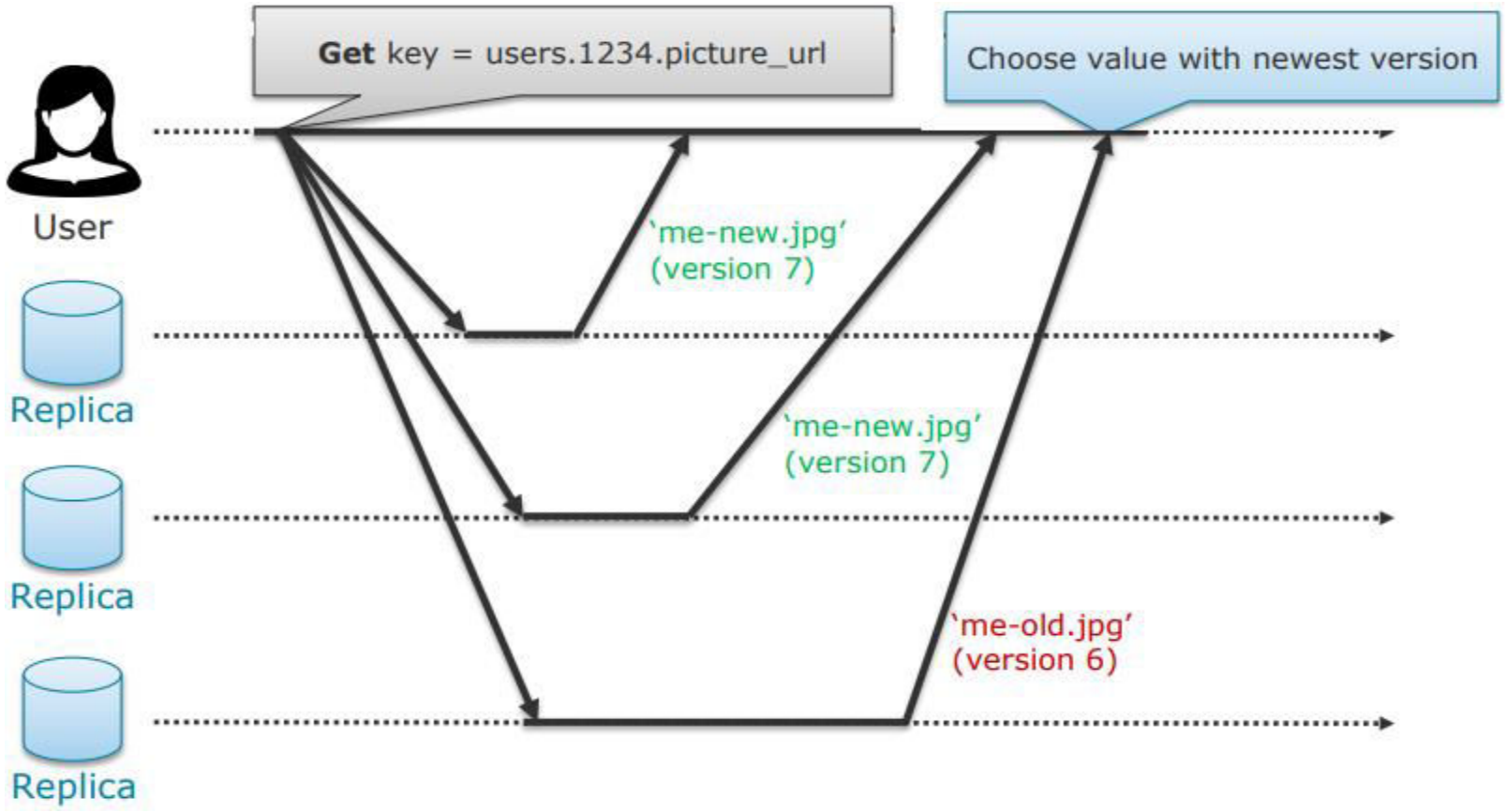
- Upon a **write**, the client broadcasts the request to all replicas instead of a special node (the leader) and waits for a certain number of ACKs.
- The write is completed and considered successful when the write must be acknowledged by at least k out of n replicas. (E.g. in the figure when two of the three replicas send ACK to the client)



Leaderless Replication

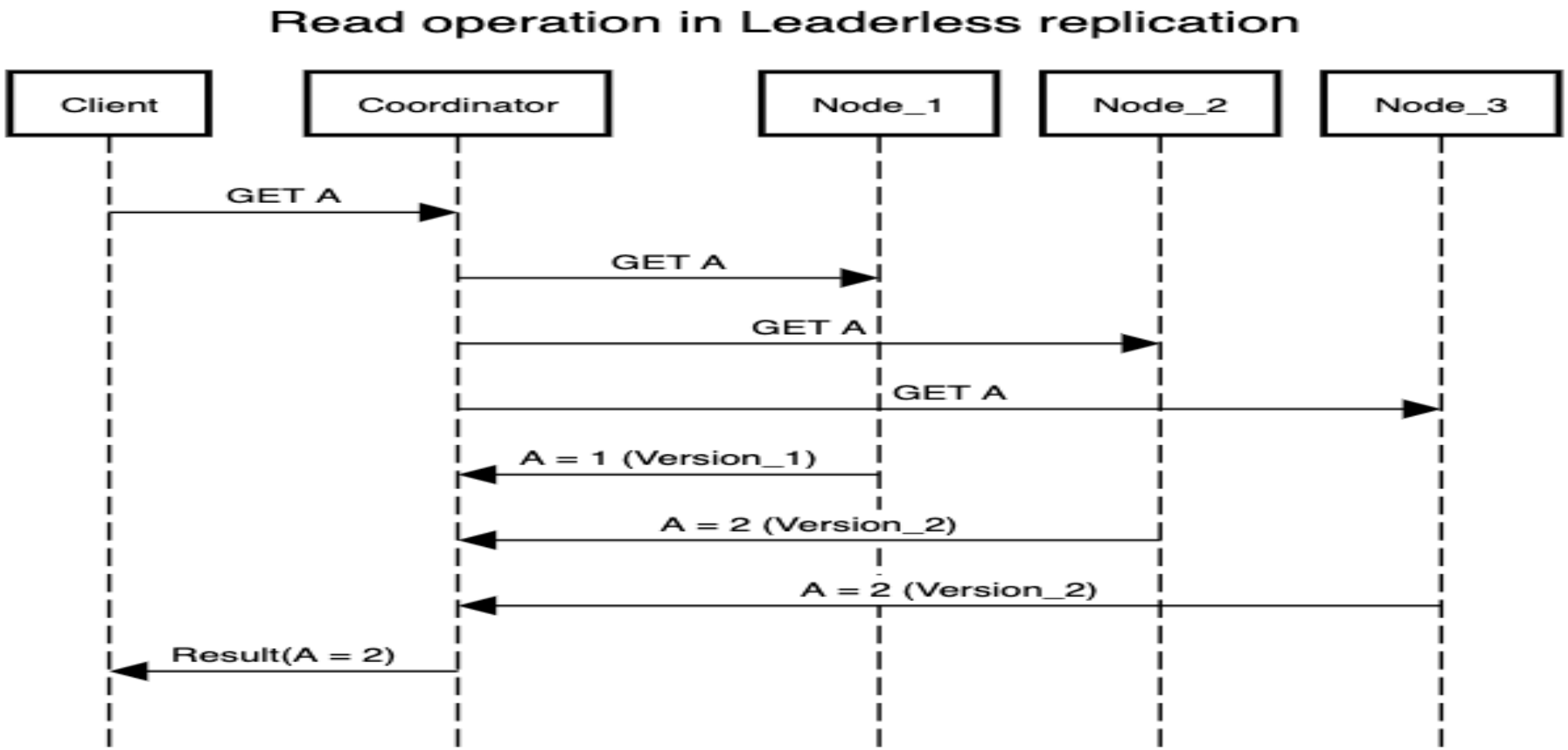
Reads with Leaderless Replication

- Upon read, the client contacts all replicas & waits for some number of responses.
- **In order to be considered successful in reading a particular value, at least k out of n reads must agree on a value. E.g read request is completed when two out of the three replicas returns the latest version.**
- This approach of the client waiting for many (quorum of) responses, the approach is known as quorum or the value k is known as quorum.
- k value when you are reading is known as read quorum and k value when you are writing is known as write quorum
- This quorum can be configured in a way to provide consistency to the copies.



Leaderless Replication

Reads with Leaderless Replication (another example)



Leaderless Replication : Potential Pitfalls - 1

Writing to the Database When a Node Is Down

- Consider three replicas with one of the replicas being currently unavailable.
- The client sends the write to all three replicas in parallel, and the two available replicas accept the write but the unavailable replica misses it.
- We consider **the write to be successful since two out of three replicas have acknowledged the write**
- The client simply ignores the fact that one of the replicas missed the write
- Suppose the unavailable node comes back online and clients start reading from it. Any writes that happened while the node was down will be missing from that node. Thus, stale (outdated) values may be read from that node as response.

An approach to address the same

- To solve this problem, read requests are also sent to several nodes in parallel.
- The client may get different responses from different nodes; i.e., **the up-to-date value from one node and a stale value from another.**
- **Version numbers can be used to determine which value is newer**

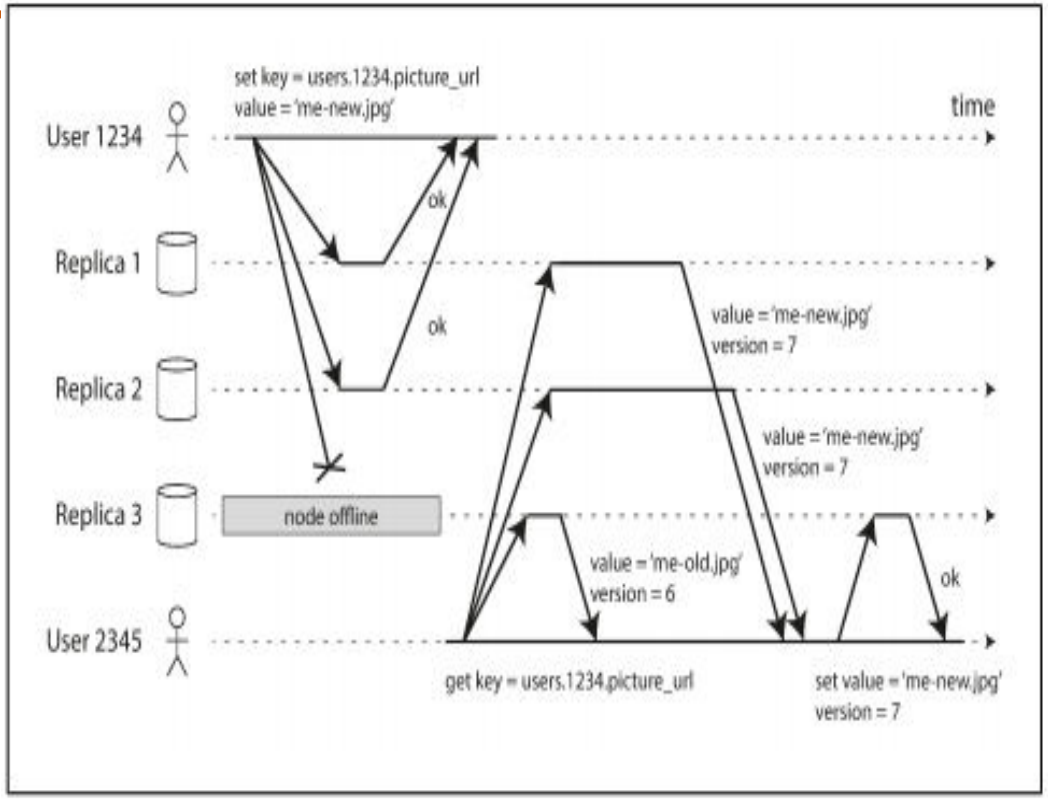
CLOUD COMPUTING

Leaderless Replication – Pitfalls - 2

How does a node catch up on the writes that it missed?

Read repair

- When a client makes a read from several nodes in parallel, it can detect any stale responses. User 2345 gets a version 6 value from replica 3 and a version 7 value from replicas 1 and 2.
- The client sees that replica 3 has a stale value and writes the newer value back to that replica.



Anti-entropy process

- Some datastores have a background process that constantly looks for differences in the data between replicas and copies any missing data from one replica to another
- There are approaches where there are agents run on the replica which periodically match their states and actively propagate the same (Using Gossip kind of approach)

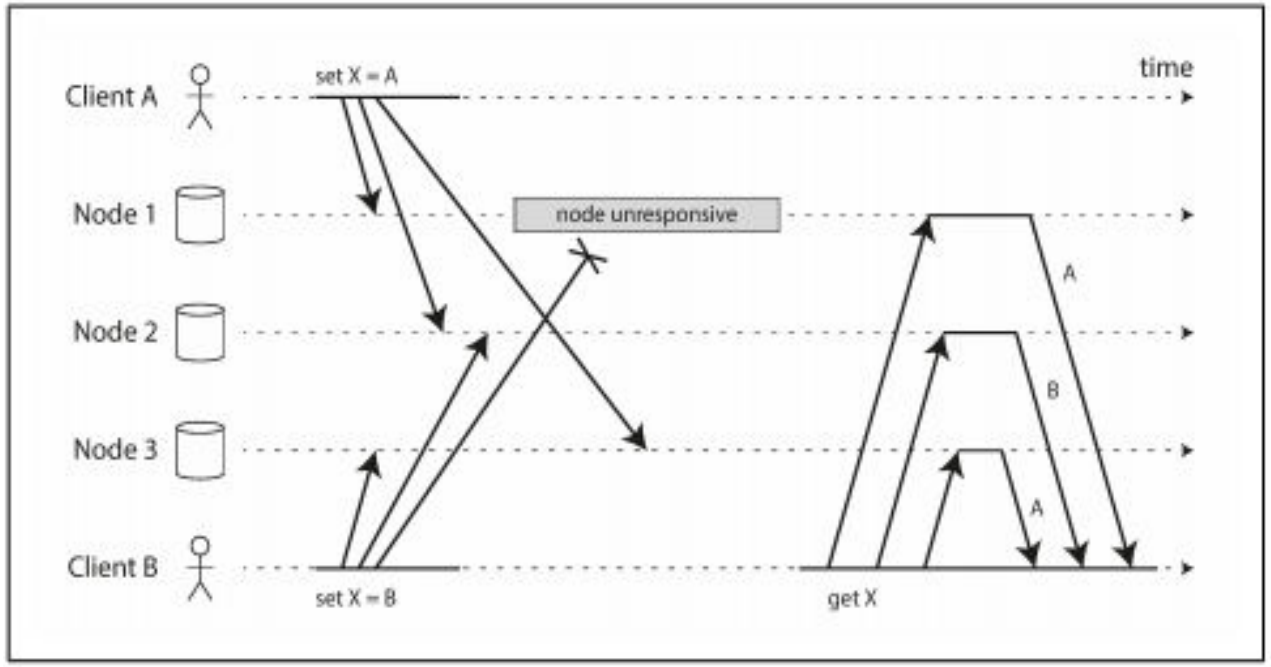
Leaderless Replication : Pitfalls - 3

Detecting Concurrent Writes

- Databases allow several clients **to concurrently write to the same key, which means that conflicts will occur even if strict quorums are used**
- Events may arrive in a **different order at different nodes, due to variable network delays and partial failures**
- **If each node simply overwrote the value for a key whenever it received a write request from a client, the nodes would become permanently inconsistent**
- In order to become eventually consistent, the replicas should **converge toward the same value**

Approaches to Address this

- **Usage of timestamps to resolve these write conflicts can be done.** This could lead to additional challenges as clocks may not be synchronized.



Leaderless Replication : Pitfalls - 3

Detecting Concurrent Writes (Cont.)

- **Last write wins** (discarding concurrent writes) - **One approach for achieving eventual convergence is to declare that each replica need only store the most "recent" value and allow "older" values to be overwritten and discarded.**
- **The "happens-before" relationship and concurrency** - How do we decide whether two operations are concurrent or not?
  - An operation A happens before another operation B if B knows about A, or depends on A, or builds upon A in some way. Whether one operation happens before another operation is the key to defining what concurrency means.
  - Server can determine whether **two operations are concurrent** by looking *at* **the version numbers**
  - In case of **multiple replicas, we need to use a version number per replica as well as per key stored in a version vector.**

    (cont.)

Leaderless Replication : Pitfalls - 3

Algorithm for determining whether two operations are concurrent by looking at the version numbers

- The server maintains a version number for every key, **increments the version number every time that key is written, and stores the new version number along with the value written**.
- When a client reads a key, **the server returns all values that have not been overwritten, as well as the latest version numbe**r. A client must read a key before writing.
- When a client writes a key, **it must include the version number from the prior read, and it must merge together all values that it received in the prior read**. (The response from a write request can be like a read, returning all current values, which allows to chain several writes.)
- When the server receives a write with a particular version number, it can overwrite all values with that version number or below (since it knows that they have been merged into the new value), **but it must keep all values with a higher version number** (because those values are concurrent with the incoming write).

Leaderless Replication : Quorum Consistency

Quorum

- Given n nodes, the quorum (w,r) specifies …
  - the number of nodes w that must acknowledge a write and
  - the number of nodes r that must answer a query

Quorum Consistency

- If w + r > n, then each query will contain the newest version of a value
  - Identify the newest value by its version (not by majority!)
- The quorum variables are usually configurable:
  - Smaller r (faster reads) causes larger w (slower writes) and vise versa
- The quorum tolerates:
  - n – w unavailable nodes for writes
  - n – r unavailable nodes for reads

A common choice is to make n an odd number (typically 3 or 5) and to set w = r = (n + 1) / 2

Monitoring staleness

- For leader-based replication, the database typically exposes metrics for the replication lag. By subtracting a **follower's current position from the leader's current position, you can measure the amount of replication lag.**
- For leaderless replication, **there is no fixed order in which writes are applied**, which makes monitoring more difficult

Multi-datacenter operation

- Leaderless replication is also suitable for **multi-datacenter operation, since it is designed to tolerate conflicting concurrent writes, network interruptions and latency spikes**.
- The number of replicas **n** includes nodes in all datacenters, and the number of replicas you want to have in each datacenter can be configured.
- **Each write from a client is sent to all replicas, regardless of datacenter, but the client usually only waits for acknowledgment from a quorum of nodes within its local datacenter so that it is unaffected by delays and interruptions on the cross-datacenter link.**
- The higher-latency writes to other datacenters are often configured to happen asynchronously

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Consistency Models

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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.

Consistency and Eventual Consistency

- The term ***consistency*** refers to the **consistency of the values in different copies of the same data item in a replicated distributed system.**
- This consistency can be **lost when there is a network issue** or there is **a difference in the time taken to write into different copies of the same data**.
- A ***consistency model*** is contract **between a distributed data store and processes**, in which the **processes agree to obey certain rules in contrast the store promises to work correctly**.
- There are applications and environments which need **strong consistency** i.e. **values across copies to be the same, which will need to use different mechanisms to achieve the same**. These mechanisms can lead to transactions slowing down and hence having an impact on the performance.
- A consistency model basically refers to **the degree of consistency that should be maintained for the shared memory data**

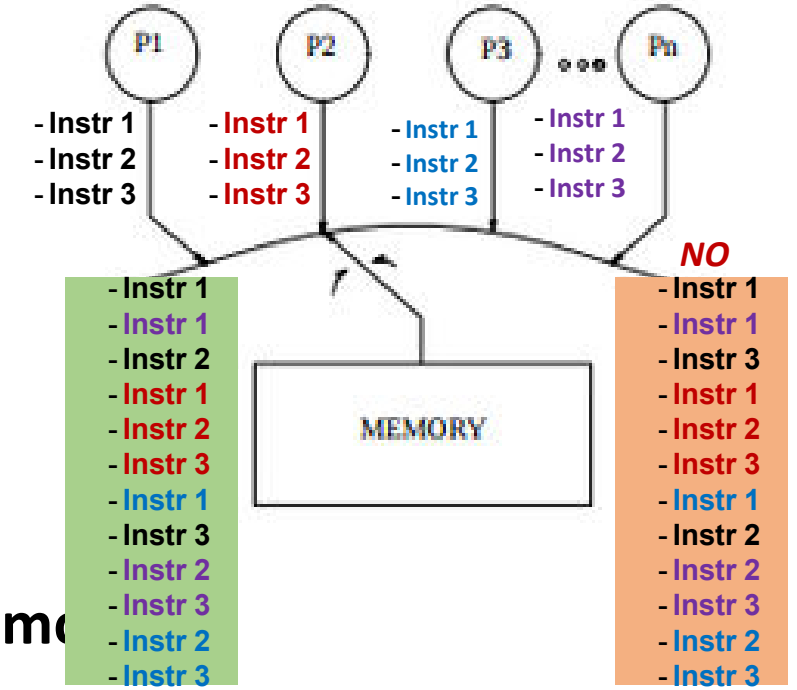
Consistency Models – Consistency Guarantees

- Eventual consistency is designing systems to eventually guarantee the copies of data to be consistent once all the current operations have been processed, but don't always have to be identical. This provides improved performance.
- Most replicated databases provide **eventual consistency**, which means that if you stop writing to the database and wait for some **unspecified length of time**, then eventually all read requests will return the same value. That is, all replicas will eventually converge to the same value
- This is a very **weak guarantee** as it doesn't say anything about when the replicas will converge
- The edge cases of eventual consistency only become apparent when there is a fault in the system or at high concurrency.

Consistency Models – Different Types

Sequential Consistency (Lamport)

- A shared-memory system is said to support the sequential consistency model, **if all processes see the same order of all memory access operations on the shared memory**.
- The exact order in which the memory access operations are interleaved does not matter**... If one process sees one of the orderings of ... three operations and another process sees a different one, the memory is not a sequentially consistent memo**



- Conceptually there is **one global memory and a switch that connects an arbitrary processor to the memory at any time.** Each **processor issues memory operations in program order** and the **switch provides the global serialization among all the processors**.
- Therefore: Sequential consistency is ***not deterministic*** because multiple execution of the distributed program might lead to a different order of operations.

Consistency Models – Different Types

Causal Consistency

- **Two events are causally related if one can influence the other**
- Relaxes the requirement of the sequential model for better concurrency. Unlike the sequential consistency model, in the causal consistency model, **all processes see only those memory reference operations in the same (correct) order that are potentially causally related.**
- Memory reference operations that are not potentially causally related **may be seen by different processes in different orders.**

PRAM (Pipelined Random-Access Memory) consistency

- **It ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed** as if all the write operations performed by a **single process are in a pipeline**.
- **Write operations performed by different processes may be seen by different processes in different orders**

Consistency Models – Different Types

Strict Consistency (also called Strong consistency or Linearizability)

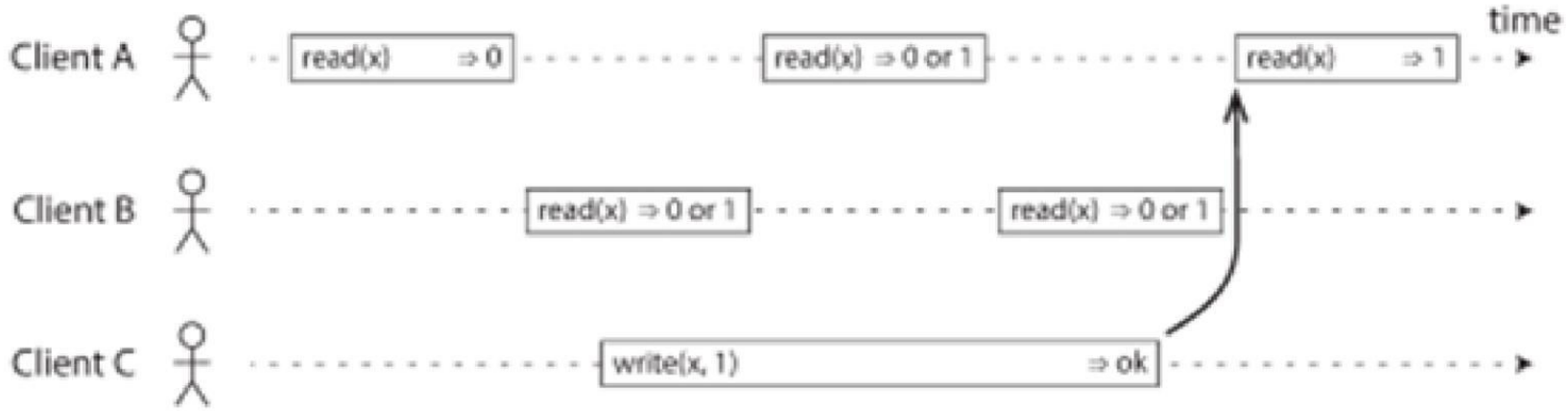
- A shared-memory system is said to support the strict consistency model **if the value returned by a read operation on a memory address is always the same as the value written by the most recent write operation to that address**, irrespective of the locations of the processes performing the read and write operations
- Like sequential consistency, but **the execution order of programs between processors must be the order** in which those operations were issued.
- Therefore: If each program in each processor is deterministic, then the distributed program is deterministic.

Linearizability

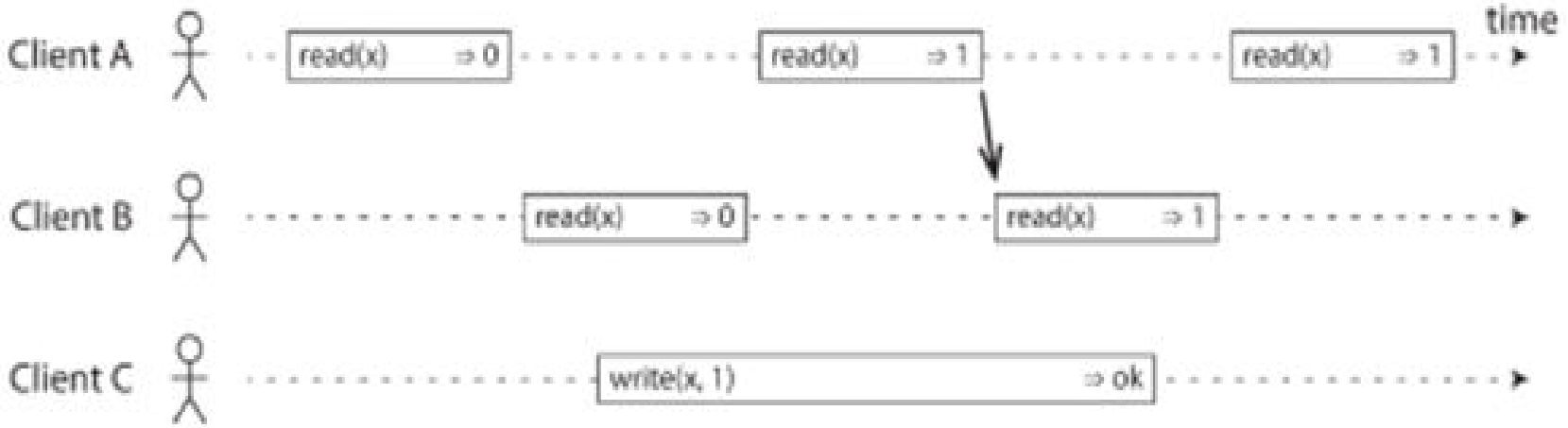
- Basic idea is to make a system appear as if there were only one copy of the data, and all operations on it are atomic.
- With this guarantee, even though there may be **multiple replicas** in reality, **the application does not need to worry about them.**
- Also known as **atomic consistency, strong consistency, immediate consistency or external consistency**
- **Linearizability** is a **recency guarantee**: a read is guaranteed to see the **latest value written**.
- In a linearizable system, as soon as **one client successfully completes a write**, **all clients reading from the database must be able to see the value** just written.
- Maintaining the illusion of a single copy of the data means guaranteeing that the value read is the most recent, up-to-date value and doesn't come from a stale cache or replica

Linearizability

- If a read request is concurrent with a write request, it may return either the old or the new value



- After any one read has returned the new value, all following reads (on the same or other clients) must also return the new value.



PES UNIVERSITY
CELEBRATING 50 YEARS

Compare and Set (cas)

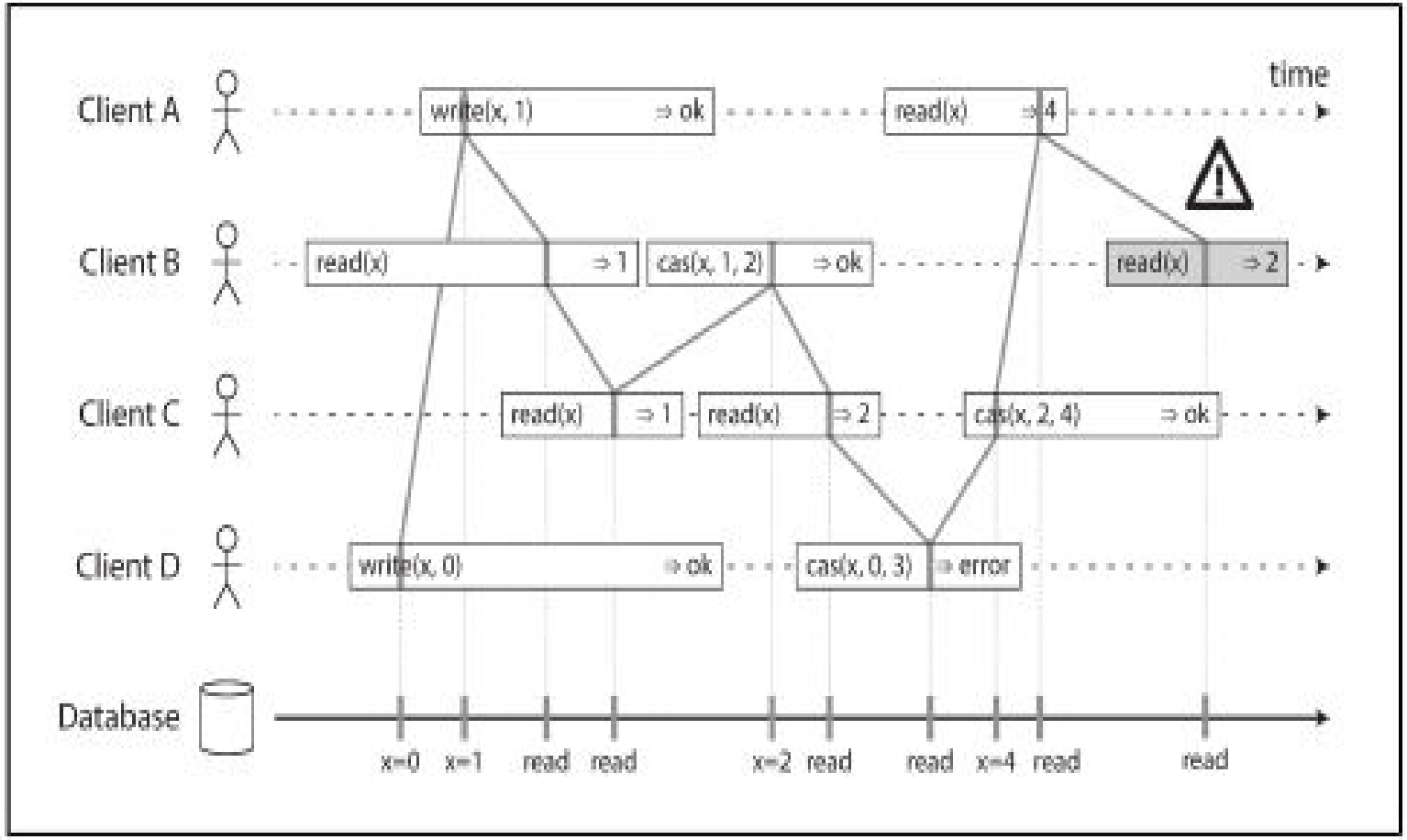
- Add a third type of operation besides read and write
- **cas(x, v_{old}, v_{new}) ⇒ r means the client requested an atomic compare-and-set operation** . If the current value of the register x equals v_{old}, it should be atomically set to v_{new}. If $x \neq v_{old}$ then the **operation should leave the register unchanged** and return an error. r is the database's response (ok or error).

CLOUD COMPUTING

Linearizability

Visualizing the points in time at which the reads and writes appear to have taken effect

- The final read by B is not linearizable
- It is possible to test whether a system's behavior is linearizable by recording the timings of all requests and responses and checking whether they can be arranged into a valid sequential order



Linearizability

Single-leader replication (potentially linearizable)

- If you make reads from the leader or from synchronously updated followers, they have the potential to be linearizable

Consensus algorithms (linearizable)

- **Consensus protocols contain measures to prevent split brain and stale replicas**.
- Consensus algorithms can implement linearizable storage safely
- Example: ZooKeeper

Multi-leader replication (not linearizable)

Leaderless replication (probably not linearizable)

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

CAP Theorem

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

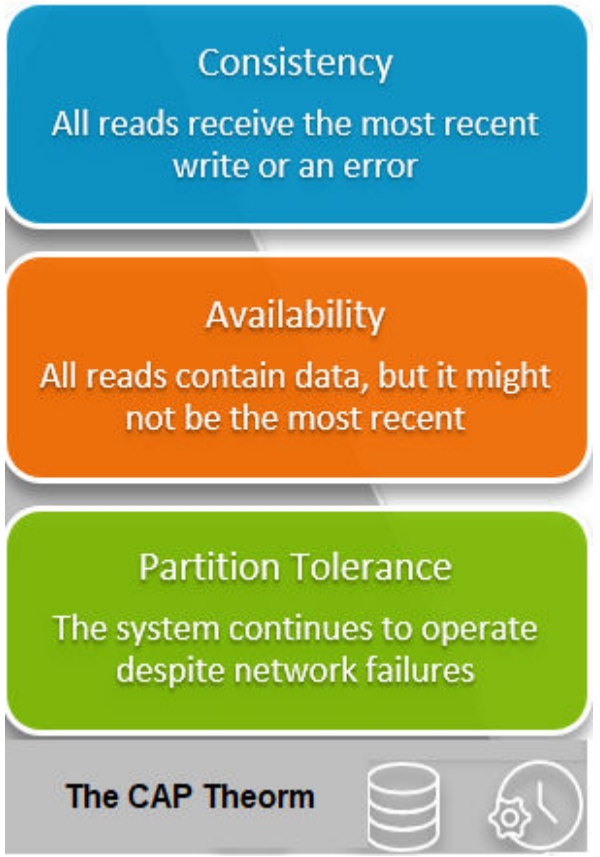
Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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

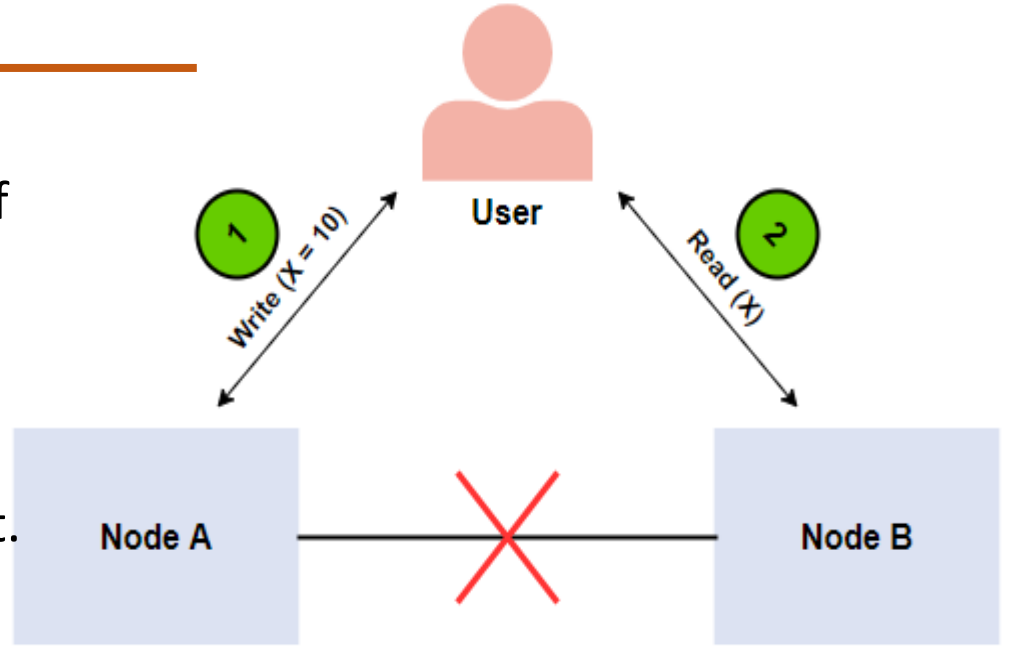
CAP Theorem

- The CAP theorem, originally introduced as the CAP principle, can be used to explain some of the competing requirements in a distributed system with replication.
- It is a tool used to make system designers aware of the trade-offs while designing “network shared-data systems” **or** A distributed system that stores data on more than one node (physical/virtual machines) at the same time
- The three letters in CAP refer to three desirable properties of distributed systems with replicated data:
  - **C - Consistency** (among replicated copies)
  - **A - Availability** (of the system for read and write operations)
  - **P - Partition tolerance (**in the face of the nodes in the system being partitioned by a network fault).
- The **CAP theorem** states that **it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication**. We can strongly support only two of the three properties



CLOUD COMPUTING

CAP Theorem Example



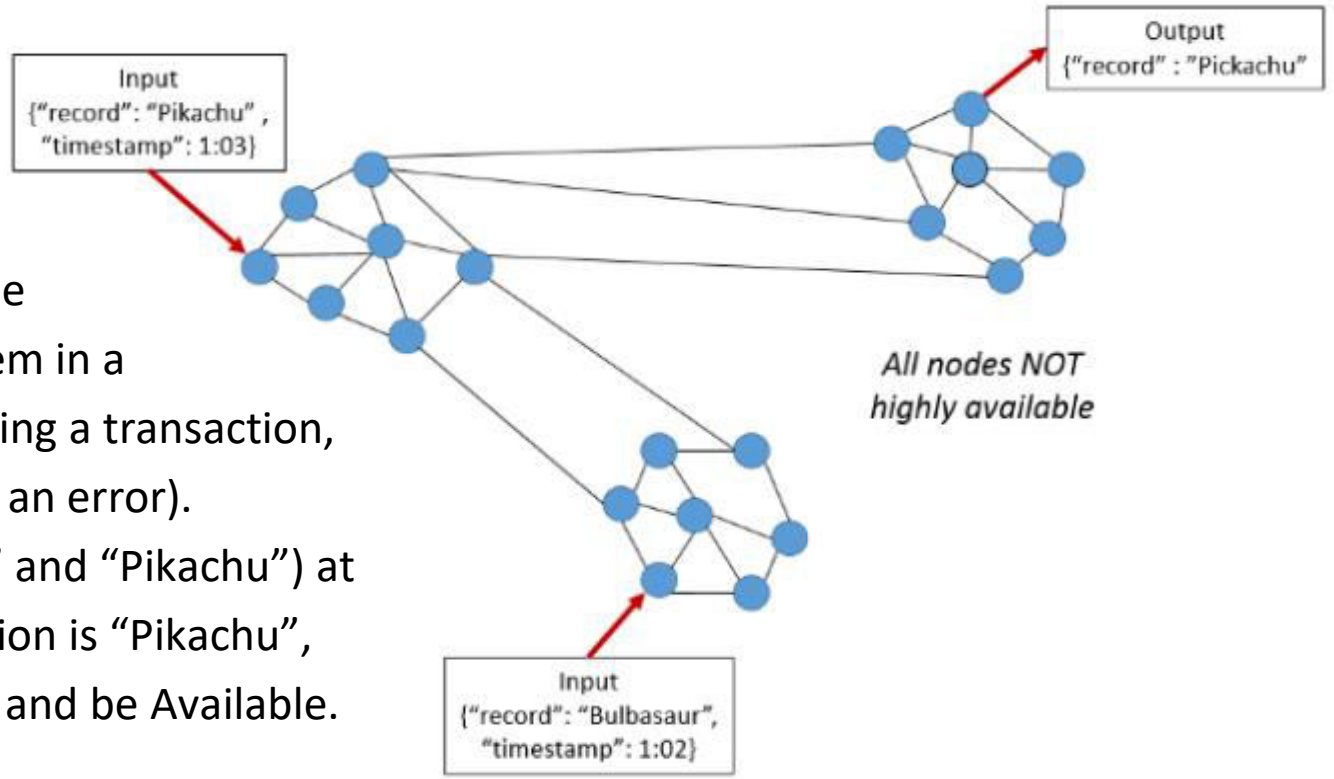
- Imagine a distributed system consisting of two nodes:
- The distributed system acts as a plain register with the value of variable X.
- There's a network failure that results in a network partition between the two nodes in the system.
- An end-user performs a write request, and then a read request.
- Let's examine a case where a different node of the system processes each request. In this case, our system has two options:
  - **It can fail at one of the requests, breaking the system's availability**
  - It can execute both requests, returning a stale value from the read request and breaking the system's consistency
- The system can't process both requests successfully while also ensuring that the read returns the latest value written by the write.
- This is because the results of the write operation can't be propagated from node A to node B because of the network partition.

CAP Theorem

1. **Consistency (C) Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions.** It's the guarantee that every node in a distributed cluster returns the same, most recently updated value of a successful write at any logical time.

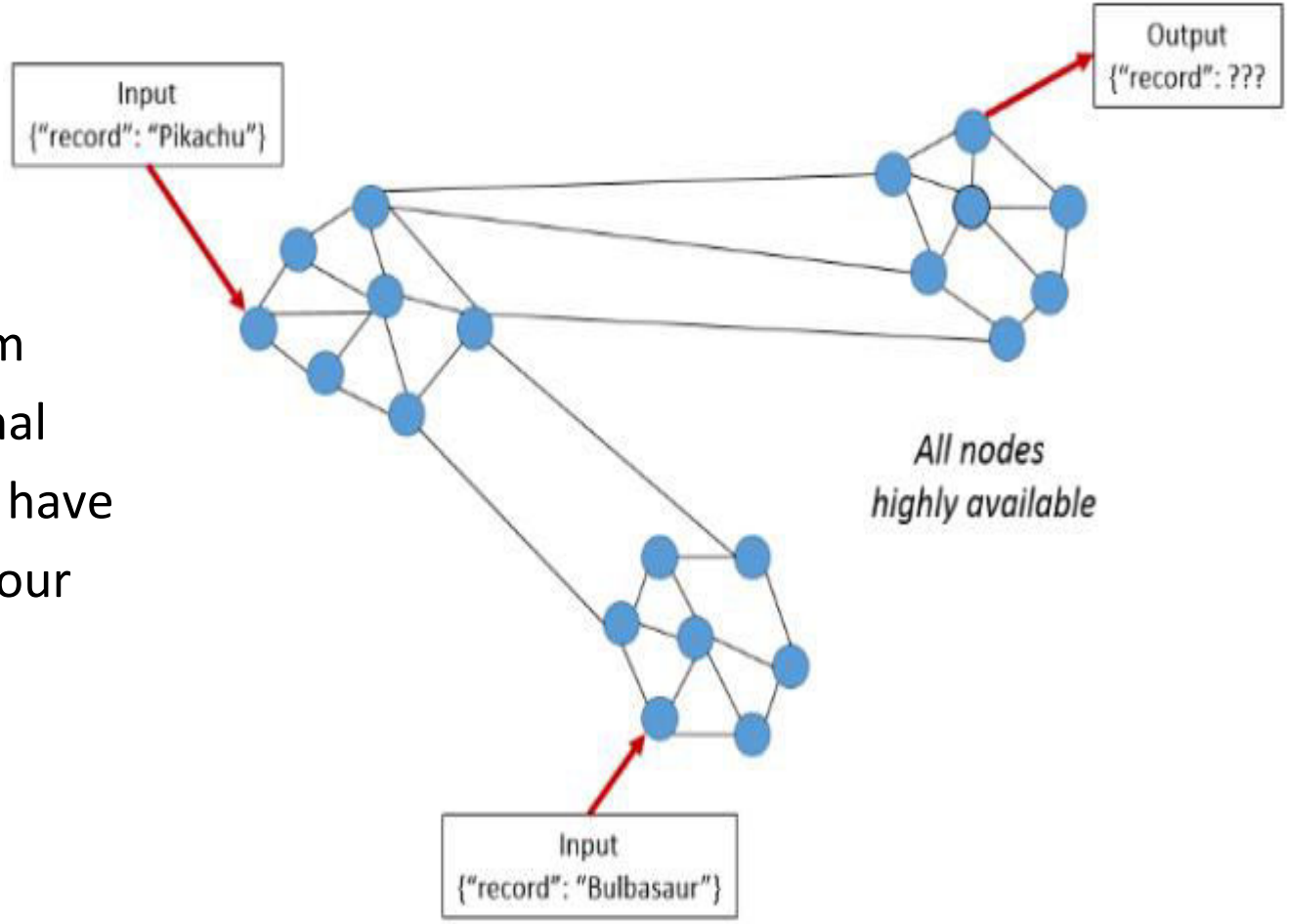
   In CAP, the term consistency refers to **the consistency of the values in different copies of the same data item in a replicated distributed system**. This can be verified if all reads initiated after a successful write return the same and latest value at any given logical time.

- Performing a read operation will return the value of the most recent write operation causing all nodes to return the same data.
- A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state (may have an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error).
- In the image, we have 2 different records ("Bulbasaur" and "Pikachu") at different timestamps. The output on the third partition is "Pikachu", the latest input although it will need time to update and be Available.

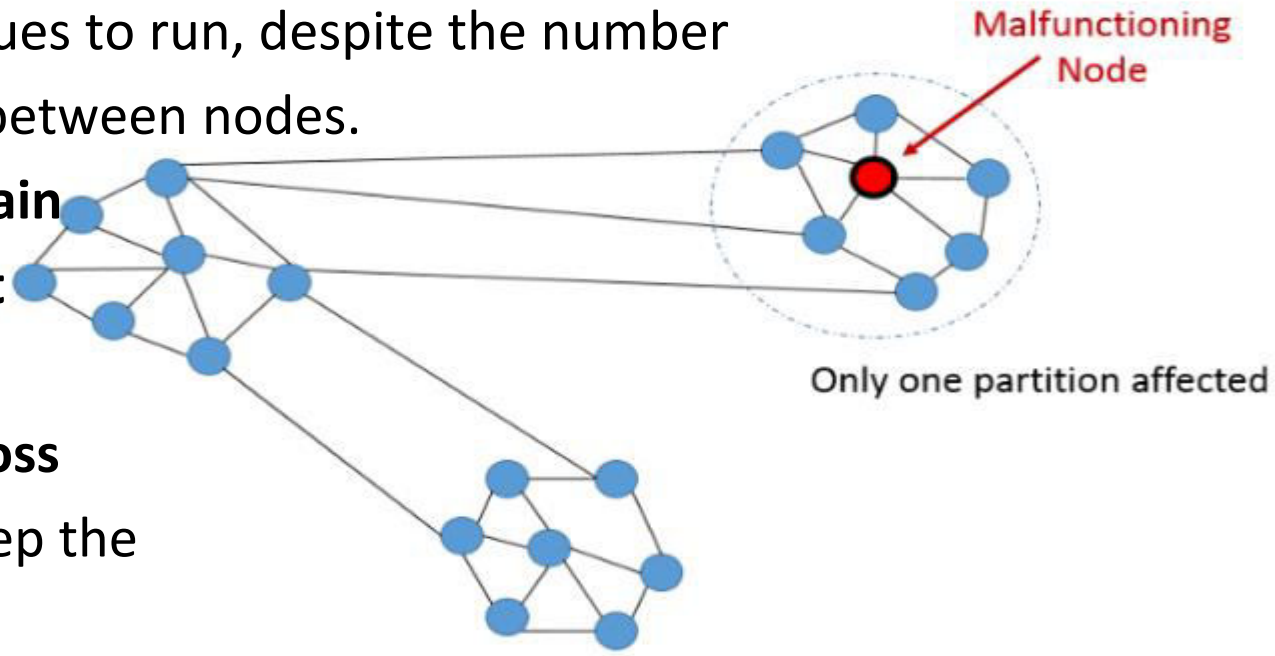


2. **Availability (A)** Availability means that each read or write request for a data item from a client to a node will either be **processed successfully or will receive a message that the operation cannot be completed** (if not in failed state).

- All working nodes in the distributed system return a valid response for any request, without exception
- Achieving availability in a distributed system requires that the system remains operational 100% of the time, which may need that we have "x" servers beyond the "n" servers serving our application



3. **Partition Tolerance** (P) Partition tolerance requires **that a system be able to re-route a communication when there are temporary breaks or failures in the network** (network partitions).
   - It means that the system **continues to function and upholds its consistency guarantees in spite of network partitions.**
   - Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.
   - This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes.
   - **A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network**.
   - **Data records are sufficiently replicated across combinations of nodes and networks** to keep the system up through intermittent outages.



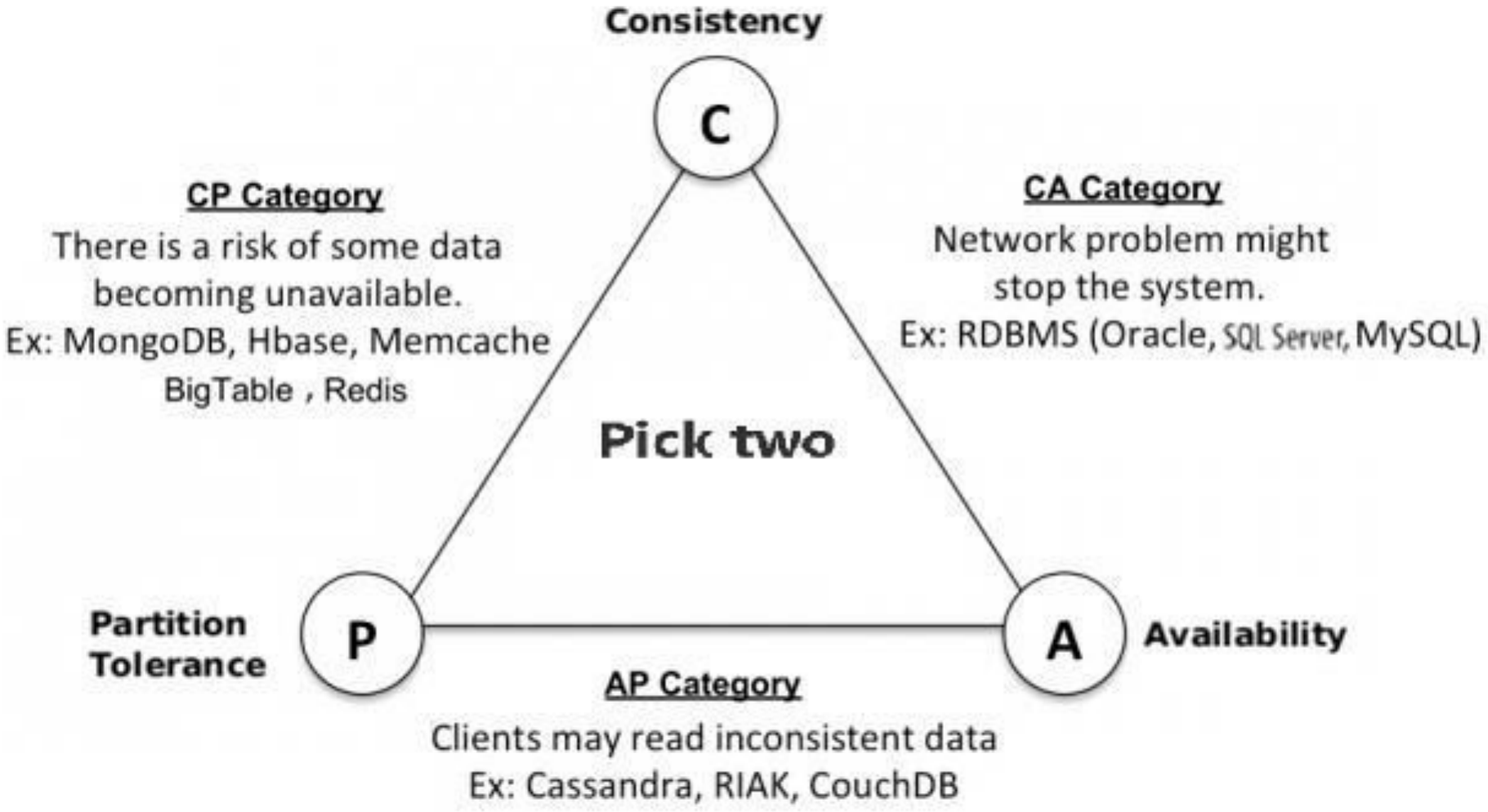
CAP Theorem : Tradeoffs between Requirements (Illustrations)

1. ***Availability and Partition-Tolerant*** ***(Compromised Consistency)***: Say you have two nodes and the link between the two is severed. **Since both nodes are up, you can design the system to accept requests on each of the nodes, which will make the system available despite the network being partitioned. However, each node will issue its own results, so by providing high availability and partition tolerance** you'll compromise *consistency*.

2. ***Consistent and Partition-Tolerant*** ***(Compromised Availability)***: **Say you have three nodes and one node loses its link with the other two. You can create a rule that, a result will be returned only when a majority of nodes agree**. So In-spite of having a partition, the system will return a consistent result, but since the separated node won't be able to reach consensus it won't be *available* even though it's u

3. ***Consistent and Available*** ***(Compromised on a Partition-Tolerance)***: Although, a system can be both consistent and available, but it may have to block on a *partition*.

CAP Theorem : Illustrative Examples of products/Application categories to choose



CAP Theorem

- The “Pick Two” expression of CAP opened the minds of designers to a wider range of systems and tradeoffs
- CAP is **NOT** a choice “at all times” as to “which one of the three guarantees to abandon”. In fact, **the choice is between consistency and availability only when a network partition or failure happens**. When there is no network failure, both availability and consistency can be satisfied
- Products like DB e.g. **Casandra supports AP but provides *eventual consistency*** by allowing clients to write to any nodes at any time and reconciling inconsistencies as quickly as possible
- Reference: https://www.youtube.com/watch?v=9uCP3qHNbWw

CAP Theorem and some of the criticisms

- Although the Theorem **doesn't specify an upper bound on response time for availability,** in practice, there's exists a timeout. **CAP Theorem ignores latency, which is an important consideration in practice**. **Timeouts are often implemented in services**. During a partition, if we cancel a request, we maintain consistency but forfeit availability. In fact, latency can be seen as another word for availability.
- In NoSQL distributed databases**, CAP Theorem has led to the belief that eventual consistency provides better availability than strong consistency**. It could be considered as an outdated notion. It's better to factor in sensitivity to network delays.
- CAP Theorem suggests a binary decision. In reality, it's a continuum. It’s more of a trade-off. There are different degrees of consistency implemented via "**read your writes, monotonic reads and causal consistency".**

CAP Theorem and how do you use them when making decisions

- CAP Theorem can feel quite abstract, but both from a technical and business perspective the trade-offs will lead to some very important questions which has practical, real-world consequences.
- These questions would be
  - Is it important to avoid throwing up errors to the client?
  - Or are we willing to sacrifice the visible user experience to ensure consistency?
  - Is consistency an actually important part of the user's experience
  - Or can we actually do what we want with a relational database and avoid the need for partition tolerance altogether?
- All of these are ultimately leading to user experience questions.
- This will need understanding of the overall goals of the project, and context in which your database solution is operating. (E.g. Is it powering an internal analytics dashboard? Or is it supporting a widely used external-facing website or application?)

CAP Theorem and its relationship to Cloud Computing

- Microservices architecture for applications is popular and prevalent on both cloud servers and on-premises data centers
- We have also seen that microservices **are loosely coupled, independently deployable application components that incorporate their own stack, including their own database and database model**, and communicate with each other over a network.
- Understanding the CAP theorem can help you choose the best database when designing a microservices-based application running from multiple locations.
  E.g. If the ability to quickly **iterate the data model and scale horizontally i**s essential to your application, **but you can tolerate eventual (as opposed to strict) consistency, an database like Cassandra or Apache CouchDB** (supporting AP - Availability and Partitioning) can meet your requirements and simplify your deployment. On the other hand, **if your application depends heavily on data consistency—as in an eCommerce application or a payment service—you might opt for a relational database like PostgreSQL.**

CAP Theorem Conclusion

- Distributed systems allow us to achieve a level of computing power and availability that were simply not available in the past.
- Our systems have higher performance, lower latency, and near 100% up-time in data centers that span the entire globe.
- These systems of today are run on commodity hardware that is easily obtainable and configurable at affordable costs.
- The disadvantage to it though is Distributed systems are more complex than their single-network counterparts.
- Understanding the complexity incurred in distributed systems, making the appropriate trade-offs for the task at hand (CAP), and selecting the right tool for the job is necessary with horizontal scaling.

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Distributed Transactions

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 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.

Transactions

- **Transaction** is an operation composed of a number of discrete steps
- All the steps must be completed for the transaction to be **committed**. The results are made permanent else the transaction is **aborted** and the state of the system **reverts** to what it was before the transaction started E.g. Buying a house
- Basic Operations
  - Transaction primitives:
    - Begin transaction: mark the start of a transaction
    - End transaction: mark the end of a transaction; try to commit
    - Abort transaction: kill the transaction, restore old values
    - Read/write data from files (or object stores): data will have to be restored if the transaction is aborted.

Transactions : Properties of Transactions (ACID)

ACID :

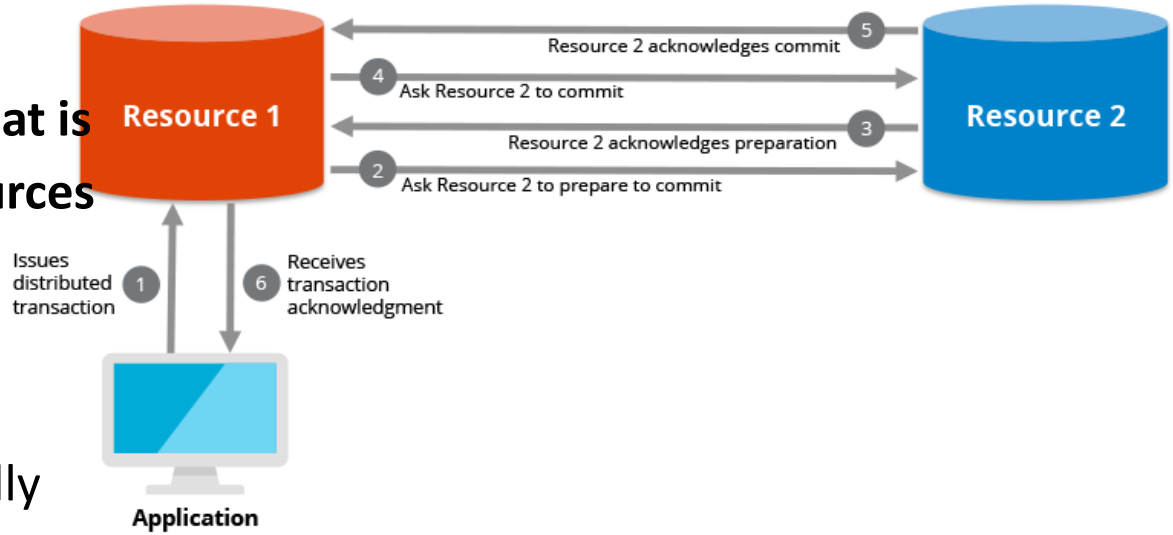
- Atomic
  - The transaction happens as a single indivisible action. Others do not see intermediate results. All or nothing.
- Consistent
  - If the system has invariants, they must hold after the transaction. E.g., total amount of money in all accounts must be the same before and after a “transfer funds” transaction.
- Isolated (Serializable)
  - If transactions run at the same time, the final result must be the same as if they executed in some serial order.
- Durable
  - Once a transaction commits, the results are made permanent. No failures after a commit will cause the results to revert.

CLOUD COMPUTING
Transactions and different types of transactions :

Nested Transactions

- Nested Transaction is a **top-level transaction which may create sub-transactions**
- Problem:
  - **Sub-transactions may commit** (results are durable) but **the parent transaction may abort.**
- One solution : private workspace
  - **Each sub-transaction is given a private copy of every object it manipulates**. On commit, the private copy displaces the parent's copy (which may also be a private copy of the parent's parent)

Distributed Transactions



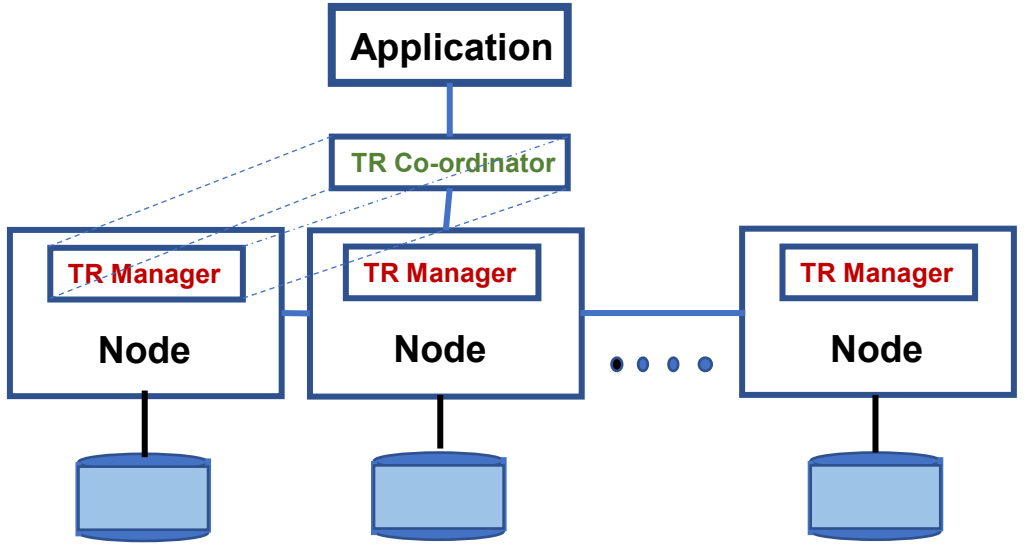
- A distributed **transaction is a set of operations on data that is performed across two or more data repositories or resources across different systems**
- Challenge: handle machine, software, & network failures while preserving transaction integrity.
- There are two possible outcomes all operations successfully **complete, or none of the operations are performed** at all due to a failure somewhere in the system
- In the second outcome if some work was completed prior to the failure, that work will be reversed to ensure no net work was done. This type of operation is in compliance with the "ACID" principles to ensure data integrity

CLOUD COMPUTING

Distributed Transaction System Architecture

- A system architecture supporting distributed transactions, **has multiple data repositories hosted on different nodes connected by a network.**
- Transaction may access data at several nodes/sites.
- Each site has a

  local transaction manager responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
  - Responsible for sub-transactions on that system
  - Performs prepare, commit and abort calls for sub-transactions
  - Each sub-transaction must agree to commit changes before the transaction can complete



  Transaction coordinator coordinating activities across the data repositories
  - Periodically a local transaction manager is nominated as a local coordinator
  - Starting the execution of transactions that originate at the site.
  - Distributing sub transactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Concurrency Control

- **All concurrency mechanisms must preserve data consistency and complete each atomic action in finite time**
- Important capabilities are
  a) Be **resilient to site** and **communication link failures**.
  b) Allow **parallelism** to enhance **performance requirements**.
  c) Incur **optimal cost** and **optimize communication delays**
  d) Place **constraints on atomic action**.

Commit Protocols

- **Commit protocols are used to ensure atomicity across sites**
  - A transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - Not acceptable to have a transaction committed at one site and aborted at another.

- The two-phase commit (2 PC) protocol is widely used.

CLOUD COMPUTING

Two Phase Commit Protocol - Phase 1

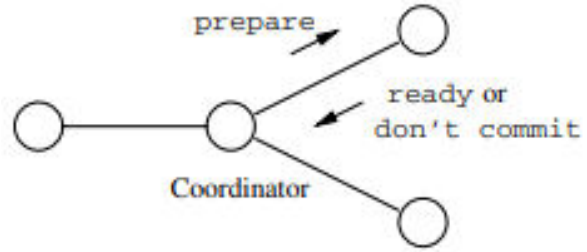
Lets consider a transaction is named as "**T**"

1. The coordinator places a log record **prepare T** on the log at its site.
2. The coordinator sends to each component's site the message **prepare T**.
3. Each site receiving the message prepare its component of the transaction "T".
4. If a site wants to commit its component, it must enter a state called **pre-committed** with a **Ready T** message. Once in the **pre-committed** state, the site cannot ***abort*** its component of T without a directive to do so from the coordinator

   So, after **prepare T** is received, Perform whatever steps necessary to be sure the local component of T will not have to abort

   If everything is fine and it ready to commit, then place the record **Ready T** on the local log and flush the log to disk and send **Ready T** message to the coordinator

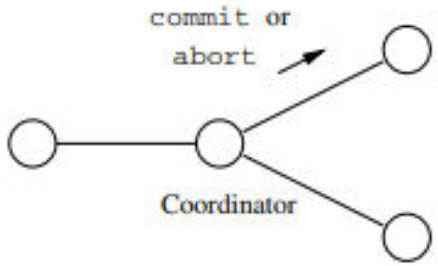
   If the site wants to abort its component of T, then it logs the record **Don't Commit T** and sends the message **Don't commit T** to the coordinator



Two Phase Commit : Phase 1	
Co-Ordinator	**Related Nodes (Sites)**
▪ Write prepare to commit T to *log* at its site ▪ Send *prepare to commit* message	▪ Receive *prepare* message ▪ Work on the components towards T ▪ If ready to commit get into pre-committed state for T and place *Ready T* in the local log and send *Ready T* message to the coordinator (holds locks..) ▪ If not ready to commit place *Don't commit T* to the local log and send *Don't commit T* message to the coordinator ▪ Wait for message from coordinator
▪ Wait for reply from each related node	

CLOUD COMPUTING

Two Phase Commit Protocol - Phase 2



PES UNIVERSITY
CELEBRATING 50 YEARS

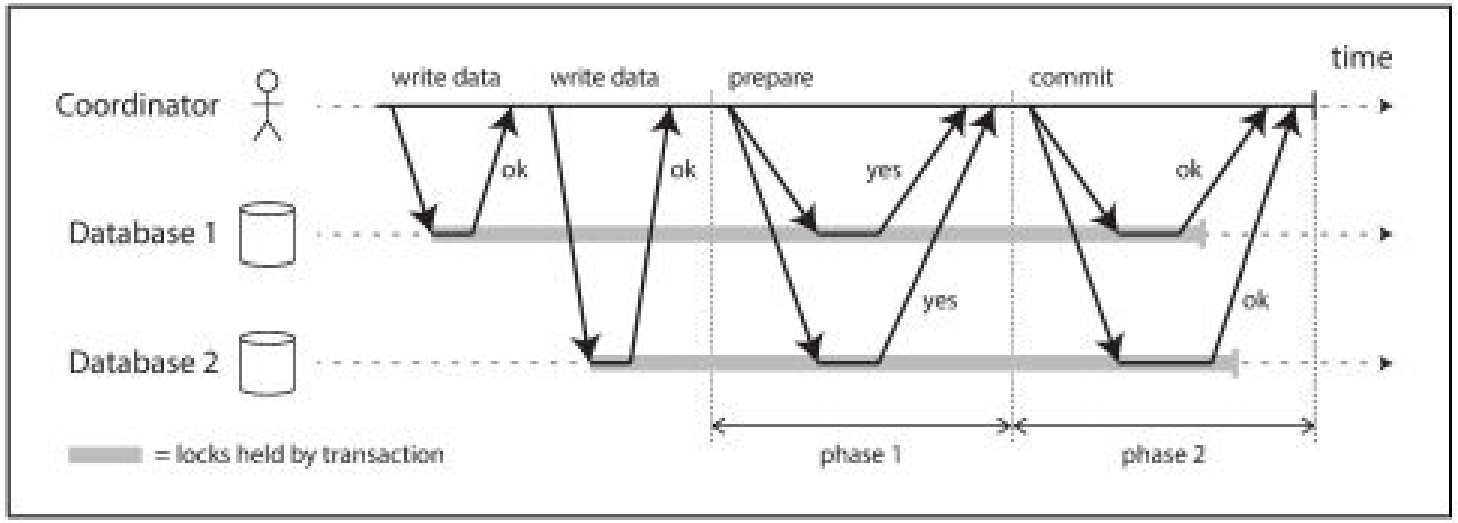
1. If the coordinator has received ready T from all components of T, then it decides to commit T. The coordinator logs **Commit T** at its site and then sends message ***commit T*** to all sites involved in T
2. If the coordinator has received don't commit T from one or more sites, it logs **Abort T** at its site and then sends ***abort T*** messages to all sites involved in T
3. If a site receives a ***commit T*** message, it commits the component of T at that site, releases the locks ..,logging **Commit T** as it does.
4. If a site receives the message ***abort T***, it aborts T, releases locks and writes the log record **Abort T**

Two Phase Commit : Phase 2	
Co-Ordinator	**Related Nodes (Sites)**
▪ If *Ready T* has been received from all nodes, then write *commit T* to the local log and Send *commit T* message ▪ If Don't commit T is received from any of the related nodes, then write *Abort T* to the local log and send *Abort T* is sent to all related nodes (sites) ▪ Wait for *Done* message and clear up all states	(If in the Pre-Committed State, continue to hold the locks) ▪ Receive *Commit T* or *Abort T* message ▪ If *Commit T* is received, commit the component of T at the site, release locks .. and place *Commit T* to local log and send *Done* message to the co-ordinator ▪ If *Abort T* is received, roll back all changes, release locks .. and place *Abort T* to the local log and send *Done* message to the Co-ordinator

CLOUD COMPUTING

Two Phase Commit Protocol

A successful execution of two-phase commit



THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu

CLOUD COMPUTING

Storage Virtualization in Different Industries - Case Studies

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. Sujatha R Upadhyaya** and would like to acknowledge and thank her for the same.. 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.

Storage Virtualization Case Studies in Different Contexts

This presentation showcases

- Five contexts where storage virtualization is used in recent years

We discuss

- The industry background
- Problem being addressed
- Solution approach
- Outcome

PES UNIVERSITY
CELEBRATING 50 YEARS

Consolidated Communications Holdings

Industry Background:

This 125 years old company, physically spans over 50,000 route miles, 20+ states, and 1.8 million connections.

The descison to go for new storage solution came up after facing an email outage that leaked its residential email clients.

Industry: Communications

Storage virtualization provider: DataCore

Solution

- The company employed
  - a SAN solution with attention to flexibility and performance
  - To keep up with the firm's requirements for caching.
- The firm uses DataCore's storage solution in two forms:
  - a traditional storage product consisting of hard drives
  - solid-state drives (SSDs)
- A hyperconverged model with greater storage through virtualized storage controllers.

Outcomes

- Consistent 100% uptime delivered over a period of 10years
- Reduced storage related spend
- Reduced time, effort, cost spent on preventive meintenance

Krishna Institute Of Medical Sciences

With a network of over 13 hospitals and 4,000 beds, Krishna Institute of Medical Sciences (KIMS) provides comprehensive healthcare services across cardiac sciences, oncology, neurosciences, organ transplantation, and 20 other specialities.

Industry: Health care

Storage virtualization provider: VMware and VMware vSAN

Problem Context

As KIMS Hospitals expanded, the workload increased threefold.

Due to insufficient information on the performance of the existing storage infrastructure, there was an increase storage options they considered (onsite, ad hoc, and pay-on-demand cloud storage options)

With storage option, the management and maintenance of storage infrastructure became ever more complicated.

Some of the applications were functioning on outdated software

Made wayvulnerable to ransomware attacks.

KIMS Hospitals needed a robust and swift IT infrastructure that could manage big data within complex systems and numerous applications

Solution

KIMS Hospitals deployed the VMware vSAN solution with Dell hardware to meet its existing needs for redundancy, scalability and cost efficiency.

VMware's virtualized storage, combined with VMware vRealize Operations optimized data exchange between networks and applications.

The modernized hyperconverged infrastructure (HCI) allowed IT administrators at KIMS Hospitals to have a virtual data plane with real-time analytics on virtual machines (VMs) storage attributes, including performance, capacity, and availability.

Outcomes:

- 80% of workloads moved from AWS to HCI, saving thousands in billings
- 98% VMware vSAN issues resolved in the pre-development phase
- No downtime encountered during maintenance activities

Seneca Family of Agencies

Founded in 1985, the community caters to over 18,000 young people and families across California and Washington State

Helps children and families see through the hardest times in their lives

offers permanency, mental health services, education, and juvenile justice.

Industry: Mental health services

Storage virtualization provider: StarWind and StarWind HCA

Problem Context

They were using a mix of dated VMware hosts and a Hyper-V cluster for its virtual environment.

The high incidences of VMware cluster renewal and Hyper-V's slow performance compelled the organization to move toward hyperconvergence.

The existing IT infrastructure was limited by storage problems and required extensive patching times done off-hours.

Solution

They implemented the HCA and ran high availability with simply two nodes with StarWind HCI Appliance (HCA)

Outcomes:

- Patching in cluster now done during business hours
- Quick migration across hosts
- Increased storage capacity

PES UNIVERSITY
CELEBRATING 50 YEARS

Grupo Alcione

With over 30 years of experience, Grupo Alcione specializes in electrical equipment and supplies.

Today, the company has expanded to 22 branches, 800 employees, and 11 Mexican states.

- **Industry:** Manufacturing
- **Storage virtualization provider:** Nutanix and Nutanix AHV

Electronics is a competitive industry; to realize better customer conversions,

Problem

- TGrupo Alcione wanted quickly access inventory in real-time
  - To update customers about pricing and budgeting information in real-time without delays
- They neededa high-performance, multicloud system with a lower-cost investment,

Problem

- Nutanix AHV built Alcione's infrastructure on nine nodes and two clusters.
- The first cluster, having six nodes, works as a production environment.
- The second cluster, with three nodes, serves the disaster recovery plan (DRP)
  - Here iImportant operational virtual servers replicate every hour
    - ensured critical service availability.

Outcomes:

- 30% savings on data center expenses
- Quick, real-time access to inventory
- Higher availability of critical services and improved reliability
- Increased operational efficiency
- Reduction in frequent customer support calls

Daegu Metropolitan City

This is not a strict storage virtualization context. It's more like cloud adoption

The third largest city in the Republic of Korea, Daegu, has a population of 2.5 million residents with eight administrative districts and 13,000+ public employees.

In 2015, the government decided to work on a three-phase project to create the Daegu

Industry: Government

Storage virtualization provider: Red Hat and Red Hat Virtualization

Project City Cloud (D-Cloud)

An ambitious project to offer important services like public notices, healthcare, and financial assistance in a timely manner.

The city government needed to update its decade-old IT infrastructure along with hardware sourced from different places.

Daegu dealt with higher operational costs, primarily because of multiplicity of hardware and software solutions.

Solution Approach

The city initially deployed Red Hat Enterprise Linux, Red Hat JBoss Enterprise Application Platform (EAP), and Red Hat Virtualization to create its new cloud-based service environment.

Red Hat Enterprise Linux created the foundation for Daegu to start building the hybrid cloud infrastructure.

Once the initial framework was laid out, the virtualization software laid out a central infrastructure for virtualized and cloud-native workloads.

Red Hat JBoss EAP then helped Daegu to overtake the managerial part, ensure user security, and attain high performance at scale.

PES UNIVERSITY
CELEBRATING 50 YEARS

Outcomes:

- Reduced operating costs by 36%
- Transformation of 50% of the legacy IT systems into cloud-based environment
- Simplified infrastructure operations and management
- Created foundation for easier resident access to information and services

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu