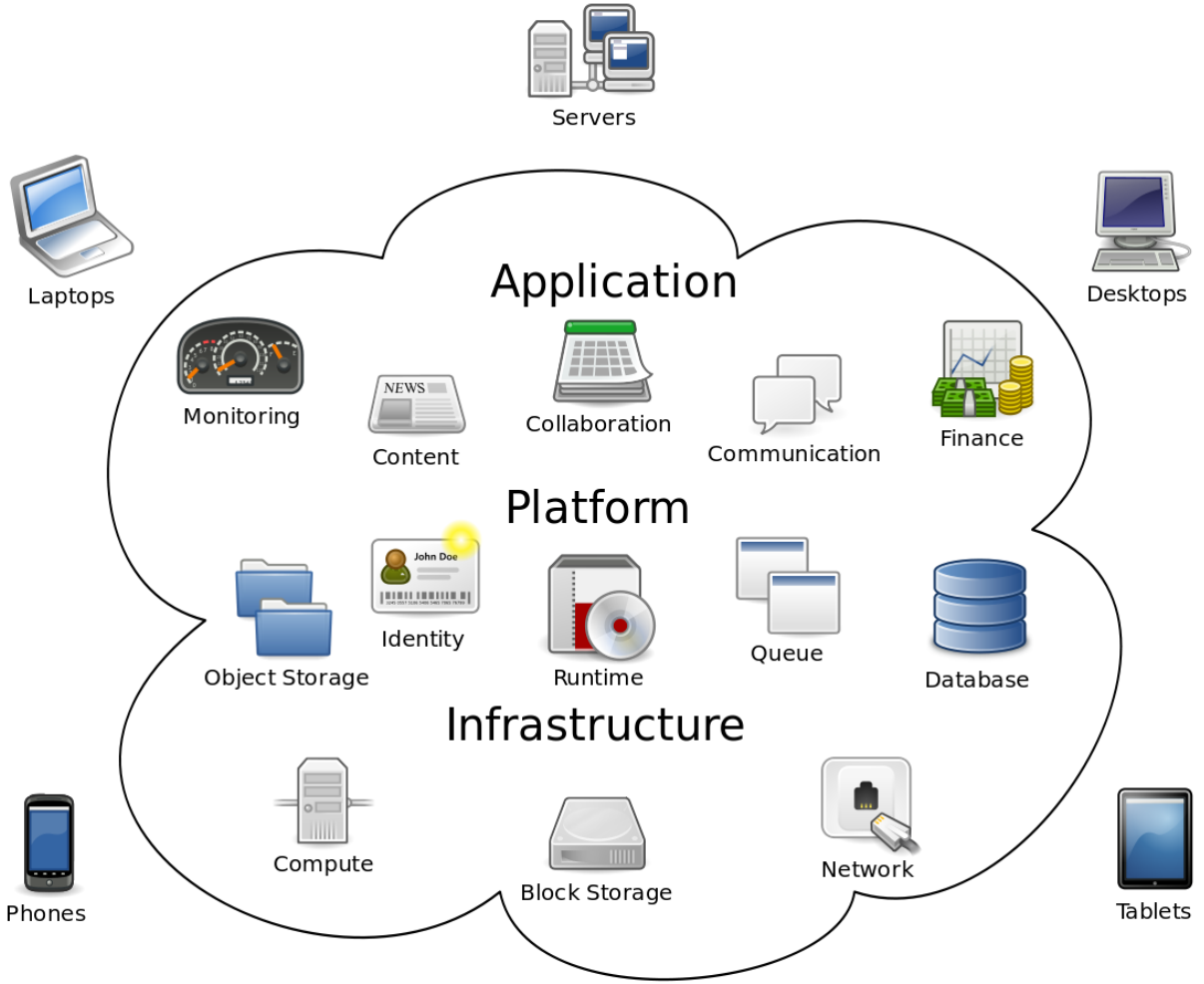
# Cloud Computing




## UNIT I

# Cloud Computing

Cloud computing is the delivery of computing services

It allows users to

1. Access & Storre the DATA,
2. Run the application ,
3. Use computing power via the internet.

It Offers

1. Faster Innovation,
2. Flexible resources,
3. Low Cast.

# Introduction to Cloud Computing

Definition:

Delivery of computing services (servers, storage, databases, networking, software) over the Internet ("the cloud").

Benefits: Cost-efficiency, scalability, flexibility, accessibility.

Deployment Models: Public, Private, Hybrid.

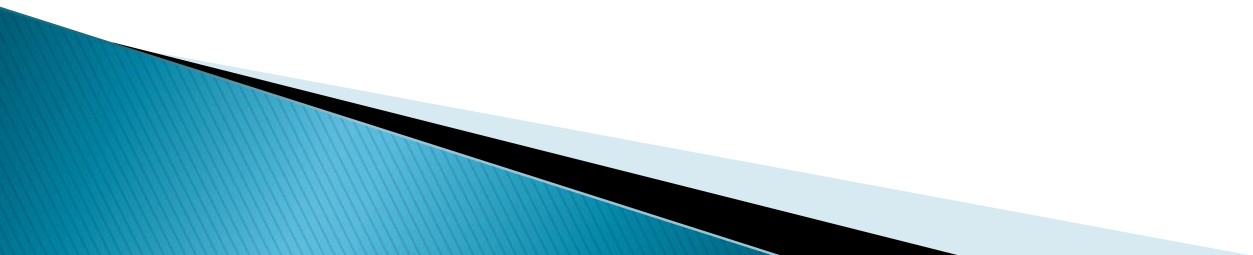
Service Models: IaaS, PaaS, SaaS.

# What is Cloud Computing?

The U.S. National Institute of Standards and Technology (NIST) defines cloud computing as:

- Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)

that can be rapidly provisioned and released with minimal management effort or service provider interaction.



# Ubiquitous computing

Ubiquitous computing also known as pervasive computing

It means computers are everywhere but they are invisible and work in background to assist us.

Example

1. Smart homes where lights, thermostats, and appliances adjust automatically based on your presence or preferences.

2. Wearable fitness tracks

3. Navigation systems in cars.

# Ubiquitous computing

Ubiquitous computing means “anywhere, anytime and anyplace computing” and refers to computing with

pervasive devices at any place and time using wired or wireless communication.

The Internet of Things (IoT) is a networked connection of everyday objects including computers, sensors, humans, etc.

The IoT is supported by Internet clouds to achieve ubiquitous computing with any object at any place and time.

# Convenient,On Demand

Convenient: The service is user-friendly and accessible from various devices (laptops, phones, tablets).

On-demand: Resources can be provisioned and released automatically, without human intervention.

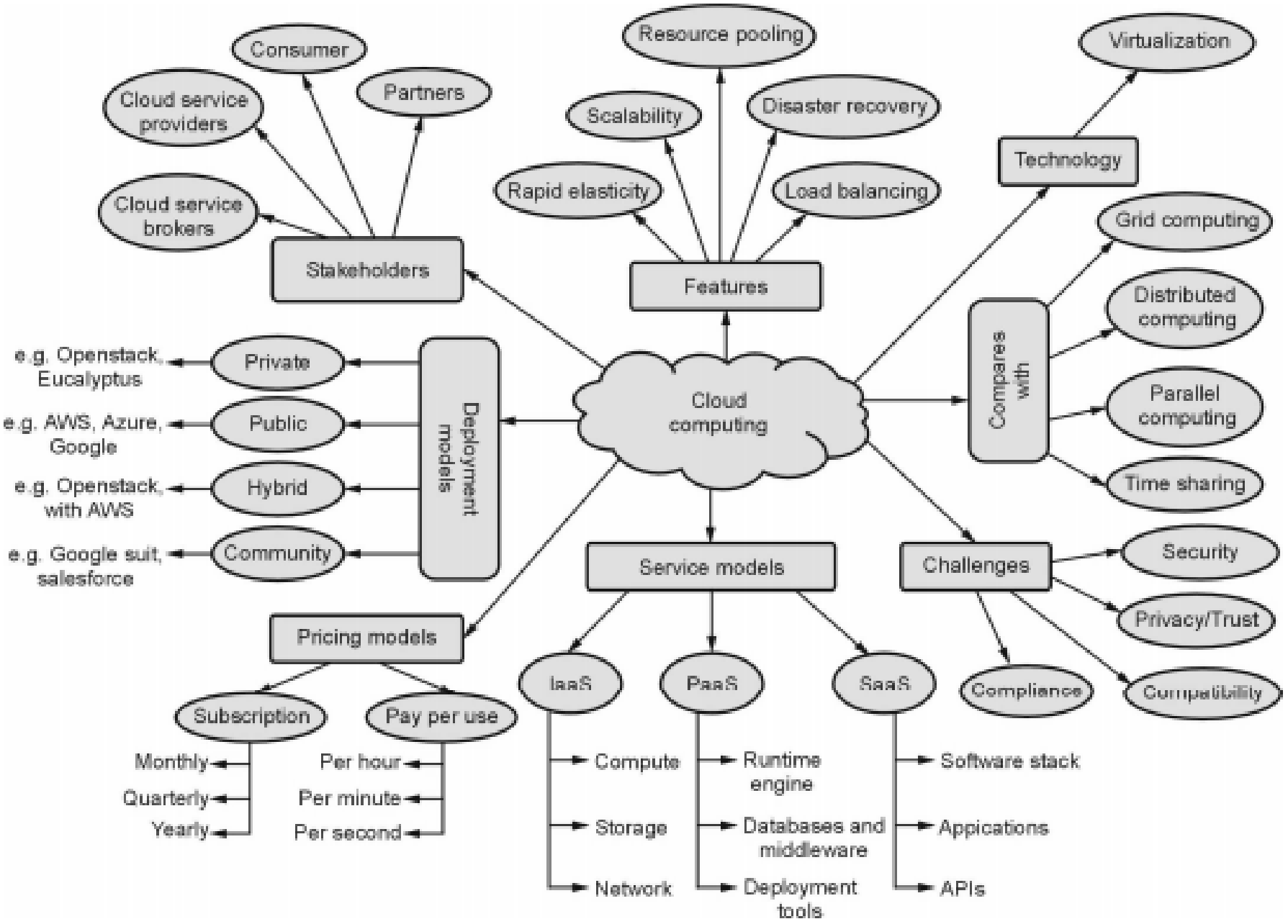
Network access: Services are delivered over a network (usually the internet), enabling remote access from anywhere.

# Various aspects of cloud computing

The various aspects of cloud computing are

1. Their service models,
2. Deployment models,
3. Pricing models,
4. Stakeholders,
5. Ttechnology used along with features and challenges

# Various aspects of cloud computing



# Cloud Computing and Other Similar Configurations

The cloud computing is often compared with other computing architectures like

1. Peer-to-Peer,
2. Client-Server,
3. Grid computing,
4. Distributed computing,
5. Cluster computing.

# Peer to Peer Architecture

Peer-to-Peer architecture has collection of hosts connected in a network intended for resource sharing, task processing, and communication.

Each host on the network act as a server and has equal right in terms of providing and using resources.

In peer to peer architecture, there is no controller or server.

limitations:

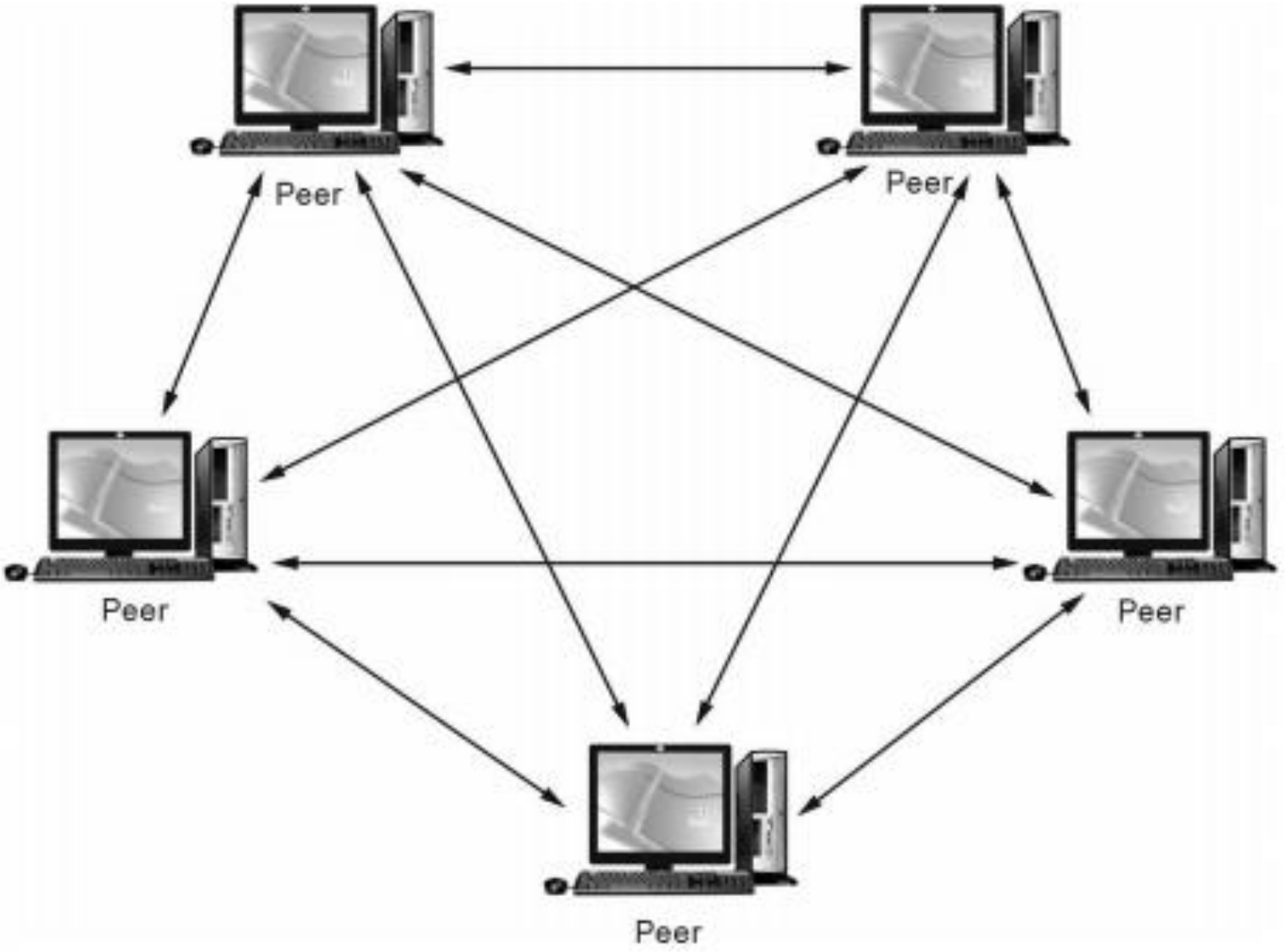
1.Lack of scalability,

2.Poor performance,

3.Low throughput,

4.Limited flexibility etc.

# Peer to Peer Architecture



# Client Server Architecture

In Client-server architecture, there is at least one specialized server which controls the communication between multiple clients.

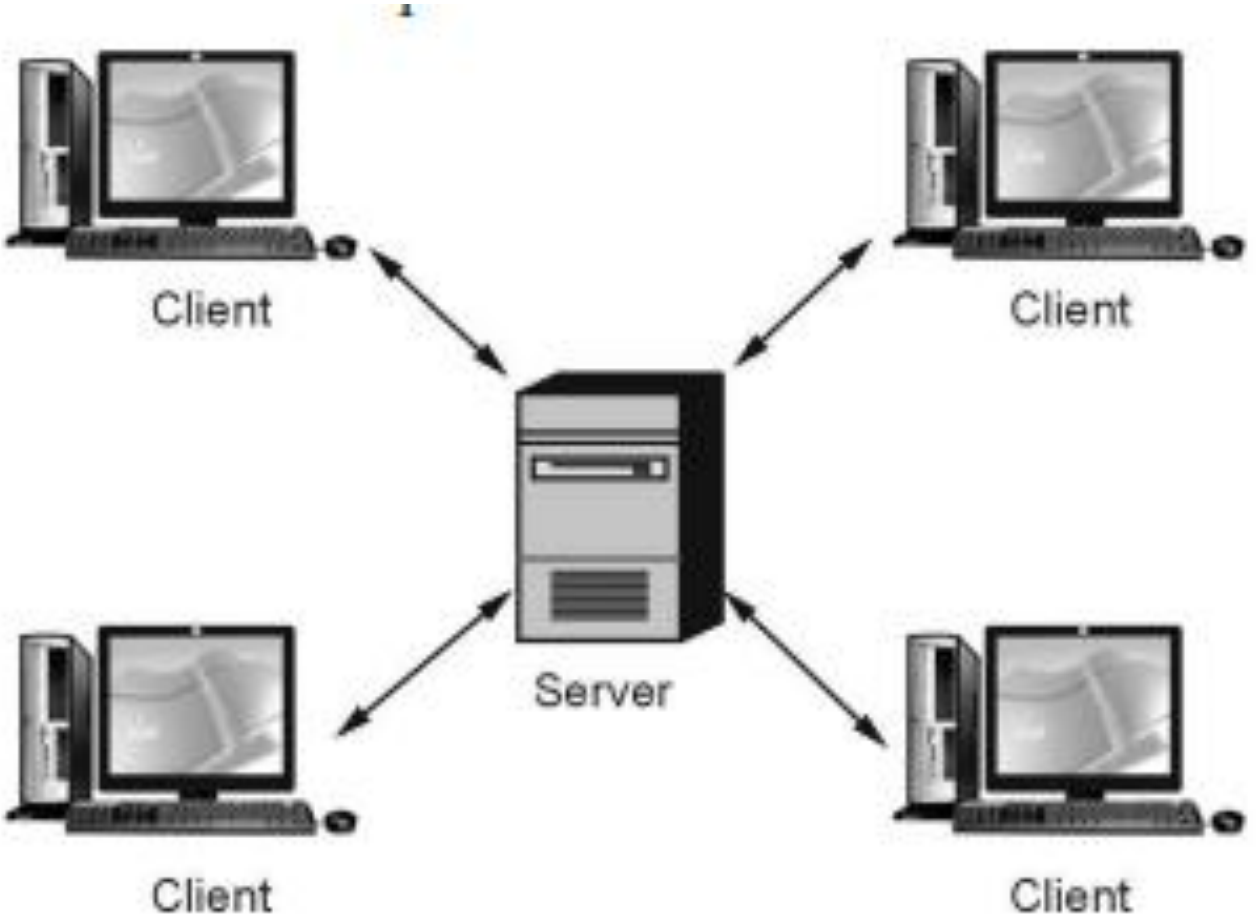
Typically, there is a server called controller.

The server is responsible for handling the resource sharing, task processing and communications between the clients.

So, clients have to rely on server for various access and services.

The client server architecture has centralized processing which makes faster communication with good performance.

# Client Server Architecture



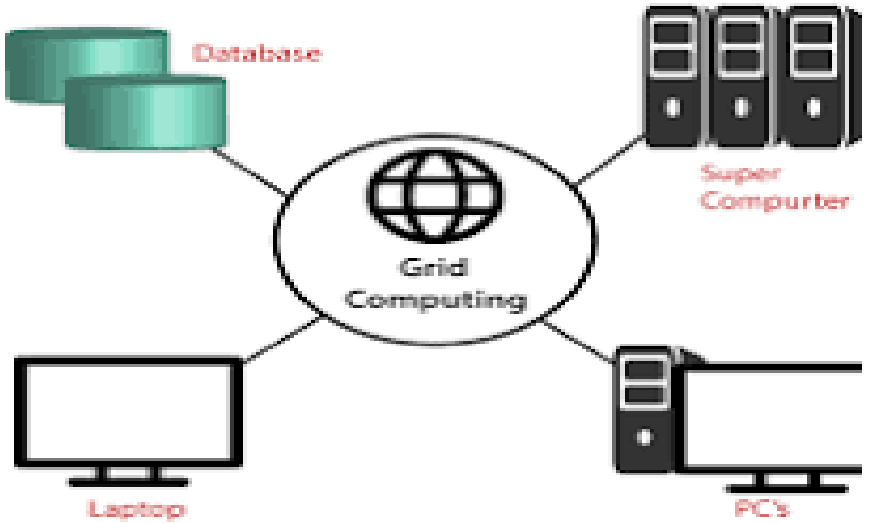
# Grid Computing

The Grid Computing architecture has geographically distributed computing resources which work together to perform a common task.

A typical grid has pool of loosely coupled computers who worked together to solve a complex computational problem.

It has heterogeneous resources which are controlled by a common node called control node like client server architecture.

# Grid Computing




**Grid Computing**

- Networked, heterogeneous resources
- Generally decentralized
- Resources can be owned by multiple entities
- Very high scalability across networks

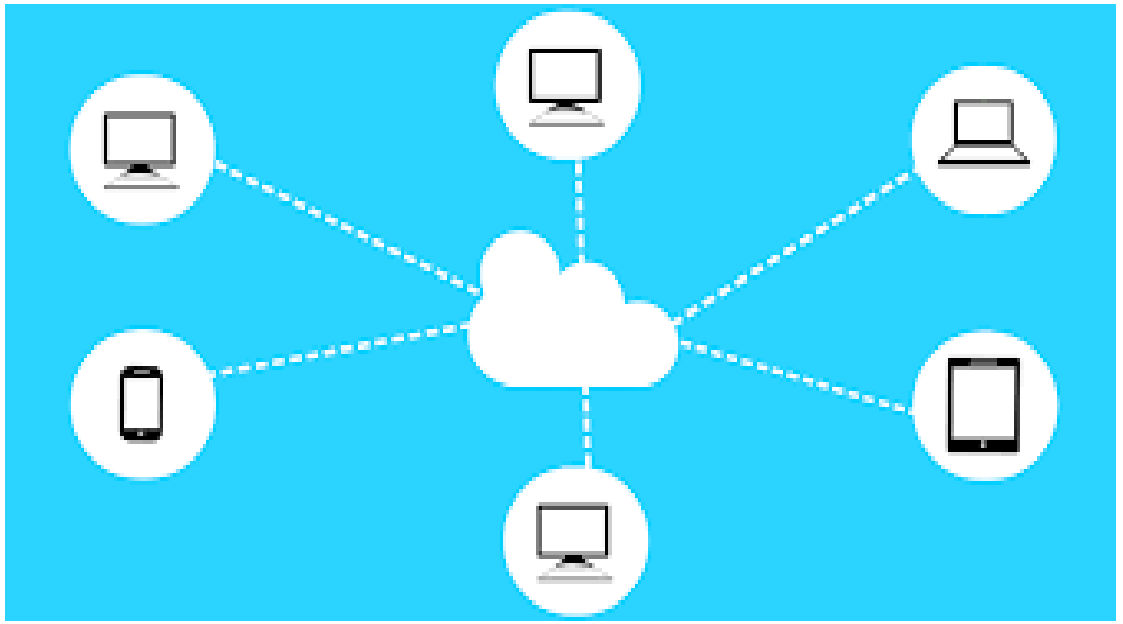
# Distributed computing

It is a computing concept that refers to multiple computer systems working on a singleproblem.

In distributed computing, a single problem is divided into many parts, and each part is executed by different computers.

As long as the computers are networked, they can communicate with each other to solve the problem.

# Distributed computing



**Distributed Computing**

- Independent nodes
- Centralized or decentralized
- Often owned by a single entity
- High scalability with added nodes

# Cluster computing

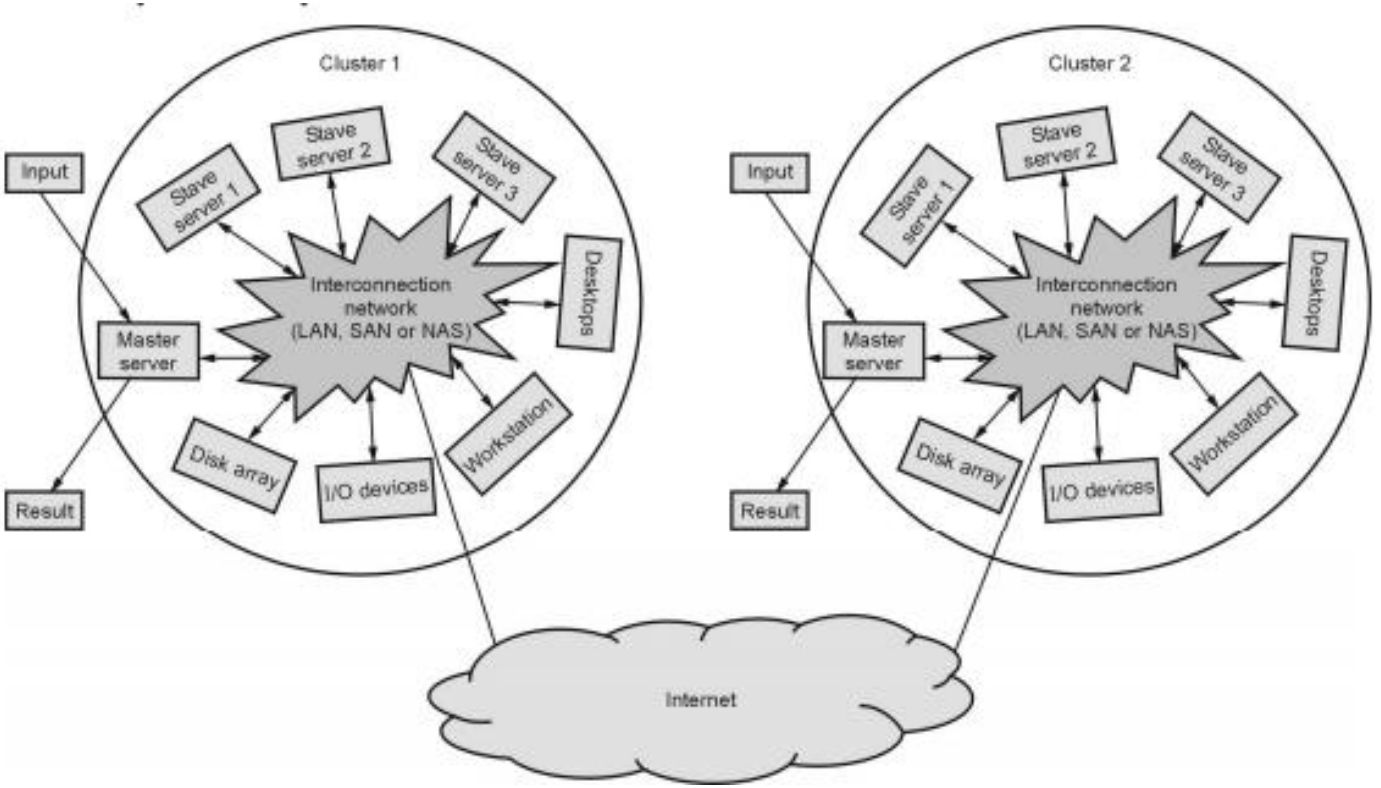
Cluster computing is also intended to solve a complex computational problem in a group of computers connected through a network.

Generally, the cluster is a collection of interconnected loosely coupled homogenous computers that work together closely, so

that in some respects they can be considered as a single computer.

Each cluster is composed of multiple standalone machines connected by a network.

# Cluster computing



# Evolution of Cloud Computing

The evolution of cloud computing with respect to hardware, internet, protocol, computing and processing technologies.

## Evolution of Hardware:

**First Generation (1930s–1940s):**

The first-generation computers like **Mark** and **Colossus** were developed in the 1930s and 1940s. They used **vacuum tubes** and **punch cards** to store and process data. These machines could perform simple binary arithmetic and were large and slow. Harvard University built an electromechanical computer in 1943, which was programmable.

**Second Generation (1940s–1950s):**

The **ENIAC**, built in 1946, was a major breakthrough. It used **thermionic valves** and early **transistors**. ENIAC could perform around **100,000 calculations per second**, making it much faster than previous computers.

**Third Generation (1958–1970s):**

This generation introduced **integrated circuits (ICs)**. IBM developed the **IBM 360 mainframe**, which offered better processing and storage. **Minicomputers** also appeared. Later, Intel released the first **microprocessor (Intel 4004)**, which allowed multiple transistors on one chip.

**Fourth Generation (1980s–Present):**

Computers now used **microprocessors** and **RAM** to execute **millions of instructions per second**. IBM launched the first **personal computer (PC)** in 1981. This era saw the development of **LSI (Large-Scale Integration)** and **VLSI (Very-Large-Scale Integration)** microchips, making computers smaller, faster, and more affordable.

# Evolution of Internet and Protocols

The idea of the internet began in the **1930s** with the concept of **MEMEX**, which was meant to store books, records, and communication. In **1957**, after the Soviet Union launched a satellite, the U.S. created **ARPA** (Advanced Research Project Agency) for military research. In **1967**, **ARPANET** was developed using **Internet Message Processors (IMPs)** for communication between sites.

At first, ARPANET used basic **host-to-host protocols**, which later evolved into **FTP** and **SMTP** in **1983**. The powerful **TCP/IP** protocol was also introduced and is still used today. The original internet protocol was **IPv4**, later updated to **IPv6**.

In **1990**, **Tim Berners-Lee** developed the first web browser, **MOSAIC**, followed by **Netscape** in **1994**. In **1995**, Microsoft released **Windows 95** with a built-in browser called **Internet Explorer**, supporting **dial-up TCP/IP**. In **1996**, the first web servers using **HTTP** were launched, followed by servers and browsers that supported scripting languages.

# Evolution of Computing Technologies

A few decades ago, **Cluster Computing** was used to solve large and complex problems. It worked by dividing the task into smaller parts and processing them using multiple connected computers. In the 1990s, this evolved into **Grid Computing**, introduced by **Ian Foster**. Grid computing connected independent computers across cities, countries, or continents to solve a single problem, similar to how people use electricity from a power grid. However, grid computing had a major issue—**data was spread across distant locations**, which made it harder to manage.

To fix this, **Cloud Computing** was developed. It uses centralized data centers to provide computing services, similar to grid computing but more organized and efficient. Cloud computing became more popular with the use of **Virtualization**. Virtualization allows multiple virtual systems to run on one physical machine. This reduces hardware costs, improves performance, and makes better use of available resources.

# Evolution of Processing Technologies

Early computers used **mechanical devices**, **vacuum tubes**, and **transistors**. With new chip technologies like **SSI**, **MSI**, **LSI**, and **VLSI**, circuits became smaller, faster, and more reliable. This helped improve processors and their performance.

There are two types of processing: **serial** and **parallel**. In serial processing, tasks are done step-by-step by one processor. In **parallel processing**, tasks run at the same time on multiple processors to save time.

Later, **multiprogramming** allowed several programs to run at once, each using the processor for a short time in turns. This evolved into **vector processing**, designed for handling **matrices and arrays** in tasks like scientific computing.

Then came **symmetric multiprocessing (SMP)**, where multiple processors share tasks equally, avoiding the master-slave system. This improved task handling and system balance.

Finally, **massively parallel processing (MPP)** connected many processors to work as one large system. Today, **single-chip MPP** is widely used in high-end applications like **artificial intelligence**, thanks to advanced integrated circuits.

# Underlying Principles of Parallel and Distributed Computing

Computing in computer technology can be defined as the execution of single or multiple programs, applications, tasks or activities, sequentially or parallelly on one or more computers.

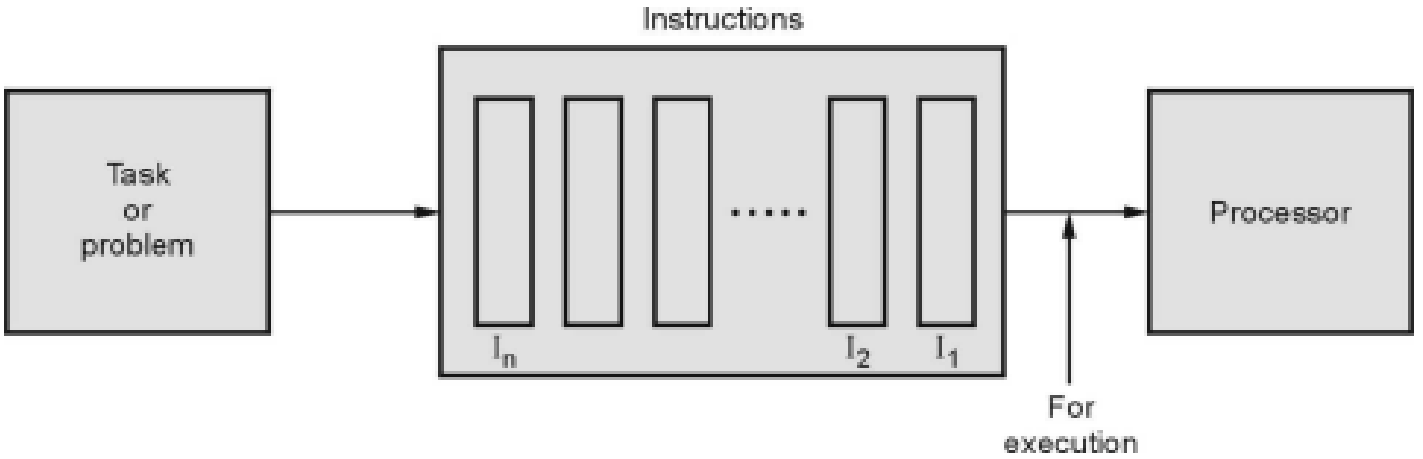
The two basic approaches of computing are serial and parallel computing.

**Serial Computing :**

In serial computing, the given problem or task is broken into a discrete series of instructions. These instructions are executed on a single processor sequentially.

The problem in serial computing is that it executes only one instruction at any moment of time.

It is mostly used in monolithic applications on single machine which do not have any time constraint.



# Parallel Computing

**Single processor systems** are now outdated for **real-time applications** that need fast computing. To solve this, **parallel computing** is used. It helps speed up the execution of tasks by using **multiple processors** at the same time.

In parallel computing, a big problem is divided into smaller parts. These parts are solved **at the same time (concurrently)** by different processors. Each part is also broken down into instructions that run together, with a **central system** managing all processors.

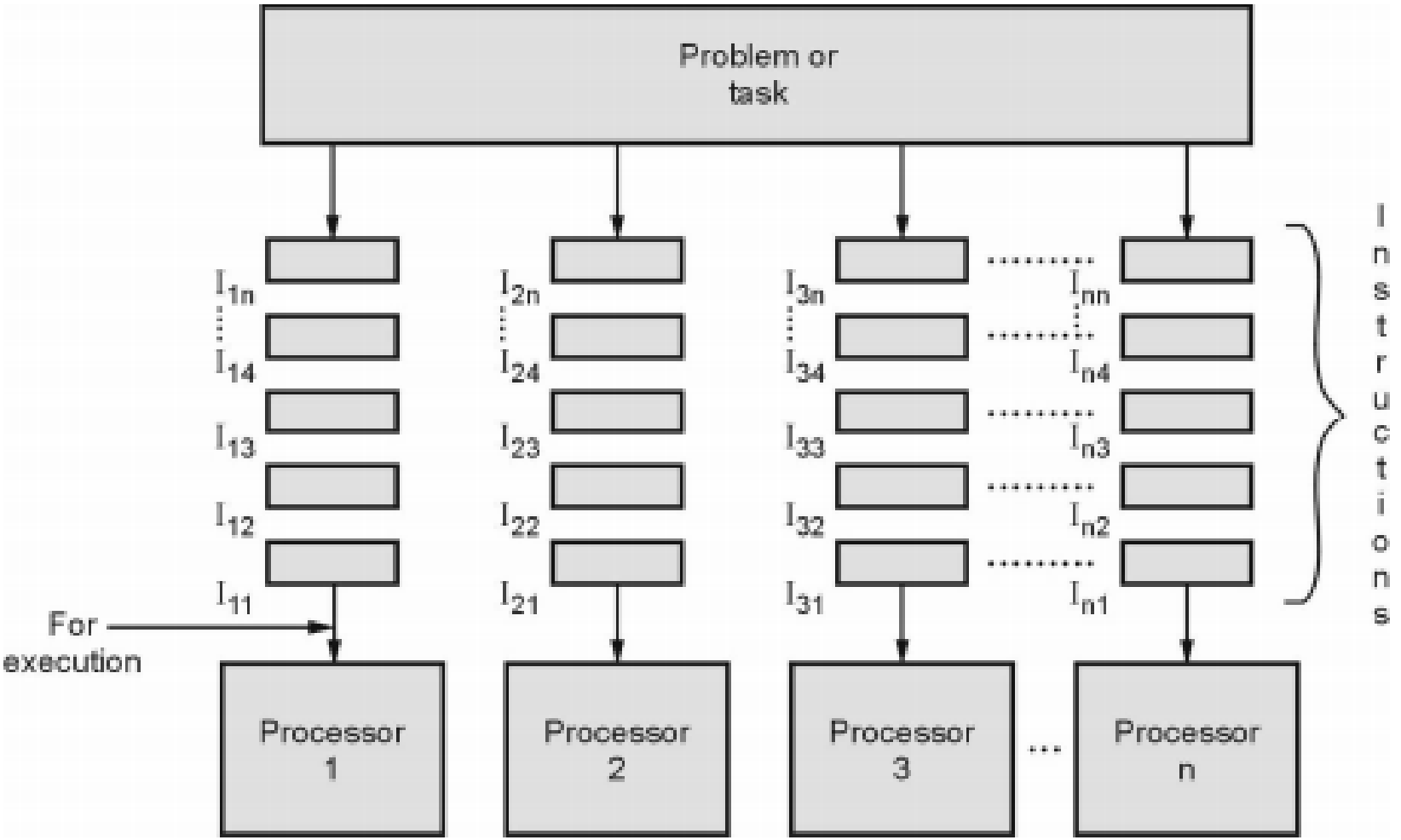
This method allows the **workload to be shared**, which gives much better performance than using just one processor.

Parallel computing is closely related to **parallel processing** and **parallel programming**.

- **Parallel processing** means running multiple tasks or subtasks at the same time on different processors.
- **Parallel programming** is writing programs that use a **divide-and-conquer** approach. The main task is split into subtasks, and each one is processed separately on different processors.

This improves speed and efficiency for large or complex tasks.

# Parallel Computing



# Hardware Architectures for Parallel Processing

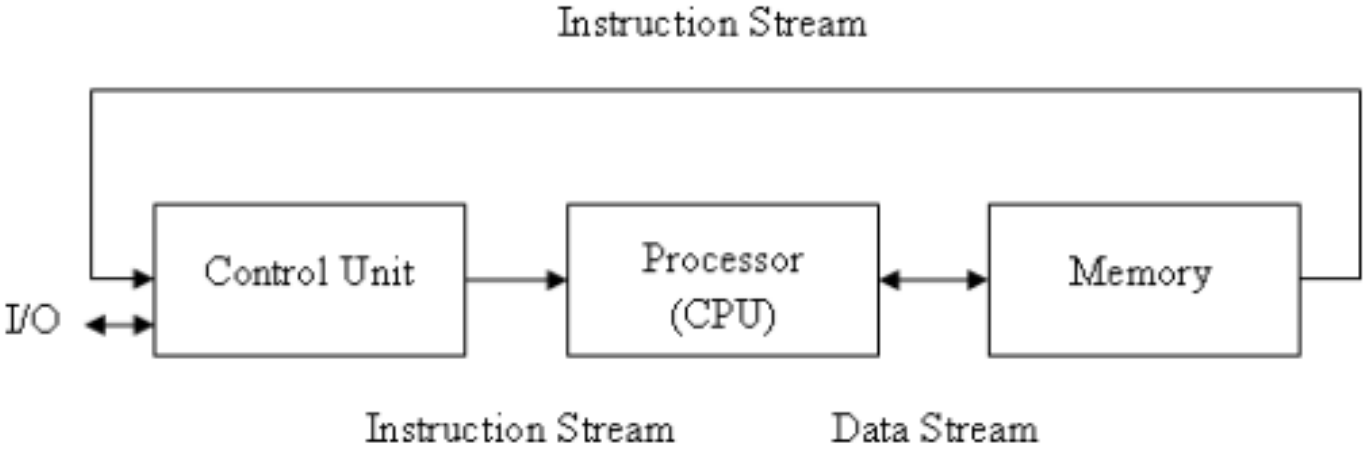
The core elements of parallel processing are CPUs.

Based on the number of instruction streams and data streams that can be processed simultaneously, computing systems are classified into four categories proposed by Michael J. Flynn in 1966.

| | |
|---|---|
| **SISD**<br>Single instruction stream<br>single data stream | **SIMD**<br>Single instruction stream<br>multiple data stream |
| **MISD**<br>Multiple instruction stream<br>single data stream | **MIMD**<br>Multiple instruction stream<br>multiple data stream |

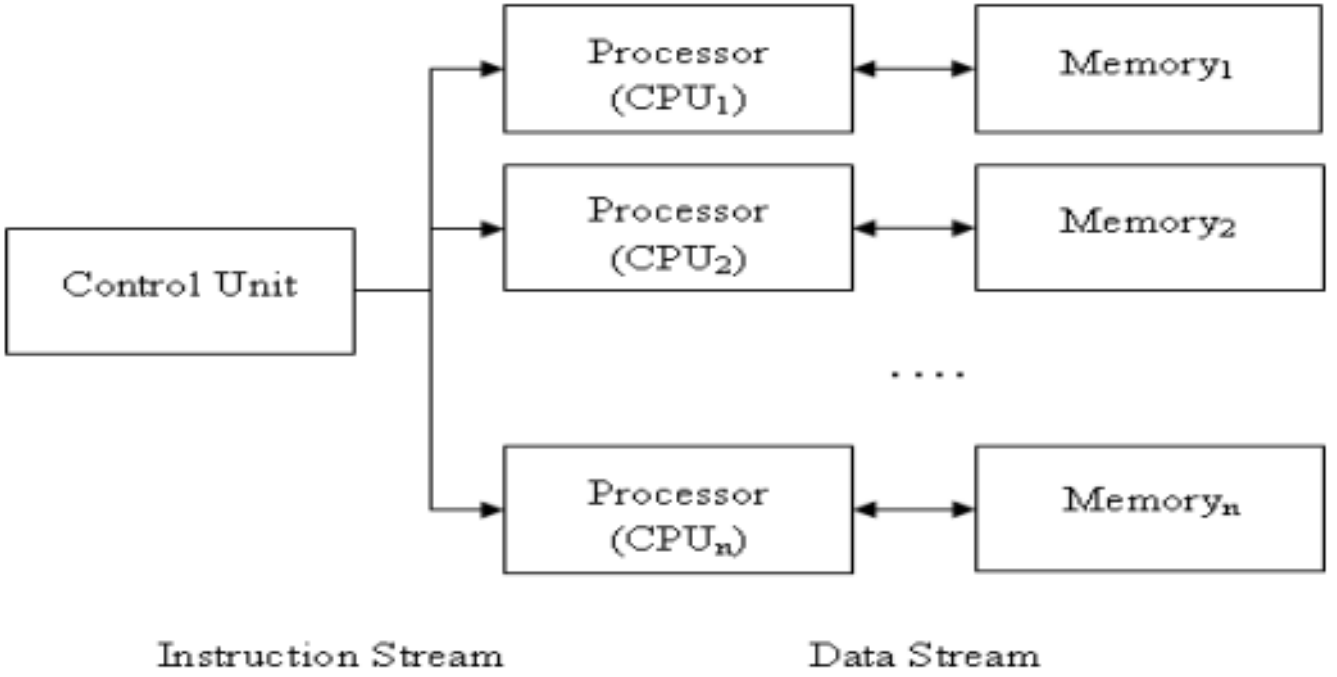
# 1.Single Instruction Single Data systems (SISD)

An SISD computing system is a uniprocessor system capable of executing a single instruction, which operates on a single data stream.



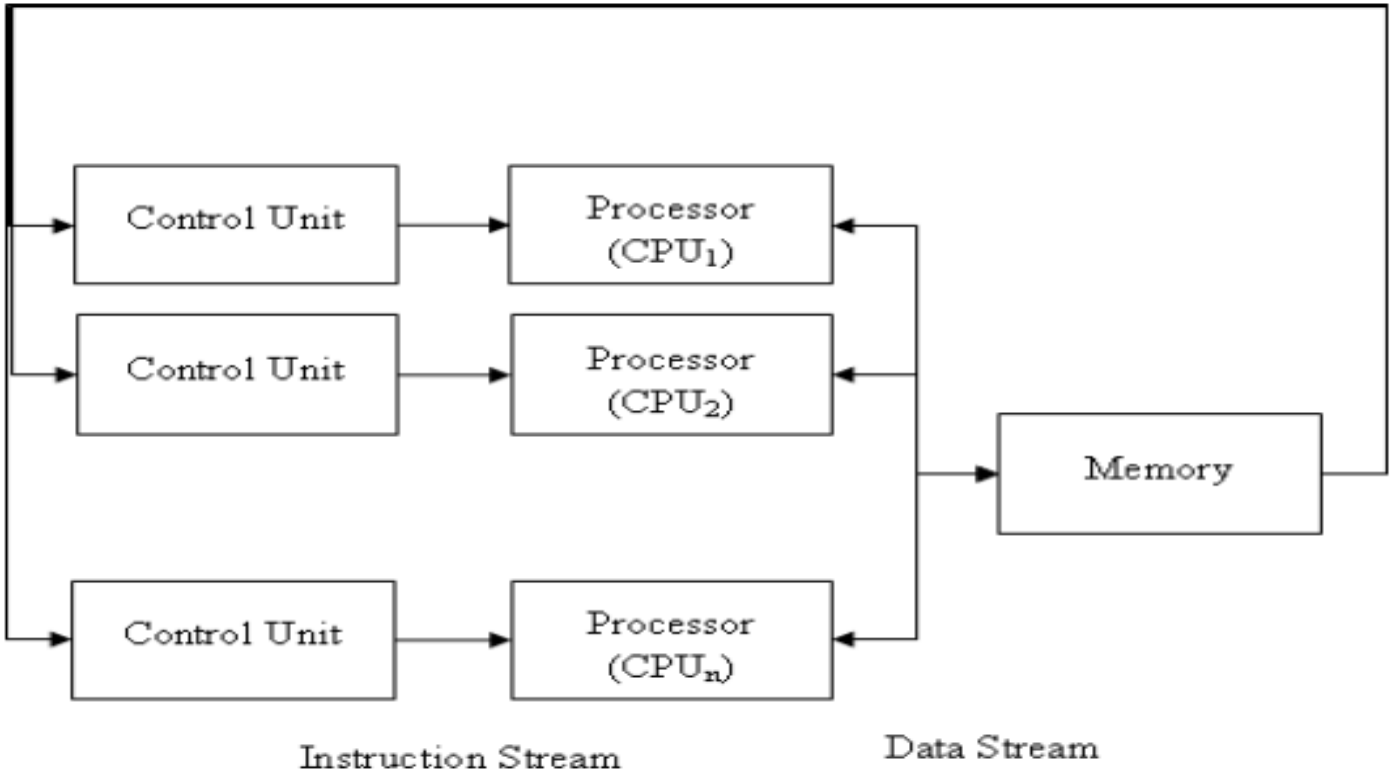
# 2.Single Instruction Multiple Data systems (SIMD)

An SIMD computing system is a multiprocessor system capable of executing the single instruction on all the CPUs but operating on different data streams.



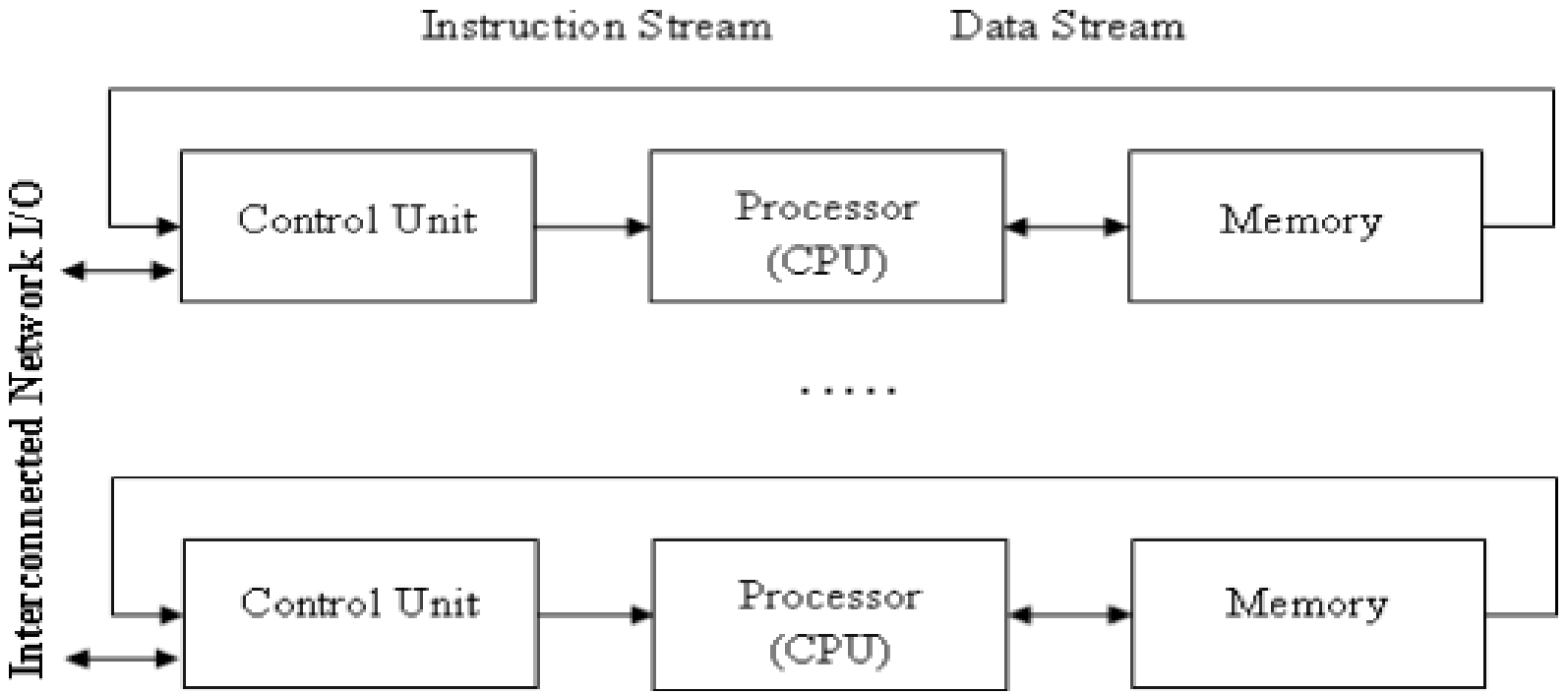
# 3.Multiple Instruction Single Data systems (MISD)

An MISD computing system is a multiprocessor system capable of executing different instructions on different processing elements but all of them operating on the same data streams.



# 4.Multiple Instruction, Multiple Data systems (MIMD)

An MIMD computing system is a multiprocessor system capable of executing multiple instructions on multiple data streams.

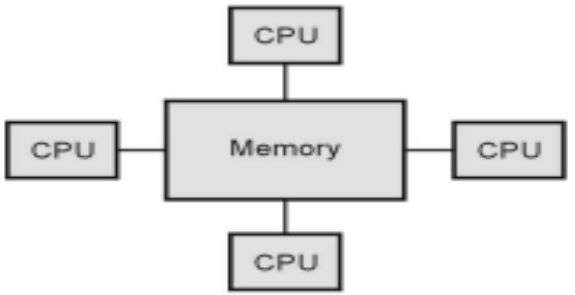


# Shared Memory Architecture for Parallel Computers

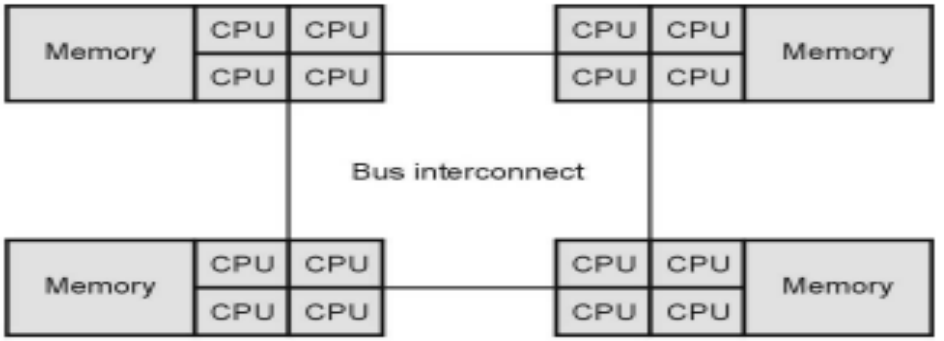
In **shared memory architecture**, there are **multiple processors**, and all share the **same memory space**. Each processor works independently but uses the **shared memory**. If one processor updates a memory location, all others can see the change.

This system is divided based on **memory access time** into two types:

1. **Uniform Memory Access (UMA):**
   In UMA, **all processors are equal** and share memory with the **same access speed**. The processors are connected using a **shared bus**, and the memory access time is **uniform**. UMA is also called **symmetric multiprocessing (SMP)**.
2. **Non-Uniform Memory Access (NUMA):**
   In NUMA, each processor has its **own local memory**. Accessing **local memory** is faster, but accessing memory from another processor (remote memory) is **slower**. This is because processors are linked through an **interconnection network**, and not all processors get equal memory access speed.



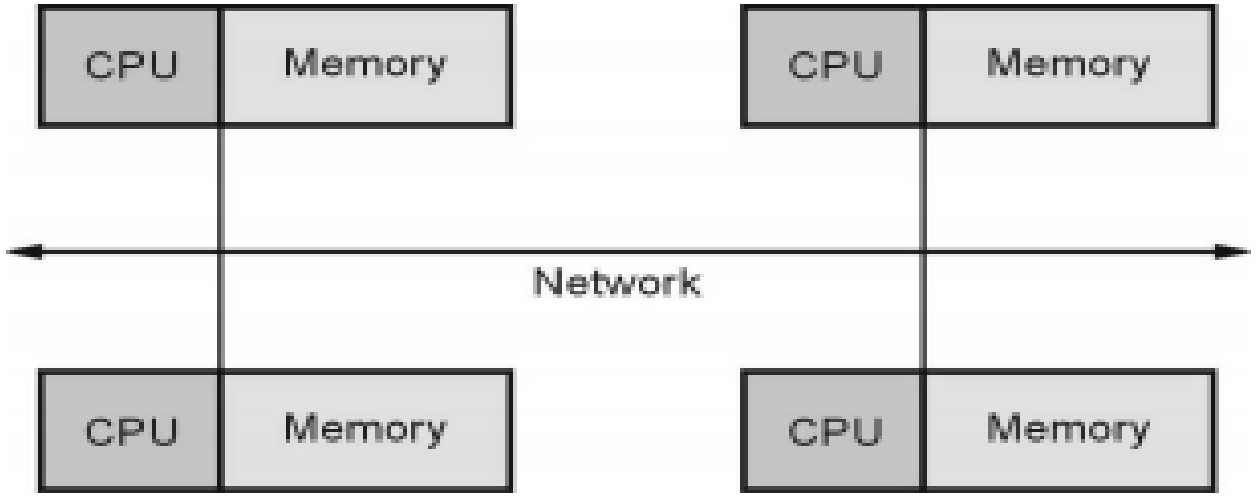

(a) UMA architecture




(b) NUMA architecture.

# Distributed Memory Architecture for Parallel Computers

In a **distributed memory system**, each processor has its **own local memory**, and there is **no shared global memory**. Any changes made by one processor do **not affect** the memory of other processors, and memory addresses are **not shared** between them. To exchange data, processors communicate through a **network connection**. This type of system is also known as a **message passing architecture**, where data is passed between processors using messages. The **performance and speed** of the system depend on how efficiently the **processors are connected** and how well they communicate with each other.



# Architectural Models for Distributed System

In a **distributed system**, many computers are connected to solve tasks together. Two main models are used: the **Client-Server model**, where clients request services from a central server, and **Peer-to-Peer (P2P)**, where all systems are equal and share resources. P2P can be complex and hard to scale.

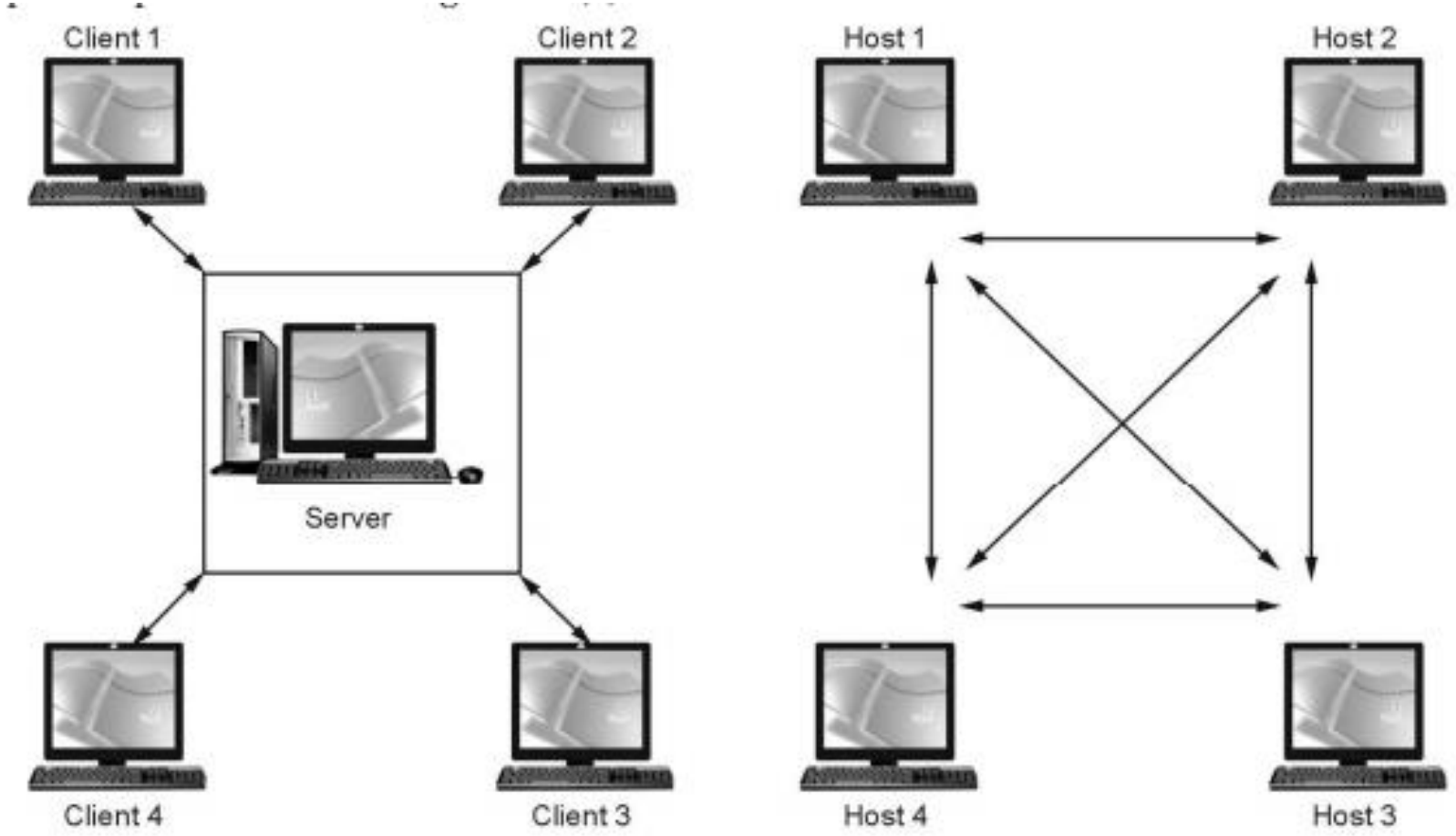
**Interprocess Communication (IPC)** lets distributed processes communicate. Key IPC models include:

- **Remote Procedure Call (RPC):** Calls a function on a remote computer as if it's local.
- **Message Oriented Middleware (MOM):** Uses queues to send and receive messages, even when the receiver is offline (e.g., email).
- **Remote Method Invocation (RMI):** Java-based model for calling remote objects but lacks support for different platforms.

**Service-Oriented Architecture (SOA)** is used to build applications from reusable services.
**Web Services** communicate using XML and **SOAP**, with services described by **WSDL** and discovered using **UDDI**.

**Cloud computing** is built on **Web 2.0** and these service models to deliver scalable, platform-independent services.

# Architectural Models for Distributed System




(a) Client-Server model

(b) Peer-to-Peer model

# Cloud Characteristics

The **NIST** defines five main characteristics of cloud computing:

1. **On-demand self-service** – Users can access computing resources like servers and storage automatically, without asking the provider.
2. **Broad network access** – Cloud services are available over the internet and can be used on phones, laptops, tablets, and other devices.
3. **Resource pooling** – Resources like memory and bandwidth are shared among users using a **multi-tenant** model, with no knowledge of the physical location.
4. **Rapid elasticity** – Resources can be quickly increased or reduced based on user demand, appearing unlimited to users.
5. **Measured service** – Usage is tracked and reported for billing and transparency using **metering tools**.

Other features include:

- Use of **open REST APIs** for easy web access
- **Location independence** and **remote access** anytime, anywhere
- **Agility** for fast resource use and reuse
- **User control** over resources (end-user computing)
- Support for **multi-tenancy**, offering scalability, reliability, and security

# Elasticity in Cloud

**Elasticity** is a key feature in cloud computing that allows resources like **CPU, memory, storage, and bandwidth** to be **automatically increased or decreased** based on demand. This is very useful for **critical business applications**, where performance cannot be compromised. When more users access an application, the system adds more resources. When usage drops, it reduces them, saving cost.

Elasticity supports **horizontal scaling** (adding/removing resources) and is common in **public cloud** with **pay-as-you-go** pricing. It helps in making the best use of infrastructure and cutting down on expenses.

For example, in an office, resources are scaled up during **peak hours (9 AM–9 PM)** and scaled down at night to reduce bills. Another example is **IRCTC**—earlier its website crashed during **Tatkal bookings**, but now, with elasticity, servers auto-scale to handle high traffic, improving **performance and reliability** for users.

Elasticity depends on the environment and must be managed carefully for apps needing stable performance.

# Elasticity in Cloud

Definition: The ability of a system to automatically increase or decrease resources as needed.

Elastic vs. Scalable: Elasticity is dynamic, scalability can be manual.

Example: Auto-scaling of servers during high traffic on an e-commerce website.

# On-demand Provisioning

**On-demand provisioning** is a key feature of cloud computing that allows users to **automatically access and use cloud services** whenever needed. These services are provided by **Cloud Service Providers (CSPs)** through a **self-service portal** over the internet. Users can select and deploy resources like **hardware, software, data, or servers** with customized settings as per their needs.

This feature helps enterprises handle **fluctuating resource demands** without manual setup. It allows them to **add or remove resources** like storage or computing power instantly. On-demand provisioning supports various **operating systems and software stacks**, making it flexible and scalable.

It enables businesses to **access services anytime, from anywhere**, using any supported device, saving time and cost while improving efficiency.

# Challenges in Cloud Computing

**Cloud computing faces several challenges:**

**Data protection** is crucial since data is stored remotely by third parties, requiring strong encryption.

**Data recovery** is hard because data is spread across multiple data centers, and users don't always know where their data is. Downtime affects service availability, which is governed by Service Level Agreements (SLAs).

**Regulatory compliance** restricts sharing data across borders, forcing providers to set up local data centers.

**Management difficulties** arise when using multiple cloud providers.

**Interoperability issues** occur because many providers use proprietary systems, making it hard to switch services.

Cloud computing enables on-demand, scalable, and elastic resource use. It supports multiple architectures and communication models like RPC, MOM, and RMI. Web services and SOA form its foundation, with elasticity and on-demand provisioning helping adapt to workload changes efficiently.

**Cloud Computing**

**Definition:**

Cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale.

**Key Points:**

On-demand availability

Pay-as-you-go pricing

Internet-based service delivery

## Features of Cloud Computing

- Scalability
- Elasticity
- Resource pooling
- Broad network access
- Measured service

# Deployment Models

- Public cloud
  - Done by service providers
- Community cloud
  - organizations from a specific community with common concerns
- Private cloud
  - operated solely for a single organization
- Hybrid cloud
  - composition of two or more clouds (private, community or public)
- Private Cloud Rentals
  - option to consider when security is a concern

# Architecture

- The software systems involved in the delivery, communicating over a loose coupling mechanism
- The Inter-cloud
  - The Inter-cloud is an interconnected global "cloud of clouds" and an extension of the Internet

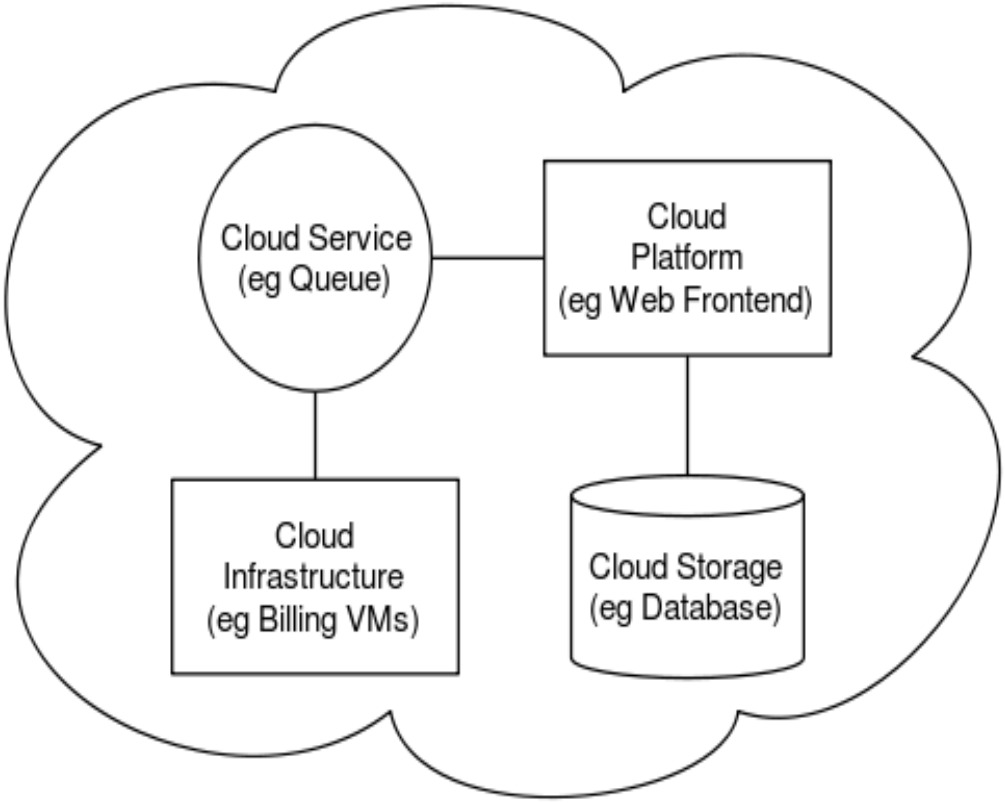



Fig. 4 Cloud Architecture [8]

## Types of Cloud Deployment Models

**Public Cloud** – Services are delivered over the public internet.

**Private Cloud** – Services are maintained on a private network.

**Hybrid Cloud** – Combination of public and private.

# Service Models

- Infrastructure as a Service (IaaS)
  - Basic, service users maintain software
- Platform as a Service (PaaS)
  - Users are given software and hardware automatically
- Software as a Service (SaaS)
  - All software and hardware is transparent
  - User only knows their own access point

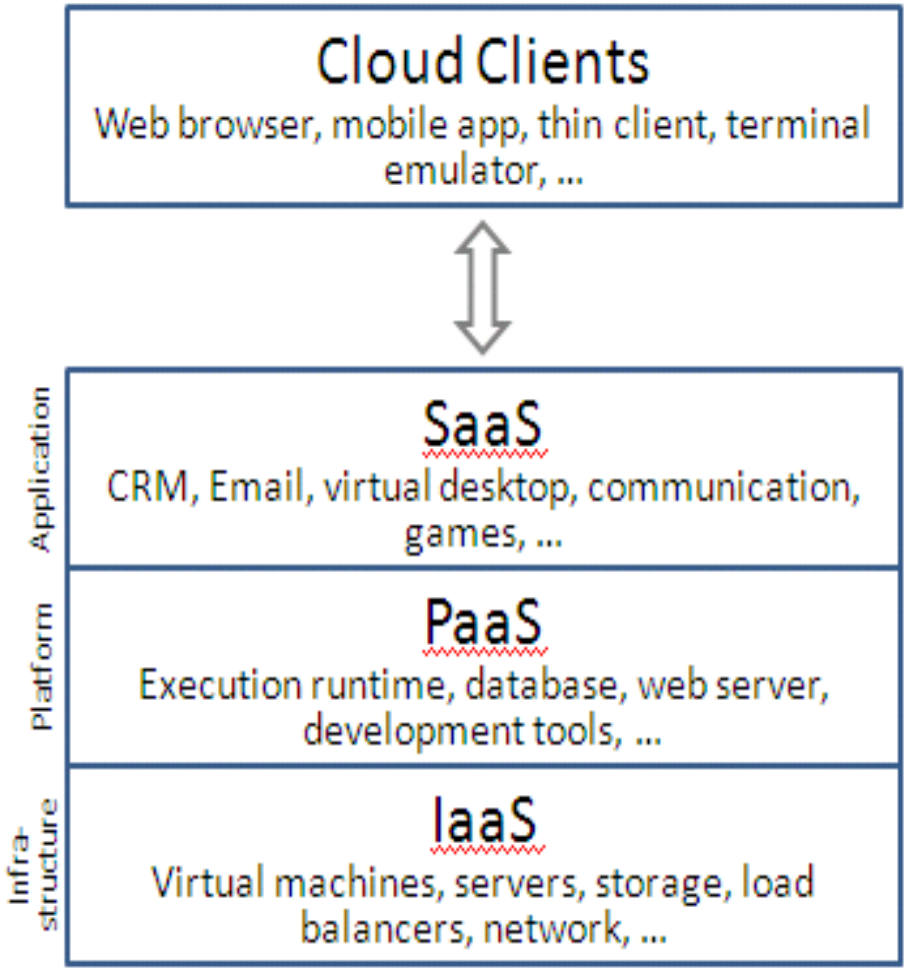



Fig. 3 Service Models of Cloud Computing [7]

# Cloud Computing Components

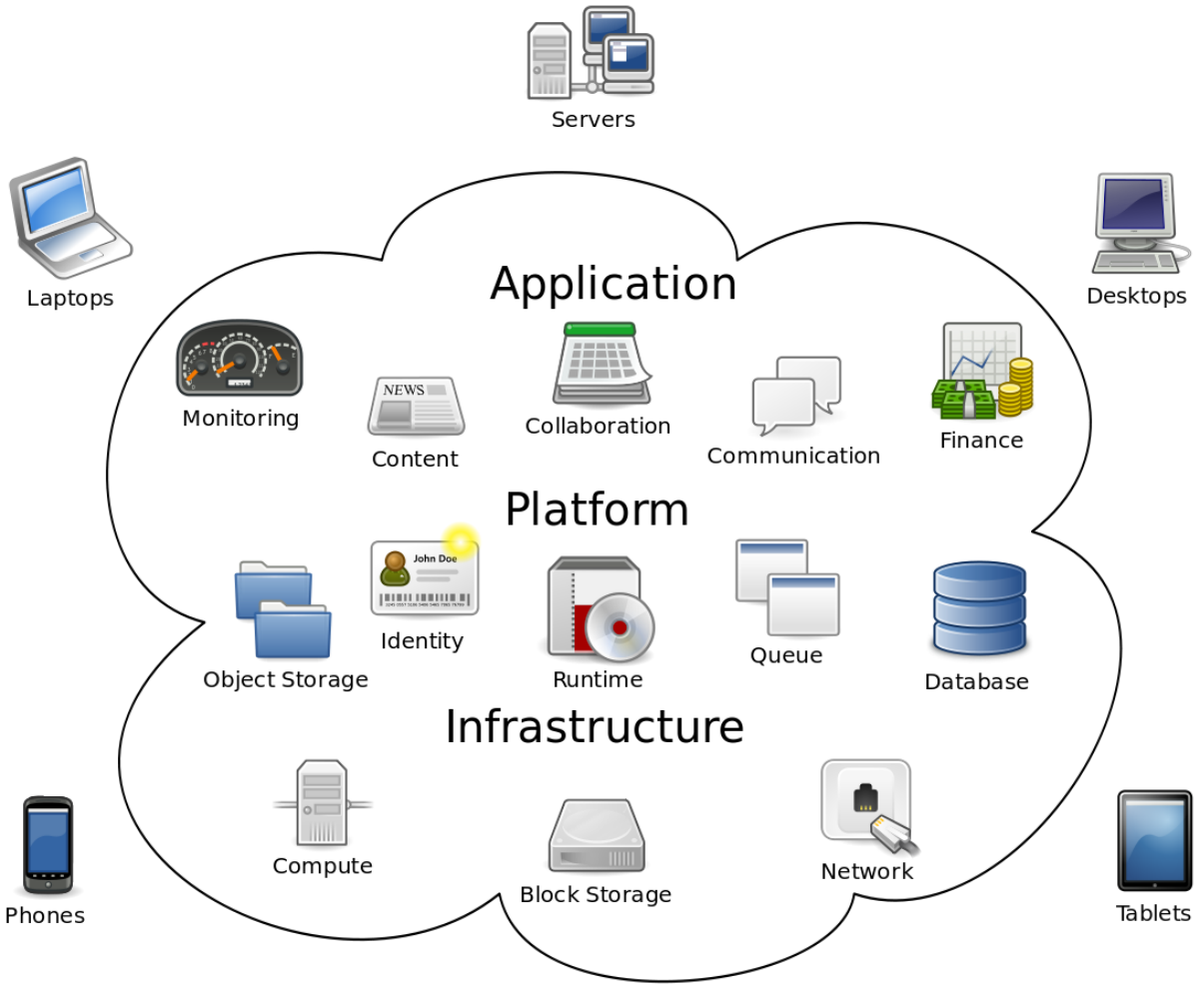



Fig. 1 Cloud Computing Components [5]

# Cloud Service Models

**IaaS** – Infrastructure as a Service (e.g., AWS EC2)

**PaaS** – Platform as a Service (e.g., Google App Engine)

**SaaS** – Software as a Service (e.g., Gmail, Microsoft 365)

# Evolution of Cloud Computing

**Timeline Overview:**

**1950s:** Mainframe computing with dumb terminals.
**1960s-70s:** Virtualization and time-sharing concepts.
**1990s:** Rise of internet and web-based services.
**2006:** Amazon launches AWS, kickstarting modern cloud era.
**2010s:** Rapid adoption by businesses, emergence of hybrid cloud.
**2020s:** Edge computing, AI integration, and serverless computing.

# Benefits of Cloud Computing

Cost efficiency

Flexibility and scalability

Business continuity

Automatic updates

Collaboration efficiency

# Challenges in Cloud Computing

Data security and privacy

Downtime and internet dependency

Compliance and legal risks

Vendor lock-in

## Real-World Applications

Cloud storage (Google Drive, Dropbox)

Streaming services (Netflix, Spotify)

Online collaboration tools (Slack, MS Teams)

E-commerce platforms (Shopify, Amazon)

# Parallel and Distributed Computing

- **Parallel Computing**

Executing multiple processes **simultaneously** on **multiple processors or cores** within the same machine.

- **Distributed Computing**

Executing processes on **multiple machines** connected over a network that work **collaboratively**.

# Principles of Parallel Computing

**Concurrency** – Tasks operate at the same time.

**Speedup** – Faster processing through multiple processors.

**Scalability** – Performance improves with more resources.

**Synchronization** – Ensures tasks work in coordination.

# Principles of Distributed Computing

**Transparency** – Users see a single system.

**Fault Tolerance** – System continues even if parts fail.

**Resource Sharing** – Multiple systems share files, processing power.

**Scalability** – Easily expand system by adding machines.

# Parallel vs. Distributed Computing

| Feature | Parallel Computing | Distributed Computing |
| --- | --- | --- |
| System | Single machine | Multiple machines |
| Communication | Shared memory | Network communication |
| Example | GPU processing | Cloud services, Hadoop |

## Connection to Cloud Computing

Cloud computing combines both **parallel** and **distributed** principles to offer:

- High availability
- Fault tolerance
- On-demand scalability
- Efficient resource utilization

# Essential Cloud Characteristics (as per NIST)

**On-Demand Self-Service**

Users provision resources as needed automatically.

**Broad Network Access**

Accessible over the internet from various devices.

**Resource Pooling**

Shared resources dynamically allocated across users.

**Rapid Elasticity**

Automatically scale resources up/down.

**Measured Service**

Resource usage is monitored, controlled, and billed.

# Additional Cloud Characteristics

**Multi-tenancy**

Multiple users share the same infrastructure securely.

**Automation**

Cloud processes are automated for speed and accuracy.

**Resiliency**

Systems are designed to recover quickly from failures.

# Introduction to Cloud Computing

Definition:

Delivery of computing services (servers, storage, databases, networking, software) over the Internet ("the cloud").

Benefits: Cost-efficiency, scalability, flexibility, accessibility.

Deployment Models: Public, Private, Hybrid.

Service Models: IaaS, PaaS, SaaS.

# Characteristics

- Empowerment
- Agility and API
- Cost and Security
- Device and location independence
- Visualization
- Multi-tenancy
- Reliability and Maintenance
- Scalability and Elasticity

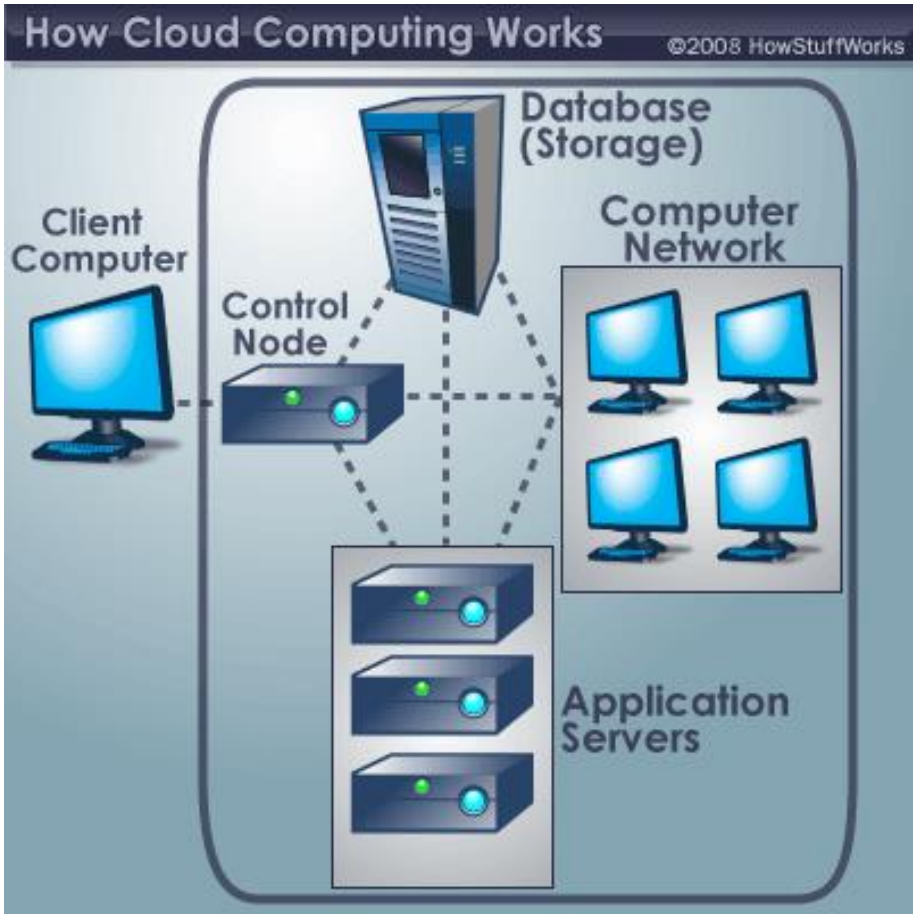



Fig. 2 How Cloud Computing Works [6]

# Characteristics of Cloud Computing

On-Demand Self-Service

Broad Network Access

Resource Pooling

Rapid Elasticity

Measured Service

# Elasticity in Cloud

Definition: The ability of a system to automatically increase or decrease resources as needed.

Elastic vs. Scalable: Elasticity is dynamic, scalability can be manual.

Example: Auto-scaling of servers during high traffic on an e-commerce website.

## Types of Elasticity

Vertical Elasticity: Adding/removing resources to a single server (CPU, RAM).

Horizontal Elasticity: Adding/removing servers to distribute load.

Automatic Elasticity: Done via policies and triggers.

## Benefits of Elasticity

Cost efficiency (pay-per-use).

High availability and performance.

Better user experience.

Disaster recovery and fault tolerance.

# On-Demand Provisioning

Definition: Cloud resources are provided when users need them without human intervention.

Based on real-time demand.

Enables agility and responsiveness.

## How On-Demand Provisioning Works

Triggered by users or system metrics.

Resources provisioned via APIs or management dashboards.

Often paired with auto-scaling.

# Security and privacy

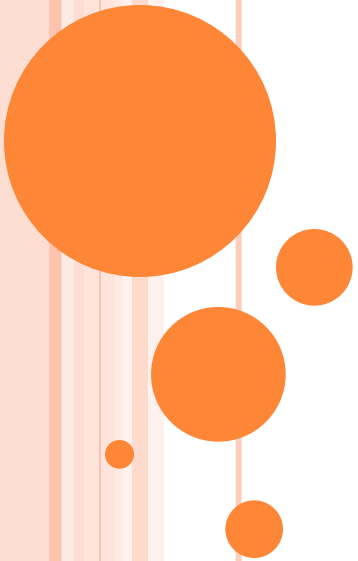
- Data protection
- Physical Control
- Identity management
- Physical and personnel security
- Availability
- Application security
- Privacy
- Legal issues

# Advantages

- Cloud  everyday use
- To save huge amounts of data
- Easier to maintain information
- Makes security easy
- Maintainability and sustainability are better

UNIT II

# CLOUD ENABLING TECHNOLOGIES



# Service-Oriented Architecture (SOA)

**What is SOA?**

- Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications.

- Each service provides a business capability, and services can also communicate with each other across platforms and languages.

Major roles within SOA:

- SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces.
- SOA enables the development of applications that are easier to handle and more secure.
- It provides a common infrastructure and documentation to develop services, with the opportunity to add new features.

**Key Features of SOA:**

•**Loosely Coupled Services:** Promote flexibility and reusability.

•**Standard Communication:** Services communicate using XML-based messages.

•**Platform Neutrality:** Services work across different systems and platforms.

•**Distributed Applications:** Supports distributed computing for scalability.

## SOA in Practice:

- Used in building **Grid and Cloud applications**.
- Applications are **interoperable, extensible**, and **efficient**.
- Inspired by **REST (Representational State Transfer)** and early distributed system models.

## (SOA) Summary

- SOA is a software design method.
- It uses services to build applications.
- Each service performs a specific task.
- Services can communicate with each other.
- SOA makes applications flexible and reusable.
- It supports different platforms and programming languages.

# SOA Architecture

- **SOA (Service-Oriented Architecture)** provides methods for:
  - Designing
  - Deploying
  - Managing **network-accessible services**
- A **service** in SOA performs a specific business function and ensures:
  - Consistency
  - Predictable results
  - Quality of Service (QoS)

## Components of SOA

SOA architecture consists of **three main components**:

**1.Service Provider**
**2.Service Consumer**
**3.Service Registry & Middleware**

These work together to enable service discovery, access, and management.

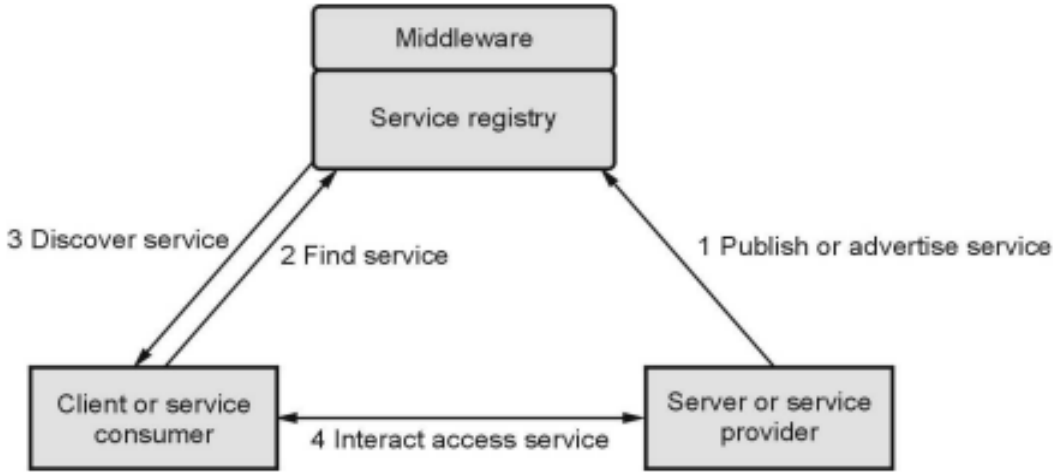



**Fig. 2.1.1 SOA architecture**

# 1.Service Provider

- Publishes services into the **Service Registry**
- Makes services accessible using **APIs and interfaces**
- Defines **QoS and security** through **Service Level Agreements (SLAs)**
- Controls who can access the service and under what conditions

# 2.Service Consumer

- Finds and **invokes services** from the registry
- Uses **standard interfaces and APIs** to interact with services
- Follows this process:
  - Searches for a service in the registry
  - Retrieves access details
  - Connects with the **Service Provider** to use the service

# 3.Service Registry & Middleware

- Stores metadata and **references of all published services**
- Allows **service discovery** by consumers
- Acts like a directory or search engine for services

# Middleware - Enterprise Service Bus (ESB)

- Acts as the **backbone** of SOA
- Integrates legacy systems and enables service interaction
- Supports:
  - **Message translation & transformation**
  - **Protocol conversion**
  - **Routing**
  - **Security & QoS management**

# Characteristics of SOA

The different characteristics of SOA are as follows :

- Provides interoperability between the services.
- Provides methods for service encapsulation, service discovery, service composition, service reusability and service integration.
- Facilitates QoS (Quality of Services) through service contract based on Service Level Agreement (SLA).
- Provides loosely couples services.
- Provides location transparency with better scalability and availability.
- Ease of maintenance with reduced cost of application development and deployment.

# Components of SOA Summary

**Service Provider**

- Creates and publishes services.
- Makes services available to users.

**Service Consumer**

- Finds and uses services.
- Requests services from providers.

**Service Registry**

- Stores information about services.
- Helps consumers find services.

**Benefits of SOA**

- Easy to manage.
- Easy to expand.
- Improves software reuse.
- Supports cloud applications.

# RESTful Systems

Representational State Transfer (REST)

- REST stands for Representational State Transfer.
- It is a style for developing web services.
- REST uses HTTP protocols. REST services are simple and scalable.

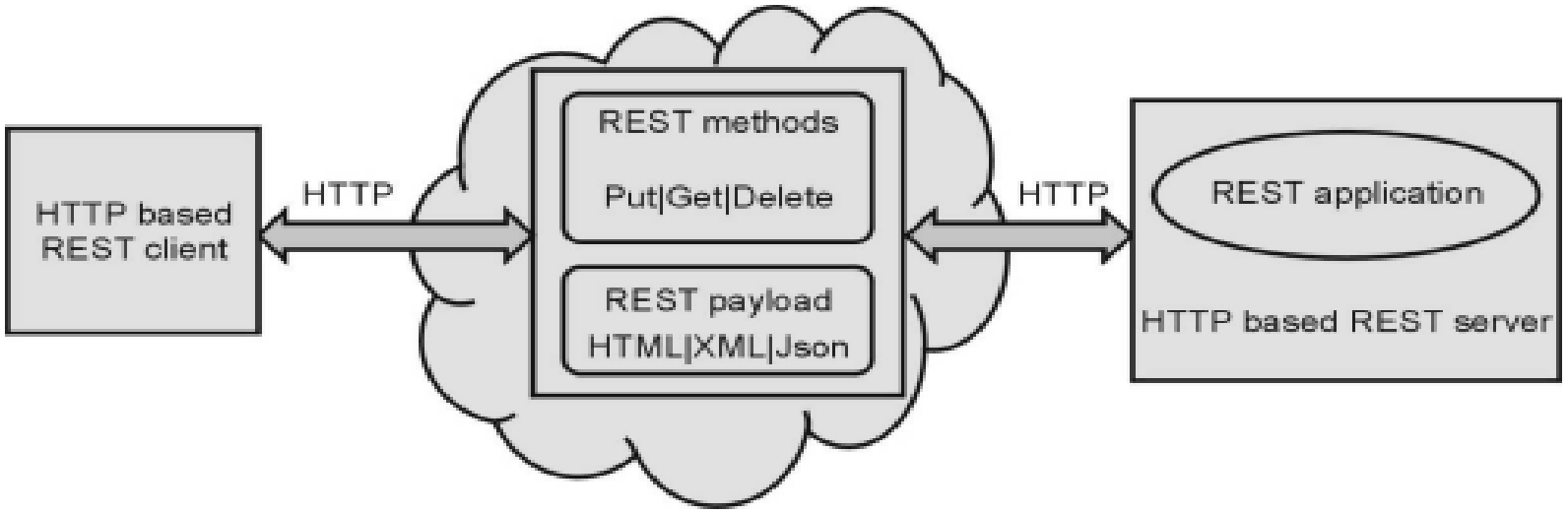



**Fig. 2.2.1 Interaction in REST with HTTP specification**

## What are RESTful Web Services?

- Web services that follow REST principles are called **RESTful Web Services**.
- These services allow clients to **access and manipulate web resources** using standard operations.
- They are designed to be **simple, scalable, and stateless**.

# REST Principles

- **Resource Identification**
- **Controlled Interfaces**
- **Self-Descriptive Messages**
- **Stateless Communication**

# A) Resource Identification

- Every resource has a unique URI.
- URI helps locate resources on the internet.
- In REST, everything is treated as a **resource** (e.g., document, image, or data).
- Each resource is uniquely identified using a **Uniform Resource Identifier (URI)**.
- URIs:
  - Provide a **global address** to resources.
  - Can be **bookmarked or shared** via hyperlinks.
  - Enable easy **service discovery** and interaction.

# B) Controlled Interfaces

- RESTful web services use a **standard interface** for communication:
  - Based on **HTTP protocols**.
  - Uses **CRUD operations**:
    - **Create** – PUT
    - **Read** – GET
    - **Update** – POST
    - **Delete** – DELETE
- These methods make it easy to perform actions on resources.

| Method | Operation |
|---|---|
| PUT | Create a new resource |
| GET | Retrieve the current state of resource |
| POST | Update or transfers a new state to a resource |
| DELETE | Delete or destroy a resource |

Table 2.2.1 REST Methods

# C) Self-Descriptive Messages

- REST messages contain **all required information** for processing.
- No need for clients or intermediaries to understand internal formats.
- Messages contain all necessary information.
- Clients can understand requests easily.
- Resources are separated from their representation (e.g., HTML, XML, JSON).
- Messages can include **metadata** for:
  - Caching
  - Authentication
  - Error detection
  - Access control

# D) Stateless Communications

- REST uses **stateless communication**:
  - Each request is **independent** of previous ones.
  - No session data stored on the server.
- Every request is independent.
- Server does not store session information.
- Improves scalability.

- Benefits:
  - **Scalability**
  - **Failure recovery**
  - **Improved visibility**
- Limitation:
  - Can lead to **repeated data transmission**, affecting performance.
- REST can still handle **stateful interactions** using:
  - **URI rewriting**
  - **Cookies**
  - **Hidden form fields**
- Future communication states can be **embedded in the current response**.

# Summary

- REST is a powerful, **lightweight architectural style** for building scalable web services.
- Key REST principles:
  - **Resource identification** using URIs
  - **Standardized operations** (CRUD)
  - **Stateless communication**
  - **Self-descriptive messaging**
- Widely adopted for **cloud services**, APIs, and **modern web applications**.

# Web Services

- Web services are **independent, modular applications** accessible over the Internet.
- They follow SOA principles: **loosely coupled, reusable, platform-independent**.
- Communicate using **XML-based messages**.
- Allow different software to **discover, bind, and interact** with services remotely.

# Components of Web Services

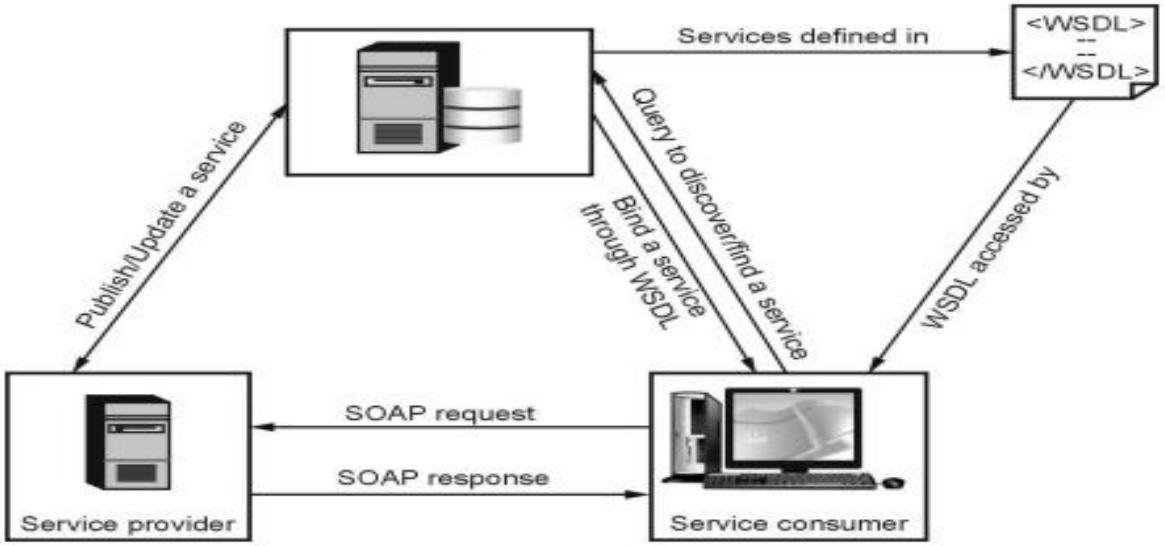



**Fig. 2.3.1 Functionality of Web services**

- **Service Provider**: Creates and publishes services to a registry.
- **Service Consumer**: Searches for and uses services from the registry.
- **Service Registry (UDDI)**: Stores references of published services for discovery.
- UDDI – Universal Description Discovery and Integration.

# SOAP - Communication Protocol

- **SOAP** (Simple Object Access Protocol)is an XML-based protocol for exchanging messages.
- Works over protocols like **HTTP, SMTP, FTP**.
- SOAP message contains:
  - **Envelope** (root element)
  - **Header** (optional metadata like authentication
  - **Body** (actual data/message)
- Enables **interoperability** between different systems.
- SOAP is an XML-based communication protocol
- It transfers messages between applications.

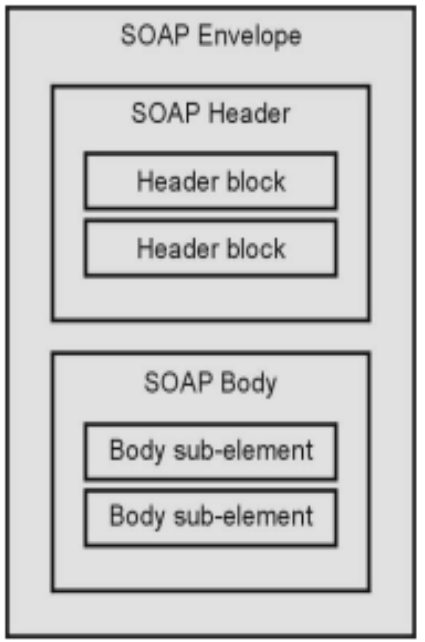



Fig. 2.3.2 Structure of SOAP message

# WSDL - Service Description

- **WSDL**(Web Service Description Language) is an XML document describing:
  - What operations the service offers.
  - Input/output message formats.
  - Communication protocols used.
- Defines how clients interact with the web service.
- Key parts:
  - **Types** (data types)
  - **Message** (data structure)
  - **PortType** (set of operations)
  - **Binding** (protocol details)
  - **Service** (endpoints)

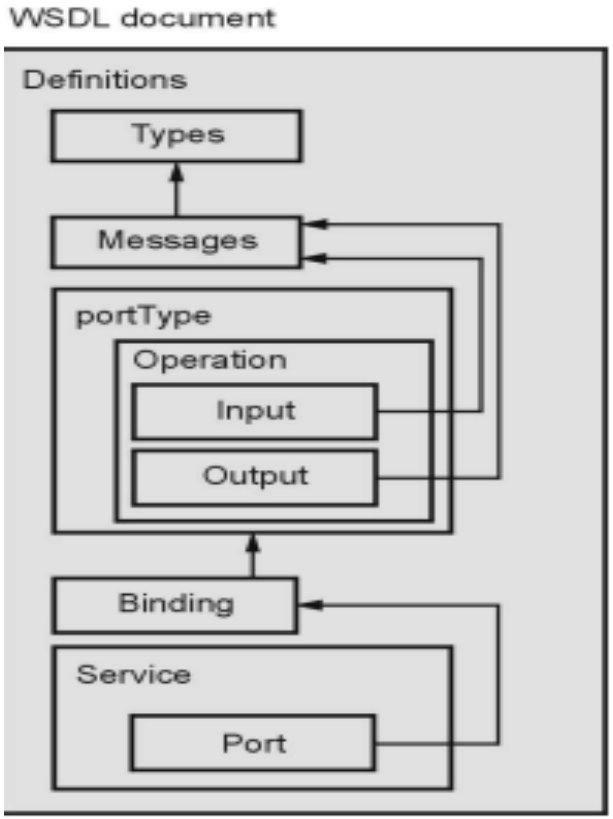



**Fig. 2.3.3 WSDL document structure**

# UDDI - Service Registry

- UDDI is a service directory.
- It helps discover web services.
- **UDDI** is a directory for publishing and discovering web services.
- Consumers search UDDI by service name, category, or identifier.
- Three main functions:
  - **Publish** services
  - **Find** services
  - **Bind** to services

# Web Services Protocol Stack

- Web services follow a protocol **stack of 6 layers**:
- **Transport Layer**: Moves messages (HTTP, SMTP, JMS).
- **Messaging Layer**: Encodes messages (SOAP, WS-Coordination).
- **Service Description Layer**: Describes service interfaces (WSDL).
- **Service Discovery Layer**: Registers and locates services (UDDI).
- **Quality of Service Layer**: Ensures reliable messaging, security (WS-Security).
- **Composition Layer**: Combines services into workflows (BPEL).

# Web Services Protocol Stack

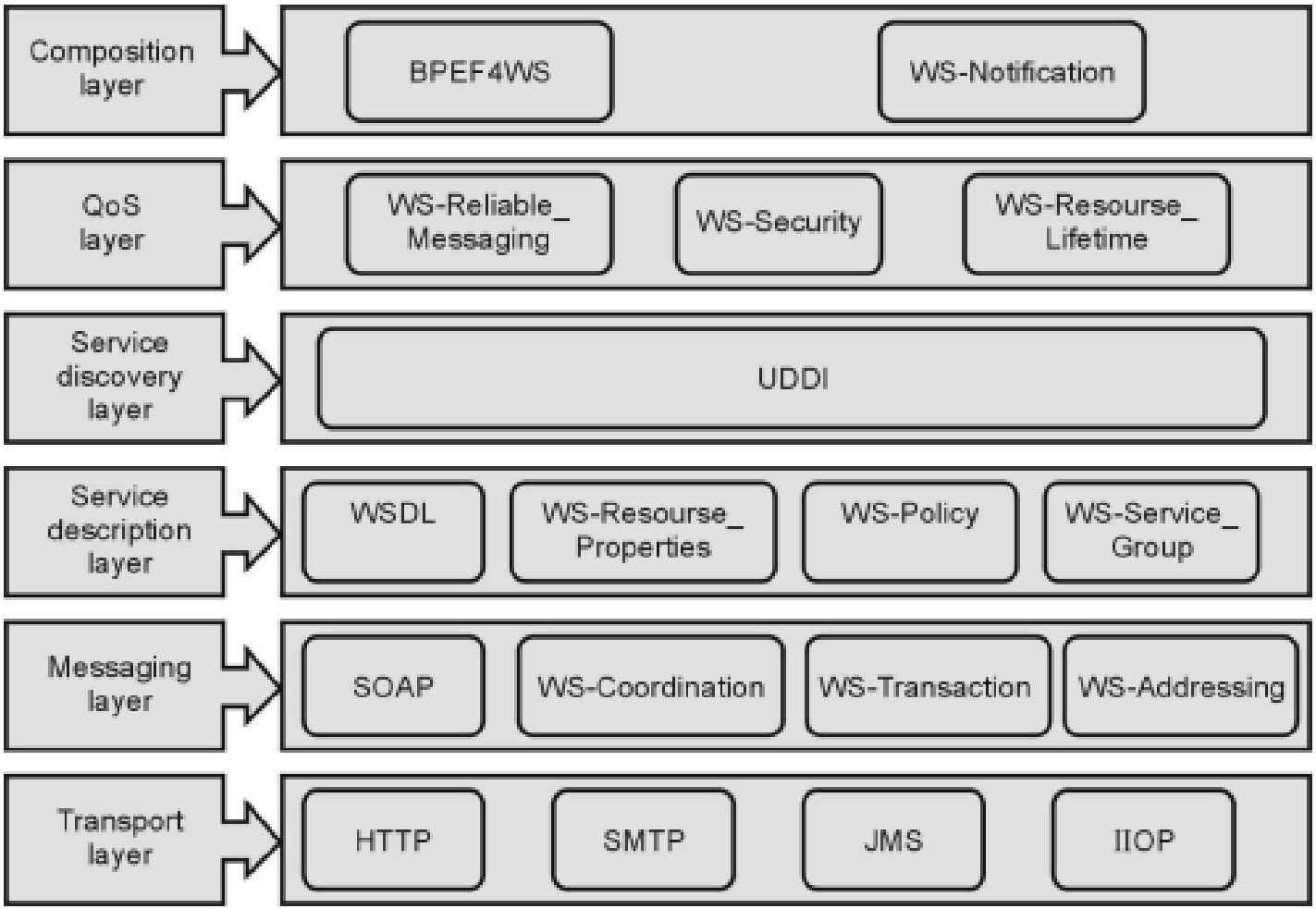



**Fig. 2.3.4 Web services Protocol Stack**

# Publish Subscribe Model

- It is a communication model.
- Publishers send messages. Subscribers receive messages.
- Publishers and subscribers do not know each other.
- A design pattern enabling **asynchronous communication** between distributed applications.
- **Publishers** send messages labeled with **topics** or filtered by **content**.
- **Subscribers** express interest in topics or content and receive matching messages.
- Supports a **many-to-many** relationship between publishers and subscribers.

# Centralized vs Distributed Systems

- **Centralized Publish-Subscribe**: One central server mediates message delivery.
  - Risk: **Single point of failure**.
- **Distributed Publish-Subscribe**: Publishers and subscribers are **decoupled**.
  - No knowledge of each other's location or data.
  - More **scalable** and **reliable**.

# Types of Publish-Subscribe Systems

**Topic-Based**:

- Messages are grouped by topics.
- Subscribers receive messages from selected topics.
  - Publishers send events under specific **topics**.
  - Subscribers subscribe to topics and receive all related messages.
  - Limitation: Topics can be too broad or inflexible.

**Content-Based**:

- Messages are filtered by content.
- Subscribers receive only relevant messages.
  - Subscribers specify **detailed filters** or conditions based on message content.
  - More **flexible** and precise filtering.
  - Matches messages to subscriptions by content attributes.

# Oracle's Publish-Subscribe Solution

- Oracle uses **Advanced Queuing** integrated into its database.
- Manages messaging between applications via queues.
- Features include:
  - **Rule-based subscription**
  - **Message broadcasting and notification**
  - **High availability, scalability, and reliability**
- Messages persist, propagate across databases, and transmit via Oracle Net Services.

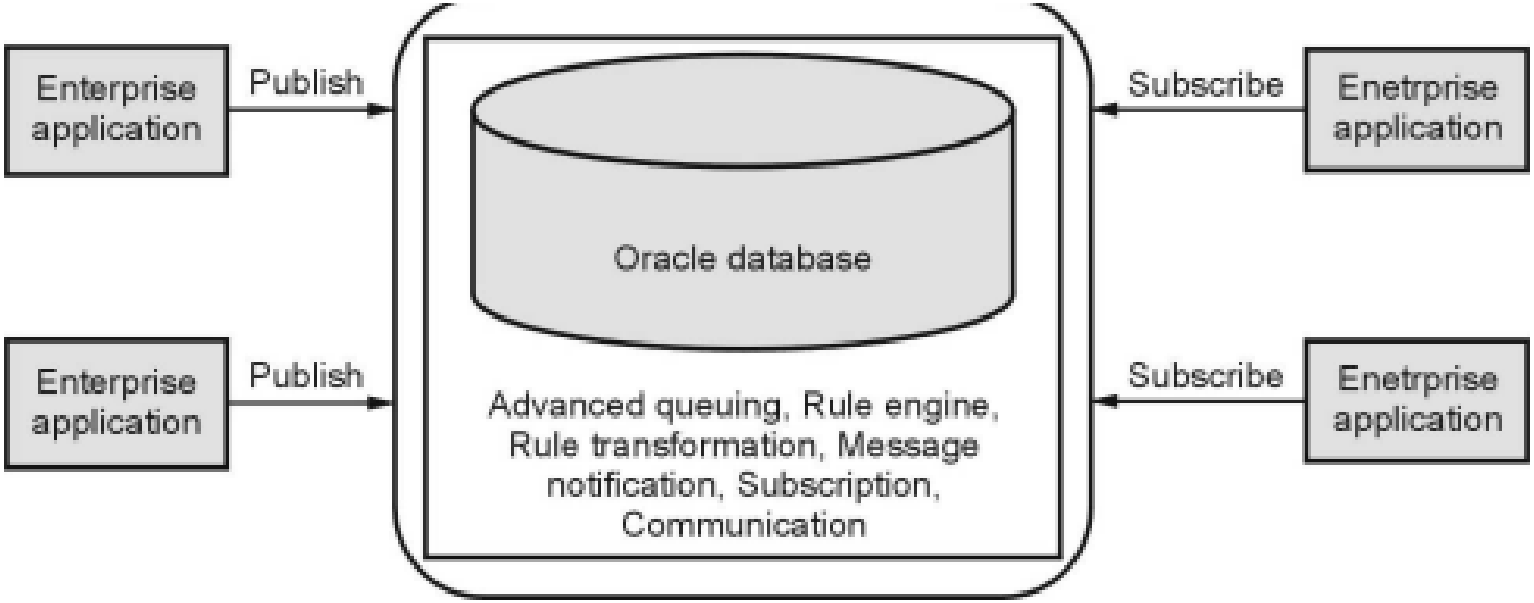



**Fig. 2.4.1 Publish subscribe model for oracle database**

# Advantages

- Supports many users.
- Improves scalability.
- Reduces communication overhead.

# Basics of Virtualization

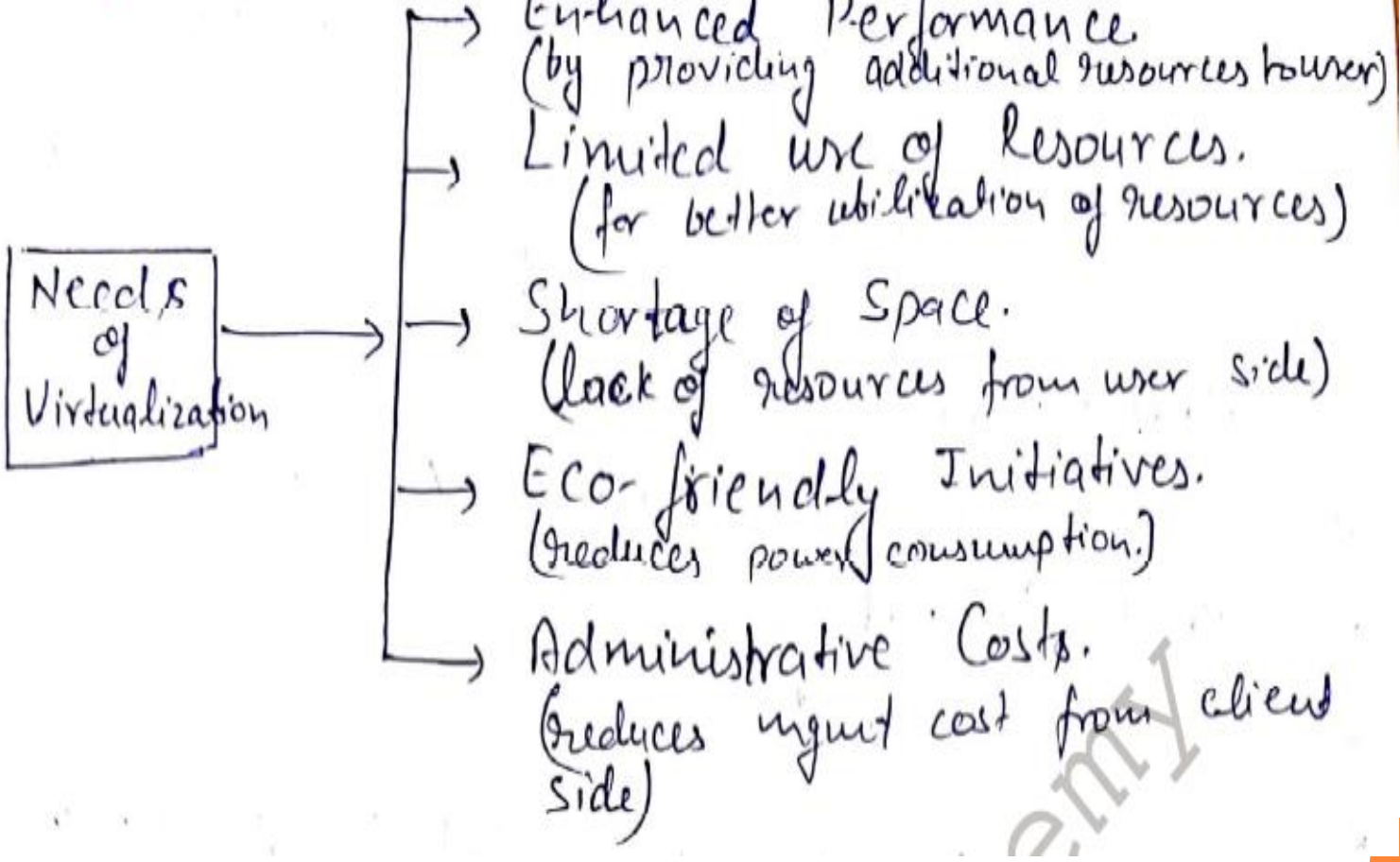
- Virtualization is a key technology used in building clusters, grids, and cloud computing systems.
- These systems need large amounts of computing, storage, and networking resources.
- Virtualization helps by combining these resources and presenting them as a single system image to users, which makes managing and using them easier and faster.
- It allows one physical machine (called a host) to run multiple virtual machines (VMs).
- Each VM can run its own operating system, known as the guest operating system.
- This makes it possible to use a single server for many tasks, improving resource usage and reducing hardware costs.

* It is the creation of virtual & version of something, such as server, a desktop, a storage device, an operating system or network resources. i.e resources are made available to the users but virtually not physically.

* It is a technique which allows sharing a single physical instance of resources to multiple users.

# Need of Virtualization :

(2.2)

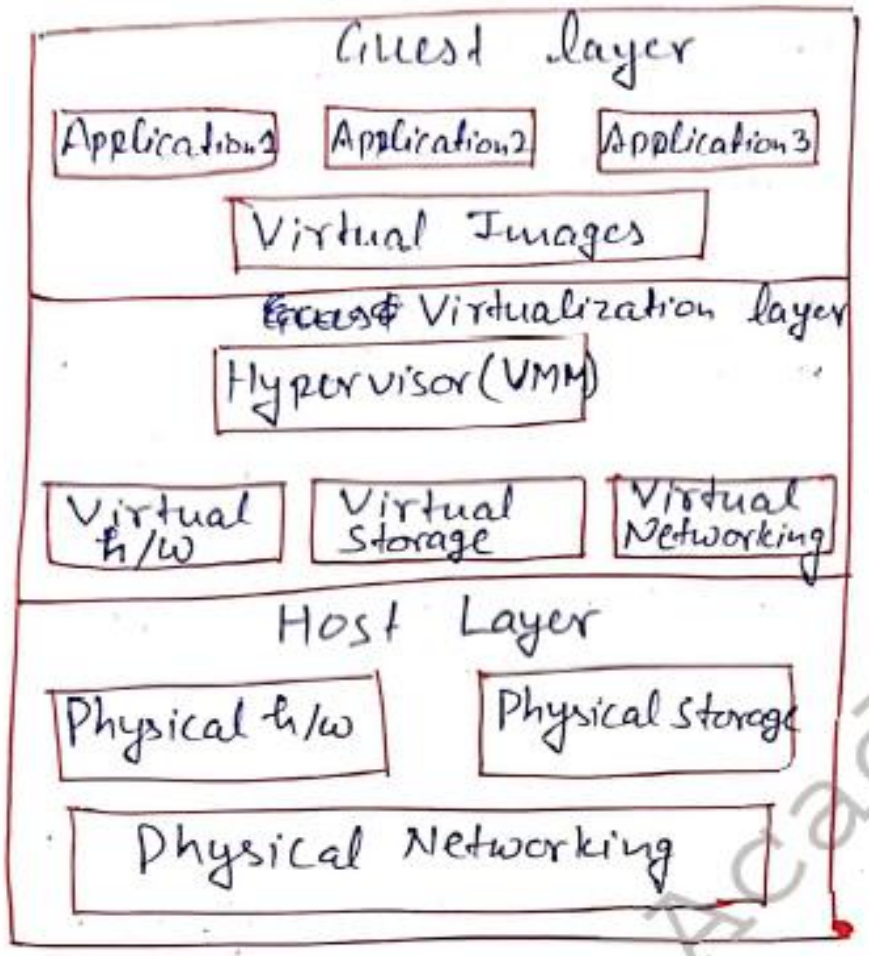


# Advantages of Virtualization:-

* Economical:- (Don't need to buy high capacity hardware and other resources.).
* Flexible Operations:- (You can access it from anywhere and customization can be done)
* Security:- Provided by cloud-service Provider.
* Eliminates the risk of system failure.
* Flexible transfer of data
* Faster provisioning of applications and resources.
* Greater business continuity and disaster recovery.
* Simplified data centre management.

# Virtualization Reference Model:-






Four major components of Virtualization reference Model is available there.

i) Host Layer:

⇒ The host layer represents the original environment where the guest is supposed to be managed.

⇒ All the physical resources and physical networking is present in this layer.

⇒ Each guest runs on the host machine using shared resources donated to it by the host.

⇒ The OS works as the host and manages the physical resources management and the device support.

## Guest Layer:

⇒ The guest represents the system components that interacts with the virtualization layer rather than with the host.

⇒ Guest usually consists of one or more virtual disk files, and a VM definition file.

⇒ Virtually machines are centrally managed by a host application that sees and manages each virtual machine as a different application.

## Virtualization Layer:

⇒ The virtualization layer is responsible for recreating the same or a different environment where the guest will operate.

⇒ It is an additional abstraction ~~layer~~ layer between a network and a storage hardware, computing, and the application running on it.

## Hypervisor:

⇒ A hypervisor, also known as Virtual Machine Monitor (VMM), is a software that creates and runs virtual machines (VMs)

⇒ A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.

# Capability of Server with and without Virtualization

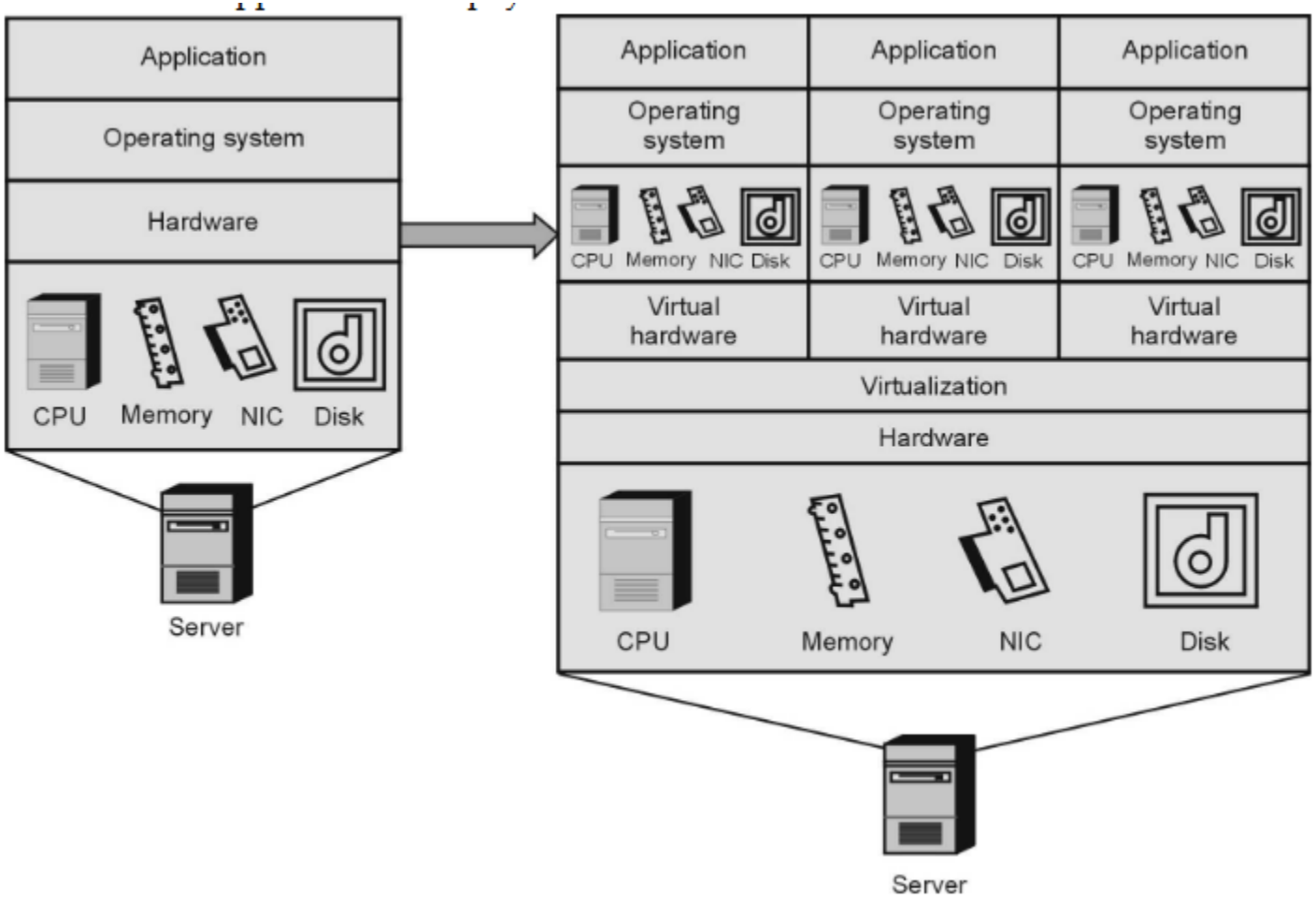



**Fig. 2.5.1 Capability of Server with and without Virtualization**

# Virtualization

**Virtualization in Cloud Computing**

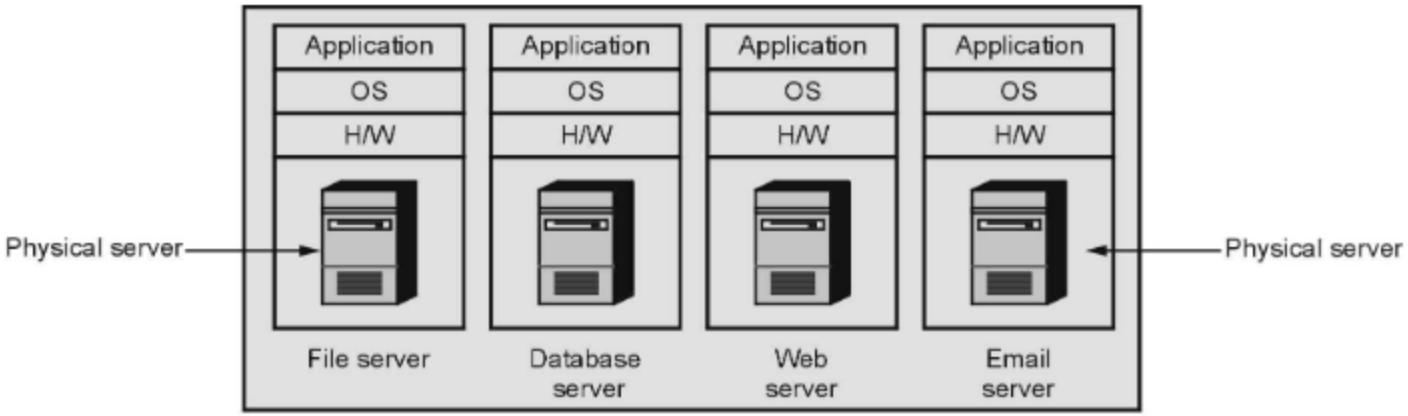
- Every cloud system uses virtualization.
- It allows resources to be given to users as needed (on demand).
- Makes cloud services scalable and secure.

**Benefits of Virtualization**

- Better use of physical hardware.
- Easy software management.
- More flexible applications.
- Improved security and isolation.
- computing services.

# Traditional Server Setup

- Earlier, industries used separate physical servers for:
  - File storage
  - Database
  - Web hosting
  - Email
- Each server needed its own hardware, OS, software, and admin.



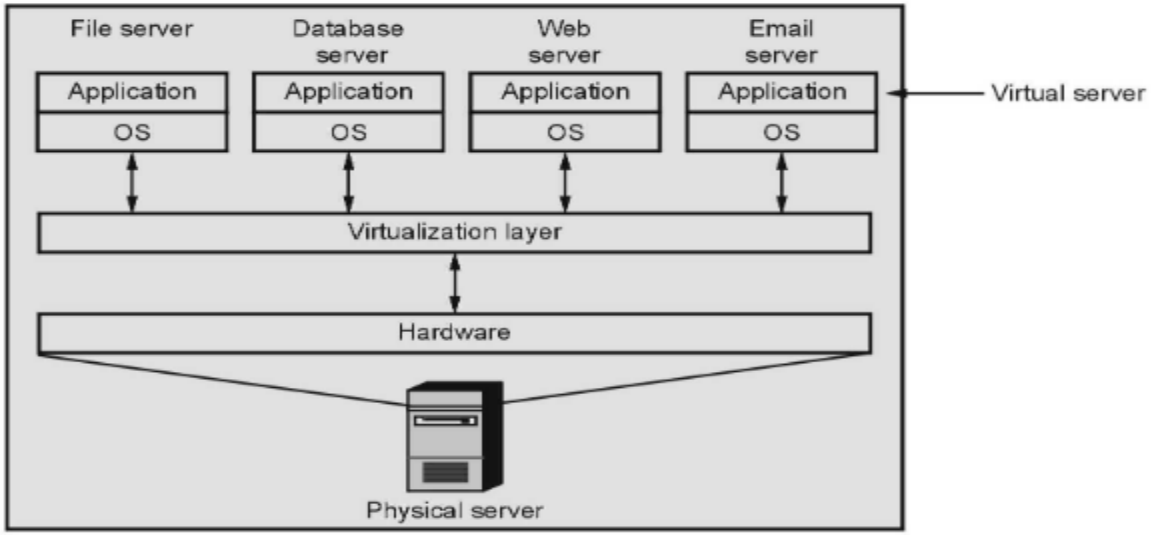
Traditional server concept

## Problems with Old Setup

- High cost for hardware and software.
- More power usage and staff needed.
- If a server failed, services stopped until fixed.
- Risk of total system failure.

## Virtualization as a Solution

- Virtualization allows multiple server OS to run on one physical machine.
- Fewer physical servers are needed for multiple tasks.




Virtual server concept

## Operations in a Virtualized Environment

- Users can:
  - Create, delete, copy, and move virtual machines (VMs)
  - Take snapshots and roll back VMs
  - Use templates and save VM states

## Purpose of Virtualization

- Share resources among multiple users
- Improve computing performance
- Maximize resource use
- Offer flexible application environments

## Role of Virtual Machine Manager (VMM)

- Also called a **Hypervisor**
- Special software that:
  - Creates and manages VMs
  - Runs multiple VMs on one physical machine (host)

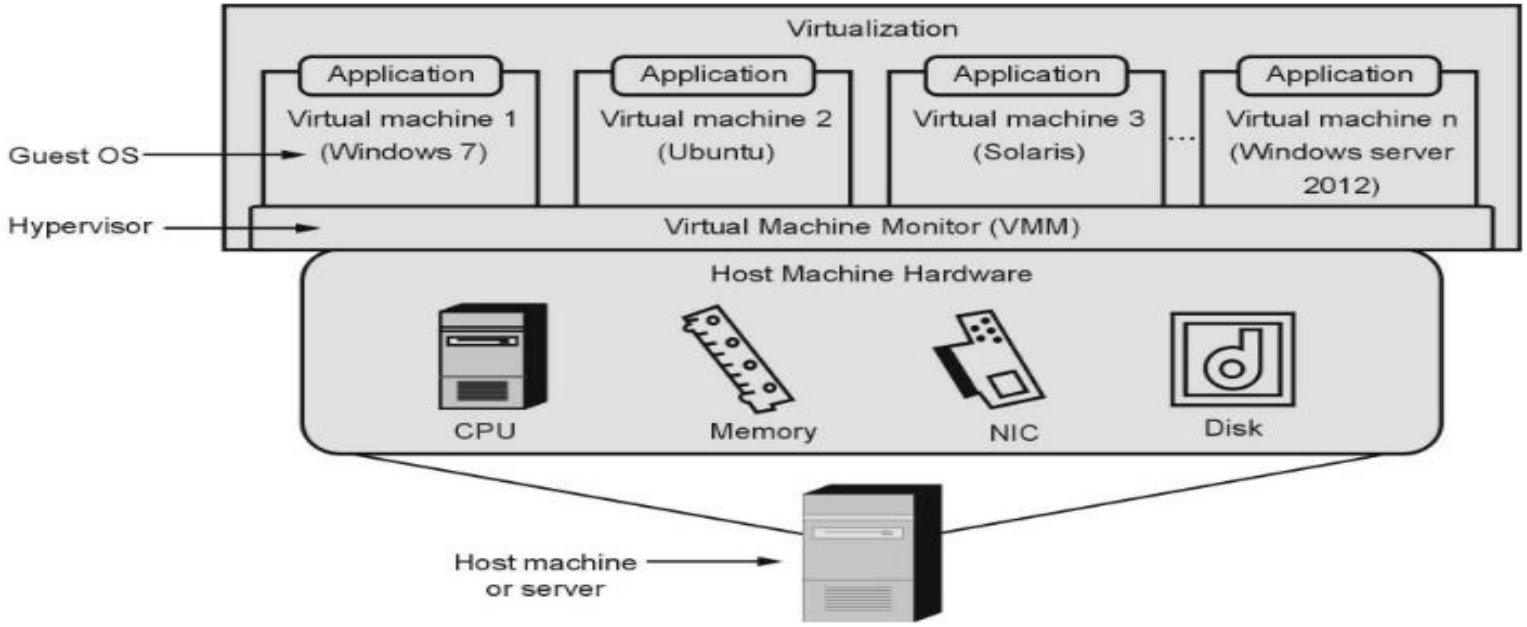



**Fig. 2.5.3 Single server running Virtualization**

**Host and Guest OS**

**Hypervisor (VMM)**

Hypervisor manages virtual machines.

It creates and controls VMs.

**Host and Guest OS**

Host OS runs on physical hardware.

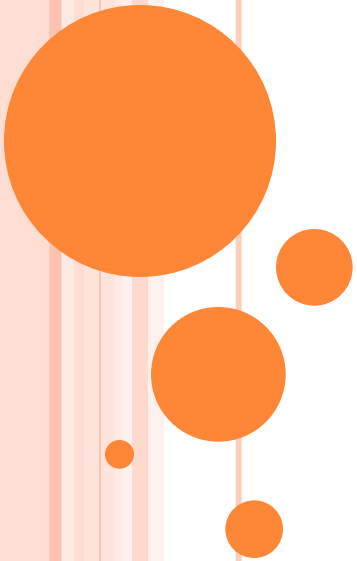
Guest OS runs inside a virtual machine.

Each VM shares:

CPU, memory, storage, network, and I/O of the host

## Summary

- Virtualization lets one server do the job of many
- Saves cost, power, and space
- Increases flexibility and control



# Characteristics of Virtualization

1) Maximum Resource Utilization

- Virtualization lets many guest OS run on one machine, using CPU and memory fully by sharing them among virtual machines.

2) Reduced Hardware & Maintenance Cost

By combining multiple servers into one, it saves money. Fewer physical machines mean less cost for setup, power, and maintenance.

3) Disaster Recovery

If a system fails due to disasters, VMs can quickly move to another machine. This avoids big losses and downtime.

# 4) Server Consolidation

- Many old and new applications can run on one server. It reduces the need for separate machines by combining servers into one.

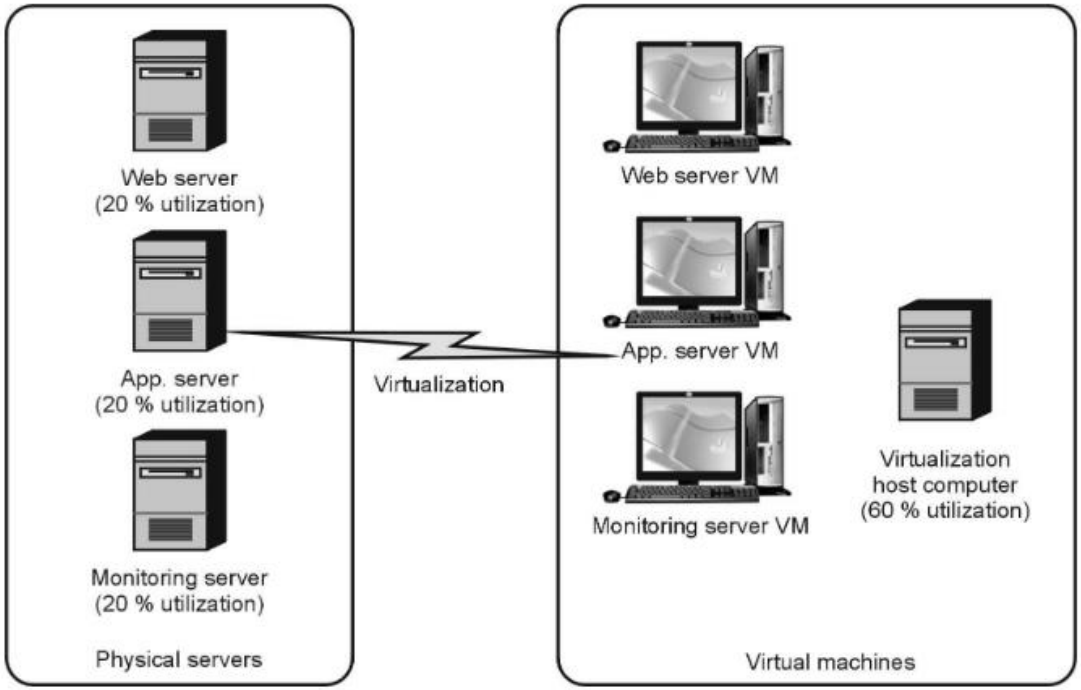



**Fig. 2.5.5 Server Consolidation**

5) Dynamic Load Balancing

Work is shared across machines, preventing overload. Virtualization shifts tasks from busy systems to less busy ones automatically.

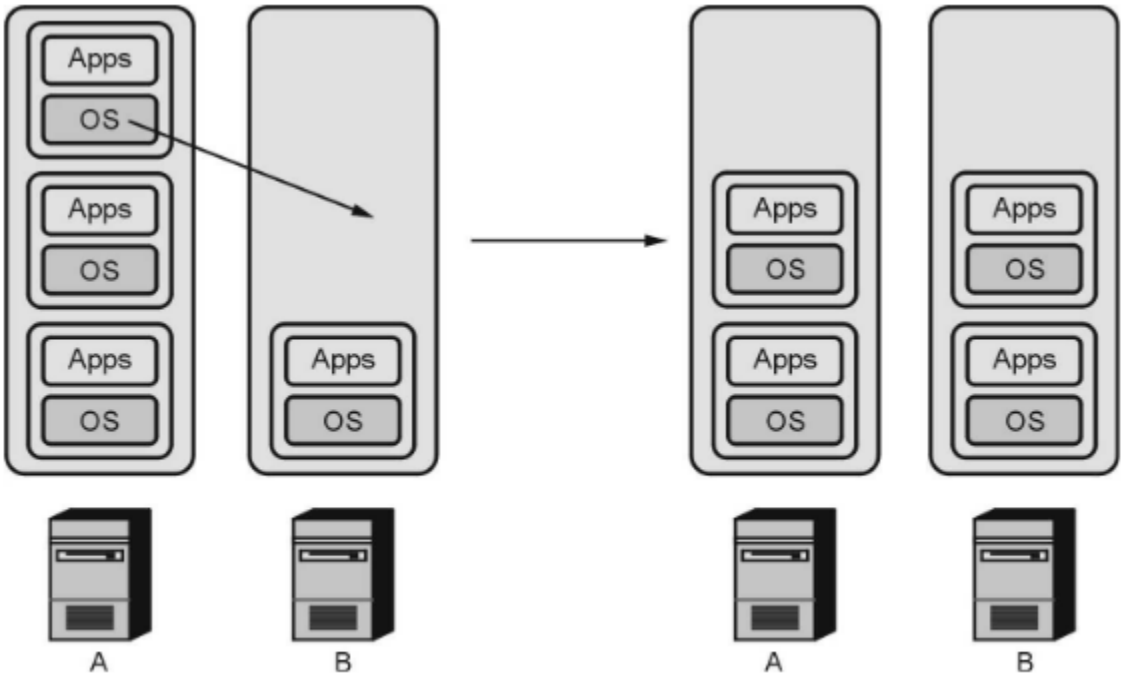



**Fig. 2.5.4 Dynamic Load balancing**

6) Easy VM Management

- VMs are easy to move, copy, or delete. They can be backed up or shifted during maintenance without downtime.

7) Legacy App Support & Beta Testing

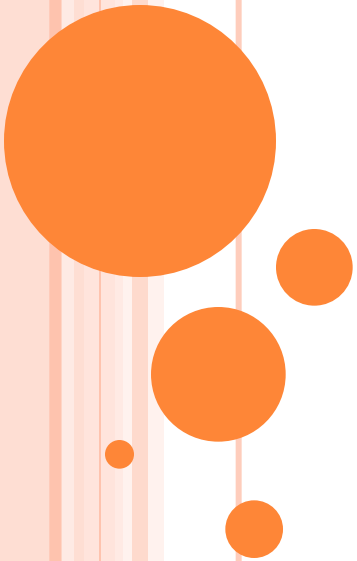
Old apps can run on older OS within VMs. New software can be tested safely without needing extra machines.

8) Sandboxing

VMs create safe, isolated spaces to run untrusted software. This increases security.

9) Virtual Hardware

Virtualization gives virtual devices like storage, Ethernet, and switches, even if real hardware is not there.



# Pros and Cons of Virtualization

# Pros of Virtualization

- 1. Cost Saving: One physical system can run many OS and apps, so fewer servers are needed.
- 2.Better Resource Use: Resources like CPU, storage, and memory are used more efficiently.
- 3. Higher ROI: Fewer machines used means less cost, more savings, and higher profits.
- 4. Easier Budgeting: IT tasks like setup and maintenance cost less.
- 5. More Flexibility: You can run many apps and OS types on one system at the same time.

# Cons of Virtualization

- 1. High Initial Cost: Setting up virtualization needs investment in tools and systems.
- 2. Performance Issues: Sometimes, virtual machines are slower than real physical systems.
- 3. Licensing Problems: Some software might not work well or need special licenses for VMs.
- 4. Troubleshooting is Harder: Virtual layers make it harder to find the root of technical problems.

# Types of Virtualization

1. **Server Virtualization**

- Divides one physical server into many virtual servers.
- Each virtual server works like a real, independent server.
- Used to run multiple applications on one machine.
- Helps reduce hardware costs and energy use.

**2. Storage Virtualization**

- Combines many physical storage devices into one virtual storage system.
- Makes it easier to manage and access data.
- Used in cloud storage services like Google Drive and Dropbox.

### 3. Network Virtualization

- Creates a virtual version of physical network resources.
- Combines bandwidth and switches to create a virtual network.
- Improves performance, security, and network management.

### 4. Desktop Virtualization

- Allows users to access their desktop from anywhere using the internet.
- The desktop runs on a remote server, not on the user's local computer.
- Useful for companies with remote workers or BYOD (Bring Your Own Device) policies.

## 5. Application Virtualization

- Runs applications on a central server but makes them available on user devices.
- The app is not installed directly on the user's computer.
- Helps manage and update apps easily.

## 6. Operating System (OS) Virtualization

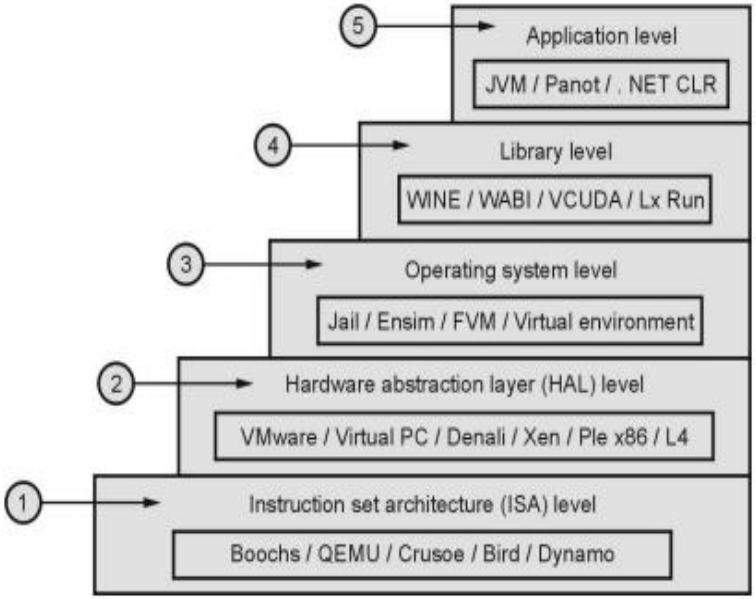
- Allows one machine to run different operating systems at the same time.
- For example, running Linux and Windows on the same system.
- Mostly used in testing, development, and training environments.

# Implementation Levels of Virtualization

**Introduction**

- Virtualization is done by adding a software layer between the host OS and guest OS.
- This software layer helps convert physical hardware into virtual resources.

- There are **5 main levels** of virtualization:
  - 1. ISA Level
  - 2. Hardware Level
  - 3. OS Level
  - 4. Library Level
  - 5. Application Level



## 1. ISA Level Virtualization

- Virtualization happens by **emulating** the entire instruction set of one system on another.
- Every instruction from the guest system is translated to match the host system.
- This allows different platforms (like x86, Sparc, Alpha) to run on other machines.
- **Downside**: It's slower because all instructions are interpreted.

**Tools**:

- Bochs
- QEMU
- Crusoe
- BIRD

## 2. Hardware Level Virtualization

- This level works directly with physical hardware using a **hypervisor**.
- It creates virtual versions of CPU, memory, and I/O devices.
- **Issue**: Some hardware like x86 isn't fully virtualizable.

**Tools**:

- VMware (uses dynamic translation to solve issues)
- Virtual PC (supports undo disk feature)
- Denali (supports lightweight VMs for scalability)

## 3. Operating System Level

- This level creates separate environments or containers for each OS.
- Allows running multiple OSes without rebooting.
- Each container is isolated and easy to manage.

**Tools**:

- Jail (FreeBSD-based, limits access to resources)
- Ensim (used to create Virtual Private Servers)

## 4. Library Level Virtualization

- Uses programming libraries to simulate another OS's environment.
- Hides OS details from the application using APIs and ABIs.
- Helps run apps made for one OS on another.

- **Example**:
  - WINE (lets Windows apps run on Linux by mimicking Windows APIs)

## 5. Application Level Virtualization

- Runs apps as if they are part of the system without installing them.
- Virtual machine runs on top of OS and executes app-specific code.
- Offers strong security and isolation.

**Example**:

- Java Virtual Machine (JVM) runs Java bytecode independently from OS.

# The comparison between different levels of virtualization

The comparison between different levels of virtualization is shown in Table 2.7.1.

| Implementation Level | Performance | Application Flexibility | Implementation Complexity | Application Isolation |
|---|---|---|---|---|
| Instruction Set Architecture Level (ISA) | Very Poor | Very Good | Medium | Medium |
| Hardware Abstraction Level (HAL) | Very Good | Medium | Very Good | Good |
| Operating System Level | Very Good | Poor | Medium | Poor |
| Library Level | Medium | Poor | Poor | Poor |
| Application Level | Poor | Poor | Very Good | Very Good |

**Table 2.7.1 Comparison between different implementation levels of virtualization**

## Conclusion

- Each virtualization level has unique use cases and tools.
- Together, they help improve resource usage, compatibility, and flexibility.
- Virtualization is a key part of modern cloud computing systems.

# Virtualization Structures

- Virtualization means creating virtual versions of hardware, OS, storage, or networks.
- One physical machine (host) can run many virtual machines (VMs).
- Each VM runs its own operating system, called a **guest OS**.
- The software that controls virtualization is called a **Hypervisor** or **VMM (Virtual Machine Monitor)**.

## Types of Virtualization Structures

Virtualization structures are based on the position of the hypervisor:

- **1. Hosted Structure (Type 2)**
- **2. Bare-Metal Structure (Type 1)**

## Hosted Structure (Type 2)

- Hypervisor runs on top of an existing OS (host OS).
- Guest OS runs on top of the hypervisor.
- The host OS provides hardware access and drivers.
- **Examples**: VMware Workstation, VirtualBox, QEMU, Microsoft Virtual PC

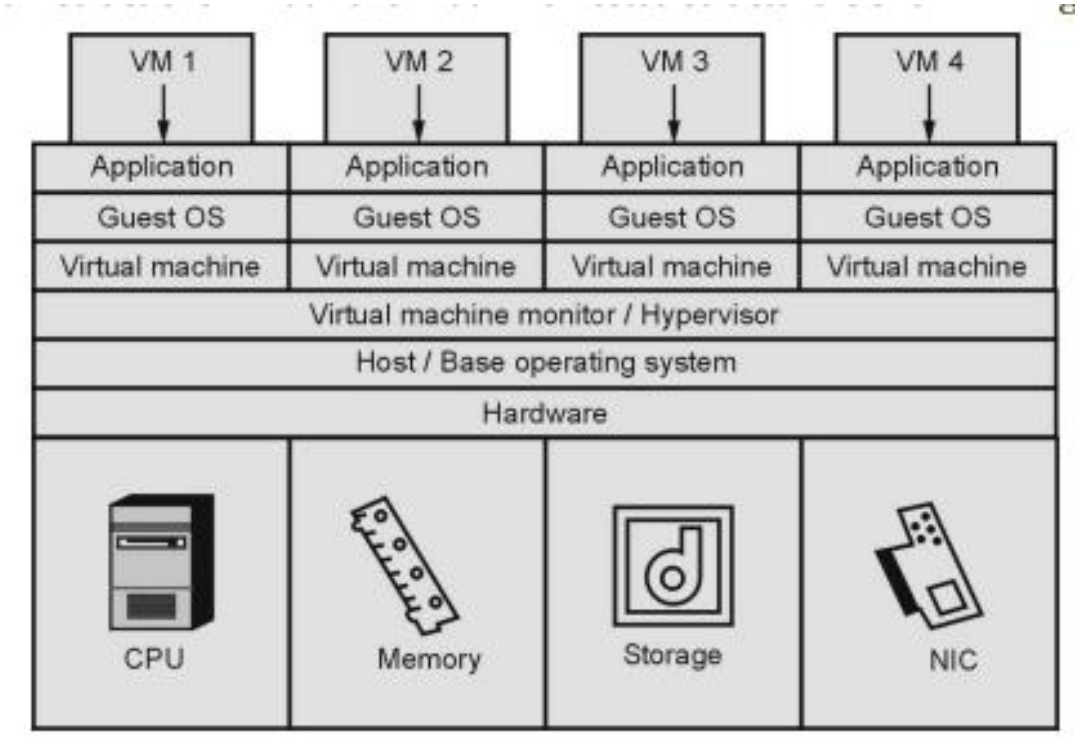



Fig. 2.8.1 Hosted Structure (Type II Hypervisor)

# Pros & Cons of Hosted Structure

## ✅ Advantages:

- Easy to install and manage
- Good hardware compatibility
- Great for testing software
- No need to install hardware drivers
- Often free to use

## ❌ Disadvantages:

- Slower performance
- Guest OS can’t directly access hardware
- Not suitable for large-scale enterprise use

## Bare-Metal Structure (Type 1)

- Hypervisor runs **directly on hardware** (no base OS).
- Offers better performance and full hardware control.
- Mostly used in data centers and enterprise servers.
- **Examples**: VMware ESXi, Microsoft Hyper-V, Citrix XenServer

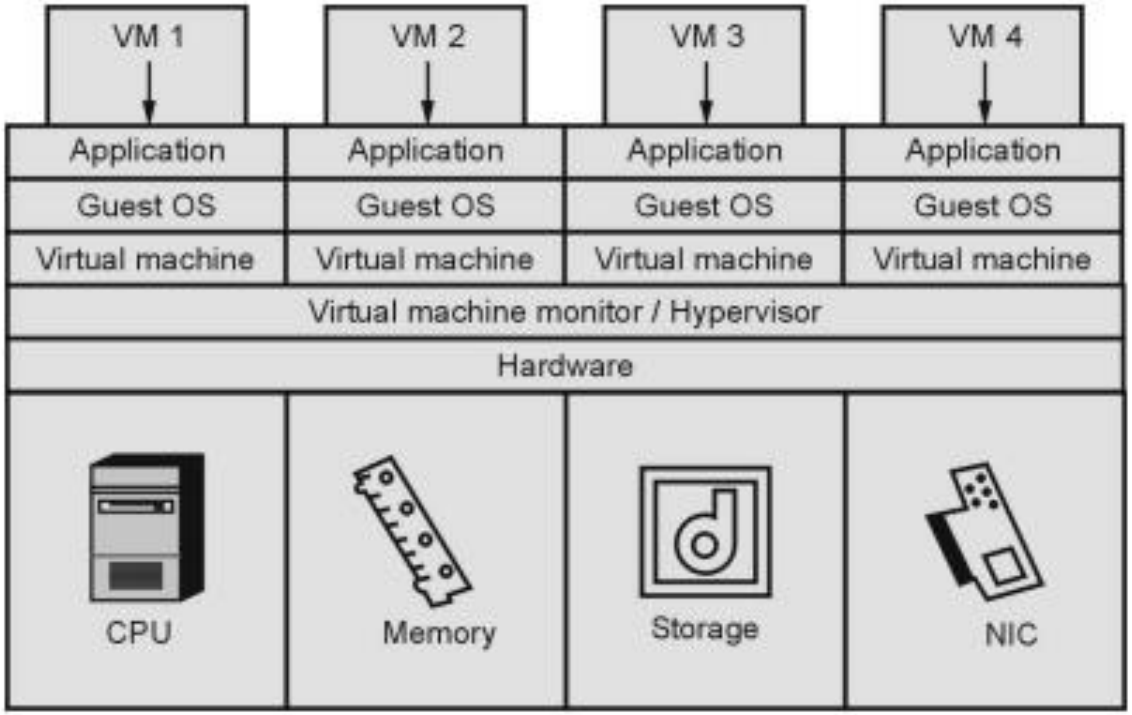



Fig. 2.8.3 Bare-Metal Structure (Type-I Hypervisor)

# Pros & Cons of Bare-Metal Structure

## ✅ Advantages:

- Faster and more efficient
- High scalability and performance
- Supports features like backup, recovery, and security
- Lower maintenance costs

## ❌ Disadvantages:

- Needs special hardware
- Higher setup and maintenance cost
- Can be complex to manage
- Limited hardware support

## Conclusion

- **Hosted Hypervisors** are simple and good for personal or testing use.
- **Bare-Metal Hypervisors** are best for businesses needing high performance.
- Choosing the right structure depends on cost, performance, and usage needs.

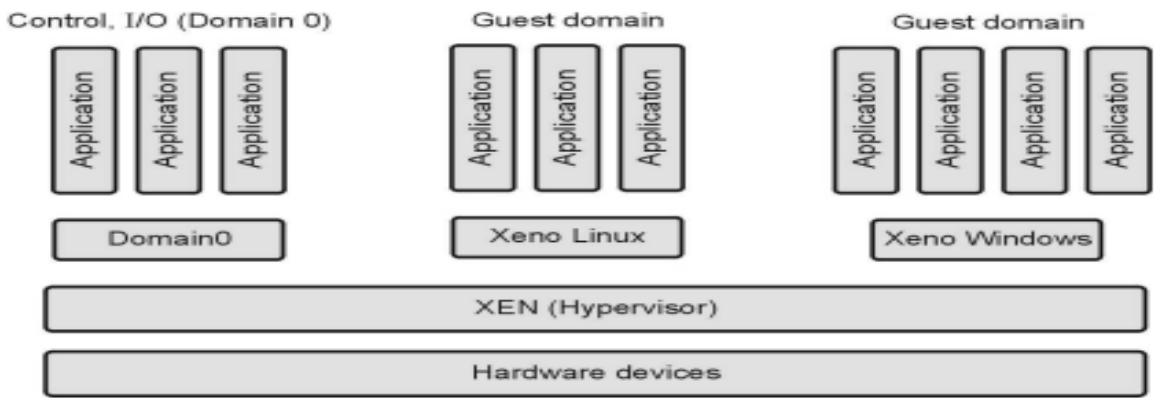
# Virtualization Tools & Mechanisms

- Hypervisors enable guest OS is to run privileged instructions via hyper calls.
- Two main hypervisor architectures:
  - **Micro-kernel** (e.g., Microsoft Hyper-V) – small, basic functions only
  - **Monolithic** (e.g., VMware ESX) – includes device drivers and full functions

# Popular Virtualization Tools

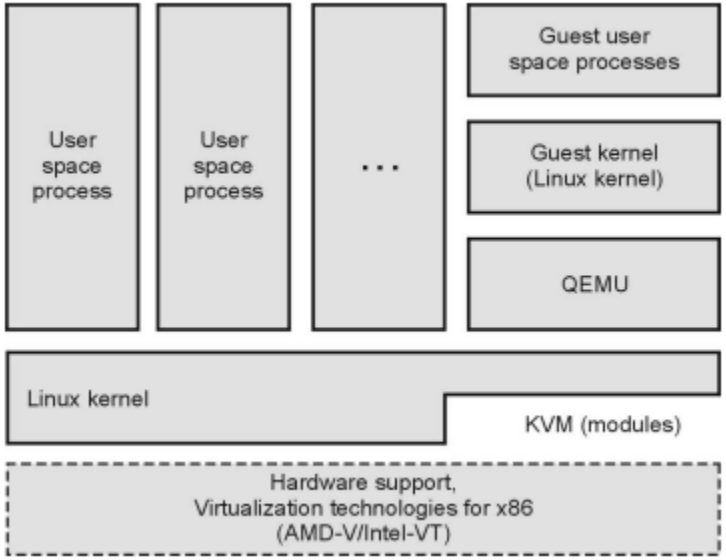
- **Xen**:
  - Open source, bare-metal (Type 1) hypervisor.
  - Runs directly on hardware with no host OS.
  - Uses micro-kernel design, small size, separates policy and mechanism.
  - Supports Windows, Linux, BSD, Solaris as guest OS.
  - Key parts: Xen hypervisor, Domain 0 (privileged guest OS), Domain U (other guests).
  - Domain 0 manages hardware access, VMs, IO, and must be secured.

The architecture of the Xen hypervisor



# KVM (Kernel-Based VM)

- Open source hosted (Type 2) hypervisor built into Linux kernel.
- Uses hardware virtualization extensions (Intel VT, AMD-V).
- Supports unmodified guest OS like Linux and Windows.
- Uses QEMU emulator for device emulation, making it fast and flexible.
- Main parts: Kernel modules (kvm.ko), user-space programs (e.g., QEMU).



**KVM Features**

- Supports 32 & 64-bit guest OS.
- Para-virtualized drivers for better performance.
- Snapshots and memory management.
- CPU and device hot plug support.
- Uses VirtIO for efficient device virtualization.

# Virtualization Mechanisms

Mechanisms control how virtualization is done:

- **1. Binary Translation**
- **2. Full Virtualization**
- **3. Host-based Virtualization**
- **4. Para-virtualization**
- **5. Hardware-assisted Virtualization**

# Binary Translation & Full Virtualization

- **1.Binary Translation**, the Virtual Machine Monitor (VMM) runs at highest privilege (Ring 0), guest OS runs at lower privilege (Ring 1).
- VMM scans guest instructions, intercepts privileged ones, and emulates them.

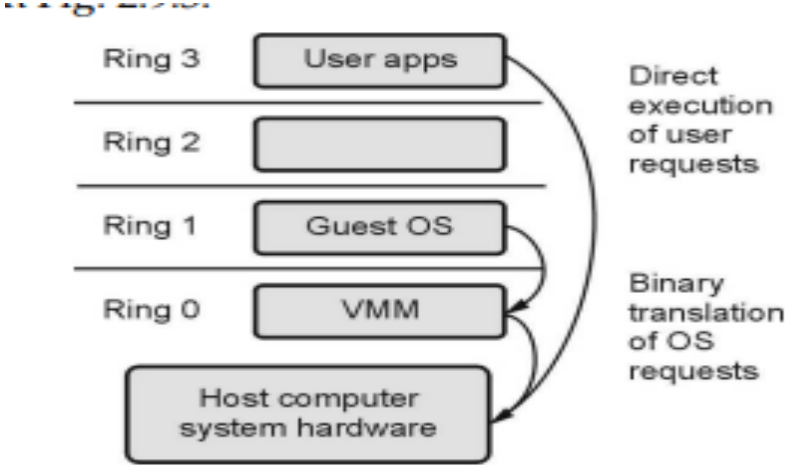



**Fig. 2.9.3 Binary Translation mechanism**

**2. Full Virtualization** uses binary translation to run guest OS without modification.

- Non-privileged instructions run directly on hardware; privileged ones trapped and emulated.
- Full virtualization allows unmodified guest OS but can reduce performance due to overhead of translation.

**3. Host-Based Virtualization**

- Virtualization layer runs on top of a host OS.
- Guest OS runs over this virtualization layer.
- Host OS manages hardware and device drivers.
- Easier to deploy but performance is lower due to overhead from multiple layers of instruction and IO handling.

## 4. Para-Virtualization

- Guest OS is modified to replace sensitive instructions with **hyper calls** to the hypervisor.
- Reduces overhead and improves performance by avoiding full emulation.
- Runs at a lower privilege level (Ring 1) to avoid conflicts.
- Examples: Xen, KVM, VMware ESXi.
- However, para-virtualization requires guest OS modifications and may cause compatibility and maintenance challenges.

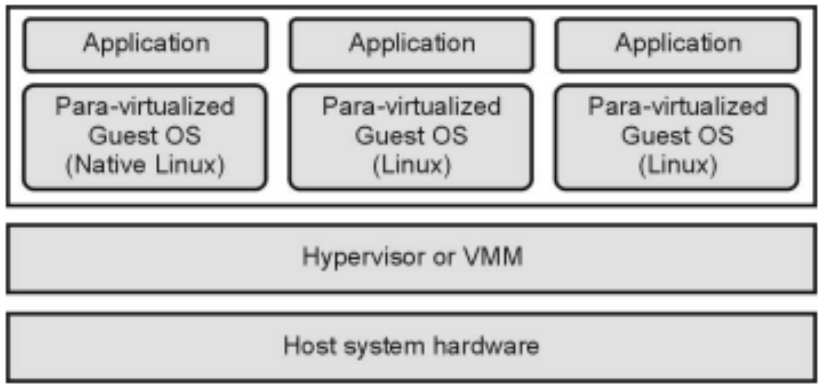



**Fig. 2.9.4 Para-virtualization architecture**

## 5. Para-Virtualization with Compiler Support

- Hyper calls are inserted at compile time instead of runtime.
- The guest OS kernel is changed to replace privileged instructions with hyper calls before execution.
- Xen uses this approach.
- Allows guest OS to run with better performance by handling virtualization calls efficiently.

## Summary

- **Microkernel hypervisors** (Xen) focus on small, secure core, with device drivers outside.
- **Monolithic hypervisors** (VMware ESX) include many functions inside hypervisor.
- **KVM** leverages Linux kernel, uses hardware support, and combines emulation via QEMU.
- Virtualization mechanisms vary by how guest OS instructions are handled:
  - Binary translation slows performance but needs no guest changes.
  - Para-virtualization is faster but needs guest OS modifications.
- Understanding these helps choose the right virtualization tool and architecture.

# CPU Virtualization

- CPU Virtualization lets multiple operating systems run on a single physical machine.
- It uses *privilege levels (rings)* in the x86 CPU architecture:
  - **Ring 0**: Kernel (most privileged)
  - **Ring 3**: Applications (least privileged)
- OS runs in Ring 0 and needs direct hardware access.
- Virtualization adds a *Virtual Machine Monitor (VMM)* between OS and hardware.

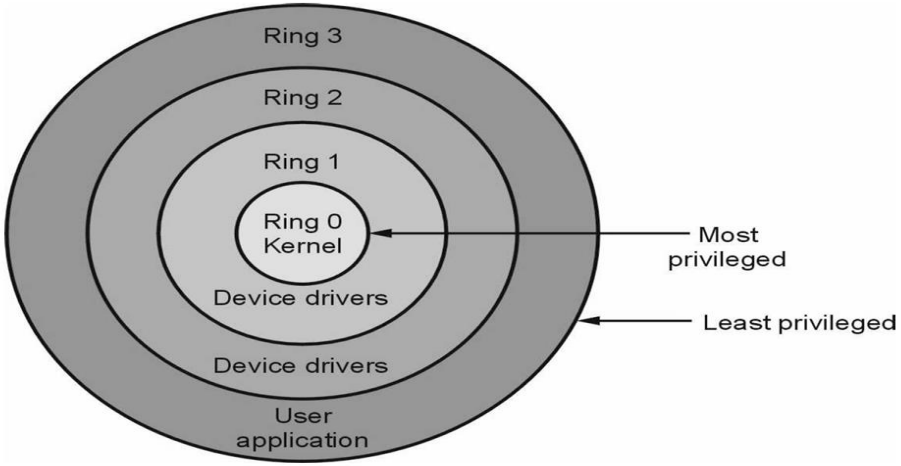



Fig. 2.10.1 CPU Privilege Rings

## Challenges in CPU Virtualization

- Some instructions (sensitive ones) behave differently depending on mode.
- These can't be easily virtualized.
- Types of sensitive instructions:
  - **Privileged**: Only work in Ring 0.
  - **Control-sensitive**: Change system settings.
  - **Behavior-sensitive**: Depend on system state. **VMM Role in Virtualization**

VMM runs in a special privileged mode.

- Guest OS runs in a lower mode.
- VMM traps and handles sensitive instructions.
- Ensures safety, correctness, and sharing of resources.

# Techniques for CPU Virtualization

## 1.Binary Translation (Full Virtualization)

- No OS changes needed.
- VMM rewrites sensitive code during runtime.
- Pros: Great isolation and no OS modification.
- Cons: Slower due to runtime translation.

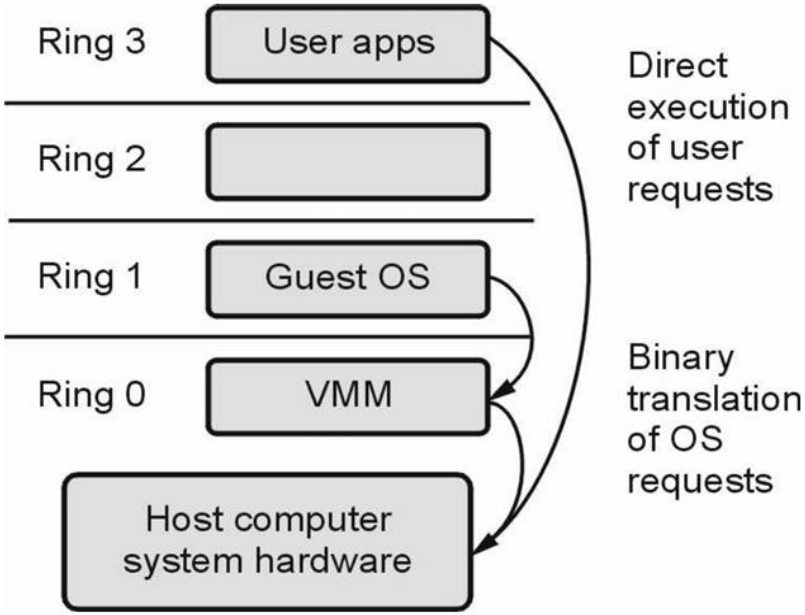



**Fig. 2.10.3 Binary Translation with Full Virtualization**

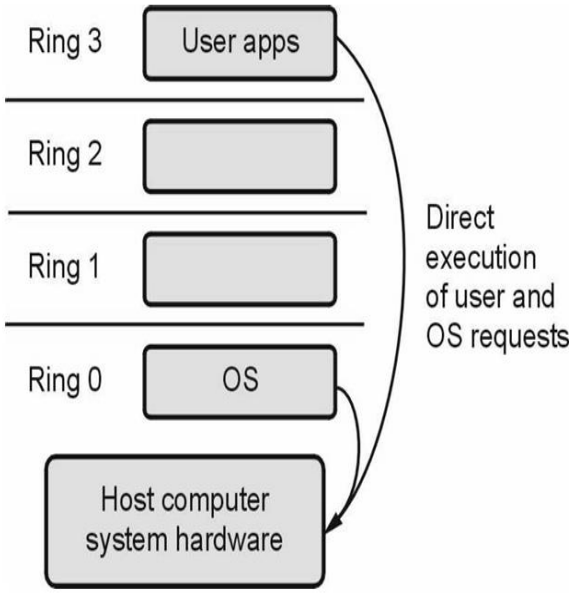



**Fig. 2.10.2 X86 privilege level architecture without virtualization**

## 2.Para-Virtualization (OS Assisted)

- Requires modifying the guest OS.
- Replaces sensitive instructions with **hyper calls**.
- Hyper calls are like system calls to the hypervisor.
- Pros: Better performance.
- Cons: OS must be changed.

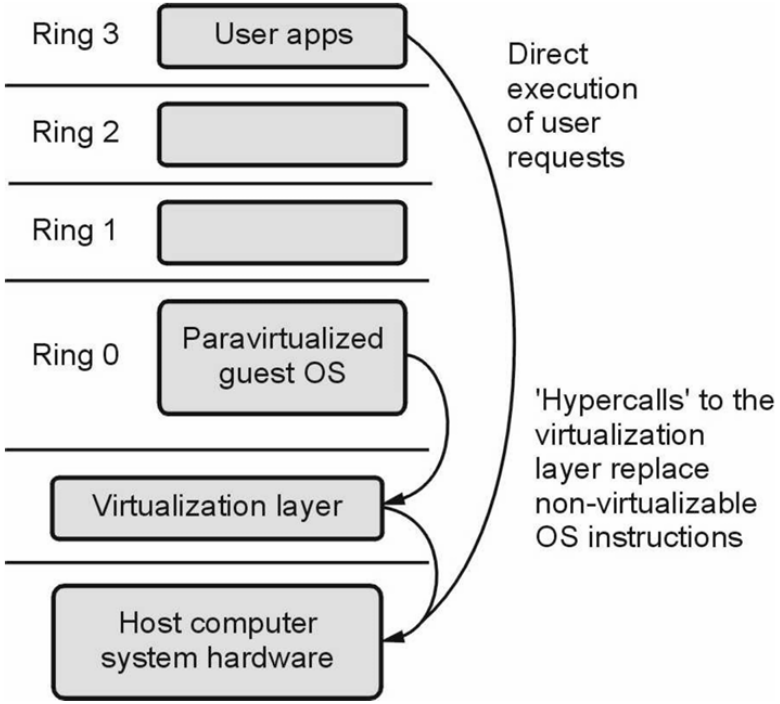



**Fig. 2.10.4 Para-virtualization**

## 3. Hardware Assisted Virtualization

- Supported by Intel VT-x and AMD-V.
- Adds a special **root mode** (Ring 0P) below Ring 0.
- Guest OS runs in a **non-root mode** (Ring 0D).
- Hardware traps sensitive instructions directly.
- Pros: No need for binary translation or OS modification.

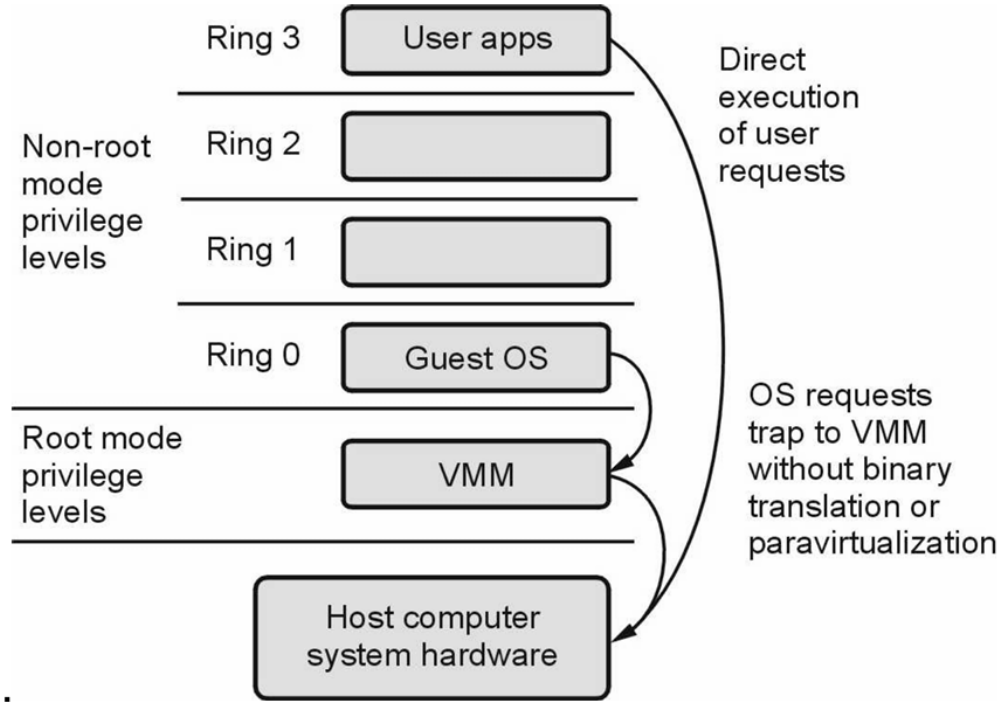



**Fig. 2.10.5 Hardware Assisted Virtualization**

## Summary

- CPU Virtualization improves system efficiency and resource sharing.
- Three techniques:
  - **Full Virtualization**: Uses binary translation.
  - **Para-Virtualization**: Needs OS changes.
  - **Hardware Virtualization**: Uses CPU support.
- VMM ensures secure, efficient virtualization of CPU resources.

# Memory Virtualization - Introduction

- Memory virtualization allows **multiple virtual machines (VMs)** to share physical memory.
- Each VM sees its own **virtual memory** as a continuous block, even if the actual memory is scattered.
- The Operating System (OS) manages the translation from **virtual memory to physical memory** using **page tables**.

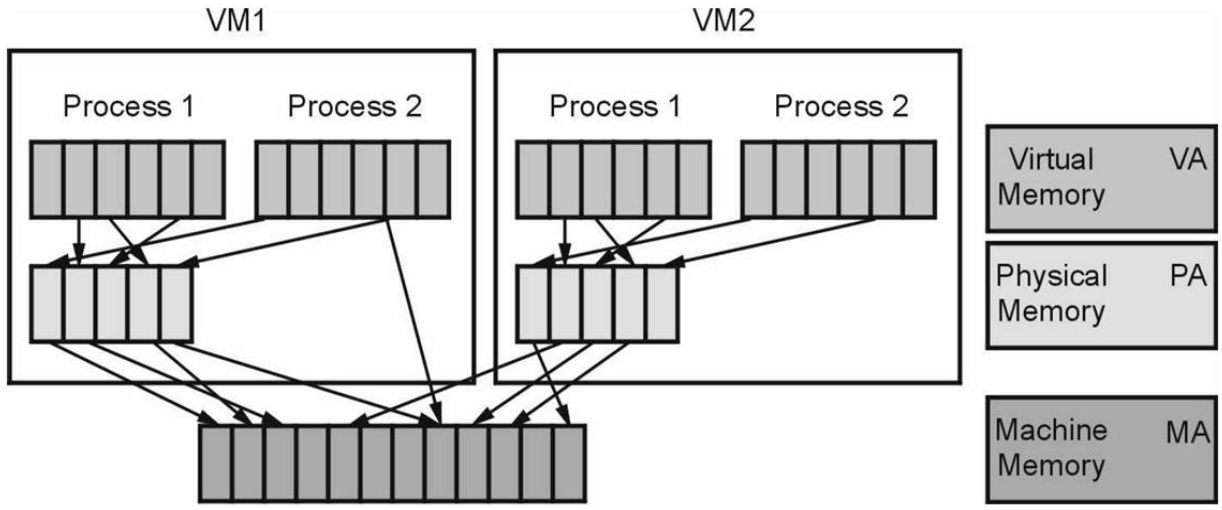



**Fig. 2.11.1 Memory Virtualization**

# Traditional vs Virtualized Memory

- In a normal (non-virtualized) system:
  - The OS maps **virtual memory → machine memory** using **one stage**.
- In a virtualized system:
  - Mapping is **two-stage**:
    - **Virtual memory → Guest physical memory** (by Guest OS)
    - **Guest physical memory → Machine memory** (by VMM)

## Role of MMU and TLB

- All modern CPUs (like x86) have a **Memory Management Unit (MMU)** and **Translation Lookaside Buffer (TLB)**.
- MMU handles memory address translations.
- TLB stores recently used translations to speed up memory access.

## How VMM Manages Memory

- Guest OS maps virtual to guest physical addresses.
- **VMM (Virtual Machine Monitor)** maps guest physical to actual machine memory.
- VMM uses **Shadow Page Tables** to keep track of these mappings.
- This avoids double translation on every memory access.

## Hardware-Assisted Memory Virtualization

- Technologies like **AMD's Nested Paging** help simplify memory virtualization.
- Hardware handles two-stage translation more efficiently.
- Improves performance by reducing the work of the VMM.

**Summary**

- Memory virtualization allows flexible sharing of physical memory among VMs.
- Two-stage translation is managed by the Guest OS and VMM.
- MMU, TLB, and Shadow Page Tables optimize performance.
- Hardware features (e.g., Nested Paging) further improve speed and efficiency.

# Virtualization of I/O Devices

## Introduction

- I/O virtualization is more complex than CPU virtualization.
- It manages I/O requests between virtual and physical devices.
- Software-based methods help enable features and simplify management.

## Role of Network in I/O Virtualization

- Network is key for communication between Virtual Machines (VMs).
- Virtual NICs and switches create virtual networks without using physical bandwidth.
- **NIC Teaming** combines multiple NICs into one, allowing failover and seamless VM relocation (e.g., VMware VMotion).

## Key Goals

- Maintain virtualization benefits with minimal CPU usage.
- Enable portability and standardization through virtual device emulation.
- Use standard device drivers across platforms.

## Methods of I/O Virtualiz

- Full Device Emulation
- Para virtualization
- Direct I/O Virtualization
- Self-Virtualized I/O

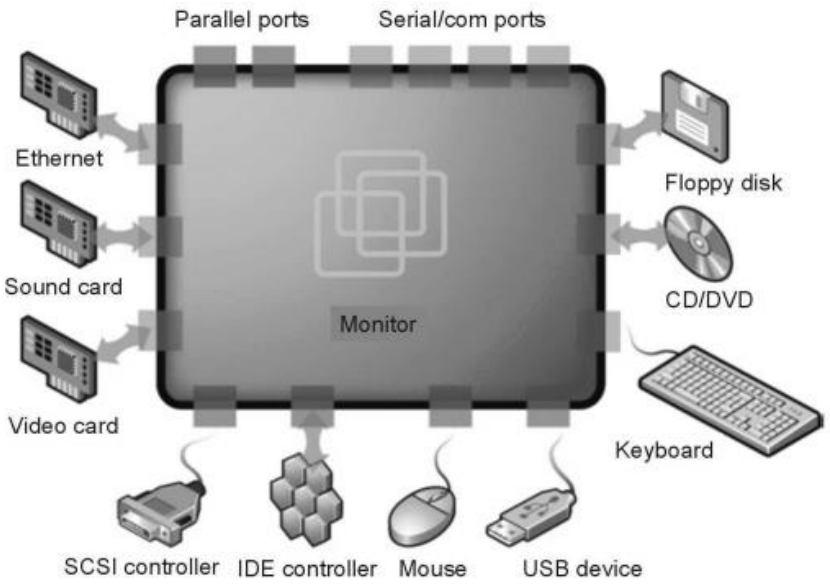



Fig. 2.12.1 Device and I/O virtualization

## 1.Full Device Emulation

- Emulates real hardware devices entirely in software.
- VMM traps I/O requests and handles them.
- Simple but **slow** due to high CPU usage.

## 2.Para virtualization

- Uses **split driver model**:
  - **Frontend driver** in Domain U (Guest OS)
  - **Backend driver** in Domain 0 (Manages real devices)
- Communicates via shared memory.
- **Better performance** than emulation but with **higher CPU usage**.

## 3.Direct I/O Virtualization

- VMs access I/O devices **directly**, no emulation or VMM needed.
- Offers **high performance** and **low CPU usage**.
- Suitable for **networking in mainframes**.

## 4.Self-Virtualized I/O

- Uses multicore processor resources.
- Provides **Virtual Interfaces (VIFs)** for each device (e.g., virtual disk, camera, network).
- Each VIF has a **unique ID** and **two message queues** (incoming & outgoing).
- Allows guest OS to interact via APIs and drivers.

## Challenges in I/O Virtualization

- Risks include:
  - System crash during device reassignment
  - Device malfunction
  - High overhead in emulation
- **Combining techniques** can help reduce these issues.

# Virtualization Support and Disaster Recovery in Cloud Computing

**Virtualization in Cloud Computing**

- Virtual machines (VMs) are containers for cloud services.
- Virtualization hides physical servers from users and developers.
- Developers don't worry about network or infrastructure issues like scalability or faults.
- Virtualization software runs hardware virtually and supports unmodified operating systems.

## Benefits of Virtualization

- Supports old software and operating systems.
- Provides ready environments for cloud app development.
- Allows provisioning of VMs on demand with great scalability.
- Offers flexibility to users and developers.
- Ensures high throughput, availability, and load balancing.
- Enables disaster recovery and centralized resource management.

## Applications of Virtualization

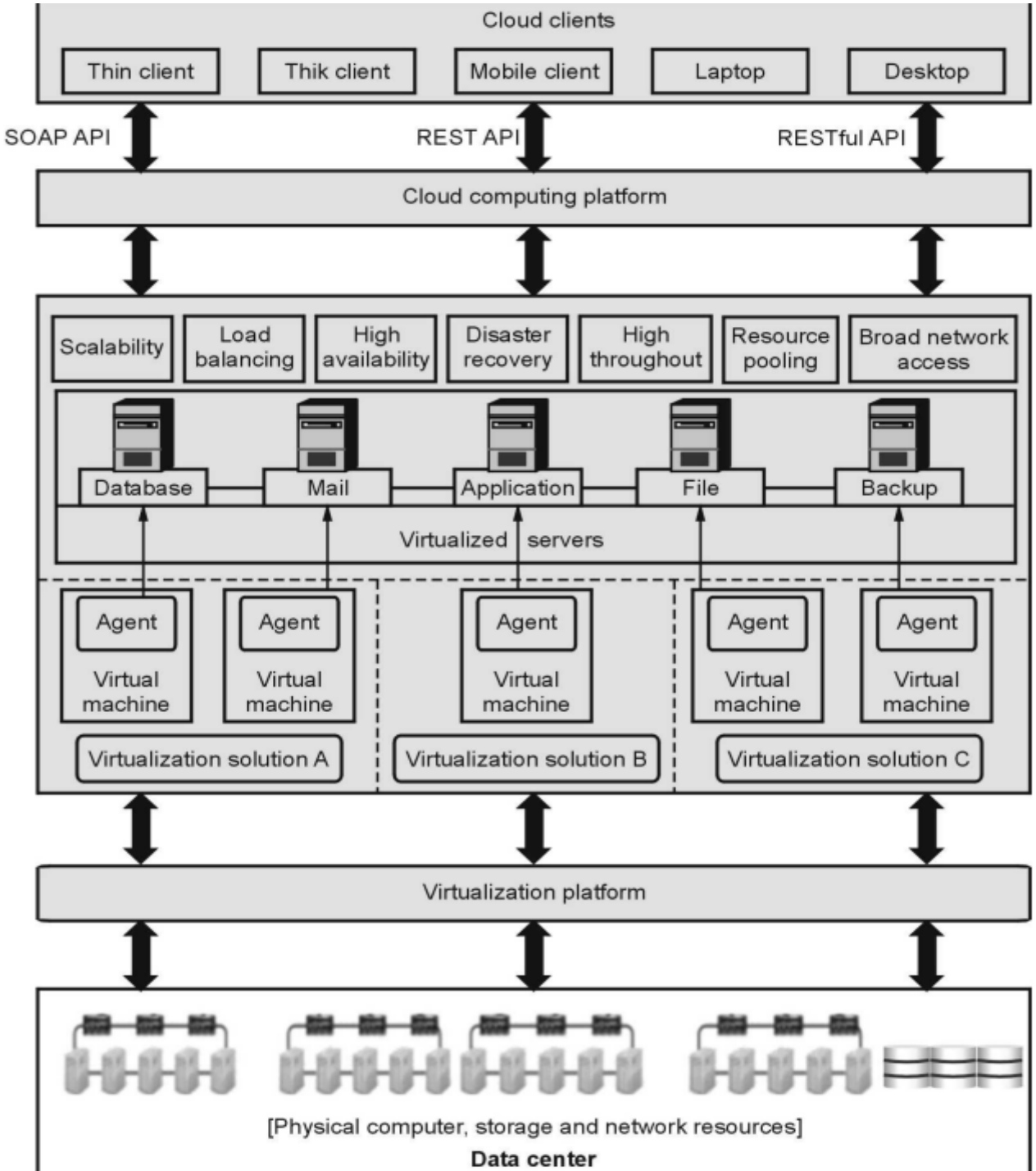
- **Public Cloud Platforms:**
  - Used by AWS, Google, Microsoft to save resources and energy.
  - Makes cloud services easy, effective, and reliable.
- **Green Data Centers:**
  - Reduces power use and costs by consolidating servers.
  - Helps fight energy crises by using fewer physical servers.

## Virtualization in IaaS

- VM technology lets users create custom cloud environments.
- Consolidates underused servers to save resources.
- Runs legacy code without conflicts.
- Improves security by sandboxing applications.
- Provides better quality of service and performance isolation.

# Virtualization and Disaster Recovery

- Disaster recovery ensures IT services continue during failures or disasters.
- Involves policies and tools to recover systems quickly.
- Traditional recovery (physical machines) is slow and costly.
- Virtualization speeds recovery by using VMs instead of physical machines.


Cloud clients
Thin client
Thik client
Mobile client
Laptop
Desktop
SOAP API
REST API
RESTful API
Cloud computing platform
Scalability
Load balancing
High availability
Disaster recovery
High throughout
Resource pooling
Broad network access
Database
Mail
Application
File
Backup
Virtualized servers
Agent
Virtual machine
Virtualization solution A
Virtualization solution B
Virtualization solution C
Virtualization platform
[Physical computer, storage and network resources]
Data center

## How Virtualization Helps Disaster Recovery

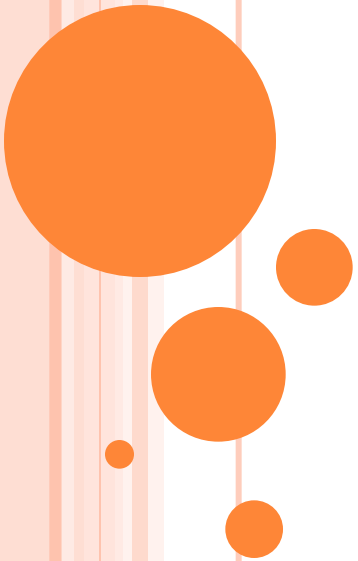
- VM platforms reduce OS install and setup time.
- Backup agents are not needed, speeding up recovery.
- VM cloning creates a copy of a VM on a remote server.
- Only one clone is active; others stay suspended until needed.
- When original VM fails, the clone activates quickly.
- Snapshots help live migration with minimal downtime.
- Provides better Recovery Point Objective (RPO) and Recovery Time Objective (RTO).

## Summary

- Virtualization is key for cloud flexibility, efficiency, and disaster recovery.
- Helps save energy, manage resources, and keep services running.
- Makes cloud computing reliable even during disasters.

## UNIT III

# CLOUD ARCHITECTURE, SERVICES AND STORAGE



# Cloud Architecture Design

**Importance of Cloud Architecture**

- Cloud architecture is key to creating simple, user-friendly cloud services.
- Main goals: scalability, reliability, efficiency, and virtualization.
- Design must support automated service delivery and management.

**Basic Requirements**

- Support latest web standards like Web 2.0 and REST APIs.
- Handle large-scale physical and virtual HPC infrastructure.
- Use loosely coupled architecture for flexibility.
- Provide easy access via self-service web portals.
- Efficiently manage user requests and resource provisioning.
- Ensure strong security for shared data center resources.
- Use cluster architecture for scalability and reliability.
- Deliver fast performance and flexible cloud services.

## Hardware and Software Role

- Advances in multicore CPUs, memory, and storage allow large data centers.
- Software standards like Web 2.0 and SOA help develop cloud services.
- SOA supports SaaS delivery by managing node status and tasks.
- Virtualization enables quick cloud delivery and disaster recovery.
- Most data centers are run by third-party providers.

# Layered Cloud Architecture

- Cloud architecture has three layers:
  - Infrastructure Layer
  - Platform Layer
  - Application Layer
- Layers use virtualization and standardization for hardware/software resources.

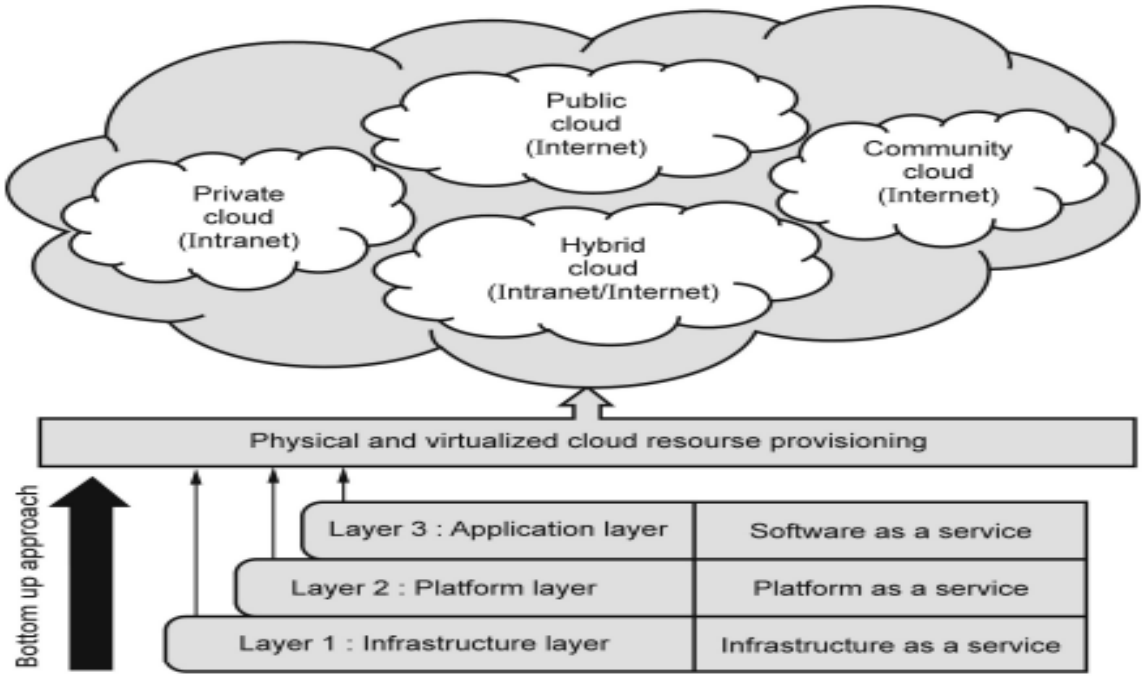



**Fig. 3.1.1 : Layered cloud architecture design**

**Infrastructure Layer (IaaS)**

- Provides computing, storage, and networking as virtual services.
- Manages virtual CPUs, memory, storage, and network resources.
- Handles automated resource provisioning and flexibility for users.
- Acts as foundation for platform and application layers.

**Platform Layer (PaaS)**

- Offers development and deployment tools for cloud apps.
- Allows users to build, test, and monitor applications online.
- Ensures scalability, reliability, and security.
- Acts as middleware between infrastructure and application layers.

**Application Layer (SaaS)**

- Provides on-demand software applications.
- Examples include office tools, CRM, finance, and supply chain software.
- Many applications use resources from multiple layers.
- Users rely on providers for hardware, platform, and software support.
- Example: Salesforce.com CRM service.

# NIST Cloud Computing Reference Architecture

- NIST provides a standard cloud model for secure cloud use.
- Works with IT vendors, industries, and governments worldwide.
- The model is vendor-neutral and supports innovation.
- It defines 5 main actors involved in cloud services:
  - Cloud Consumer
  - Cloud Provider
  - Cloud Auditor
  - Cloud Broker
  - Cloud Carrier

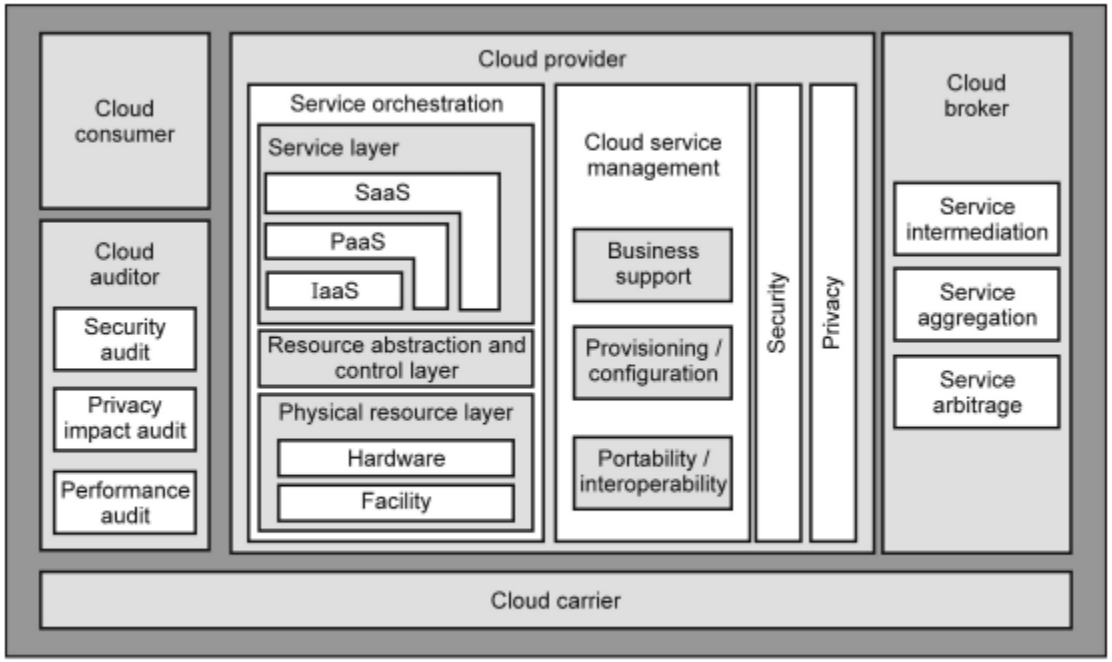



**Fig. 3.2.1 : Conceptual cloud reference model showing different actors and entities**

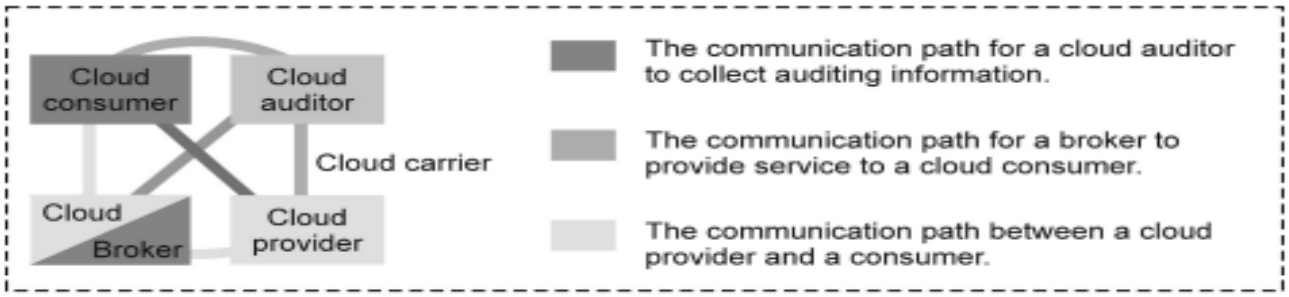



**Fig. 3.2.2 : Interactions between different actors in a cloud**

- **Cloud Consumer**
- The most important actor who uses cloud services.
- Can get services directly from a provider or via a broker.
- Chooses services and signs contracts; gets billed for use.

- Examples:
  - 1. Requesting services from brokers who combine or improve services.
  - 2. Using cloud services delivered via a carrier with secure SLAs.
  - 3. Cloud auditor checks security and operations for consumers.

- 1. Requesting services from brokers who combine or improve services.


**Fig. 3.2.3 : Cloud broker interacting with cloud consumer**

- 2. Using cloud services delivered via a carrier with secure SLAs.

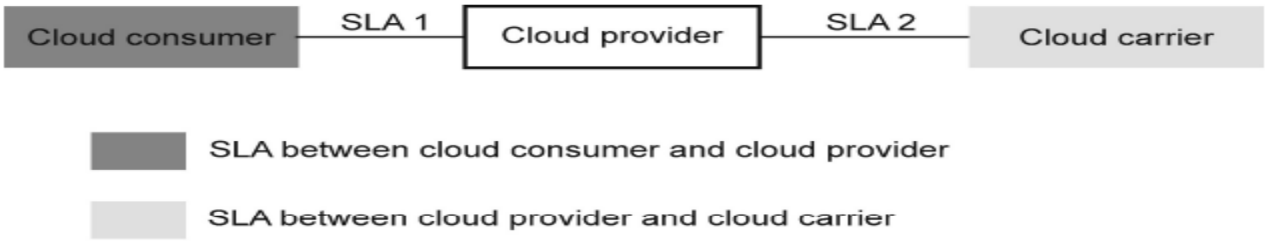



**Fig. 3.2.4 : Scenario for cloud carrier**

- 3. Cloud auditor checks security and operations for consumers.

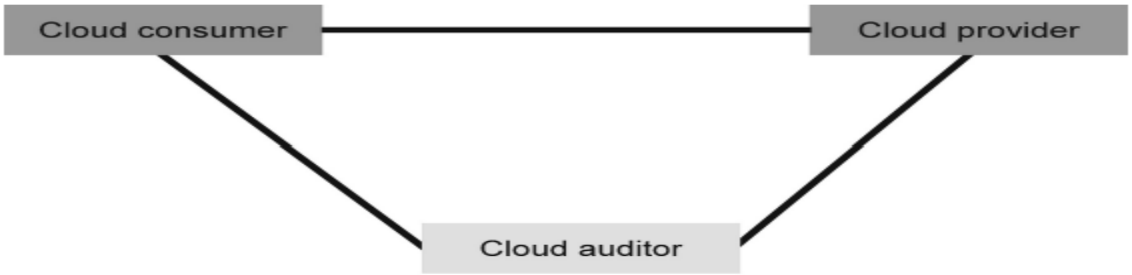



**Fig. 3.2.5 : Usage scenario involving a cloud auditor**

# Cloud Consumer Service Types

- **SaaS users** access software like office tools and CRM via the internet; billed by usage.
- **PaaS users** get tools and platforms for app development and testing; billed by resource use.
- **IaaS users** get virtual computers, storage, and networks; billed for hardware, bandwidth, and CPU time.

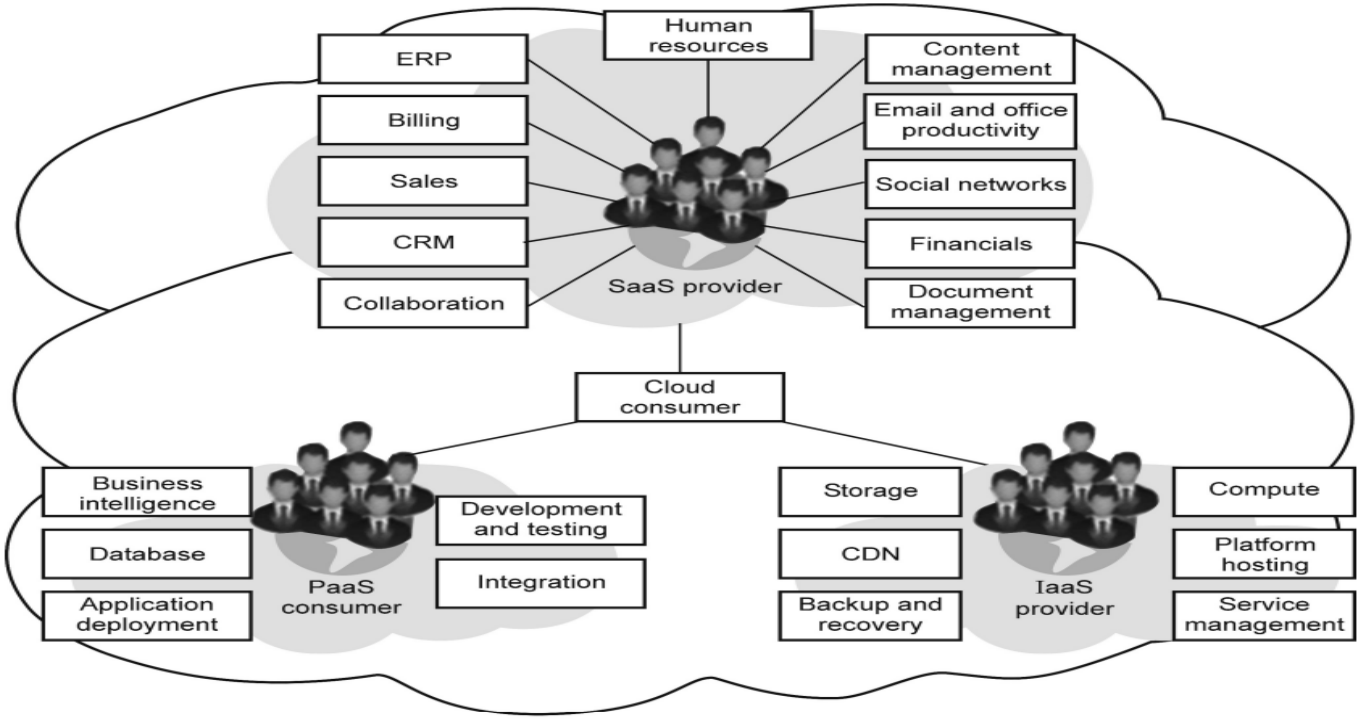



Fig. 3.2.6 : Example of cloud services available to cloud consumers

- **Cloud Provider**
- Provides cloud services and manages the infrastructure.
- Operates software to deliver services over networks.

- Responsibilities vary by service type:
  - **SaaS providers:** Manage applications and updates; users have limited control.
  - **PaaS providers:** Manage platform and tools; users control applications but not the infrastructure.
  - **IaaS providers:** Manage physical resources and virtual machines; users control software running on VMs.

## Cloud Provider Main Activities

- **Service Deployment:** Sets up private, public, hybrid clouds.
- **Service Orchestration:** Coordinates cloud resources for cost-effectiveness and business needs.
- **Cloud Service Management:** Operates and manages services for users.
- **Security:** Ensures security from physical layers to applications.
- **Privacy:** Protects user data, controls access, and maintains privacy.

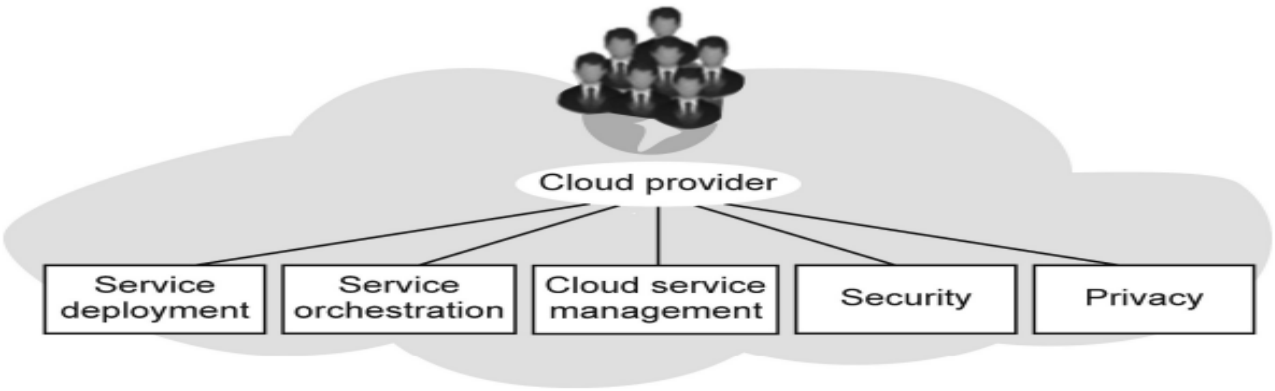



Fig. 3.2.7 : Major activities of a cloud provider

## Cloud Auditor

- Independently checks cloud services for security, privacy, and performance.
- Provides honest reports based on objective evidence.
- Helps both consumers and providers ensure compliance with standards.

## Cloud Broker

- Acts as a middleman between consumers and providers.
- Manages service requests, performance, and delivery.
- Types of broker services:
  - **Service Intermediation:** Adds extra value to services.
  - **Service Aggregation:** Combines multiple services into one.
  - **Service Arbitrage:** Chooses the best services from different providers.

## Cloud Carrier

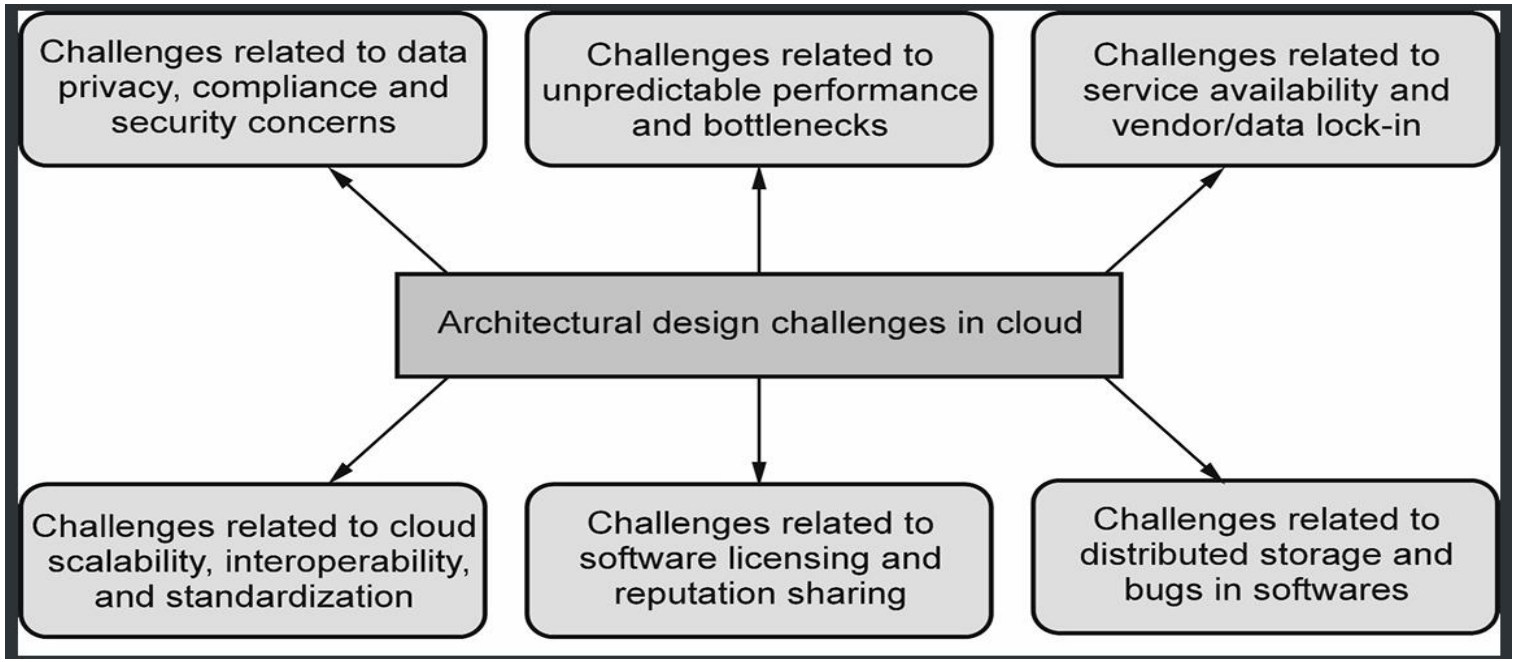
- Connects cloud consumers and providers.
- Provides network access via telecom links for devices (computers, tablets, phones).
- Sets up SLAs with providers to ensure reliable service transport.
- Offers secure, fast, and dedicated network links.

## Summary

- NIST architecture defines roles and their responsibilities clearly.
- Cloud consumers use services; providers manage infrastructure and delivery.
- Auditors ensure trust and compliance.
- Brokers simplify service access and improve offerings.
- Carriers maintain reliable connections between all parties.

# Architectural Design Challenges in Cloud Computing

- The cloud architecture design plays an important role in making cloud services successful in all aspects, but still it has some challenges. The major challenges involved in architectural design of cloud computing are



# Data Privacy, Compliance & Security

- Cloud systems mostly use public networks, which are vulnerable to attacks.
- Common threats: malware, spyware, DoS, VM rootkits, etc.
- New risks: VM hijacking, guest-hopping, and man-in-the-middle attacks.
- Use encryption, firewalls, and secure network tools to prevent threats.
- Follow local data protection laws and review cloud SLAs for compliance.

## Performance & Bottlenecks

- VMs share hardware (CPU, memory, I/O), leading to performance issues.
- Shared I/O can slow down systems and create bottlenecks.
- Performance worsens when I/O is distributed across cloud regions.
- Fixes:
  - Upgrade I/O systems.
  - Remove weak servers.
  - Optimize operating systems for virtualization.

## Service Availability & Lock-In

- Businesses rely on cloud for critical apps — outages = big losses.
- A single failure point can impact entire operations.
- Solution:
  - Use **multiple cloud providers** for better uptime.
  - Use scalable infrastructure to handle DDoS attacks.
- Vendor/data lock-in makes switching providers hard.
- Use open software stacks and standard APIs to ease migration.

## Scalability, Interoperability & Standards

- Pay-as-you-go billing helps scale resources as needed.
- Different platforms (Google, AWS) use different models.
- Need for common VM formats like **OVF** for better portability.
- Virtual appliances should work across platforms.
- Support cross-platform migrations (e.g., Intel ↔ AMD).

## Software Licensing & Reputation

- Cloud providers mostly use open-source due to flexible licensing.
- Commercial licenses must adapt to cloud's usage model.
- Bad behavior (e.g., spam) by one user can harm the provider's reputation.
- Build tools to protect cloud service reputations.
- SLAs must clearly state legal responsibilities.

## Storage & Debugging Issues

- Cloud databases keep growing — need flexible, scalable storage (like SANs).
- Data consistency is hard to test in large distributed environments.
- Bugs in real systems are hard to reproduce.
- Use **VMs** and **simulators** for better debugging and testing.

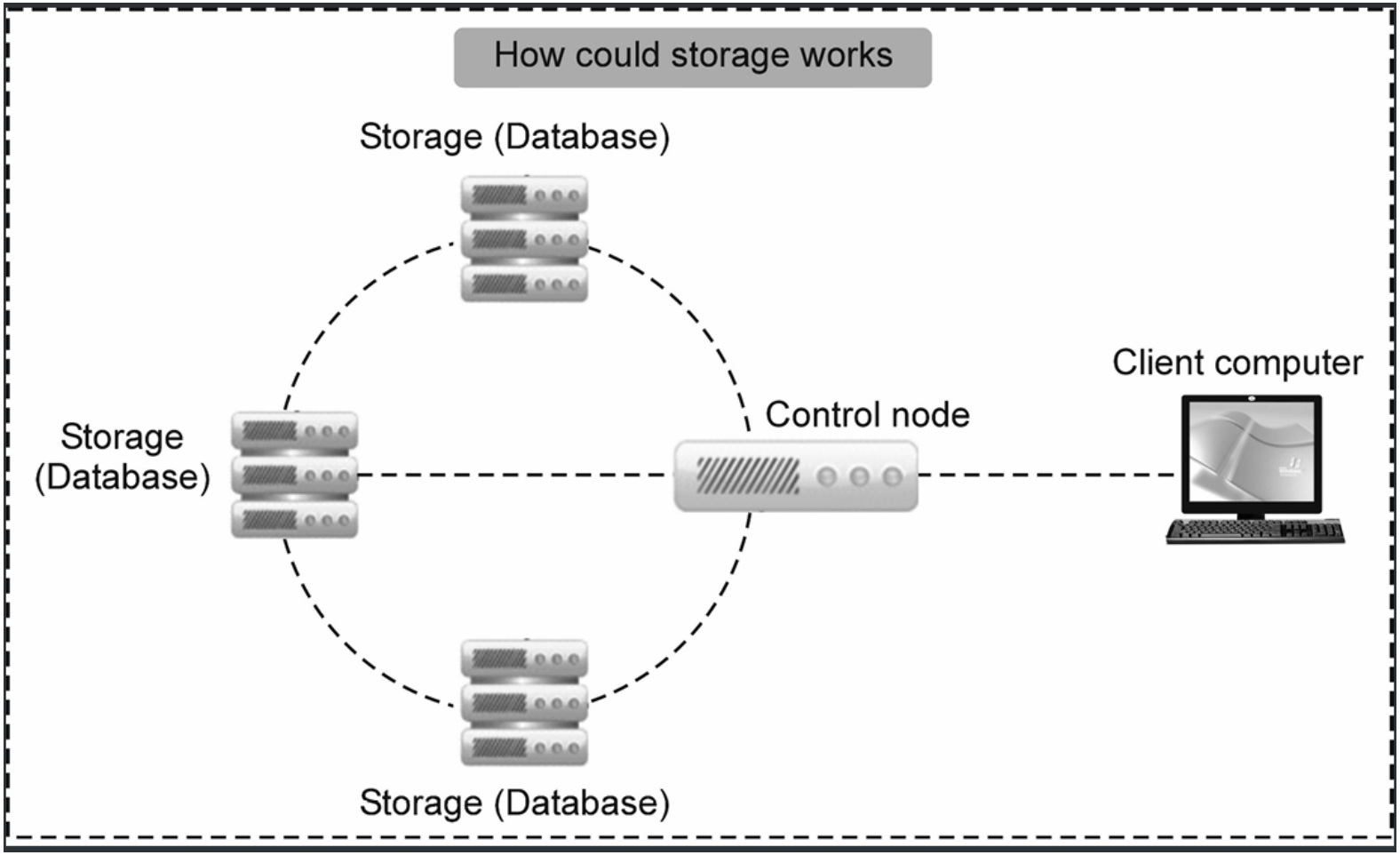
# Cloud Storage

**Cloud Storage**

- Cloud storage is saving digital data in online storage instead of local devices.
- Data is stored in logical pools managed by a **cloud storage provider**.
- Example: Gmail stores emails on their servers, not your device.

## Traditional vs Cloud Storage

- Earlier: Data was saved on **hard drives**, **CDs**, or **USBs**.
- Now: Data is stored on **remote servers** accessed via the **internet**.
- Advantage: Saves local space and allows access from anywhere.
- **How Cloud Storage Works**
- A client computer sends data to **cloud servers** through the internet.
- Data is stored on **multiple servers** for safety (redundancy).
- Users can **access**, **edit**, or **download** files using a **web interface**.
- Storage providers manage hardware, backups, and security.


How could storage works
Storage (Database)
Storage (Database)
Control node
Client computer
Storage (Database)

## Benefits of Cloud Storage

- Store huge amounts of data (even exabytes).
- Access files instantly from anywhere.
- Easy integration into apps with unified APIs.
- Data is kept safe, backed up, and always available.

## Redundancy & Reliability

- Cloud providers use **multiple servers** to avoid data loss.
- If one server fails, another has a copy of your data.
- This ensures **high availability** and **data protection**.

## Cloud Sync vs Cloud Backup

- **Cloud Sync:**
- Keeps files updated across all devices.
- Works in one-way or two-way sync.
- Good for people using **multiple devices**.
- Examples: Google Drive, Dropbox, iCloud, OneDrive.
- **Cloud Backup:**
- Sends a **copy of files** to the cloud for safekeeping.
- Works **automatically** in the background.
- Useful for disaster recovery.
- Examples: Backblaze, Carbonite, iBackup.

## Conclusion

- Cloud storage is a safe, scalable, and flexible way to store data.
- You can **store**, **sync**, or **back up** your files based on your needs.
- It helps keep your data accessible, protected, and easy to manage.

# Storage as a Service (STaaS)

## Introduction

- Storage as a Service (STaaS) = Renting storage from third-party providers.
- Useful for small & medium businesses with:
  - Limited budget
  - No technical staff
  - No storage infrastructure.
- Customers pay **only for data stored or transferred**, not for hardware.

## How It Works

- Data stored remotely on cloud servers.
- Providers manage storage hardware & software.
- Customers access data anytime, anywhere through client software.
- Supports **public and private cloud connections**.

## Benefits

- **Cost savings** – No need to buy storage hardware.
- **Disaster recovery** – Protects and restores data after failures.
- **Business continuity** – Ensures availability of data.
- **Scalability** – Pay as you grow (per GB or transfer).
- **Saves physical space and staff effort**.

## Features & Methods

- **Backup & Restore** – Protects data loss.
- **Disaster Recovery** – Replicates VMs for emergencies.
- **Block Storage** – Low-latency I/O.
- **SSD Storage** – High performance for intensive operations.
- **Object Storage** – For analytics & cloud apps.
- **Cold Storage** – Long-term, low-cost retention.
- **Bulk Data Transfer** – Moving large datasets efficiently.

**Popular STaaS Providers**

- **Amazon S3, Google Cloud, IBM, HPE, Dell EMC, NetApp, Rackspace**.
- **Google Drive** – 15 GB free, scalable to 10 TB.
- **Microsoft OneDrive** – 5 GB free, scalable to 5 TB; ransomware protection.
- **Dropbox** – 2 GB free, up to 5 TB; works on all OS.
- **MediaMax & Strongspace** – Rent cloud storage for any digital data.

**Conclusion**

- STaaS = **affordable, flexible, and scalable** storage option.
- Ideal for small/medium businesses & individuals.
- Ensures **data security, disaster recovery, and continuity** with minimal investment.

## Advantages of Storage as a Service (STaaS)

- **1. Cost Efficiency**
- Reduces high expenses of traditional backup.
- Offers ample cloud space at low monthly charges.

**2. Space & Visibility**

- No physical storage required → saves office space.
- Cloud storage is invisible in deployment.

**3. Security & Automation**

- Data is encrypted in transit & at rest.
- Automated backups: users choose *what* & *when*, system handles the rest.

**4. Accessibility & Syncing**

- Access data from any device (smartphones, tablets, PCs).
- Files sync automatically across devices → always updated.

**5. Sharing & Collaboration**

- Easy file sharing with few clicks.
- Multiple users can edit same file → avoids confusion on latest versions.

**6. Data Protection & Recovery**

- Cloud protects data from natural disasters & human errors.
- Copies stored at multiple sites ensure **business continuity**.
- Supports disaster recovery for critical systems.

- **Disadvantages of Storage as a Service**

**1. Downtime Issues**

- Vendor/server failures can cause service unavailability.
- Serious risk for mission-critical data.

**2. Limited Customization**

- Infrastructure owned by provider → less flexibility for users.

**3. Vendor Lock-In**

- Hard to migrate to another provider once data is stored.

**4. Reliability Concerns**

- Smaller providers may be unstable.
- Risk of crashes or loss of access.
- Redundancy often used to improve reliability.

# Advantages of Cloud Storage

**1. Accessibility**

- Access files anytime, anywhere with internet.
- Works across devices (PC, laptop, mobile, tablet).
- Eliminates stress of file transfers.

**2. Collaboration**

- Easy sharing of files/folders without long email chains.
- Real-time collaboration → auto-saves updates.
- Improves teamwork & productivity.

**3. Security**

- Encryption prevents unauthorized access.
- Providers add extra security layers.
- Protects confidential & sensitive data.

**4. Cost Efficiency**

- No need for physical drives or servers.
- Saves cost of external storage devices.
- Cheaper per GB than hardware storage.

**5. Backup & Recovery**

- Quick recovery from hardware failures.
- Acts as backup for locally stored files.
- Ensures minimal downtime.

**6. Syncing & Disaster Recovery**

- Automatic updates across all devices.
- Cloud keeps secondary copies of critical files.
- Supports disaster recovery & business continuity.

## Risks in Cloud Storage

### 1. Vendor Dependency (Lock-in)

- Difficult to migrate data between providers.
- Limited user control over infrastructure.

### 2. Unintended Permanence

- Deleted files may still exist on servers.
- Users rely heavily on provider's backup system.

### 3. Insecure Interfaces & APIs

- SOAP/REST APIs vulnerable to attacks.
- Requires strong authentication & encryption.

### 4. Compliance Risks

- Providers may fail audits or industry standards.
- Serious for regulated industries (healthcare, finance).

# Disadvantages of Cloud Storage

**1. Privacy Concerns**

- Data stored on third-party platforms.
- Risk if providers outsource to unknown firms.

**2. Internet Dependency**

- No access/upload if internet fails.
- Affects data transfer & recovery.

**3. Compliance Issues**

- Some nations restrict cross-border data transfer.
- Risk of penalties or service shutdown.

**4. Vulnerability to Attacks**

- Cloud data exposed to hacking attempts.
- Sensitive data may still be at risk.

**5. Data Management Challenges**

- Cloud structures may not match business systems.

**6. Protection Concerns**

- Involves trusting third-party with sensitive details.
- Critical to select reliable, established providers.

# Cloud Storage Providers (MSPs)

## Overview

- Cloud Storage Providers = Managed Service Providers (MSPs).
- Host customer data in **remote data centers**.
- Customers lease storage **monthly or on-demand**.
- Pay-as-you-go model: storage, computing, software, security.
- Access files anytime via internet instead of local devices.

## Benefits for Organizations

- **Cost control** – low per-GB pricing.
- **Elasticity** – scale resources on demand.
- **Self-service** – manage resources independently.
- Removes limits of on-site storage.
- Helpful for **SMEs with limited IT budgets/staff**.

## Popular Cloud Storage Providers

### 1. Amazon S3

- Simple interface to store/retrieve unlimited data.
- Highly scalable – same infrastructure used by Amazon.com.
- Goal: leverage economies of scale & pass benefits to developers.

### 2. Google Bigtable Datastore

- Fast, highly scalable datastore.
- Scales across **thousands of servers** storing petabytes.
- High speed, versatility & scalability.
- Part of **Google App Engine** for developers.

**3. Microsoft Live Mesh**

- Free file synchronization for Windows & Mac.
- Syncs files/folders across multiple devices.
- Supports formats: Atom, RSS, JSON, XML.
- Uses Live Framework APIs for secure sharing.

**4. Nirvanix CloudNAS**

- Offers **public, hybrid & private cloud storage**.
- Usage-based pricing model.
- Ideal for **archival, backup & unstructured data**.
- Features: built-in **disaster recovery** & **auto-replication** across 3+ locations.

# Amazon Simple Storage Service (S3)

**Overview**

- Amazon S3 = Cloud-based storage & backup system.
- Provides **secure, scalable, fast & low-cost** storage.
- Same infrastructure used by Amazon's global websites.
- Supports **online data backup & archival storage**.

**Key Features**

- Store & retrieve **any amount of data** from anywhere.
- Data objects: size range **1 byte – 5 GB**.
- Stored in **buckets** (containers, not hierarchical file systems).
- Buckets must have **unique names** in S3 namespace.
- Access via **SOAP or REST API** (REST preferred).

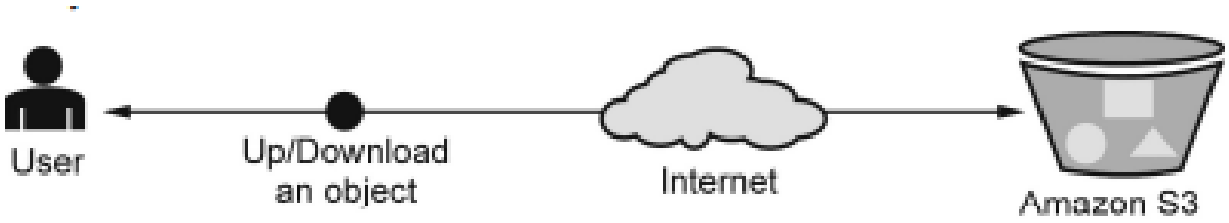



Fig. 3.10.1 : AWS S3

## Functionalities (APIs)

- Create, modify, or delete buckets.
- Upload/download objects.
- Search & identify objects.
- Access metadata of objects/buckets.
- Specify bucket storage location.
- Allow public access when needed.

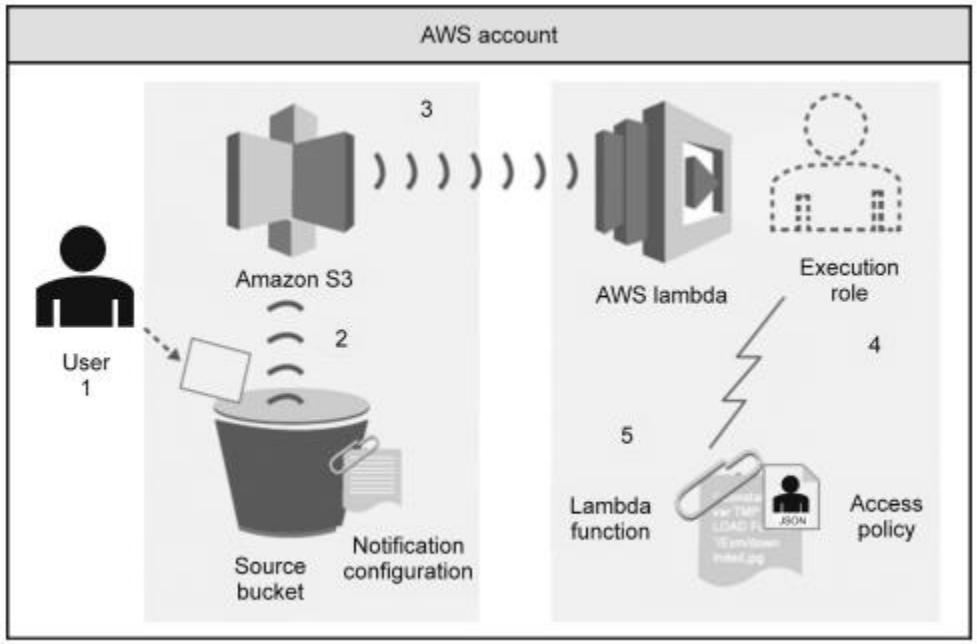



Fig. 3.10.2 : Source bucket

## Performance & Use Cases

- Ideal for **data archiving, retrieval, disk backups**.
- Popular with sites storing/sharing large files (e.g., images, photos).
- Provides **large, reliable, protected storage** with low-bandwidth access.

**Backup Method (3-2-1 Rule)**

1 → Original data.

2 → Local copy.

3 → Off-site copy in S3.

Ensures **data redundancy & protection**.

**Versioning in S3**

- Must be enabled by the user.
- Retains **all versions** of stored objects.
- Operations (PUT, POST, COPY, DELETE) create new versions.
- GET retrieves latest version, but old ones remain for recovery.
- Useful for **archiving & undoing mistakes**.

## Amazon Glacier

- **Overview**
- Low-cost, secure, durable **archival storage service**.
- For data **not frequently accessed**.
- Retrieval time: **3–5 hours**.
- Stores **any format, any size** of data (ZIP, TAR common).

## Glacier Vs S3

- Both amazon S3 and amazon glacier work almost the same way. However, there are certain important aspects that can reflect the difference between them.

| Amazon Glacier | Amazon S3 |
|---|---|
| It supports 40 TB archives | It supports 5 TB objects |
| It is recognised by archive IDs which are system generated | It can use "friendly" key names |
| It encrypts the archives automatically | It is optional to encrypt the data automatically |
| It is extremely low - cost storage | Its cost is much higher than Amazon Glacier |

# Cloud Architecture Design – Key Points

- **Design Goals**: Scalability, reliability, efficiency, and virtualization are the four essential goals of cloud platforms.
- **Layered Architecture**:
  - Three layers: **Infrastructure, Platform, Application**
  - Built on **virtualization & standardization** of hardware and software.
- **NIST Cloud Reference Architecture**:
  - Developed with IT vendors, industries, standards developers, and government agencies.
  - Ensures **cloud security standards** and their continuous development.
- **Deployment Models**:
  - **Public Cloud** – runs on internet, accessible to all.
  - **Private Cloud** – used internally, often on intranet.
  - **Hybrid Cloud** – mix of multiple models, offering combined benefits.
  - **Community Cloud** – shared by many organizations for a common cause.

- **Cloud Service Models**:
  - **IaaS** – Provides servers, storage, networking, virtualization.
  - **PaaS** – Provides a platform to develop and run applications without managing infrastructure.
  - **SaaS** – On-demand software delivery for end users.
- **Challenges in Cloud Design**: Data privacy, security, compliance, performance, interoperability, standardization, service availability, licensing, storage issues, and bugs.
- **Cloud Storage**:
  - Stores, manages, and backs up data remotely.
  - Scales up to **exabytes of data** efficiently.
- **Storage as a Service (StaaS)**:
  - Outsourced model – third-party providers rent storage space to users lacking budget.
  - Providers are known as **Managed Service Providers (MSPs)**.
- **Amazon S3**:
  - Web service for storing and retrieving unlimited data.
  - Secure, scalable, fast, low-cost.
  - Same infrastructure Amazon uses for its global website.

# UNIT IV

# RESOURCE MANAGEMENT AND SECURITY IN CLOUD

# Inter Cloud Resource Management

**Introduction**

- Resource management = allocation of compute, storage, networking & energy resources.
- Goal: meet performance needs of **providers, users, and applications**.
- Challenges: large data centers, resource heterogeneity, interdependence, unpredictable loads, diverse objectives.

**Importance of Inter-Cloud Resource Management**

- Services deployed on **common resource pools** from federated servers.
- Cloud brokers may use **shared servers across platforms**.
- Downtime of a single server may cause **business loss**.
- Inter-cloud management ensures **efficient resource provisioning** & continuity.

## Cloud Architecture Layers

- Based on **NIST Architecture** → Infrastructure, Platform, Application.
- Services: **IaaS, PaaS, SaaS** (built on top of each other).
- **Five functional layers in practice**:
  - **HaaS** – hardware resources.
  - **IaaS** – compute, storage, network.
  - **NaaS & LaaS** – networking & colocation control.
  - **PaaS** – application deployment platform.
  - **SaaS** – on-demand application delivery.

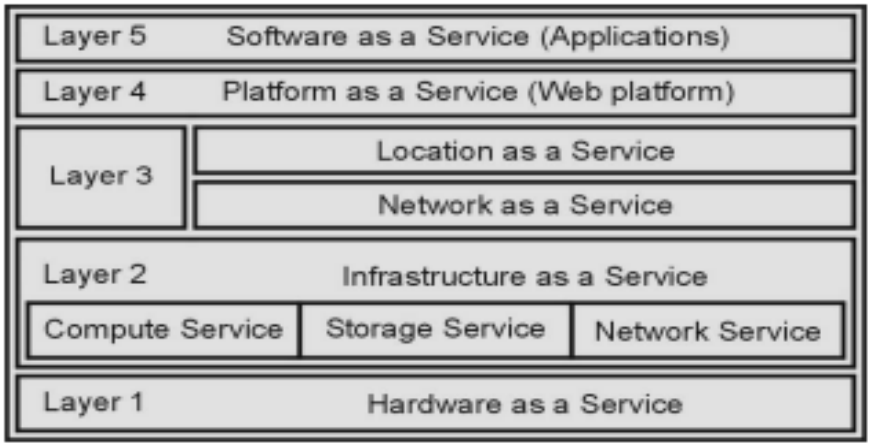



**Fig. 4.1.1 Functional layers of Cloud computing**

## Provider & User Concerns

- **Providers**: focus on infrastructure performance.
- **Users**: demand QoS, service delivery, and security.
- Example: **CRM** was the first SaaS success → sales, marketing, promotions.
- SaaS also supports **collaboration, HR, finance**.

## Inter-Cloud Resource Provisioning

- Must ensure: **High throughput, High Availability (HA), Fault tolerance**.
- Cloud uses **physical & virtual servers** for flexibility.
- **VMs decouple services** from specific hardware.
- Software layer enables **large-scale data storage**.

**Runtime Support**

- **Cluster monitoring**: tracks cloud performance.
- **Schedulers**: queue tasks & allocate nodes.
- Ensures **high efficiency & scalability**.
- SaaS removes customer need for servers/software investment → **low cost & flexibility**.
- Data stored in **private/public cloud via PaaS & IaaS**.

# Resource Provisioning and Methods

**Introduction**

- Cloud computing improves **software & hardware design**.
- Focus: **VM instances, CPU cores, cluster-level parallelism**.
- Resource provisioning = allocating **compute, storage, network, energy** resources.
- Challenges: unpredictable demand, heterogeneity, failures, power management, SLA conflicts.

## Provisioning of Compute Resources

- Providers sign **SLAs** committing CPU, memory, bandwidth.
- **Under-provisioning** → SLA violations & penalties.
- **Over-provisioning** → underuse, reduced revenue.
- Requires **automated VM provisioning** (deployment, migration, failure recovery).
- Examples:
  - **Amazon EC2** – Xen VMM, VM templates.
  - **IBM Blue Cloud** – Xen, no templates.
  - **Microsoft Azure** – virtualization-based provisioning.
- Key concern: **energy efficiency** (cooling, caching, query processing).

## Provisioning of Storage Resources

- Storage sits on **physical/virtual servers**.
- Needs **distributed file systems** & large-scale databases.
- Examples:
  - **GFS, HDFS, Azure Cosmos FS** → large-scale distributed storage.
  - **Amazon DynamoDB** (Key-Value NOSQL), **Amazon S3** (object storage).
- Storage technologies: **SCSI, SATA, SSDs, Flash** → reliable high-performance storage.
- Databases: **Google BigTable, Amazon SimpleDB, DynamoDB, Azure SQL**.

## Provisioning in Dynamic Deployment

- Uses **VMs as building blocks** across resource sites.
- **Inter Grid model**: builds cloud execution across grids via gateways (IGG).
- IGG tasks:
  - Request VMs
  - Authorize leases
  - Deploy VMs
- Provides **resource sharing, scalability, and fault tolerance**.
- Distributed Virtual Environments (DVEs) manage isolation & provisioning.

## Resource Provisioning Methods

- **Demand-Driven Provisioning**
  - Allocates resources dynamically based on usage thresholds.
  - Example: AWS Auto-Scaling on EC2.
  - Pros: simple, efficient.
  - Cons: fails with abrupt workload spikes.
- **Event-Driven Provisioning**
  - Adds/removes resources at specific **time events**.
  - Good for **seasonal/predictable workloads**.
  - Risk: wrong predictions → wasted resources.
- **Popularity-Driven Provisioning**
  - Allocates based on **application popularity trends**.
  - Useful for **viral apps, trending traffic**.
  - Risk: wrong prediction → wasted resources.

# Global Exchange of Cloud Resources

## Introduction

- IaaS providers build **datacenters worldwide** for redundancy & reliability.
- Example: **Amazon** asks SaaS customers to choose hosting regions.
- Weaknesses:
  - Customers don't know consumer origins → hard to pick best location.
  - QoS requirements may not be met globally.

## Challenges in Global Cloud Hosting

- A single provider **cannot build datacenters everywhere**.
- Difficult to guarantee QoS for global SaaS customers.
- Global businesses (e.g., **media hosting, Internet services, Web 2.0 apps**) need multi-provider support.

# Intercloud Architecture

- Proposed to **federate multiple cloud providers**.
- Enables **resource sharing** and **brokerage** across clouds.
- Benefits:
  - Scalable applications.
  - Competitive, market-driven leasing of compute & storage.
  - Reliable, affordable, QoS-aware services.
- Uses **virtualization** to virtualize software services & federated infrastructures.

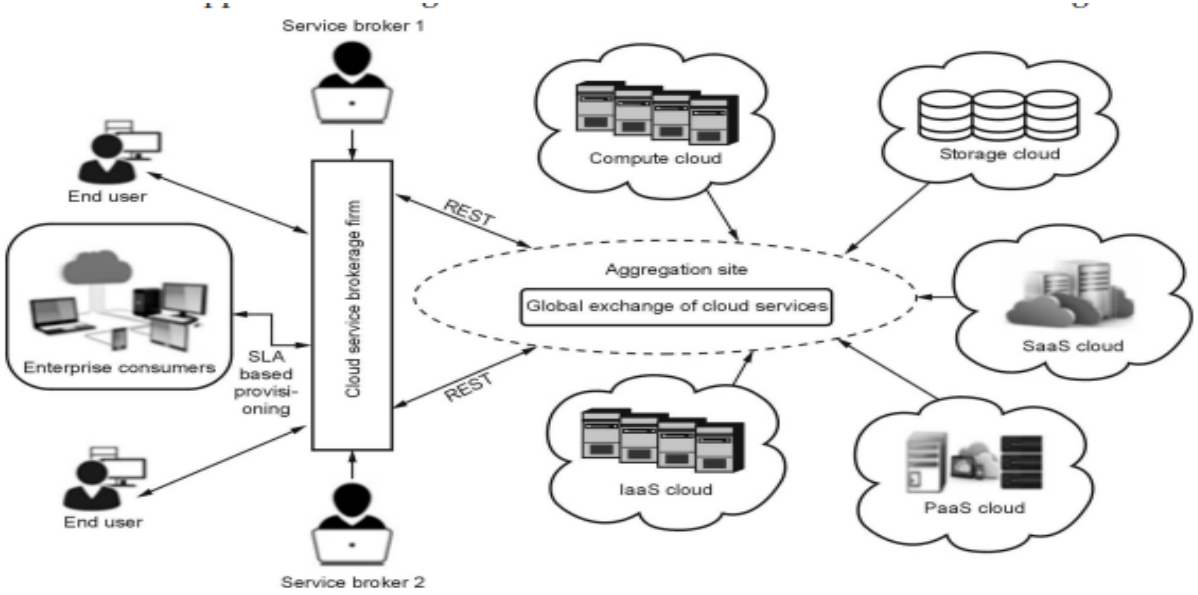



**Fig. 4.3.1 Intercloud Architecture**

## Federation Advantages

- Providers can **expand/redimension capacity** by leasing from others.
- Example: **Salesforce.com** hosting services via SLA-based federation.
- Supports **market-based utility models** for heterogeneous applications.

## Cloud Exchange (CEX)

- Core element of Intercloud:
  - Analyzes **application broker demands vs. provider supply**.
  - Acts as a **market authority** for cloud service trading.
  - Uses **economic models** → commodity pricing, auctions.
- Provides **scheduling, resource allocation, workload migration**.
- Ensures **on-demand, adaptable, reliable infrastructure access**.

## SLA (Service Level Agreement)

- Defines service metrics, incentives, and penalties.
- Ensures **safe, reliable transactions** between participants.
- Guarantees QoS in federated cloud environments.

# Global Exchange of Cloud Resources

## 📌 Need for Global Cloud Expansion

- IaaS providers set up datacenters worldwide for redundancy & reliability.
- Example: Amazon requires SaaS customers to select hosting regions.

Limitations:

- Customers can't predict user origins.
- SaaS providers face difficulty meeting QoS across regions.

## ⚠ Challenges

- No single provider can set up datacenters everywhere.
- Hard to ensure global QoS standards.
- Businesses with worldwide apps (Internet services, media hosting, Web 2.0) need **multi-provider solutions**.

# Inter cloud Architecture

- Proposed to **federate multiple cloud providers**.
- Enables brokerage & sharing of resources across clouds.
- Expands capacity dynamically by leasing compute & storage from others.
- Supports **scalability, cost efficiency & QoS guarantees**.

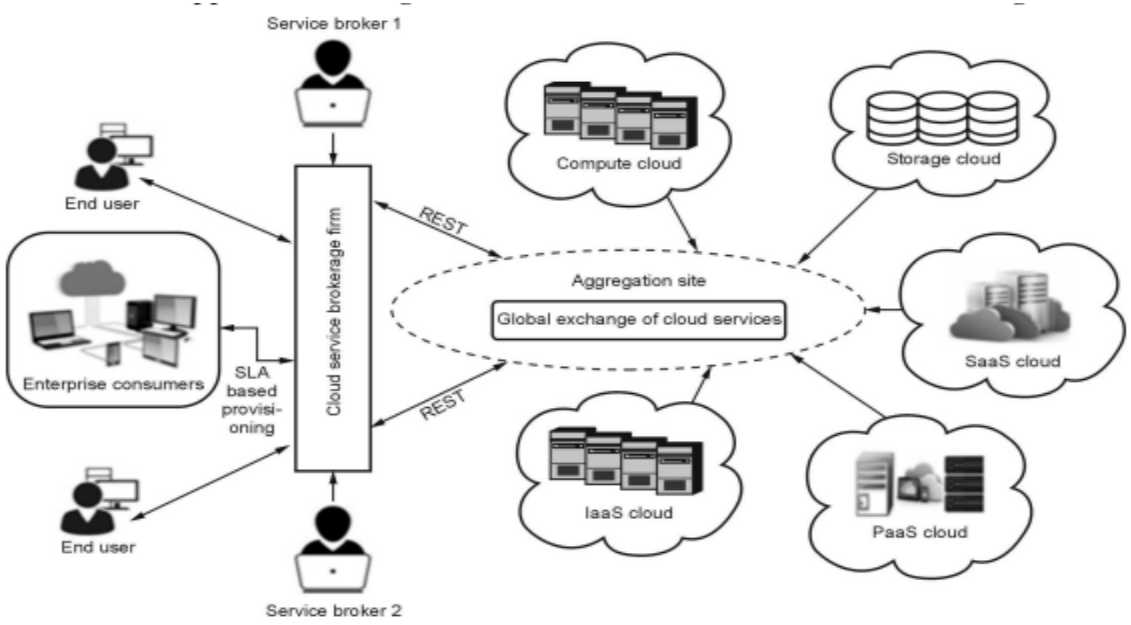



Fig. 4.3.1 Intercloud Architecture

## Features & Benefits

- SLA-based resource leasing (e.g., Salesforce.com).
- Virtualization ensures:
  - Reliable & on-demand services.
  - Affordable operations.
  - QoS-aware service delivery.
- Market-driven federation using **utility models** for heterogeneous apps.

## Functional Components

- **Intercloud architecture** consolidates distributed storage & compute into a single abstraction.
- Includes:
  - Client brokerage & coordination services.
  - Scheduling, resource allocation, workload migration.
- Facilitates cross-domain integration for **adaptable & reliable infrastructure**.

# Cloud Exchange (CEX)

- Matches demand (application brokers) with supply (providers).
- Acts as a **market authority** for service trading.
- Uses **competitive economic models** (commodity pricing, auctions).
- SLA framework ensures:
  - Agreed service metrics.
  - Incentives & penalties.
  - Secure and reliable transactions.

# Security Overview in Cloud Computing

## 📌 Why Companies Move to Cloud

- On-demand service with high compute & storage.
- Benefits: low cost, no maintenance, scalability, load balancing, location independence, quick deployment.

## ⚠ Key Security Concerns

- Data privacy, trust, confidentiality & integrity.
- Migration of critical apps & sensitive data → beyond traditional perimeter defense.
- Lack of trust between providers & users slows adoption.
- New threats due to virtualization require new data protection models.

# Enforcement of Cloud Security

- **Physical Security**: biometrics, CCTV, motion detection, man-traps.
- **Network Security**: firewalls, IDS, vulnerability assessments.
- **Data Security**: SSL, encryption, strong password policies, malware defense.

# Cloud Infrastructure Security

## Importance

- Infrastructure = core of SPI model (IaaS, PaaS, SaaS).
- Needs protection at **Network, Host & Application levels**.

## 1. Network Level Security

- Challenges in **public, private & hybrid clouds**.
- Key factors:
  - Data confidentiality & integrity (HTTPS, encryption).
  - Access control (authentication, authorization, auditing).
  - Availability of internet-facing resources (e.g., BGP prefix hijacking).
  - Replace zones/tiers with **domain-based security models**.

## 2. Host Level Security

- SaaS/PaaS: CSP responsible for host security, with NDA-based assurance.
- IaaS: Customer must secure **hypervisor & VM instances**.
- Common threats:
  - Account hijacking, key theft, unpatched services, trojans.
- Recommendations:
  - Disable password-based logins; use key-based access.
  - Configure host firewalls, enable IDS/IPS.
  - Regular log reviews, VM integrity protection, role-based access.

## 🛡 3. Application Level Security

- Critical for SaaS & PaaS success.
- Joint effort: Dev teams + Security teams.
- Methods:
  - Secure coding standards, penetration testing, firewalls, IDS/IPS.
- Common cloud app attacks:
  - Cross-site scripting (XSS), SQL injection, malicious file execution.
- SaaS: secure hosted apps.
- PaaS: secure platform, DB, runtime engines.

## ✅ Conclusion:

Cloud adoption requires **multi-layered security** across network, host & applications. With robust controls, trust, privacy & compliance can be ensured for safe cloud operations.

# Cloud Security Challenges

## Key Concerns

- Virtualization & SaaS increase efficiency but raise **security risks**.
- Shared cloud environments → loss of **control & visibility**.
- Risk of **data seizure** if another tenant violates compliance.
- **Vendor lock-in**: incompatibility between storage platforms (e.g., AWS S3 vs IBM Blue Cloud).

## Data Security Issues

- Need for **end-to-end encryption** (SSL in transit & storage).
- Unclear ownership of **encryption/decryption keys** (client vs vendor).
- **Data integrity**: no common standard to ensure correctness/consistency.

## ⚠ SaaS Development Risks

- Secure **SDLC** is critical for in-house cloud coding.
- Mashup/web service combinations → new vulnerabilities.
- SaaS providers must supply **real-time logs** for compliance.
- Rapid app updates (every few weeks) challenge traditional **security cycles**.

## 🔄 Business Continuity

- **Failover mechanisms** often overlooked → downtime risk.
- Security must shift to **device & data level** for protection everywhere.

## Compliance Challenges

- Cloud complicates enforcement of **IT security standards**.
- Regulations require:
  - Data segregation on shared servers.
  - Residency rules (e.g., financial data stays in-country).
- Mobile/cloud access bypassing corporate networks increases risk.

## Virtualization Risks

- Multi-tenant VMs on shared hardware expand attack surface.
- Internet-based management access → more exposure.
- VM migration between clouds → **new vulnerabilities**.
- Hard to **audit security status** of VMs.

## 🔧 Security Responsibility

- Subscriber (not CSP) responsible for **patching VMs**.
- Perimeter security: firewalls, IDS/IPS, segmentation, DMZs.
- CSP must ensure **privacy & protection of customer data**.

## 🔒 Privacy Issues in Cloud

- **Compliance**: must follow country-specific laws (HIPAA, GLBA, FISMA, Patriot Act).
- **Storage**: multi-copies across datacenters, user unaware of location.
- **Retention**: CSP policies define how long data is kept.
- **Access**: users have right to view/delete their data.
- **Auditing**: organizations must verify CSP log & monitoring practices.
- **Destruction**: uncertainty if data is truly deleted at end-of-life.
- **Breaches**: lack of CSP transparency may hide security incidents.

# SaaS Security in Cloud Computing

## 📌 SaaS in Cloud Future

- SaaS will remain **dominant cloud service model**.
- Supported by Web 2.0 collaboration & XaaS models.
- Creates **new business opportunities** but also **security challenges**.

## 🔑 Importance of SaaS Security

- Vendors act like **Managed Service Providers (MSPs)**.
- Businesses must review **data protection policies** before adoption.
- Security practices & monitoring = **critical requirements**.

# Gartner's 7 SaaS Security Concerns

**1 Data Location**

- Can provider disclose **where data resides**?

**2 Data Segregation**

- **Encryption must be effective & expert-tested**.

**3 Recovery**

- Disaster recovery:
  - Will data be fully restored?
  - **How long will it take?**

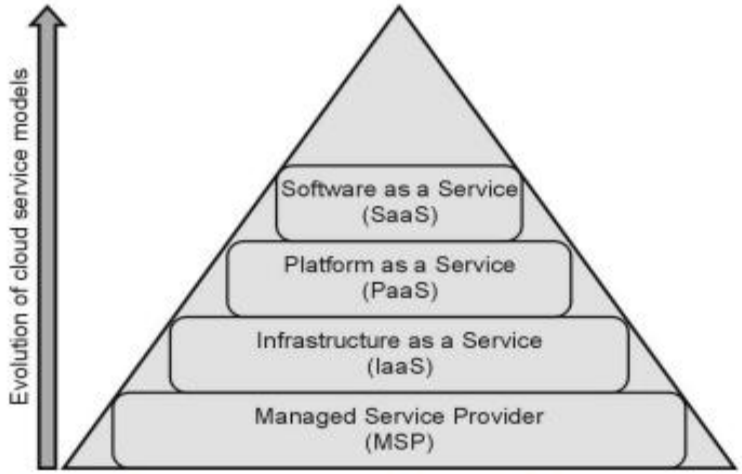



**Fig. 4.6.1 Evolutionary steps in the cloud service models**

**4Privileged User Access**

- Who has **administrative access**?
- How are they vetted & managed?

**5Regulatory Compliance**

- Provider must allow **external audits** & certifications.

**6 Long-Term Viability**

- If vendor fails:
  - What happens to data?
  - In what **format is data restored**?

**7Investigative Support**

- Ability to track & investigate **illegal activity**.

## 🔒 Data Protection Practices

- **Self-Encryption**
- Users should encrypt sensitive data themselves.
- Trustworthy algorithms + user-held **private keys**.
- Challenge: **key management** in pay-per-use cloud.

## 🧰 Evolving SaaS Security

- SaaS providers must:
  - Enhance existing **MSP-style practices**.
  - Develop **new methods** for evolving cloud threats.
- Structured **security agreements & leadership**:
  - Clear roles & responsibilities.
  - Encourages **shared ownership & accountability**.

## ✅ Conclusion:

SaaS dominance demands **robust encryption, compliance, recovery planning, and clear governance** to protect data and maintain trust.

# Security Governance in Cloud

## Overview

- Security management committee aligns **IT & business strategies**.
- Outcome: **Security Charter** – defines roles & responsibilities.
- Lack of governance → risks, compliance failures, missed opportunities.

## 🛡 Key Components of Security Governance

**1.Risk Assessment**

- Informed decisions balancing **business goals & asset protection**.
- Threat modeling for **apps, infra & processes**.

**2.Risk Management**

- Identify assets, data, ownership.
- Ensure **confidentiality, integrity, availability**.
- Maintain **asset repository** & continuity plan.

**3.Third-Party Risk Management**

- Assess vendors & partners.
- Failure → **reputation loss, legal issues, revenue impact**.

**4.Security Awareness**

- Train staff based on **roles**.
- Prevent social engineering, data leaks

**5.Security Portfolio Management**

- Prioritize projects via **portfolio discipline**.
- Use trained managers + formal methodologies.

**6.Standards, Guidelines & Policies**

- Create, document, review annually.
- Ensure **relevance & compliance** with evolving business needs.

**7.Training & Education**

- Build **skills in security, risk, data protection**.
- Continuous mentorship & role-based programs.
- **8.Monitoring & Incident Response**
- Centralized **SIEM & SOC** integration.
- Expand to app & data-level monitoring.

**9.Sales Support & RFIs**

- Security team supports **customer queries (RFI/RFP)**.
- Ensures **trust, compliance & competitiveness**.

**10.Business Continuity & DR**

- Minimize downtime & costs.
- SaaS enables **resilient BC/DR solutions**.

# 11. **Secure Software Development Life Cycle (SSDLC)**

- Defines threats & risks.
- Develops controls to combat them.
- Ensures **consistency, repeatability & conformity**.
- 6 Phases of SSDLC.

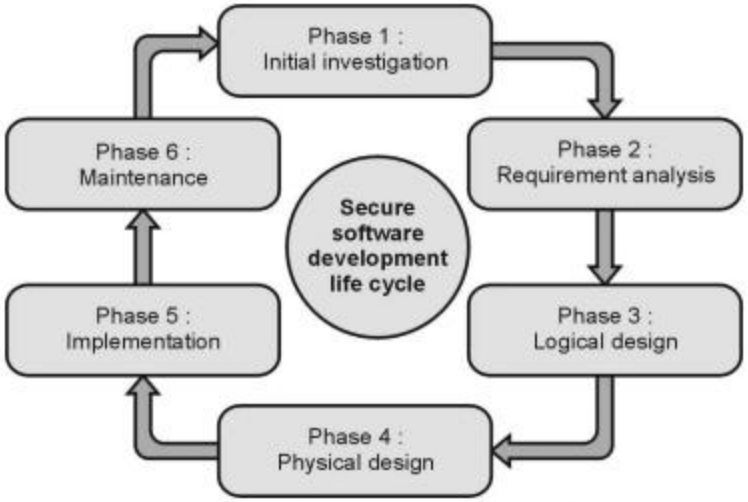



**Fig. 4.7.1 Secure Software Development Life Cycle**

**1.Initial Investigation**

- Define/document **project goals & processes**.
- Align with **security policy**.

**2.Requirement Analysis**

- Review **policies & systems**.
- Assess **emerging threats & legal issues**.
- Perform **risk analysis**.

**3.Logical Design**

- Create **security plan**.
- Plan **incident & disaster response**.
- Decide on **in-house or outsourced execution**.

**4.Physical Design**

- Select **technologies & solutions**.
- Design **physical & technical controls**.
- Review & approve security plans.

**5.Implementation**

- Build or purchase security solutions.
- Deliver **tested package for approval**.

**6.Maintenance**

- Continuous **monitoring & testing**.
- Include **penetration testing & data classification**.
- Provide **formal training & awareness**.

12. **Security Architectural Framework**

- Implements:
  - **Authentication, Authorization, Access Control**.
  - **Confidentiality, Integrity, Non-repudiation, Security mgmt.**
- Applies across all **enterprise applications**.

# Virtual Machine Security

## Traditional Threats

- Buffer overflows
- DoS attacks
- Spyware, malware, rootkits
- Trojan horses, worms

## Cloud-Specific VM Threats

- **Hypervisor malware**
- **Guest hopping**
- **VM hijacking / rootkits**
- **Man-in-the-middle attacks** (during VM migration)
- Passive attacks → steal sensitive data
- Active attacks → manipulate kernel structures

## 🛡 Defense Mechanisms

- **Network-level IDS**
- **Hardware-level IDS**
- Shepherding programs (code execution control)
- Verification mechanisms

## ⚙ Additional Security Technologies

- **RIO's vSafe & vShield**
- **Hypervisor enforcement**
- **Intel VPro**
- Hardened OS & Sandboxing
- Dynamic optimization infrastructure

## 🔥 Security Controls in VMs

- Firewalls
- Intrusion detection & prevention
- Integrity monitoring
- Log inspection

## (IAM)

### 📌 Importance of IAM

- Vital for every organization
- SaaS customers expect **least privilege principle**
- Access only when needed & for minimum time
- Balancing **usability vs. security** is critical

### 📌 Cloud Challenges for IAM

- Dynamic, on-demand services
- Evolving authentication & authorization models
- Need for **end-to-end trust & identity** across enterprises

## IAM Framework

- Policy & governance-driven
- Manages **digital identities** & controlled access
- Involves **processes, components, services, and practices**

## IAM Key Features

- User management
- Authentication & Authorization management
- Credential & Attribute management
- Compliance management
- Monitoring & Auditing

# Identity Lifecycle

- Creation → Maintenance → Termination of identities
- Provisioning & Deprovisioning support

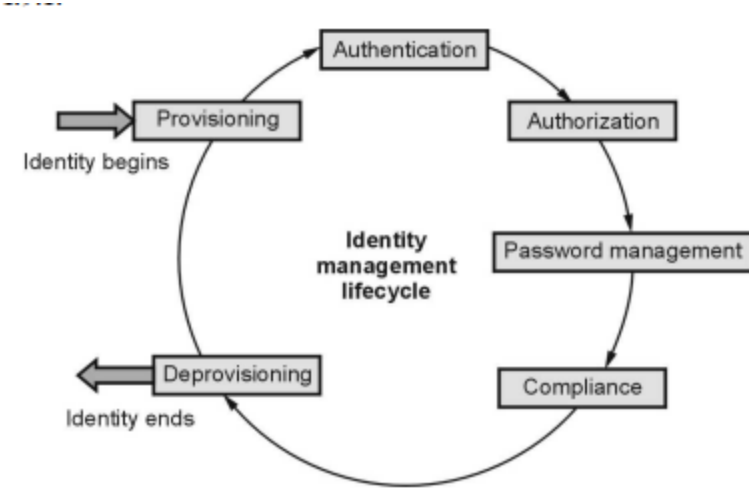



Fig. 4.9.1 Lifecycle of Identity Management

# ⚙ IAM Core Processes (Fig. 4.9.2)

- User Management
- Authentication Management
- Access Management
- Data Management
- Authorization Management
- Monitoring & Auditing

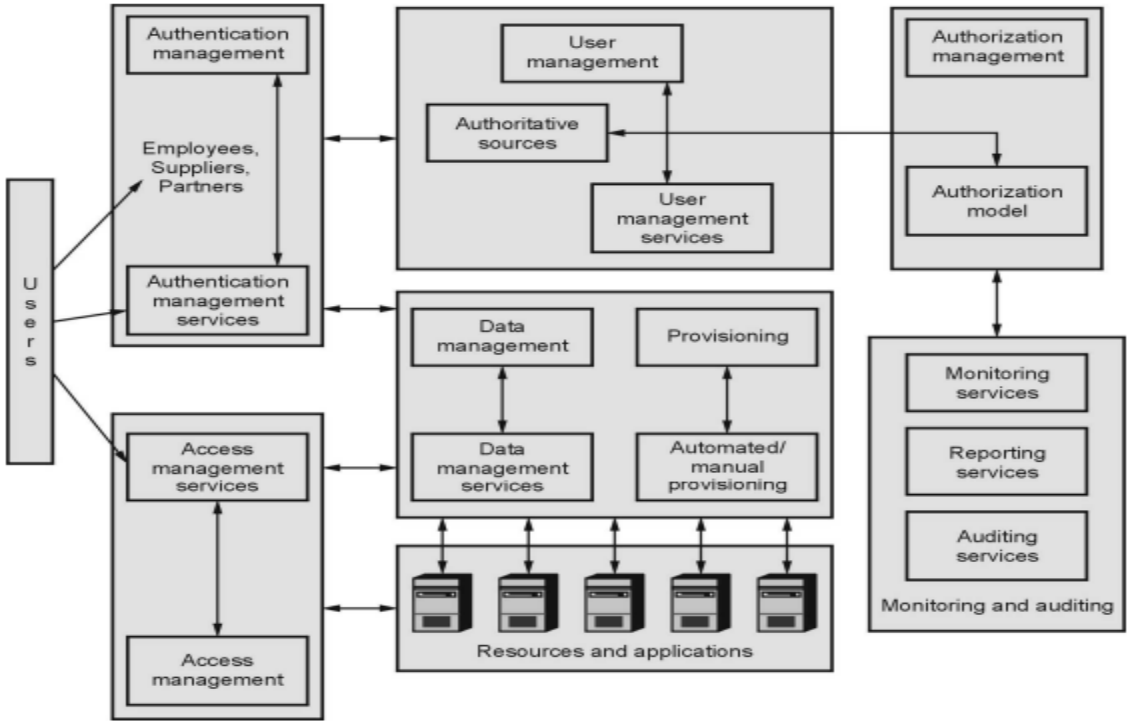



Fig.4.9.2 IAM Architecture

## 👤 User Management

- Create, modify, suspend, delete identities
- Manage passwords & accounts
- Provisioning → Grant access
- Deprovisioning → Remove access

## 🔑 Credential & Attribute Management

- Prevents impersonation
- Manages credentials & attributes
- Password standards, encryption, revocation

## ✅ Compliance Management

- Ensures **security policy adherence**
- Auditing, reporting, access monitoring
- Helps meet regulatory standards

## 🌍 Identity Federation Management

- Manages trust **beyond enterprise boundaries**
- Enables secure info sharing across organizations

## 📜 Entitlement Management

- Defines **authorization policies**
- Provisioning/deprovisioning of privileges
- Controls access to systems, apps, databases

## 🎯 Benefits of IAM

- Secure, policy-driven identity governance
- Enhanced compliance & auditing
- Stronger protection in SaaS & cloud models

# Security Standards in Cloud

## Importance

- Define processes, measures & practices for secure cloud environment
- Protect **confidential data & privacy**
- Ensure trusted, compliant operations

## Key Standards

- **Security Assertion Markup Language (SAML)**
- **Open Authentication (OAuth)**
- **SSL & TLS Protocols**

## 🔑 SAML (Security Assertion Markup Language)

- Developed by **OASIS**
- Enables **Single Sign-On (SSO)**
- Open standard for **authentication & authorization data exchange**
- Uses **XML-based assertions** between Identity Provider (IdP) & Service Provider (SP)

## ⚙ SAML Features

- Assertions include authentication, access permission & attributes
- Supports **digital signatures & encryption** (XML standards)
- Protocols follow **request–response model**
- Widely used in federated identity management

## OAuth (Open Authentication)

- Open standard for **secure API authorization**
- Allows users to share data **without sharing passwords**
- Used by **Google, Facebook, Amazon, Microsoft, Twitter**
- Provides **access tokens** issued by authorization servers

## OAuth Features

- Secure delegation of access to third-party apps
- Works across **web, mobile, desktop apps**
- Protects user credentials while enabling data sharing

## SSL & TLS

- Cryptographic protocols for **secure TCP/IP communications**
- Protects against **eavesdropping, tampering & forgery**
- Widely used in **web, email, messaging, VoIP**

## TLS Features

- Endpoint authentication & confidentiality
- Uses **certificates from trusted CAs**
- Supports **mutual authentication** (both client & server validated)
- Key exchange: **public key algorithms**
- Message authentication: **symmetric encryption & hash functions**

## 🎯 Summary

- **SAML → Authentication & SSO**
- **OAuth → Secure API authorization**
- **SSL/TLS → Encrypted communication & integrity**
- Together ensure **cloud security, privacy & trust**

# ☁ Resource Management & Security in Cloud

## 📌 Resource Management in Cloud

- Allocation of **computing, storage, networking & energy resources**
- Goal: Meet performance needs of **providers, users & applications**

## 🔄 Inter-Cloud Resource Provisioning

- Systems must support **high throughput, HA & fault tolerance**
- Infrastructure may be **physical or virtual servers**
- Ensures scalability & optimized usage

## ⚙ Resource Provisioning Schemes

- Enable **rapid discovery** of services & data
- Storage linked to: **distributed file systems, databases & storage tech**
- **Three methods:**
  - **Demand-Driven**
  - **Event-Driven**
  - **Popularity-Driven**

## 🌐 Inter-Cloud Principles

- Providers can **expand/redimension capacity** dynamically
- Achieved via **leasing resources from other cloud providers**
- Promotes **competitive & scalable service delivery**

## ⚠Security Challenges in Cloud

- Despite benefits, **security issues slow adoption**
- Key concerns:
  - **Trust & privacy**
  - **Lack of security & copyright protection**

## 🔑 Privacy Issues

- **Compliance & regulatory risks**
- **Storage & retention concerns**
- **Access control & data sharing risks**
- **Auditing & monitoring challenges**

## 🚨 Lack of Strategy Risks

- Leads to **unsupported operating models**
- Weakens **security posture**

## 🛡 Security Governance Factors

- **Risk assessment & management**
- **Security awareness & training**
- **Policy, guidelines & standards**
- **Monitoring & incident response**
- **Business continuity & disaster recovery**

## 🖥 Virtual Machine (VM) Security

- Threats: **hypervisor malware, VM hopping, hijacking, rootkits**
- **Mitigation:**
  - Network/Hardware IDS
  - Code shepherding & execution control
  - Security tech: **vSafe, vShield, Intel VPro, sandboxing**

## 🔑 Identity & Access Management (IAM)

- Framework for **creation, maintenance & termination** of digital identities
- Ensures **controlled access to shared resources**
- Combines **policy, governance & standard practices**

## 📌 Key Security Standards

- **SAML** – Single Sign-On, authentication & authorization
- **OAuth** – Delegated API access without passwords
- **SSL/TLS** – Secure, encrypted communication

## 🔒 Role of Standards

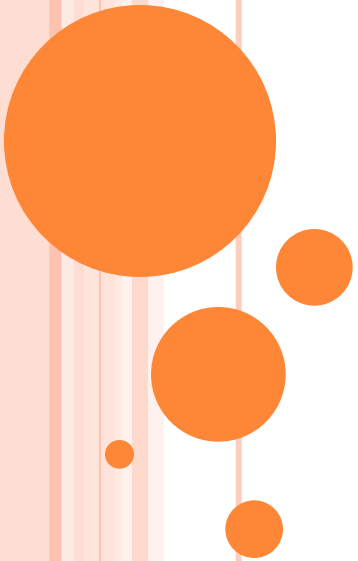
- Define **processes, measures & practices**
- Ensure secure operations in **web & network environments**
- Enable **trust, compliance & interoperability**

## 🎯 Summary

- **Resource management** → Efficient allocation & provisioning
- **Security challenges** → Privacy, trust, compliance
- **Governance & IAM** → Strengthen protection
- **Standards (SAML, OAuth, SSL/TLS)** → Foundation for cloud security

# UNIT V

# CLOUD TECHNOLOGIESAND ADVANCEMENTS



# HADOOP

## Introduction to Hadoop

- With the growth of the Internet and technology, organizations need:
  - High computing power
  - Large data storage
  - Fast data processing
- Every day, a huge amount of data is produced by companies.

**Real-World Examples**

- **Facebook:** Produces 600+ TB data per day and analyzes 30+ Petabytes.
- **Boeing Jet:** Generates 10+ TB data per flight (maps, images, etc.).
- **Walmart:** Handles 1 million+ transactions every hour (2.5+ Petabytes stored).

## Hadoop

- Apache Hadoop is an **open-source framework** by Apache.
- It allows **distributed processing** of big data across many computers.
- Scales from one server to thousands.
- Provides **fault tolerance** and **massive storage**.

## Need for Hadoop

- Traditional databases cannot handle big data efficiently.
- Hadoop helps to **store, process, and analyze** large data easily.

## Hadoop Core Components

- **HDFS (Hadoop Distributed File System):**
  - Stores data across multiple systems.
  - Inspired by Google File System (GFS).
- **MapReduce:**
  - Processing engine that analyzes data stored in HDFS.

## Hadoop Ecosystem

- Built around HDFS and MapReduce.
- Supports many tools for data storage, analysis, and management.

# Big Data

- Big Data means very large and complex data sets.
- It needs special tools to **store, process, and analyze** efficiently.

## Challenges of Big Data

- **Volume:**
  - The size of data is growing fast.
  - Hard to store and manage huge data.
- **Variety:**
  - Data comes in different formats (text, video, images).
  - Unstructured data is difficult to handle.
- **Velocity:**
  - Data is generated very quickly.
  - Need to process it in real time.
- **Veracity:**
  - Data may be incomplete or inaccurate.
  - Leads to poor data quality.

## Hadoop Ecosystem Components

- Hadoop is not only made up of **HDFS** and **MapReduce**, but it also includes many other tools and frameworks.
- These tools together form the **Hadoop Ecosystem**.
- The ecosystem helps in **storing**, **processing**, and **analyzing** Big Data efficiently.
- Each component has a specific role to handle data in different ways.

Some popular Hadoop ecosystem components are:

- **HDFS** – Storage system
- **MapReduce** – Processing engine
- **YARN** – Resource management
- **Pig, Hive, HBase, Sqoop, Flume, Oozie, Zookeeper** – Data tools for analysis, storage, and workflow management

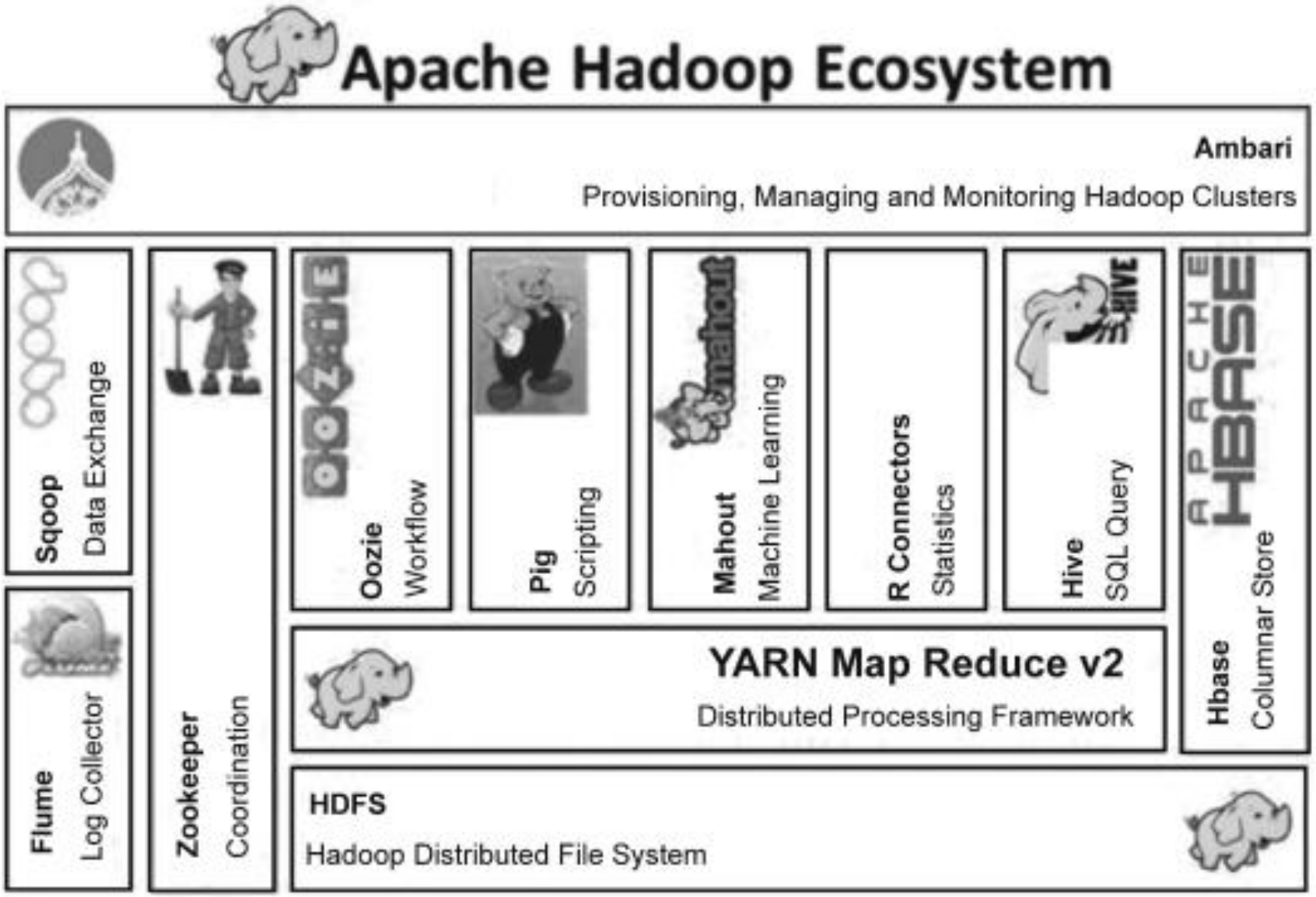



**Fig. 5.1.1 : Hadoop ecosystem components**

# Components of Hadoop ecosystem

| Sr. No. | Name of Component | Description |
|---|---|---|
| 1) | HDFS | It is a Hadoop distributed file system which is used to split the data in to blocks and stored amongst distributed servers for processing. It runs multiple clusters to store several copies of data blocks those can be used in case failure occurs. |
| 2) | MapReduce | It's a programming model to process the big data. It comprising of two programs written in Java such as mapper and reducer. The mapper extracts data from HDFS and put in to maps while reducer aggregate the results generated by mappers. |
| 3) | Zookeeper | It is a centralized service used for maintaining configuration information with distributed synchronization and coordination. |
| 4) | HBase | It is a Column-oriented database service used as NoSQL solution for big data |
| 5) | Pig | It is a platform used for analyzing the large data sets using a high-level language. It uses dataflow language and provides parallel execution framework. |
| 6) | Hive | It provides data warehouse infrastructure for big data |
| 7) | Flume | It provides distributed and reliable service for efficiently collecting, aggregating, and moving large amounts of log data. |
| 8) | Scoop | It is a tool designed for efficiently transferring bulk data between Hadoop and structured data stores such as relational databases. |
| 9) | Mahaout | It provides libraries for scalable machine learning algorithms implemented on the top of Hadoop implemented using MapReduce framework. |
| 10) | Oozie | It is a workflow scheduler system to manage the Hadoop jobs. |
| 11) | Ambari | It provides a software framework for provisioning, managing and monitoring Hadoop clusters. |

**Table 5.1.1 : Different components of Hadoop ecosystem**

| Sr. No. | Name of Component | Description |
|---|---|---|
| 1) | HDFS | It is a Hadoop distributed file system which is used to split the data in to blocks and stored amongst distributed servers for processing. It runs multiple clusters to store several copies of data blocks those can be used in case failure occurs. |
| 2) | MapReduce | It's a programming model to process the big data. It comprising of two programs written in Java such as mapper and reducer. The mapper extracts data from HDFS and put in to maps while reducer aggregate the results generated by mappers. |
| 3) | Zookeeper | It is a centralized service used for maintaining configuration information with distributed synchronization and coordination. |
| 4) | HBase | It is a Column-oriented database service used as NoSQL solution for big data |
| 5) | Pig | It is a platform used for analyzing the large data sets using a high-level language. It uses dataflow language and provides parallel execution framework. |
| 6) | Hive | It provides data warehouse infrastructure for big data |
| 7) | Flume | It provides distributed and reliable service for efficiently collecting, aggregating, and moving large amounts of log data. |
| 8) | Scoop | It is a tool designed for efficiently transferring bulk data between Hadoop and structured data stores such as relational databases. |
| 9) | Mahaout | It provides libraries for scalable machine learning algorithms implemented on the top of Hadoop implemented using MapReduce framework. |
| 10) | Oozie | It is a workflow scheduler system to manage the Hadoop jobs. |
| 11) | Ambari | It provides a software framework for provisioning, managing and monitoring Hadoop clusters. |

**Table 5.1.1 : Different components of Hadoop ecosystem**

## Introduction to HDFS

- **HDFS (Hadoop Distributed File System)** is the main storage system of Hadoop.
- It is designed to **store very large files** and provide **fast access** to many users at once.
- HDFS runs on **low-cost (commodity) hardware** and is **highly fault tolerant**.
- Data is stored across many computers with **multiple copies (replicas)** to prevent data loss.

## Structure of HDFS

- HDFS has two main parts:
  - **NameNode** – The master node that manages file structure and metadata.
  - **DataNodes** – The worker nodes that actually store the data blocks.
- A single cluster has one active NameNode connected to many DataNodes.

## Working of HDFS

- Large files are divided into **fixed-size blocks** and stored across DataNodes.
- The NameNode keeps information about:
  - File type, size, creation date
  - Properties and block location of each file
- DataNodes send regular **"heartbeats"** to the NameNode to confirm they are working.
- If a DataNode fails, HDFS **automatically recovers data** using replicated copies.

## Fault Tolerance and Replication

- Every data block is **replicated** across multiple DataNodes.
- The **replication factor** (number of copies) can be set by the user.
- This ensures data safety even if one or more nodes fail.
- HDFS also supports **metadata replication** and **snapshots** for rollback during failures.

## Key Features of HDFS

- Provides **streaming access** to large files.
- Supports **read, write, delete** operations (but not update).
- Optimized for **high-speed read operations**.
- Offers **Java API** and **command-line interface** for access.
- Maintains **file permissions and authentication**.
- Performs **data rebalancing** to equalize storage load.
- Uses **checksums and digital signatures** for data integrity.

# HDFS Architecture

- The **Hadoop Distributed File System (HDFS)** follows a **Master–Slave architecture**.
- It includes:
  - **One Master Node** → NameNode
  - **Multiple Slave Nodes** → DataNodes
- Files are divided into **fixed-size blocks** and stored across the cluster.
- This design ensures **fault tolerance, scalability, and fast data access**.

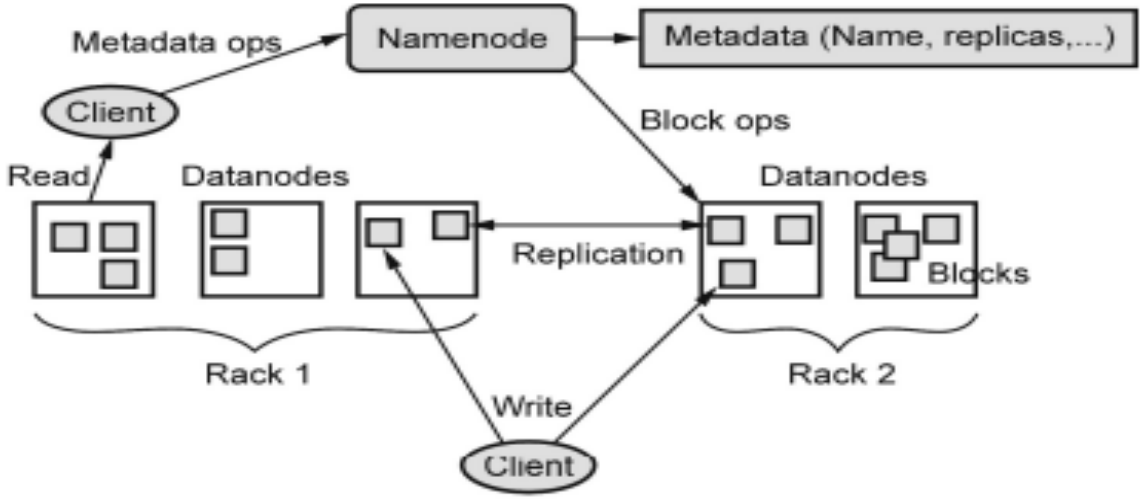



Fig. 5.2.1 : HDFS architecture

# NameNode (Master Node)

- The **NameNode** acts as the **master server** in HDFS.
- It manages the **file system namespace** and **controls client access**.
- It stores **metadata** such as:
  - File names, types, sizes, and locations
- Metadata is kept in memory for **faster access**.
- Two key files are maintained:
  - **FsImage:** Contains the full directory structure.
  - **EditLog:** Records all recent file changes.

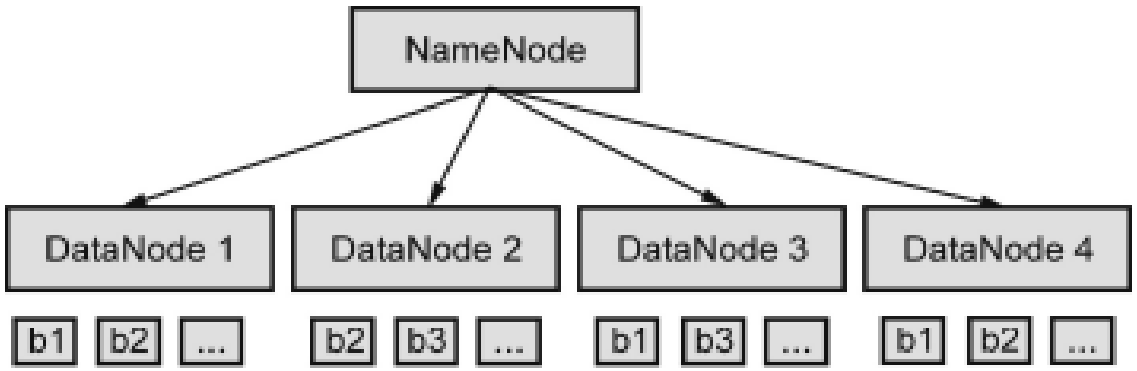



**Fig. 5.2.2 : Representation of name node and data nodes**

## DataNodes (Slave Nodes)

- **DataNodes** store the **actual data blocks** of files.
- They handle client **read/write requests**.
- Perform tasks like **block creation, deletion, and replication** as directed by the NameNode.
- Each block is stored as a separate file.
- Multiple copies of each block are stored on different DataNodes for **data safety**.

## HDFS Client

- Users and applications access HDFS through the **HDFS Client**.
- Clients can perform operations like **read, write, delete**, and **create directories**.
- When reading, the client asks the **NameNode** for block locations and connects directly to the **DataNodes**.
- When writing, the NameNode selects DataNodes for storing **replica blocks**.
- Clients send data in a **pipeline** from one node to another.

## HDFS Blocks and Job Tracker

- Files are stored as **blocks** (default size: **64 MB**, adjustable).
- If a DataNode fails, HDFS automatically **re-replicates** lost blocks.
- Communication between NameNode and DataNodes happens through **heartbeats**.
- **Job Tracker** (master) and **Task Trackers** (slaves) manage MapReduce jobs.
- Job Tracker assigns tasks to nodes where data already exists for **efficiency and fault tolerance**.

# MapReduce

- **MapReduce** is a **programming model** in Hadoop used to process **large amounts of data**.
- It allows **distributed computation** across many computers or clusters.
- Helps in **scaling data processing** easily on multiple nodes.
- Every MapReduce program has two parts:
  - **Mapper**
  - **Reducer**

## Working of MapReduce

- The **Mapper** takes input as key-value pairs and converts them into **intermediate key-value pairs**.
- The **Reducer** takes these intermediate pairs and combines them into **final output key-value pairs**.
- The process divides a large problem into **small independent sub-tasks**.
- Each sub-task runs **in parallel** on different machines.
- Finally, all results are combined to produce the complete output.

## Features of MapReduce

- **Synchronization:**
  - Supports running many tasks at once.
  - Maintains synchronization using shared variables and state information.
- **Data Locality:**
  - Although data is stored on many clusters, it appears **local to the application**.
  - Placing code and data on the same machine improves performance.
- **Error Handling:**
  - If a node fails, MapReduce automatically **detects and reschedules** tasks on another node.
- **Scheduling:**
  - The engine **divides and assigns tasks** efficiently to nodes for faster execution.

## MapReduce Framework

- MapReduce is a programming model in Hadoop.
- It processes large data sets using parallel computing.
- The main unit of work in MapReduce is called a **job**.

## Phases of MapReduce

- Every MapReduce job has two main parts:
  - **Map Phase**
  - **Reduce Phase**
- Input and output are always in the form of **<key, value> pairs**.

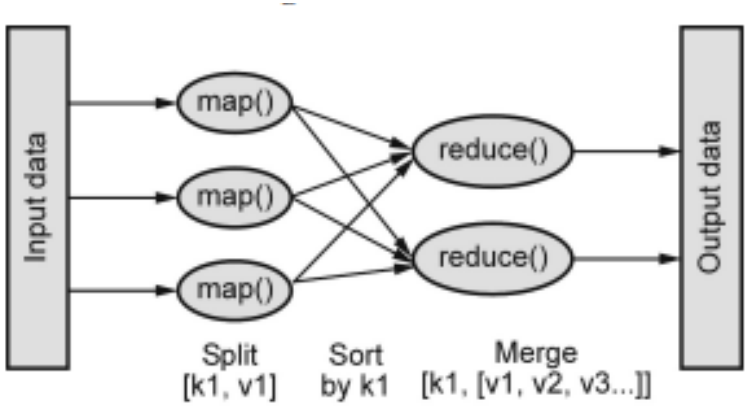



Fig. 5.3.1 : MapReduce operations

Every MapReduce program undergoes different phases of execution. Each phase has its own significance in MapReduce framework

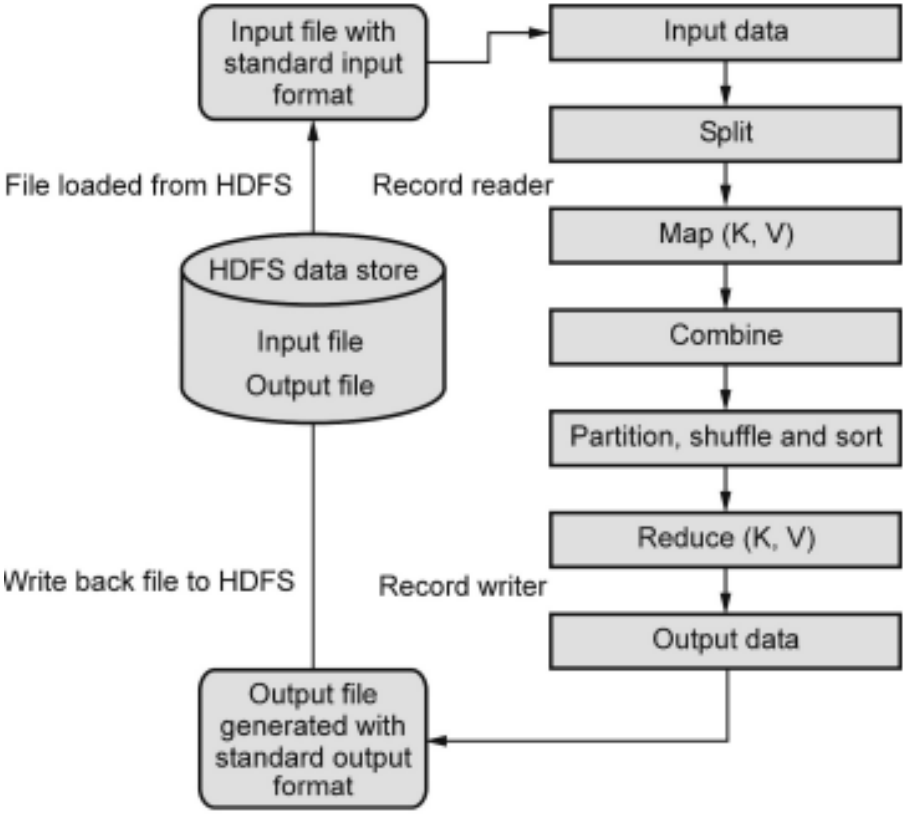



**Fig. 5.3.2 : Different phases of execution in MapReduce**

## Input Phase

- Large input data is stored in **HDFS (Hadoop Distributed File System)**.
- Input data is divided into **splits** for processing.

## Map Phase

- Each split is given to a **Mapper**.
- Mapper reads data, processes it, and produces **intermediate <key, value> pairs**.

## Combiner Phase

- The **Combiner** reduces data size before sending it to the reducer.
- It is also called a **semi-reducer**.

## Shuffle and Sort Phase

- **Shuffle:** Moves mapper output to the correct reducer.
- **Sort:** Arranges keys before reduction.
- Both happen simultaneously.

## Reduce and Output Phase

- **Reducer** processes sorted data and produces final <key, value> pairs.
- The **final output** is written to **HDFS** as output files.

# VirtualBox

- VirtualBox is a **virtualization software** that lets users run multiple operating systems on one computer.
- It was created by **Innotek GmbH**, later owned by **Sun Microsystems**, and now by **Oracle Corporation**.
- VirtualBox is a **cross-platform** software.
- It works on **Windows, Linux, Mac OS, and Solaris**.
- Useful for **testing, developing, and deploying** software on different operating systems.

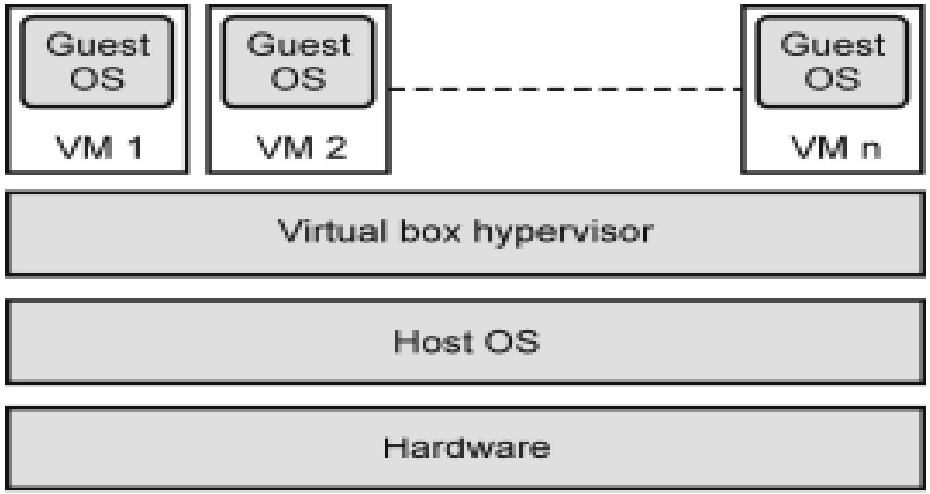



**Fig. 5.4.1 : Functional architecture of Virtual Box Hypervisor**

## Type of Hypervisor

- VirtualBox is a **Type II (hosted) hypervisor**.
- It runs as an application on an existing operating system.
- Inside it, users can install and run other systems called **Guest OS**.

## Virtual Machines

- Each guest OS runs inside a **Virtual Machine (VM)**.
- Every VM has its own **virtual hardware** (CPU, memory, disk, network).
- The guest OS works as if it's on a real computer.

## Performance and Flexibility

- Lightweight and powerful virtualization engine.
- Supports running software smoothly inside VMs.

## Cloud and Compatibility

- The latest version supports **cloud and container development**.
- It can use **VMware (.vmdk)** and **Virtual PC (.vhd)** disk images.

# Google App Engine

- **Google App Engine (GAE)** is a **Platform-as-a-Service (PaaS)** cloud model.
- It provides a **scalable runtime environment** to build and run **web applications**.
- Managed completely by **Google Cloud**.

## Purpose of GAE

- Helps developers **create and deploy** web apps easily.
- No need to manage servers or infrastructure manually.
- GAE automatically handles **scaling, load balancing, and updates**.

## Programming Language Support

- Supports multiple languages such as:
  **Java, Python, .NET, PHP, Ruby, Node.js, and Go.**
- Developers can use **third-party libraries and frameworks**.

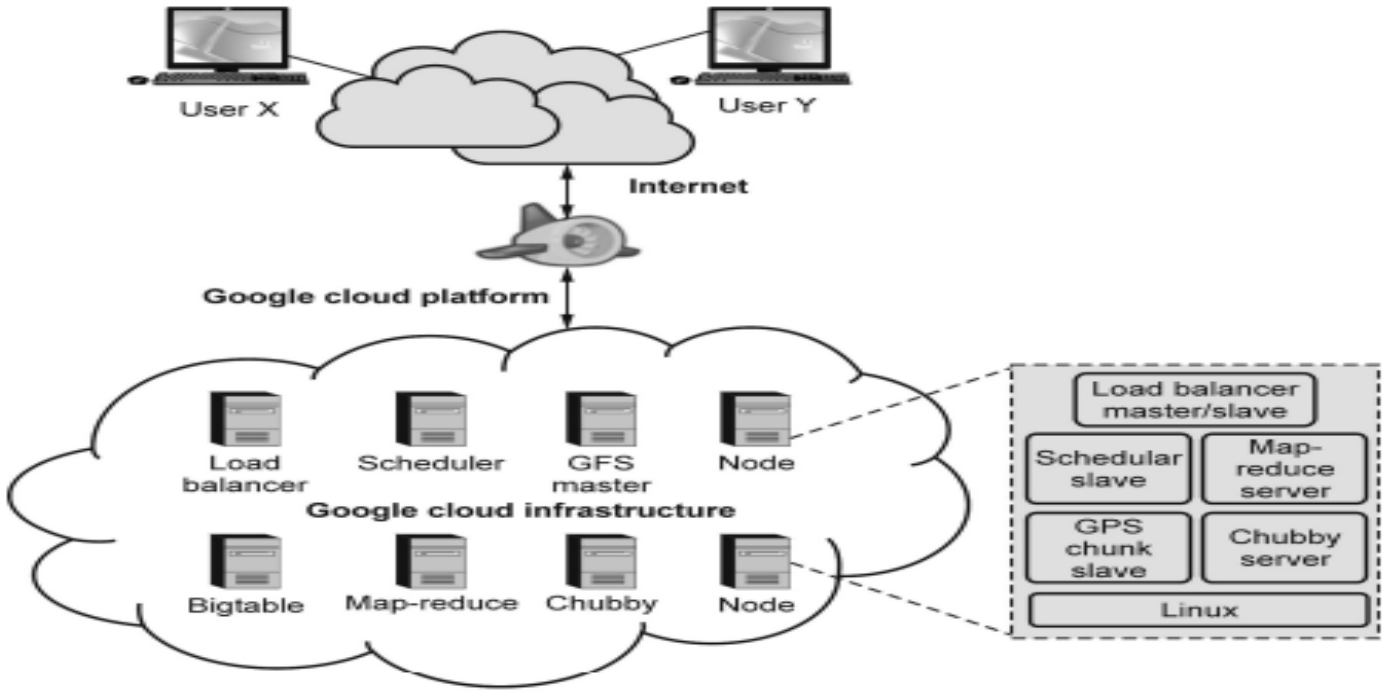



**Fig. 5.5.1 : Functional architecture of the Google cloud platform for app engine**

The GAE platform comprises five main components like

## Role of SDK (Software Development Kit)

- **SDK** is used to set up, develop, deploy, and manage apps.
- It provides local development tools and integration support for cloud deployment.

## Serverless and Scalable Platform

- GAE is **serverless**, meaning Google manages servers automatically.
- It scales applications up or down based on user demand.

## Google's Infrastructure

- GAE runs on **Google's global data centers**.
- Each data center contains thousands of servers organized in clusters.
- These servers host all Google cloud applications and services.

## Core Components of GAE Infrastructure

- **Google File System (GFS):** Stores large data sets.
- **MapReduce:** Handles large-scale data processing.
- **BigTable:** Provides structured and unstructured data storage.
- **Chubby:** Offers distributed locking and coordination services.

## Main Components of GAE Platform

- **Application Runtime Environment** – for scalable web execution.
- **SDK** – for local development and deployment.
- **Datastore** – object-oriented data storage (based on BigTable).
- **Admin Console** – for easy app and resource management.
- **GAE Web Service** – provides APIs and interfaces.

# Google App Engine (GAE)

## Introduction

- Google App Engine (GAE) supports cloud applications through systems like:
  - **Google File System (GFS)**
  - **BigTable**
  - **Chubby**
- These systems ensure **high performance**, **reliability**, and **scalability**.

# GFS

- **Google File System (GFS)** is a **distributed file system** designed by Google.
- It is used for storing and processing **huge amounts of data** efficiently.
- GFS supports Google's large-scale data applications.

## Design Goals

- Ensure **availability**, **performance**, and **fault tolerance**.
- Handle **component failures automatically**.
- Optimize for **large files** and **high data throughput**.

## Key Design Assumptions

- Automatic recovery from failures.
- Efficient handling of large and small files.
- Support for **small random reads** and **large sequential writes**.
- Maintain **atomicity** and **synchronization** with minimal overhead.

# GFS Architecture

- A GFS cluster has:
  - **One GFS Master**
  - **Several Chunk Servers**
  - **Multiple Clients**
- Clients and servers run as **user-level processes** on Linux machines.

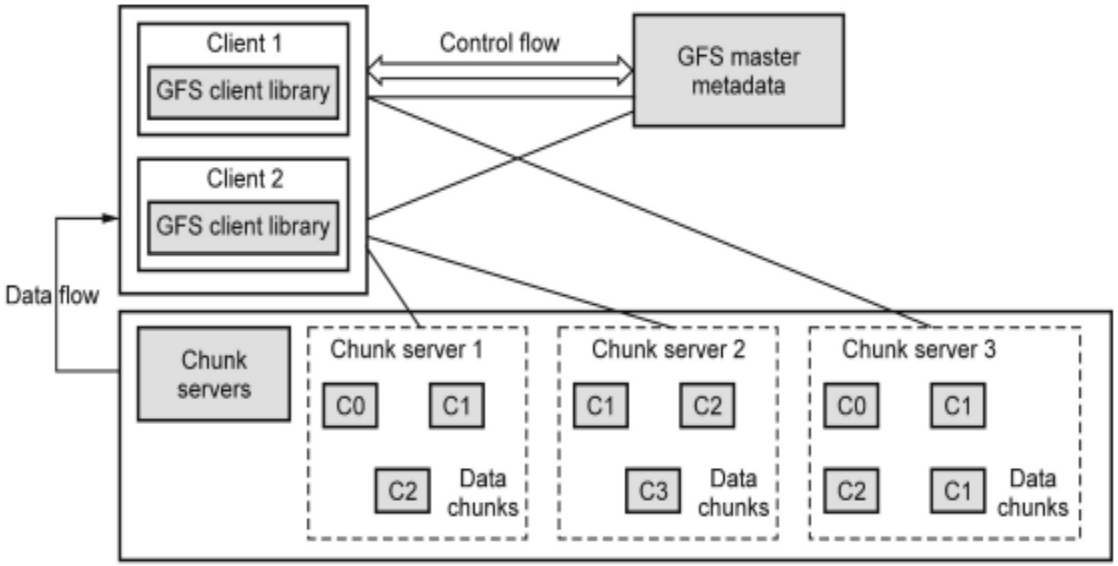



Fig. 5.6.1 : Architecture of GFS clusters

## Working of GFS

- Files are divided into **chunks (64 MB each)**.
- Each chunk has a unique **64-bit handle**.
- Chunks are **replicated (usually 3 copies)** across different machines for safety.

## Communication Flow

- The **GFS Master** manages **metadata** and coordinates operations.
- Clients interact directly with **Chunk Servers** for data transfer.
- Data is read/written using **GFS APIs** (create, read, write, delete).

## Special Features

- **Snapshots** – create file copies quickly.
- **Atomic Appends** – allow multiple clients to add data safely.
- No need for **POSIX API** or **Linux vnode hooks**.

## Main Features of GFS

- Large-scale data processing and fault tolerance.
- Automatic recovery and checksum-based data protection.
- High throughput for concurrent operations.
- Centralized but efficient master design.
- Includes **caching** and **logging** for performance and debugging.

## BigTable

- **BigTable** is Google's **distributed storage system**.
- It stores **huge amounts of structured and unstructured data**.
- Designed for **speed, reliability, scalability, and efficiency**.

## Purpose of BigTable

- Handles **billions of web pages**, **satellite images**, and **user data**.
- Used for storing and managing **semi-structured data** efficiently.
- Supports **high-speed concurrent processing**.

## Design Requirements

- High speed
- Reliability
- Scalability
- Efficiency
- High performance
- Track data changes over time

# Architecture of BigTable

- Consists of **Client**, **BigTable Master**, and **Tablet Servers**.
- Runs on clusters similar to **Google File System (GFS)**.
- Each **tablet** is 100–200 MB in size.

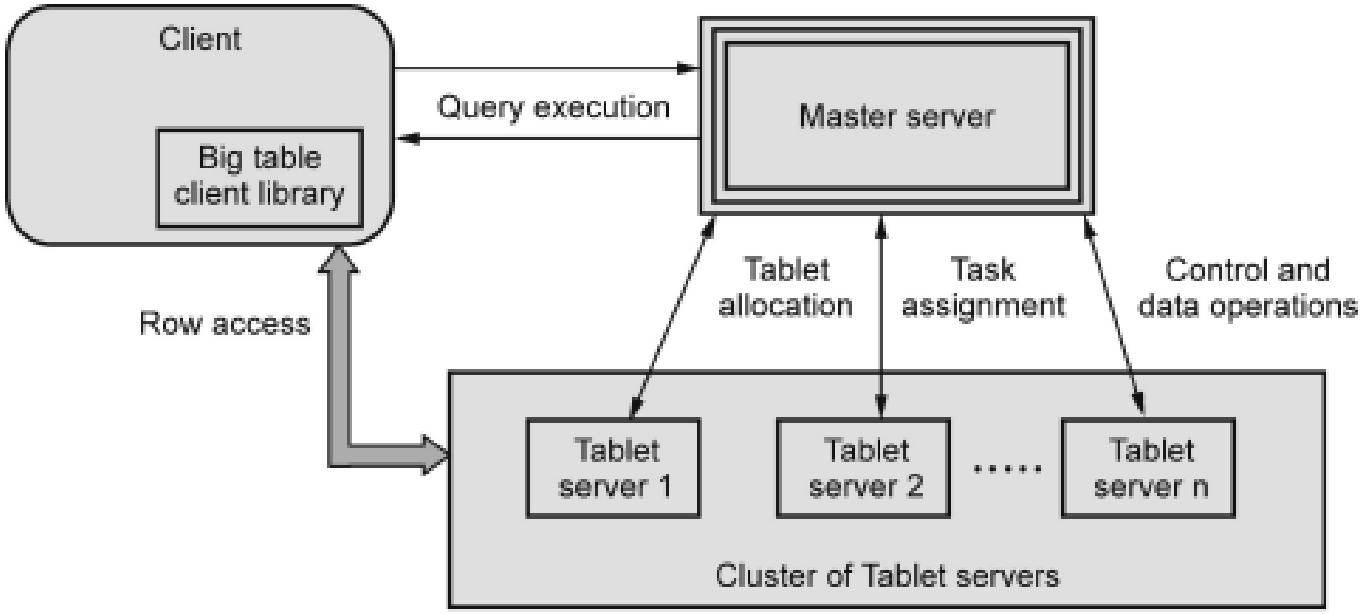



**Fig. 5.6.2 : Generalized architecture of Big table**

## Role of Master and Tablet Servers

- **Master Server:** Assigns tablets, manages load, and monitors servers.
- **Tablet Servers:** Execute read/write tasks and provide row access to clients.

## Data Model

- Organized as a **sorted map** with three dimensions: **Row**, **Column**, and **Timestamp**.
- Rows have **row keys (URLs)** and store entity-specific data.

## Column Structure

- Columns are grouped into **Column Families**.
- Columns use the format **family:qualifier**.
- Families define **data type** and **access control**.

## Timestamp Function

- Stores **multiple versions** of data in a cell.
- Tracks **latest or historical data** using timestamps.

## BigTable API Operations

- Create/Delete tables and column families.
- **Set()**, **DeleteCells()**, and **DeleteRow()** for atomic write operations.
- Provides fast, scalable, and reliable data management.

# Chubby

- **Chubby** is a **Google service** used for **storage and coordination** in systems like **GFS** and **BigTable**.
- It is a **distributed locking service** for synchronizing many servers.

## Purpose of Chubby

- Helps in **naming, synchronization**, and **coordinator election**.
- Provides **reliable storage** and ensures **consistent availability**.

## Design Features

- Works in **large-scale, high-speed networks**.
- Synchronizes **distributed clients** and ensures **agreement** between them.
- Main goal: **High reliability and availability**.

# Chubby Architecture

- Has two parts: **Server** and **Client Library**.
- Communicate through **Remote Procedure Calls (RPCs)**.
- A **Chubby cell** has **five replica servers**.

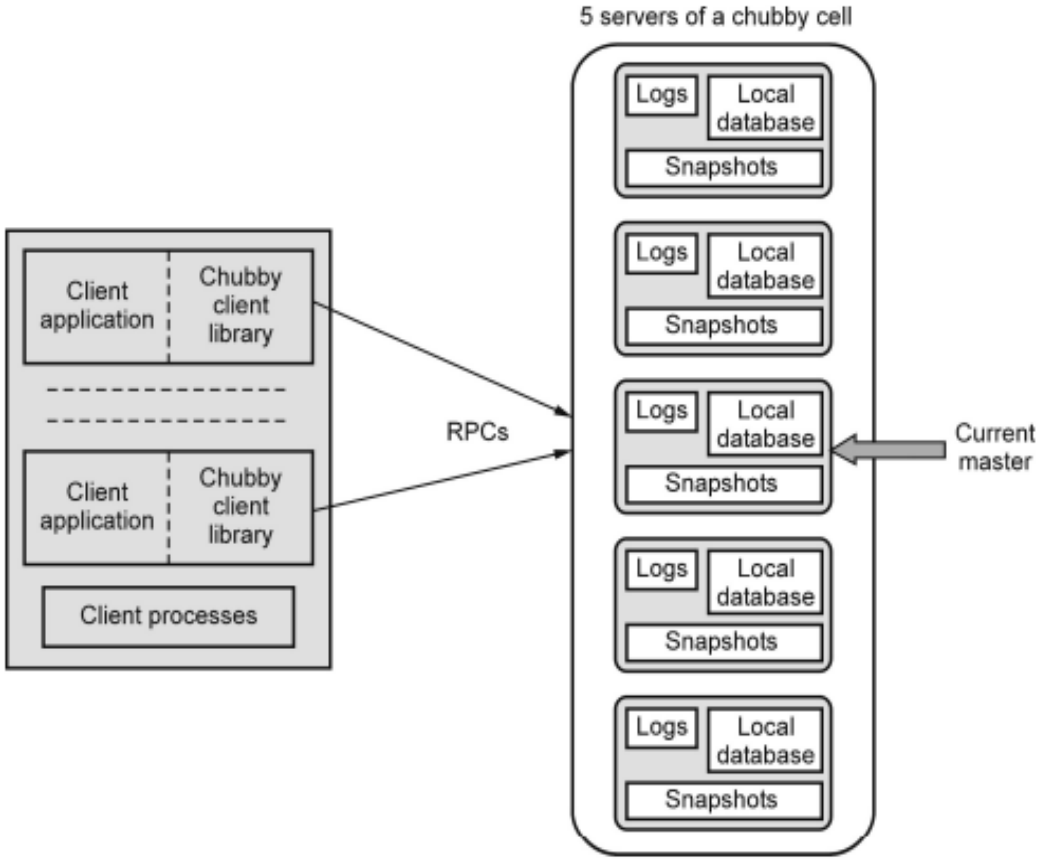



Fig. 5.6.4 : Structure of a Chubby system

## Master Election

- One **Master** is elected among replicas using a **consensus protocol**.
- Uses **Master lease** time to prevent multiple Masters.

## File System

- Works like a **simple Unix file system**.
- Files and directories are called **nodes**.
- Nodes store **metadata** and use **handles** for access.

## Locking System

- Two types: **Reader locks (shared)** and **Writer locks (exclusive)**.
- Locks are **advisory**; conflicts occur on repeated requests.
- Uses **sequencers** to describe lock states.

## Events and Caching

- Clients get notified on events like file change, lock request, or Master failure.
- Clients **cache** data to reduce load and improve speed.

# Sessions and APIs

- Sessions kept alive using **keep-alive messages**.
- Handles released if session fails.
- Main APIs: **open()**, **close()**, **Acquire()**, **Release()**, **GetStat()**, **Delete()**, etc.

| API | Description |
|---|---|
| *Open* | Opens the file or directory and returns a handle |
| *Close* | Closes the file or directory and returns the associated handle |
| *Delete* | Deletes the file or directory |
| *ReadDir* | Returns the contents of a directory |
| *SetContents* | Writes the contents of a file |
| *GetStat* | Returns the metadata |
| *GetContentsAndStat* | Writes the file contents and return metadata associated with the file |
| *Acquire* | Acquires a lock on a file |
| *Release* | Releases a lock on a file |

Table 5.6.1 : APIs in Chubby

# Google APIs

- **Google APIs** are **Application Programming Interfaces** developed by Google.
- They allow communication between **applications and Google services**.
- Help developers integrate **Google's features** into other systems.

## Purpose of Google APIs

- Enable **easy access** to Google services like Maps, Drive, and YouTube.
- Support integration with **external applications** and platforms.
- Improve **efficiency, connectivity, and automation**.

## Benefits of Google APIs

- Saves **development time**.
- Provides **reliable and secure** data access.
- Supports **cross-platform** compatibility.

## Google App Engine

- A **cloud platform** to deploy and manage APIs and web apps.
- Developers don't need to manage **infrastructure** manually.
- Scales automatically based on traffic.

## Google Cloud Endpoints

- A set of **tools and libraries** for building APIs.
- Works with **Google App Engine** to create and host APIs.
- Helps generate **client libraries** automatically.

## API Hosting

- Endpoints are **hosted on Google App Engine**.
- Manages **requests, responses, and authentication**.

## Advantages for Developers

- Simplifies **data sharing** between apps.
- Avoids writing complex **network communication code**.
- Provides **ready-to-use libraries** for multiple platforms.

## Summary

- Google APIs connect apps with Google services.
- **Google App Engine + Cloud Endpoints** make API creation easier.
- Ensures **faster, scalable, and secure integration** for developers.

## OpenStack

- **OpenStack** is an **open-source cloud operating system**.
- It manages **computing, storage, and networking** resources in data centers.
- Used to build **private, public, or hybrid clouds**.
- OpenStack provides **Infrastructure-as-a-Service (IaaS)**.
- It helps in **creating, managing, and deploying virtual machines (VMs)**.
- Free and **highly scalable** platform.

## Main Objectives

- Global and open-source
- Freely available and easy to use
- Highly scalable and interoperable
- Suitable for both **private and public clouds**

- Components of OpenStack+

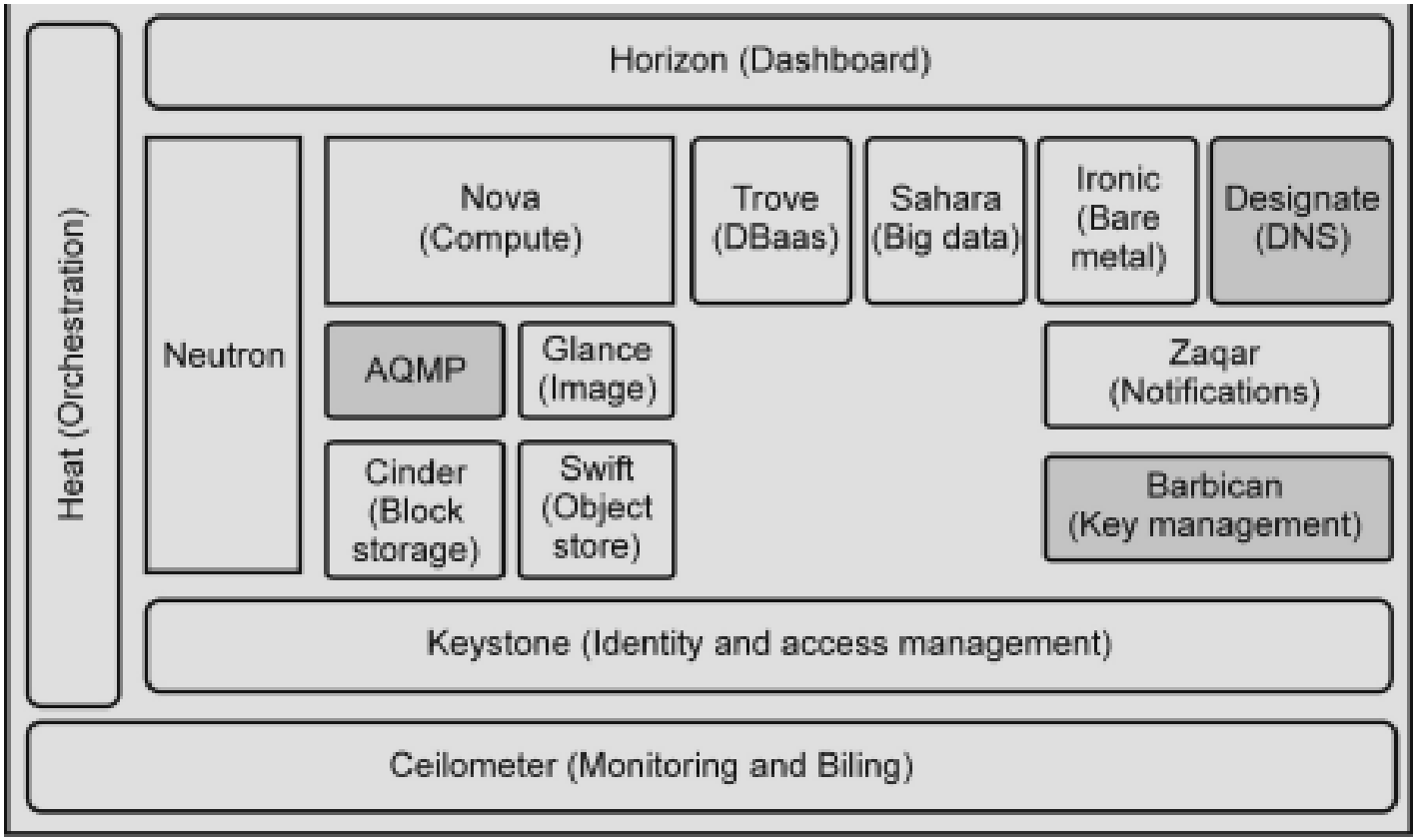



**Fig. 5.7.1 : Components of open stack architecture**

**Components**

- **Nova:** Compute service to manage VMs.
- **Swift:** Object storage service (like Amazon S3).
- **Cinder:** Block storage (like AWS EBS).
- **Glance:** Image service for VM templates.
- **Neutron:** Provides networking and load balancing.
- **Heat:** Orchestration tool for managing applications.
- **Keystone:** Identity and access management.
- **Horizon:** Web dashboard for management.
- **Ceilometer:** Metering and billing service.
- **Trove:** Database-as-a-Service.
- **Sahara:** Big data processing (Hadoop).
- **Zaqar:** Messaging service.
- **Ironic:** Bare-metal provisioning.

## 🔧 Security Responsibility

- Subscriber (not CSP) responsible for **patching VMs**.
- Perimeter security: firewalls, IDS/IPS, segmentation, DMZs.
- CSP must ensure **privacy & protection of customer data**.

## 🔒 Privacy Issues in Cloud

- **Compliance**: must follow country-specific laws (HIPAA, GLBA, FISMA, Patriot Act).
- **Storage**: multi-copies across datacenters, user unaware of location.
- **Retention**: CSP policies define how long data is kept.
- **Access**: users have right to view/delete their data.
- **Auditing**: organizations must verify CSP log & monitoring practices.
- **Destruction**: uncertainty if data is truly deleted at end-of-life.
- **Breaches**: lack of CSP transparency may hide security incidents.

# SaaS Security in Cloud Computing

## 📌 SaaS in Cloud Future

- SaaS will remain **dominant cloud service model**.
- Supported by Web 2.0 collaboration & XaaS models.
- Creates **new business opportunities** but also **security challenges**.

## 🔑 Importance of SaaS Security

- Vendors act like **Managed Service Providers (MSPs)**.
- Businesses must review **data protection policies** before adoption.
- Security practices & monitoring = **critical requirements**.

# Cloud Federation

- **Cloud Federation** connects multiple cloud providers using common standards.
- It helps in **managing decentralized services** and ensures **consistent connectivity**.
- Allows **data movement**, **workload distribution**, and **security management** across clouds.

## Key Features of Federated Cloud

- **Resource redundancy**
- **Resource migration**
- **Combination of complementary resources**
- Enables enterprises to **use multiple cloud services as one**.

## Jabber XCP

- Developed by **Cisco** and **Sun Microsystems**.
- A **programmable messaging and presence platform**.
- Supports multiple protocols like **SIMPLE** and **IMPS**.
- Ideal for **real-time communication** and **scalable cloud apps**.

## XMPP **(Extensible Messaging and Presence Protocol)**

- An **open standard** for instant messaging.
- Uses **XML-based communication** for flexibility.
- Each user is identified by a **Jabber ID (JID)**.

# Four Levels of Federation

## Introduction

- **Federation in XMPP** defines how servers in different domains exchange XML-based messages.
- As per **XEP-0238**, there are **four levels of federation** — from least secure to most secure.

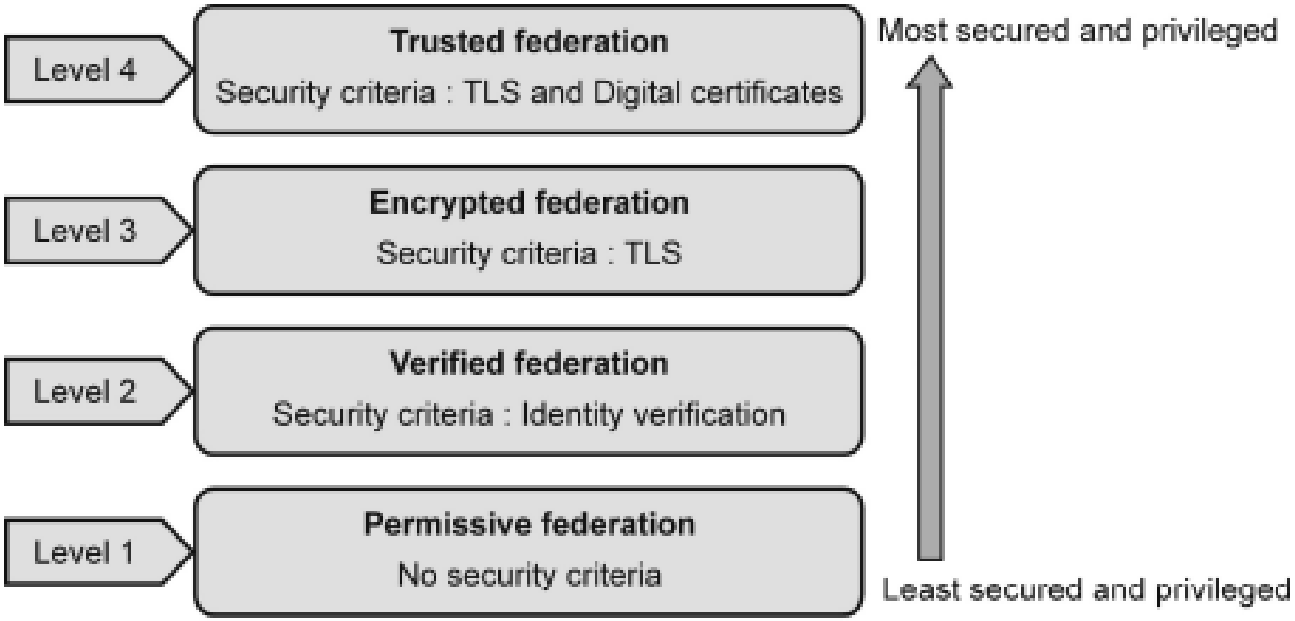



**Fig. 5.9.1 : Four levels of federation**

**Level 1 – Permissive Federation**

- Lowest level of federation.
- Servers **accept connections without verifying identity** (no DNS or certificates).
- **No authentication → prone to domain spoofing and spam.**
- Used in early web applications but now obsolete with Jabberd 1.2.

**Level 2 – Verified Federation**

- Servers **verify the identity** of peers before connecting.
- Uses **DNS-based key exchange** for verification.
- **Not encrypted**, but prevents domain spoofing.
- Requires proper **DNS setup**, though still vulnerable to DNS poisoning.
- Default for open XMPP after Jabberd 1.2.

**Level 3 – Encrypted Federation**

- Builds on verified federation.
- Requires **Transport Layer Security (TLS)** for encrypted communication.
- Uses **Server Dialback Protocol (XEP-0220)** for weak identity verification.
- Prevents most address spoofing; certificates are **self-signed**.

**Level 4 – Trusted Federation**

- **Most secure** level.
- Requires **TLS + CA-issued digital certificates** from trusted authorities.
- Ensures **strong authentication** and **secure communication channel**.
- Provides **high trust and reliability** between federated domains.

**Summary**

- **Level 1 → No security**
- **Level 2 → Verified identity**
- **Level 3 → Encrypted channel**
- **Level 4 → Trusted, authenticated federation**
- As levels increase, **security and trust improve** significantly.