Q. Explain distributed system and list the advantages and disadvantages.

A **distributed system** is a network of independent computers that work together and appear as a single system to users. It involves both hardware (the physical machines) and software (the programs running on those machines). The key idea is that users don't need to know about the differences between the computers or how they communicate.

Characteristics of Distributed Systems:

1. **Resource Sharing**: Computers in the system can share data and resources, like printers, with each other.
2. **Transparency**: Users don't need to know how the computers talk to each other.
3. **Openness**: Different applications can communicate in a consistent and standard way.
4. **Scalability**: It can handle more users or resources without performance dropping.
5. **Heterogeneity**: The computers in the system can be from different brands and models.
6. **Security**: Protecting the system from cyber-attacks is important.
7. **Fault Tolerance**: The system can keep working even if some computers fail or network issues happen.
8. **Concurrency**: Multiple users can access shared resources at the same time.
9. **Quality of Service**: Ensuring things like reliability, speed, and availability.

Advantages of Distributed Systems:

1. **Resource Sharing**: Computers can easily share data with each other.
2. **Scalability**: New computers can be added easily as the system grows.
3. **Fault Tolerance**: If one computer fails, the rest can still communicate, so the system doesn't crash.
4. **Shared Resources**: Things like printers can be shared by multiple computers, not just one.

Disadvantages of Distributed Systems:

1. **Security Issues**: It's harder to secure all the computers and their connections.
2. **Message Loss**: Some data can be lost during transmission between computers.
3. **Complexity**: Managing the database across multiple computers is more complicated than on a single computer.

4. **Network Overload**: If all computers try to send data at once, the network might become slow or overloaded.

Q. Explain Architectures styles in distributed System

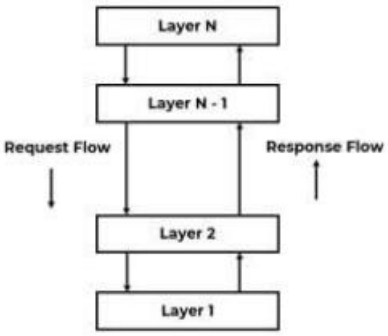
A **distributed system** is a collection of independent computers that appear as a single system to users. For example, the **World Wide Web** is a distributed system.

1. Layered Architecture:

- **Idea**: Components are arranged in layers, like a cake.
- **How it works**: Each layer can talk to the layer directly below it. The control flows from the top to the bottom, and the results go back up.
- **Example**: A web application where the user interface is at the top, business logic is in the middle, and data storage is at the bottom.

Key points:

- Requests move down (from top to bottom).
- Results come back up (from bottom to top).



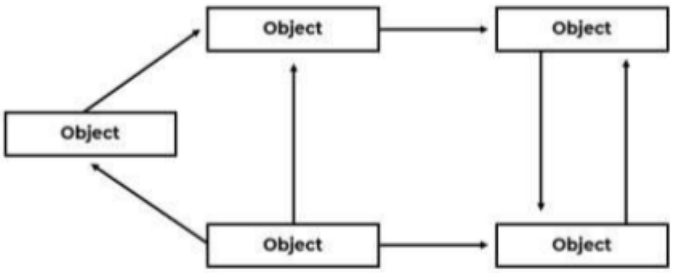
2. Object-based Architecture:

- **Idea**: Uses "Remote Procedure Calls" (RPC) to let different objects (components) talk to each other.
- **How it works**: One object can call another object remotely to request data or services, and the other object can send data back.
- **Example**: In a banking system, one object (bank account) might call another object (transaction service) to transfer money.

Key points:

- Objects communicate through Remote Procedure Calls (RPC).

- Any object can call any other object.

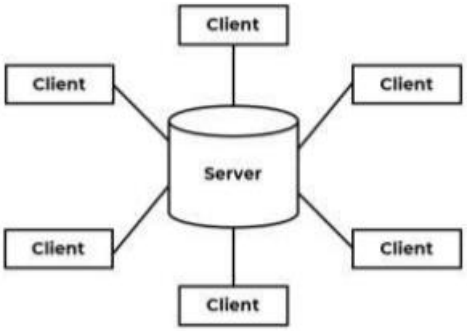


3. Data-Centered Architecture:

- **Idea**: A central server or database is at the center of the system, and clients are connected to it.
- **How it works**: The server holds data and sends it to clients whenever they need it.
- **Example**: A cloud storage service like Google Drive, where the server stores the data, and users (clients) access it.

Key points:

- Central server stores and manages data.
- Clients fetch data from the server.



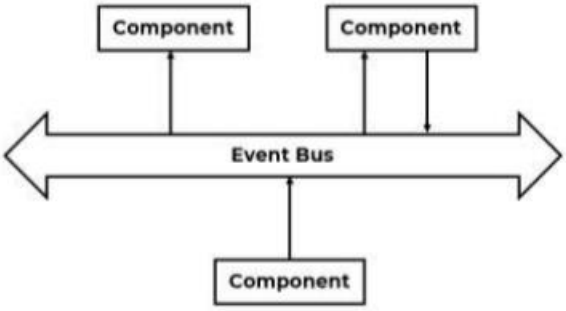
4. Event-Based Architecture:

- **Idea**: Communication happens through "events," like notifications.
- **How it works**: When something important happens (like a user action), an event is created and sent to a shared "event bus." Other parts of the system can listen for these events and take action.
- **Example**: In a shopping app, when an item is added to the cart, an event is triggered. Other services (like inventory) can listen to that event and update stock levels.

Key points:

- Communication happens through events.
- Processes are loosely connected.

- When an event happens, it can carry information that other processes can use.



Q. Explain Distributed System Model

1. Physical Model

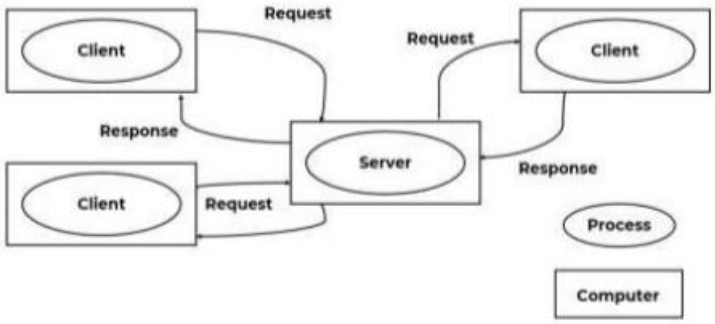
This model describes the **hardware** in a distributed system, like the computers, devices, and their connections (network). It shows how these components are physically set up.

- **Early Distributed Systems**:
    - In the late 1970s and early 1980s, small networks of 10 to 100 computers were connected by **Local Area Networks (LANs)**. These systems were very basic, and the internet wasn't widely used.
    - Example: shared printers, file servers.
- **Internet-Scale Distributed Systems**:
    - In the 1990s, as the **internet grew**, systems started connecting many computers (nodes) across the globe. These systems became larger and more diverse (more different kinds of hardware and software).
    - This includes modern systems that can scale up to millions of nodes worldwide.
- **Contemporary Distributed Systems**:
    - With **mobile computing** and **cloud computing**, computers can be anywhere and are **location-independent**. These systems are massive and work on an even larger scale.
    - Example: cloud services (Google Cloud, AWS).

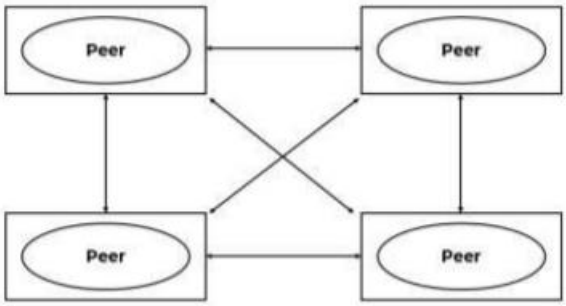
2. Architectural Model

This model explains how the different **parts of the system** interact with each other and how they are set up on the network. It helps in understanding the system's structure and responsibilities.

- **Client-Server Model**:
    - In this model, the system is divided into two main parts: **client** and **server**.
        - **Server**: This is the part that provides a specific service. For example, it could be a **file server** or a **database server**.
        - **Client**: This is the part that requests a service from the server. For example, when you open a website, your web browser (client) asks the server for the web page.
    - Communication between the client and server usually follows a simple **request/reply** pattern.
    - Sometimes, a **server** can also act as a client by asking another server for services.



- **Peer-to-Peer (P2P) Model**:
    - In this model, there is **no central control**. Every computer (node) in the system can be both a **client and a server**.
    - **Peers** (nodes) can either **request services** from others or **provide services**. So, any node can behave as a client at one time and as a server at another time.
    - This model is very flexible and doesn't depend on a single central server. Example: **file-sharing** systems like **BitTorrent**.
    - New nodes join the network and can either ask for services or provide them.



3. Fundamental Model

This model talks about **assumptions** made when building the system. It covers how processes interact with each other, how faults (errors) happen, and how security is handled.

Interaction Model

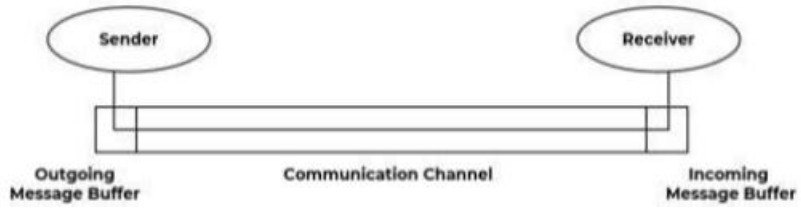
This model is all about how processes **communicate** and how time plays a role in this communication.

- **Synchronous Distributed Systems**:
    - In these systems, everything happens in a **strict order**, with clear timing rules.
        - There are limits on how long messages can take to travel between processes.
        - The system guarantees that messages are sent in order (like a first-in, first-out queue).
        - All clocks in the system are **synchronized**, meaning the time is the same across all computers.
        - Processes in the system work in **lock-step**, meaning they all perform their tasks at the same time.
    - These systems are **predictable** in terms of timing.
- **Asynchronous Distributed Systems**:
    - These systems **don't have strict timing rules**. Things can happen at any time, and there's no guarantee when something will finish.
        - There are no strict limits on message delays or how fast processes execute.
        - There's no global clock, so each computer works on its **own local time**.
        - Asynchronous systems are more **flexible** and **realistic** because they better represent how systems work in the real world.
        - However, they are **less predictable** in terms of timing.

Fault Model

This model explains the types of **failures** that can happen in the system, like when a process or a communication channel fails.

- **Omission Fault**:
    - This happens when something is **missing**. For example, a process might crash and stop working, or a message may not be sent or received.
        - **Process Omission**: If a process crashes, it stops performing its tasks.
        - **Communication Omission**: If a message is lost during transmission or is never received.



- **Arbitrary Fault** (Byzantine Fault):
    - This is when things go **wrong in unexpected ways**. Processes might give wrong answers, or messages might be corrupted.
    - Example: A process sends **incorrect data**, or a channel might **duplicate or corrupt messages**.
    - These faults are hard to detect and can cause big problems, especially when processes need to work together.
- **Timing Fault**:
    - This happens in **synchronous systems** when something doesn't follow the timing rules.
    - For example, if a message takes longer to arrive than expected, or if the system's clock doesn't stay synchronized.
    - **Real-world systems** are mostly **asynchronous**, so timing faults are less common.

Security Model

This model is about **protecting** the system from unauthorized access and ensuring data is safe.

- **Protecting Access**:
    - Only **trusted users** and **trusted systems** should be allowed to access certain parts of the system.

- **Authentication** ensures that only the right people (clients, users, or processes) can request services or access data.

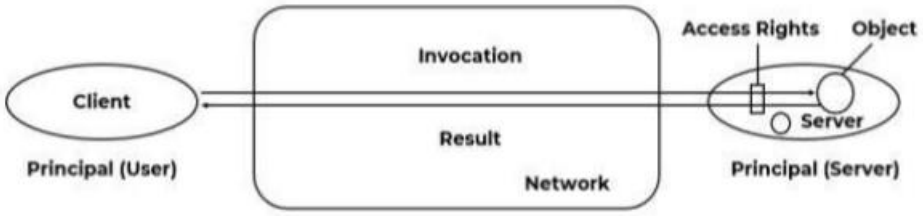


Figure 1.8: Projecting Objects.

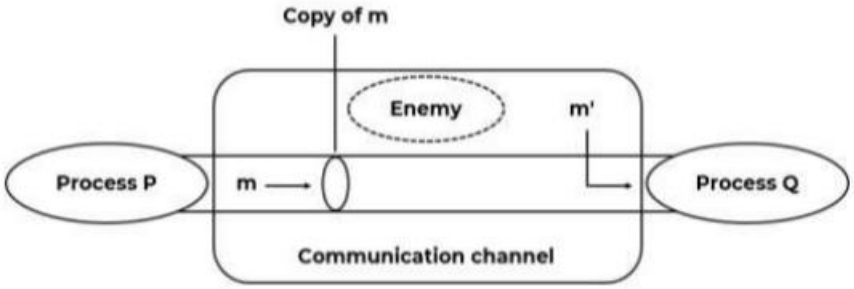


Figure 1.9: Protecting process

- **Protecting Communication**:
  - Communication between different parts of the system should be secure.
    - **Encryption** can be used to make sure that messages are private and not tampered with during transmission.
    - Each message should have a **timestamp** to prevent it from being **reordered** or **replayed** later.
- **Threats**:
  - **Man-in-the-middle attacks** can happen, where an attacker intercepts messages between two parts of the system and alters them.
  - **Secure Channels**: Using **cryptography** ensures that messages are not altered or copied by unauthorized users.

Q. Explain designing issues of distributed systems.

1. Transparency:

Transparency means the system should appear as one single unit to the users, even though it's made of multiple computers. Users shouldn't have to worry about where things are located or how they are handled. There are **8 types of transparency**:

- **Access Transparency**: Users shouldn’t know if resources are local or remote.
- **Location Transparency**: Users shouldn't know where the resource is physically located.
- **Replication Transparency**: Users should be unaware of multiple copies of resources.
- **Failure Transparency**: Users shouldn’t be affected if something fails (like a network issue).
- **Migration Transparency**: If a resource moves from one computer to another, users shouldn’t notice.
- **Concurrency Transparency**: Users should feel like they have the system all to themselves, even if others are using it too.
- **Performance Transparency**: The system should automatically improve its performance without affecting users.
- **Scaling Transparency**: The system should be able to grow (add more resources or users) without interrupting users.

2. Reliability:

A distributed system should be **more reliable** than a single computer system. If one computer fails, another should take over the task. The system should:

- Be highly **available** (always working).
- Ensure **data consistency** (no lost or corrupted data), especially when multiple copies of data exist.

3. Failure Handling:

In a distributed system, failures can happen to some parts of the system while others continue working. The system should be designed to:

- **Detect and handle failures** correctly.
- Ensure that partial failures don't affect the overall operation.

4. Flexibility:

Flexibility is the ability to **easily change** or **add new features** to the system. A flexible system should:

- Be easy to modify (adjust system components as needed).
- Be easy to enhance (add new functionalities without breaking the system).

5. Performance:

Performance is crucial in a distributed system. Even though the system may have many computers working together, it should:

- Provide performance at least as good as a **single computer** system.
- Ensure that background operations (like data management) don't slow down the user experience.

6. Scalability:

A **scalable** system can grow easily. Whether the system is small (like an intranet) or large (like the internet), it should be able to:

- Handle **more users** and **more resources** without a drop in performance.

7. Heterogeneity:

A distributed system is often made up of **different types of hardware, software, and network systems**. The system should:

- Be able to work with **different devices** (computers, servers, etc.) and **different software** (operating systems, applications).
- Handle the challenges of working across **varied environments**.

8. Security:

Security is vital in a distributed system and can be challenging. The system should protect:

- **Confidentiality**: Keep data safe from unauthorized users.
- **Integrity**: Prevent data from being tampered with.
- **Availability**: Make sure resources are always accessible when needed.

9. Concurrency:

Multiple users may try to access the same resource at the same time. The system must ensure that:

- Resources are accessed safely by multiple users without errors.
- **Read, write, and update operations** happen correctly even in a concurrent environment.

10. Openness:

An **open system** follows **standard rules** and protocols for interaction. This means:

- The system uses **open standards** for communication and services, allowing different components to work together easily.

Q. Explain goals of distributed systems.

1. Resource Sharing:

- **Goal**: Make it easy for users to access and share resources (like printers, files, or web pages) with others.
- **How it works**: A distributed system should allow users to exchange information and resources easily. But, this must be done in a secure way to avoid security issues.
- **Example**: Sharing a printer or accessing files from a remote server.

2. Openness:

- **Goal**: The system should allow easy addition of new services and components that can be used by different client programs.
- **How it works**: A distributed system should follow standard rules for communication, making it flexible and easy to add new parts without disrupting the rest of the system.
- **Example**: Adding new hardware (like a printer) or software (like a new app) to the system without breaking anything.
- **Key Point**: It should work across different **hardware**, **platforms**, and **languages**.

3. Transparency:

- **Goal**: Users should not be aware that the system is made up of multiple computers. The system should act like a single, unified system.
- **How it works**: Transparency hides the complexity of the system. There are different types of transparency:
    - **Access Transparency**: Hides differences in how data is accessed.
    - **Location Transparency**: Hides where resources are located.
    - **Migration Transparency**: Hides when a resource moves from one location to another.
    - **Relocation Transparency**: Hides when resources move while in use.
    - **Replication Transparency**: Hides when resources are copied or replicated.
    - **Concurrency Transparency**: Hides when multiple users share the same resource.
    - **Failure Transparency**: Hides failures and recovery processes.

4. Scalability:

- **Goal**: The system should be able to handle growth without problems.
- **How it works**: Scalability is measured in three ways:

1. **Size**: The system should work well even as the number of users and resources increases.
2. **Geographical**: The system should support users and resources that are spread out in different locations.
3. **Management**: The system should be able to manage resources and users from different organizations or administrative areas.

Q. Types of distributed system

1. Cluster Computing

- **How it works**: A **cluster** is a group of connected computers (often called **nodes**) that work together to perform tasks. The computers in the cluster act as a single unit, so they can share processing power and storage.
- **Key feature**: All nodes are usually in the same location and are tightly connected.
- **Example**: A **data center** where many servers work together to process large amounts of data for websites or cloud services.
- **Use Case**: Used in **high-performance computing** (HPC), big data processing, and server farms.

2. Grid Computing

- **How it works**: In **grid computing**, a collection of computers (often geographically distributed) works together to solve a large task. Unlike clusters, the computers in grid computing might not be in the same place and may have different hardware or software.
- **Key feature**: Resources are shared across a **wide area** and can be from different organizations or locations.
- **Example**: Projects like **SETI@Home**, where people contribute spare processing power from their personal computers to analyze data from space.
- **Use Case**: **Scientific research**, **simulations**, and **large-scale data analysis** across different organizations or locations.

3. Cloud Computing

- **How it works**: Cloud computing is a model where computing resources (like storage, processing power, and software) are provided over the internet by a cloud service provider. You can access these resources from anywhere.

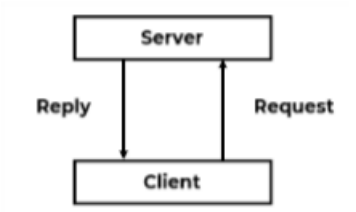
- **Key feature**: **On-demand access** to resources, where users pay for what they use (pay-per-use model).
- **Example**: Services like **Amazon Web Services (AWS)**, **Microsoft Azure**, and **Google Cloud**.
- **Use Case**: **Web hosting**, **data storage**, and **software as a service (SaaS)**.

Q. What are the different models of middleware

Middleware is software that helps different applications or systems communicate with each other. Here are the common models of middleware explained simply:

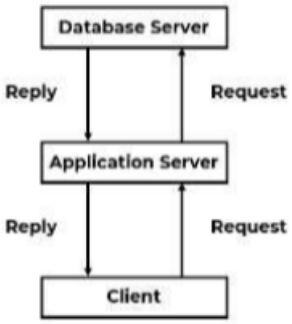
1. Client-Server Model:

- **What it is**: This is the most commonly used middleware model.
- **How it works**: One machine (called the **server**) provides services, while other machines (called **clients**) request those services.
- **Example**: When you visit a website, your browser (client) sends a request to the server, and the server responds by sending the website data.
- **Key point**: Clients make requests, and the server provides the services.



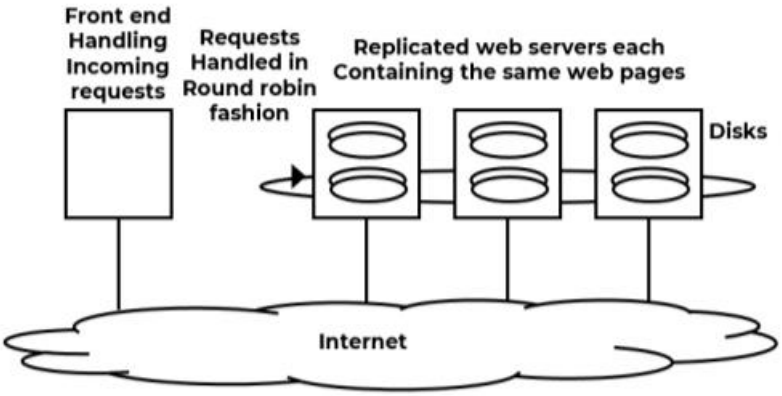
2. Vertical Distribution Model (Multi-Tier Model):

- **What it is**: This model is an extension of the client-server model.
- **How it works**: It uses multiple servers arranged in layers or tiers. The client request is sent to the first server, which processes it and passes it on to the next server, and so on, until the final server handles the request.
- **Example**: A web application where the request goes from the client to a web server, then to a database server, and finally to a data storage server.
- **Key point**: Multiple servers are involved, and the request travels from one to another until it's processed.



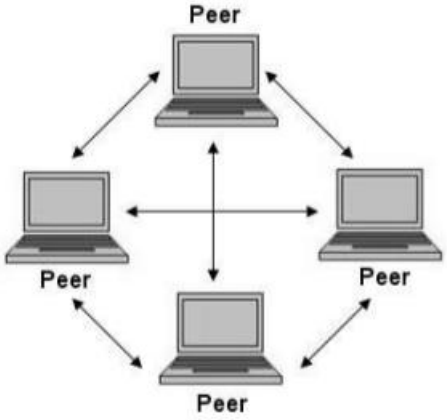
3. Horizontal Distribution Model:

- **What it is**: This model is used to improve **scalability** (ability to handle more users) and **reliability**.
- **How it works**: The server's functions are copied to multiple machines. Each machine has a full copy of the server's data, and client requests are sent to any of these servers.
- **Example**: Websites like Google or Amazon that use many servers to share the load, so if one server fails, others can continue serving requests.
- **Key point**: Multiple servers hold the same data to balance the load and ensure no downtime.



4. Peer-to-Peer (P2P) Model:

- **What it is**: This is a **decentralized** communication model.
- **How it works**: In this model, every node (computer) acts as both a client and a server. Any computer can request services and also provide services to others.
- **Example**: File-sharing systems like **BitTorrent** or **Skype**, where each user's computer can both share files and download files from others.
- **Key point**: No central server; all nodes are equal and can act as both client and server.



5. Hybrid Model:

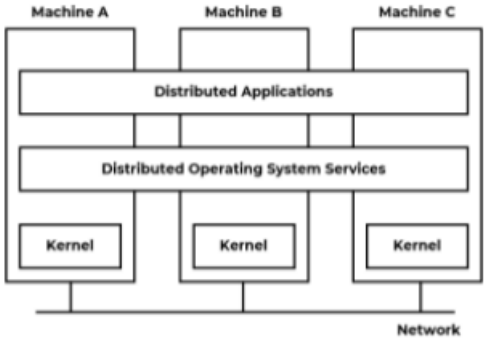
- **What it is**: This model combines two or more of the models above.
- **How it works**: It uses a combination of client-server, peer-to-peer, and/or other models to meet the needs of the system.
- **Examples**:
    - **Super-Peer Networks**: A mix of client-server and peer-to-peer models. **BitTorrent** is a good example, where some peers act as servers (super-peers) and others act as clients.
    - **Edge-Server Networks**: Servers are placed at the edges of the network (close to users), like in **Internet Service Providers (ISPs)**. Users access these nearby edge servers instead of distant main servers, improving speed.
- **Key point**: It combines the benefits of different models for better performance or specific needs.

Q. Explain in brief the software concept of distributed systems

1. Distributed Operating System:

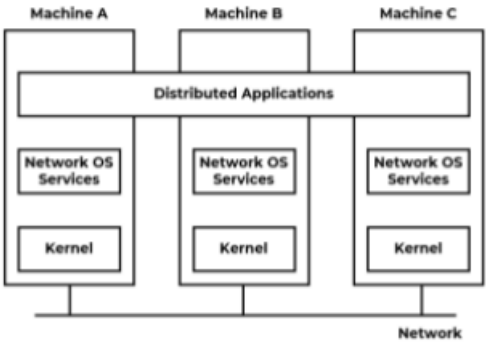
- **What it is**: This is a system where multiple computers are connected, and applications run on them as if they were one single system.
- **How it works**: The system hides the differences between the computers and makes them appear as a single machine. It helps manage resources, communication, and operations across different computers.
- **Example**: A network of computers running applications together and sharing resources like printers, storage, etc.

- **Goal**: To manage and hide the hardware complexities from users and provide a unified experience.
- **Advantages**:
    - Sharing resources easily.
    - High reliability.
    - Better communication.
    - Faster computations.



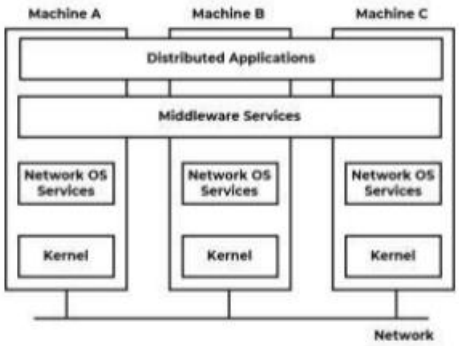
2. Network Operating System:

- **What it is**: This system connects several autonomous computers in a network. Each computer still manages its own resources, but they can communicate and share information.
- **How it works**: A network operating system allows different computers to connect through a network (like a LAN or internet) and access resources from each other.
- **Example**: When you access a file or print from a shared printer in a local network, the system provides connectivity and communication.
- **Goal**: To provide services to local computers and allow them to access remote resources.
- **Advantages**:
    - Centralized servers are stable and easier to manage.
    - Better security through server management.
    - Allows remote access to services.



3. Middleware:

- **What it is**: Middleware is software that acts as a bridge between the network operating system and applications. It provides a layer of services to make it easier for applications to communicate across different computers.
- **How it works**: It helps hide the complexity of the network and makes it easier to develop distributed applications. Middleware provides services like communication, security, and data handling.
- **Example**: Think of it as a service that connects a web server and a database, allowing them to interact smoothly.
- **Goal**: To make it easy for applications to work together in a distributed system without worrying about the underlying hardware or network details.
- **Services provided**:
    - **Communication Services**: Allows different systems to communicate with each other.
    - **Information System Services**: Helps manage data across systems.
    - **Control Services**: Manages how the system functions.
    - **Security Services**: Ensures that the system is secure and protected from unauthorized access.



Q. Explain in difference between network operating and distributed operating system and middleware

Parameter	Network Operating System (NOS)	Distributed Operating System (DOS)	Middleware
Definition	NOS is an operating system for computers connected on a LAN (Local Area Network).	DOS is an operating system that manages multiple computers and devices in a distributed system.	Middleware is software that connects different systems and applications in a distributed environment.
Goal	The goal of NOS is to offer local services to remote clients.	The goal of DOS is to hide and manage hardware resources across multiple computers.	Middleware's goal is to provide a smooth interaction and distribution of services between different applications.
OS Type	NOS is a **tightly coupled** system (each computer manages its own resources).	DOS is a **loosely coupled** system (multiple computers work together as one system).	Middleware works alongside an existing operating system to facilitate communication between distributed systems.
Used For	Used for connecting and managing **heterogeneous** computers on a network.	Used for managing **homogeneous** systems, often with multi-processors working together.	Used for connecting **heterogeneous** systems and allowing them to communicate with each other.
Architecture	Follows a **2-tier client-server architecture** (simple server-client communication).	Follows a **n-tier client-server architecture** (more complex communication between multiple systems).	Middleware doesn’t have a specific architecture; it supports communication and integration across various architectures.
Types	Can be used for **multicomputer** and	Used for **peer-to-peer** and **client-server** architectures.	Middleware connects various applications and

Parameter	Network Operating System (NOS)	Distributed Operating System (DOS)	Middleware
	multiprocessor systems.		can support different types of systems.
Communication Mode	Uses **files** for communication between systems.	Uses **messages** for communication between systems.	Middleware handles communication by sending messages between different distributed systems.
Transparency	Provides **low transparency** (users may be aware of the network's existence).	Provides **high transparency** (hides the distribution of systems from users).	Middleware hides the complexity of the system, offering transparent communication.
Reliability	**Low reliability** due to limited management across systems.	**High reliability** due to better integration and management of resources.	Middleware can improve system reliability by offering services like error handling and security.
Example	Example: **Novell NetWare**.	Example: **Microsoft Distributed Component Object Model (DCOM)**.	Example: **Message Oriented Middleware (MOM)**.

Q.