Distributed Computing & Distributed Systems

Summary For Week #1 to Week #7

Dr. Abdulla Alali

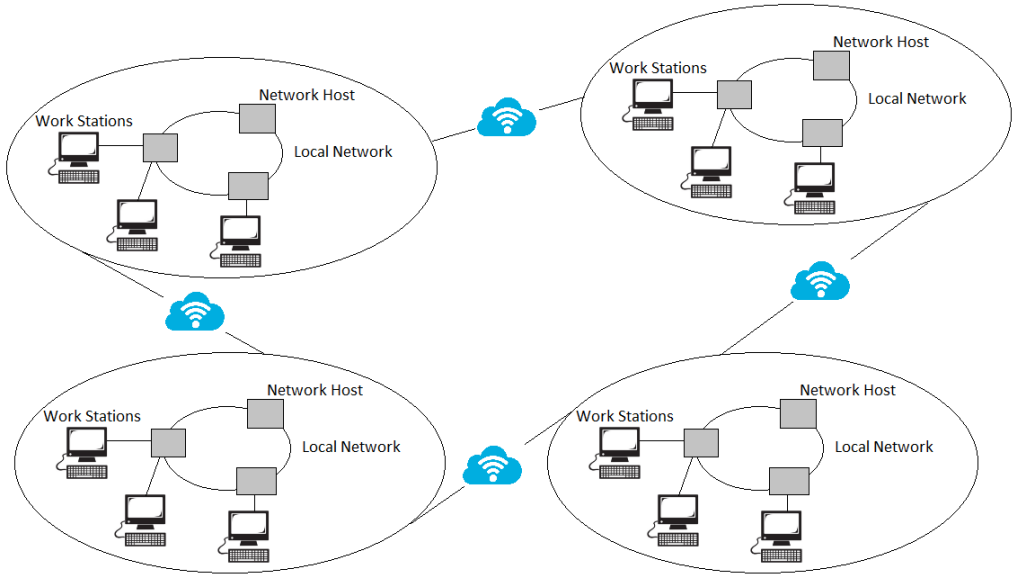
- Ahmed Al-Tamari
- 4th Year
- Computer Since

Table of Content

Overview

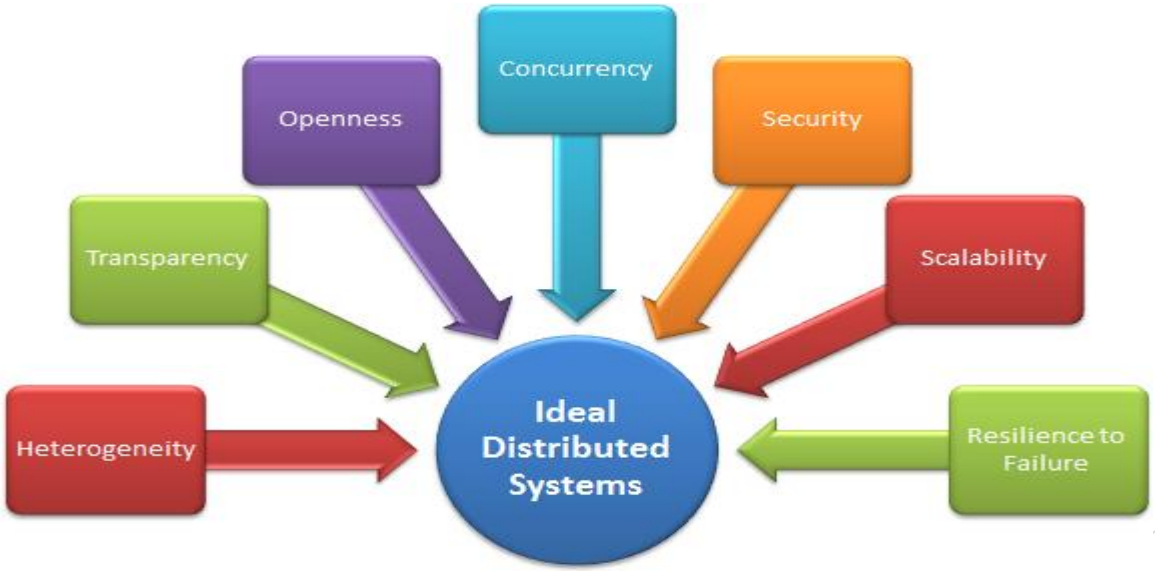
Distributed Computing & Distributed Systems: Distributed computing refers to the use of multiple computers or nodes working together in a coordinated manner to solve a problem or perform a task.

A distributed system is a software system in which components or processes located on different networked computers communicate and coordinate their actions by passing messages.



Characteristics of Distributed Systems:

- 1. Concurrency: Multiple tasks can be executed concurrently.
- 2. Transparency: Users perceive the system as a single, coherent entity.
- 3. Scalability: Ability to accommodate increasing workloads by adding more nodes.
- 4. Fault Tolerance: Ability to continue functioning despite node failures.
- 5. Interoperability: Components from different vendors can work together seamlessly.



Advantages of Distributed Systems:

- 1. Scalability: Distributed systems can handle increasing loads by adding more nodes.
- 2. Fault Tolerance: They are resilient to failures as tasks can be rerouted to other nodes.
- 3. Performance: Tasks can be executed concurrently, leading to potentially faster processing times.
- 4. Resource Sharing: Resources such as data storage and processing power can be shared among nodes.
- 5. Geographical Distribution: Allows for systems to be spread across different locations, enabling global reach.

Disadvantages of Distributed Systems:

- 1. Complexity: Designing and managing distributed systems can be complex due to the need for coordination and communication between nodes.
- 2. Security: Security risks increase with more nodes and communication channels.
- 3. Consistency: Ensuring data consistency across distributed systems can be challenging.
- 4. Latency: Communication overhead between nodes can lead to increased latency.
- 5. Cost: Setting up and maintaining distributed systems can be expensive due to the need for multiple nodes and networking infrastructure.

Examples of Distributed Systems:

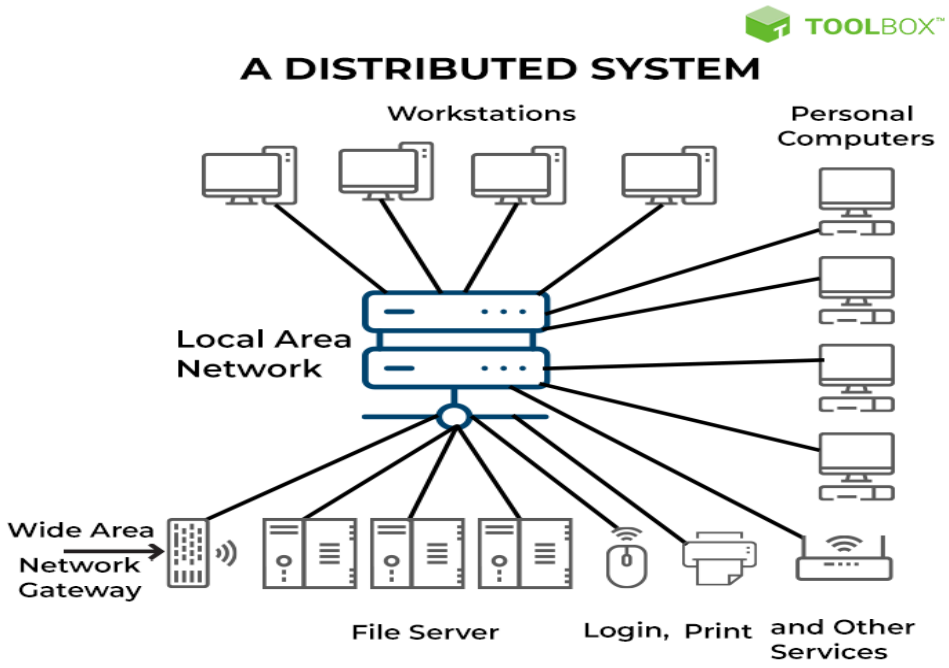
- 1. World Wide Web: A network of servers and clients distributed globally, sharing information through hyperlinks.
- 2. Cloud Computing Platforms: Services like Amazon Web Services (AWS) and Microsoft Azure provide distributed computing resources over the internet.
- 3. Peer-to-Peer Networks: File-sharing networks like BitTorrent utilize distributed systems to distribute files among users.
- 4. Distributed Databases: Systems like Apache Cassandra and MongoDB store data across multiple nodes for scalability and fault tolerance.

Resource Sharing & The Web:

- Resource sharing in distributed systems refers to

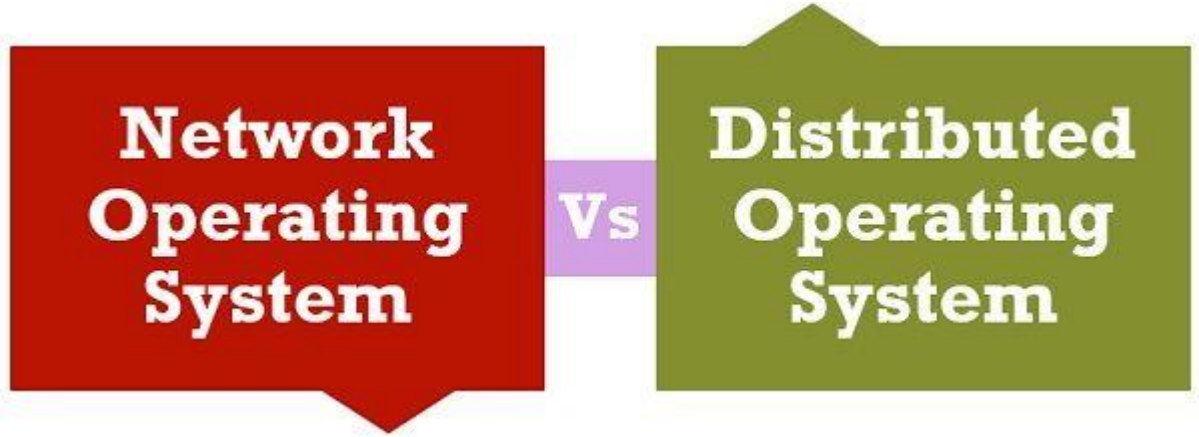
the ability of multiple nodes to access and utilize shared resources such as data storage, processing power, and peripherals.

The World Wide Web is a prime example of resource sharing in action, where millions of users access and share resources like web pages, images, and videos hosted on distributed servers worldwide.



Networks vs. Distributed Systems:

- Distributed systems , involve multiple interconnected computers or nodes working together to achieve a common goal. While networks provide the underlying communication infrastructure, distributed systems utilize this network to enable collaboration and resource sharing among nodes.

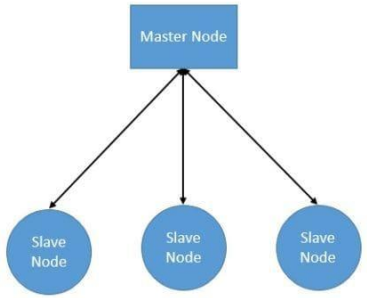


- Networks refer to the infrastructure that enables communication between computers or devices, allowing data to be exchanged.

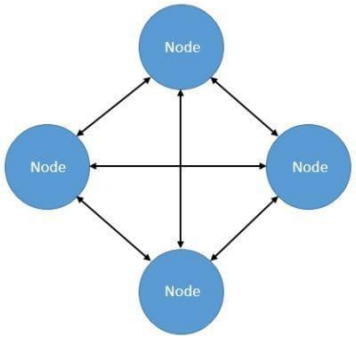
Architectures of Distributed Systems:

- Distributed systems can be categorized into various architectural models, including
- Client-server architecture.
- Peer-to-peer architecture.
- Hybrid models.

In client-server architecture, clients request services from centralized servers, while in peer-to-peer architecture, nodes communicate directly with each other without a central server. Hybrid models combine elements of both client-server and peer-to-peer architectures, offering flexibility and scalability.



Master-Slave Model



Peer-to-Peer Model

Distributed Systems: Challenges:

- 1. Complexity: Designing, implementing, and managing distributed systems can be complex due to the need for coordination and communication between nodes.
- 2. Security: Ensuring the security of data and communication channels in distributed systems is challenging due to the increased attack surface.
- 3. Consistency: Maintaining data consistency across distributed nodes can be difficult, especially in the presence of network partitions or failures.
- 4. Latency: Communication overhead between distributed nodes can lead to increased latency, affecting system performance.
- 5. Fault Detection and Recovery: Detecting and recovering from node failures in a timely manner is crucial for maintaining system reliability and availability.
- 6. Heterogeneity: Distributed systems often involve nodes with different hardware, operating systems, and software platforms, posing compatibility and interoperability challenges.

Physical Models:

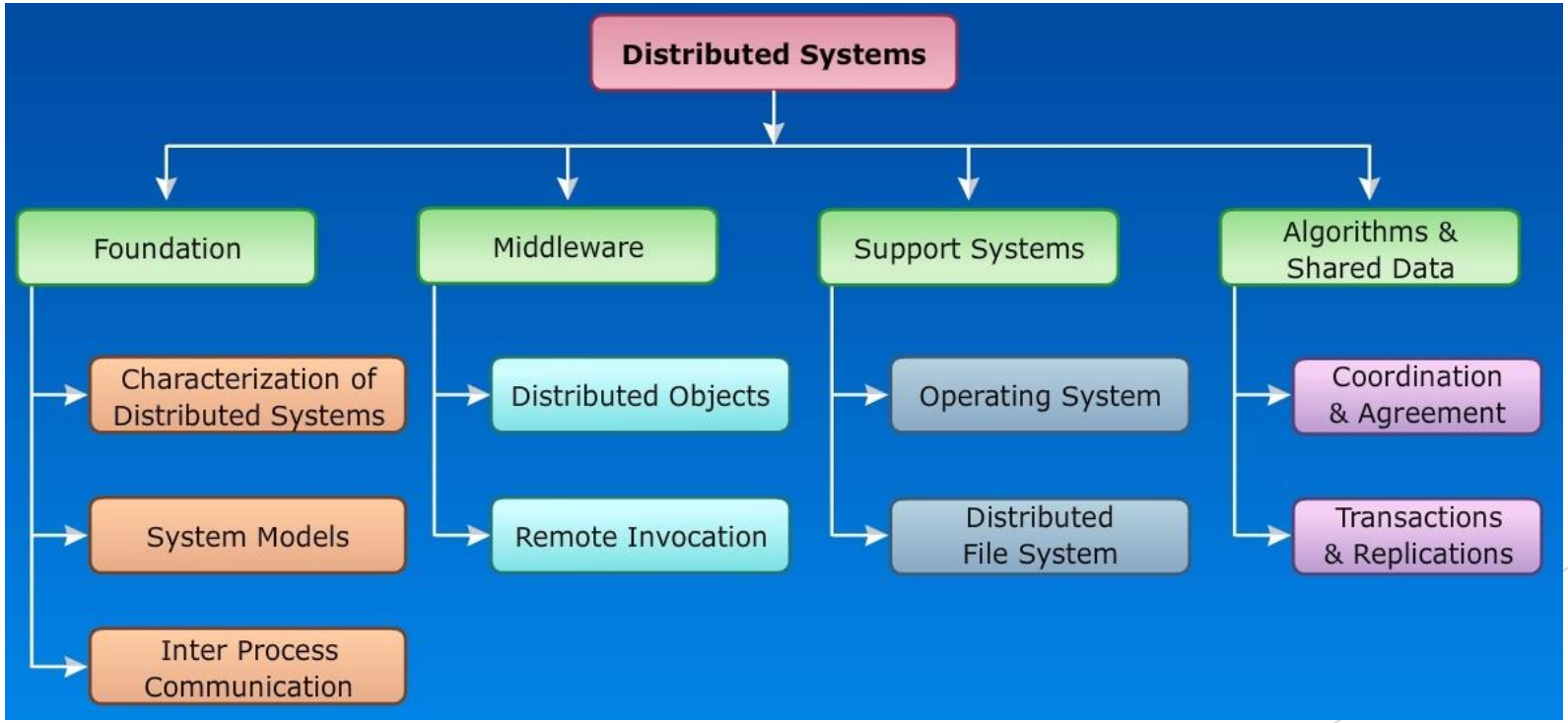
- Physical models are representations of real-world systems or objects designed to mimic their physical characteristics and behaviors.
- These models are used across various fields such as physics, engineering, and biology to understand, analyze, and predict the behavior of complex systems or phenomena.

Physical models

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

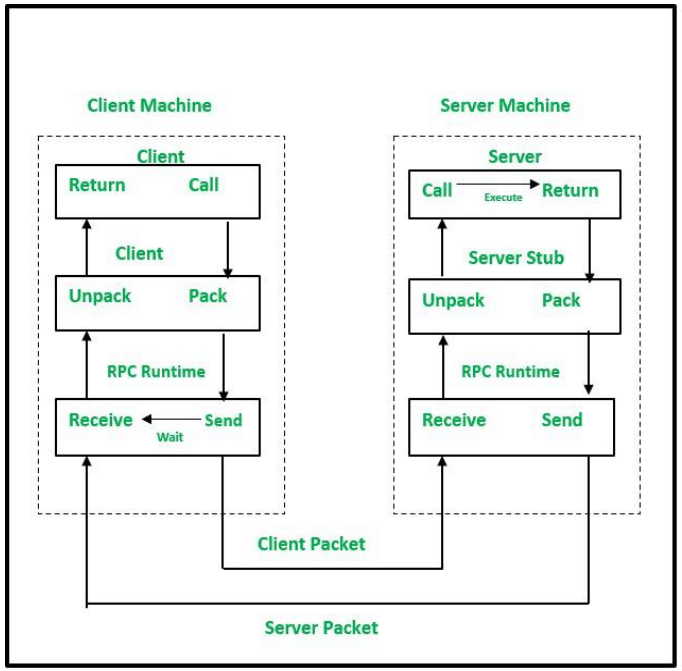
Architectural Models:

- Architectural models provide a high-level overview of the structure and organization of a system or object. In the context of physical models, architectural models define the overall design, layout, and components of a system, providing a blueprint for its construction. These models often include details such as the arrangement of components, connections between them, and the overall spatial relationships.



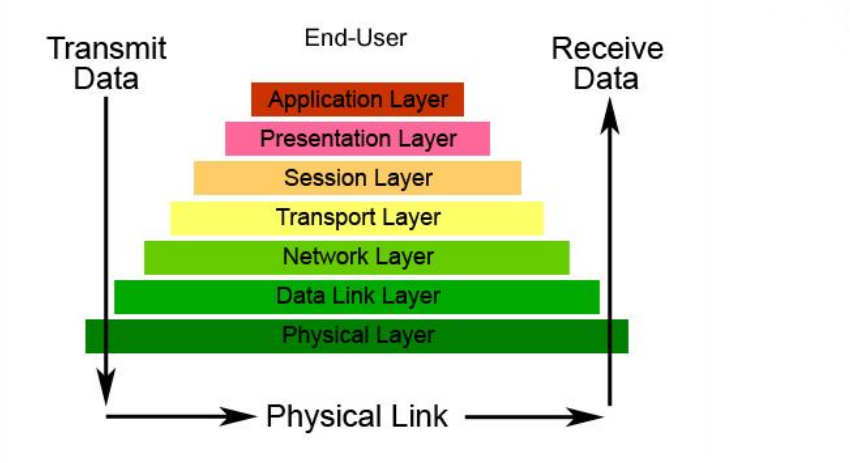
Fundamental Models – Formal Description:

- Fundamental models in physical modeling provide a formal description of the underlying principles, laws, or equations governing the behavior of a system. These models are typically based on fundamental scientific principles and mathematical formulations. They serve as the basis for more complex models and simulations used to analyze and predict the behavior of physical systems.



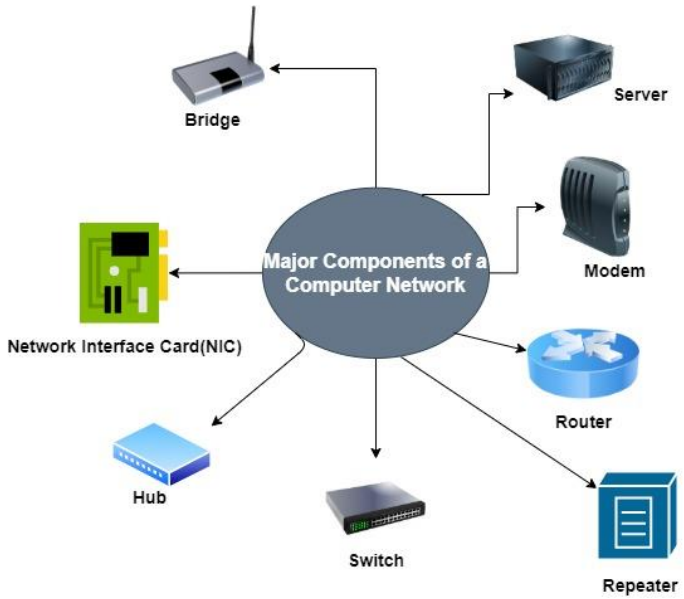
Network Layers of the Model:

- Computer networks are typically organized into layers, each responsible for specific functions and interactions. The OSI (Open Systems Interconnection) model and the TCP/IP (Transmission Control Protocol/Internet Protocol) model are two commonly used reference models that define these layers. The OSI model consists of seven layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application. The TCP/IP model combines the OSI model's last three layers into one Application layer, resulting in four layers: Link, Internet, Transport, and Application. These layers work together to facilitate communication between devices over a network, providing a standardized framework for network protocols and services.



Computer Networks:

- Computer networks enable devices to communicate and share resources, facilitating data exchange and collaboration. Networks can be classified based on their geographical scope (LANs, MANs, WANs), their connection method (wired or wireless), or their topologies (star, bus, ring). Common network components include routers, switches, hubs, and modems. Networks utilize various protocols and technologies to transmit data, including Ethernet,Wi-Fi, TCP/IP, and HTTP. They play a crucial role in modern computing, enabling the internet, intranets, and other communication systems.



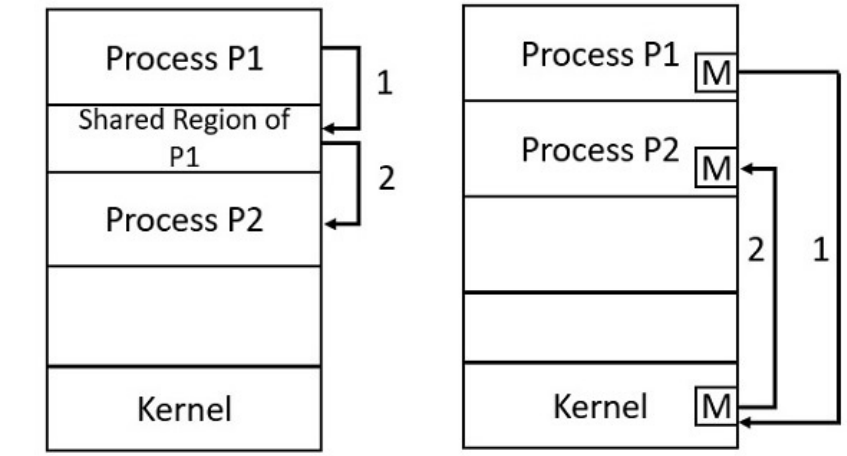
Operating Systems:

- An operating system (OS) is software that manages computer hardware and provides a platform for running applications. Network OS, Distributed OS, and Middleware are specialized types of operating systems that serve specific purposes in distributed computing environments:
- • Network OS: Network operating systems manage resources and services across a network of interconnected computers. They provide features such as file sharing, printing, and user authentication. Examples include Novell NetWare and Microsoft Windows Server.
- • Distributed OS: Distributed operating systems extend the capabilities of traditional OS to manage resources distributed across multiple computers or nodes. They facilitate transparent access to distributed resources and provide mechanisms for communication and coordination between nodes. Examples include Amoeba and Google's Android.
- • Middleware: Middleware is software that acts as an intermediary between different applications or systems, providing a communication layer and enabling interoperability. It abstracts the complexities of distributed computing, allowing applications to interact with distributed resources seamlessly. Examples include CORBA (Common Object Request Broker Architecture) and RPC (Remote Procedure Call).

Inter-process Communication (IPC):

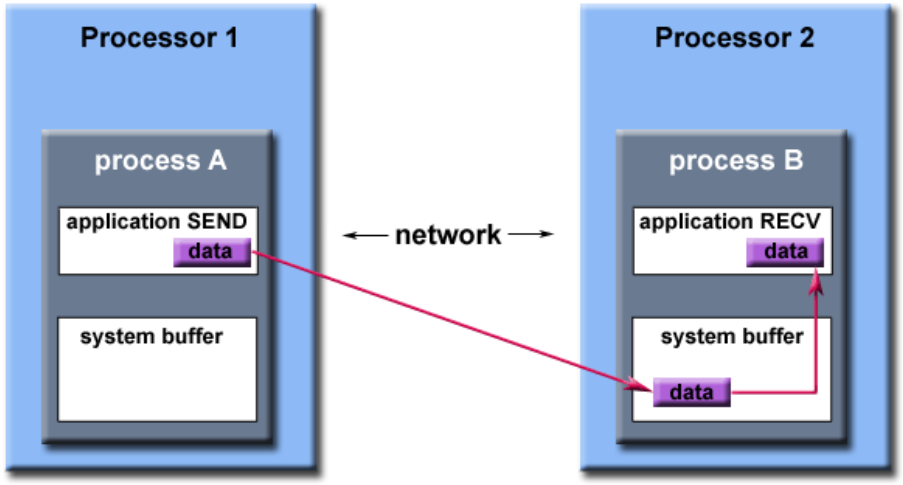
- Inter-process communication (IPC)

refers to the mechanisms and techniques used by processes or threads running on a computer to communicate and share data with each other. IPC allows processes to collaborate, exchange information, and synchronize their activities. There are various IPC mechanisms, including message passing, shared memory, pipes, sockets, and remote procedure calls (RPC), each suited for different communication scenarios and requirements.



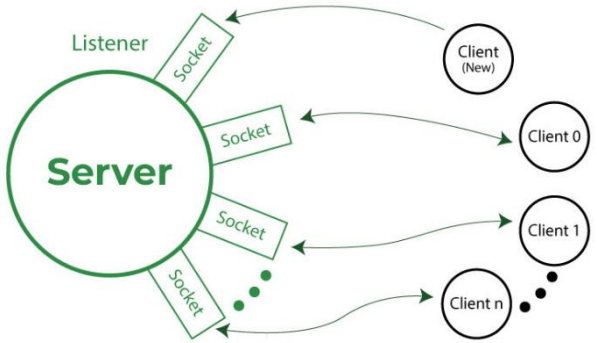
Message Passing and Buffer:

- Message passing is a fundamental IPC mechanism where processes communicate by sending and receiving messages. Messages typically contain data or control information and are exchanged through predefined communication channels. A buffer is a temporary storage area used to hold messages before they are sent or after they are received. Buffers help manage the flow of data between processes, ensuring that messages are delivered efficiently and reliably.



Path of a message buffered at the receiving process

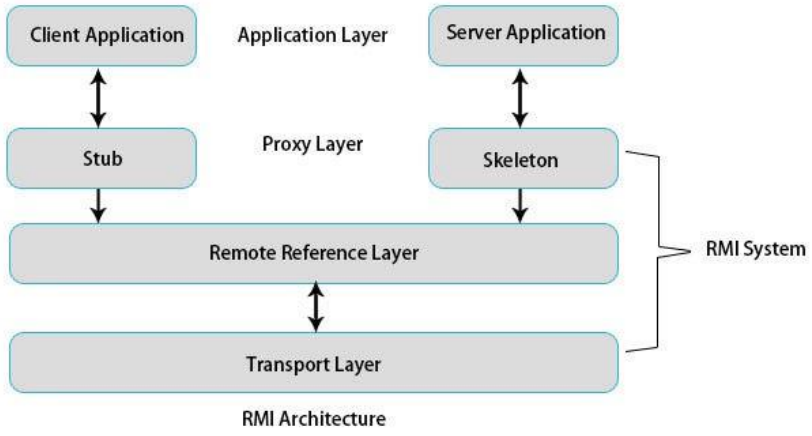
Sockets, RPC, and RMI:



- • Sockets: Sockets are a communication endpoint that enables bidirectional data flow between processes over a network. They provide a flexible and widely used IPC mechanism for networked applications. Sockets support various communication protocols such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) and can be implemented in programming languages like C, Python, and Java.
- • Remote Procedure Call (RPC): RPC is a protocol that allows a process to invoke a procedure or function in another process running on a remote system as if it were a local procedure call. RPC hides the complexities of network communication, enabling distributed applications to interact transparently. Common RPC frameworks include ONC RPC (Open Network Computing Remote Procedure Call) and gRPC (Google Remote Procedure Call).
- • Remote Method Invocation (RMI): RMI is a Java-specific mechanism for enabling communication between Java objects running in different Java Virtual Machines (JVMs). RMI allows Java objects to invoke methods on remote objects as if they were local objects, abstracting the details of network communication. It is commonly used in distributed Java applications, such as client-server systems and distributed computing platforms.

RMI (Remote Method Invocation):

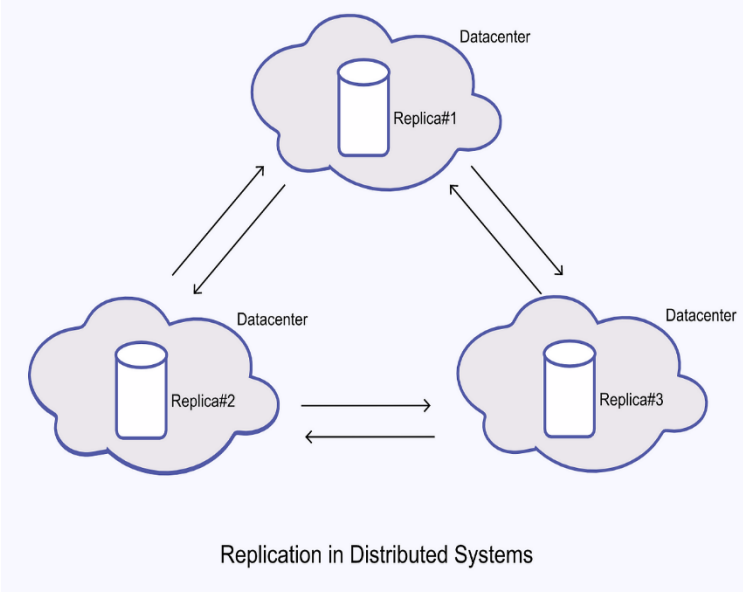
- RMI, or Remote Method Invocation, is a Java-specific mechanism for enabling communication between Java objects running in different Java Virtual Machines (JVMs). RMI allows Java objects to invoke methods on remote objects as if they were local objects, abstracting the details of network communication. RMI facilitates distributed computing by providing a straightforward way for Java applications to interact across network boundaries. It is commonly used in client-server architectures and distributed systems built on the Java platform.



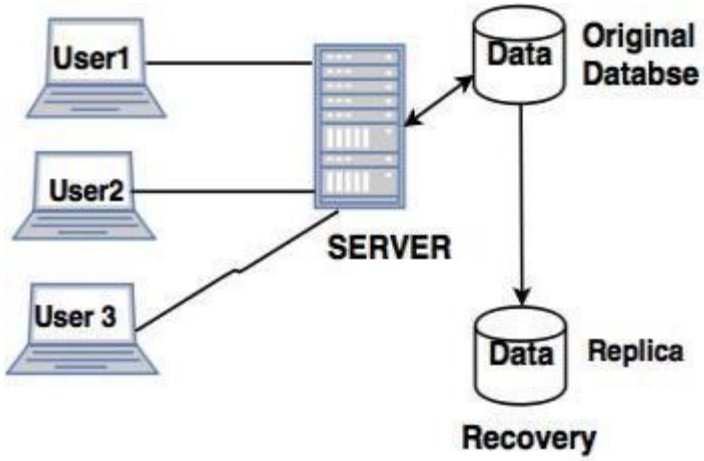
RMI Architecture

Replication Definition

- Replication in distributed systems refers to the process of creating and maintaining multiple copies of data, software, or entire systems across different nodes or locations in a network. The primary goal of replication is to enhance system performance, availability, and fault tolerance by distributing workload and data access.



Replication in Distributed Systems



Full Replication Process In Distributed System

Motivations for Replication - Part 1

- Replication enhances services by improving performance and providing replication transparency.

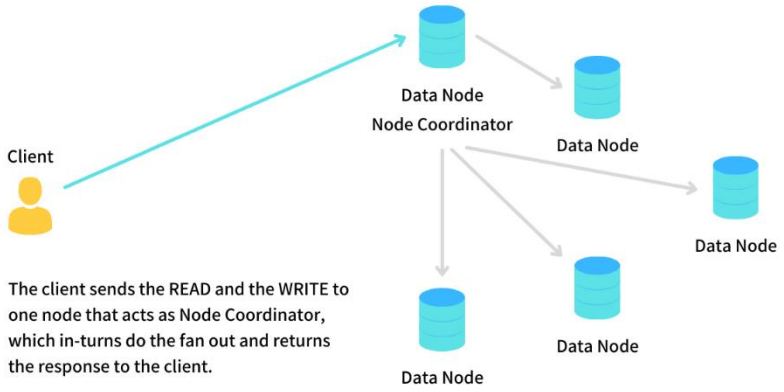
- 1. Performance Enhancement:
- - Caching of data at clients and servers improves performance.
- Example: Browsers and proxy servers cache web resources to reduce latency.
- - Workload is shared between servers using DNS round-robin or more sophisticated load-balancing strategies.

Motivations for Replication - Part 2

- 2. Increased Availability:
- - Services should be highly available, with minimal downtime.
- - Factors affecting availability: server failures, network partitions, and disconnected operations.
- - Caching systems do not necessarily enhance availability at the application level.
- - Replicating server data improves availability.
- Example: For a system with two servers, each with a 5% failure probability, availability is 99.75%.

Motivations for Replication - Part 3

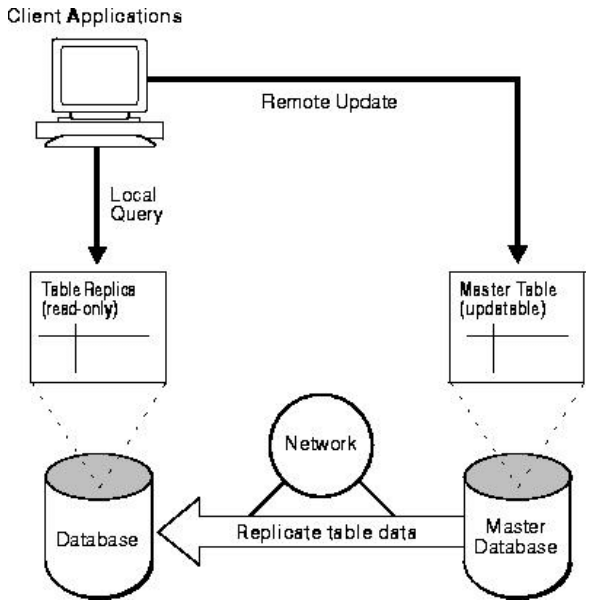
- 3. Fault Tolerance:
- - Highly available data may not always be strictly correct.
- - A fault-tolerant service guarantees correct behavior despite faults.
- - Correctness includes data freshness and the effects of client operations.
- Example: Air traffic control systems require timely and correct data.
- - Replication provides both high availability and fault tolerance.



System Model

- Data consists of objects, implemented by physical copies called replicas.

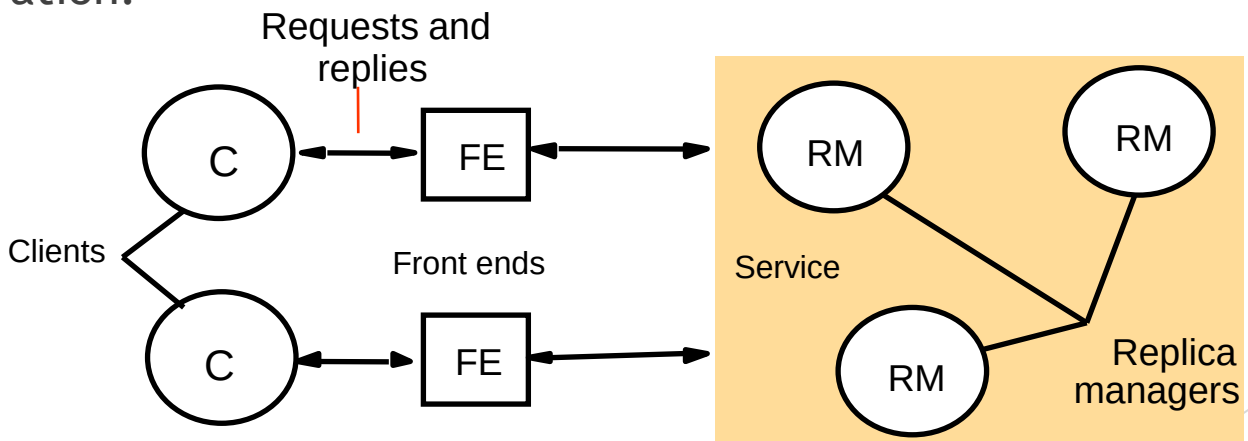
- - Replicas are stored on different computers and tied to consistency by the systems operation.
- - Replicas of an object may not be identical at all times, as updates may vary.



Architectural Model

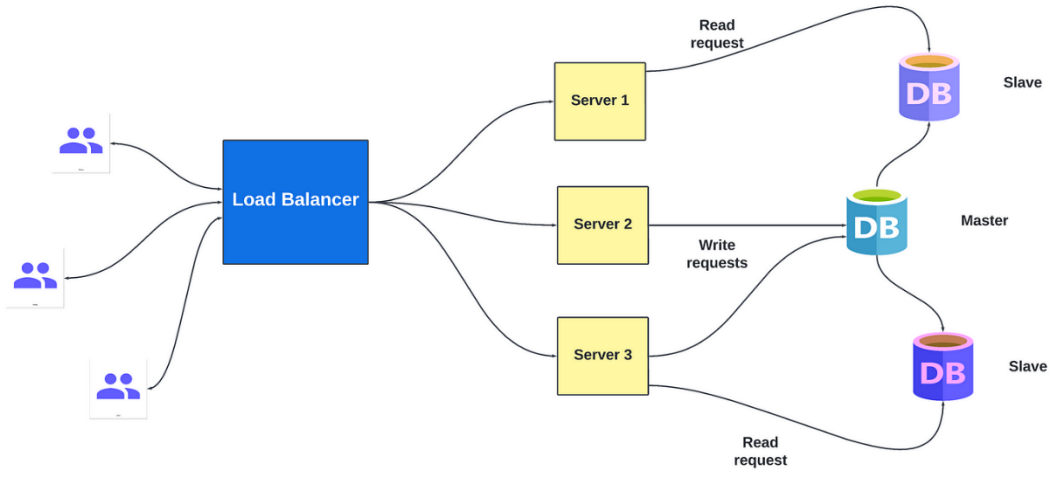
- Client requests are handled by a front end (FE), which communicates with replica managers (RMs).

- - The FE makes replication transparent.
- - Replica managers hold replicas and perform operations directly on them.
- - Operations are coordinated to ensure consistency, even if failures occur mid-operation.



Phases of a Single Request

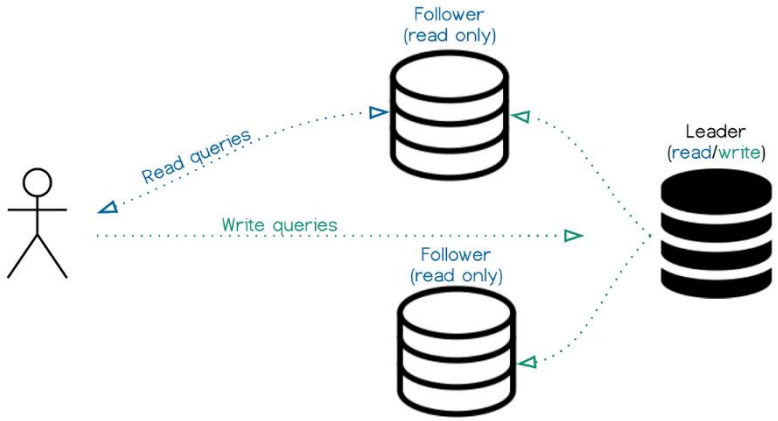
- Five phases of a request:
- 1. Request: FE issues the request to RMs.
- 2. Coordination: RMs coordinate to prepare for the request.
- 3. Execution: RMs execute the request.
- 4. Agreement: RMs exchange updates.
- 5. Response: RMs respond to the FE.



View-Synchronous Group Communication

- View-synchronous group communication ensures ordered delivery of messages and view notifications.

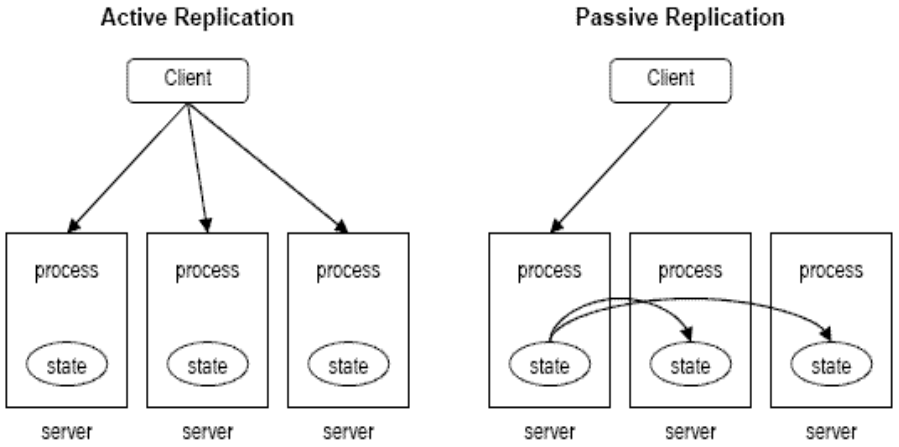
- - Agreement: All correct processes deliver messages in the same view.
- - Integrity: Messages are not delivered more than once.
- - Validity: Correct processes deliver their own messages.

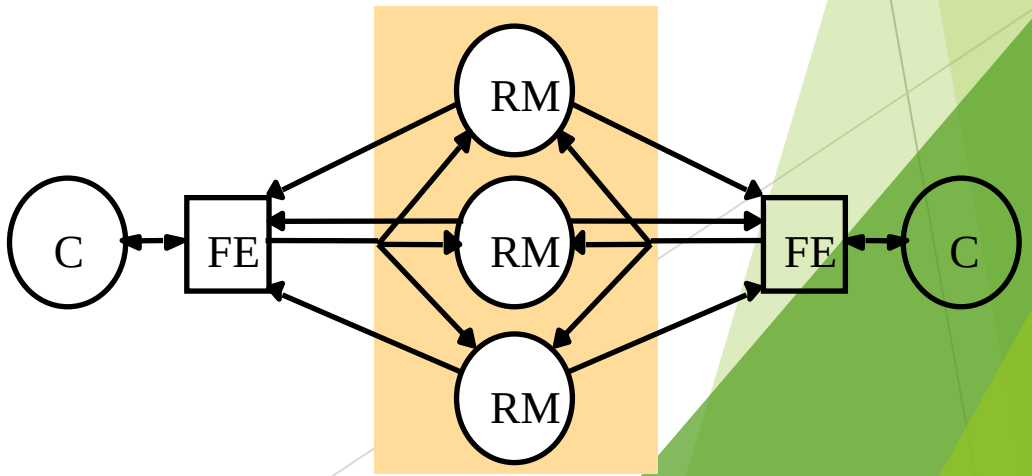


Fault Tolerance Models

- 1. Passive (Primary-Backup) Model:
- - FE sends request to primary RM, which coordinates with backups.
- - Primary RM responds to FE.

- 2. Active Replication Model:
- - FE multicasts request to all RMs.
- - All RMs execute the request and respond to FE.

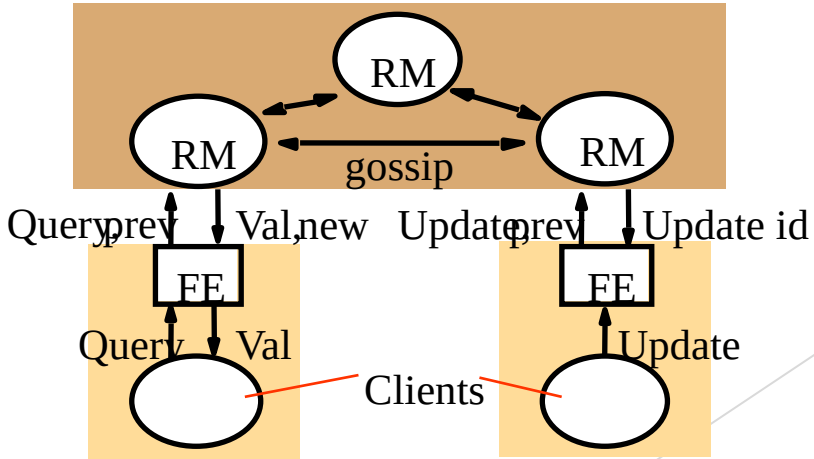




Gossip-Based Replication

- Gossip services process queries and updates, ensuring relaxed consistency between replicas.

- - FEs send requests to a single RM.
- - RMs exchange gossip messages to update each other.
- - Guarantees consistent service over time and supports causal update ordering.



Thank You!

Any Questions?