**Message Passing in Distributed System :**

* Message passing in [distributed systems](https://www.geeksforgeeks.org/what-is-a-distributed-system/) refers to the communication medium used by nodes (computers or processes) to communicate information and coordinate their actions.
* It involves transferring and entering messages between nodes to achieve various goals such as coordination, synchronization, and data sharing.

**What is Message Passing in Distributed Systems?**

* The method by which entities or processes in a distributed system communicate and exchange data is known as message passing.
* It enables several components, which could be operating on different computers or nodes connected by a network, to plan their actions, exchange data, and work together to accomplish shared objectives.
* Models like synchronous and asynchronous message passing offer different [synchronization](https://www.geeksforgeeks.org/synchronization-in-distributed-systems/) and communication semantics to suit system requirements.
* Synchronous message passing ensures sender and receiver synchronization, while asynchronous message passing allows concurrent execution and non-blocking communication.

**Types of Message Passing in Distributed Systems :**

Message passing in distributed systems can be categorized into two types based on the timing and synchronization between sender and receiver:

**1. Synchronous Message Passing**

In synchronous message passing, the sender and receiver coordinate tightly, and the sender waits for an acknowledgment or response before continuing execution. This type of communication typically follows a **request-response pattern**.

* **Key Features**:
  + **Timing Coordination**: The sender blocks execution until the receiver confirms receipt or finishes processing the message.
  + **Request-Response Paradigm**: Common in client-server models where the sender sends a request and waits for the receiver's response.

**2. Asynchronous Message Passing**

Asynchronous message passing allows the sender to continue execution immediately after sending the message, without waiting for an acknowledgment. The receiver processes the message at its own pace, often following an **event-driven model**.

* **Key Features**:
  + **Decoupled Timing**: The sender and receiver operate independently, enabling non-blocking communication.
  + **Event-Driven**: Processes handle messages or events as they occur, enabling greater flexibility and responsiveness.

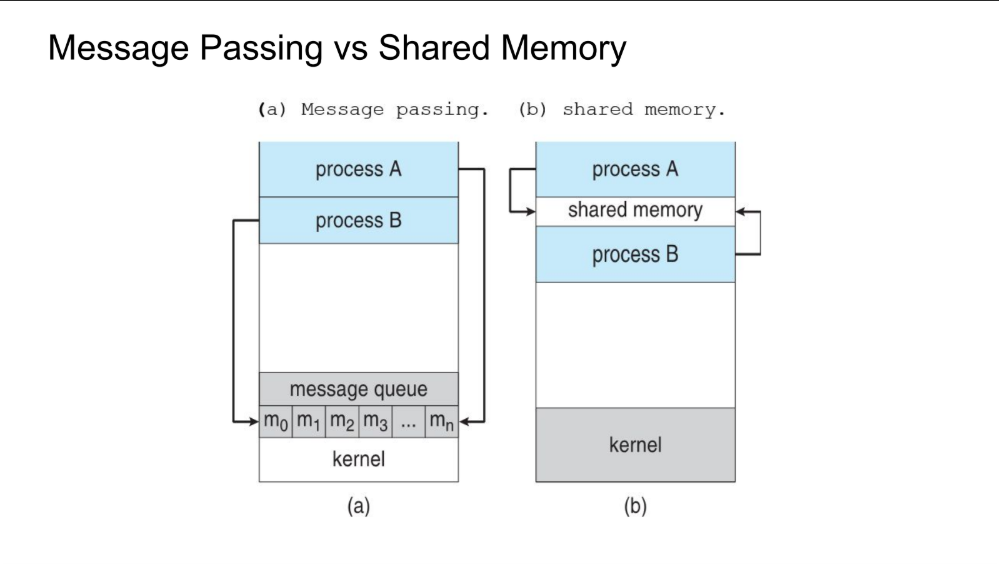
**3. Unicast Messaging**

Unicast messaging is a one-to-one communication where a message is sent from a single sender to a specific receiver.

**4. Multicast Messaging**

Multicast messaging enables one-to-many communication, where a message is sent from one sender to a specific group of receivers.

**Daigram :**



**Distributed Shared Memory (DSM):**

* DSM is a mechanism that allows processes across multiple nodes to access shared data seamlessly, without explicit inter-process communication.
* It creates the illusion of shared memory, even though no physical shared memory exists.
* Each node has its own memory and offers memory read/write services, ensuring data consistency through specific protocols.
* DSM manages a shared virtual address space, enabling transparent data sharing.

**Shared Memory Systems:**

1. **Shared Address Space:**  
   All processors use the same memory space, making it easy to share and access data directly.
2. **Synchronization:**  
   To avoid problems like two processors trying to update the same data at the same time, tools like semaphores or locks are used to coordinate access.
3. **Message Passing in Multicomputers**: Systems without shared memory (e.g., NUMA) use **message passing**, where processors exchange information by sending and receiving messages.

**Emulating Message Passing in Shared Memory:**

* **Dividing Memory:**Shared memory is split into sections, with each processor managing its own part.
* **Mailboxes:**Processors communicate by writing messages into specific memory locations (mailboxes) assigned to other processors.
* **Synchronization:**Rules and tools ensure that processors don’t read or write at the wrong time, keeping communication organized.

**Emulating Shared Memory in Message-Passing Systems:**

1. **Send/Receive Messages for Data:** Instead of direct memory access, processors send messages to write or request data.
2. **Message Passing**: Shared memory operations (like reading and writing) are replaced with sending and receiving messages to the process that "owns" the shared memory location.
3. **High Latency**: Since processors need to send messages for every read or write, communication is slower compared to real shared memory.
4. **Hybrid Systems**: Some systems use both shared memory and message passing to balance speed and flexibility.
5. **MIMD Systems**: In **MIMD (Multiple Instruction, Multiple Data)** systems, processors in a single computer use shared memory, but communication across multiple computers happens via messages.

**Message Passing**

**Advantages:**

1. **Flexible:** Easy to add new parts to the system.
2. **Reliable:** One part can fail without affecting the rest.
3. **Independent:** Parts can work separately and in different locations.
4. **Secure:** Each part is isolated, enhancing security.

**Disadvantages:**

1. **Complex:** Managing communication between parts can be tricky.
2. **Slow:** Messages can take time to travel, causing delays.
3. **Error-Prone:** Messages can get lost or duplicated, needing good error handling.
4. **Resource-Intensive:** Uses more network and CPU resources.

**Shared Memory**

**Advantages:**

1. **Fast:** Data access is quick since memory is shared.
2. **Efficient:** Lower communication overhead.
3. **Simple:** Easier to implement on the same machine.
4. **Immediate:** Data is available without delays.

**Disadvantages:**

1. **Sync Issues:** Requires careful synchronization to avoid data corruption.
2. **Limited Scalability:** Best suited for single machine use.
3. **Less Secure:** Processes can interfere with each other’s data.
4. **Complex:** Managing access and synchronization can get complicated as the system grows.