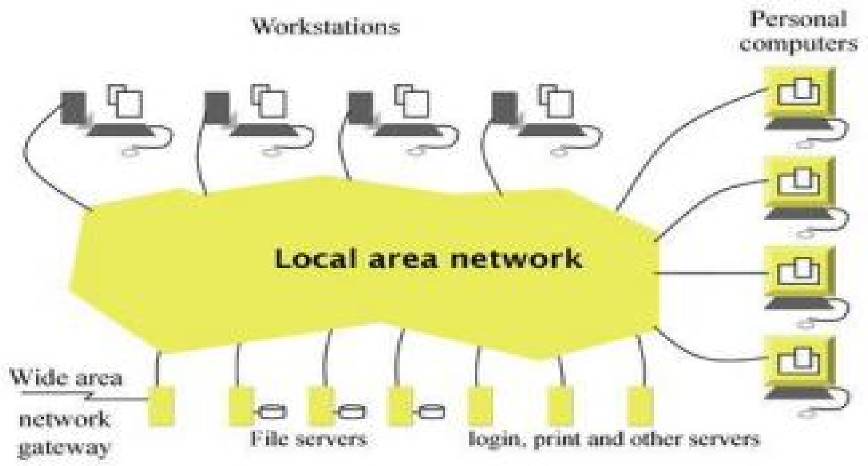
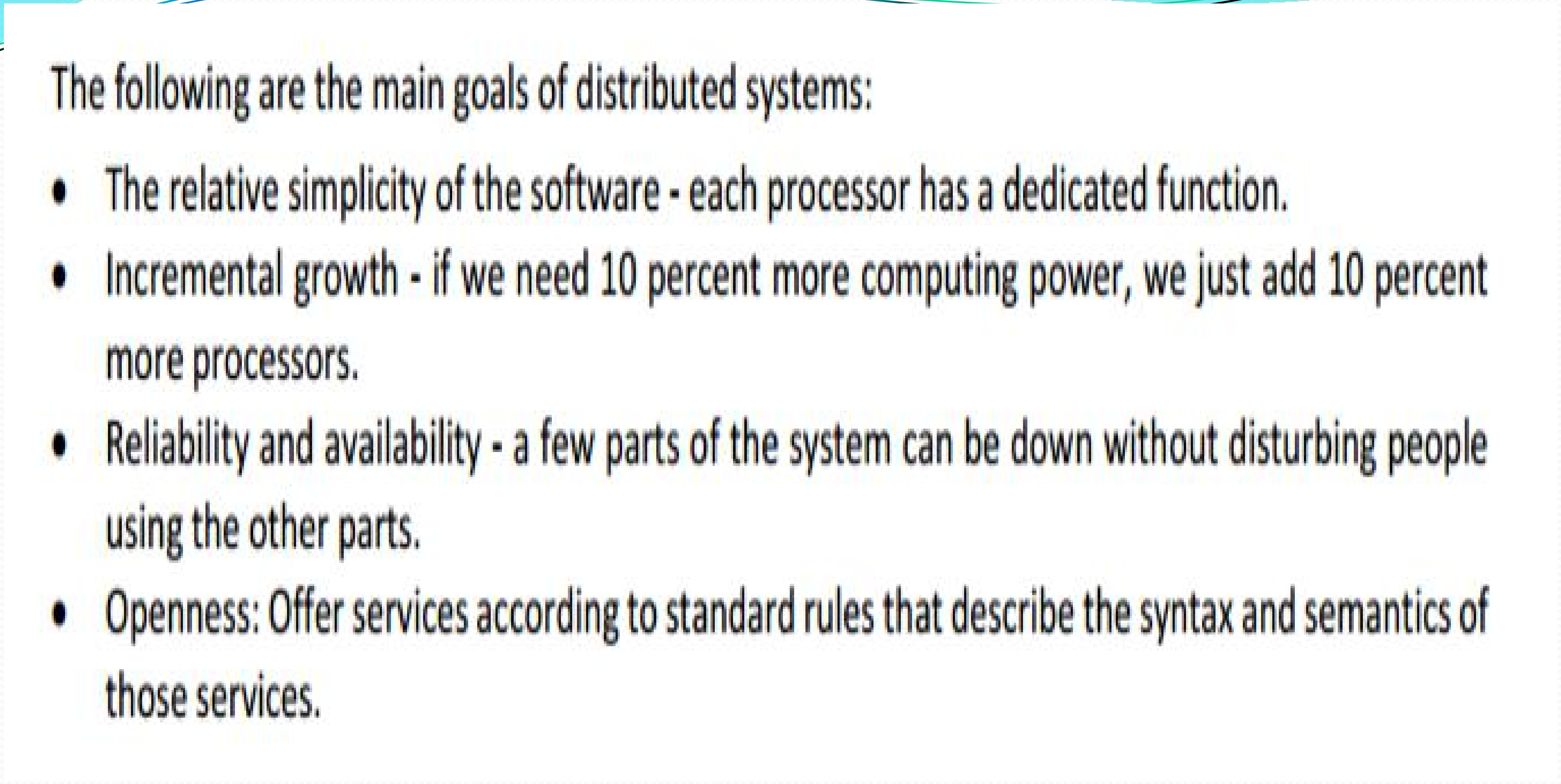
BCA
Fourth Semester
“ Operating System“

Define Distributed Operating System and Explain Goals of Distributed System.

- A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- Using high performance computers connected by equally high speed communication links, it is possible to build a single system consisting of multiple computer and using it as a single consolidated system.
- In such a system, multiple resources work together to deliver the required processing speed and the operating system takes care of maintaining the system and overall maintenance.
- In distributed system computers are not independent but interconnected by a high speed network.
- It means that many computers, be they workstation or desktop systems linked together, can do work of a high performance supercomputer.
- If user thinks that the entire interlinked system is a single and unified computer, there has to be software envelope that joins multiple operating system together and offers a seamless interface to user.

A Distributed System





The following are the main goals of distributed systems:

- The relative simplicity of the software - each processor has a dedicated function.
- Incremental growth - if we need 10 percent more computing power, we just add 10 percent more processors.
- Reliability and availability - a few parts of the system can be down without disturbing people using the other parts.
- Openness: Offer services according to standard rules that describe the syntax and semantics of those services.

Explain Advantage of Distributed system over centralized system.

Economics

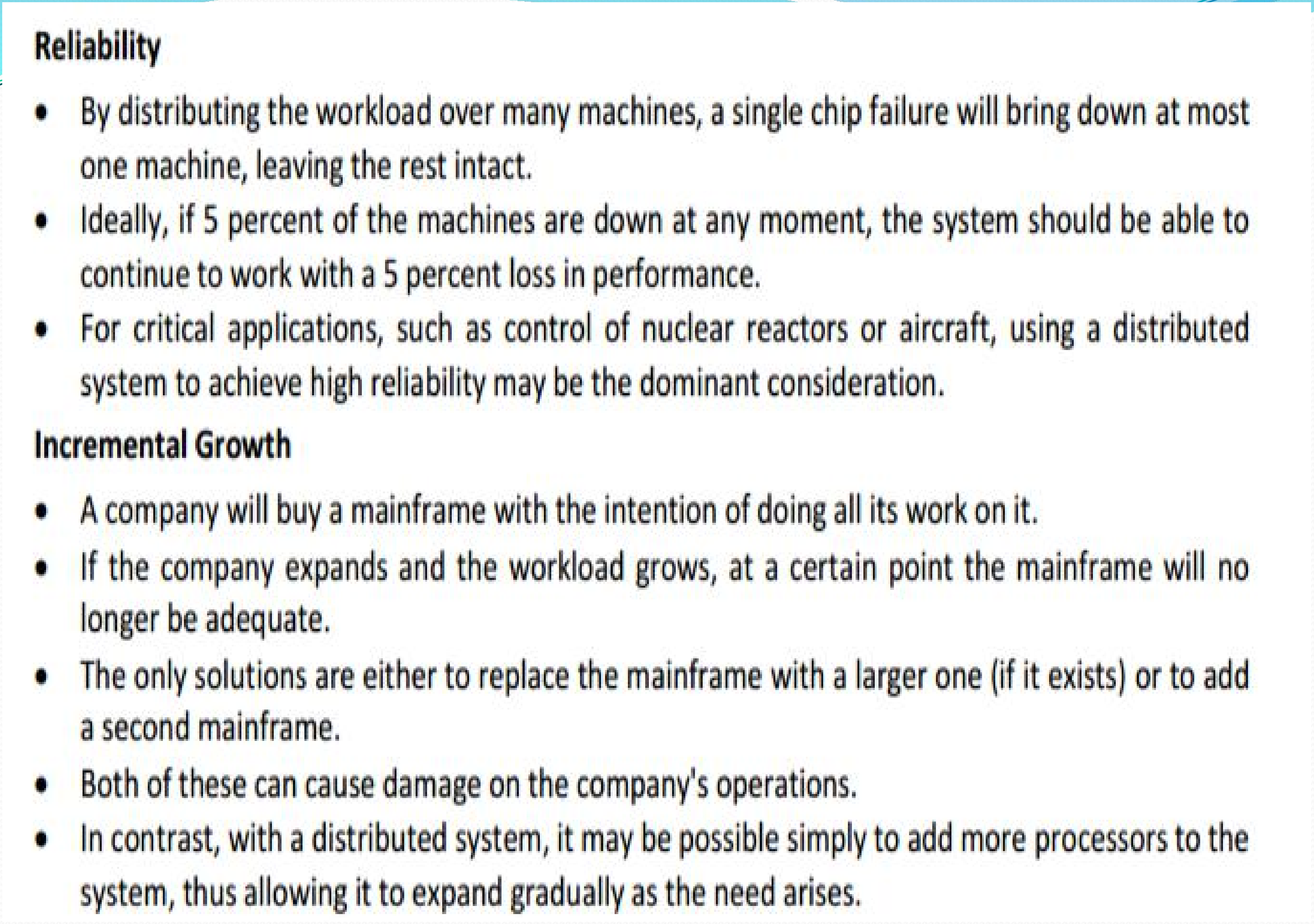
- A quarter century ago, according to Grosch's law: the computing power of a CPU is proportional to the square of its price.
- By paying twice as much you could get four times the performance.
- This observation fit the mainframe technology of its time.
- With microprocessor technology, Grosch's law no longer holds.
- For a few hundred dollars you can get the CPU chip that can execute more instructions per second than one of the largest 1980s mainframe.

Speed

- A collection of microprocessors cannot only give a better price/performance ratio than a single mainframe, but also give an absolute performance that no mainframe can achieve at any price.
- For example, with current technology it is possible to build a system from 10,000 modern CPU chips, each of which runs at 50 MIPS (Millions of Instructions Per Second), for a total performance of 500,000 MIPS.
- For a single processor (i.e., CPU) to achieve this, it would have to execute an instruction in 0.002 nsec (2 picosecond).

Inherent Distribution

- Many institutions have applications which are more suitable for distributed computation.
- Assume that there is large company buying and selling goods from different countries.
- Its offices situated in those countries are geographically diverse.
- If a company wishes unified computing system, it should implement a distributed computing system.
- Consider the global employee database of such a multinational company.
- Branch offices would create local database and then link them for global viewing.



Reliability

- By distributing the workload over many machines, a single chip failure will bring down at most one machine, leaving the rest intact.
- Ideally, if 5 percent of the machines are down at any moment, the system should be able to continue to work with a 5 percent loss in performance.
- For critical applications, such as control of nuclear reactors or aircraft, using a distributed system to achieve high reliability may be the dominant consideration.

Incremental Growth

- A company will buy a mainframe with the intention of doing all its work on it.
- If the company expands and the workload grows, at a certain point the mainframe will no longer be adequate.
- The only solutions are either to replace the mainframe with a larger one (if it exists) or to add a second mainframe.
- Both of these can cause damage on the company's operations.
- In contrast, with a distributed system, it may be possible simply to add more processors to the system, thus allowing it to expand gradually as the need arises.

Explain Advantage of Distributed system over Independent PCs.

Data sharing

- Many users need to share data.
- For example, airline reservation clerks need access to the master data base of flights and existing reservations.
- Giving each clerk his own private copy of the entire data base would not work, since nobody would know which seats the other clerks had already sold.
- Shared data are absolutely essential for this and many other applications, so the machines must be interconnected.

Resource sharing

- Expensive peripherals, such as color laser printers, phototypesetters, and massive archival storage devices (e.g., optical jukeboxes), can also be shared.

Communication

- For many people, electronic mail has numerous attractions over paper mail, telephone, and FAX.
- It is much faster than paper mail, does not require both parties to be available at the same time as does the telephone, and unlike FAX, produces documents that can be edited, rearranged, stored in the computer, and manipulated with text processing programs.

Flexibility

- A distributed system is more flexible than giving each user an isolated personal computer.
- One way is to give each person a personal computer and connect them all with a LAN, this is not the only possibility.
- Another one is to have a mixture of personal and shared computers, perhaps of different sizes, and let jobs run on the most appropriate one, rather than always on the owner's machine.
- In this way, the workload can be spread over the computers more effectively, and the loss of a few machines may be compensated for by letting people run their jobs elsewhere.

Disadvantages of Distributed System

- Security problem due to sharing.
- Some message can be lost in the network system.
- Bandwidth is another problem if there is large data then all network wires to be replaced which tends to become expensive.
- Overloading is another problem in distributed system.
- If there is database connection on local system and many users accessing that database through remote or distributed way then performance become slow.

Classify Distributed operating system.

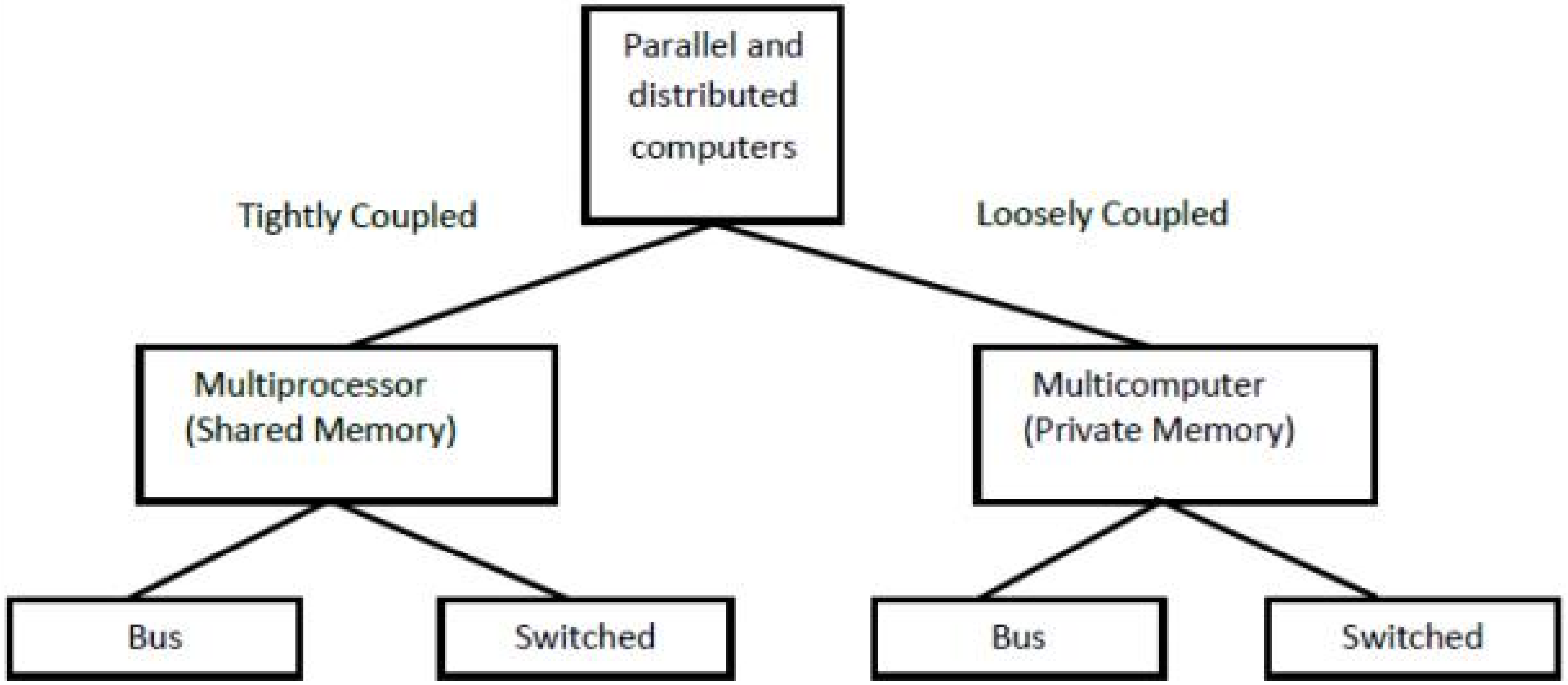

Parallel and distributed computers
Tightly Coupled
Loosely Coupled
Multiprocessor (Shared Memory)
Multicomputer (Private Memory)
Bus
Switched
Bus
Switched

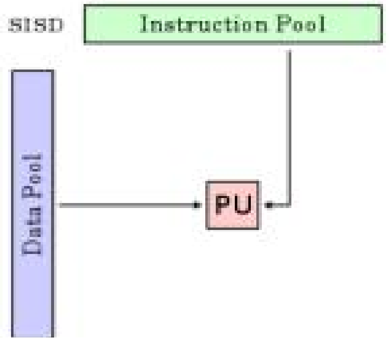
Figure: Classification of distributed system

Hardware Concepts

- Even though all distributed systems consist of multiple CPUs, there are several different ways the hardware can be organized, especially in terms of how they are interconnected and how they communicate.
- Various classification schemes for multiple CPU computer systems have been proposed over the years.
- According to Flynn, classification can be done based on, the number of instruction streams and number of data streams.

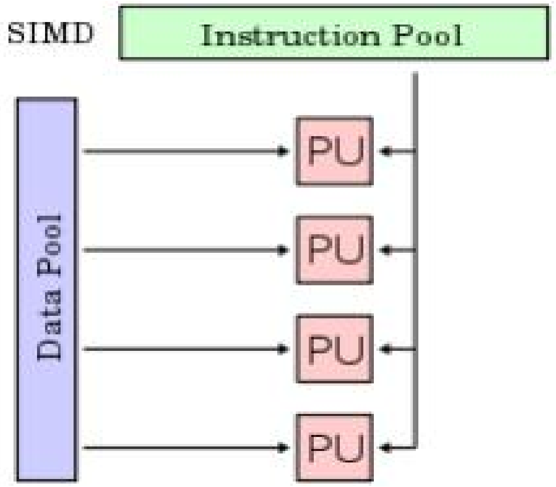
SISD

- A computer with a single instruction stream and a single data stream is called SISD.
- All traditional uniprocessor computers (i.e., those having only one CPU) fall in this category, from personal computers to large mainframes.



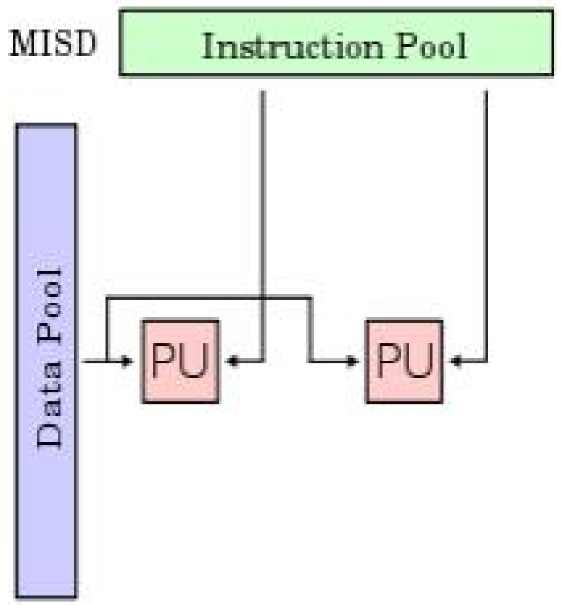
SIMD

- The next category is SIMD, single instruction stream, multiple data stream.
- This type refers to array processors with one instruction unit that fetches an instruction, and then commands many data units to carry it out in parallel, each with its own data.
- These machines are useful for computations that repeat the same calculation on many sets of data, for example, adding up all the elements of 64 independent vectors.
- Some supercomputers are SIMD.



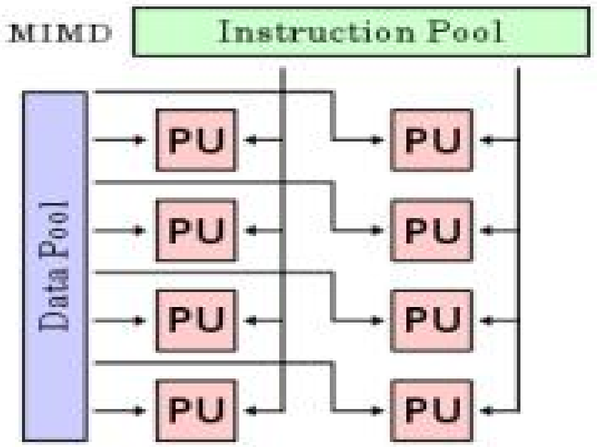
MISD

- The next category is MISD, multiple instruction stream, single data stream, no known computers fit this model.



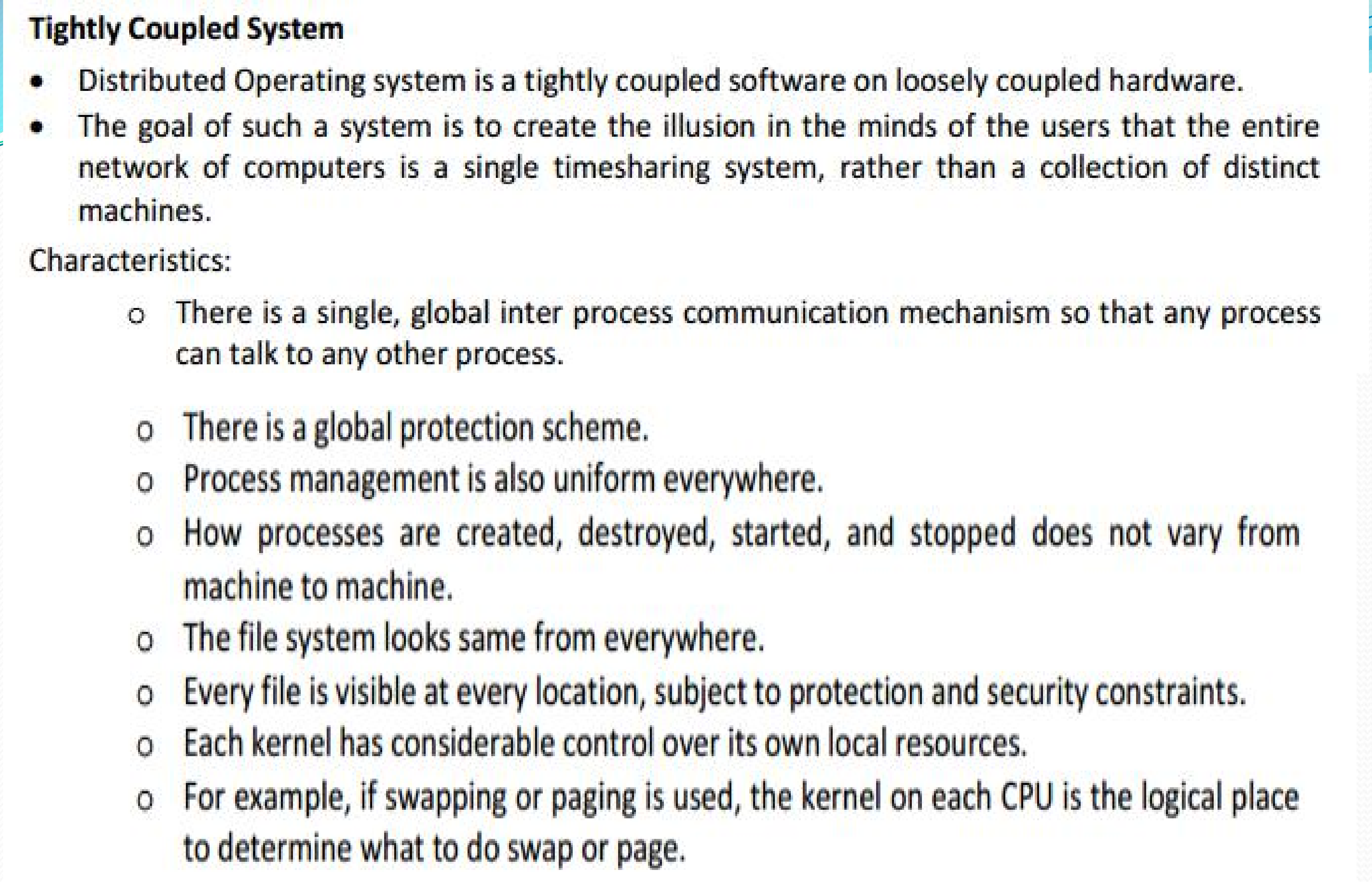
MIMD

- Finally, comes MIMD, which essentially means a group of independent computers, each with its own program counter, program, and data.
- All distributed systems are MIMD.
- We divide all MIMD computers into two groups: those that have shared memory, usually called multiprocessors, and those that do not, sometimes called multicomputer.



Software Concepts

- The image that a system presents to its users, and how they think about the system, is largely determined by the operating system software, not the hardware.
- There are basic two types of operating system namely (1) Tightly coupled operating system and (2) loosely couple operating system; for multiprocessor and multicomputer.
- Loosely-coupled software allows machines and users of a distributed system to be fundamentally independent of one another.
- Consider a group of personal computers, each of which has its own CPU, its own memory, its own hard disk, and its own operating system, but which share some resources, such as laser printers and data bases, over a LAN.
- This system is loosely coupled, since the individual machines are clearly distinguishable, each with its own job to do.
- If the network should go down for some reason, the individual machines can still continue to run to a considerable degree, although some functionality may be lost.
- For tightly coupled system consider a multiprocessor dedicated to running a single chess program in parallel.
- Each CPU is assigned a board to evaluate, and it spends its time examining that board and all the boards that can be generated from it.
- When the evaluation is finished, the CPU reports back the results and is given a new board to work on.
- The software for this system, both the application program and the operating system required to support it, is clearly much more tightly coupled than in our previous example.



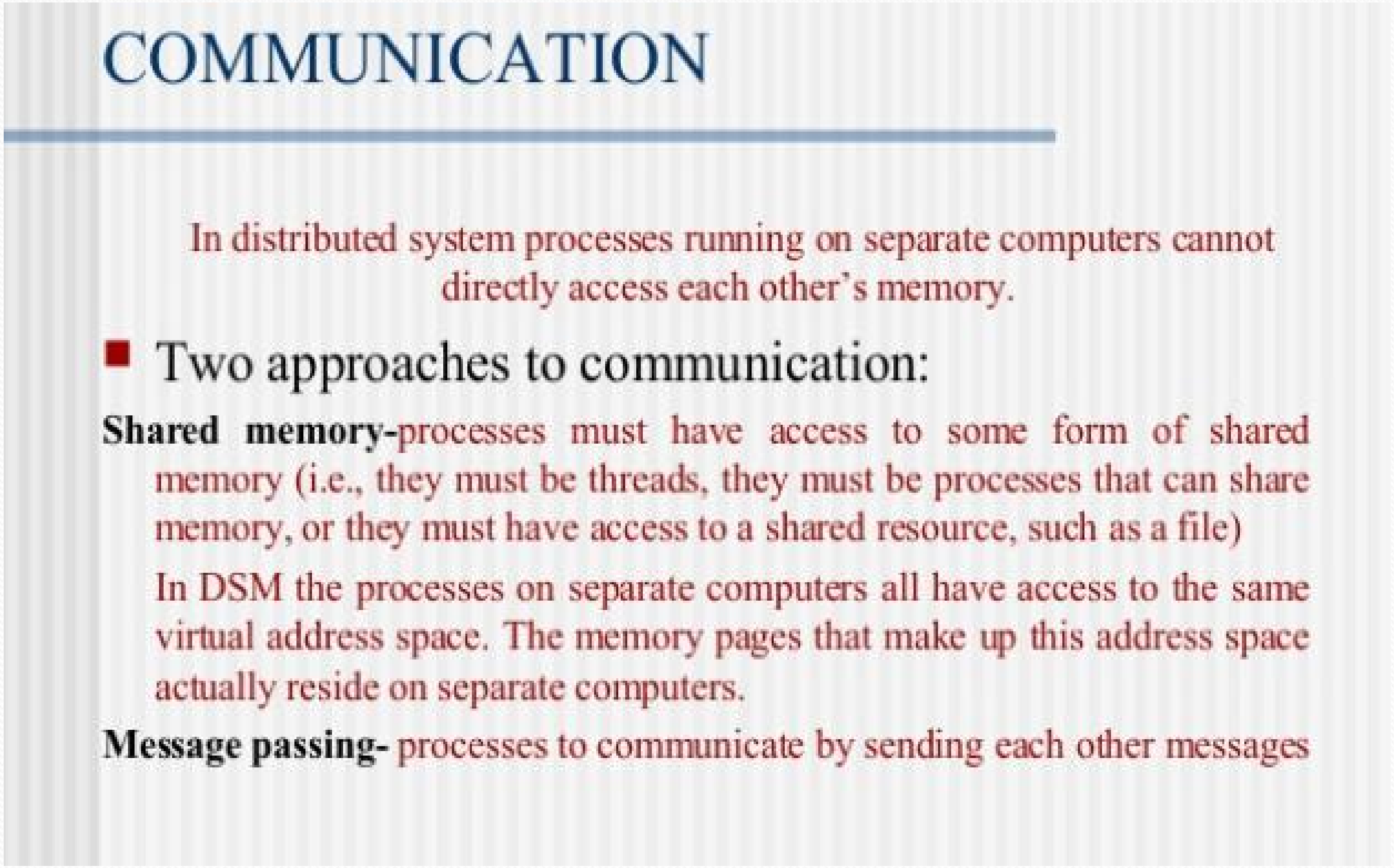
Tightly Coupled System

- Distributed Operating system is a tightly coupled software on loosely coupled hardware.
- The goal of such a system is to create the illusion in the minds of the users that the entire network of computers is a single timesharing system, rather than a collection of distinct machines.

Characteristics:

- There is a single, global inter process communication mechanism so that any process can talk to any other process.
- There is a global protection scheme.
- Process management is also uniform everywhere.
- How processes are created, destroyed, started, and stopped does not vary from machine to machine.
- The file system looks same from everywhere.
- Every file is visible at every location, subject to protection and security constraints.
- Each kernel has considerable control over its own local resources.
- For example, if swapping or paging is used, the kernel on each CPU is the logical place to determine what to do swap or page.

Communication in Distributed System



COMMUNICATION

In distributed system processes running on separate computers cannot directly access each other's memory.

- Two approaches to communication:

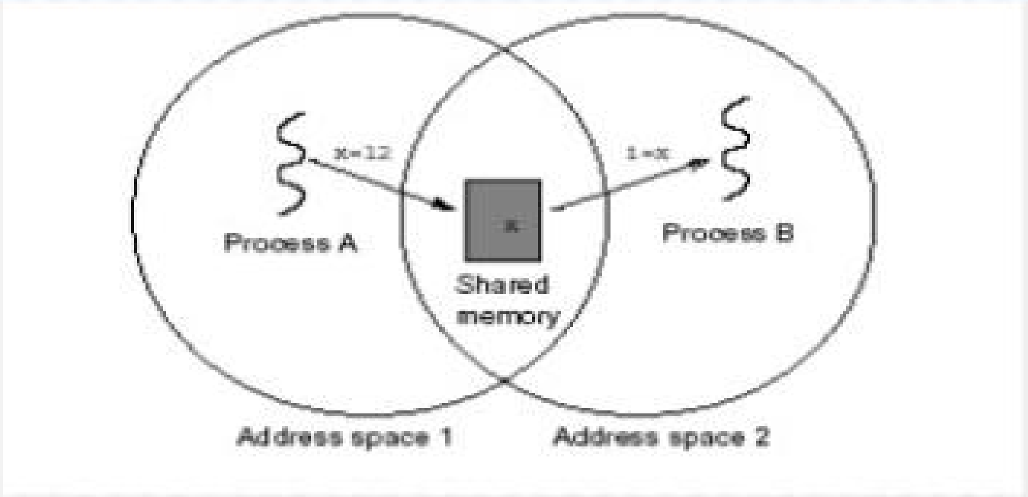
Shared memory-processes must have access to some form of shared memory (i.e., they must be threads, they must be processes that can share memory, or they must have access to a shared resource, such as a file)

In DSM the processes on separate computers all have access to the same virtual address space. The memory pages that make up this address space actually reside on separate computers.

Message passing- processes to communicate by sending each other messages

Communication in Distributed System

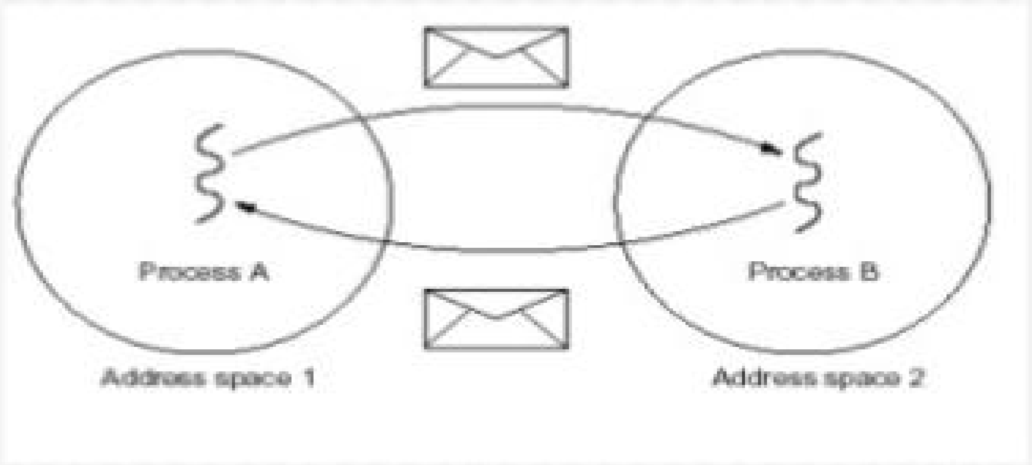
Shared Memory



- **Shared Memory:**
  - There is no way to physically share memory
  - Distributed Shared Memory

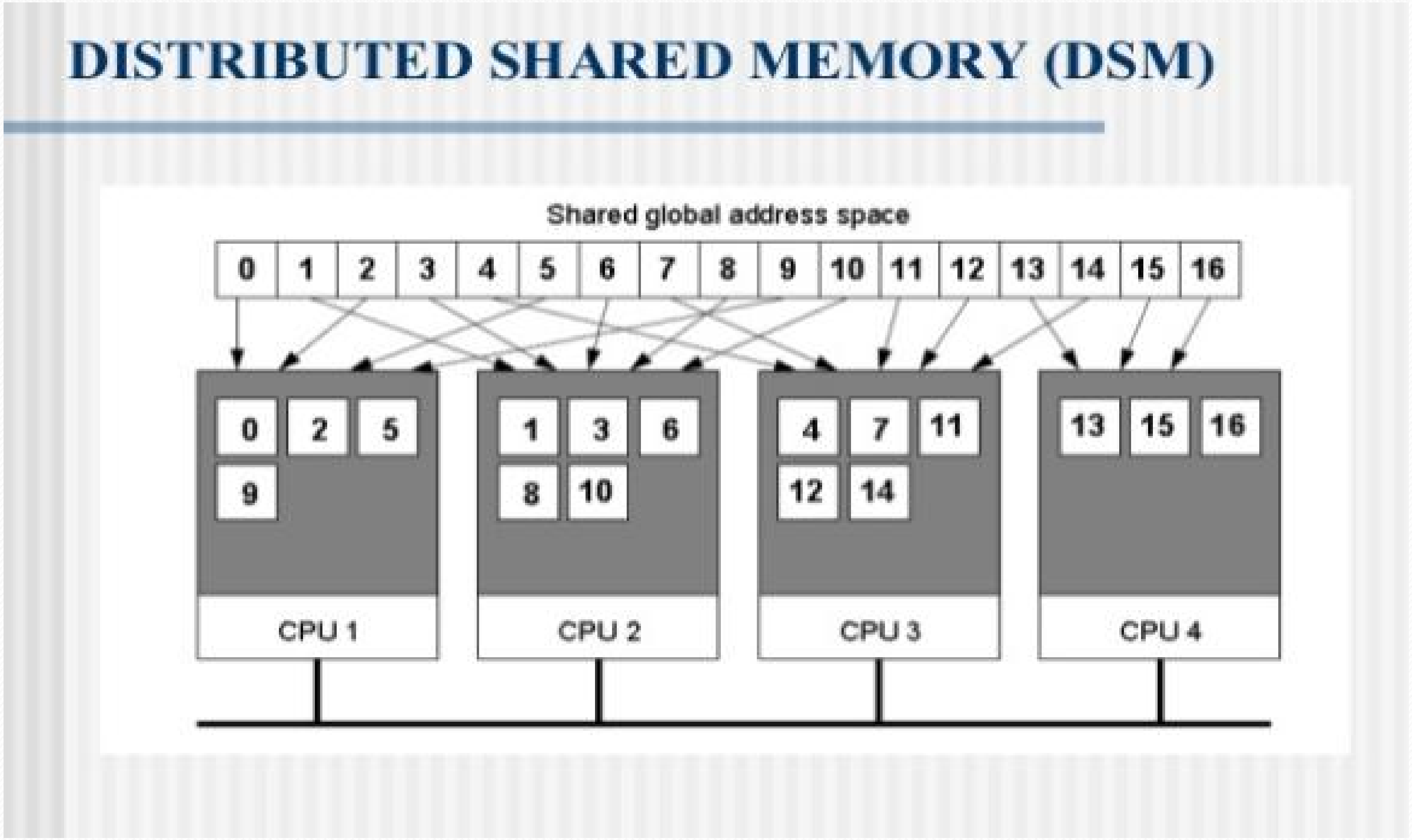
Communication in Distributed System

Message Passing

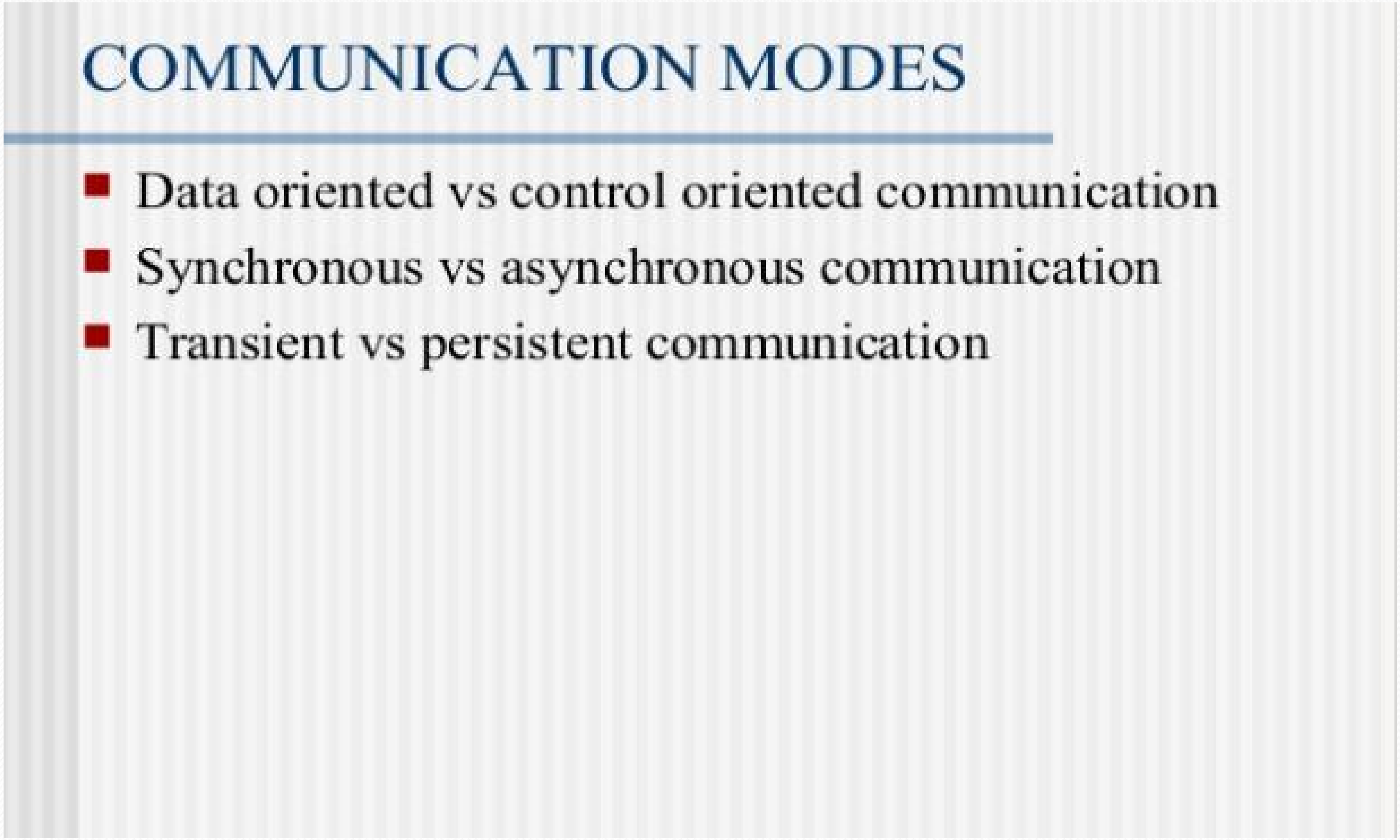


- **Message Passing:**
  - Over the network
  - Introduces higher chances of failure
  - Heterogeneity introduces possible incompatibilities

Communication in Distributed System



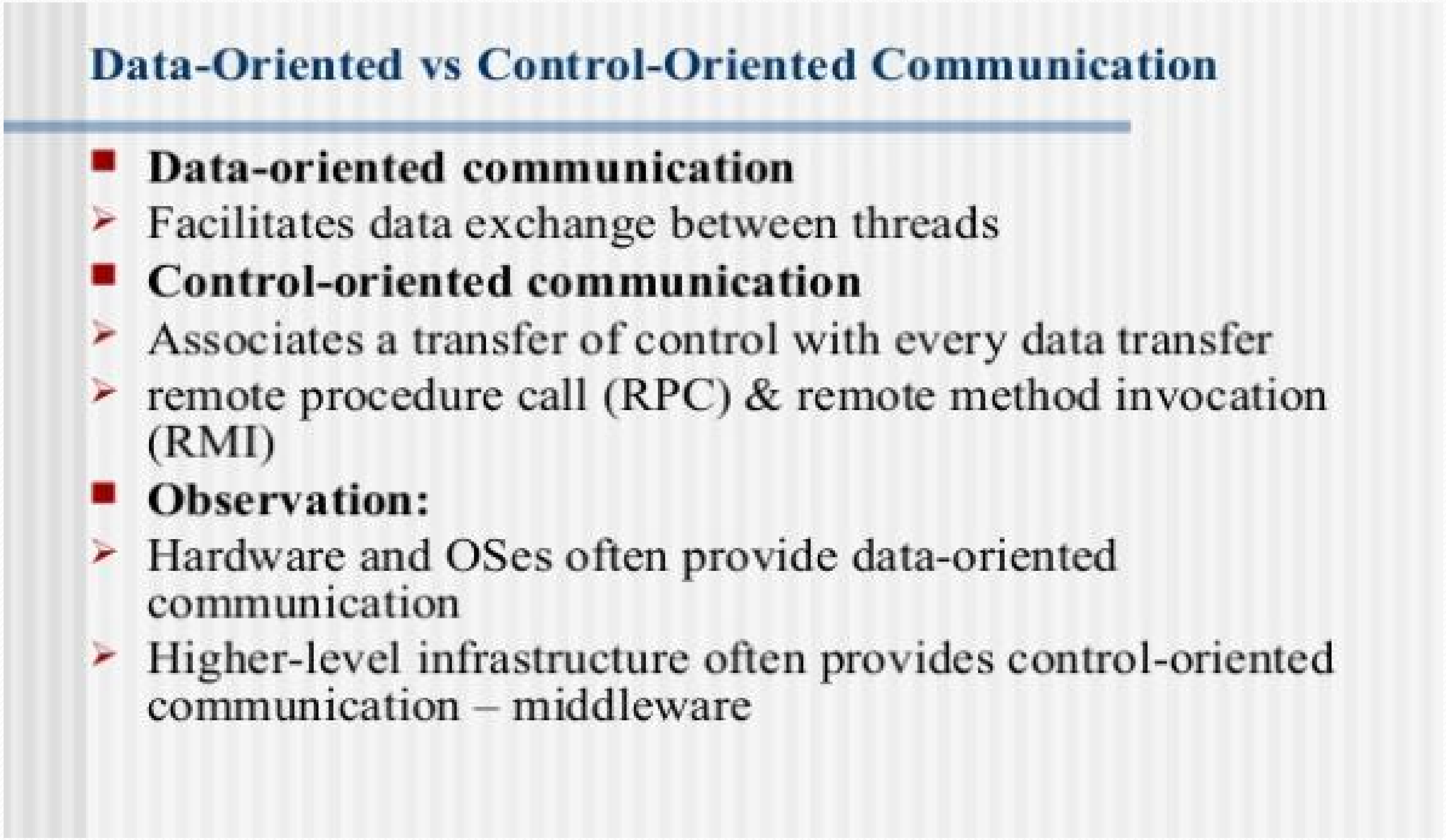
Communication in Distributed System



COMMUNICATION MODES

- Data oriented vs control oriented communication
- Synchronous vs asynchronous communication
- Transient vs persistent communication

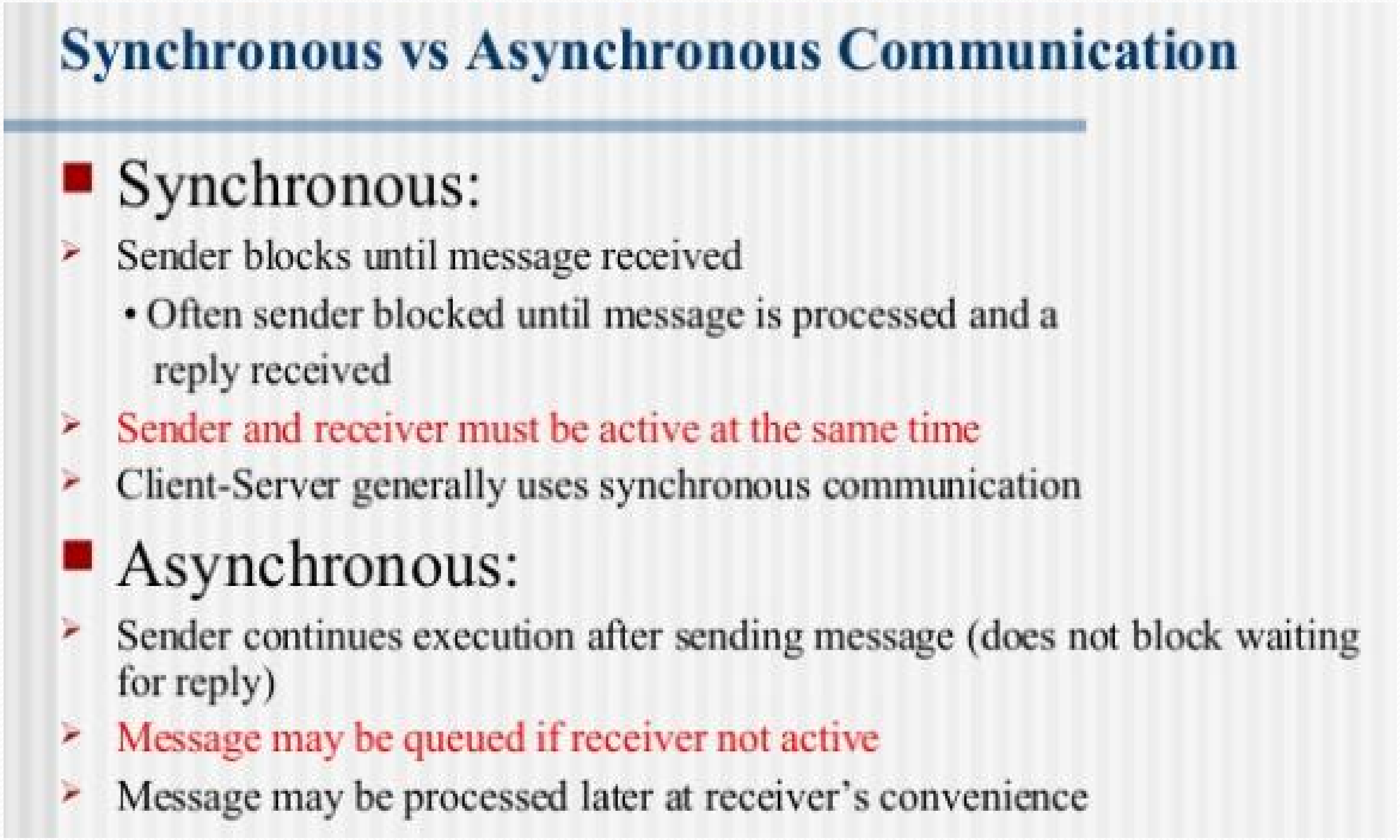
Communication in Distributed System



Data-Oriented vs Control-Oriented Communication

- **Data-oriented communication**
  - Facilitates data exchange between threads
- **Control-oriented communication**
  - Associates a transfer of control with every data transfer
  - remote procedure call (RPC) & remote method invocation (RMI)
- **Observation:**
  - Hardware and OSes often provide data-oriented communication
  - Higher-level infrastructure often provides control-oriented communication – middleware

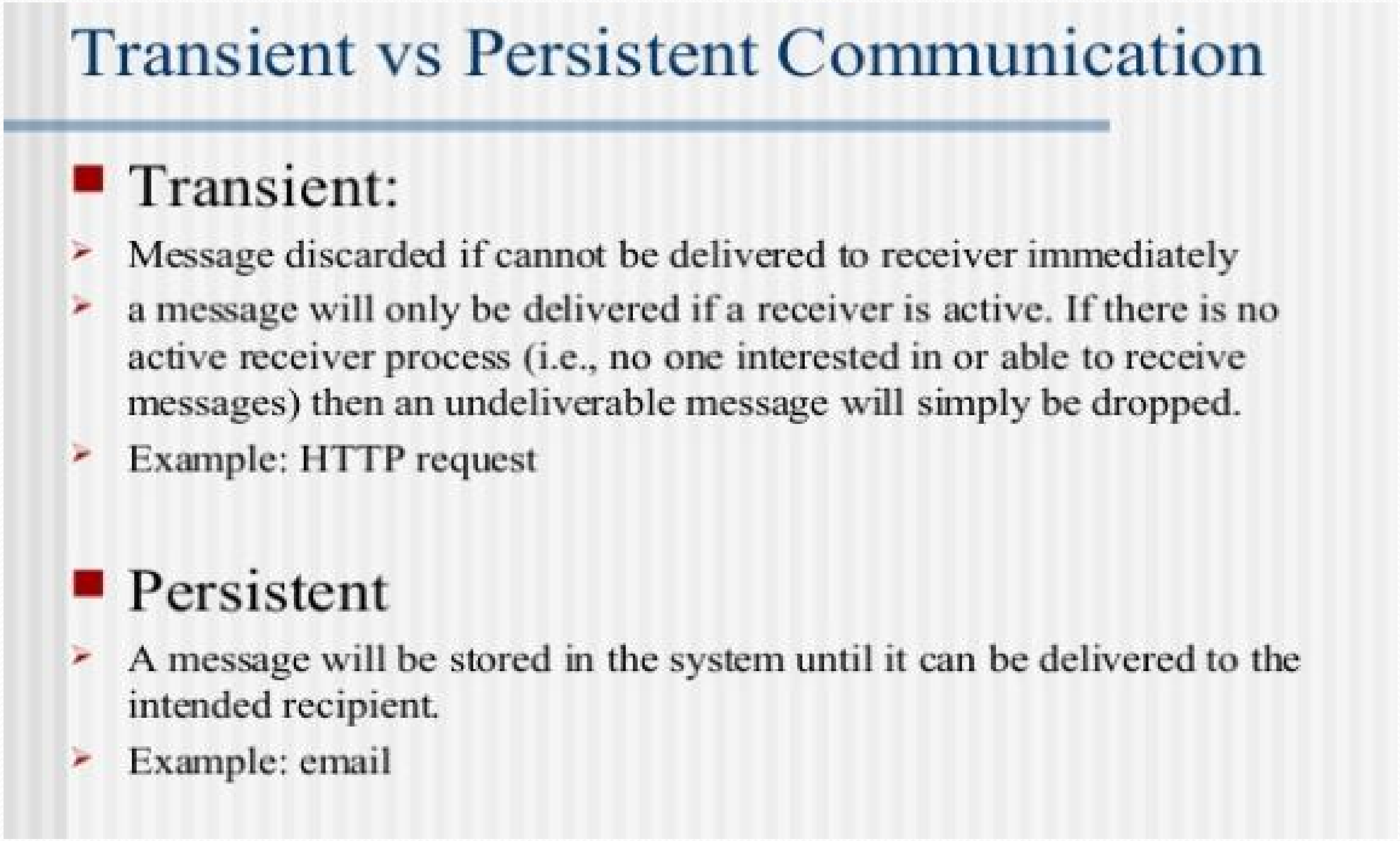
Communication in Distributed System



Synchronous vs Asynchronous Communication

- **Synchronous:**
  - ➢ Sender blocks until message received
    - Often sender blocked until message is processed and a reply received
  - ➢ Sender and receiver must be active at the same time
  - ➢ Client-Server generally uses synchronous communication
- **Asynchronous:**
  - ➢ Sender continues execution after sending message (does not block waiting for reply)
  - ➢ Message may be queued if receiver not active
  - ➢ Message may be processed later at receiver's convenience

Communication in Distributed System

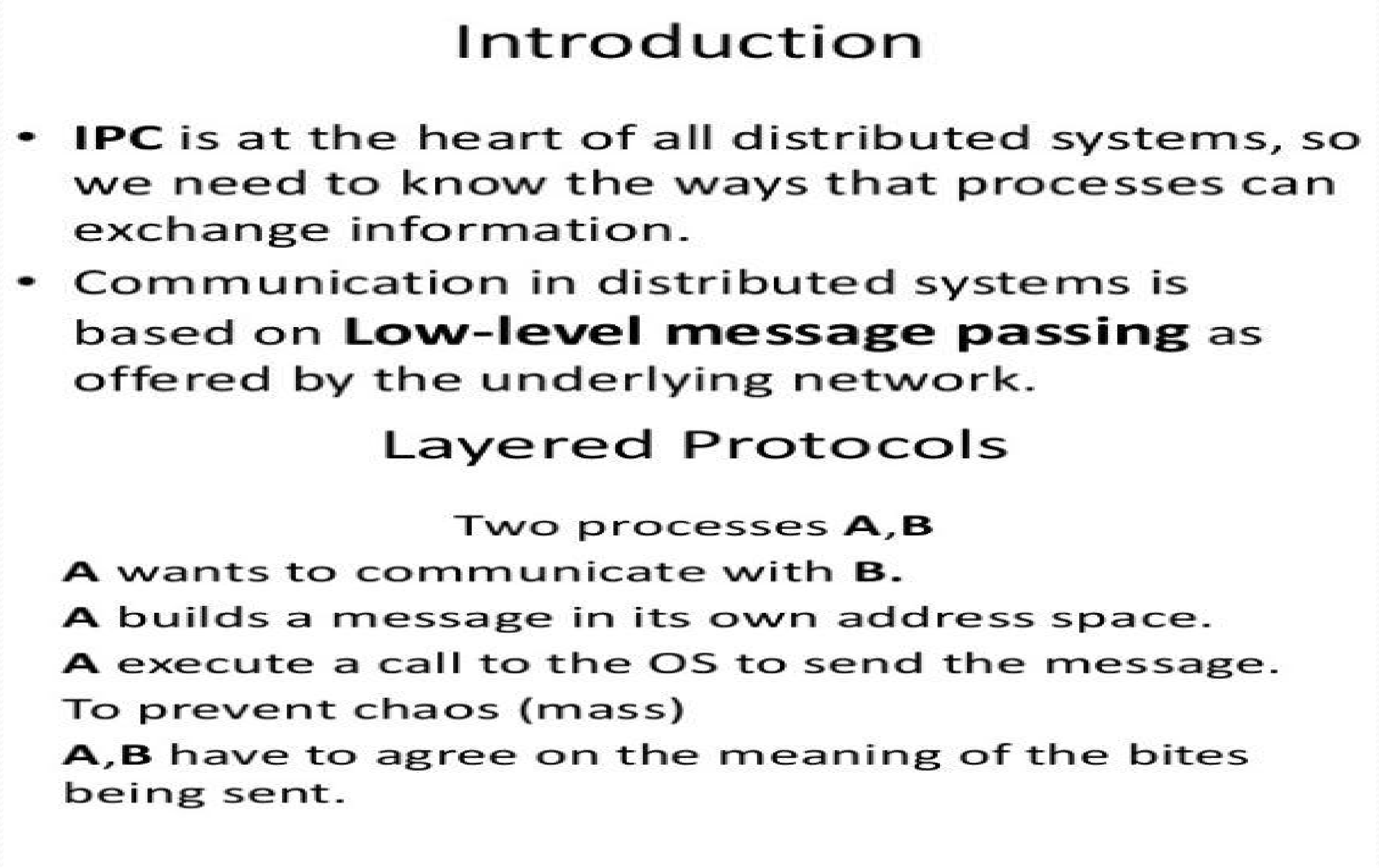


Transient vs Persistent Communication

- **Transient:**
  - Message discarded if cannot be delivered to receiver immediately
  - a message will only be delivered if a receiver is active. If there is no active receiver process (i.e., no one interested in or able to receive messages) then an undeliverable message will simply be dropped.
  - Example: HTTP request
- **Persistent**
  - A message will be stored in the system until it can be delivered to the intended recipient.
  - Example: email

Communication in Distributed System

IPC (Inter Processs Communication)



Introduction

- **IPC** is at the heart of all distributed systems, so we need to know the ways that processes can exchange information.
- Communication in distributed systems is based on **Low-level message passing** as offered by the underlying network.

Layered Protocols

Two processes **A,B**

A wants to communicate with **B.**

A builds a message in its own address space.

A execute a call to the OS to send the message.

To prevent chaos (mass)

A,B have to agree on the meaning of the bites being sent.

Communication in Distributed System

- The International Standards Organization (ISO) developed a **reference model** that clearly identifies the various levels involved **Open Systems Interconnection (OSI).**
- It provides a description of how network hardware and software work together in a layered fashion to make communications possible.

Communication in Distributed System

- Communication divided into **7 layers.**
- Each layer has a responsibility to perform specific tasks but all of the layers are needed for a message to reach its destination.
- Message **send** (downwards) Message **received** (upwards).

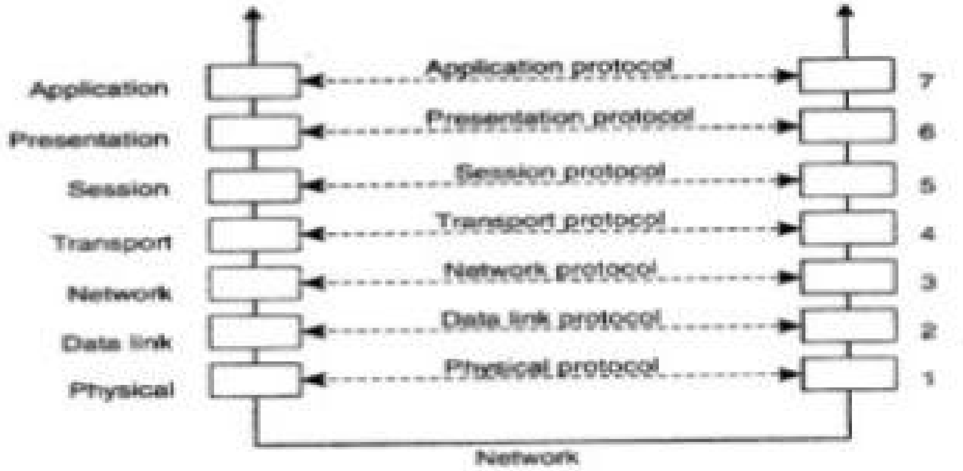
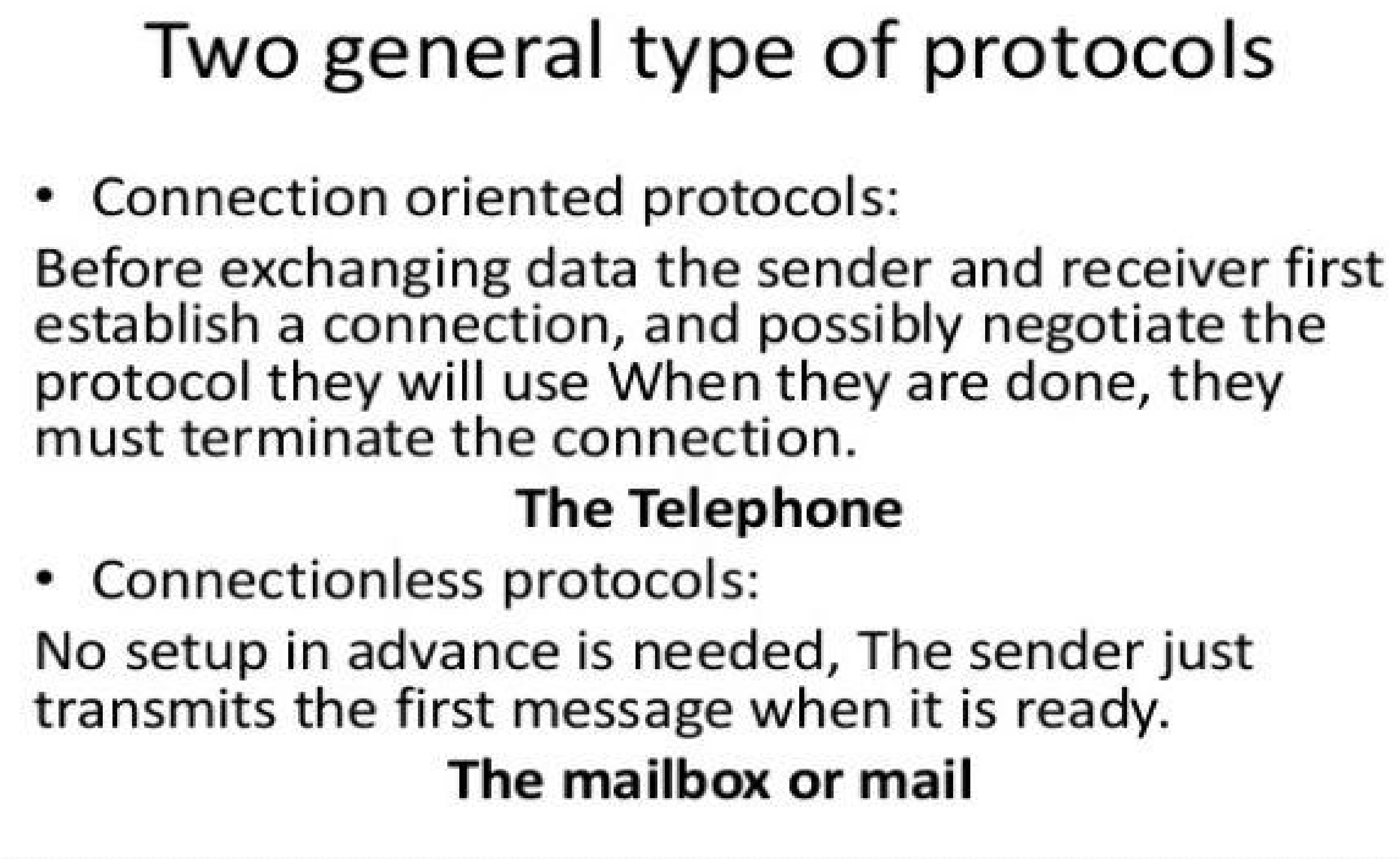


Figure 4-1. Layers, interfaces, and protocols in the OSI model.

Communication in Distributed System



Two general type of protocols

- Connection oriented protocols:

Before exchanging data the sender and receiver first establish a connection, and possibly negotiate the protocol they will use When they are done, they must terminate the connection.

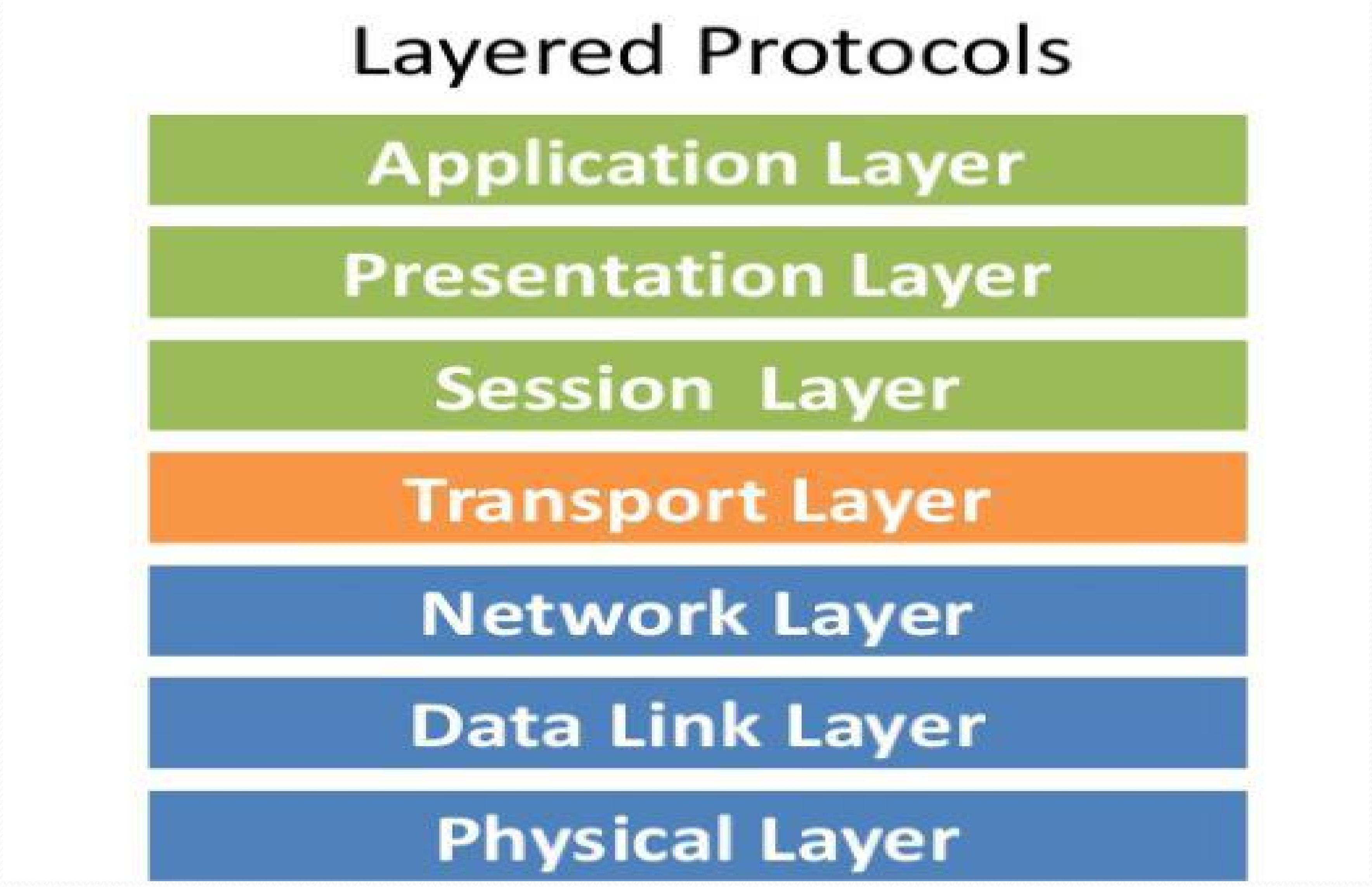
The Telephone

- Connectionless protocols:

No setup in advance is needed, The sender just transmits the first message when it is ready.

The mailbox or mail

Communication in Distributed System



Communication in Distributed System

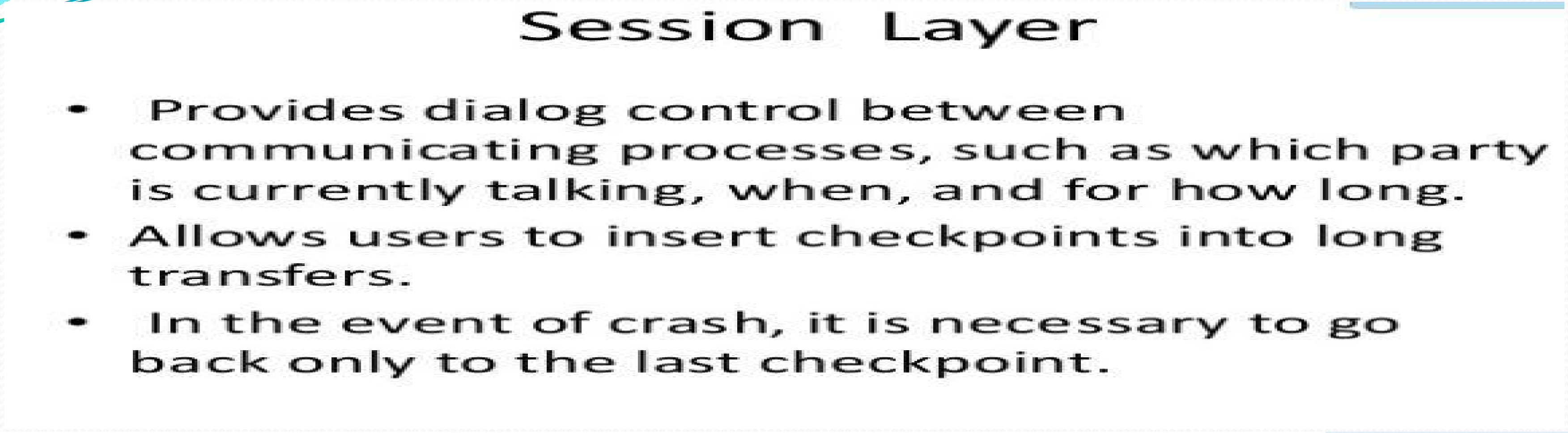
Application Layer

- It is the software that the end-user interacts with.
- It serves as a window through which application processes can access network services.

Presentation Layer

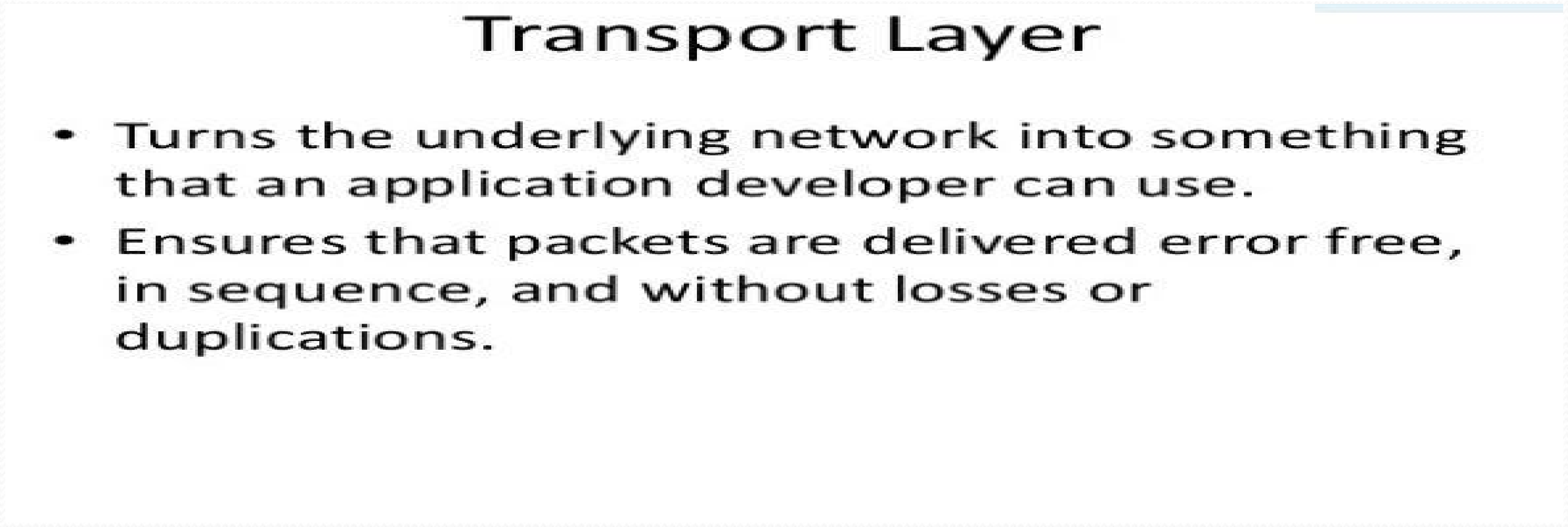
- Think of it as the network's translator.
- It is responsible for translating the data, encrypting the data, changing or converting the character set.
- Manages data compression to reduce the number of bits that need to be transmitted.

Communication in Distributed System



Session Layer

- Provides dialog control between communicating processes, such as which party is currently talking, when, and for how long.
- Allows users to insert checkpoints into long transfers.
- In the event of crash, it is necessary to go back only to the last checkpoint.



Transport Layer

- Turns the underlying network into something that an application developer can use.
- Ensures that packets are delivered error free, in sequence, and without losses or duplications.

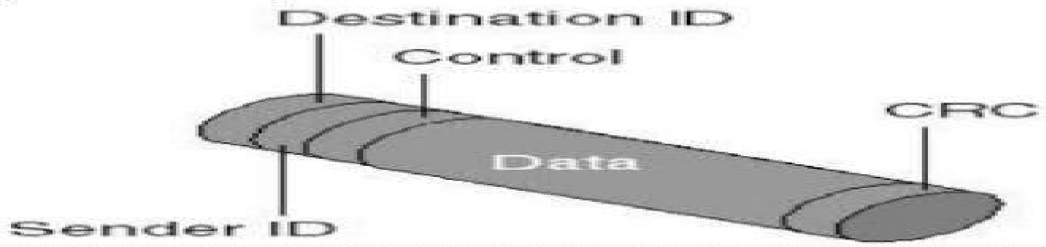
Communication in Distributed System

Network Layer

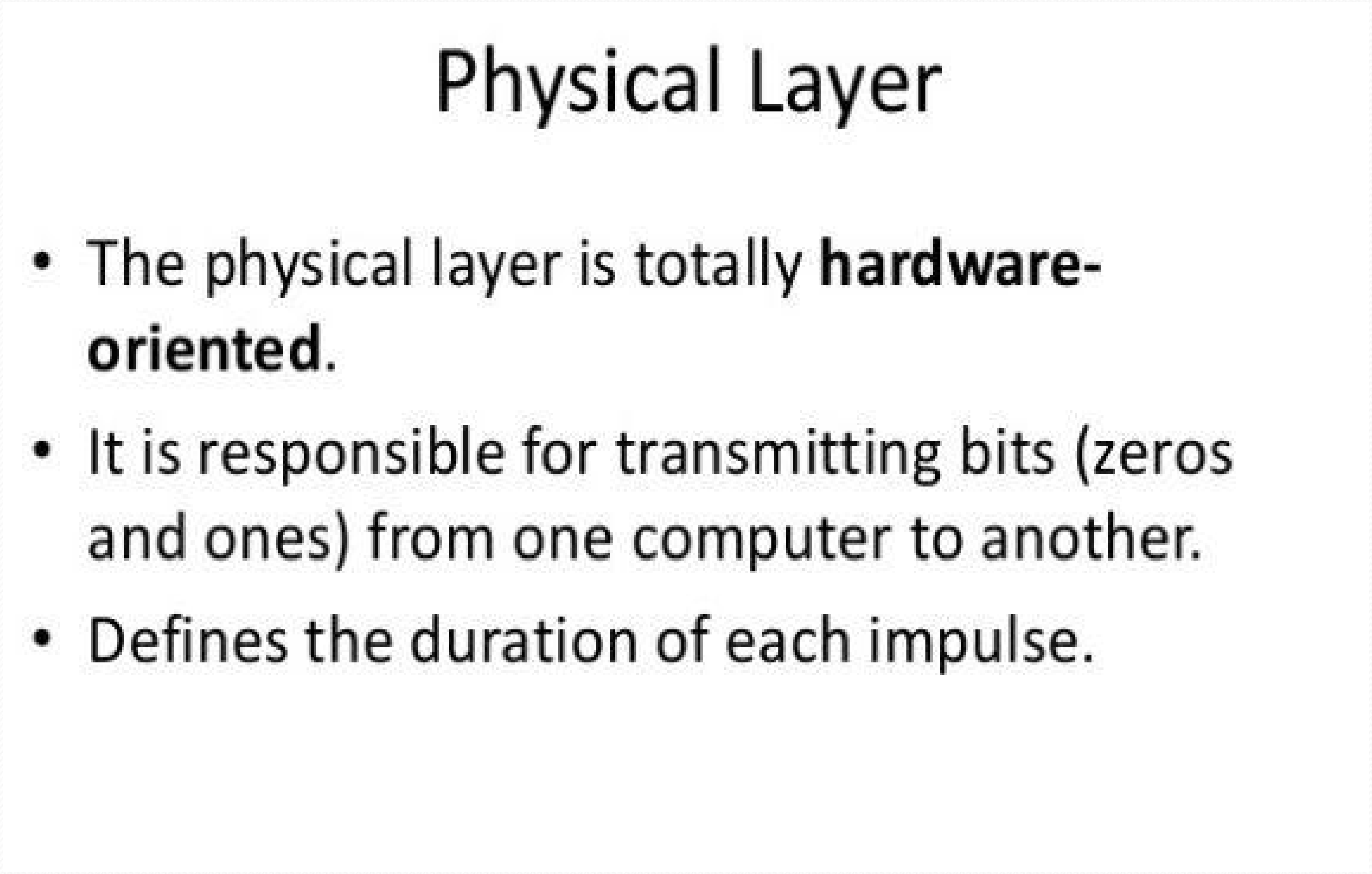
- Addresses the packets and translating logical addresses and names into **physical addresses**.
- Determines the route from the source to the destination computer (**Delay-Wise**).

Data Link Layer

- Groups bits into frames and sees that each frame is correctly received (checksum), If checksums differ, requests a retransmission.
- Sends data frames from the network layer to the physical layer



Communication in Distributed System

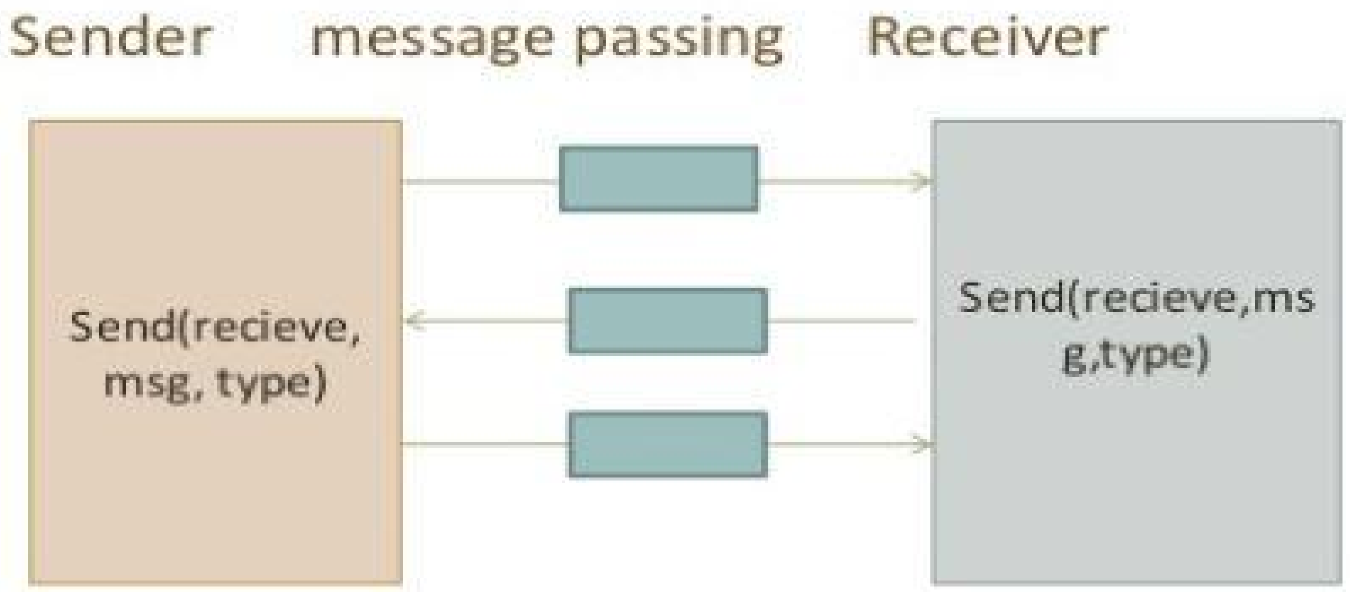


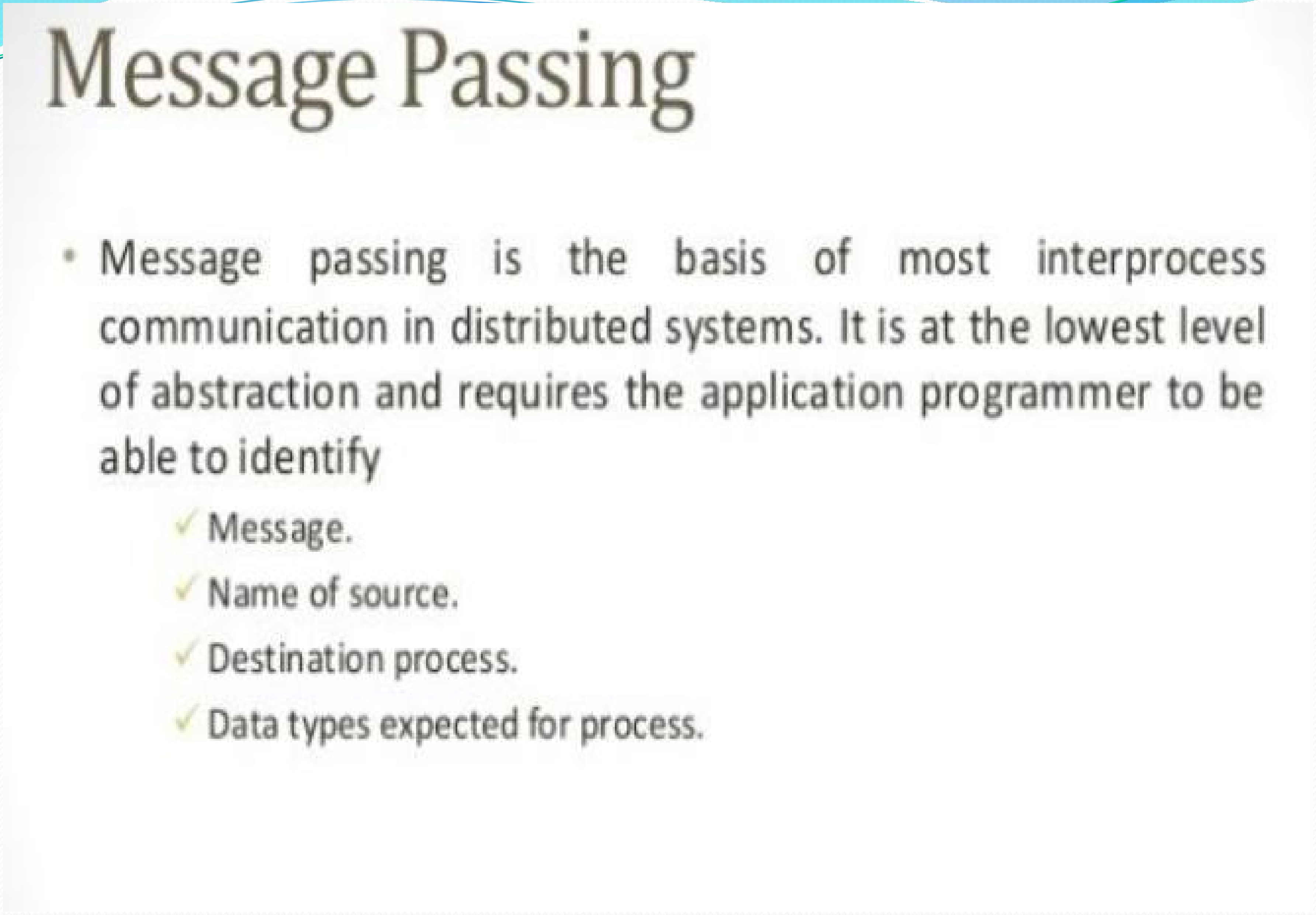
Physical Layer

- The physical layer is totally **hardware-oriented**.
- It is responsible for transmitting bits (zeros and ones) from one computer to another.
- Defines the duration of each impulse.

Message Passing - Introduction

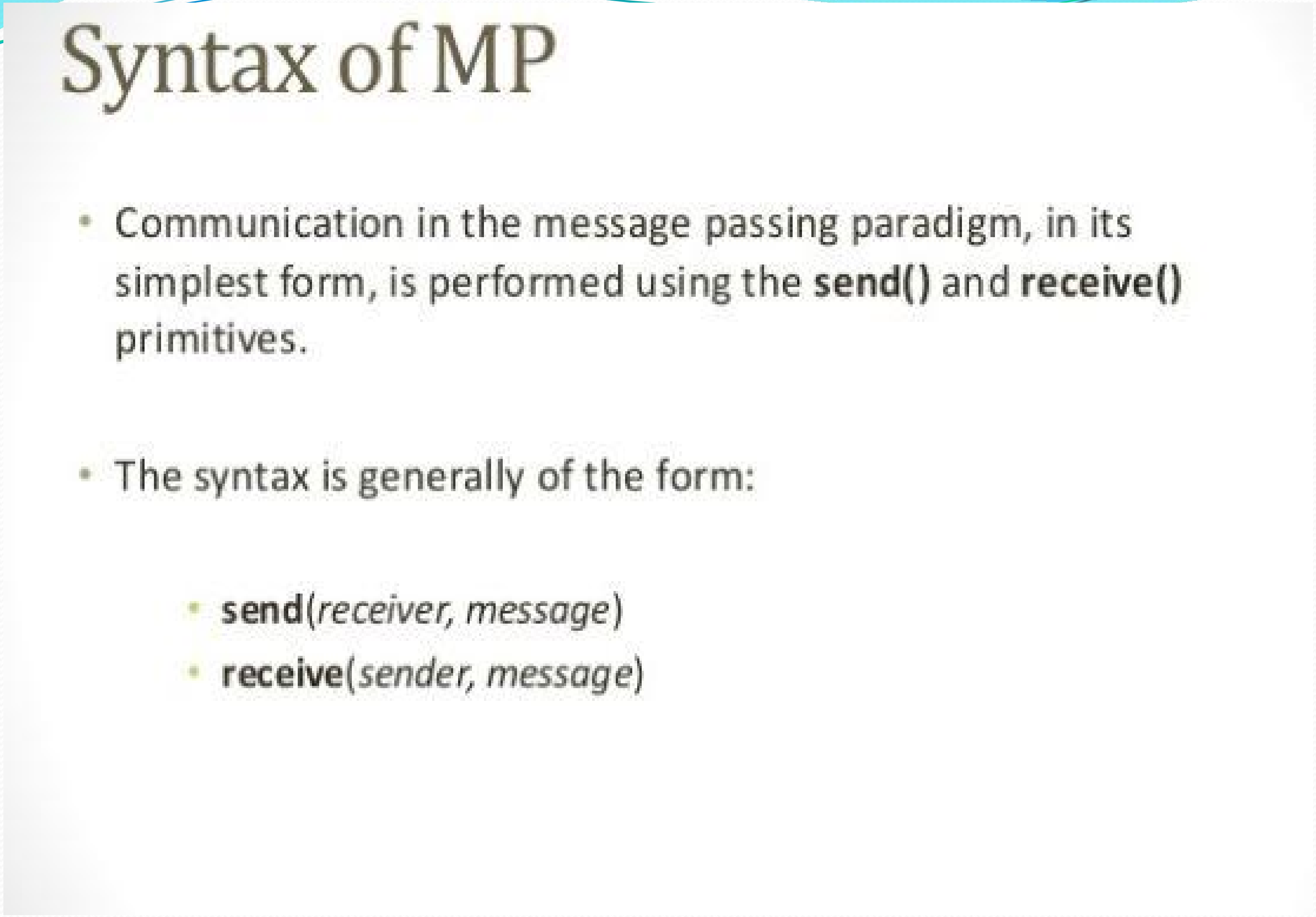
- It requires the programmer to know
  - ✓ Message
  - ✓ Name of source
  - ✓ Destination process.





Message Passing

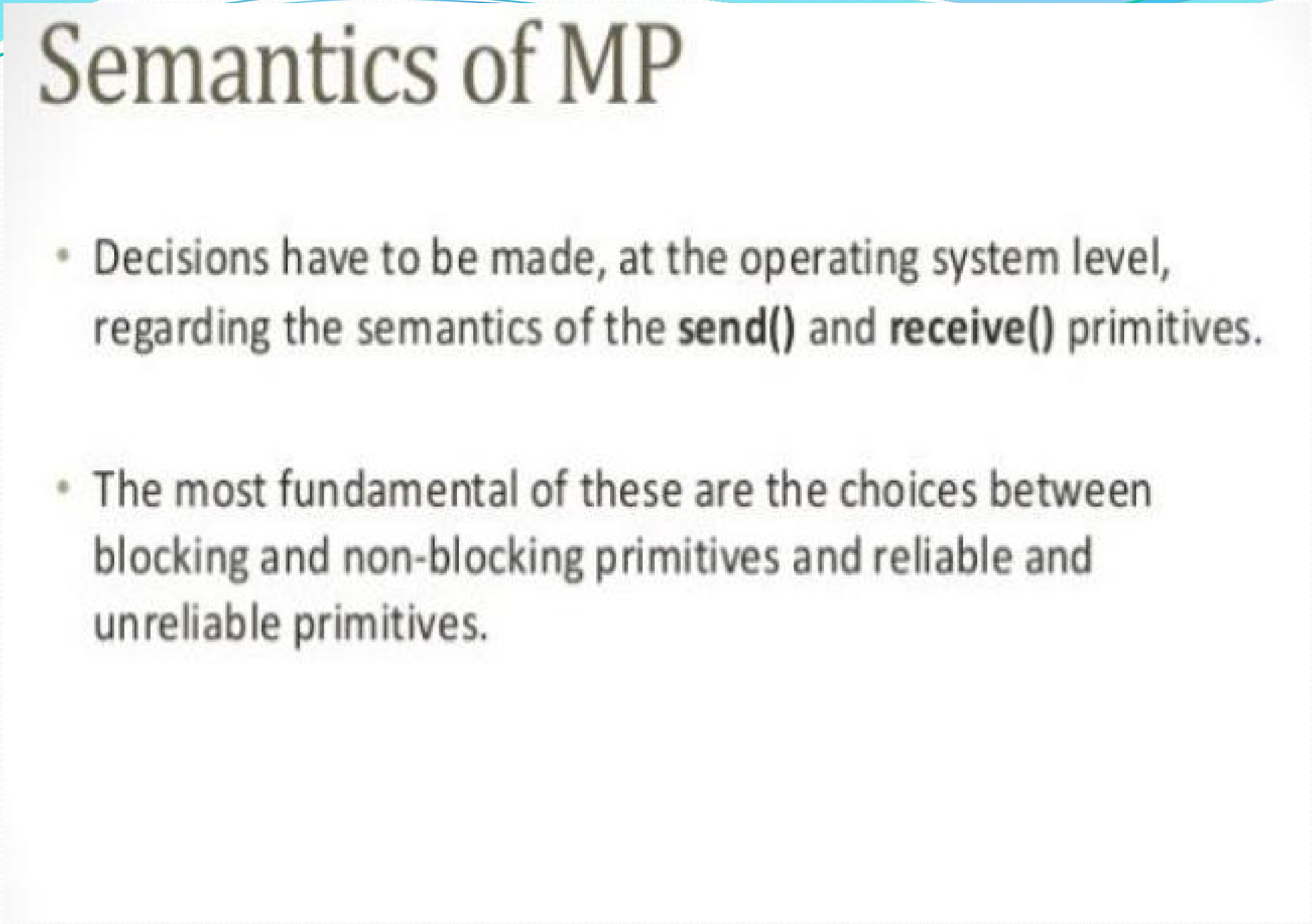
- Message passing is the basis of most interprocess communication in distributed systems. It is at the lowest level of abstraction and requires the application programmer to be able to identify
  - ✓ Message.
  - ✓ Name of source.
  - ✓ Destination process.
  - ✓ Data types expected for process.



Syntax of MP

- Communication in the message passing paradigm, in its simplest form, is performed using the **send()** and **receive()** primitives.

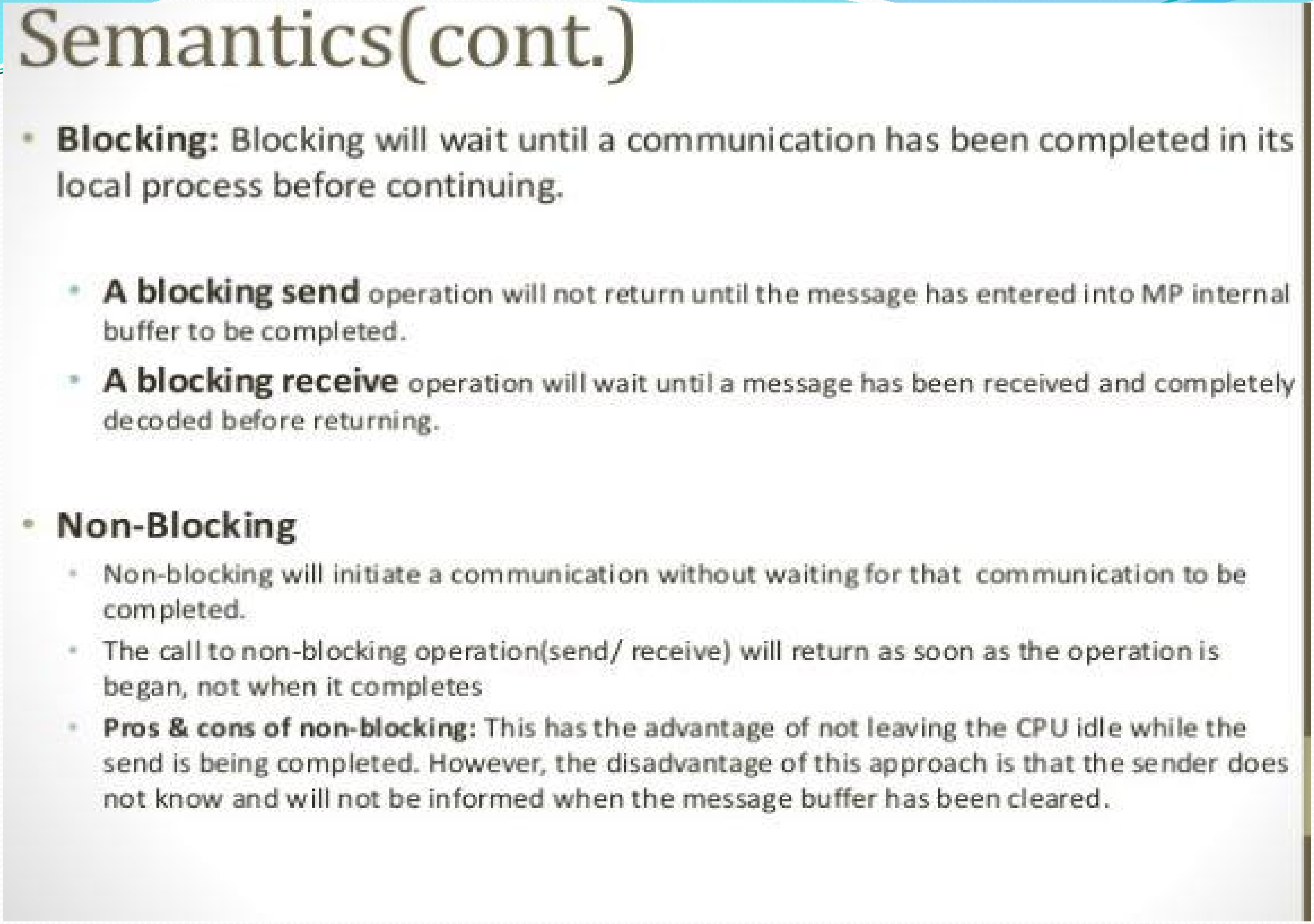
- The syntax is generally of the form:
  - **send**(*receiver, message*)
  - **receive**(*sender, message*)



Semantics of MP

- Decisions have to be made, at the operating system level, regarding the semantics of the **send()** and **receive()** primitives.

- The most fundamental of these are the choices between blocking and non-blocking primitives and reliable and unreliable primitives.



Semantics(cont.)

- **Blocking:** Blocking will wait until a communication has been completed in its local process before continuing.
  - **A blocking send** operation will not return until the message has entered into MP internal buffer to be completed.
  - **A blocking receive** operation will wait until a message has been received and completely decoded before returning.
- **Non-Blocking**
  - Non-blocking will initiate a communication without waiting for that communication to be completed.
  - The call to non-blocking operation(send/ receive) will return as soon as the operation is began, not when it completes
  - **Pros & cons of non-blocking:** This has the advantage of not leaving the CPU idle while the send is being completed. However, the disadvantage of this approach is that the sender does not know and will not be informed when the message buffer has been cleared.

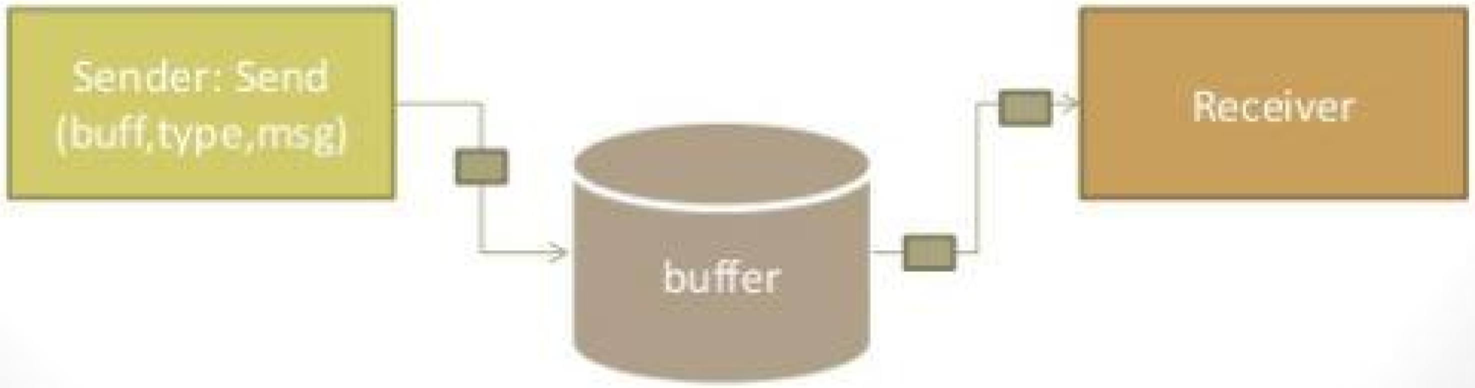
Semantics(cont.)

- **Buffering**
  - **A unbuffered send & receive** operation means that the sending process sends the message directly to the receiving process rather than a message buffer. The address, *receiver,* in the **send()** is the address of the process and same as **recieve()**
  - There is a problem, in the unbuffered case, if the **send()** is called before the **receive()** because the address in the send does not refer to any existing process on the server machine.

sender process
receiver process

Semantics (Cont…)

- **A buffered send & receive**
  - operation will send address of buffer in **send()** and **recieve()** as destination parameter to communicate with other process.
  - Buffered messages save in buffer until the server process is ready to process them.



Semantics (Cont.) - Reliability

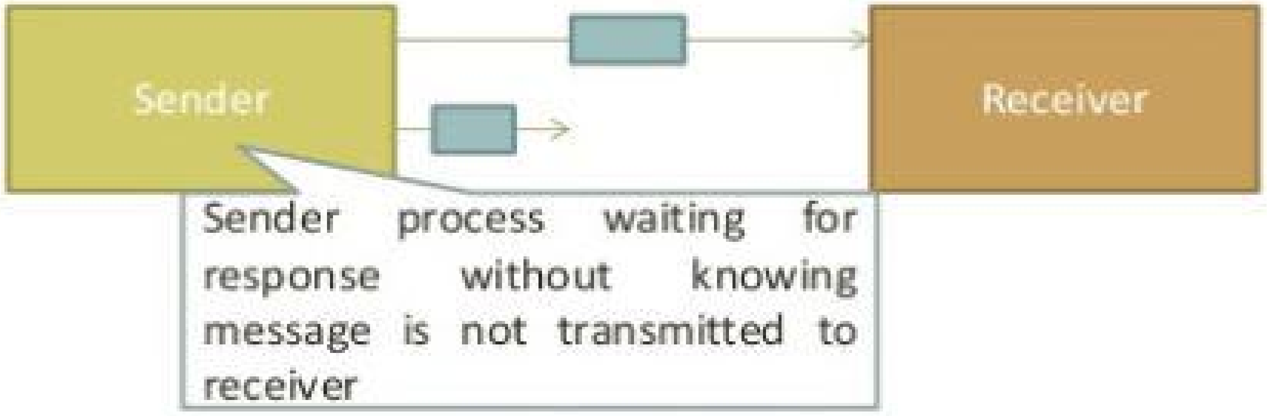

Sender
Receiver
Sender process waiting for response without knowing message is not transmitted to receiver

Fig: A Unreliable Send

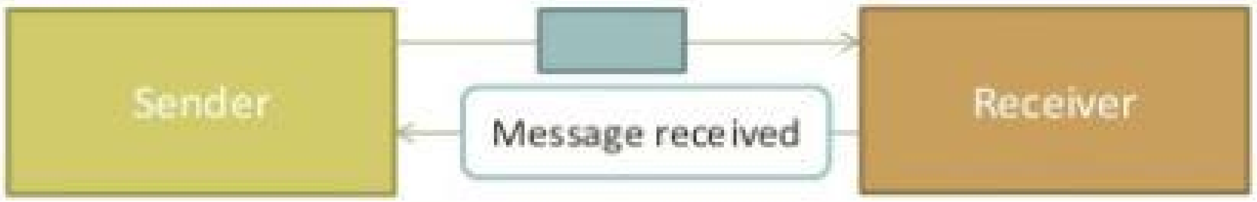
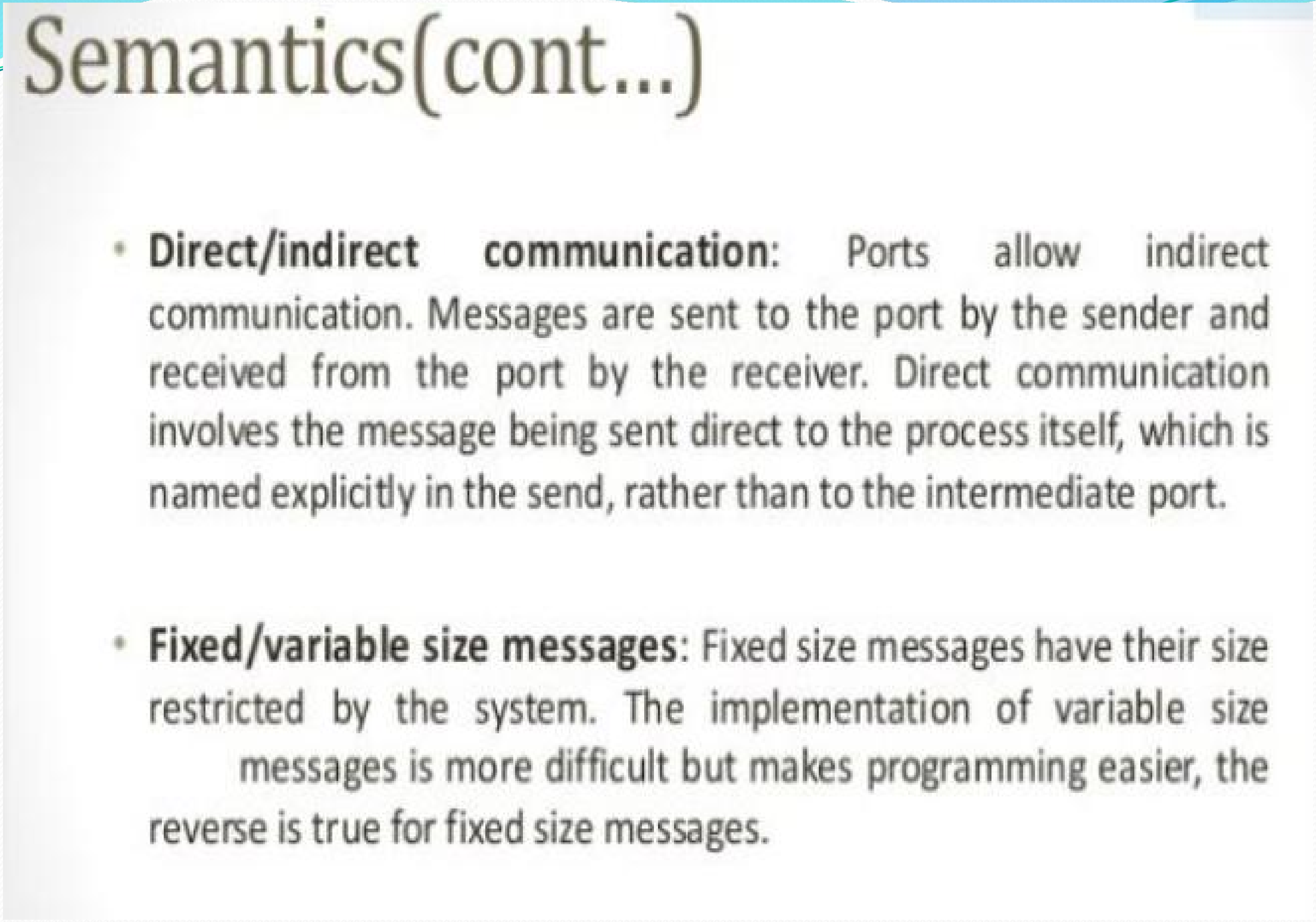


Fig: A Reliable Send



Semantics(cont...)

- **Direct/indirect communication:** Ports allow indirect communication. Messages are sent to the port by the sender and received from the port by the receiver. Direct communication involves the message being sent direct to the process itself, which is named explicitly in the send, rather than to the intermediate port.

- **Fixed/variable size messages**: Fixed size messages have their size restricted by the system. The implementation of variable size messages is more difficult but makes programming easier, the reverse is true for fixed size messages.

Explain Remote procedure call (RPC) and its implementation.

- Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.
- A procedure call is also sometimes known as a function call or a subroutine call.
- RPC uses the client-server model.

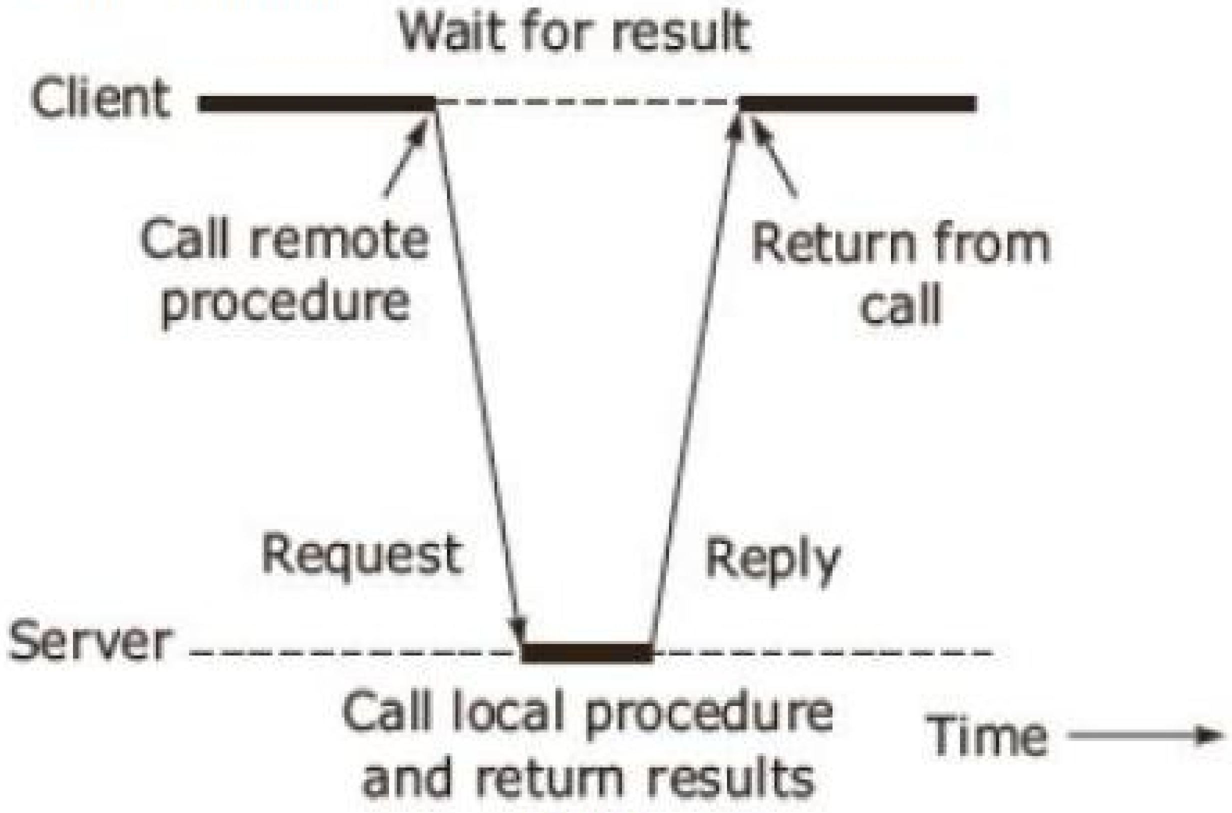
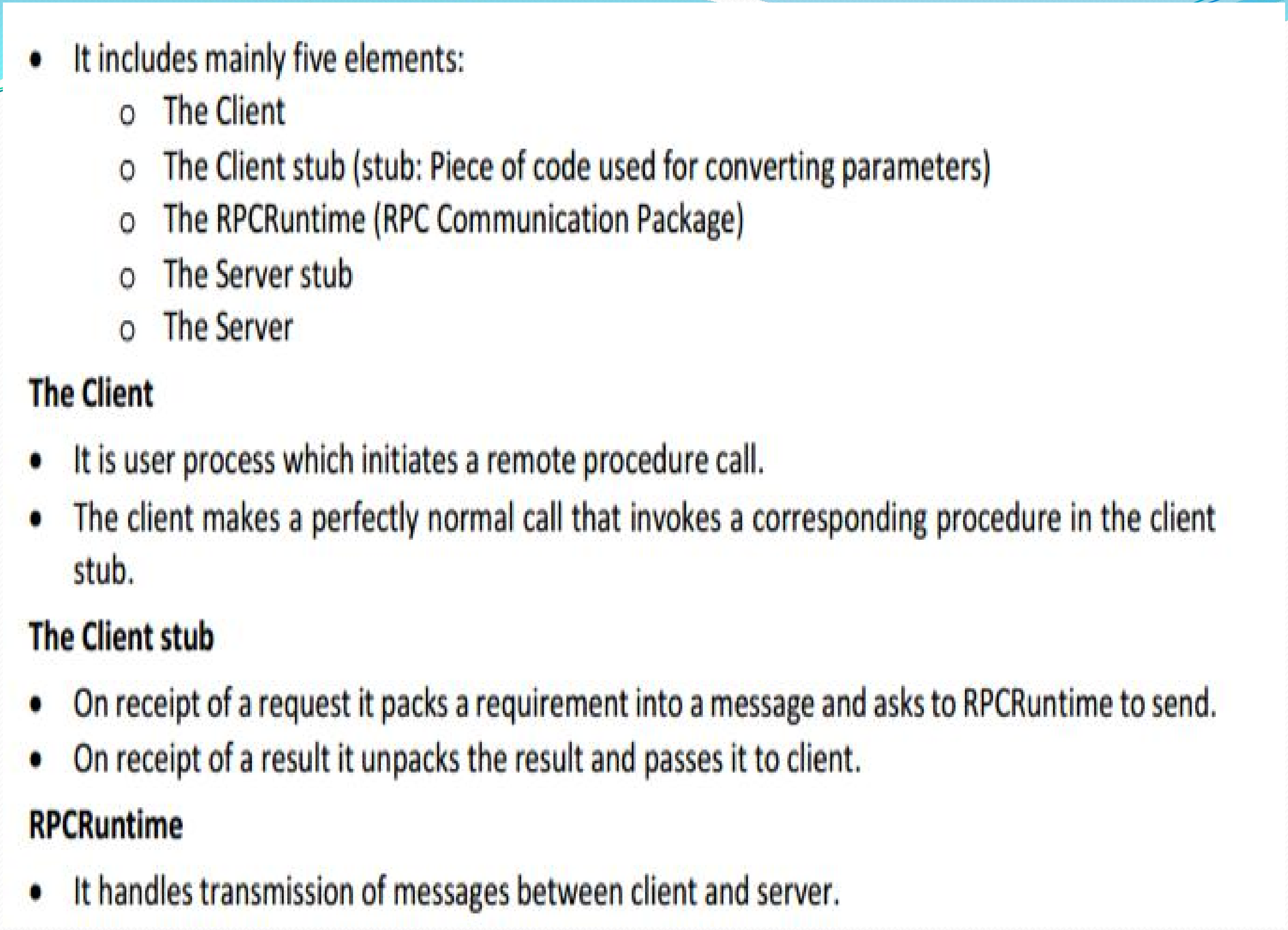


Figure: RPC Model



- It includes mainly five elements:
  - The Client
  - The Client stub (stub: Piece of code used for converting parameters)
  - The RPCRuntime (RPC Communication Package)
  - The Server stub
  - The Server

The Client

- It is user process which initiates a remote procedure call.
- The client makes a perfectly normal call that invokes a corresponding procedure in the client stub.

The Client stub

- On receipt of a request it packs a requirement into a message and asks to RPCRuntime to send.
- On receipt of a result it unpacks the result and passes it to client.

RPCRuntime

- It handles transmission of messages between client and server.

The Server stub

- It unpacks a call request and make a perfectly normal call to invoke the appropriate procedure in the server.
- On receipt of a result of procedure execution it packs the result and asks to RPCRuntime to send.

The Server

- It executes an appropriate procedure and returns the result from a server stub.

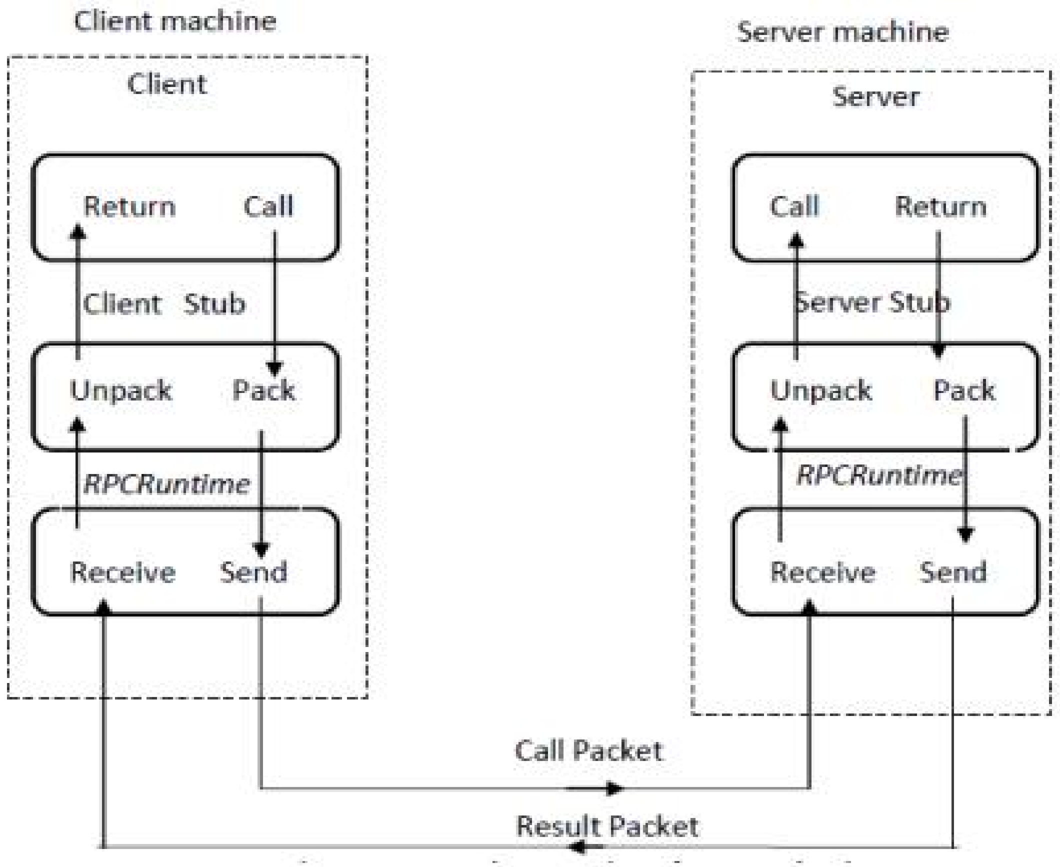
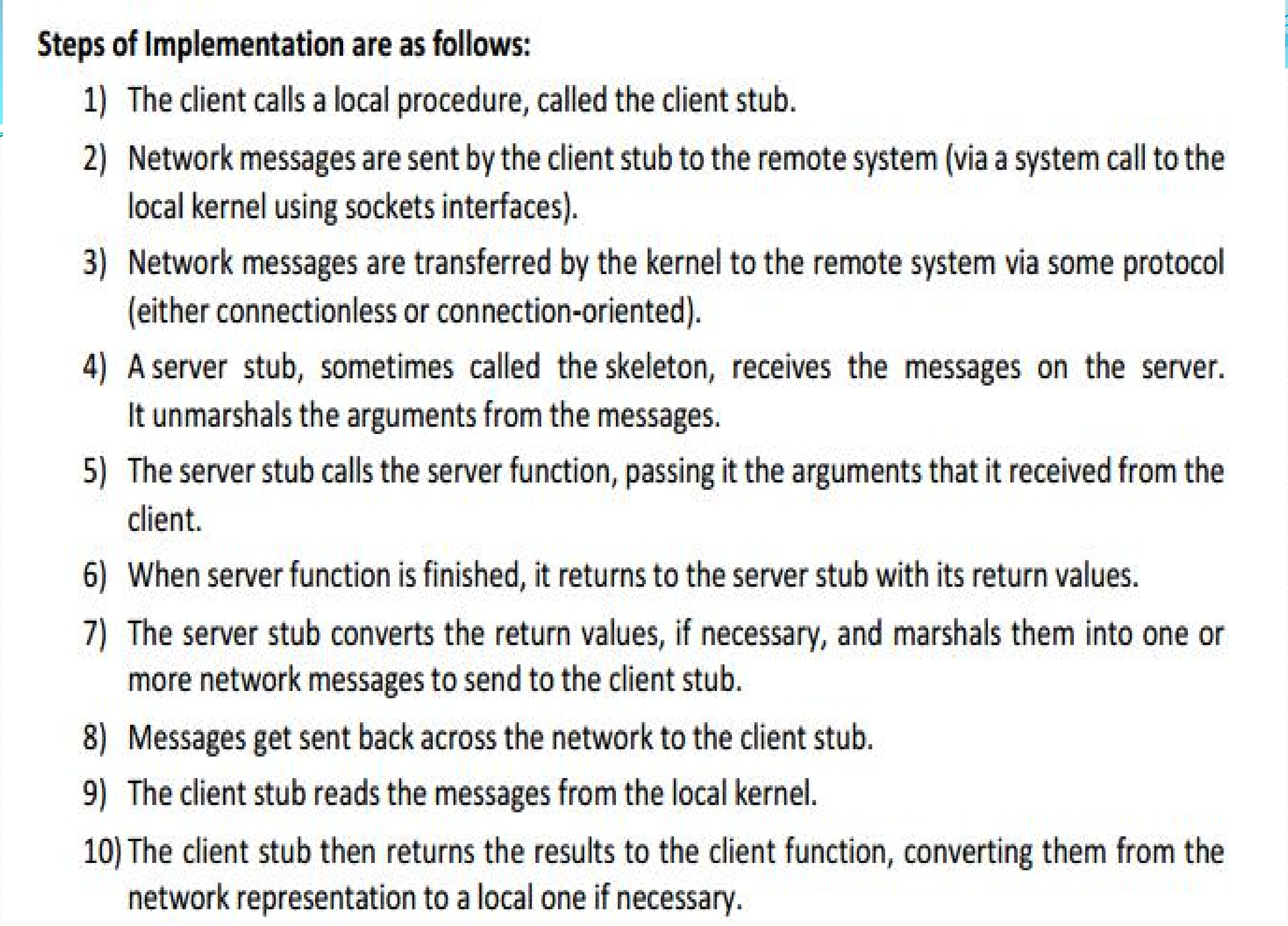


Figure: Implementation of RPC mechanism



Steps of Implementation are as follows:

1) The client calls a local procedure, called the client stub.
2) Network messages are sent by the client stub to the remote system (via a system call to the local kernel using sockets interfaces).
3) Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4) A server stub, sometimes called the skeleton, receives the messages on the server. It unmarshals the arguments from the messages.
5) The server stub calls the server function, passing it the arguments that it received from the client.
6) When server function is finished, it returns to the server stub with its return values.
7) The server stub converts the return values, if necessary, and marshals them into one or more network messages to send to the client stub.
8) Messages get sent back across the network to the client stub.
9) The client stub reads the messages from the local kernel.
10) The client stub then returns the results to the client function, converting them from the network representation to a local one if necessary.

For more: see the notes provided in class

Clock Synchronization

- See the pdf provided in class

END