

# **Communication: Distributed System**

# Building A Distributed System

- Two questions:
  1. Where to place the hardware?
  2. Where to place the software?

# Architecture

- The software architecture of distributed systems deals with how software components are organized and how they work together, i.e., communicate with each other.
- Typical software architectures include the layered, object-oriented, data-centred, and event-based architectures.

## **Logical organization of software components:**

- ☐ Layered
- ☐ Object-oriented
- ☐ Data-centered
- ☐ Event-based

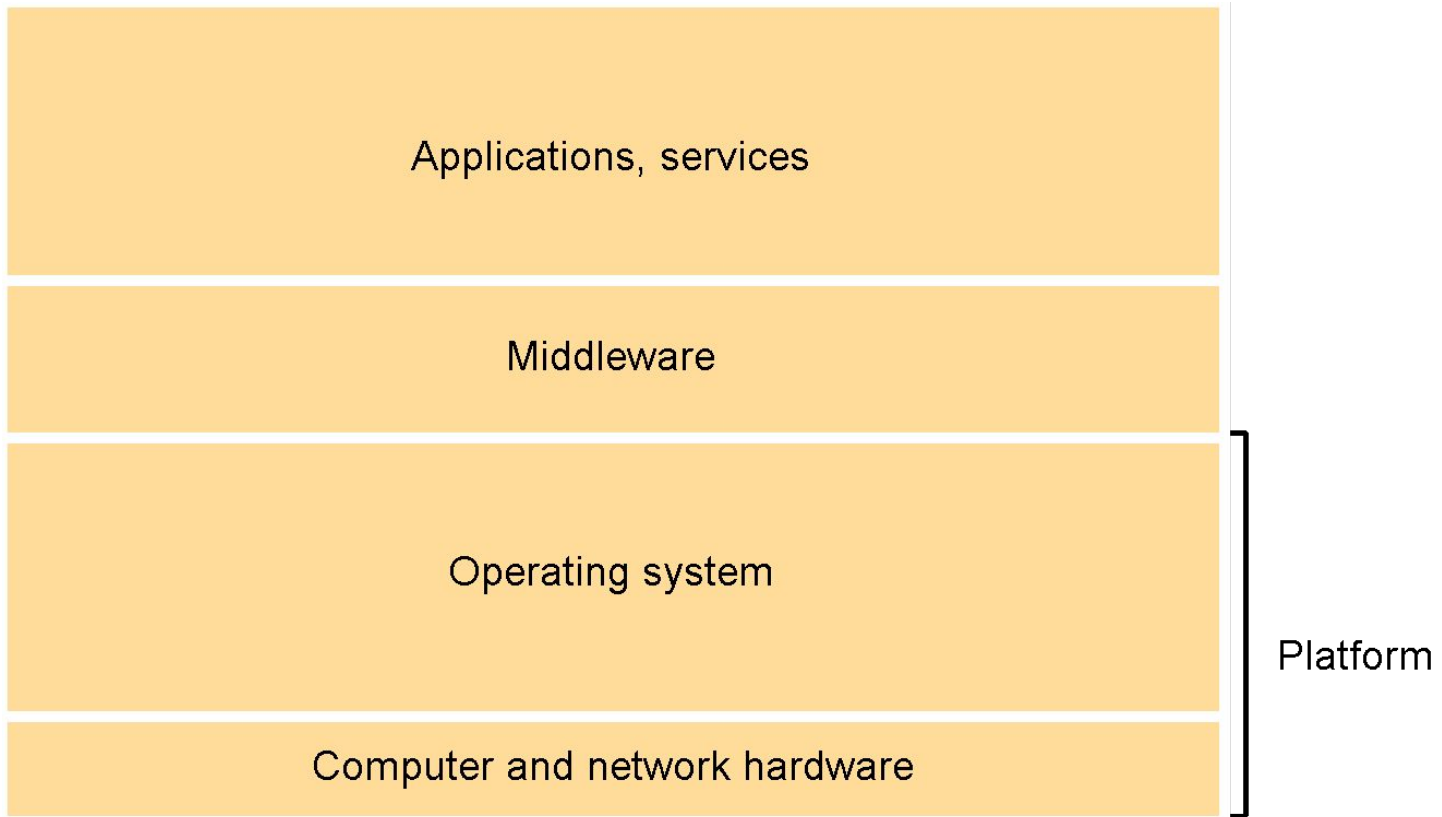
## **System Architecture:**

- ☐ placement of machines
- ☐ placement of software on machines

## **There is no single best architecture:**

- ☐ The best architecture for a particular system depends on the application requirements and the environment.

# Layers in Distributed Systems



- The key difference between a distributed system and a uniprocessor system is the interprocess communication.
- In a uniprocessor system, interprocess communication assumes the existence of shared memory.
- A typical example is the producer-consumer problem.
- One process writes to a buffer - another process reads from it
- The most basic form of synchronization, the semaphore requires one word (the semaphore variable) to be shared.

- In a distributed system, there's no shared memory, so the entire nature of interprocess communication must be completely rethought from scratch.
- All communication in distributed system is based on message passing.

For ex. Process 1 wants to communicate with process 2

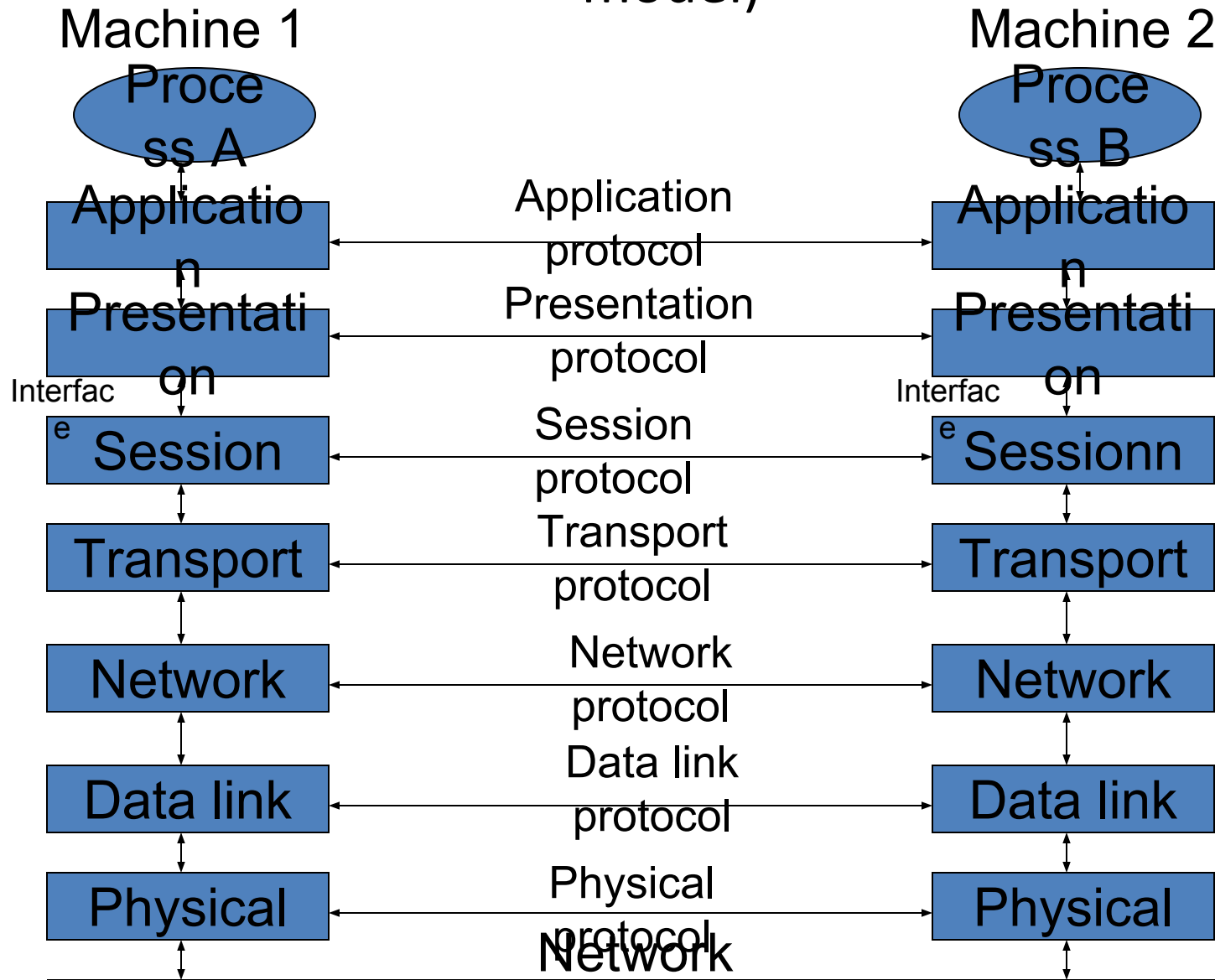
1.It first builds a message in its own address space

2.It executes a system call

3.The OS fetches the message and sends it through network to 2.

- 1 and 2 have to agree on the meaning of the bits being sent. For example,
  - How many volts should be used to signal a 0-bit? 1-bit?
  - How does the receiver know which is the last bit of the message?
  - How can it detect if a message has been damaged or lost?
  - What should it do if it finds out?
  - How long are numbers, strings, and other data items? And how are they represented?

# OSI (Open System Interconnection Reference model)

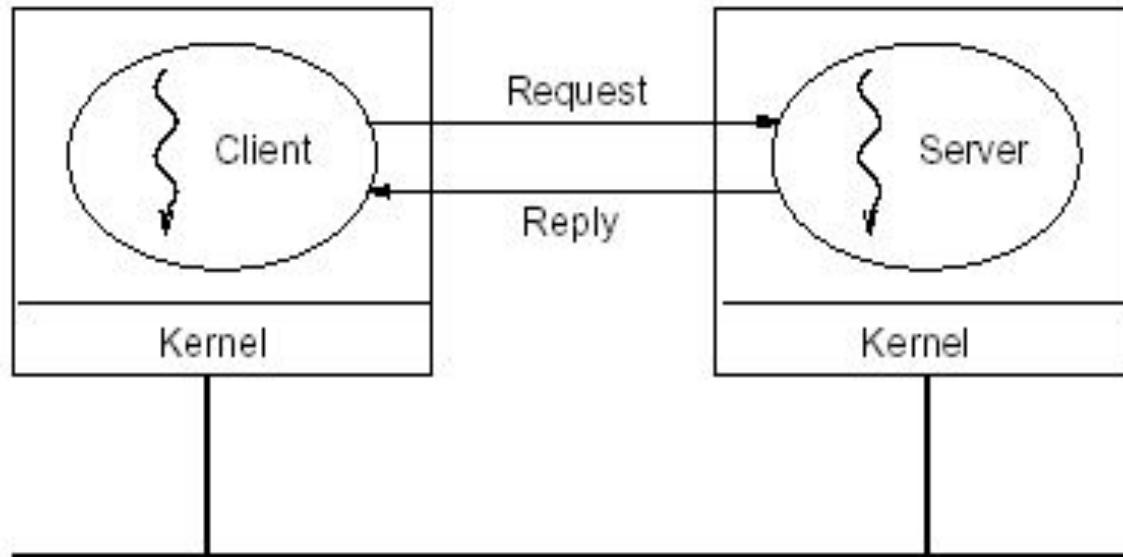




# Client-Server Model Layer

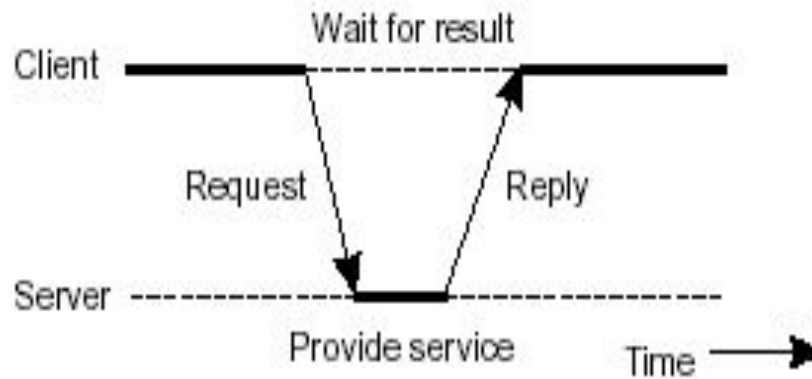
7	
6	
5	Request/Reply
4	
3	
2	Data link
1	Physical

# Client-Server



# Client-Server from another perspective

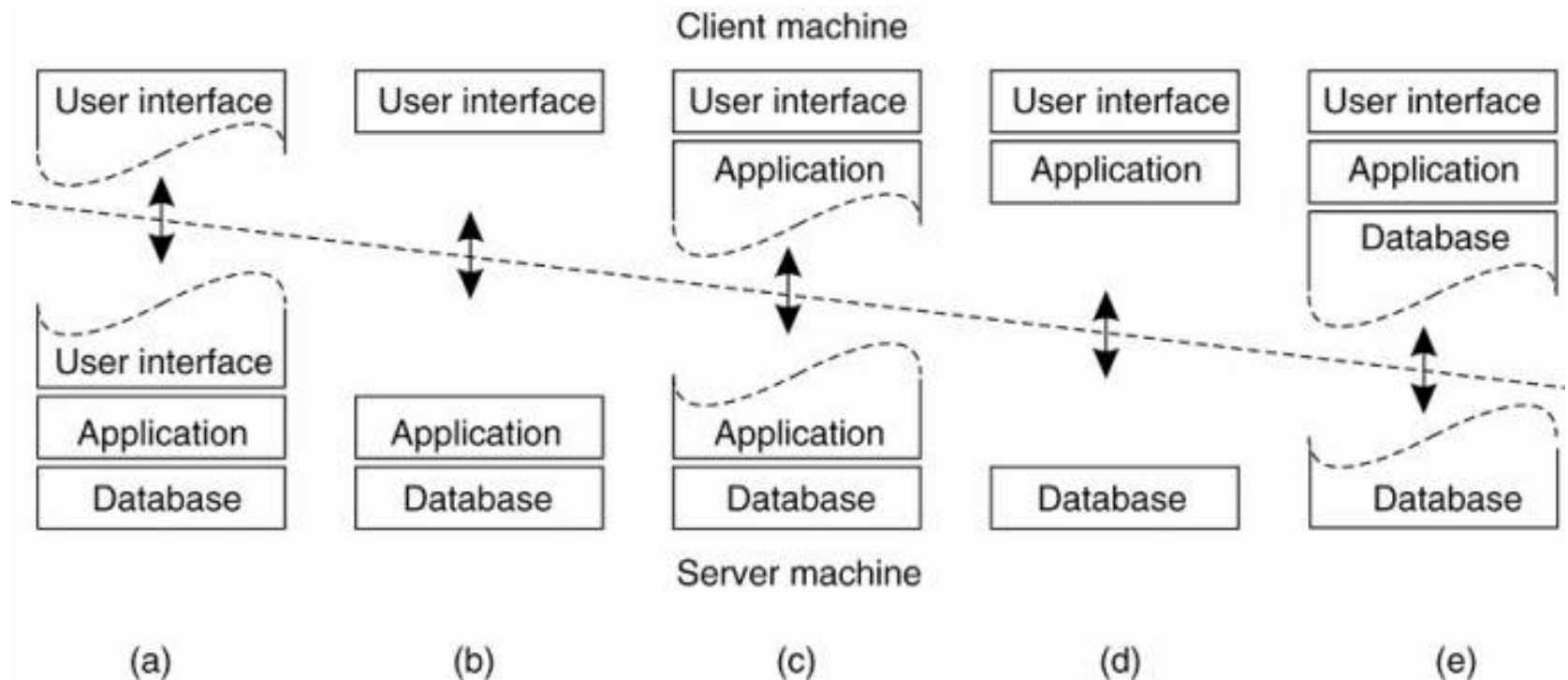
- A typical client-server application can be decomposed into three logical parts: the interface part, the application logic part, and the data part.
- Implementation of the client-server architecture vary with regards to how the parts are separated over the client and server roles.



# Advantages

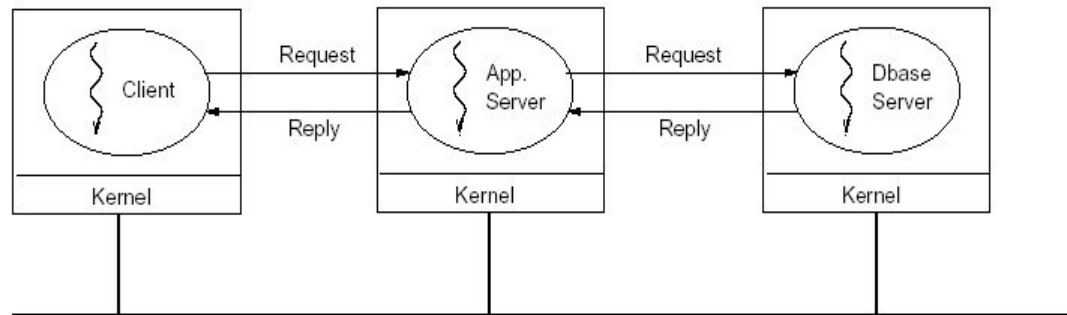
- Simplicity: The client sends a request and gets an answer. No connection has to be established.
- Efficiency: just 3 layers. Getting packets from client to server and back is handled by 1 and 2 by hardware: an Ethernet or Token ring. No routing is needed and no connections are established, so layers 3 and 4 are not needed. Layer 5 defines the set of legal requests and replies to these requests.
- two system calls: send (dest, &mptr), receive (addr, &mptr)

# Alternative Client-Server Architectures



# Vertical Distribution (Multi-Tier)

- Splitting up a server's functionality over multiple computers
- An extension of the client-server architecture, the vertical distribution, or multi-tier, architecture distributes the traditional server functionality over multiple servers. A client request is sent to the first server.

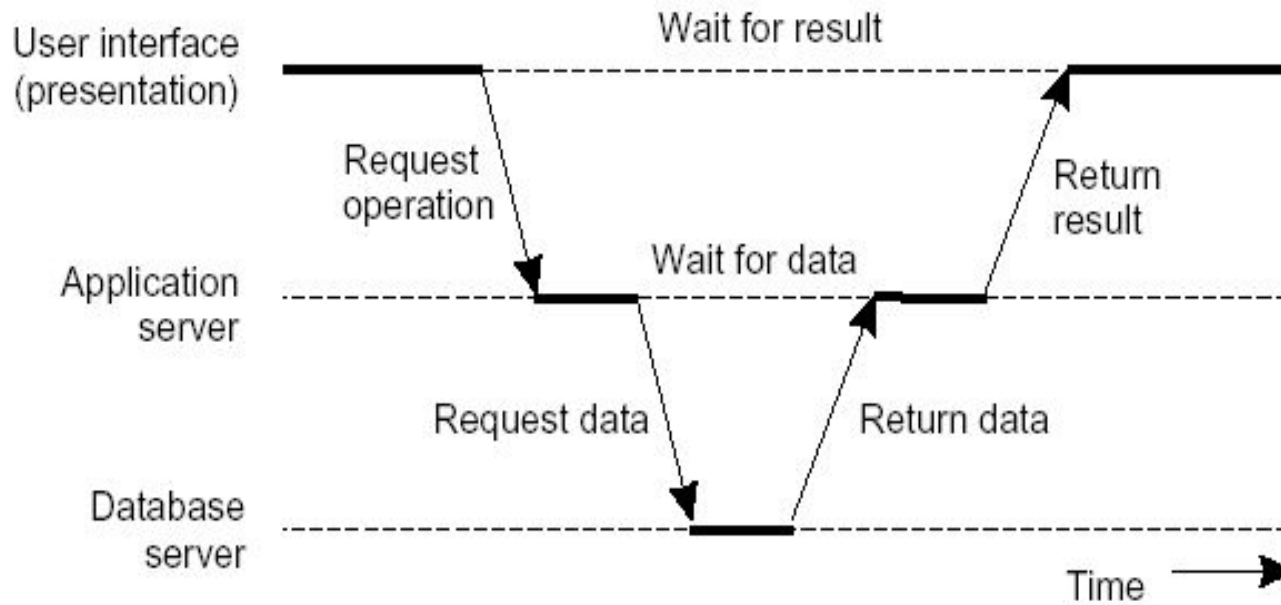


## Three layers' of functionality:

- User interface
- Processing/Application logic
- Data
- Splitting up the server functionality in this way is beneficial to a system's scalability as well as its flexibility.
- Scalability is improved because the processing load on each individual server is reduced, and the whole system can therefore accommodate more users.

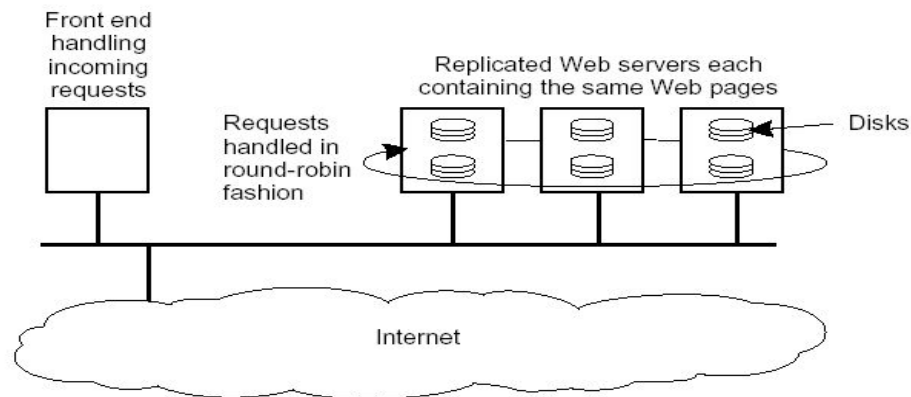
**Logically different components on different machines**

# Vertical Distribution from another perspective



# Horizontal Distribution

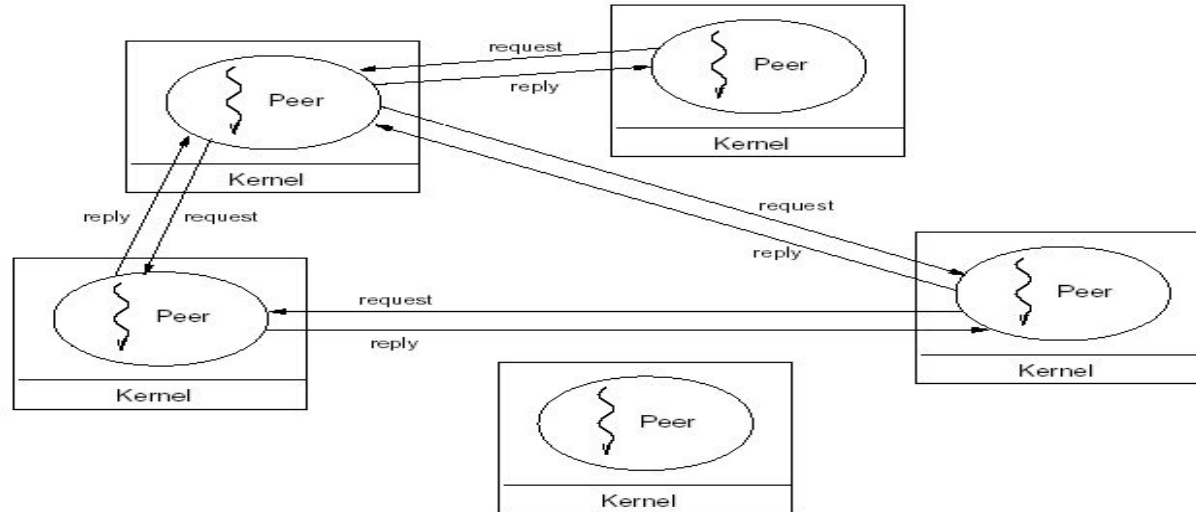
- Replicating a server's functionality over multiple computers
- In this case each server machine contains a complete copy of all hosted Web pages and client requests are passed on to the servers in a round robin fashion.
- The horizontal distribution architecture is generally used to improve scalability (by reducing the load on individual servers) and reliability (by providing redundancy).
- Logically equivalent components replicated on different machines.





# Peer To Peer Communication Architecture

- All p
- With peer track

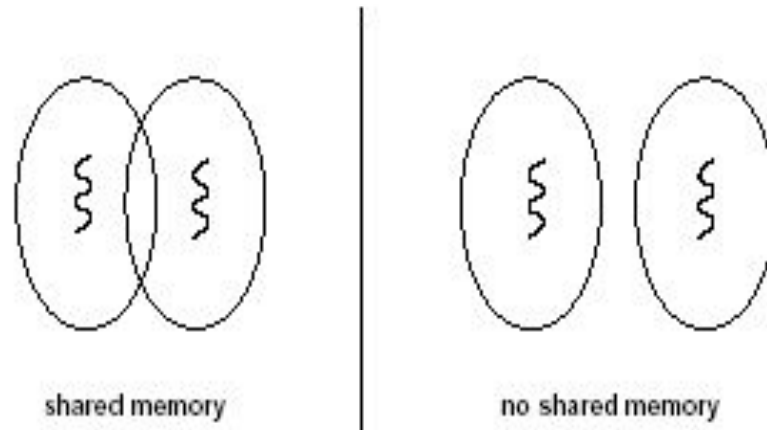


peer to keep  
ffer.

# Processes and Threads

- **Process:** Single thread of control per address space
- **Thread:** Multiple threads of control per address space

## Memory access:



# Stateful Vs Stateless Servers

- Stateful:

- Keeps persistent information about clients
- Improved performance
- Expensive recovery
- Must track clients

- Stateless:

- Does not keep state of clients
- soft state design: limited client state
- Can change own state without informing clients
- Increased communication

# Communication: Non distributed system

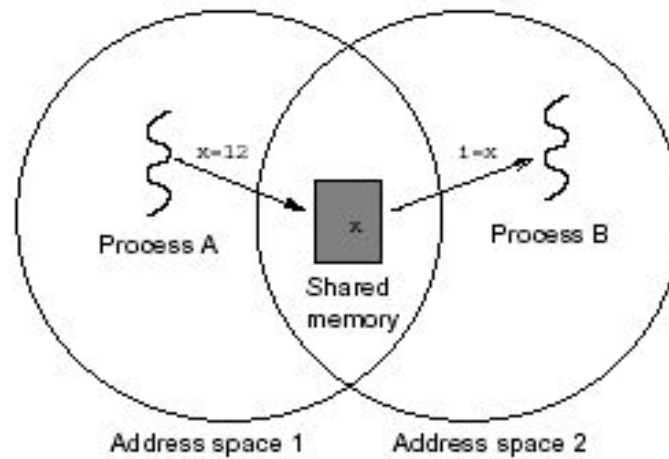
## □ Cooperating processes need to communicate:

- For synchronization and control (Processes synchronize in order to coordinate their activities)
- To share data (Processes share data about tasks that they are cooperatively working on)

## □ 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)
- **Message passing**- OS's IPC mechanisms

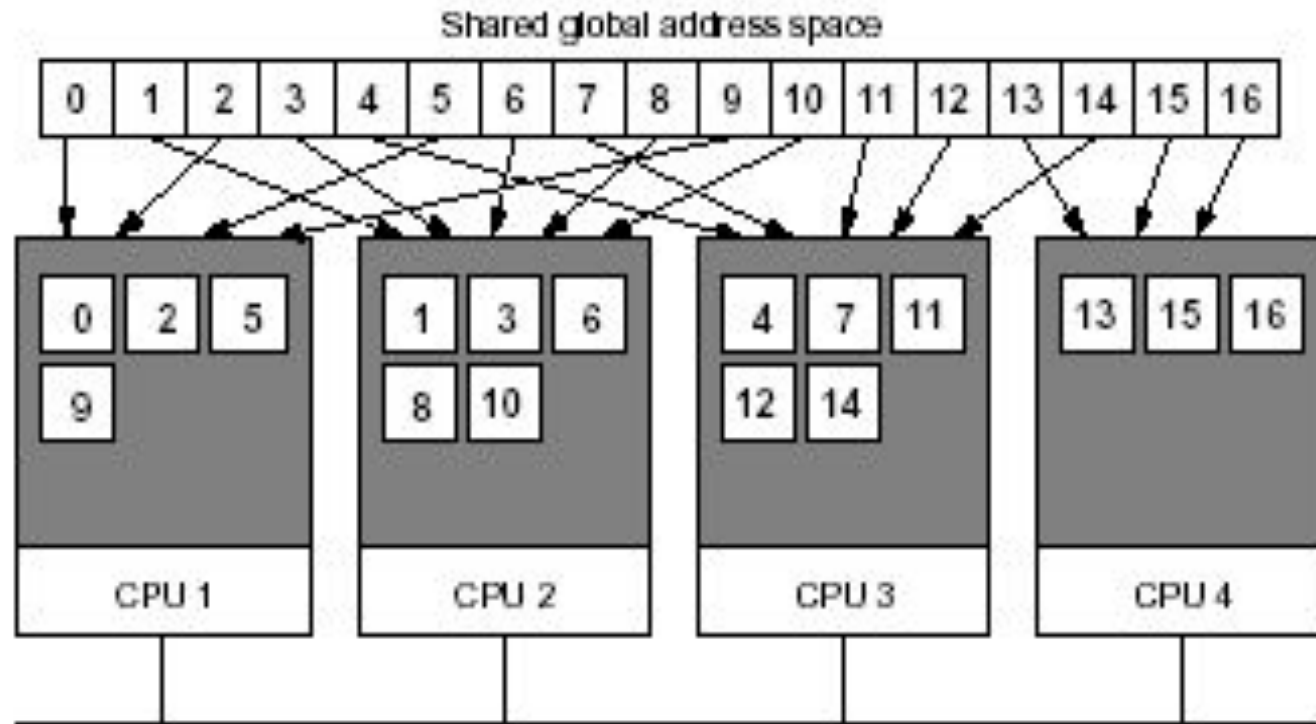
# Communication: Distributed System



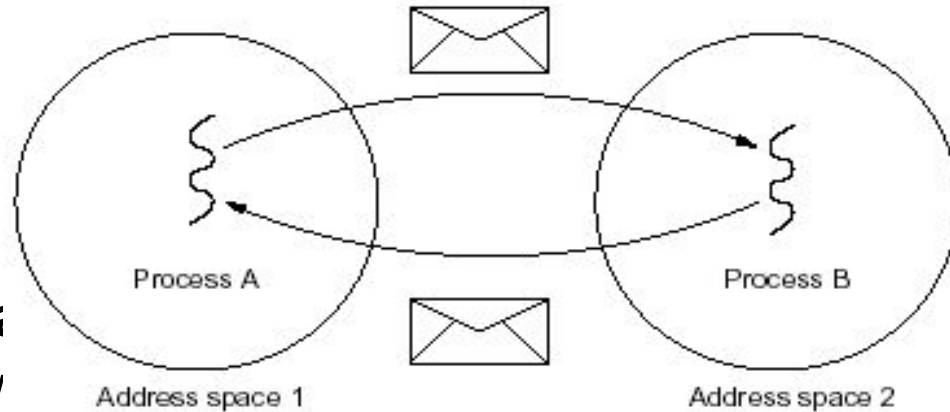
☐ Shared Me

- There is no way, . . . , . . . ,
- Distributed Shared Memory

# Distributed Shared Memory (DSM)



# Message Passing



## □ Message Pa

- Over the netw
- Introduces lat
- Introduces higher chances of failure
- Heterogeneity introduces possible incompatibilities

# Coupling

- Dependency between sender and receiver
- Temporal: do sender and receiver have to be active at the same time?
- Spatial: do sender and receiver have to know about each other?  
Explicitly address each other?
- Semantic: do sender and receiver have to share knowledge of content  
syntax and semantics?
- Platform: do sender and receiver have to use the same platform?
- Tight Vs loose coupling: yes Vs no



# Message Passing

- **Basics**
  - Send()
  - Receive()
- **Variations:**
  - Connection oriented Vs Connectionless
  - Point-to-point Vs Group
  - Synchronous Vs Asynchronous
  - Buffered Vs Unbuffered
  - Reliable Vs Unreliable
  - Message ordering guarantees
- **Data Representation:**
  - Marshalling
  - Endianness

# Desirable features of a good message-passing system

- Simplicity
- Uniform semantics
- Efficiency
- Reliability
- Correctness
- Flexibility
- Security
- portability

# Communication Modes

- Synchronous Vs asynchronous communication
- Data oriented Vs control oriented communication
- Transient Vs persistent communication
- Provider-Initiated Vs Consumer-Initiated Communication
- Direct-Addressing Vs Indirect-Addressing Communication

# Synchronous Communication

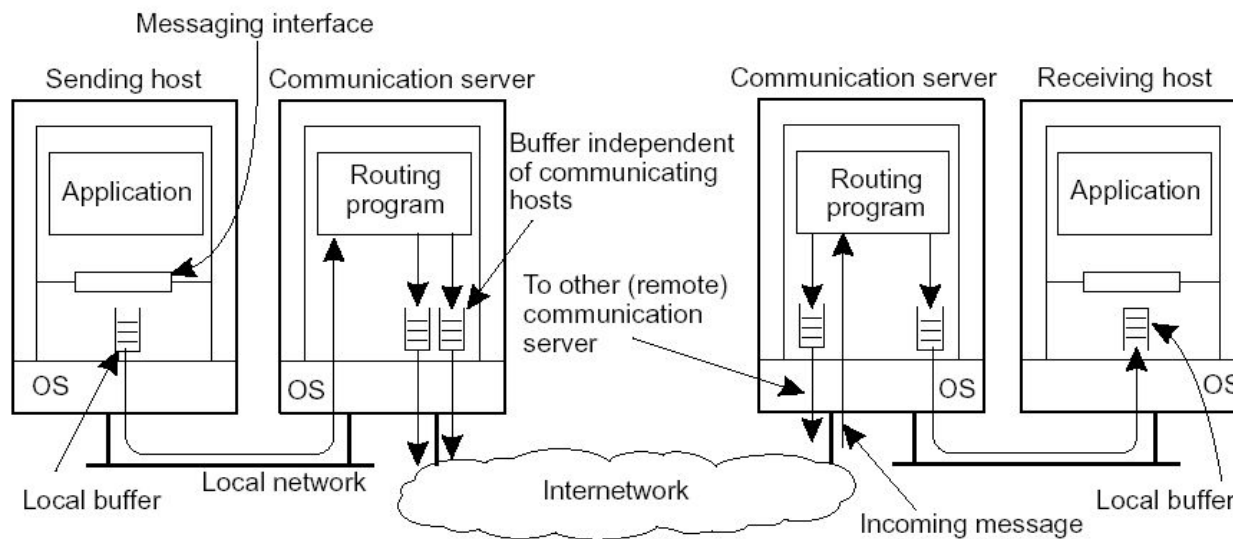
- Client/Server computing is generally based on a model of **synchronous communication**
- Client and server have to be active at the time of communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

## **Drawbacks of synchronous communication:**

- Client cannot do any other work while waiting for reply
- Failures have to be dealt with immediately (the client is waiting)
- In many cases the model is simply not appropriate (mail, news)

# Asynchronous Communication

- 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



# Data-Oriented Vs Control-Oriented Communication

## □ Data-oriented communication

- Facilitates data exchange between threads
- Shared address space, shared memory & message passing

## □ Control-oriented communication

- Associates a transfer of control with every data transfer
- remote procedure call (RPC) & remote method invocation (RMI)

□ Hardware and OSes often provide data-oriented communication

□ Higher-level infrastructure often provides control-oriented communication -- middleware

# Transient vs Persistent Communication

## □ Transient:

- Message discarded if cannot be delivered to receiver immediately
- Example: HTTP request

## □ Persistent

- Message stored (somewhere) until receiver can accept it
- Example: email

# Provider-Initiated Vs Consumer-Initiated Communication

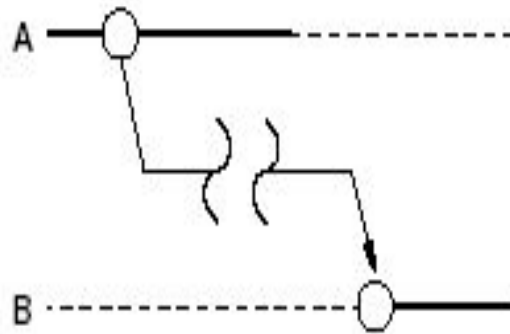
- Provider-Initiated
  - Message sent when data is available
  - Example: notifications
- Consumer-Initiated
  - Request sent for data
  - Example: HTTP request



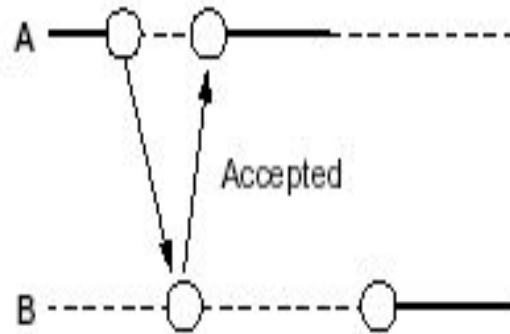
# Direct-Addressing Vs Indirect-Addressing Communication

- Direct-Addressing
  - Message sent directly to receiver
  - Example: HTTP request
- Indirect-Addressing
  - Message not sent to a particular receiver
  - Example: broadcast, publish/subscribe

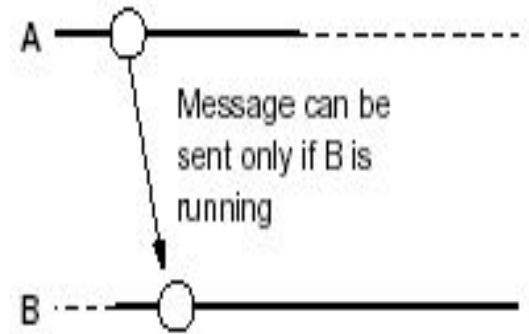
# Messaging Combinations



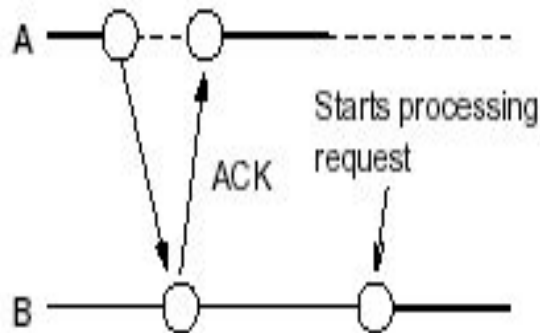
Persistent Asynchronous



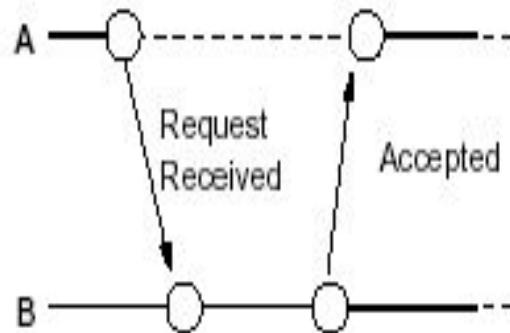
Persistent Synchronous



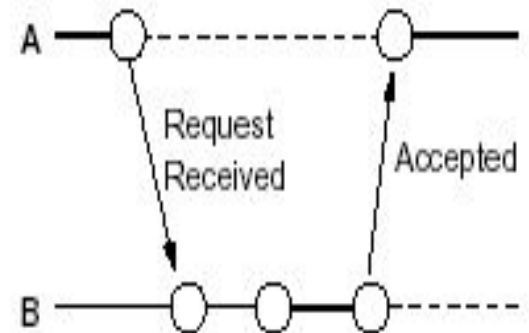
Transient Asynchronous



Transient Synchronous  
(Receipt Based)



Transient Synchronous  
(Delivery Based)



Transient Synchronous  
(Response Based)

# Communication Abstractions

- Number of communication abstractions that make writing distributed applications easier.

## **Provided by higher level APIs:**

- Remote Procedure Call (RPC)
- Remote Method Invocation (RMI)
- Message-Oriented Communication
- Group Communication
- Streams

# Remote Procedure Call (RPC)

- Replace message passing model by execution of a procedure call on a remote node.
- Synchronous - based on blocking messages
- Asynchronous - when no result to return - adding money, records in db, start remote service, etc.
- Message-passing details hidden from application
- Procedure call parameters used to transmit data
- Client calls local “stub” which does messaging and marshaling

# Remote Method Invocation (RMI)

- The transition from Remote Procedure Call (RPC) to Remote Method Invocation (RMI) is a transition from the server metaphor to the object metaphor.

## Why is this important?

- Using RPC: programmers must explicitly **specify the server** on which they want to perform the call.
- Using RMI: programmers invoke methods on remote objects.
- More natural resource management and error handling

# Message-oriented Communication

## Message-oriented Middleware (MOM)

- Message-oriented communication is provided by message-oriented middleware (MOM).

## Middleware support for message passing:-

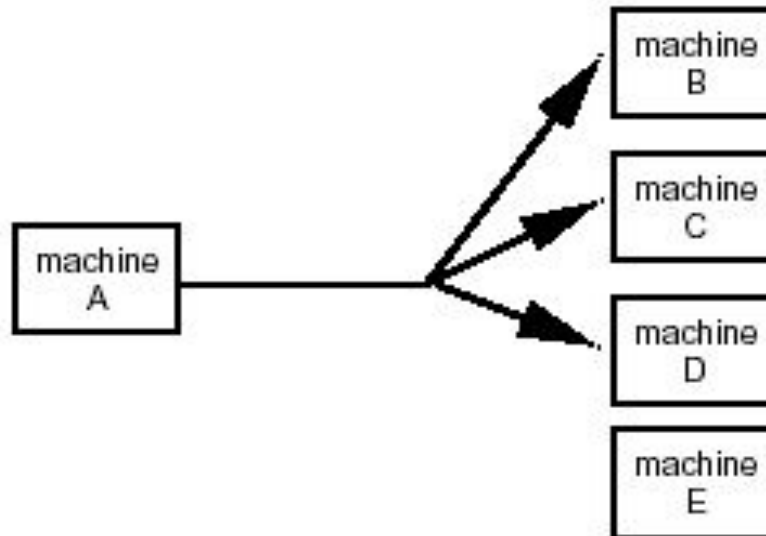
- Asynchronous or Synchronous communication
- Persistent or Transient messages
- Various combinations of the above

## What more does it provide than send()/receive()?

- Persistent communication (Message queues)
- Hides implementation details
- Marshaling (packing of function parameters into a message packet)

# Group Communication

□ Send



# Group Communication

- Two kinds of group communication
  - Broadcast(msg sent to everyone)
  - Multicast(msg sent to specific group)
- Used for
  - Replication of services
  - Replication of data
  - Service discovery
  - Event notification
- Issues
  - Reliability
  - Ordering
- Example: IP multicast



# Stream-Oriented Communication

- Support for continuous media
- Streams in distributed systems
- Stream management

# Continuous Media

- All communication facilities discussed so far are essentially based on a discrete, that is time independent exchange of information

**Continuous media:** Characterized by the fact that values are time dependent:

- Audio
- Video
- Animations
- Sensor data (temperature, pressure, etc.)

**Transmission modes:** Different timing guarantees with respect to data transfer:

- **Asynchronous:** no restrictions with respect to *when* data is to be delivered
- **Synchronous:** define a maximum end-to-end delay for individual data packets
- **Isochronous:** define a maximum and minimum end-to-end delay (**jitter** is bounded)

# Stream

- A (continuous) data stream is a connection-oriented communication facility that supports isochronous data transmission

## Some common stream characteristics:

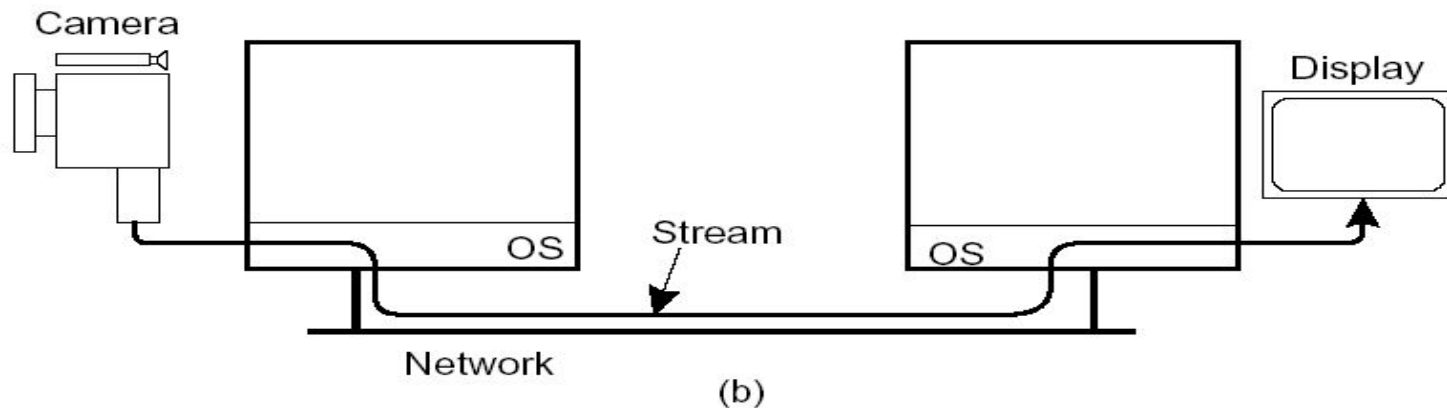
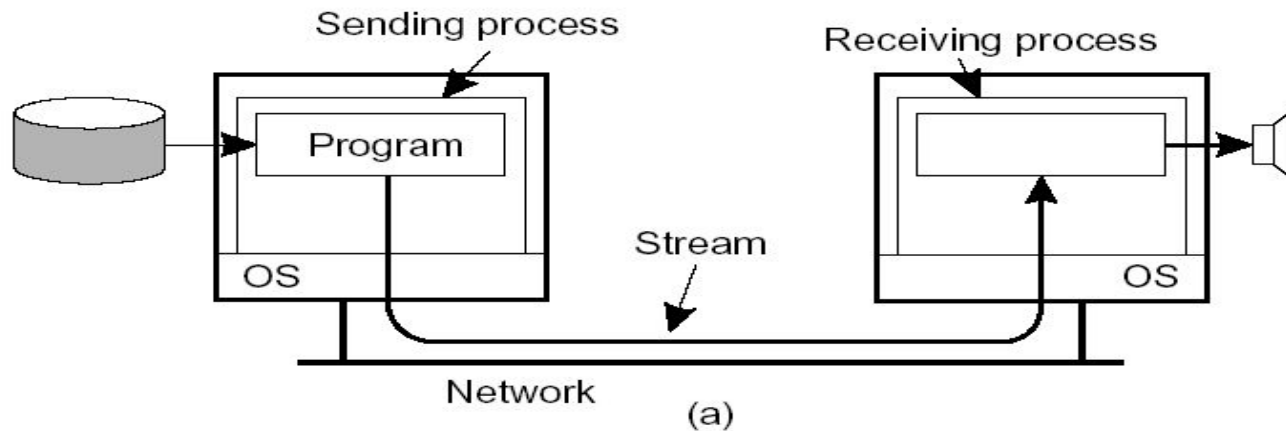
- Streams are unidirectional. There is generally a single **source**, and one or more **sinks**
- Often, either the sink and/or source is a wrapper around hardware (e.g., camera, CD device, TV monitor, dedicated storage)

## Stream types:

- **Simple**: consists of a single flow of data, e.g., audio or video
- **Complex**: multiple data flows, e.g., stereo audio or combination audio/video

# Stream

Streams can be set up between two processes at different machines, or directly between two different devices. Combinations are possible as well.

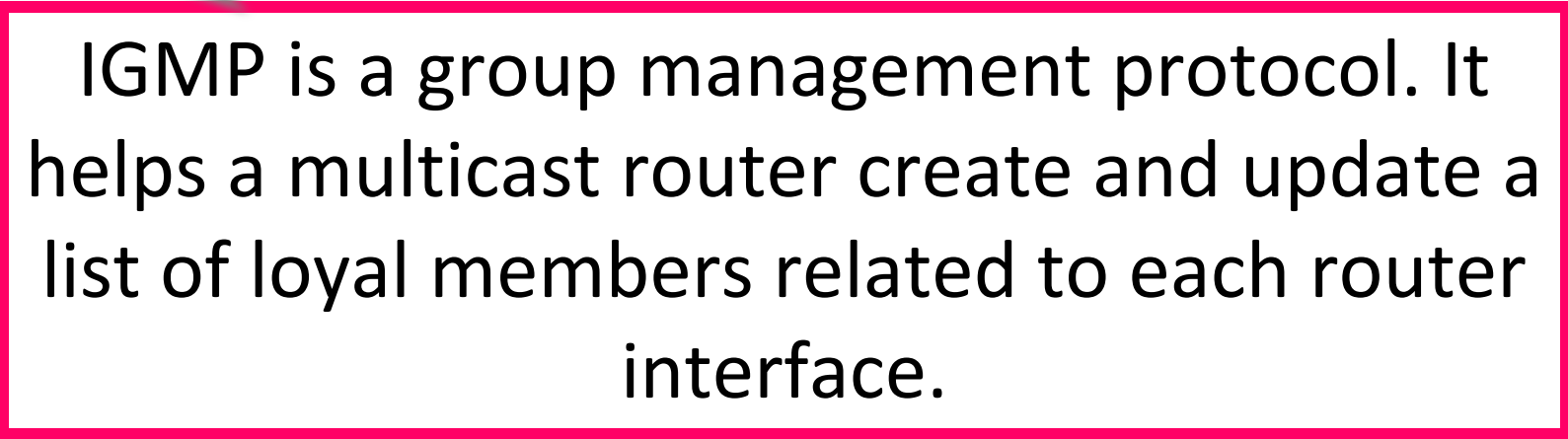


# Internet Group Management Protocol

- IP protocol can be involved in two types of communication: Unicasting and Multicasting.
- Multicasting has many applications. For example, multiple stockbrokers can simultaneously be informed of changes in a stock price.
- IGMP is a protocol that manages group membership. The IGMP protocol gives the multicast routers information about the membership status of hosts (routers) connected to the network.

A yellow speech bubble with a black outline and a small tail pointing towards the bottom left.

Note:

A pink rectangular box with a thick border.

IGMP is a group management protocol. It helps a multicast router create and update a list of loyal members related to each router interface.

# IGMP message types

