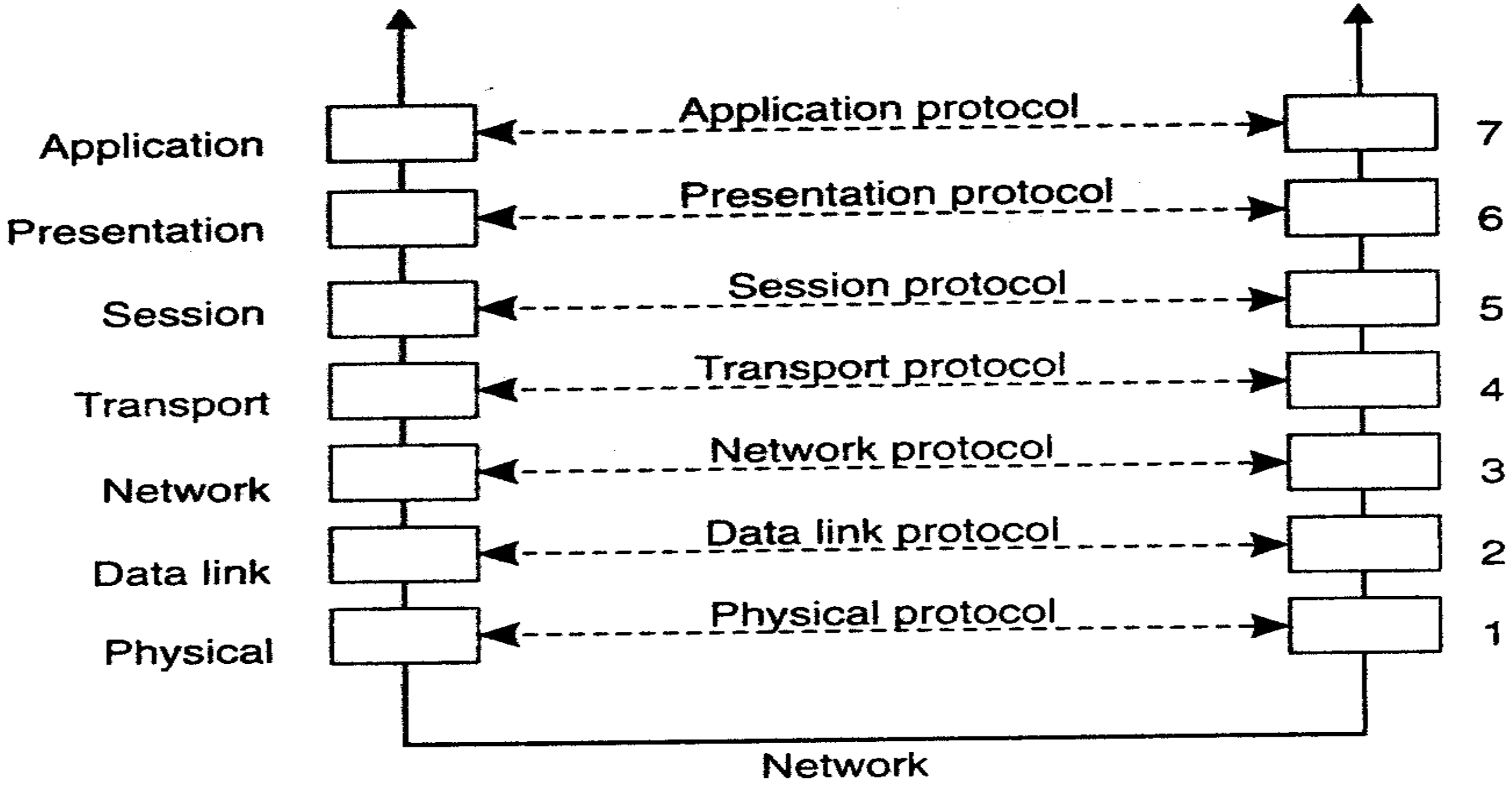
Unit-4
Communication

- Inter-process communication is at the heart of all distributed systems.
- **Inter-process Communication** is a process of exchanging the data between two or more independent process in a distributed environment.
- Expressing communication through message passing is harder than using primitives based on shared memory, as available for non distributed platforms.
- Modem distributed systems often consist of thousands or even millions of processes scattered across a network with unreliable communication such as the Internet.

4.1 Foundations

- **Layered Protocols**
- Due to the absence of shared memory, all communication in distributed systems is based on sending and receiving (low level) messages.
- When process *A* wants to communicate with process *B*, it first builds a message in its own address space. Then, it executes a system call that causes the operating system to send the message over the network to *B*.
- The OSI model is designed to allow open systems to communicate. An open system is one that is prepared to communicate with any other open system by using standard rules that govern the format, contents, and meaning of the messages sent and received.
- These rules are formalized in what are called protocols. To allow a group of computers to communicate over a network, they must all agree on the protocols to be used.

- **With connection oriented protocols**, before exchanging data the sender and receiver first explicitly establish a connection, and possibly negotiate the protocol they will use.
- When they are done, they must release (terminate) the connection. The telephone is a connection-oriented communication system.
- **With connectionless protocols**, no setup in advance is needed. The sender just transmits the first message when it is ready.
- Dropping a letter in a mailbox is an example of connectionless communication.
- With computers, both connection-oriented and connectionless communication are common.


Application
Presentation
Session
Transport
Network
Data link
Physical
Application protocol
Presentation protocol
Session protocol
Transport protocol
Network protocol
Data link protocol
Physical protocol
7
6
5
4
3
2
1
Network

- Feature of OSI Model

1. Big picture of communication over network is understandable through this OSI model.
2. We see how hardware and software work together.
3. We can understand new technologies as they are developed.
4. Troubleshooting is easier by separate networks.
5. Can be used to compare basic functional relationships on different networks.

- Principles of OSI Reference Model
- The OSI reference model has 7 layers. The principles that were applied to arrive at the seven layers can be briefly summarized as follows:

1. A layer should be created where a different abstraction is needed.
2. Each layer should perform a well-defined function.
3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.
4. The layer boundaries should be chosen to minimize the information flow across the interfaces.
5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity and small enough that architecture does not become unwieldly.

- OSI Model Layer 1: The Physical Layer

1. Physical Layer is the lowest layer of the OSI Model.
2. It activates, maintains and deactivates the physical connection.
3. It is responsible for transmission and reception of the unstructured raw data over network.
4. Voltages and data rates needed for transmission is defined in the physical layer.
5. It converts the digital/analog bits into electrical signal or optical signals.
6. Data encoding is also done in this layer.

- OSI Model Layer 2: Data Link Layer

1. Data link layer synchronizes the information which is to be transmitted over the physical layer.
2. The main function of this layer is to make sure data transfer is error free from one node to another, over the physical layer.
3. Transmitting and receiving data frames sequentially is managed by this layer.
4. This layer sends and expects acknowledgements for frames received and sent respectively. Resending of non-acknowledgement received frames is also handled by this layer.
5. This layer establishes a logical layer between two nodes and also manages the Frame traffic control over the network. It signals the transmitting node to stop, when the frame buffers are full.

- OSI Model Layer 3: The Network Layer

1. Network Layer routes the signal through different channels from one node to other.
2. It acts as a network controller. It manages the Subnet traffic.
3. It decides by which route data should take.
4. It divides the outgoing messages into packets and assembles the incoming packets into messages for higher levels

- OSI Model Layer 4: Transport Layer

1. Transport Layer decides if data transmission should be on parallel path or single path.
2. Functions such as Multiplexing, Segmenting or Splitting on the data are done by this layer
3. It receives messages from the Session layer above it, convert the message into smaller units and passes it on to the Network layer.
4. Transport layer can be very complex, depending upon the network requirements.

- OSI Model Layer 5: The Session Layer

1. Session Layer manages and synchronize the conversation between two different applications.
2. Transfer of data from source to destination session layer streams of data are marked and are resynchronized properly, so that the ends of the messages are not cut prematurely and data loss is avoided.

- OSI Model Layer 6: The Presentation Layer

1. Presentation Layer takes care that the data is sent in such a way that the receiver will understand the information (data) and will be able to use the data.
2. While receiving the data, presentation layer transforms the data to be ready for the application layer.
3. Languages(syntax) can be different of the two communicating systems. Under this condition presentation layer plays a role of translator.
4. It performs Data compression, Data encryption, Data conversion etc.

- OSI Model Layer 7: Application Layer

1. Application Layer is the topmost layer.
2. Transferring of files distributing the results to the user is also done in this layer. Mail services, directory services, network resource etc are services provided by application layer.
3. This layer mainly holds application programs to act upon the received and to be sent data.

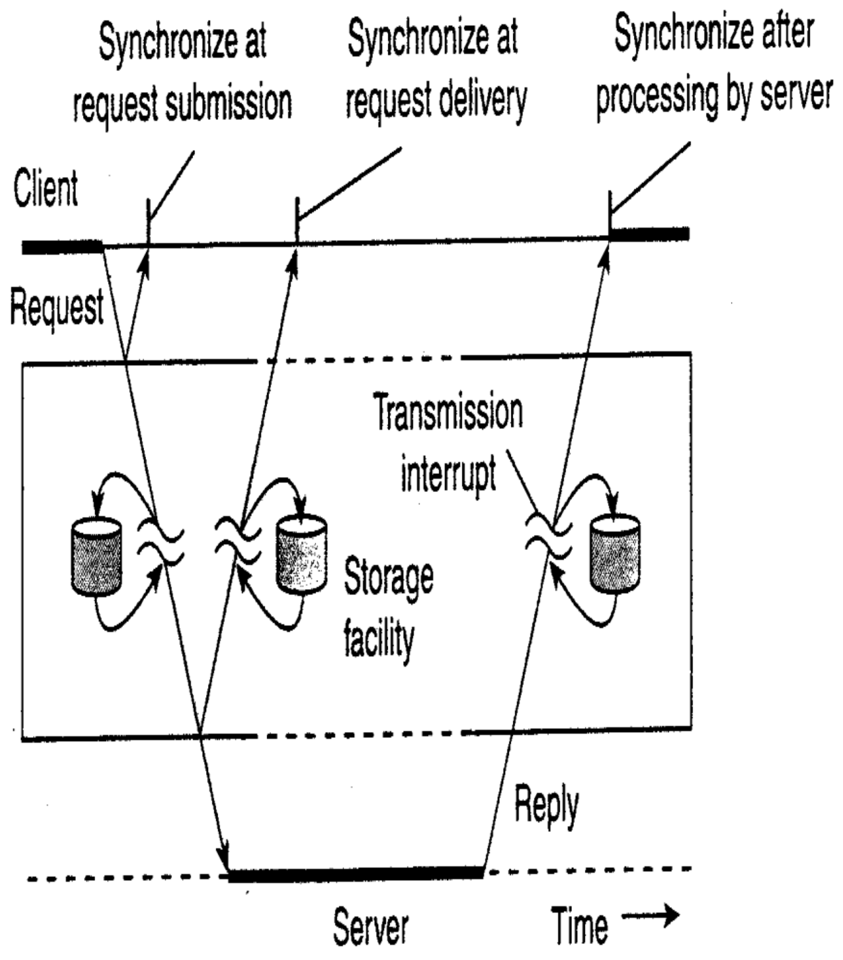
- Merits of OSI reference model

1. OSI model distinguishes well between the services, interfaces and protocols.
2. Protocols of OSI model are very well hidden.
3. Protocols can be replaced by new protocols as technology changes.
4. Supports connection oriented services as well as connectionless service

- Demerits of OSI reference model

1. Model was devised before the invention of protocols.
2. Fitting of protocols is tedious task.
3. It is just used as a reference model.

- To understand the various alternatives in communication that middleware can offer to applications, we view the middleware as an additional service in client-server computing, as shown in Fig.
- Consider, for example an electronic mail system.
- In principle, the core of the mail delivery system can be seen as a middleware communication service. Each host runs a user agent allowing users to compose, send, and receive e-mail. A sending user agent passes such mail to the mail delivery system, expecting it, in turn, to eventually deliver the mail to the intended recipient.
- Likewise, the user agent at the receiver's side connects to the mail delivery system to see whether any mail has come in. If so, the messages are transferred to the user agent so that they can be displayed and read by the user.


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- An electronic mail system is a typical example in which communication is persistent.
- **With persistent communication**, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver.
- In this case, the middleware will store the message at one or several of the storage facilities shown in Fig.
- As a consequence, it is not necessary for the sending application to continue execution after submitting the message.
- Likewise, the receiving application need not be executing when the message is submitted.

- In contrast, with **transient communication**, a message is stored by the communication system only as long as the sending and receiving application are executing.
- More precisely, in terms of Fig., the middleware cannot deliver a message due to a transmission interrupt, or because the recipient is currently not active, it will simply be discarded.
- Typically, all transport-level communication services offer only transient communication.
- In this case, the communication system consists traditional store-and-forward routers. If a router cannot deliver a message to the next one or the destination host, it will simply drop the message.

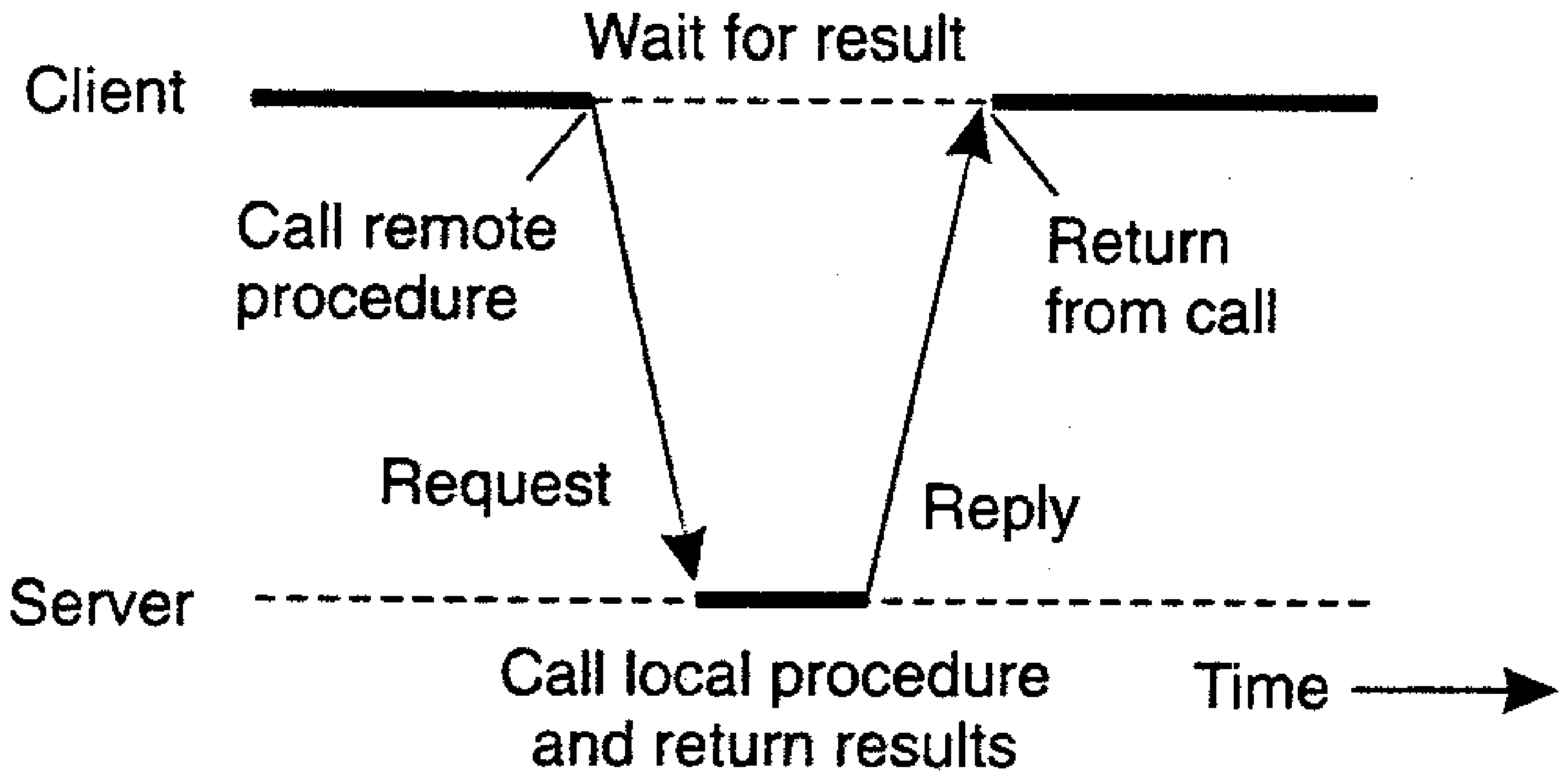
- The characteristic feature of **asynchronous** communication is that a sender continues immediately after it has submitted its message for transmission. This means that the message is (temporarily) stored immediately by the middleware upon submission.
- With **synchronous** communication, the sender is blocked until its request is known to be accepted. **There are essentially three points where synchronization can take place.**
- First, the sender may be blocked until the middleware notifies that it will take over transmission of the request.
- Second, the sender may synchronize until its request has been delivered to the intended recipient.
- Third, synchronization may take place by letting the sender wait until its request has been fully processed, that is, up the time that the recipient returns a response.

4.2 REMOTE PROCEDURE CALL

- When a process on machine *A* calls' a procedure on machine *B,* the calling process on *A* is suspended, and execution of the called procedure takes place on *B.*
- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing at all is visible to the programmer. This method is known as **Remote Procedure Call,** or often just RPC.
- 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.
- RPC is used to call other processes on the remote systems like a local system. A procedure call is also sometimes known as a *function call* or a *subroutine call.*
- RPC uses the client-server model. The requesting program is a client, and the service-providing program is the server.

<u>Client and Server Stubs</u>

- The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In other words, we want RPC to be transparent-the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa.
- Suppose that a program needs to read some data from a file. The programmer puts a call to read in the code to get the data.
- In a traditional (single-processor) system, the read routine is extracted from the library by the linker and inserted into the object program. It is a short procedure, which is generally implemented by calling an equivalent read system call. In other words, the read procedure is a kind of interface between the user code and the local operating system.


Wait for result
Client
Call remote procedure
Return from call
Request
Reply
Server
Call local procedure and return results
Time

To summarize, a remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.

2. The client stub(A **stub** in distributed computing is a piece of code that converts parameters passed between client and server during a remote procedure call (RPC).) builds a message and calls the local operating system.

3. The client sends the message to the remote server.

4. The remote gives the message to the server stub.

5. The server stub unpacks the parameters and calls the server.

6. The server does the work and returns the result to the stub.

7. The server stub packs it in a message and calls its local server.

8. The server sends the message to the client.

9. The client side gives the message to the client stub.

10. The stub unpacks the result and returns to the client.

The advantages of Remote Procedure Call include the following:

- helps clients communicate with servers via the traditional use of procedure calls in high-level languages;
- can be used in a distributed environment, as well as the local environment;
- supports process-oriented and thread-oriented models;
- hides the internal message-passing mechanism from the user;
- requires only minimal effort to rewrite and redevelop the code;
- provides abstraction, i.e., the message-passing nature of network communication is hidden from the user; and
- omits many of the protocol layers to improve performance.

Some of the disadvantages of RPC include the following:

- The client and server use different execution environments for their respective routines, and the use of resources -- e.g., files -- is also more complex. Consequently, RPC systems aren't really suited for transferring large amounts of data.
- RPC is highly vulnerable to failure because it involves a communication system, another machine and another process.
- There is no uniform standard for RPC; it can be implemented in a variety of ways.
- RPC is only interaction-based, and as such, it doesn't offer any flexibility when it comes to hardware architecture

4.3 MESSAGE-ORIENTED COMMUNICATION

- Remote procedure calls and remote object invocations contribute to hiding communication in distributed systems, that is, they enhance access transparency.
- Unfortunately, neither mechanism is always appropriate.
- In particular, when it cannot be assumed that the receiving side is executing at the time a request is issued, alternative communication services are needed.
- Likewise, the inherent synchronous nature of RPCs, by which a client is blocked until its request has been processed, sometimes needs to be replaced by something else. **That something else is messaging.**

- Message Oriented Communication can be viewed along 2 axes:
- **persistence** (whether the system is persistent or transient); and
- **synchronicity** (whether it is synchronous or asynchronous).
- **Persistence :** In persistent communication, messages are stored at each intermediate hop along the way until the next node is ready to take delivery of the message.
- It is also called a store-and-forward based delivery paradigm. Example: Postal system, email, etc.
- In transient communication, messages are buffered only for small periods of time (as long as sending/receiving applications are executing). If the message cannot be delivered or the next host is down, it is discarded. Example: General TCP/IP communication.

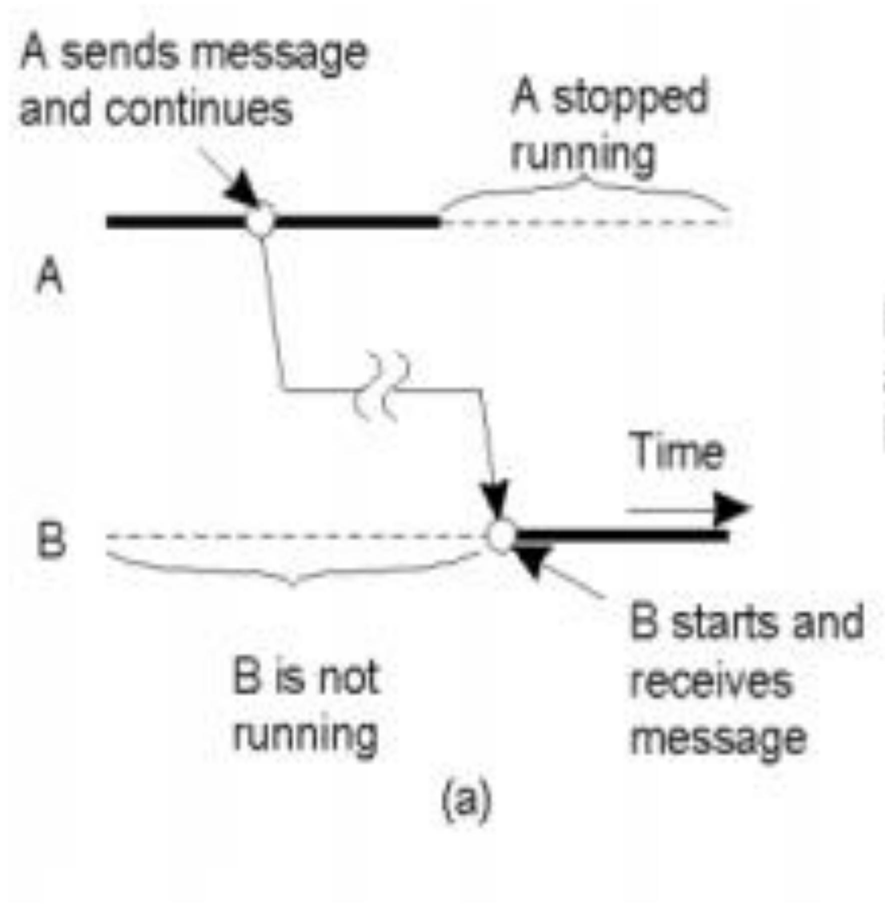
- Synchronicity :
- In synchronous communication. the sender blocks further operations until some sort of an acknowledgement or response is received, hence the name blocking communication.
- In asynchronous or non-blocking communication, the sender continues execution without waiting for any acknowledgement or response. This form needs a local buffer at the sender to deal with it at a later stage

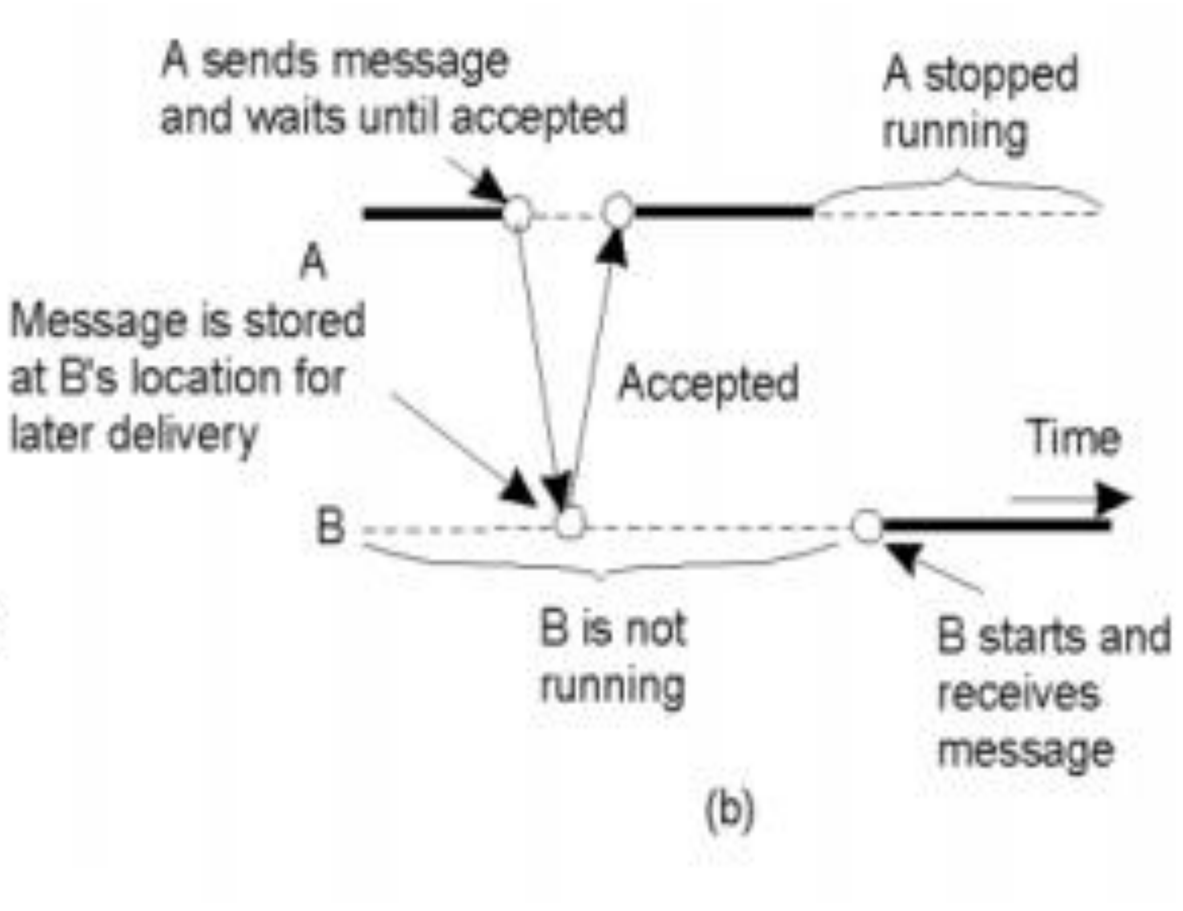
- **Persistent Asynchronous**
- Figure (a) gives us a time-line map for persistent asynchronous communication.
- A is the sender and B is the receiver. Dark line depicts execution. A sends a message and keeps executing without blocking. The message sent from A will take an arbitrary amount of time to reach B.
- A may or may not be running by the time the message reaches B. Disks or multiple memory queues could be used for storage at B’s side.
- There is a guarantee that the message will eventually reach B. B receives this request and processes it then. Example: Emails can take an arbitrary amount of time to reach, and when the receiver reads the email, the sender might not even be running.

- **Persistent Synchronous**
- Figure (b) explains persistent synchronous communication.
- Here, A sends a message, and is then blocked (depicted by the dotted line). The acknowledgement is for receipt (not delivery/response).
- Because this model is persistent, the message may stay in B's queue (or in any router along the way) for an arbitrary amount of time.
- Example: Messaging and chat applications; Many messaging systems are persistent, and can tell us the delivery status

(a) Persistent + Asynchronous

(b) Persistent + Synchronous

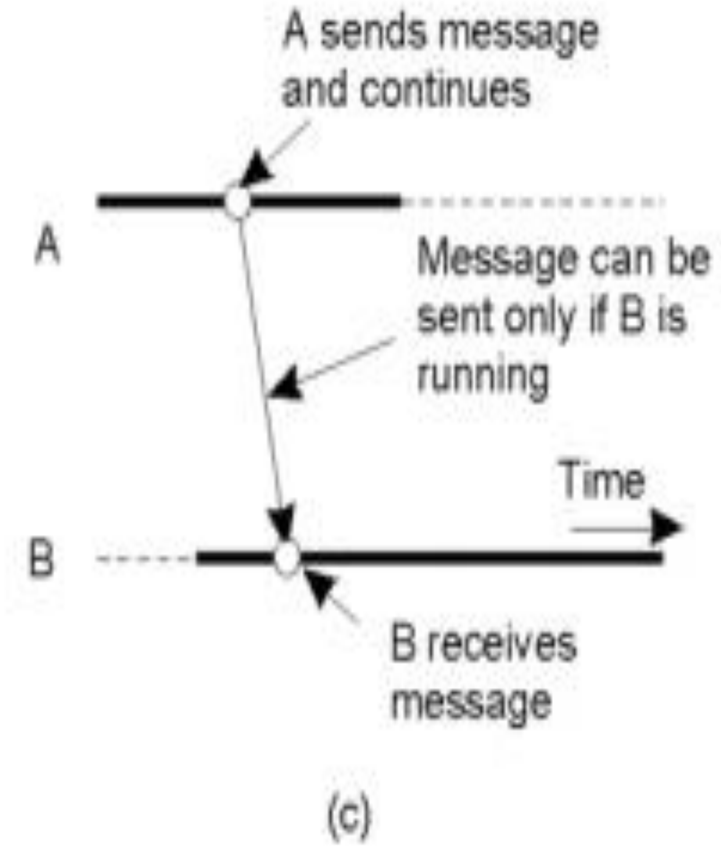


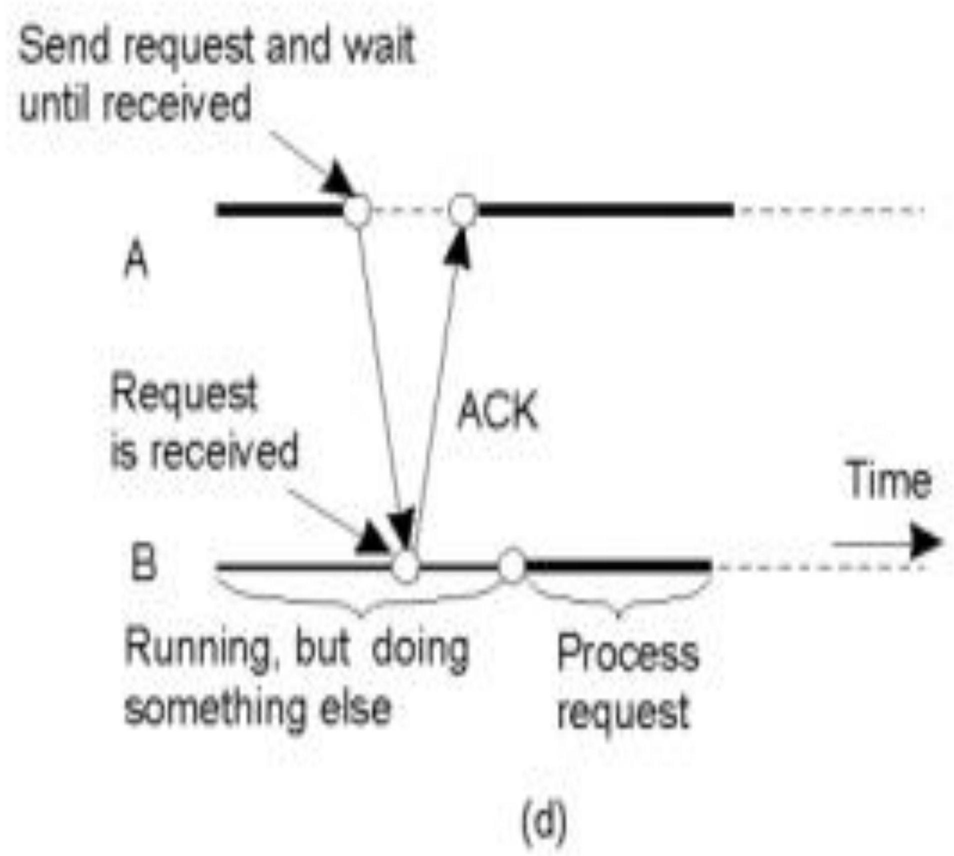

A sends message and waits until accepted
A stopped running
A
Message is stored at B's location for later delivery
Accepted
Time
B
B is not running
B starts and receives message
(b)

- **Transient Asynchronous** Figure (c) explains this form of communication. A sends the message and continues execution (non- blocking). B has to be running, because if it is not running the message will be discarded. Even if any router along the way is down, the message will be discarded. UDP(User Datagram Protocol – a **communications** protocol that facilitates the exchange of messages between computing devices in a network. It's an alternative to the transmission control protocol (TCP)- communication is an example of transient asynchronous communication.
- **Receipt-based Transient Synchronous** Figure (d) explains receipt-based transient synchronous communication. A's message to B is blocking (synchronous) until an ack is received. This ack simply tells us that the message was received at the other end. It does not tell us anything about whether the process has started.

(c) Transient + Asynchronous

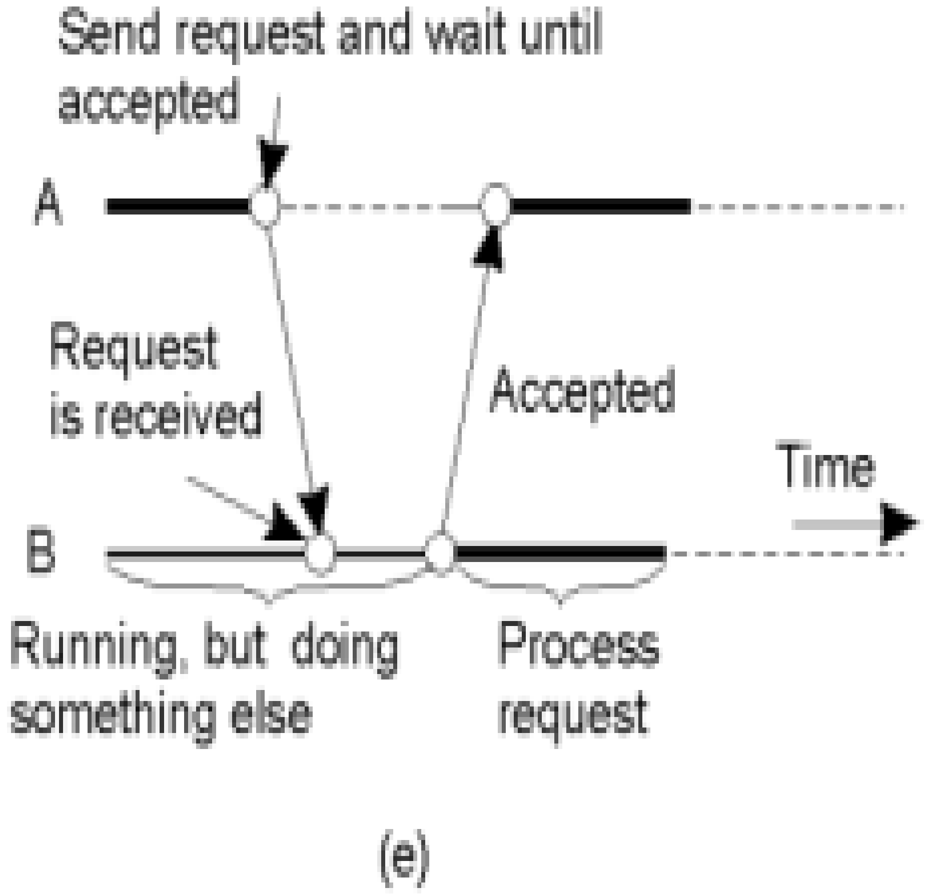
(d) Receipt-based Transient Synchronous

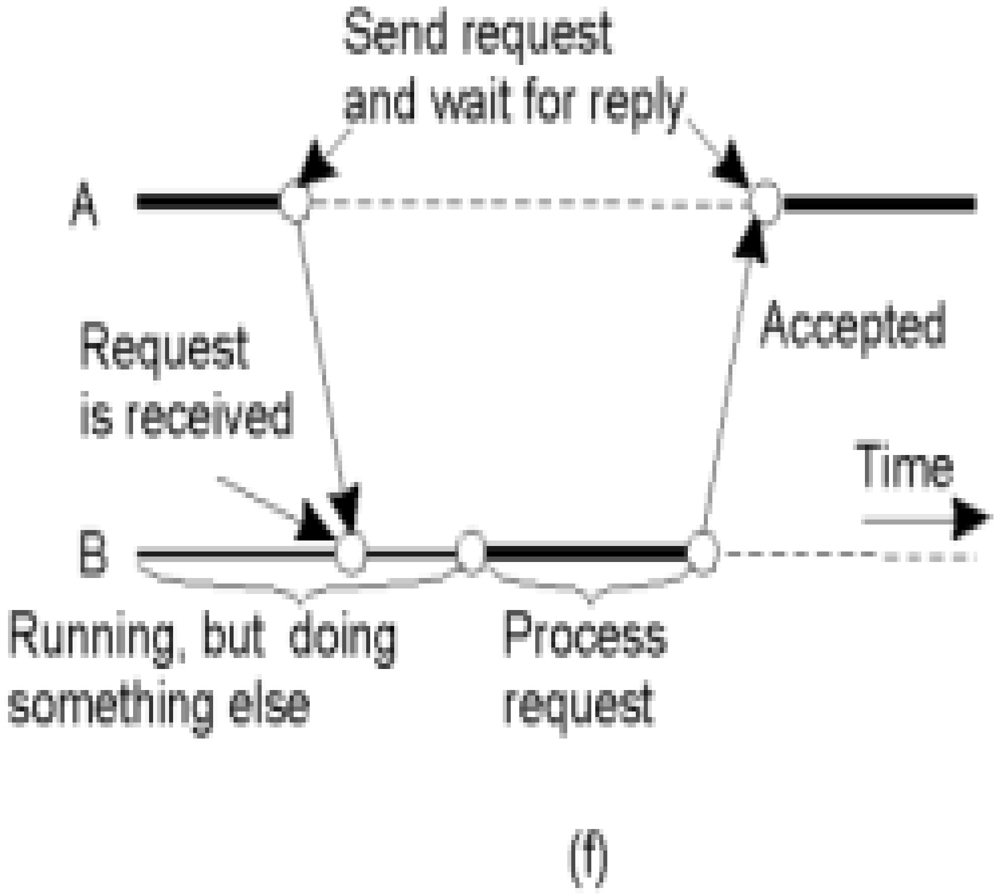

A sends message and continues
A
Message can be sent only if B is running
Time
B
B receives message
(c)


Send request and wait until received
A
Request is received
ACK
Time
B
Running, but doing something else
Process request
(d)

- **Delivery-based Transient Synchronous** This is an extension of receipt-based transient synchronous communication.
- As shown in figure (e), A will resume running when B takes the delivery of the message. The ack comes a little bit later than the previous method. This is essentially asynchronous RPC because from the perspective of an RPC, we are not blocking for the reply.
- **Response-based Transient Synchronous** Figure (f) explains this type of communication. A resumes execution upon receiving a response. That is, not only has the message been delivered, but it has been processed and the ack comes back in the form of a reply. This is traditional RPC. The client is blocked for the entire duration the reply comes back.

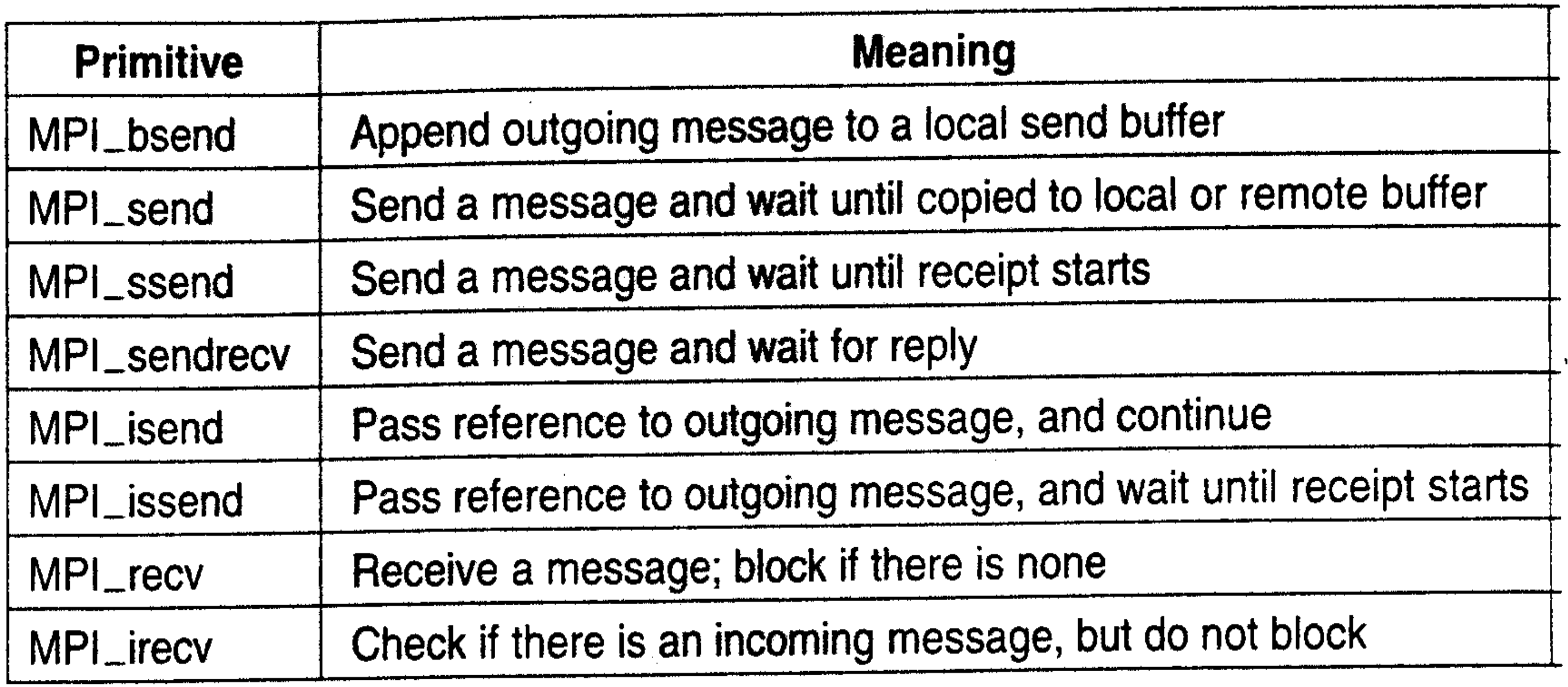
(e) Delivery-based Transient Synchronous (f) Response-based Transient Synchronous


Send request and wait until accepted
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(e)


Send request and wait for reply
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(f)

Message Oriented Transient Communication

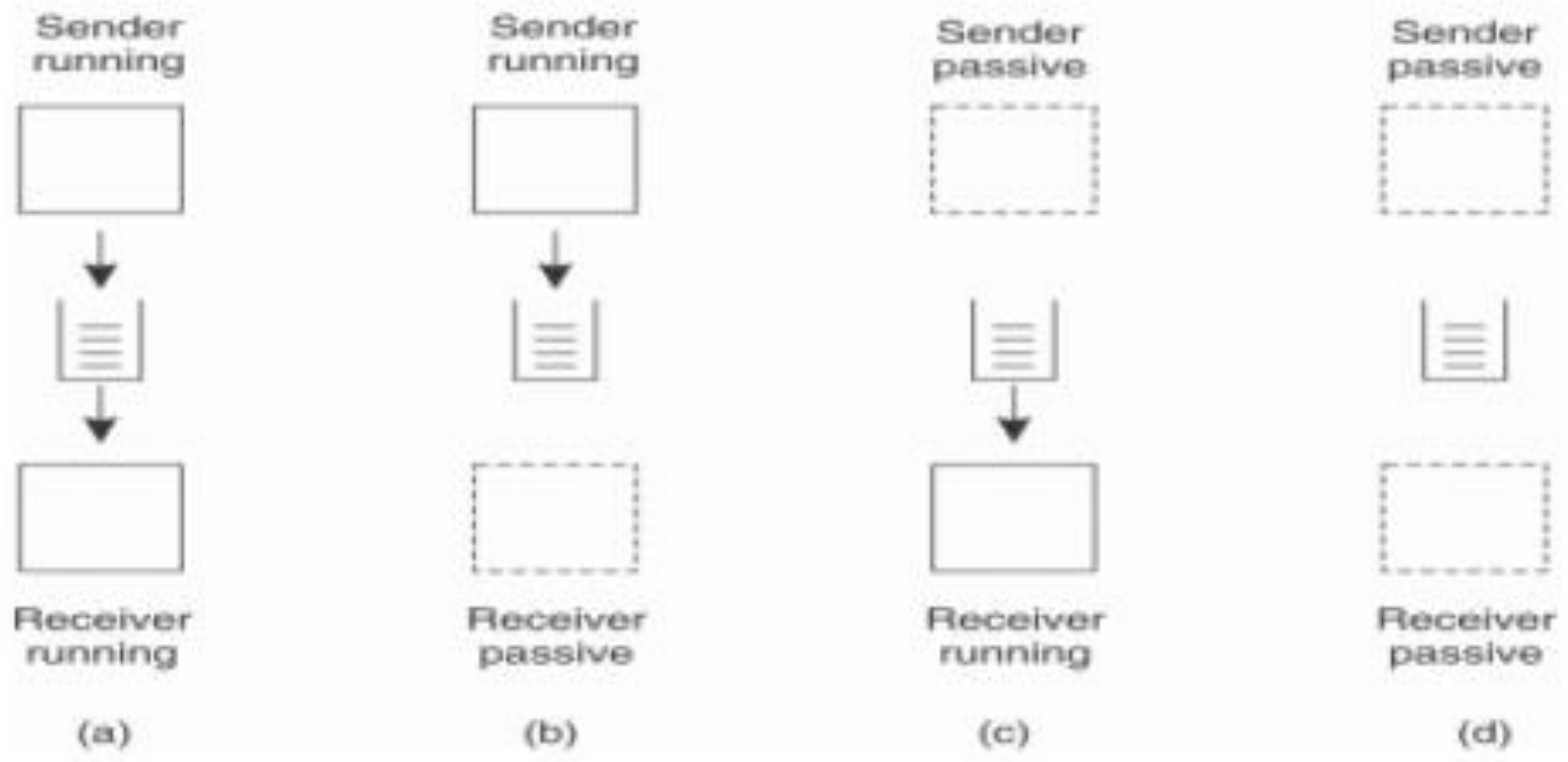
- Berkeley Socket Primitives, and Message-Passing Interface (MPI) are examples of message-oriented transient communication.
- These systems do not use on-disk queues for message storing.
- Below, we map the MPI primitive functions to one of the above types. MPI bsend simply appends the outgoing message to the local send buffer, and then keeps running. Thus, it is **transient asynchronous communication**. MPI send sends a message and waits until copied to the remote buffer. This is similar to delivery-based transient synchronous communication. MPI ssend waits until receipt start and MPI sendrecv waits for a reply, classifying each to receipt based and reply based transient synchronous communication respectively.
- Hence, these constructs map very well to the 6 different communication types mentioned above.
- MPI allows you to choose what kinds of combination of Sync communication you wish to use. It is a much richer communication system where the programmer can choose what form of communication to use from a spectrum of choices.



Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Message Oriented Persistent Communication

- This class of communication is called Message Queuing Systems (MQS). They use on-disk buffers to ensure persistence of messages over long periods of time. Common open-source examples include Kafka, RabbitMQ, ActiveMQ, ZeroMQ, etc. An application level example for this is emails. All emails are stored on disks till delivery. The queues have names/addresses which are used to explicitly put messages into the queue. This is a form of store-and-forward communication. Persistence neither guarantees the duration in which the message will be delivered, nor the reading of the message. It only ensures delivery. Thus, it is **a loosely coupled communication**.
- Figure below explains the working of an MQS. The figure represents a series of queues(doing hop-by hop delivery) as a single abstract persistent queue. Dark boxes indicate execution, while dotted boxes indicate that the system is not running. The first case (a) is the simplest where both A and B are running, and the message is delivered immediately.


Sender running
Receiver running
(a)
Sender running
Receiver passive
(b)
Sender passive
Receiver running
(c)
Sender passive
Receiver passive
(d)

- In case (b) where B is not running, the message is stored at the queue and B takes delivery at a later point.
- Example - email.
- In case (c), the sender is not running when B receives the message. Example - Reading email
- In case (d), the message is queued for an arbitrary amount of time as both A and B are passive. This is **loose communication because the sender and the receiver are not participating in the communication at the same time.**
- The typical abstractions used in the message queuing model are get and put. put allows us to insert into a specific queue. get removes a message from the head of the queue. get and put are analogous to send and receive. There is a non-blocking poll as well, which checks the queue for a message and returns the message if the queue is non-empty.Example - email clients checking servers. notify (also non-blocking) is an async message from the queue to the receiver notifying it of a message arrival.

4.4 Multicast Communication

- An important topic in communication in distributed systems is the support for sending data to multiple receivers, also known as multicast communication.
- Multicast is Communication of One-to-Group of devices, where group is created based on some criteria .

 Each process in a group can send and receive message. A message sent by any process in the group can be received by each participating process in the group.

 Multicast operation is an operation that sends a single message from one process to each of the members of a group of processes.

 In an application or network service which makes use of multicasting, a set of processes form a group, called a multicast group.

 Provides no guarantees about message delivery or ordering.

Characteristics of Multicast

- Fault tolerance based on replicated services
- A replicated service consists of a group of members.
- Client requests are multicast to all the members of the group, each of which performs an identical operation.
- Even if one fails the clients can be still served
- Finding the discovery servers in spontaneous networking
- Multicast messages can be used by servers and clients to locate available discovery services in order to register their interfaces or to look up the interfaces of other services in the distributed system.
- Better performance through replicated data
- Data are replicated to increase the performance of a service.
- Propagation of event notifications
- Multicast to a group may be used to notify processes when something happens.

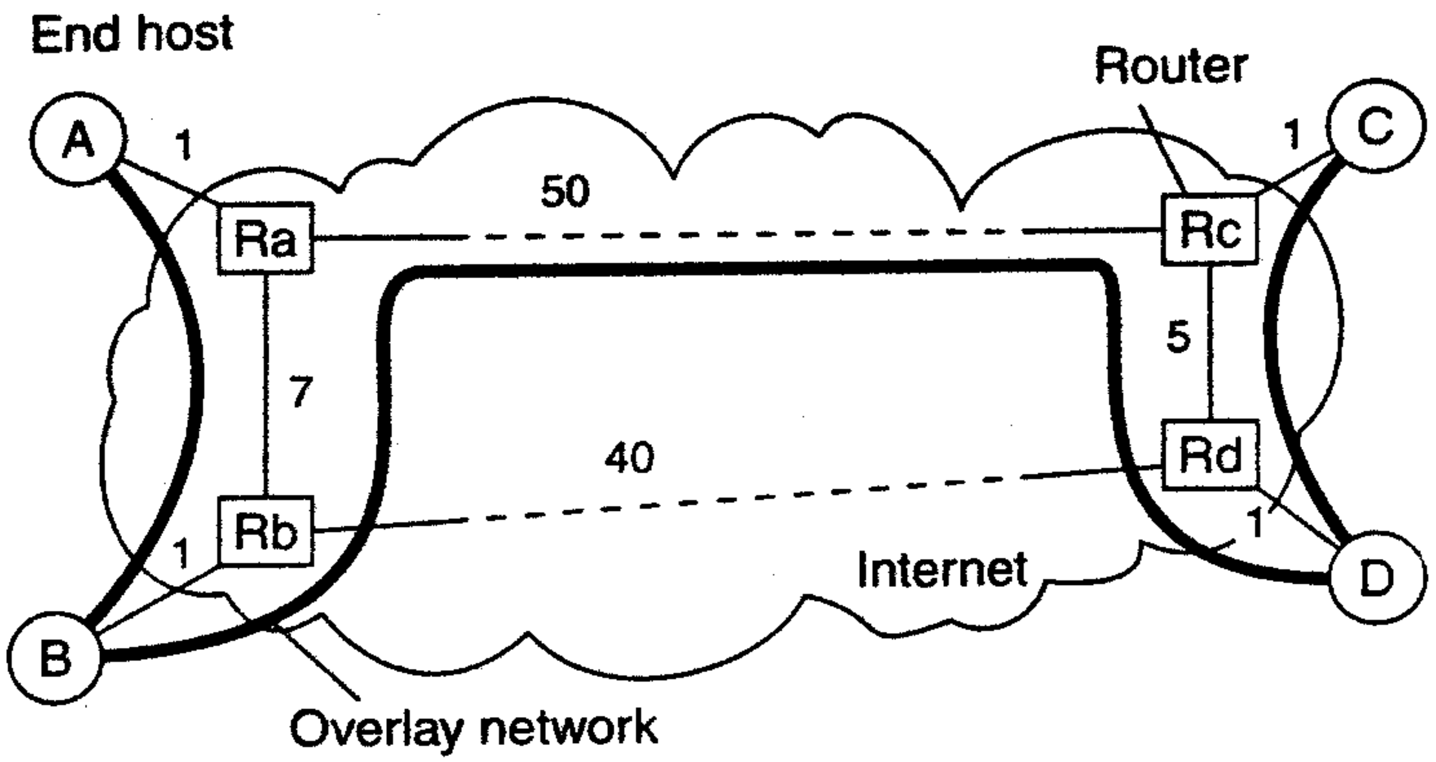
Multicast application examples

- Financial services
- Delivery of news, financial indices, etc
- Remote conferencing/e-learning
- Streaming audio and video to many participants (clients, students)
- Interactive communication between participants
- Data distribution

Application-Level Multicasting

- The basic idea in application-level multicasting is that nodes organize into an overlay network, which is then used to disseminate information to its members.
- An important observation is that network routers are not involved in group membership.
- As a consequence, the connections between nodes in the overlay network may cross several physical links, and as such, routing messages within the overlay may not be optimal in comparison to what could have been achieved by network-level routing.

- A crucial design issue is the construction of the overlay network. In essence, there are two approaches (El-Sayed, 2003).
- First, nodes may organize themselves directly into a tree, meaning that there is a unique (overlay) path between every pair of nodes.
- An alternative approach is that nodes organize into a mesh network in which every node will have multiple neighbors and, in general, there exist multiple paths between every pair of nodes.
- The main difference between the two is that the latter generally provides higher robustness: if a connection breaks (e.g., because a node fails), there will still be an opportunity to disseminate information without having to immediately reorganize the entire overlay network.


End host
Router
A
1
50
C
1
Ra
Rc
7
5
40
Rd
Rb
1
1
Internet
D
B
Overlay network

- To understand the problem at hand, take a look at Fig. above, which shows a small set of four nodes that are organized in a simple overlay network, with node *A* forming the root of a multicast tree.
- The costs for traversing a physical link are also shown.
- Now, whenever *A* multicasts a message to the other nodes, it is seen that this message will traverse(travel across or through) each of the links *<.B,Rb», <Ra, Rb>, «Rc, Rd»,* and *<D, Rd»* twice.
- The overlay network would have been more efficient if we had not constructed an overlay link from *B* to *D*, but instead from *A* to C. Such a configuration would have saved the double traversal across links *«Ra, Rb>* and *<Rc, Rd>.*

Gossip-Based Data Dissemination

- An increasingly important technique for disseminating information is to rely on *epidemic behavior.*
- Observing how diseases spread among people, researchers have since long investigated whether simple techniques could be developed for spreading information in very large-scale distributed systems.
- The main goal of these epidemic protocols is to rapidly propagate information among a large collection of nodes using only local information.
- In other words, there is no central component by which information dissemination is coordinated.

- The gossip-based data dissemination protocol performs three primary functions on a Hyperledger Fabric network:

1. Manages peer discovery and channel membership, by continually identifying available member peers, and eventually detecting peers that have gone offline.
2. Disseminates ledger data across all peers on a channel. Any peer with data that is out of sync with the rest of the channel identifies the missing blocks and syncs itself by copying the correct data.
3. Bring newly connected peers up to speed by allowing peer-to-peer state transfer update of ledger data.