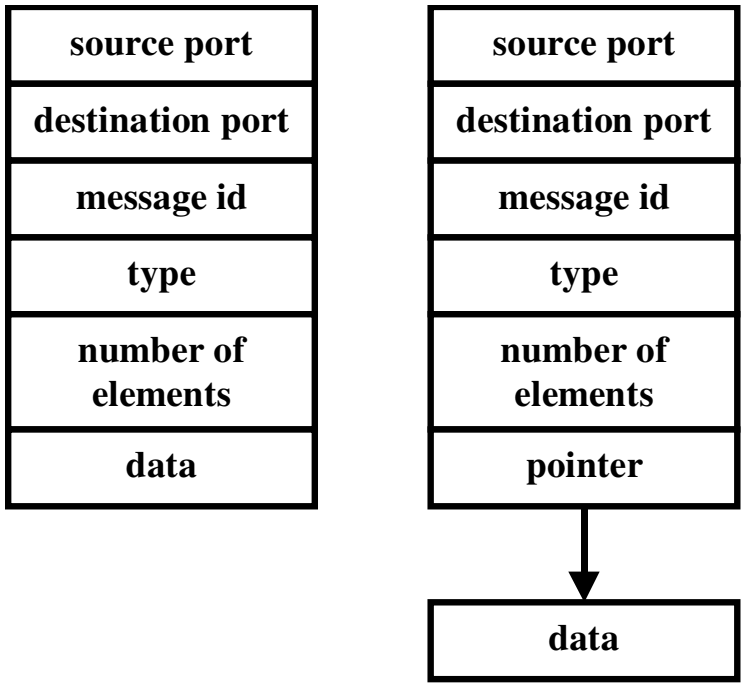
INTERPROCESS PRIMITIVES: MESSAGE PASSING

- in message passing communication the user is explicitly aware of the messages and mechanisms used to deliver and receive messages

MESSAGE PASSING

MESSAGE = typed collection of data objects consisting a fixed size header and a variable length body



message formats

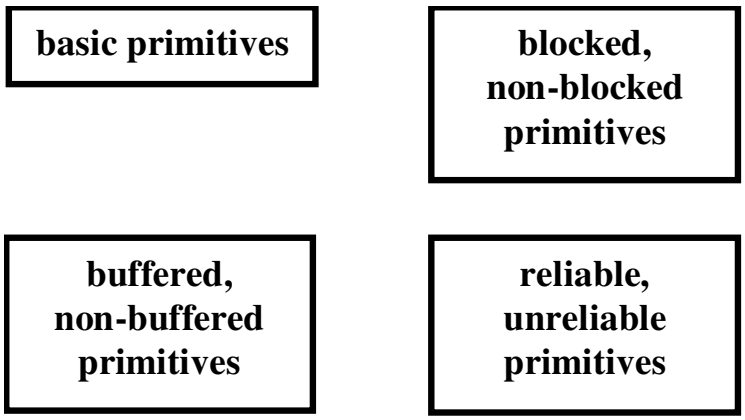
- in communication one process sends a message to an another process, and then waits for the reply,

- there are two basic issues of message passing communication:

1. **What set of communication primitives are used?**

2. **What are semantics of communication primitives?**

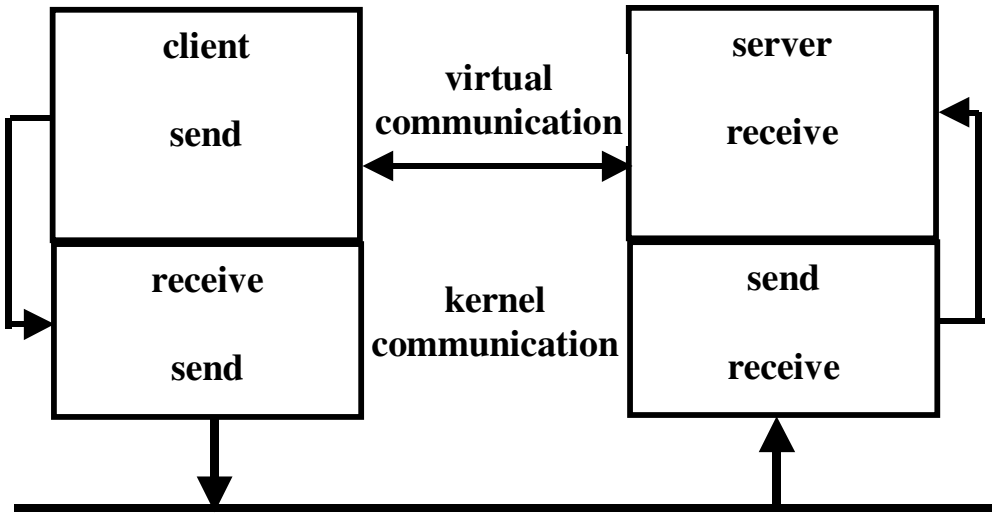
Message passing primitives



- in message passing communication the message is passed from the sending process to the sender's kernel, which transmits it to the communication network,

- next, the receiver's kernel accepts the message and forwards to the receiving process.

- the message reaches the destination's kernel, which delivers the message to the destination process,



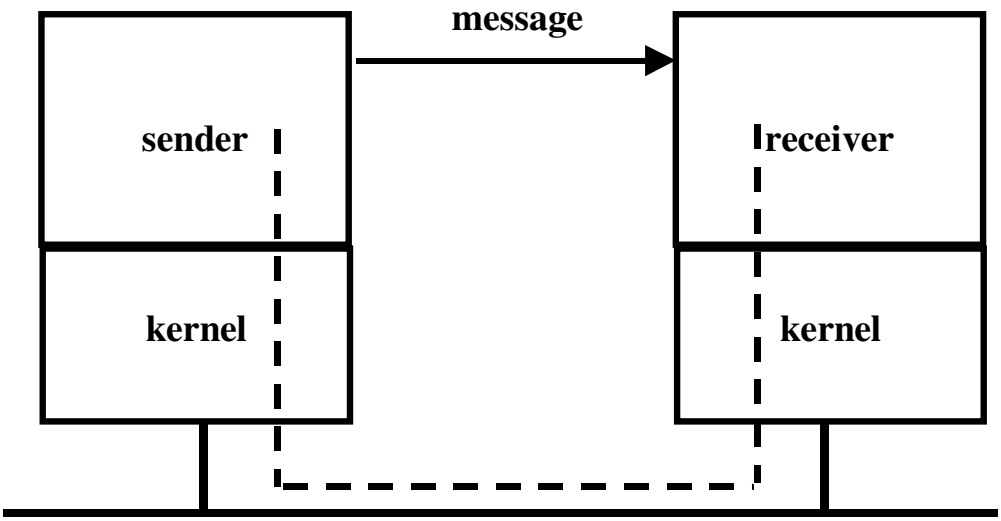
1. basic message passing primitives

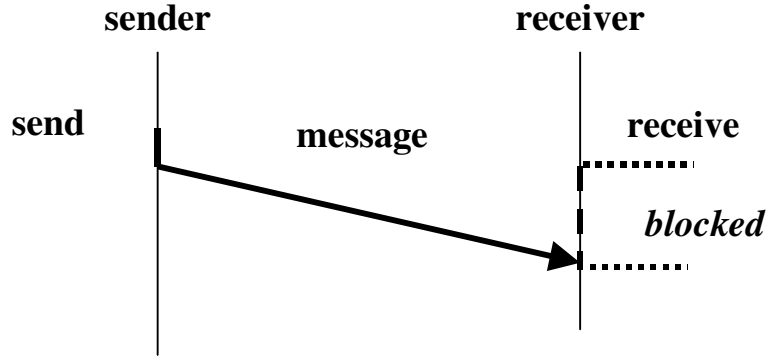
send(var, did)

- **the message contains the variables given by the variable list "var" and the destination address given by the destination identifier "did", where the message should be forwarded,**
- **a computer sends a message to an another computer for example a sender transmits message including variables "var" to the destination address given by "did",**

receive(var, sid)

- **the message contains the variables given by the variable list "var" and the source address given by the source identifier "sid", where the message came from,**
- **a computer receives a message from an another computer for example a receiver accepts a message, including variables "var", from the source address given by "sid", respectively**





2. buffered/non-buffered message passing primitives

- **in some communication systems message are non-buffered and in others are buffered between the time they are sent and received in order to speed up communication,**

- **there are three types of buffering:**

1. **buffer with unbounded capacity – sending process is never delayed thus, the sending process can get far ahead of a receiving process (asynchronous communication)**

2. **buffer with bounded capacity – execution of sending process depends upon the state of the buffer i.e. full or not full,**

3. **buffer with zero capacity – the sending process is always delayed until the corresponding receive is executed, then the message is transferred (synchronous communication)**

non-buffered primitives:

- **message is copied from one user buffer to another user buffer directly, thus a process issuing a send primitive should avoid reusing the user buffer until the message has been transmitted,**

- **in a system without a buffer, processes must be synchronised for a message transfer (synchronous communication),**

- **thus, the send operation is always delayed (or blocked) until the corresponding receive operation is not ready,**

- **there are two types of non-buffered (or synchronous) communication: local and remote rendezvous,**

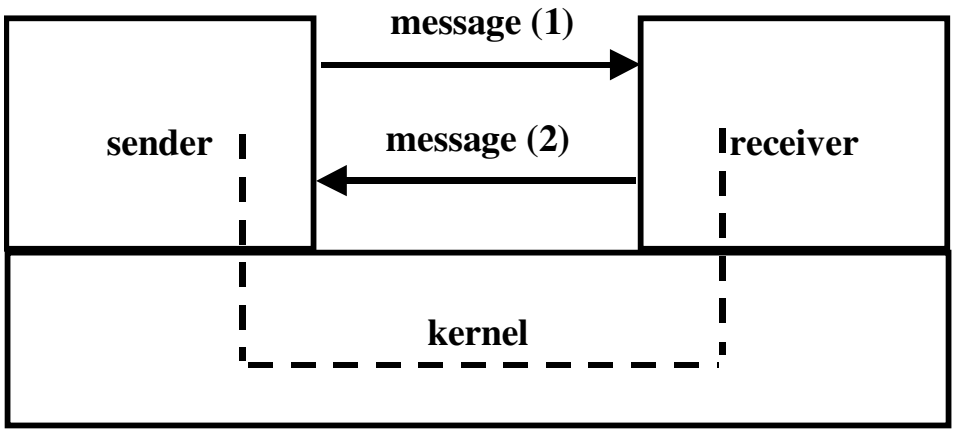
a. local rendezvous

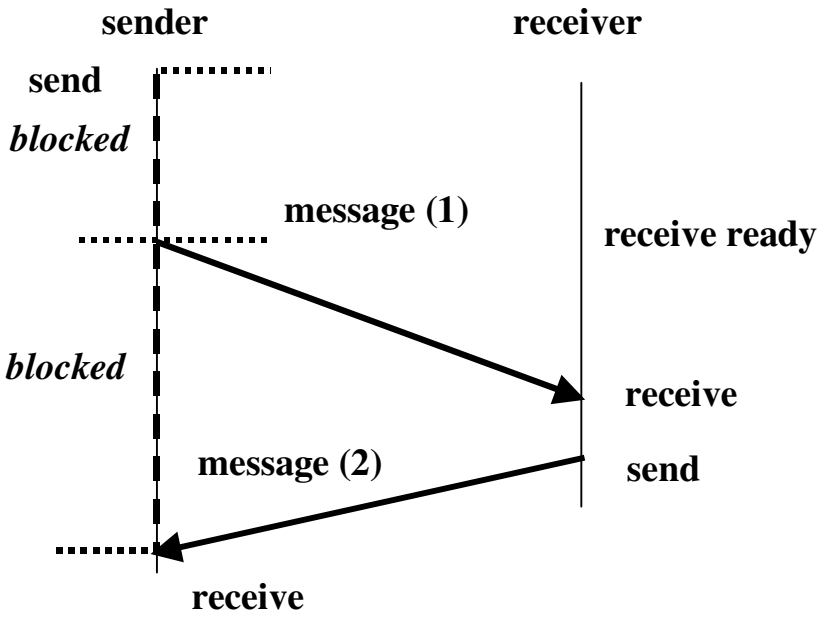
- **it supports the bi-directional communication between two processes located in the same address space,**

- **it occurs after invoking an entry by a process in the interface of an another process and the second process enables a call on that entry,**

- **the entry may have both input and output parameters passed by the**

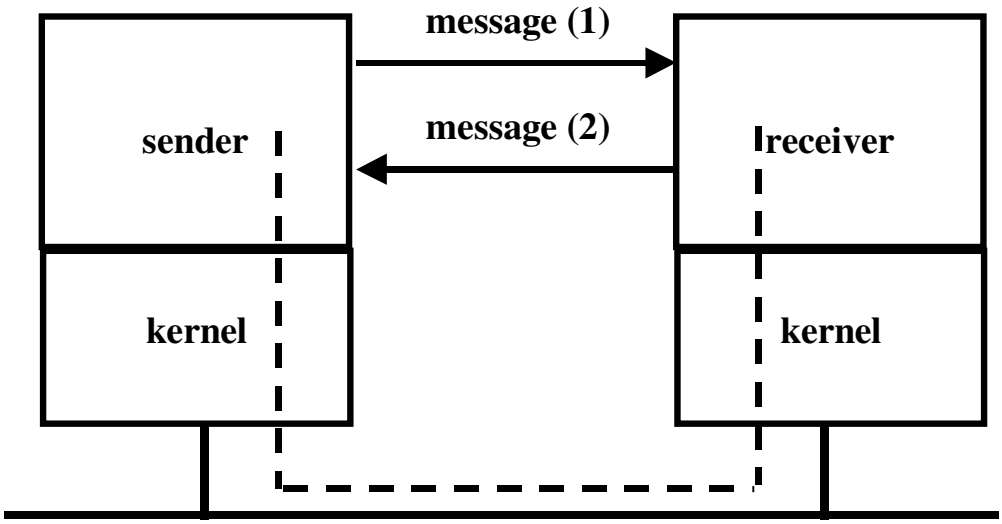
sender or the receiver, respectively at the beginning and the end of the rendezvous,

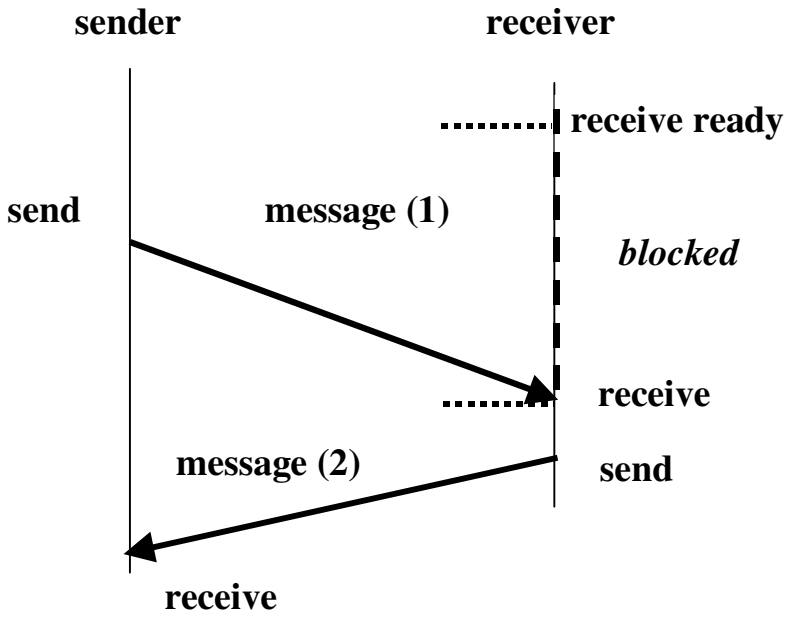




a. remote rendezvous

- **it is an extension of the local rendezvous for processes located in different address spaces,**
- **semantics of remote rendezvous are, as far as possible, the same as the semantics of the local rendezvous,**
- **to perform the remote rendezvous the process should have the address and the copy of the interface of the remote process,**
- **having this information the remote rendezvous is performed in the same way as a local rendezvous,**





non-buffered send(var, did)

- **the sending process copies the message to the internal buffer of the sending process,**
- **if a receiving process has not yet executed the receive primitives, the sending process is blocked until the receive primitive is completed**
- **after performing the receive primitive the message is copied from the sender to the receiver,**

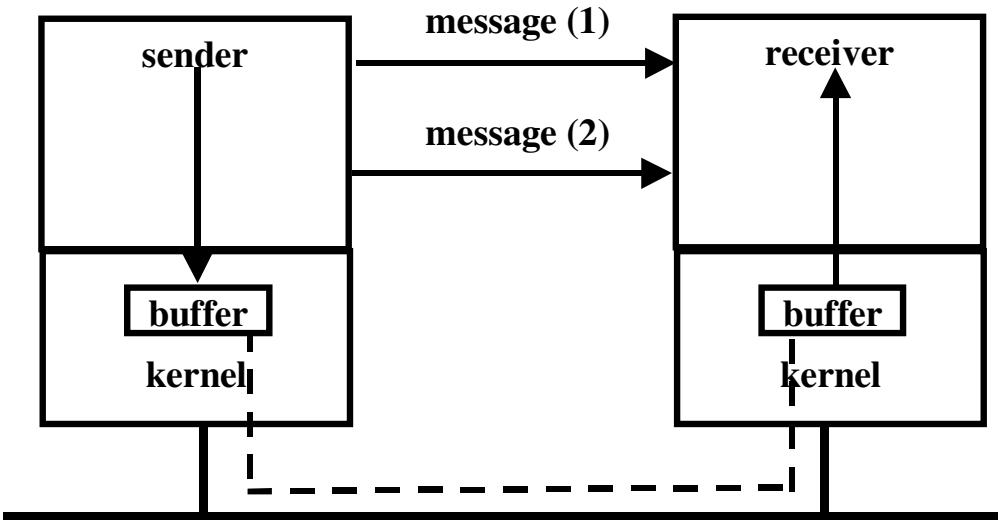
non-buffered receive(var, sid)

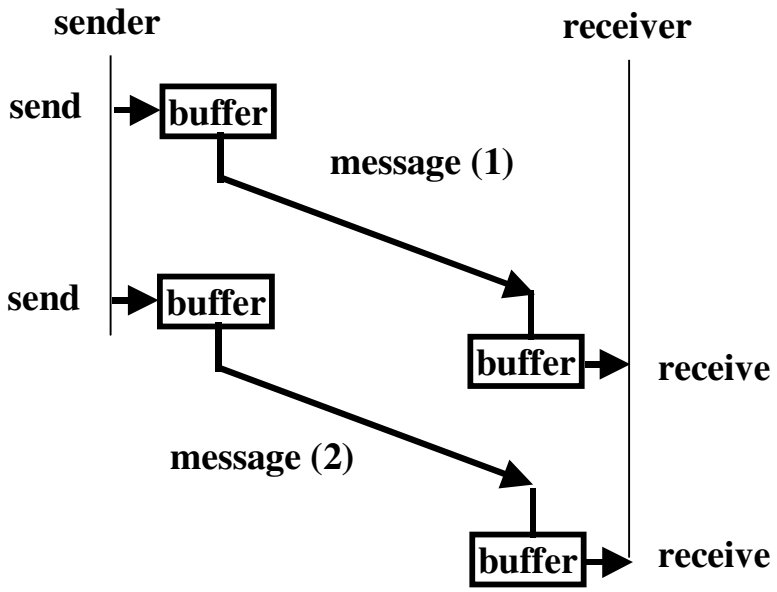
- **the non-buffered receive primitive is similar to the basic receive primitive**

buffered primitives:

- **there are buffers, which are placed in either process or kernel address space, or both of them with a limited capacity to deal with message passing,**

- **thus, message may be copied three times first, from the user buffer to the kernel buffer secondly, from the kernel buffer on the sending computer to the kernel buffer of the receiving computer thirdly, from this kernel buffer to a user buffer on the receiving computer,**





- **buffered message passing requires creation, management and destruction of buffers,**

- **senders and receivers run asynchronously thus, they generate and manage messages at different rates, buffers handle difference in rates**

and increase the message transfer throughput,

buffered send(var, did)

- **send primitive puts messages in a buffer until there is a space in the buffer,**
- **when the buffer is full, the send operation either may return a "code" to the sender informing that the buffer is full or may delay until there is a space in the buffer**

buffered receive(var, sid)

- **when no message is in the buffer, the receiver is either delayed or terminated with a "code" indicating that no message was available**
- **the primitives removes a message from the buffer and passes it the receiving process,**

advantages

- **sending process may perform multiple send primitive using the buffer, thus the sending process is not delayed**

disadvantages

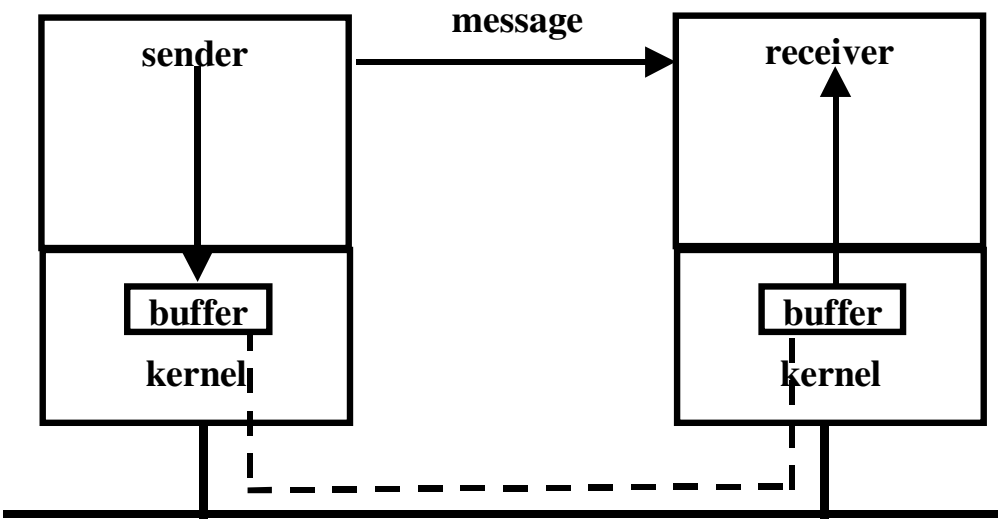
- **they more complex than non-buffered message passing because they require management of buffers,**
- **they generate protection problems**

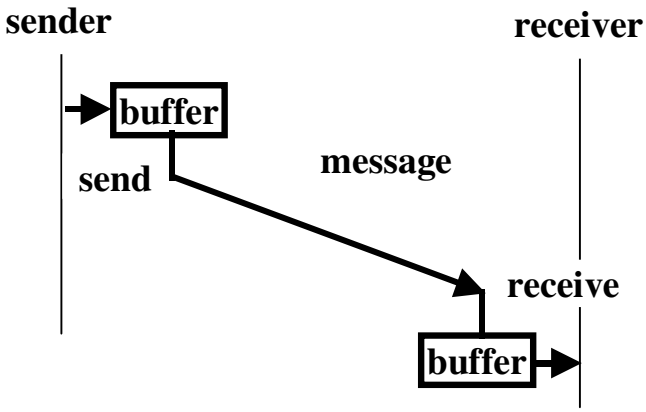
3. blocking/non-blocking message passing primitives

non-blocking primitives (or asynchronous primitives):

non-blocking send(var, did)

- **send primitive returns the control to the user program after the message has been put into the kernel buffer, and the sending process can continue performing its activities in parallel with message transmission,**
- **when the message has been transmitted from the buffer, the process is interrupted to inform it that the buffer is ready to accept an another message,**





non-blocking receive(var, sid)

- **receive primitive signals its intention to receive a message and provides a buffer to copy the message,**

- **the receiving process may either periodically check for the arrival of a message or be signalled by interrupt upon arrival of a message,**

- **when a message is arrived to the buffer the primitive moves the message from the buffer to the address space of the receiver process**

advantages

- **they provide maximum flexibility for users,**

- **the sending process can continue processing in parallel with the message transmission,**

disadvantages

- **they make programming difficult, especially in real-time programming environment,**

- **they prevent access or change to message contents before or during transmission,**

blocking primitives (or synchronous primitives):

- **they combine the data transfer with the synchronisation of sending and receiving processes,**

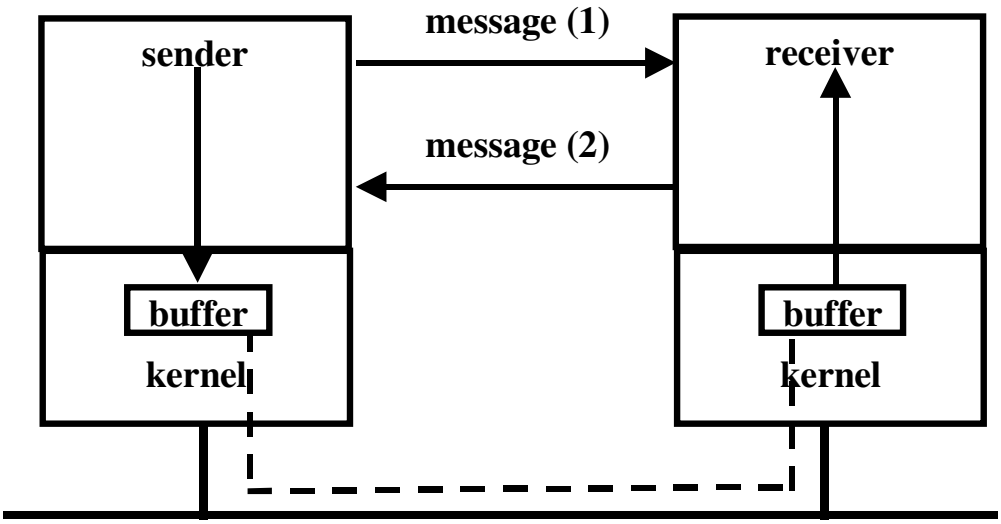
blocking send(var, did)

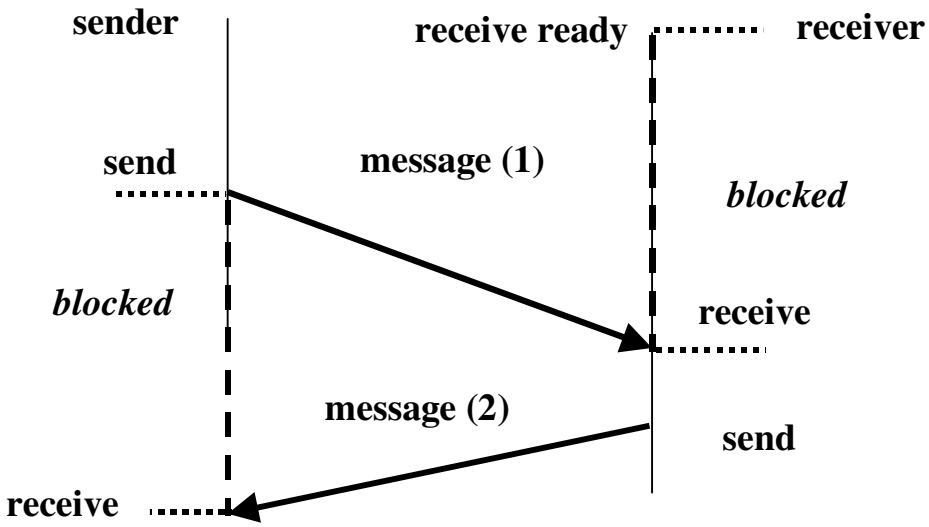
- **the primitive does not return the control to the user program, i.e. it is blocked, until a request message has been transmitted to the network (blocking send, - the process may modify the buffer) or until a message has been received by the destination's kernel (reliable blocking send - the process may modify the buffer) or until a reply message has been received (explicit blocking send),**

blocking receive(var, sid)

- **receive primitive does not return the control until a message has been placed in the buffer,**

- **the receiving process has nothing else to do while awaiting a message,**





advantages

- **they make programming easy because relation among sending and receiving processes are well-defined,**

disadvantages

- **they disable parallelism i.e. a process blocked in a send and a receive operation cannot do any useful work,**
- **they do not enable a process to be blocked waiting for more than one kind of event,**

- **in many cases non-blocking send and blocking receive is combined because the sending process can continue its activity after the message has been composed and copied to the sender's kernel; the receiving process needs to wait for the message arrival to do its work,**

4. reliable/unreliable message passing primitives

- **communication system failure can cause severe consequences and damages in DSs to manage them reliable primitives are used,**

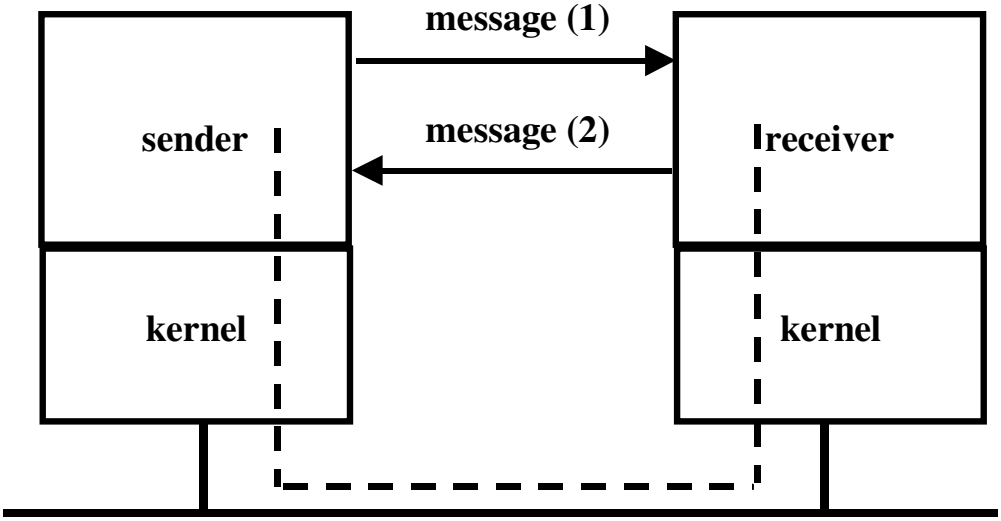
unreliable primitives:

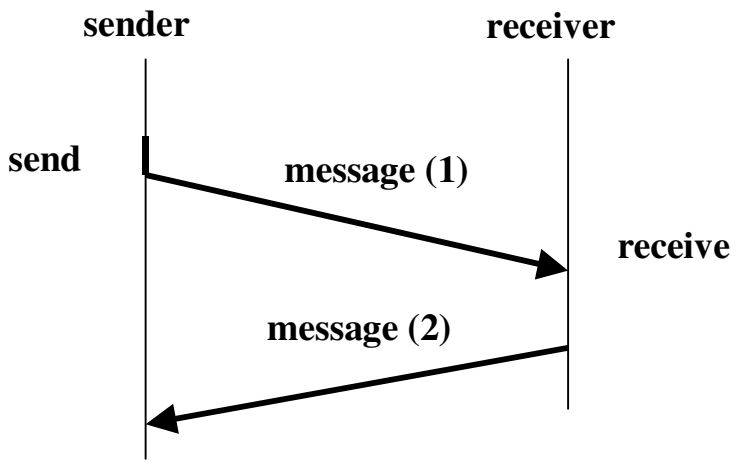
unreliable send(var, did)

- **the primitive sends a message to the network and there is neither automatic retransmission nor guarantee of delivery**

unreliable receive(var, sid)

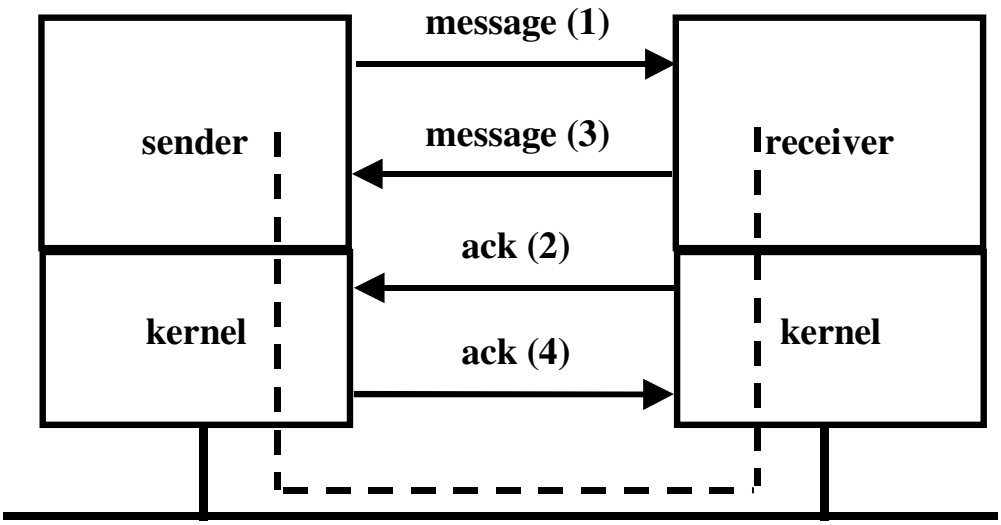
- **the receiver does not send any acknowledgement to confirm the message reception**

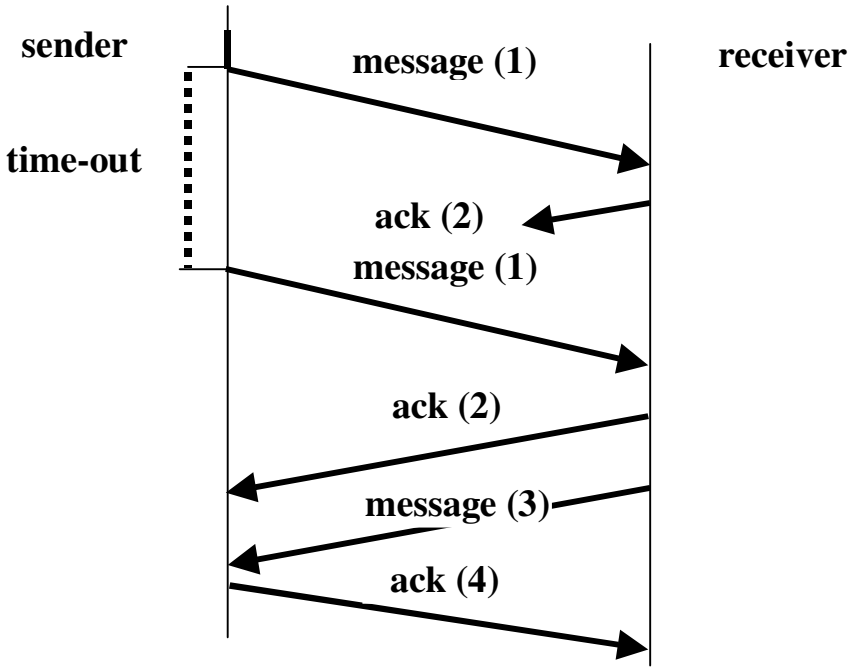




reliable primitives:

- **if the sender requested any data, the receiver sends a response, thus the sender has to set a receive primitive up to receive any messages from the receiver**





reliable send(var, did)

- **the primitive manages lost messages using internal retransmissions and acknowledgements, which are handled at the kernel level, using time-outs**

- **as a result, when the send primitives terminates, the sender is sure that the message was received and acknowledged**

reliable receive(var, sid)

- **the primitive automatically sends an acknowledgement confirming the message reception,**

advantages

- **they prevent information losses produced by communication problems**

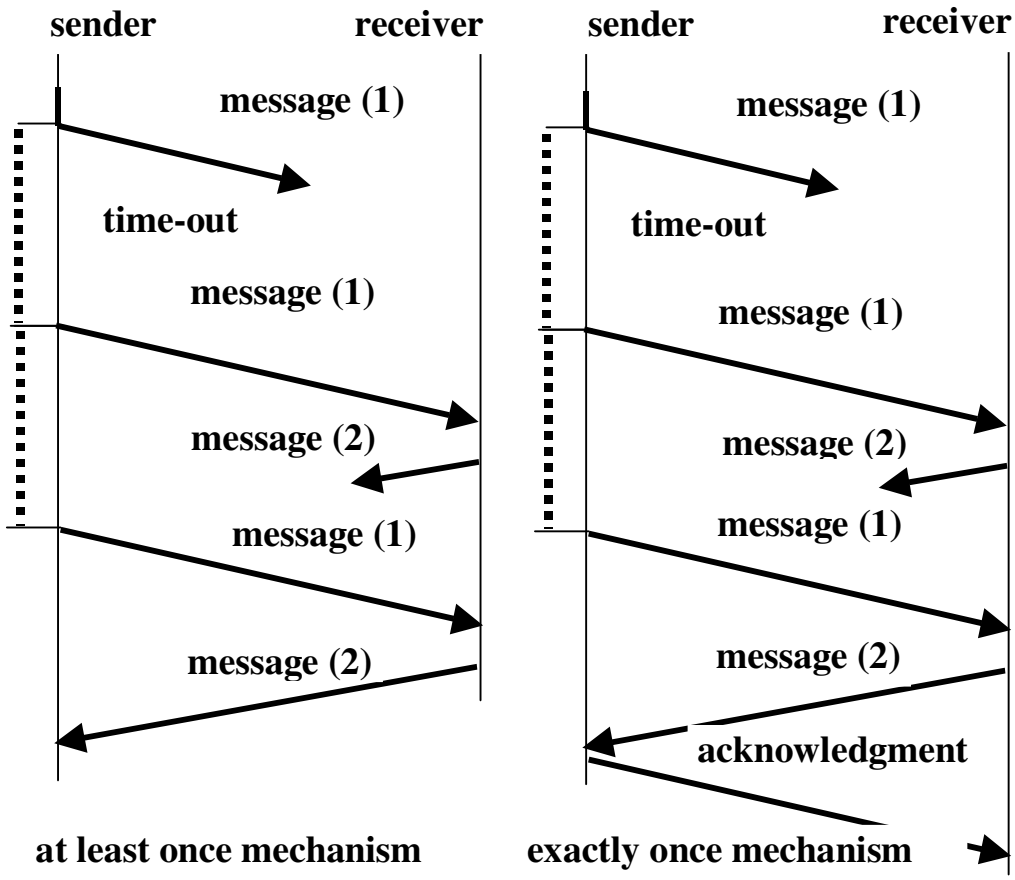
recovery mechanism:

a. at least once recovery mechanism

- it repeats the processing of the message, even if only one execution was described

b. exactly once recovery mechanism

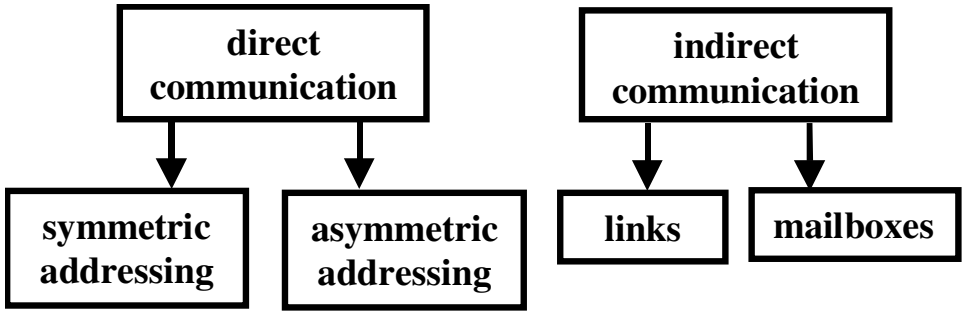
- it ensures that the receiver's operation is only once executed in order to avoid repeated execution that could destroy the consistency of information



Addressing in message passing

- in message passing two addressing approaches are used:

- **direct communication – the sender designates a fixed destination process,**
- **indirect communication – the sender designates a fixed location for a receipt of a message,**



1. direct communication

- a communication path exists between a pair of processes, which is completely transparent and there is no notion of connection in the message transfer,

symmetric addressing:

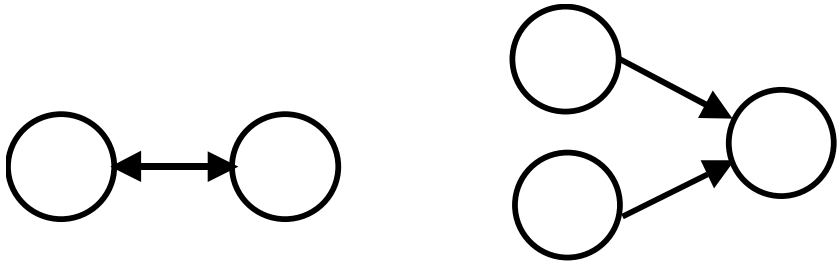
- **the addressing is symmetric because each process that participates in communication must explicitly name the sender and receiver of communication,**
- **a global process "id" contains the network host address and local process "id",**
- **only one communication path exists between any pair of sending and receiving process**

send(proc, mes)

- **the primitive sends a message "mes" to a process "proc",**

receive(proc, mes)

- **the primitive receives a message "mes" from a process "proc"**



symmetric addressing

asymmetric addressing

asymmetric addressing:

- **only the sending process names the receiving process because it is sometimes convenient to receive messages from unknown processes,**
- **the addressing is asymmetric because only the sending process should indicate the receiving process,**

send(proc, mes)

- **the primitive sends a message "mes" to a process "proc",**

receive(sid, mes)

- **the primitive receives a message "mes" from a process given by the identifier "sid"**

advantages

- **simple communication management,**

disadvantages

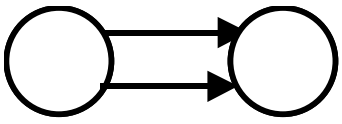
- **it allows the communication between only one client and server, i.e. it does not support sending one request to more than one server,**
- **changing the name of the process may require the modification of all references made to the old process,**

2. indirect communication

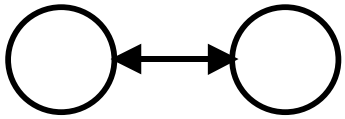
- direct addressing provides one bi-directional communication path between each pair of communicating processes, in contrast indirect communication is able to identify multiple paths,

link:

- **one-way communication channel in connection-oriented communication systems, which is protected object, that is provided for user process, system process or kernel,**
- **multiple communication points in a process can be identified by using multiple links,**
- **one-way communication path is asymmetric because first, only the sending process has to keep a link to the receiving process secondly, receiving process must hold the a link to the sending process only for the period of time from receiving the message until a response is sent,**
- **each process has a table of links associated with it, each entry of this table contains a communication path available to the process,**
- **messages sent through a link are mapped to a network communication path and delivered to the remote computer,**
- **the remote computer naps the message to the input link in the receiving process,**



one-way link



two-way link

<u>advantages</u>

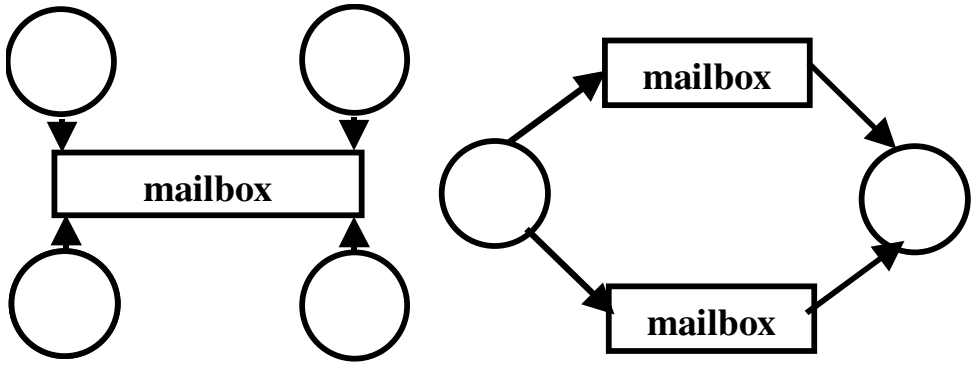
- **provides reliable and sequenced message delivery,**

<u>disadvantages</u>

- **overhead of link management is substantial,**

mailbox (port):

- **destination designator in send and source designator in receive operations,**
- **mailboxes are global data structures shared by some sending (producer) and receiving (consumer) processes,**
- **mailboxes require synchronisation across the communication network, which is a difficult task,**



multi-point communication **multi-path communication**

- **ports are special mailboxes, which are protected kernel objects, into that messages may be placed by processes and from that messages may be removed,**
- **each port has a FIFO queue of finite length, i.e. messages have been sent to a port, but which have not yet been removed from it by a process are in the queue,**
- **messages can be appended to or removed from the queue by send and receive operations through a communication path,**
- **ports are similar to links except that they are bi-directional and buffered**
- **processes may communicate with other processes by using different ports,**

send(port, mes)

- **the primitive sends a message "mes" to a port "port",**

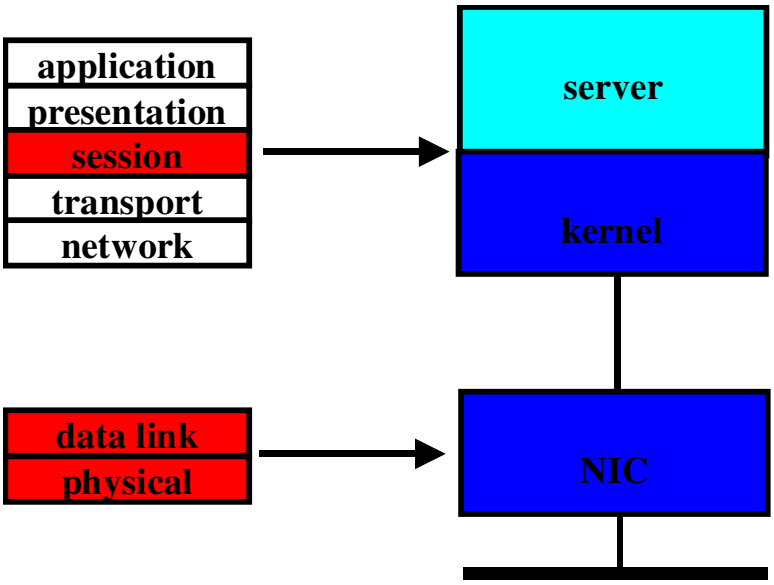
receive(port, mes)

- **the primitive receives a message "mes" from a port "port"**

CLIENT-SERVER COMMUNICATION

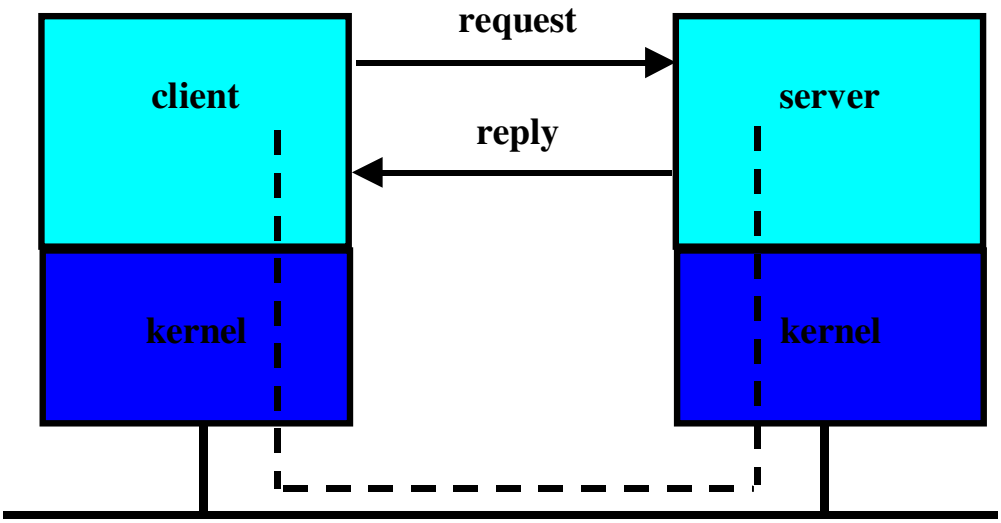
servers = a group of co-operating processes that provide OS services for clients

clients = user processes that request services from servers



- to avoid overhead of the connection-oriented protocols, the client-server model is based on a simple connectionless request/reply protocol,

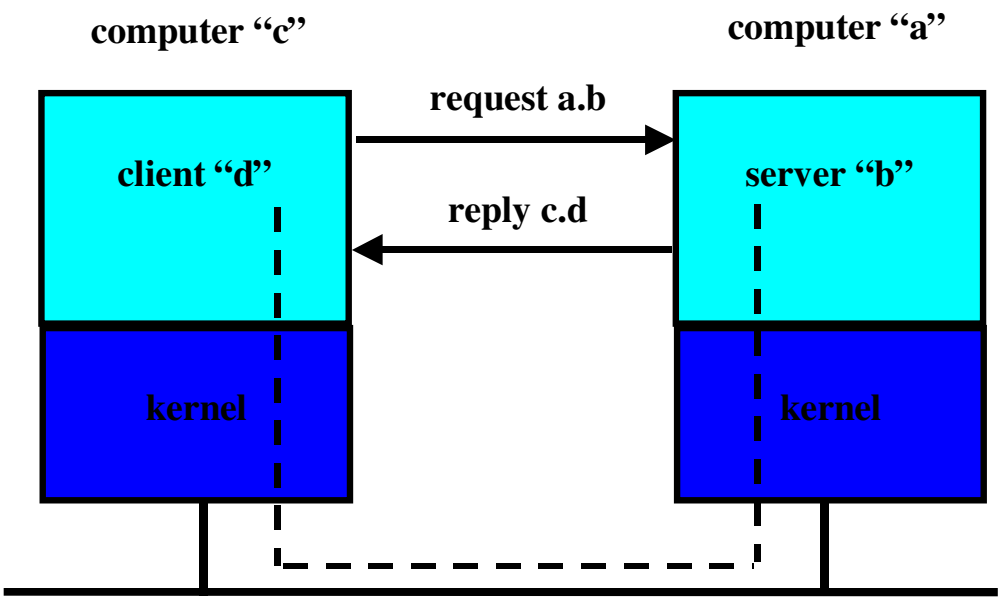
- i.e. the client sends a request message to a server to perform some service, next the server performs the service and returns the data requested,



Addressing in client-server communication

1. "simple" addressing

- address has two parts, the first one specifies the computer and the second one the process,
- the computer number is used by the kernel to get the message from other computers and the process number is for identification which process the message is for,



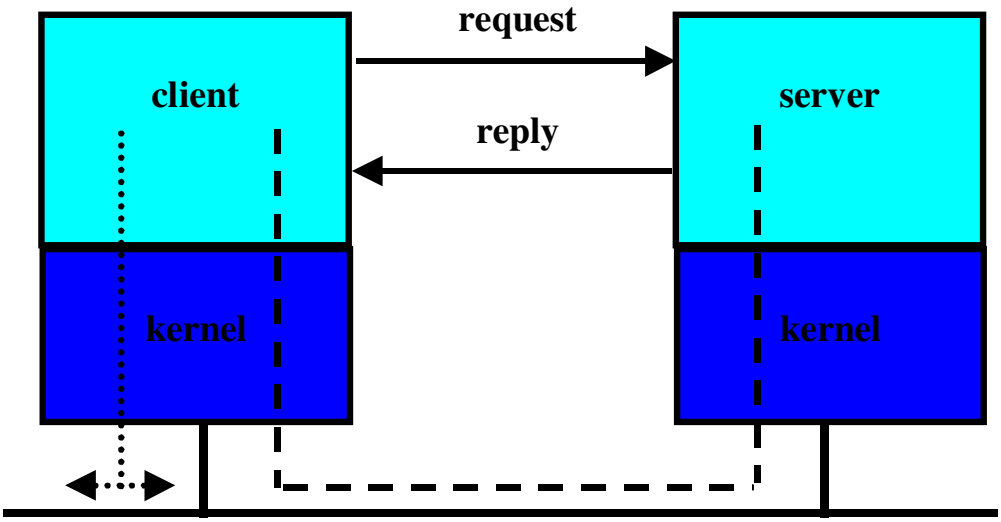
- this addressing mode is not transparent because the destination, i.e. the location of the computer, is given directly in the address,
- if the server is down, processes using direct addressing cannot access any other server,

2. broadcast based addressing

- each process picks a process identifier from a large sparse address space for example from the space of 64-bit binary integers,
- the sender broadcasts a special locate message containing the address of the destination process,
- all the kernels check to see whether the address is theirs, and if so they send back a "here I am" message giving their network address (node or

computer number),

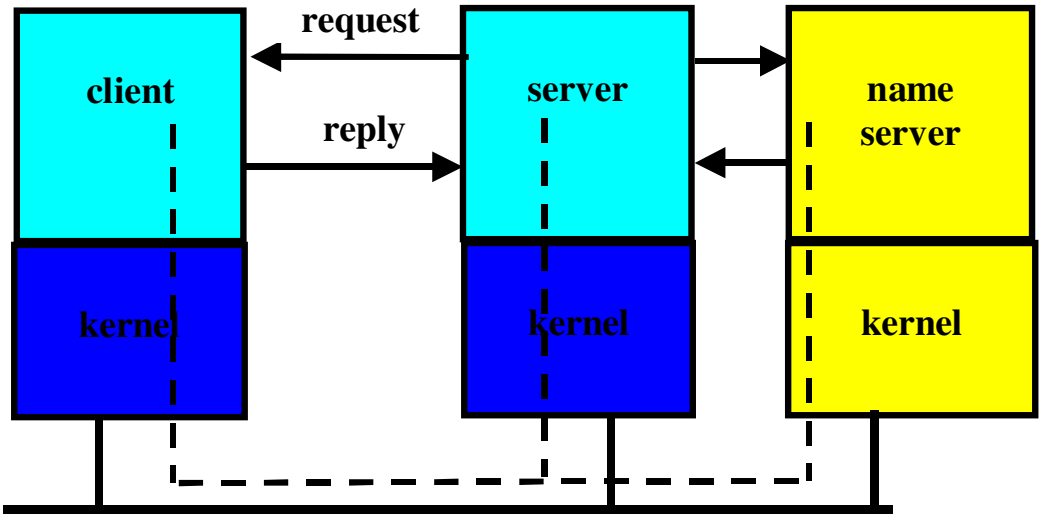
- the kernel of the sending process uses this address to access to corresponding server,



3. addressing based on name server

- on the first attempt to use a server a client sends a query message to a special server called name server, asking it for the node number which is the address of the node,

- having the address of the node, the request of the client can be sent directly to the corresponding server



Communication in client-server systems

- client-server message exchange is performed by a request-reply message pair, which requires one send and one receive operation at both the client and server side,

- there are the following types of messages in the client-server communication:

request **= requests a service from a server,**

reply **= sends a reply to a client after performing a service**

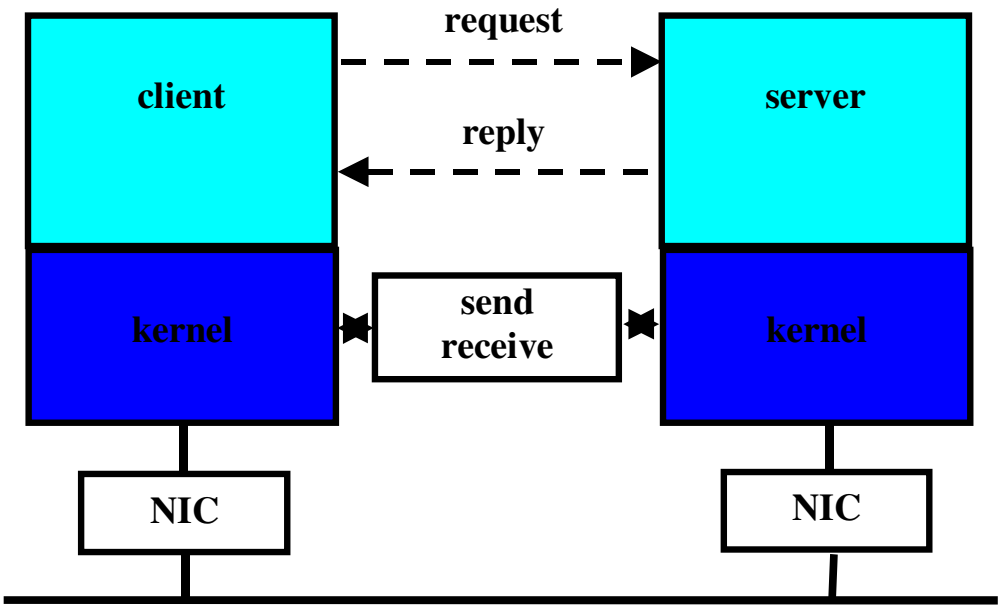
acknowledgement **= confirms arrival of the previous message,**

are you alive **= checks whether a server is off/on,**

i am alive **= confirms that a server is on,**

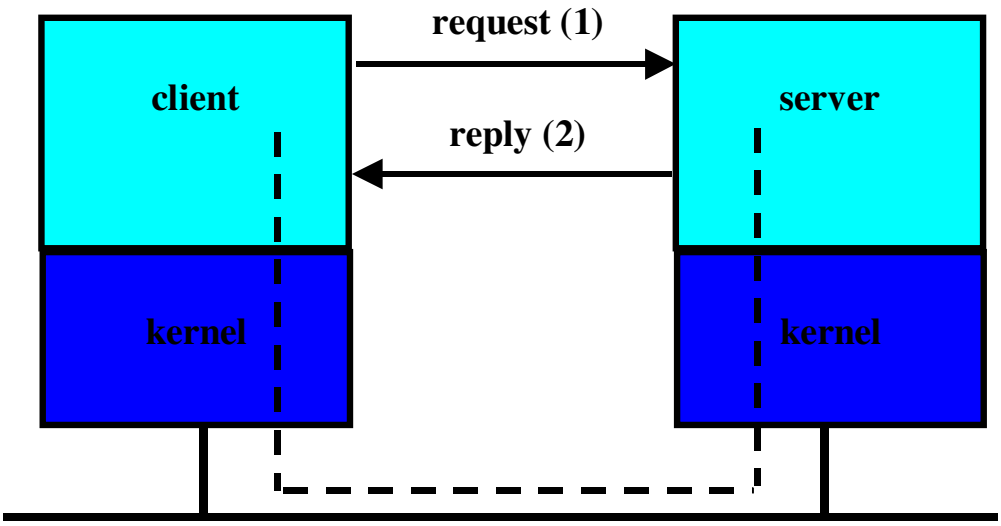
try again **= buffer of a server is full and try again later,**

address unknown **= no process is assigned to the address**



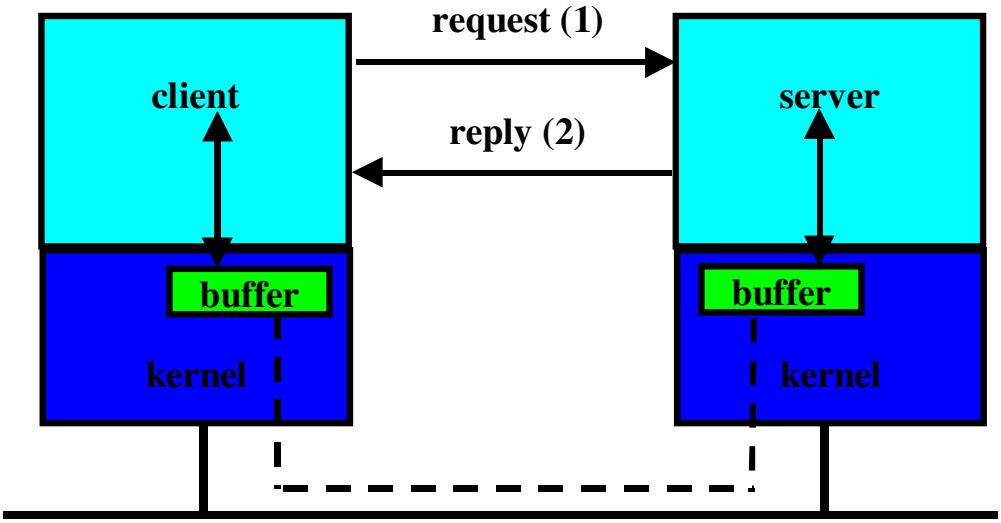
non-buffered communication

- **server should perform a receive operation before client perform a send operation because neither has a buffer,**



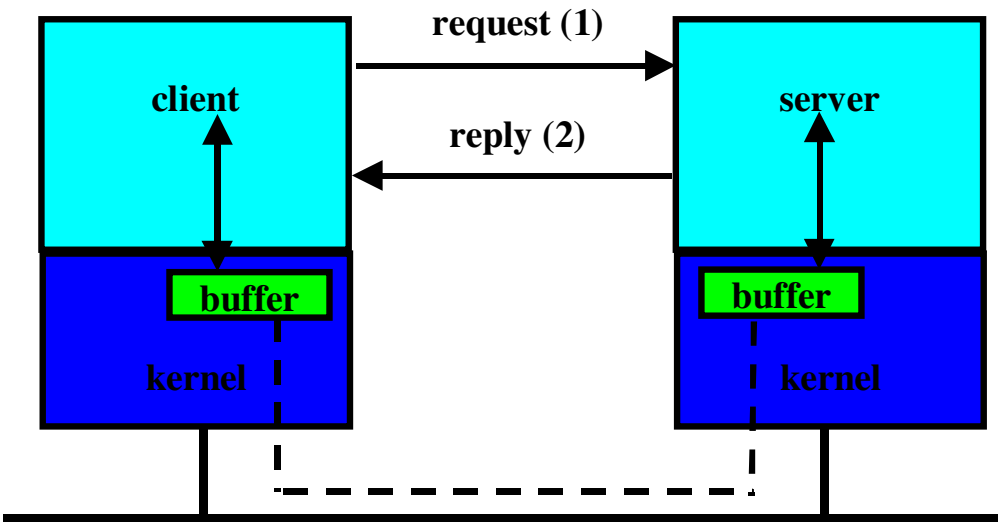
buffered communication

- **message is sent to the kernel buffer (mailbox) by the client process,**
- **the message forwarded from client buffer through the communication network to the server buffer,**
- **the kernel removes the message from the kernel (server) buffer (mailbox) to the server process,**



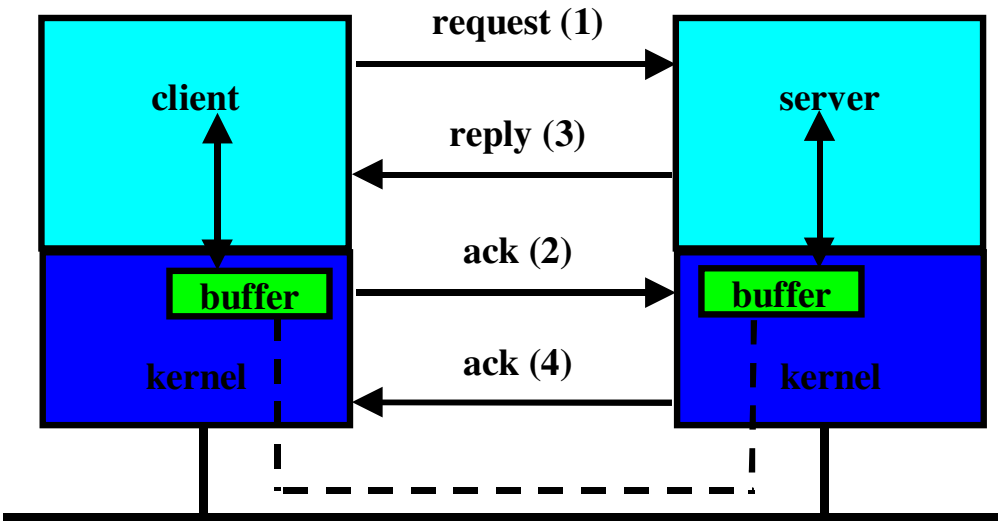
unreliable communication

- **neither the request nor the reply message is not confirmed,**
- **if either request or reply messages are lost, recovery should be done at the client-server level,**



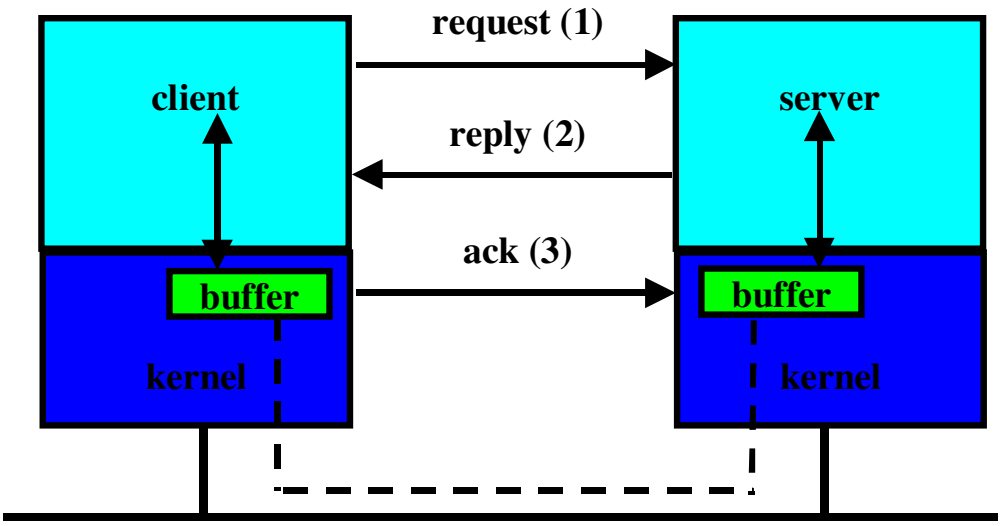
reliable communication

- **both request and reply messages are acknowledged by the kernel in order to handle communication failures,**
- **timeout and retransmission manage recovery of lost messages at the kernel level**



- **to decrease burden of the communication network, the reply message**

serves as an acknowledgement to the request and the client process sends an acknowledgement message to confirm the completion of a particular service,



GROUP COMMUNICATION

Group

- collection of processes that act together by message passing in distributed systems,

- groups allow processes to deal with collections of processes as single abstraction,

- when a message is sent to the group, all members of the group receive it because it is one-to-many communication,

- a process can send a message to a group of servers without having to know how many there are or where they are,

- a process can join a group or leave a group, it can be a member of several groups at the same time,

Properties of group communication

1. atomicity

- **each server receives all requests thus, all the servers provide the same service and they are in the same state at any time,**
- **when any process sends a message to the group, it does not to have worry about what to do if some of them do not get it,**
- **it makes programming of distributed systems easier because it does not have to worry about what to do if some processes do not get it,**

atomic communication algorithm:

1. **sending process send a message to all members of the group**
2. **when a process receives a message, if it has not yet received it, then sends the message to all members of the group, if it has already received the message, then it is discarded,**

2. message ordering

- **replicated servers may all need to handle requests in the same order, even with multiple clients,**
- **without message ordering when two processes multicast to a group at about the same time, their message may not arrive in the same relative order at members of the group,**
- **atomic and reliable multicasts preserve message order between any process pairs,**
- **types of ordering**

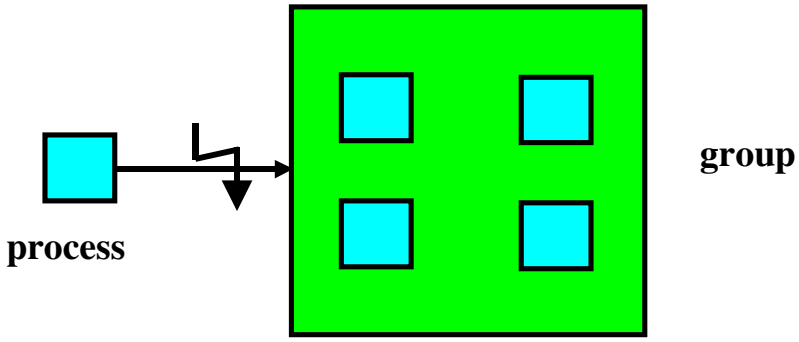
  a. totally (global) ordered multicast: all messages (from all senders) are received in the same order at all members of the group as they were sent

  b. consistent time ordered multicast: if two messages are sent close together in time, the system picks one of them as first and delivers it to all group members, followed by the other

Types of groups

1. closed groups

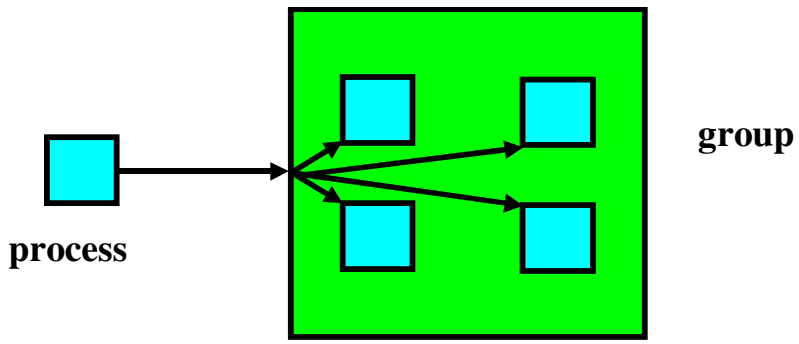
- **processes out of the group cannot send messages to the whole group, although they are able to forward messages to individual members of the group,**
- **only processes of the group can send messages to the group,**
- **closed group is applied to parallel processing, for example a collection of processes working together, which have their own goal,**



closed group

2. open groups

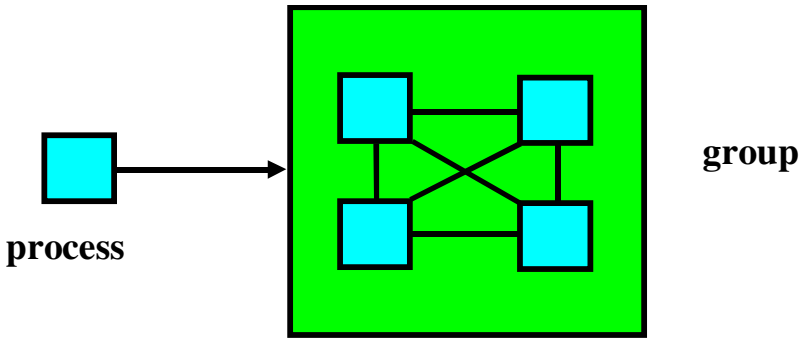
- **any process in the system can send messages to any group because there is free access to members of open groups,**



open group

3. non-hierarchical (or peer) groups

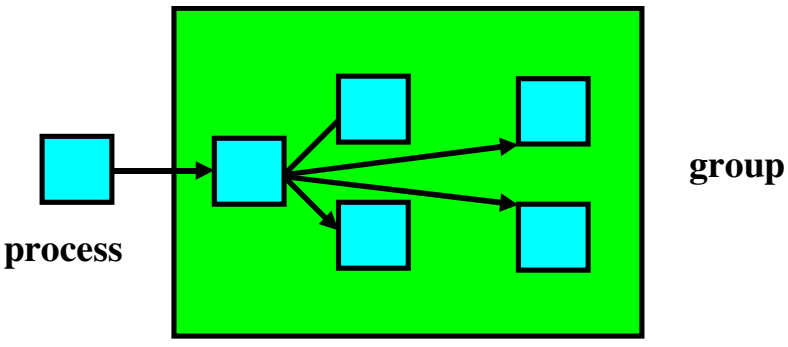
- **all the processes in the group are equal and all decisions are made collectively,**
- **have no single point of failure, i.e. of one of the processes crashes, the group simply becomes smaller, but can further work,**



non-hierarchical group

4. hierarchical groups

- **some kind of hierarchy exists i.e. one process is the co-ordinator and all others are workers,**
- **when a request is generated it is sent to the co-ordinator, then the co-ordinator decides which worker is best suited to complete it, and forwards it there,**
- **the failure of the co-ordinator brings the entire group to halt,**



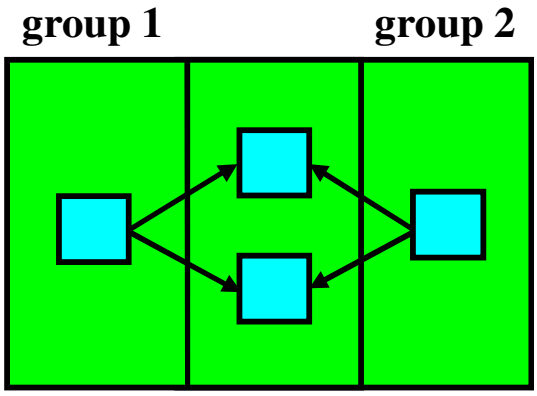
co-ordinator

hierarchical group

5. overlapping groups

- **a process can be a member of multiple groups at the same time,**

  problem: inconsistency when processes want to send messages to several groups



overlapping group

Group membership

- to add, manage and delete processes within a group, a group server is used, which can perform all these operations,

- there are two types of group servers:

1. group server (centralized approach)

- **it creates and deletes groups, and allows processes to join and leave groups**

- **the group server maintains a database of all the groups and their members,**

2. group server (distributed approach)

- **a process sends a "hello" message to all group members before joining a group, and a "goodbye" message before leaving the group,**

Group addressing

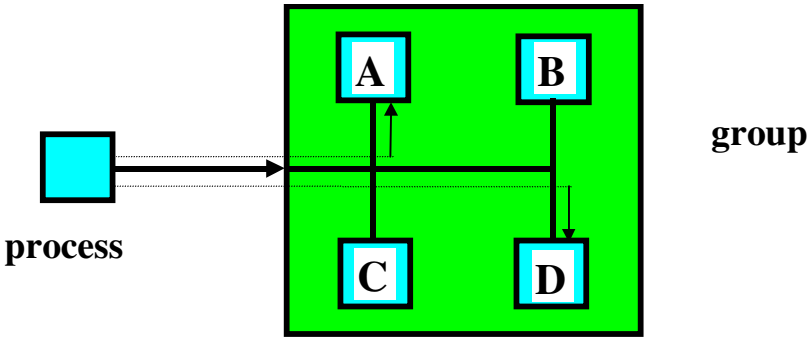
- a process has to access to a group using group address, which is a unique identifier,

1. each group has a unique group address

- **a process sends a message to a group address and it is delivered to all the members**

a. unicast addressing

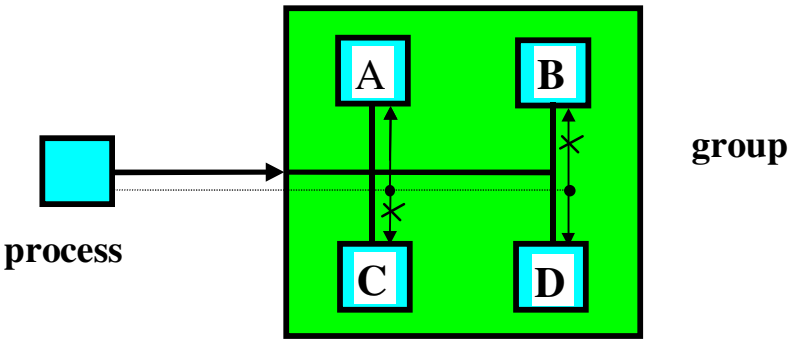
- **the kernel sends a point-to-point message to each process of the group having the list of processes belonging to the group,**



unicast

b. broadcast addressing

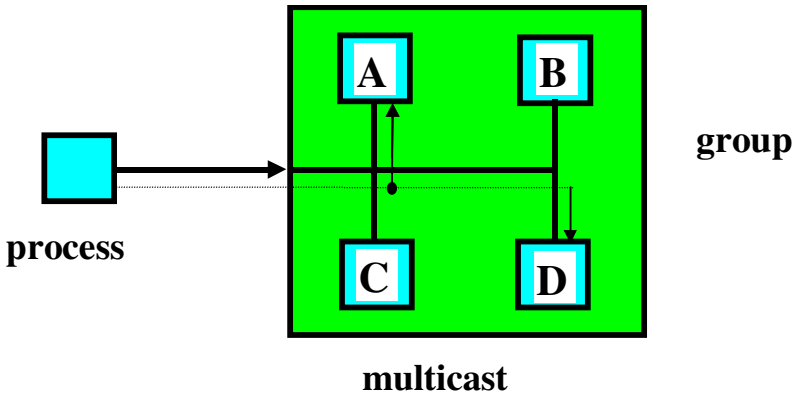
- **each kernel gets the message and extracts the group address, if none of the processes is a member of the group the message is discarded**



broadcast

c. multicast addressing

- **there is a special network address, to which multiple computers can listen,**
- **the kernel sends a message to all those nodes that need it, and no others,**



types of multicasts:

a. <u>atomic multicast:</u> a message is either received by all members or none of them. (failed processes lose their membership.)

b. <u>reliable multicast:</u> message transmission makes the best effort to deliver to all members of a group but does not guarantee to do so,

- **advantages of multicast messages:**

1. fault-tolerance based on replicated servers

- clients requests are multicasted to all the members of the group, each which performs an identical operation

2. locating objects in distributed services

- multicast messages can be used for locating objects within a distributed service

3. better performance through replication (updates are multicast),

- data are replicated to increase the performance of the service,

- each time the data changes, the new value is multicast to the processes managing the replicas

4. multiple update

- multicast to a group may be used to notify processes when something happens

2. the caller provides an explicit list of all destination