Distributed Computing (2020)

Communication

Outline of presentation

1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

Desirable Features of a Good Message Passing System

a) Simplicity
b) Uniform Semantics
c) Efficiency
d) Reliability
e) Correctness
f) Flexibility
g) Security
h) Portability

Issues in IPC by Message Passing

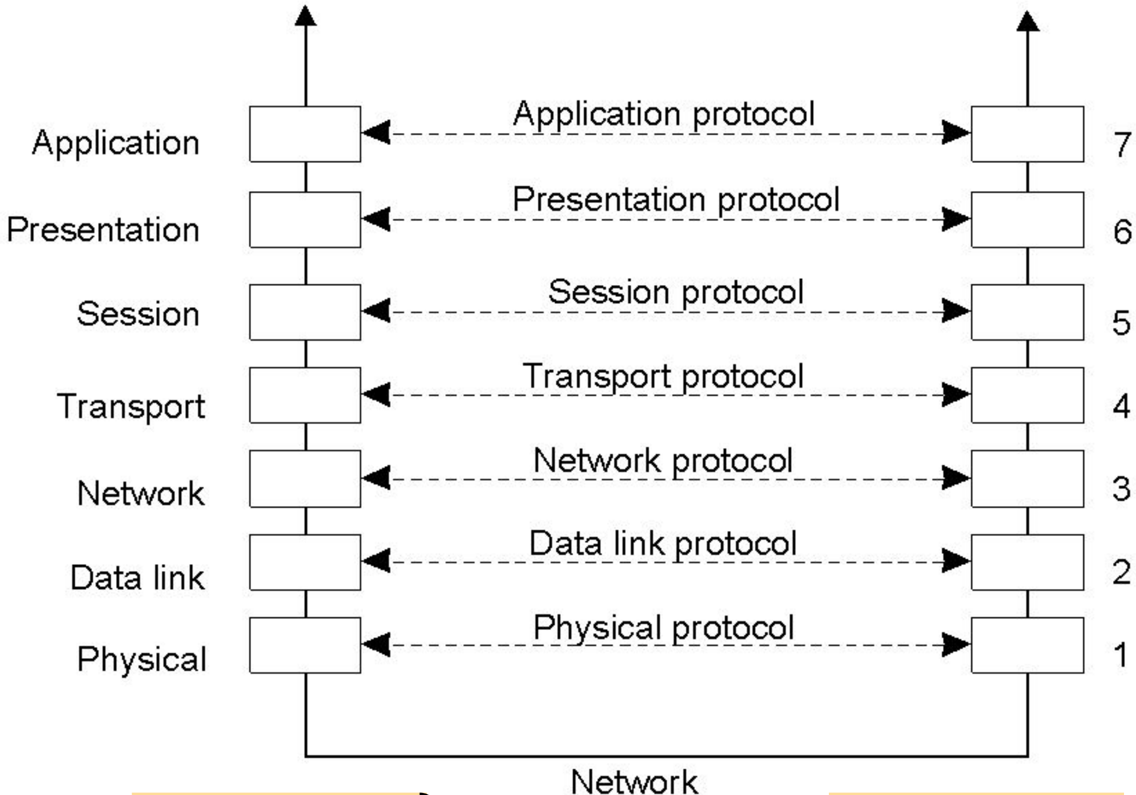
a) Address
b) Sequence No.
c) Structure information
d) Synchronization
e) Buffering

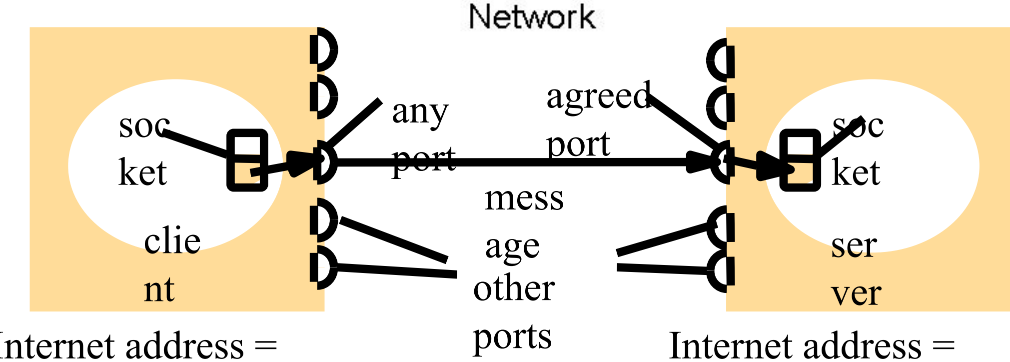
Outline of presentation

1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

Layered Protocols (1)

Layers, interfaces, and protocols in the OSI model.

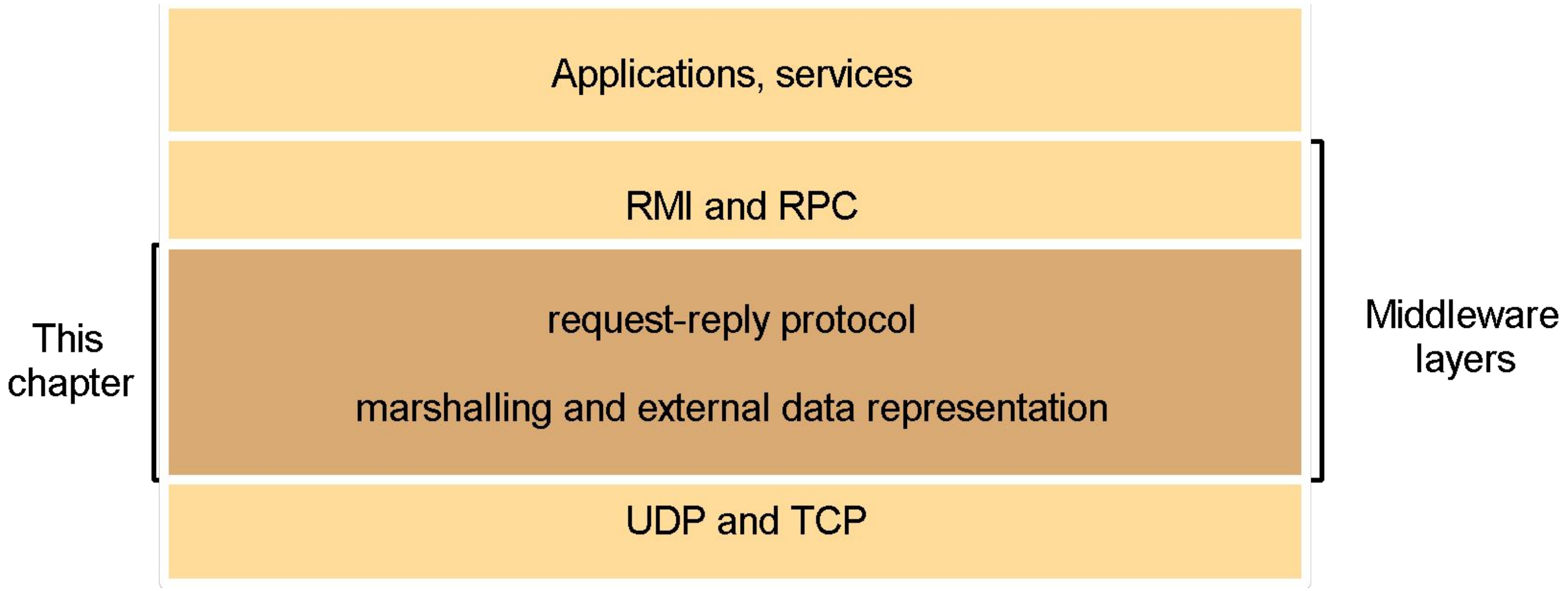




Internet address = 138.37.94.248

Internet address = 138.37.88.249

Middleware layers



Outline of presentation

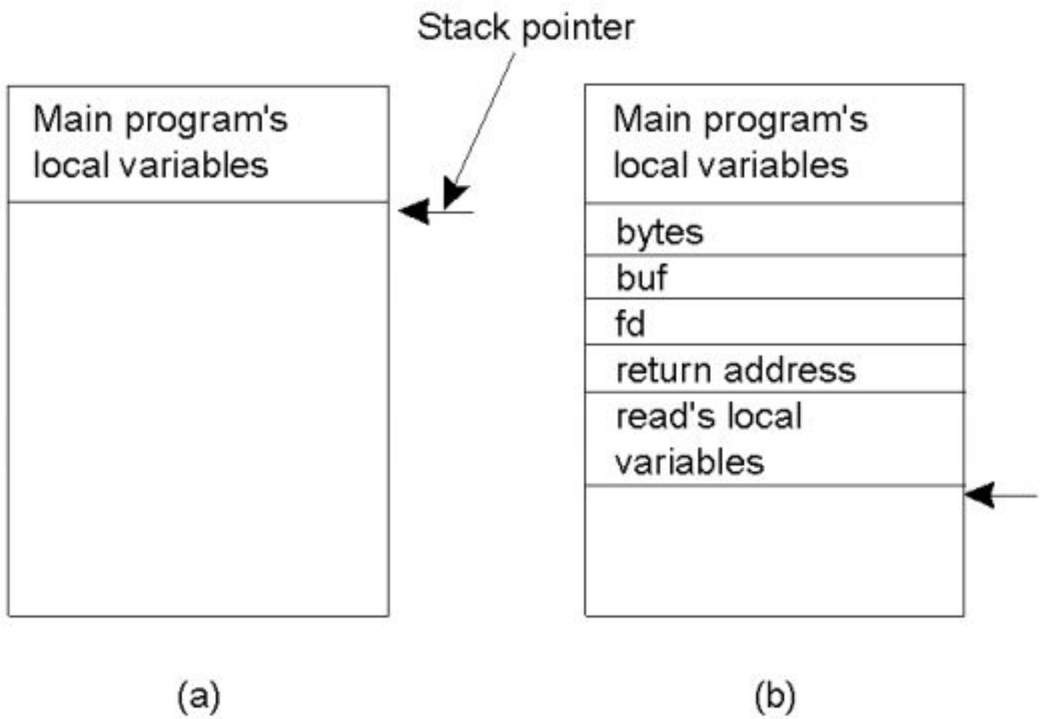
1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

- In Distributed systems: the callee may be on a different system –Remote Procedure Call (RPC) –NO EXPLICIT MESSAGE PASSING
- Goal: Make RPC look like local procedure call

- Parameter passing
- Binding
- Reliability/How to handle failures –
- messages losses –client crash –server crash
- Performance and implementation issues
- Exception handling
- Interface definition

Conventional Procedure call

count=read(fd,buf,nbytes);



a) Parameter passing in a local procedure call: the stack before the call to

b) The stack while the called procedure is active

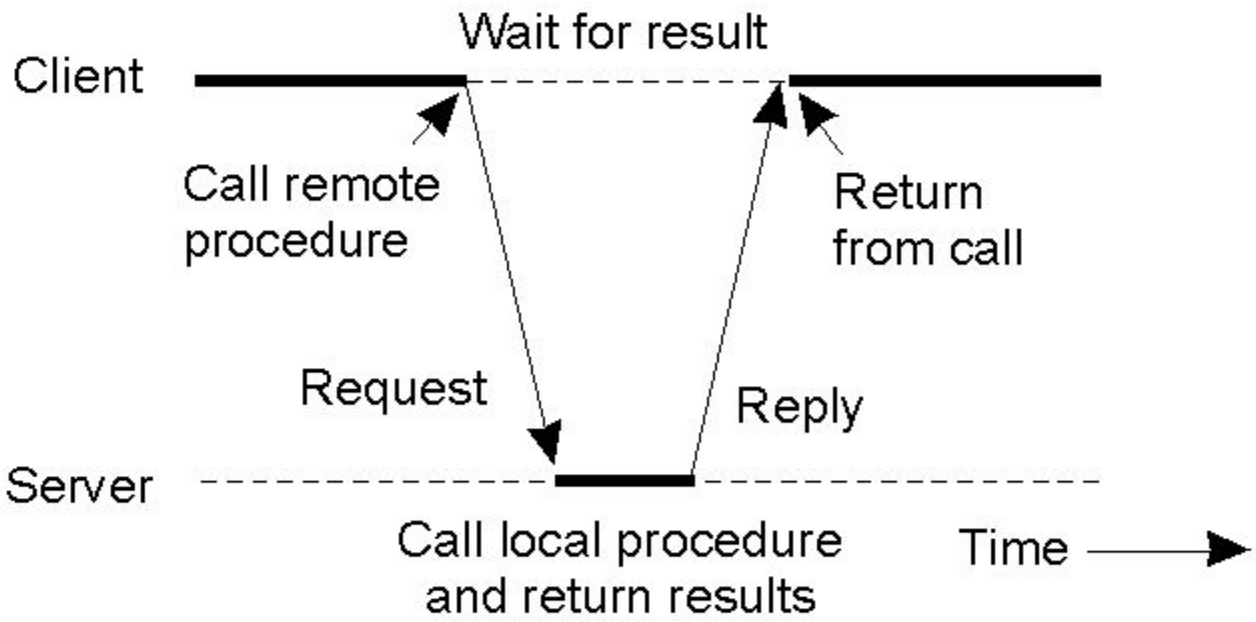
Observations

- Parameters (in C):
  - call-by-reference OR call-by-value

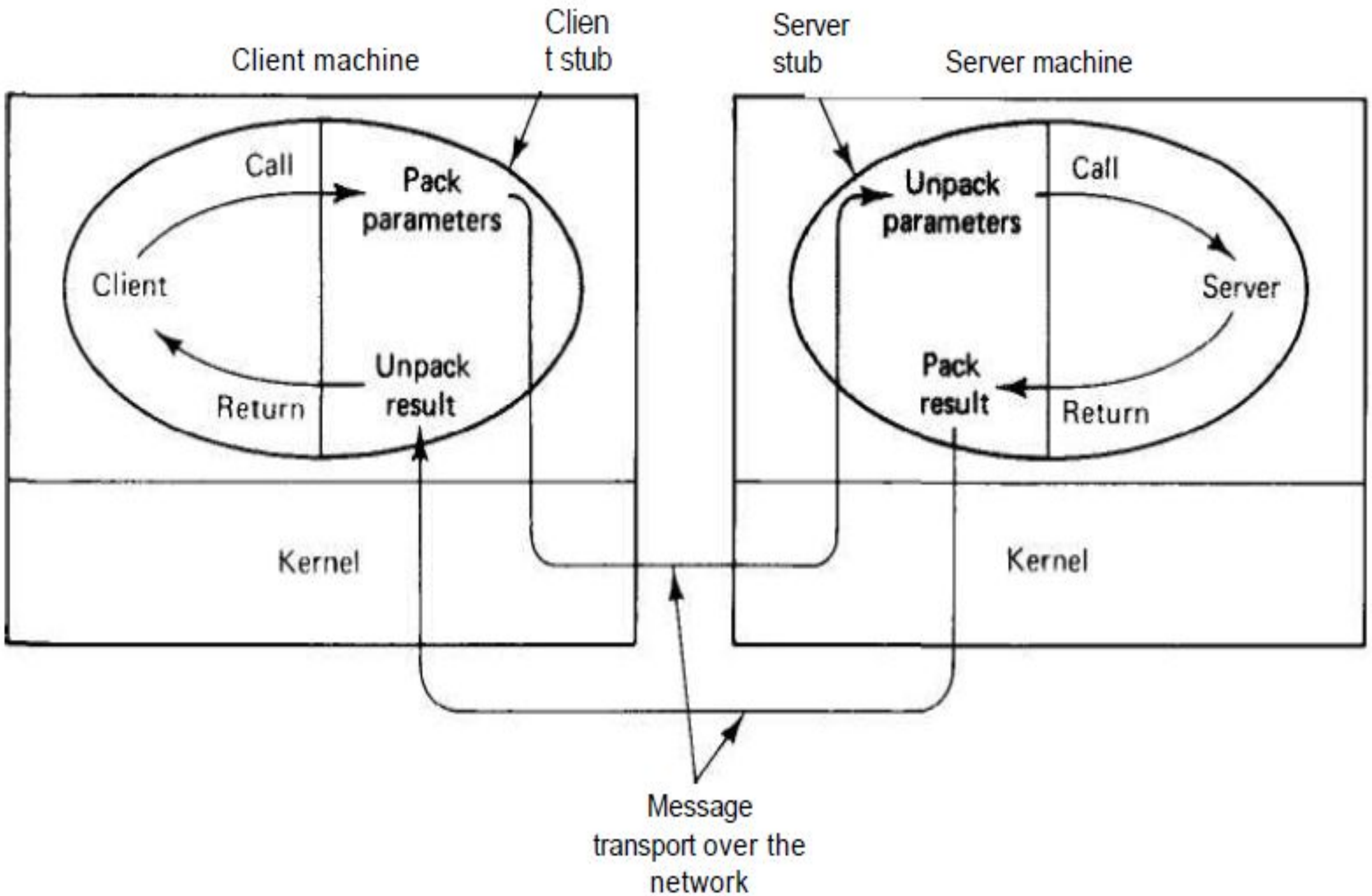
- Value parameter (e.g., fd, nbytes)
- Reference parameter (array buf)
- Many options are language dependent

RPC

- Principle of RPC between a client and server program.

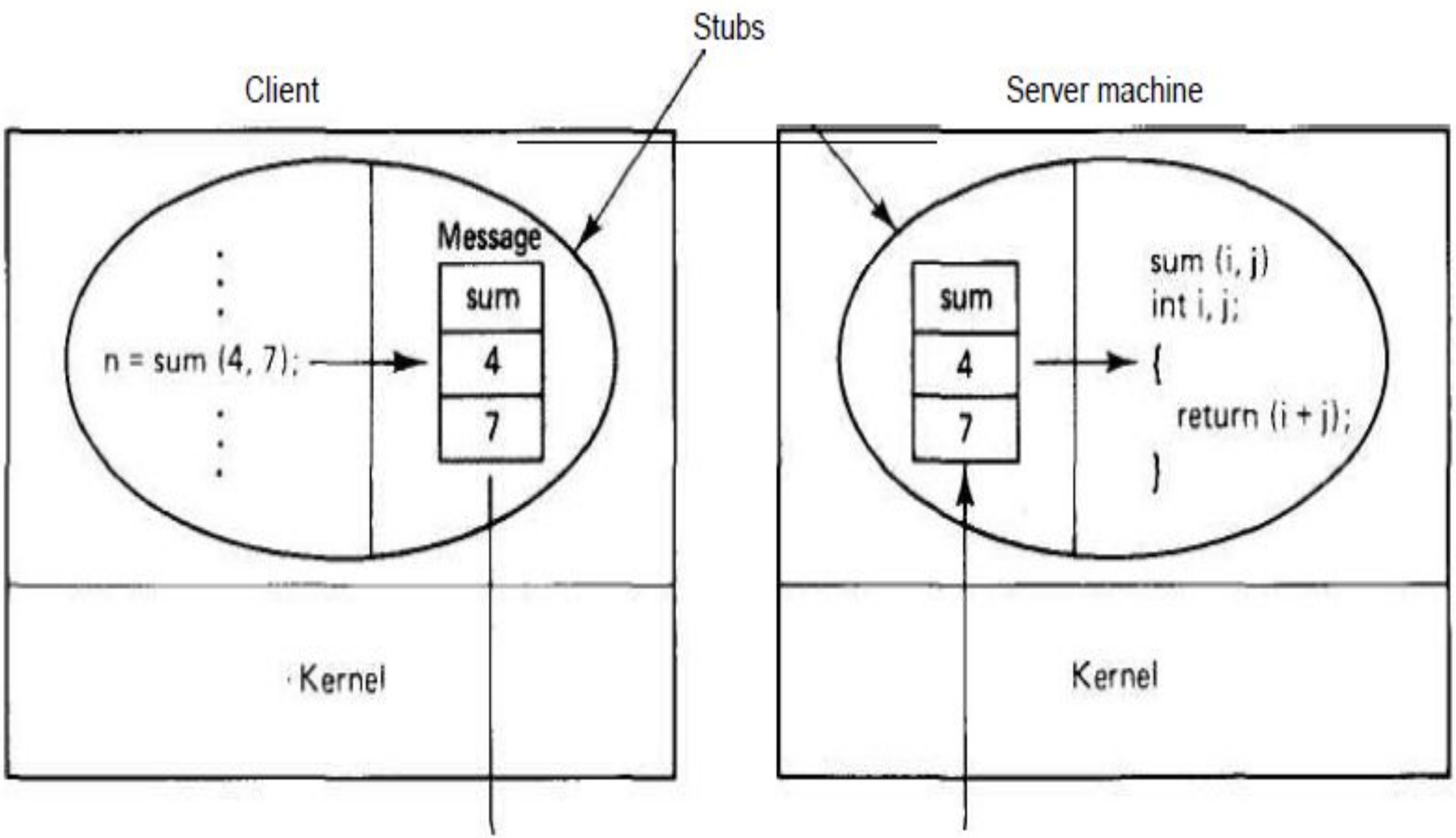


Remote Procedure Call : Model



*

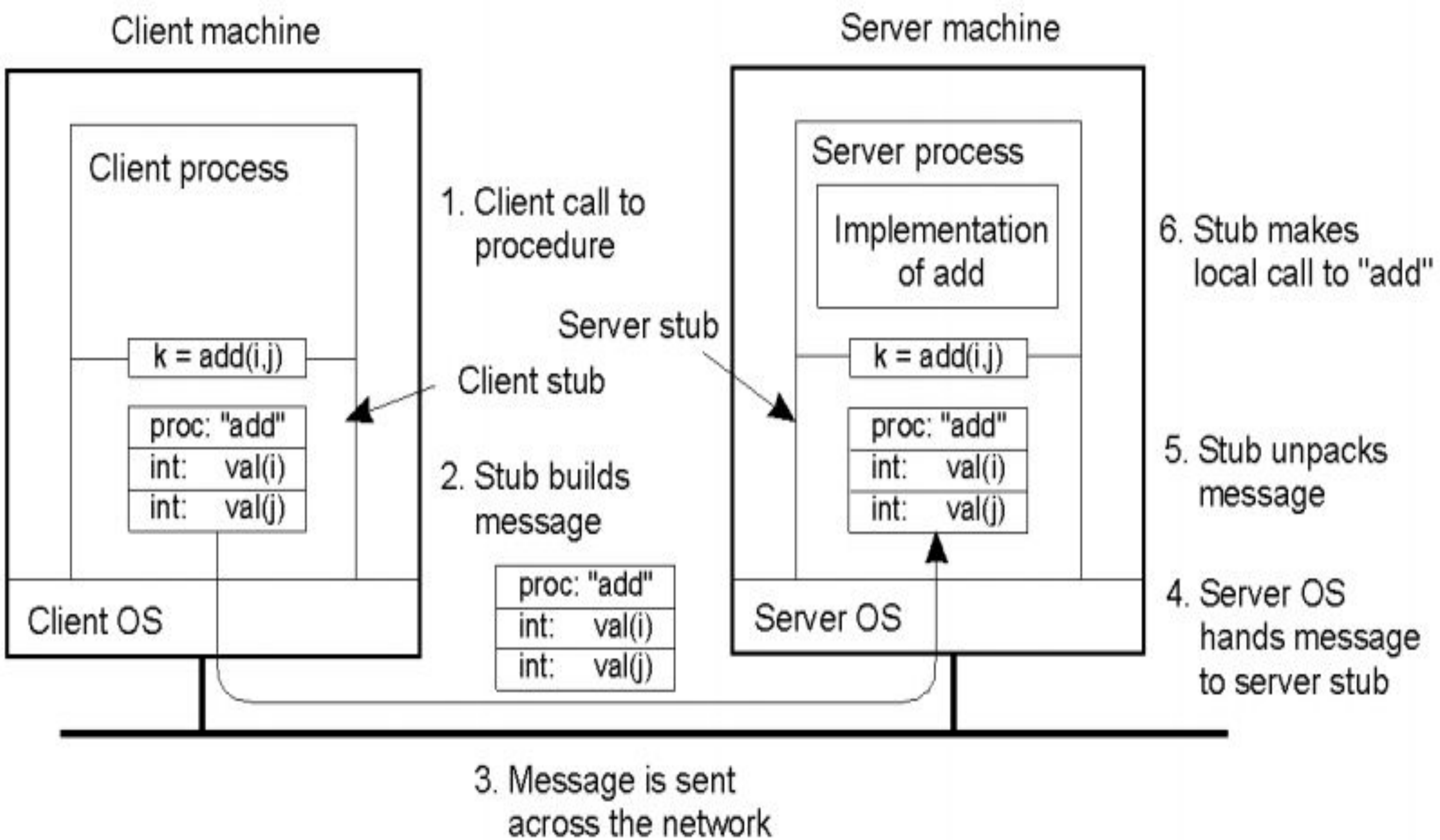
Parameter Passing :RPC



*

Marshalling Value parameter

- Steps involved in doing remote computation through RPC



Remote Procedure Call

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and traps to the kernel.
3. The kernel sends the message to the remote kernel.
4. The remote kernel 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 traps to the kernel.
8. The remote kernel sends the message to the client's kernel.
9. The client's kernel gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

*

Outline of presentation

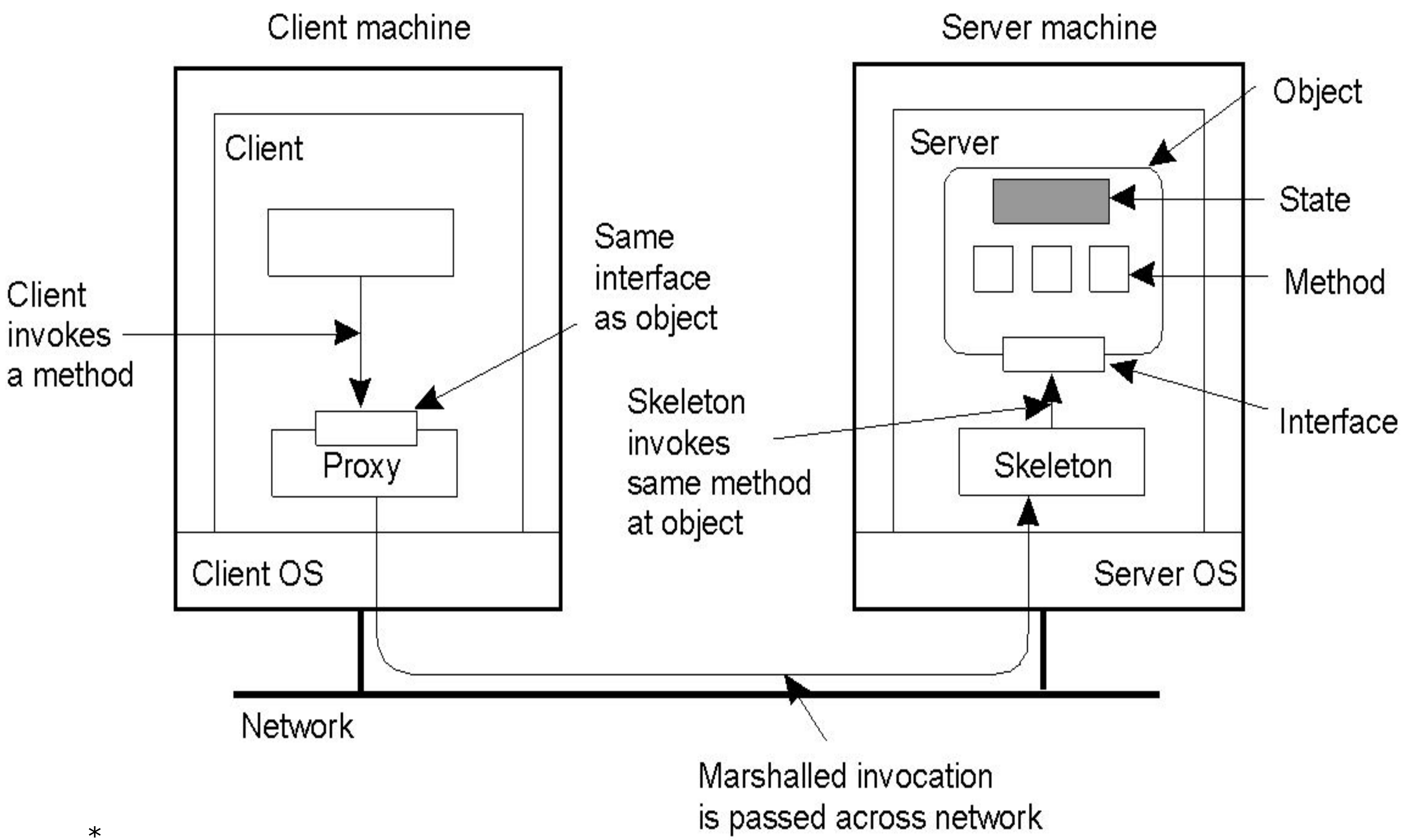
1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

Message Passing : RPC

Message Id	Type	Client Id	RPC Id.			Arguments
			Procedure No.	Version	Program no.	

- RPC Protocol Violation
- Service not Accepted
- Version not Accepted
- Could not Decode Arguments
- Exception while execution (divide by zero)
- Reply (error/result)

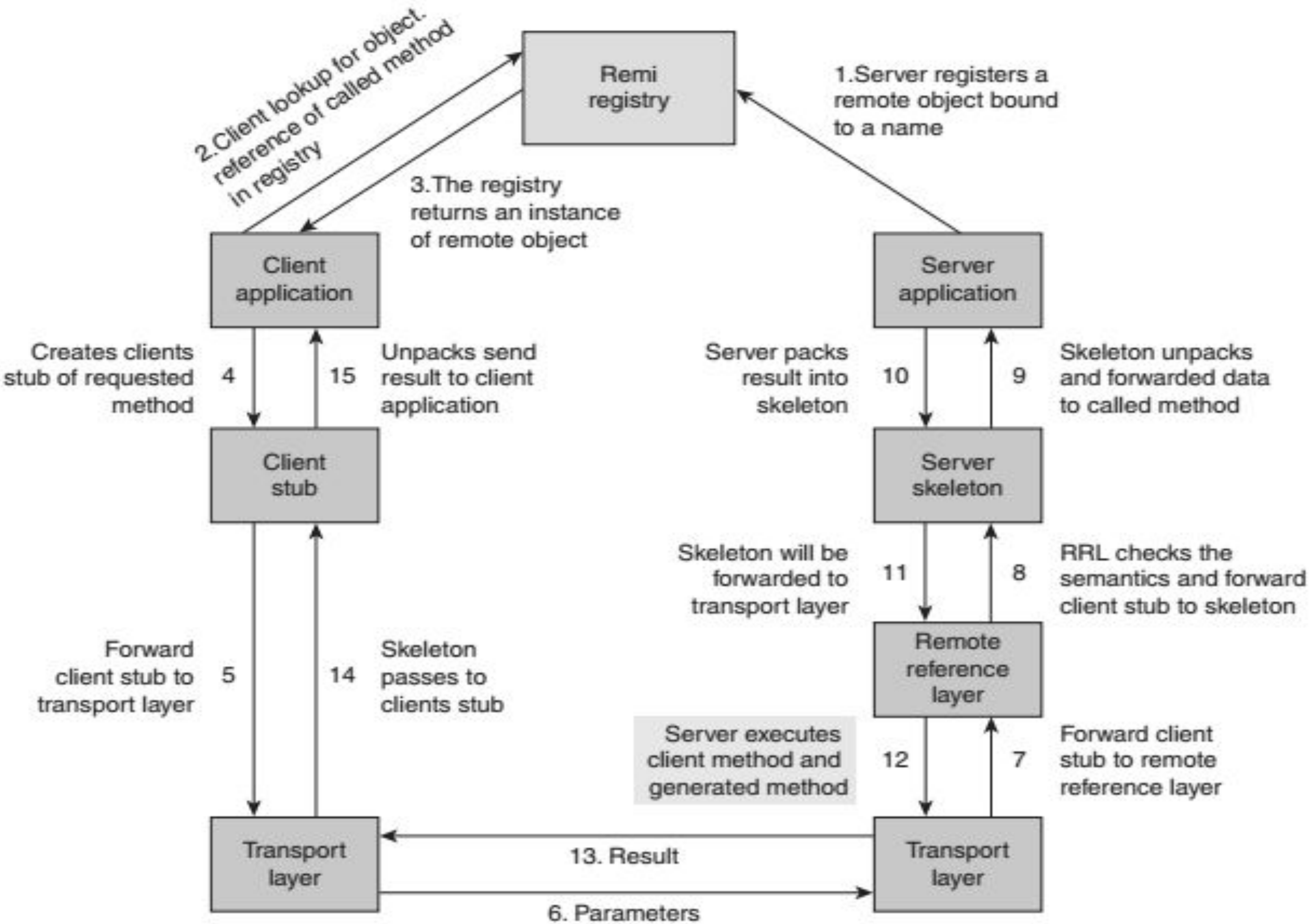
RMI : The Object Model



*

Steps : Building the Application

- **Server Creates Remote Object.**
- **Server Registers Remote Object.**
- **Client requests object from Registry.**
- **Registry returns remote reference**
- **(and stub gets created).**
- **Client invokes stub method.**
- **Stub talks to skeleton.**
- **Skeleton invokes remote object method.**



Remi registry
2.Client lookup for object. reference of called method in registry
3.The registry returns an instance of remote object
1.Server registers a remote object bound to a name
Client application
Server application
Creates clients stub of requested method
4
15
Unpacks send result to client application
Server packs result into skeleton
10
9
Skeleton unpacks and forwarded data to called method
Client stub
Server skeleton
Skeleton will be forwarded to transport layer
11
8
RRL checks the semantics and forward client stub to skeleton
Forward client stub to transport layer
5
14
Skeleton passes to clients stub
Remote reference layer
Server executes client method and generated method
12
7
Forward client stub to remote reference layer
Transport layer
13. Result
Transport layer
6. Parameters

Stub Generation : IDL

An **interface description language** (or alternatively, **interface definition language**), or **IDL** for short, is a specification language used to describe a software component's interface.

IDLs describe an interface in a language-independent way, enabling communication between software components that do not share a language

*

JAVA RMI :

- **In RMI, a common *remote interface* is the minimum amount of information that must be shared in advance between “client” and “server” machines.**

- **A remote interface is a *normal Java interface*, which must extend the marker interface java.rmi.Remote.**
  - **Corollaries: because the visible parts of a remote object are defined through a Java interface, constructors, static methods and non-constant fields are *not* remotely accessible (because Java interfaces can’t contain such things).**

- **All methods in a remote interface *must* be declared to throw the java.rmi.RemoteException exception.(to deal with unexpected network failures)**

*

The Calculator interface

```
public interface Calculator
    extends java.rmi.Remote
    {
        public long add(long a, long b) throws
    java.rmi.RemoteException;
            public long sub(long a, long b) throws
        java.rmi.RemoteException;
     }
```

*

The Remote Object

- **A remote object is an instance of a class that implements a remote interface.**
- **Most often this class also extends the library class java.rmi.server.UnicastRemoteObject. This class includes a constructor that *exports* the object to the RMI system when it is created, thus making the object visible to the outside world.**
- **your remote object classes just have to extend it**
- **A fairly common convention is to name the class of the remote object after the name of the remote interface it implements, but append "Impl" to the end.**

The Interface Implementation

```
public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject
    implements Calculator
{
   public CalculatorImpl() throws java.rmi.RemoteException
     { super(); }
                public long add(long a, long b) throws
    java.rmi.RemoteException { return a + b; }
                public long sub(long a, long b) throws
    java.rmi.RemoteException { return a - b; }
 }
```

The Stub and Skeleton Generation.

- >rmic CalculatorImpl
  - **CalculatorImpl_skel.class**
  - **CalculatorImpl_stub.class**

The Server Program.

```
import java.rmi.Naming;
public class CalculatorServer
{
   public CalculatorServer() {
   try
   {
    Calculator c = new CalculatorImpl();
    Naming.rebind("rmi://localhost:1099/CalculatorService", c);
   }
   catch (Exception e) { System.out.println("Trouble: " + e);}
 }
 public static void main(String args[])
  { new CalculatorServer(); }
}
```

*

Anatomy of the Server

- **This program does two things:**
  - **It creates a remote object with local name server.**
  - **It publishes a remote reference to that object with external name "CalculatorService".**
- **The call to Naming.rebind() places a reference to server in an *RMI registry* running on the local host (i.e., the host where the calculatorServer program is running).**
- **Client programs can obtain a reference to the remote object by looking it up in this registry.**

The Client Program

```
public class CalculatorClient
 {
  public static void main(String[] args)
 {
   try
   { Calculator c =
     (Calculator)Naming.lookup("rmi://localhost/C
     alculatorService");
     System.out.println( c.sub(4, 3) );
     System.out.println( c.add(4, 5) );
    } catch (/*some exception*/) {/*___*/}
  }
 }
```

*

Server Management

- Server States
  - Stateful
  - Stateless
- Server Creation Semantics
  - Instance - Per - Call
  - Instance - Per – Session
  - Persistent Servers

Parameter Passing

- Call – by – value
- Call by Reference

Call Semantics

- May- be
- Last-one call
- Last – of –many
- At – least once
- Exactly once

Communication Protocols

a) The Request Protocol (may-be)
b) The Request/Reply Protocol (at-least once)
c) The Request/Reply/Acknowledge Protocol
d) RPC involving Long-duration calls /gaps
  - Periodic probing of the server by the client

  (server crash/link failure)

  - Periodic generation of an acknowledgement by the server

e) RPC involving arguments and or results that are large to fit in a single datagram packet.

Multiple RPC (multiple ack.) / Multidatagram (single ack)

Outline of presentation

1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

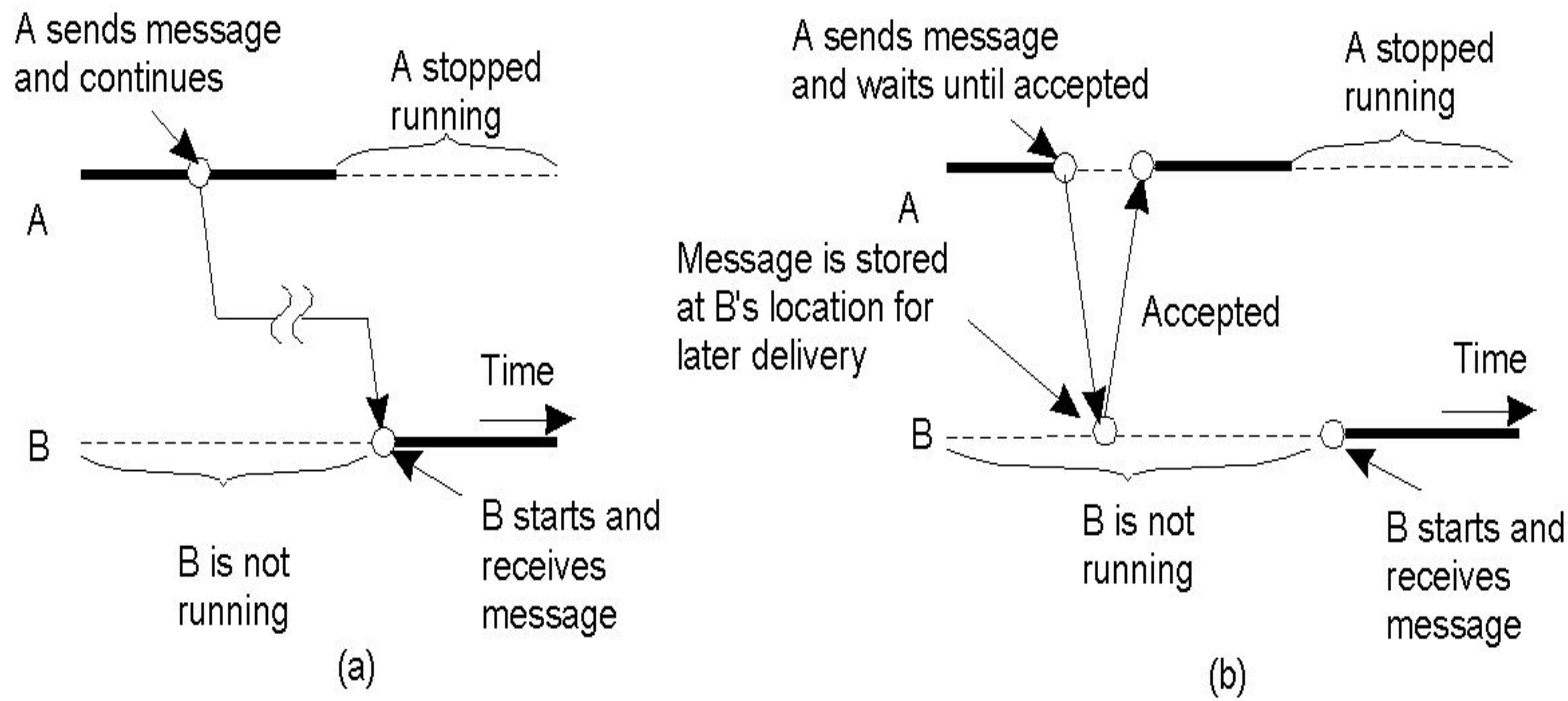
Synchronisation

a) Blocking : Synchronous
    a) Polling
    b) Interrupt
b) Non blocking: Asynchronous

Buffering

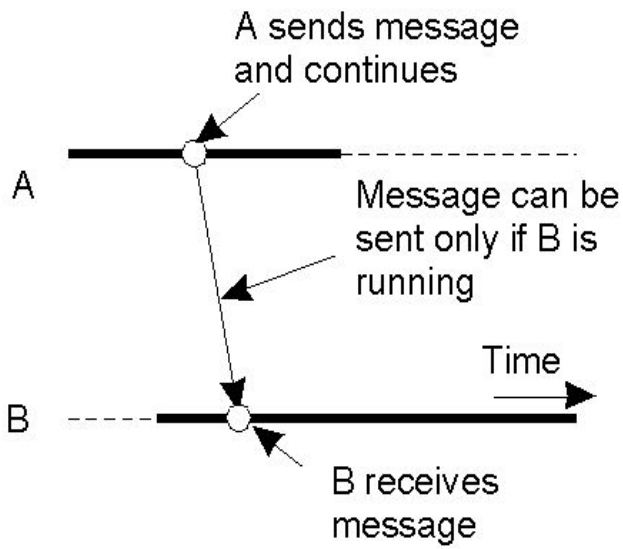
a) Buffered : Persistent
    a) Single message , finite bound, multiple-message buffers (unsuccesful /flow controlled)
b) Non Buffered : Non persistent / transient

Persistence and Synchronicity in Communication (3)



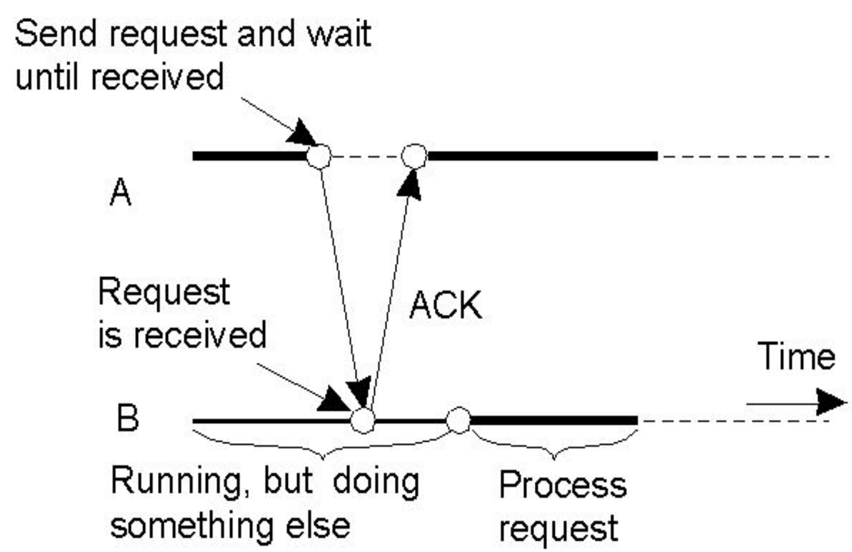
a) Persistent asynchronous communication
b) Persistent synchronous communication

Persistence and Synchronicity in Communication (4)



(c)

TCP

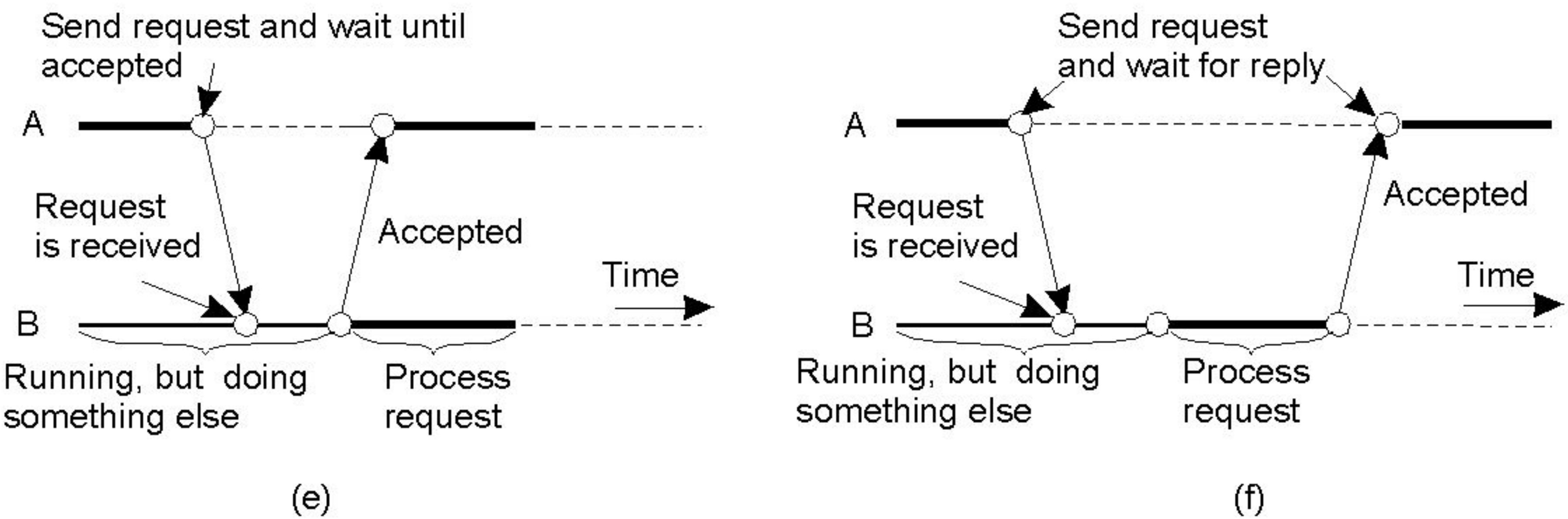


(d)

c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication (5)

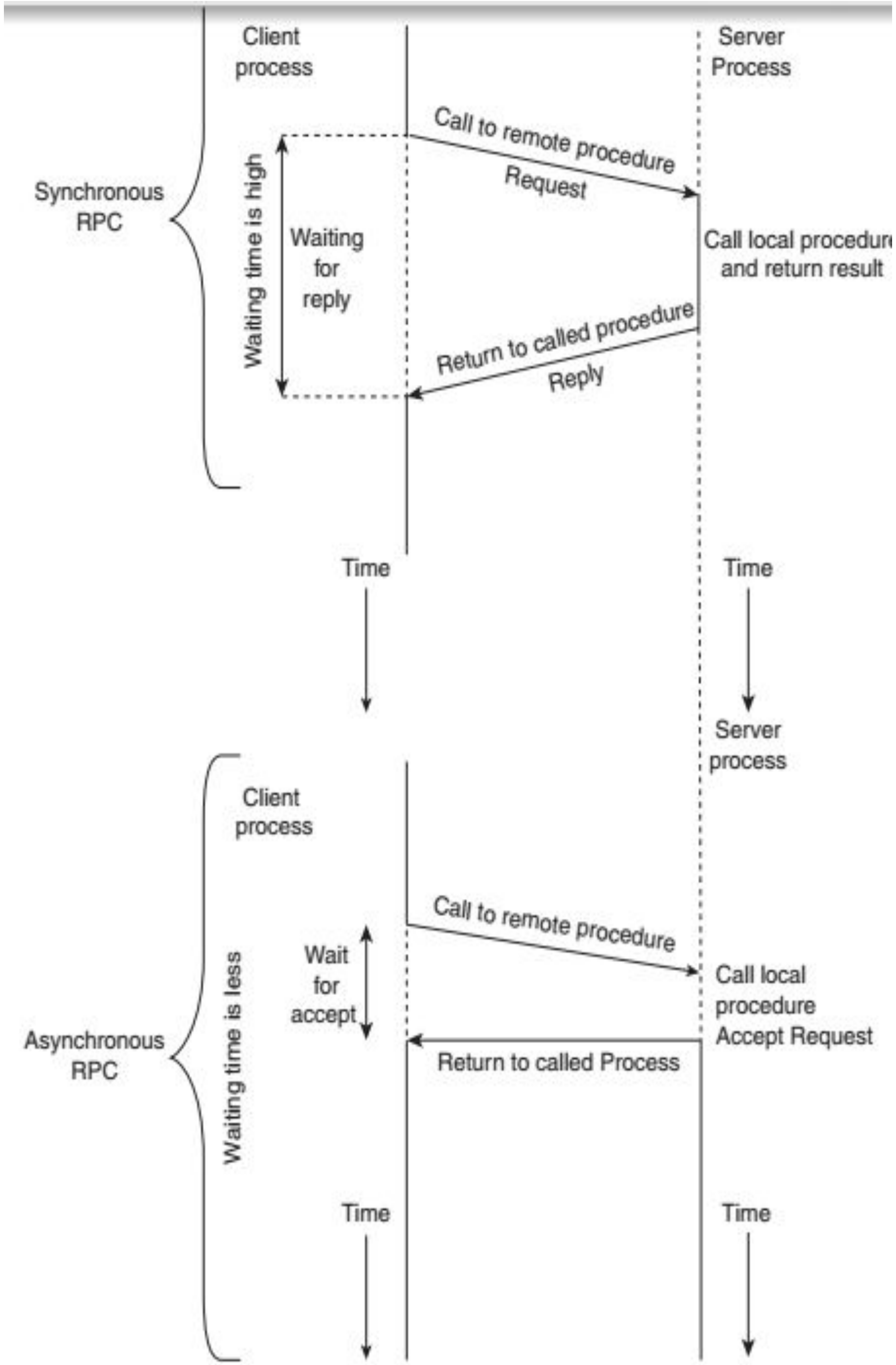


RPC/RMI

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

RPC Message Passing Model



Synchronous RPC.

Asynchronous RPC.

Outline of presentation

1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

Group Communication

a) One to many Communication

- Multi cast
  - Closed group/Open group
  - Group server : managing group activity (poor reliability, scalability)
  - Group addressing : A two level naming scheme
  - Message Delivery to Receiver Process : via mapping of high level name to IP address
  - Buffered and un-buffered : Asynchronous
  - Send-to-All and Bulletin-Board Semantics :

    - A copy of the message.

    -The message is sent to a channel (bulletin board) (receive access right)

    -Relevance of a message to a particular receiver may depend on the receiver’s state.

    -Message received after a certain time after transmission may no longer be relevant (their value may depend on the sender’s state)
  - Flexible Reliability in multicast communication

    -0-Reliable/1-Reliable/m-out-of-n-reliable/All reliable
  - Atomic Multicast

Group Communication

- One to One Communication, point to point or unicast
- Single sender process sends a message to a single receiver process
- Several highly parallel distributed applications require that a message passing system should support group communication facility
  - 3 Types of Group Communication
    - One to many (Single sender multiple receiver)
    - Many to one (multiple senders Single receiver)
    - Many to many (multiple sender many receivers)

Group Communication

- One to Many Communication
- Multicast communication
- A special case of multicast communication is broadcast communication in which the message is sent to all processors connected to network
- Applicable in server management where server manage group of server processes
- Group Management
- In One to many communication, receiver processes of message form a group:
- Closed Group- only members of group can send and receive messages
- Open Group- any process in the system can send a message to the group

Group Communication

- Closed group or Open group is application dependent
- Message passing system provides flexibility to create and delete groups dynamically, and allow a process to join or leave a group at any time
- They have mechanism to manage the groups and their membership information
- Centralized group server process can manage group
- All requests to create, delete group, add member to group, remove member from group sent to this process

Group Communication

Many to One Communication

- Multiple senders send messages to a single receiver
- The receiver may be selective or non selective
- A selective receiver specifies a unique sender, message exchange takes place if that sender sends a message
- A nonselective receiver specifies a set of senders, and if any one sender in the set sends a message to this receiver a message exchange takes place

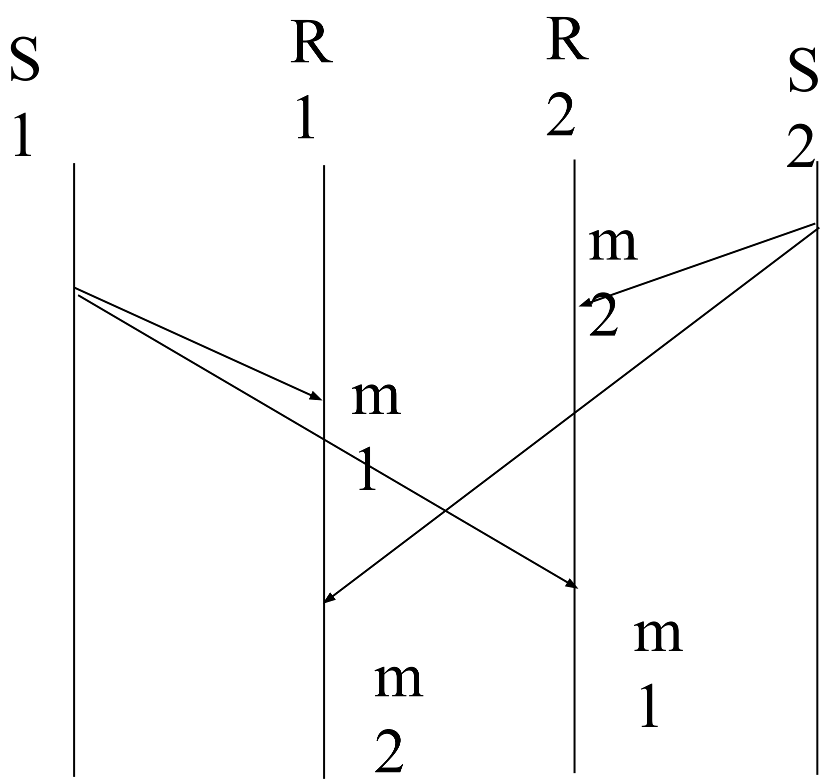
Group Communication

- Many to Many Communication
- Multiple senders send messages to a Multiple receivers
- Important issue in this is “Ordered Message Delivery”
- Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application
- e.g database update request
- Ordered message delivery requires message sequencing

Group Communication

- Ordered Massage delivery through message sequencing
  - Absolute Ordering
  - Consistent Ordering
  - Causal Ordering

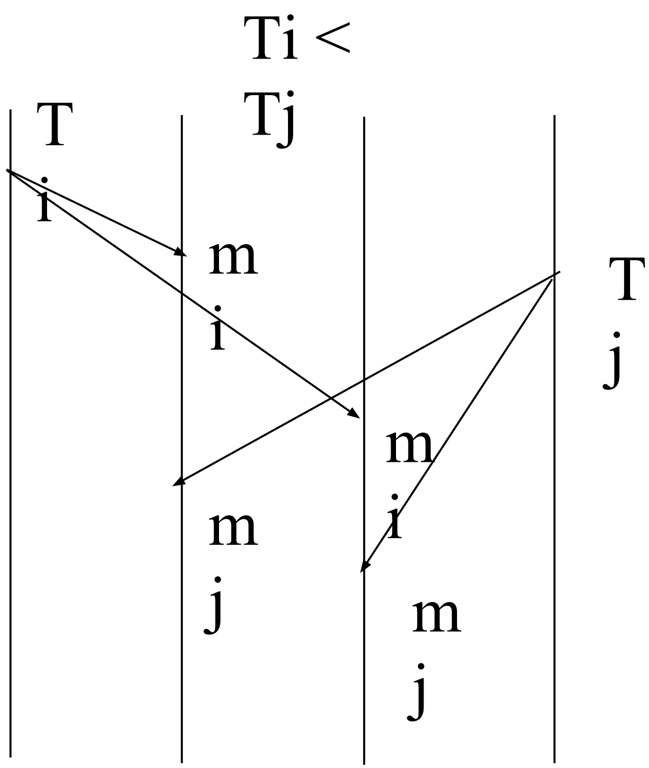
Message Ordering



- R1 and R2 receive m1 and m2 in a different order!
- Some message ordering required
  - Absolute ordering
  - Consistent/total ordering
  - Causal ordering

Fig- No ordering constraint for message delivery

Absolute Ordering



Rule:Global Timestamp

Mi must be delivered before mj if Ti < Tj

Implementation:

A clock synchronized among machines

A sliding time window used to commit message delivery whose timestamp is in this window.

Example:

Distributed simulation

Drawback

Too strict constraint

No absolute synchronized clock

No guarantee to catch all tardy messages

Absolute Ordering

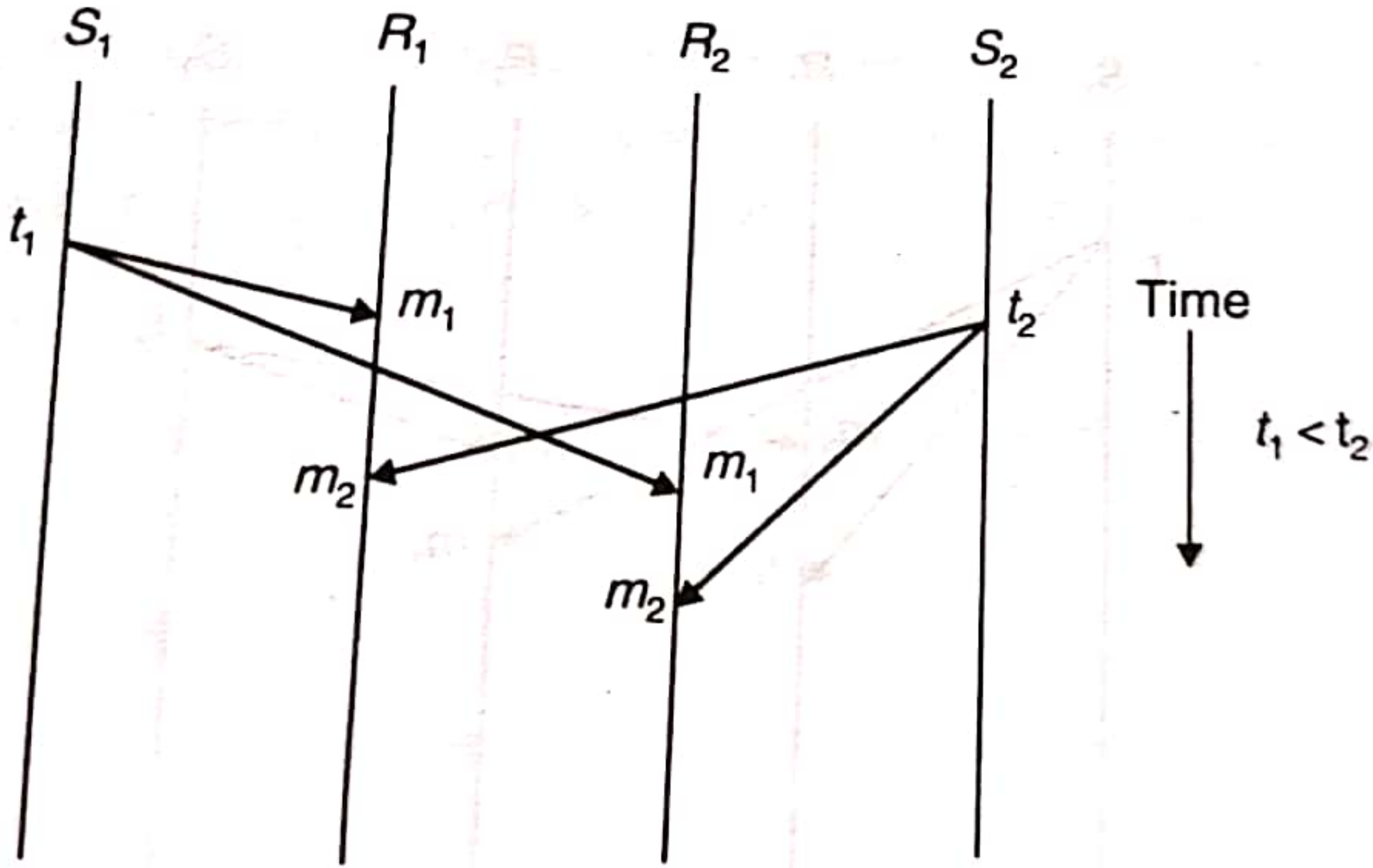


Fig. 3.15 Absolute ordering of messages.

Consistent Ordering

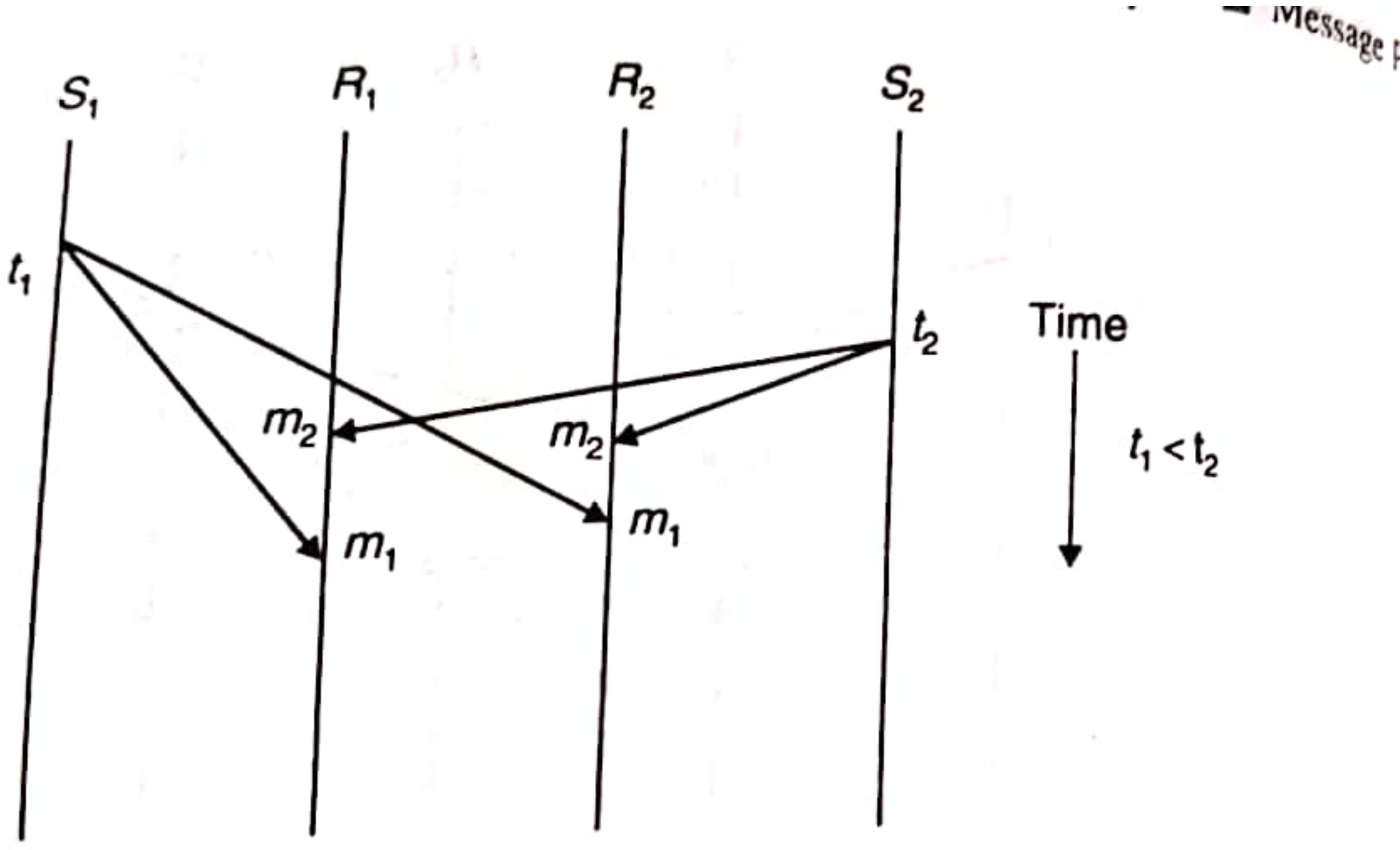


Fig. 3.16 Consistent ordering of messages.

Casual Ordering

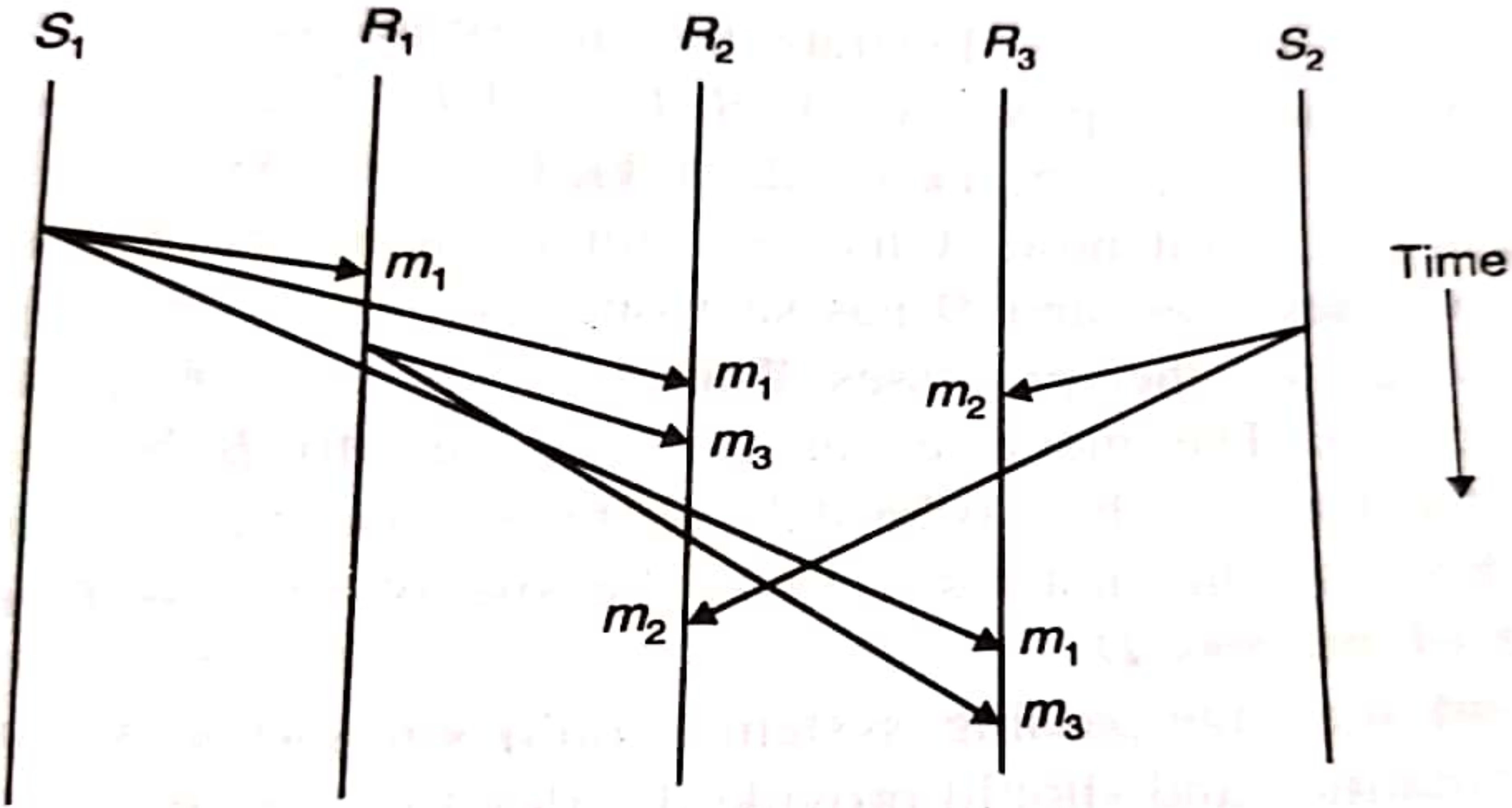
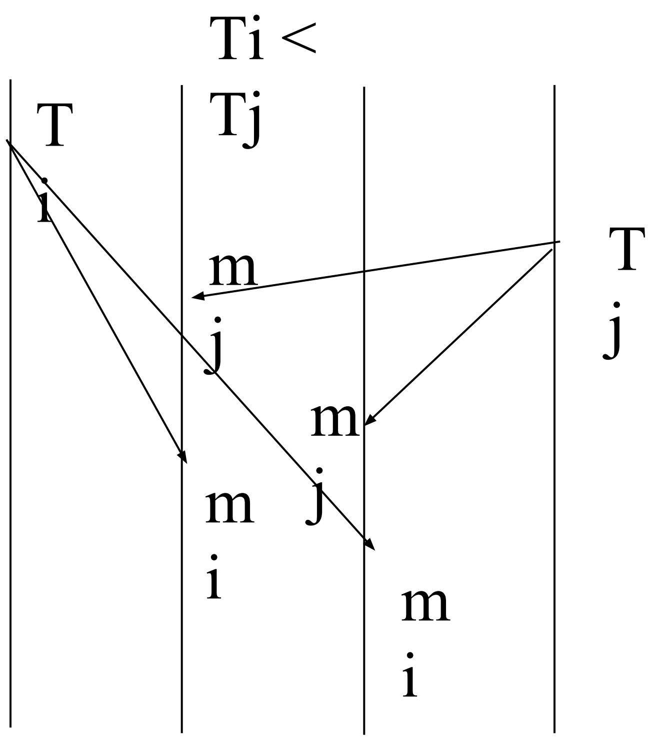


Fig. 3.17 Causal ordering of messages.

Consistent/Total Ordering



Rule:

Messages received in the same order (regardless of their timestamp).

Implementation:

A message sent to a sequencer, assigned a sequence number, and finally multicast to receivers

A message retrieved in incremental order at a receiver

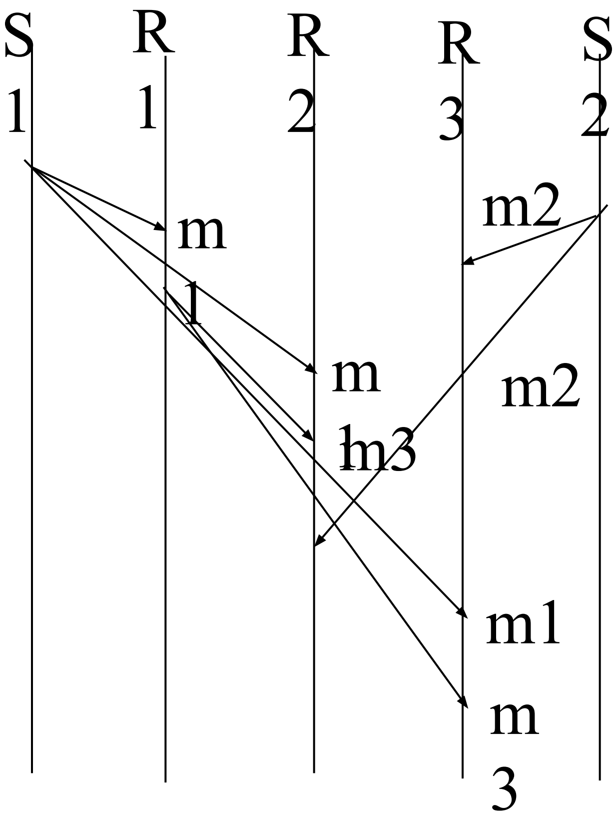
Example:

Replicated database updates

Drawback:

A centralized algorithm

Causal Ordering



From R2's view point m1 →m3, and event m3 is causally dependent on m1 So order of m1 and m3 should be preserved in at all receiver

Where as order of m2 does not matter

Outline of presentation

1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

MOM Vs SOM

- Message-oriented communication
  - Persistence and synchronicity
  - Message-oriented transient communication
    - Berkeley socket
    - MPI
  - Message-oriented persistent communication
- Stream-oriented communication
  - Data stream
  - Quality of services
  - Stream synchronization

- Message-oriented communication: request-response
  - When communication occurs and speed do not affect correctness
- Timing is crucial in certain forms of communication
  - Examples: audio and video (“continuous media”)
  - 30 frames/s video => receive and display a frame every 33ms
- Stream oriented comm is required!

Message oriented communication

1. Message oriented transient communication

- Berkeley Sockets
- Message Passing Interface

2. Message oriented persistent communication

- Message queuing model
- Message Brokers

Berkeley Sockets

- Socket is communication endpoint to which an application can write data that are to be sent out over the underlying network and from which incoming data can be read
- Socket interface was introduced in 1970

Berkeley Sockets

- socket- caller creates a new communication end point for a specific transport protocol
- Bind- associates a local address with the newly created socket
- Listen- used in connection oriented communication
- Accept- block the caller until connection request arrives
- Connect- Require caller specific transport level address to which a connection request is to be sent

Berkeley Sockets

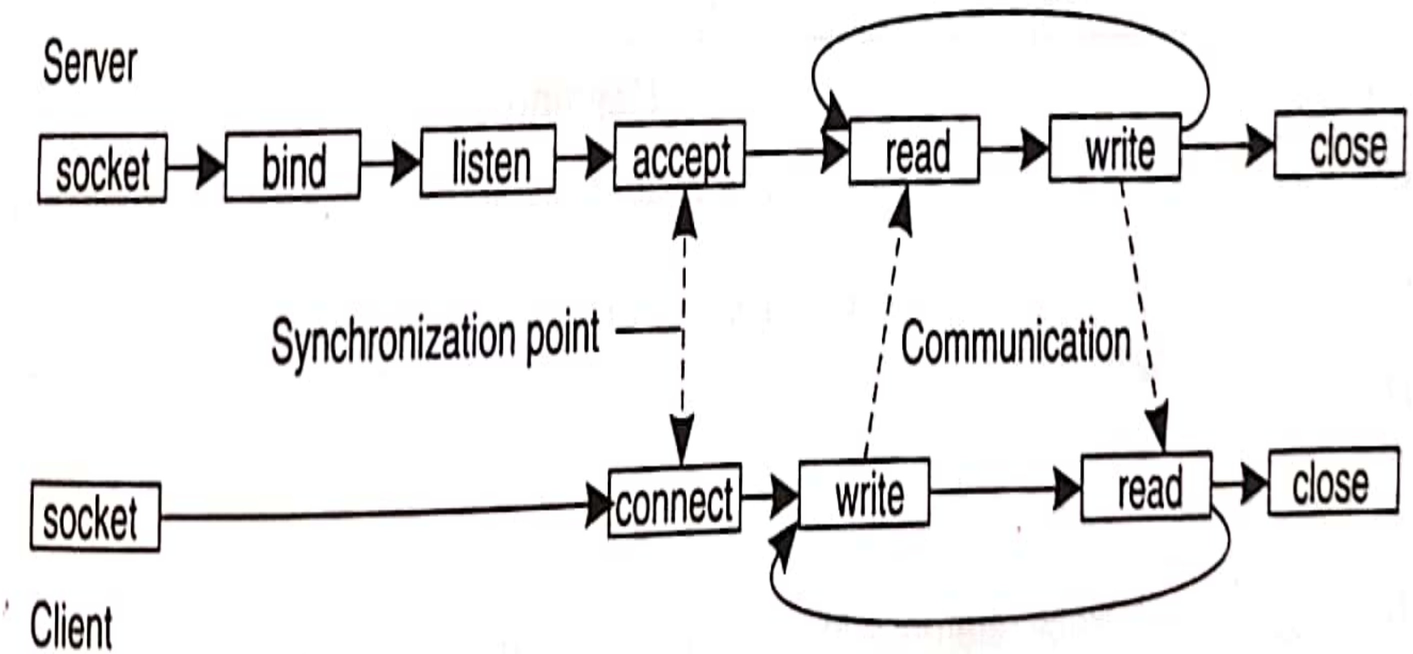


Figure 4-15. Connection-oriented communication pattern using sockets.

COMMUNICATION

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Message passing interface (MPI)

- Standard Message passing interface
- MPI is designed for parallel applications
- It makes direct use of underlying network
- MPI assume that communication taken place within known group of processes
- Each group assigned identifier
- Each process have (groupID,processID)

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

Figure 4-16. Some of the most intuitive message-passing primitives of MPI.

Message Oriented Persistent Communication

- Message queuing systems provide extensive support for persistent asynchronous communication

- They offer intermediate term storage capacity for messages, without requiring either sender or receiver to be active during message transmission

- Applications communicate by inserting messages in specific queues

Message-Queuing Model

- These messages are forwarded over a series of communication servers and eventually delivered to the destination. Even if it was down when the messages was sent
- Each application has its own private queue to which other applications can send messages
- A queue can be read only by its associated application, but it is also possible for multiple applications to share a single queue
- An important aspect is a sender is generally given only the guarantees that its message will eventually be inserted in the recipients' queue, no guarantees about when

Message-Queuing Model

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

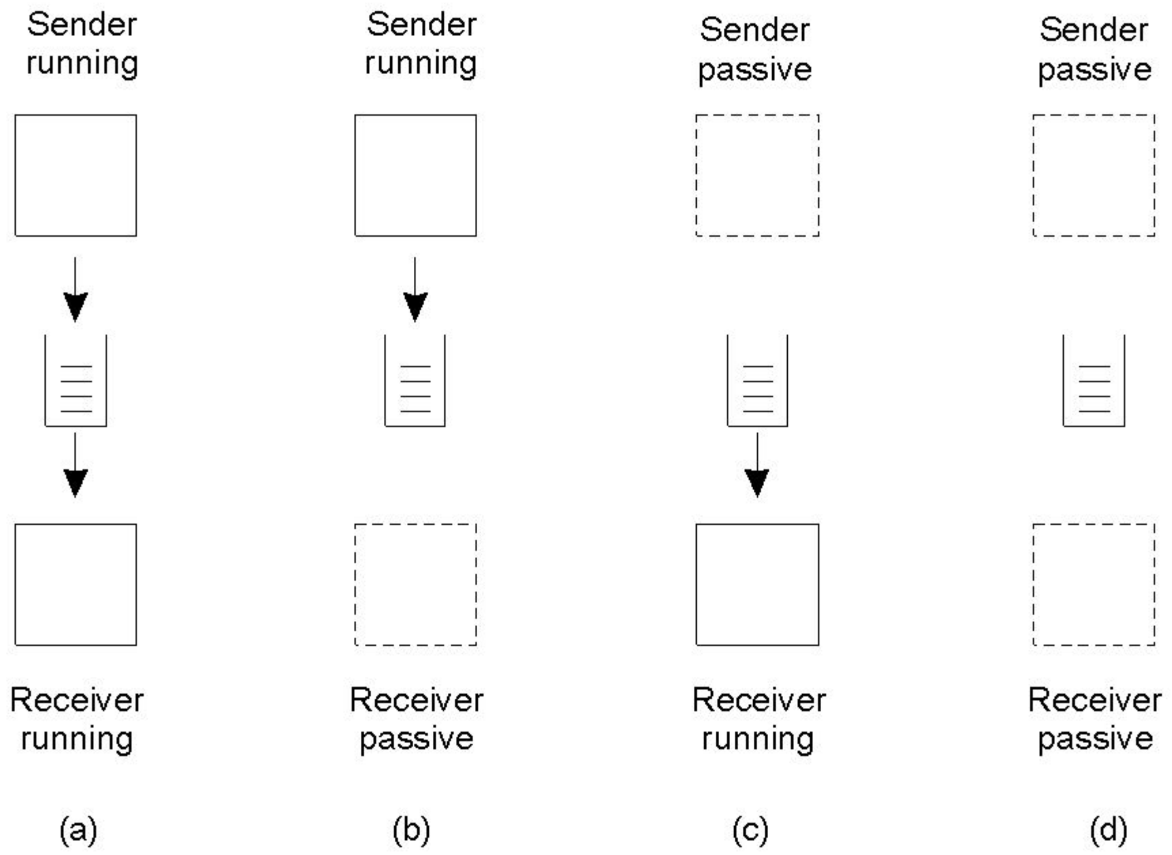
Message-Queuing Model

- Put- primitive is called by sender to pass a message to the underlying system

- Get- it is blocking call by which an authorized process can remove the longest pending message in the specified queue.
- poll- searching for a specific message in the queue

- Notify- allow a process to install a handler as a callback function which is automatically invoked whenever a message is put in queue

Message-Queuing Model (Message Oriented Model) (MOM)



Four combinations for loosely-coupled communications using queues.

Message-Queuing Model

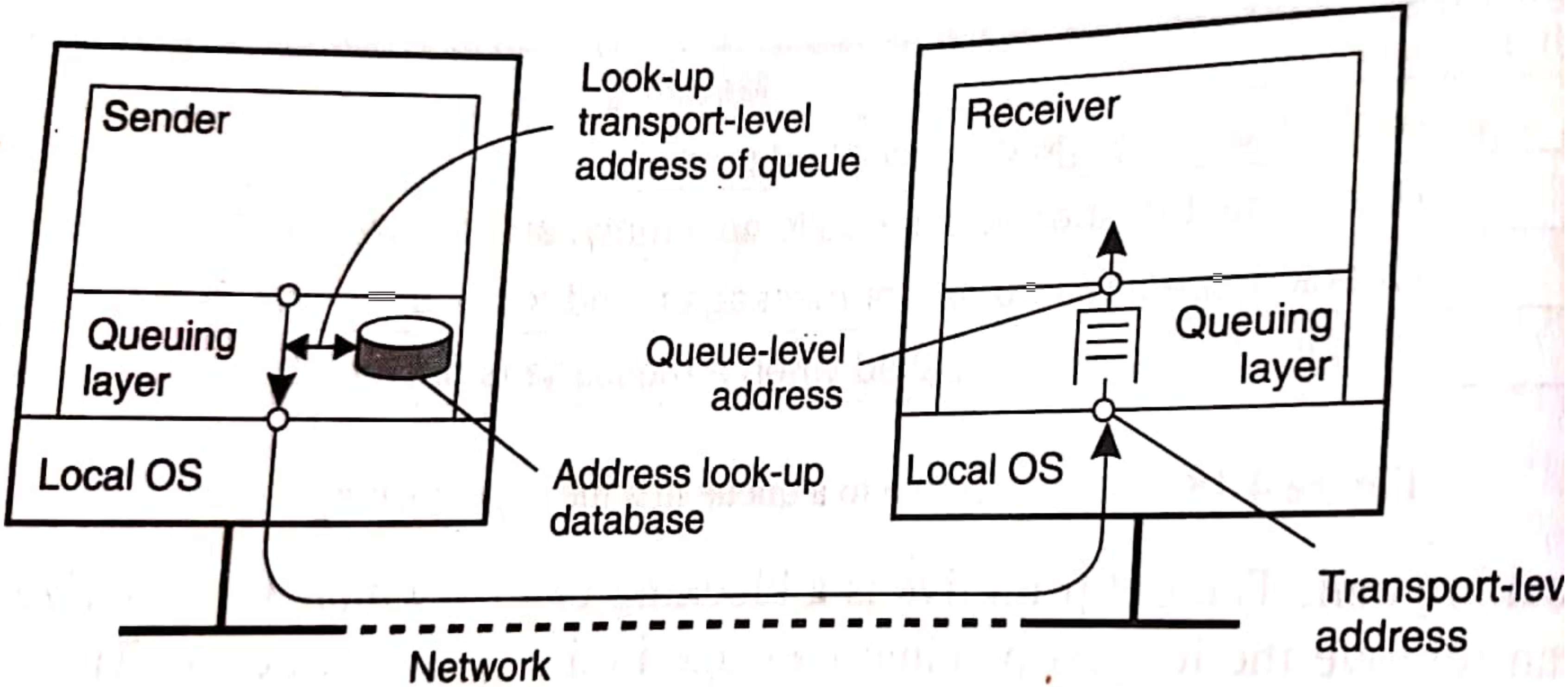


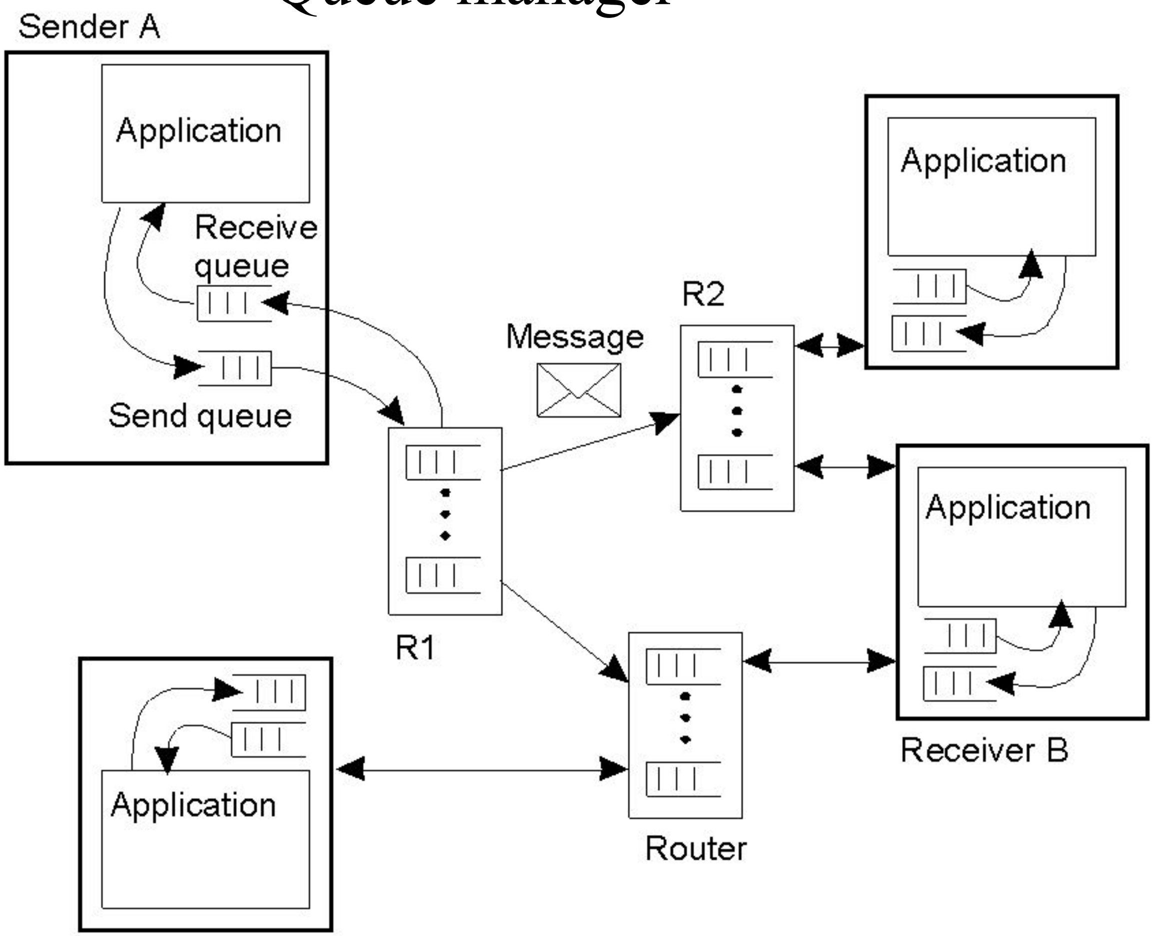
Figure 4-19. The relationship between queue-level addressing and network-level addressing.

Message-Queuing Model

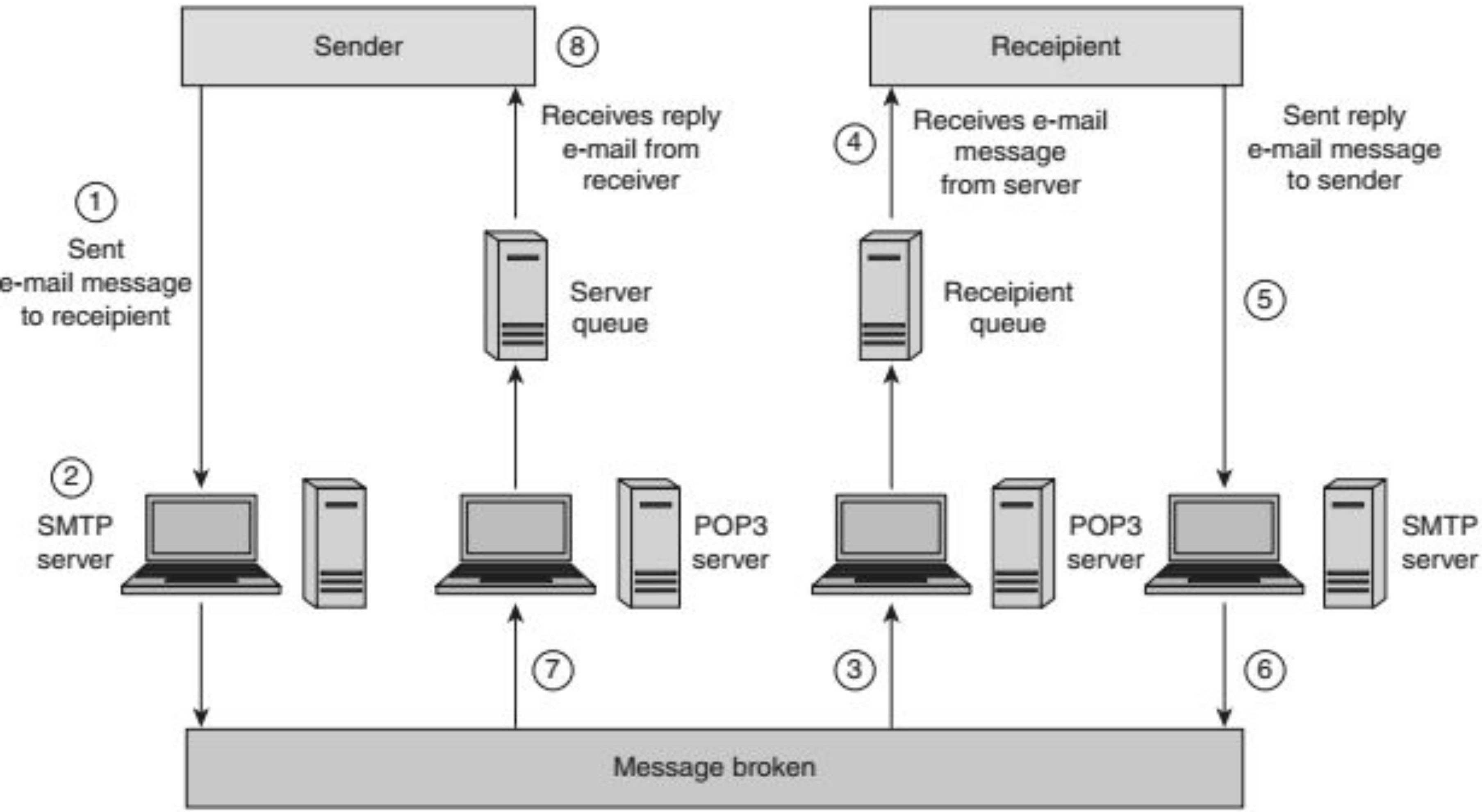
- Source queue- messages can put only in queue that are local to the sender , queue on same machine
- Destination queue- specification of destination queue to which it should be transferred
- Mapping of queues to n/w - database of queue names to network locations
- queue managers- Queue manager interacts directly with the applications that is sending or receiving a message
- relay- special queue manager, secondary processing of message, multicasting, build scalable message queuing system

General Architecture of a Message-Queuing System (2)

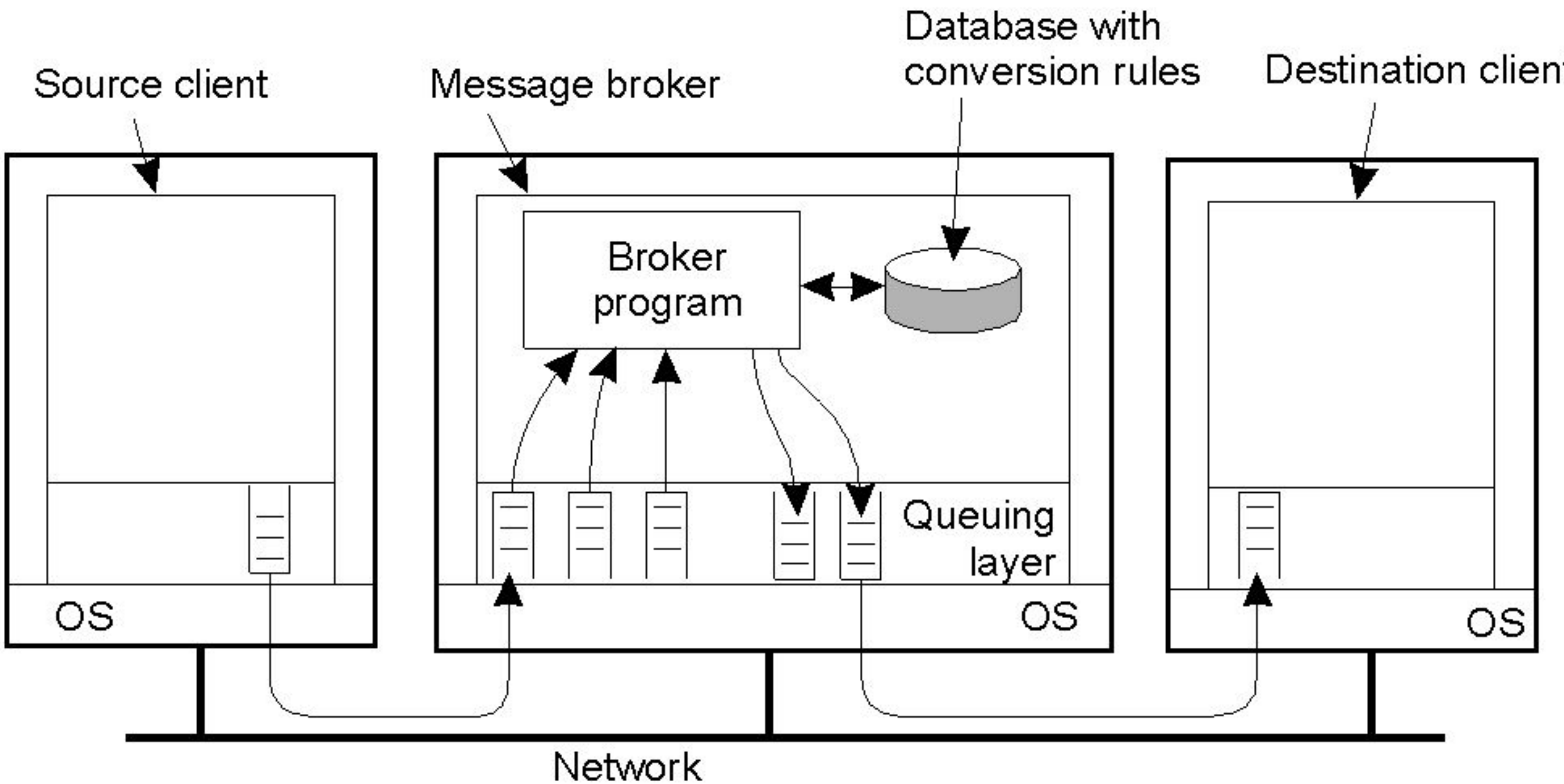
Queue manager



The general organization of a message-queuing system with routers.


Sender
8
Receipient
Receives reply e-mail from receiver
4
Receives e-mail message from server
Sent reply e-mail message to sender
1
Sent e-mail message to receipient
Server queue
Receipient queue
5
2
SMTP server
POP3 server
POP3 server
SMTP server
7
3
6
Message broken

Message Brokers



The general organization of a message broker in a message-queuing system.

Outline of presentation

1. Message Passing : IPC
2. OSI Model and Middleware
3. RPC
4. RMI
5. Message Communication Models
6. Group Communication
7. MOM
8. Stream Oriented Communication

Stream-Oriented Communication

Distributed system support for exchange of time-dependent information such as audio and video stream

Continuous Media- playing an audio stream, video file

Discrete Media- Representation of text and still images, object code,

Data Stream (1)

- A data stream is a sequence of data units
- Discrete or continuous:
  - Discrete stream
  - Continuous stream
- For continuous stream, three transmission modes:
  - Asynchronous transmission mode
    - No timing requirements
  - Synchronous transmission mode
    - Maximum end-to-end delay
  - Isochronous transmission mode
    - Both minimum and maximum end-to-end delay

Stream

Streams can be simple or complex

Simple – only single sequence of data

Complex- several related simple stream

In multimedia data, video and audio need to be compressed in order to reduce the required storage and network capacity

Stream

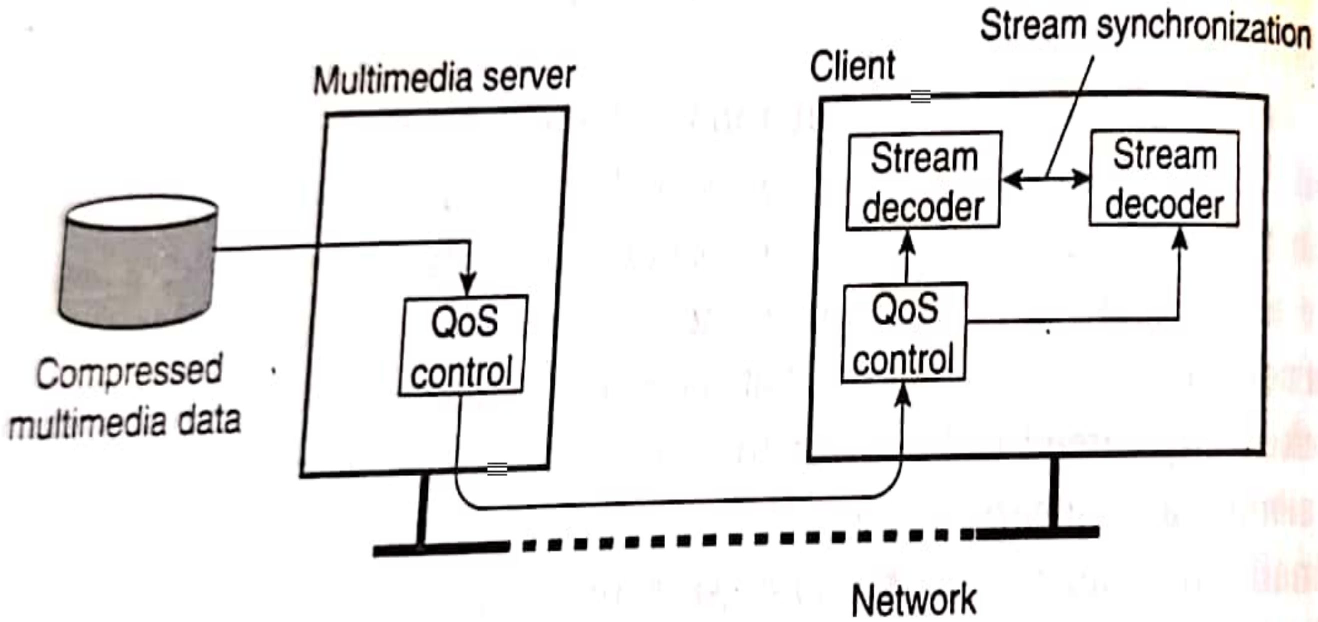


Figure 4-26. A general architecture for streaming stored multimedia data over a network.

Streams and Quality of Service

Timing requirements are expressed as Quality of Service (QoS) requirements

1. The required bit rate at which data should be transported.
2. The maximum delay until a session has been set up (i.e., when an application can start sending data).
3. The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient).
4. The maximum delay variance, or jitter.
5. The maximum round-trip delay.

Streams and Quality of Service

Enforcing QoS

- Differentiated services
- Expedited forwarding
- Assured forwarding

Enforcing QoS

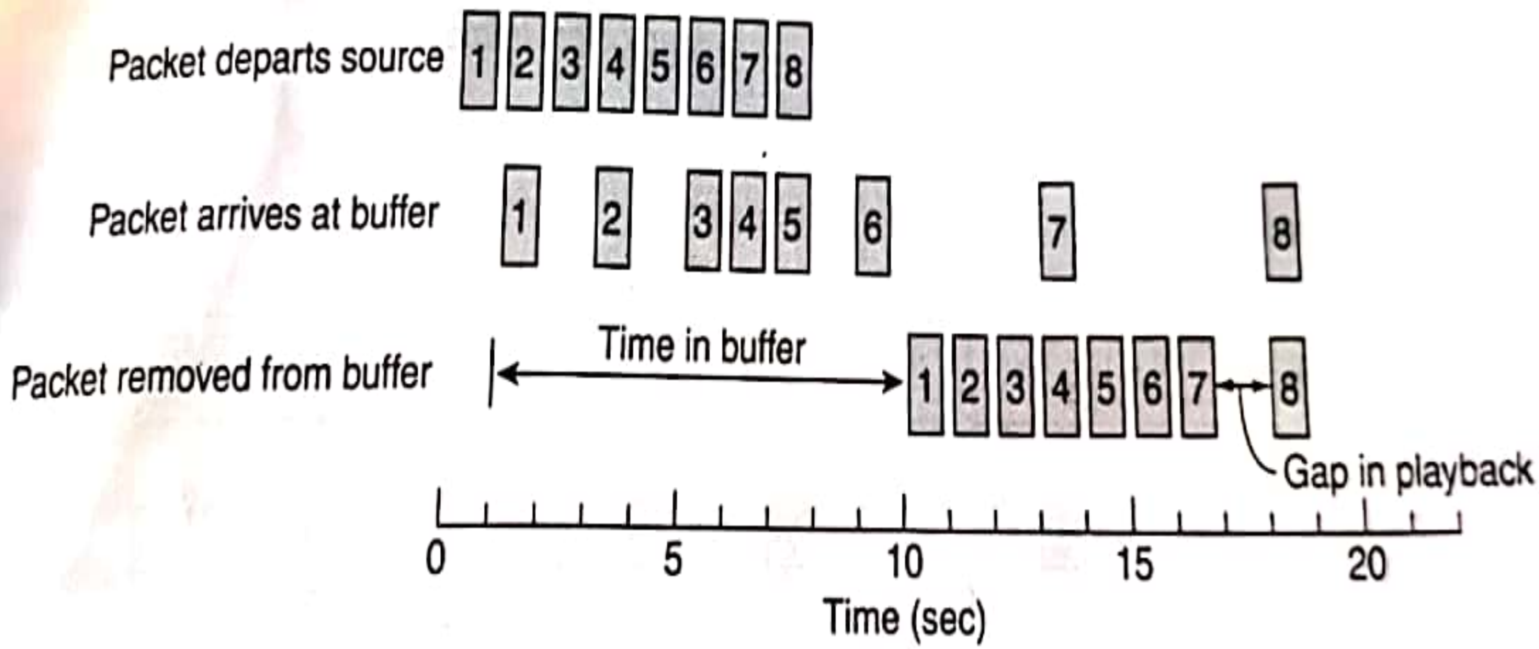


Figure 4-27. Using a buffer to reduce jitter.

Enforcing QoS

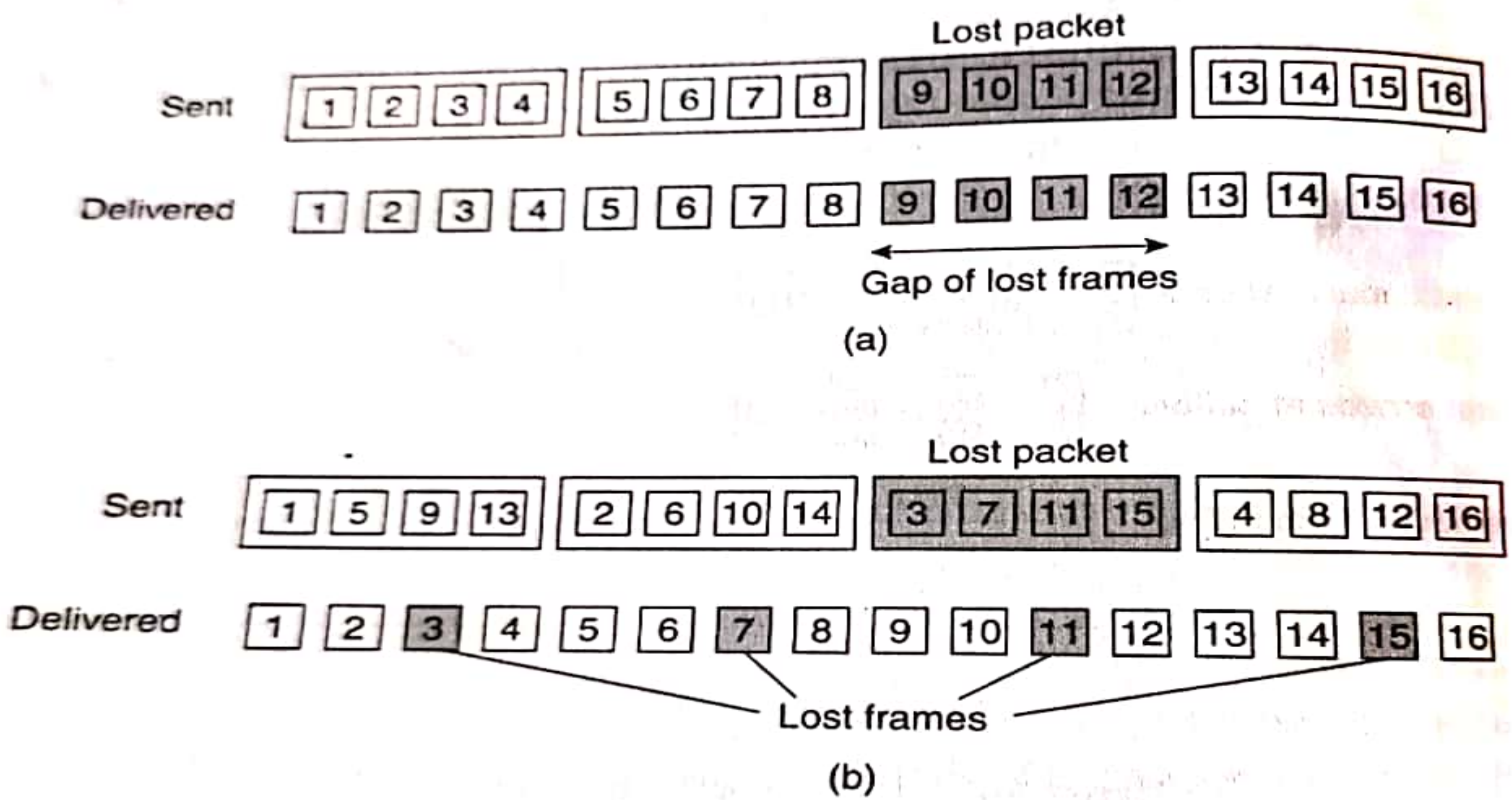
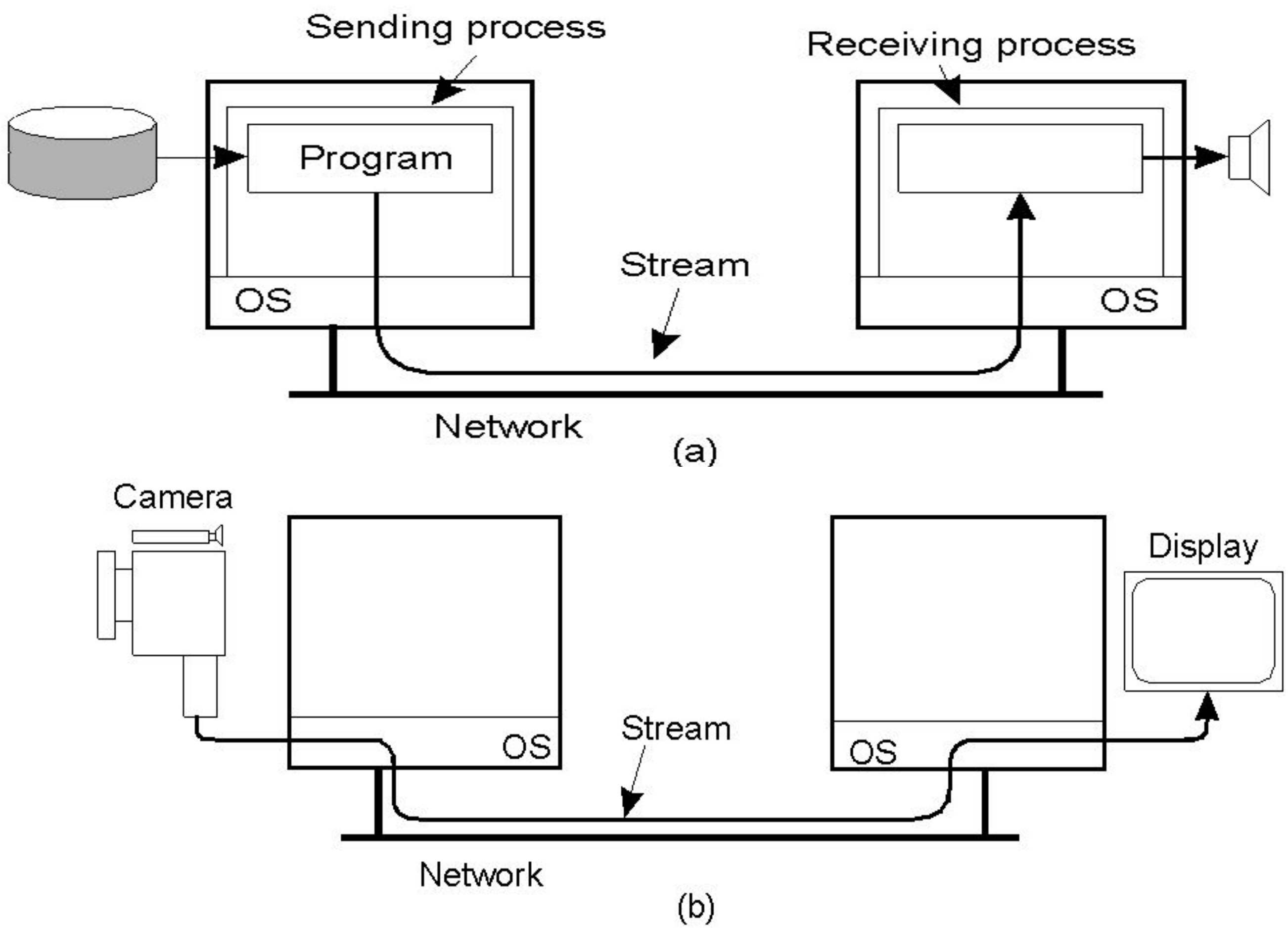


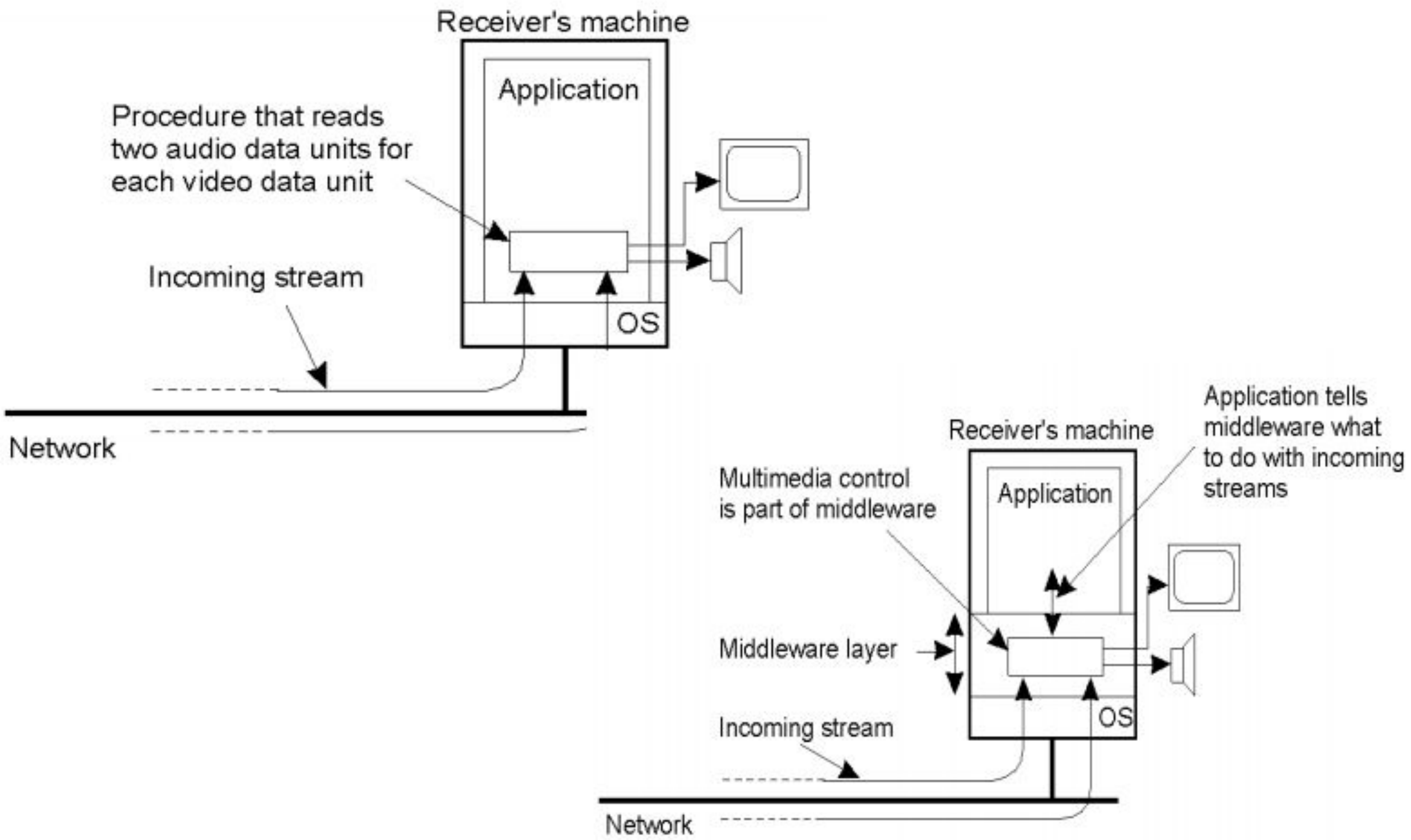
Figure 4-28. The effect of packet loss in (a) noninterleaved transmission and (b) interleaved transmission.

Data Stream (1)



- Setting up a stream between two processes across a network.

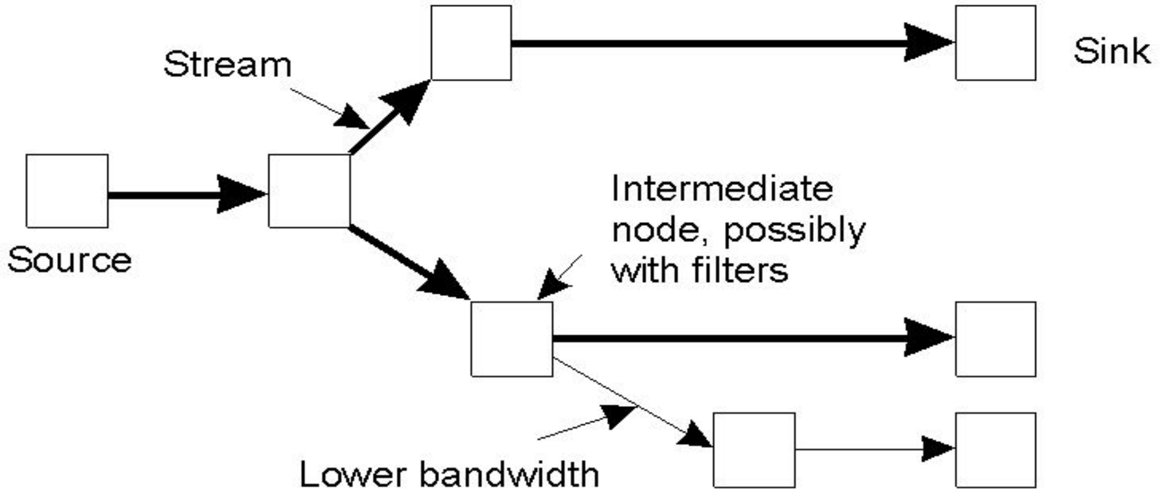
Data Stream (1)



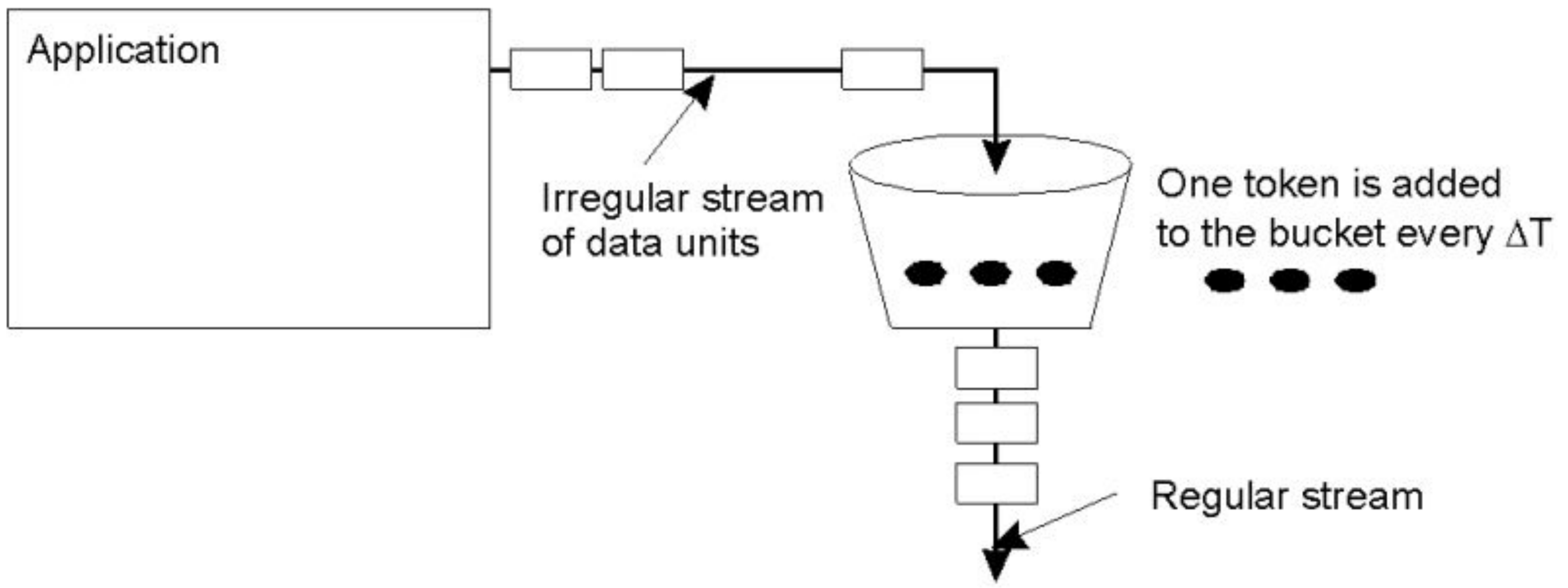
- Time-dependent and other requirements are specified as *quality of service (QoS)*
  - Requirements/desired guarantees from the underlying systems
  - Application specifies workload and requests a certain service quality
  - Contract between the application and the system

Data Stream

- An example of multicasting a stream to several receivers.



Token Bucket



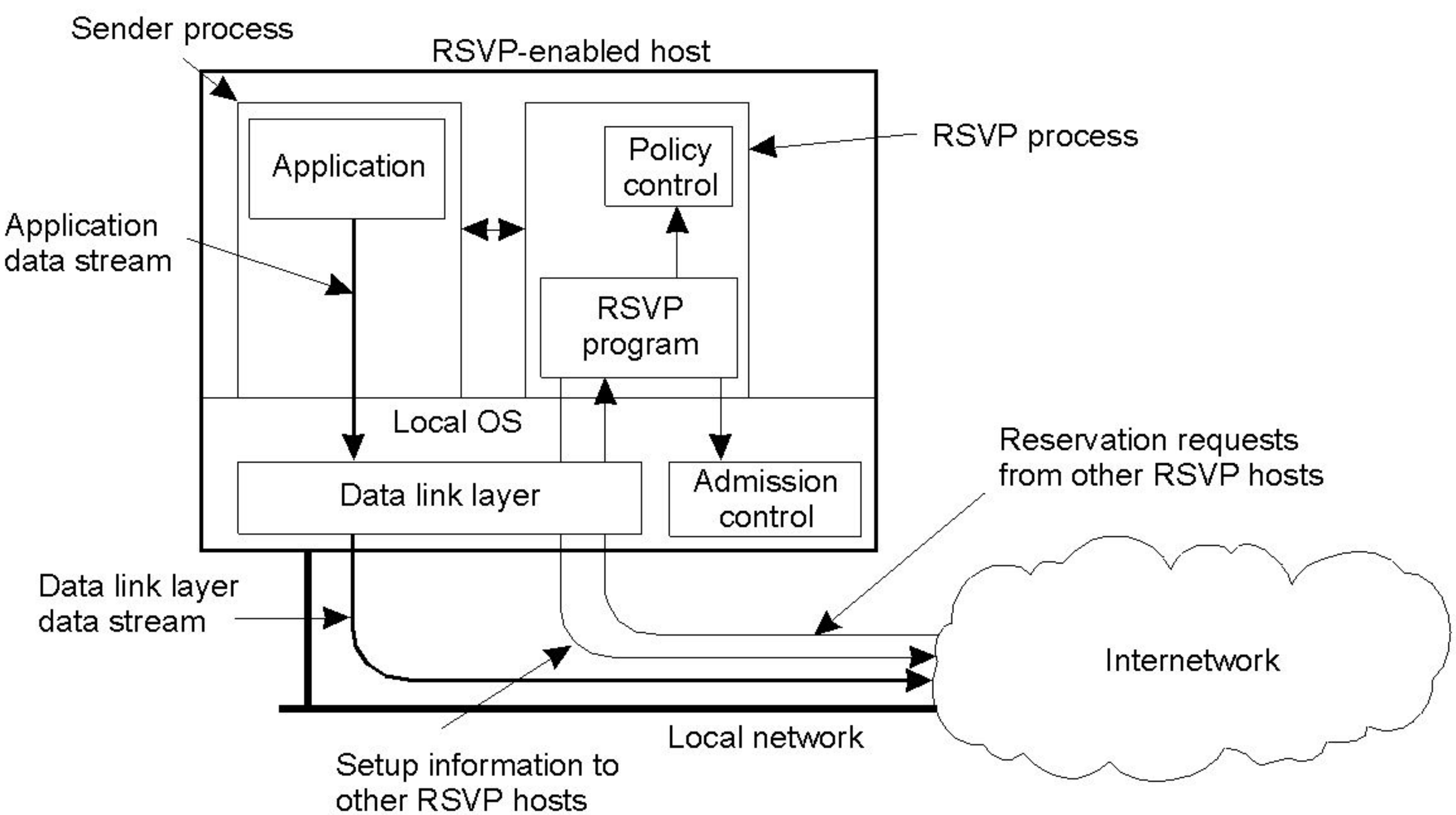
- The principle of a token bucket algorithm
  - Parameters (rate r, burst b)
  - Rate is the average rate, burst is the maximum number of packets that can arrive simultaneously

Specifying QoS (1)

Characteristics of the Input	Service Required
• maximum data unit size (bytes) • Token bucket rate (bytes/sec) • Toke bucket size (bytes) • Maximum transmission rate (bytes/sec)	• Loss sensitivity (bytes) • Loss interval (µsec) • Burst loss sensitivity (data units) • Minimum delay noticed (µsec) • Maximum delay variation (µsec) • Quality of guarantee

- A flow specification.

Setting Up a Stream



- The basic organization of RSVP for resource reservation in a distributed system.