VIT®
BHOPAL
www.vitbhopal.ac.in

CSE3009 - Parallel and Distributed Computing

Course Type: LTP Credits: 4

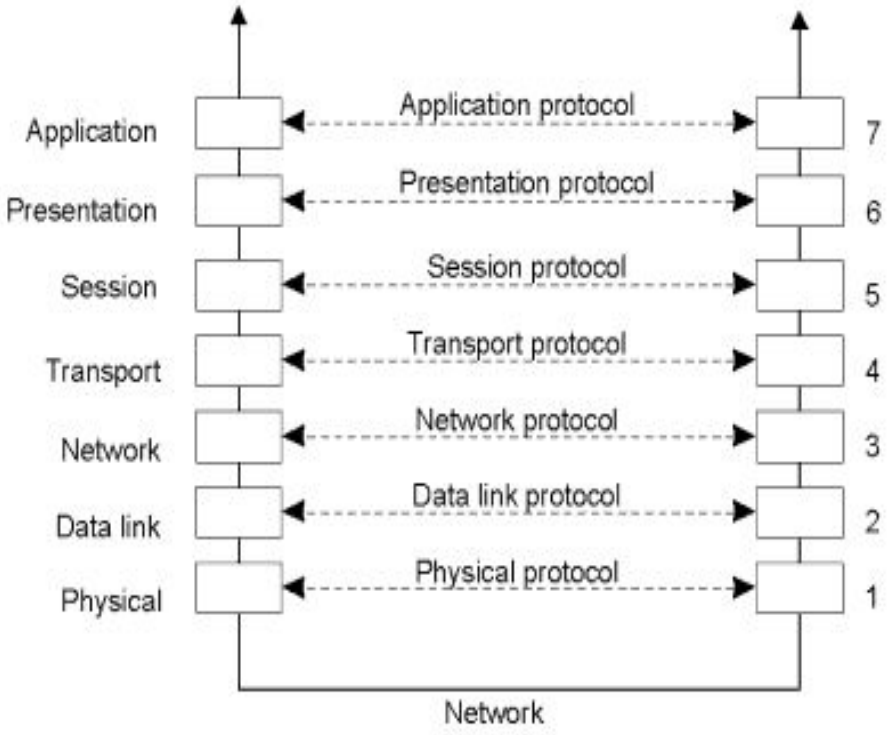
Prepared by
Dr Komarasamy G
Associate Professor (Grade-2)
School of Computing Science and Engineering
VIT Bhopal University

Unit-3

- Introduction to Distributed Systems – Definition, Issues, Goals, Types of distributed systems, Distributed System Models, Hardware concepts, Software Concept, Design Issues.
- **Communication – Layered Protocols, Remote Procedure Call, Remote Object Invocation, Message Oriented Communication, Stream Oriented Communication – Case Study (RPC and Java RMI).**
- Parallel Random Access Machine (PRAM) model, PRAM architecture.

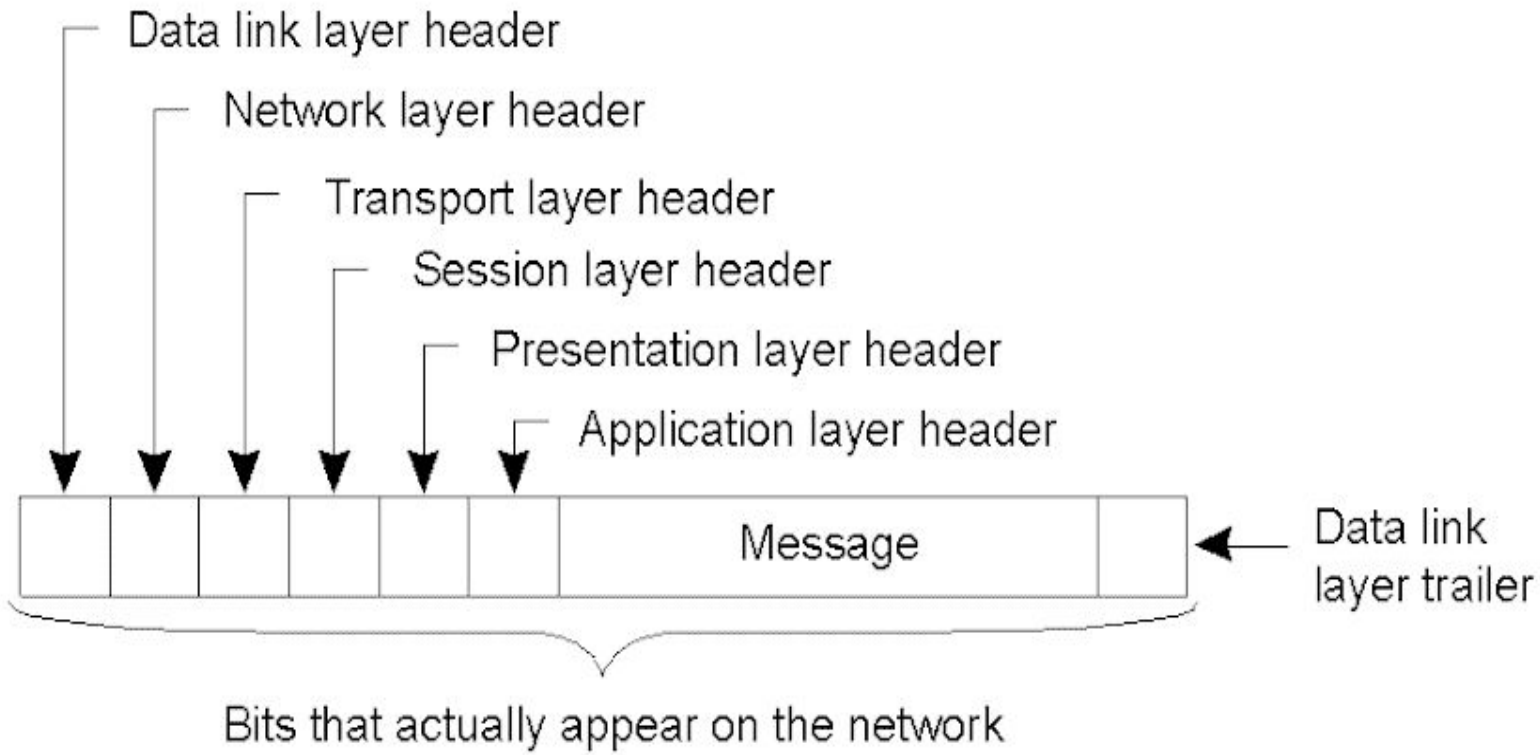
Layered Protocols

- **OSI**: Open Systems Interconnection
- Developed by the International Organization for Standardization (ISO)
- Provides a **generic framework** to discuss the layers and functionalities of communication protocols.



Layers, interfaces, and protocols in the OSI model.

OSI Model (con.t)



A typical message as it appears on the network.

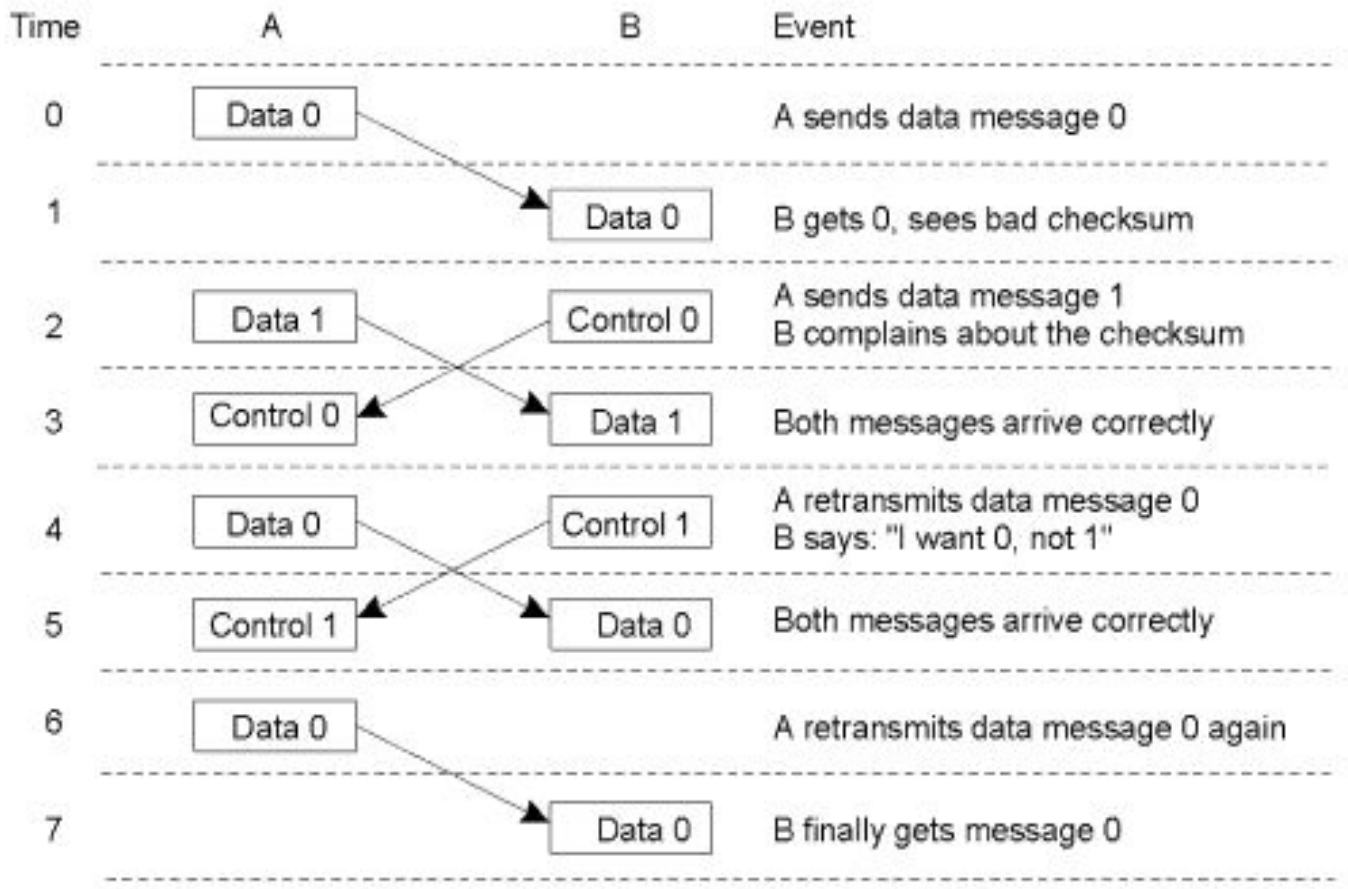
OSI Protocol Layers

- **Physical layer**
  - Deals with the transmission of bits
  - Physical interface between data transmission device
  - (e.g. computer) and transmission medium or network
  - **Concerned with:**
    - Characteristics of transmission medium, Signal levels, Data rates

- **Data link layer:**
  - Deals with detecting and correcting bit transmission errors
  - Bits are group into frames
  - Checksums are used for integrity

OSI Protocol Layers (con.t)

- **Discussion between a receiver and a sender in the data link layer.**

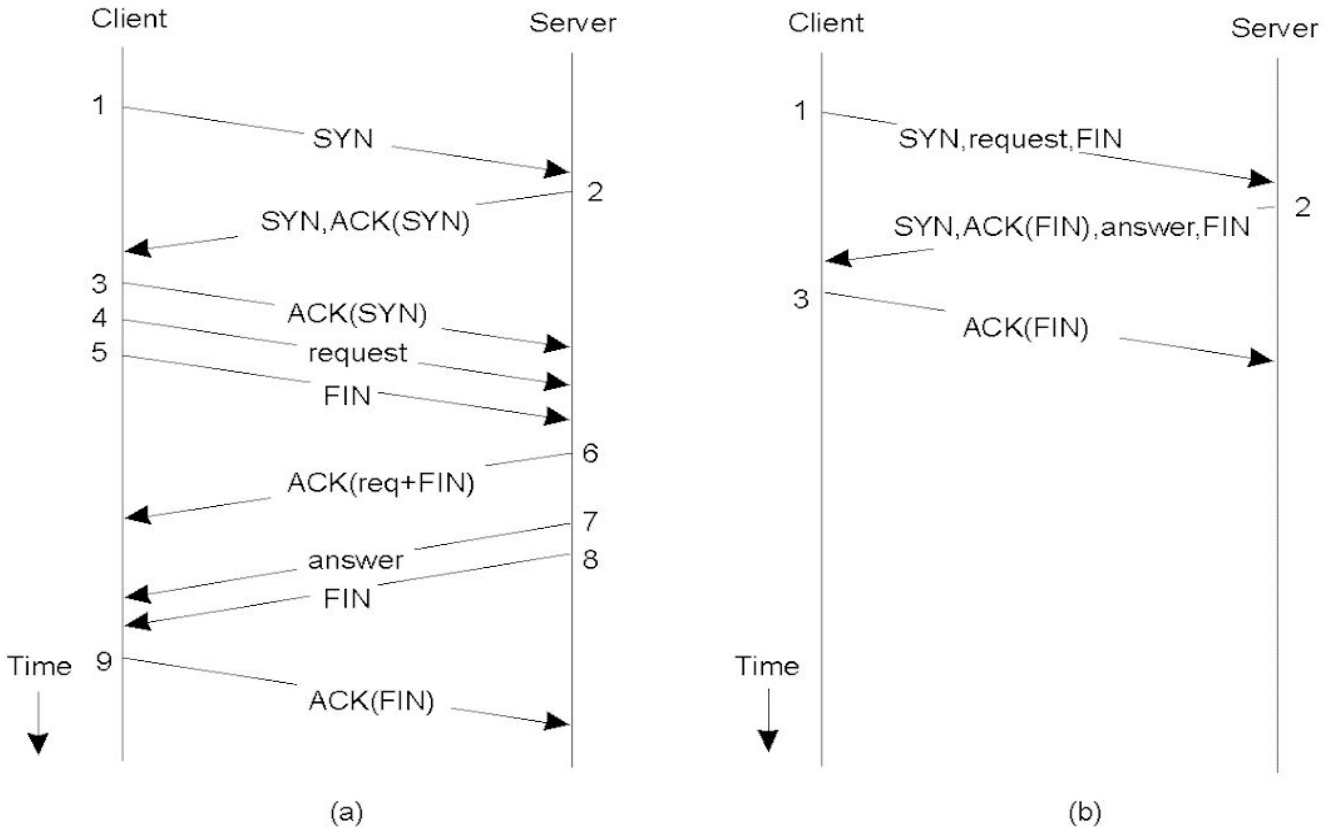


OSI Protocol Layers (con.t)

- **Network layer:**
  - Performs multi-hop routing across multiple networks
  - Implemented in end systems and router
- **Transport layer:**
  - Packing of data
  - Reliable delivery of data (breaks message into pieces small enough, assign each one a sequence number and then send them)
  - Ordering of delivery
  - **Examples:**
    - TCP (connection-oriented)
    - UDP (connectionless)
    - RTP (Real-time Transport Protocol)

OSI Protocol Layers (con.t)

Client-Server TCP protocol



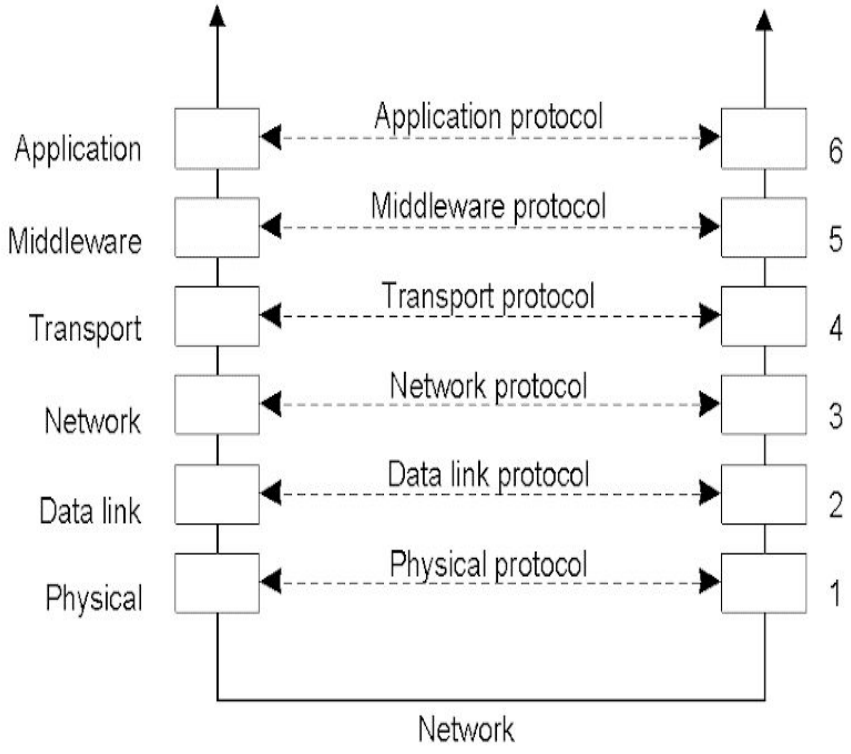
(a) Normal operation of TCP. (b) Transactional TCP.

OSI Protocol Layers (con.t)

- **Session layer**
  - Provide dialog control to keep track of which party is talking and it provides synchronization facilities
- **Presentation layer**
  - Deals with non-uniform data representation and with compression and encryption
- **Application layer**
  - Support for user applications
  - e.g. HTTP, SMPT, FTP

Middleware Protocols

- **Support high-level communication services**
- **The session and presentation layers are merged into the middleware layer,**
- **Ex: Microsoft ODBC (Open Database Connectivity), OLE DB…**



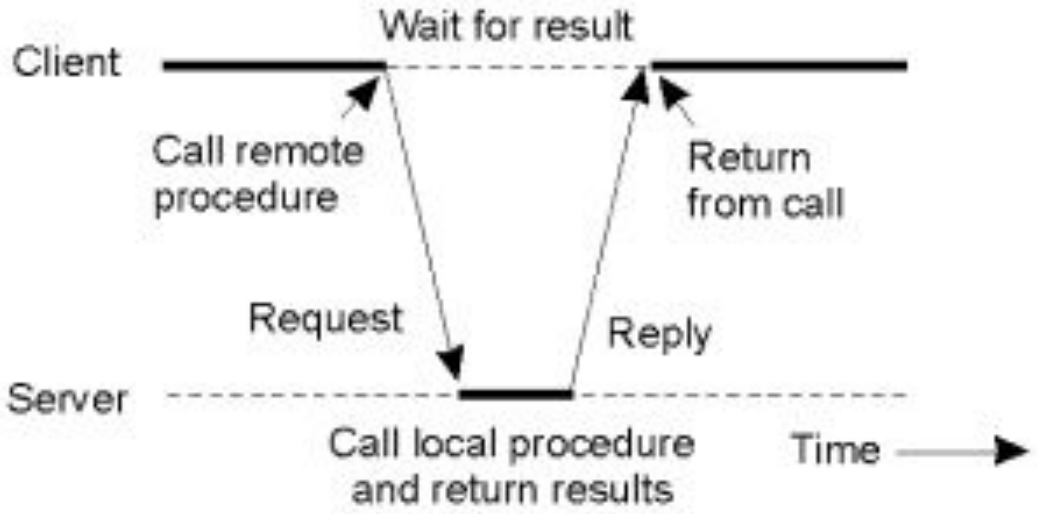
An adapted reference model for networked communication.

Remote Procedure call

- **Basic idea:** To execute a procedure at a remote site and ship the results back.
- **Goal**: To make this operation as distribution transparent as possible (i.e., the remote procedure call should look like a local one to the calling procedure).

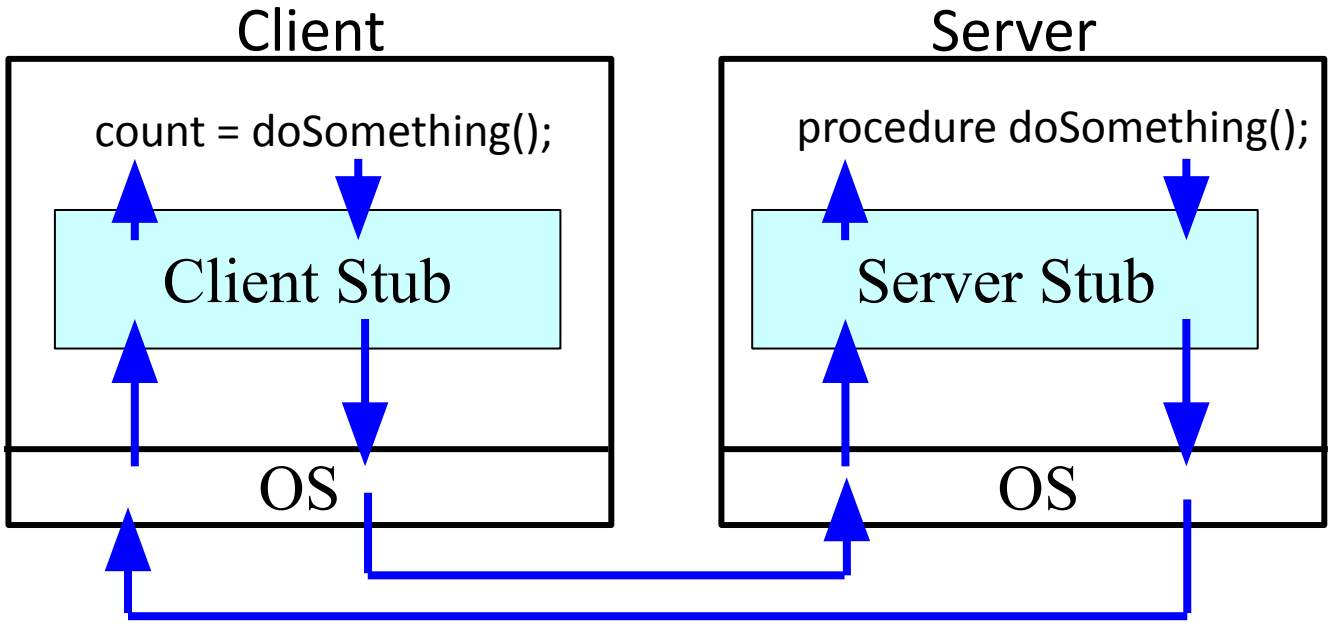
Example:

read(fd, buf, nbytes)



Client and Server Stubs

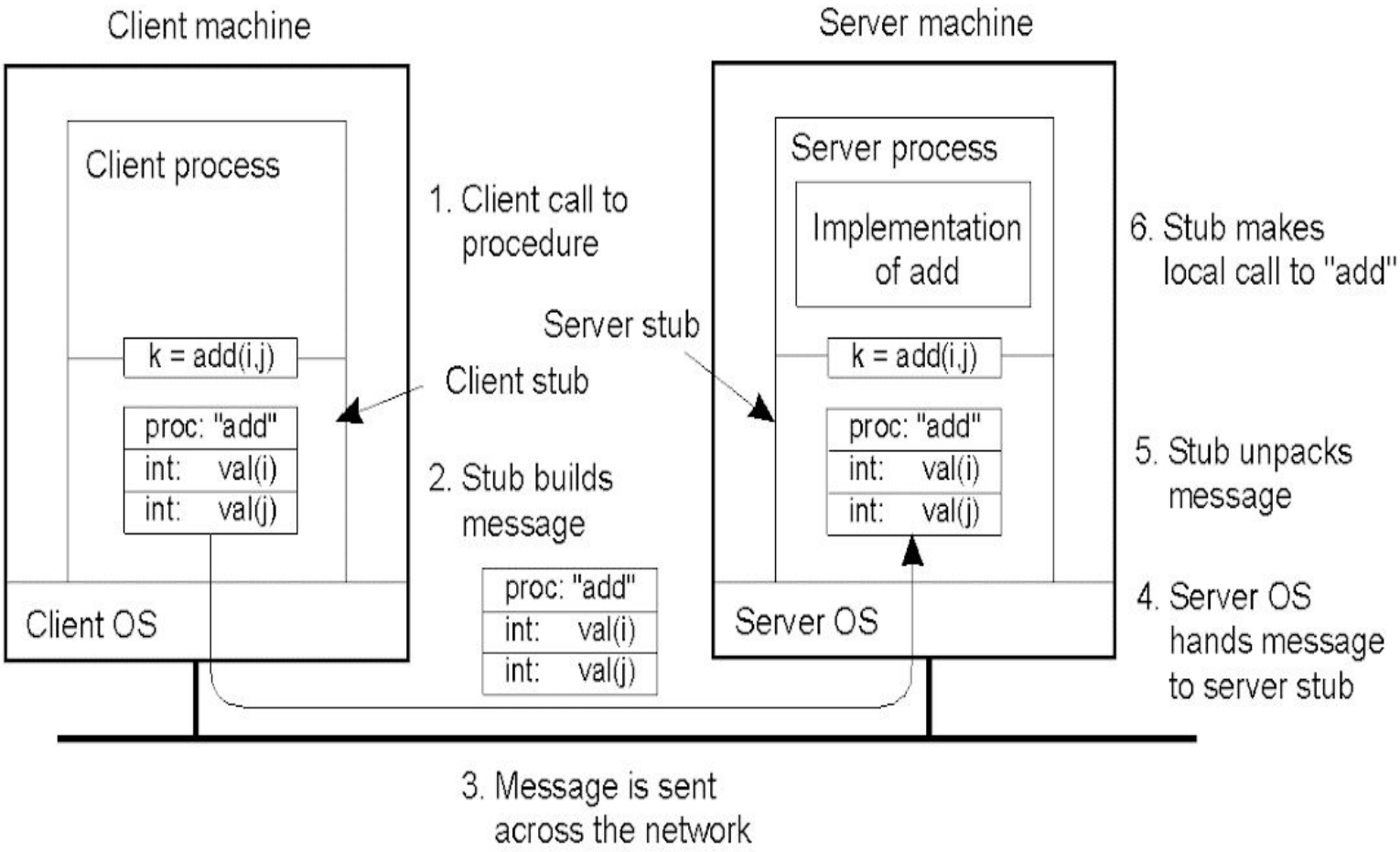
Definition: Are additional functions which are added to the main functions in order to support for RPC



Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

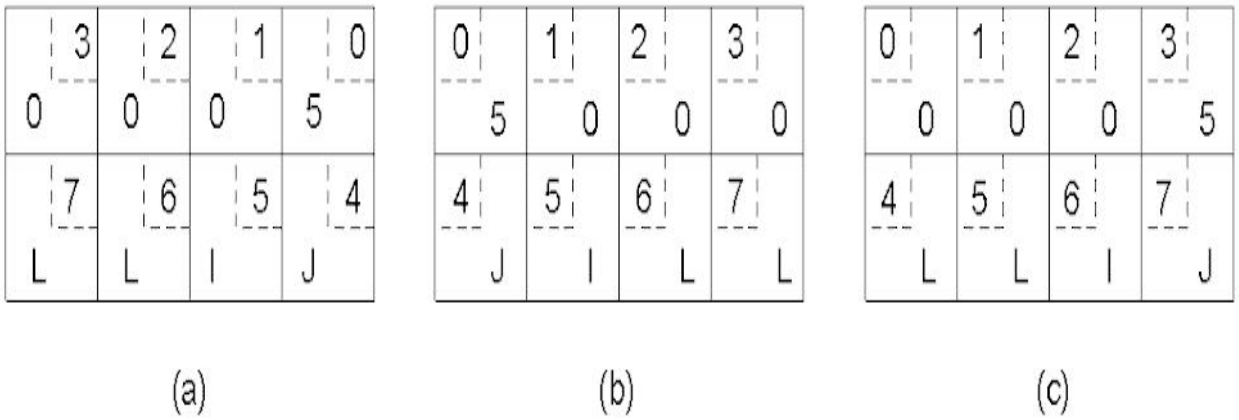
Passing Value Parameters (1)



Steps involved in doing remote computation through RPC

Passing Value Parameters (2)

- In a large distributed system, multiple machine types are present
- Each machine has its own representation for number, characters, and others data items.



(a) (b) (c)

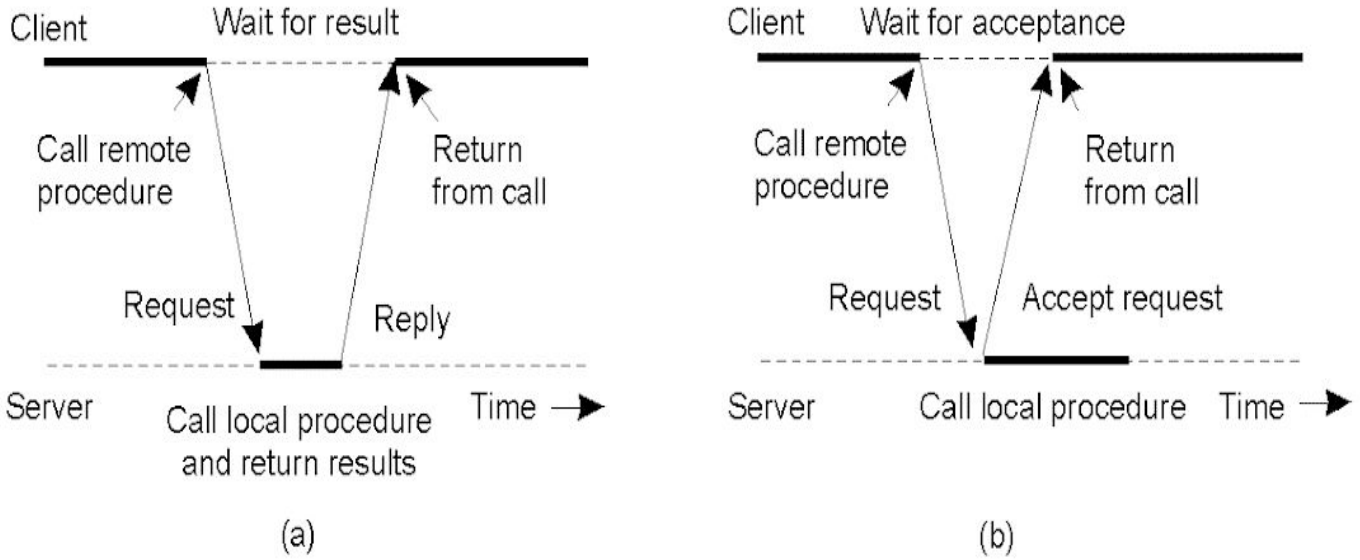
a) Original message on the Pentium (little-endian)

b) The message after receipt on the SPARC (big-endian)

c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Asynchronous RPC (1)

- Avoids blocking of the client process.
- Allows the client to proceed without getting the final result of the call.

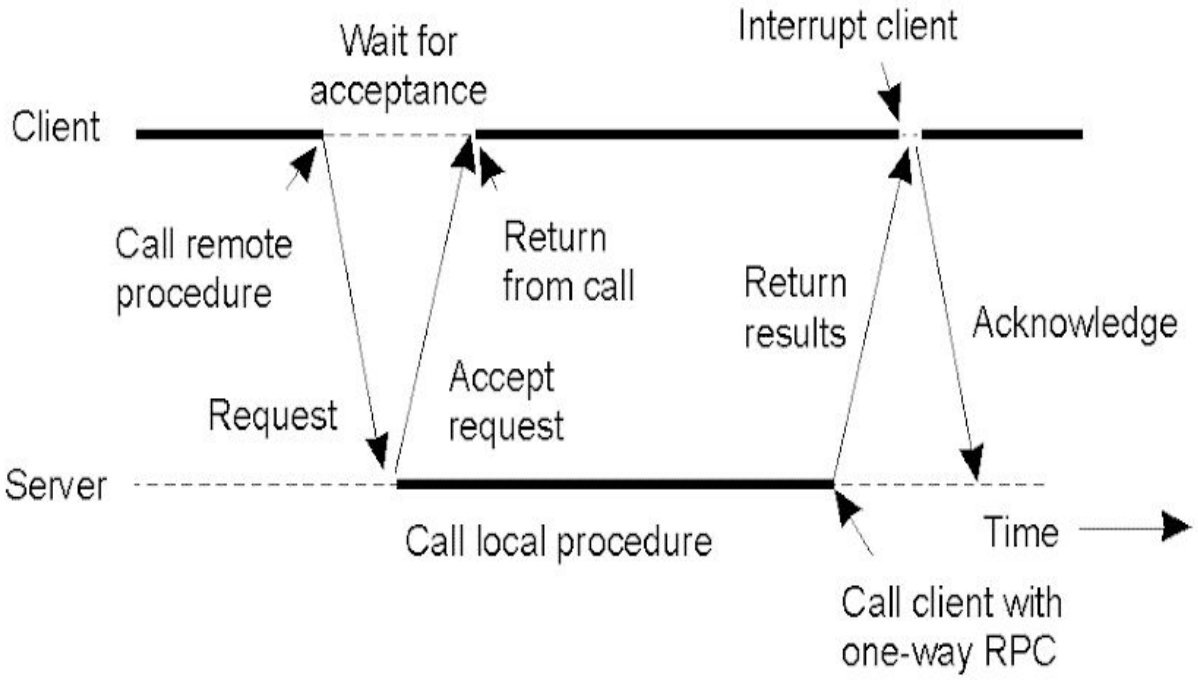


a) The interconnection between client and server in a traditional RPC

b) The interaction using asynchronous RPC

Differed synchronous

One-way RPC model: client does not wait for an acknowledgement of the server's acceptance of the request.



A client and server interacting through two asynchronous RPCs

Example DCE RPC

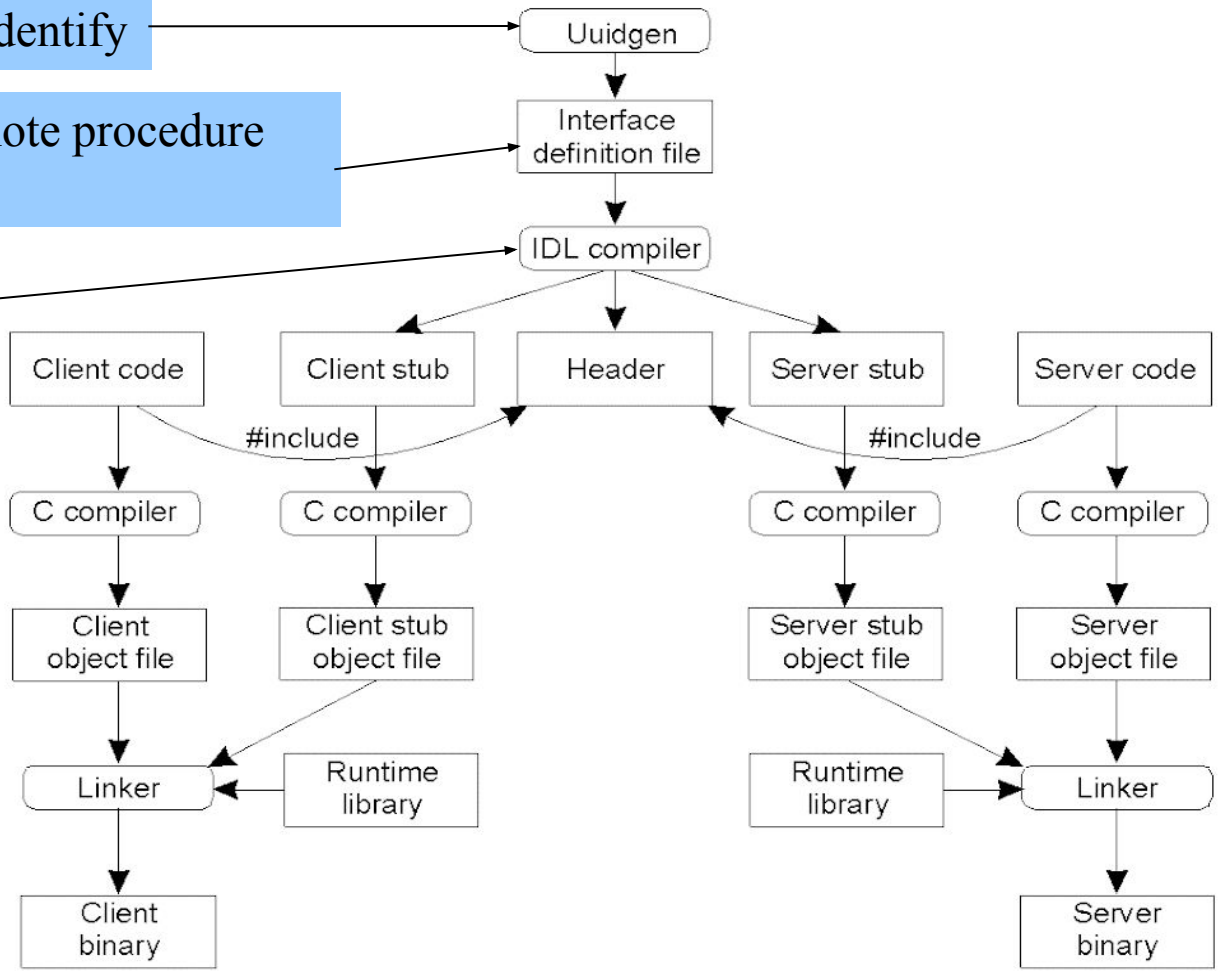
- **What is DCE ?** (Distributed Computing Environment)
  - DCE is a true middleware system in that it is designed to execute as a layer of abstraction between exiting (network) operating system and distributed application.
- **-Goals of DCE RPC**
  - Makes it possible for client to access a remote service by simply calling a local procedure.
- **Components:**
  - Languages
  - Libraries
  - Daemon
  - Utility programs
  - Others

Writing a Client and a Server

Generate a prototype IDL file containing an interface identify

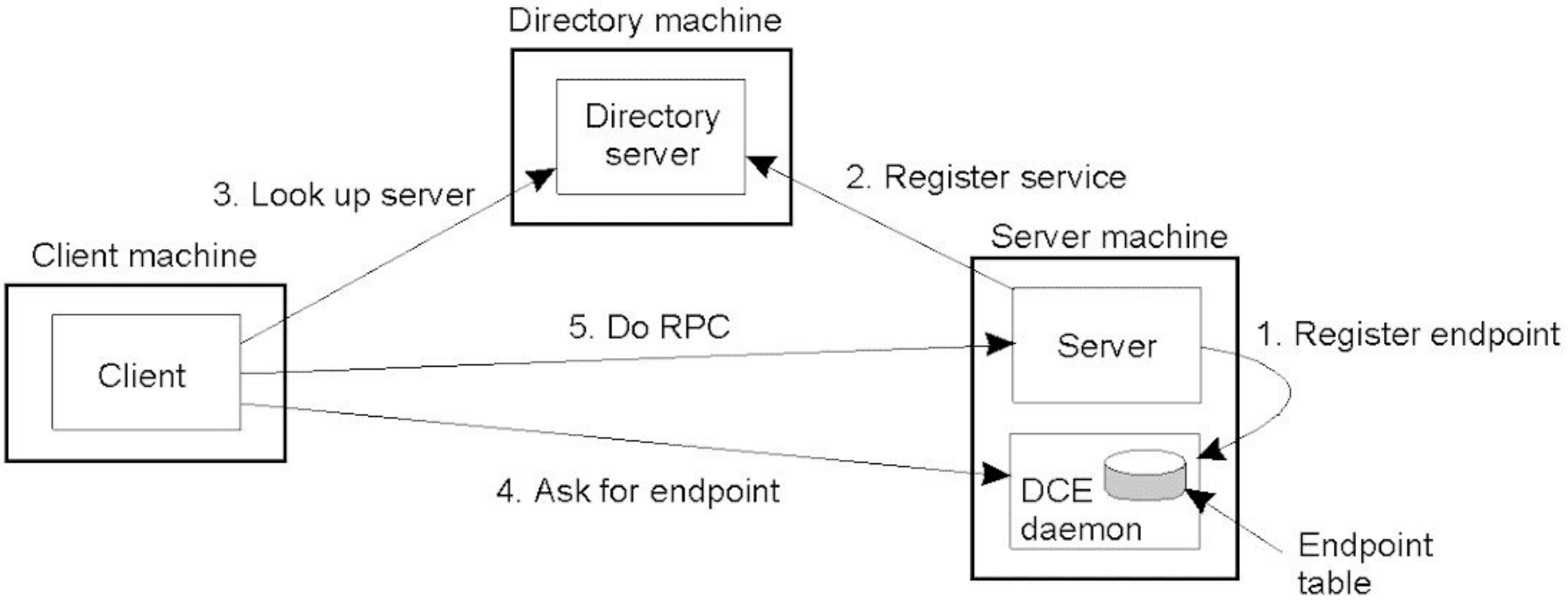
Filling in the names of remote procedure and their parameters

IDL compiler is call to compile into three files



The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server

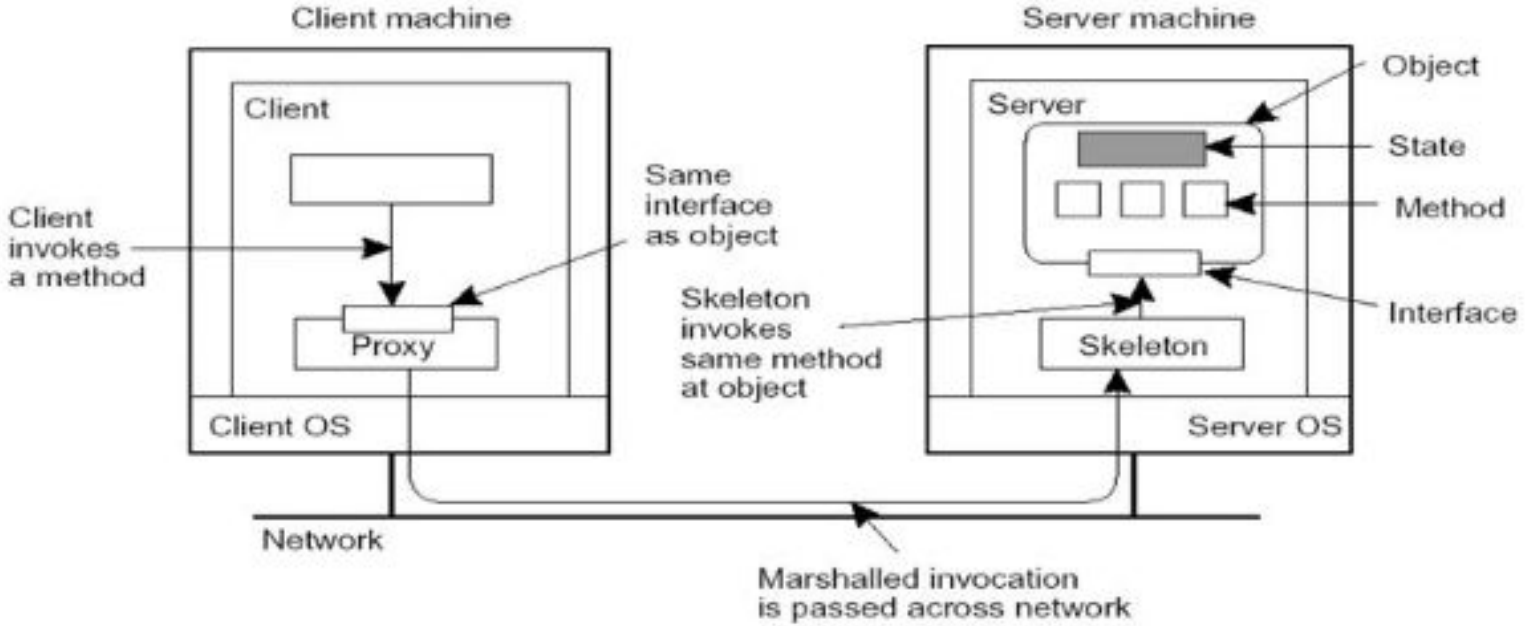


Endpoint (port) is used by server's OS to distinguish incoming message

Client-to-server binding in DCE.

Remote Object Invocation

- Data and operations **encapsulated** in an object
- Operations are implemented as **methods**, and are accessible through **interfaces**
- Object offers only its **interface** to clients
- **Object server** is responsible for a collection of objects
- **Client stub (proxy)** implements the interface
- **Server skeleton** handles (un)marshaling and object invocation



Remote Object Invocation

Compile-time objects: Language-level objects, from which proxy and skeletons are automatically generated.

Runtime objects: Can be implemented in any language, but require use of an **object adapter** that makes the implementation *appear* as an object.

Transient objects: live only by virtue of a server: if the server exits, so will the object.

Persistent objects: live independently from a server: if a server exits, the object's state and code remain (passively) on disk.

Remote Object Invocation

Object reference: Having an object reference allows a client to **bind** to an object:

- Reference denotes server, object, and communication protocol
- Client loads associated stub code
- Stub is instantiated and initialized for specific object

Two ways of binding:

- **Implicit:** Invoke methods directly on the referenced object
- **Explicit:** Client must first explicitly bind to object before invoking it

RMI vs RPC

- The primary difference between RMI and RPC:
  - PRC is used for procedure – based application.
  - RMI is used for distributed object systems.
  - RMI is object oriented.
  - PRC is procedural.

Message-Oriented Communication

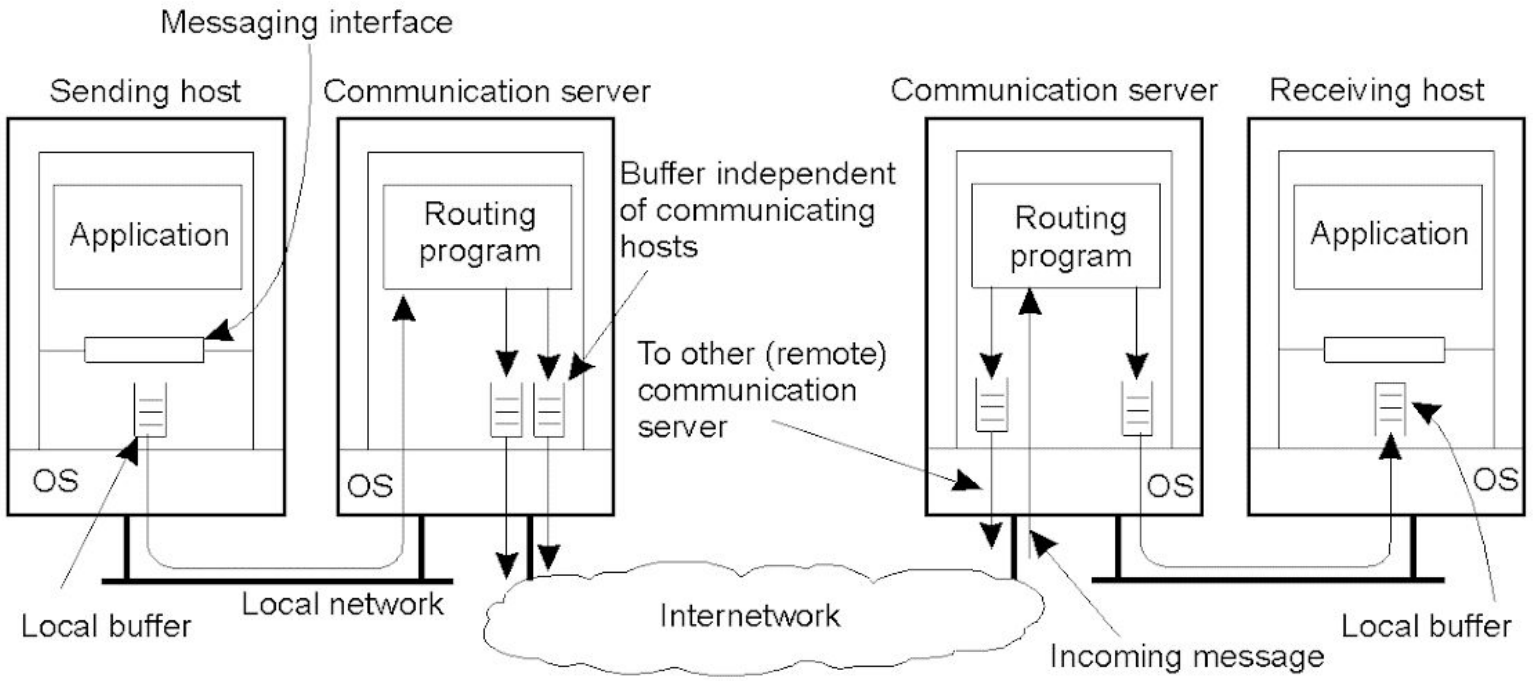
- RPC and RMI enhanced access transparency
- But client have to blocked until its request has been processed
- Message-Oriented Communication can solve this problem by ‘messaging’
- Index:
  - Meaning of synchronous behavior and its implication
  - Various messaging systems
  - Message-queuing system

Message-Oriented Communication

- Remote procedure calls and remote object invocations contribute to **hiding communication in distributed systems**, that is, they enhance access transparency.
- 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.

Message-Oriented Communication

- General communication system connected through a network
  - Each host is connected to a single communication server.
  - Hosts and communication servers can have buffers.



Message-Oriented Communication

- **Message-Oriented Transient Communication**
- Many distributed systems and applications are built directly on top of the **simple message-oriented model** offered by the transport layer.
- To better understand and appreciate the message-oriented systems as part of middleware solutions, we first discuss messaging through transport-level sockets.
- **Berkeley Sockets**
- Special attention has been paid to standardizing the interface of the transport layer to allow programmers to make use of its entire suite of (messaging) protocols through a simple set of primitives.
- Also, standard interfaces make it easier to port an application to a different machine.

Message-Oriented Communication

- **Socket is a communication end point 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.
- A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol. In the following text, we concentrate on the socket primitives for TCP, which are **shown in Fig.**

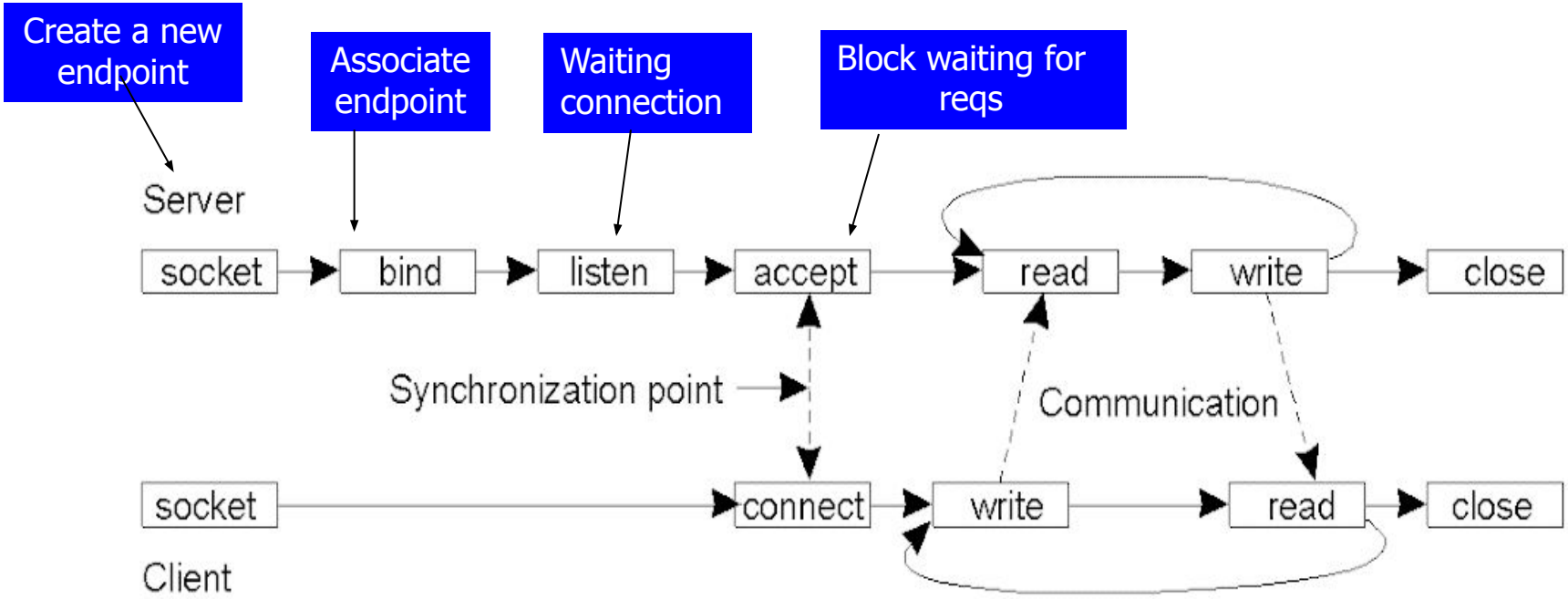
Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive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

Message-Oriented Communication

- Let us now take a look at the **client side.** Here, too, a socket must first be created using the socket primitive, but explicitly binding the socket to a local address is not necessary, since the operating system can dynamically allocate a port when the connection is set up.
- The connect primitive requires that the caller specifies the transport-level address to which a connection request is to be sent.
- The client is blocked until a connection has been set up successfully, after which both sides can start exchanging information through the send and receive primitives.
- Finally, closing a connection is symmetric when using sockets, and is established by having both the client and server call the close primitive.

Message-Orient Transient Communication

- Berkeley Sockets
  - Connection-oriented communication pattern using sockets
  - Sockets considered insufficient because:
    - Support only send and receive primitives
    - Designed for communication using general-purpose protocol such as TCP/IP



Message-Oriented Communication

- **The Message-Passing Interface (MPI)**
- With the advent of high-performance multi computers, developers have been looking for message-oriented primitives that would allow them to **easily write highly efficient applications**.
- This means that the primitives should be at a convenient level of abstraction (to ease application development), and that their implementation incurs only minimal overhead.
- **Sockets were deemed insufficient for two reasons.**
- First, they were at the wrong level of abstraction by supporting only simple send and receive primitives.
- Second, sockets had been designed to communicate across networks using general-purpose protocol stacks such as TCP/IP.

Message-Oriented Communication

- The Message-Passing Interface(MPI)
  - Some of the most intuitive message-passing primitives of MPI

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 are none
MPI_irecv	Check if there is an incoming message, but do not block

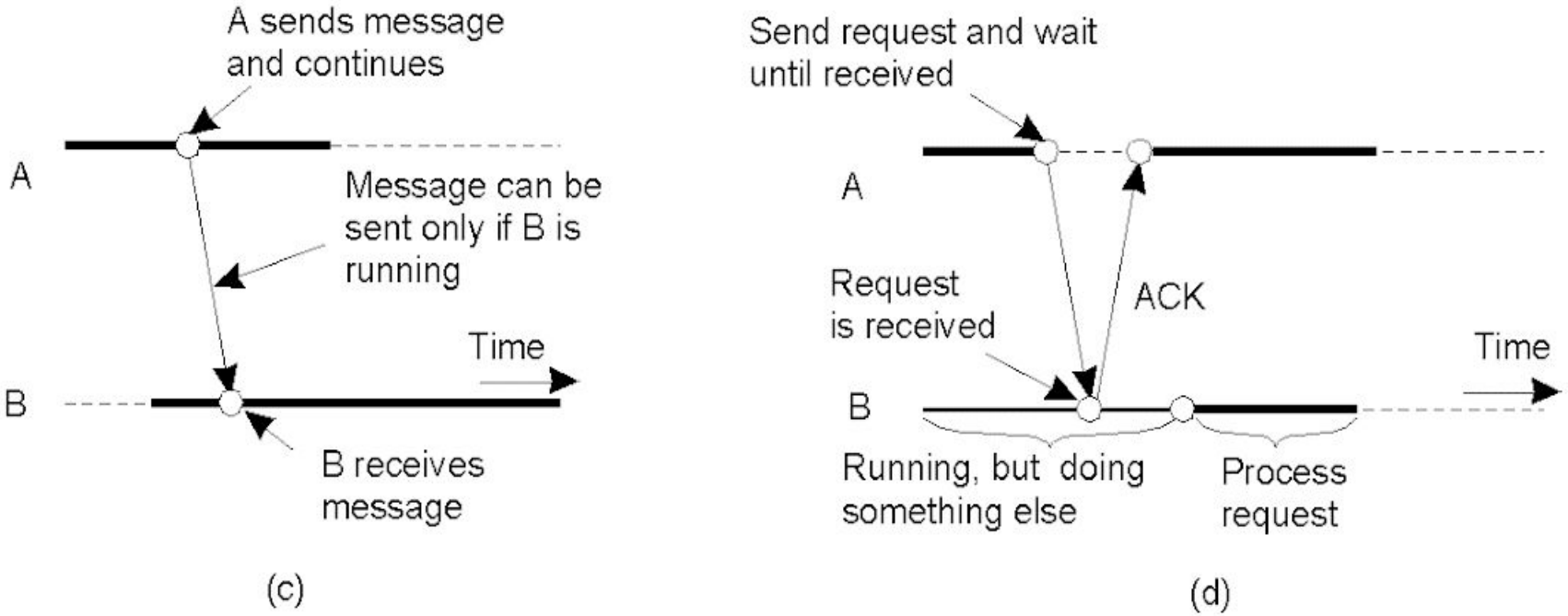
Message-Orient Persistent Communication

- Message-Queuing Model
  - Apps communicate by inserting messages in specific queues
    - Loosely-couple communication
  - Support for:
    - Persistent asynchronous communication
    - Longer message transfers(e.g., e-mail systems)
  - Basic interface to a queue in a message-queuing system:

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

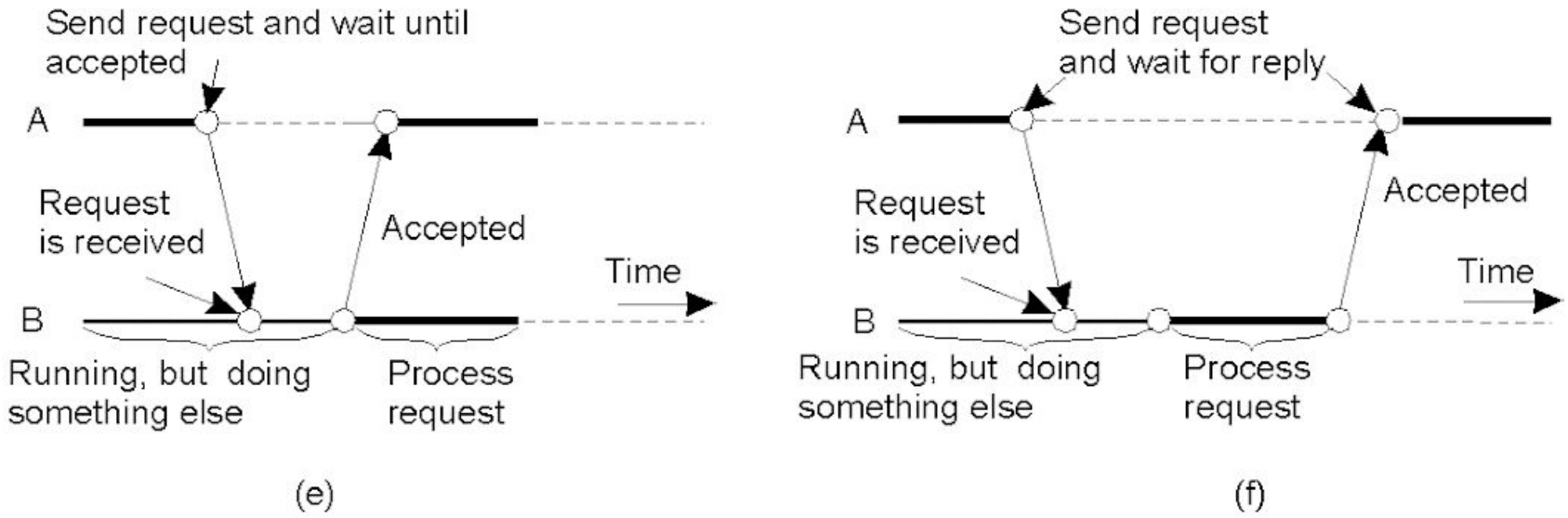
Persistence and Synchronicity in Communication(5)

- Transient asynchronous/Receipt-based transient synchronous communication



(c) Transient asynchronous communication / (d) receipt-based transient synchronous communication

Persistence and Synchronicity in Communication(6)



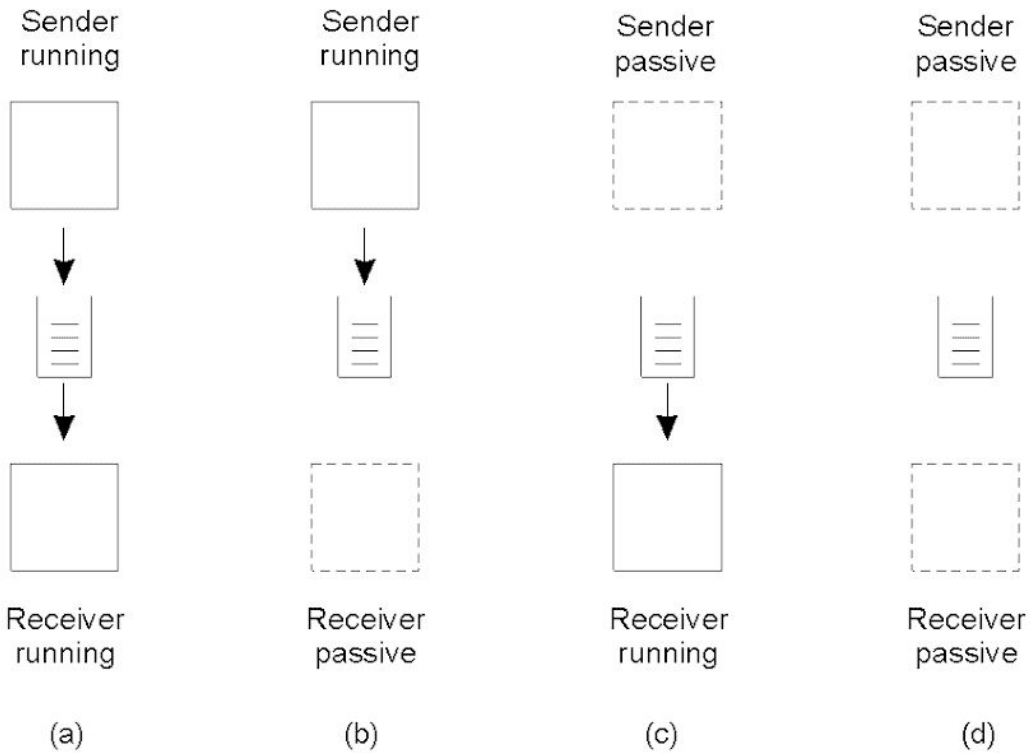
(e) Delivery-based transient synchronous communication at message delivery
(f) Response-based transient synchronous communication

Message-Orient Transient Communication

- **Message-Oriented Model**
  - Many distributed systems and applications are built on top of the simple message-oriented model
  - These models are offered by Transport Layer
  - Message-oriented models
    - Berkeley Sockets: Socket interface as introduced in Berkeley UNIX
    - The Message-Passing Interface(MPI): designed for parallel applications and as such in tailored to transient communication

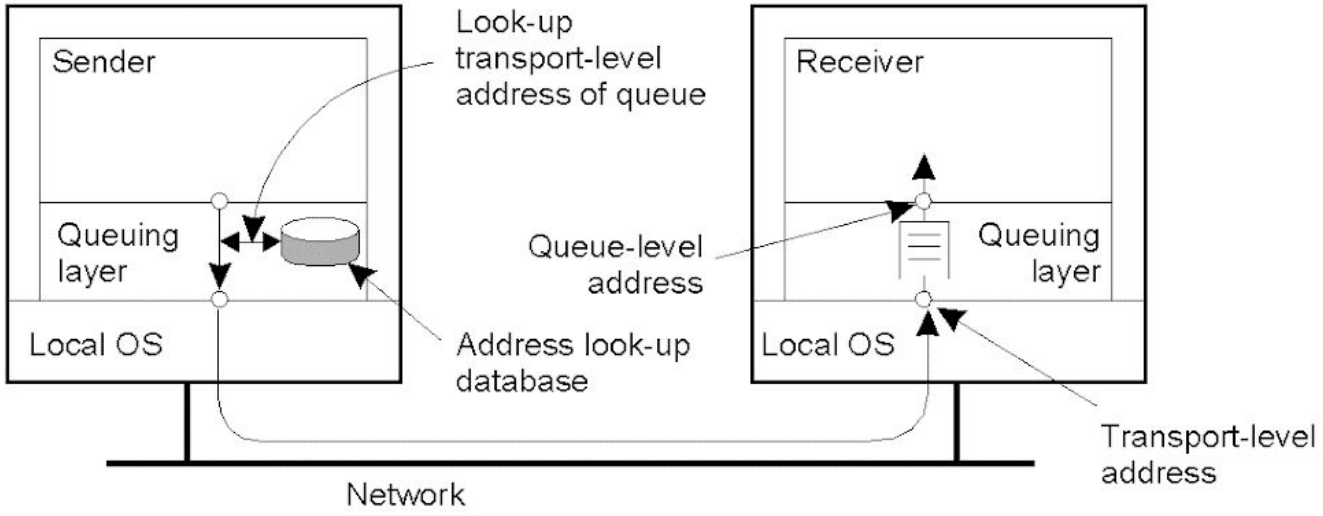
Message-Orient Persistent Communication

- **Message-Queuing Model**
  - Four combinations for loosely-coupled communication using queues:



Message-Orient Persistent Communication

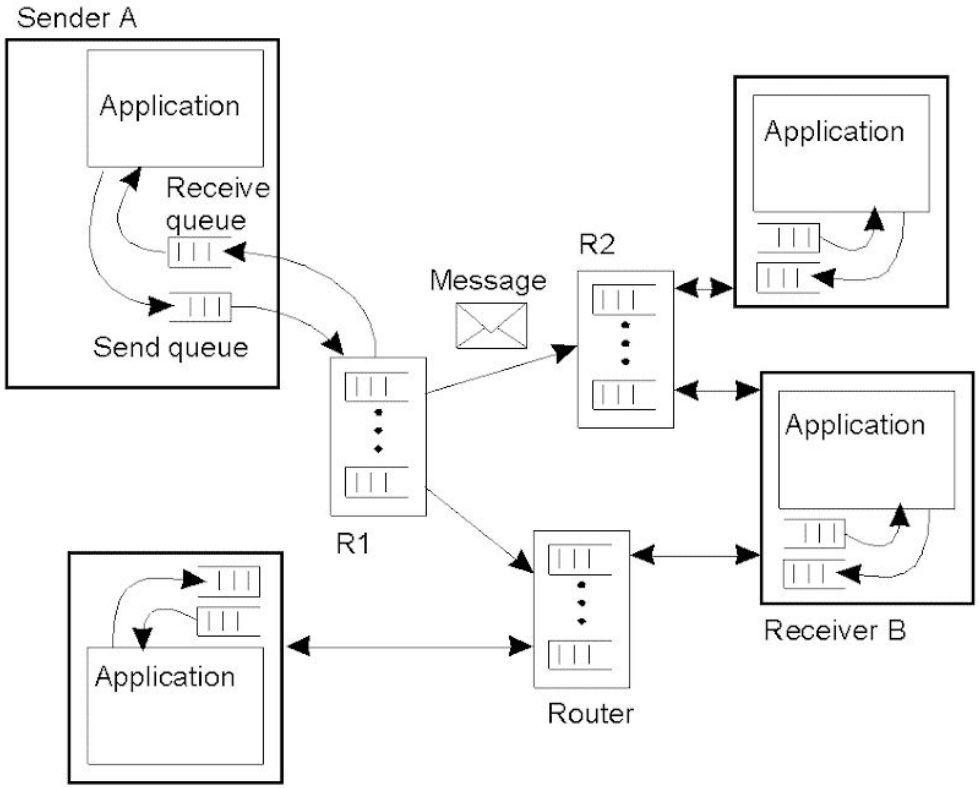
- **General architecture of a Message-Queuing System**
  - Messages can only be put and received from local queues
  - Ensure transmitting the messages between the source queues and destination queues, meanwhile storing the messages as long as necessary
  - Each queue is maintained by a queue manager



The relationship between queue-level addressing and network-level addressing

Message-Orient Persistent Communication

- **General architecture of a Message-Queuing System**
  - Queue managers are not only responsible for directly interacting with applications but are also responsible for acting as relays (or routers)

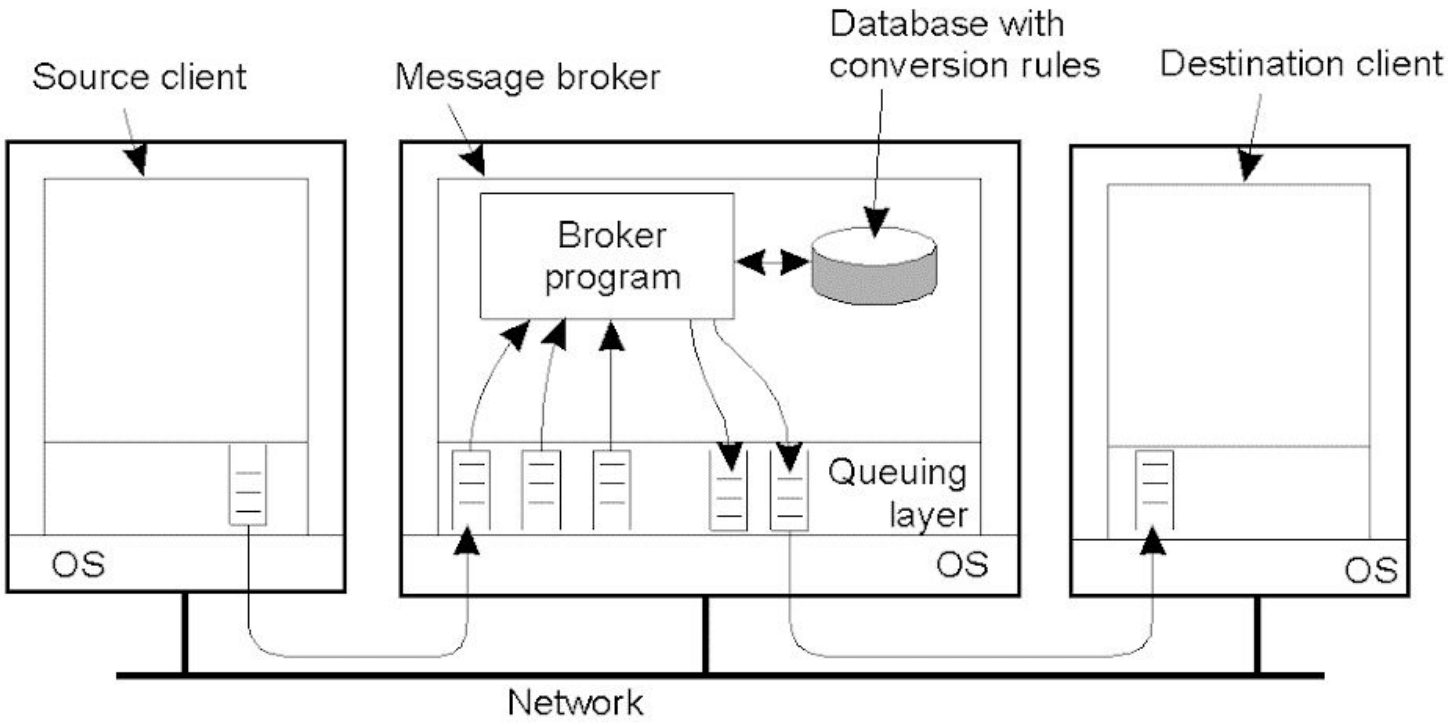


Queue managers form an overlay network, acting as routers

Message-Orient Persistent Communication(5)

- **General–purpose of a Message-Queuing System**
  - Enable persistent communication between processes
  - Handling access to database
  - In wide range of application, include:
    - Email
    - Groupware
    - Batch processing

Message-Orient Persistent Communication



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

Summary & Conclusion

- **Summary**
  - Two different communication concept 'Transient vs. Persistent'
    - Persistent messages are stored as long as necessary
    - Transient messages are discarded when they cannot be delivered
  - Message-Oriented Transient Comm.
    - Berkeley socket and MPI
  - Message-Oriented Persistent Comm.
    - Message-Queuing Model and Message Broker
- **Conclusion**
  - Message-Oriented communication solve the blocking problems that may occur in general communication between Server/Client
  - Message-Queuing systems can users(including applications) to do Persistent communication

Stream-Oriented Communication

- **Types of media**
  - **Continuous media**
    - Temporal dependence between data items
    - ex) Motion - series of images
  - **Discrete media**
    - No temporal dependence between data items
    - ex) text, still images, object code or executable files

Stream-Oriented Communication

- **Data Stream**
  - A data stream is **nothing but a sequence of data units**. Data streams can be applied to discrete as well as continuous media. For example, UNIX pipes or TCPIIP connections are typical examples of (byte-oriented) discrete data streams.
  - Playing an audio file typically requires setting up a continuous data stream between the file and the audio device.
  - To capture the exchange of time-dependent information, distributed systems generally provide support for data streams.

Stream-Oriented Communication

- **Transmission modes**
- In synchronous transmission mode, **there is a maximum end-to-end delay defined for each unit in a data stream.**
- Whether a data unit is transferred much faster than the maximum tolerated delay is not important.
- For example, a **sensor may sample temperature at a certain rate and pass it through a network to an operator.**
- In that case, it may be important that the end-to-end propagation time through the network is guaranteed to be lower than the time interval between taking samples, but it cannot do any harm if samples are propagated much faster than necessary.

Stream-Oriented Communication

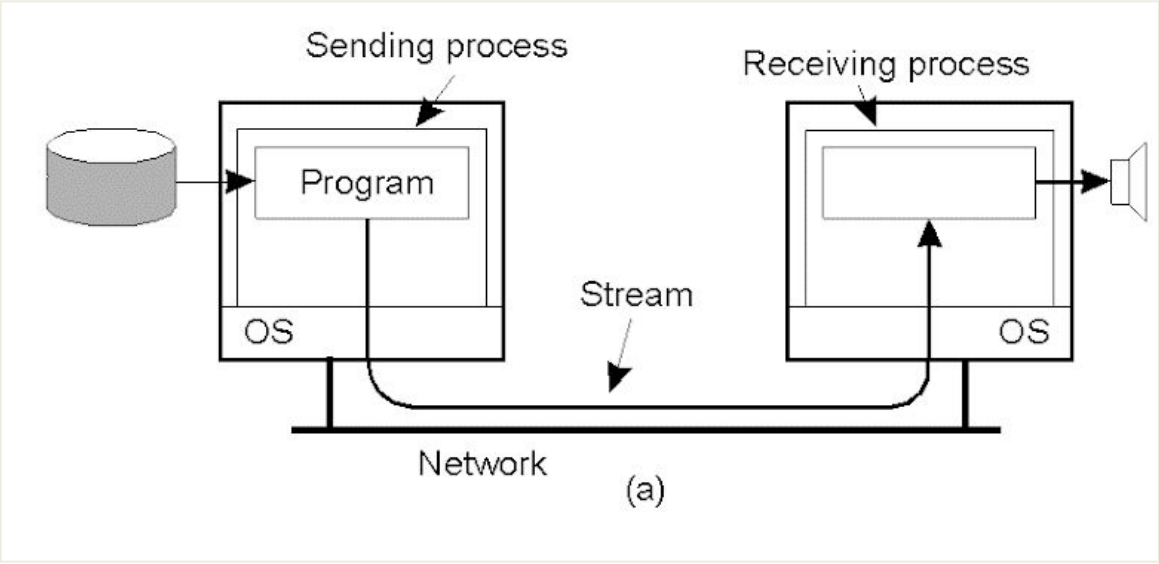
- Types of stream
  - Simple stream
    - consist of only a single sequence of data
  - Complex stream
    - consist of several related simple stream
    - ex) stereo audio, movie
  - Substream
    - related simple stream
    - ex) stereo audio channel

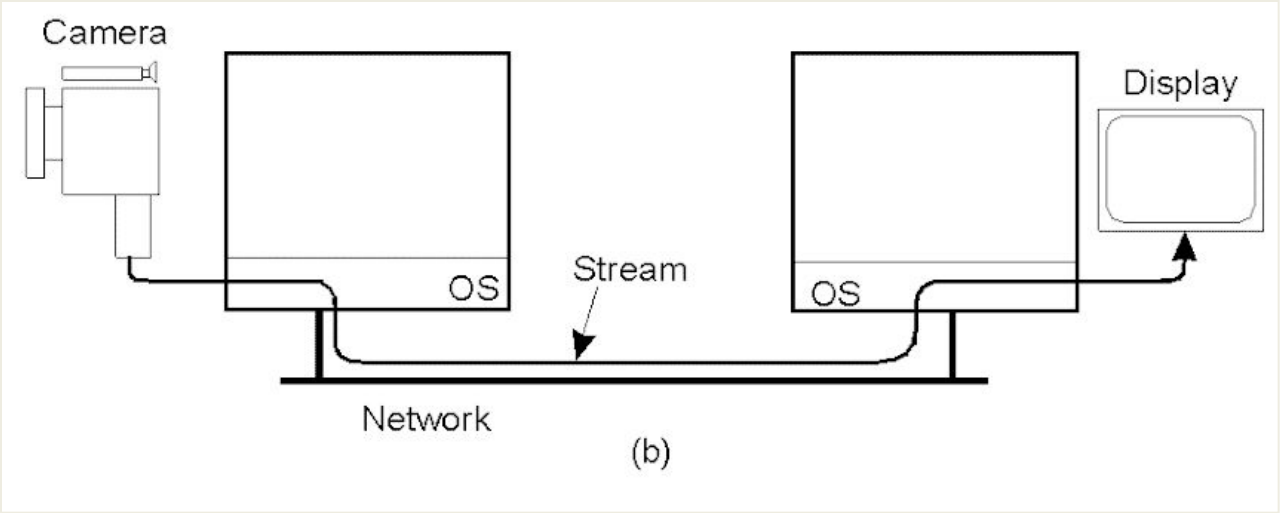
Stream-Oriented Communication

Figure. 2-35

(a) Setting up a stream between two processes across a network

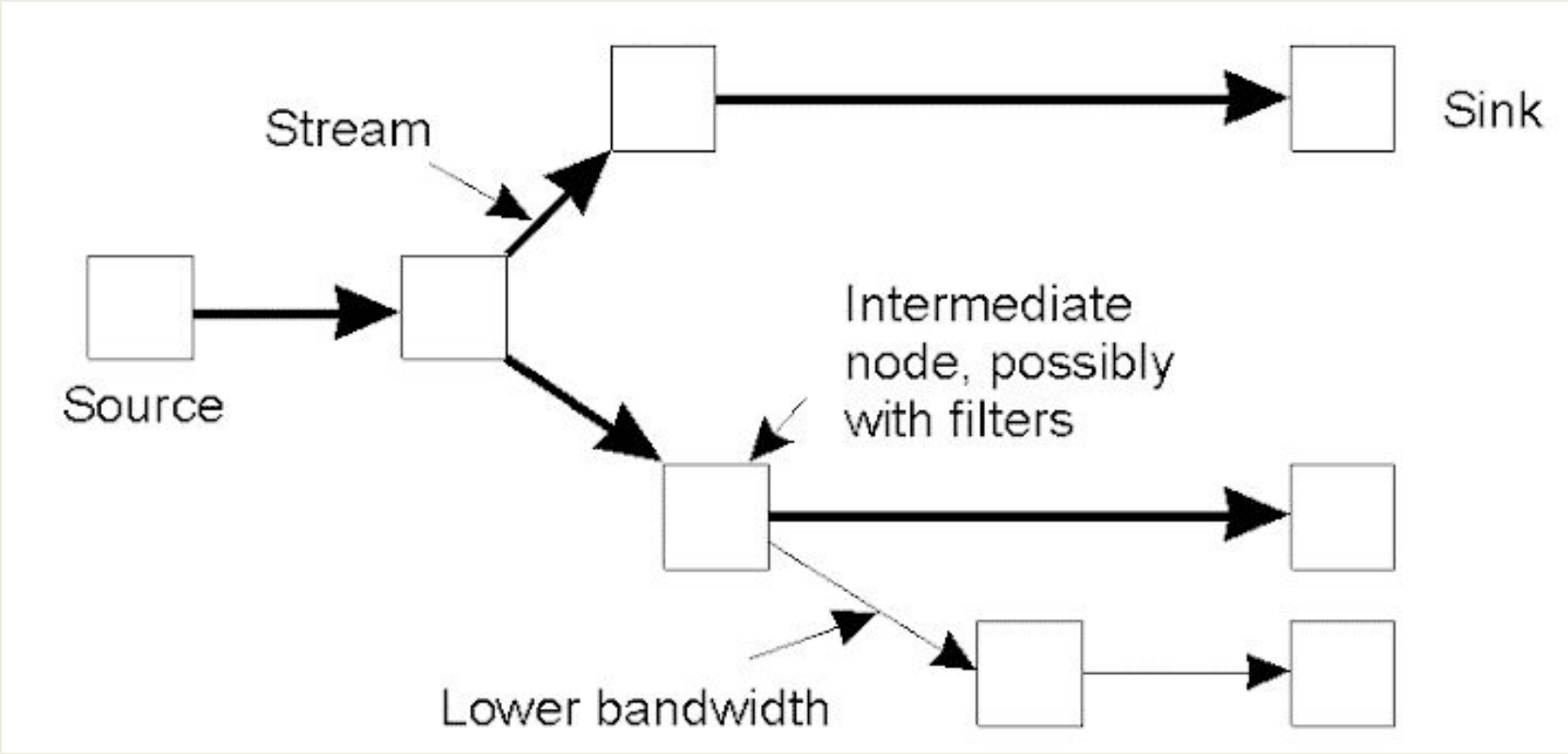
(b) Setting up a stream directly between two devices





Stream-Oriented Communication

Figure. 2-36
An example of multicasting a stream to several receivers



Streams and Quality of Service

- **Timing** (and other nonfunctional) requirements are generally expressed as **Quality of Service (QoS)** requirements.
- These requirements describe what is needed from the underlying distributed system and network to ensure that, for example, the temporal relationships in a stream can be preserved.
- QoS for continuous data streams mainly concerns **timeliness, volume, and reliability.**
- From an application's perspective, in many cases it boils down to specifying a few important properties (Halsall, 2001):
  - 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

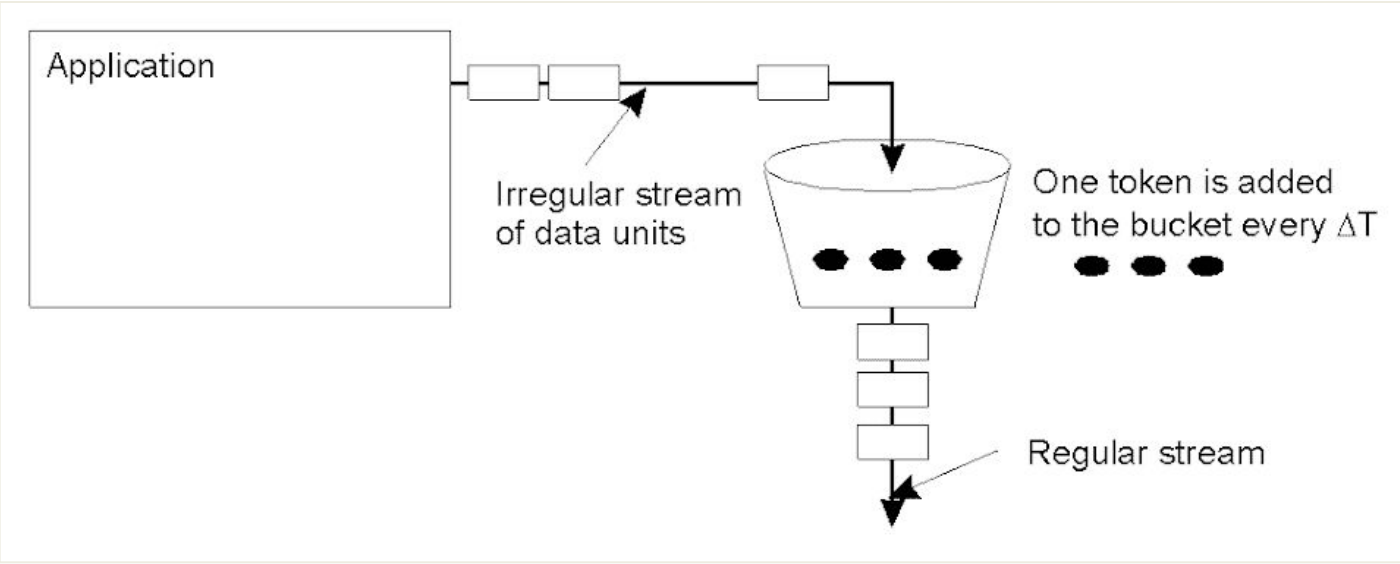
- **Specifying QoS**
  - Flow specification
  - To provide a precise factors(bandwith, transmission rates and delay, etc.)
  - Example of flow specification developed by Partridge

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

Streams and Quality of Service

- Specifying QoS
  - Token bucket algorithms
    - Tokens are generated at a constant rate
    - Token is fixed number of bytes that an application is allowed to pass to the network

Figure. 2-38
The Principle of a token bucket algorithm



Streams and Quality of Service

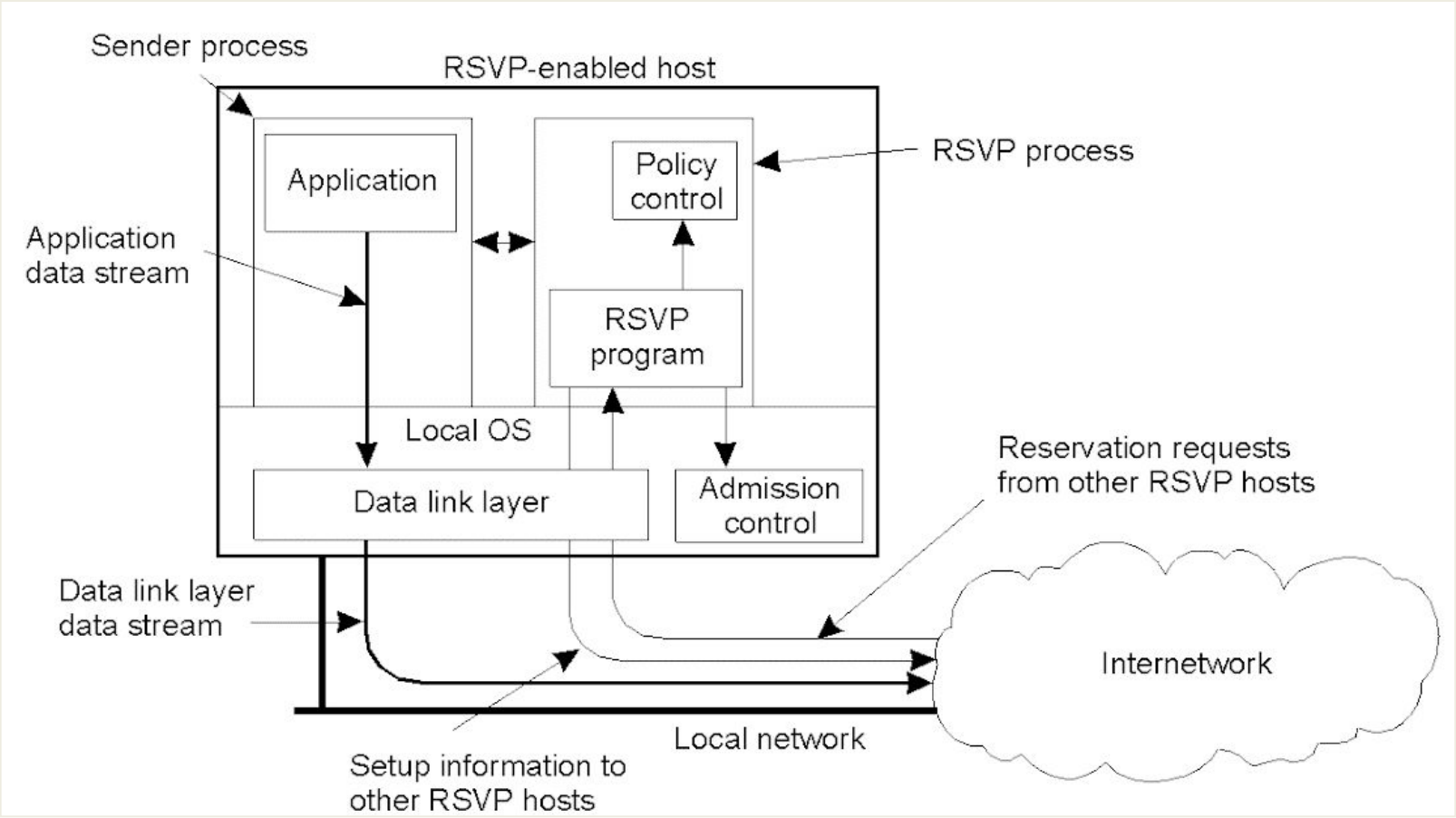
- **Setting up a stream**
  - Resource reSerVation Protocol(RSVP)
    - Transport-level control protocol for enabling resource reservation in network router
    - Used to provide QoS for continuous data streams by reserving resources (bandwidth, delay, jitter and so on)
    - Issue: How to translate QoS parameters to resource usage?
      - **Two ways to translate**

        1. RSVP translates QoS parameters into data link layer parameters

        2. Data link layer provides its own set of parameters (as in ATM)

Streams and Quality of Service

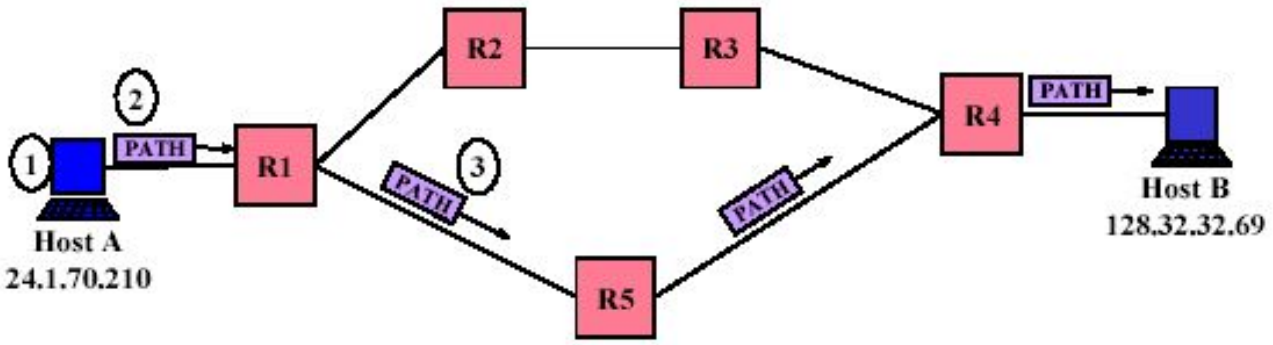
The basic organization of RSVP for resource reservation in a distributed system



Streams and Quality of Service

RSVP Reservation

- Senders advertise using PATH message
- Receivers reserve using RESV message
  - Travels upstream in reverse direction of Path message



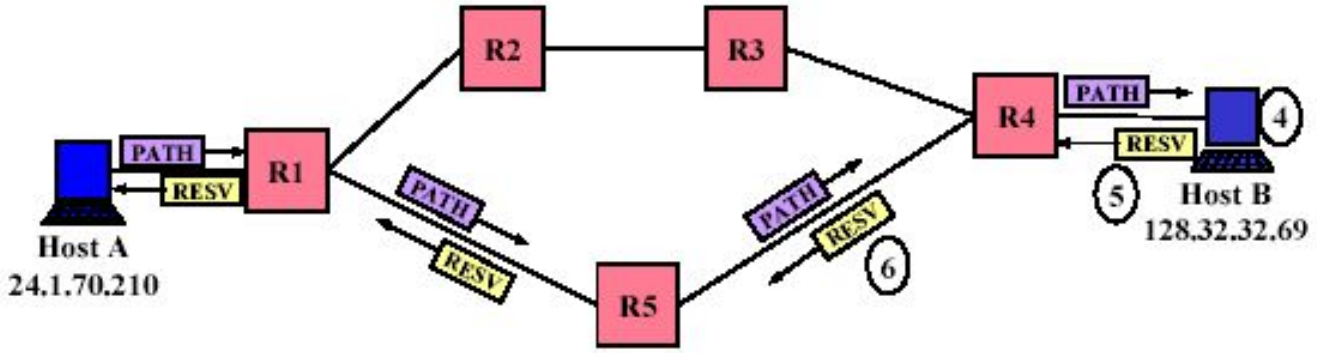
1. An application on Host A creates a session, 128.32.32.69/4078, by communicating with the RSVP daemon on Host A.

2. The Host A RSVP daemon generates a PATH message that is sent to the next hop RSVP router, R1, in the direction of the session address, 128.32.32.69.

3. The PATH message follows the next hop path through R5 and R4 until it gets to Host B. Each router on the path creates soft session state with the reservation parameters.

Streams and Quality of Service

RSVP UDP Reservation



4. An application on Host B communicates with the local RSVP daemon and asks for a reservation in session 128.32.32.69/4078. The daemon checks for and finds existing session state.

5. The Host B RSVP daemon generates a RESV message that is sent to the next hop RSVP router, R4, in the direction of the source address, 24.1.70.210.

6. The RESV message continues to follow the next hop path through R5 and R1 until it gets to Host A. Each router on the path makes a resource reservation.

Stream Synchronization

- Basic ideas
  - Synchronize transmission of **data units**
  - Synchronization take place when the data stream is made up
    - Stereo audio with CD quality(16 bit samples)
      - Sampling rate 44.1 KHz -> synchronize 22.6 micro sec
  - Synchronization between audio stream and video stream for lip sync.
    - NTSC 30Hz(a frame every 33.33ms), CD Quality sound
      - Synchronized every 1470 sound samples

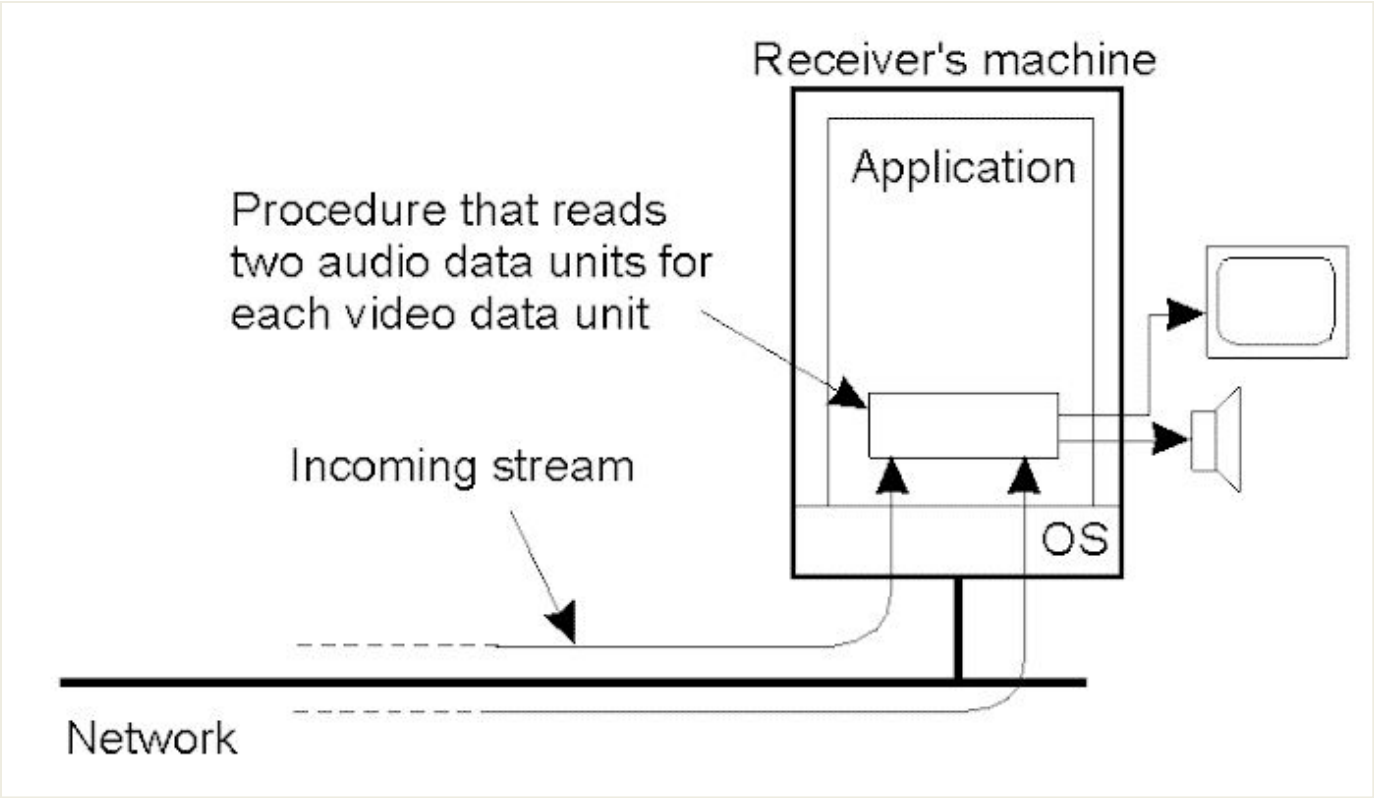
Stream Synchronization

- An important issue in multimedia systems is that different streams, possibly in the form of a complex stream, are mutually synchronized.
- Synchronization of streams deals with maintaining temporal relations between streams. Two types of synchronization occur.
- The simplest form of synchronization is that between a **discrete data stream and a continuous data stream.**
- Consider, for example, a slide show on the Web that has been enhanced with audio.
- Each slide is transferred from the server to the client in the form of a discrete data stream.
- At the same time, the client should play out a specific (part of an) audio stream that matches the current slide that is also fetched from the server. In this case, the audio stream is to be 'synchronized with the presentation of slides.

Stream Synchronization

- Synchronization Mechanisms

The principle of explicit synchronization on the level data units



Stream Synchronization

- Synchronization Mechanisms

The principle of synchronization as supported by high-level interfaces