Communication

Chapter 3

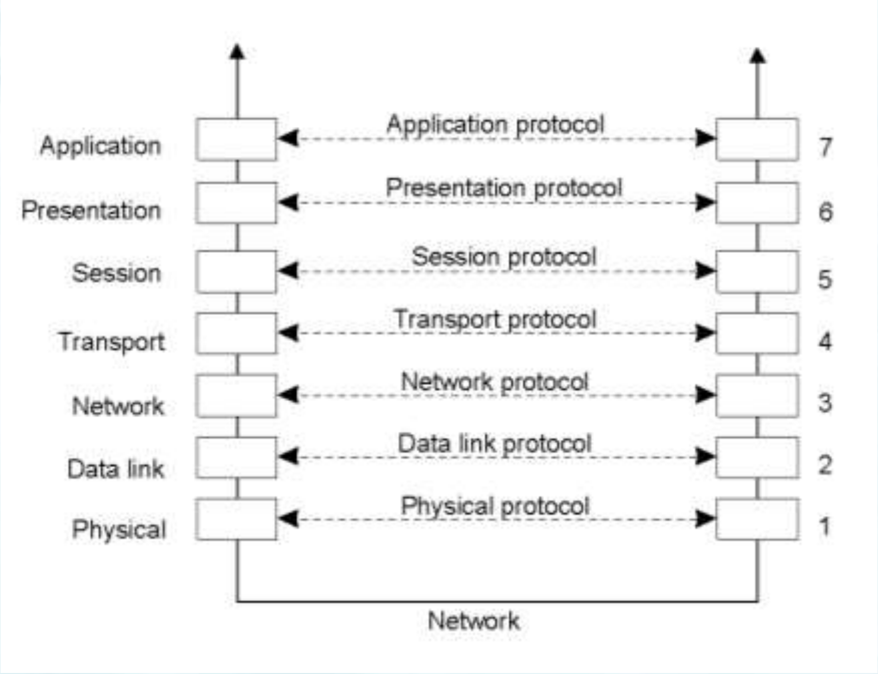
Agenda

- 3.1 Layered protocols
- 3.2 Remote Procedure
- 3.3 Remote Object Invocation
- 3.4 Message-Oriented Communication
- 3.5 Stream-Oriented Communication

3.1 Layered Protocols

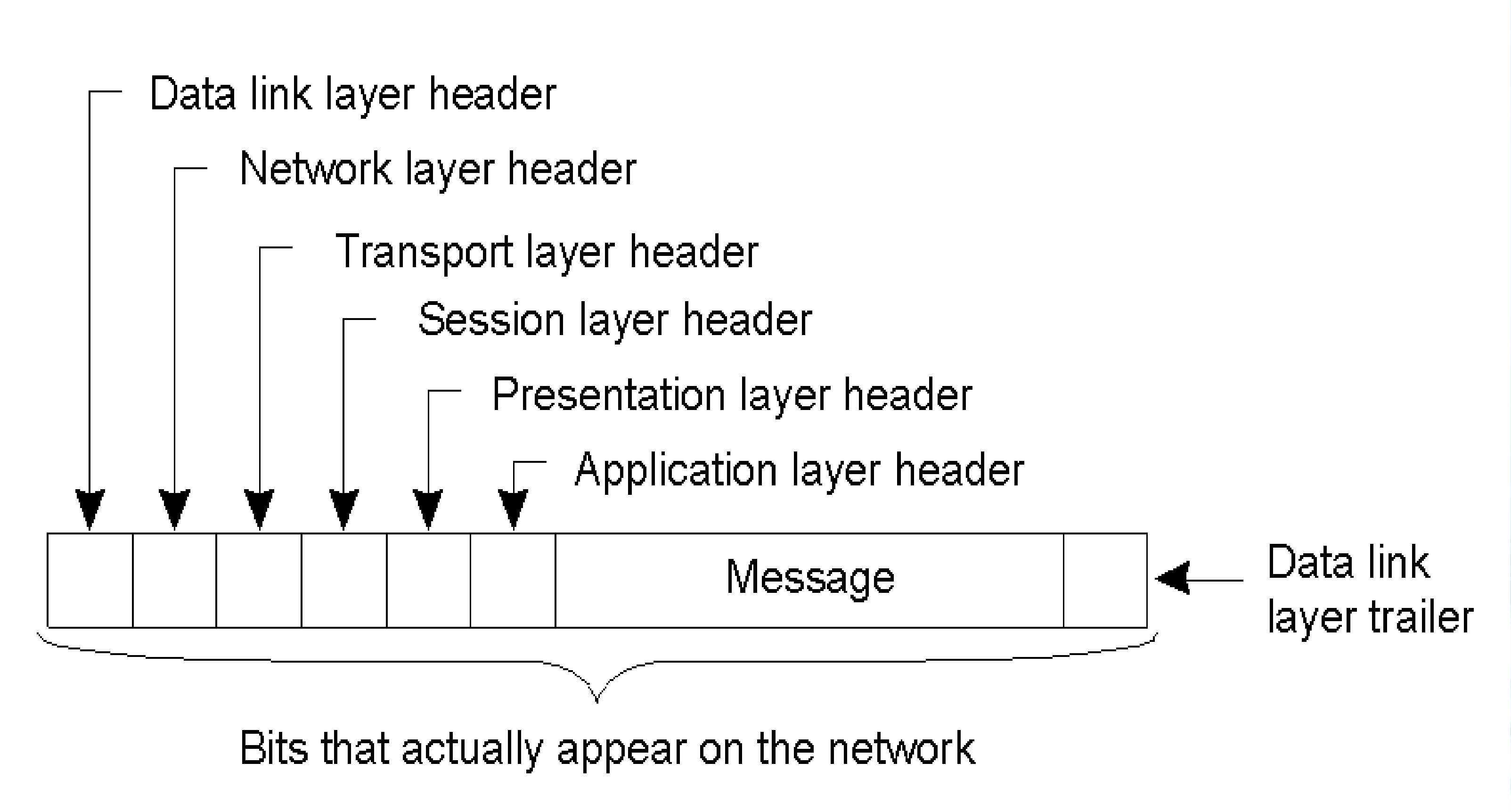
1 - Layered Protocols

- **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.

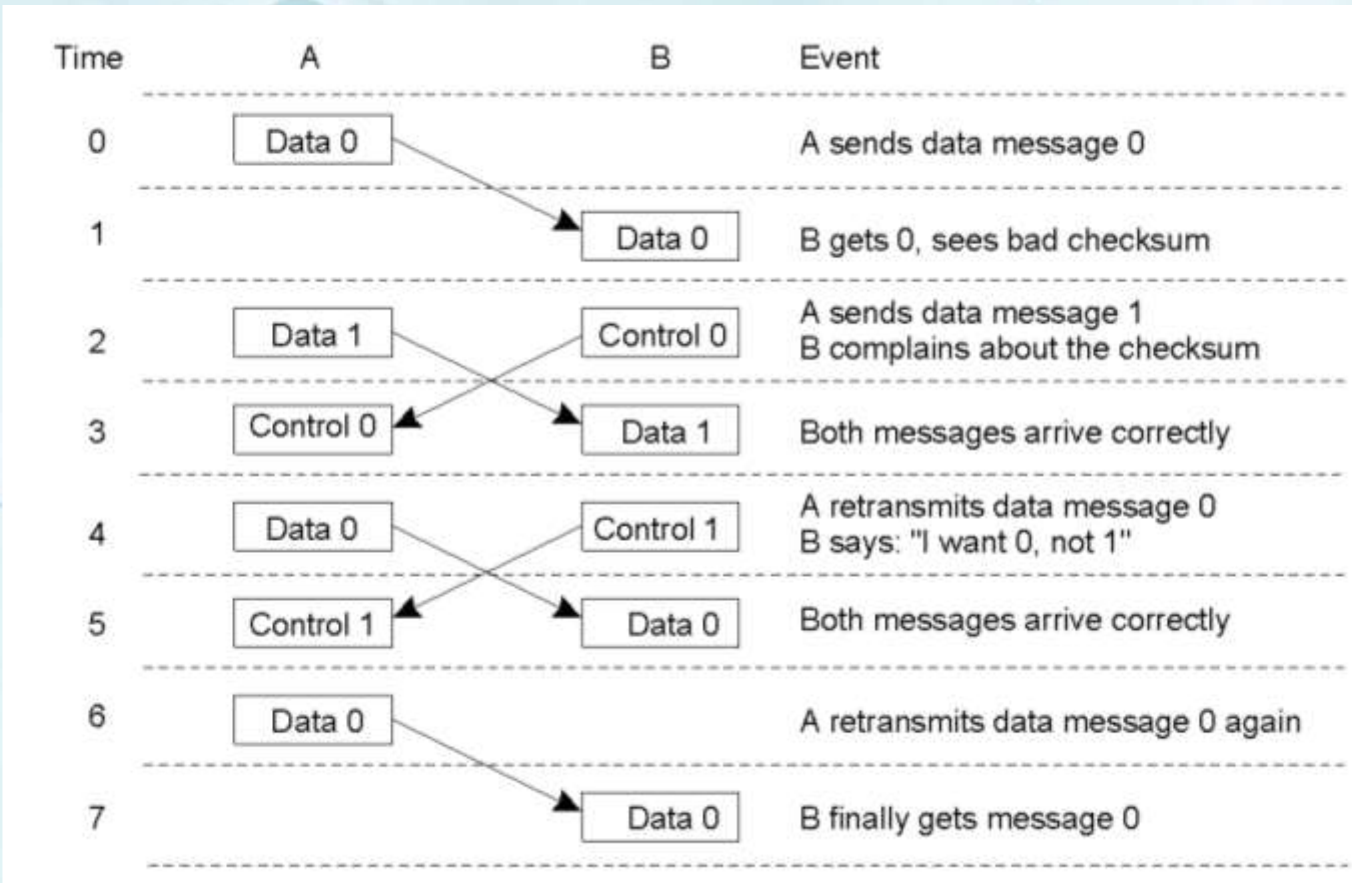
OSI Protocol Layers

- **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.



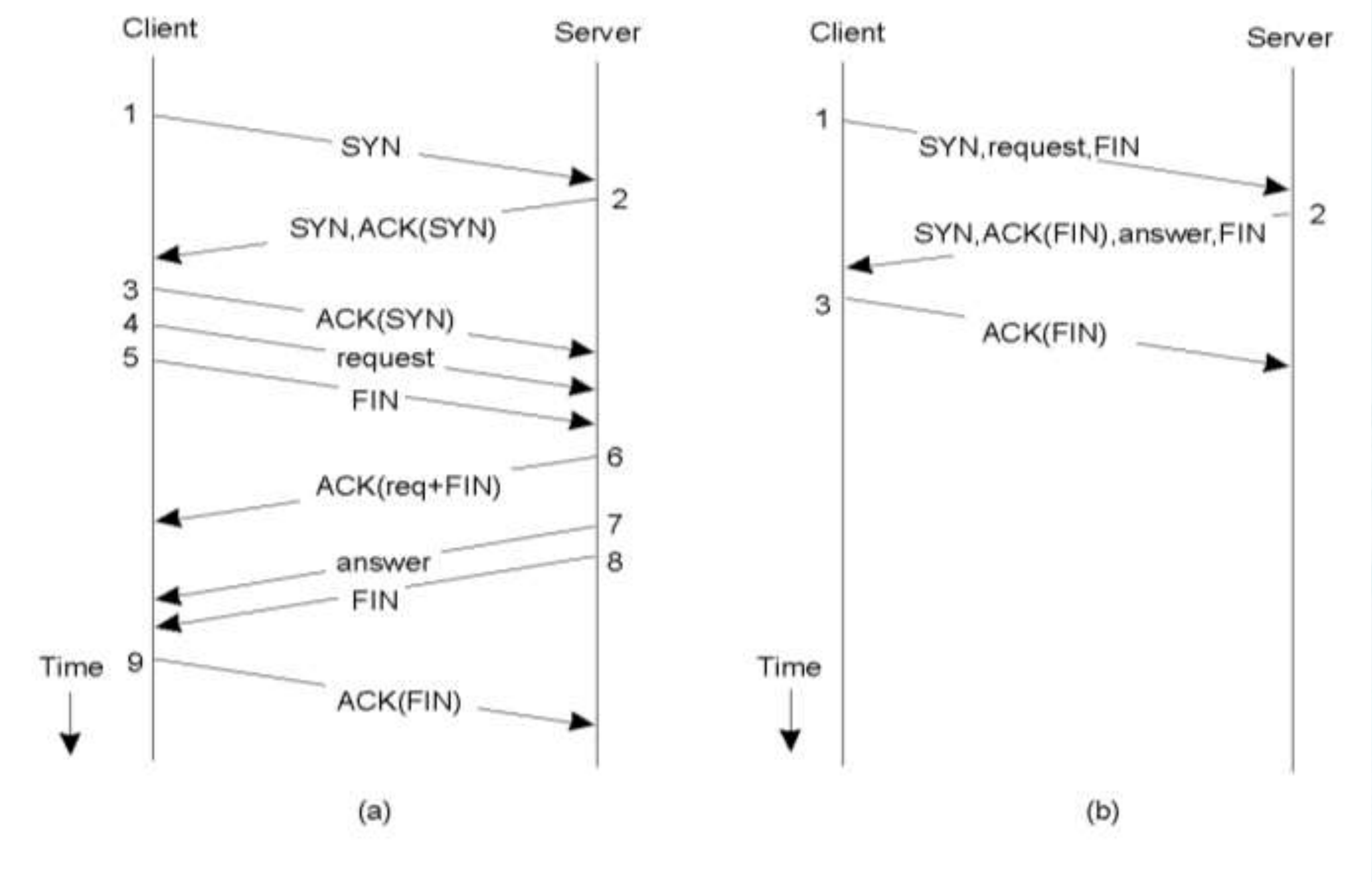
OSI Protocol Layers (con.t)

- **Network layer:**
  - Performs multi-hop routing across multiple networks
  - Implemented in end systems and routers

- **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



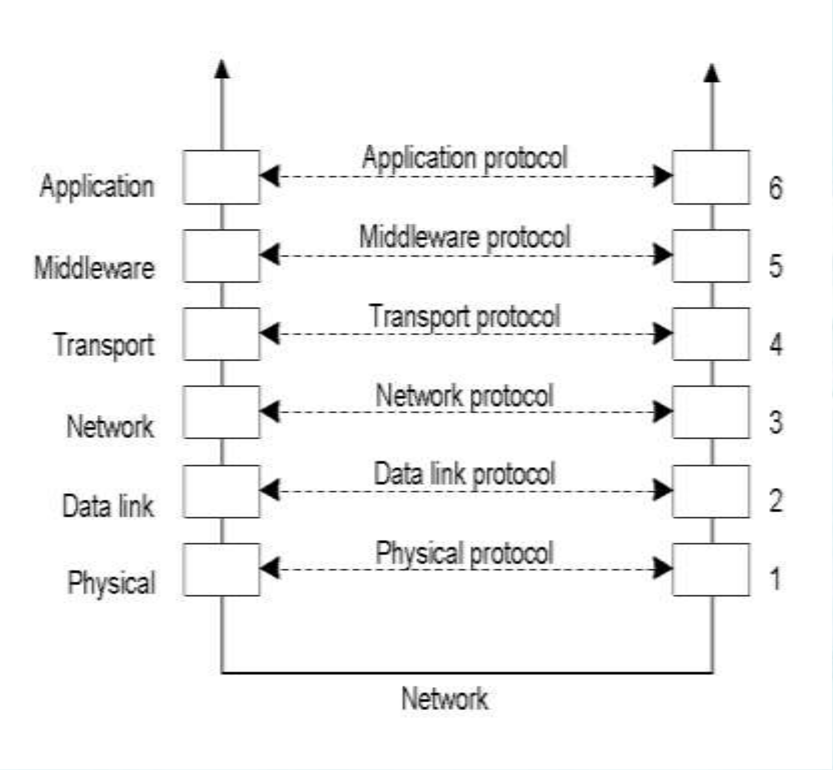
(a) Normal operation of TCP. (b) Transactional TCP.

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…

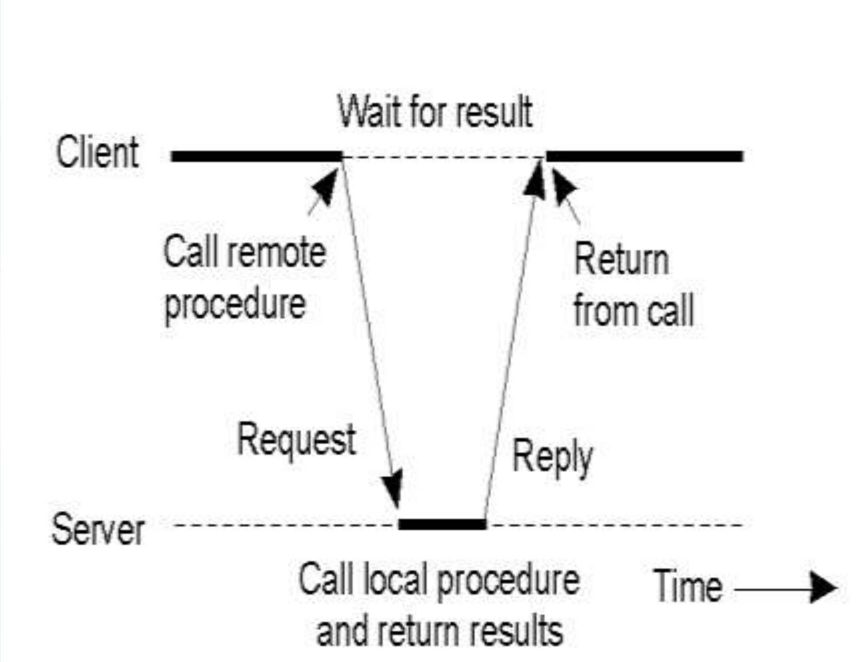


An adapted reference model for networked communication.

3.2 Remote Procedure Call

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).

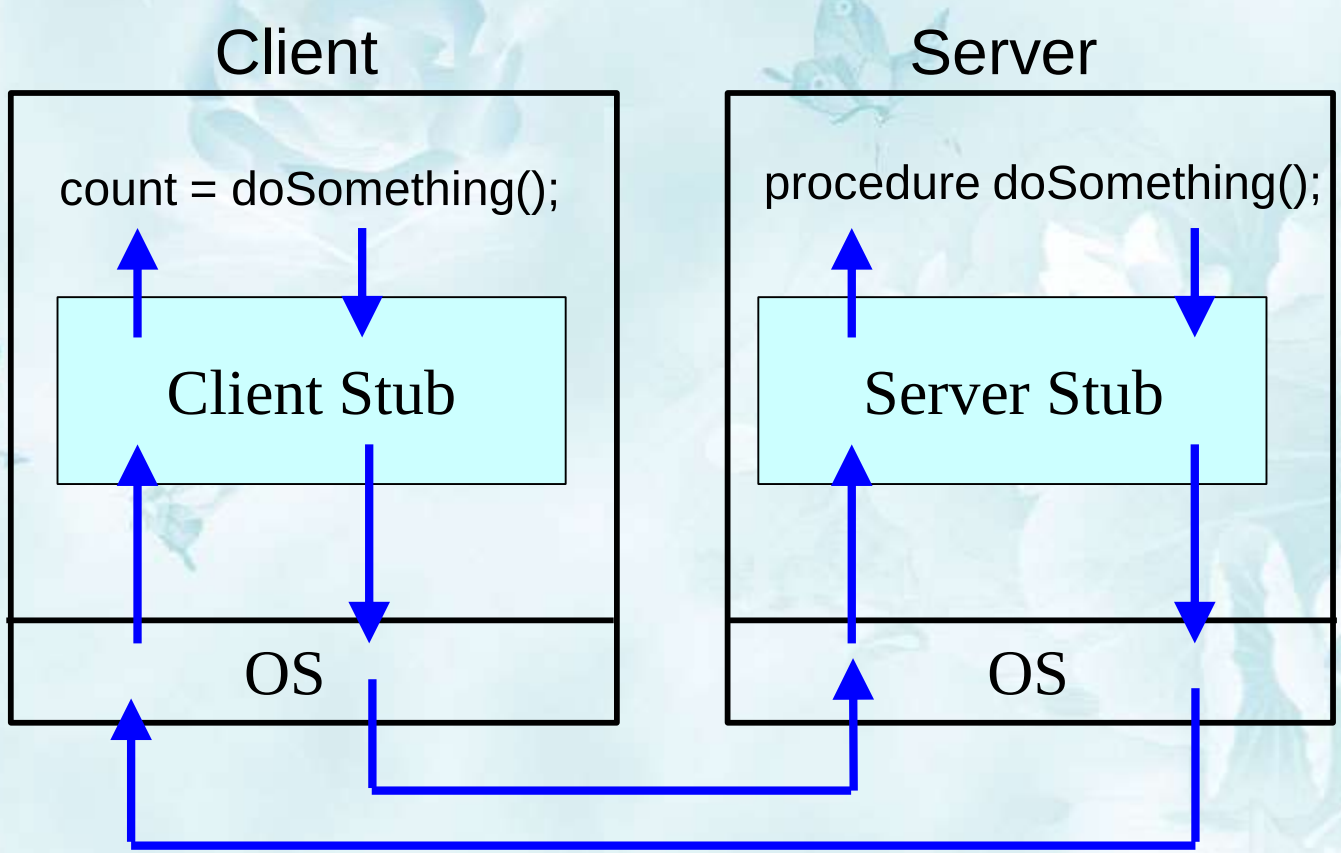


Example:

read(fd, buf, nbytes)

Client and Server Stubs

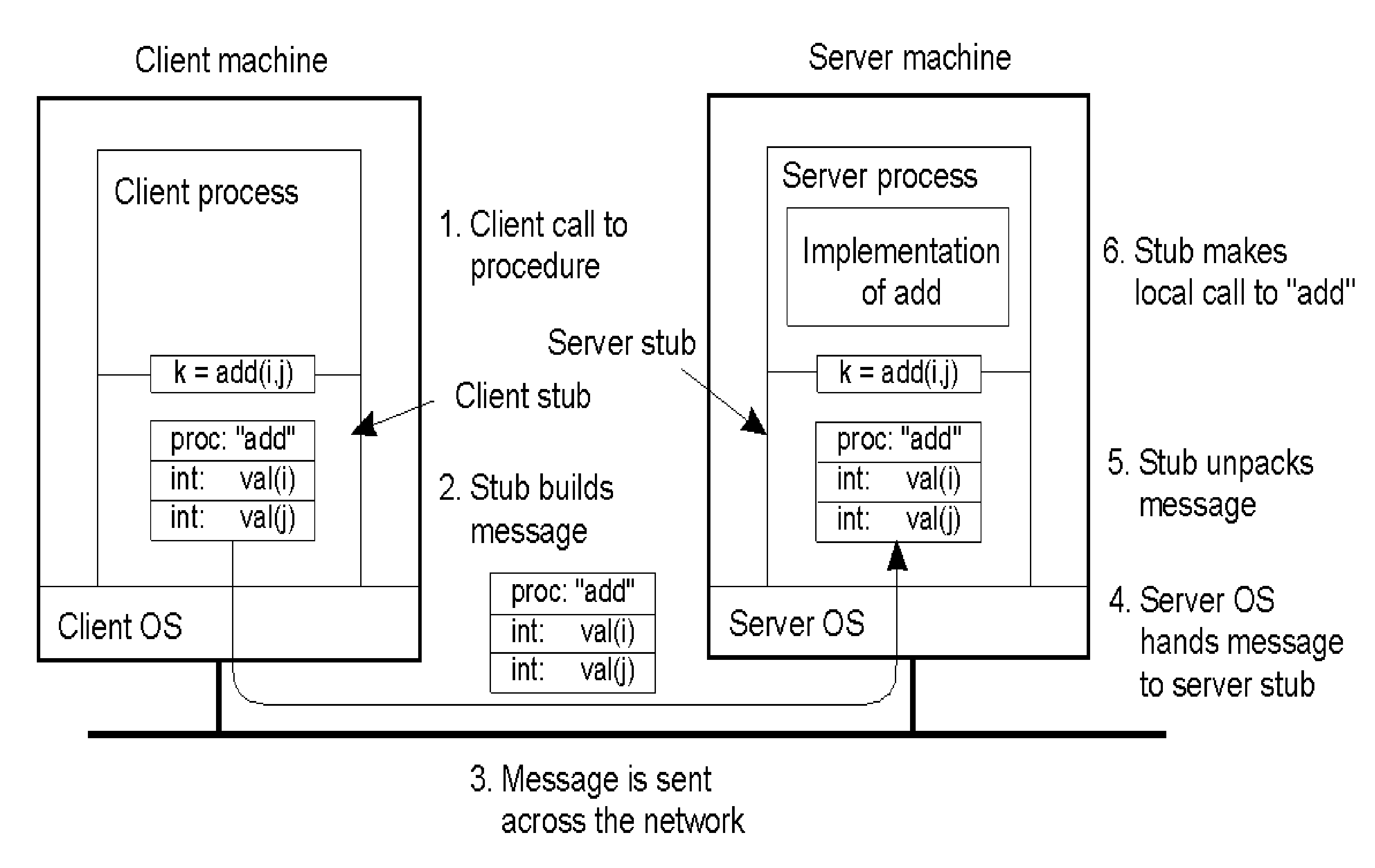
Definition: Are additional functions which are added to the main functions in order to support for RPC



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

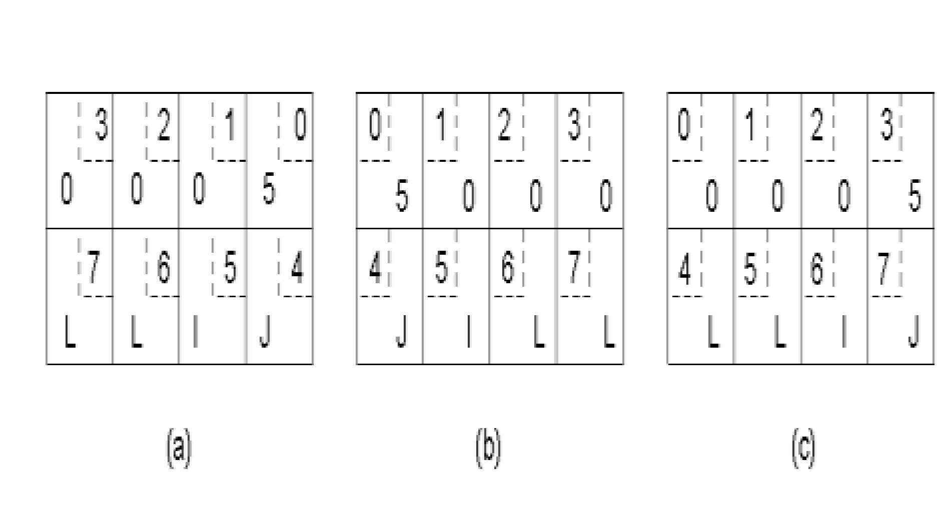
Passing Value Parameters (1)



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) 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

Parameter Specification

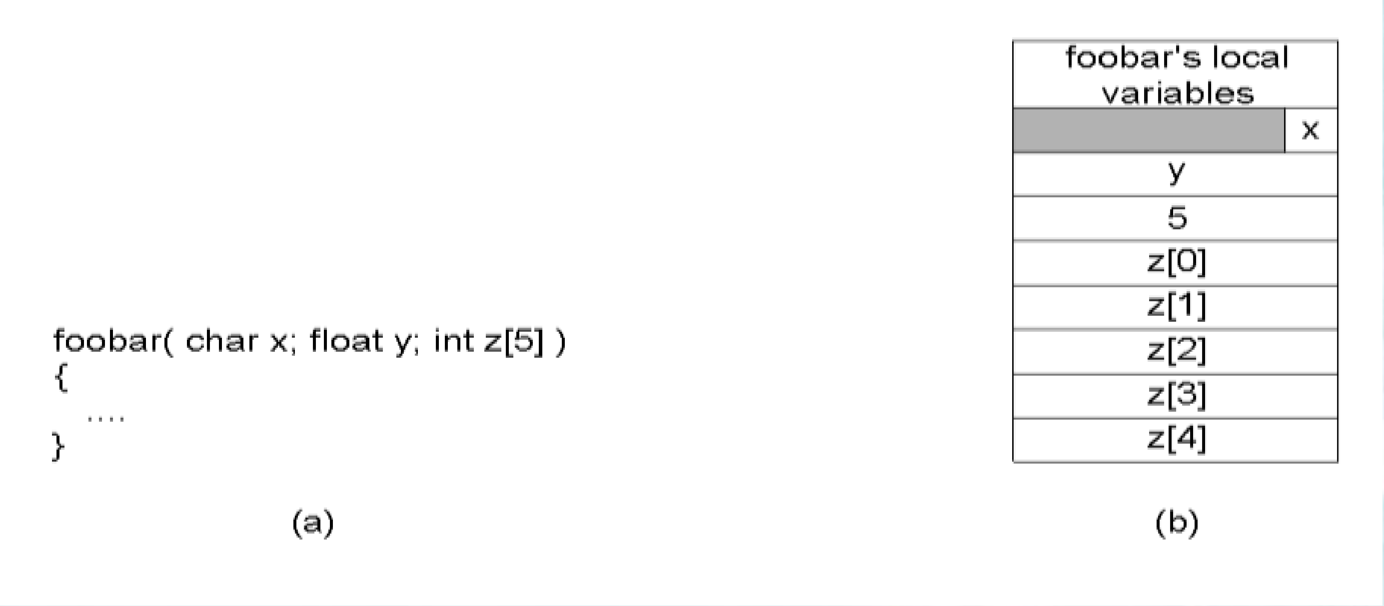
- Caller and callee **agree on the format** of message they exchange

  Ex: word = 4 bytes

  float = 1 word

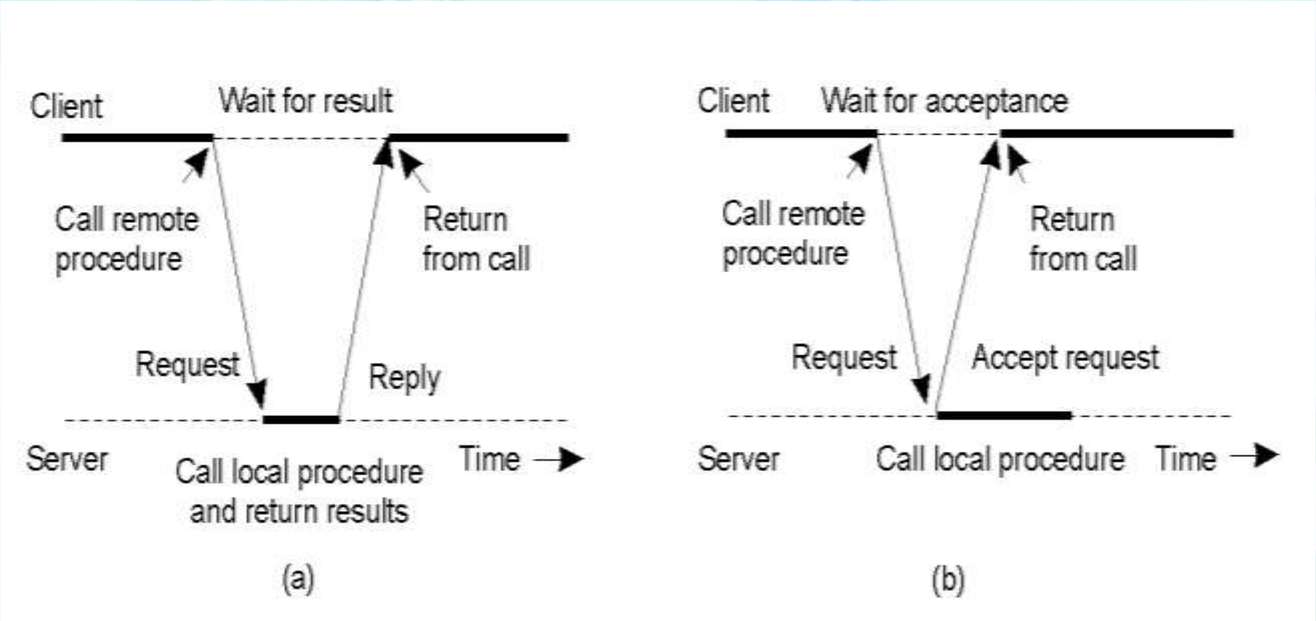
  character is the rightmost byte of word

=> the client stub must use this format and the server stub know that incoming message for foobar has this format



Asynchronous RPC (1)

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

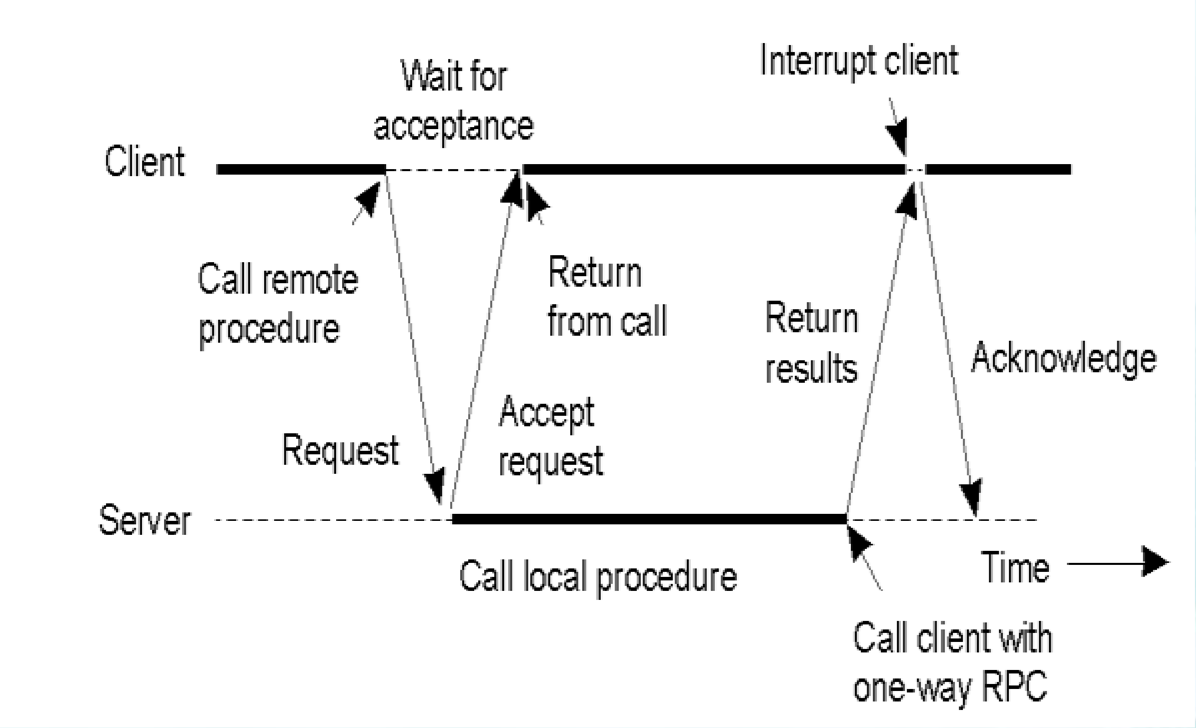


a) The interconnection between client and server in a traditional RPC

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

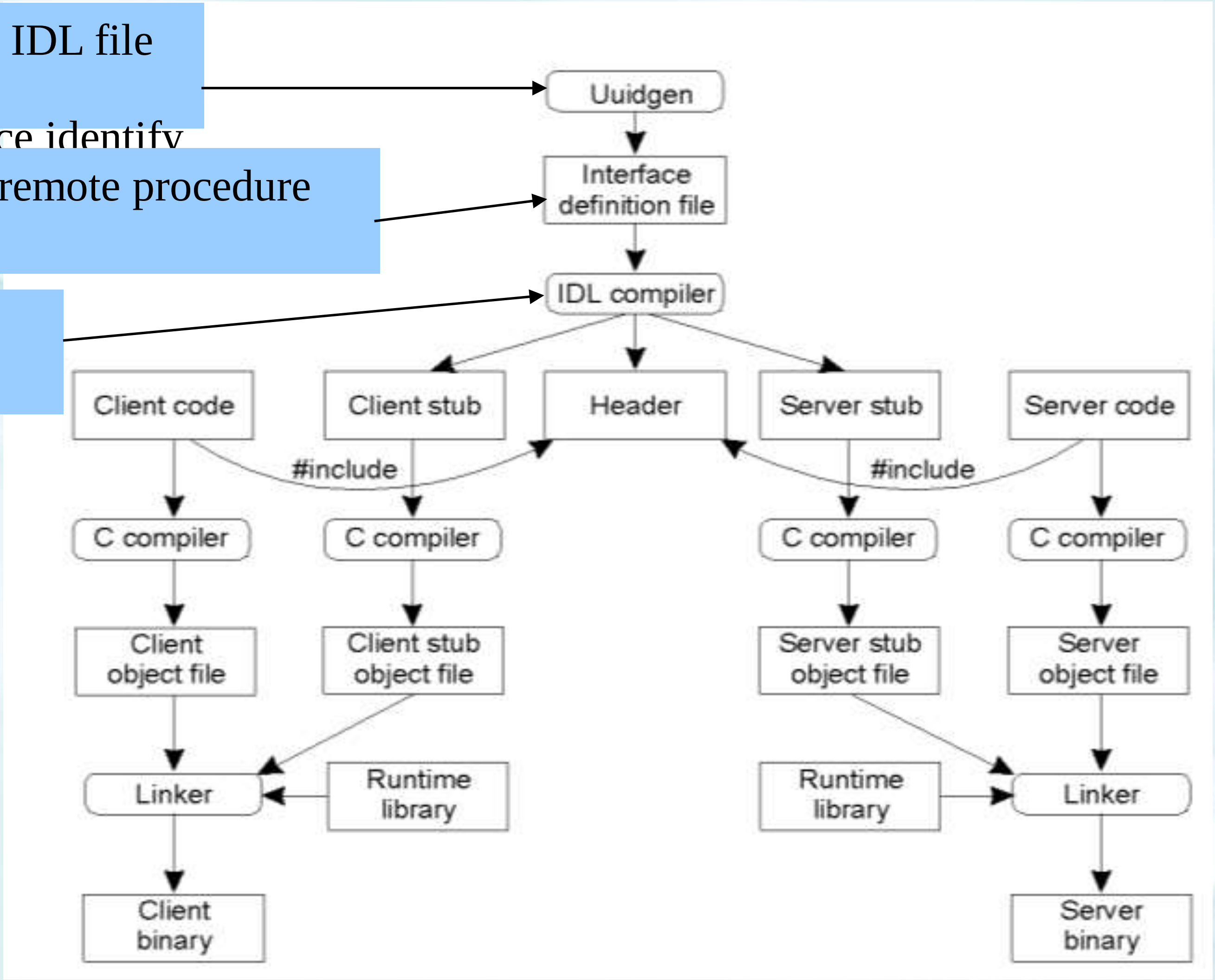
Writing a Client and a Server

Generate a prototype IDL file

containing an interface identify

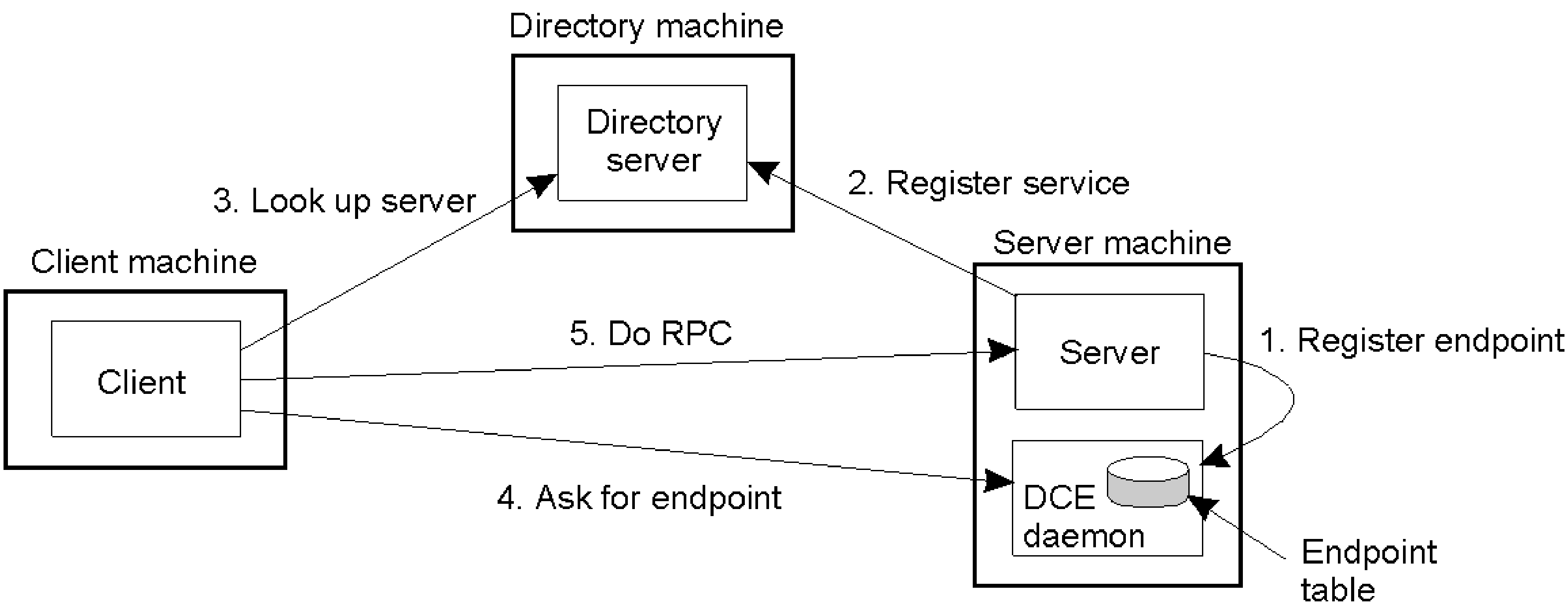
Filling in the names of remote procedure and their parameters

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.

3.3 Remote Object Invocation

Objectives

- RMI vs RPC
- Remote Distributed Objects
- Binding a client to object
- Parameter Passing
- Java RMI
- Example
- Summary

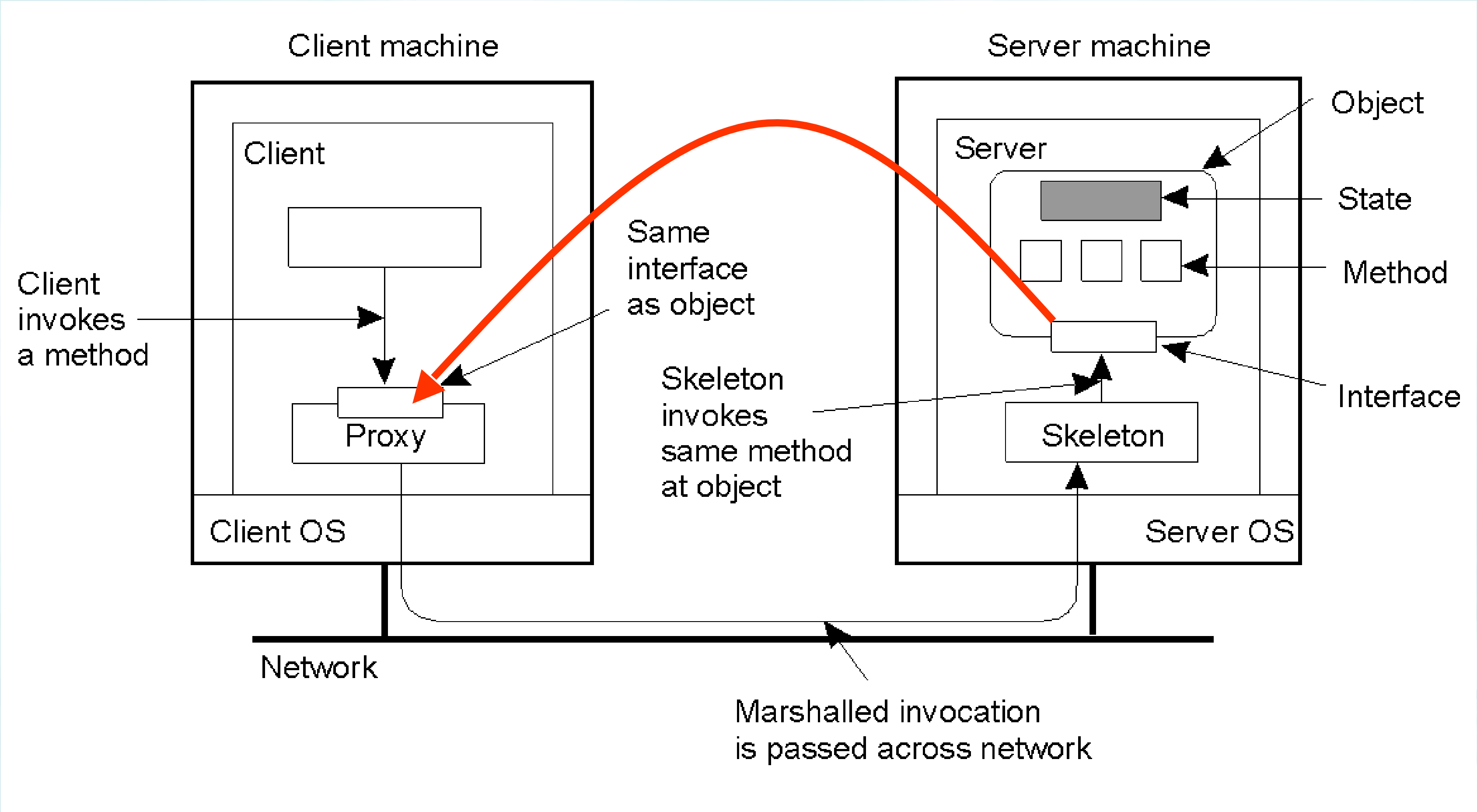
RMI vs RPC

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

Remote Distributed Objects(1)

- Objects separate their actual implementation from their interface
- Distributed object = an object which publishes its interface on other machines
- Remote object = a distributed object whose state is encapsulated (their state is not distributed)

Remote Distributed Objects(2)



- Common organization of a remote object with client-side proxy.

Binding a Client to an Object (1)

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

Binding a Client to an Object (2)

```
Distr_object* obj_ref;        //Declare a systemwide object reference
obj_ref = …;                  // Initialize the reference to a distributed object
obj_ref-> do_something();     // Implicitly bind and invoke a method
```

(a)

```
Distr_object objPref;         //Declare a systemwide object reference
Local_object* obj_ptr;        //Declare a pointer to local objects
obj_ref = …;                  //Initialize the reference to a distributed object
obj_ptr = bind(obj_ref);      //Explicitly bind and obtain a pointer to the local proxy
obj_ptr -> do_something();    //Invoke a method on the local proxy
```

(b)

a) An example with implicit binding using only global references

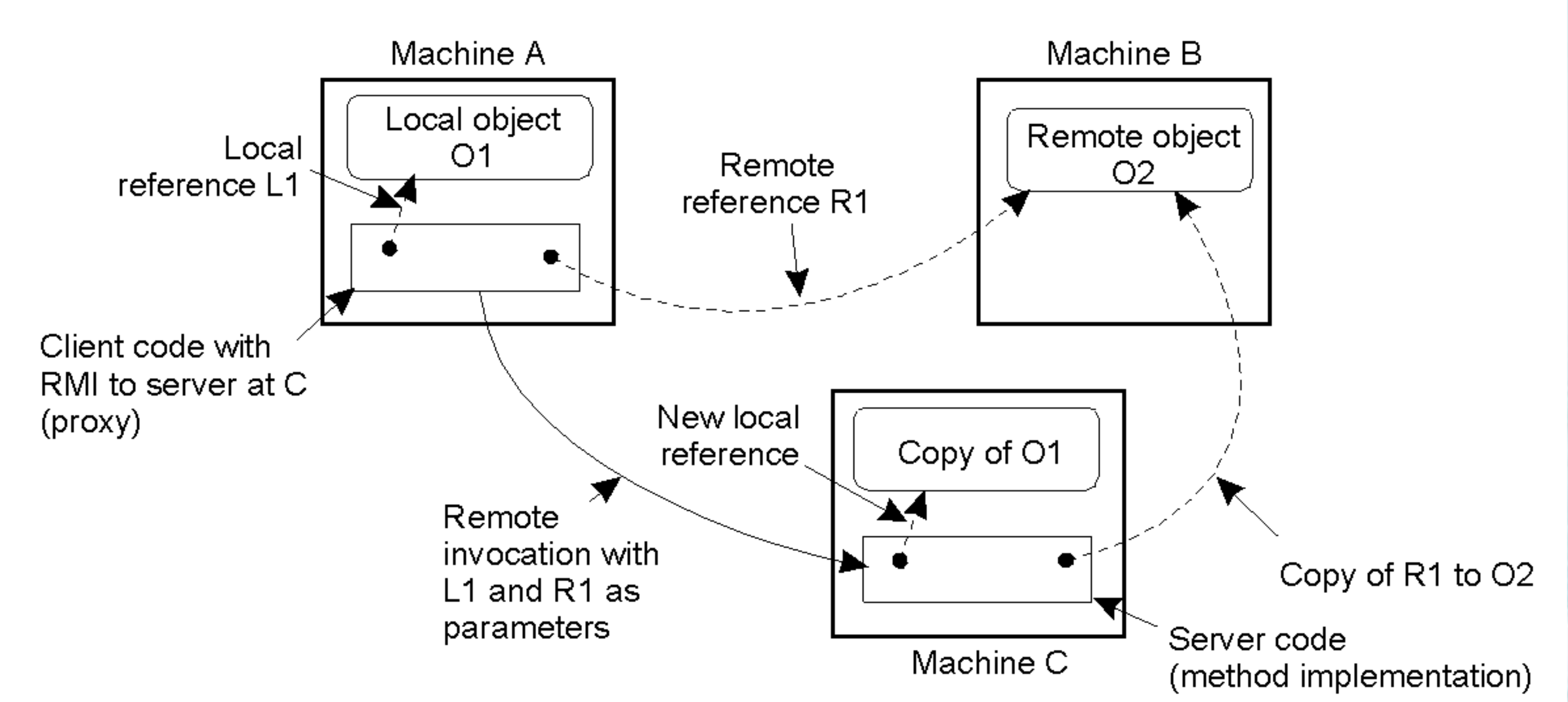
b) An example with explicit binding using global and local references

Parameter Passing (1)

- Pass **remote object** by reference
- Pass **local object** by value
- Local object = an object in the client's address space

Parameter Passing (2)

- **The situation when passing an object by reference or by value.**

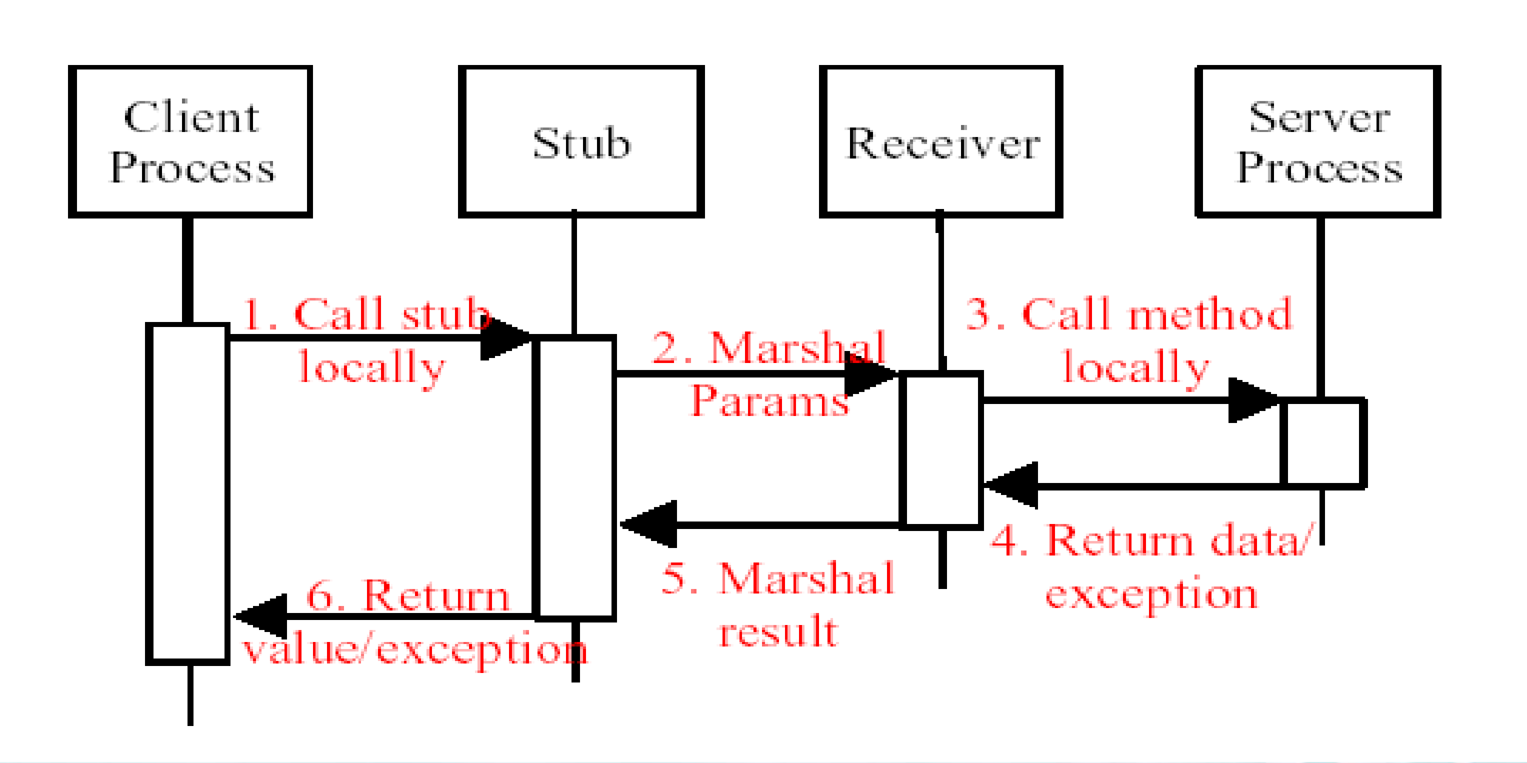


program running on machine A, server program on machine B.
The client has a reference to a local on object O1 that it uses as a parameter when calling the server program on machine C.
-And Client holds a reference to a remote object O2 residing at machine B which is also used as a parameter.
-When calling the server, a copy of O1 is passed to the server on machine C, along with only a copy of the reference to O2.
This figure show the situation when passing an object by reference or value.

Java RMI (1)

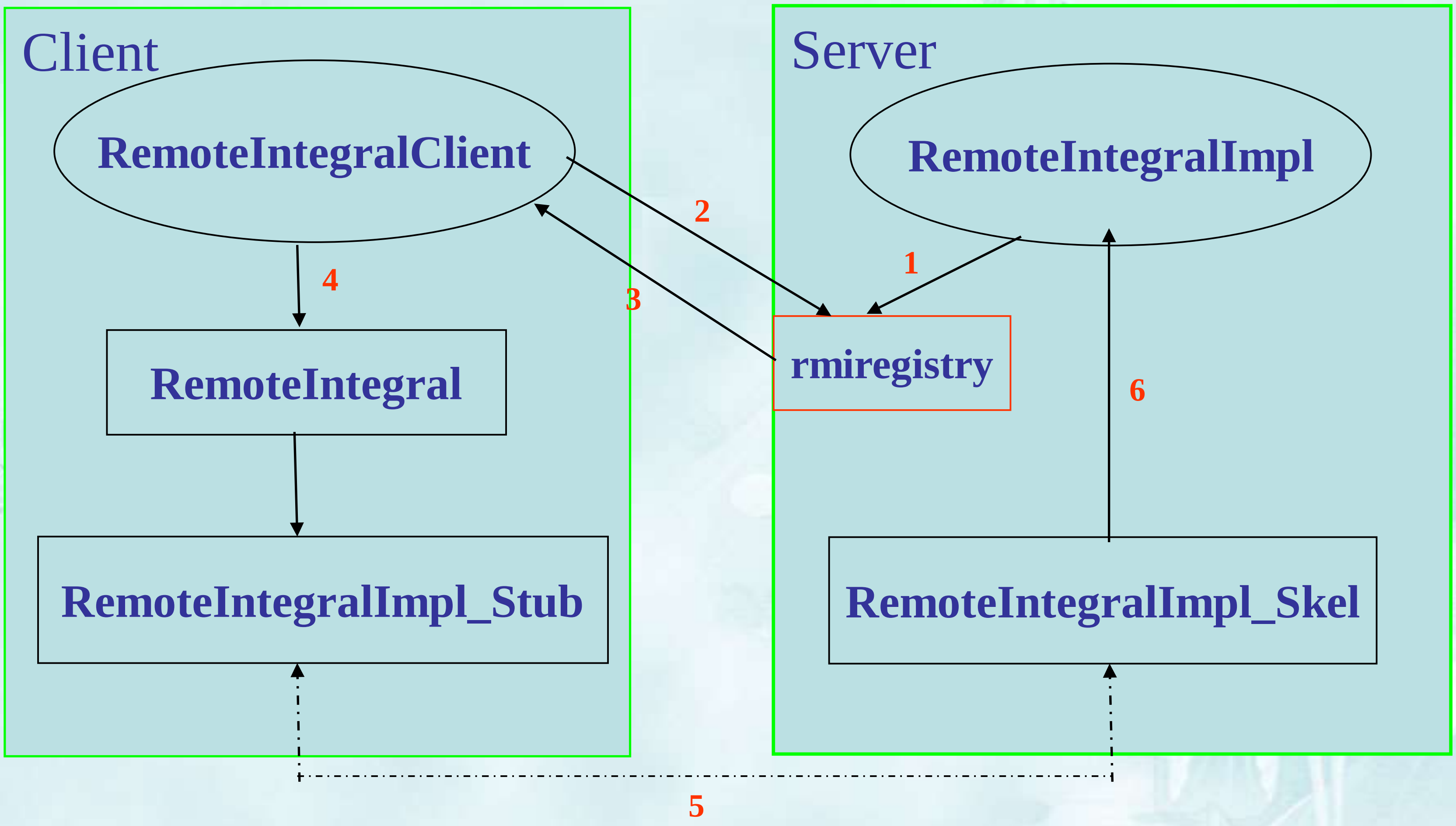
- Distributed objects integrated into the Language
- Goal is to achieve *transparency*!
- Any serializable object type can be a parameter to an RMI
- Local objects are passed by value, remote objects are passed by reference
- Proxy can be used as a reference to a remote object: *Possible to serialize the proxy and send it to another server*

Java RMI (2)



Client Process
Stub
Receiver
Server Process
1. Call stub locally
2. Marshal Params
3. Call method locally
4. Return data/ exception
5. Marshal result
6. Return value/exception

Example



Client
RemoteIntegralClient
RemoteIntegral
RemoteIntegralImpl_Stub
Server
RemoteIntegralImpl
rmiregistry
RemoteIntegralImpl_Skel
1
2
3
4
5
6

Summary

- RMI specific for a remote object
- Remote object offer better transparence
- RMIs allow object references to be passes as parameter
- Local objects are passed by value
- Remote objects are passed by reference

3.4 Message-Oriented Communication

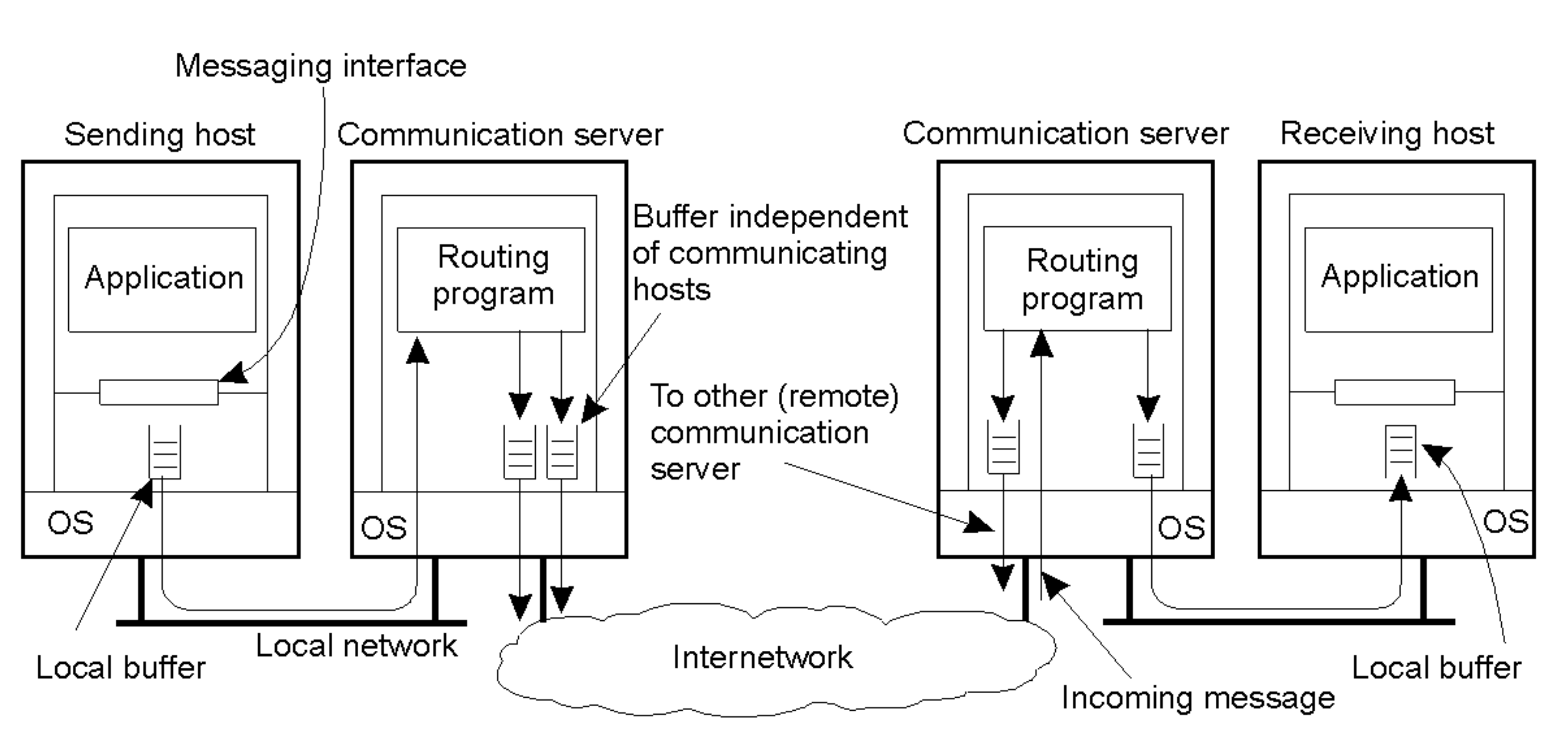
Manhyung Han(smiley@oslab.khu.ac.kr)

Introduction

- 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

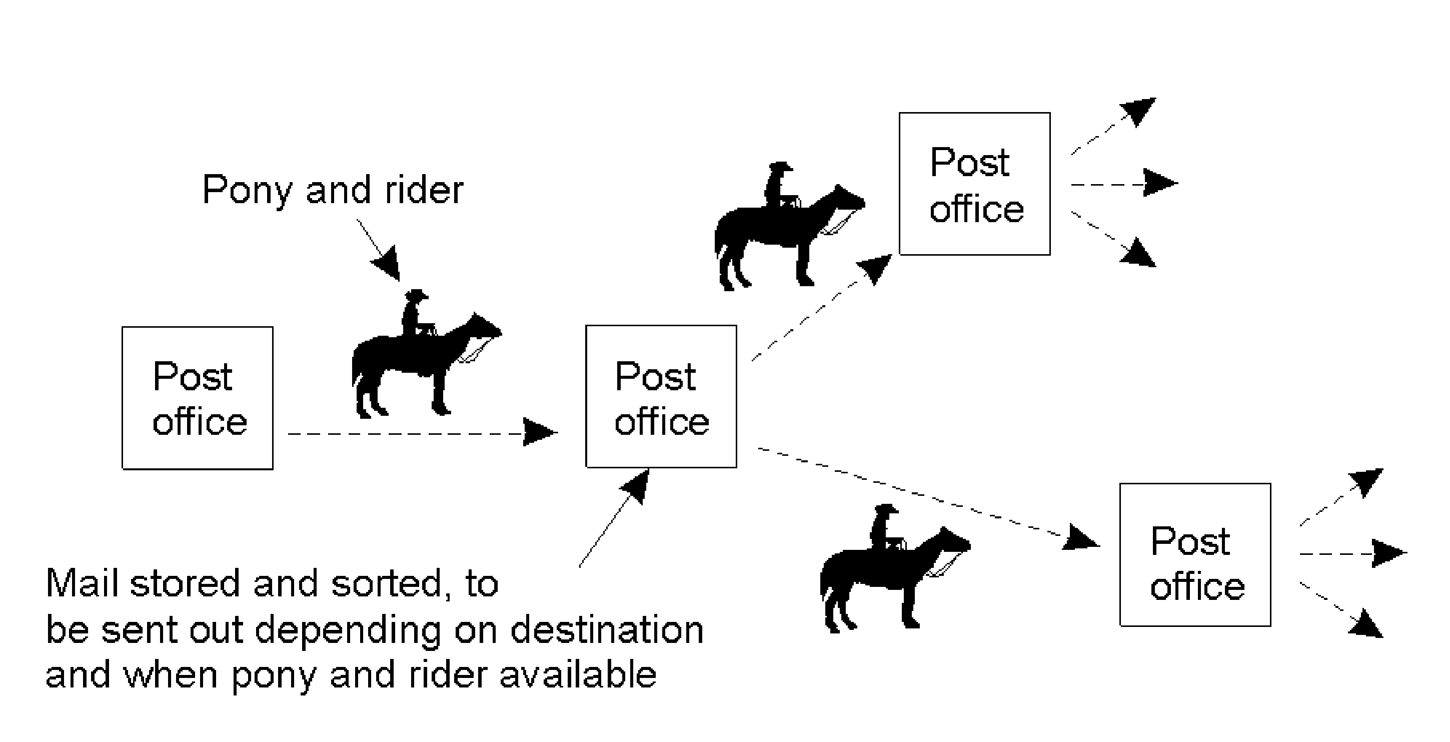
Persistence and Synchronicity in Communication(1)

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



Persistence and Synchronicity in Communication(2)

- Persistent communication
  - An example of persistent communication – Letter back in the days of Pony Express

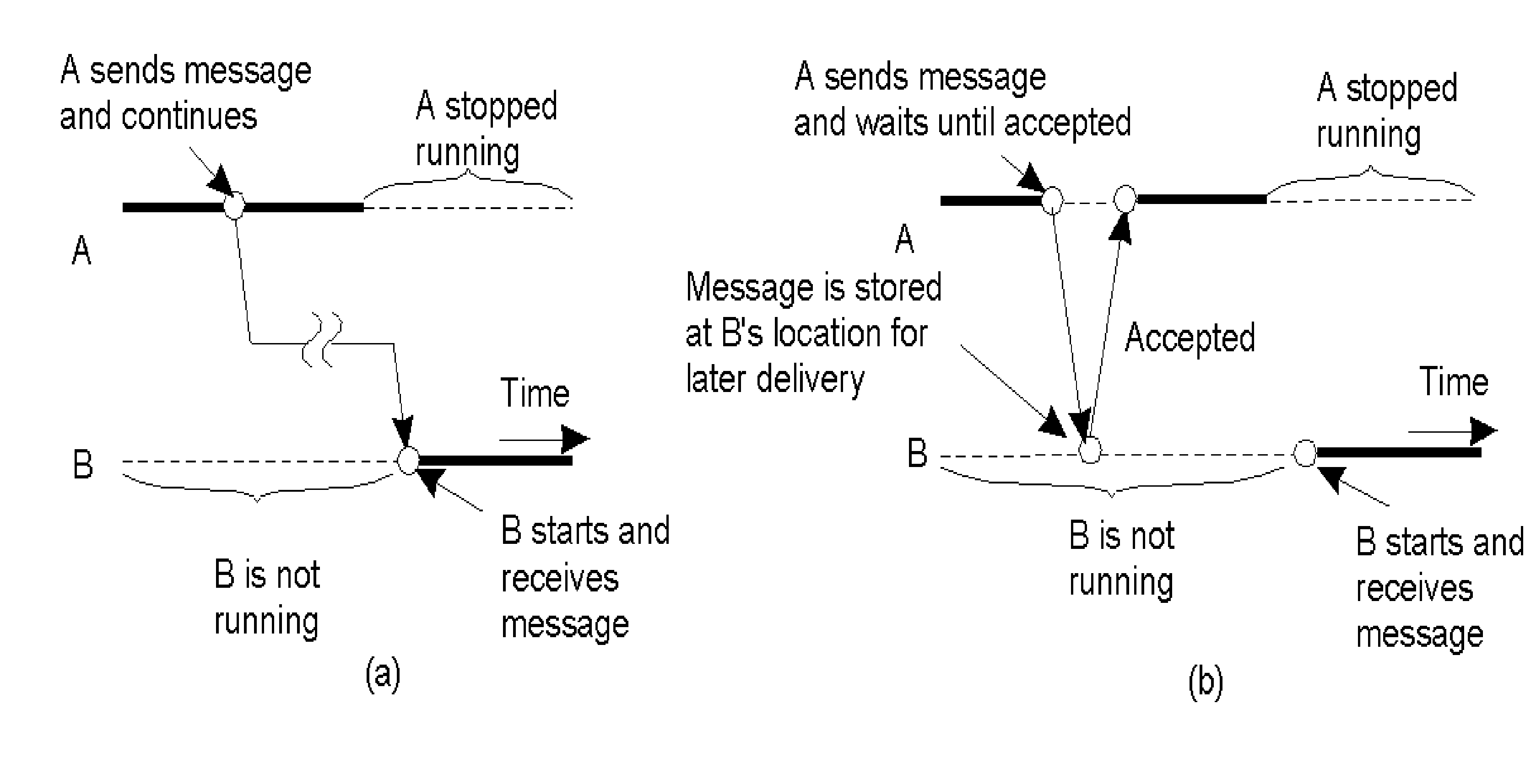


Persistence and Synchronicity in Communication(3)

- Persistent vs. Transient
  - Persistent messages are stored as long as necessary by the communication system (e.g., e-mail)
  - Transient messages are discarded when they cannot be delivered (e.g., TCP/IP)
- Synchronous vs. Asynchronous
  - Asynchronous implies sender proceeds as soon as it sends the message no blocking
  - Synchronous implies sender blocks till the receiving host buffers the message

Persistence and Synchronicity in Communication(4)

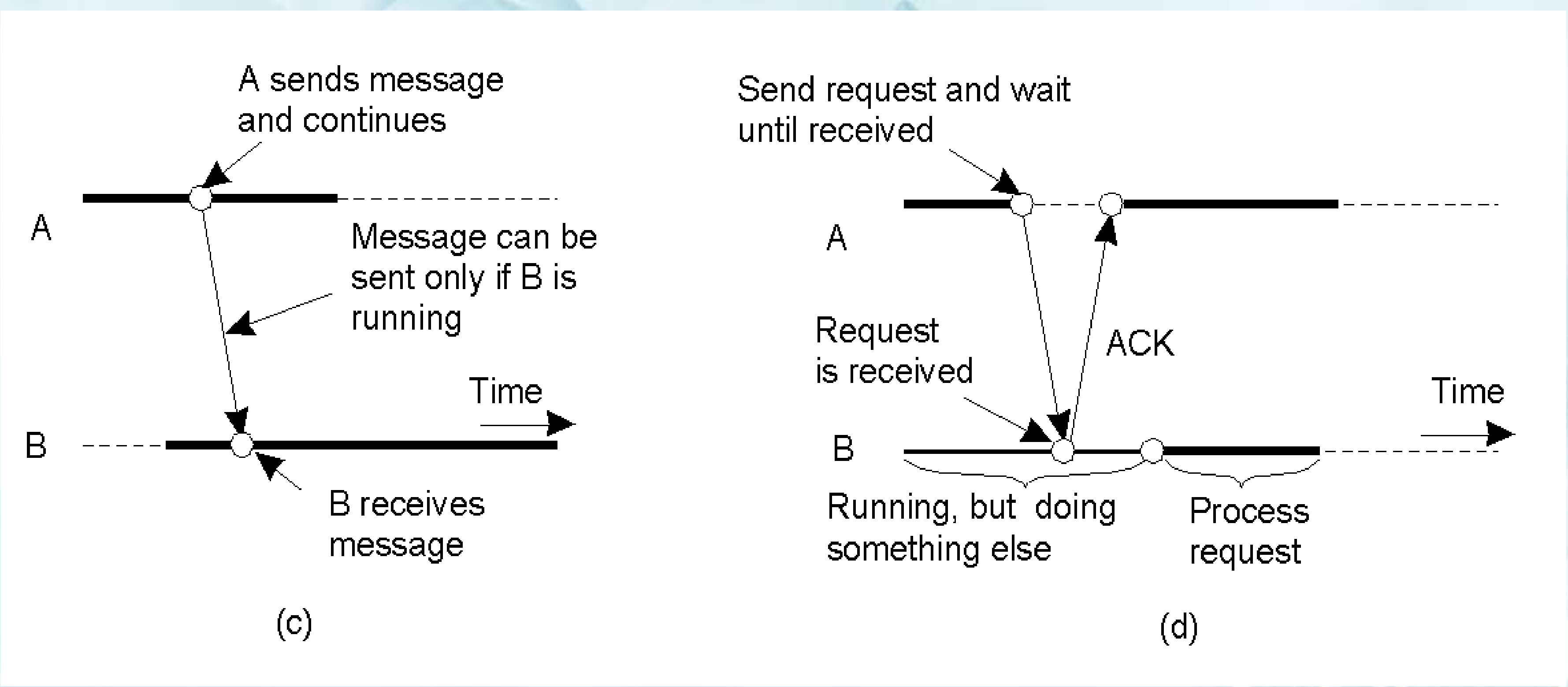
- Persistent asynchronous/synchronous communication



(a) Persistent asynchronous communication / (b) Persistent synchronous communication

Persistence and Synchronicity in Communication(5)

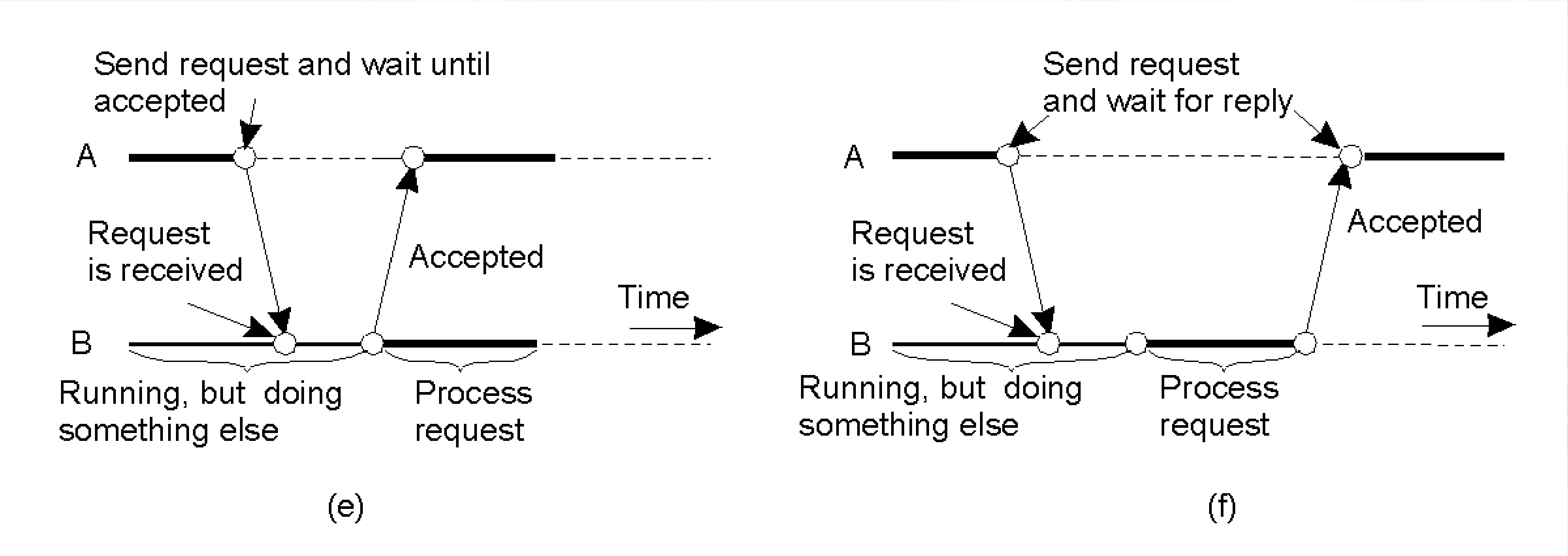
- Transient asynchronous/Receipt-based transient synchronous communication



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

Persistence and Synchronicity in Communication(6)

- Other transient synchronous communications



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

Message-Orient Transient Communication(1)

- 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

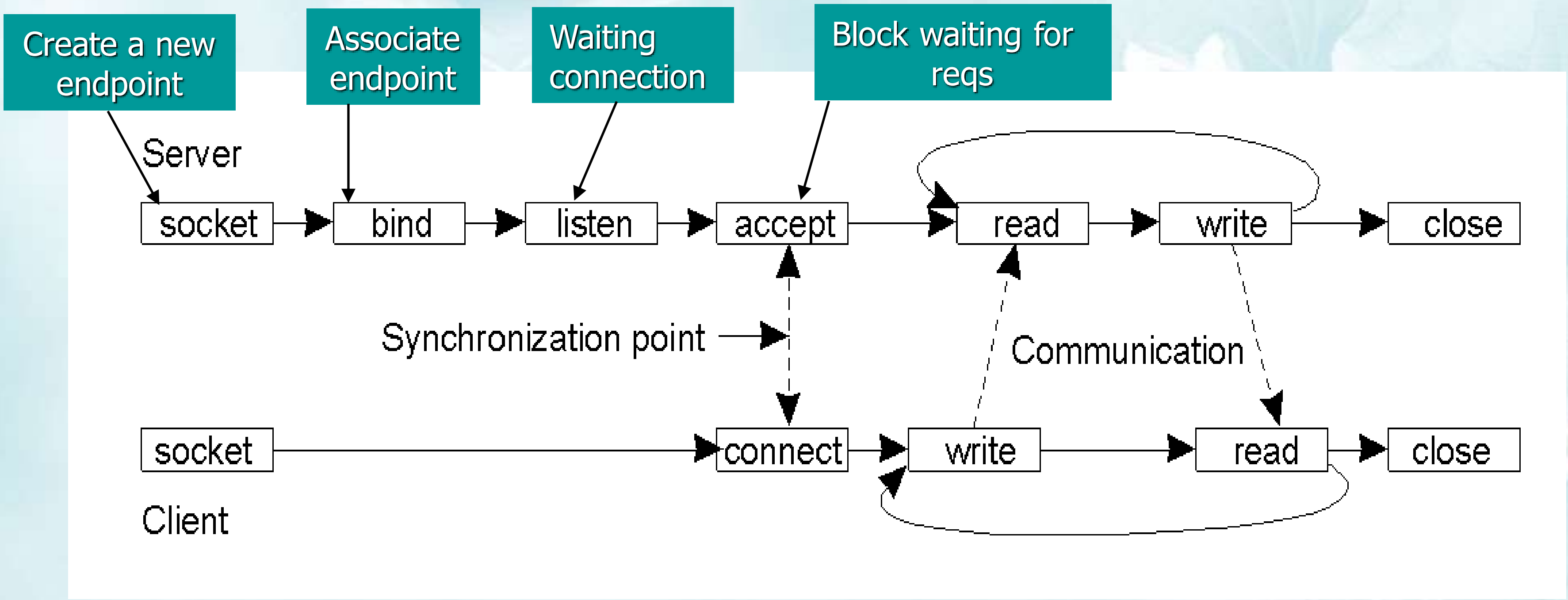
Message-Orient Transient Communication(2)

- Berkeley Sockets(1)
  - Meaning of Socket: a communication endpoint to which an application can write data (be sent to network) and read incoming data
  - The socket primitives for TCP/IP

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-Orient Transient Communication(3)

- Berkeley Sockets(2)
  - 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-Orient Transient Communication(4)

- The Message-Passing Interface(MPI)(1)
  - Designed for multiprocessor machines and high-performance parallel programming
  - Provides a high-level of abstraction than sockets
  - Support diverse forms of buffering and synchronization (over 100 functions)

Message-Orient Transient Communication(5)

- The Message-Passing Interface(MPI)(2)
  - 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

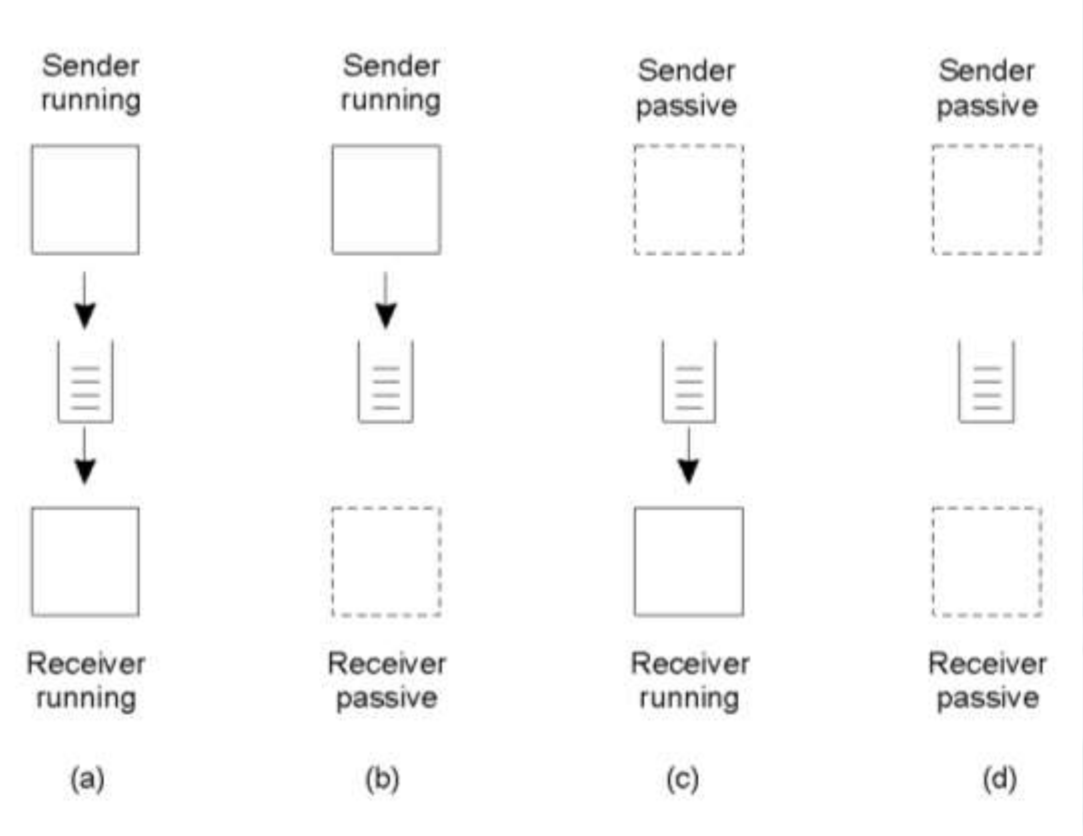
Message-Orient Persistent Communication(1)

- Message-Queuing Model(1)
  - 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

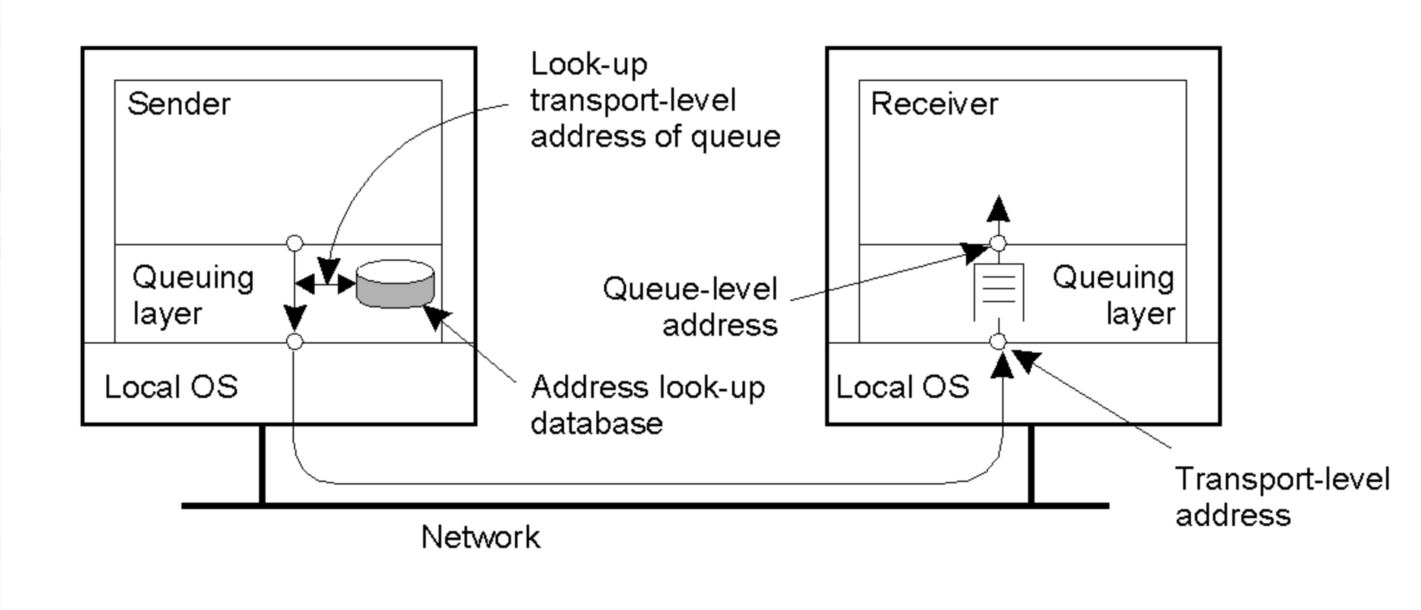
Message-Orient Persistent Communication(2)

- Message-Queuing Model(2)
- ‘
  - Four combinations for loosely-coupled communication using queues:



Message-Orient Persistent Communication(3)

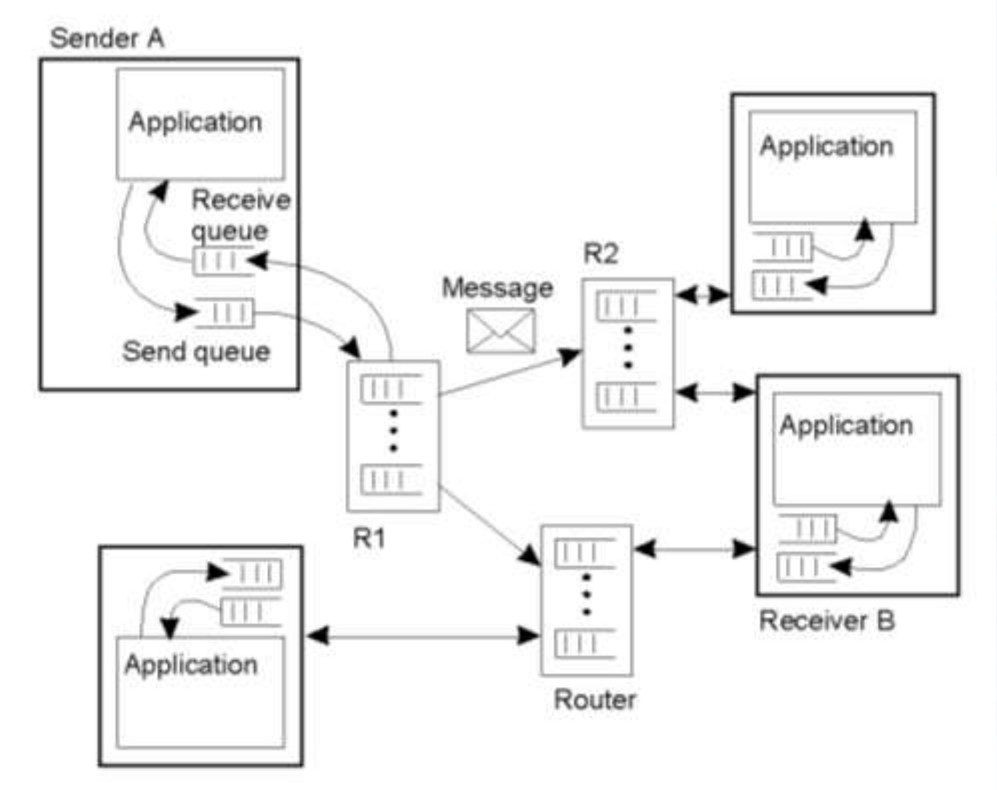
- General architecture of a Message-Queuing System(1)
  - Messages can only be put and received from local queues
  - Ensure transmitting the messages between the source queues and destination queues, meanwhile storing the messages as long as necessary
  - Each queue is maintained by a queue manager



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

Message-Orient Persistent Communication(4)

- General architecture of a Message-Queuing System(2)
  - Queue managers are not only responsible for directly interacting with applications but are also responsible for acting as relays (or routers)



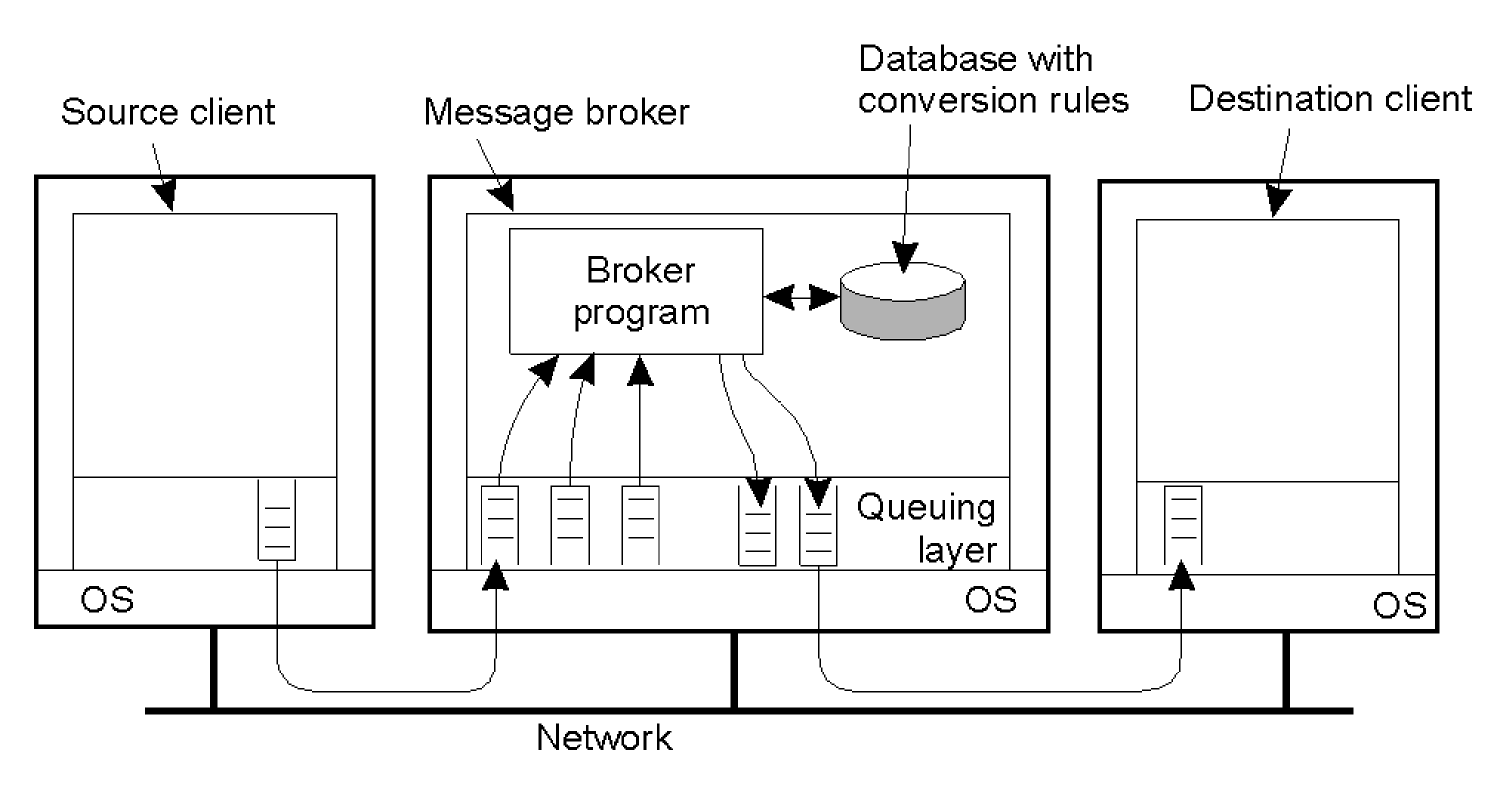
Queue managers form an overlay network, acting as routers

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(6)

- Message Broker



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

2.5 Stream-Oriented Communication

Manhyung Han(smiley@oslab.khu.ac.kr)

Support for Continuous Media(1)

- 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

Support for Continuous Media(2)

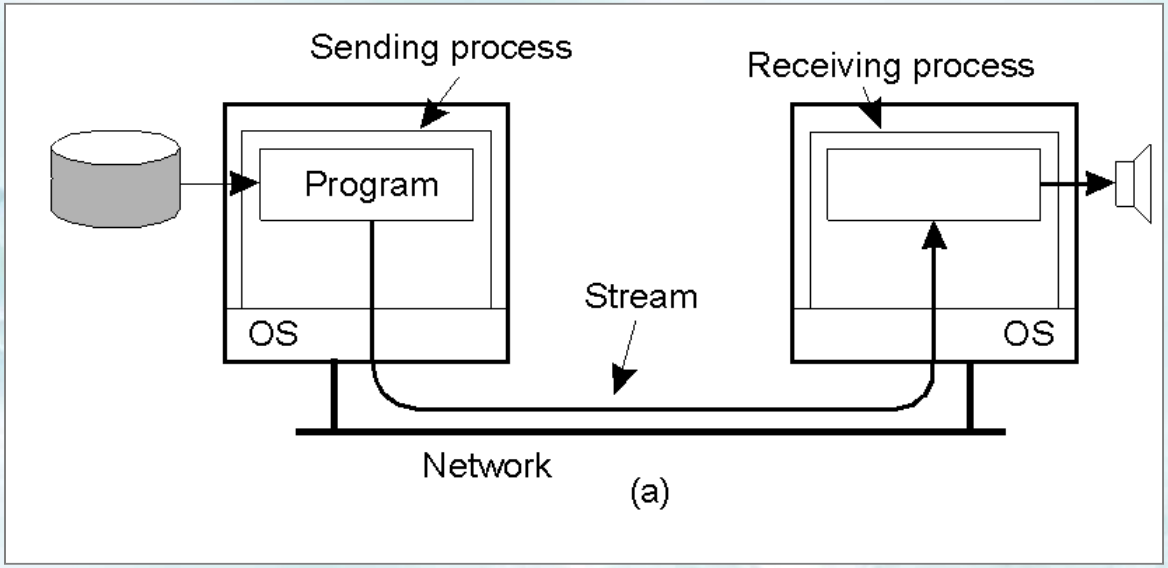
- Data Stream
  - Sequence of data units
  - Discrete data stream: UNIX pipes or TCP/IP connection
  - Continuous data stream: audio file (connection between file and audio devices)
- Transmission modes
  - Asynchronous: no timing constraints
  - Synchronous: upper bound on propagation delay
  - Isochronous: upper and lower bounds on propagation delay (transfer on time)

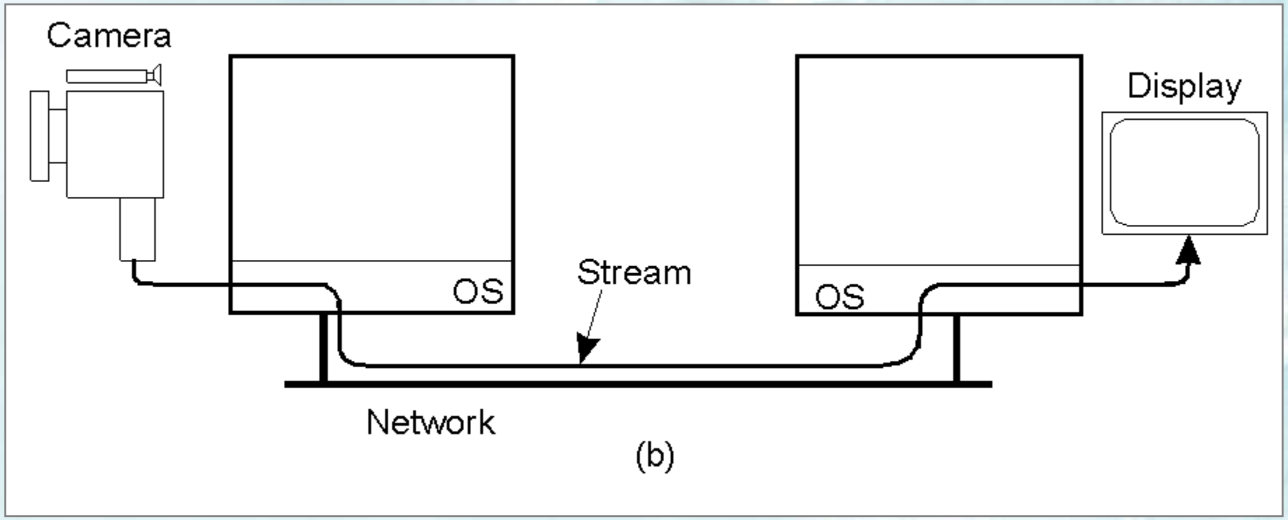
Support for Continuous Media(3)

- 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

Support for Continuous Media(4)

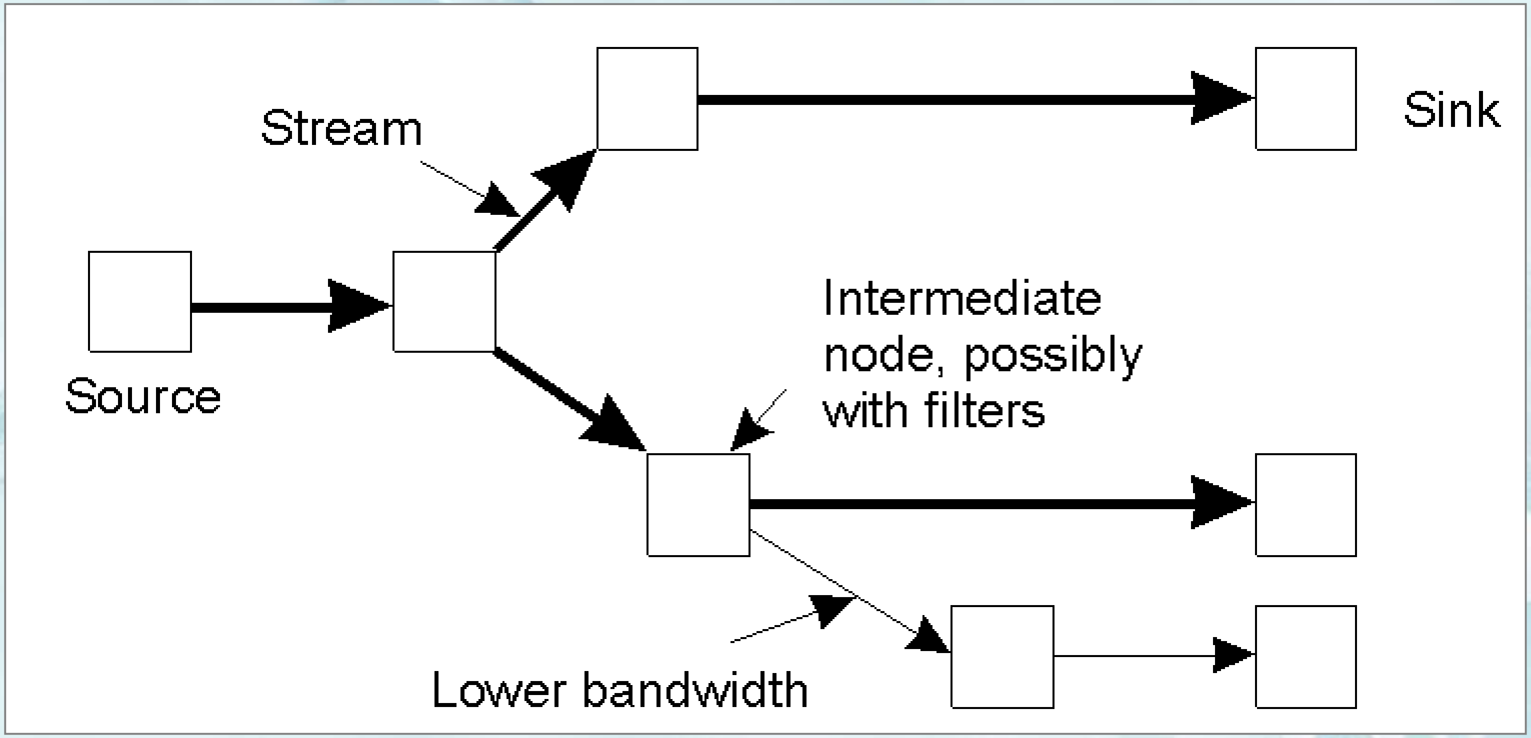
Figure. 2-35
(a) Setting up a stream between two processes across a network
(b) Setting up a stream directly between two devices





Support for Continuous Media(5)

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



Streams and Quality of Service(1)

- Specifying QoS(1)
  - Flow specification
  - To provide a precise factors(bandwith, transmission rates and delay, etc.)
  - Example of flow specification developed by Partridge

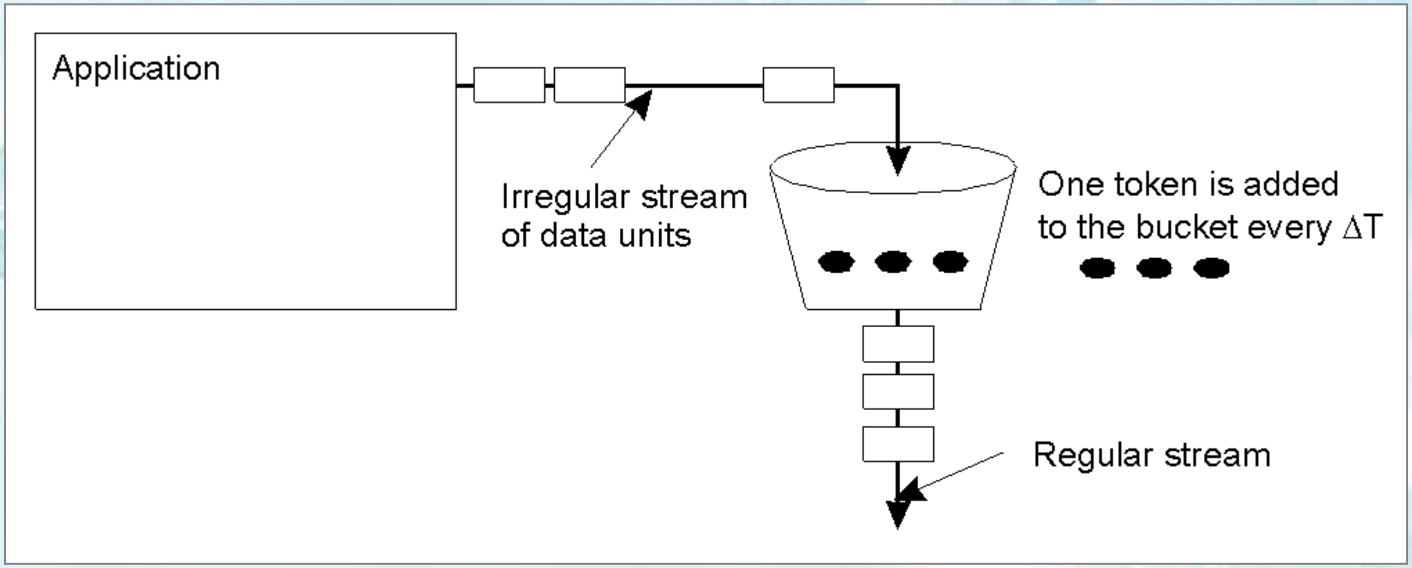
Figure. 2-37 *A flow specification*

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(2)

- Specifying QoS(2)
  - 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



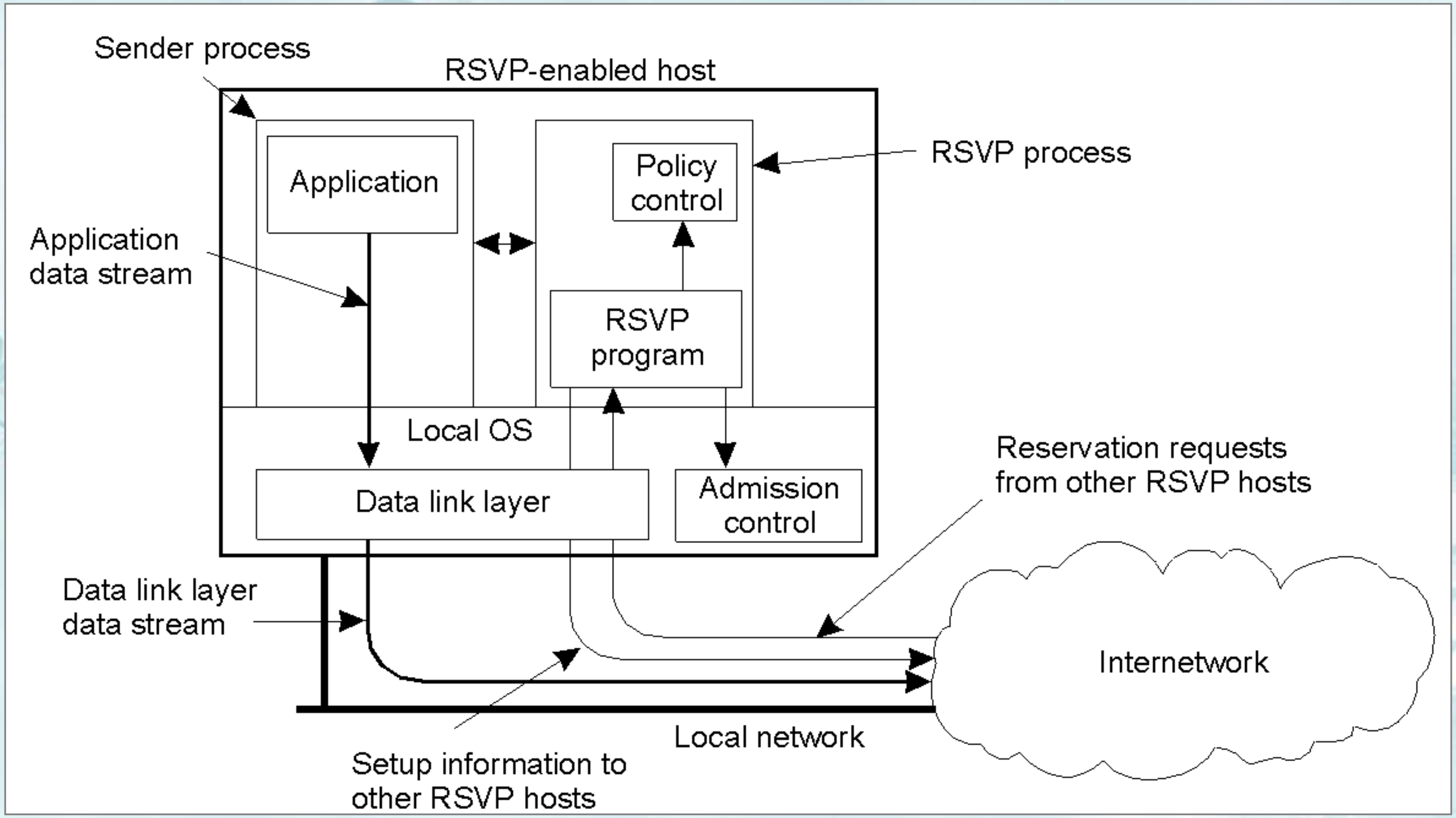
Streams and Quality of Service(3)

- 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(4)

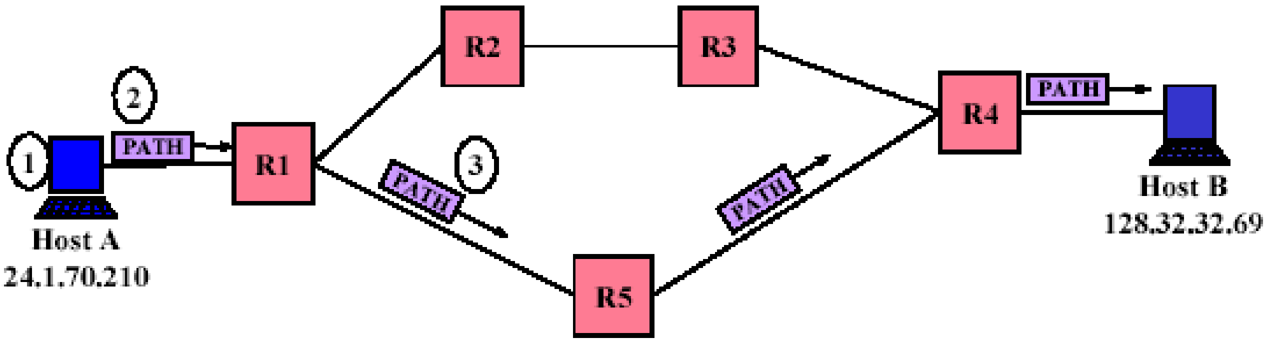
Figure. 2-39
The basic organization of RSVP for resource reservation in a distributed system



Streams and Quality of Service(5)

RSVP Reservation

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



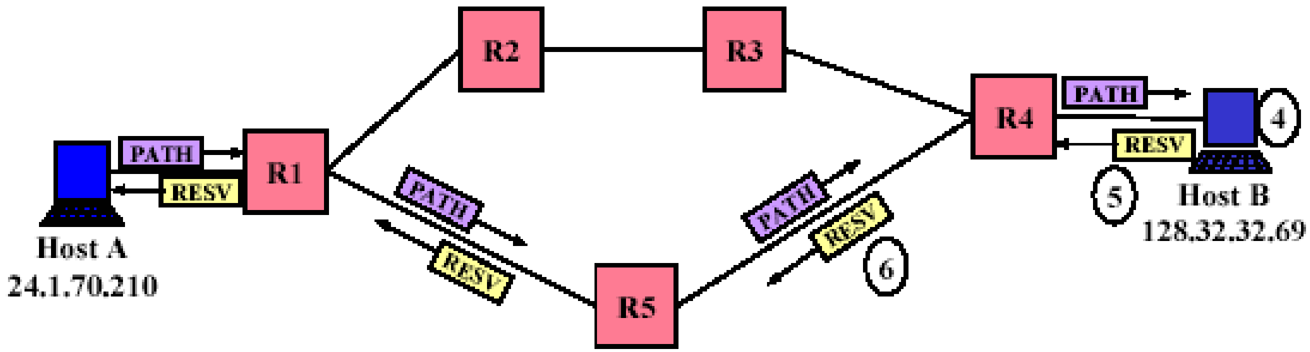
1. An application on Host A creates a session, 128.32.32.69/4078, by communicating with the RSVP daemon on Host A.

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(6)

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.

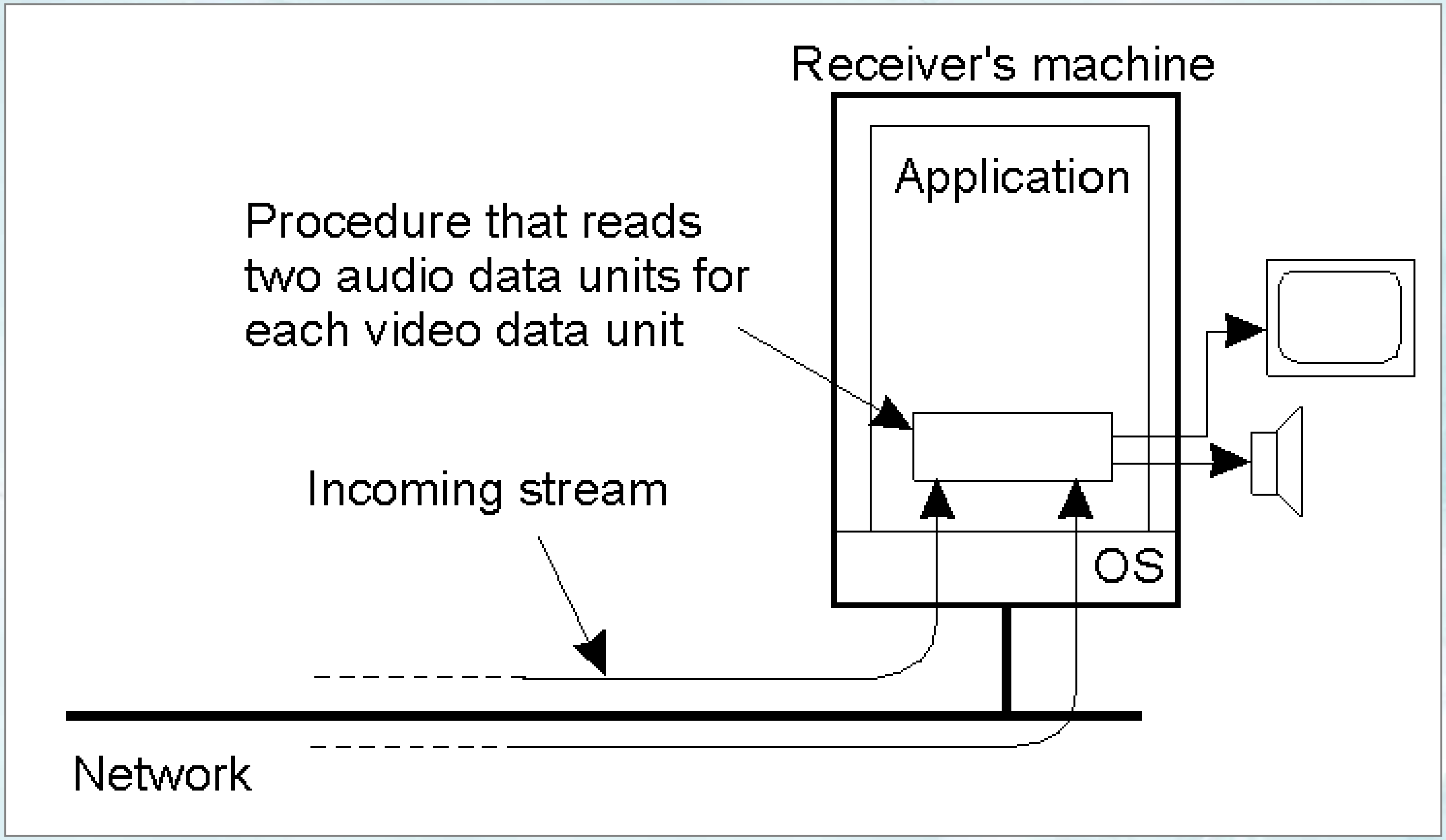
Stream Synchronization(1)

- 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(2)

- Synchronization Mechanisms(1)

Figure. 2-40 *The principle of explicit synchronization on on the level data units*



Stream Synchronization(3)

- Synchronization Mechanisms(2)

Figure. 2-41
The principle of synchronization as supported by high-level interfaces