Parallelism

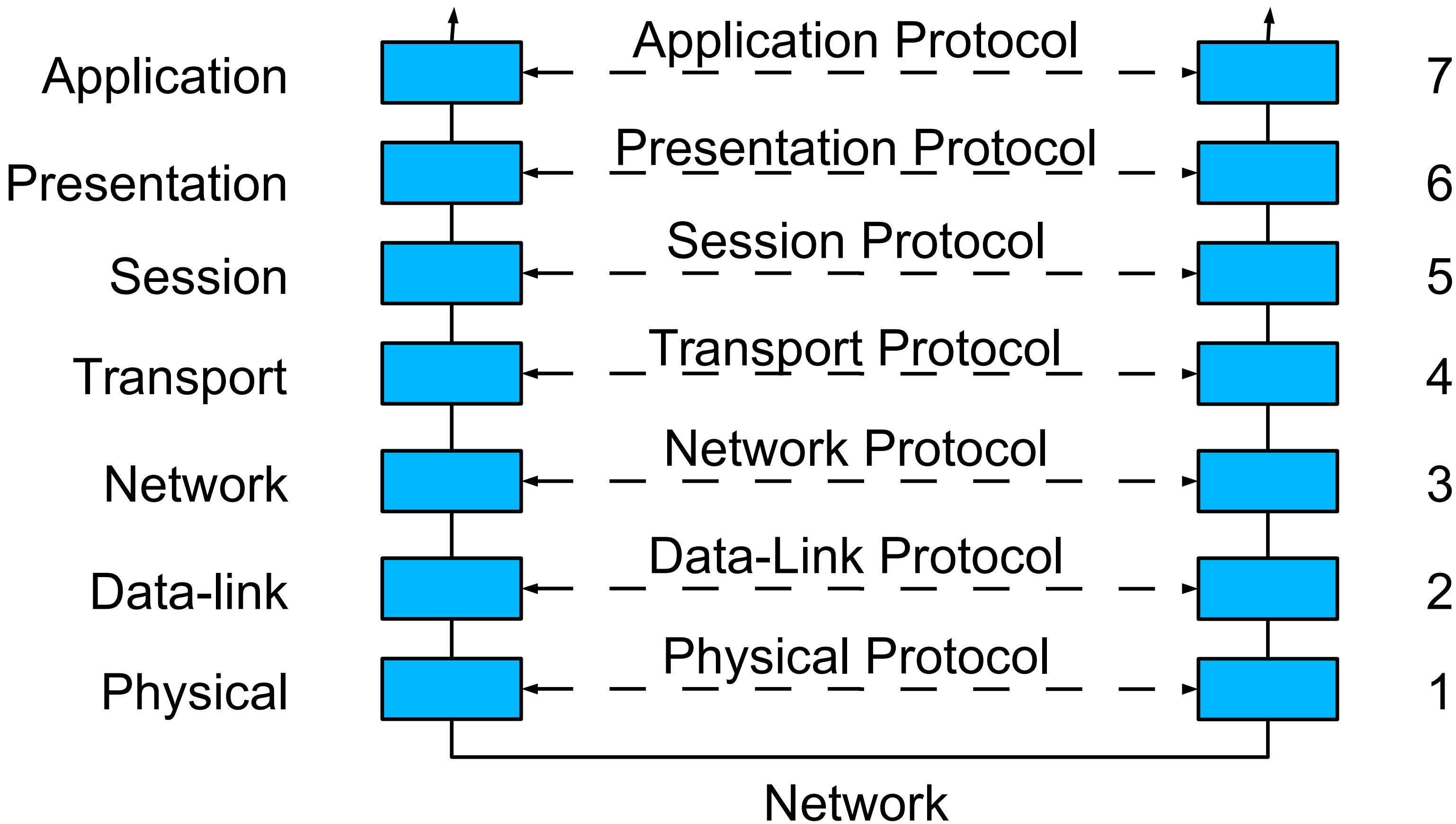
Master 1 International

Andrea G. B. Tettamanzi
Université de Nice Sophia Antipolis
Département Informatique
andrea.tettamanzi@unice.fr

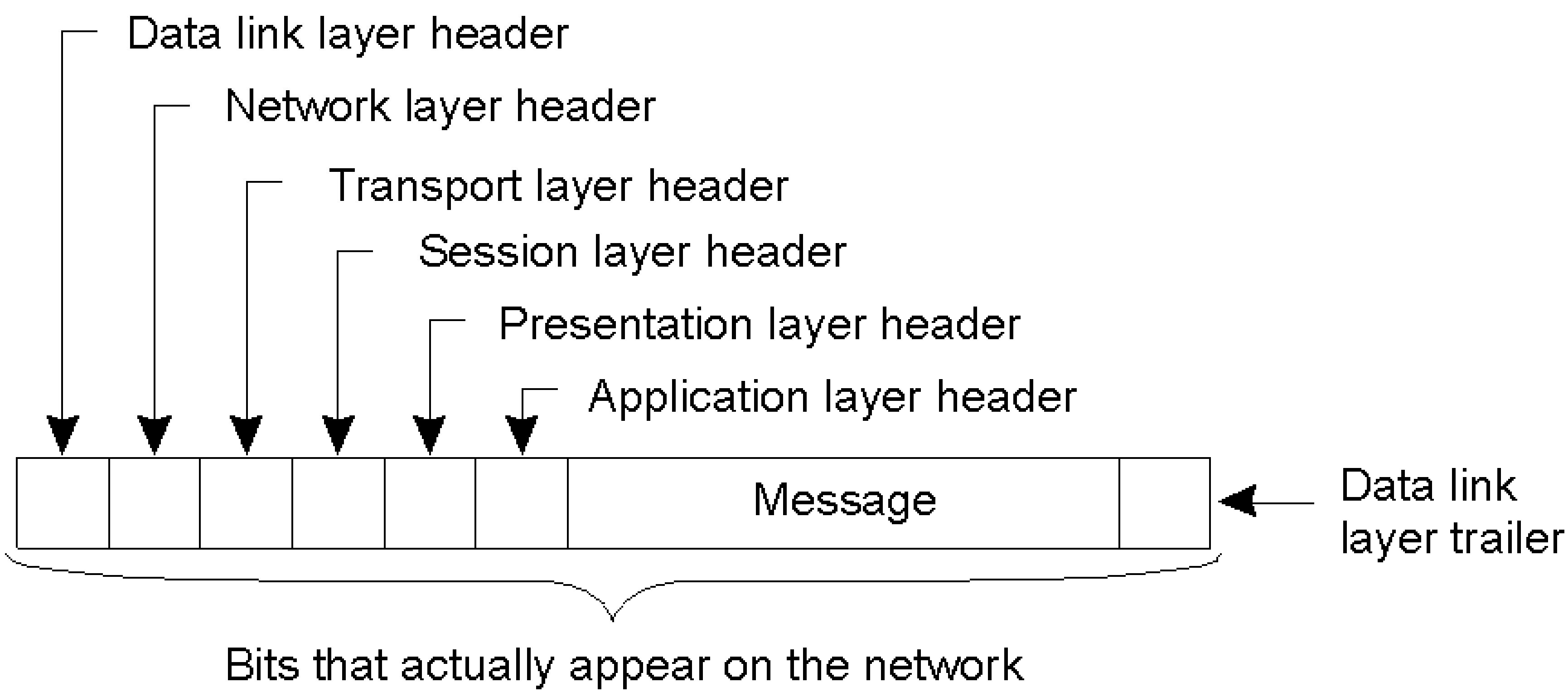
Lecture 2

Communication

Layered Protocols: the ISO/OSI Stack



Layered Protocols: Messages

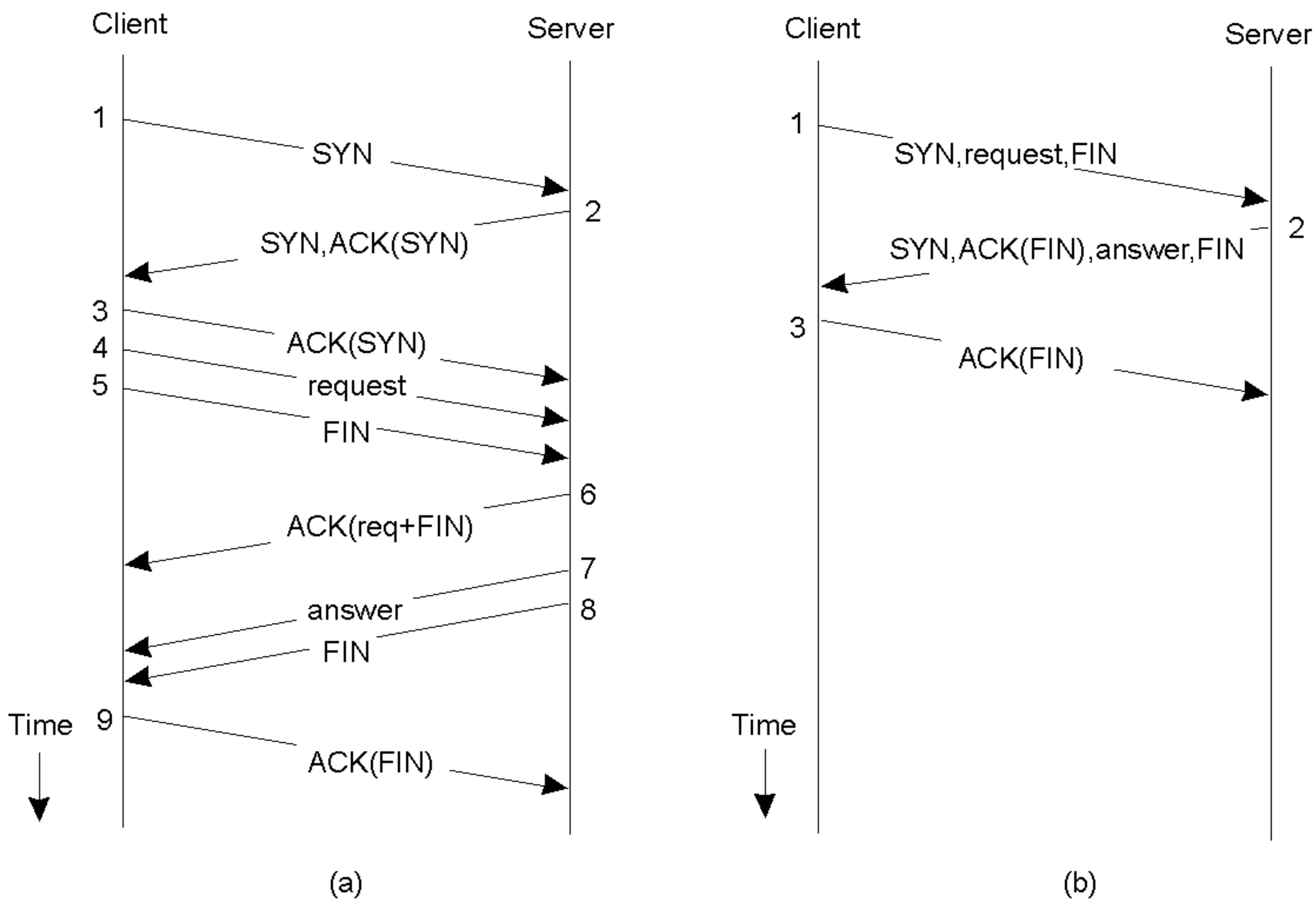


A typical message as it appears on the network.

Data Link Layer

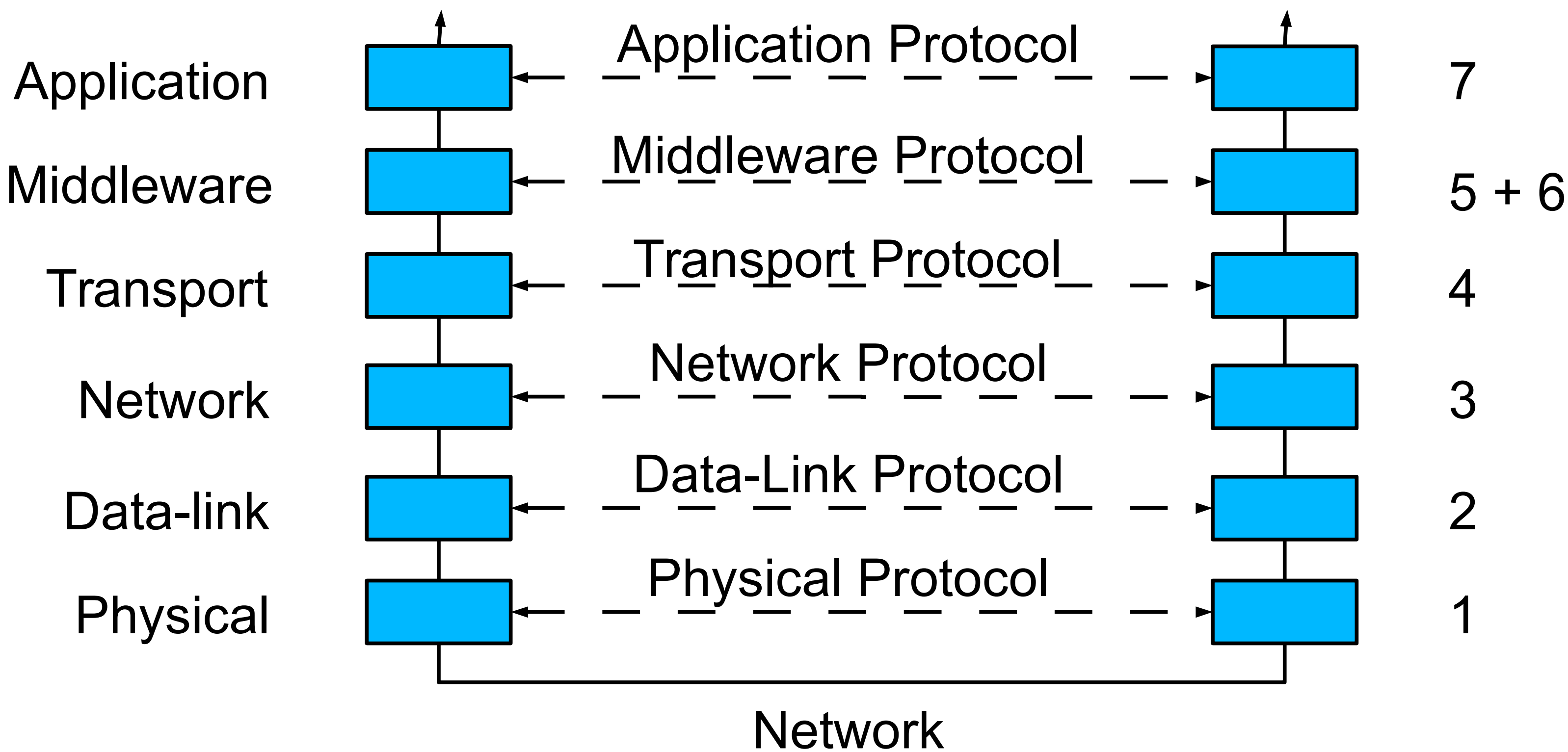
Time	A	B	Event
0	Data 0		A sends data message 0
1		Data 0	B gets 0, sees bad checksum
2	Data 1	Control 0	A sends data message 1 B complains about the checksum
3	Control 0	Data 1	Both messages arrive correctly
4	Data 0	Control 1	A retransmits data message 0 B says: "I want 0, not 1"
5	Control 1	Data 0	Both messages arrive correctly
6	Data 0		A retransmits data message 0 again
7		Data 0	B finally gets message 0

Client-Server TCP

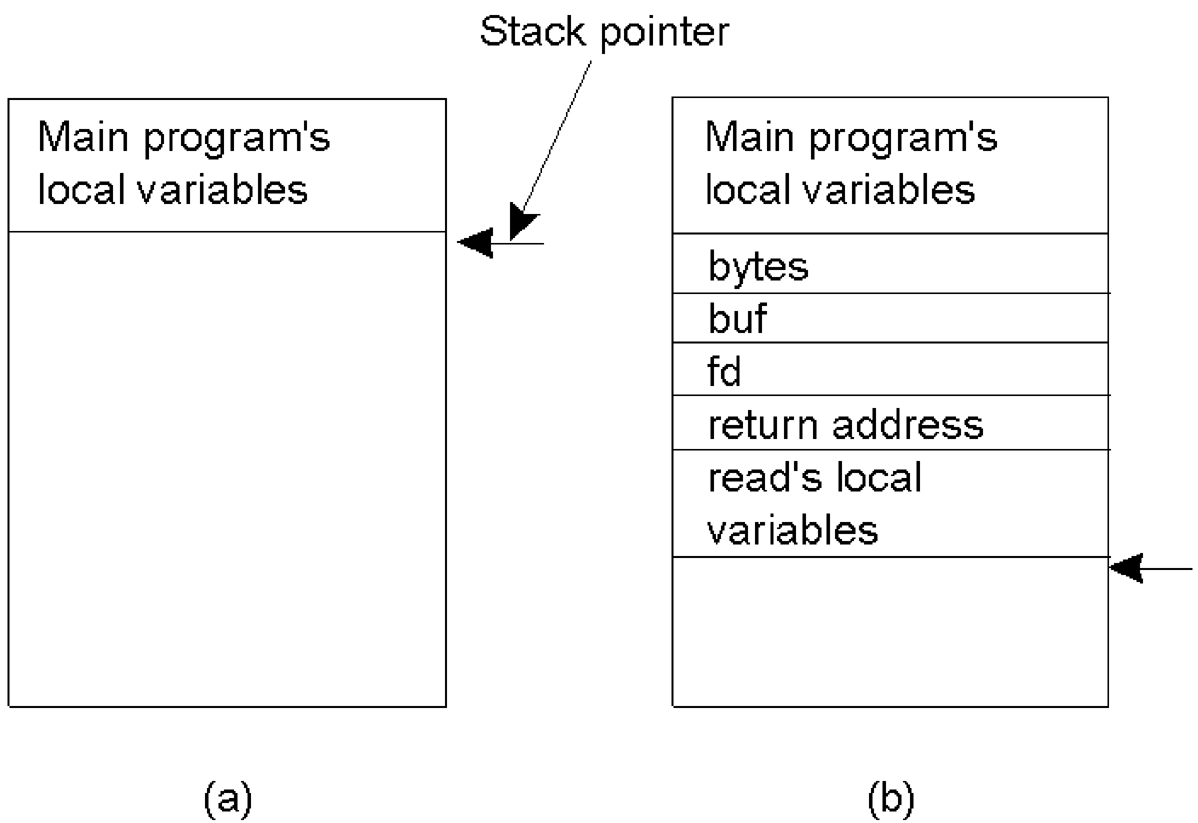


a) Normal operation of TCP. b) Transactional TCP.

Middleware Protocols: An adaptation of the ISO/OSI Stack

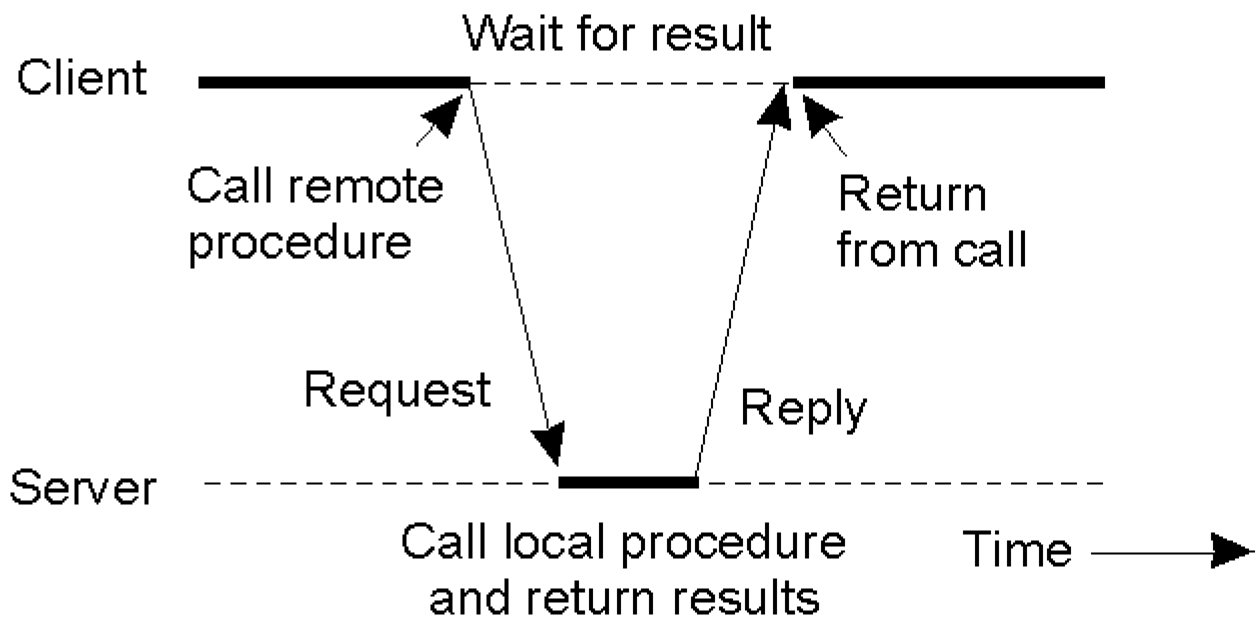


Conventional Procedure Call



a) Parameter passing in a local procedure call: the stack before the call
b) The stack while the called procedure is active

Client and Server Stubs

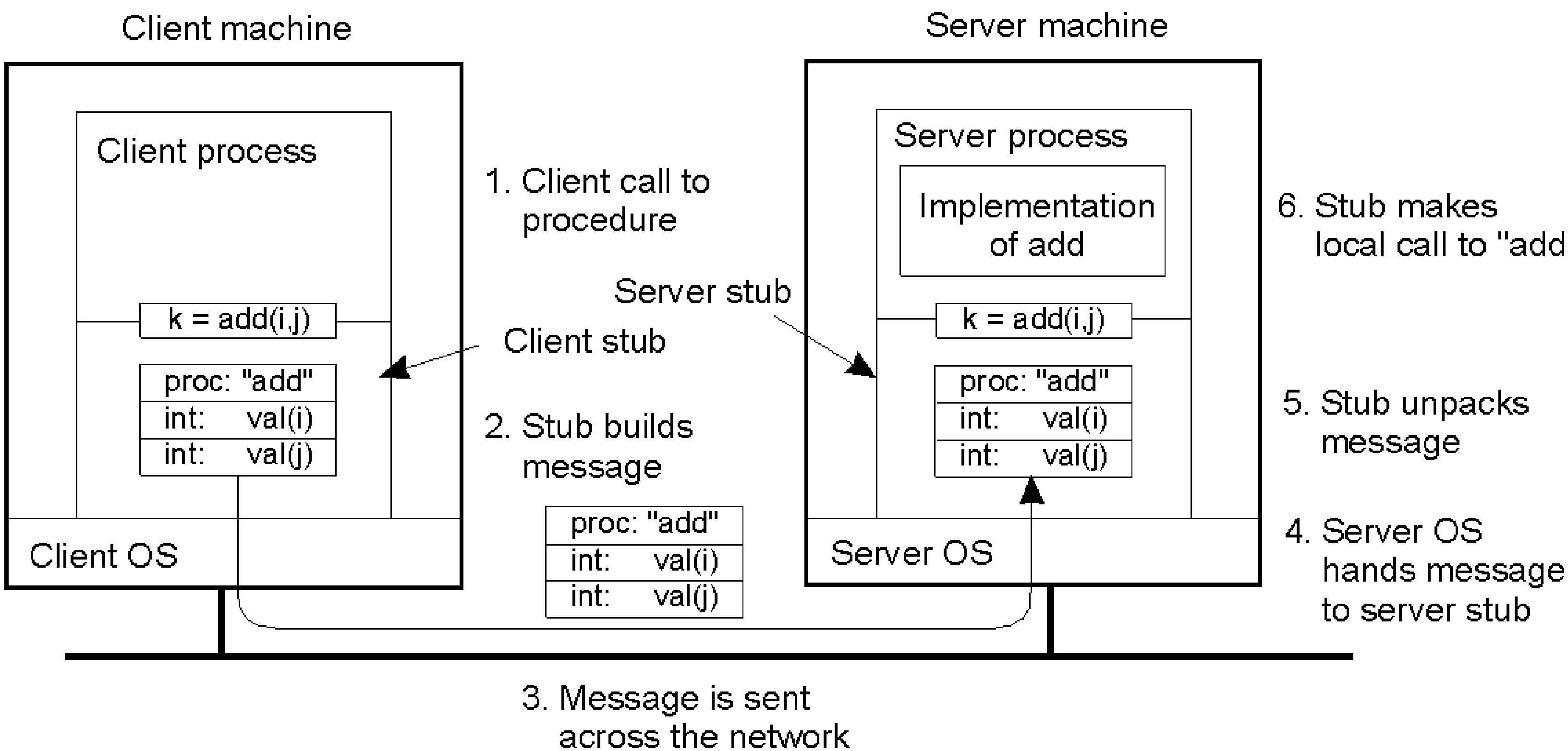


Principle of RPC between a client and server program.

Steps of a Remote Procedure Call

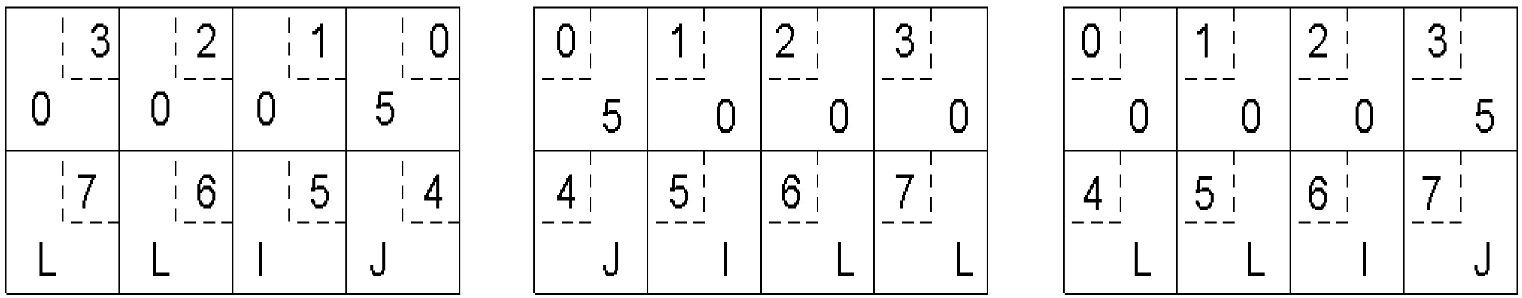
1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it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



(a) (b) (c)

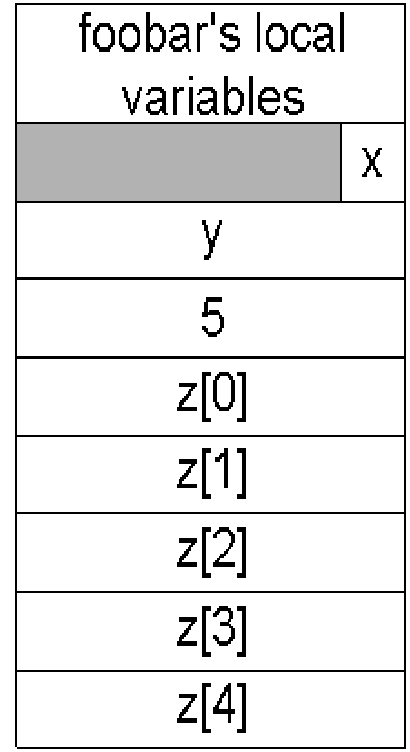
a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

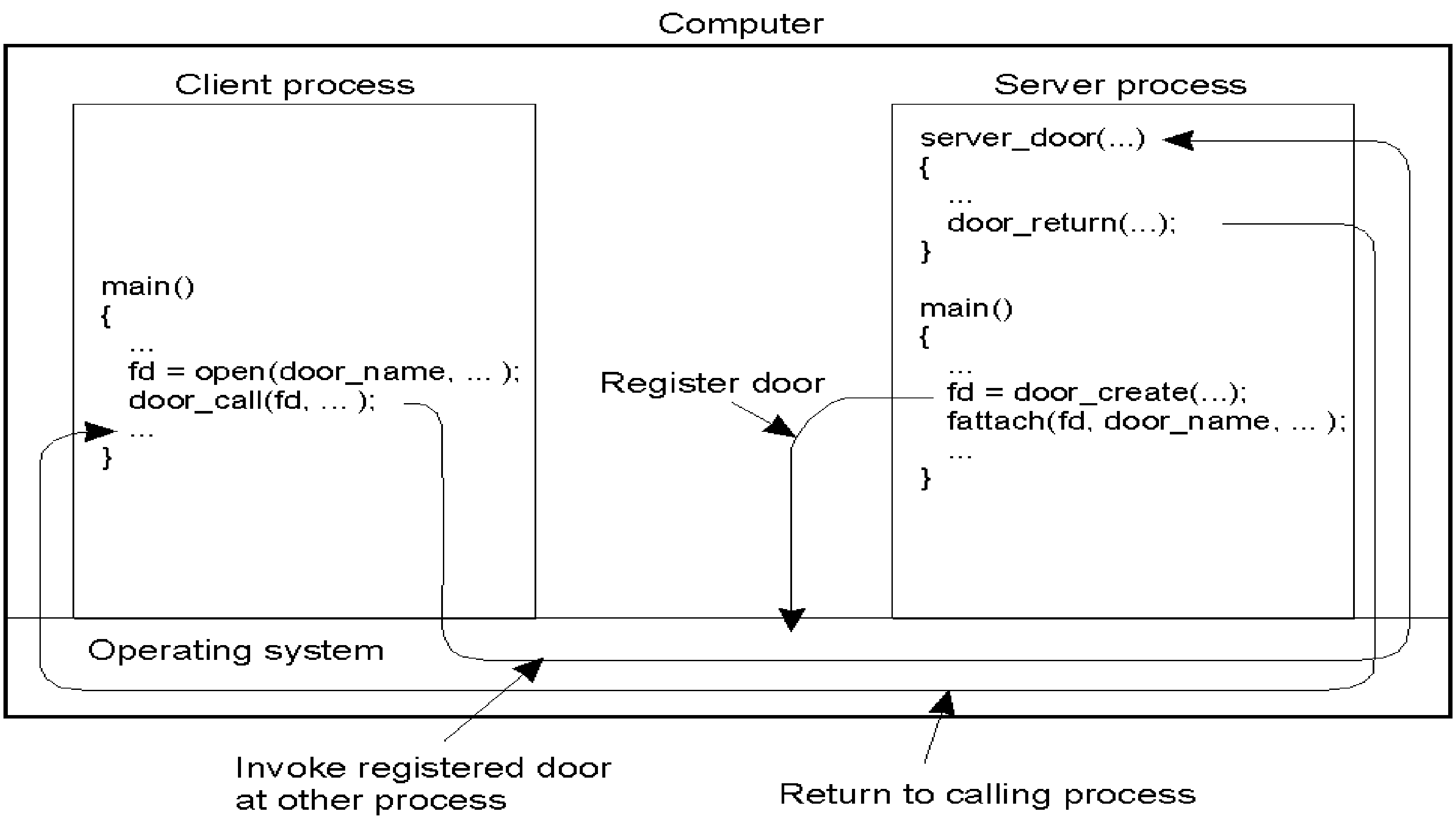
```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



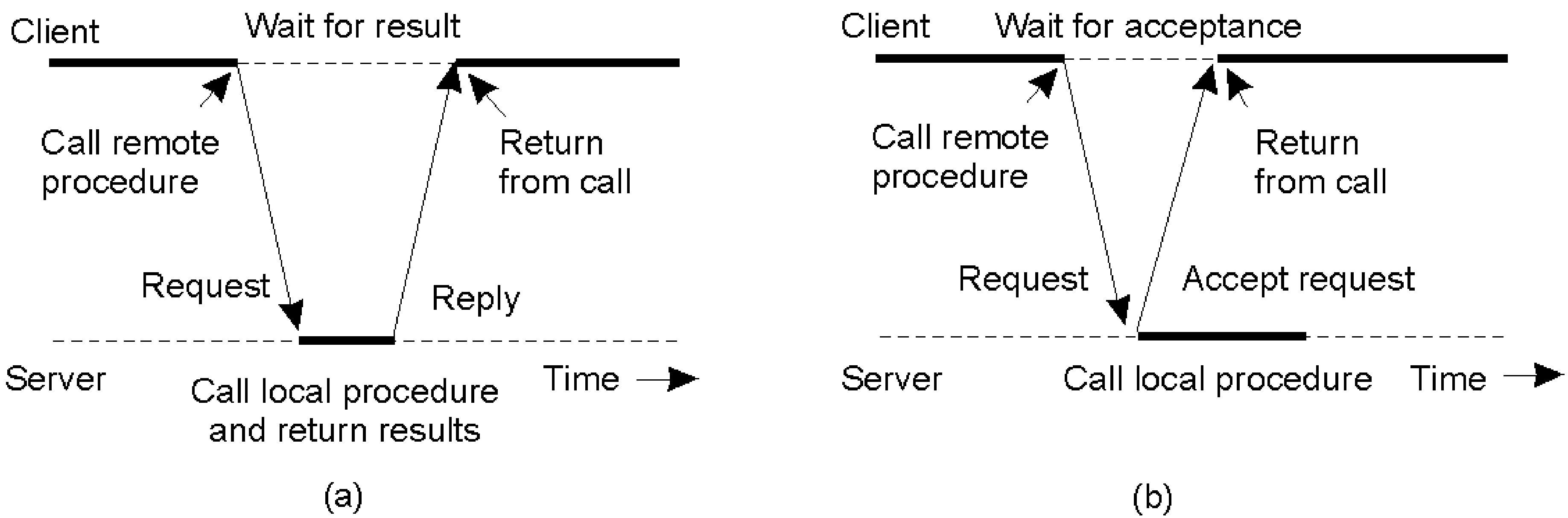
(b)

Doors



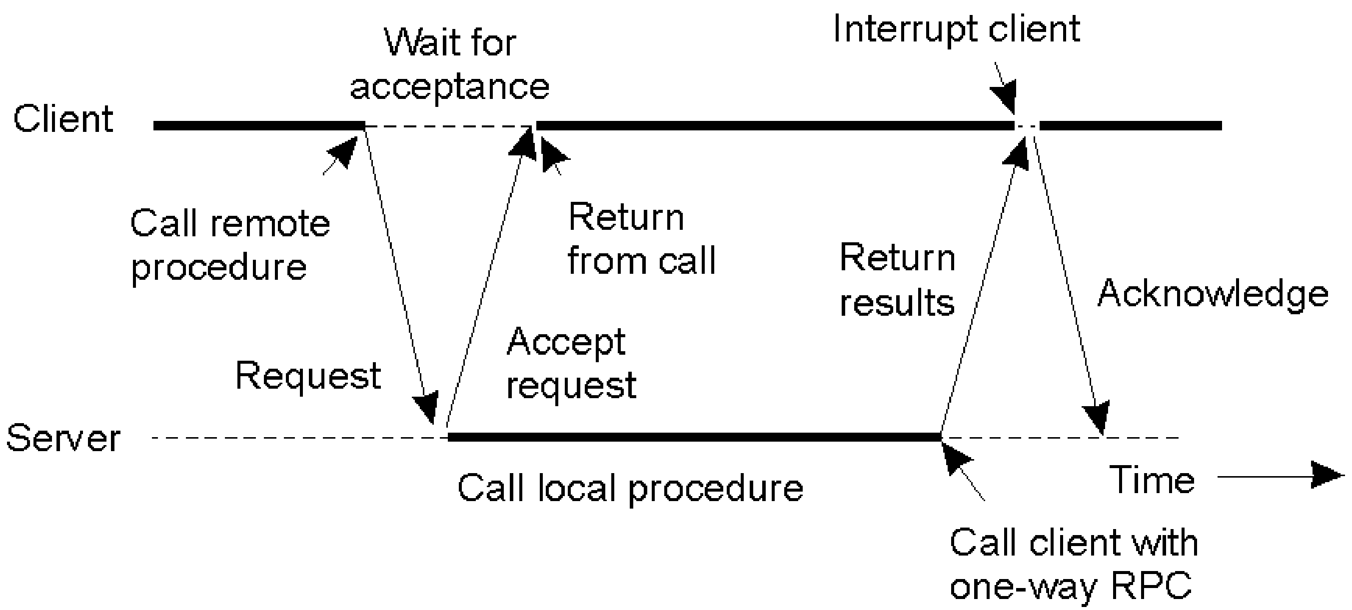
The principle of using doors as IPC mechanism.

Asynchronous RPC (1)



a) The interconnection between client and server in a traditional RPC
b) The interaction using asynchronous RPC

Asynchronous RPC (2)

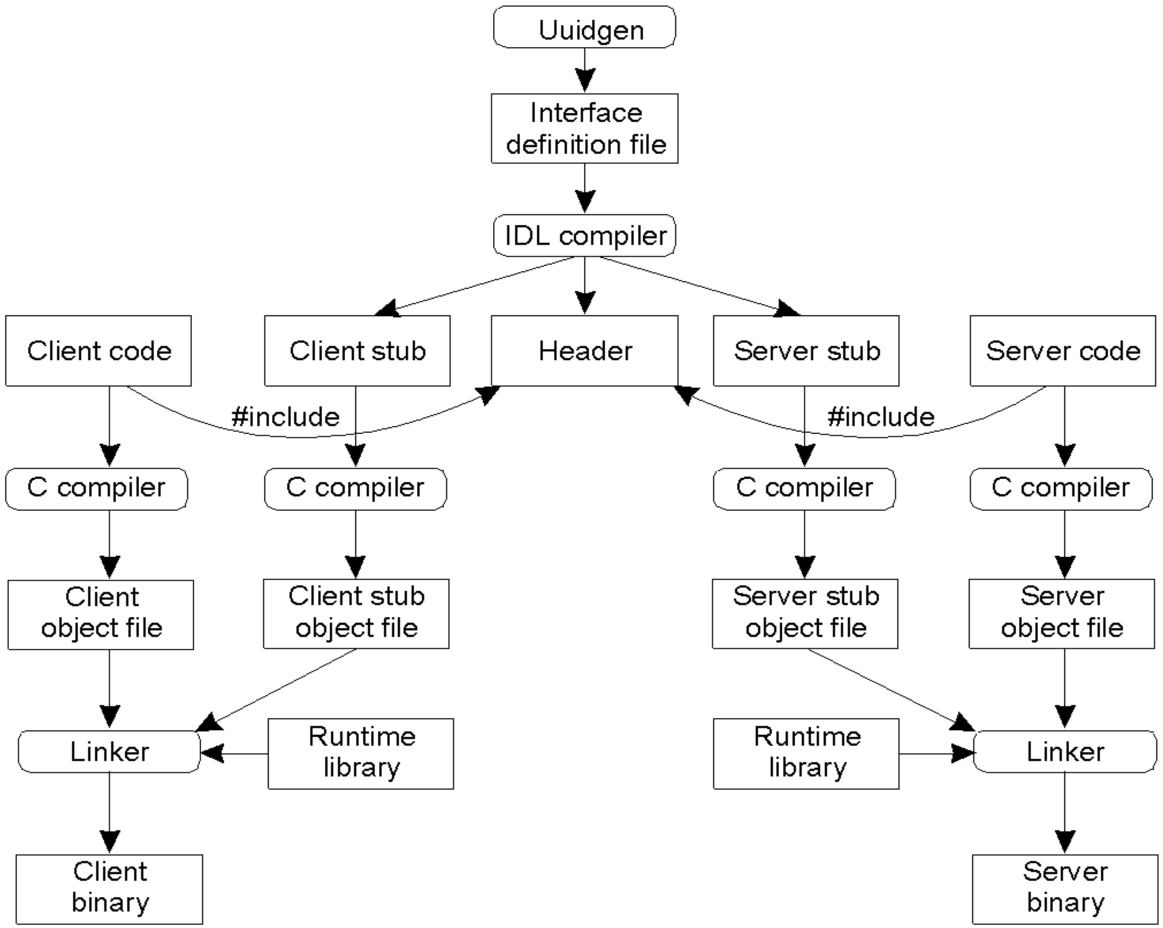


A client and server interacting through two asynchronous RPCs

DCE

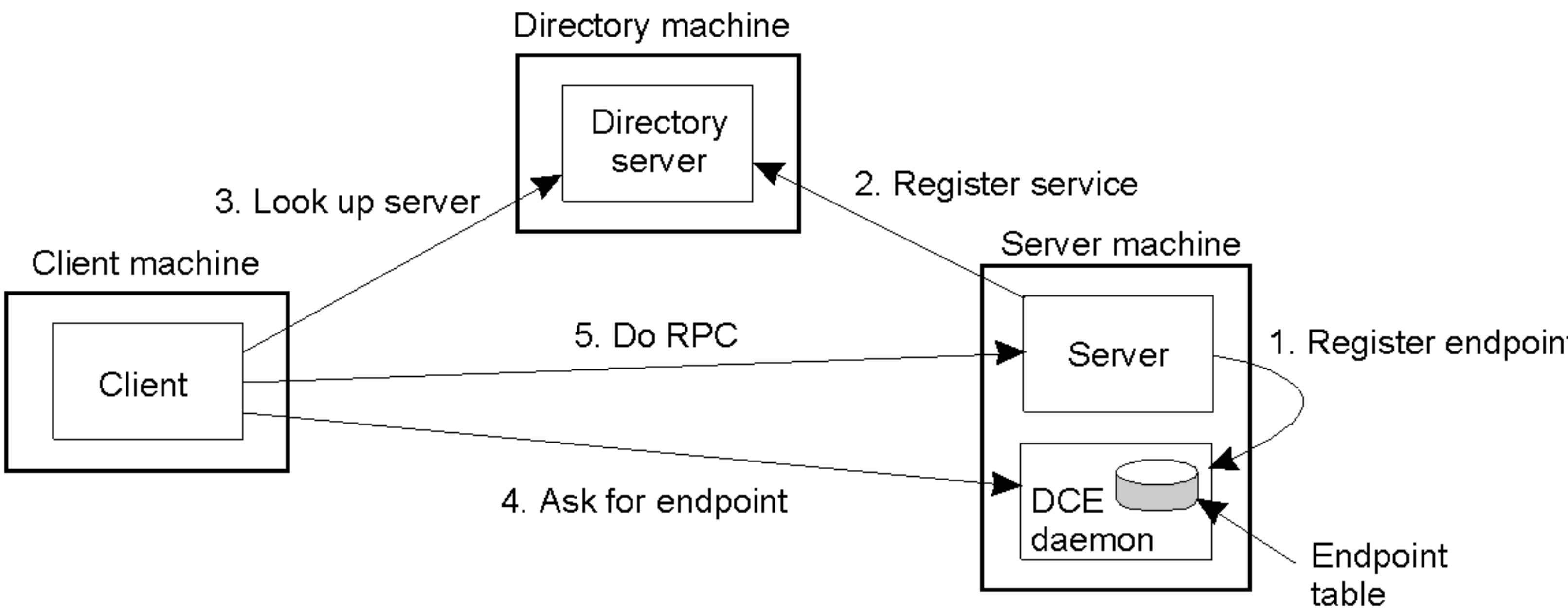
- DCE = Distributed Computing Environment
- Developed in the early '90 by a consortium of Apollo Computer (later acquired by HP), IBM, DEC, and others
- DCE supplies a framework for client/server applications
- The framework includes :
  - DCE/RPC, a remote procedure call mechanism
  - A naming service
  - A time service
  - An authentication service
  - DCE/DFS, a distributed file system
- Now OpenDCE: http://www.opengroup.org/dce/

Writing a Client and a Server



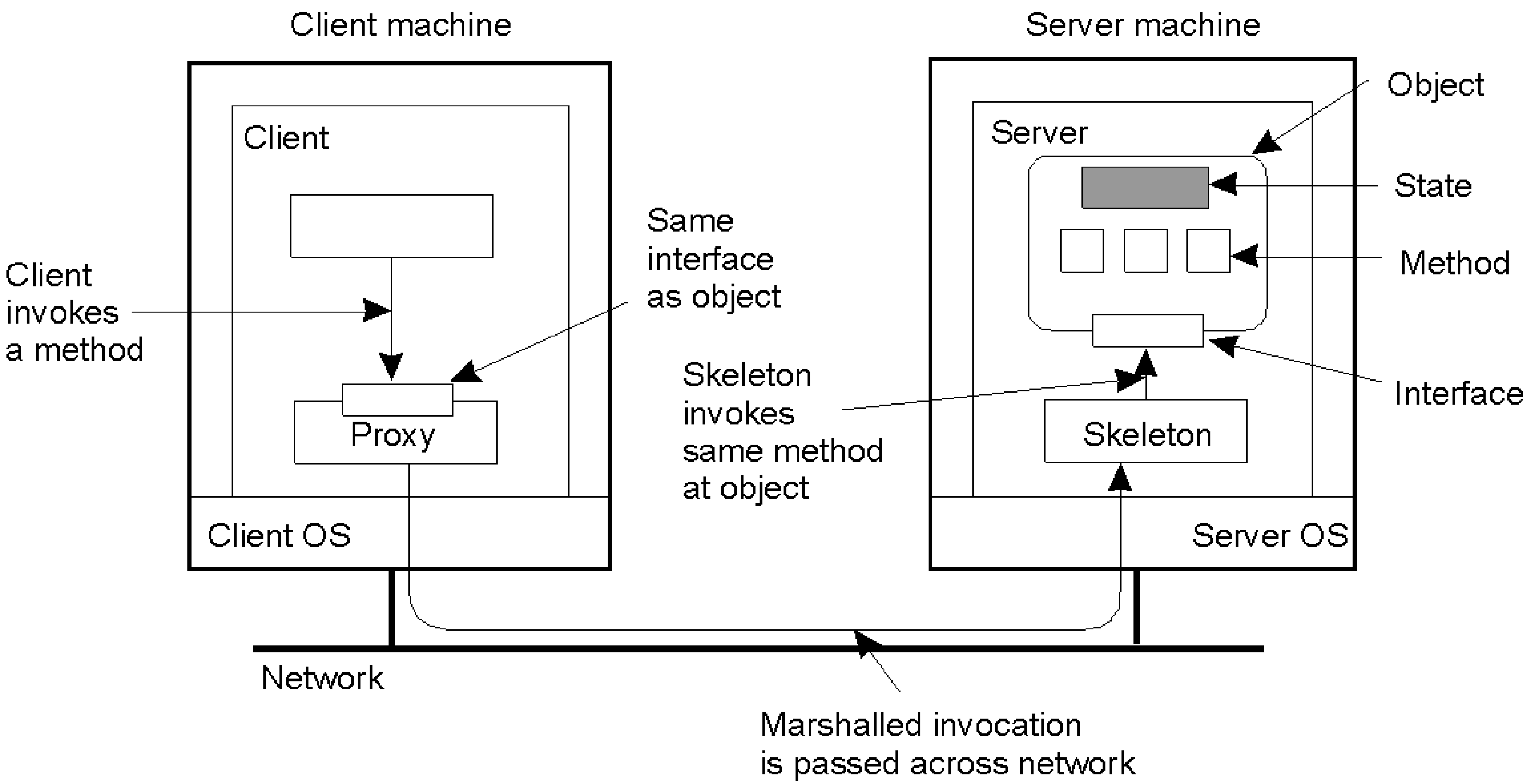
The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server



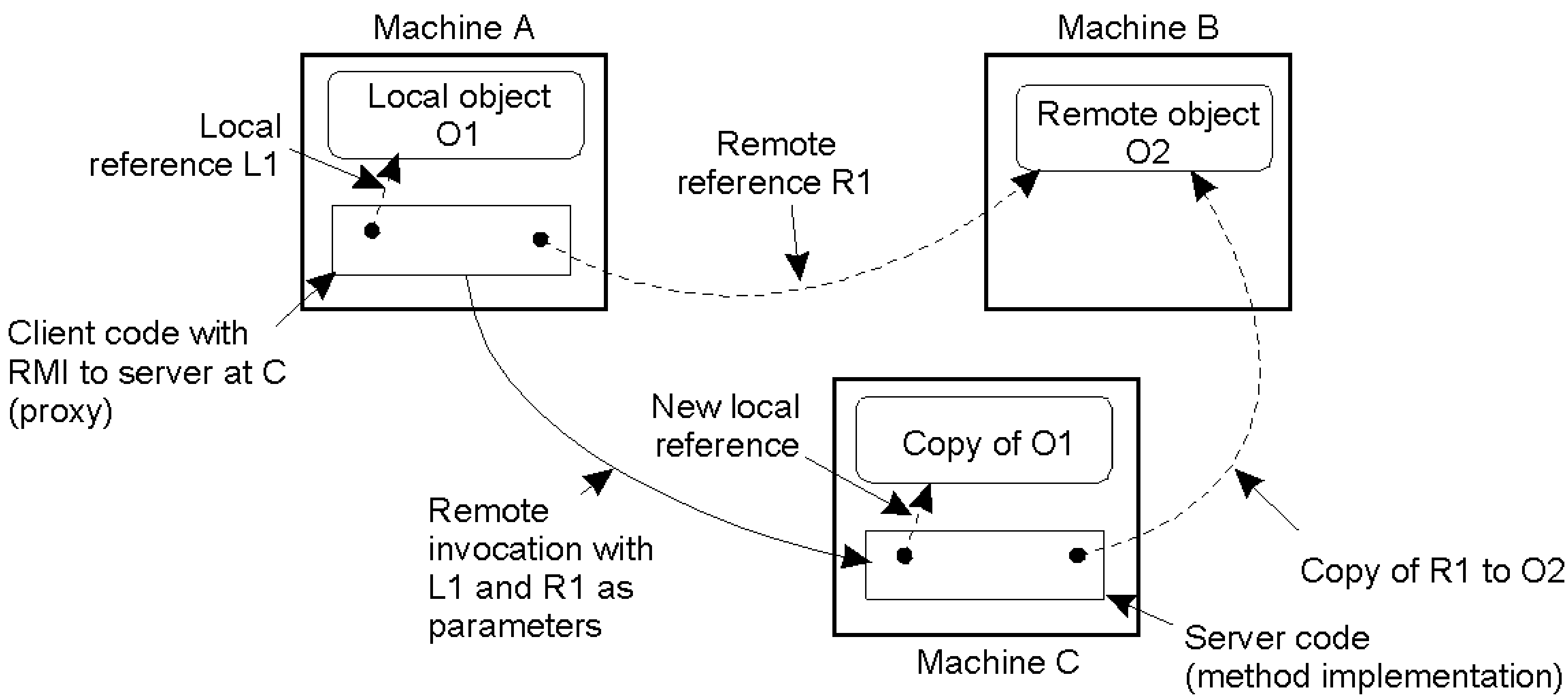
Client-to-server binding in DCE.

Distributed Objects



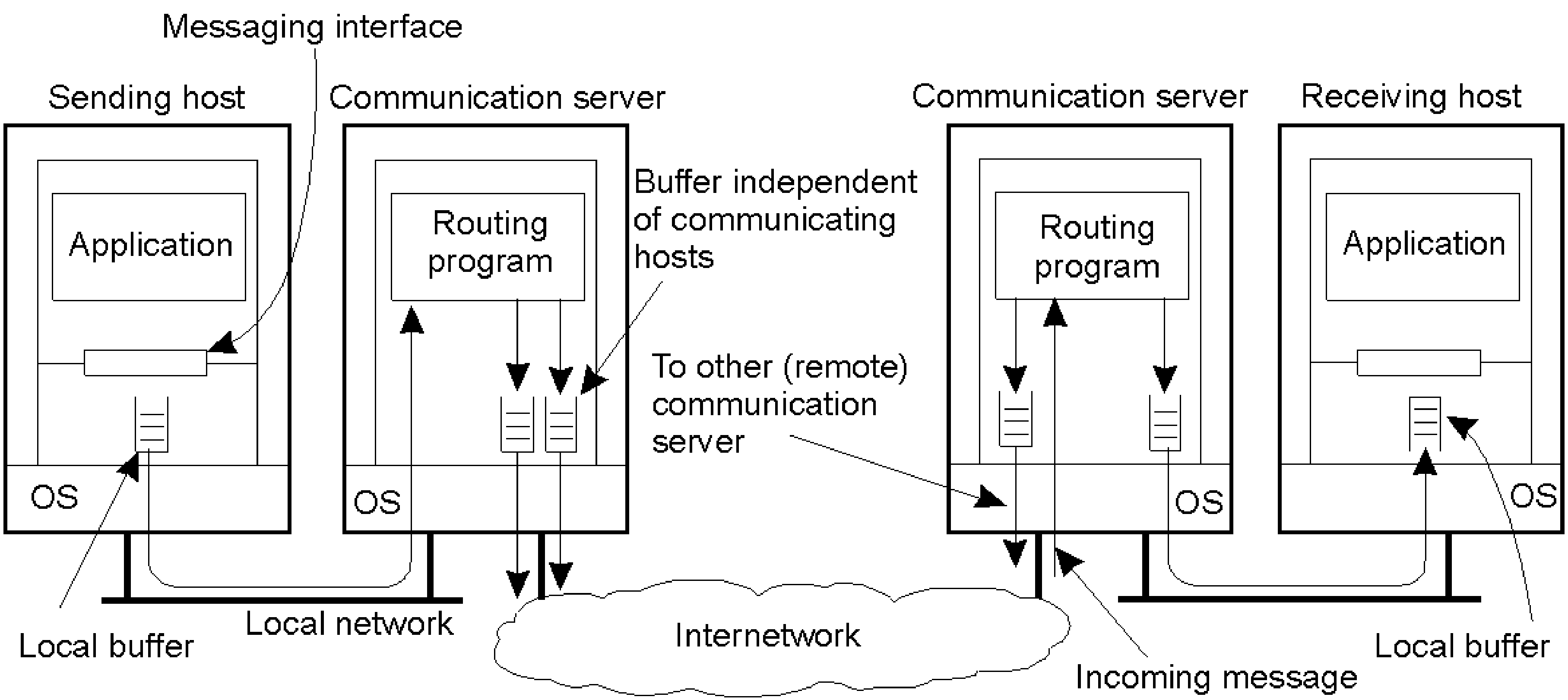
Common organization of a remote object with client-side proxy.

Parameter Passing



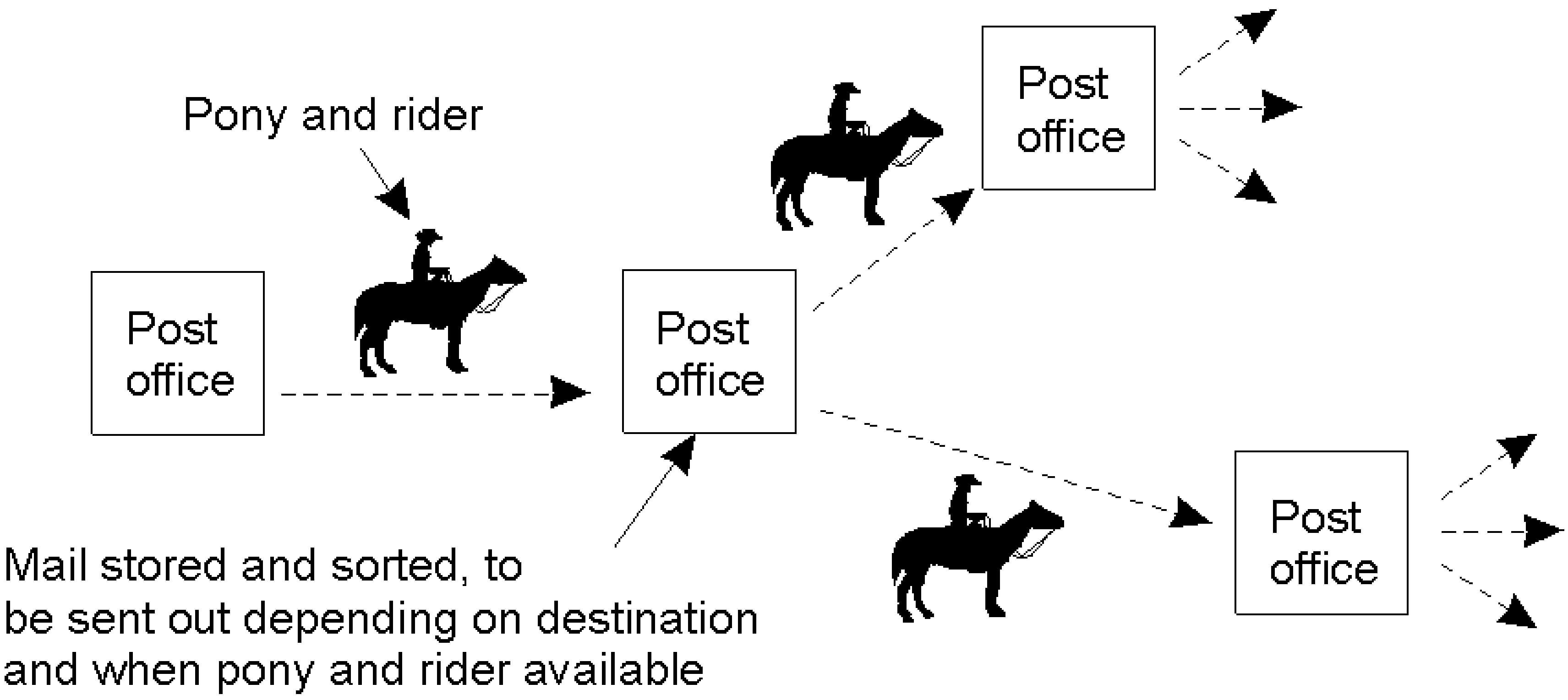
The situation when passing an object by reference or by value.

Persistence and Synchronicity (1)



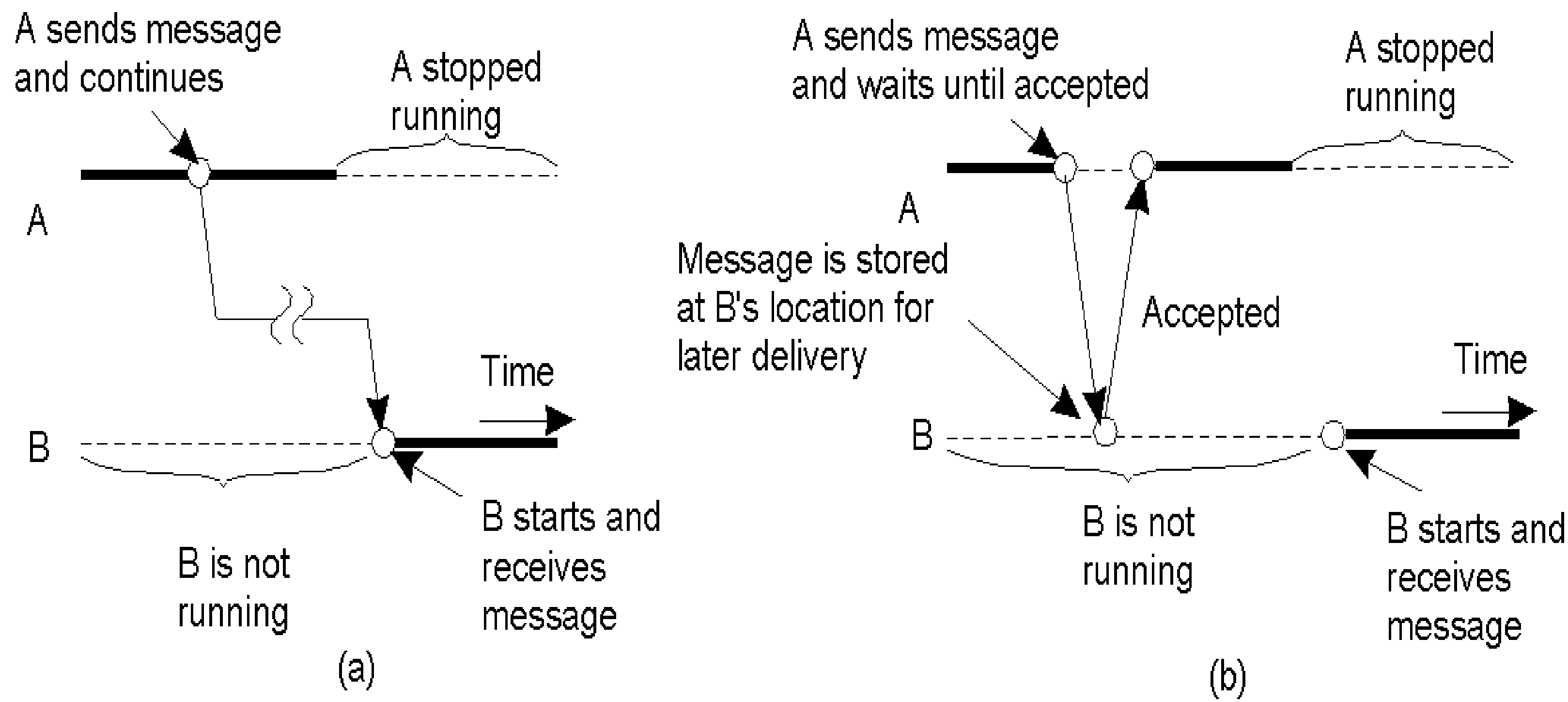
General organization of a communication system in which hosts are connected through a network

Persistence and Synchronicity (2)



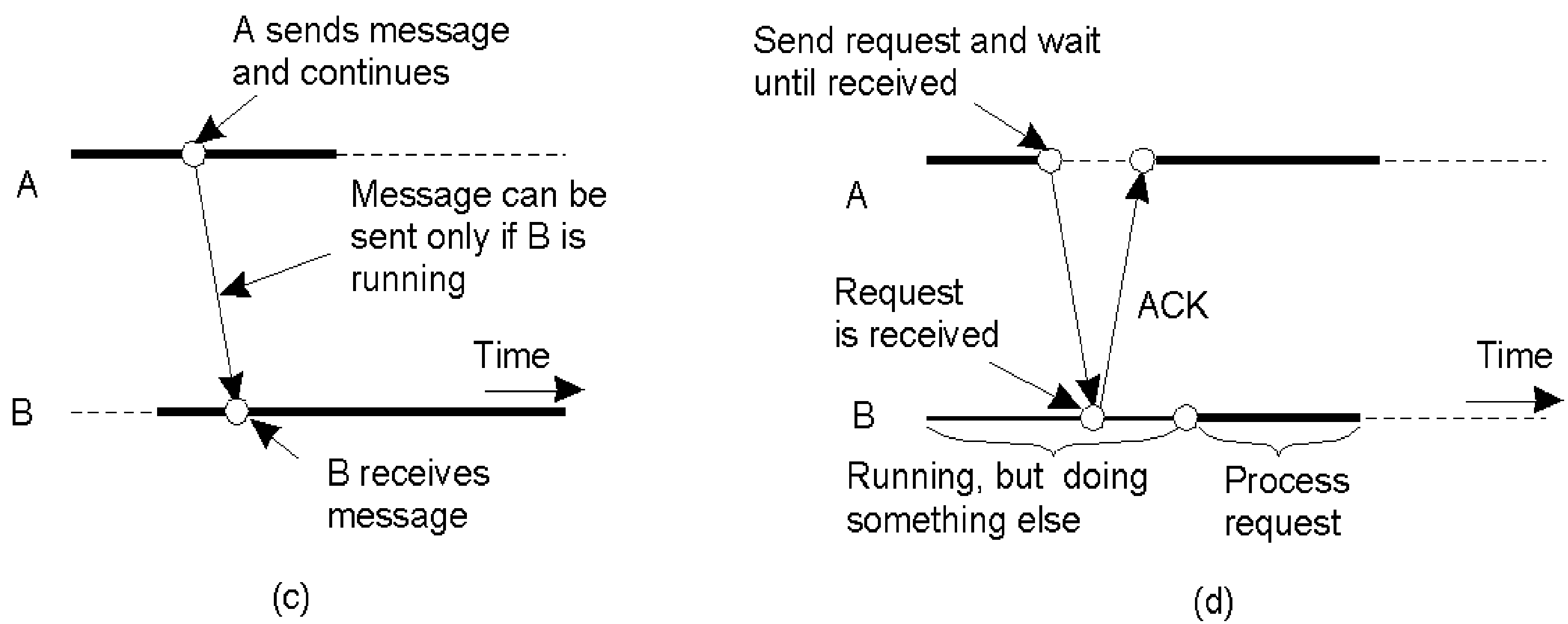
Persistent communication in the days of the Pony Express.

Persistence and Synchronicity (3)



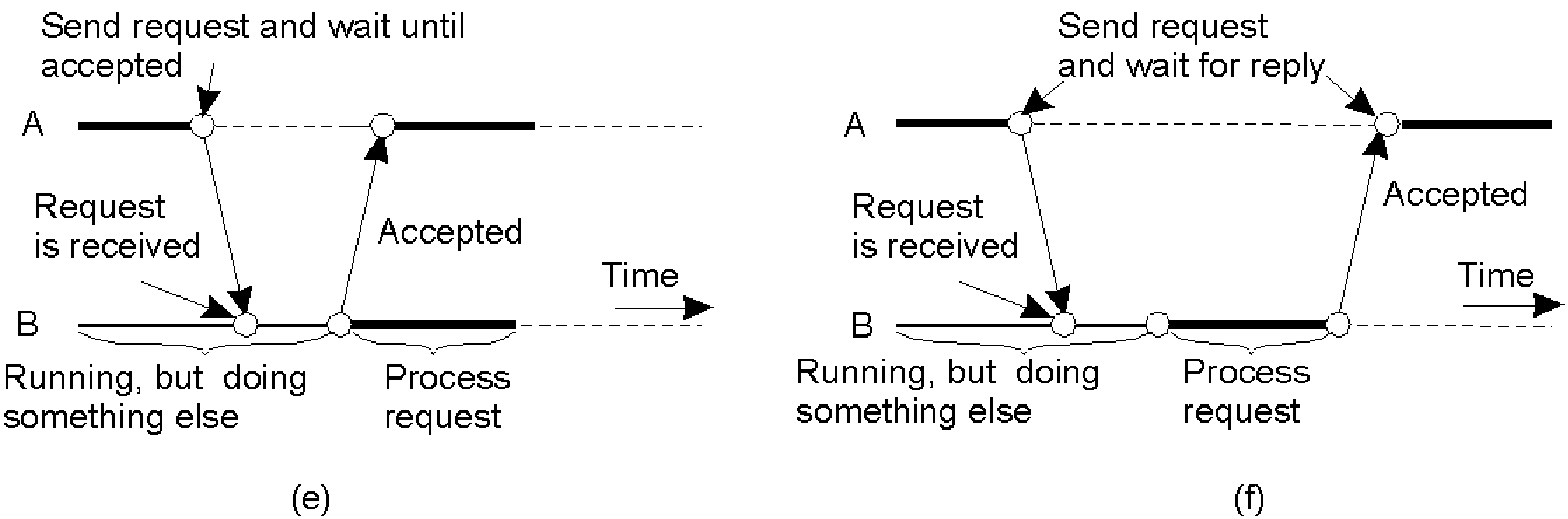
a) Persistent asynchronous communication
b) Persistent synchronous communication

Persistence and Synchronicity (4)



c) Transient asynchronous communication
d) Receipt-based transient synchronous communication

Persistence and Synchronicity (5)



e) Delivery-based transient synchronous communication at message delivery

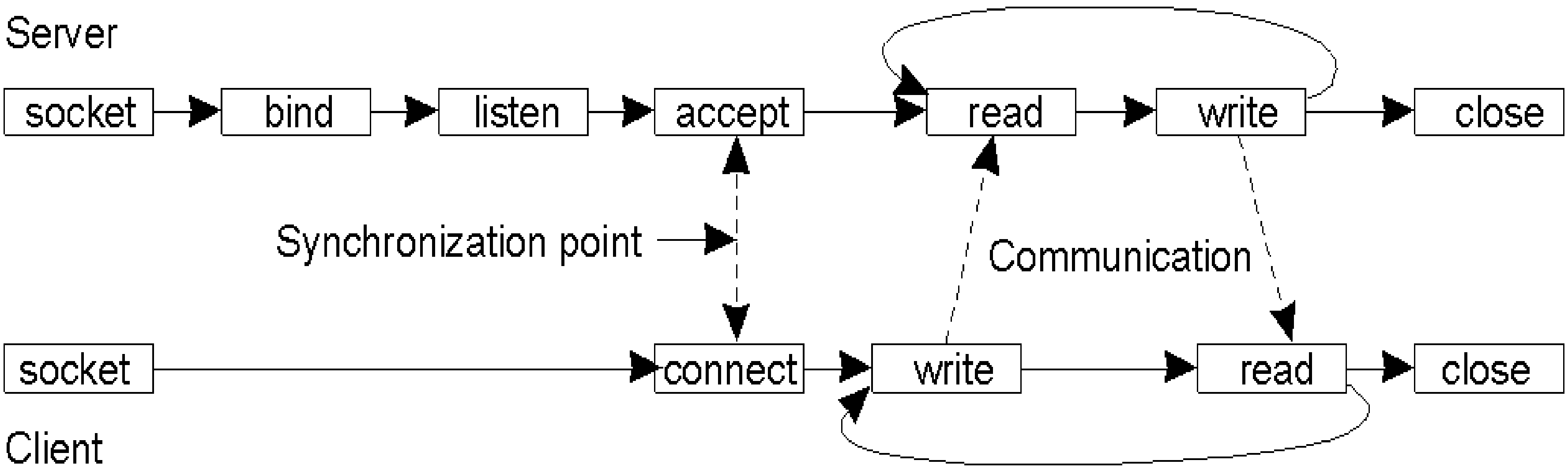
f) Response-based transient synchronous communication

Berkeley Sockets (1)

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

Socket primitives for TCP/IP.

Berkeley Sockets (2)



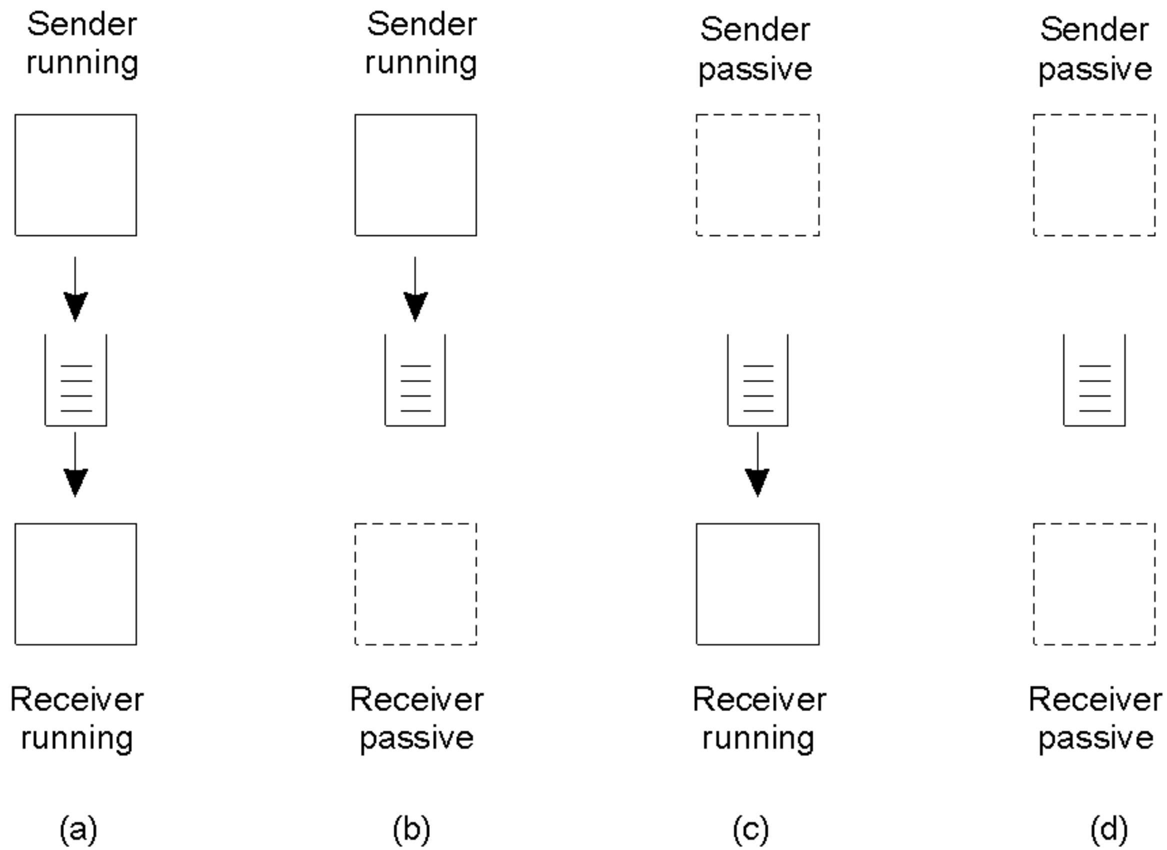
Connection-oriented communication pattern using sockets.

The Message-Passing Interface (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

Some of the most intuitive message-passing primitives of MPI.

Message-Queuing Model (1)



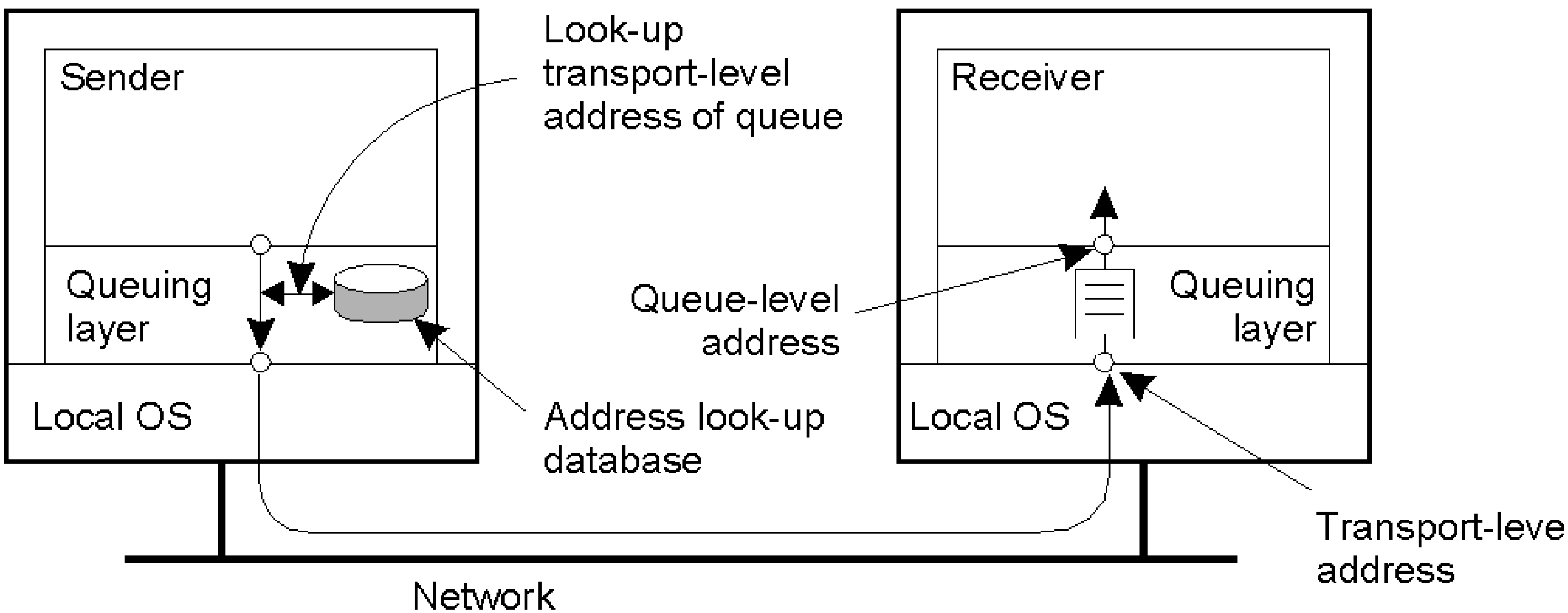
4 combinations for loosely-coupled communications w/ queues

Message-Queuing Model (2)

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.

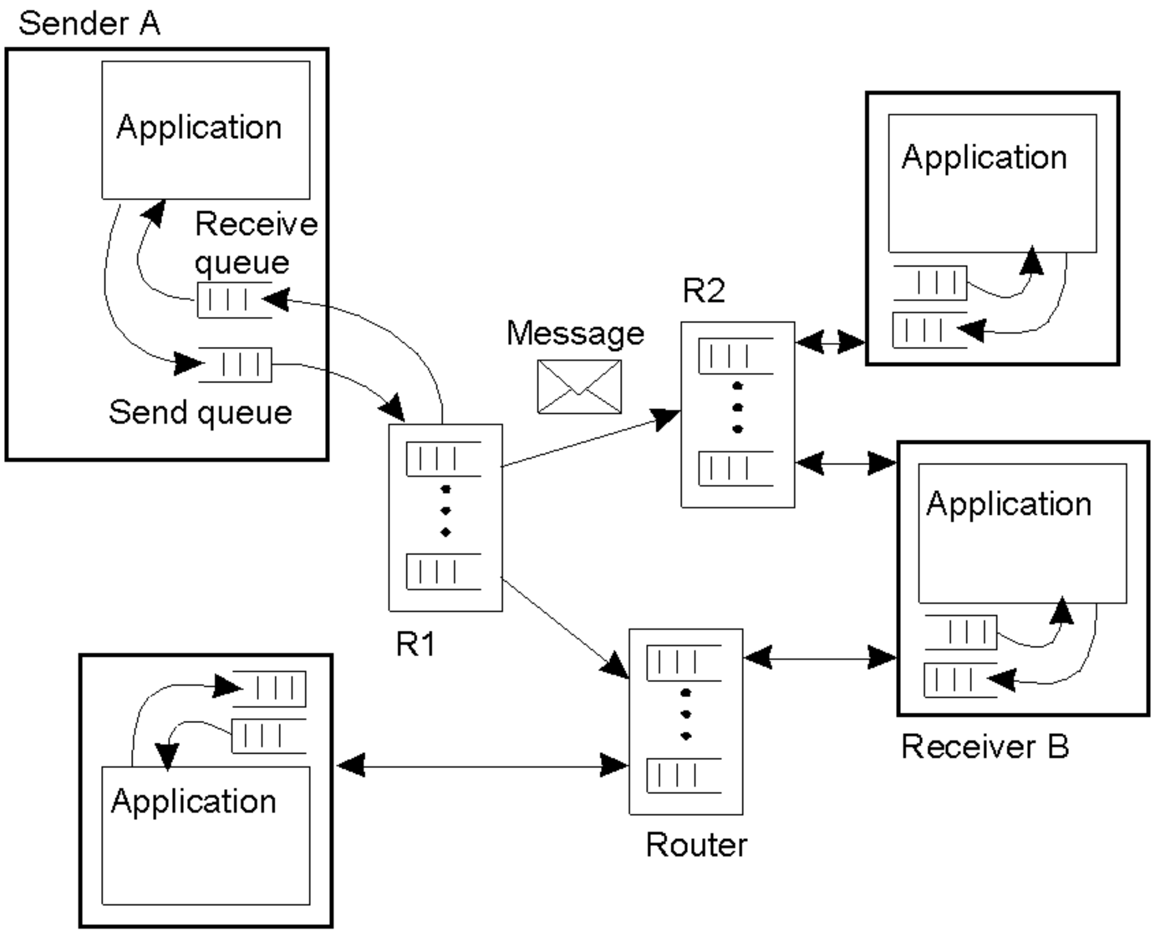
Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System (1)



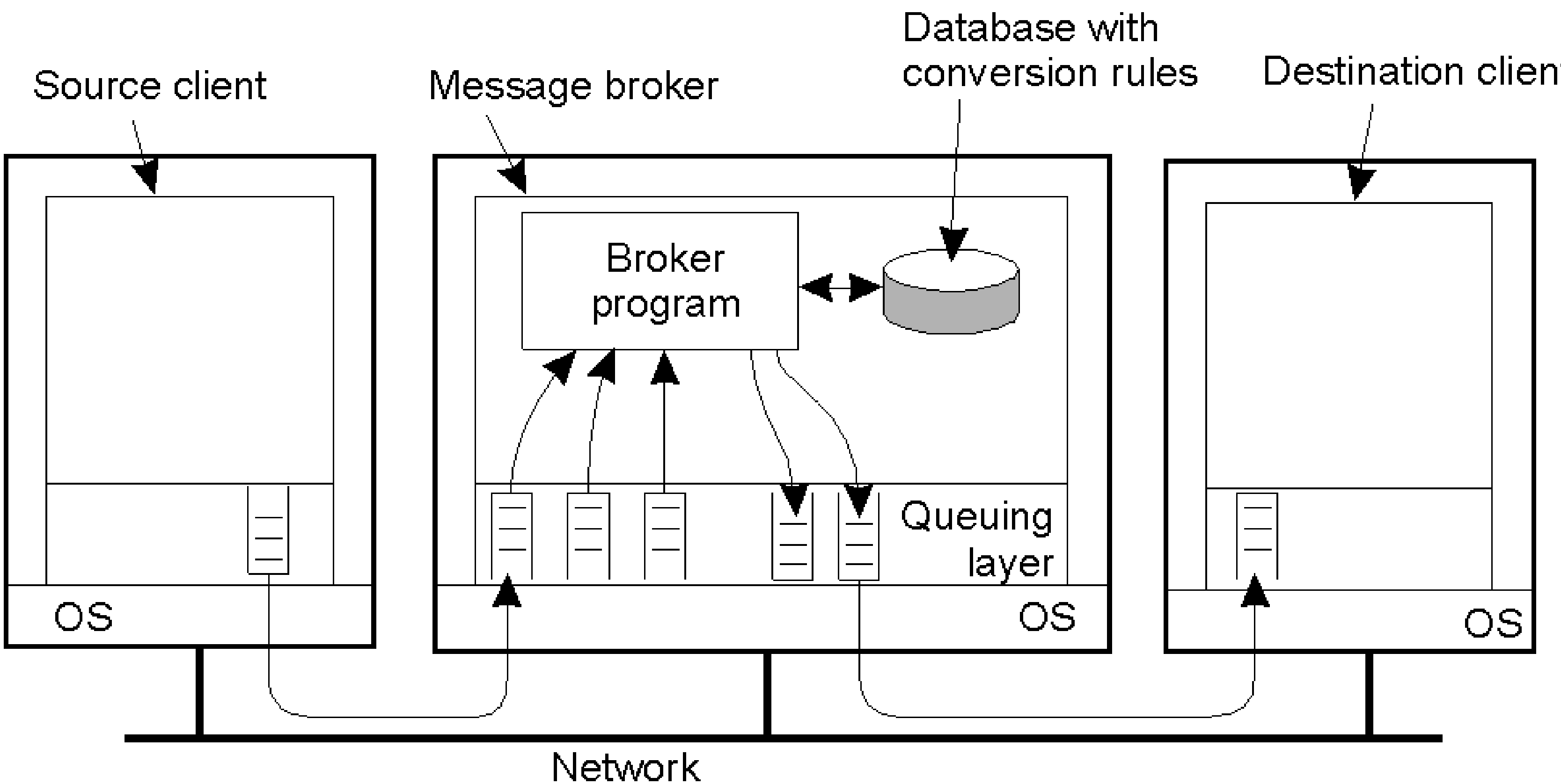
The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System (2)



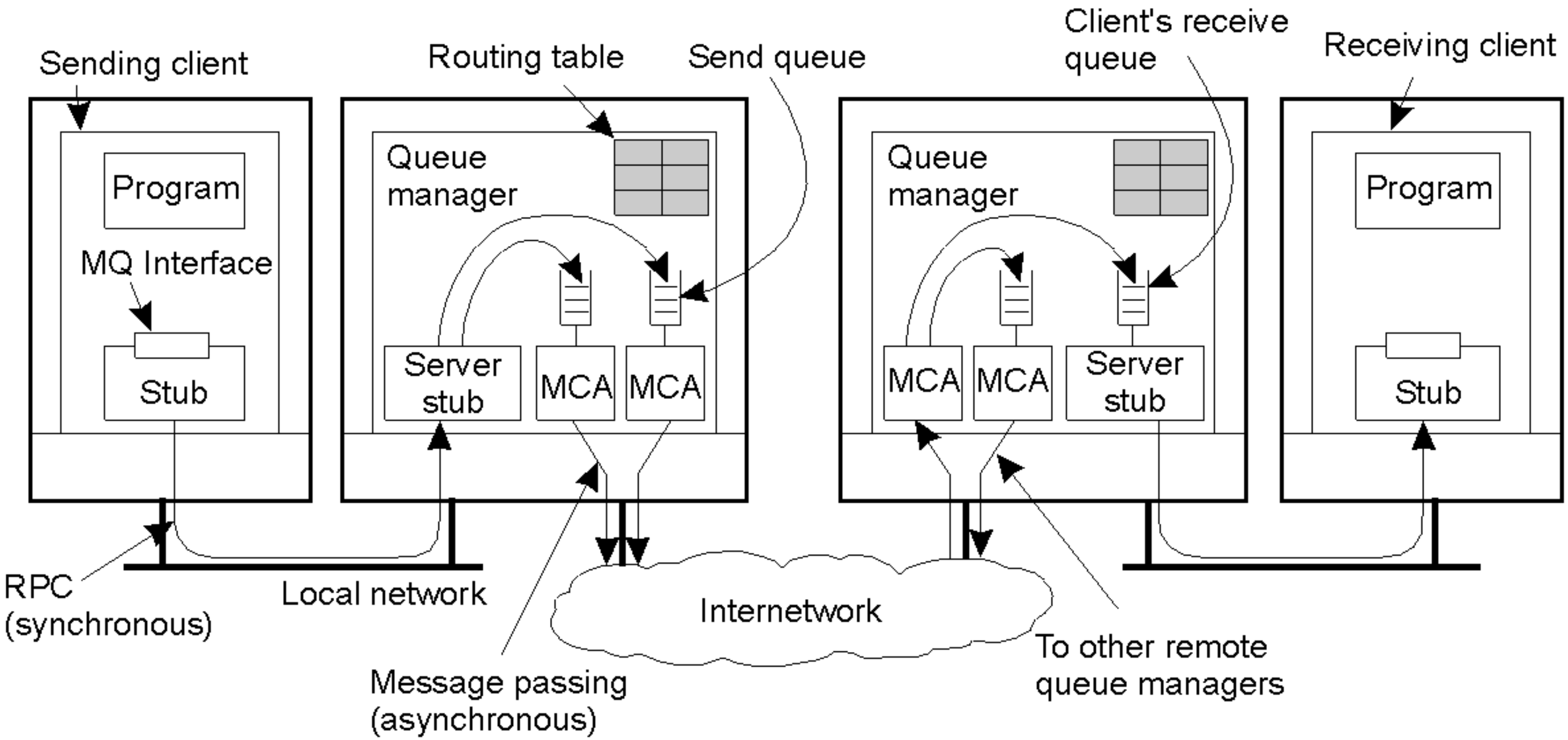
The general organization of a message-queuing system with routers.

Message Brokers



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

Example: IBM MQSeries



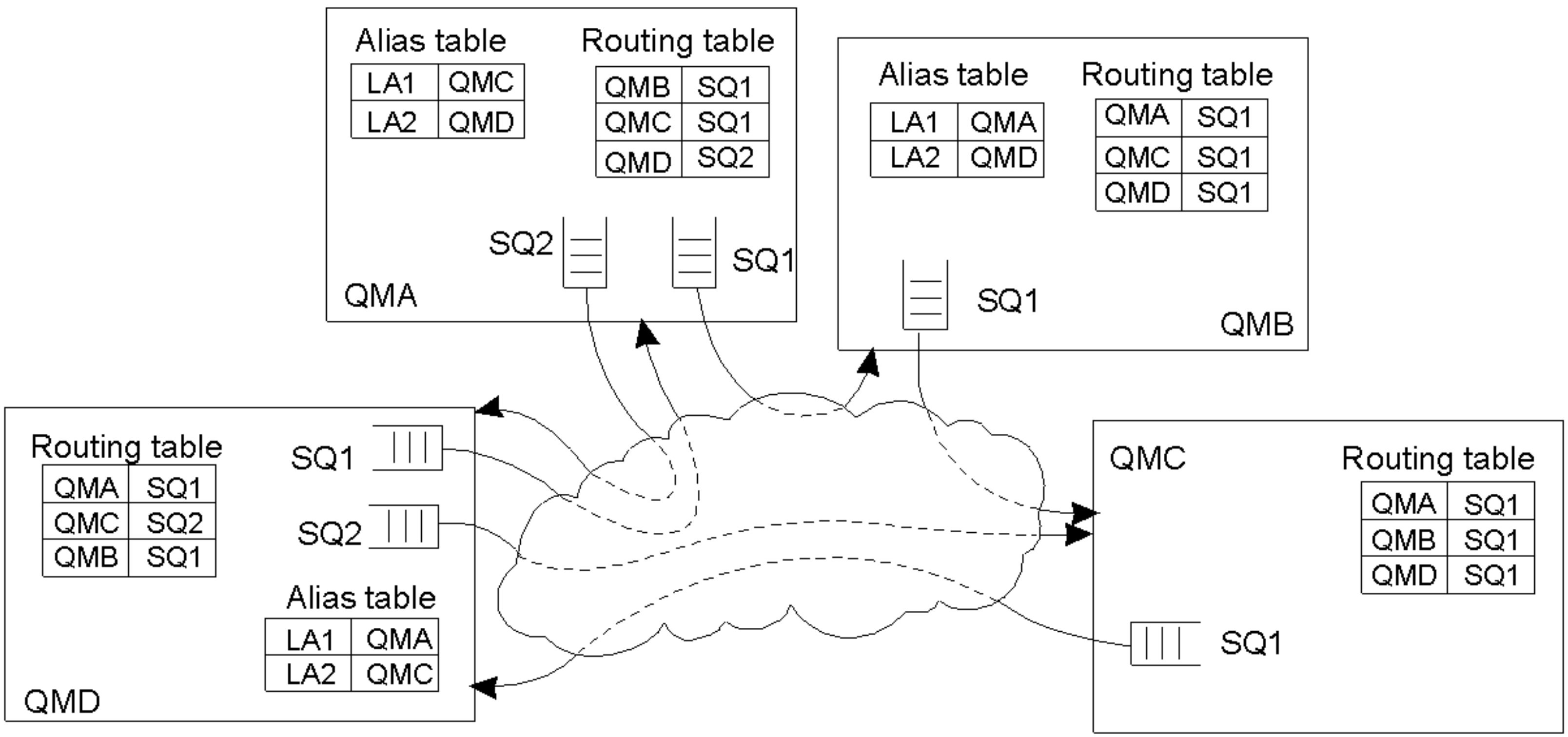
General organization of IBM's MQSeries message-queuing system.

Channels

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

Some attributes associated with message channel agents.

Message Transfer (1)



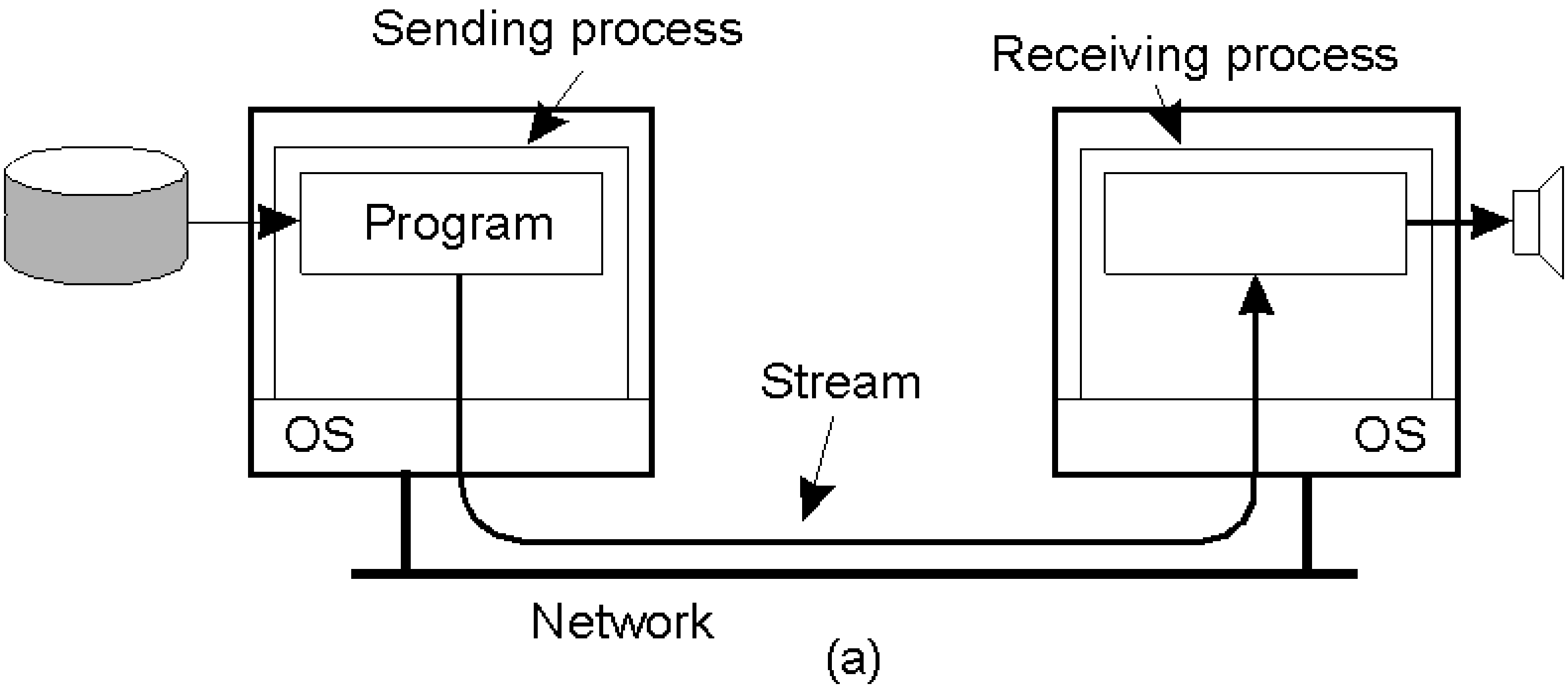
The general organization of an MQSeries queuing network using routing tables and aliases.

Message Transfer (2)

Primitive	Description
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

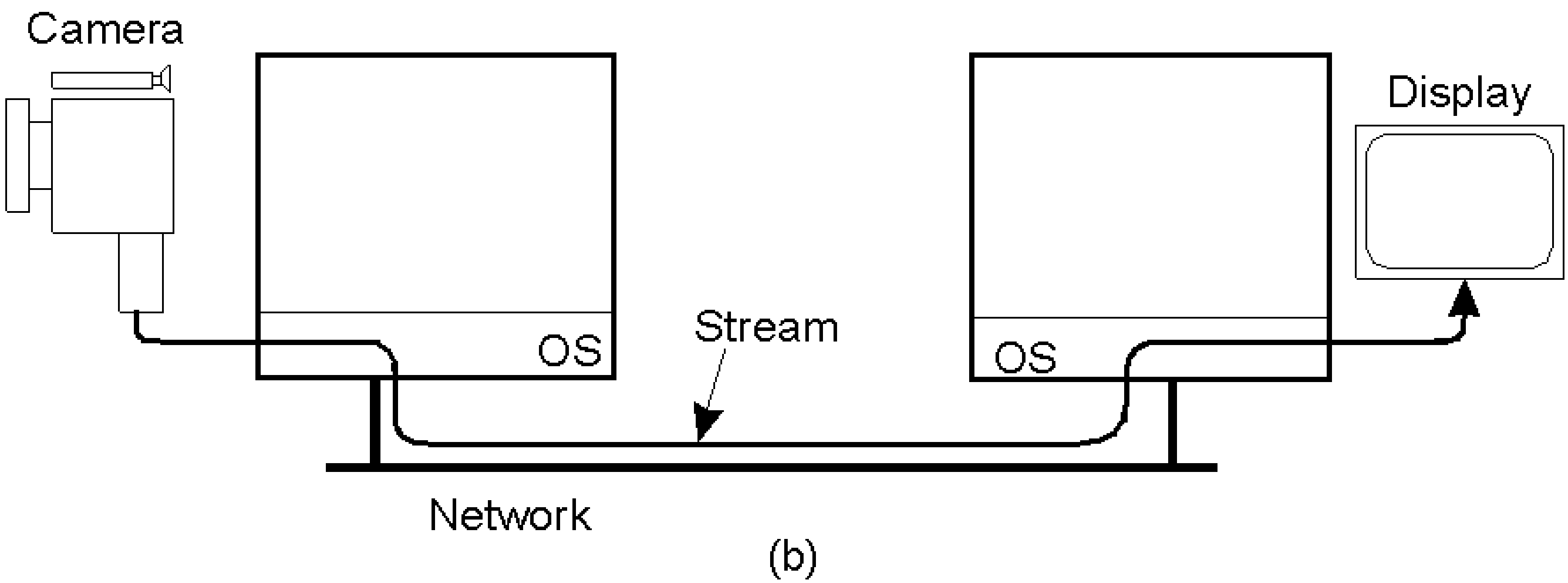
Primitives available in an IBM MQSeries MQI

Data Stream (1)



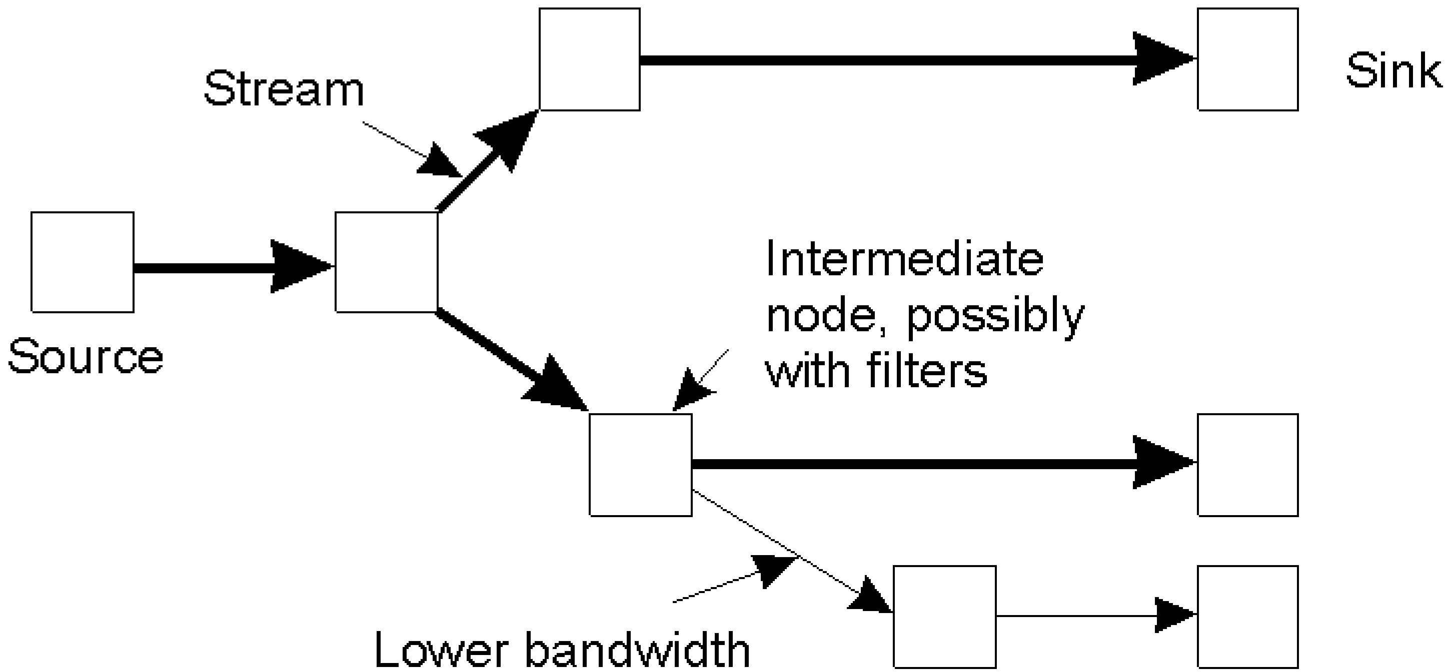
Setting up a stream between two processes across a network.

Data Stream (2)



Setting up a stream directly between two devices.

Data Stream (3)



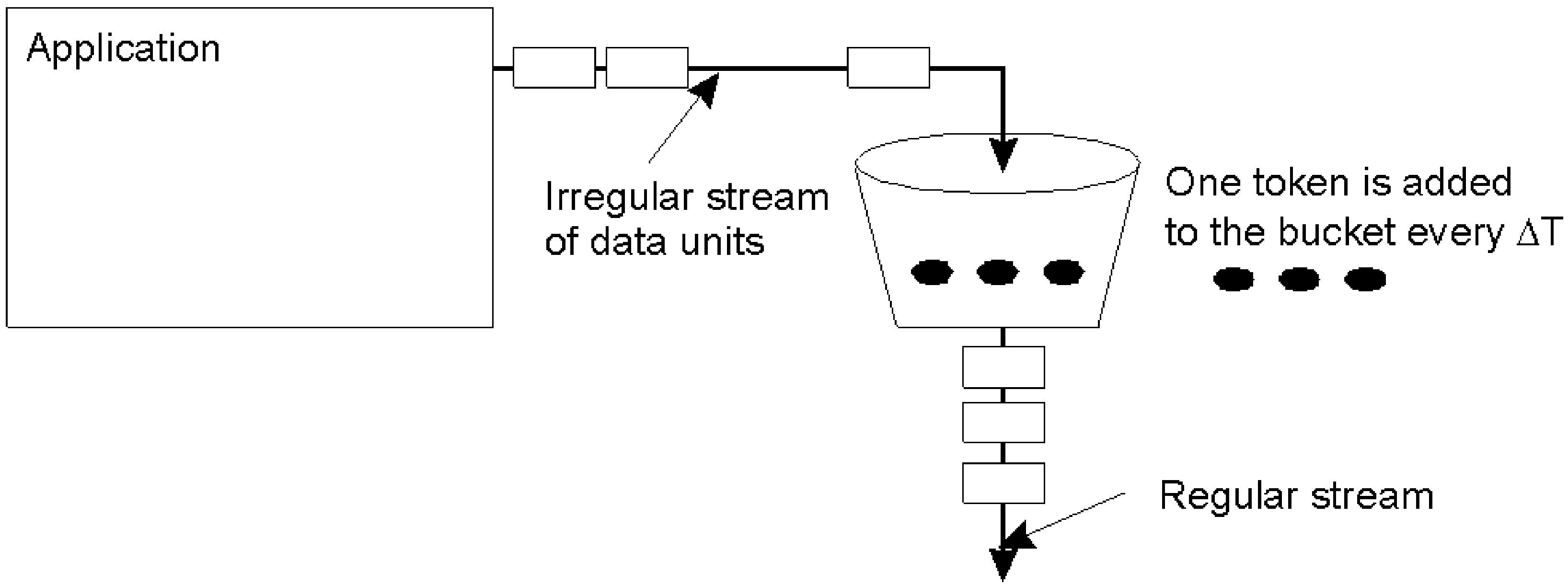
An example of multicasting a stream to several receivers.

Specifying QoS (1)

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

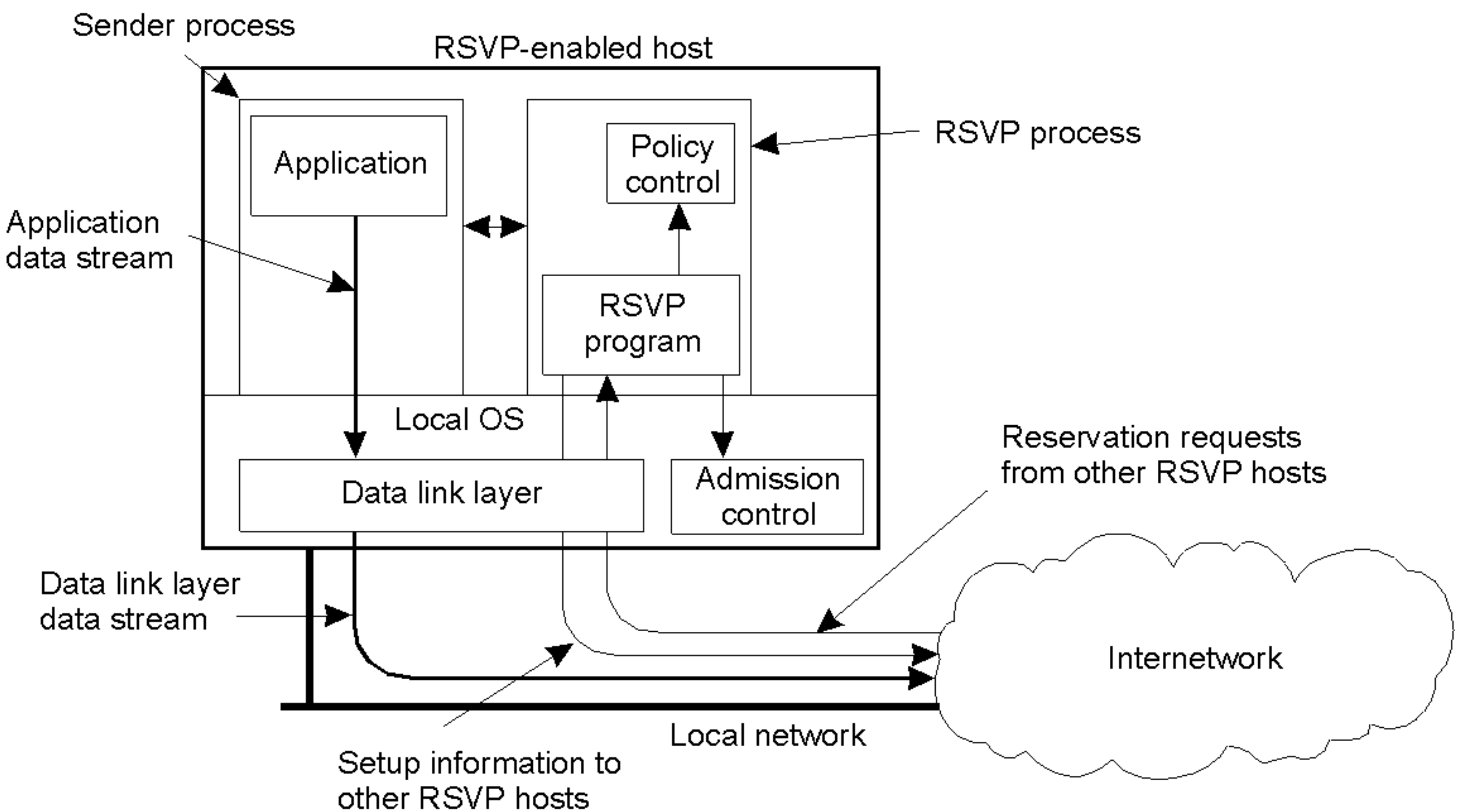
A flow specification.

Specifying QoS (2)



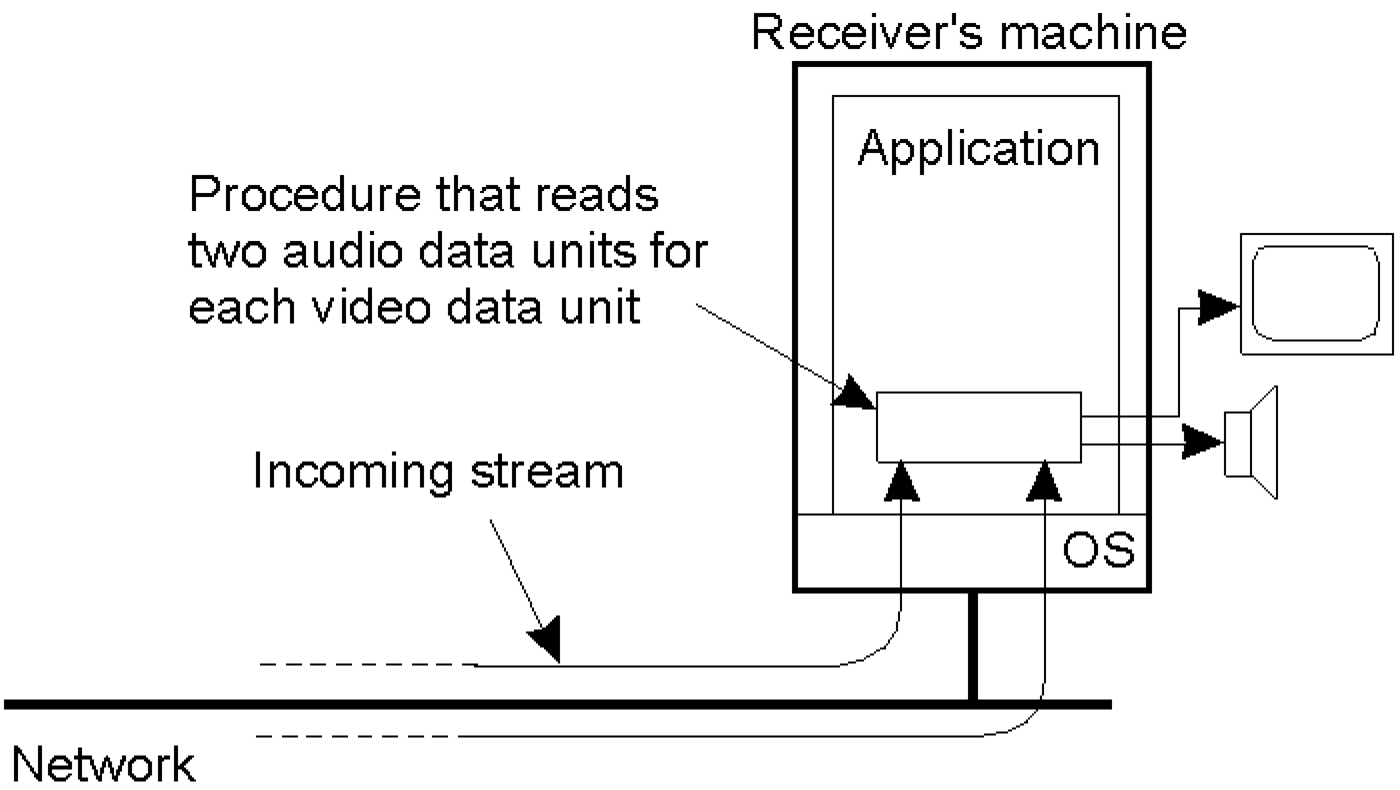
The principle of a token bucket algorithm.

Setting Up a Stream



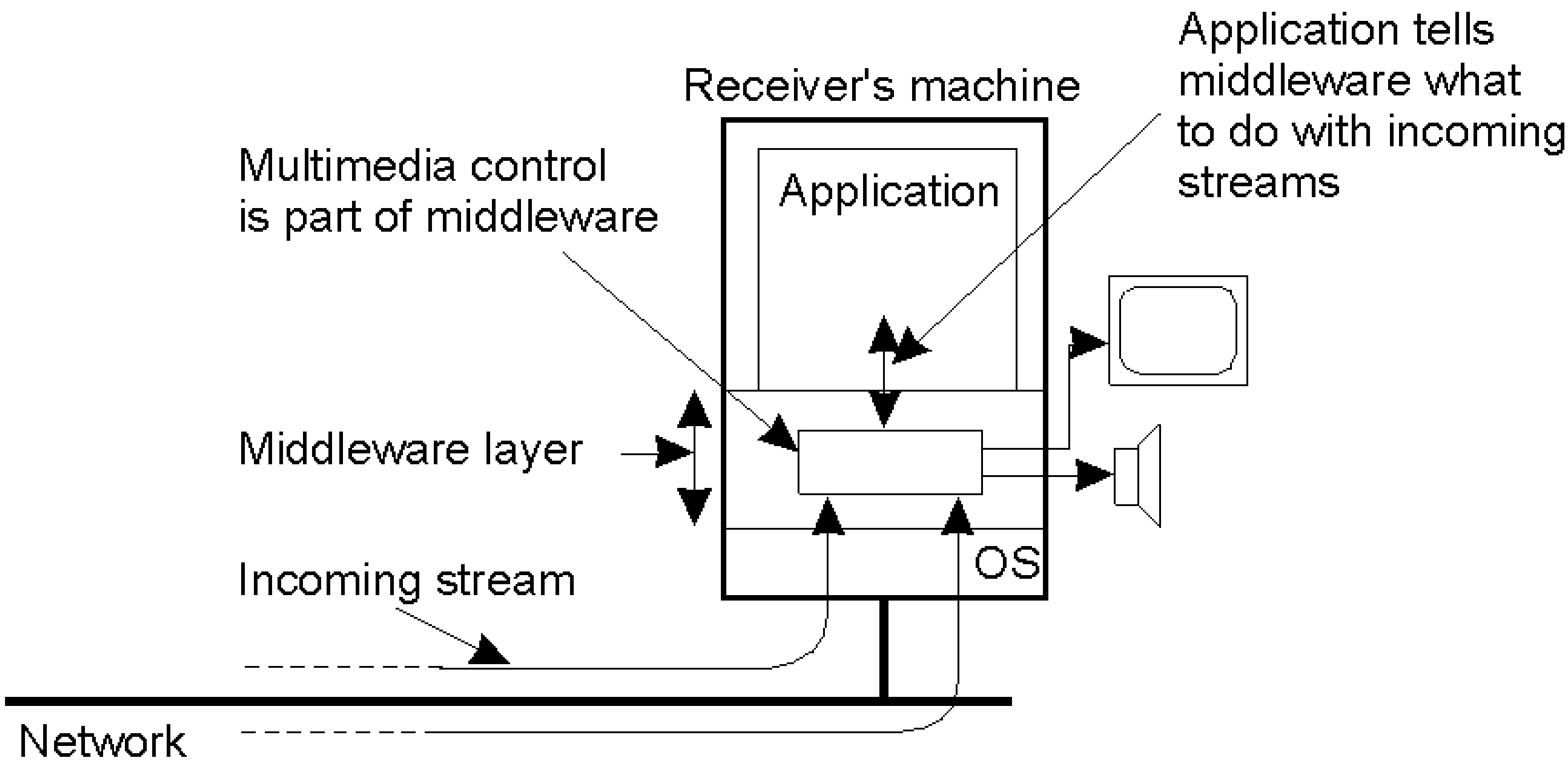
The basic organization of RSVP for resource reservation in a distributed system.

Synchronization Mechanisms (1)



The principle of explicit synchronization on the level data units.

Synchronization Mechanisms (2)



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

Multicasting

Transport or application level

Distribution trees

Gossip

Level

Multicast in Network Protocols:

- Creating *communication paths*
- Enormous management effort
- ISP reluctant to implement

Multicast at the Application Level

- Has become possible in the age of P2P
- *Communication paths* as *overlay networks*
- Two techniques:
  - explicit communication paths
  - gossiping

Application-Level Multicasting

Basic idea: nodes organized in an *overlay network*

N.B.: *routers* are not part of the overlay network!

Basic design element: overlay network construction

Two approaches are possible:

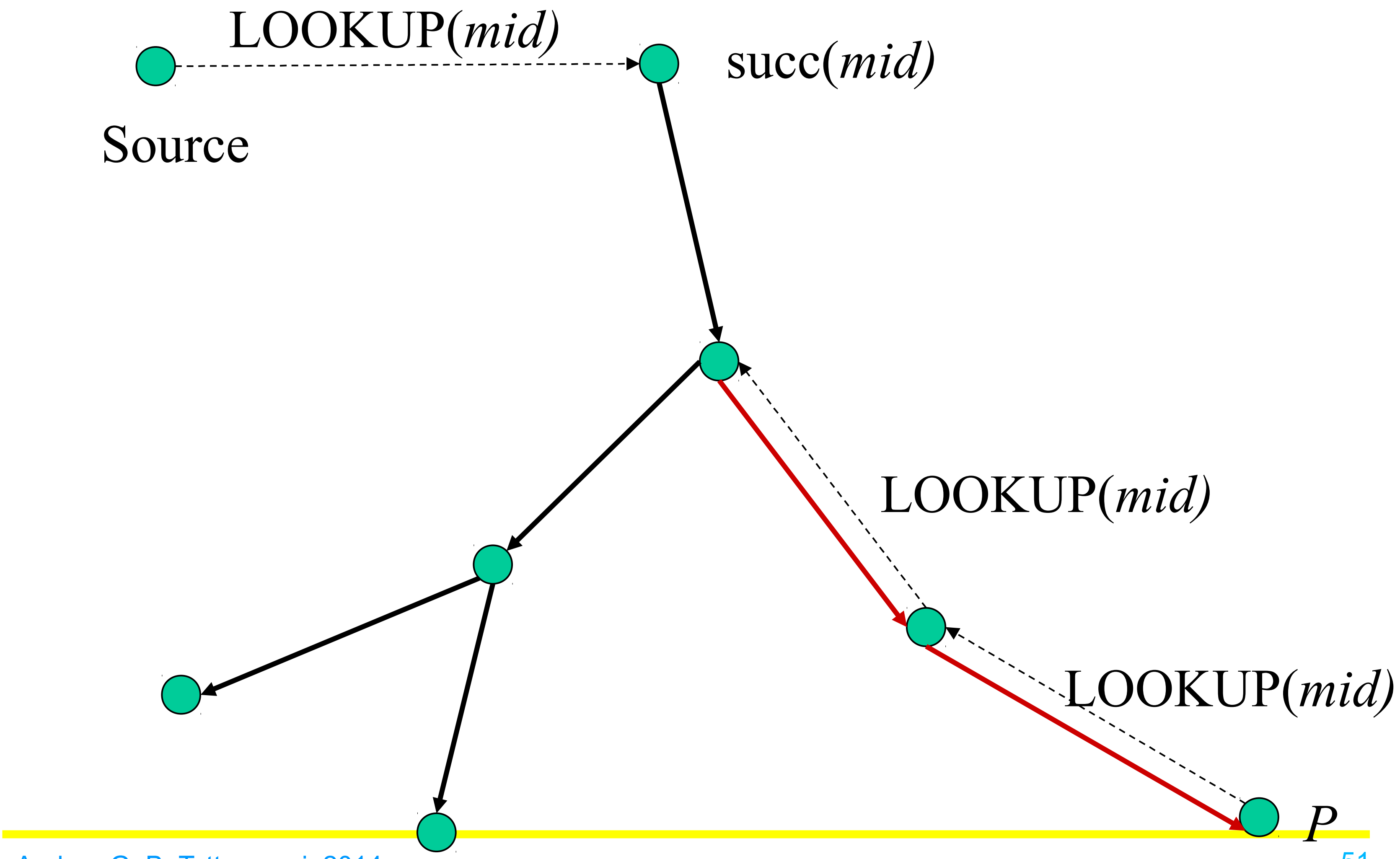
- Distribution tree layout
- Mesh layout (multiple paths are possible)

Multicast Tree Construction

Method used in the CHORD system (DHT):

- The node that initiates a multicast session generates a 160 bit random identifier, *mid*;
- Look succ(*mid*) up and make it the tree root;
- If a node *P* wishes to “register” to the tree, it sends a message to succ(*mid*), which will go through other nodes
- The nodes traversed either are already in the tree, or they become *forwarders* on behalf of *P*.

Multicast Tree Construction



Quality of a Multicast Tree

Link Stress:

- How many times the same packet goes through the same link

Stretch or relative delay penalty (RDP)

- $d_{\text{overlay}}(A, B)/d_{\text{phis}}(A, B) \geq 1$

Cost of the Tree

- A global measure, relevant to controlling the resources used by multicast communication

Information Diffusion Models

Epidemic Behavior

Information spreads “by contagion”

- *Infected node* = has the data that have to be spread
- *Susceptible node* = does not have the data
- *Removed node* = has the data but it does not spread them

Fully Local Techniques

Anti-Entropy

P randomly picks another node *Q*

Three possible approaches:

1. *Push: P* sends its data to *Q*
2. *Pull: P* requests data from *Q*
3. *Push-Pull: P and Q* exchange data

Push approach is inefficient

Push-pull approach is optimal

All nodes get updated in $O(\log N)$ "rounds".

Gossiping

When node P gets to know some new information, it starts contacting other arbitrary nodes (random, neighbors, ...) to tell them.

Every time a contacted node turns out to already know, with probability $1/k$, P decides to give up "gossiping" and becomes "removed".

Problem: the fraction of "susceptible" nodes tends to

$s = \exp[-(k + 1)(1 - s)]$

Data Elimination

A problem, because, if data are removed, a node becomes again susceptible

“Deat Certificate” technique

These to have to be eliminated, after a while:

“Inactive Death Certificates”

Thank you for your attention