Concurrency and Parallelism

Master 1 International

Andrea G. B. Tettamanzi

Université de Nice Sophia Antipolis

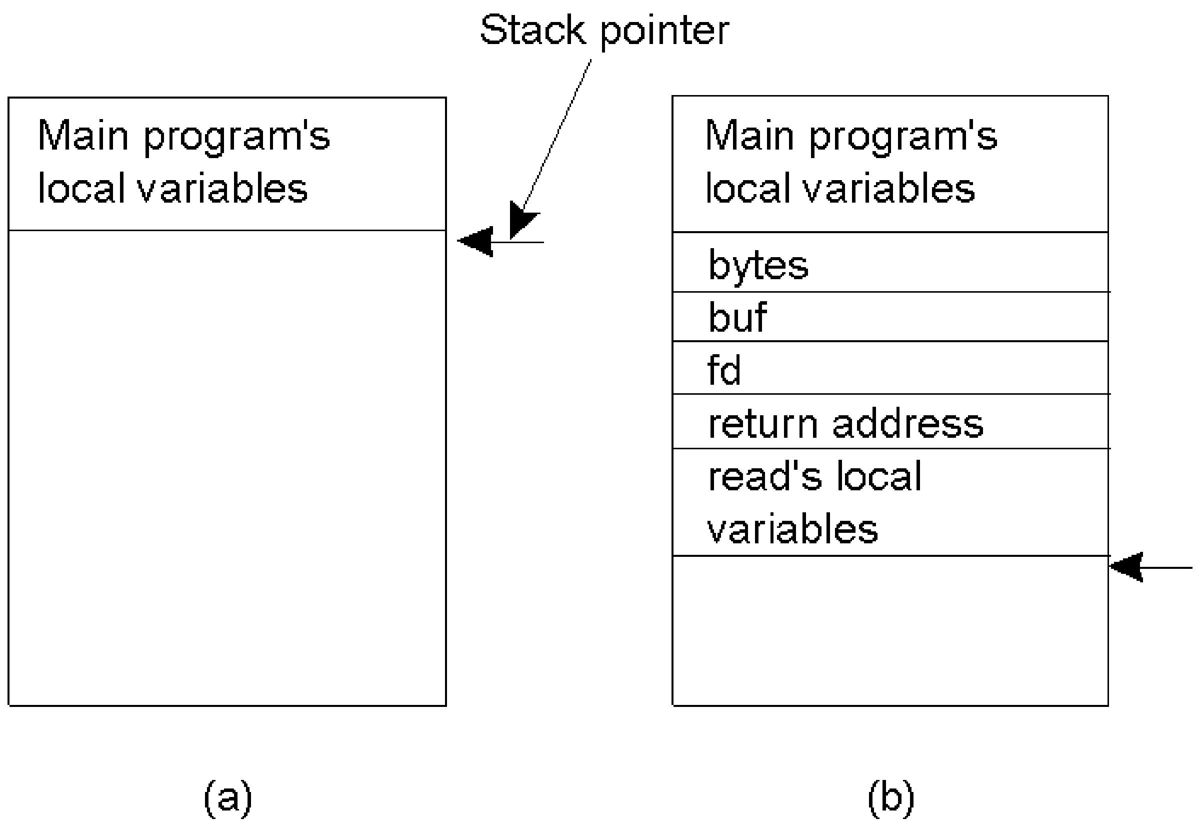
Département Informatique

andrea.tettamanzi@unice.fr

Lecture 2

Communication and Synchronization

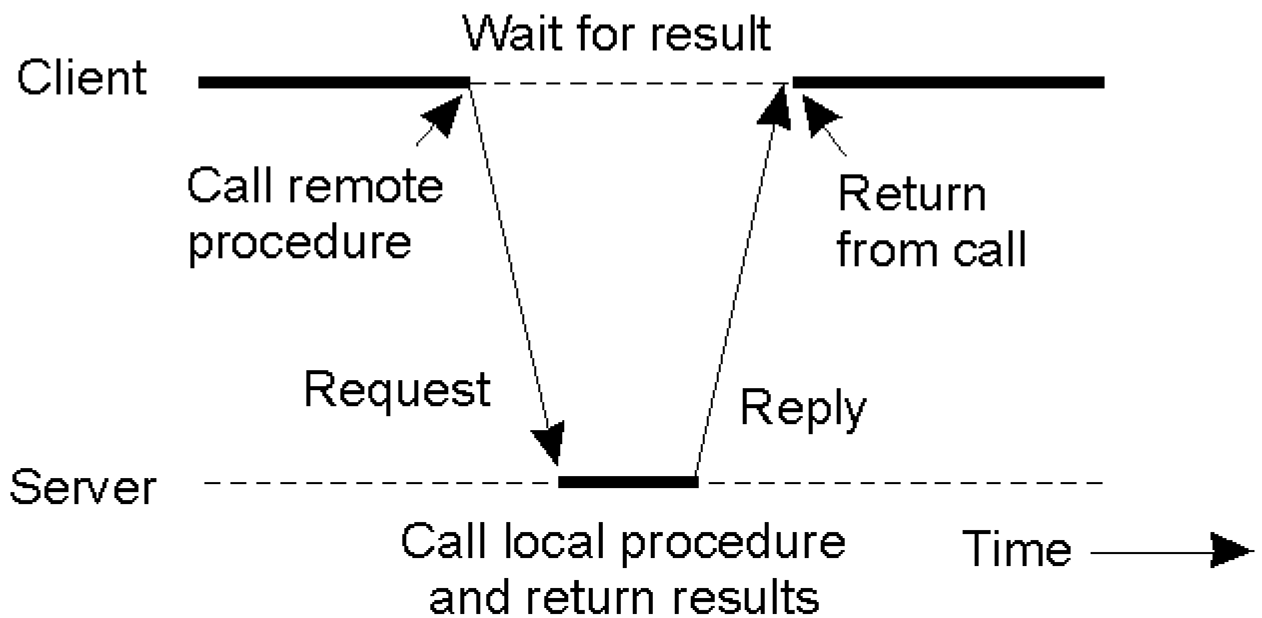
Conventional Procedure Call



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

b) The stack while the called procedure is active

Client and Server Stubs

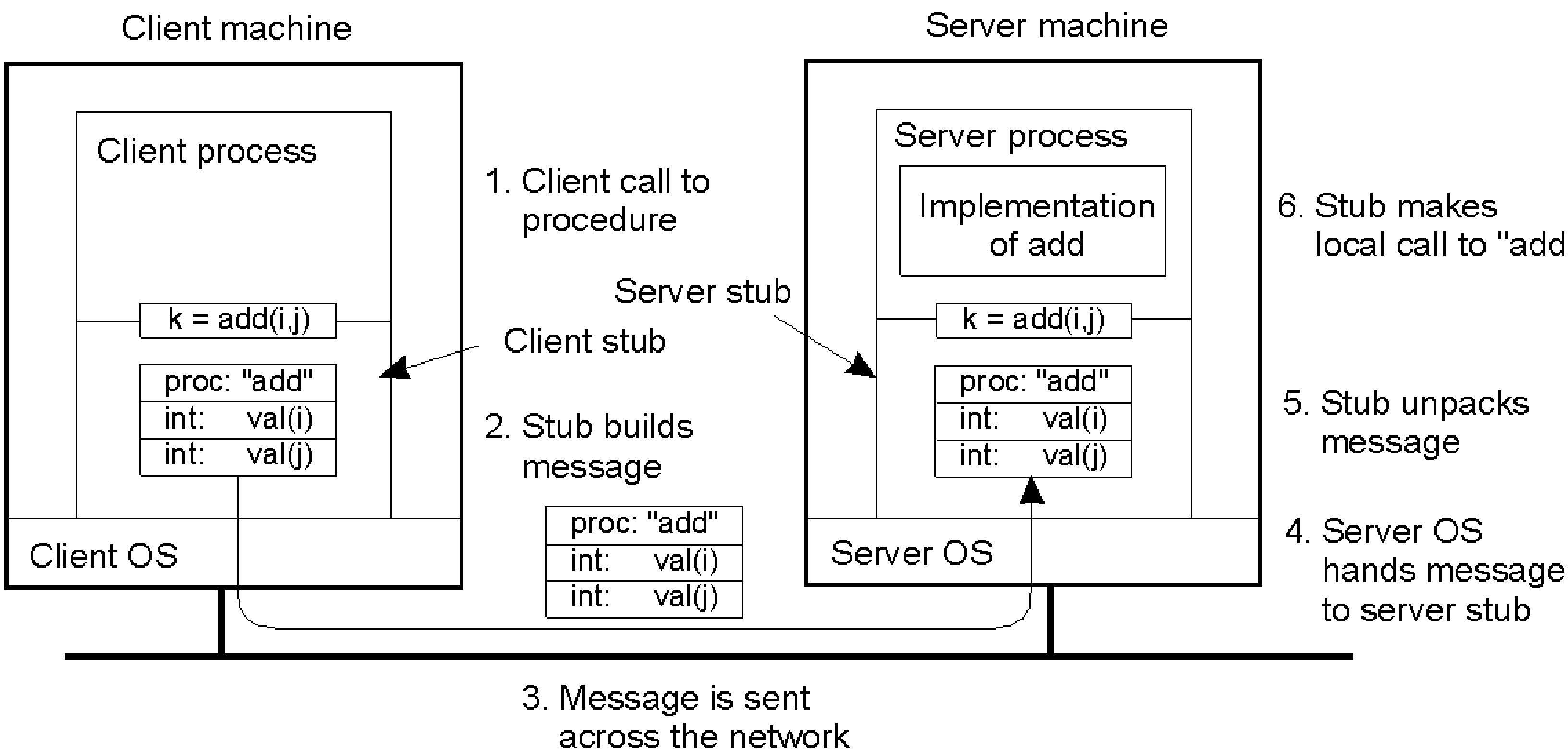


Principle of RPC between a client and server program.

Steps of a Remote Procedure Call

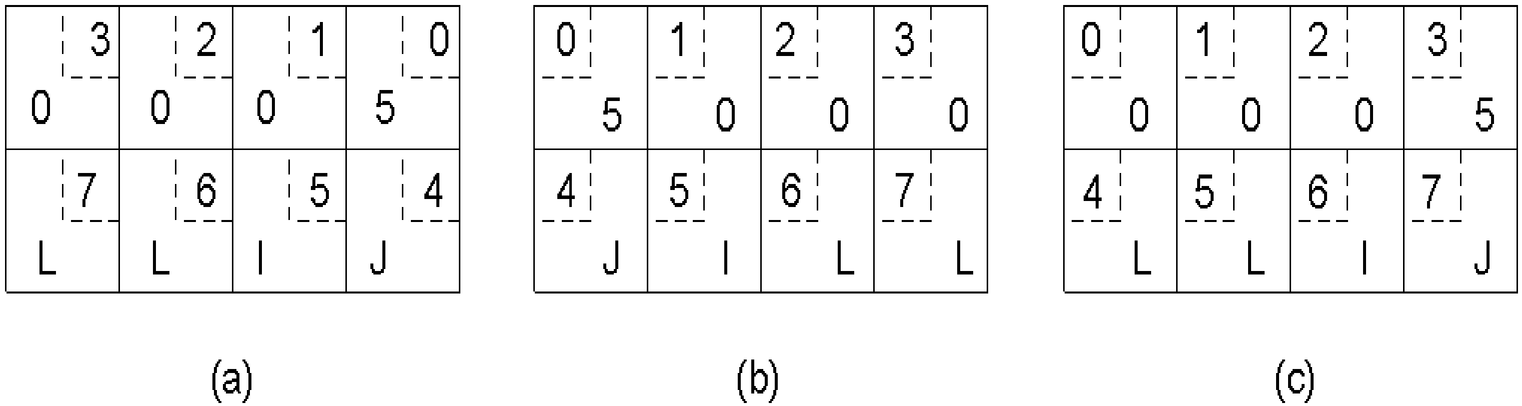
1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it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



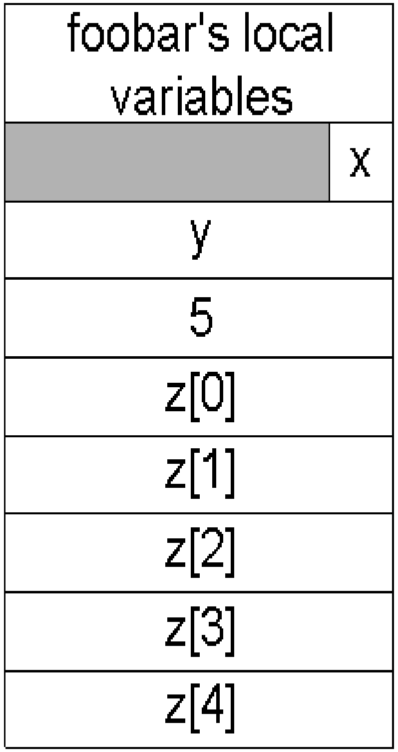
a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

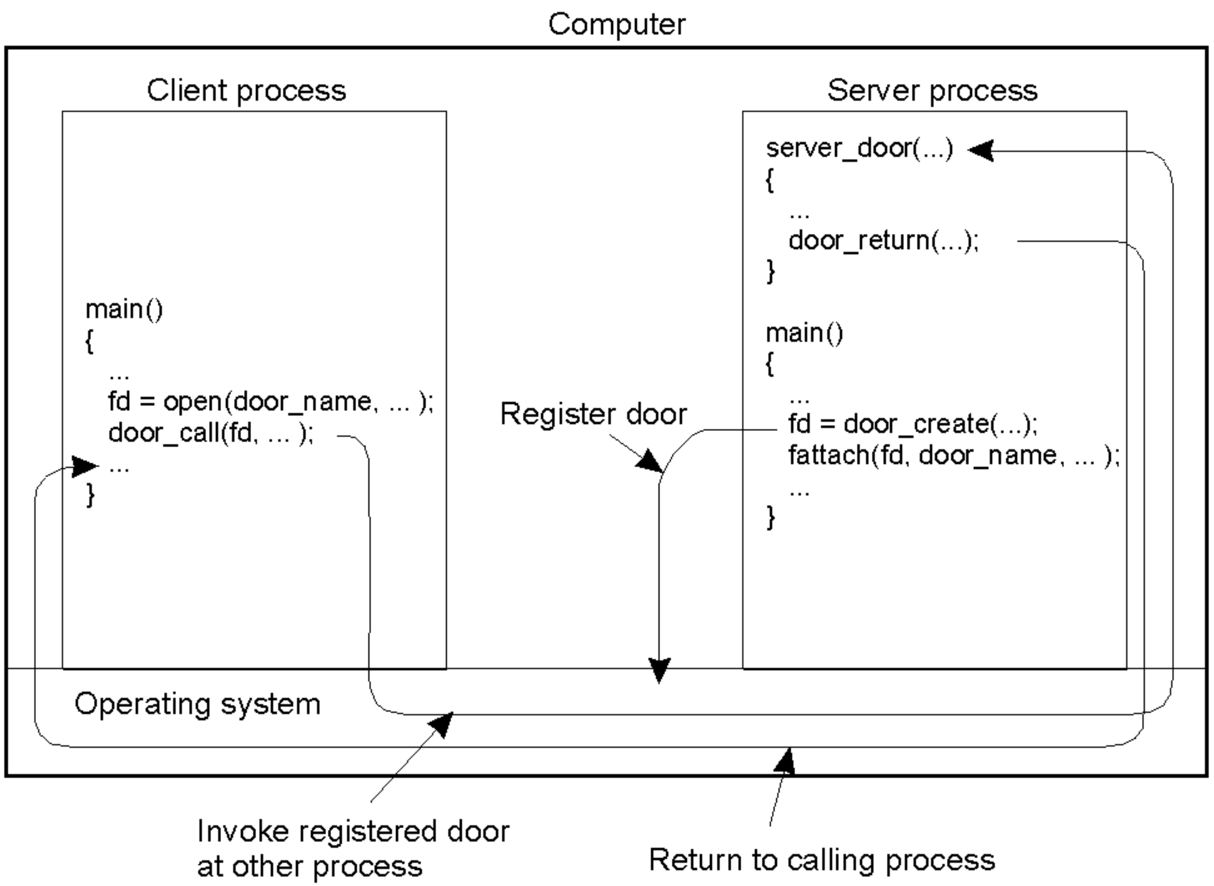
```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



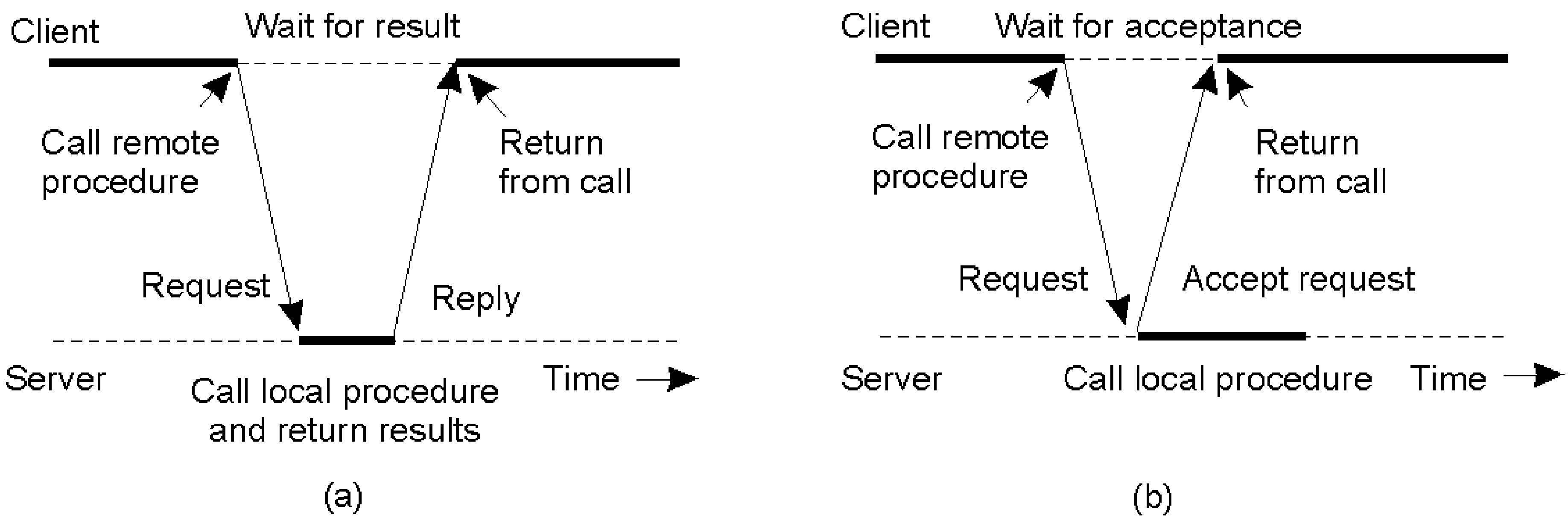
(b)

Doors



The principle of using doors as IPC mechanism.

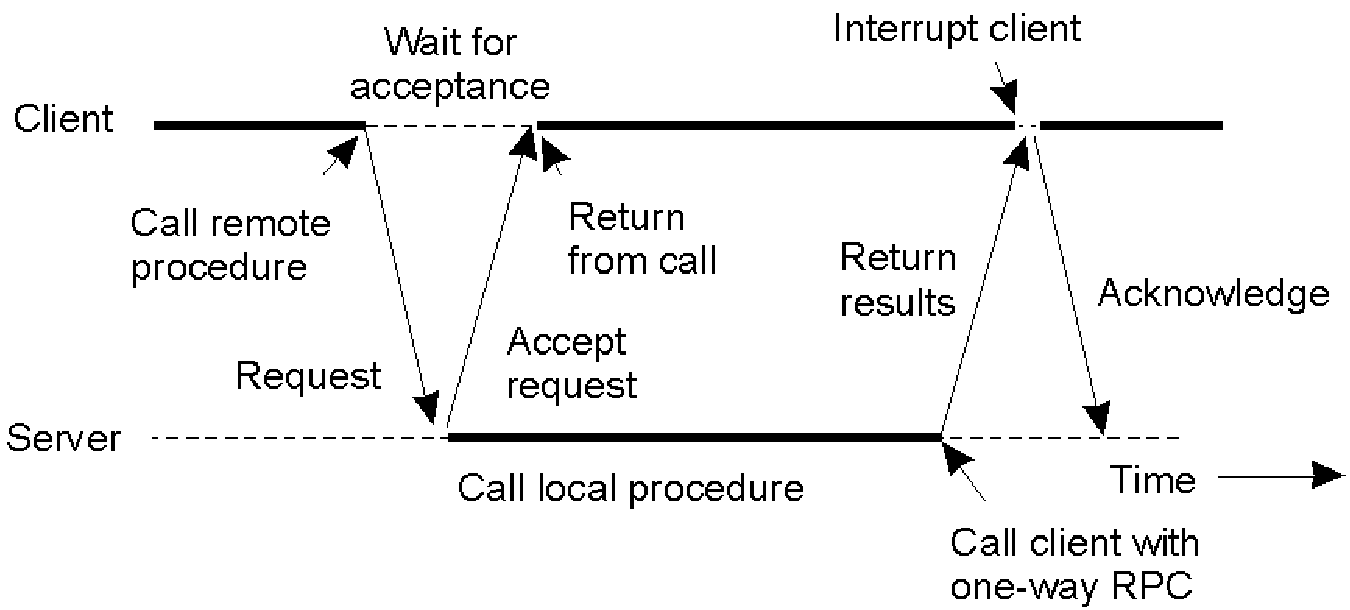
Asynchronous RPC (1)



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

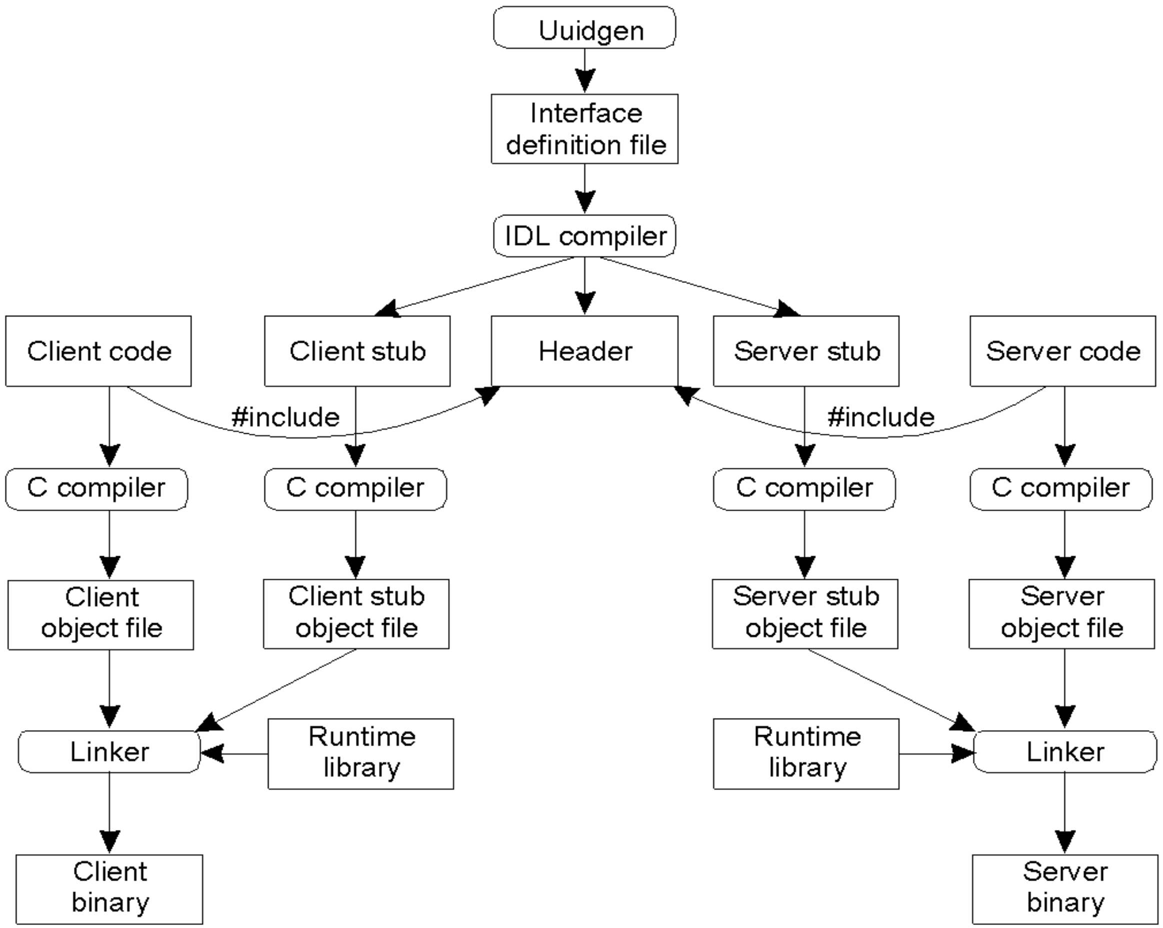
b) The interaction using asynchronous RPC

Asynchronous RPC (2)



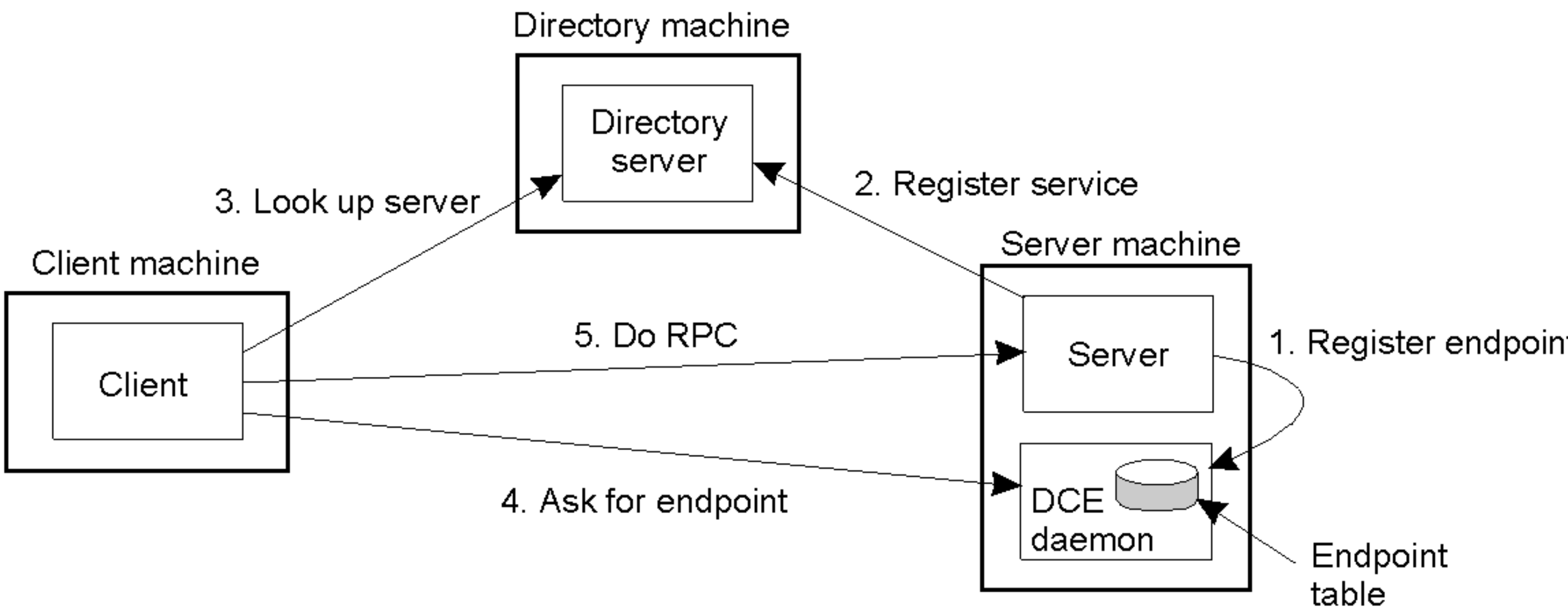
A client and server interacting through two asynchronous RPCs

Writing a Client and a Server



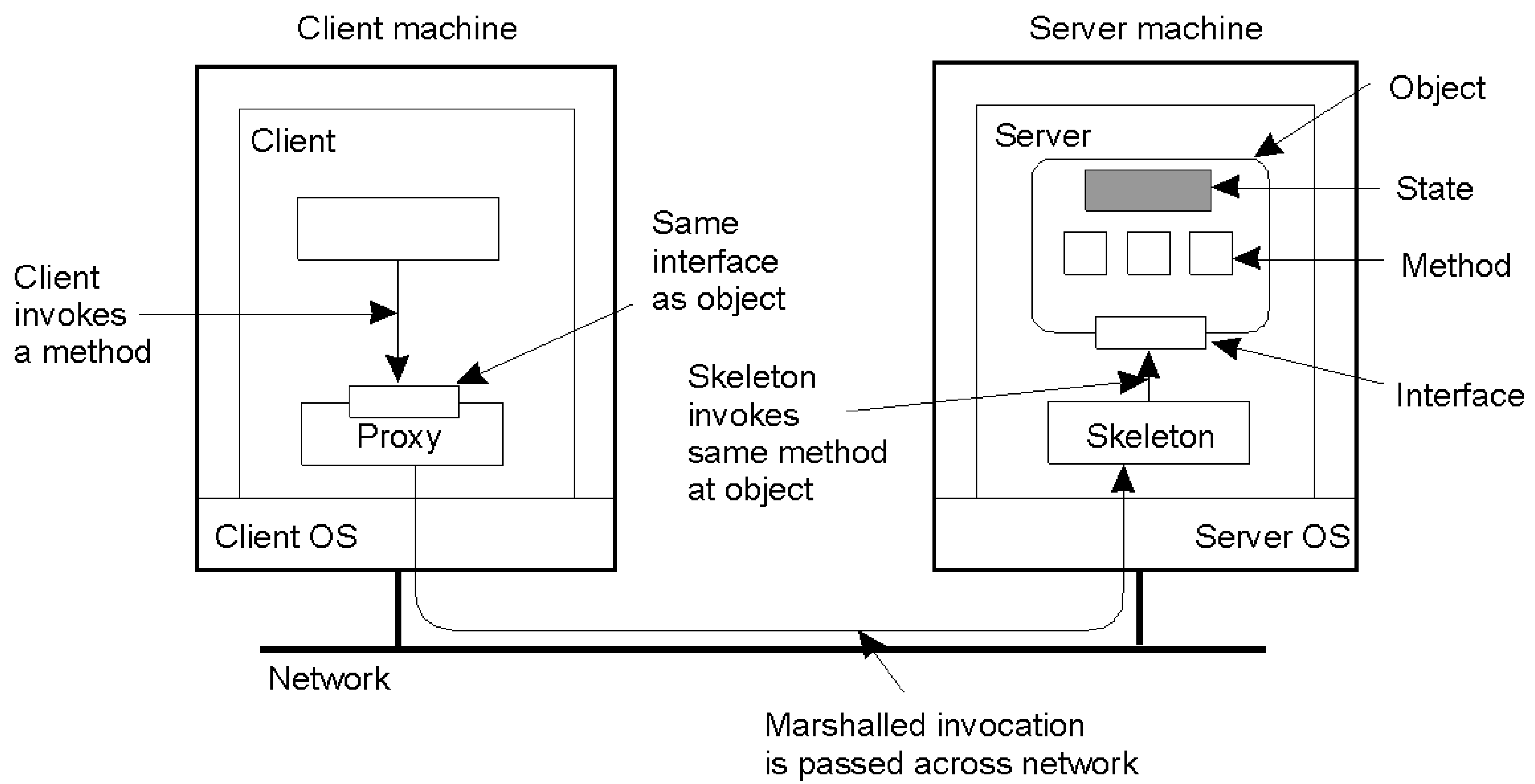
The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server



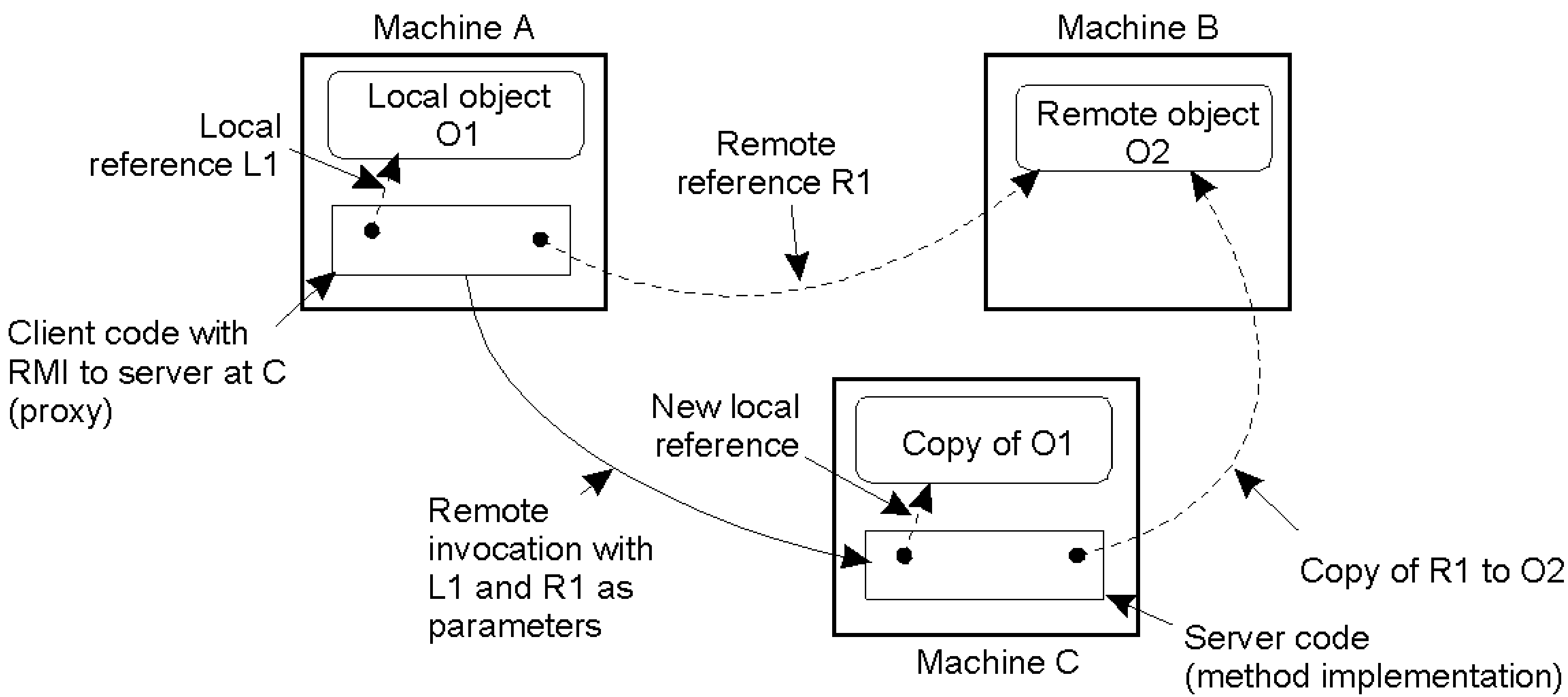
Client-to-server binding in DCE.

Distributed Objects



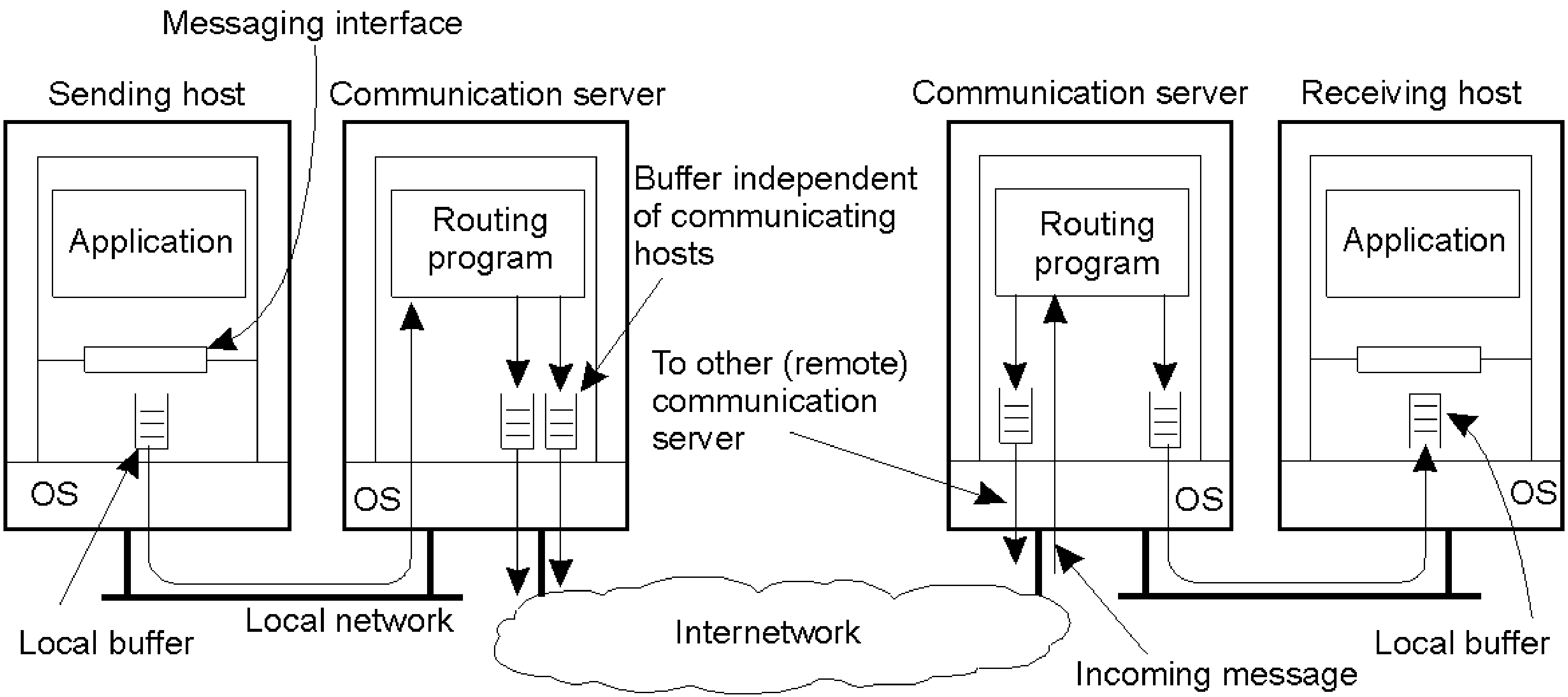
Common organization of a remote object with client-side proxy.

Parameter Passing



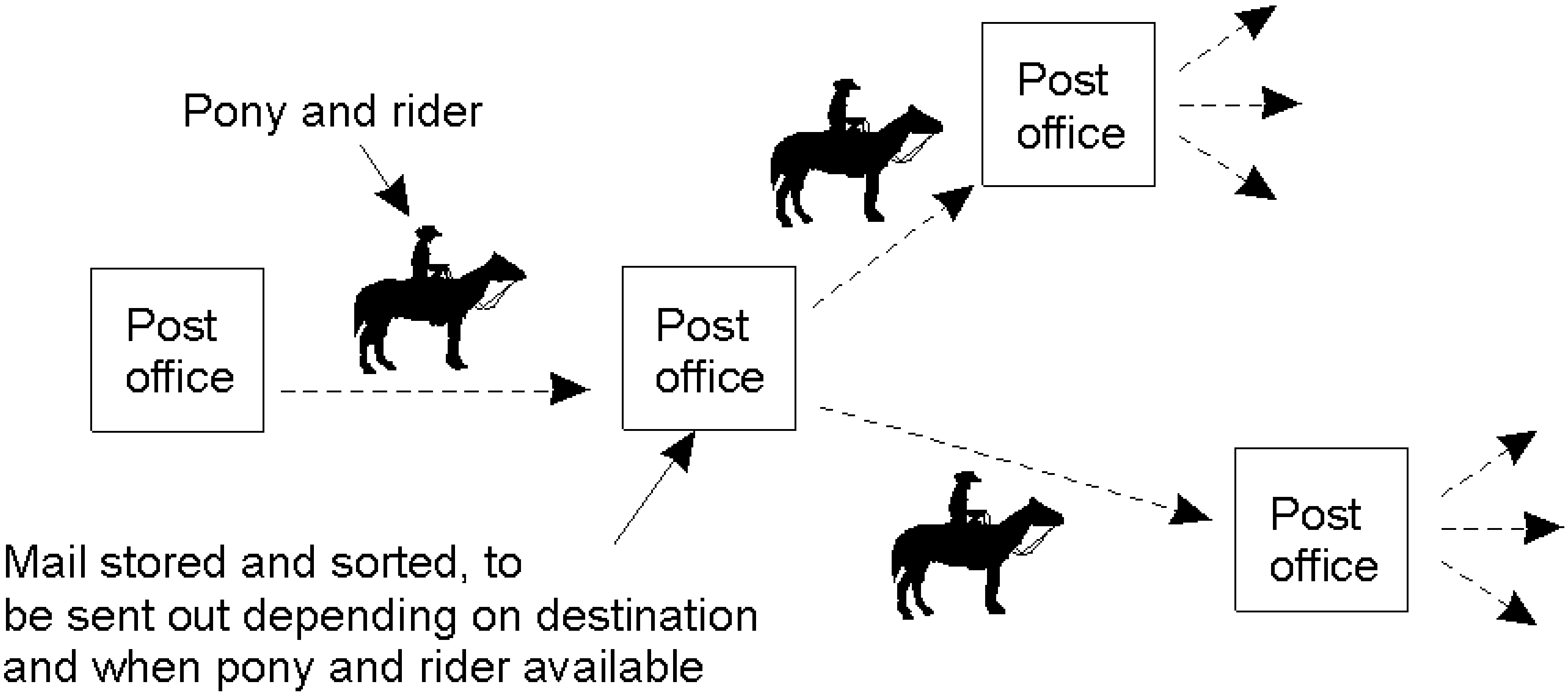
The situation when passing an object by reference or by value.

Persistence and Synchronicity in Communication (1)



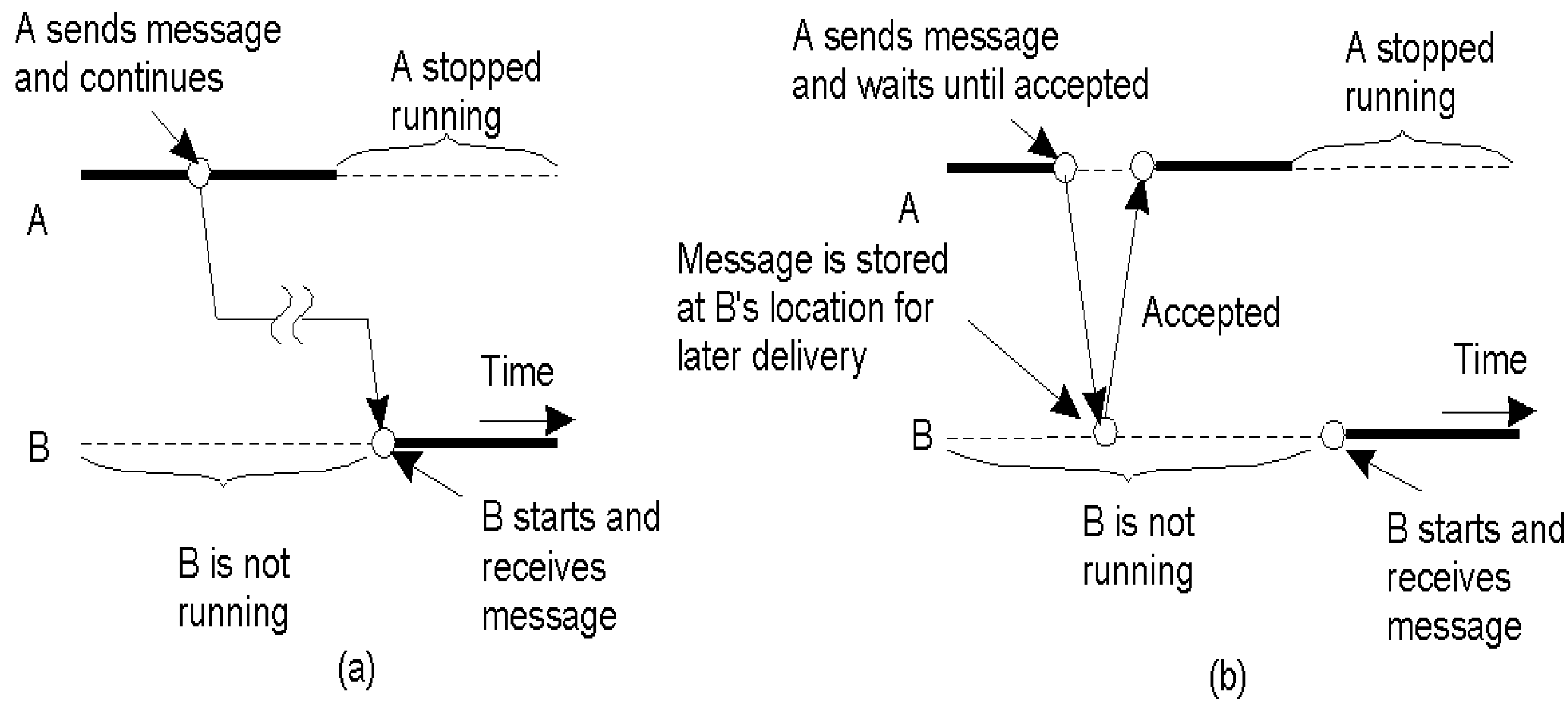
General organization of a communication system in which hosts are connected through a network

Persistence and Synchronicity in Communication (2)



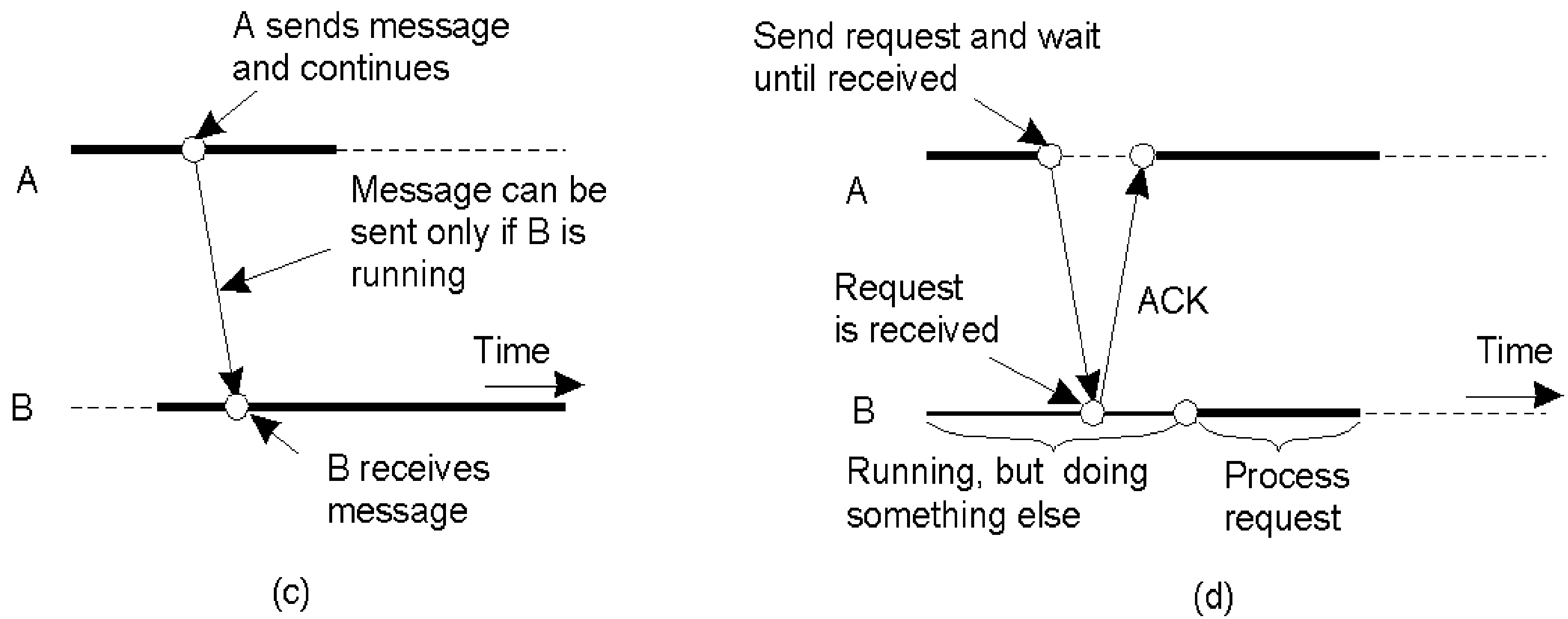
Persistent communication of letters back in the days of the Pony Express.

Persistence and Synchronicity in Communication (3)



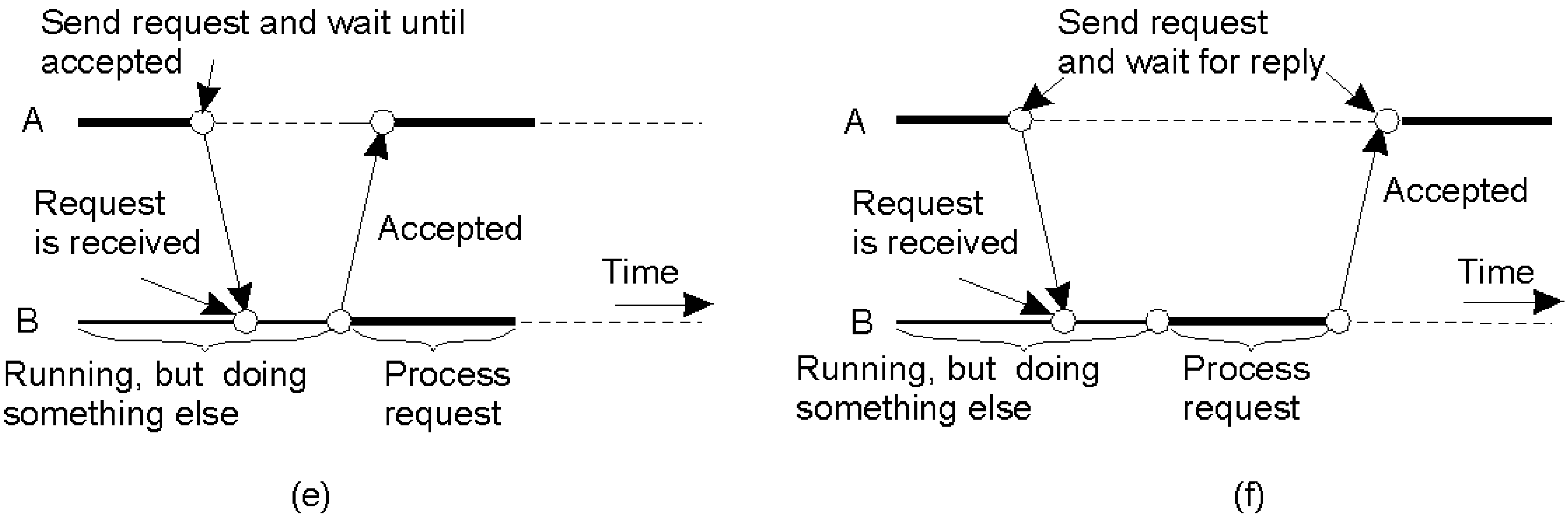
a) Persistent asynchronous communication
b) Persistent synchronous communication

Persistence and Synchronicity in Communication (4)



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

Persistence and Synchronicity in Communication (5)



e) Delivery-based transient synchronous communication at message delivery

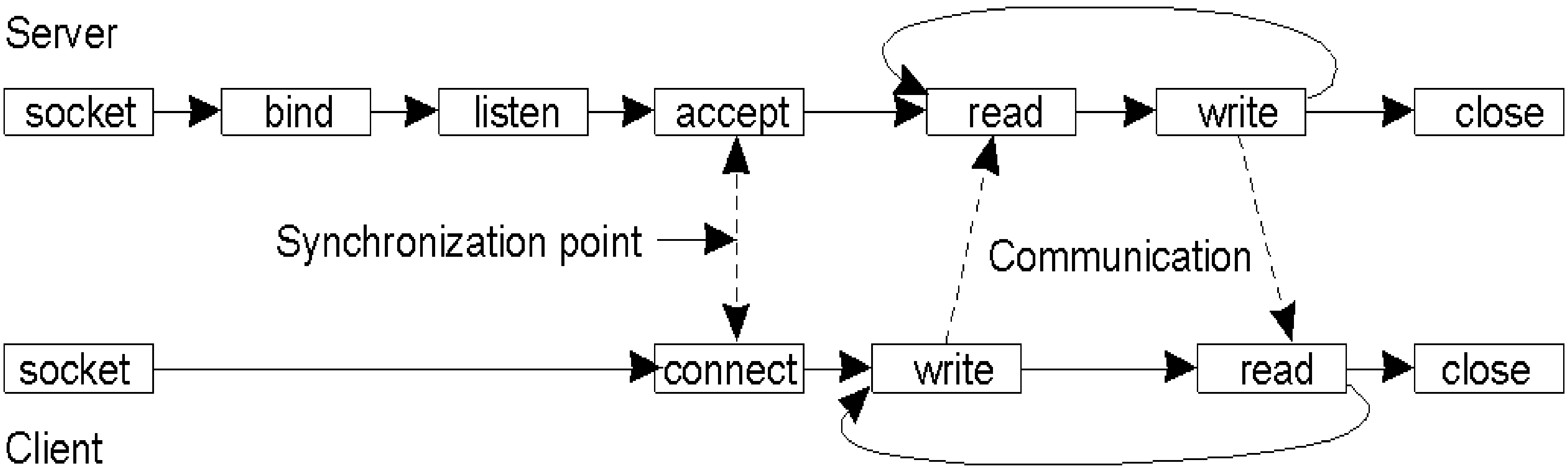
f) Response-based transient synchronous communication

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)



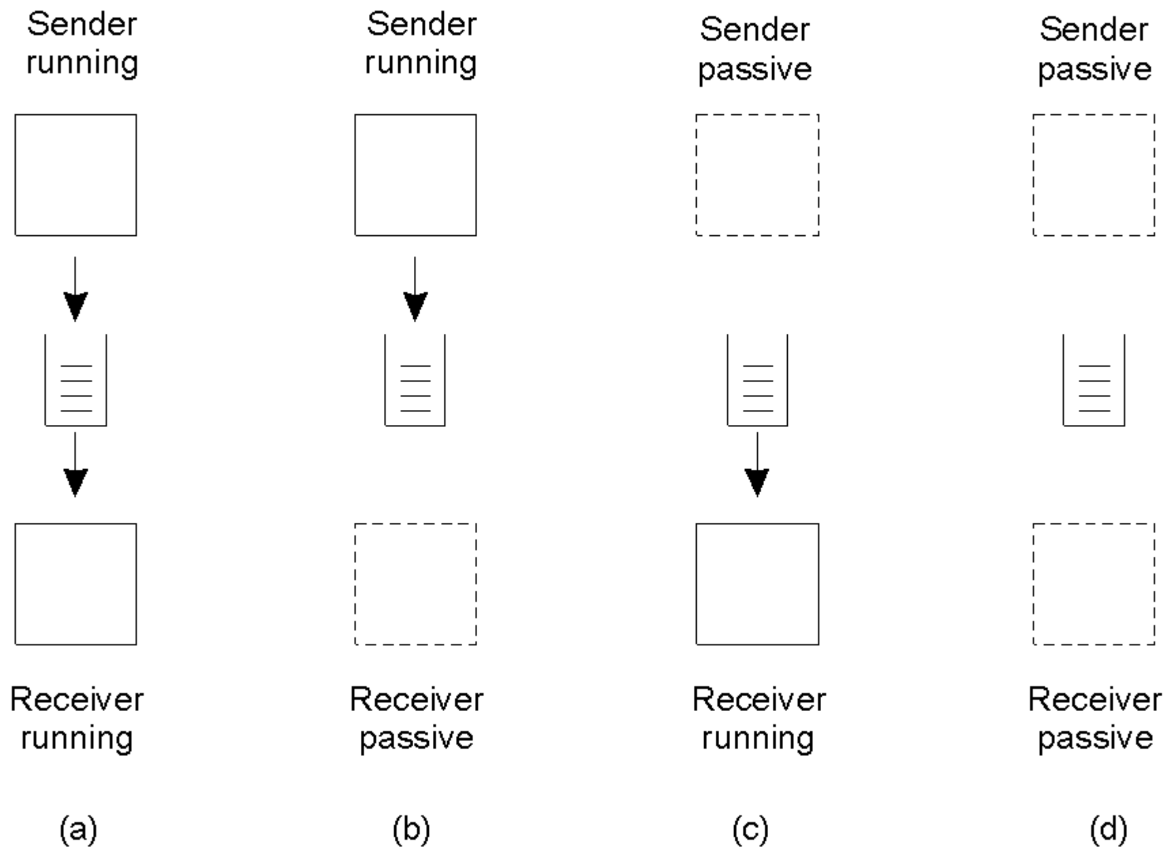
Connection-oriented communication pattern using sockets.

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)



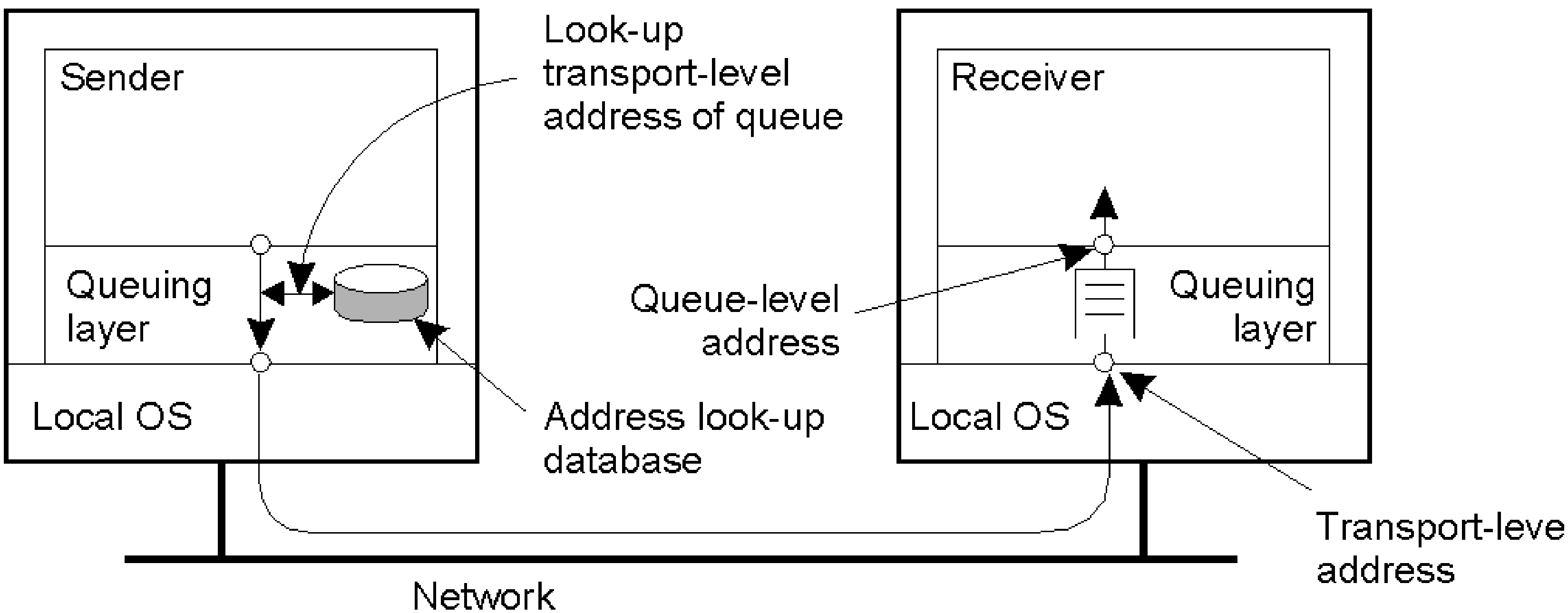
Four combinations for loosely-coupled communications using queues.

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.

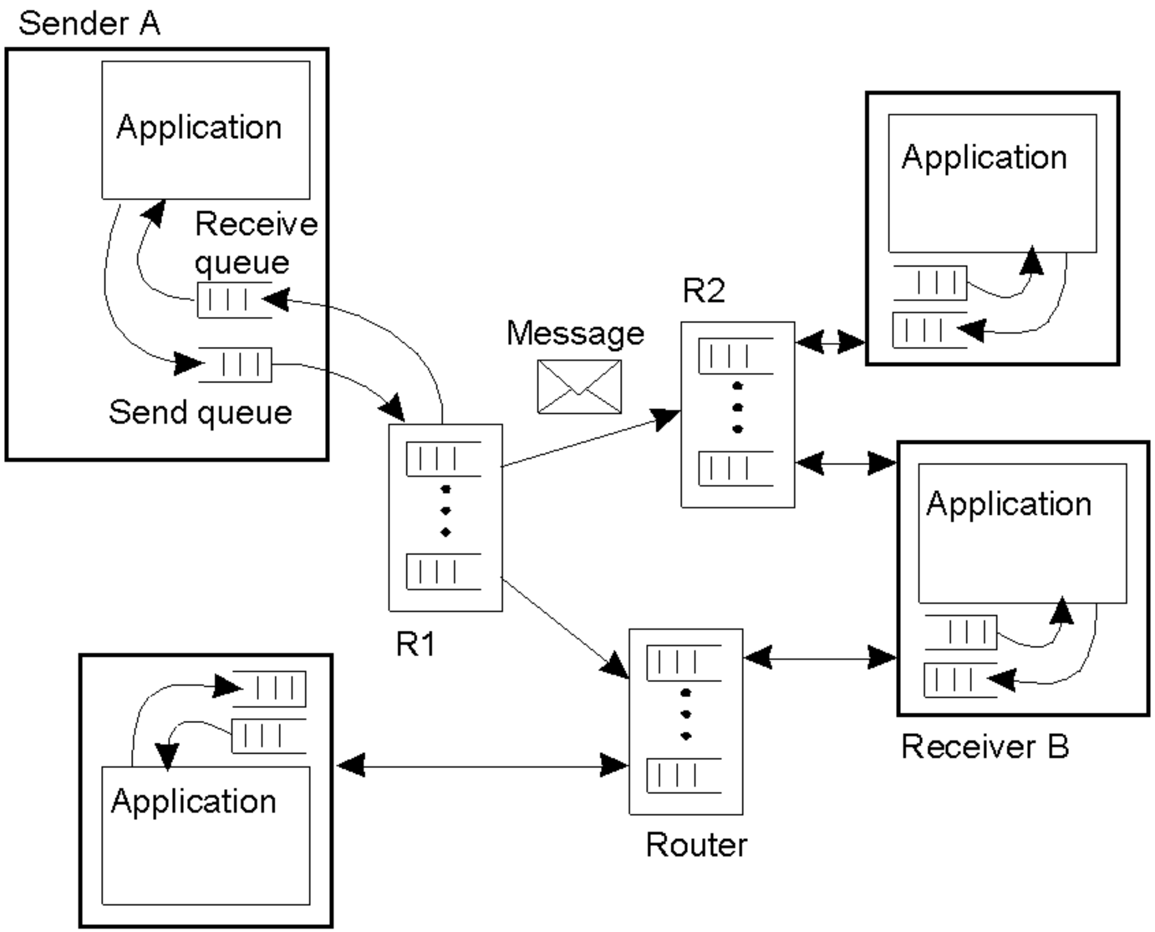
Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System (1)



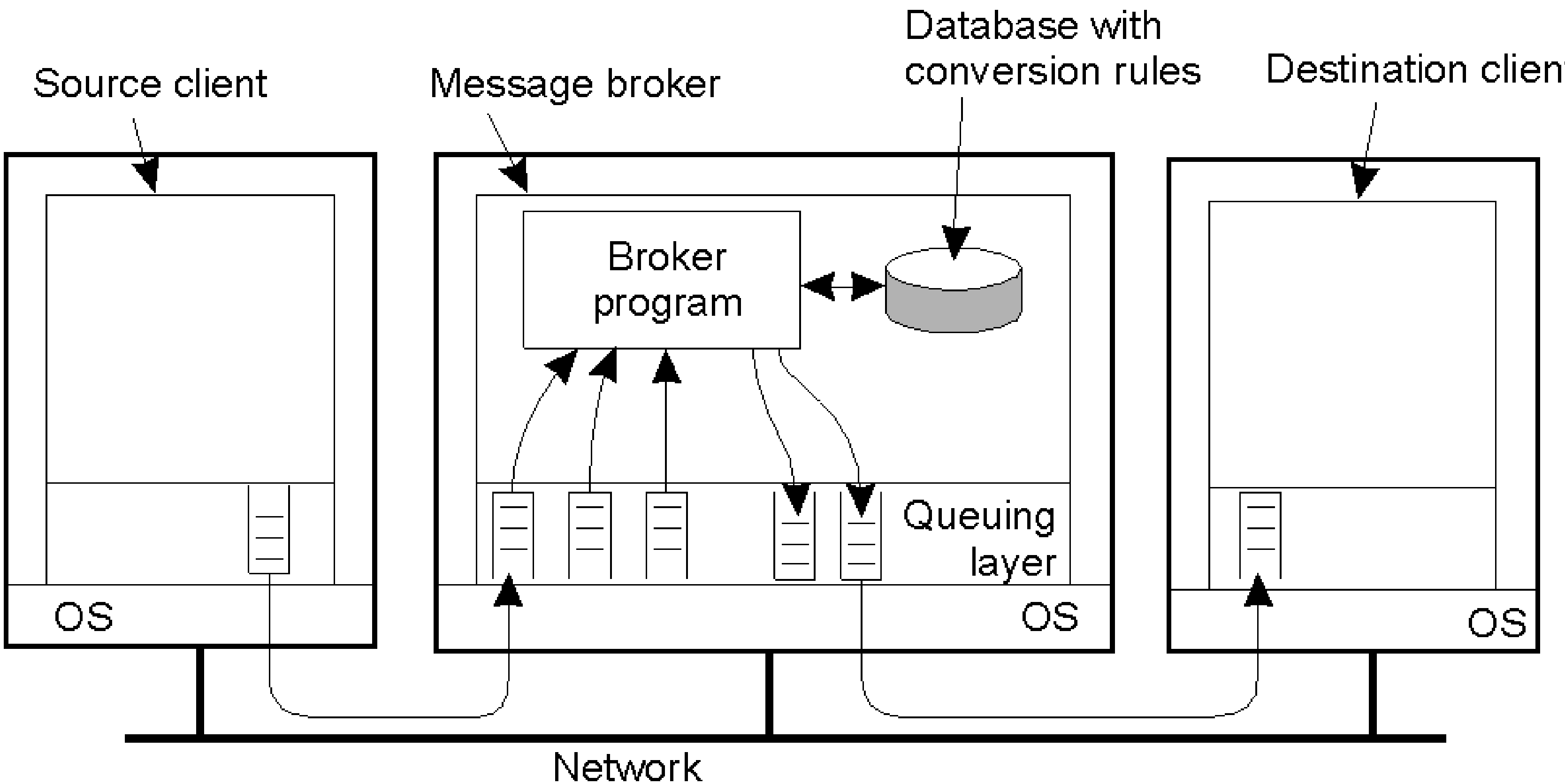
The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System (2)



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

Message Brokers

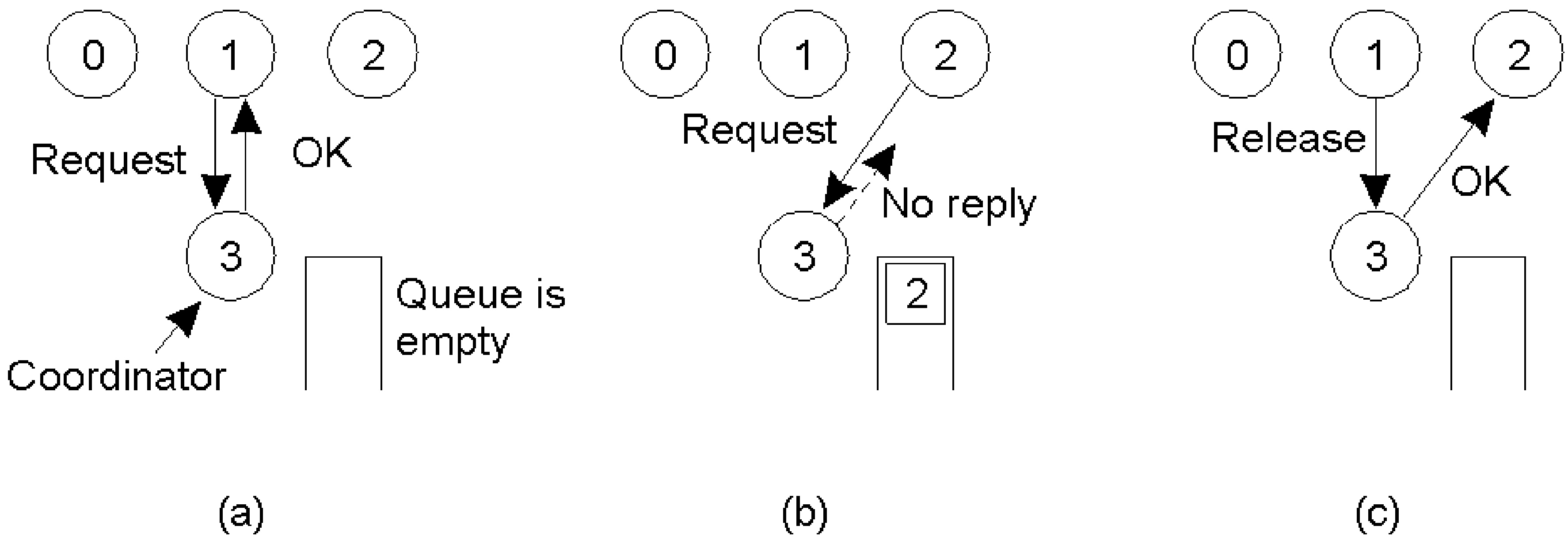


The general organization of a message broker in a message-queuing system.

Lamport's Logical Clocks

- Relation →
  - If a and b are events in the same thread and a comes before b, then a → b
  - If a is the sending of a message by a thread and b is the receipt of the same message by a different thread, then a → b
- Clock Condition: for any events, a and b,
  - If a → b then C(a) < C(b)
- Implementation
  - Each thread increments its clock between any two successive events
  - A massage contains C(a) as its timestamp; upon receiving it, the receiving thread sets its clock to max{clock, C(a) + 1}

Mutual Exclusion: A Centralized Algorithm



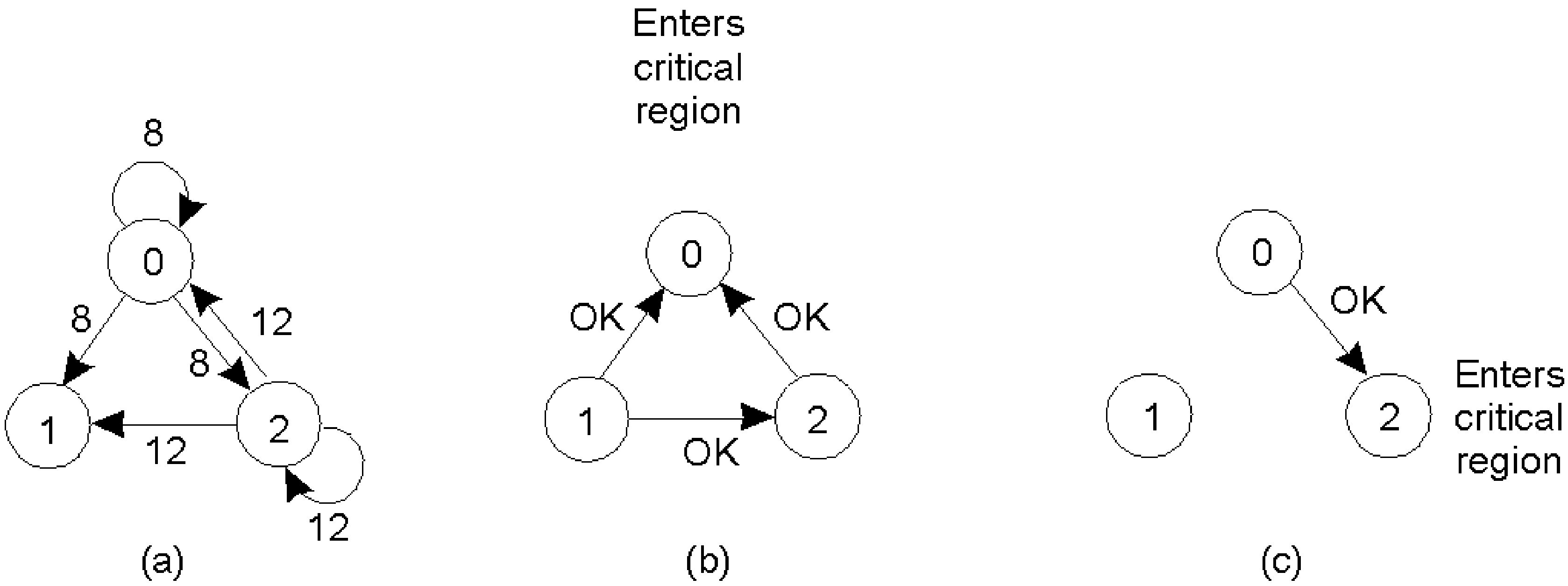
a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
c) When process 1 exits the critical region, it tells the coordinator, when then replies to 2

A Decentralized Algorithm

- For each resource, n coordinators
- Access granted with $m > n/2$ authorizations
- Let p = prob that a coordinator resets in Δt,
- P[k] = k coordinators reset

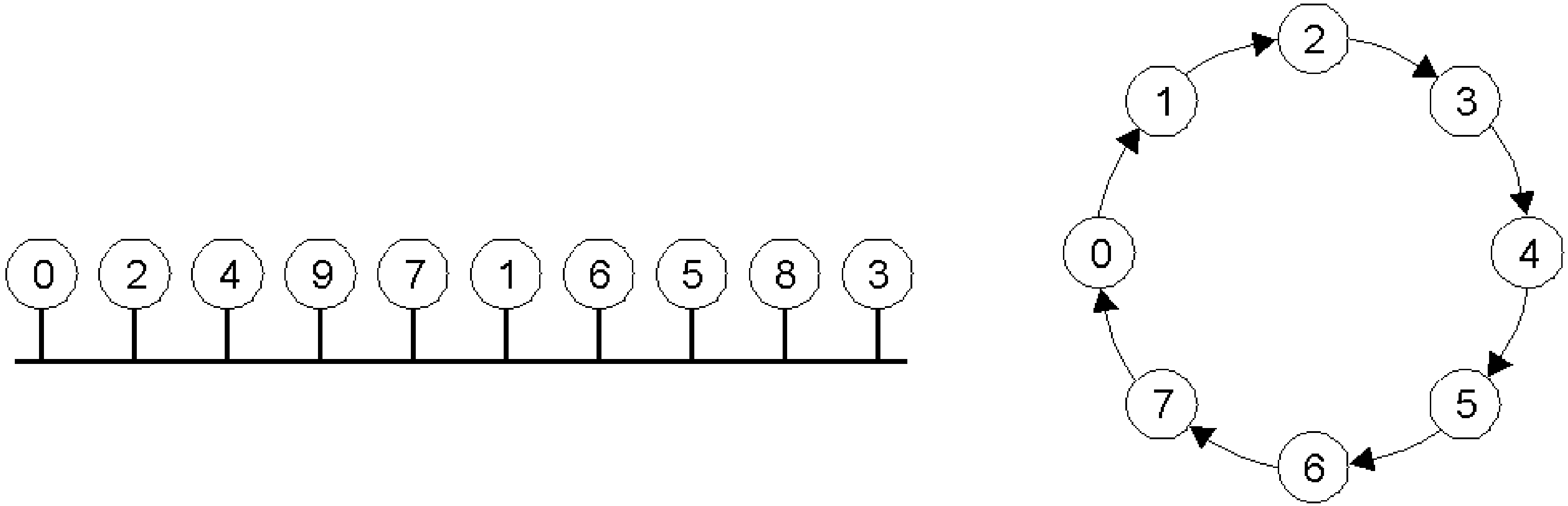
$$P[k] = \binom{m}{k} p^k (1-p)^{m-k}$$

A Distributed Algorithm



a) Two processes want to enter the same critical region at the same moment.
b) Process 0 has the lowest timestamp, so it wins.
c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

A Token Ring Algorithm



(a) (b)

a) An unordered group of processes on a network.

b) A logical ring constructed in software.

Comparison

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ◎	0 to n – 1	Lost token, process crash

A comparison of three mutual exclusion algorithms.

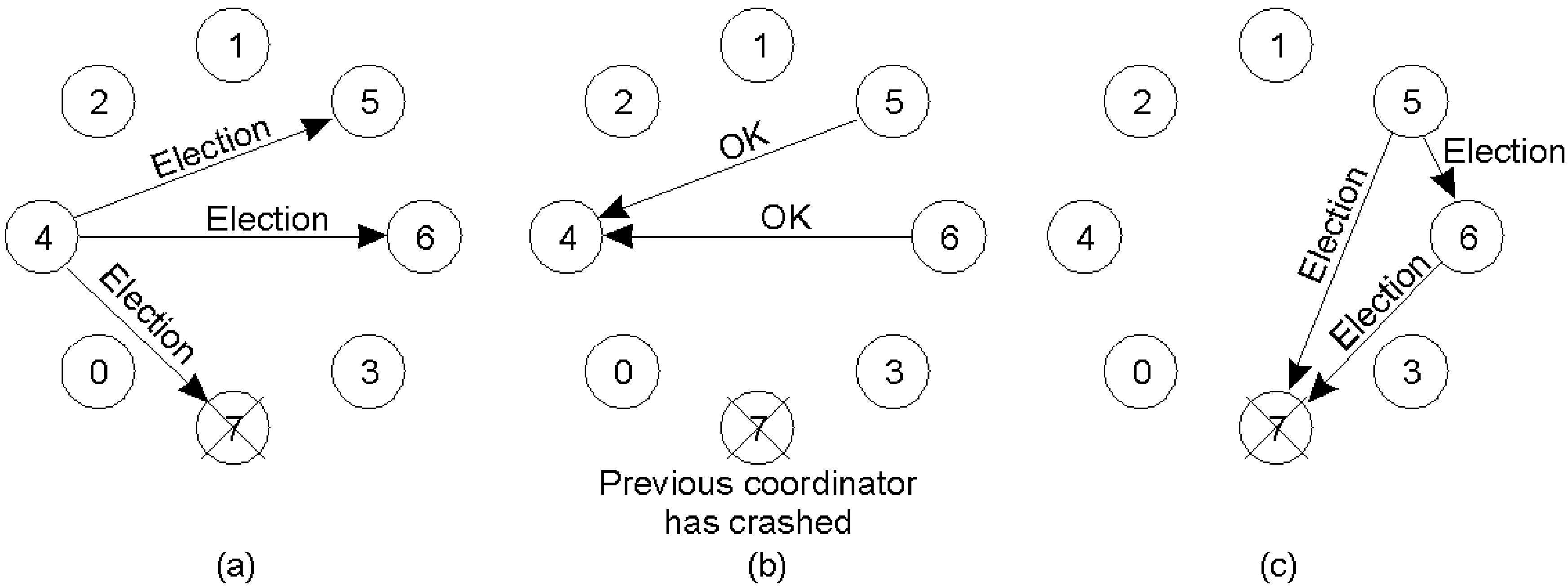
Election Algorithms

- How is coordinator to be selected dynamically?
- N.B.: in some systems, chosen by hand (e.g., file server) → single point of failure
-
- Questions:

Centralized or decentralized?

Which is more robust?

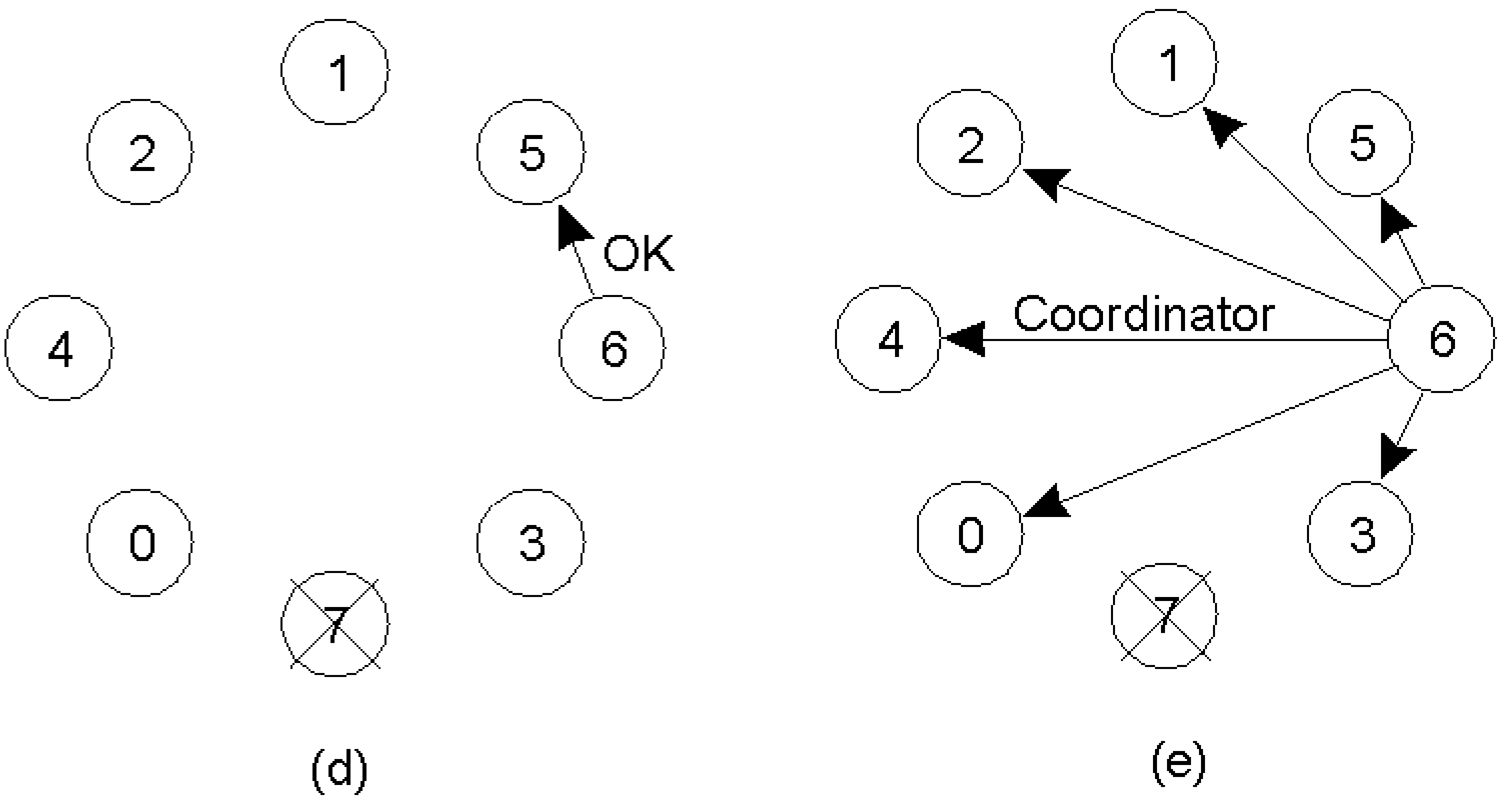
The Bully Algorithm (1)



The bully election algorithm

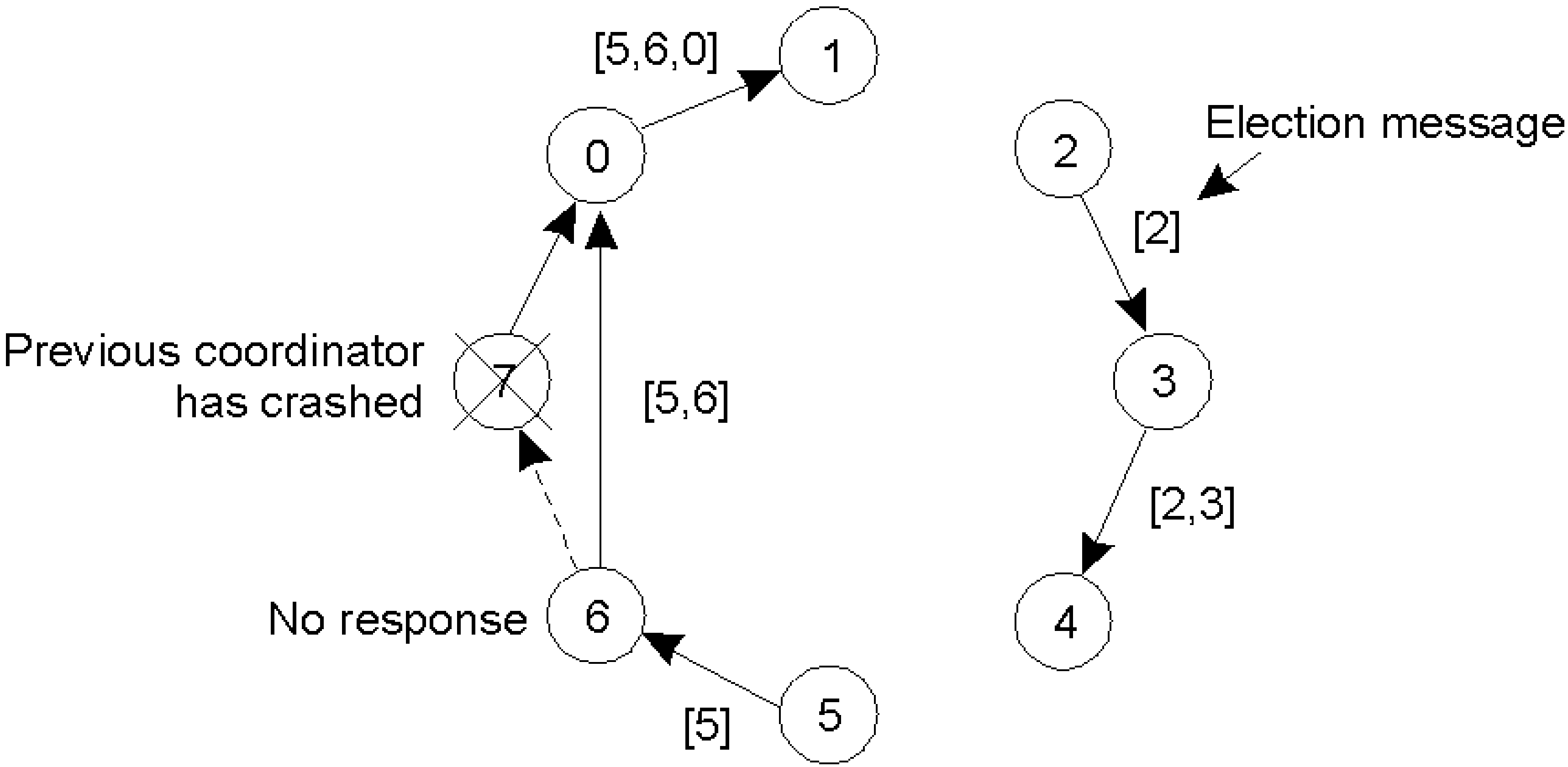
- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

The Bully Algorithm (2)



d) Process 6 tells 5 to stop
e) Process 6 wins and tells everyone

A Ring Algorithm



Election algorithm using a ring.

Thank you for your attention