Distributed Operating Systems

Communication in Distributed Systems

Topics

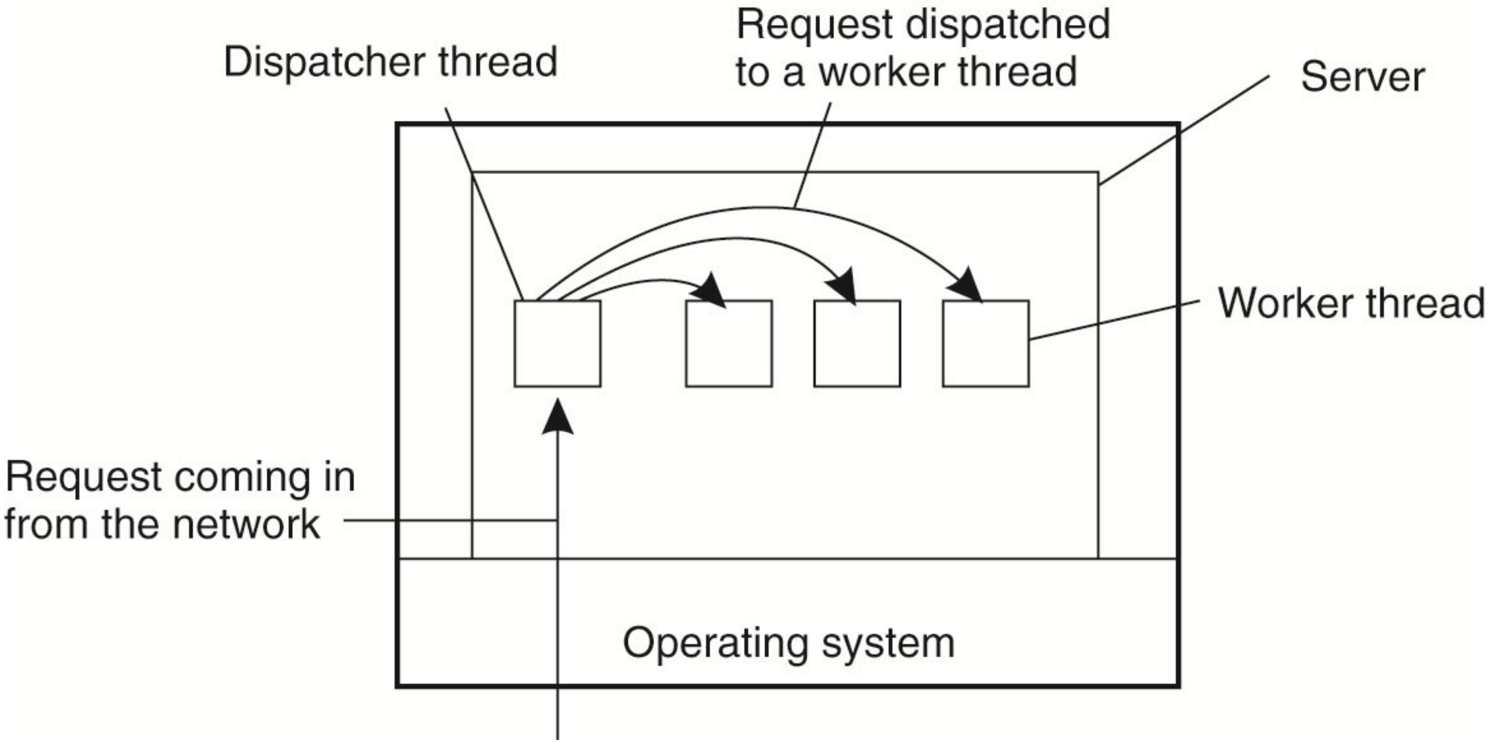
- Transparency in a Distributed System
- Client-Side Distribution Transparency
- Server-Side Distribution Transparency
- Remote Procedure Calls (RPC)
- Asynchronous RPC
- Message Passing
- Message Brokers

Definition of a Distributed System

- A distributed system is:

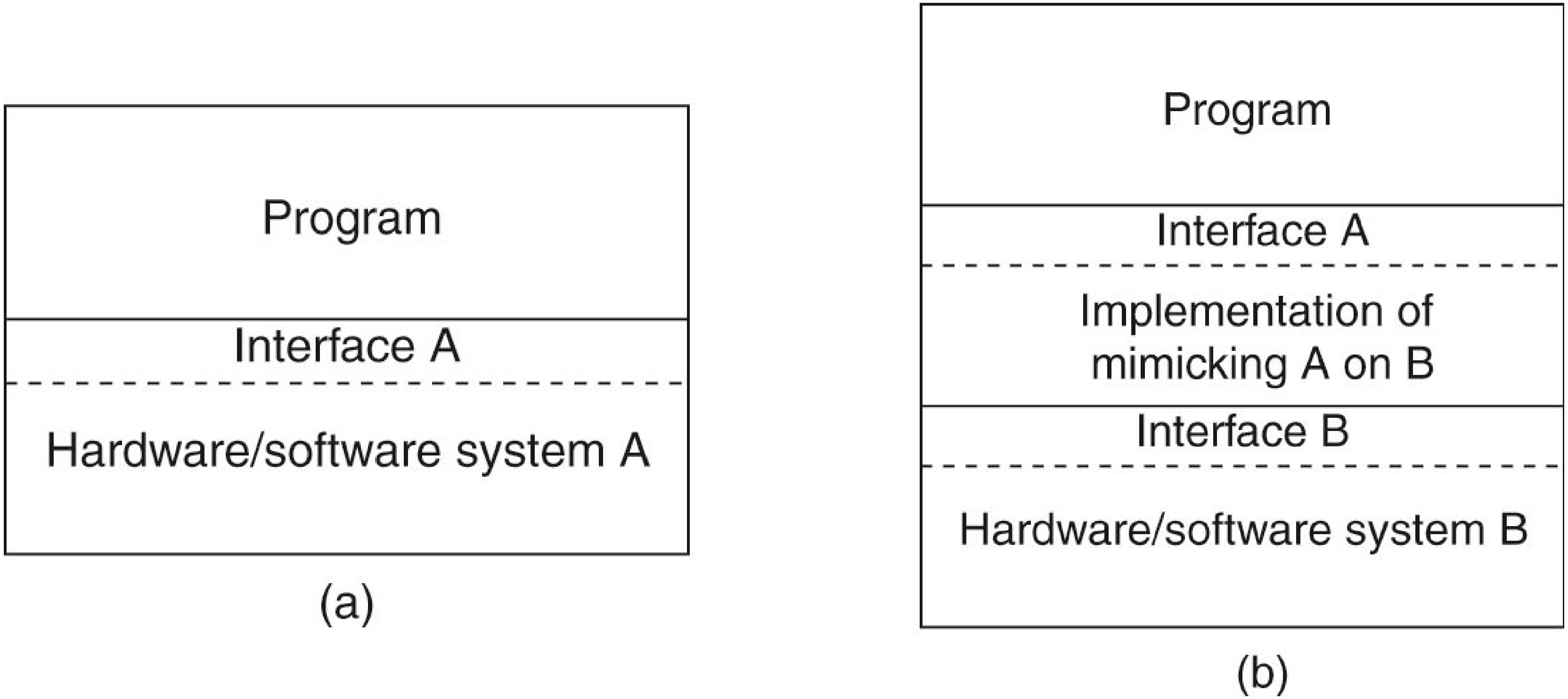
  A collection of independent computers that appears to its users as a single coherent system.

Multithreaded Servers


Dispatcher thread
Request dispatched to a worker thread
Server
Worker thread
Request coming in from the network
Operating system

A multithreaded server organized in a dispatcher/worker model.

The Role of Virtualization in Distributed Systems



Program

Interface A

Hardware/software system A

(a)

Program

Interface A

Implementation of mimicking A on B

Interface B

Hardware/software system B

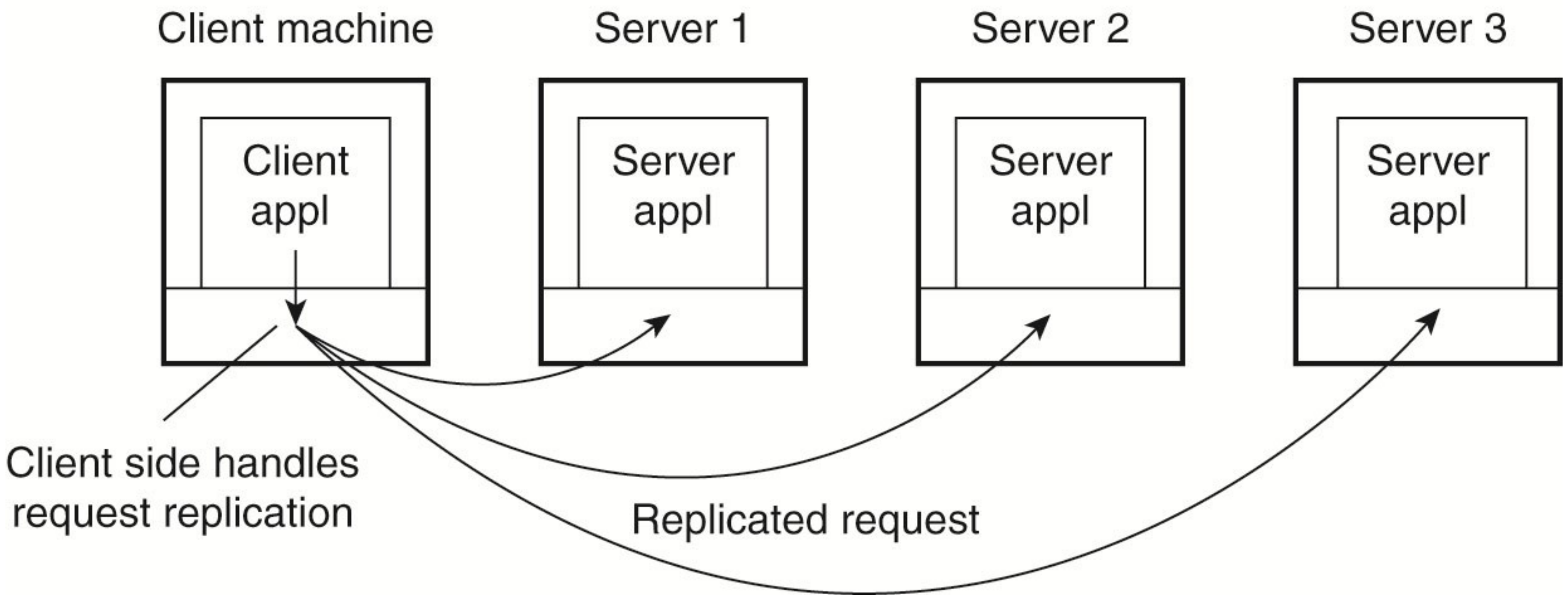
(b)

(a) General organization between a program, interface, and system.

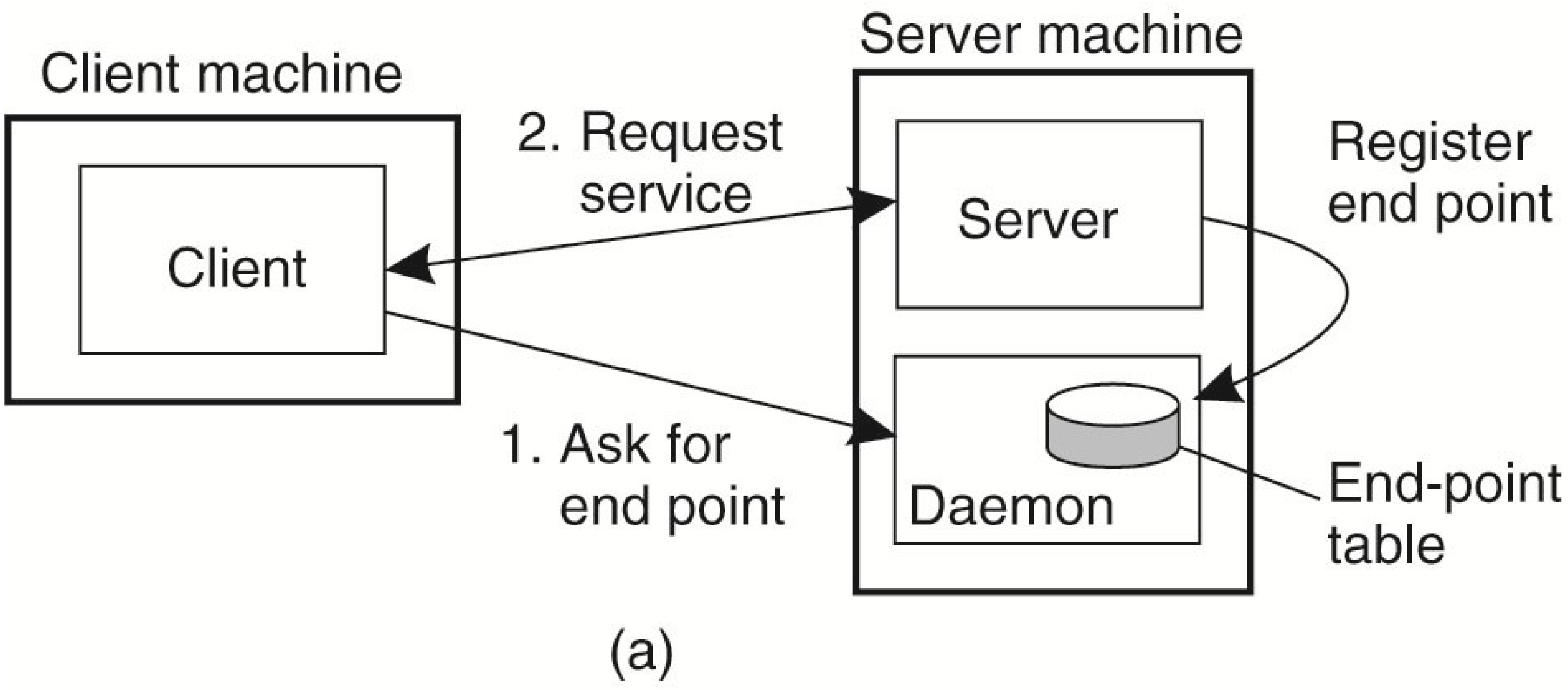
(b) General organization of virtualizing system A on top of system B.

Client-Side Software for Distribution Transparency

- Transparent replication of a server using a client-side solution.



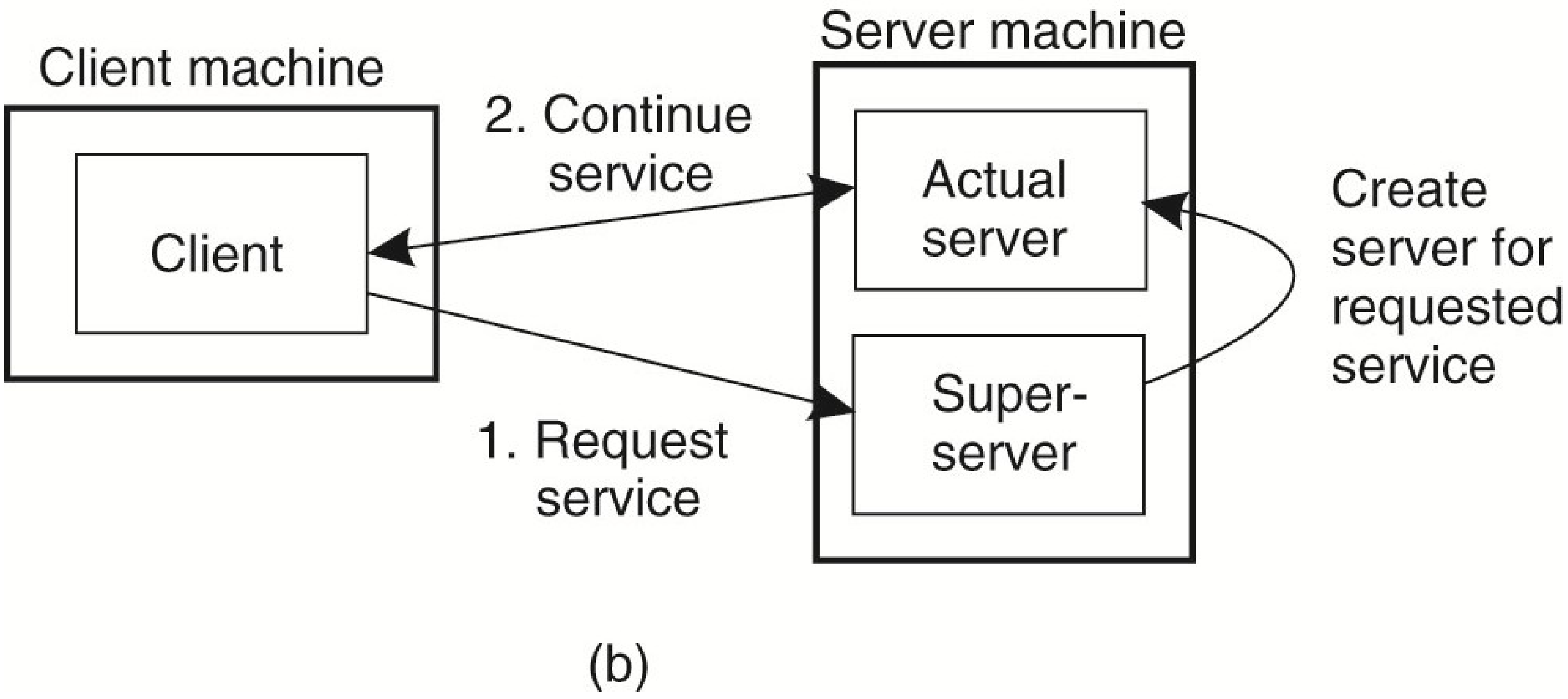
General Design Issues (1)



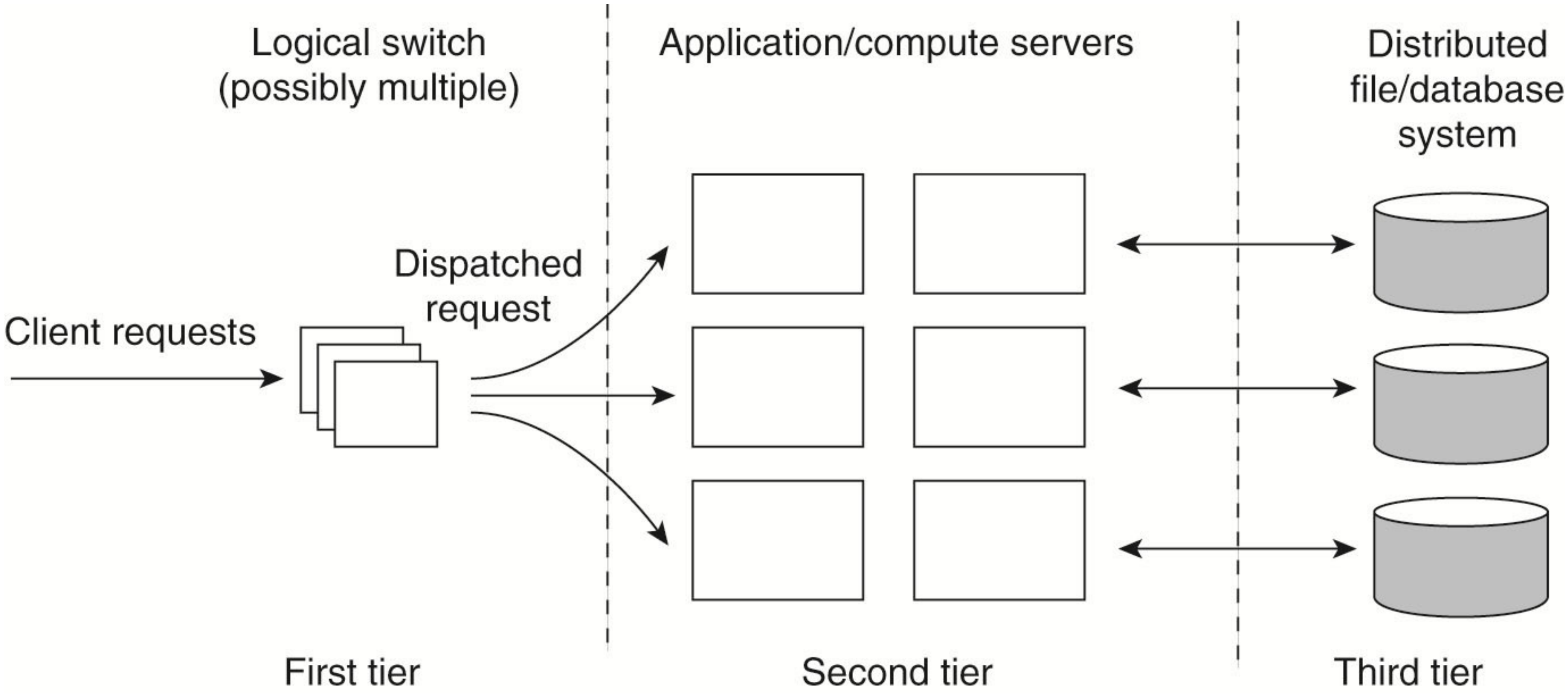
Client-to-server binding using a daemon.

General Design Issues (2)

r.



Server Clusters



Remote Procedure Calls (1)

- A remote procedure call occurs in the following steps:

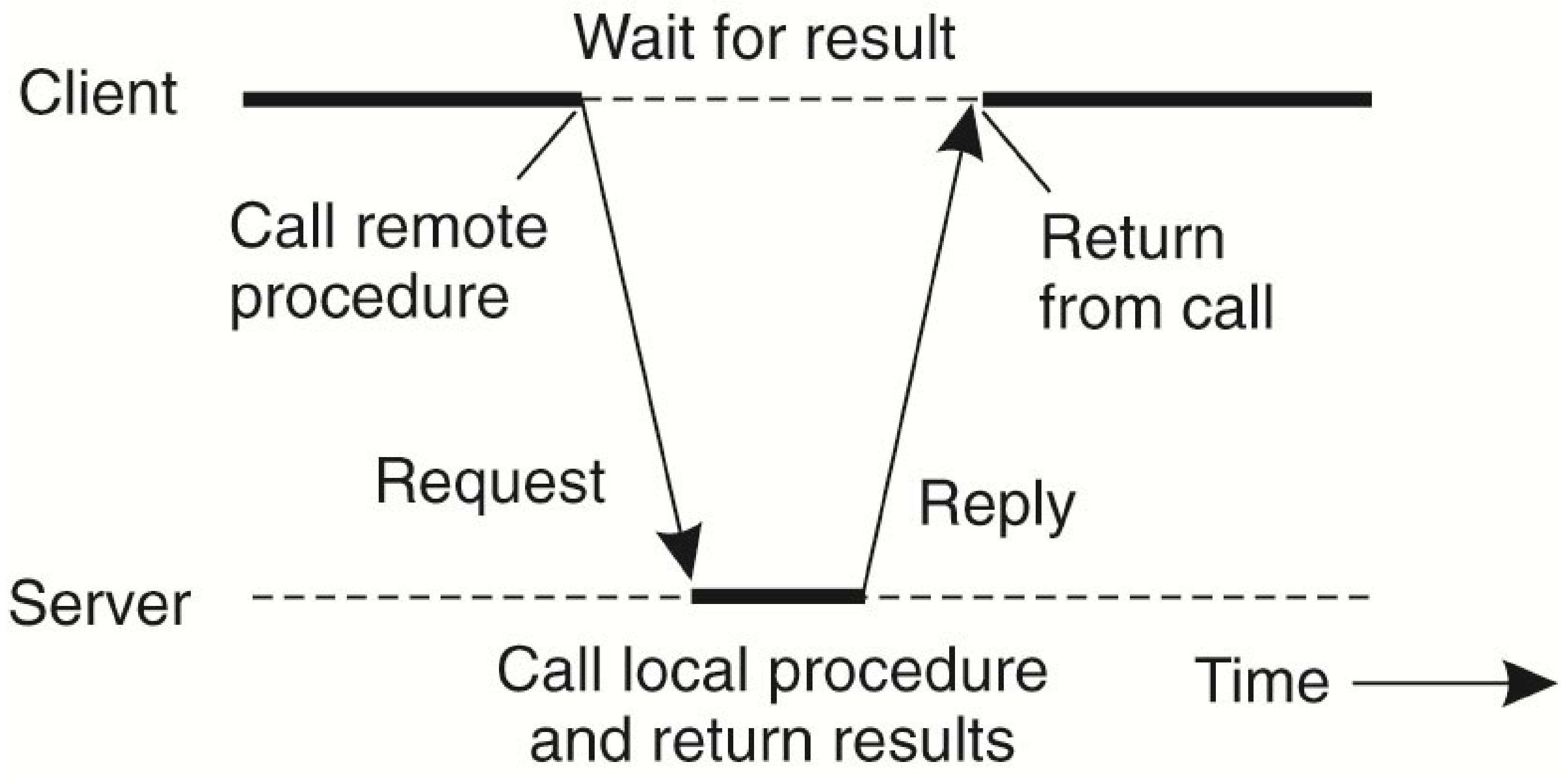
1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.

Remote Procedure Calls (2)

- A remote procedure call occurs in the following steps (continued):

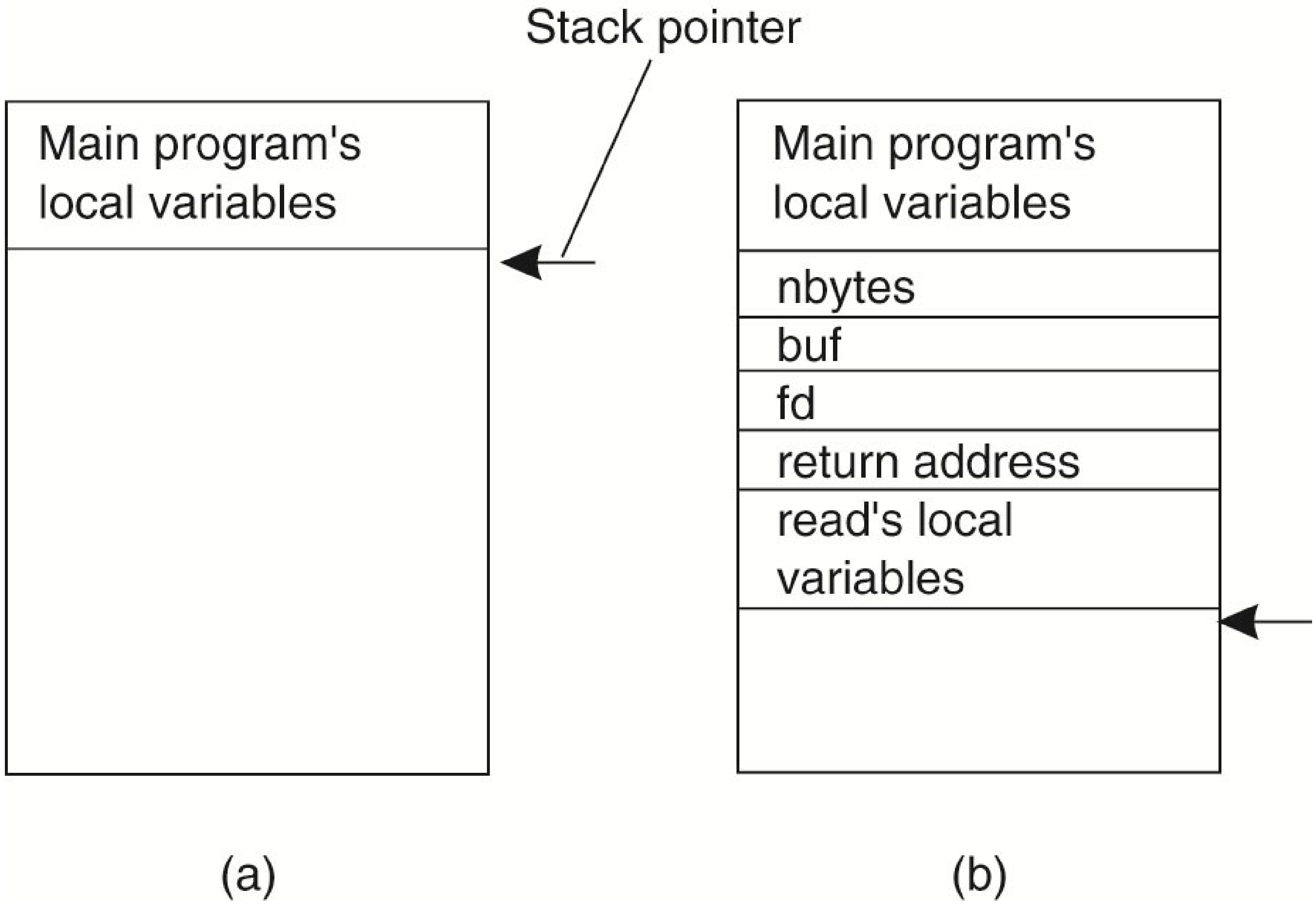
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Client and Server Stubs



Principle of RPC between a client and server program.

Conventional Procedure Call

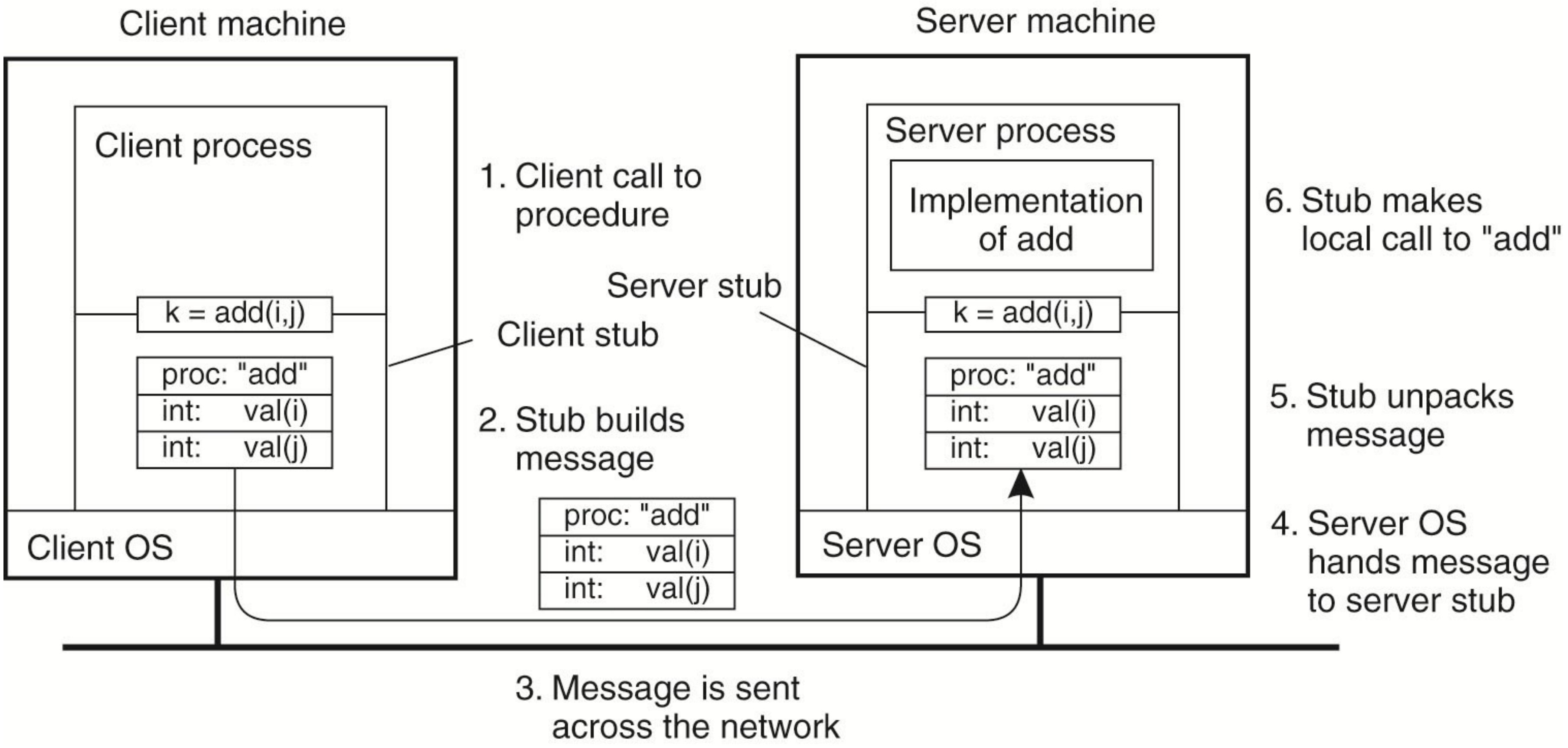


Parameter passing in a local procedure call:

(a) The stack before the call to read.

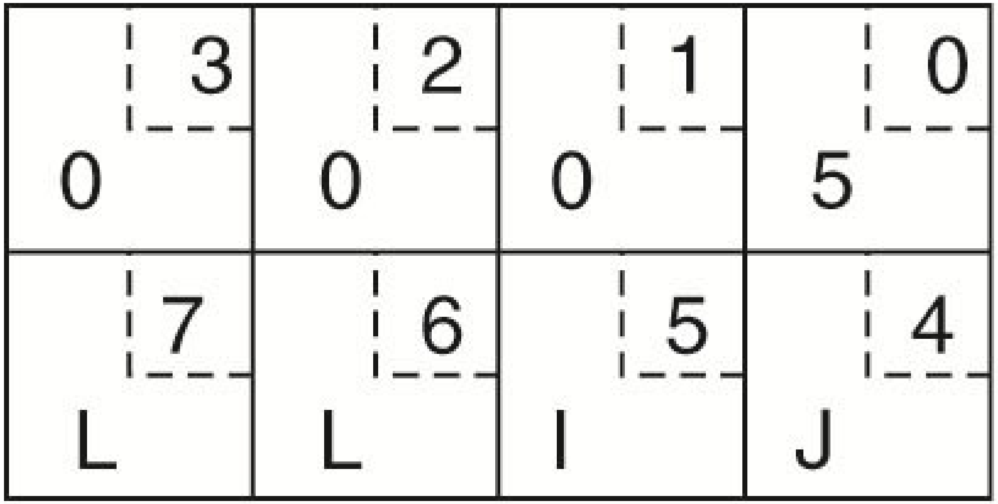
(b) The stack while the called procedure is active.

Passing Value Parameters (1)



Passing Value Parameters (2)

The original message on the Pentium.

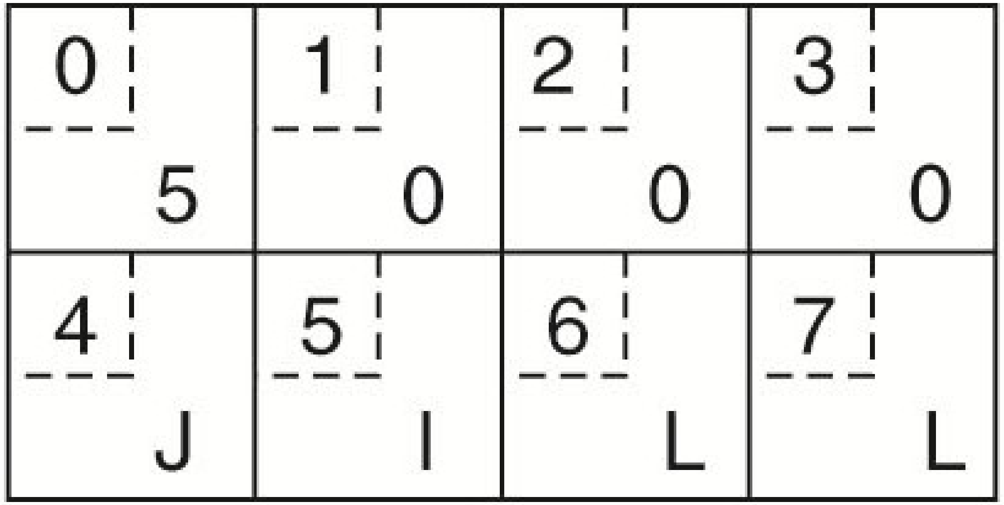


3 0	2 0	1 0	0 5
7 L	6 L	5 I	4 J

(a)

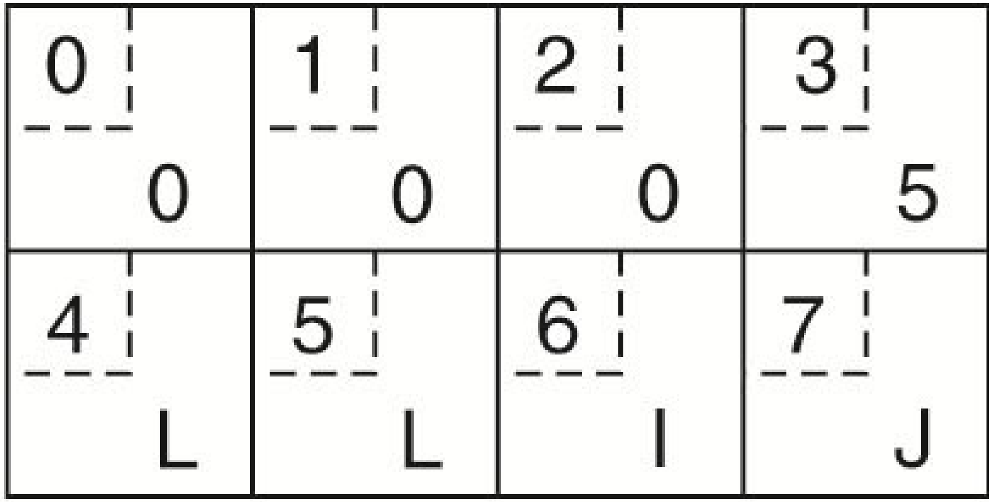
Passing Value Parameters (3)

The original message on the Pentium.



(b)

Passing Value Parameters (4)



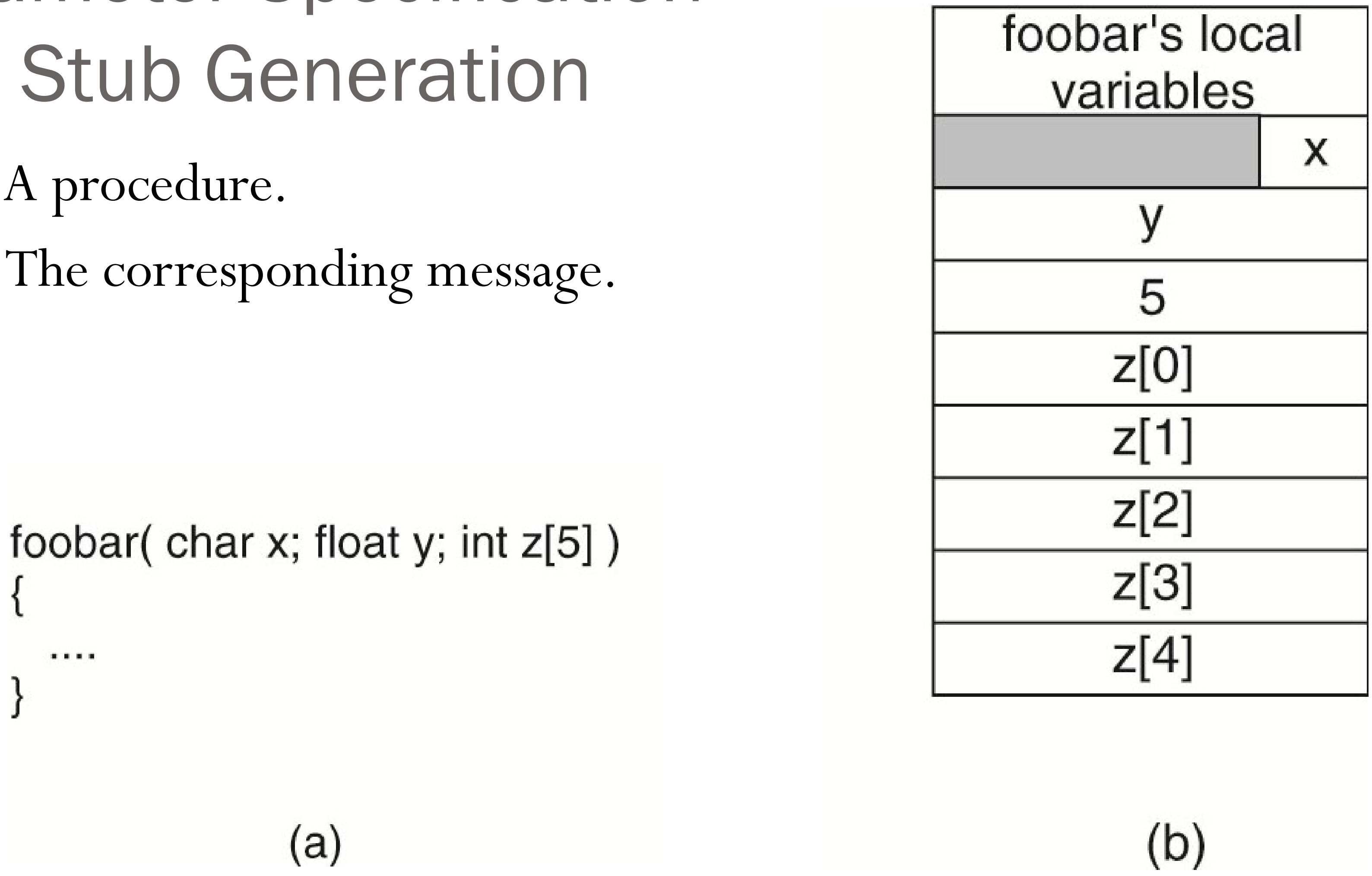
0 0	1 0	2 0	3 5
4 L	5 L	6 I	7 J

(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)

foobar's local variables

x

y

5

z[0]

z[1]

z[2]

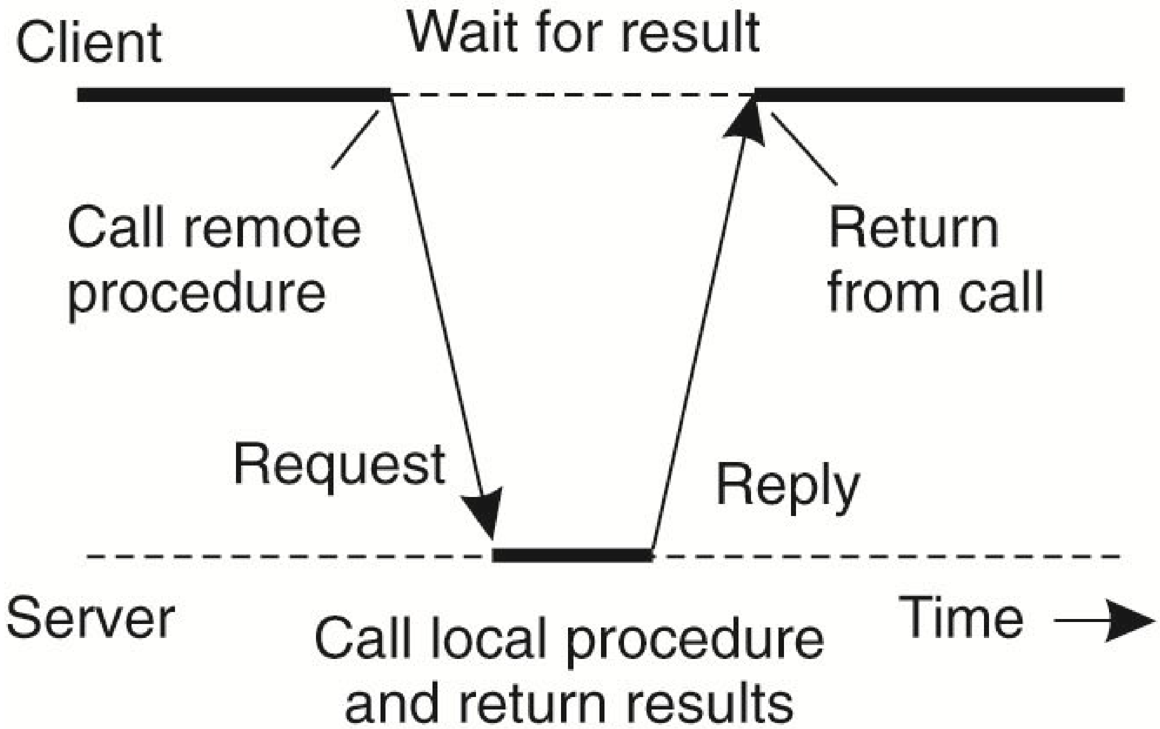
z[3]

z[4]

(b)

Asynchronous RPC (1)

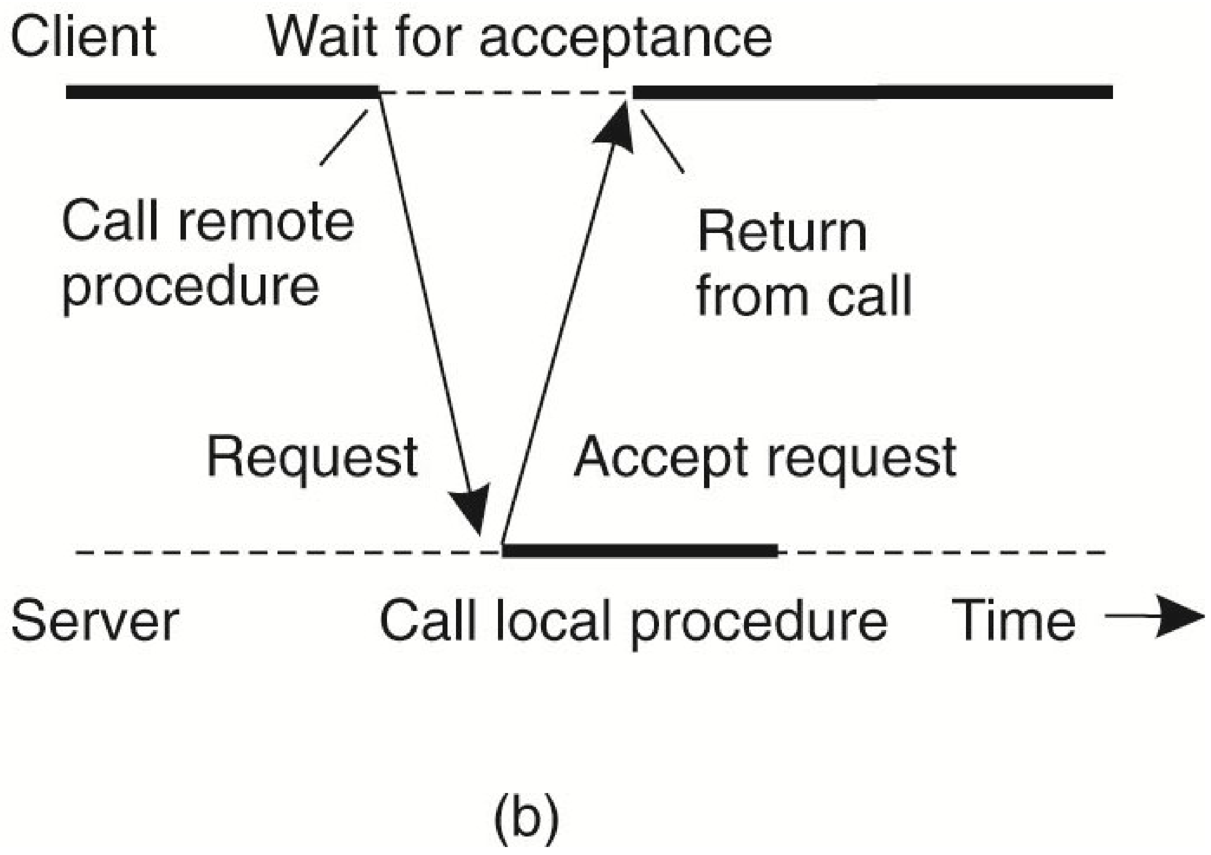
The interaction between client and server in a traditional RPC.



(a)

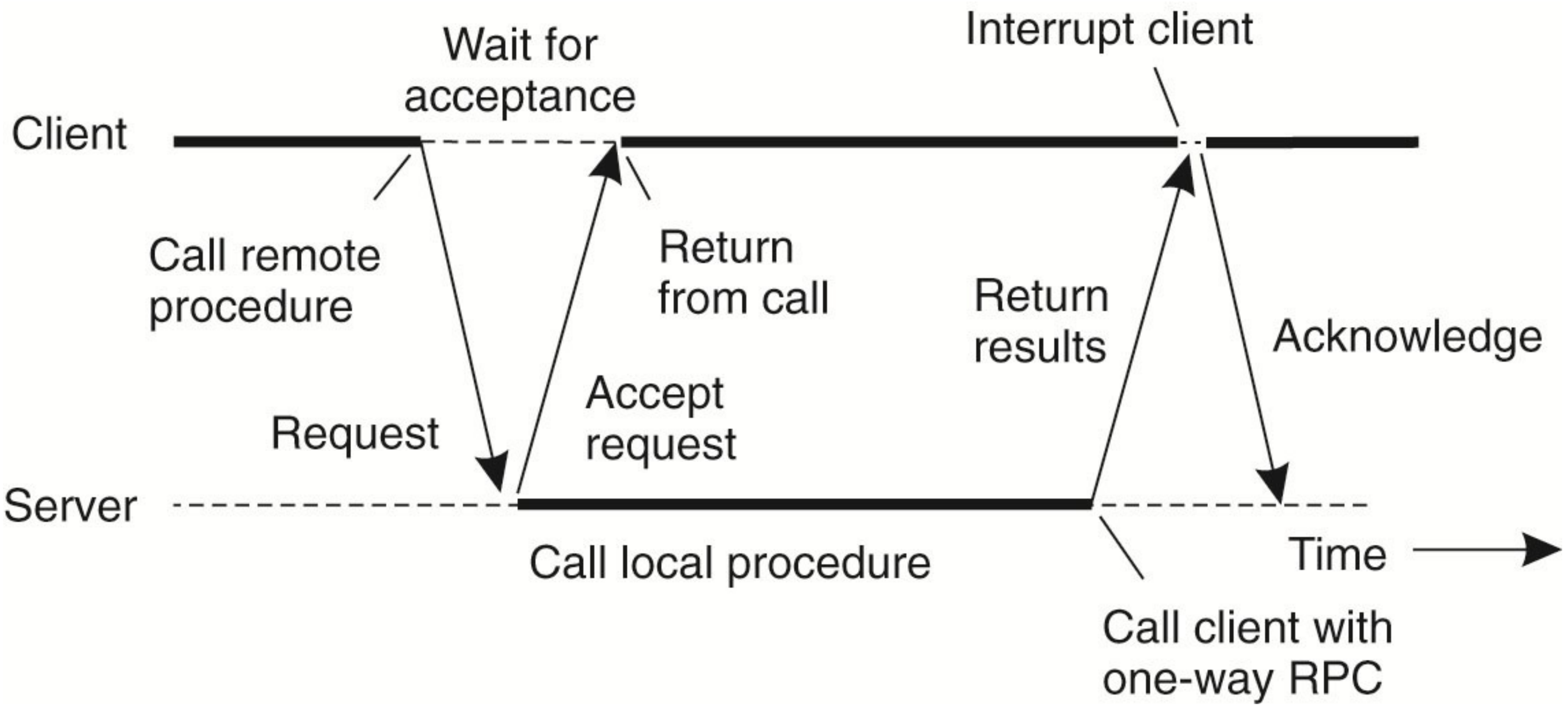
Asynchronous RPC (2)

The interaction using asynchronous RPC.



Asynchronous RPC (3)

A client and server interacting through two asynchronous RPCs.



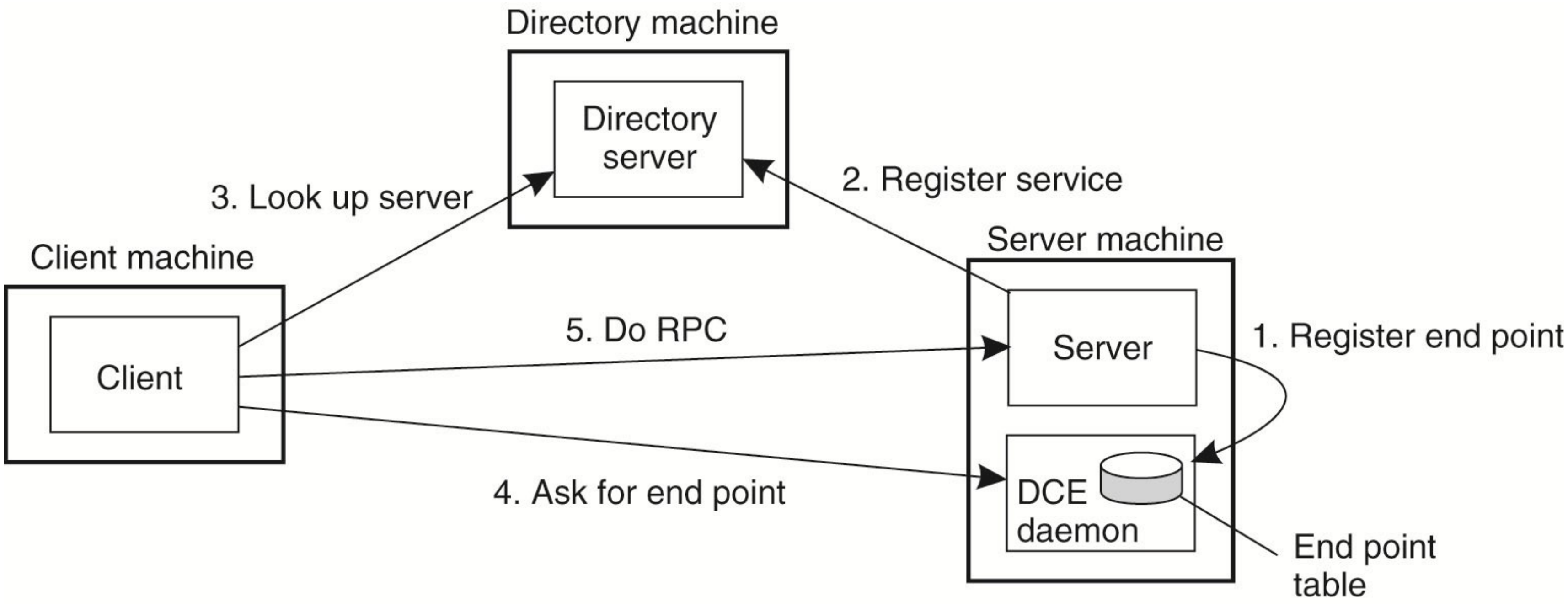
Binding a Client to a Server (1)

- Registration of a server makes it possible for a client to locate the server and bind to it.

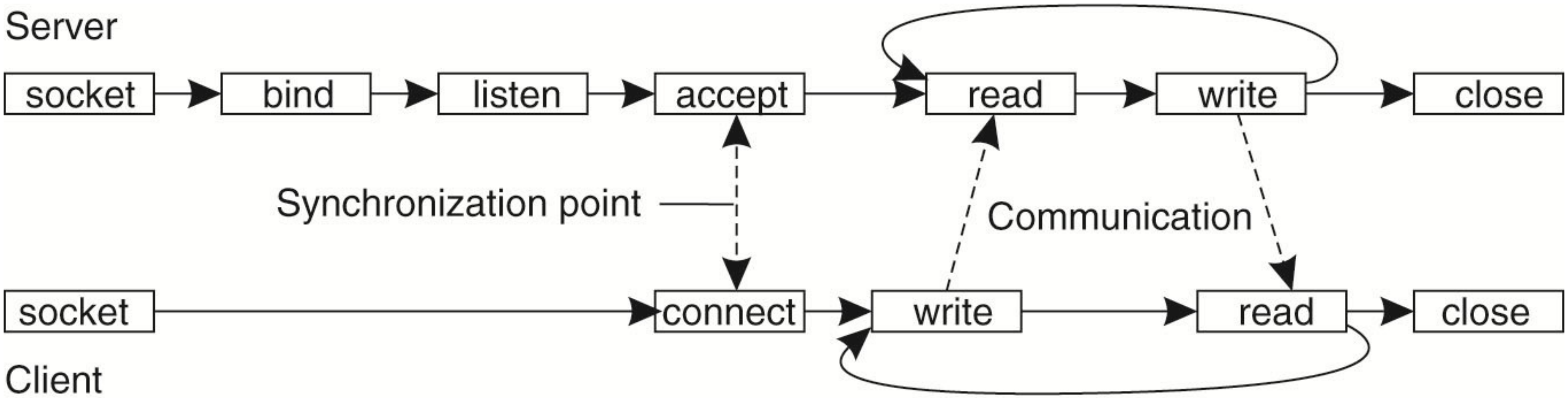
- Server location is done in two steps:
  1. Locate the server’s machine.
  2. Locate the server on that machine.

Binding a Client to a Server (2)

Client-to-server binding in DCE.

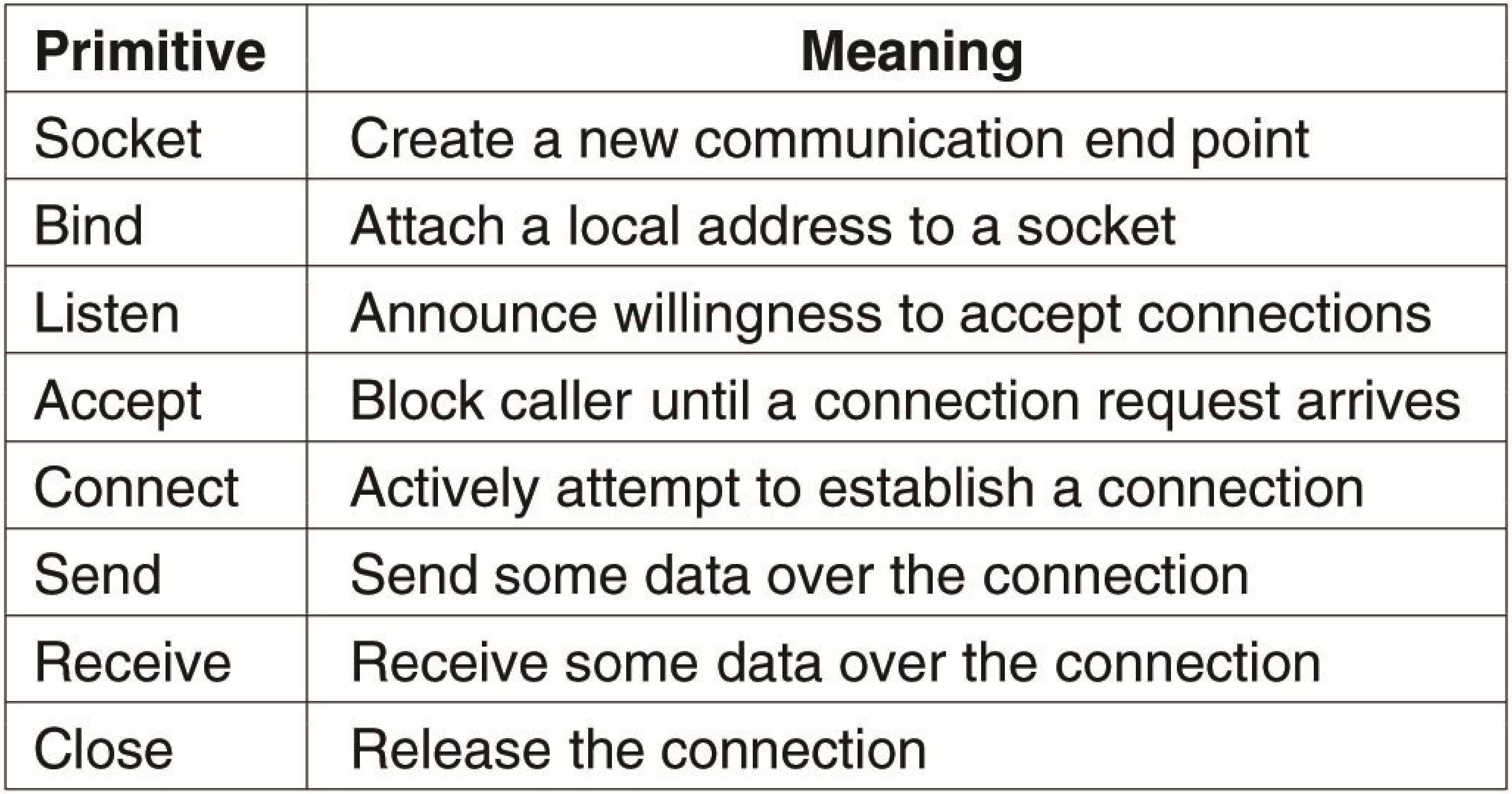


The Message-Passing Interface (1)



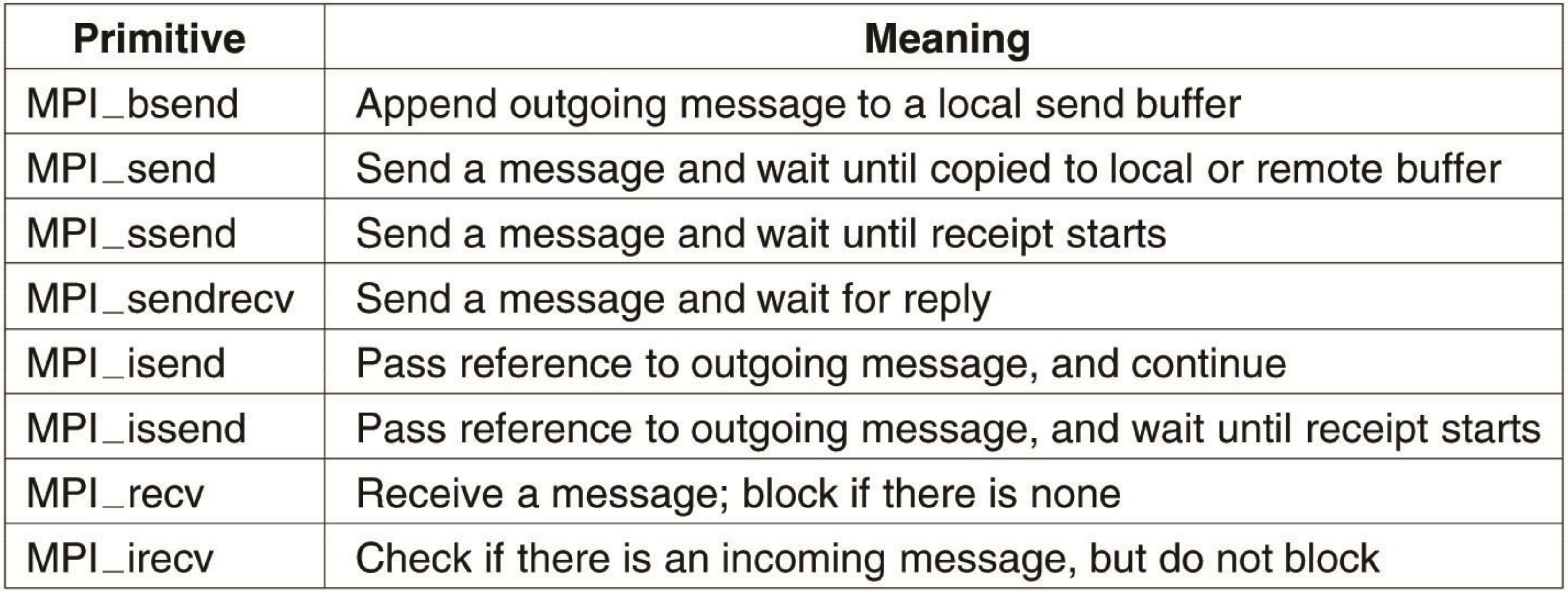
Connection-oriented communication pattern using sockets.

The Message-Passing Interface (2)



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

The Message-Passing Interface (3)

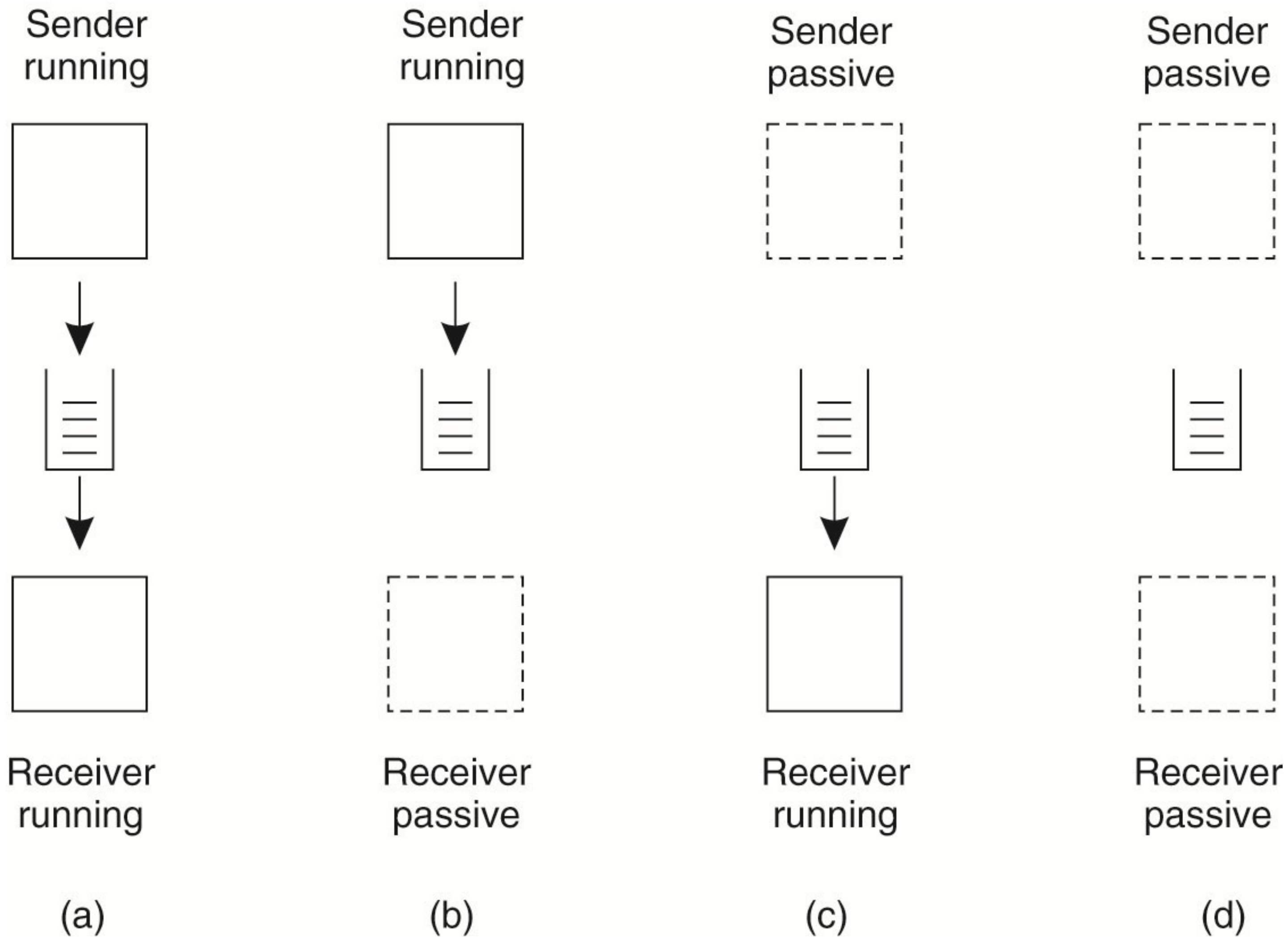


Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
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

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

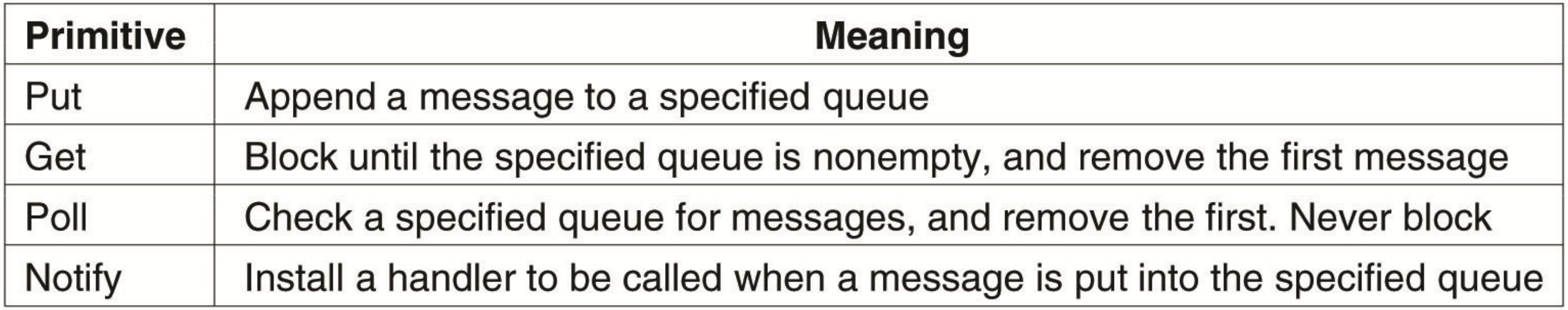
Message-Queuing Model (1)

Four combinations for loosely-coupled communications using queues.



Message-Queuing Model (2)

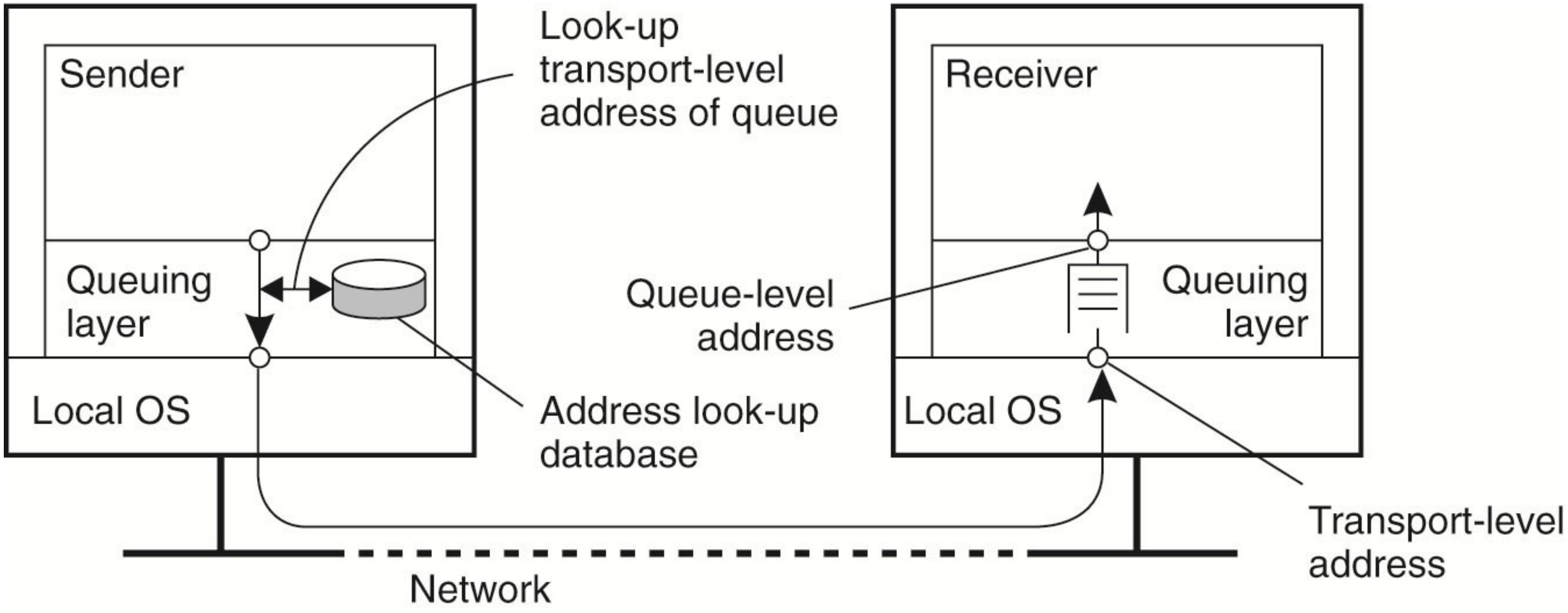
Basic interface to a queue in a message-queuing system.



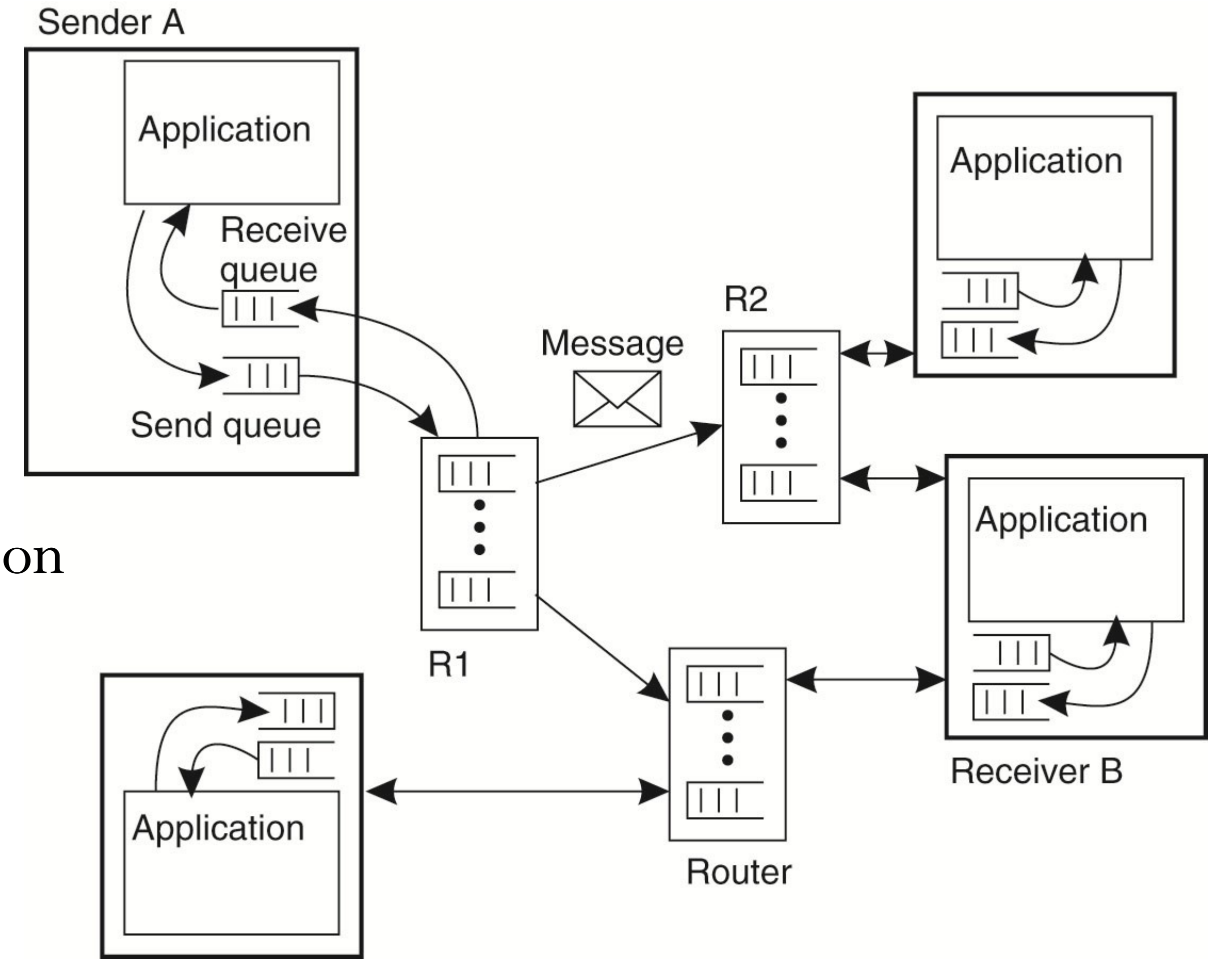
Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

General Architecture of a Message-Queuing System (1)

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

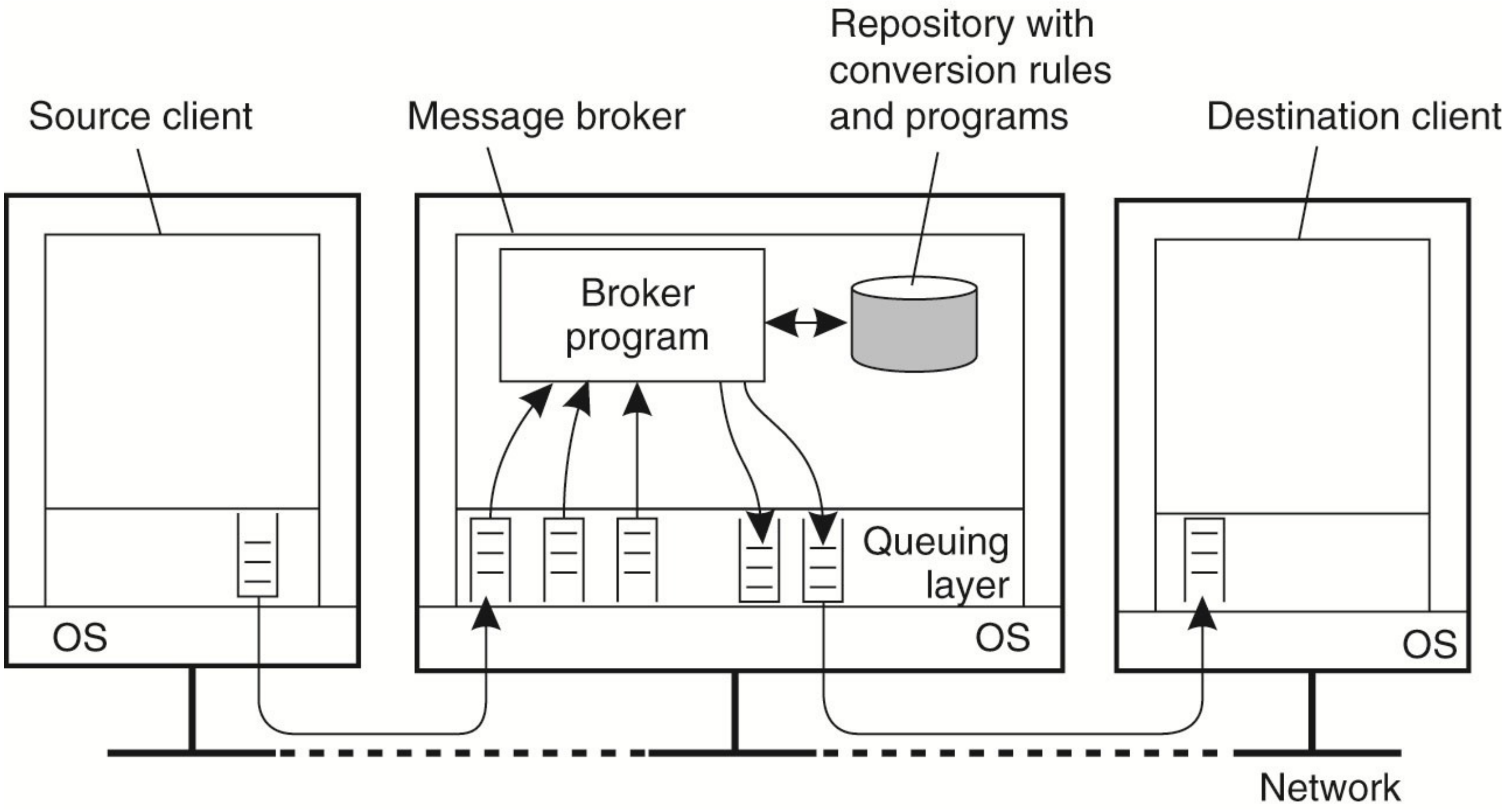


General Architecture of a Message-Queuing System (2)



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

Message Brokers



Questions?