DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 4
Communication

Some Basic Definitions

- **ISO** – International standards organization reference
- model

- **OSI** – Open System Interconnection

- **Open system** – a system ready to communicate with any other open system by using standard rules that govern the format, contents, and meaning of the messages sent and received

Some Basic Definitions

- **Protocols** – formalized rules agreed by a group of computers to communicate over a network
  - Connection-oriented – the sender and the receiver sets a connection before exchanging any data
  - Connectionless – no advance set up is needed
- **Layers –** to divide the communication task into manageable pieces, where each layer deals with specific aspect of communication
- **Interface –** each layer has an interface which consists of a set of operations that define the service of the layer for its users

Layered Protocols

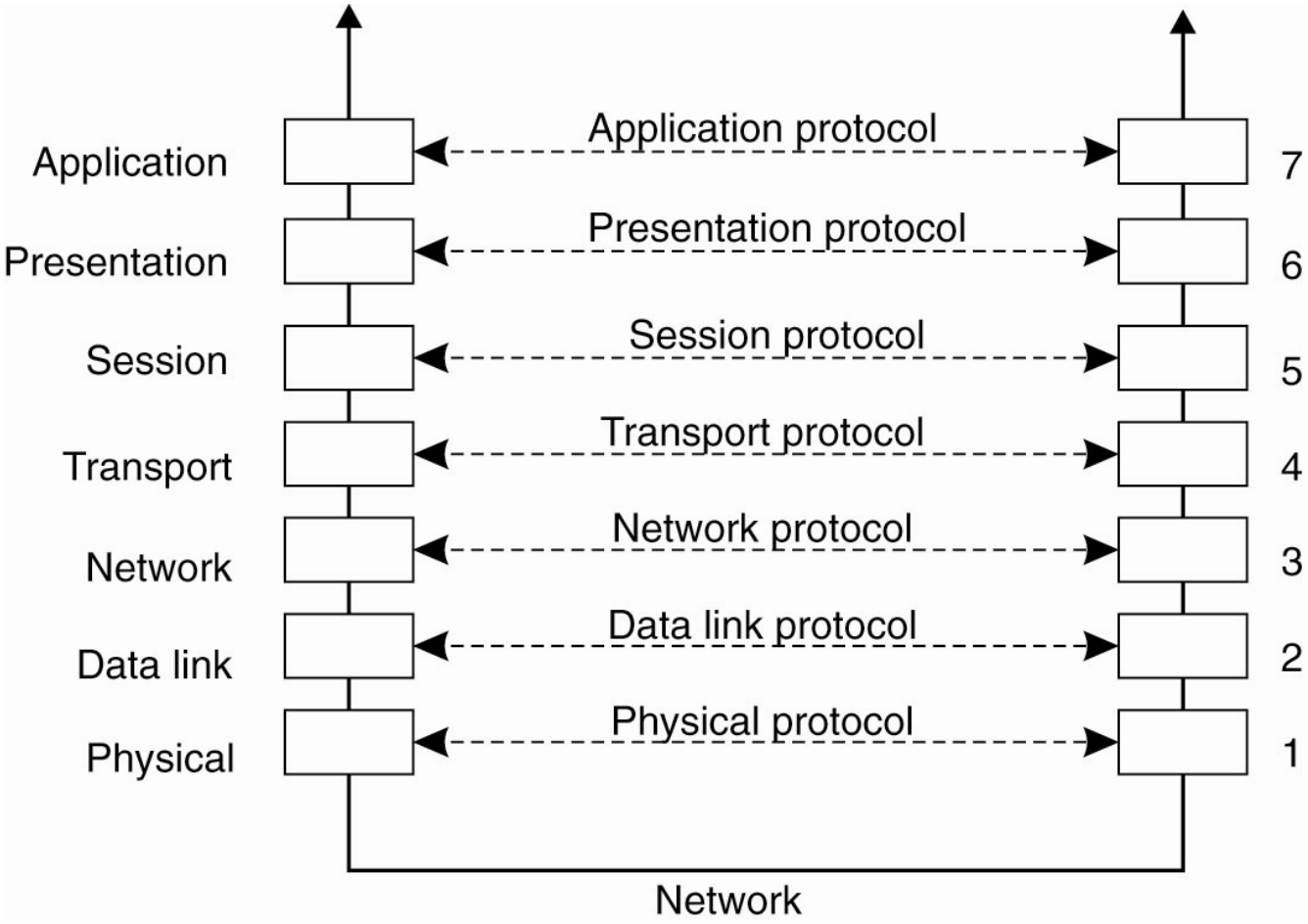


Figure 4-1. Layers, interfaces, and protocols in the OSI model.

Layered Protocols (cont.)

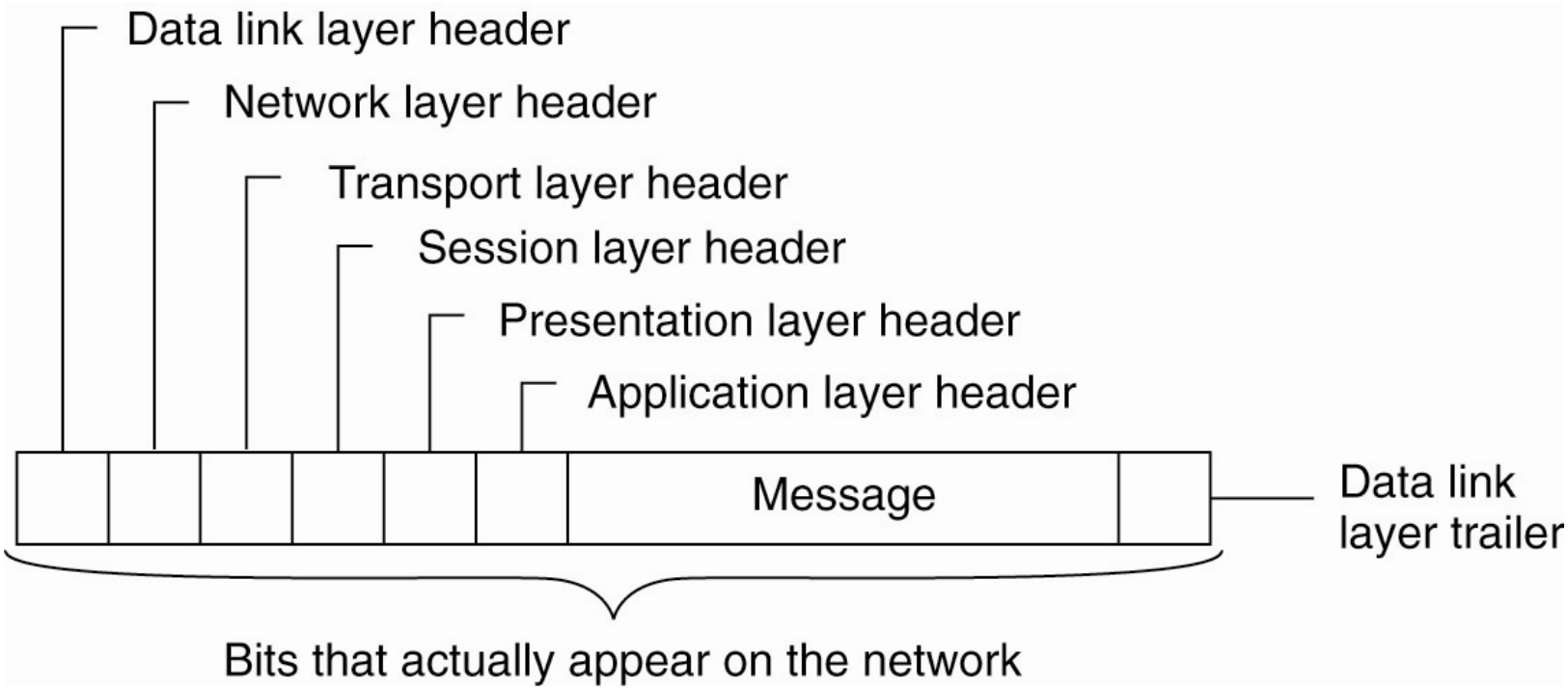


Figure 4-2. A typical message as it appears on the network.

Layers

- Transport
  - Reliable transfer of packets
    - which packets have been sent, which have been received, how many more the receiver has room to accept, which should be transmitted
  - Breaks a message into pieces, assigns each one a sequence number before sending
  - Internet transport protocol
    - TCP (Transmission Control Protocol) – connection oriented, combination of TCP/IP, more reliable and overhead
    - UDP (Universal Datagram Protocol) – connectionless, just IP with slight modifications, less reliable and overhead

Layers (cont.)

- Session
  - An advanced version of transport layer
  - Provides dialogue control – tracks which party is currently talking
  - Provides synchronization facilities – useful for users to insert checkpoints into long transfers to utilize them in case of crash
  - Only in few applications, not present in the Internet protocol suite

Layers (cont.)

- Presentation
  - The lower layers are concerned with transmitting bits reliably and efficiently, this layer is concerned with the meaning of bits
  - Define records in terms of fields (e.g., name, address) and have the sender notify the receiver about record format information -easier for communicating between machines with different internal representations

Layers (cont.)

- Application
  - Contains a collection of network applications – email, file transfer, terminal emulation, and others – File Transfer Protocol (FTP), Hyper Text Transfer Protocol (HTTP)

Middleware Protocols

- Adapted reference model for networked communication – session and presentation layer have been replaced by a single middleware layer
- An application that contains many general purpose protocols independent of other specific applications
- Examples of high level middleware services – RPC, RMI, message queuing services, media transfer through streams

Middleware Protocols (cont.)

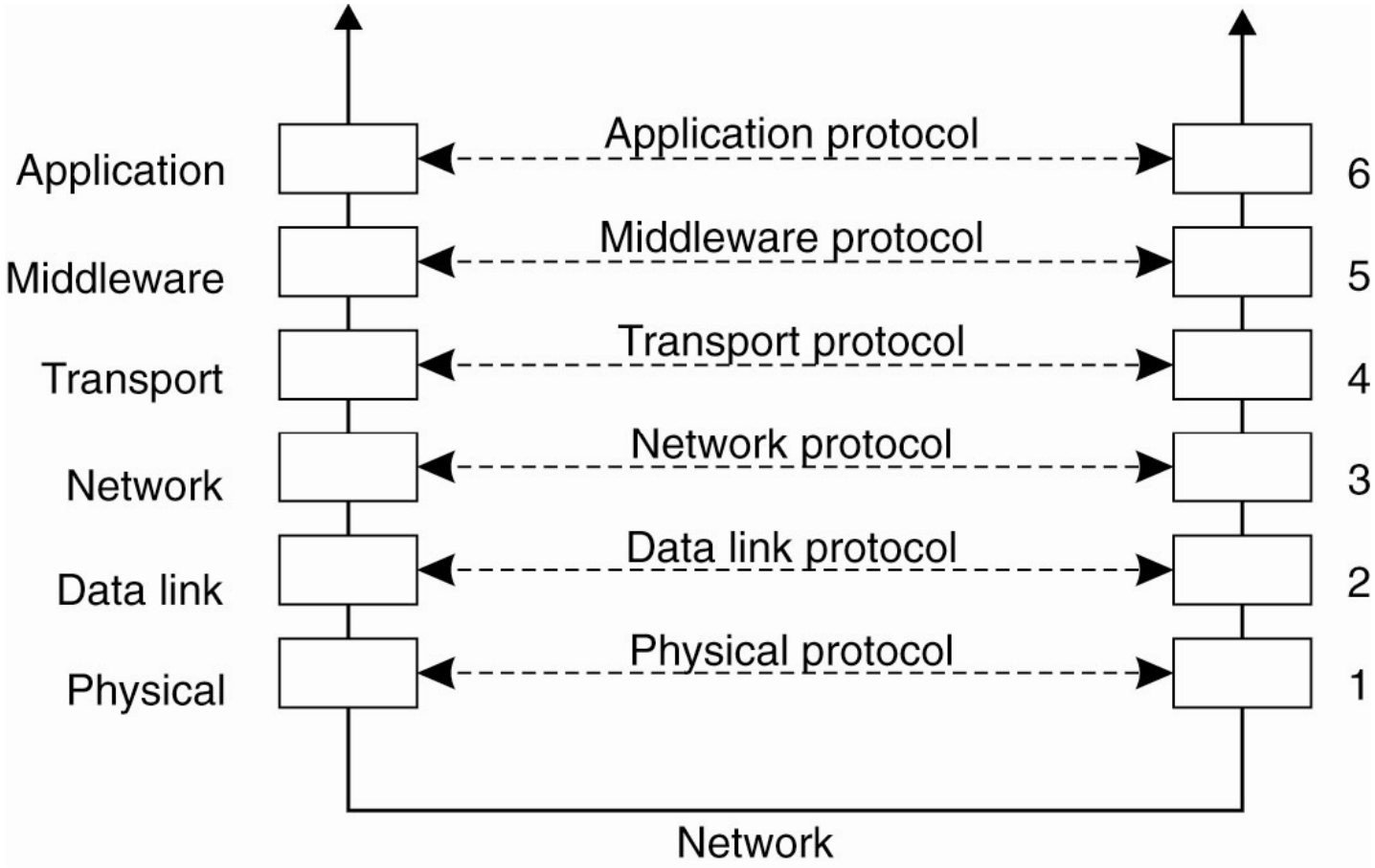


Figure 4-3. An adapted reference model for networked communication.

Types of Communication

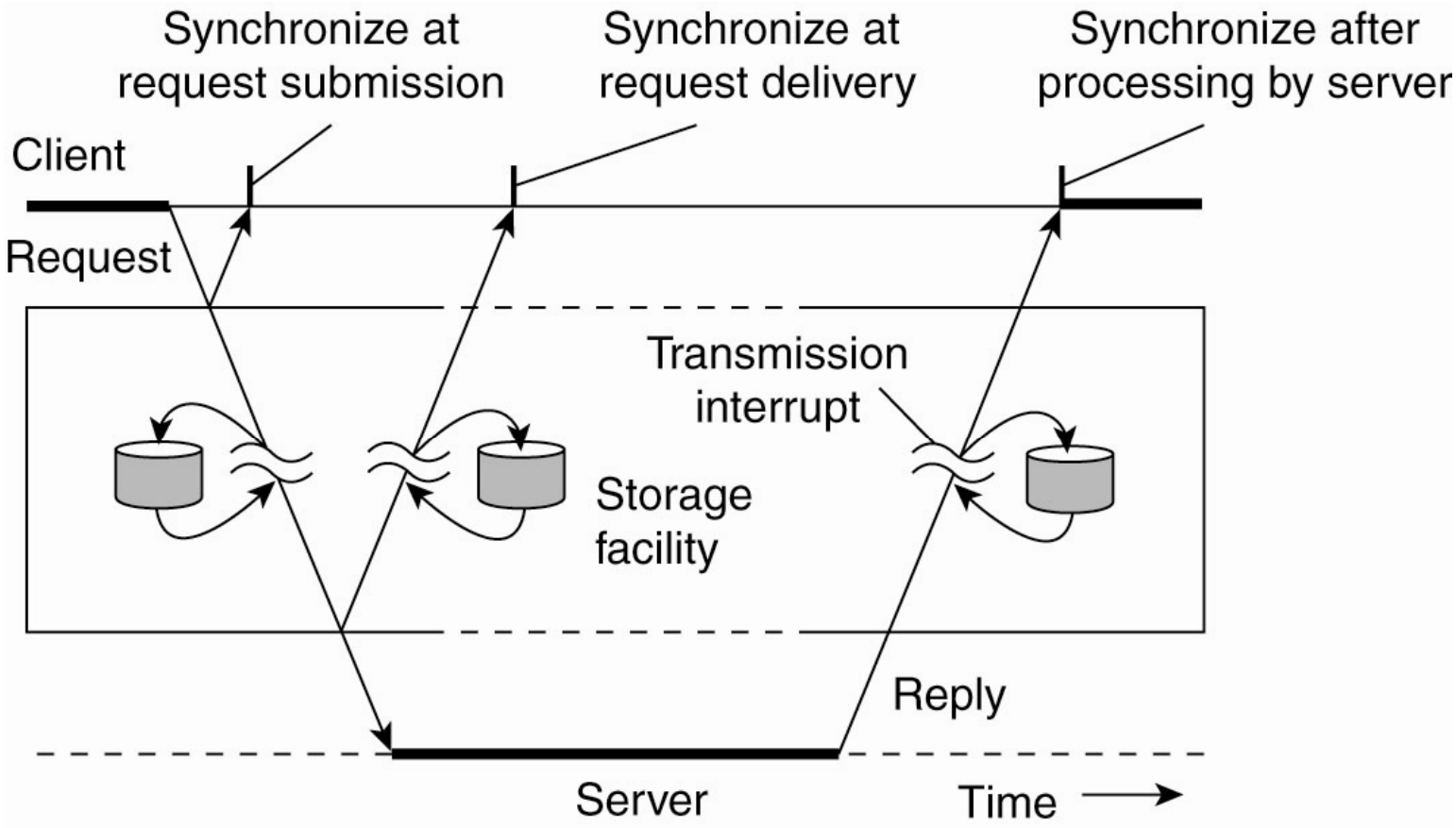


Figure 4-4. Viewing middleware as an intermediate (distributed) service in application-level communication.

Conventional Procedure Call

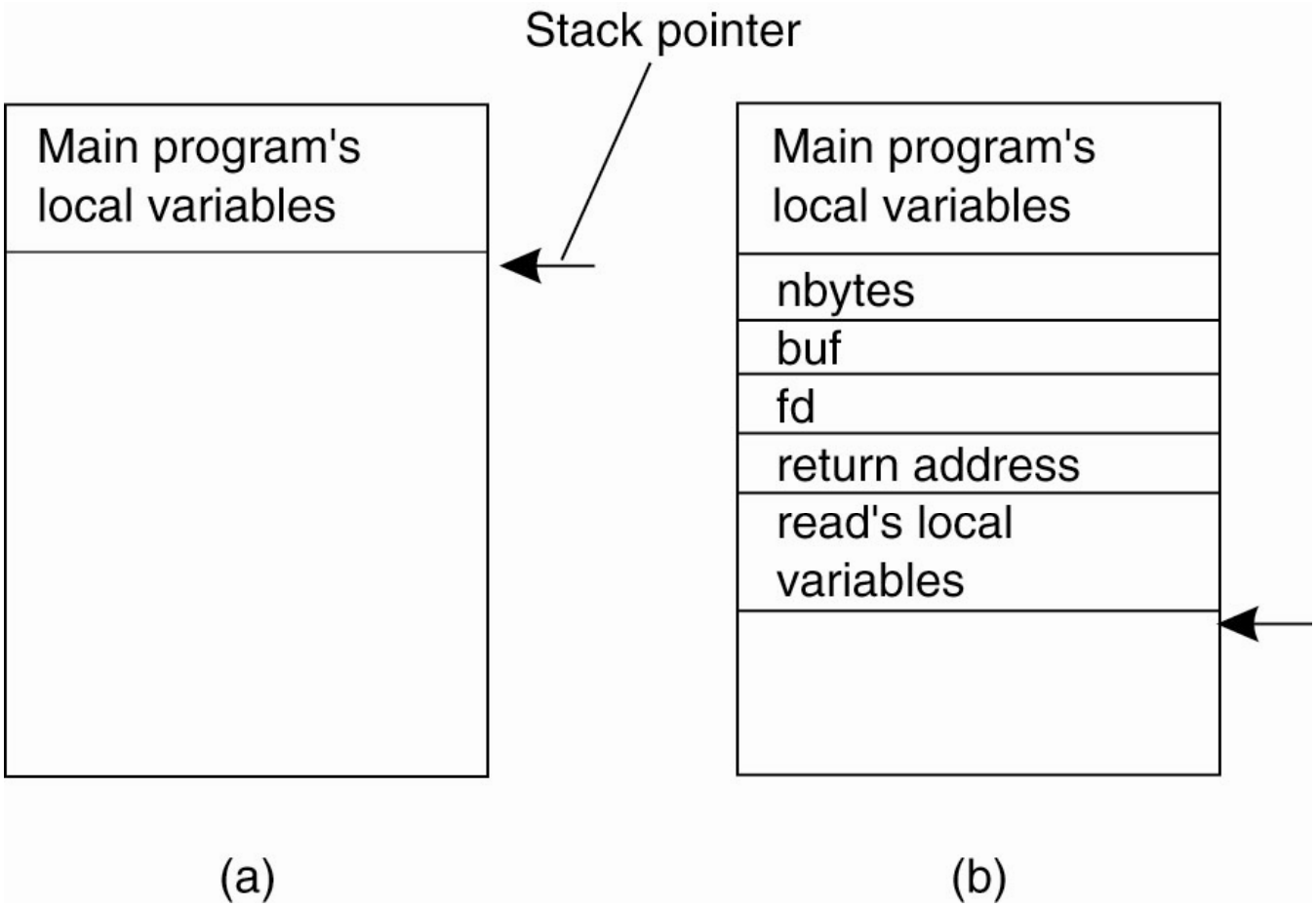


Figure 4-5. (a) Parameter passing in a local procedure call: the stack before the call to read. (b) The stack while the called procedure is active.

Client and Server Stubs

- **Client Stub** - Takes its parameters, packs them into a message (marshalling), and sends them to the server stub
- **Server Stub** – Transforms requests coming in over the network into local procedure calls

Client and Server Stubs

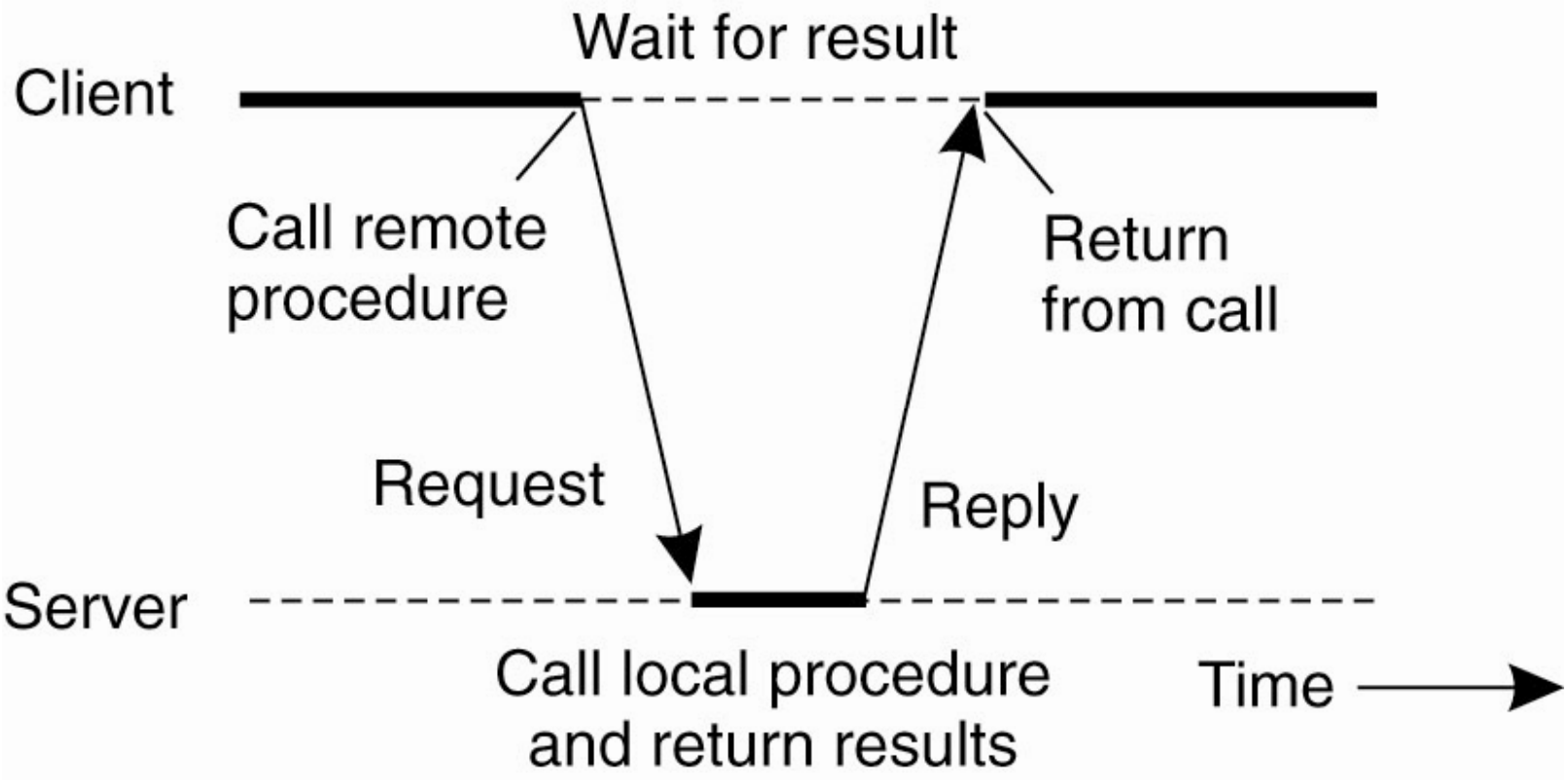


Figure 4-6. Principle of RPC between a client and server program.

Remote Procedure Calls

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.

Continued …

Remote Procedure Calls (cont.)

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.

Passing Value Parameters

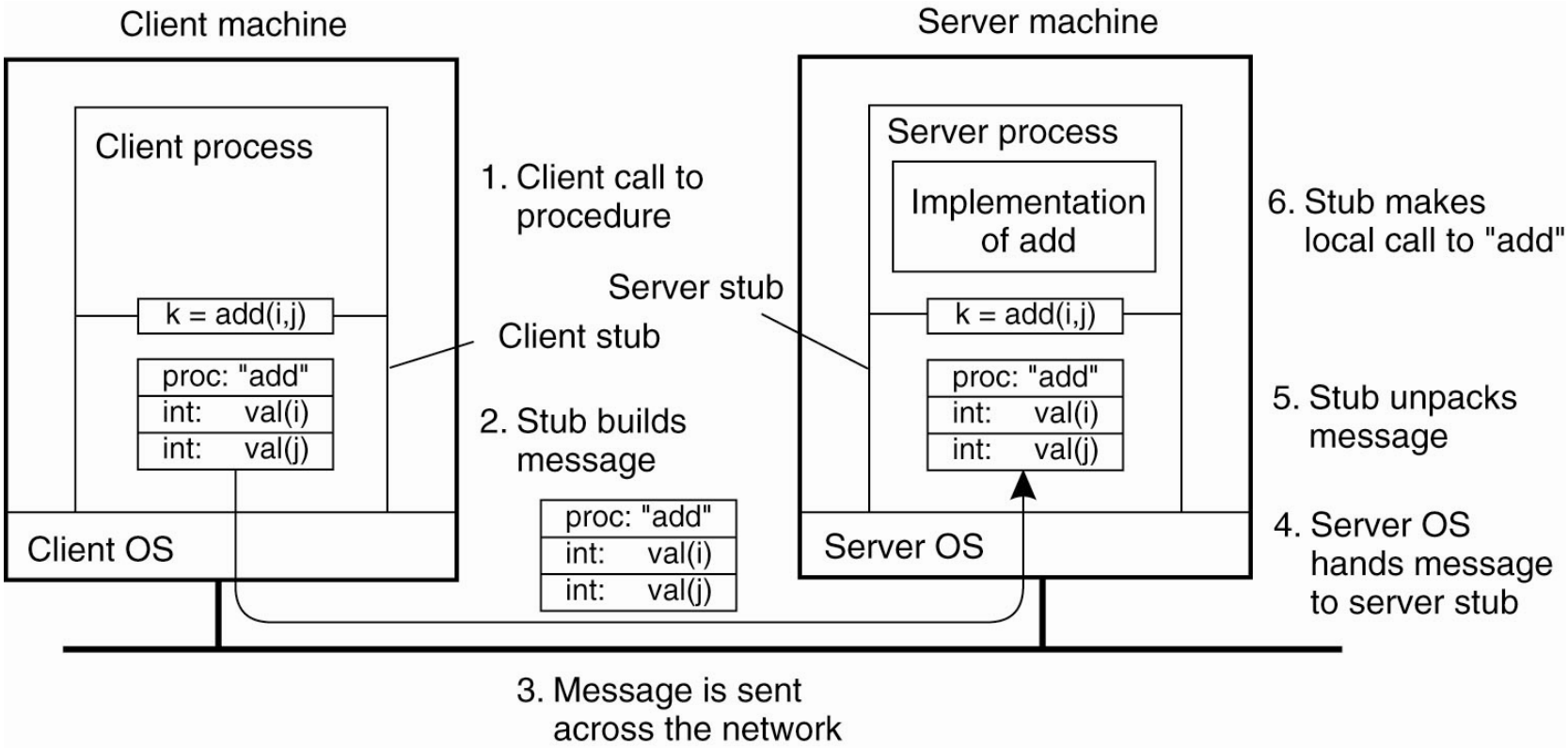


Figure 4-7. The steps involved in a doing a remote computation through RPC.

Passing Value Parameters (cont.)

- Passing as value
  - Representation of data structures such as integers, characters
  - Order in numbering bytes
- An Example Problem: Pentium -> SPARC
  - Intel Pentium number their bytes from right to left while (little endian) Sun SPARC (big endian) numbers in the opposite way

Passing Value Parameters (cont.)

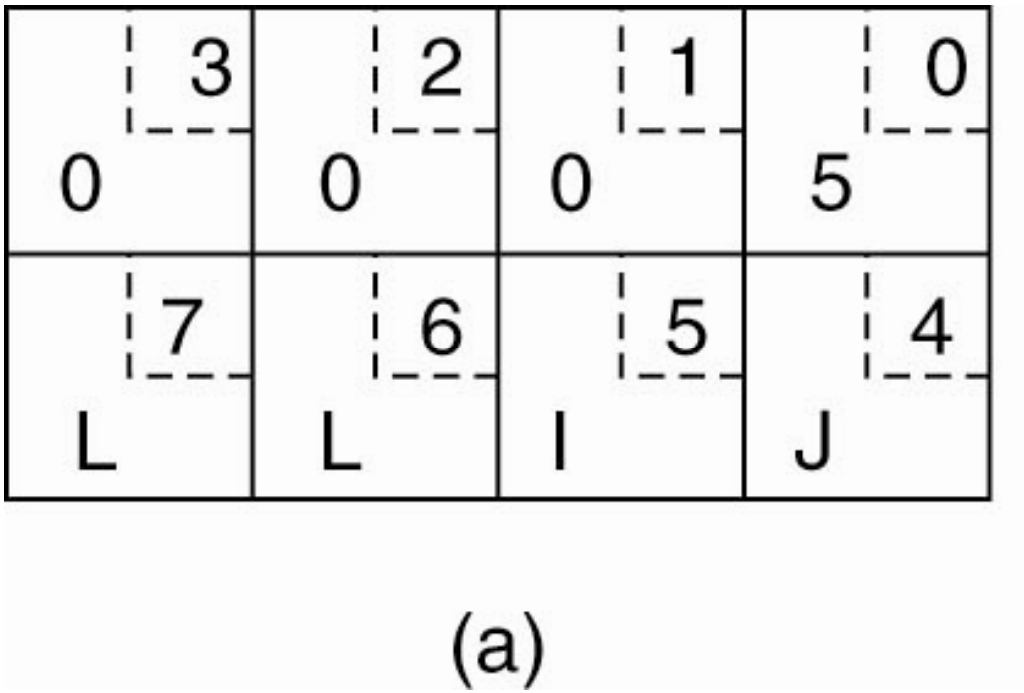


Figure 4-8. (a) The original message on the Pentium.

Passing Value Parameters (cont.)

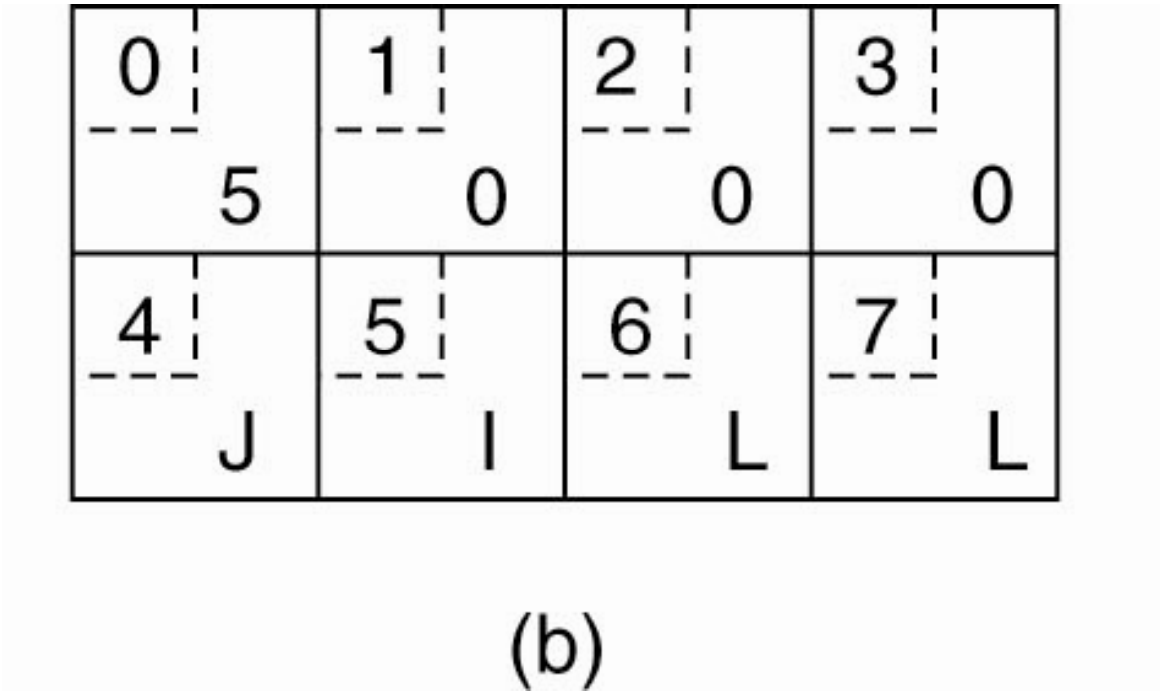


Figure 4-8. (a) The original message on the Pentium.

Passing Value Parameters (cont.)

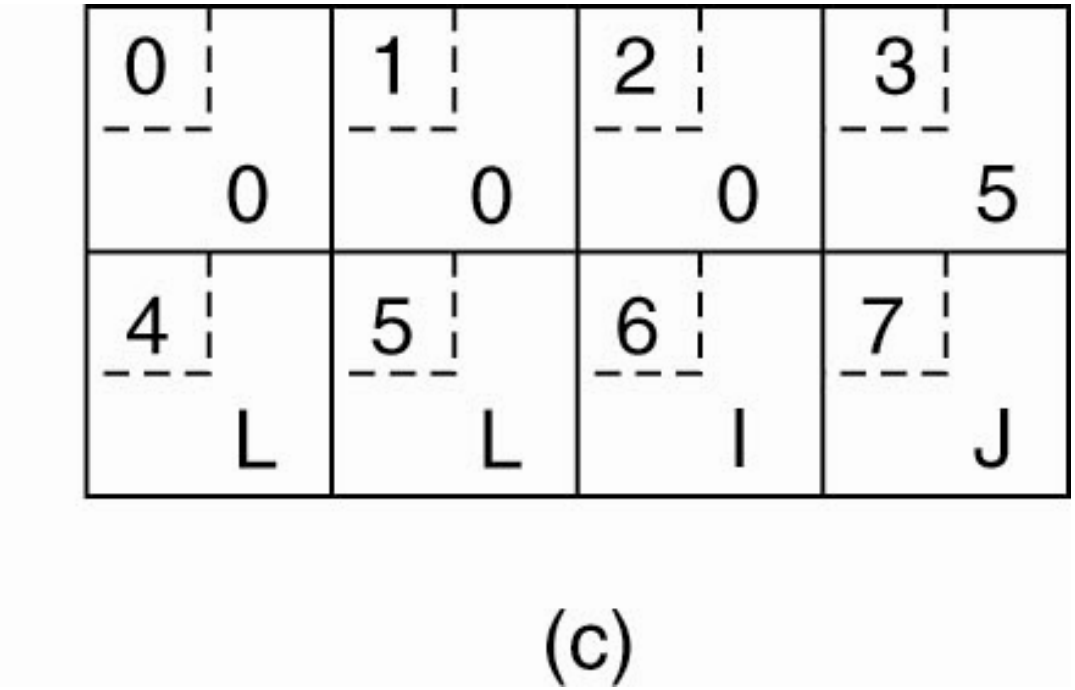
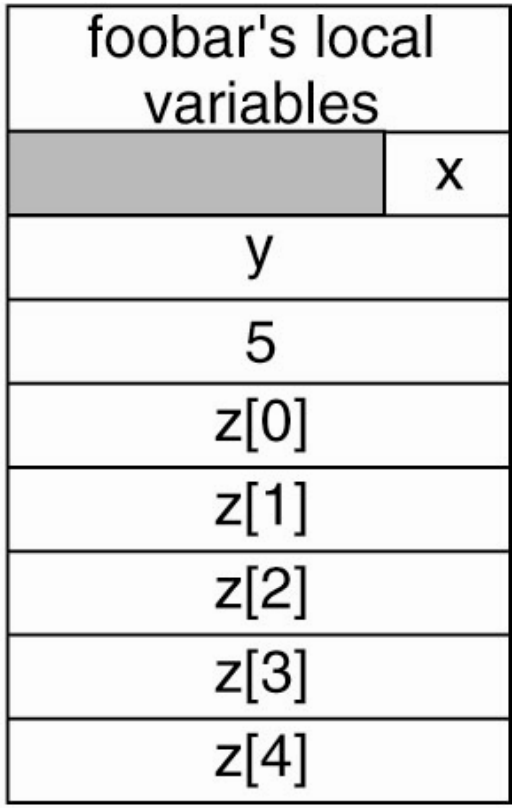


Figure 4-8. (c) The message after being inverted. The little numbers in boxes indicate the address of each byte.

Parameter Specification and Stub Generation

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

(a)



(b)

Figure 4-9. (a) A procedure. (b) The corresponding message.

Asynchronous RPC

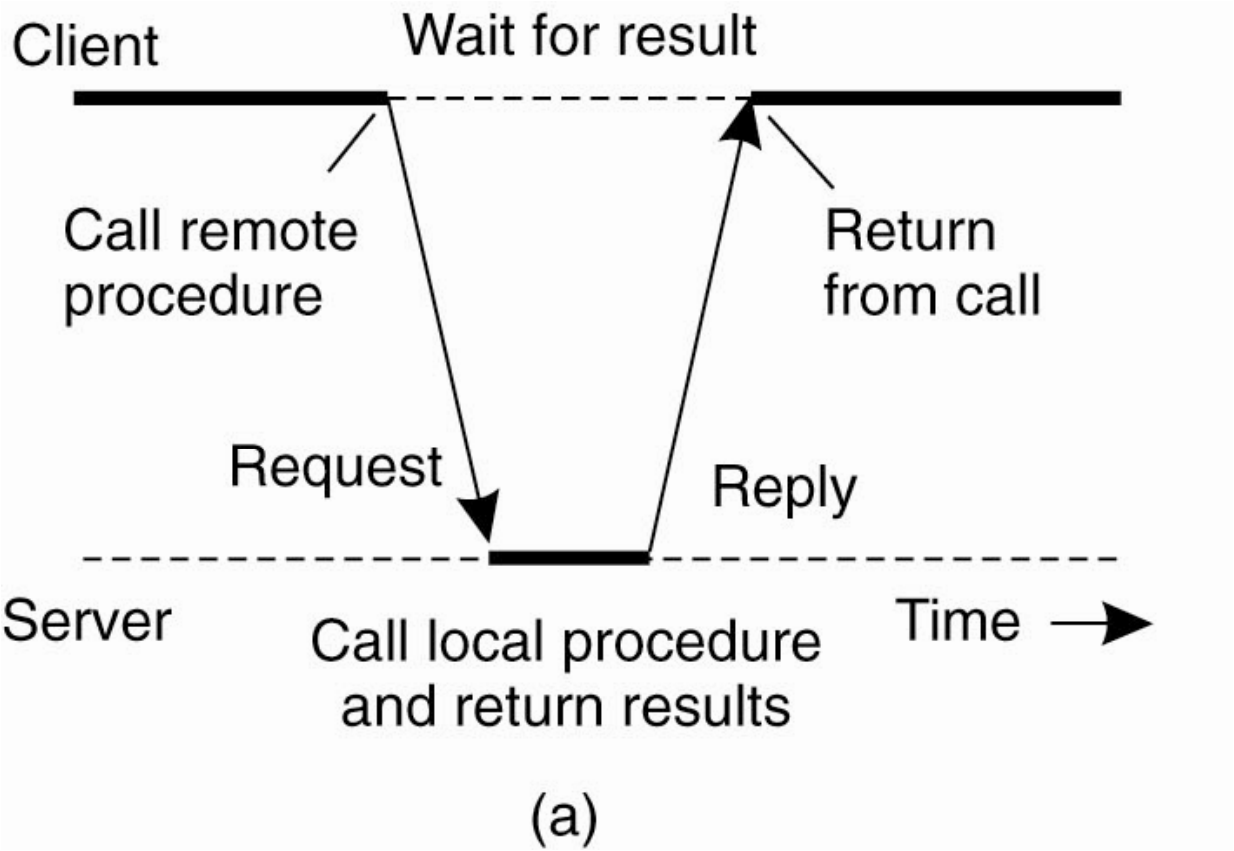


Figure 4-10. (a) The interaction between client and server in a traditional RPC.

Asynchronous RPC (cont.)

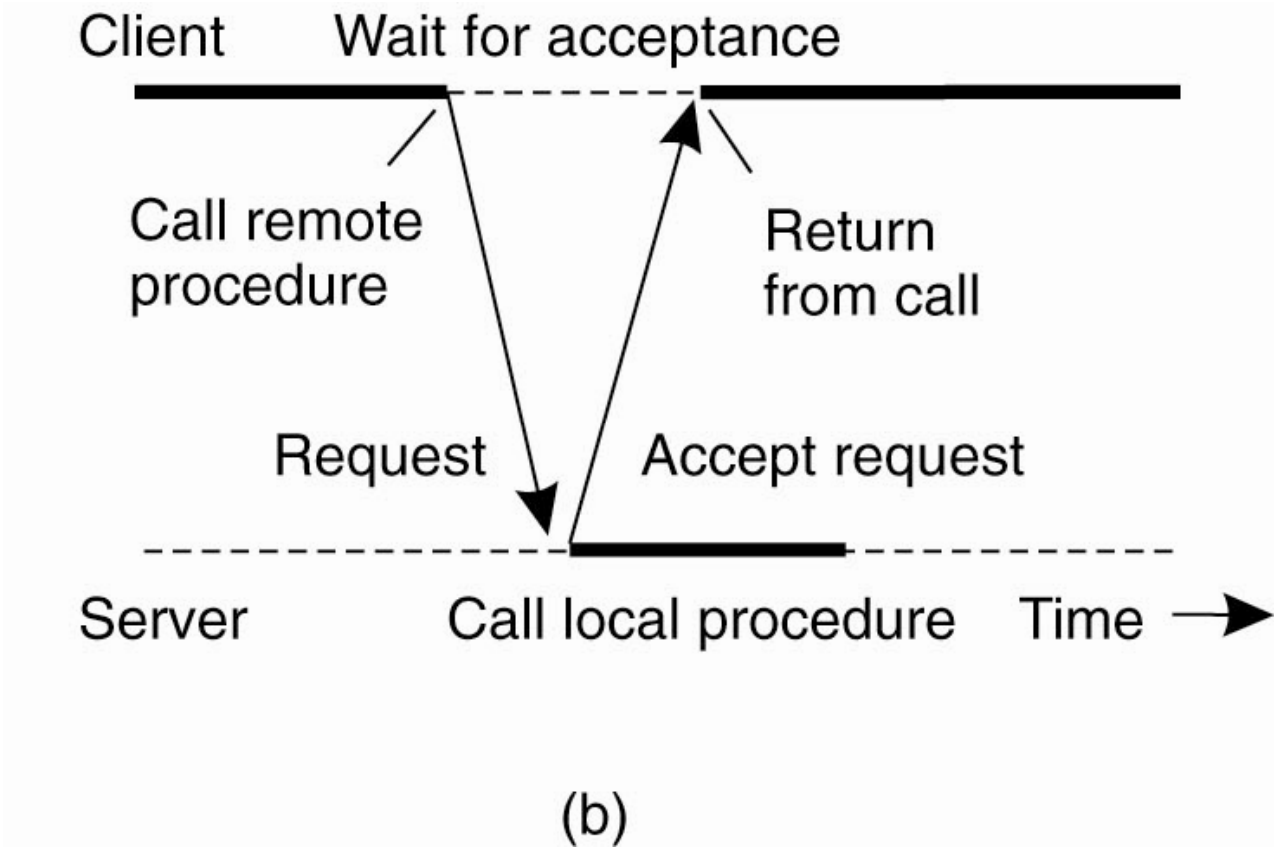


Figure 4-10. (b) The interaction using asynchronous RPC.

Asynchronous RPC (cont.)

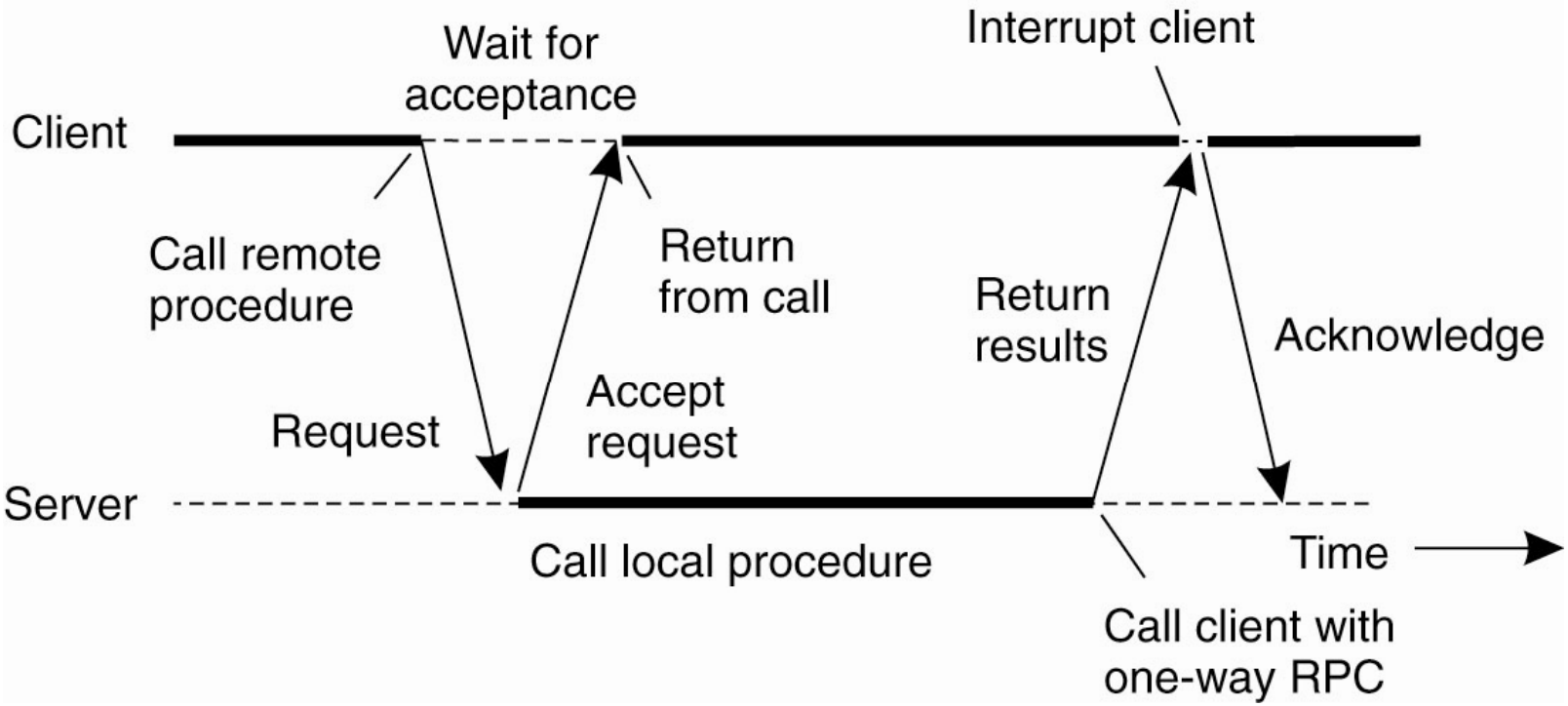


Figure 4-11. A client and server interacting through two asynchronous RPCs.

Binding a Client to a Server

- 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 (cont.)

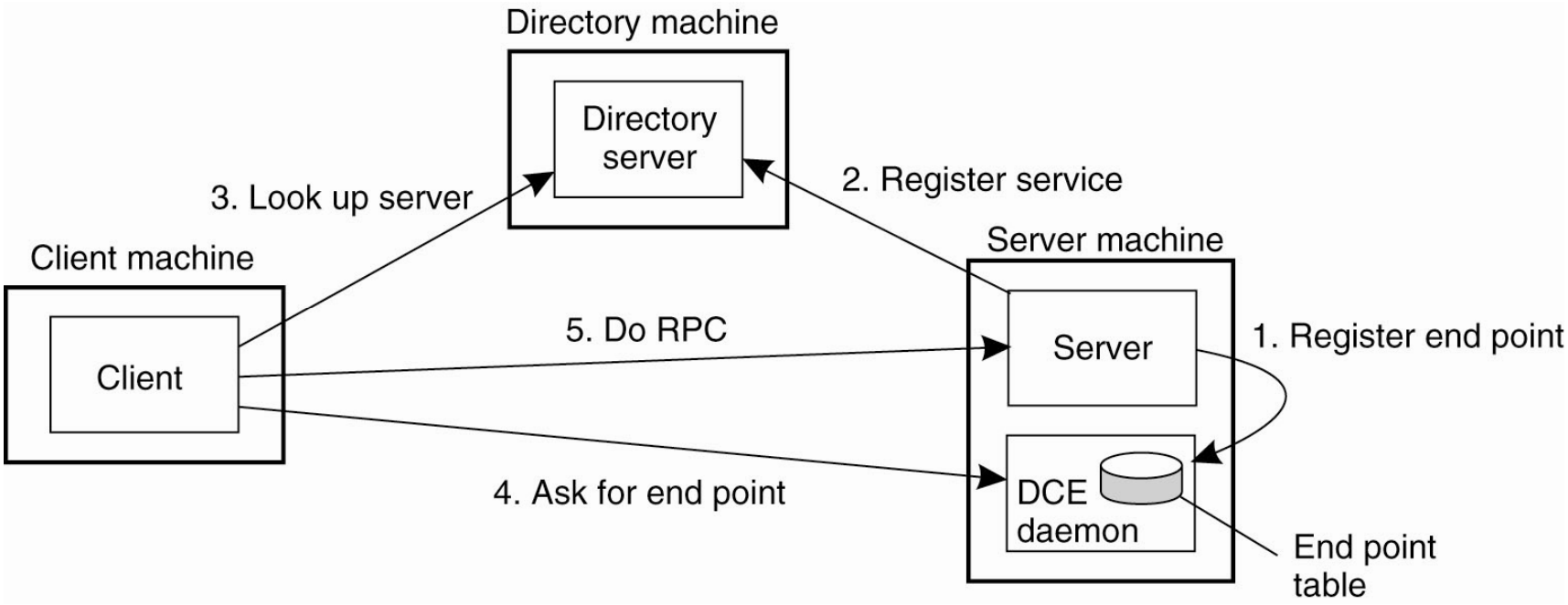


Figure 4-13. Client-to-server binding in DCE.

Message Oriented Transient Communication

- Message Oriented Communication
  - Message Oriented Transient Communication
  - Message Oriented Persistent Communication

- Example of Message Oriented Transient Communication - **Sockets**
  - A communication end point for an application to write (send) and read (receive) data to/from the underlying network
  - An abstraction over the actual communication end point that is used by the local OS for a specific transport protocol

Berkeley Sockets

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

Figure 4-14. The socket primitives for TCP/IP.

Berkeley Sockets (cont.)

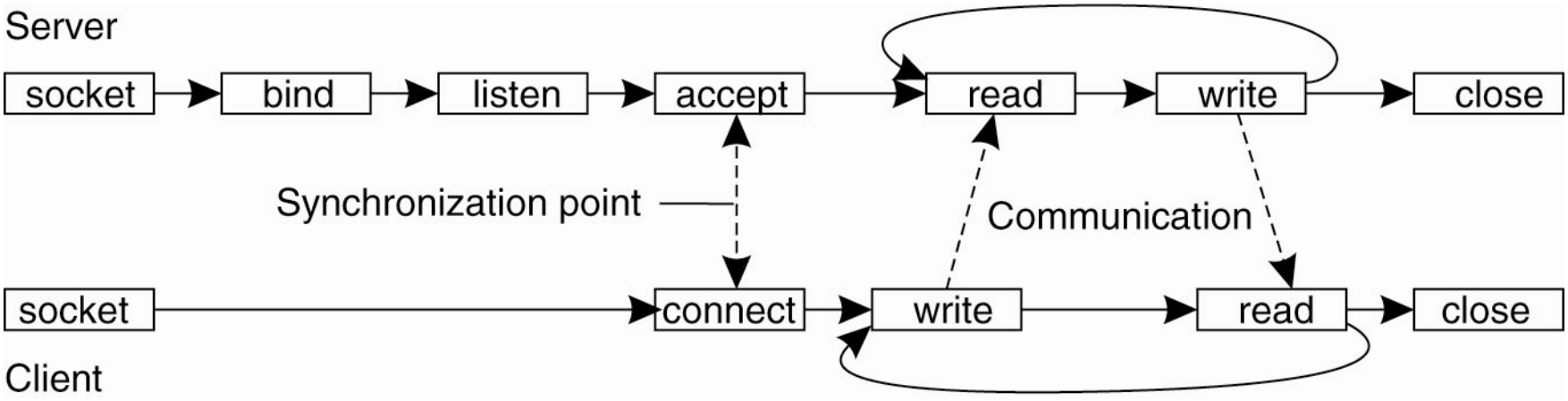


Figure 4-15. Connection-oriented communication pattern using sockets.

The Message-Passing Interface

- **MPI**
  - Platform independent standard for message passing
  - Directly uses the underlying networks – no communication servers
  - Assumes that communications take place within known group of processes
  - Each process is identified by an ID (groupID, processID)
  - The above address is used to send a message instead of a transport-level address
  - Assumes no recovery from serious failures such as process crashes or network partitions

The Message-Passing Interface (cont.)

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

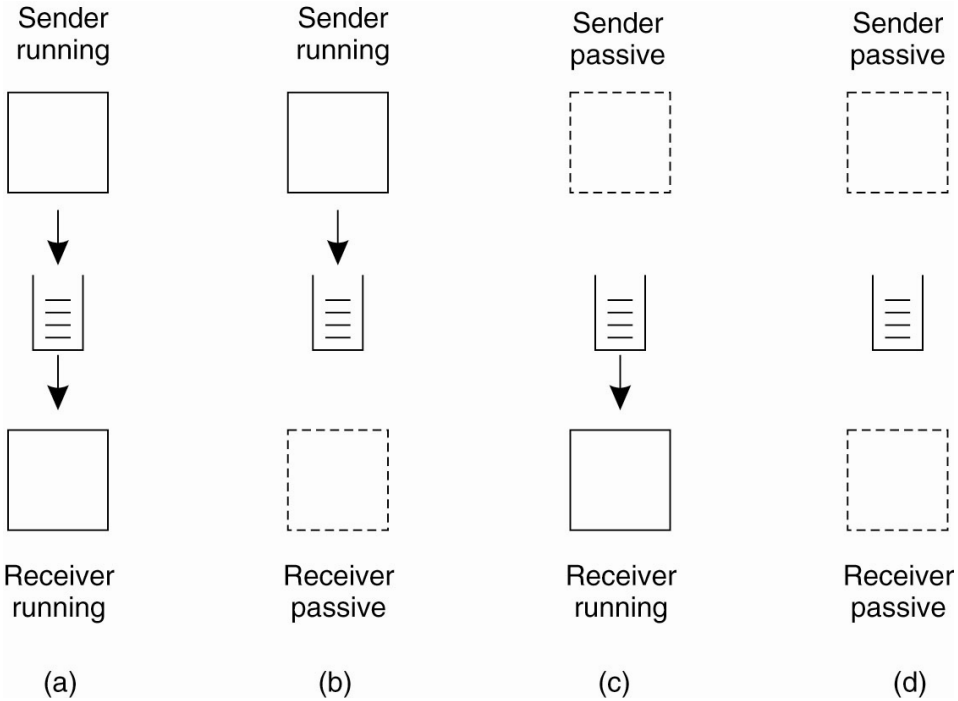
Figure 4-16. Some of the most intuitive message-passing primitives of MPI.

Message Oriented Persistent Communication

- Message Queuing Model or Message-Oriented Middleware (MOM)
  - Intermediate storage capacity for messages – do not require either the sender or receiver to be active during message transmission
  - Intended to support message transfers that are allowed to take minutes instead of sec/ms (contrary to Berkley Sockets and MPI)

Message-Queuing Model

- The sender is usually given only the guarantees that its message will eventually be inserted in the receiver's queue
- No guarantees are made about when, or even whether the message will actually be read
- Four combinations



Message-Queuing Model (cont.)

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

Figure 4-18. Basic interface to a queue in a message-queuing system.

Message-Queuing System

- Transfers messages from a source to a destination
  - **Source queue** – A message can be put only into this local queue, and the message has a destination queue address
  - **Destination queue** - A message can be read only from here
  - **Queue names** – A database of queue names – maintained by queuing system
  - **Queue managers** –interact directly with the application that is sending or receiving a message
  - **Special queue managers** - Relays/Routers – forward incoming messages to other queue manager

General Architecture of a Message-Queuing System

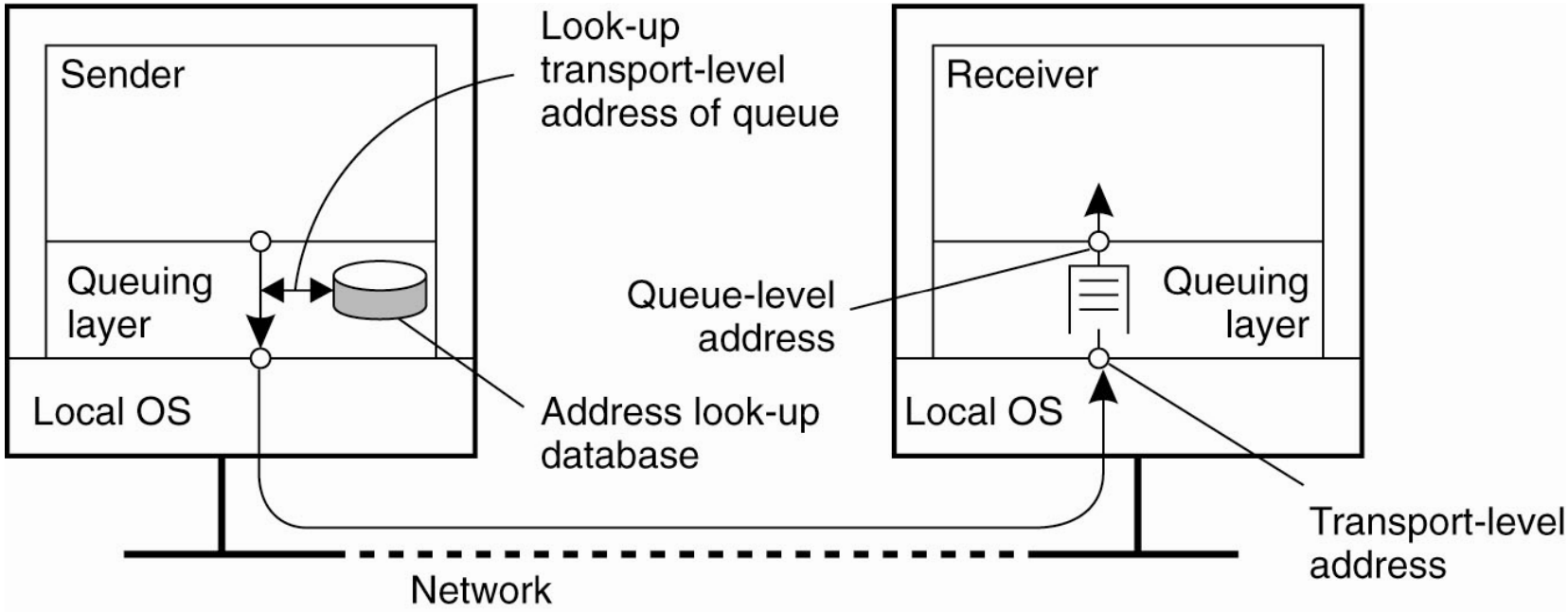


Figure 4-19. The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System (cont.)

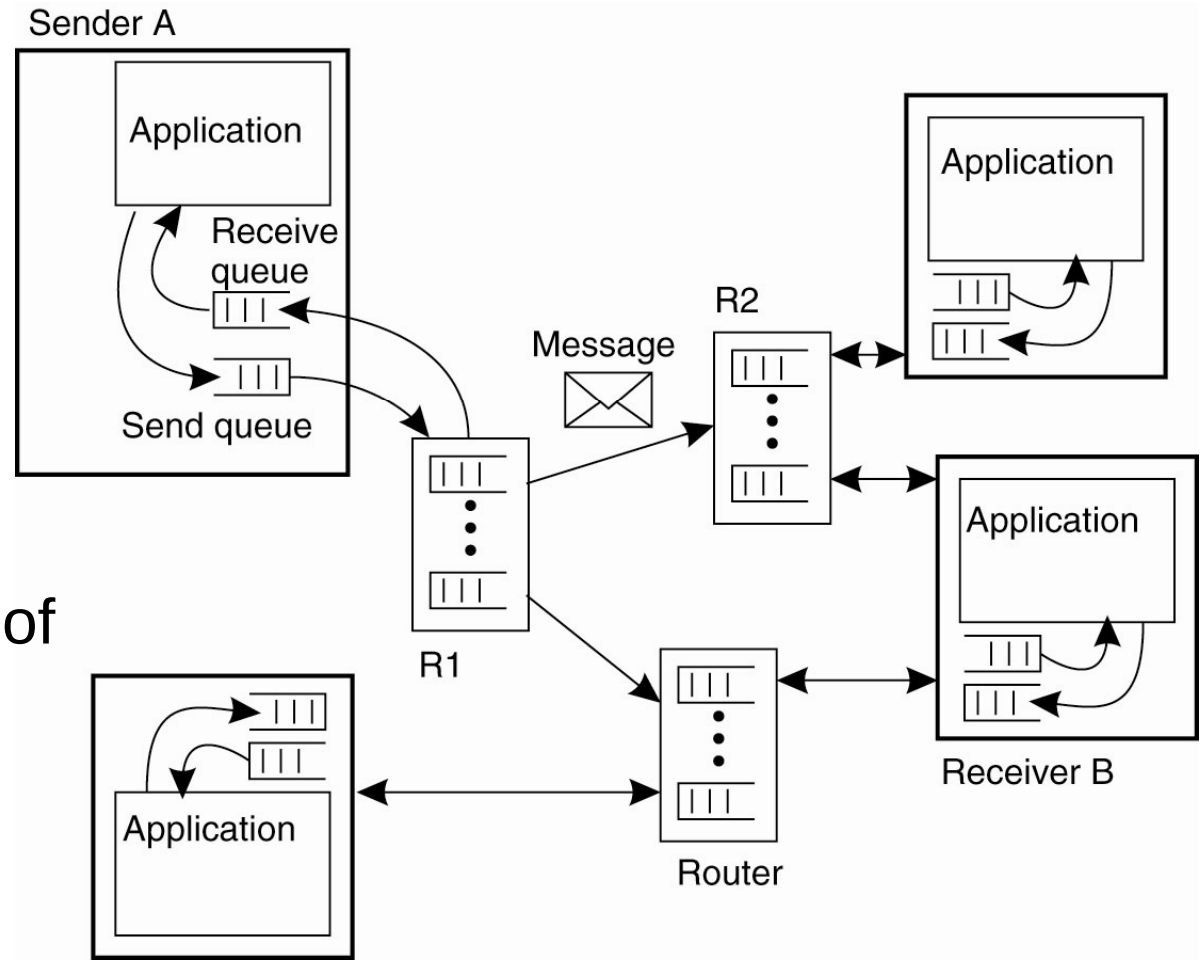
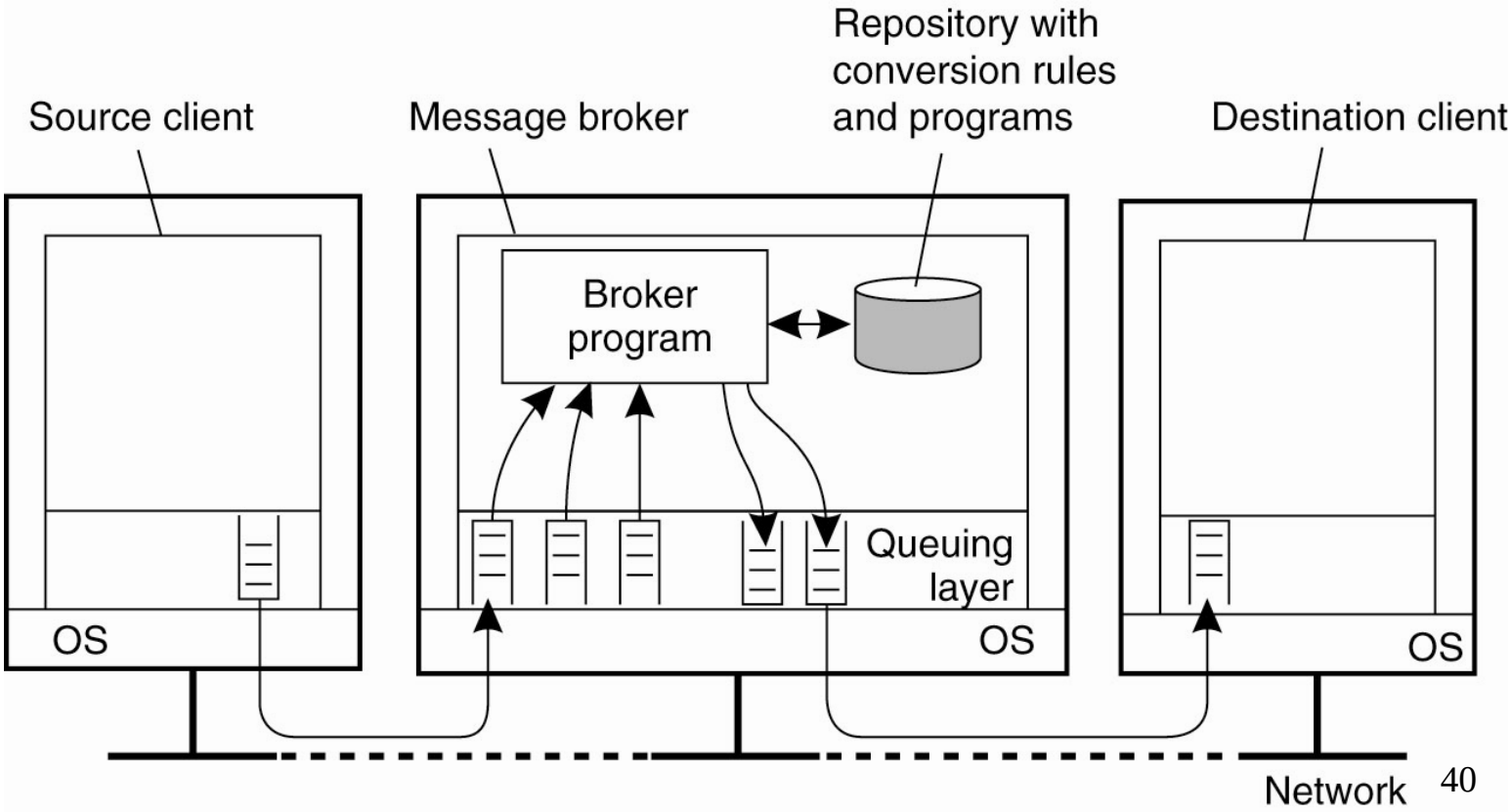


Figure 4-20. The general organization of a message-queuing system with routers.

Message Brokers

- **Message Broker:** A special node in a message-queuing system that handles the conversation
- **Main task:** convert incoming messages to a format that can be understood by the receiver application



Data Stream

- **Data Stream** – a sequence of data units
  - **Simple stream** –consists of only a sequence of data
  - **Complex stream** – consists of several related simple streams called sub-streams – e.g., transmitting a movie

- **Types of Media**
  - **Continuous media** –the temporal relationships between different data items are fundamental to correctly interpreting the data –motion pictures require that successive images display at a uniform spacing T in time
  - **Discrete media** – temporal relationships of between data items are not fundamental to correctly interpreting the data – text, still images, executable files

Data Stream

- **Asynchronous**
  - Data items are in a stream are transmitted one after the other, but there are no further timing constraints on when transmission of items should take place
  - Specifically true for discrete data streams – e.g., file transfer
- **Synchronous**
  - Maximum end-to-end delay defined for each unit in a data stream
  - A data unit may be transferred much faster than the maximum tolerated delay
  - E.g., a sensor sampling temperature at a certain rate and send it via network
- **Isochronous**
  - Data units are constrained by maximum and minimum end-to-end delay – distributed multimedia system (audio and video)

Data Stream

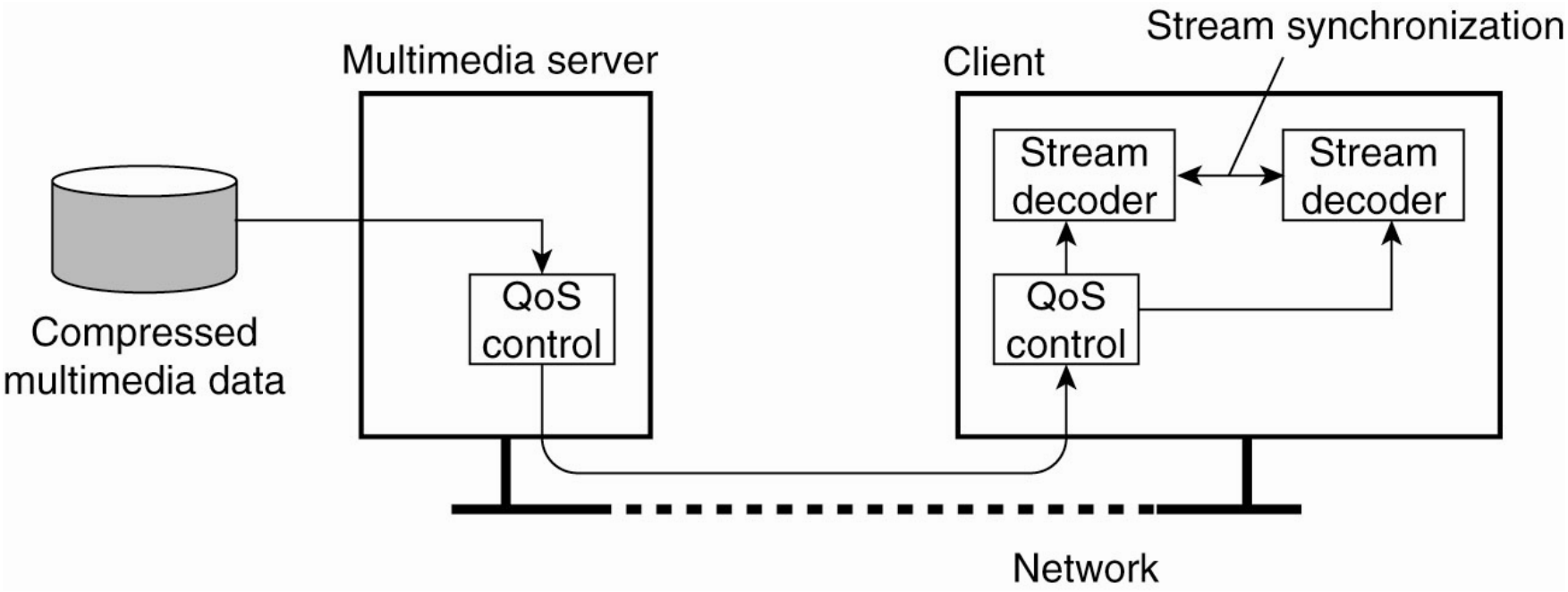


Figure 4-26. A general architecture for streaming stored multimedia data over a network.

Streams and Quality of Service

Properties for Quality of Service:

- The required bit rate at which data should be transported.
- The maximum delay until a session has been set up
- The maximum end-to-end delay .
- The maximum delay variance, or jitter.
- The maximum round-trip delay.

Enforcing QoS

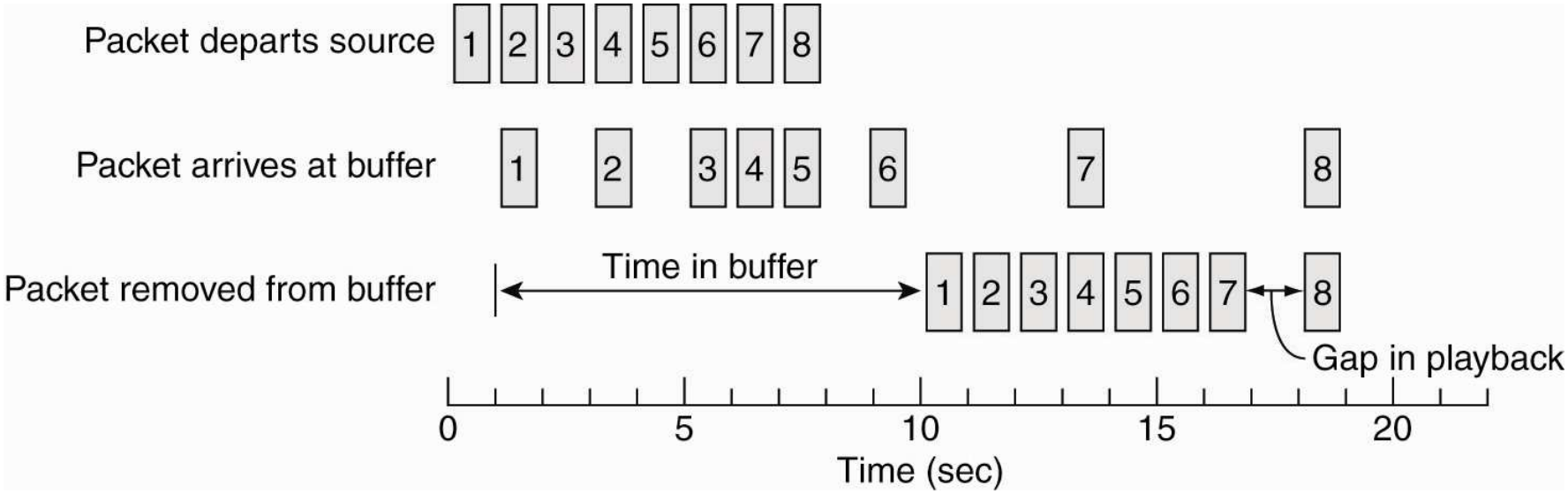


Figure 4-27. Using a buffer to reduce jitter.

Enforcing QoS (cont.)

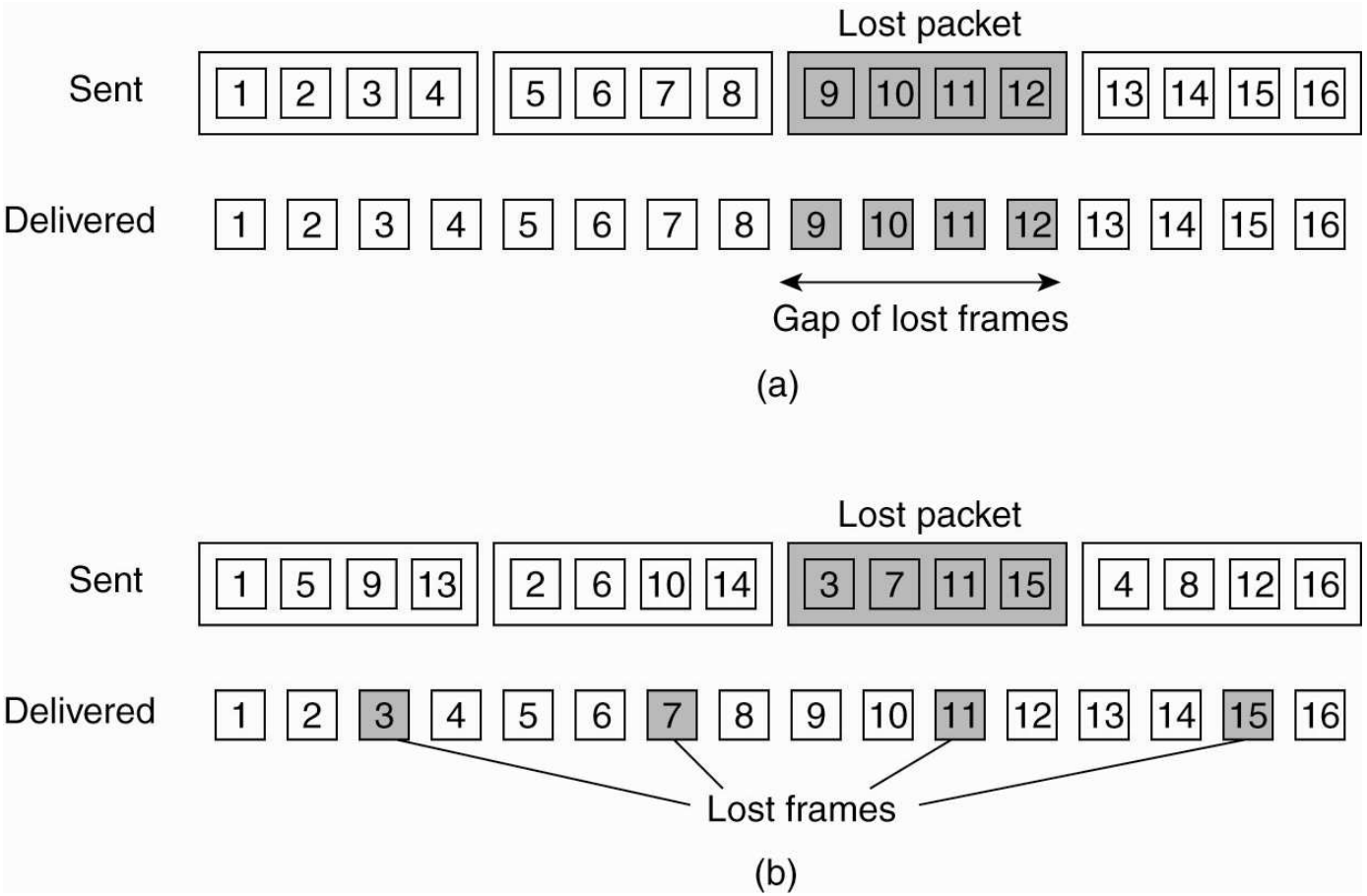


Figure 4-28. The effect of packet loss in (a) non interleaved transmission and (b) interleaved transmission.

Synchronization Mechanisms

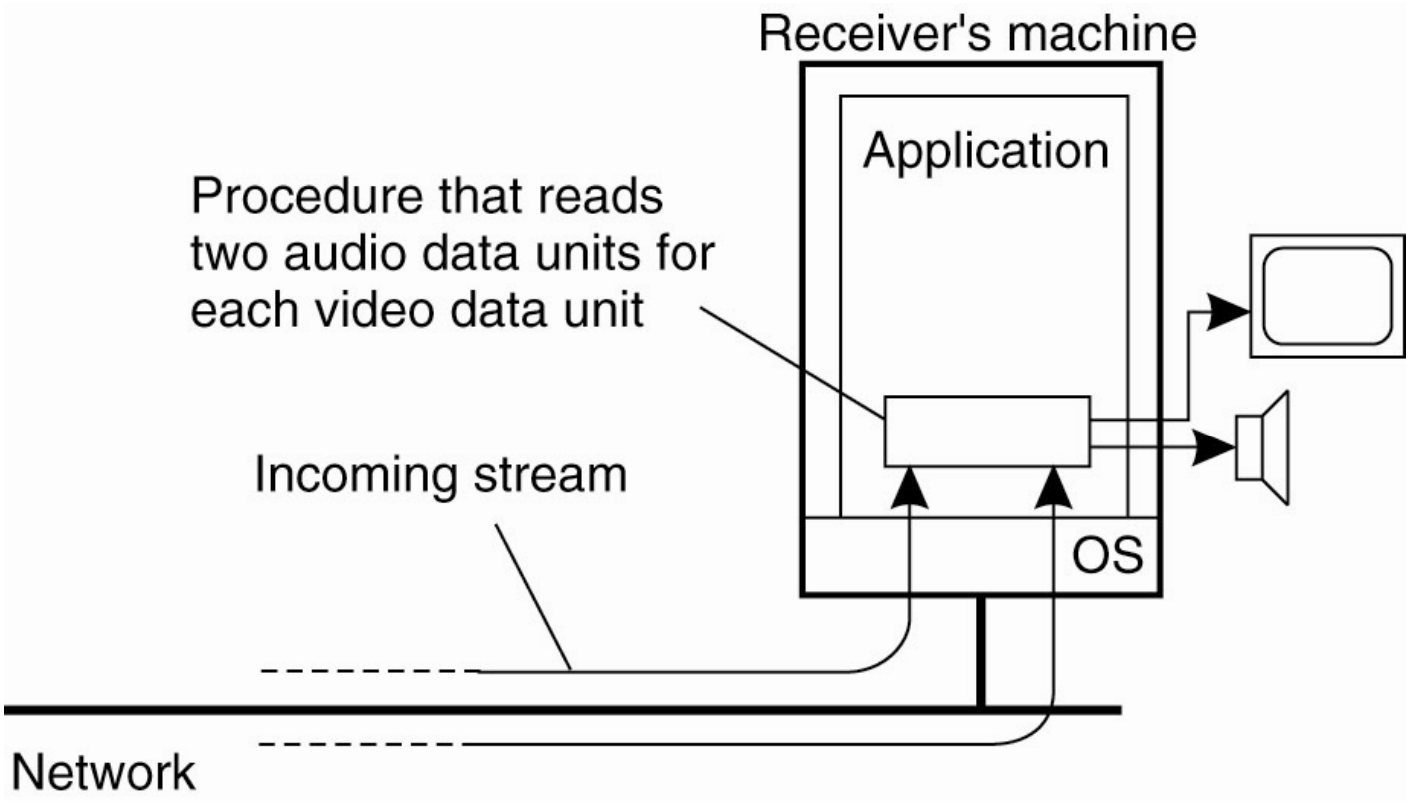


Figure 4-29. The principle of explicit synchronization on the level data units.

Synchronization Mechanisms (cont.)

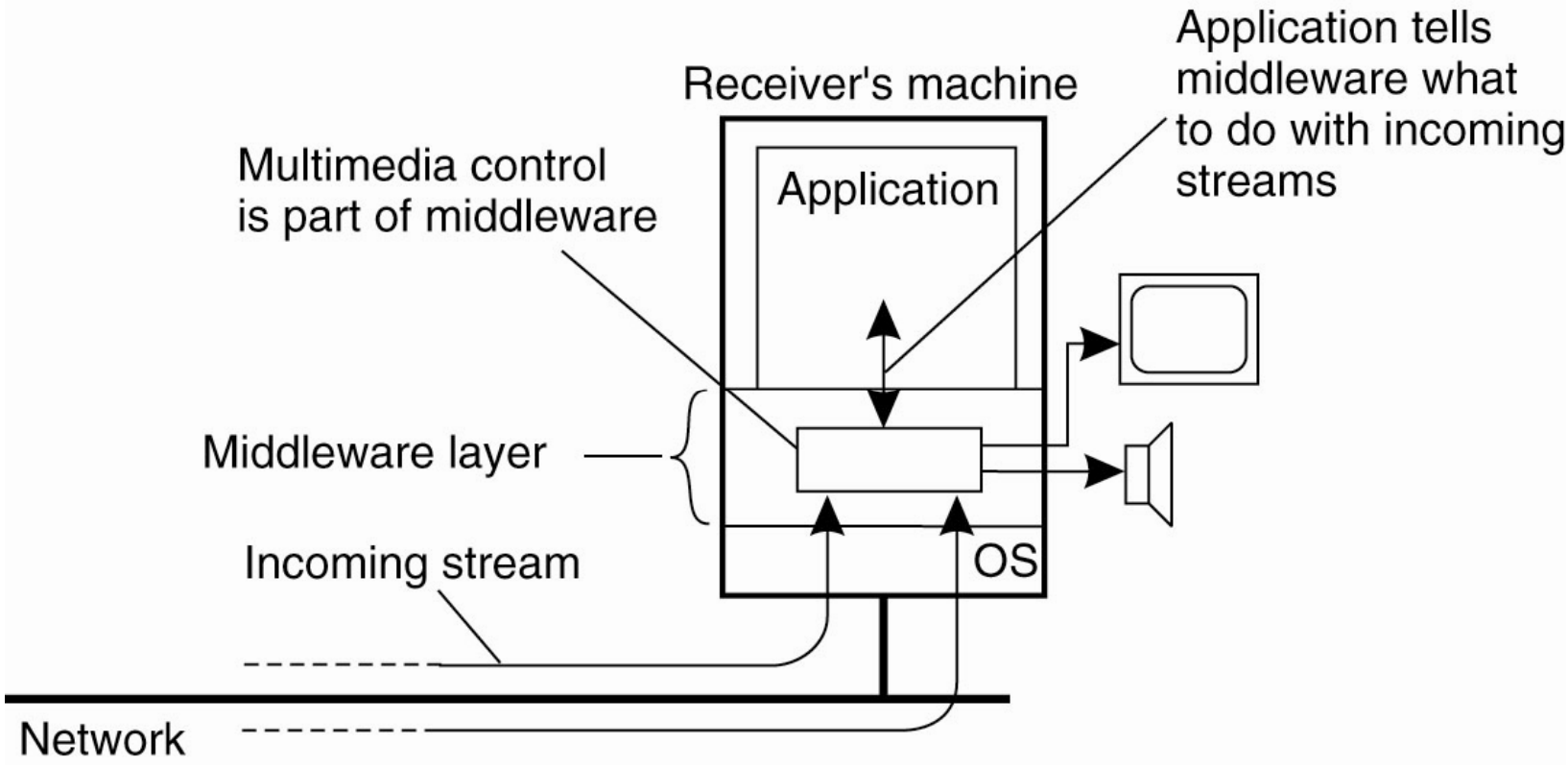


Figure 4-30. The principle of synchronization as supported by high-level interfaces.