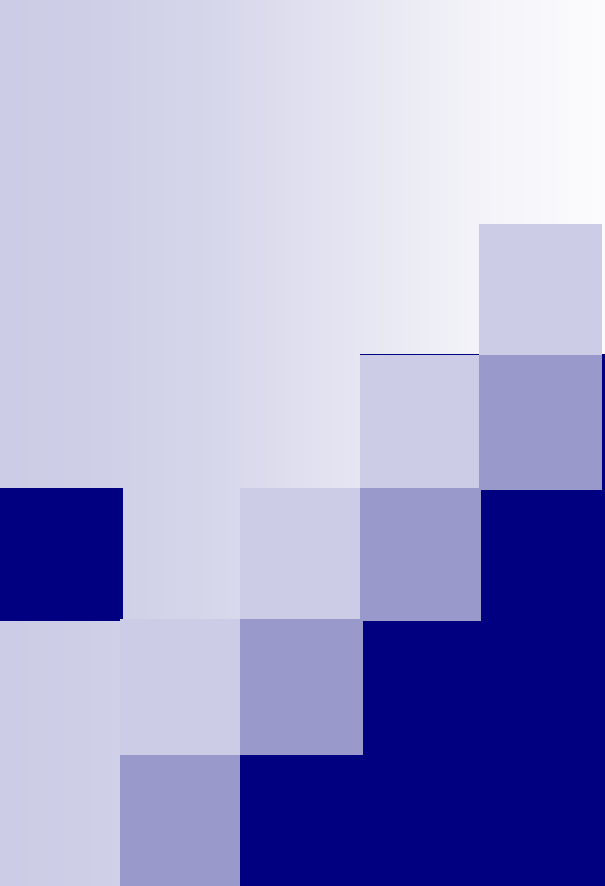




Distributed Systems

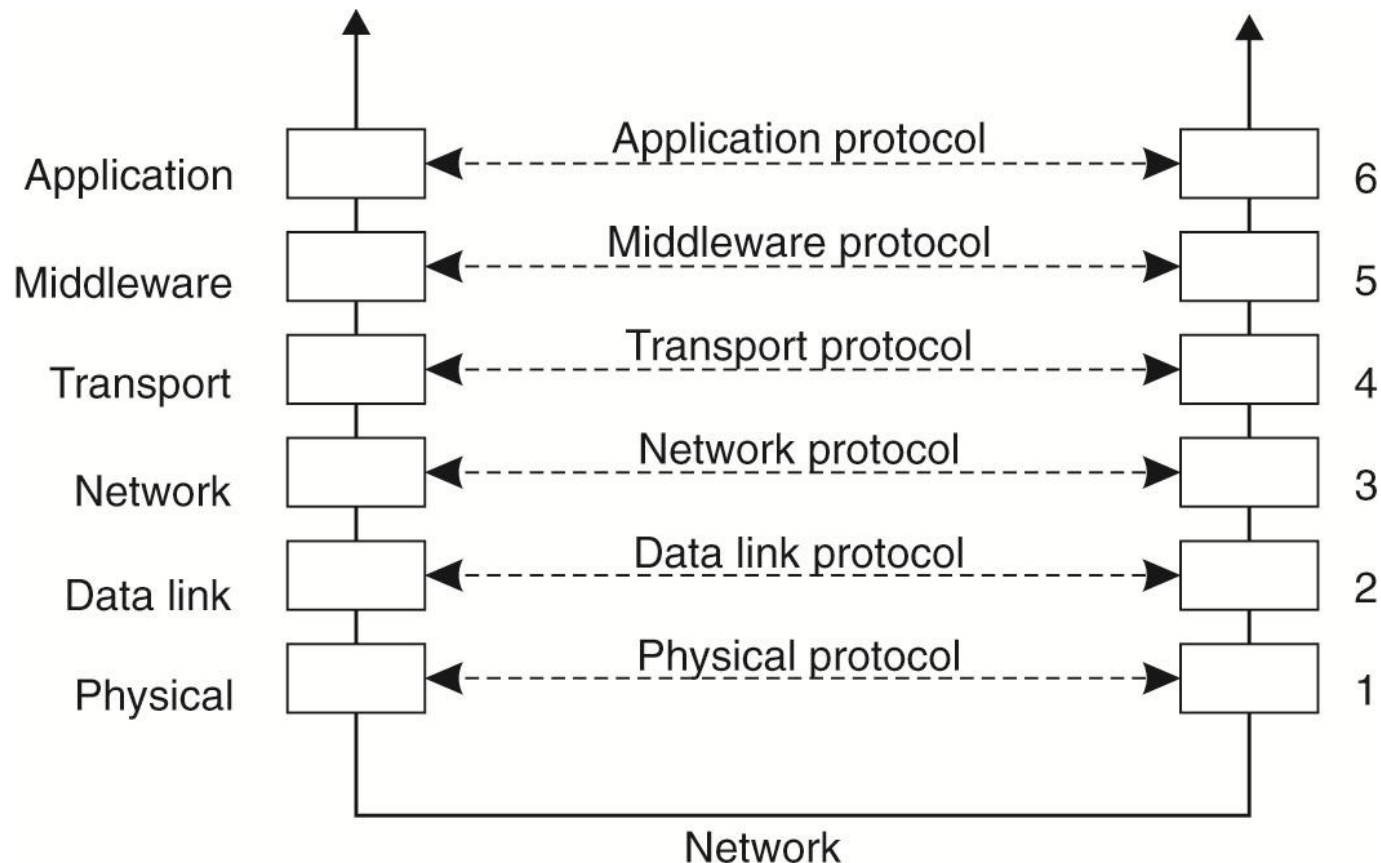
Lecture 4

Professor Louise E. Moser

Spring 2013

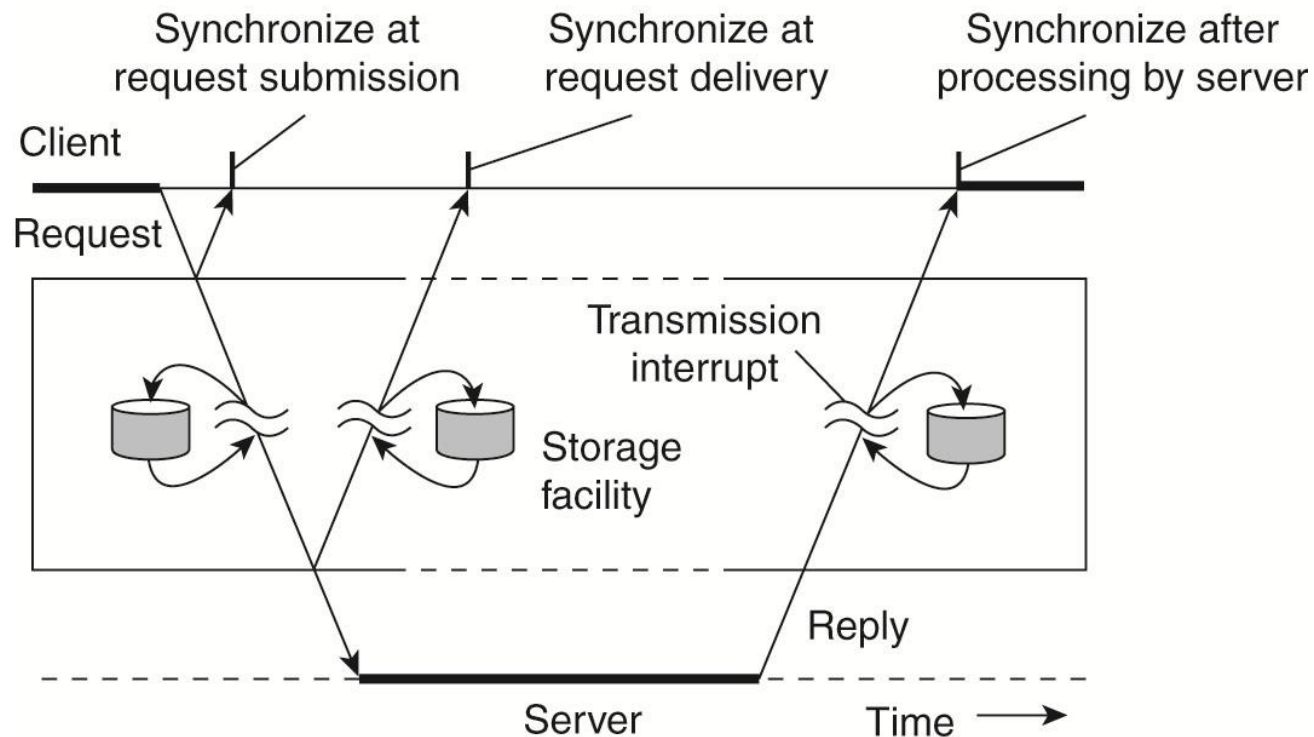
Layered Protocols

- A reference model for distributed systems, adapted from the ISO/OSI model



Middleware in the Communication Path

- Middleware as an intermediate (distributed) service in application-level communication
- Persistent communication in an e-mail system

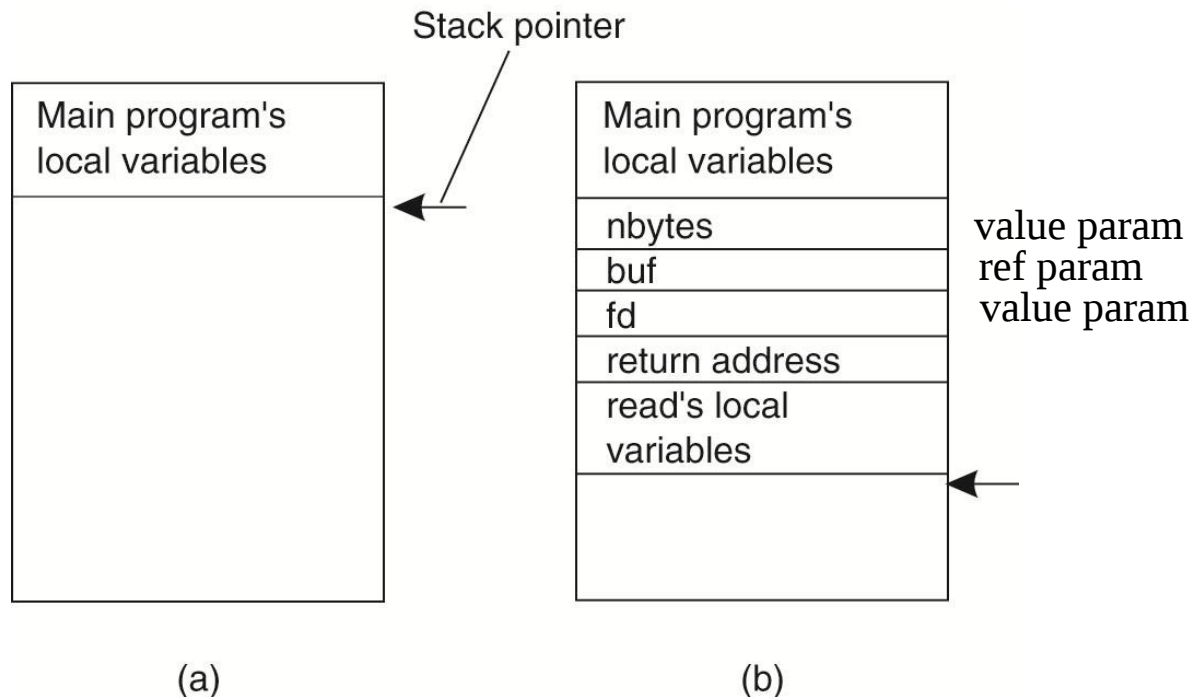


Conventional Local Procedure Call

- Parameter passing in a local procedure call
read(fd, buf, nbytes)

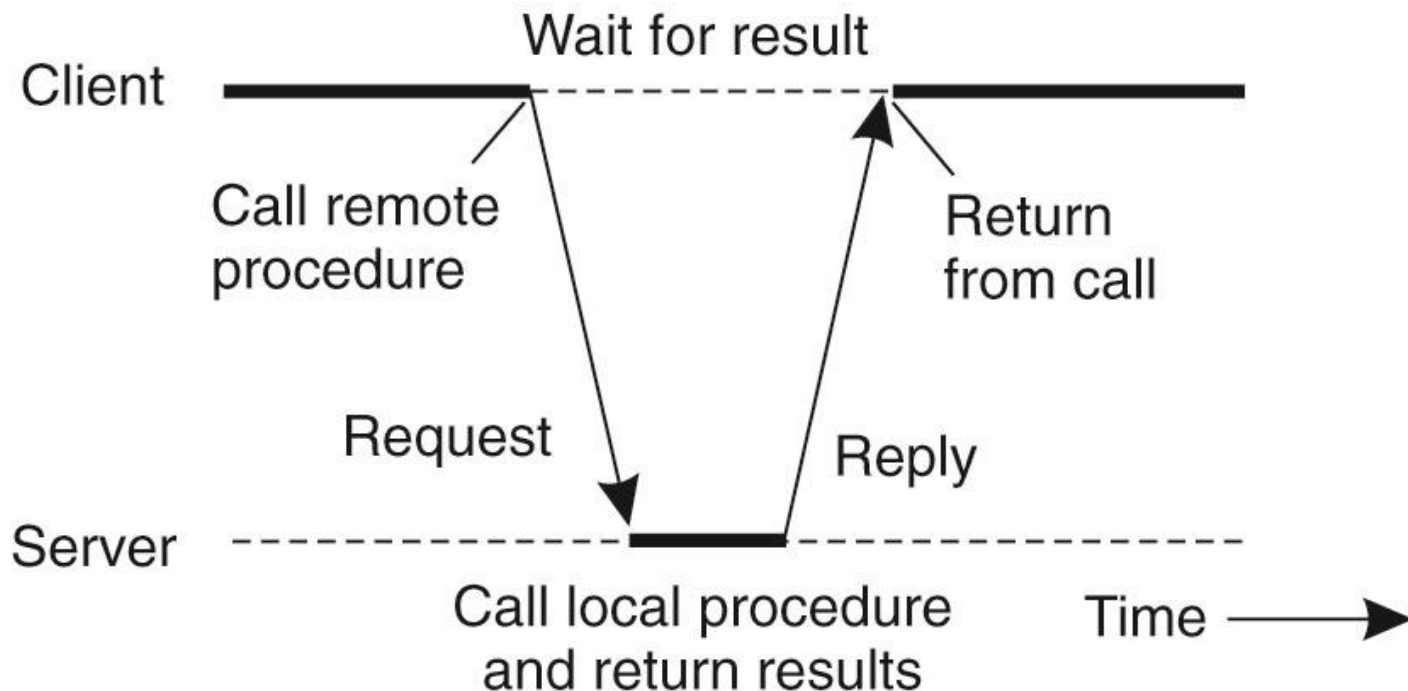
(a) The stack before the call to read

(b) The stack while the called procedure is active



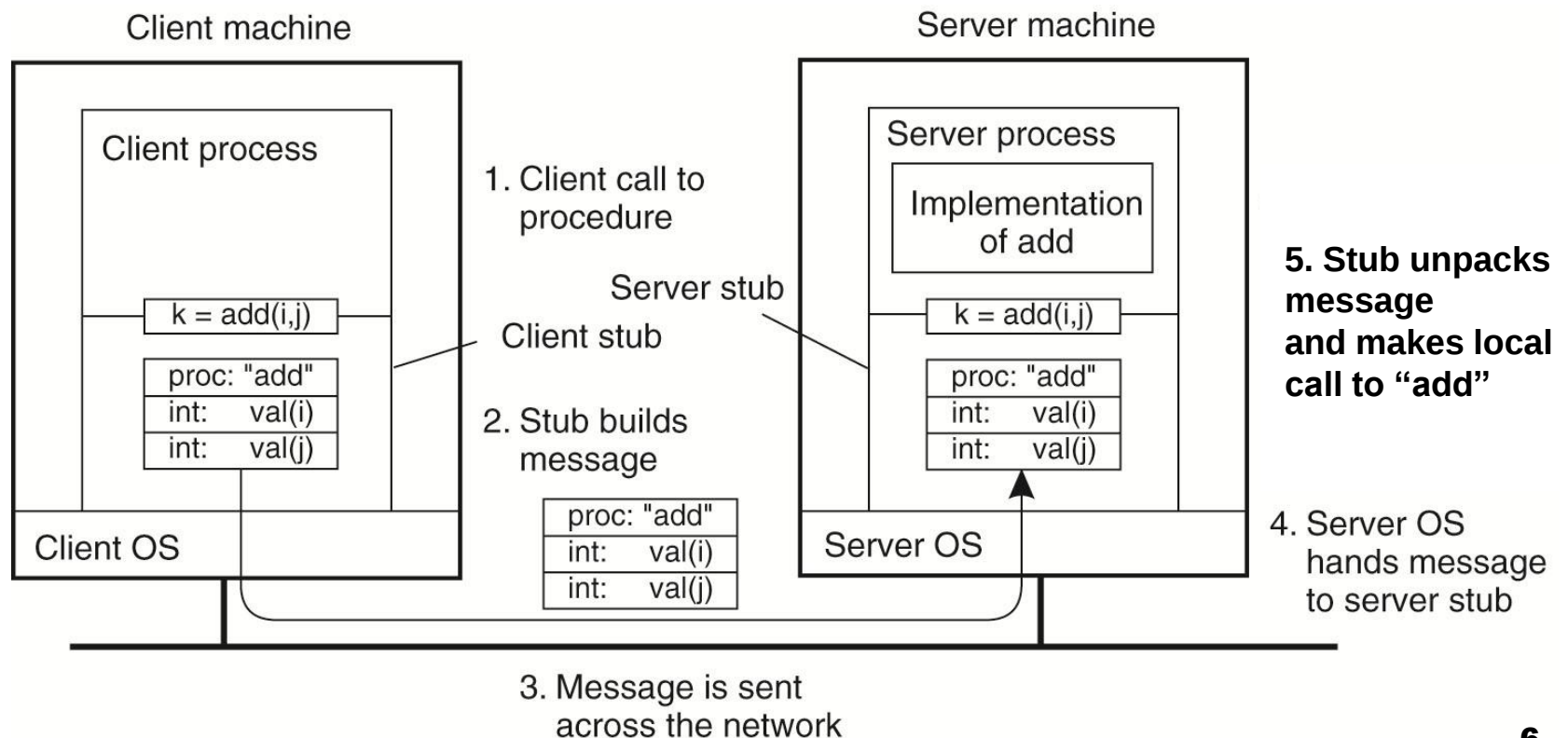
Remote Procedure Call

- Principle of **Remote Procedure Call (RPC)** between a client program and a server program



Remote Procedure Call

- The steps involved in doing a remote computation using RPC



Remote Procedure Call

- A remote procedure call involves the following steps:
 1. The client procedure calls the client stub
 2. The client stub builds a message and calls its local OS
 3. The client OS sends the message to the remote server OS
 4. The server OS passes the message to the server stub
 5. The server stub unpacks the parameters and calls the server
 6. The server executes the procedure and returns the result to the server stub
 7. The server stub packs the result in a message and calls its local OS
 8. The server OS sends the message to the client OS
 9. The client OS passes the message to the client stub
 10. The client stub unpacks the result and returns the result to the client

Passing Value Parameters

(a) The original message on the Pentium (little endian - most significant digit in low address byte)

(b) The message after receipt on the SPARC

(c) The message after being inverted on the SPARC (big endian - most significant digit in high address byte)

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

(a)

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

(b)

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

(c)

- The numbers in the small 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] )  
{  
    ....  
}
```

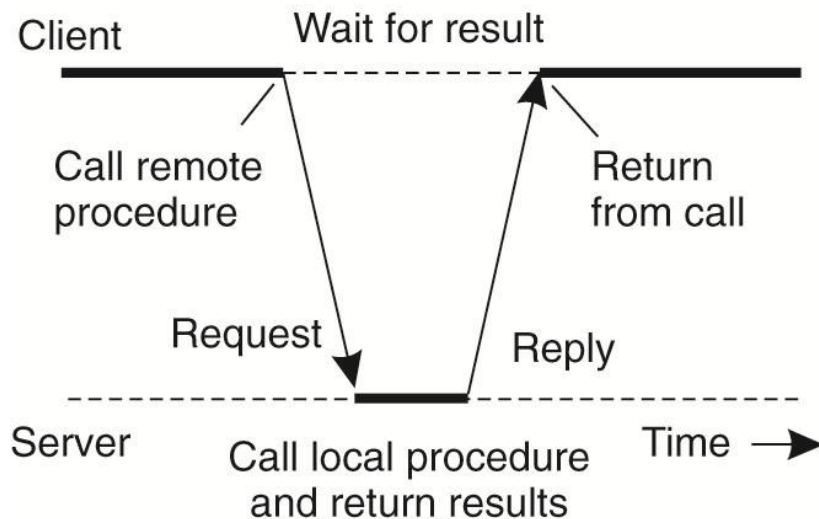
(a)

foobar's local variables	
	x
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

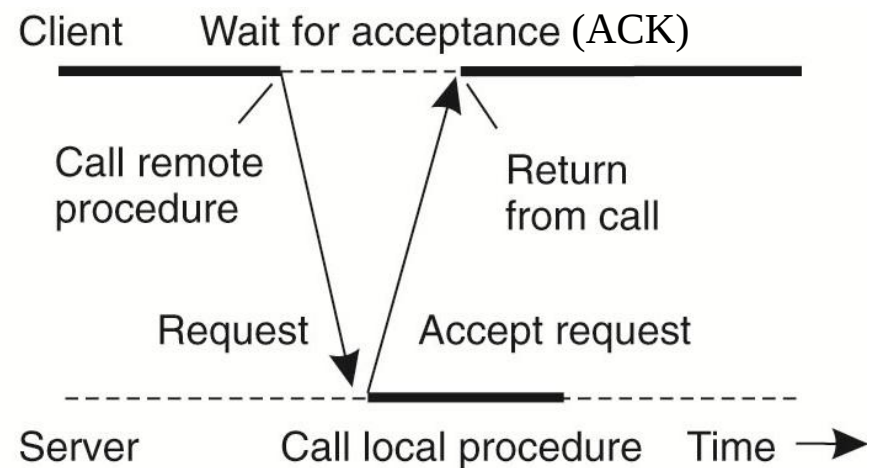
(b)

Asynchronous RPC

- (a) **Traditional Synchronous RPC:** Client calls remote procedure and waits (blocks) until it receives the result from the server
- (b) **Asynchronous RPC:** Client calls remote procedure and waits until it receives an ACK of its request from the server (see next slide)



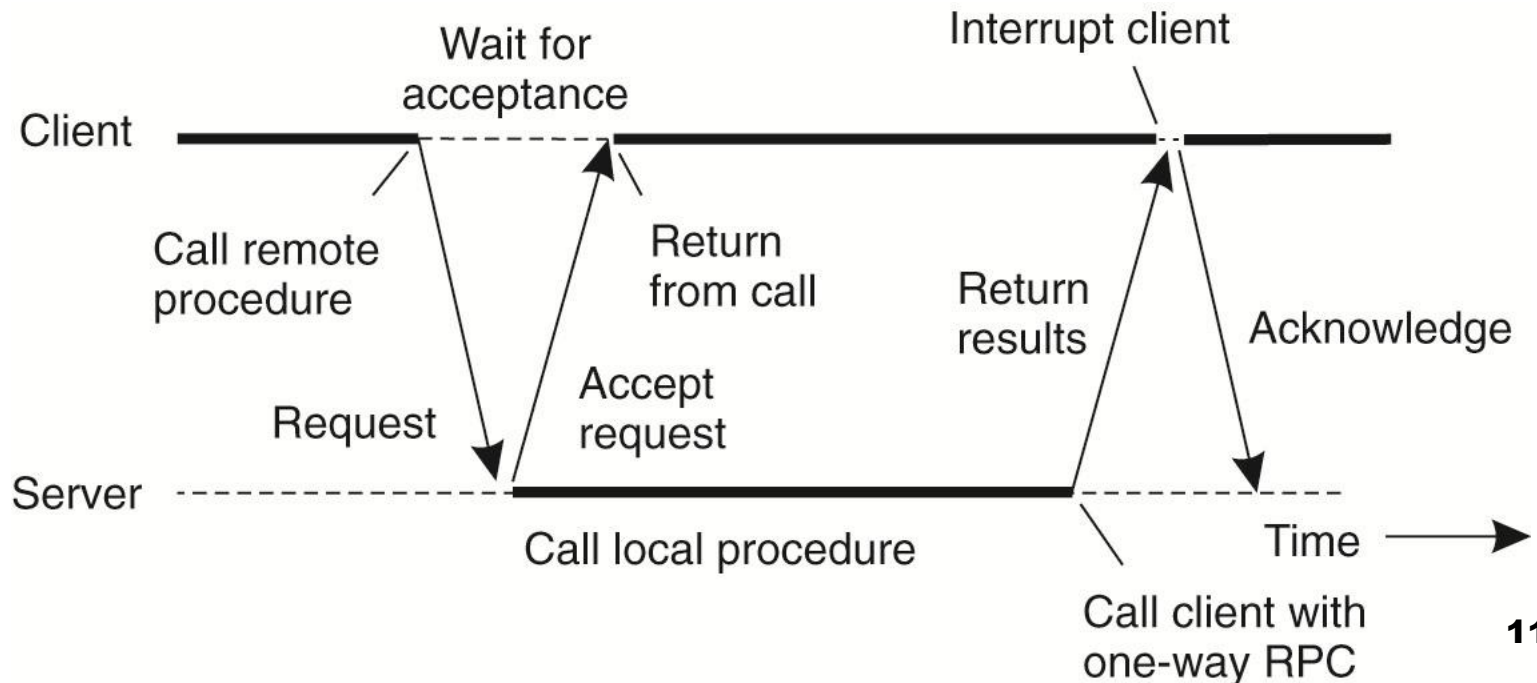
(a)



(b)

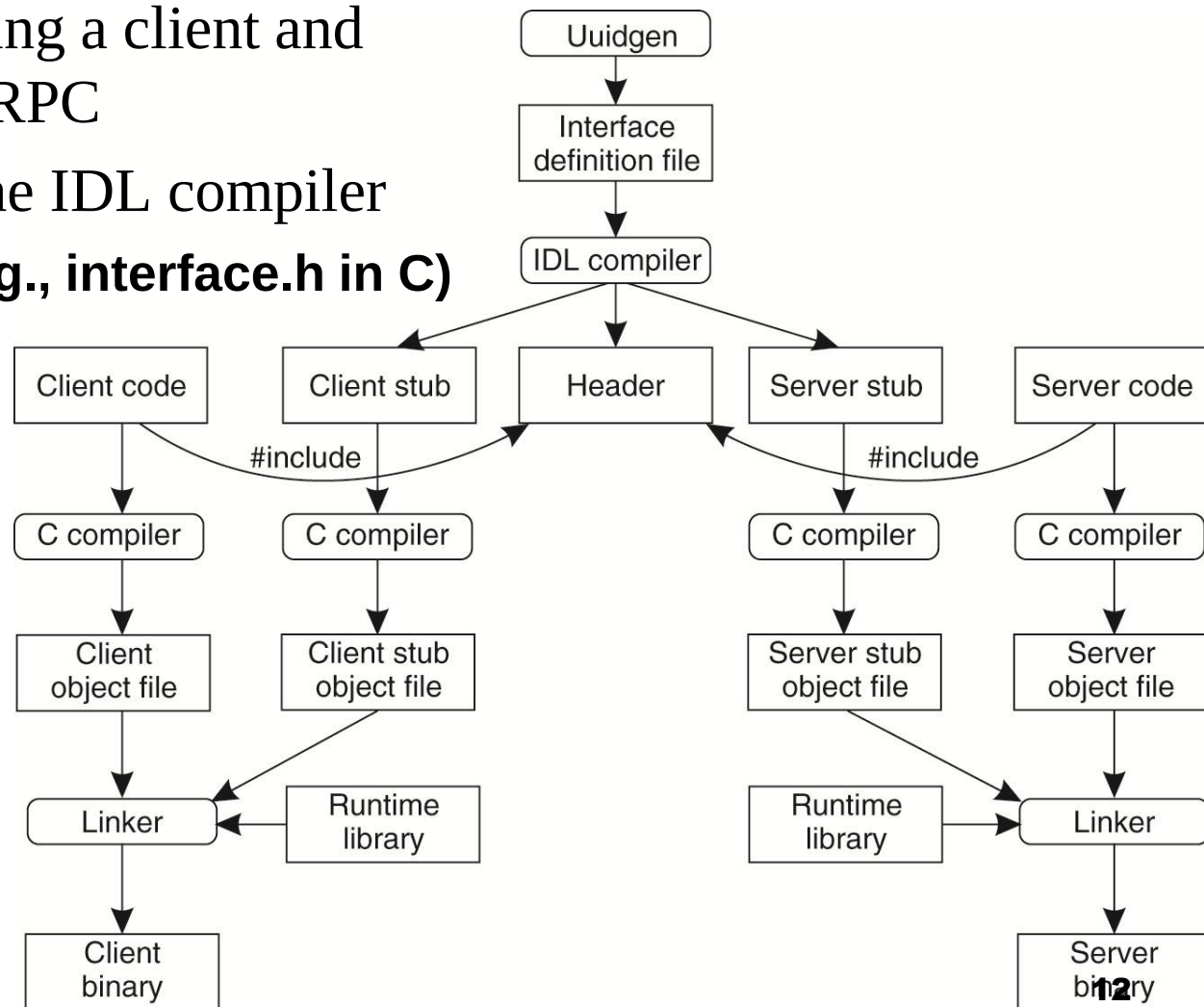
Asynchronous RPC

- Using two asynchronous (one-way) messages
 - Client calls remote procedure and waits until it receives an ACK of its request from the server
 - Server performs the computation and returns the results to the client and waits until it receives an ACK for its response from the client



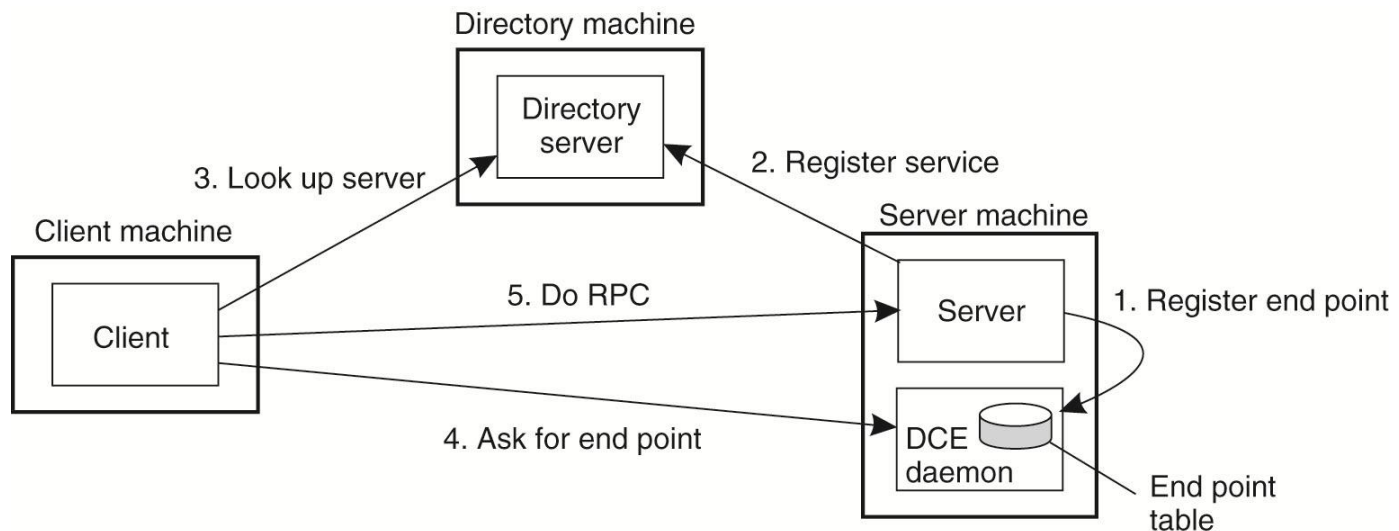
Writing a Client and a Server

- The steps in writing a client and a server in DCE RPC
- Files output by the IDL compiler
 - Header file (e.g., interface.h in C)
 - Client stub
 - Server stub



Binding a Client to a Server in DCE

- Registration of a server is done in two steps
 1. Server registers its endpoint with the DCE daemon
 2. Server registers its service with the Directory server
- Location of the server is done in two steps
 3. Client locates the server machine using the Directory server
 4. Client locates the server on that machine using the DCE daemon





Message Oriented Communication

- Synchronous vs. Asynchronous Communication
- Message Queueing Systems
- Message Brokers
- Example: IBM's MQSeries

Synchronous Communication

- Client-server computing is based on a **synchronous** communication model
 - Client and server must be active at the time of the communication
 - Server waits passively for incoming requests, and processes them when they arrive
 - Client issues request and blocks until it receives a reply from the server
- Drawbacks of synchronous communication
 - Client cannot do any other work while waiting for the server's reply
 - Failures in the server must be dealt with immediately (the client is waiting)
 - In many cases, the model is not appropriate (e.g., e-mail, news)

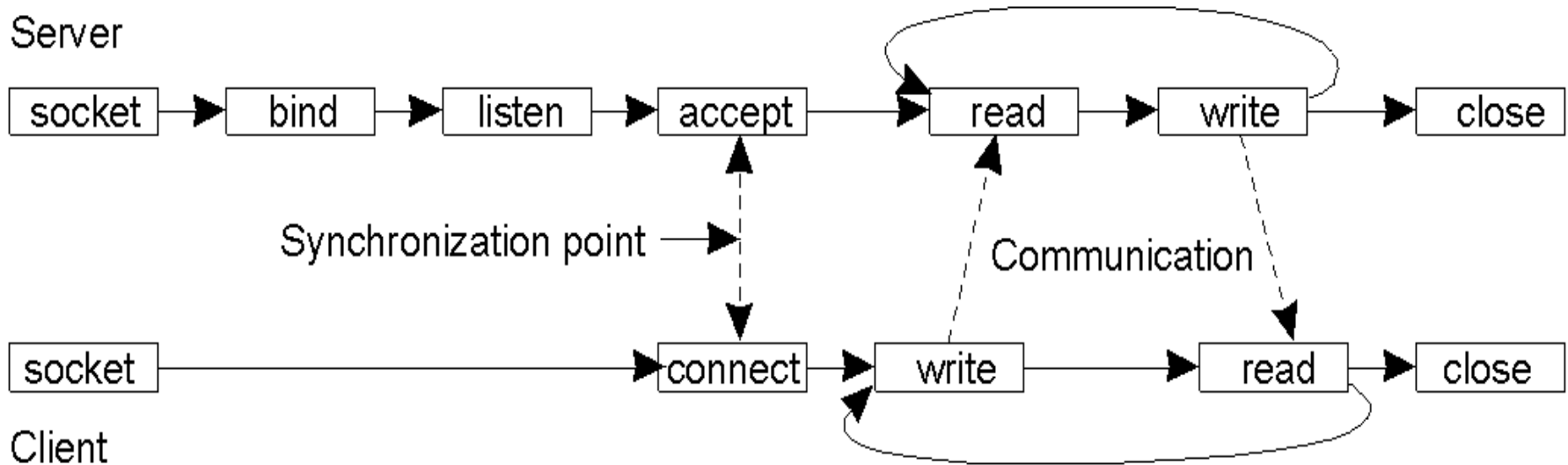
Berkeley Sockets

- Socket primitives for the Transmission Control Protocol (TCP)

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 (Write)	Send some data over the connection
Receive (Read)	Receive some data over the connection
Close	Release the connection

Berkeley Sockets

- Connection-oriented communication pattern using sockets



Message Passing Interface

- Primitives for the Message Passing Interface (MPI), used for parallel applications in high-performance clusters

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_issend	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

Asynchronous Communication

- **Message Oriented Middleware (MOM)** provides high-level **asynchronous** communication
 - Processes send messages to each other
 - Messages are queued
 - Sender does not need to wait for an immediate reply, but can do other processing
- Message oriented middleware facilitates fault tolerance

Message Queueing Model

- Four combinations for loosely-coupled communication using message queues

Sender running



Receiver running

(a)

Sender running



Receiver passive

(b)

Sender passive



Receiver running

(c)

Sender passive



Receiver passive

(d)

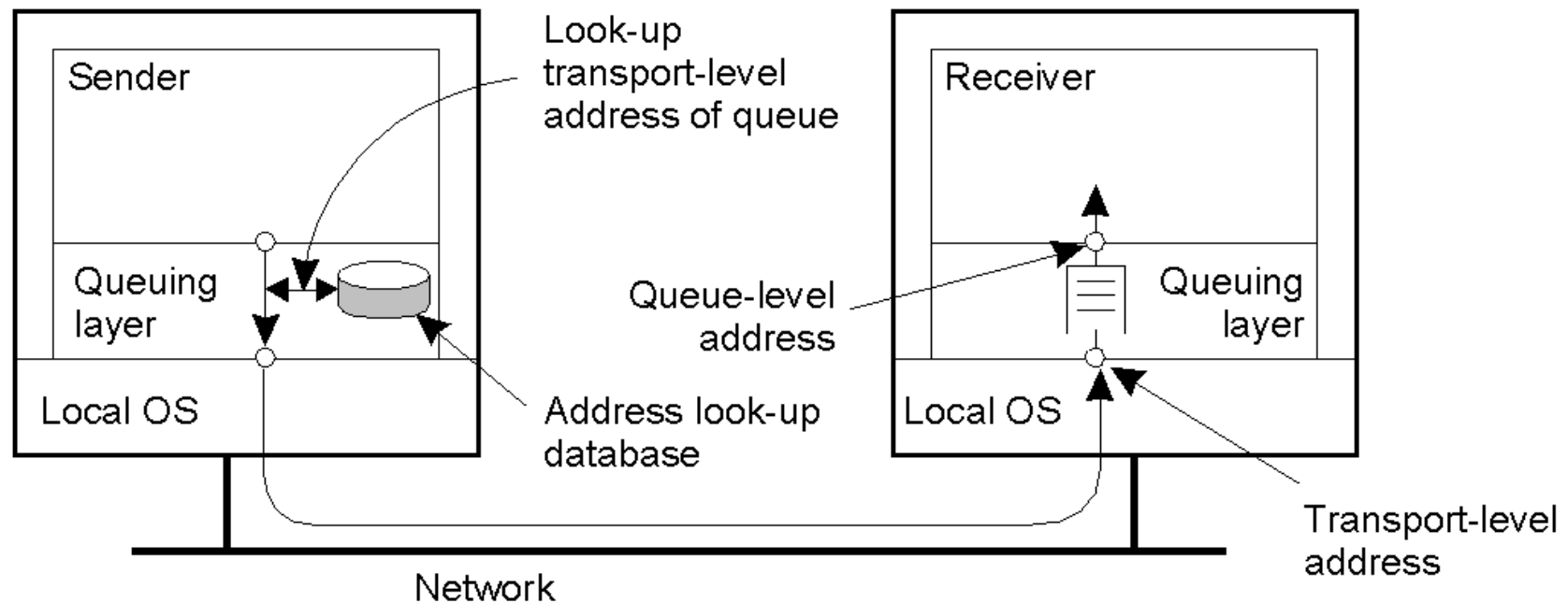
Message Queueing Model

- Basic primitives that provide access to a queue in a message queueing system

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

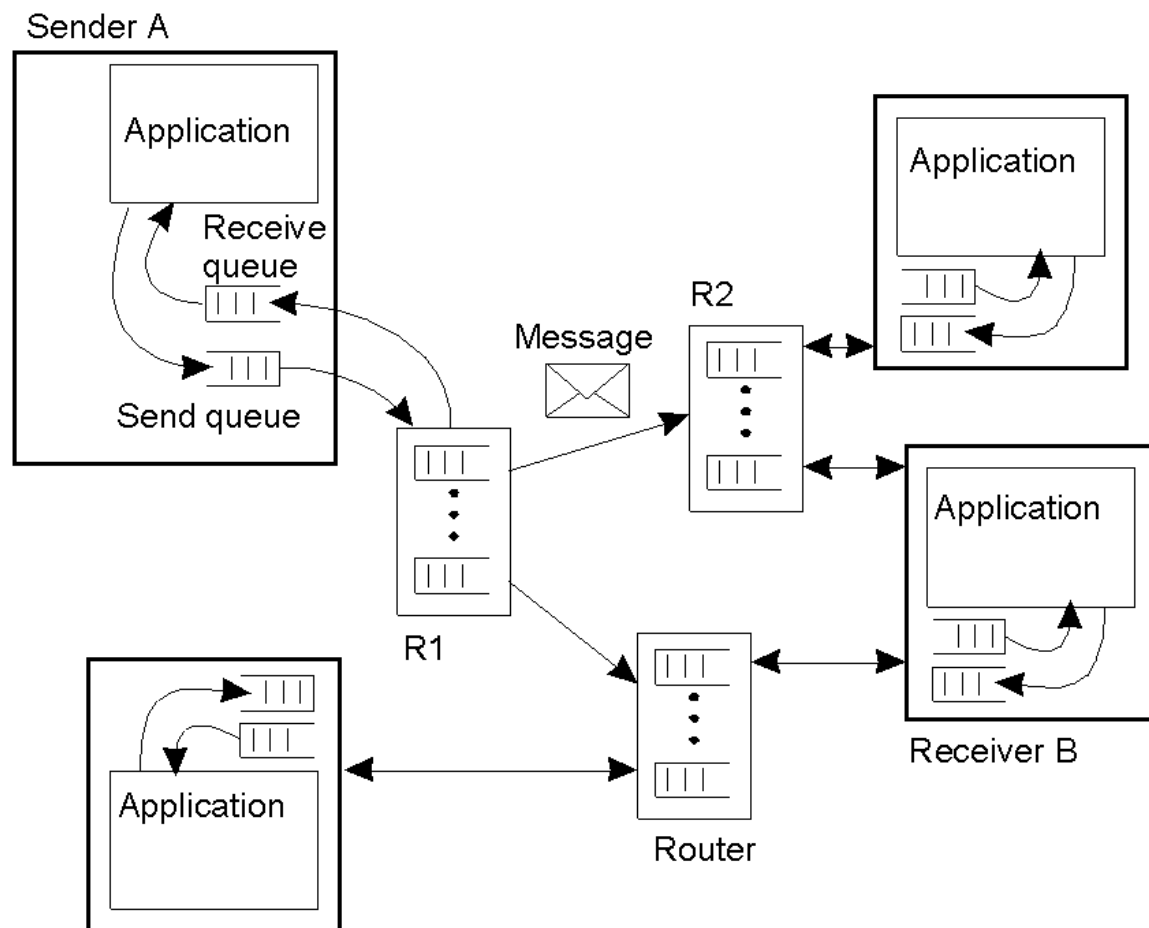
Architecture of a Message Queueing System

- The relationship between Transport-level addressing and queue-level addressing in the middleware



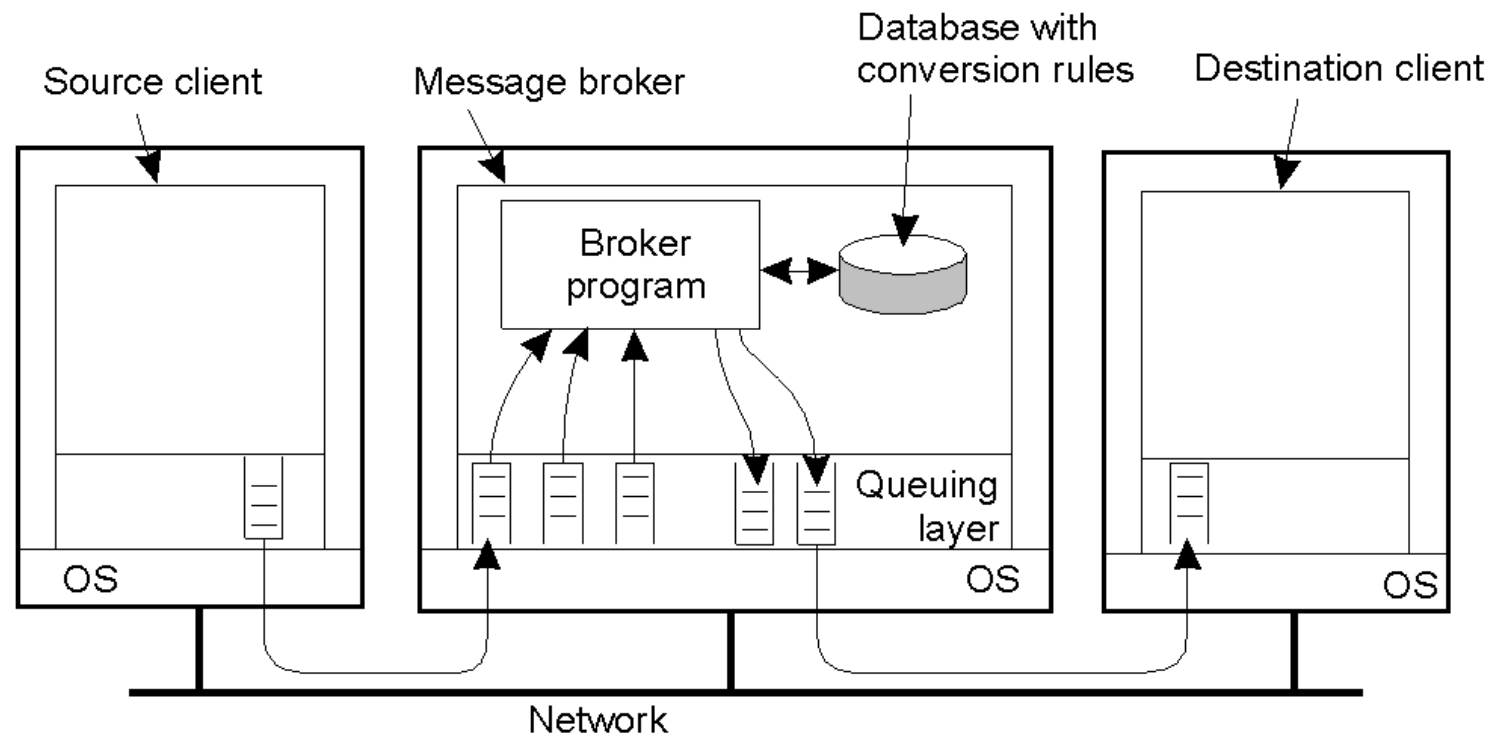
Architecture of a Message Queueing System

- A message queueing system with routers



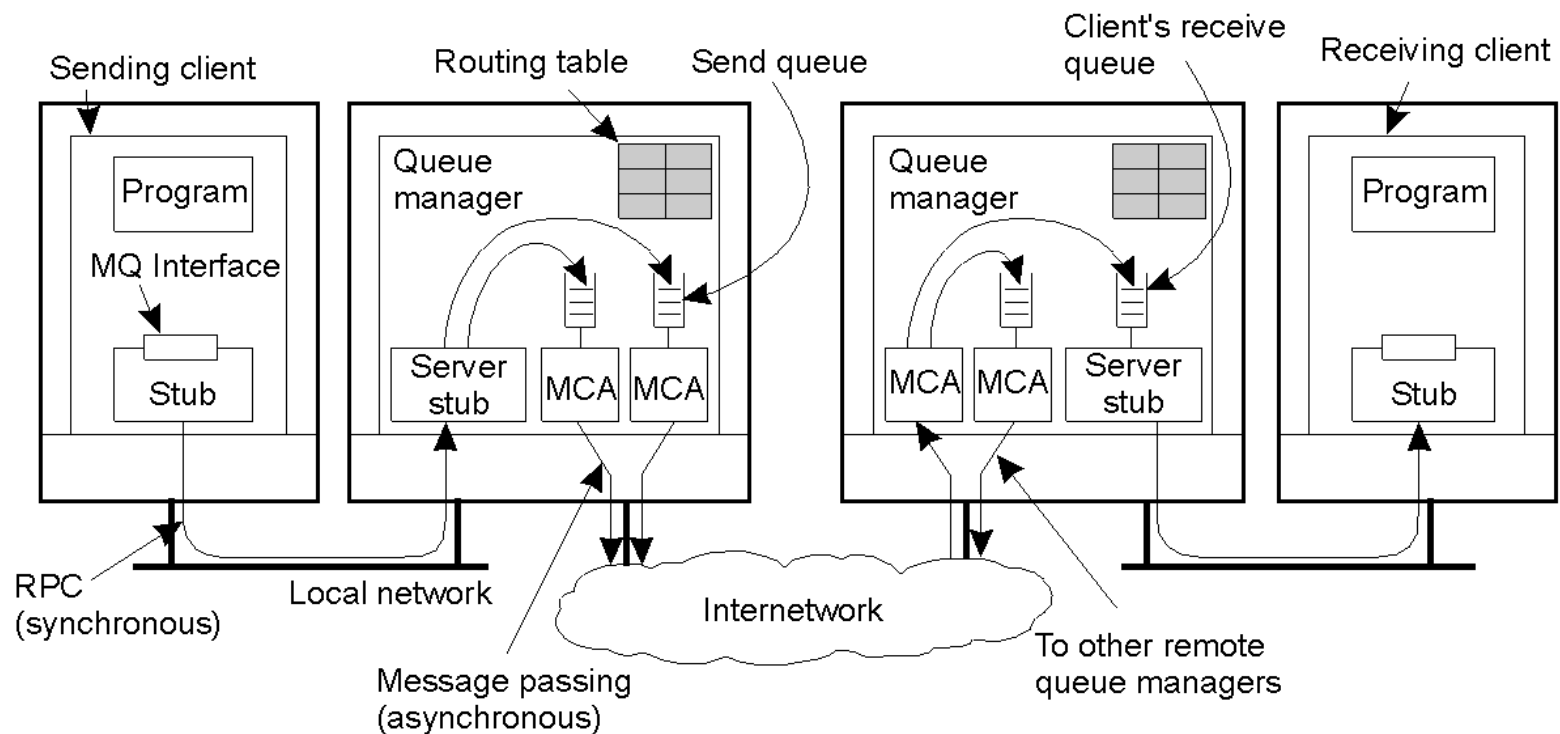
Message Brokers

- A message broker (middleman) in a message queueing system, e.g., for reformatting a message or transforming a message from one type to another



Example: IBM's MQSeries

■ IBM's MQSeries message queueing system



■ Queue Manager

- Removes messages from send queue and forwards to other Queue Managers
- Receives incoming messages and stores them in receive queue

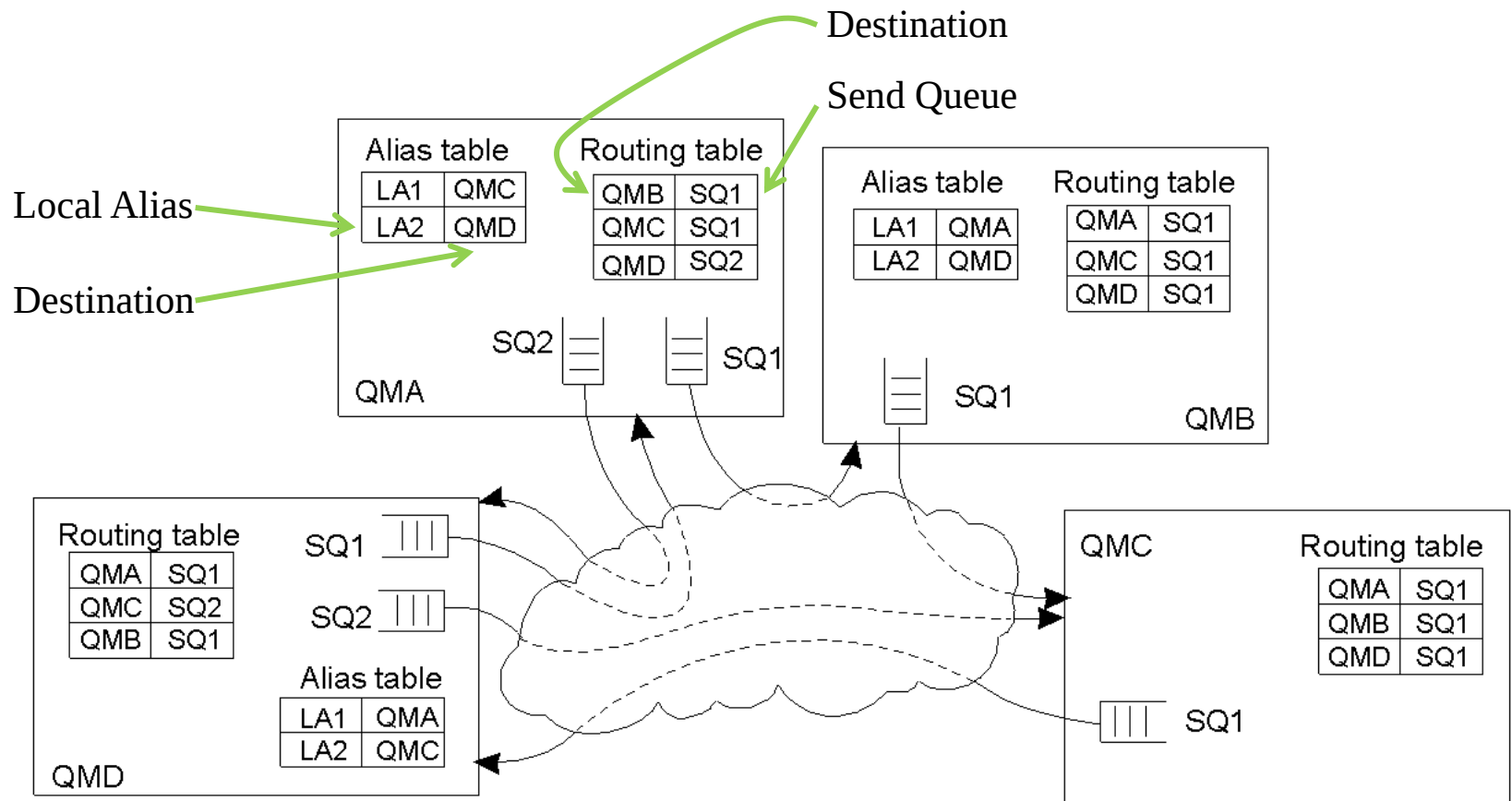
MQSeries: Message Channels

- Message Channel – Unidirectional, reliable connection between sending and receiving Message Queue Managers, through which queued messages are transported
- Message Channel Agent (MCA) – Manages one end of a message channel
 - **Attributes associated with Message Channel Agents**

Attribute	Description
Transport type	Determines the Transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order 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

MQSeries: Message Transfer

- An MQSeries queueing network using Alias and Routing tables



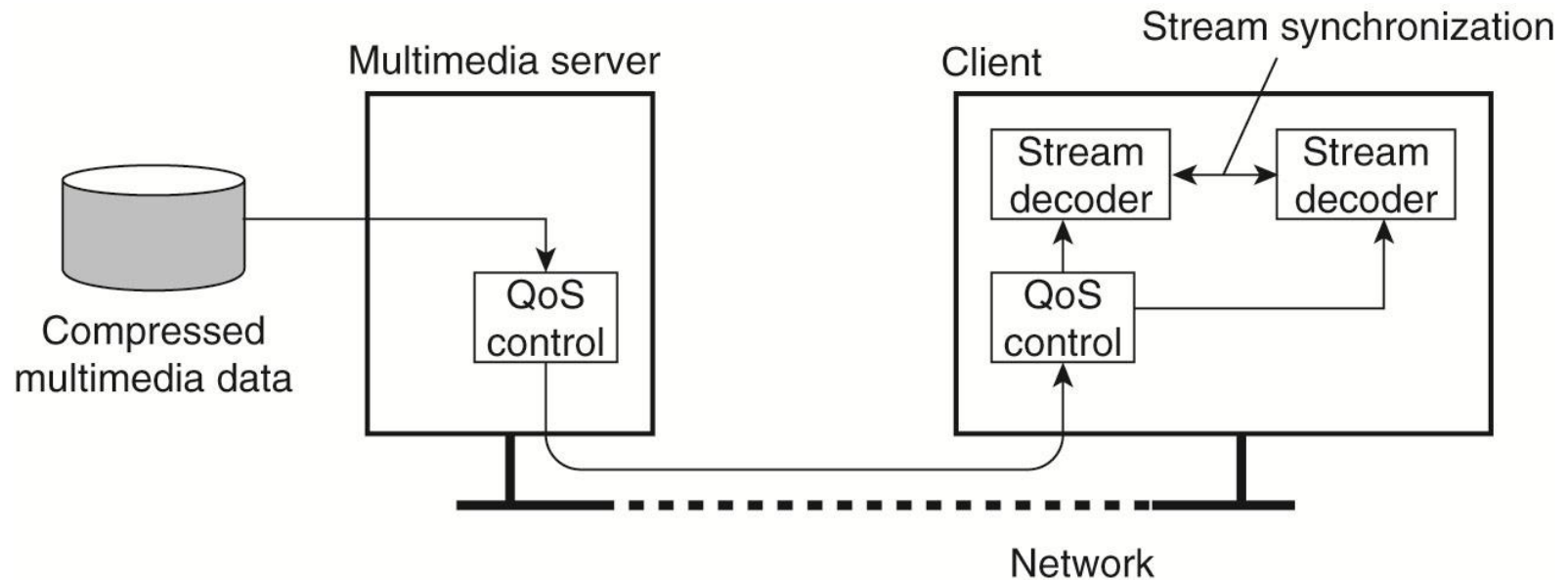
MQSeries: Message Queue Interface

- Primitives for the Message Queue Interface (MQI)

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

Multimedia Data Streams

- An architecture for streaming stored (rather than live) multimedia data over a network
 - **Storing provides more opportunity for fine tuning**

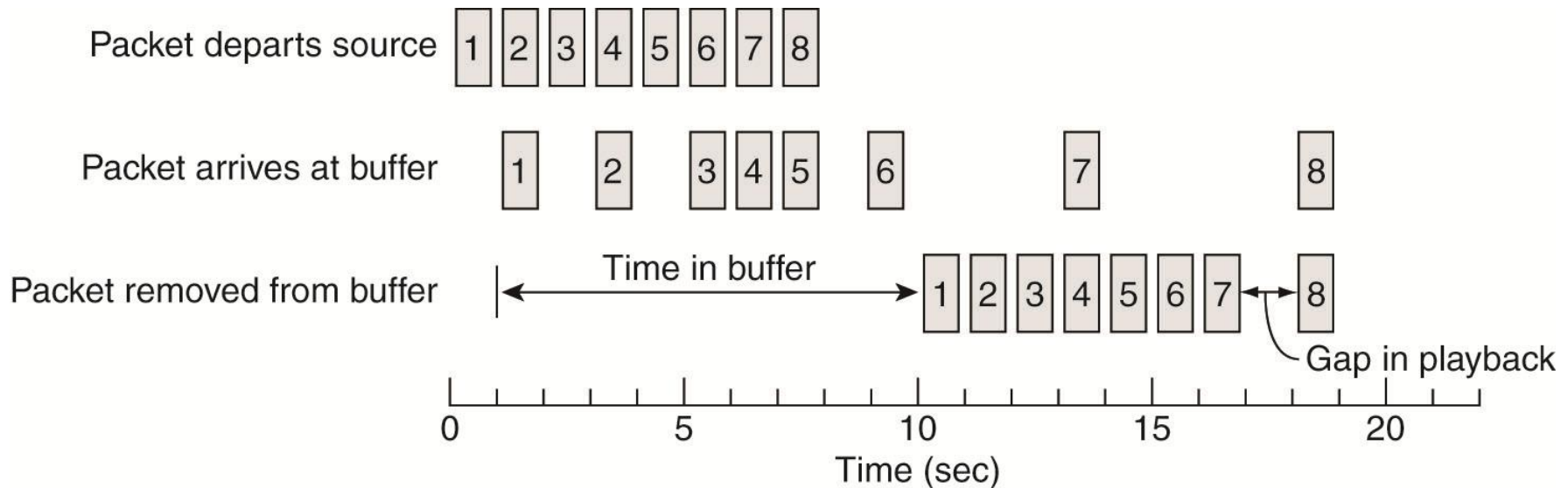


Quality of Service

- Quality of Service (QoS) metrics
 - Required bit rate at which data should be transported
 - Maximum delay (latency) until a session has been set up
 - Maximum end-to-end delay (latency)
 - Maximum delay variance (jitter)
 - Maximum round-trip delay (latency)

Enforcing QoS

- Using a buffer to store packets before delivering them to the destination in order to reduce jitter



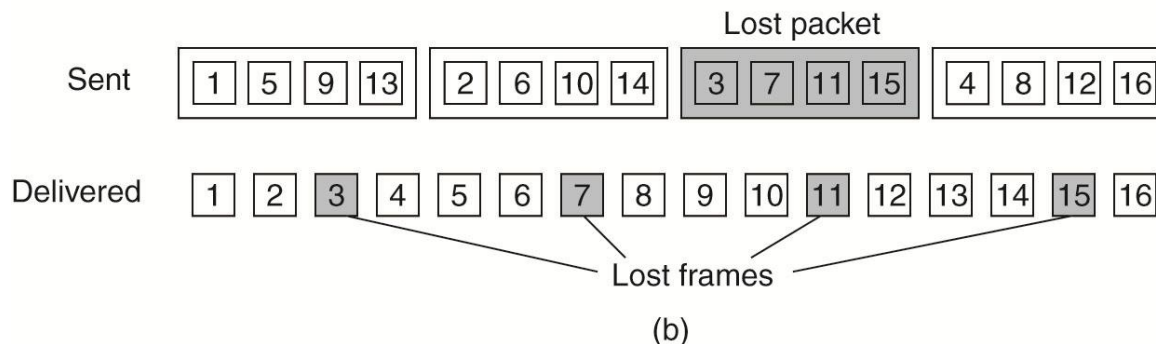
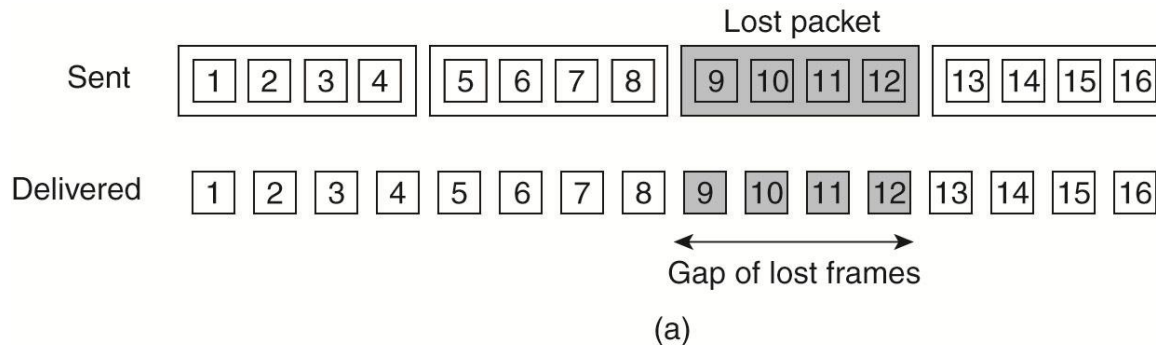
- Gap in playback results because
 - Size of receiver's buffer corresponds to 9 sec
 - Packet 8 took 11 sec to reach the destination

Enforcing QoS

■ Effect of packet loss

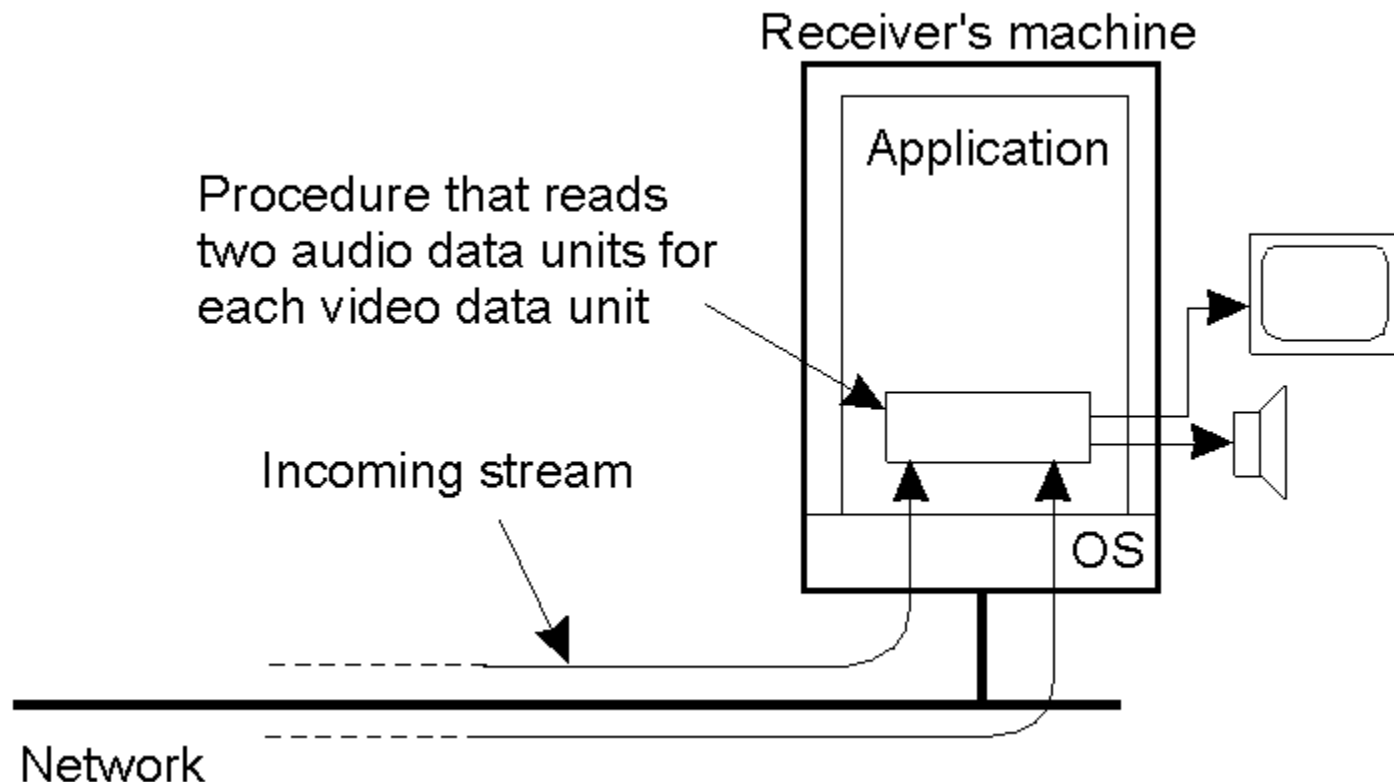
(a) Non-interleaved transmission of frames

(b) Interleaved transmission distributes gap over time, but requires a larger buffer at receiver



Synchronization Mechanisms

- Explicit synchronization at the level of data units



Synchronization Mechanisms

- Synchronization supported by high-level interfaces

