CSE 5306
Distributed Systems

Communication

Jia Rao

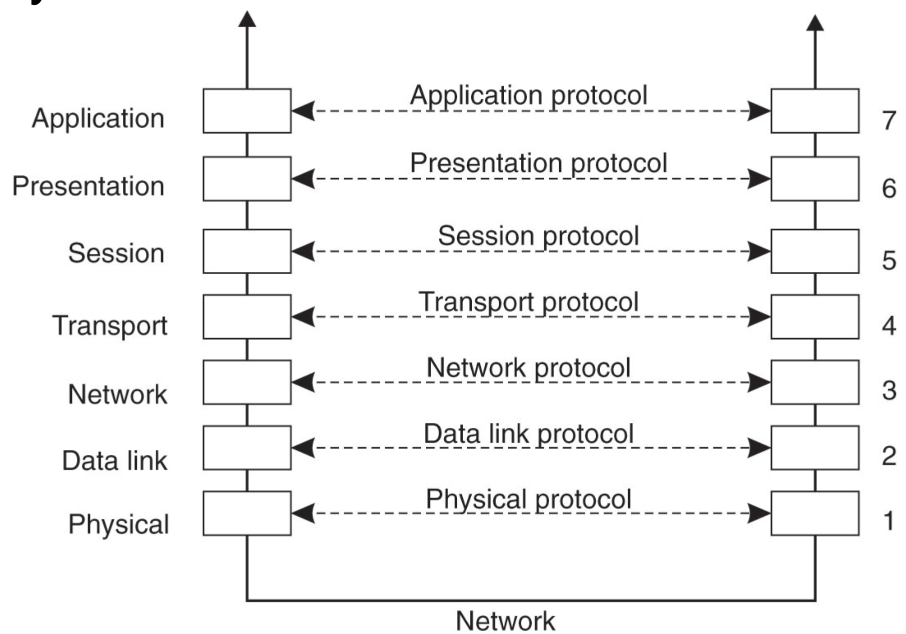
http://ranger.uta.edu/~jrao/

Interprocess Communication (IPC)

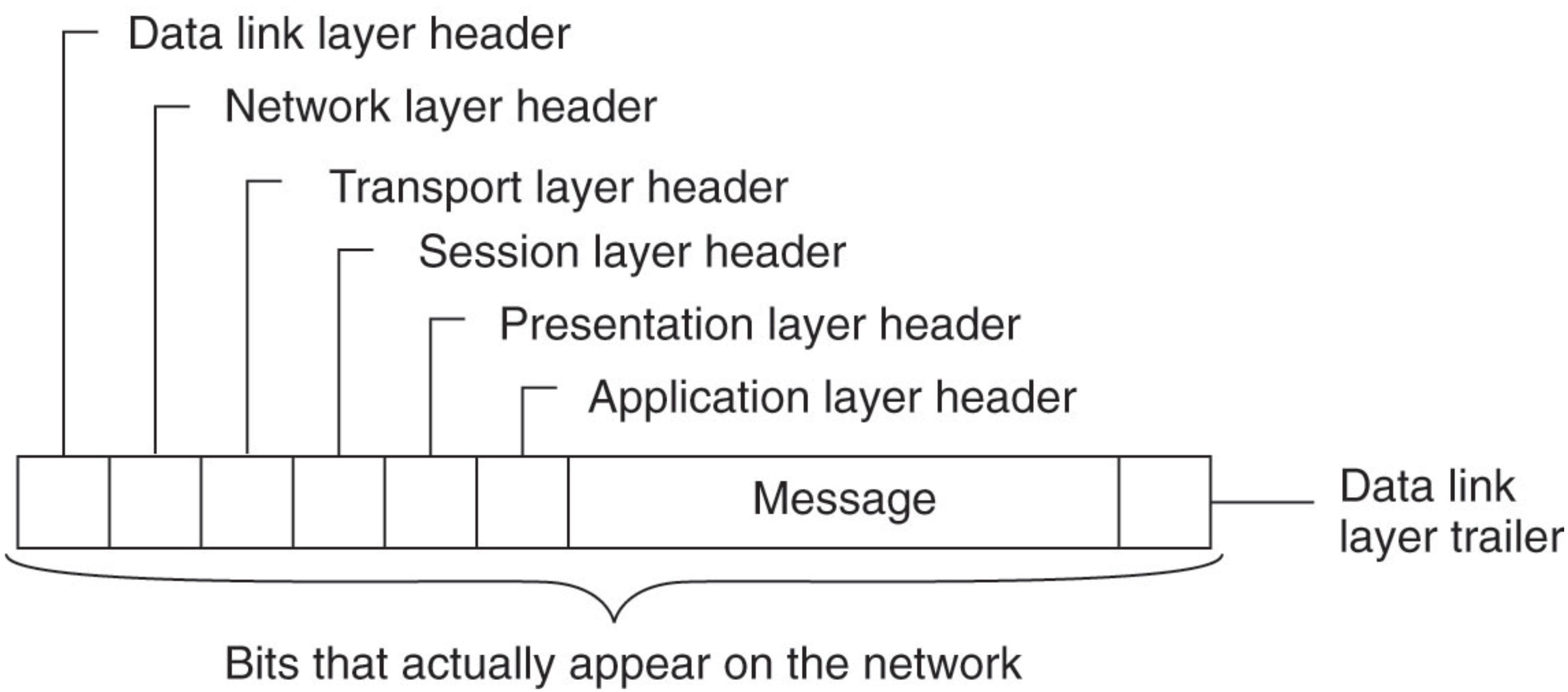
- A major concern in all distributed systems
  - How do we exchange information between processes located on different computers? No shared memory or clock
  - IPC is the heart of all distributed systems
- IPC is often built on low-level message passing offered by the underlying network, however
  - Low-level message passing is unreliable and slow
- Topics
  - Fundamentals of communication in distributed systems
  - Three widely used models: RPC, MOM, Streaming
  - Multicasting

Fundamentals

- Communication needs many levels of agreements
  - ✓ Representations of 0 and 1, end of message
  - ✓ Error detection
- ISO OSI 7-layer model



Message Format



Low-level Protocols

- Physical Layer sends bits
  - ✓ Bit may be corrupted
- Data link layer handles bit errors by
  - ✓ Grouping bits into frames and adding a checksum
- Network layer handles the delivery of a message to a destination (routing)
  - ✓ Each node has a unique ID, e.g., IP address

Transport Layer Protocol

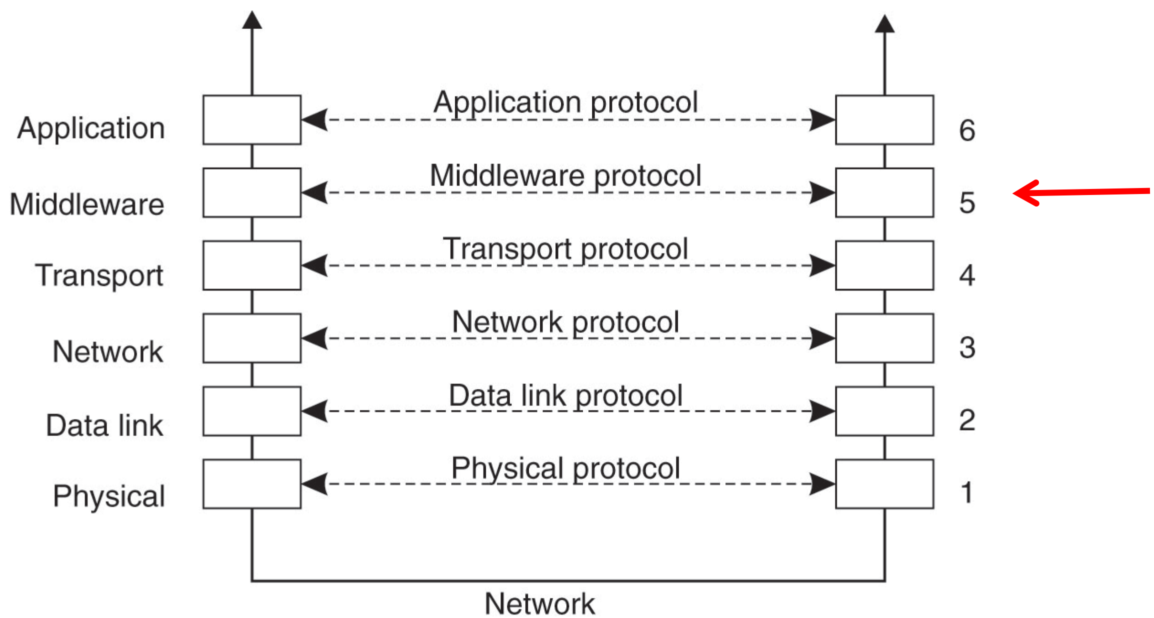
- Provides services needed for building network applications
  - ✓ Turn the underlying network into something that a developer can easily use
- Example transport layer protocols
  - ✓ TCP: connection-oriented, reliable communication
  - ✓ UDP: connectionless, unreliable, application handles error
  - ✓ RTP: real-time, supports real-time data transfer

Higher-level Protocols

- Session layer is an enhanced version of transport layer
  - ✓ Provides dialog control, e.g., keeps track of who is talking and provide synchronization
- Presentation layer is mainly concerned with the meaning of bits
- Application layer contains all applications and protocols that do not fit into one of the underlying layer
  - ✓ FTP, HTTP, NFS

Middleware Protocols

- In some cases, there are general-purpose protocols that are useful to many applications, but cannot be classified as transport layer protocols
  - ✓ e.g., entity authentication protocol, distributed commit and locking protocol



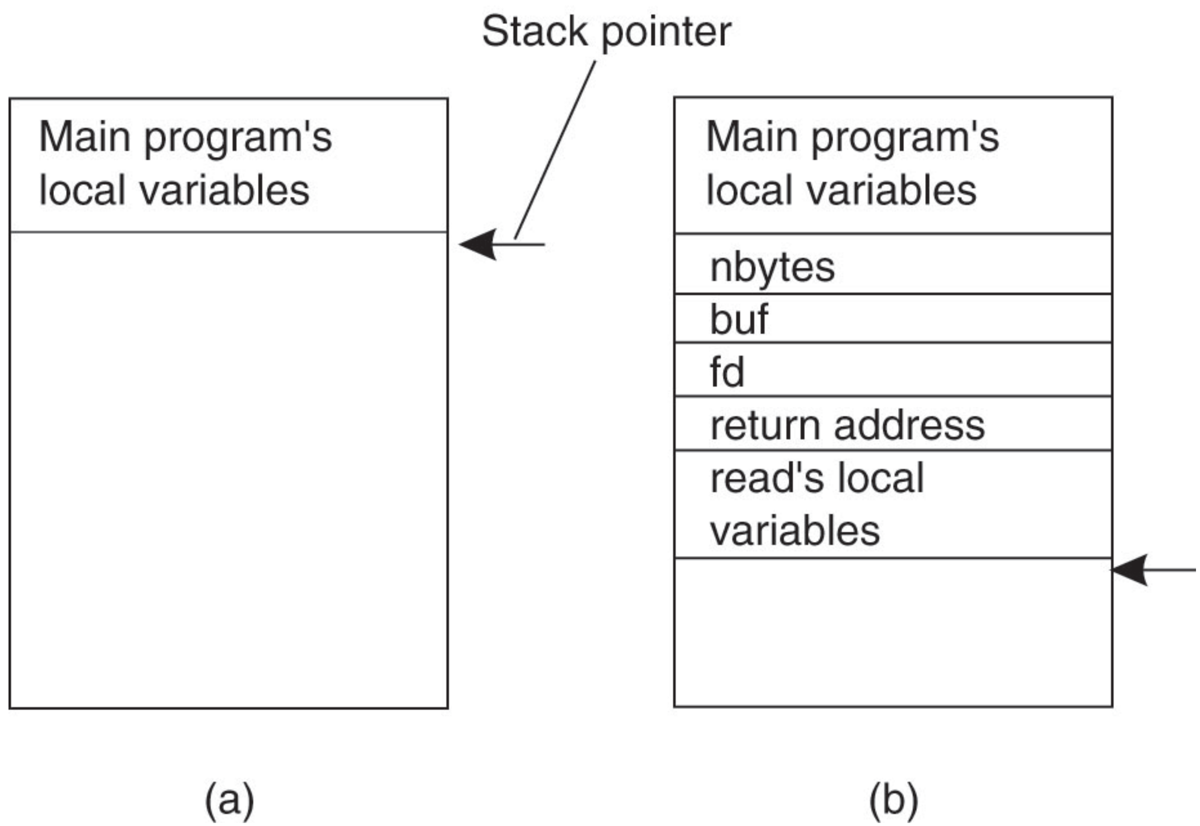
Types of Communications

- Persistent communication
  - ✓ Message that has been submitted for transmission is stored by the communication middleware
- Transient communication
  - ✓ Message is stored by the communication system only if the sending and receiving applications are executing
- Asynchronous communication
  - ✓ Sender continues immediately after submitting a message
- Synchronous communication
  - ✓ Sender will wait until it is certain that the message is delivered
- Discrete communication v.s. streaming communication

Remote Procedure Call (RPC)

- Access transparency v.s. explicit send/receive
- RPC allows programs to call procedures/functions located on remote machines
  - ✓ Parameters passed to the callee and only the results comes back to the caller
- Issues to be addressed
  - ✓ Different address spaces causes complications, e.g. pointers
  - ✓ Different machines may represent numbers, characters, etc., in a different way

Conventional Procedure Call

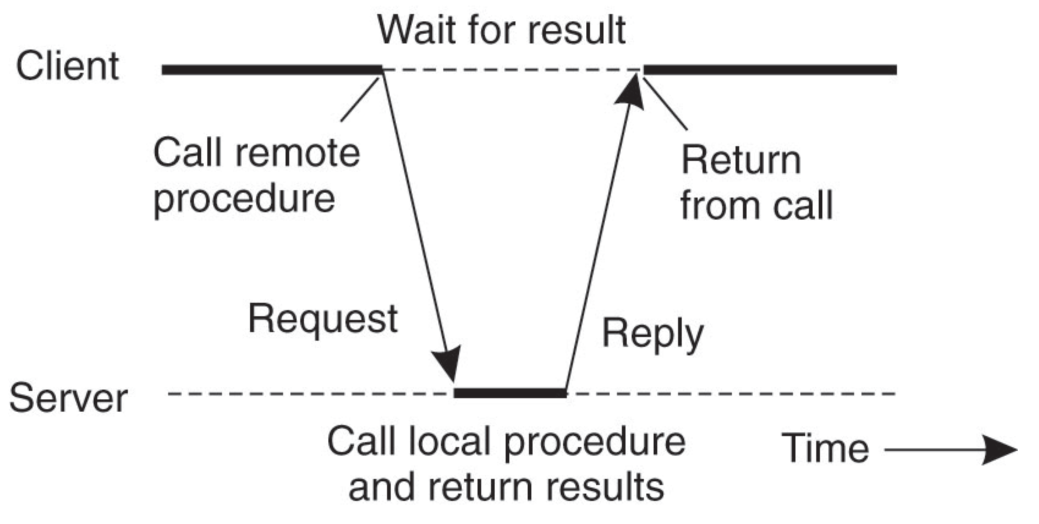


(a) Parameter passing in a local procedure call: the stack before the call to read.

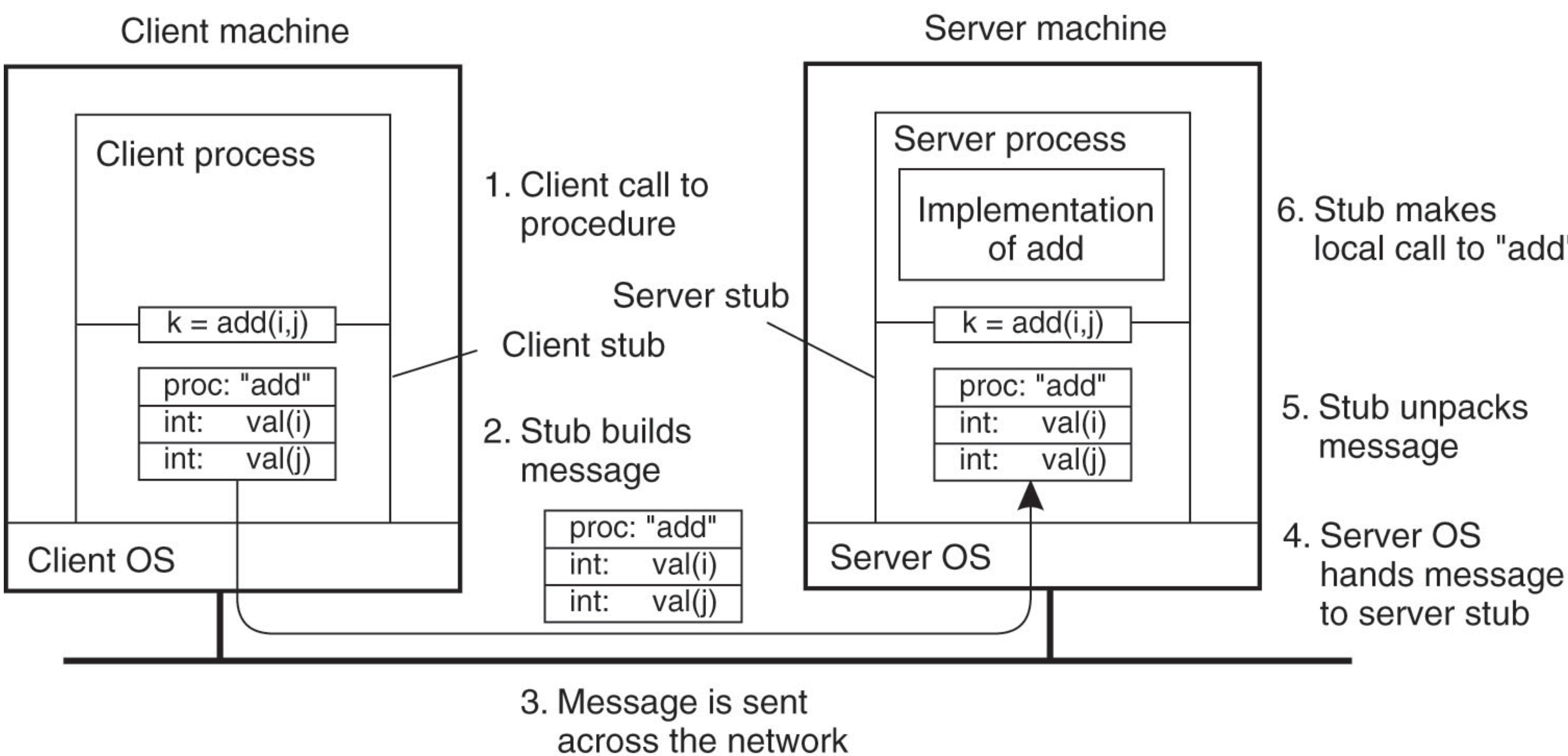
(b) The stack while the called procedure is active.

RPC between Client and Server

- RPC achieves transparency through client and server stubs
  - ✓ Client stub packs parameters into a message to the server, and copies the result from the server to the client
  - ✓ Server stub unpacks parameters, calls the procedure, and packs the result to the client



Passing Value Parameters



Pass Reference Parameters

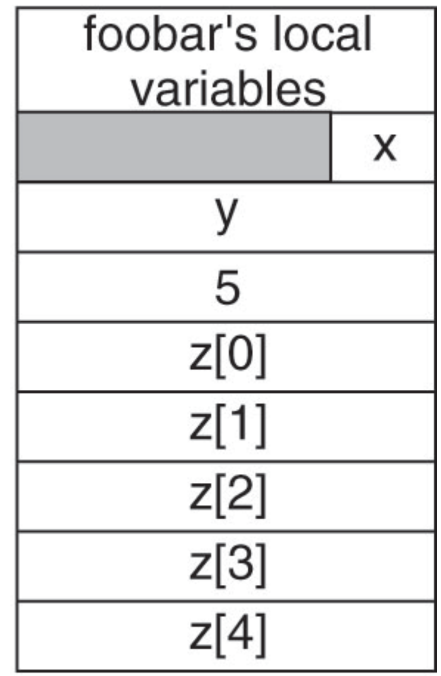
- Forbid pointers and reference parameters
- Call-by-copy/store
  - ✓ Copy the referenced data to the server and copy back the result from the server
- One optimization
  - ✓ If input parameters only, no copy back
  - ✓ If output parameters only, no copy to server
- However, we still cannot handle the general case
  - ✓ e.g., an arbitrary data structure such as a complex graph

Parameter Spec. & Stub Generation

- Agreement has to be made between client and server stubs

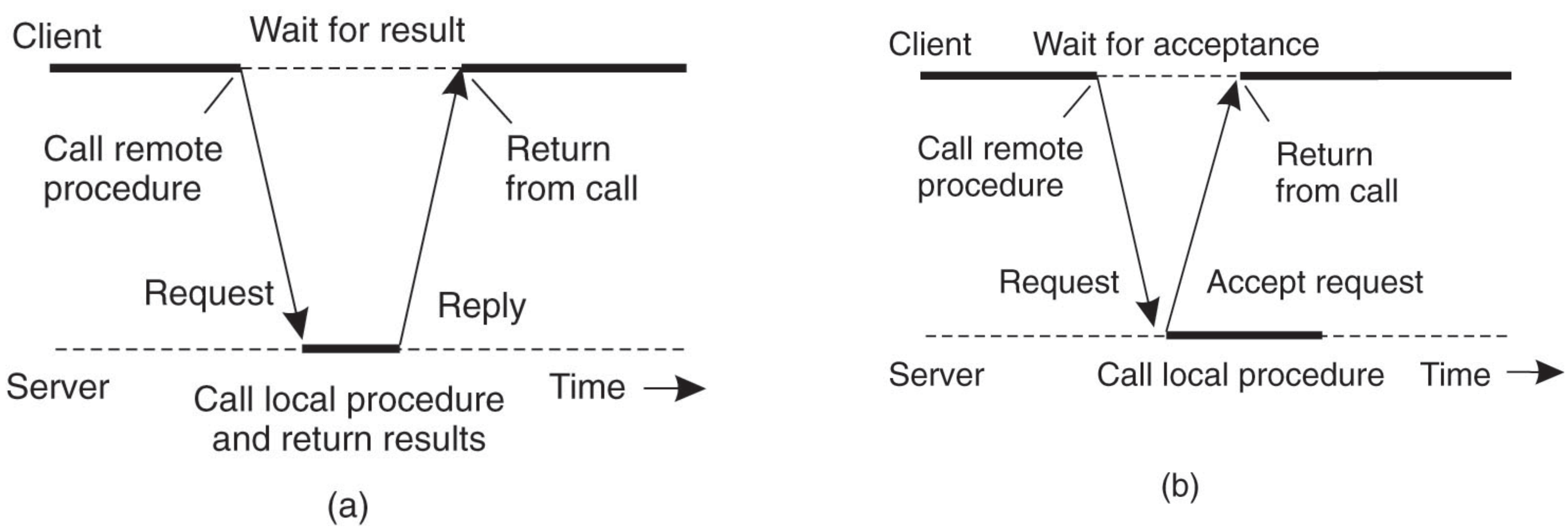
```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

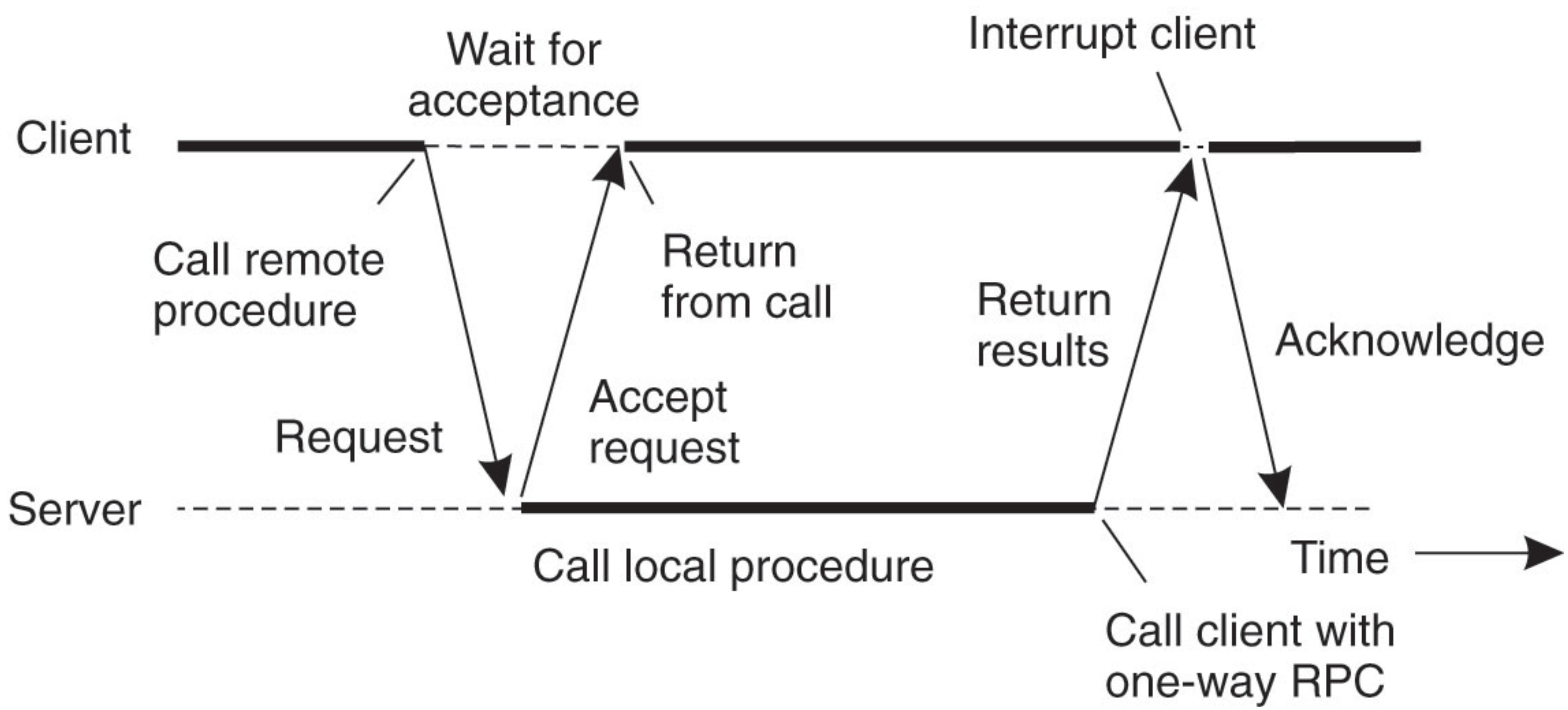
Asynchronous RPC



The interaction between client and server in

(a) a traditional RPC and (b) an asynchronous RPC.

Deferred Synchronous RPC



A client and server interacting through
two asynchronous RPCs.

Message-oriented Communication

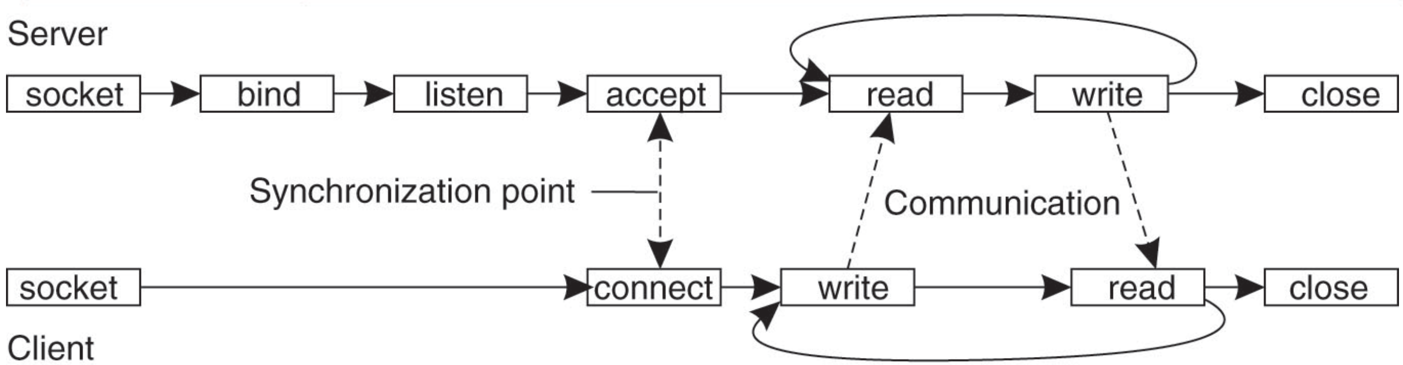
- RPC assumes that the server is active when a request is issued
- A client is often blocked until its request has been processed at the server
- Message-oriented communication is an alternative approach

Message-oriented Transient Comm.

- Transport layer offers a simple message-oriented communication model
  - ✓ Many applications directly build on top of such a model
- A typical example is sockets
  - ✓ A socket is a comm. end-point for I/O at the transport layer
  - ✓ A server binds its IP address together with a port # to a socket

Berkeley Sockets Primitives

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



Message-passing Interface (MPI)

- Sockets only support simple comm. primitives
- The need for a hardware/platform independent standard and a more powerful library for message passing
- Message-passing interface (MPI)
    - ✓ Mostly used for “transient” communication
    - ✓ MPI assumes communication takes place within a known group of processes

Some MPI Primitives

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

Blocking vs. Non-blocking (1/2)

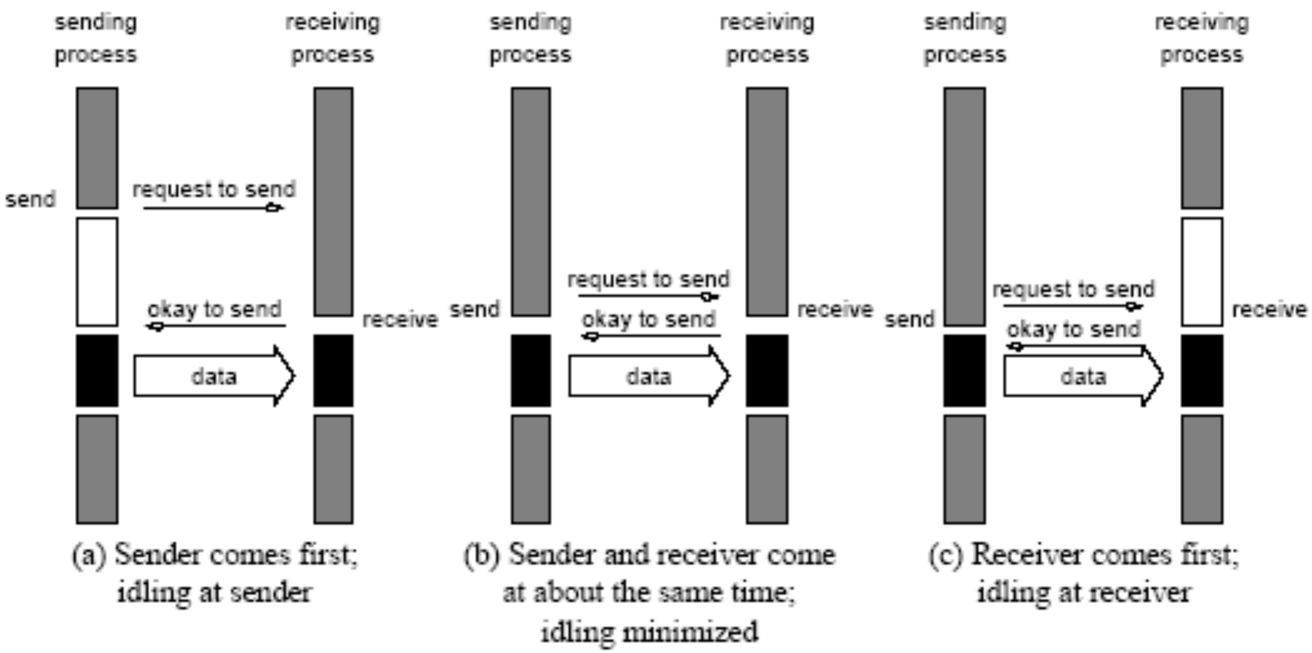
A blocking send routine will only "return" after it is safe to modify the application buffer (your send data) for reuse.

– Safe does not imply that the data was actually received - it may very well be sitting in a system buffer.

– A blocking send can be synchronous which means there is handshaking occurring with the receive process to confirm a safe send.

– A blocking send can be asynchronous if a system buffer is used to hold the data for eventual delivery to the receive.

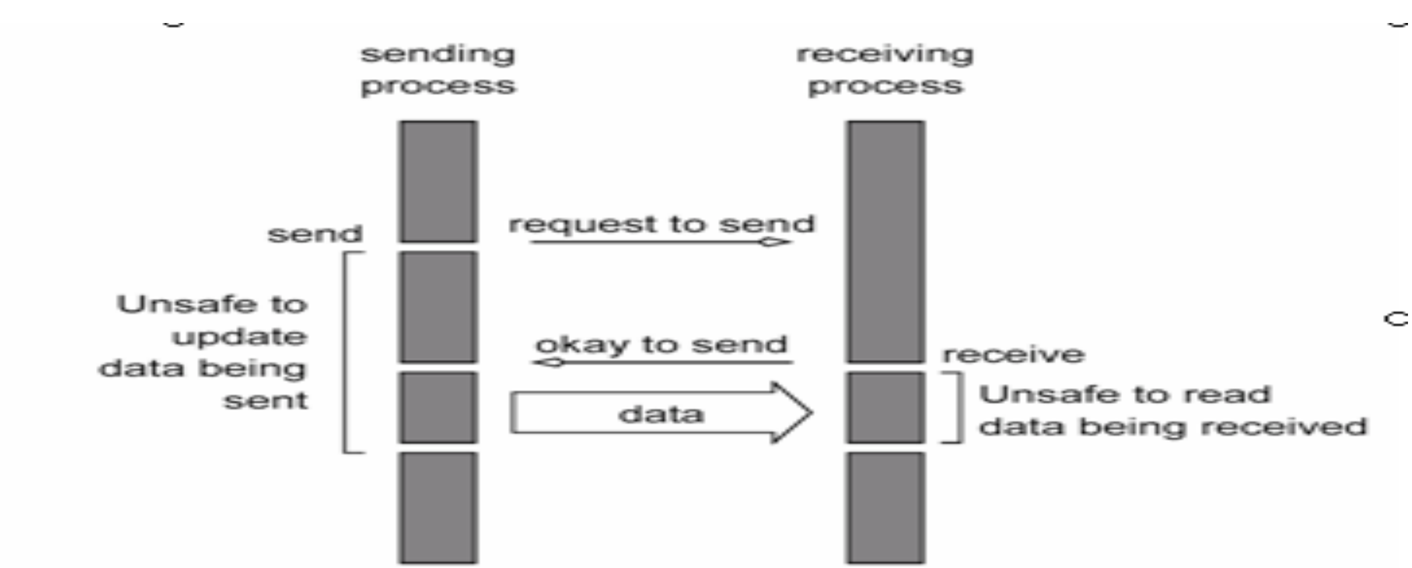
– A blocking receive only "returns" after the data has arrived and is ready for use by the program.



synchronous blocking

Blocking vs. Non-blocking (2/2)

- Non-blocking send and receive return almost immediately.
  - They do not wait for any communication events to complete, such as message copying from user memory to system buffer space or the actual arrival of message.
  - Non-blocking operations simply "request" the MPI library to perform the operation when it is able to. The user cannot predict when that will happen.
  - It is unsafe to modify the application buffer (your variable space) until you know for sure the requested non-blocking operation was actually performed by the library. There are "wait" routines used to do this.
  - Non-blocking communications are primarily used to overlap computation with communication and exploit possible performance gains.



High-level Topologies and Embedding

- MPI view of process topology => one-dimensional with linear ordering.
  - ➢ Need to map each MPI process into a higher dimensional topology Cartesian (grid) and Graph.
- Virtual MPI topologies
  - ➢ Provide convenience for applications whose communication patterns match an MPI topology structure.
  - ➢ Provide opportunity for implementation to optimize process mapping based on the physical characteristics of a given parallel machine.

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

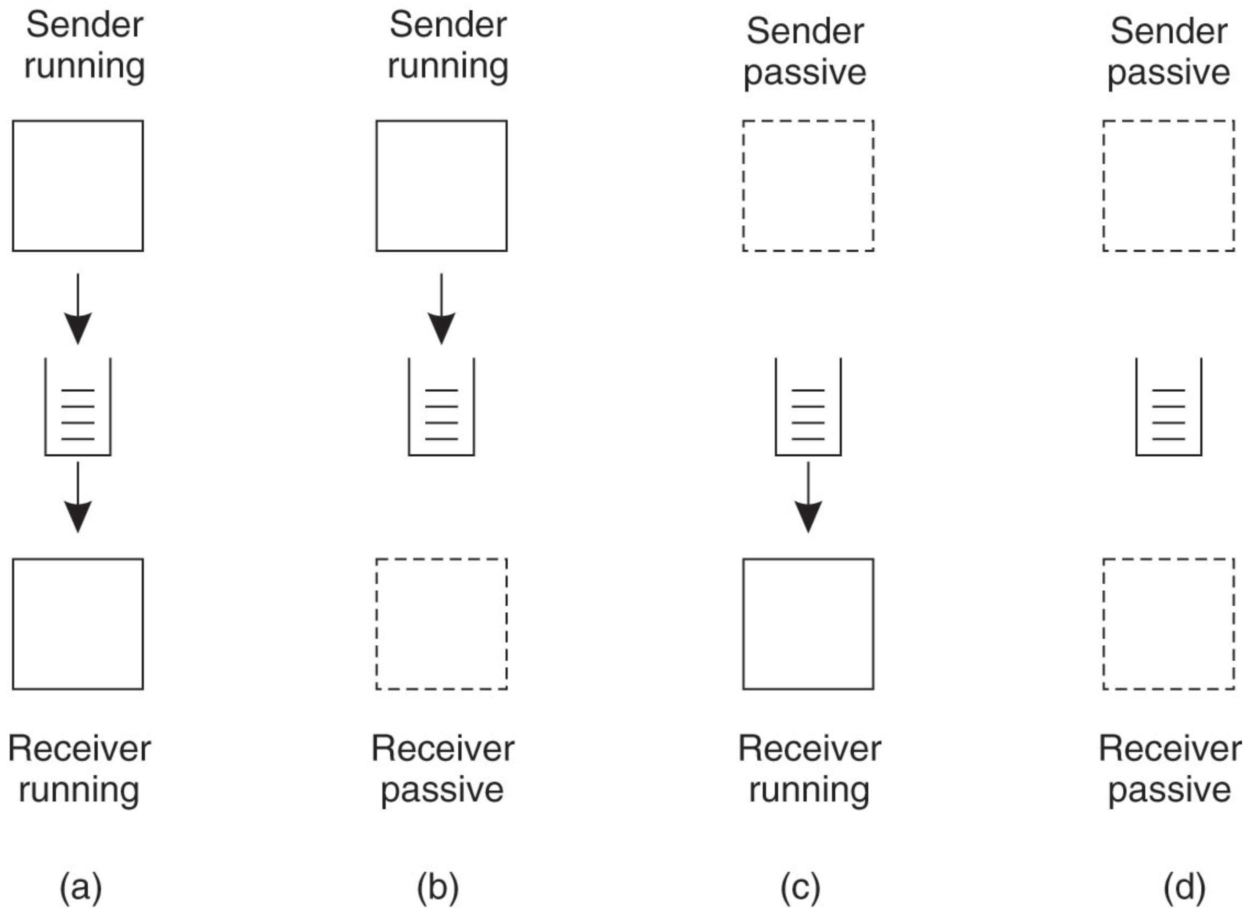
Message-oriented Persistent Comm.

- Message-queuing systems, or just message-oriented middleware (MOM)
- The essence of these systems is
  - ✓ Offer intermediate storage for messages
  - ✓ Do not require sender/receiver to be active during transmission
- Compared to sockets
  - ✓ They are typically used to support message transfers that take minutes, instead of seconds

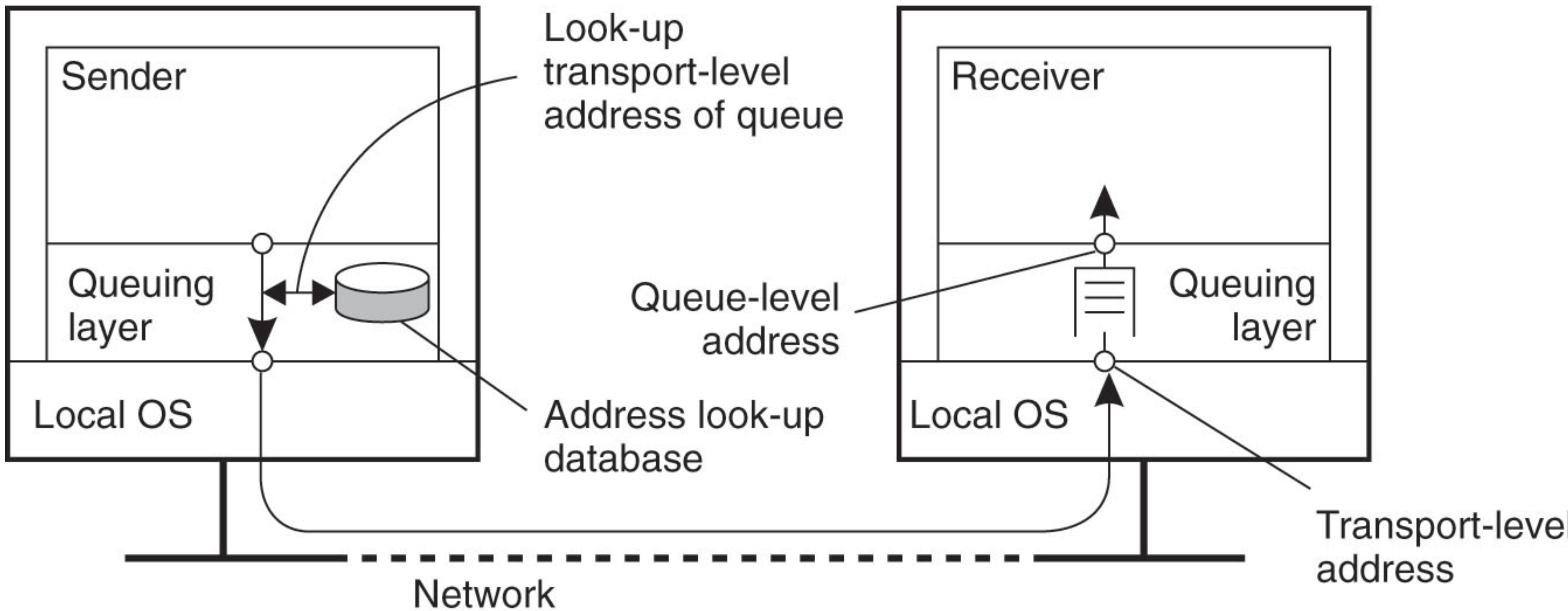
Message-queuing Model

- Insert messages in specific queues
- Messages are sent through various communication servers (e.g., mail servers)
- No guarantee of when the message is delivered
- Basic MQM primitives
  - ✓ Put: append a message to a queue
  - ✓ Get: retrieve (remove) the first message of a queue
  - ✓ Poll: check a specific queue for messages
  - ✓ Notify: install a handler to be called when a message is put into the specified queue

Four Combinations for MQM



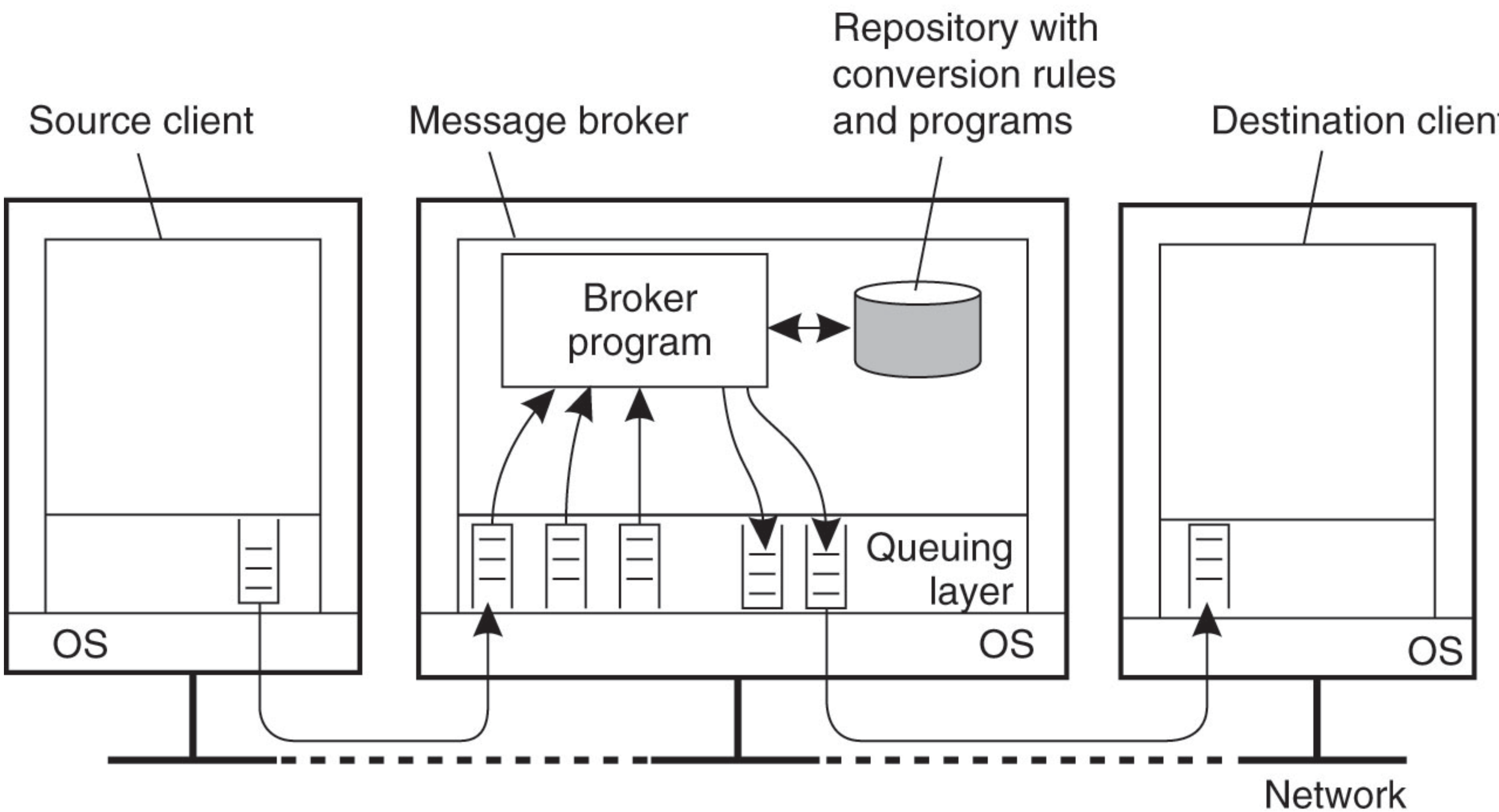
General Architecture



Message Brokers (1/2)

- Message-queuing systems can be used to integrate existing and new applications into a single, coherent distributed information systems
- Senders and receivers have to agree on the format of messages
- Message brokers
  - ✓ The main task is to convert messages so that they can be understood by the receiving applications

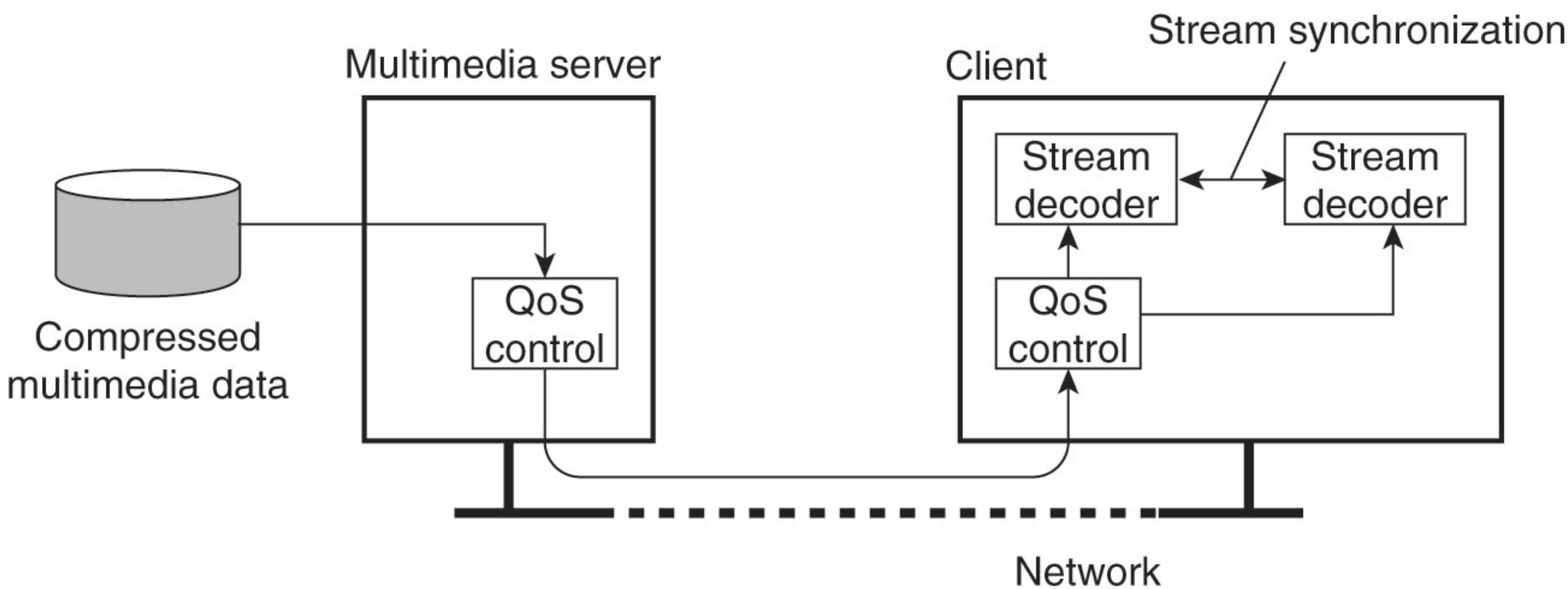
Message Brokers (2/2)



Stream-oriented Communication

- Communication discussed so far focuses on transmission of independent and complete unit of information
- The transmission of time-dependent information, e.g., video and audio streams is different
- Asynchronous transmission: streams transmitted one after the other (i.e., no constraint on when transmission should take place)
- Synchronous transmission: A maximum end-to-end delay is defined
- Isochronous transmission: It is necessary that data items are transmitted on time
    - ✓ e.g., bounded jitter

Streaming Stored Multimedia Data



A general architecture for streaming stored multimedia data over a network.

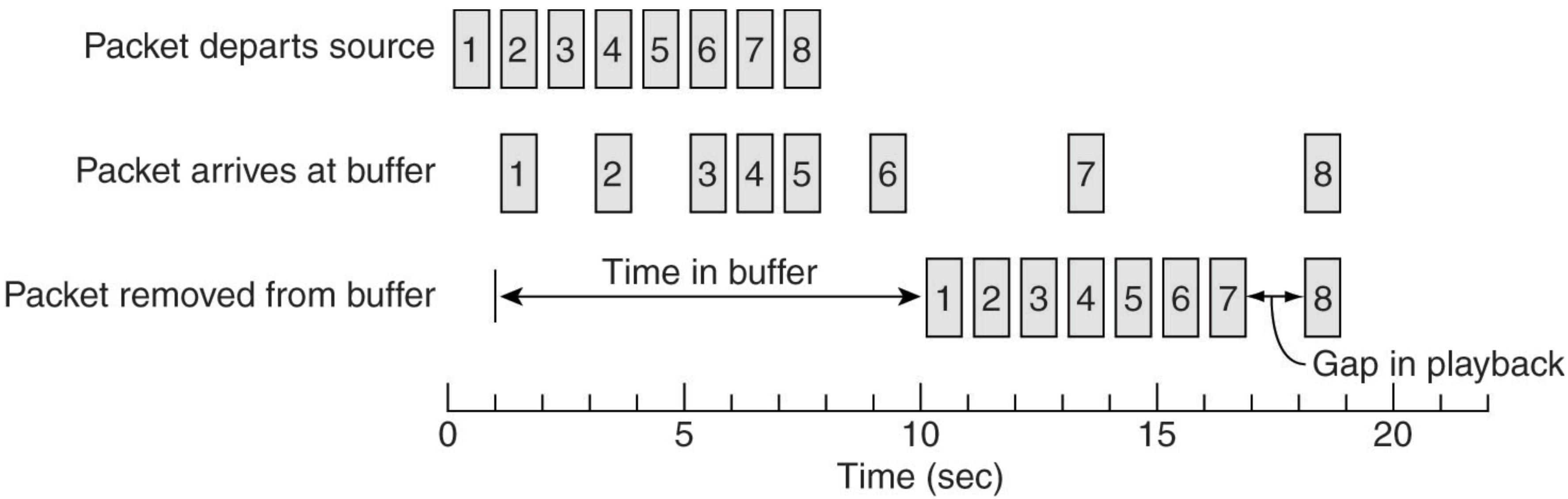
Stream and Quality of Service (QoS)

- Important streaming QoS requirements
  - ✓ The required bit rate
  - ✓ 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

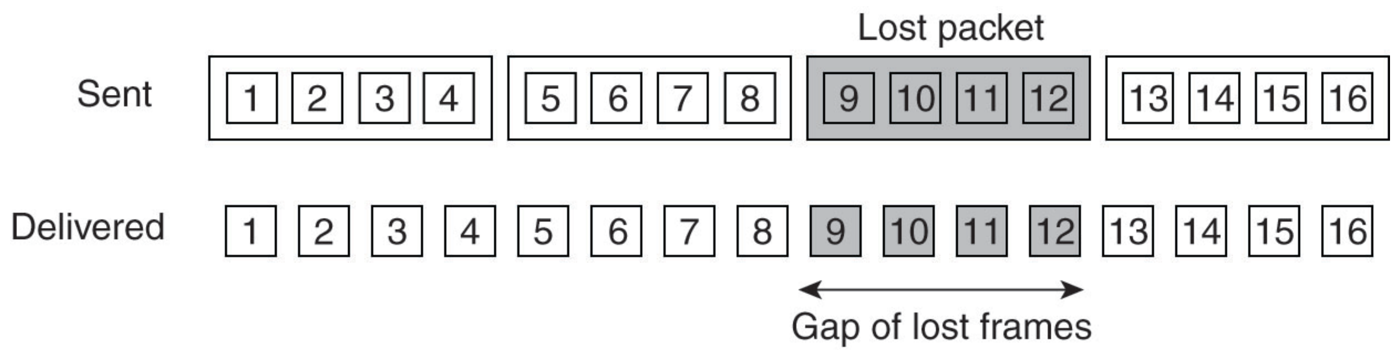
Enforcing QoS

- Very challenging if the underlying system only offers a best-effort delivery service
- A distributed system can try to conceal as much as possible of the lack of QoS
  - ✓ Differentiated services provided by Internet
  - ✓ Using a buffer to reduce jitter
  - ✓ Forward error correction
  - ✓ Interleaving communication

Using a Buffer to Reduce Jitter

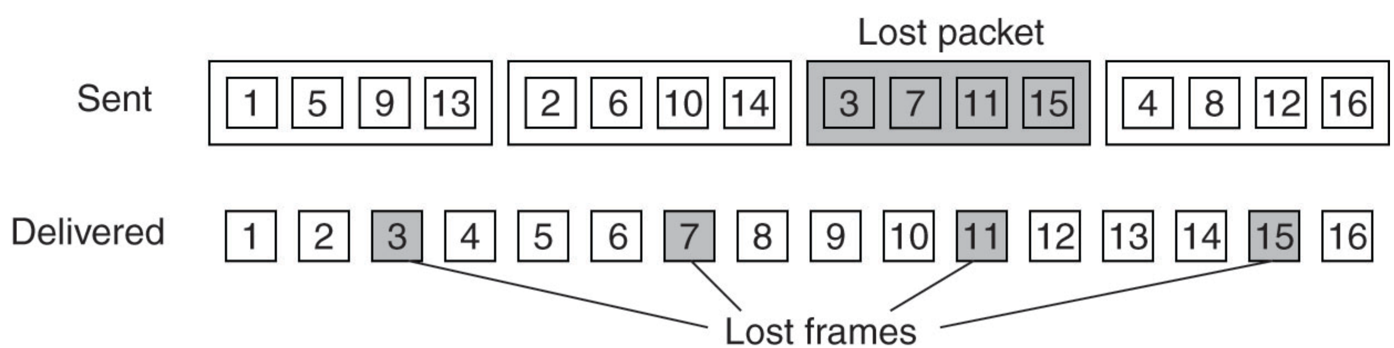


Interleaving Transmission



(a)

Lost frames felt badly



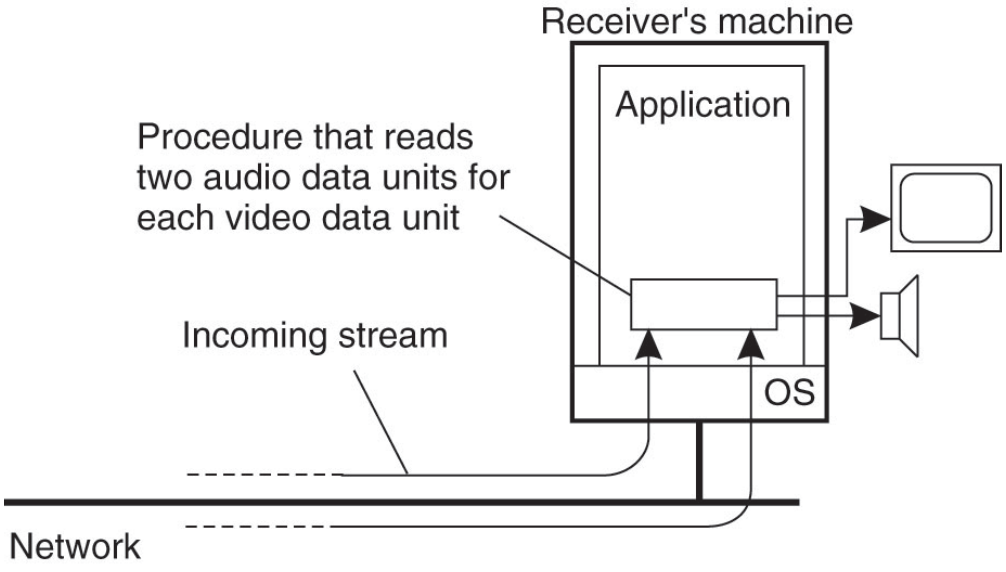
(b)

Pro: gap distributed over time

Con: must have 4 packets for playing

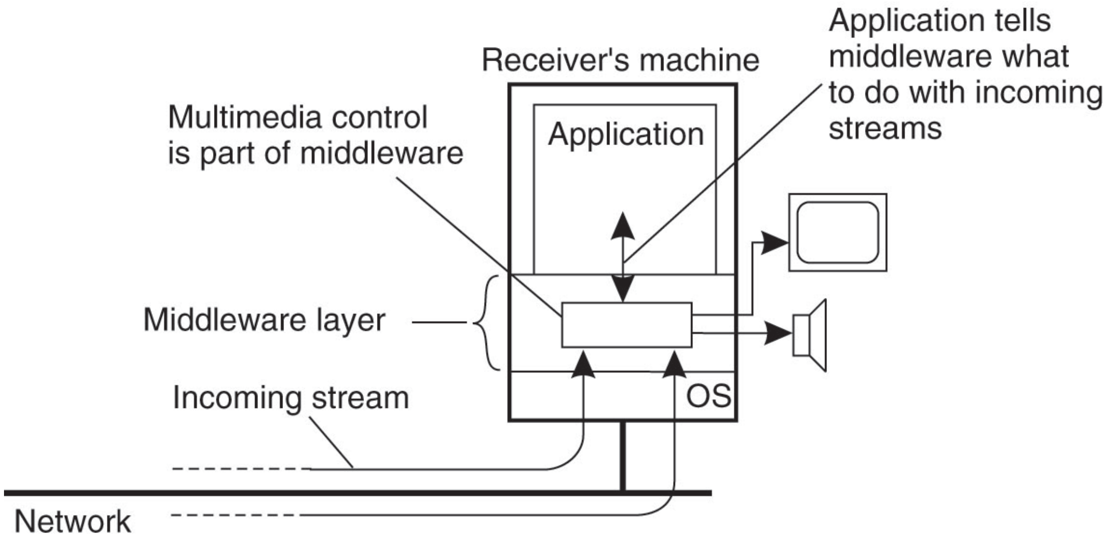
Stream Synchronization

- Different streams may be mutually synchronized
  - ✓ e.g., audio and video streams in movies
- Explicit synchronization on the level of data units



Synchronization by Middleware

- Problem with explicit synchronization
  - ✓ Applications are responsible for implementing synchronization



Multicasting

- Sending data to multiple receivers simultaneously
- Node organize into an overlay network, within which messages can be disseminated to members
- Two ways for organizing the overlay network
  - ✓ Tree-based network: a unique path between every pair of nodes
  - ✓ Mesh network: every node will have multiple neighbors, hence multiple paths between nodes

Quality of Multicast Tree

- Link stress: number of packets cross a link
- Stretch or relative delay penalty (RDP)
  - ✓ Measures the ratio in the delay between two nodes and the delay that those two would experience in the underlying network
- Tree cost: related to minimizing the aggregated link cost

Gossip-based Data Dissemination

- Disseminating information based on epidemic behavior
- Epidemic protocol
  - ✓ Rapidly propagate information among a large collection of nodes using only local information
- A popular propagation model for epidemic protocol is the anti-entropy model
  - ✓ Node P picks another Q at random
  - ✓ Subsequently exchanges updates with Q
    - P only pushes updates to Q
    - P only pulls updates from Q
    - P and Q send updates to each other

Example: Gossiping

- Keep sending info to all other nodes until you first find out that they have already received it
- Directional gossiping
  - ✓ Take network topology into account
  - ✓ e.g., send to small-degree nodes with a higher probability