COP 6611 Advanced Operating System

Communication

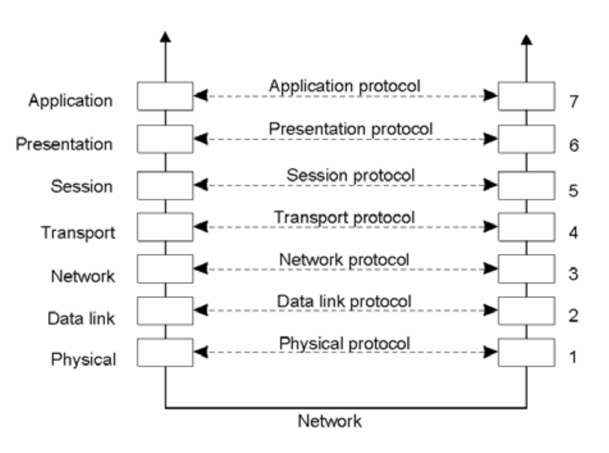
Chi Zhang

czhang@cs.fiu.edu

Outline

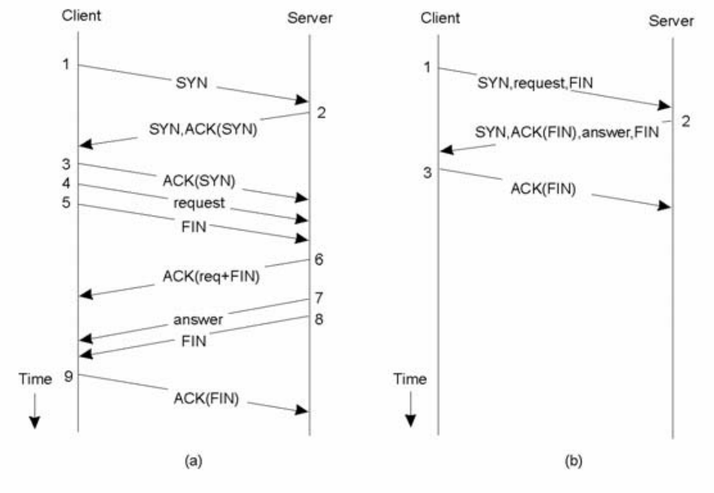
- Layered Protocols
- Remote Procedure Call (RPC)
- Remote Object Invocation
- Message-Oriented Communication

Layered Protocols



Each layer can be changed without the other ones being affected.

Client-Server TCP



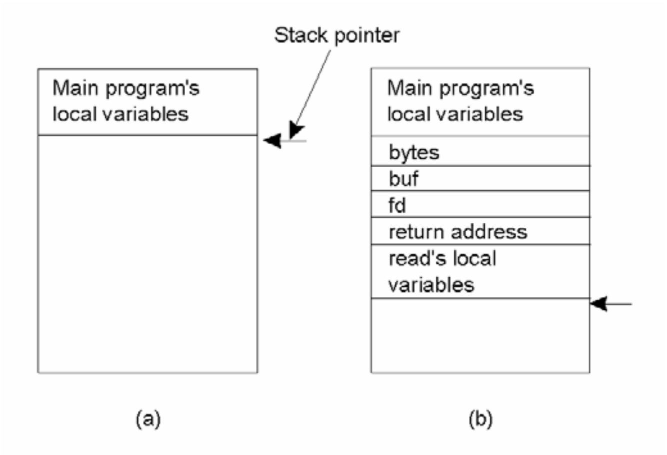
a) Normal operation of TCP.
b) Transactional TCP.

Application Protocols

- Application Protocols ≠ Applications
- HyperText Transfer Protocol (HTTP)
  - HTML
  - Java RMI
  - XML / SOAP
  - Client-Server (Request-Reply)
  - Not blocked by firewall

5

Conventional Procedure Call

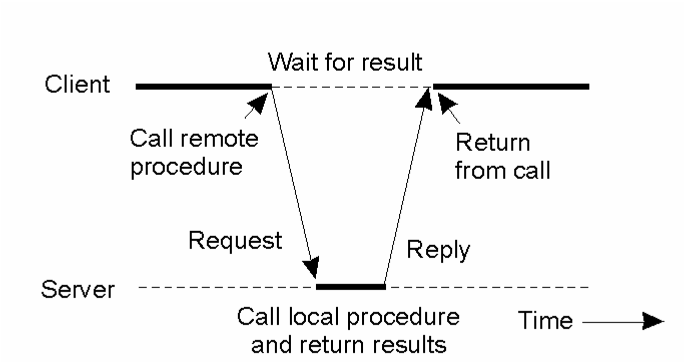


a) Parameter passing in a local procedure call: the stack before the call to read

b) The stack while the called procedure is active

6

Client and Server Stubs



Principle of RPC between a client and server program.

Steps of a Remote Procedure Call

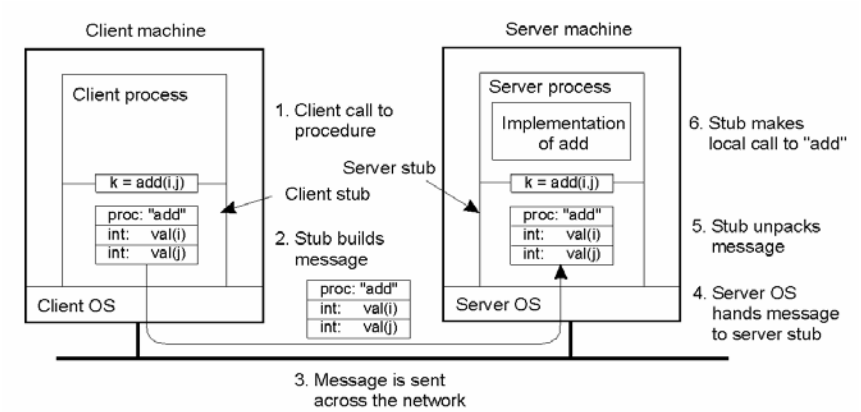
1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it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

Net effect: RPC as if LPC (local)

Passing Value Parameters (1)

- Which procedures call?
- Machines have different data representations.
- Big Endians and Endian?
- How to pass pointers?
  - Copy array into messages.
  - Input or Output?
  - Cannot handle lists
- Interface Definition Languages (IDL)
  - A collection of procedures
  - Compiled into client or server stub.

Passing Value Parameters (2)

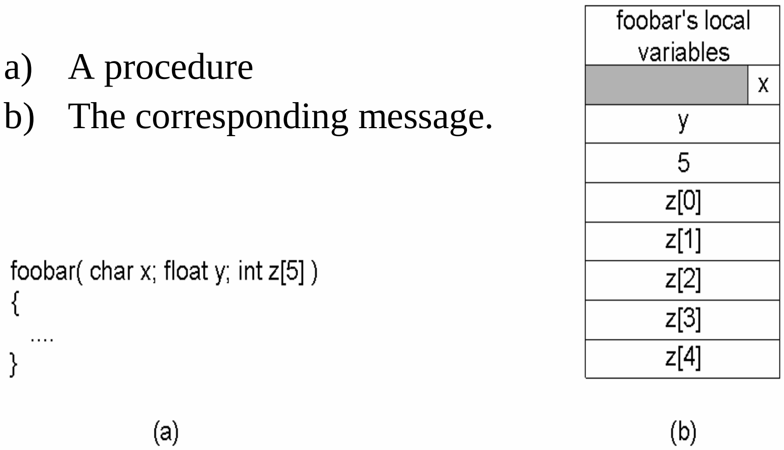


Steps involved in doing remote computation through RPC

Parameter Specification and Stub Generation

a) A procedure

b) The corresponding message.



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

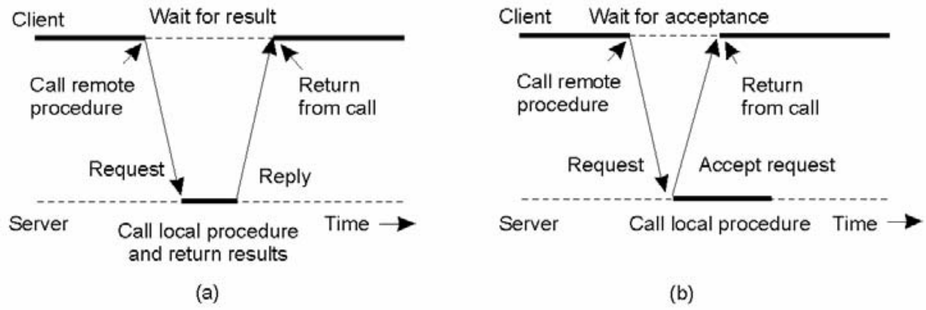
(a)

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

(b)

11

Asynchronous RPC

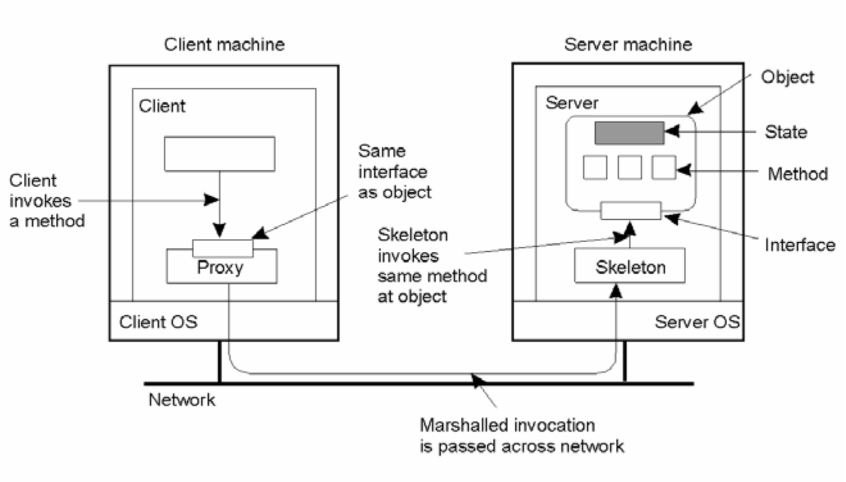


a) The interconnection between client and server in a traditional RPC

b) The interaction using asynchronous RPC

12

Distributed Objects



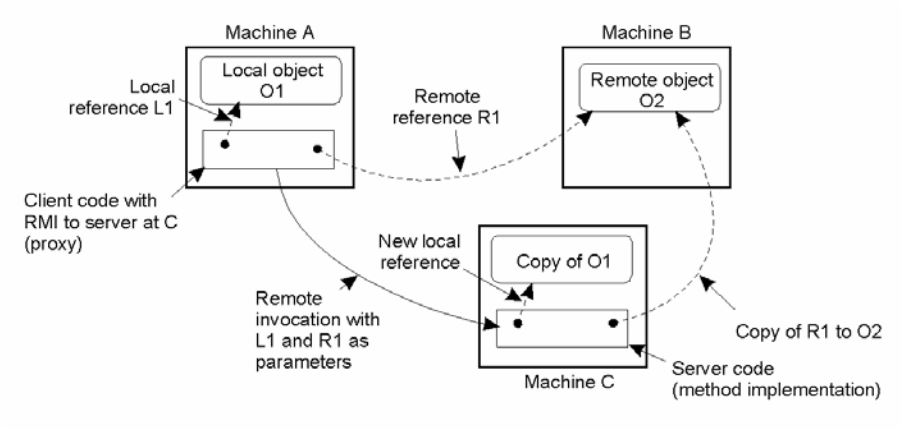
Common organization of a remote object with client-side proxy.

Binding a Client to an Object

```
Distr_object* obj_ref;                //Declare a systemwide object reference
obj_ref = ...;                        // Initialize the reference to a distributed object
obj_ref-> do_something();             // Implicitly bind and invoke a method

                              (a)

Distr_object objPref;                 //Declare a systemwide object reference
Local_object* obj_ptr;                //Declare a pointer to local objects
obj_ref = ...;                        //Initialize the reference to a distributed object
obj_ptr = bind(obj_ref);              //Explicitly bind and obtain a pointer to the local proxy
obj_ptr -> do_something();            //Invoke a method on the local proxy

                              (b)
```

a) (a) Example with implicit binding using only global references
b) (b) Example with explicit binding using global and local references

Parameter Passing



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

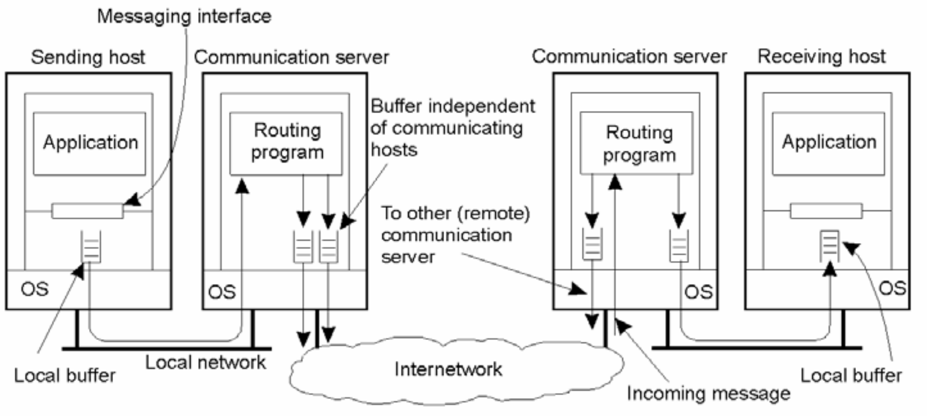
15

Other Issues

- Object Reference
  - network address + endpoint (TCP port) + object ID
  - remote registry
- Static vs. Dynamic Invocation
  - fobject.append(int)
  - invoke(fobject, id(append), int)
- Clone
- Synchronization
- A proxy can be serialized and sent to another process, to be used as a reference.

16

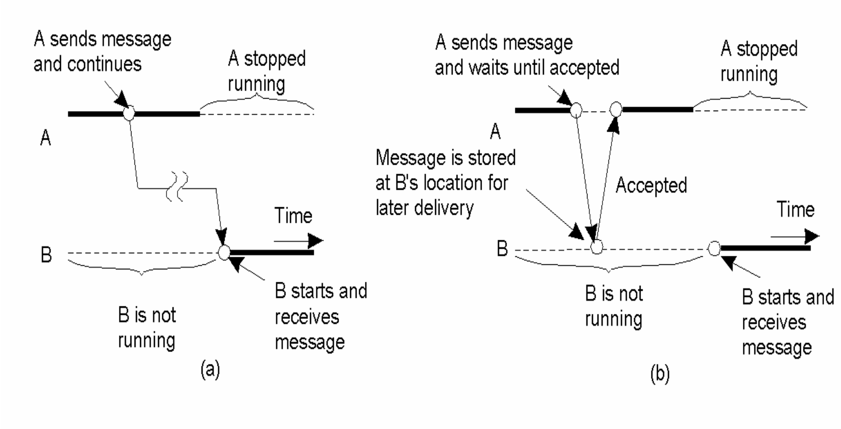
Persistence and Synchronicity in Communication (1)



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

17

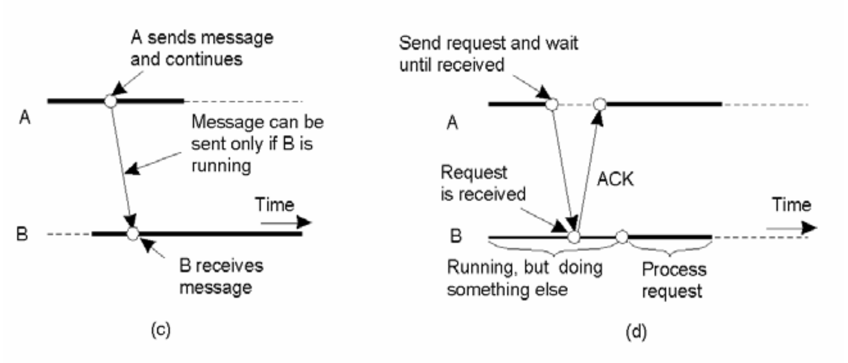
Persistence and Synchronicity in Communication (2)


A sends message and continues
A stopped running
A
B
Time
B is not running
B starts and receives message
(a)
A sends message and waits until accepted
A stopped running
A
Message is stored at B's location for later delivery
Accepted
Time
B
B is not running
B starts and receives message
(b)

a) Persistent asynchronous communication
b) Persistent synchronous communication

18

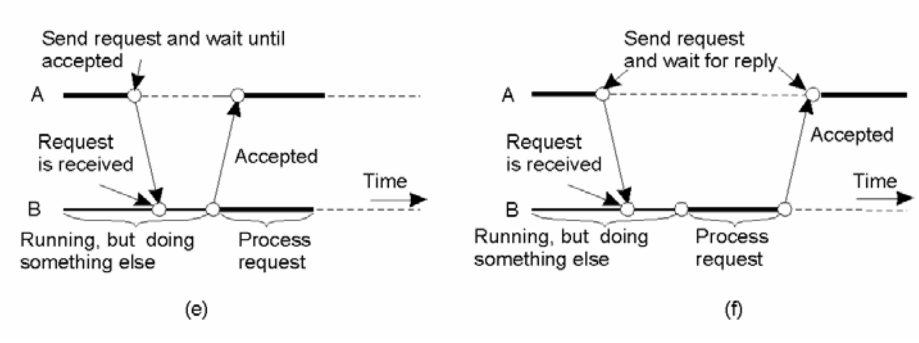
Persistence and Synchronicity in Communication (3)


A sends message and continues
A
Message can be sent only if B is running
Time
B
B receives message
(c)
Send request and wait until received
A
Request is received
ACK
Time
B
Running, but doing something else
Process request
(d)

c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication (4)


Send request and wait until accepted
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(e)
Send request and wait for reply
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(f)

e) Delivery-based transient synchronous communication at message delivery

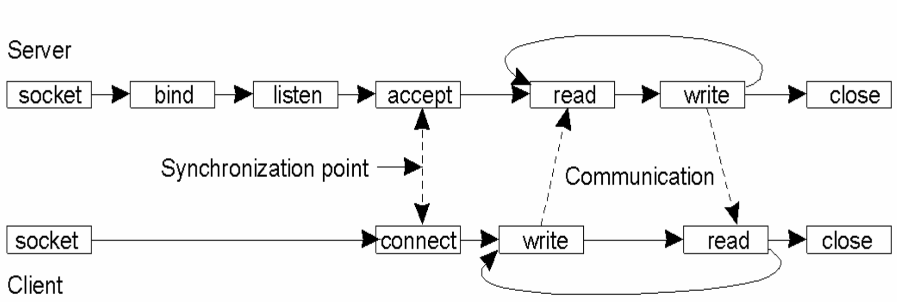
f) Response-based transient synchronous communication

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)



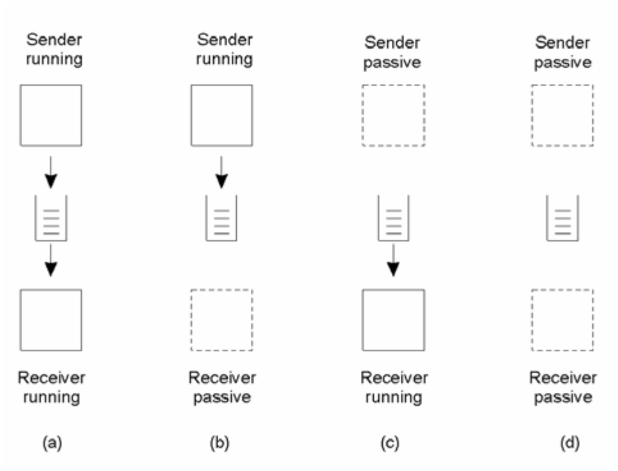
Connection-oriented communication pattern using sockets.

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-oriented Transient Communication

Message-Queuing Model (1)



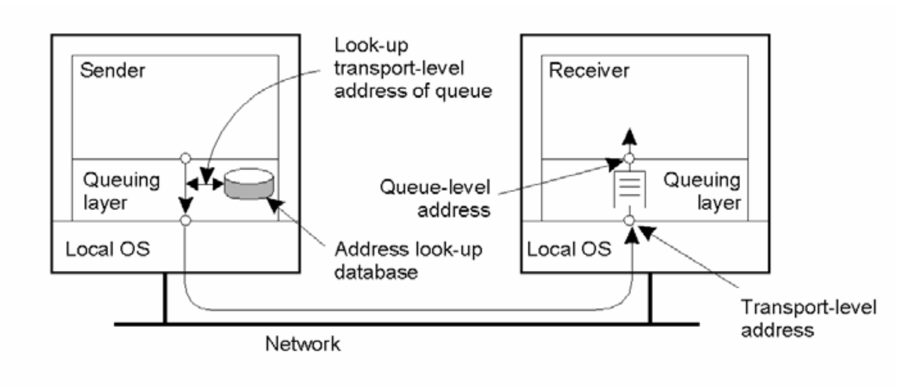
Persistent Asynchronous Communication: Intermediate-term storage for messages, without requiring the sender or receiver always active

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.

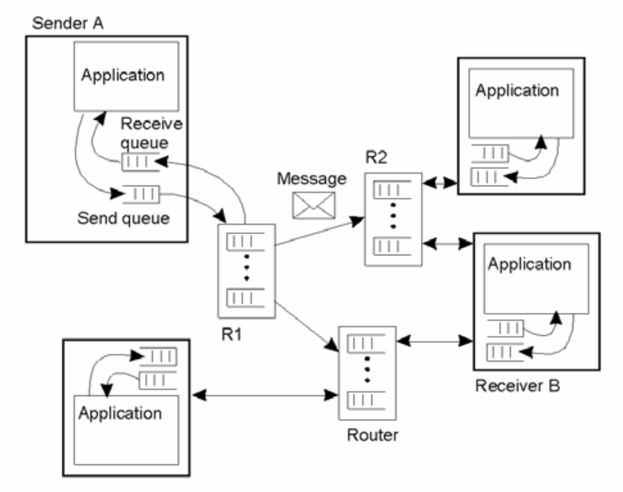
Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System (1)



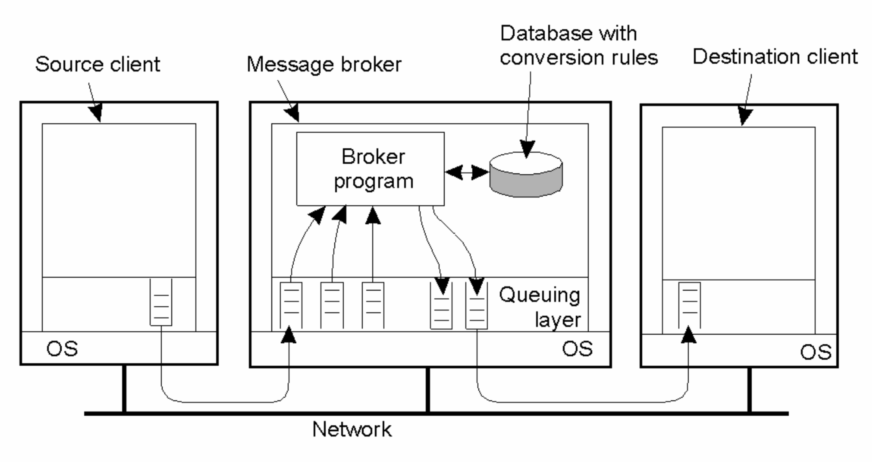
The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System (2)



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

Message Brokers



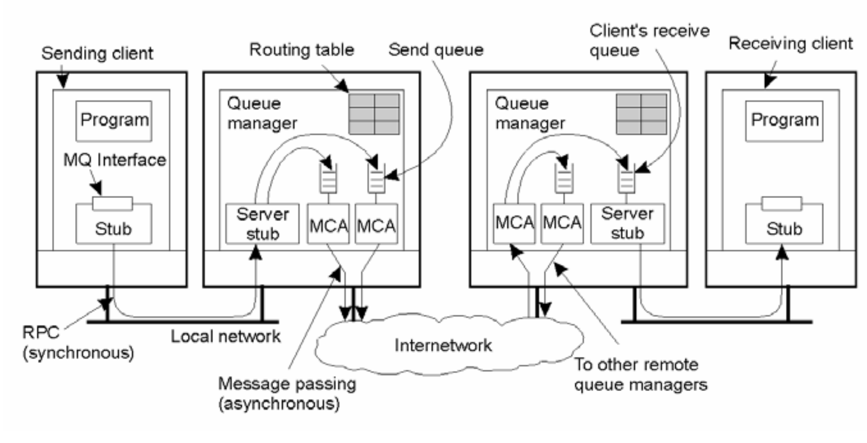
The general organization of a message broker in a message-queuing system.

Applications of Message-Queuing Systems

- Applications with more complex requirements than Emails
  - Guaranteed delivery
  - Priorities
  - Logging
  - Multicast
  - Load balancing
  - Fault tolerating
  - Transaction
- E.g. Integrate a collection of database applications

29

Example: IBM MQSeries



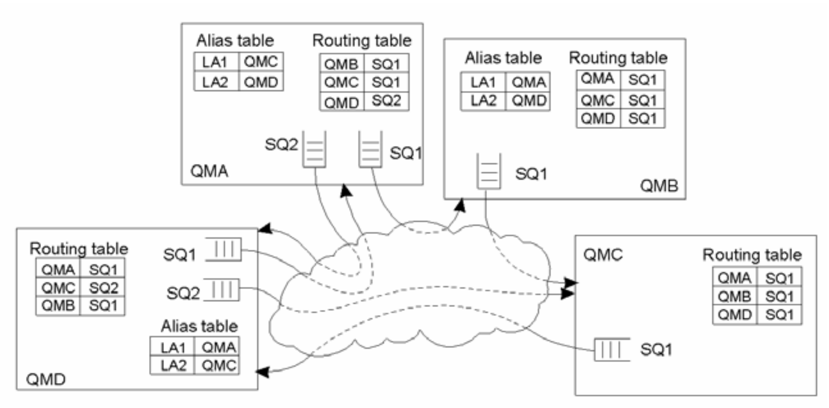
General organization of IBM's MQSeries message-queuing system.

30

Channels

- Transfer along the channel can take place only if both its sending and receiving MCA are running.
- Configure the send queue to set off a trigger when a message is enqueued.
  - The trigger starts the sending MCA
- The sending MCA sends a control message requesting the other MCA to be started.
  - A daemon listens to a well-known address
- Channels are stopped automatically after a specified idle time.

Message Transfer (1)



QMA

Alias table

LA1	QMC
LA2	QMD

Routing table

QMB	SQ1
QMC	SQ1
QMD	SQ2

QMB

Alias table

LA1	QMA
LA2	QMD

Routing table

QMA	SQ1
QMC	SQ1
QMD	SQ1

QMC

Routing table

QMA	SQ1
QMB	SQ1
QMD	SQ1

QMD

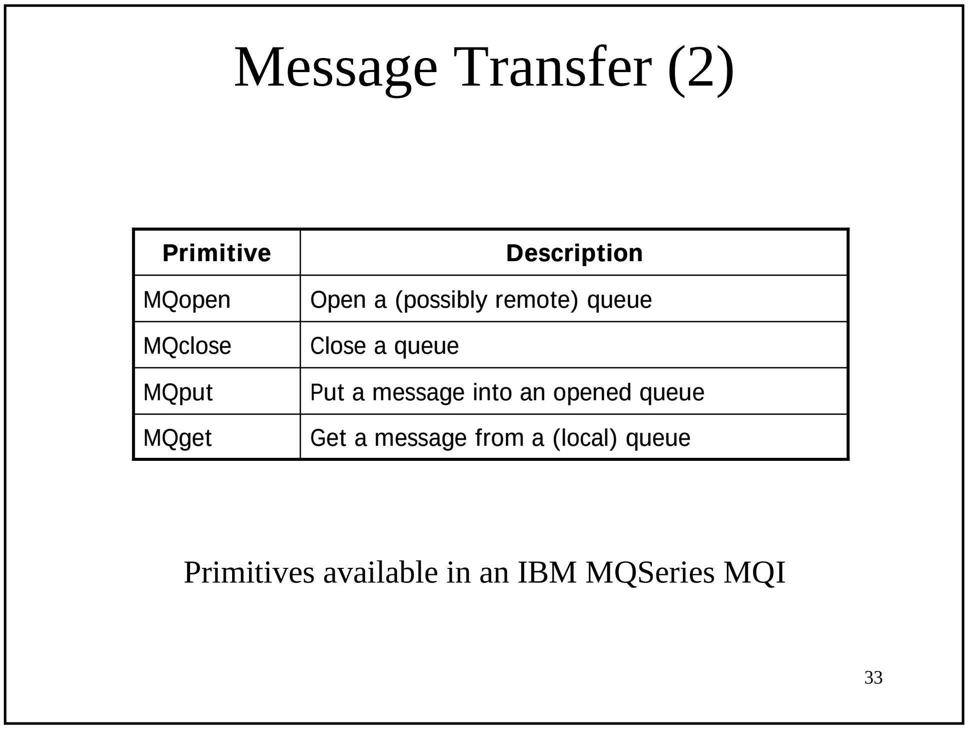
Routing table

QMA	SQ1
QMC	SQ2
QMB	SQ1

Alias table

LA1	QMA
LA2	QMC

The general organization of an MQSeries queuing network using routing tables and aliases.



Message Transfer (2)

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

Primitives available in an IBM MQSeries MQI