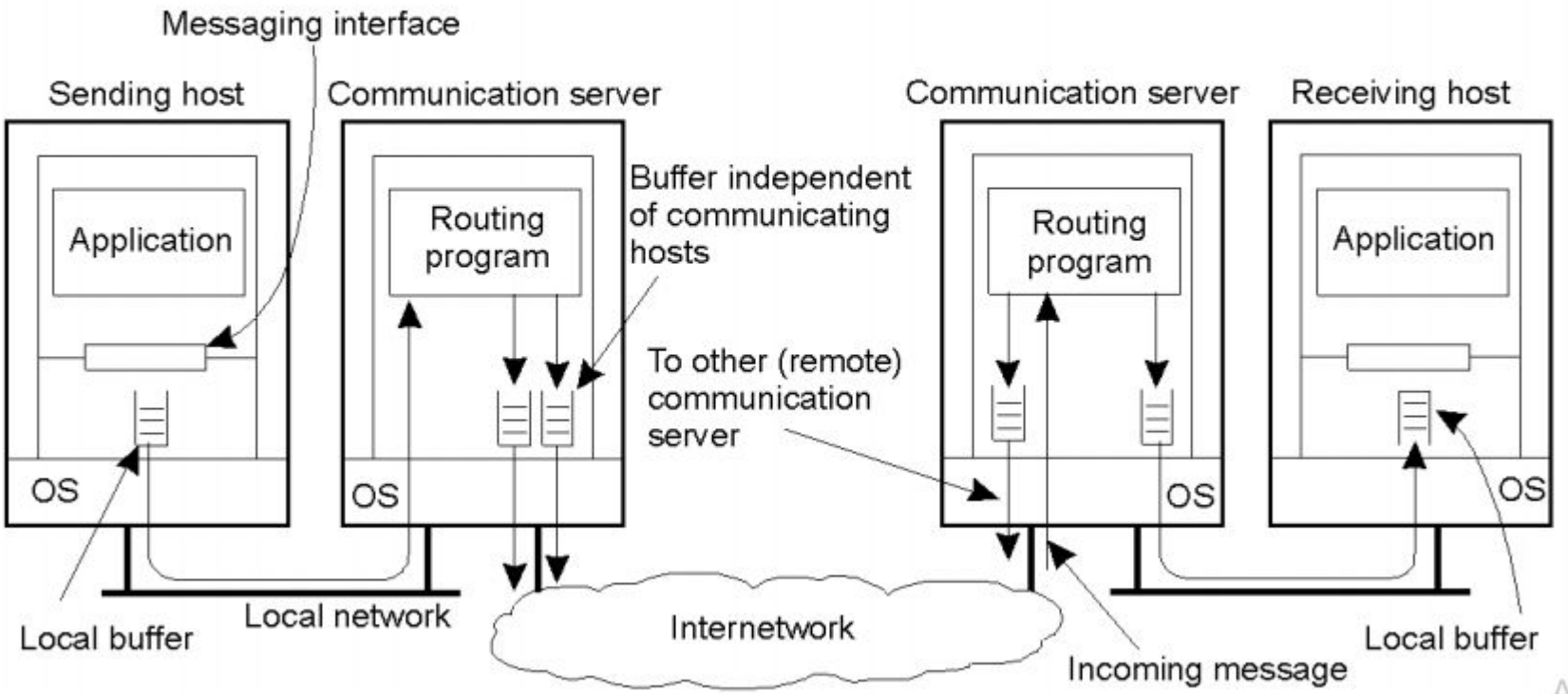
Message-Oriented Communication

Message-Oriented Communication

- Applications are executed on hosts
- The hosts are connected through a network of communication servers
- Each host is connected to some communication server
- **Example: Electronic Mail System**
  - Each host runs an application by which a user can compose, send, receive and read messages.
  - Each host is connected to a mail server
  - Each message is first stored in one of the output buffers of the local mail server.
  - The server removes messages from its buffers and sends them to their destination.
  - The target mail server stores the message in an input buffer for the designated receiver (in the receiver's mailbox).
  - The interface at the receiving host offers a service to the receiver's user agent by which the latter can regularly check for incoming mail.

Persistence and Synchronicity in Communication



Persistence and Synchronicity in Communication

- Persistent communication
  - a message that has been submitted for transmission is stored by the communication system as long as it takes to deliver it to the receiver.
- Transient communication
  - a message is stored by the communication system only as long as the sending and receiving application are executing.
    - If a communication server cannot deliver a message to the next communication server or the receiver, the message will be discarded
    - It works like a traditional store-and-forward router
- Asynchronous Communication
  - A sender continues its execution immediately after it has submitted its message for transmission
- Synchronous Communication
  - The sender is blocked until its message is stored in a local buffer at the receiving host, or actually delivered to the receiver.

Persistent vs. Transient Communication

Persistent communication: A message is stored at a communication server as long as it takes to deliver it at the receiver.

Transient communication: A message is discarded by a communication server as soon as it cannot be delivered at the next server, or at the receiver.

Synchronous Communication

Some observations: Client/Server computing is generally based on a model of synchronous communication:

- **Client and server have to be active at the time of communication**
- **Client issues request and blocks until it receives reply**
- **Server essentially waits only for incoming requests, and subsequently processes them**

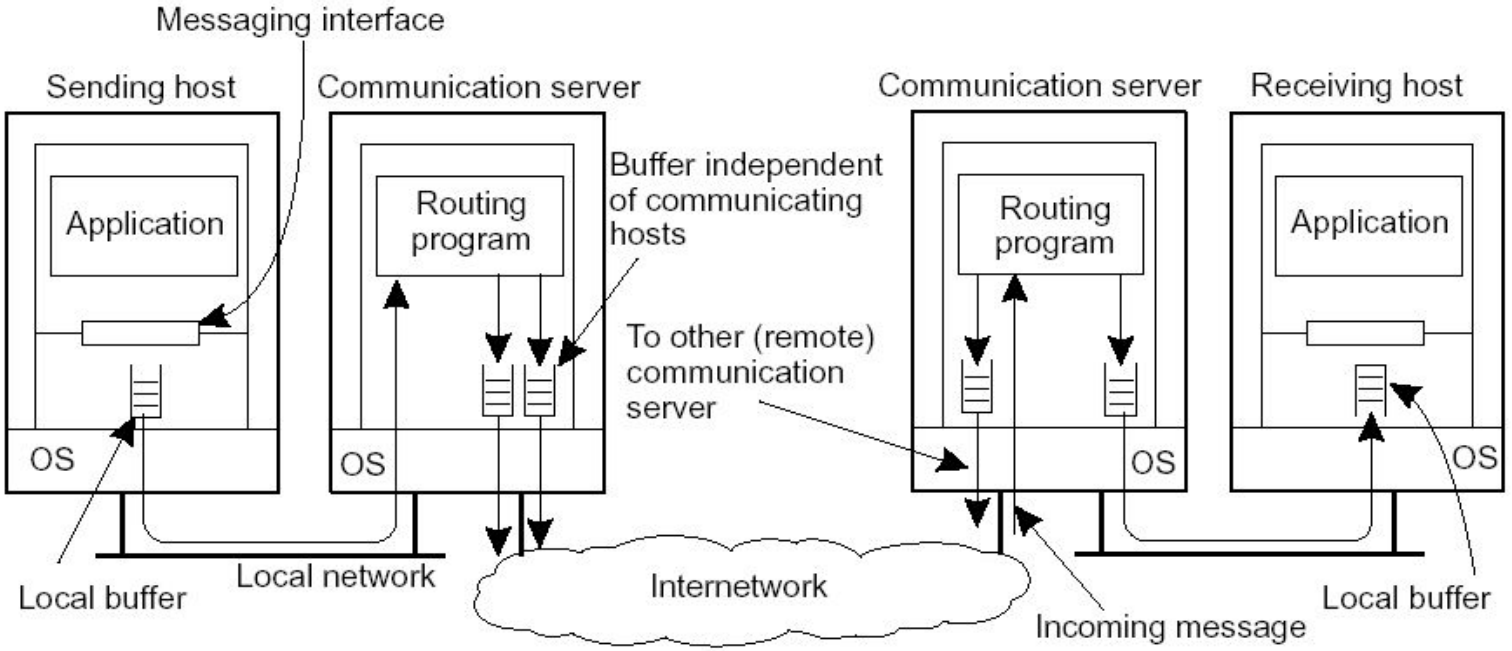
Drawbacks of synchronous communication:

- **Client cannot do any other work while waiting for reply**
- **Failures have to be dealt with immediately (the client is waiting)**
- **In many cases the model is simply not appropriate (mail, news)**

Asynchronous Communication: Messaging

Message-oriented middleware: Aims at high-level asynchronous communication:

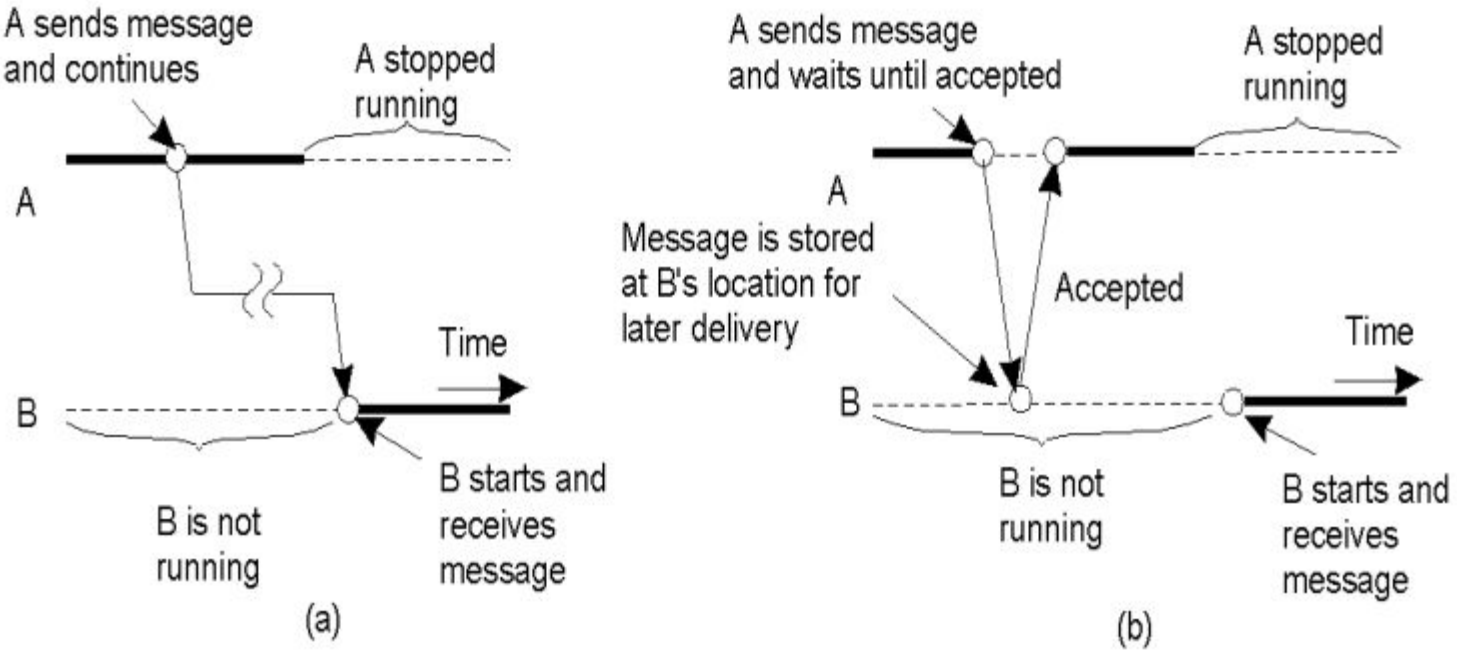
- **Processes send each other messages, which are queued**
- **Sender need not wait for immediate reply, but can do other things**
- **Middleware often ensures fault tolerance**



Persistence and Synchronicity in Communication

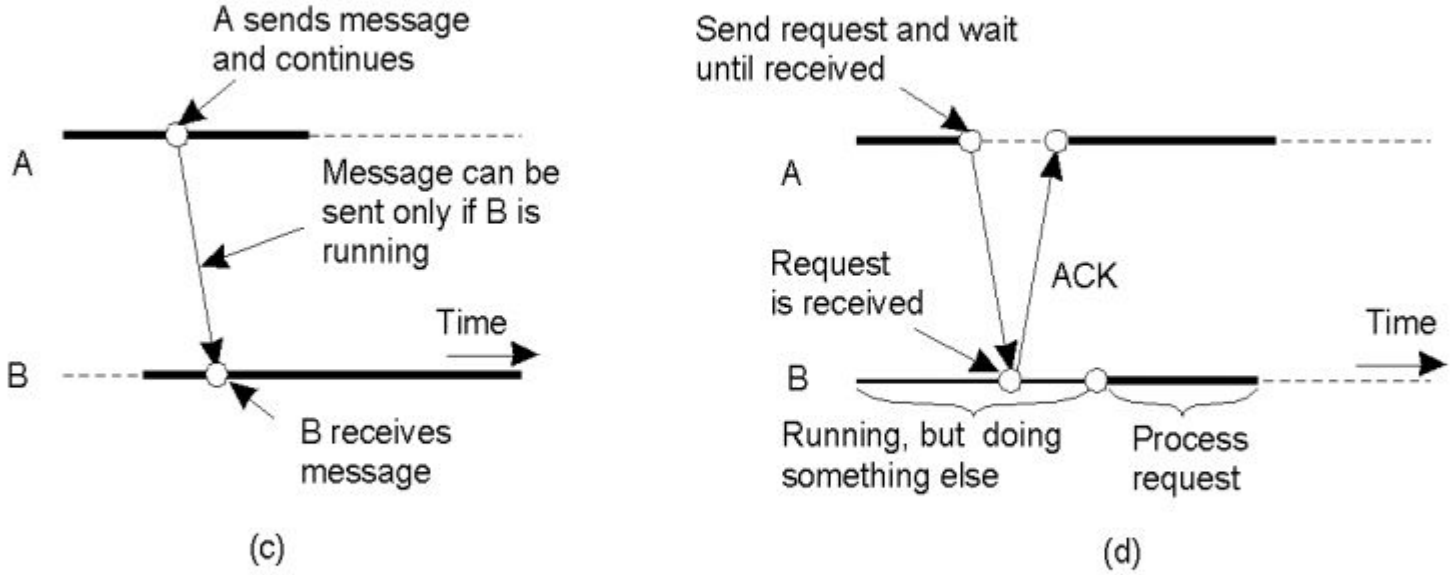
- Persistent Asynchronous Communication
  - Each message is either persistently stored in a buffer at the local host or at the first communication server.
    - A e-mail system is an example
- Persistent Synchronous Communication
  - Messages can be persistently stored at the receiving host and a sender is blocked until this happens
- Transient Asynchronous Communication
  - The message is temporarily stored at a local buffer at the sending host, after which the sender immediately continues
  - UDP is an example
- Transient Synchronous Communication
  - The sender is blocked until the message is stored in a local buffer at the receiving host, or
  - until the message is delivered to the receiver for further processing, or
  - until it receives a reply message from the other side (RPCs,RMIs)

Persistence and Synchronicity in Communication



a) Persistent asynchronous communication
b) Persistent synchronous communication

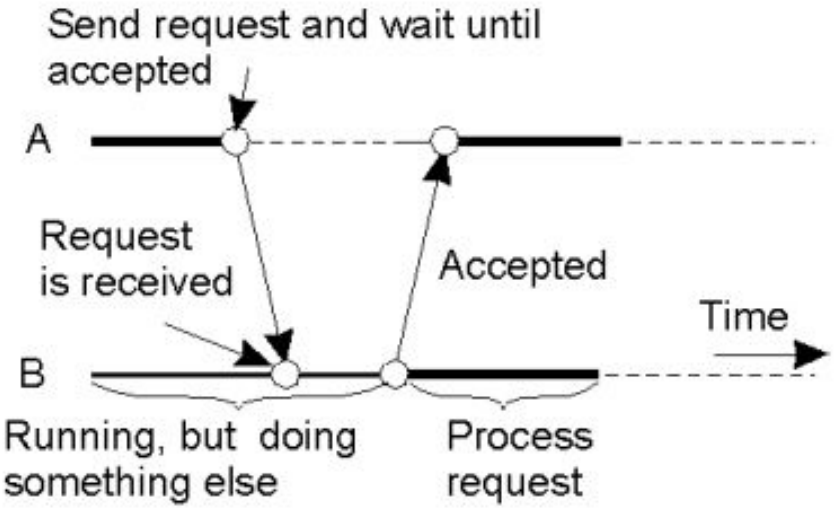
Persistence and Synchronicity in Communication



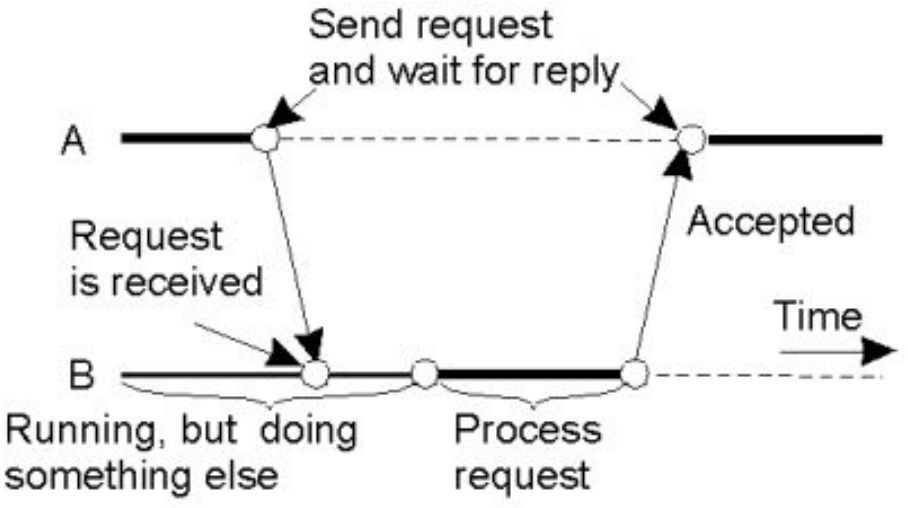
c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication



(e)



(f)

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

Message-Oriented Middleware

Essence: Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers.

Canonical example: IBM MQSeries

IBM MQSeries (1/3)

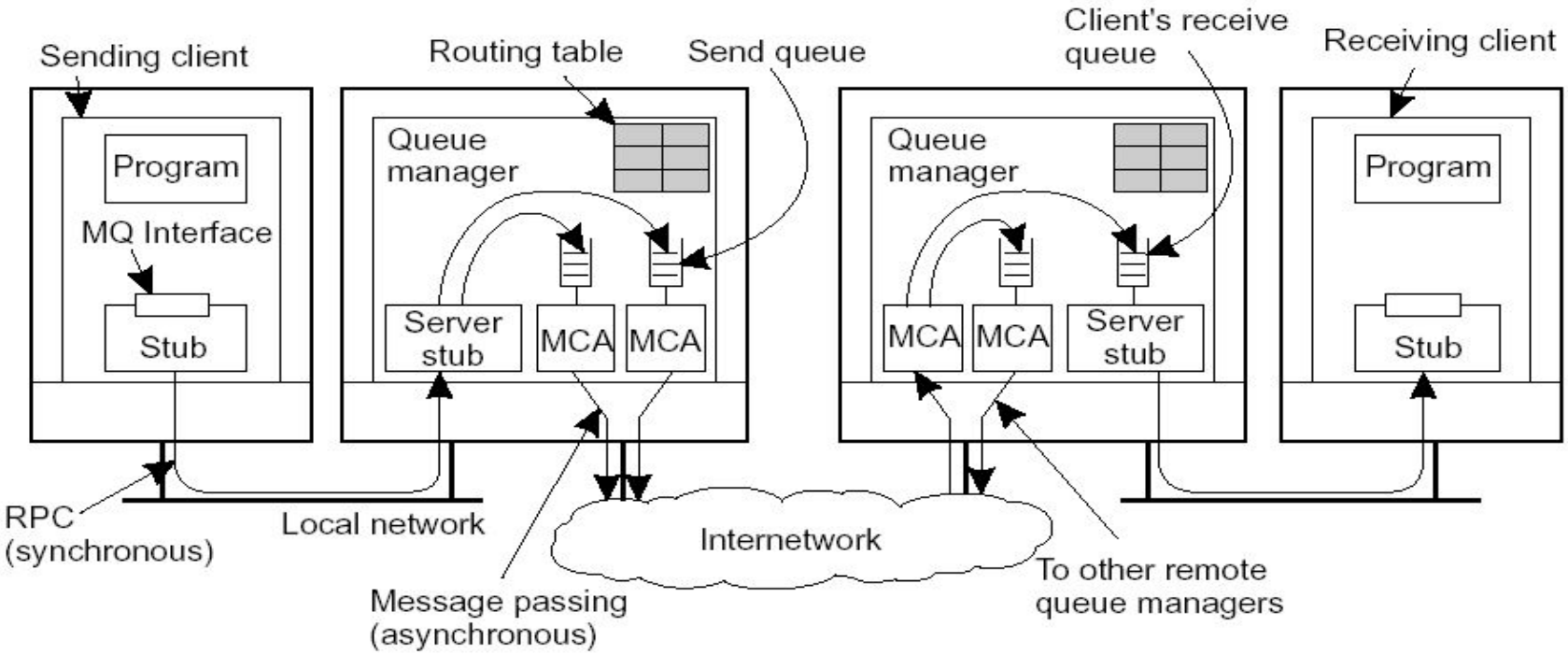
Basic concepts:

- **Application-specific messages are put into, and removed from queues**
- **Queues always reside under the regime of a queue manager**
- **Processes can put messages only in local queues, or through an RPC mechanism**

Message transfer:

- **Messages are transferred between queues**
- **Message transfer between queues at different processes, requires a channel**
- **At each endpoint of channel is a message channel agent (MCA)**
  - **Setting up channels using lower-level network communication facilities (e.g., TCP/IP)**
  - **(Un)wrapping messages from/in transport-level packets**
  - **Sending/receiving packets**

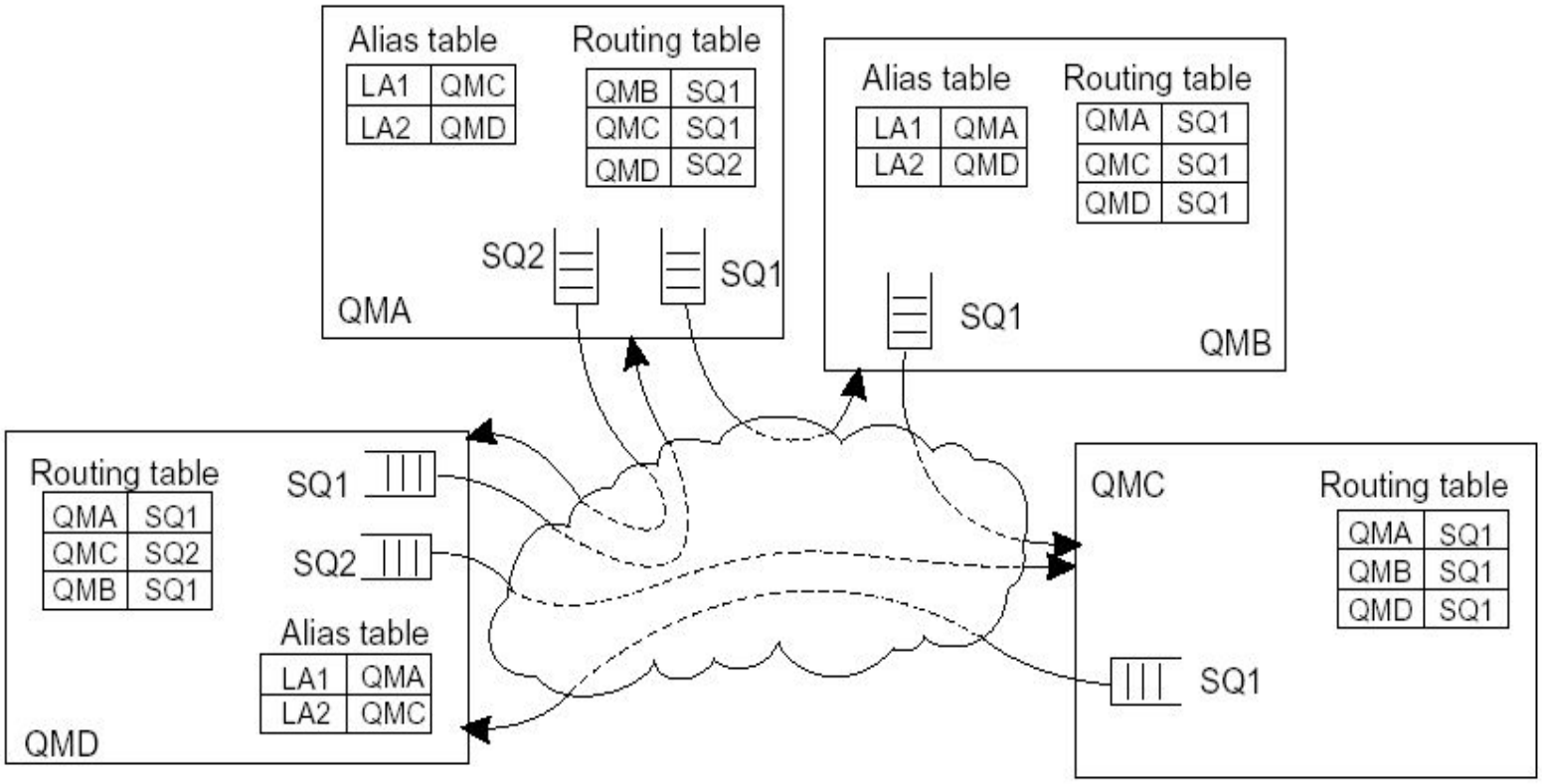
IBM MQSeries (2/3)



- **Channels are inherently unidirectional**
- **MQSeries provides mechanisms to automatically start MCAs when messages arrive, or to have a receiver set up a channel**
- **Any network of queue managers can be created; routes are set up manually (system administration)**

IBM MQSeries (3/3)

Routing: By using logical names, in combination with name resolution to local queues, it is possible to put a message in a remote queue

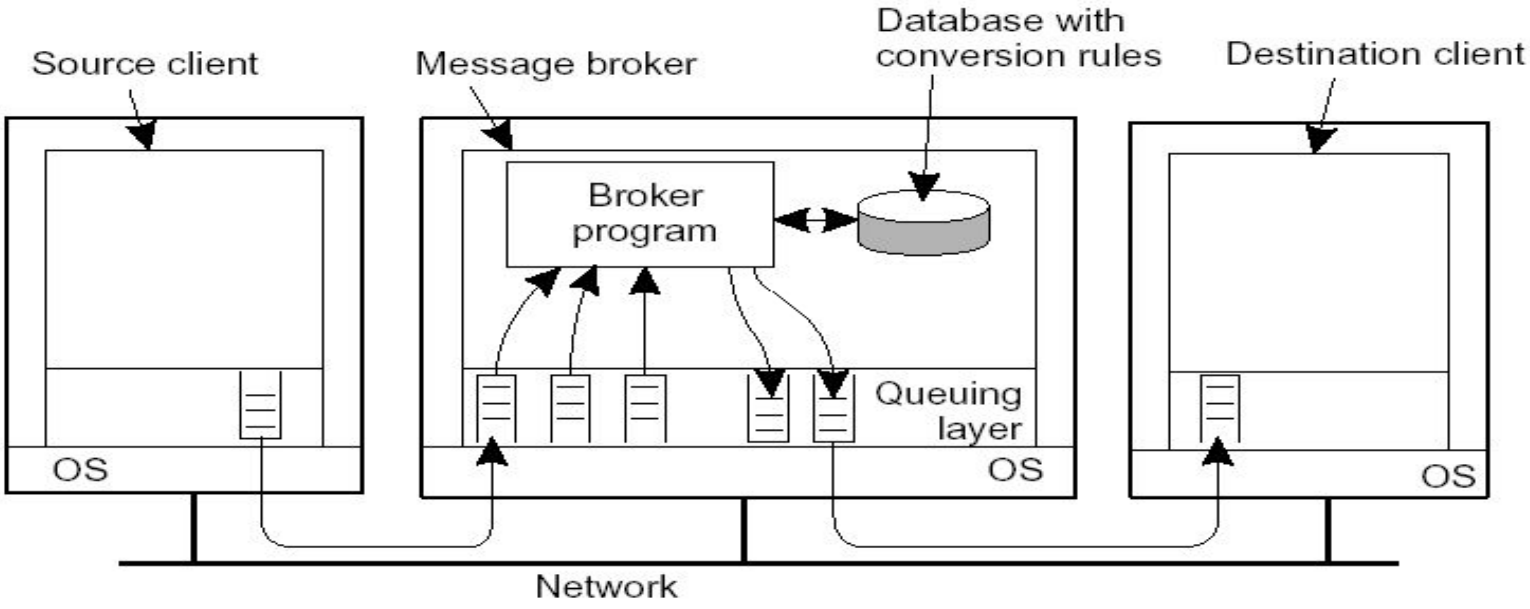


Message Broker

Observation: Message queuing systems assume a *common messaging protocol*: all applications agree on message format (i.e., structure and data representation)

Message broker: Centralized component that takes care of application heterogeneity in a message-queuing system:

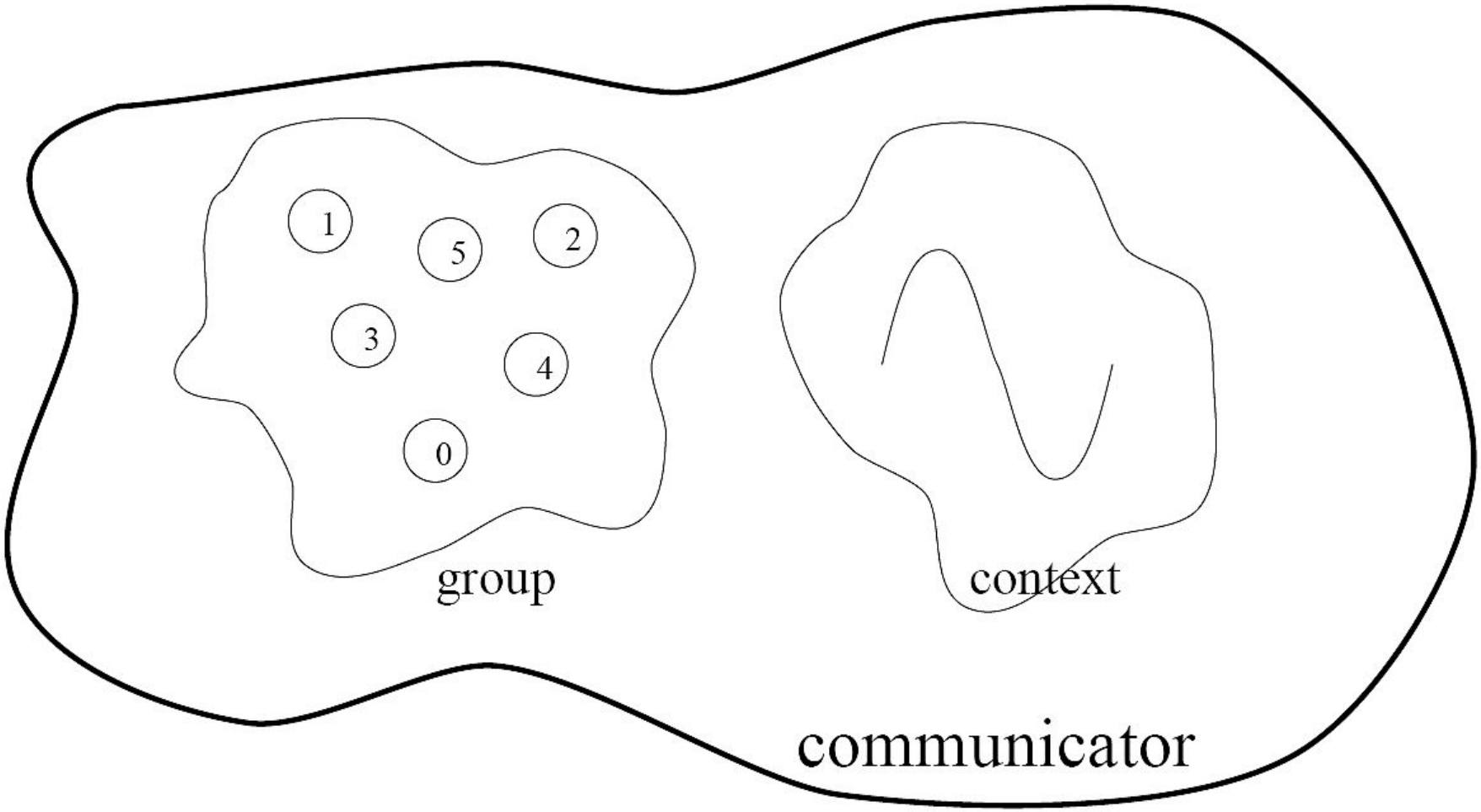
- Transforms incoming messages to target format, possibly using intermediate representation
- May provide *subject-based* routing capabilities
- Acts very much like an *application gateway*



What is MPI?

- **A message-passing library specifications:**
  - **Extended message-passing model**
  - **Not a language or compiler specification**
  - **Not a specific implementation or product**
- **For parallel computers, clusters, and heterogeneous networks.**
- **Communication modes: *standard*, *synchronous*, *buffered,* and *ready*.**
- **Designed to permit the development of parallel software libraries.**
- **Designed to provide access to advanced parallel hardware for**
  - **End users**
  - **Library writers**
  - **Tool developers**

Group and Context



Group and Context (cont.)

- **Are two important and indivisible concepts of MPI.**
- **Group: is the set of processes that communicate with one another.**
- **Context: it is somehow similar to the frequency in radio communications.**
- **Communicator: is the central object for communication in MPI. Each communicator is associated with a group and a context.**

Communication Modes

- **Based on the type of send:**
  - **Synchronous: Completes once the acknowledgement is received by the sender.**
  - **Buffered send: completes immediately, unless if an error occurs.**
  - **Standard send: completes once the message has been sent, which may or may not imply that the message has arrived at its destination.**
  - **Ready send: completes immediately, if the receiver is ready for the message it will get it, otherwise the message is dropped silently.**

Blocking vs. Non-Blocking

- **Blocking, means the program will not continue until the communication is completed.**

- **Non-Blocking, means the program will continue, without waiting for the communication to be completed.**

Features of MPI

- **General**
  - **Communications combine context and group for message security.**
  - **Thread safety can't be assumed for MPI programs.**

Why to use MPI?

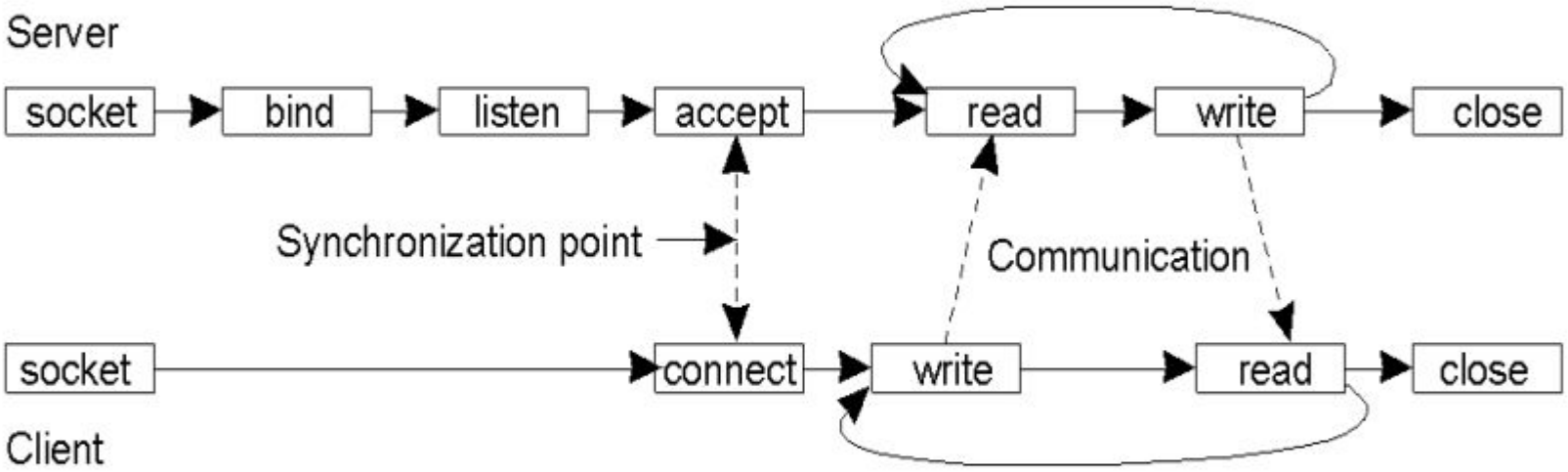
- **MPI provides a powerful, efficient, and portable way to express parallel programs.**
- **MPI was explicitly designed to enable libraries which may eliminate the need for many users to learn (much of) MPI.**
- **Portable !!!!!!!!!!!!!!!!!!!!!!!!!**
- **Good way to learn about subtle issues in parallel computing**

Berkeley Sockets

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

- A socket is a communication endpoint to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read.
- Socket primitives for TCP/IP.

Berkeley Sockets



- Connection-oriented communication pattern using sockets.