CAT3053/N
Distributed Computing

Communication in Distributed Systems

Objectives

- To identify different layers of communication from protocols to Application Programming Interfaces (APIs).
- To present the various ways and widely-used models of communication in distributed systems
- To specify different paradigms of communications
- To be able to program applications with distributed objects and Internet protocols

References

1. Distributed Systems: Principles and Paradigms by Tanenbaum (Chap 2, 3)
2. Distributed Systems: Concepts and Design by Coulouris (Chap 3, 4, 5, 6)
3. Distributed and Parallel Computing by El-Rewini and Lewis (10,11)

Overview

- Protocols
- Models
  - Remote Procedure Call
  - Remote Method Invocation
  - Message Passing Interface Standard (MPI)
  - Streams
- Topology of communication
  - Client server
  - Peer to peer
  - Group

What is MPI?

- A message passing library specification
  - Message-passing model
    - Programming model which assumes each processor has its own local memory. Data are transferred by means of message passing.
  - Not a compiler specification (i.e. not a language)
  - Not a specific product
- Designed for parallel computers, clusters, and heterogeneous networks

The MPI Process

- Development began in early 1992
- Open process/Broad participation
  - IBM,Intel, TMC, Meiko, Cray, Convex, Ncube
  - PVM, p4, Express, Linda, …
  - Laboratories, Universities, Government
- Final version of draft in May 1994
- Public (LAM/MPI, MPICH) and vendor implementations are now widely available
- *de facto* standard for message-passing-based application.
- ….Supersedes earlier libraries.

MPI-1 Services

- Supports point-to-point communication
  - communication modes - blocking and non-blocking (standard, buffered, synchronise and ready)
- collective communication
  - data movement (gather/scatter)
  - reduction/collective computation
- virtual topologies - conceptualise processes in an application-oriented topology (grids and graphs)
- derived datatype - user-defined datatype

MPI-2 Services

- Dynamic task creation
- Client/server functions
- One-sided communication
- Parallel I/O

Communicators

- The notion of context and group are combined in a single communicator (object)
- All MPI communication calls take a communicator handle as a parameter, which is effectively the context in which the communication will take place
- E.g. MPI_INIT defines a communicator called MPI_COMM_WORLD for each process that calls it (default)

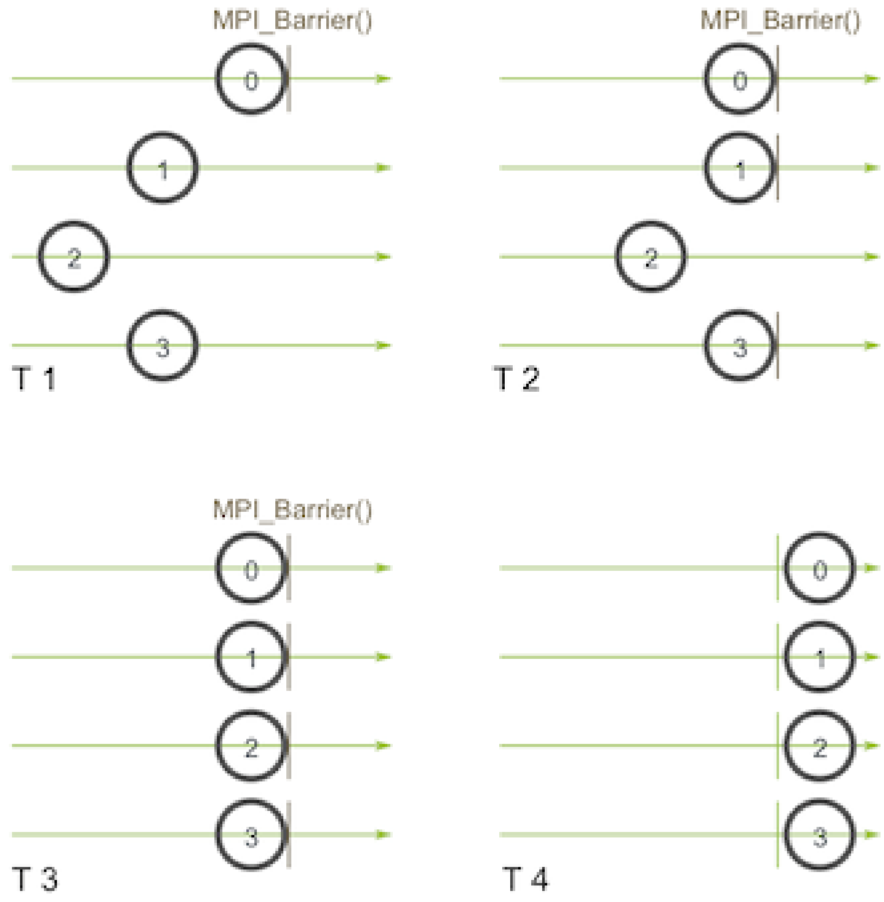
Communicator (cont.)

- Every communicator contains a group which is a list of processes
- The processes are ordered and numbered consecutively from 0.
- The number of each process is known as its rank
  - The rank identifies each process within the communicator
- The group of MPI_COMM_WORLD is the set of all MPI processes

Types of Collective Transfers

- Barrier
  - Synchronizes processors
  - No data is exchanged but the barrier blocks until all processes have called the barrier routine
- Broadcast (sometimes multicast)
  - A broadcast is a one-to-many communication
  - One processor sends one message to several destinations
- Reduction
  - Often useful in a many-to-one communication

Barrier



https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/

What's in a Message?

- An MPI message is an array of elements of a particular MPI datatype
- All MPI messages are typed
  - The type of the contents must be specified in both the send and the receive

Basic C Datatypes in MPI

- *advanced* datatype (derived datatype) - user construct own datatypes using MPI routines and data can be non-contiguous

MPI Datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_INT	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI Basic

- comprises 129 functions
- a complete message passing programs can be written using 6 functions

MPI_INIT	**Initiate an MPI computation.**
MPI_FINALIZE	**Terminate a computation.**
MPI_COMM_SIZE	**Determine num. of processes**
MPI_COMM_RANK	**Determine my process id.**
MPI_SEND	**Send a message.**
MPI_RECV	**Receive a message.**

Skeleton MPI Program

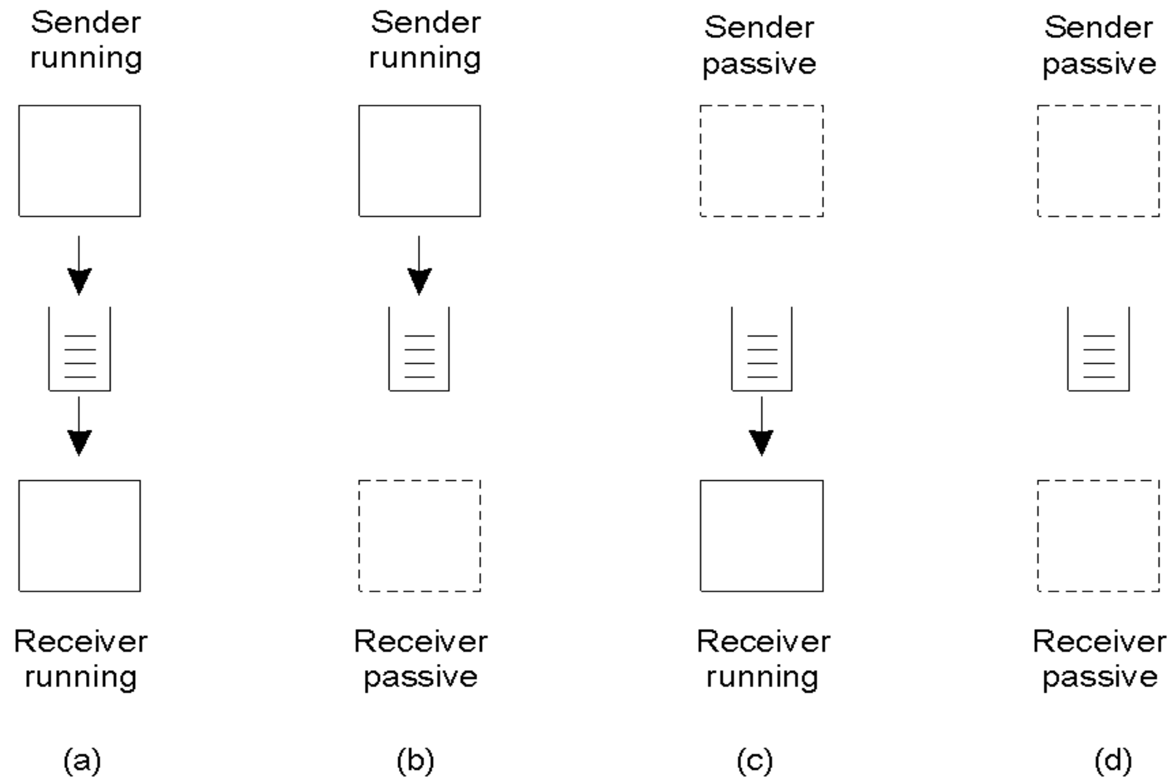
```
program main
begin
    MPI_INIT ( )
    MPI_COMM_SIZE (MPI_COMM_WORLD, count)
    MPI_COMM_RANK (MPI_COMM_WORLD, myid)

    print ("I am", myid, "of", count)

    MPI_FINALIZE ( )
end
```

Message-Queuing Model (1)

- Four combinations for loosely-coupled communications using queues.

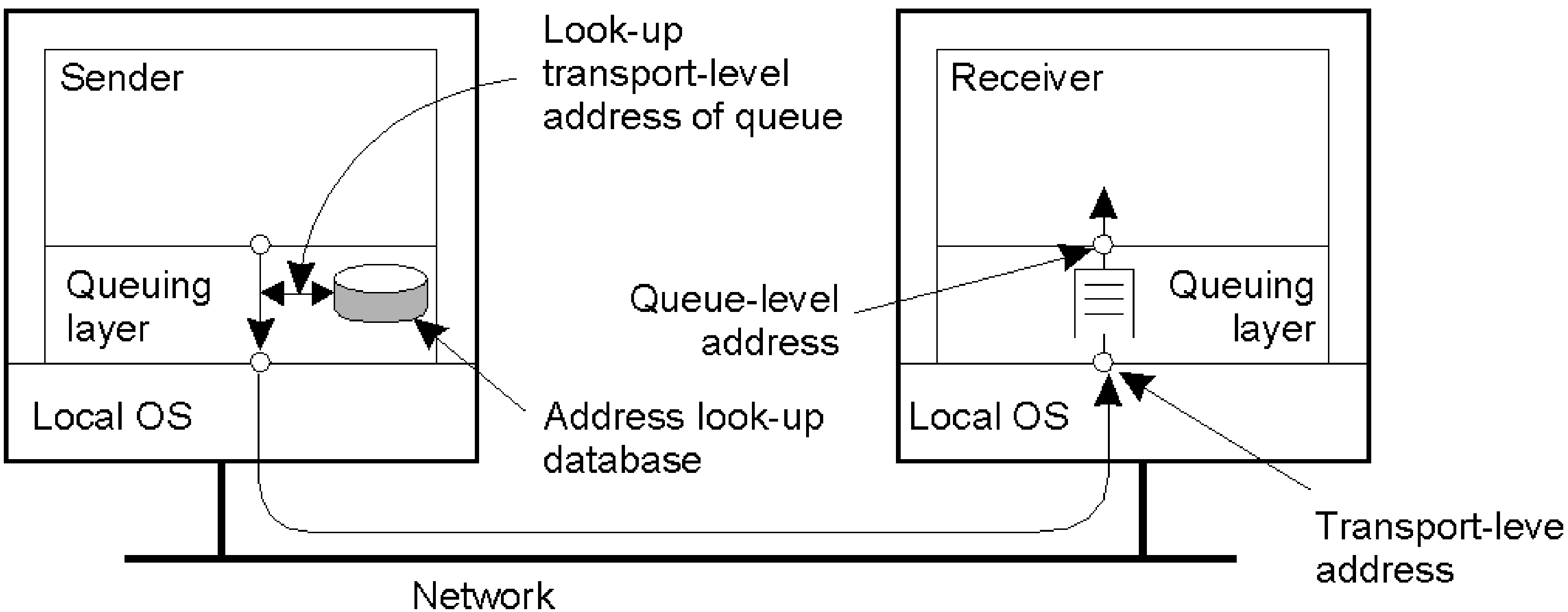


Message-Queuing Model (2)

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

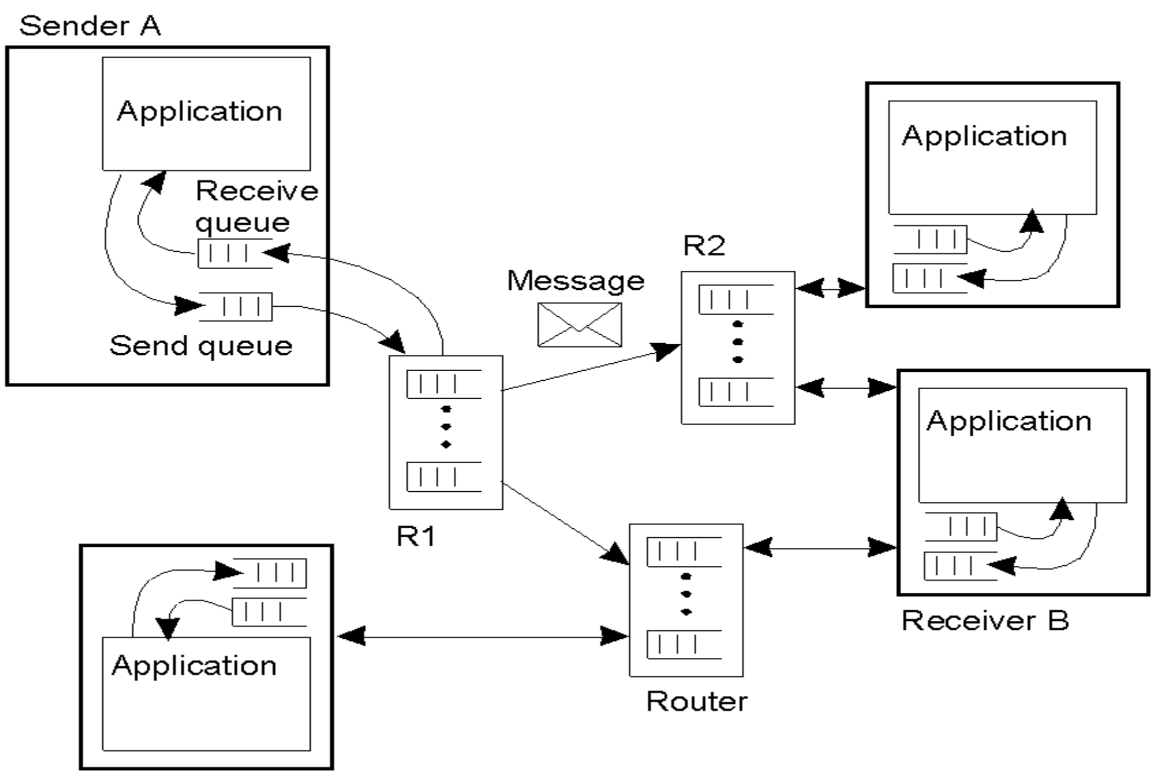
Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

General Architecture of a Message-Queuing System (1)

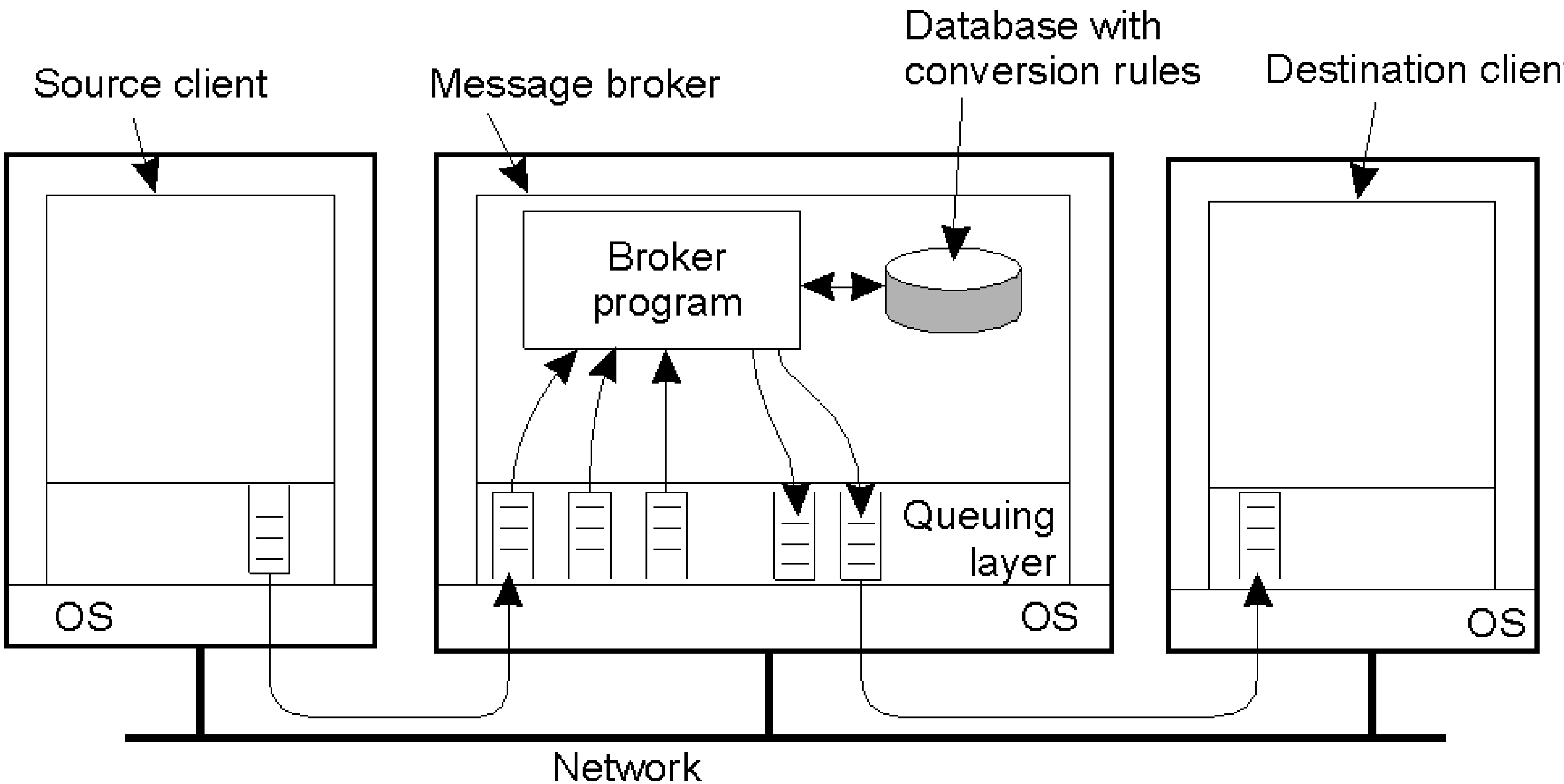


General Architecture of a Message-Queuing System (2)

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

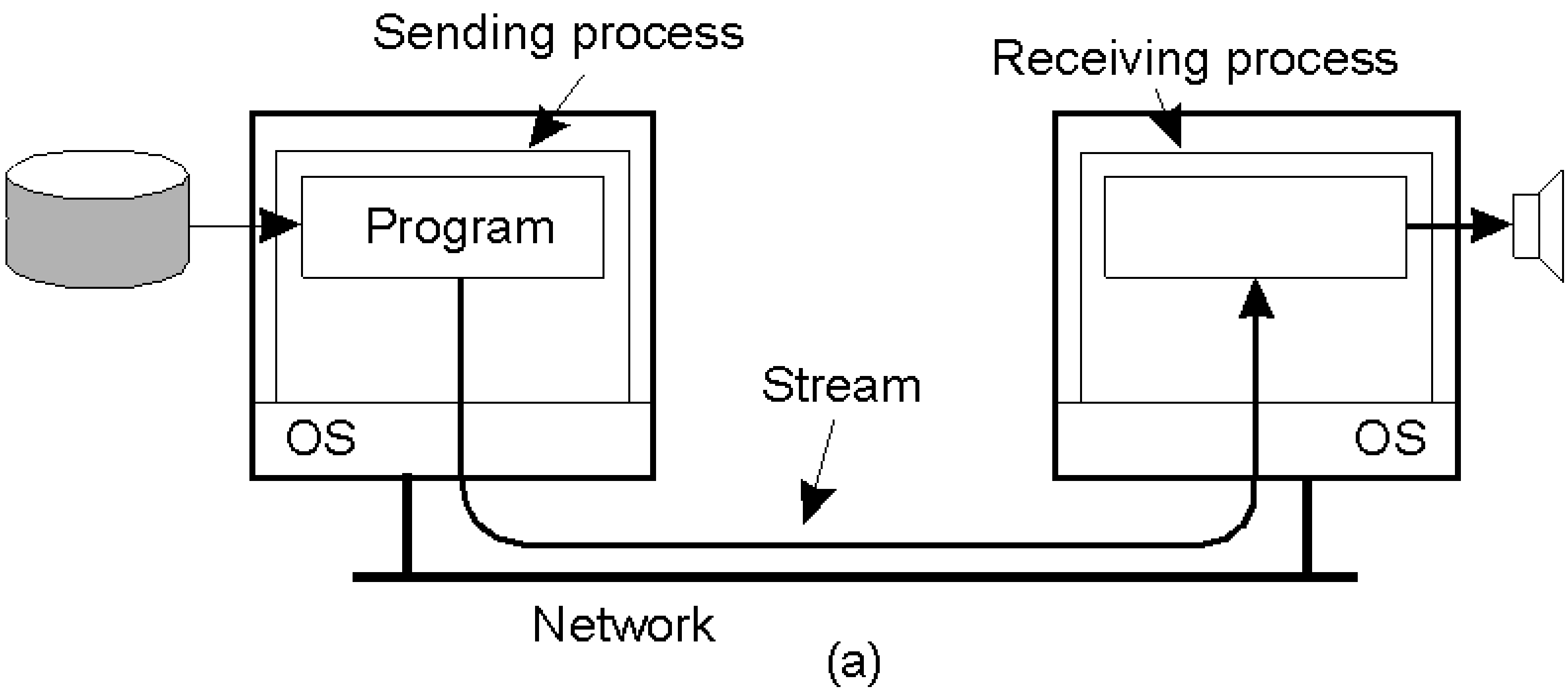


Message Brokers



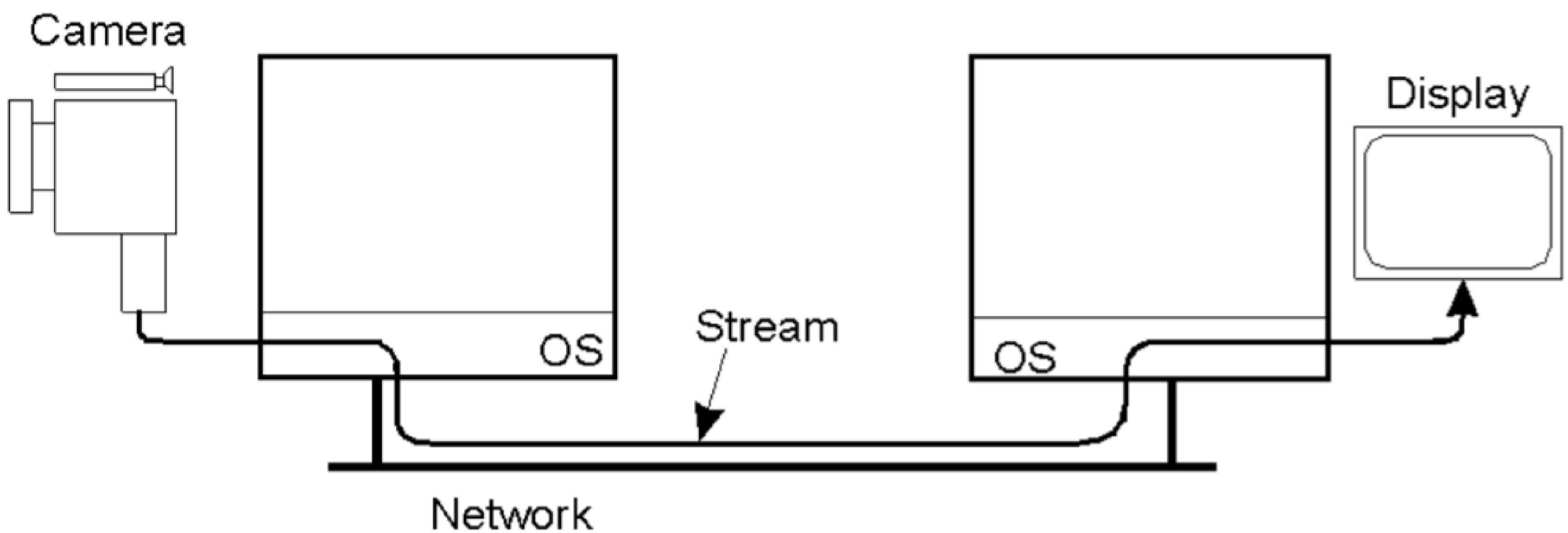
Data Stream (1)

Setting up a stream between two processes across a network.



Data Stream (2)

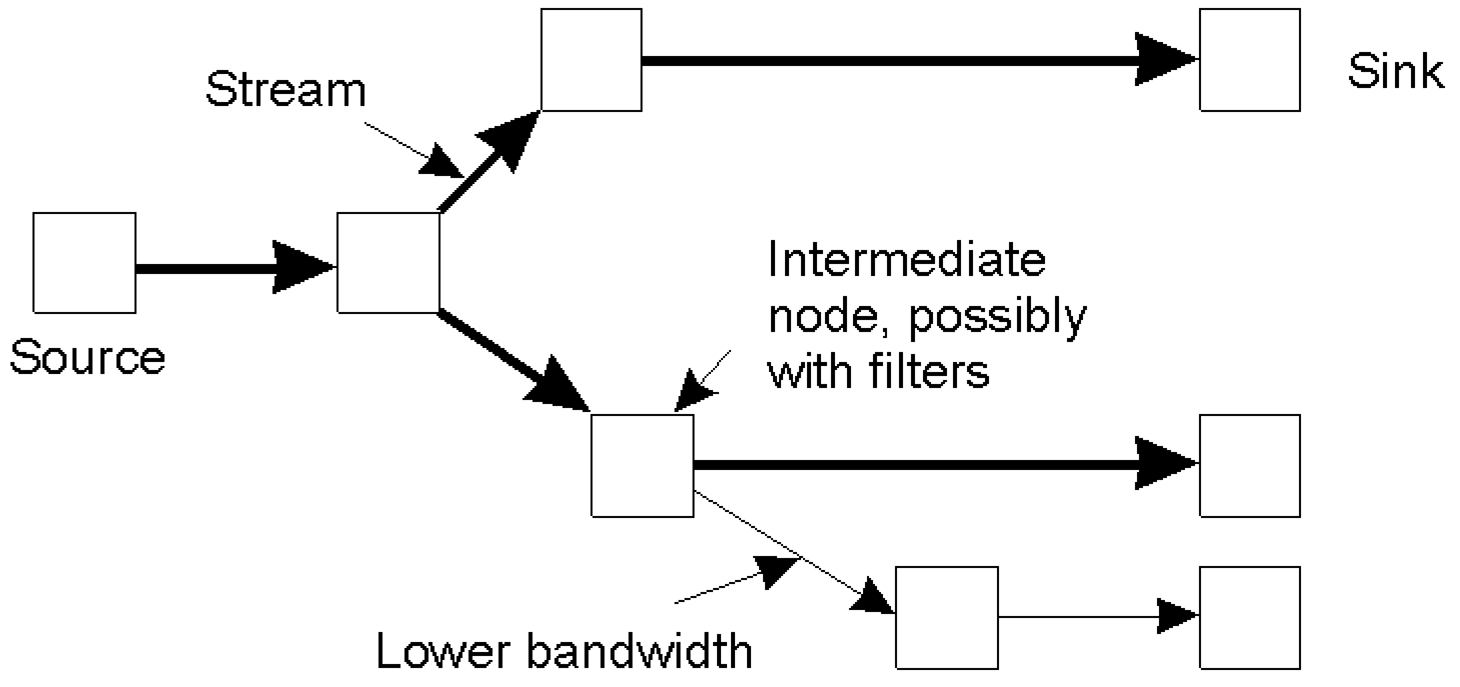
- Setting up a stream directly between two devices.



(b)

Data Stream (3)

- An example of multicasting a stream to several receivers.



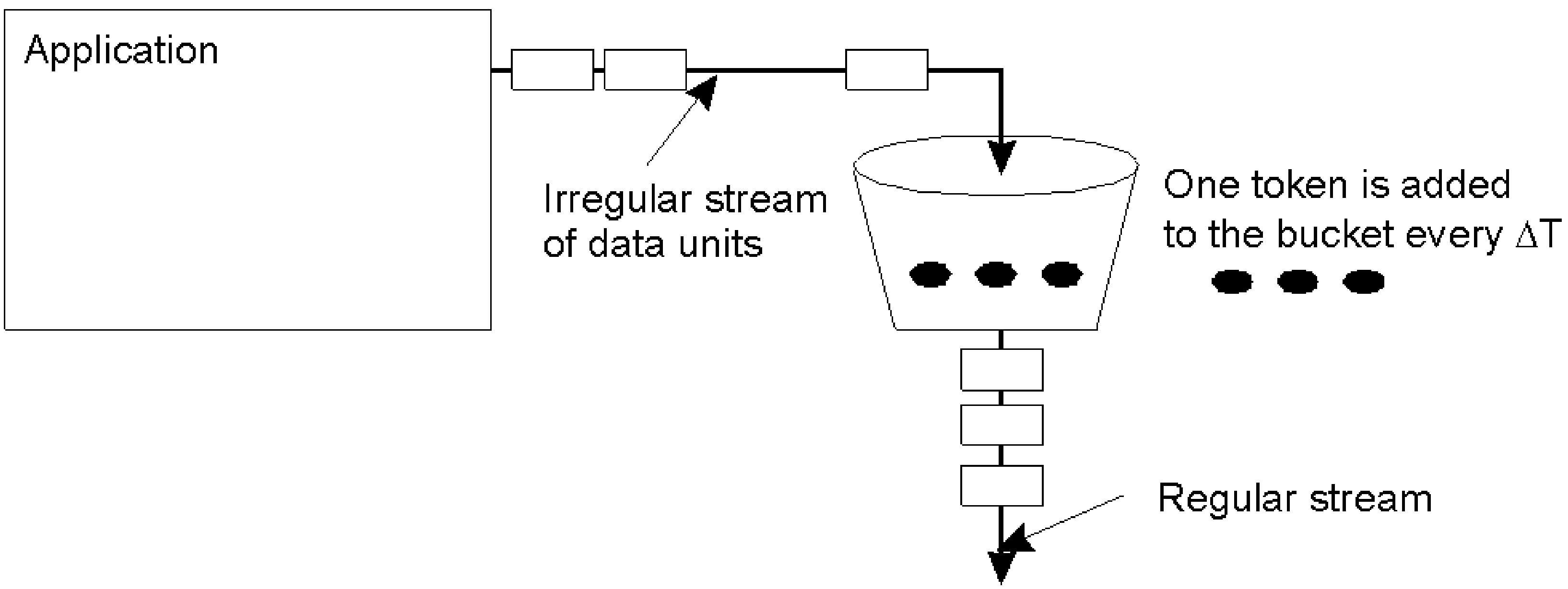
Specifying QoS (1)

- A flow specification.

Characteristics of the Input	Service Required
maximum data unit size (bytes) Token bucket rate (bytes/sec) Toke bucket size (bytes) Maximum transmission rate (bytes/sec)	Loss sensitivity (bytes) Loss interval (μsec) Burst loss sensitivity (data units) Minimum delay noticed (μsec) Maximum delay variation (μsec) Quality of guarantee

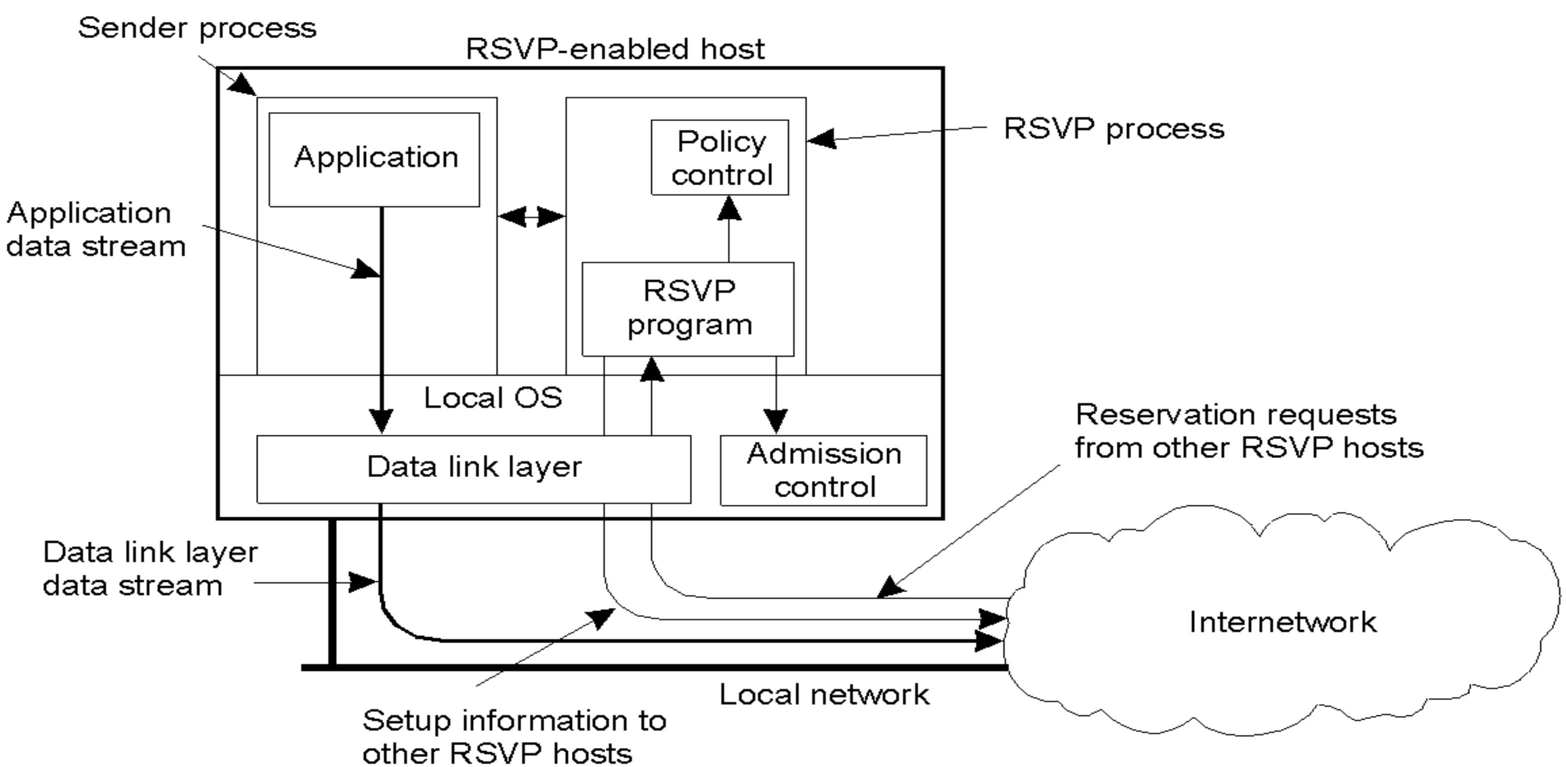
Specifying QoS (2)

- The principle of a token bucket algorithm.



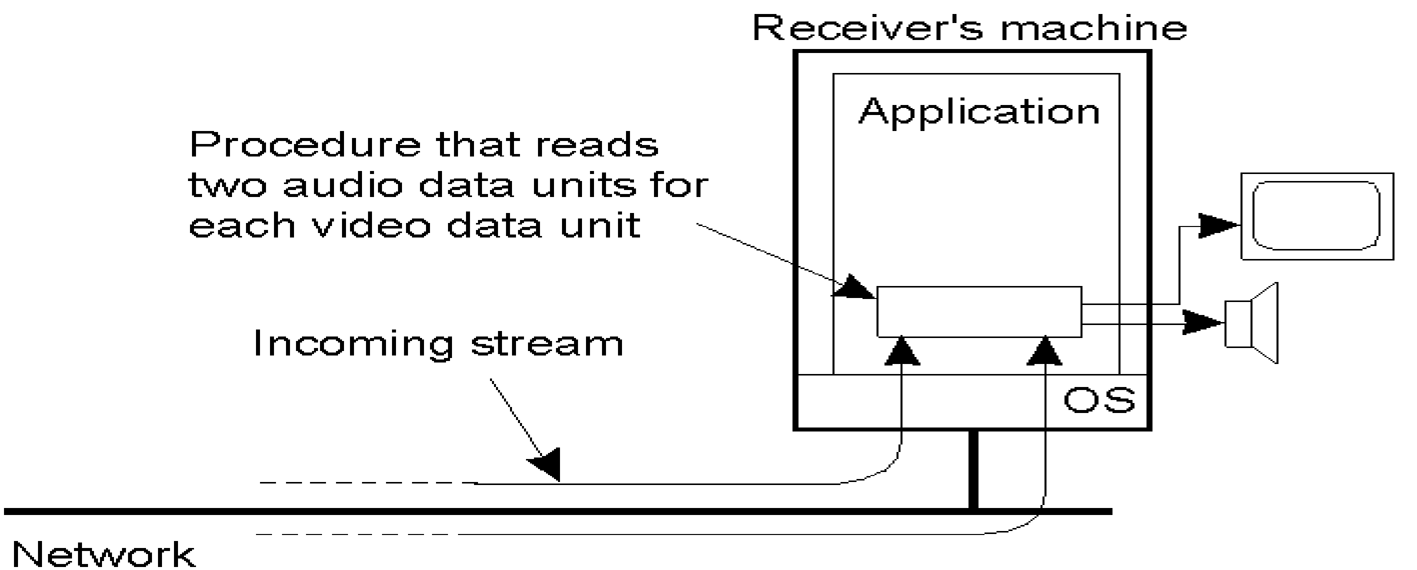
Setting Up a Stream

- The basic organization of RSVP for resource reservation in a distributed system.



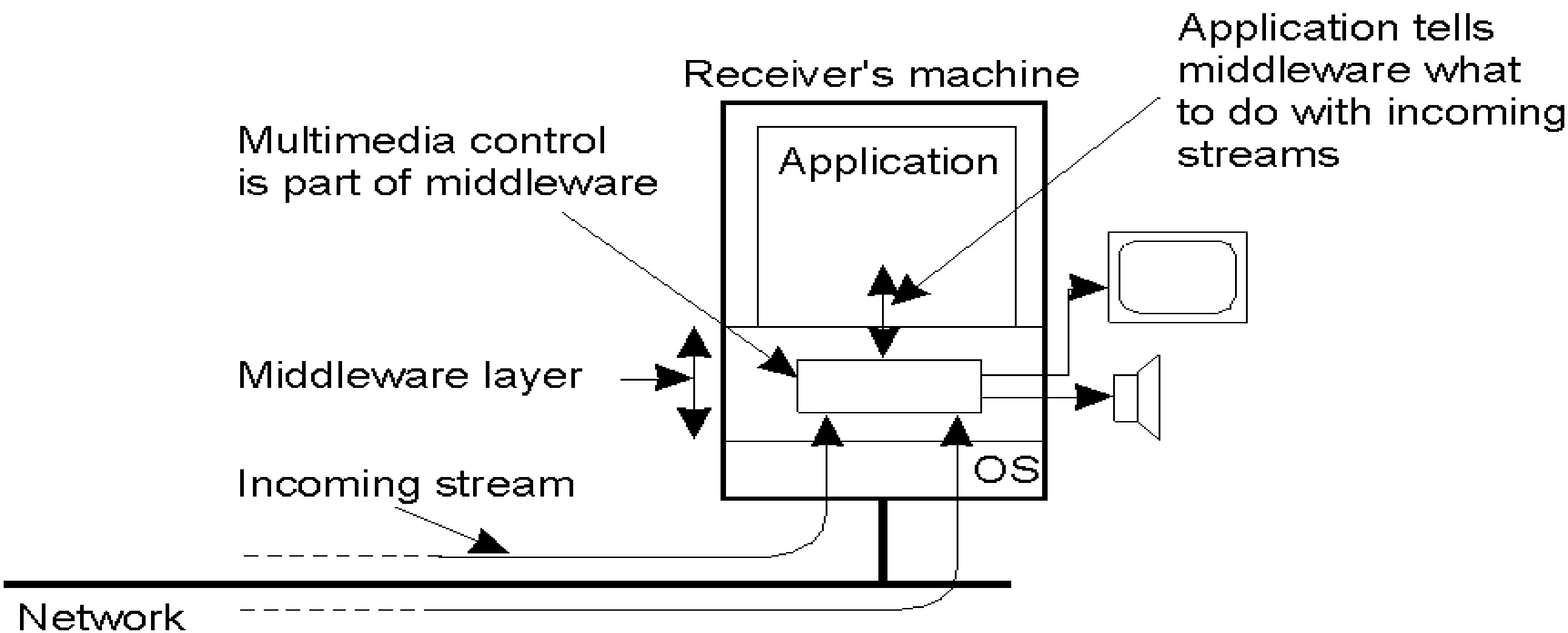
Synchronization Mechanisms (1)

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



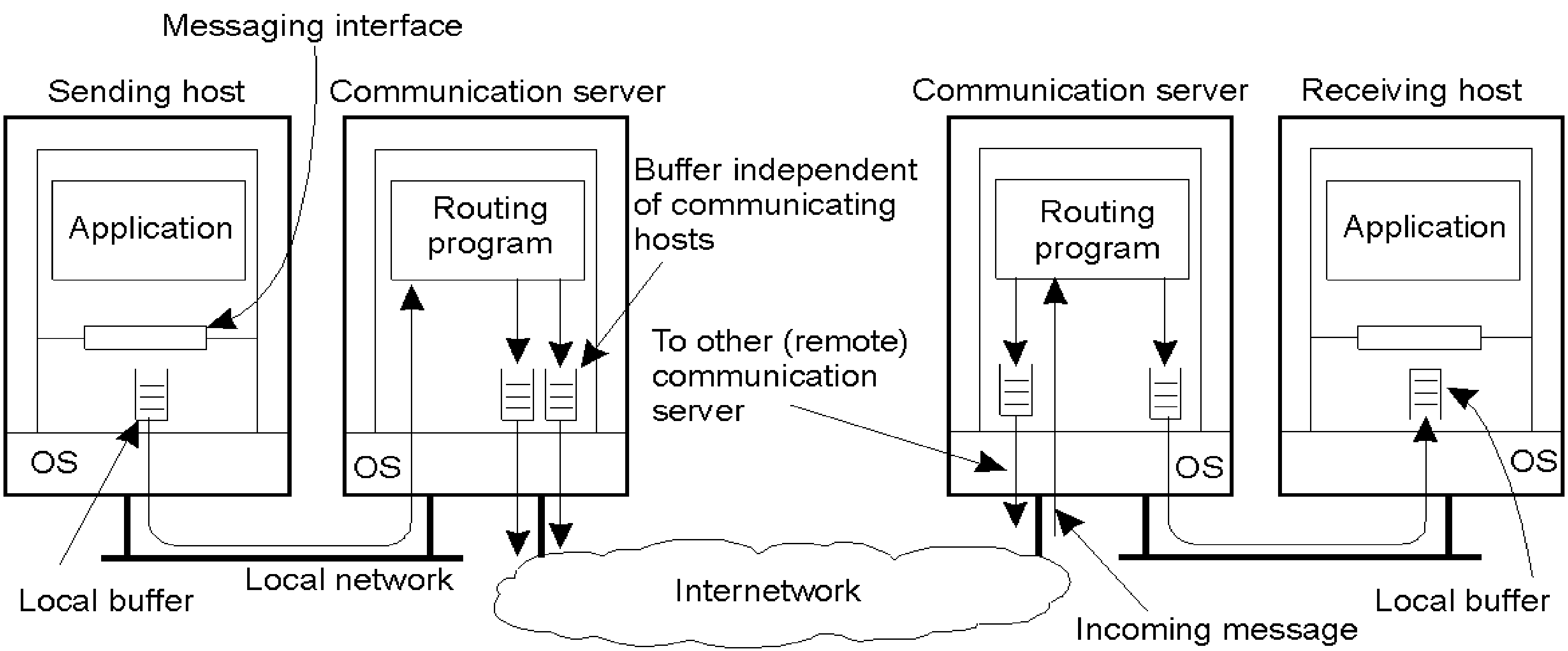
Synchronization Mechanisms (2)

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



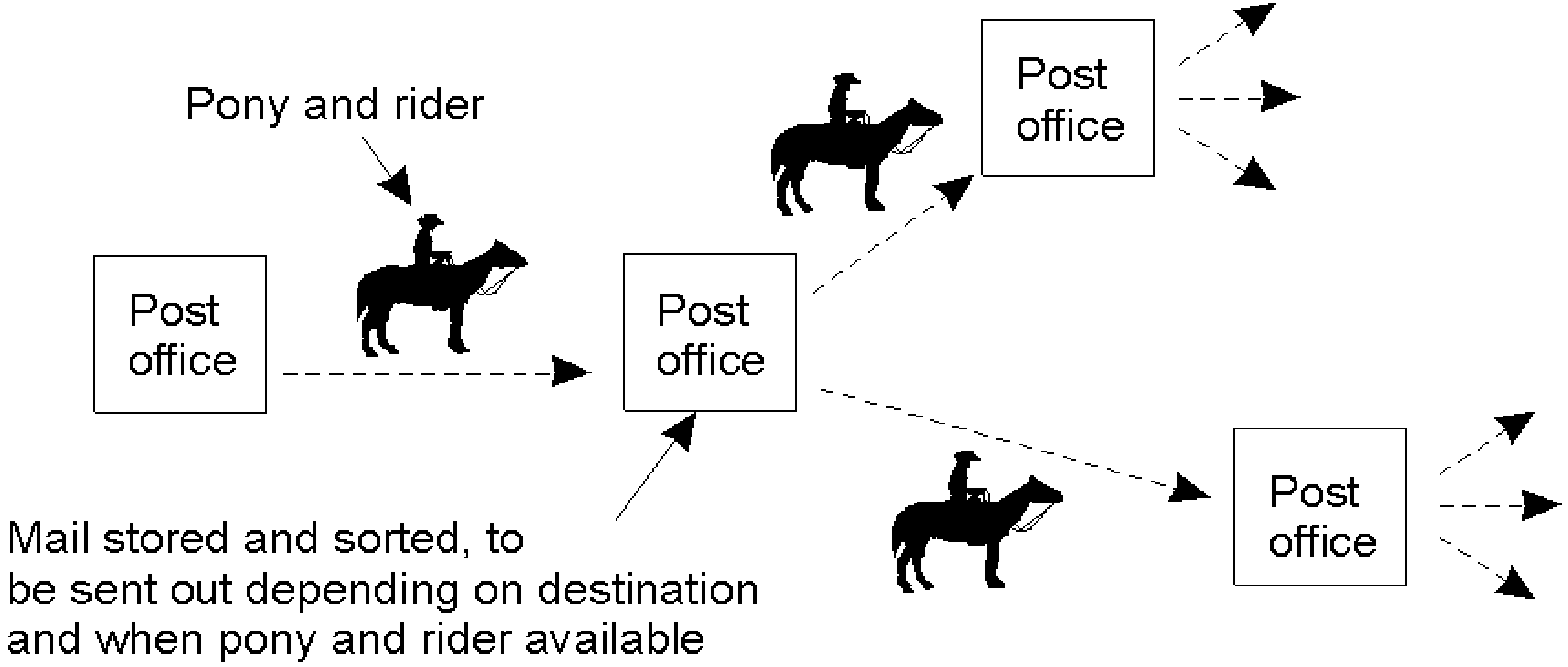
Persistence and Synchronicity in Communication (1)

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



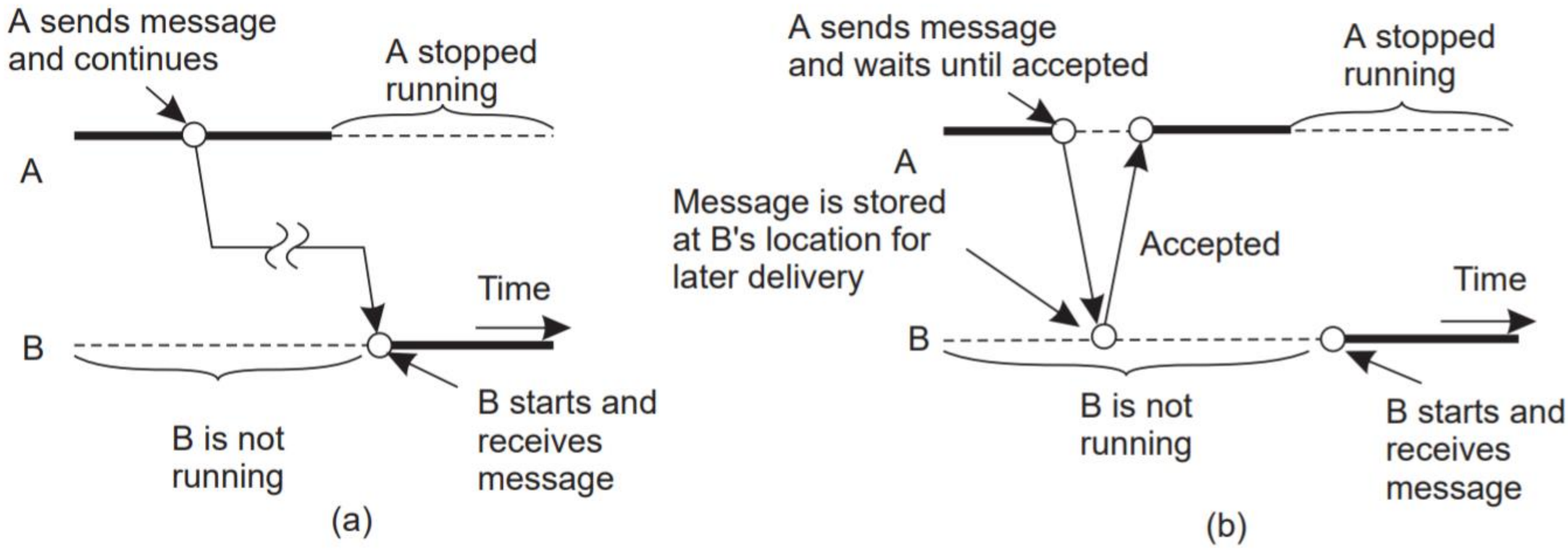
Persistence and Synchronicity in Communication (2)

- Persistent communication of letters back in the days of the Pony Express.

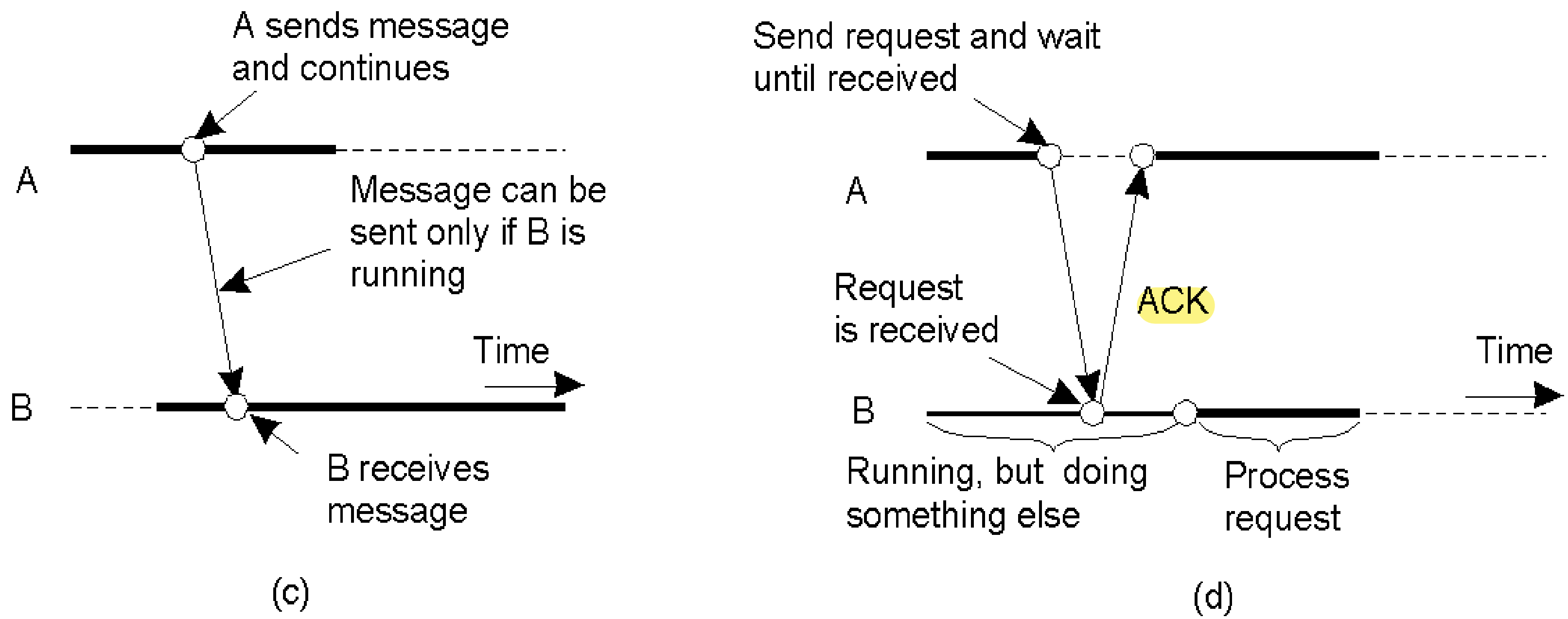


Persistence and Synchronicity in Communication (3)

a) Persistent asynchronous communication
b) Persistent synchronous communication



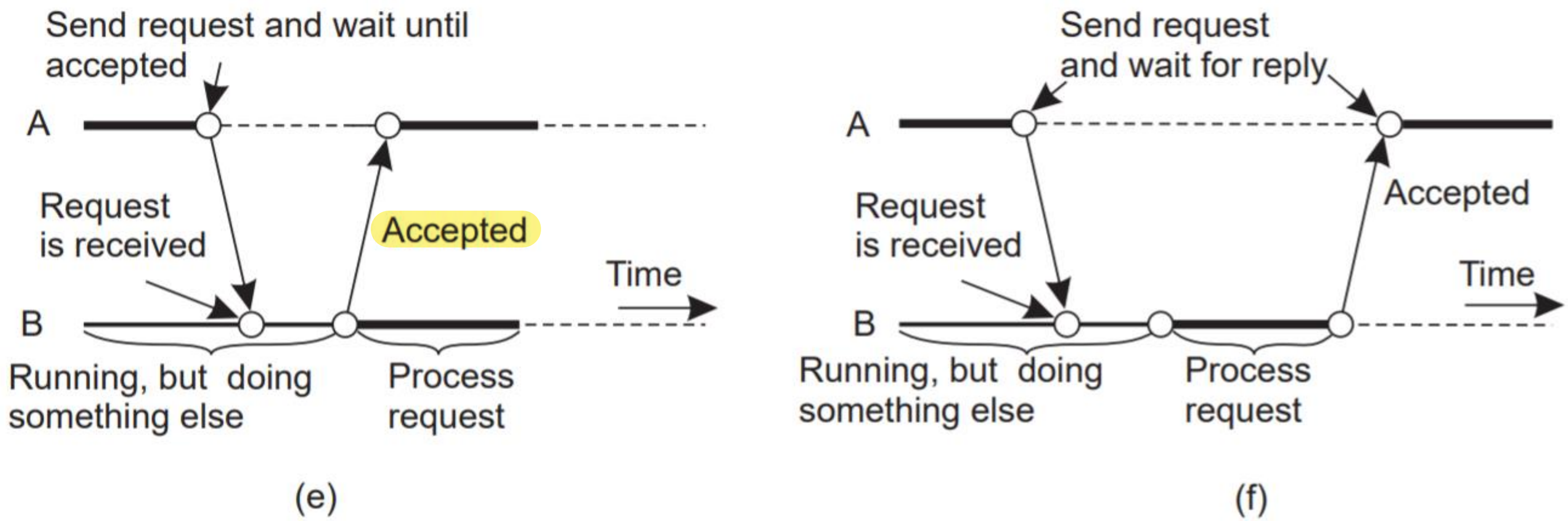
Persistence and Synchronicity in Communication (4)



c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication (5)



e) Delivery-based transient synchronous communication at message delivery

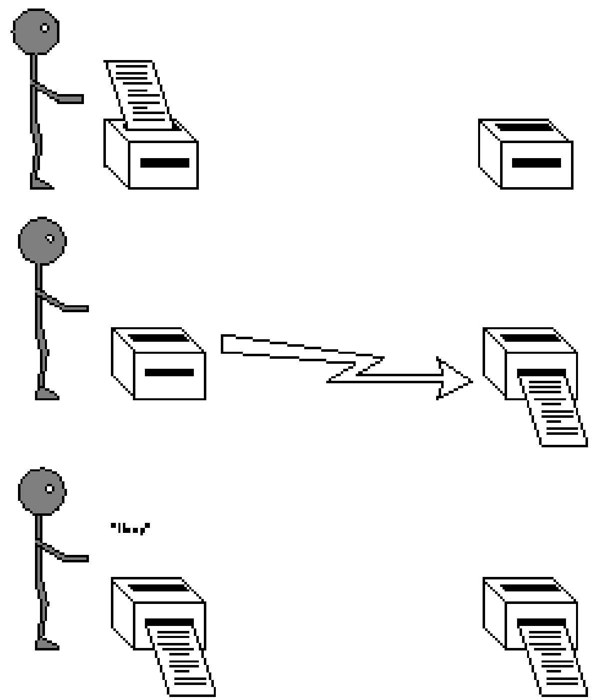
f) Response-based transient synchronous communication

Point-to-point Communication

- Always involves exactly two processes
- Example
  - *MPI_send (buf, count, datatype, dest, tag, comm)*
    - *dest* is identified by its rank within the communicator
    - *comm identifies a group of processes and a communication context*
  - *MPI_Recv (buf, count, datatype, source, tag, comm, status)*
- Several variations on how the sending of a message can interact with a program

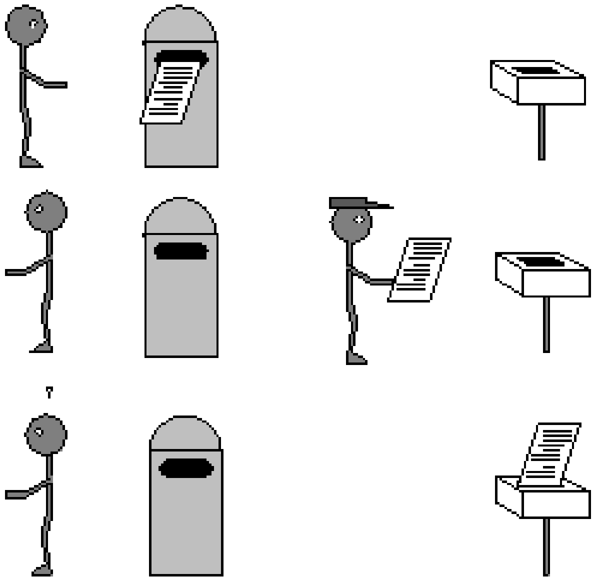
Synchronous

- A synchronous communication does not complete until the message has been received
  - A FAX or registered mail



Asynchronous

- An asynchronous communication completes as soon as the message is on the way.
  - A post card or email



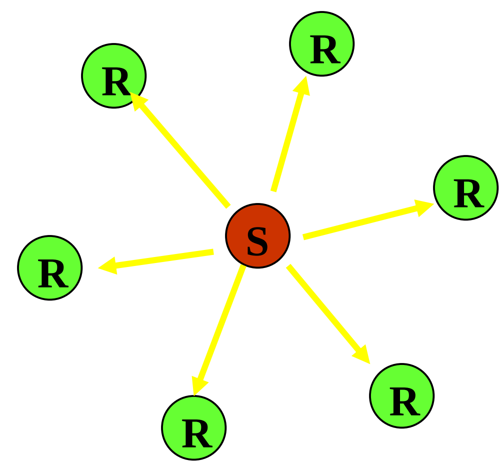
Blocking and Non-blocking

- Blocking operations only return when the operation has been completed
  - Normal FAX machines
- Non-blocking operations return right away and allow the program to do other work
  - Receiving a FAX

Group Communication

- RPC – communication involves two processes (point-to-point)
- Can RPC handle one-to-many?
  - E.g. a client has to send message to all servers.

Point-to-point



One-to-many

Group Communications

- Characteristics:
  - Dynamics – create/destroy group, process join or leave and can be member to more than one group
- Multicast – sends a single message from one process to each of the members of a group of process.
- Broadcast – message delivered to all machines

Collective Communications

- Point-to-point communications involve pairs of processes.
- Many message passing systems provide operations which allow larger numbers of processes to participate

Thank you