HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Interprocess Communication

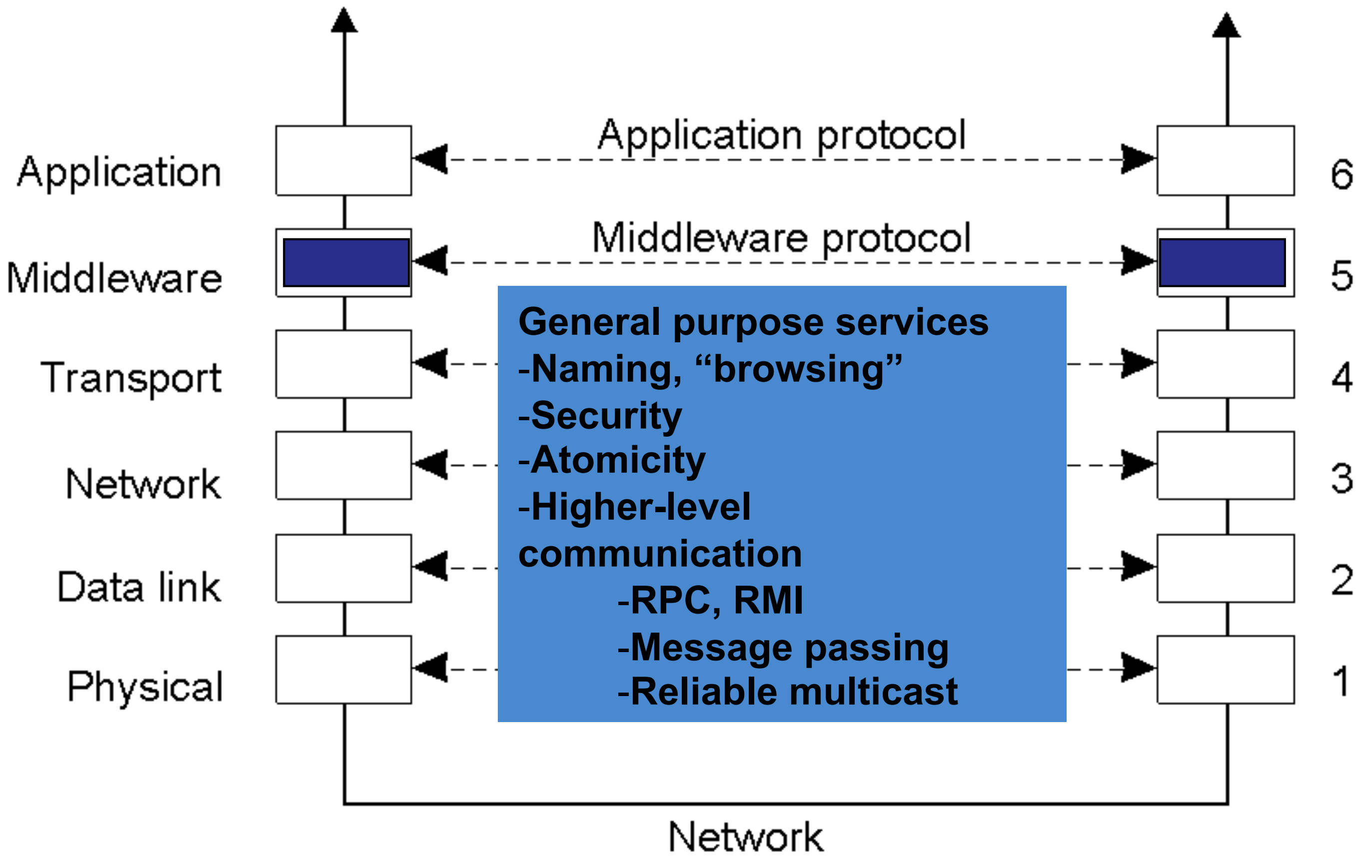
Tanenbaum, van Steen: Ch4, Ch 10
CoDoKi: Ch2, Ch3, Ch5

Fall 2010
Jussi Kangasharju

Chapter Outline

- Overview of interprocess communication
- Remote invocations (RPC etc.)
- Message passing
- Streams
- Publish/subscribe
- Multicast

Middleware Protocols



An adapted reference model for networked communication.

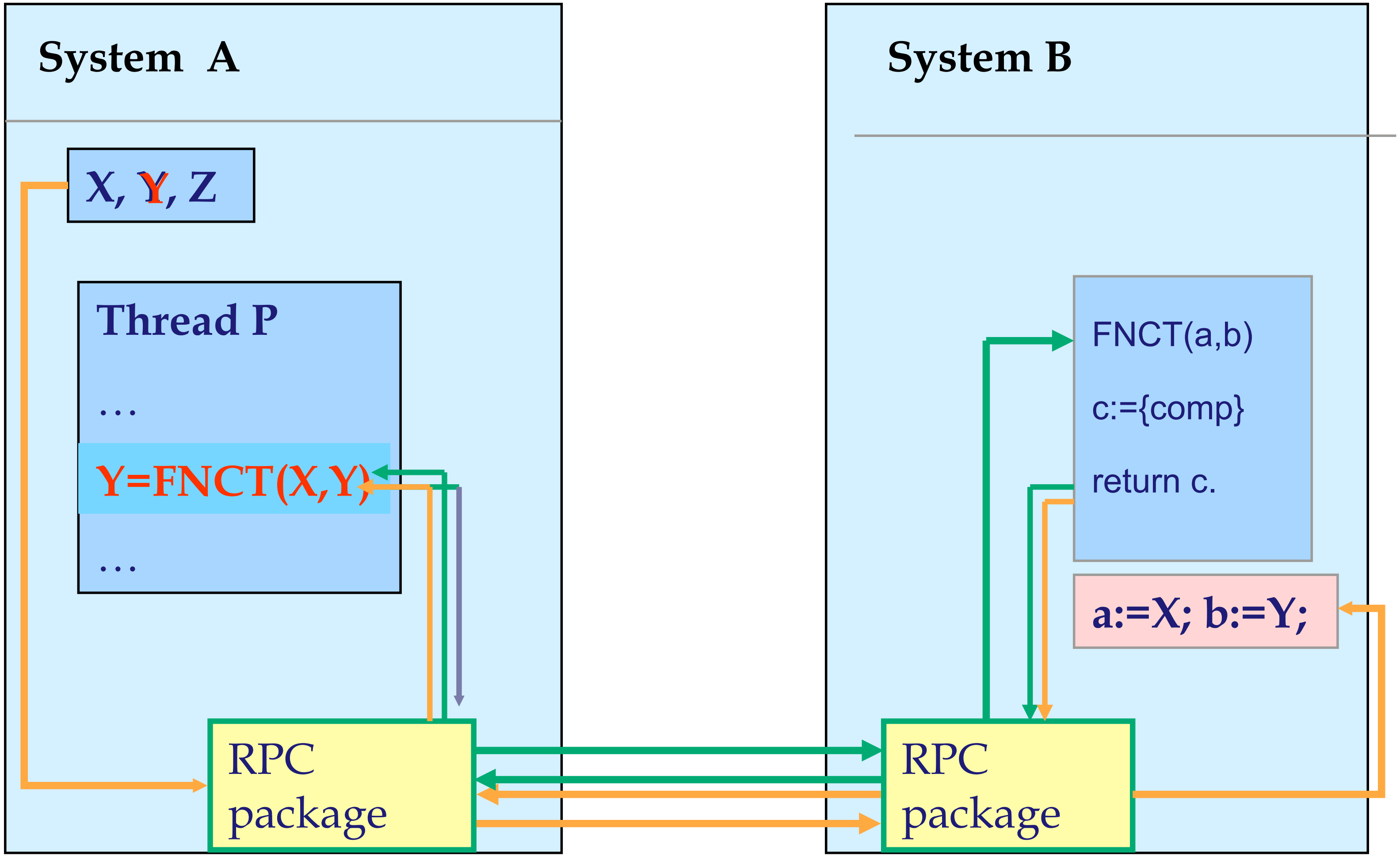
Remote Procedure Calls

- Basic idea:
  - “passive” routines
  - Available for remote clients
  - Executed by a local worker process, invoked by local infrastructure
- See examples in book

RPC goals

- Achieve access transparent procedure call
- Cannot fully imitate
  - naming, failures, performance
  - global variables, context dependent variables, pointers
  - Call-by-reference vs. call-by-value
- Call semantics
  - Maybe, at-least-once, at-most-once
  - Exception delivery
- Can be enhanced with other properties
  - Asynchronous RPC
  - Multicast, broadcast
  - Location transparency, migration transparency, …
  - Concurrent processing

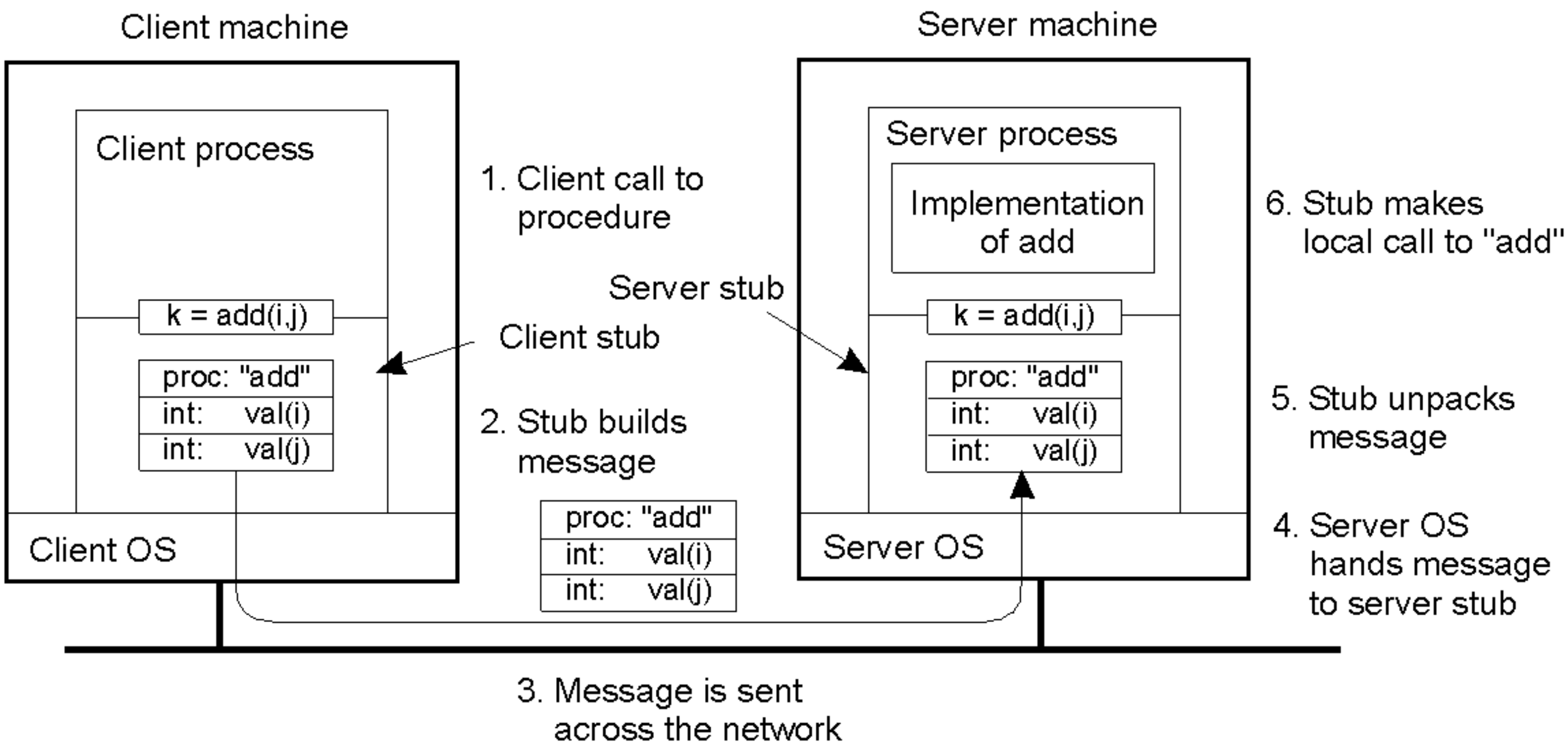
RPC: a Schematic View



Implementation of RPC

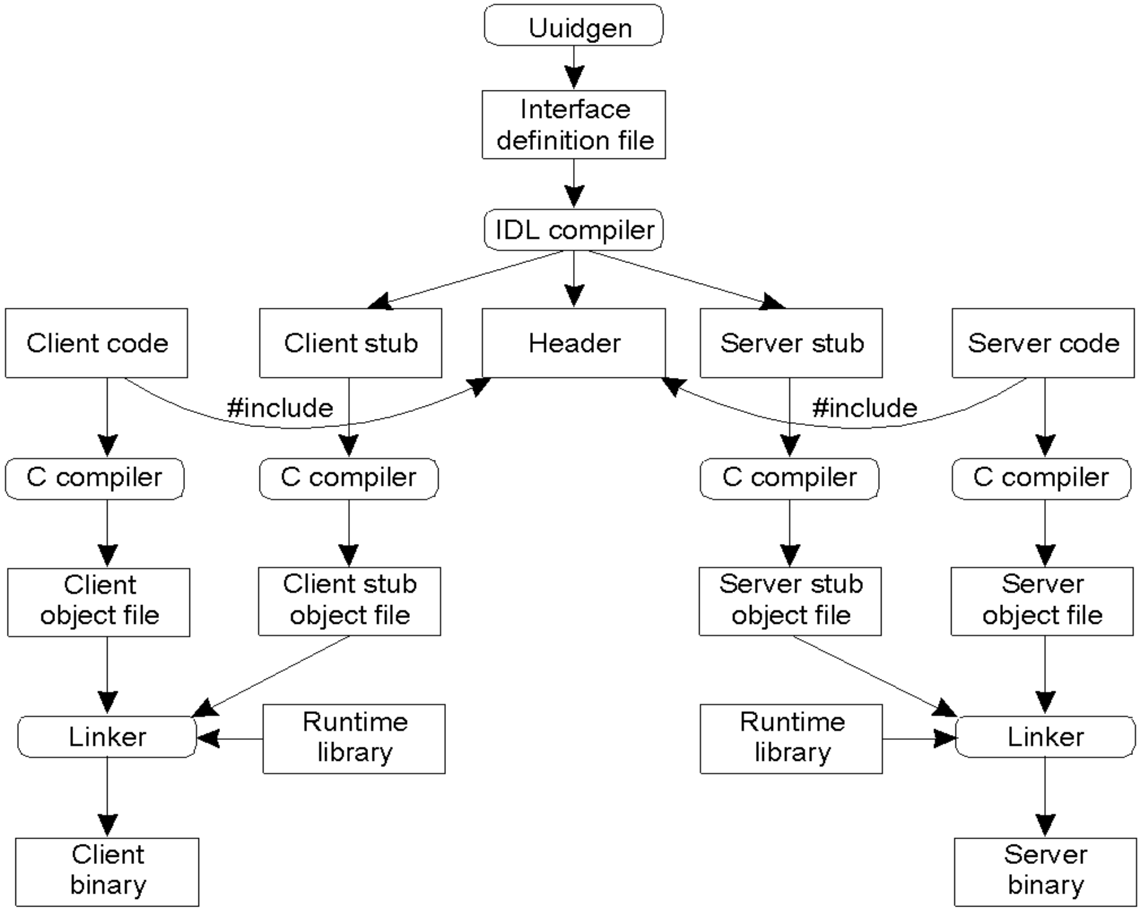
- RPC components:
  - RPC Service (two stubs)
    - interpretation of the service interface
    - packing of parameters for transportation
  - Transportation service: node to node
    - responsible for message passing
    - part of the operating system
- Name service: look up, binding
  - name of procedure, interface definition

Passing Value Parameters



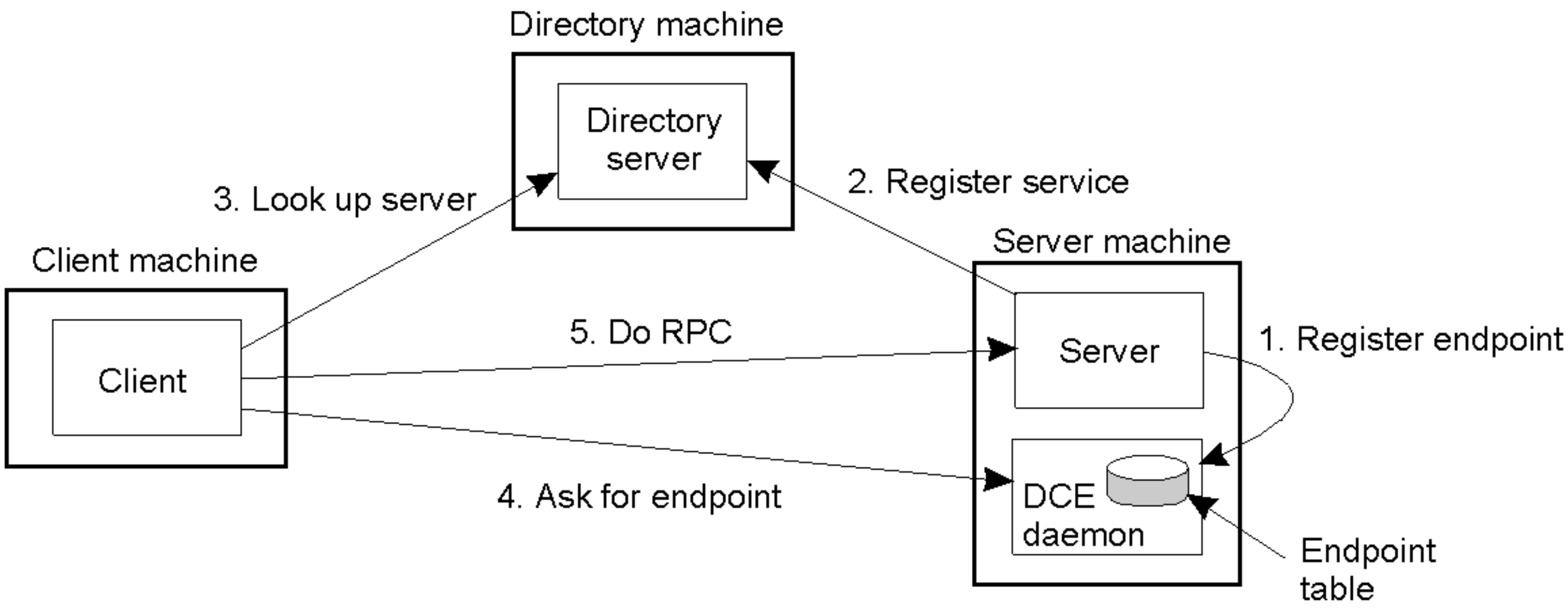
Steps involved in doing remote computation through RPC

Writing a Client and a Server



The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server



Client-to-server binding in DCE.

Implementation of RPC

- Server: who will execute the procedure?
- One server process
  - infinite loop, waiting in “receive”
  - call arrives : the process starts to execute
  - one call at a time, no mutual exclusion problems
- A process is created to execute the procedure
  - parallelism possible
  - overhead
  - mutual exclusion problems to be solved
- One process, a set of thread skeletons:
  - one thread allocated for each call

Distributed Objects

- Remote Method Invocation ~ RPC
- A distributed interface
  - binding: download the interface to the client => proxy
  - “server stub” ~ skeleton
- The object
  - resides on a single machine (possible distribution: hidden)
  - if needed: “object look” through an adapter
  - an object may be persistent or transient
- Object references:
  - typically: system-wide
  - binding: implicit or explicit resolving of an object reference
- Binding and invocation
- Examples: CORBA, DCOM (Ch. 10)

Distributed Objects

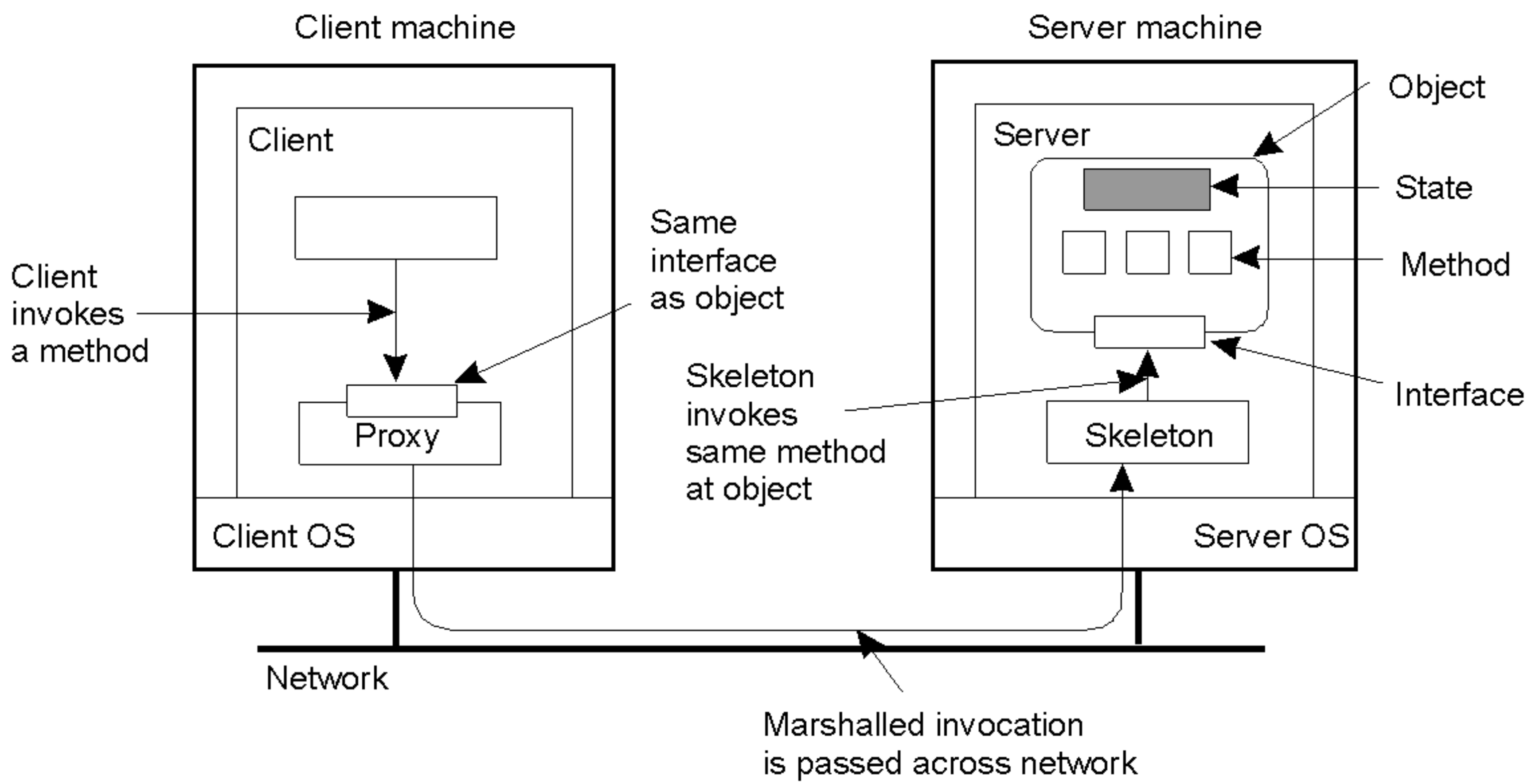


Fig. 2-16. Common organization of a remote object with client-side proxy.

Object Adapter

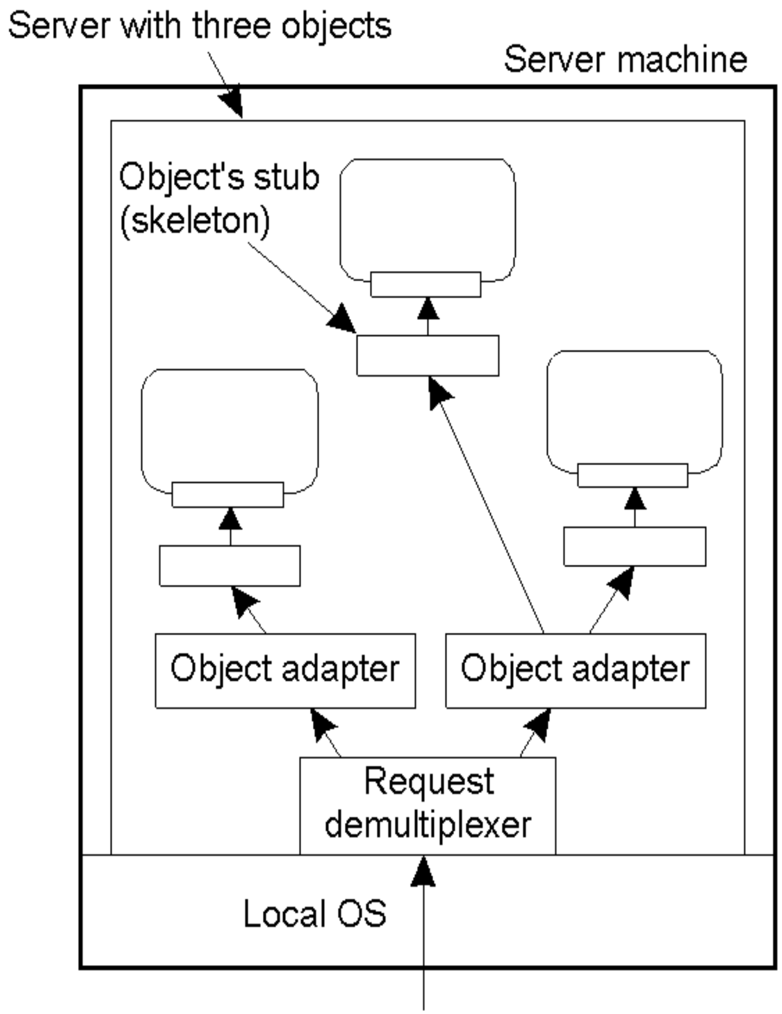


Fig. 3-8.

Organization of an object server supporting different activation policies.

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)

Fig. 2-17.

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

Parameter Passing

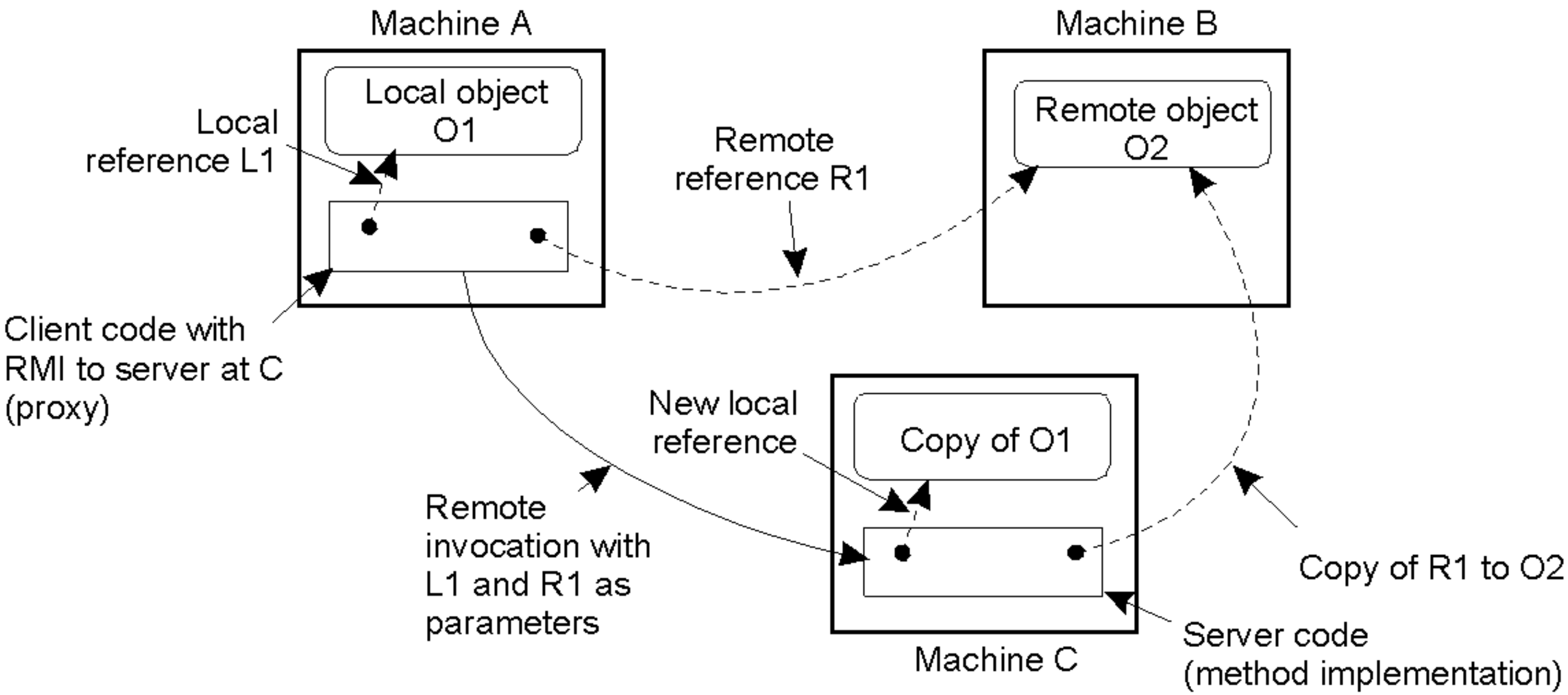


Fig. 2-18.The situation when passing an object by reference or by value.

Copying must not be hidden! ***Why?***

Design Issues

- Language independent interface definition
- Exception handling
- Delivery guarantees
  - RPC / RMI semantics
  - maybe
  - at-least-once
  - at-most-once
  - (un-achievable: exactly-once)
- Transparency (algorithmic vs. behavioral)

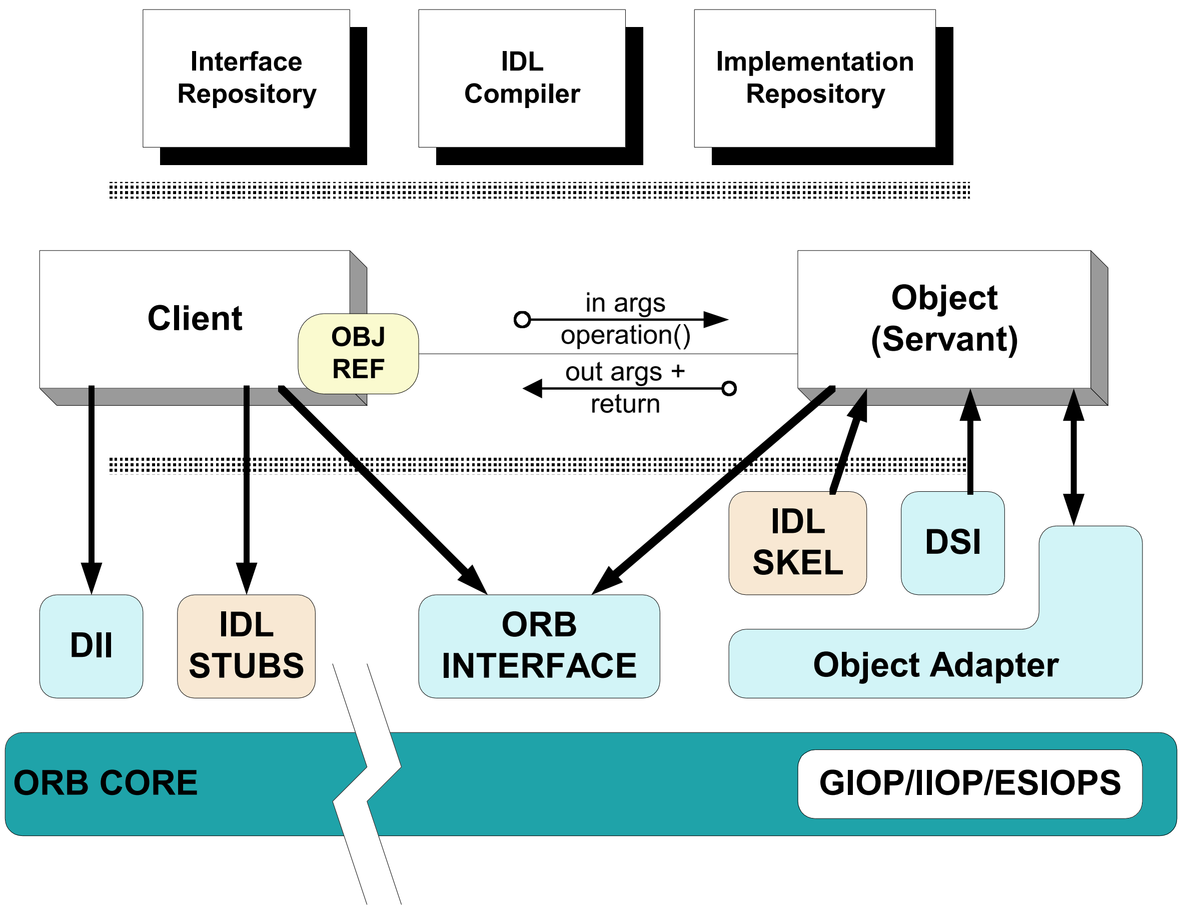
RPC: Types of failures

- Client unable to locate server
- Request message lost
  - retransmit a fixed number of times
- Server crashes after receiving a request or reply message lost (cannot be told apart!)
  - Client resubmits request, server chooses:
    - Re-execute procedure: service should be idempotent
    - Filter duplicates: server should hold on to results until acknowledged
- Client crashes after sending a request
  - Orphan detection: reincarnations, expirations
- Reporting failures breaks transparency

Fault tolerance measures

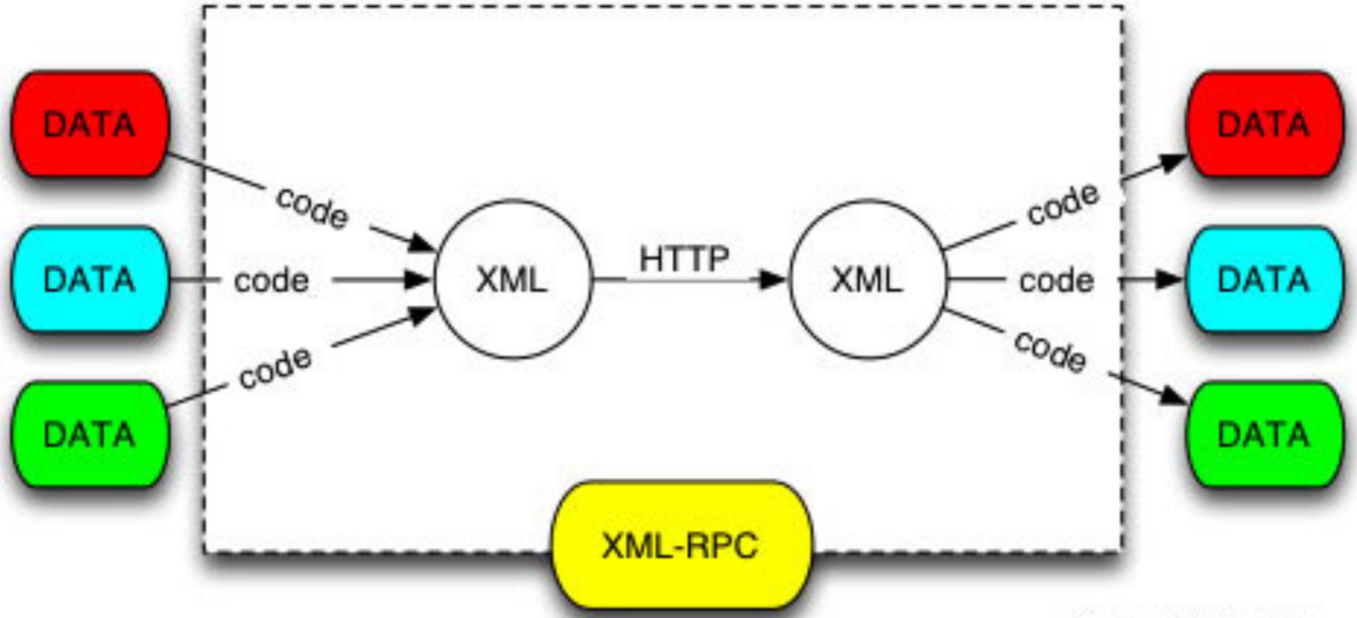
Retransmit request	Duplicate filtering	Re-execute/ retransmit	invocation semantics
no	N/A	N/A	maybe
yes	no	re-execute	at-least-once
yes	yes	retransmit reply	at-most-once

CORBA



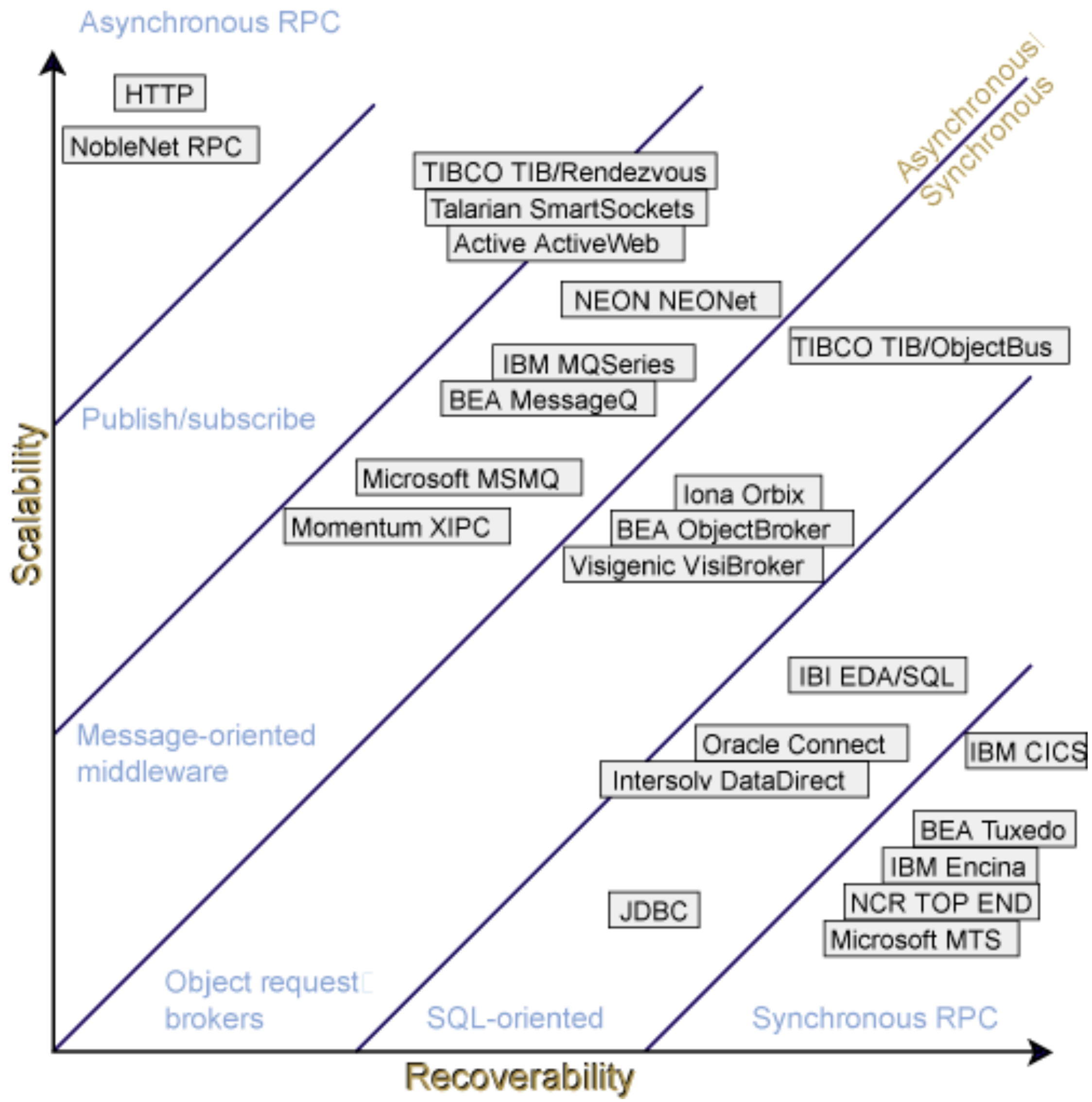
- CORBA shields applications from heterogeneous platform *dependencies*
  - *e.g.*, languages, operating systems, networking protocols, hardware

XML RPC

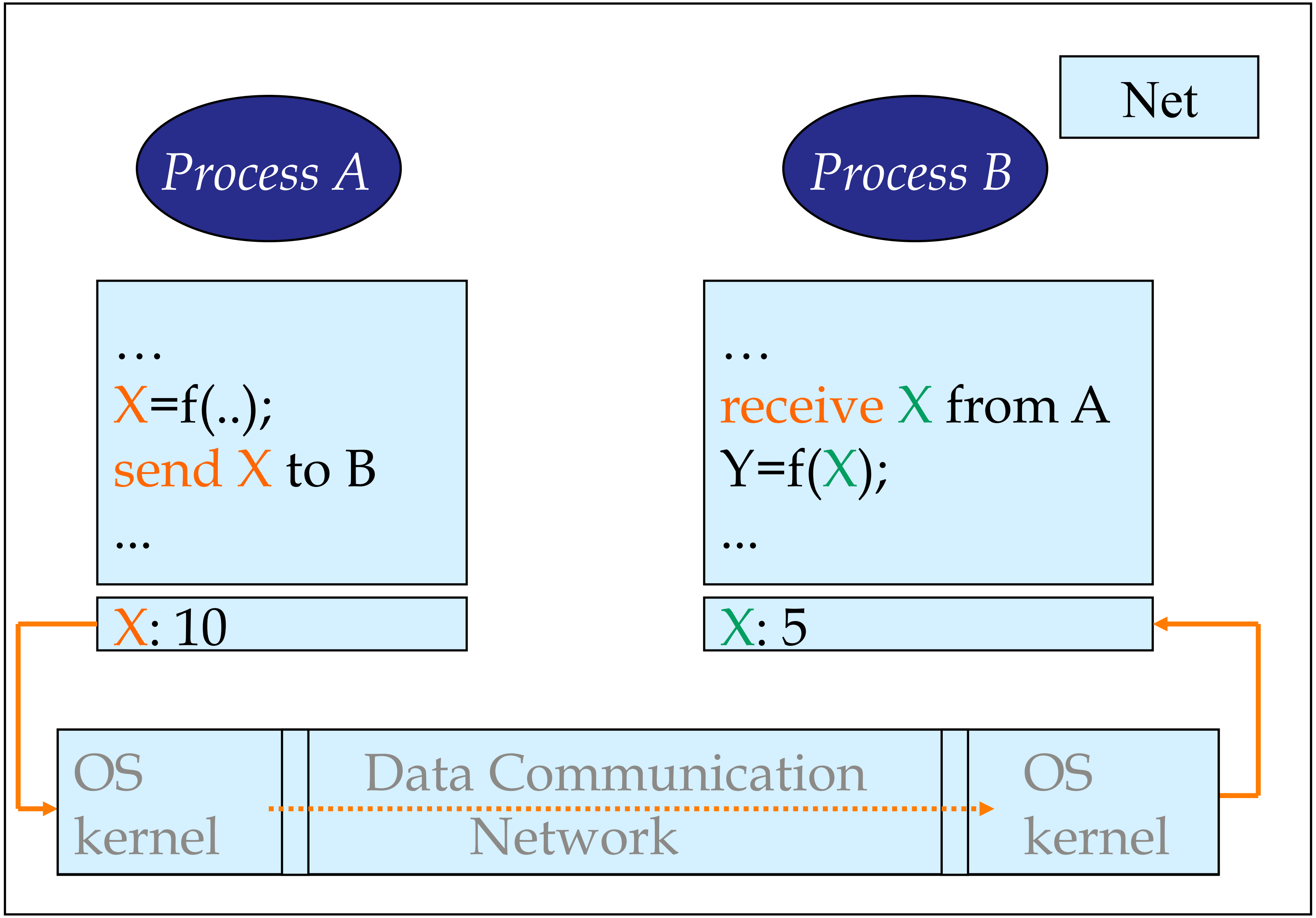


Source: JY Stervinou

RPC: Different Systems



Communication: Message Passing



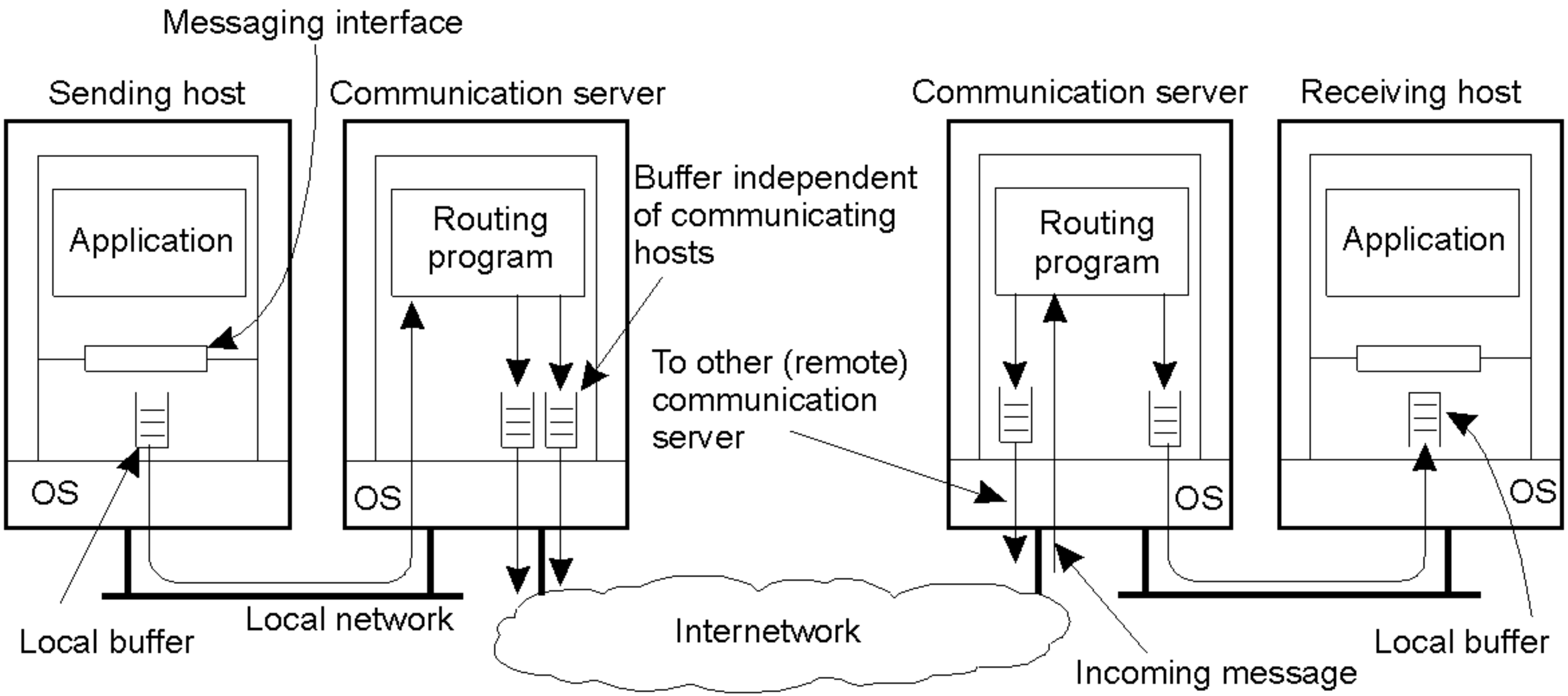
Binding (1)

- Structure of communication network
  - one-to-one (two partners, one shared channel)
  - many-to-one (client-server)
  - one-to-many, many-to-many (client-service; group communication)
- Types of message passing
  - send, multicast, broadcast
  - on any channel structure

Binding (2)

- Time of binding
  - static naming (at programming time)
  - dynamic naming (at execution time)
    - explicit binding of channels
    - implicit binding through name service

Persistence and Synchronicity in Communication

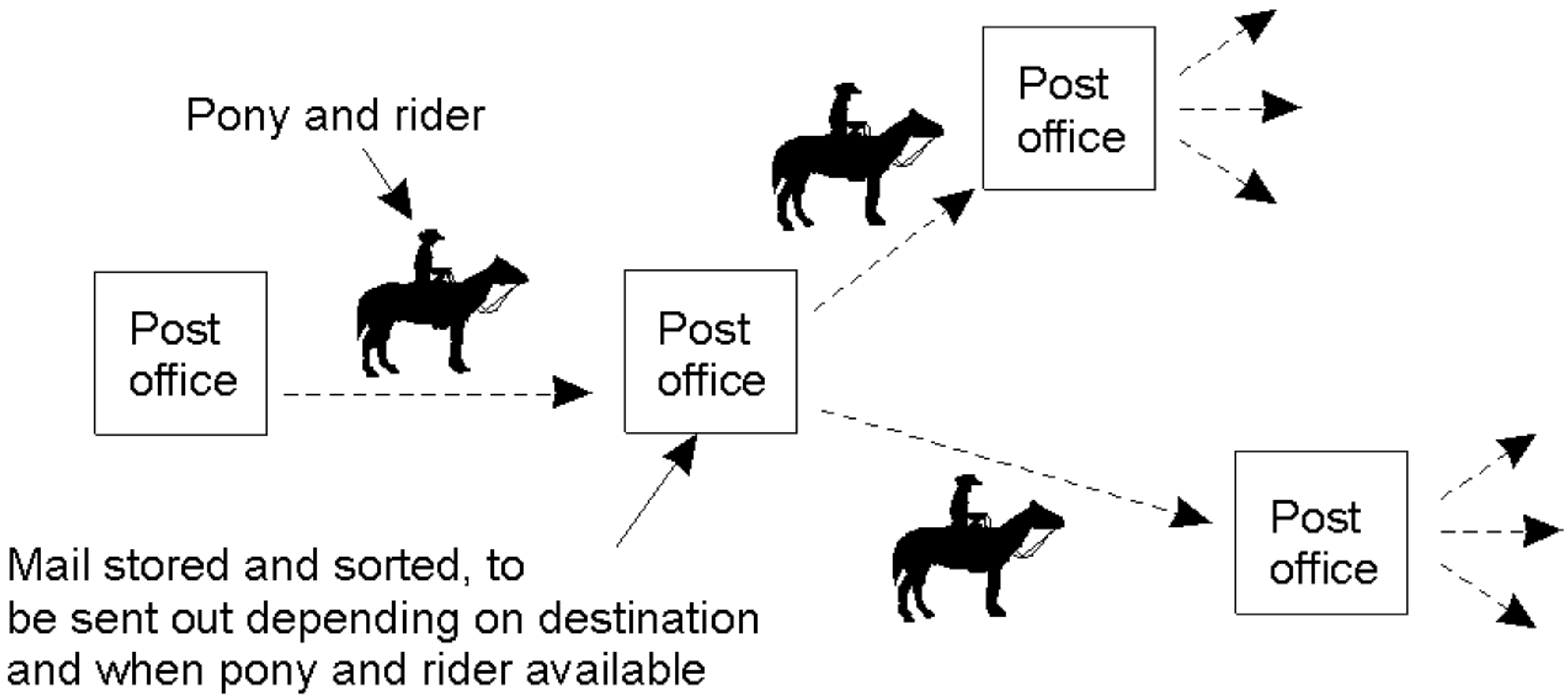


General organization of a communication system in which hosts are connected through a network

Persistence and Synchronicity in Communication

- Persistent communication
  - a submitted message is stored in the system until delivered to the receiver
  - (the receiver may start later, the sender may stop earlier)
- Transient communication
  - a message is stored only as long as the sending and receiving applications are executing
  - (the sender and the receiver must be executing in parallel)

Persistence and Synchronicity in Communication

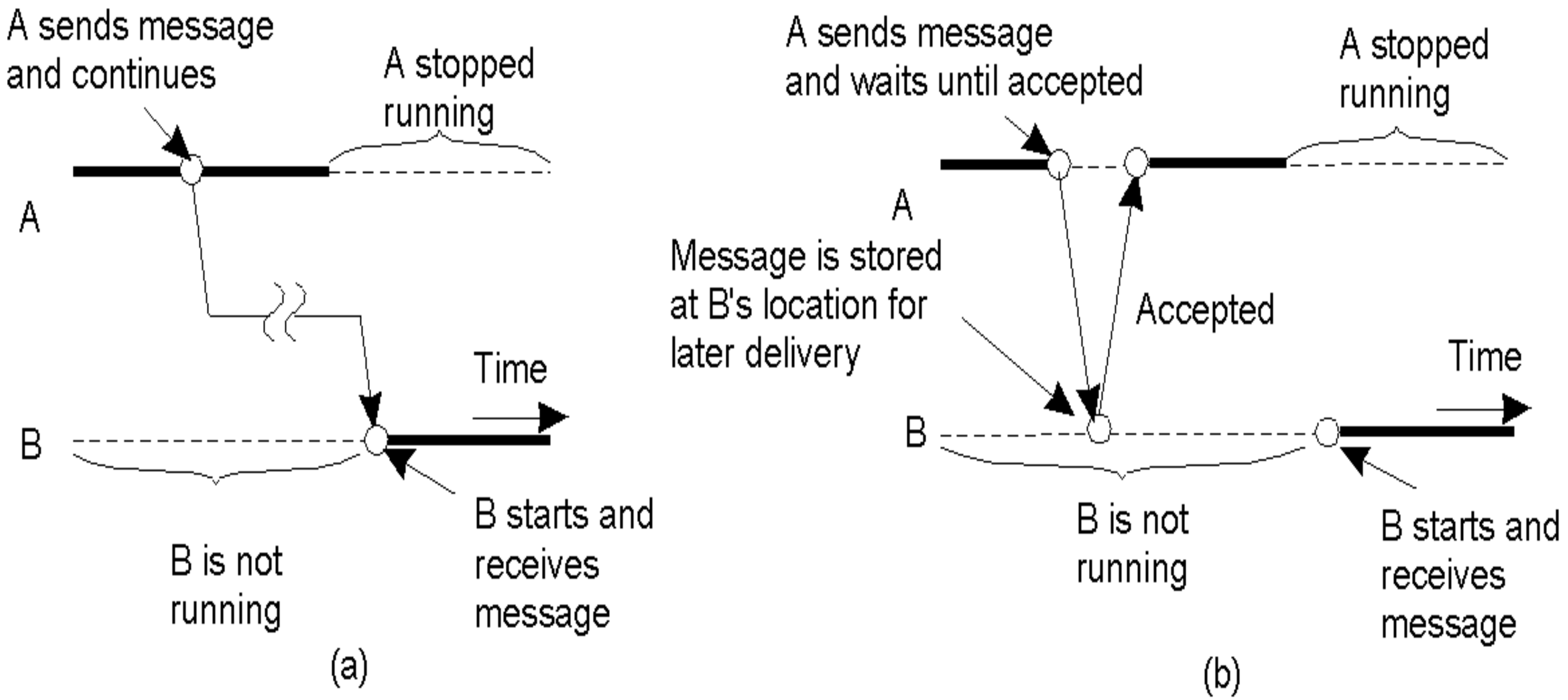


Persistent communication of letters back in the days of the Pony Express.

Persistence and Synchronicity in Communication

- Asynchronous communication
  - the sender continues immediately after submission
- Synchronous communication
  - the sender is blocked until
    - the message is stored at the receiving host (receipt-based synchrony)
    - the message is delivered to the receiver (delivery based)
    - the response has arrived (response based)

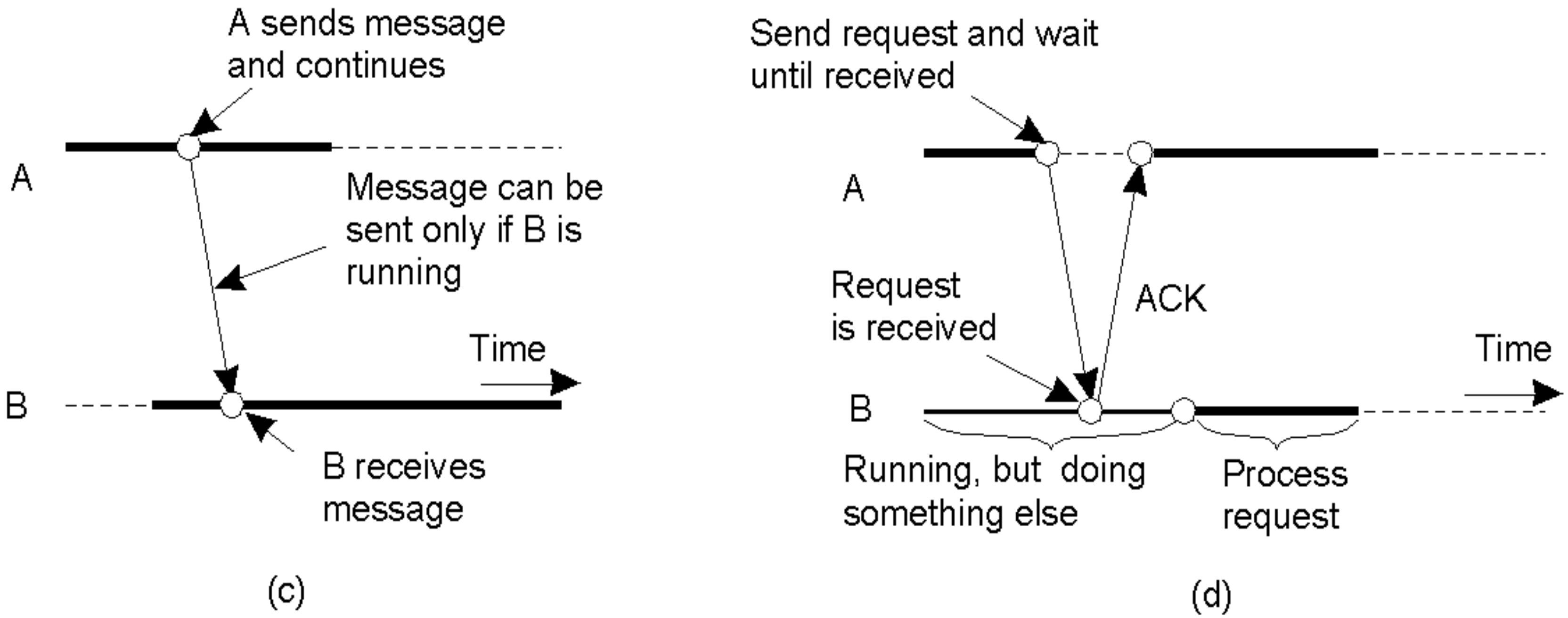
Persistence and Synchronicity in Communication



a) Persistent asynchronous communication

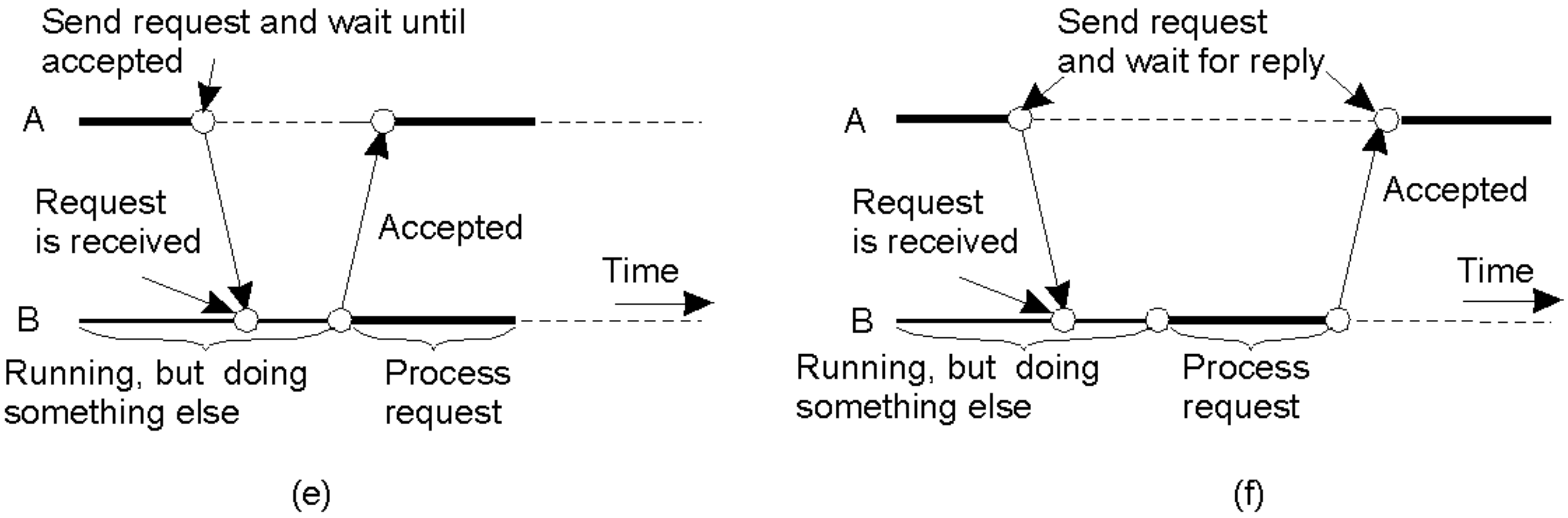
b) Persistent synchronous communication

Persistence and Synchronicity in Communication



c) Transient asynchronous communication
d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication



e) Delivery-based transient synchronous communication at message delivery
f) Response-based transient synchronous communication

The Message-Passing Interface (MPI)

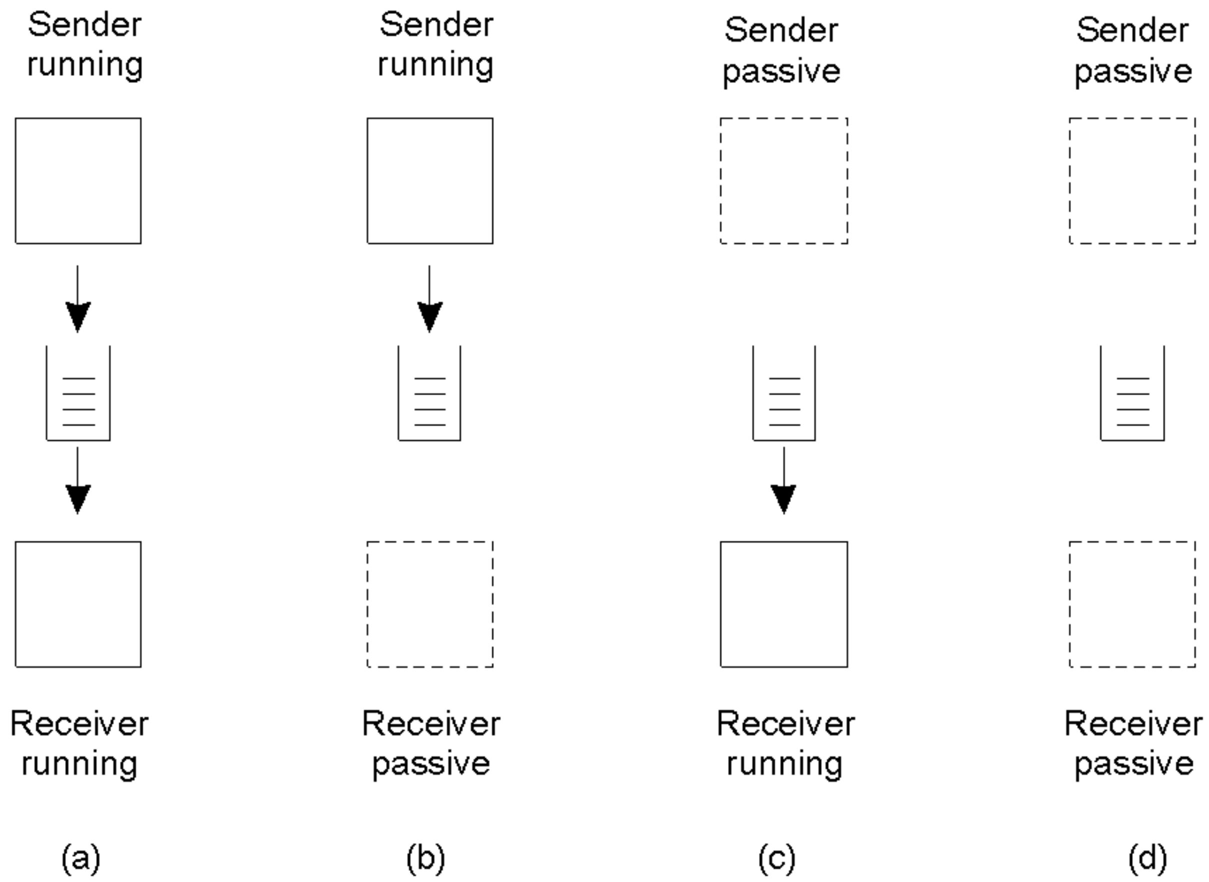
- Traditional communication: sockets
- Platform of concern: high-performance multicomputers
- Issue: easy-to-use communication for applications
- Sockets? No: wrong level, non-suitable protocols
- a new message passing standard: MPI
  - designed for parallel applications, transient communication
  - no communication servers
  - no failures (worth to be recovered from)

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-Queuing Model (1)



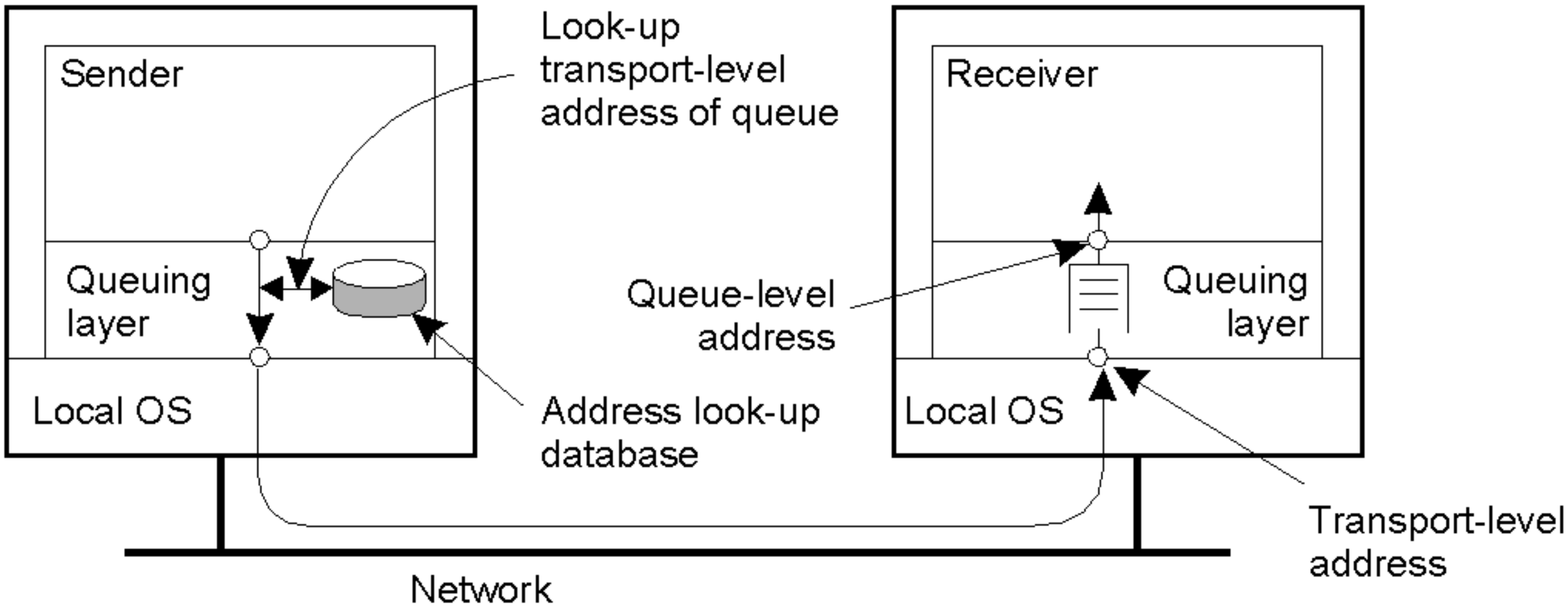
Four combinations for loosely-coupled communications using queues.

Message-Queuing Model

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.

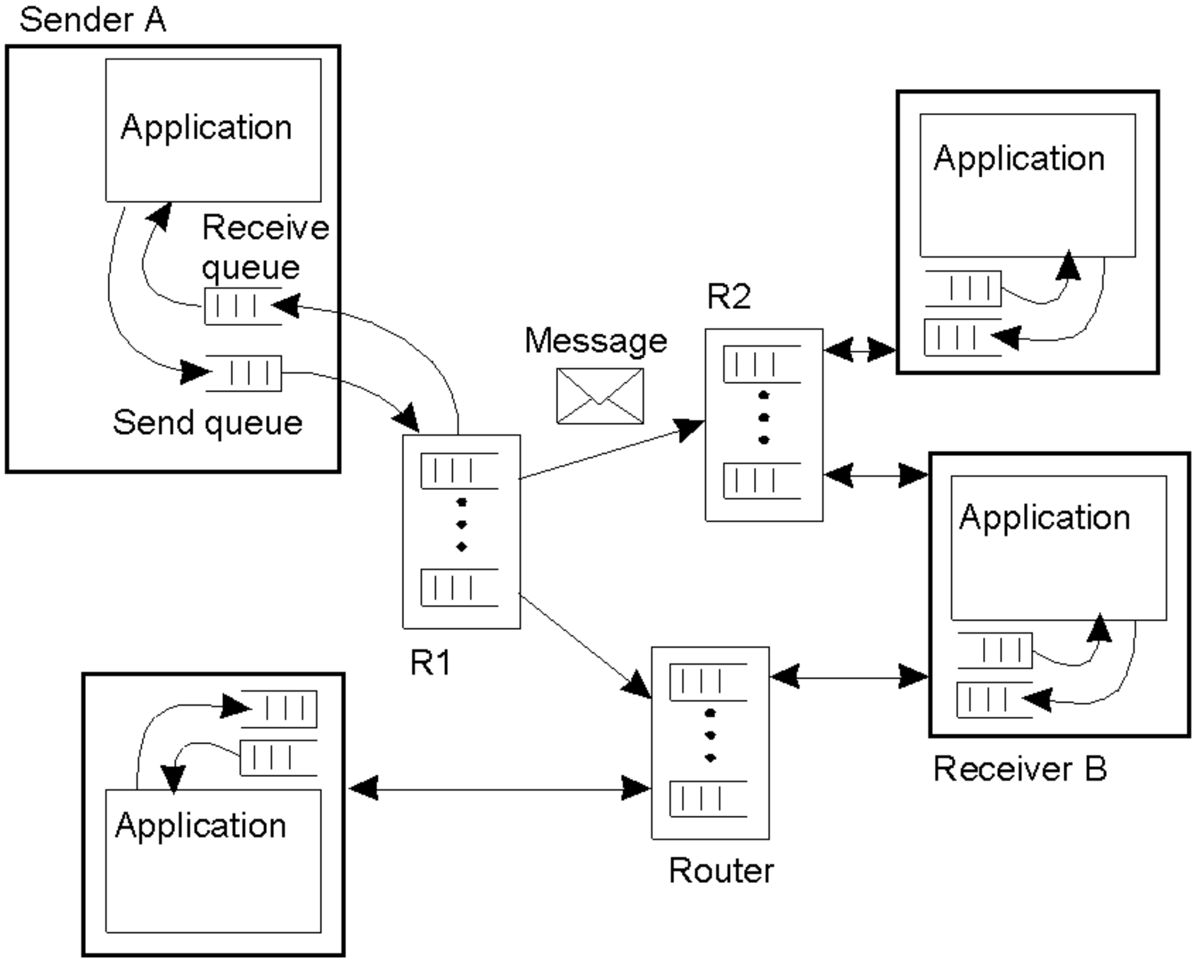
Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System



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

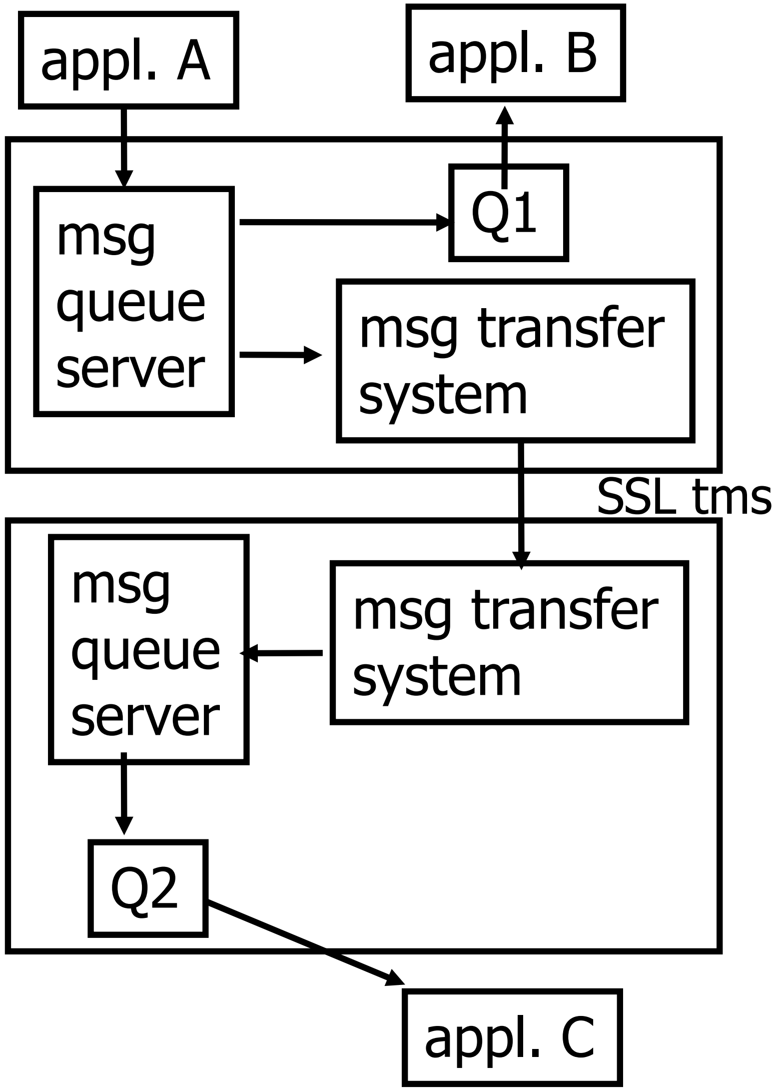
General Architecture of a Message-Queuing System



2-29. The general organization of a message-queuing system with routers.

Message oriented middleware

- asynchronous messages
  - reliable, fault-tolerant
  - no loss, duplication, permutation, cluttering
- persistent subscriptions
- models supported
  - message queue
  - request-response
  - multicast
  - publish-subscribe



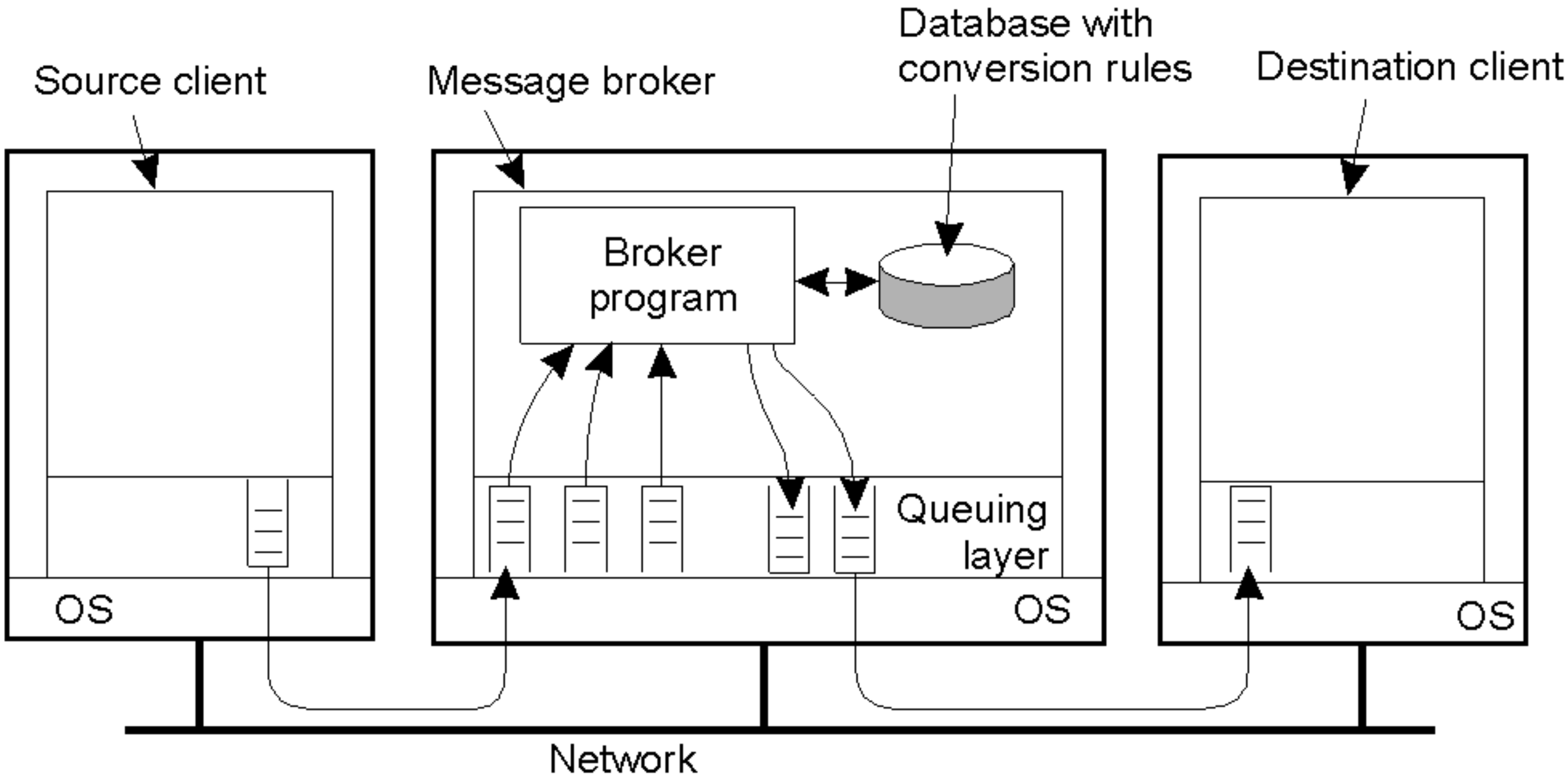
MOM = message oriented middleware

- Basic model: pipe between client and server
  - asynchronous messaging natural, synchronous communication cumbersome
  - message queues support reliability of message transport
  - violates access transparency, no support for data heterogeneity unless in programming language mapping, no support for transactions
  - suitable for event notifications, publish/subscribe-based architectures
  - persistent message queues support fault tolerance

MOM Topics

- Topics for variation and development
  - persistent/transient msgs
  - FIFO/priority queues
  - translations of msgs
  - abstractions on msg ordering
  - multithreading, automatic load balancing
  - msg routing (source, cost, changes in topology etc)
  - secure transfer of msgs (at least between msg servers)

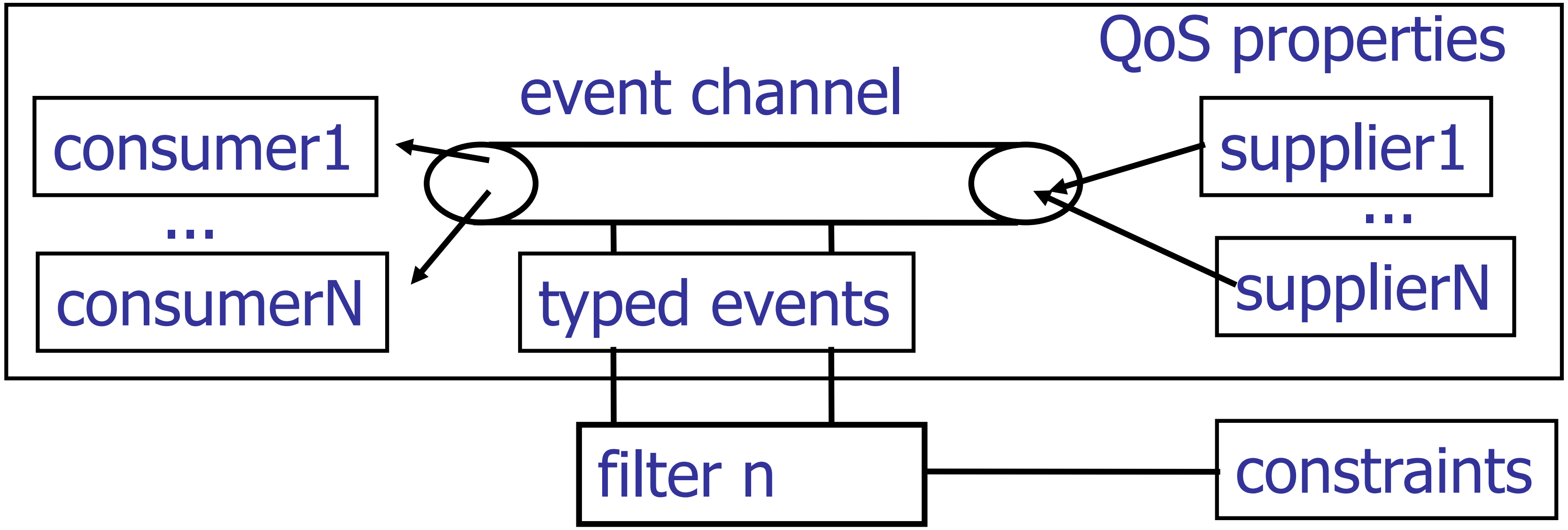
Message Brokers



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

CORBA Events & Notifications

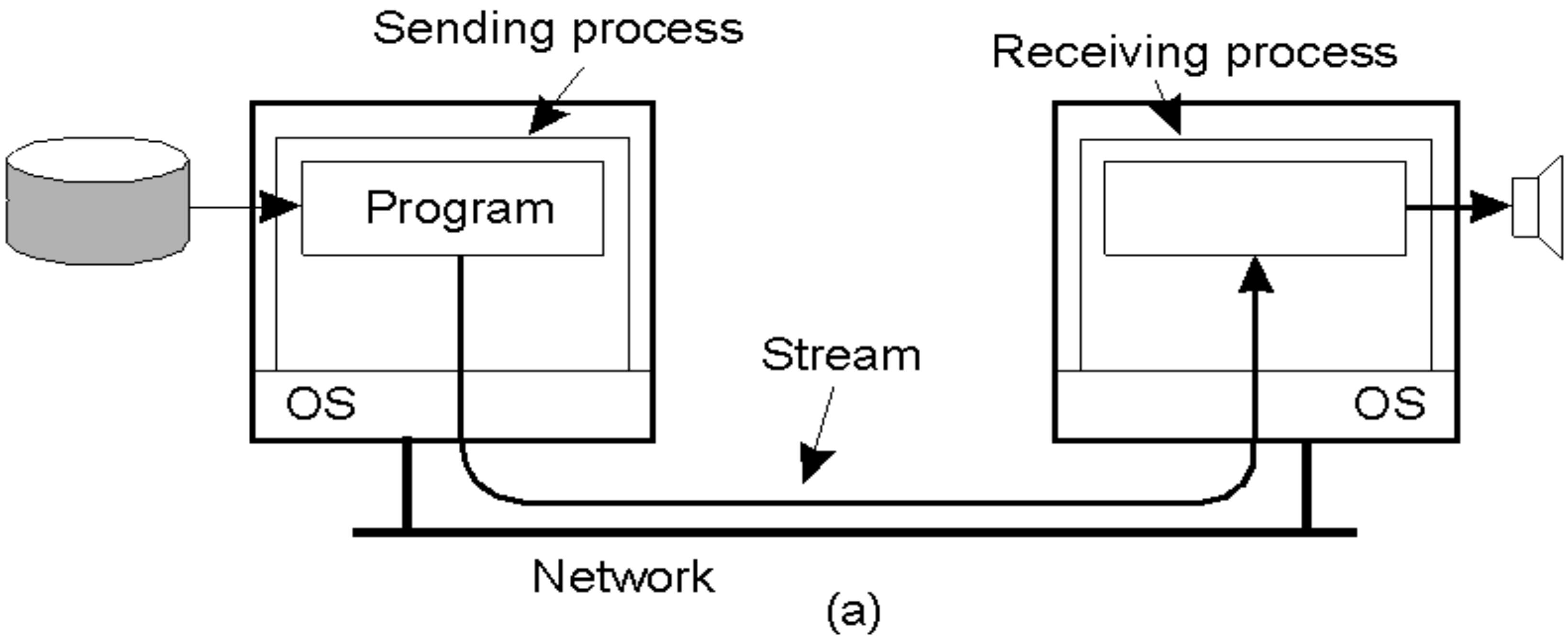
- Event namespace (names and attributes)
- Typed events (header+body; fixed + other)
- Consumer event filtering, event batching, event priority, event expiration, logging, internationalization, flow control mechanism



Publish-subscribe

- shared mailbox, everyone can send to it
- subscribers can select what filter to use
- guaranteed delivery of all relevant messages to all subscribers
- models: header-based, topic-based
- problems
  - scalability: comparing filters and messages
  - ordering of messages

Stream communication



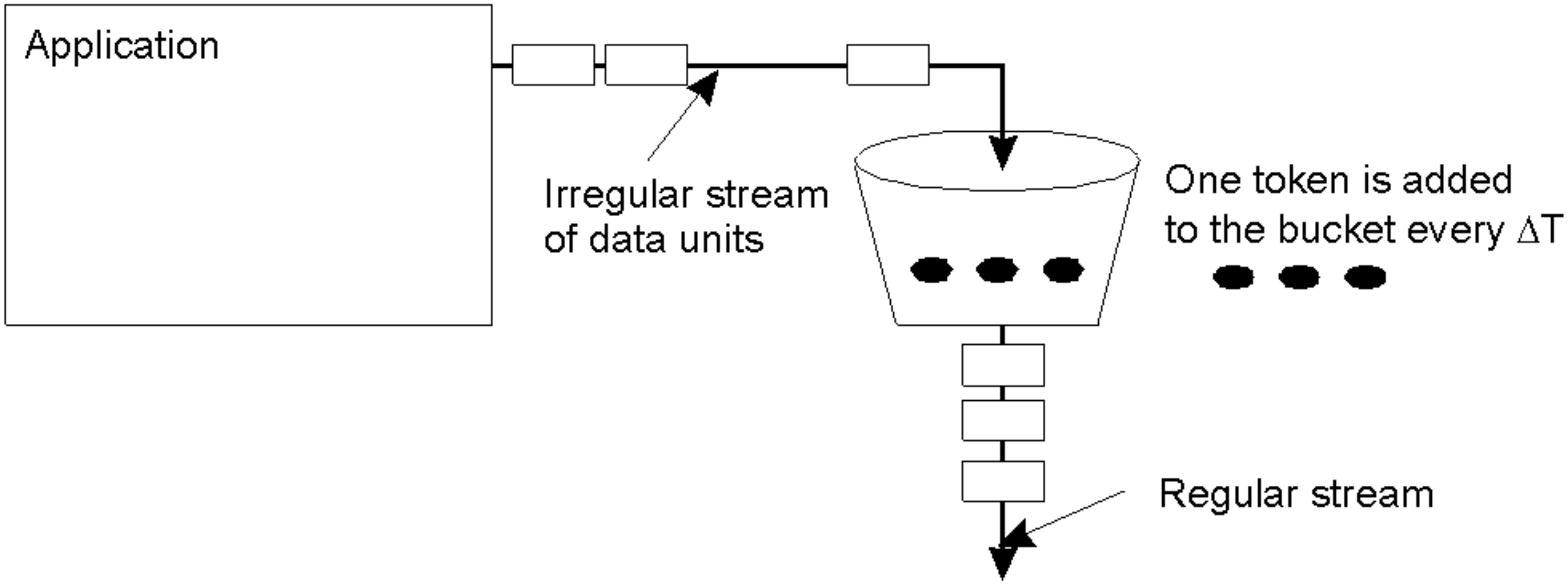
- Setting up a stream between two processes across a network.

Specifying QoS (1)

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

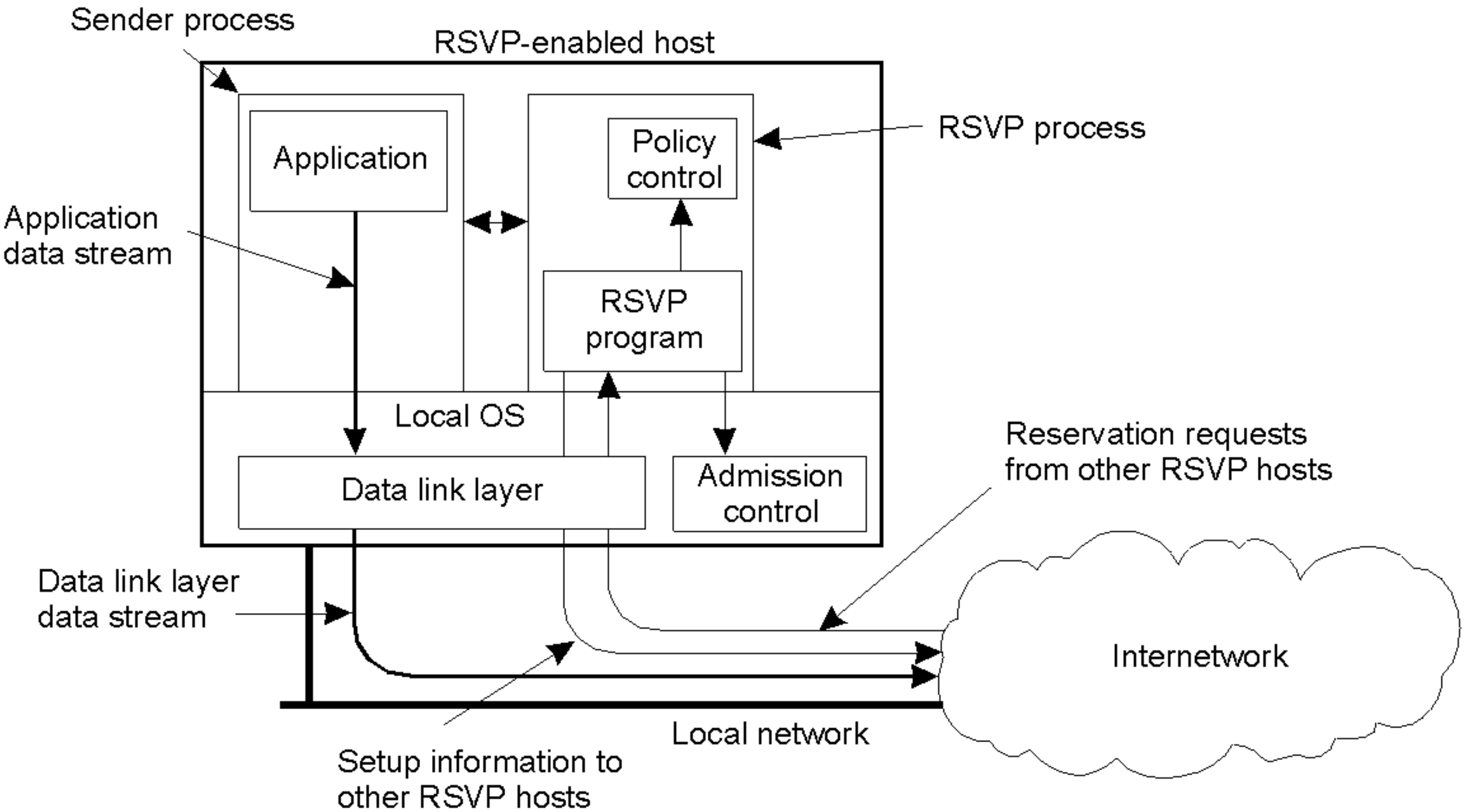
- A flow specification.

Specifying QoS (2)



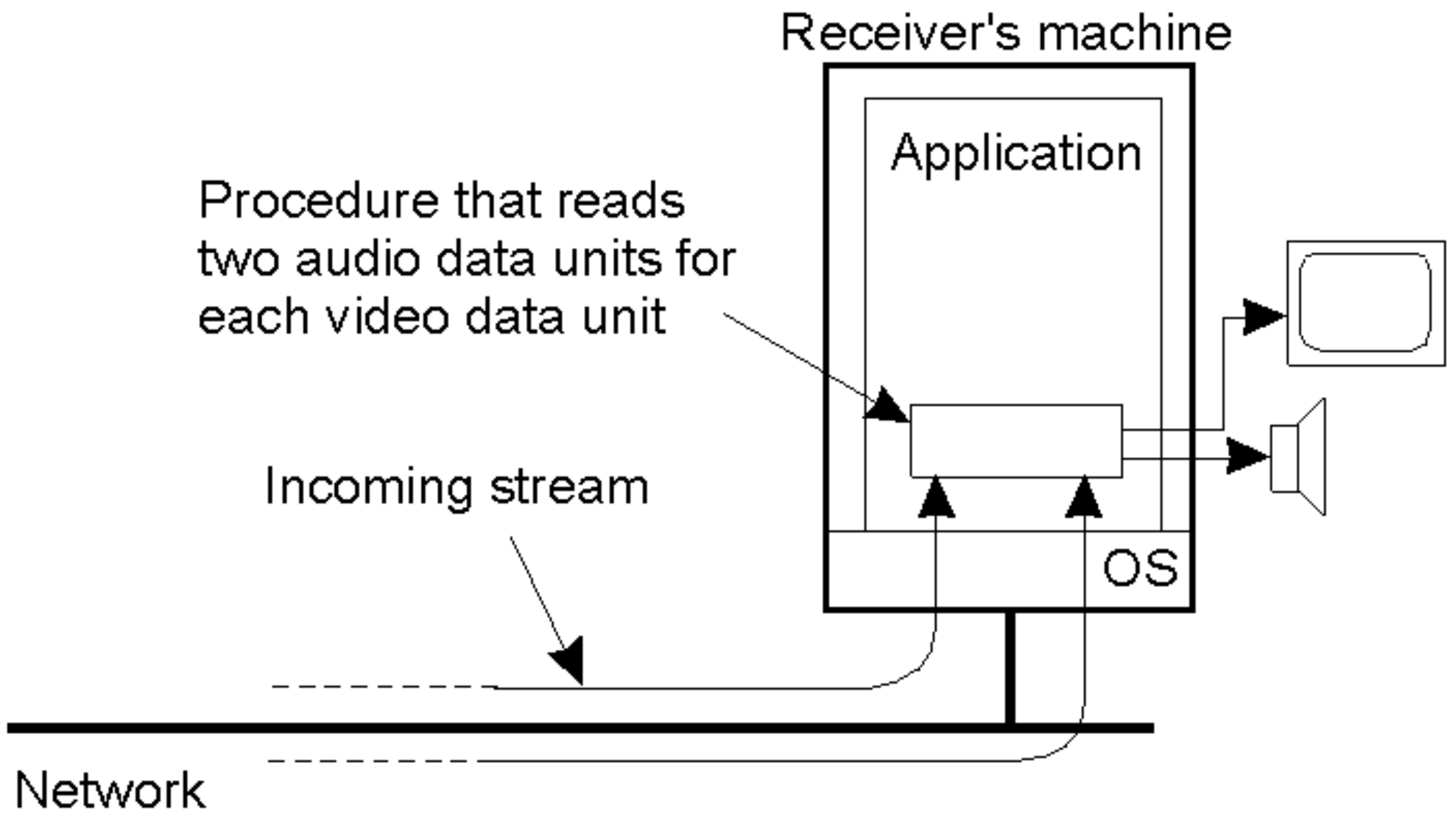
- The principle of a token bucket algorithm.

Setting Up a Stream



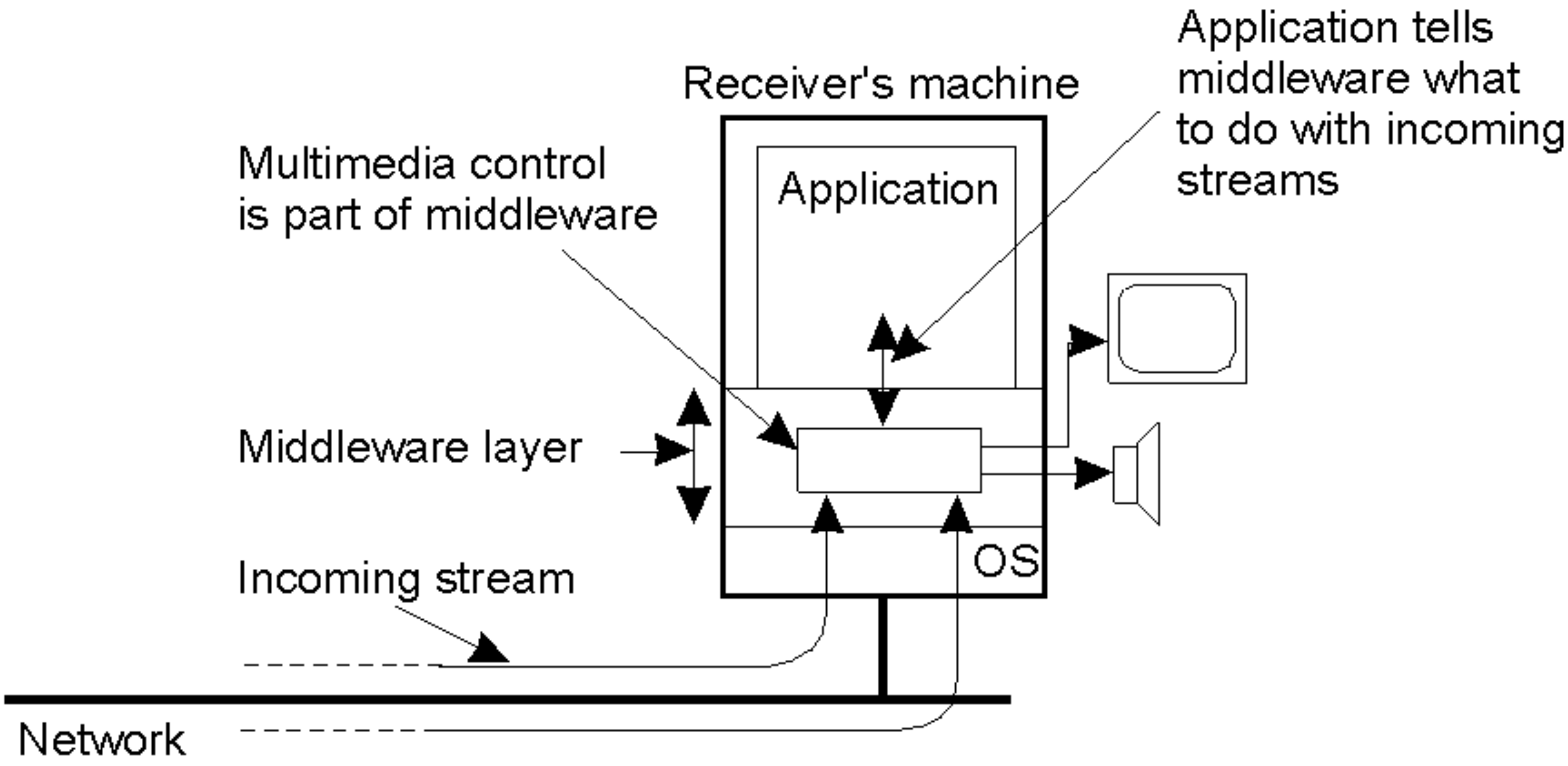
- The basic organization of RSVP for resource reservation in a distributed system.

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.

Other forms of communication

- Multicast (application level)
  - overlay network where relays not members of group (tree, mesh)
- Gossip-based data dissemination
  - infect other nodes with useful data by an epidemic algorithm
  - periodically exchange information with a random node
  - states: infected, susceptible, data removed

Chapter Summary

- Overview of different interprocess communication techniques and solutions
- Remote invocations (RPC etc.)
- Message passing
- Streams
- Publish/subscribe
- Multicast (more on this later)