

HELSINGIN YLIOPISTO  
HELSINGFORS UNIVERSITET  
UNIVERSITY OF HELSINKI

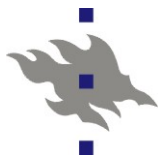
# **Chapter 2: Distributed Systems: Interprocess communication**

Fall 2013  
*Jussi Kangasharju*

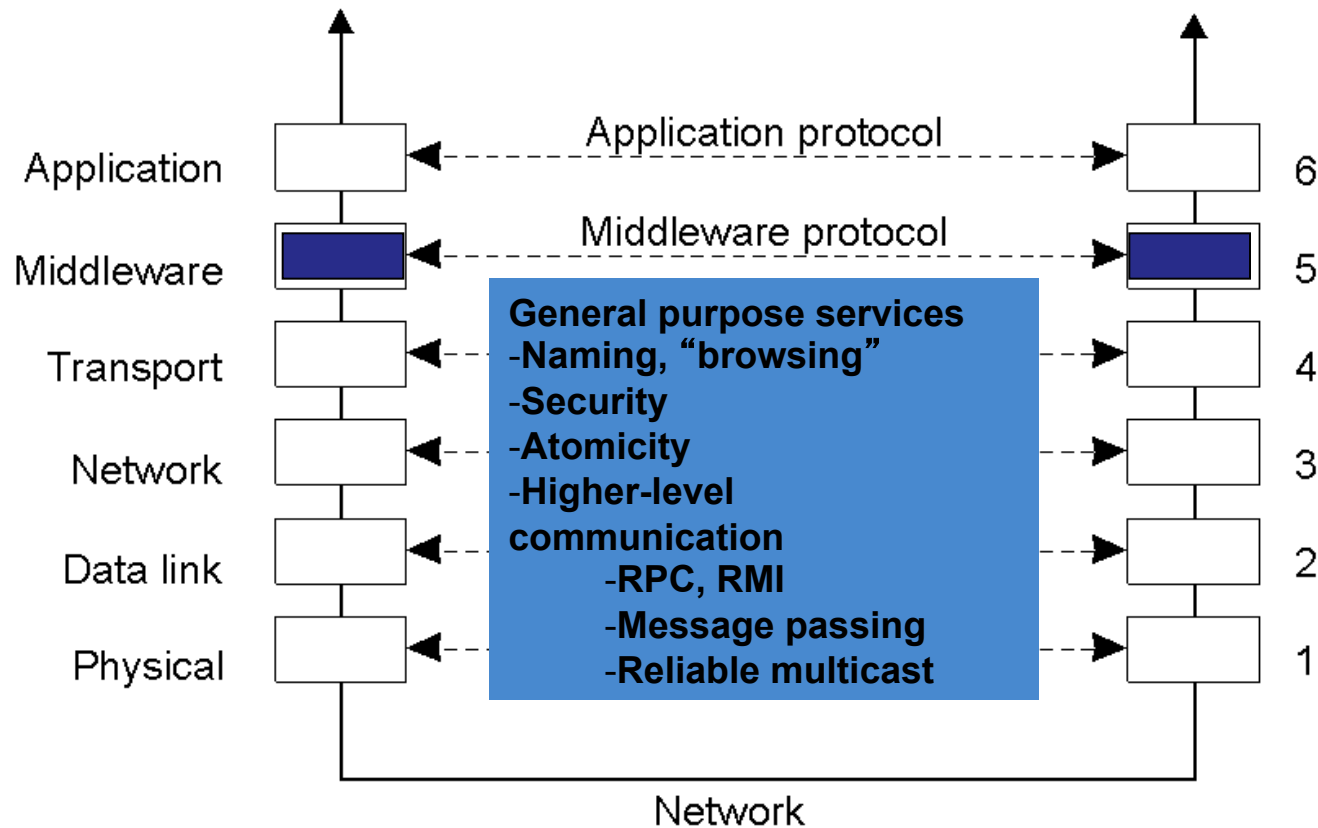


## Chapter Outline

- Overview of interprocess communication
- Remote invocations (RPC etc.)
- Persistence and synchronicity



# 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

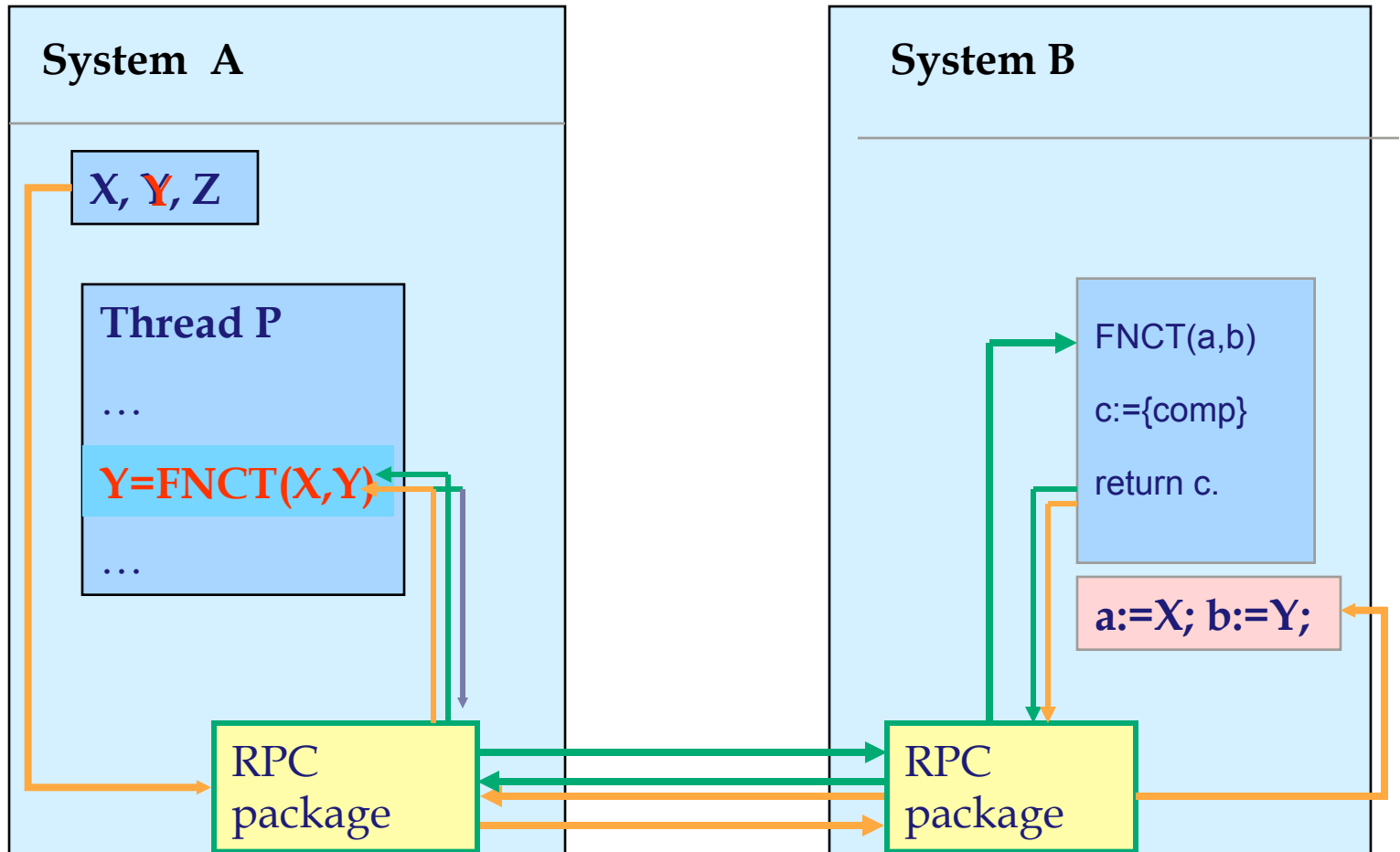


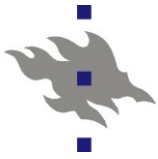
## RPC goals

- 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

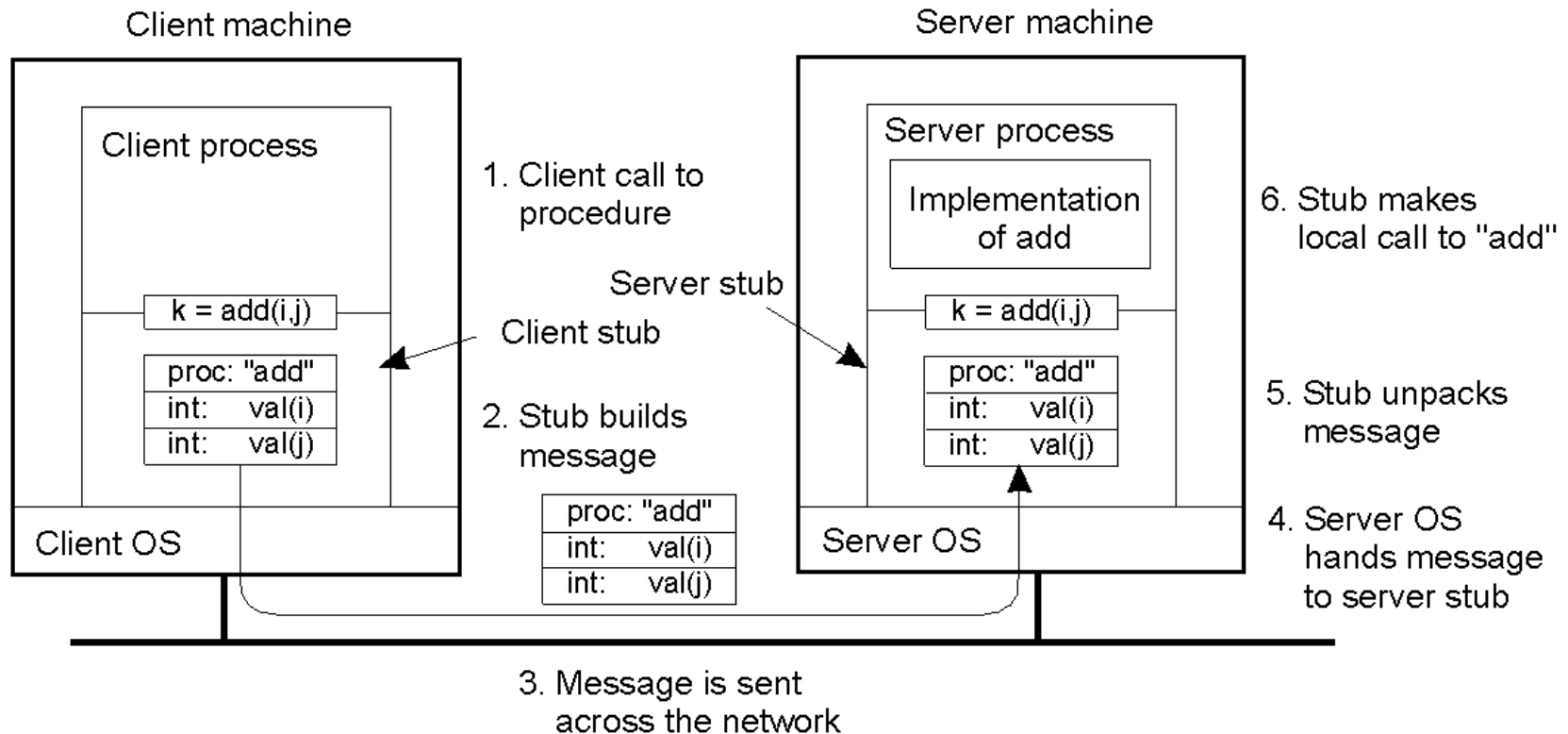
- responsible for message passing
- part of the operating system

## ■ Name service: look up, binding

- name of procedure, interface definition



# Passing Value Parameters

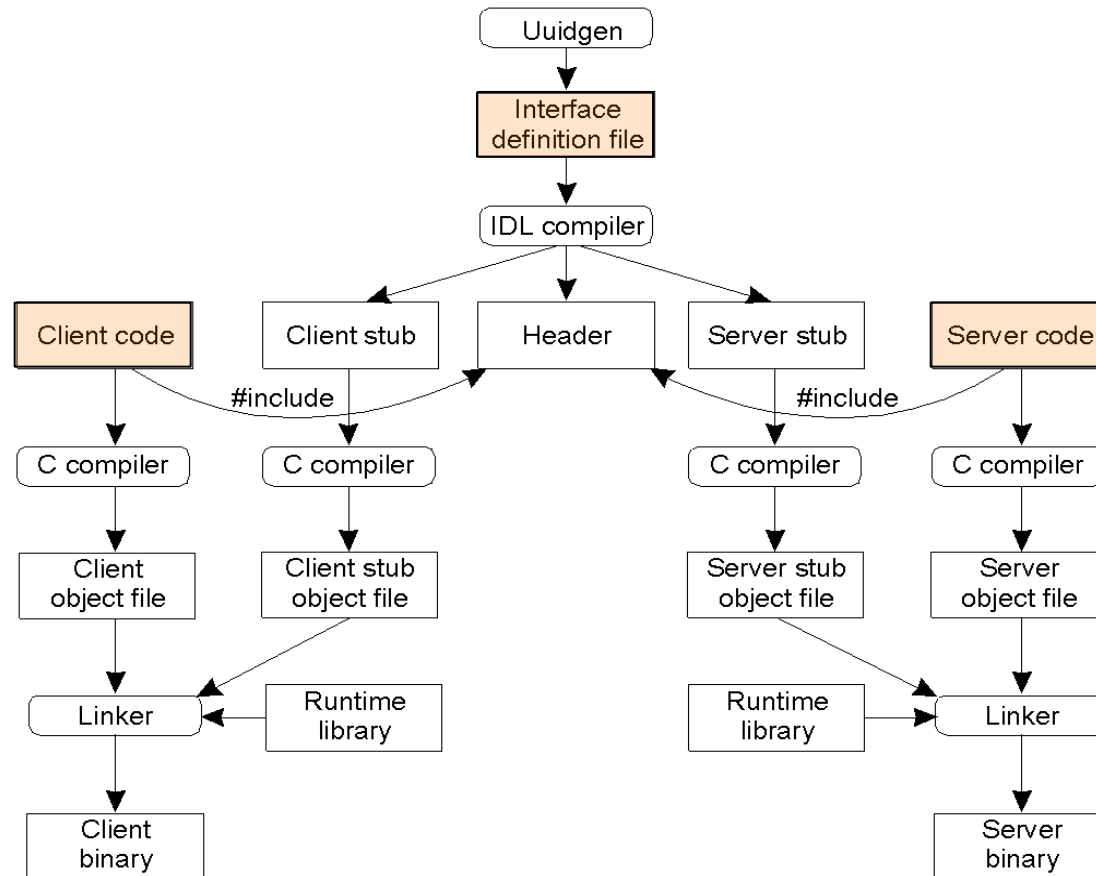


Steps involved in doing remote computation through RPC

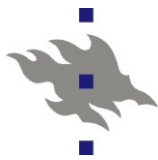




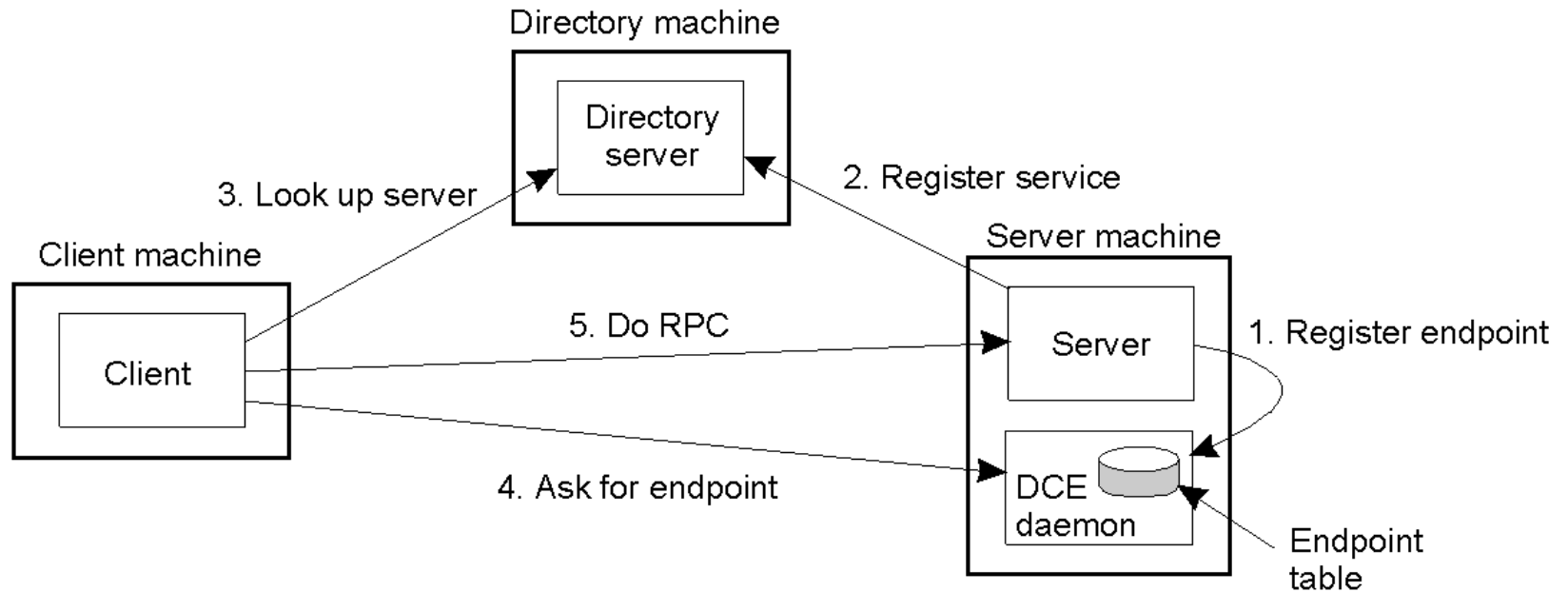
## Writing a Client and a Server



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



## 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)



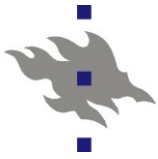
## RPC: Types of failures

- 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

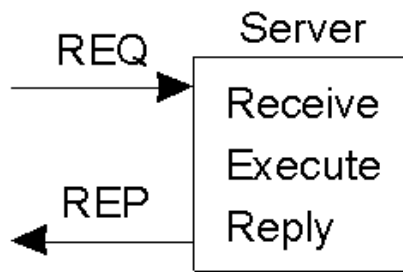


# Reliable Client-Server Communication

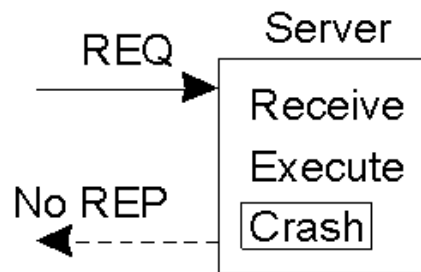
1. Point-to-Point Communication (“reliable”)
  - masked: omission, value
  - not masked: crash, (timing)
2. RPC semantics
  - the client unable to locate the server
  - the message is lost (request / reply)
  - the server crashes (before / during / after service)
  - the client crashes



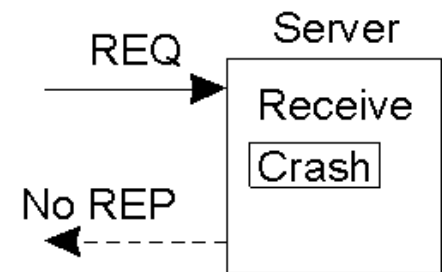
## Server Crashes (1)



(a)



(b)

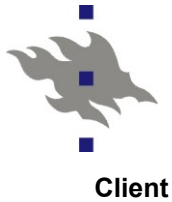


(c)

A server in client-server communication

- a) Normal case
- b) Crash after execution
- c) Crash before execution





## Server Crashes (2)

Server

Strategy M -> P

Strategy P -> M

Reissue strategy
Always
Never
Only when ACKed
Only when not ACKed

MPC	MC(P)	C(MP)
DUP	OK	OK
OK	ZERO	ZERO
DUP	OK	ZERO
OK	ZERO	OK

PMC	PC(M)	C(PM)
DUP	DUP	OK
OK	OK	ZERO
DUP	OK	ZERO
OK	DUP	OK

Different combinations of client and server strategies in the presence of server crashes (client's continuation after server's recovery: reissue the request?)

M: send the completion message

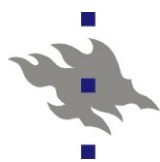
P: print the text

C: crash

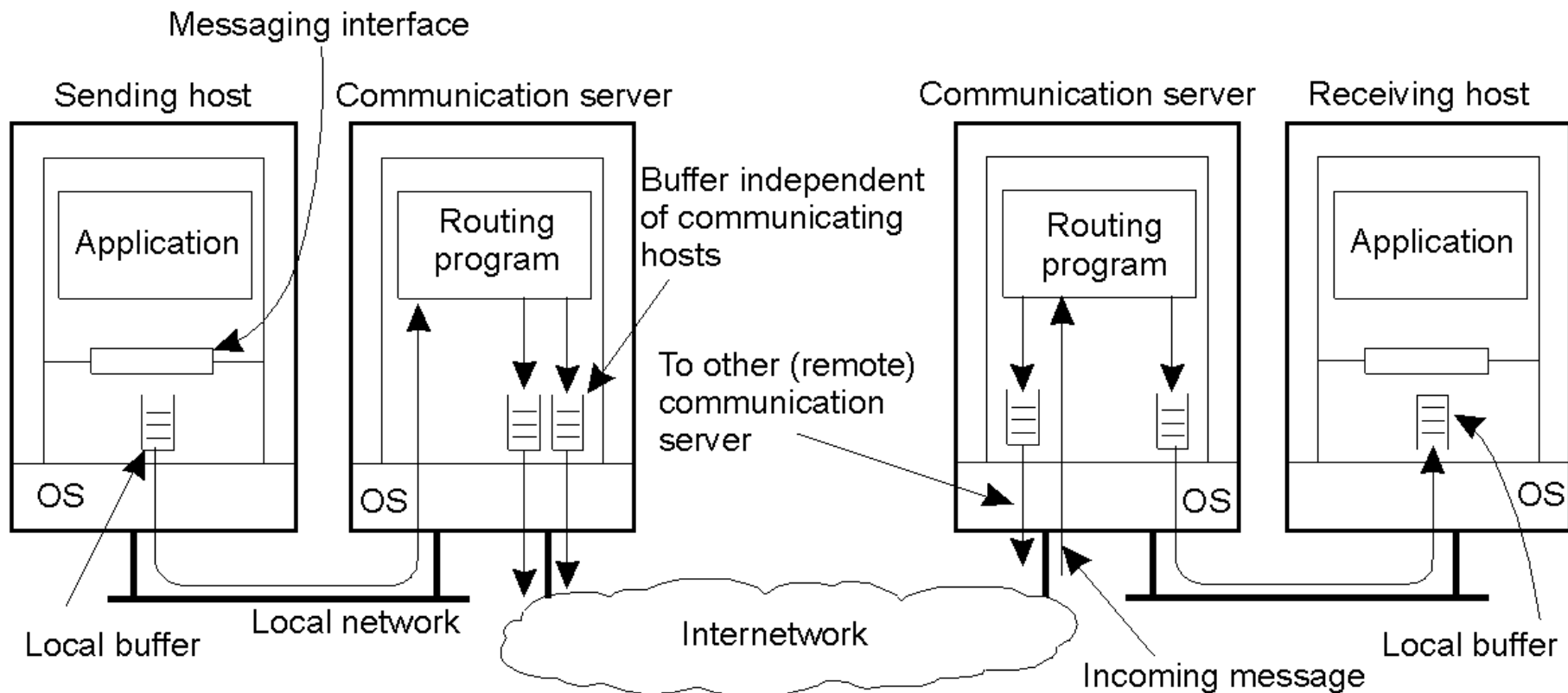


## Client Crashes

- Orphan: an active computation looking for a non-existing parent
- Solutions
  - extermination: the client stub records all calls, after crash recovery all orphans are killed
  - reincarnation: time is divided into epochs, client reboot => broadcast “new epoch” => servers kill orphans
  - gentle incarnation: “new epoch” => only “real orphans” are killed
  - expiration: a “time-to-live” for each RPC (+ possibility to request for a further time slice)
- New problems: grandorphans, reserved locks, entries in remote queues, ....



## Persistence and Synchronicity in Communication



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



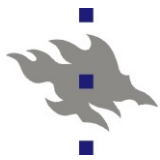
# Persistence and Synchronicity in Communication

## ■ Persistent communication

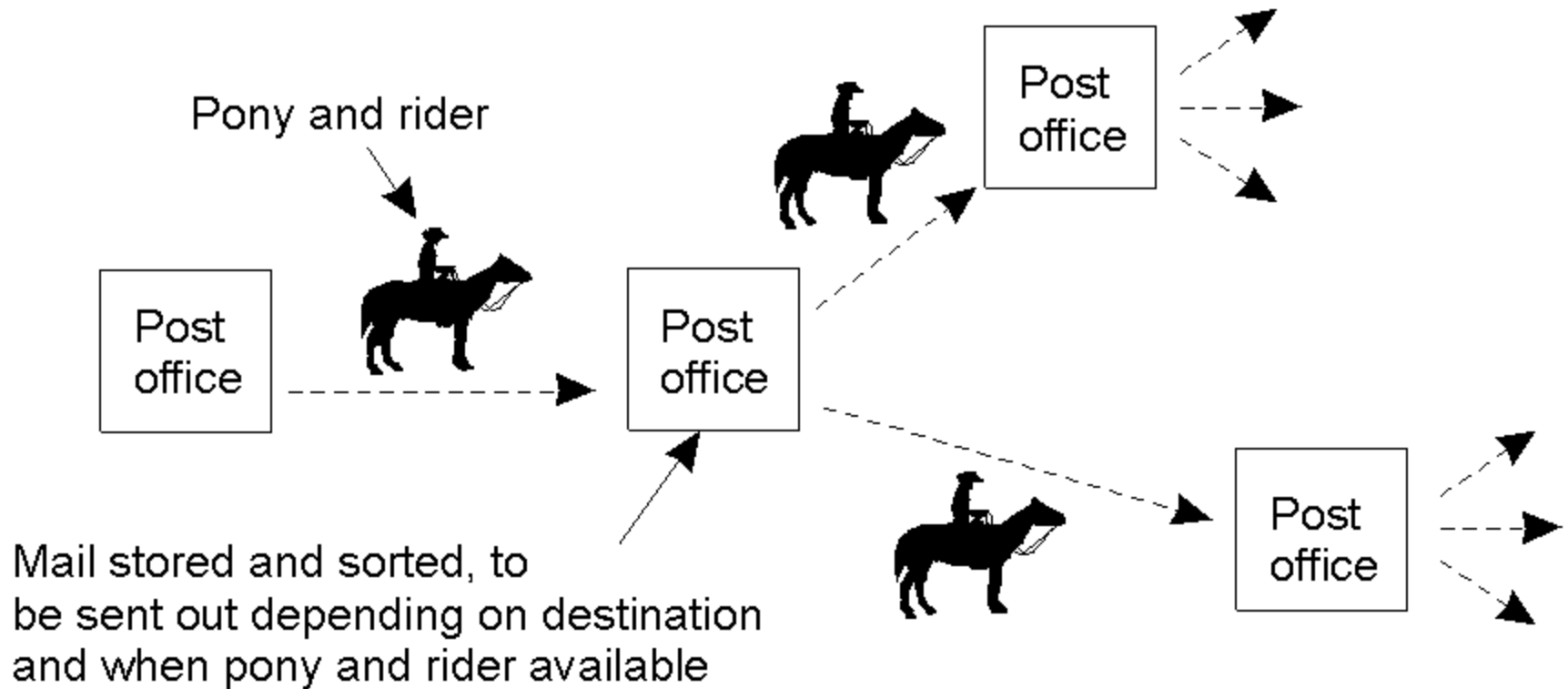
- a submitted message is stored in the system until delivered to the receiver
- (the receiver may start later, the sender may stop earlier)

## ■ 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

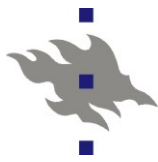


Persistent communication of letters back in the days of the Pony Express.

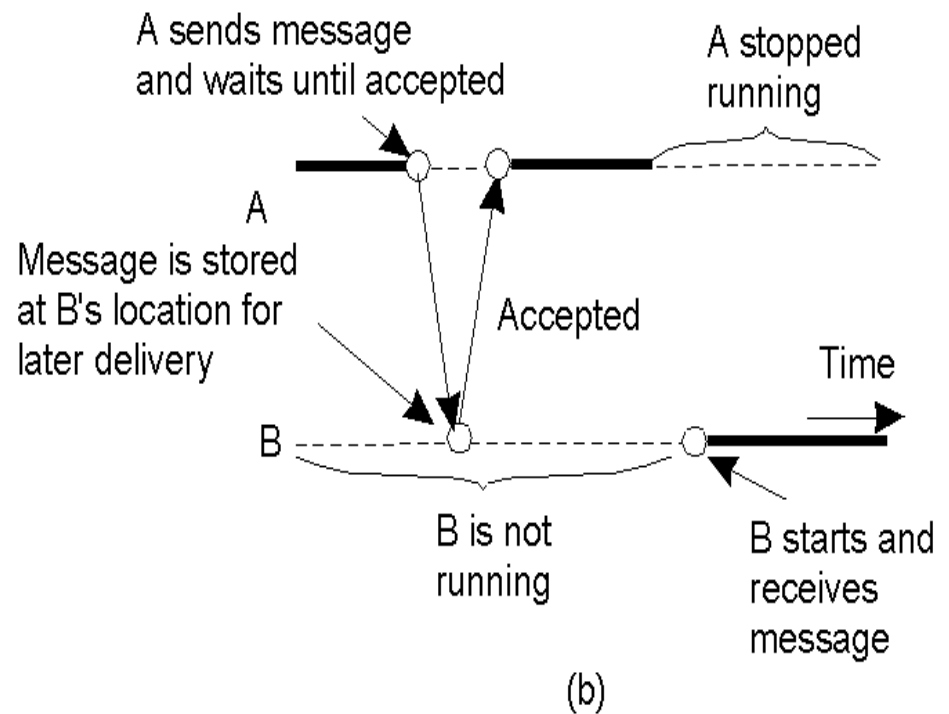
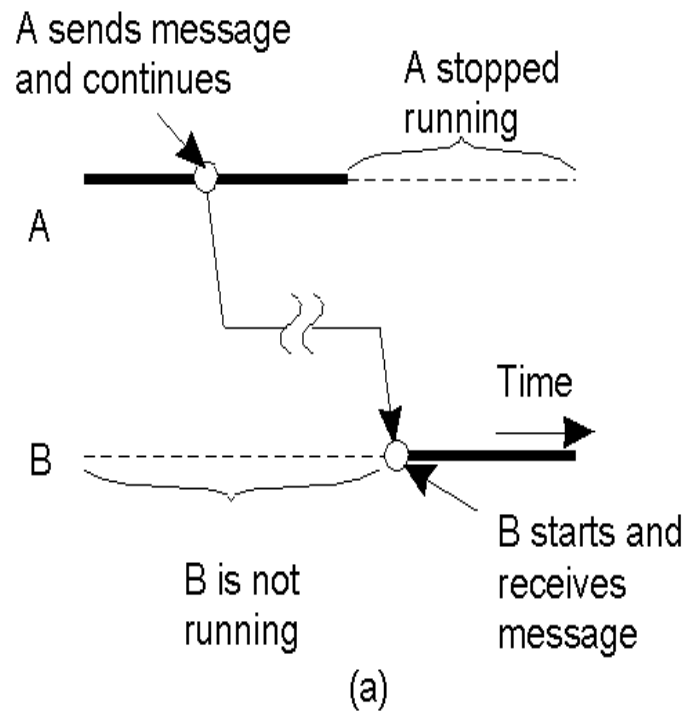


# Persistence and Synchronicity in Communication

- 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)

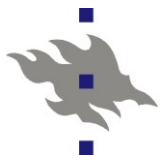


## Persistence and Synchronicity in Communication

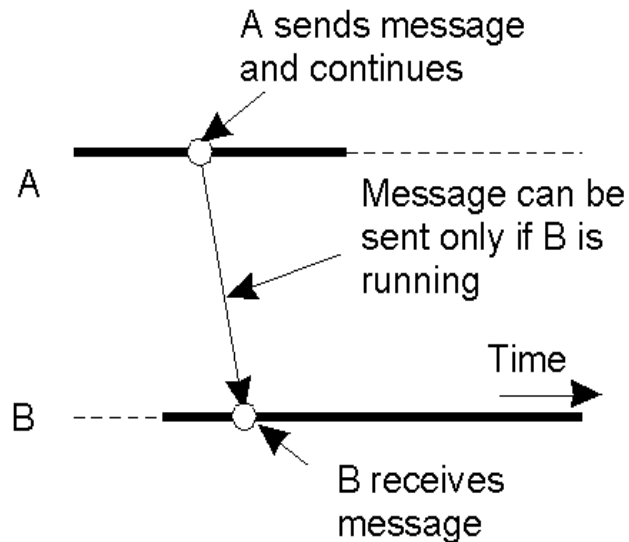


a) Persistent asynchronous communication

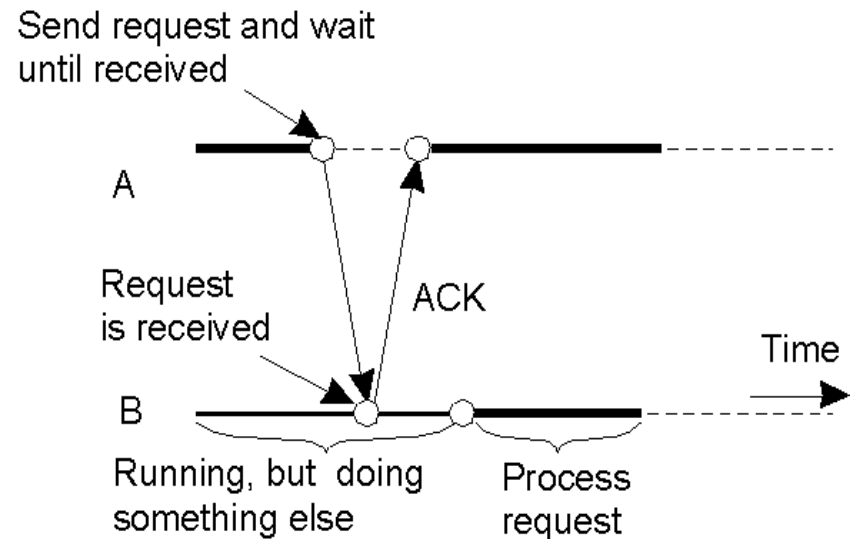
b) Persistent synchronous communication



## Persistence and Synchronicity in Communication



(c)

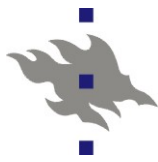


(d)

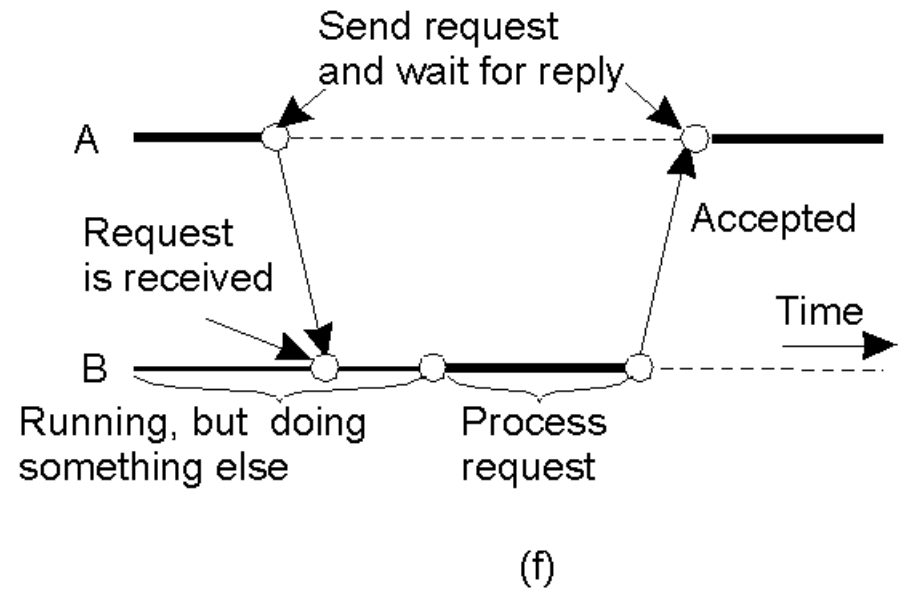
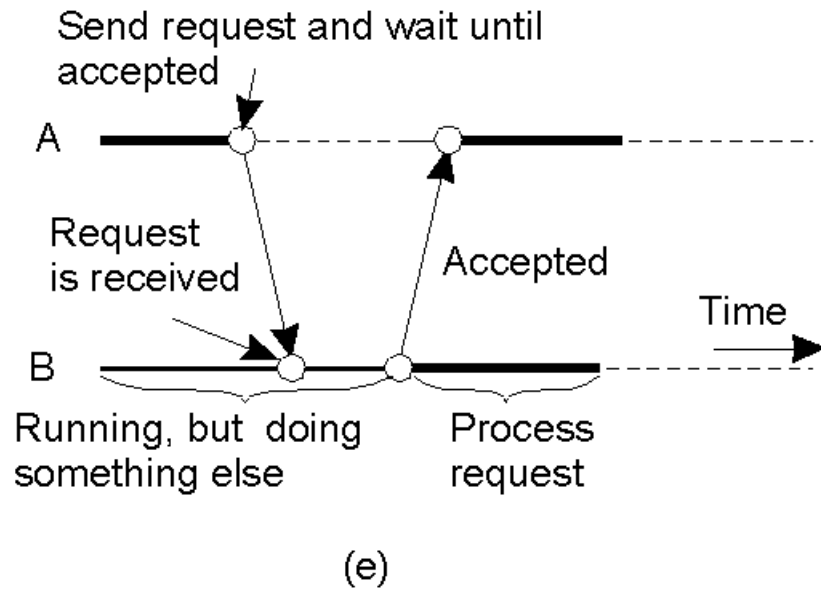
c)  
d)

Transient asynchronous communication  
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



## Chapter Summary

- Overview of interprocess communication
- Remote invocations (RPC etc.)
- Persistence and synchronicity