HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Chapter 2:
Distributed Systems:
Interprocess communication

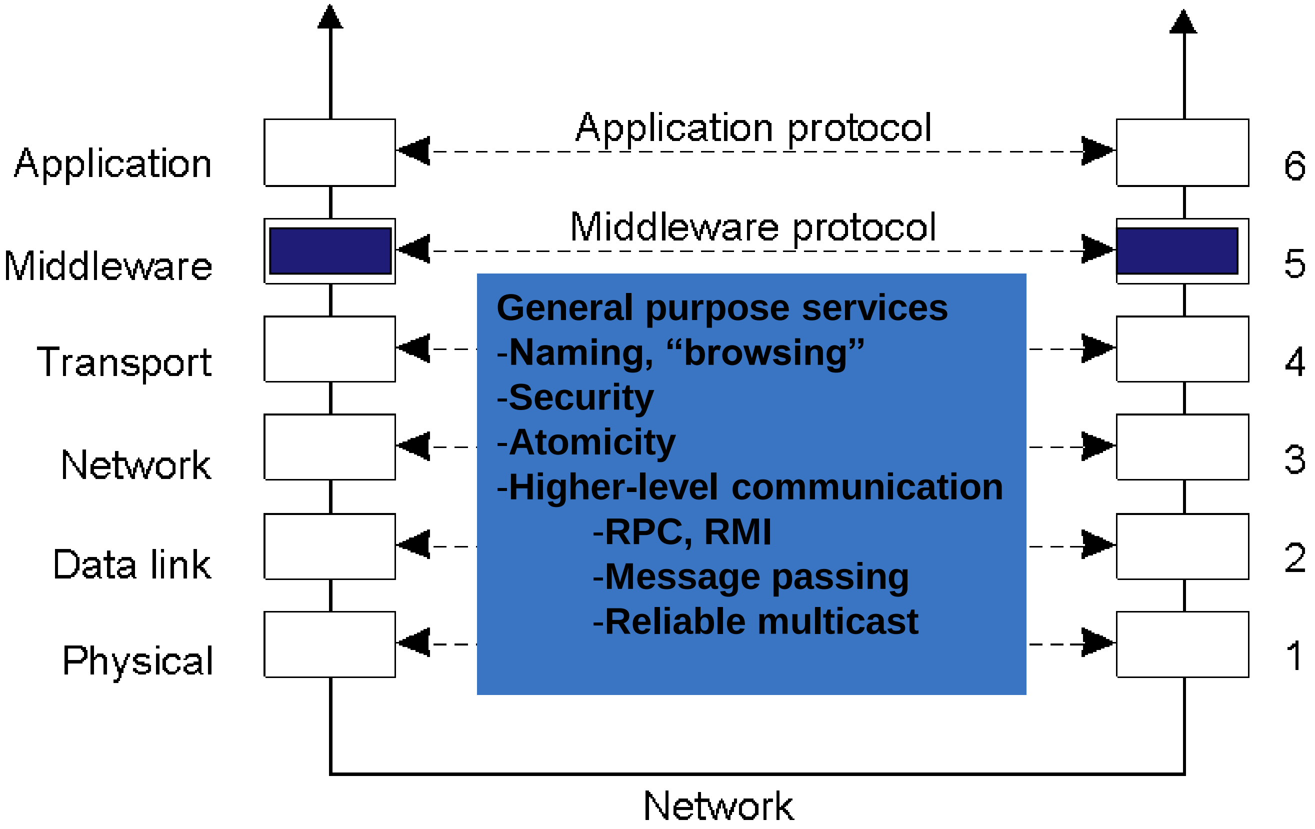
Fall 2012

Sini Ruohomaa
Slides joint work with Jussi Kangasharju et al.

Chapter Outline

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

Middleware Protocols



General purpose services
-Naming, “browsing”
-Security
-Atomicity
-Higher-level communication
- **-RPC, RMI**
- **-Message passing**
- **-Reliable multicast**

An adapted reference model for networked communication.

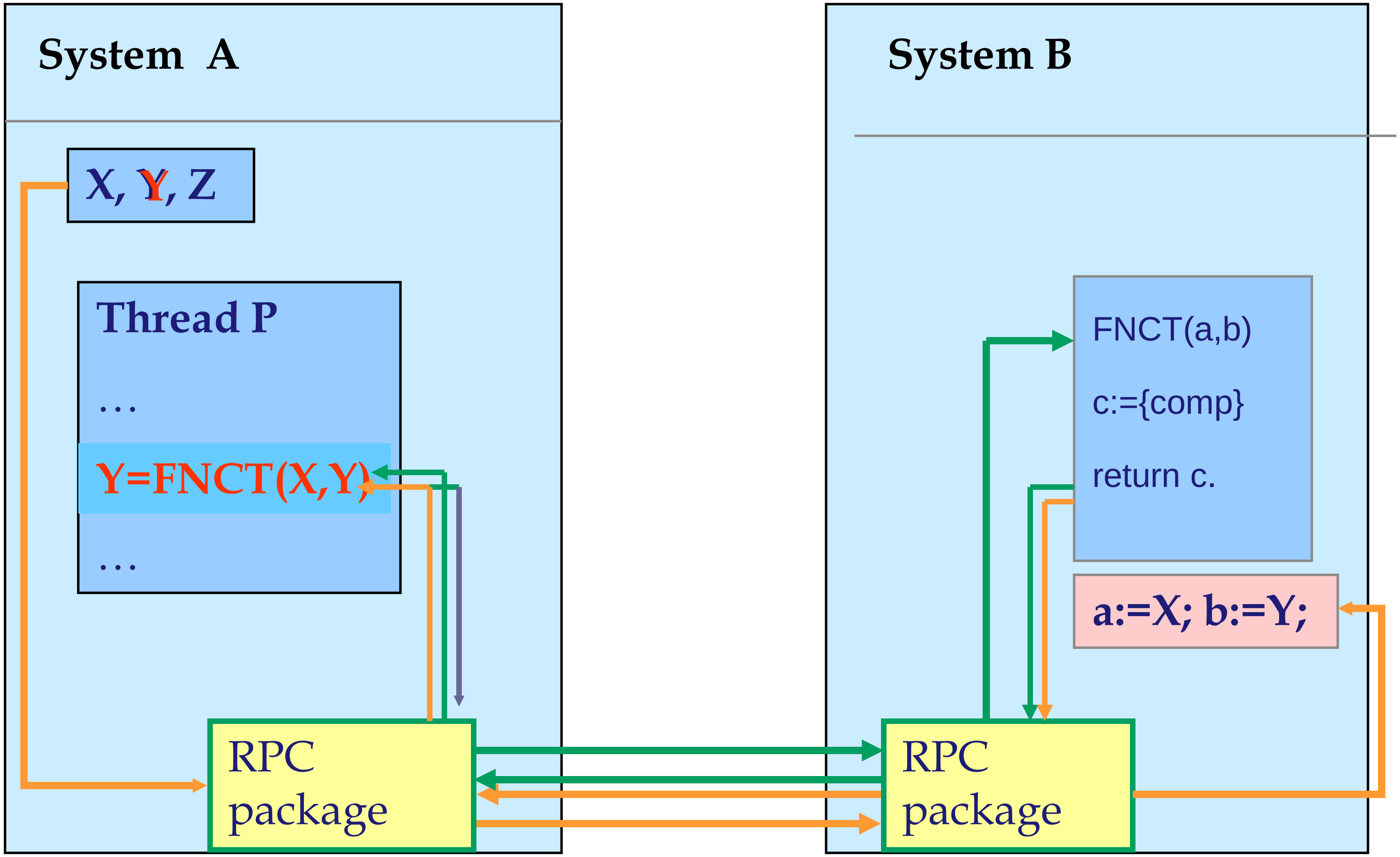
Remote Procedure Calls

- 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 local calls:
  - 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

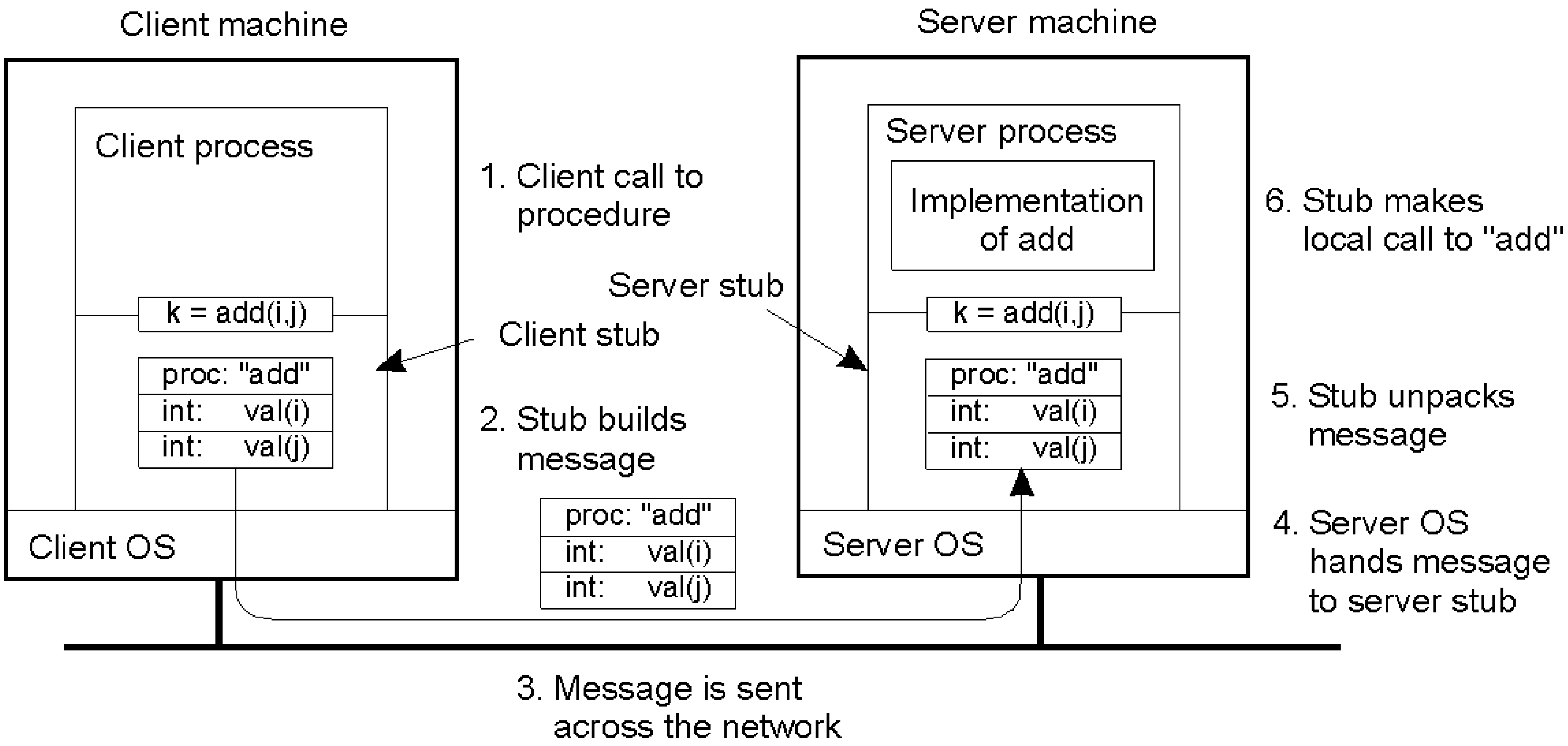
RPC: a Schematic View



Implementation of RPC

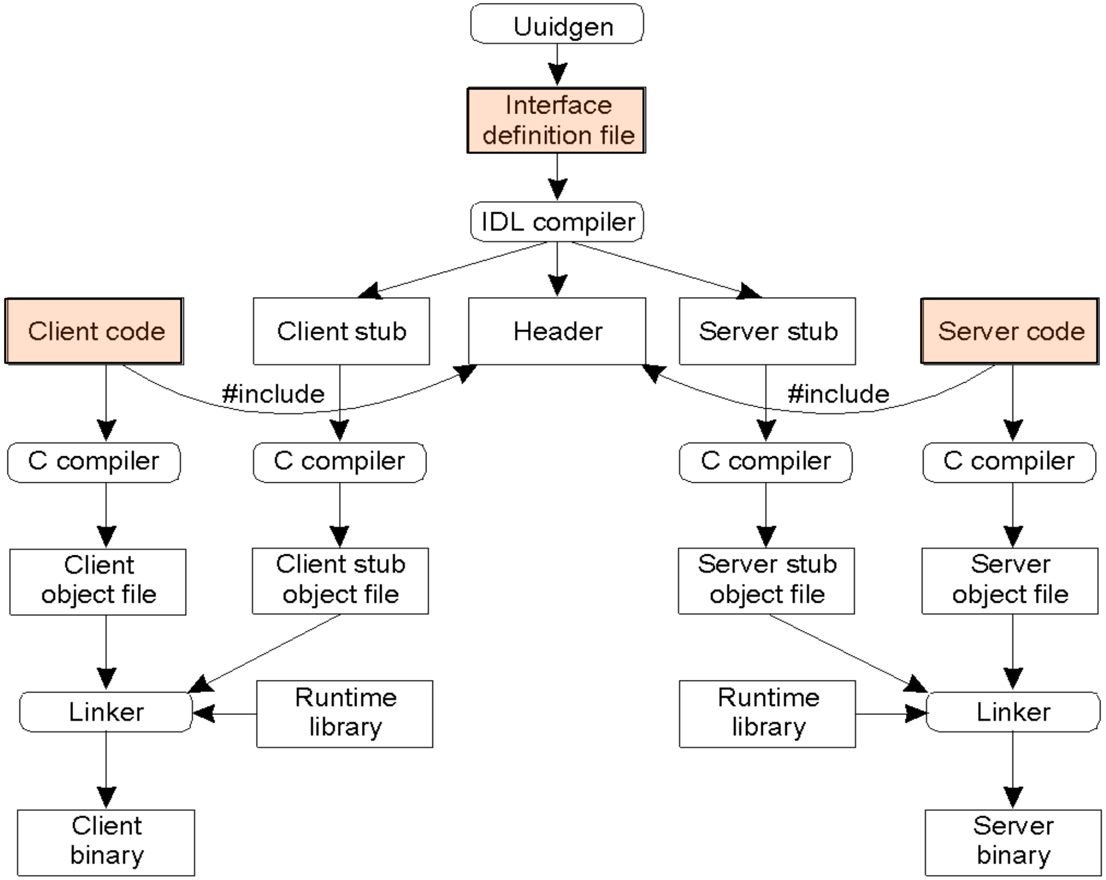
- RPC components:
  - RPC Service (two stubs)
    - interpretation of the service interface
    - packing of parameters for transportation
  - Transportation service: node to node
    - responsible for message passing
    - part of the operating system
- Name service: look up, binding
  - name of procedure,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

Passing Value Parameters



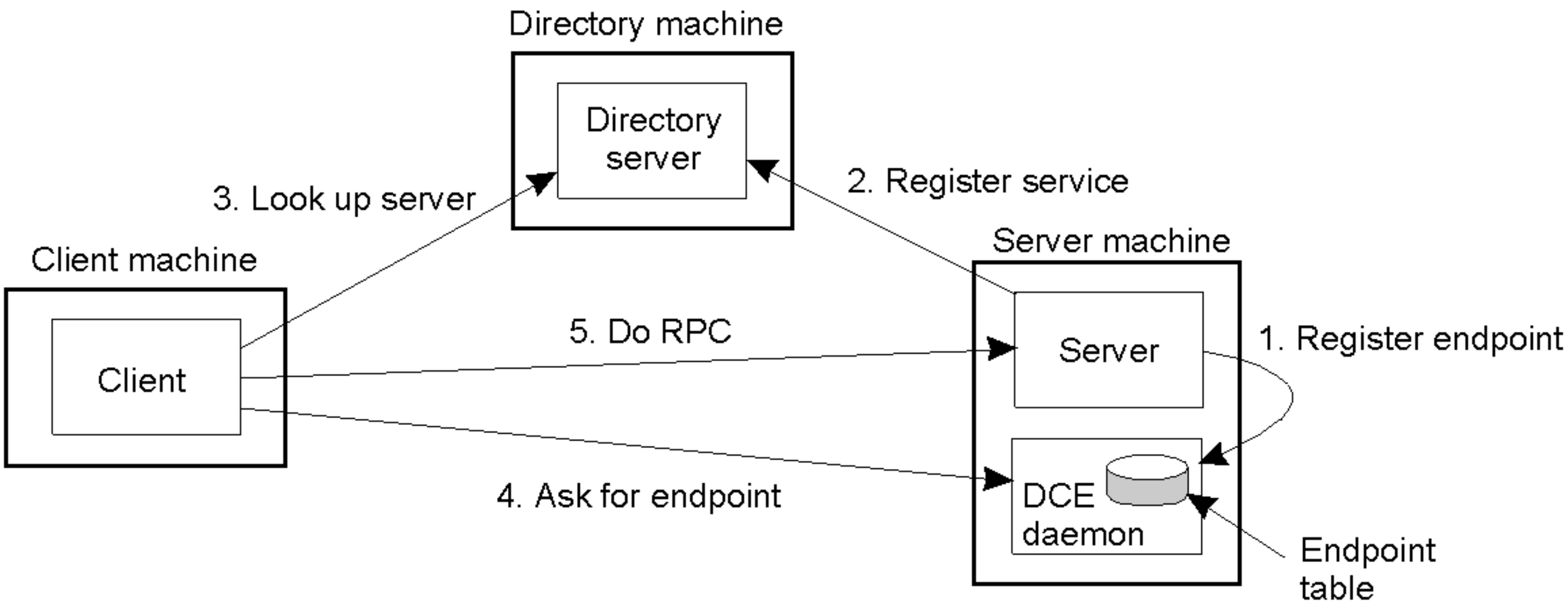
Steps involved in doing remote computation through RPC

Writing a Client and a Server



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

Binding a Client to a Server



Example: 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

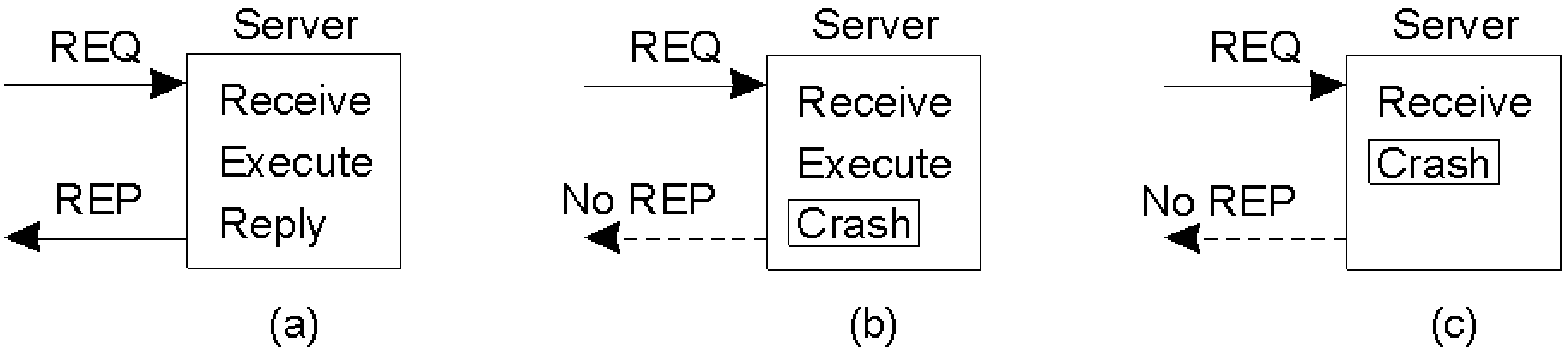
Fault tolerance measures

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. Recall the RPC failure classes:
   - the client unable to locate the server
   - a message is lost (request / reply)
   - the server crashes (before / during / after service)
   - the client crashes

Server Crashes



A server in client-server communication

a) Normal case

b) Crash after execution

c) Crash before execution

E.g.: Printer server crashes (Fig. 8-8)

Client	**Printer Server (“print queue”)**					
	Strategy: Message client, then Print			**Strategy: Print, then Message**		
Client’s request reissue strategy	**MPC**	**MC(P)**	**C(MP)**	**PMC**	**PC(M)**	**C(PM)**
Always (*at-least-once semantics*)	DUP	OK	OK	DUP	DUP	OK
Never (*maybe semantics*)	OK	ZERO	ZERO	OK	OK	ZERO
Only when not ACKed (*depends*)	OK	ZERO	OK	OK	DUP	OK
Only when ACKed (*madness!*)	**DUP**	OK	ZERO	**DUP**	OK	ZERO

Different combinations of client and server strategies in the presence of server crashes (client hears of crash, decides: reissue request / not?)

M: send the completion message
P: tell printer to print text
C: crash
ACK: Receipt of the completion message

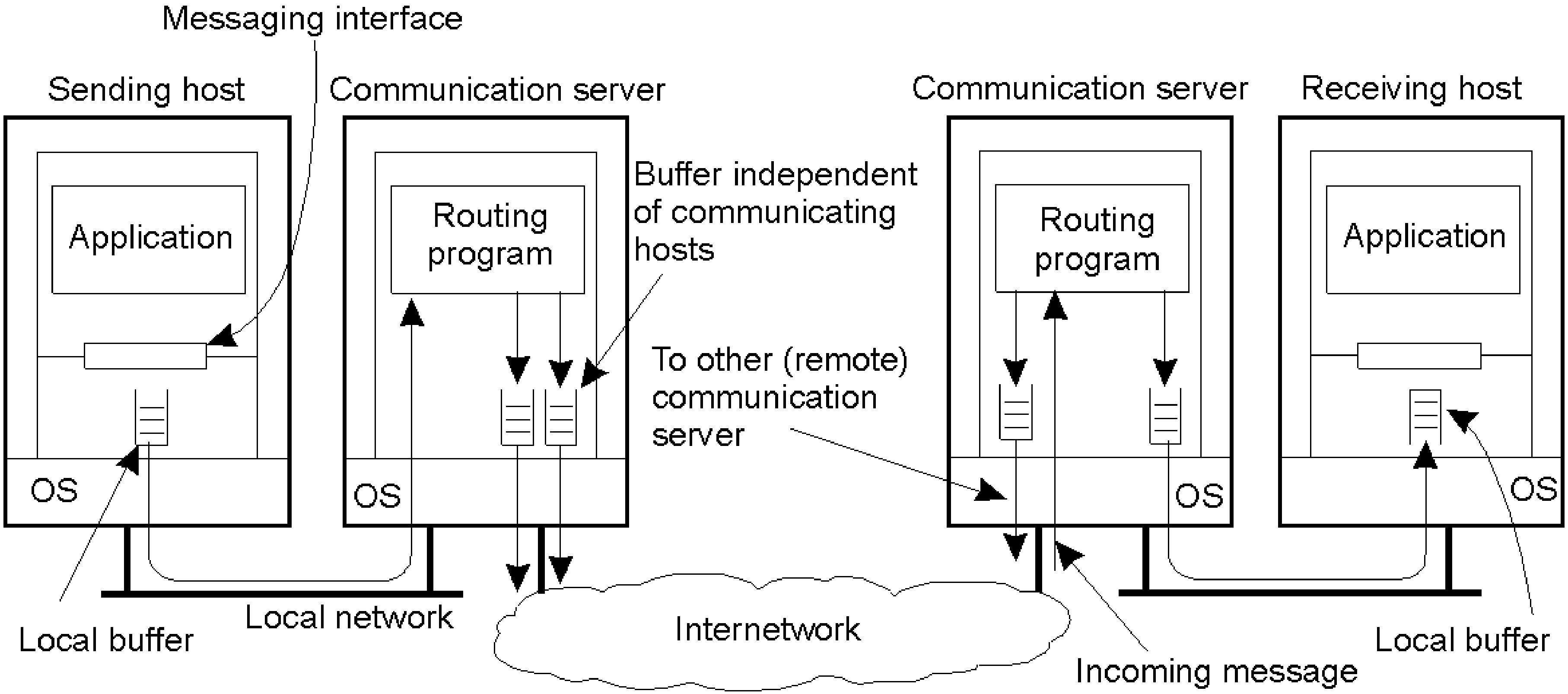
OK = Text printed once
DUP = Text printed twice
ZERO = Text not printed

Client Crashes: No one there to receive a reply

- Orphan: an active computation looking for a non-existing parent

- Solutions
  - extermination: the client stub records all calls,
    after reboot any orphans on record are explicitly killed
  - reincarnation: time is divided into epochs, client reboot =>
    broadcast “new epoch” => servers kill the client’s old requests
  - gentle incarnation: “new epoch” => look for parents, kill real orphans
  - 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

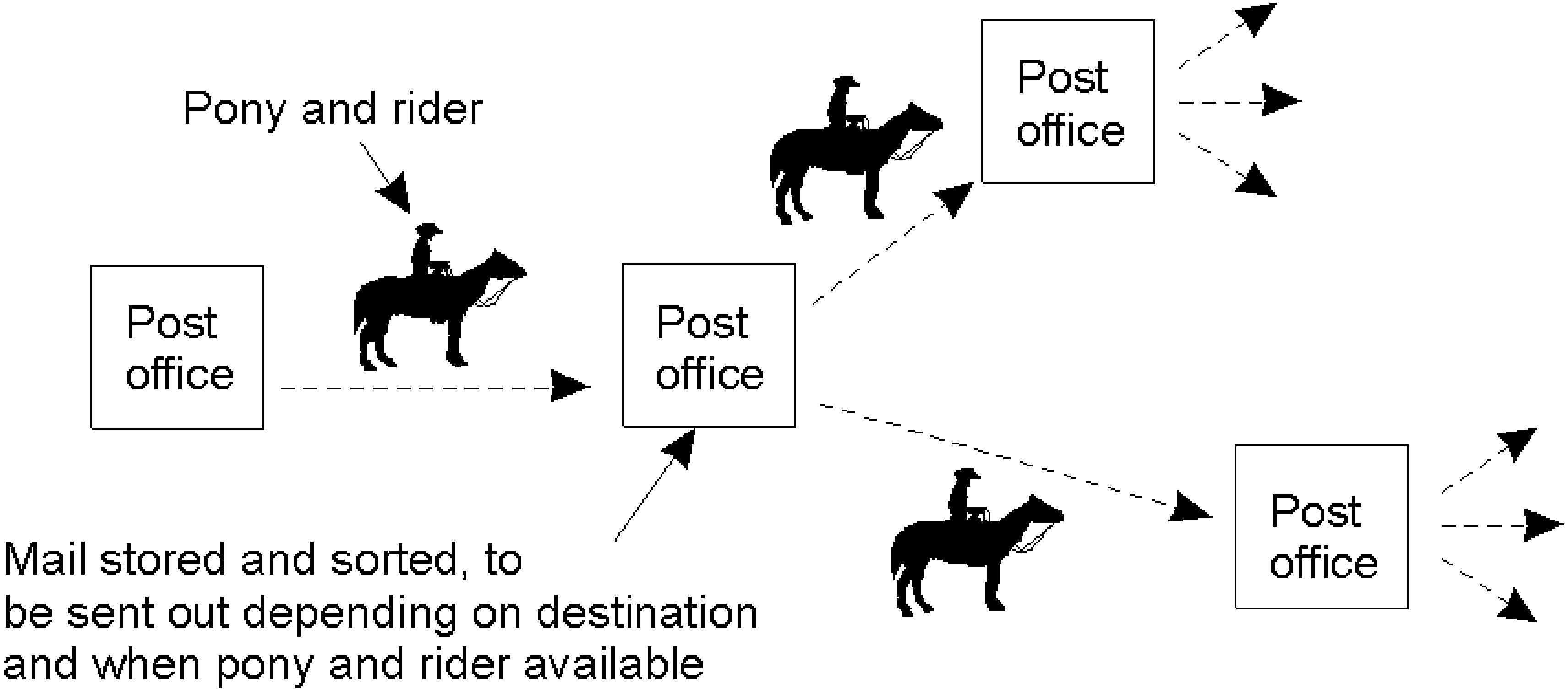


General organization of a communication system in which hosts are connected through a network

Persistent vs. Transient 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)

Persistent Communication – Pony Express Style

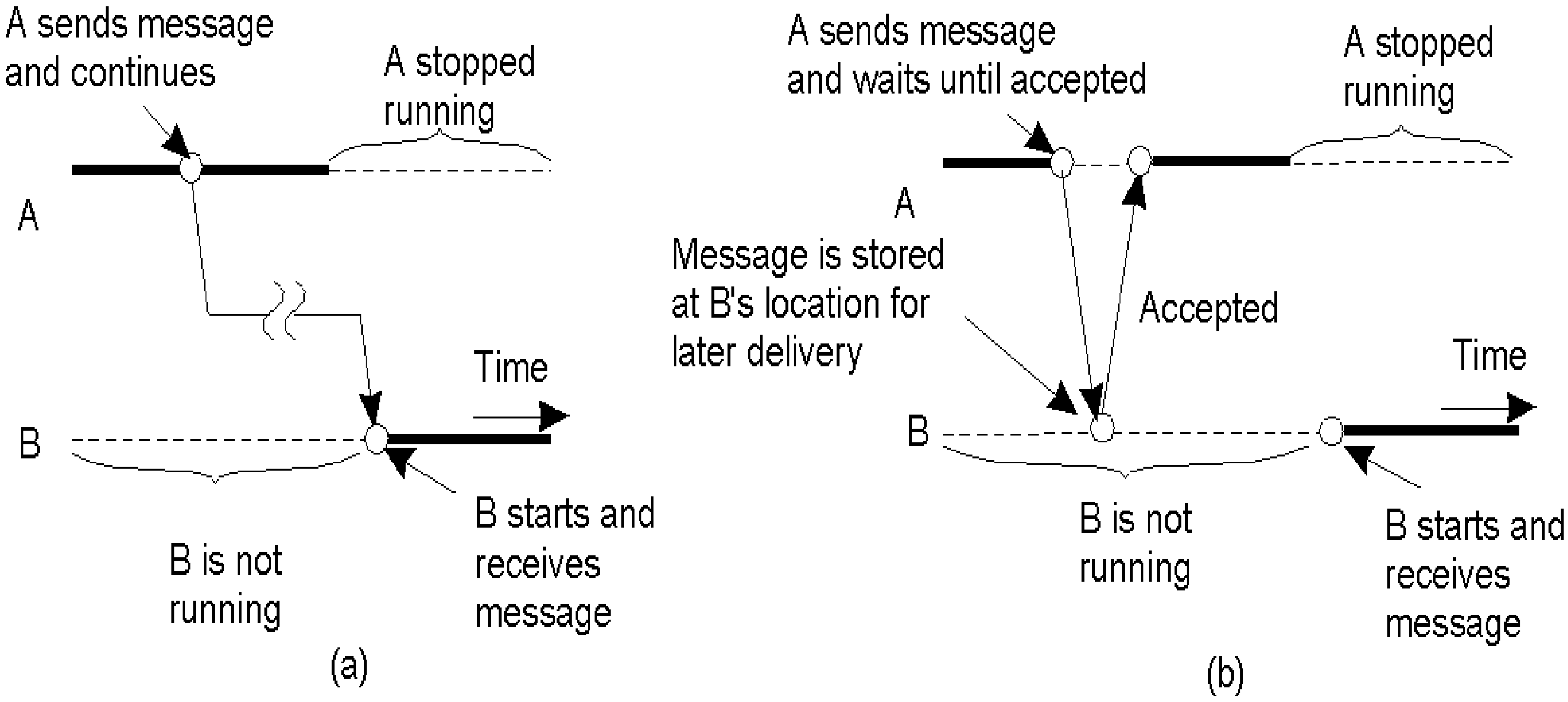

Pony and rider
Post office
Post office
Post office
Post office
Mail stored and sorted, to
be sent out depending on destination
and when pony and rider available

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

Sychronous and Asynchronous Communication

- Asynchronous communication
  - the sender continues immediately after submission; something else takes care of the rest
- Synchronous communication
  - the sender is blocked until
    - the message is received by e.g. middleware to deliver to target application (receipt-based synchrony)
    - the message is delivered to the target (delivery based)
    - the response to it has arrived (response based)

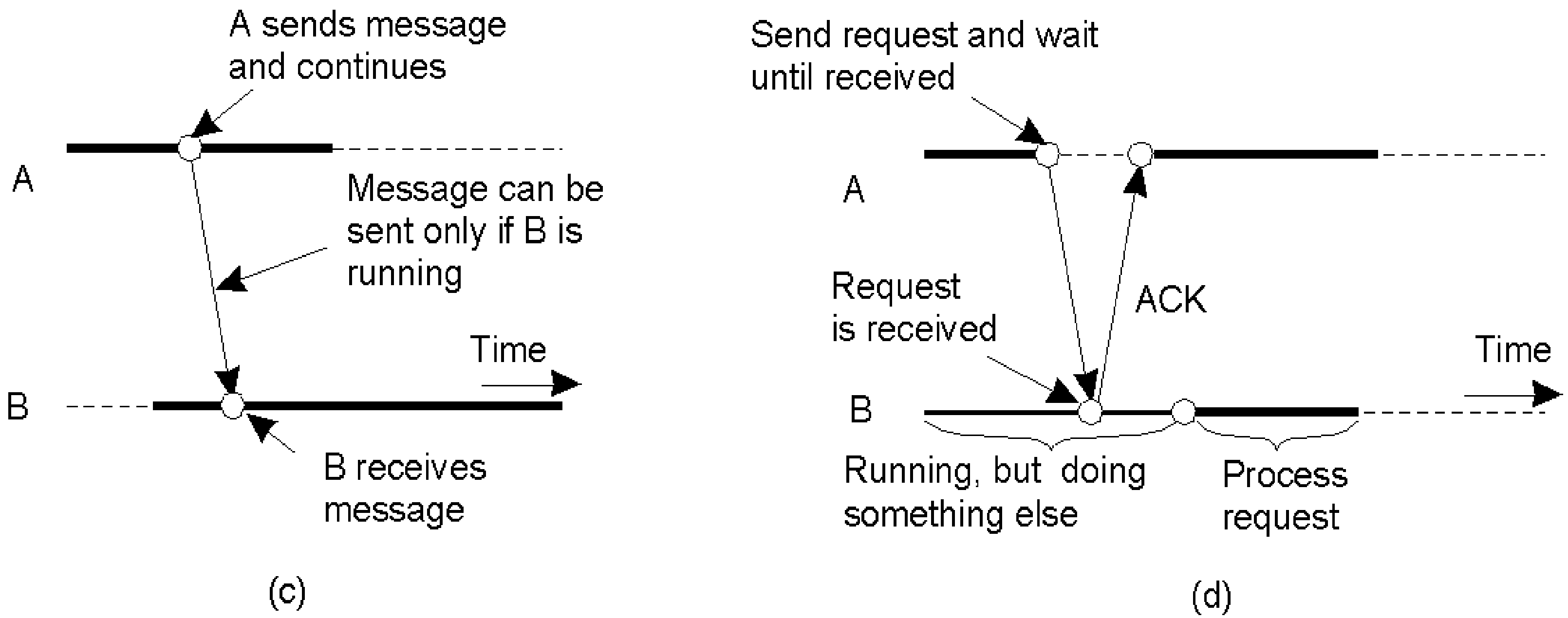
Persistence and Synchronicity in Communication


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

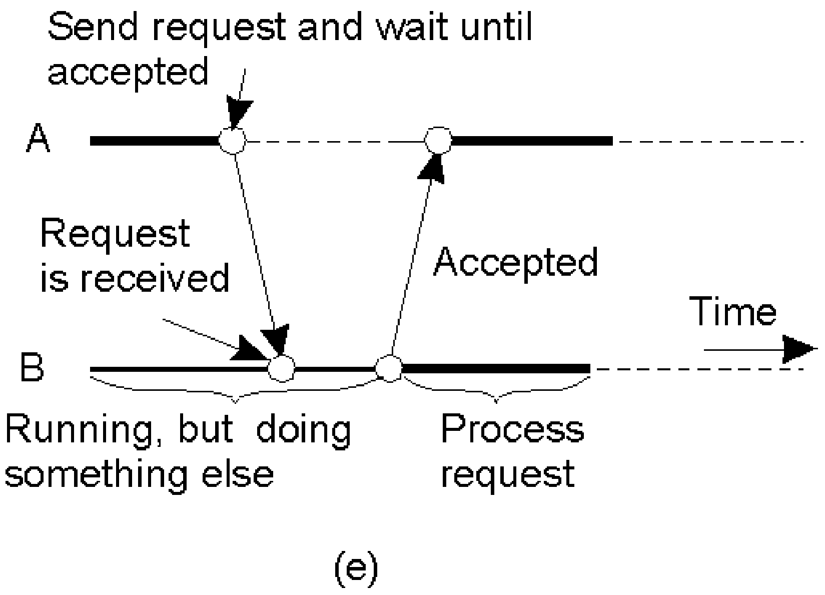
b) Persistent (delivery-based) synchronous communication

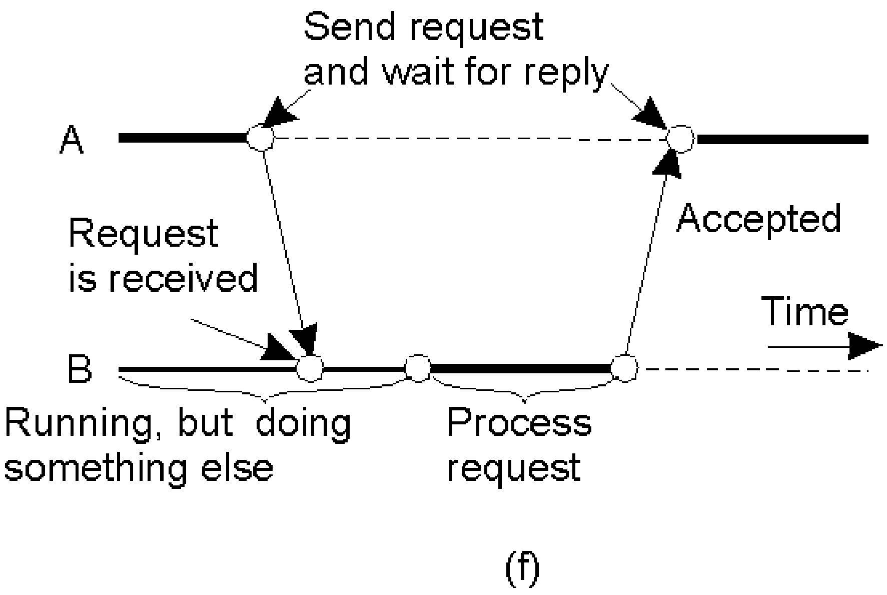
Persistence and Synchronicity in Communication


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


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

Chapter Summary

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