Fault Tolerance

Chapter 7

Lecture 11: Fault tolerance (Chapter 7)

- Goal: dependable systems
- The problem (and opportunity) of *partial* failures
- Failure models
- Detecting and recovering from failures
  - Process failures
  - Communication failures
- Reliable communication
  - Client-server communication
  - Group communication

Basic Concepts

Dependability includes

- Availability: at any instant
- Reliability: over a period of time
- Safety: no catastrophes
- Maintainability: repairability

Terminology

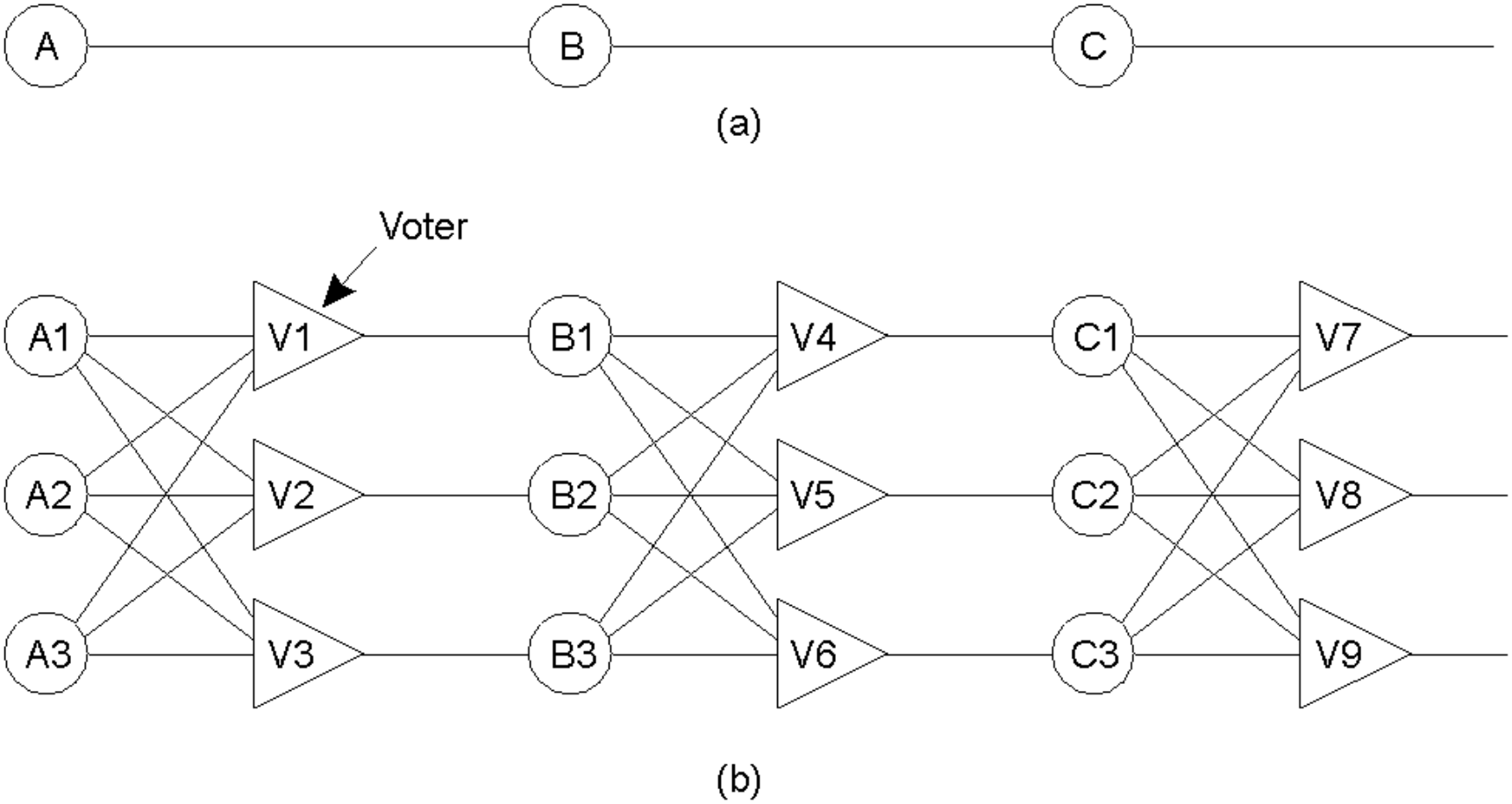
- *Failure*: system cannot provide its promised service
- *Error*: system enters an undesirable state (e.g. message is damaged in transmission)
- *Fault*: cause of an error
  - Transient
  - Intermittent
  - Permanent
- Goal of debugging is to find the faults. When there is a failure, we try to find the errors and then trace to the fault(s)

Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Different types of failures.

Failure Masking by Redundancy

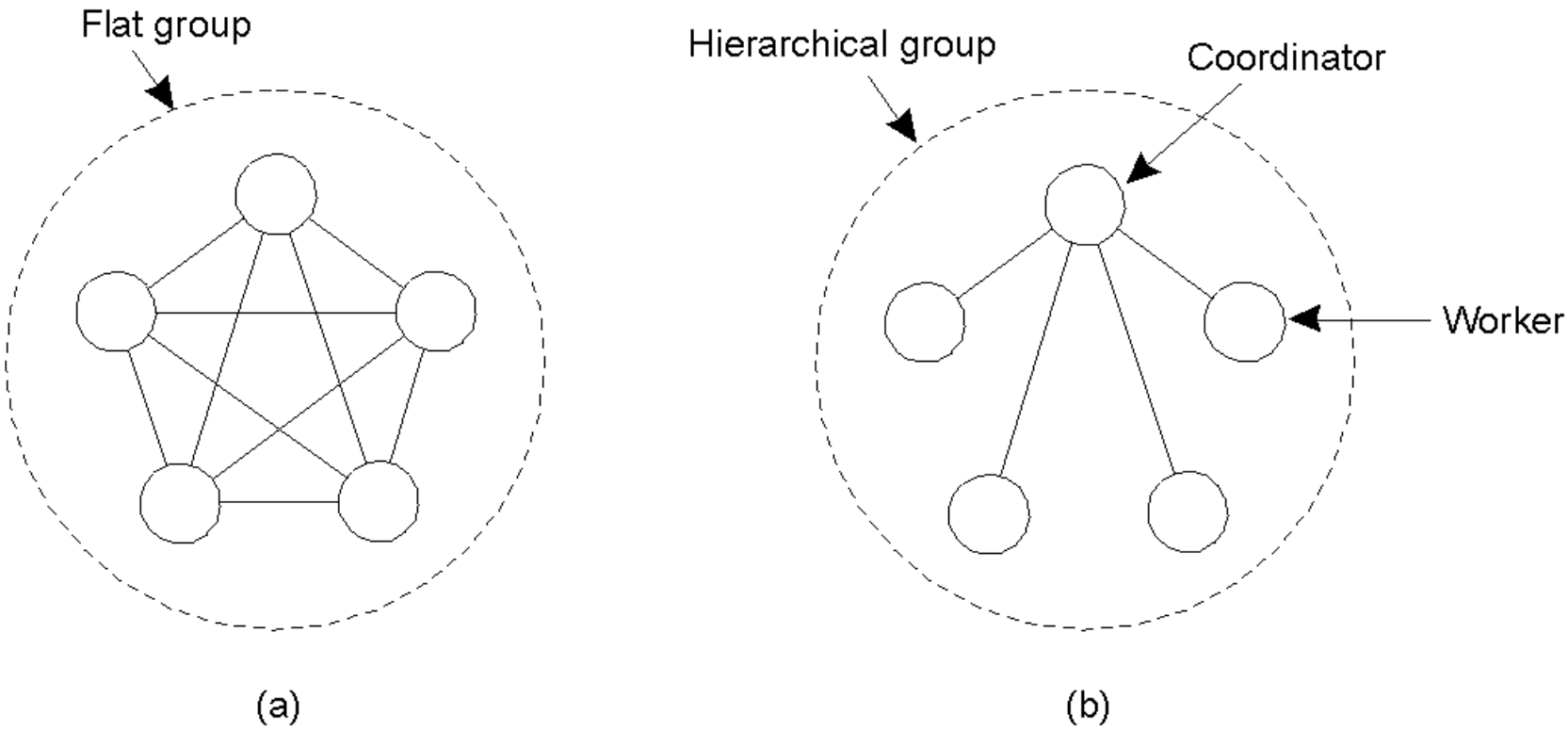


Triple modular redundancy.

Dealing with process failures

- As in hardware, we can introduce redundancy to cope with process failures
- Process groups: replicate a process
  - Addressing
  - Communication
  - Membership
- As long as a sufficient number of processes are present in a group, service can be provided despite faults in some processes. The non-faulty processes must *agree* on the result.

Flat Groups versus Hierarchical Groups



a) Communication in a flat group.

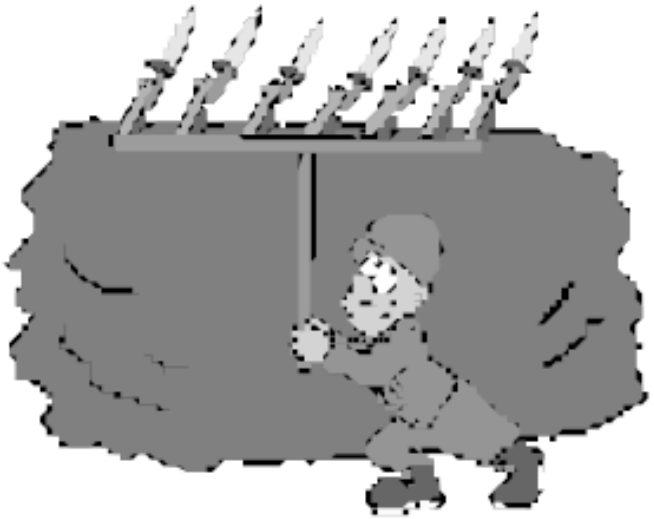
b) Communication in a simple hierarchical group

Agreement in non-faulty systems



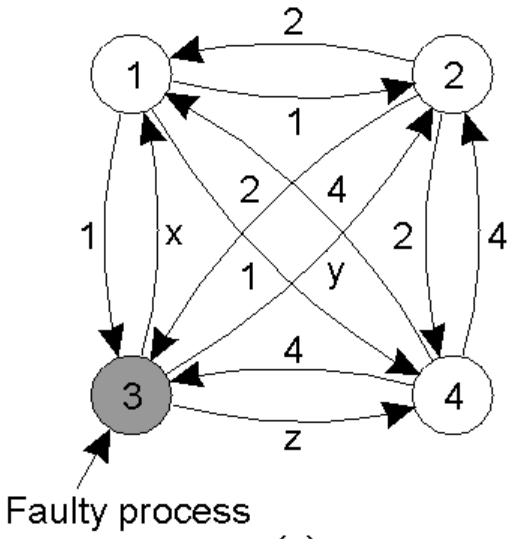
The two-army problem:

1. Sparta and Carthage together can beat Bad guys but not individually. Therefore, they have to decide to attack at exactly the same time.
2. Sparta general sends a message to Carthage general to attack at noon
3. How does he know that Carthage general received the message?



Messenger (unreliable channel)

Agreement in Faulty Systems (1)



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

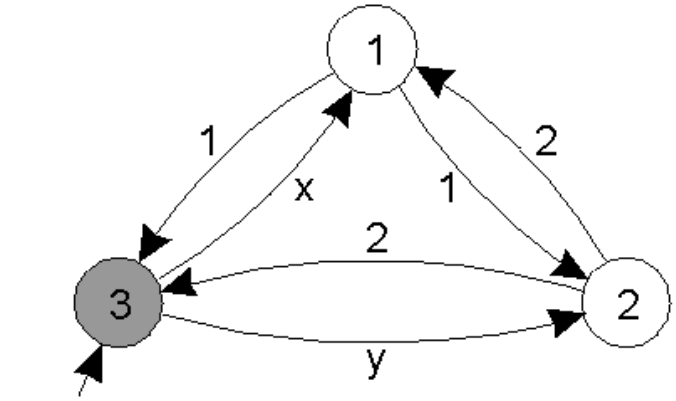
The Byzantine generals problem for 3 loyal generals and1 traitor.

a) The generals announce their troop strengths (in units of 1 kilosoldiers).

b) The vectors that each general assembles based on (a)

c) The vectors that each general receives in step 3.

Agreement in Faulty Systems (2)



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

The same as in previous slide, except now with 2 loyal generals and one traitor.

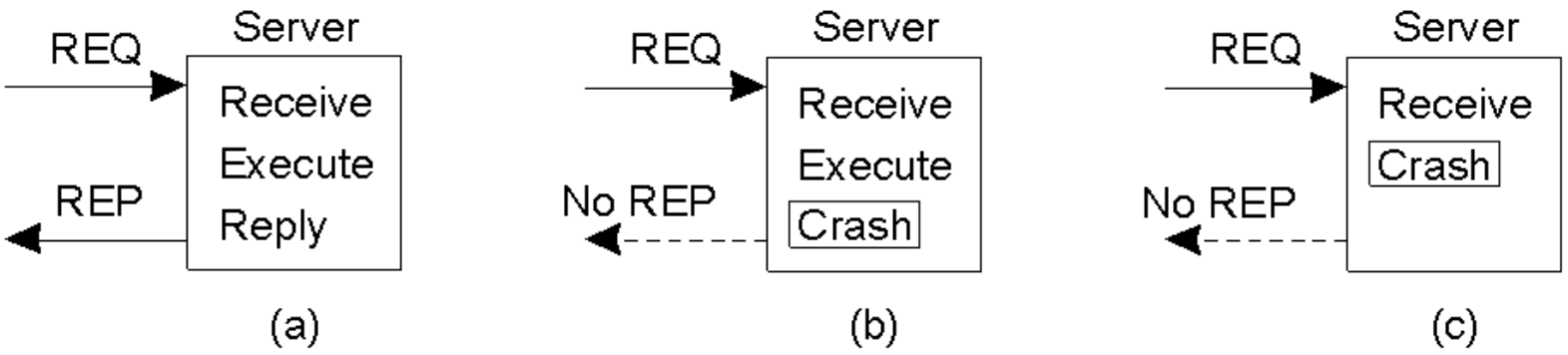
Reliable Communication Client/Server

Potential failures in client-server communication

- Client cannot locate server
- Client request is lost
- Server crashes after receiving request
- Reply message is lost
- Client crashes after sending request

Terima kasih

Lost Request Messages Server Crashes (1)



A server in client-server communication

a) Normal case

b) Crash after execution

c) Crash before execution

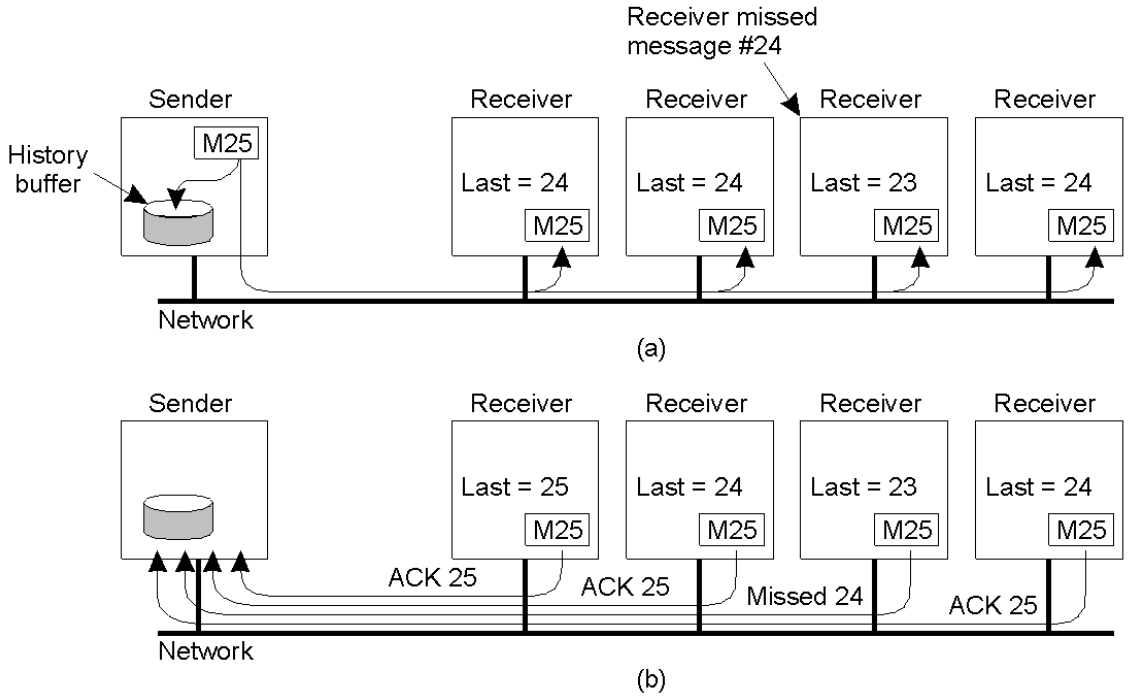
Server Crashes (2)

Client	Server					
	Strategy M -> P			Strategy P -> M		
Reissue strategy	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Always	DUP	OK	OK	DUP	DUP	OK
Never	OK	ZERO	ZERO	OK	OK	ZERO
Only when ACKed	DUP	OK	ZERO	DUP	OK	ZERO
Only when not ACKed	OK	ZERO	OK	OK	DUP	OK

Different combinations of client and server strategies in the presence of server crashes.

Communication Group

Basic Reliable-Multicasting Schemes (1)



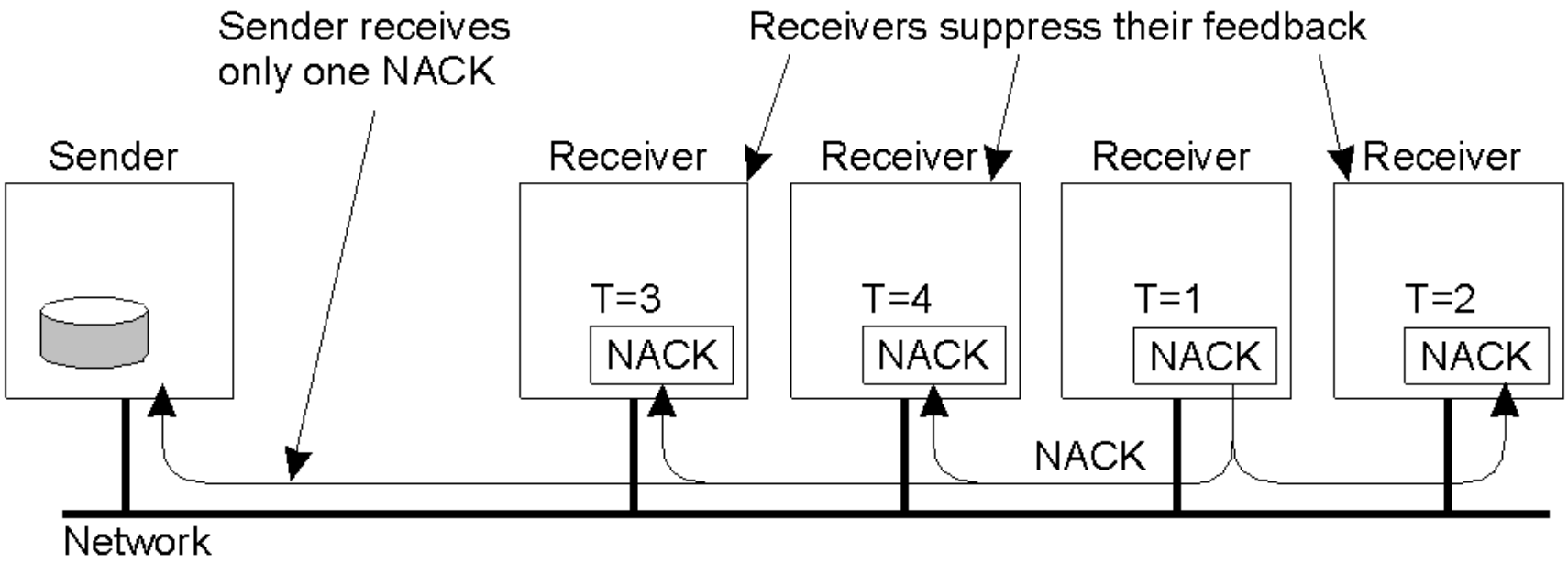
A simple solution to reliable multicasting when all receivers are known and are assumed not to fail

a) Message transmission

b) Reporting feedback

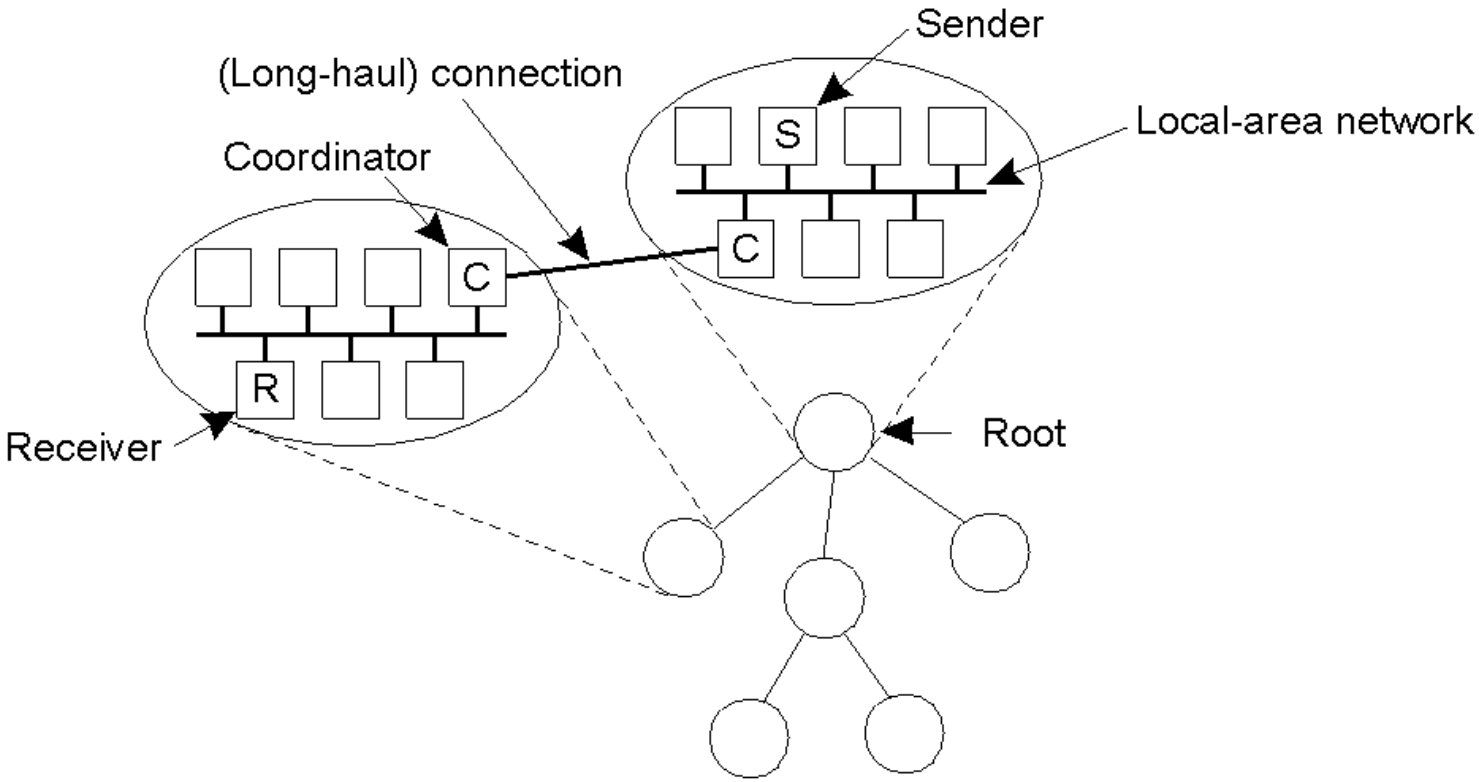
Scalability Reliable-Multicasting Schemes (2)

Nonhierarchical Feedback Control (1)



Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.

Hierarchical Feedback Control (2)

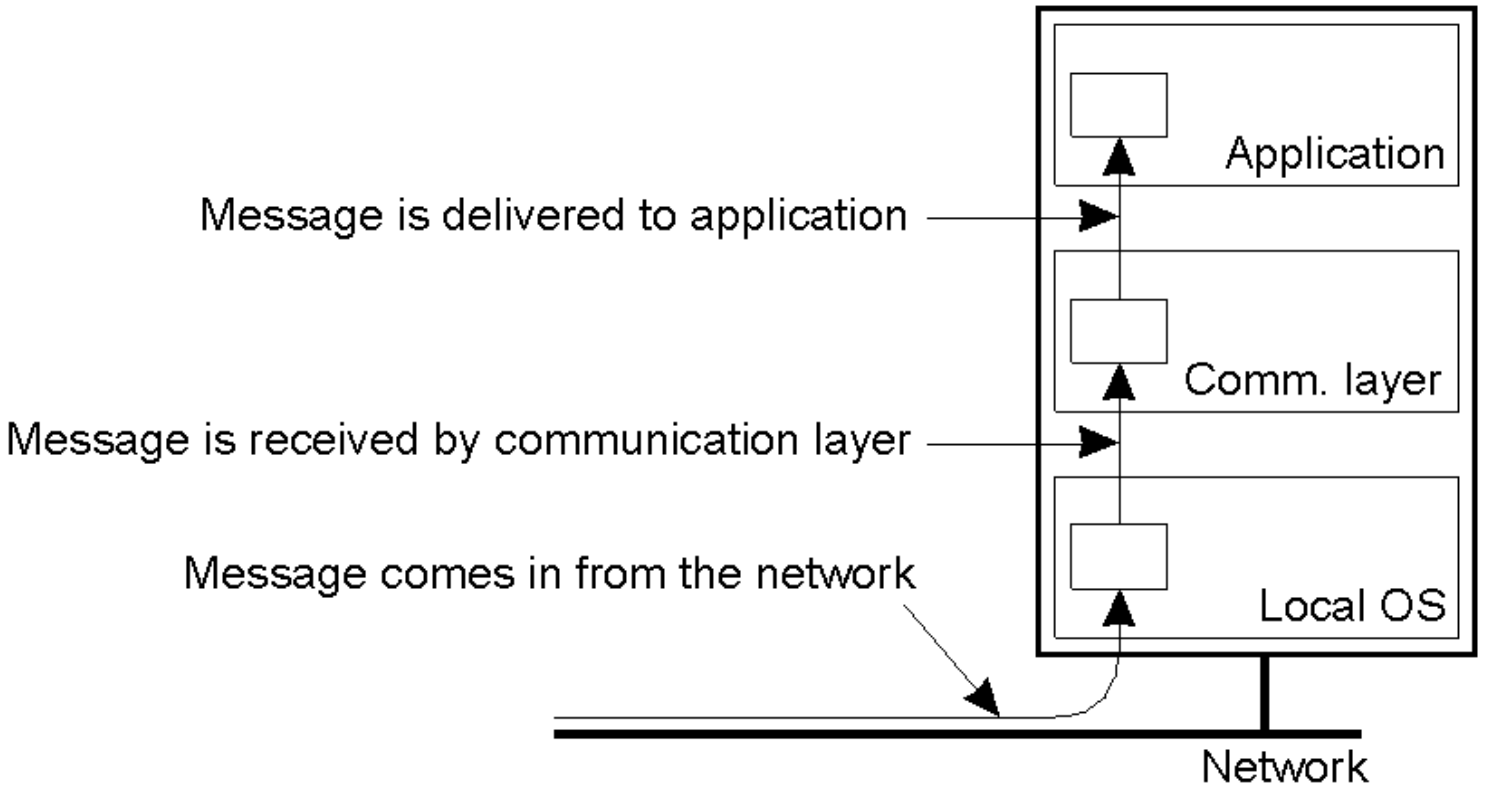


The essence of hierarchical reliable multicasting.

a) Each local coordinator forwards the message to its children.

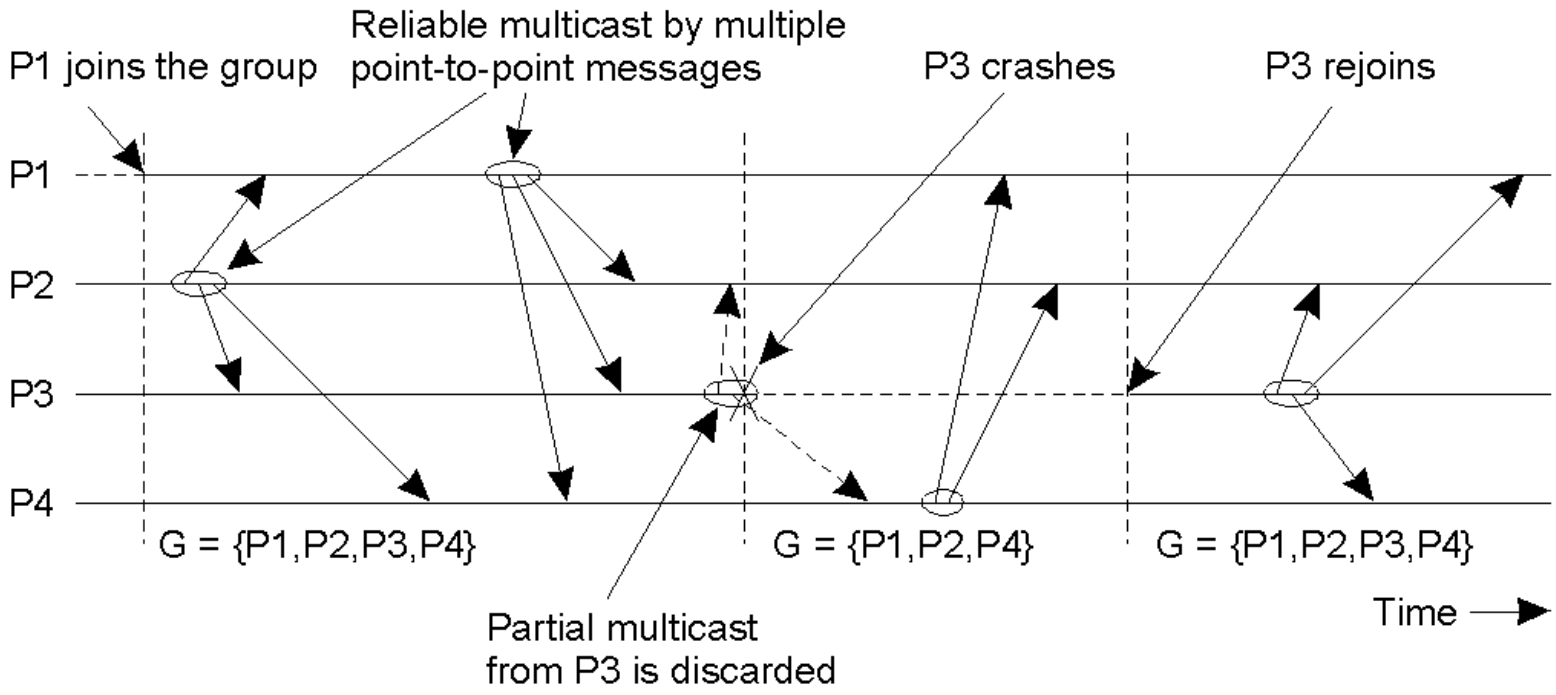
b) A local coordinator handles retransmission requests.

Atomic Multicast (3)
Virtual Synchrony (1)



The logical organization of a distributed system to distinguish between message receipt and message delivery

Virtual Synchrony (2)



The principle of virtual synchronous multicast.

Message Ordering (1)

Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

Three communicating processes in the same group. The ordering of events per process is shown along the vertical axis.

Message Ordering (2)

Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

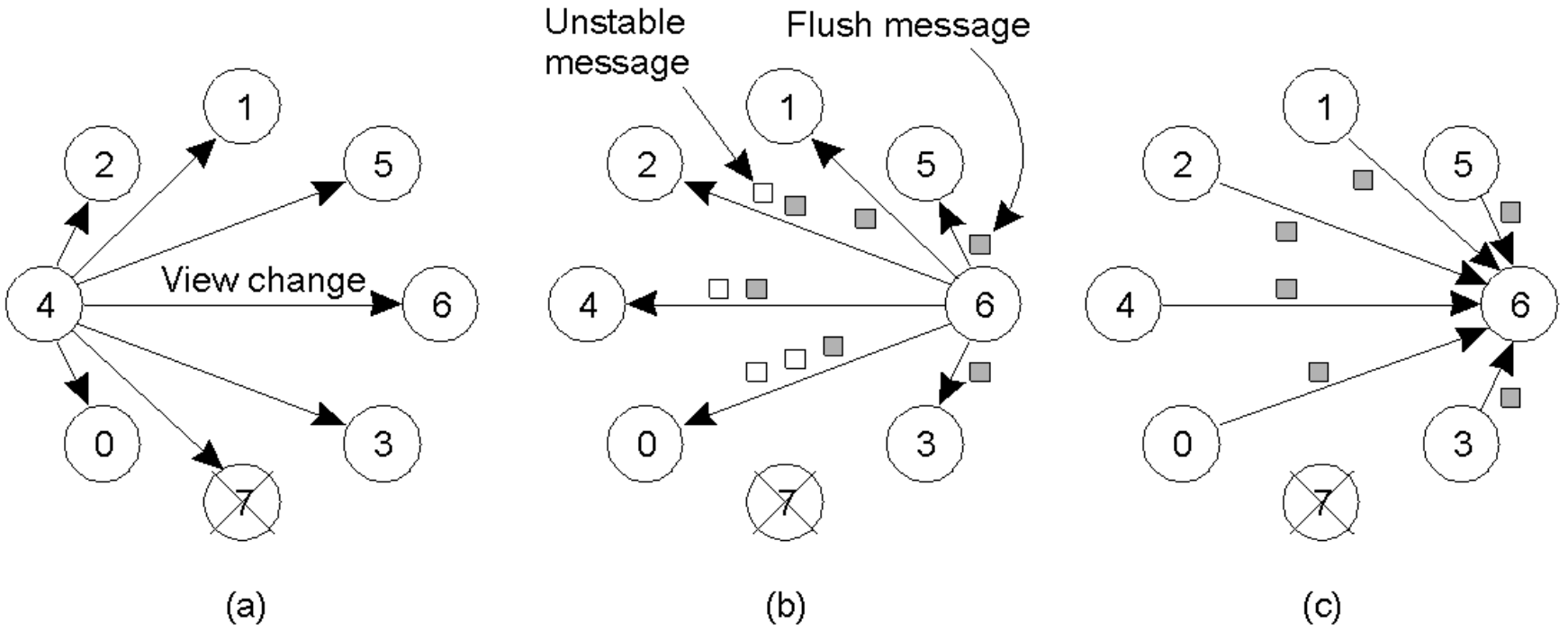
Four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting

Implementing Virtual Synchrony (1)

Multicast	Basic Message Ordering	Total-ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

Six different versions of virtually synchronous reliable multicasting.

Implementing Virtual Synchrony (2)

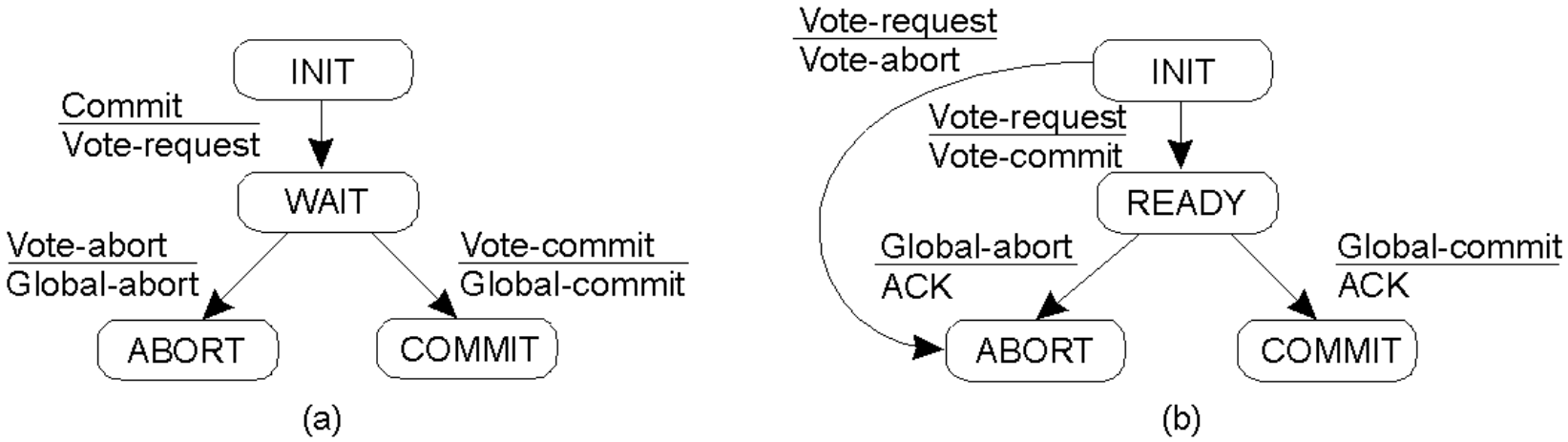


a) Process 4 notices that process 7 has crashed, sends a view change

b) Process 6 sends out all its unstable messages, followed by a flush message

c) Process 6 installs the new view when it has received a flush message from everyone else

Distributed Commit
Two-Phase Commit (1)



a) The finite state machine for the coordinator in 2PC.

b) The finite state machine for a participant.

Two-Phase Commit (2)

State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

Actions taken by a participant *P* when residing in state *READY* and having contacted another participant *Q*.

Two-Phase Commit (3)

actions by coordinator:

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        while GLOBAL_ABORT to local log;
        multicast  GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
    write GLOBAL_COMMIT to local log;
    multicast GLOBAL_COMMIT to all participants;
} else {
    write GLOBAL_ABORT  to local log;
    multicast GLOBAL_ABORT to all participants;
}
```

Outline of the steps taken by the coordinator in a two phase commit protocol

Two-Phase Commit (4)

Steps taken by participant process in 2PC.

actions by participant:

```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
    write VOTE_ABORT to local log;
    exit;
}
if participant votes COMMIT {
    write VOTE_COMMIT to local log;
    send VOTE_COMMIT to coordinator;
    wait for DECISION from coordinator;
    if timeout {
        multicast DECISION_REQUEST to other participants;
        wait until DECISION is received; /* remain blocked */
        write DECISION to local log;
    }
    if DECISION == GLOBAL_COMMIT
        write GLOBAL_COMMIT to local log;
    else if DECISION == GLOBAL_ABORT
        write GLOBAL_ABORT to local log;
} else {
    write VOTE_ABORT to local log;
    send  VOTE ABORT to coordinator;
}
```

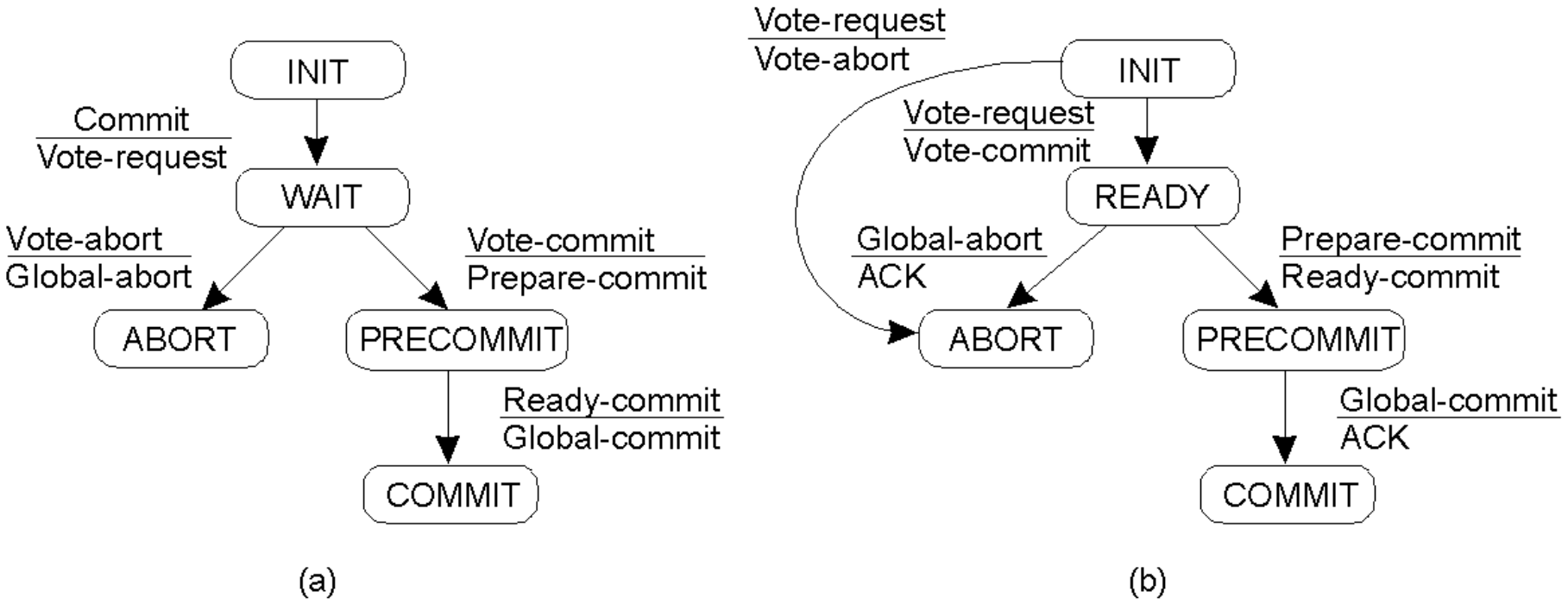
Two-Phase Commit (5)

actions for handling decision requests: /* executed by separate thread */

```
while true {
    wait until any incoming DECISION_REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip;  /* participant remains blocked */
```

Steps taken for handling incoming decision requests.

Three-Phase Commit

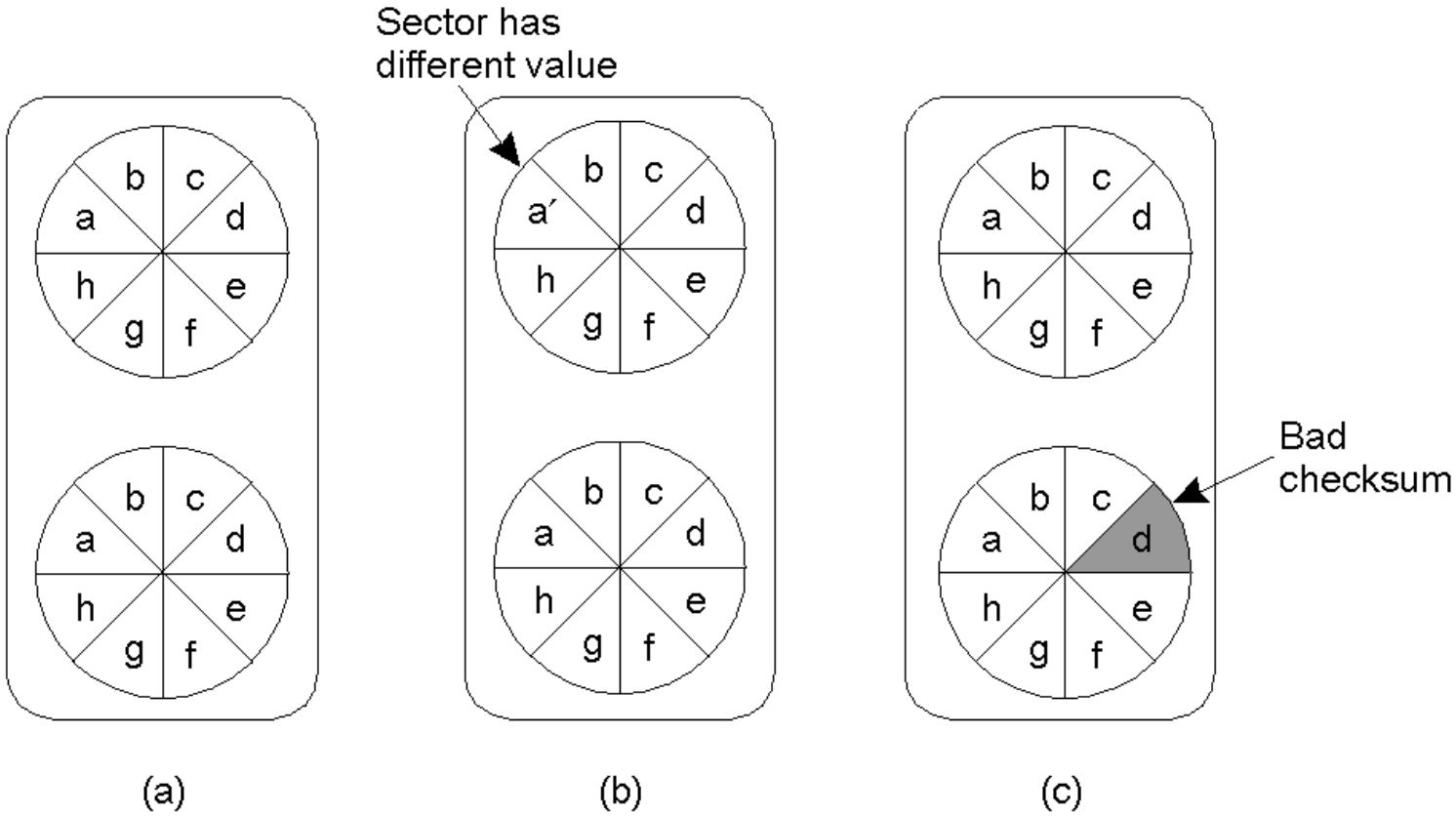


a) Finite state machine for the coordinator in 3PC

b) Finite state machine for a participant

RECOVERY

Recovery Stable Storage

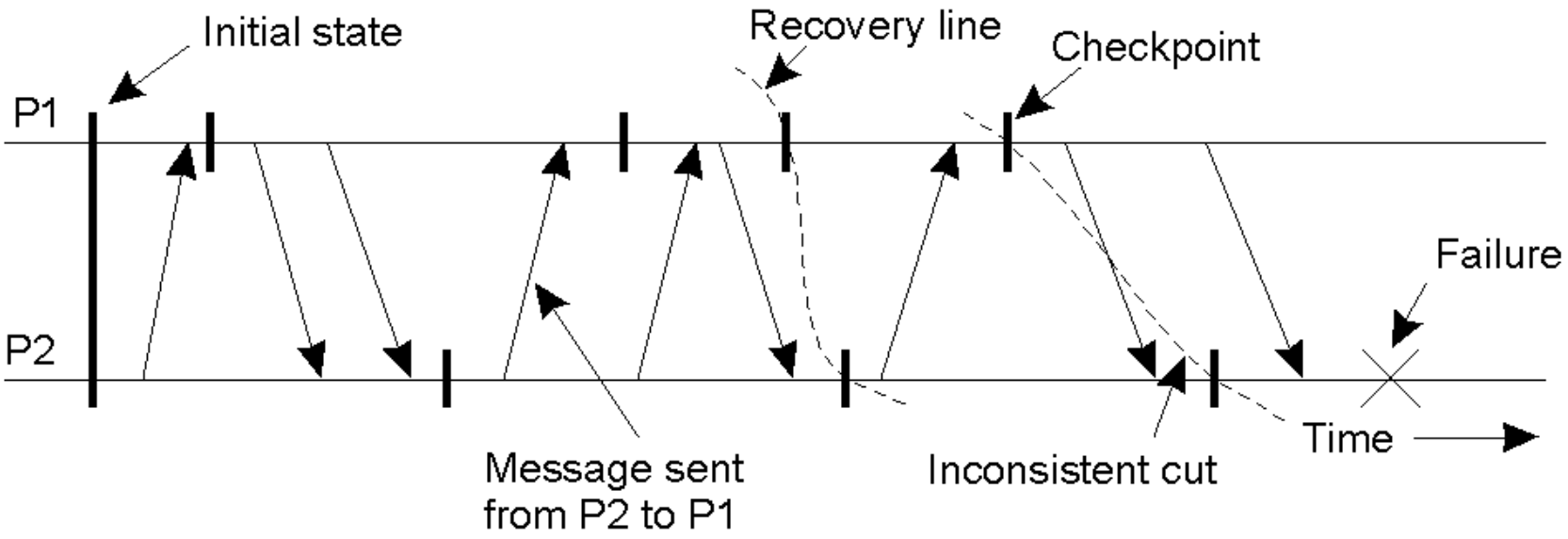


a) Stable Storage

b) Crash after drive 1 is updated

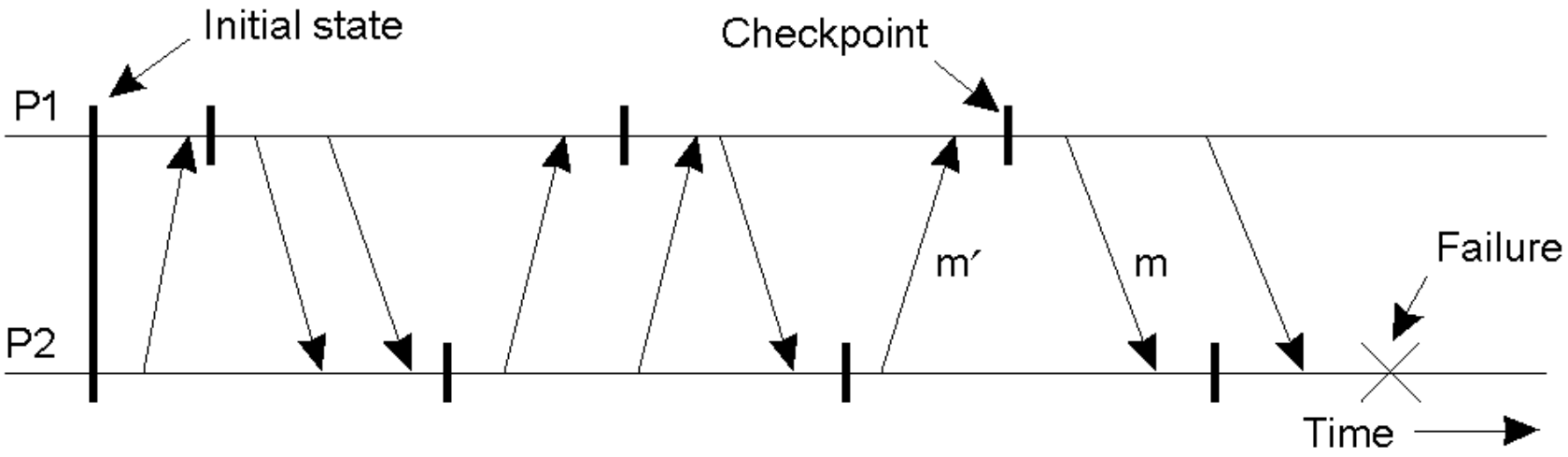
c) Bad spot

Checkpointing



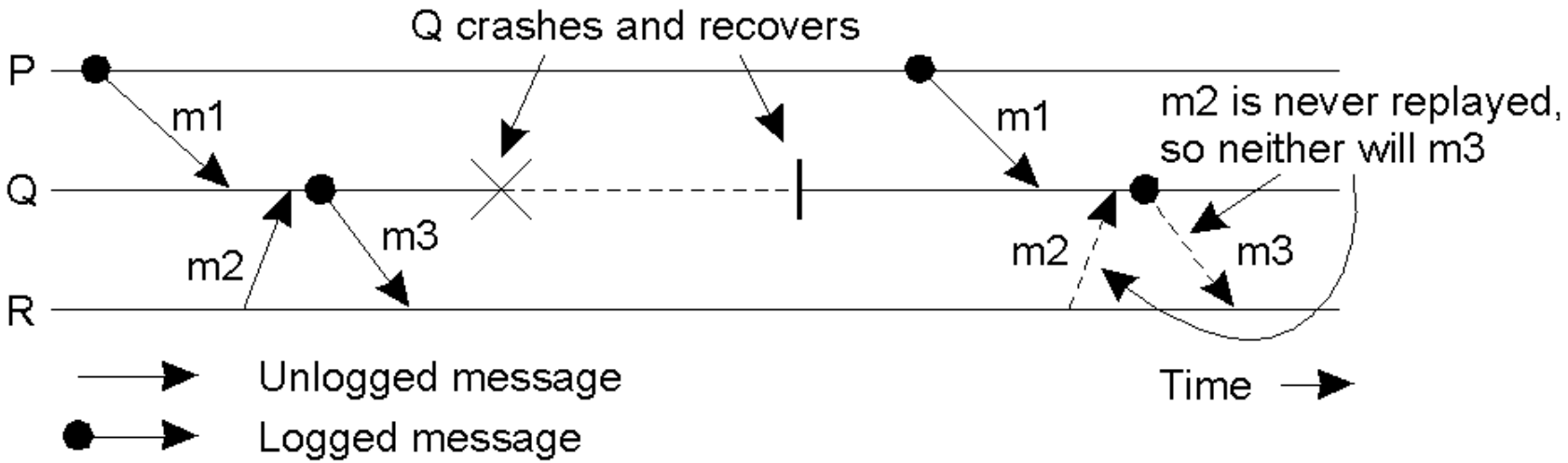
A recovery line.

Independent Checkpointing



The domino effect.

Message Logging



Incorrect replay of messages after recovery,
leading to an orphan process.

Summary

- Distributed systems can suffer partial failures
- Distributed systems can provide fault-tolerance
- Faults can be due to process failures or communication failures
- Process replication (process groups) can help deal with process failures
- Reliable communication can be built on top of unreliable communication mechanisms
- Reliable group communication supports the construction of fault-tolerant systems