

# Fault Tolerance

## Chapter 7

# Basic Concepts

## Dependability Includes

- Availability
- Reliability
- Safety
- Maintainability

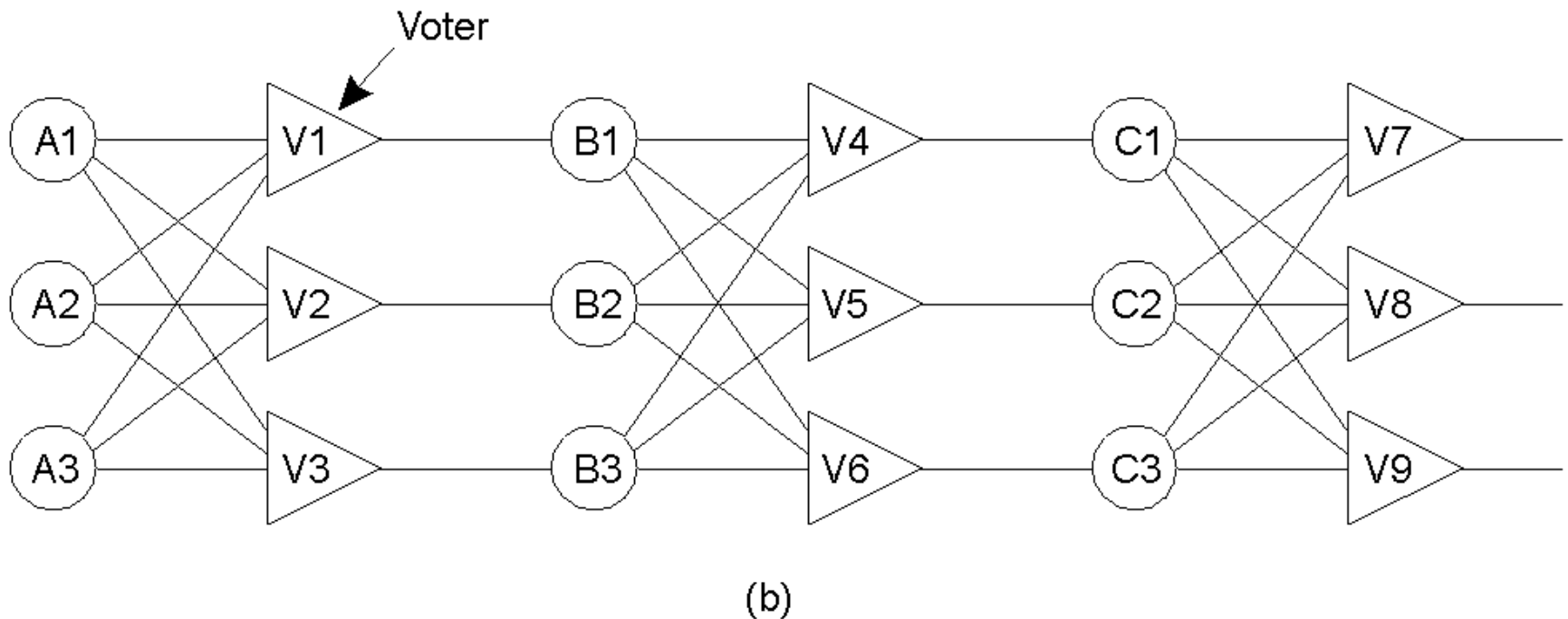
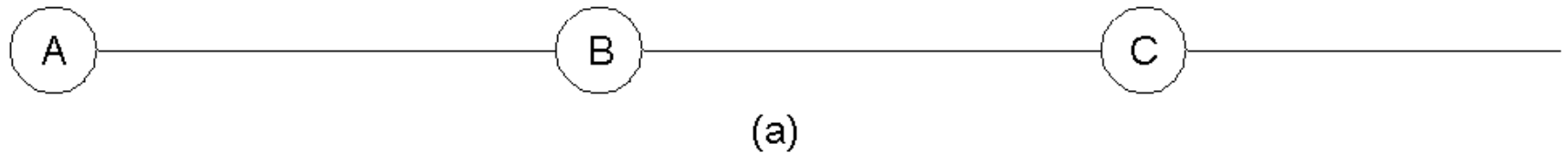
# Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure <i>Receive omission</i> <i>Send omission</i>	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 <i>Value failure</i> <i>State transition failure</i>	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.

What about “fail-stop,” “fail-silent,” and “fail-safe”?

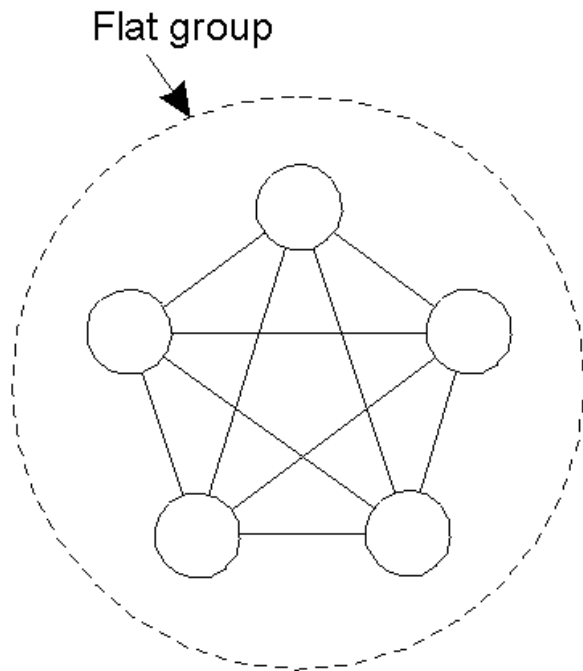
# Failure Masking by Redundancy



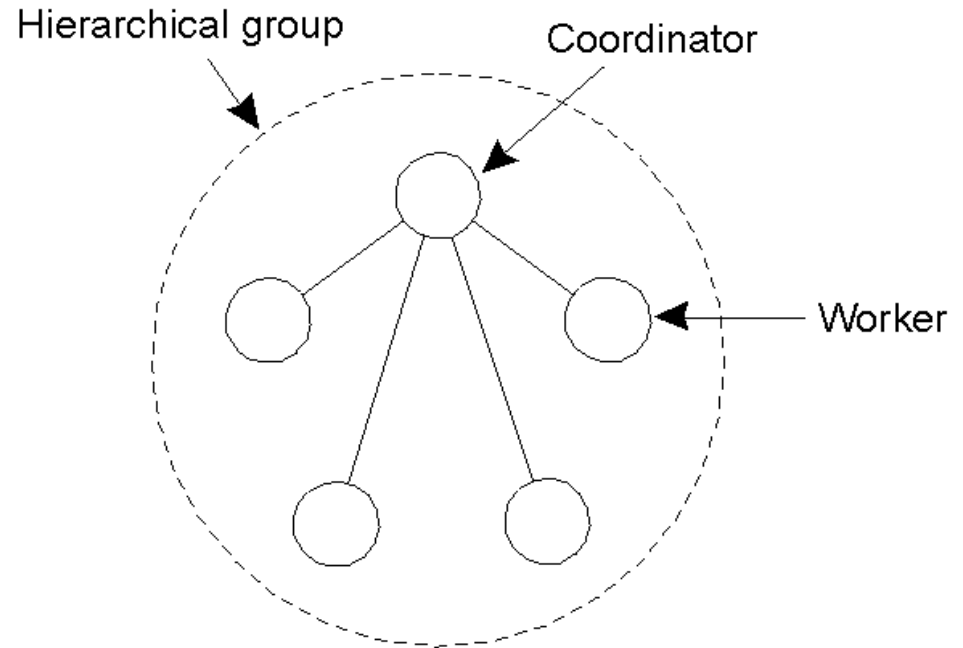
Triple modular redundancy.

Q: If we have three voters, why would we need three voting booths?

# Flat Groups versus Hierarchical Groups



(a)

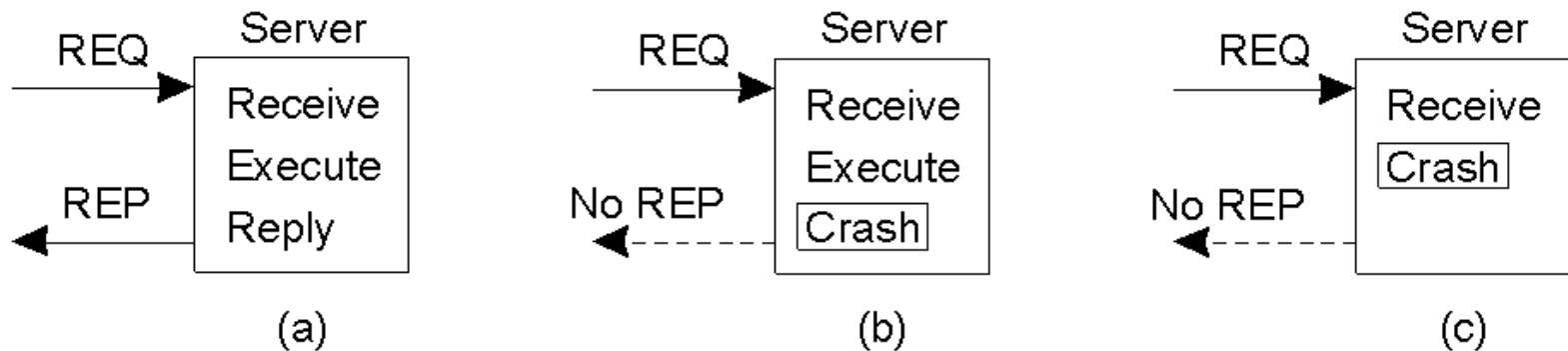


(b)

- a) Communication in a flat group.
- b) Communication in a simple hierarchical group

# 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)**

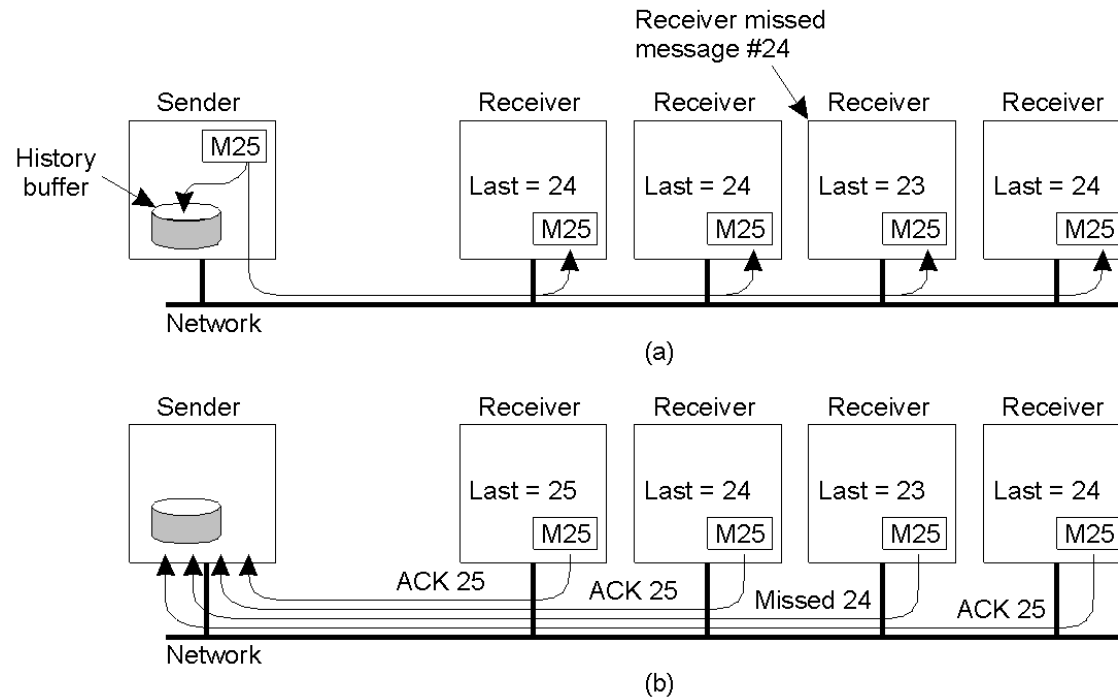
Always
Never
Only when ACKed
Only when not ACKed

DUP	OK	OK
OK	ZERO	ZERO
DUP	OK	ZERO
OK	ZERO	OK

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.

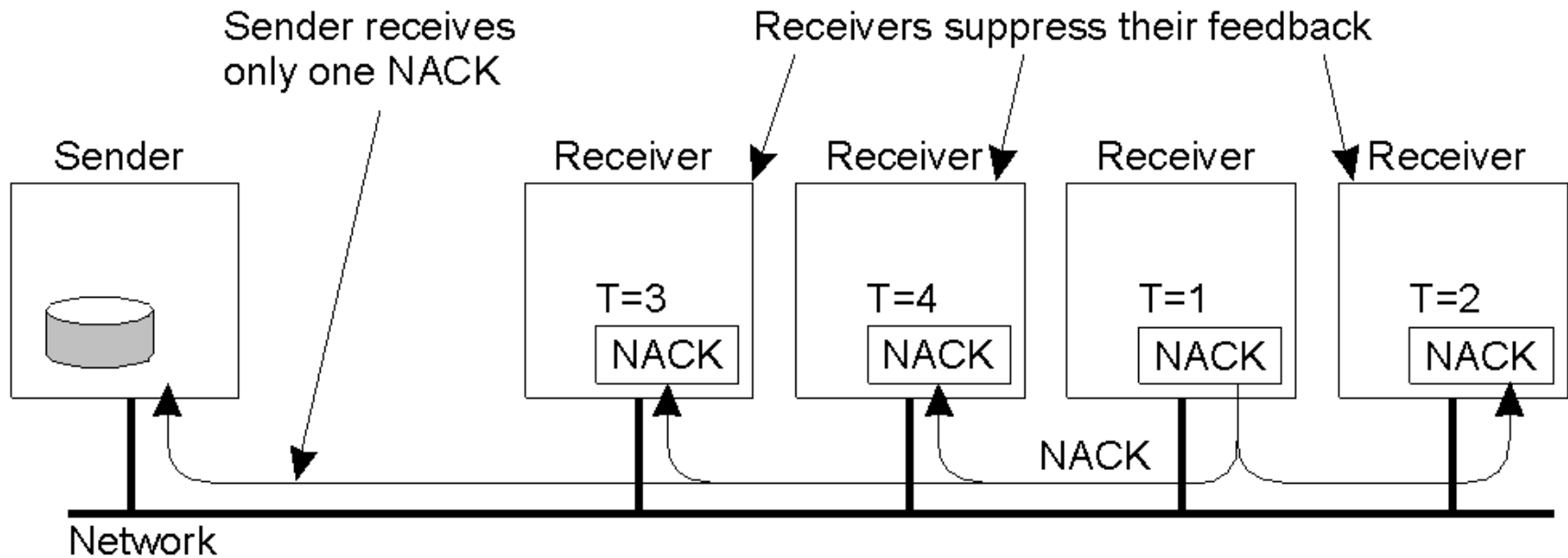
# Basic Reliable-Multicasting Schemes



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

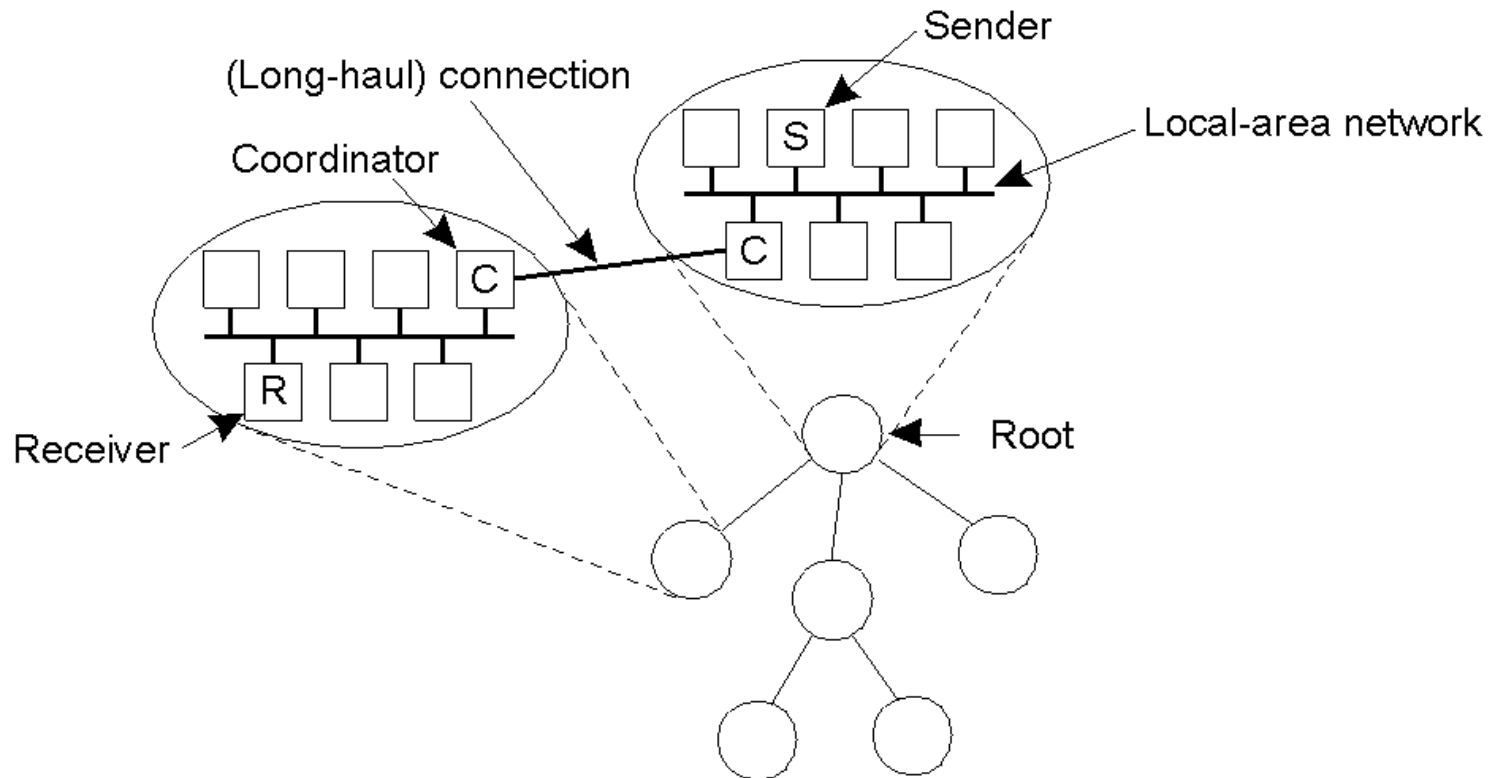
- a) Message transmission
- b) Reporting feedback

# Nonhierarchical Feedback Control



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

# Hierarchical Feedback Control

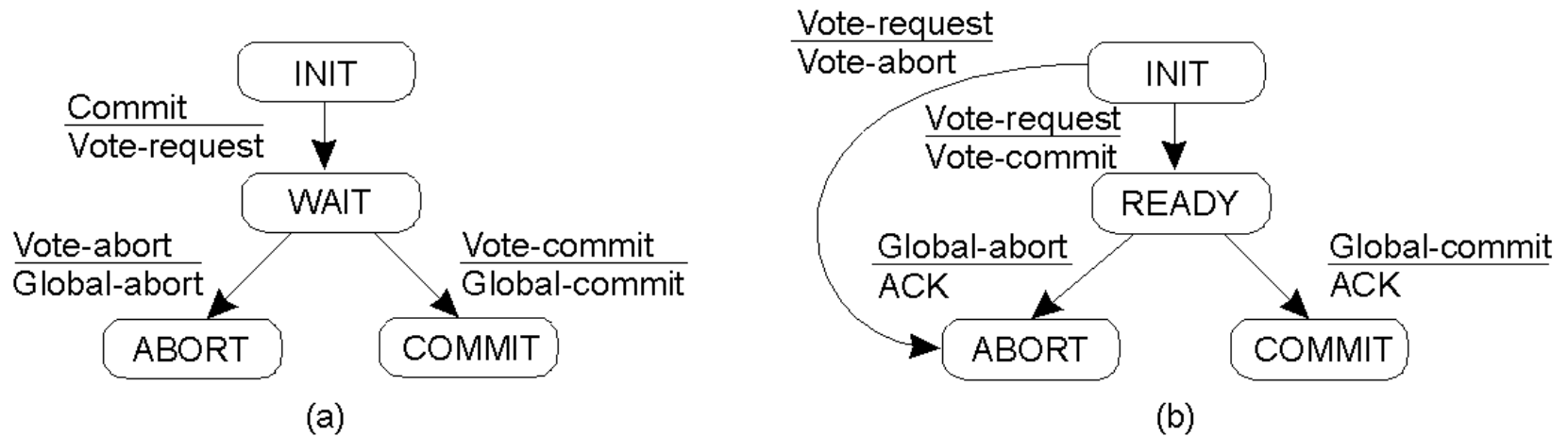


The essence of hierarchical reliable multicasting.

- a) Each local coordinator forwards the message to its children.
- b) A local coordinator handles retransmission requests.

Remember ... this is with reliable processes, what about voting?

# 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;
}
```

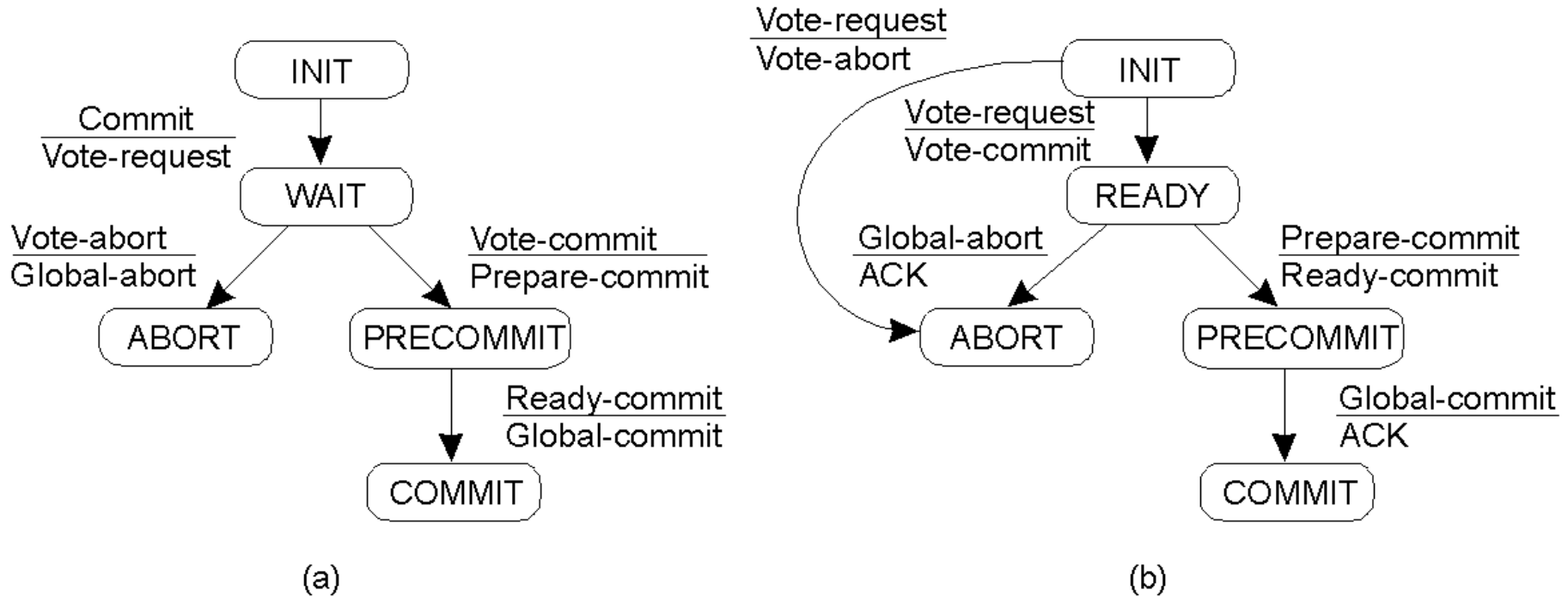
# Two-Phase Commit (5)

**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 */  
}
```

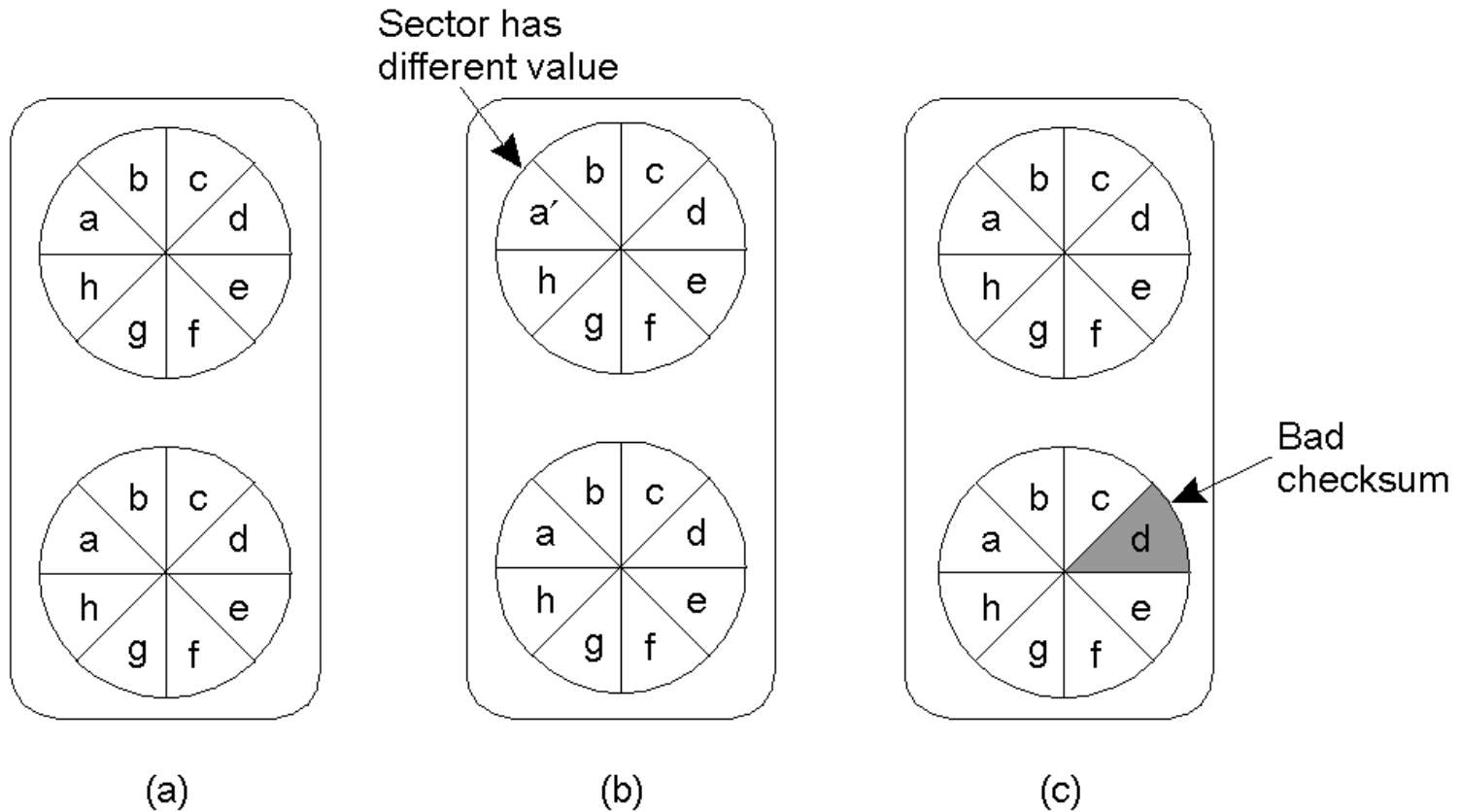
Steps taken for handling incoming decision requests.

# Three-Phase Commit



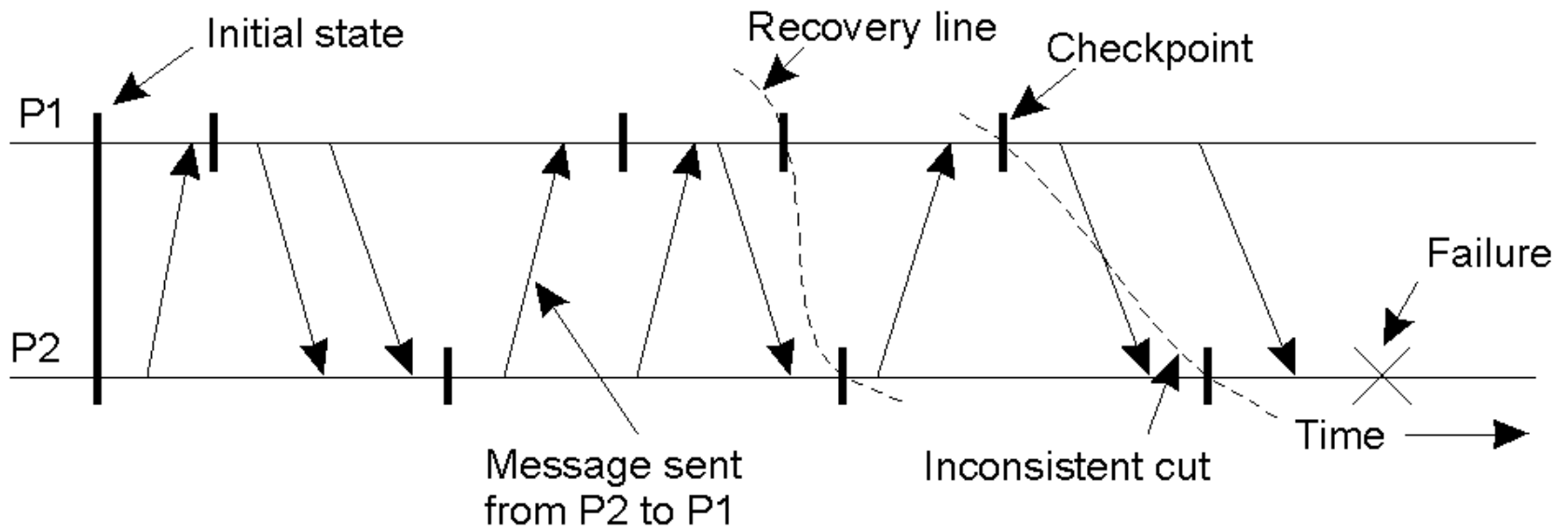
- a) Finite state machine for the coordinator in 3PC
- b) Finite state machine for a participant

# Recovery Stable Storage



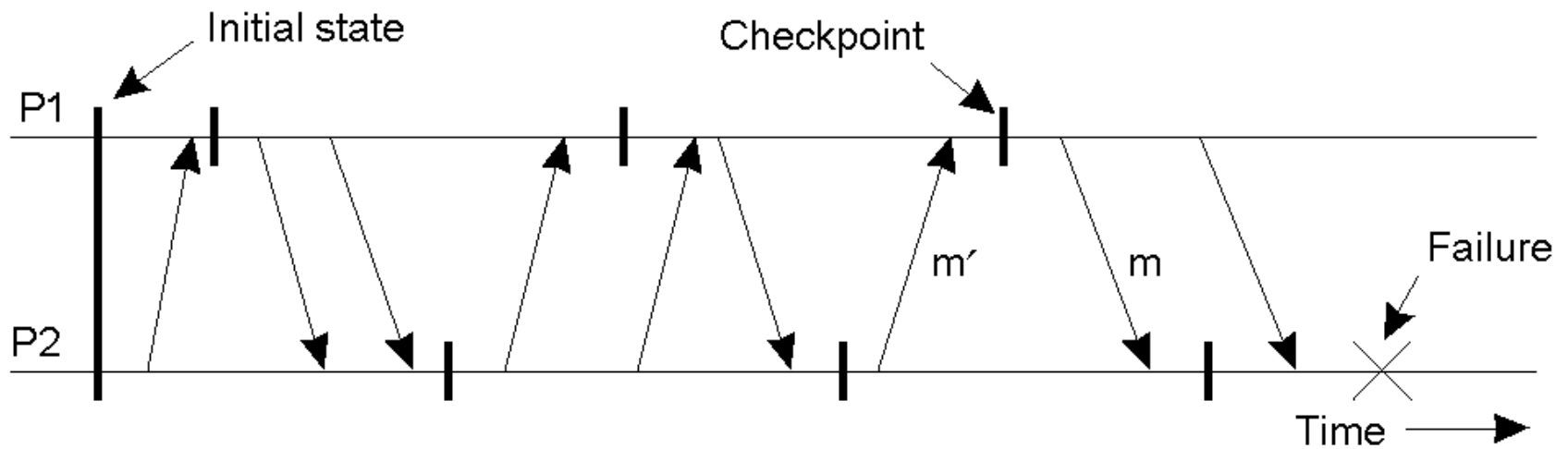
- a) Stable Storage
- b) Crash after drive 1 is updated
- c) Bad spot

# Checkpointing



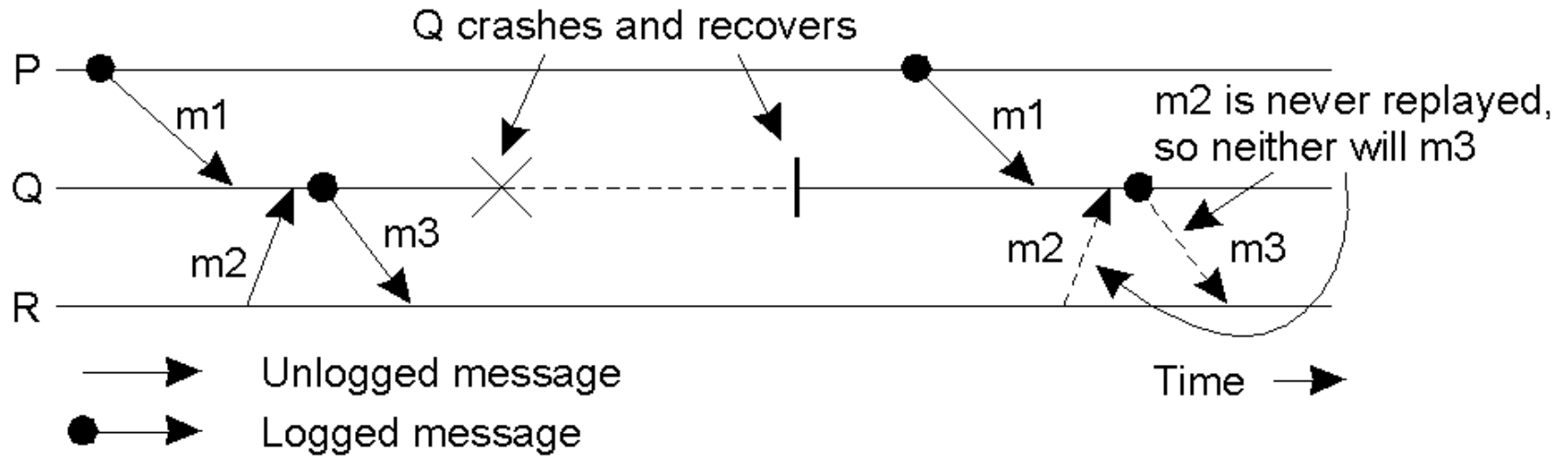
A recovery line.

# Independent Checkpointing



The domino effect.

# Message Logging



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