DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

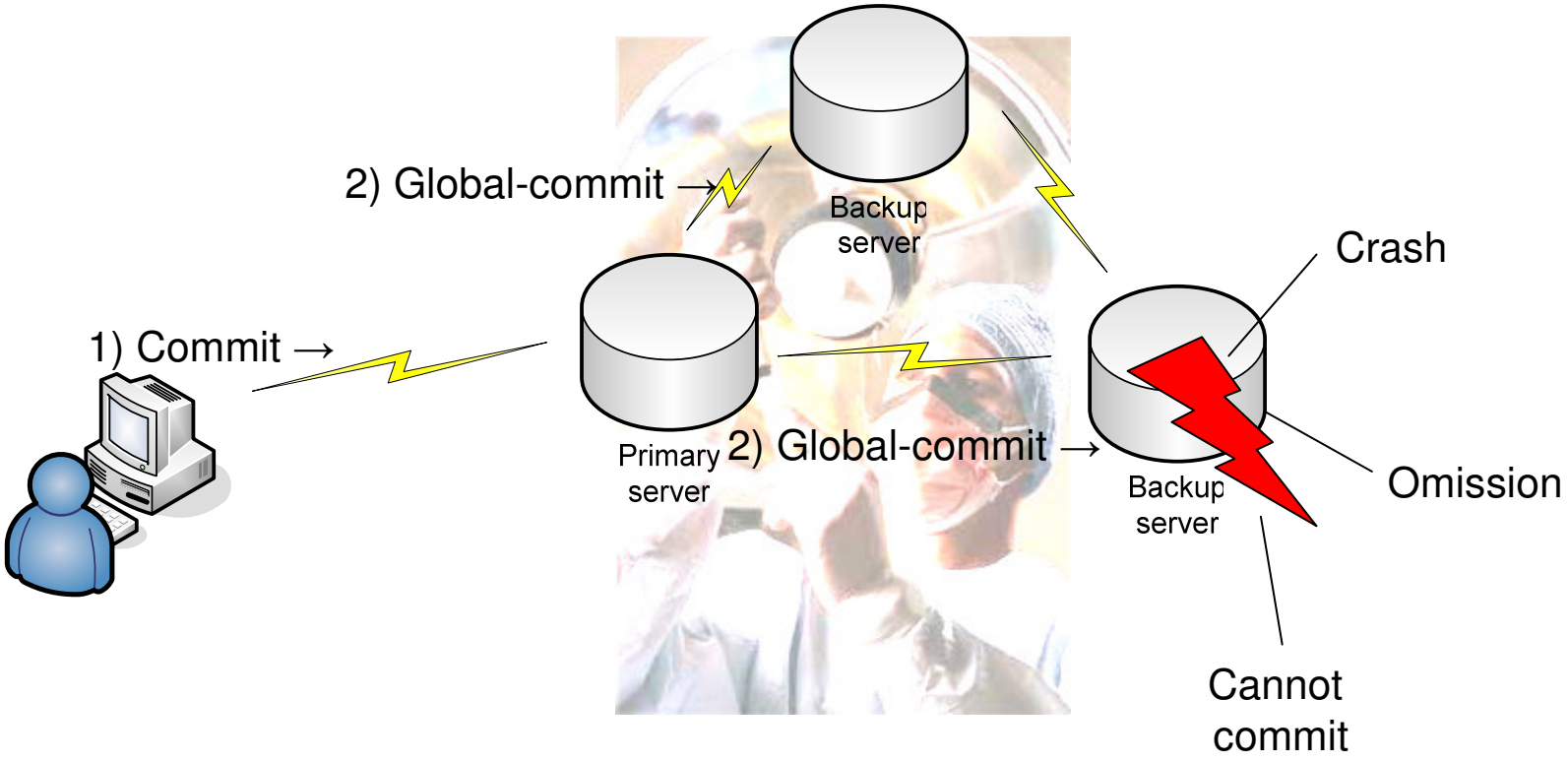
Chapter 8
Fault Tolerance (2)

Plan

- Basic Concepts
- Process Resilience
- Reliable communication
  - Client-server communication
  - Group communication
- Distributed commit
- Recovery

Tolerating failures

What can go wrong?

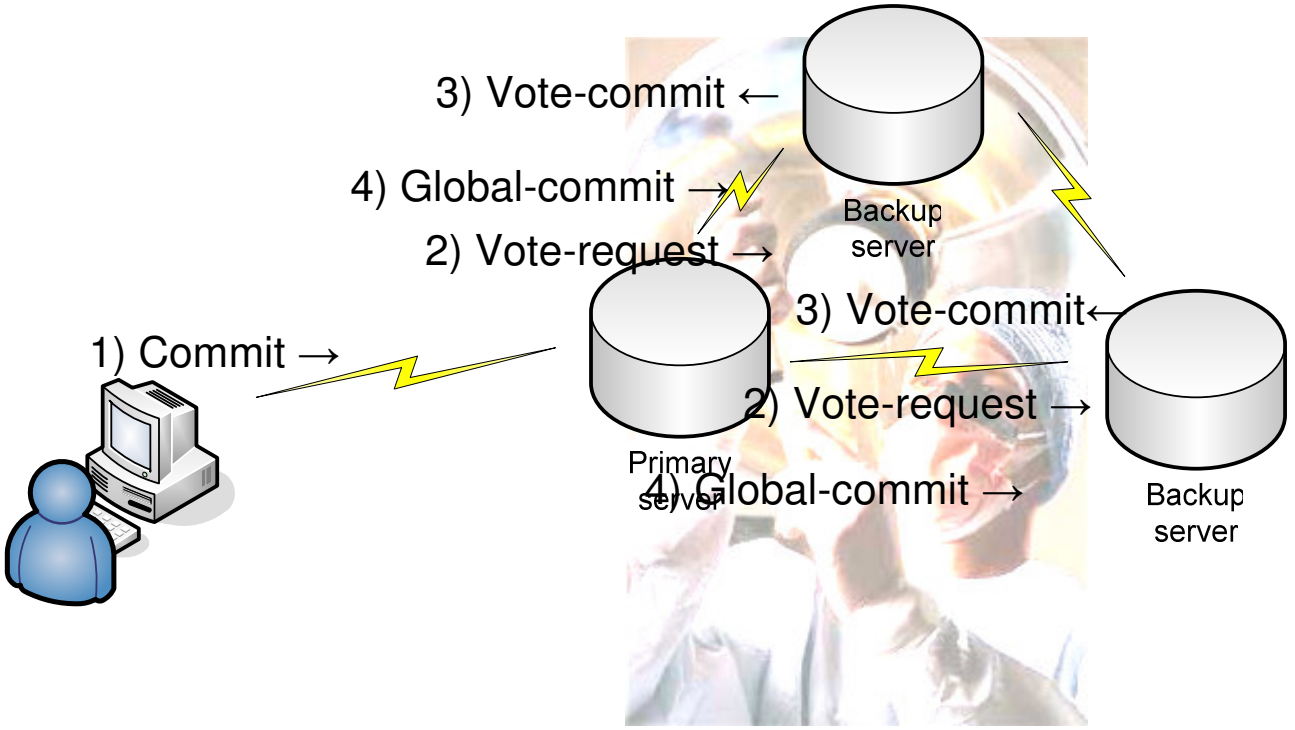


Distributed Commit

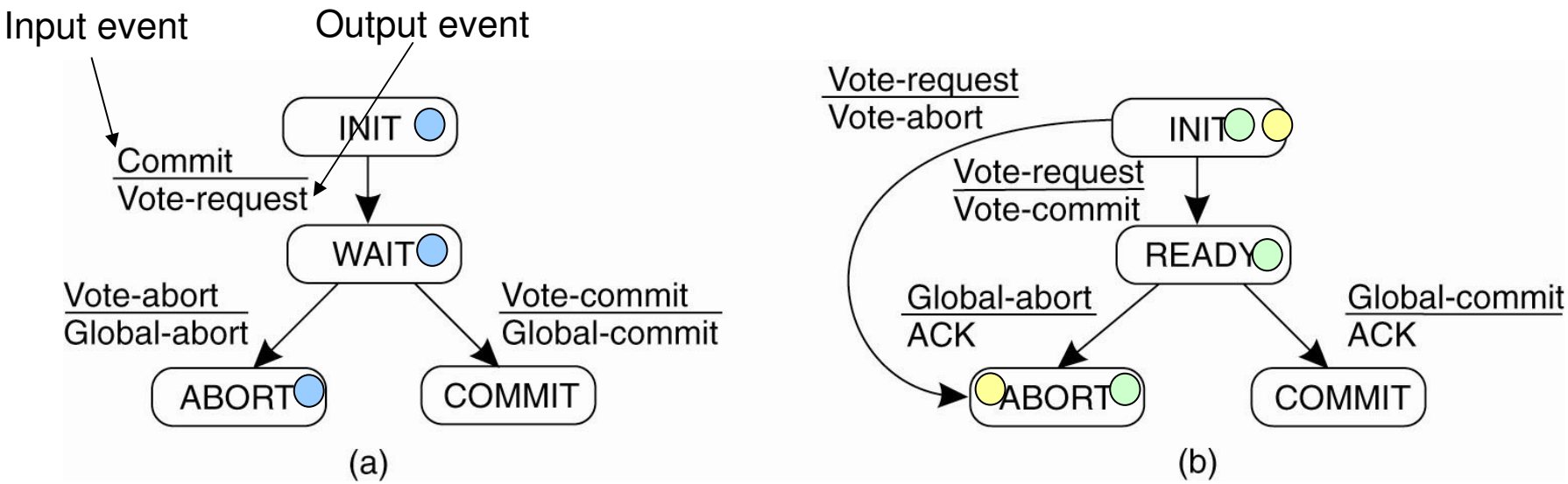
- Given a process group and an operation
- Either everybody *commits* or everybody *aborts*
  - Consistency, validity, termination

- Can we do this with Virtual Synchrony?
  - Cheating, but…
    - Coordinator multicasts vote request
    - All processes respond to request
    - Coordinator multicasts vote result
  - Some instances of failures handled by this
  - Essentially *two-phase commit*

Two-Phase Commit



Two-Phase Commit



- Figure 8-18. (a) The finite state machine for the coordinator in 2PC. (b) The finite state machine for a participant.

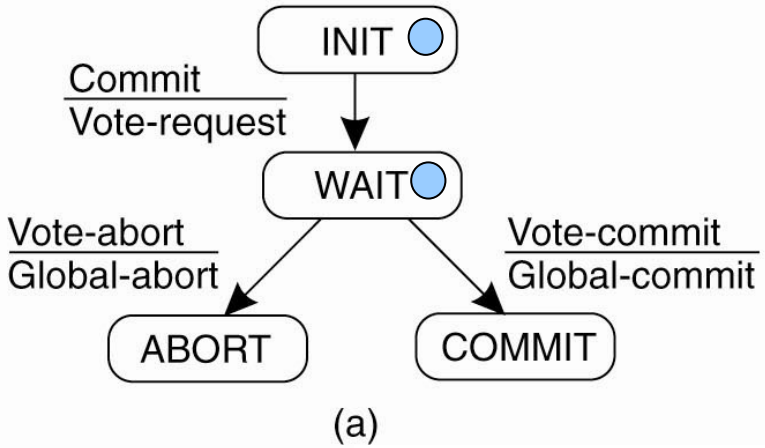
Two-Phase Commit

- Failures
  - Crash and omission
  - Detect via timeouts
- Processes may recover
  - Need for logging states

Two-Phase Commit

Actions by coordinator:

```
write 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 {
        write GLOBAL_ABORT to local log;
        multicast GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
```



(a)

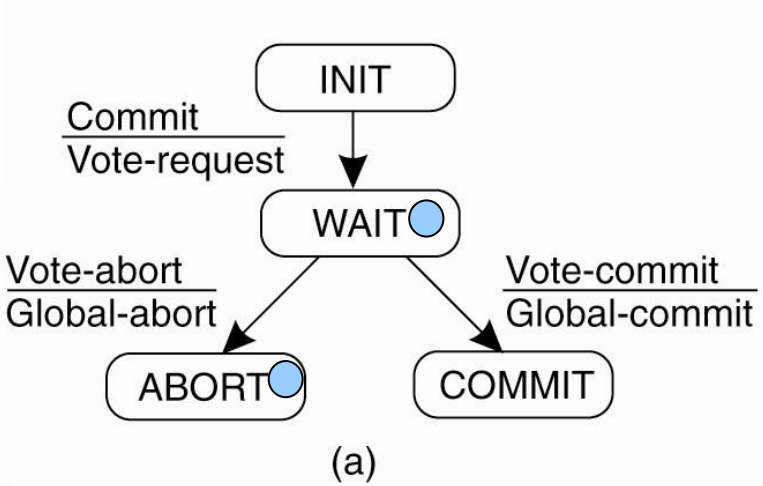
. . .

- Figure 8-20. Outline of the steps taken by the coordinator in a two-phase commit protocol.

Two-Phase Commit

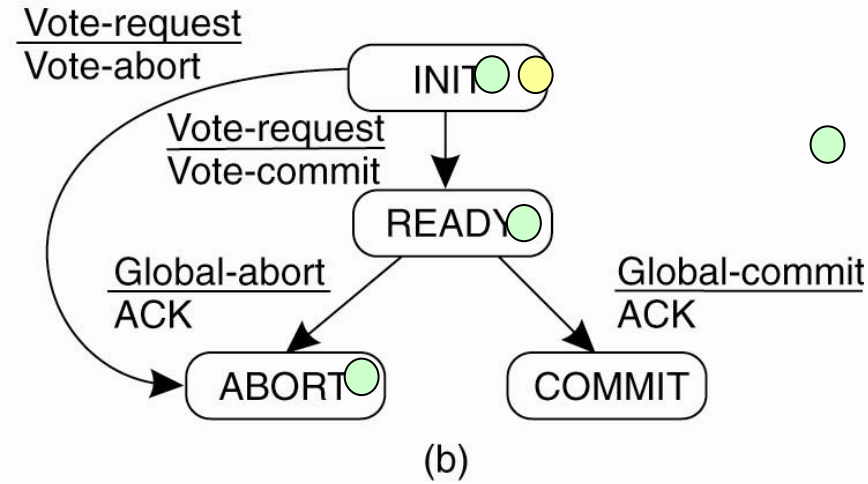
```
. . .

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;
}
```



- Figure 8-20. Outline of the steps taken by the coordinator in a two-phase commit protocol.

Two-Phase Commit



(b)

actions by participant:

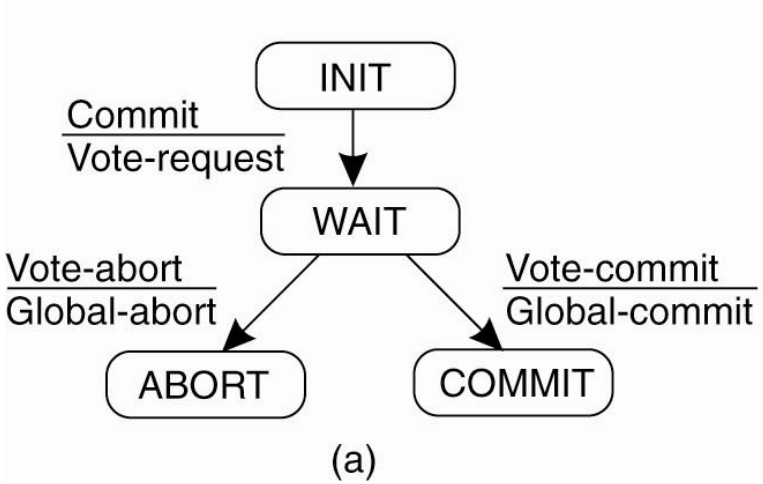
```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
      write VOTE_ABORT to local log;
      exit;
}
if participant votes COMMIT {
      write VOTE_COMMIT to local log;
      send VOTE_COMMIT to coordinator;
      wait for DECISION from coordinator;
      if timeout {
            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;
}
```

(a)

- Figure 8-21. (a) The steps taken by a participant process in 2PC.

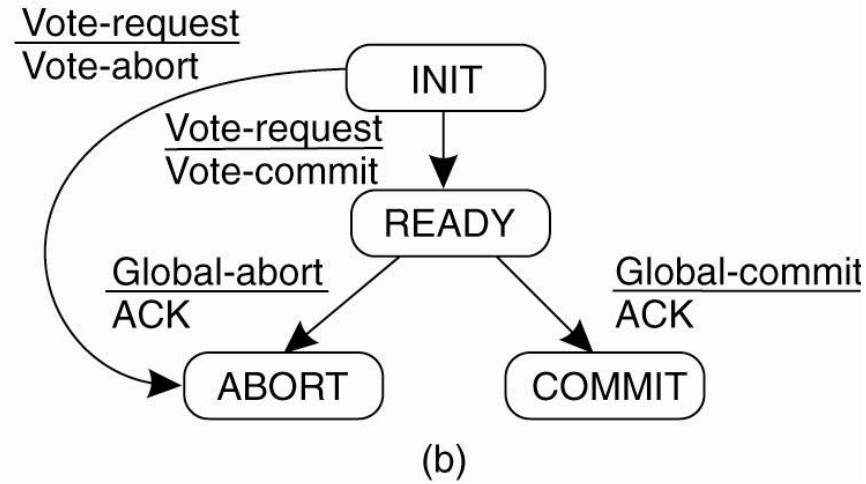
Coordinator Perspective

- Blocks in WAIT
  - Participant may have failed



Participant Perspective

- Blocks in READY
  - Coordinator may have failed
- What to do?
  - Some participants may already have committed…



Two-Phase Commit

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 */
}
```

(b)

- Figure 8-21. (b) The steps for handling incoming decision requests..

Two-Phase Commit

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

We know that coordinator managed to start commit

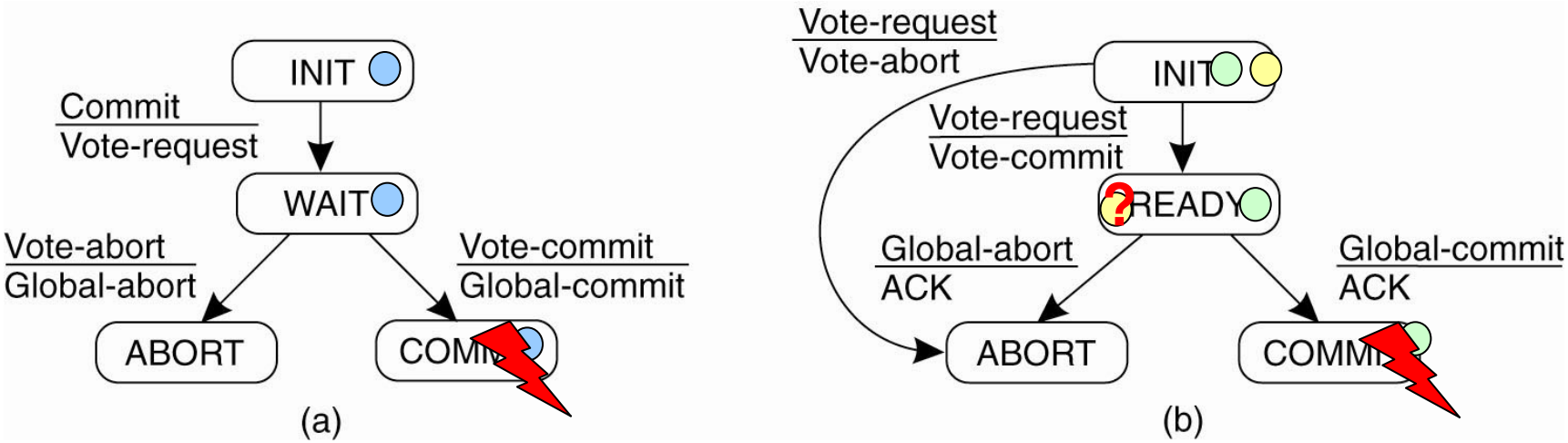
At least one participant aborted and coordinator noticed

Q did not even receive vote-request

Safe to abort?

- Figure 8-19. Actions taken by a participant P when residing in state READY and having contacted another participant Q.

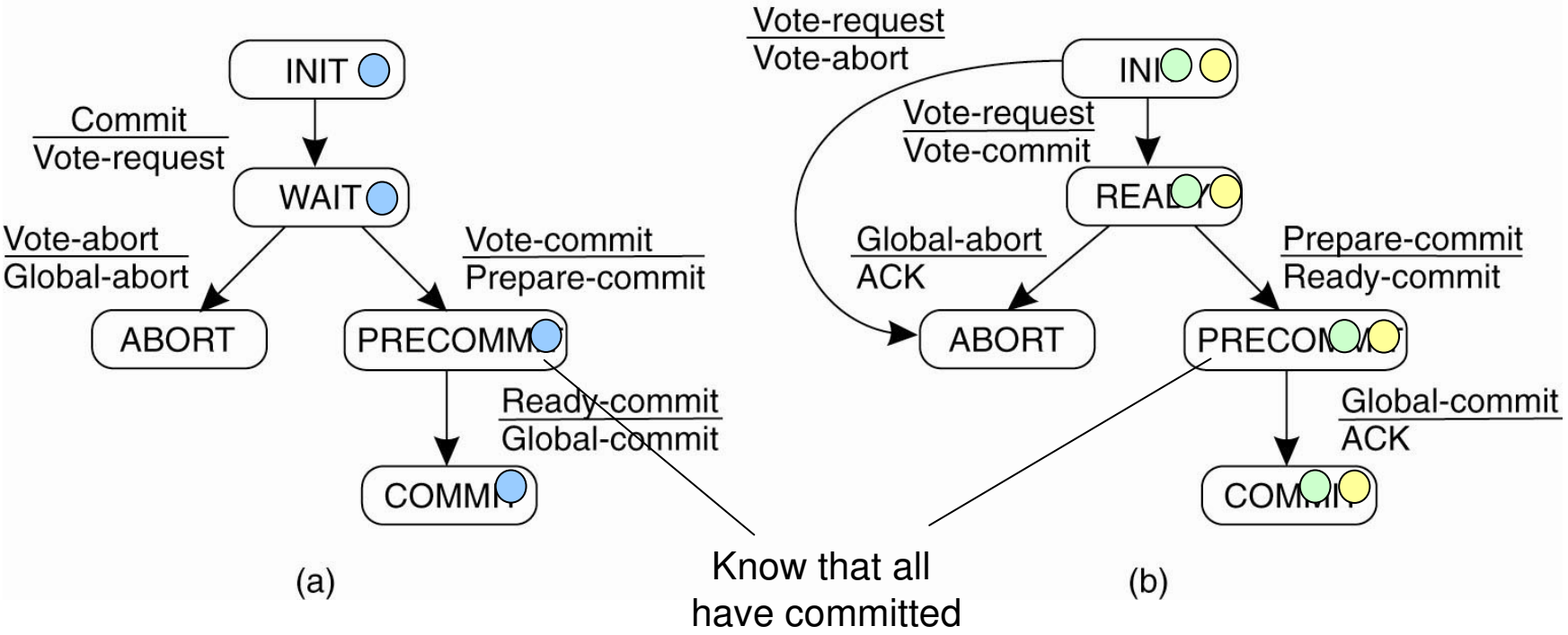
Bad State



Three-Phase Commit

- The states of the coordinator and each participant satisfy the following two conditions:
  1. There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state.
  2. There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made.

Three-Phase Commit

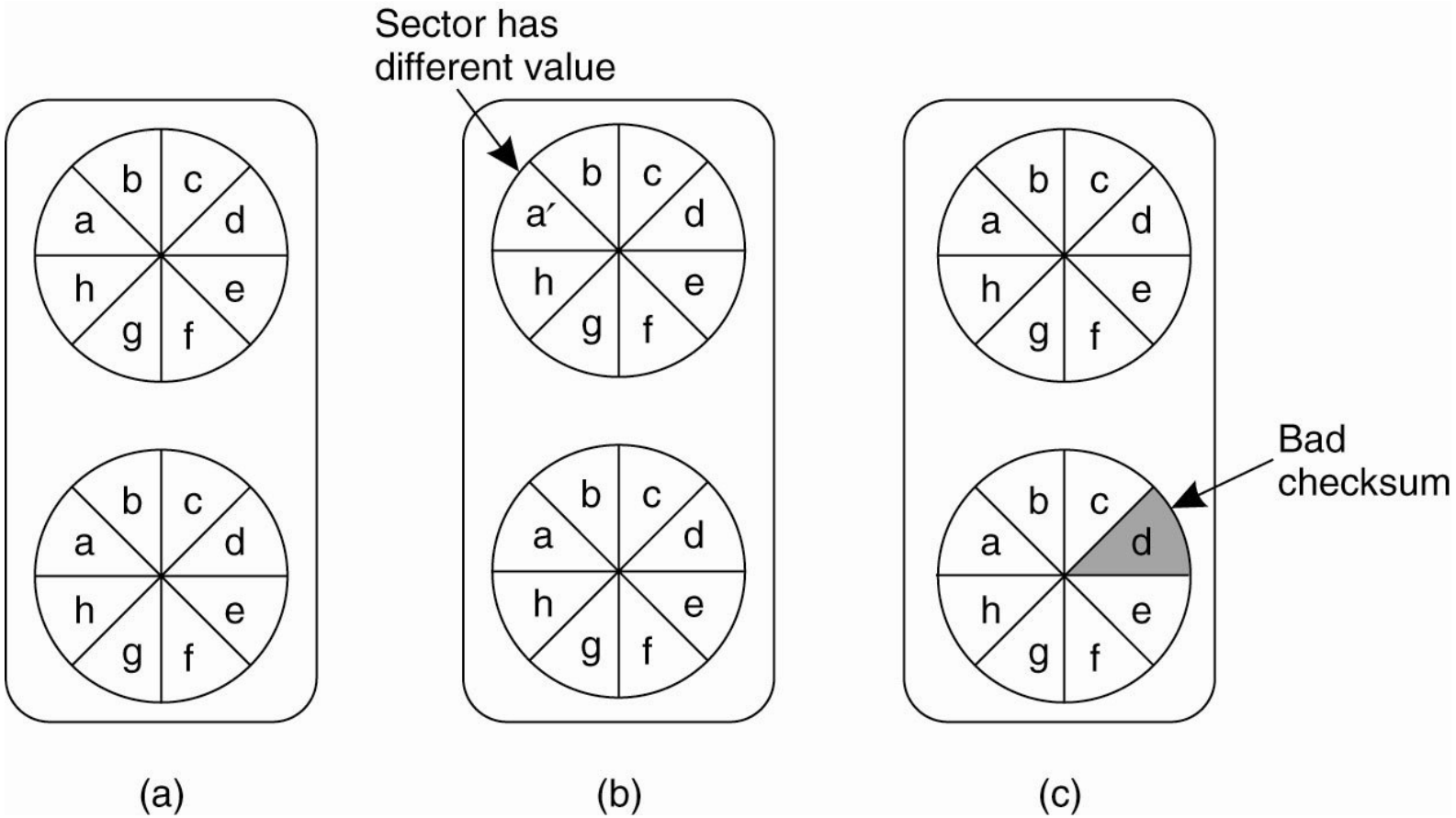


- Figure 8-22. (a) The finite state machine for the coordinator in 3PC. (b) The finite state machine for a participant.

Recovery

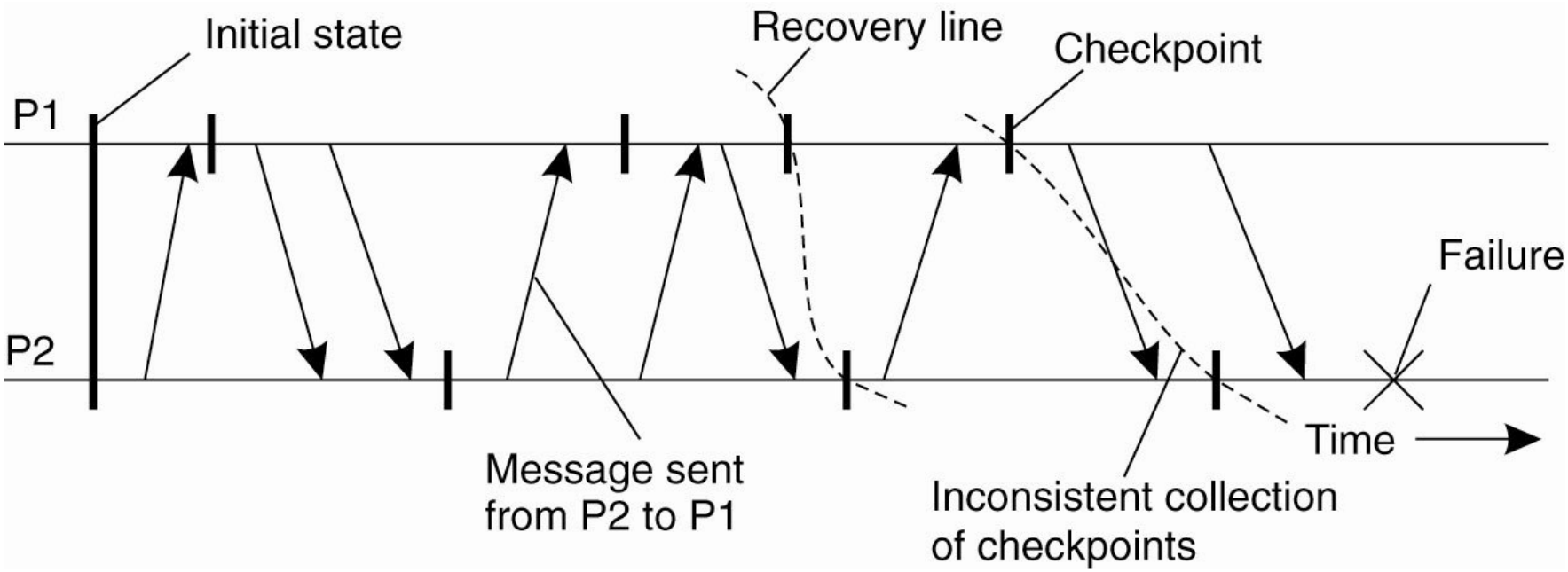
- So what if a failure occurs?
  - Need to be able to recover to a correct state
- Backward recovery
  - Bring the system to backward to a correct, previous state
  - Restore
- Forward recovery
  - Bring the system forward to a correct, new state
- What did we do in 2PC and 3PC?

Recovery – Stable Storage



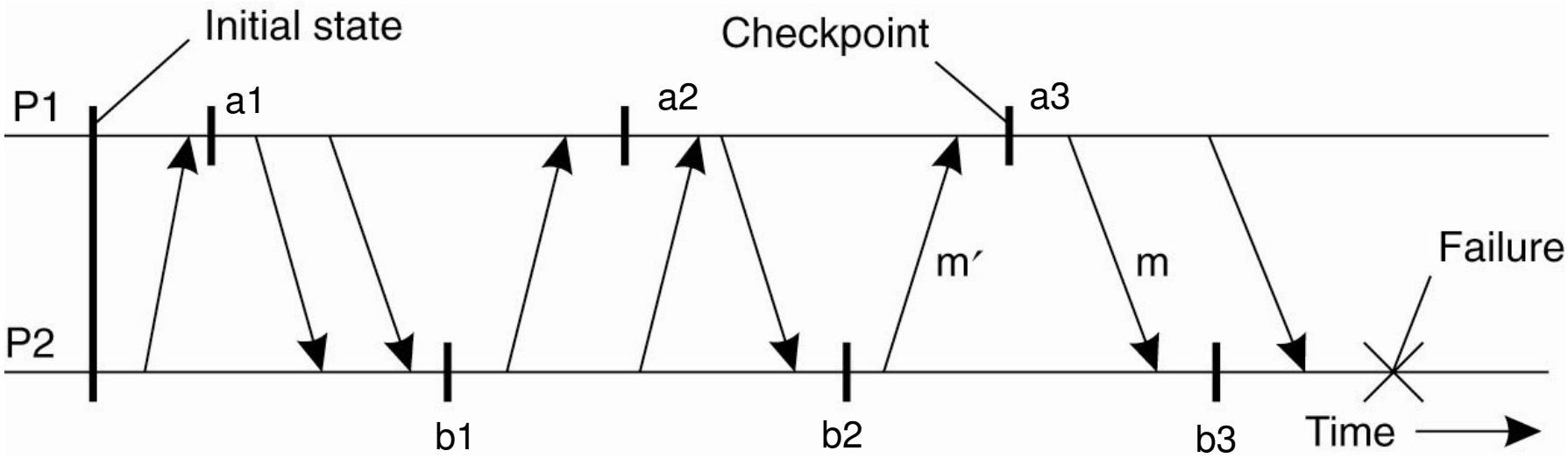
- Figure 8-23. (a) Stable storage on two disks.
  (b) Crash of drive 2 after drive 1 is updated. (c) Bad spot on drive 2.

Recovery – Checkpointing



- Figure 8-24. A recovery line.
- We want a consistent distributed snapshot
  - System saves state on stable storage
  - If recv(m) is in snapshot, then send(m) is also there

Independent Checkpointing



- Figure 8-25. The domino effect.
  - a3,b3?
  - a3,b2?
  - …?

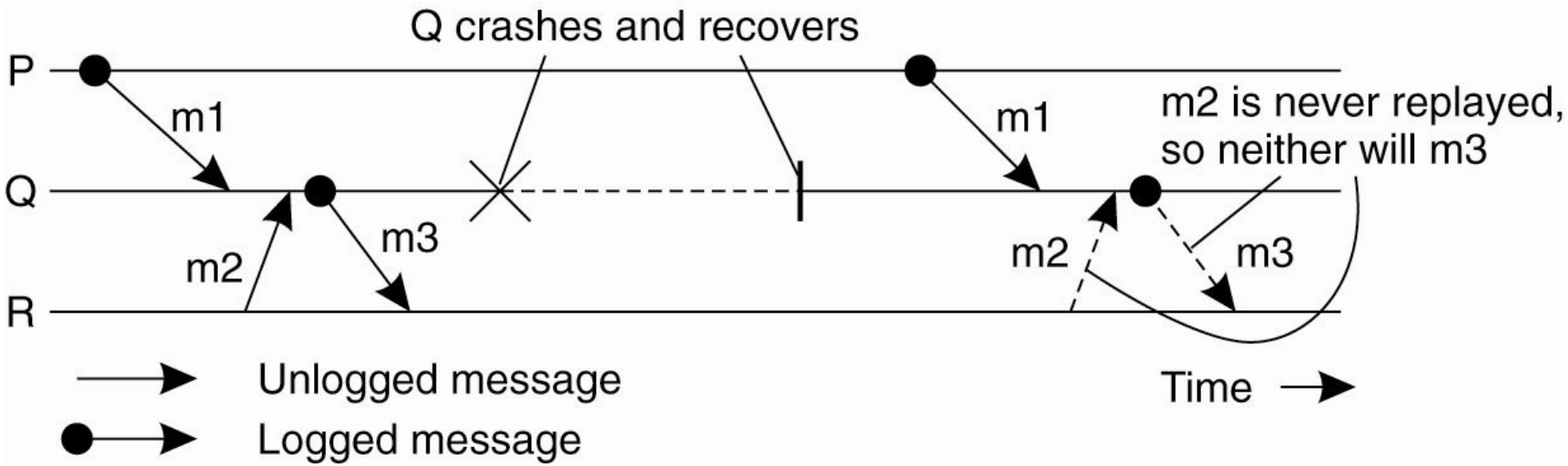
Coordinated Checkpointing

- Make sure that processes are synchronized when doing the checkpoint
- Two-phase blocking protocol

1. Coordinator multicasts *CHECKPOINT_REQUEST*
2. Processes take local checkpoint
   - Delay further sends
   - Acknowledge to coordinator
   - Send state
3. Coordinator multicasts *CHECKPOINT_DONE*

- Ordering constraints?

Characterizing Message-Logging Schemes



- Figure 8-26. Incorrect replay of messages after recovery, leading to an orphan process.

Summary

- Looked at last pieces of fault tolerance
- Distributed commit
  - 2PC – blocking, has bad state
  - 3PC – non-blocking, but not widely used in practice
- Recovery
  - Storage
  - Checkpointing