CSci 5105

Introduction to Distributed Systems

Fault Tolerance

Last Time

- Replication and Consistency

Today

- Fault tolerance
- Chapter 8 TVS

Fault Tolerance Basics

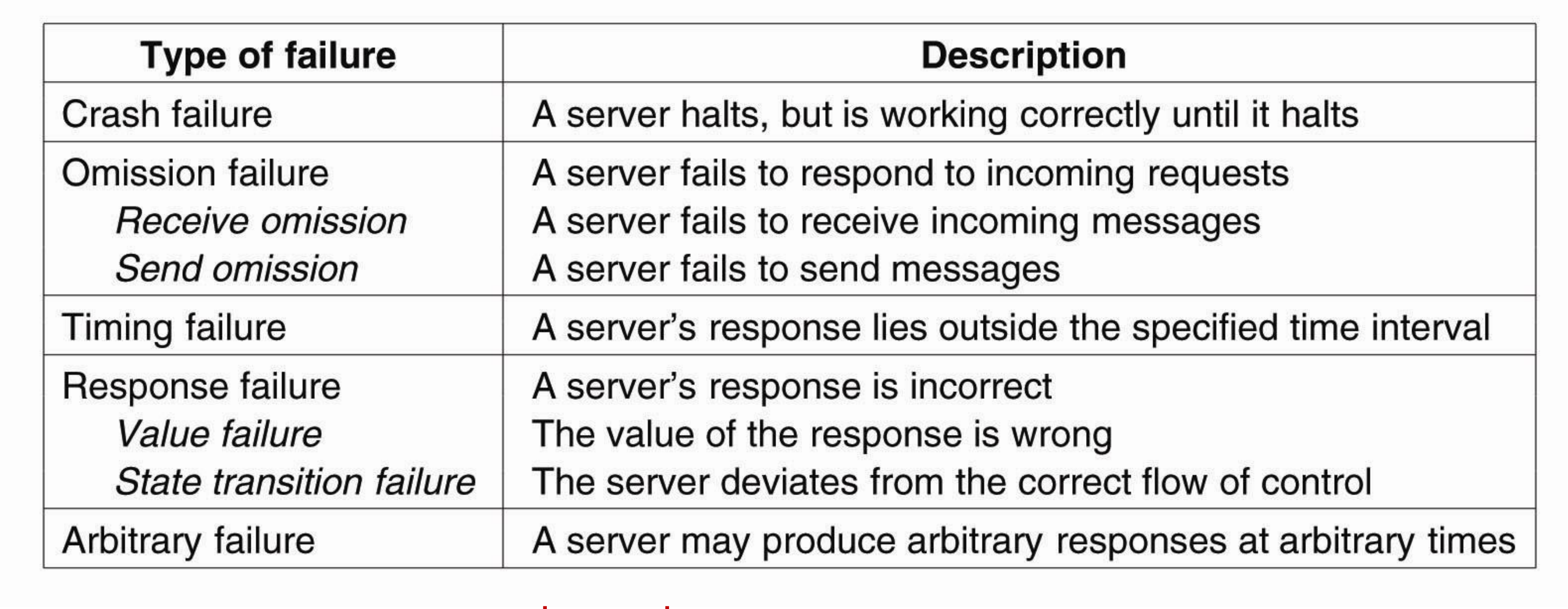
- Availability
  - short time horizon
  - e.g down 1 msec every hour => 99.9999 avail
- Reliability
  - over longer time horizon
  - e.g. but not that reliable, no job can run > 1 hr
- Safety: temporary failure # catastrophe
- Maintainability: ease of repair

Brewer Avail

More Definition

- Fail: cannot meet promises
- Error: system state may => failure
- Fault: cause of an error
- Tolerate faults => operate correctly
- Fault types
  - Transient, intermittent, permanent

Failure Models



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

byzantine

Failure Types

- fail-stop ~ crash failure
  - failed process stops producing output; easily detected as failed without ambiguity
  - machine on my local network
- fail-silent
  - failure not so obvious: really slow or failed?
  - remote communicating process
- fail-safe
  - arbitrary failures that are recognized as such

RPC Failures

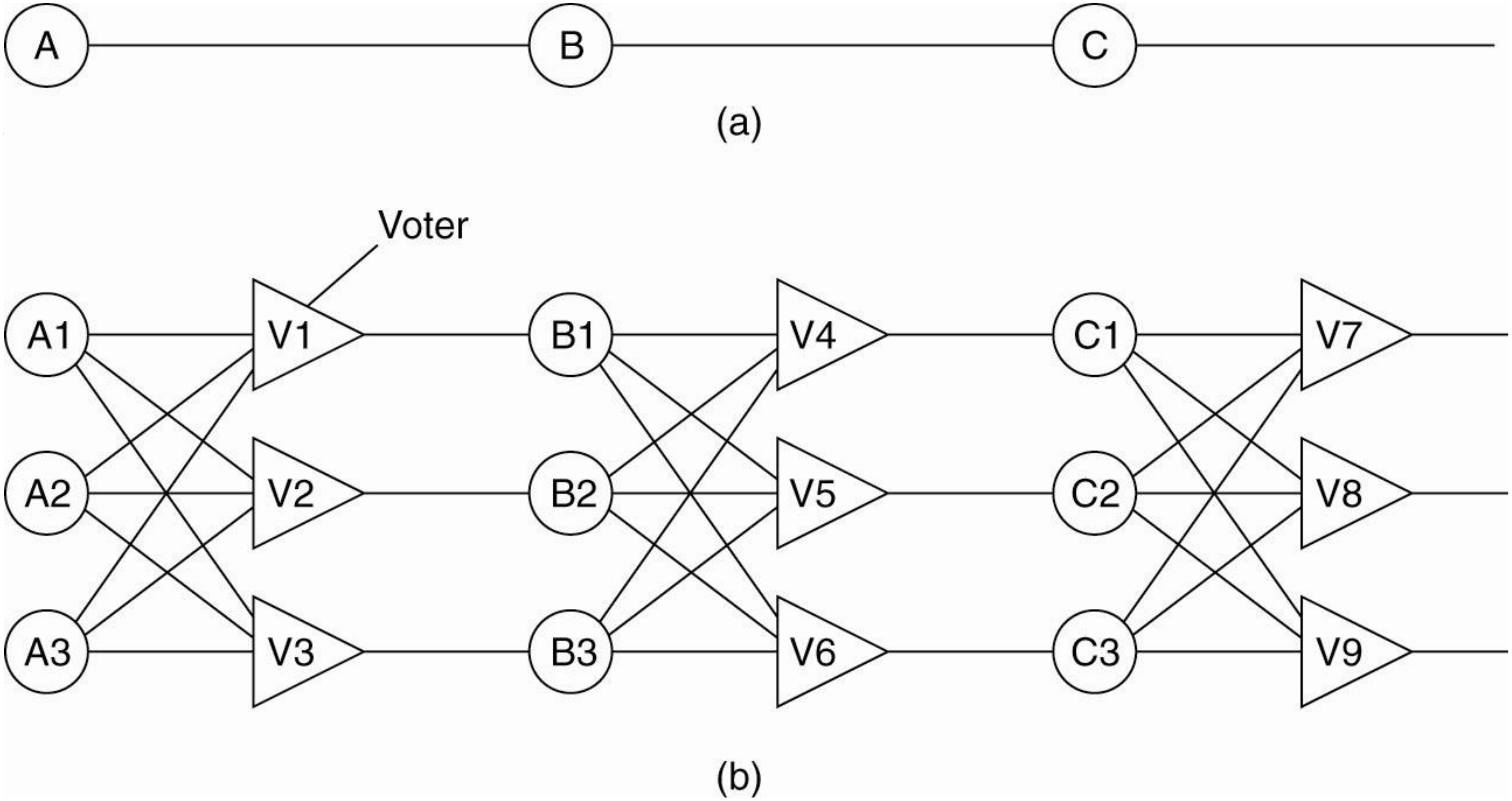
1. The client is unable to locate the server
    - raise exception
2. The req. message from the client to the server is lost
3. The server crashes after receiving a request
4. The reply message to the client is lost

2-4 Detect via time-out; take action (retransmit or not)

The client crashes after sending a request

- orphan – problem?

Failure Masking by Redundancy

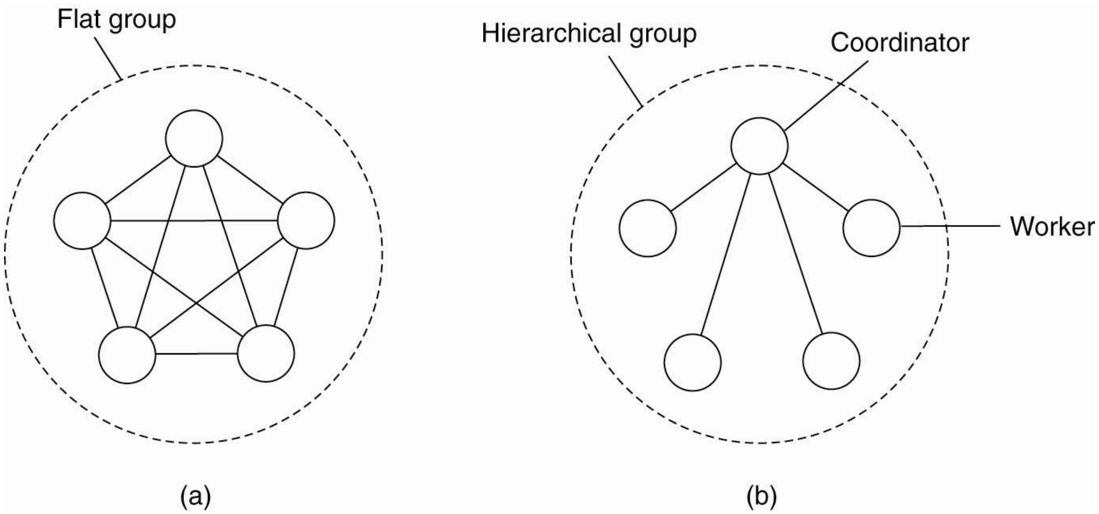


Classic TMR: throwing hardware at the problem

Assumptions?

Process Failures

- Process replication or groups
- Need to have group consensus
- Group can change: group management becomes key



- Compare?

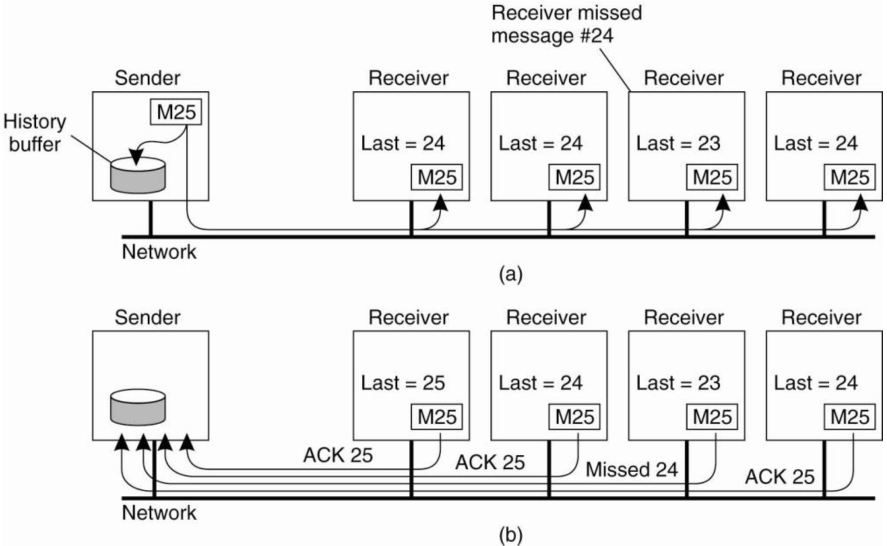
~ primary backup

Failure Masking + Replication

- General groups
  - K fault tolerant (K failaures)
    - fail-stop/fail-silent =>
    - byzantine failures =>

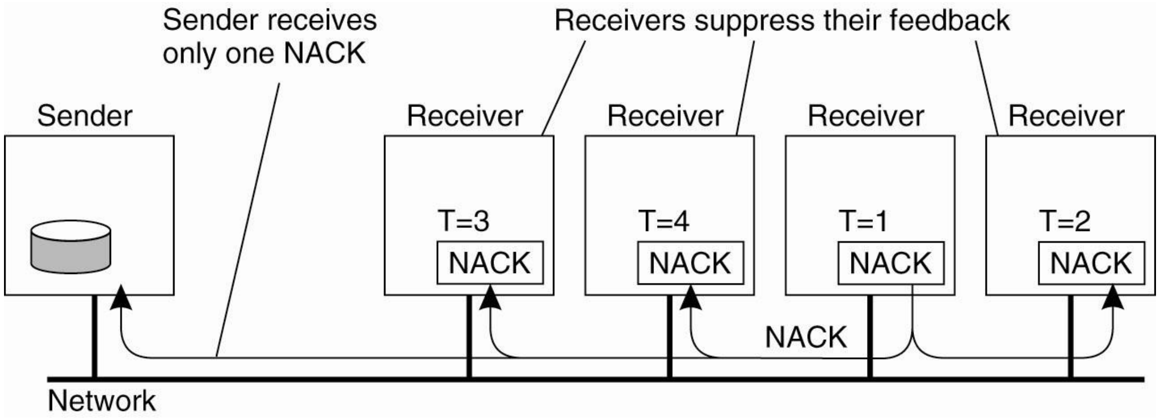
Agreement in Faulty Systems

- Examples
  - voting, leader election, multicast
- Reliable multicast
  - group is fixed
  - failure reported via feedback



Feedback Control

- Missing a message can unicast or multicast
- K missing: K unicasts or multicasts
- Latter: nice optimization
  - delay a little before requesting retransmission
  - another node may do it
  - So maybe 1 retransmitted multicast will suffice



Atomic Multicast

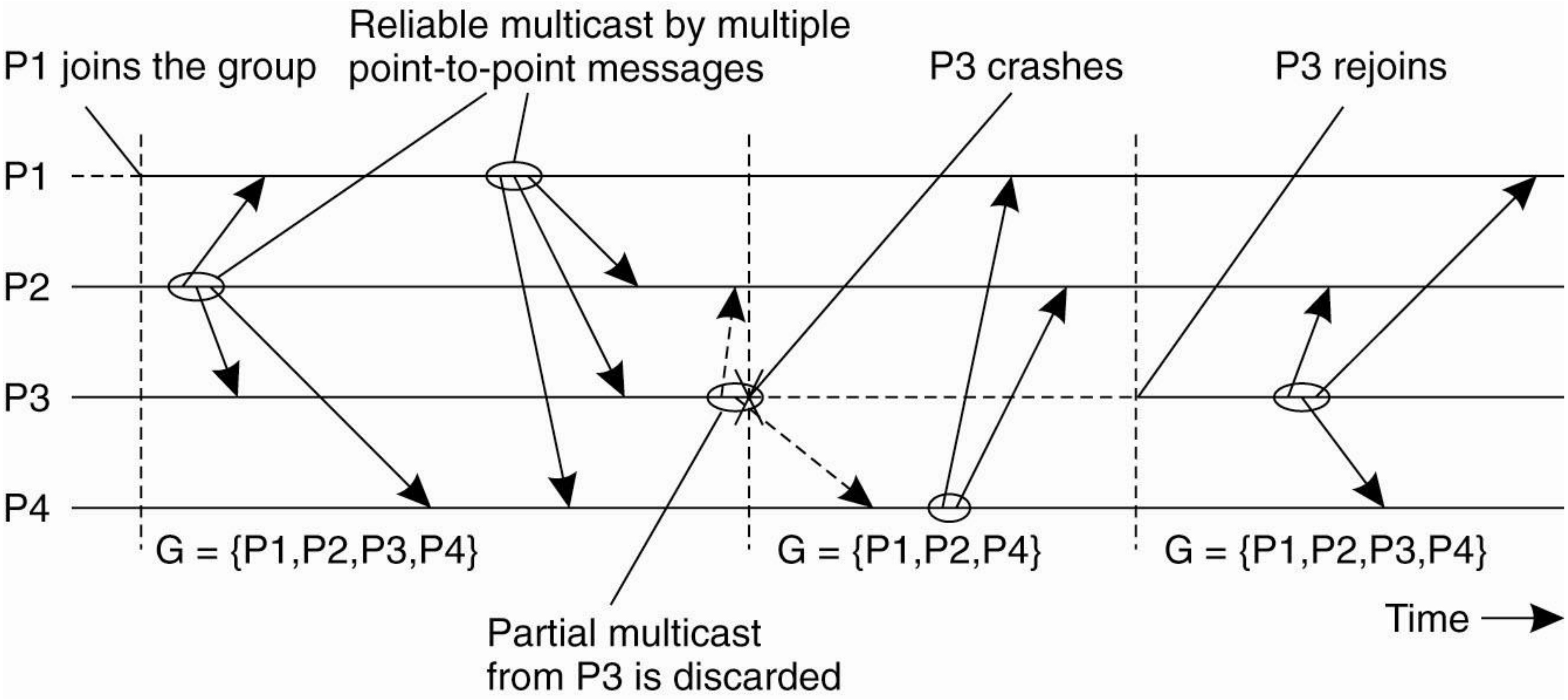
- Reliable multicast and ordering
- Everyone sees same message order or none
- Eg. Consistency => DB updates
- Problem: group members come and go
- Agree who is in the group
  - View synchronous

Virtual Synchrony

- Group view
  - When message M is sent; everyone agrees who is in the group
  - If group state changes during M
    - M delivered to all before group change or to none

- This is known as virtual synchrony

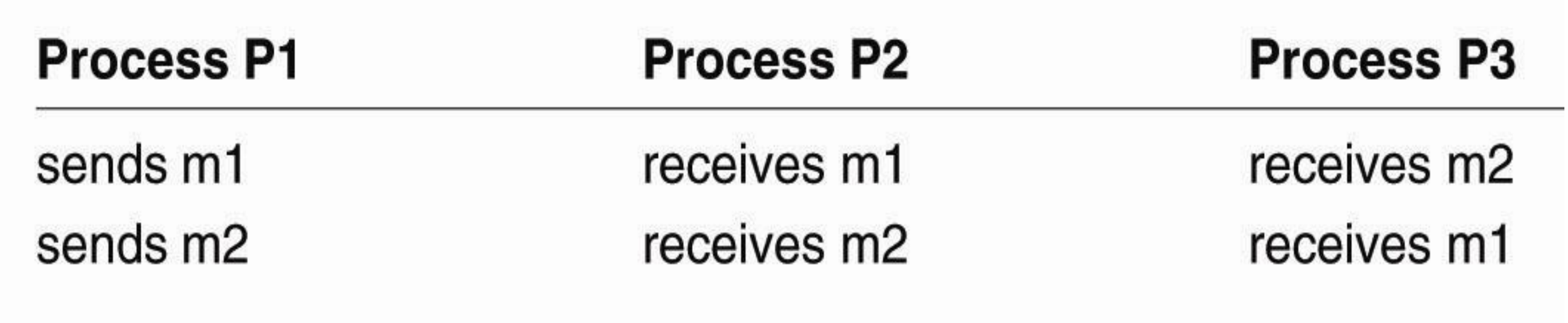
Virtual Synchrony



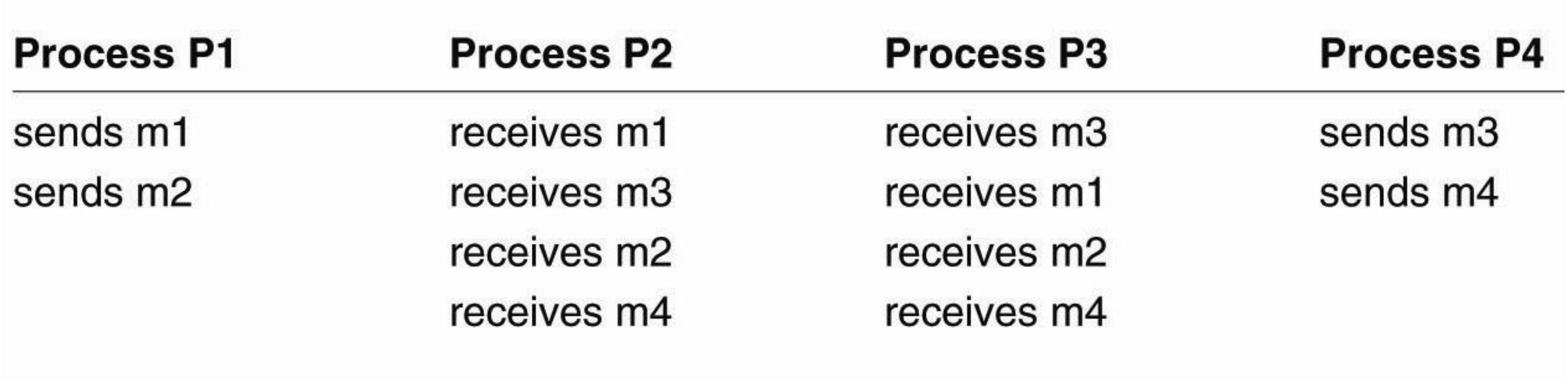
Multicast Message Ordering

- Unordered multicasts
- FIFO-ordered multicasts
  - Easy: issue message in sequence order
- Causally-ordered multicasts
  - Harder: need vector time-stamps
- Totally-ordered multicasts
  - Need a global sequencer
  - Each multicast message is given a global #: 1, 2, 3, …

Message Ordering



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



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	

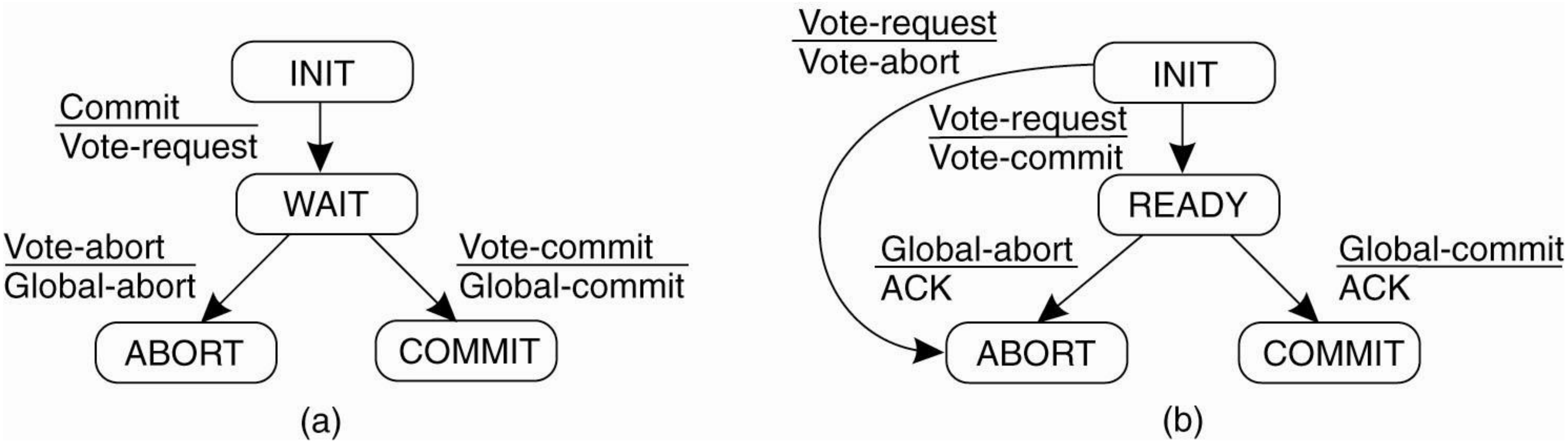
- What ordering do these satisfy?

Two-Phase Commit (2PC)

- Send message and have everyone either act on message or not
- Typical action: commit a transaction
- Multi-step
  - Vote-request
  - Vote-commit or vote-abort
  - Global-commit or global-abort
- Impressions?

Two-Phase Commit (2PC)

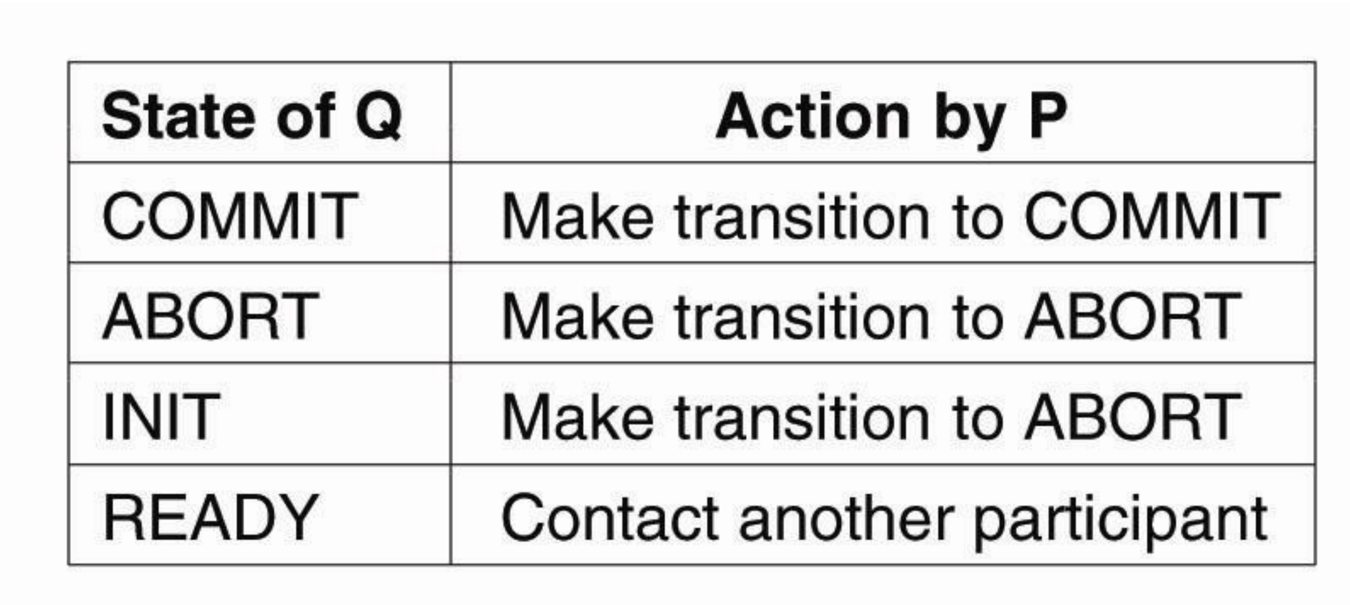
Coordinator participant



- Distributed commit – all or none

What about failure?

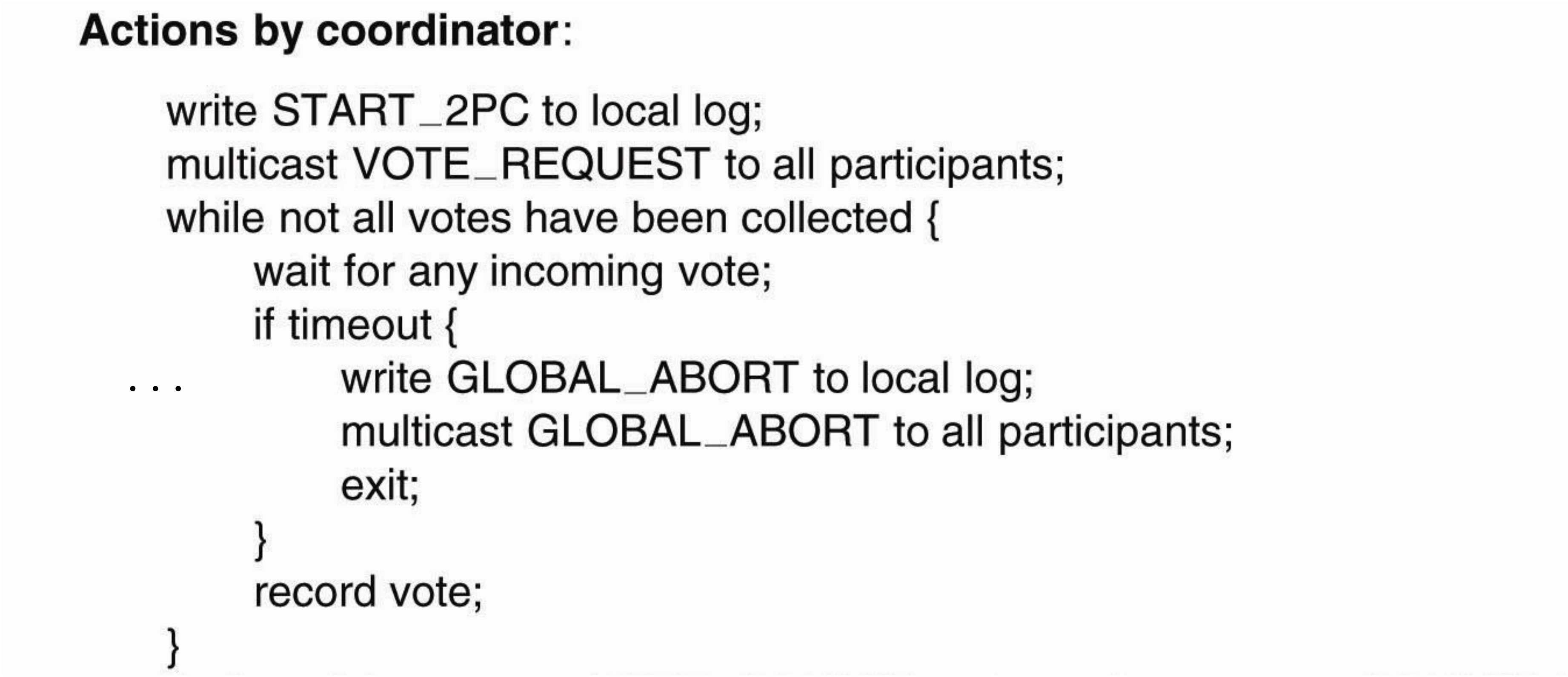
- Coordinator failure
- Node P in READY state and times out
- Asks node Q



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

2PC Failure/Recovery

- Nodes fail and may recover
- Use logging

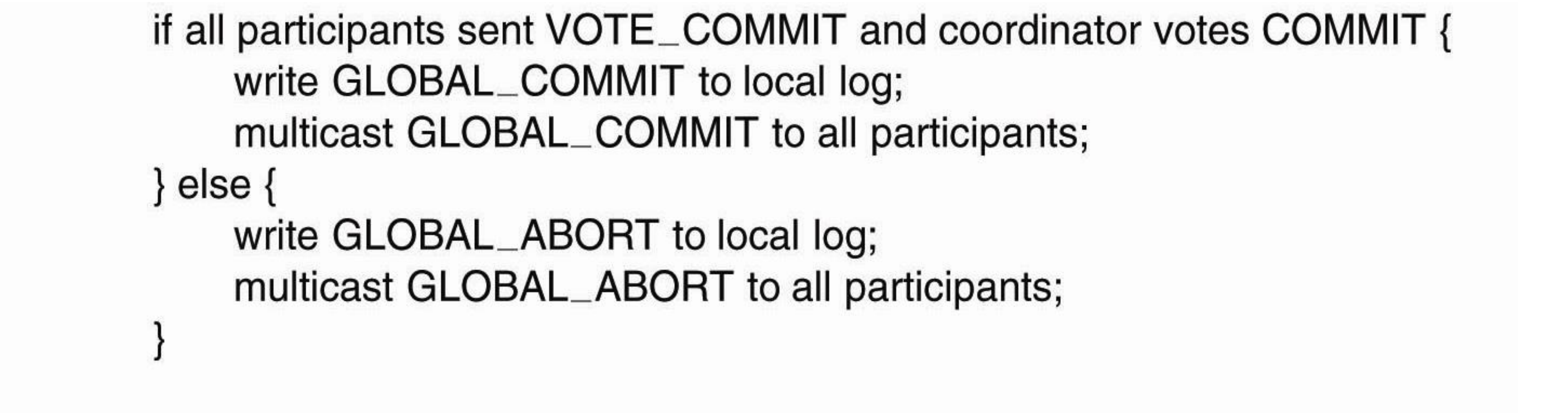


Actions by coordinator:

```
    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;
    }
```

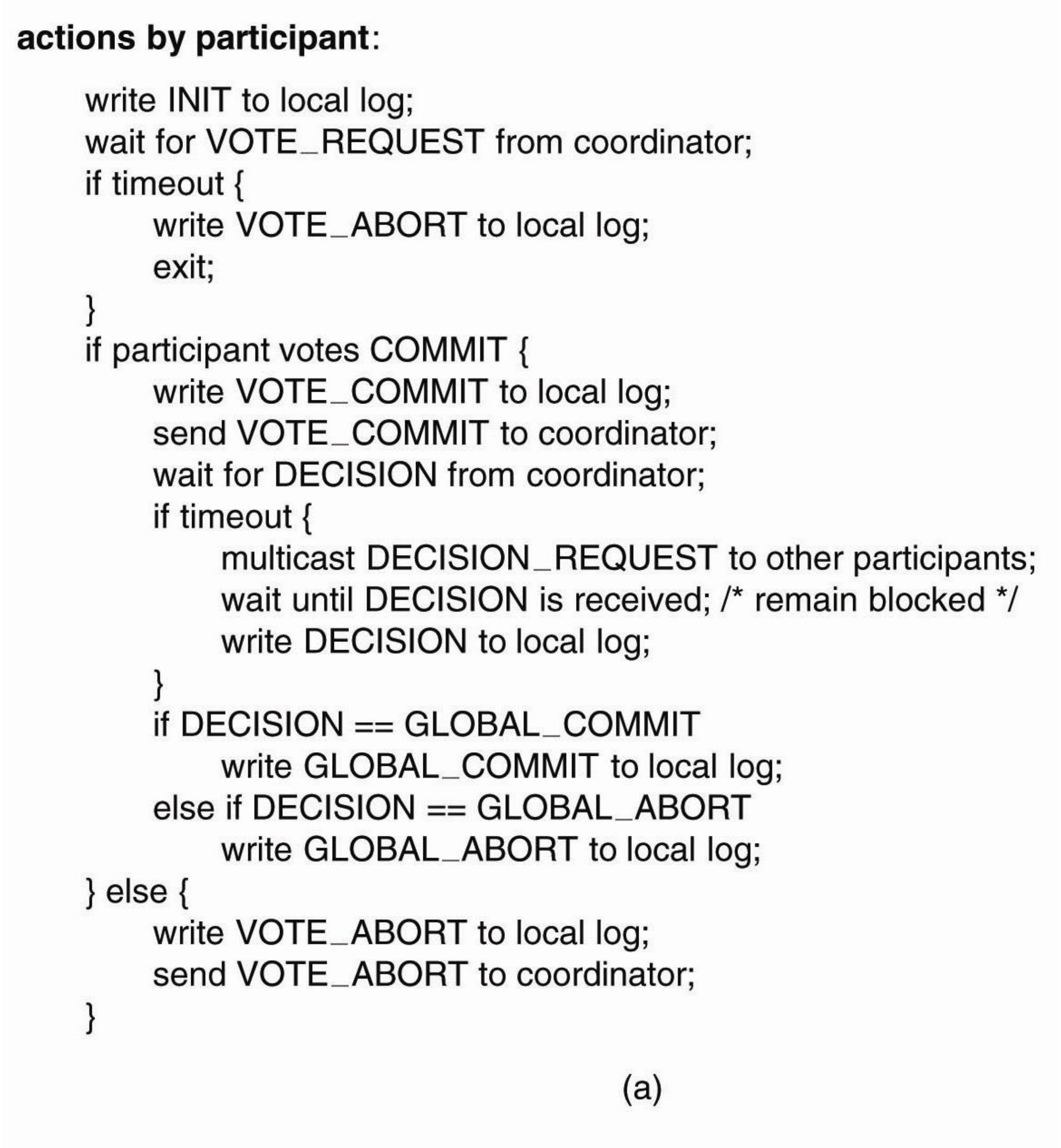
2PC Failure/Recovery (cont’d)

. . .



```
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;
}
```

2PC: Participant recovery

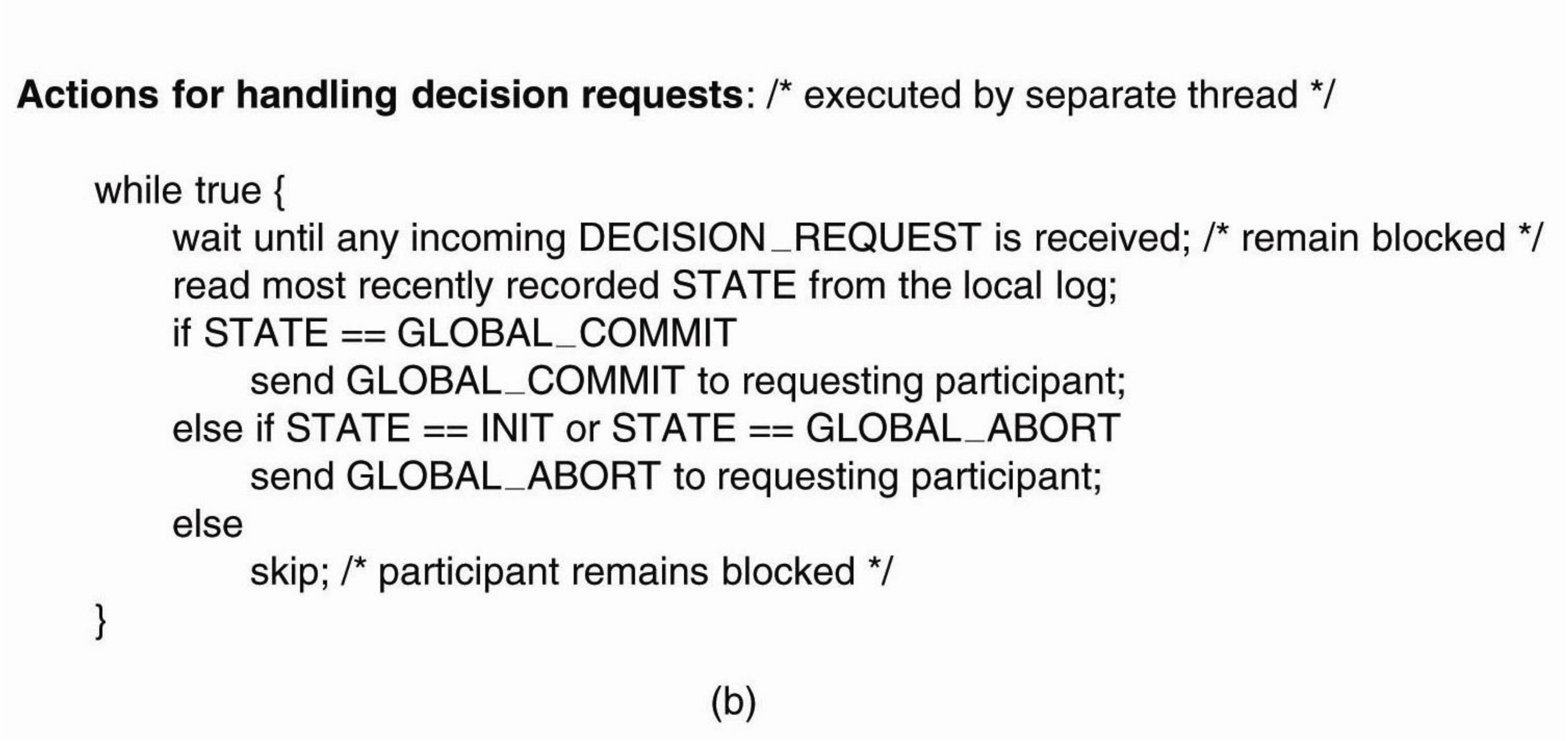


actions by participant:

```
    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)

2PC: Participant recovery (cont’d)



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)

- Used to help other participants

Next Time

- Byzantine Agreement and Recovery
- Read Chapter 8 TVS and FT* paper