

Module 8 - Fault Tolerance

Dependability

■ Reliability

- A measure of success with which a system conforms to some authoritative specification of its behavior.
- Probability that the system has not experienced any failures within a given time period.
- Typically used to describe systems that cannot be repaired or where the continuous operation of the system is critical.

■ Availability

- The fraction of the time that a system meets its specification.
- The probability that the system is operational at a given time t .

■ Safety

- When the system temporarily fails to conform to its specification, nothing catastrophic occurs.

■ Maintainability

- Measure of how easy it is to repair a system.

Fundamental Definitions

■ Failure

- The deviation of a system from the behavior that is described in its specification.

■ Erroneous state

- The internal state of a system such that there exist circumstances in which further processing, by the normal algorithms of the system, will lead to a failure which is not attributed to a subsequent fault.

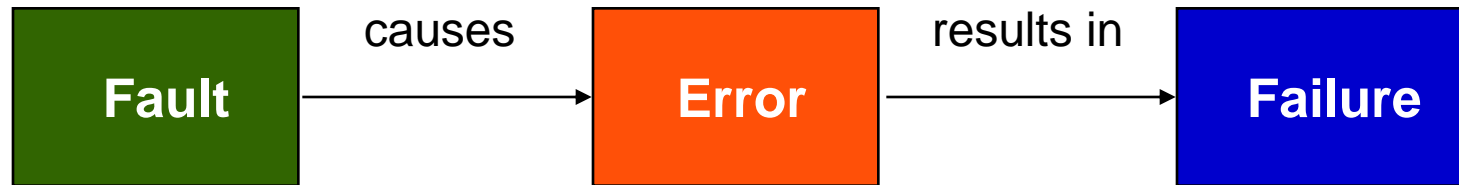
■ Error

- The part of the state which is incorrect.

■ Fault

- An error in the internal states of the components of a system or in the design of a system.

Faults to Failures



Types of Faults

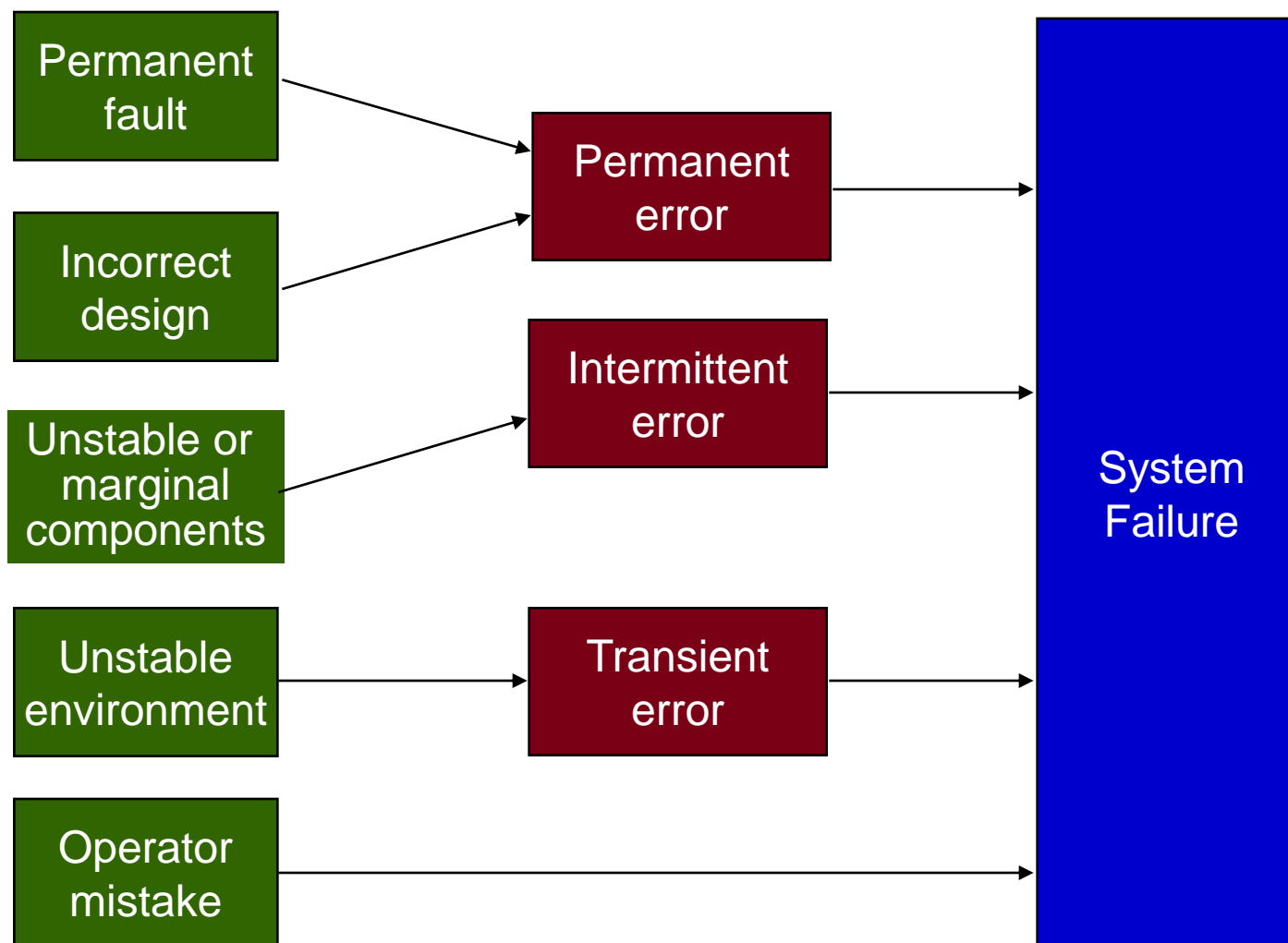
■ Hard faults

- Permanent
- Resulting failures are called hard failures

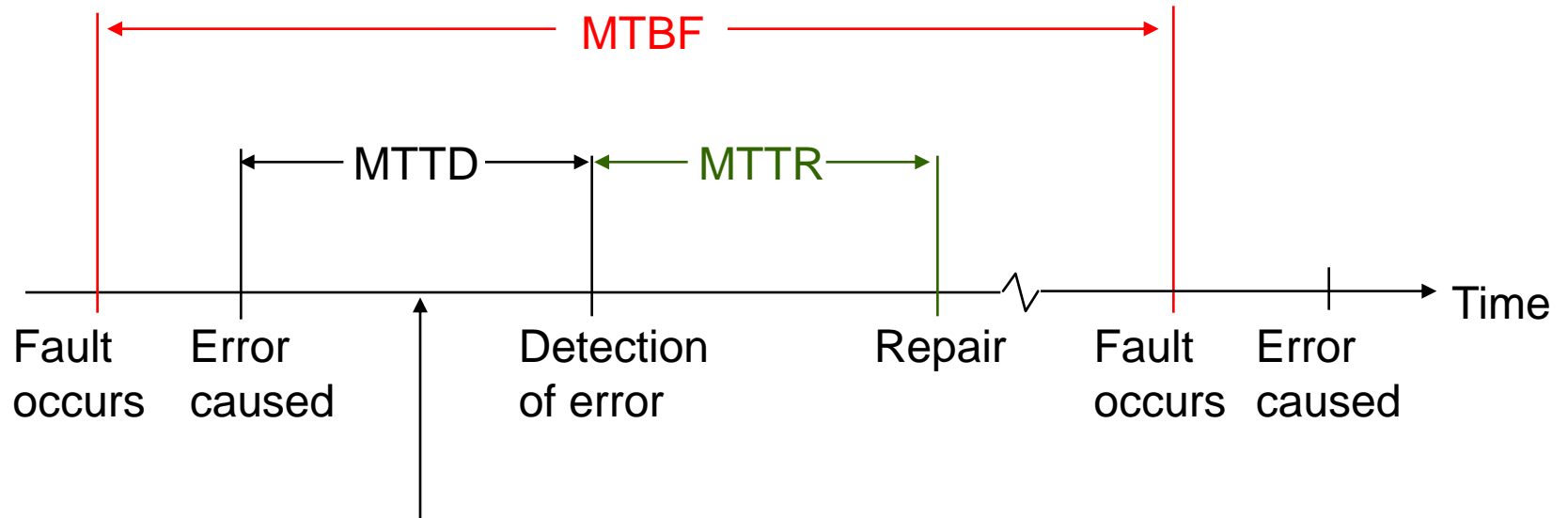
■ Soft faults

- Transient or intermittent
- Account for more than 90% of all failures
- Resulting failures are called soft failures

Fault Classification



Failures



Multiple errors can occur during this period

Fault Tolerance Measures

Reliability

$$R(t) = \Pr\{0 \text{ failures in time } [0,t] \mid \text{no failures at } t=0\}$$

If occurrence of failures is Poisson

$$R(t) = \Pr\{0 \text{ failures in time } [0,t]\}$$

Then

$$\Pr(k \text{ failures in time } [0,t]) = \frac{e^{-m(t)}[m(t)]^k}{k!}$$

where $m(t)$ is known as the *hazard function* which gives the time-dependent failure rate of the component and is defined as

$$m(t) = \int_0^t z(x)dx$$

Fault-Tolerance Measures

Reliability

The mean number of failures in time $[0, t]$ can be computed as

$$E[k] = \sum_{k=0}^{\infty} k \frac{e^{-m(t)} [m(t)]^k}{k!} = m(t)$$

and the variance can be computed as

$$\text{Var}[k] = E[k^2] - (E[k])^2 = m(t)$$

Thus, reliability of a single component is

$$R(t) = e^{-m(t)}$$

and of a system consisting of n non-redundant components as

$$R_{\text{sys}}(t) = \prod_{i=1}^n R_i(t)$$

Fault-Tolerance Measures

Availability

$$A(t) = \Pr\{\text{system is operational at time } t\}$$

Assume

- Poisson failures with rate λ
- Repair time is exponentially distributed with mean $1/\mu$

Then, steady-state availability

$$A = \lim_{t \rightarrow \infty} A(t) = \frac{\mu}{\lambda + \mu}$$

Fault-Tolerance Measures

MTBF

Mean time between failures

$$\text{MTBF} = \int_0^{\infty} R(t)dt$$

MTTR

Mean time to repair

Availability

$$\frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

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

How to Improve Dependability

■ Mask failures by redundancy

● Information redundancy

→ E.g., add extra bits to detect and recovered data transmission errors

● Time redundancy

→ Transactions; e.g., when a transaction aborts re-execute it without adverse effects.

● Physical redundancy

→ Hardware redundancy

- Take a distributed system with 4 file servers, each with a 0.95 chance of being up at any instant
- The probability of all 4 being down simultaneously is $0.05^4 = 0.000006$
- So the probability of at least one being available (i.e., the reliability of the full system) is 0.999994, far better than 0.95
- If there are 2 servers, then the reliability of the system is $(1 - 0.05^2) = 0.9975$

→ Software redundancy

- Process redundancy with similar considerations

■ A design that does not require simultaneous functioning of a substantial number of critical components.

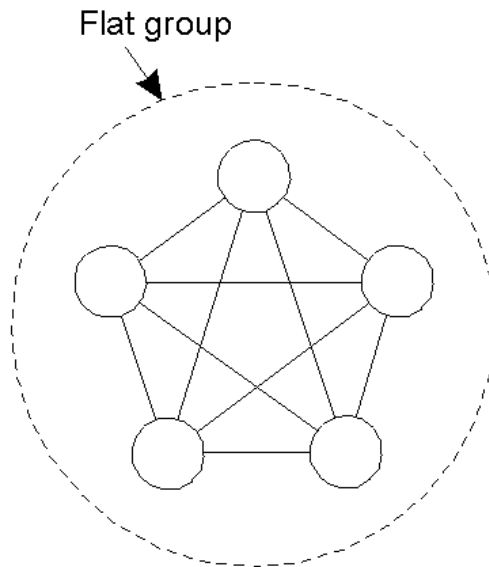
Hardware Redundancy

- Two computers are employed for a single application, one acting as a standby
 - Very costly, but often very effective solution
- Redundancy can be planned at a finer grain
 - Individual servers can be replicated
 - Redundant hardware can be used for non-critical activities when no faults are present
 - Redundant routes in network

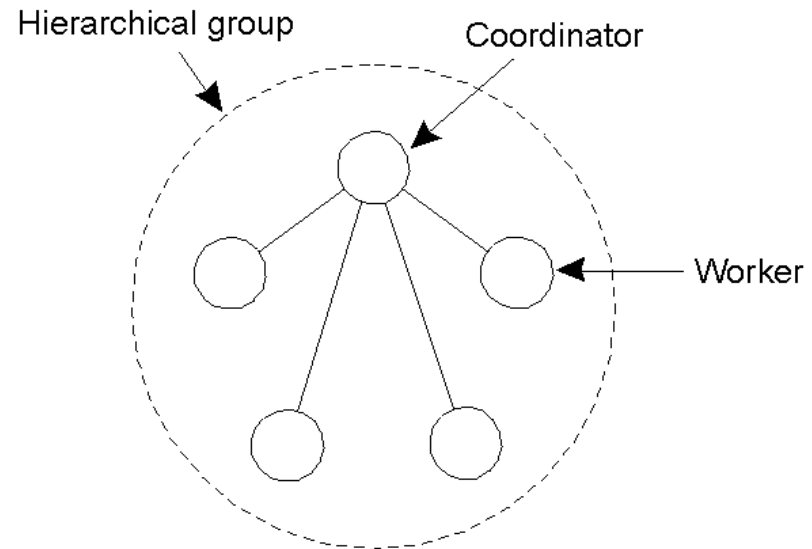
Process Redundancy

■ Process groups

- All members of a group receive a message sent to the group.
- If one process fails, others can take over.
- Can be dynamic; processes can have multiple memberships.
- Flat versus hierarchical groups:



(a)



(b)

Management of Replicated Processes

■ Primary copy

- Primary-backup setup
- Coordinator is the primary that coordinates all updates
- If coordinator fails, one backup takes over (usually through an election procedure)
- Processes are organized hierarchically

■ Replicated-writes

- Active replication and quorum-based protocols
- Flat group organization
- No single points of failure

Fault Tolerance of Process Groups

- A system is *k fault tolerant* if it can survive faults in k components and still meets its specification.
- If failures are safe (silent), then $k+1$ processes are sufficient to get k fault tolerance.
- In case of arbitrary failures, $2k+1$ processes are required (since k failing processes can all generate the same result by chance)
 - This assumes that each process reaches its decision independently
 - What if processes gang up to produce wrong results? General problem is having a process group reach an *agreement*.
 - In this case you need $2k+1$ correctly functioning processes, for a total of $3k+1$ processes.
- In both cases we assume that communication failures do not occur.

Communication Failures

- Point-to-point communication
 - TCP protocol masks the failures by acknowledgement and retransmission
- RPC and RMI semantics in the presence of failures
 - We studies these under the respective topics
 - See slides 3-13 to 3-19 for RPC
 - See slides 3-39 to 3-41 for RMI

Transactional Dependability

- Problem: How to maintain
 - atomicity
 - durabilityproperties of transactions
- Focus is on data: Failures that affect ACID properties of data
 - Transaction failures
 - Transaction aborts (unilaterally or due to deadlock)
 - Avg. 3% of transactions abort abnormally
 - System (site) failures
 - Failure of processor, main memory, power supply, ...
 - Main memory contents are lost, but secondary storage contents are safe
 - Partial vs. total failure
 - Media failures
 - Failure of secondary storage devices such that the stored data is lost
 - Head crash/controller failure (?)
 - Communication failures
 - Lost/undeliverable messages
 - Network partitioning

Distributed Reliability Protocols

■ Commit protocols

- How to execute commit command for distributed transactions.
- Issue: how to ensure atomicity and durability?
 - A transaction that executes at different sites has to complete (i.e., abort or commit) the same way everywhere

■ Termination protocols

- If a failure occurs, how can the remaining operational sites deal with it.
- **Non-blocking** : the occurrence of failures should not force the sites to wait until the failure is repaired to terminate the transaction.

■ Recovery protocols

- When a failure occurs, how do the sites where the failure occurred deal with it.
- **Independent** : a failed site can determine the outcome of a transaction without having to obtain remote information.

Two-Phase Commit (2PC)

Phase 1: The coordinator gets the participants ready to commit their writes

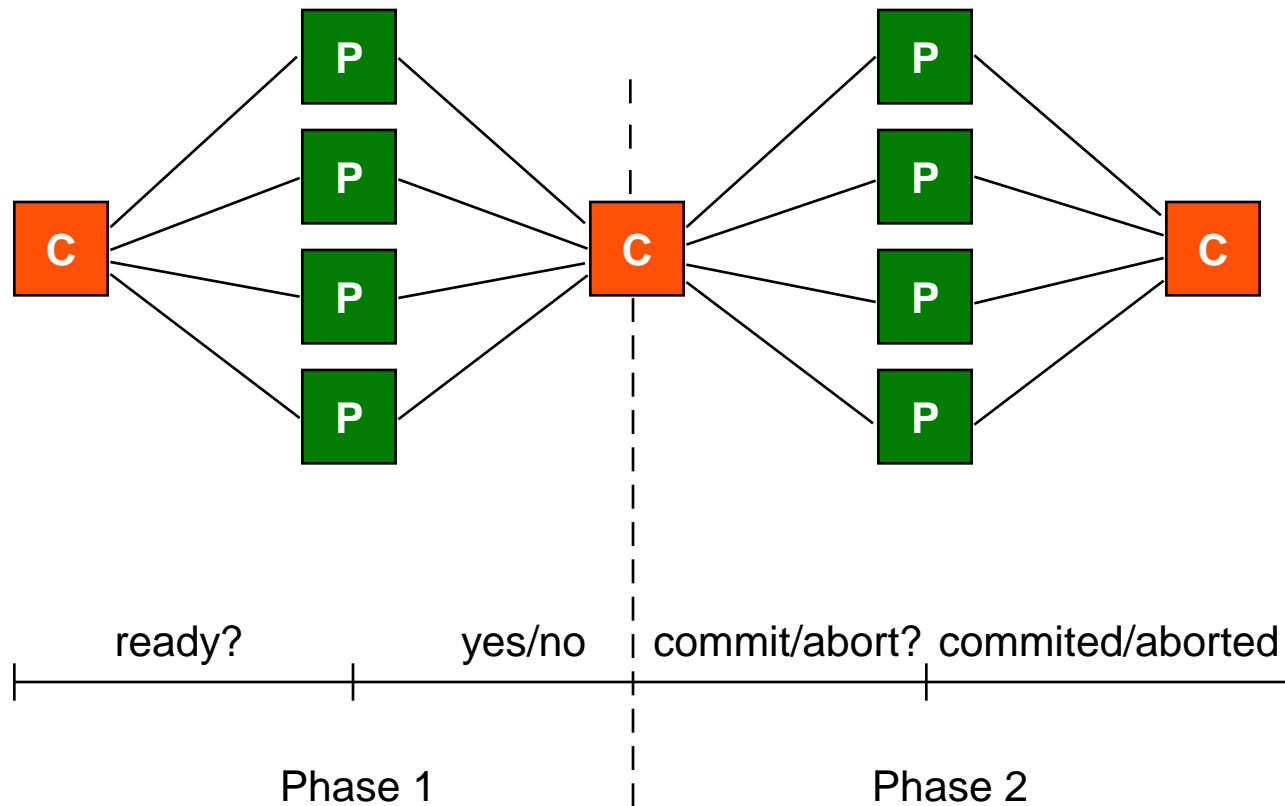
Phase 2: Everybody commits

- **Coordinator** :The process at the site where the transaction originates and which controls the execution
- **Participant** :The process at the other sites that participate in executing the transaction

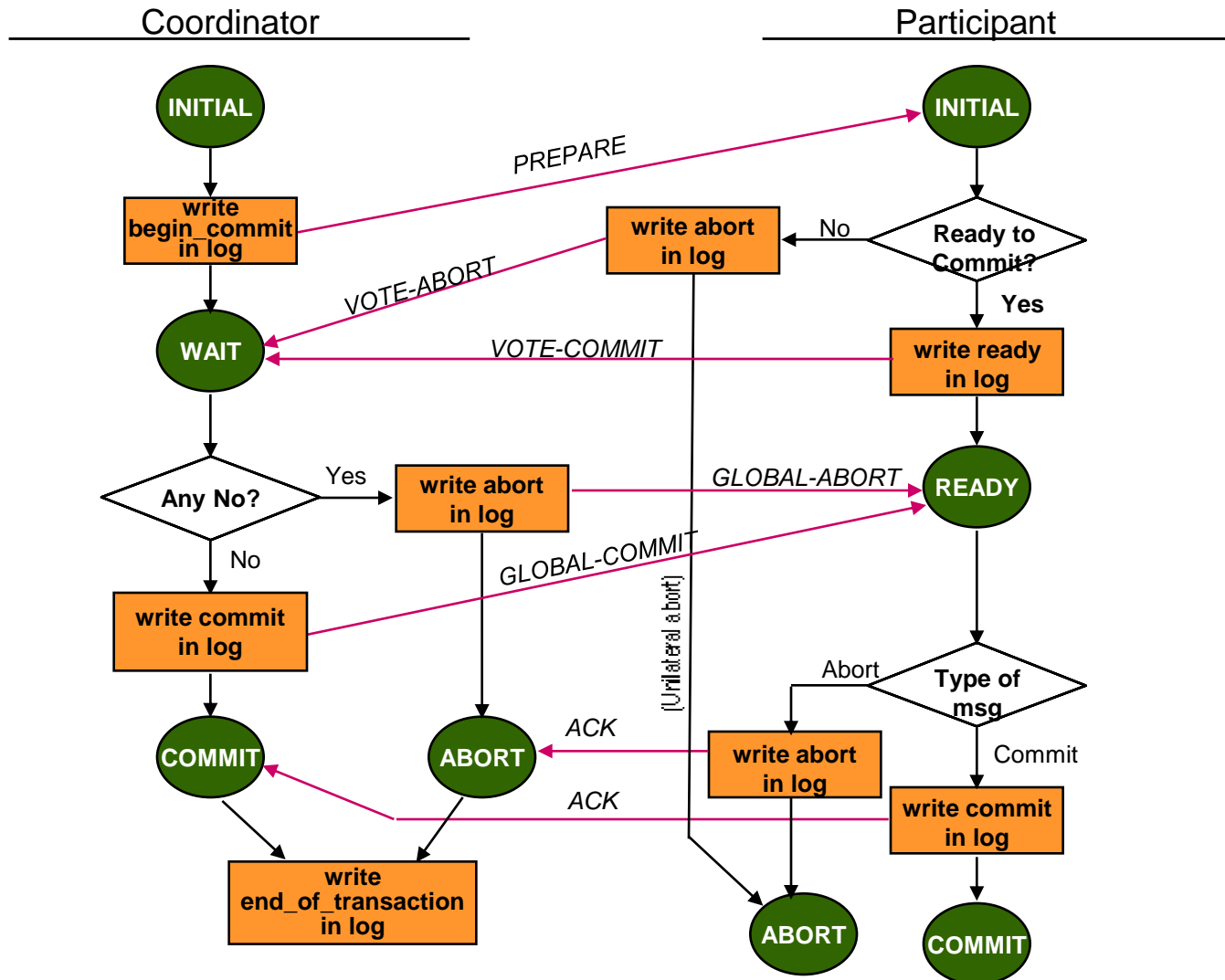
Global Commit Rule:

- ☆ The coordinator aborts a transaction if and only if at least one participant votes to abort it.
- 🕒 The coordinator commits a transaction if and only if all of the participants vote to commit it.

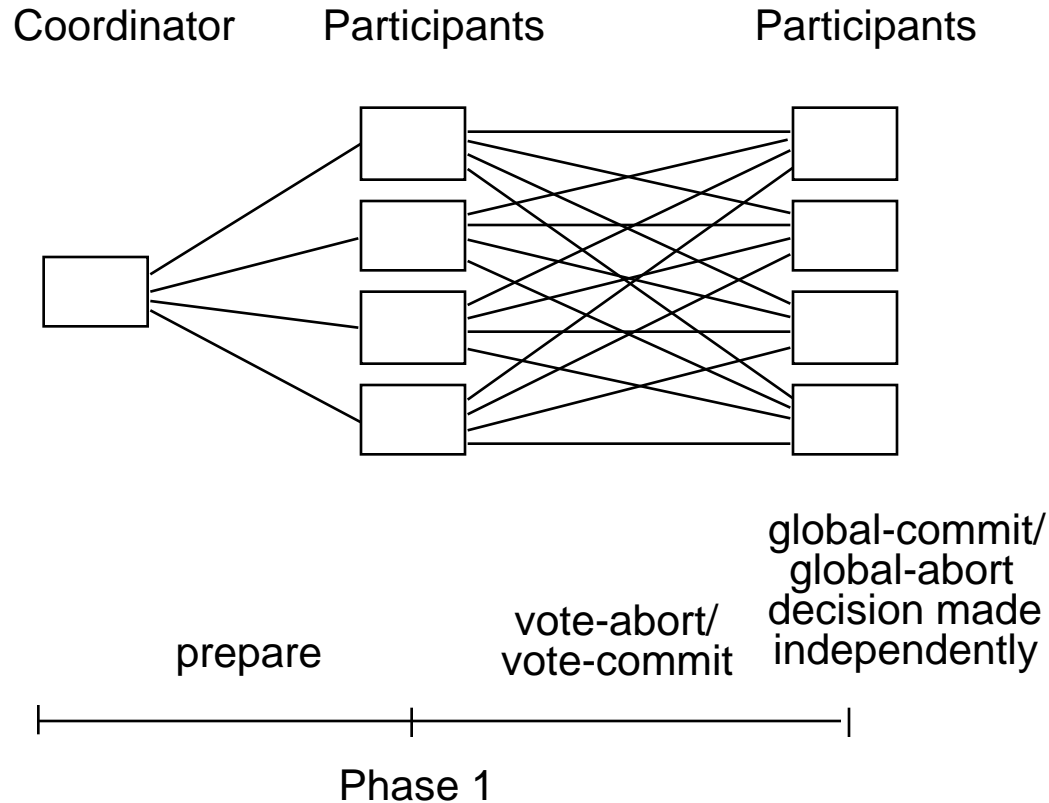
Centralized 2PC



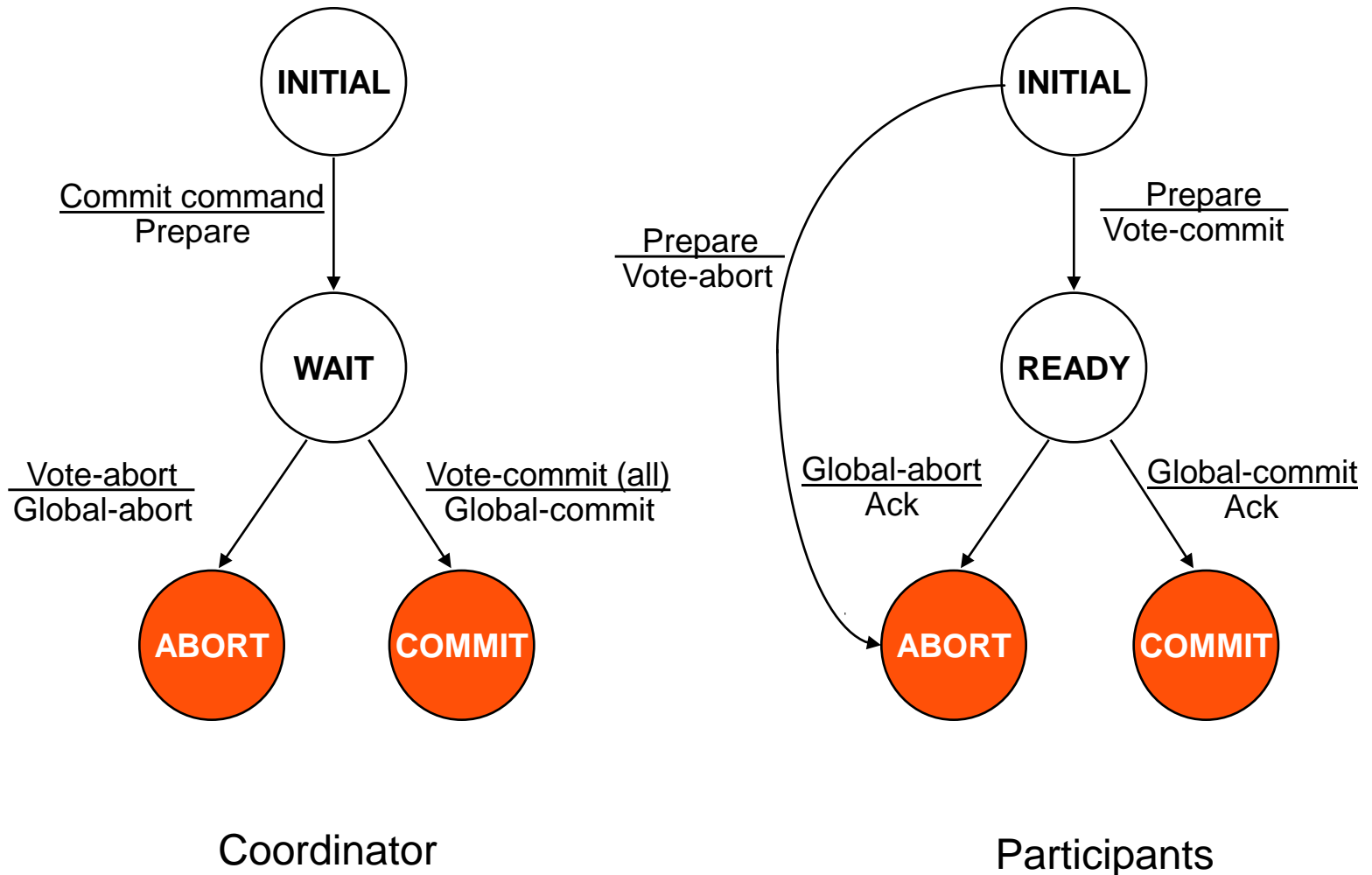
2PC Protocol Actions



Distributed 2PC

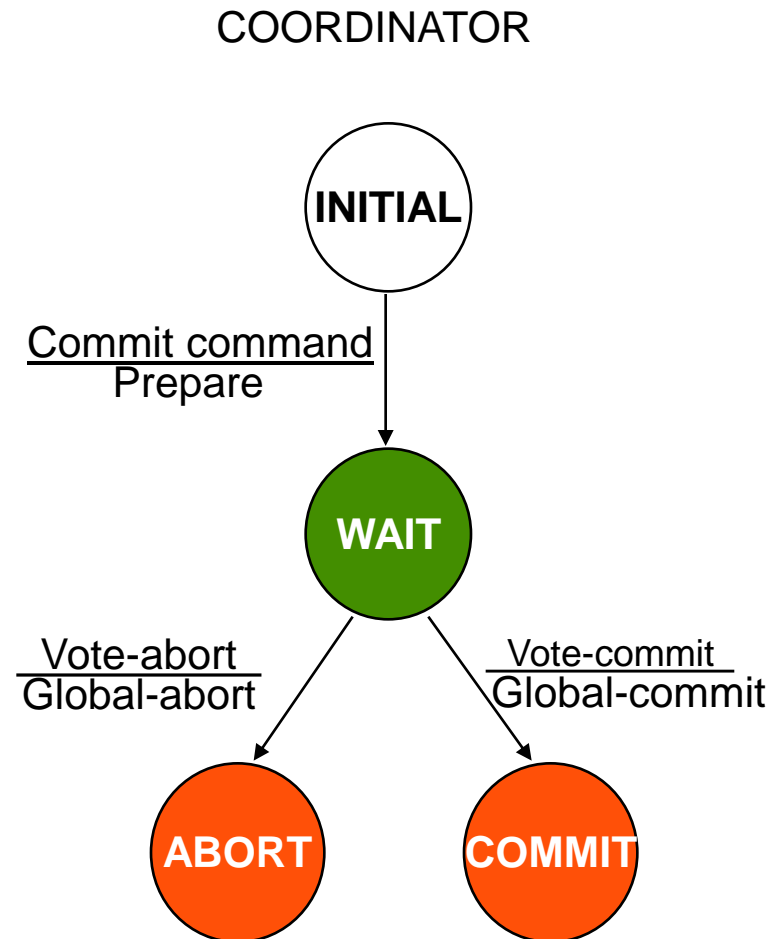


State Transitions in 2PC



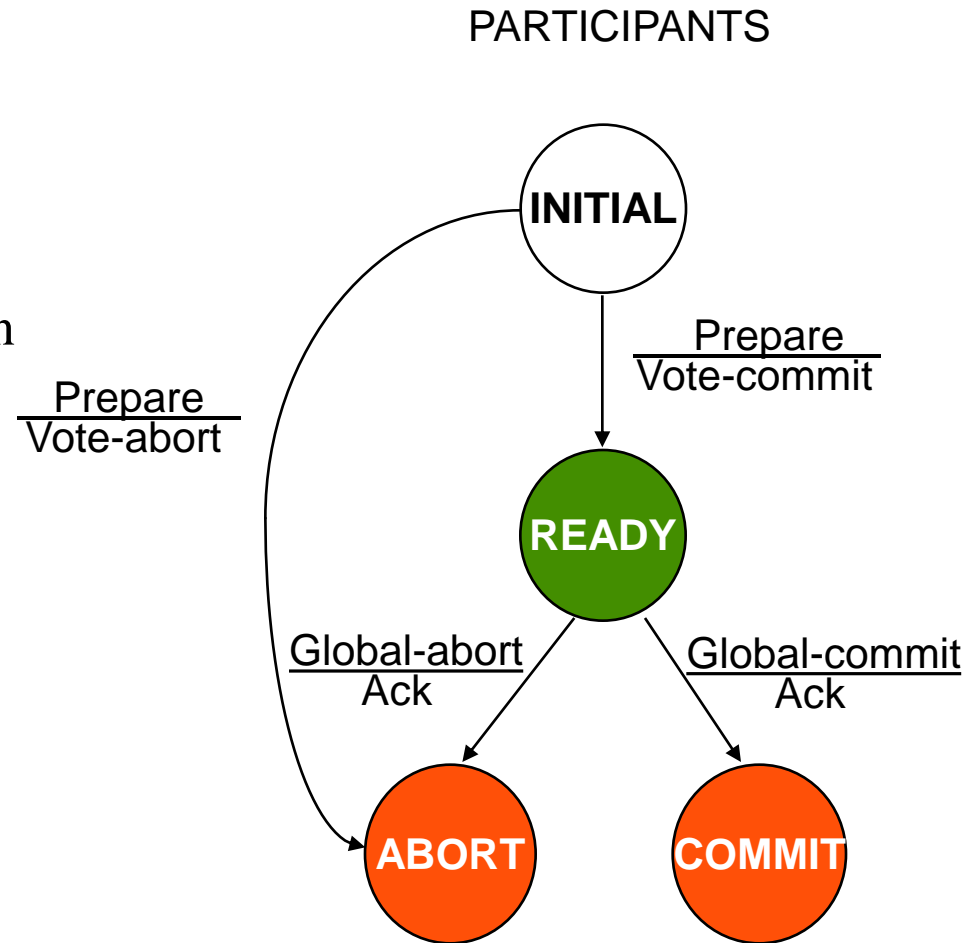
Site Failures - 2PC Termination

- Timeout in INITIAL
 - Who cares
- Timeout in WAIT
 - Cannot unilaterally commit
 - Can unilaterally abort
- Timeout in ABORT or COMMIT
 - Stay blocked and wait for the acks



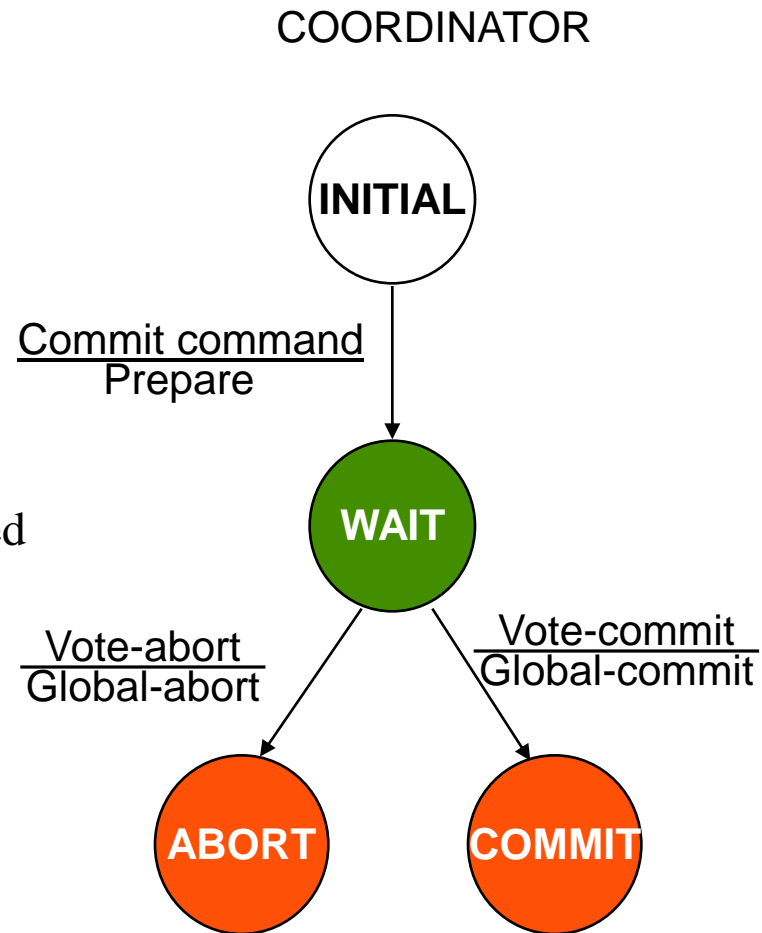
Site Failures - 2PC Termination

- Timeout in INITIAL
 - Coordinator must have failed in INITIAL state
 - Unilaterally abort
- Timeout in READY
 - Stay blocked



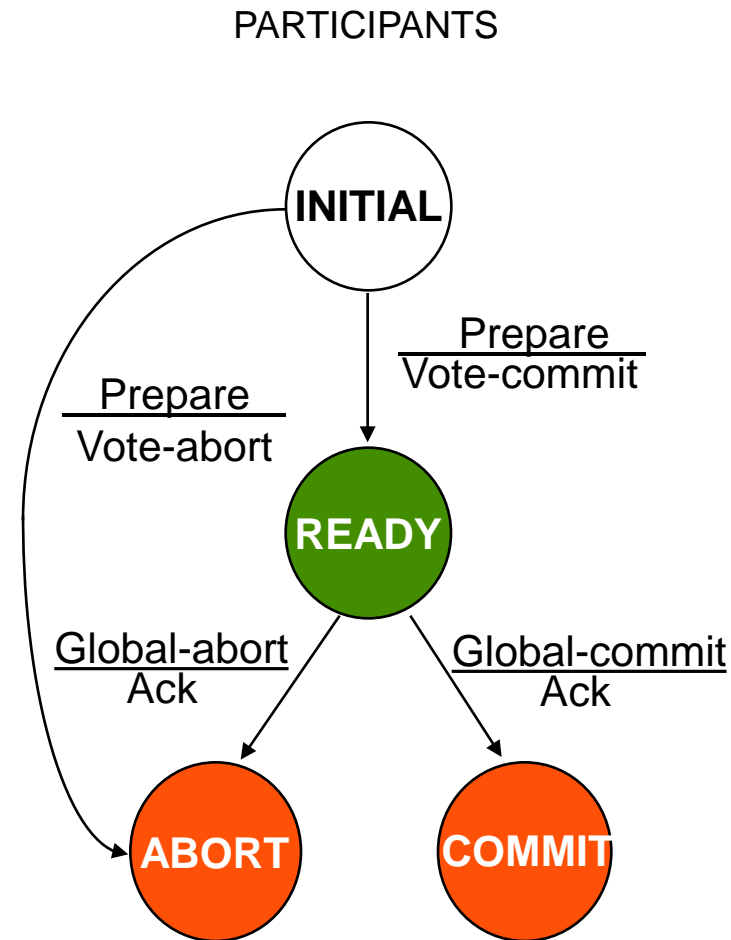
Site Failures - 2PC Recovery

- Failure in INITIAL
 - Start the commit process upon recovery
- Failure in WAIT
 - Restart the commit process upon recovery
- Failure in ABORT or COMMIT
 - Nothing special if all the acks have been received
 - Otherwise the termination protocol is involved



Site Failures - 2PC Recovery

- Failure in INITIAL
 - Unilaterally abort upon recovery
- Failure in READY
 - The coordinator has been informed about the local decision
 - Treat as timeout in READY state and invoke the termination protocol
- Failure in ABORT or COMMIT
 - Nothing special needs to be done



2PC Recovery Protocols – Additional Cases

- Arise due to non-atomicity of log and message send actions
- Coordinator site fails after writing “begin_commit” log and before sending “prepare” command
 - treat it as a failure in WAIT state; send “prepare” command
- Participant site fails after writing “ready” record in log but before “vote-commit” is sent
 - treat it as failure in READY state
 - alternatively, can send “vote-commit” upon recovery
- Participant site fails after writing “abort” record in log but before “vote-abort” is sent
 - no need to do anything upon recovery

2PC Recovery Protocols – Additional Case

- Coordinator site fails after logging its final decision record but before sending its decision to the participants
 - coordinator treats it as a failure in COMMIT or ABORT state
 - participants treat it as timeout in the READY state
- Participant site fails after writing “abort” or “commit” record in log but before acknowledgement is sent
 - participant treats it as failure in COMMIT or ABORT state
 - coordinator will handle it by timeout in COMMIT or ABORT state

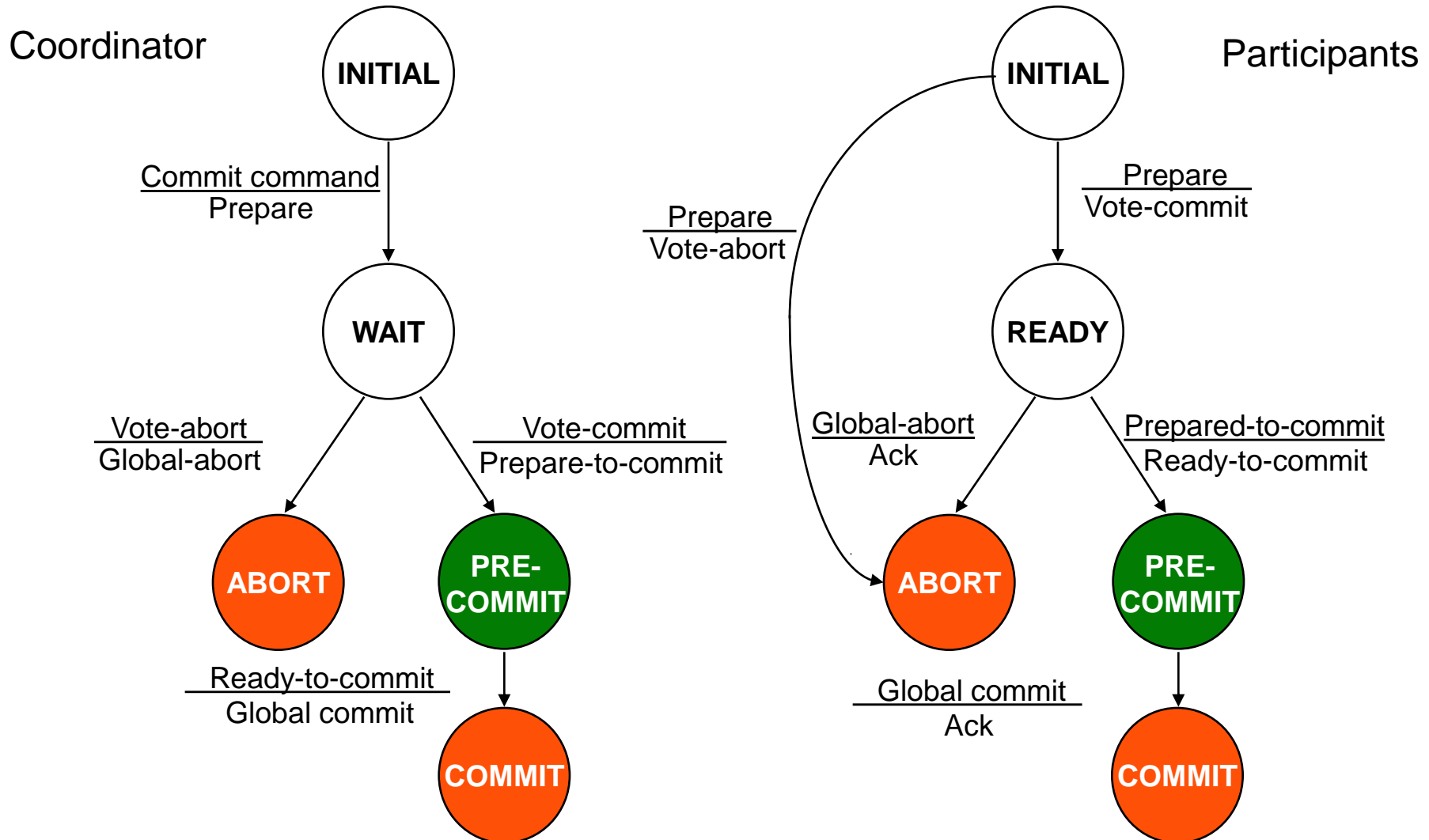
Problem With 2PC

- Blocking
 - Ready implies that the participant waits for the coordinator
 - If coordinator fails, site is blocked until recovery
 - Blocking reduces availability
- Independent recovery is not possible
- However, it is known that:
 - Independent recovery protocols exist only for single site failures; no independent recovery protocol exists which is resilient to multiple-site failures.
- So we search for these protocols – 3PC

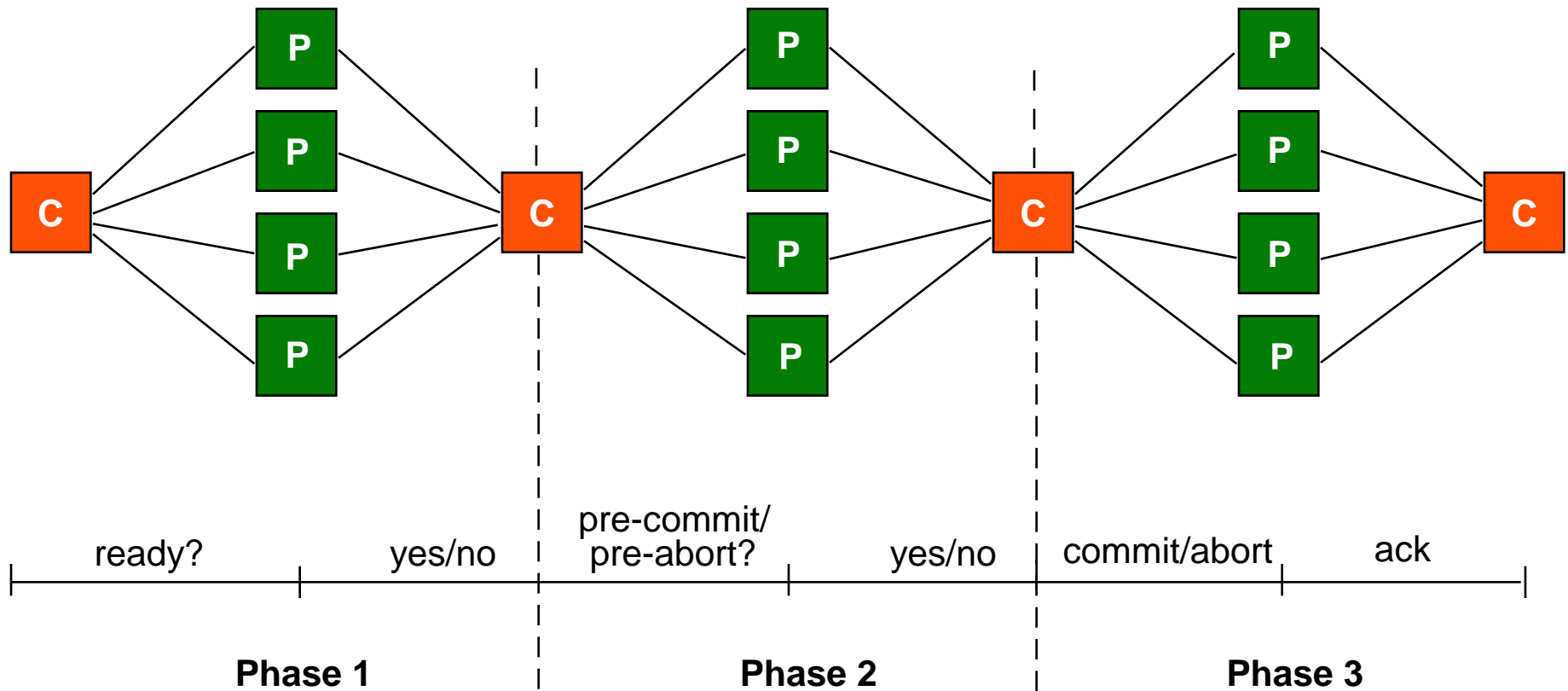
Three-Phase Commit

- 3PC is non-blocking.
- A commit protocols is non-blocking iff
 - it is synchronous within one state transition, and
 - its state transition diagram contains
 - no state which is “adjacent” to both a commit and an abort state, and
 - no non-committable state which is “adjacent” to a commit state
- Adjacent: possible to go from one state to another with a single state transition
- Committable: all sites have voted to commit a transaction
 - e.g.: COMMIT state

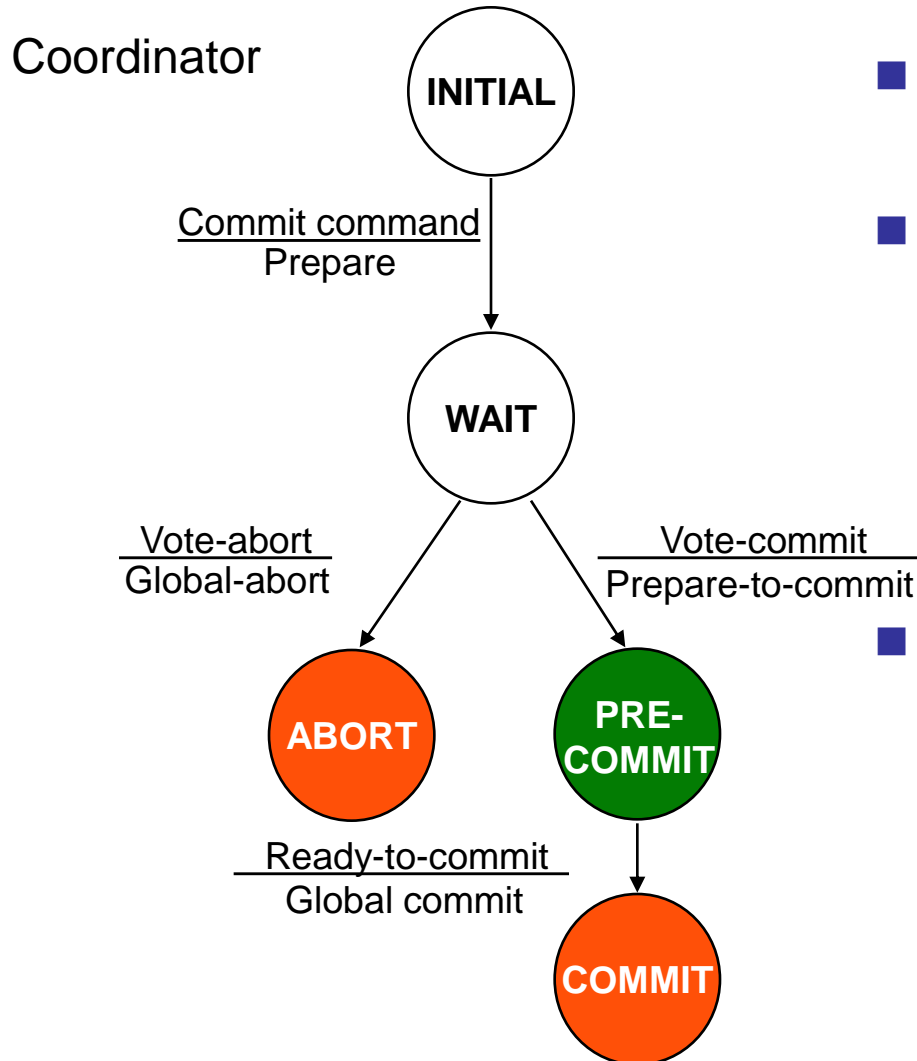
State Transitions in 3PC



Communication Structure

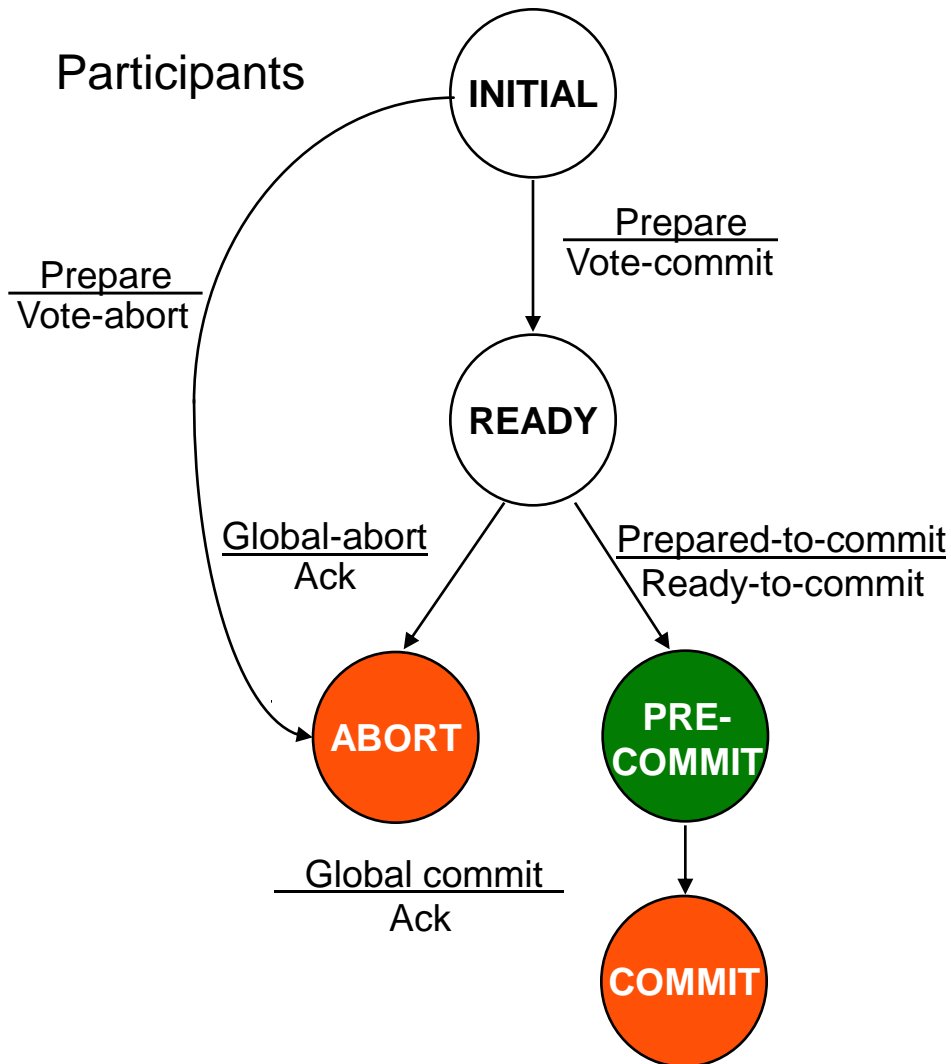


Site Failures – 3PC Termination



- Timeout in WAIT
 - Unilaterally abort
- Timeout in PRECOMMIT
 - Participants may not be in PRE-COMMIT, but at least in READY
 - Move all the participants to PRECOMMIT state
 - Terminate by globally committing
- Timeout in ABORT or COMMIT
 - Just ignore and treat the transaction as completed
 - participants are either in PRECOMMIT or READY state and can follow their termination protocols

Site Failures – 3PC Termination



■ Timeout in INITIAL

- Coordinator must have failed in INITIAL state
- Unilaterally abort

■ Timeout in READY

- Voted to commit, but does not know the coordinator's decision
- Elect a new coordinator and terminate using a special protocol

■ Timeout in PRECOMMIT

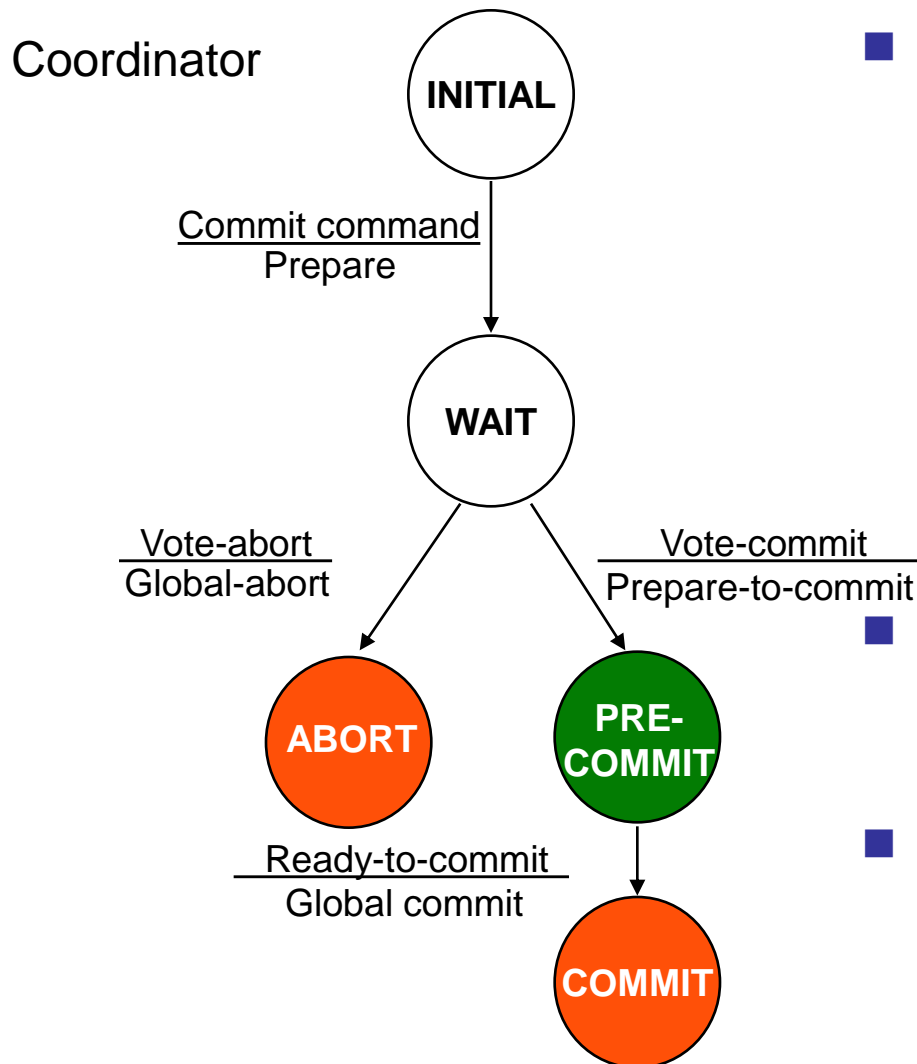
- Handle it the same as timeout in READY state

Termination Protocol Upon Coordinator Election

New coordinator can be in one of four states: WAIT, PRECOMMIT, COMMIT, ABORT

- ☆ Coordinator sends its state to all of the participants asking them to assume its state.
- 🕒 Participants “back-up” and reply with appropriate messages, except those in ABORT and COMMIT states. Those in these states respond with “Ack” but stay in their states.
- 🕒 Coordinator guides the participants towards termination:
 - If the new coordinator is in the WAIT state, participants can be in INITIAL, READY, ABORT or PRECOMMIT states. New coordinator globally aborts the transaction.
 - If the new coordinator is in the PRECOMMIT state, the participants can be in READY, PRECOMMIT or COMMIT states. The new coordinator will globally commit the transaction.
 - If the new coordinator is in the ABORT or COMMIT states, at the end of the first phase, the participants will have moved to that state as well.

Site Failures – 3PC Recovery



■ Failure in WAIT

- the participants may have elected a new coordinator and terminated the transaction
- the new coordinator could be in WAIT or ABORT states ⇐ transaction aborted
- ask around for the fate of the transaction

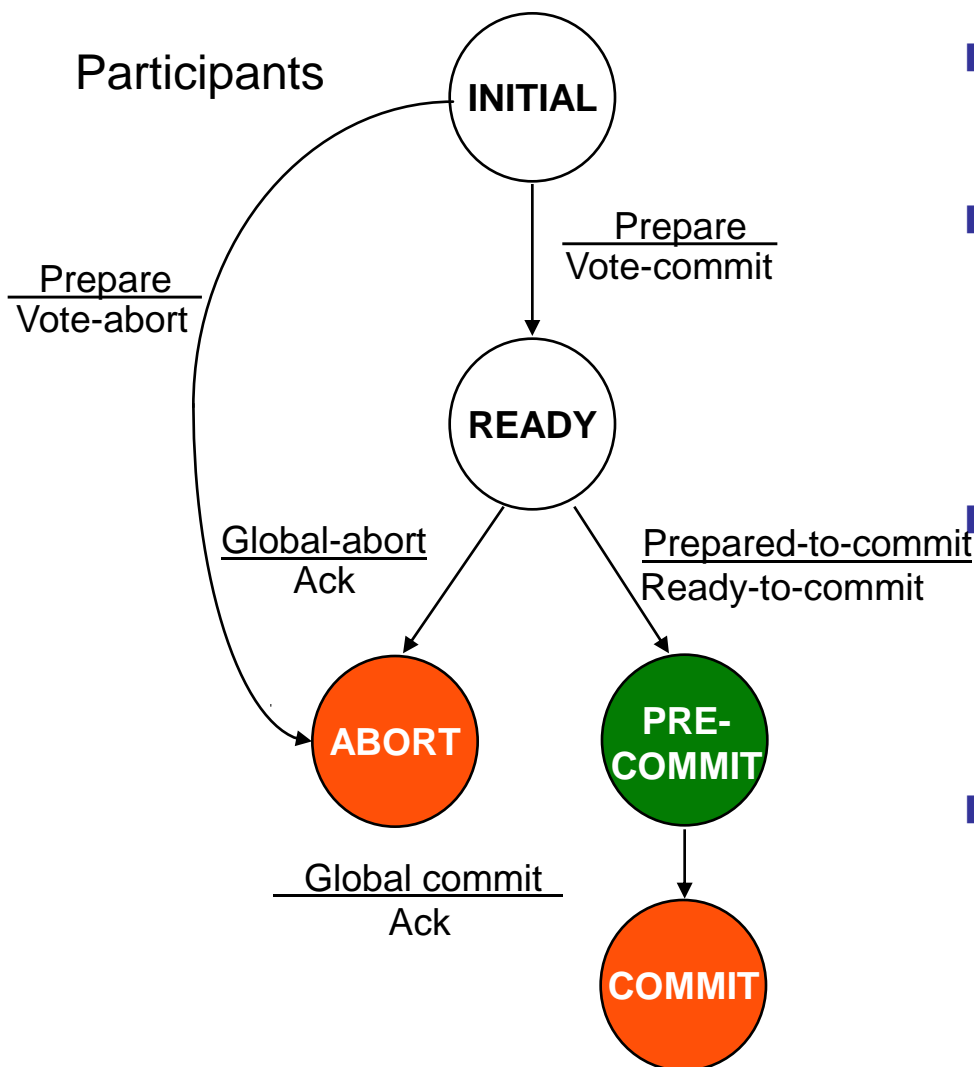
■ Failure in PRECOMMIT

- ask around for the fate of the transaction

■ Failure in COMMIT or ABORT

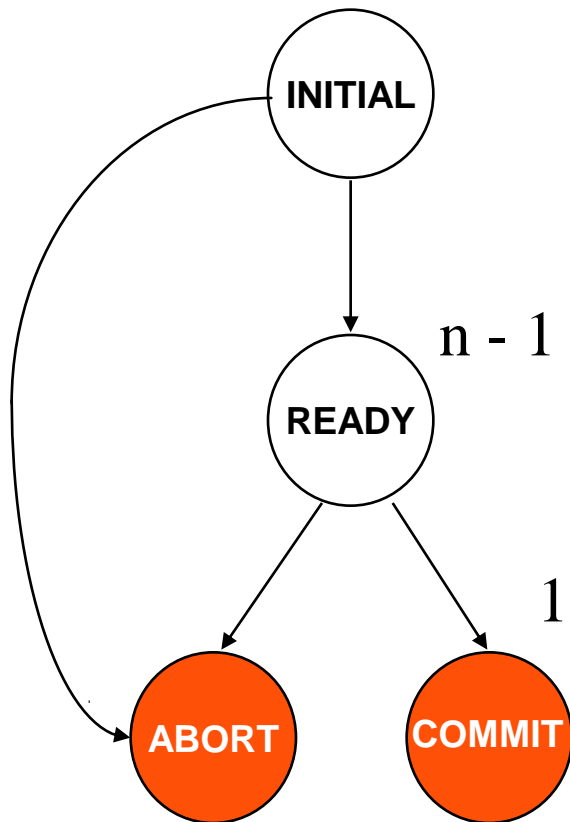
- Nothing special if all the acknowledgements have been received; otherwise the termination protocol is involved

Site Failures – 3PC Recovery



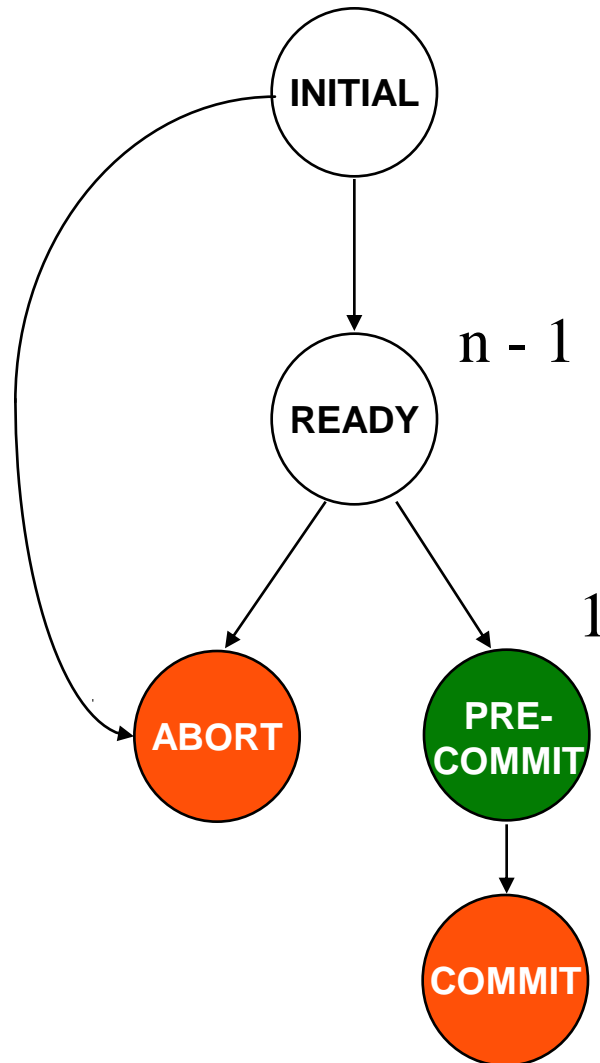
- Failure in INITIAL
 - unilaterally abort upon recovery
- Failure in READY
 - the coordinator has been informed about the local decision
 - upon recovery, ask around
- Failure in PRECOMMIT
 - ask around to determine how the other participants have terminated the transaction
- Failure in COMMIT or ABORT
 - no need to do anything

Intuition Behind 3PC



- For 2PC, it is possible that a coordinator failure can result in $n-1$ of the participants being in **READY** state and 1 participant in **COMMIT** state.
- The system **MUST** commit at this point. Therefore, it is not safe to abort the $n-1$ participants in the **READY** state if the participant in **COMMIT** also fails.

Intuition Behind 3PC



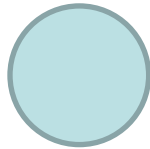
- In 3PC a coordinator failure can leave the system with $n-1$ participants in **READY** and 1 in **PRE-COMMIT**.
- Even if the participant in **PRE-COMMIT** fails, the system can abort as the participant in the **PRE-COMMIT** state can still abort.
- If instead there is any alive participants in **PRE-COMMIT**, then the transaction can commit without blocking on the failed nodes

Network Partitioning

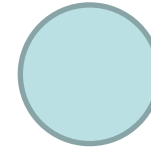
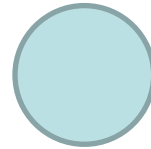
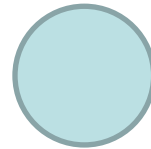
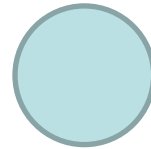
- Simple partitioning
 - Only two partitions
- Multiple partitioning
 - More than two partitions
- Formal bounds (due to Skeen):
 - There exists no non-blocking protocol that is resilient to a network partition if messages are lost when partition occurs.
 - There exist non-blocking protocols which are resilient to a single network partition if all undeliverable messages are returned to sender.
 - There exists no non-blocking protocol which is resilient to a multiple partition.

Transactions and Network Partitions

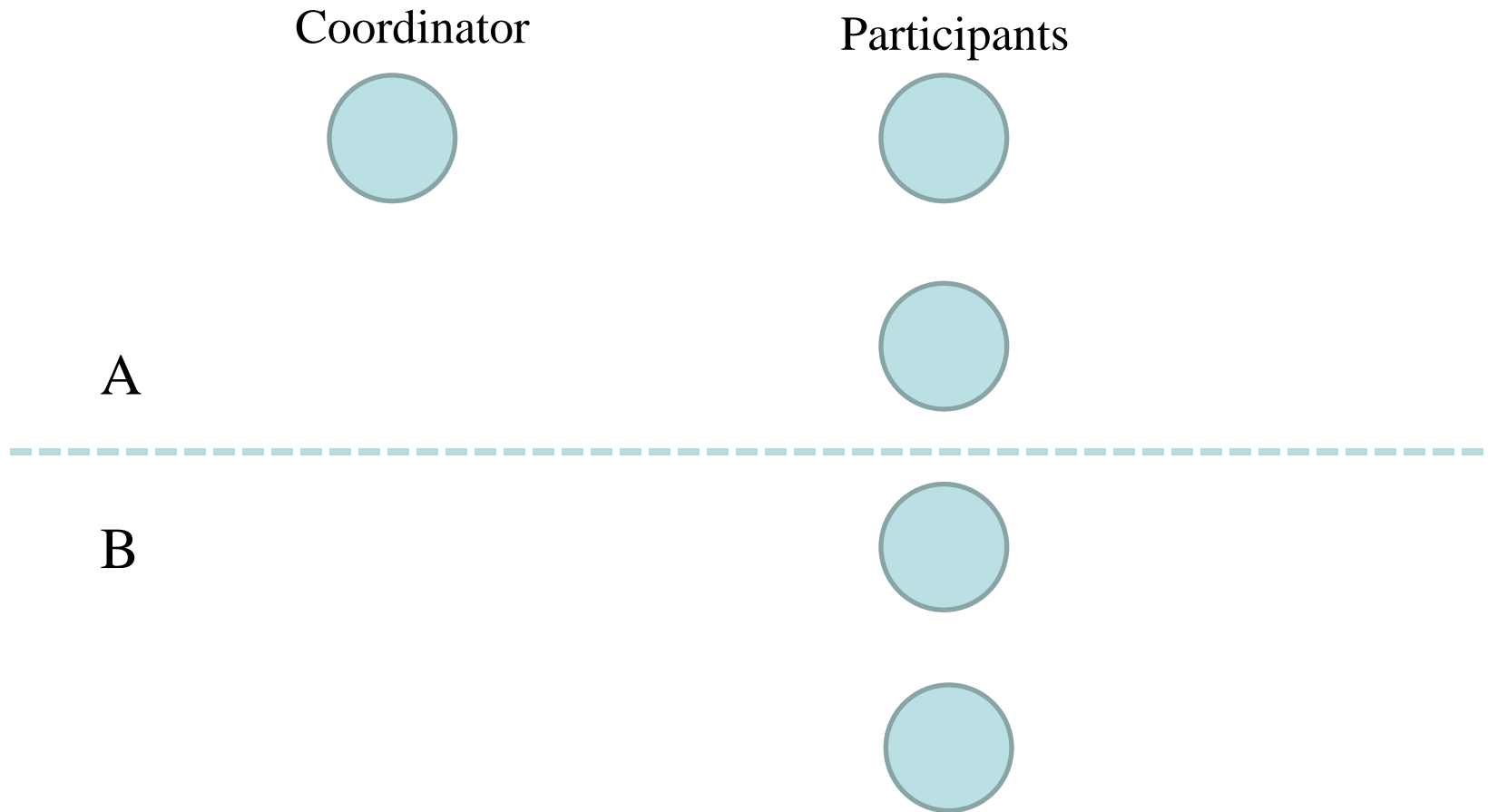
Coordinator



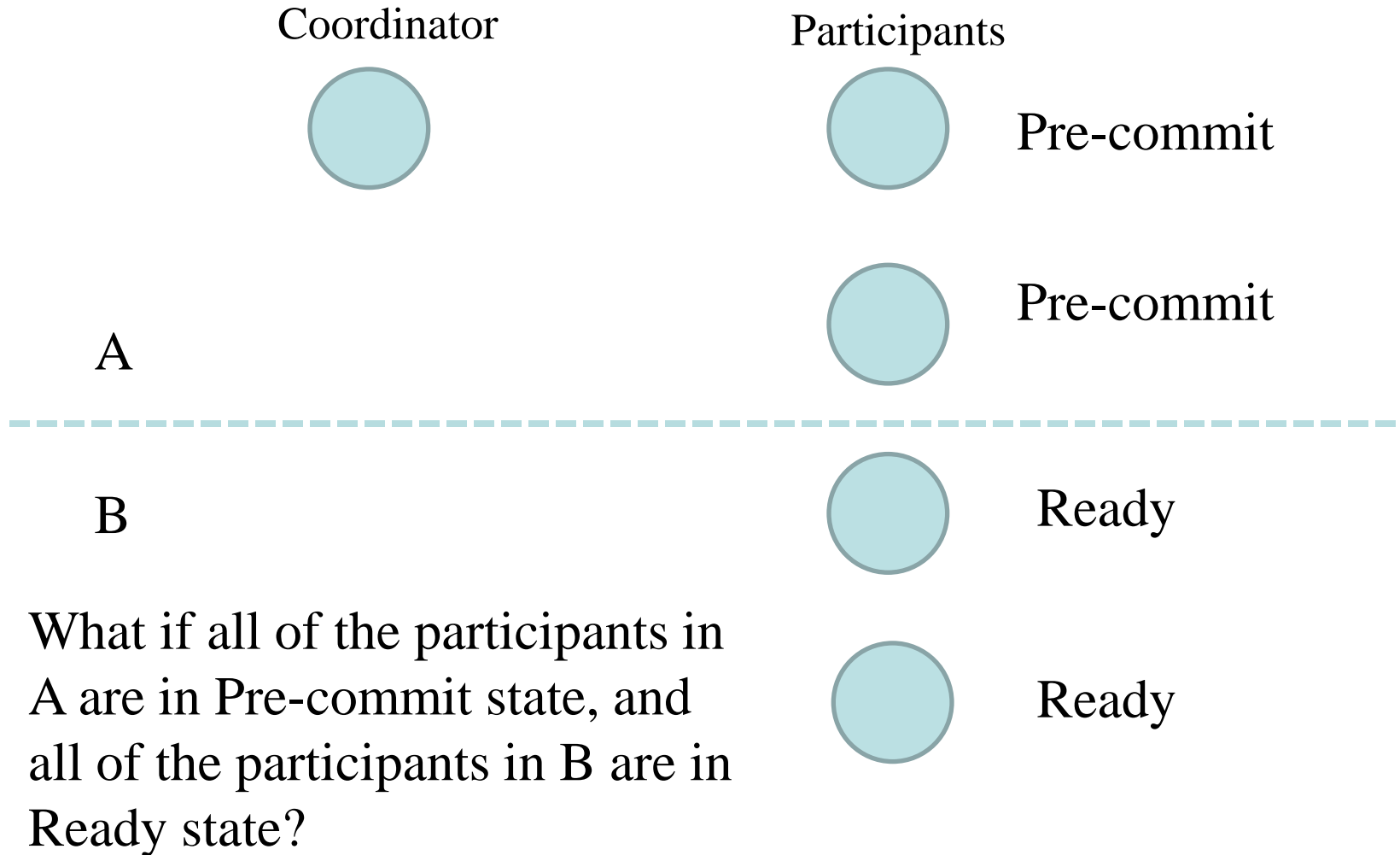
Participants



Transactions and Network Partitions



Transactions and Network Partitions



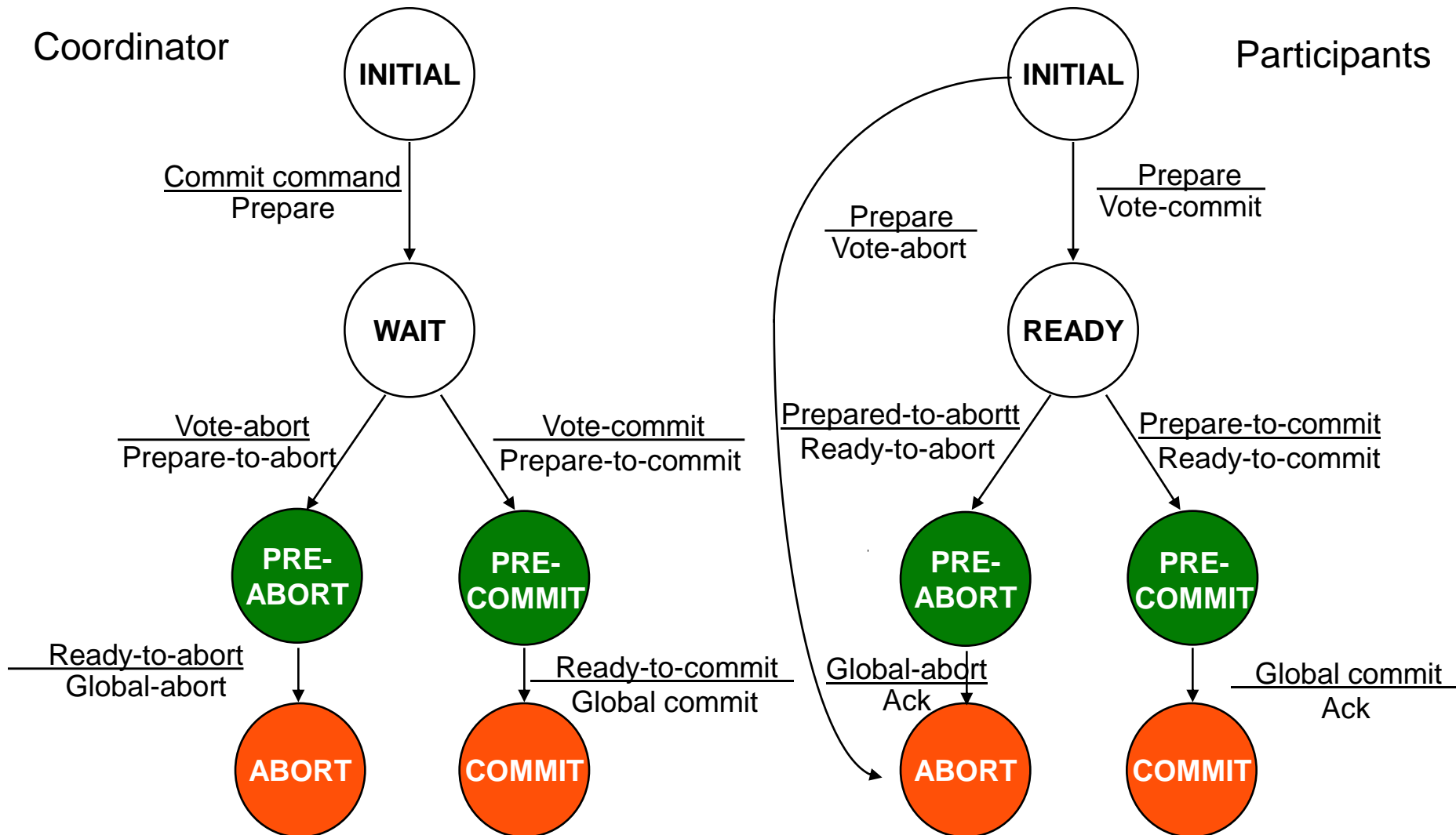
Independent Recovery in the case of Network Partitioning

- No general solution possible
 - allow one group to terminate while the other is blocked
 - improve availability
- How to determine which group to proceed?
 - The group with a majority
- How does a group know if it has majority?
 - centralized
 - whichever partitions contains the central site should terminate the transaction
 - voting-based (quorum)
 - different for replicated vs non-replicated databases

Quorum Protocols for Network Partitioning

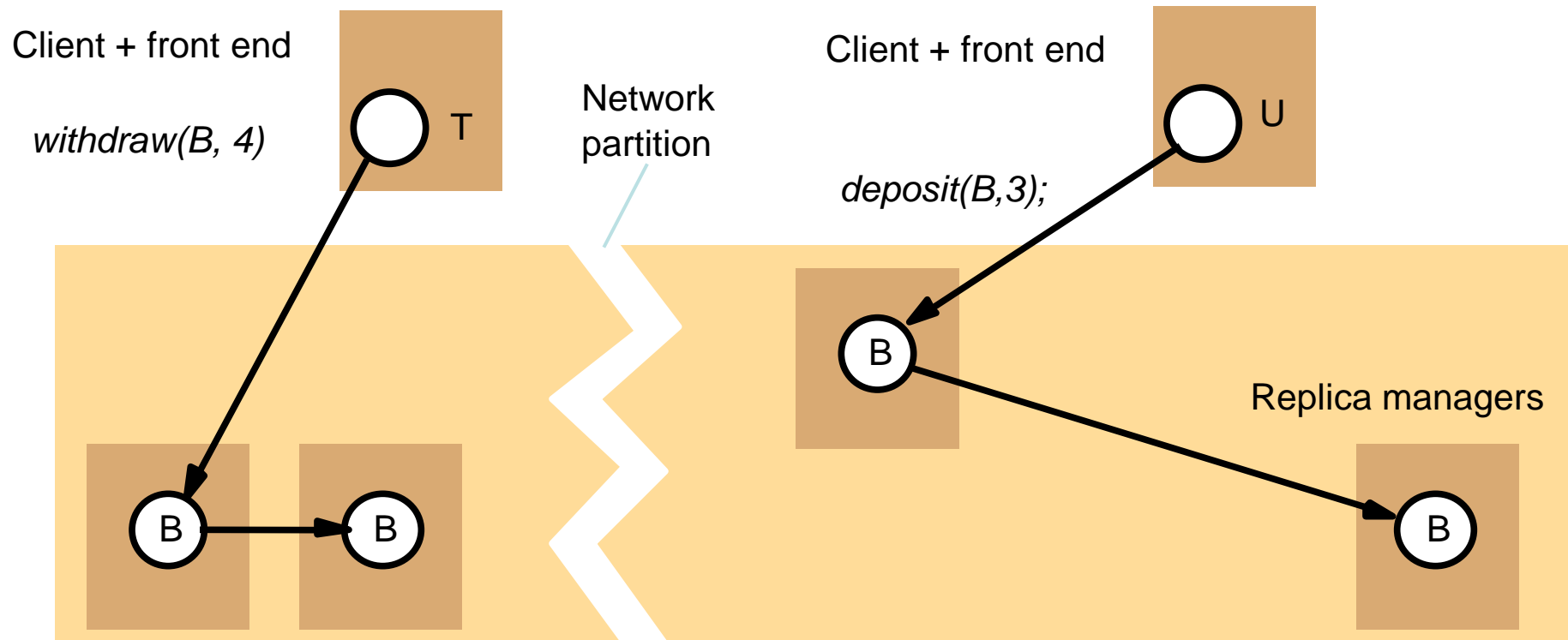
- The network partitioning problem is handled by the commit protocol.
- Every site is assigned a vote V_i .
- Total number of votes in the system V
- Abort quorum V_a , commit quorum V_c
 - $V_a + V_c > V$ where $0 \leq V_a, V_c \leq V$
 - Before a transaction commits, it must obtain a commit quorum V_c
 - Before a transaction aborts, it must obtain an abort quorum V_a

State Transitions in Quorum Protocols



Network Partitioning & Replication

- A group of replica managers are partitioned into two or more subgroups such that members of one subgroup can communicate with each other but members of different subgroups cannot communicate with one another.



Replica Management with Network Partitioning

■ Optimistic approach

- Available copies with validation
 - Let each partition perform updates freely
 - Validate when partition is repaired and those that violate one-copy serializability are aborted.

■ Pessimistic

- Quorum-based
 - Reduce availability (even when there is no partitioning)
 - Updates can only occur in the partition that has a majority of the replica managers