Distributed Transaction Management

Distributed Reliability

Reliability

Problem:

How to maintain

atomicity

durability

properties of transactions

Failures

Contents

- Failures in Distributed DBMS
  - Transaction, Site (System), Media, and Communication Failures
- Local Reliability Protocols
  - Architectural Considerations
  - Recovery Information

Contents

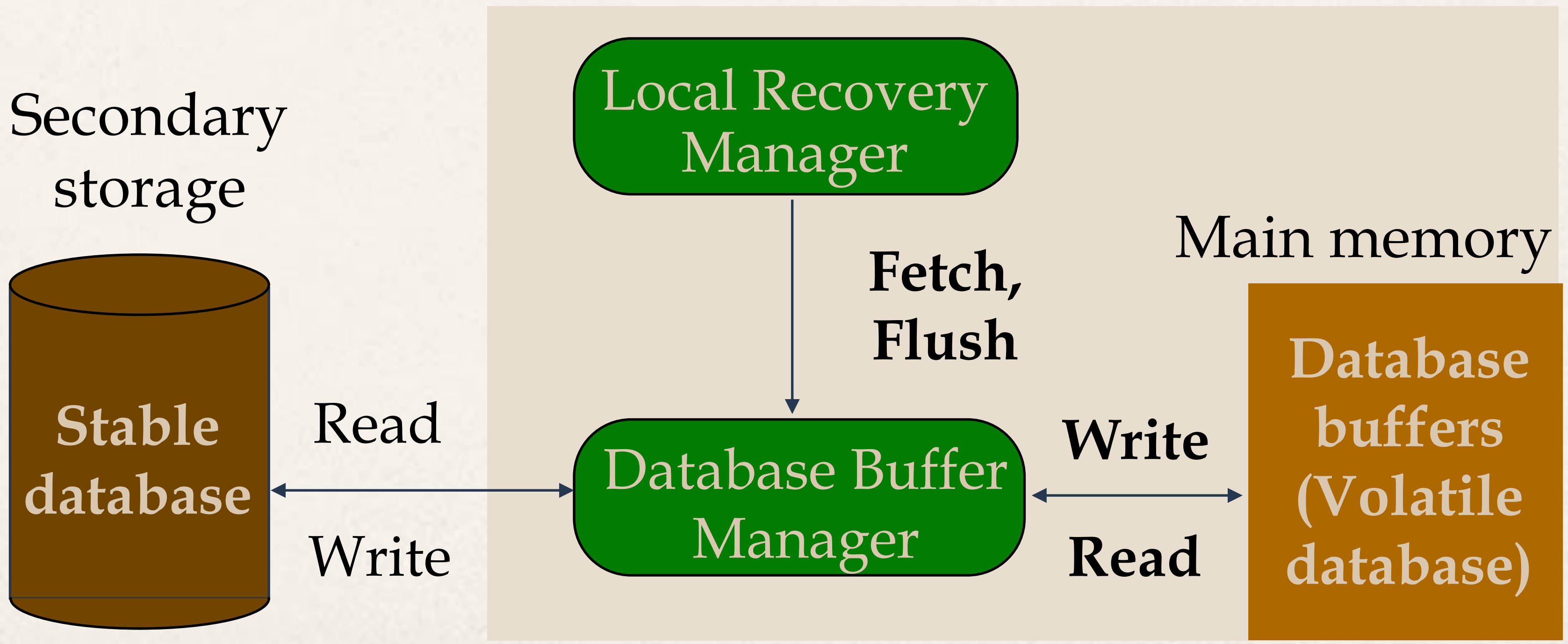
- Distributed Reliability Protocols
  - Two-Phase Commit Protocol
  - Variations of 2PC
- Dealing with Site Failures
  - Termination and Recovery Protocols for 2PC
  - Three-Phase Commit Protocol
- Network Partitioning

Failures in Distributed DBMS

- Transaction failures
  - **Transaction aborts** (unilaterally or due to deadlock)
- Site failures
  - **Crash - recover**
  - Failure of processor, main memory, power supply, …
  - Main memory contents are lost, but secondary storage contents are safe
  - Partial vs. total failure
- Media (disk) failures
  - Failure of secondary storage devices such that the stored data is lost
  - Head crash/controller failure
- Communication failures
  - Line failures
  - **Network partitioning**

Local Reliability Protocols

- Volatile storage
  - Consists of the main memory of the computer system (RAM).
- Stable storage
  - Implemented via a combination of hardware (RAID 1) and software (stable-write, stable-read, clean-up) components.



Architecture

Recovery Information

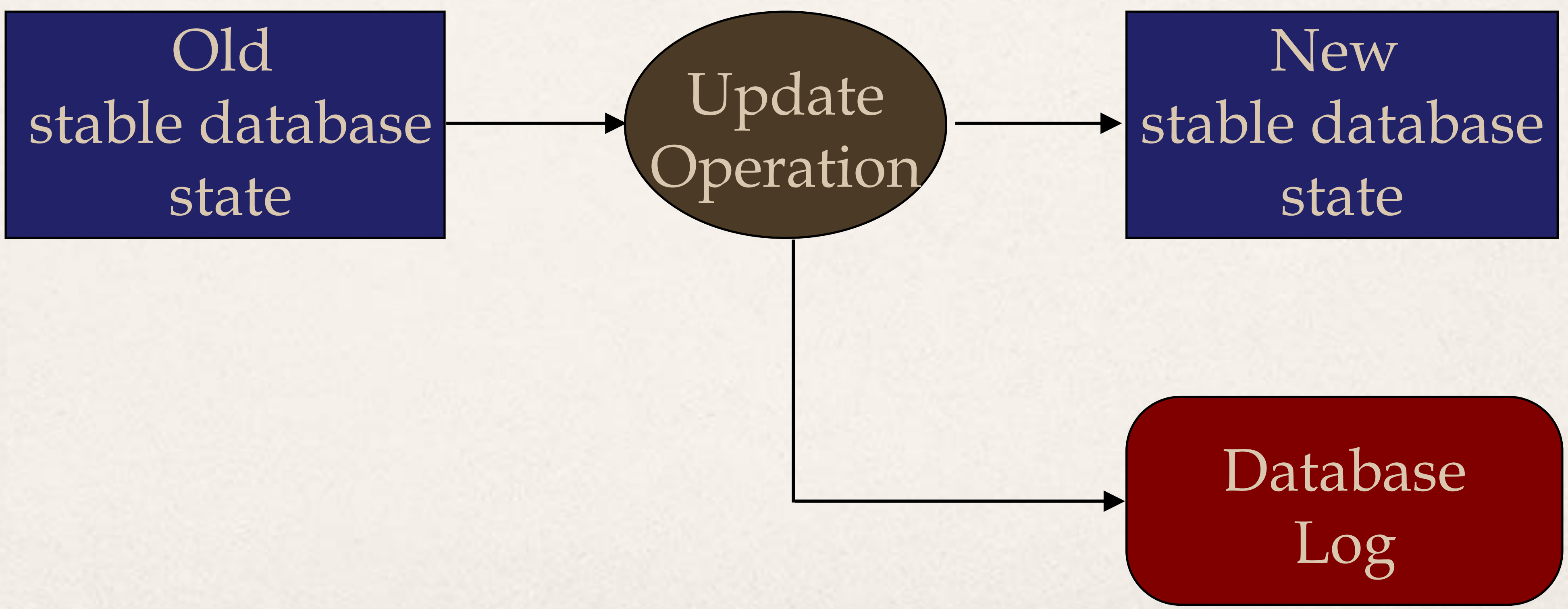
- When a system failure occurs, the volatile database is lost.
- The recovery information that the system maintains is dependent on the method of executing updates.

- In-place update
  - Each update causes a change in the value of the data item in the stable database
- Out-of-place update
  - Each update causes the new value of the data item to be stored separate from the old value

In-Place Update Recovery Information

Database Log

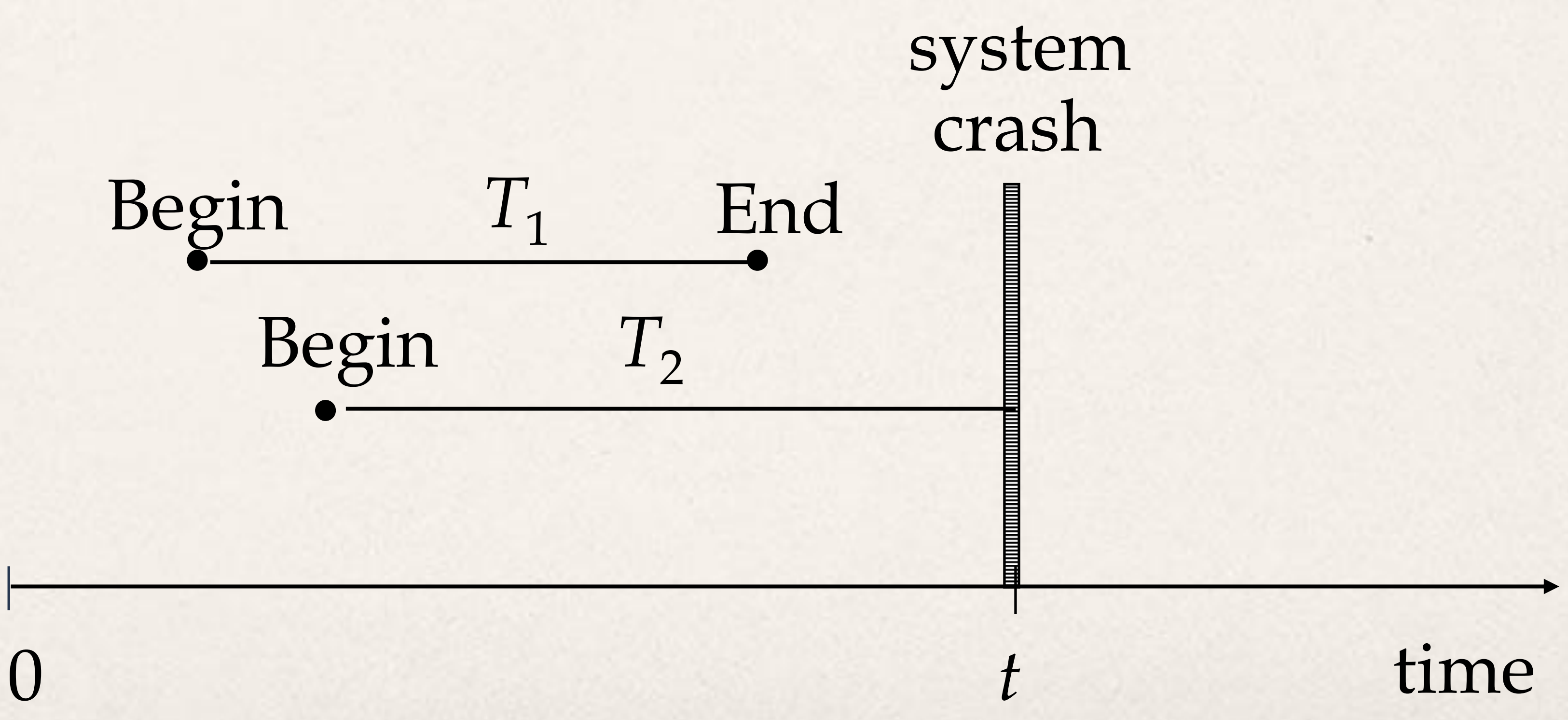
Every action of a transaction must not only perform the action but must also write a *log* record to an append-only file.



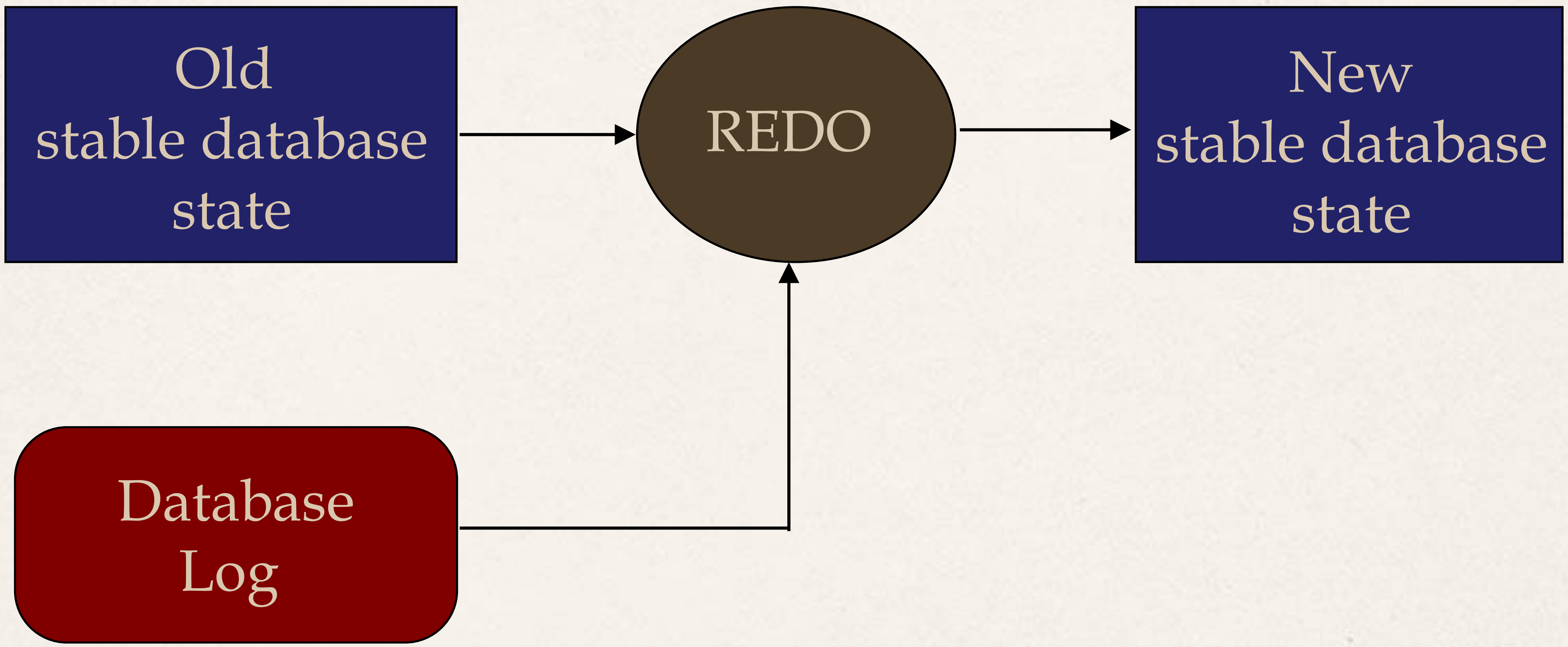
Why Logging?

Upon recovery:

- all of T_1's effects should be reflected in the database (REDO if necessary due to a failure)
- none of T_2's effects should be reflected in the database (UNDO if necessary)

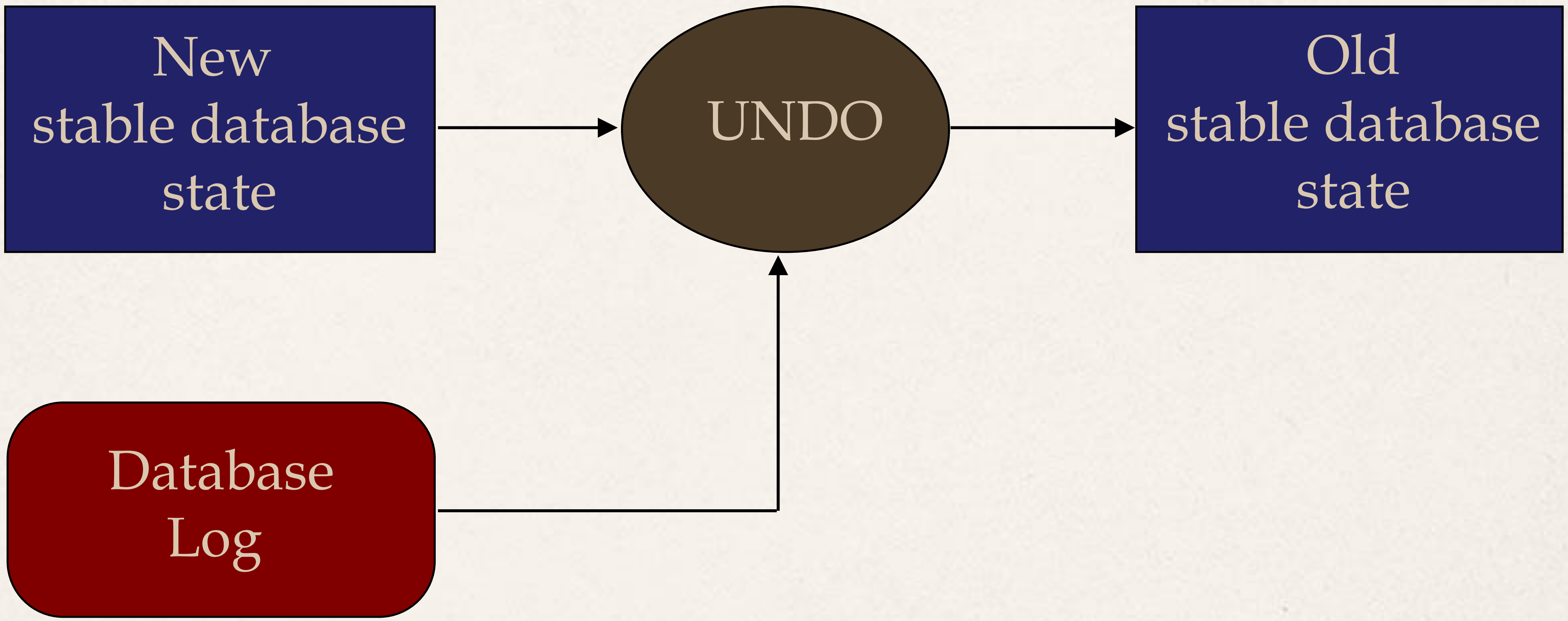


REDO Protocol



- REDO'ing an action means performing it again to generate the new value of the object
- The REDO operation uses the log information and performs the action that might have been done before, or not done due to failures.

UNDO Protocol



- UNDO'ing an action means to restore the object to its *before image*.
- The UNDO operation uses the log information and restores the old value of the object.

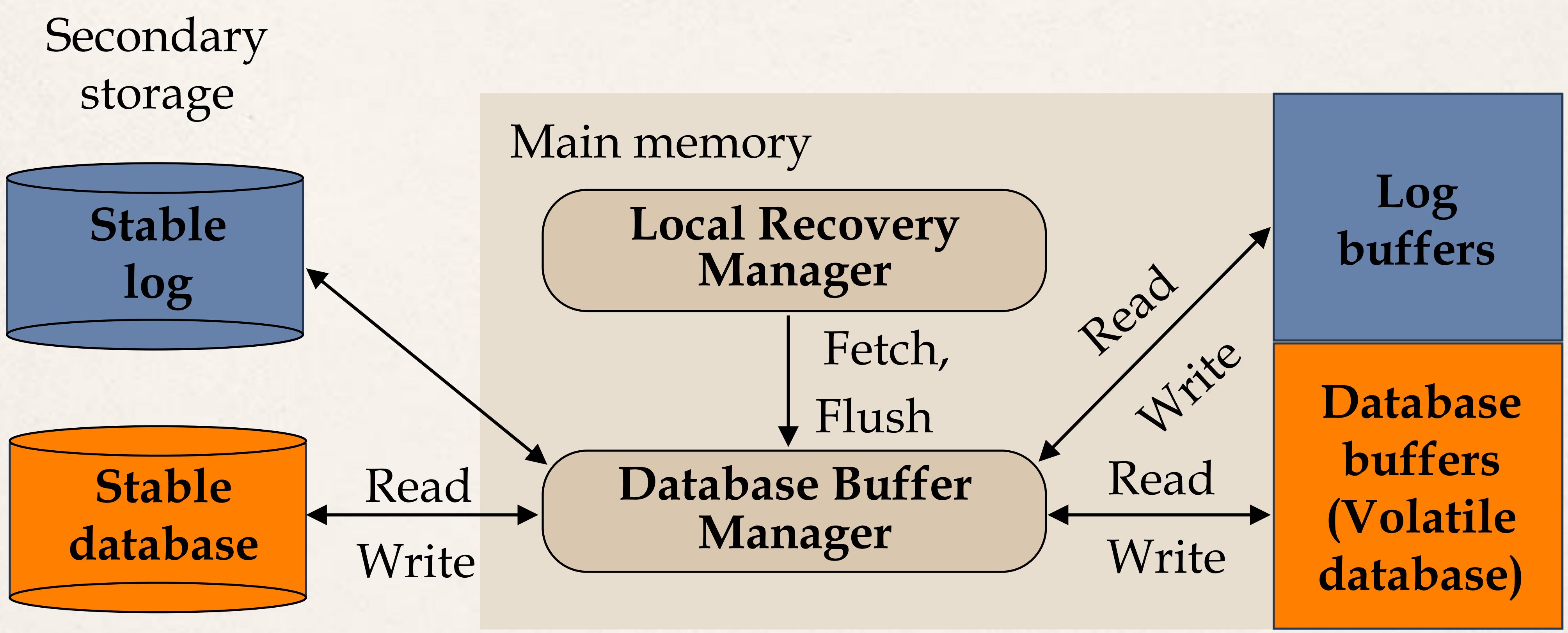
Logging

The log contains information used by the recovery process to restore the consistency of a system. This information may include

- transaction identifier
- type of operation (action)
- items accessed by the transaction to perform the action
- old value (state) of item (before image)
- new value (state) of item (after image)

…

Logging Interface



When to Write Log Records Into Stable Store

Assume a transaction T updates a page P

- Fortunate case
  - System writes P in stable database
  - System updates stable log for this update
  - SYSTEM FAILURE OCCURS!... (before T commits)

  We can recover (undo) by restoring P to its old state by using the log
- Unfortunate case
  - System writes P in stable database
  - SYSTEM FAILURE OCCURS!... (before stable log is updated)

  We cannot recover from this failure because there is no log record to restore the old value.
- Solution: Write-Ahead Logging (WAL) protocol

Write–Ahead Logging Protocol

- WAL protocol :
  1. Before a stable database is updated, (perhaps due to actions of a yet uncommitted transaction), the before images should be stored in the stable log.
     - This facilitates undo
  2. When a transaction commits, the after images have to be stored in the stable log prior to the updating of the stable database.
     - This facilitates redo

Out-of-Place Update Recovery Information

- Shadowing
  - When an update occurs, don't change the old page, but create a shadow page with the new values and write it into the stable database.
  - Update the access paths so that subsequent accesses are to the new shadow page.
  - The old page retained for recovery.
- Differential files
  - For each file F maintain
    - a read only part FR
    - a differential file consisting of insertions part DF^+ and deletions part DF^-
    - Thus, $F = (FR \cup DF^+) - DF^-$
  - Updates treated as delete old value, insert new value

Distributed Reliability Protocols

- Commit protocols
  - How to execute commit command for distributed transactions.
  - Issue: how to ensure atomicity and durability?
- 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.

Two-Phase Commit (2PC)

- **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

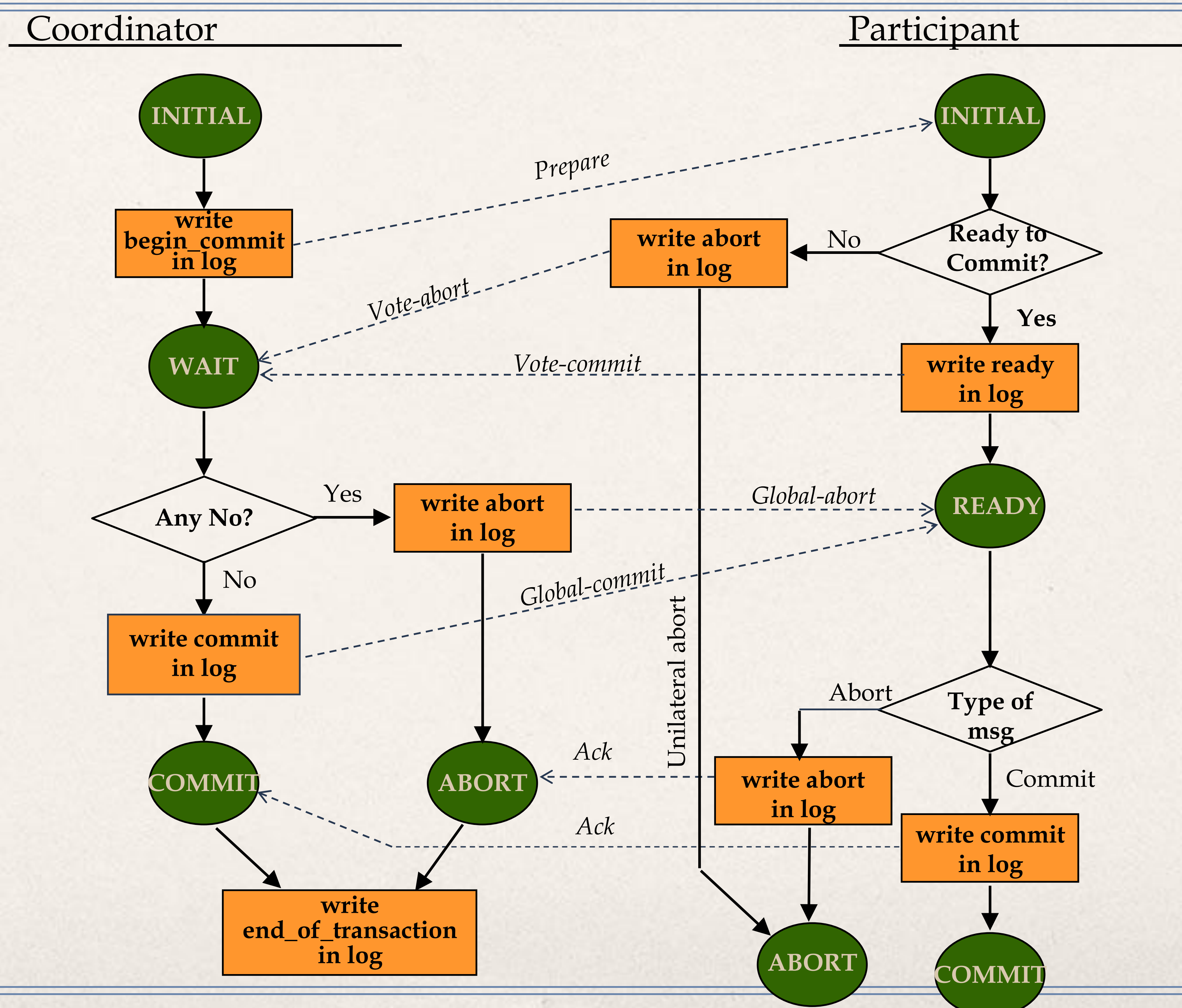
Phase 1 : The coordinator gets the participants ready to write the results into the database

Phase 2 : Everybody writes the results into the database

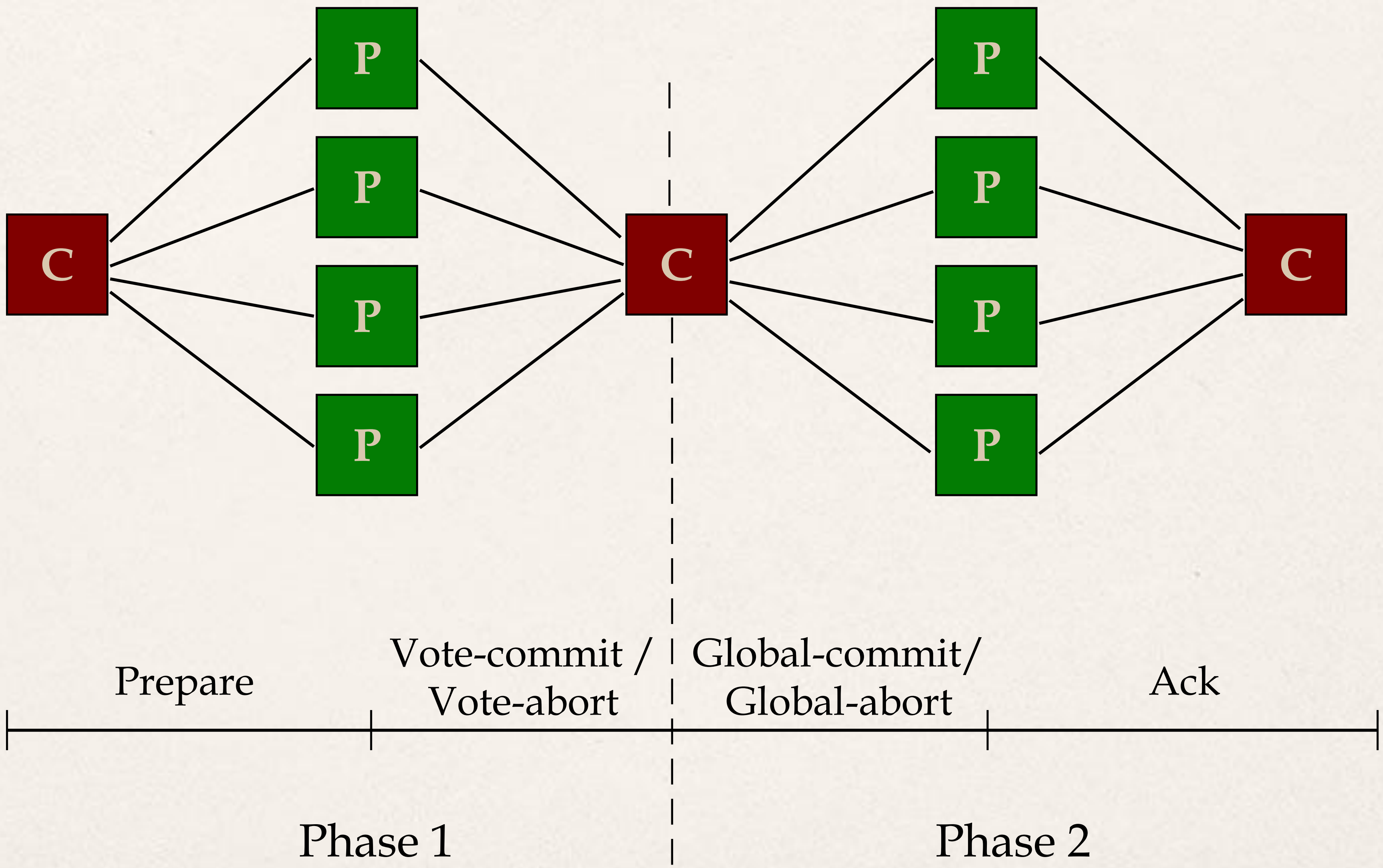
Global Commit Rule:

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

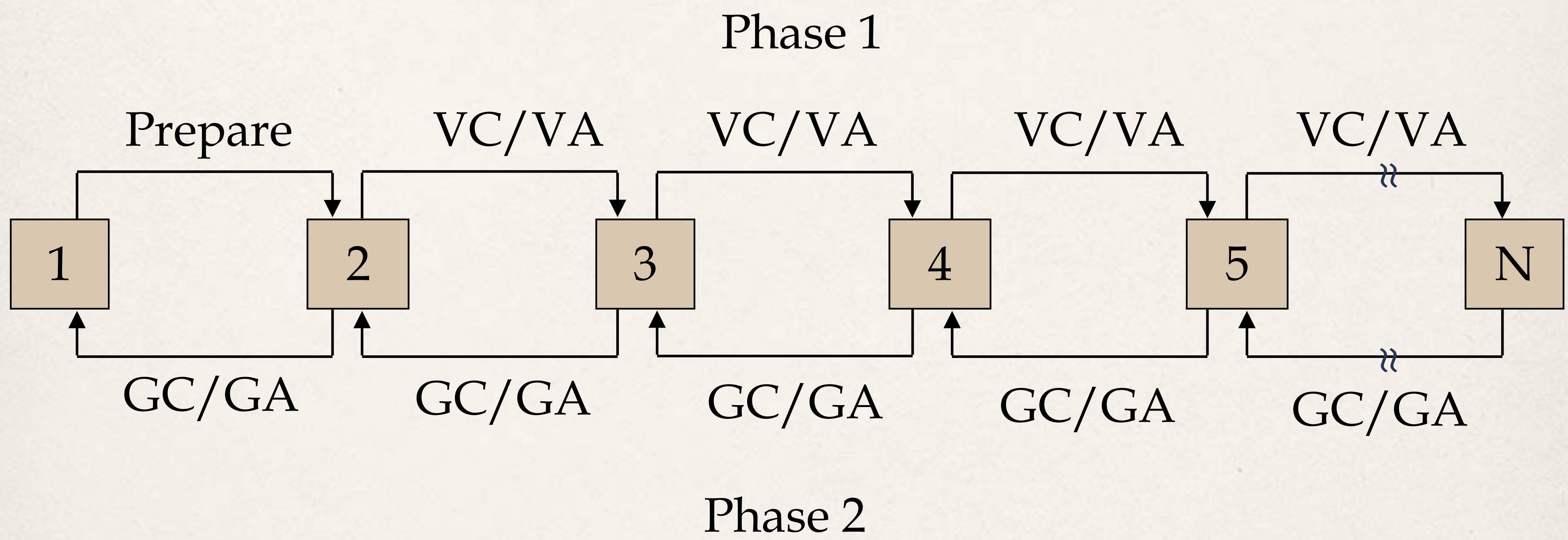
2PC Protocol Actions



Centralized 2PC

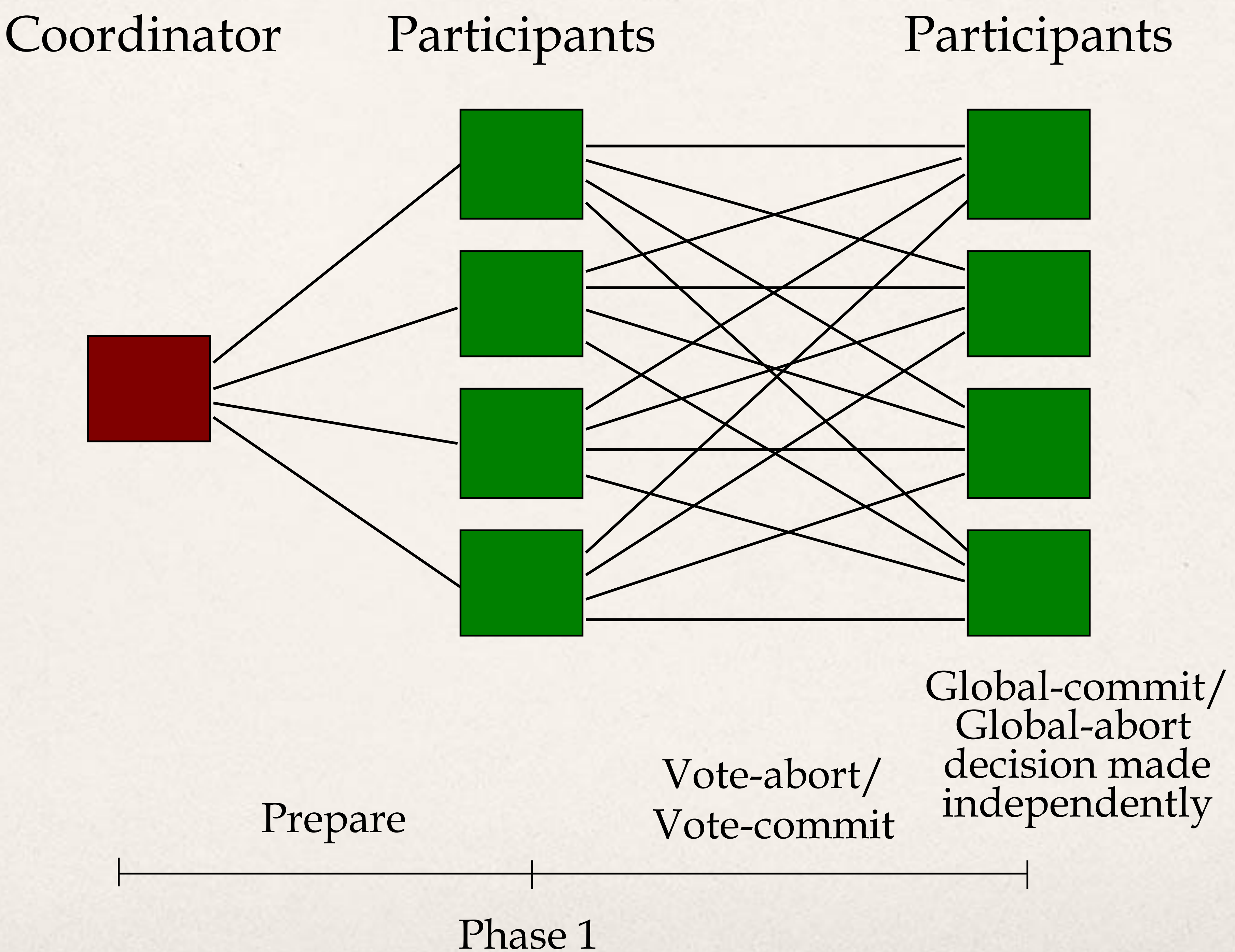


Linear 2PC - alternative

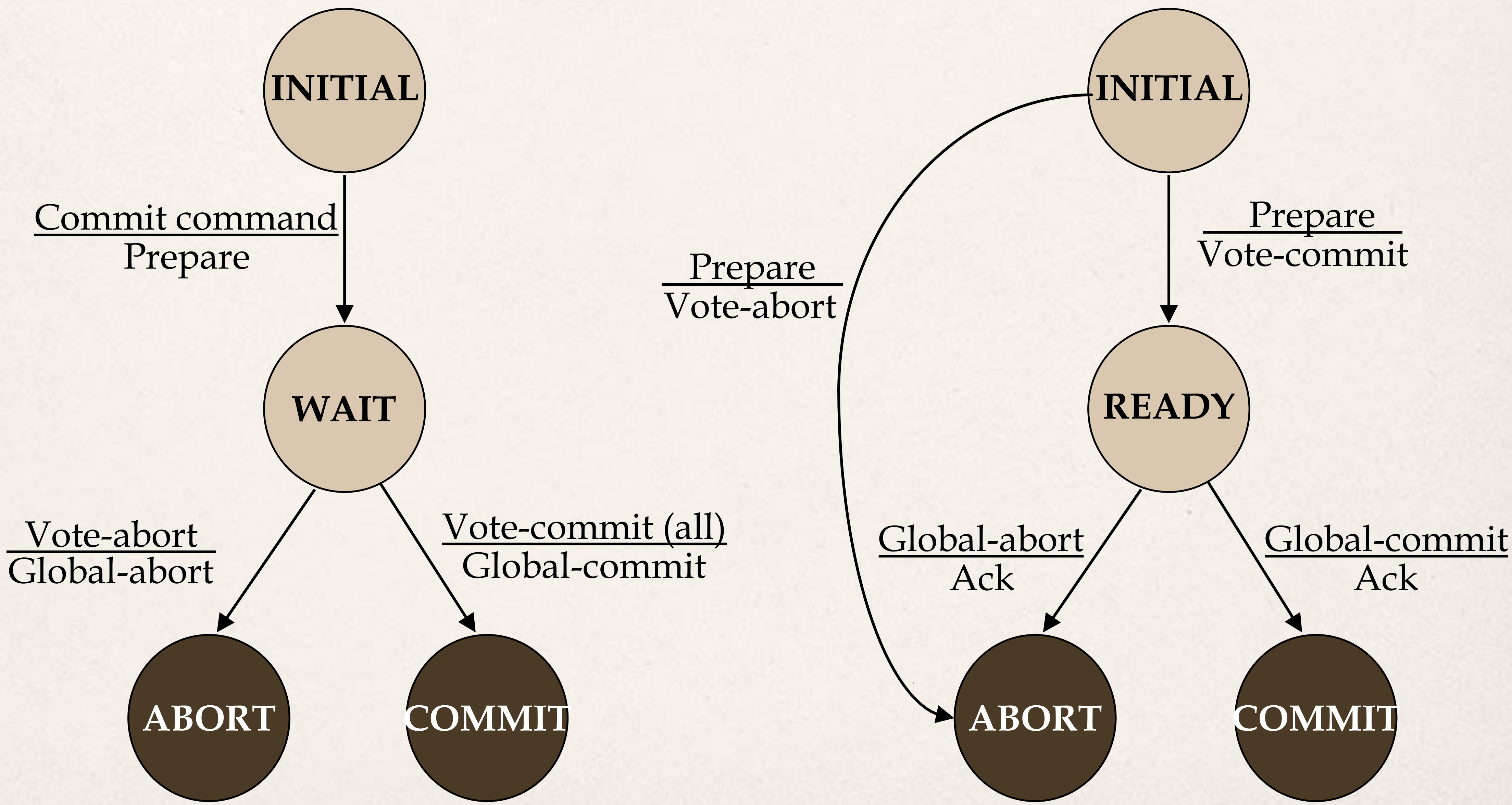


VC: Vote-Commit, VA: Vote-Abort, GC: Global-commit, GA: Global-abort

Distributed 2PC - alternative



State Transitions in 2PC

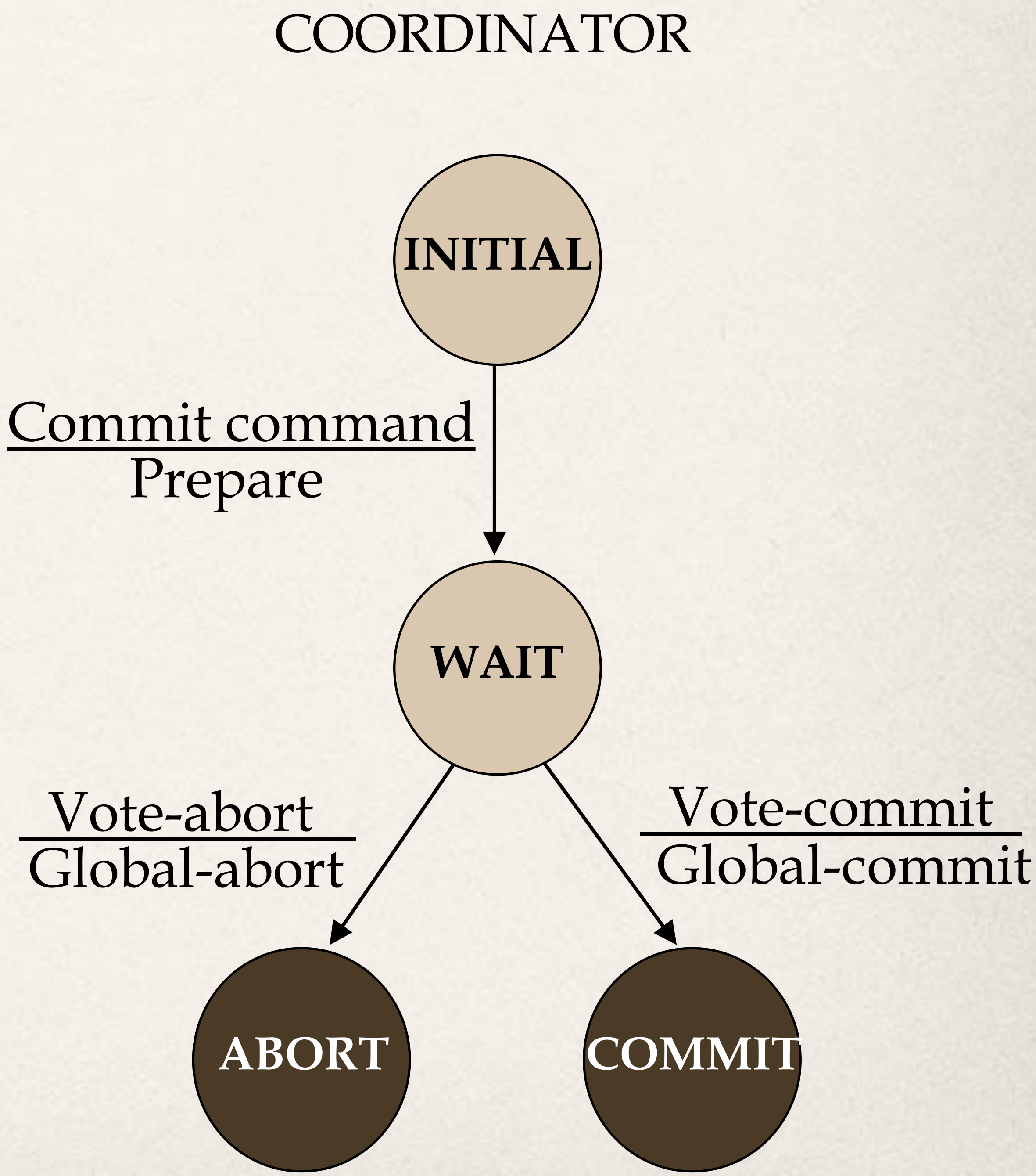


Coordinator

Participants

Site Failures - 2PC Termination

- Timeout in WAIT
  - Cannot unilaterally commit
  - Can unilaterally abort
- Timeout in ABORT or COMMIT
  - Stay blocked and wait for the Acks



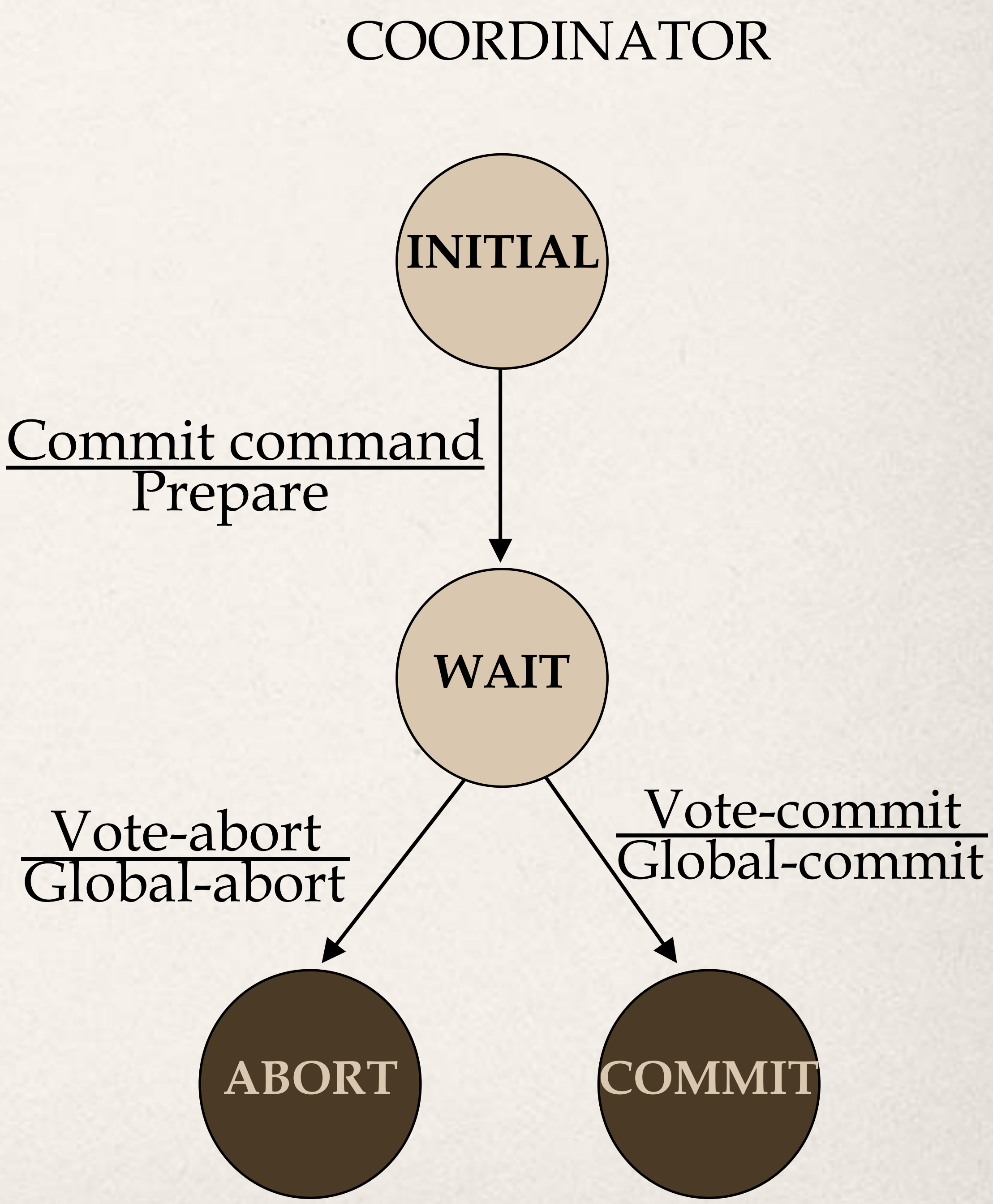
Site Failures - 2PC Termination

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

PARTICIPANTS

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

Problem With 2PC

- **Blocking**
  - Ready implies that the participant waits for the coordinator
  - If coordinator fails, site is blocked until recovery
  - Blocking reduces availability

- So, we search for nonblocking protocols – 3PC

Nonblocking Commit Protocols

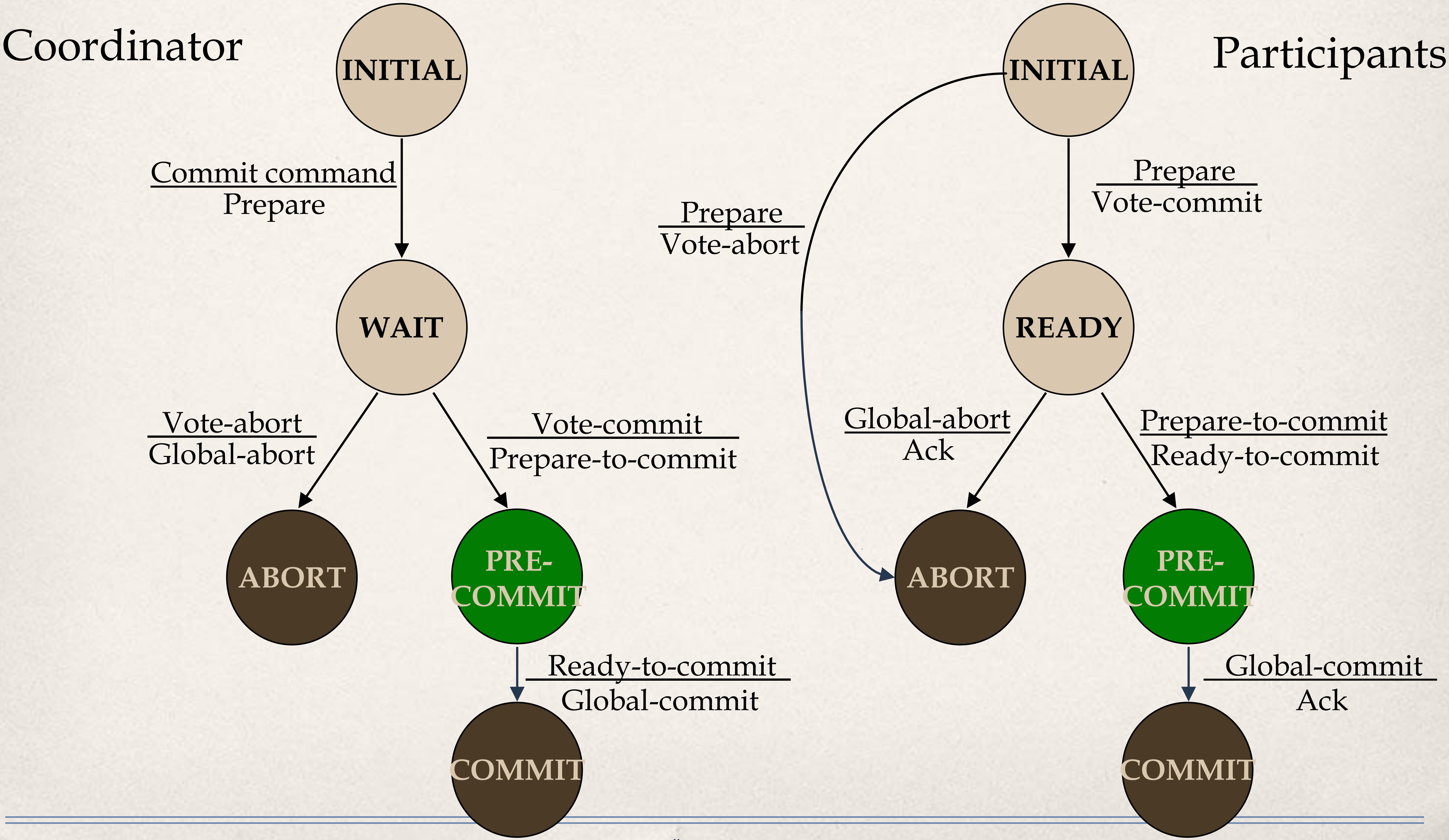
Skeen, D. (1981)

Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 133–142.

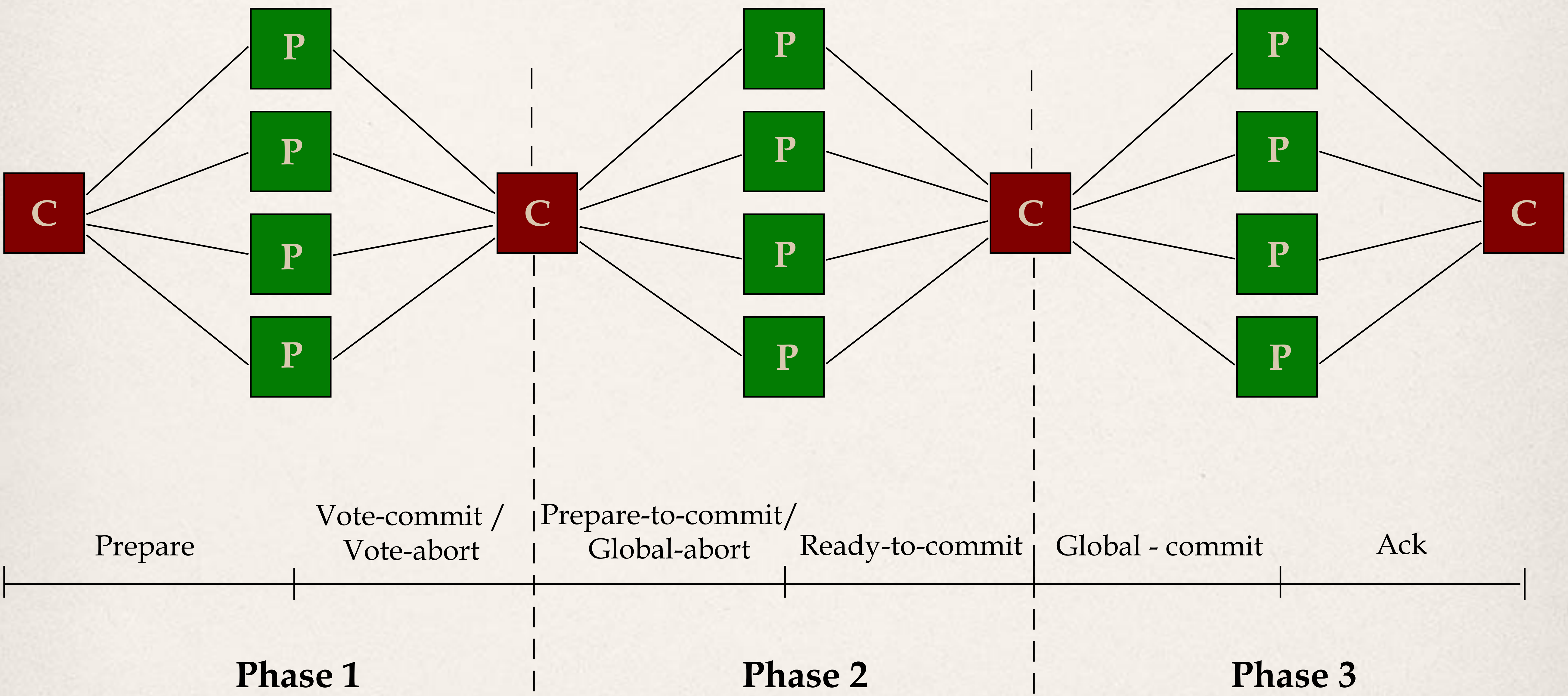
Three-Phase Commit

- 3PC is non-blocking.
- A commit protocols is non-blocking iff
  - it is synchronous within one state transition, and
  - its state transition diagram does not contain
    - a state that is "adjacent" to both a commit and an abort state, and
    - a non-committable state that 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.: 2PC COMMIT state

State Transitions in 3PC

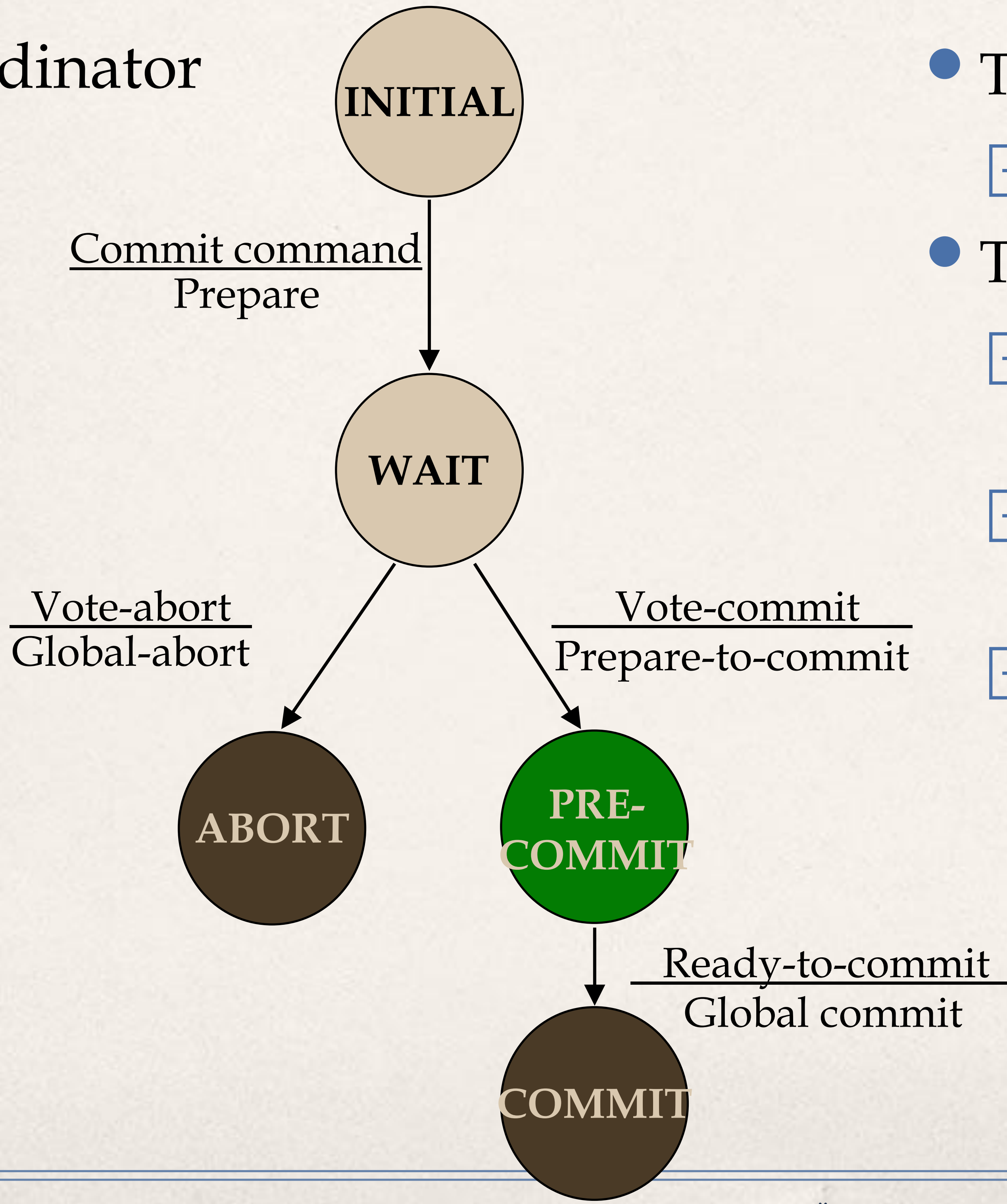


Communication Structure



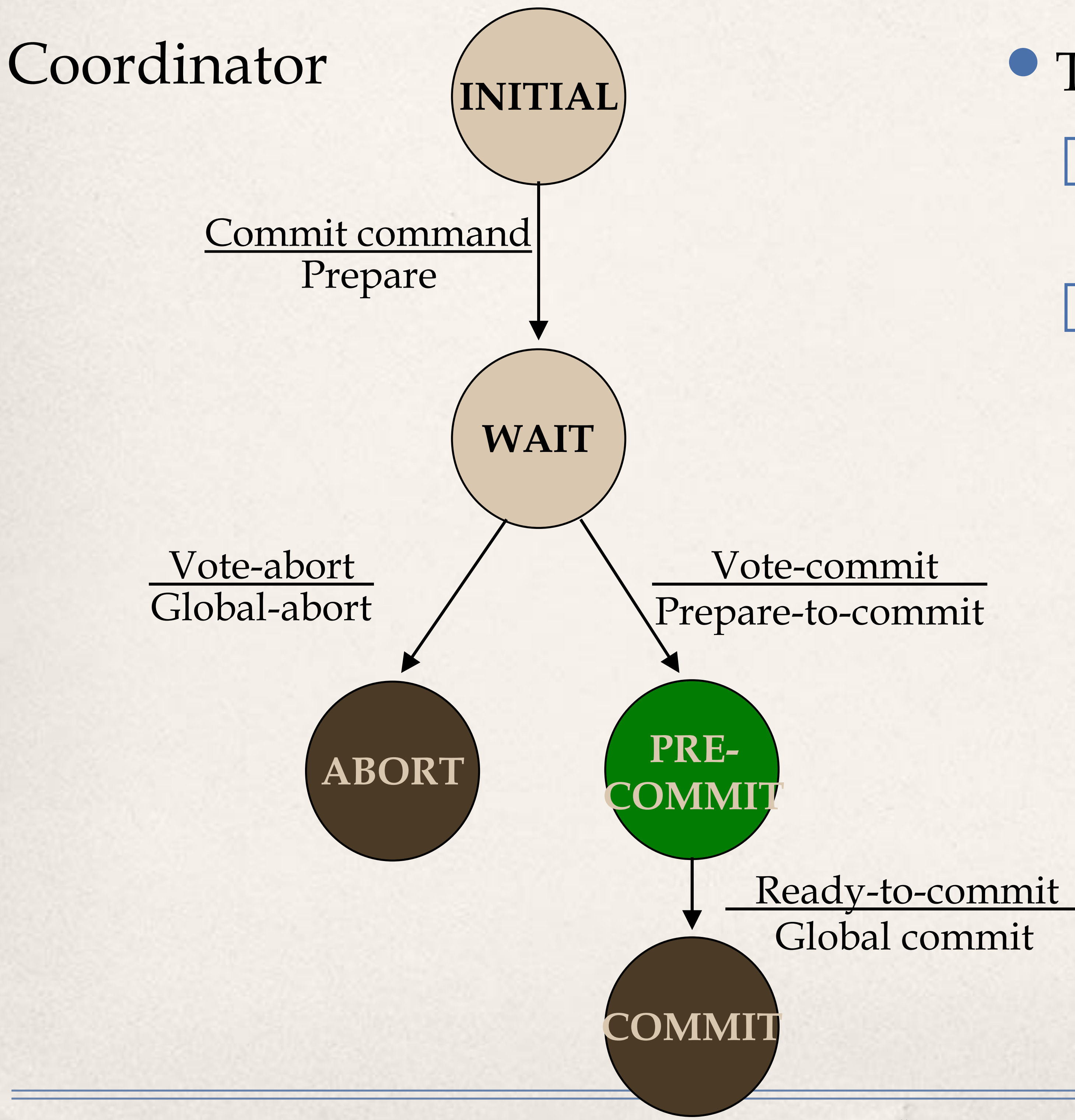
Site Failures – 3PC Termination

Coordinator



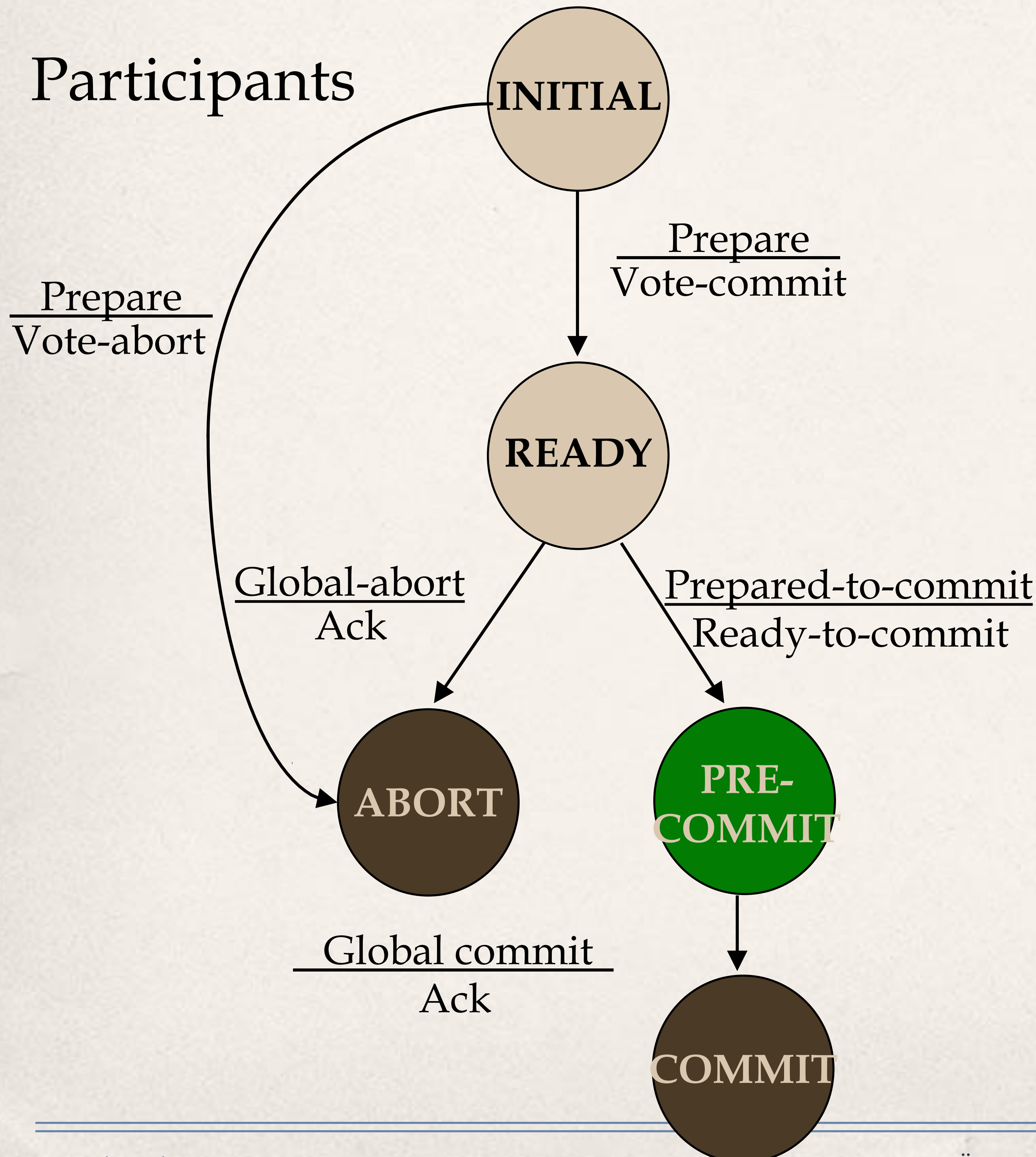
- Timeout in WAIT
  - Unilaterally abort
- Timeout in PRECOMMIT
  - Participants may not be in PRE-COMMIT, but at least in READY
  - Move all the participants to PRECOMMIT state
  - Terminate by globally committing

Site Failures – 3PC Termination



- 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



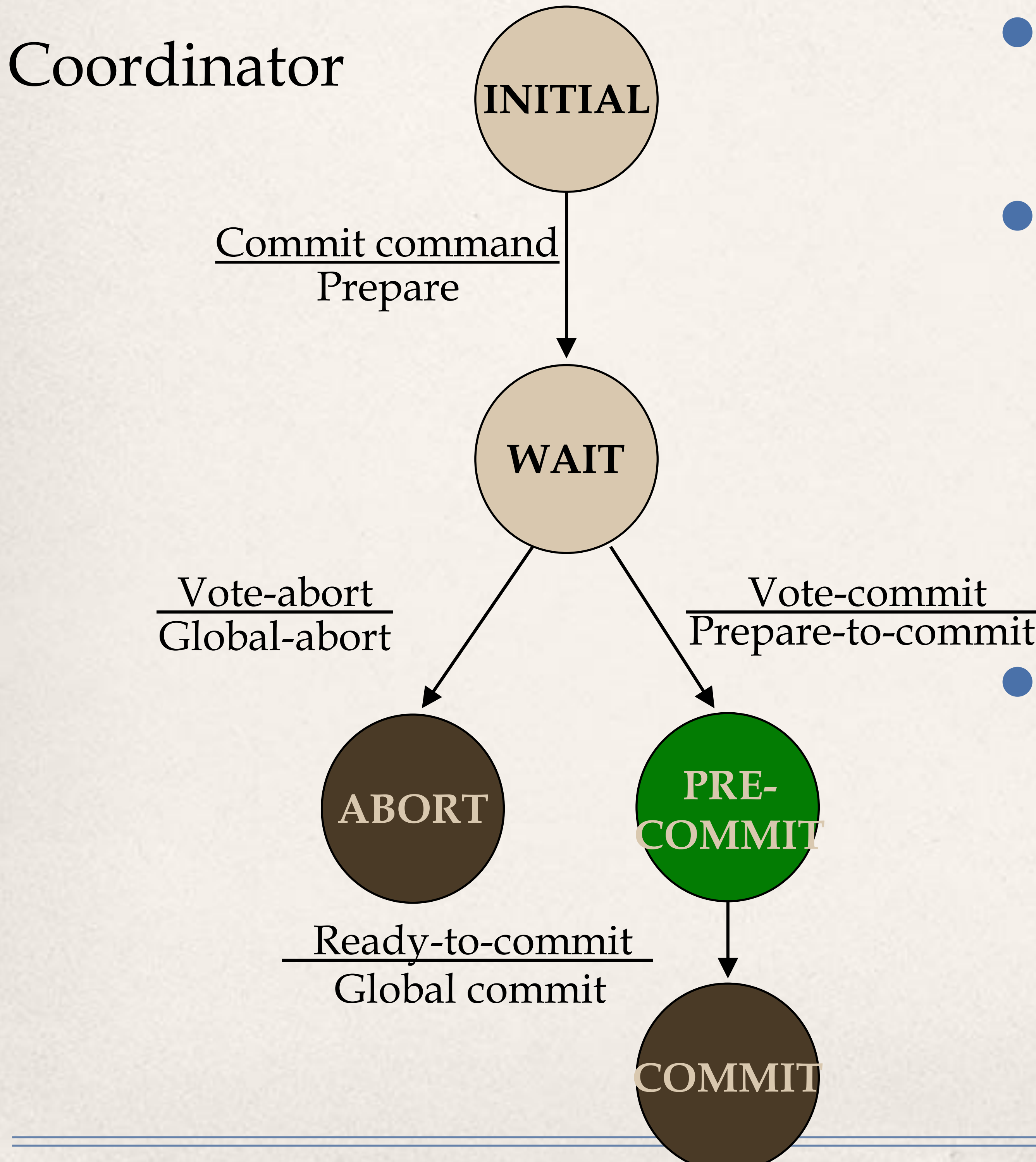
- Timeout in INITIAL
  - → Coordinator must have failed in INITIAL state
  - → Unilaterally abort
- Timeout in READY
  - → Voted to commit, but does not know the coordinator's decision
  - → Elect a new coordinator and terminate using a special protocol
- Timeout in PRECOMMIT
  - → Handle it the same as timeout in READY state

Termination Protocol Upon Coordinator Election

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

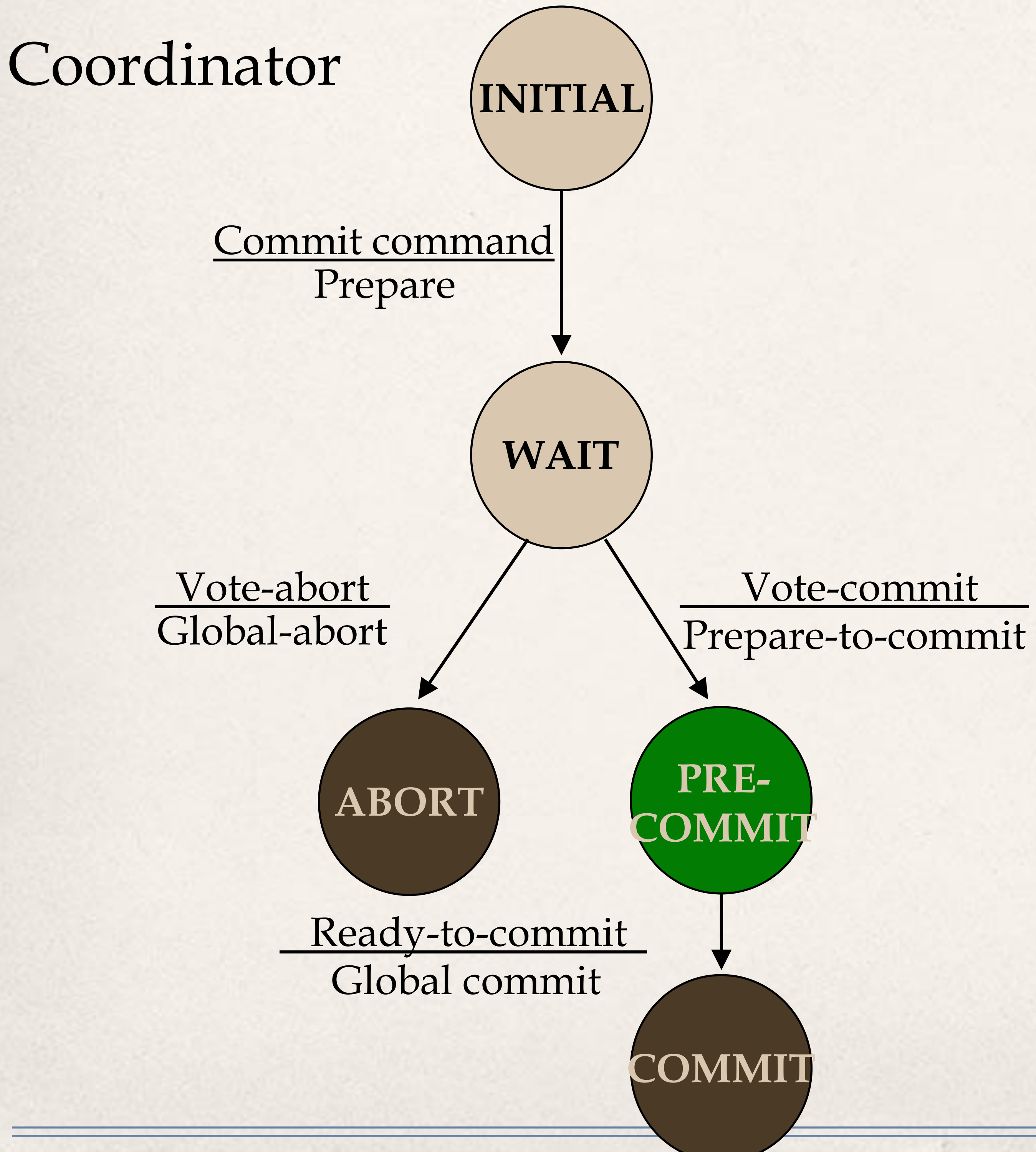
1. Coordinator sends its state to all of the participants asking them to assume its state (if they are not ahead).
2. Participants make their state transitions and reply with appropriate messages.
3. Coordinator then guides the participants towards termination:
   - If the new coordinator is in the WAIT 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



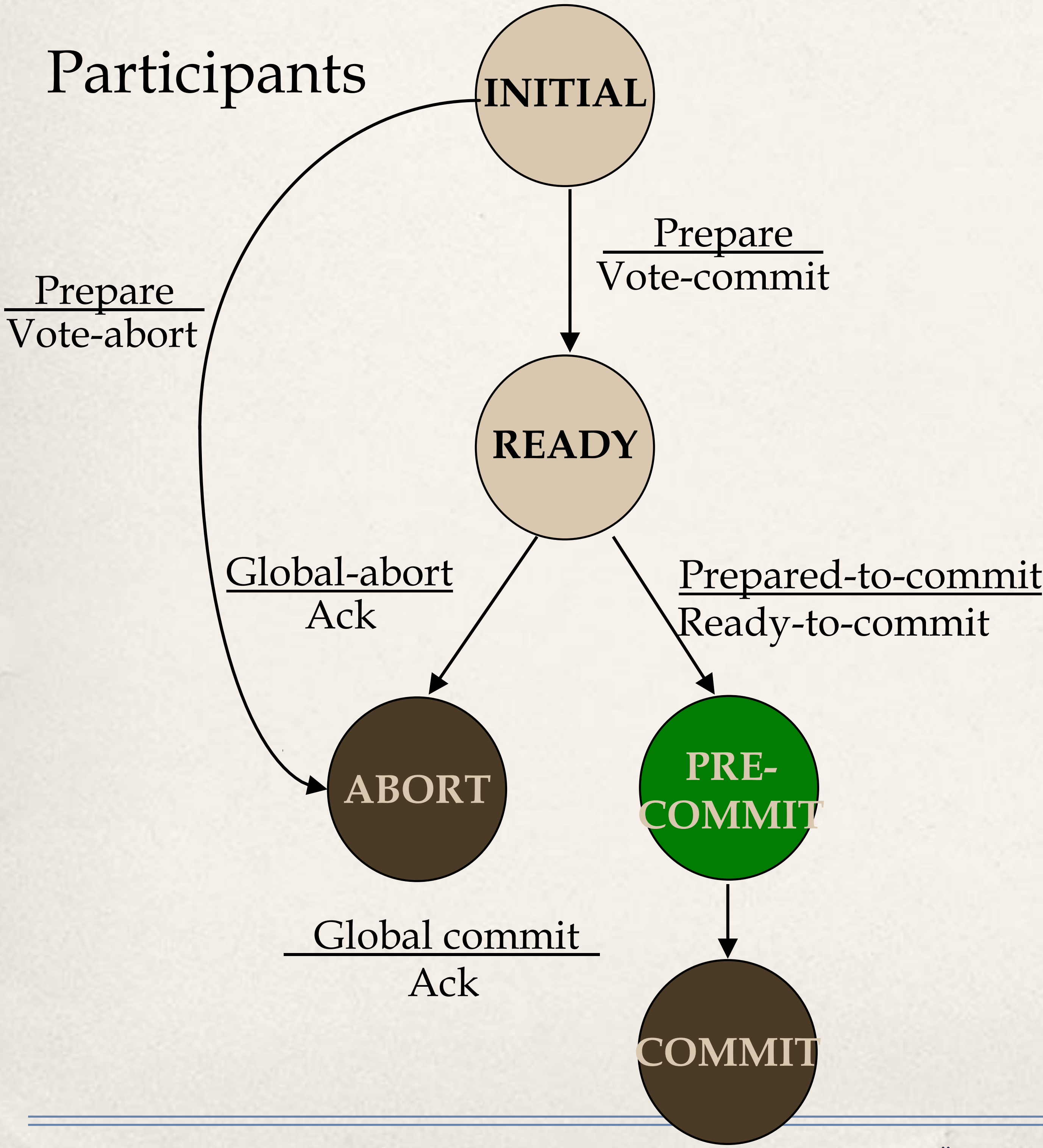
- Failure in INITIAL
  - Start commit process upon recovery
- Failure in WAIT
  - Participants have elected a new coordinator and terminated the transaction
  - Ask around for the fate of the transaction
- Failure in PRECOMMIT
  - Handle it the same as failure in WAIT state

Site Failures – 3PC Recovery



- Failure in COMMIT or ABORT
  - Nothing special if all the Acks 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
  - Ask around upon recovery
- 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

Network Partitioning

- Simple partitioning
  - Only two partitions
- Multiple partitioning
  - More than two partitions

- Formal bounds:
  - There exist non-blocking protocols which are resilient to simple partitions.
  - There exists no non-blocking protocol which is resilient to multiple partitions.

Consensus on Transaction Commit

Gray, J., & Lamport, L. (2006)

ACM Transactions on Database Systems,

Vol.31, No. 1, pp. 133 – 160.

Paxos Commit Algorithm

- It runs a Paxos consensus algorithm on the commit/abort decision of each participant to obtain a transaction commit protocol that uses 2F + 1 coordinators and makes progress if at least F+1 of them are working properly.

- "Not-synchronous" commit algorithm
- Fault-tolerant (unlike 2PC)
- Safety is guaranteed (unlike 3PC)

- Formally specified and checked
- Can be optimized to the theoretically best performance

Spanner, TrueTime & the CAP theorem

Brewer, E. (2017)

https://research.google.com/pubs/pub45855.html

Google Spanner

- As with most ACID databases, Spanner uses two-phase commit (2PC) and strict two-phase locking (2PL) to ensure isolation and strong consistency.

- 2PC has been called the "anti-availability" protocol because all members must be up for it to work.

- Spanner mitigates this by having each member be a Paxos group, thus ensuring each 2PC "member" is highly available even if some of its Paxos participants are down.

Google Spanner

- It is Google's wide-area network, plus many years of operational improvements, that greatly limit partitions in practice, and thus enable high availability.

- Google runs its own private global network.
- Each data center typically has at least three independent fibers connecting it to the private global network, thus ensuring path diversity for every pair of data centers.
- Similarly, there is redundancy of equipment and paths within a datacenter.

Google Spanner

- One subtle thing about Spanner is that it gets serializability from locks, but it gets external consistency from TrueTime.
- Spanner's external consistency invariant is that for any two transactions, T1 and T2 (even if on opposite sides of the globe): if T2 starts to commit after T1 finishes committing, then the timestamp for T2 is greater than the timestamp for T1 .
- Spanner's use of TrueTime as the clock ensures the invariant holds. In particular, during a commit, the leader may have to wait until it is sure the commit time is in the past (based on the error bounds).

Distributed Transaction Management

Distributed Reliability

Contents

- Reliability Concepts and Measures
  - Reliability and Availability
  - Mean Time Between Failures/Mean Time To Repair

Fundamental Definitions

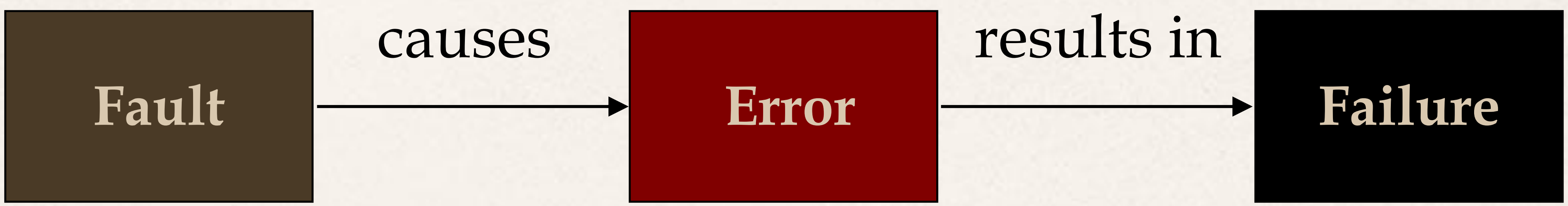
- Reliability
  - Probability that the system does not experience any failures in a given time interval.
  - Typically used to describe systems that cannot be repaired or where the operation of the system is so critical that no downtime for repair can be tolerated.

- Availability
  - Probability that the system is operational according to its specification at a given point in time *t*.
  - Refers to systems that can be repaired

Fundamental Definitions

- Failure
  - The observable deviation of a system from the behavior that is described in its specification.
- Erroneous state
  - An internal state that may not obey its specification, further transitions from this state would eventually cause a failure.
- Error
  - The part of the state which is incorrect.
- Fault
  - Any deficiency in the internal states of the components of a system or in the design of a system.

Faults to Failures

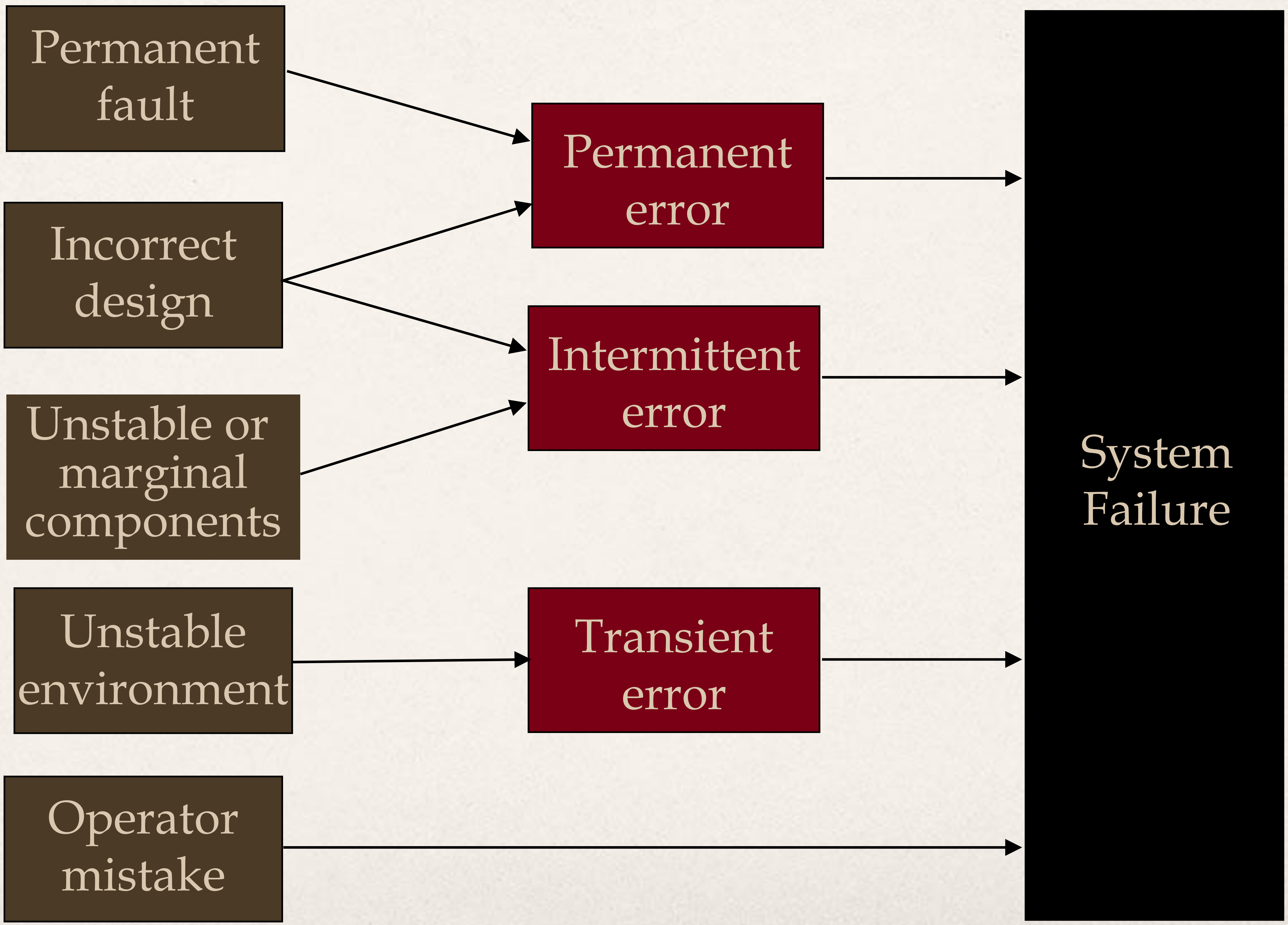


Chain of events leading to system failure

Types of Faults

- Hard faults
  - Permanent
  - Recovery from them requires intervention to “repair”
- Soft faults
  - Transient or intermittent

Sources of System Failure



Fault-Tolerance Measures

- Two measures have become popular to model the behavior of systems:
  - Mean time between failures (MTBF) - expected time between subsequent failures in a system with repair.
  - Mean time to repair (MTTR) - expected time to repair a failed system.

Availability

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

Contents

- Local Reliability Protocols
  - Architectural Considerations
  - Recovery Information
  - Execution of LRM Commands
  - Checkpointing
  - Handling Media Failures

Execution of LRM Commands

Commands to consider:

begin_transaction
read
write
commit
abort
recover

Independent of execution strategy for LRM

Execution Strategies

- Dependent upon
  - Does the buffer manager have to wait for LRM to instruct it to write the buffer pages being accessed by a transaction into stable storage or can it decide by itself to do it during the execution of that transaction?
    - fix/no-fix decision
  - Does the LRM force the buffer manager to write certain buffer pages into stable database at the end of a transaction's execution?
    - flush/no-flush decision
- Possible execution strategies:
  - no-fix/no-flush
  - no-fix/flush
  - fix/no-flush
  - fix/flush

No-Fix/No-Flush

- Abort
  - Buffer manager may have written some of the updated pages into stable database
  - LRM performs partial undo (or transaction undo)
- Commit
  - LRM writes an “end_of_transaction” record into the log
- Recover
  - For those transactions that only have a “begin_transaction” in the log, a global undo is executed by LRM
  - For those transactions that have both a “begin_transaction” and an “end_of_transaction” record in the log, a partial redo is initiated by LRM

No-Fix/Flush

- Abort
  - Buffer manager may have written some of the updated pages into stable database
  - LRM performs partial undo
- Commit
  - LRM issues a `flush` command to the buffer manager for all updated pages
  - LRM writes an "end_of_transaction" record into the log
- Recover
  - Perform global undo
  - No need to perform redo

Fix/No-Flush

- Abort
  - None of the updated pages have been written into stable database
  - Release the `fixed` pages (`unfix` command)
- Commit
  - LRM writes an “end_of_transaction” record into the log.
  - LRM sends an `unfix` command to the buffer manager for all pages that were previously `fixed` in the database buffers.
- Recover
  - No need to perform global undo
  - Perform partial redo

Fix/Flush

- Abort
  - None of the updated pages have been written into stable database
  - Release the `fix`ed pages
- Commit (the following have to be done atomically)
  - LRM sends an `unfix` command to the buffer manager for all pages that were previously `fixed`
  - LRM issues a `flush` command to the buffer manager for all updated pages
  - LRM writes an "end_of_transaction" record into the log.
- Recover
  - No need to do anything

Checkpointing

- Simplifies the task of determining actions of transactions that need to be undone or redone when a failure occurs.
- A checkpoint record contains a list of active transactions.
- Steps:
  1. Write a begin_checkpoint record into the log
  2. Collect the checkpoint data into the stable storage
  3. Write an end_checkpoint record into the log

Handling Media Failures