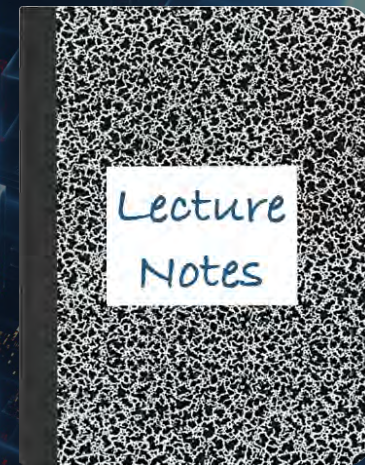


CS 417 – DISTRIBUTED SYSTEMS

# Week 8: Distributed Transactions

## Part 1: Two-Phase Commit



Paul Krzyzanowski

© 2021 Paul Krzyzanowski. No part of this content, may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

# Atomic Transactions

## Transaction

- An operation composed of a sequence of discrete steps.
- All the steps must be completed for the transaction to be **committed**. The results are made permanent.
- Otherwise, the transaction is **aborted** and the state of the system **reverts** to what it was before the transaction started.
  - **rollback** = reverting to a previous state

# Example: buying a house

- Make an offer
- Sign contract
- Deposit money in escrow
- Inspect the house
- Critical problems from inspection?
- Get a mortgage
- Have seller make repairs

**Commit:** sign closing papers & transfer deed

**Abort:** return escrow and revert to pre-purchase state

*All or nothing property*

# Another Example

Book a flight from Allentown, PA to Inyokern, CA

No non-stop flights are available:

*Transaction begin*

1. Reserve a seat for Allentown to O'Hare (ABE→ORD)
2. Reserve a seat for O'Hare to Los Angeles (ORD→LAX)
3. Reserve a seat for Los Angeles to Inyokern (LAX→IYK)

*Transaction end*

If there are no seats available on the LAX→IYK leg of the journey, the entire transaction is *aborted* and reservations for (1) and (2) are undone.

# Basic Operations

## Transaction primitives:

- **Begin transaction (*prepare*)**: mark the start of a transaction

*Then read/write/compute data* – modify files, objects, program state

- But data will have to be restored if the transaction is aborted
- **End transaction**: mark the end of a transaction – no more tasks
- **Commit transaction**: make the results permanent
- **Abort transaction**: kill the transaction, restore old values

# Properties of transactions: ACID

- **Atomic**
  - The transaction happens as a single **indivisible** action. Everything succeeds or else the entire transaction is rolled back. Others do not see intermediate results.
- **Consistent**
  - A transaction cannot leave the database in an inconsistent state & all invariants must be preserved. E.g., total amount of money in all accounts must be the same before and after a *transfer funds* transaction.
- **Isolated (Serializable)**
  - Transactions cannot interfere with each other or see intermediate results  
If transactions run at the same time, the result must be the same as if they executed in some serial order.
- **Durable**
  - Once a transaction commits, the results are made permanent.

# Distributed Transaction

**Transaction that updates data on two or more systems**

Implemented as a set of sub-transactions

## Challenge

Handle machine, software, & network failures while preserving transaction integrity

# Distributed Transactions

Each computer runs a **transaction manager**

- Responsible for sub-transactions on that system
- Performs *prepare*, *commit*, and *abort* calls for sub-transactions

Every sub-transaction must agree to commit changes before the overall transaction can complete



# Commits Among Sub-transactions = Consensus

- Remember consensus?
  - Agree on a value proposed by at least one process
- BUT – here we need unanimous agreement to commit
- The coordinator proposes to commit a transaction
  - All participants agree  $\Rightarrow$  all participants then commit
  - Not all participants agree  $\Rightarrow$  all participants then abort

# Core Properties

The algorithm must have these properties:

## 1. **Safety** (the algorithm must work correctly)

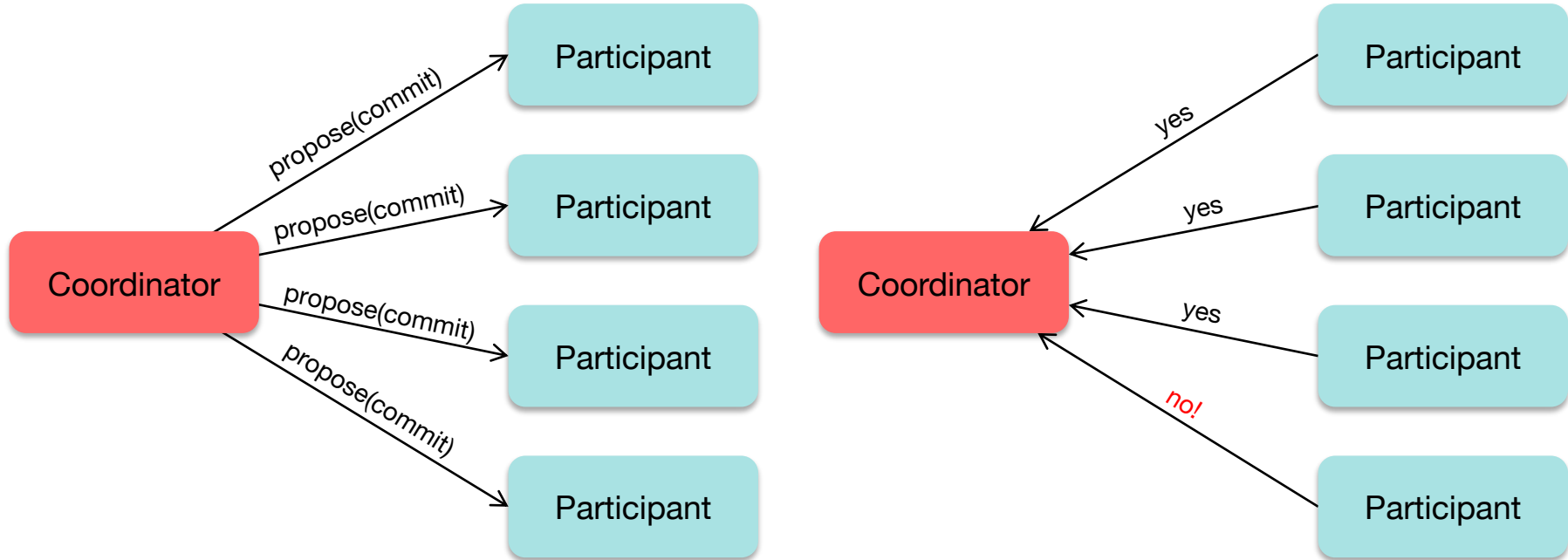
- If one sub-transaction commits, no other sub-transaction will abort
- If one sub-transaction needs to abort, no sub-transactions will commit

## 2. **Liveness** (the algorithm must make progress & reach its goal)

- If no sub-transactions fail, the transaction will commit
- If any sub-transactions fail, the algorithm will reach a conclusion to abort

# Two-Phase Commit Protocol

# Two-Phase Commit Protocol



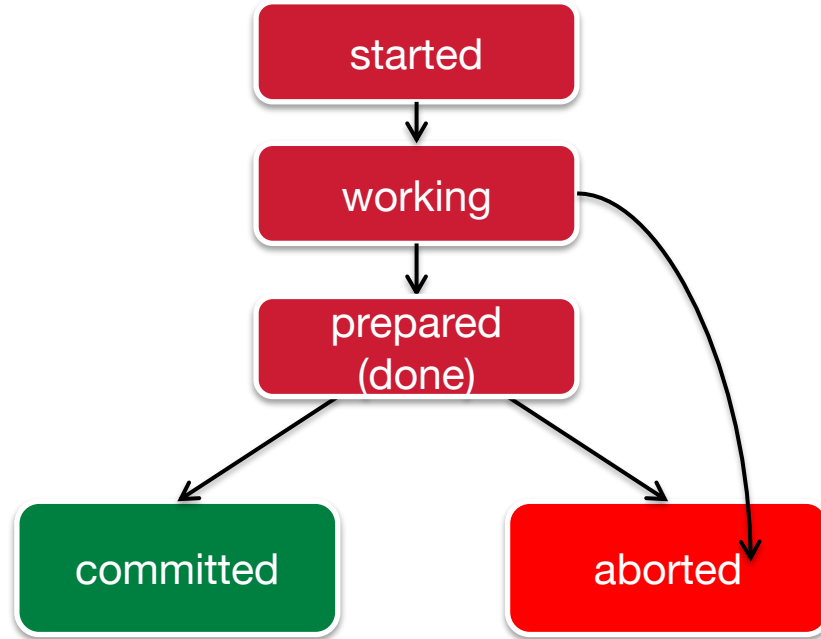
# Two-phase commit protocol

## Goal:

*Reliably agree to **commit** or **abort** a collection of sub-transactions*

- All processes in the transaction will agree to commit or abort
- Consensus: all processes agree on whether or not to commit
- One transaction manager is *elected* as a **coordinator**
  - the rest are **participants**
- Assume:
  - **write-ahead log** in **stable storage**
    - Enables recording transaction state if system restarts and saving old values of data in case they need to be restored
  - No system dies forever
  - Systems can always communicate with each other

# Transaction States

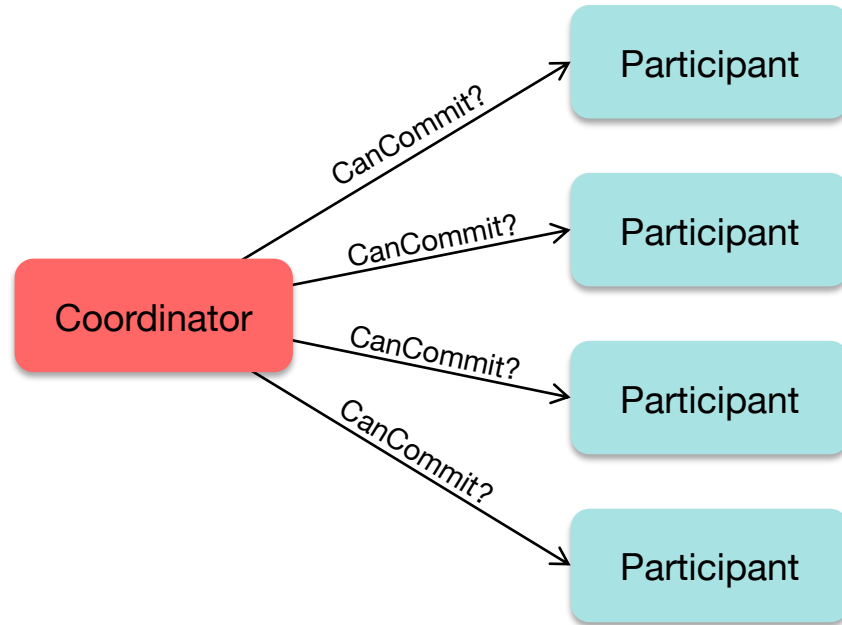


When a participant enters the *prepared* state, it contacts the coordinator to start the commit protocol to commit the entire transaction

# Two-Phase Commit Protocol

## Phase 1: Voting Phase

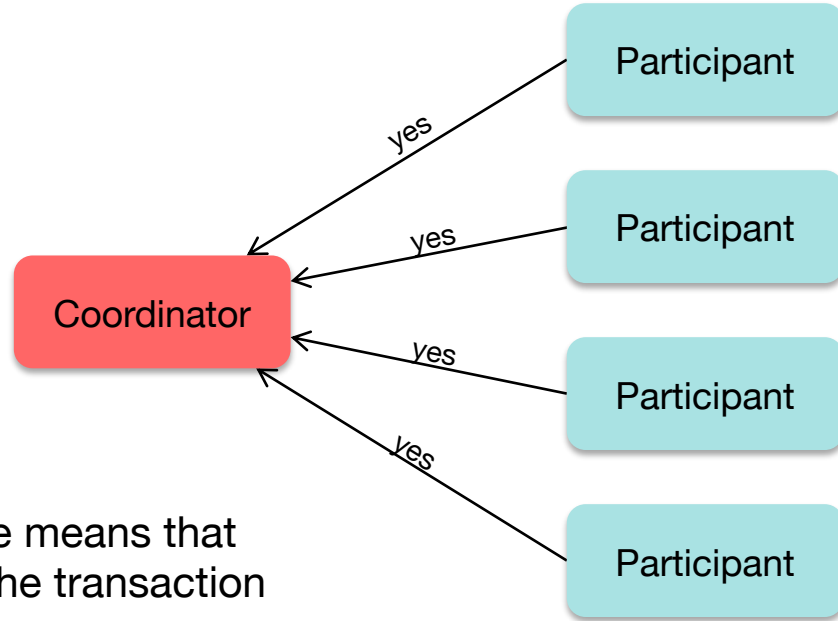
Get commit agreement from every participant



# Two-Phase Commit Protocol

## Phase 1: Voting Phase

Get commit agreement from every participant



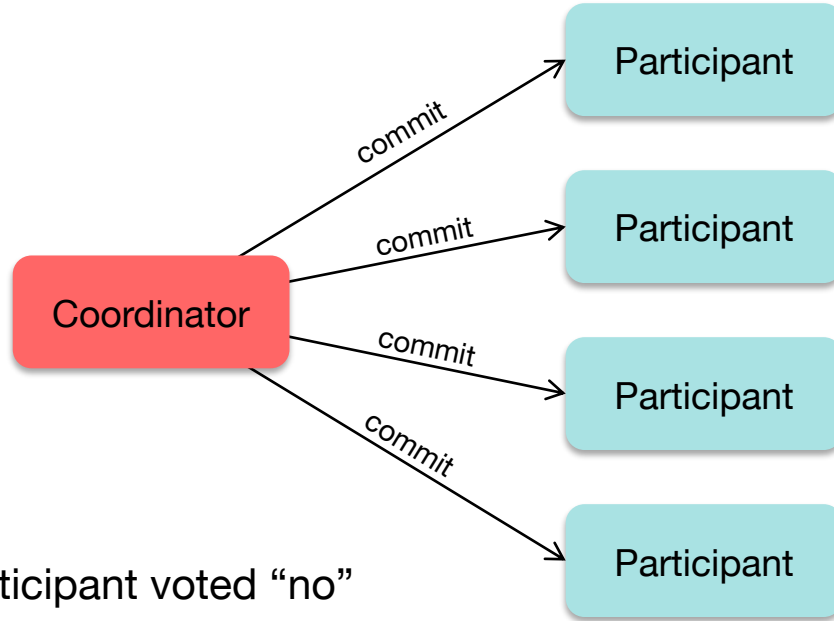
A single “no” response means that we will have to abort the transaction



# Two-Phase Commit Protocol

## Phase 2: Commit Phase

Send the results of the vote to every participant

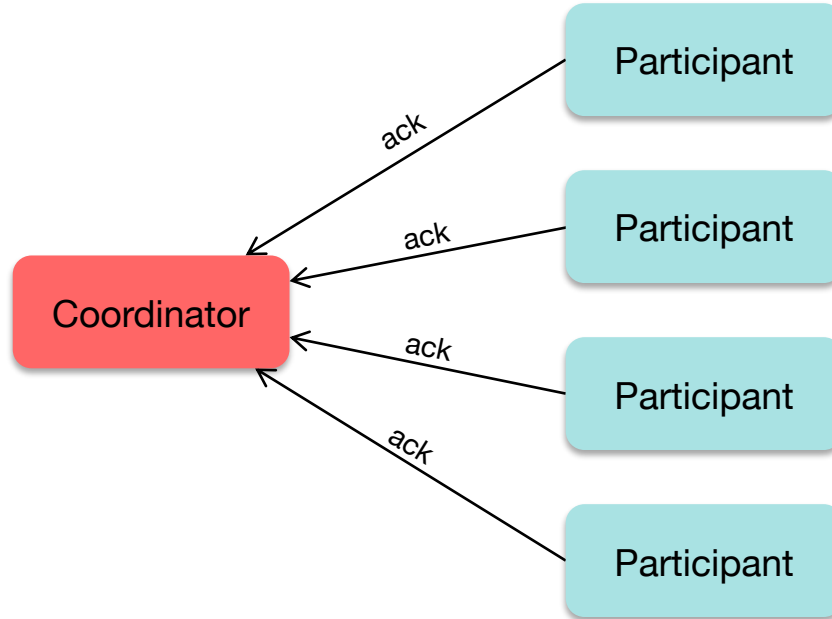


Send *abort* if any participant voted "no"

# Two-Phase Commit Protocol

## Phase 2: Commit Phase

Get “I have committed” acknowledgements from *every* participant



# Dealing with failure

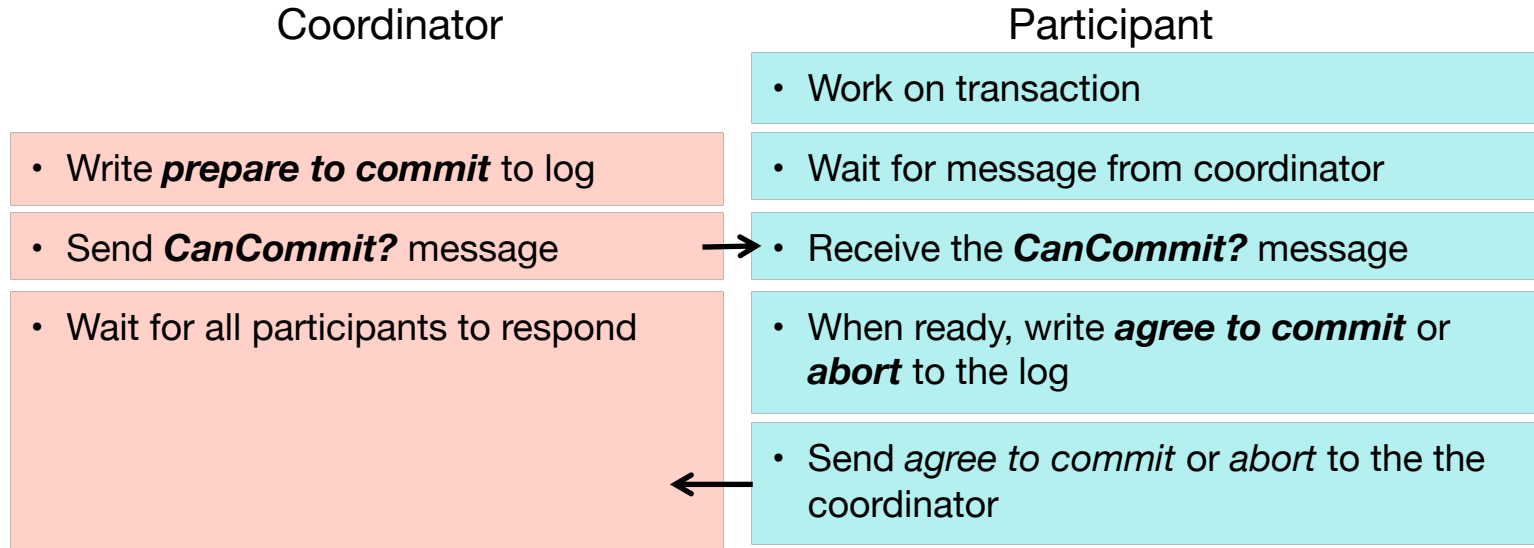
- 2PC assumes a *fail-recover model*
  - Any failed system will eventually recover
- A recovered system cannot change its mind
  - If a node agreed to commit and then crashed, it must be willing and able to commit upon recovery

We need to handle fail-restart failure & support recovery

- Each system will use a *write-ahead (transaction) log*
  - Keep track of where it is in the protocol (and what it agreed to)
  - As well as values to enable commit or abort (rollback)

# Two-Phase Commit Protocol: Phase 1

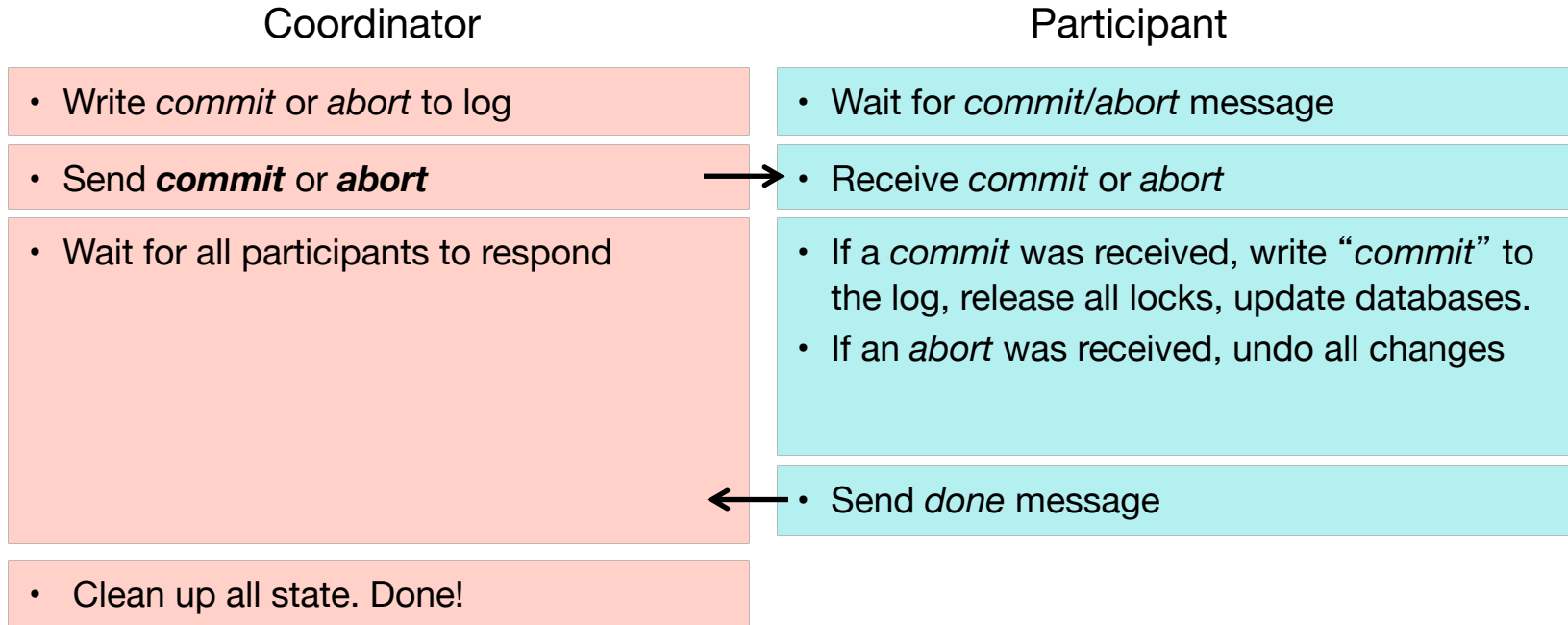
## 1. Voting Phase



**Get distributed agreement:** the coordinator asked each participant if it will commit or abort and received replies from each coordinator.

# Two-Phase Commit Protocol: Phase 2

## 2. Commit Phase



Tell *all participants* to *commit* or *abort*  
Get everyone's response that they're done.

# Consensus Properties

- **Validity property**

- Aborts in every case except when every process agrees to commit
- The final value (commit or not) has been voted on by at least one process

- **Uniform Agreement property**

- Every process agrees on the value proposed by the coordinator if and only if they are instructed to do so by the coordinator in phase 2

- **Integrity property**

- Every process proposes only a single value (commit or abort) and does not change its mind

- **Termination property**

- Every process is guaranteed to make progress and eventually return a vote to the coordinator

# Dealing with failure

## Failure during Phase 1 (voting)

### Coordinator dies

Some participants may have responded; others have no clue

⇒ Coordinator **restarts voting**: checks log; sees that voting was in progress

### Participant dies

The participant may have died before or after sending its vote to the coordinator

⇒ If the coordinator received the vote, it waits for other votes and **go to phase 2**

⇒ Otherwise: **wait for the participant** to recover and respond (keep querying it)

# Dealing with failure

## Failure during Phase 2 (commit/abort)

### Coordinator dies

Some participants may have been given commit/abort instructions

⇒ **Coordinator restarts**; checks log; **informs all participants** of chosen action

### Participant dies

The participant may have died before or after getting the commit/abort request

⇒ Coordinator **keeps trying to contact the participant** with the request

⇒ **Participant recovers**; checks log; gets request from coordinator

- If it committed/aborted, acknowledge the request
- Otherwise, process the commit/abort request and send back the acknowledgement



# Adding a recovery coordinator

- Another system can take over for the coordinator
  - It could be a participant that detected a timeout to the coordinator
- Recovery node needs to find the state of the protocol
  - Contact ALL participants to see how they voted
  - If we get **voting results from all participants**
    - We know that Phase 1 has completed
    - If all participants voted to commit ⇒ send *commit* request
    - Otherwise send *abort* request
  - If **ANY participant states that it has not voted**
    - We know that Phase 1 has *not* completed
    - ⇒ Restart the protocol
- But ... if any participant node also crashes, we're stuck!
  - Must wait for recovery

The End