Discussion 12

Distributed Transactions

Announcements

Vitamin 12 (Distributed Transactions) due Monday April 22 at 11:59pm

Project 5 (Recover) due Tuesday April 23 at 11:59pm

Project 5 party project on Friday April 19 5-7pm in Cory 521

Agenda

I. Distributed Transactions
II. Worksheet

Distributed Consensus

Distributed Computing

- Shared-nothing parallel architecture
  - No shared memory/disk - all communication over network
- Network is unreliable
  - Packets may be delayed, duplicated, reordered, dropped
- Clocks are not synchronized
  - Cannot just use clock to order messages - may be 12:03:05 on one machine and 12:01:55 on another
- Cannot even accurately check if a node is up or down
  - How do you distinguish between slow network, node that is busy (but will respond eventually), and node that is down?

Distributed Computing

- Data is partitioned over multiple nodes
  - Operations may span multiple nodes (recall parallel query processing)
- How do we commit a transaction?
  - If any node has a problem (e.g. there's a deadlock on one node's pages, a local operation results in a failure), then we cannot save all changes:
    - Remember: atomicity means either all or no changes get saved
    - We must abort, which is not saving any changes in this case
  - Otherwise, we can proceed to commit the transaction

Two-Phase Commit (2PC)

- Let's say you want to meet with some friends at 8pm tomorrow, and want *all* of them to be there
  - Contact each of them, and ask if the time (8pm tomorrow) works for them
    - If it works for a friend, tell the friend to reserve that time slot *for now*
  - If they *all* can, contact each friend again, and tell them that everyone is indeed meeting at 8pm tomorrow
  - If *one* of the friends can't, tell each friend that the meeting isn't happening
  - Keep trying to contact each friend until they all respond back
- Not related to 2PL (Two-Phase Locking)!
- Coordinator + Participants

Two-Phase Commit (2PC)

- Splits commit into two phases
  - **Phase 1 (voting):** Coordinator asks Participants to vote (YES, commit the transaction, or NO, abort the transaction), and collects the votes
    - Coordinator ➜ Participants: PREPARE
    - Participants ➜ Coordinator: VOTE YES/VOTE NO
  - **Phase 2 (results):** Coordinator tells Participants the result of the vote
    - Coordinator ➜ Participants: COMMIT/ABORT
    - Participants ➜ Coordinator: ACK
  - Requires **unanimous** agreement to commit
    - if *any* node can't commit, stuck with the "nothing" part of atomicity

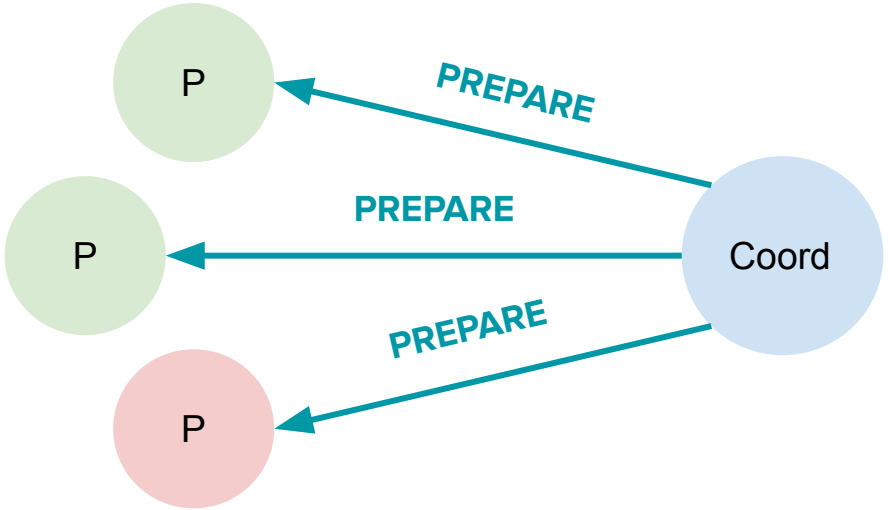
Two-Phase Commit (2PC)

- Let's say you (**the coordinator**) want to meet with some friends (**the participants**) at 8pm tomorrow, and want *all* of them to be there
  - Contact each of them, and ask if the time (8pm tomorrow) works for them (**PREPARE**)
    - If it works for a friend, tell the friend to reserve that time slot *for now* (**participants acquire locks, VOTE YES/NO**)
  - If they *all* can, contact each friend again, and tell them that everyone is indeed meeting at 8pm tomorrow (**COMMIT**)
  - If *one* of the friends can't, tell each friend that the meeting isn't happen (**ABORT**)
  - Keep trying to contact each friend until they all respond back (**ACK**)

Two-Phase Commit (2PC) [Abort Example]

Phase 1: Coordinator asks Participants to vote (YES, commit the transaction, or NO, abort the transaction), and collects the votes

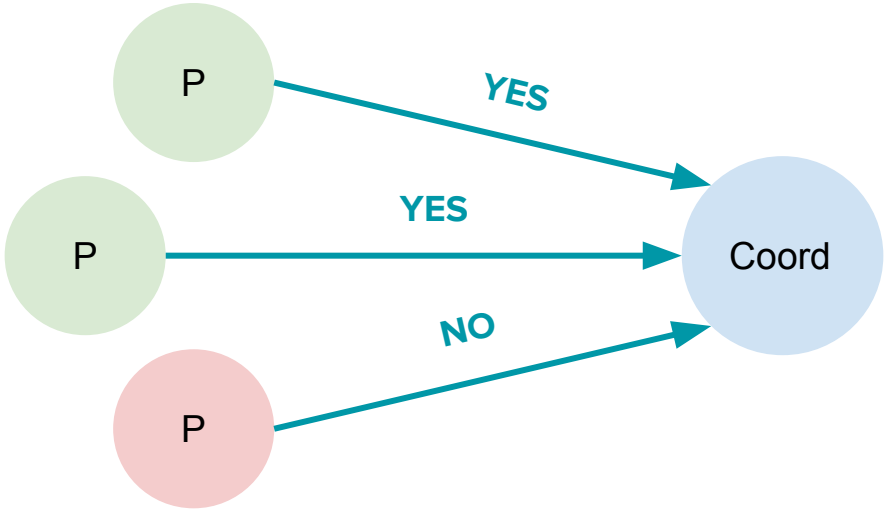
- Coordinator ➔ Participants: PREPARE
- Participants ➔ Coordinator: VOTE YES/VOTE NO



Two-Phase Commit (2PC) [Abort Example]

Phase 1: Coordinator asks Participants to vote (YES, commit the transaction, or NO, abort the transaction), and collects the votes

- Coordinator ➔ Participants: PREPARE
- Participants ➔ Coordinator: VOTE YES/VOTE NO



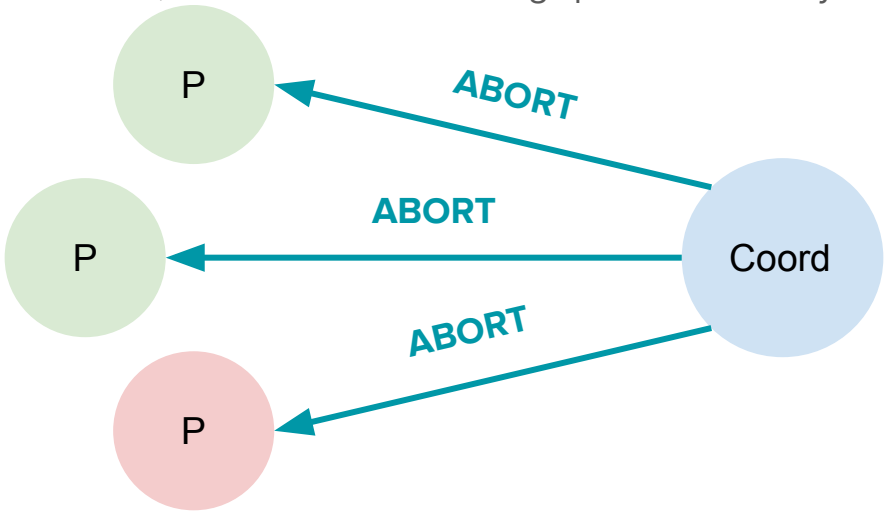
Not a unanimous agreement!

Two-Phase Commit (2PC) [Abort Example]

Phase 2: Coordinator tells Participants the result of the vote

- Coordinator ➔ Participants: COMMIT/ABORT
- Participants ➔ Coordinator: ACK

- Requires unanimous agreement to commit
  - if any node can't commit, stuck with the "nothing" part of atomicity

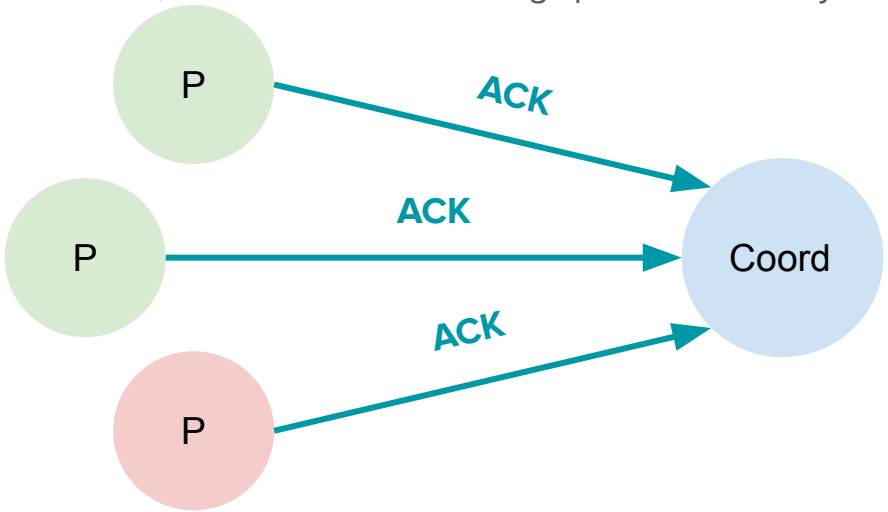


Two-Phase Commit (2PC) [Abort Example]

Phase 2: Coordinator tells Participants the result of the vote

- Coordinator ➔ Participants: COMMIT/ABORT
- Participants ➔ Coordinator: ACK

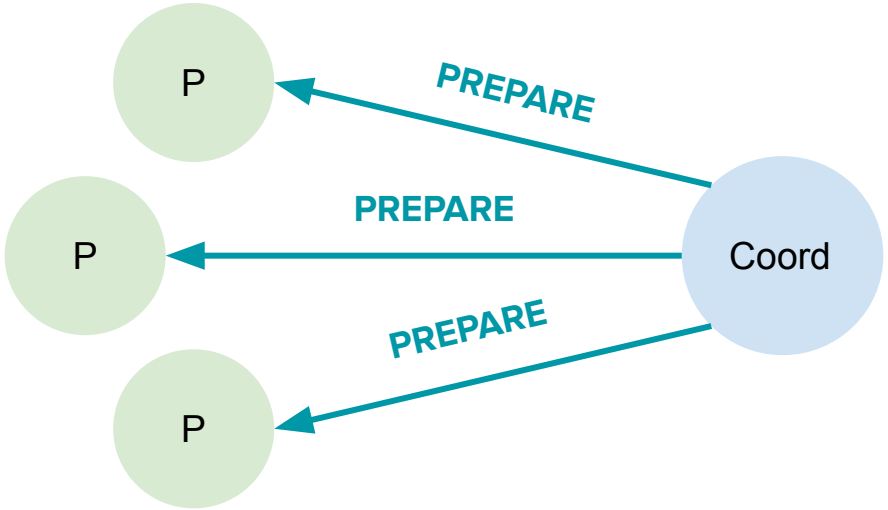
- Requires unanimous agreement to commit
  - if any node can't commit, stuck with the "nothing" part of atomicity



Two-Phase Commit (2PC) [Commit Example]

Phase 1: Coordinator asks Participants to vote (YES, commit the transaction, or NO, abort the transaction), and collects the votes

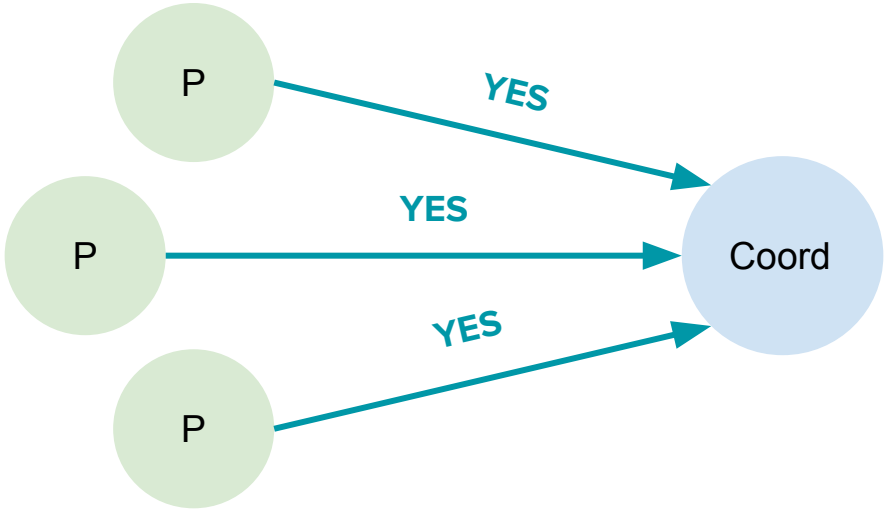
- Coordinator ➔ Participants: PREPARE
- Participants ➔ Coordinator: VOTE YES/VOTE NO



Two-Phase Commit (2PC) [Commit Example]

Phase 1: Coordinator asks Participants to vote (YES, commit the transaction, or NO, abort the transaction), and collects the votes

- Coordinator ➔ Participants: PREPARE
- Participants ➔ Coordinator: VOTE YES/VOTE NO



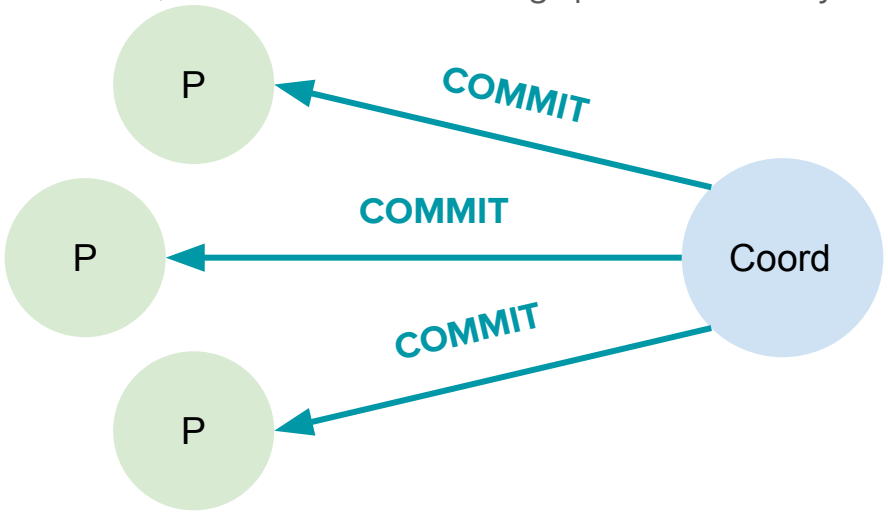
A unanimous agreement!

Two-Phase Commit (2PC) [Commit Example]

Phase 2: Coordinator tells Participants the result of the vote

- Coordinator ➔ Participants: COMMIT/ABORT
- Participants ➔ Coordinator: ACK

- Requires unanimous agreement to commit
  - if any node can't commit, stuck with the "nothing" part of atomicity

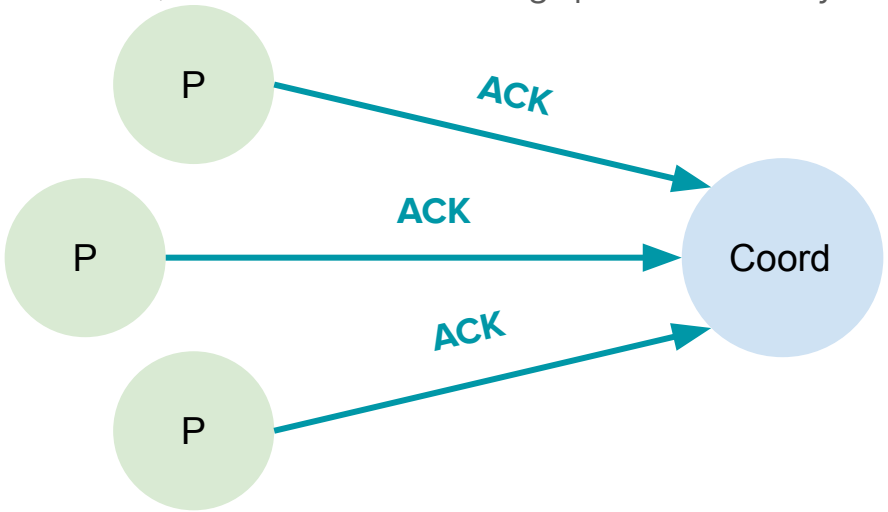


Two-Phase Commit (2PC) [Commit Example]

Phase 2: Coordinator tells Participants the result of the vote

- Coordinator ➔ Participants: COMMIT/ABORT
- Participants ➔ Coordinator: ACK

- Requires unanimous agreement to commit
  - if any node can't commit, stuck with the "nothing" part of atomicity



Two-Phase Commit (2PC): Logging

- **Phase 1** (voting)
  - **Coordinator → Participants:** PREPARE message
  - Participants: log *and flush* either a PREPARE or ABORT record to log
    - log entry keeps track of coordinator ID
    - if a participant votes NO, it knows for sure the transaction will abort, and can begin cleaning up the transaction immediately
  - **Participants → Coordinator:** VOTE YES or VOTE NO
  - Coordinator: log *and flush* either a COMMIT or ABORT record to log
    - commit and abort log entry contains all participant IDs

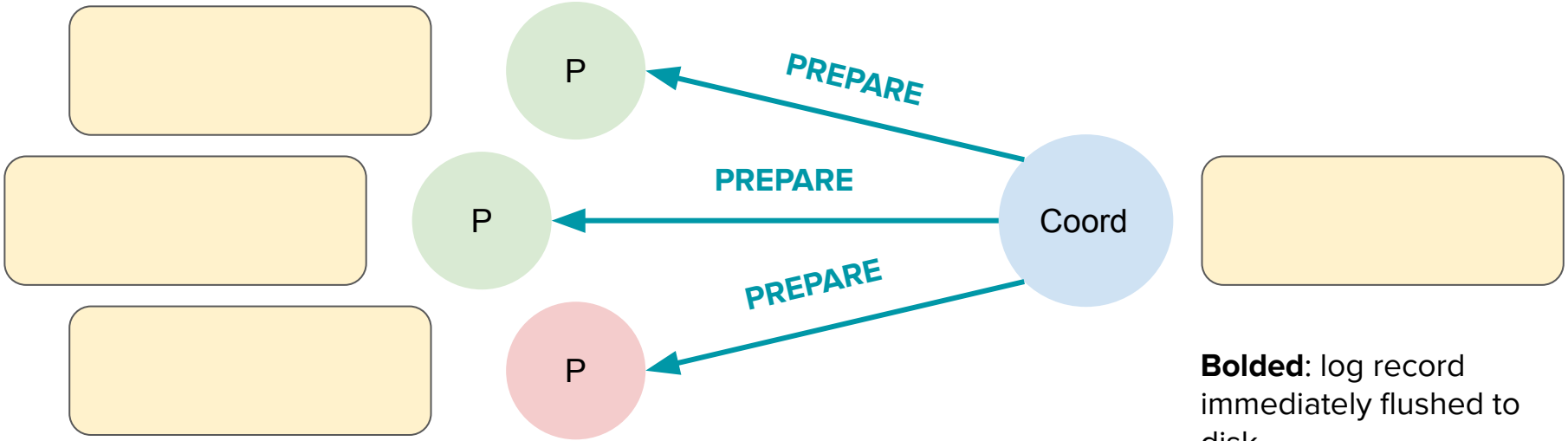
Two-Phase Commit (2PC): Logging

- **Phase 2** (results)
  - **Coordinator → Participants:** COMMIT or ABORT message
  - Participants: log *and flush* COMMIT or ABORT record to log
  - **Participants → Coordinator:** ACK message
  - Coordinator: log (no immediate flush required) END record to log and remove from transaction table

Two-Phase Commit (2PC) [Abort Example]

Phase 1:

- Coordinator ➔ Participants: PREPARE message

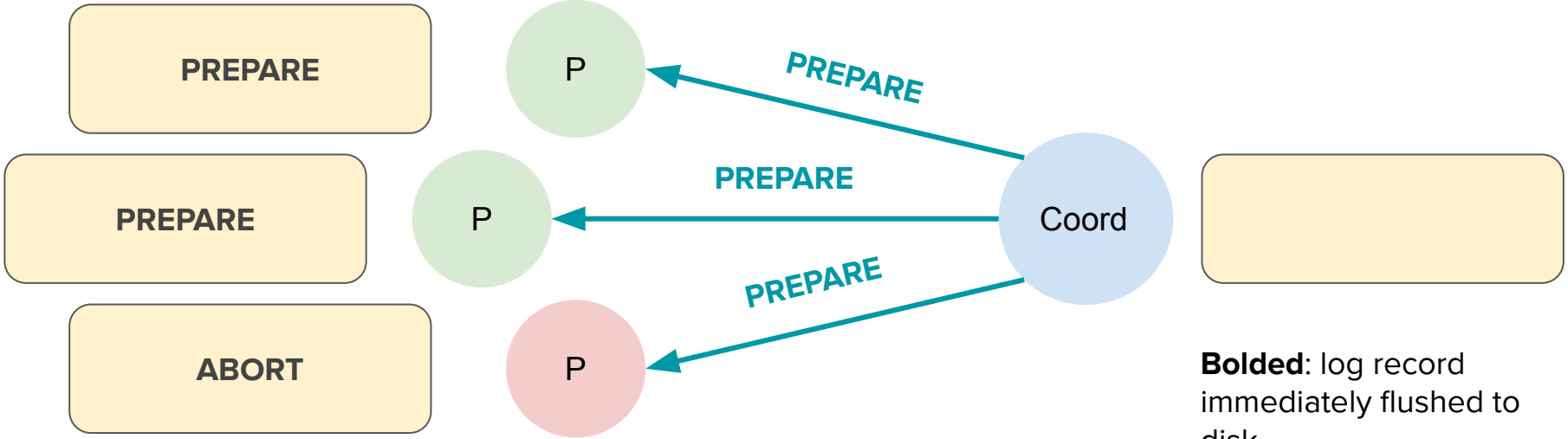


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 1:

- Participants: log *and flush* either a **PREPARE** or **ABORT** record to log
  - Log entry keeps track of coordinator ID
  - *If a participant votes NO*, it knows for sure the transaction will abort, and can begin cleaning up the transaction immediately

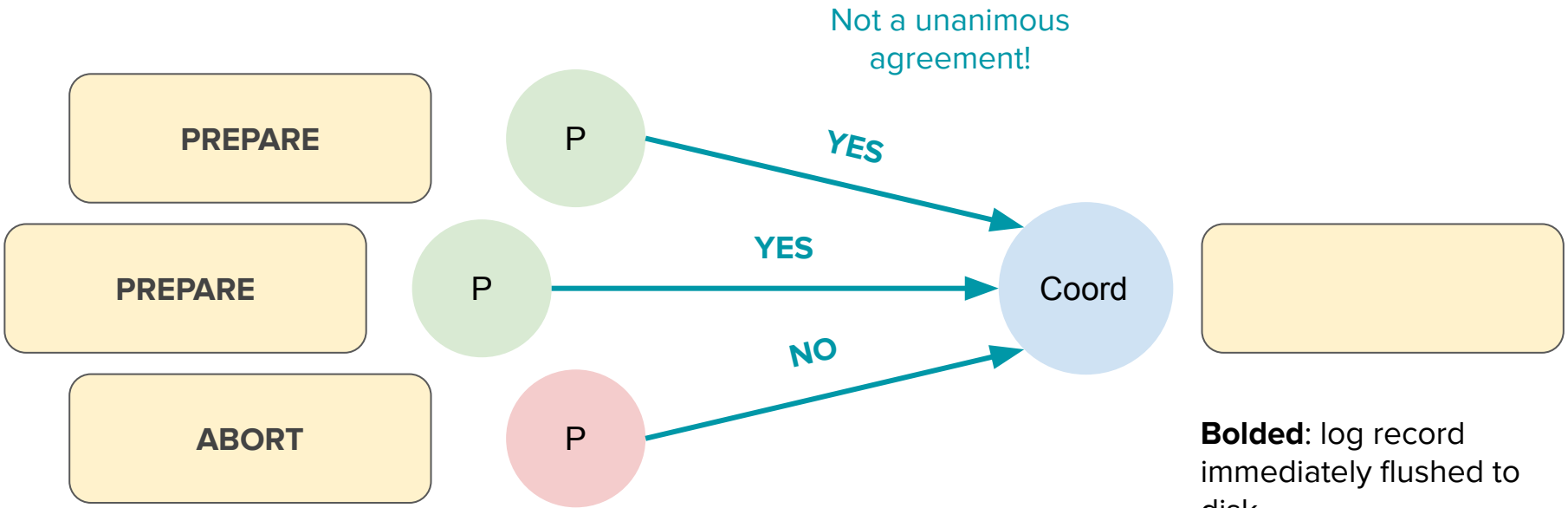


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 1:

- Participants ➜ Coordinator: VOTE YES or VOTE NO

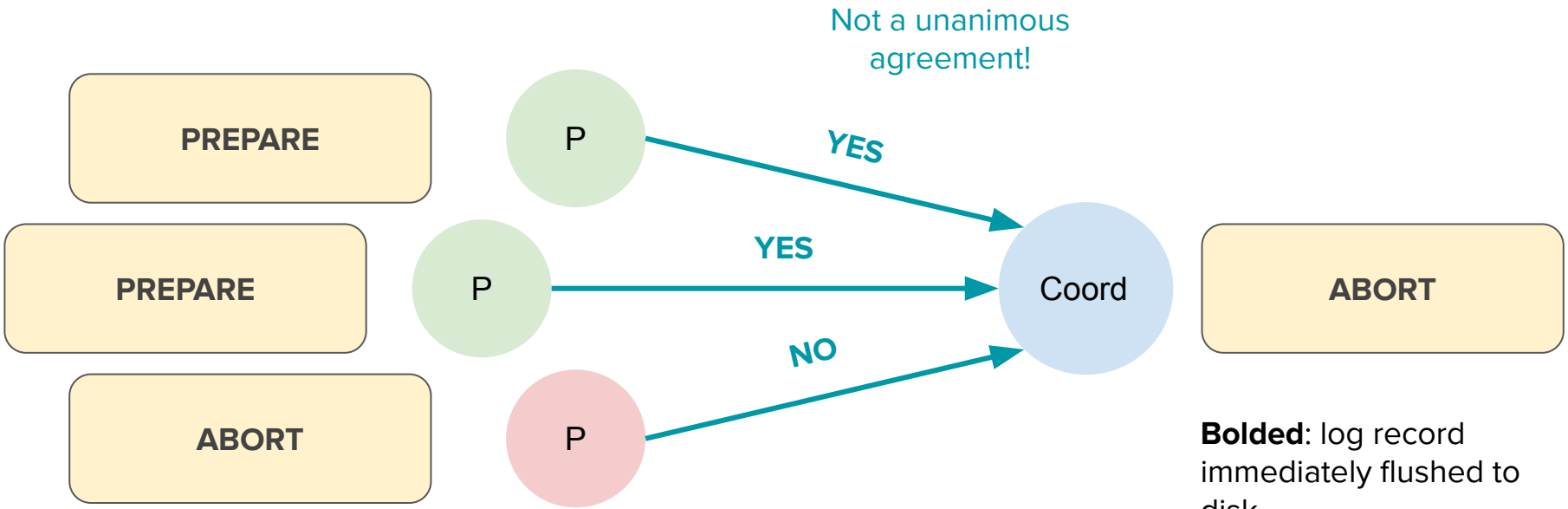


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 1:

- Coordinator: log *and flush* either a COMMIT or ABORT record to log
  - commit log entry contains all participant IDs

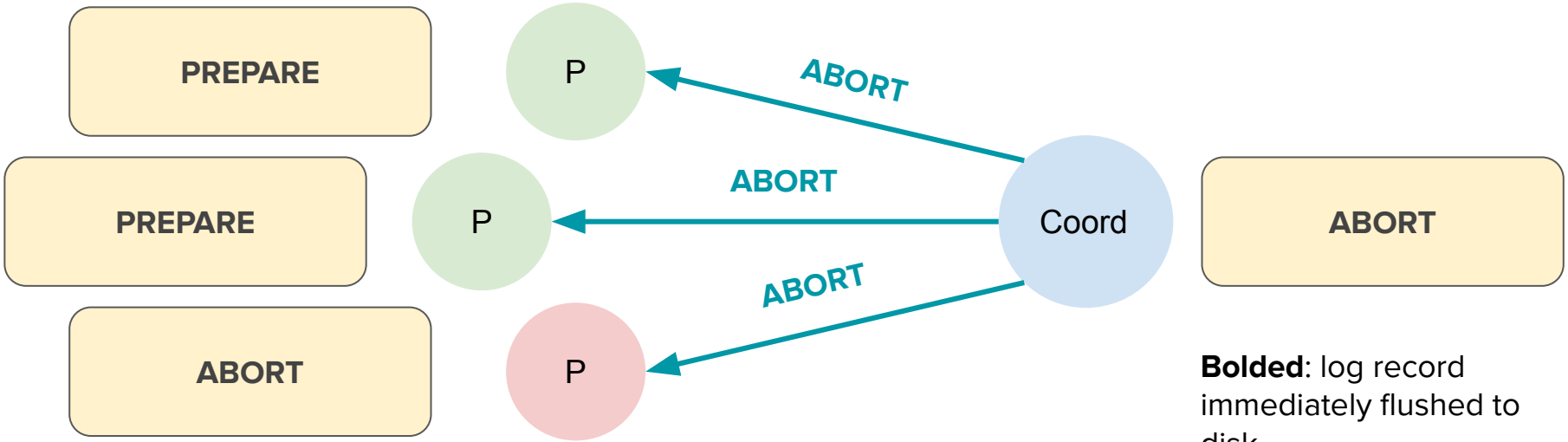


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 2:

- Coordinator → Participants: COMMIT or ABORT message

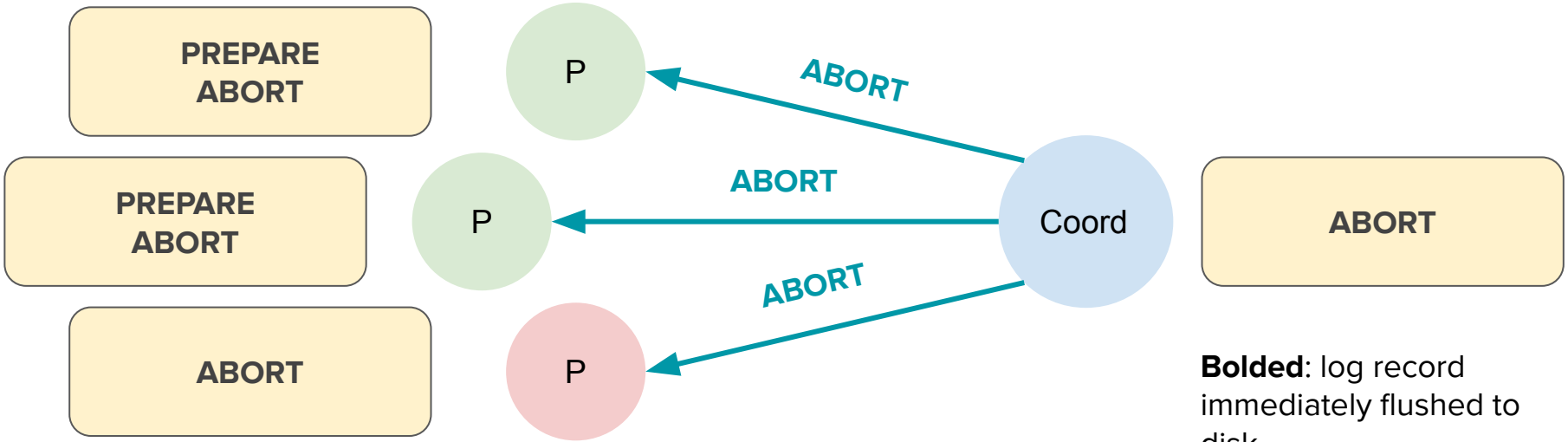


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 2:

- Participants: log *and flush* COMMIT or ABORT record to log

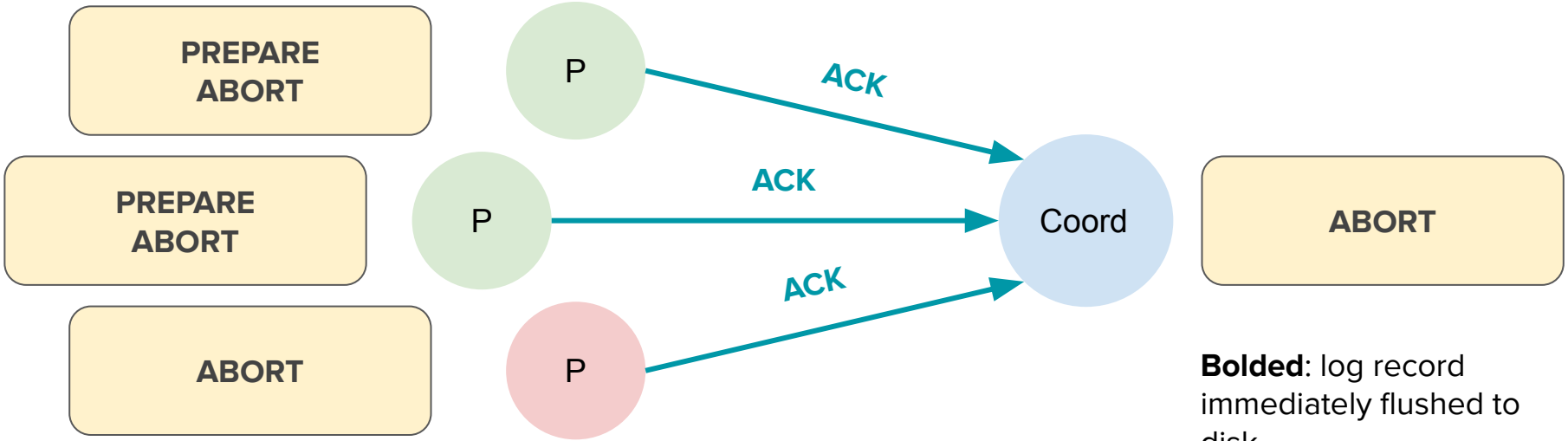


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 2:

- Participants ➜ Coordinator: ACK message

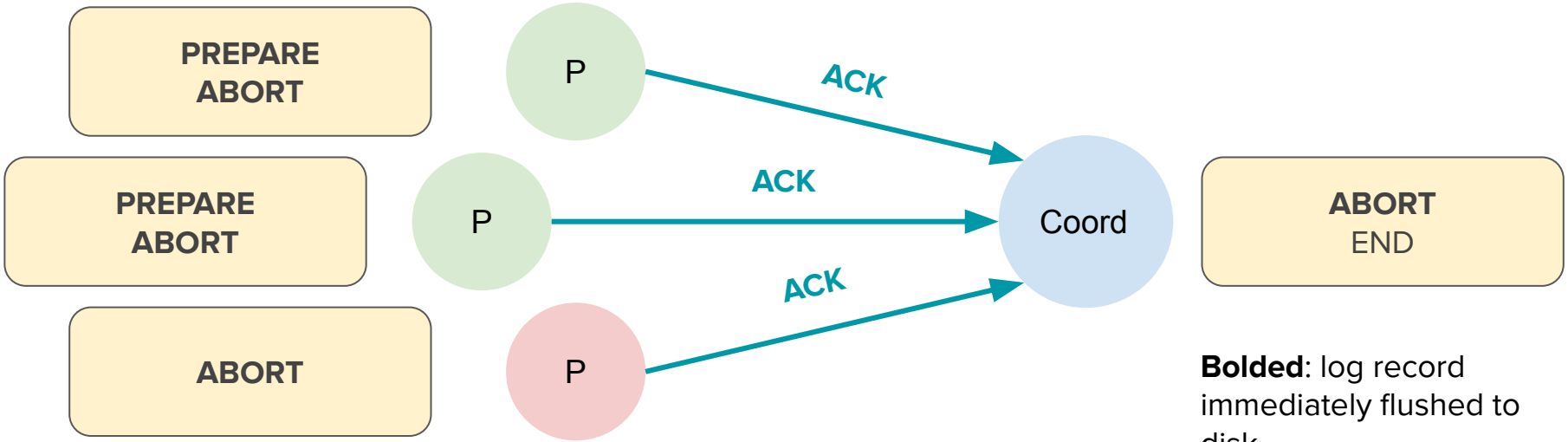


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Abort Example]

Phase 2:

- Coordinator: log (no flush required) END record to log and remove from transaction table

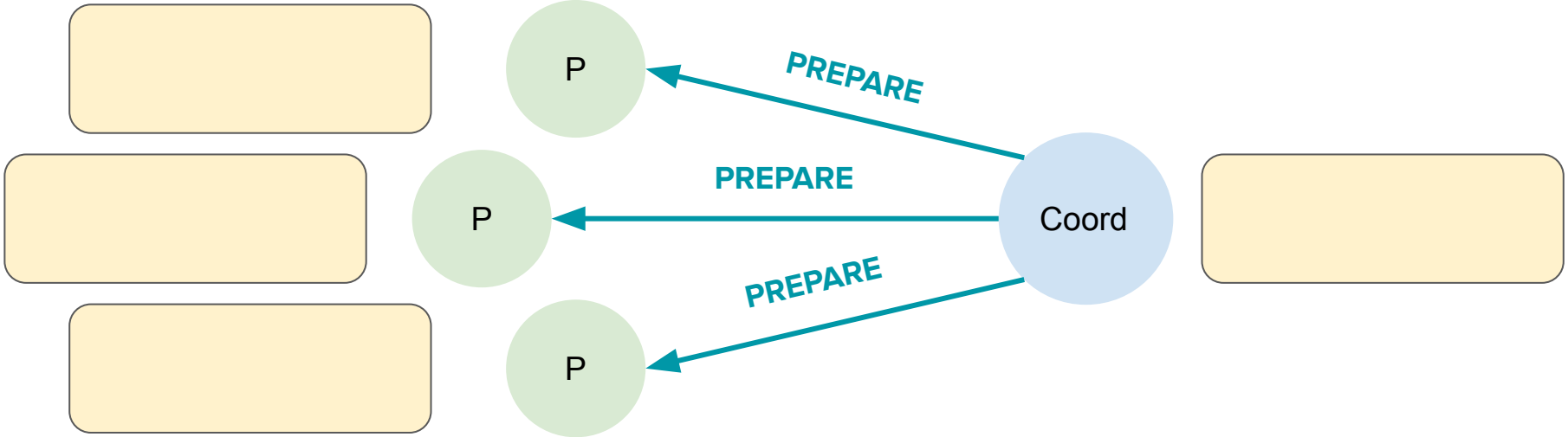


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

Phase 1:

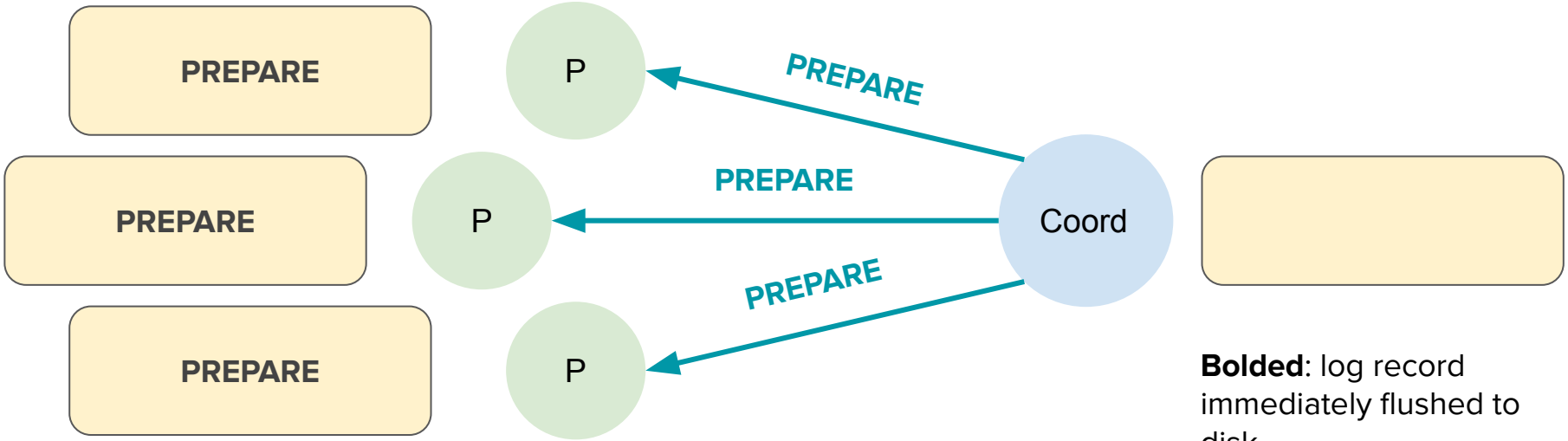
- Coordinator ➔ Participants: PREPARE message



Two-Phase Commit (2PC) [Commit Example]

Phase 1:

- Participants: log *and flush* either a **PREPARE** or **ABORT** record to log
  - Log entry keeps track of coordinator ID
  - *If a participant votes NO*, it knows for sure the transaction will abort, and can begin cleaning up the transaction immediately

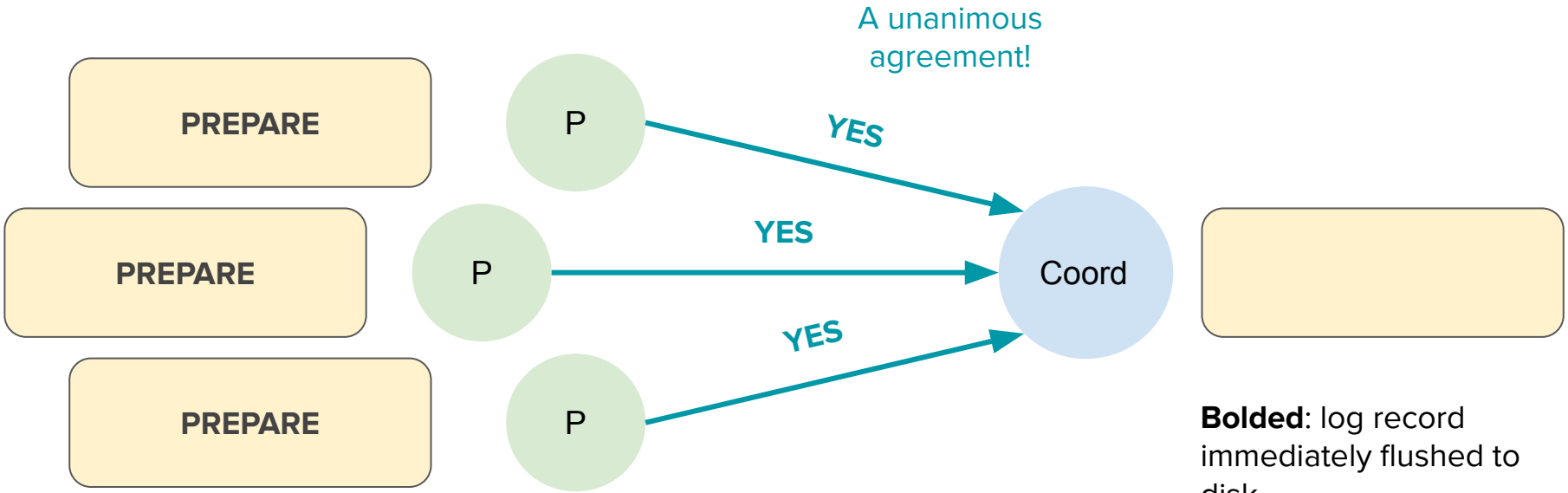


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

Phase 1:

- Participants ➔ Coordinator: VOTE YES or VOTE NO

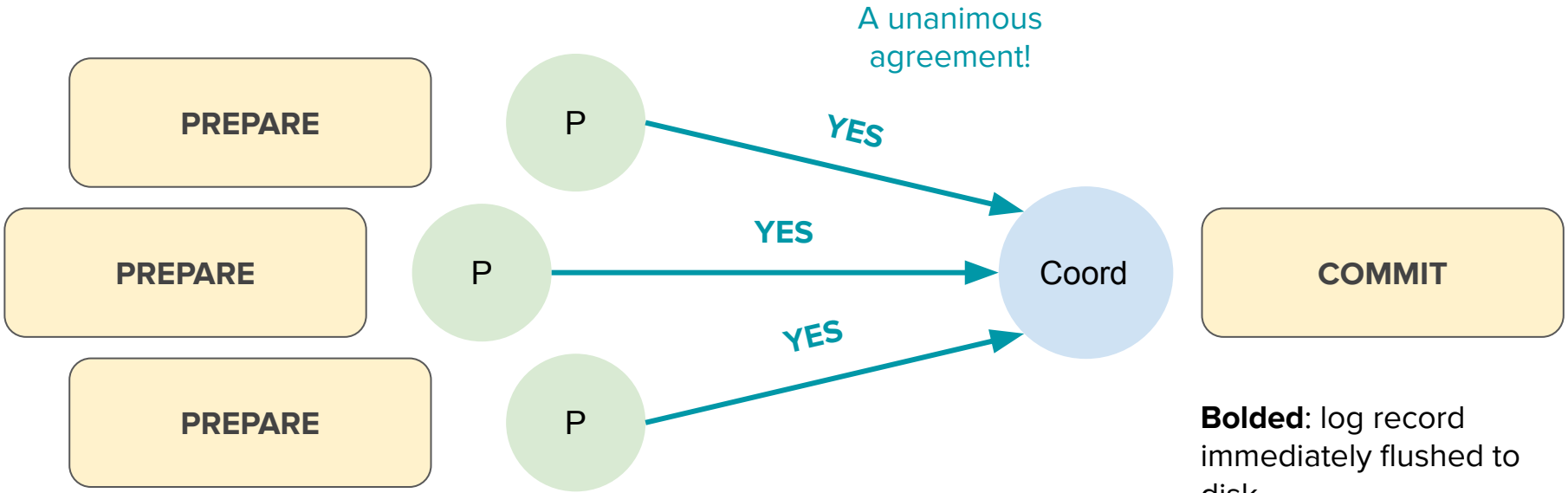


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

Phase 1:

- Coordinator: log *and flush* either a COMMIT or ABORT record to log
  - commit log entry contains all participant IDs

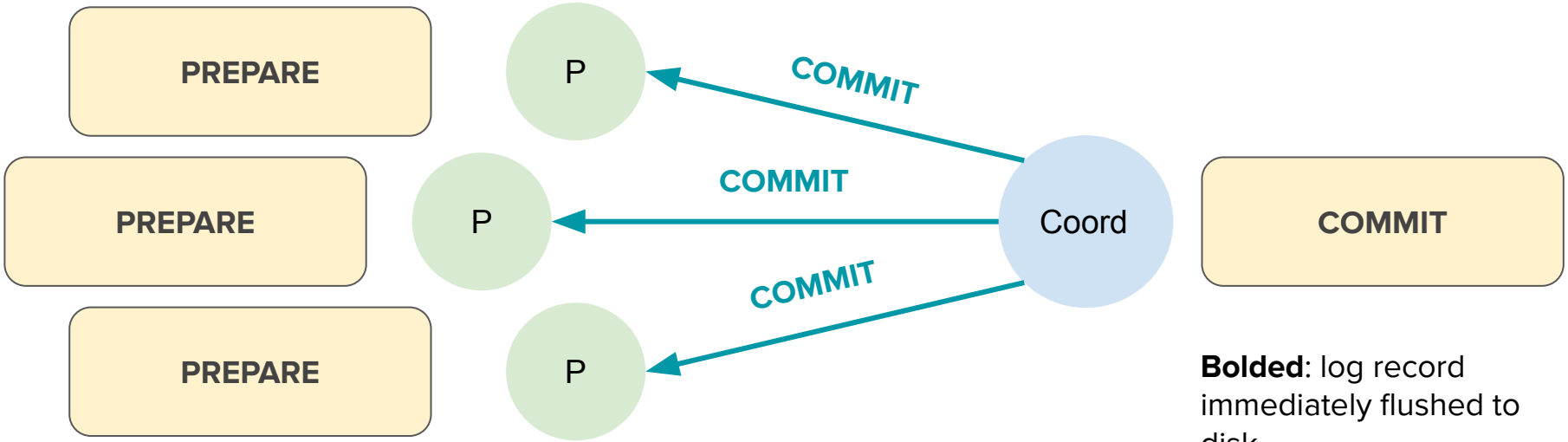


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

Phase 2:

- Coordinator → Participants: COMMIT or ABORT message

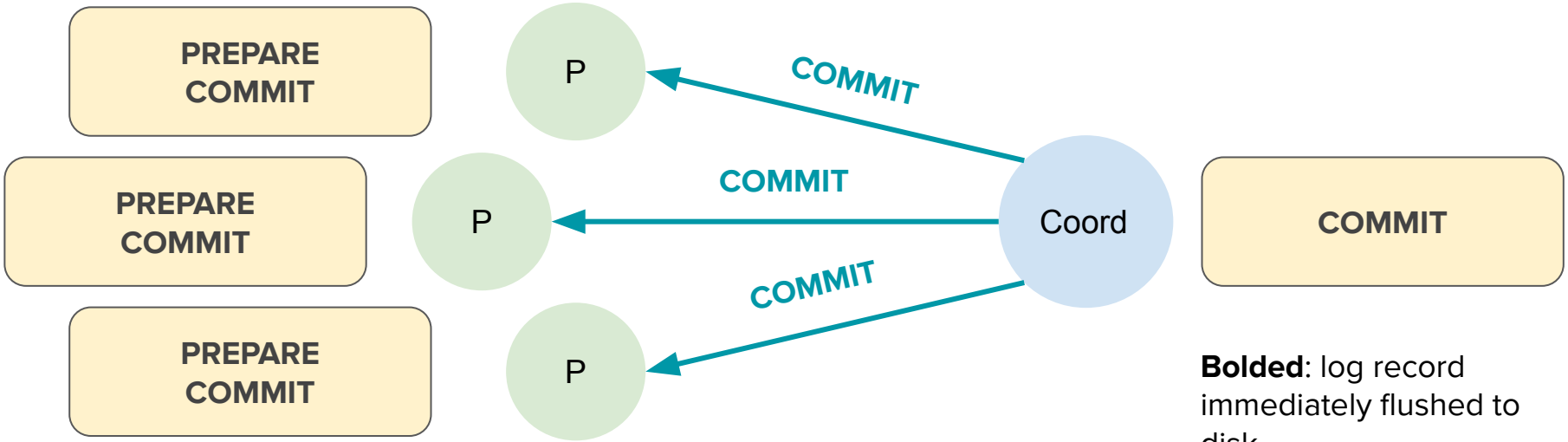


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

Phase 2:

- Participants: log *and flush* COMMIT or ABORT record to log

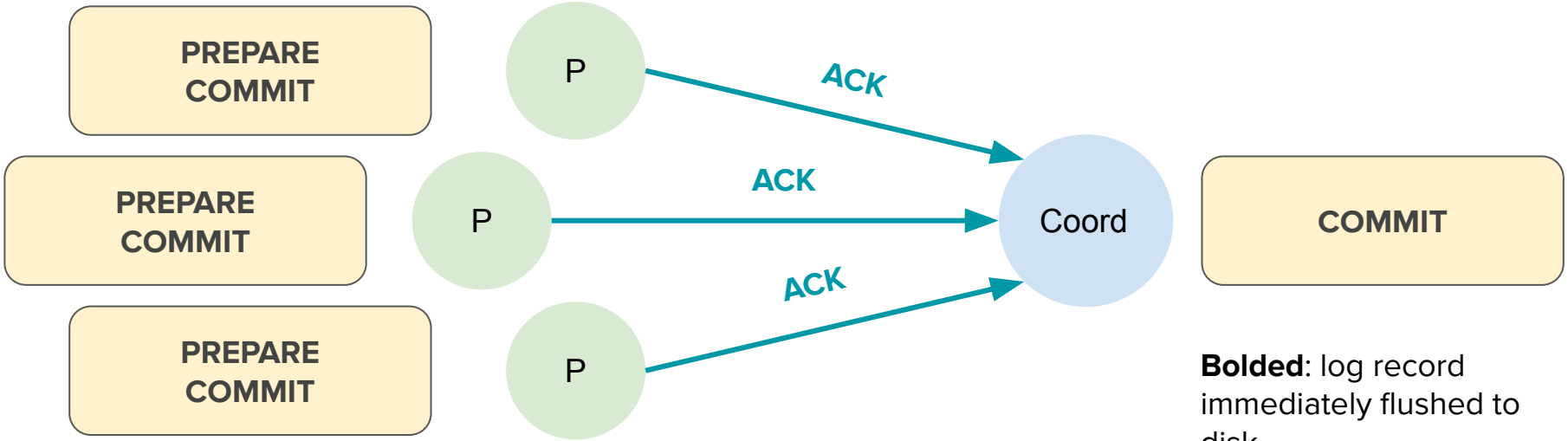


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

Phase 2:

- Participants ➜ Coordinator: ACK message

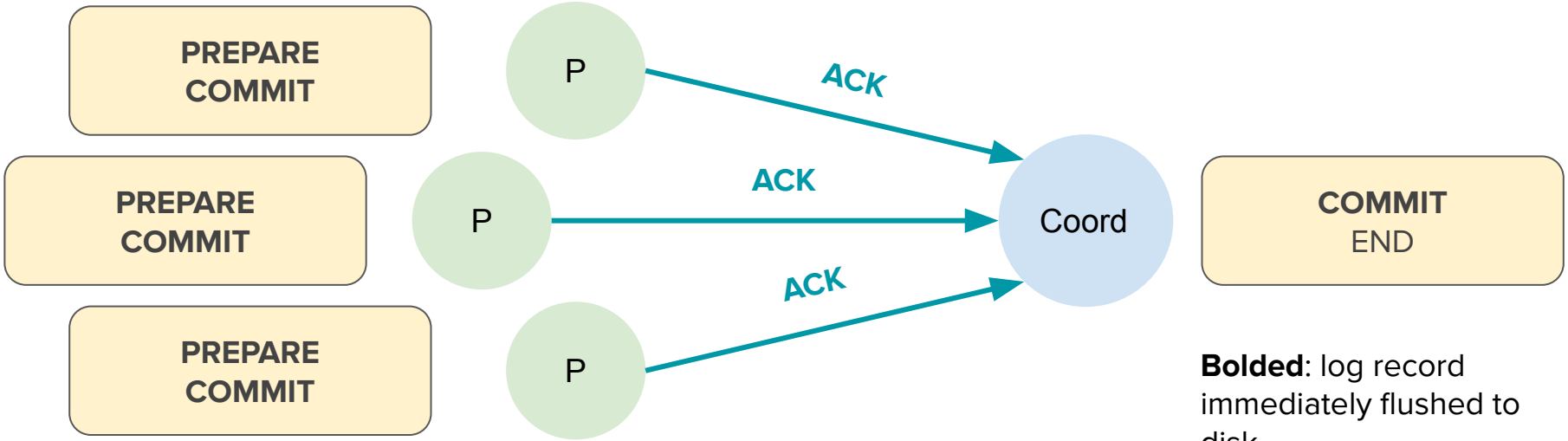


Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC) [Commit Example]

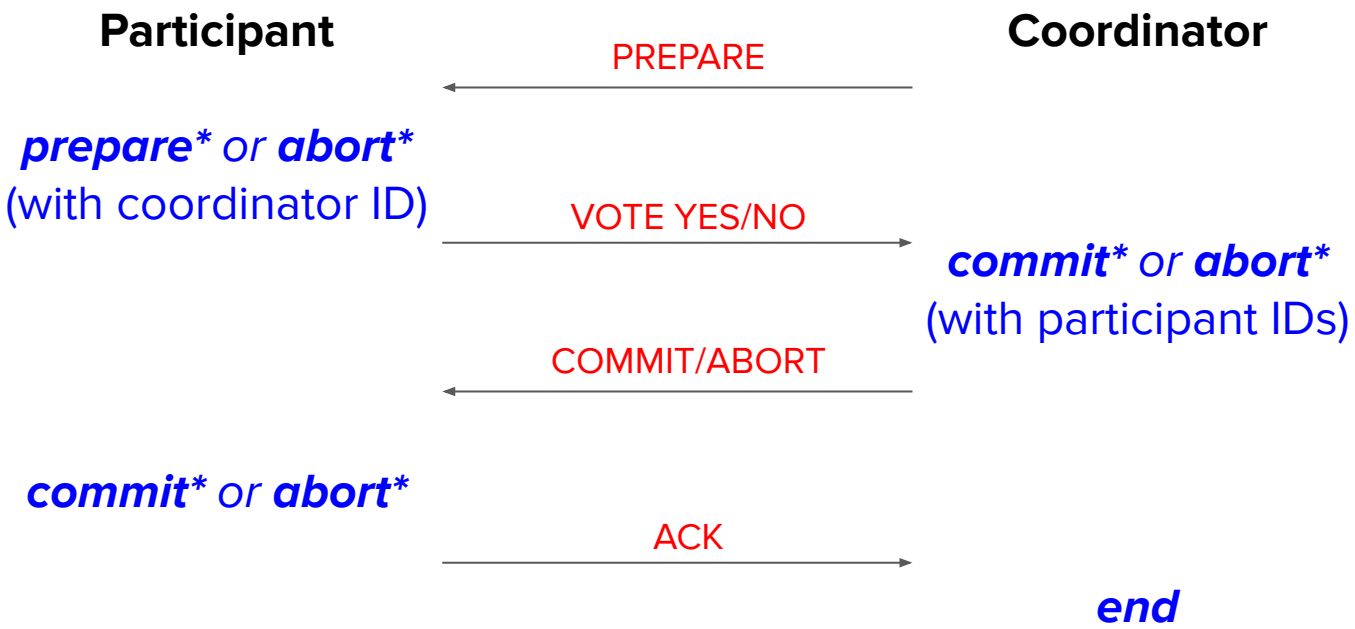
Phase 2:

- Coordinator: log (no immediate flush required) END record to log and remove from transaction table



Bolded: log record immediately flushed to disk

Two-Phase Commit (2PC): Logging



* indicates that the record must be flushed to disk immediately

Worksheet #1

We have one coordinator and three participants. It takes 30ms for a coordinator to send messages to all participants; 5, 10, and 15ms for participant 1, 2, and 3 to send a message to the coordinator respectively; and 10ms for each machine to flush a record.

Assume for the same message, each participant receives it from the coordinator at the same time. Under proper 2PC and logging protocols, how long does the whole 2PC process take (from start until all log records on the coordinator for the transaction have been flushed) for a successful commit in the best case?

Worksheet #1

We have one coordinator and three participants. It takes 30ms for a coordinator to send messages to all participants; 5, 10, and 15ms for participant 1, 2, and 3 to send a message to the coordinator respectively; and 10ms for each machine to flush a record.

Assume for the same message, each participant receives it from the coordinator at the same time. Under proper 2PC and logging protocols, how long does the whole 2PC process take (from start until all log records on the coordinator for the transaction have been flushed) for a successful commit in the best case?

Phase 1: 30ms (PREPARE) + 10ms (flush particip. prepare record) + 15ms (max time for VOTE YES messages) + 10ms (flush coord. commit record)
Phase 2: 30ms (COMMIT) + 10ms (flush particip. commit record) + 15ms (max time for ACK messages) + 10ms (flush coord. end record)
Total: 130ms (note: can take longer, since END may be flushed whenever)

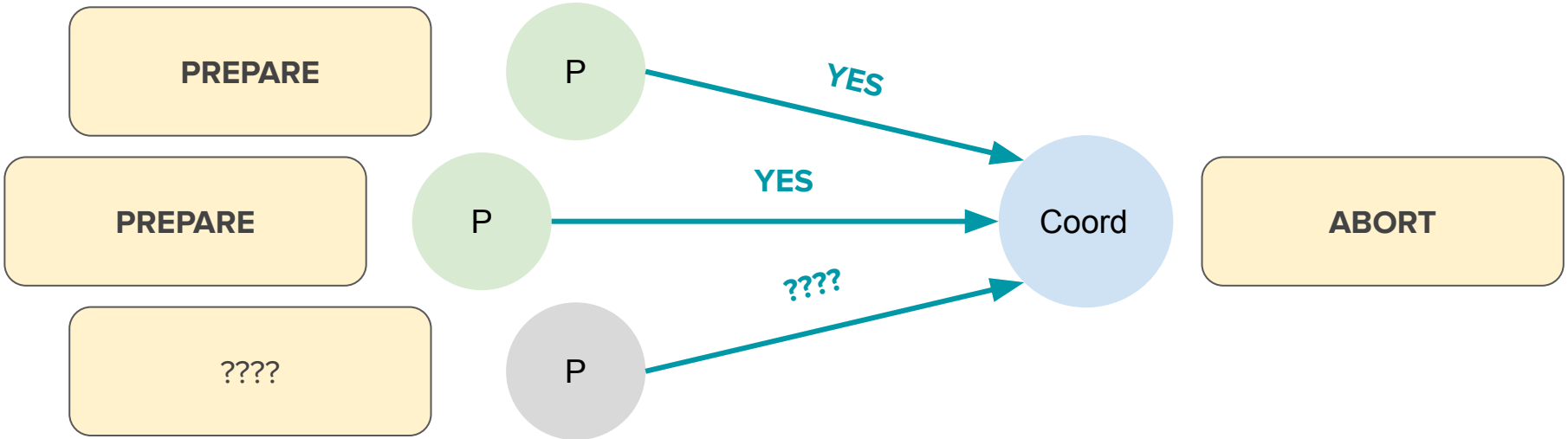
Two-Phase Commit (2PC): Handling Failures

- **We assume that all machines eventually recover**
- If coordinator *thinks* a participant went down:
  - abort transaction if participant didn't vote yet
  - perform recovery if waiting for an ACK
    - we must finish committing if we're committing
- If a participant *thinks* the coordinator went down:
  - if prepare record not logged, abort unilaterally (i.e. VOTE NO)
  - if prepare record logged, perform recovery
    - participant already voted YES, cannot proceed unilaterally
- Nodes may not actually be sure that other nodes went down (e.g. if we use a timeout, it could either be a node that went down, or a congested network)

Two-Phase Commit (2PC): Handling Failures

We assume that all machines eventually recover

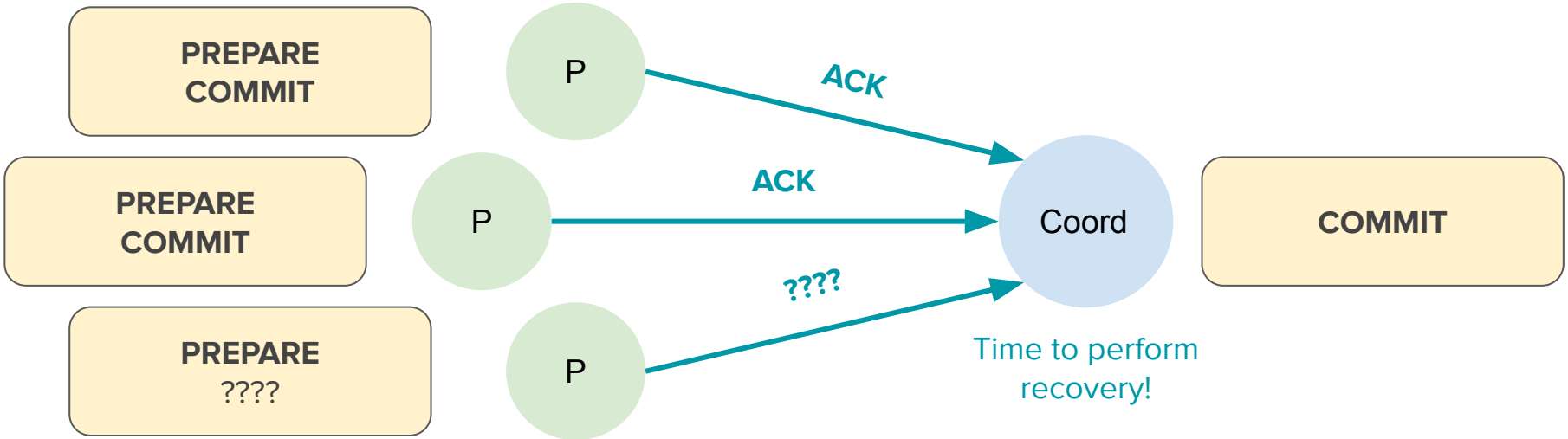
- If coordinator *thinks* a participant went down:
  - abort transaction if participant didn't vote yet
  - perform recovery if waiting for an ACK
    - we must finish committing if we're committing



Two-Phase Commit (2PC): Handling Failures

We assume that all machines eventually recover

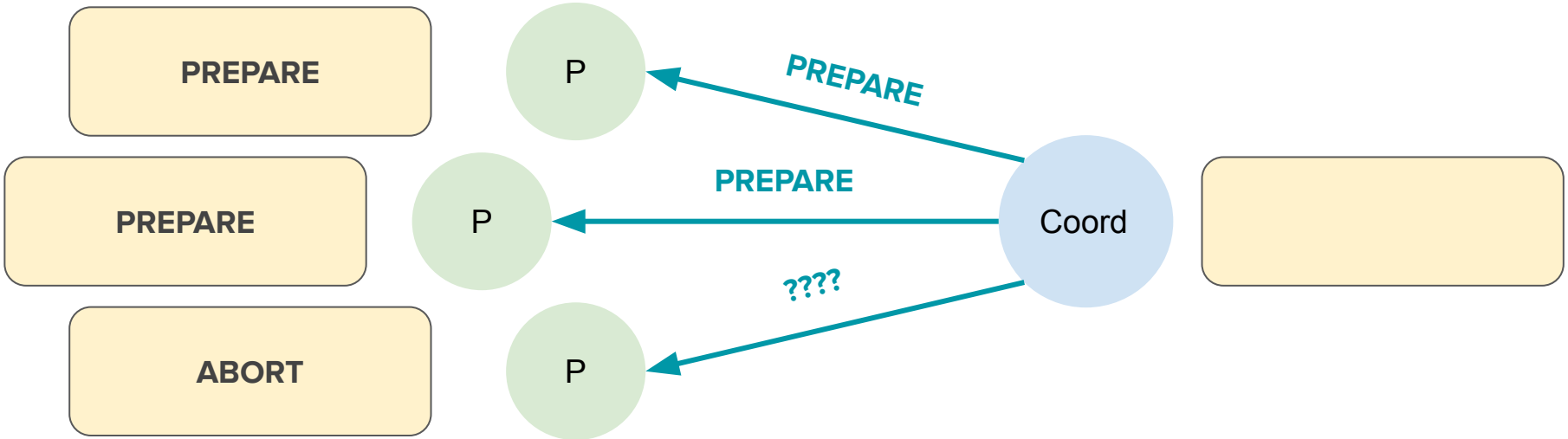
- If coordinator *thinks* a participant went down:
  - abort transaction if participant didn't vote yet
  - perform recovery if waiting for an ACK
    - we must finish committing if we're committing



Two-Phase Commit (2PC): Handling Failures

We assume that all machines eventually recover

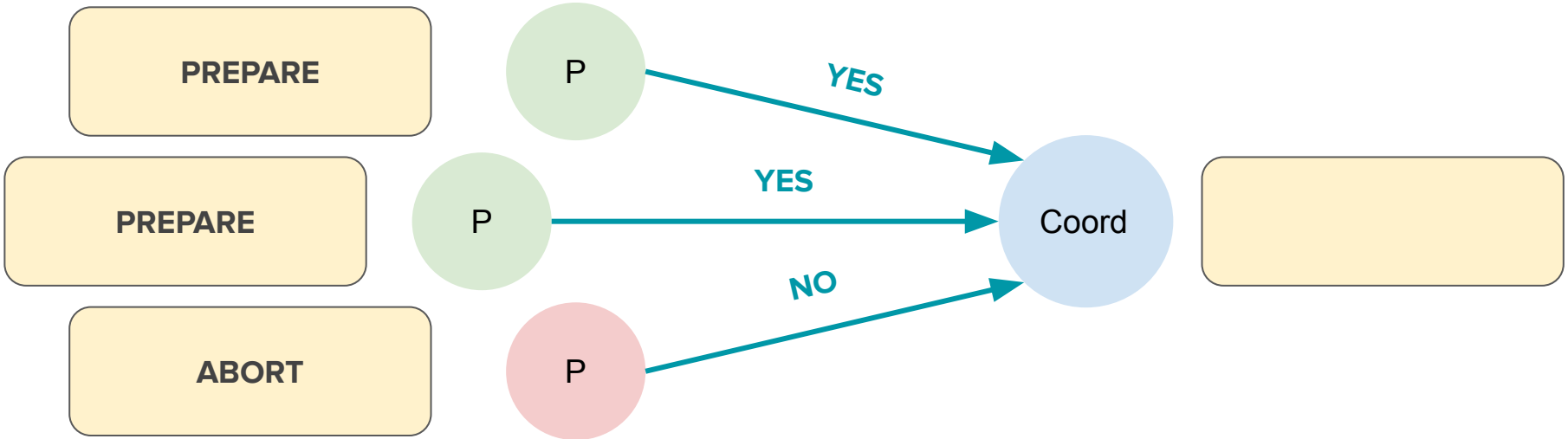
- If a participant *thinks* the coordinator went down:
  - if prepare record not logged, abort unilaterally (i.e. VOTE NO)
  - if prepare record logged, perform recovery
    - participant already voted YES, cannot proceed unilaterally



Two-Phase Commit (2PC): Handling Failures

We assume that all machines eventually recover

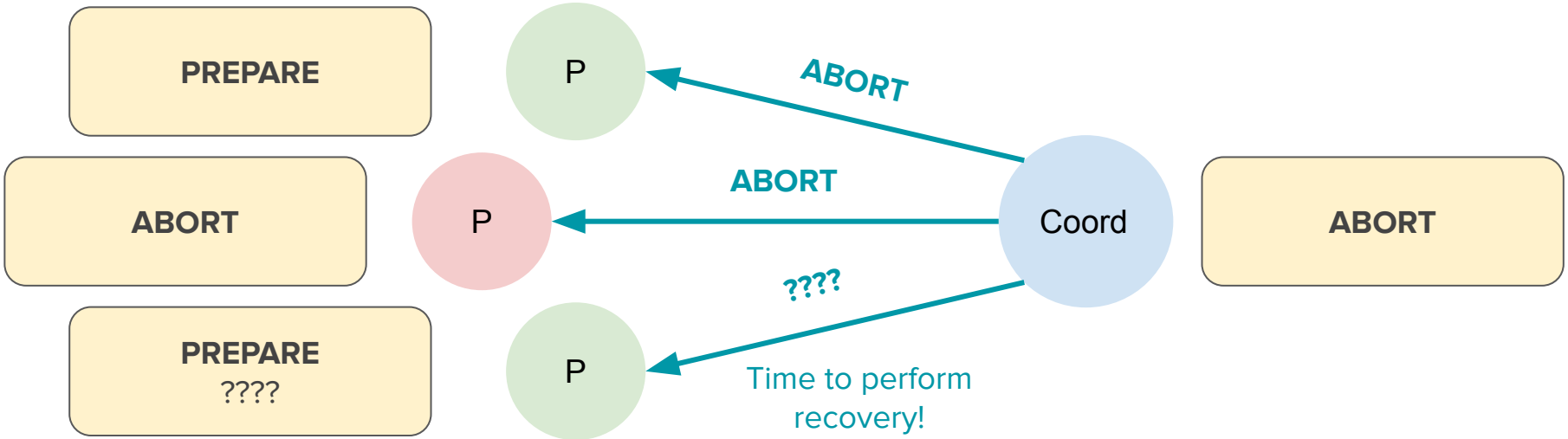
- If a participant *thinks* the coordinator went down:
  - if prepare record not logged, abort unilaterally (i.e. VOTE NO)
  - if prepare record logged, perform recovery
    - participant already voted YES, cannot proceed unilaterally



Two-Phase Commit (2PC): Handling Failures

We assume that all machines eventually recover

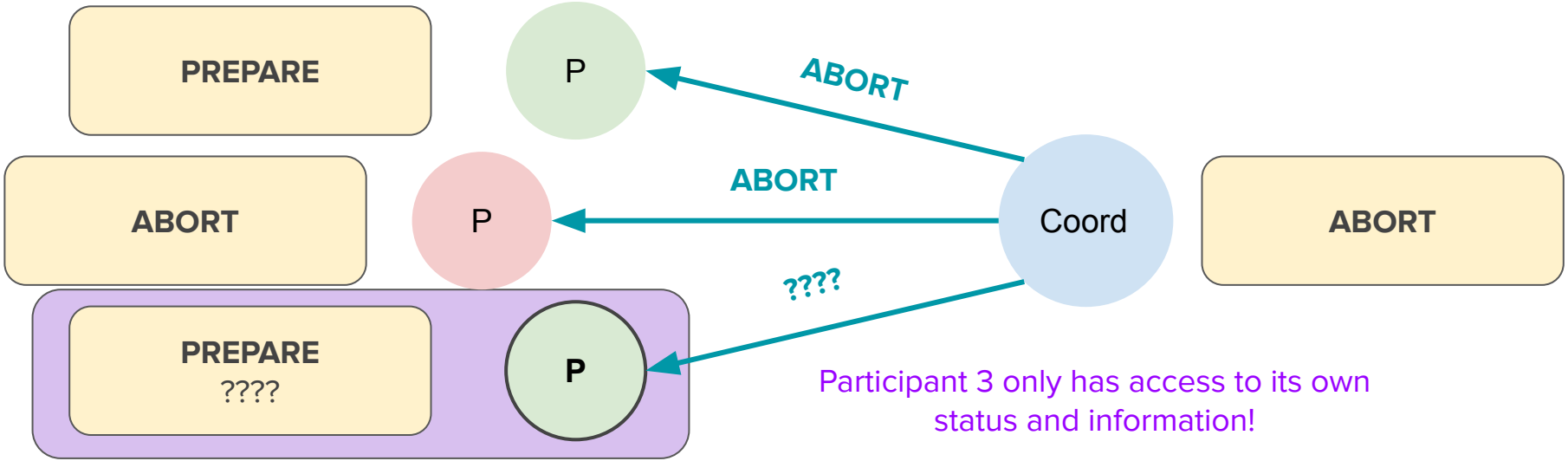
- If a participant *thinks* the coordinator went down:
  - if prepare record not logged, abort unilaterally (i.e. VOTE NO)
  - if prepare record logged, perform recovery
    - participant already voted YES, cannot proceed unilaterally



Two-Phase Commit (2PC): Handling Failures

We assume that all machines eventually recover

- Nodes may not actually be sure that other nodes went down!!
- (e.g. if we use a timeout, it could either be a node that went down, or a congested network)



Two-Phase Commit (2PC): Recovery

- If we have a commit/abort log record
  - We know what to do
  - If this is the coordinator (record contains participant IDs), periodically send COMMIT/ABORT messages until we get ACKs back
- If we have a prepare log record, but no commit/abort log record
  - This is a participant node
  - Participant sends a message to Coordinator (coordinator ID stored in prepare log record) **inquiring** about status of transaction
- If we have no prepare, commit, or abort log record
  - We did not vote YES, and can unilaterally abort the transaction
  - If we receive VOTE YES/NO messages, then this is the coordinator - respond with ABORT

Worksheet #2

Describe what happens in the 2PC protocol if a participant receives a PREPARE message, replies with a YES vote, crashes, and restarts (assume all other participants voted yes and did not crash).

Worksheet #2

Describe what happens in the 2PC protocol if a participant receives a PREPARE message, replies with a YES vote, crashes, and restarts (assume all other participants voted yes and did not crash).

The participant will send a message to the coordinator asking about the state of the transaction (committing), and then write (and flush) a COMMIT record and send an ACK message to the coordinator.

Two-Phase Commit (2PC): Optimization

- ACK message is used by coordinator to decide when we can “forget” about a transaction (delete from transaction table)
  - Must keep transaction around until it receives *all* ACKs
- If coordinator crashes after sending out PREPARE messages, we unilaterally abort the transaction (coordinator can vote no on it), and tell anyone that asks that it has aborted
  - We abort in absence of information
- This leads to an optimization called presumed abort

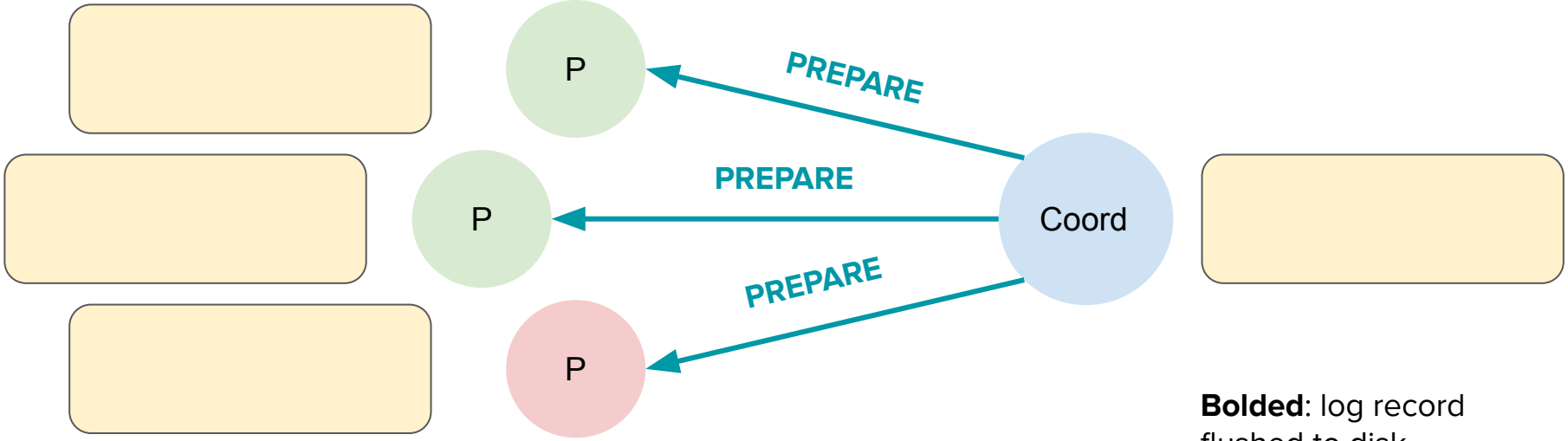
Two-Phase Commit (2PC): Optimization

- **Presumed abort**: assume that a transaction aborts if we have no log records (and reduce messages sent/logging as a result)
- When a transaction aborts,
  - Coordinator cleans up locally, and can remove transaction from table without ACKs
  - Participants that receive ABORT messages do not send ACKs
  - If Participants don't hear from Coordinator about status, send inquiry - if transaction not in coordinator's transaction table, return ABORT
  - Participant IDs do not need to be stored in abort records (since we aren't waiting for ACKs)
  - Abort records do not need to be flushed (if we crash, and don't see an abort record, we still assume it aborts)

2PC Presumed Abort [Abort Example]

Phase 1:

- Coordinator → Participants: **PREPARE** message

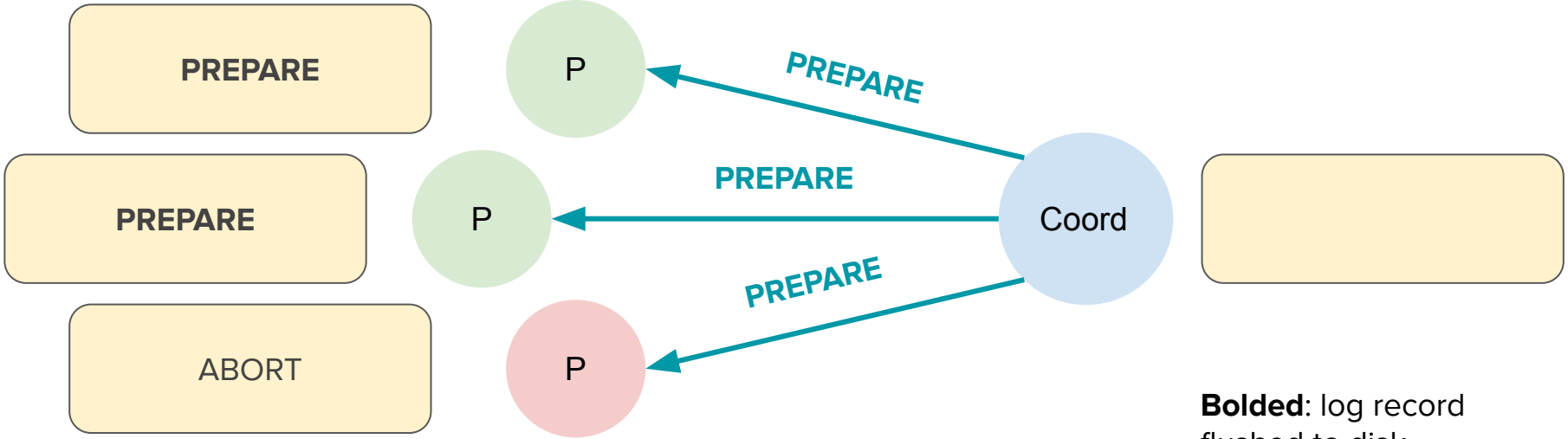


Bolded: log record flushed to disk

2PC Presumed Abort [Abort Example]

Phase 1:

- Participants: log *and flush* a **PREPARE** record, or log a **ABORT** record
  - Log entry keeps track of coordinator ID
  - *If a participant votes NO*, it doesn't need to flush the ABORT record (if we crash and don't see an ABORT record, we still assume it aborts)

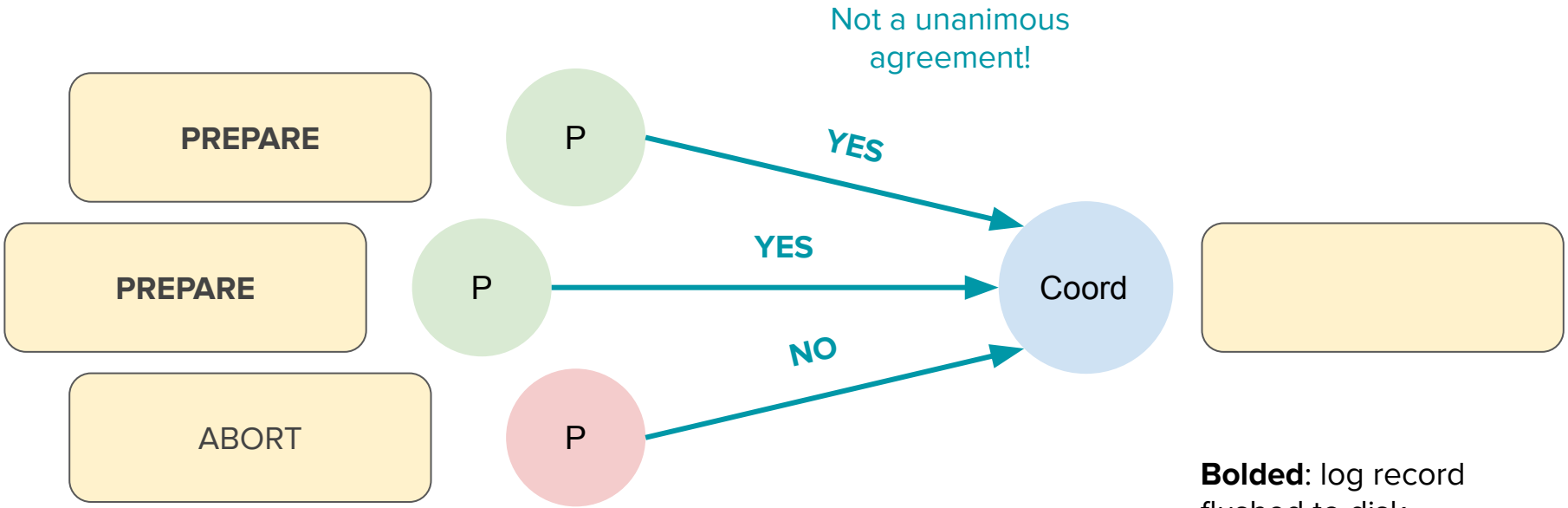


Bolded: log record flushed to disk

2PC Presumed Abort [Abort Example]

Phase 1:

- Participants → Coordinator: VOTE **YES** or VOTE **NO**

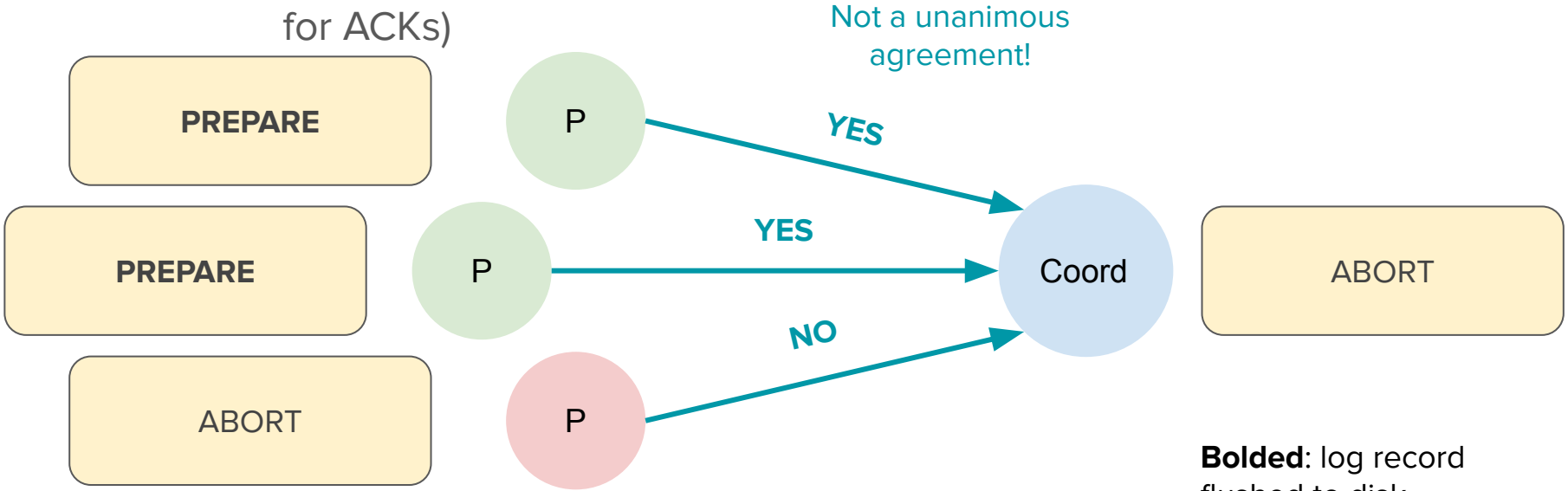


Bolded: log record flushed to disk

2PC Presumed Abort [Abort Example]

Phase 1:

- Coordinator: log *and flush* a **COMMIT** record, or log an **ABORT** record
  - COMMIT log entry contains all participant IDs
  - ABORT log entry doesn't need participant IDs (since we aren't waiting for ACKs)

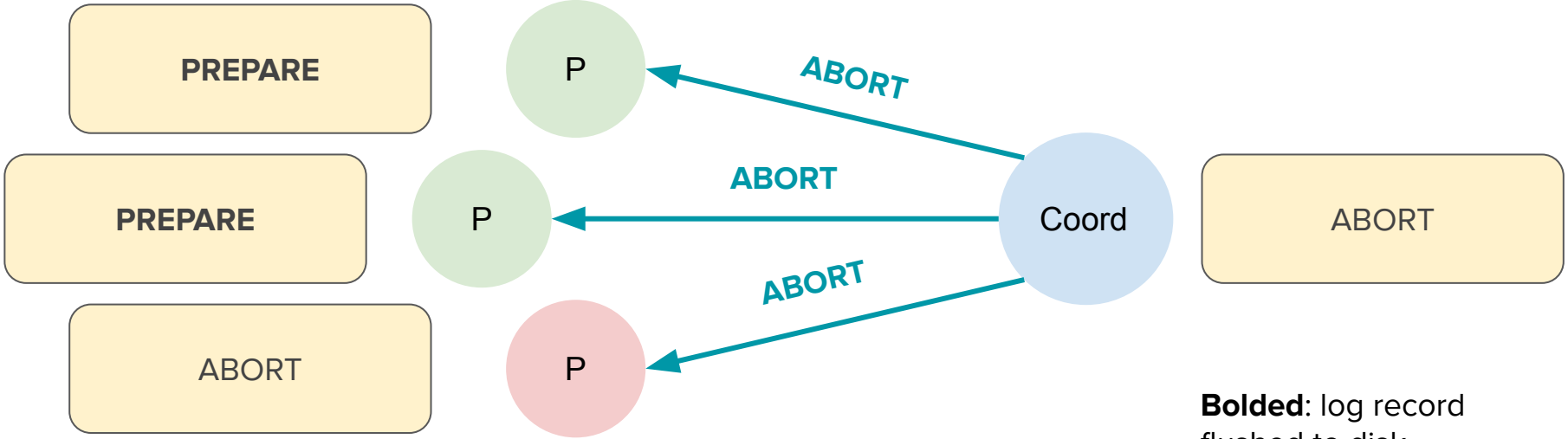


Bolded: log record flushed to disk

2PC Presumed Abort [Abort Example]

Phase 2:

- Coordinator → Participants: **COMMIT** or **ABORT** message
- Coordinator cleans up locally, can remove Xact from table without waiting for ACKs

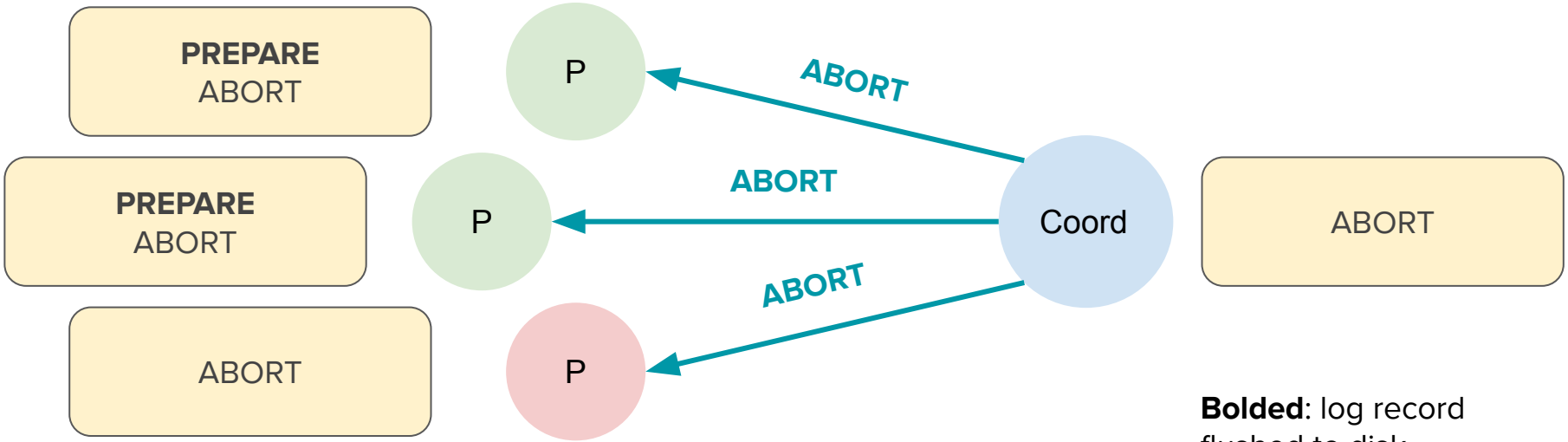


Bolded: log record flushed to disk

2PC Presumed Abort [Abort Example]

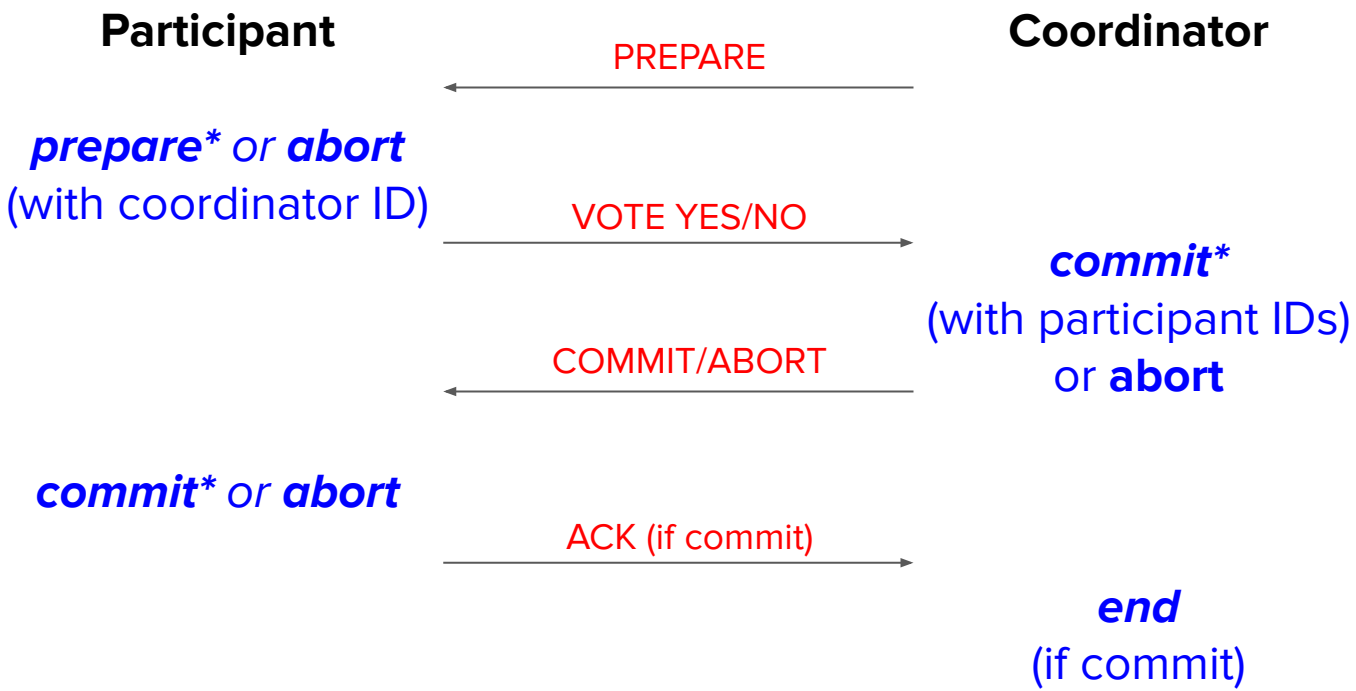
Phase 2:

- Participants: log *and flush* COMMIT or just log ABORT record
- No need for Participants to send an ACK message if they receive an ABORT message



Bolded: log record flushed to disk

Two-Phase Commit (2PC): Optimization



* indicates that the record must be flushed to disk immediately

Two-Phase Commit (2PC): Blocking

- What happens when a node goes down during the first phase?
  - Any participant that voted yes has to keep locks, waiting for commit/abort message
- What if a participant doesn't recover?
  - Coordinator can respawn participant + recover from log, and we can continue
  - (if old participant comes up again, tell it to recycle itself)
- What if the coordinator doesn't recover?
  - Pretty much screwed
  - See: 3PC, Paxos, Raft (these are not in scope)

Worksheet #3

Suppose that the coordinator sends PREPARE message to Participants 1 and 2. Participant 1 sends a "VOTE YES" message, and Participant 2 sends a "VOTE NO" message back to the coordinator.

(a) Before receiving the result of the commit/abort vote from the coordinator, Participant 1 crashes. Upon recovery, what actions does Participant 1 take 1) if we were not using presumed abort, and 2) if we were using presumed abort?

Worksheet #3

Suppose that the coordinator sends PREPARE message to Participants 1 and 2. Participant 1 sends a "VOTE YES" message, and Participant 2 sends a "VOTE NO" message back to the coordinator.

(a) Before receiving the result of the commit/abort vote from the coordinator, Participant 1 crashes. Upon recovery, what actions does Participant 1 take 1) if we were not using presumed abort, and 2) if we were using presumed abort?

Upon recovery, Participant 1 will find a PREPARE message for this transaction in their log. In both cases, Participant 1 sends an inquiry to the coordinator for the status of this transaction and sees that the transaction aborted. Participant 1 will abort the transaction locally.

Worksheet #3

Suppose that the coordinator sends PREPARE message to Participants 1 and 2. Participant 1 sends a "VOTE YES" message, and Participant 2 sends a "VOTE NO" message back to the coordinator.

(b) Before receiving the result of the commit/abort vote from the coordinator, Participant 2 crashes. Upon recovery, what actions does Participant 2 take 1) if we were not using presumed abort, and 2) if we were using presumed abort?

Worksheet #3

Suppose that the coordinator sends PREPARE message to Participants 1 and 2. Participant 1 sends a "VOTE YES" message, and Participant 2 sends a "VOTE NO" message back to the coordinator.

(b) Before receiving the result of the commit/abort vote from the coordinator, Participant 2 crashes. Upon recovery, what actions does Participant 2 take 1) if we were not using presumed abort, and 2) if we were using presumed abort?

Without presumed abort: Upon recovery, Participant 2 sees an abort record for this transaction in their log and sends the "VOTE NO" messages back to the coordinator.

With presumed abort: Upon recovery, Participant 2 sees no log records for this transaction, and they do not need to send any messages to the coordinator.

Worksheet #4

Suppose we are running 2PC *with presumed abort*. The coordinator sent PREPARE messages to the participants and crashed, before receiving any votes.

(a) What sequence of operations does the coordinator take after it recovers?

Worksheet #4

Suppose we are running 2PC *with presumed abort*. The coordinator sent PREPARE messages to the participants and crashed, before receiving any votes.

(a) What sequence of operations does the coordinator take after it recovers?

Will see transaction running at time of crash, no commit log record, and abort locally (presumed abort - no ABORT message is sent across the network).

An ABORT record is written to log (no log flush required) and CLRs are emitted as needed (from recovery).

Worksheet #4

Suppose we are running 2PC *with presumed abort*. The coordinator sent PREPARE messages to the participants and crashed, before receiving any votes.

(b) What sequence of operations does a participant who received the message and replied NO before the coordinator crashed take?

Worksheet #4

Suppose we are running 2PC *with presumed abort*. The coordinator sent PREPARE messages to the participants and crashed, before receiving any votes.

(b) What sequence of operations does a participant who received the message and replied NO before the coordinator crashed take?

The transaction aborts the local effects of the transaction without caring that the coordinator crashed.

Worksheet #4

Suppose we are running 2PC *with presumed abort*. The coordinator sent PREPARE messages to the participants and crashed, before receiving any votes.

(c) What sequence of operations does a participant who received the message and replied YES before the coordinator crashed take?

Worksheet #4

Suppose we are running 2PC *with presumed abort*. The coordinator sent PREPARE messages to the participants and crashed, before receiving any votes.

(c) What sequence of operations does a participant who received the message and replied YES before the coordinator crashed take?

The transaction cannot make any unilateral decisions. After the coordinator has recovered (as detected via timeout or some other mechanism), the participant will ask the coordinator about the state of the transaction (aborted). The recovery process will abort the transaction locally (not sending an ABORT message) by undoing its actions, if any, using the UNDO log records, and writing an abort record.

Worksheet #4

(d) Let's say that the coordinator instead crashes after successfully receiving votes from all participants, with all participants voting YES except for one NO vote. Assuming the coordinator sees no records for this transaction in its log after coming back online, how does this affect the answers to parts (a)-(c)?

Worksheet #4

(d) Let's say that the coordinator instead crashes after successfully receiving votes from all participants, with all participants voting YES except for one NO vote. Assuming the coordinator sees no records for this transaction in its log after coming back online, how does this affect the answers to parts (a)-(c)?

For part (a) - What sequence of operations does the coordinator take after it recovers?

No change because the coordinator likewise sees no log records for the transaction. The recovery process will abort the transaction locally, and since there is no information about the participant IDs in the log, the coordinator cannot send abort records to the participants.

Worksheet #4

(d) Let's say that the coordinator instead crashes after successfully receiving votes from all participants, with all participants voting YES except for one NO vote. Assuming the coordinator sees no records for this transaction in its log after coming back online, how does this affect the answers to parts (a)-(c)?

For part (b) - What sequence of operations does a participant who received the message and replied NO before the coordinator crashed take?

No change because with presumed abort, even if the participant itself crashes after sending the NO vote, the participant will proceed with abort since that is the presumption given no log records.

Worksheet #4

(d) Let's say that the coordinator instead crashes after successfully receiving votes from all participants, with all participants voting YES except for one NO vote. Assuming the coordinator sees no records for this transaction in its log after coming back online, how does this affect the answers to parts (a)-(c)?

For part (c) - What sequence of operations does a participant who received the message and replied YES before the coordinator crashed take?

No change because with presumed abort, when the coordinator comes back online and sees no information about the transaction in its log, the transaction is unilaterally aborted. This change is communicated when participants who voted YES ping the coordinator to find out how the transaction should be resolved, and the coordinator will respond with an ABORT message.

Attendance Link

https://cs186berkeley.net/attendance