Distributed Transactions with Two-Phase Commit II

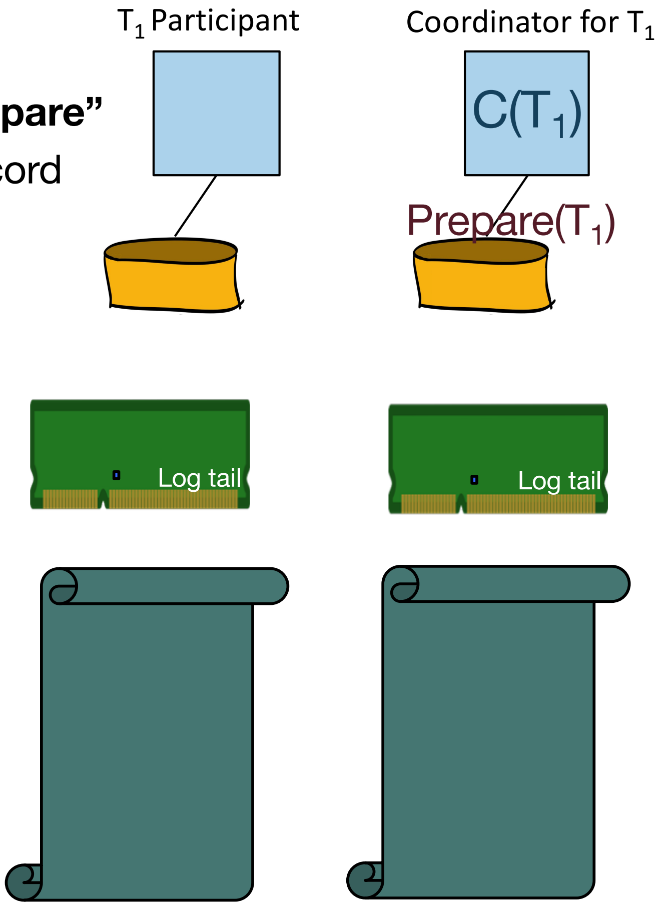
Recovery and Locking

Alvin Cheung
Aditya Parameswaran

R&G - Chapter 20

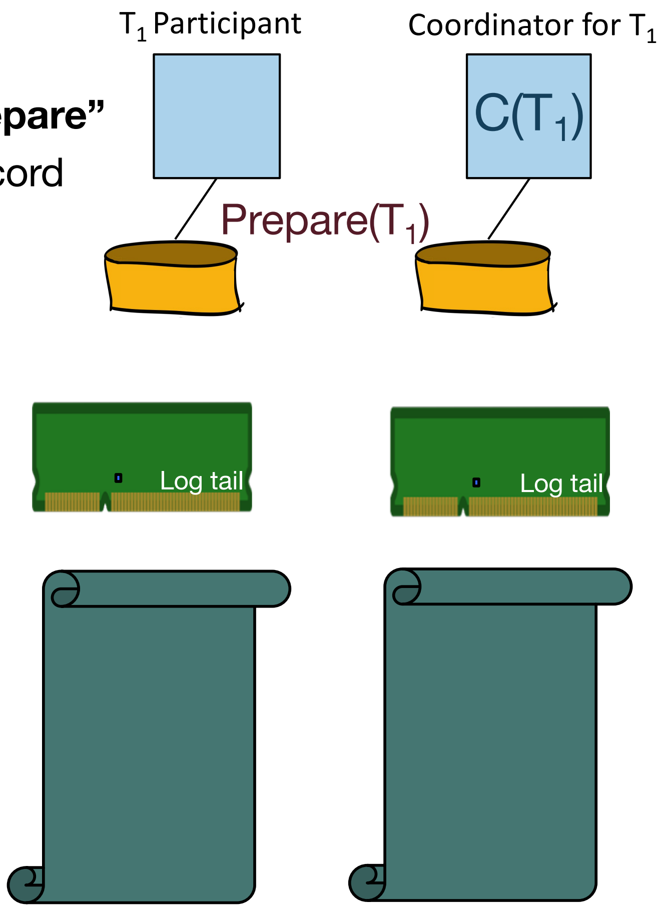
One More Time, With Logging

- **Phase 1**
- **Coordinator tells participants to "prepare"**
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- Participants respond with yes/no votes
- Coordinator generates commit record
- Coordinator flushes commit record



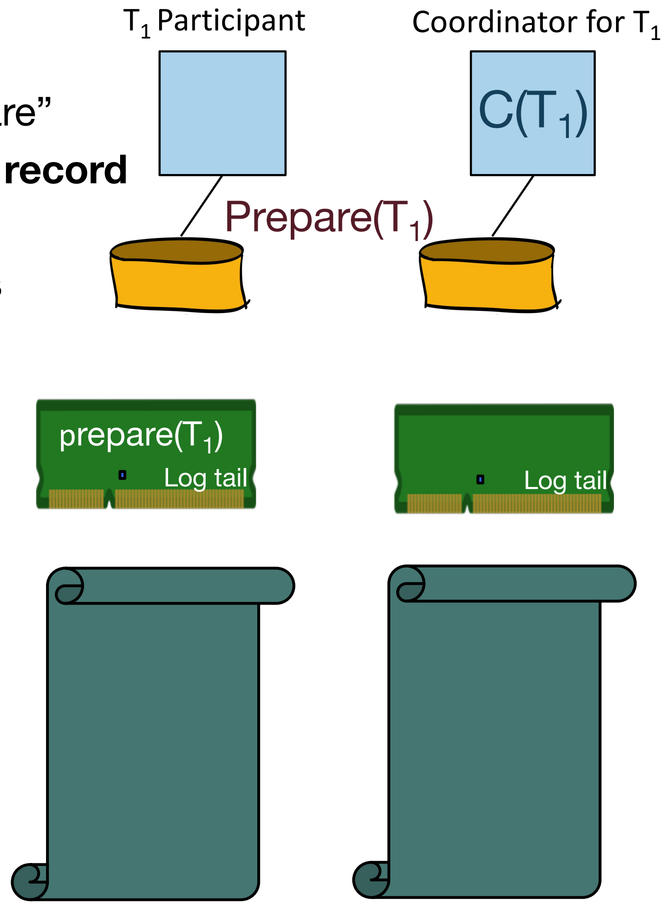
One More Time, With Logging, Part 2

- **Phase 1**
- **Coordinator tells participants to "prepare"**
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- Participants respond with yes/no votes
- Coordinator generates commit record
- Coordinator flushes commit record



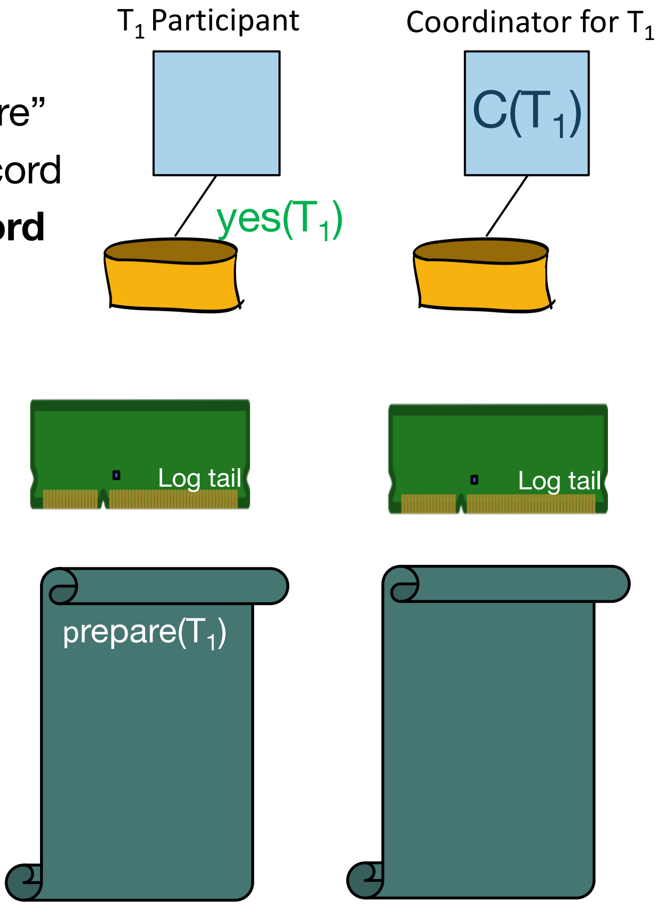
One More Time, With Logging, Part 3

- **Phase 1**
- Coordinator tells participants to “prepare”
- **Participants generate prepare/abort record**
- Participants flush prepare/abort record
- Participants respond with yes/no votes
- Coordinator generates commit record
- Coordinator flushes commit record



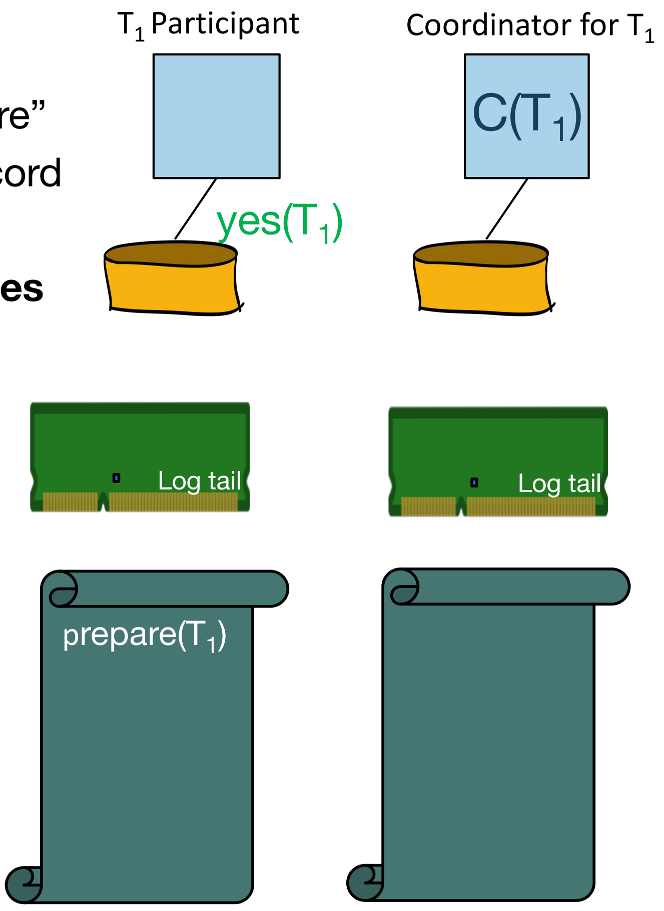
One More Time, With Logging, Part 4

- **Phase 1**
- Coordinator tells participants to “prepare”
- Participants generate prepare/abort record
- **Participants flush prepare/abort record**
- Participants respond with yes/no votes
- Coordinator generates commit record
- Coordinator flushes commit record



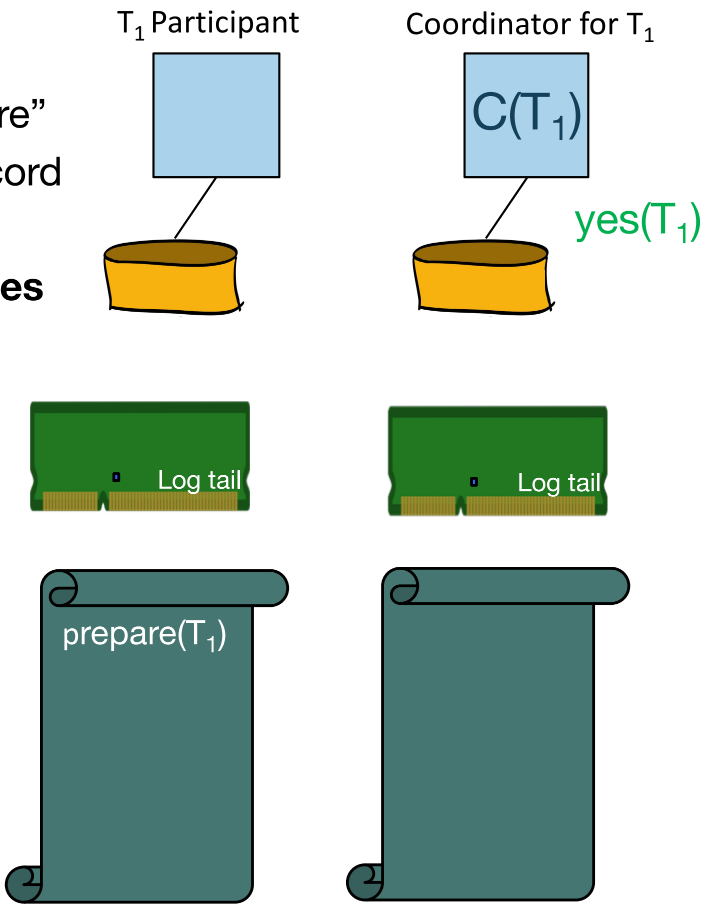
One More Time, With Logging, Part 5

- **Phase 1**
- Coordinator tells participants to “prepare”
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- **Participants respond with yes/no votes**
- Coordinator generates commit record
- Coordinator flushes commit record



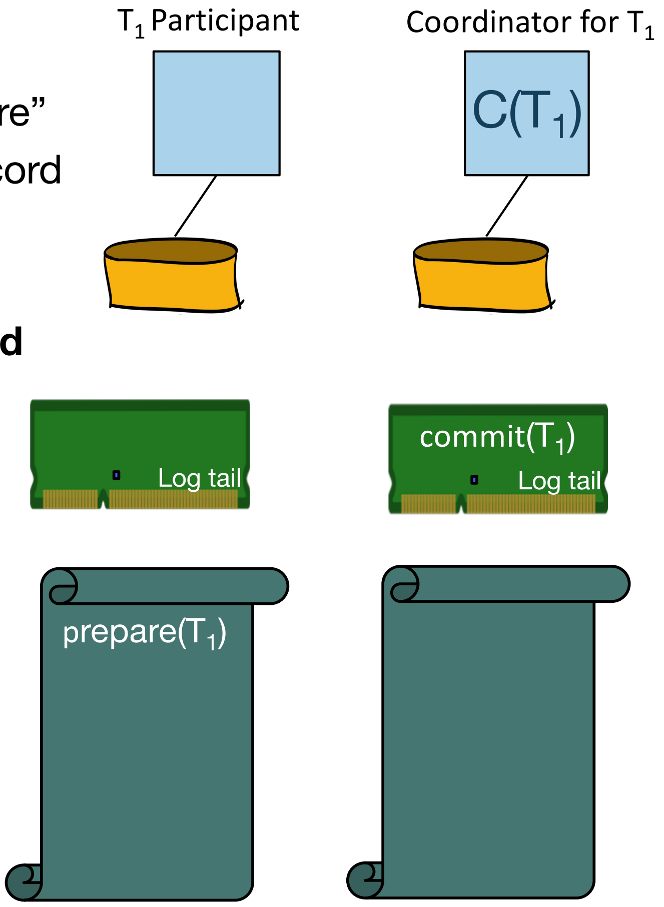
One More Time, With Logging, Part 6

- **Phase 1**
- Coordinator tells participants to “prepare”
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- **Participants respond with yes/no votes**
- Coordinator generates commit record
- Coordinator flushes commit record



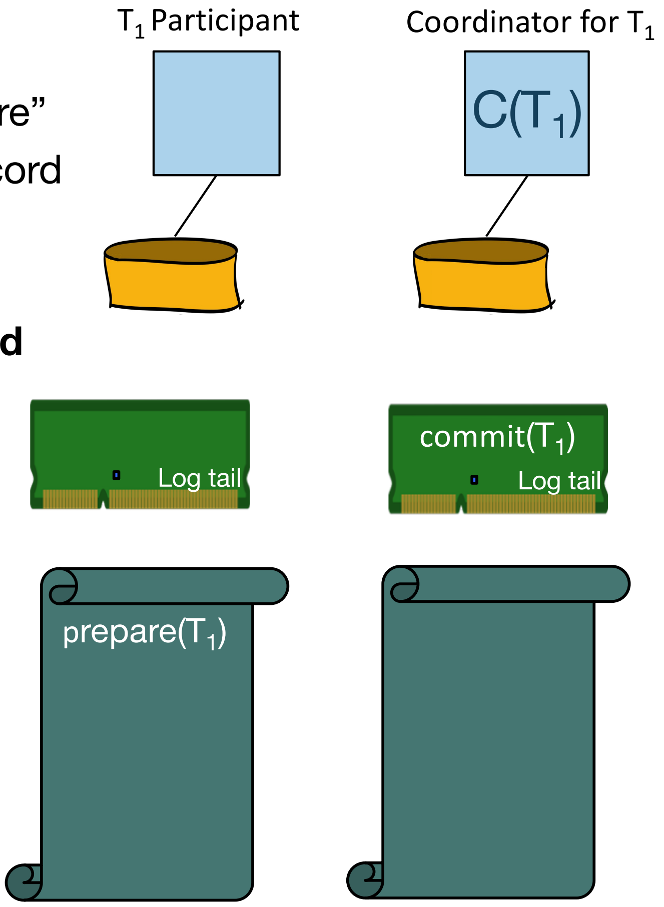
One More Time, With Logging, Part 7

- **Phase 1**
- Coordinator tells participants to “prepare”
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- Participants respond with yes/no votes
- **Coordinator generates commit record**
- Coordinator flushes commit record



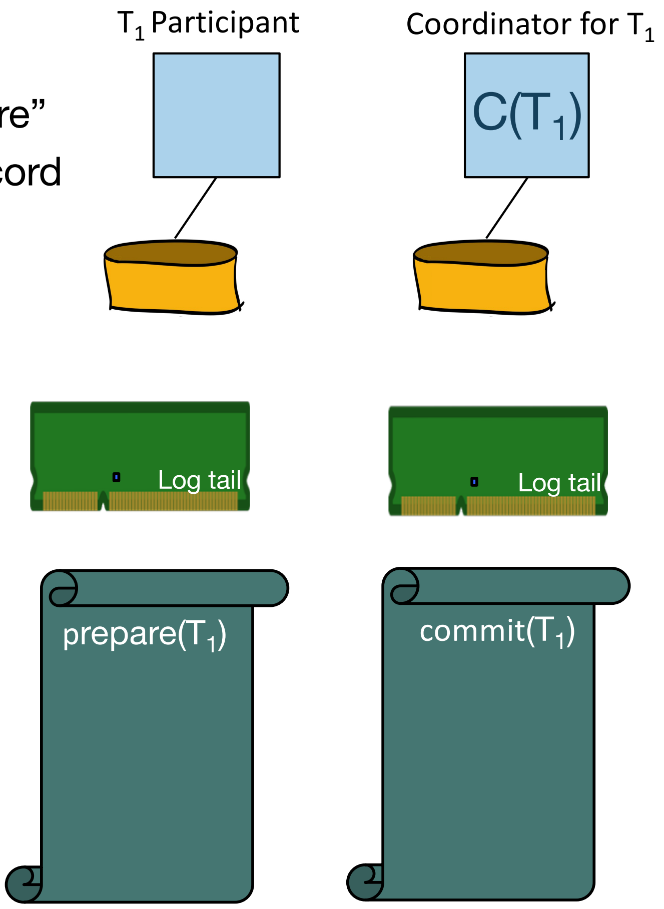
One More Time, With Logging, Part 8

- **Phase 1**
- Coordinator tells participants to "prepare"
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- Participants respond with yes/no votes
- **Coordinator generates commit record**
- Coordinator flushes commit record



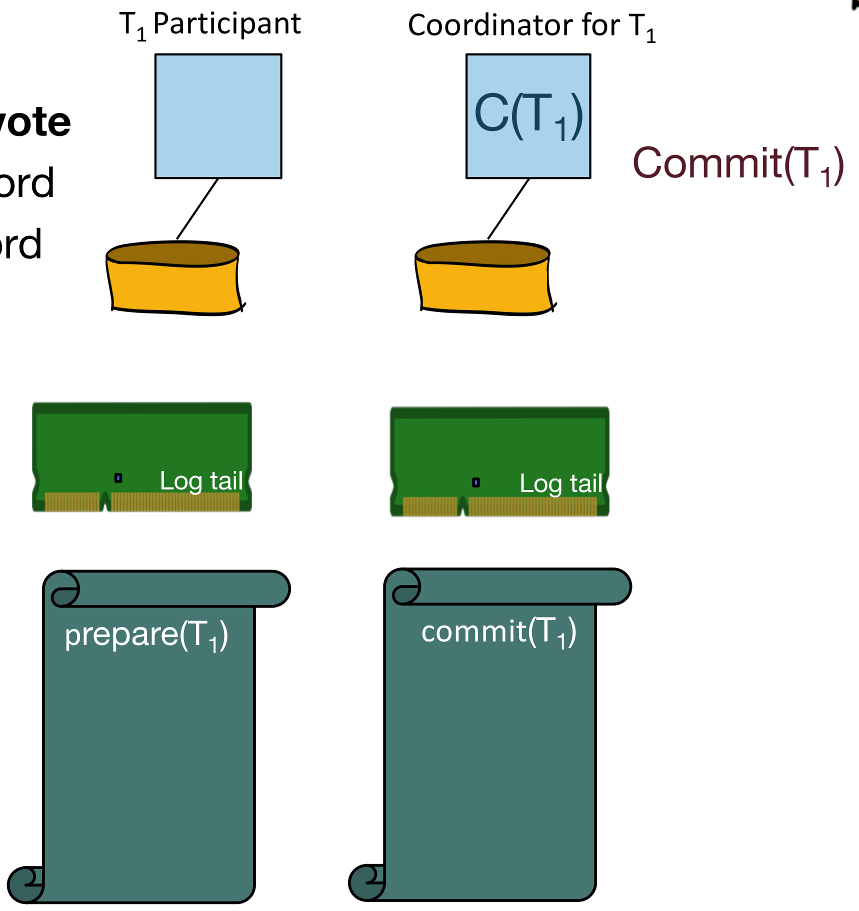
One More Time, With Logging, Part 9

- **Phase 1**
- Coordinator tells participants to “prepare”
- Participants generate prepare/abort record
- Participants flush prepare/abort record
- Participants respond with yes/no votes
- Coordinator generates commit record
- **Coordinator flushes commit record**



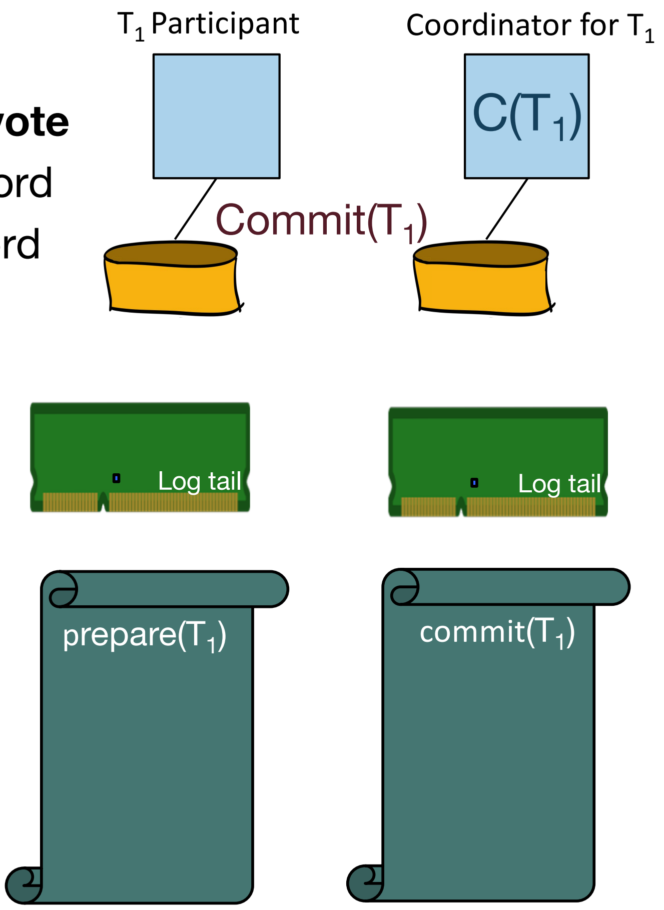
One More Time, With Logging, Part 10

- **Phase 2:**
- **Coordinator broadcasts result of vote**
- Participants make commit/abort record
- Participants flush commit/abort record
- Participants respond with Ack
- Coordinator generates end record
- Coordinator flushes end record



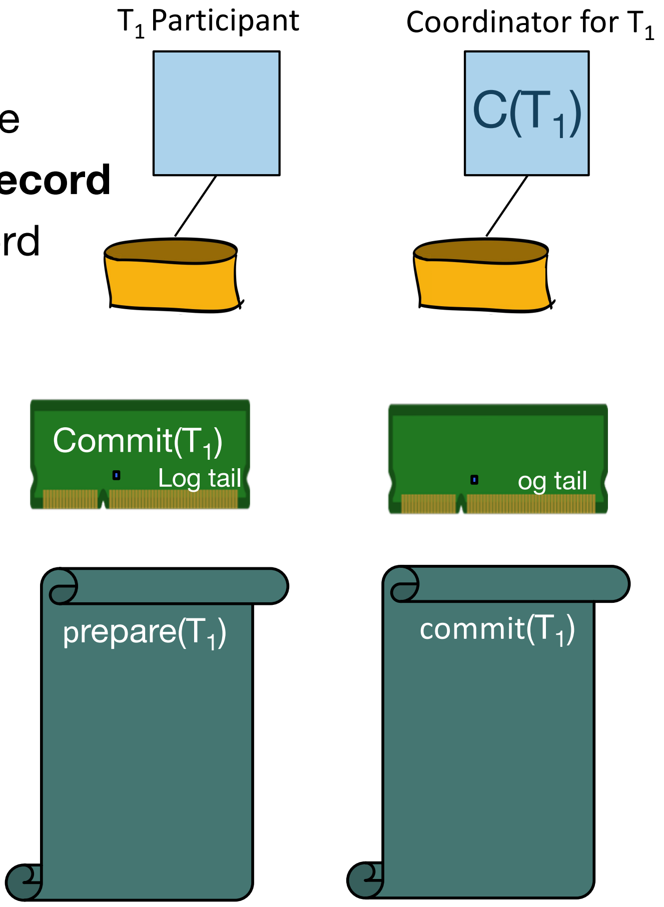
One More Time, With Logging, Part 11

- **Phase 2:**
- **Coordinator broadcasts result of vote**
- Participants make commit/abort record
- Participants flush commit/abort record
- Participants respond with Ack
- Coordinator generates end record
- Coordinator flushes end record



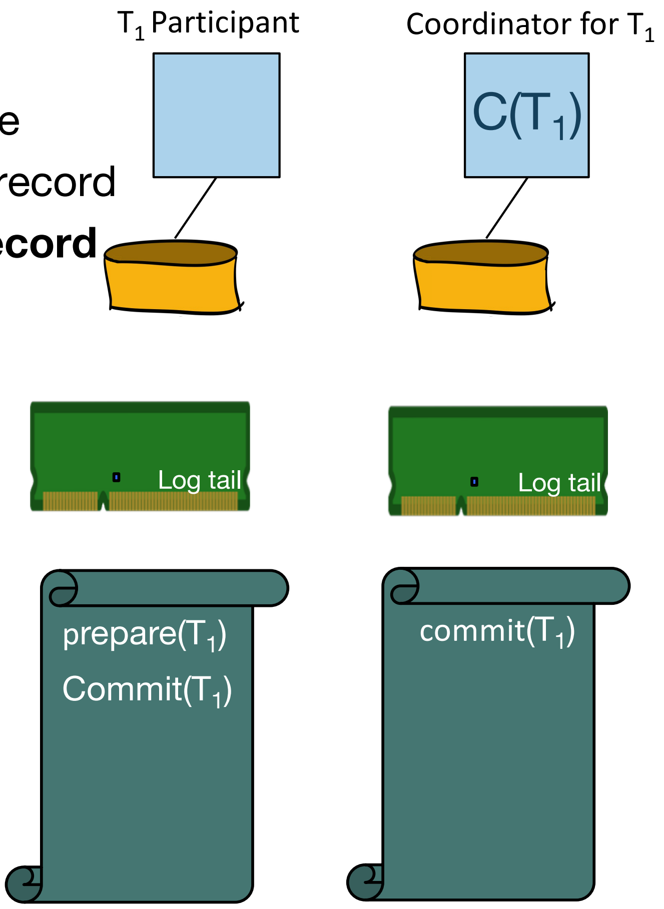
One More Time, With Logging, Part 12

- **Phase 2:**
- Coordinator broadcasts result of vote
- **Participants make commit/abort record**
- Participants flush commit/abort record
- Participants respond with Ack
- Coordinator generates end record
- Coordinator flushes end record



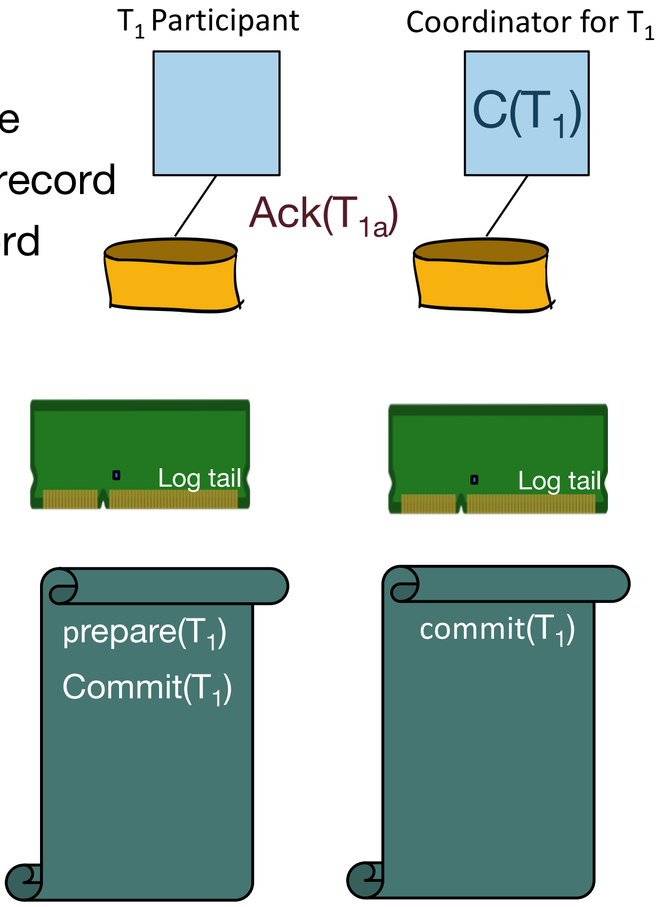
One More Time, With Logging, Part 13

- **Phase 2:**
- Coordinator broadcasts result of vote
- Participants generate commit/abort record
- **Participants flush commit/abort record**
- Participants respond with Ack
- Coordinator generates end record
- Coordinator flushes end record



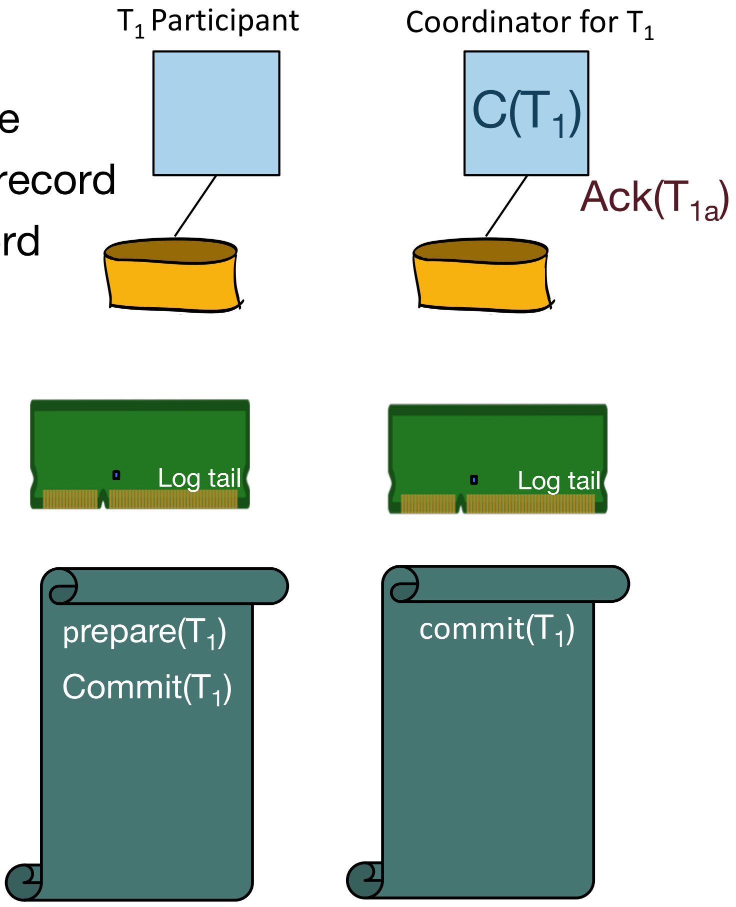
One More Time, With Logging, Part 14

- **Phase 2:**
- Coordinator broadcasts result of vote
- Participants generate commit/abort record
- Participants flush commit/abort record
- **Participants respond with Ack**
- Coordinator generates end record
- Coordinator flushes end record



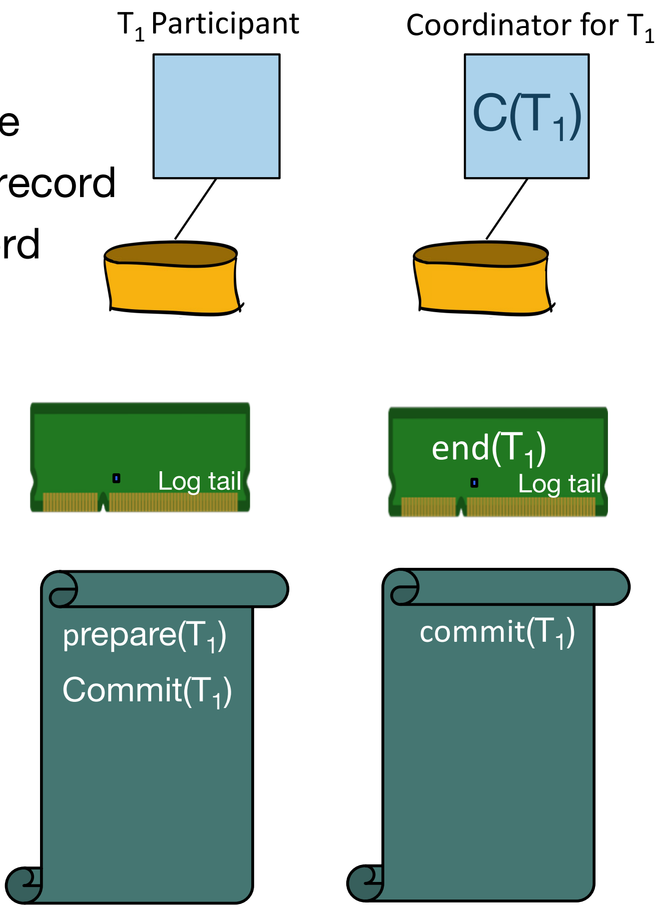
One More Time, With Logging, Part 15

- **Phase 2:**
- Coordinator broadcasts result of vote
- Participants generate commit/abort record
- Participants flush commit/abort record
- **Participants respond with Ack**
- Coordinator generates end record
- Coordinator flushes end record



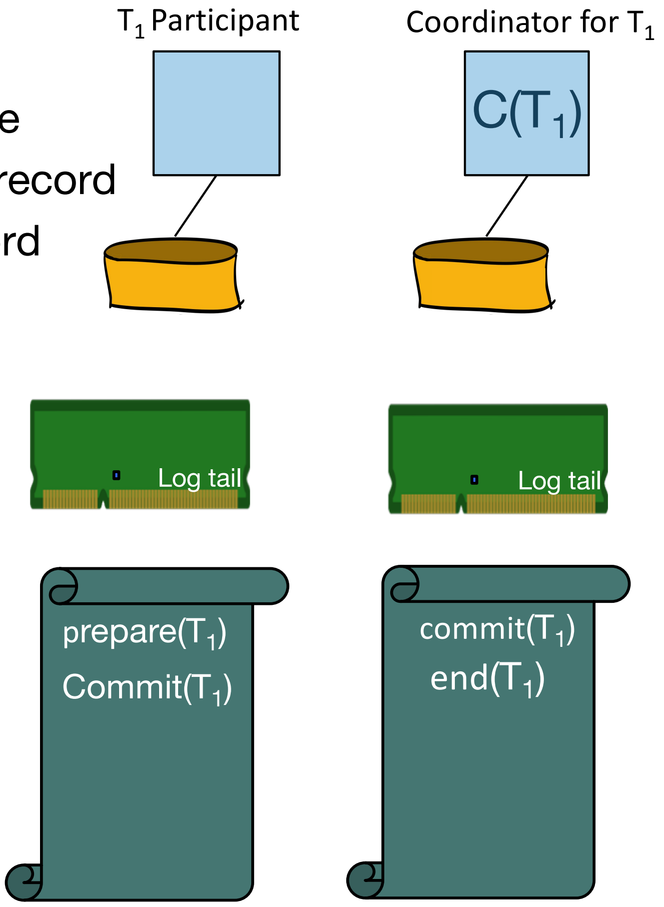
One More Time, With Logging, Part 16

- **Phase 2:**
- Coordinator broadcasts result of vote
- Participants generate commit/abort record
- Participants flush commit/abort record
- Participants respond with Ack
- **Coordinator generates end record**
- Coordinator flushes end record

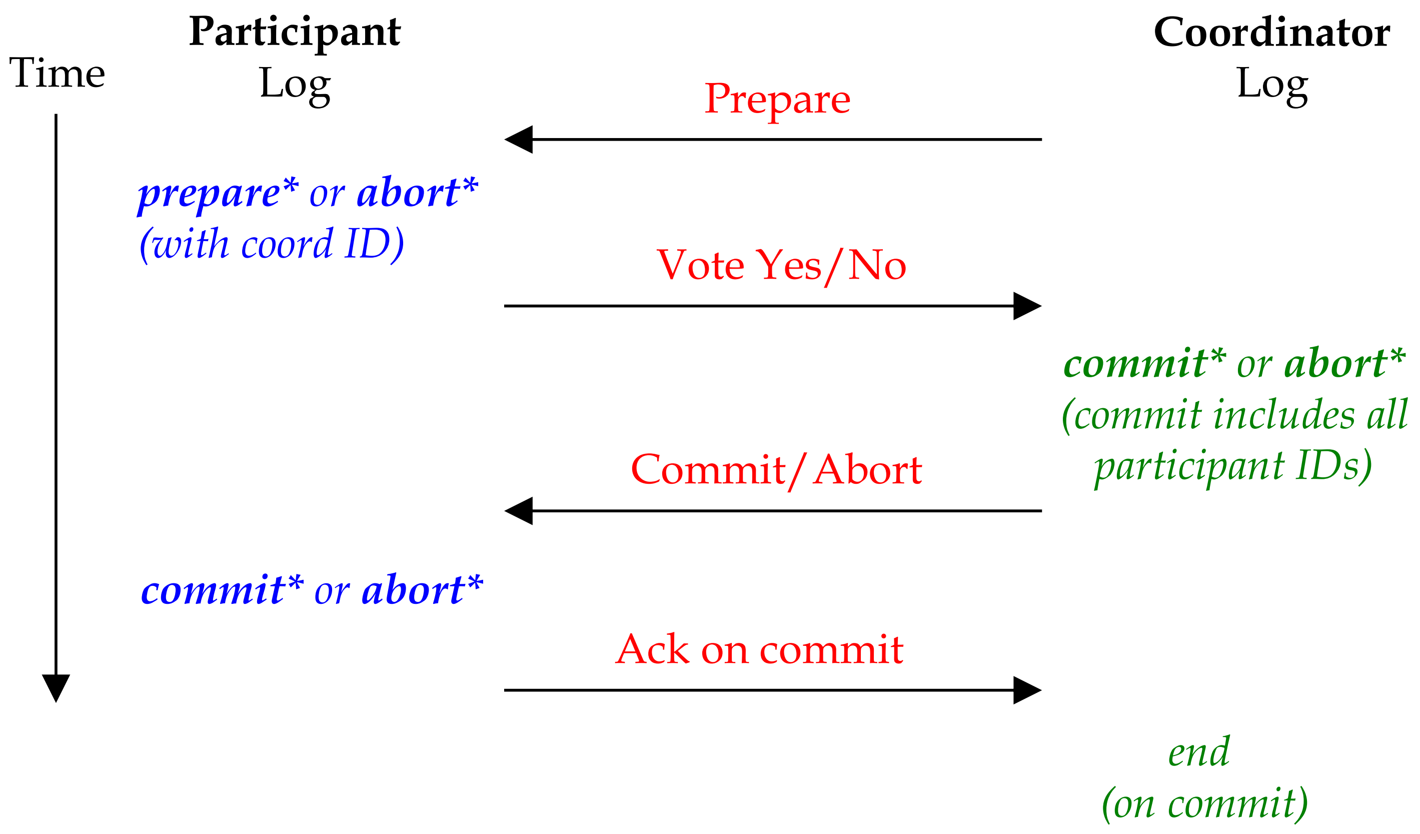


One More Time, With Logging, Part 17

- **Phase 2:**
- Coordinator broadcasts result of vote
- Participants generate commit/abort record
- Participants flush commit/abort record
- Participants respond with Ack
- Coordinator generates end record
- **Coordinator flushes end record**



2PC In a Nutshell


Time
Participant
Log
Coordinator
Log
Prepare
prepare* or ***abort****
(with coord ID)
Vote Yes/No
commit* or ***abort****
(commit includes all participant IDs)
Commit/Abort
commit* or ***abort****
Ack on commit
end
(on commit)

NOTE

asterisk*: wait for log flush before sending next msg

RECOVERY AND 2PC

Failure Handling

- Assume everybody recovers eventually
    - Big assumption!
    - Depends on WAL (and short downtimes)
- Coordinator notices a Participant is down?
    - If participant hasn't voted yet, coordinator aborts transaction
    - If waiting for a commit Ack, hand to "recovery process"
- Participant notices Coordinator is down?
    - If it hasn't yet logged prepare, then abort unilaterally
    - If it has logged prepare, hand to "recovery process"
- Note
    - Thinking a node is "down" may be incorrect!

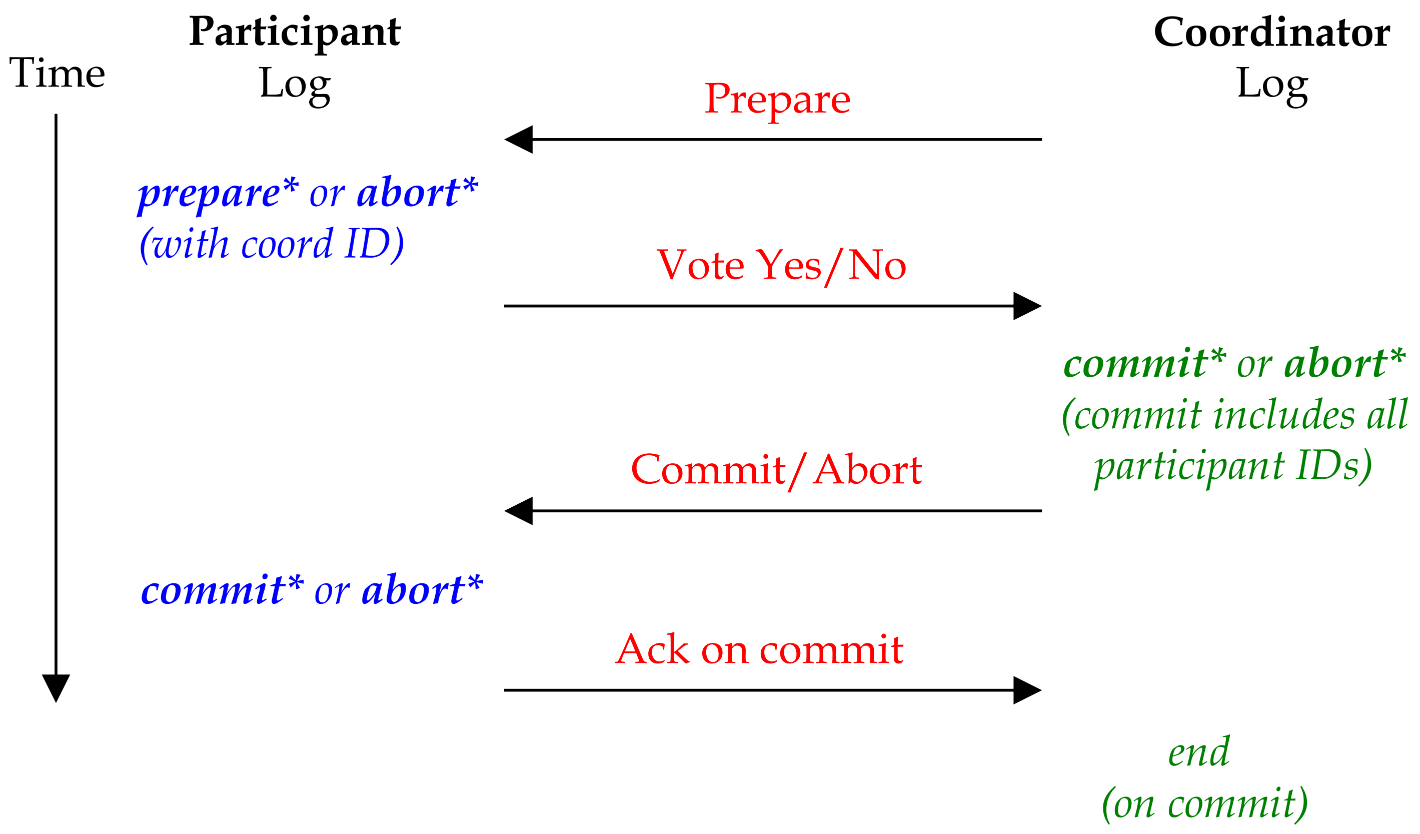
Integration with ARIES Recovery

- On recovery
  - Assume there’s a “Recovery Process” at each node
  - It will be given tasks to do by the Analysis phase of ARIES
  - These tasks can run in the background (asynchronously)
- Note: multiple roles on a single node
  - Coordinator for some xacts, Participant for others

How Does Recovery Process Work?

- Coordinator recovery process gets inquiry from a “prepared” participant
  - If transaction table at coordinator says aborting/committing
    - send appropriate response and continue protocol on both sides
  - If transaction table at coordinator says nothing: send ABORT
    - Only happens if coordinator had also crashed before writing commit/abort
    - Inquirer does the abort on its end

2PC In a Nutshell

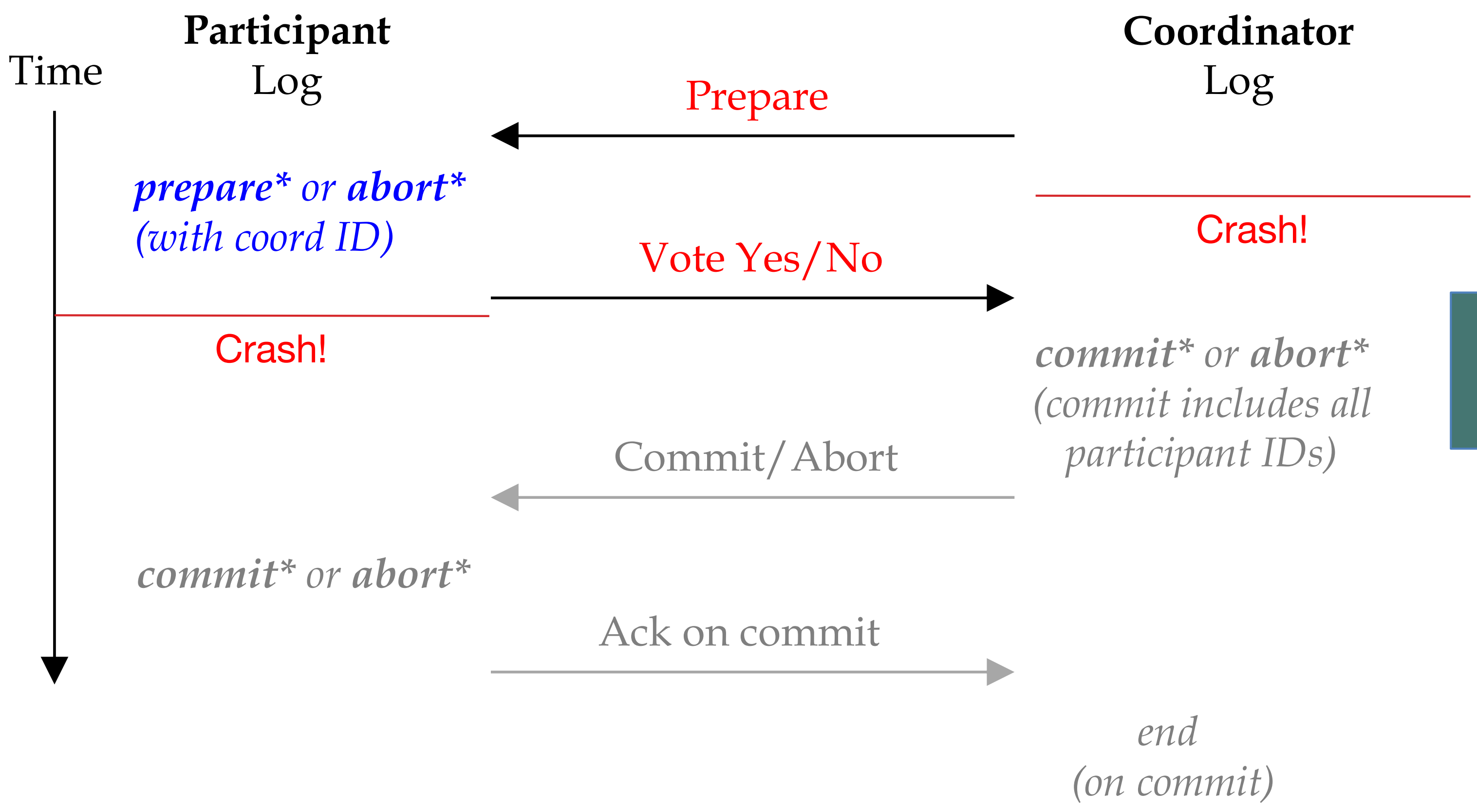


Time
Participant
Log
Coordinator
Log
Prepare
prepare* or ***abort****
(with coord ID)
Vote Yes/No
commit* or ***abort****
(commit includes all participant IDs)
Commit/Abort
commit* or ***abort****
Ack on commit
end
(on commit)

NOTE

asterisk*: wait for log flush before sending next msg

2PC In a Nutshell



NOTE

asterisk*: wait for log flush before sending next msg

Recovery: Think it through

- What happens when coordinator recovers?
  - With “commit” and “end”? **Nothing**
  - With just “commit”? **Rerun Phase 2!**
  - With “abort”? **Nothing (Presumed Abort)**
- What happens when participant recovers:
  - With no prepare/commit/abort? **Nothing (Presumed Abort)**
  - With “prepare” & “commit”? **Send Ack to coordinator.**
  - With just “prepare”? **Send inquiry to Coordinator**
  - With “abort”? **Nothing (Presumed Abort)**

Commit iff coordinator logged a commit

2PC + 2PL

- Ensure point-to-point messages are densely ordered
  - 1,2,3,4,5…
  - Dense per (sender/receiver/XID)
  - Receiver can detect anything missing or out-of-order
  - Receiver buffers message k+1 until [1..k] received
- Commit:
  - When a participant processes Commit request, it has all the locks it needs
  - Flush log records and drop locks atomically
- Abort:
  - Its safe to abort autonomously, locally: no cascade.
  - Log appropriately to 2PC (presumed abort in our case)
  - Perform local Undo, drop locks atomically

Availability Concerns

- What happens while a node is down?
  - Other nodes may be in limbo, holding locks
  - So certain data is unavailable
  - This may be bad...
- Dead Participants? Respawned by coordinator
  - Recover from log
  - And if the old participant comes back from the dead, just ignore it and tell it to recycle itself
- Dead Coordinator?
  - This is a problem!
  - 3-Phase Commit was an early attempt to solve it
  - Paxos Commit provides a more comprehensive solution
    - Gray+Lamport paper! Out of scope for this class.

Summing Up

- Distributed Databases
  - A central aspect of Distributed Systems
- Partitioning provides Scale-Up
- Can also partition lock tables and logs
- But need to do some global coordination:
  - Deadlock detection: easy
  - Commit: trickier
- Two-phase commit is a classic distributed consensus protocol
  - Logging/recovery aspects unique:
    - many distributed protocols gloss over
  - But 2PC is unavailable on any single failure
  - This is bad news for scale-up,
    - because odds of failure go up with #machines
  - Paxos Commit (Gray+Lamport) addresses that problem