Distributed Transactions with Two-Phase Commit

Alvin Cheung

Aditya Parameswaran

R&G - Chapter 22

Distributed vs. Parallel?

- Earlier we discussed Parallel DBMSs
  - Shared-memory
  - Shared-disk
  - Shared-nothing
- Distributed is basically shared-nothing parallel
  - Perhaps with a slower network
    - Possibly thanks to being geographically distributed

What's Special About Distributed Computing?

- Inherited from shared-nothing parallel computation
  - Parallel computation
  - No shared memory/disk
- Unreliable Networks
  - Delay, reordering, loss of packets
- Unsynchronized clocks
  - Impossible to have perfect synchrony
- Partial failure: can't know what's up, what's down

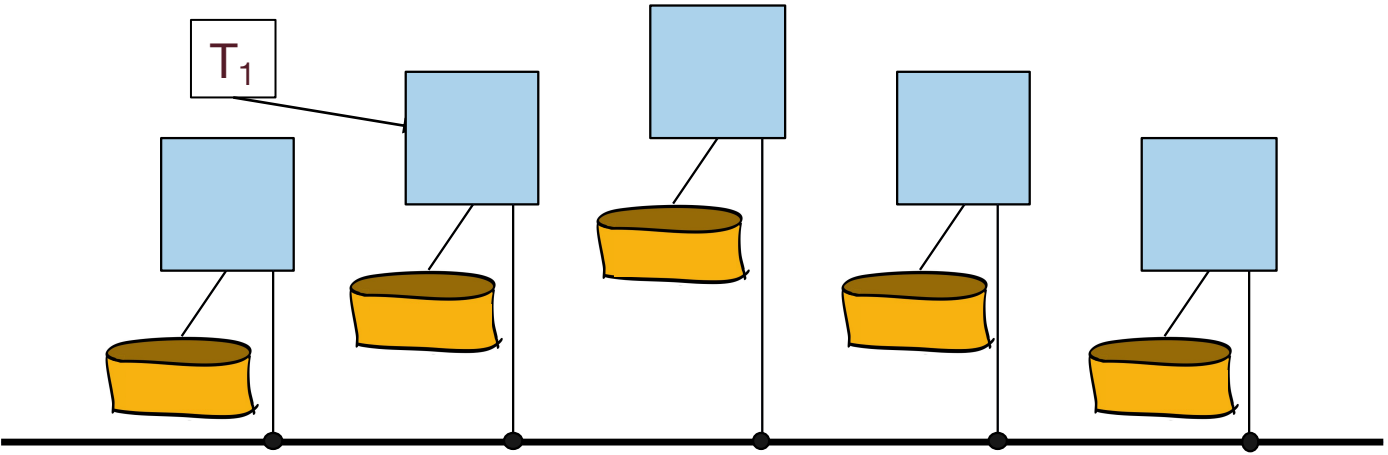
Distributed Database Systems

- DBMS an influential special case of distributed computing
  - The trickiest part of distributed computing is state, i.e. Data
  - Transactions provide an influential model for concurrency/parallelism
  - DBMSs worried about fault handling early on
- Special-case because not all distributed programs are written transactionally
  - And if not, database techniques may not apply
- Many of today's most complex distributed systems are databases
  - Cloud SQL databases like Google Spanner, AWS Aurora, Azure SQL
  - NoSQL databases like DynamoDB, Cassandra, MongoDB, Couchbase…
- We'll focus on transactional concurrency control and recovery
  - You already know many lessons of distributed query processing

DISTRIBUTED LOCKING

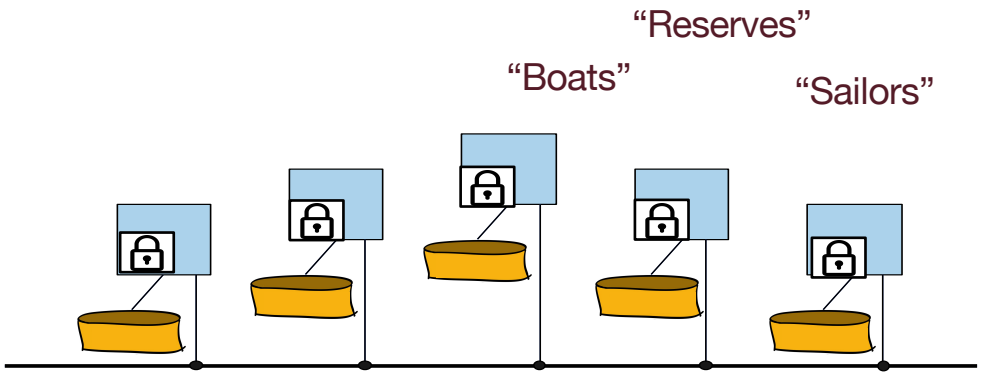
Distributed Concurrency Control

- Consider a shared-nothing distributed DBMS
- For today, assume partitioning but no replication of data
- Each transaction arrives at some node:
  - The “coordinator” for the transaction
    - Can be designated or assigned on the fly



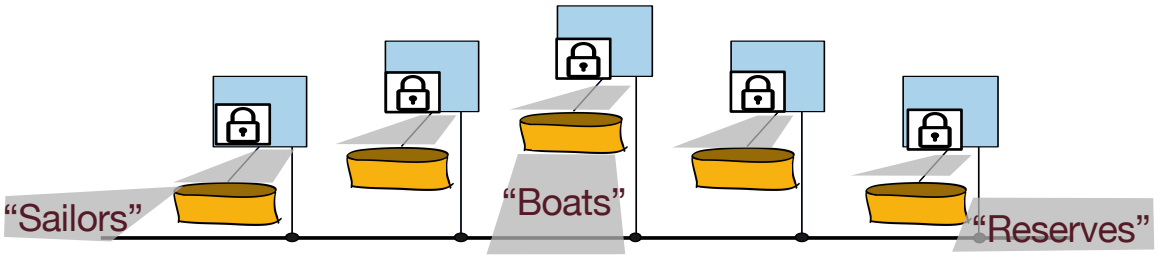
Where is the Lock Table

- Typical design: Locks partitioned with the data
  - Independent: each node manages “its own” lock table
  - Works for objects that fit on one node (pages, tuples)
- For coarser-grained locks, assign a “home” node
  - Object being locked (table, DB) exists across nodes



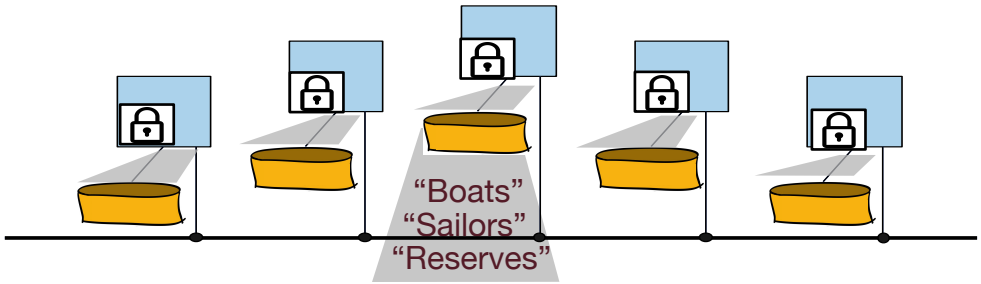
Where is the Lock Table, Pt 2

- Typical design: Locks partitioned with the data
  - Independent: each node manages “its own” lock table
  - Works for objects that fit on one node (pages, tuples)
- For coarser-grained locks, assign a “home” node
  - Object being locked (table, DB) exists across nodes
  - These coarse-grained locks can be partitioned across nodes



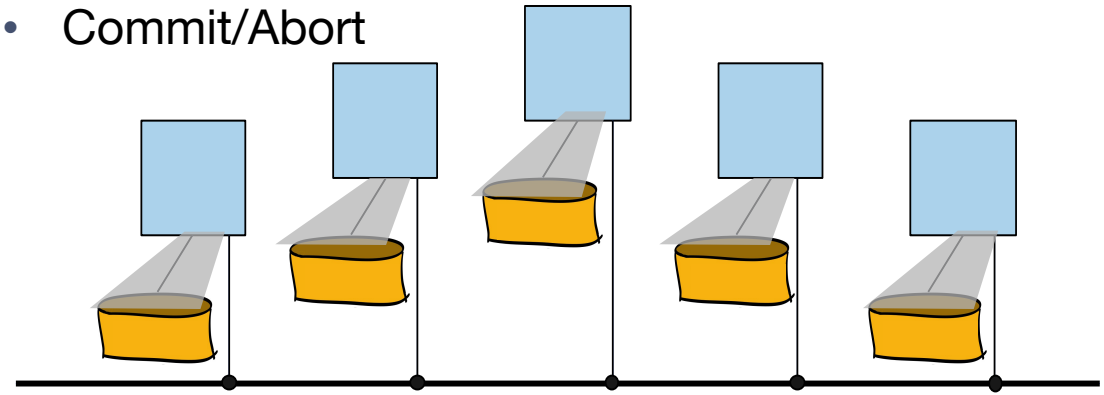
Where is the Lock Table, Pt 3

- Typical design: Locks partitioned with the data
  - Independent: each node manages “its own” lock table
  - Works for objects that fit on one node (pages, tuples)
- For coarser-grained locks, assign a “home” node
  - Object being locked (table, DB) exists across nodes
  - These coarse-grained locks can be partitioned across nodes
  - Or centralized at a master node



Ignore global coarse-grained locks for a moment…

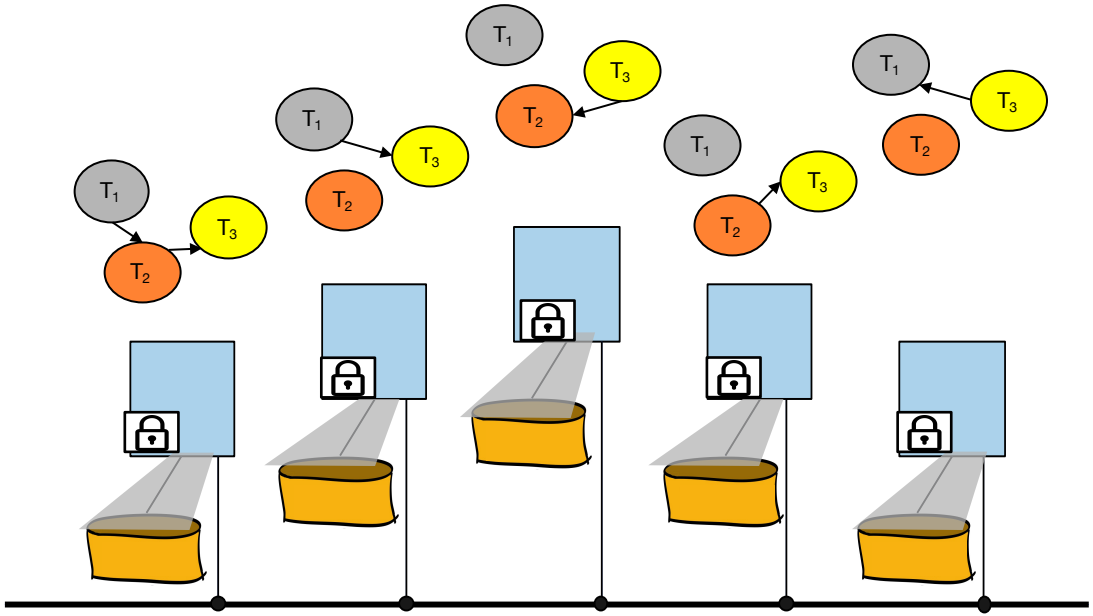
- Every node does its own locking
  - Clean and efficient
  - Nicely generalizes the single-node setting
- “Global” issues remain:
  - Deadlock
  - Commit/Abort



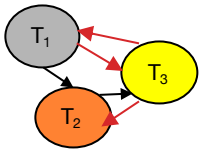
DISTRIBUTED DEADLOCK DETECTION

What Could Go Wrong? #1

- Deadlock detection via waits—for graphs

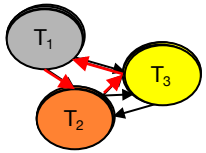


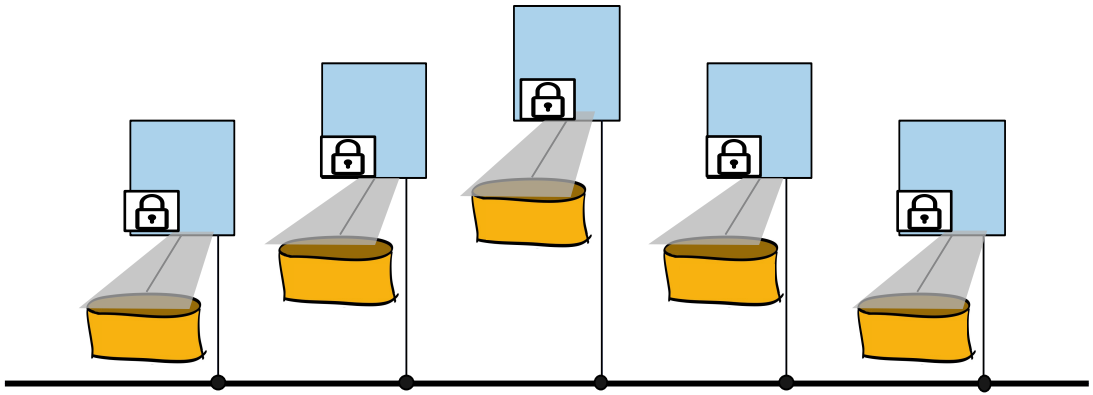
Each machine doesn't have a cycle, but there is a global cycle



What Could Go Wrong? #1 Part 2

- Deadlock detection via waits—for graphs
  - Easy fix: periodically union at designated coordinator

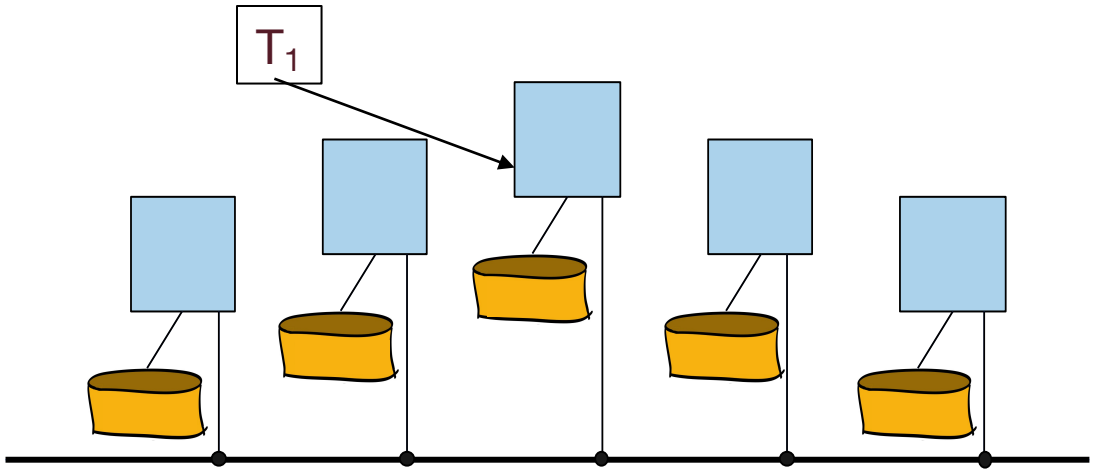




DISTRIBUTED COMMIT: 2PC

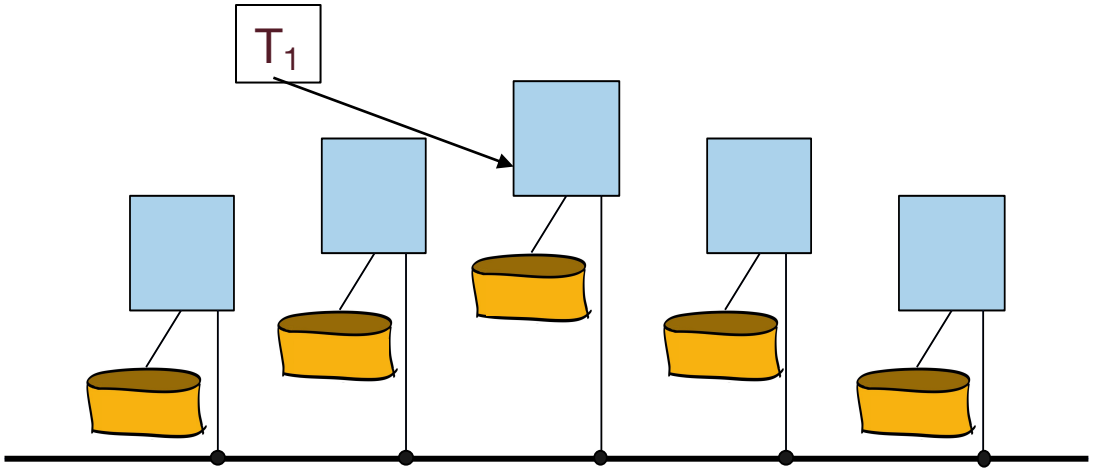
Strawman: Coordinator makes Decision

- Recall that every txn has a coordinator node
- Coordinator decides if the txn is going to commit or abort.
- Lets all the other nodes know.
- Q: Why is this scheme problematic?



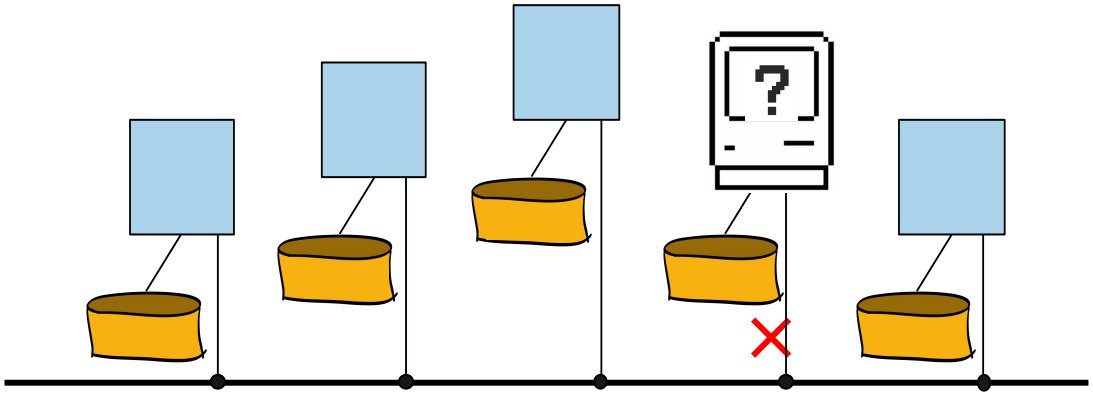
Strawman: Coordinator makes Decision

- Recall that every txn has a coordinator node
- Coordinator decides if the txn is going to commit or abort.
- Lets all the other nodes know.
- Q: Why is this scheme problematic?
  - Among other things, one of the nodes may want to abort, even if the coordinator wants to commit
  - Some nodes may actually be down (so any txn touching their data shouldn't proceed)



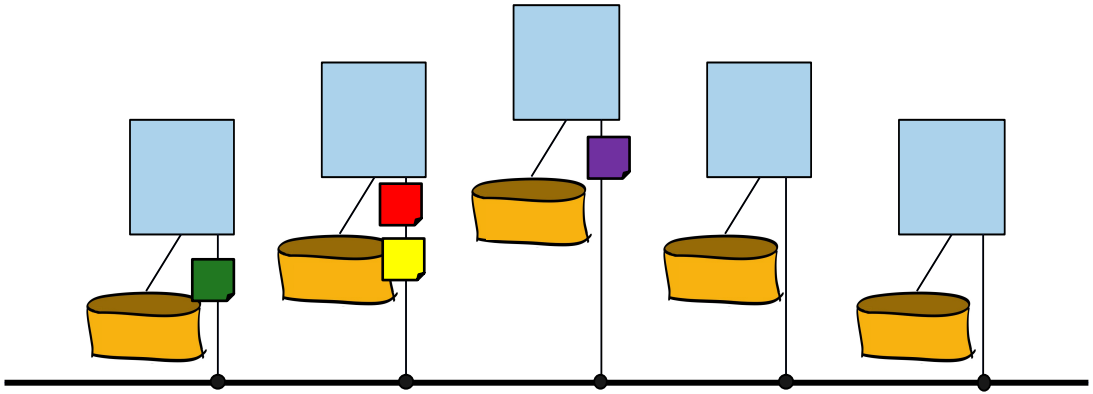
In General, What Could Go Wrong? #2

- Failures/Delays: Nodes
  - Commit? Abort?
    - If we haven't heard from a node, we don't know if is alive or dead.
    - The decision may hinge on this node (imagine a FK violation at that node)
  - When the node comes back, how does it recover in a world that moved forward?



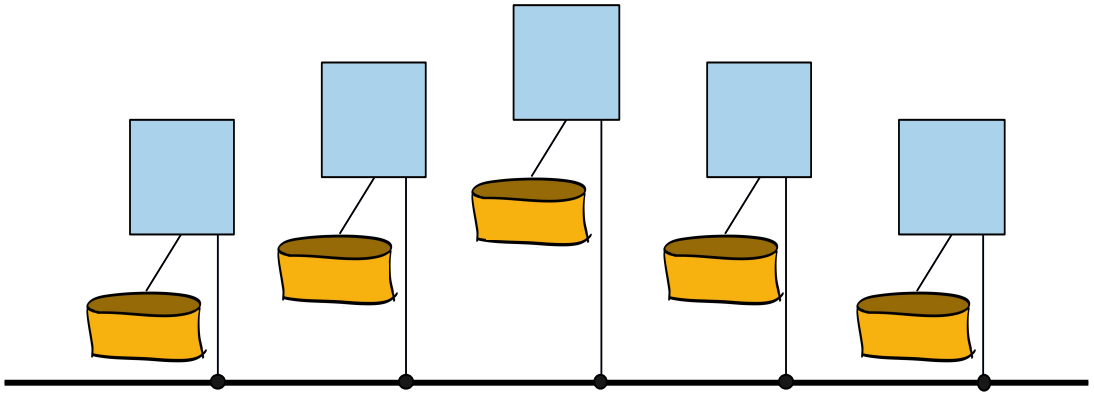
What Could Go Wrong? #2, Part 2

- Failures/Delays: Nodes
- Failures/Delays: Messages
  - Non-deterministic reordering per channel, interleaving across channels
  - “Lost” (very delayed) messages
    - How long should we wait for this?



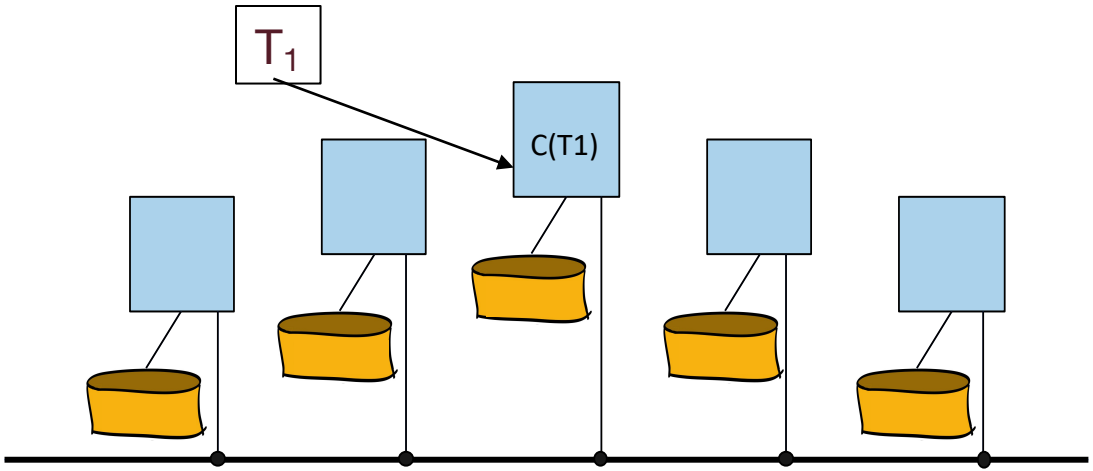
What Could Go Wrong? #2, Part 3

- Failures/Delays: Nodes
- Failures/Delays: Messages
  - Non-deterministic reordering per channel, interleaving across channels
  - “Lost” (very delayed) messages
- Given this, how do all nodes agree on Commit vs. Abort?



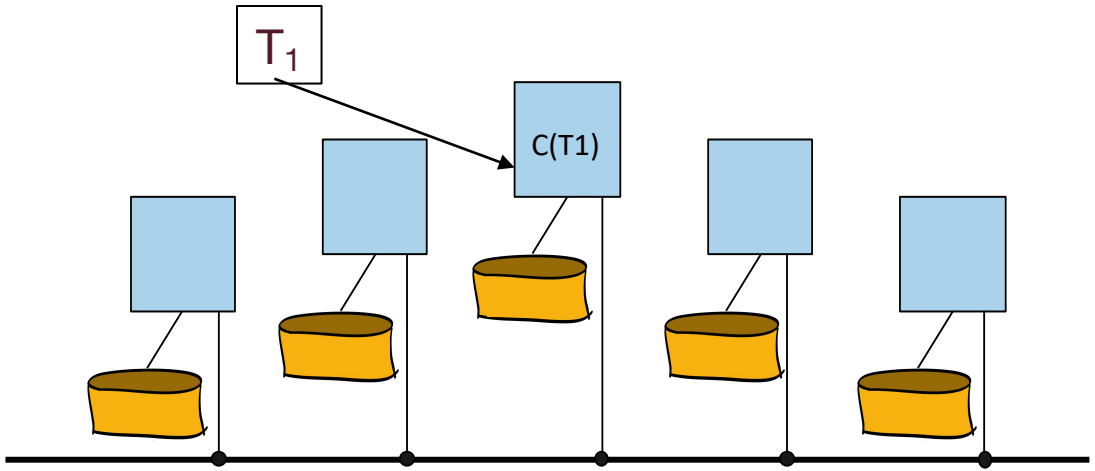
Basic Idea: Distributed Voting

- Vote for Commitment
  - How many votes does a commit need to win?
  - Any single node could observe a problem (e.g. deadlock, constraint violation)
  - Hence must be unanimous.



Distributed voting? How?

- How do we implement distributed voting?!
  - In the face of message/node failure/delay?

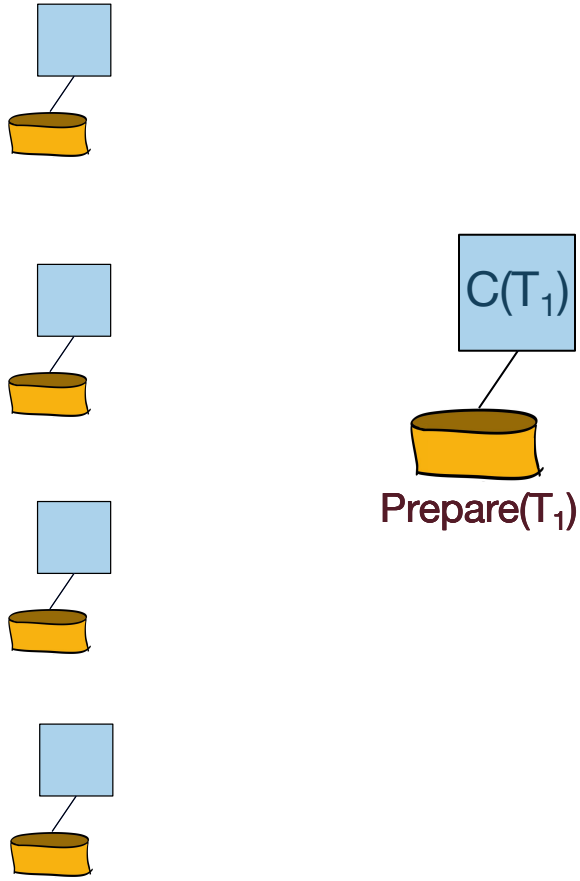


2-Phase Commit

- A.k.a. 2PC. (Not to be confused with 2PL!)
- Like a wedding ceremony!
- **Phase 1: “do you take this person...”**
  - Coordinator tells participants to “prepare”
  - Participants respond with yes/no votes
    - Unanimity required for yes!
- **Phase 2: “I now pronounce you...”**
  - Coordinator disseminates result of the vote
- Need to do some logging for failure handling....

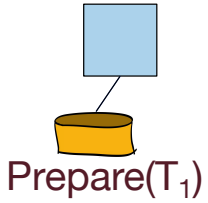
2-Phase Commit, Part 1

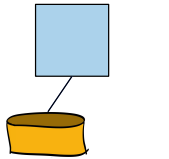
- **Phase 1:**
  - **Coordinator tells participants to “prepare”**
  - Participants respond with yes/no votes
    - Unanimity required for commit!
- Phase 2:
  - Coordinator disseminates result of the vote
  - Participants respond with Ack

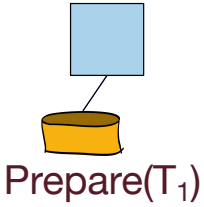


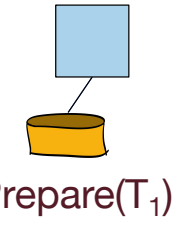
2-Phase Commit, Part 2

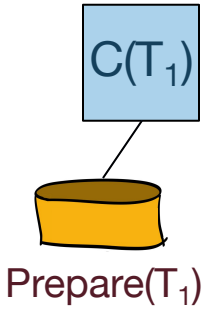
- **Phase 1:**
  - **Coordinator tells participants to “prepare”**
  - Participants respond with yes/no votes
    - Unanimity required for commit!
- Phase 2:
  - Coordinator disseminates result of the vote
  - Participants respond with Ack





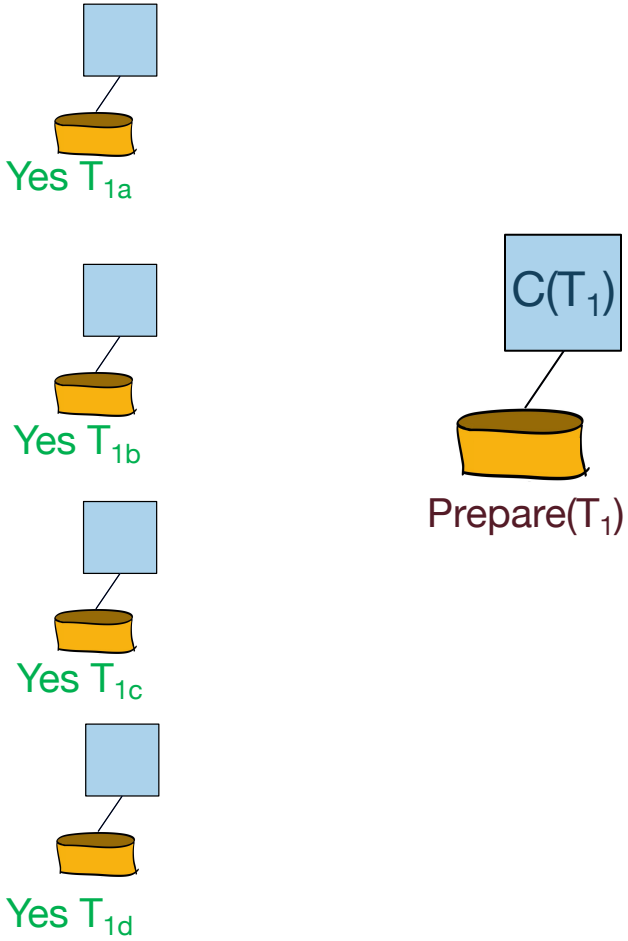






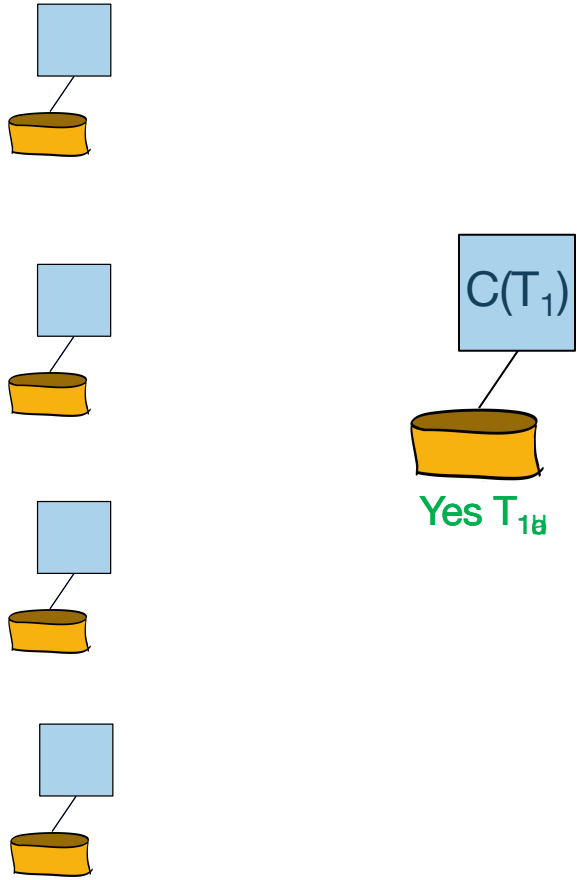
2-Phase Commit, Part 3

- **Phase 1:**
  - Coordinator tells participants to “prepare”
  - **Participants respond with yes/no votes**
    - **Unanimity required for commit!**
- Phase 2:
  - Coordinator disseminates result of the vote
  - Participants respond with Ack



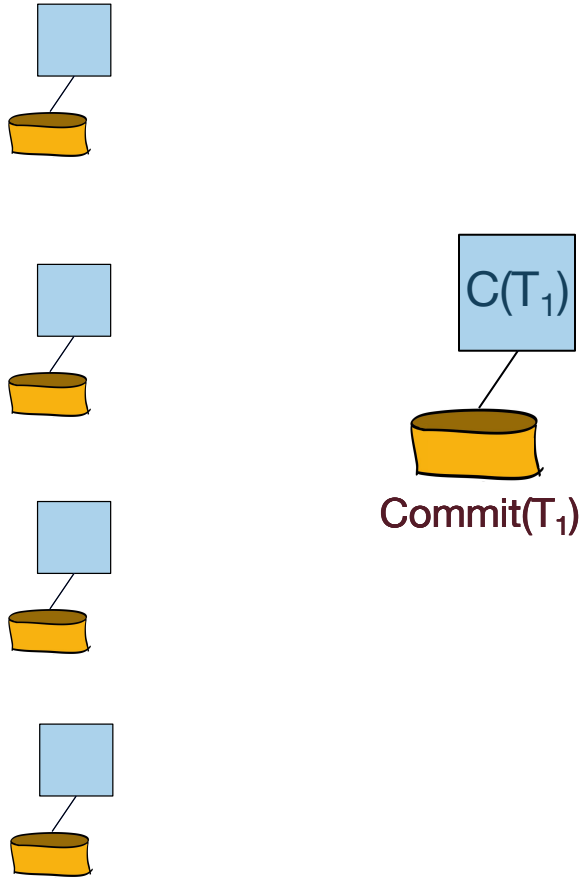
2-Phase Commit, Part 4

- **Phase 1:**
  - Coordinator tells participants to “prepare”
  - **Participants respond with yes/no votes**
    - **Unanimity required for commit!**
- Phase 2:
  - Coordinator disseminates result of the vote
  - Participants respond with Ack



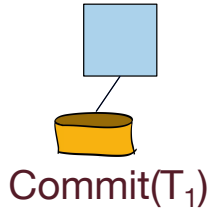
2-Phase Commit, Part 5

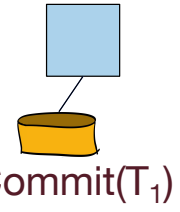
- Phase 1:
  - Coordinator tells participants to “prepare”
  - Participants respond with yes/no votes
    - Unanimity required for commit!
- **Phase 2:**
  - **Coordinator disseminates result of the vote**
  - Participants respond with Ack

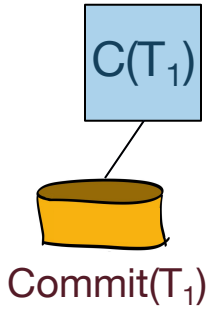


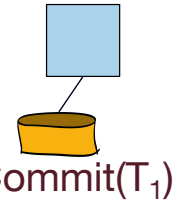
2-Phase Commit, Part 6

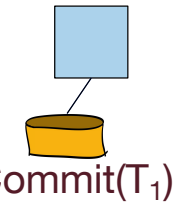
- Phase 1:
  - Coordinator tells participants to “prepare”
  - Participants respond with yes/no votes
    - Unanimity required for commit!
- **Phase 2:**
  - **Coordinator disseminates result of the vote**
  - Participants respond with Ack





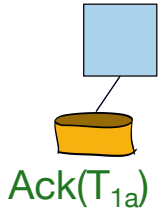


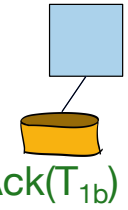


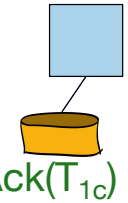


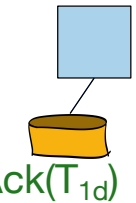
2-Phase Commit, Part 7

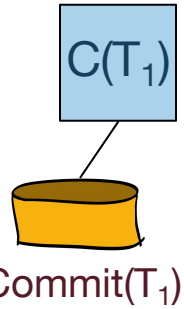
- Phase 1:
  - Coordinator tells participants to “prepare”
  - Participants respond with yes/no votes
    - Unanimity required for commit!
- **Phase 2:**
  - Coordinator disseminates result of the vote
  - **Participants respond with Ack**





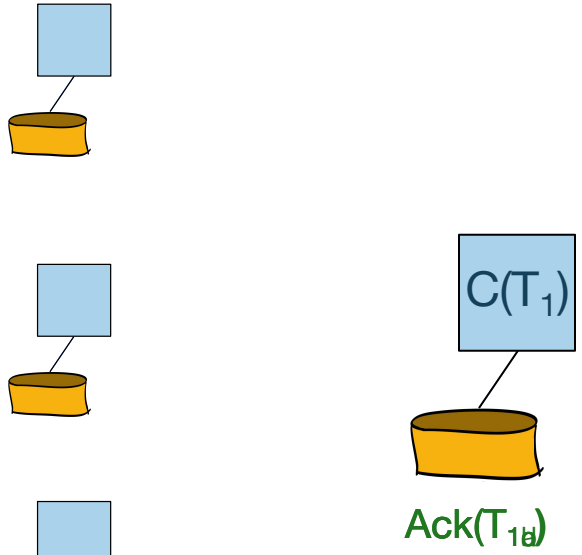






2-Phase Commit, Part 8

- Phase 1:
  - Coordinator tells participants to “prepare”
  - Participants respond with yes/no votes
    - Unanimity required for commit!
- **Phase 2:**
  - Coordinator disseminates result of the vote
  - **Participants respond with Ack**



When the coordinator receives messages from all participants, txn is complete