Practical Lab

Cloud Systems Engineering

(cloud-lab)

https://dse.in.tum.de/

Task #4:
Distributed Transactions (Txs)

Learning Goals

In this week's task you will:

- Learn about single-node and distributed transactions
- Learn about the two-phase commit
- Build your own end-to-end distributed transactional KVs

Background

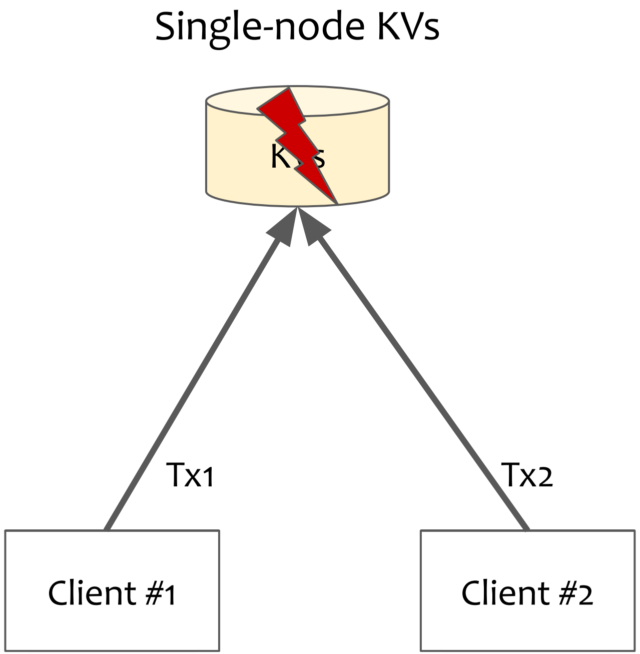
What about transactions?

A set of operations that process multiple data in an atomic manner

<u>Properties:</u>

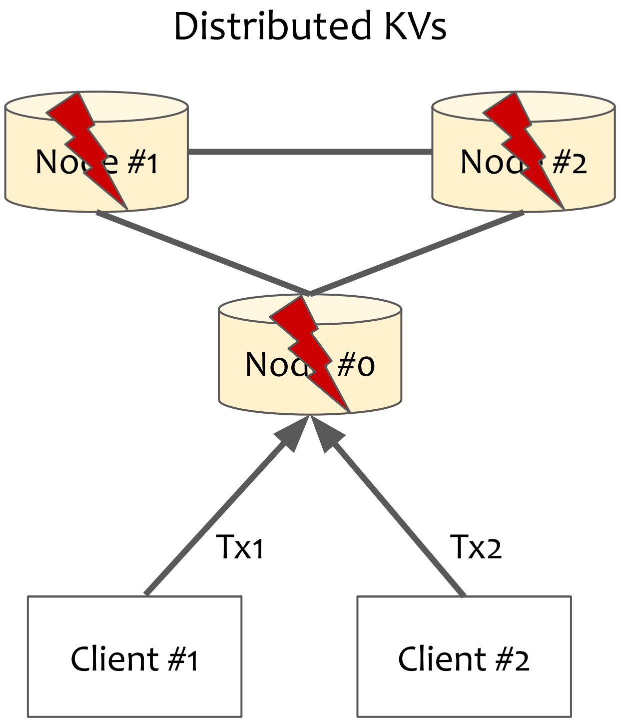
- Programmability
  - process massive datasets in a serializable manner
- Transparency & unlimited resources
  - Access to files located on different machines
  - The view of a single giant machine
- Data durability
  - Updates survive after a node fails

Single-node Transactions



- ACID properties:
  - the "all-or-nothing" property (**A**tomicity)
  - correctness/serializability (**C**onsistency + **I**solation)
  - fault-tolerance (**D**urability)

Distributed Transactions



- ACID properties in distributed settings
- Two (or more) machines agree to either do something or not do something
  - If no node fails and all nodes are ready to commit, then all nodes COMMIT
  - Otherwise ABORT at all nodes

Challenge

How to offer ACID properties for single-node and distributed transactions?

Single-node Txs:

- Concurrency control
    - resolves data races
    - serializases operations
- Logging mechanism
    - durability & recovery

Distributed Txs:

- (distributed) Concurrency control
- (distributed) Logging mechanism
- Distributed atomic commit protocol
    - ensures all involved nodes will agree even nodes fail independently from each other

Concurrency control

- Pessimistic concurrency control (Two-phase locking (2PL))
  - Operations take locks as they go along
  - Locks are released after commit/abort
- Optimistic concurrency control (Versioning protocol)
  - Operations are executed as if they didn't have competitors
  - Validation takes place at the commit time
- Timestamp ordering concurrency control
  - Operations are ordered based on a "clock" (e.g., global walltime clock, client timestamps, etc.)

Two-phase Commit (2PC) (1/2)

- A distributed protocol of two phases
  - Prepare phase: ensures all or nothing property
  - Commit phase: updates are made to the database

- Extra tool: persistent log*
  - If a machine fails, remember what happened

*Assume that the log is not corrupted

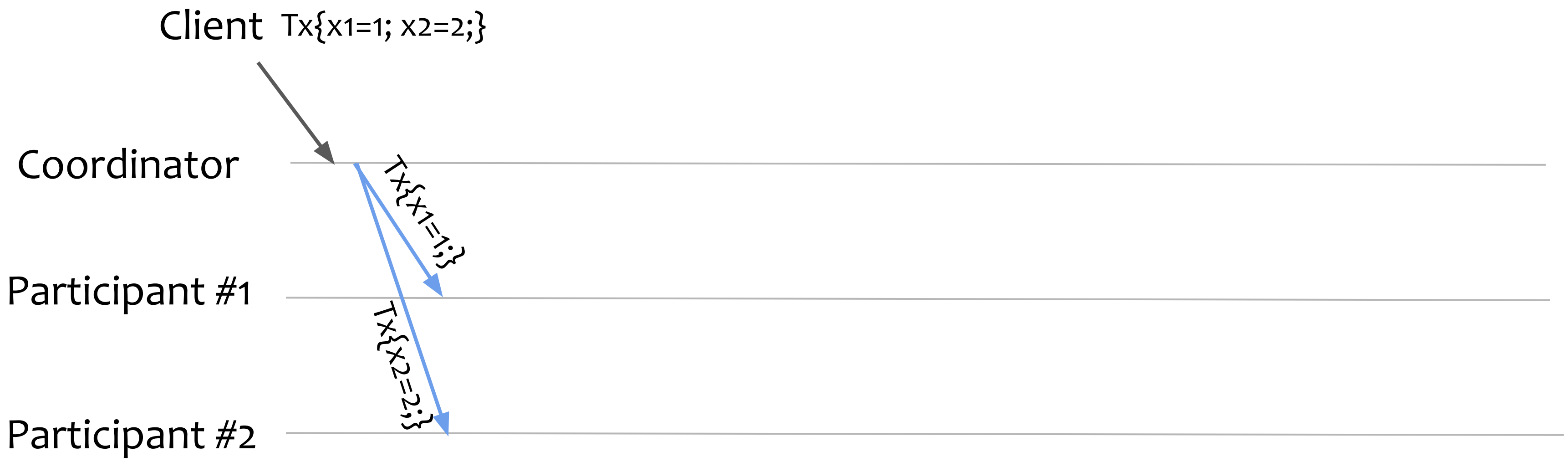
Two-phase commit with two-phase locking

Coordinator

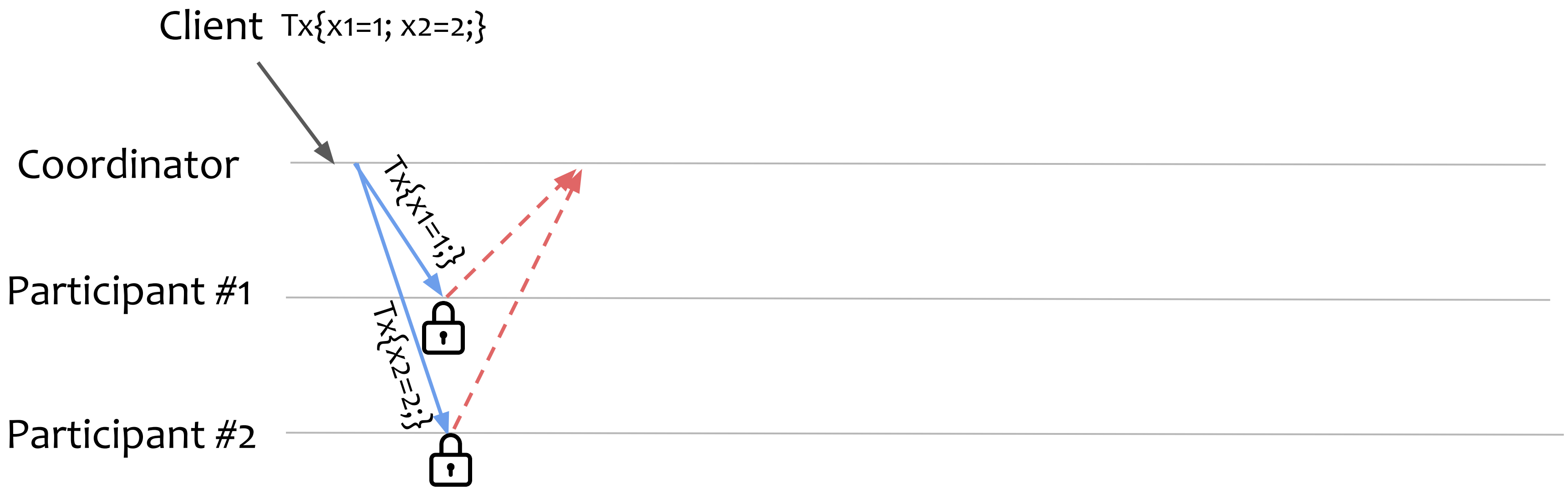
Participant #1

Participant #2

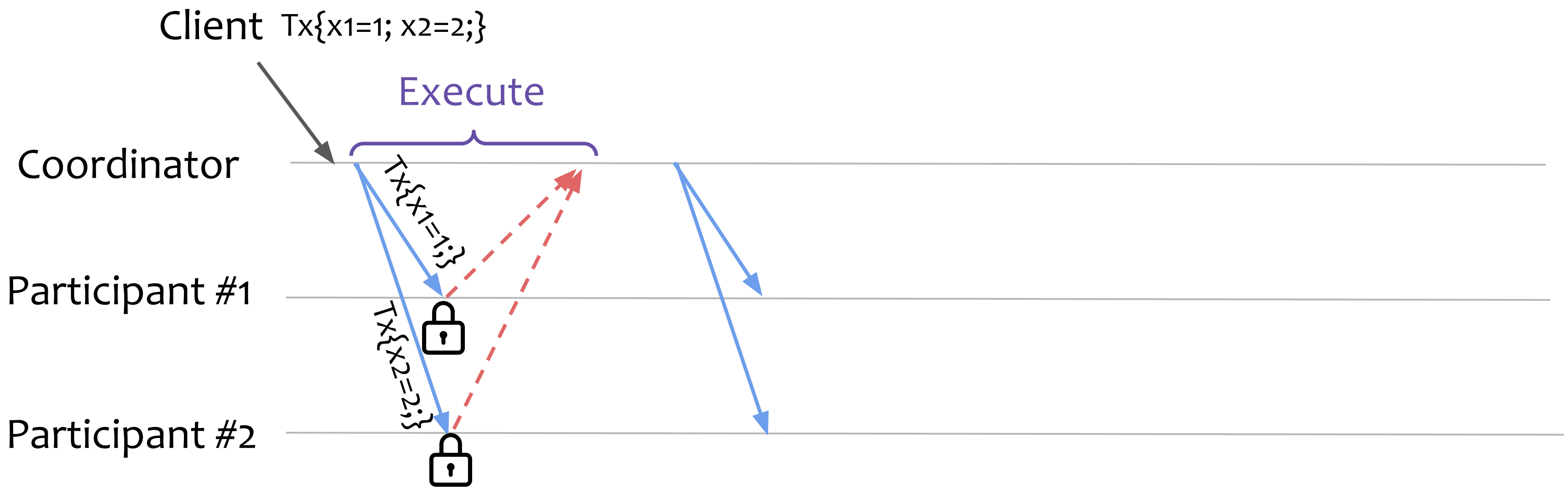
Two-phase commit with two-phase locking



Two-phase commit with two-phase locking

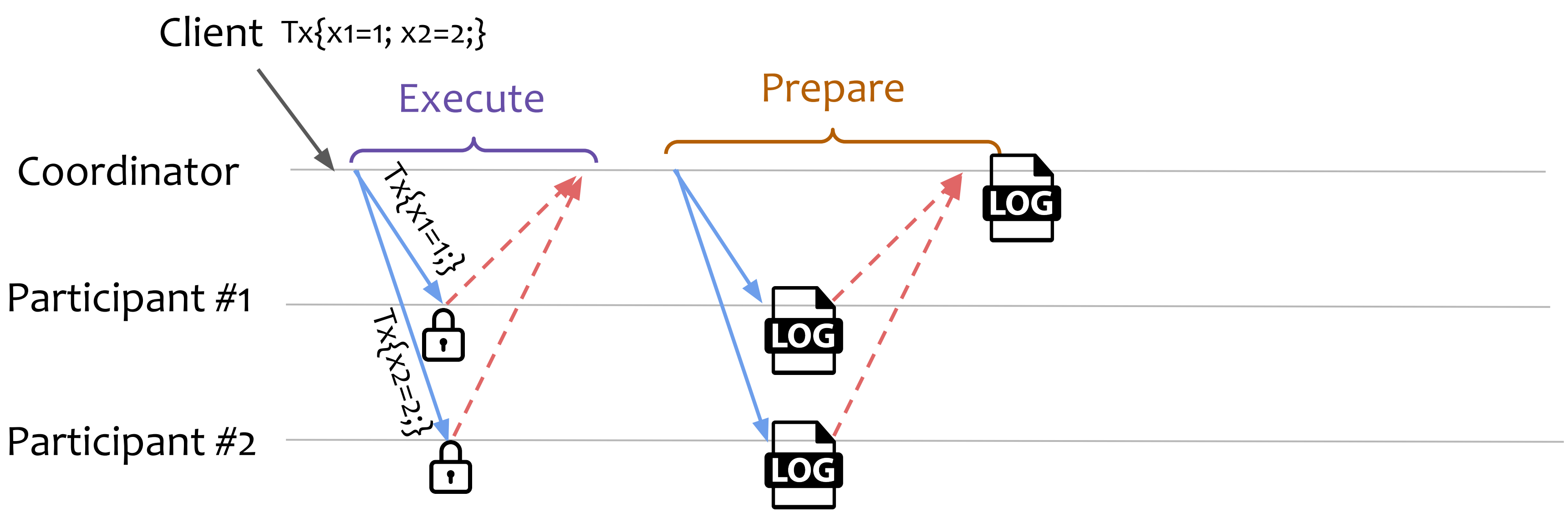


Two-phase commit with two-phase locking

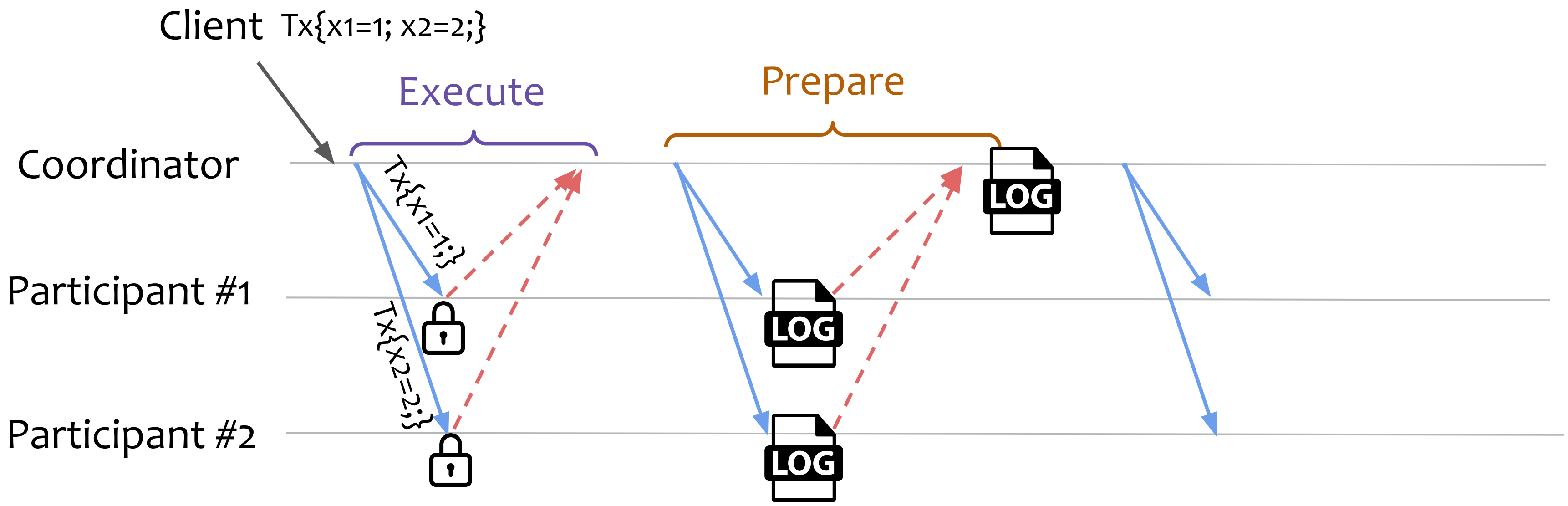


Two-phase commit with two-phase locking

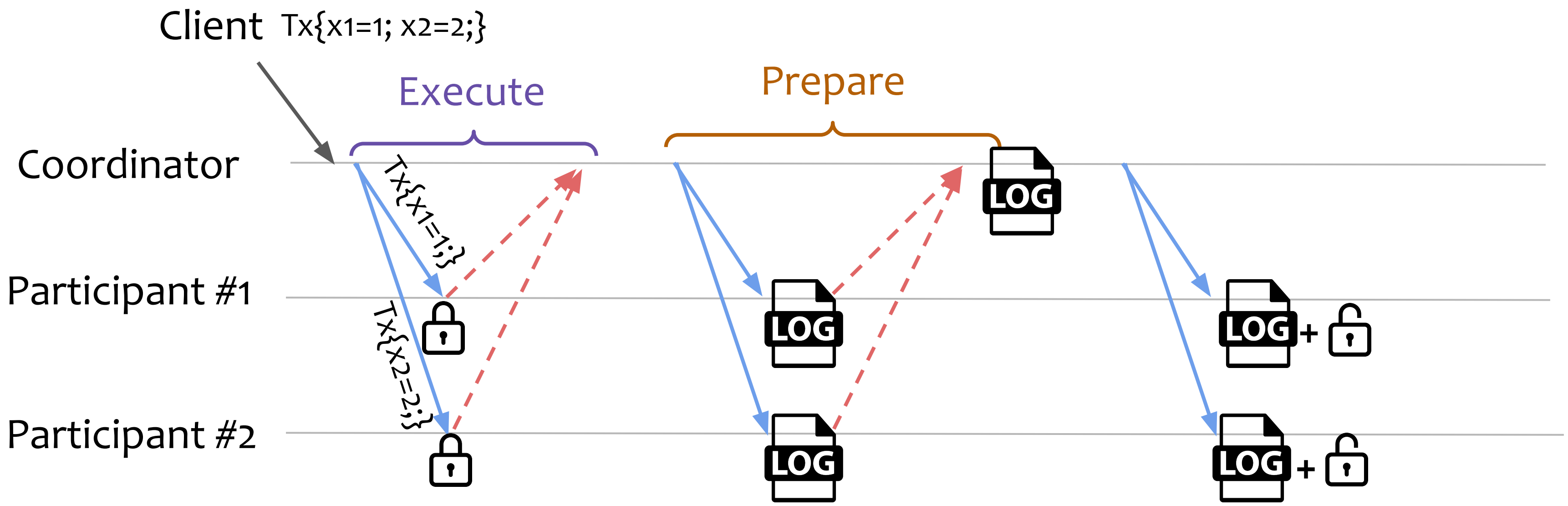
Two-phase commit with two-phase locking



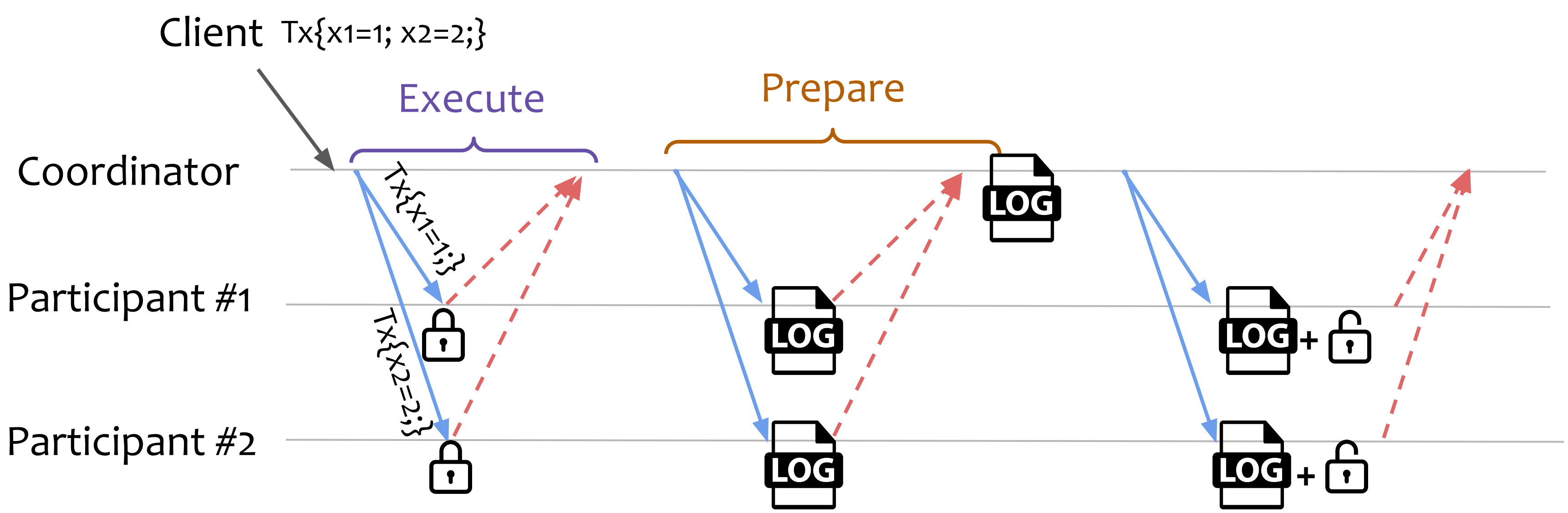
Two-phase commit with two-phase locking



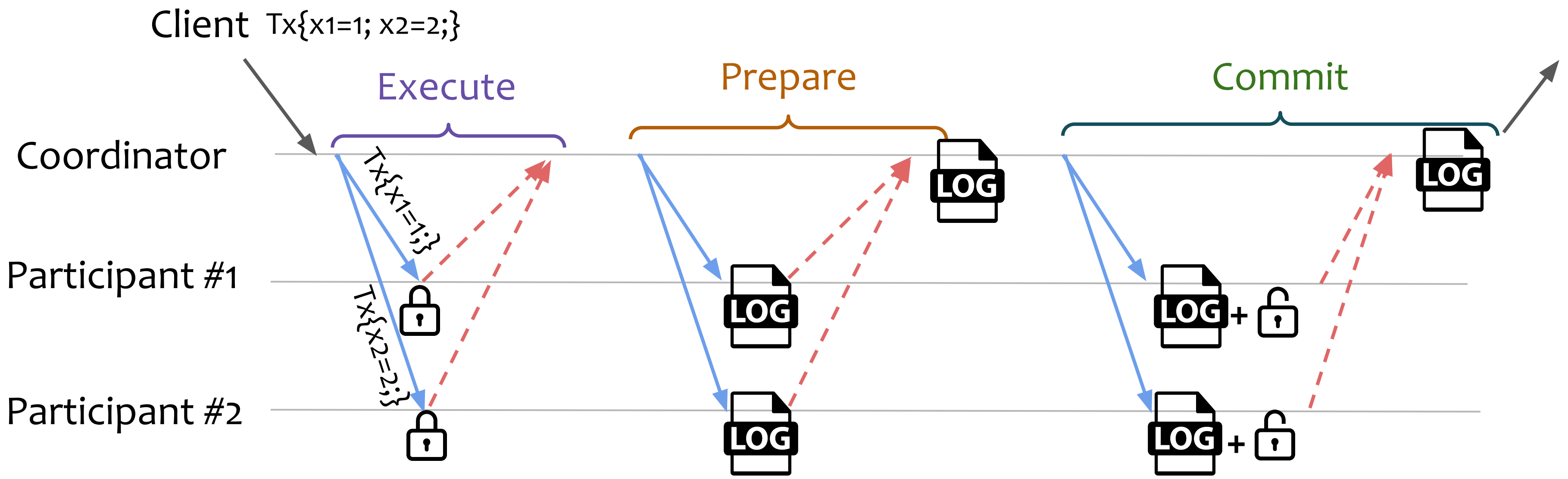
Two-phase commit with two-phase locking



Two-phase commit with two-phase locking



Two-phase commit with two-phase locking



Failures

Participant failures:

- (might) Block the protocol for a specific transaction
- If the transaction is not prepared the coordinator might abort it after a timeout

Coordinator failures:

- Execution blocks until the coordinator recovers

Recovery

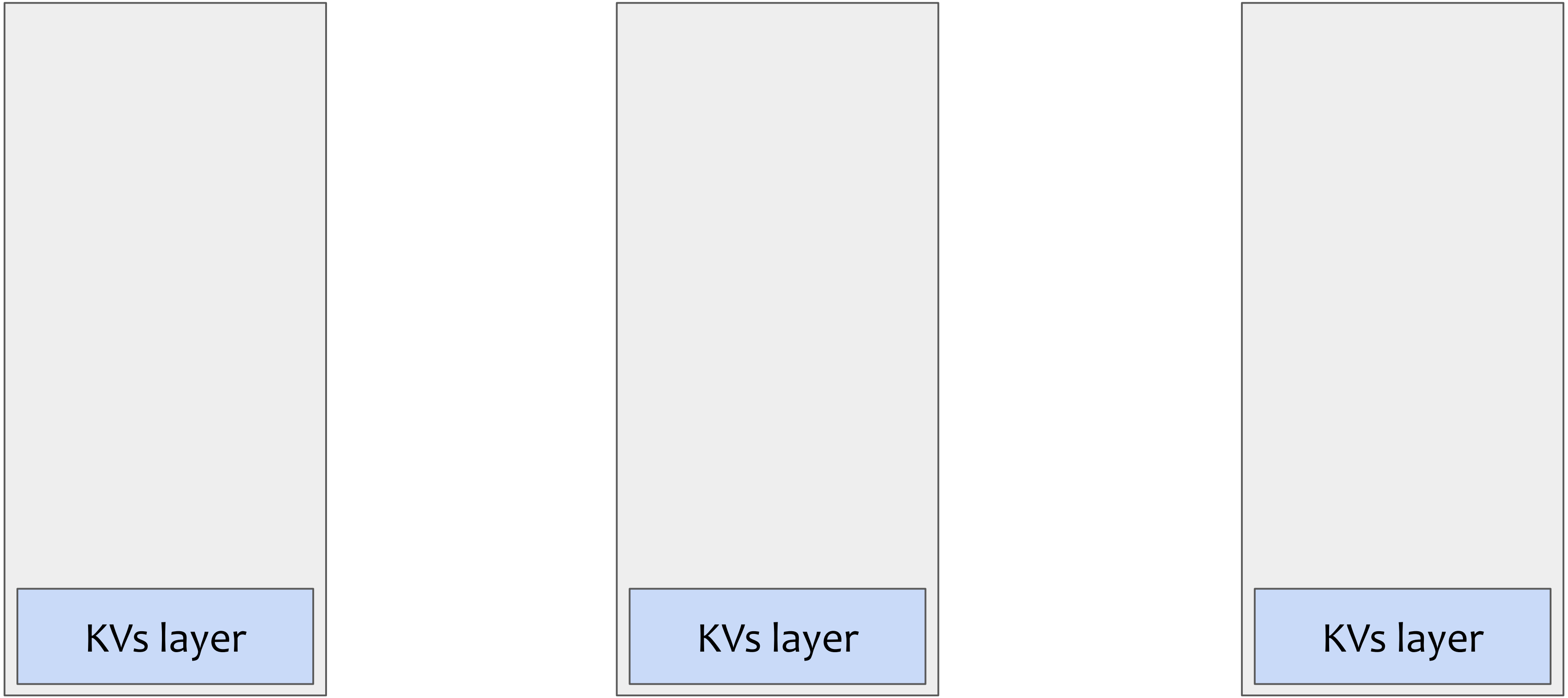
- Nodes need to know what state they are in when they come back from a failure
- We log events on nodes' persistent storages (logging mechanism)
- Task4: RocksDB comes with a Write-Ahead-Log (WAL)
  - prior to updating the MemTable, the write-phase logs the data to the WAL

How to avoid blocking upon failures?

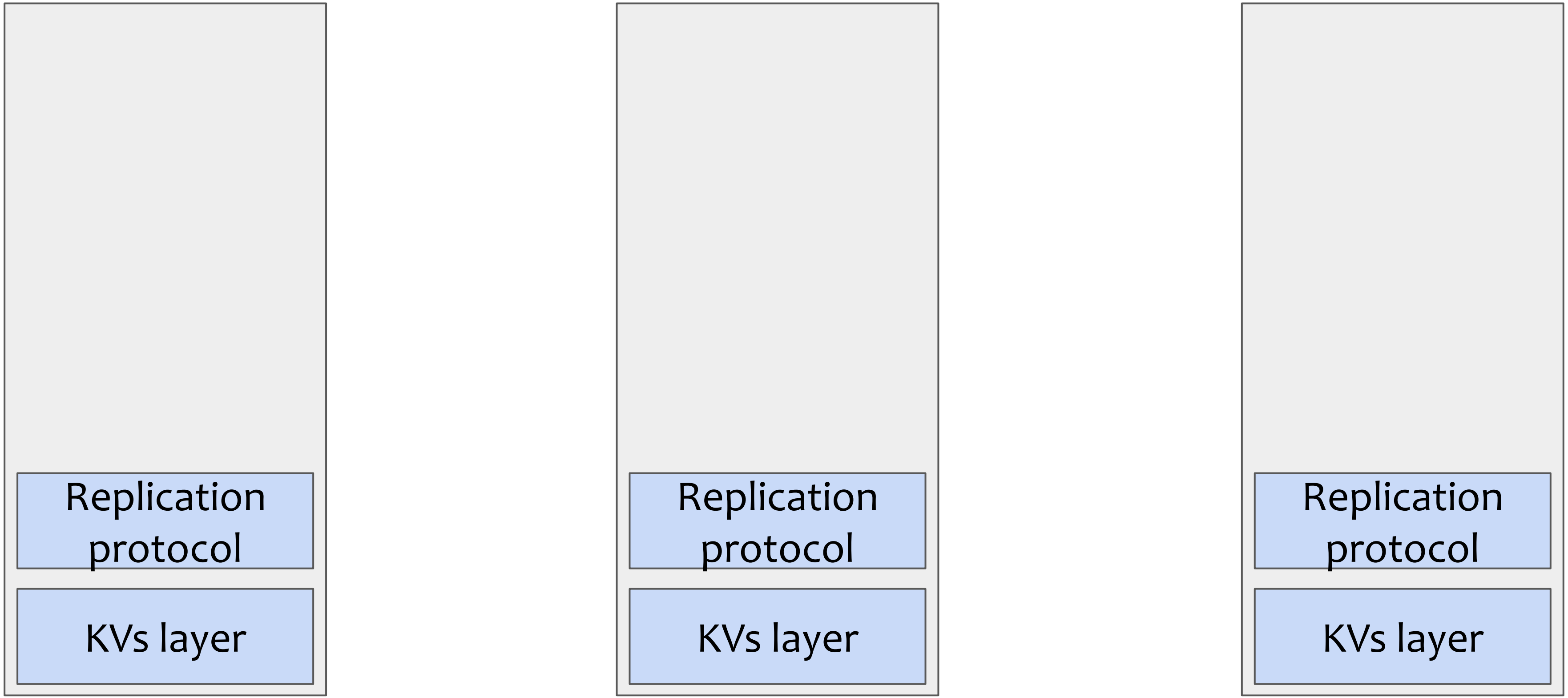
- Three-phase commit protocol (3PC)
  - introduces a (extra) pre-commit phase
  - a participant can shepherd the protocol

- Replication similarly to task #3!
  - Raft leaders are participating in transactions execution
  - If a leader fails, a new leader takes over and continues the protocol

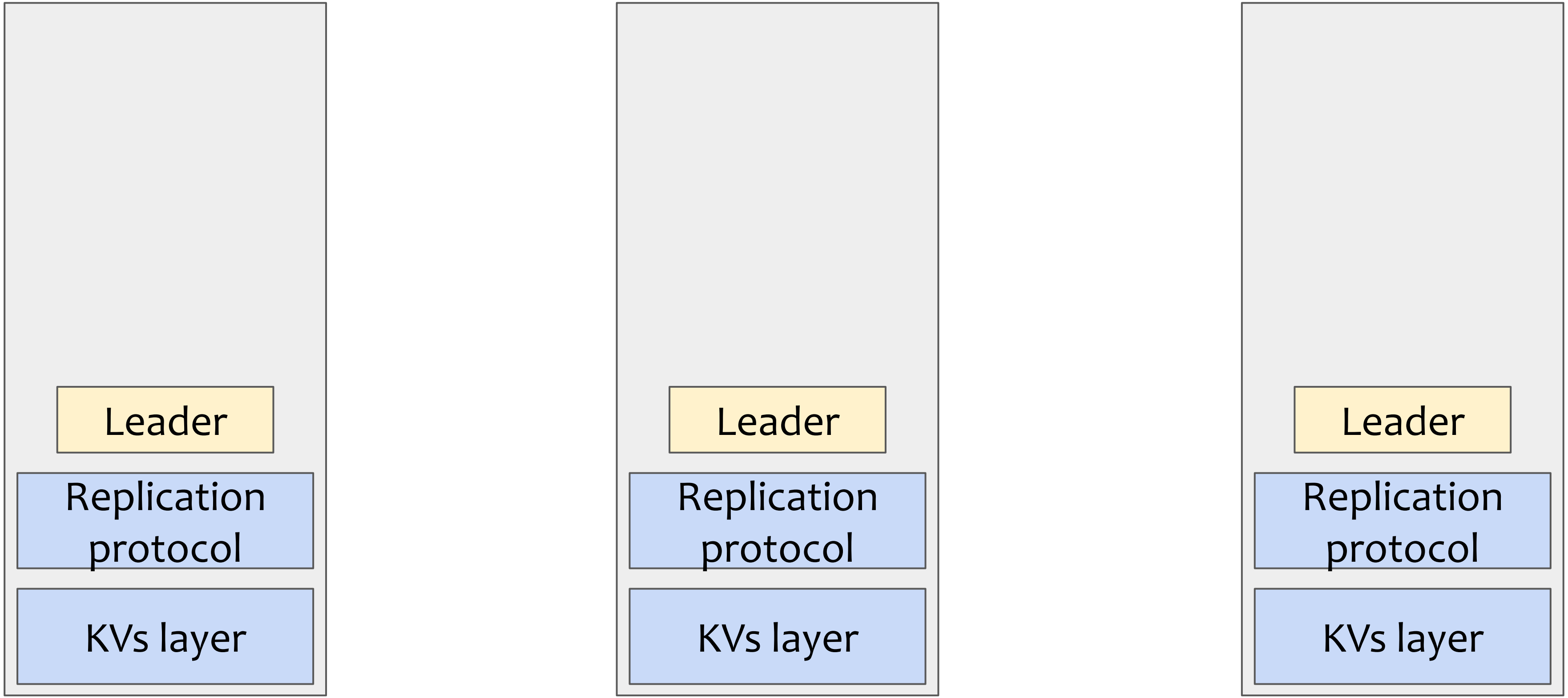
Transactions with replication



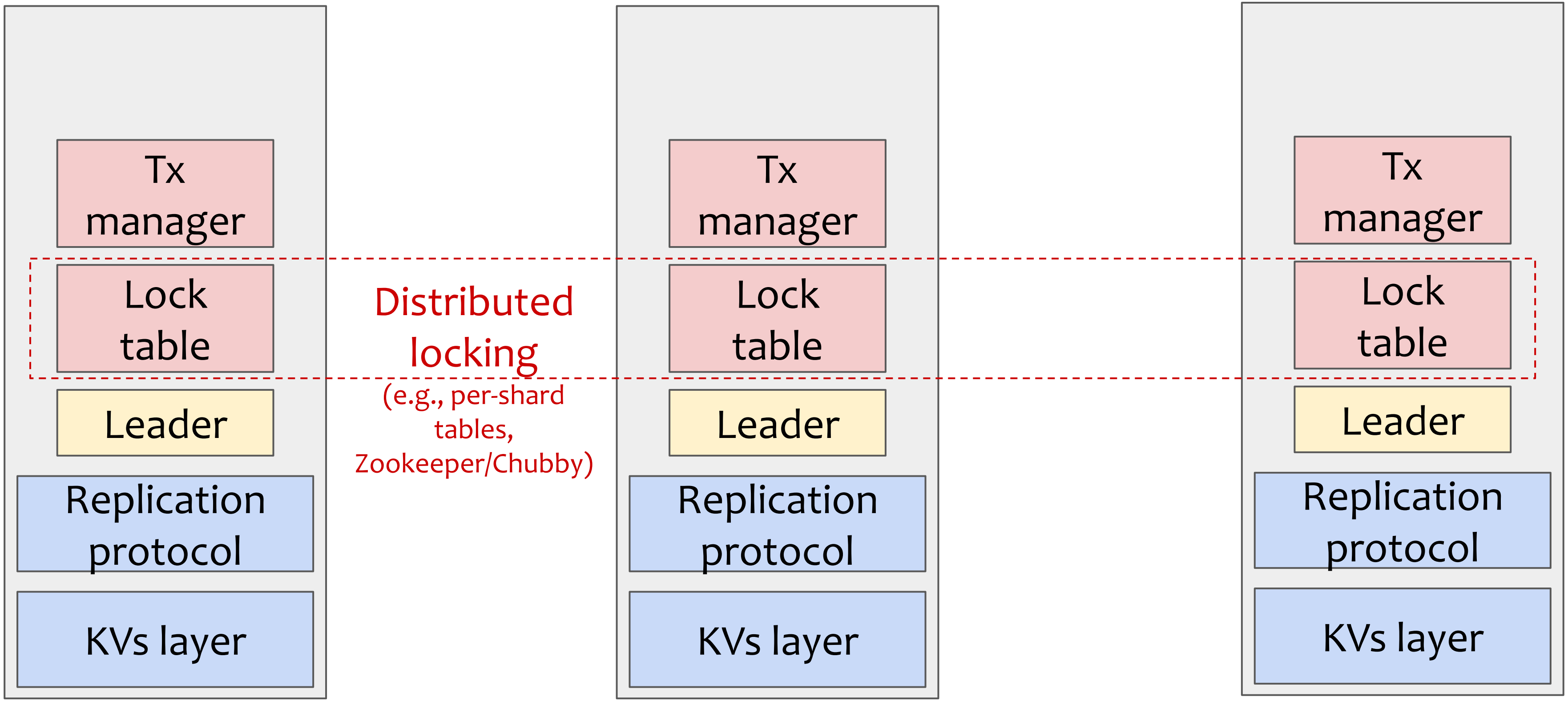
Transactions with replication



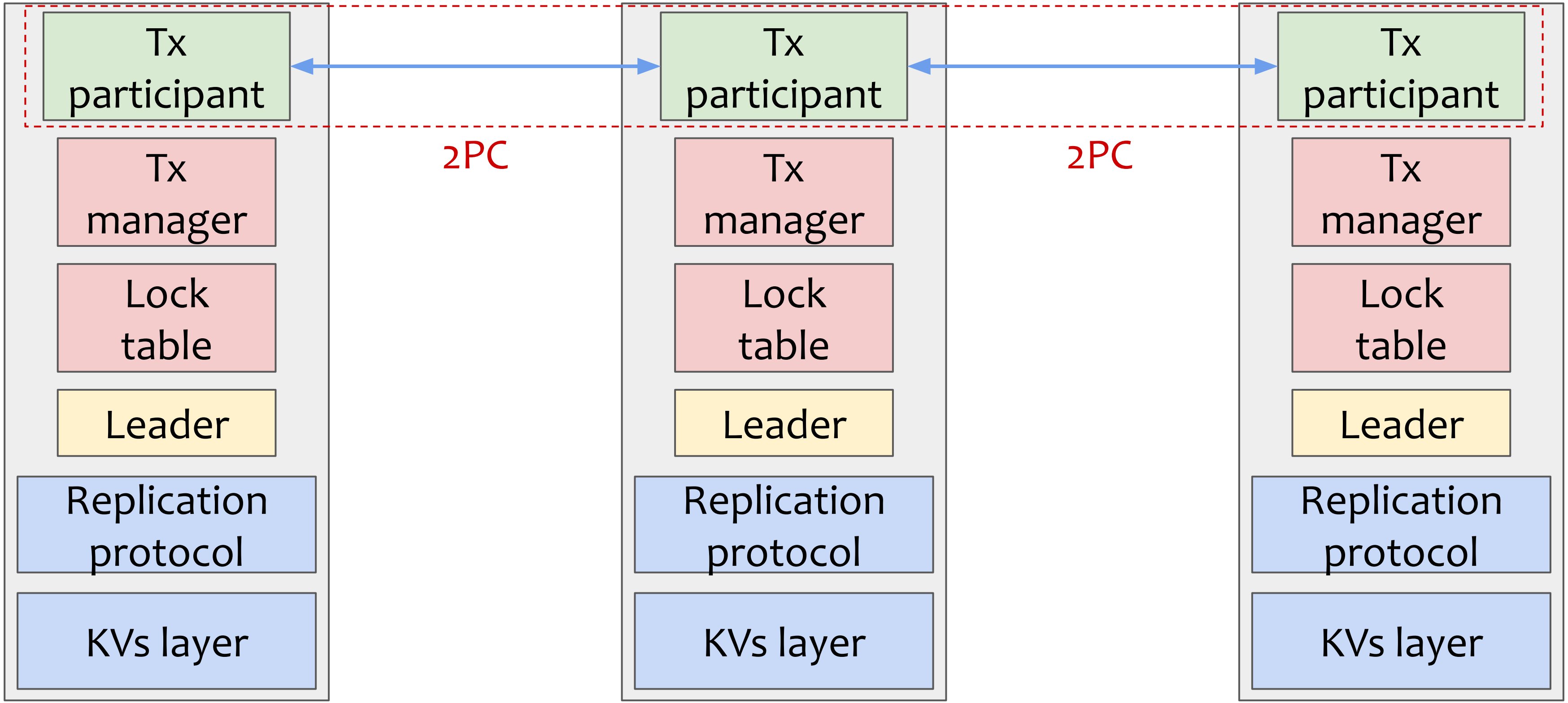
Transactions with replication



Transactions with replication



Transactions with replication



Further reading

- *Concurrency Control and Recovery in Database Systems*, Chapter 7: Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987)

- Spanner: Google's Globally-Distributed Database:
    - https://static.googleusercontent.com/media/research.google.com/en//archive/spanner-osdi2012.pdf

- Martin Kleppmann's online presentations:
    - Two-phase commit
    - Google Spanner

Task #4

Make a distributed KVs to support distributed transactions:

1. Implement single-node Txs
   a. explore RocksDB's transactions and recovery mechanisms
2. Implement distributed Txs **w/o replication**
   a. implement the 2PC with the prepare and commit phases
3. Implement a 2PL protocol
   a. operations take locks as they go along
   b. locks are released after commit/abort