CS 564: Database Management Systems

Lecture 37: Distributed Transactional Database

Xiangyao Yu

4/24/2024

Announcement

Past final exams posted on canvas

Final exam

- May 6th, 10:05am-12:05pm
- Room 1: VAN VLECK B130 (**lastname ≤ 'Mao'**)
- Room 2: NOLAND 132 (**lastname > 'Mao'**)
- Two cheat sheets allowed; can reuse the midterm one
- Cumulative with roughly 70% content after midterm
- Contact instructor by **May 1st** if need McBurney accommodation

Course evaluation: https://heliocampusac.wisc.edu/

- 35 out of 250 responded

Agenda

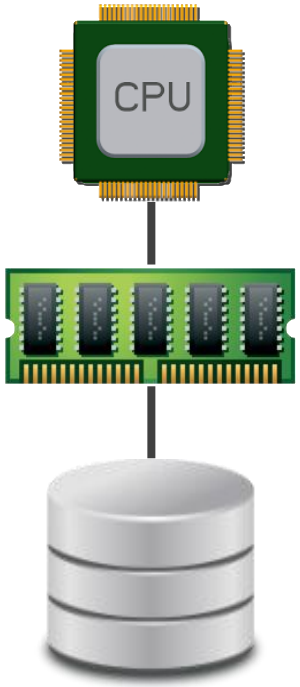
Partitioned Distributed DBMS

- **Distributed concurrency control**
- Distributed index
- Two-phase commit

Data replication

- Primary-backup (active-passive) replication
- Active-active replication

Centralized 2PL



CPU

Memory

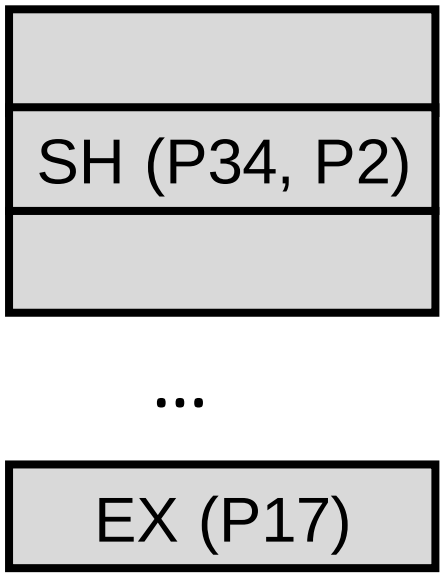
Disk

Centralized database

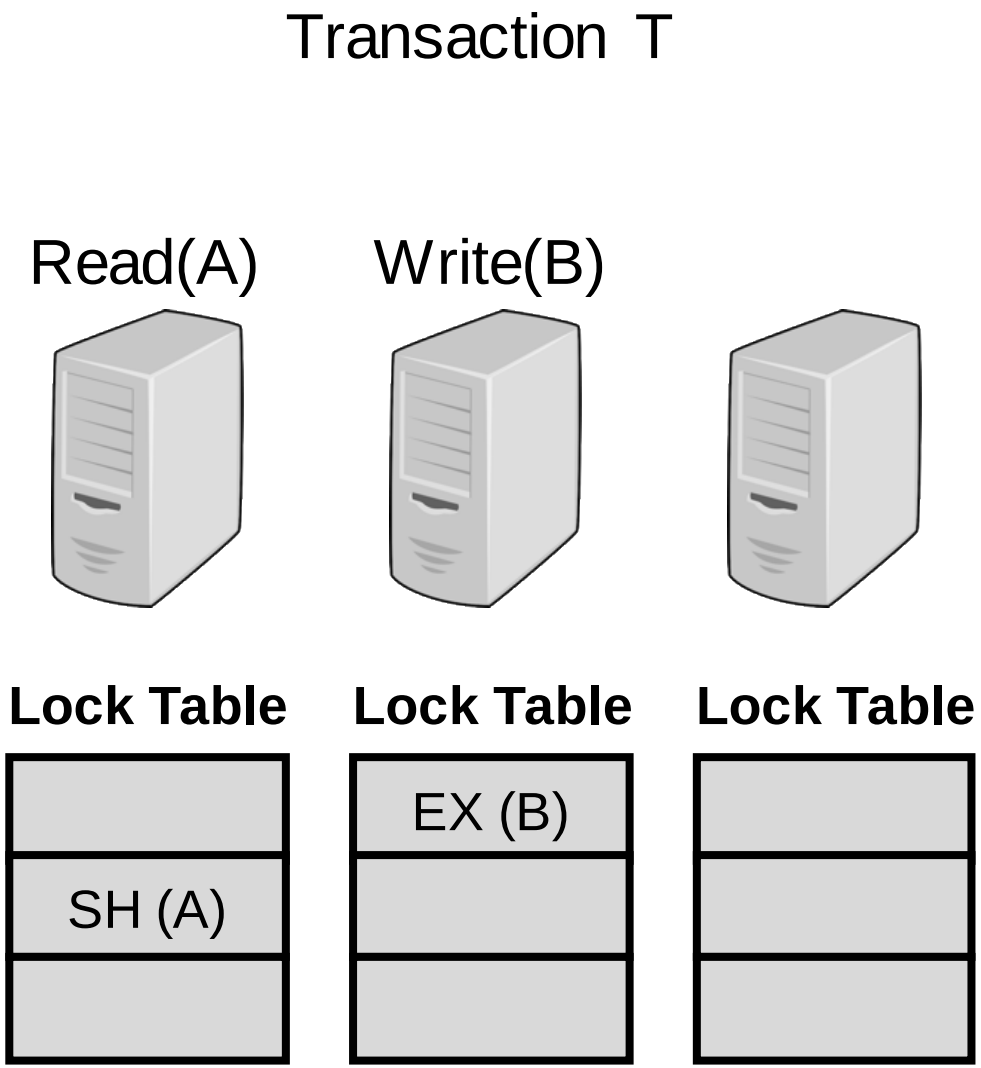
Two-phase locking (2PL)

– Hold locks through a lock table

Lock Table



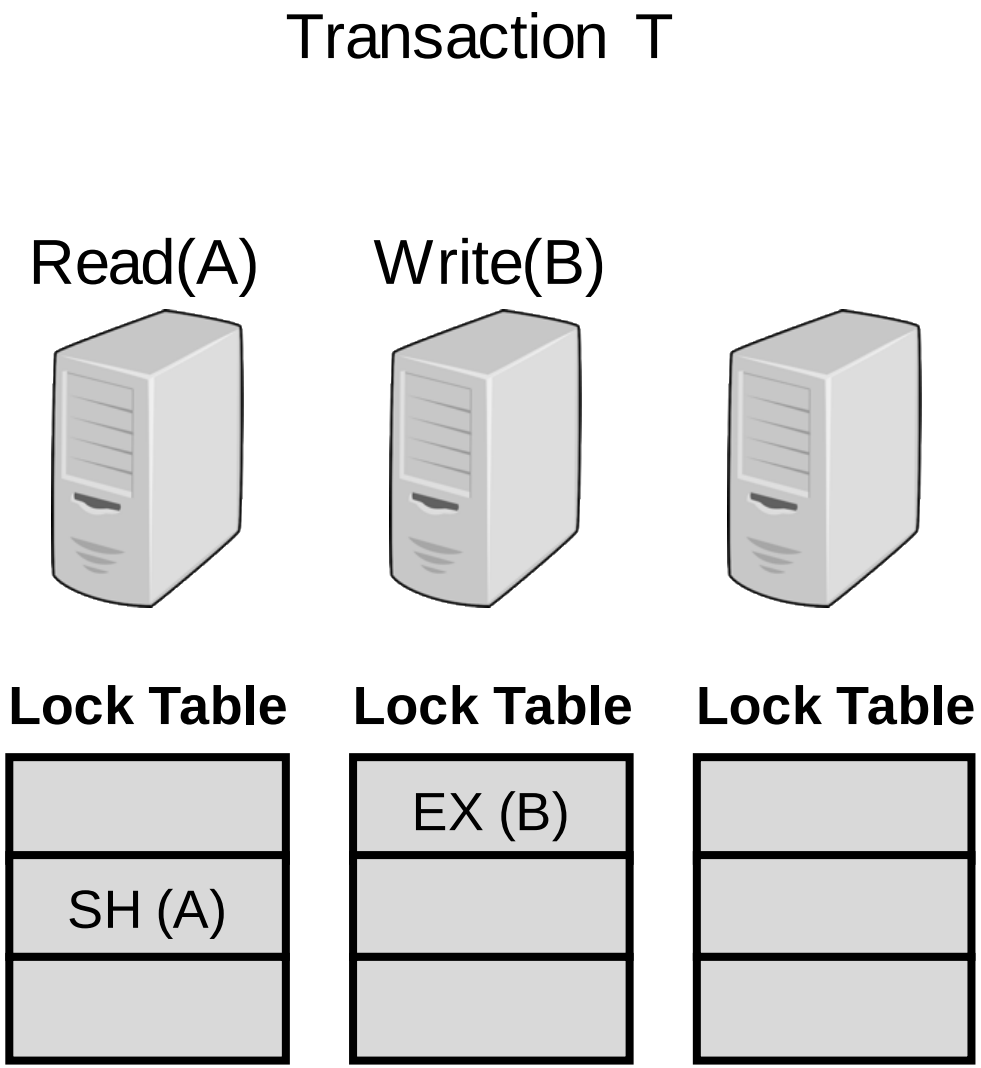
Distributed 2PL



Two-phase locking (2PL)

- Each node has a local lock table
- A transaction locks a page in the corresponding lock table

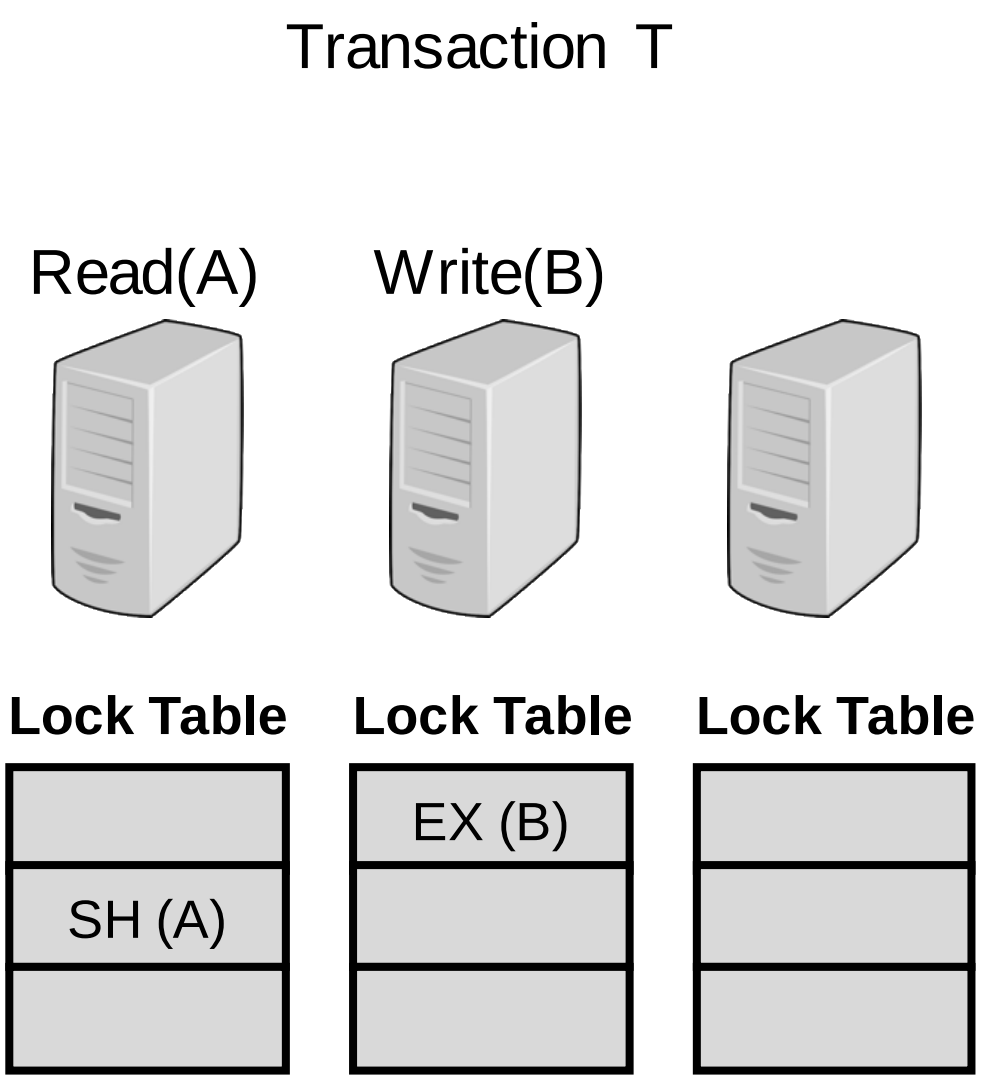
Distributed 2PL—Deadlock



How to handle deadlock ?

- **Solution 1**: Cycle detection in waits-for graph
- **Challenge 1:** Hard to perform cycle detection in a distributed graph
- **Challenge 2**: Maintaining a centralized graph creates a performance bottleneck

Distributed 2PL—Deadlock



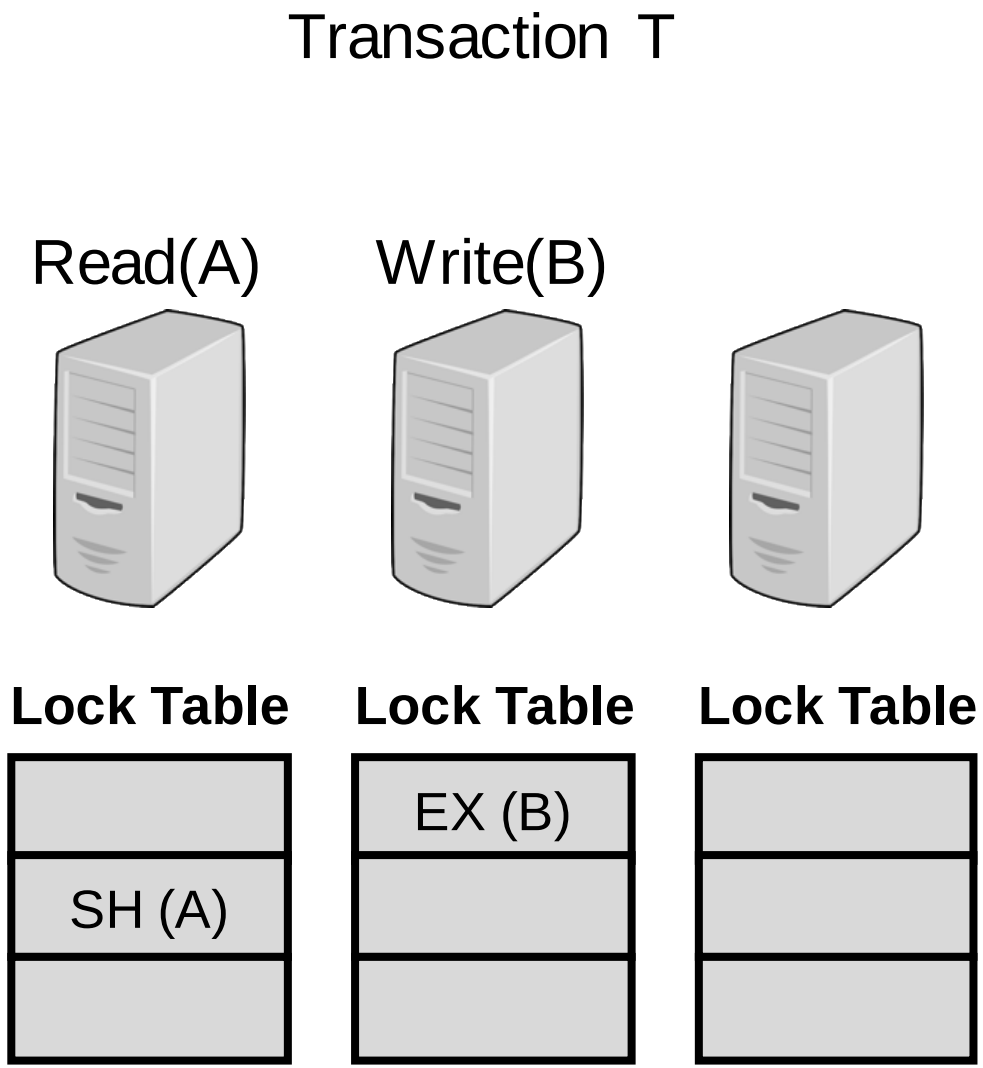
How to handle deadlock ?

– **Solution 2:** Deadlock prevention

NO WAIT

– Do not wait for lock

Distributed 2PL—Deadlock



How to handle deadlock ?

- **Solution 2:** Deadlock prevention

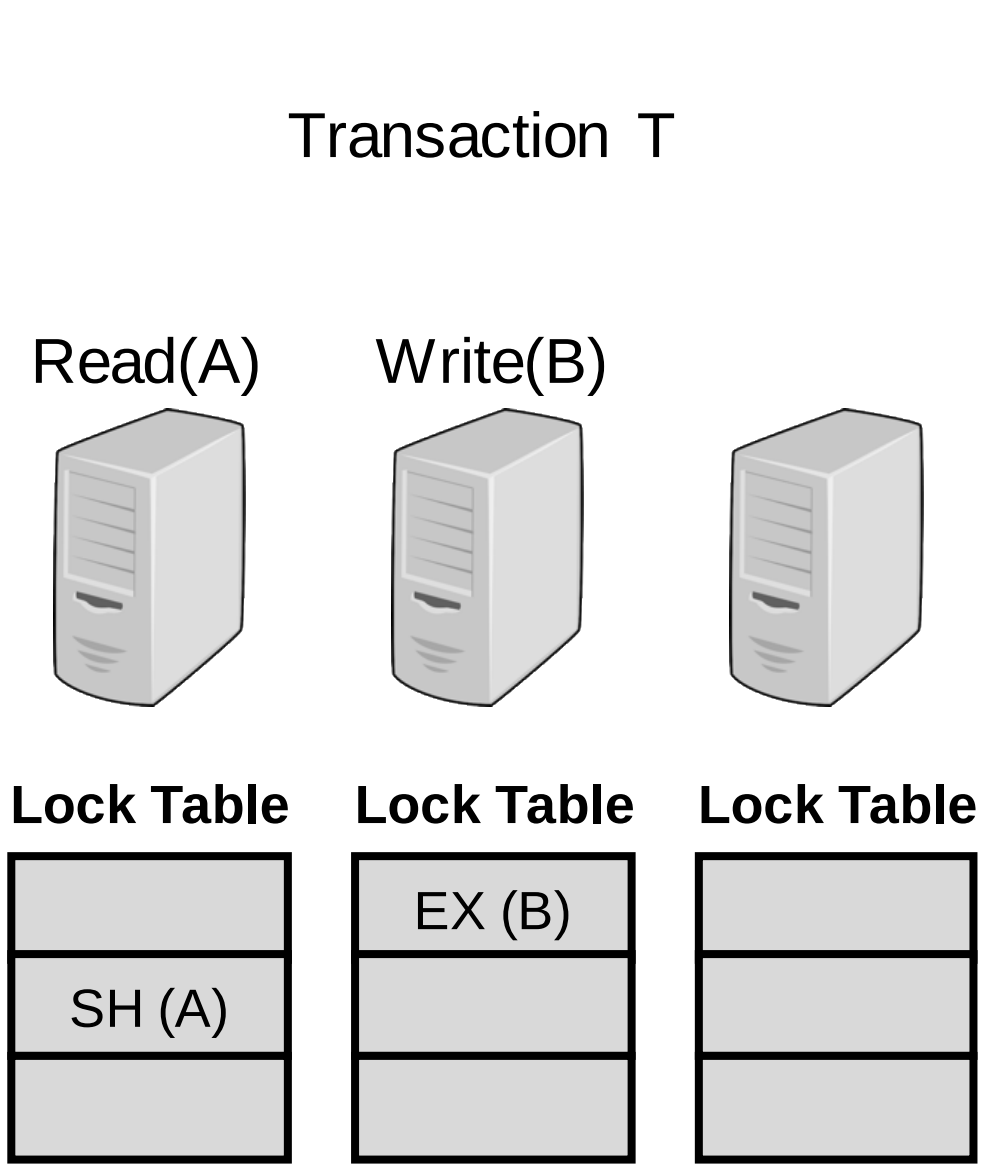
NO WAIT

- Do not wait for lock

WAIT DIE

- Only high-priority txn waits for low-priority txn; low-priority txn self-aborts.
- E.g., MS Orleans

Distributed 2PL—Deadlock



How to handle deadlock ?

- **Solution 2:** Deadlock prevention

NO WAIT

- Do not wait for lock

WAIT DIE

- Only high-priority txn waits for low-priority txn; low-priority txn self-aborts.
- E.g., MS Orleans

WOUND WAIT

- Only low-priority txn waits for high-priority txn; high-priority txn preemptively aborts low-priority txns
- E.g., Google Spanner

Distributed OCC

Transaction T

Read(A) Read(B) Write(C)

A.version B.version

Each read returns the value and a version number

Each committed write update both the value and the version number

During validation, check whether any record in the read-set has been modified since the earlier read, by comparing the version numbers

Distributed MVCC

Need to generate monotonically increasing timestamp

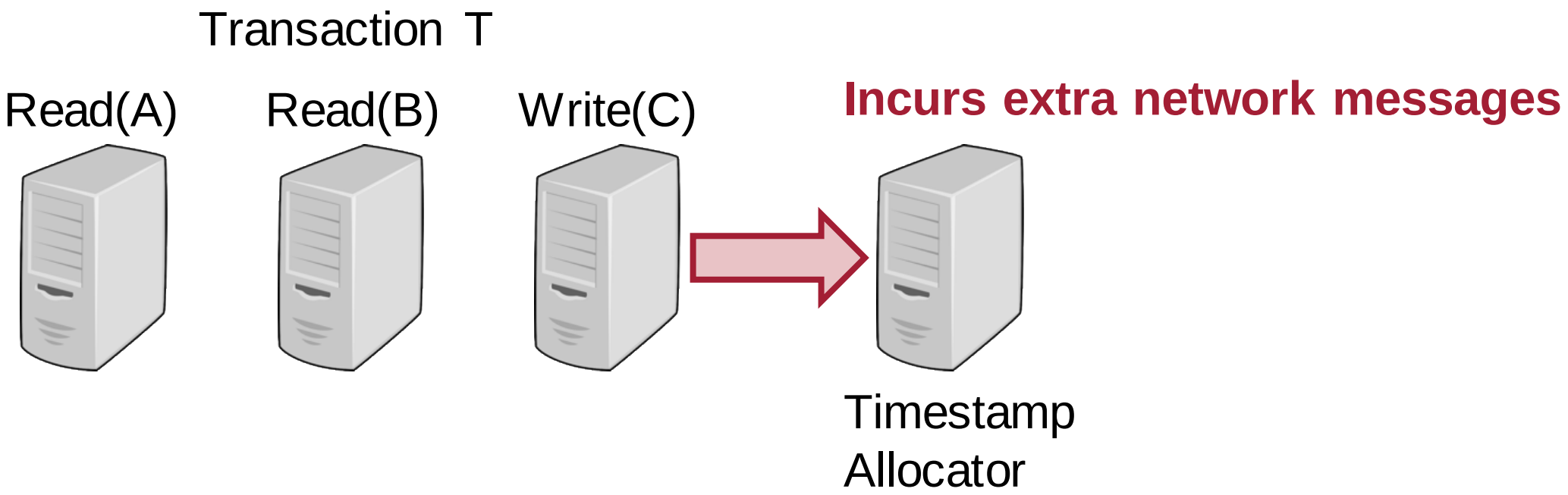
- Each transaction is assigned a timestamp (*ts*) when the transaction starts
- Each record version has a write timestamp (wts) and a read timestamp (rts)

Distributed MVCC

Need to generate monotonically increasing timestamp

Solution 1: centralized timestamp allocator

- Every txn must contact the timestamp allocator for a timestamp

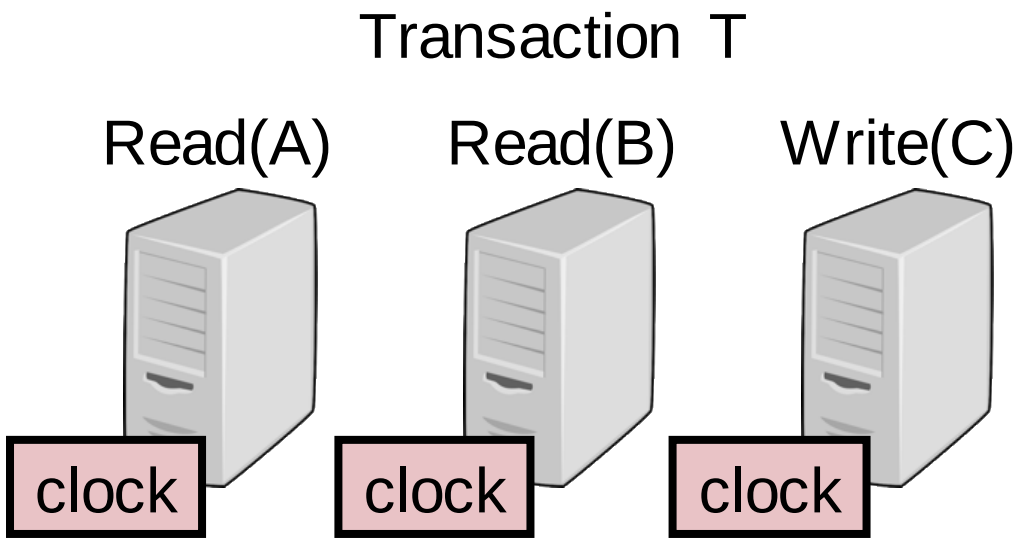


Distributed MVCC

Need to generate monotonically increasing timestamp

Solution 2: synchronized clocks

- Use atomic clock and GPS to synchronize clocks across all servers



Incurs extra cost

Agenda

Partitioned Distributed DBMS

- Distributed concurrency control
- **Distributed index**
- Two-phase commit

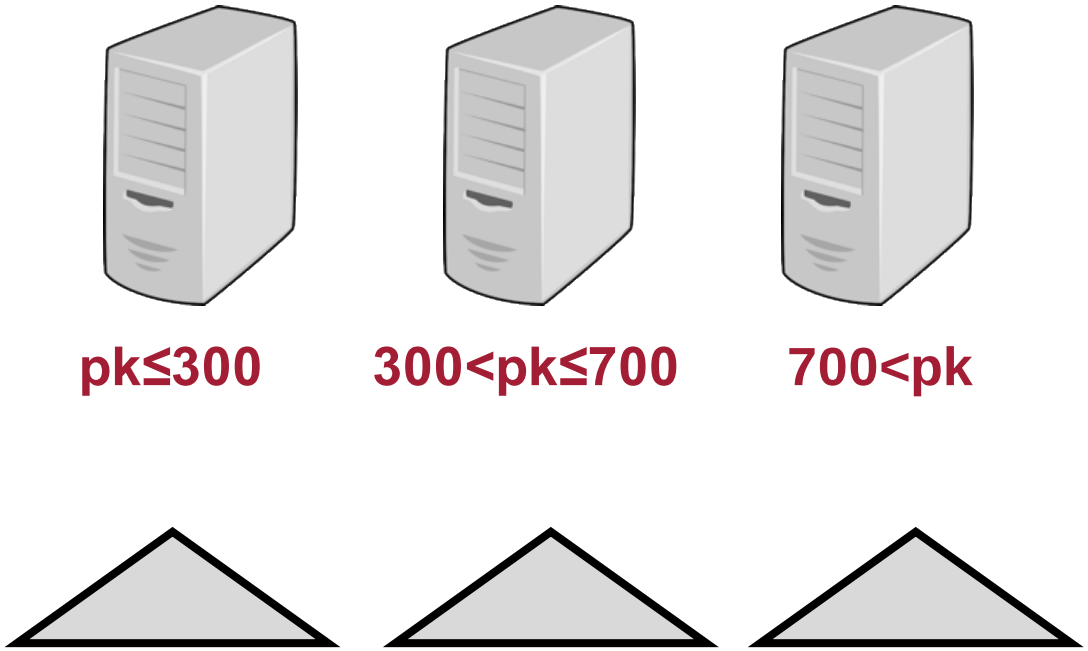
Data replication

- Primary-backup (active-passive) replication
- Active-active replication

Distributed Index

Index must be distributed across servers

Typically partition data based on primary key

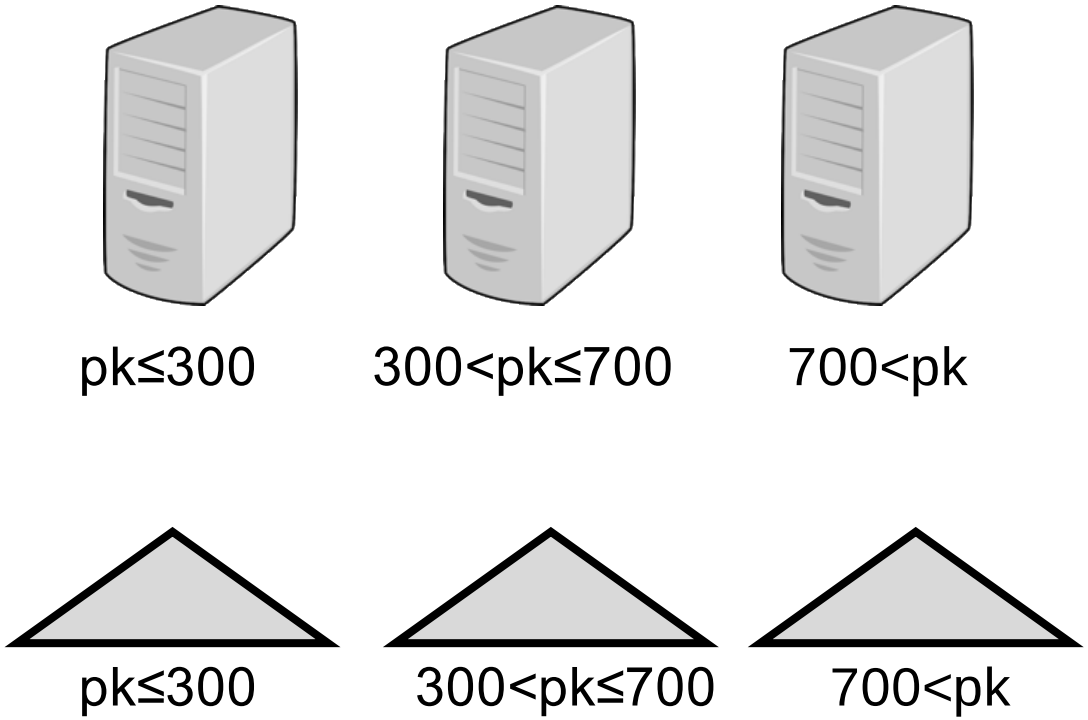


Distributed Primary Index

Each node maintains a local index for its local partition of data

Index search on PK

- Uses partition function to find the right partition
- Search the local index in that partition to locate record

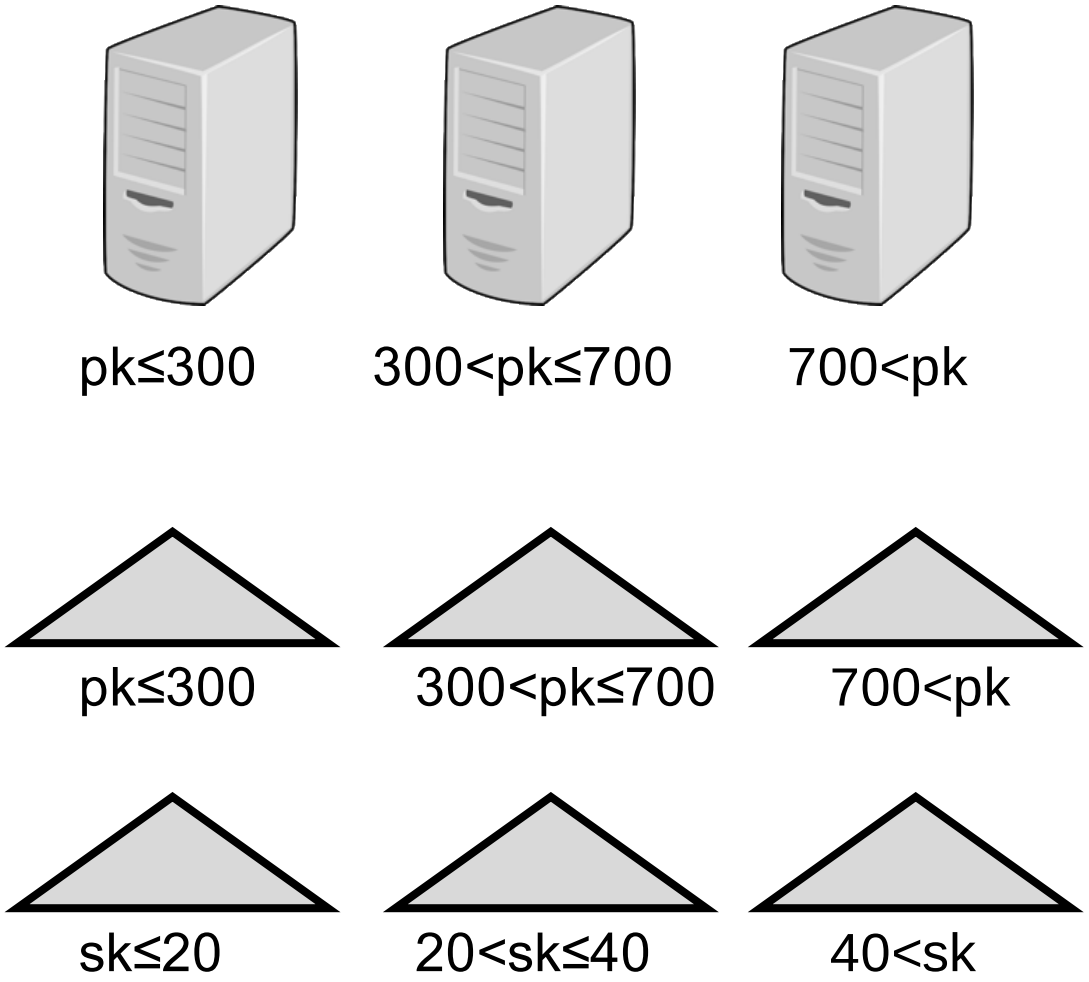


Distributed Secondary Index

Each node maintains a local partition of the secondary index

Index search on Secondary Key (SK)

- Uses SK partition function to find the right partition
- Search the local SK index in that partition to find PK
- Search PK index to locate record



Agenda

Partitioned Distributed DBMS

- Distributed concurrency control
- Distributed index
- **Two-phase commit**

Data replication

- Primary-backup (active-passive) replication
- Active-active replication

Atomic Commit Protocol (ACP)

Atomic commit protocol: all partitions reach the same commit or abort decision of a transaction

Example:

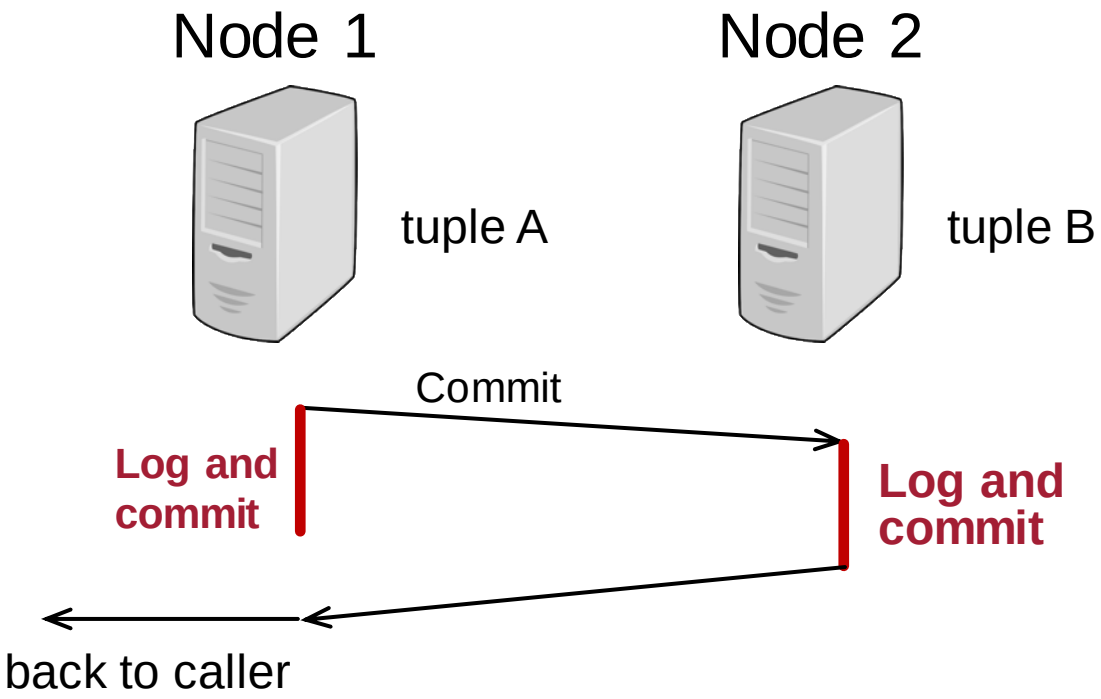
tuple A

tuple B

Transaction T:
write(A)
write(B)

The two updates must commit or abort atomically

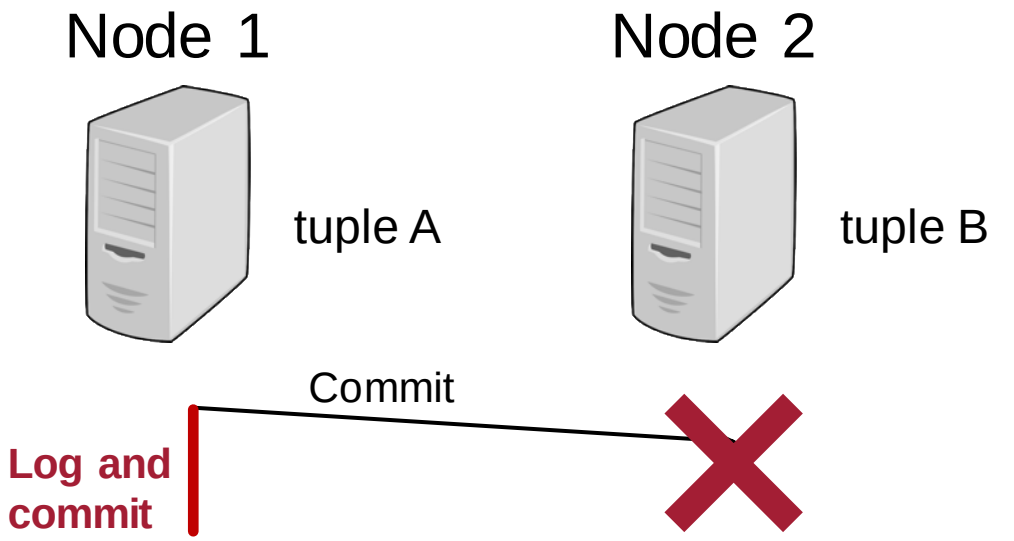
The Challenge of Atomic Commit



Transaction T:
write(A)
write(B)

A naïve approach: all nodes log and commit independently

The Challenge of Atomic Commit



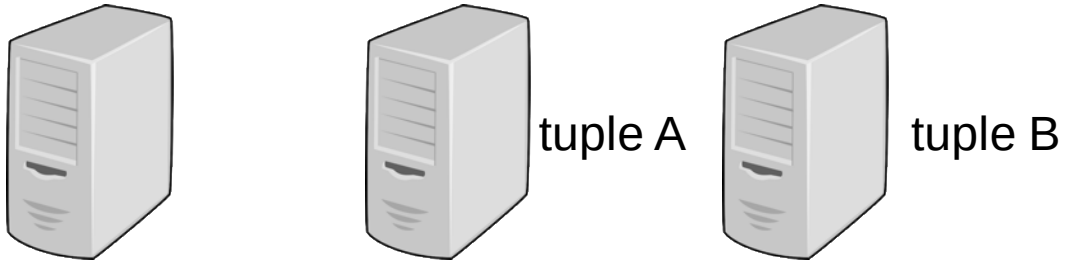
Transaction T:
write(A)
write(B)

A naïve approach: all nodes log and commit independently

Node 2 crashes before logging

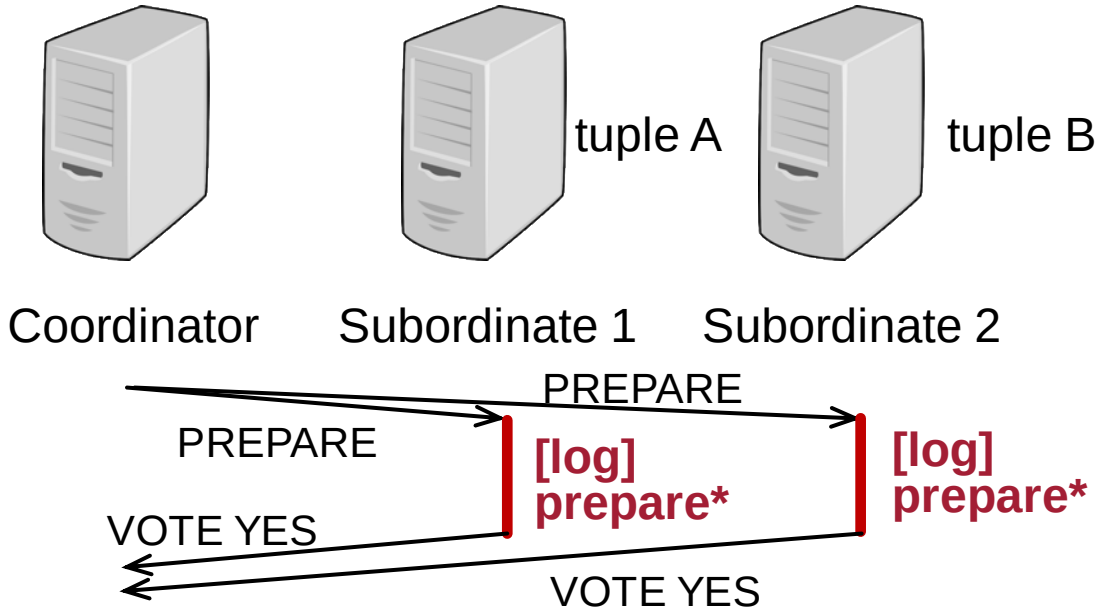
- Transaction T commits in node 1 but not in node 2

Two-Phase Commit (2PC)



Key idea: let the coordinator log the final commit/abort decision

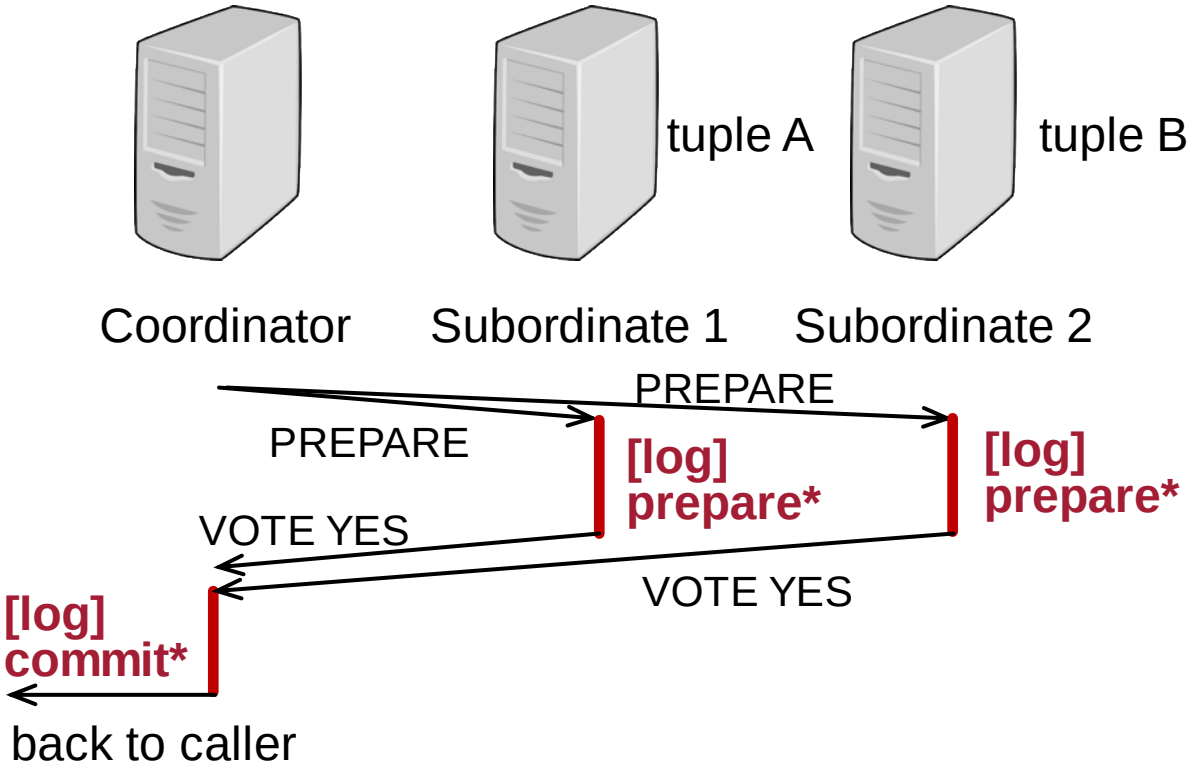
Two-Phase Commit (2PC)



Key idea: let the coordinator log the final commit/abort decision

Phase 1: prepare phase

Two-Phase Commit (2PC)



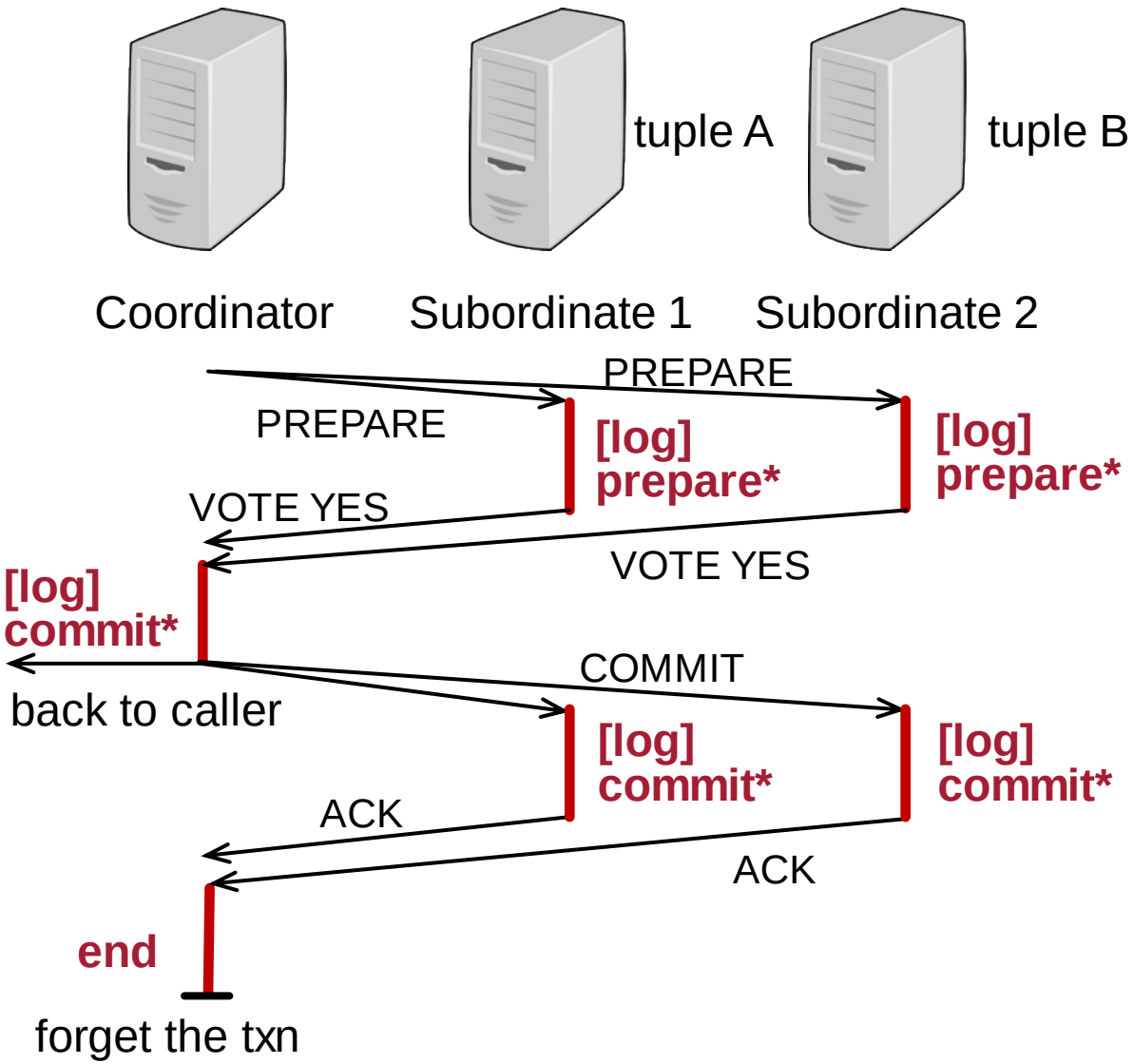
Key idea: let the coordinator log the final commit/abort decision

Phase 1: prepare phase

Phase 2: commit phase

- Coordinator logs the decision

Two-Phase Commit (2PC)



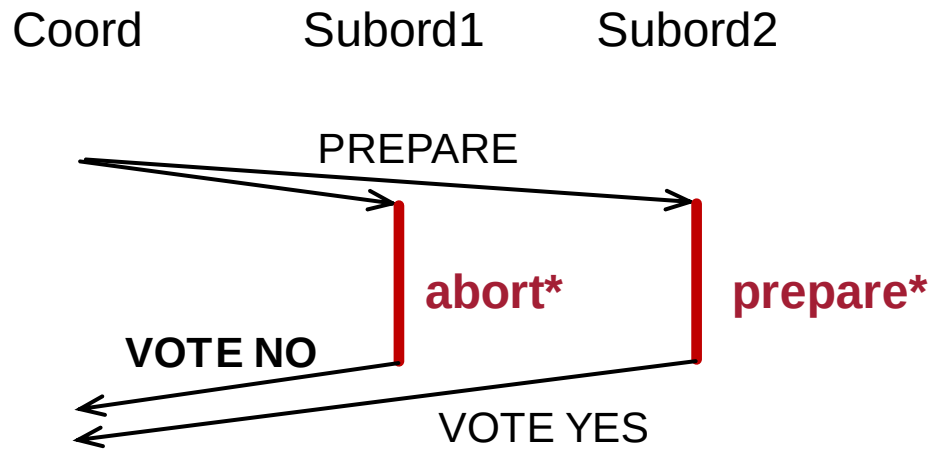
Key idea: let the coordinator log the final commit/abort decision

Phase 1: prepare phase

Phase 2: commit phase

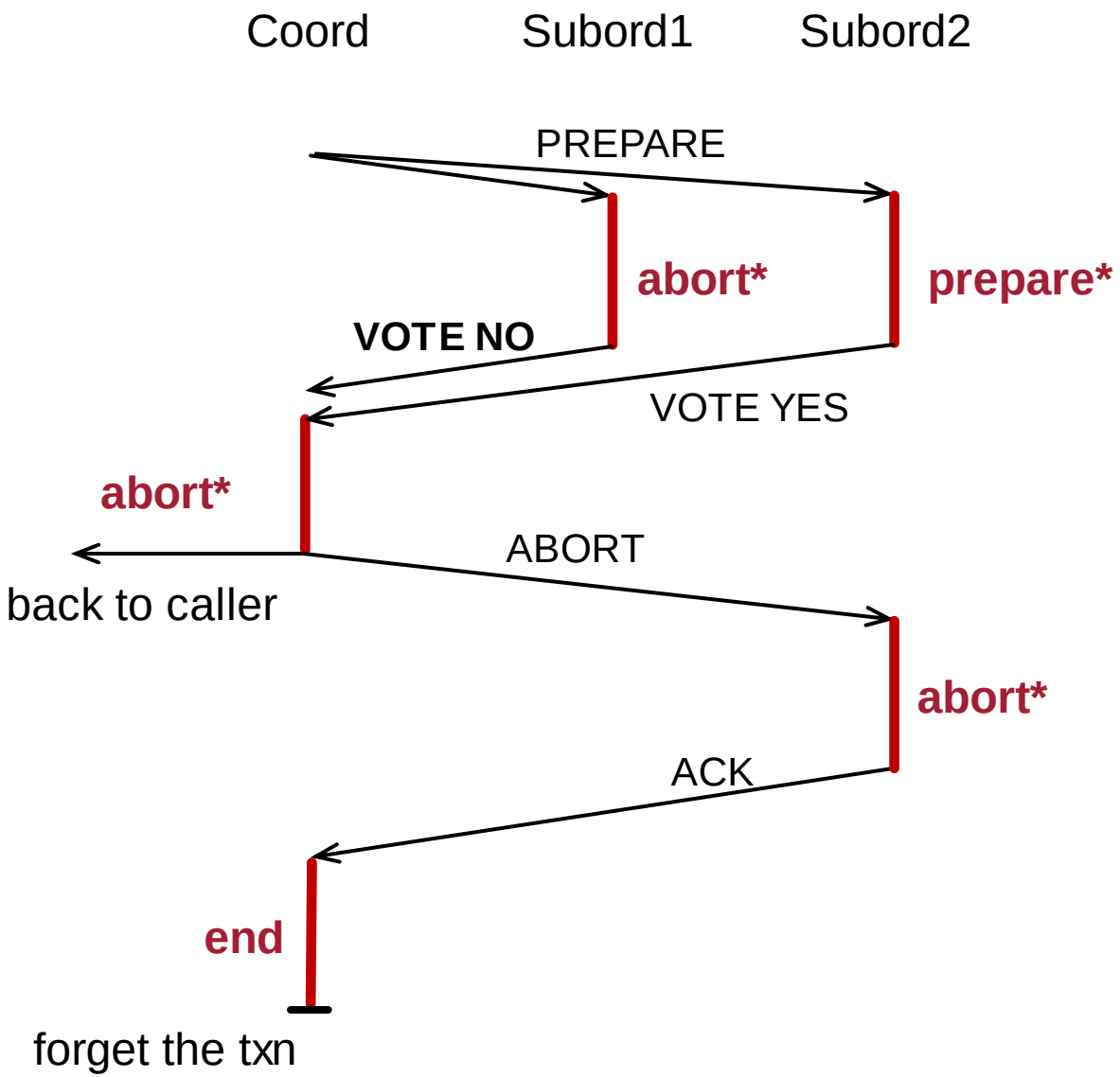
- Coordinator logs the decision
- Coordinator sends the decision to subordinates
- Coordinator forgets the transaction after receiving ACKs

2PC – Abort Example



- Subordinate returns VOTE NO if the transaction is aborted
  - Subordinate can release locks and forget the transaction

2PC – Abort Example

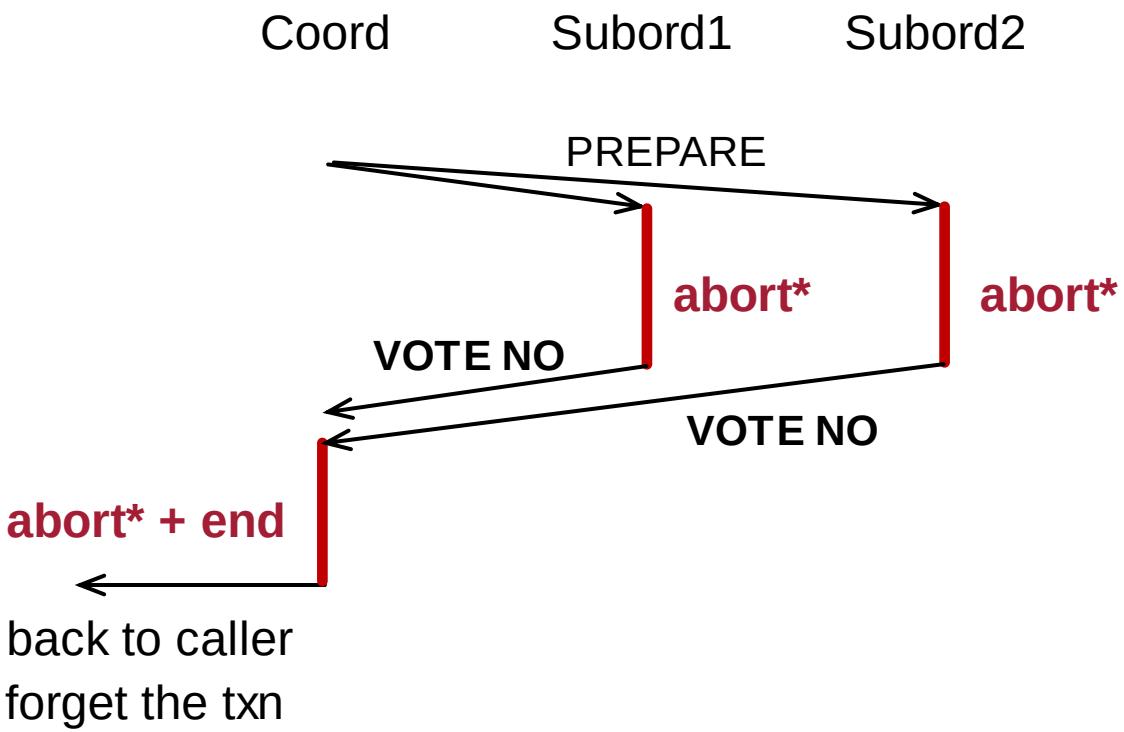


Subordinate returns VOTE NO if the transaction is aborted

- Subordinate can release locks and forget the transaction

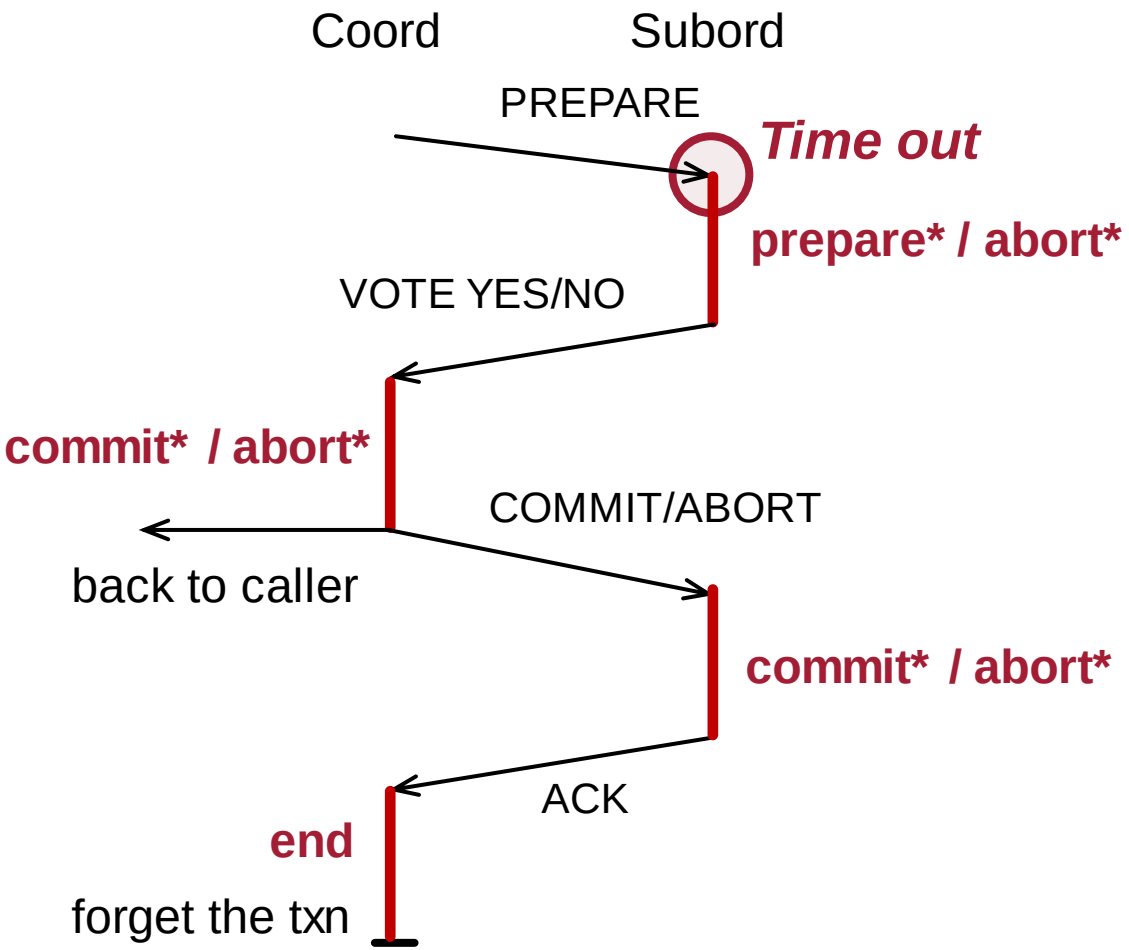
Skip the commit phase for aborted subordinates

2PC – All Subordinates Abort



Skip the second phase entirely if the transaction aborts at all the subordinates

2PC – Failures

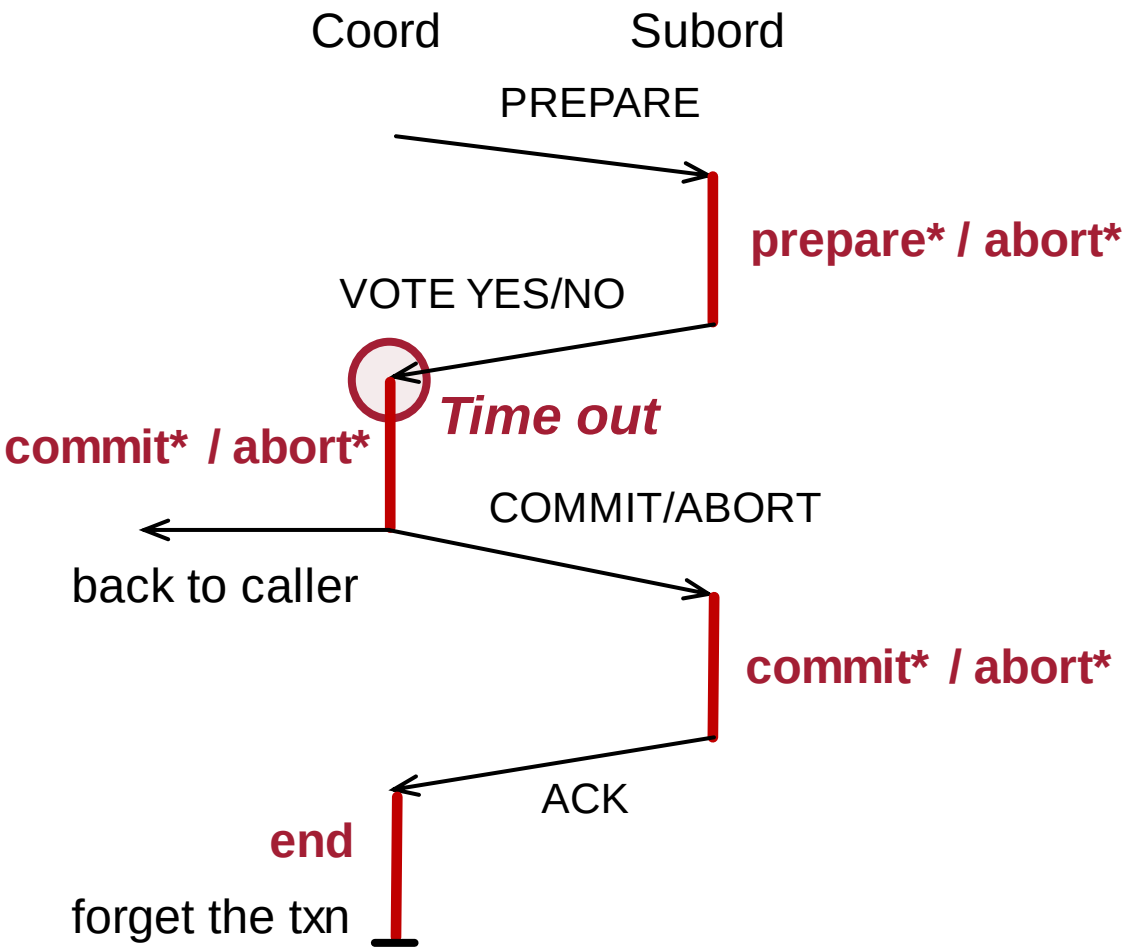


Use timeout to detect failures

Subordinate timeout

– Waiting for PREPARE: self abort

2PC – Failures

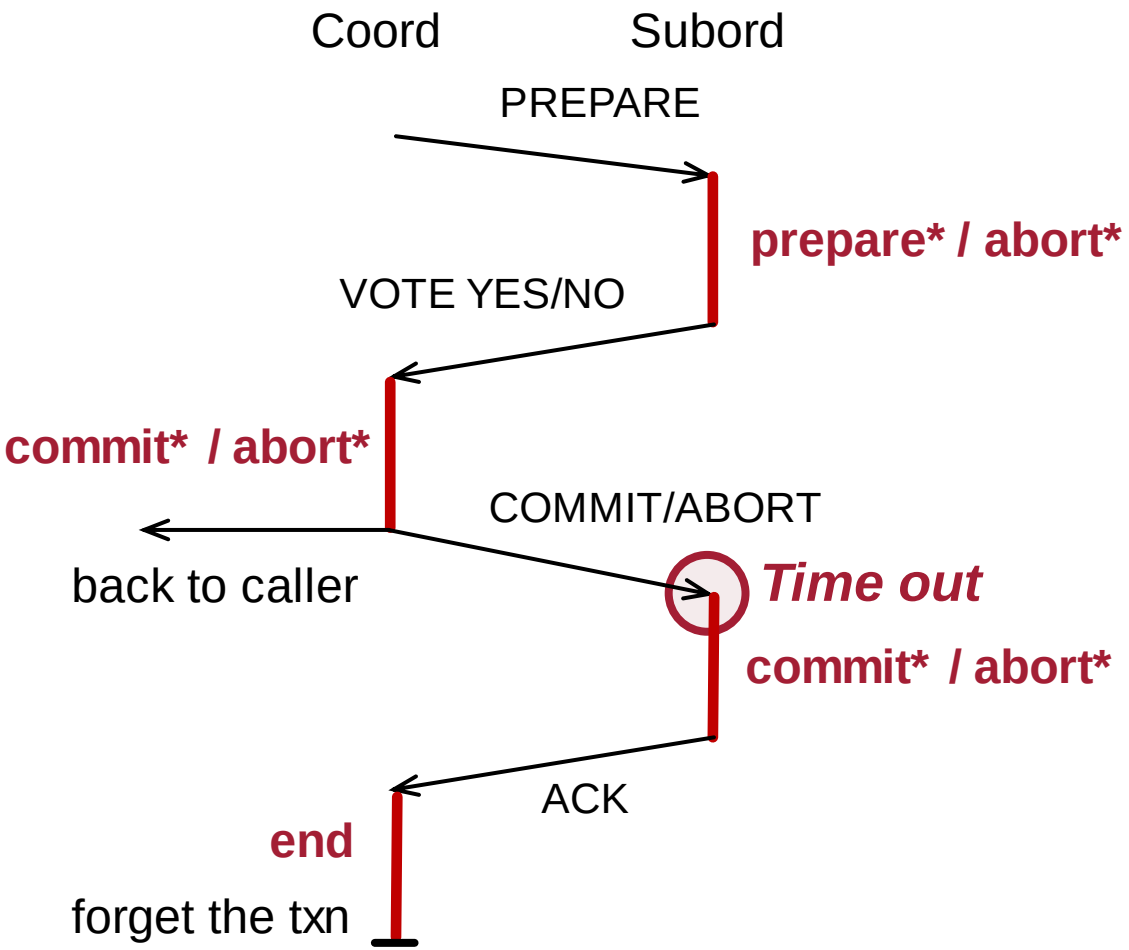


Use timeout to detect failures

Coordinator timeout

- Waiting for vote: self abort

2PC – Failures

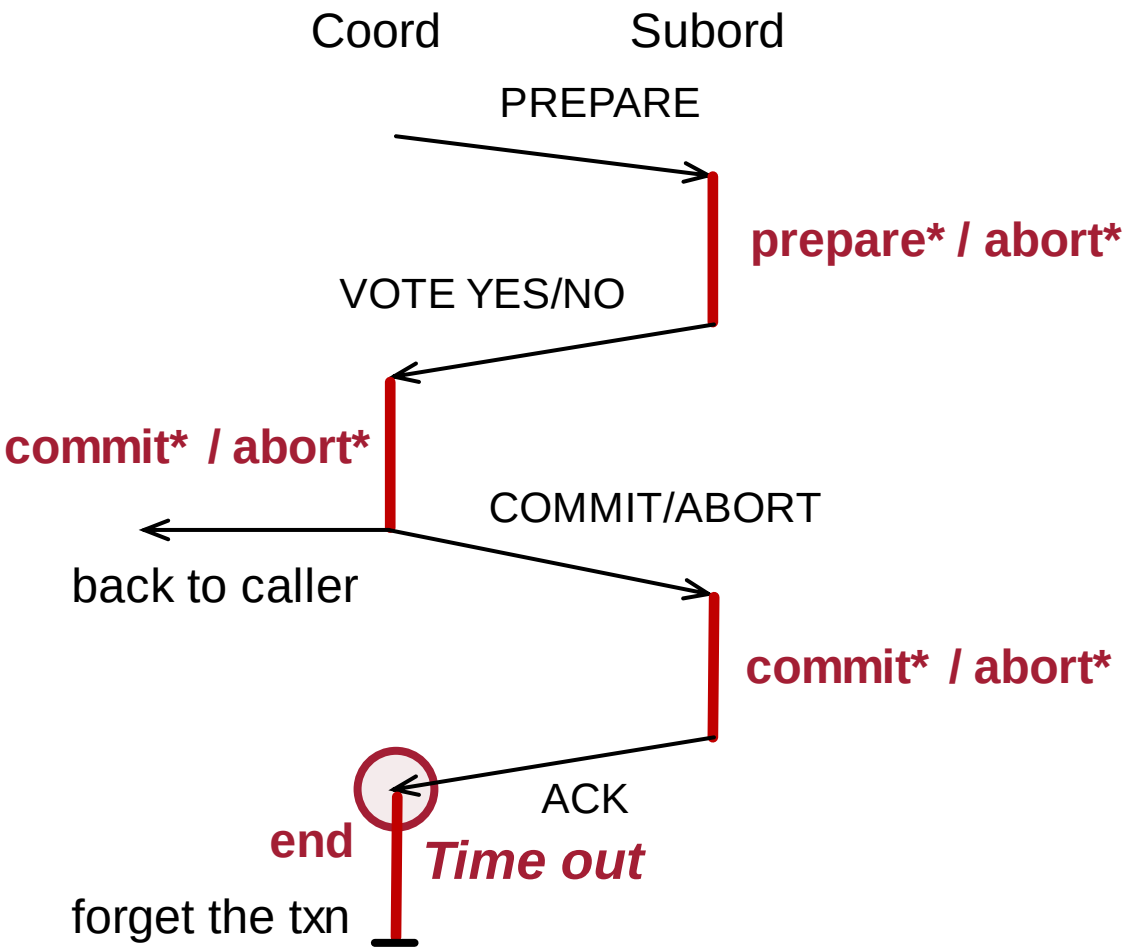


Use timeout to detect failures

Subordinate timeout

- Waiting for decision: contact coordinator or peer subordinates (**may block until the coordinator recovers**)

2PC – Failures

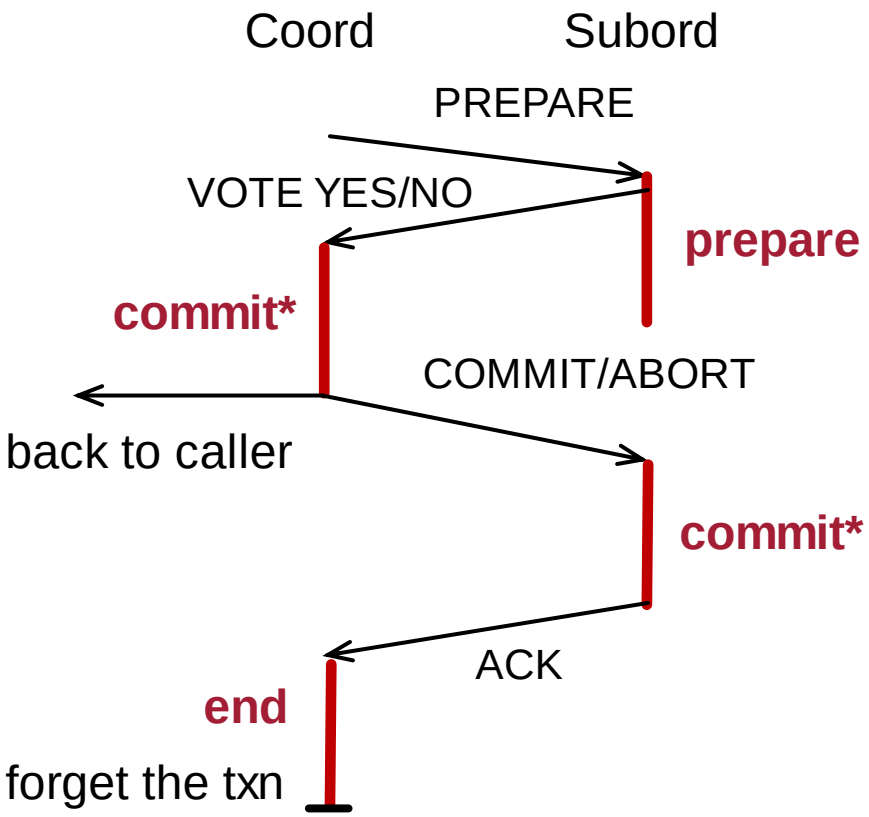


Use timeout to detect failures

Coordinator timeout

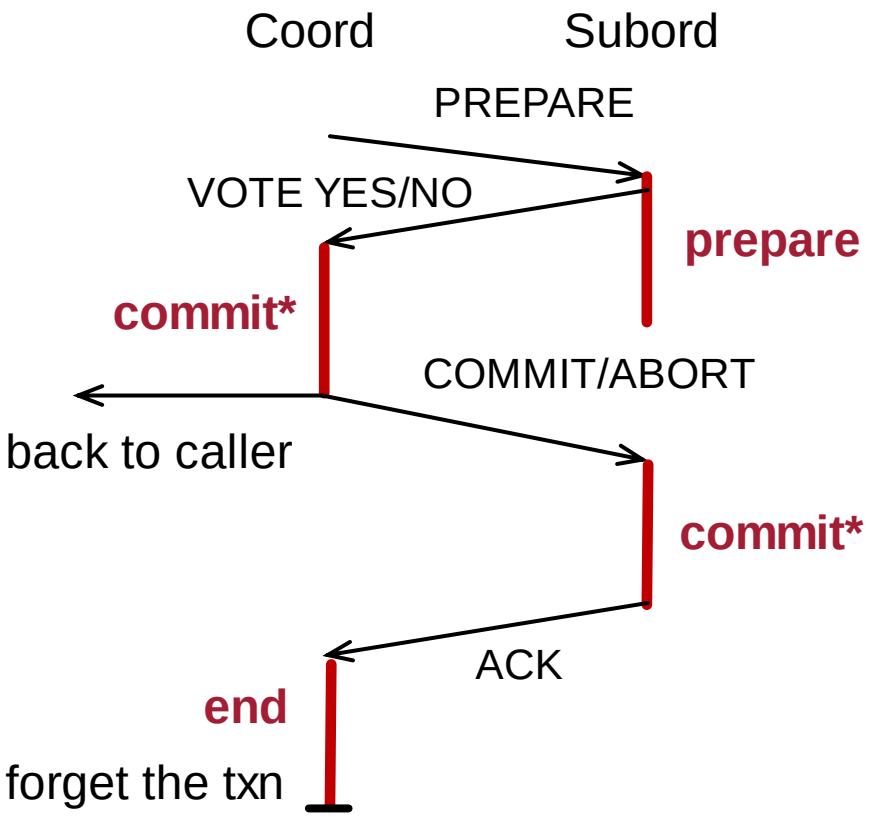
- Waiting for ACK: contact subordinates

2PC – Alternative Designs?



Subordinate returns vote to coordinator before logging prepare?

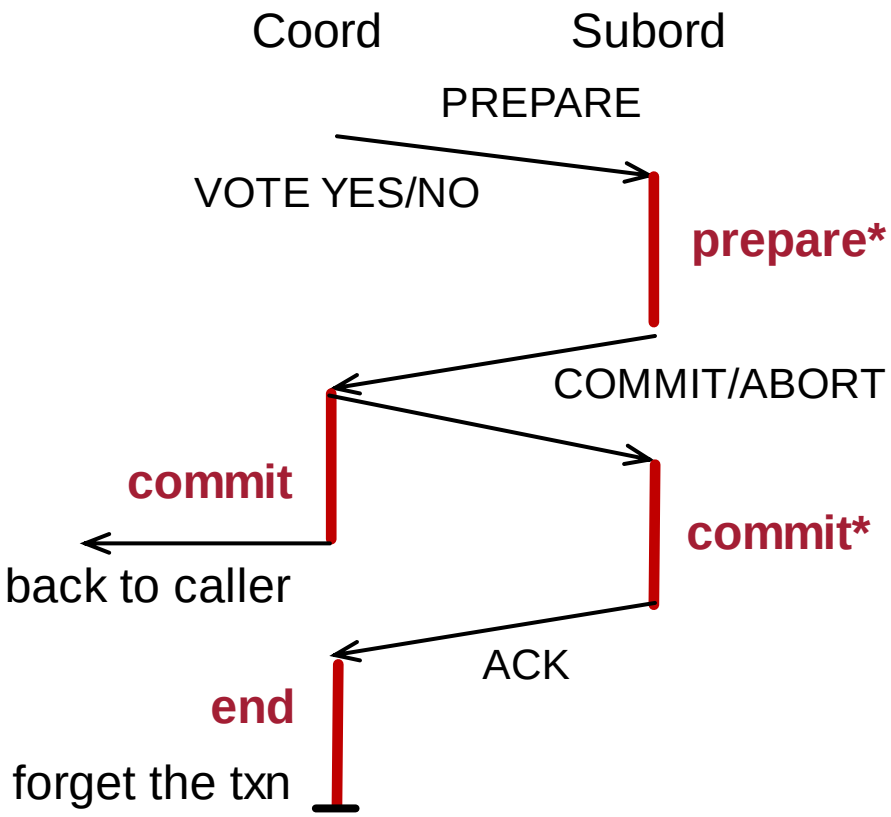
2PC – Alternative Designs?



Subordinate returns vote to coordinator before logging prepare?

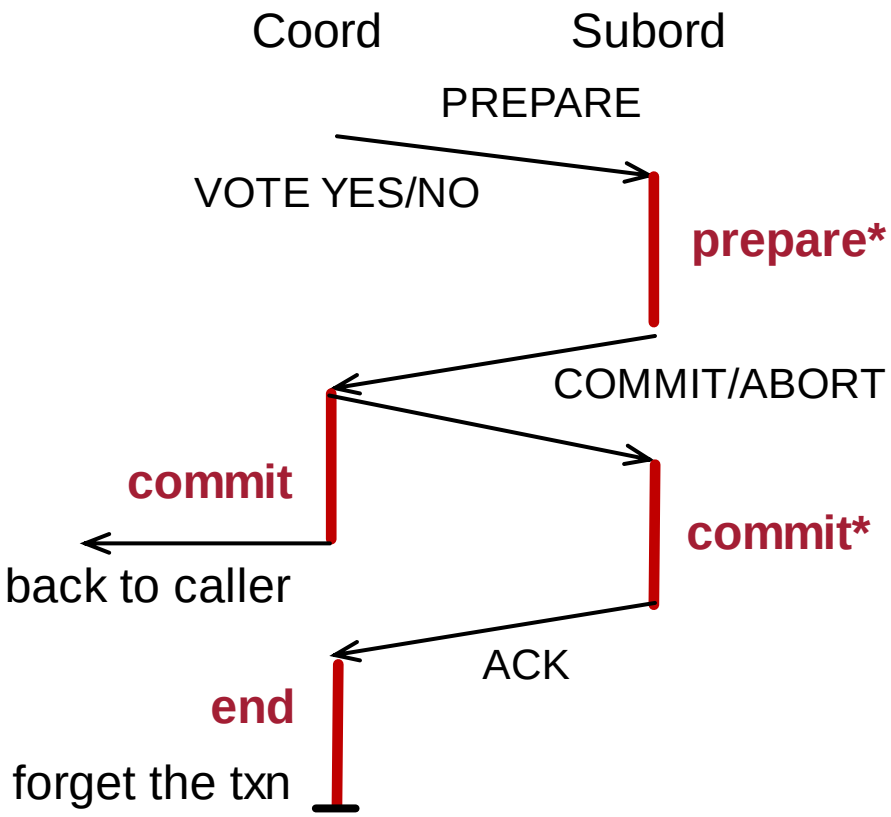
Problem: subordinate may crash before the log record is written to disk. The log record is thus lost but the coordinator already committed the transaction

2PC – Alternative Designs?



Coordinator sends decision to subordinates before logging the decision?

2PC – Alternative Designs?



Coordinator sends decision to subordinates before logging the decision?

Problem: coordinator crashes before logging the decision and decides to abort after restart

Agenda

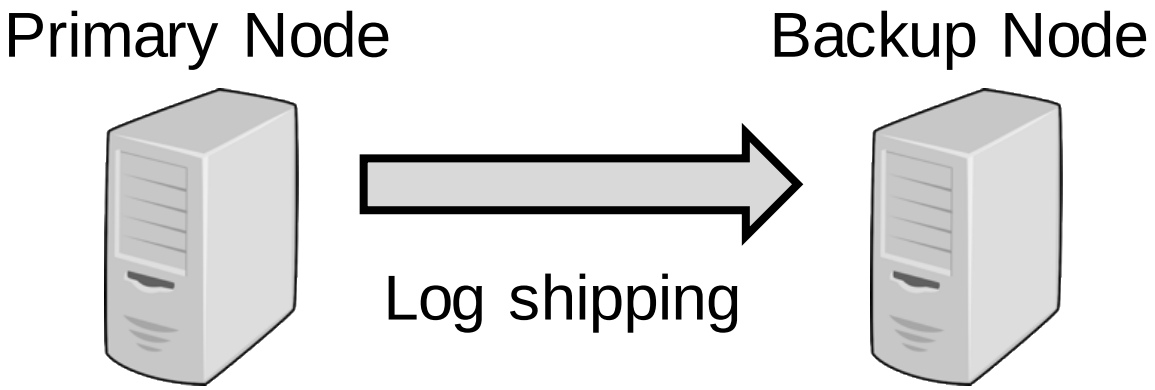
Partitioned Distributed DBMS

- Distributed concurrency control
- Distributed index
- Two-phase commit

Data replication

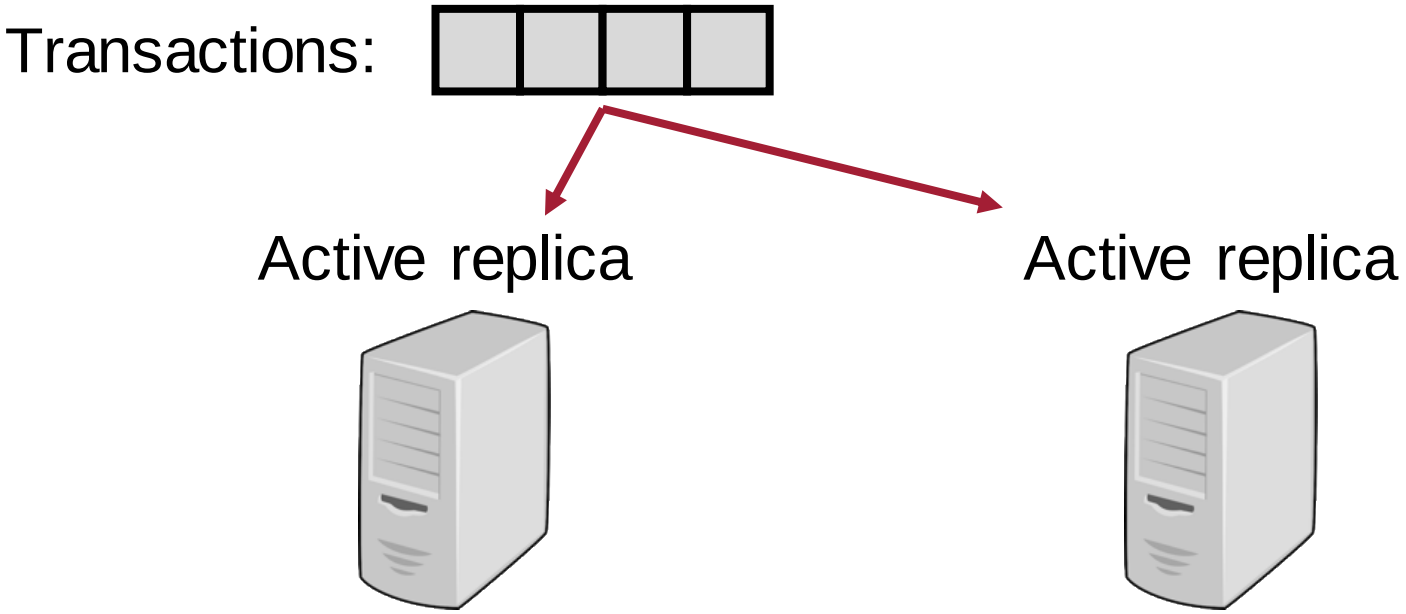
- Primary-backup (active-passive) replication
- Active-active replication

Primary Backup Replication (Active-Passive)



- The primary node ships log to the backup node
- The backup node replays the log
- If the primary node crashes, the backup node is promoted to be the new primary

Active-Active Replication



- Same sequence of transactions are sent to all the active replicas
- Each replica executes the transactions deterministically, such that all replicas produce the same results

Summary

Partitioned Distributed DBMS

- Distributed concurrency control
- Distributed index
- Two-phase commit

Data replication

- Primary-backup (active-passive) replication
- Active-active replication