
# CS 764: Topics in Database Management Systems

# Lecture 20: Two-Phase Commit (2PC)

Xiangyao Yu

11/14/2022

# Today's Paper: Distributed Transactions in R*

## Transaction Management in the R* Distributed Database Management System

C. MOHAN, B. LINDSAY, and R. OBERMARCK
IBM Almaden Research Center

This paper deals with the transaction management aspects of the R* distributed database system. It concentrates primarily on the description of the R* commit protocols, Presumed Abort (PA) and Presumed Commit (PC). PA and PC are extensions of the well-known, two-phase (2P) commit protocol. PA is optimized for read-only transactions and a class of multisite update transactions, and PC is optimized for other classes of multisite update transactions. The optimizations result in reduced intersite message traffic and log writes, and, consequently, a better response time. The paper also discusses R*'s approach toward distributed deadlock detection and resolution.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed databases*; D.4.1 [**Operating Systems**]: Process Management—*concurrency; deadlocks; synchronization*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*; H.2.0 [**Database Management**]: General—*concurrency control*; H.2.2 [**Database Management**]: Physical Design—*recovery and restart*; H.2.4 [**Database Management**]: Systems—*distributed systems; transaction processing*; H.2.7 [**Database Management**]: Database Administration—*logging and recovery*

General Terms: Algorithms, Design, Reliability

Additional Key Words and Phrases: Commit protocols, deadlock victim selection

### 1. INTRODUCTION

R* is an experimental, distributed database management system (DDBMS) developed and operational at the IBM San Jose Research Laboratory (now renamed the IBM Almaden Research Center) [18, 20]. In a distributed database system, the actions of a transaction (an atomic unit of consistency and recovery [13]) may occur at more than one site. Our model of a transaction, unlike that of some other researchers' [25, 28], permits multiple data manipulation and definition statements to constitute a single transaction. When a transaction

**ACM Trans. Database Syst. 1986.**

# Announcement

Updated schedule for future lectures

**Next lecture**: Cornus (optimized 2PC in cloud)

**Last lecture**: GPU databases

# Agenda

Two-phase commit

Presumed abort (PA)

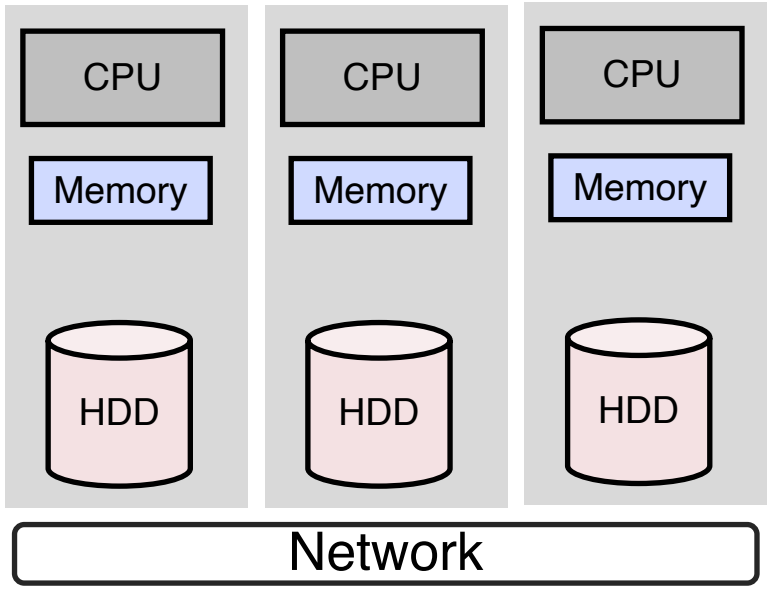
Presumed Commit (PC)

# Distributed Transactions

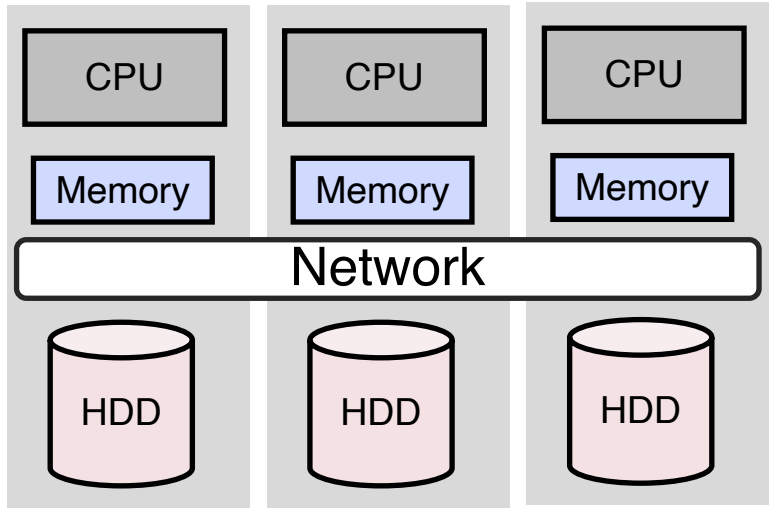
Architectures: shared-nothing vs. shared-disk

Data is partitioned and stored in each server

A distributed transaction accesses data across multiple partitions




Shared Nothing



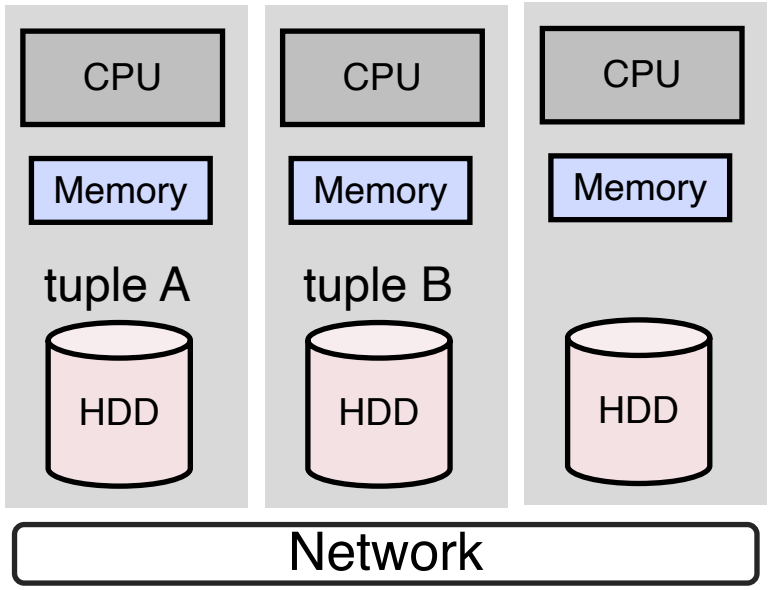

Shared Disk

# Distributed Transactions

Architectures: shared-nothing vs. shared-disk

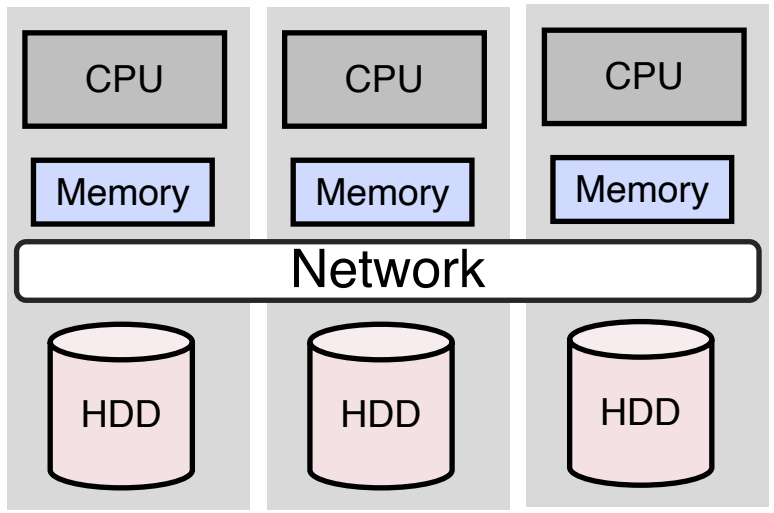
Data is partitioned and stored in each server

A distributed transaction accesses data across multiple partitions




Shared Nothing

Transaction T:
write(A)
write(B)




Shared Disk

# Atomic Commit Protocol (ACP)

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

Example:

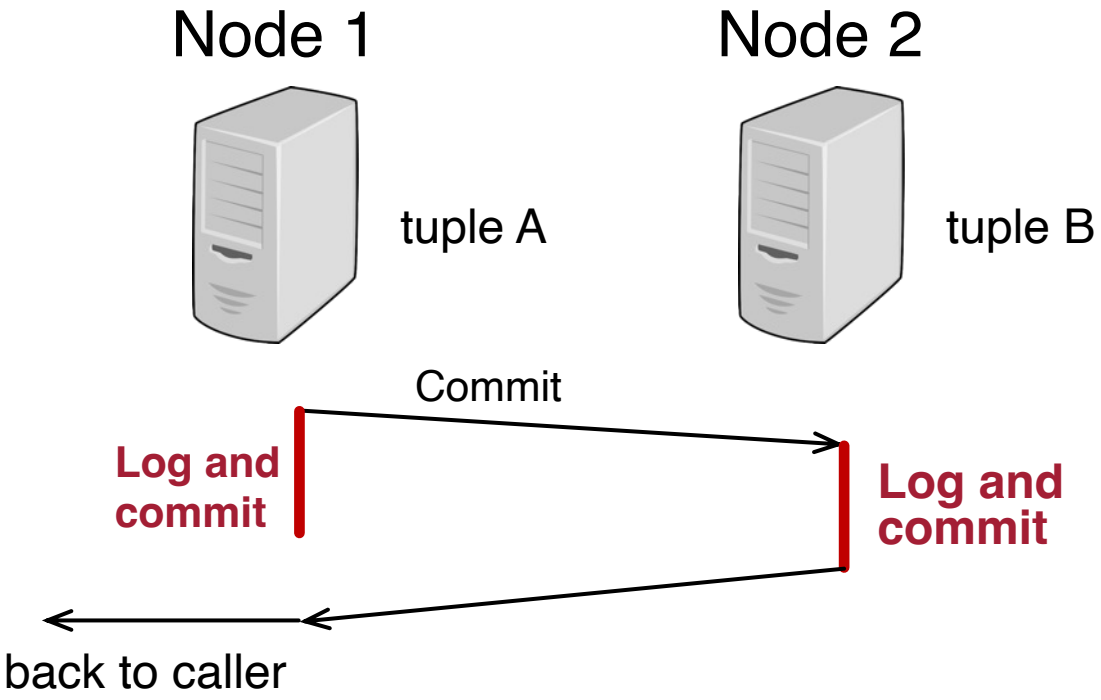
tuple A

tuple B

Transaction T:
write(A)
write(B)

The two updates must commit or abort atomically

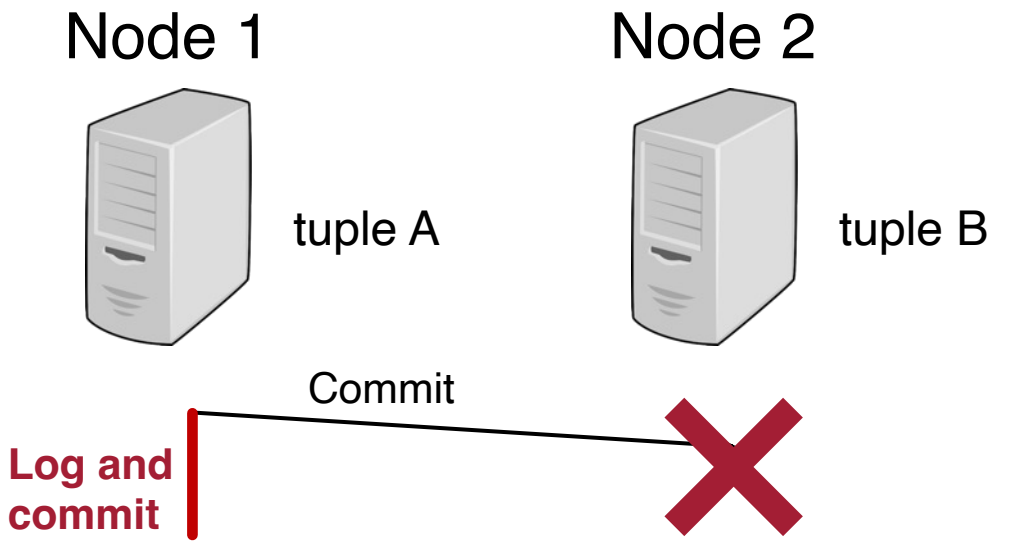
# The Challenge of Atomic Commit




Transaction T:
write(A)
write(B)

A naïve approach: all nodes log and commit independently

# The Challenge of Atomic Commit



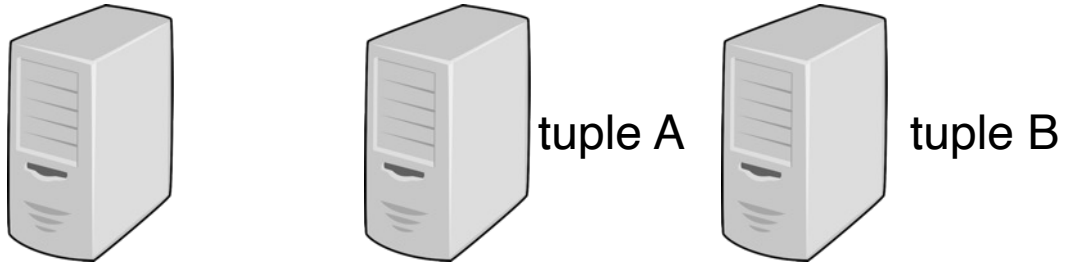

Transaction T:
write(A)
write(B)

A naïve approach: all nodes log and commit independently

Node 2 crashes before logging

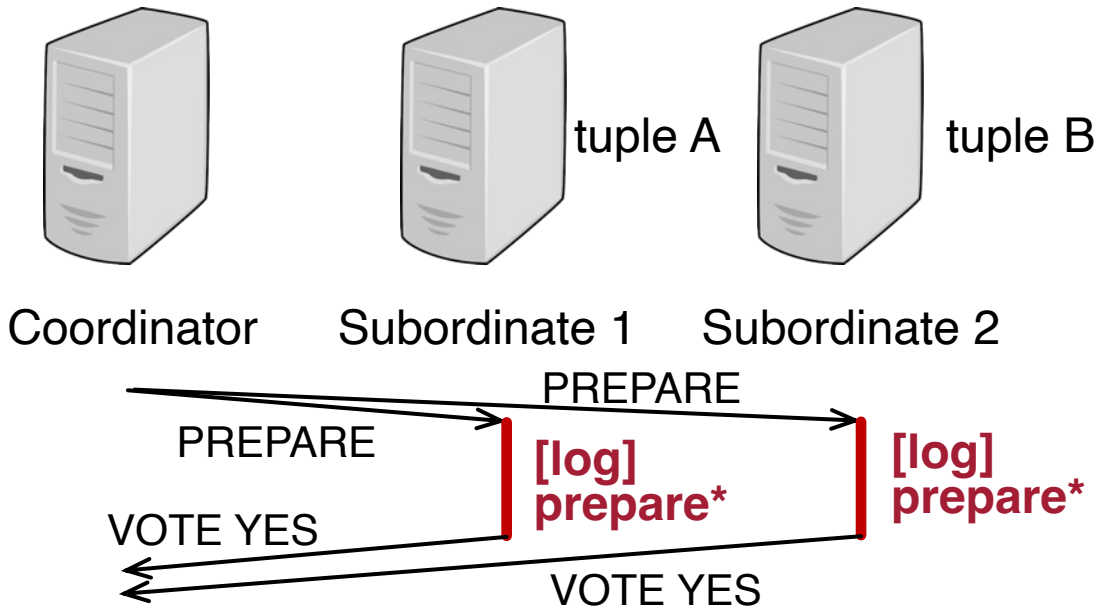
- Transaction T commits in node 1 but not in node 2

# Two-Phase Commit (2PC)




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

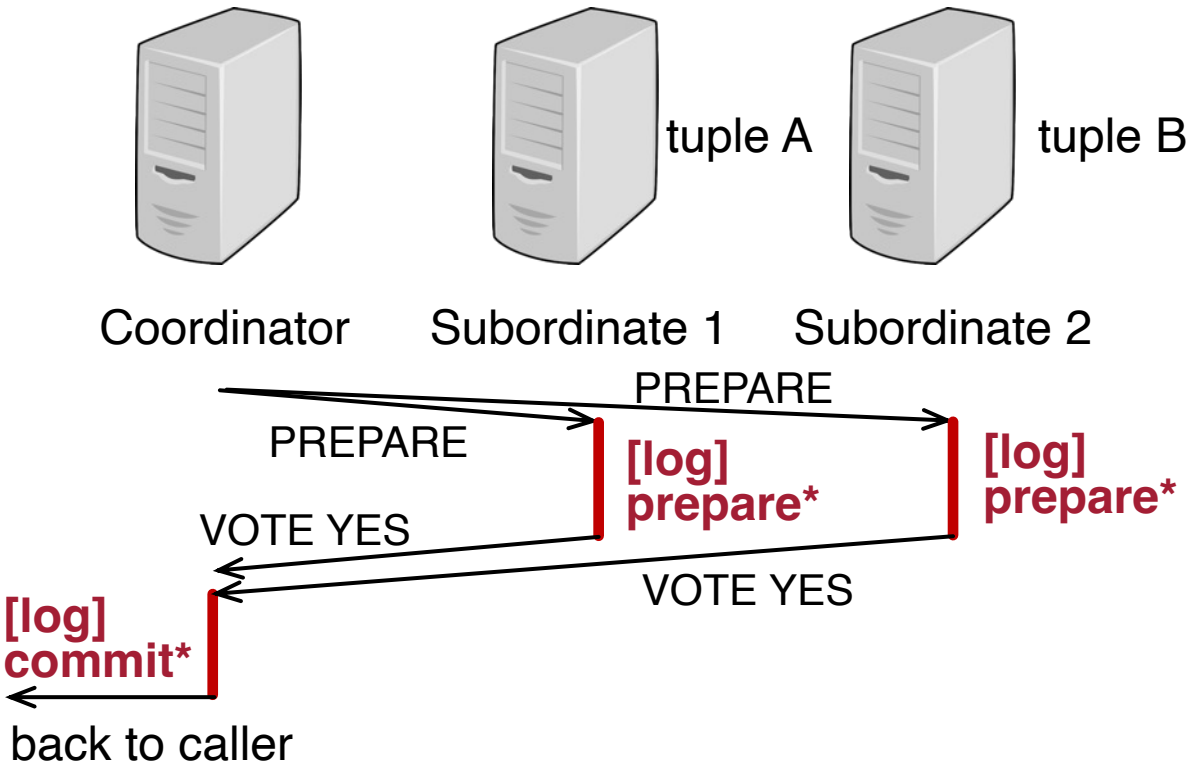
# Two-Phase Commit (2PC)




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

Phase 1: prepare phase

# Two-Phase Commit (2PC)

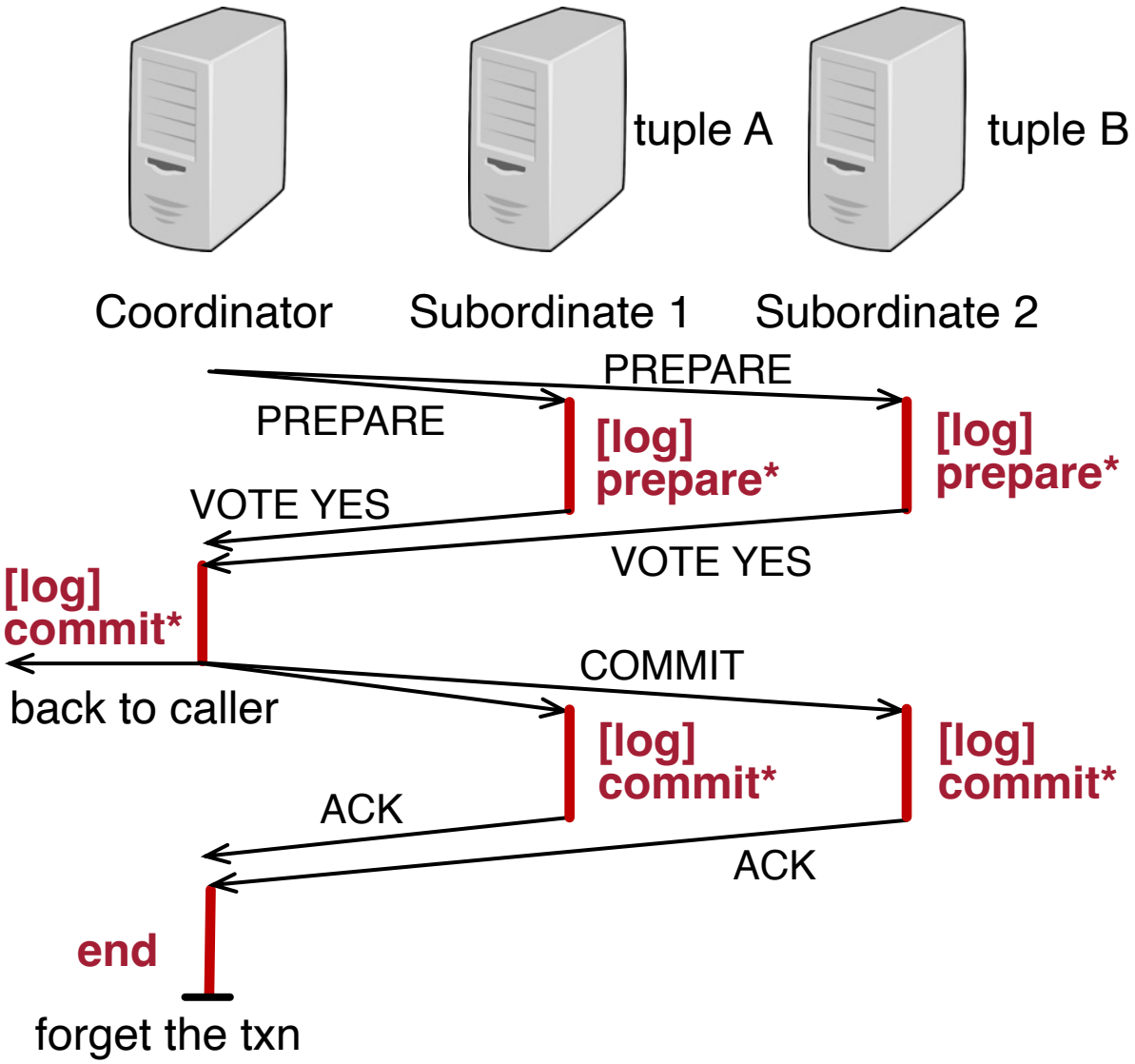



Key idea: let the coordinator log the final commit/abort decision

Phase 1: prepare phase
Phase 2: commit phase

- Coordinator logs the decision

# Two-Phase Commit (2PC)

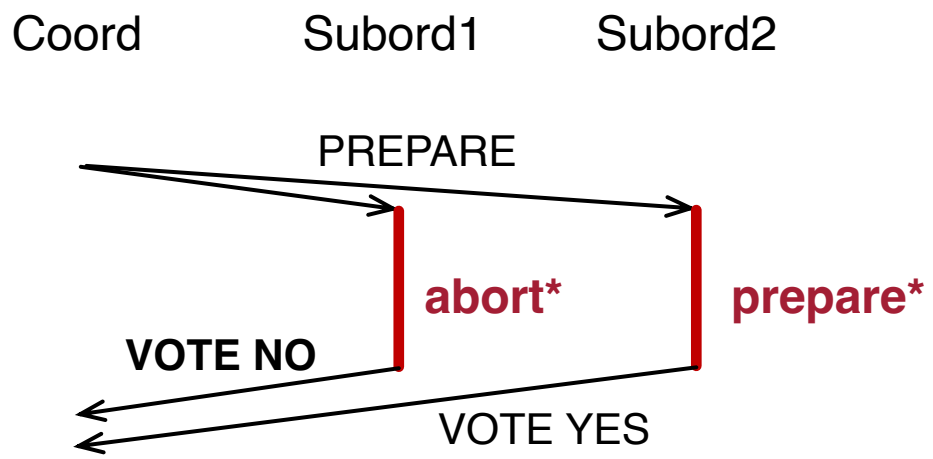



Key idea: let the coordinator log the final commit/abort decision

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

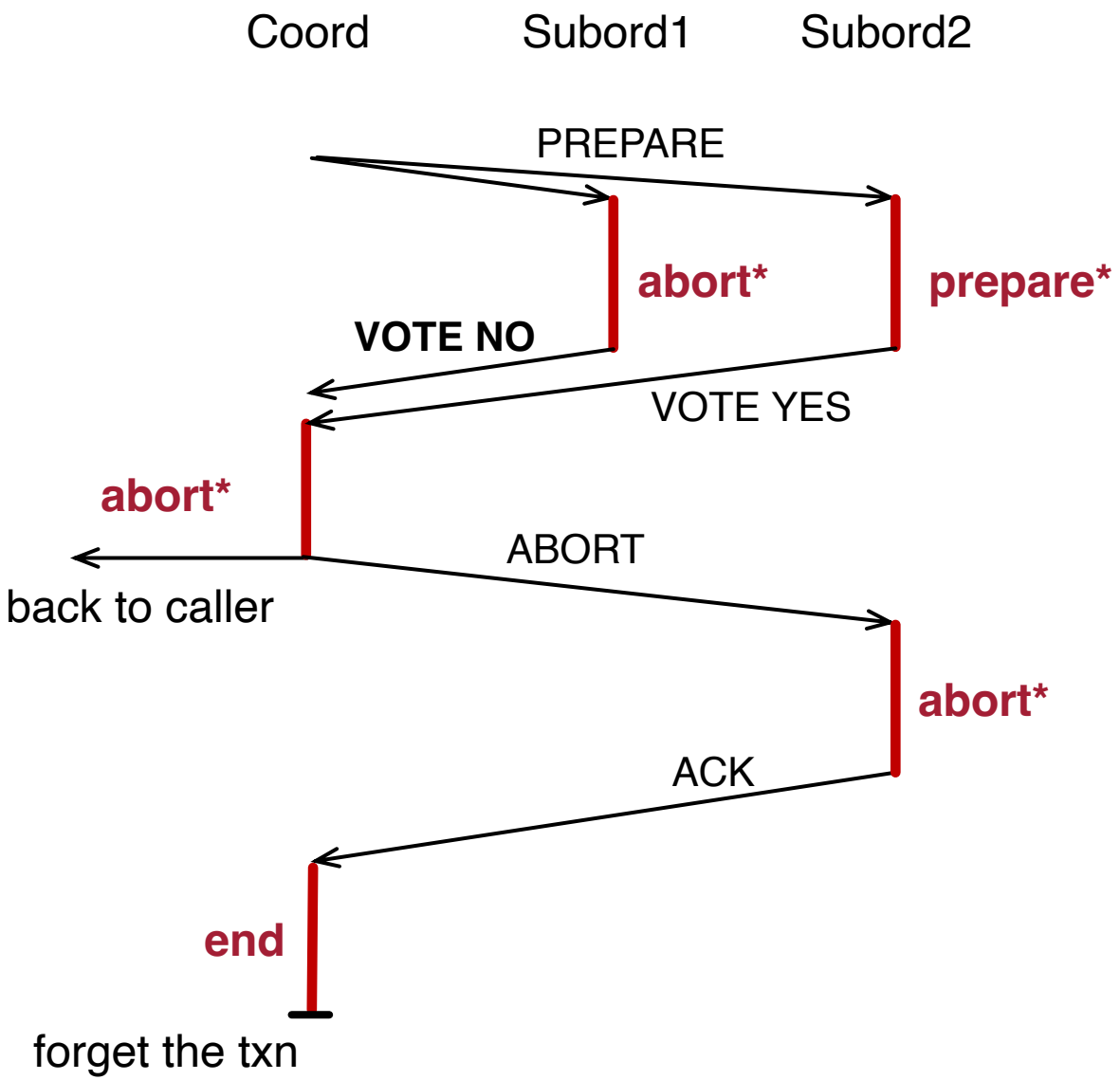
# 2PC – Abort Example




Subordinate returns VOTE NO if the transaction is aborted

- Subordinate can release locks and forget the transaction

# 2PC – Abort Example

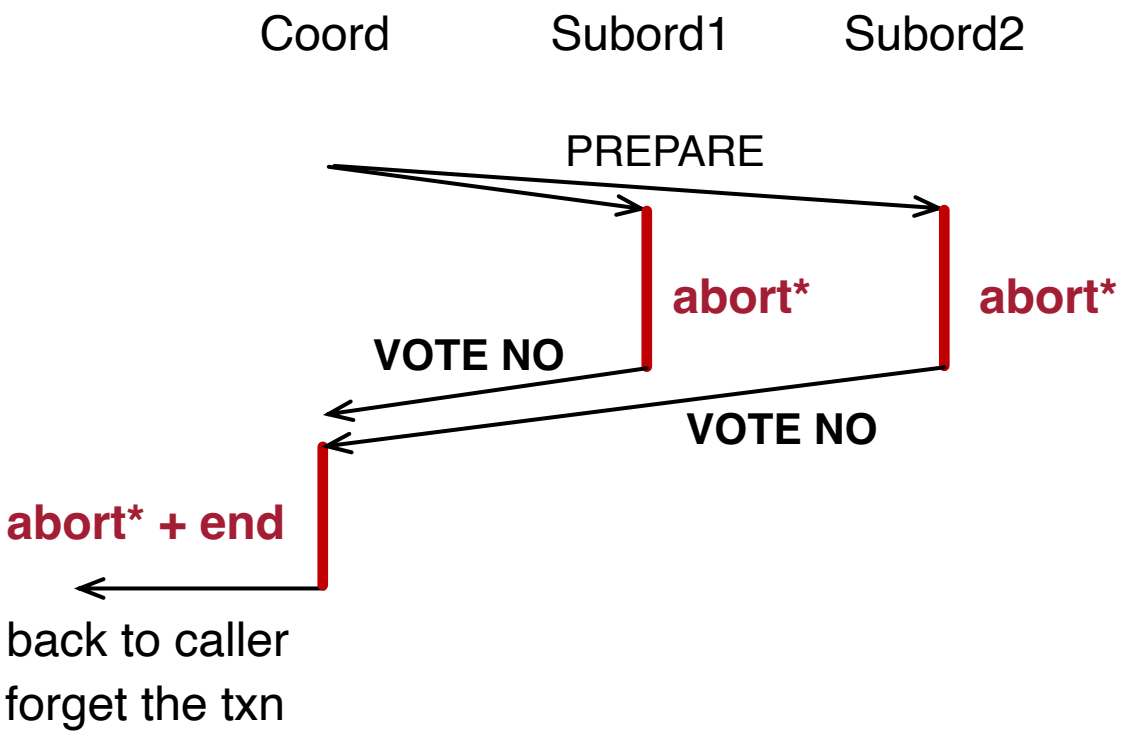



Subordinate returns VOTE NO if the transaction is aborted

- Subordinate can release locks and forget the transaction

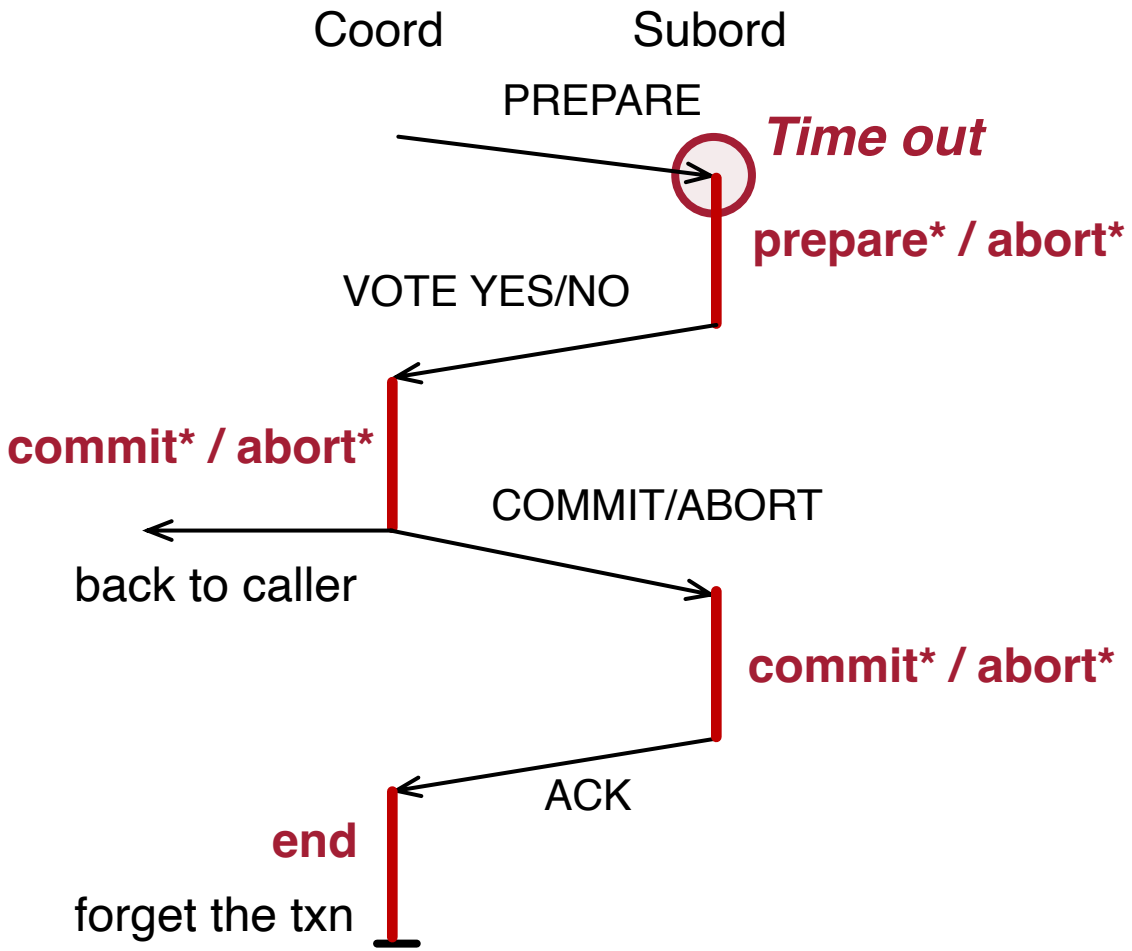
Skip the commit phase for aborted subordinates

# 2PC – All Subordinates Abort




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

# 2PC – Failures

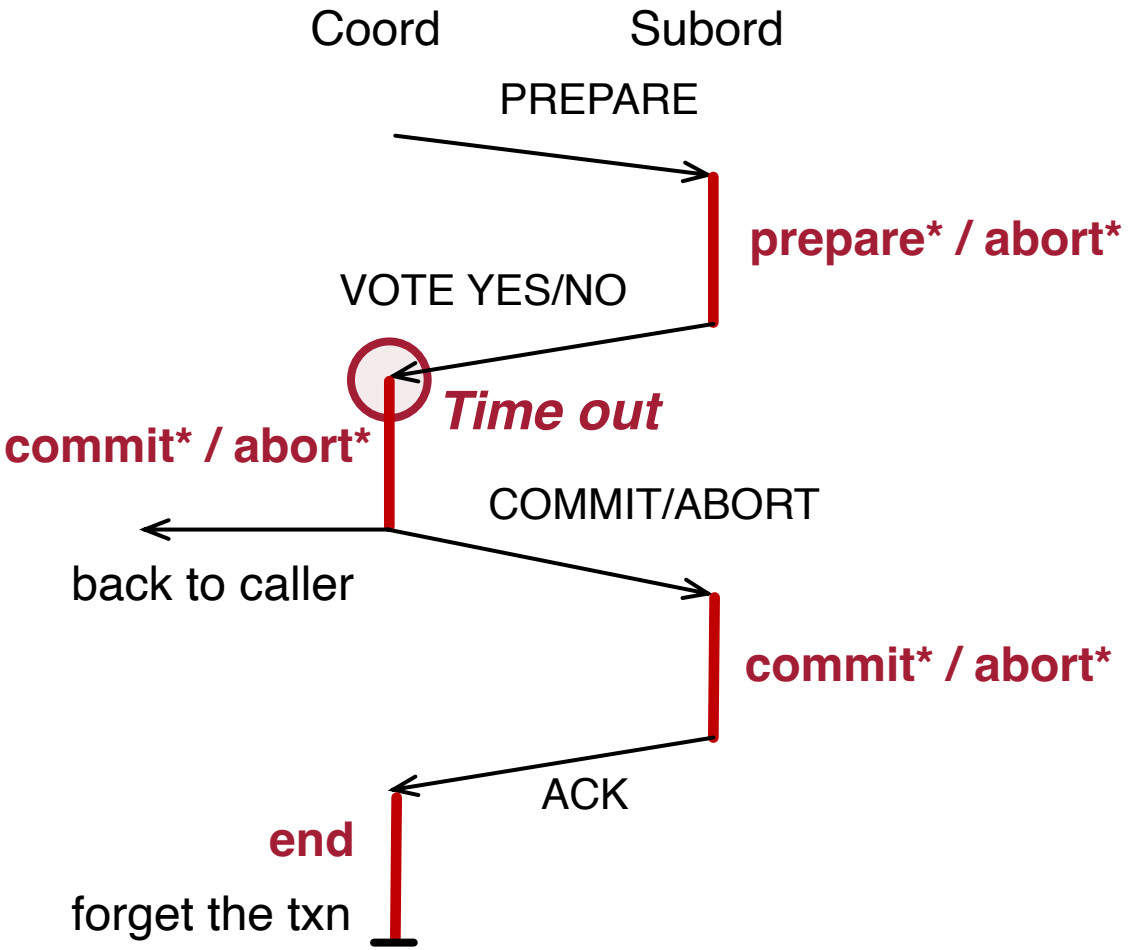



Use timeout to detect failures

Subordinate timeout

- Waiting for PREPARE: self abort

# 2PC – Failures

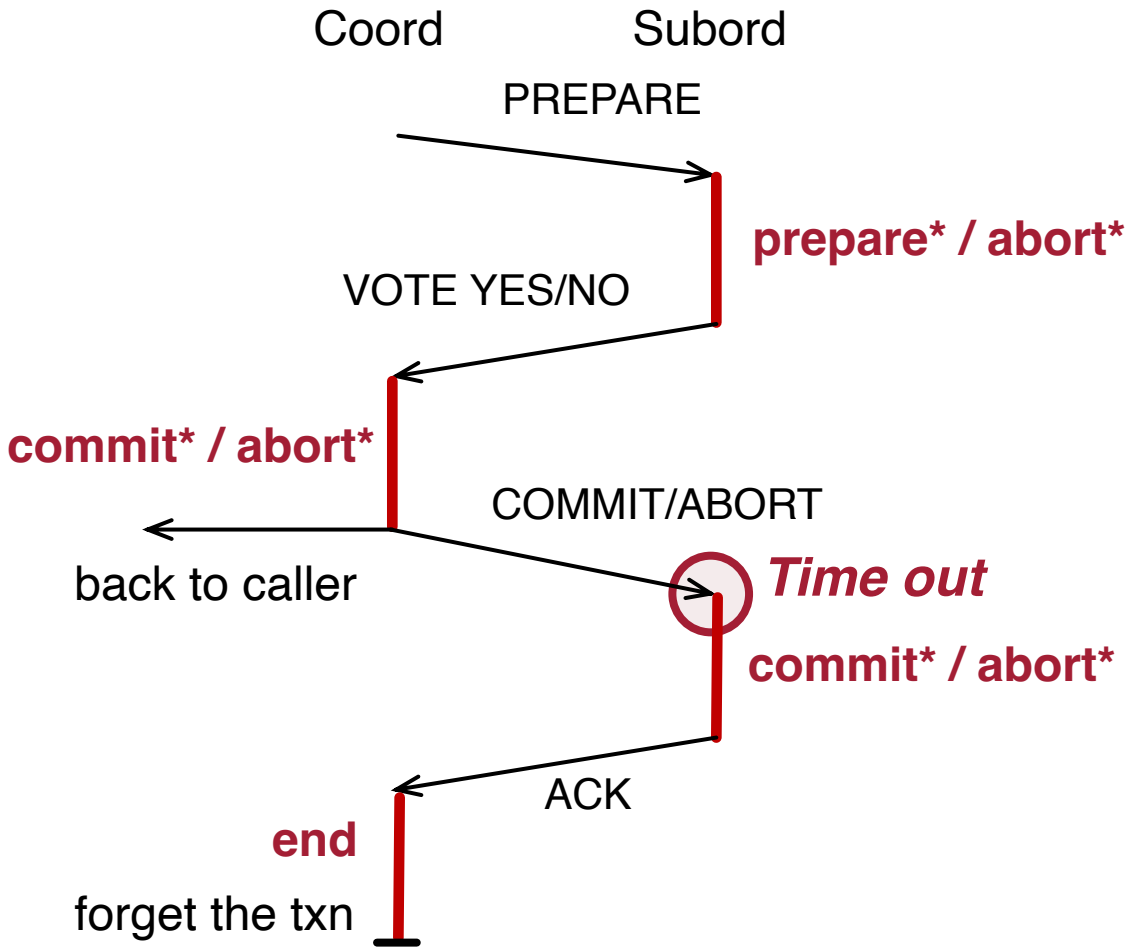



Use timeout to detect failures

Coordinator timeout

- Waiting for vote: self abort

# 2PC – Failures

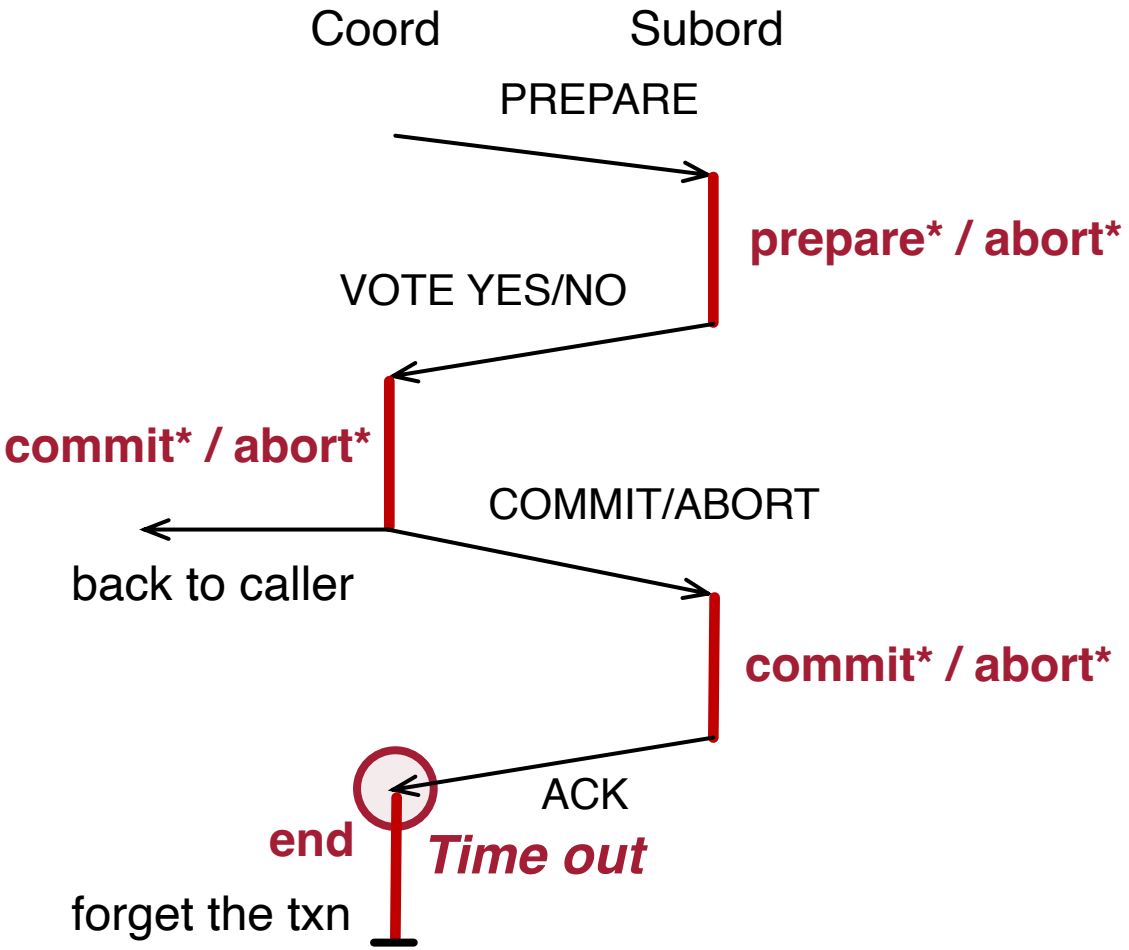



Use timeout to detect failures

Subordinate timeout

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

# 2PC – Failures

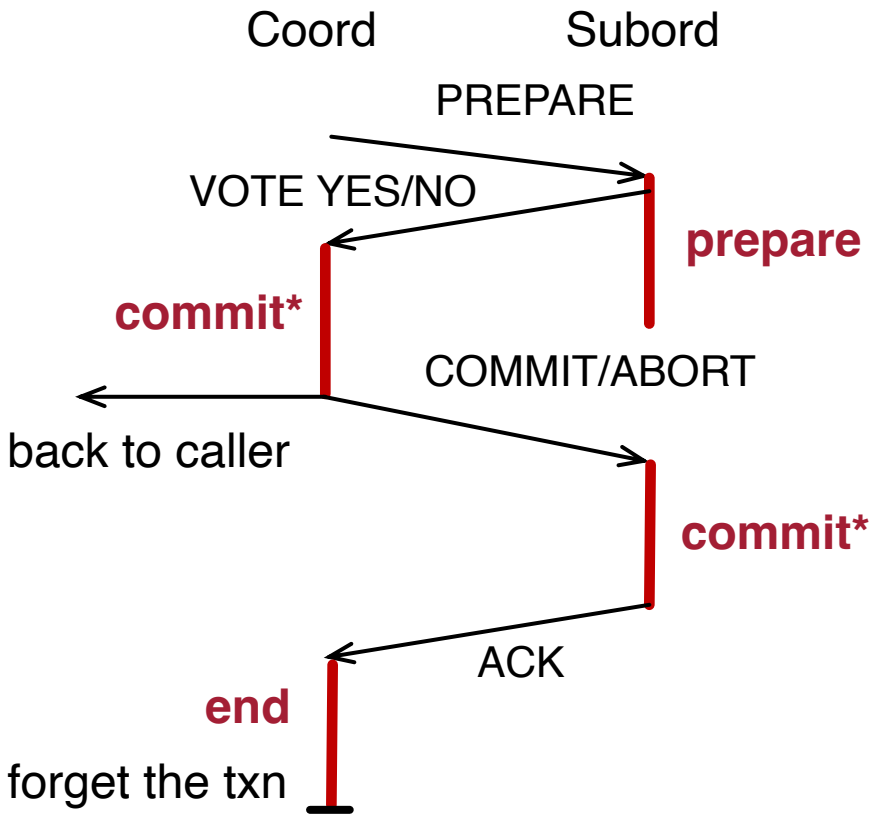



Use timeout to detect failures

Coordinator timeout

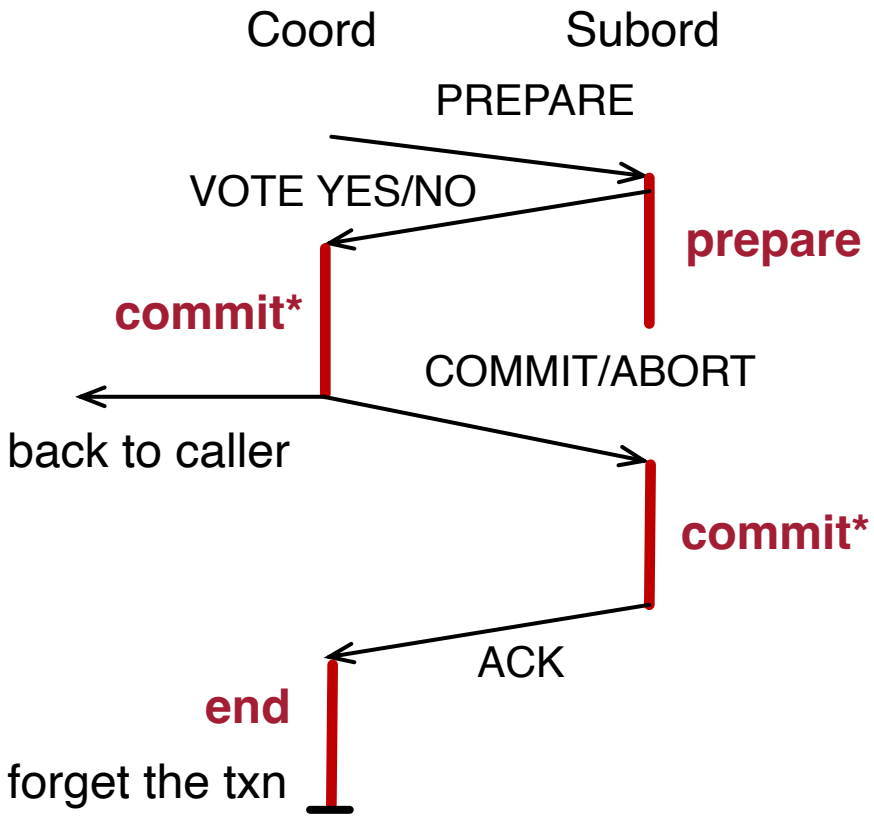
- Waiting for ACK: contact subordinates

# 2PC – Alternative Designs?




Subordinate returns vote to coordinator before logging prepare?

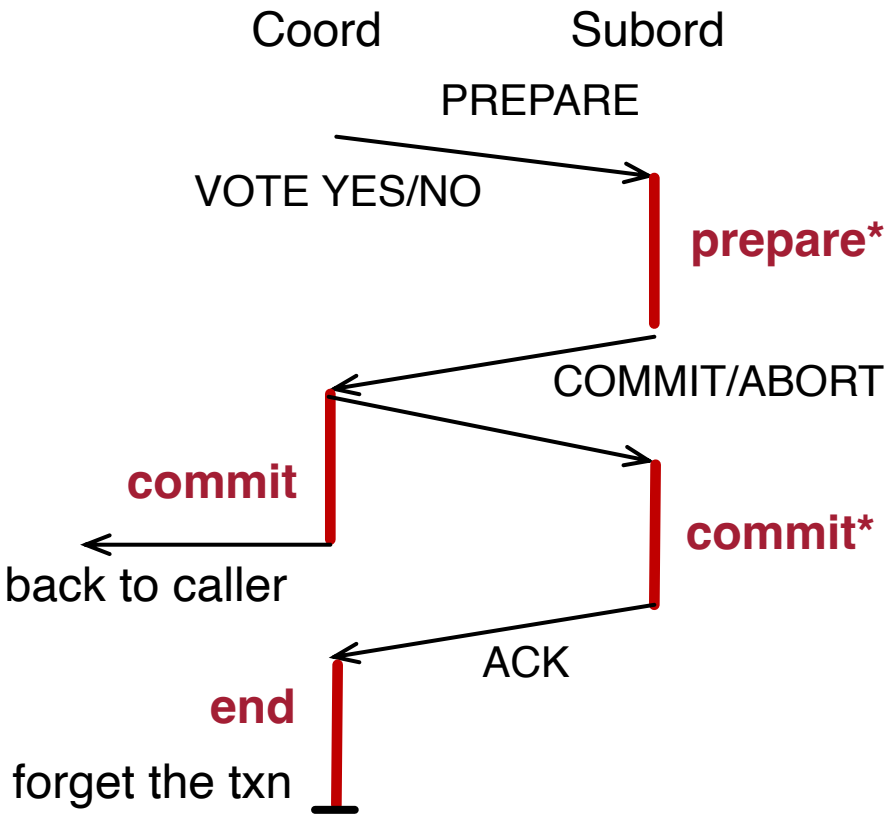
# 2PC – Alternative Designs?




Subordinate returns vote to coordinator before logging prepare?

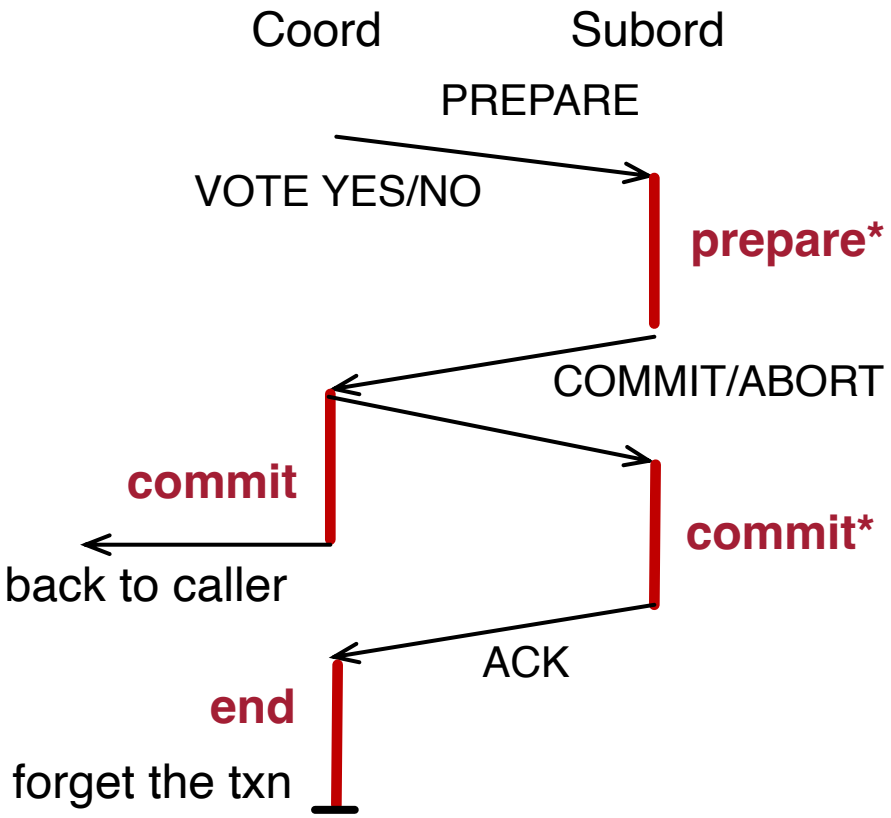
**Problem**: subordinate may crash before the log record is written to disk. The log record is thus lost but the coordinator already committed the transaction

# 2PC – Alternative Designs?




Coordinator sends decision to subordinates before logging the decision?

# 2PC – Alternative Designs?



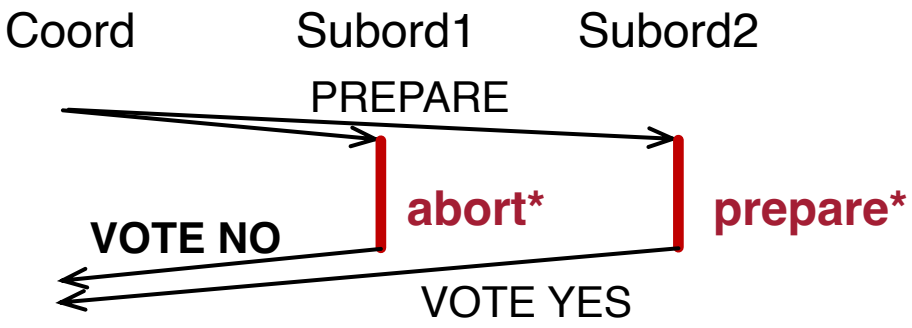

Coordinator sends decision to subordinates before logging the decision?

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

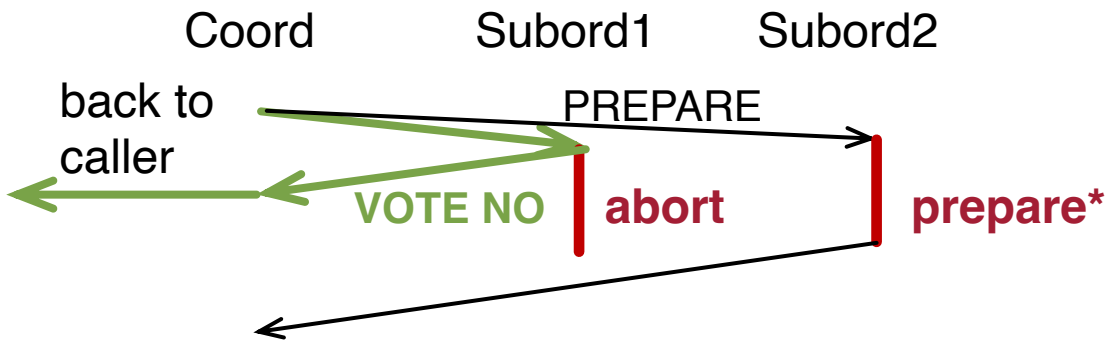
# Optimization 1: Presumed Abort (PA)

**Observation**: It is safe for a coordinator to “forget” a transaction immediately after it makes the decision to abort it and to write an abort record

# PA: Aborted Transaction



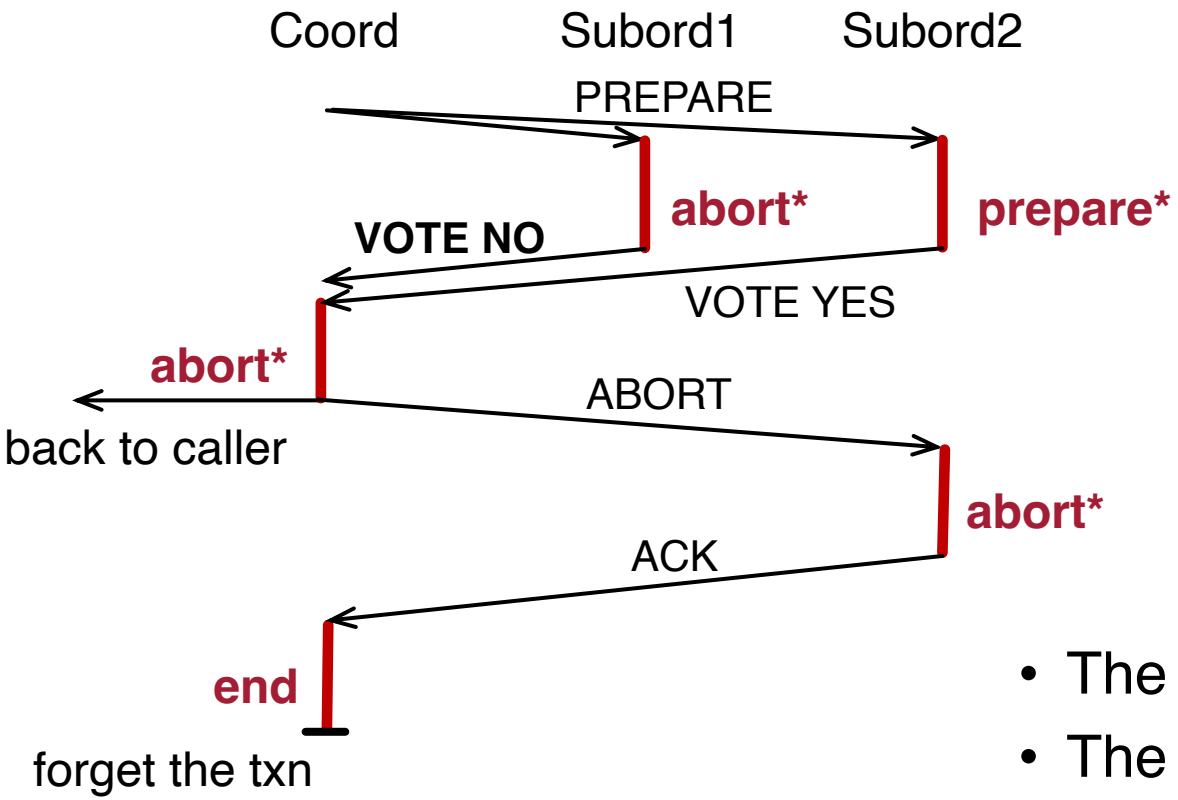

**Standard 2PC**



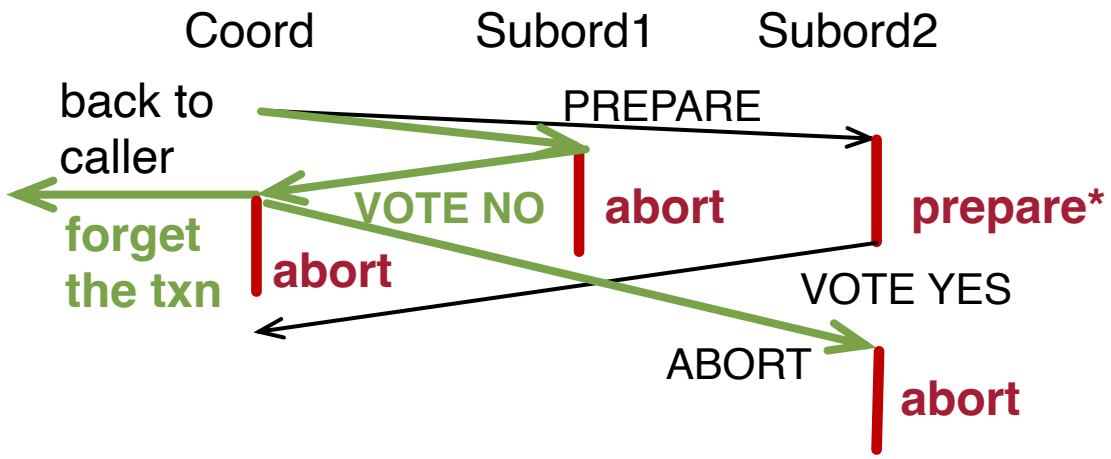

**Presumed Abort**

- The abort record is not forced in subordinate

# PA: Aborted Transaction



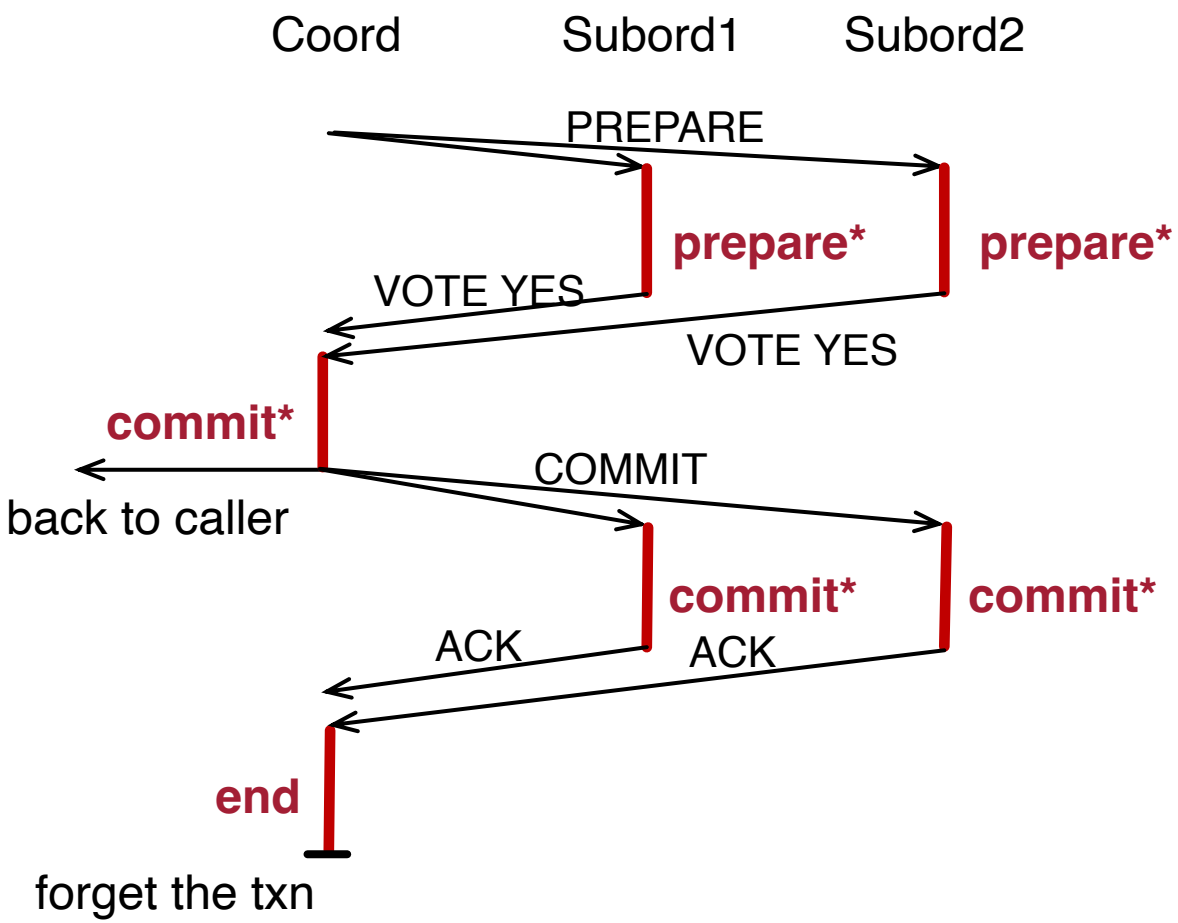

**Standard 2PC**

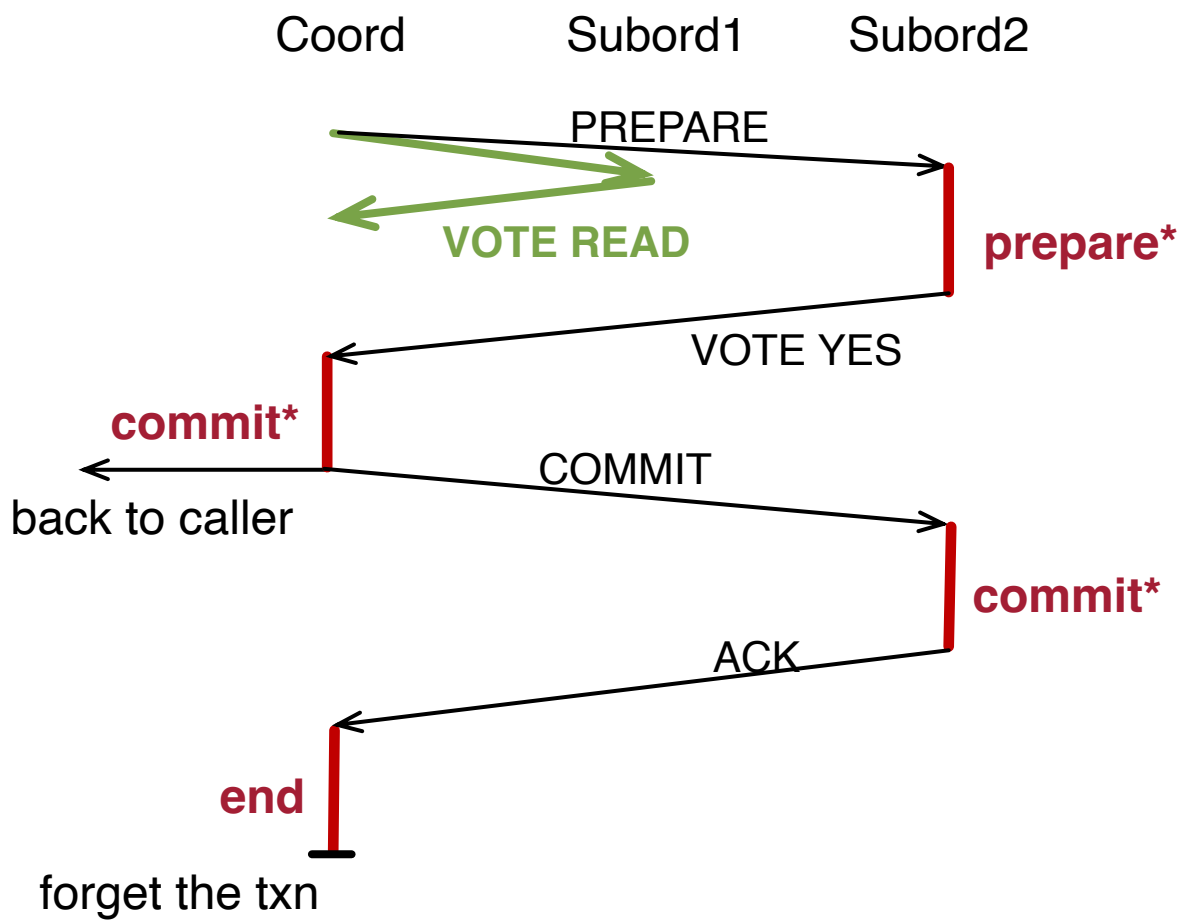



**Presumed Abort**

- The abort record is not forced in subordinate
- The abort record is not forced in coordinator
- Coordinator forgets the transaction early
- No ACK for aborts
- **Behavior of committed transactions unchanged**

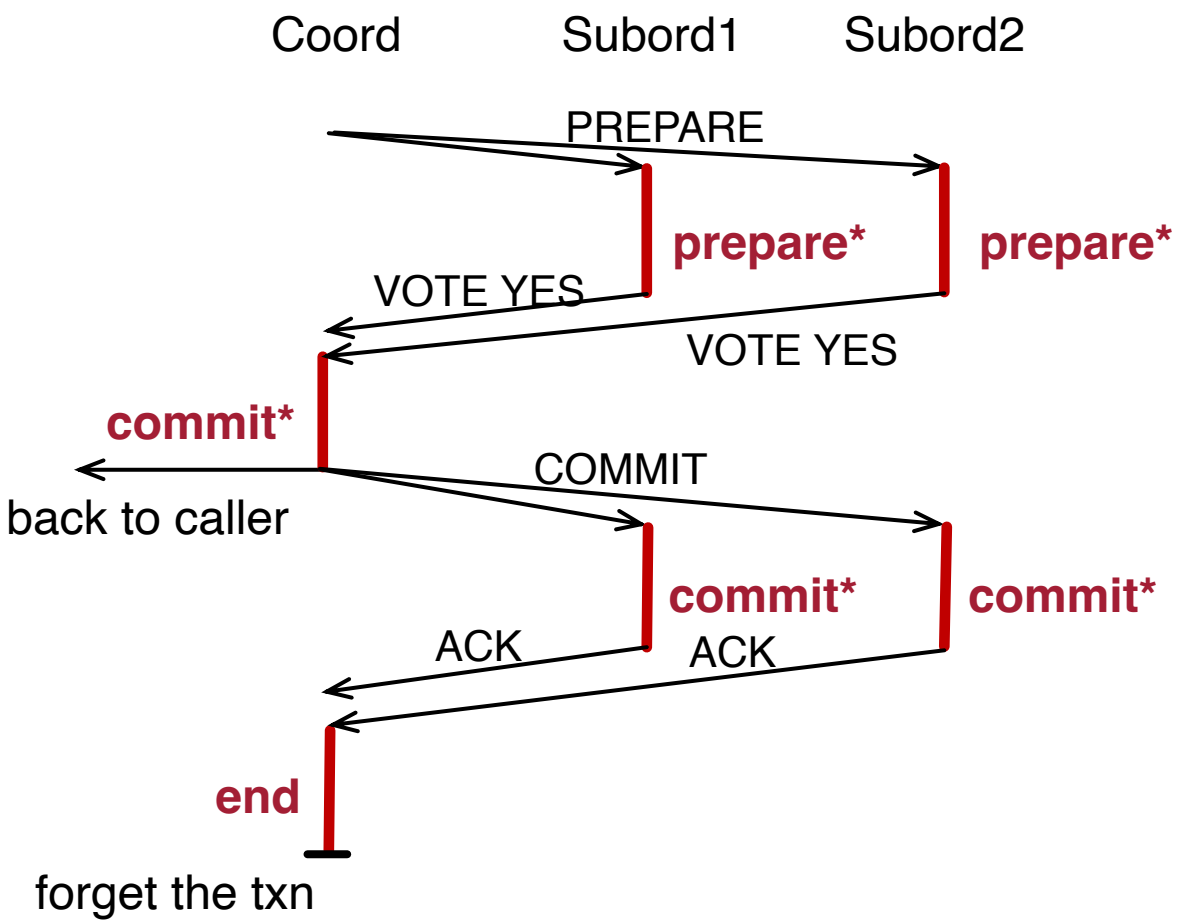
# PA: Partially Readonly Transactions

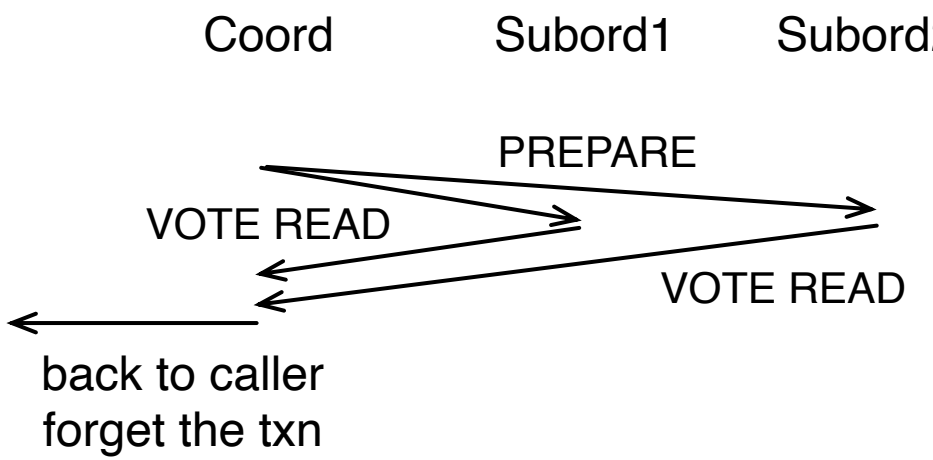



Readonly subordinate does not log in prepare phase and skips commit phase

# PA: Completely Readonly Transactions



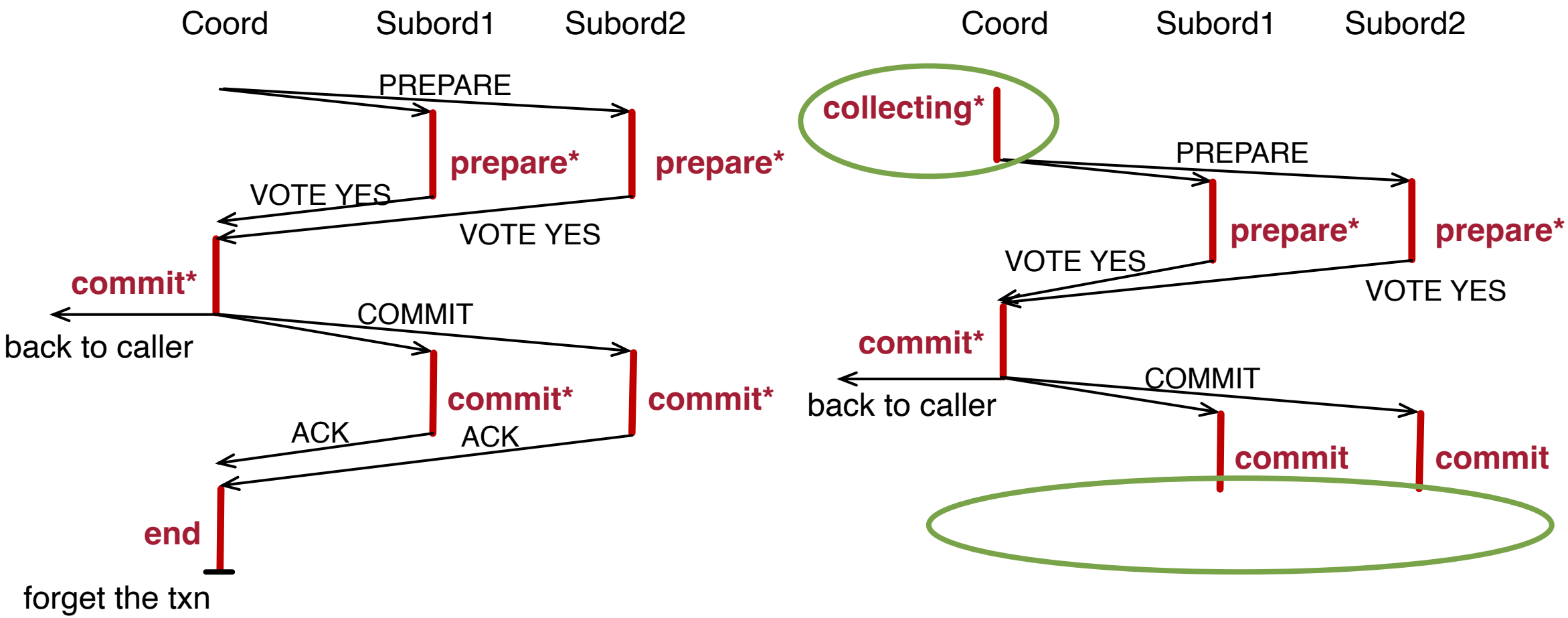




Completely readonly transactions skip the commit phase entirely

# Optimization 2: Presumed Commit (PC)

Since most transactions are expected to commit, can we make commits cheaper by eliminating the ACKs for COMMITS?

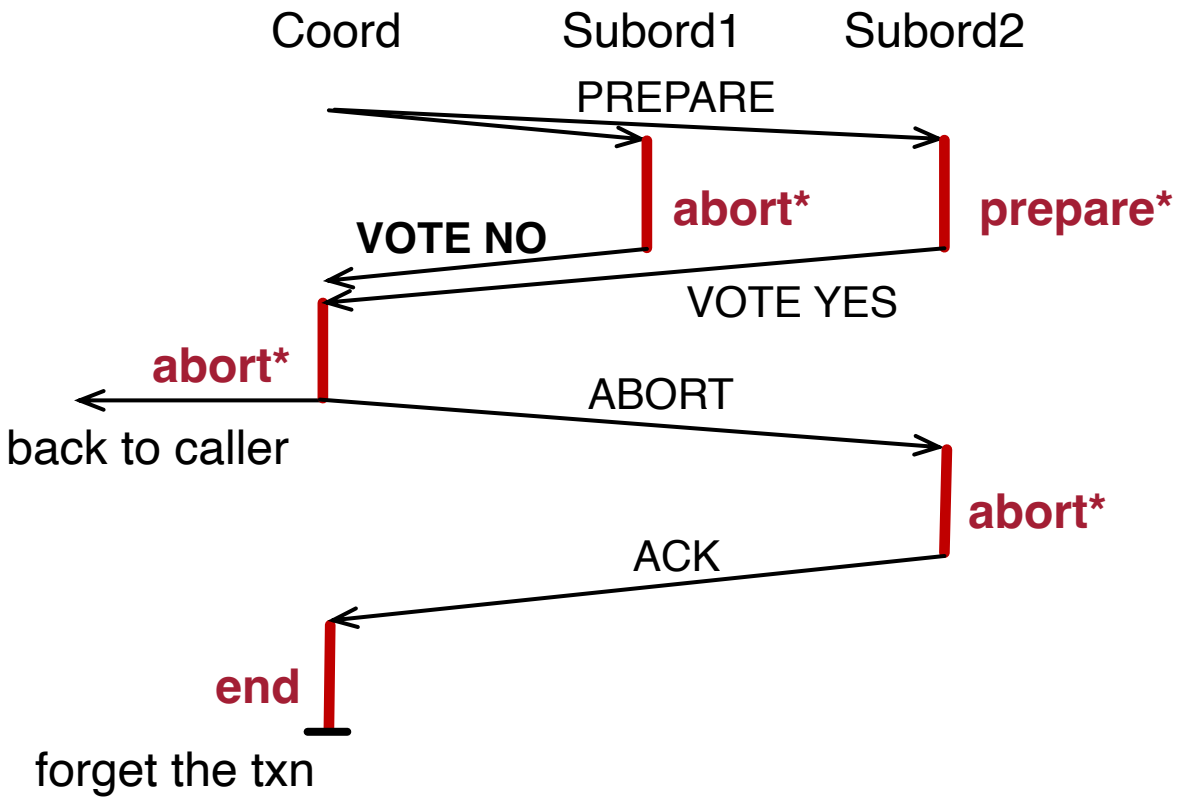
# PC: Committed Transaction

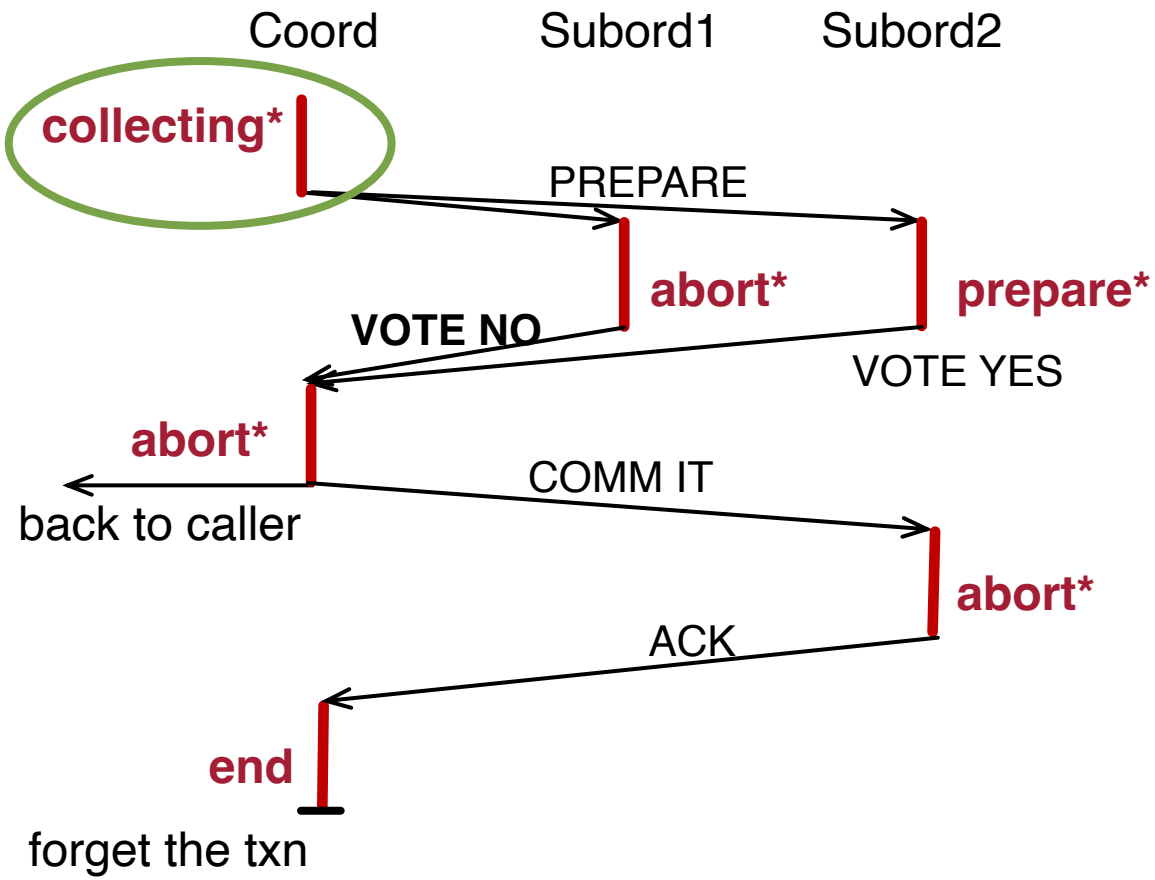



Need to force log **collecting** due to potential abort of coordinator

No need to send ACK for COMMITS

# PC: Aborted Transaction




Abort behavior is similar to standard 2PC but requires logging **collecting**

# Summary

| Process Type / Protocol Type | Coordinator | | | Subordinate | |
|---|---|---|---|---|---|
| | U<br>Yes US | U<br>No US | R | US | RS |
| Standard 2P | 2,1,-,2 | - | - | 2,2,2 | - |
| Presumed Abort | 2,1,1,2 | 1,1,1 | 0,0,1 | 2,2,2 | 0,0,1 |
| Presumed Commit | 2,2,1,2 | 2,2,1 | 2,1,1 | 2,1,1 | 0,0,1 |

U - Update Transaction

R - Read-Only Transaction

RS - Read-Only Subordinate

US - Update Subordinate

m,n,o,p - m Records Written, n of Them Forced

o For a Coordinator: # of Messages Sent to Each RS

For a Subordinate: # of Messages Sent to Coordinator

p # of Messages Sent to Each US

Presumed Abort (PA) is better than standard 2PC (widely used in practice)

Presumed Commit (PC) is worse than PA in most cases

# Conclusions

Distributed transaction requires an atomic commit protocol

**Two-phase commit** (2PC) is the most widely used atomic commit protocol

- Standard 2PC
- Optimization 1: presumed abort (PA) — most commonly used in practice
- Optimization 2: presumed commit (PC)

# Q/A – Two Phase Commit

More performant alternatives to 2PC?

Transactions in today's distributed DBMS?

2PC in replicated and non-replicated data systems?

Distributed deadlocks possible in shared-nothing database?

Is coordinator a single point of failure?

What if a long-running txn fails before reaching commit or abort?

Cope with message lost during network transmission?

2PC vs. Paxos?

# Next Lecture

Zhihan Guo, et al., Cornus: Atomic Commit for a Cloud DBMS with Storage Disaggregation. arXiv 2102.10185 (to appear in VLDB), 2022