CS 764: Topics in Database Management Systems

Lecture 22: Two-Phase Commit (2PC)

Xiangyao Yu

11/18/2025

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.

Cornus: Atomic Commit for a Cloud DBMS with Storage Disaggregation

Zhihan Guo
Xinyu Zeng
Kan Wu
University of Wisconsin-Madison
{zhihan,xinyu,kanwu}@cs.wisc.edu

Wuh-Chwen Hwang
Ziwei Ren
Xiangyao Yu
University of Wisconsin-Madison
{wuh-chwen,ziwei,yxy}@cs.wisc.edu

Mahesh Balakrishnan
Confluent, Inc.
mbalakrishnan@confluent.io

Philip A. Bernstein
Microsoft Research
philbe@microsoft.com

ABSTRACT

Two-phase commit (2PC) is widely used in distributed databases to ensure atomicity of distributed transactions. Conventional 2PC was originally designed for the shared-nothing architecture and has two limitations: *long latency* due to two eager log writes on the critical path, and *blocking* of progress when a coordinator fails.

Modern cloud-native databases are moving to a storage disaggregation architecture where storage is a shared highly-available service. Our key observation is that disaggregated storage enables protocol innovations that can address both the long-latency and blocking problems. We develop Cornus, an optimized 2PC protocol to achieve this goal. The only extra functionality Cornus requires is an atomic compare-and-swap capability in the storage layer, which many existing storage services already support. We present Cornus in detail and show how it addresses the two limitations. We also deploy it on real storage services including Azure Blob Storage and Redis. Empirical evaluations show that Cornus can achieve up to 1.9× latency reduction over conventional 2PC.

PVLDB Reference Format:
Zhihan Guo, Xinyu Zeng, Kan Wu, Wuh-Chwen Hwang, Ziwei Ren, Xiangyao Yu, Mahesh Balakrishnan, Philip A. Bernstein. Cornus: Atomic Commit for a Cloud DBMS with Storage Disaggregation. PVLDB, 16(2): 379 - 392, 2022.
doi:10.14778/3565816.3565837

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CloudOLTP/Cornus.

1 INTRODUCTION

Databases are migrating to the cloud because of desirable features such as elasticity, high availability, and cost competitiveness. Modern cloud-native databases feature a *storage-disaggregation* architecture where the storage is decoupled from computation as a standalone service as shown in Figure 1b. This architecture allows independent scaling and billing of computation and storage, which can improve resource utilization, reduce operational cost, and enable flexible cloud deployment with heterogeneous configurations. Many cloud-native database systems adopt such an architecture for both OLTP [21, 49, 62, 67] and OLAP [14–16, 23, 30, 60]. Nowadays, as storage services offer essential functions such as fault tolerance, scalability, and security at low-cost, systems start to layer their designs on the existing disaggregated storage services [22, 26].

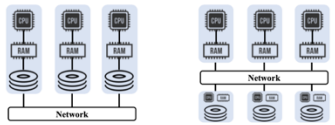


Figure 1: Shared-Nothing vs. Storage-Disaggregation.

This paper focuses on efficient deployment of the two-phase commit protocol on existing storage services. Two-phase commit (2PC) is the most widely used atomic commit protocol, which ensures that distributed transactions commit in either all or none of the involved data partitions. 2PC was originally designed for the *shared-nothing* architecture and suffers from two major problems. The first is *long latency*: 2PC requires two round-trip network messages and associated logging operations. Previous work has demonstrated that the majority of a transaction's execution time can be attributed to 2PC [19, 20, 32, 42, 50, 52, 64]. The second problem is *blocking* [24, 25, 53]. Blocking occurs if a coordinator crashes before notifying participants of the final decision. These two problems greatly limit the performance of 2PC, especially in a storage disaggregation architecture

Various techniques have been proposed to address these two problems with 2PC. Some proposed optimizations target the shared-nothing architecture and do not solve both problems simultaneously. These protocols either reduce latency by making strong assumptions about the workload and/or system that are not always practical for disaggregated storage [18–20, 25, 45, 46, 55, 56], or they mitigate the blocking problem by adding an extra phase and

Proceedings of the VLDB Endowment, Vol. 16, No. 2 ISSN 2150-8097.
doi:10.14778/3565816.3565837

VLDB 2022

Agenda

Two-phase commit

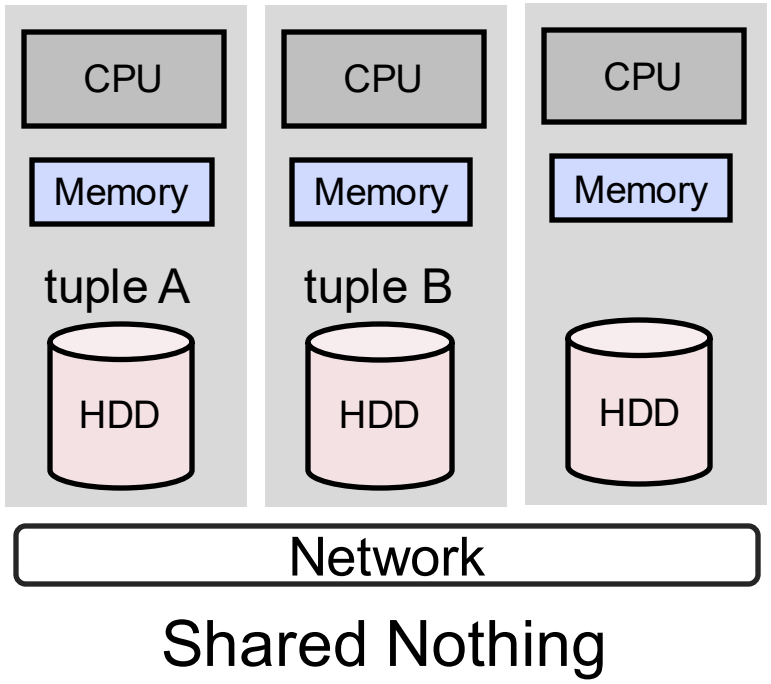
- Presumed abort (PA)
- Presumed Commit (PC)

Cornus: 2PC optimized for storage disaggregation

Distributed Transactions

Data is partitioned and stored in each server

A distributed transaction accesses data across multiple partitions



Shared Nothing

Transaction T:
write(A)
write(B)

Atomic Commit Protocol (ACP)

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

Example:

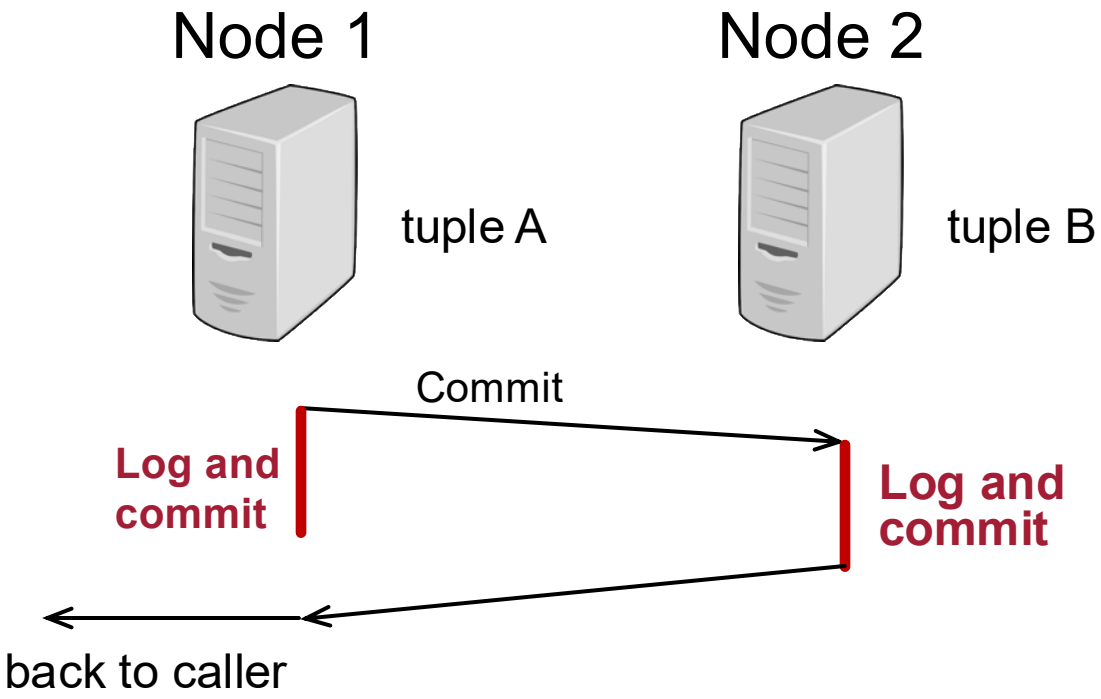
tuple A

tuple B

Transaction T:
write(A)
write(B)

The two updates must commit or abort atomically

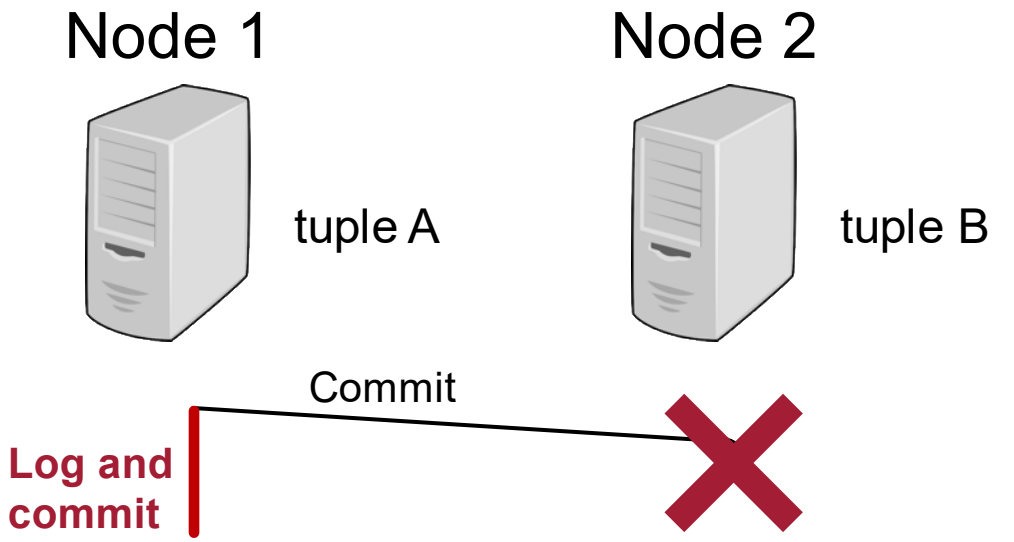
The Challenge of Atomic Commit



Transaction T:
write(A)
write(B)

A naïve approach: all nodes log and commit independently

The Challenge of Atomic Commit



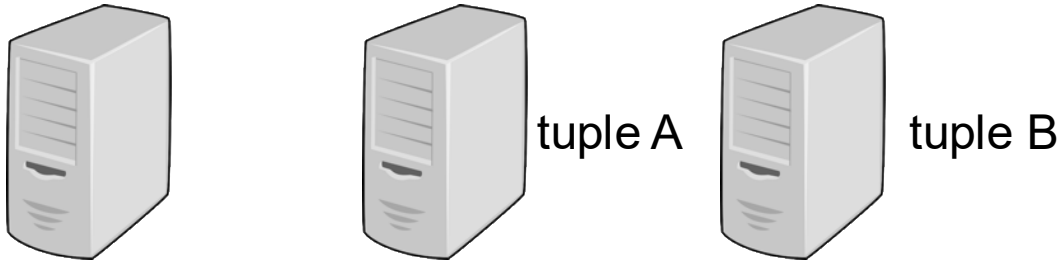
Transaction T:
write(A)
write(B)

A naïve approach: all nodes log and commit independently

Node 2 crashes before logging

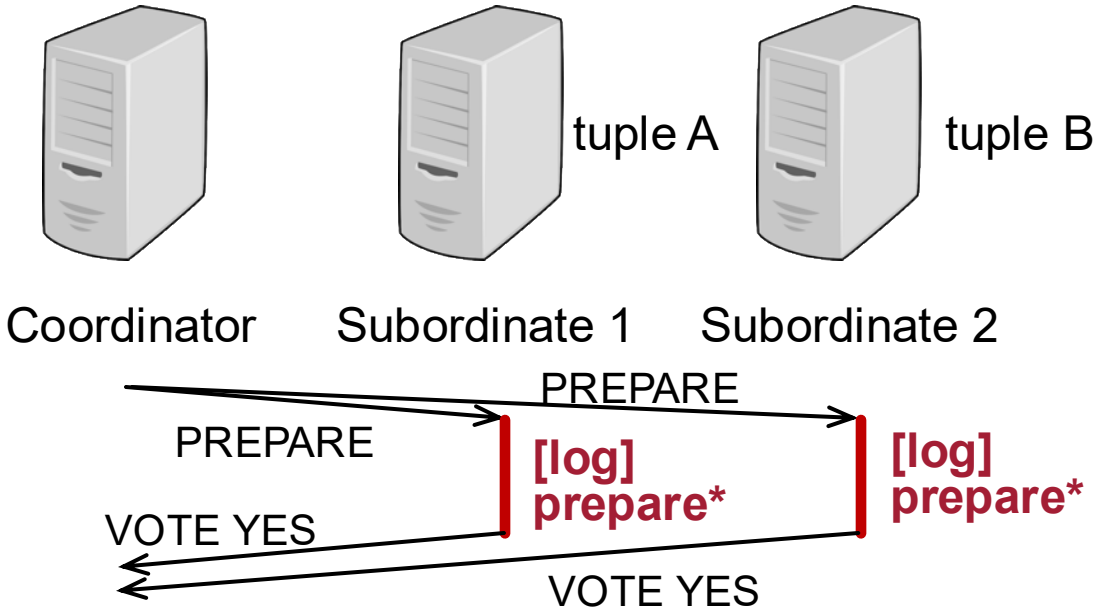
- Transaction T commits in node 1 but not in node 2

Two-Phase Commit (2PC)



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

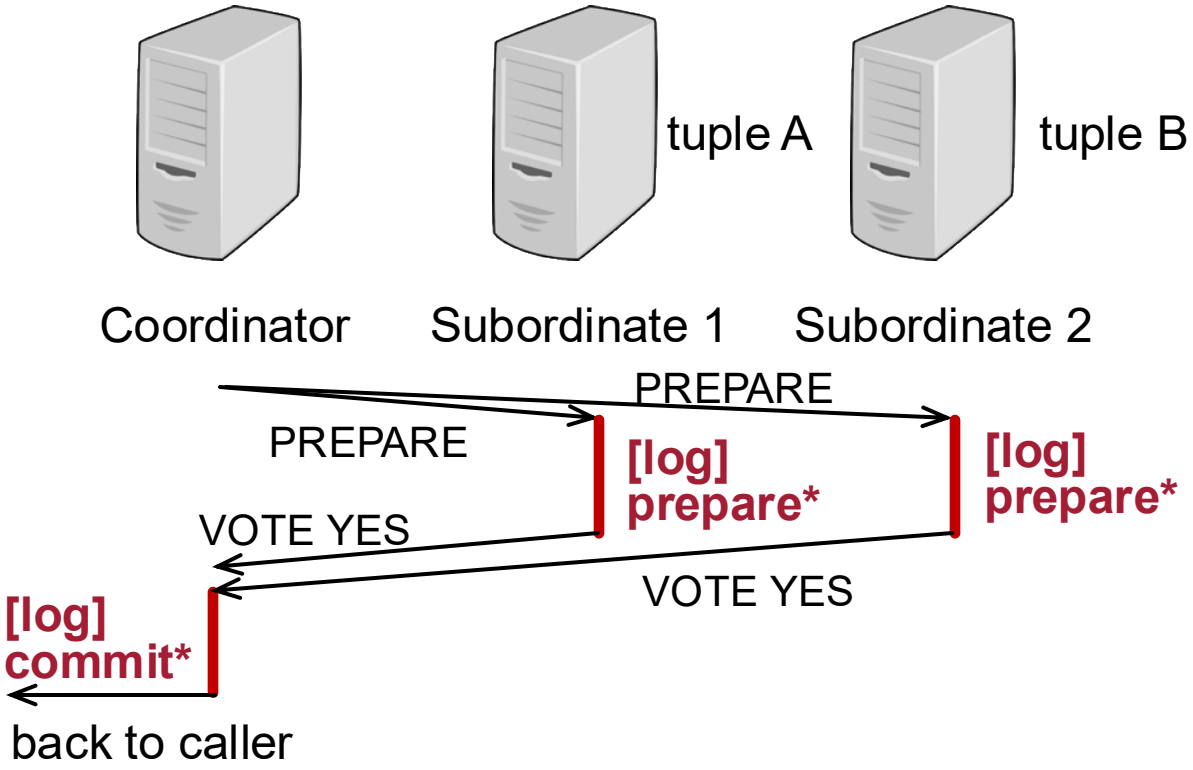
Two-Phase Commit (2PC)



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

Phase 1: prepare phase

Two-Phase Commit (2PC)



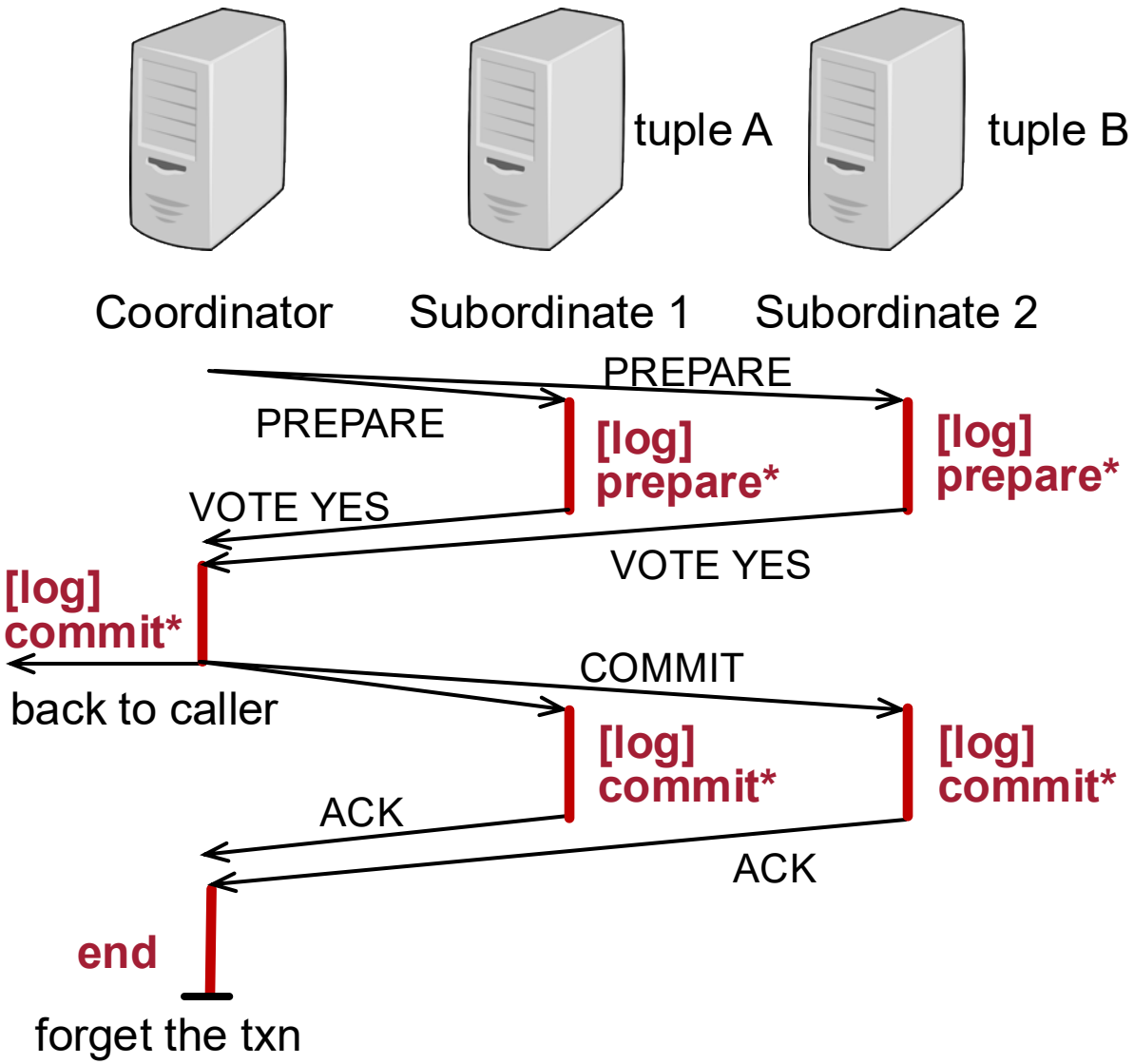
Key idea: let the coordinator log the final commit/abort decision

Phase 1: prepare phase

Phase 2: commit phase

- Coordinator logs the decision

Two-Phase Commit (2PC)



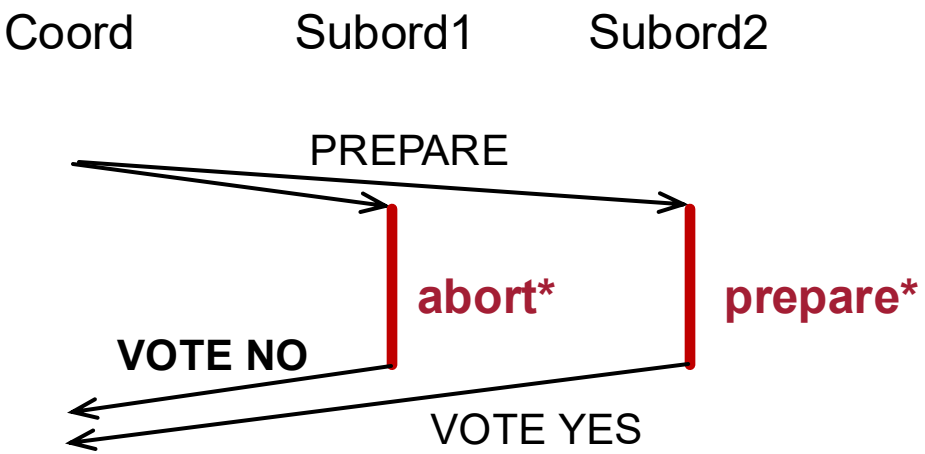
Key idea: let the coordinator log the final commit/abort decision

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

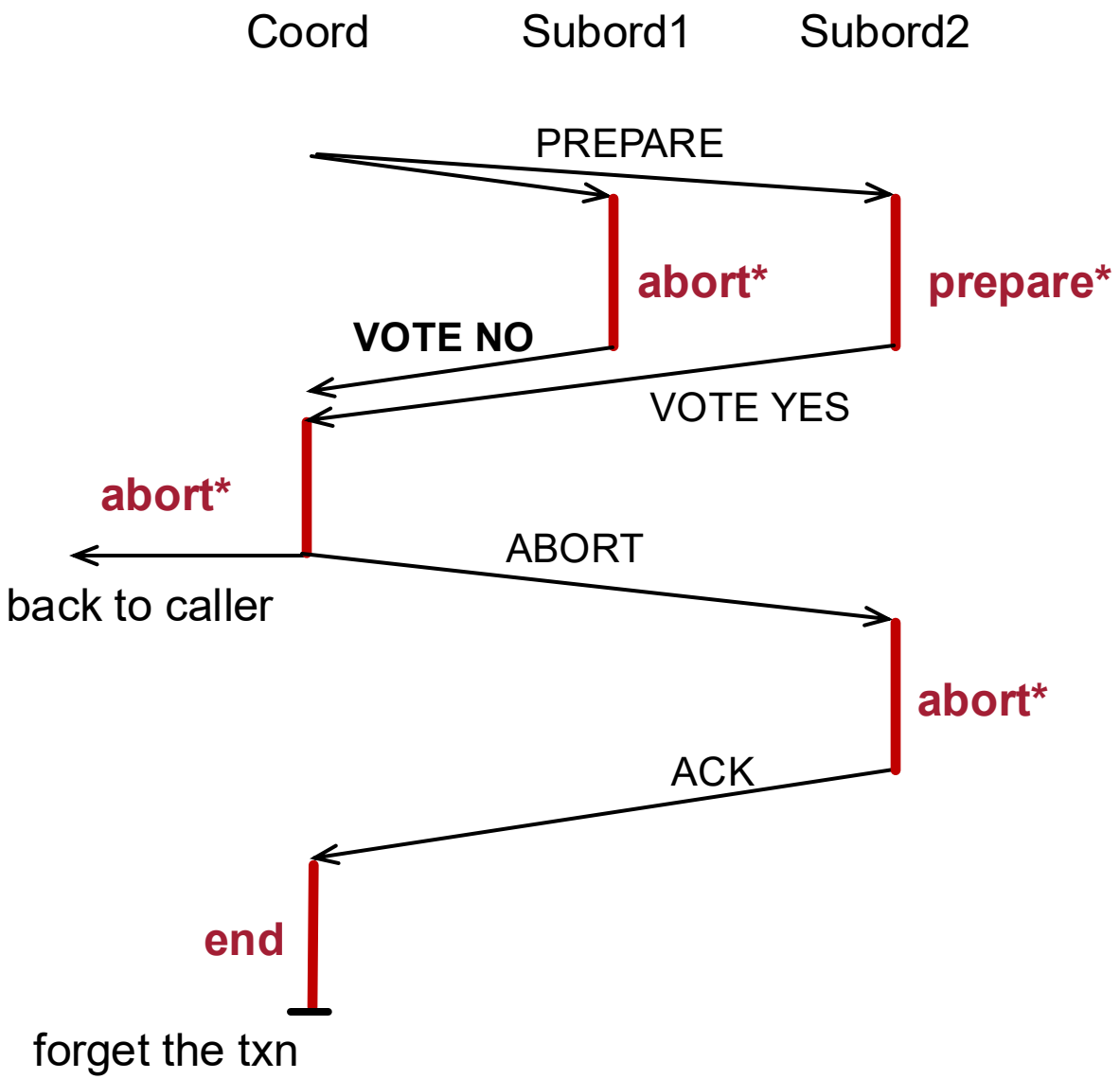
2PC – Abort Example



Subordinate returns VOTE NO if the transaction is aborted

- Subordinate can release locks and forget the transaction

2PC – Abort Example

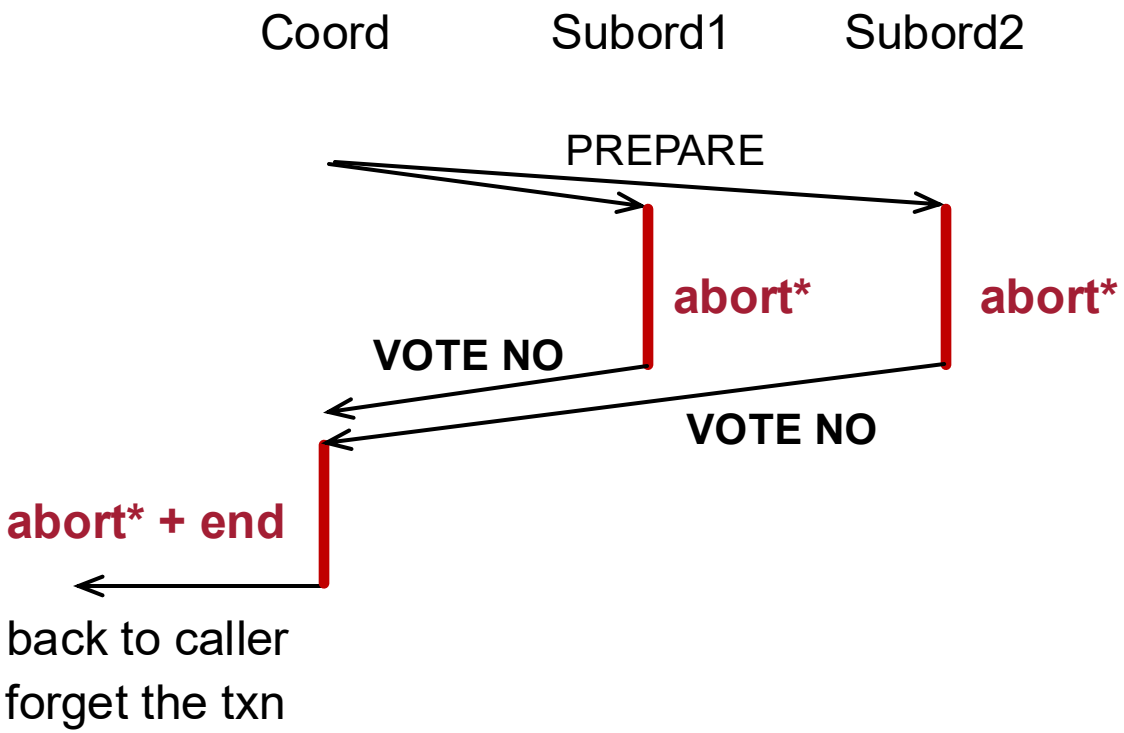


Subordinate returns VOTE NO if the transaction is aborted

- Subordinate can release locks and forget the transaction

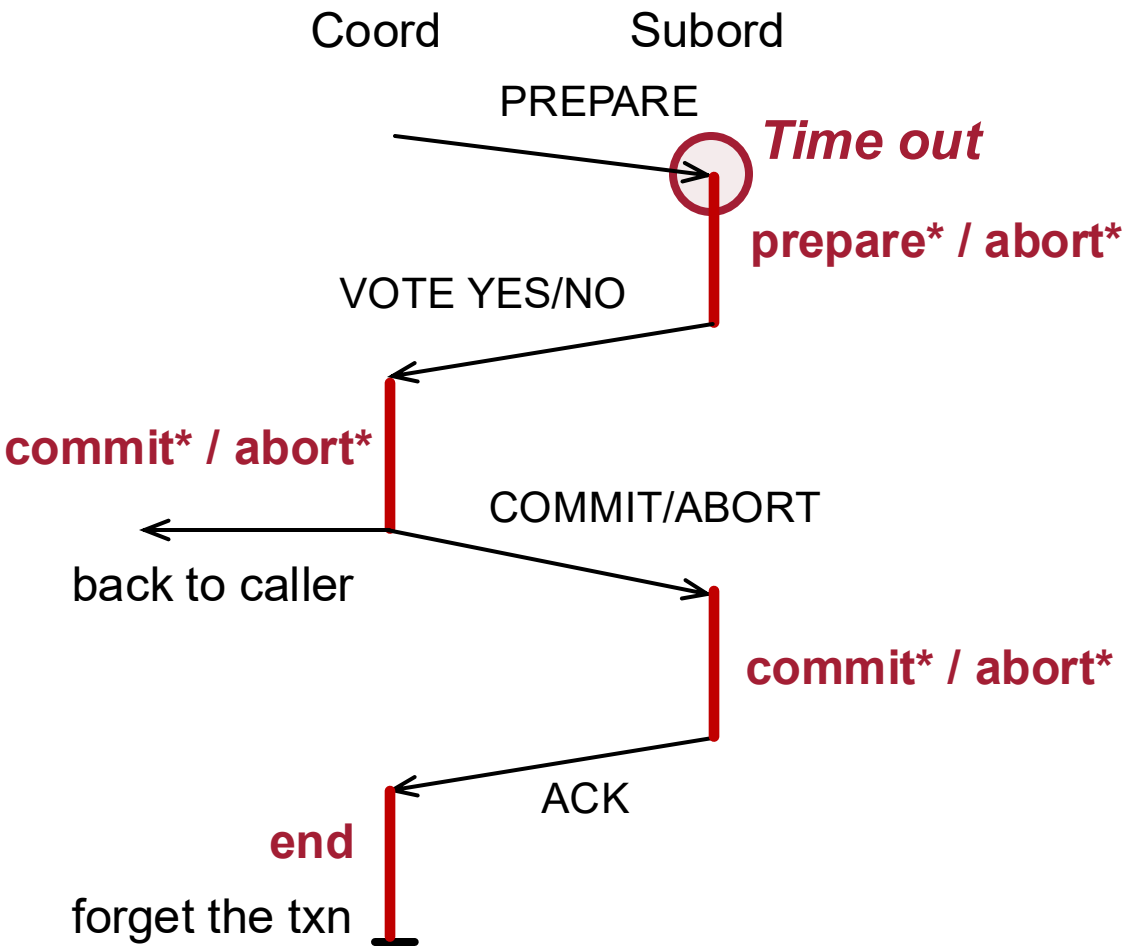
Skip the commit phase for aborted subordinates

2PC – All Subordinates Abort



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

2PC – Failures

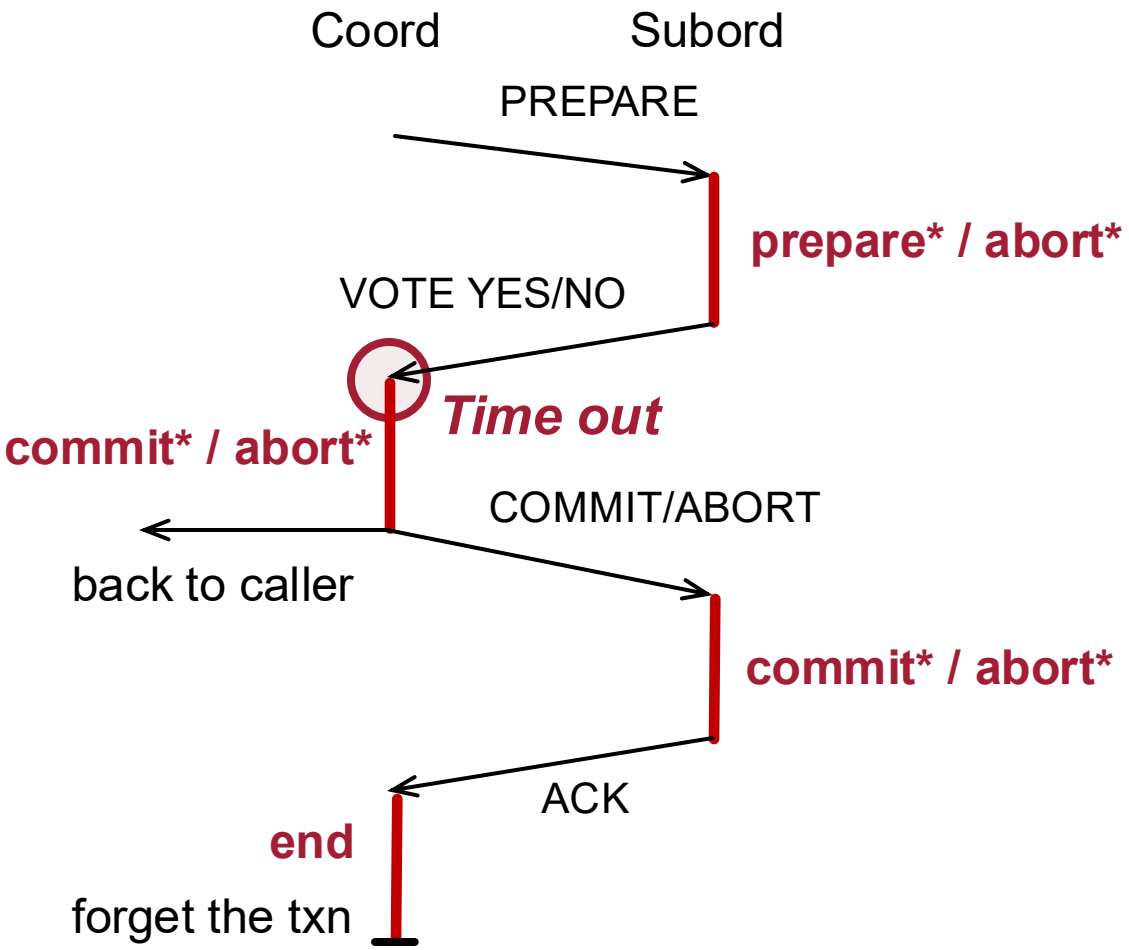


Use timeout to detect failures

Subordinate timeout

- Waiting for PREPARE: self abort

2PC – Failures

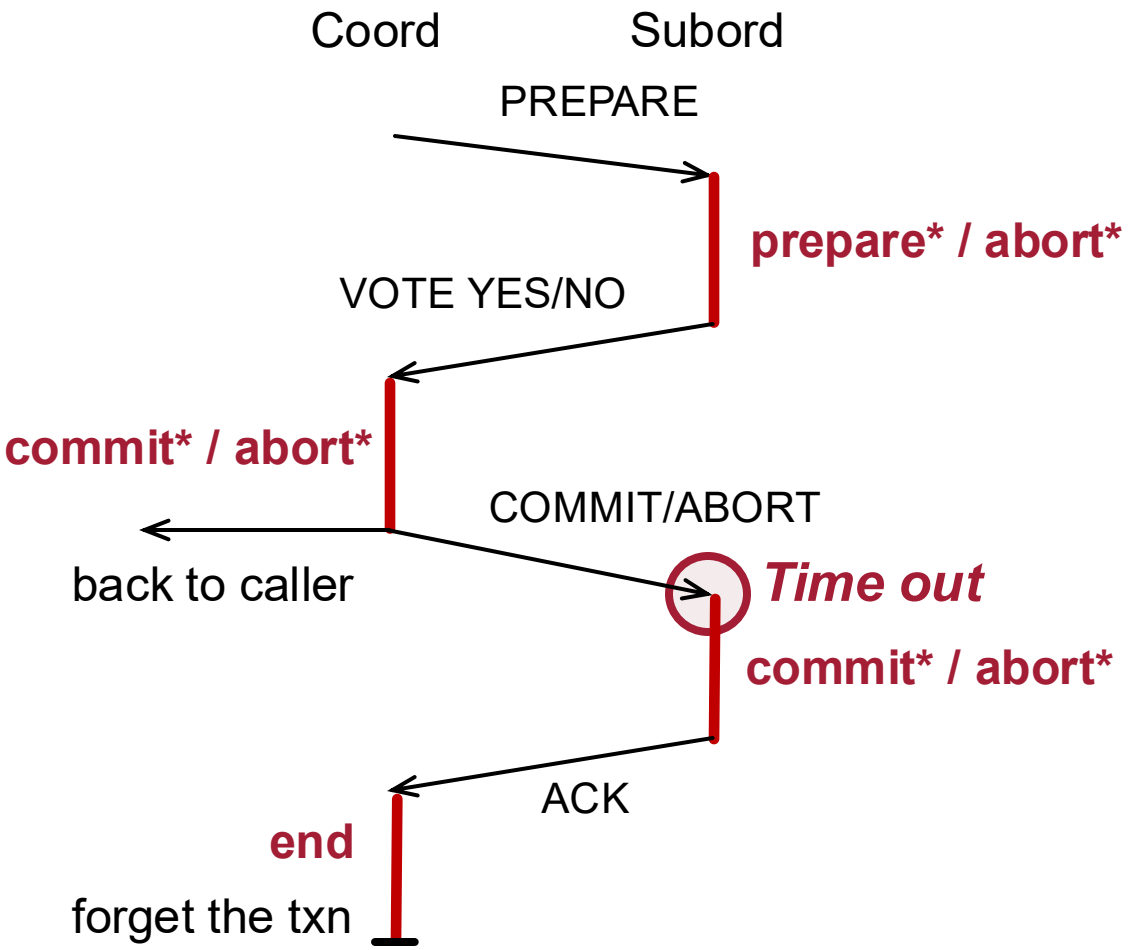


Use timeout to detect failures

Coordinator timeout

- Waiting for vote: self abort

2PC – Failures

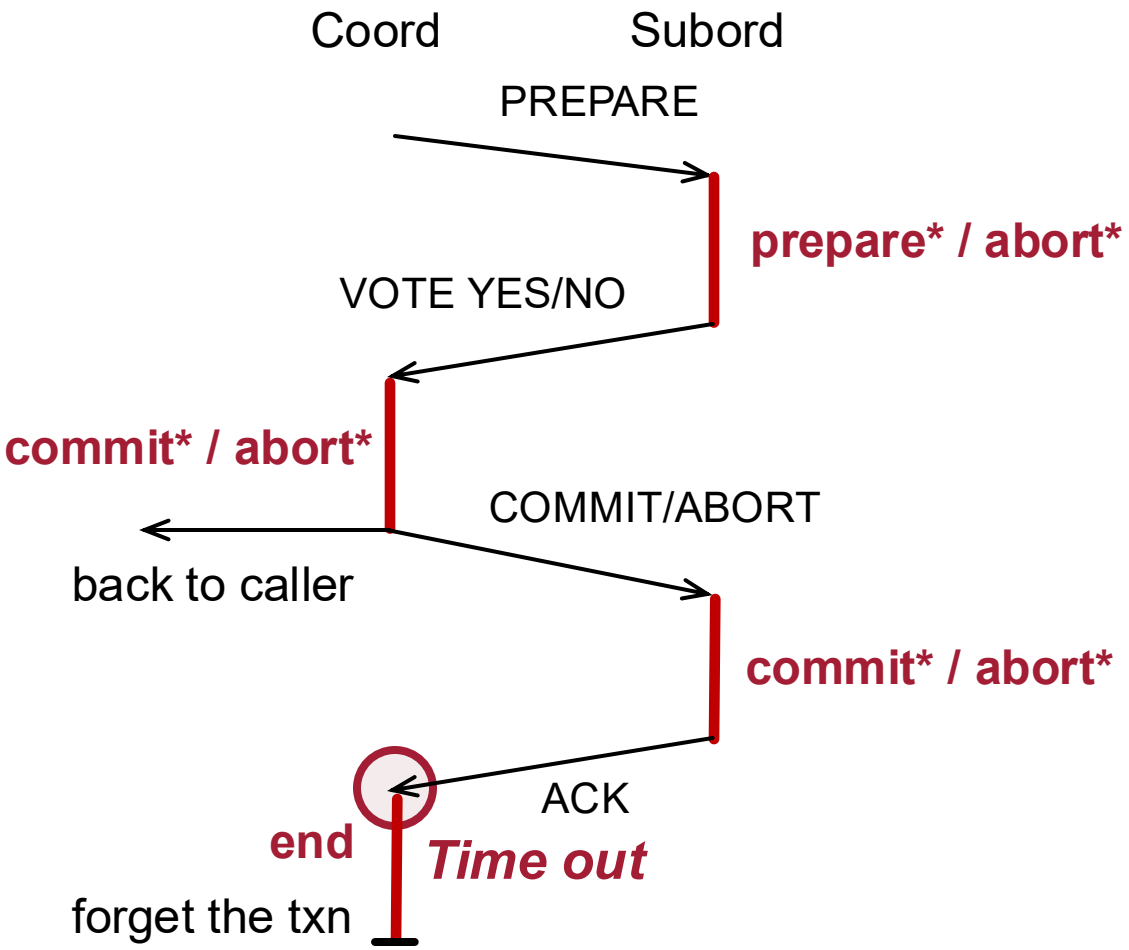


Use timeout to detect failures

Subordinate timeout

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

2PC – Failures

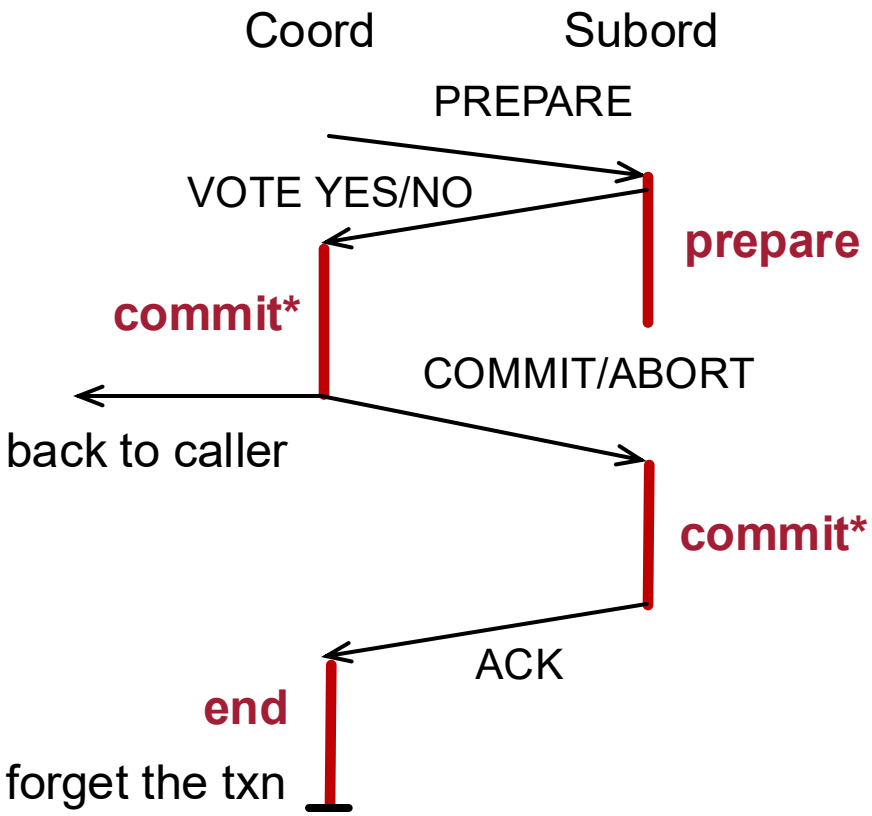


Use timeout to detect failures

Coordinator timeout

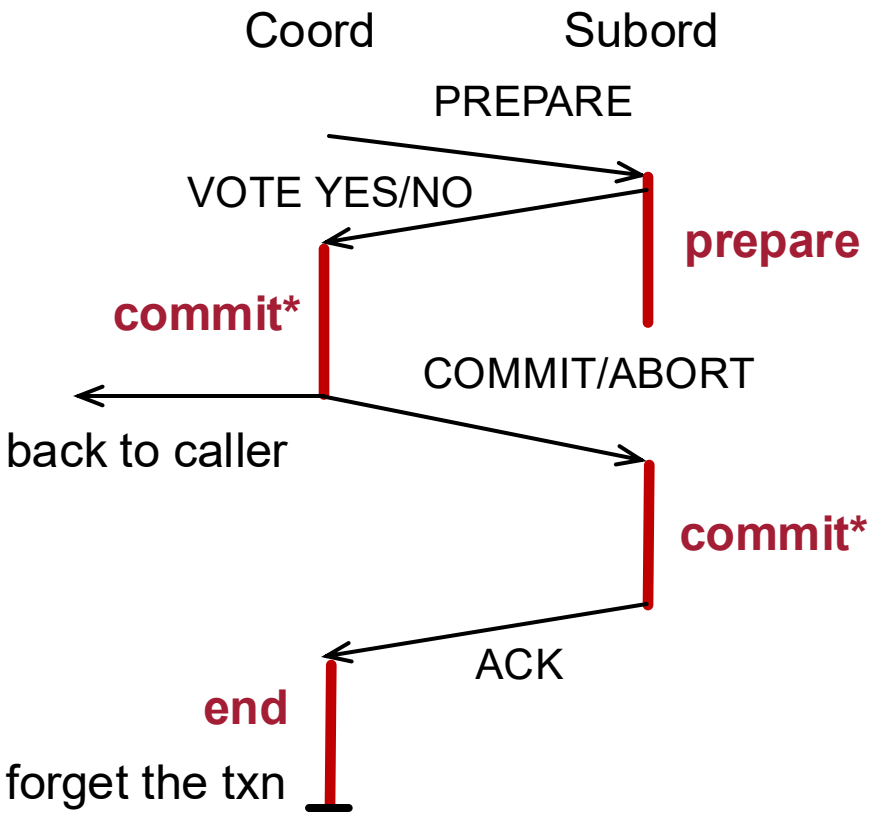
- Waiting for ACK: contact subordinates

2PC – Alternative Designs?



Subordinate returns vote to coordinator before logging prepare?

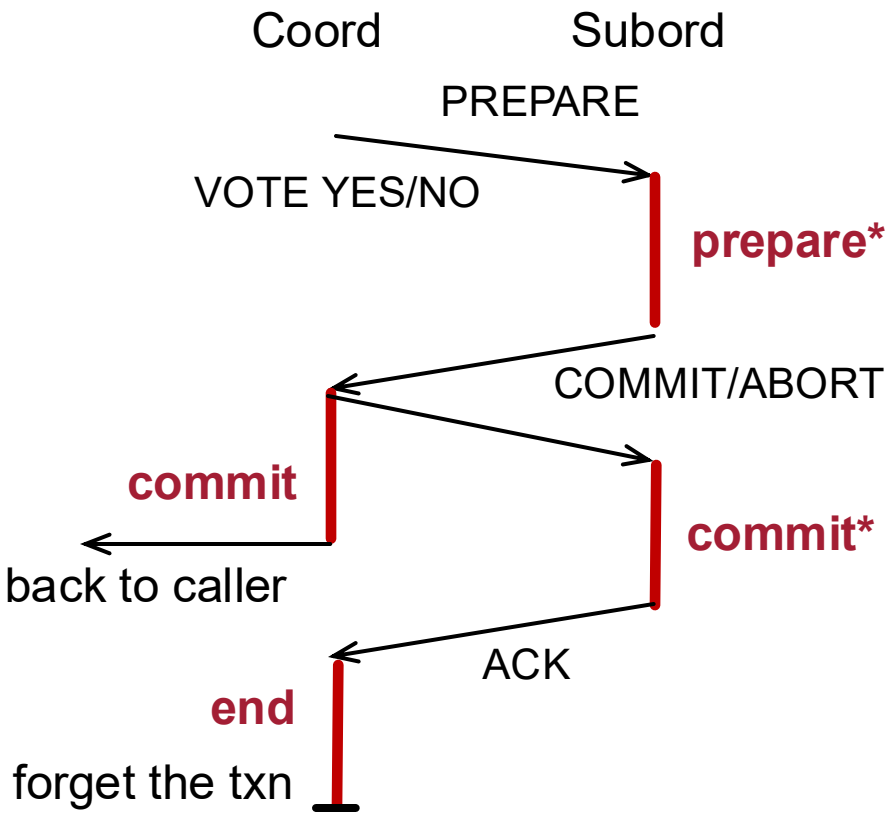
2PC – Alternative Designs?



Subordinate returns vote to coordinator before logging prepare?

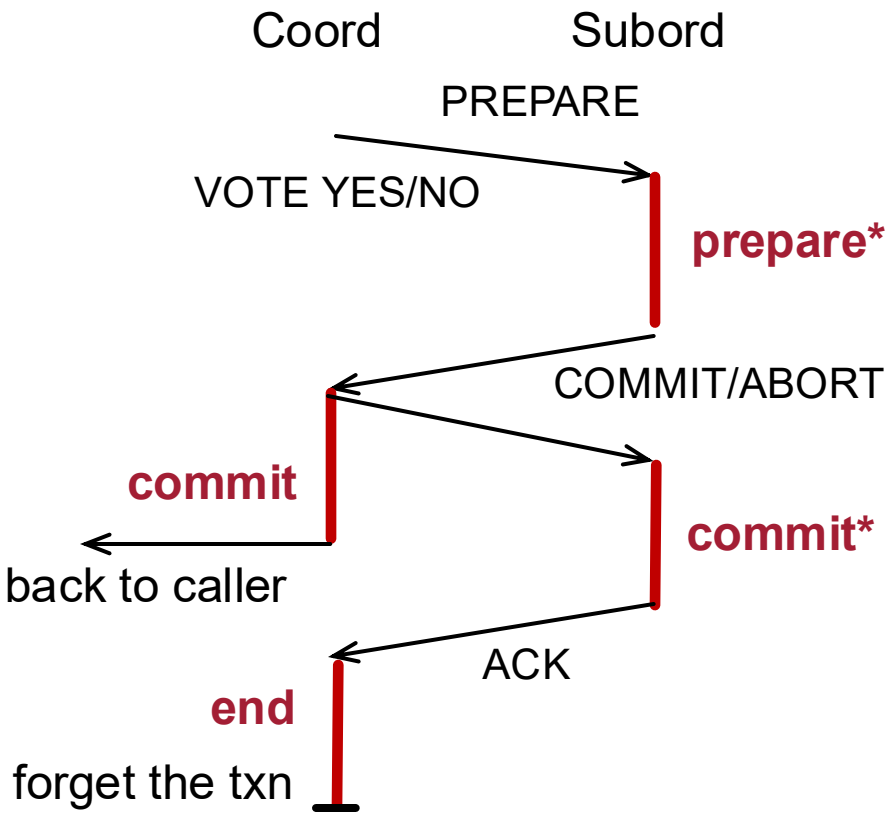
Problem: subordinate may crash before the log record is written to disk. The log record is thus lost but the coordinator already committed the transaction

2PC – Alternative Designs?



Coordinator sends decision to subordinates before logging the decision?

2PC – Alternative Designs?



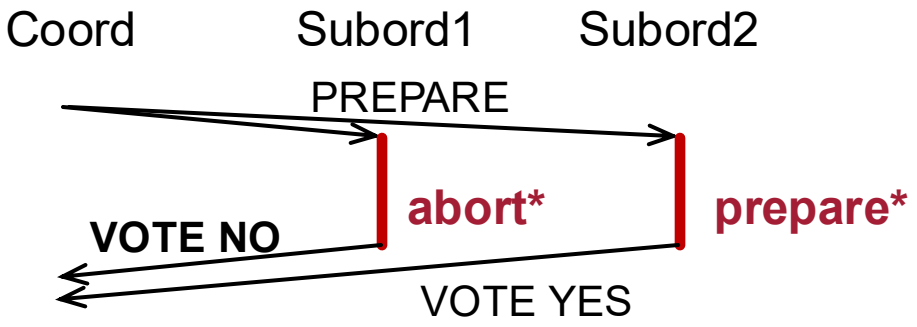
Coordinator sends decision to subordinates before logging the decision?

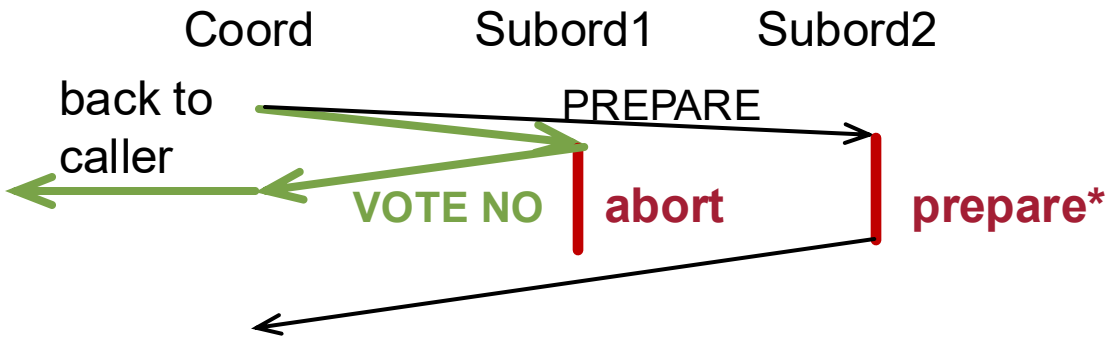
Problem: coordinator crashes before logging the decision and decides to abort after restart

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



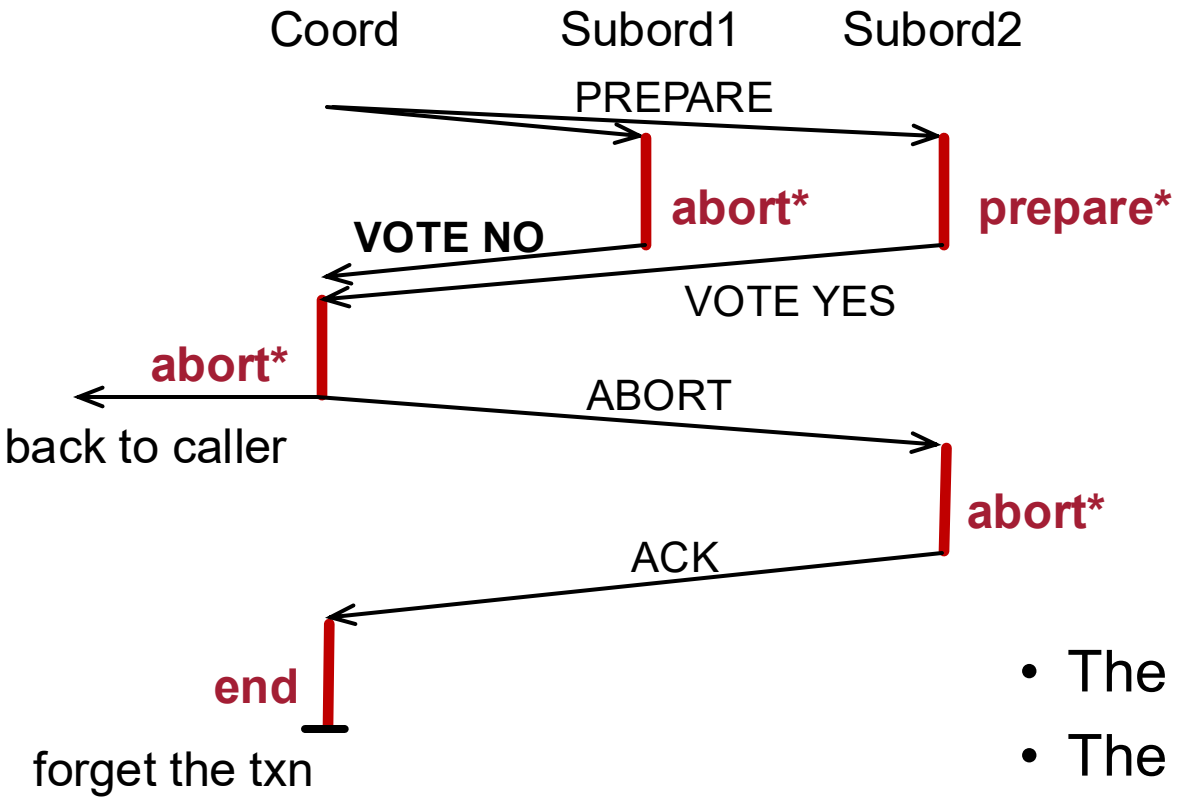


Presumed Abort

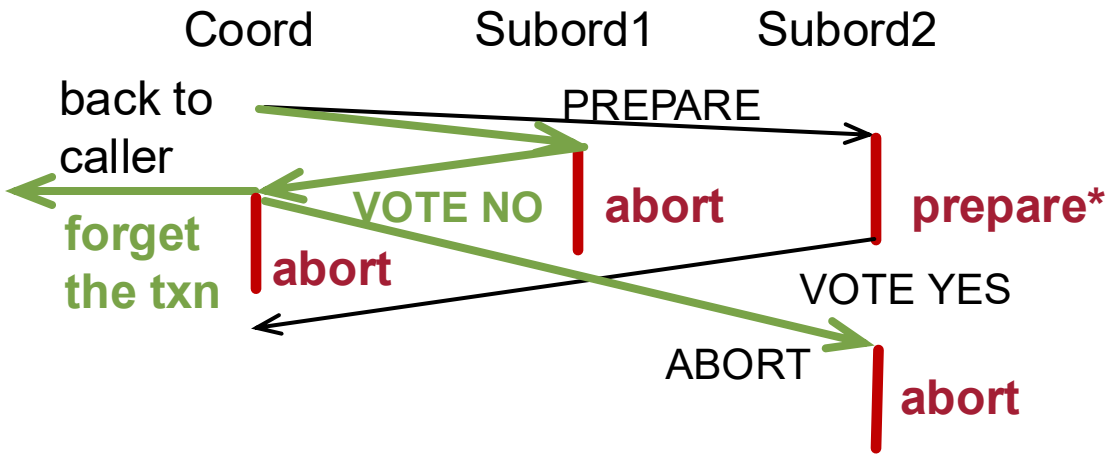
- The abort record is not forced in subordinate

Standard 2PC

PA: Aborted Transaction



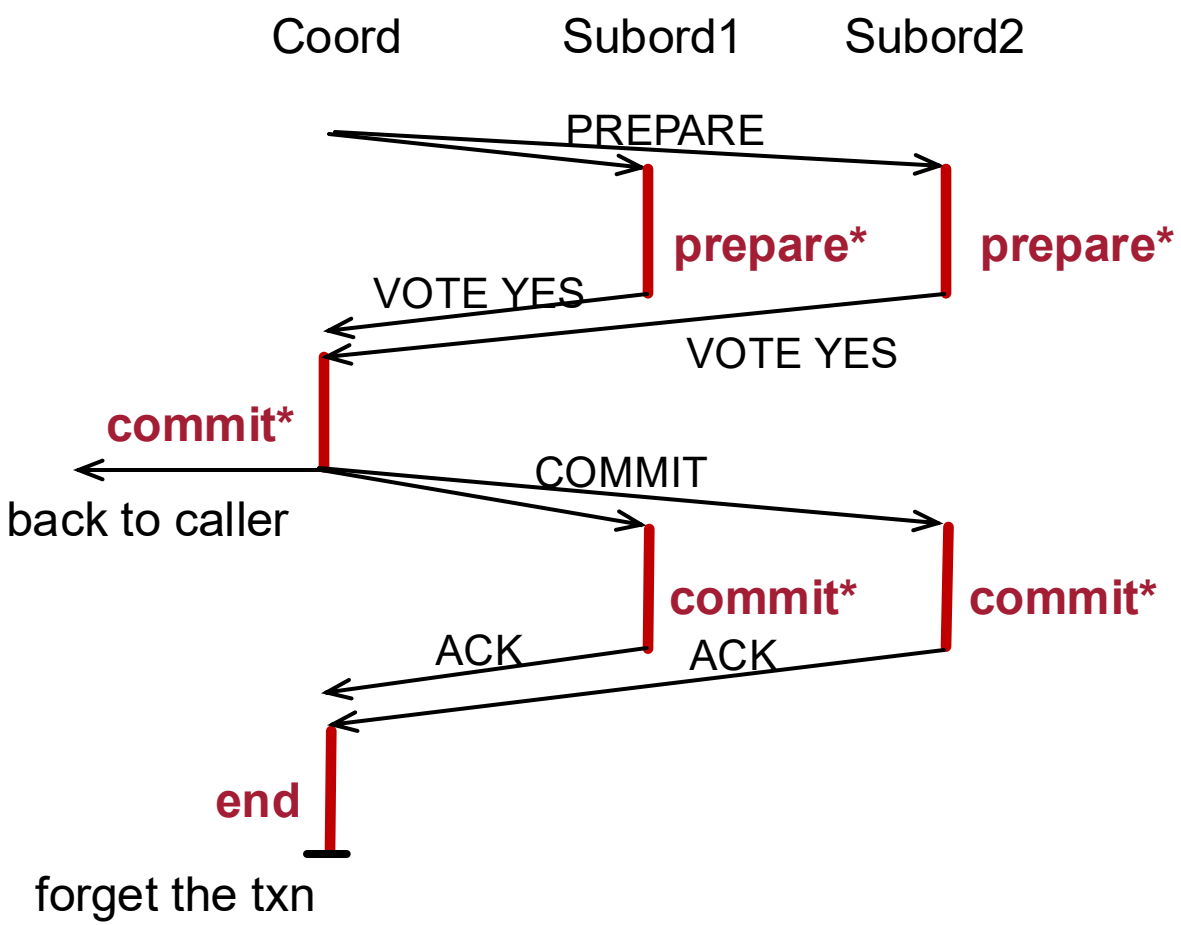
Standard 2PC

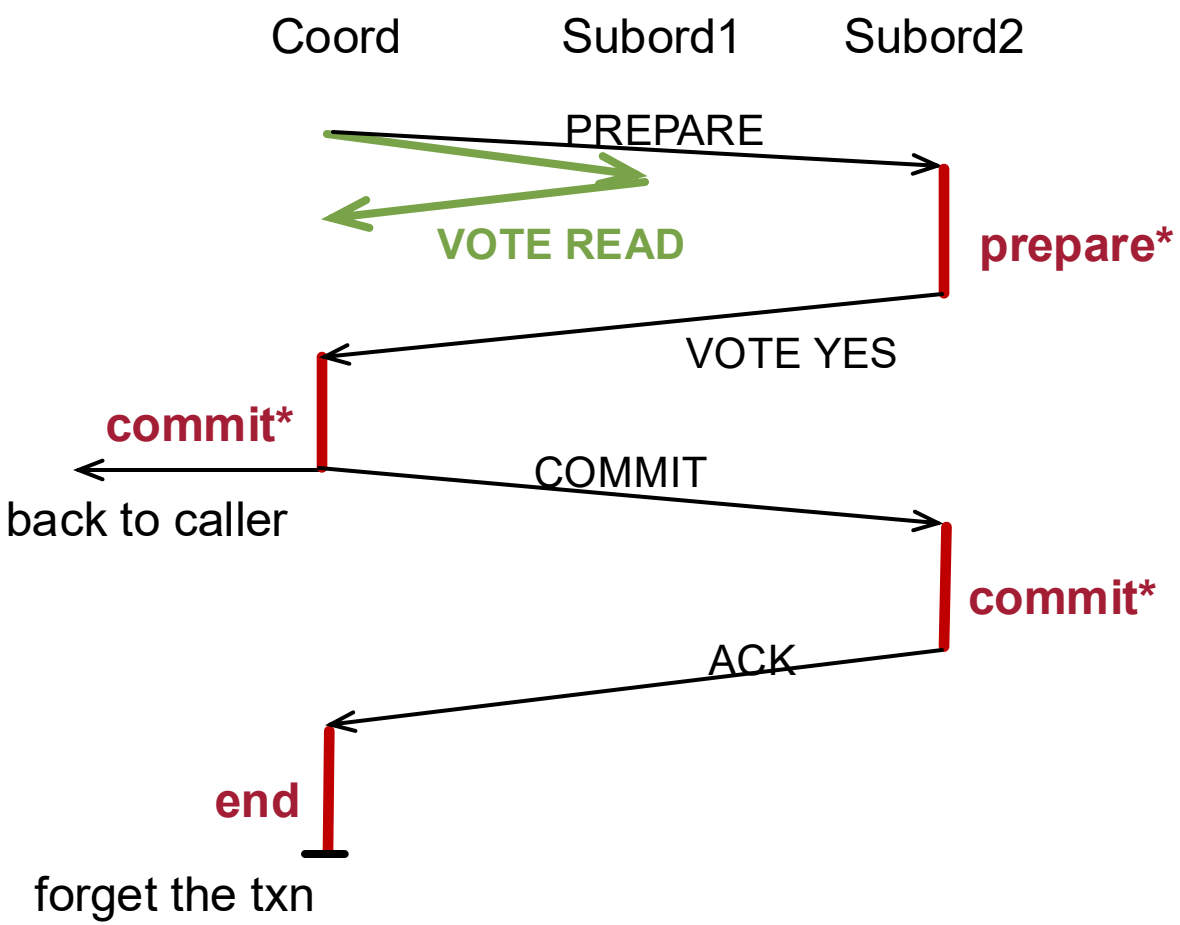


Presumed Abort

- 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**

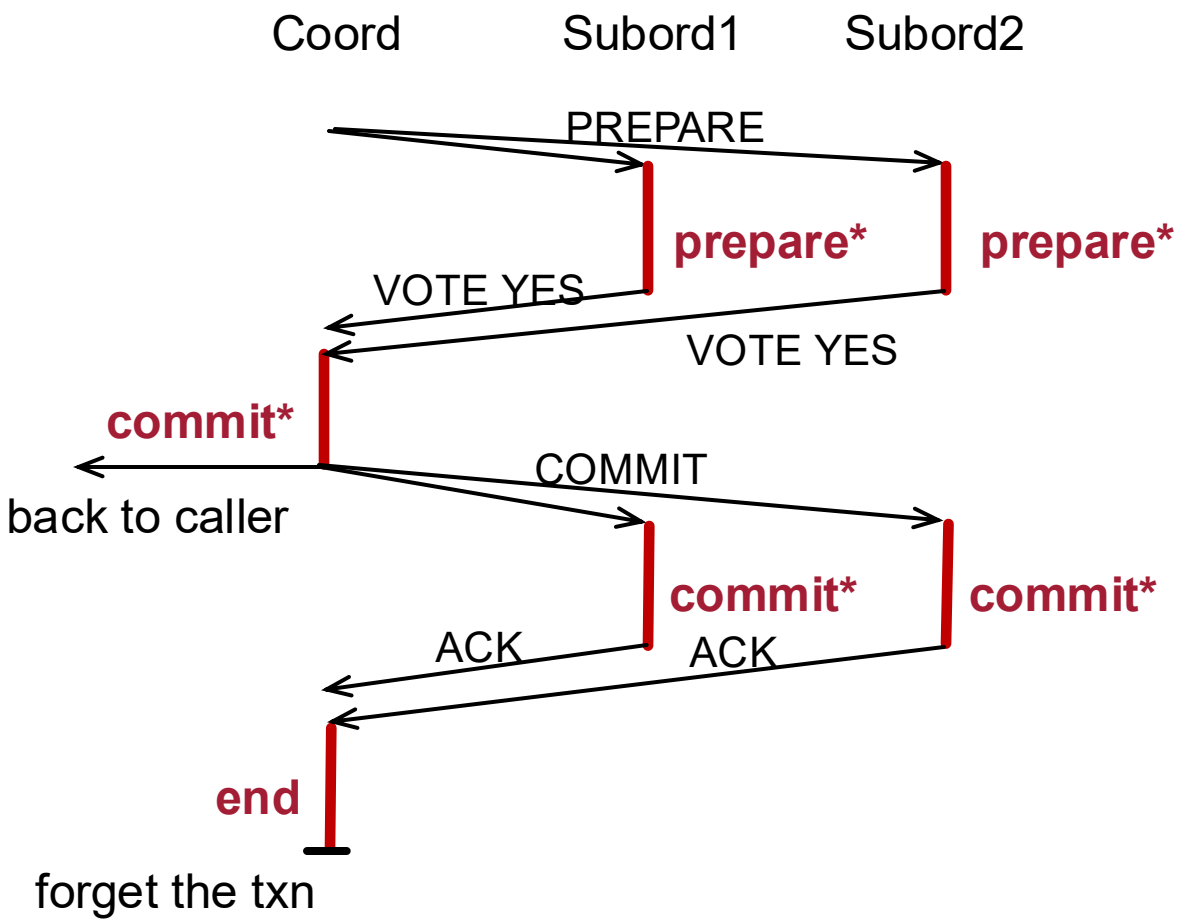
PA: Partially Readonly Transactions

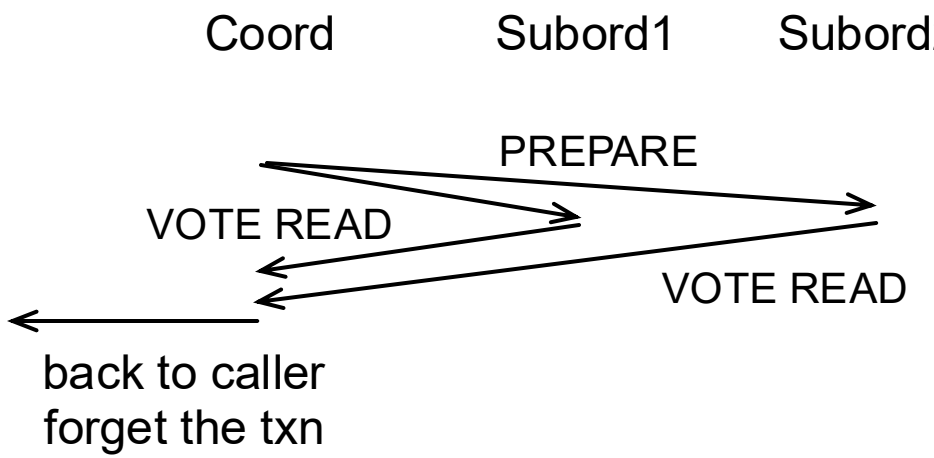




Readonly subordinate does not log in prepare phase and skips commit phase

PA: Completely Readonly Transactions



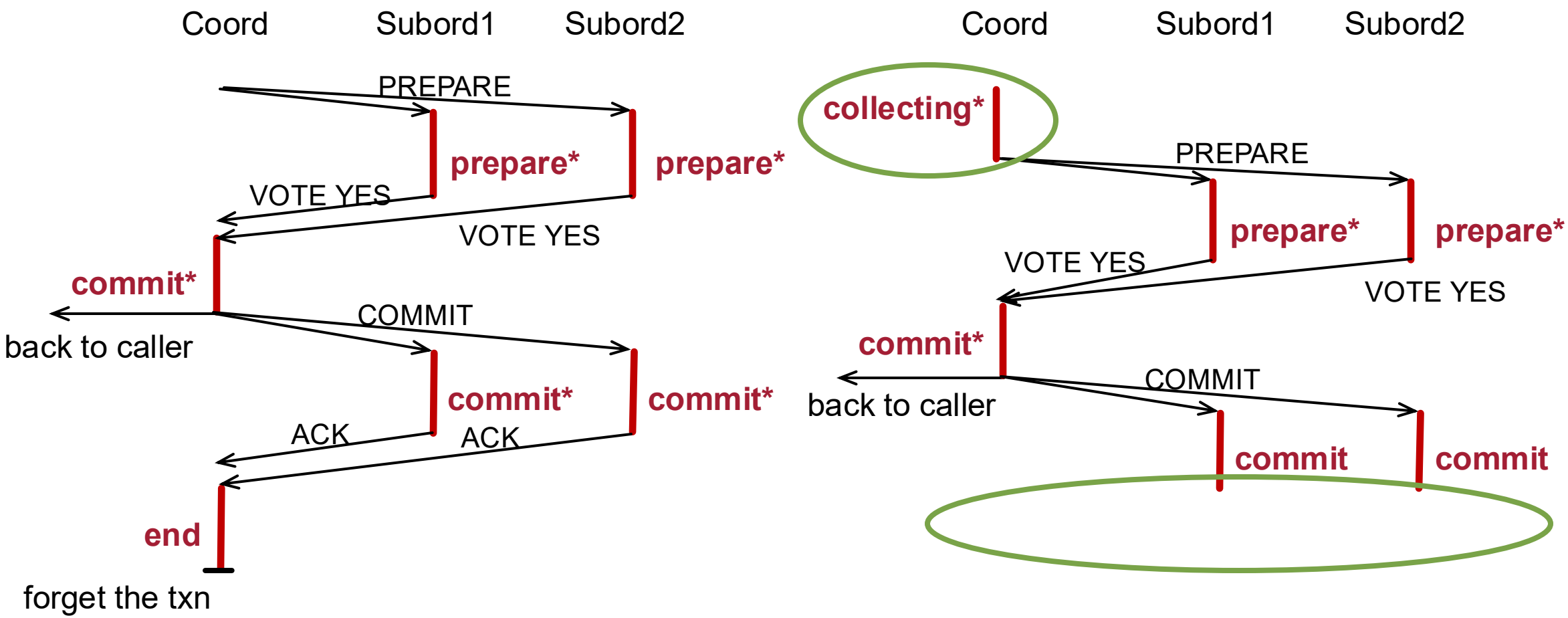


Completely readonly transactions skip the commit phase entirely

Optimization 2: Presumed Commit (PC)

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

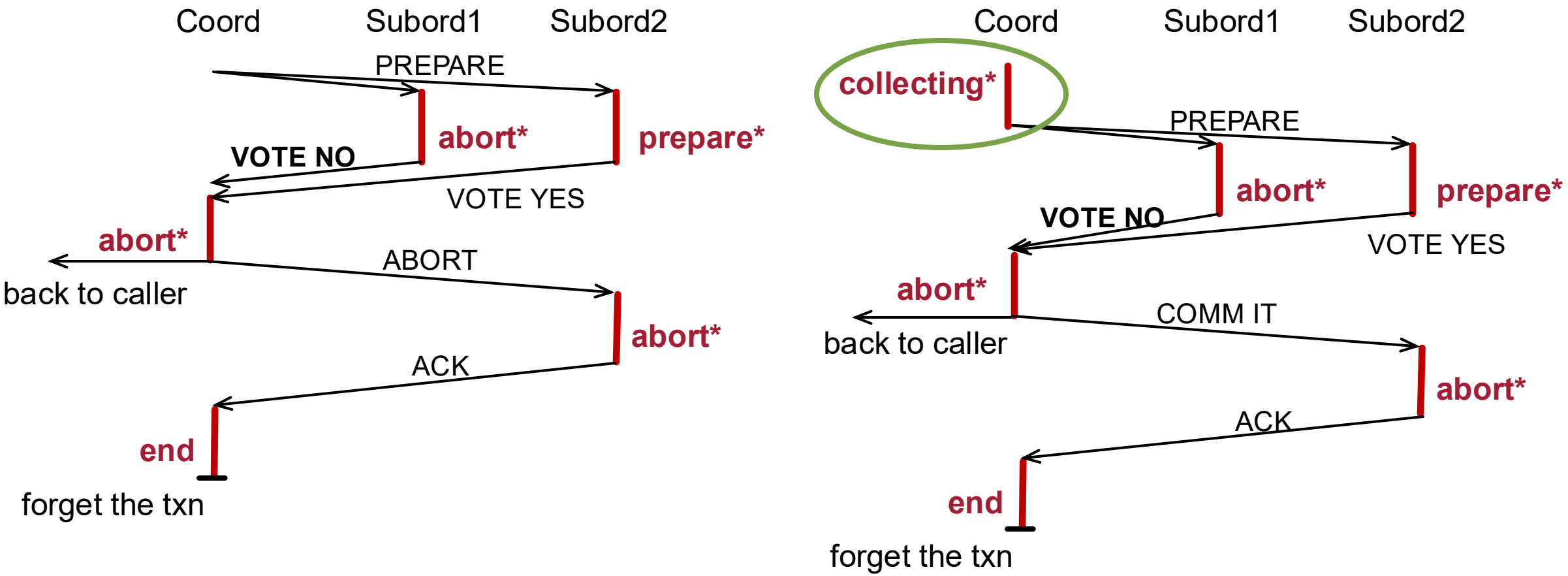
PC: Committed Transaction



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 Yes US	U 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

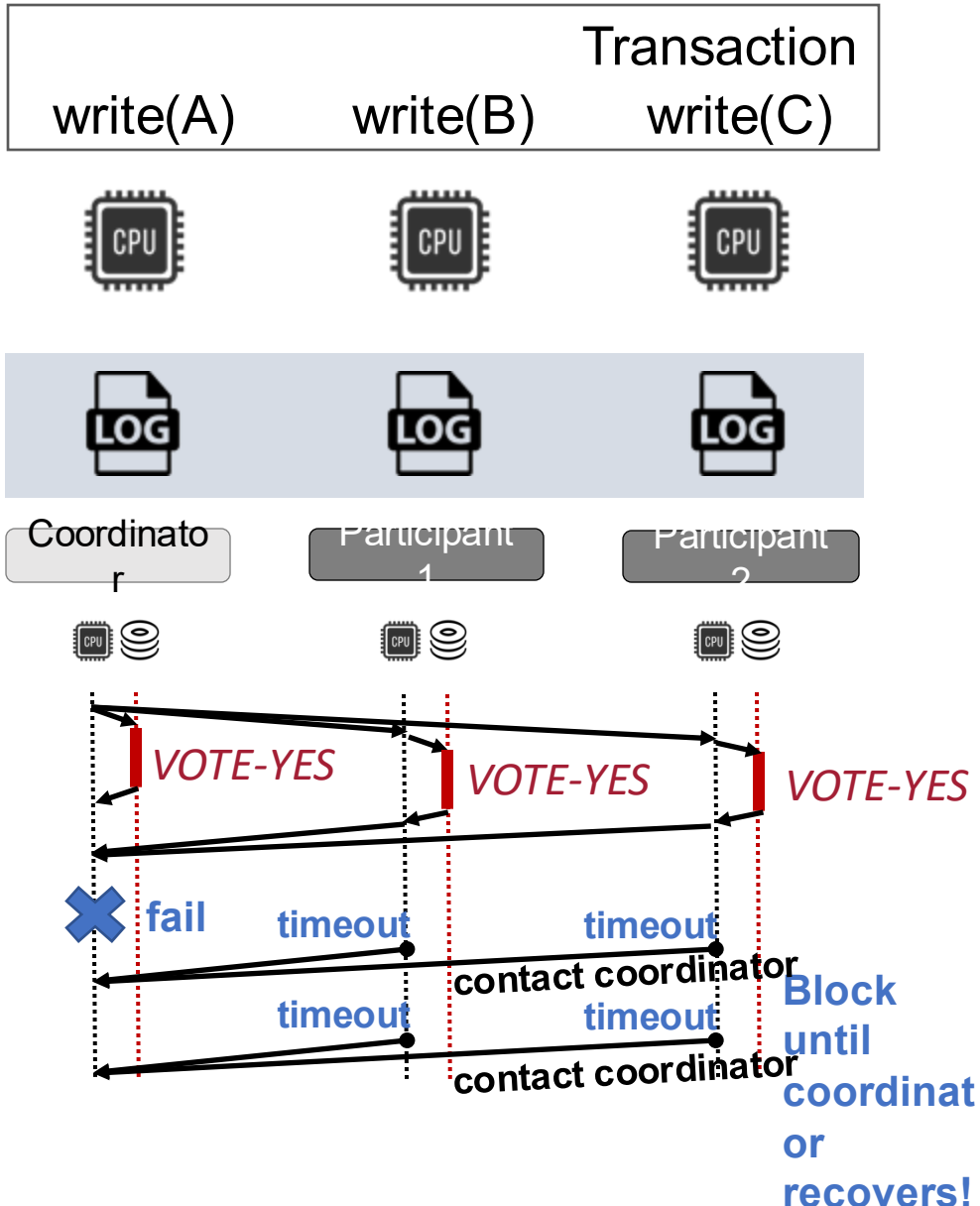
Agenda

Two-phase commit

- Presumed abort (PA)
- Presumed Commit (PC)

Cornus: 2PC optimized for storage disaggregation

Limitations of 2PC



Limitation #1: **Long latency**

- User experiences latency of two logging operations

Limitation #2: **Blocking problem**

- Participants are blocked if the coordinator fails

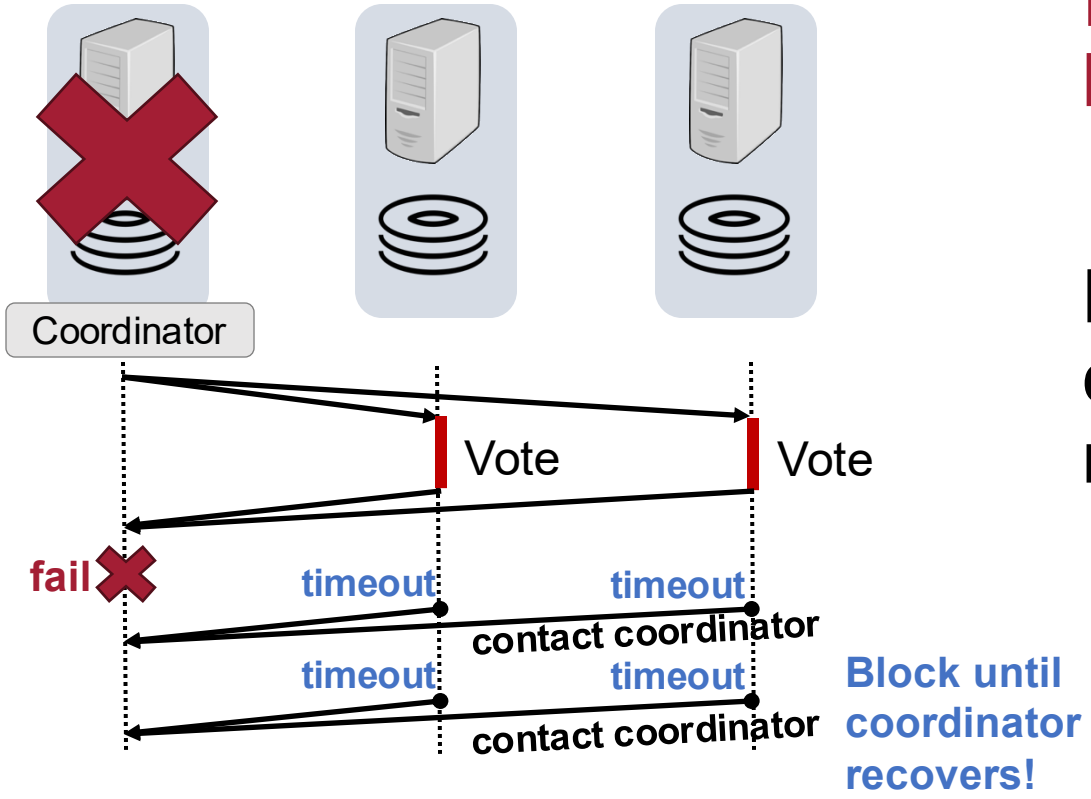
2PC Limitations – Prior Solutions

Solutions	Example systems	Limitations in prior solutions
Reduce latency	Coordinator log [1] Implicit yes vote [2] Early prepare [3]	• Extra system or workload assumptions • Violate site autonomy
Non-blocking	Three-phase commit (3PC) [4]	• Requires extra latency and/or network messages
Codesign 2PC with replication	Paxos commit [5] MDCC [6] Parallel commit [7] TAPIR [8]	• Extra design complexity • Custom-designed consensus protocol

[1] James W Stamos and Flaviu Cristian. *Coordinator log transaction execution protocol*. Distributed and Parallel Databases 1993
[2] Y Al-Houmaily and P Chrysanthis. *Two-phase commit in gigabit-networked distributed databases.* PDCS, 1995
[3] James W Stamos and Flaviu Cristian. *A low-cost atomic commit protocol*. Symposium on Reliable Distributed Systems, 1990
[4] Dale Skeen. *Nonblocking commit protocols*. SIGMOD 1981
[5] Jim Gray and Leslie Lamport. *Consensus on Transaction Commit*. ACM Trans. Database Syst, 2006
[6] TimKraska, et al. *MDCC: Multi-data center consistency*. European Conference on Computer Systems, 2013
[7] Rebecca Taft, et al. *Cockroachdb: The resilient geo-distributed SQL database*. SIGMOD 2020
[8] Irene Zhang, et al. *Building consistent transactions with inconsistent replication*. TOCS 2018

Rethink 2PC with Storage Disaggregation

Shared Nothing

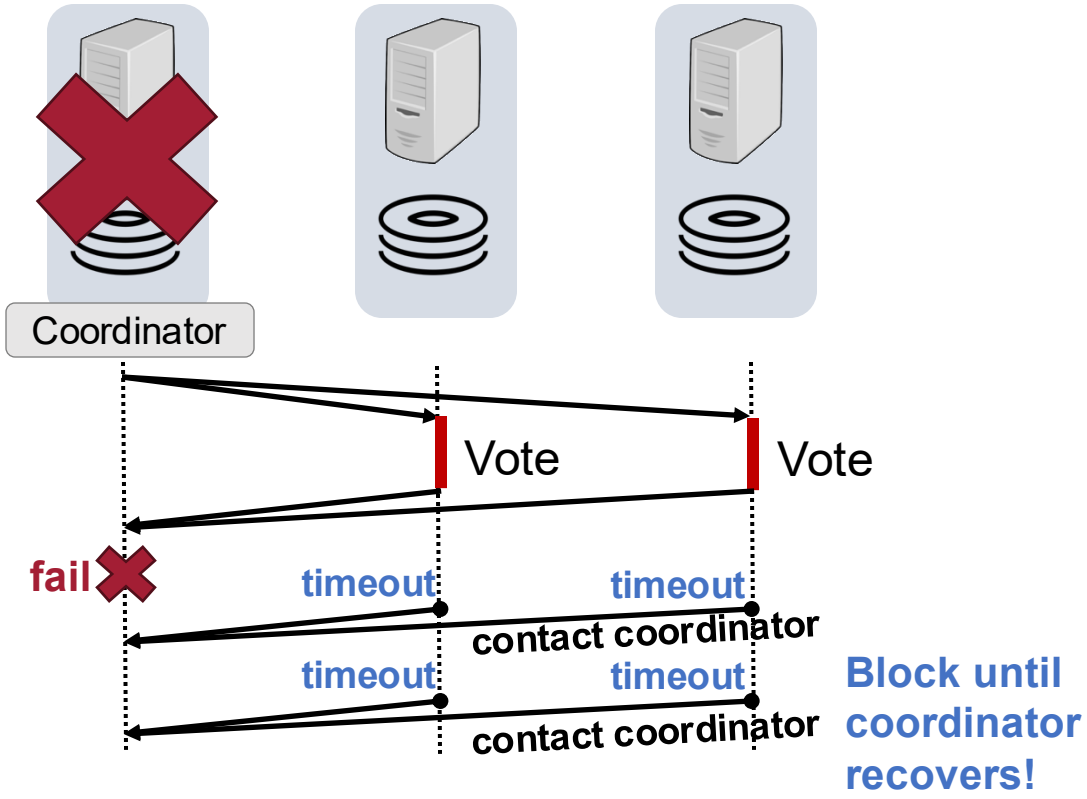


In shared-nothing 2PC, **coordinator failure ⇒ potentially unbounded blocking**

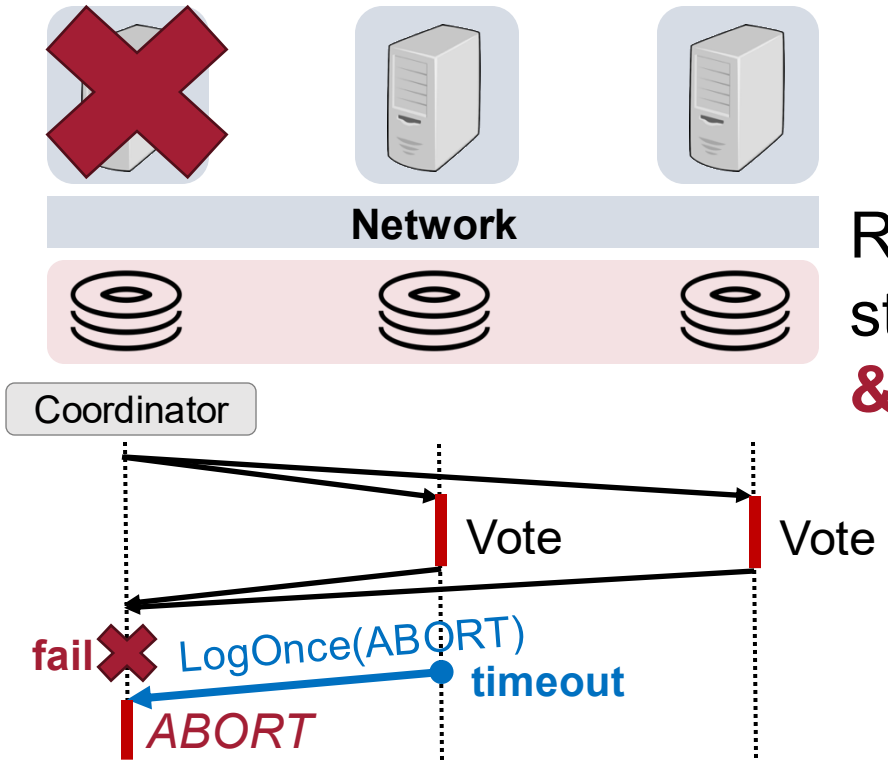
Decision (commit/abort) lives in the coordinator's local log; others cannot read it during failure

Rethink 2PC with Storage Disaggregation

Shared Nothing



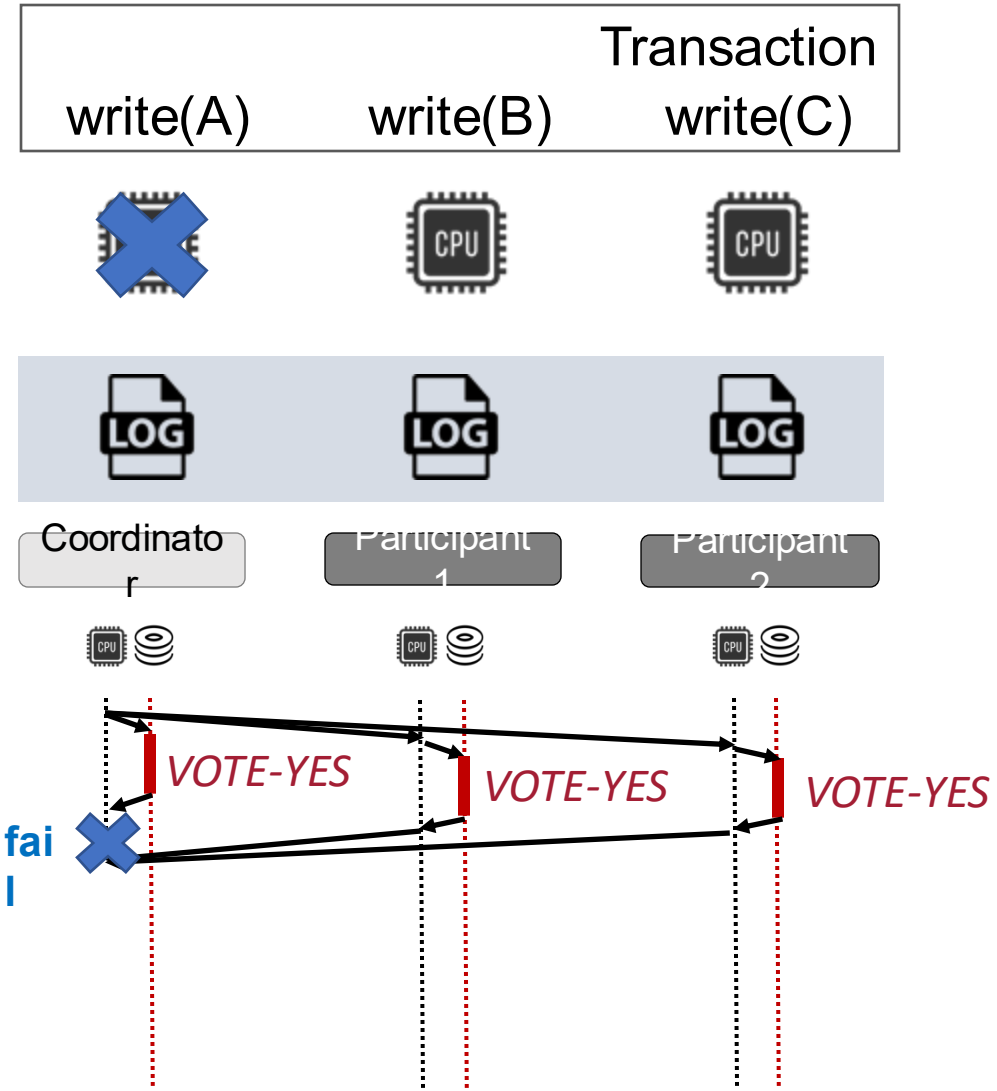
Disaggregation



Replicated storage: **HA & durability**

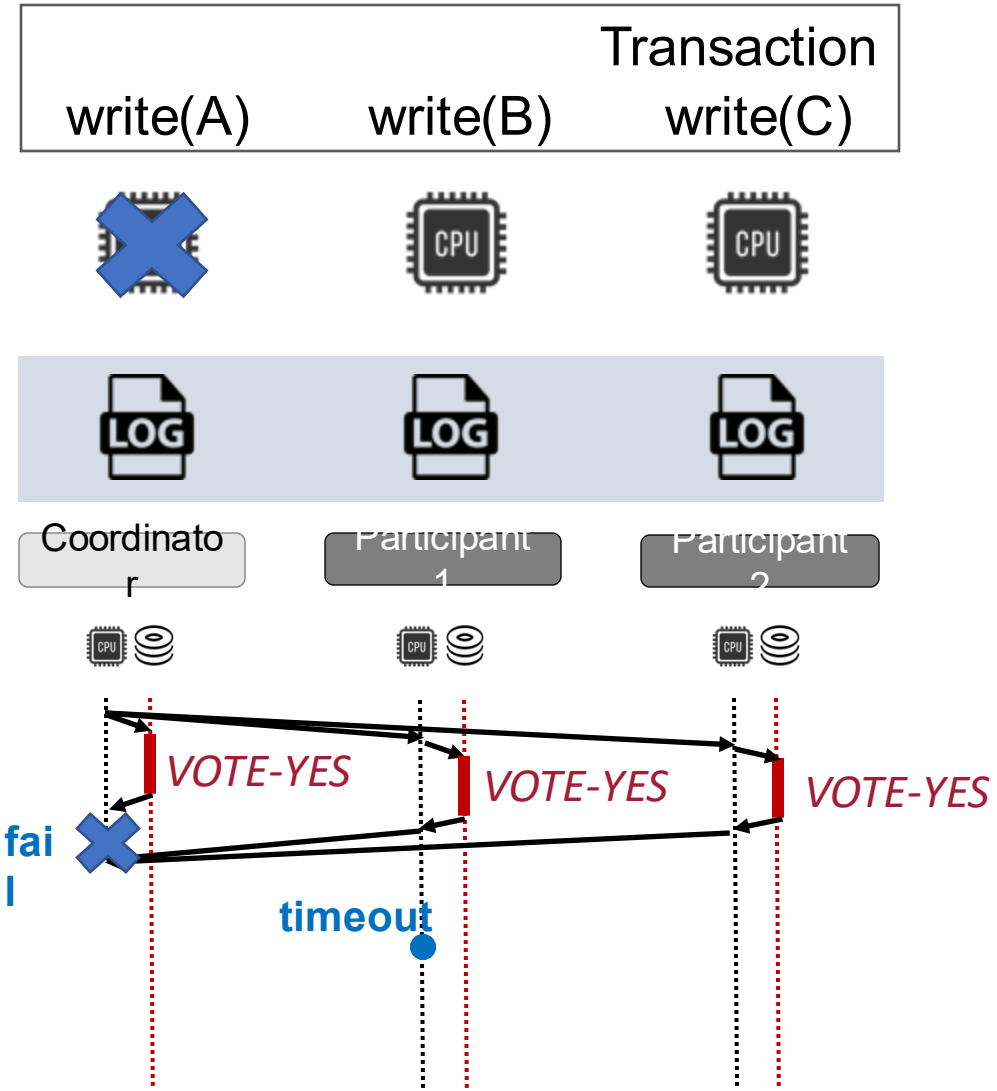
With disaggregated storage, the 2PC log is highly available; any active node can **read peers' votes and CAS-finalize the decision**

Cornus Failure Example



Coordinator fails

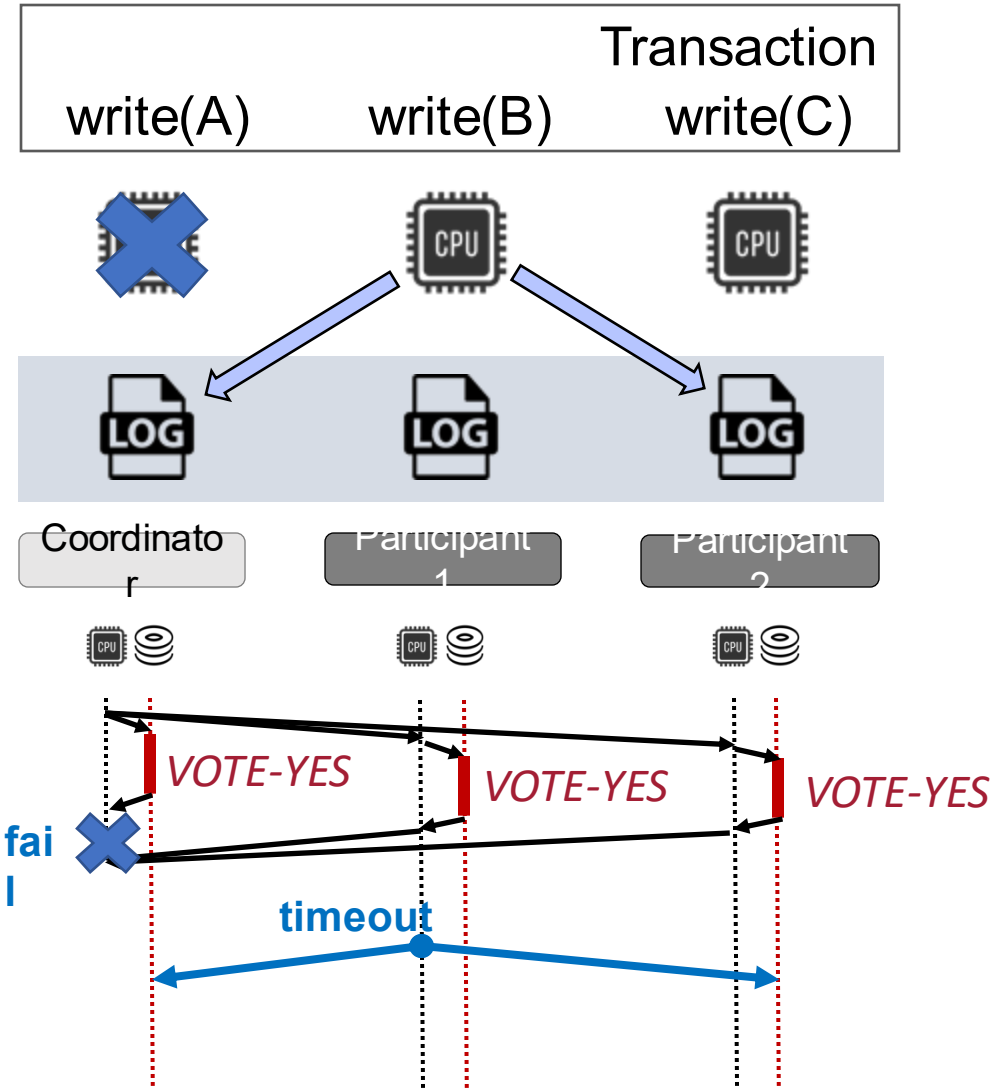
Cornus Failure Example



Coordinator fails

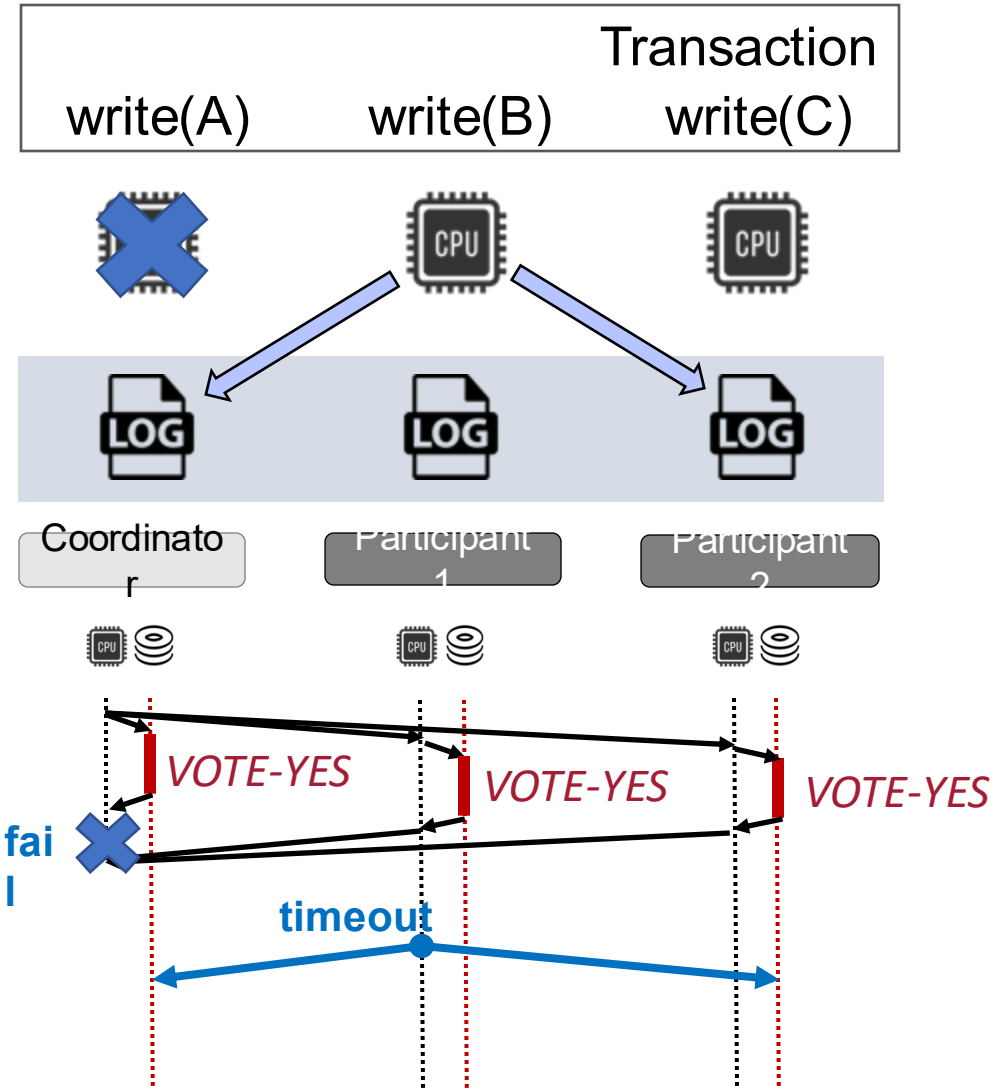
Timeout in participant 1 waiting for coordinator's message

Cornus Failure Example



Use LogOnce() to write ABORT to other nodes' log files

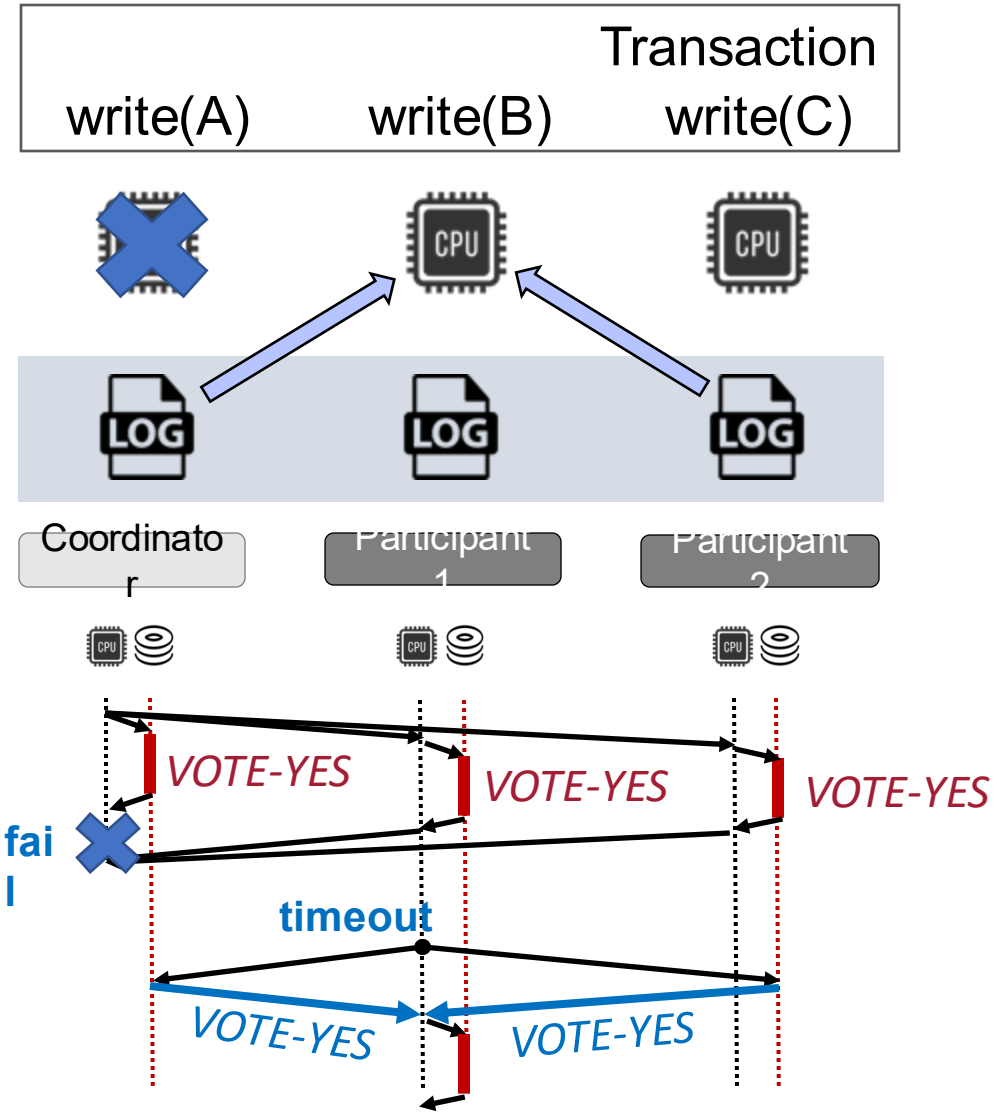
Cornus Failure Example



Use LogOnce() to write ABORT to other nodes' log files

VOTE-YES already exists, LogOnce() does not modify log content

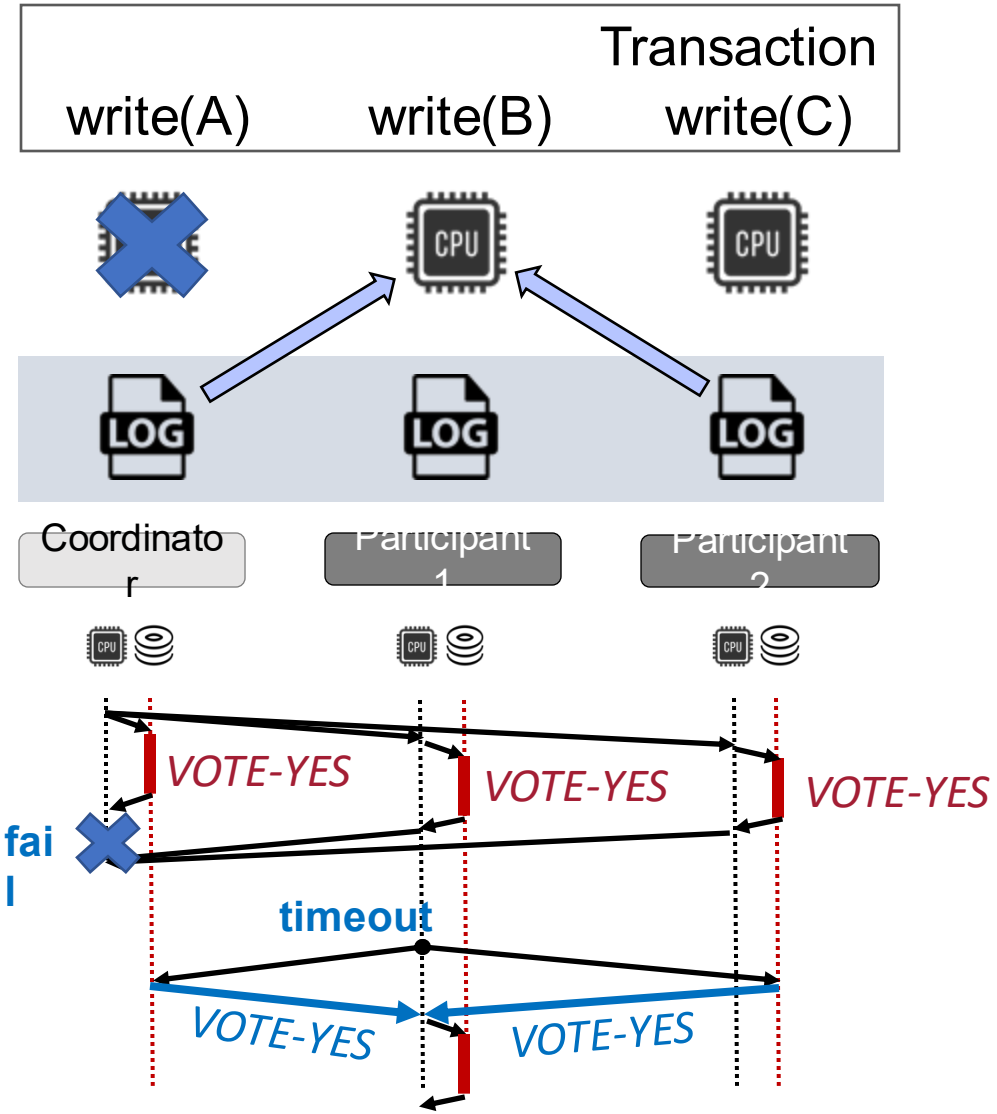
Cornus Failure Example



Storage service returns *VOTE-YES* without updating the logs

Participant 1 logs the *COMMIT* decision

Cornus Failure Example



Storage service returns *VOTE-YES* without updating the logs

Participant 1 logs the *COMMIT* decision

Same process can happen for other participants (e.g., Participant 2)

Q/A – Two Phase Commit

OCC instead of 2PL in distributed OLTP?

Paxos/Raft vs. 2PC-based protocols?

2PC for specific workload or with lower logging overhead?

Do modern systems use 2PC?

Next Lecture

Yi Lu, et al., Aria: A Fast and Practical Deterministic OLTP Database. VLDB, 2020