

CSE 444: Database Internals

Lecture 24

Two-Phase Commit (2PC)

References

- Ullman book: Section 20.5
- Ramakrishnan book: Chapter 22

We are Learning about Scaling DBMSs

- Scaling the execution of a query

- Parallel DBMS

- MapReduce

- Spark



- Scaling transactions

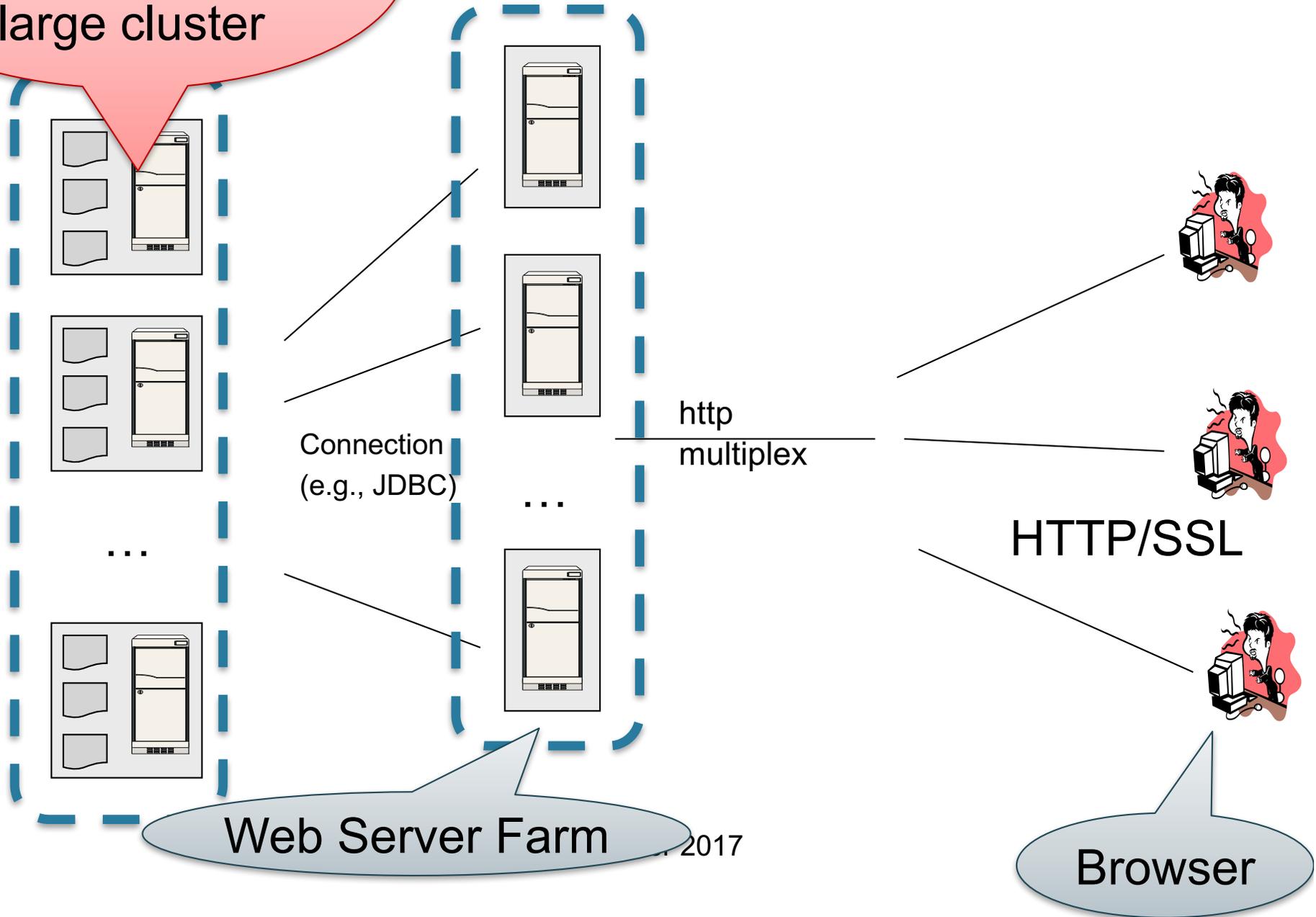
- Distributed transactions

- Replication

- Scaling with NoSQL and NewSQL

Run many transactions in a large cluster

Our Goal



Transaction Scaling Challenges

- **Distribution**
 - There is a limit on transactions/sec on one server
 - Need to partition the database across multiple servers
 - If a transaction touches one machine, life is good!
 - If a transaction touches multiple machines, ACID becomes extremely expensive! Need two-phase commit
- **Replication**
 - Replication can help to increase throughput and lower latency
 - Create multiple copies of each database partition
 - Spread queries across these replicas
 - Easy for reads but writes, once again, become expensive!

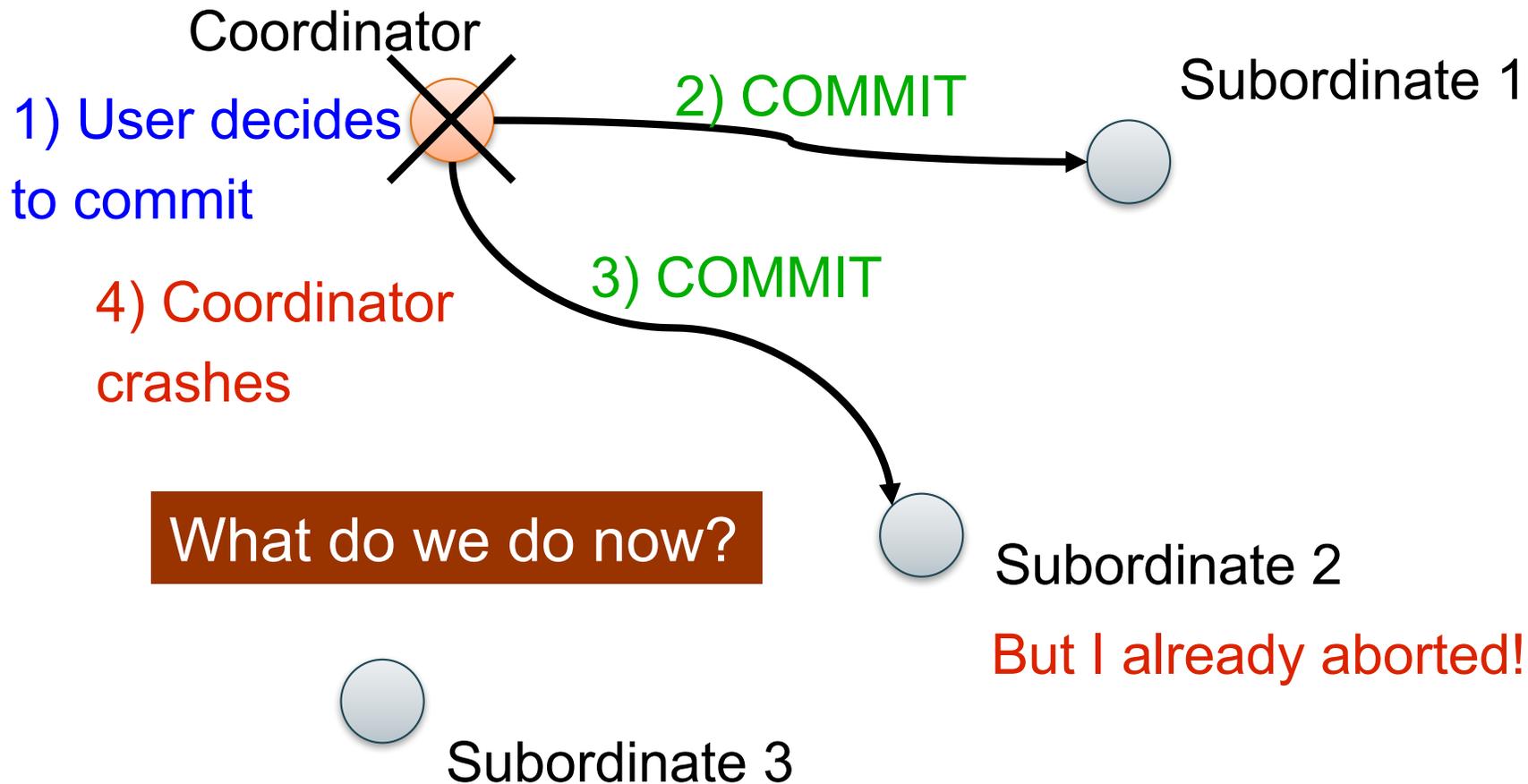
Distributed Transactions

- Concurrency control
- Failure recovery
 - Transaction must be committed at all sites or at none of the sites!
 - No matter what failures occur and when they occur
 - Two-phase commit protocol (2PC)

Distributed Concurrency Control

- In theory, different techniques are possible
 - Pessimistic, optimistic, locking, timestamps
- In practice, distributed two-phase locking
 - Simultaneously hold locks at all sites involved
- Deadlock detection techniques
 - Global wait-for graph (not very practical)
 - Timeouts
- If deadlock: abort least costly local transaction

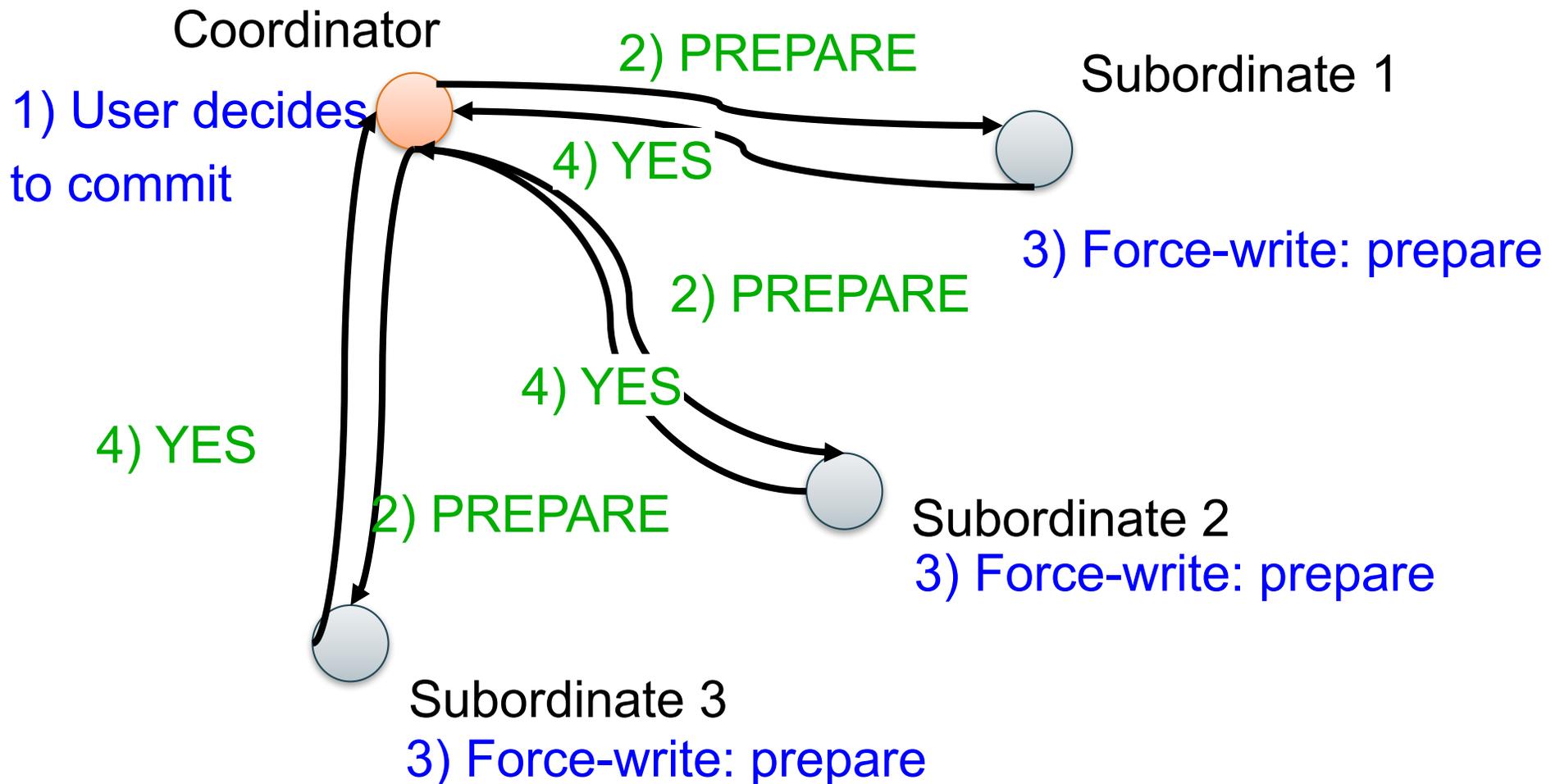
Two-Phase Commit: Motivation



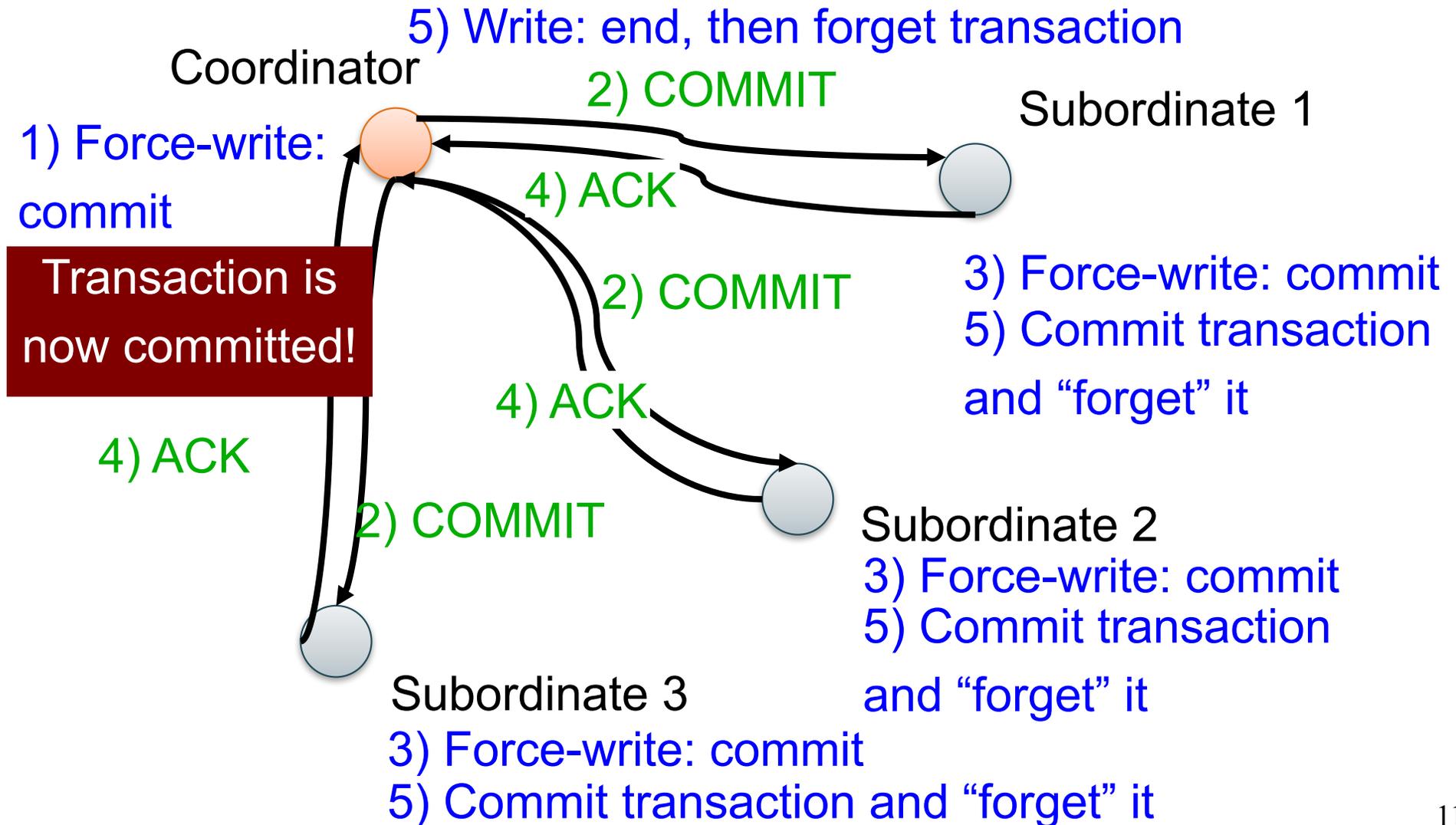
Two-Phase Commit Protocol

- One coordinator and many subordinates
 - Phase 1: prepare
 - All subordinates must flush tail of write-ahead log to disk before ack
 - Must ensure that if coordinator decides to commit, they can commit!
 - Phase 2: commit or abort
 - Log records for 2PC include transaction and coordinator ids
 - Coordinator also logs ids of all subordinates
- Principle
 - Whenever a process makes a decision: vote yes/no or commit/abort
 - Or whenever a subordinate wants to respond to a message: ack
 - **First force-write a log record** (to make sure it survives a failure)
 - **Only then send message about decision**

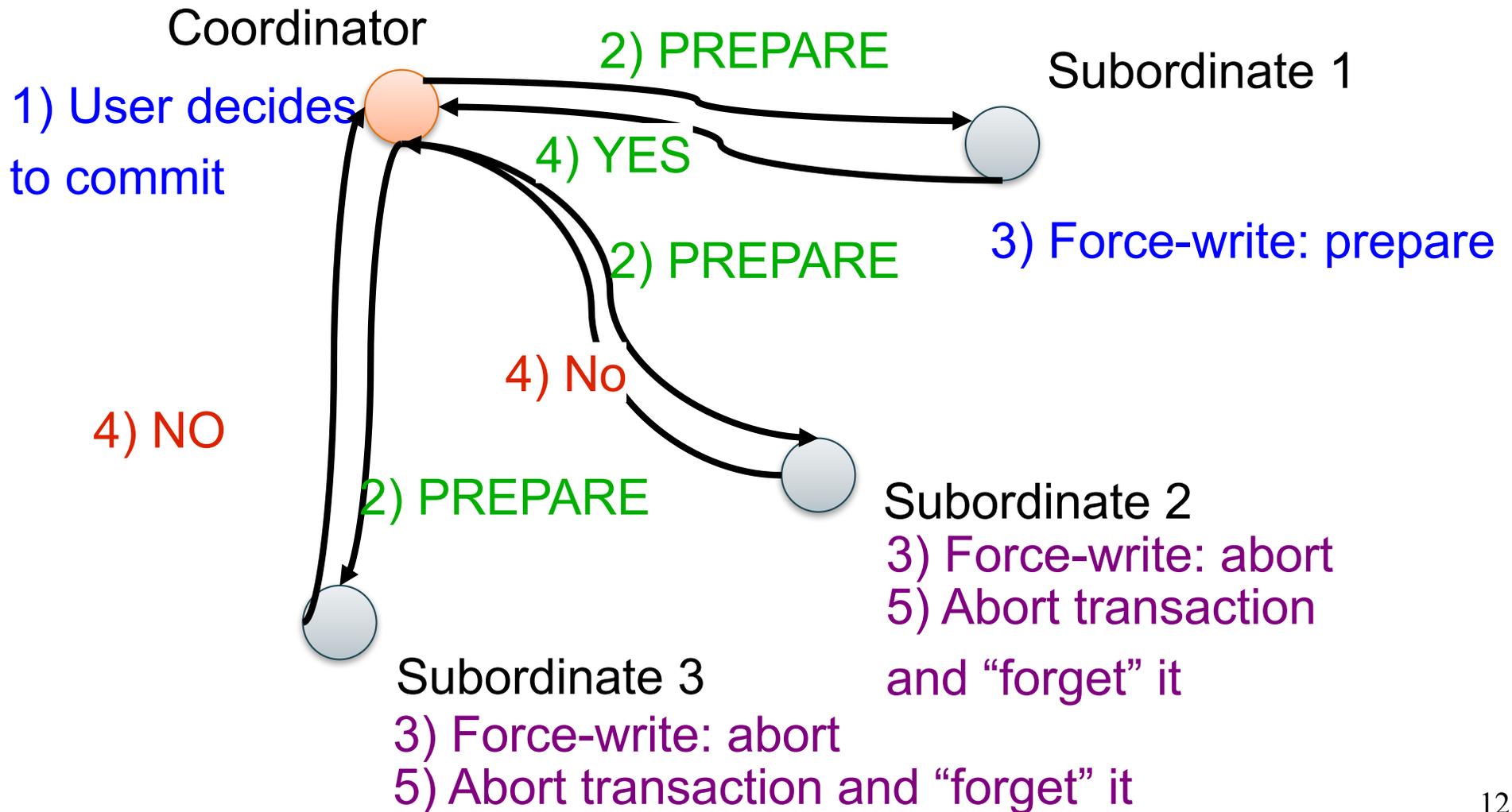
2PC: Phase 1, Prepare



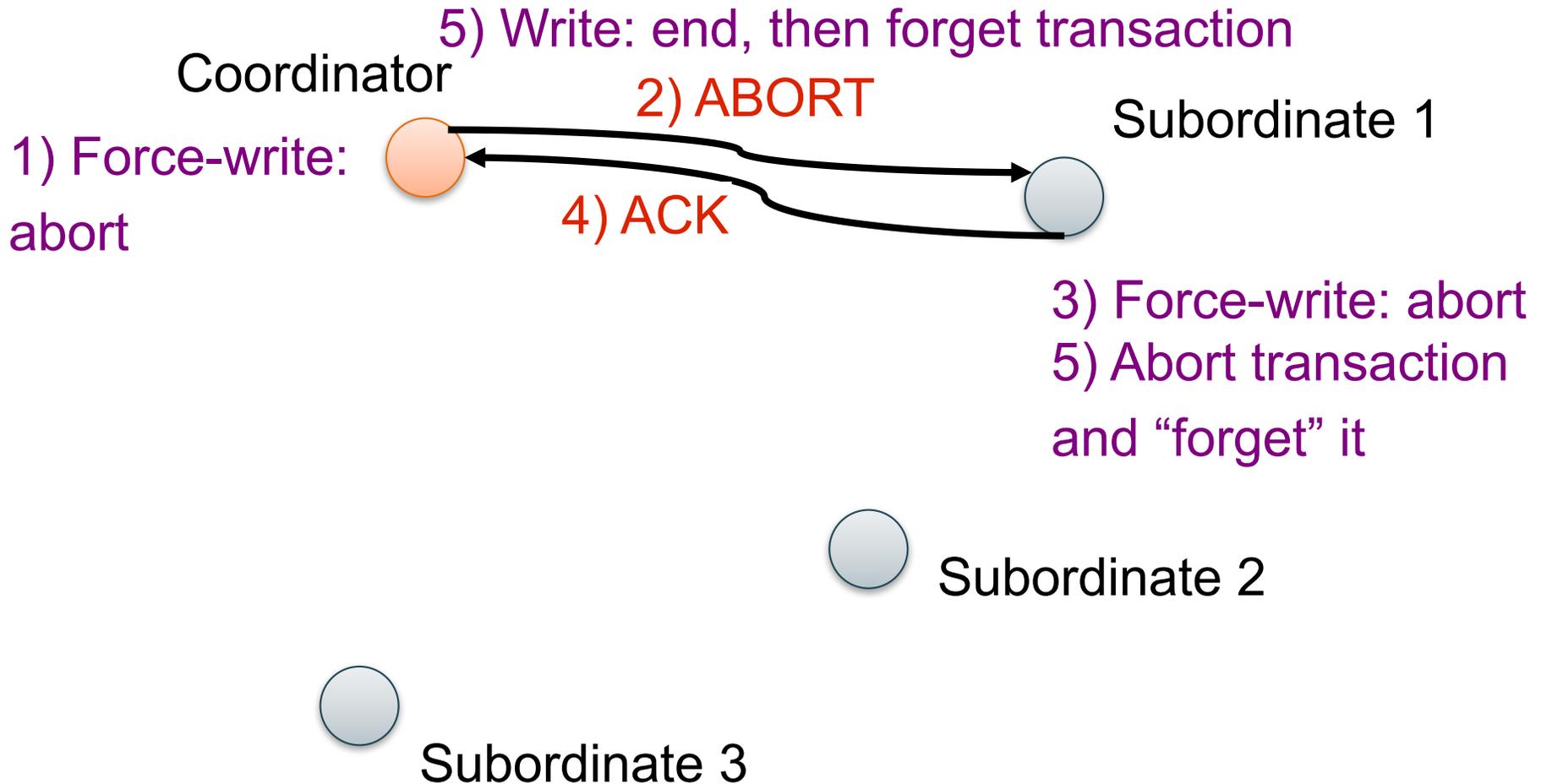
2PC: Phase 2, Commit



2PC with Abort

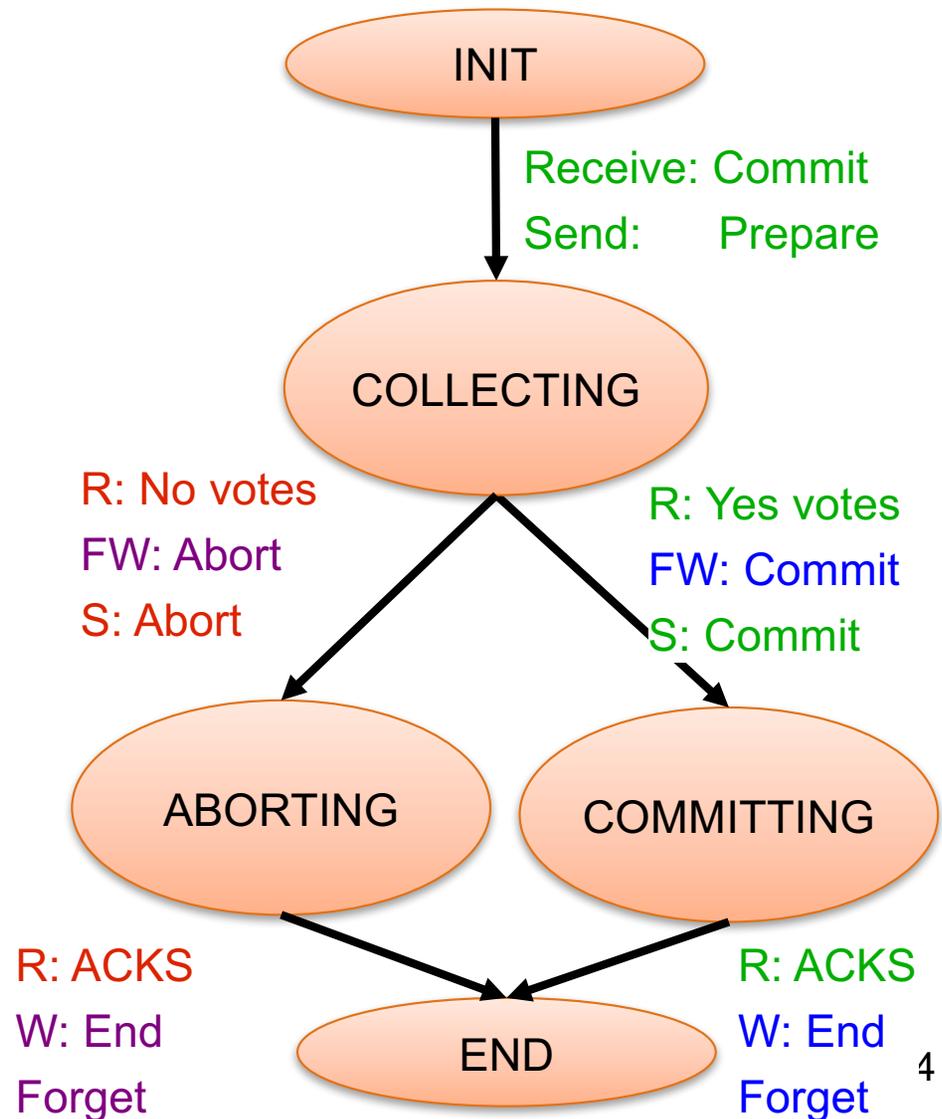


2PC with Abort



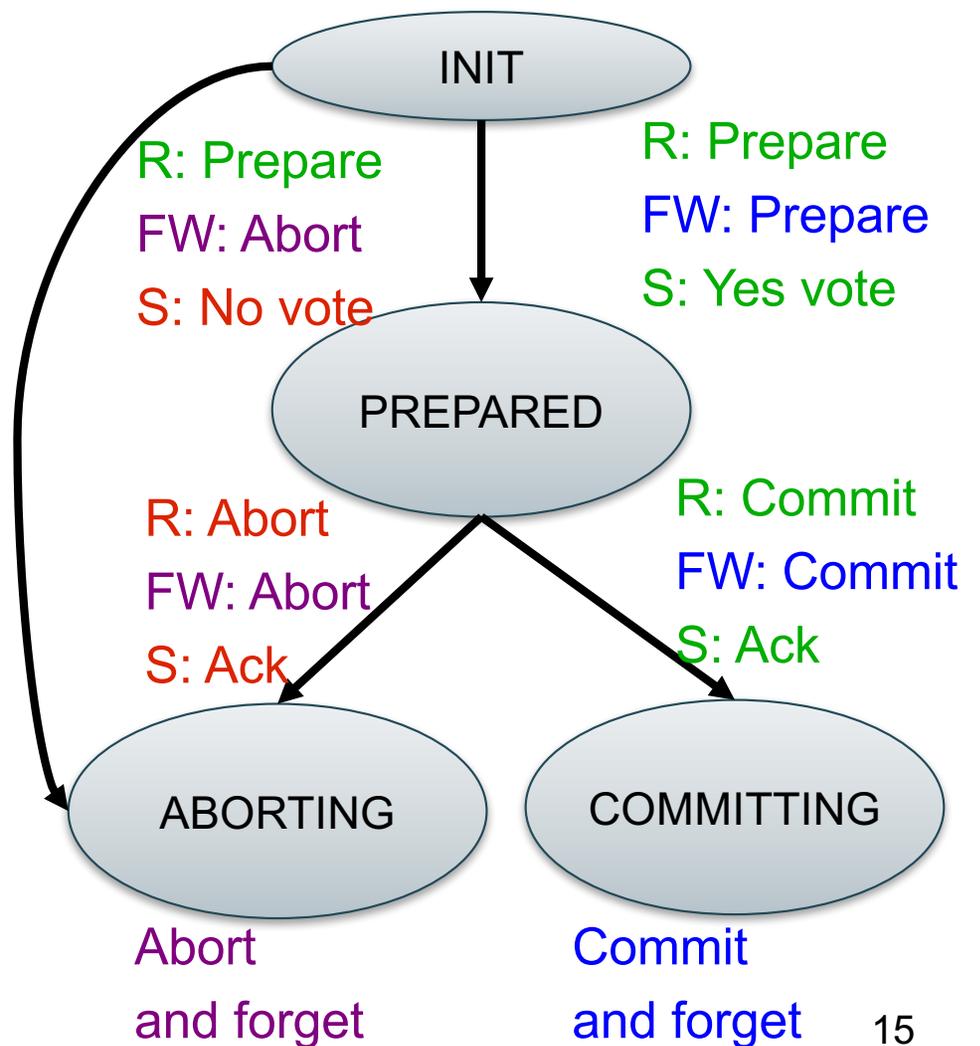
Coordinator State Machine

- All states involve **waiting** for messages



Subordinate State Machine

- INIT and PREPARED involve waiting



Handling Site Failures

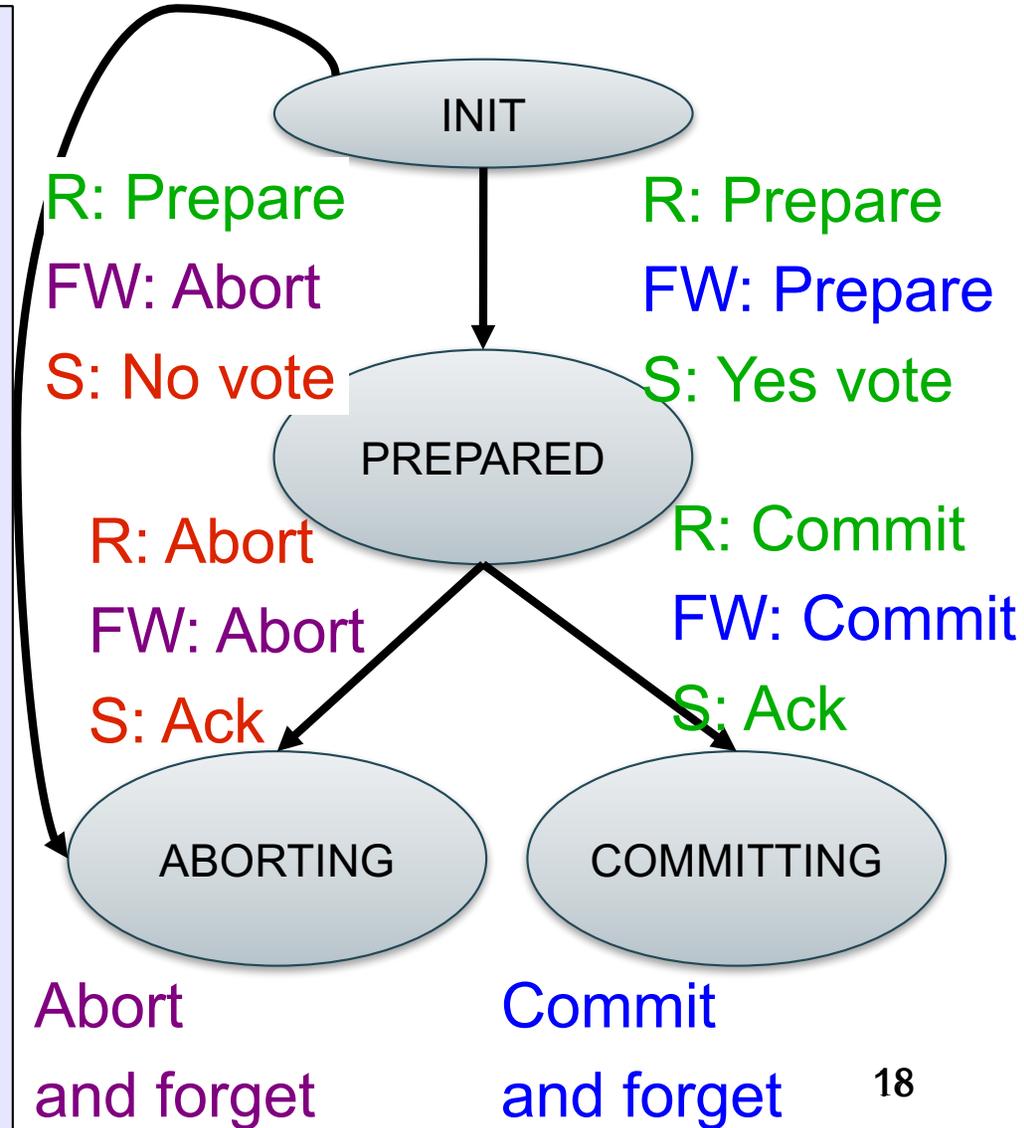
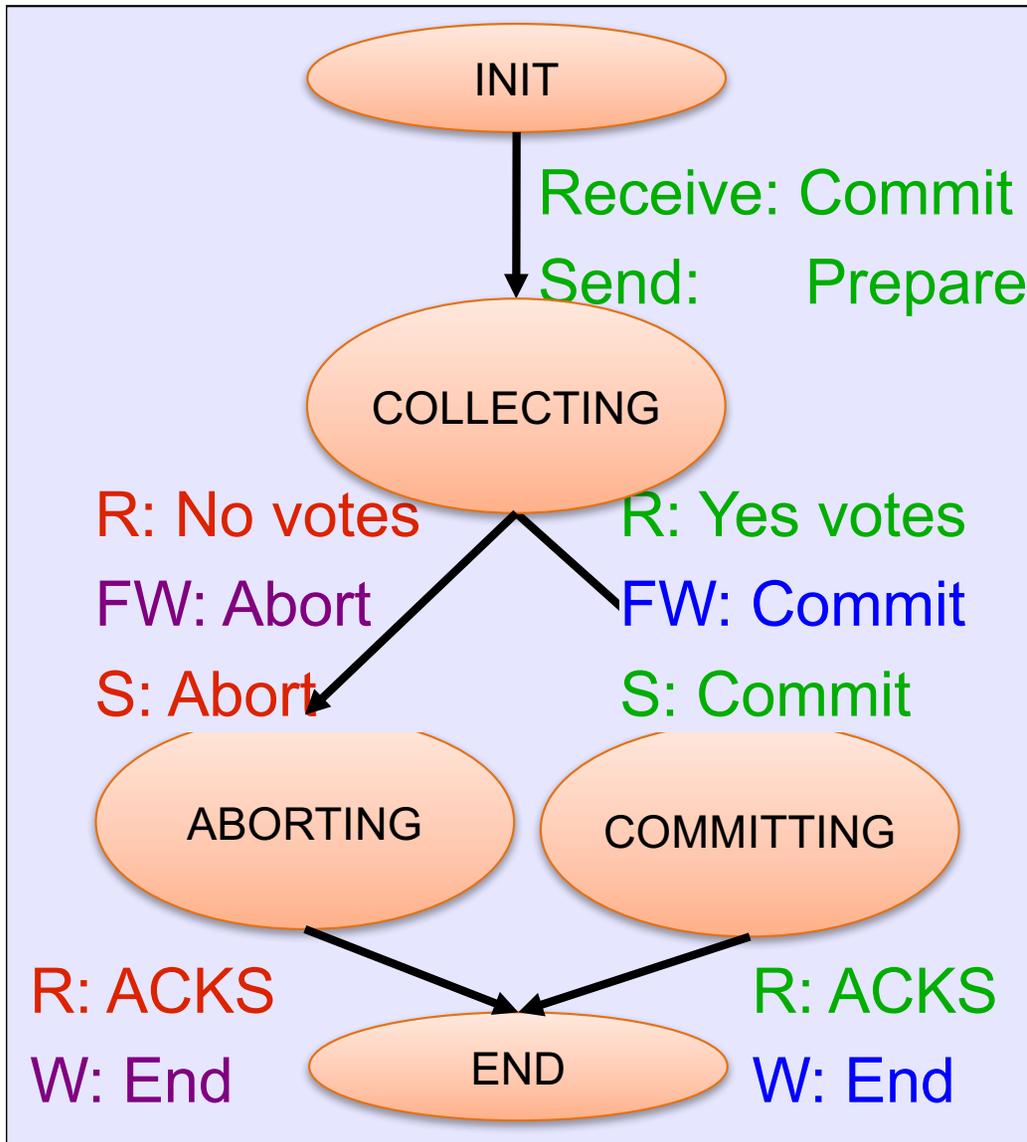
- Approach 1: no site failure detection
 - Can only do retrying & blocking
- Approach 2: timeouts
 - Since **unilateral abort is ok**,
 - Subordinate can **timeout in init** state
 - Coordinator can **timeout in collecting** state
 - **Prepared state is still blocking**
- **2PC is a blocking protocol**

Site Failure Handling Principles

- Retry mechanism
 - In prepared state, periodically query coordinator
 - In committing/aborting state, periodically resend messages to subordinates
- If doesn't know anything about transaction respond “abort” to inquiry messages about fate of transaction
- If there are no log records for a transaction after a crash then abort transaction and “forget” it

Site Failure Scenarios

Examples on the board (please take notes)



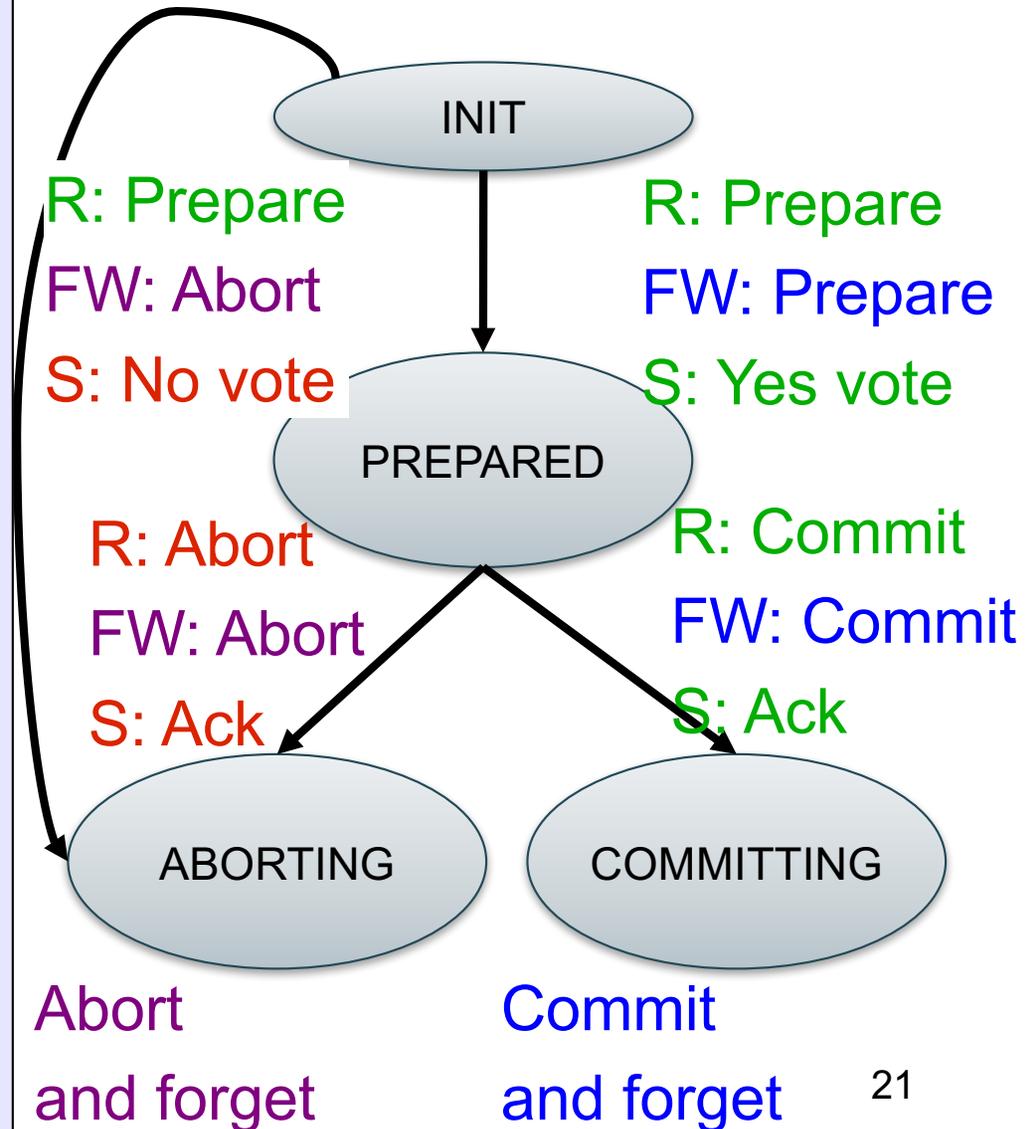
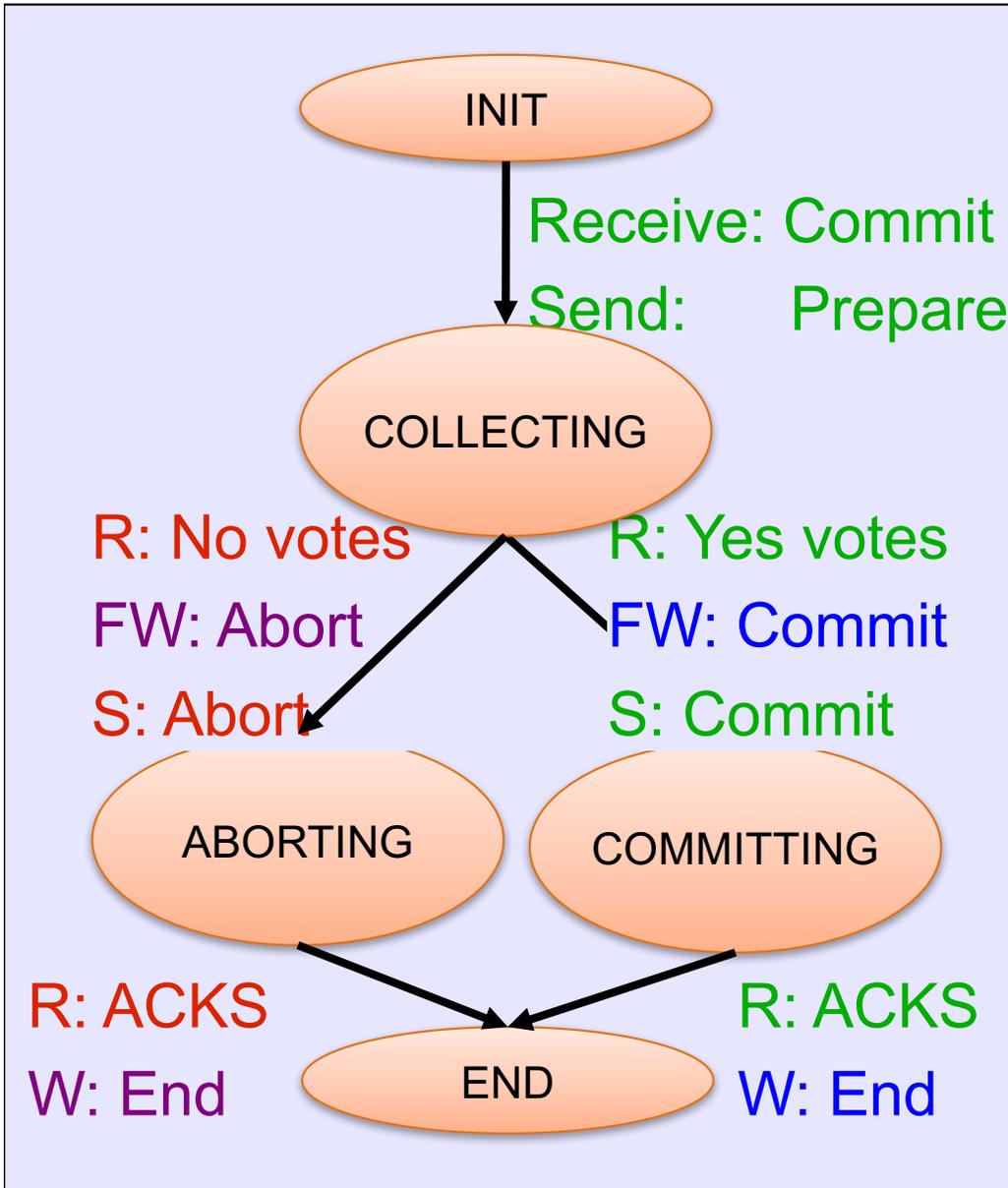
Observations

- Coordinator keeps transaction in transactions table until it receives all acks
 - To ensure subordinates know to commit or abort
 - So acks enable coordinator to “forget” about transaction
- After crash, if recovery process finds no log records for a transaction, the transaction is presumed to have aborted
- Read-only subtransactions: no changes ever need to be undone nor redone

Presumed Abort Protocol

- Optimization goals
 - Fewer **messages** and fewer **force-writes**
- Principle
 - If nothing known about a transaction, assume ABORT
- Aborting transactions need no force-writing
- Avoid log records for **read-only transactions**
 - Reply with a **READ** vote instead of **YES** vote
- Optimizes read-only transactions

2PC State Machines (repeat)



Presumed Abort State Machines

