CS 582: Distributed Systems

Consistency and Consensus

Dr. Zafar Ayyub Qazi

Fall 2024

Acknowledgements: Slides use some material from Wyatt Lloyd (Princeton University)

Today's Agenda

- Consensus
- Consistency Protocols

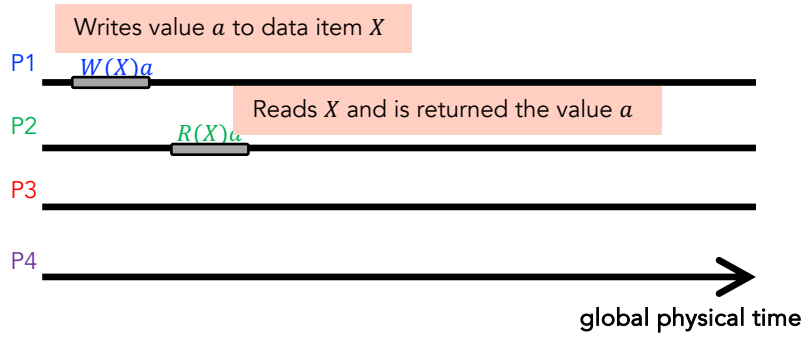
Specific learning outcomes

By the end of today's lecture, you should be able to:

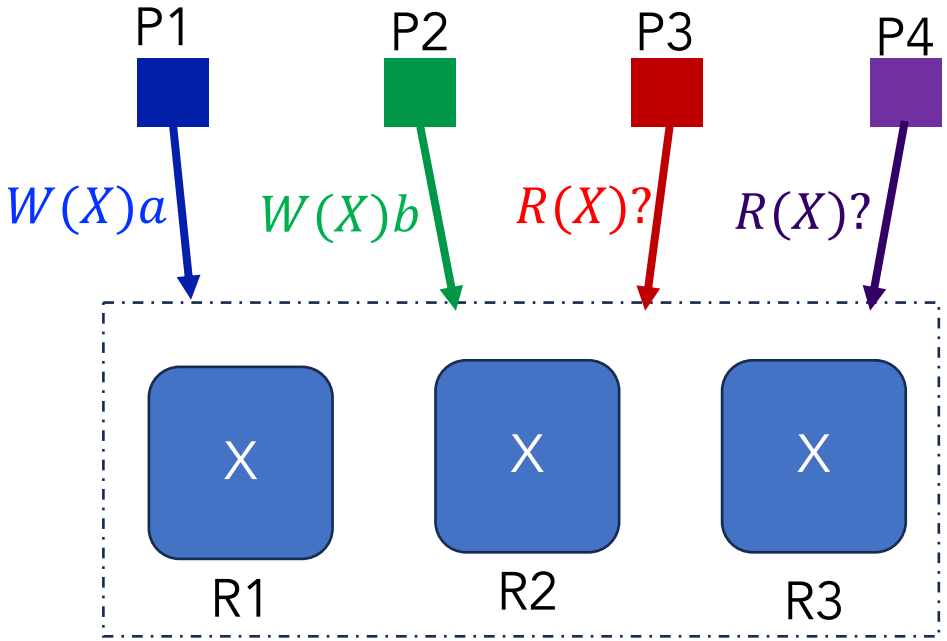
- ❑ Explain what is consensus and the key requirements for consensus
- ❑ Describe the key insight of the FLP result
- ❑ Explain what is *safety* and *liveness* in the context of consensus
- ❑ Analyze the different types of consistency protocols
- ❑ Explain how 2-PC works
- ❑ Analyze 2-PC in terms of safety and liveness

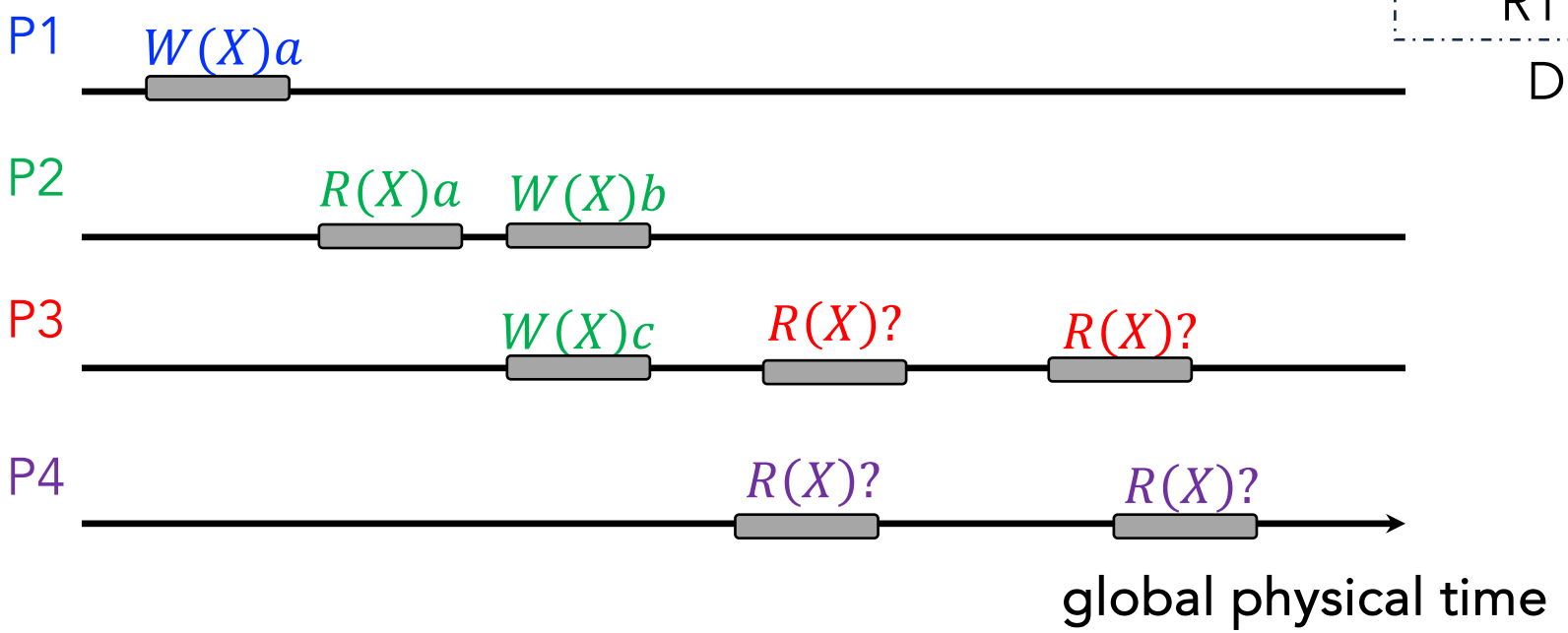
Recap: Consistency Models

- Linearizability
- Sequential consistency
- Casual consistency
- Eventual Consistency



Recap: Example





Recap: Causal Consistency

- Causal consistency is strictly weaker than sequential consistency
  - If system is sequentially consistent → it is also causally consistent
- BUT: it also offers more possibilities for concurrency
  - Concurrent operations (which are not causally-dependent) can be executed in different orders by different replicas
  - In contrast to sequential consistency, we do not need to enforce a global ordering of all operations
  - Hence, one can get better performance than sequential consistency

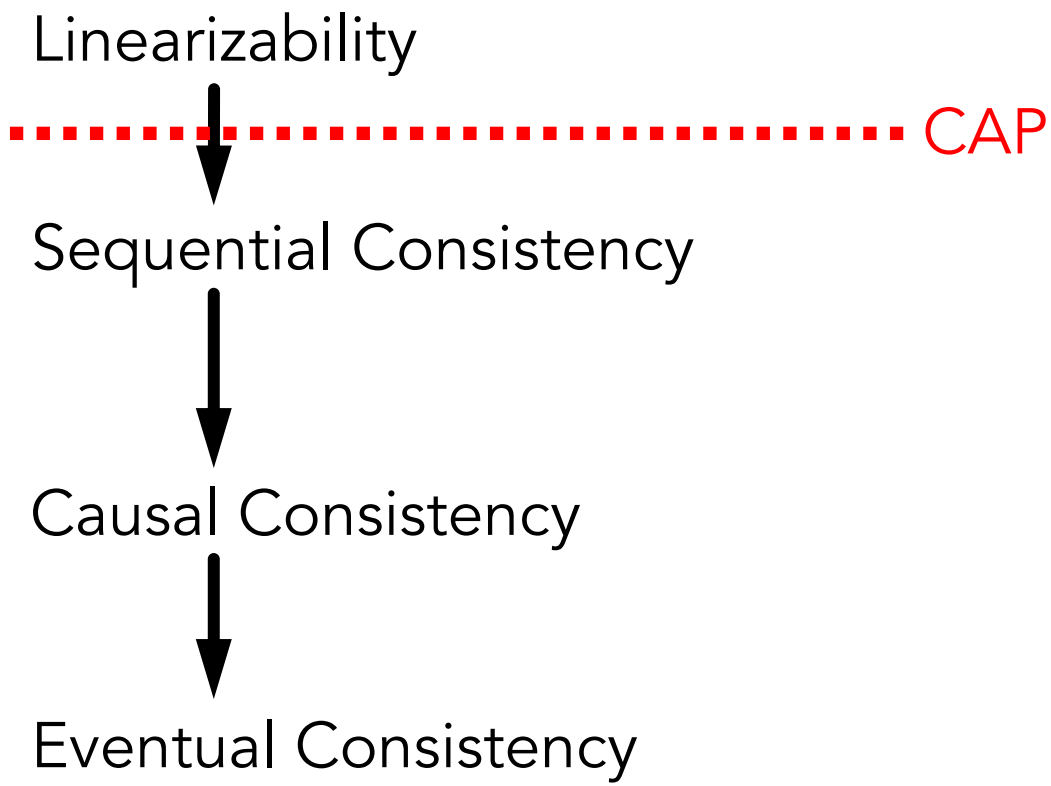
Recap: Eventual Consistency

- Allow divergent replicas
- Allow reads to see stale or conflicting data
- Resolve multiple versions when failures go away
- **Eventually** the replicas in the system reach a convergent state

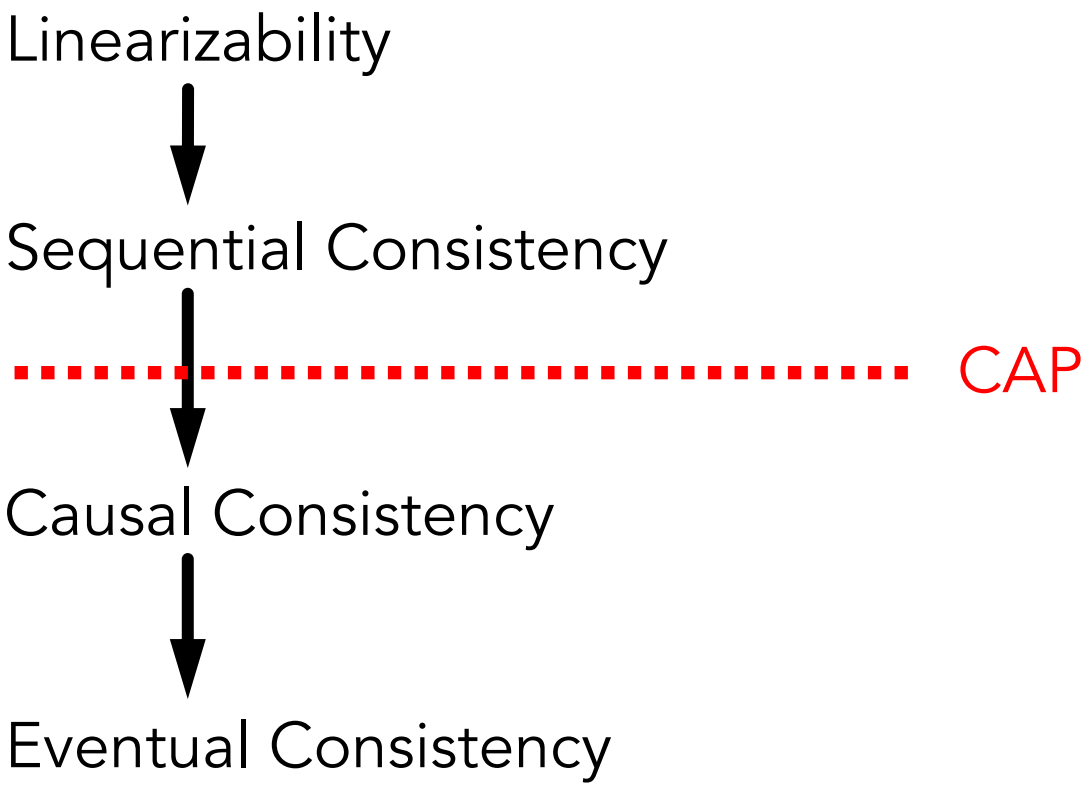
Recap: Why Eventual Consistency?

- Support disconnected operations or network partitions
  - Better to read a stale value than nothing
  - Better to save writes somewhere than nothing
- Support for increased parallelism
- Issues
  - Potentially anomalous application behavior
  - Stale reads and conflicting writes…

Consistency Hierarchy



Consistency Hierarchy



Consistency Models

Linearizable

Sequential

Causal

Eventual

Unavailable

Not available during network partitions. Replicas will have to pause operations to ensure safety.

Sticky Available

Available on every non-faulty node, as long as clients only talk to the same replicas, instead of switching to new ones

Totally Available

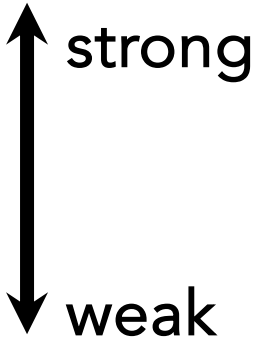
Available on every non-faulty node, even when there is a network partition

Other Consistency Models

- Serializability
- External consistency
- Read your writes
- …
- Read the book from Tanenbaum, Chapter 7.3, if interested

Summary: Consistency Models

- Consistency models
  - Linearizability
  - Sequential consistency
  - Casual consistency
  - Eventual consistency



Variations in:
- Ordering of writes
- Staleness of reads

- Linearizability and sequential consistency:
  - Same sequence of updates at all replicas
  - All replicas agree on the order of the updates

Consensus

Consensus

- General Definition:
  - A general agreement about something

Examples of Consensus in Distributed Systems

- Group of servers attempting to:
  - Make sure all servers in the group execute the **same updates in the same order**
  - Elect a leader in the group, and inform everybody, **so they agree on who the leader is**
  - Maintain own lists of who is a current member of the group, and update lists when somebody leaves/fails, and inform others, **so they agree on who has failed**

Whats common in these problems?

Whats common in these problems?

- All of servers are attempting to coordinate with each other and agree on the value of something
  - The ordering of messages
  - Who the leader is
  - The up/down status of a suspected failed process

Key Requirements for Consensus

- Given a set of processors, each with an initial value:
  - Termination: All non-faulty processes decide on a value
  - Agreement: All processes that decide, do so on the same value
  - Validity: The value that has been decided must have been proposed by some process

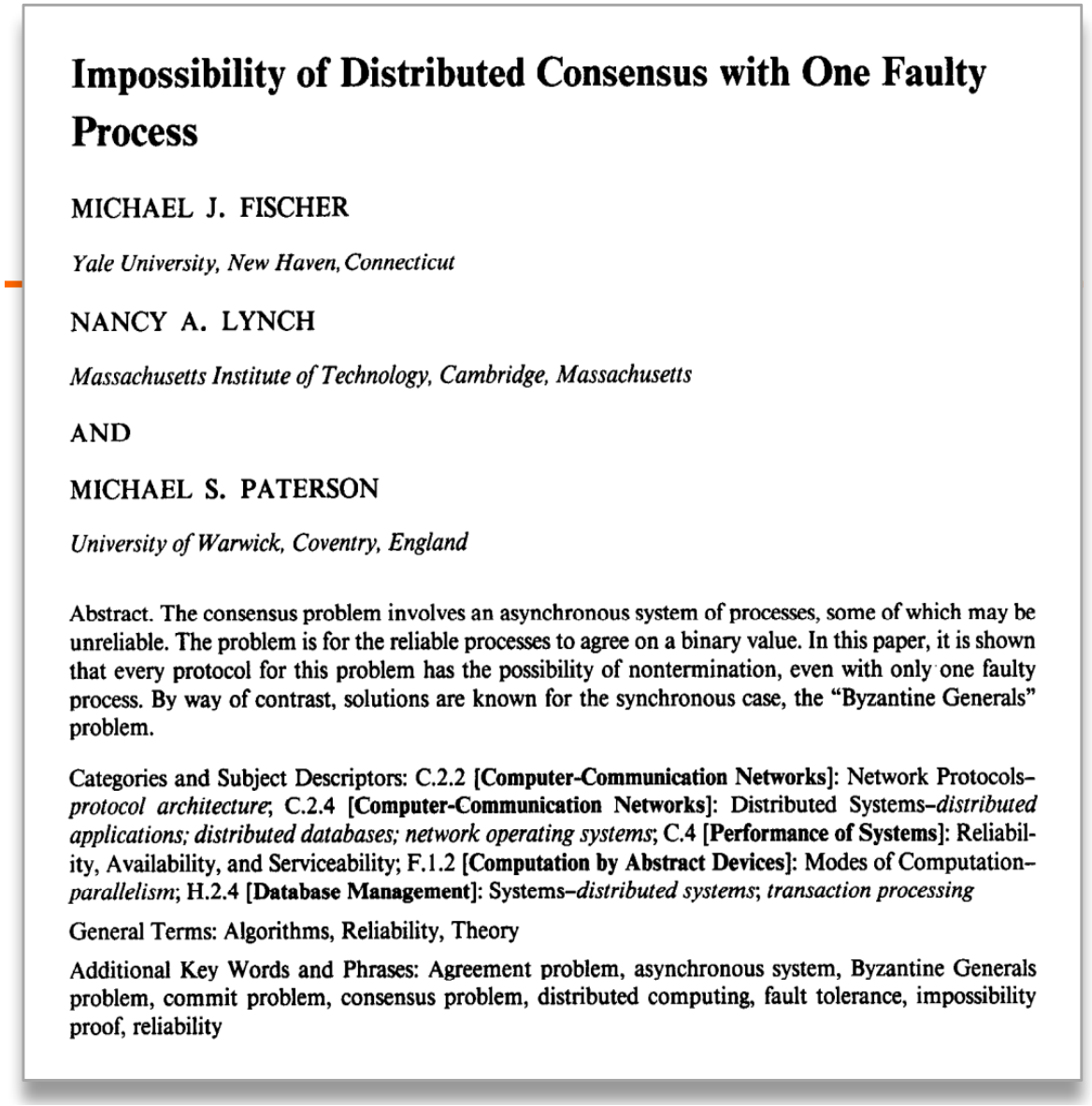
Is Consensus Possible?

Is Consensus Possible?

- In the synchronous system model
  - Consensus is solvable
- In the asynchronous system model
  - Consensus is impossible to solve
  - Whatever algorithm you suggest, there is always a worst-case possible execution (with failures and messages delays) that prevents the system from reaching consensus
  - More specifically, both agreement & termination can not be guaranteed

FLP Result

- No deterministic 1-crash-robust consensus algorithm exists for the asynchronous model
- Key idea of the proof:
  - It is impossible to distinguish a failed process from one that is just very very slow. Hence, the rest of the processes may stay ambivalent (forever) when it comes to deciding.



Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols–*protocol architecture*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation–*parallelism*; H.2.4 [**Database Management**]: Systems–*distributed systems*; *transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

Two Important Classes of Properties

1. Safety
   - Guarantee that sth bad will never happen in the system

- Example
  - If one server commits an update, no other replica rejects it
    - If one rejects it, no one commits

Two Important Classes of Properties

2. Liveness
   - Guarantee that something good will happen, eventually
   - Does not imply a time bound, but if you allow the system to run long enough …

- Example
  - Update will eventually be committed

- FLP: Can't guarantee both liveness and safety in an asynchronous distributed system given failures

Summary so far

- To implement linearizability/sequential consistency
  - All replicas must execute updates in the same sequence
  - This in turn requires agreement btw replicas
  - This is requires consensus

- FLP Result: No deterministic 1-crash-robust consensus algorithm exists for asynchronous model
  - More specifically, both safety & liveness can not be guaranteed

Design Implications

- Let's say you want to design a system that can provide linearizability or sequential consistency.

- Both these models require some total order, which in turn requires nodes to agree on global order, i.e., reach consensus

- How do you go about doing this given the FLP result?

Consensus Goals

- Paxos, Raft, etc., aim to guarantee Safety
  - While providing Liveliness under certain assumptions
    - As long as any majority of nodes are up and can communicate with each other with reasonable timeliness

Driving question for the next few classes

How can we get nodes to agree or reach consensus given FLP result?

Homework Problem

- Can we use totally ordered multicast with Lamport Clocks or Vector Clocks to provide linearizability or sequential consistency?

- Analyze the safety and liveness properties of these protocols

Types of Consistency Protocols

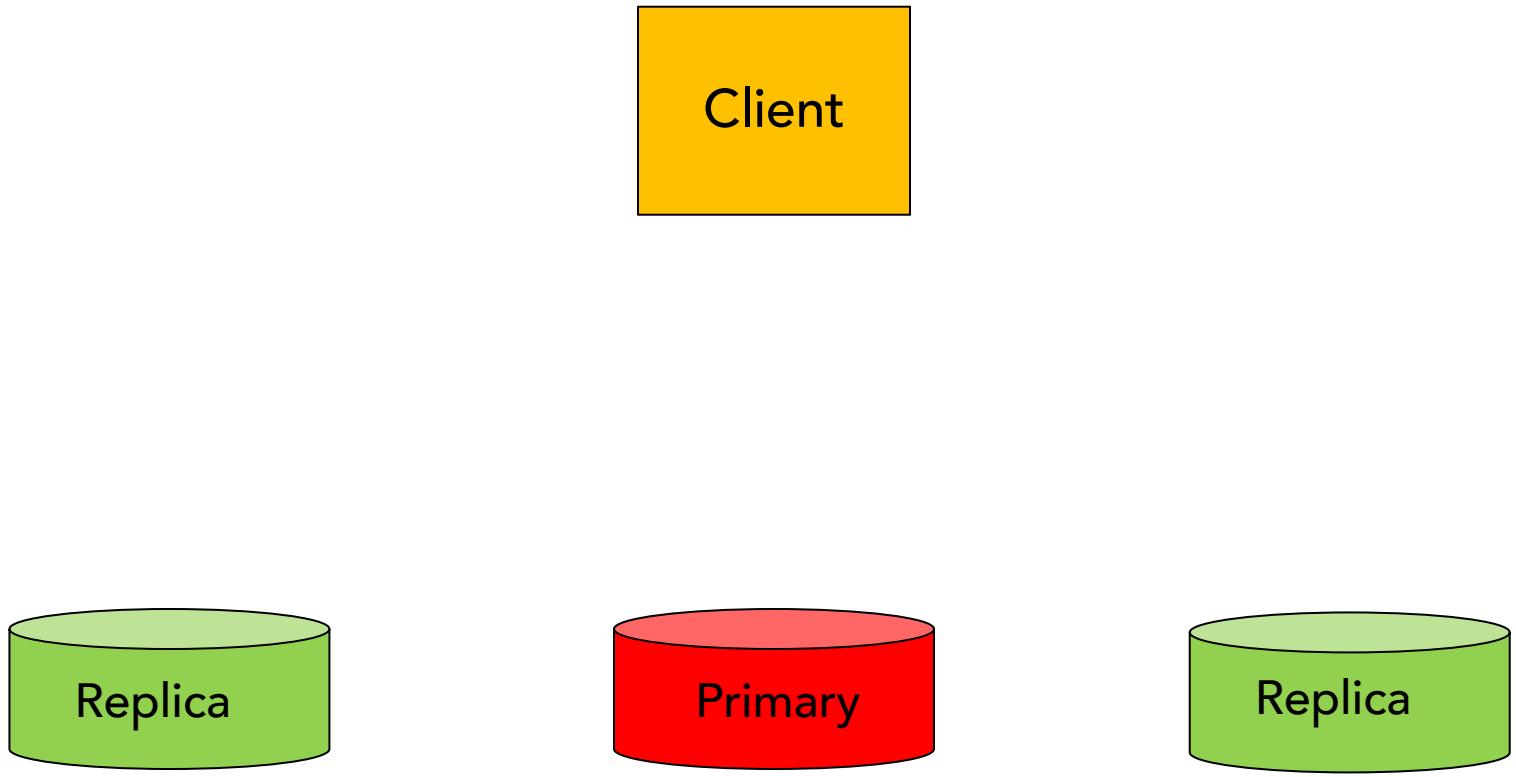
Primary-backup protocols

Primary-backup protocols

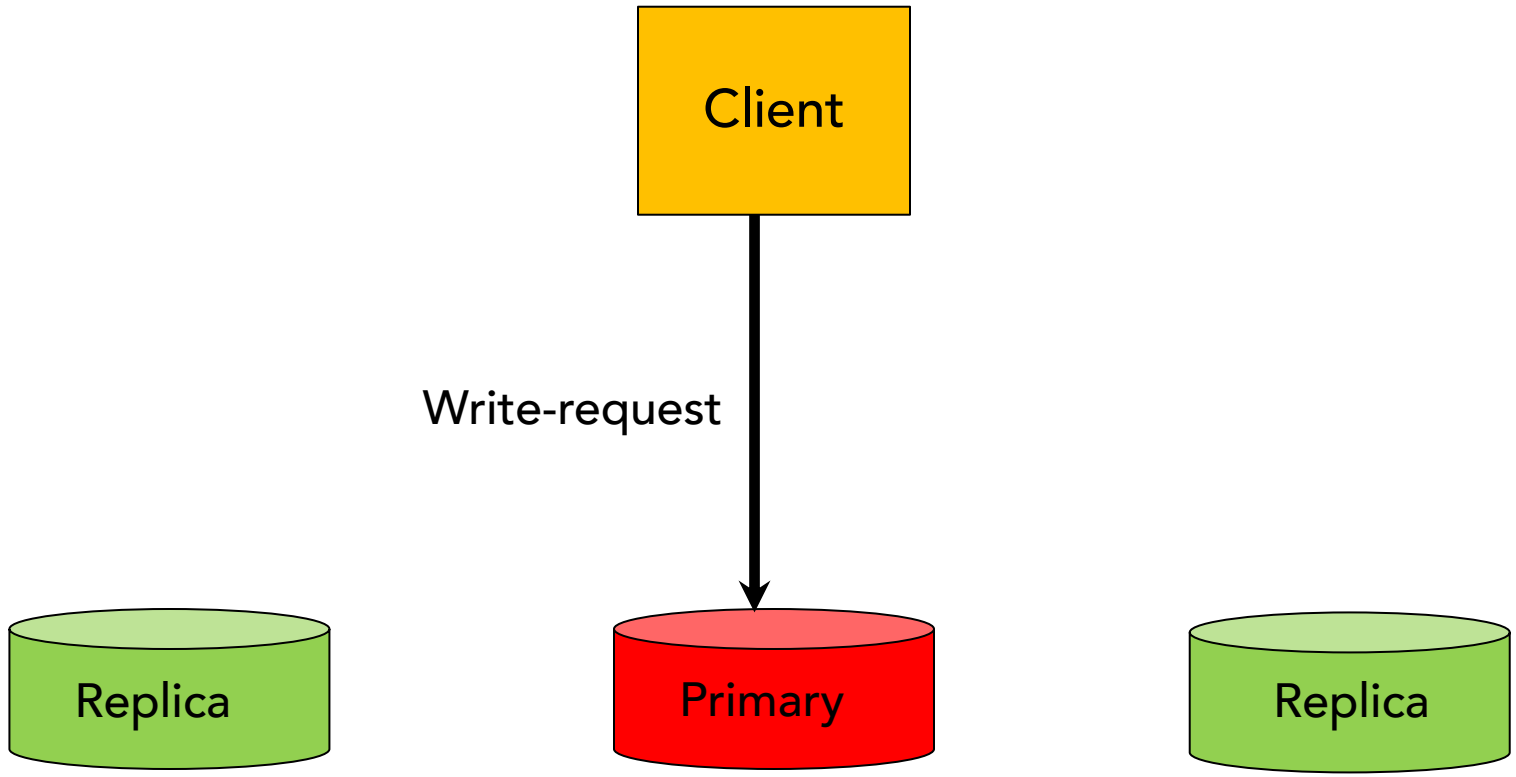
- One special node (primary) orders requests
- Route all updates through the primary server
  - Assigns an order to the updates
  - All replicas commit updates in the assigned order
  - Simple to implement

Primary-backup protocol

Assume:
Nodes fail only by crashing but can come back (recover) after a failure



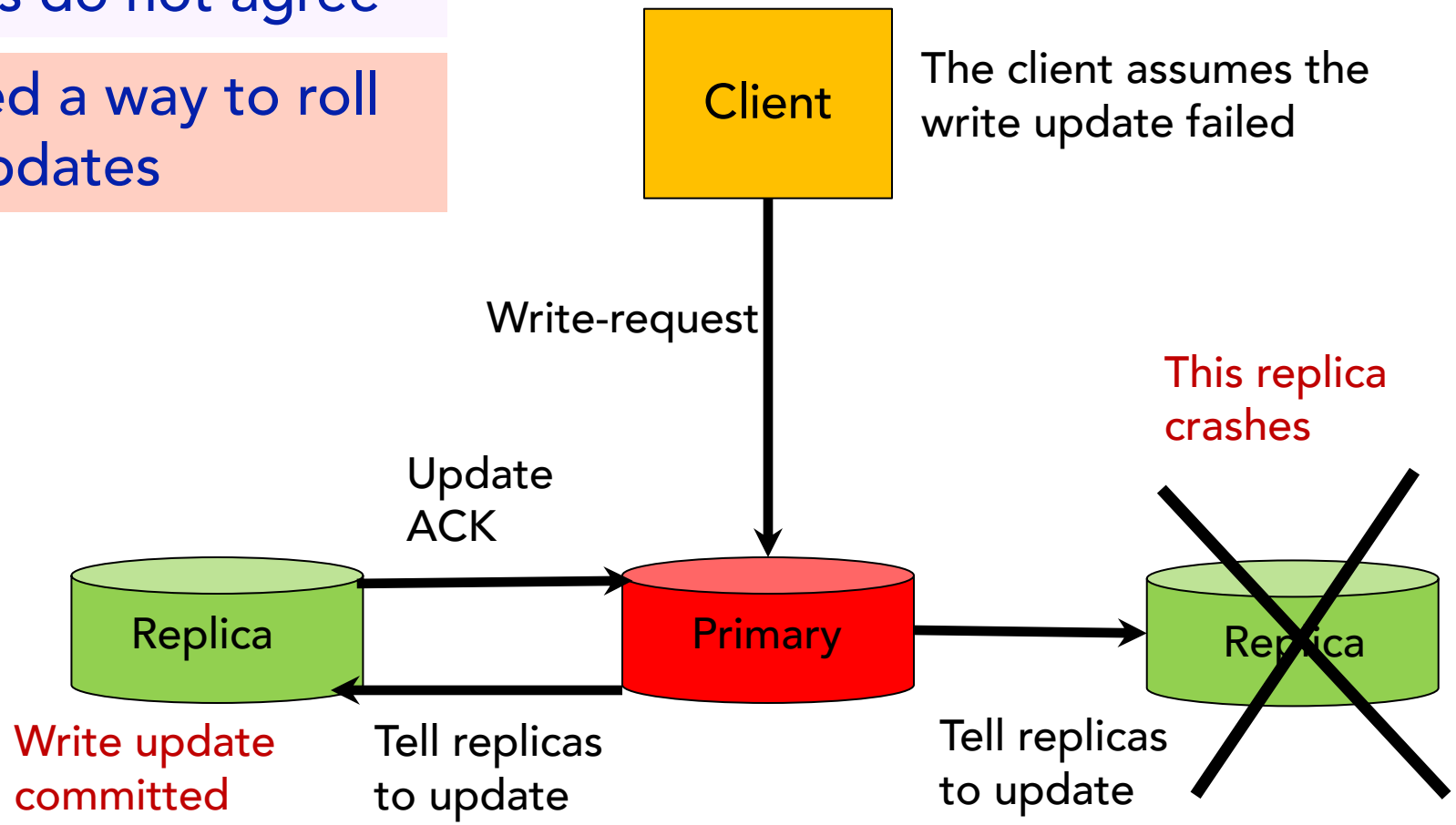
Primary-backup protocol



Failure Scenario

Replicas do not agree

We need a way to roll back updates



Two Phase Commit (2-PC)

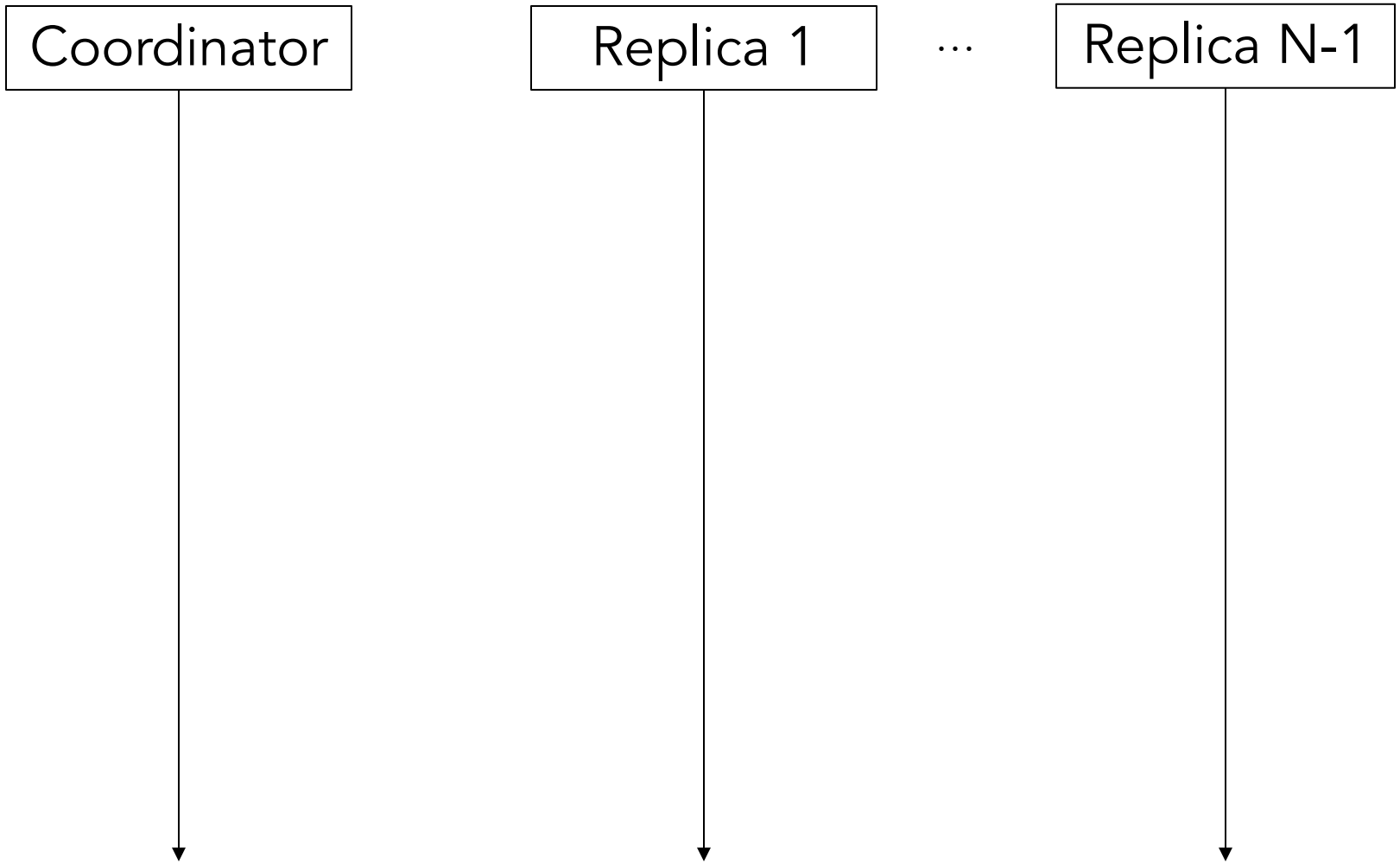
- Consists of two distinct phases
- Used in several distributed systems
- Also used in Google Spanner

Key idea

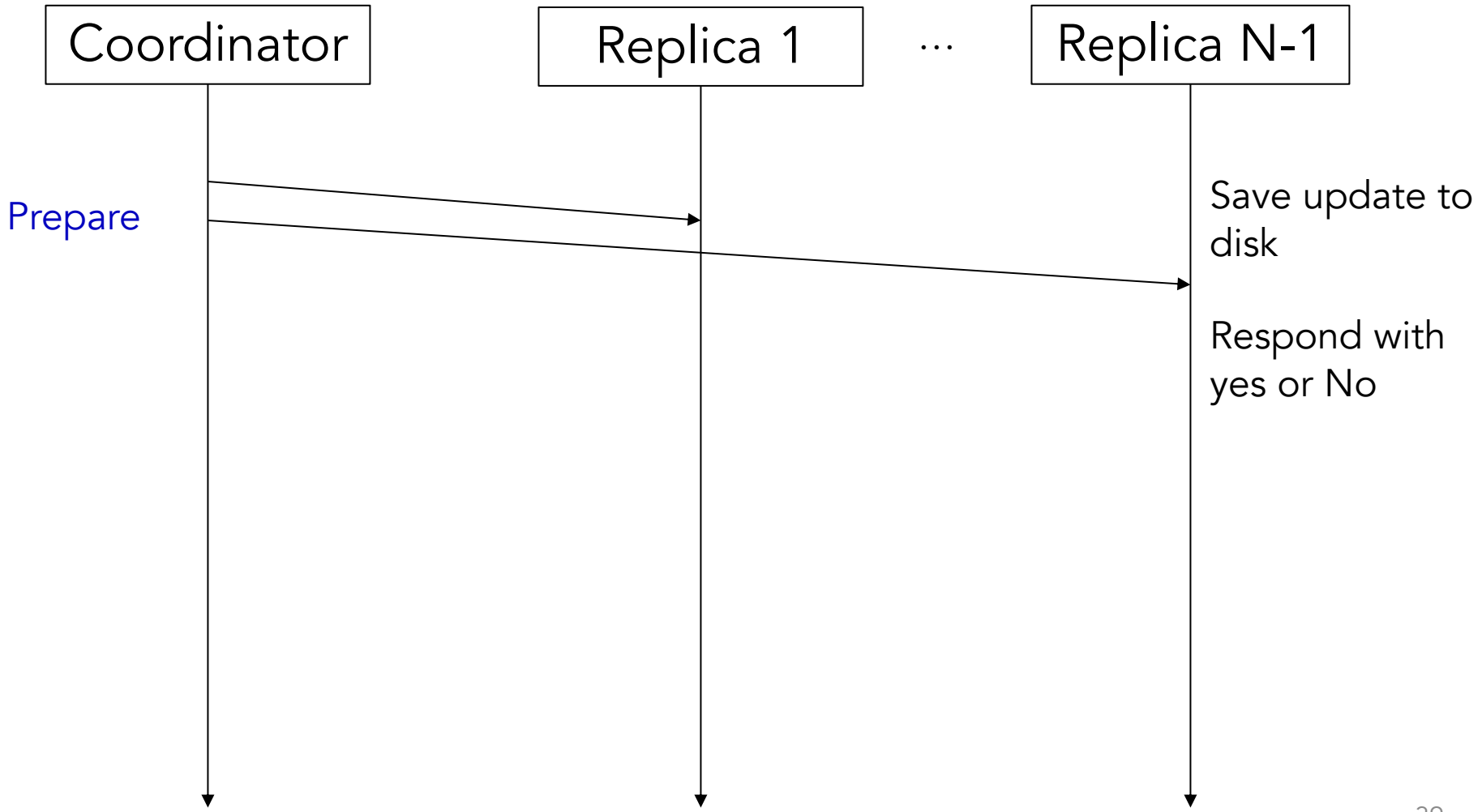
- Allow the system to roll back updates on failures
  - By using two phases

- This is in contrast to single-phase primary-based protocols
  - Where there is no step for rolling back an operation that has failed on some nodes and succeeded on other nodes

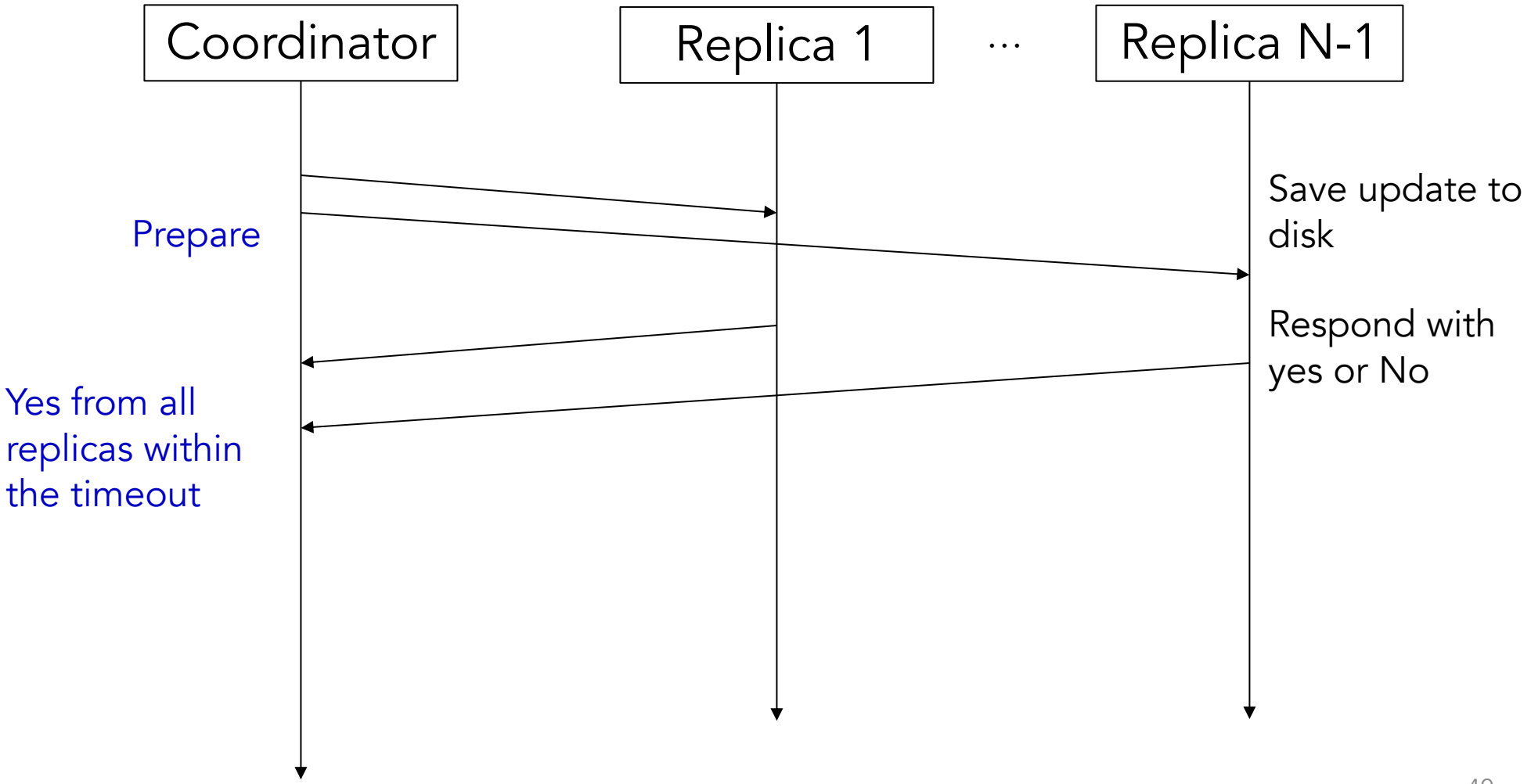
2-PC



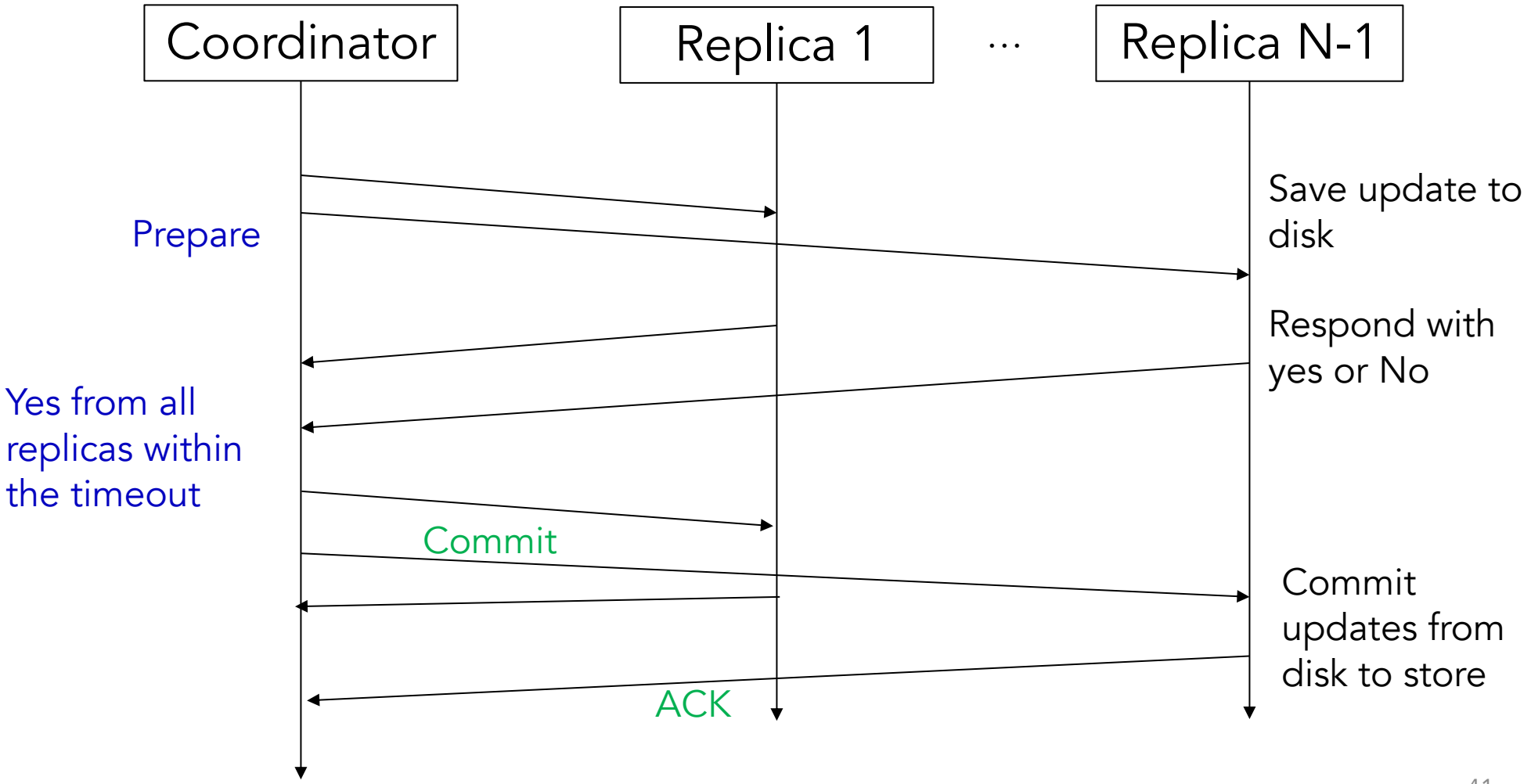
2-PC


Coordinator
Replica 1
…
Replica N-1
Prepare
Save update to disk
Respond with yes or No

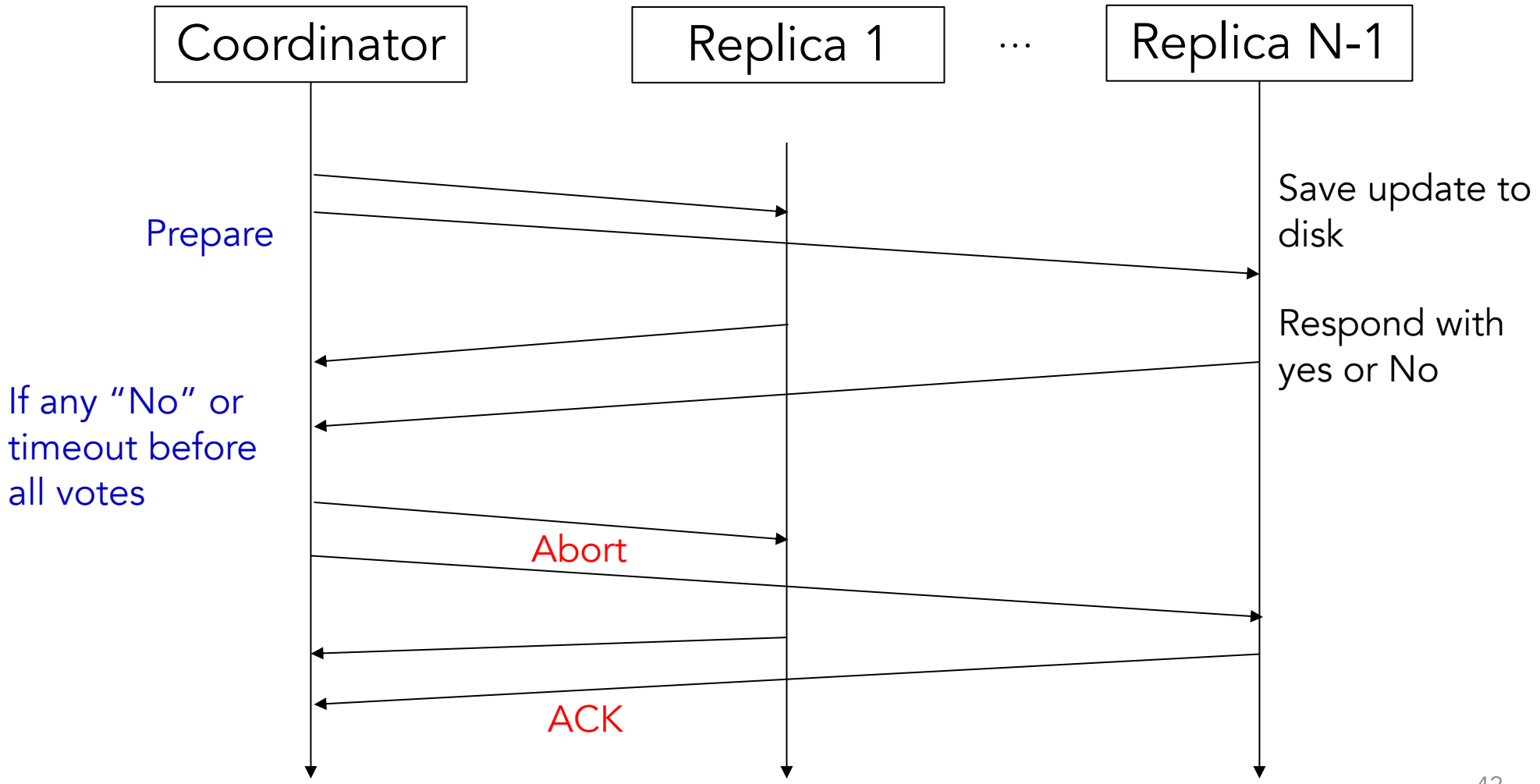
2-PC


Coordinator
Replica 1
…
Replica N-1
Prepare
Save update to disk
Respond with yes or No
Yes from all replicas within the timeout

2-PC


Coordinator
Replica 1
…
Replica N-1
Prepare
Save update to disk
Respond with yes or No
Yes from all replicas within the timeout
Commit
Commit updates from disk to store
ACK

2-PC


Coordinator
Replica 1
…
Replica N-1
Prepare
Save update to disk
Respond with yes or No
If any "No" or timeout before all votes
Abort
ACK

Failures in 2-PC

- To deal with replica crashes
  - Each replica saves tentative updates into permanent storage, right before applying Yes/No in the first phase.
  - Retrievable after crash recovery

- To deal with coordinator crashes
  - Coordinator logs all decisions and received/sent messages on disk
  - After recovery or new election → new coordinator takes over

Safety and Liveness of 2-PC

- Safety: All hosts that decide reach the same decision
  - No commit unless everyone says “yes”
- Liveness: If no failures and all say “yes” then commit
  - But if failures then 2PC might block
- Doesn’t tolerate faults well: must wait for repair
  - Failure of any process can result in non-progress

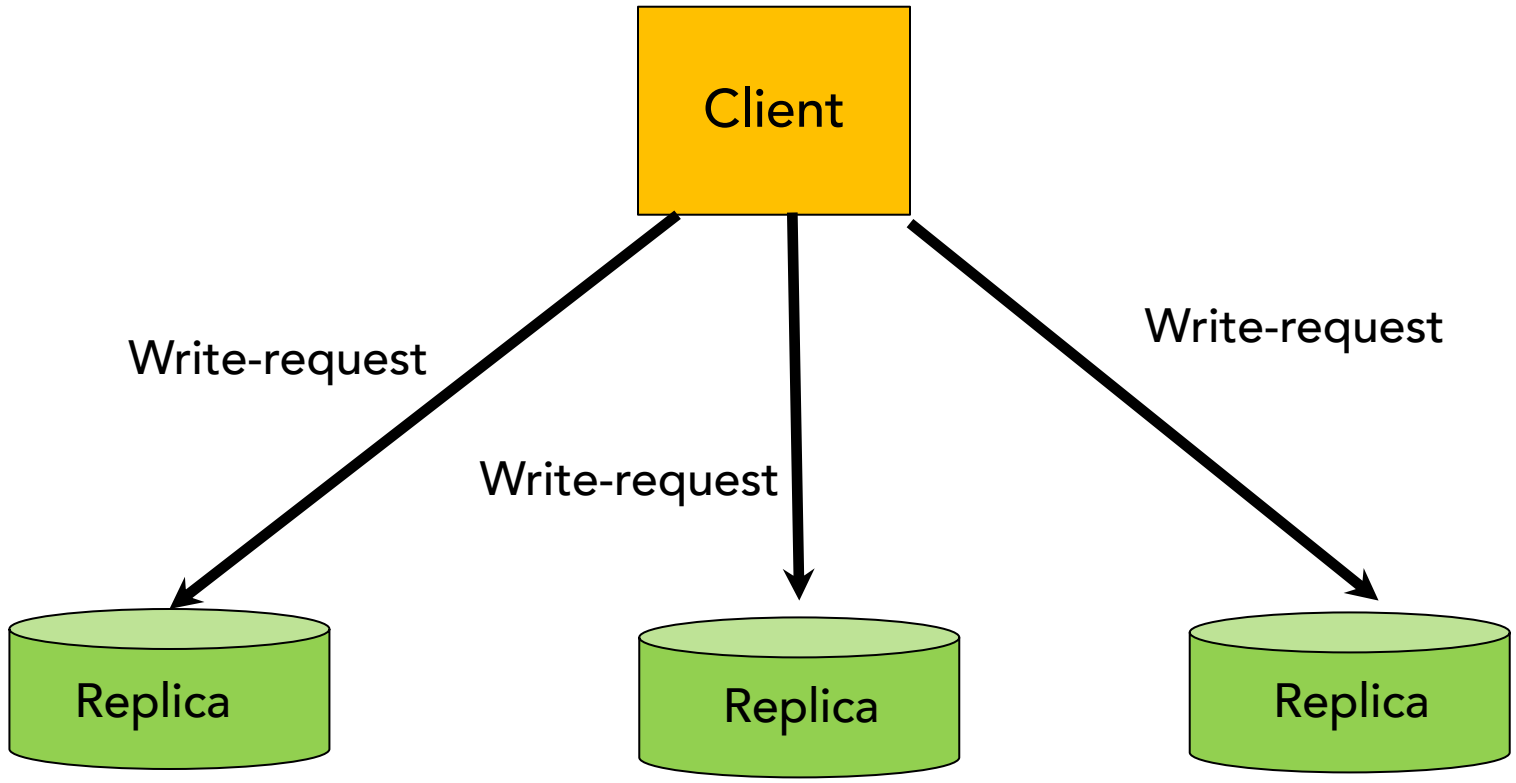
In Conclusion …

- Single phase primary-backup schemes
  - Safety: Can't roll back updates, so safety can get violated
  - Liveness: doen't provide liveness with failures
- Two Phase Commit (2-PC)
  - Safety: Allows for rolling back updates and making nodes consistent
  - Liveness: blocks in case of failures

Replicated-Write Protocol

- Clients send updates to all replicas (instead of only primary)
- Then wait for all or a subset of replicas to reply back

Replicated-Write Protocols



Typical Schemes

- In a system with N replica servers
- Read quorum
  - Clients needs an arbitrary collection of N_R servers
- Write quorum
  - Needs a arbitrary collection of N_w servers
- Conditions that must be satisfied (if you want agreement):
  - $N_w > N/2$
  - $N_R + N_w > N$
  - How do these conditions help?

Next Lecture

- Paxos