

Distributed Consensus

CS 675: Distributed Systems (Spring 2020)

Lecture 5

Yue Cheng

Some material taken/derived from:

- Princeton COS-418 materials created by Michael Freedman and Wyatt Lloyd.
- MIT 6.824 by Robert Morris, Frans Kaashoek, and Nickolai Zeldovich.
- Utah CS6450 by Ryan Stutsman.

Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

Today's outline

1. View changes in primary-backup replication

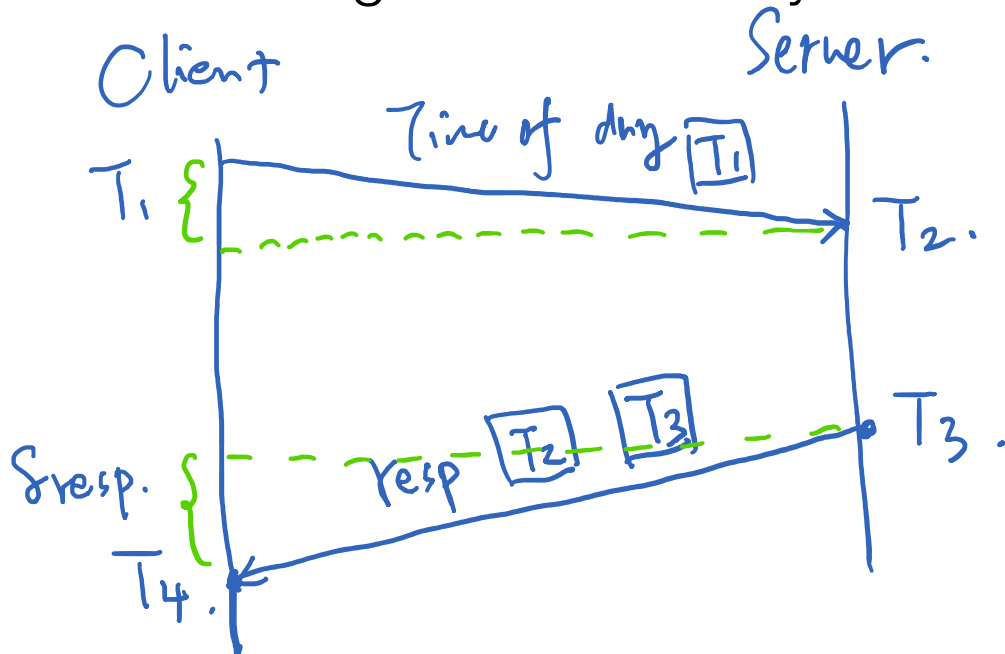
2. Consensus

- Paxos

- Raft

Review: Time & Clocks, PB

- Wall clock drift, so they are all skewed
 - Synchronize to bound skew, but still left with uncertainty
 - NTP sync sometimes sufficient
 - Getting less sub-ms sync challenging due to network



$$\delta = (T_4 - T_1) - (T_3 - T_2)$$

$$T_{sync} = T_3 + \frac{\delta_{resp}}{2}$$

$$\delta_{resp} = \frac{\delta}{2}$$

$$\underline{T_{sync} \approx T_3 + \frac{\delta}{2}}$$

Review: Time & Clocks, PB

- Wall clock drift, so they are all skewed
 - Synchronize to bound skew, but still left with uncertainty
 - NTP sync sometimes sufficient
 - Getting less sub-ms sync challenging due to network
- Logical Clock algorithm
 - Guarantees if $a \rightarrow b$, then $C(a) < C(b)$
 - How to generate a total order of events (even if events may happen independently)
- Vector Clock algorithm
 - If $V(a) < V(b)$, then $a \rightarrow b$
 - If $V(a) \not< V(b)$ and $V(b) \not< V(a)$, then $a \parallel b$
 - Can use to infer when an event b was aware of/influenced by a

With multiple replicas, don't need to wait for all...

- Viewstamped Replication:

VR

RSM

- State Machine Replication for any number of replicas
- Replica group: Group of $2f + 1$ replicas
 - Protocol can tolerate f replica crashes

$$2f + 1 = 3$$
$$f = 1$$

simple majority = $f + 1$
= 2


- Assumptions

1. Handles crash failures only: Replicas fail only by completely stopping
2. Unreliable network: Messages might be lost, duplicated, delayed, or delivered out-of-order

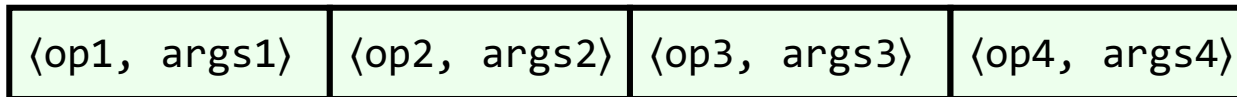
5. 3

Replica state

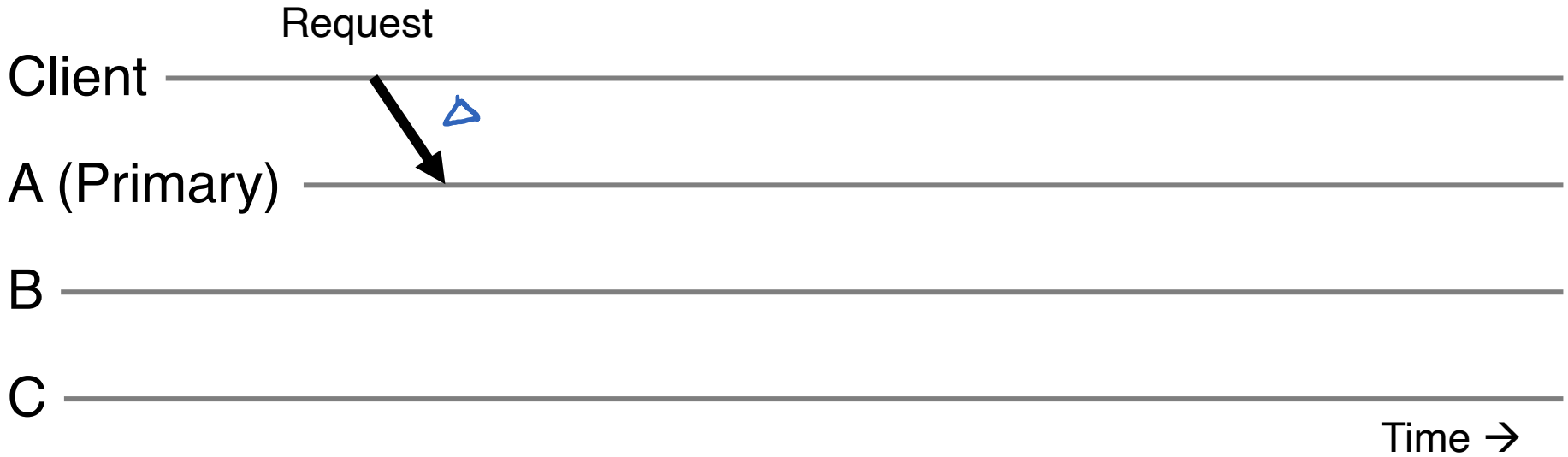
1. Configuration: identities of all $2f+1$ replicas

on-disk 

2. ~~In-memory~~ log with clients' requests in assigned order

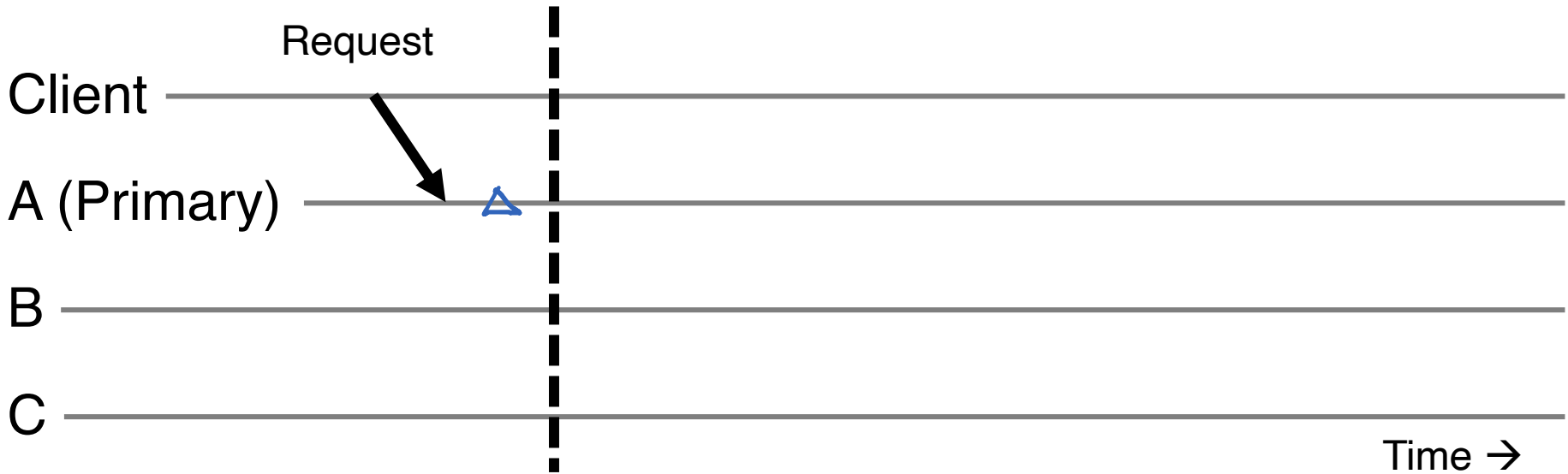


Normal operation *Two-phase Commit (2PC)* ($f=1$)



Normal operation

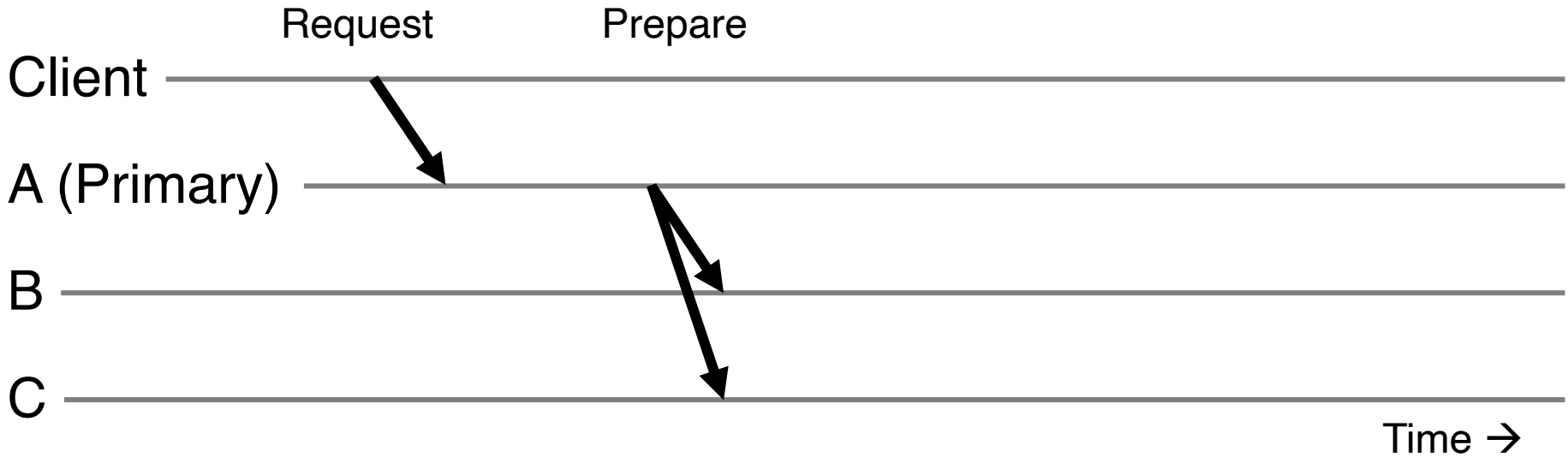
($f = 1$)



1. Primary adds request to end of its log

Normal operation

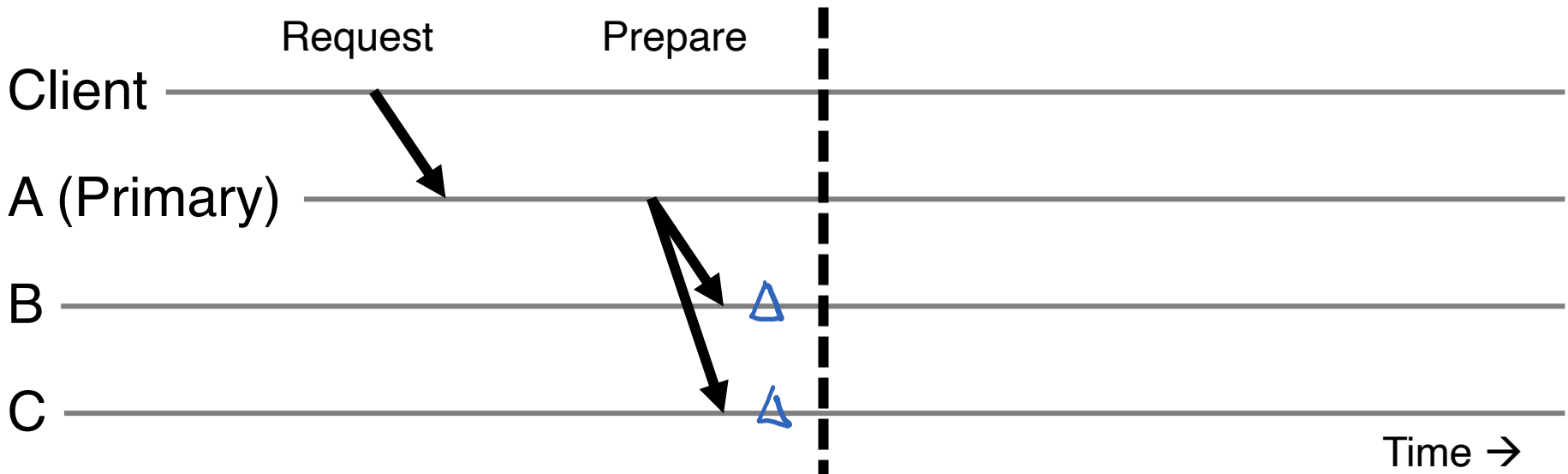
($f = 1$)



1. Primary adds request to end of its log

Normal operation

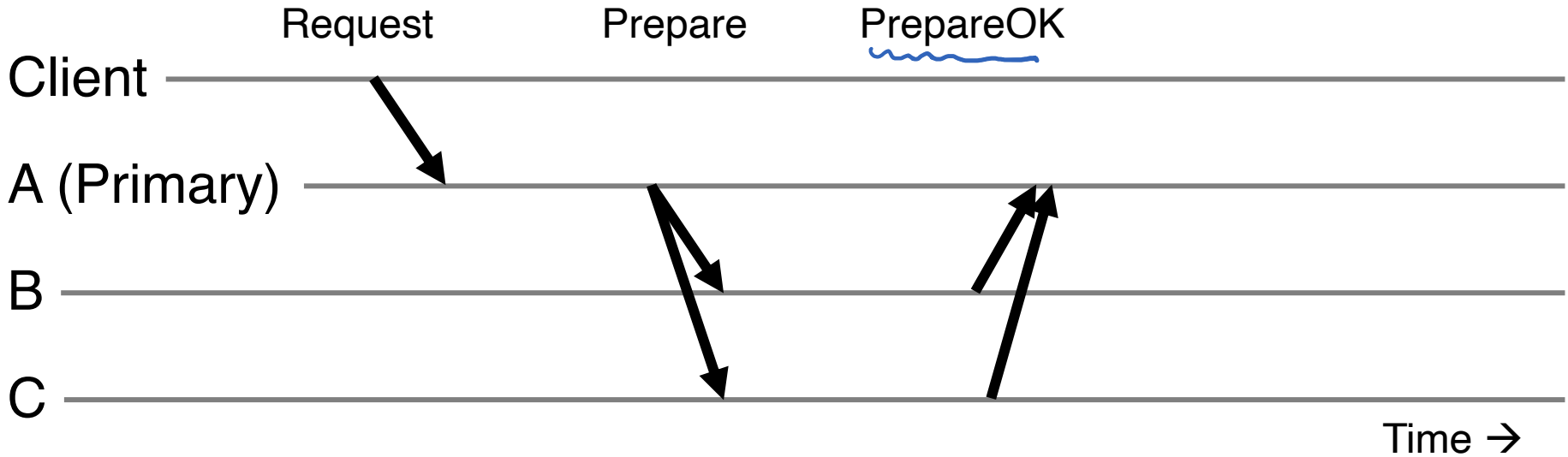
($f = 1$)



1. Primary adds request to end of its log
2. Replicas add requests to their logs in primary's log order

Normal operation

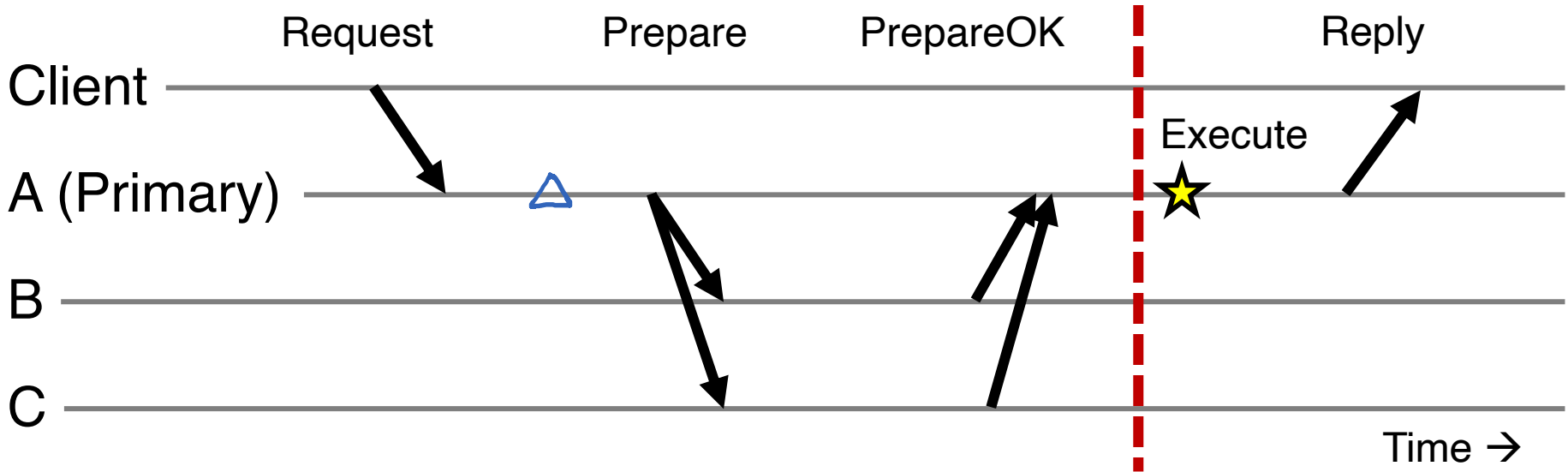
($f = 1$)



1. Primary adds request to end of its log
2. Replicas add requests to their logs in primary's log order

Normal operation

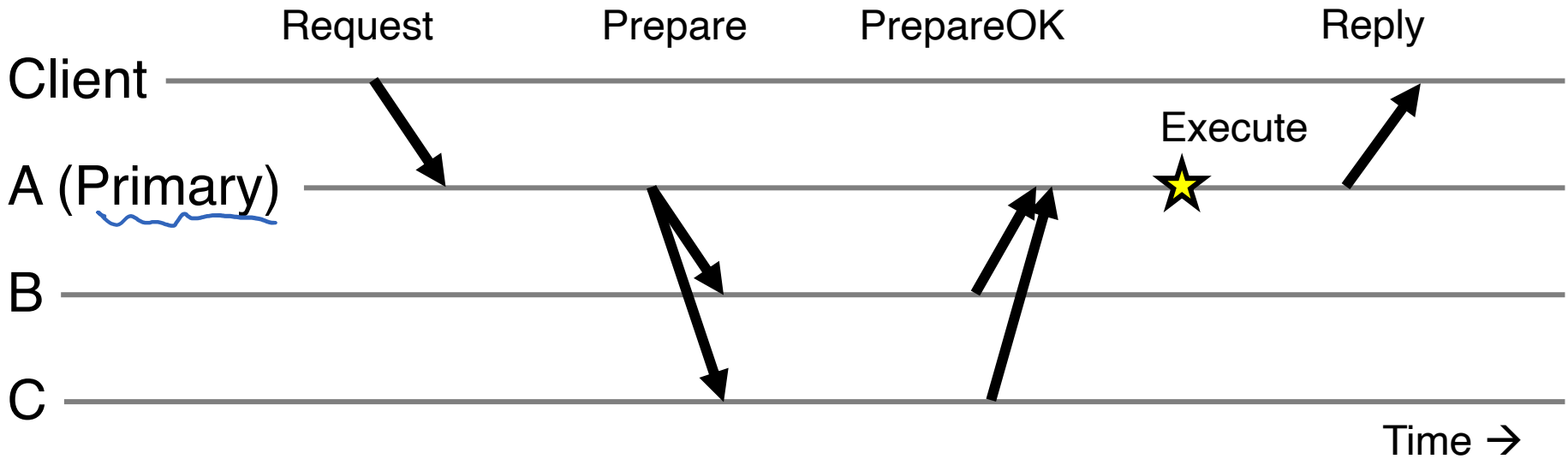
($f = 1$)



1. Primary adds request to end of its log
2. Replicas add requests to their logs in primary's log order
3. Primary **waits for f PrepareOKs** → request is **committed**

Normal operation: Key points

($f = 1$)

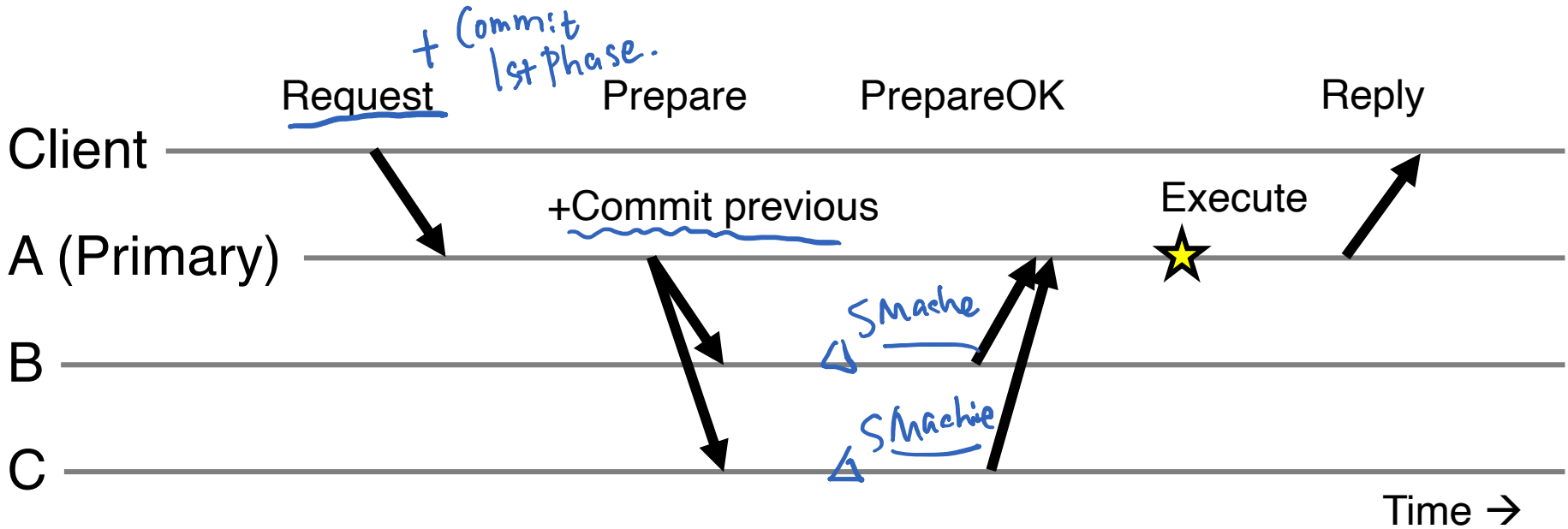


- Protocol provides state machine replication
- On execute, primary knows request in $f + 1 = 2$ nodes' logs
 - Even if $f = 1$ then **crash**, ≥ 1 **retains request in log**

Piggybacked commits

2nd phase

($f = 1$)



- Previous Request's commit **piggybacked** on current Prepare

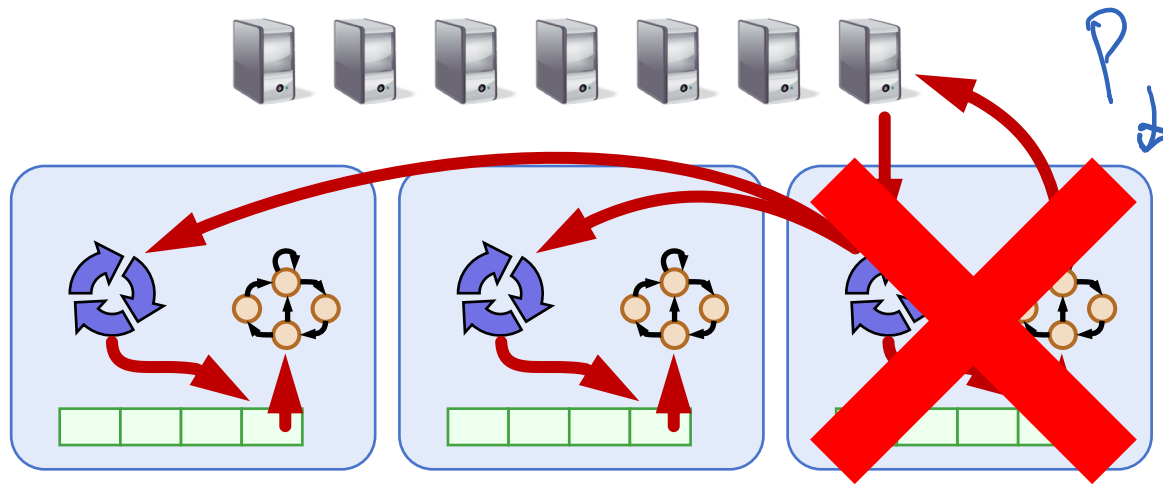
- No client Request after a timeout period?
 - Primary sends Commit message to all backups

The need for a view change

- So far: **Works** for f failed backup replicas

The need for a view change

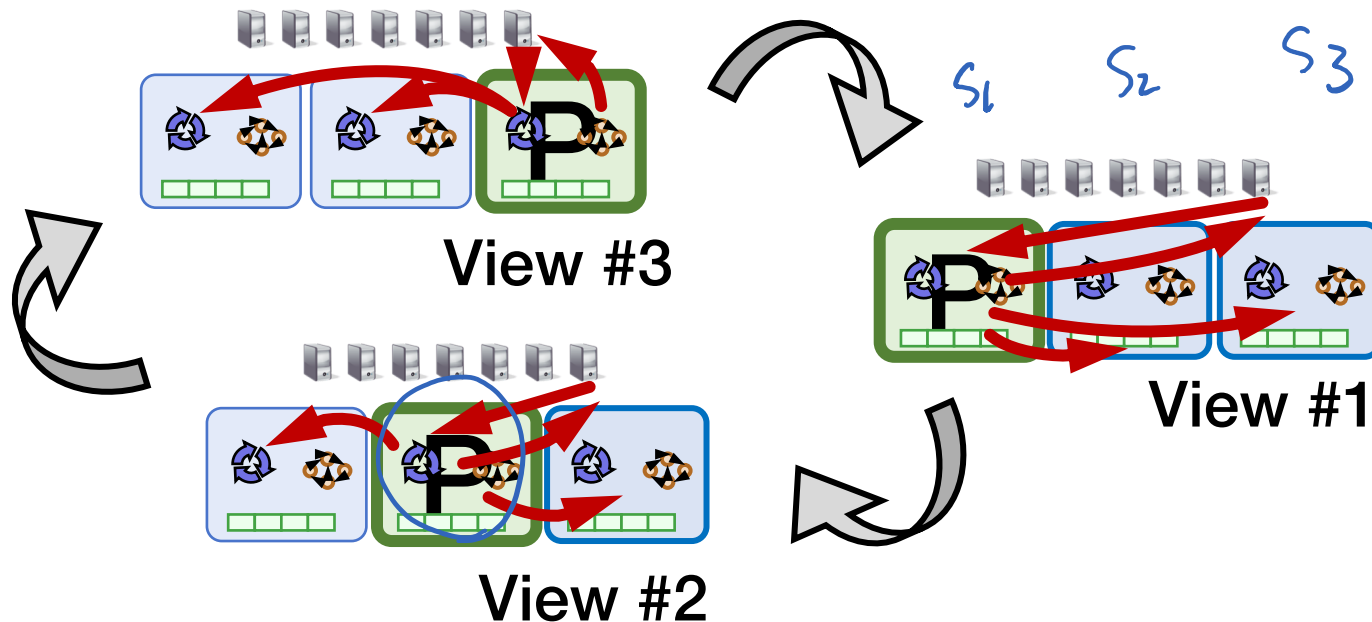
- So far: **Works** for f failed backup replicas
- But what if the f failures include a **failed primary**?
 - All clients' requests go to the failed primary
 - System halts despite **merely f failures**



Views

logical clock

- Let **different replicas** assume role of primary over time
- System moves through a sequence of views
 - **View** = (view number, primary id, backup id, ...)



Correctly changing views

- View changes happen locally at each replica
- Old primary executes requests in the old view, new primary executes requests in the new view
- Want to ensure state machine replication

Correctly changing views

- View changes happen locally at each replica
- Old primary executes requests in the old view, new primary executes requests in the new view
- Want to ensure state machine replication
- So correctness condition: Executed requests
 1. Survive in the new view
 2. Retain the same order in the new view

Correctly changing views

- View changes happen locally at each replica
- Old primary executes requests in the old view,

How do they **agree on the new primary**?

What if both backup nodes attempt to become the new primary simultaneously?

2. Retain the same order in the new view

Today's outline

1. View changes in primary-backup replication

2. Consensus

- Paxos
- Raft

Consensus

- Definition:
 1. A **general agreement** about something
 2. An idea or opinion that is **shared by all** the people in a group

Consensus used in systems

Group of servers attempting:

Consensus used in systems

Group of servers attempting:

- Make sure all servers in group receive the same updates in the same order as each other

Consensus used in systems

Group of servers attempting:

- Make sure all servers in group receive the same updates in the same order as each other
- Maintain own lists (views) on who is a current member of the group, and update lists when somebody leaves/fails

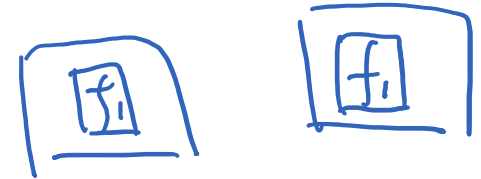
Consensus used in systems

Group of servers attempting:

- Make sure all servers in group receive the same updates in the same order as each other
- Maintain own lists (views) on who is a current member of the group, and update lists when somebody leaves/fails
- Elect a leader in group, and inform everybody

Consensus used in systems

Group of servers attempting:



- Make sure all servers in group receive the same updates in the same order as each other
- Maintain own lists (views) on who is a current member of the group, and update lists when somebody leaves/fails
- Elect a leader in group, and inform everybody
- Ensure mutually exclusive (one process at a time only) access to a critical resource like a file

Consensus



Given a set of processors, each with an initial value:

- **Termination:** All non-faulty processes eventually decide on a value
- **Agreement:** All processes that decide do so on the same value
- **Validity:** Value decided must have proposed by some process

Paxos

- Safety (bad things never happen)
 - Only a single value is chosen — agreement
 - Only chosen values are learned by processes
 - Only a proposed value can be chosen — validity
- Liveness (good things eventually happen)




Paxos

- Safety (bad things never happen)
 - Only a single value is chosen  **agreement**
 - Only chosen values are learned by processes
 - Only a proposed value can be chosen  **validity**
- Liveness (good things eventually happen)

Paxos

- Safety (bad things never happen)
 - Only a single value is chosen ← agreement
 - Only chosen values are learned by processes
 - Only a proposed value can be chosen ← validity
- Liveness (good things eventually happen)
 - Some proposed value eventually chosen if fewer than half of processes fail ↗ termination
 - If value is chosen, a process eventually learns it

Paxos

- Safety (bad things never happen)
 - Only a single value is chosen  **agreement**
 - Only chosen values are learned by processes
 - Only a proposed value can be chosen  **validity**
- Liveness (good things eventually happen)
 - Some proposed value eventually chosen if fewer than half of processes fail  **termination**
 - If value is chosen, a process eventually learns it