Distributed Consensus

CS 675: Distributed Systems (Spring 2020)
Lecture 5

Yue Cheng

Today's outline

1. View changes in primary-backup replication

2. Consensus

    - Paxos

    - Raft

Review: Time & Clocks, PB

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

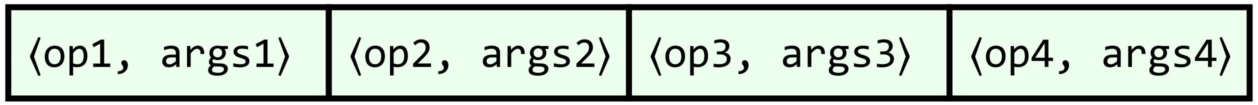
- Viewstamped Replication:
  - State Machine Replication for any number of replicas
  - Replica group: Group of 2f + 1 replicas
    - Protocol can tolerate f replica crashes

- 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

Replica state

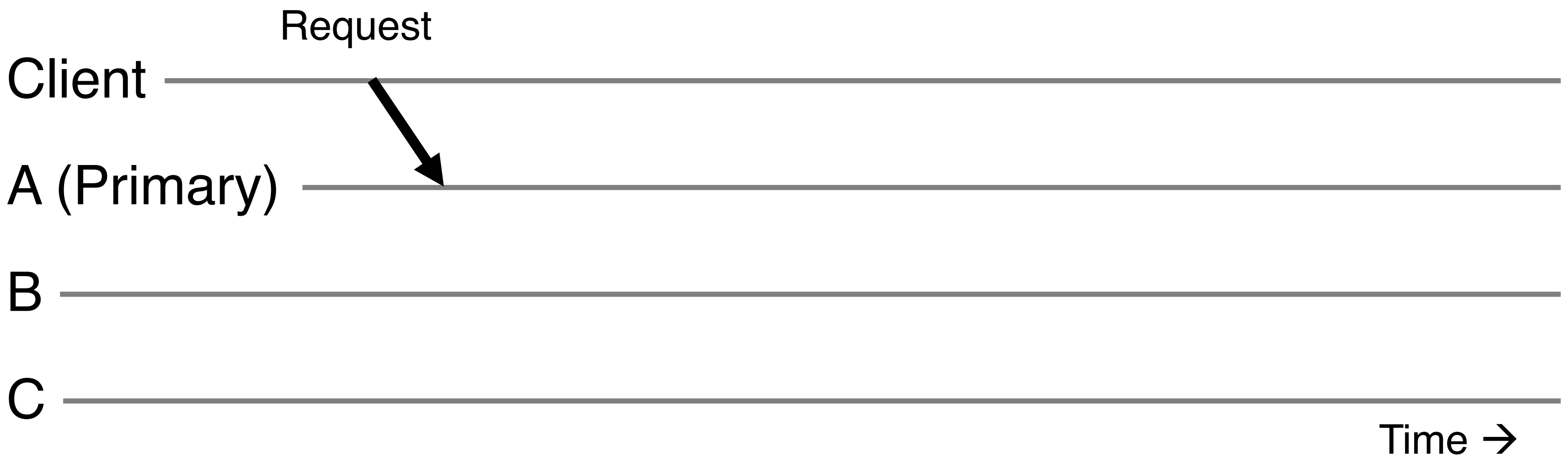
1. Configuration: identities of all 2f+1 replicas

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



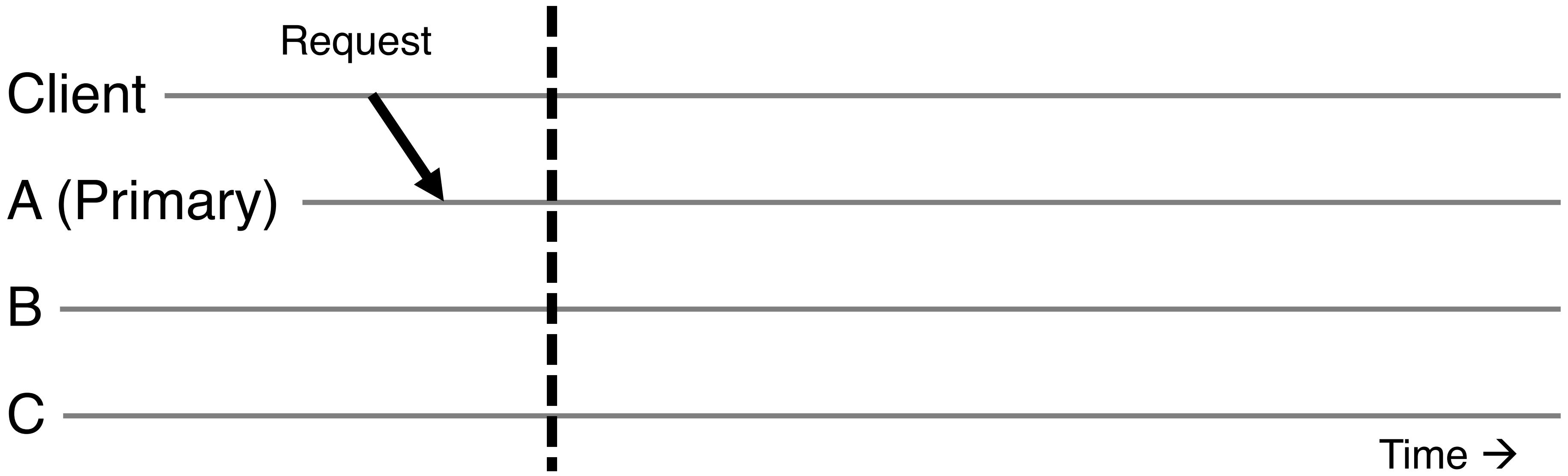
Normal operation

(*f* = 1)



Normal operation

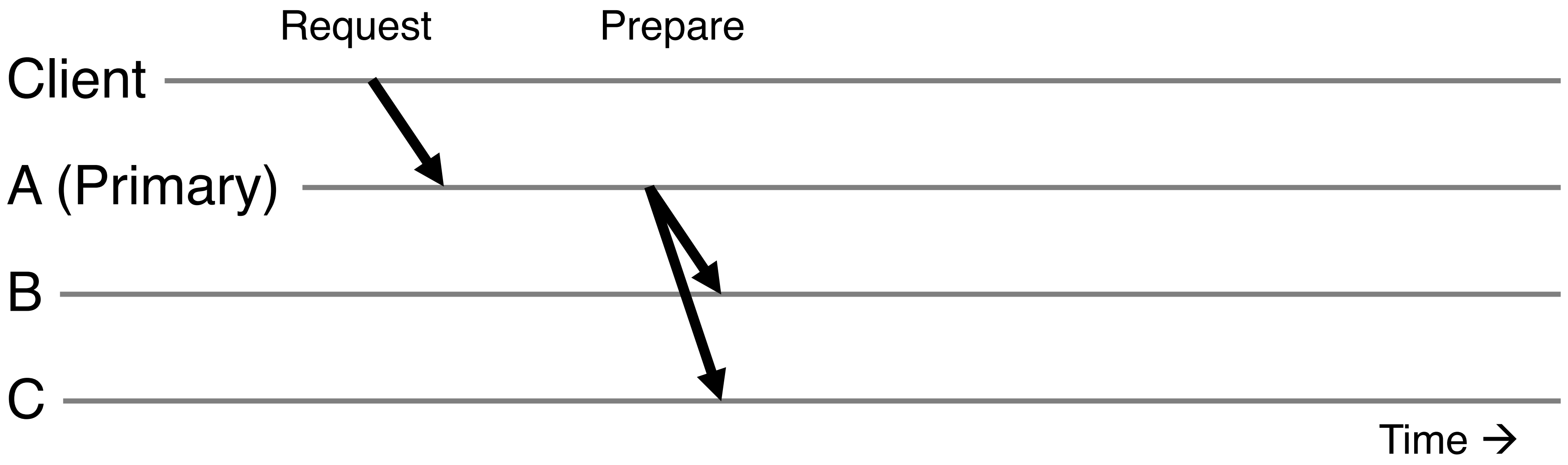
(f = 1)



1. Primary adds request to end of its log

Normal operation

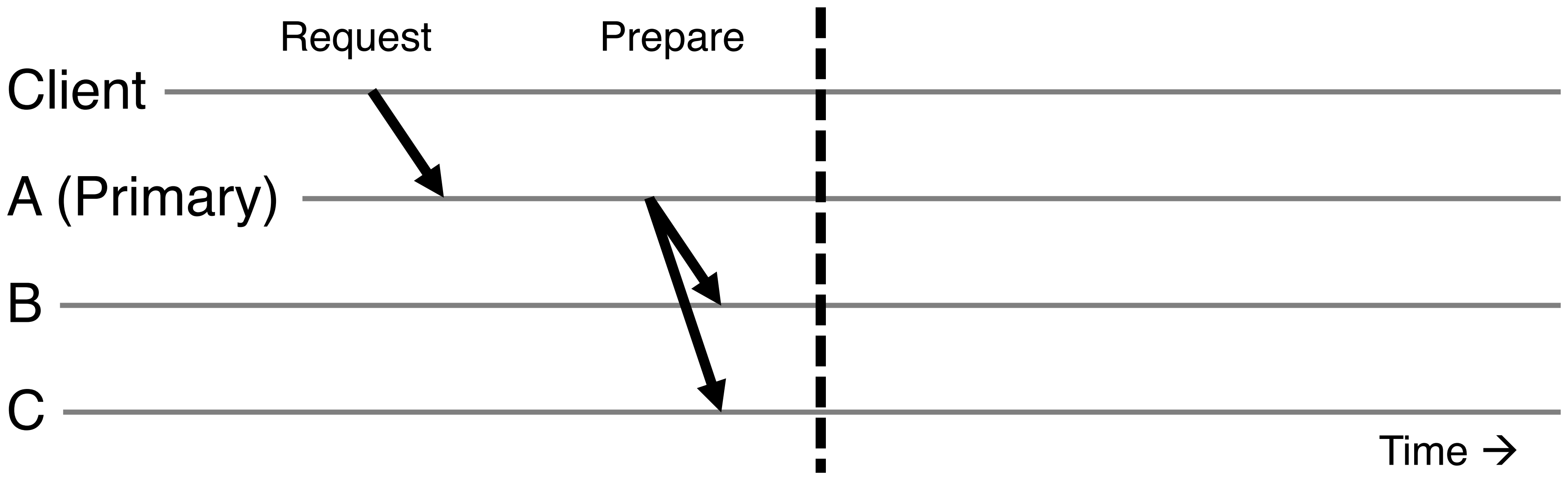
(*f* = 1)



1. Primary adds request to end of its log

Normal operation

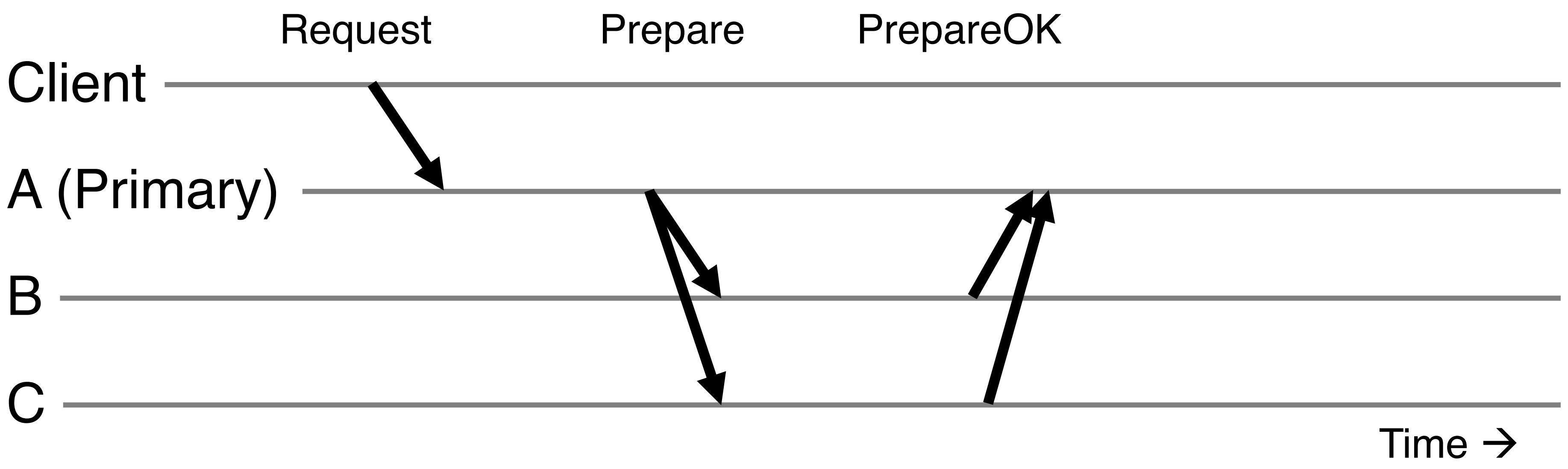
(*f* = 1)



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

Normal operation

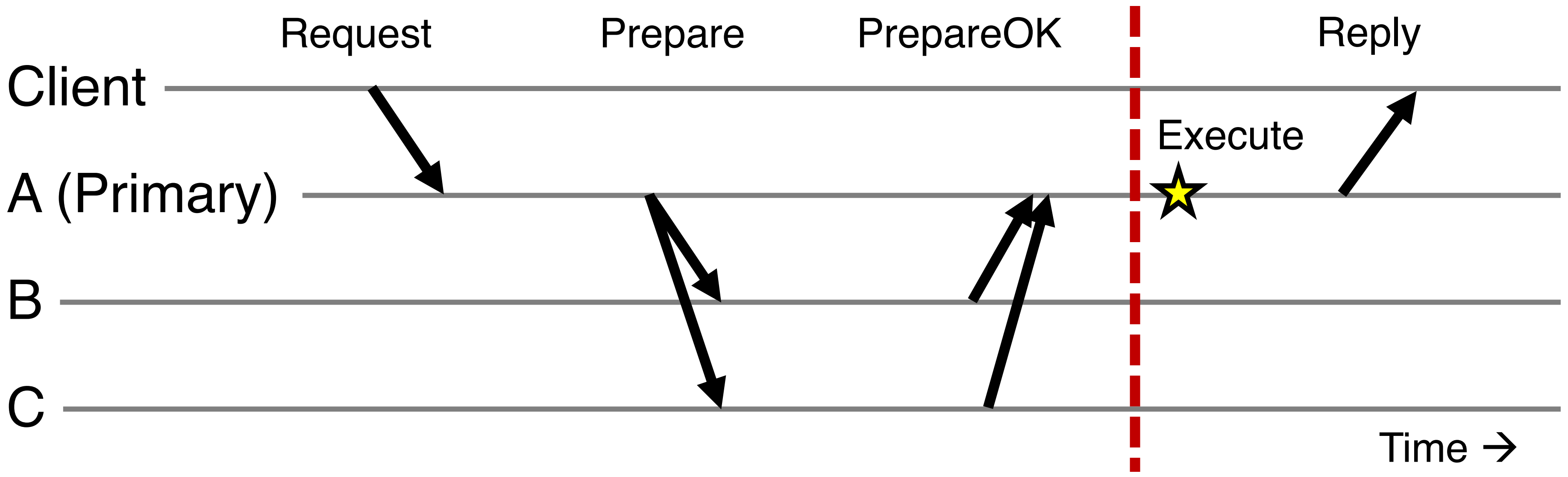
(*f* = 1)



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

Normal operation

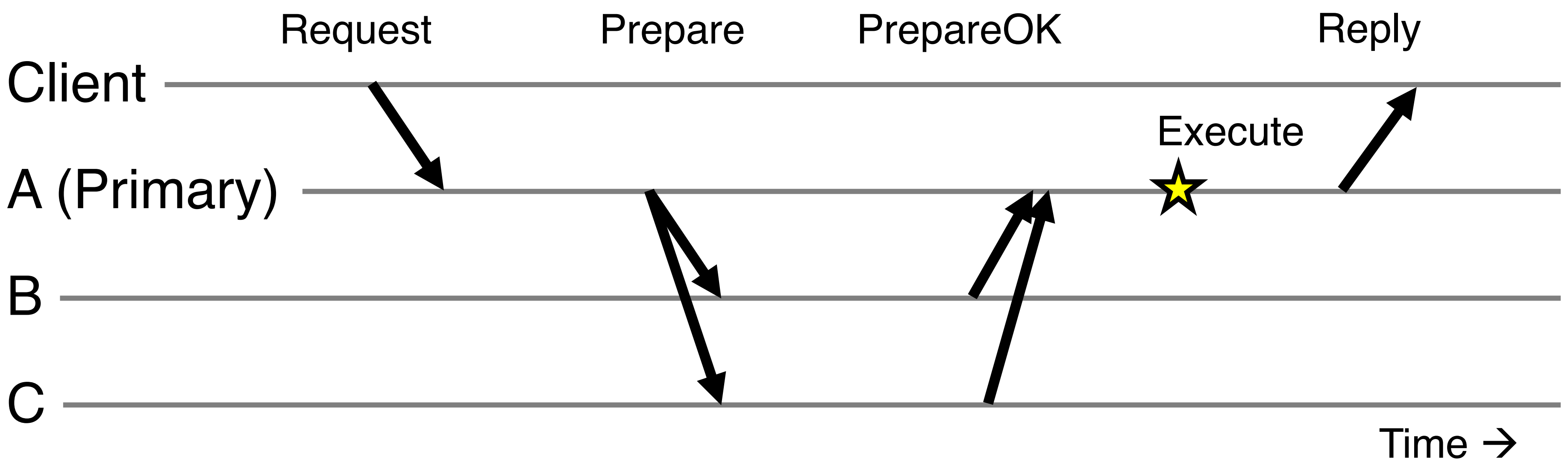
(*f* = 1)



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

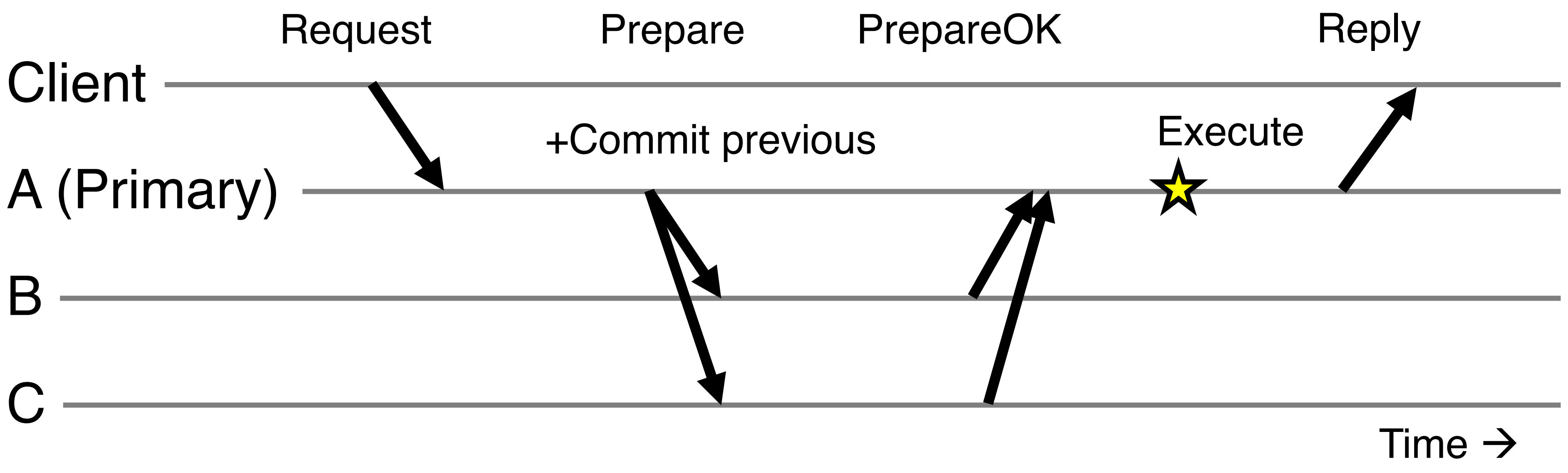
($f = 1$)



- 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

(*f* = 1)



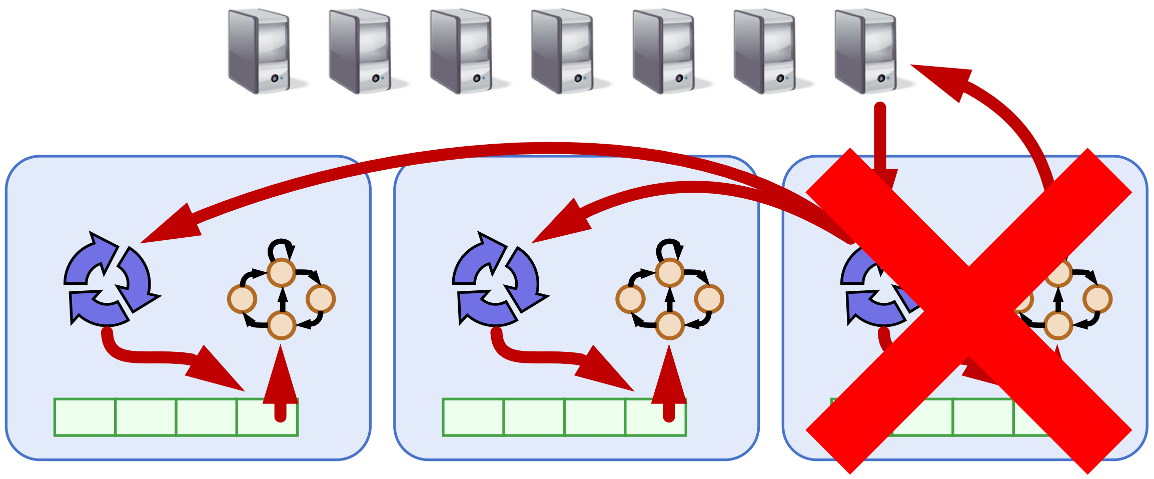
- Previous Request's commit piggybacked on current Prepare
- No client Request after a timeout period?
  - Primary sends Commit message to all backups

The need for a view change

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

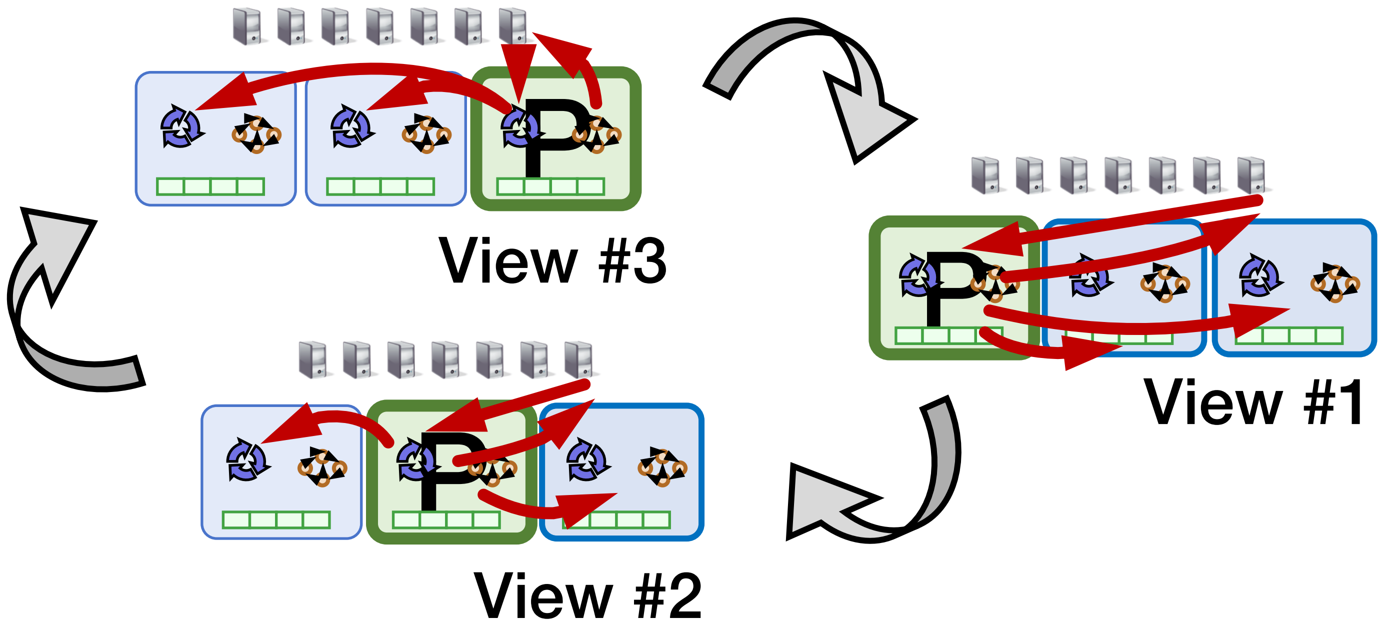
The need for a view change

- 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

- 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

Safety vs. Liveness properties

- Safety (bad things never happen)

- Liveness (good things eventually happen)

Paxos

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

Paxos

- Safety (bad things never happen)
  - Only a single value is chosen $\leftarrow$ agreement
  - Only chosen values are learned by processes
  - Only a proposed value can be chosen $\leftarrow$ 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
  - 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