View Change Protocols and Reconfiguration

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS 240: Computing Systems and Concurrency
Lecture 12

Marco Canini

Today

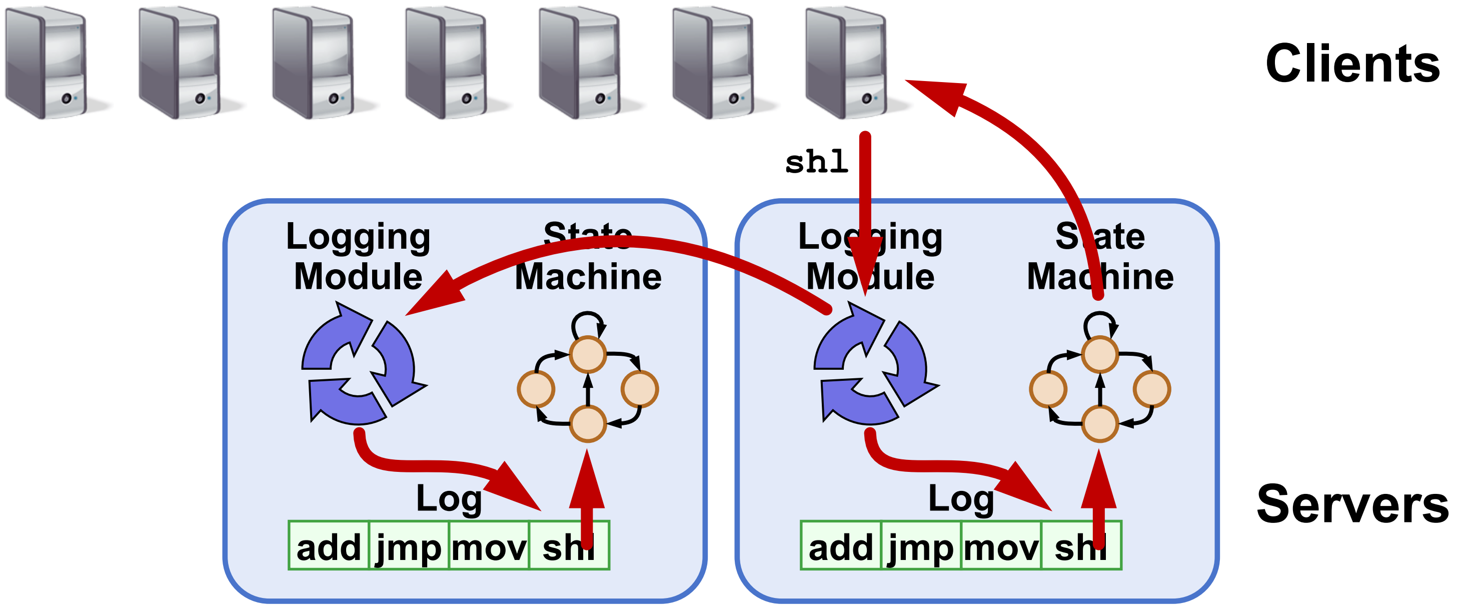
1. **More primary-backup replication**

2. View changes

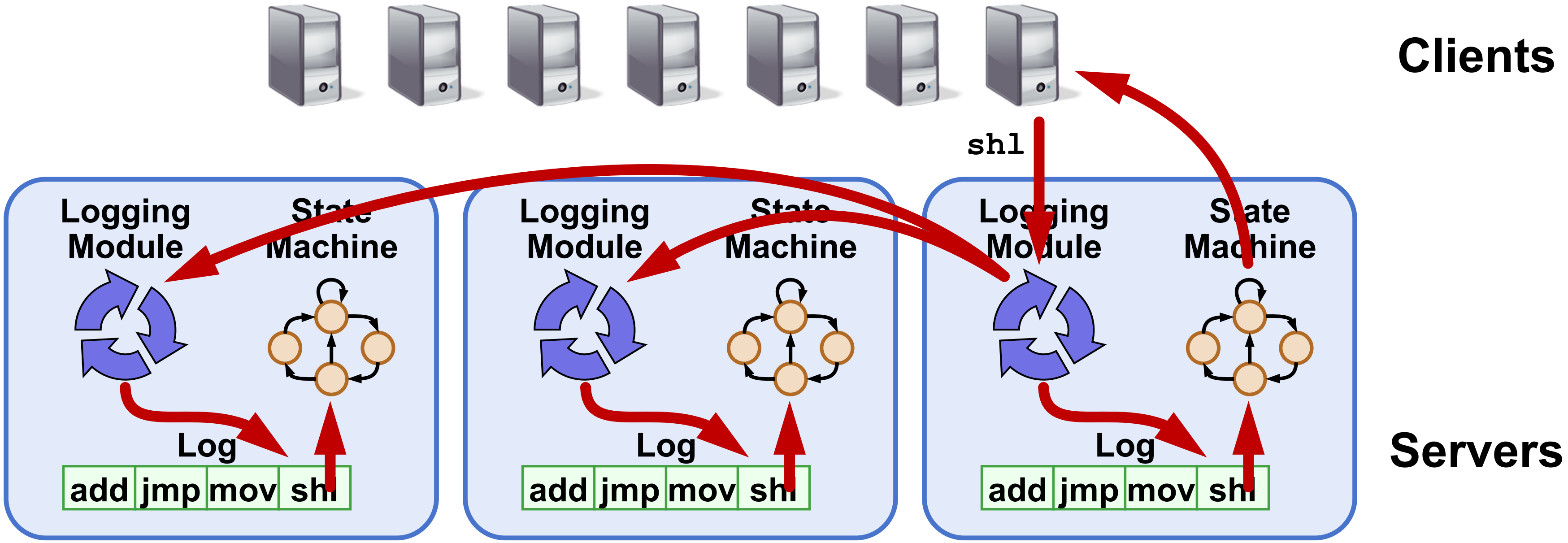
3. Reconfiguration

Review: primary-backup replication

- Nominate one replica ***primary***
  - Clients send all requests to **primary**
  - Primary **orders** clients' requests



From two to many replicas



- Primary-Backup with many replicas
  - Primary waits for acknowledgement from **all** backups
  - All updates to set of replicas needs to update shared disk (recall VM-FT)

What else can we do with more replicas?

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

- Differences with primary-backup
  - No shared disk (no reliable failure detection)
  - Don’t need to wait for **all** replicas to reply
  - Need more replicas to handle *f* failures ($2f$+1 vs f+1)

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

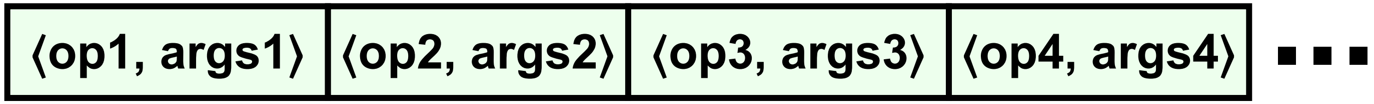
- **Viewstamped Replication:**
  - **State Machine Replication** for any number of replicas
  - ***Replica group:*** Group of **2*f* + 1** replicas
    - Protocol can tolerate ***f* replica** crashes

- 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

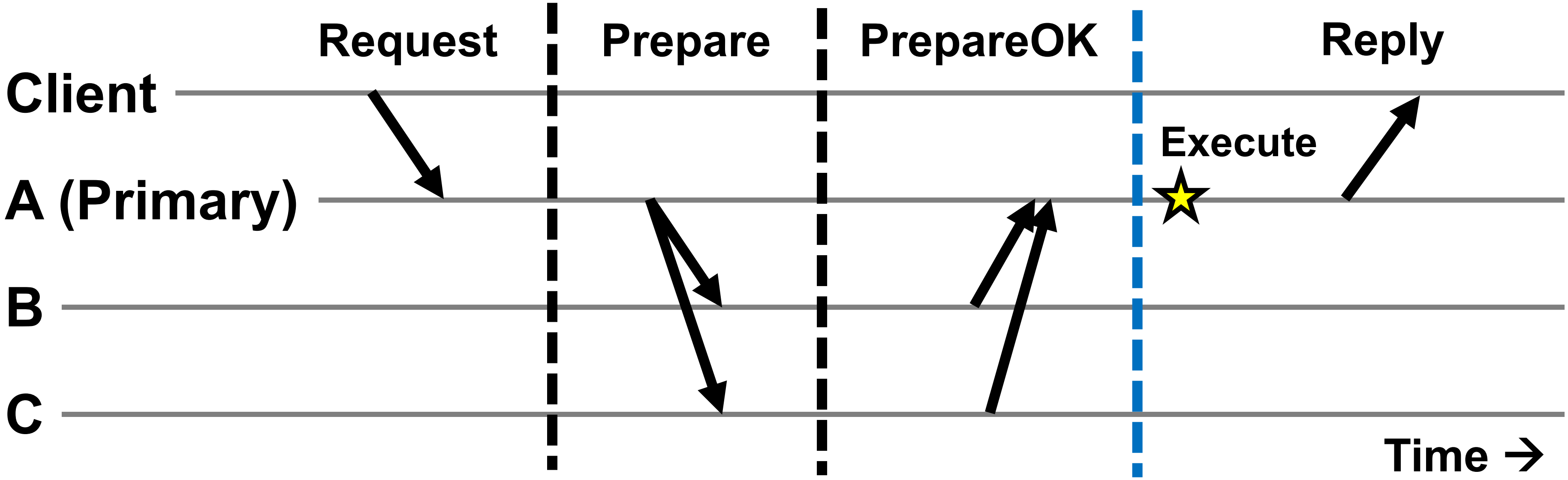
2. In-memory ***log*** with clients' requests in assigned order



⟨op1, args1⟩ ⟨op2, args2⟩ ⟨op3, args3⟩ ⟨op4, args4⟩ ■ ■ ■

Normal operation

(*f* = 1)

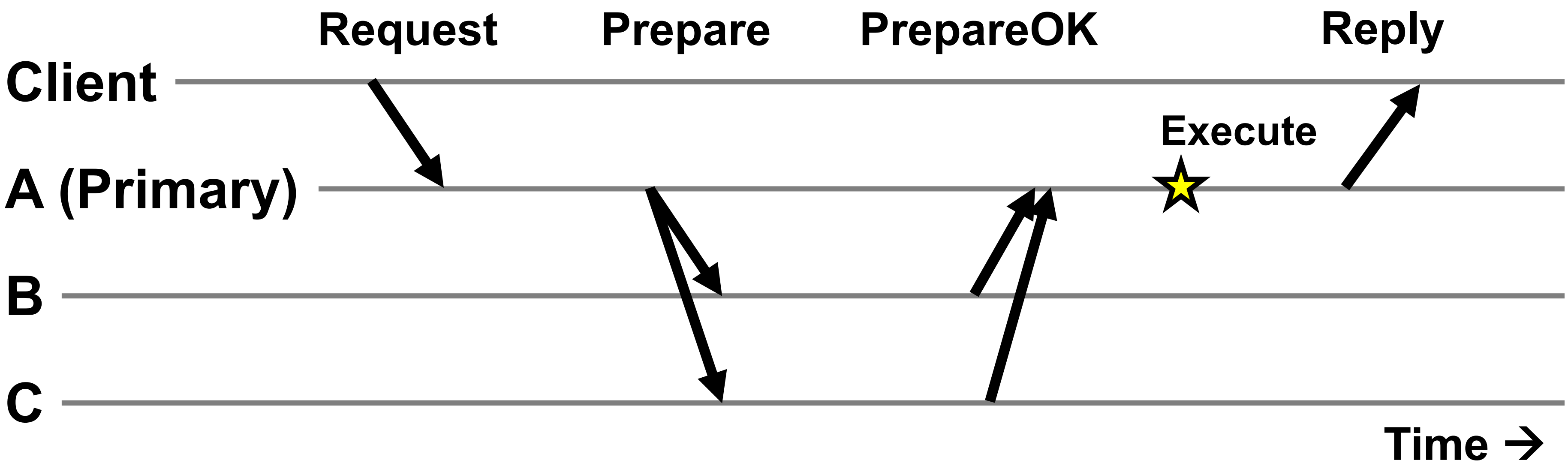


1. Primary adds request to end of its log

2. Replicas add requests to their logs in primary's log order

3. Primary **waits for *f*** PrepareOKs → request is ***committed***
   – Makes up-call to execute the operation ★

Normal operation: Key points (*f* = 1)

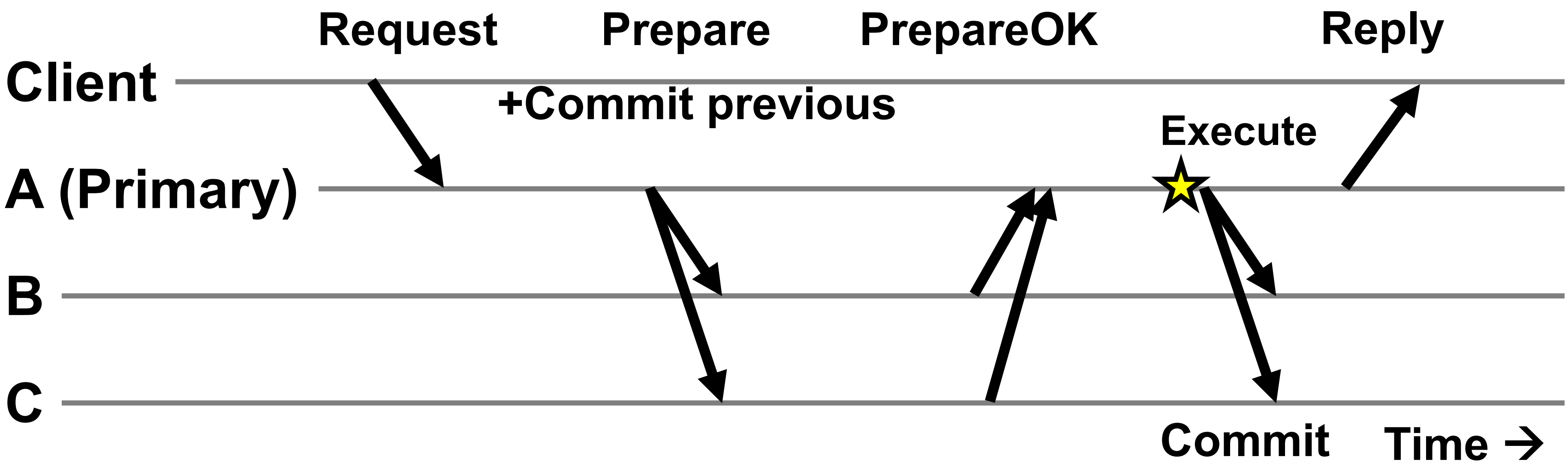


- Protocol guarantees **state machine replication**

- 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)



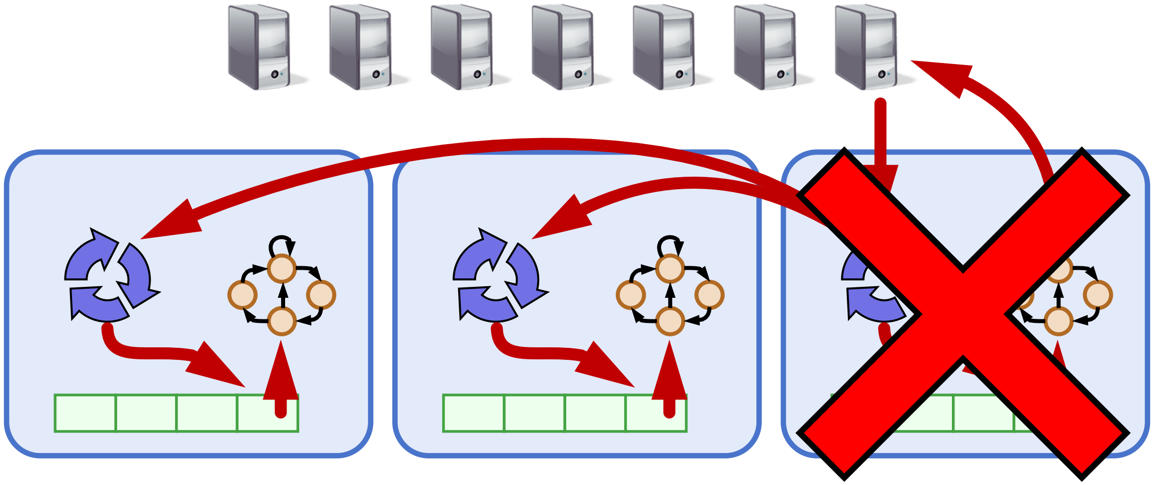
- Previous Request's commit **piggybacked** on current **Prepare**

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



Today

1. More primary-backup replication

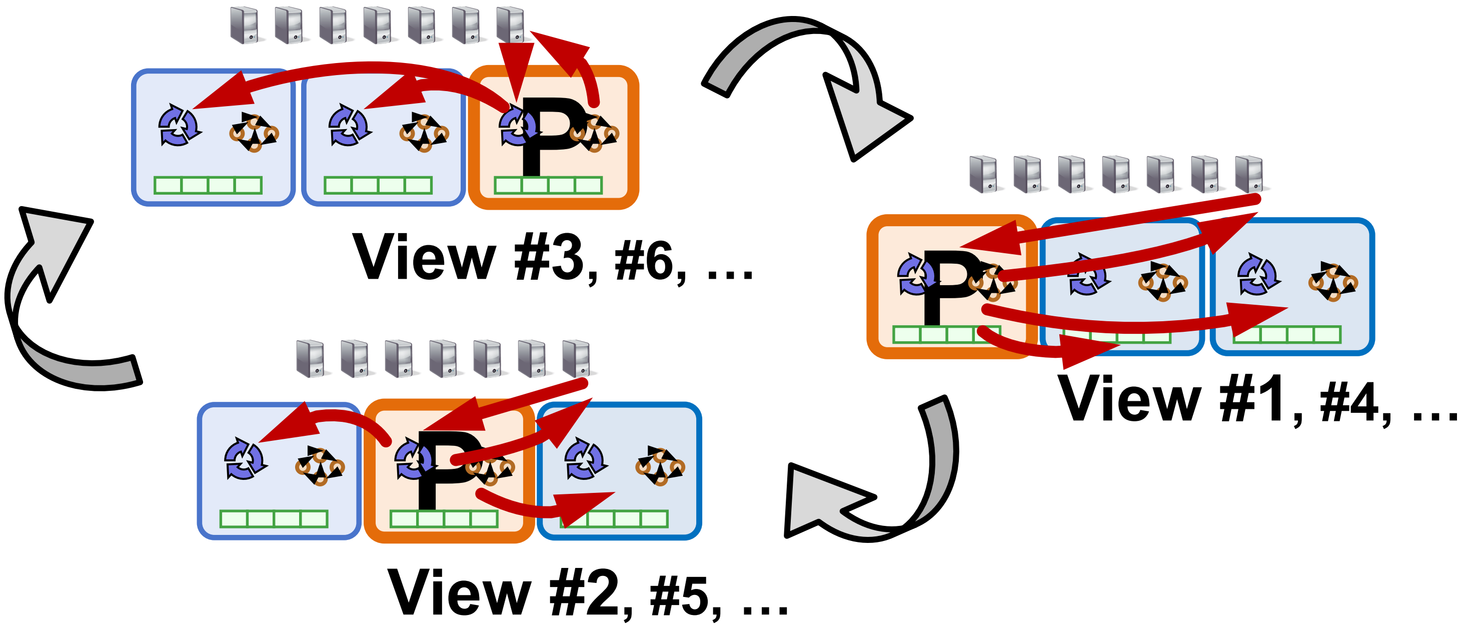
2. **View changes**
   - **With Viewstamped Replication**
   - Using a View Server

3. Reconfiguration

Views

- Let **different replicas** assume role of primary **over time**

- System moves through a sequence of **views**
  - ***View*** **=** (view number, primary id, backup id, ...)



View change protocol

- Backup replicas **monitor** primary

- If primary seems **faulty** (no Prepare/Commit):
  - Backups execute the ***view change protocol*** to select new primary
    - View changes execute **automatically**, **rapidly**

- Need to keep clients and replicas in sync: same **local** state of **the current view**
  - Same local state at **clients**
  - Same local state at **replicas**

Making the view change correct

- 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: Committed requests**
  1. **Survive** in the new view
  2. Retain the **same order** in the new view

Replica state (for view change)

1. ***configuration:*** **sorted** identities of all 2*f* + 1 replicas

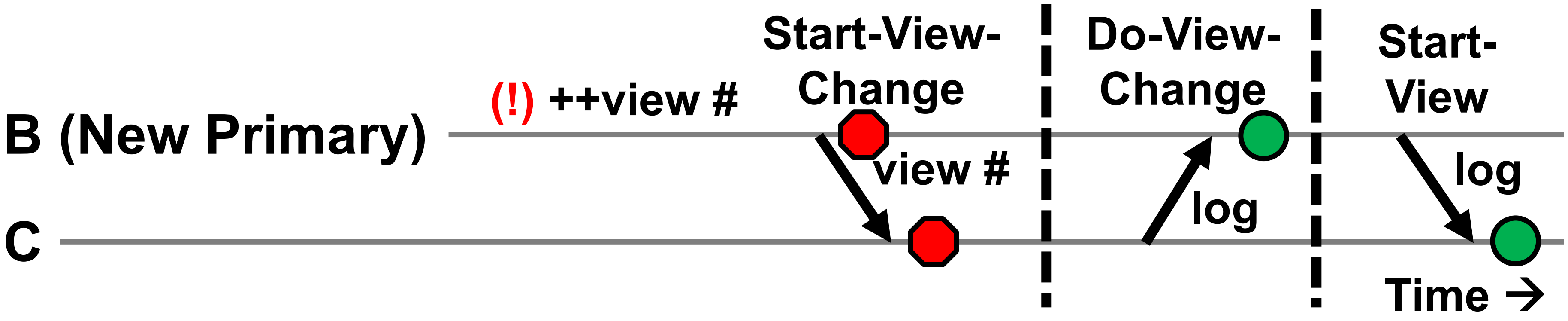
2. In-memory ***log*** with clients' requests in assigned order

3. ***view-number:*** identifies primary in configuration list

4. ***status:*** **normal** or in a **view-change**

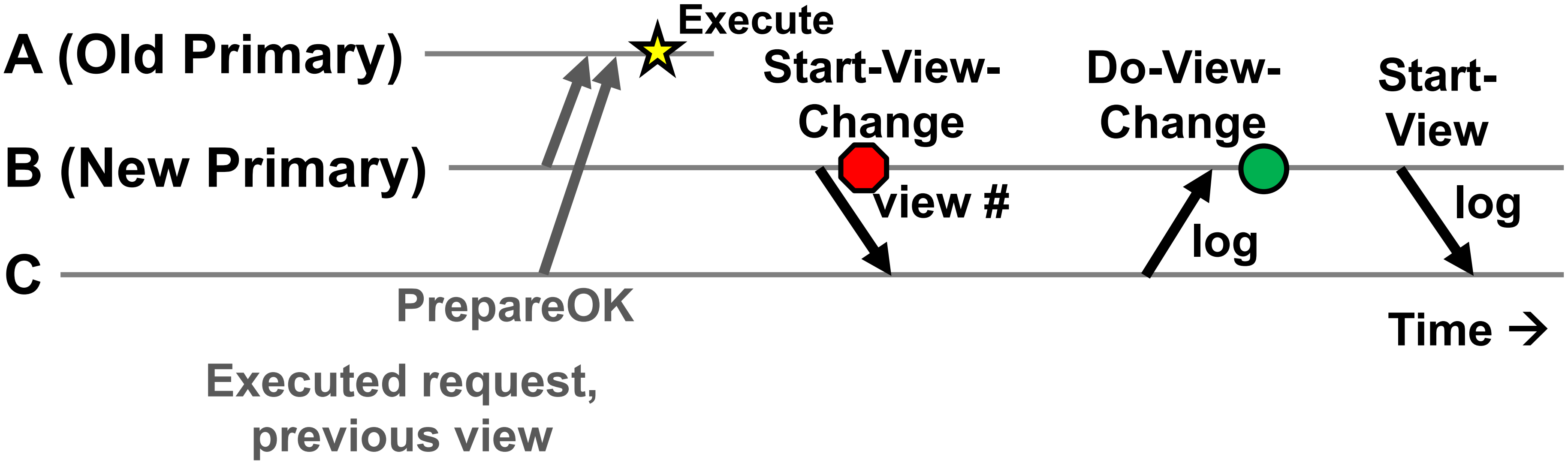
View change protocol

(*f* = 1)



1. B notices A has failed, sends **Start-View-Change**
2. C replies **Do-View-Change** to new primary, with its log
3. B waits for *f* replies, then sends **Start-View**
4. On receipt of Start-View, C replays log, accepts new ops

View change protocol: Correctness (*f* = 1)

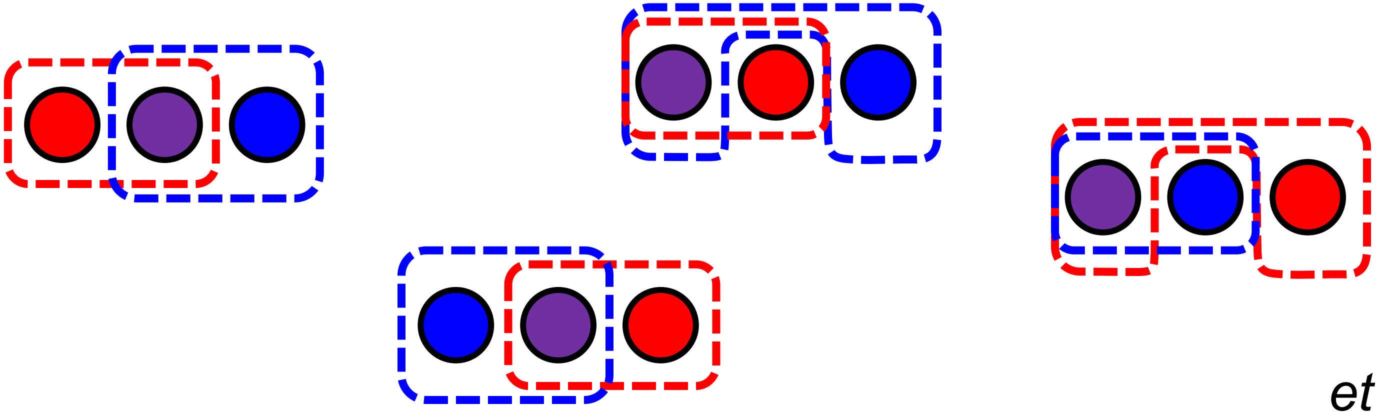


- Old primary **A** must have received one or two **PrepareOK** replies for that request (*why?*)

- Request is in B's or C's **log (or both):** so it **will survive** into new view

Principle: Quorums

(*f* = 1)



et cetera...

- Any **group of *f* + 1 replicas** is called a ***quorum***

- **Quorum intersection property:** Two quorums in 2*f* + 1 replicas must **intersect** at **at least one replica**

Applying the quorum principle

Normal Operation:

- Quorum that processes one request: **Q1**
  - ...and 2nd request: **Q2**

- **Q1** ∩ **Q2** has at least **one replica** →
  - Second request **reads first request's effects**

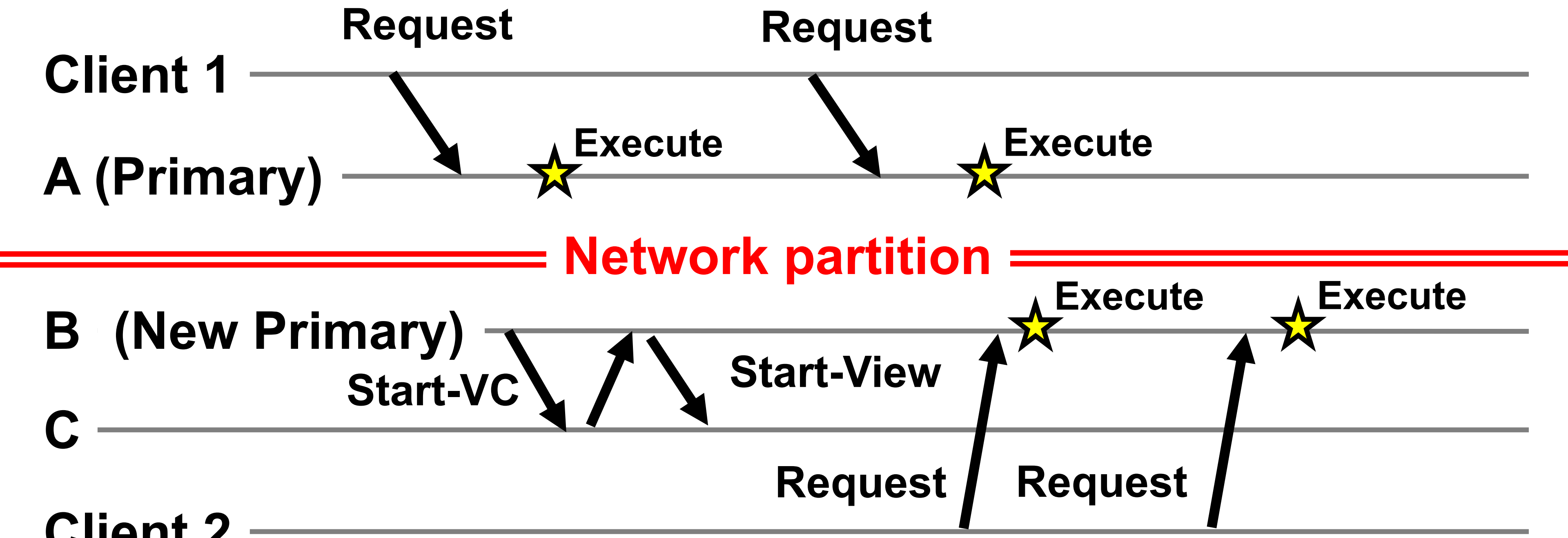
Applying the quorum principle

View Change:

- Quorum processes previous (committed) request: **Q1**
  - ...and that processes **Start-View-Change: Q2**
- **Q1 ∩ Q2** has at least **one replica →**
  - View Change **contains committed request**

Split Brain

(not all protocol messages shown)



- What's **undesirable** about this sequence of events?
- Why won't this ever happen? What **happens instead?**

Today

1. More primary-backup replication

2. **View changes**
   - With Viewstamped Replication
   - **Using a View Server**

3. Reconfiguration

Would centralization simplify design?

- A single ***View Server*** could **decide who** is primary
  - Clients and servers depend on view server
    - Don’t decide on their own (might not agree)

- Goal in designing the VS:
  - Only **one primary** at a time for correct **state machine replication**

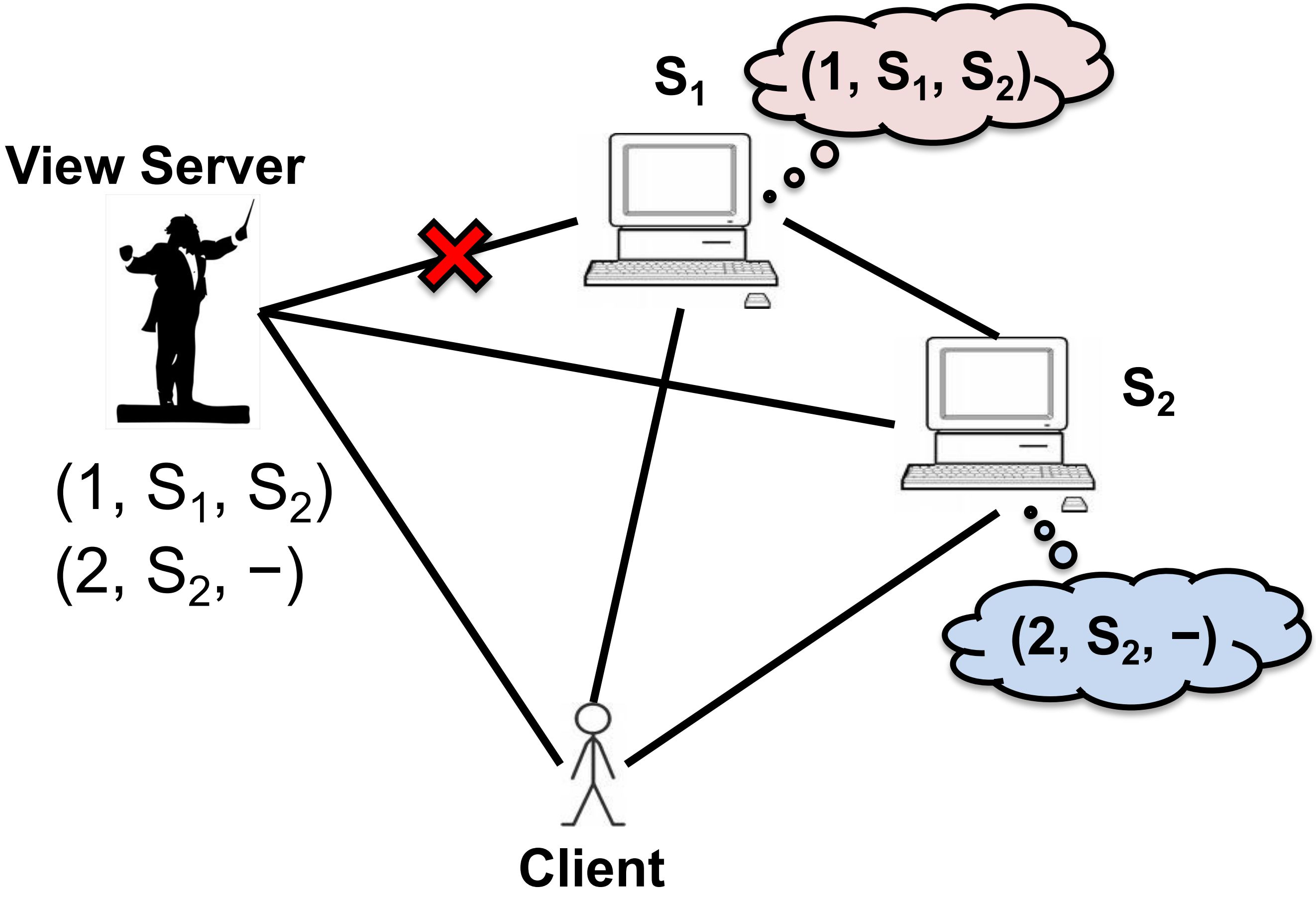
View Server protocol operation

- For now, **assume** VS **never fails**

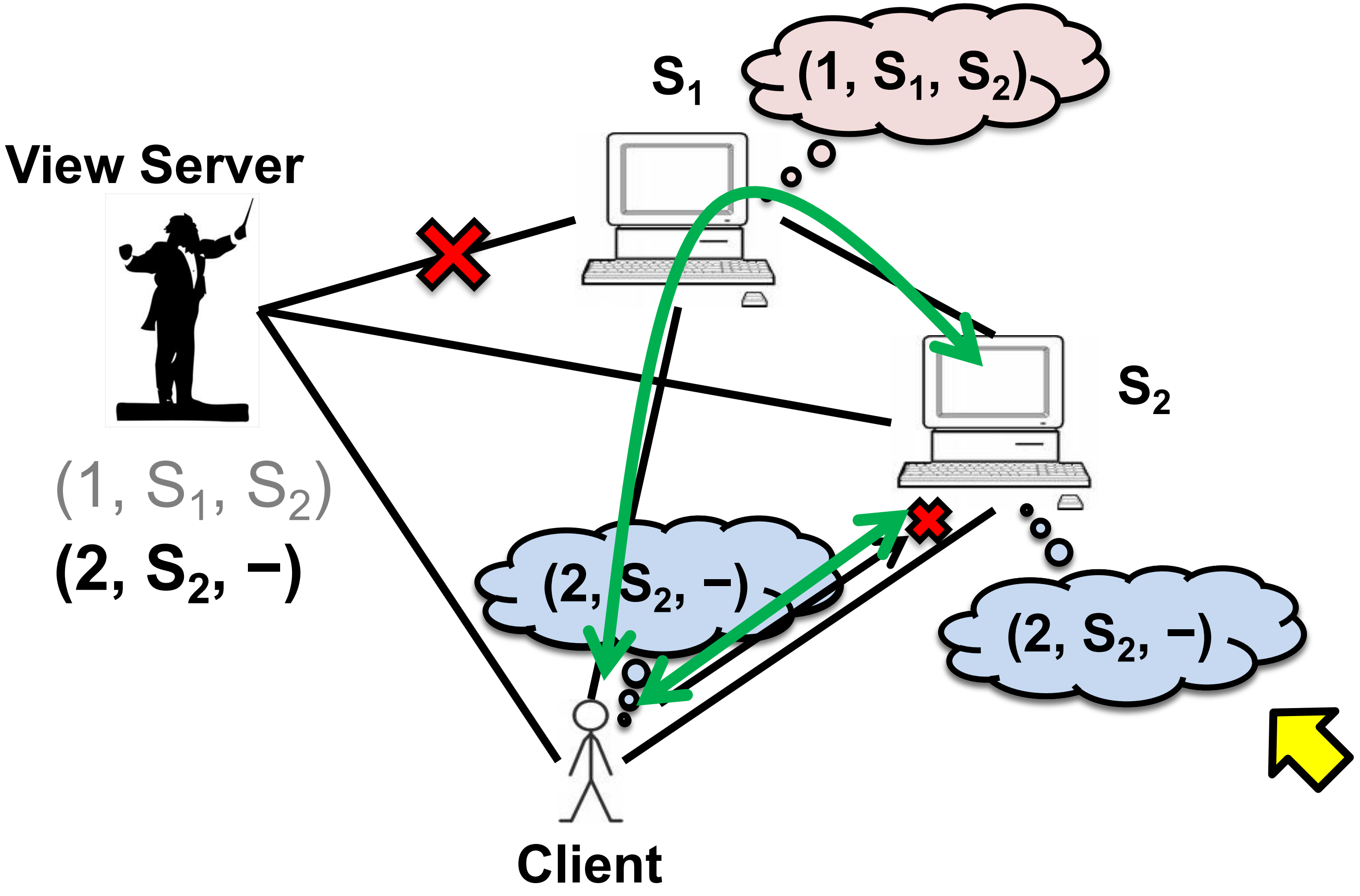
- Each replica now periodically ***pings*** the VS
  - VS declares replica ***dead*** if missed *N* pings in a row
  - Considers replica ***alive*** after a single ping received

- **Problem:** Replica can **be alive but because of network connectivity, be declared "dead"**

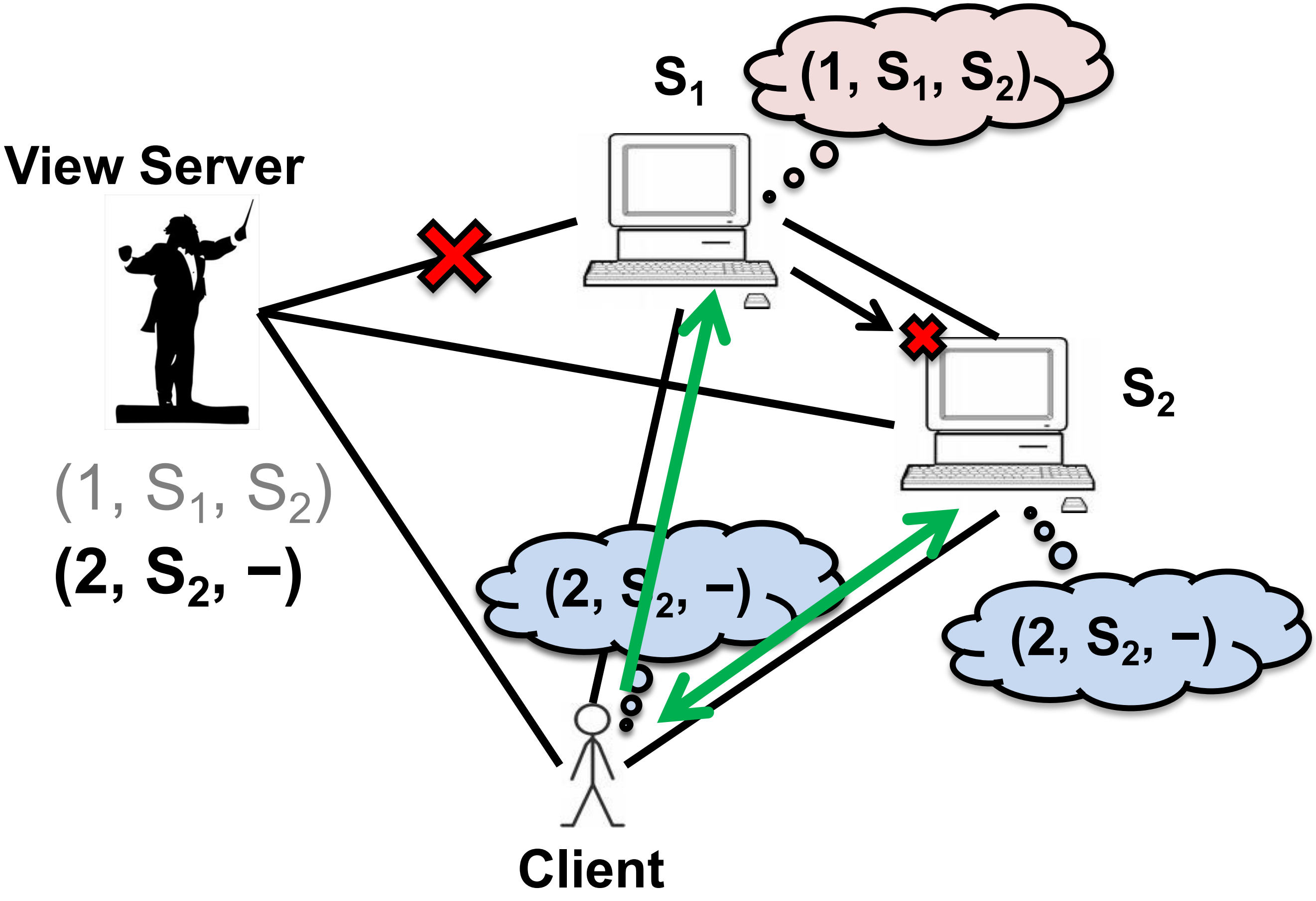
View Server: Split Brain



One possibility: S_2 in old view



Also possible: S_2 in new view



Split Brain and view changes

Take-away points:

- Split Brain problem **can be avoided** both:
  - In a **decentralized** design (VR)
  - With **centralized** control (VS)

- But protocol must be **designed carefully** so that replica state does not **diverge**

Today

1. More primary-backup replication

2. View changes

3. **Reconfiguration**

The need for reconfiguration

- What if we want to **replace a faulty replica** with a different machine?
  - For example, one of the **backups may fail**

- What if we want to **change the replica group size?**
  - **Decommission** a replica
  - **Add** another replica (increase ***f***, possibly)

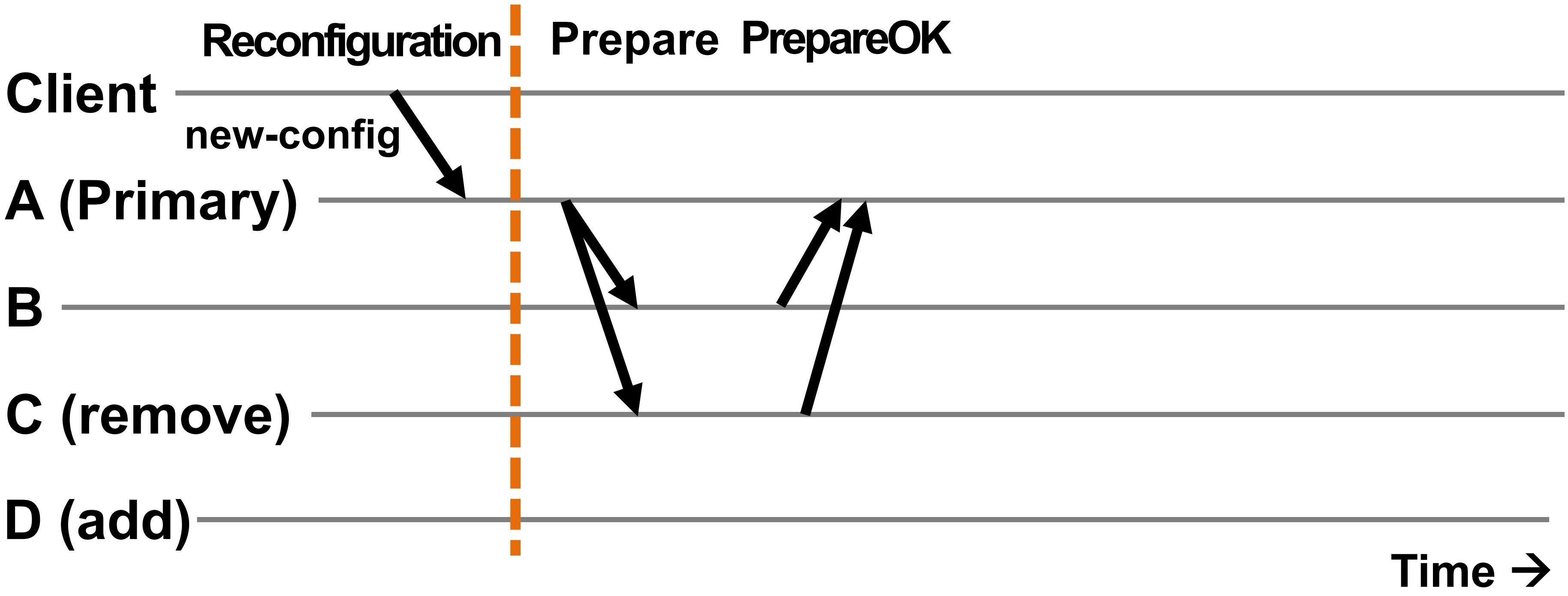
- Protocol that handles these possibilities is called the ***reconfiguration protocol***

Replica state (for reconfiguration)

1. ***configuration:*** **sorted** identities of all 2*f* + 1 replicas
2. In-memory ***log*** with clients' requests in assigned order
3. ***view-number:*** identifies primary in configuration list
4. ***status:*** **normal** or in a **view-change**
5. ***epoch-number:*** indexes configurations

Reconfiguration (1)

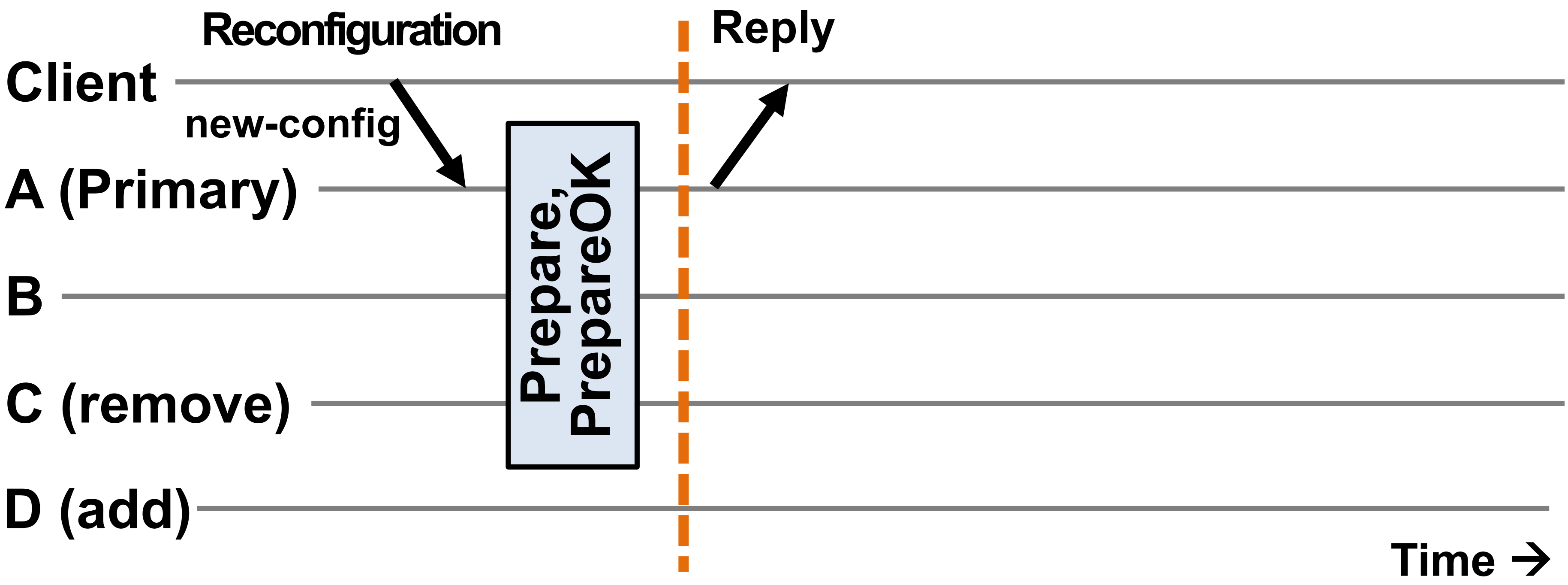
(*f* = 1)



- Primary immediately **stops** accepting new requests

Reconfiguration (2)

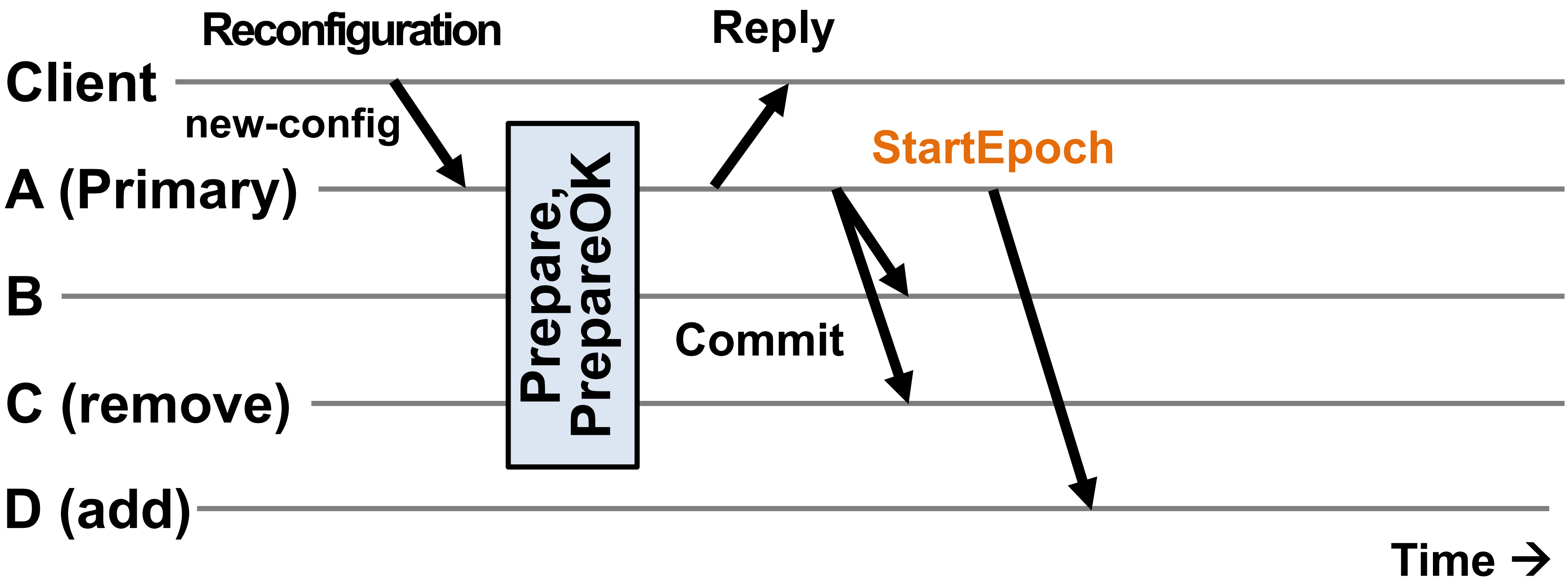
(*f* = 1)



- Primary immediately **stops** accepting new requests
- No up-call to RSM for executing this request

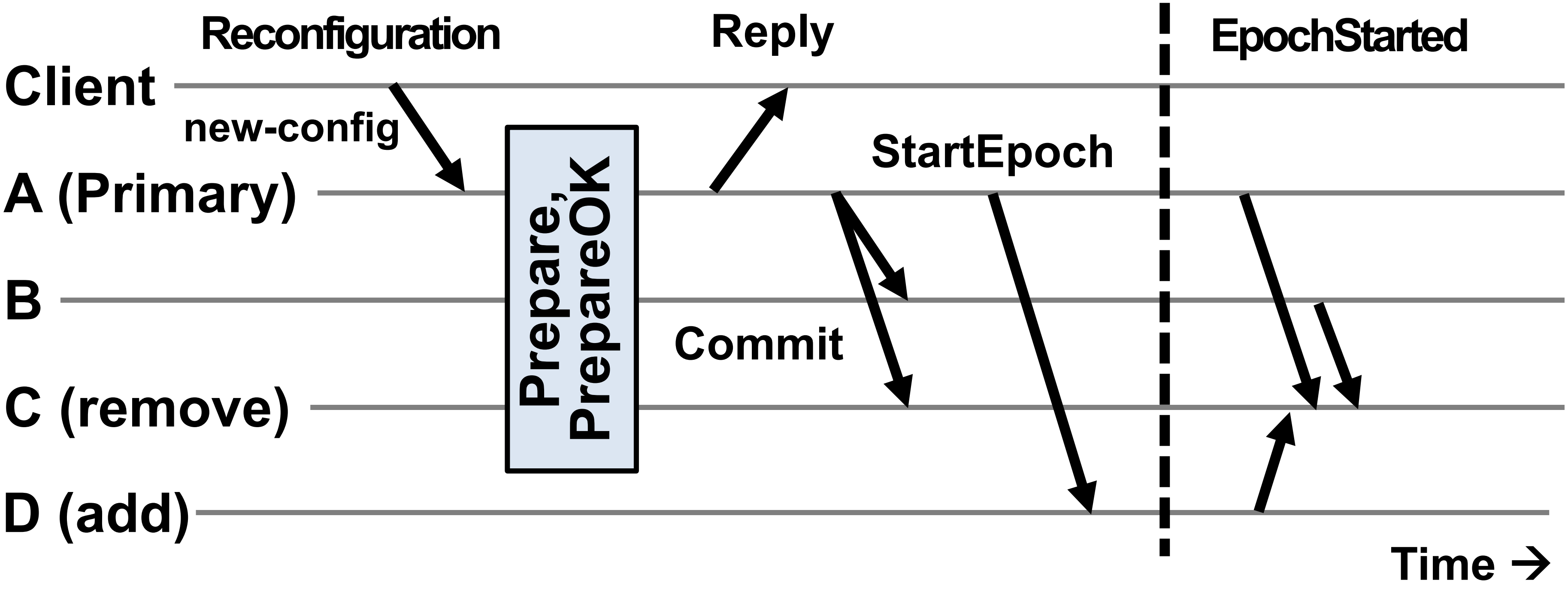
Reconfiguration (3)

(*f* = 1)



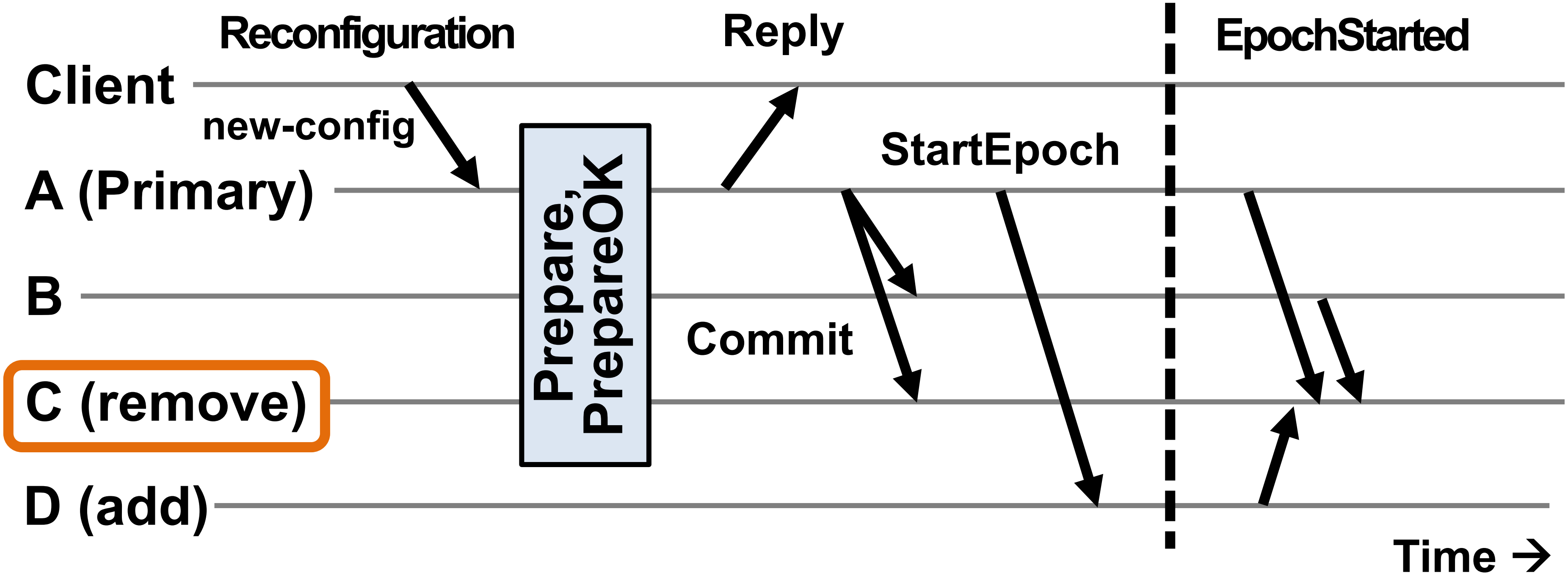
- Primary sends Commit messages to **old** replicas
- Primary sends **StartEpoch** message to **new** replica(s)

Reconfiguration in new group {A, B, D}



1. Update state with new **epoch-number**
2. Fetch state from old replicas, update log
3. Send **EpochStarted** msgs to replicas being removed

Reconfiguration at replaced replicas {C}



1. Respond to state transfer requests from others
   - Waits until it receives f’ + 1 **EpochStarted** msgs, f’ is fault tolerance of new epoch
2. Send **StartEpoch** messages to **new** replicas if they **don’t hear EpochStarted** (not shown above)

Shutting down old replicas

- If admin **doesn't wait** for reconfiguration to complete, may cause **> f failures in old group**
  - Can't shut down replicas on receiving Reply at client

- Must ensure committed requests survive reconfiguration!

- **Fix:** A new type of request **CheckEpoch** reports the current epoch
  - Goes thru normal request processing (no up-call)
  - Return indicates reconfiguration is complete

VR: Take-away ideas

- **Viewstamped Replication** is a state machine replication protocol that tolerates f crash failures in a replica group of 2f + 1 replicas

- The protocol uses replicated state to provide persistence without any use of disk

- f + 1 replicas serve as a quorum that ensures correctness; in every step of the protocol there is at least one replica that knows about the request

- There’s actually sub-protocols that operate to address distinct concerns (see next slide)

What’s useful when

- **Backups fail** or has network connectivity problems?
- Minority partitioned from primary?

→ **Quorums allow primary to continue**

- **Primary fails** or has network connectivity problems?
- Majority partitioned from primary?

→ **Rapidly execute view change**

- Replica **permanently fails** or is **removed?**
- Replica **added?**

→ **Administrator initiates reconfiguration protocol**