CAPE 272 PAPER PRESENTATION

VIEWSTAMP REPLICATION

INTRODUCTION

- Replication protocol for distributed systems
- Designed around the log replication concept of state machines
- Consensus algorithm
  - replicas must agree on the replicated state
- Guarantee a consistent view over replicated data
- Pluggable protocol
  - works on the communication layer between client-servers/server-server

FAULT TOLERANCE

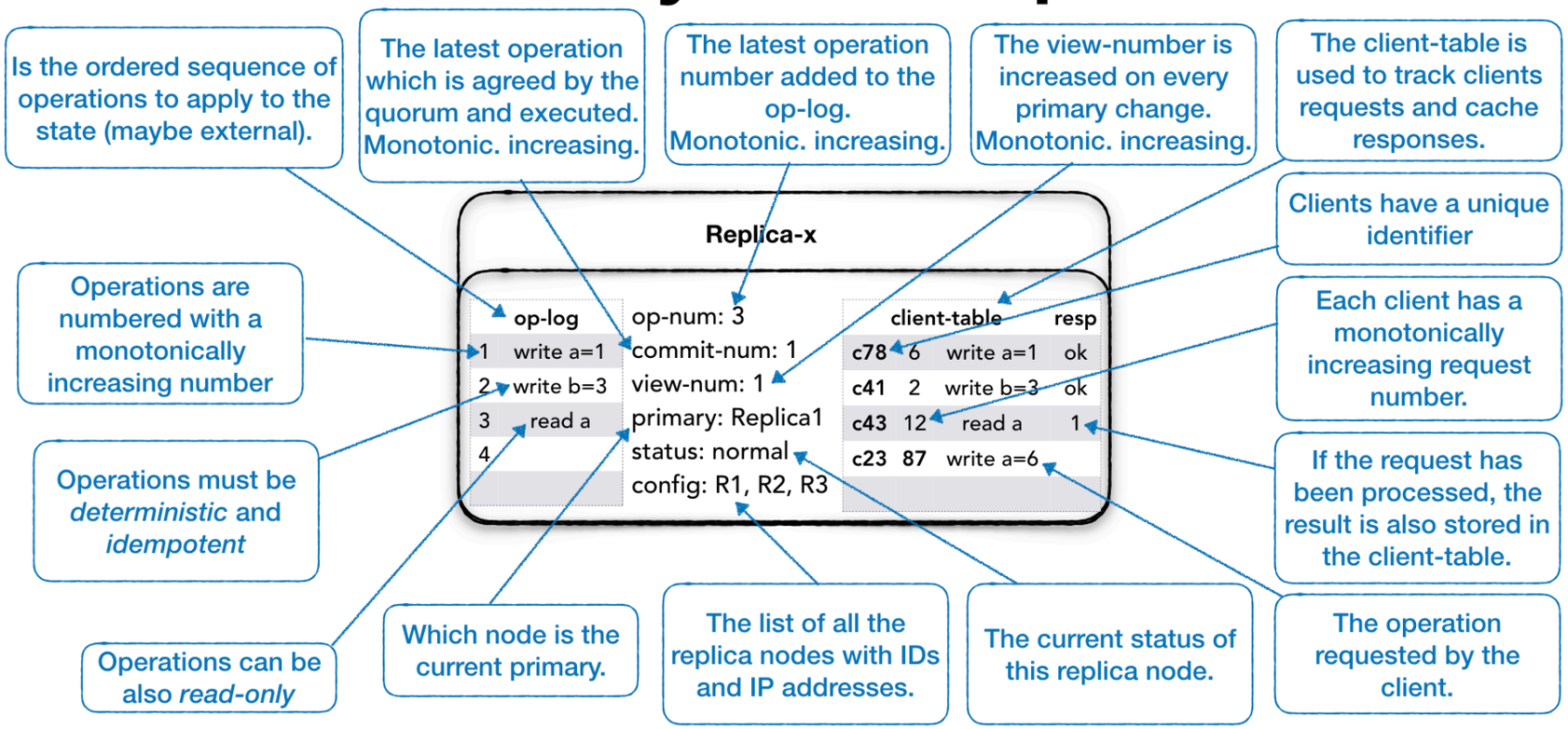
- **Achieve fault-tolerance by -**
  - Redundancy in time - to combat unreliability of network
    - allow replaying requests to overcome dropped/reordered/delayed messages
  - Redundancy in space – to make system available
    - Add redundant copies of data in different fault-domain isolate machines
    - Increases system-up time in case of hardware failure/system crashes
  - Together redundancy in time and space provide ability to tolerate and recover from system partitions, failures and network issues

FAULT TOLERANCE (CONT.)

- Limit on number of failures that can be tolerated
- To tolerate ***f*** failures, number of nodes (ensemble/cluster size) must be ***2f + 1***
- Can tolerate non-Byzantine failures-
  - All nodes will be either in working or failed state or isolated
- Quorum = ***f + 1***
- Quorum not possible with less than 3 nodes
  - Minimum ensemble size = 3

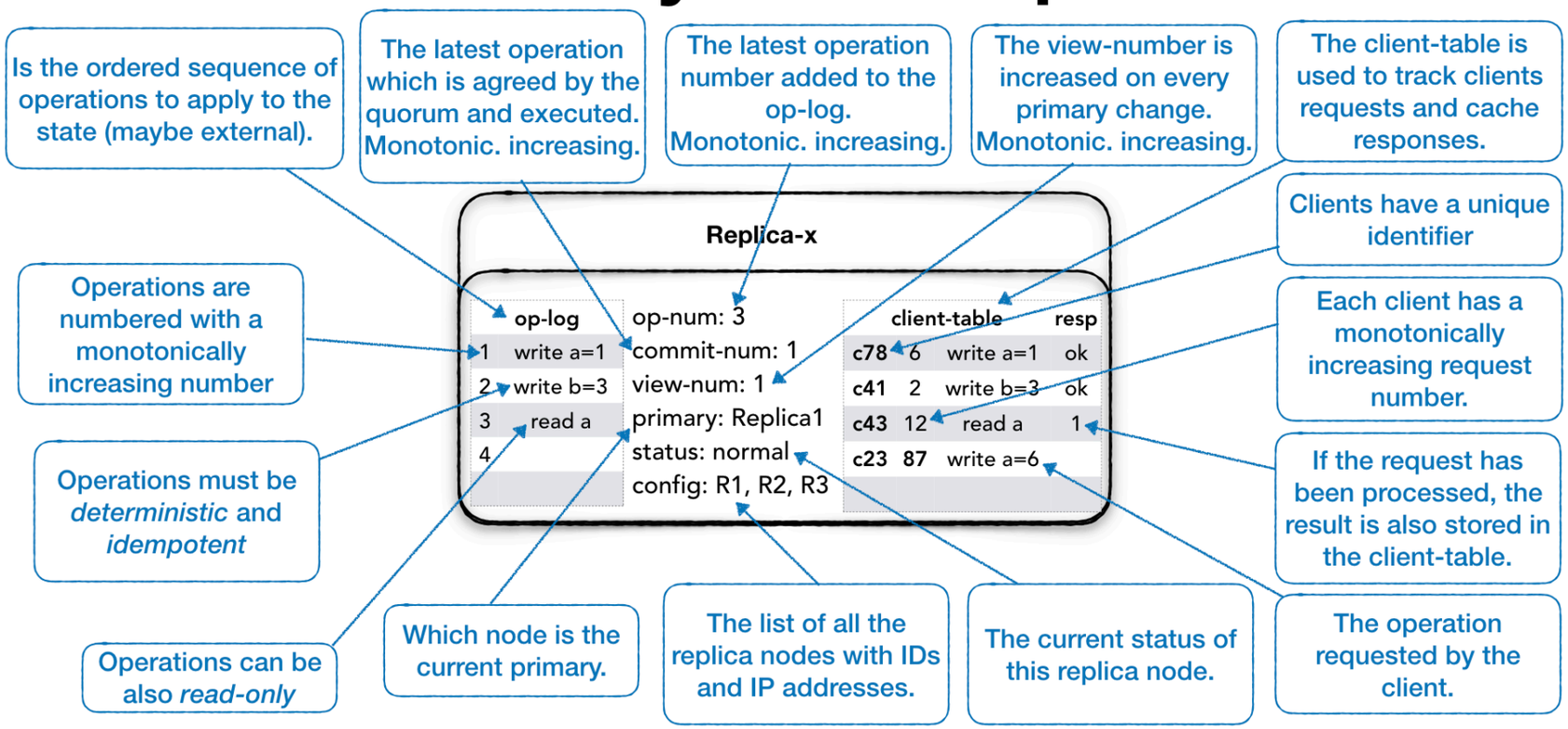
ANATOMY OF A REPLICATION

Anatomy of a Replica



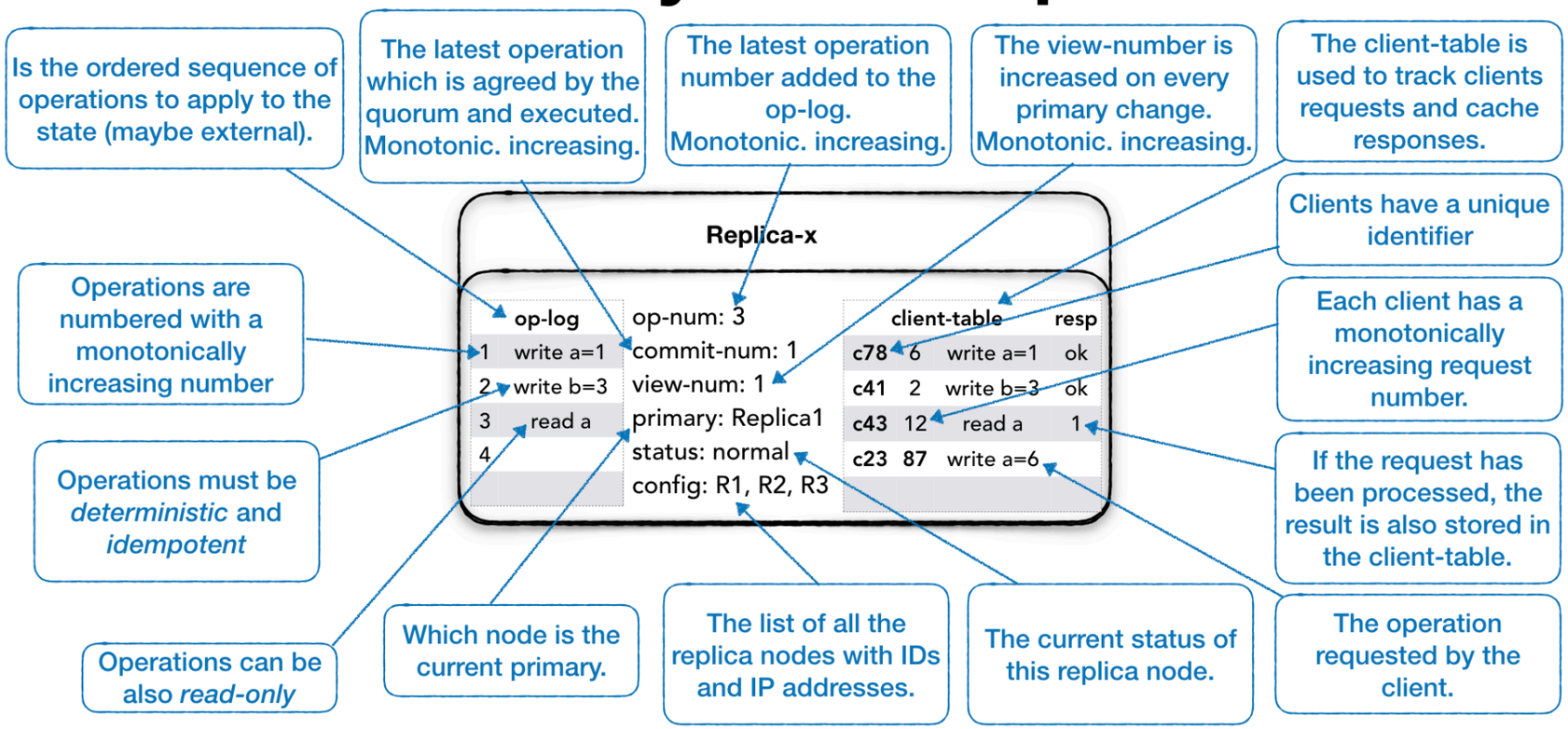
- The operation log (**Op-Log**) is a (mostly) append-only sequence of operations.
- Operations must be *deterministic* which means that every application of the same operation with the same arguments must produce the same result.
- Operations must be *idempotent* which means that if an operation is called more than once with the same input parameters, then it must not have any additional effect.
- Each operation in the operation-log has a positional identifier which is a monotonically increasing number.

Anatomy of a Replica



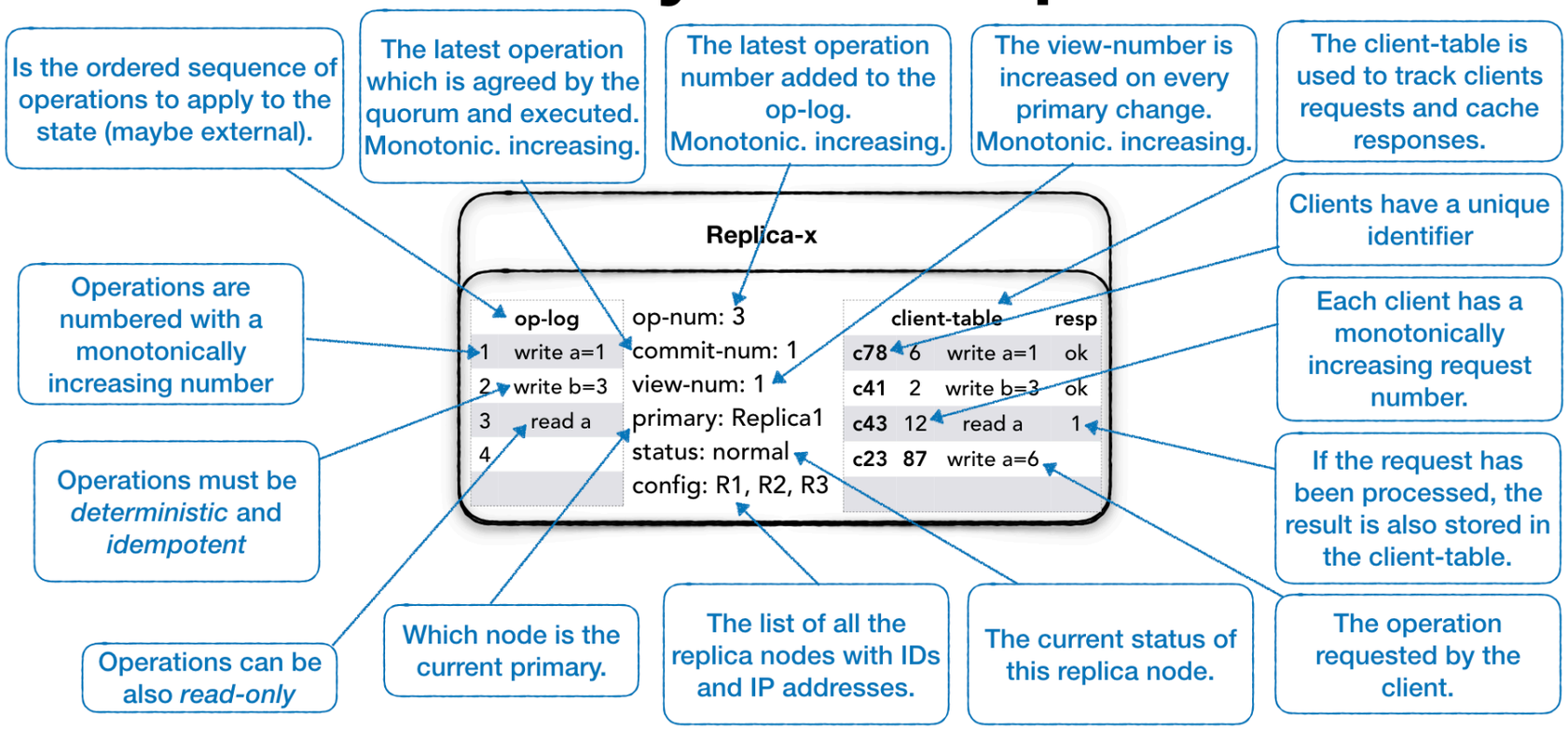
- (**Op-num**) represents the latest operation number added to the log.It is monotonic and increasing.
- Operations in the (**Op-Log**) are appended first, then shared with other replicas and once there is a confirmation that enough replicas have received the operation then it is actually executed.
- The (**commit-num**) represents the number of the last operation which was executed in the replica. It also implies that all the previous operations have been executed as well. (**commit-num**) is a monotonically increasing number.

Anatomy of a Replica



- The *view-number* (**view-num**) is a monotonically increasing number which changes every time there is a change of primary replica in the ensemble.
- Each replica must also know who the current primary replica is. This is stored in the **primary** field.
- The **status** field shows the current replica operation mode. As we will see later, the **status** can assume different values depending on whether the replica is ready to process client requests, or it is getting ready and doing internal preparation.

Anatomy of a Replica

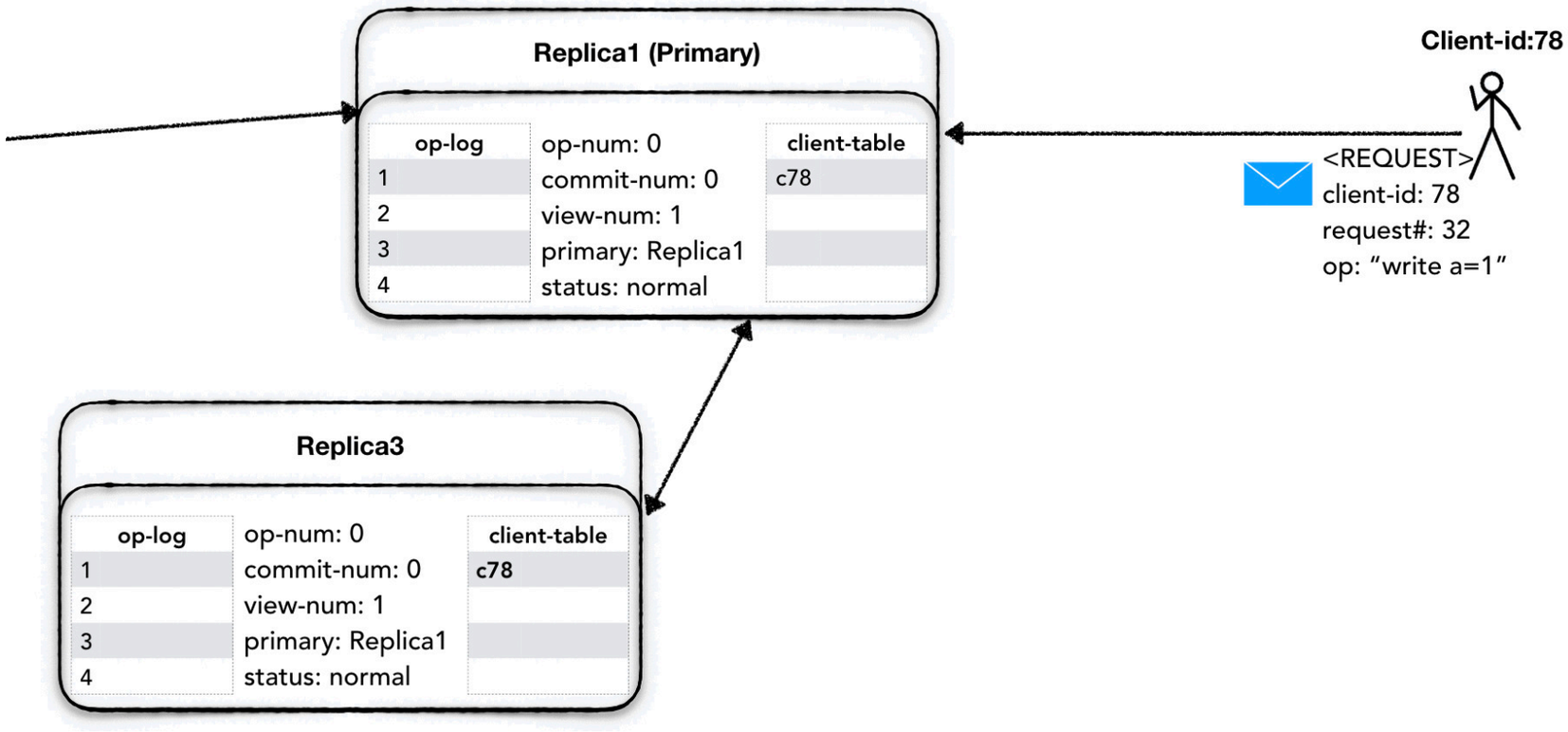


- Every replica node will also have a list of all the replica nodes in the ensemble with their IP addresses and their unique identifiers. Some parts of the protocol require the replicas to communicate with other replicas therefore they must know how to contact the other nodes.
- The **client-table** is used to keep track of client's requests. Clients are identified by a unique id, and each client can only make one request at the time. Since communication is assumed to be unreliable, clients can re-issue the last request without the risk of duplication in the system. Every client request has a monotonically increasing number which identifies the request of a particular client. Each time the primary receives a client requests it add the request to the client table. If the client re-sends the same requests because didn't receive the response the primary can verify that the request was already processed and send the cached response.

CLIENT REQUEST HANDLING

THE PROTOCOL

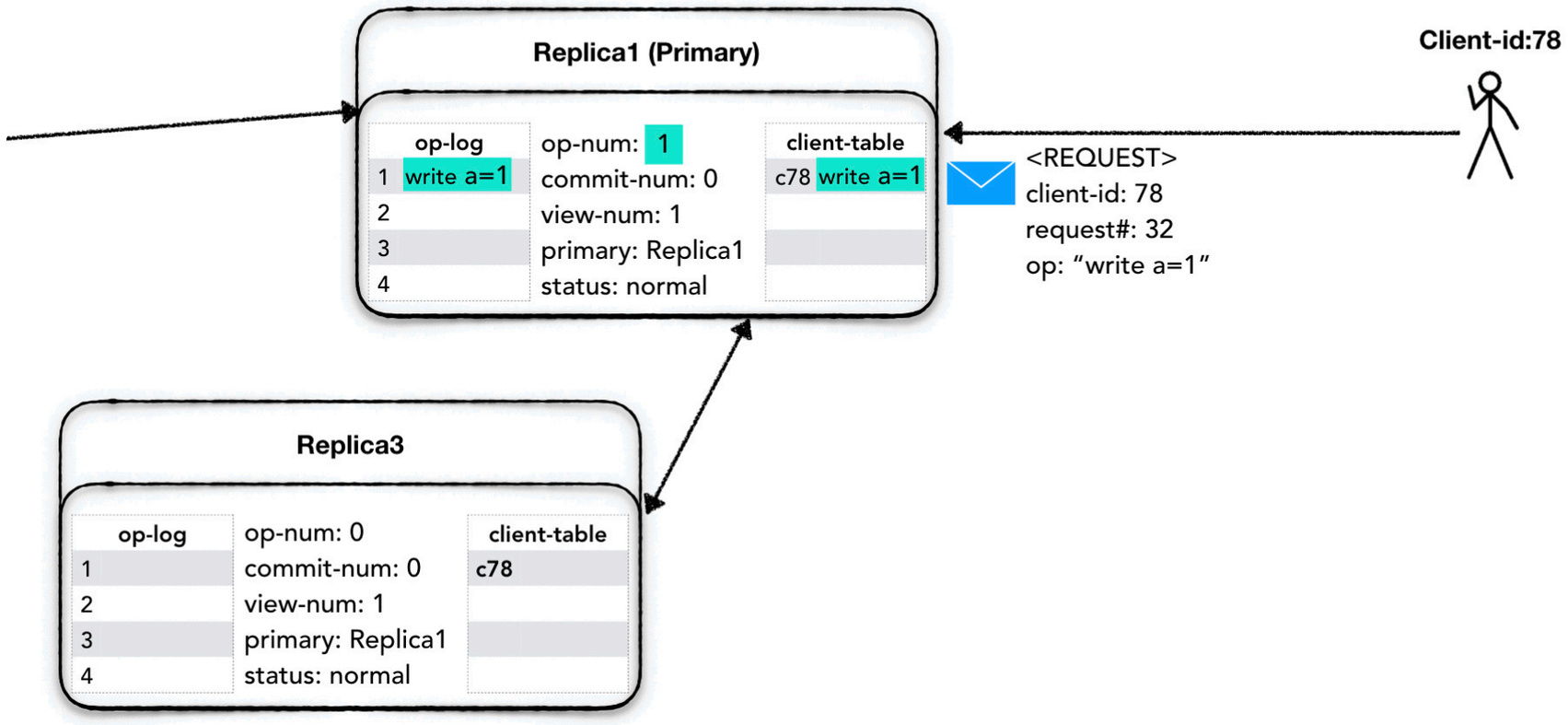
Protocol: client requests



Clients have a unique identifier (client-id), and they communicate only with the primary.

The client prepares <REQUEST> message which contains the client-id, the request (request#) and the operation (op) to perform.

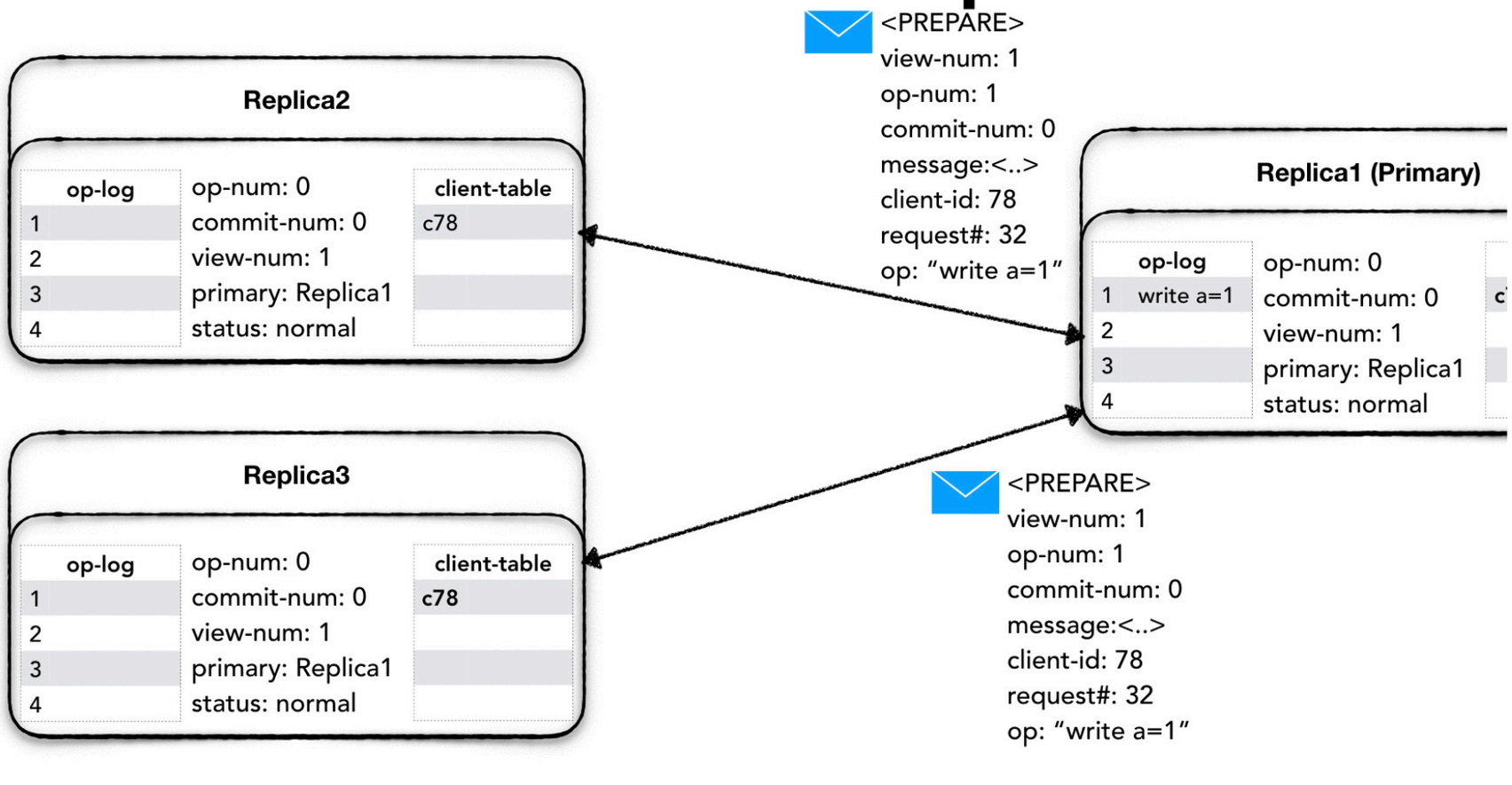
Protocol: client requests



If the request’s number is greater of the one in the client table then it is a new request.

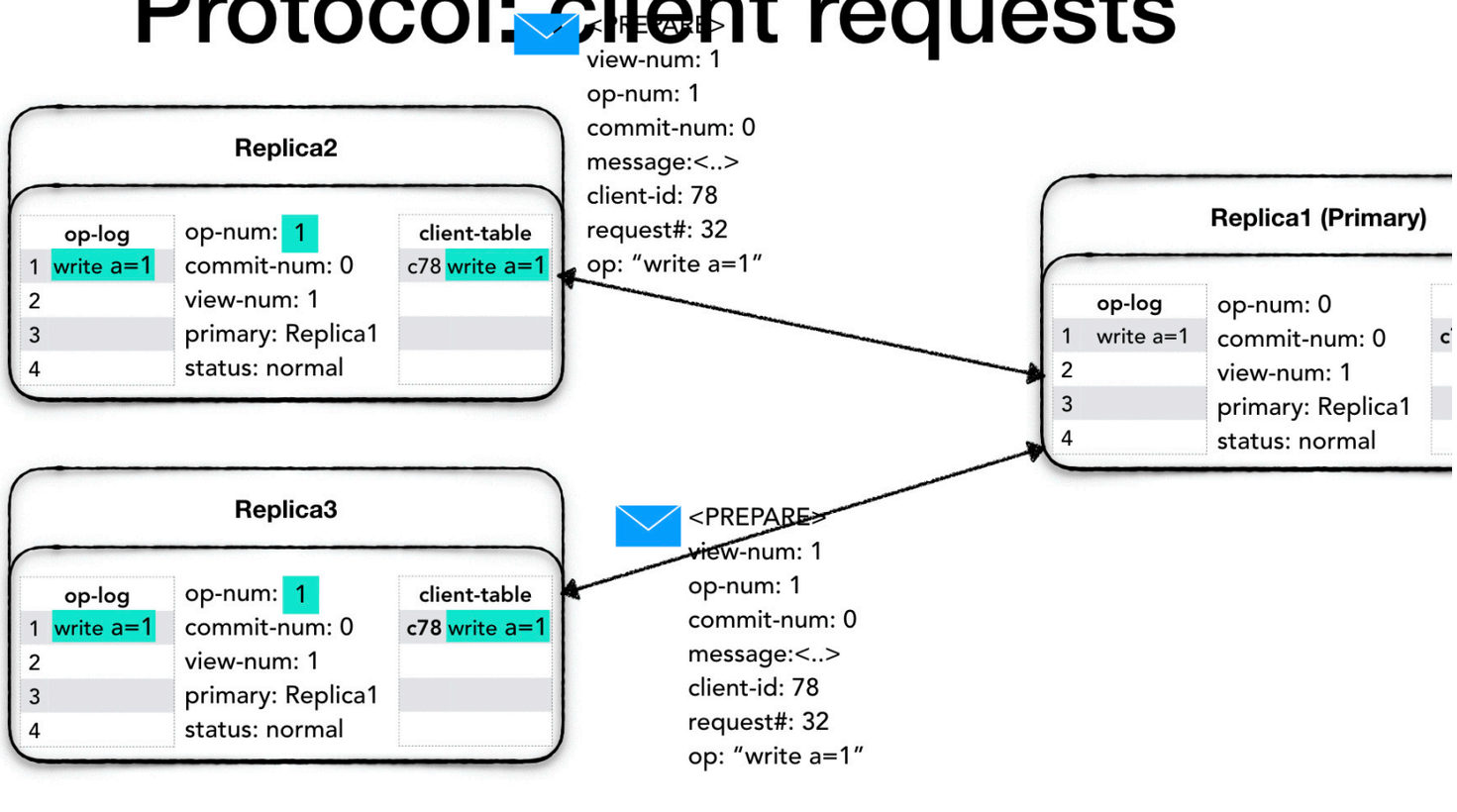
If it is a new request, then the primary increases the operation number (op-num), it appends the requested operation to its operation log (op-log) and it updates the client-table with the new request.

Protocol: client requests



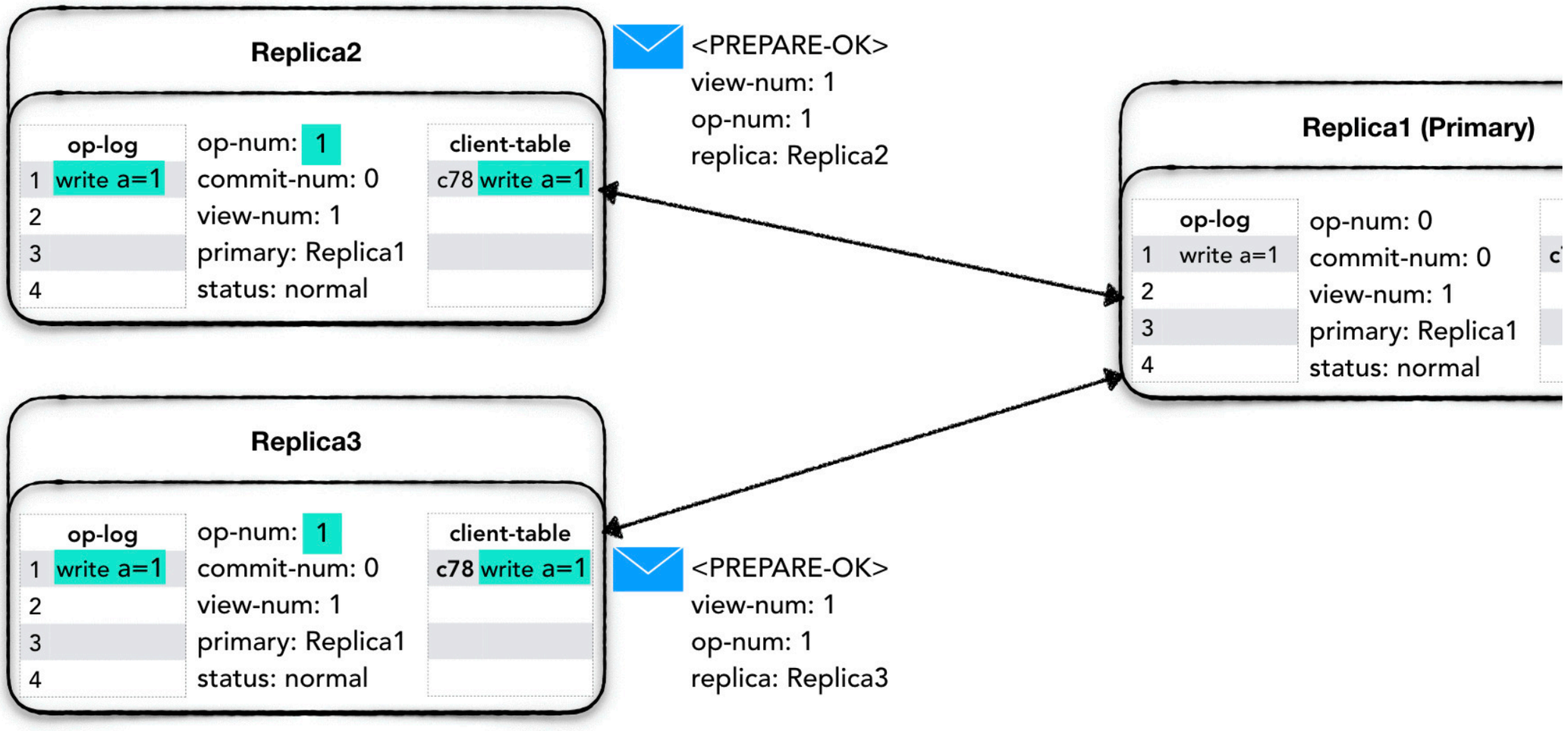
When a replica receives a <PREPARE> request, it first checks whether the view number is the same view-num, if its view number is different than the message view-num it means that a new primary was nominated and, depending on who is behind, it needs to get up to date.

Protocol: client requests



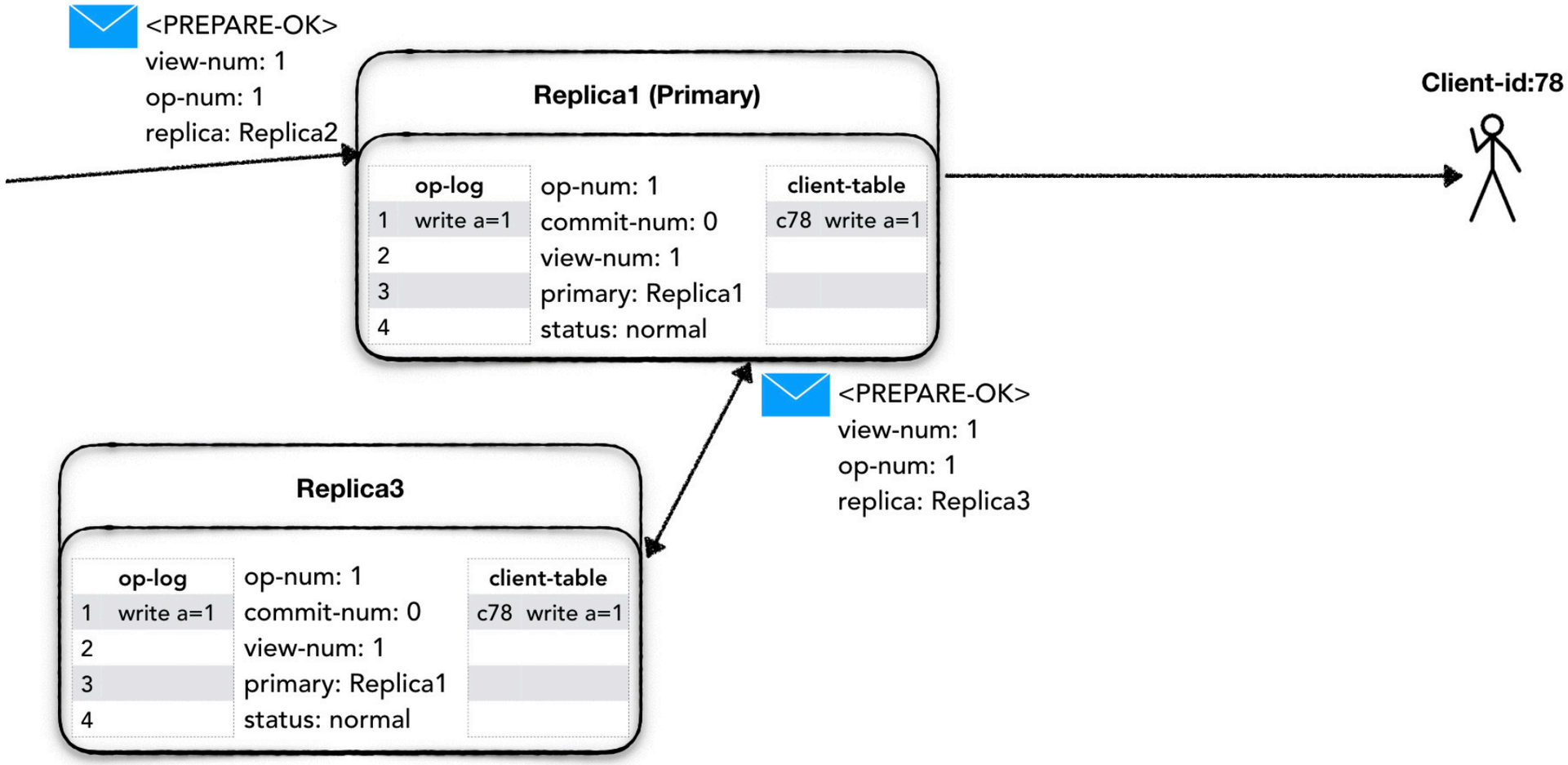
If the op-num is strictly consecutive then it increments its op-num, appends the operation to the op-log and updates the client table.If there are gaps it means that this replica missed one or more <PREPARE> messages, so it drops the message and it initiates a recovery with a state transfer.

Protocol: client requests



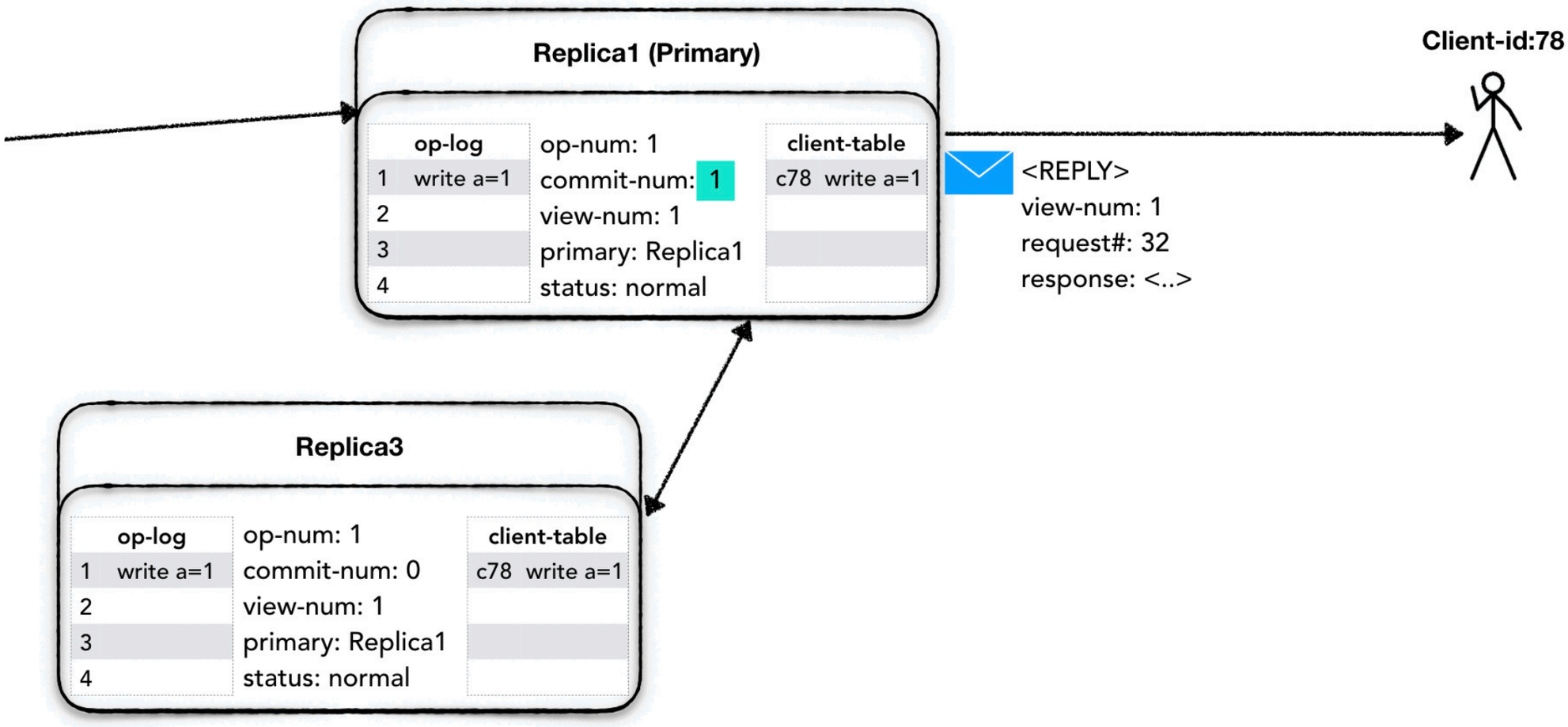
Replica creates a <PREPARE-OK> message with its view-num, the op-num and its identity. Sending a <PREPARE-OK> for a given op-num means that all the previous operations in the log have been prepared as well (no gaps).

Protocol: client requests



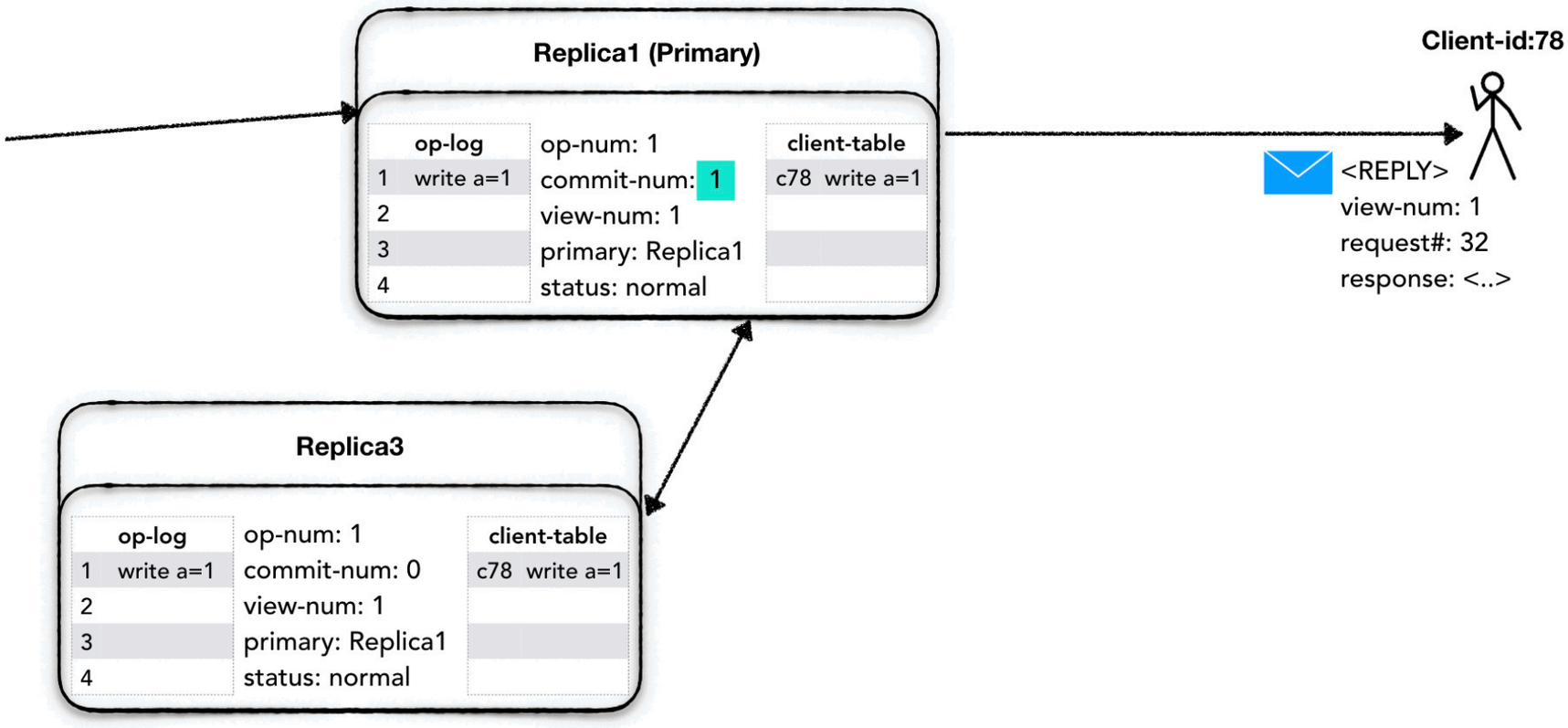
The primary waits for f + 1 including itself <PREPARE-OK> messages, at which point it knows that a quorum of nodes knows about the operation to perform therefore it is considered safe to proceed as it is guaranteed that the operation will survive the loss of f nodes.

Protocol: client requests



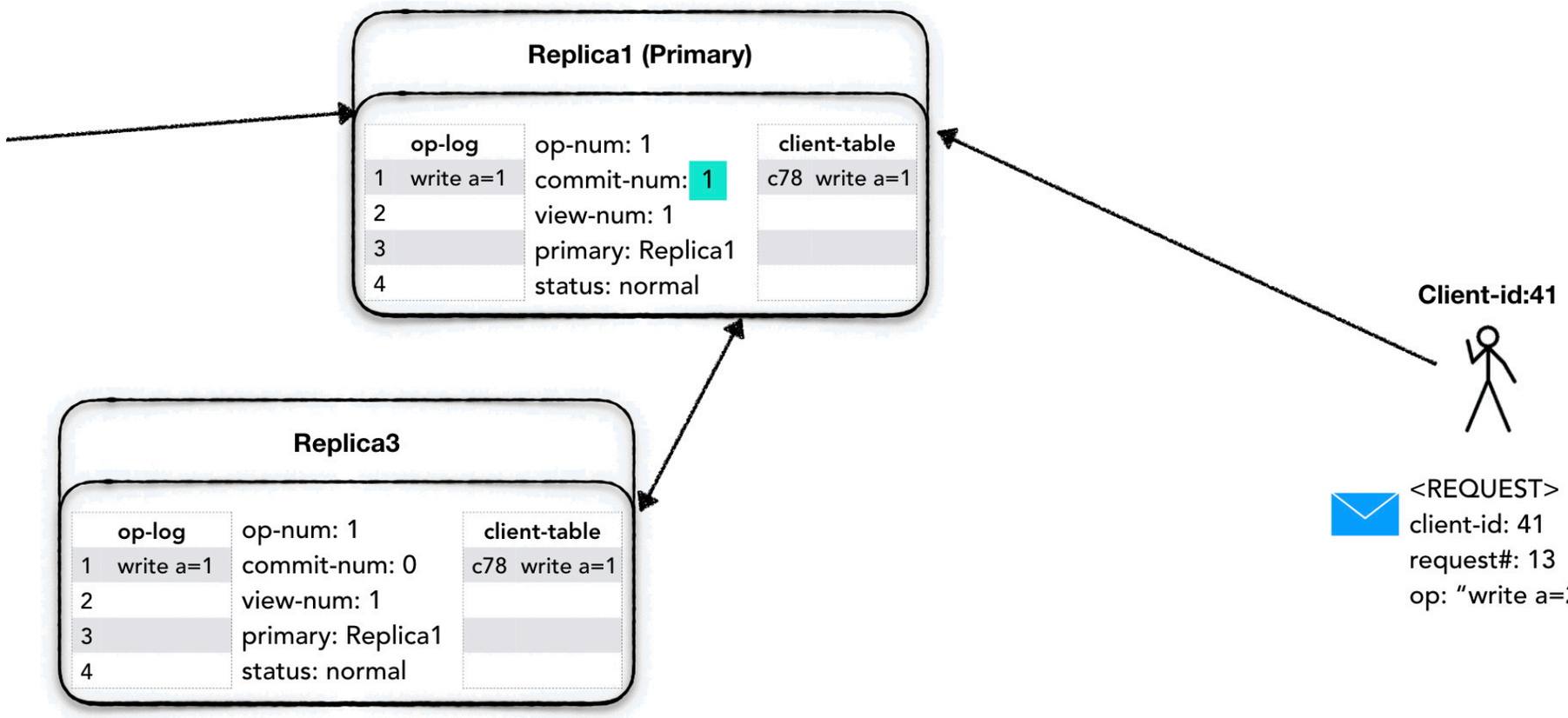
When it receives enough <PREPARE-OK> ,then it performs the operation, it increases its commit number commit-num, and update the client table with the operation result.

Protocol: client requests



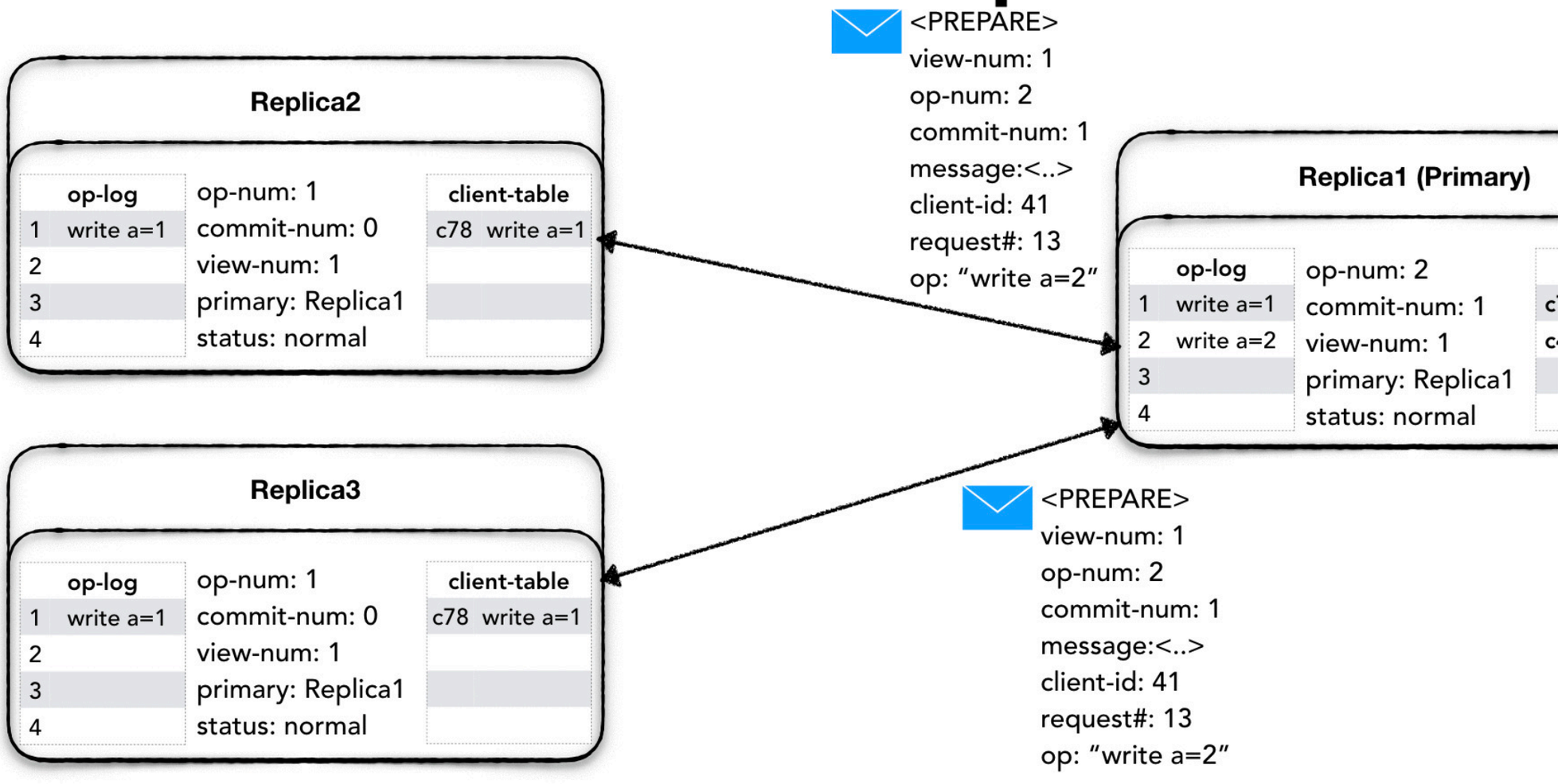
Finally, it prepares a <REPLY> message with the current view-num the client request number request# and the operation result (response) and it sends it to the client.

Protocol: client requests

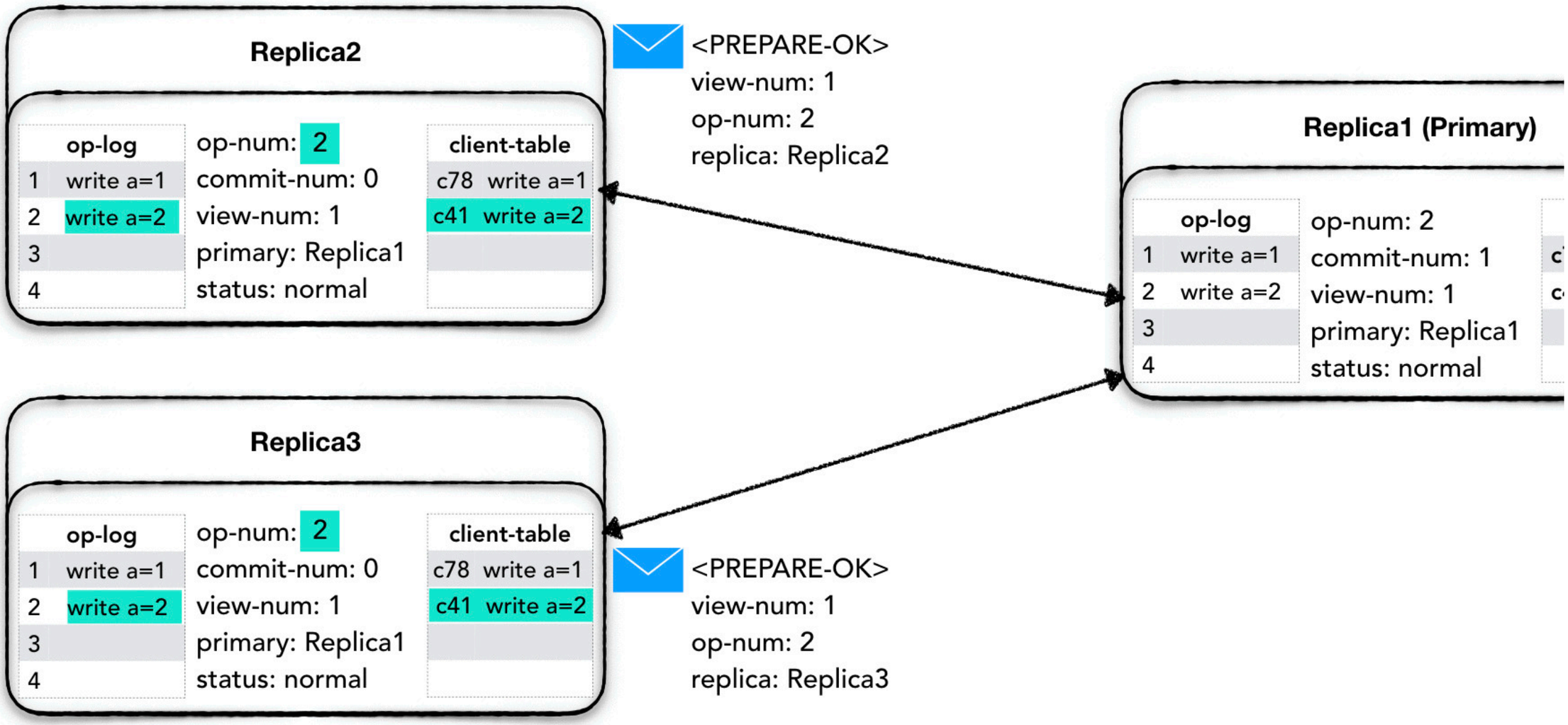


Now there is a new request from client, the whole process would happen again but this time primary also includes its commit-num which now shows that the primary advanced its position.

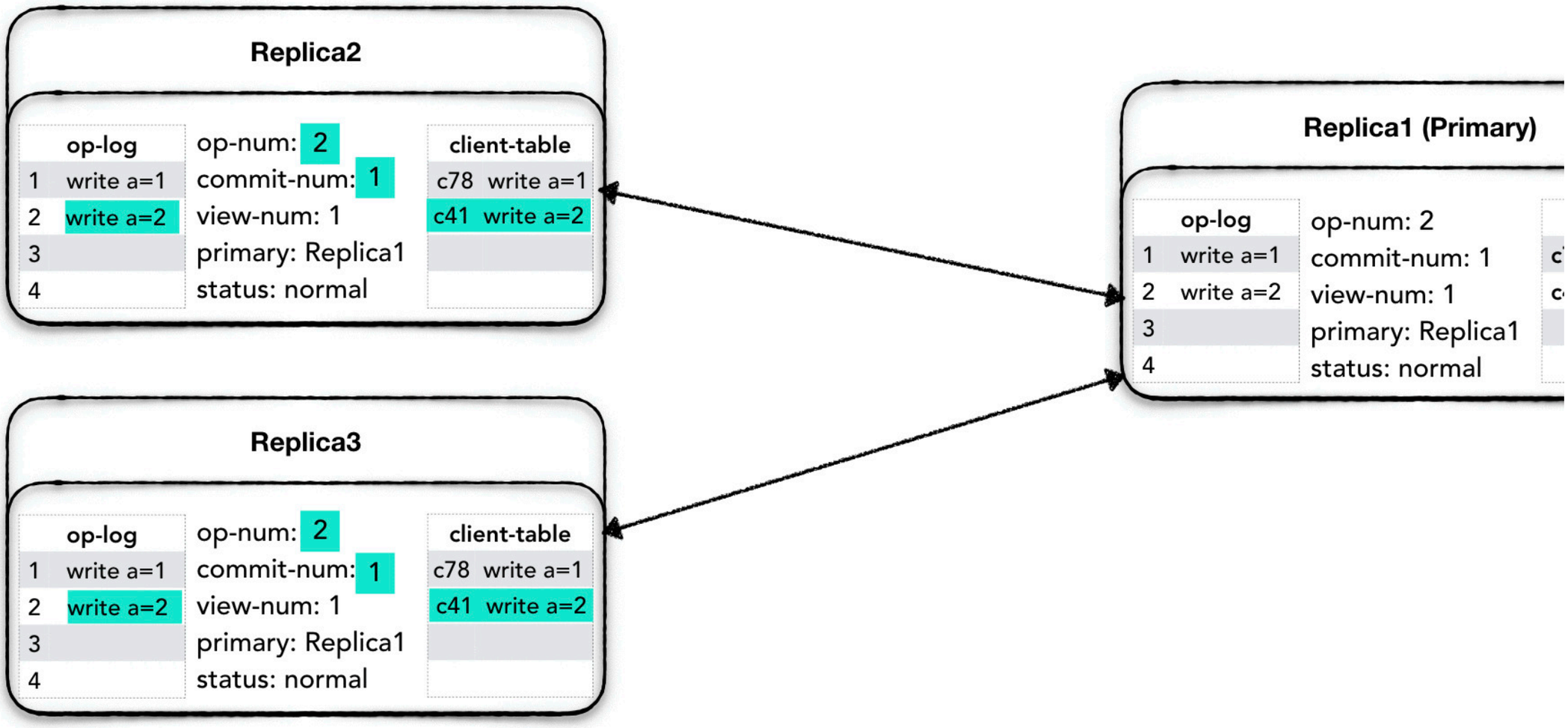
Protocol: client requests



Protocol: client requests



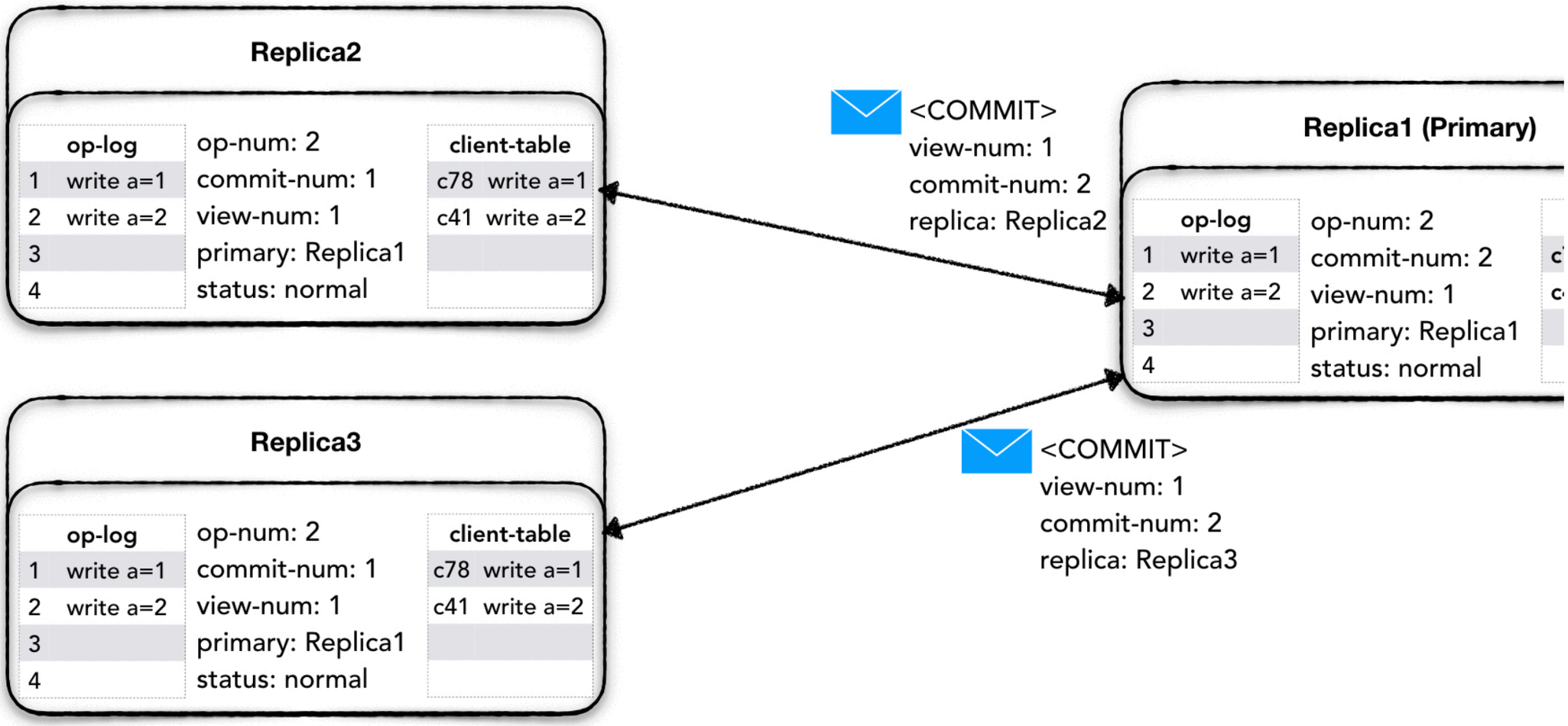
Protocol: client requests



What if the system is experiencing a low request rate, maybe because it is the middle of the night and all users are asleep?

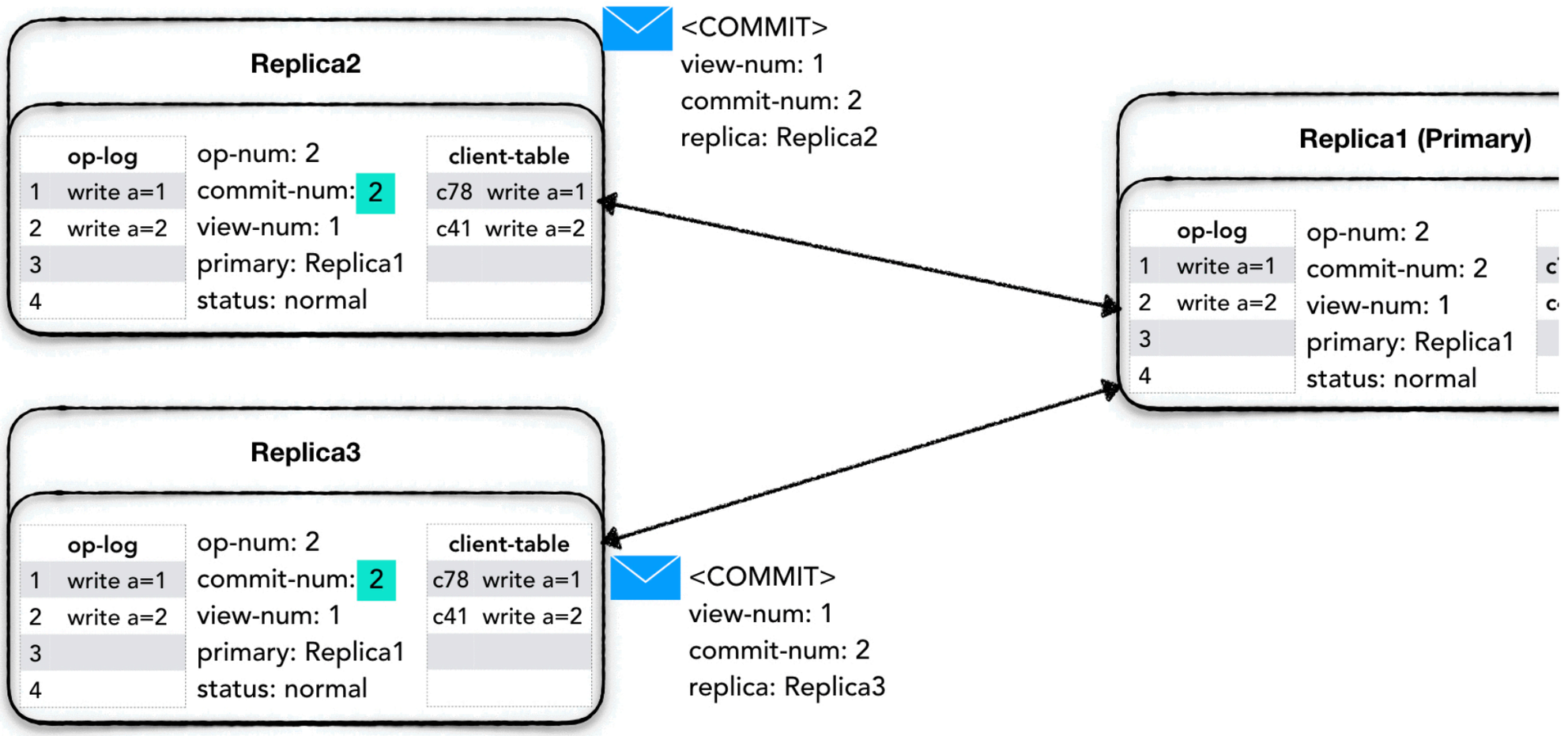
If the primary doesn't receive a new client request within a certain time then it will create a <COMMIT> message to inform replicas about which operations can now be performed.

Protocol: client requests



The <COMMIT> message will only contain the current view number (view-num) and the last committed operation (commit-num).

Protocol: client requests



The <PREPARE> message together with the <COMMIT> message act as a sort of heartbeat for the primary. Replicas use this to identify when a primary might be dead and elect a new primary. This process is called view change and it is what Akshay is going to discuss next.

VIEW CHANGE PROTOCOL

VIEW CHANGE

- The view change protocol is used when one of the replicas detect that the current primary is unavailable.
- If a quorum of nodes agrees then the system will transition to a new primary.

LEADER SELECTION

- Uses a **deterministic function** over a fixed property of the replica nodes.
- A unique identifier or IP address.
- A simple sort function can be used to determine the next primary node.

PRIMARY SELECTION EXAMPLE

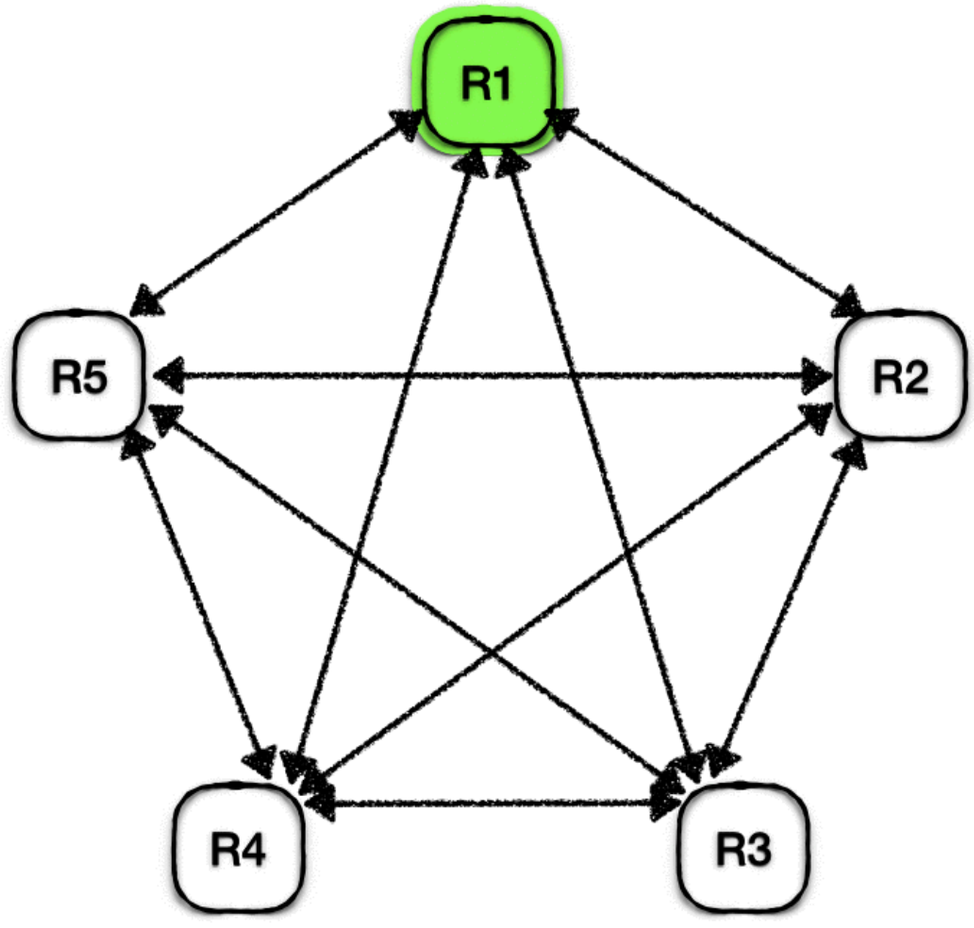
```
10.5.3.12                10.5.1.20  (1)
10.5.1.20  ~sort~>       10.5.2.28  (2)
10.5.2.28                10.5.3.12  (3)
```

Sorting a list of IP addresses

- The The node **`10.5.1.20`** will be the first primary
- When that node dies **`10.5.2.28`** becomes the next primary.
- followed by **`10.5.3.12`** and then starting from the beginning again.

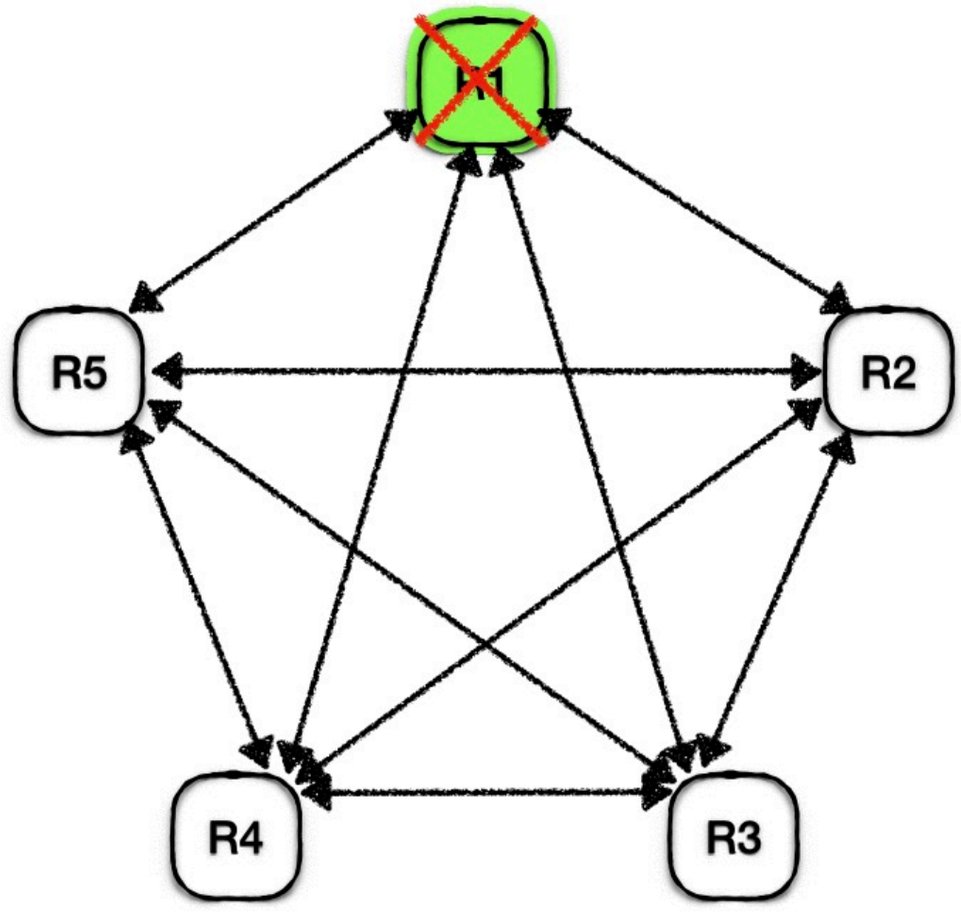
FAULT TOLERANCE EXAMPLE

- Replicas R1 to R5 are all up and running.
- R1 is the first primary node. Next is R2.
- The cluster can survive 2 failed nodes.
- R1 is accepting client's requests.
- For each client request, a **<Prepare>** message is sent to all replicas
- They reply with a **<Prepare-Ok>**.
- **<commit>** messages are also used to signal which operations are committed by the primary.



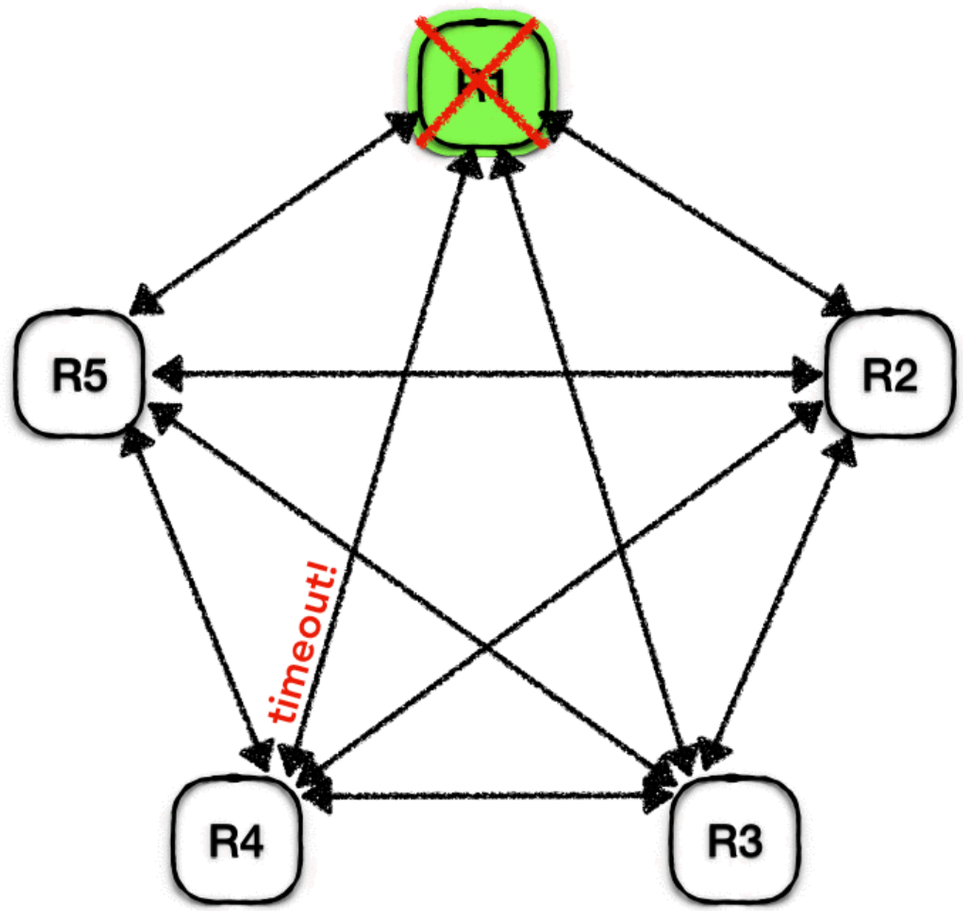
FAULT TOLERANCE EXAMPLE

- R1 fails or crashes.
- No more commit messages are sent.
- Commit acted as the heartbeat.



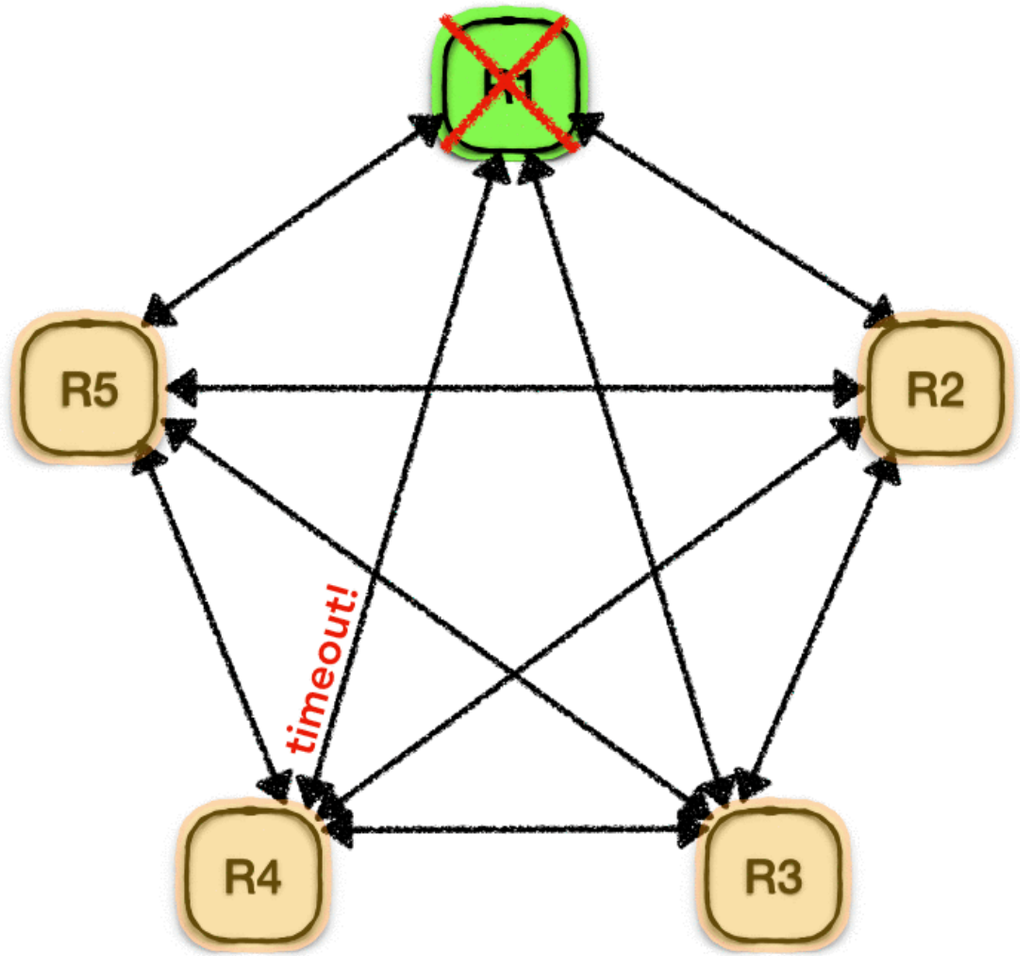
FAULT TOLERANCE EXAMPLE

- Timeout happens for one of the replicas
- R4 detects the primary's failure



FAULT TOLERANCE EXAMPLE

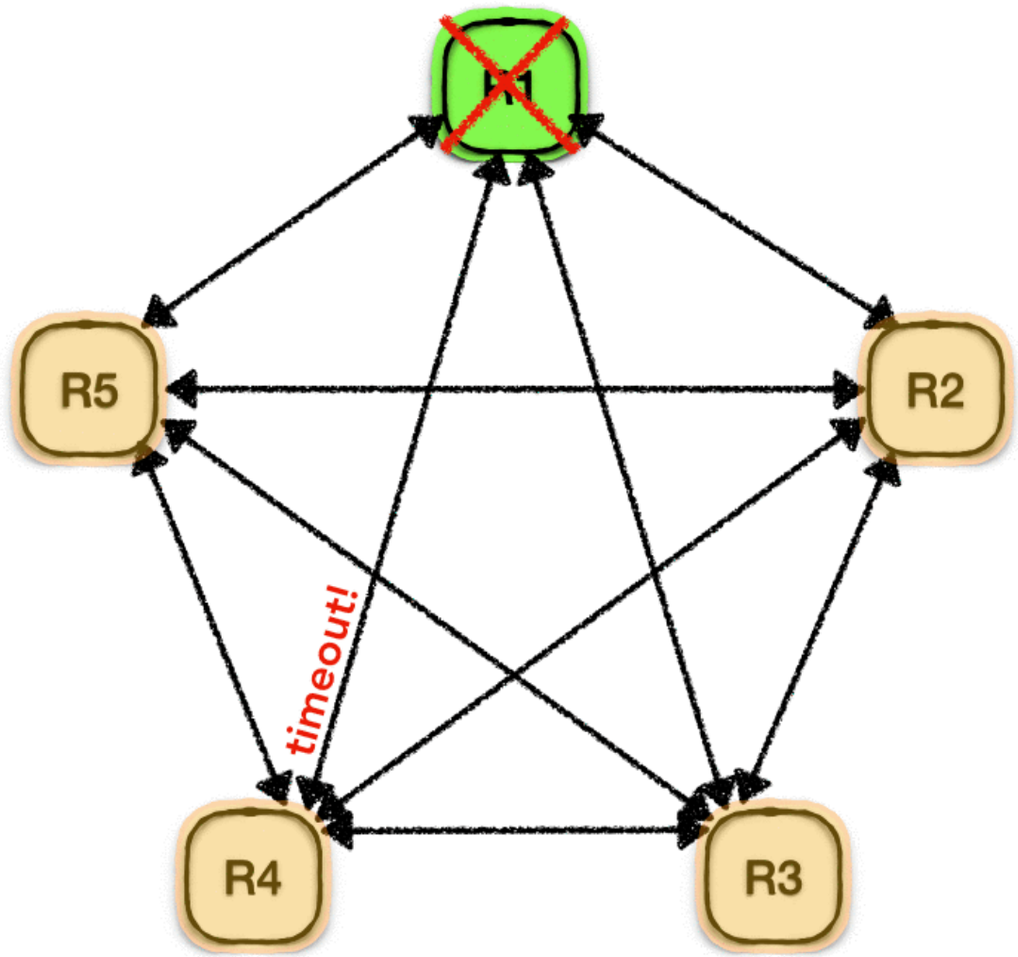
- R4 changes :
  - `status` to `view-change`
  - Increase its view number `(view-num)`
  - Creates a `<start-view-change>` message with new view-num
  - It will then send this message to all the replicas.



<START-VIEW-CHANGE>
view-num: 2
replica: R4

FAULT TOLERANCE EXAMPLE

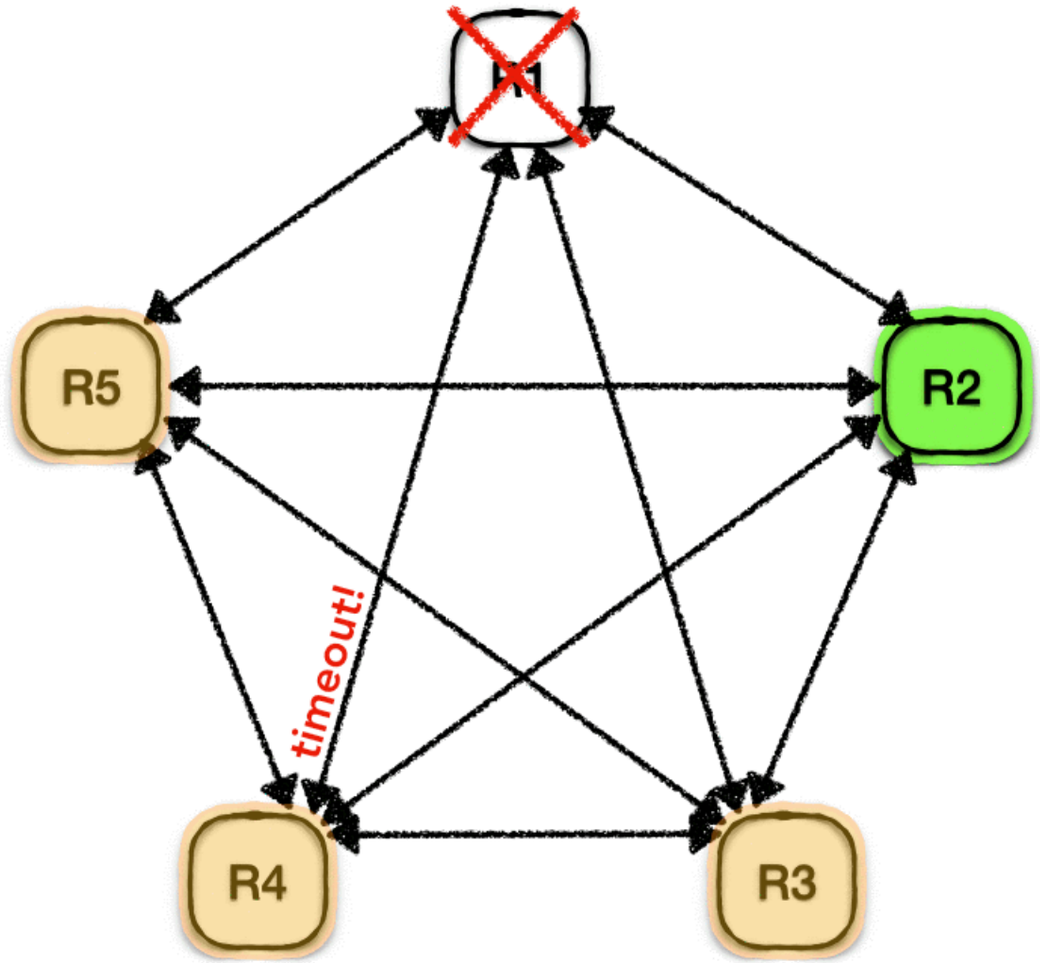
- If the received view number is bigger than their own:
  - `status` to `view-change`
  - Set their `view-num` to the received `view-num`
  - Send a `<START-VIEW-CHANGE>` message to all replicas.



<START-VIEW-CHANGE>
view-num: 2
replica: R4

FAULT TOLERANCE EXAMPLE

- When a replica receives f+1 **<START-VIEW-CHANGE>** messages (including itself):
  - Sends a **<DO-VIEW-CHANGE>** message to the new primary.

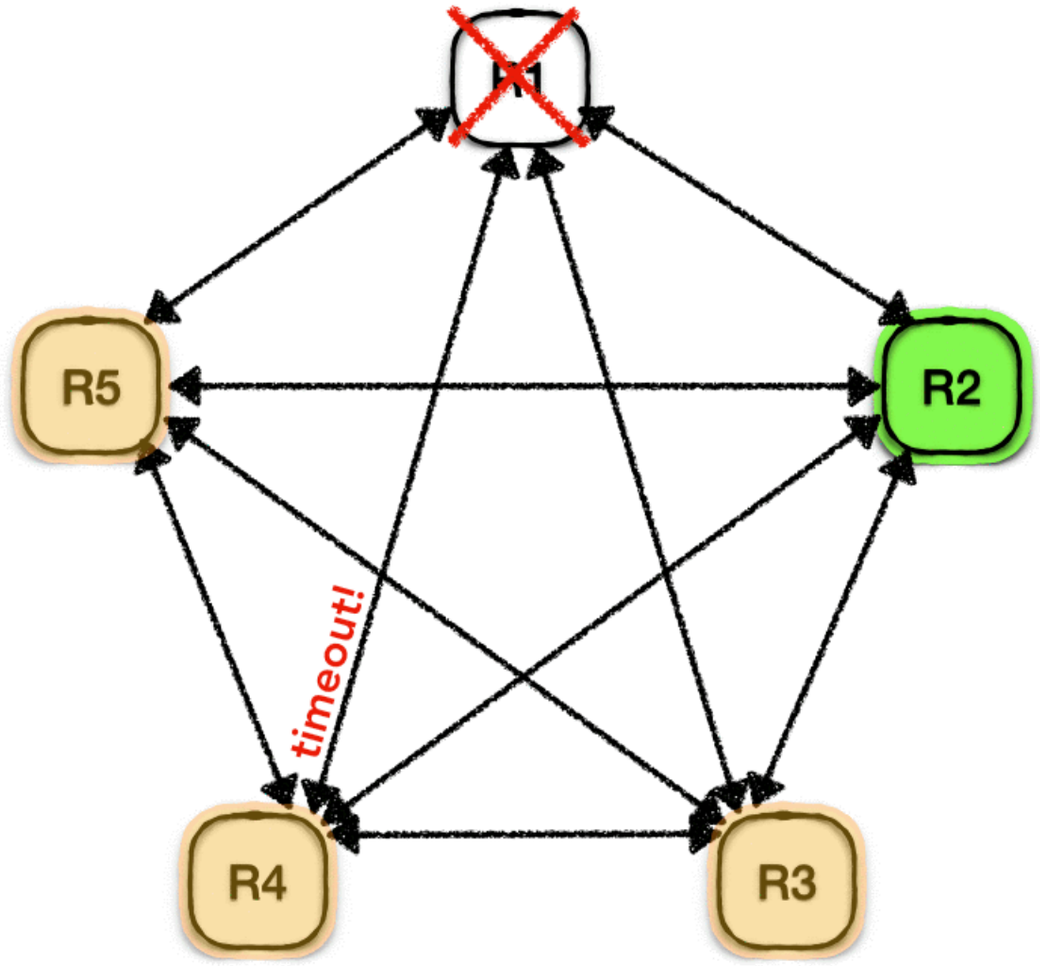


<DO-VIEW-CHANGE>
view-num: 2
old-view-number: 1
op-log: {…}
op-num: α
commit-num: β

<START-VIEW-CHANGE>
view-num: 2
replica: R4

FAULT TOLERANCE EXAMPLE

- When new primary receives f+1 `<DO-VIEW-CHANGE>` messages (including itself):
  - It compares the messages against its own information.
  - and pick the most up-to-date information.

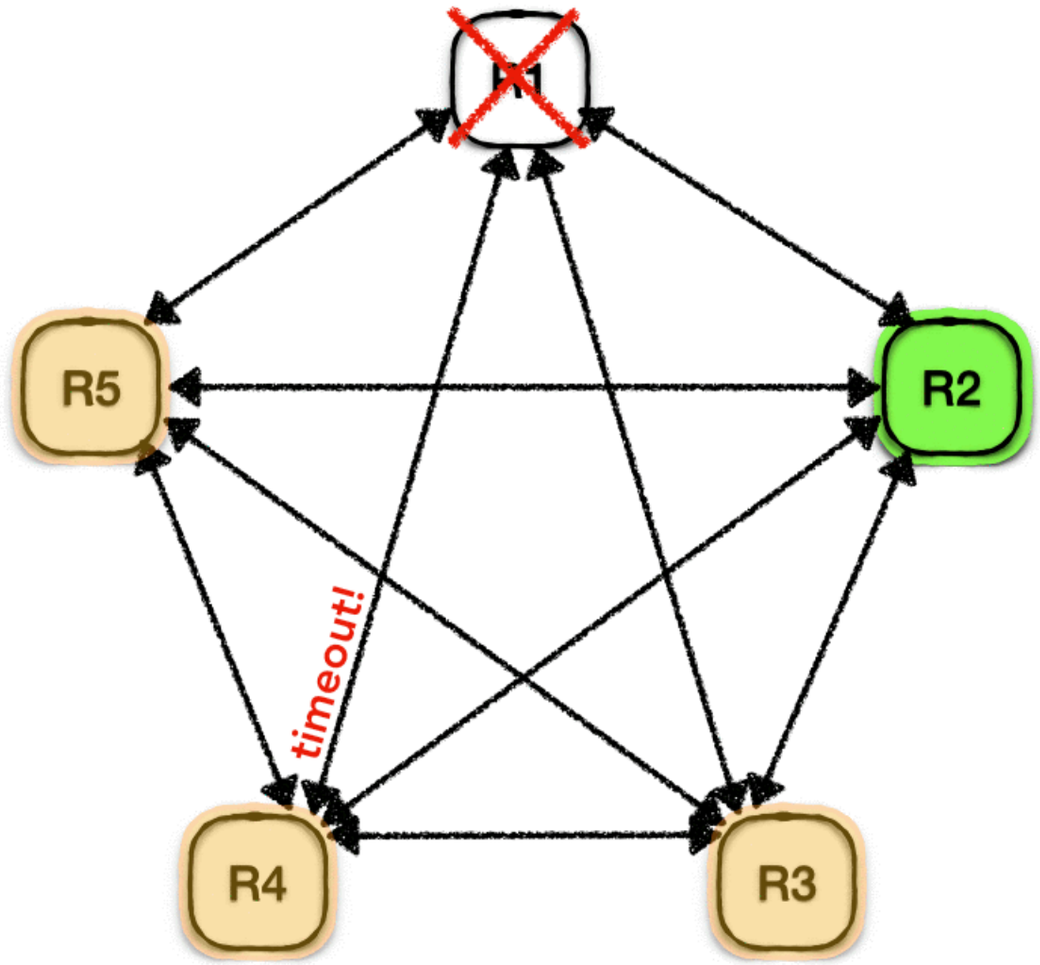


<DO-VIEW-CHANGE>
view-num: 2
old-view-number: 1
op-log: {…}
op-num: α
commit-num: β

<START-VIEW-CHANGE>
view-num: 2
replica: R4

FAULT TOLERANCE EXAMPLE

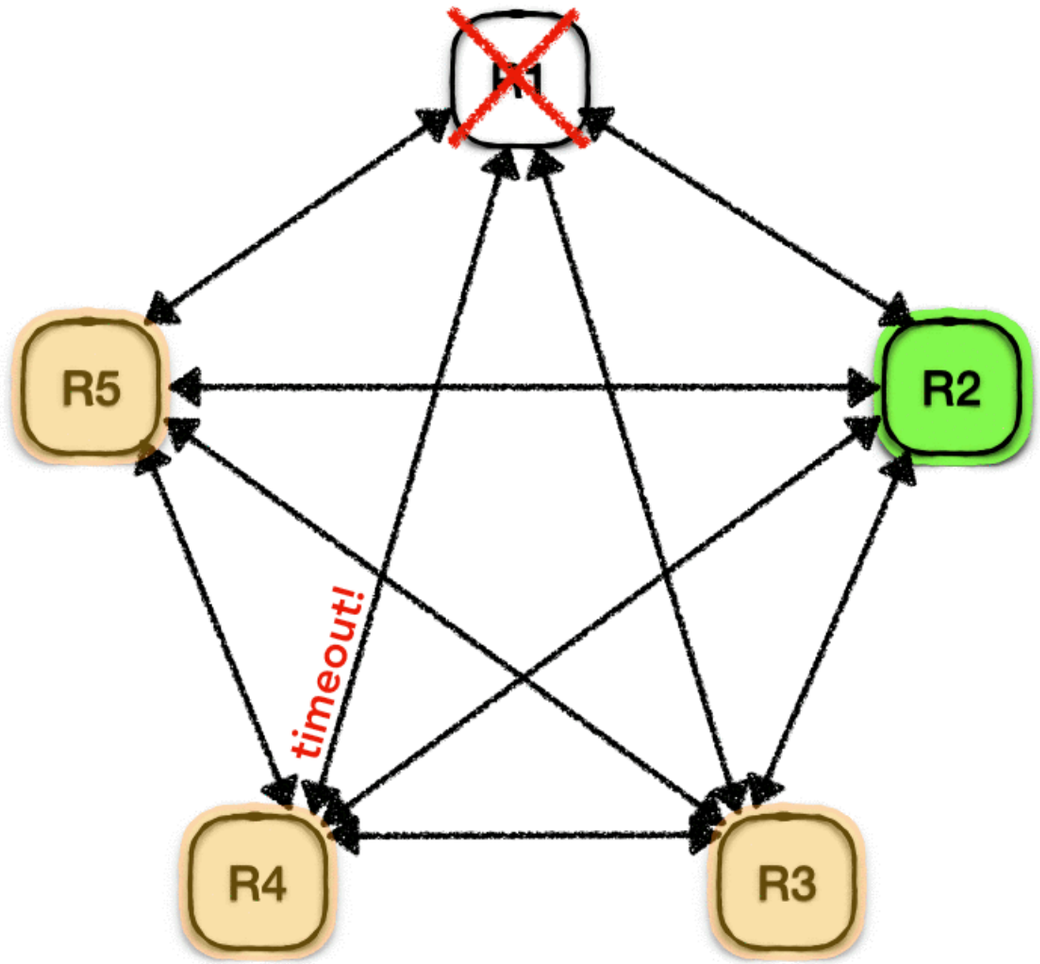
- New primary ready to accept requests:
  - Changes status to normal.
  - Sends a <START-VIEW> message to all the replicas.
- Most up-to-date view-num, commit-num, op-num, op-log.



<START-VIEW>
view-num: 2
op-log: {…}
op-num: α
commit-num: β

FAULT TOLERANCE EXAMPLE

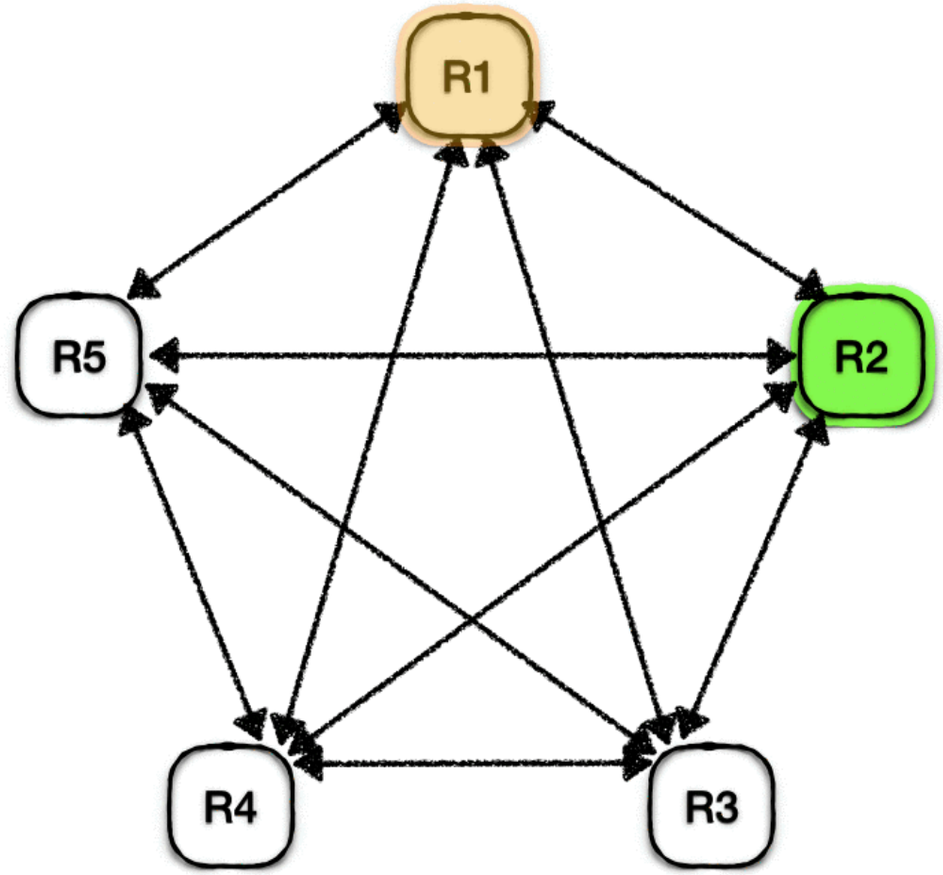
- All replicas receive <START-VIEW>
  - Replace their info with info in <START-VIE> message.
  - Change their status to normal.



<START-VIEW>
view-num: 2
op-log: {…}
op-num: α
commit-num: β

FAULT TOLERANCE EXAMPLE

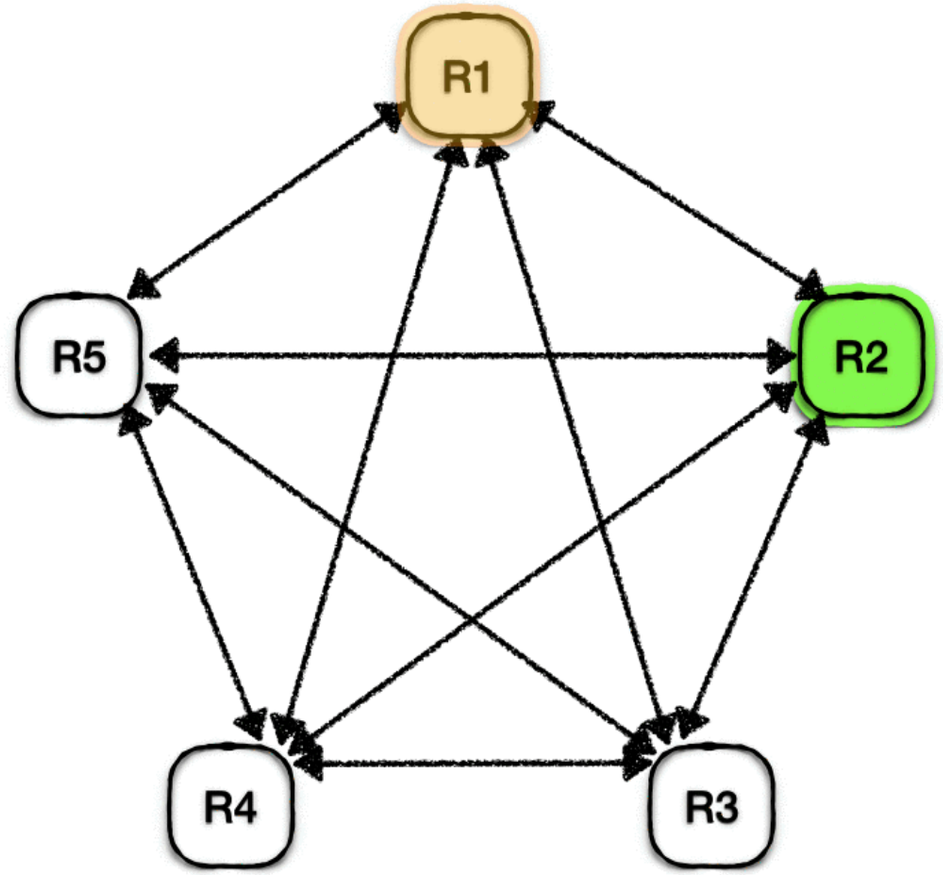
- Old primary might come back after some time
  - Change their status to recovery.
- Sends a <GET-STATE> message.



<GET-STATE>
view-num: 1
op-num: α'
commit-num: β'
replica: R1

FAULT TOLERANCE EXAMPLE

- The replicas receiving the <GET-STATE> message replies with <NEW-STATE>



<NEW-STATE>
view-num: 2
op-log: {β'…}
op-num: α
commit-num: β