Replicated Distributed Systems

For ~~Fun~~ and ~~Profit~~

Performance Reliability

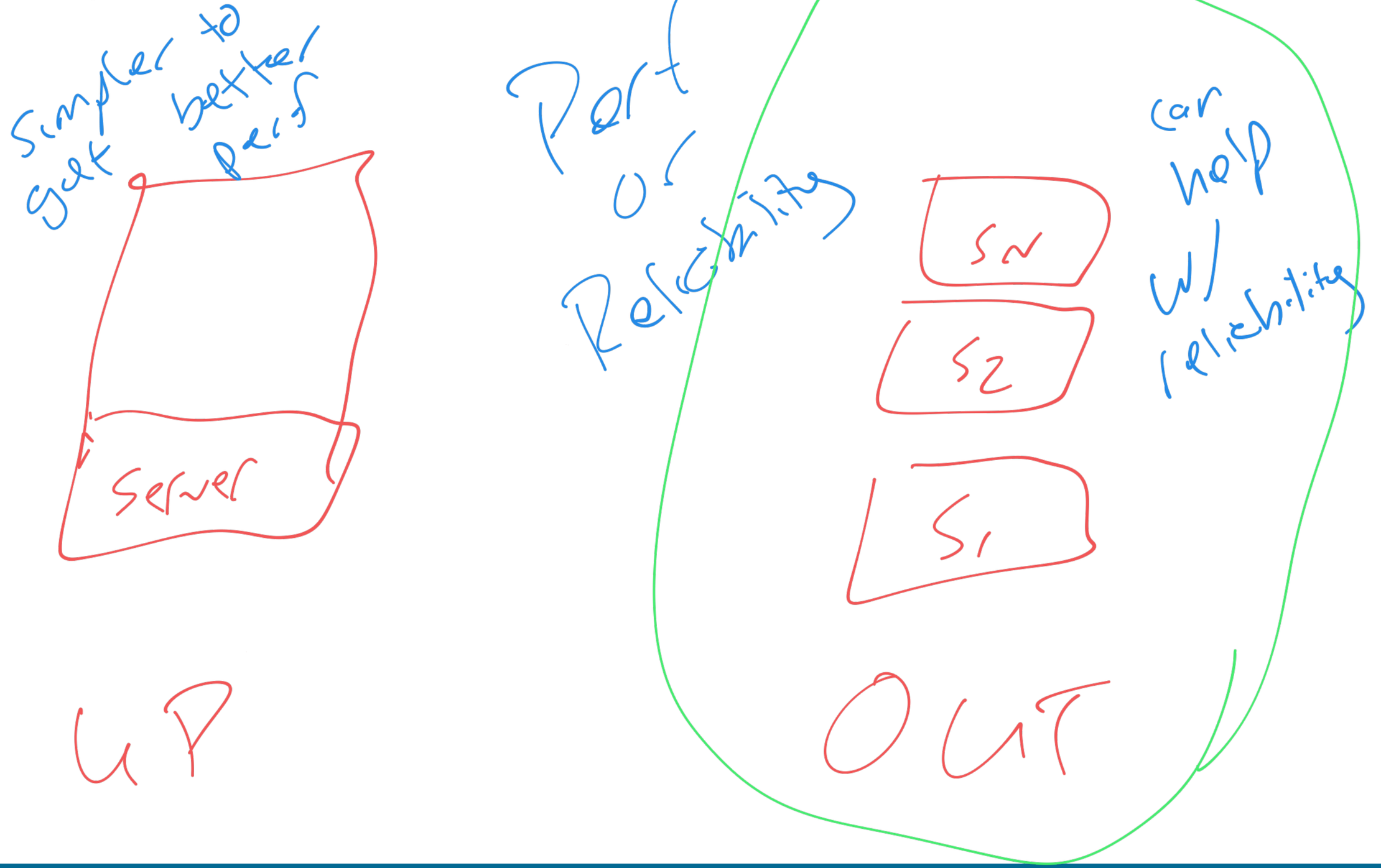
My new startup…

FaceTwit

- Post a picture of yourself and a 144 character note
- No history of past pictures/posts

Growing FaceTwit

Scale Up vs Scale Out



Growing FaceTwit

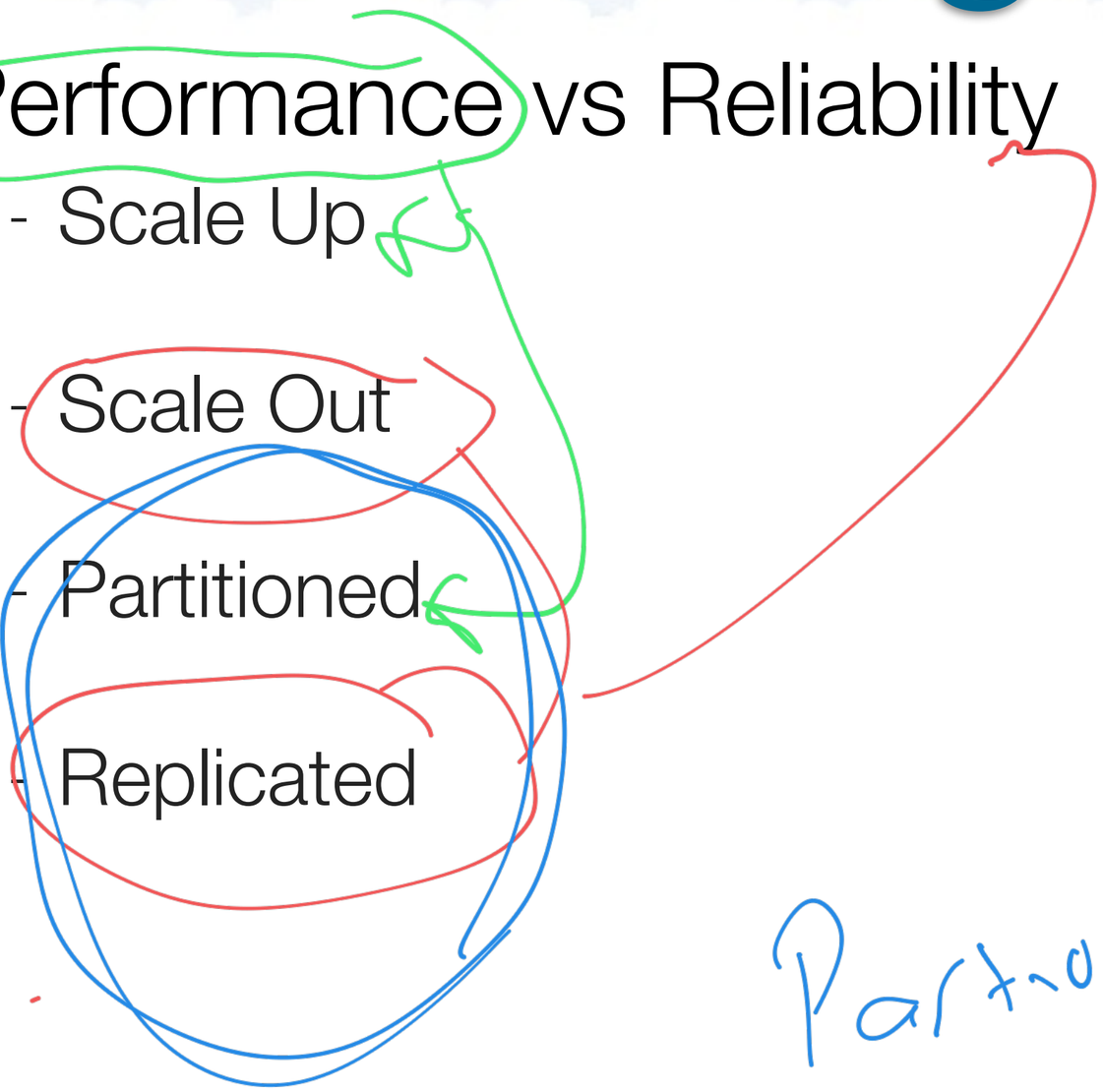
Replicated vs Partitioned

ID	Name	Face	Post
1	Tim	t.jpg	“Last class…”
2	Lucas	l.jpg	“At Capital …”
…	…	…	…
5	Gabe	g.jpg	“Composite…”

Growing FaceTwit

Performance vs Reliability

- Scale Up
- Scale Out
- Partitioned
- Replicated



FaceTwit 2.0!

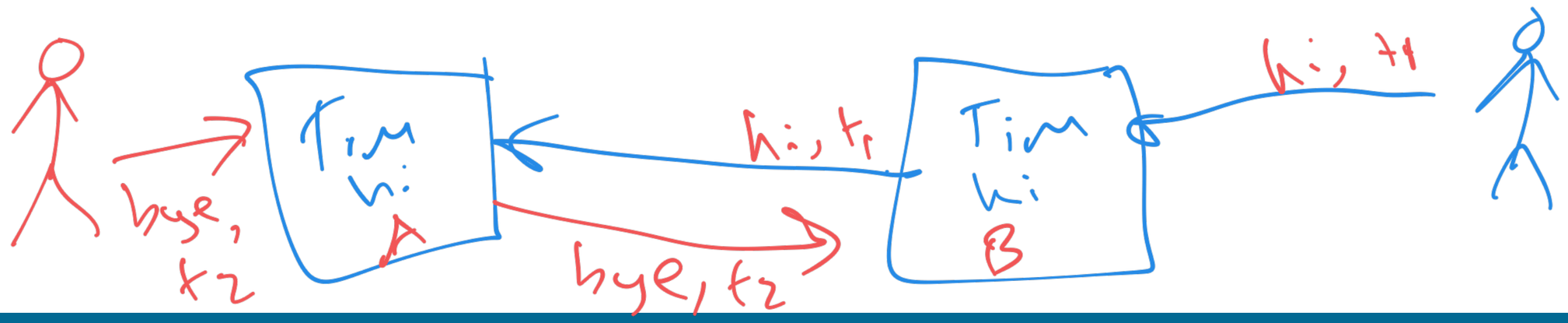
FaceTwit

- Post a picture of yourself and a 144 character note
- No history of past pictures/posts

v2.0 adds…

- Other users can post a comment on a profile page
- My page should show the most recent comment posted

Why will this make consistency more challenging?

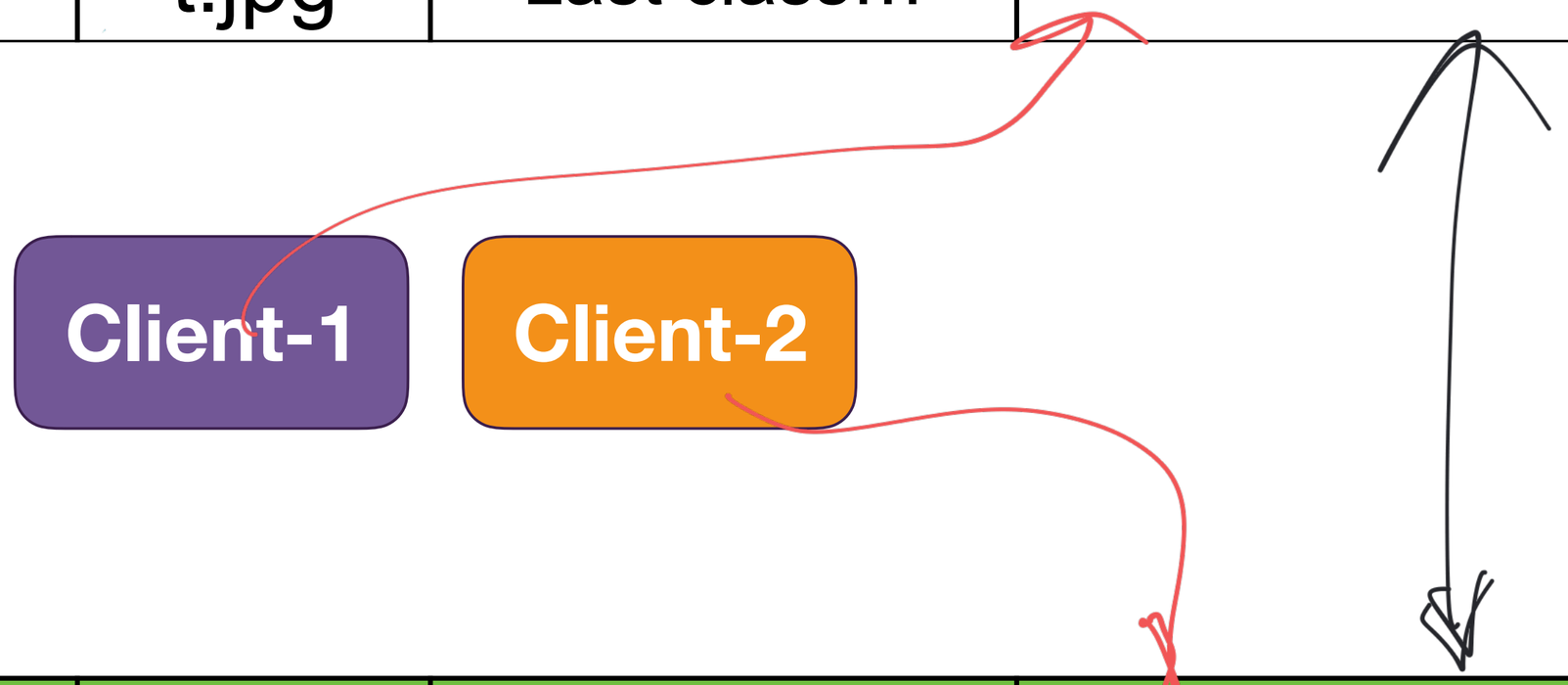


FaceTwit 2.0!

Concurrent updates to replicated state

node-1

ID	Name	Face	Post	Comment
1	Tim	t.jpg	“Last class…”	



node-2

ID	Name	Face	Post	Comment
1	Tim	t.jpg	“Last class…”	

Consistency Models

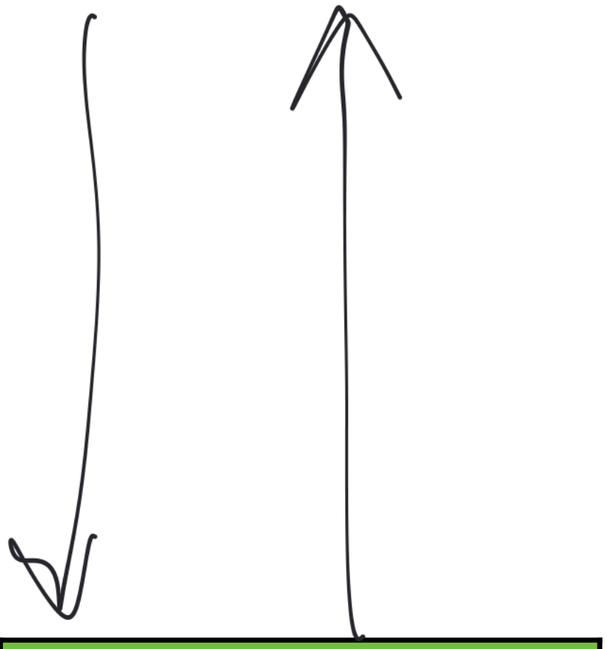
What does it mean to be consistent?

Eventual Consistency

If there are no new updates, **eventually** all replicas will have the most recent value

node-1

ID	Name	Face	Post	Comment
1	Tim	t.jpg	“Last class…”	



Client-1 **Client-2**

node-2

ID	Name	Face	Post	Comment
1	Tim	t.jpg	“Last class…”	

Eventual Consistency

- Nodes exchange updates with a sequence number
- Eventually clients will see latest value

Problem

How to sync seq number?

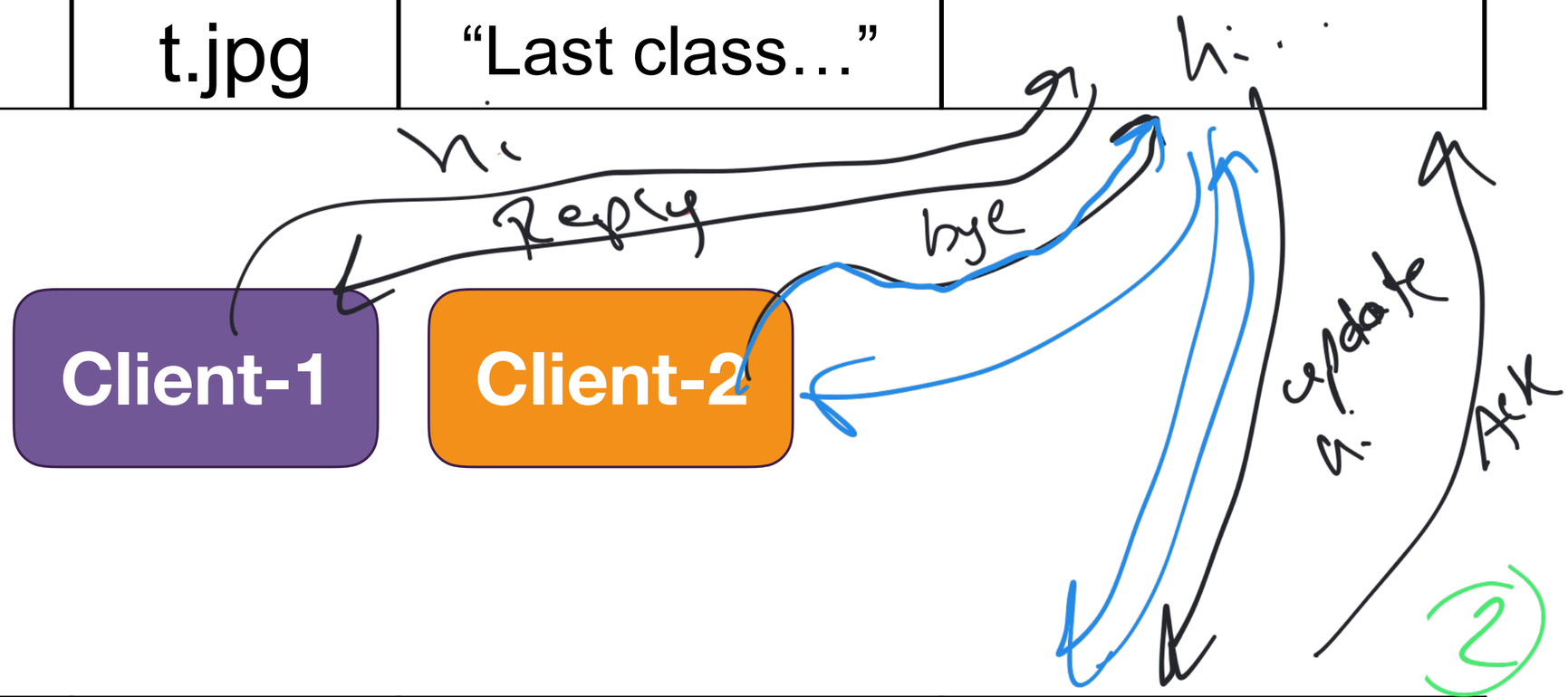
Positive? Efficient way to keep nodes synchronized

Strong Consistency

After each update, all replicas will **immediately** have the most recent value

node-1

ID	Name	Face	Post	Comment
1	Tim	t.jpg	“Last class…”	



node-2

ID	Name	Face	Post	Comment
1	Tim	t.jpg	“Last class…”	hi

Strong Consistency

- Guarantee that if a client gets a reply, all other replicas have the updated value
  - Primary will order incoming write requests
  - Reads can be sent to any replica
    - Higher latency for writes compared to consistency

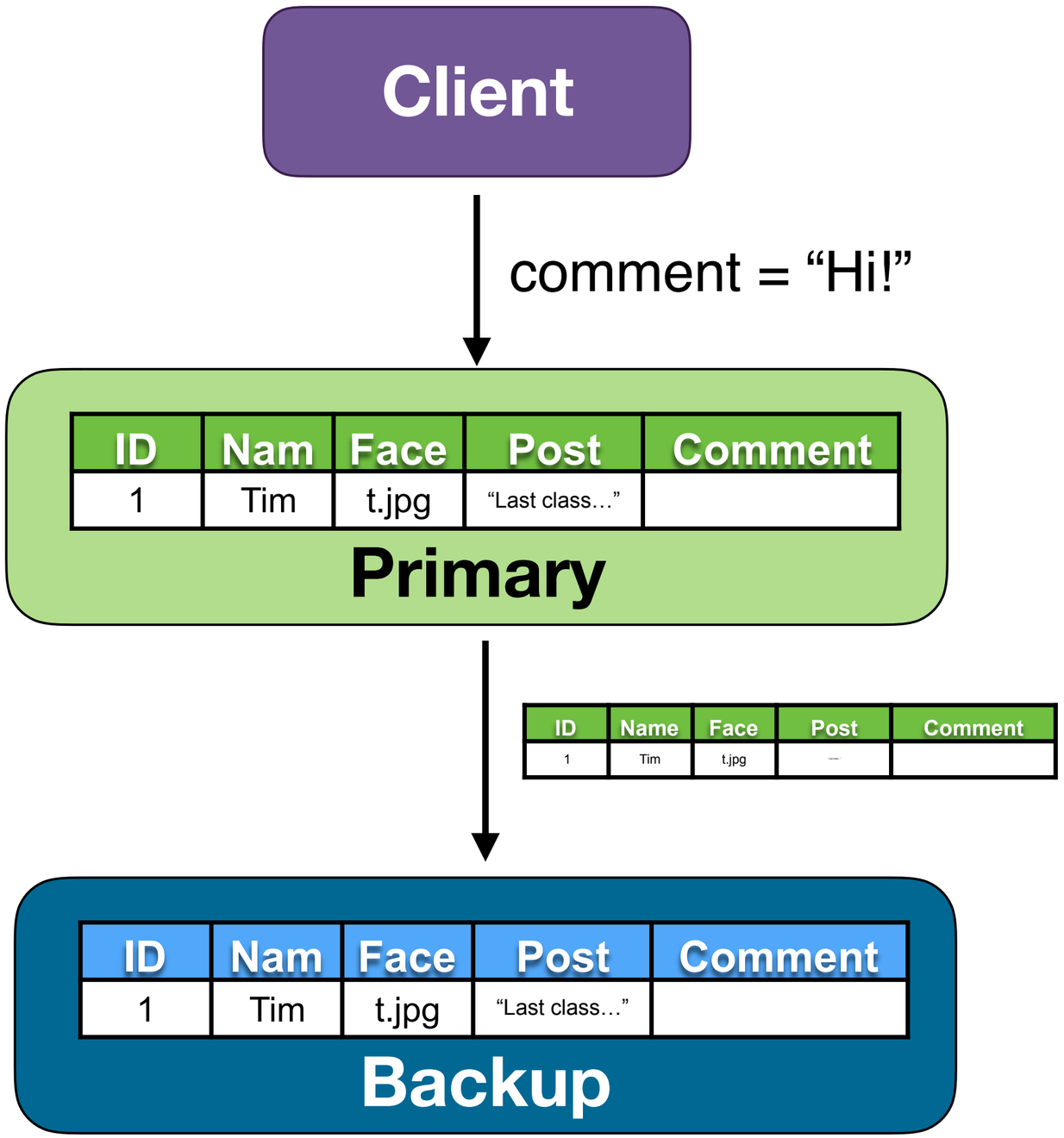
Implementing Strong Consistency

With State Machine Replication

Primary-Backup Replication

Client contacts Primary

Primary replicates to backup(s)



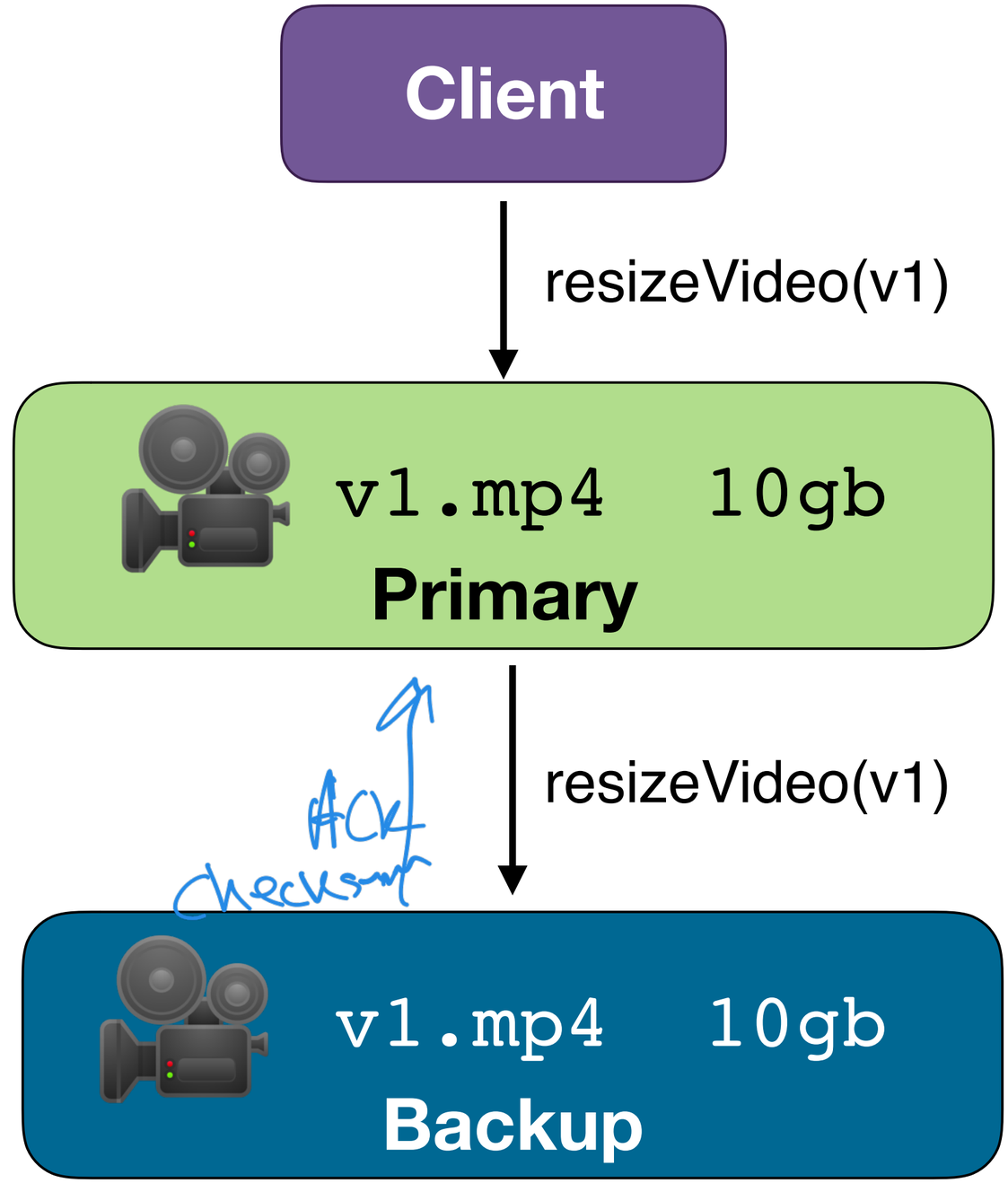
Where do reads go?

What to replicate?

State Machine Replication

Sometimes data is big!

Replicate the **operation** to be performed, not the data!



Treat like a state machine

- Incoming requests just perform some operation on that data
- If all replicas perform same operations, state is consistent

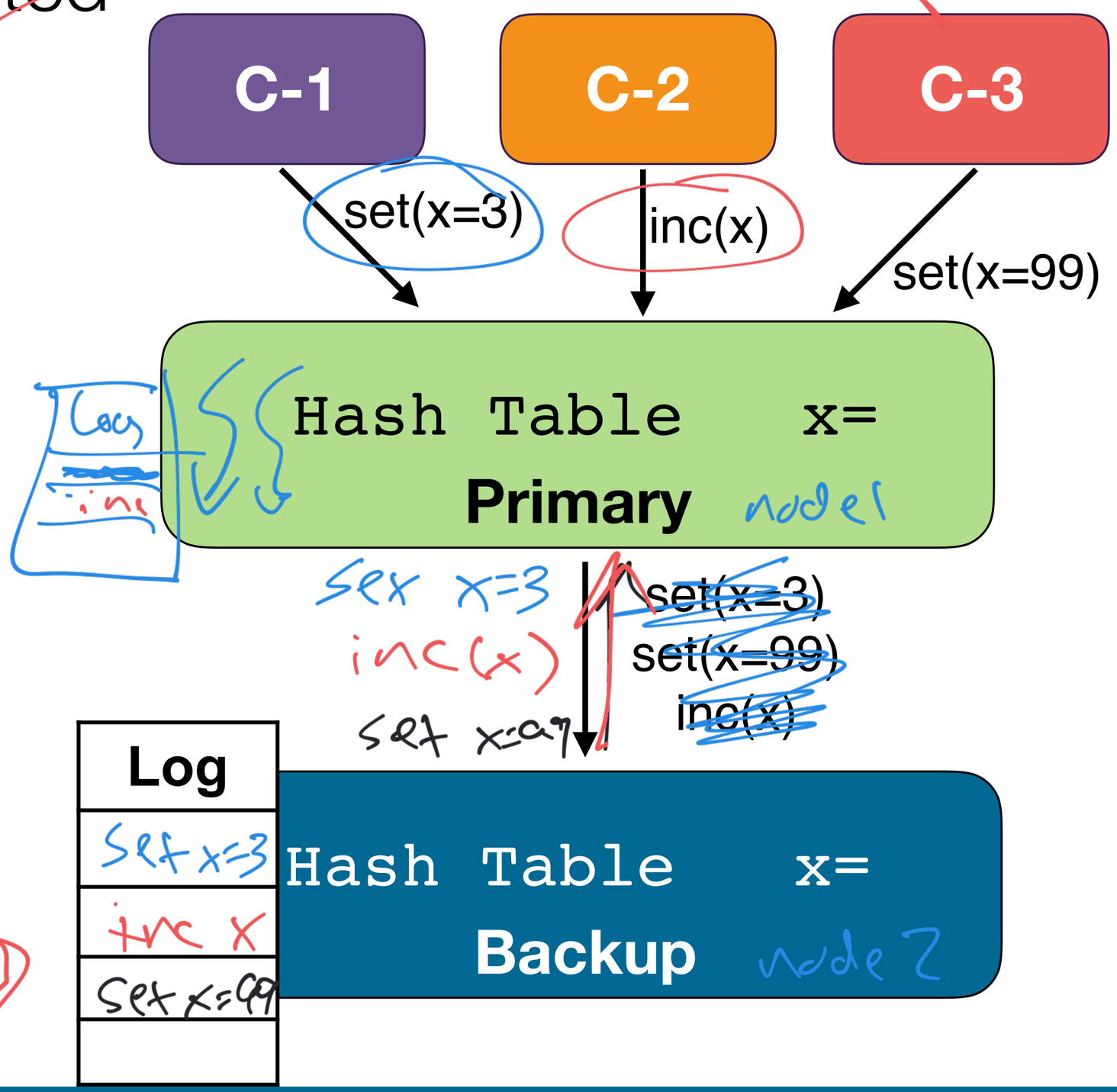
State Machine Replication

SMR creates a replicated log of actions to be performed

Primary orders log

Actions must be deterministic

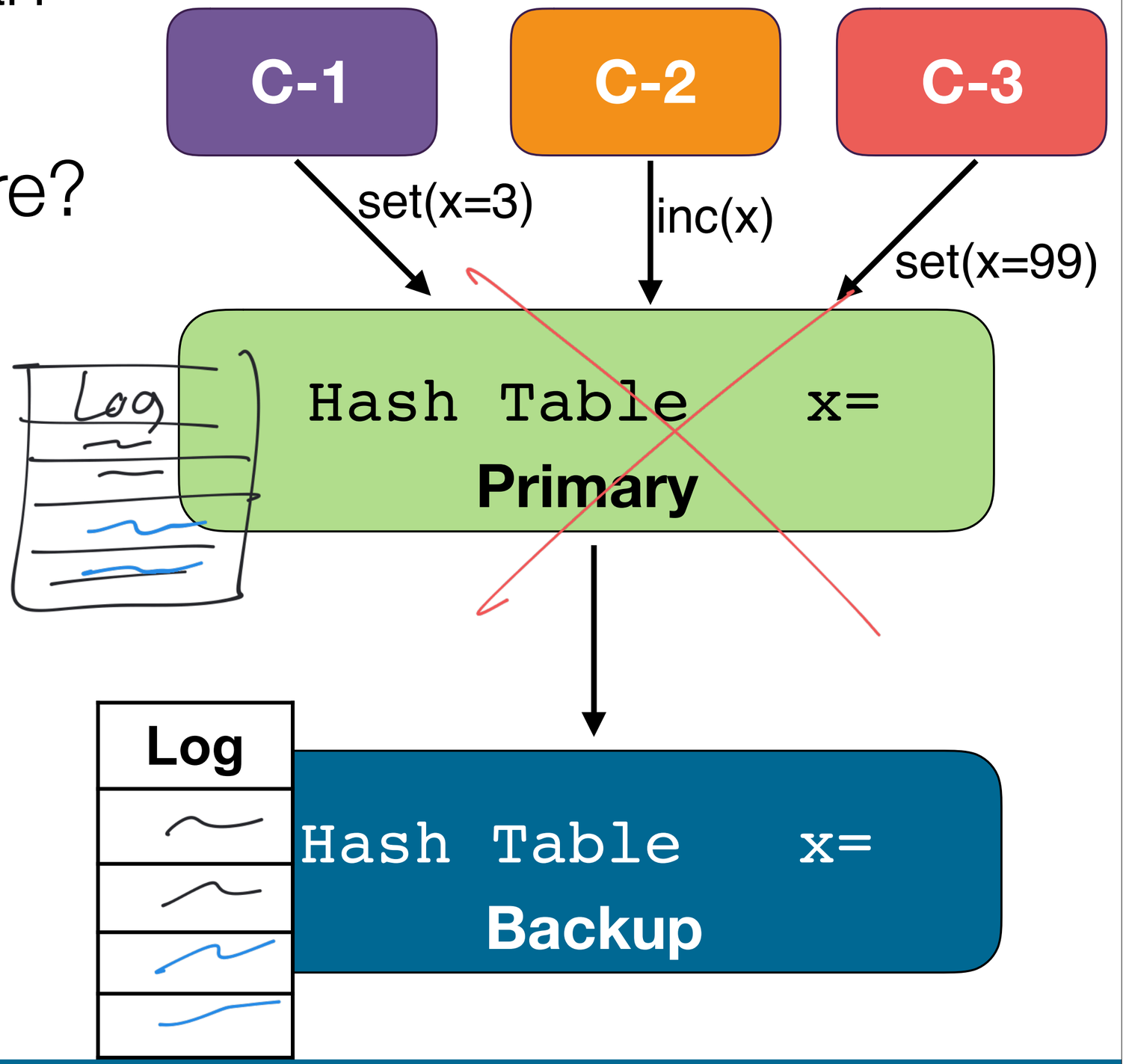
We can keep adding more backup replicas



SMR Failures

How many failures can we handle?

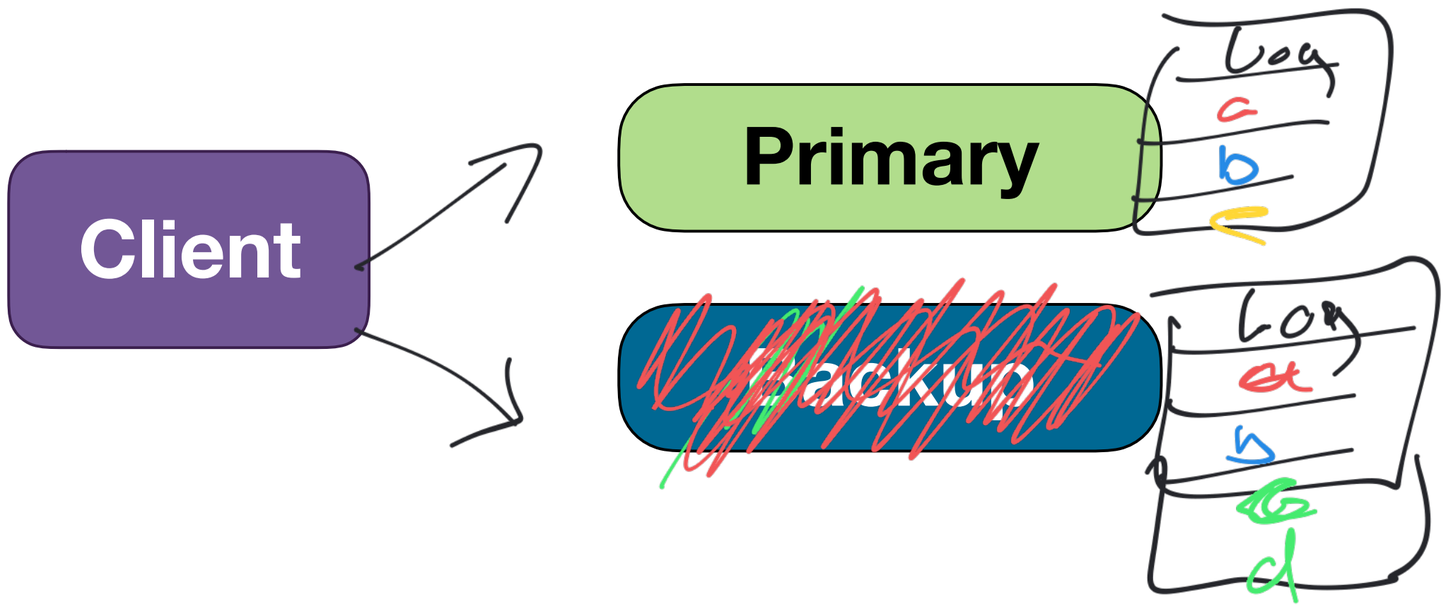
What to do on a failure?



Handling Failures

max # of failures at one time

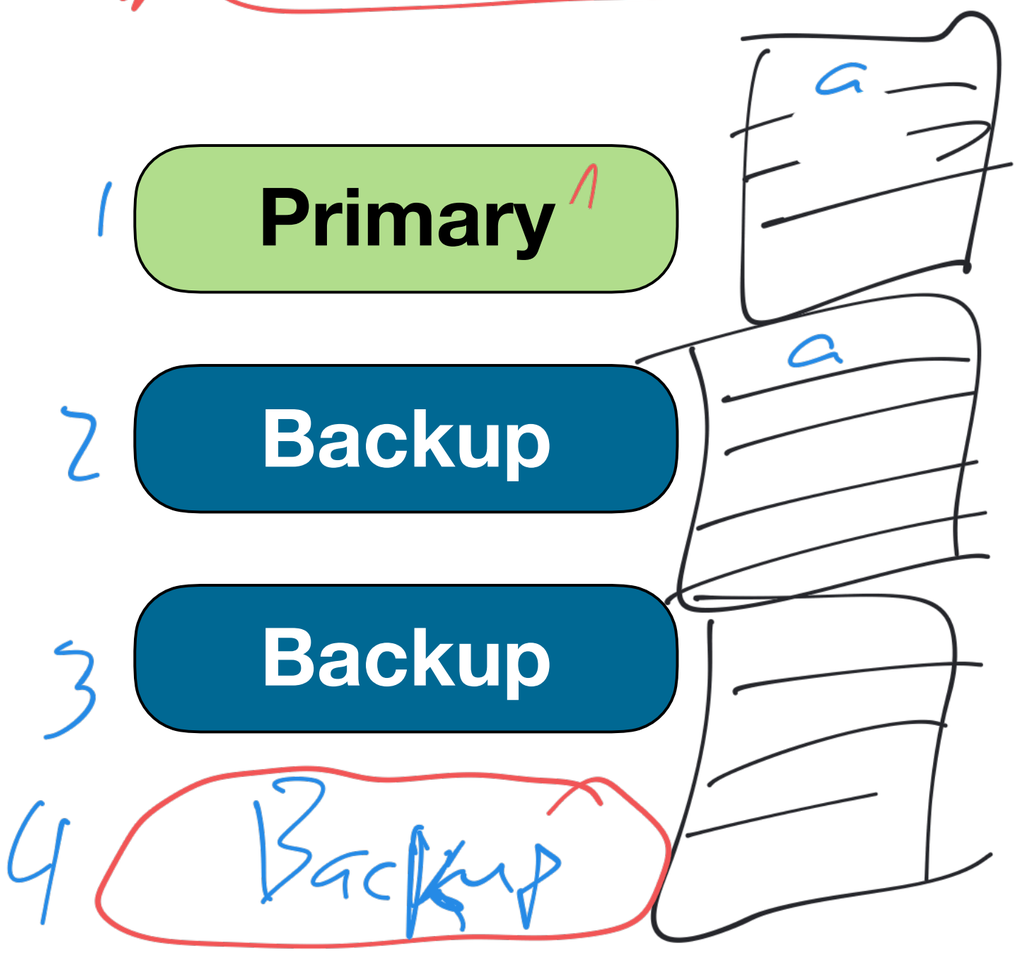
f=1, f+1 replicas



We can't handle flip-flop failure

Handling Failures

f=1, f+2 replicas

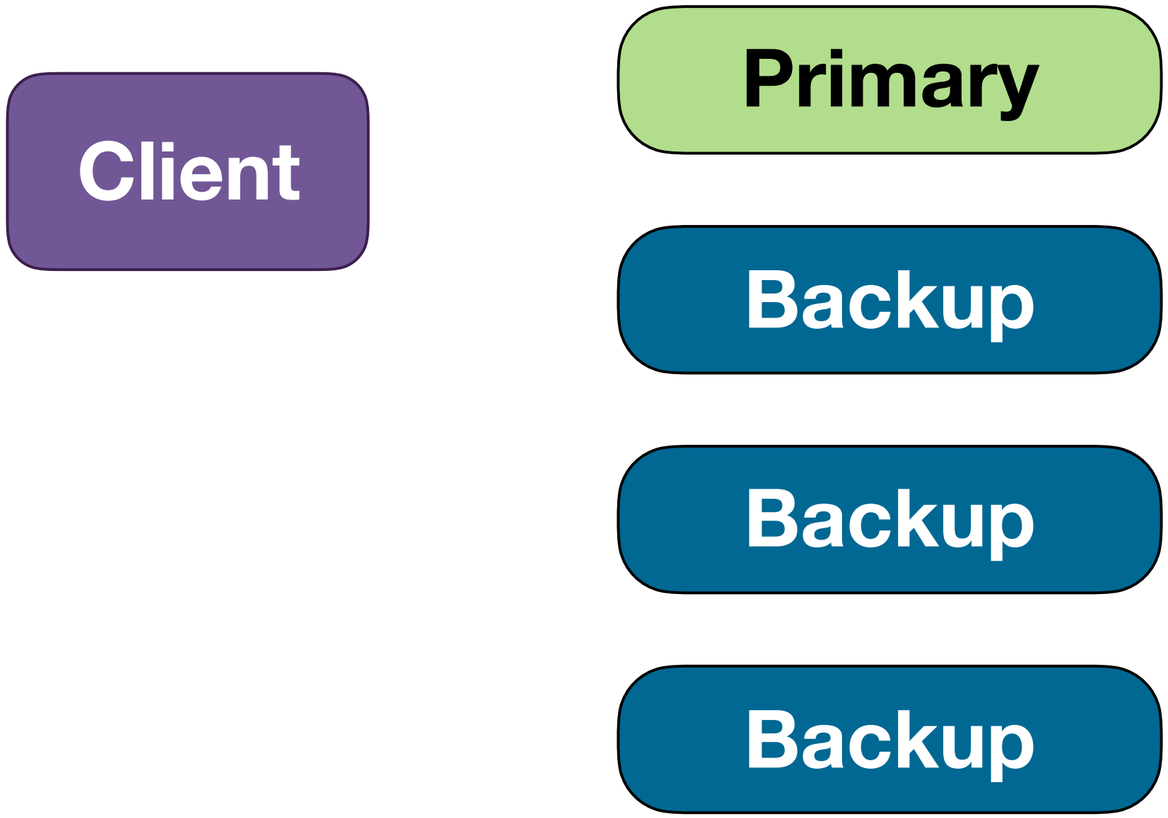


extra replica solves flip flop problem?

No!

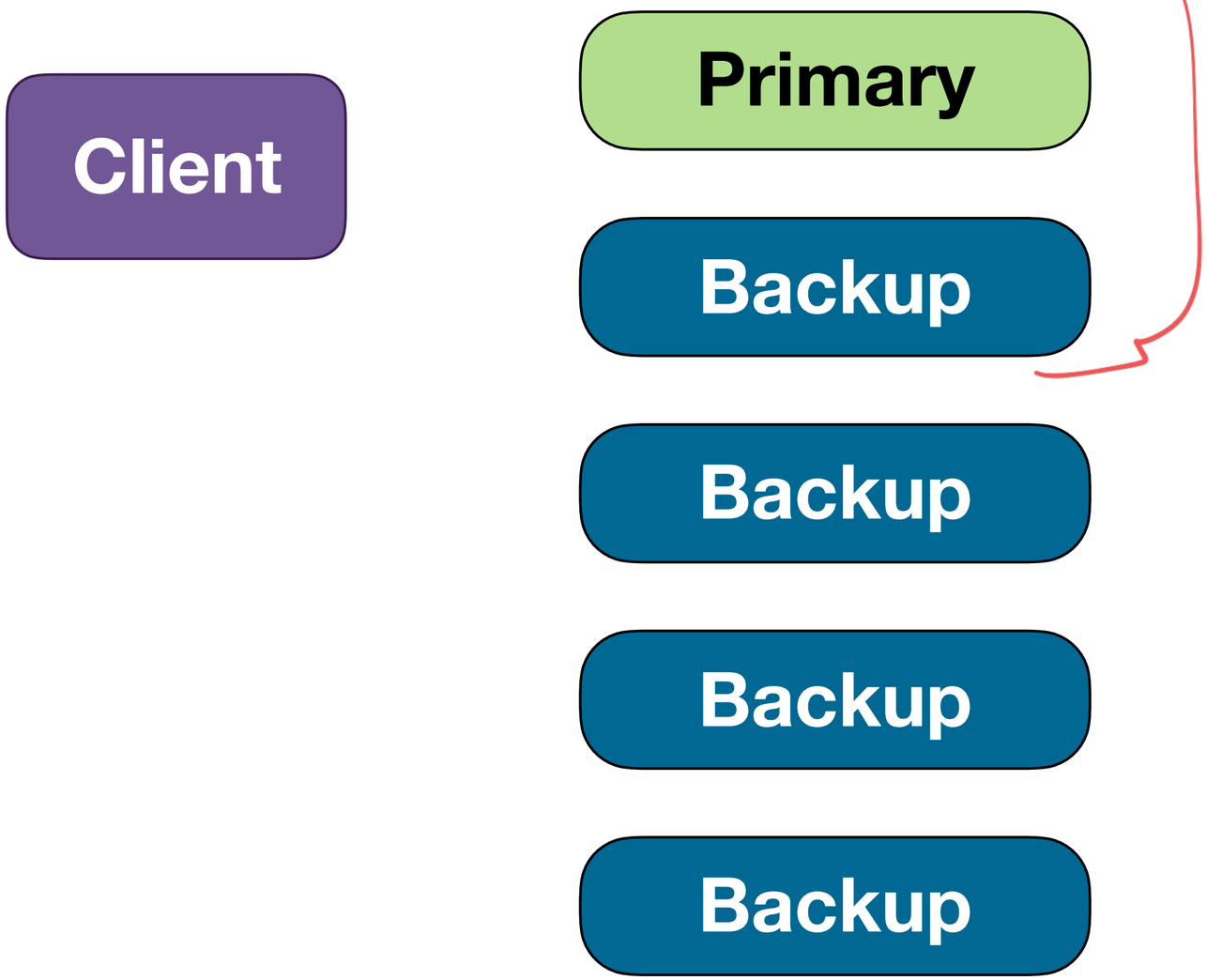
Handling Failures

f=2, f+2 replicas



Handling Failures

f=2, 2f+1 replicas



State Machine Replication

Provides a generic **fault tolerance** mechanism

- Application just needs to have well defined operations and a way to avoid non-determinism

Primary orders requests into log

Backups execute log in order

Log allows out of date replicas to recover

Need **2f+1** replicas to tolerate **f** failures

But how do we pick who should be primary…?

- Tune in next time!