Primary/Backup

Doug Woos

Logistics notes

Lab 2 posted

HW1 up Friday

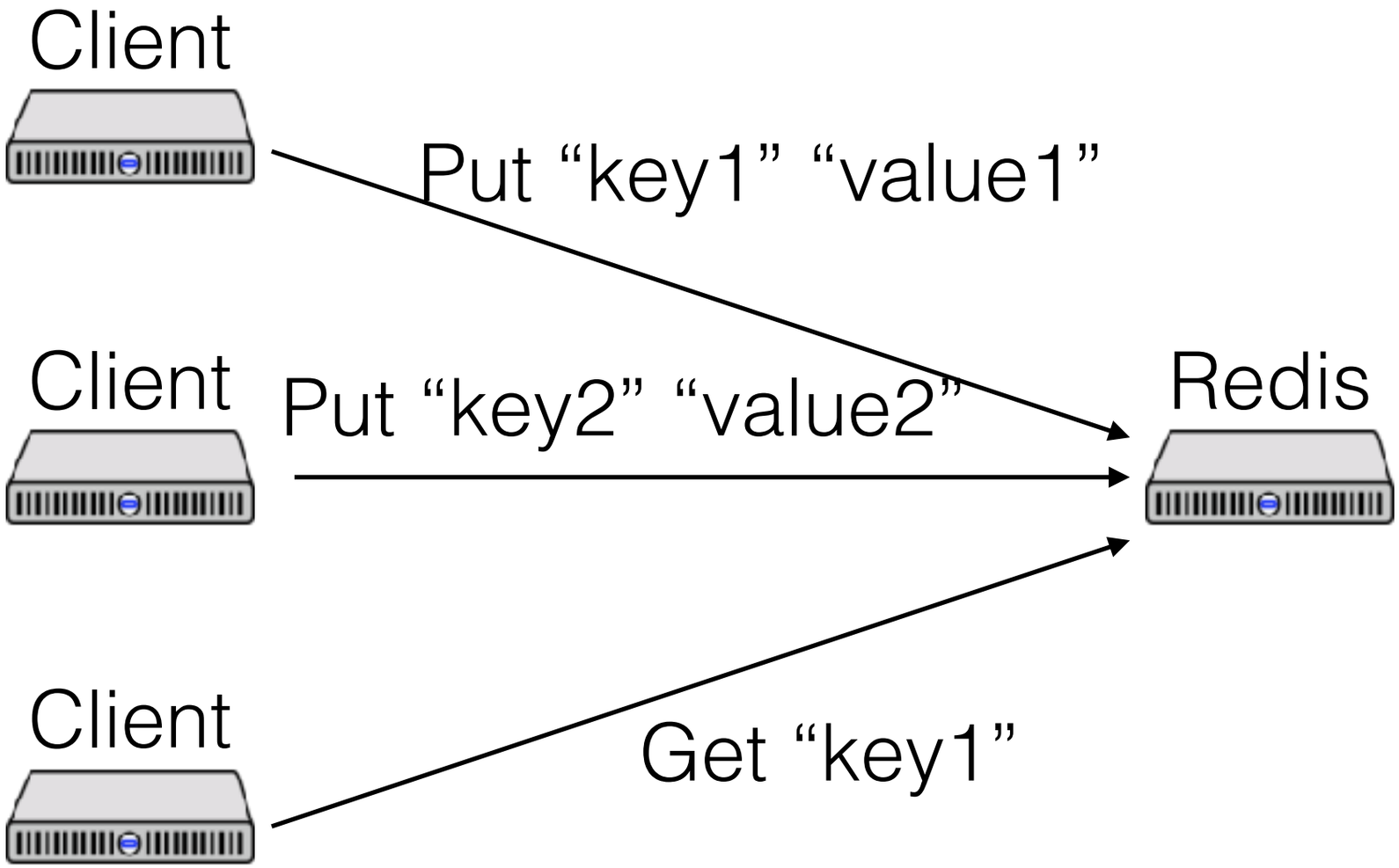
Next week’s papers posted

Today

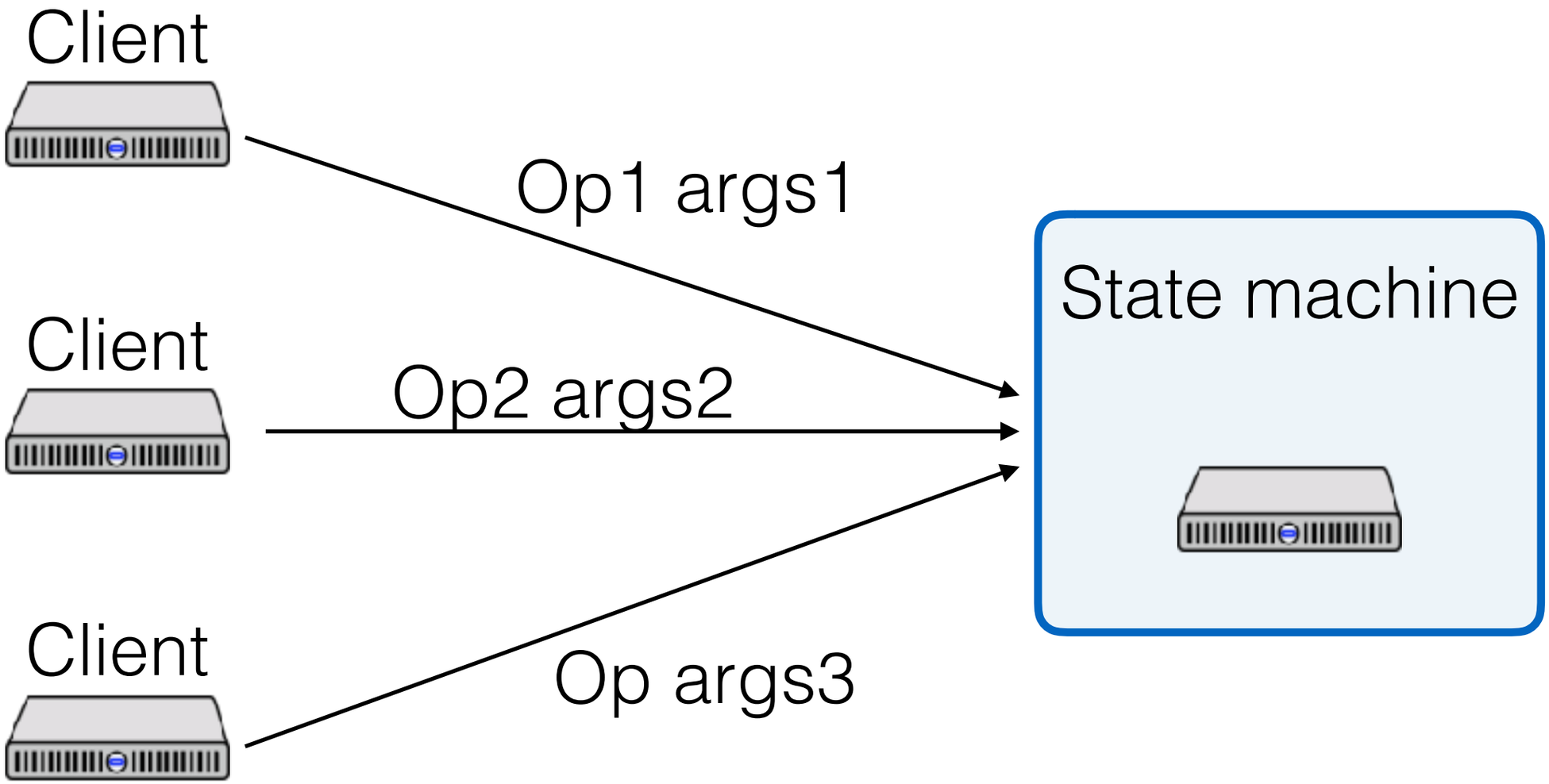
State machine replication

Primary/Backup

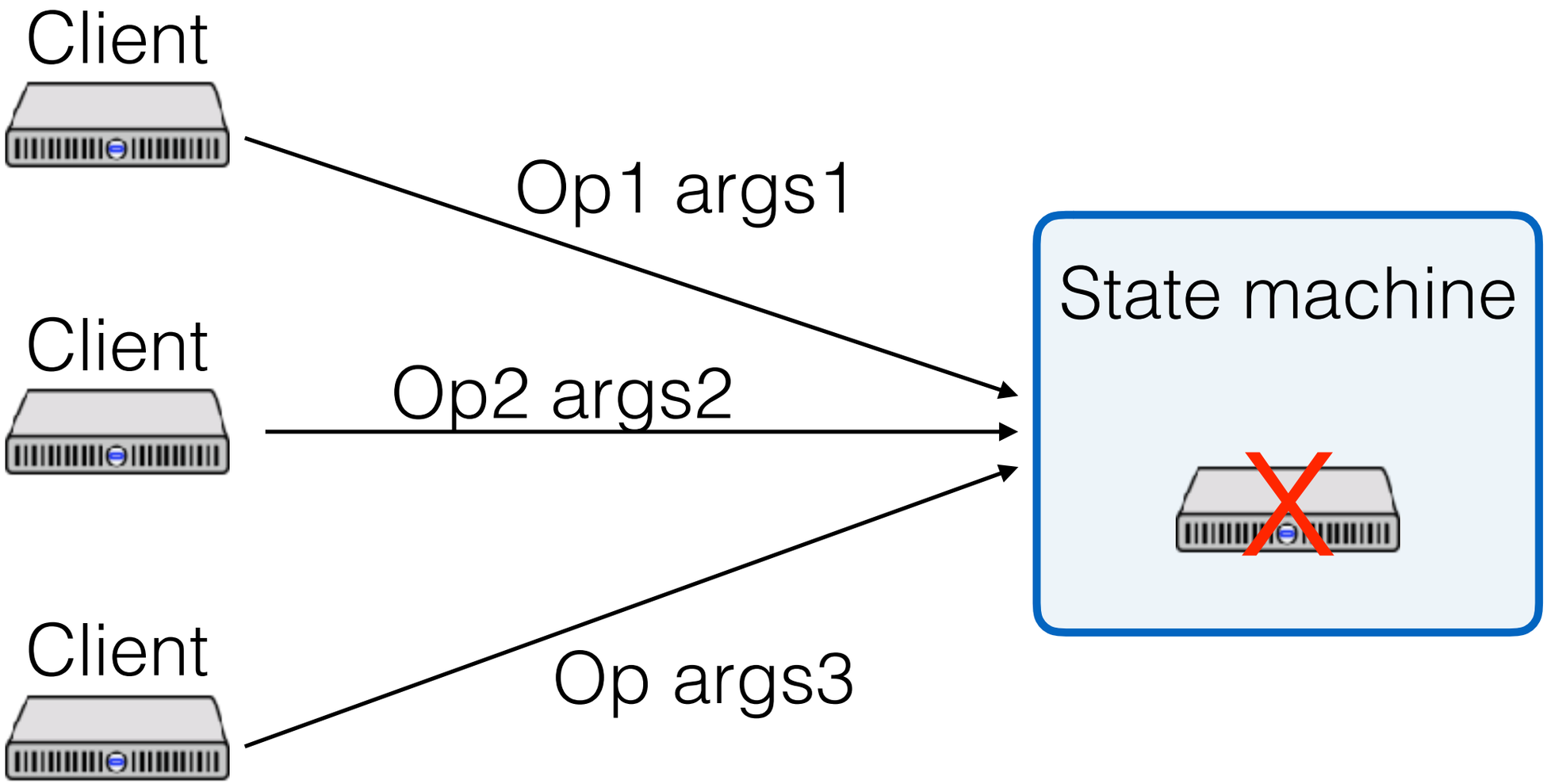
Single-node key/value store



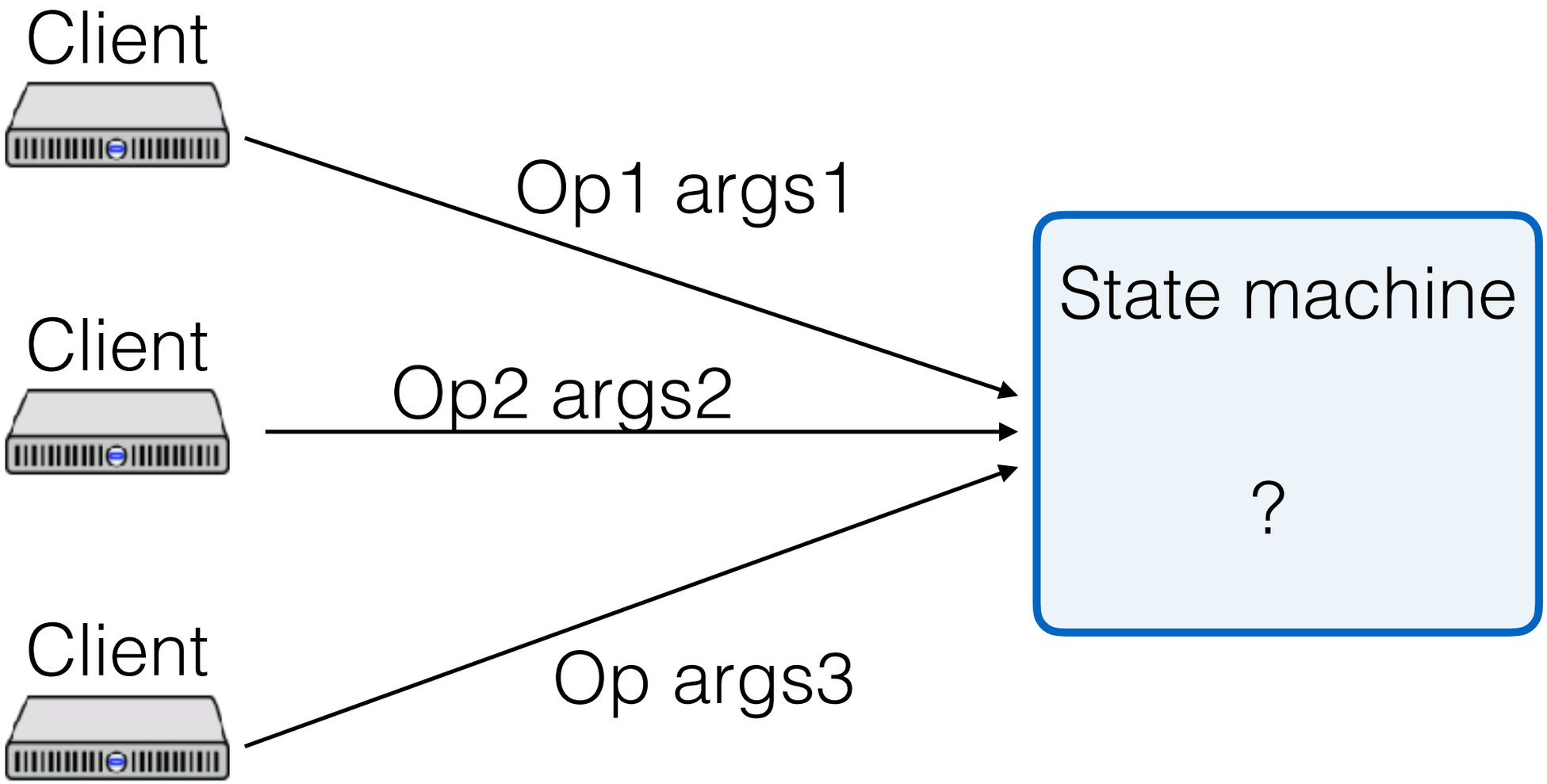
Single-node state machine



Single-node state machine



Single-node state machine



State machine replication

Replicate the state machine across multiple servers

Clients can view all servers as one state machine

What's the simplest form of replication?

Two servers!

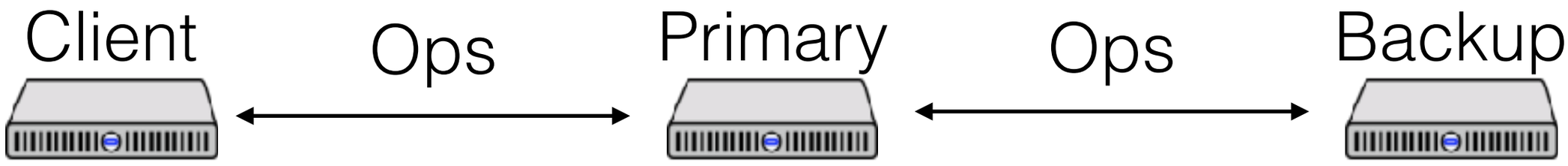
At a given time:

- Clients talk to one server, the primary
- Data are replicated on primary and backup
- If the primary fails, the backup becomes primary

Goals:

- Correct and available
- Despite *some* failures

Basic operation



Clients send operations (Put, Get) to primary

Primary decides on order of ops

Primary forwards sequence of ops to backup

Backup performs ops in same order (hot standby)

- Or just saves the log of operations (cold standby)

After backup has saved ops, primary replies to client

Challenges

Non-deterministic operations

Dropped messages

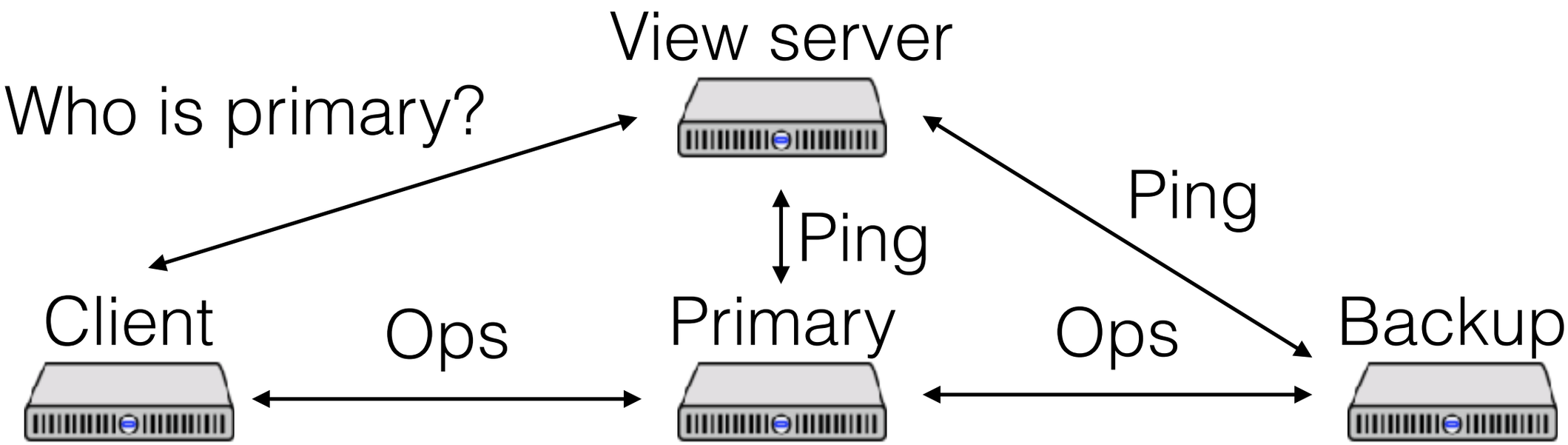
State transfer between primary and backup

- Write log? Write state?

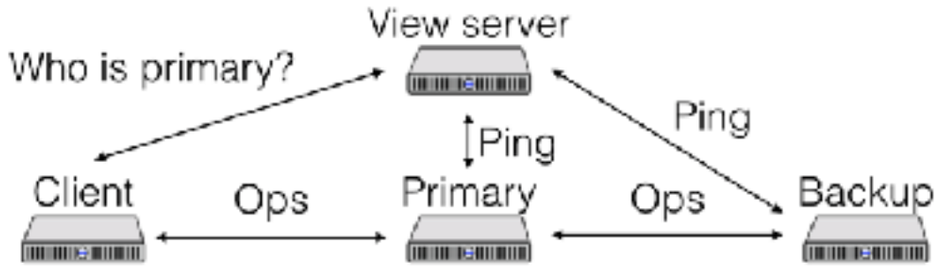
There can be only one primary at a time

- Clients, primary and backup need to agree

The View Service



The View service



View server decides who is primary and backup

- Clients and servers depend on view server

The hard part:

- Must be only one primary at a time

- Clients shouldn’t communicate with view server on every request

- Careful protocol design

View server is a single point of failure (fixed in Lab 3)

On failure

Primary fails

View server declares a new “view”, moves backup to primary

View server promotes an idle server as new backup

Primary initializes new backup’s state

Now ready to process ops, OK if primary fails

“Views”

Comes from Viewstamped Replication (I think?)

A view is a version of the current roles in the system

Logically, time is a sequence of views

View 1	View 2	View 3
Primary = A	Primary = B	Primary = C
Backup = B	Backup = C	Backup = A

Detecting failure

Each server periodically pings (Ping RPC) view server

- “dead” if missed *n* Pings
- “live” after a single Ping

Can a server ever be up but declared dead?

Managing servers

Any number of servers can send Pings

- If more than two servers, extras are “idle”
- Can be promoted to backup

If primary dies

- New view with old backup as primary

If backup is dead, or no backup

- New view with idle server as backup

OK to have a view with a primary and no backup

- Why?

Question

How to ensure new primary has up-to-date state?

- Only promote the backup -> primary
- Idle server can become primary at startup (why?)

What if the backup hasn’t gotten the state yet?

- Remember, first thing = transfer state to backup

View 1
Primary = A
Backup = B

A stops pinging

View 2
Primary = B
Backup = C

B *immediately* stops pinging

View 3
Primary = C
Backup = _

Can't move to View 3 until C gets state
How does view server know C has state?

Primary acks

Track whether primary has acked (with ping) current view

MUST stay with current view until ack

Even if primary seems to have failed

This is another weakness of this protocol

Question

Can more than one server think it’s primary?

Split brain

1: A, B

A is still up, but can’t reach view server

2: B, _

B learns it is promoted to primary
A still thinks it is primary

Split brain

Can more than one server *act* as primary?

- Act as = respond to clients

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer