Week 3
Spyros Mastorakis

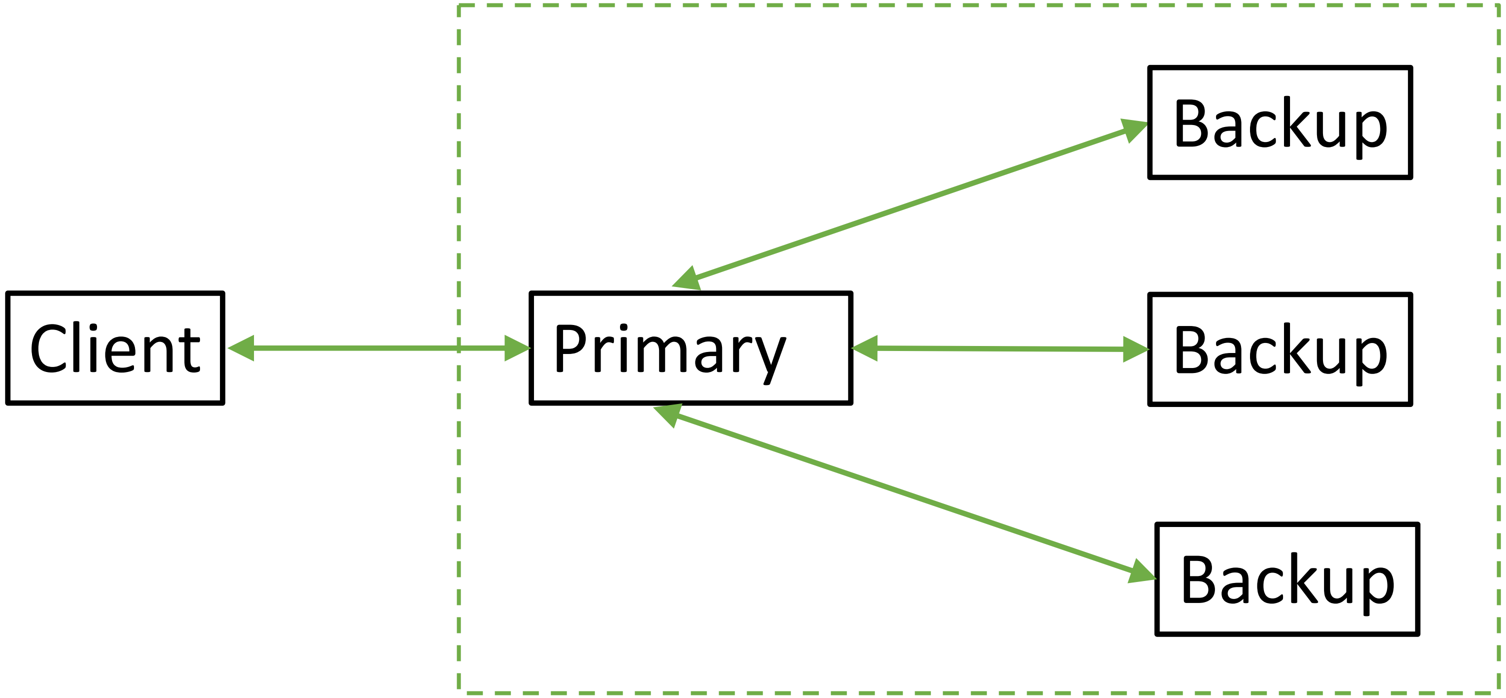
Outline

- Primary backup replication
- Assignment 2

What to do with fail-stop failures?

- Last time: crash failures
  - Replicas can resume execution with saved state
- Today: fail-stop failures
  - All state is lost upon failure
  - Need to replicate state proactively
- Approach: primary backup replication
- Challenges
  - What if primary or backup fails?
  - How can we keep them in sync?

Primary Backup Replication



Key Idea: transparently have two replicas, primary and backup

- Primary interacts with client
- Backup stores copy of primary’s state

Handling failures

- How to handle primary failure?
  - Promote one of the backups to be the primary

- How to handle backup failure?
  - Add another machine as a backup

When to sync?

- It is okay for the primary to be out of sync until a change is externally visible
  - External consistency: primary backup in sync to external world
- Implications for when primary should sync with backups?
  - Sync must happen before any state change is externally visible

What to transfer during sync

- Heavyweight: send snapshot of primary's state
    - **Slow!**
    - When is this necessary?
        - When bootstrapping a new backup

- Lightweight: send every operation
    - Why is this okay?
        - Leverage determinism of state machine
        - Send any state necessary for backup to mimic execution

Transparent Primary Backup

- Application relies on library to keep primary and backups in sync

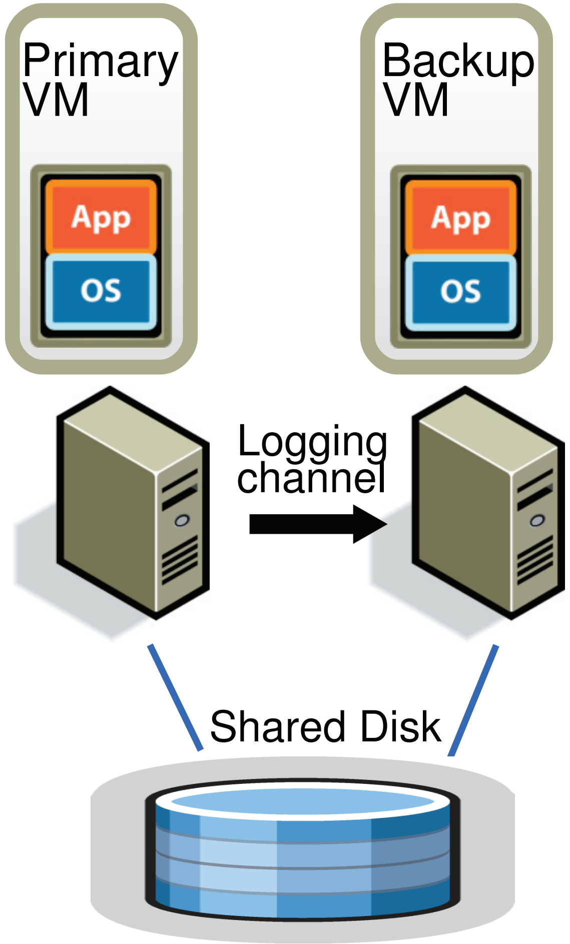
- Library functions
  - Receive message from client
  - Sync with backups before sending response to client

- Will this solution work?
  - No! Does not capture nondeterminism in execution

VMM—based Primary Backup

- Primary and backup execute on two VMs
  - Primary logs inputs and outputs to log
  - Backup applies inputs from log
  - Primary waits for backup output

- Primary-backup monitor each other
  - If primary fails, backup takes over



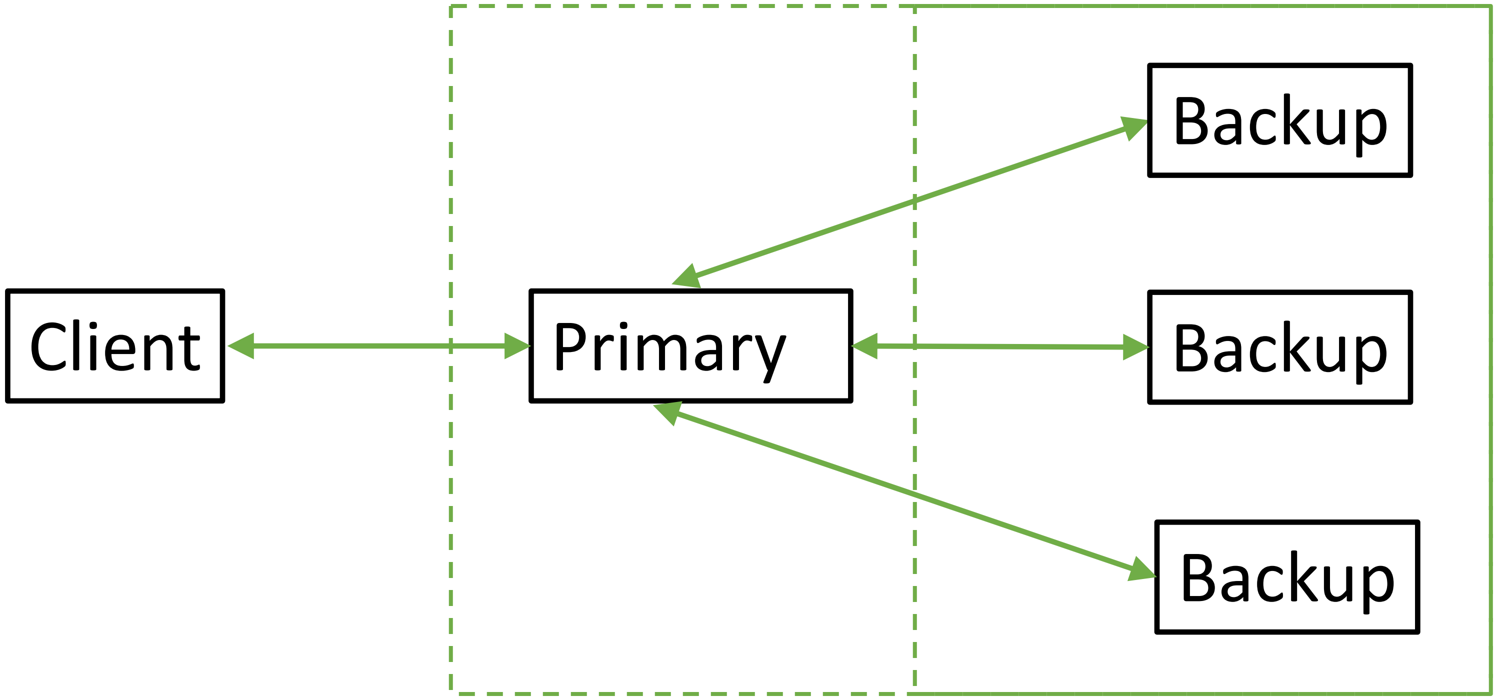
Log-based VM replication

- Primary VMM sends log entries to backup VMM over the logging channel

- Backup VMM (hypervisor) replays log entries
  - Stops backup VM at next input event or nondeterministic instruction
  - Delivers same input event as primary used
  - Delivers same nondeterministic value as primary

- Ensures backup and primary state never diverge

Primary Backup Replication



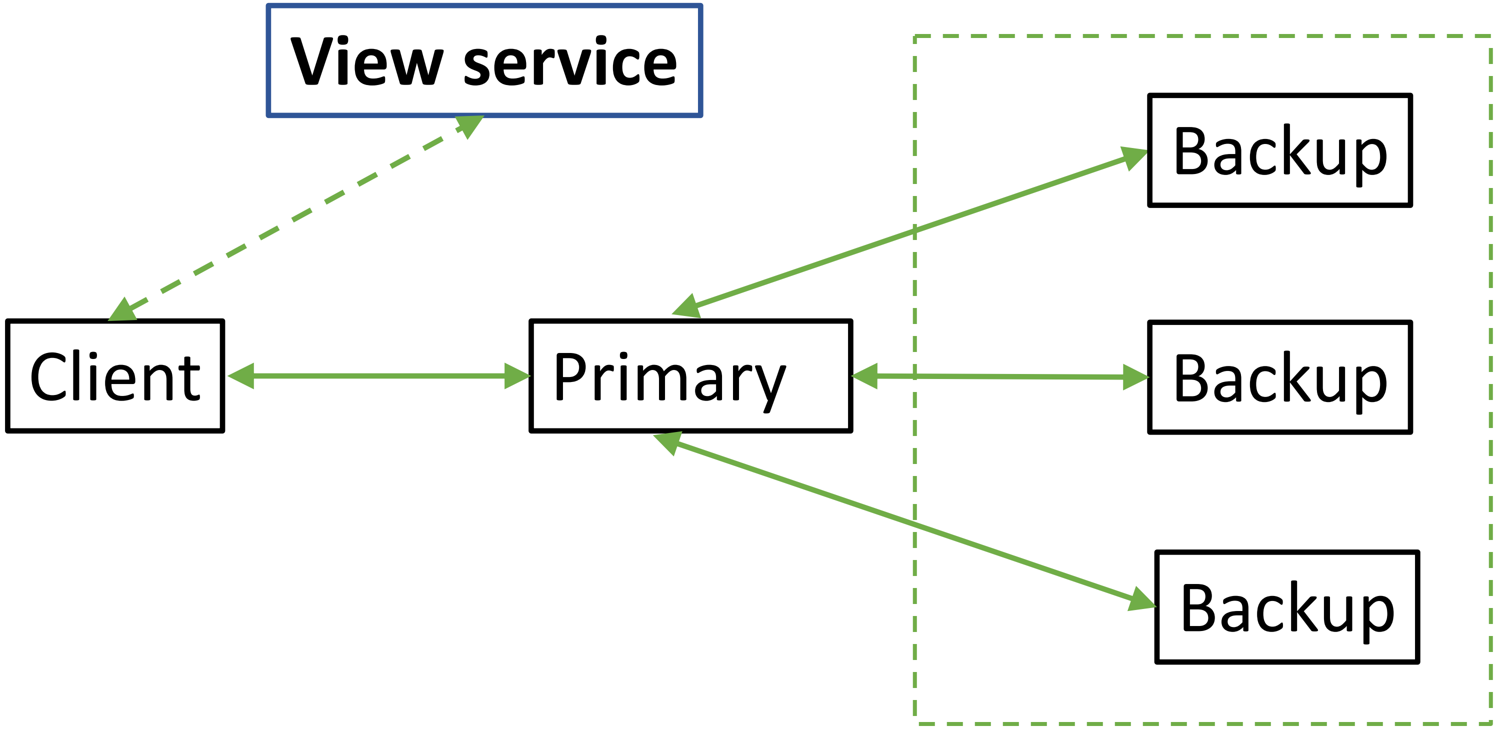
Client perspective

- What does a worker need to know in order to register itself with the replicated master?
  - Needs to know which machine is primary

- Can the primary be hardcoded into client app code?
  - No! Primary will get replaced when it fails

- How does the client discover the current primary?
  - Needs reliable service to do primary lookups

Primary Backup Replication



View Service

- Maintains current membership of primary-backup service (i.e., **view**)
  - Each view → (view number, primary, backup)

- When does a view service change the view?
  - When primary or any backup fails
  - Periodically exchange heartbeat messages to detect failures

- What if view service is down or not reachable?

Transitioning between views

- View service broadcasts view change to all replicas

- Primary must ACK new view once backup is up to date

- Two implications
  - Liveness detection timeout > state transfer time
  - Cannot change view if primary fails during sync

View Service

- Summary: view change has three steps
  - View service announces new view to all replicas
  - Primary syncs with new backup if there is one
  - Primary acknowledges new view

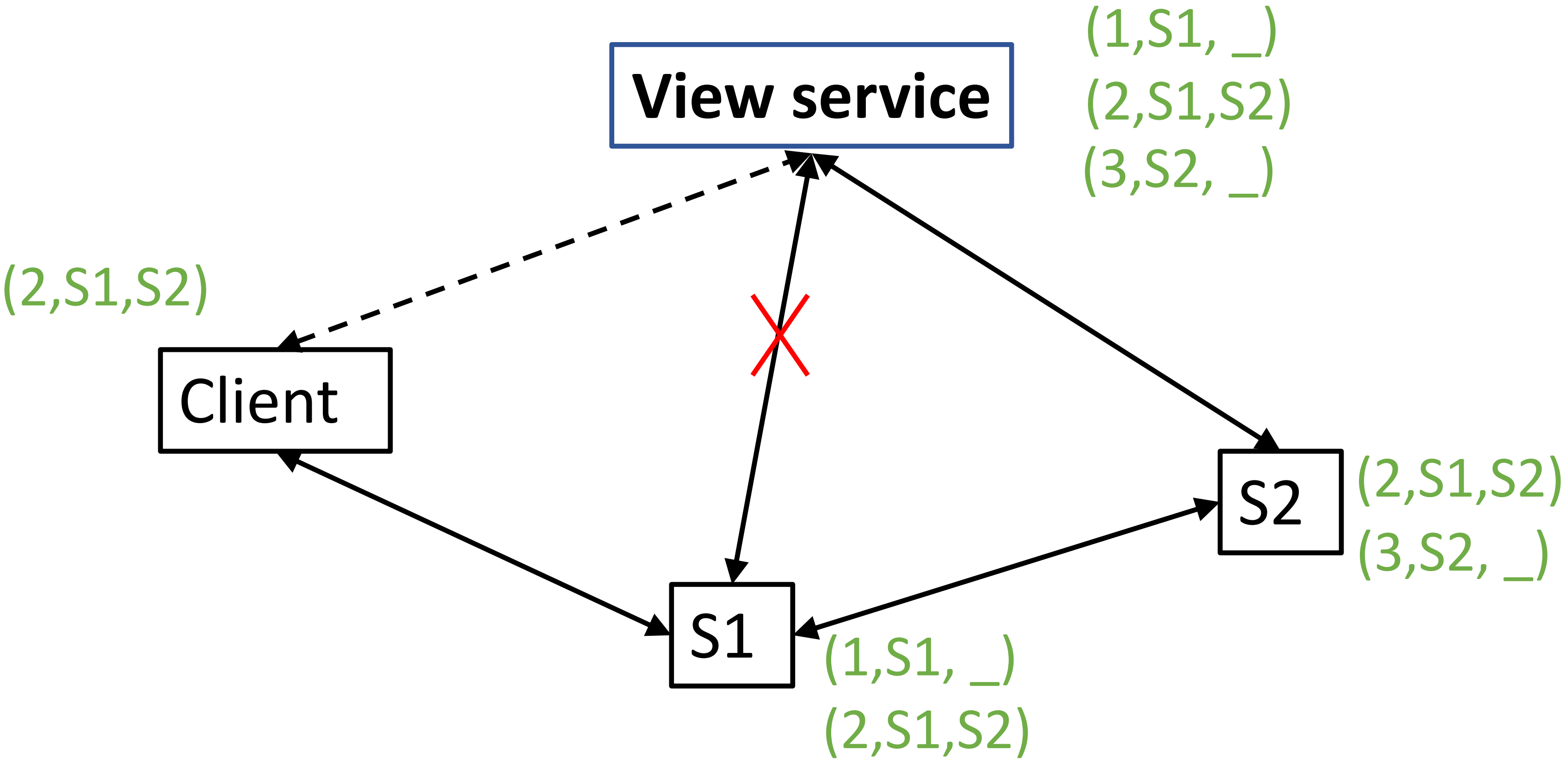
- Stuck if primary fails in the midst of this process

Scalability of View Service

- Does every client need to contact view service before any operation?
  - No--clients can cache view across operations

- When to invalidate cached view?
  - Client invalidates cache when no response or negative response from replica it thinks is primary

Split Brain scenario



Avoiding Split Brain

- Primary must forward all operations to backups
  - Goal: get ACKs from backups that they too recognize primary

- Why can’t backups be mistaken about who is primary?
  - Only a backup can be promoted as a primary

View Service Sequence Examples

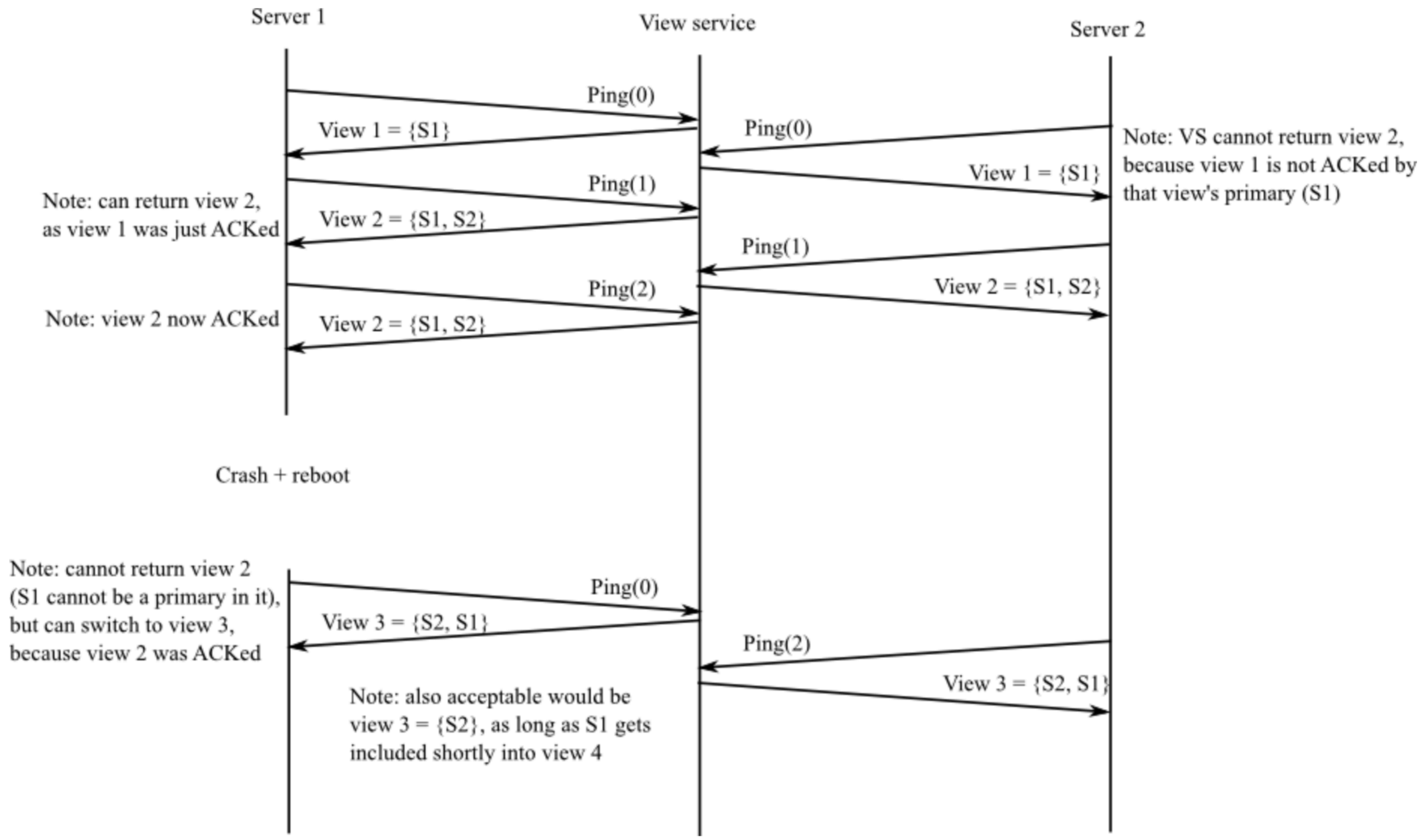
- Valid sequence of views
  - (1, S1, _) → (2, S1, S2) → (3, S1, S3) → (4, S3, S4) → (5, S4, _)

- Examples of invalid transitions between views:
  - (1, S1, S2) → (2, S3, S4)
  - (1, S1, S2) → (2, _, S2)
  - (1, S1, _) → (2, S2, S1)
  - (2, S1, _) → (1, S1, S2)

Assignment 2

- Primary backup replication for a key/value service
- Coordination on who is the primary and who is a backup server through viewservice
- Viewservice monitors whether each server is alive or dead

Assignment 2 (cont’d)



Server 1

View service

Server 2

Ping(0)

View 1 = {S1}

Ping(0)

Note: VS cannot return view 2, because view 1 is not ACKed by that view's primary (S1)

View 1 = {S1}

Ping(1)

Note: can return view 2, as view 1 was just ACKed

View 2 = {S1, S2}

Ping(1)

Ping(2)

View 2 = {S1, S2}

Note: view 2 now ACKed

View 2 = {S1, S2}

Crash + reboot

Note: cannot return view 2 (S1 cannot be a primary in it), but can switch to view 3, because view 2 was ACKed

Ping(0)

View 3 = {S2, S1}

Ping(2)

View 3 = {S2, S1}

Note: also acceptable would be view 3 = {S2}, as long as S1 gets included shortly into view 4

Part A: Implementing the viewservice

- Based on previous figure, add field(s) to ViewServer in server.go in order to keep track of the most recent time at which the viewservice has heard a Ping from each server
- Viewservice keeps track of whether the primary for the current view has acknowledged it
- Viewservice needs to make periodic decisions, for example to promote the backup if the viewservice has missed DeadPings pings from the primary
- Study the test cases before you start programming
  - If you fail a test, you may have to look at the test code in test_test.go to figure out the failure scenario is

Part B: Primary backup key/value service

- **Assumption:** the viewservice never halts or crashes
- Implement the client and server parts
- Clients use the service by creating a Clerk object (see client.go) and calling its methods, which send RPCs to the service:
  - If the current primary does not respond, or doesn't think it's the primary, have the client consult the viewservice
- Server side:
  - Pings viewservice to find the current view
  - Primary forwards updates to backup
  - When a server becomes the backup in a new view, the primary should send it the primary's complete key/value database