Section 2: Lab 2 Intro Primary/Backup

CS 5414 Spring 2024

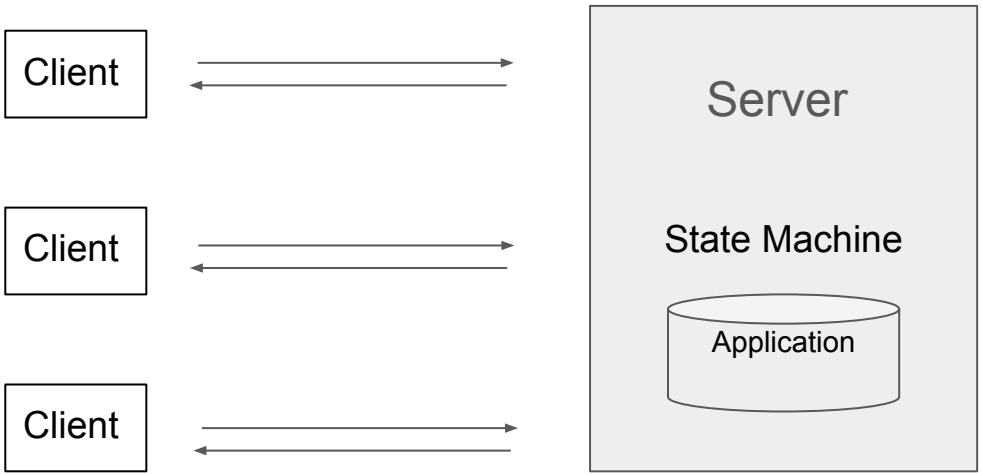
Reference:
https://docs.google.com/presentation/d/1cWEqbuaSnup29gvdvYf5fXlz21OIIzYPyAjOdafsZZU/edit#slide=id.gcee6685f1b_3_4
https://docs.google.com/presentation/d/16e-1KiHBmperZ0VflsBXOPUJXQVkdFWIjRjRfBA4OA4/edit#slide=id.g22bd37c3dde_3_0
https://docs.google.com/presentation/d/1Odupg44Qr-NqOIwdWKz5ASp1tc-AoBQPjfcghnLNDNY/edit#slide=id.g36eceac7fa_3_88

Congratulations on finishing Lab 1 and design doc!!!

Plan for today

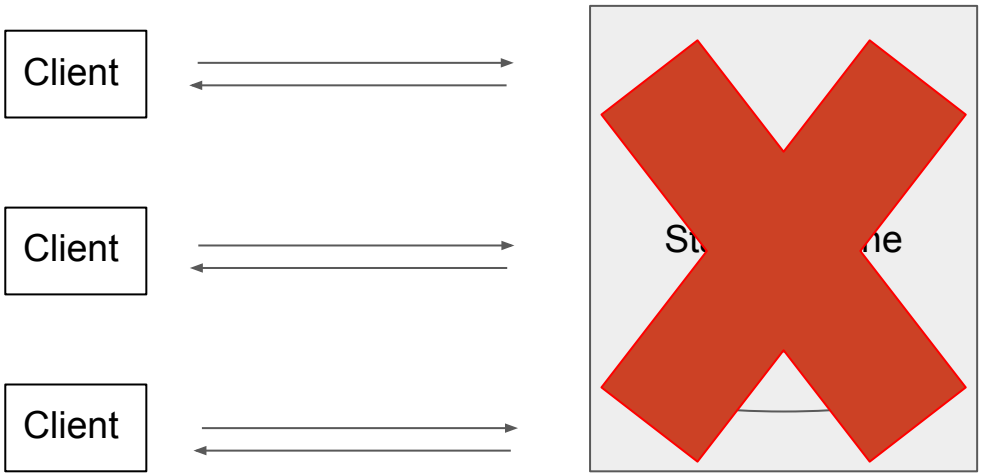
- Lab 2 is out and its design doc is due on next Friday at **noon**
- Lab 2 introduction
  - Part 1: View Server
  - Part 2: Primary/Backup KV Service
- Q&A

Single Node State-Machine



* In this case state machines are NOT Deterministic Finite Automata (DFA) and Nondeterministic Finite Automata (NFA)

Single Node State-Machine



It can fail and we will lose all the states (data).

* In this case state machines are NOT Deterministic Finite Automata (DFA) and Nondeterministic Finite Automata (NFA)

How can we improve?

State machine replication

Coordinate a set of deterministic replicas

each of which can crash

so that,

together

they can implement the abstraction of a

single, correct server

Primary-backup

More about this in class, but…

- One of the replicas operates as the primary, receiving client requests
- Other replicas serve as backups, ready to step in if the primary fails
  - If multiple backups, when primary fails, need to elect a new primary; lab2 has one backup.
- Primary/backup must coordinate so that, if primary fails and a new primary is elected, none of the state of the old primary that a client has observed can be lost
- **Assumption**: In any view, at most one of the primary and backup can fail

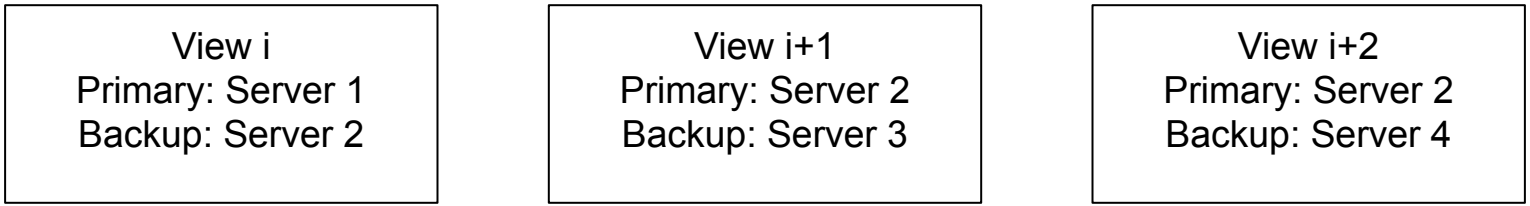
1. Replicate the state machine across multiple servers
2. Clients now view all servers as one state machine
3. But how do clients know whom to talk to?

The View Server

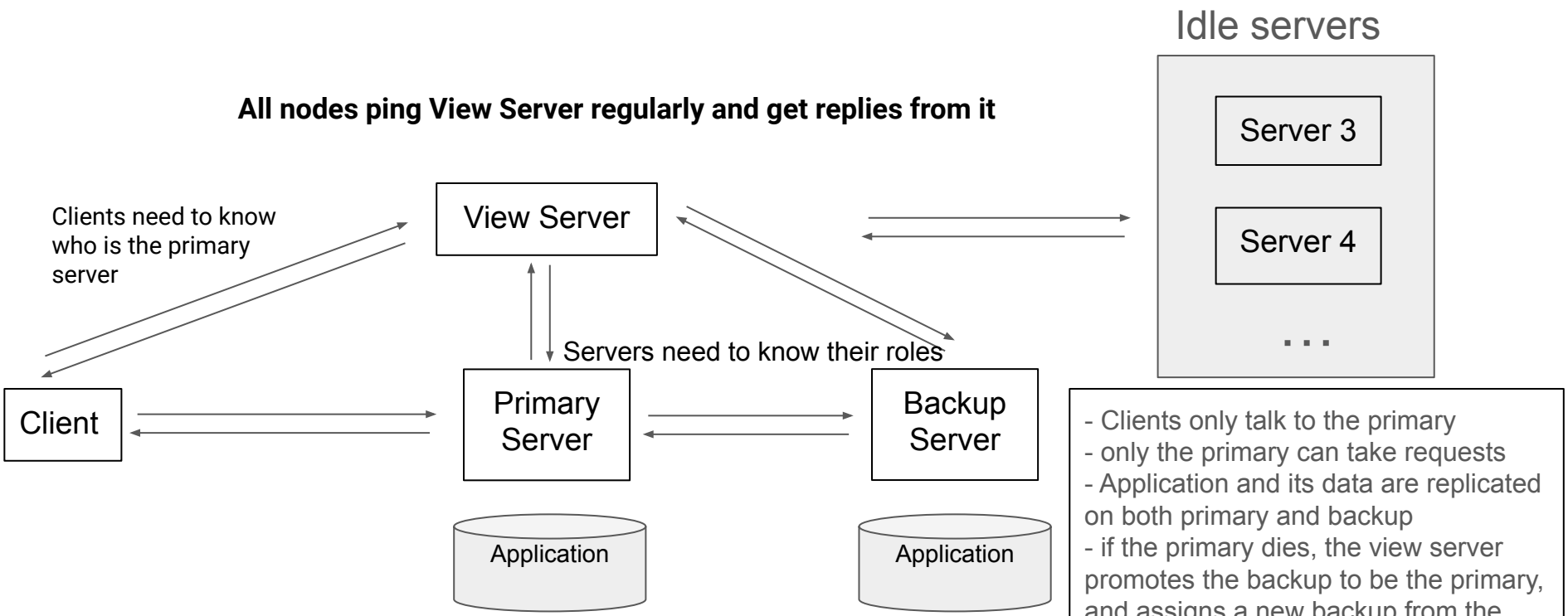
- Monitors all other servers in the system
- If a server goes down, the view server will detect its failure
- When a primary/backup failed, perform some failover to replace that server
- Uses View to let all nodes know who is the primary and who is the backup
- HOWEVER, if the view server crashes, then we can do nothing
    - In the next lab, you will learn a new protocol to tolerate this node failure

Views - a statement about the current roles in the system

- Form a sequence in time



Primary-Backup State-Machine Replication and the View Service



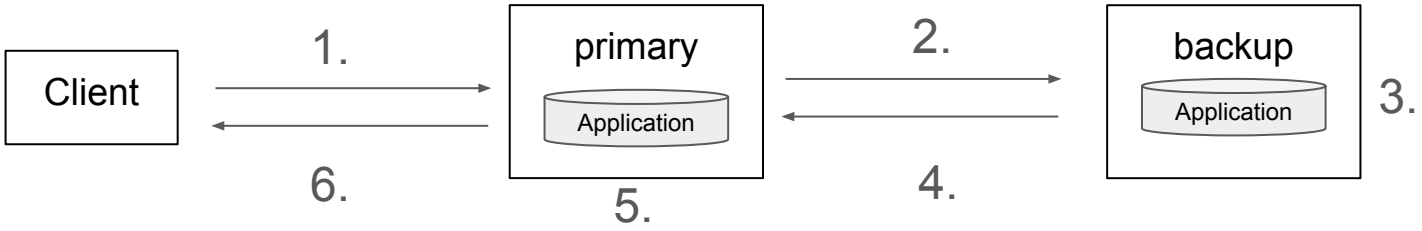
- Clients only talk to the primary
- only the primary can take requests
- Application and its data are replicated on both primary and backup
- if the primary dies, the view server promotes the backup to be the primary, and assigns a new backup from the idle server (if one is available)
- if the backup dies, the view server selects an idle server to be the new backup

viewReply: View X: (primary, backup)

General Case: the view server knows who are primary/backup

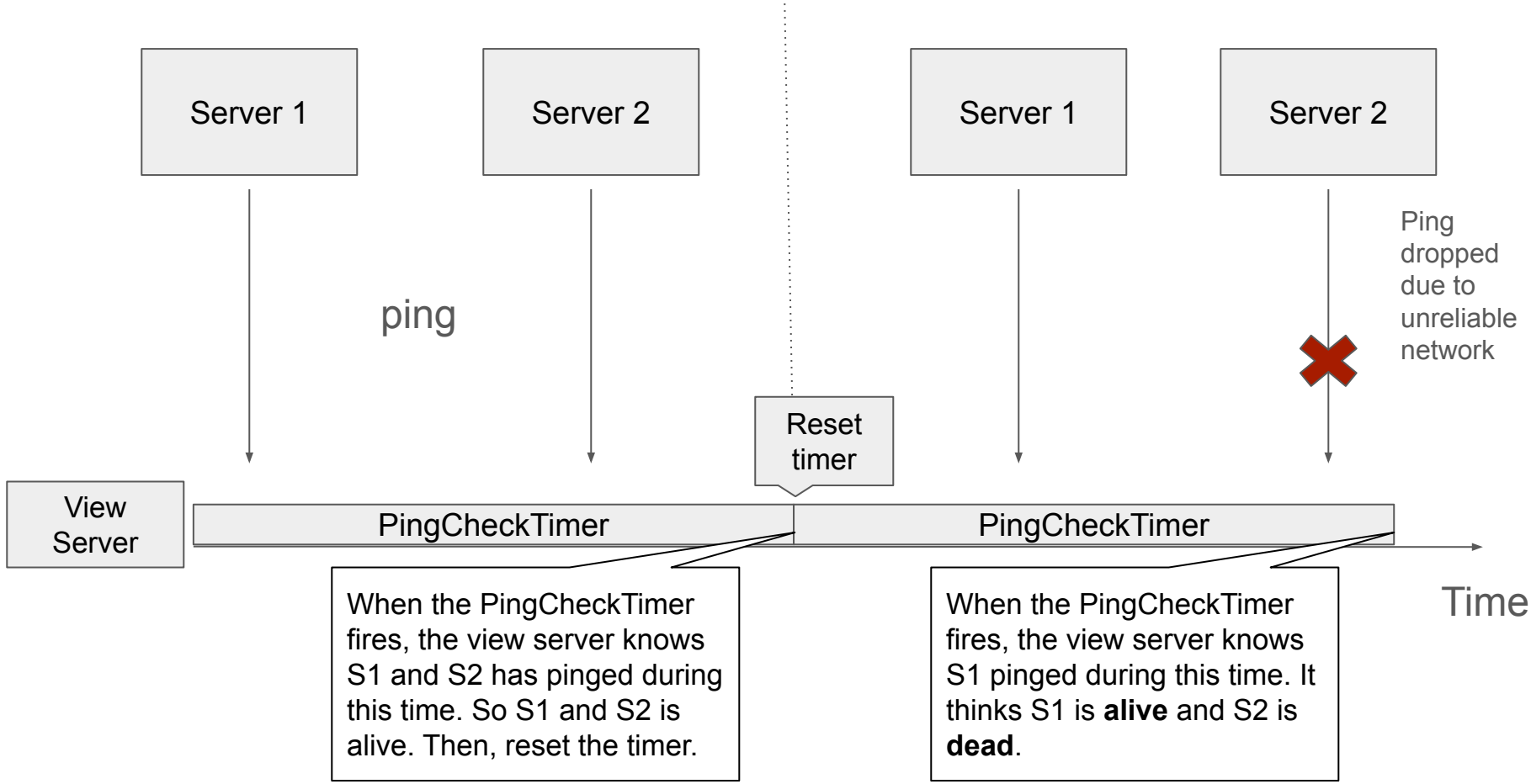
Under the condition where all servers and clients are on the same view told by the view server:

1. The client sends a request to the primary
2. The primary forwards the client's request to the backup, but **hasn't** executed it yet
3. The backup executes the request
4. The backup sends an acknowledgement message to the primary
5. The primary **now** can execute the request
6. The primary sends the result response to the client



Detecting Failure

- All servers ping the view server periodically
- If the View Server doesn't receive a Ping from a server since the last PingCheckTimer, it should consider the server to be “dead”

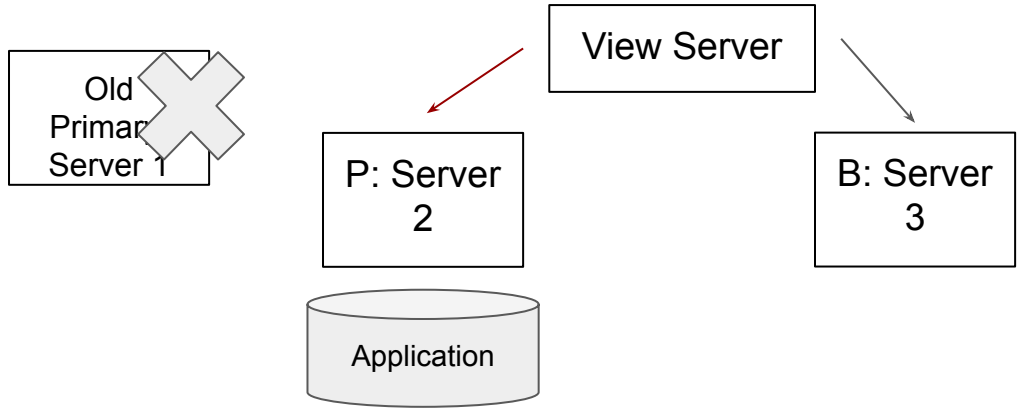

Server 1
Server 2
Server 1
Server 2
ping
Ping dropped due to unreliable network
Reset timer
View Server
PingCheckTimer
PingCheckTimer
When the PingCheckTimer fires, the view server knows S1 and S2 has pinged during this time. So S1 and S2 is alive. Then, reset the timer.
When the PingCheckTimer fires, the view server knows S1 pinged during this time. It thinks S1 is **alive** and S2 is **dead**.
Time

Detecting Failure

- All servers ping the view server periodically
- If the View Server doesn't receive a Ping from a server since the last PingCheckTimer, it should consider the server to be “dead”
- The “dead” servers can be “alive” again if the view server receives pings from them during a new PingCheckTimer. Those servers will be put to the pool of idle servers, waiting to be assigned roles.
  - “Dead” : servers may be crashed or the message sent to the view server is dropped/delayed
  - “Alive” : The view server receives the ping from others servers with a view of ….
    - Same view? Or even a different view in some case? Your design choice here:)
- If the primary is dead, chooses the backup to be the new primary in the new view.
- If backup is dead, selects an idle server from the pool as new backup in the new view

State Transfer - new primary

Now that the backup is the primary, needs to replicate application state to *its* backup

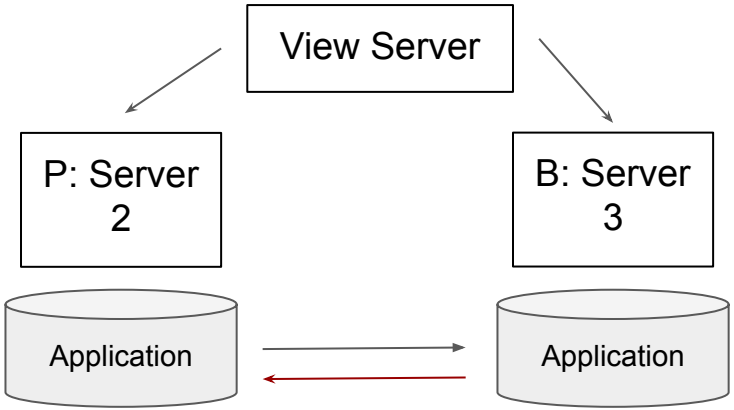


Server 3 knows its role as a backup

1) Server 2 got a view reply from the view server and becomes the new primary. It knows server 3 is its backup. It will send the entire AMOapp to server 3, initiating the state transfer.

~~View i: (Server 1, Server 2)~~
View i + 1: (Server 2, Server 3)

State Transfer - new primary

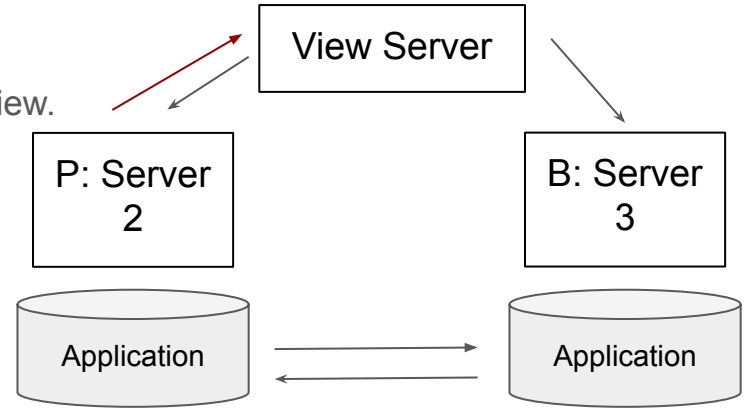


1) Server 2 got a view reply from the view server and becomes the new primary. It knows server 3 is its backup. It will send the entire AMOapp to server 3, initiating the state transfer.

2) After Server 3 received the AMOapp, it will sent a state transfer acknowledgement message back to its primary, “I have backed up what you sent to me”.

State Transfer - new primary

3) Then, after receiving the ACK, Server 2 tells the view server that the state transfer is complete. Acknowledged the view.

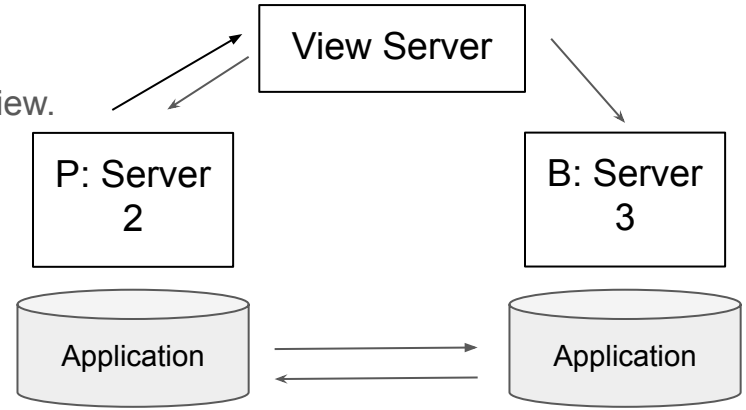


1) Server 2 got a view reply from the view server and becomes the new primary. It knows server 3 is its backup. It will send the entire AMOapp to server 3, initiating the state transfer.

2) After Server 3 received the AMOapp, it will sent a state transfer acknowledgement message back to its primary, “I have backed up what you sent to me”.

State Transfer - new primary

3) Then, after receiving the ACK, Server 2 tells the view server that the state transfer is complete. Acknowledged the view.



During state transfer:

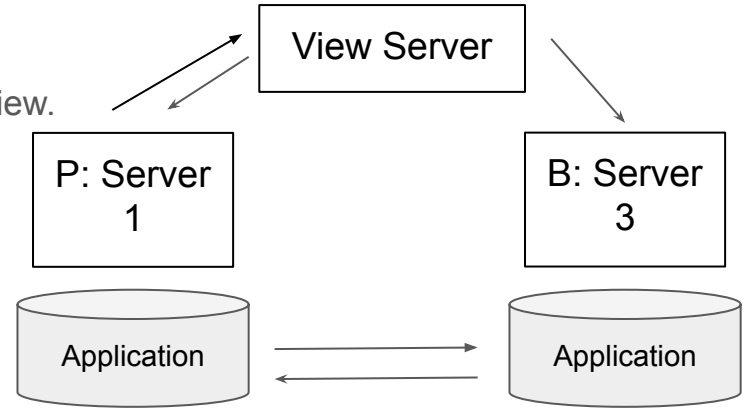
- The new primary will not accept any client request.
- The View server cannot move to the next view

1) Server 2 got a view reply from the view server and becomes the new primary. It knows server 3 is its backup. It will send the entire AMOapp to server 3, initiating the state transfer.

2) After Server 3 received the AMOapp, it will sent a state transfer acknowledgement message back to its primary, “I have backed up what you sent to me”.

State transfer - new backup

3) Then, after receiving the ACK, Server 1 tells the view server that the state transfer is complete. Acknowledged the view.



1) Server 1 got a view reply from the view server and becomes the new primary. It knows server 3 is its backup. It will send the entire AMOapp to server 3, initiating the state transfer.

2) After Server 3 received the AMOapp, it will sent a state transfer acknowledgement message back to its primary, “I have backed up what you sent to me”.

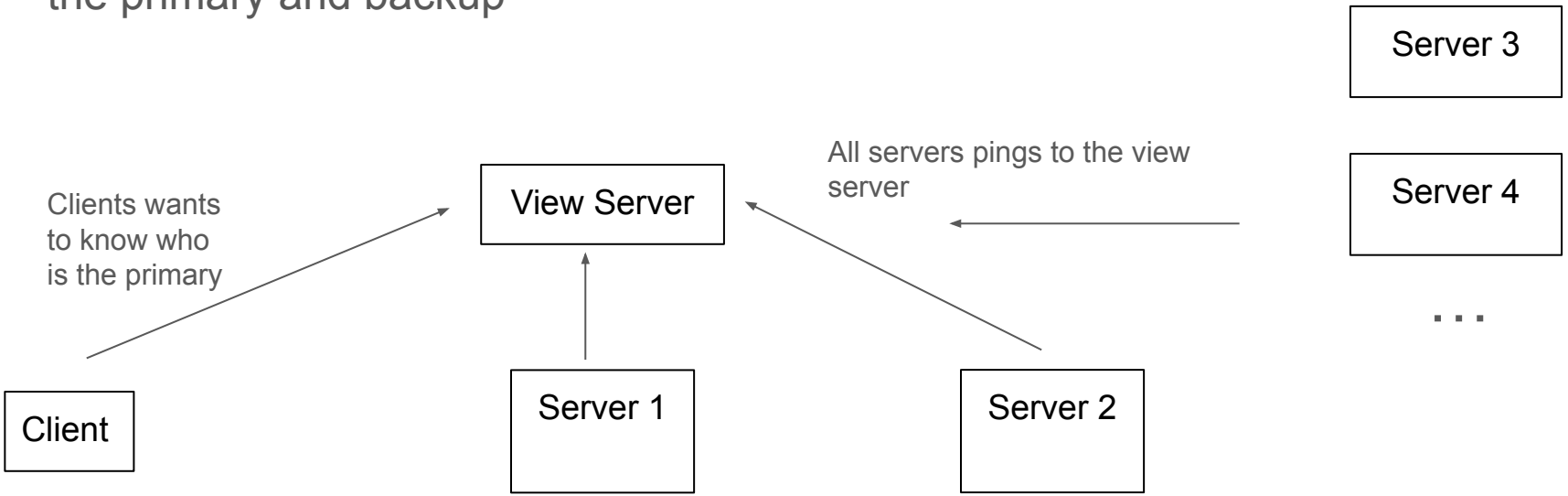
State transfer

- Pings from primary during state transfer should reflect old view
  - Primary should still ping to view server so view server knows primary is alive.
    - So the view server can accept view num i - 1 from the primary?
  - The primary only moves to new view **once** state transfer is complete.
- Backup can receive duplicated/delayed state transfer messages from its primary
  - Need to ensure that state on backup is only overwritten **once** per view change
  - What happens if the state transfer ack gets dropped?
    - Resend state transfer message, but backup should **not** overwrite their state if they have already completed the state transfer

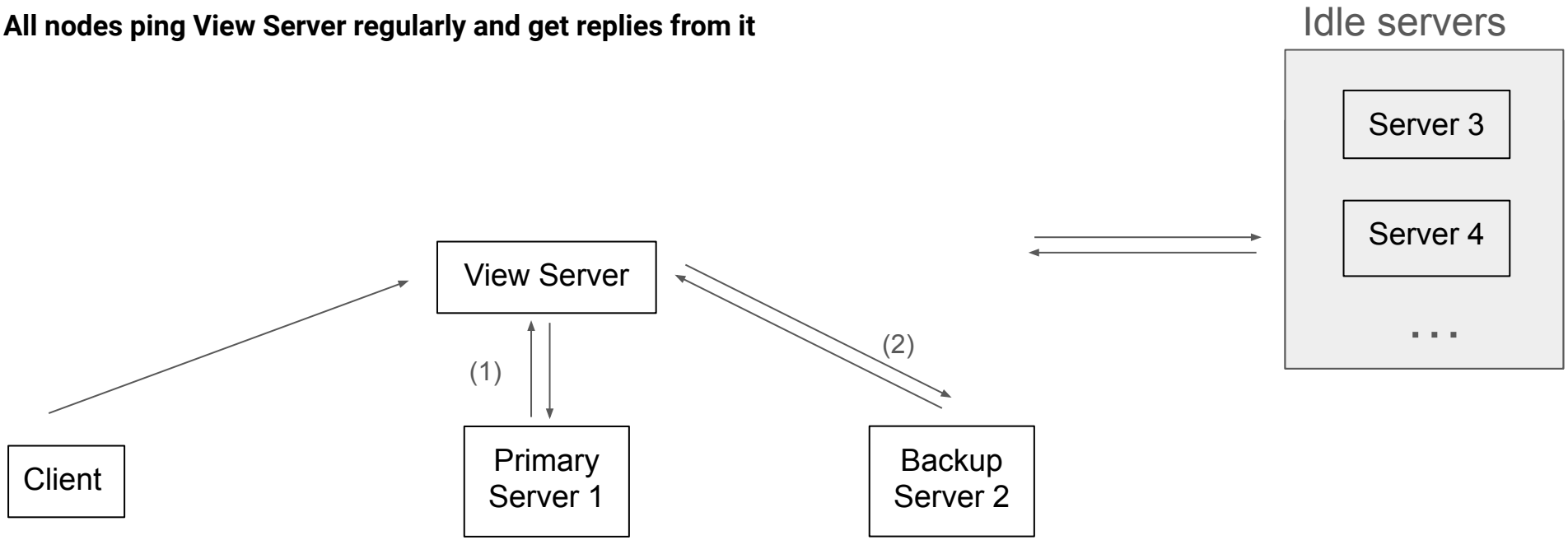
Special Case - at start-up

View 0 : (unknown, unknown)

Initially, the view server does not know who are the primary and backup



All nodes ping View Server regularly and get replies from it



(1) The view server receives Server 1's ping first, so it becomes the primary

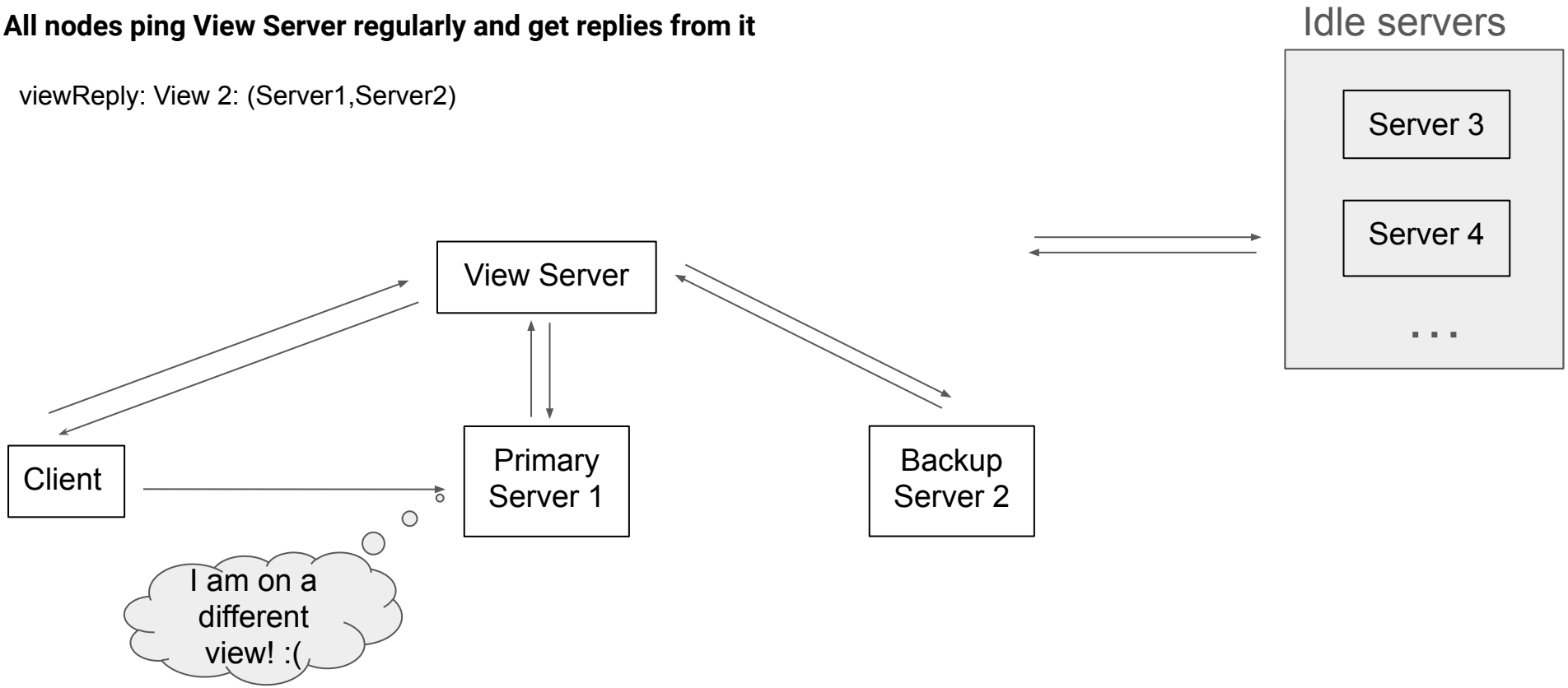
View 1 : (Server 1, unknown)

(2) The view server receives Server 2's ping second, so it becomes the backup

View 2 : (Server 1, Server 2)

All nodes ping View Server regularly and get replies from it

viewReply: View 2: (Server1,Server2)

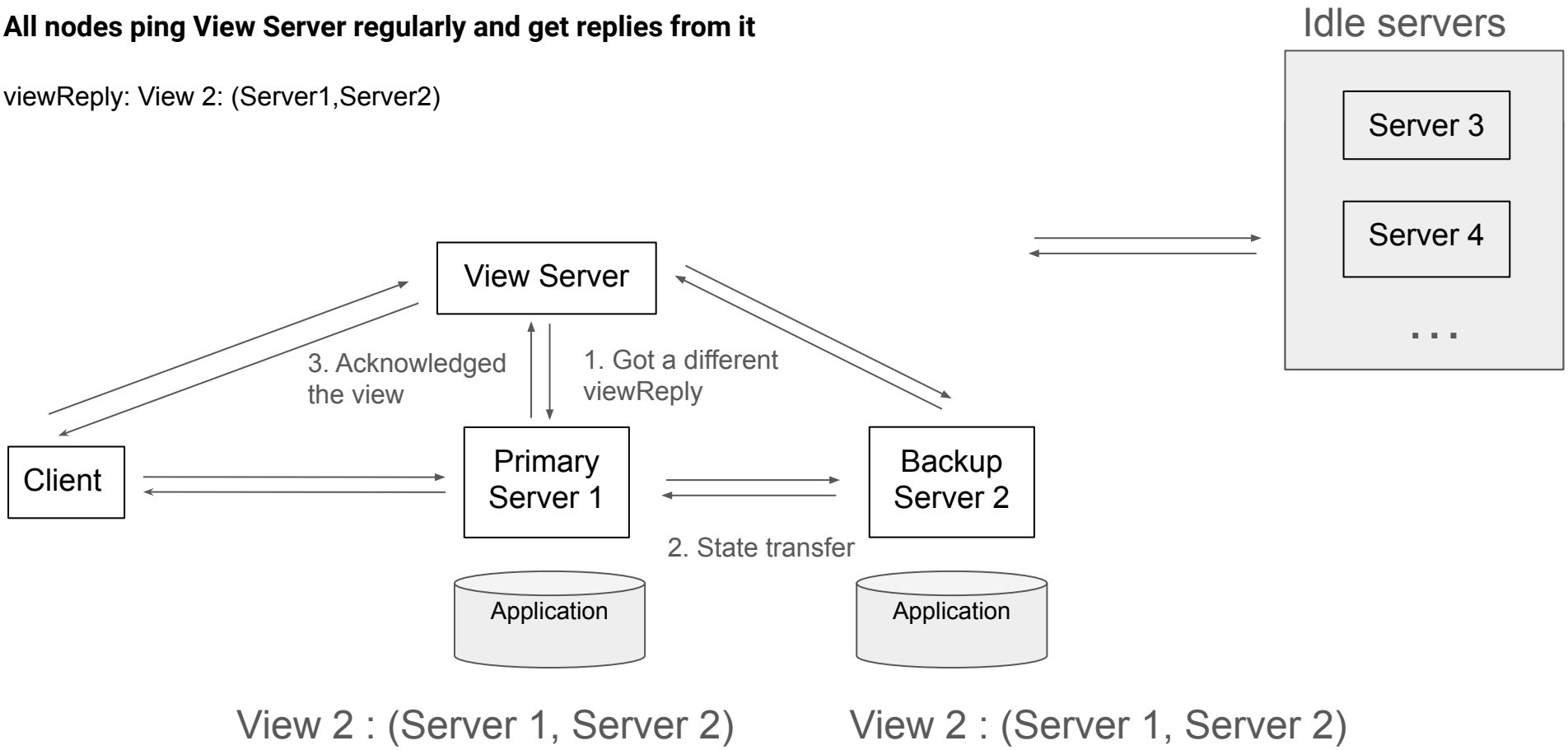


View 1 : (Server 1, unknown)

View 2 : (Server 1, Server 2)

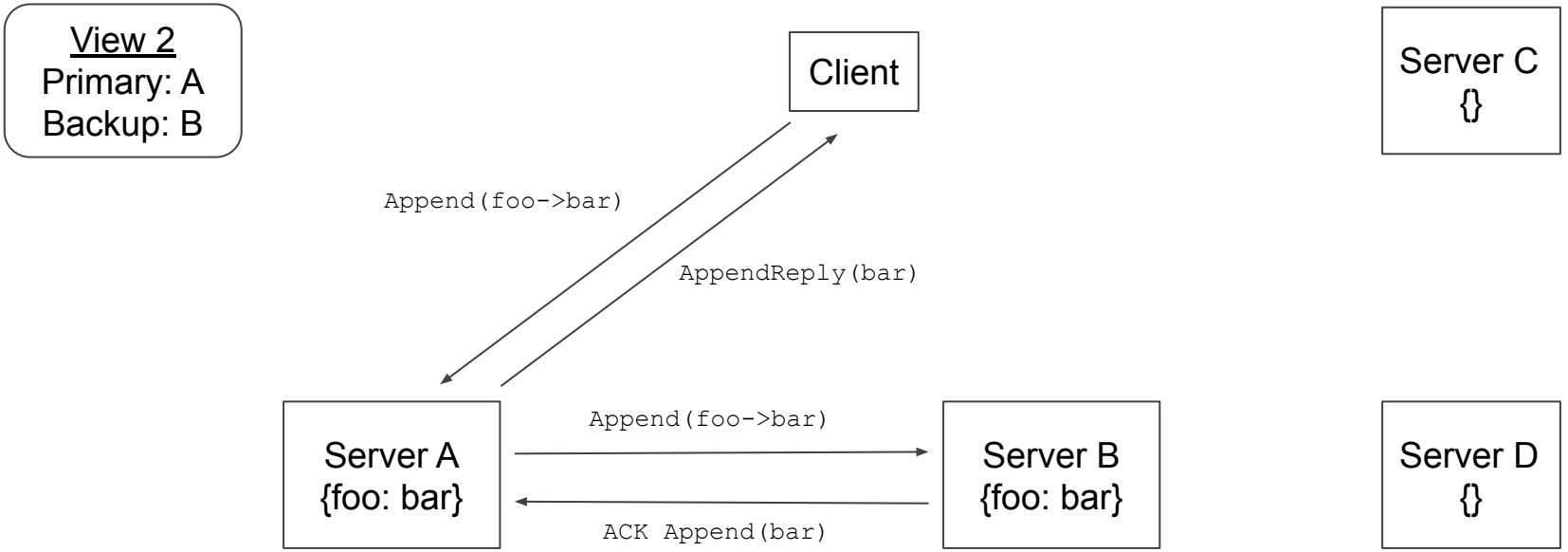
All nodes ping View Server regularly and get replies from it

viewReply: View 2: (Server1,Server2)



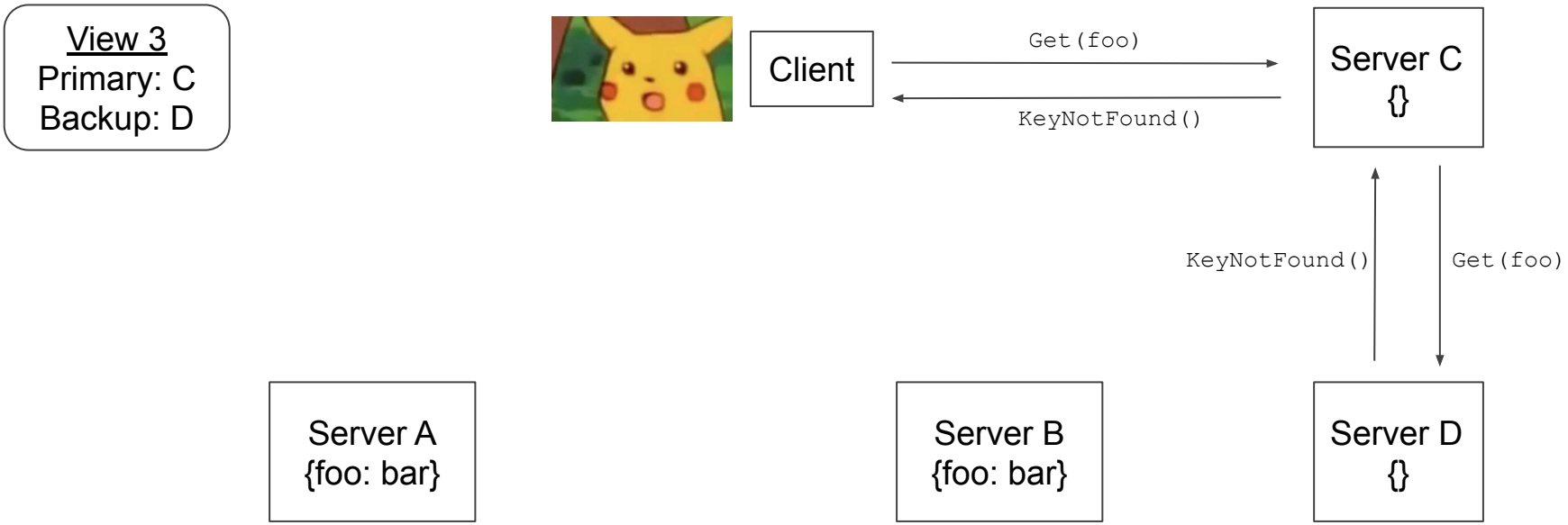
Rules to follow:

1. Primary in view i+1 must have been backup or primary in view i



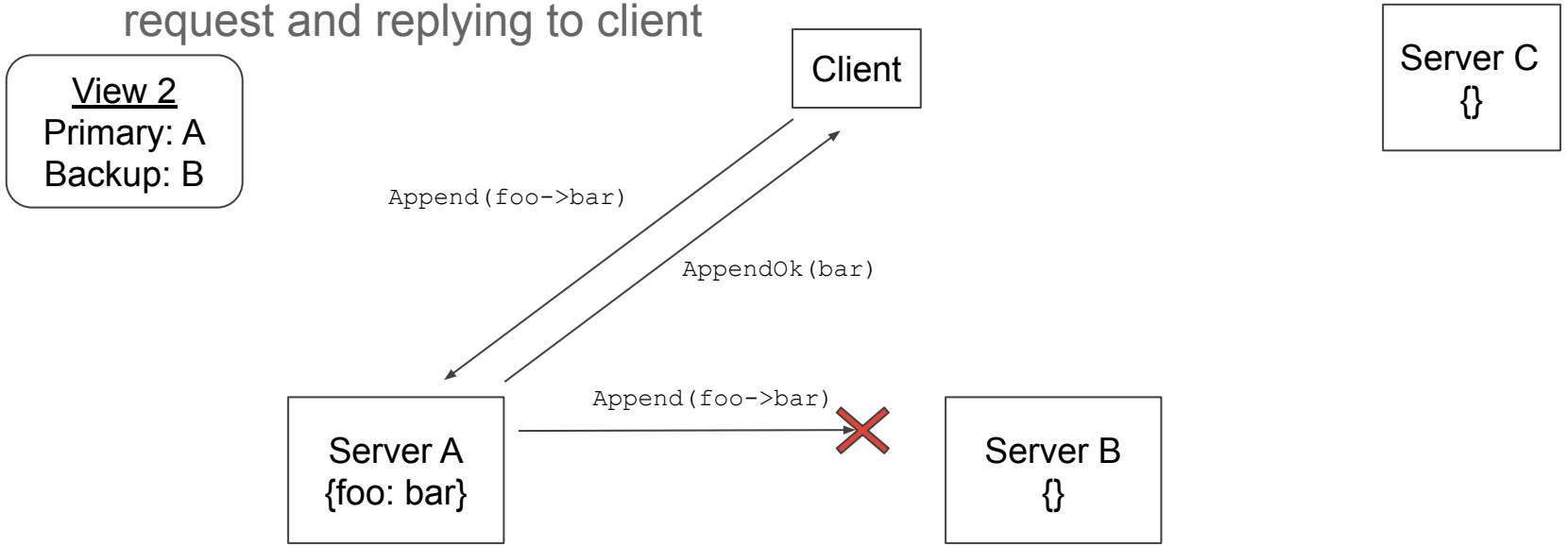
Rules to follow:

1. Primary in view i+1 must have been backup or primary in view i

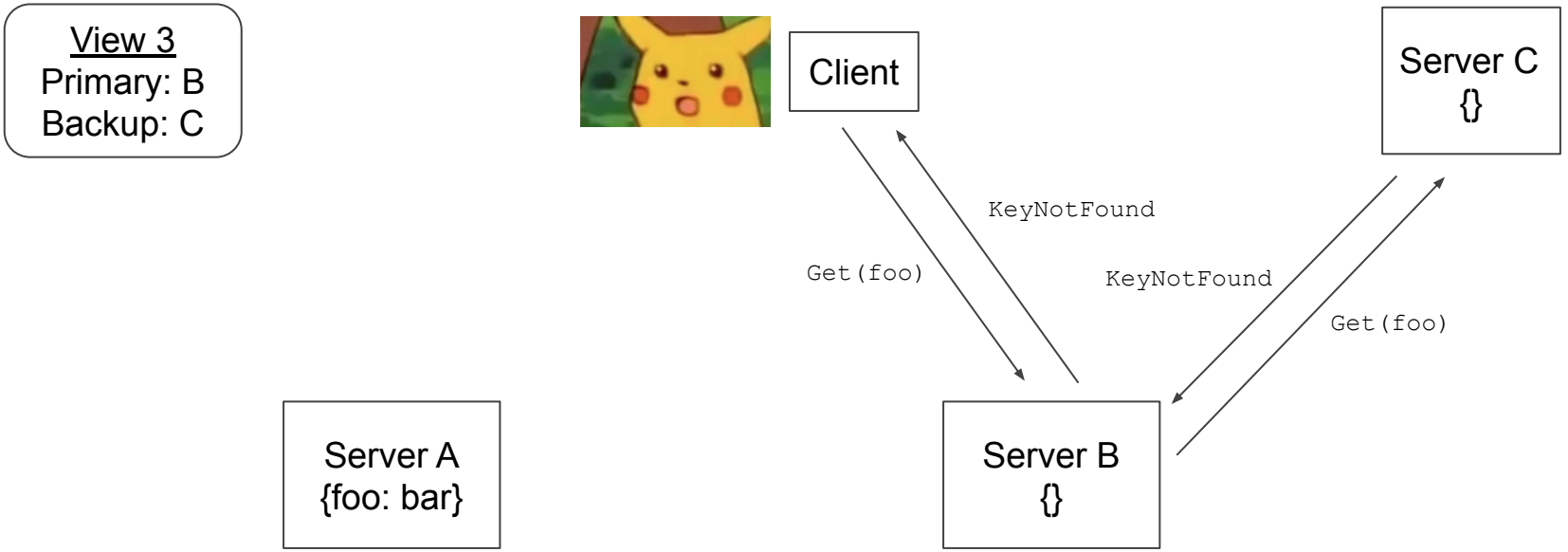


Rules to follow:

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 request before doing request and replying to client



2. Primary must wait for backup to accept/execute each request before doing request and replying to client



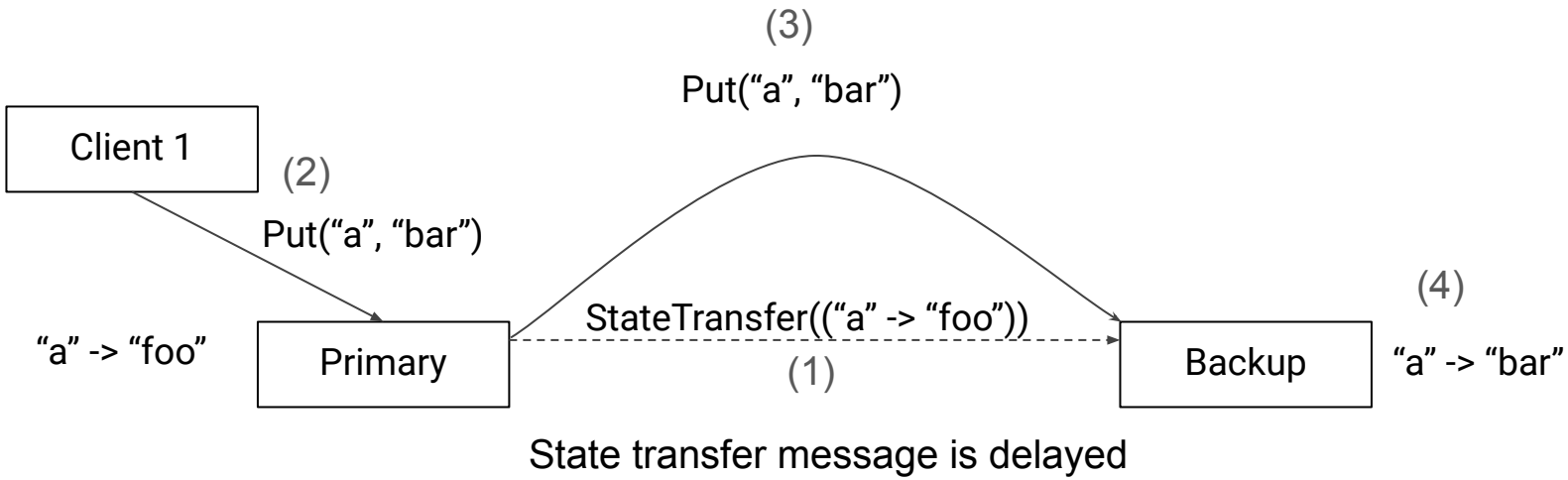
Rules to follow:

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 request before doing request and replying to client
3. Backup must only accept forwarded client requests from its primary, which is indicated from the current view
    a. Hint: **Send view with every message** and check view number on receive
4. Non-primary must reject client requests

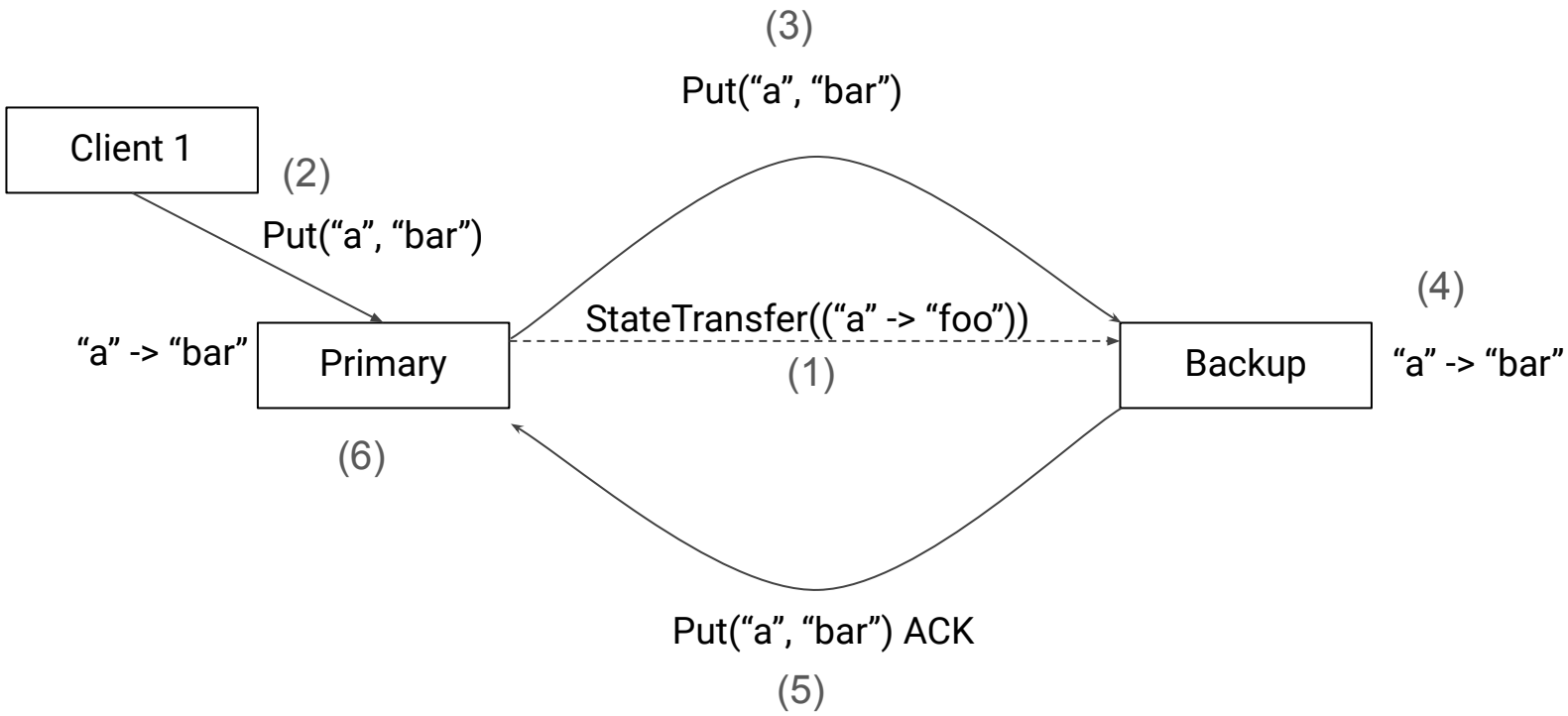
Rules to follow:

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 request before doing request and replying to client
3. Backup must only accept forwarded client requests from its primary, which is indicated from the current view
    a. Hint: **Send view with every message** and check view number on receive
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer
    a. Problem with processing requests during a state transfer

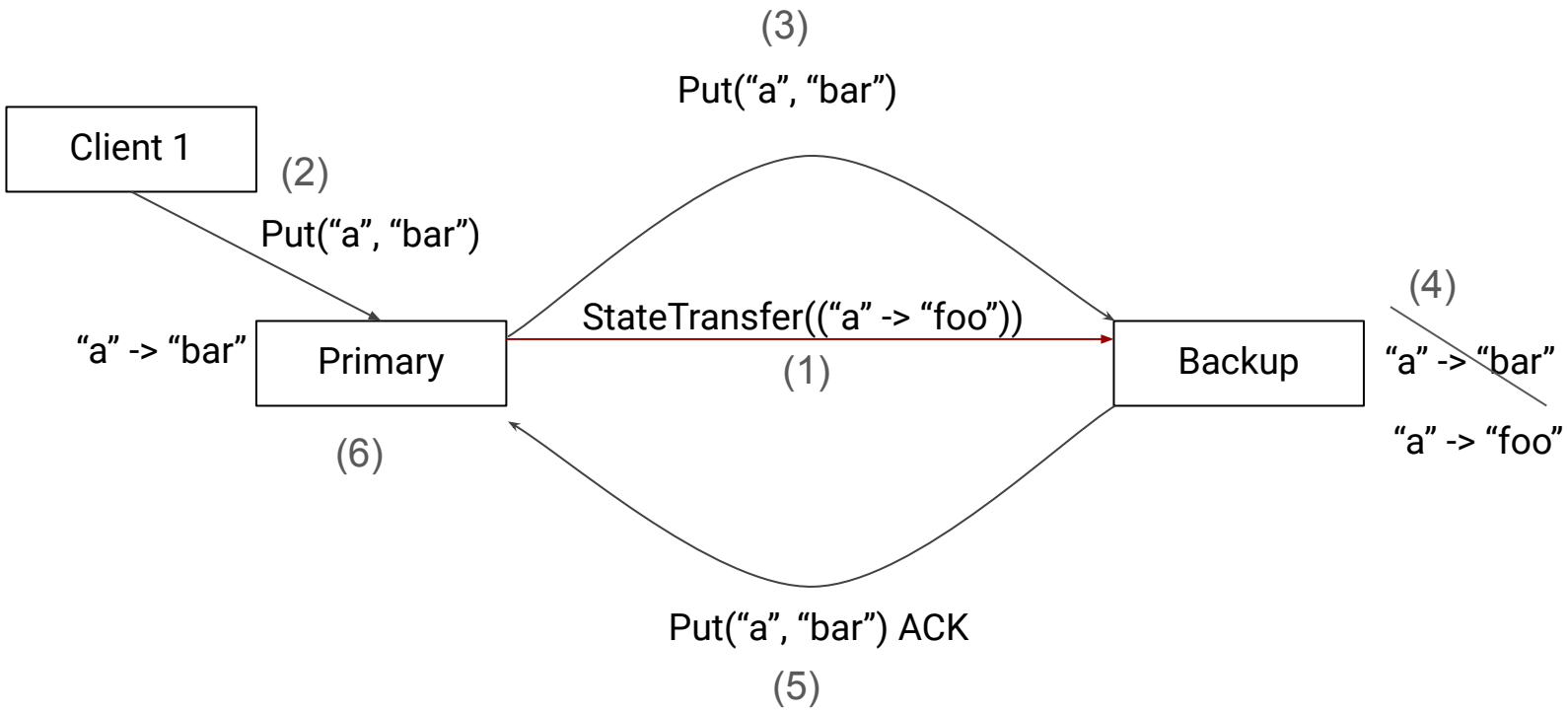
Process client during state transfer



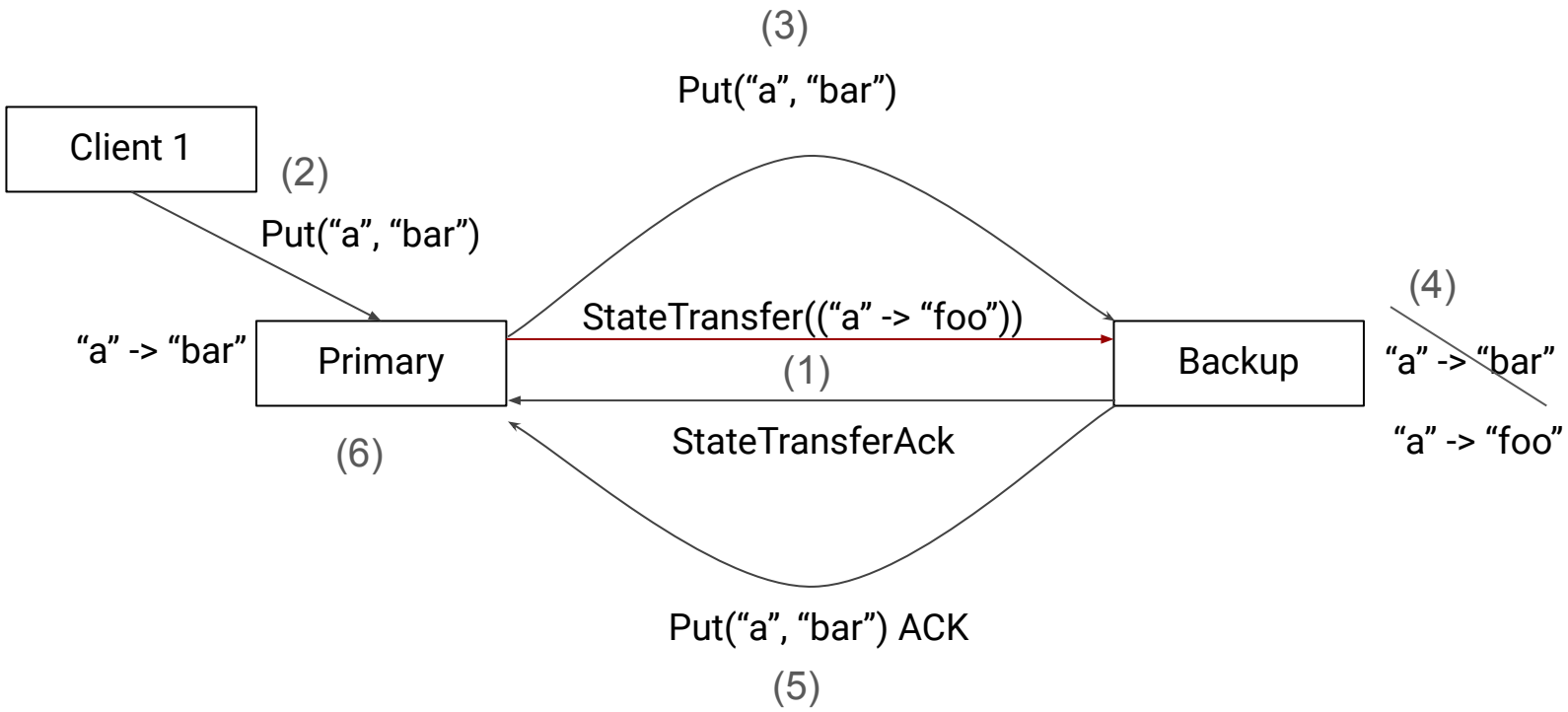
Process client during state transfer



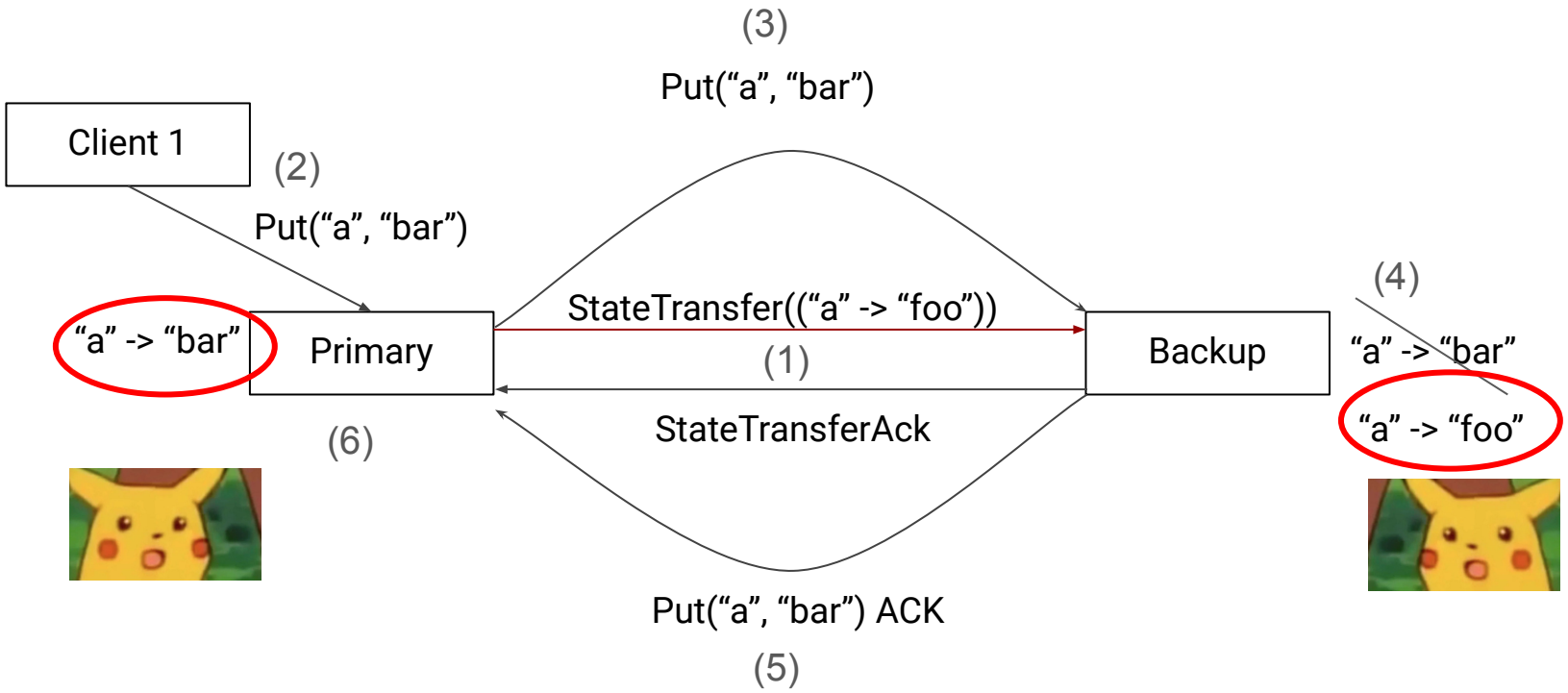
Process client during state transfer



Process client during state transfer



Process client during state transfer



Rules to follow:

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 request before doing request and replying to client
3. Backup must only accept forwarded client requests from its primary, which is indicated from its current view
    a. Hint: **Send view with every message** and check view number on receive
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer
    a. Problem with processing requests during a state transfer

What to design

- Messages
  - Request / Reply
  - Forward request from primary to backup
  - State transfer
  - Acknowledgements
  - Others?
- Timer handlers
  - Only add necessary timers
- States
  - AMOApplication (only server)
  - Sequence number (on client)
  - Current View
  - Others?

Hint

- Don't need 'curr view' and 'next view' in PBServer
  - in handleViewReply, we don't care what the node WAS, we only care about what it is becoming in the new view, so it’s unnecessary to store more than 1 view
- Consider primary pinging with old view or not pinging at all during state transfer
- On client: easiest to ask for current view whenever there is a timeout
- Things that can increase the number of states to explore:
  - Every state transition: setting new timers, sending new messages
  - Unnecessary state information, e.g. a retry counter that just keeps incrementing

Looking forward to discussing with your designs on next Friday