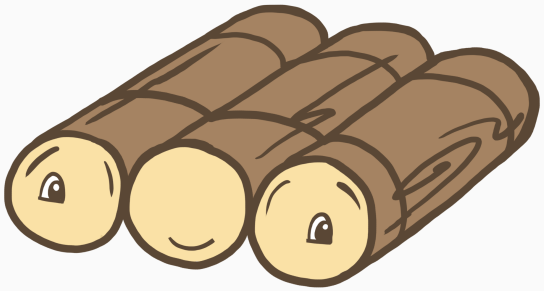


Implementing Raft

Xuangui Huang David Stalfa

Raft Basics

Consensus protocol for log replication

Strong Leadership

- All instructions flow from leader to followers

Randomized Leader Elections

- Random timeouts determine when leader has failed

Joint Consensus for Membership Changes

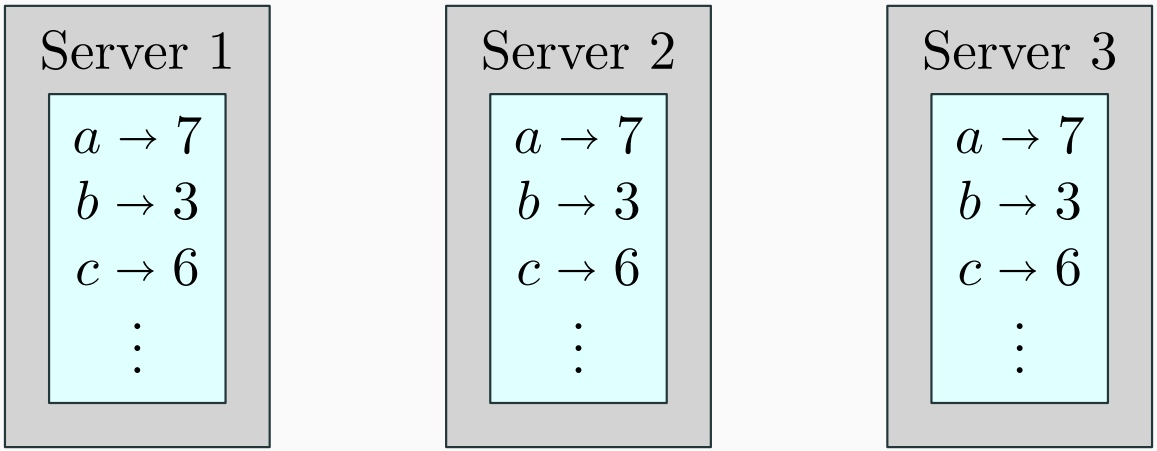
- Intermediate phase where leader manages both old and new configurations

Key-Value Store

Each server holds replica of key-value store

Client can request to

- **put** a key-value pair in the store
- **get** from the store the value associated with a given key



Implemented Protocols

Log Replication

Leader Election

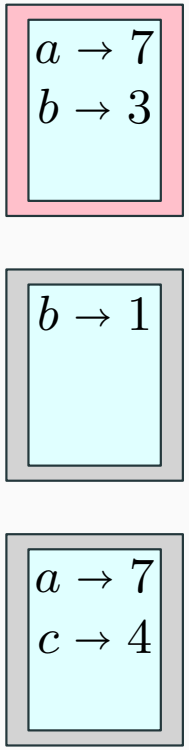
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

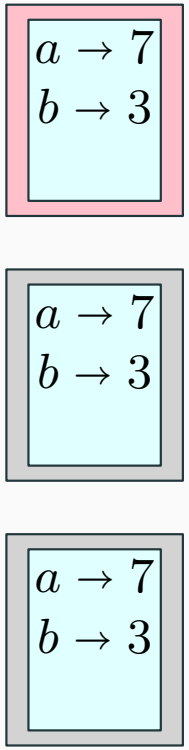
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

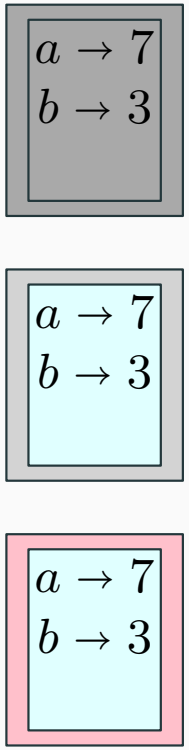
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

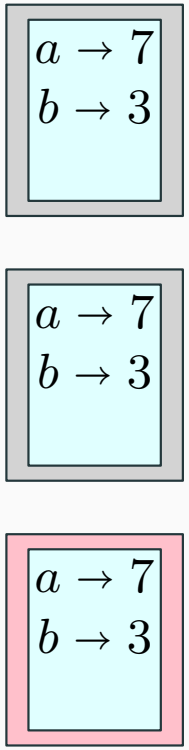
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

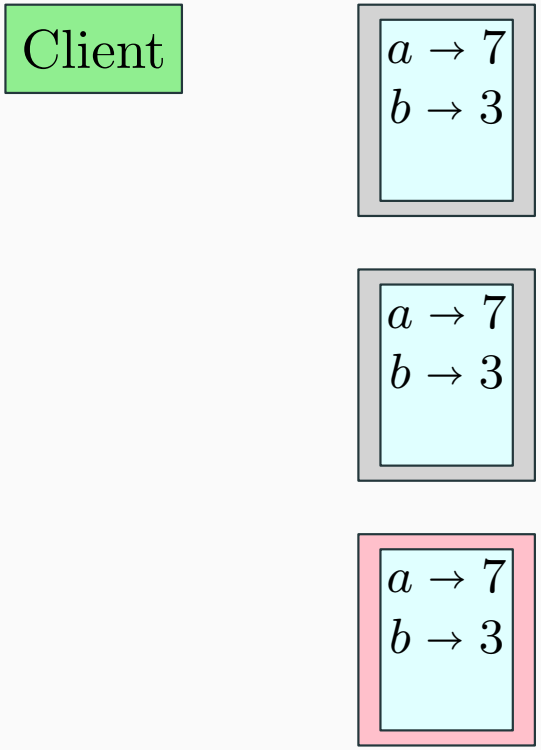
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

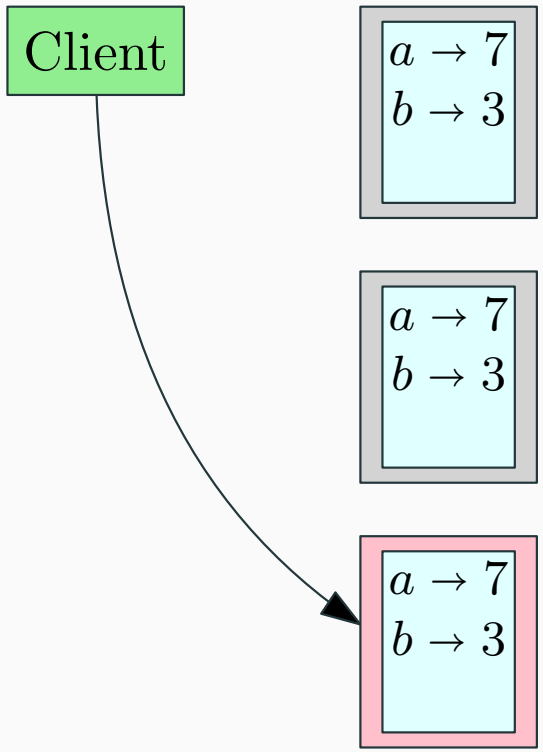
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

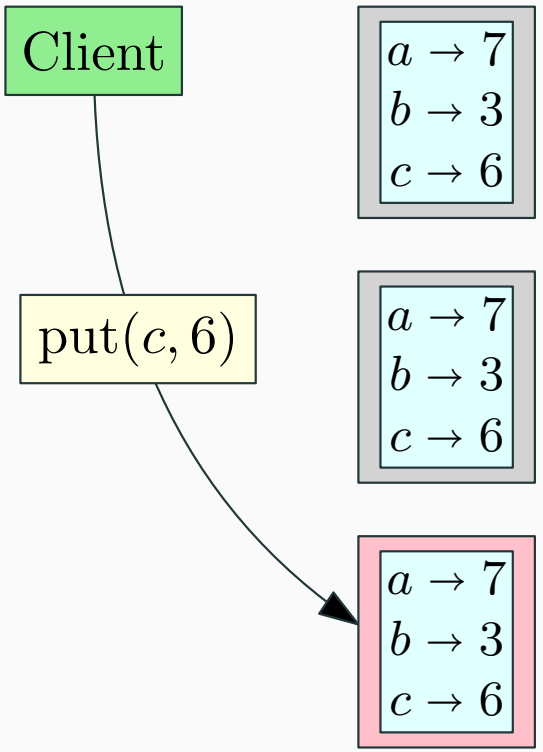
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Implemented Protocols

Log Replication

Leader Election

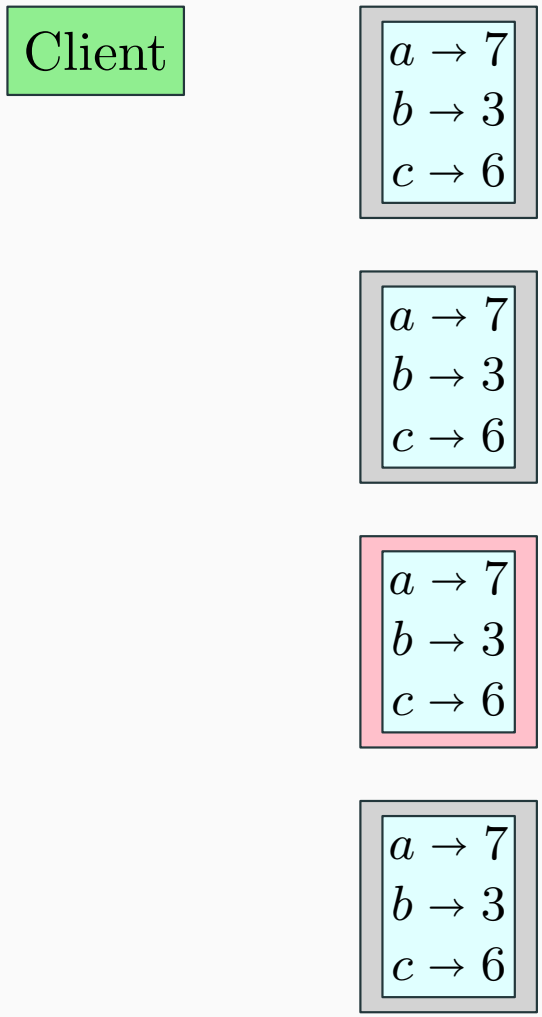
Persistent Storage

Client Interaction

- Find leader
- Issue **put**/**get** requests

Membership Change

- Supports adding servers to configuration
- No downtime



Technical Details of Implementation

Implemented in Python3.6

- Nine previous implementations of Raft in Python[1]

Uses Asynchronous I/O (asyncio) library

- One previous implementation in Python using asyncio[2]

Servers communicate using UDP

Runs on Northeastern VDI's

Client commands issued as standard input

[1] `https://raft.github.io/`
[2] Ibid.

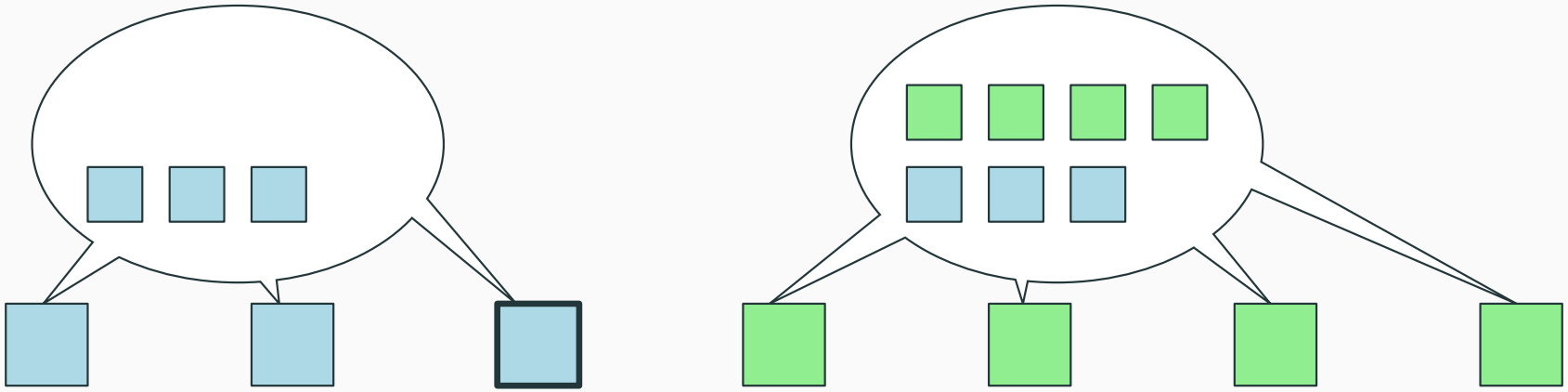
Design Decisions

Add-Only Membership Changes

Implementation supports adding but not removing servers

Still requires non-trivial application of joint consensus

Otherwise, might end up with separate majorities and multiple leaders

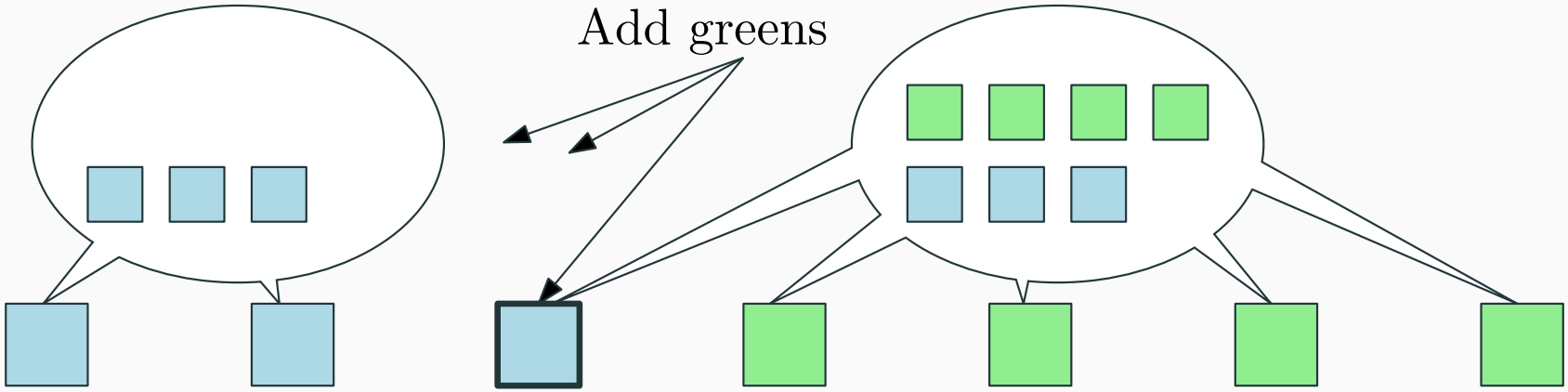


Add-Only Membership Changes

Implementation supports adding but not removing servers

Still requires non-trivial application of joint consensus

Otherwise, might end up with separate majorities and multiple leaders

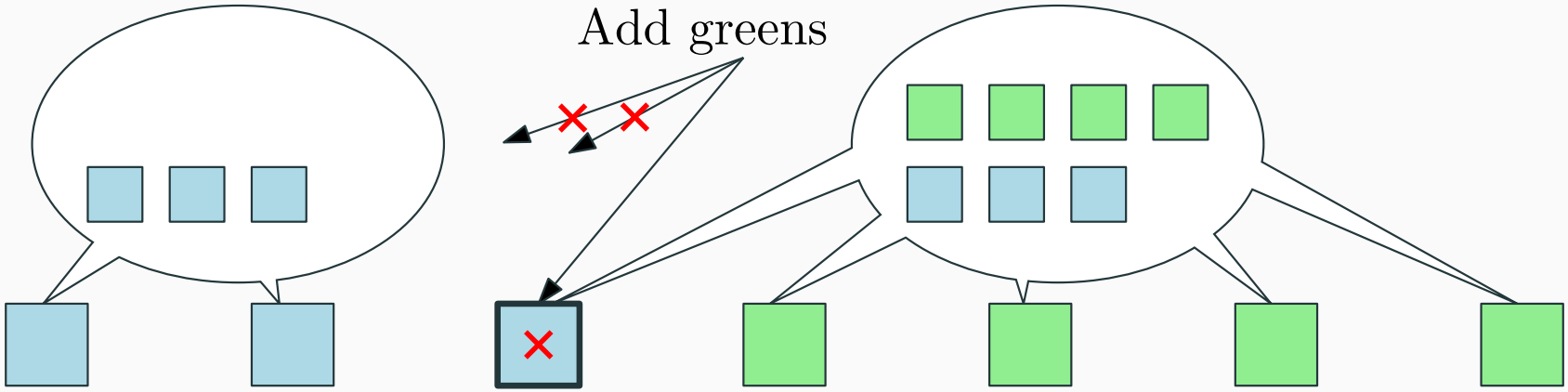


Add-Only Membership Changes

Implementation supports adding but not removing servers

Still requires non-trivial application of joint consensus

Otherwise, might end up with separate majorities and multiple leaders

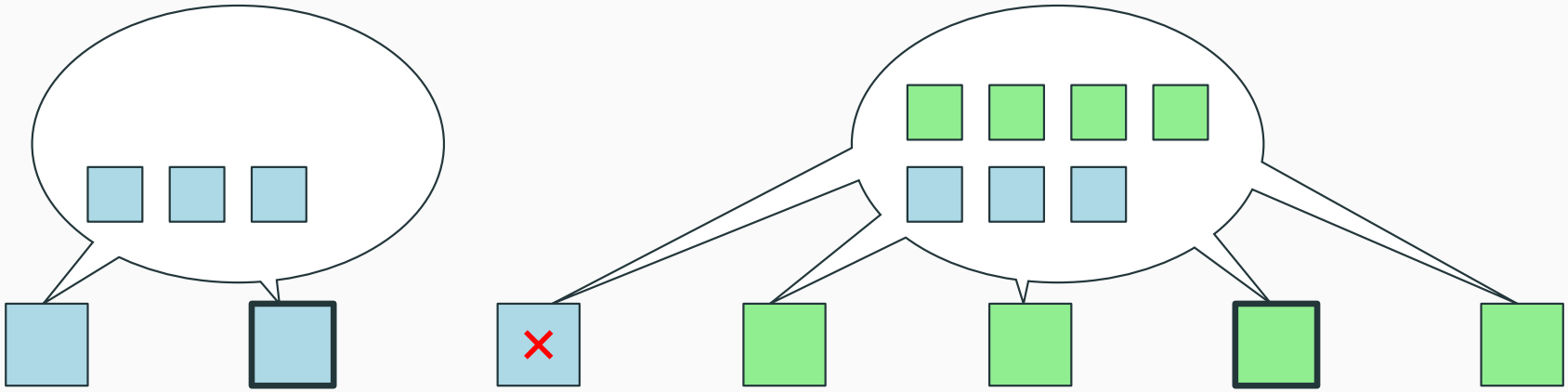


Add-Only Membership Changes

Implementation supports adding but not removing servers

Still requires non-trivial application of joint consensus

Otherwise, might end up with separate majorities and multiple leaders



Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

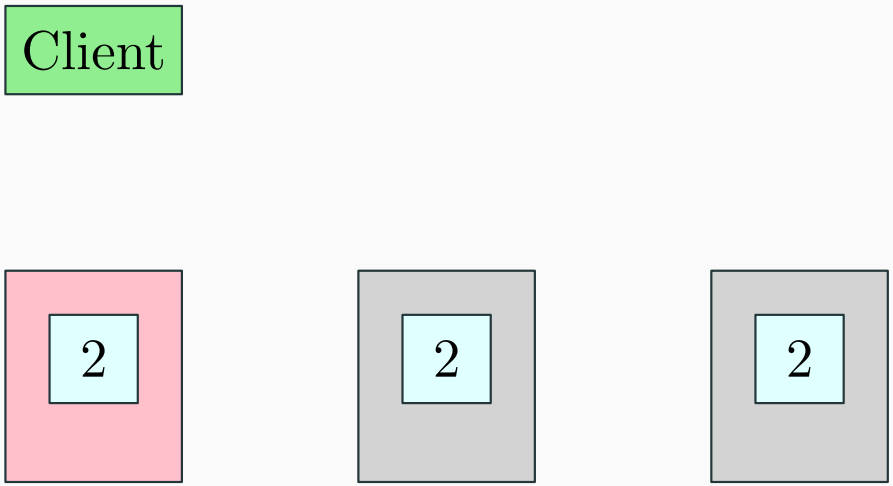
In general, processing a client request twice violates linearizability.

Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

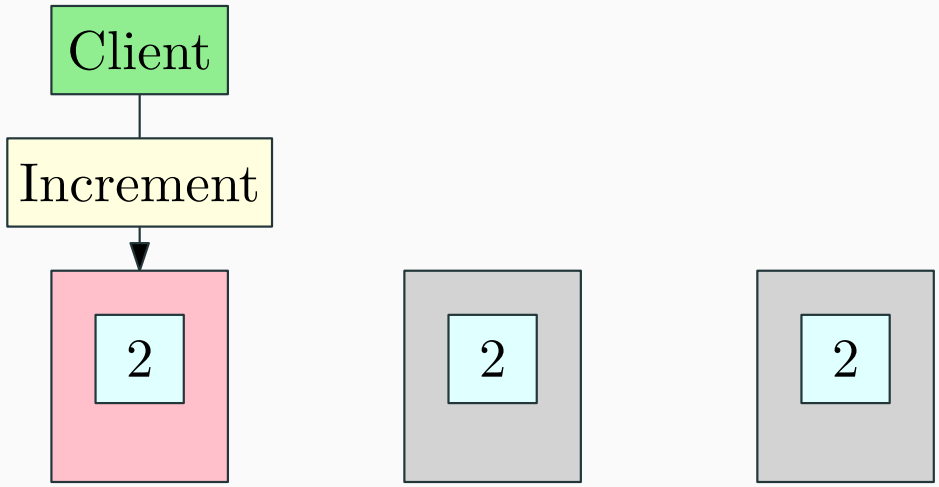


Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

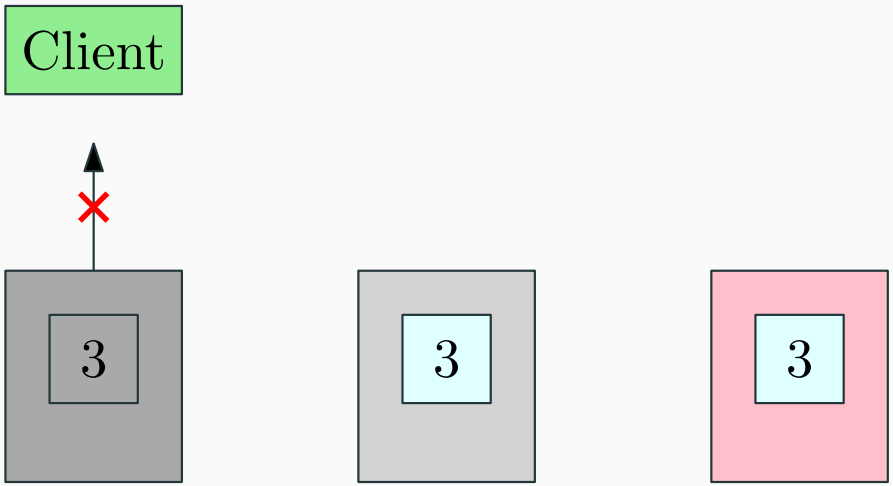


Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

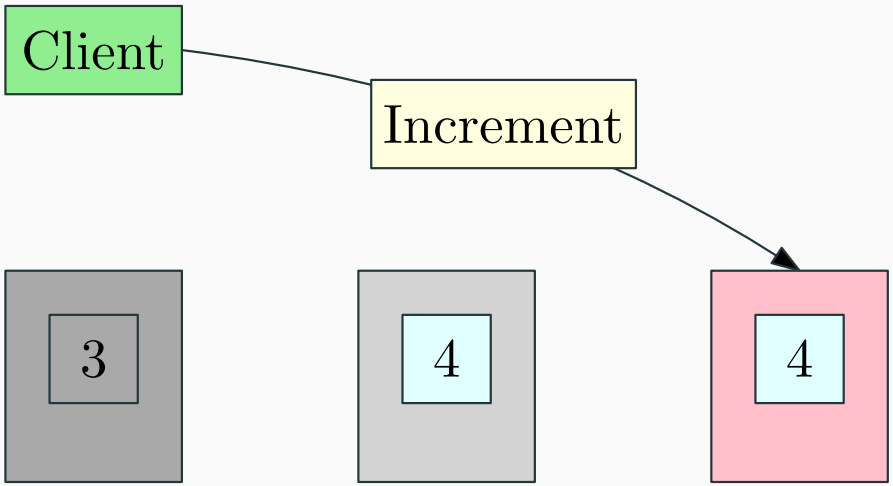


Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.



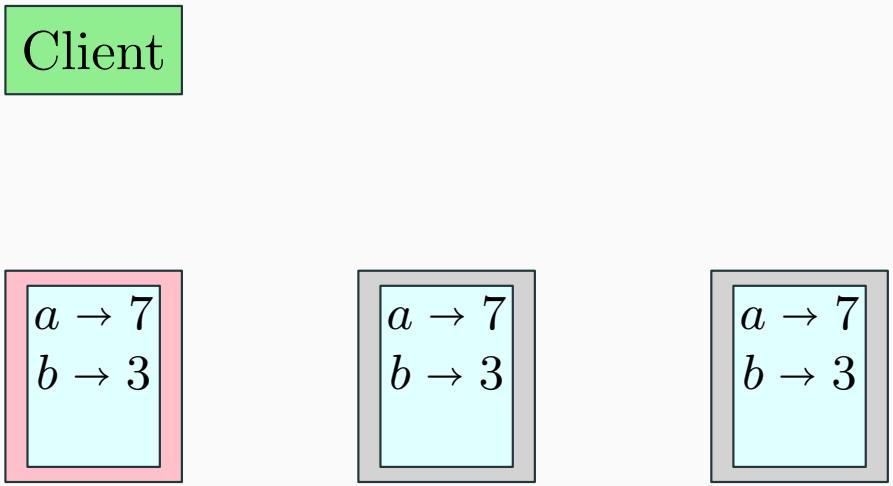
Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

We allow client requests to be processed multiple times. Since we implement a key-value store, our semantics is still linearizable.



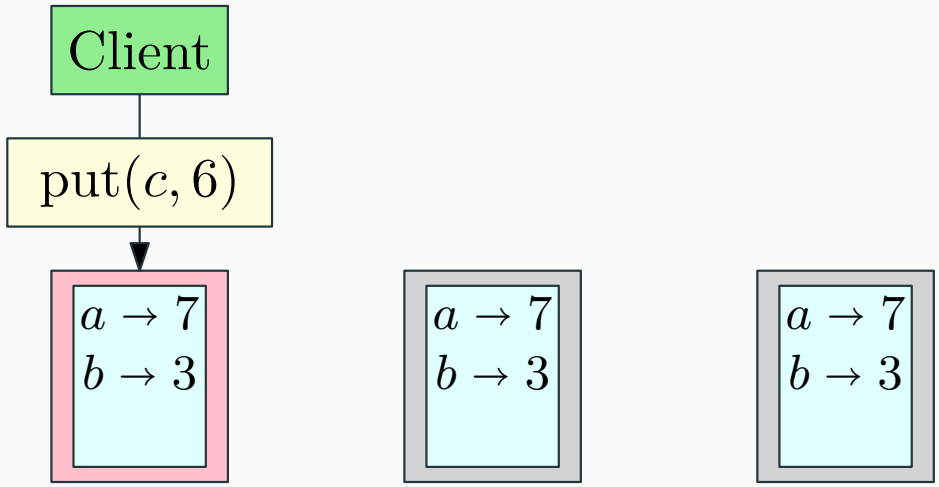
Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

We allow client requests to be processed multiple times. Since we implement a key-value store, our semantics is still linearizable.



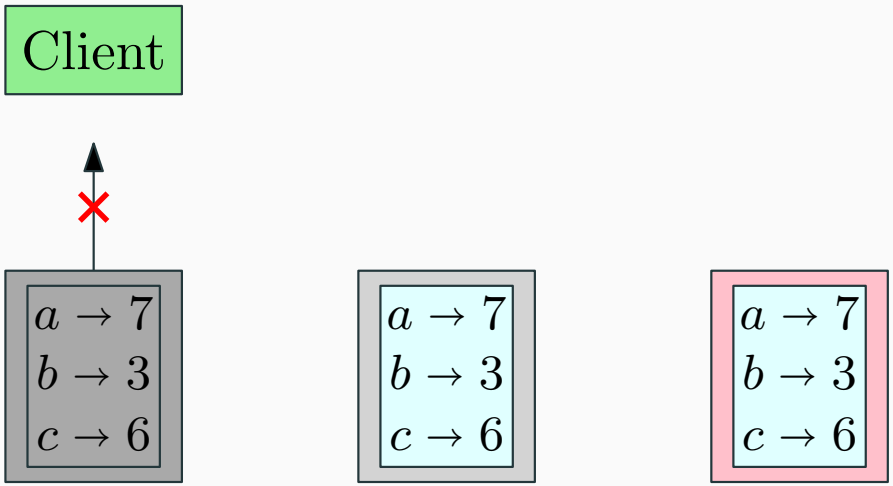
Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

We allow client requests to be processed multiple times. Since we implement a key-value store, our semantics is still linearizable.



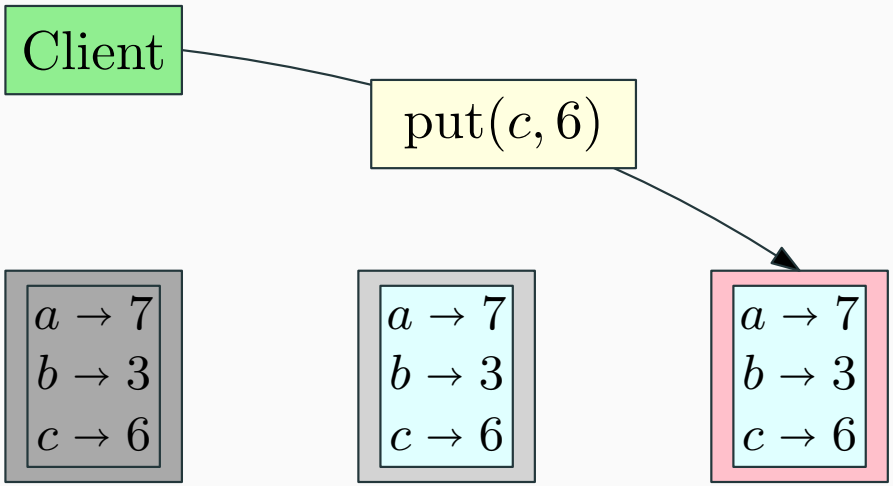
Linearizable Semantics

Correctness requires that our protocol's semantics be *linearizable*.

Informally, a protocol's semantics is linearizable if the result of the protocol is identical to one in which each completed client request is executed instantaneously.

In general, processing a client request twice violates linearizability.

We allow client requests to be processed multiple times. Since we implement a key-value store, our semantics is still linearizable.



Other Design Decisions

Handle read (get) requests the same as write (put) requests

- Raft optimizes read-only requests by exchanging heartbeats rather than the full commit protocol

Lead server appends only a single log entry at a time

- Raft optimizes this operation by appending several entries at a time

Invoke membership changes with a call to **add** from the Client

- Requests are made in parallel with put/get requests

Add servers from a pool with known addresses

Demos

Basic Functionality

Put key-value pairs into the store

Get values from the store by key

System remains available and consistent:

- the leader crashes
- any majority of servers are online
- servers are added

System is unavailable if:

- a majority of servers are offline

If system is ever unavailable, it becomes available again when enough servers recover

`https://drive.google.com/file/d/1Fc1NDyUKlPlhp1eA9LdL2zFACjuh3_mf/view?usp=sharing`

Active Membership Change

Raft promises that the system remains available throughout a membership change, assuming a majority of servers are online.

Here, we slow down a configuration change and make several put/get requests to show that our implementation meets this condition.

`https://drive.google.com/file/d/1we4QX9g0P_KZiqyHGwliSJmmu6tWnHpo/view?usp=sharing`

Joint Consensus Block

Joint consensus requires that decisions made between the start and completion of a membership change must be approved by majorities in both the old and new configurations.

Here we simulate a majority of old servers crashing in the middle of a membership change, and show that in this case the system blocks.

`https://drive.google.com/file/d/1PWhHiyu0tOEqNMZirqFsyxgGTR2QwJb5/view?usp=sharing`

Leader Election Constraint

Raft requires that the log of any newly elected leader contains all committed entries from previous terms.

Here we show that if a server does not have an up-to-date log, then its request vote messages are rejected.

`https://drive.google.com/file/d/1_slALcMnZH8yNUO2sgO0BHBhTjNwocaX/view?usp=sharing`

Thank you!

Q & A