ZooKeeper Tutorial

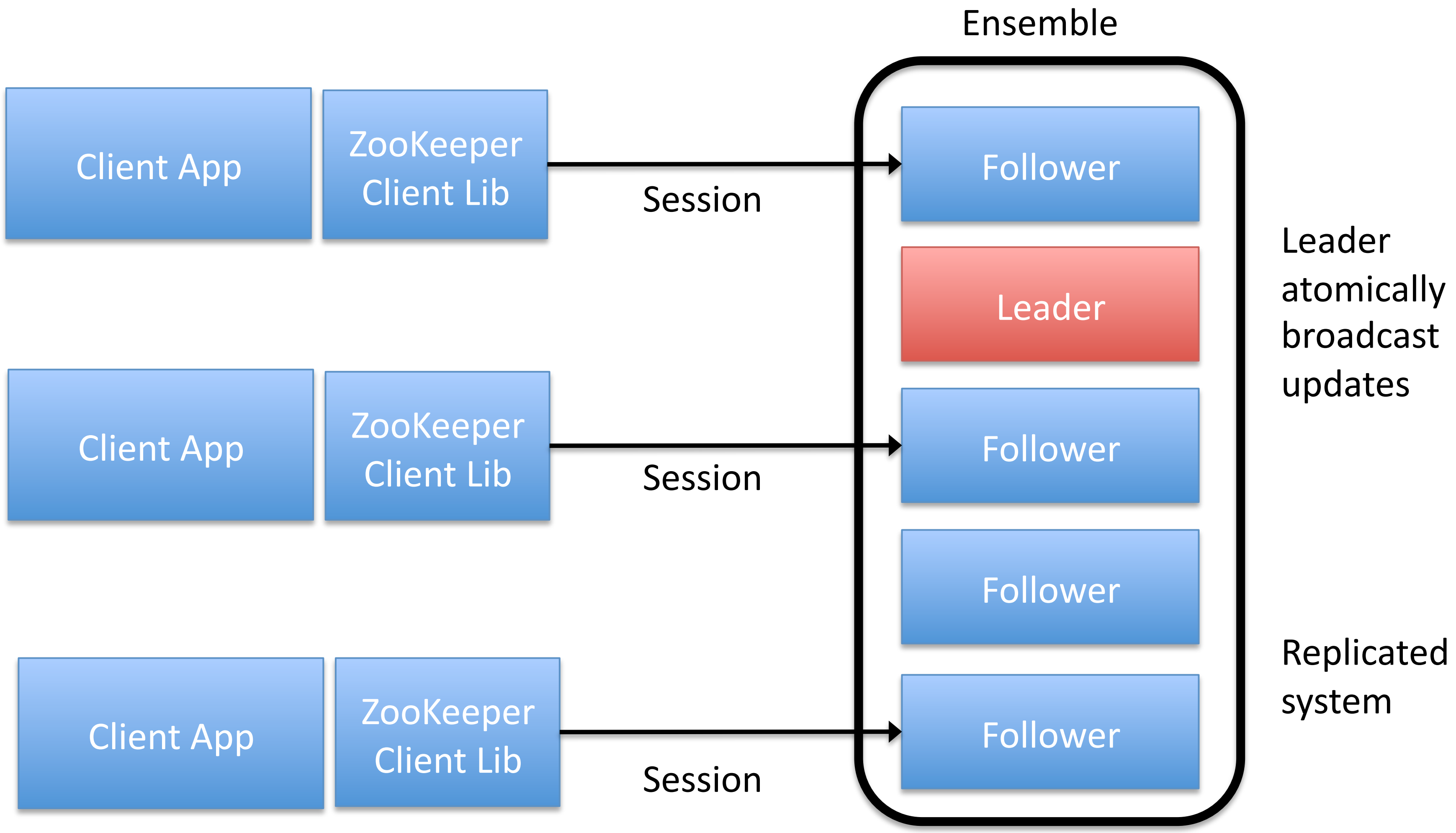
Part 2

The service

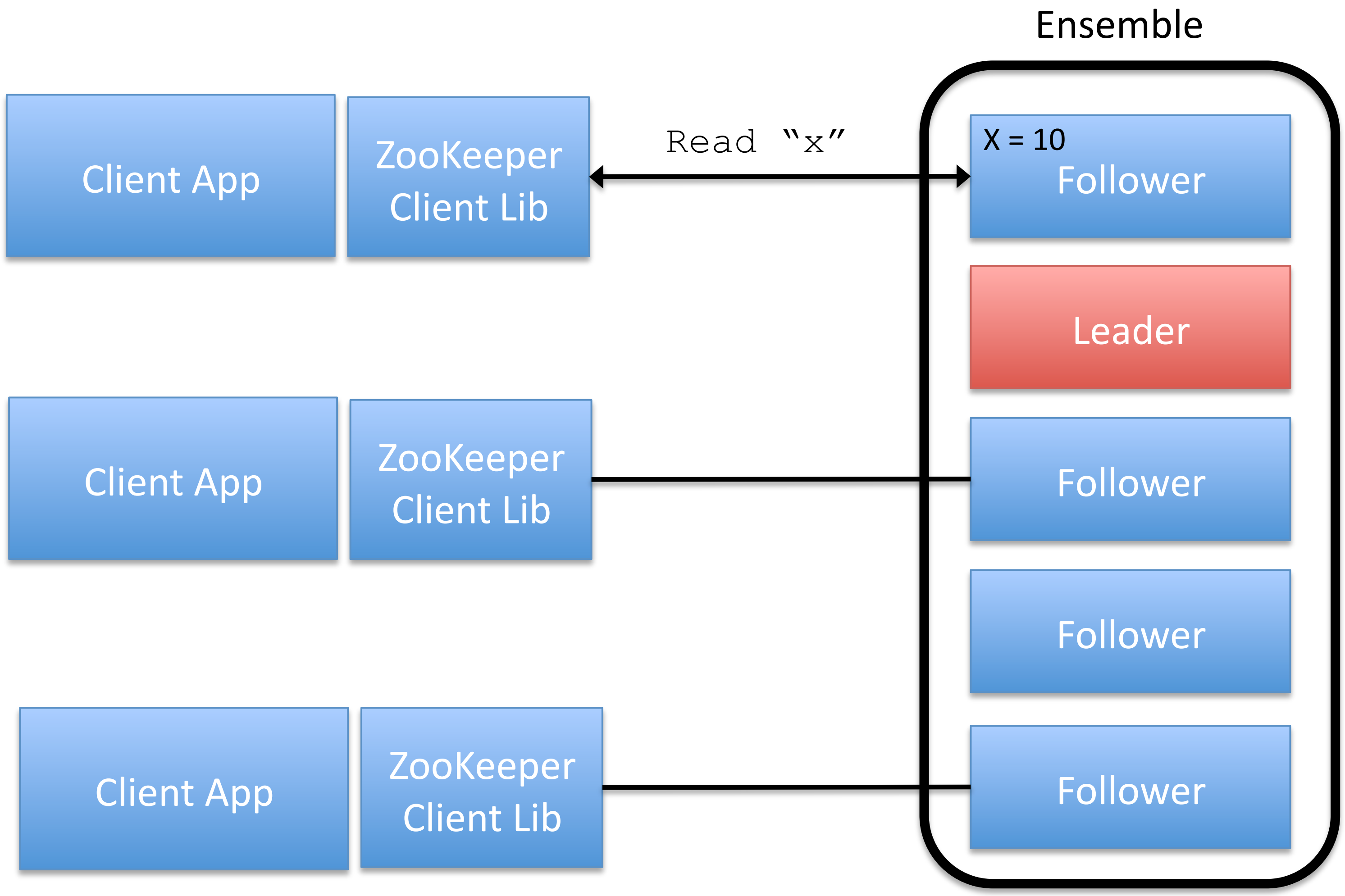
ZooKeeper Introduction

- Coordination kernel
  - Does not export concrete primitives
  - Recipes to implement primitives
- File system based API
  - Manipulate small data nodes: *znodes*

ZooKeeper: Overview

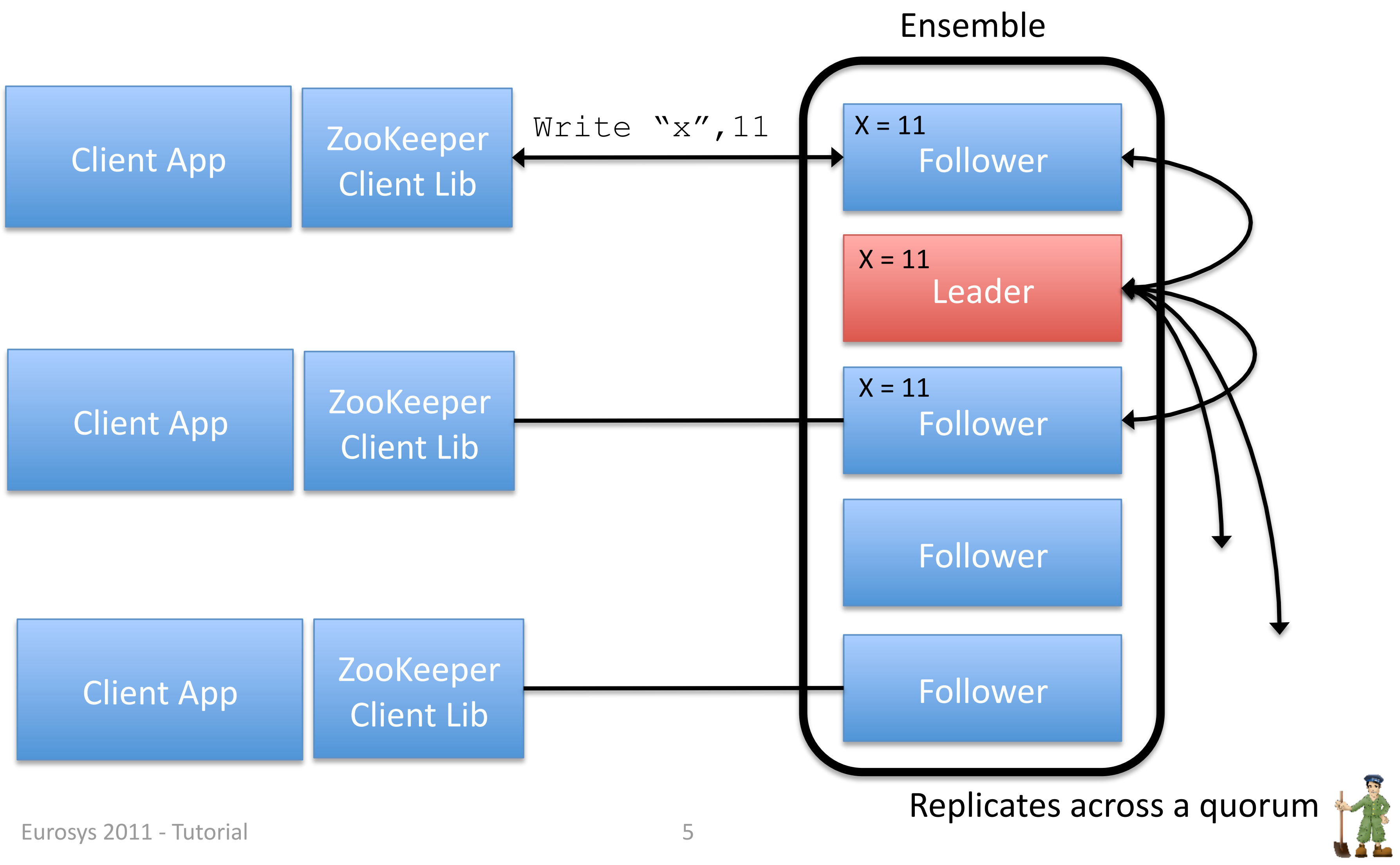


ZooKeeper: Read operations



Read operations processed locally

ZooKeeper: Write operations



ZooKeeper: Semantics of Sessions

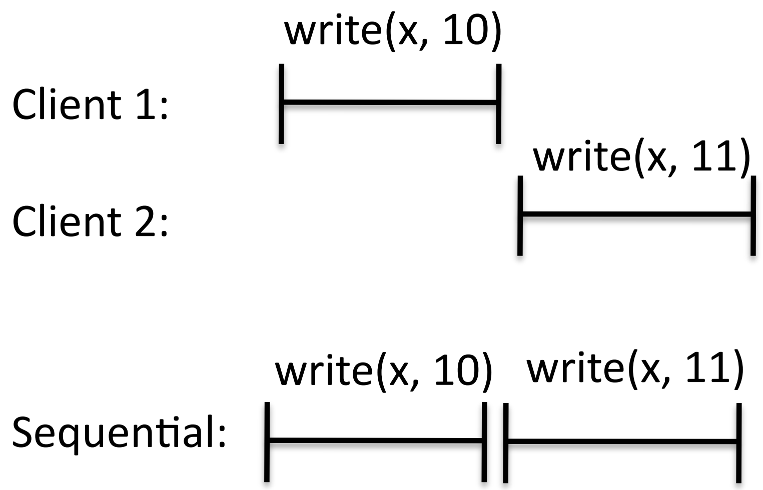
- A prefix of operations submitted through a session are executed
- Upon disconnection
  - Client lib tries to contact another server
  - Before session expires: connect to new server
  - Server must have seen a transaction id at least as large as the session

ZooKeeper: API

- Create znodes: `create`
  - Persistent, sequential, ephemeral
- Read and modify data: `setData, getData`
- Read the children of znode: `getChildren`
- Check if znode exists: `exists`
- Delete a znode: `delete`

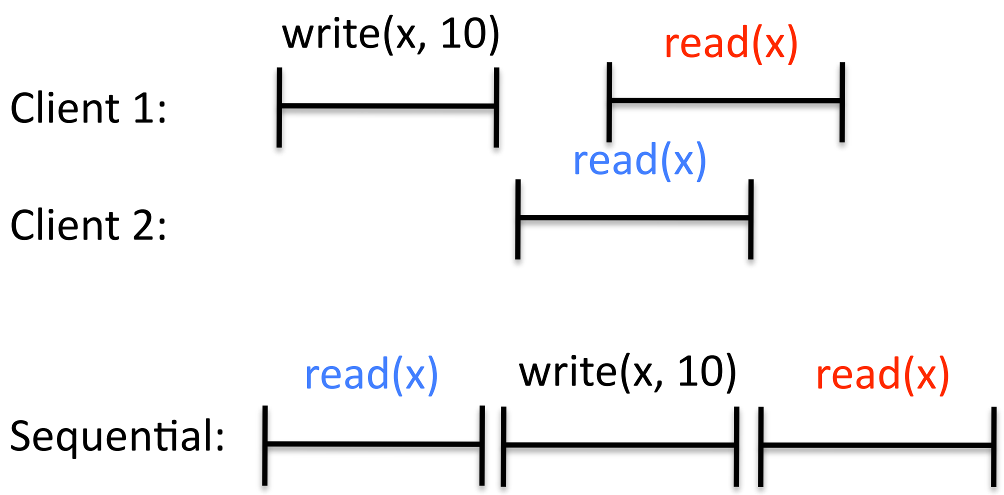
ZooKeeper: API

- Order
  - Updates: Totally ordered, linearizable
  - FIFO order for client operations
  - Read: sequentially ordered



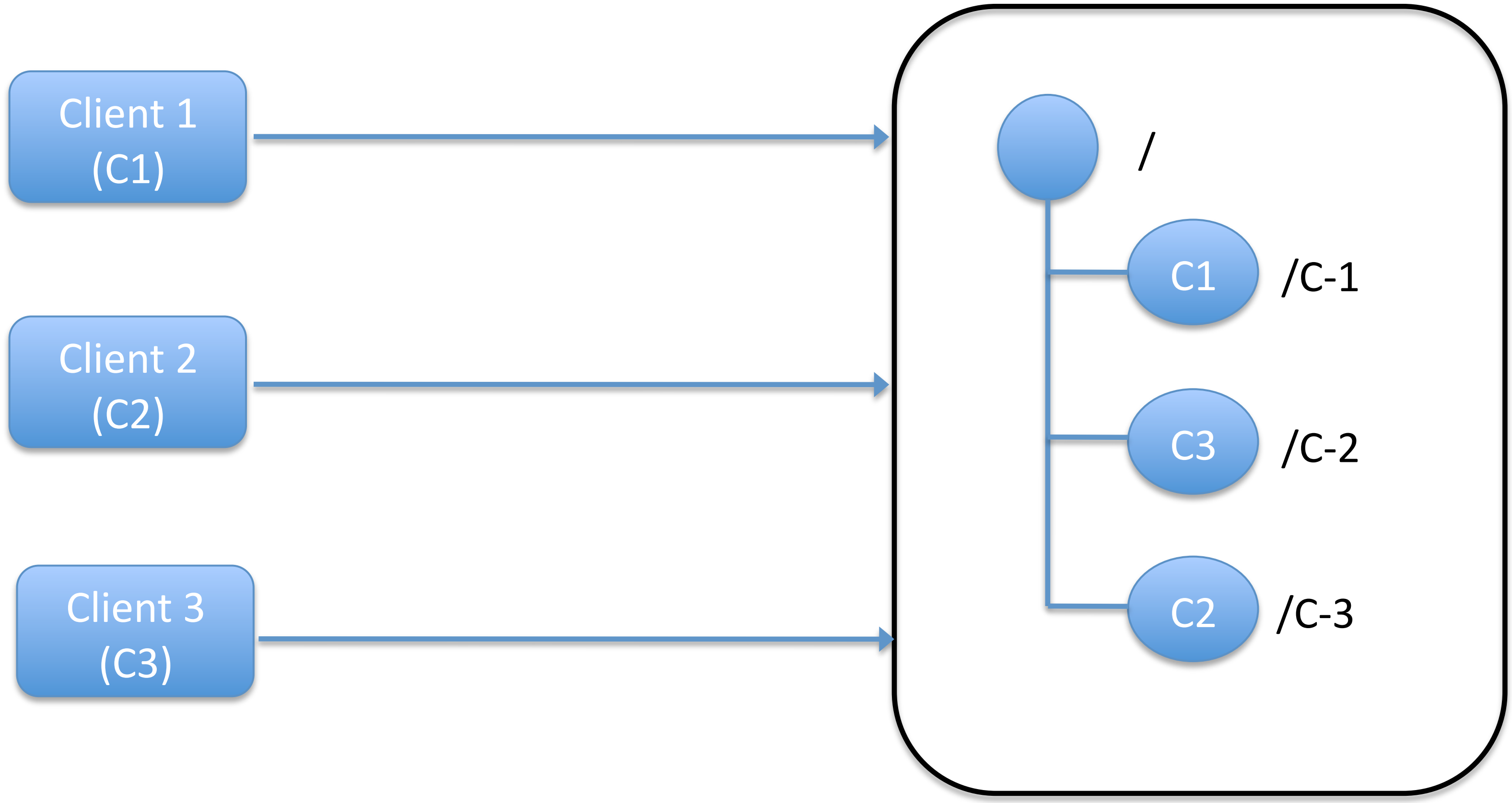
ZooKeeper: API

- Order
  - Updates: Totally ordered, linearizable
  - FIFO order for client operations
  - Read: sequentially ordered



ZooKeeper: Example

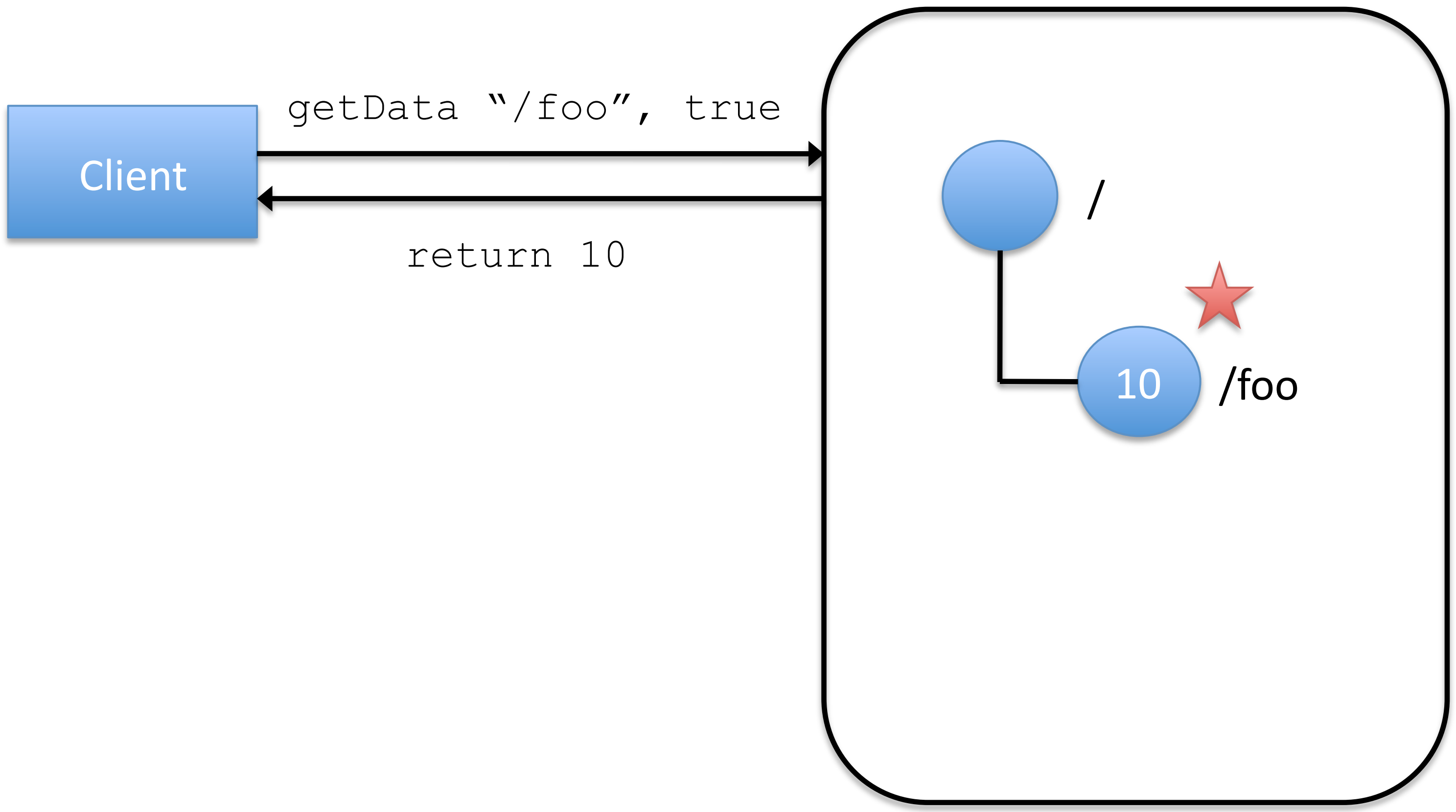
1- `create` "/C-", "C*i*", sequential, ephemeral
2- `getChildren` "/"
3- If not leader, `getData` "first node"



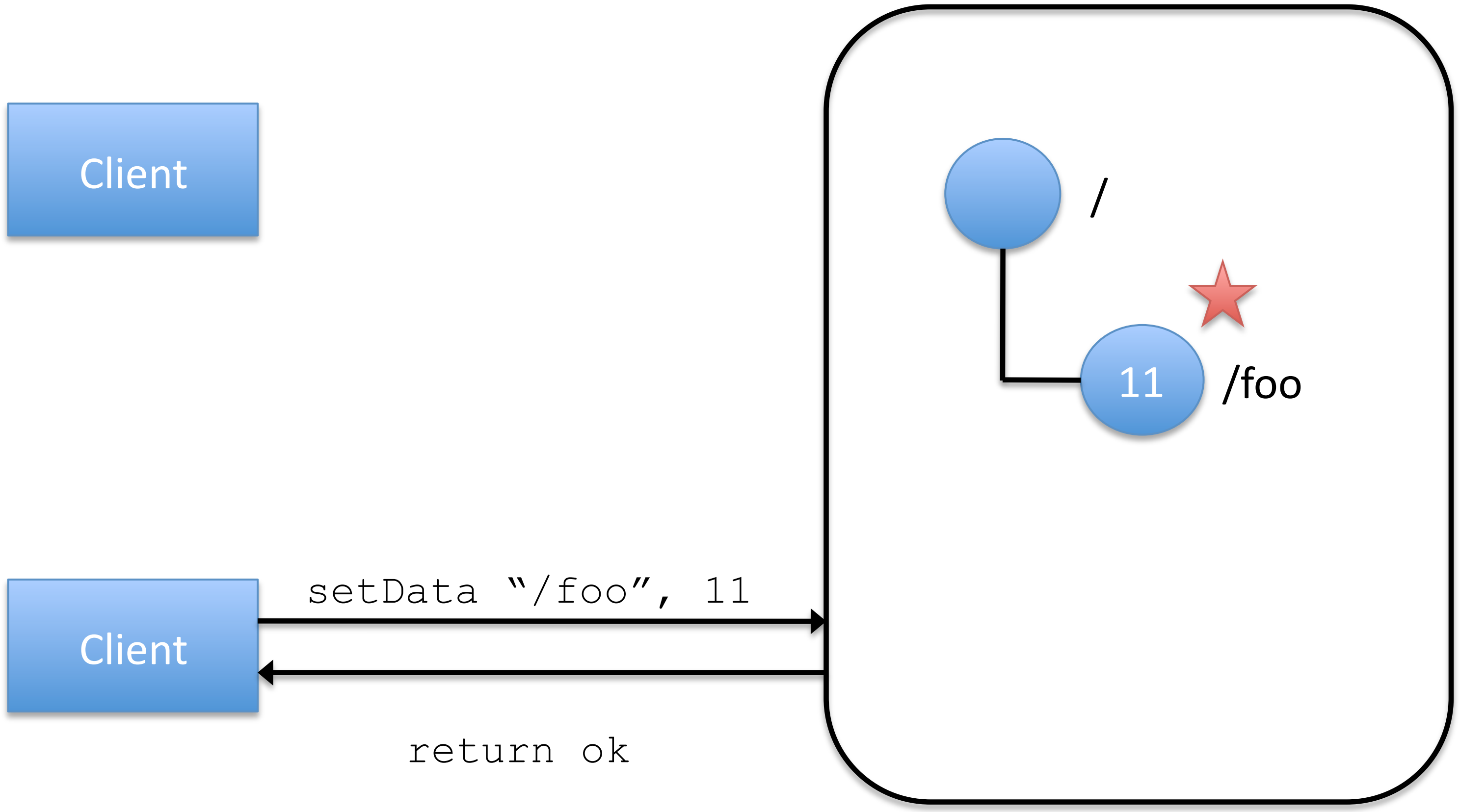
ZooKeeper: Znode changes

- Znode changes
  - Data is set
  - Node is created or deleted
  - *Etc…*
- To learn of znode changes
  - Set a *watch*
  - Upon change, client receives a *notification*
  - Notification ordered before new updates

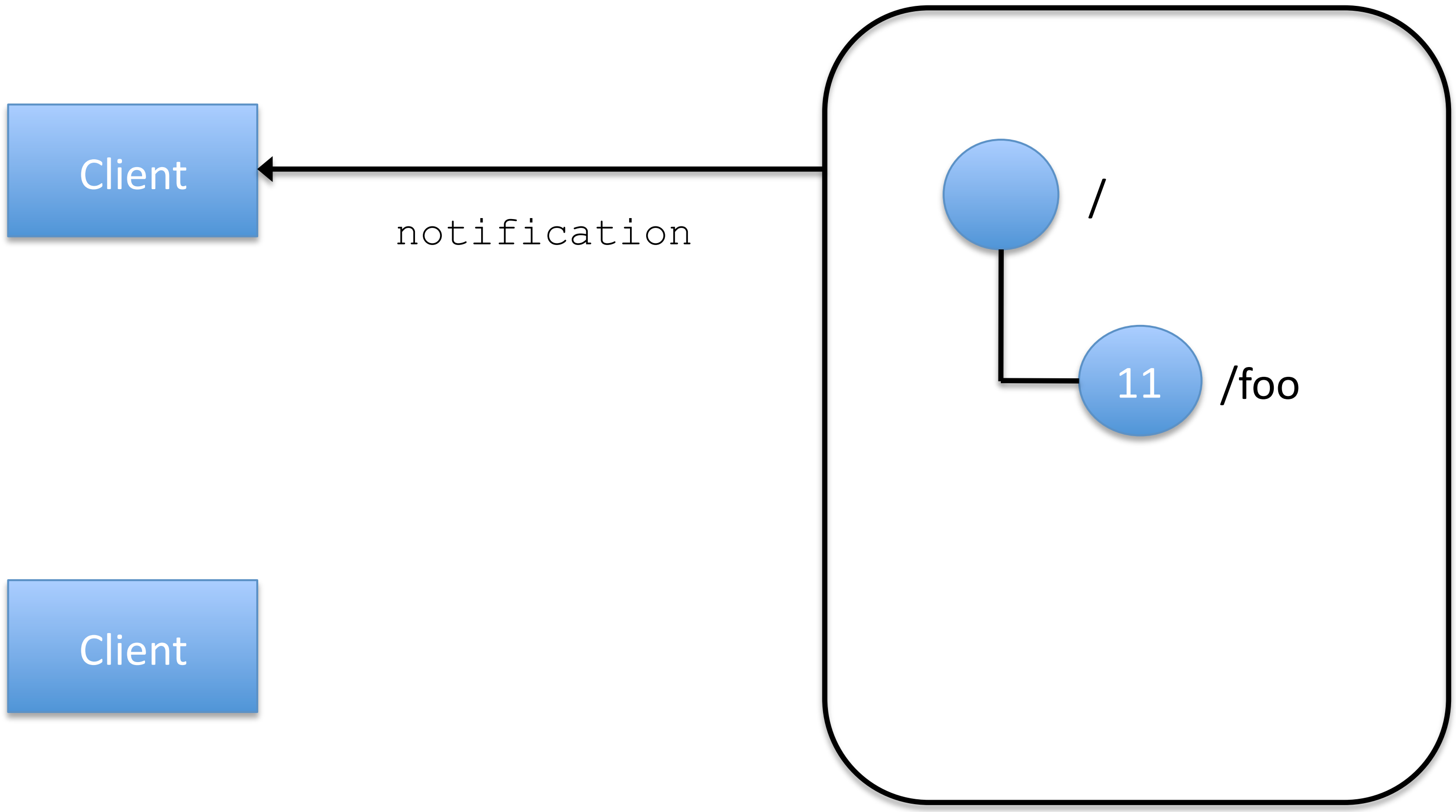
ZooKeeper: Watches



ZooKeeper: Watches



ZooKeeper: Watches

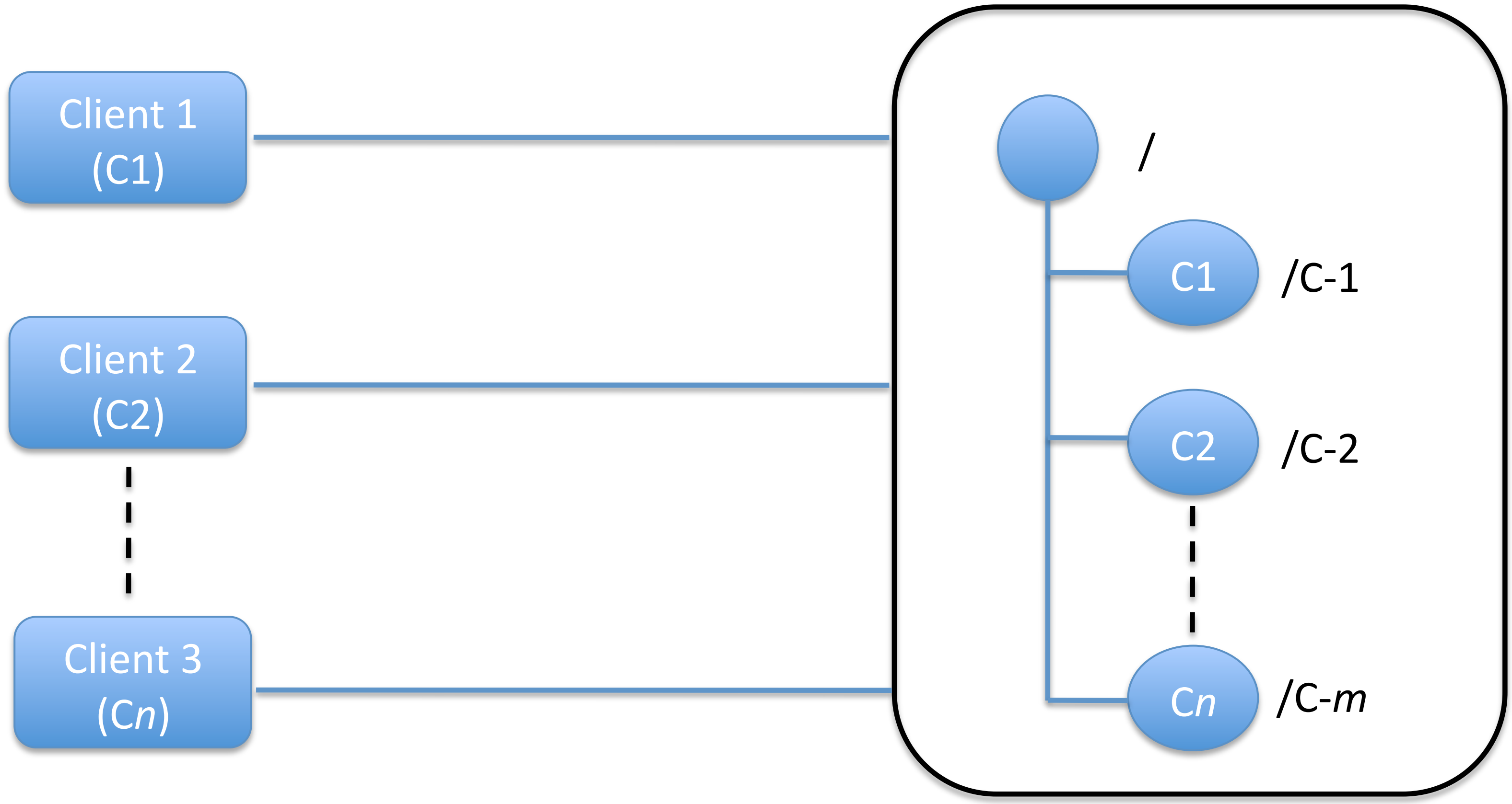


Watches, Locks, and the herd effect

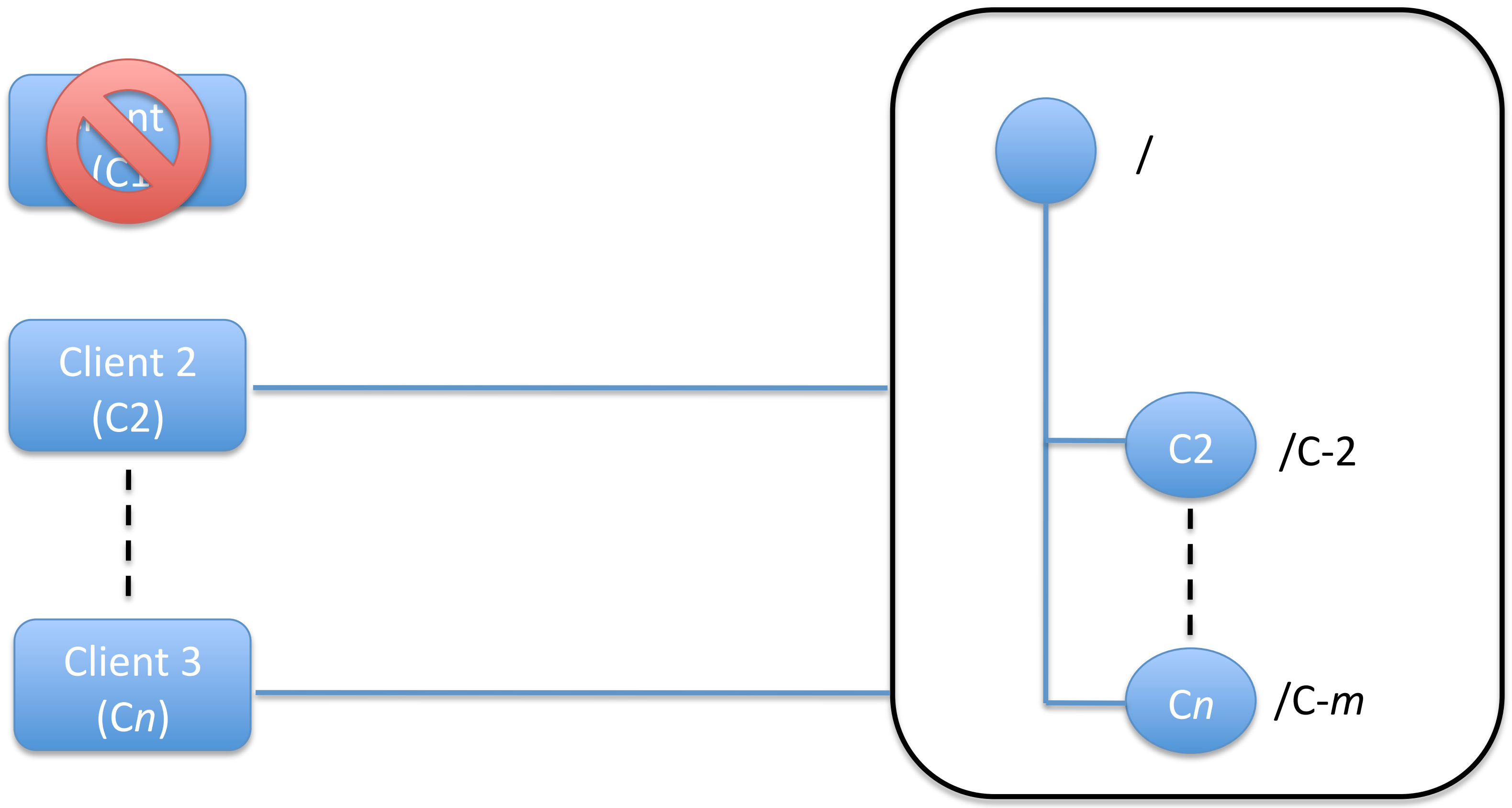
- Herd effect
  - Large number of clients wake up simultaneously

- Load spikes
  - Undesirable

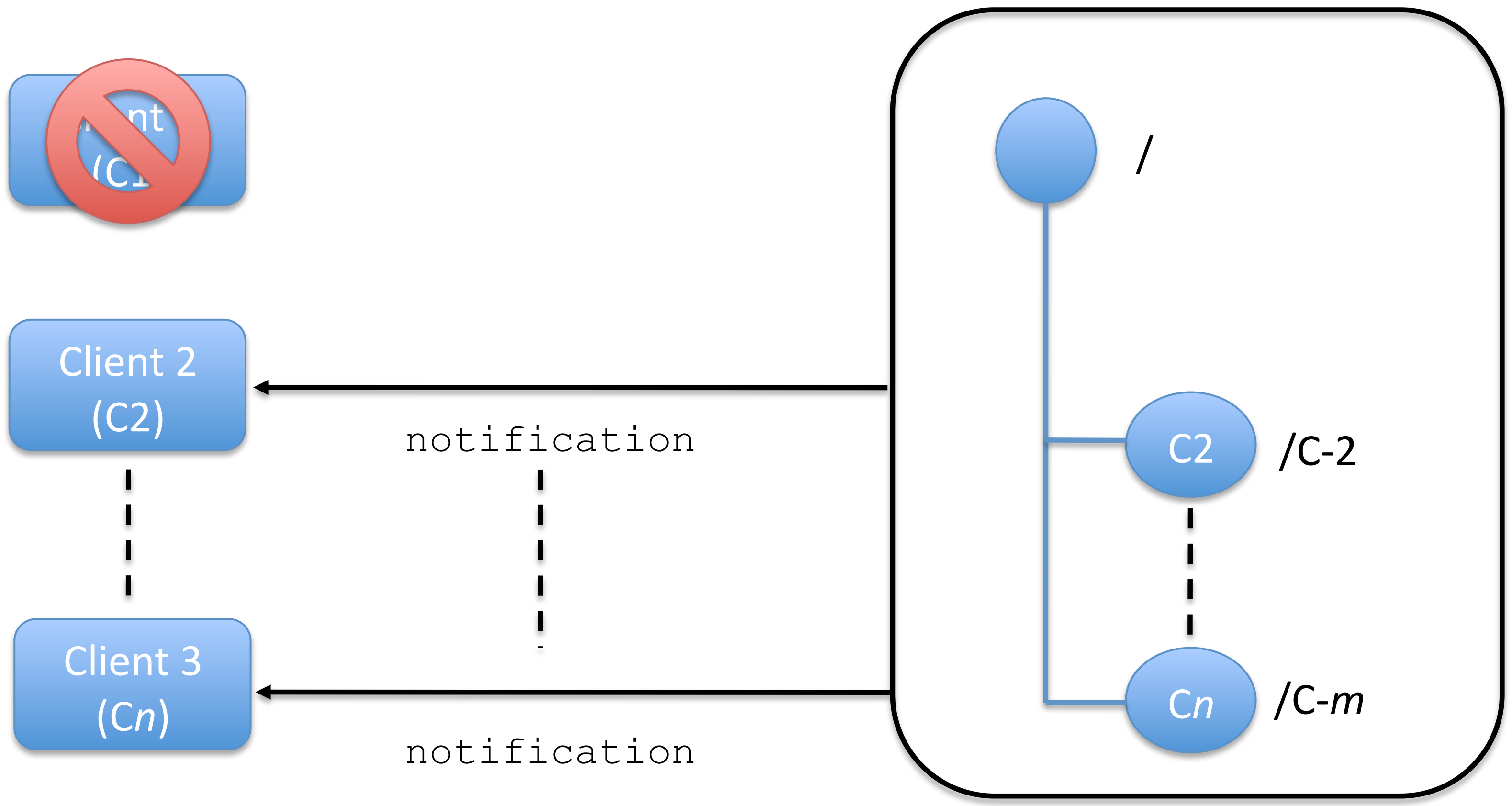
Watches, Locks, and the herd effect



Watches, Locks, and the herd effect



Watches, Locks, and the herd effect

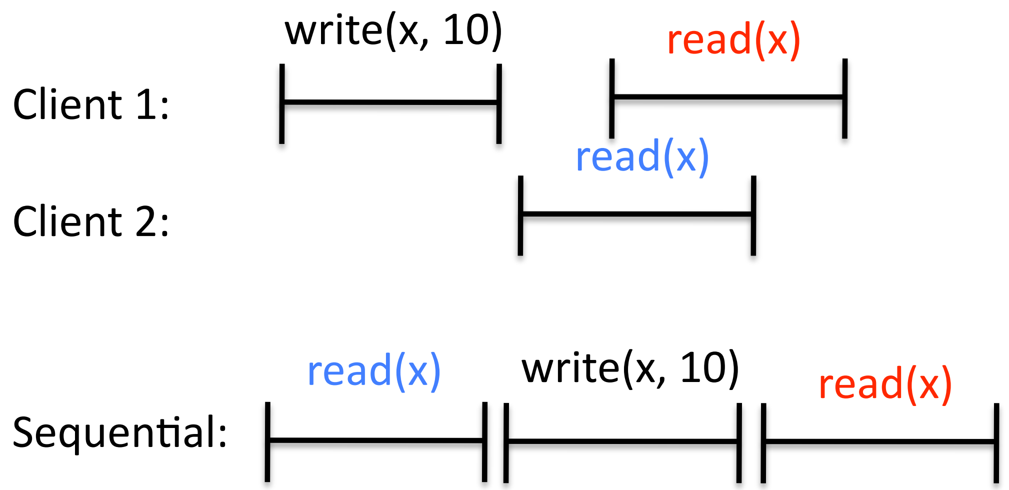


Watches, Locks, and the herd effect

- A solution
  - Use order of clients
  - Each client
    - Determines the znode *z* preceding its own znode in the sequential order
    - Watch *z*
  - A single notification is generated upon a crash
- Disadvantage for leader election
  - One client is notified of a leader change

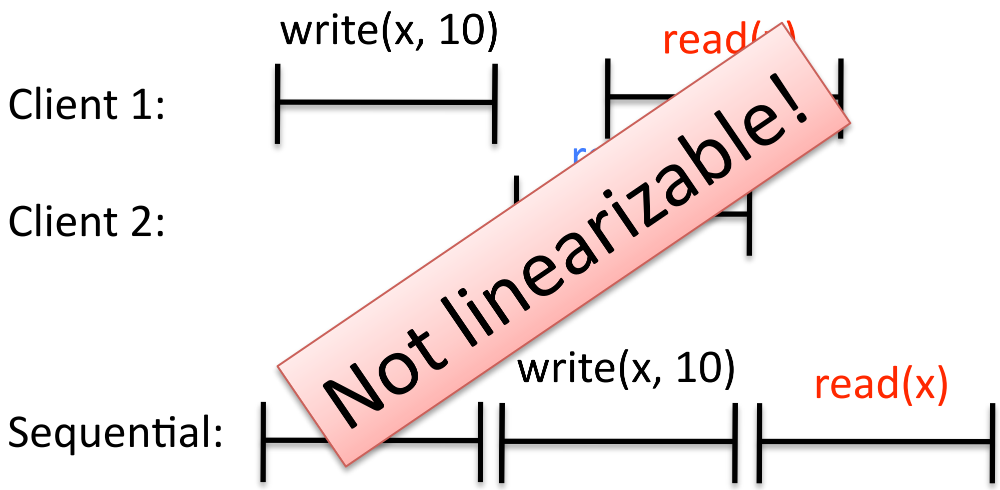
Linearizability

- Correctness condition
- Informal definition
  - Order of operations is equivalent to a sequential execution
  - Equivalent order satisfies real time precedence order



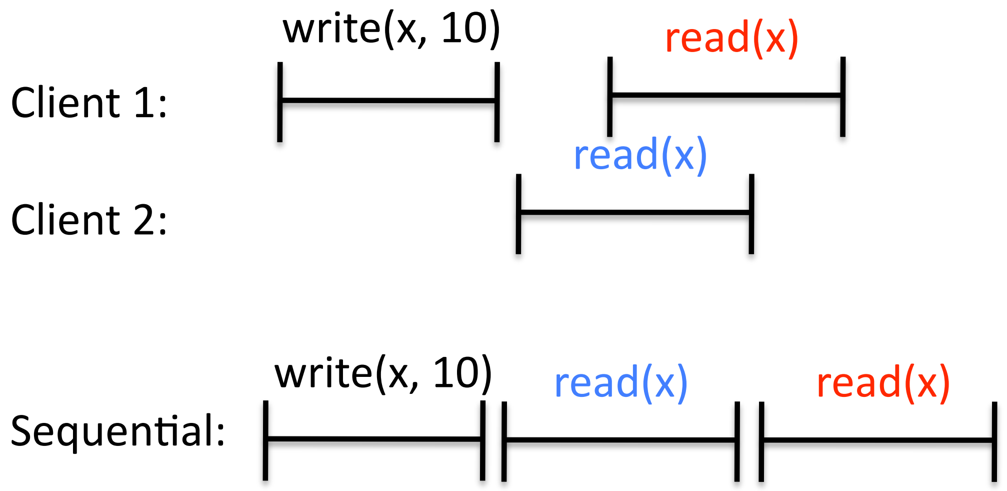
Linearizability

- Correctness condition
- Informal definition
  - Order of operations is equivalent to a sequential execution
  - Equivalent order satisfies real time precedence order



Linearizability

- Correctness condition
- Informal definition
  - Order of operations is equivalent to a sequential execution
  - Equivalent order satisfies real time precedence order



Linearizability

- Is it important? It depends…
- Implements universal object
  - Herlihy's result
  - Implement consensus for *n* processes

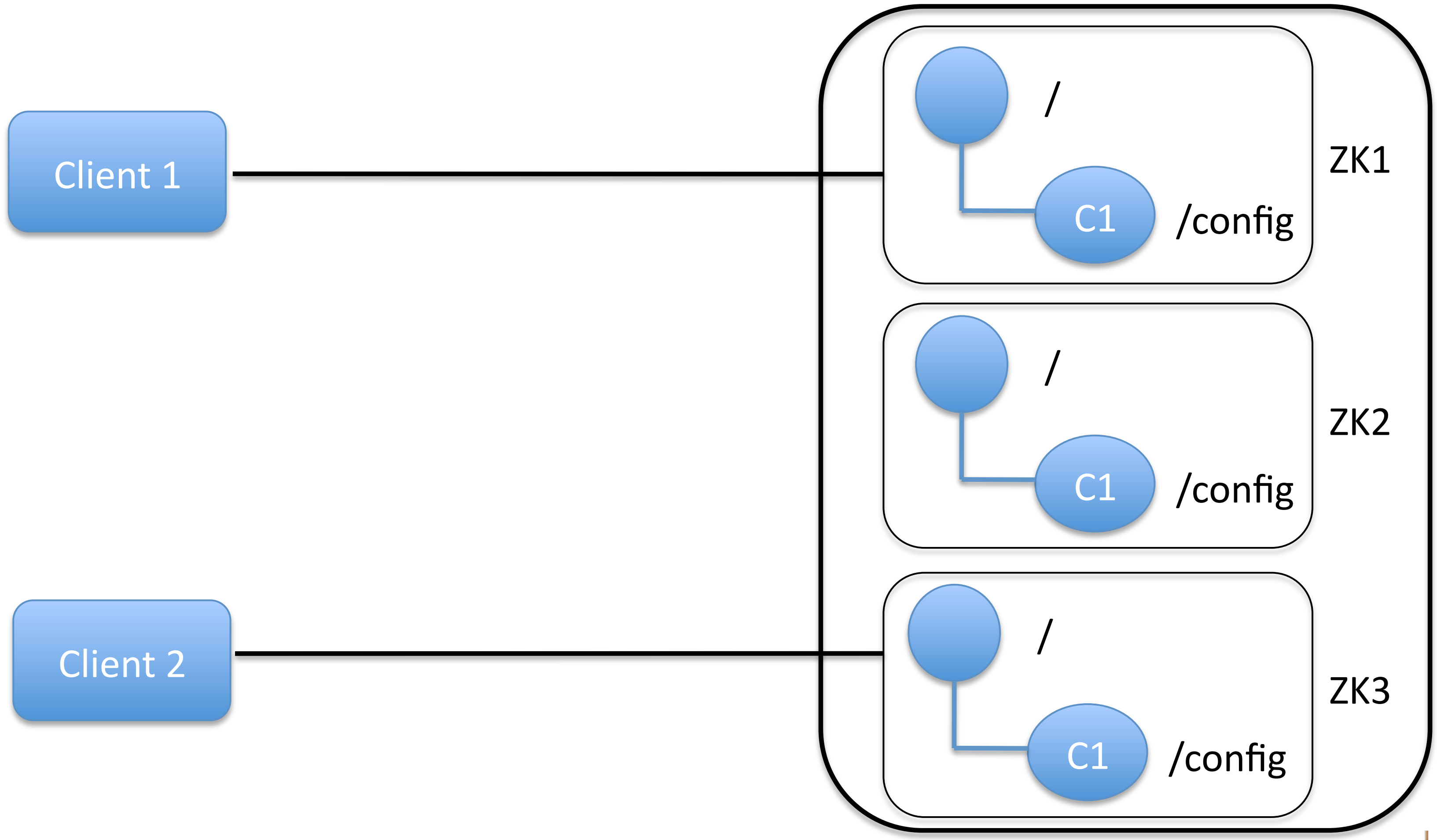
Implementing consensus

- Each process *p* proposes then decides
- `Propose(v)`
  - `setData “/c/proposal-”, “v”, sequential`
- `Decide()`
  - `getChildren “/c”`
  - Select znode `z` with smallest sequence number
  - `v’ = getData “/c/z”`
  - Decide upon `v’`

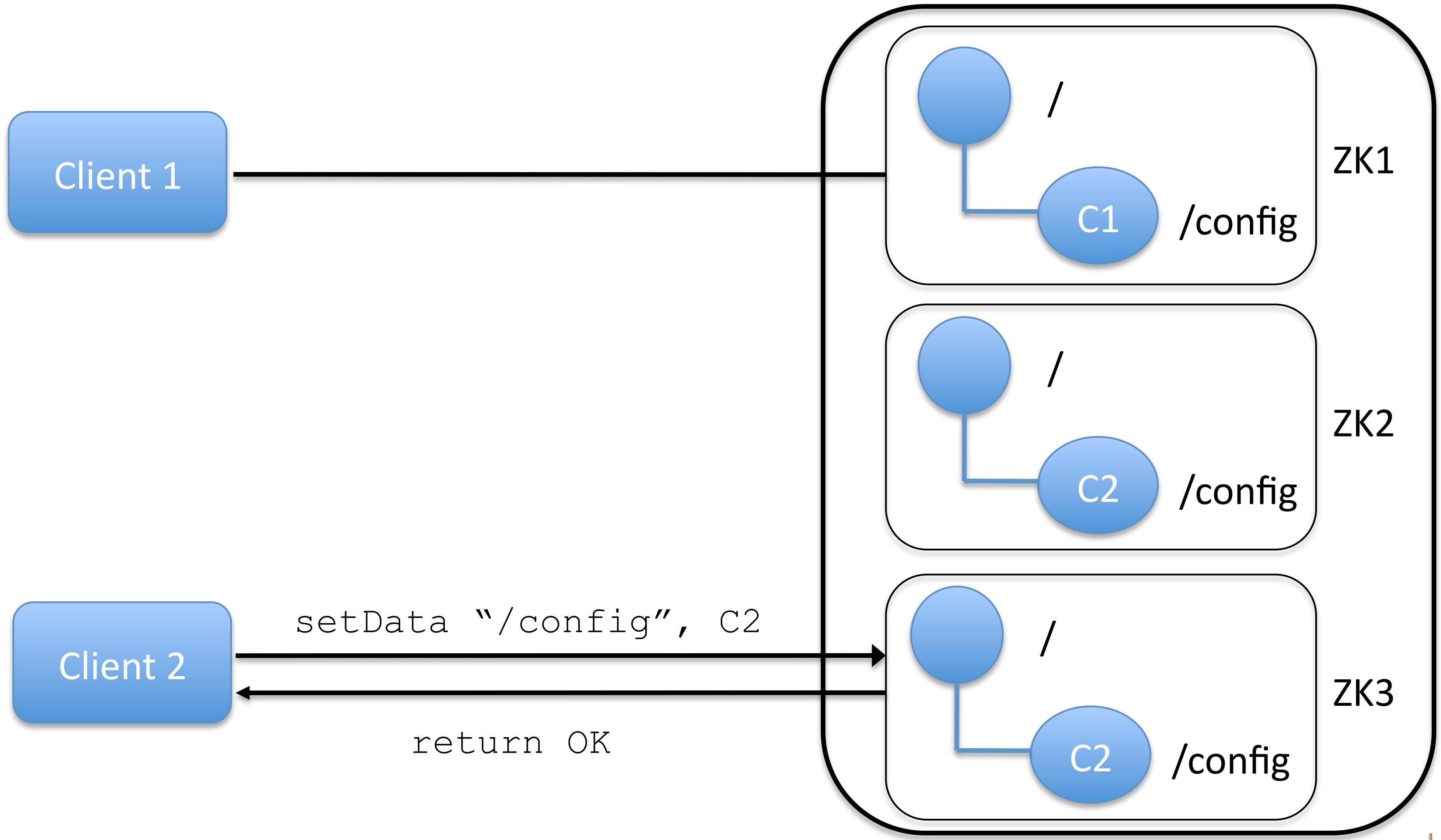
Linearizability

- Is it important? It depends…
- Implements universal object
  - Herlihy’s result
  - Implement consensus for *n* processes
  - … but it is affected by hidden channels

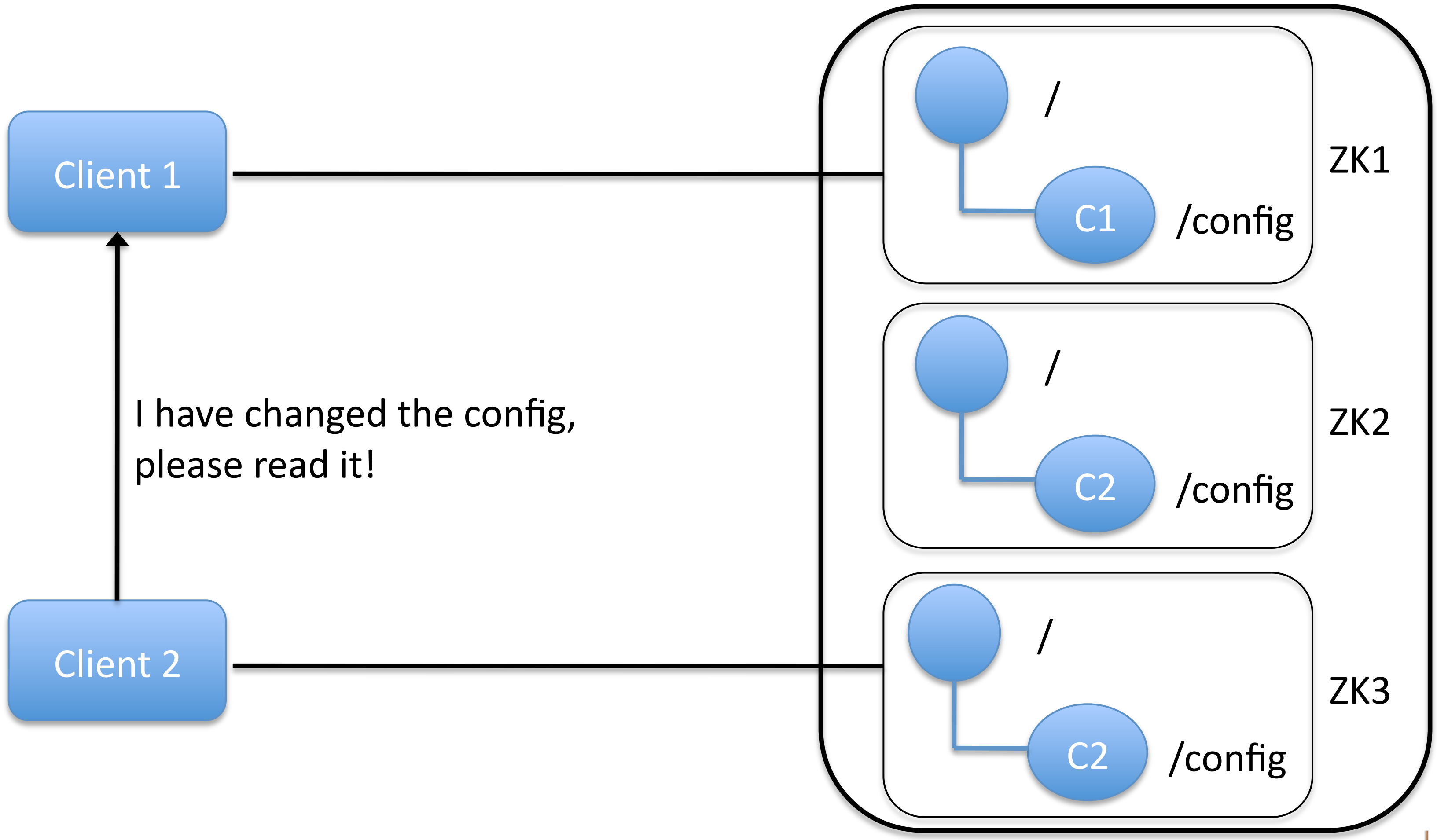
Hidden channels



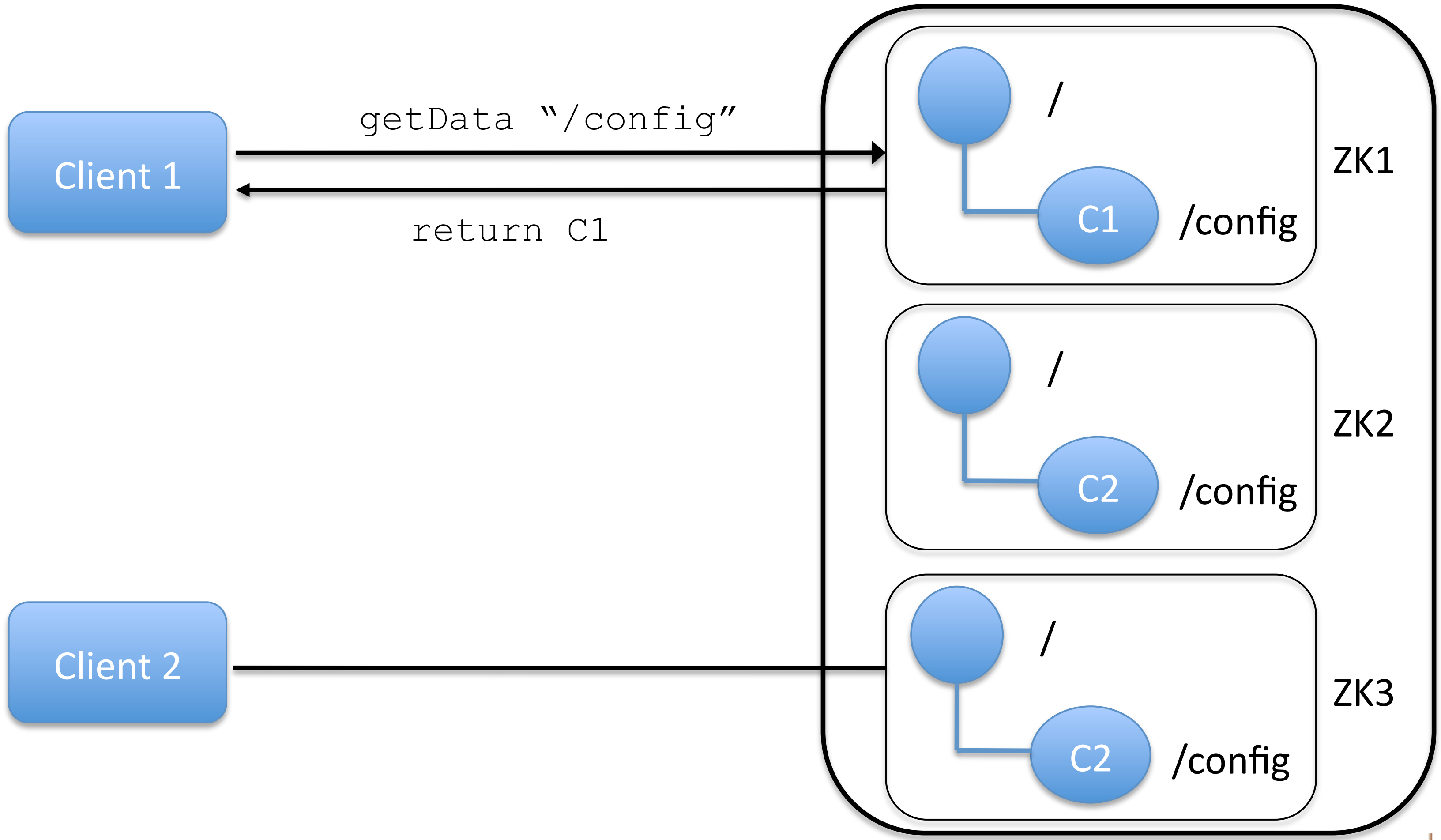
Hidden channels



Hidden channels

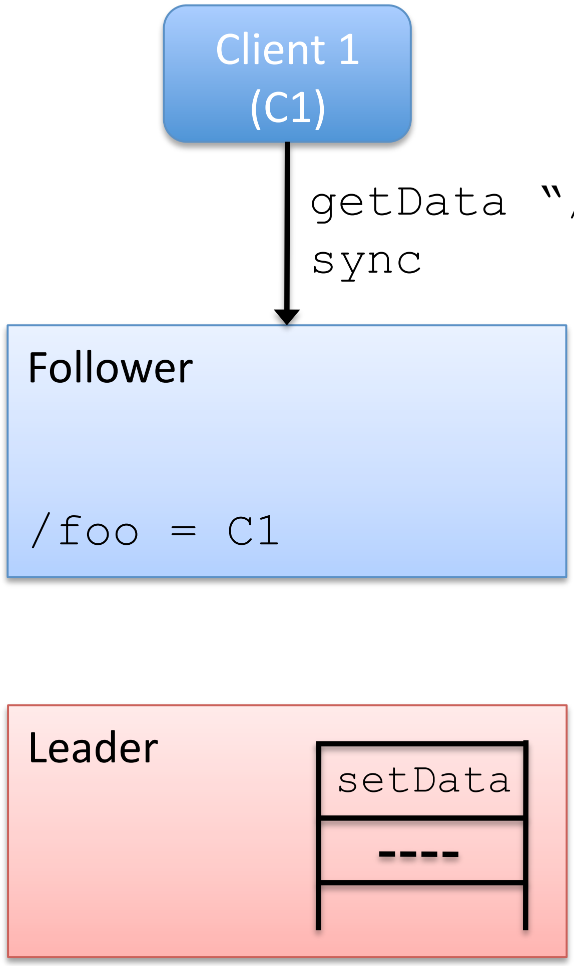


Hidden channels



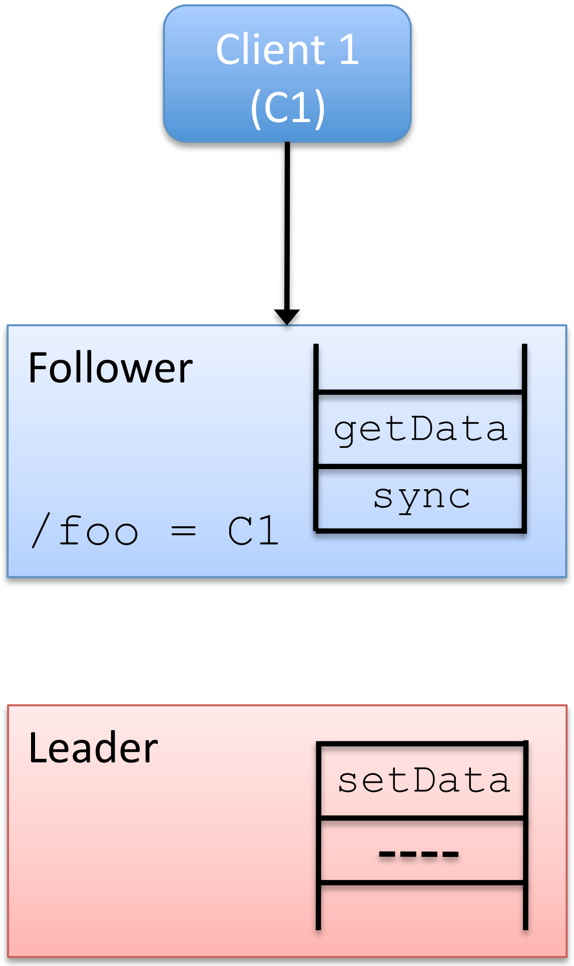
A hat trick…

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



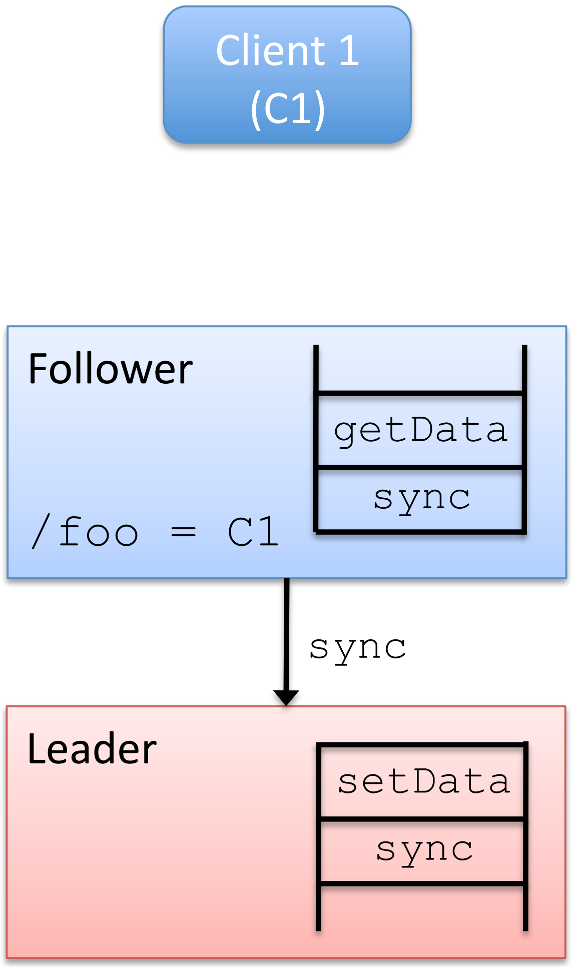
A hat trick…

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



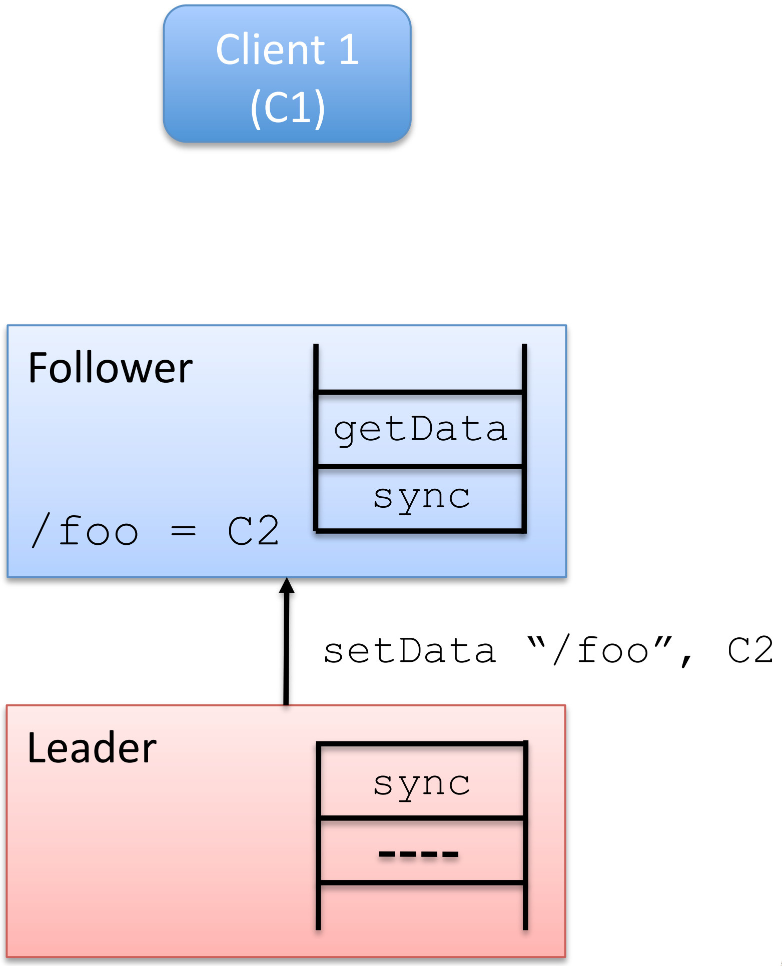
A hat trick…

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



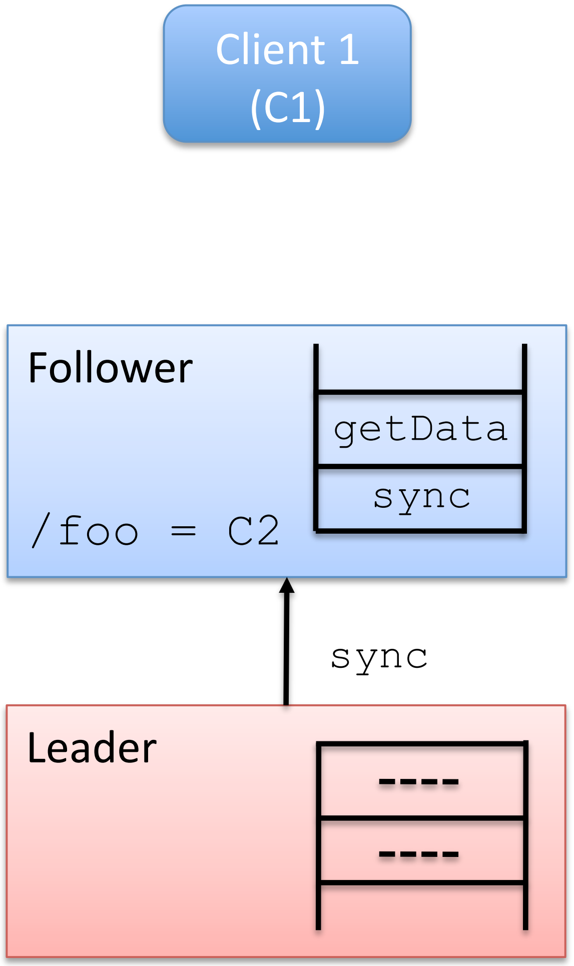
A hat trick…

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



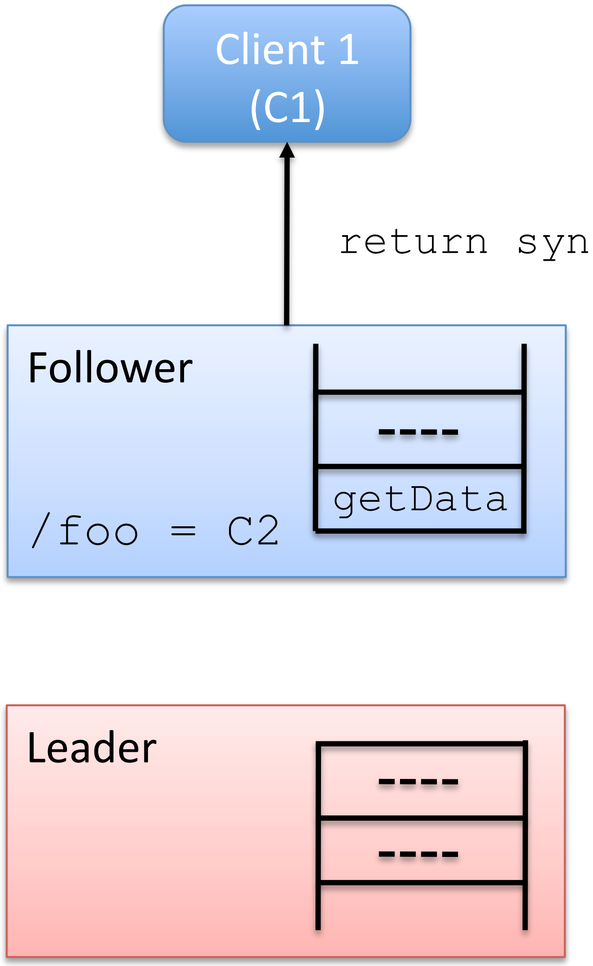
A hat trick...

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



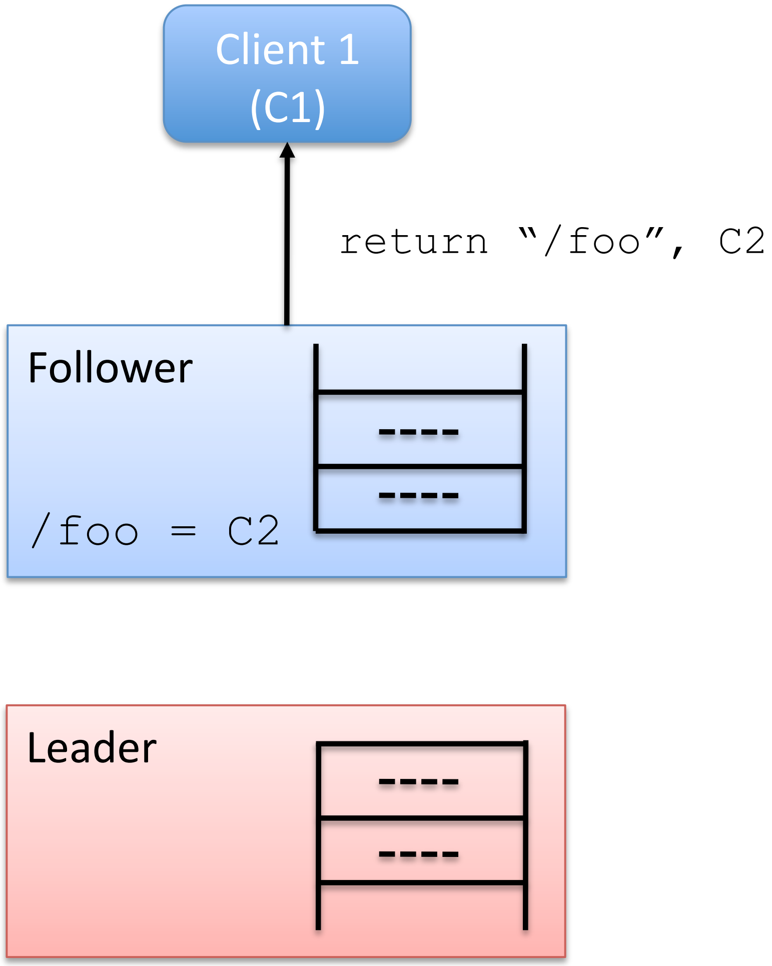
A hat trick…

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



A hat trick...

- `sync`
  - Asynchronous operation
  - Before read operations
  - Flushes the channel between follower and leader
  - Makes operations linearizable



Summary of Part 2

- ZooKeeper
  - Replicated service
  - Propagate updates with a broadcast protocol
- Updates use consensus
- Reads served locally
- Workload not linearizable because of reads
- `sync()` makes it linearizable