ZooKeeper Tutorial

Flavio Junqueira
Benjamin Reed

Yahoo! Research

hCps://cwiki.apache.org/confluence/display/ZOOKEEPER/EurosysTutorial

Used for 14-848 Discussion, 11/6/2017 by Gregory Kesden

Plan for today

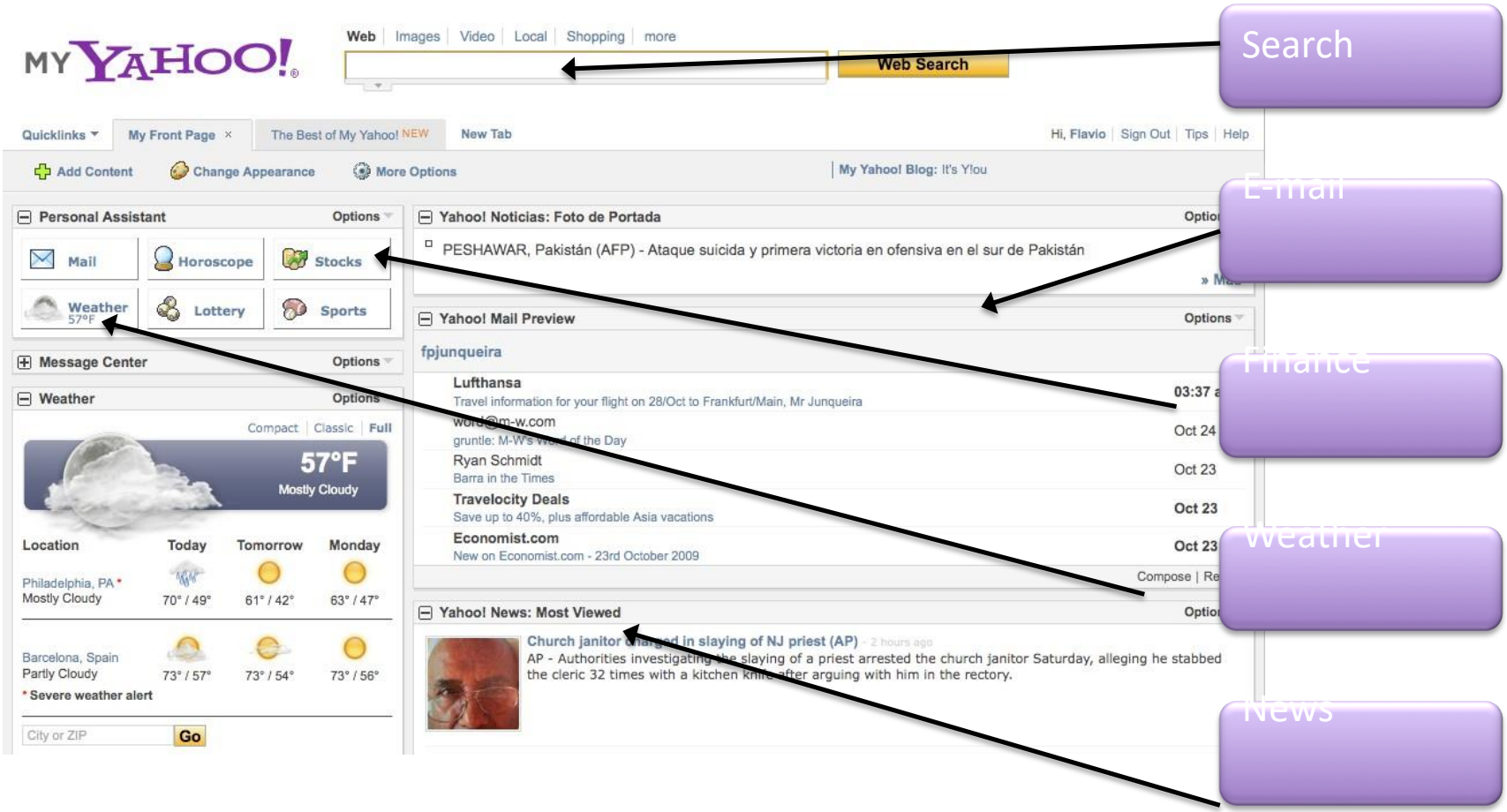
- First half
  - Part 1
    - Motivation and background
  - Part 2
    - How ZooKeeper works on paper
- Second half
  - Part 3
    - Share some practical experience
    - Programming exercises
  - Part 4
    - Some caveats
    - Wrap up

ZooKeeper Tutorial

Part 1

Fundamentals

Yahoo! Portal



Yahoo!: Workload generated

- Home page
  - 38 million users a day (USA)
  - 2.5 billion users a month (USA)
- Web search
  - 3 billion queries a month
- E-mail
  - 90 million actual users
  - 10 min/visit

Yahoo! Infrastructure

- Lots of servers
- Lots of processes
- High volumes of data
- Highly complex soaware systems
- … and developers are mere mortals

Yahoo! Lockport Data Center

Coordination is important

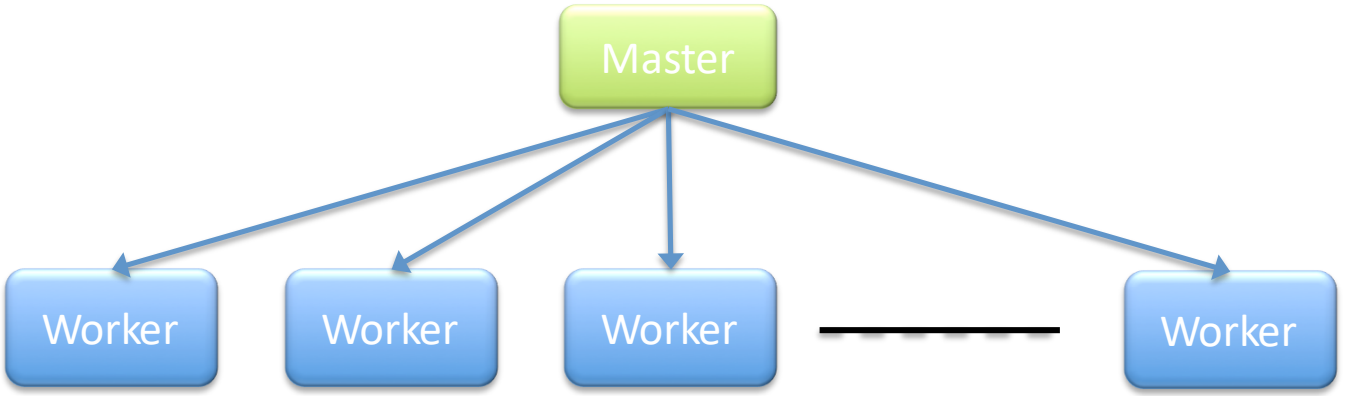
Coordination primitives

- Semaphores
- Queues
- Leader election
- Group membership
- Barriers
- Configuration

Even small is hard…

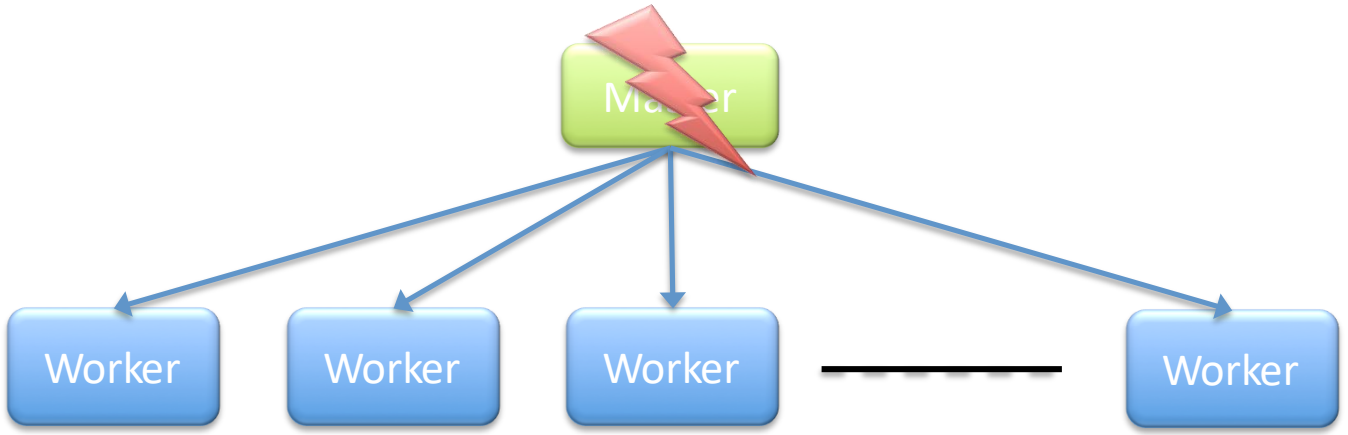
A simple model

- Work assignment
  - Master assigns work
  - Workers execute tasks assigned by master



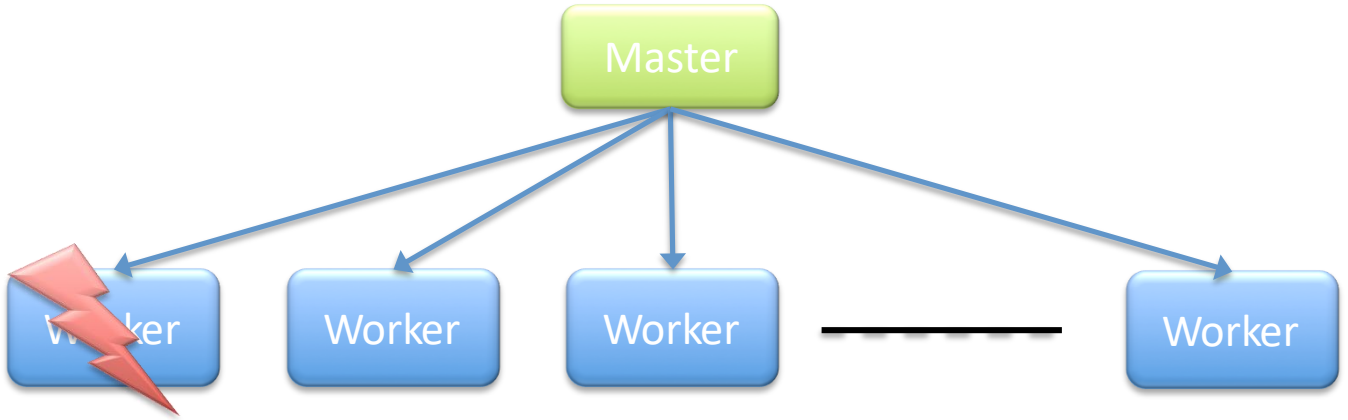
Master crashes

- Single point of failure
- No work is assigned
- Need to select a new master



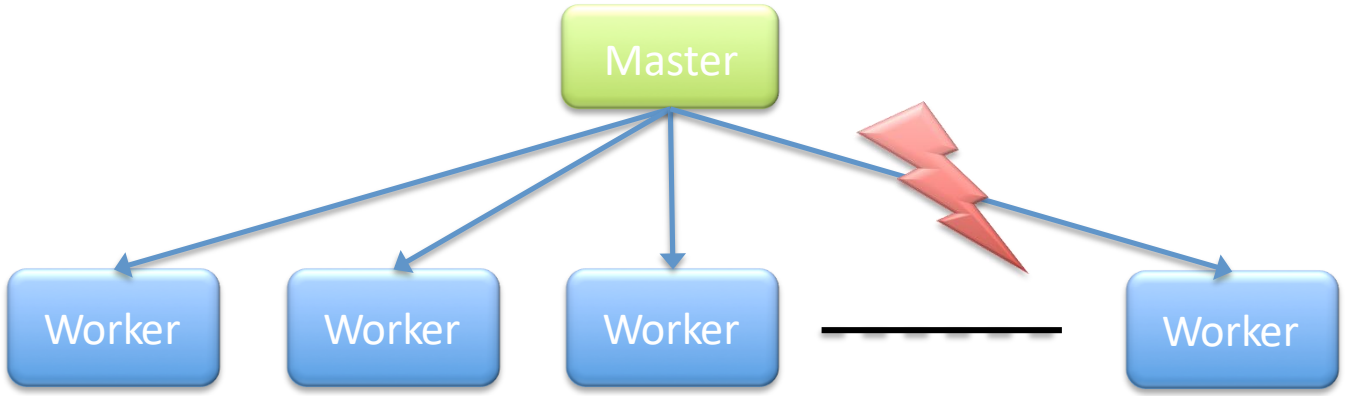
Worker crashes

- Not as bad… Overall system still works
  - Does not work if there are dependencies
- Some tasks will never be executed
- Need to detect crashed workers

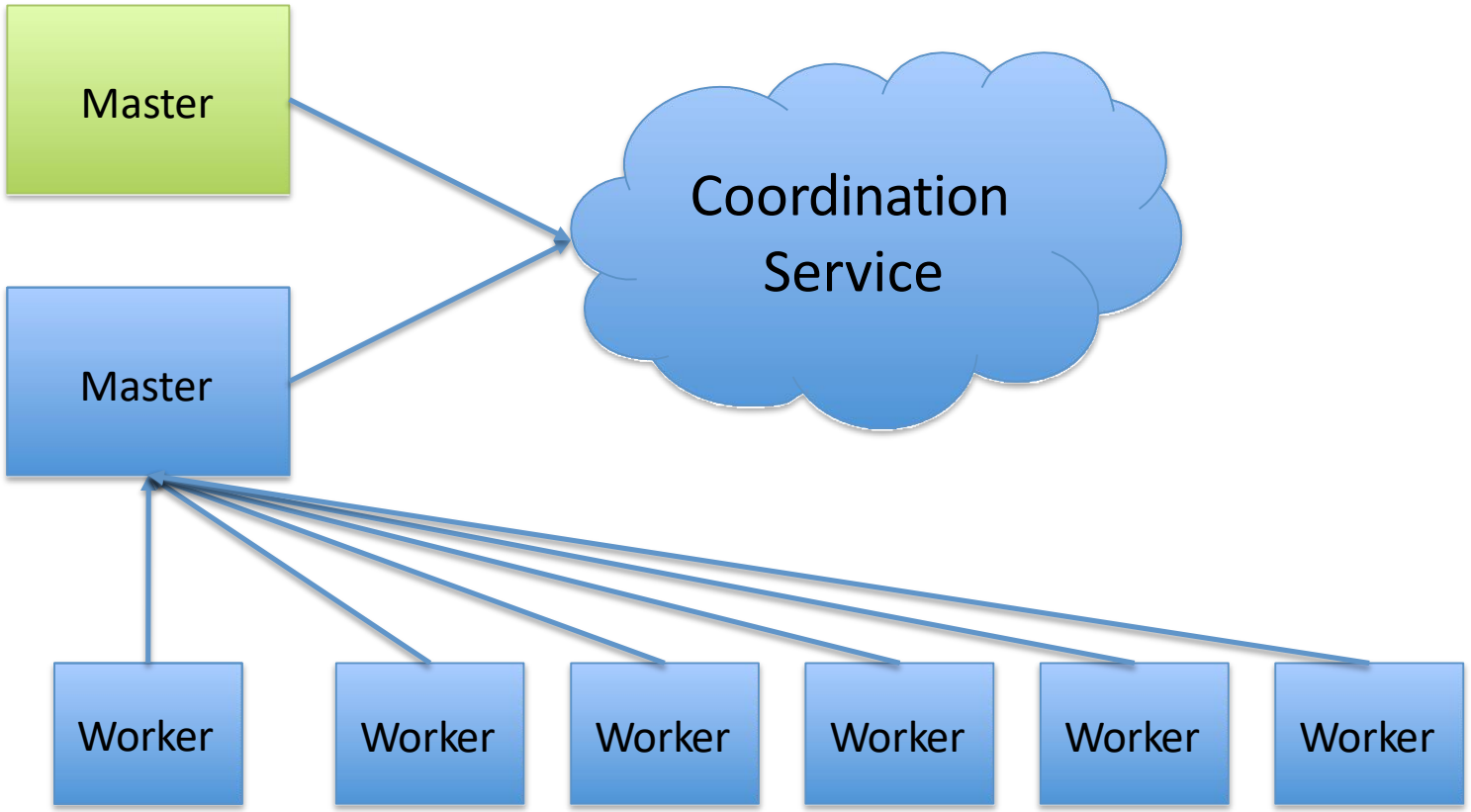


Worker does not receive assignment

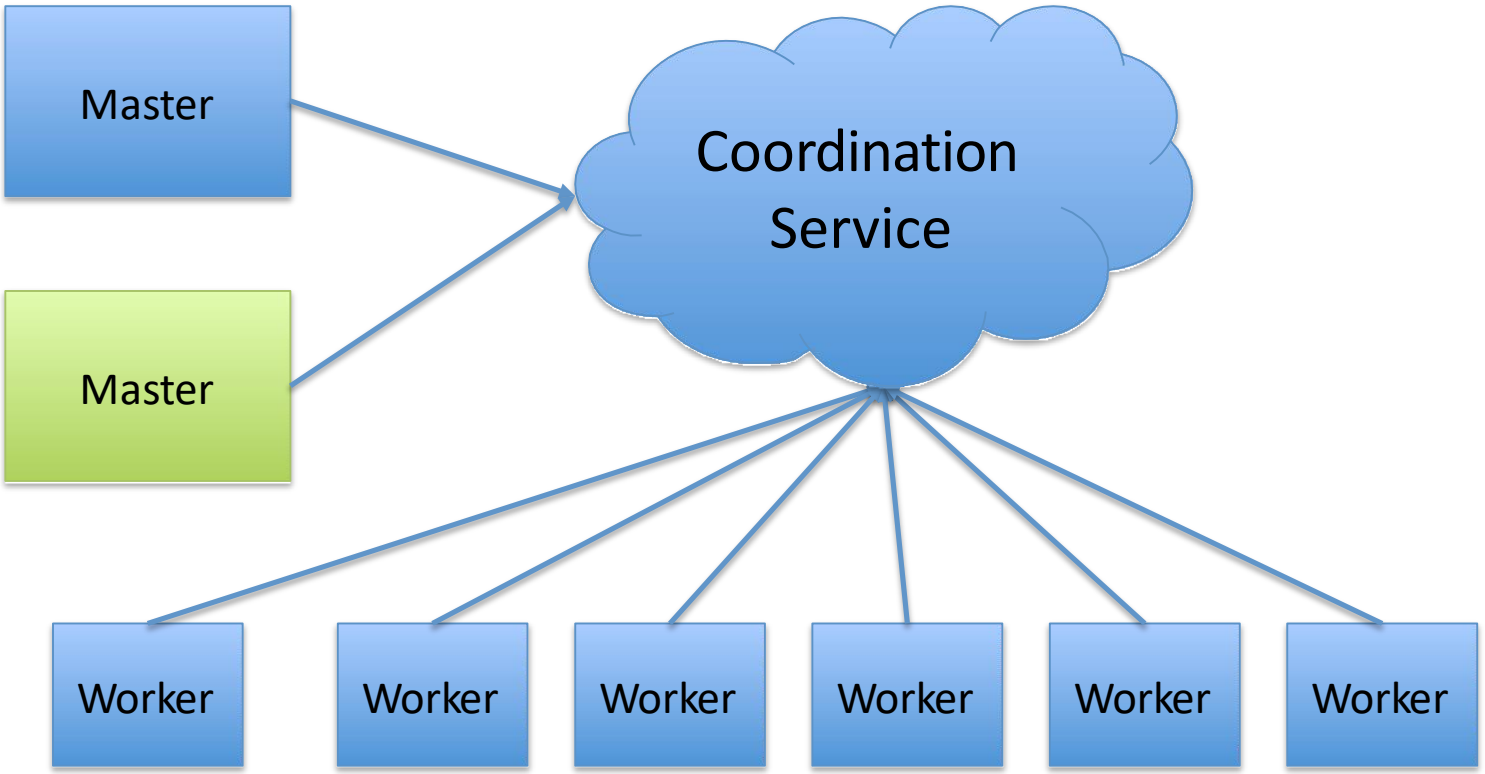
- Same problem as before
- Some tasks may not be executed
- Need to guarantee that worker receives assignment



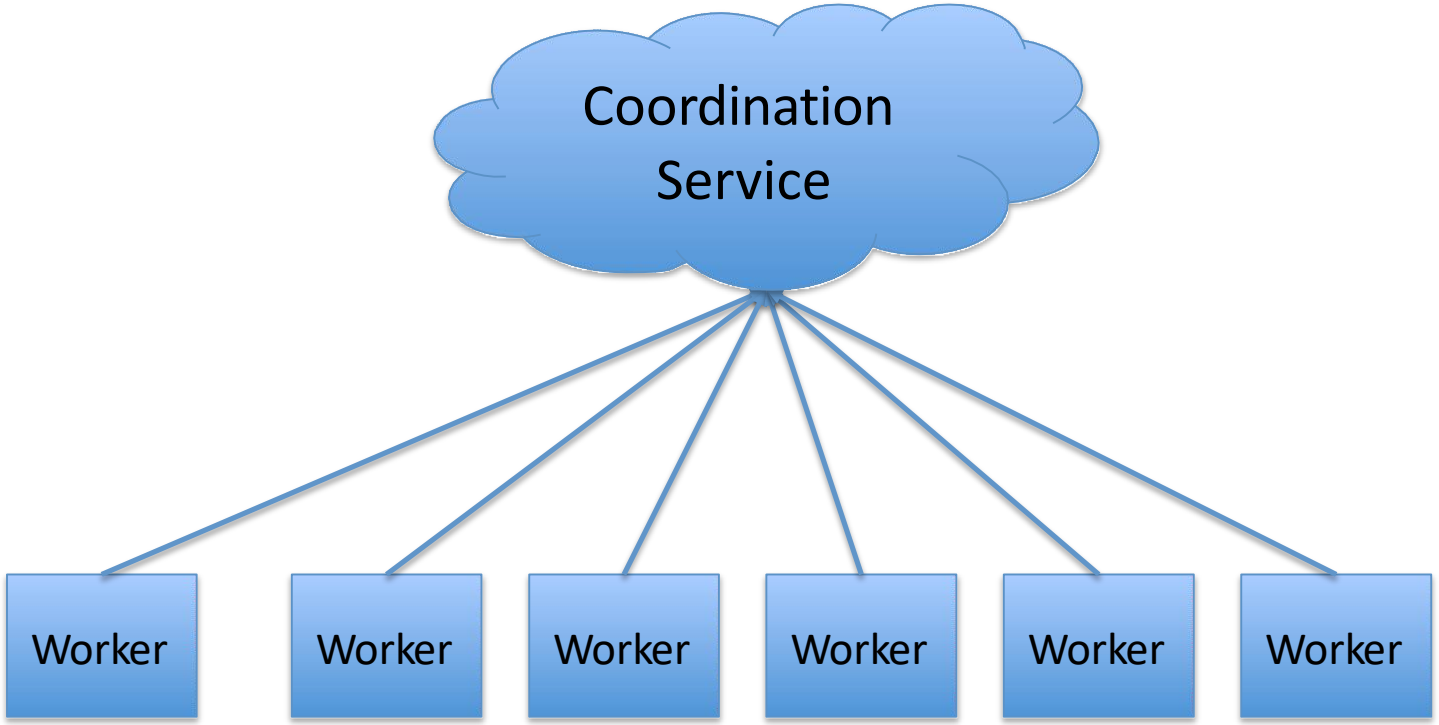
Fault-tolerant distributed system



Fault-tolerant distributed system



Fully distributed

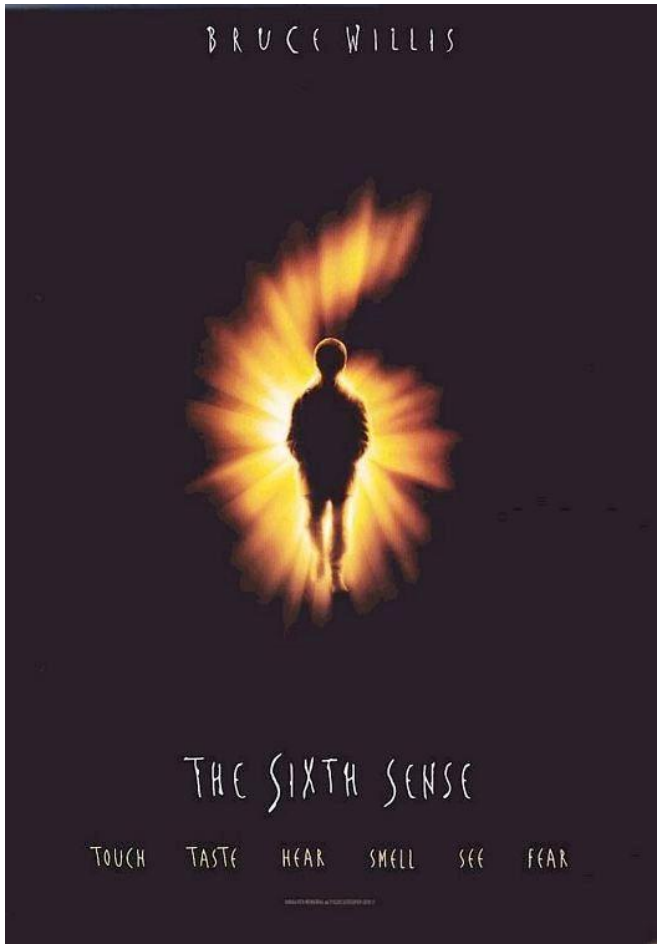


Fallacies of distributed computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Peter Deutsch, http://blogs.sun.com/jag/resource/Fallacies.html

One more fallacy



- You know who is alive

Why is it difficult?

- FLP impossibility result
  - Asynchronous systems
  - Consensus is impossible if a single process can crash

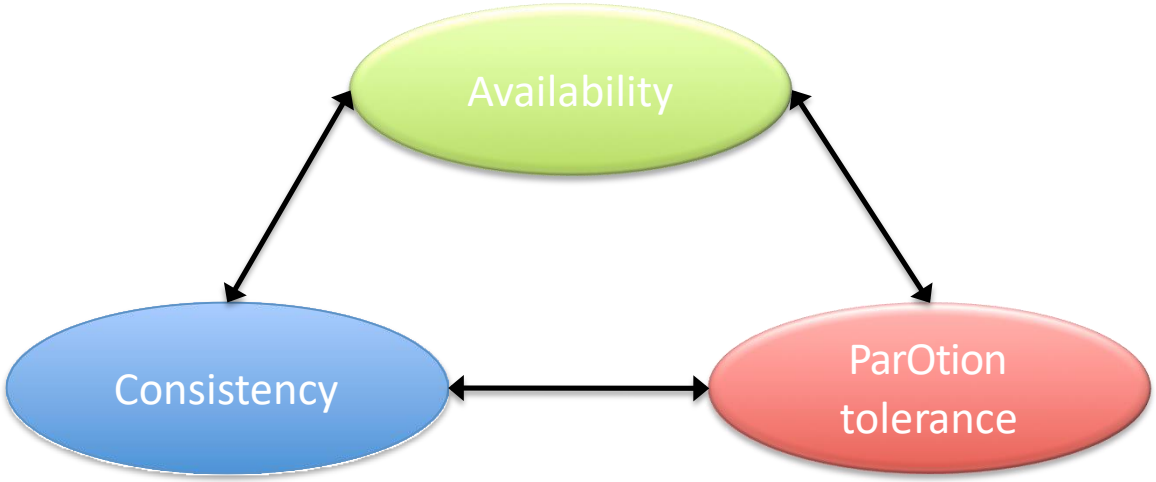
Fischer, Lynch, Paterson, ACM PODS, 1983

- According to Herlihy, we do need consensus
  - Wait-free synchronization
  - Wait-free: completion in a finite number of steps
  - Universal object: equivalent to solving consensus for *n* processes

Herlihy, ACM TOPLAS, 1991

Why is it difficult?

- CAP principle
  - Can't obtain availability, consistency, and parOtion tolerance simultaneously



Gilbert, Lynch, ACM SIGACT NEWS, 2002

The case for a coordination service

- Many impossibility results
- Many fallacies to stumble upon
- Several common requirements across applications
  - Duplicating is bad
  - Duplicating poorly is even worse
- Coordination service
  - Implement it once and well
  - Share by a number of applications

Current systems

- Chubby, Google
  - Lock service

Burrows, USENIX OSDI, 2006

- Centrifuge, Microsoft
  - Lease service

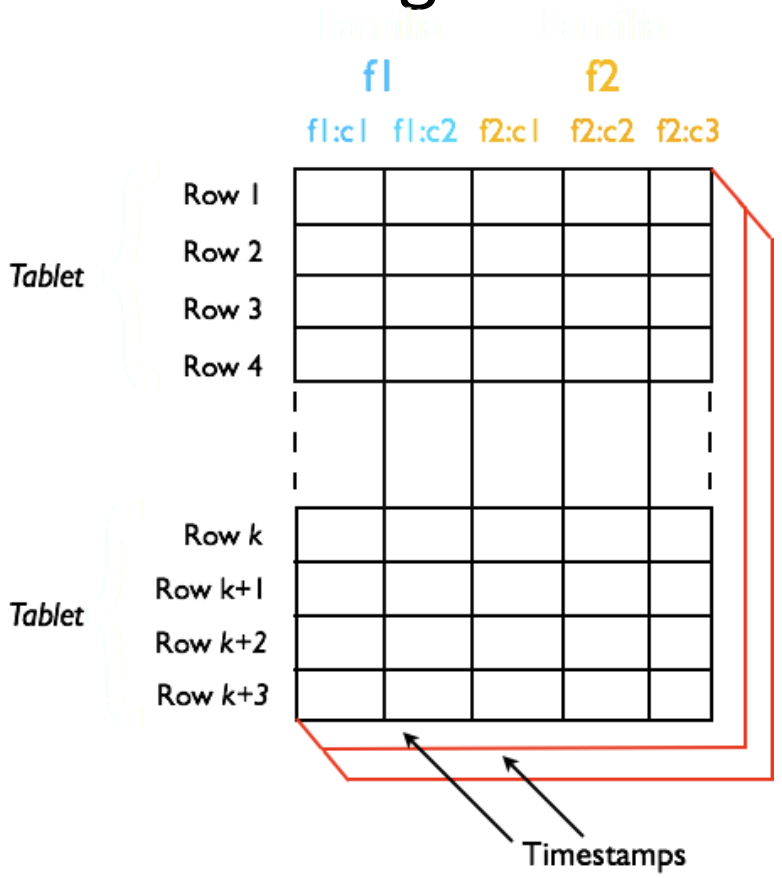
Adya et al., USENIX NSDI, 2010

- ZooKeeper, Yahoo!
  - Coordination kernel
  - On Apache since 2008

Hunt et al., USENIX ATC, 2010

Example – Bigtable, HBase

- Sparse column-oriented data storage
  - Tablet: range of rows
  - Unit of distribution
- Architecture
  - Master
  - Tablet servers

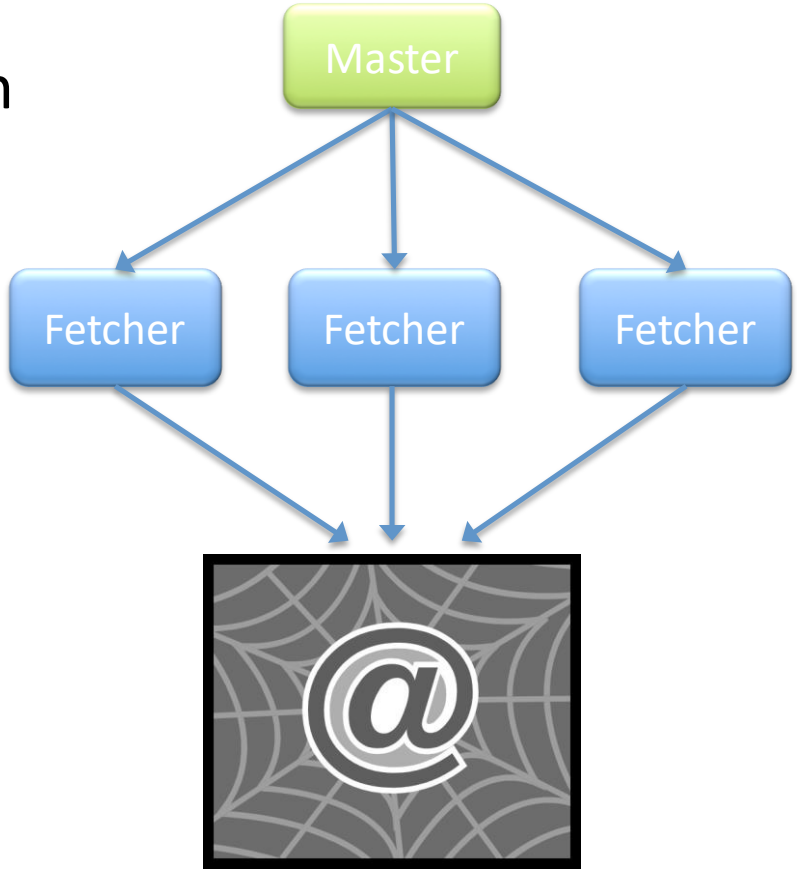


Example – Bigtable, HBase

- Master election
  - Tolerate master crashes
- Metadata management
  - ACLs, Tablet metadata
- Rendezvous
  - Find tablet server
- Crash detection
  - Live tablet servers

Example – Web crawling

- Fetching service
  - Fetch Web pages for search engine
- Master election
  - Assign work
- Metadata management
  - Politeness constraints
  - Shards
- Crash detection
  - Live workers



And more examples…

- GFS – Google File System
  - Master election
  - File system metadata
- KaCa - Document indexing system
  - Shard information
  - Index version coordination
- Hedwig – Pub-Sub system
  - Topic metadata
  - Topic assignment

Summary of Part 1

- Large infrastructures require coordination
- Fallacies of distributed compuOng
- Theory results: FLP, CAP
- Coordination services
- Examples
  - Web search
  - Storage systems

ZooKeeper Tutorial

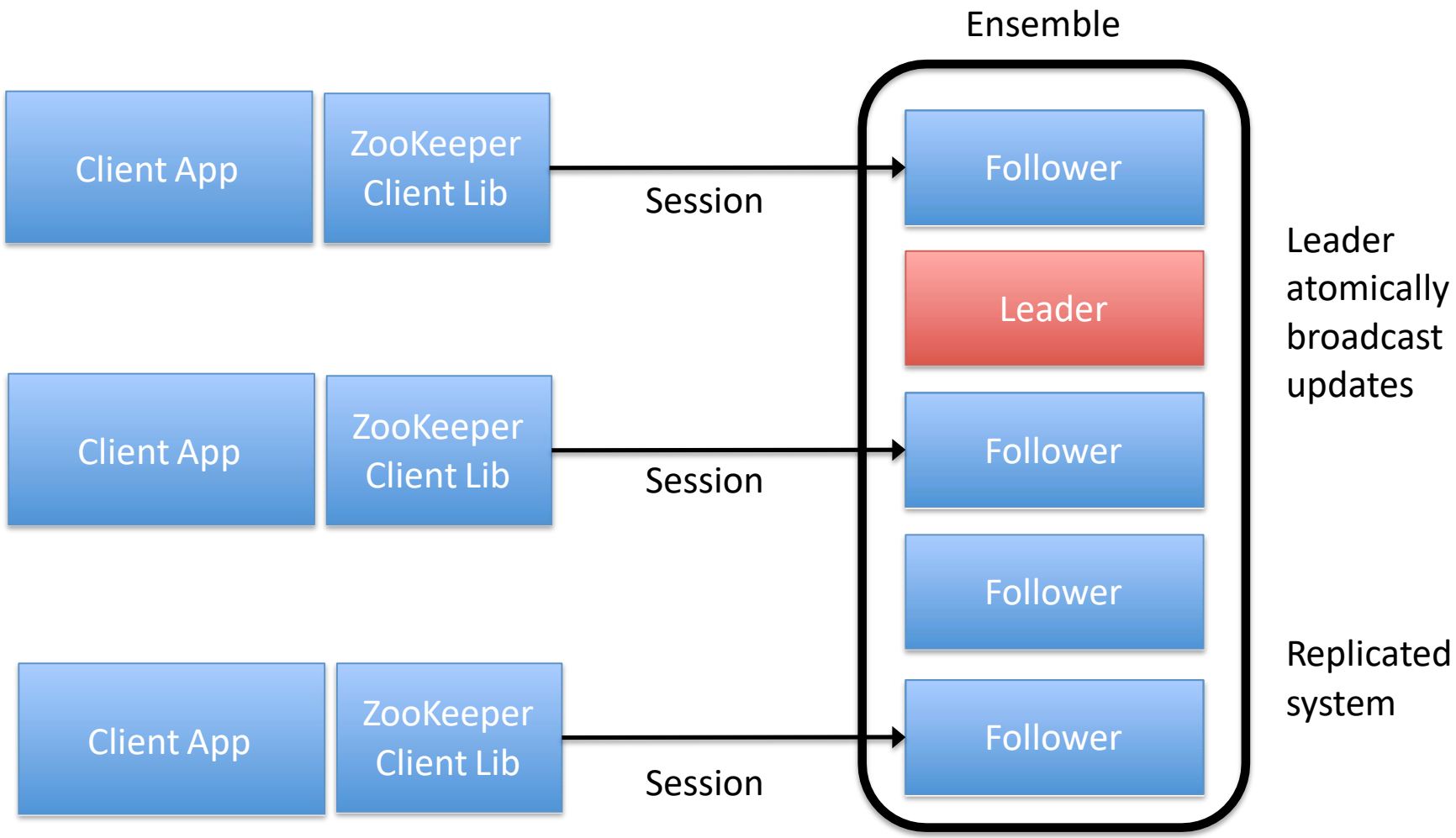
Part 2

The service

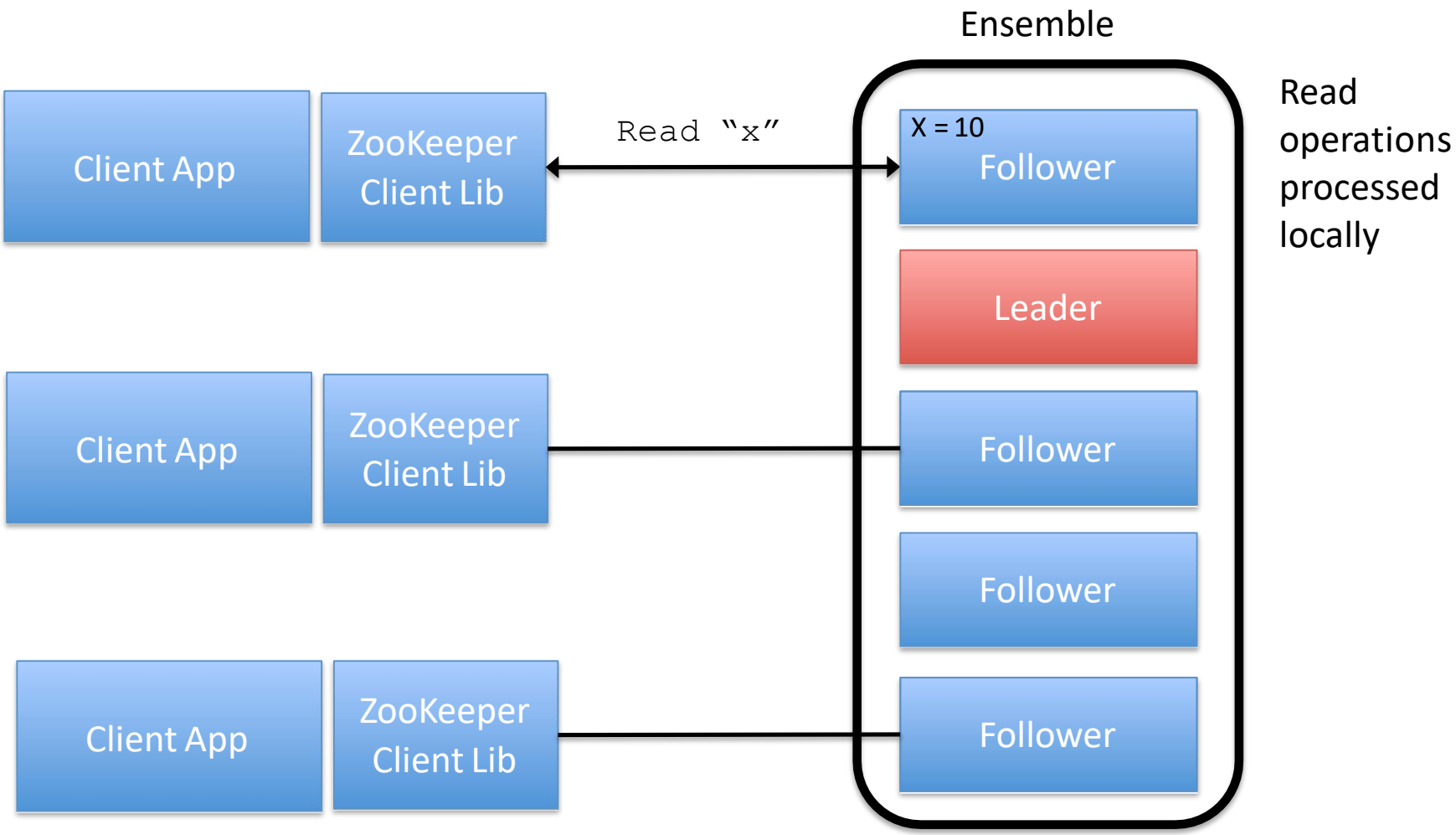
ZooKeeper Introduction

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

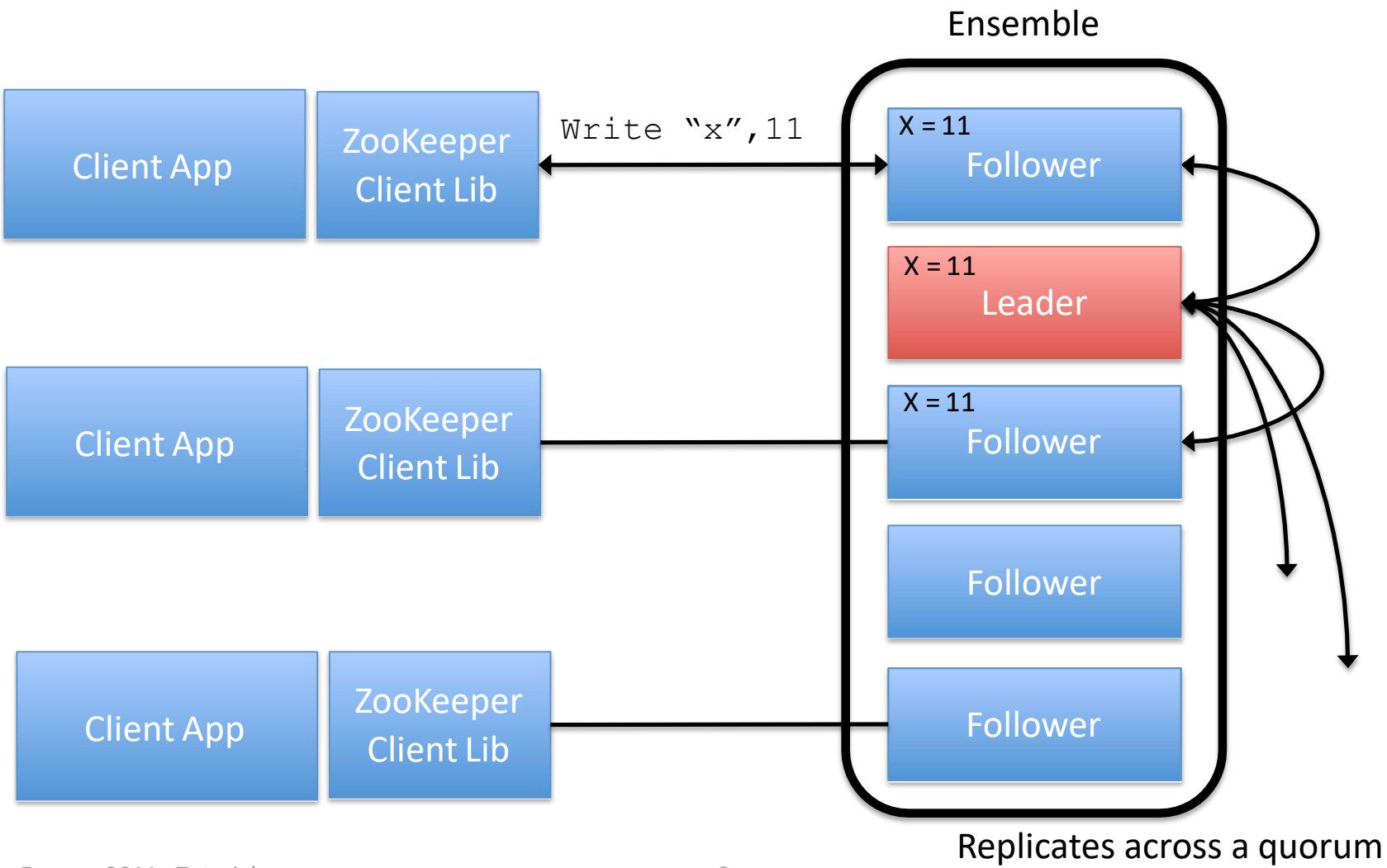
ZooKeeper: Overview



ZooKeeper: Read operations



ZooKeeper: Write operations



ZooKeeper: Semantics of Sessions

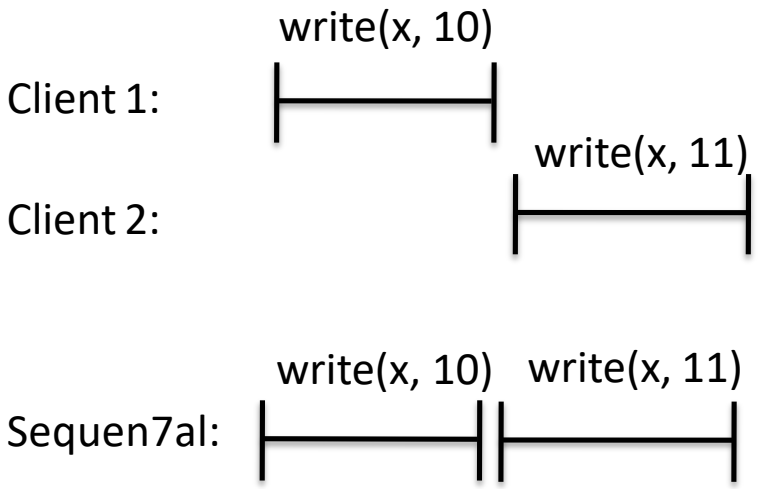
- A prefix of operations submitted through a session are executed
- Upon disconnection
  - Client lib tries to contact another server
  - Before session expires: connect to new server
  - Server must have seen a transaction id at least as large as the session

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`

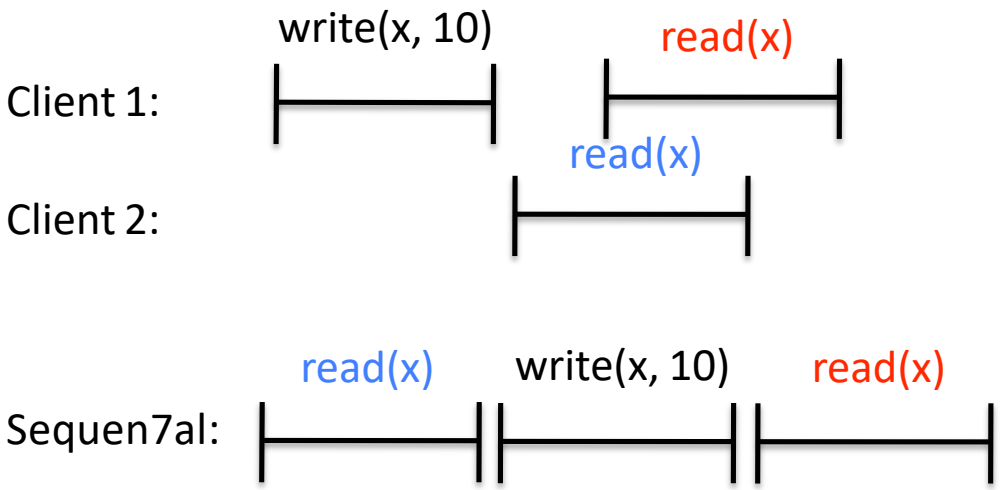
ZooKeeper: API

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



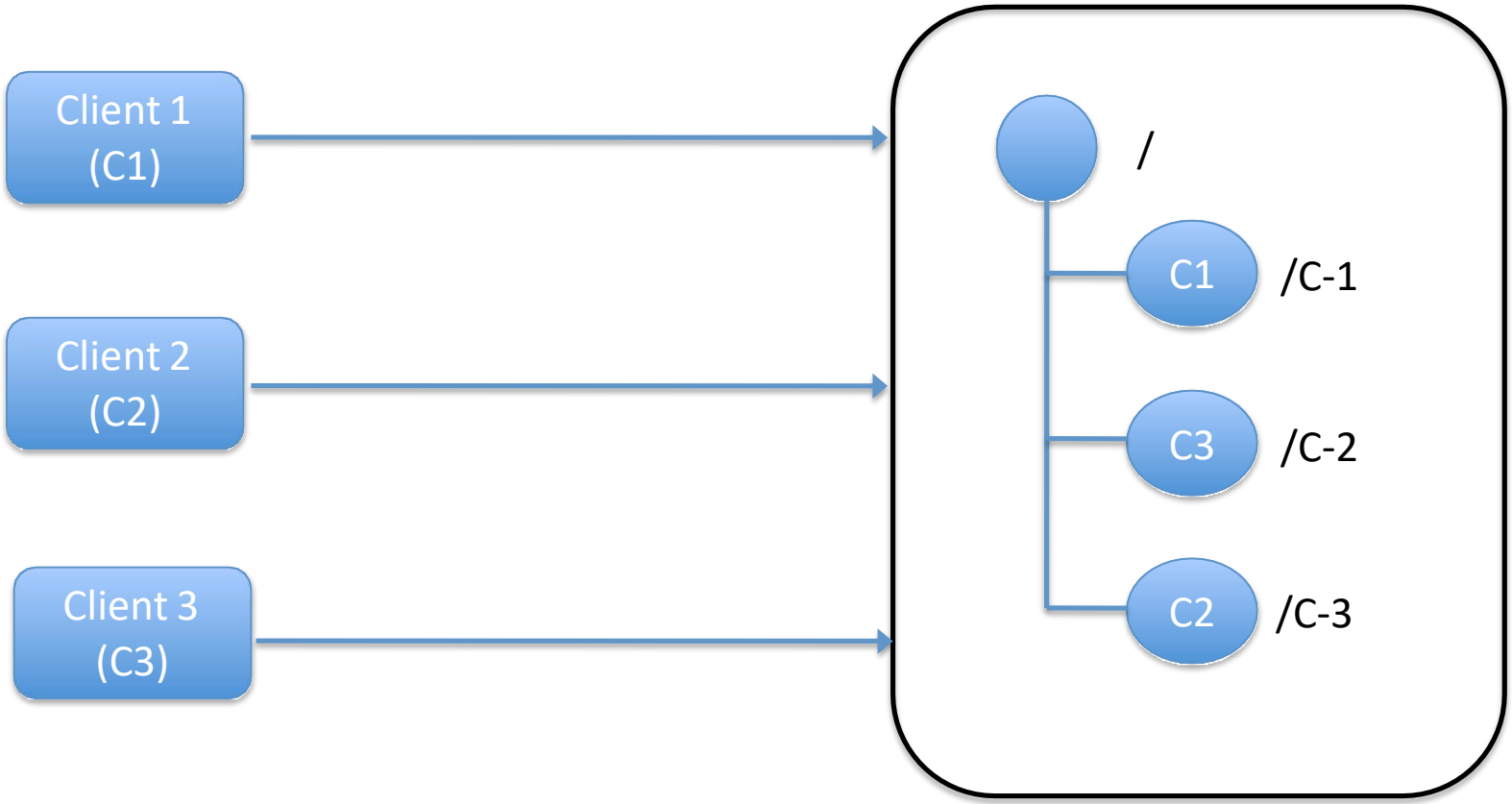
ZooKeeper: API

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



ZooKeeper: Example

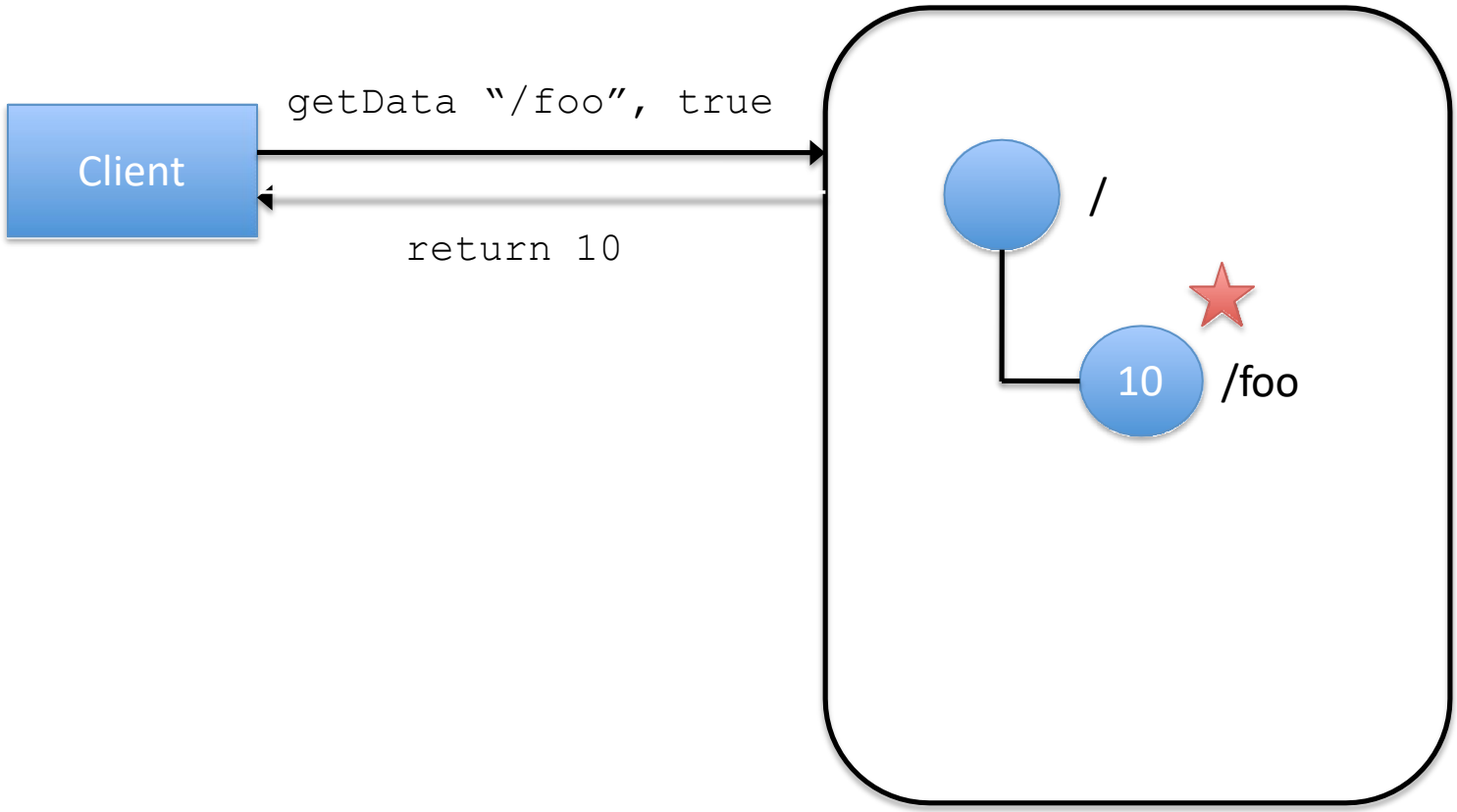
1- `create` "/C-", "C*i*", sequential, ephemeral
2- `getChildren` "/"
3- If not leader, `getData` "first node"



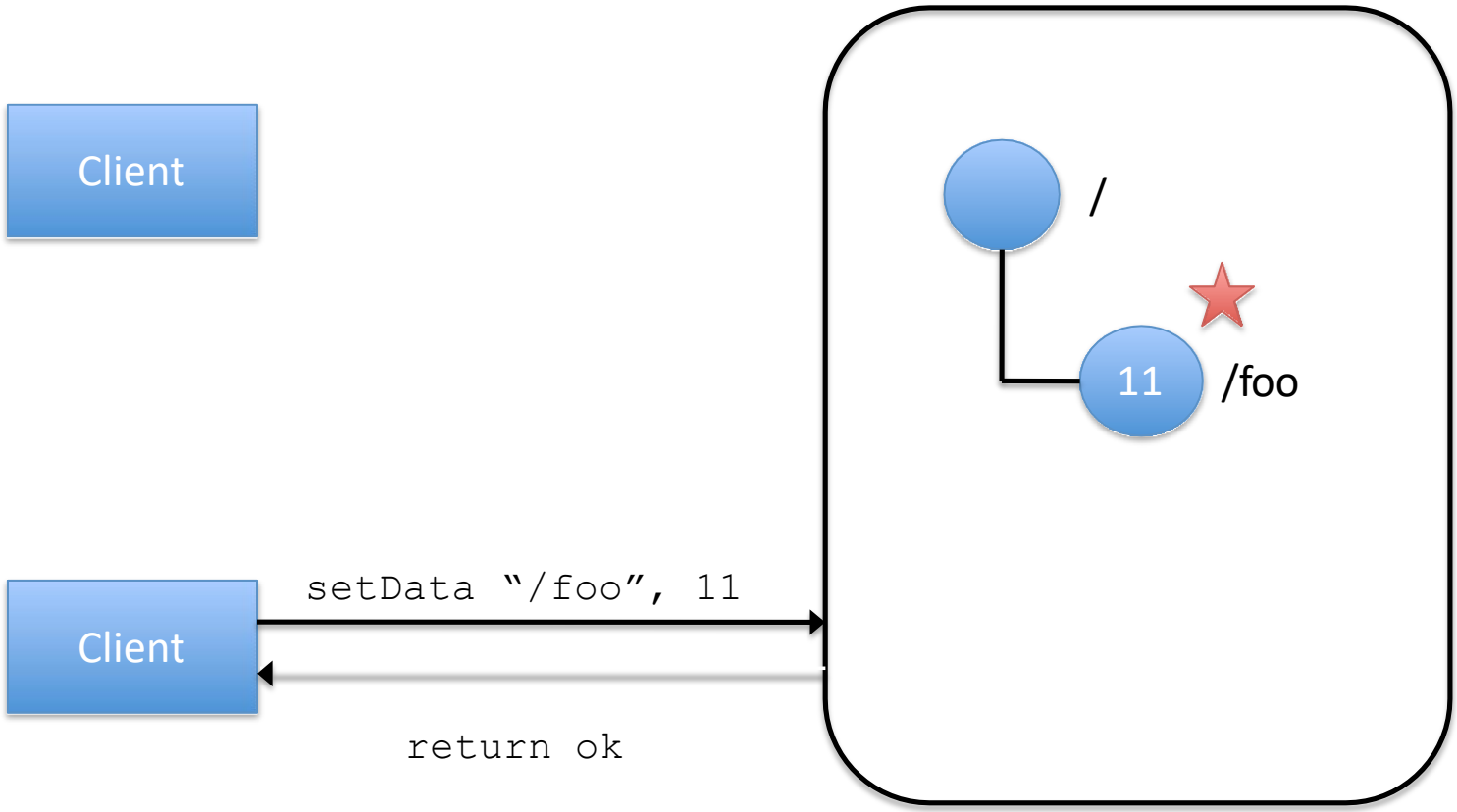
ZooKeeper: Znode changes

- 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

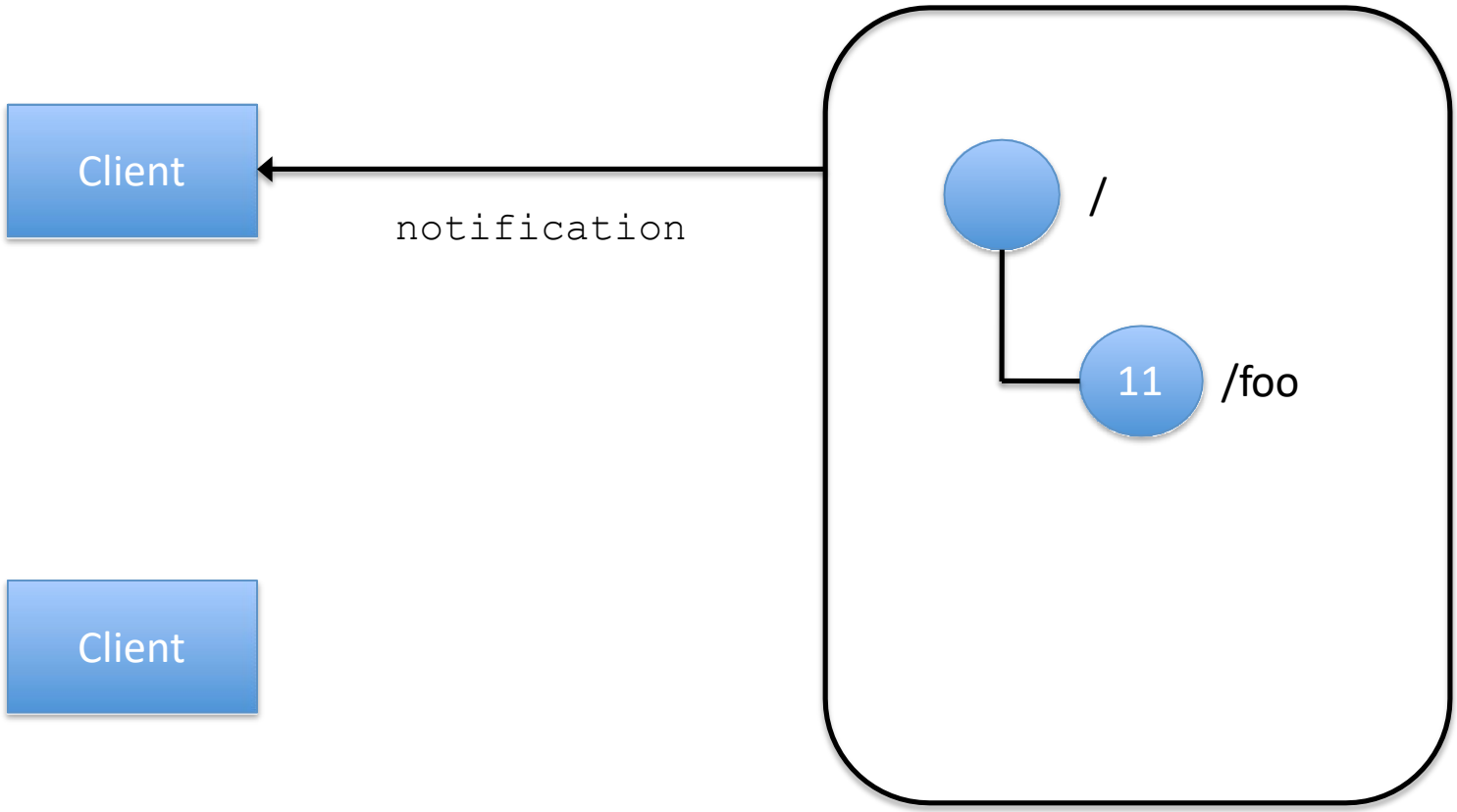
ZooKeeper: Watches



ZooKeeper: Watches



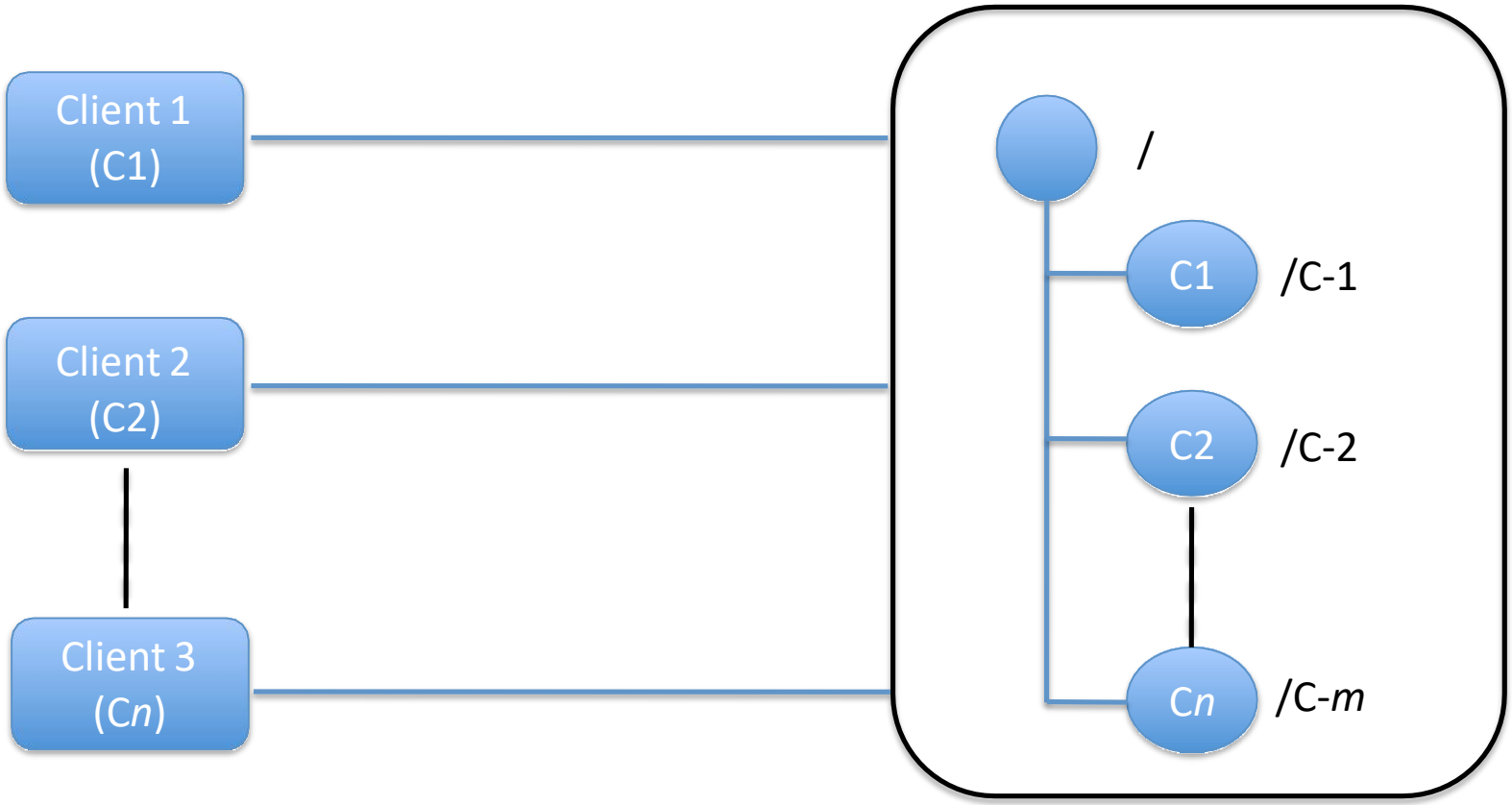
ZooKeeper: Watches



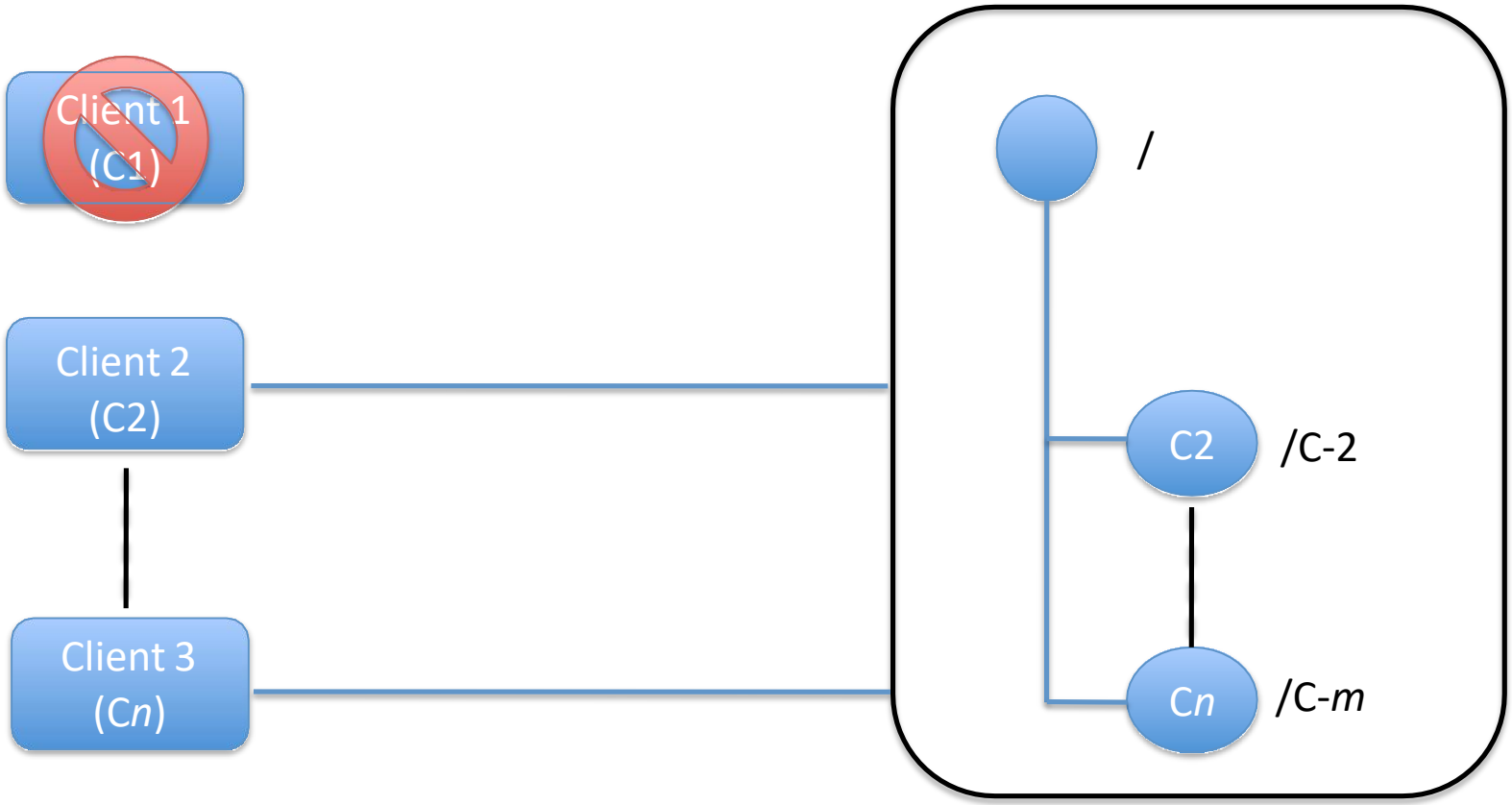
Watches, Locks, and the herd effect

- Herd effect
  - Large number of clients wake up simultaneously
- Load spikes
  - Undesirable

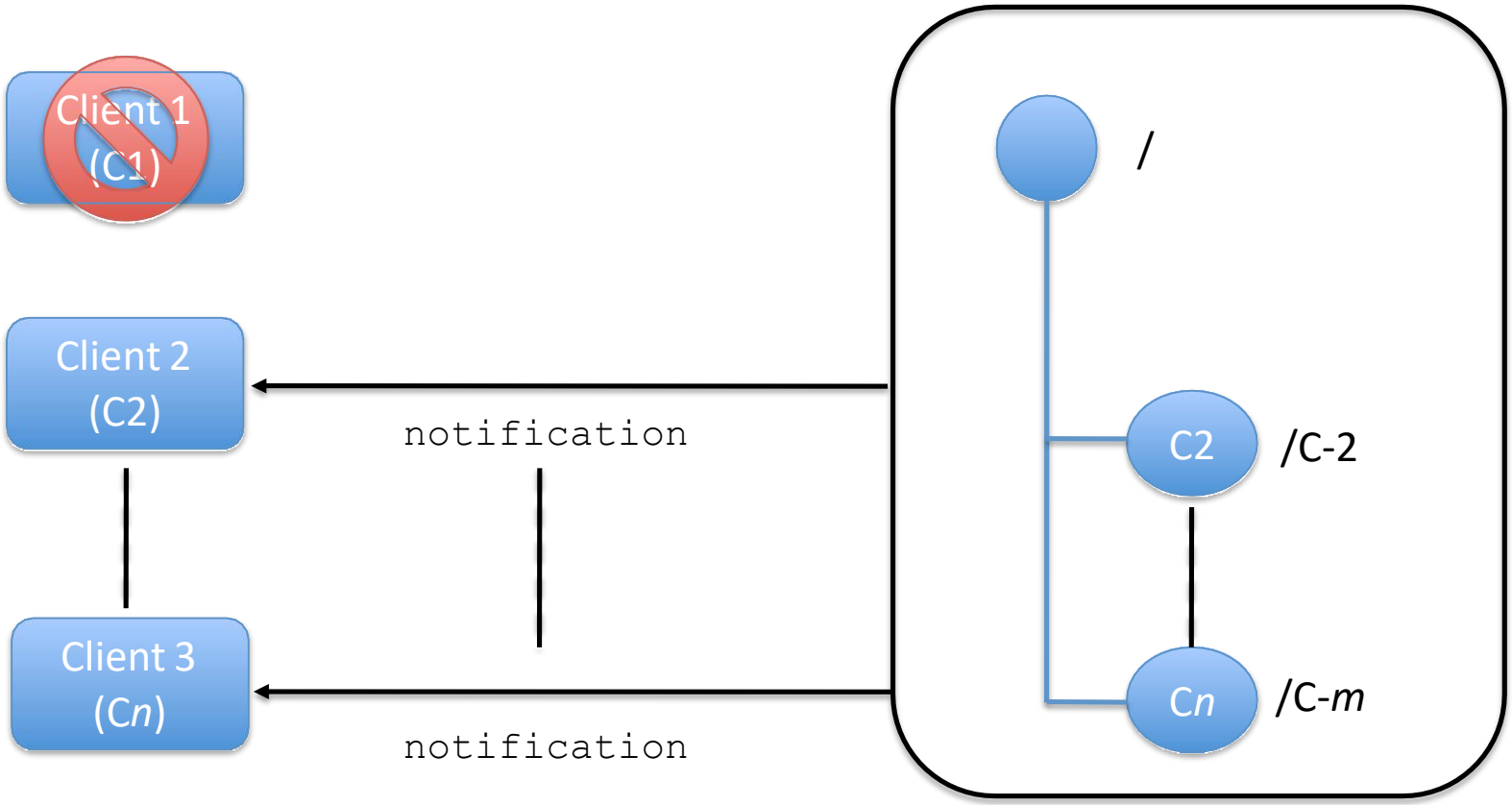
Watches, Locks, and the herd effect



Watches, Locks, and the herd effect



Watches, Locks, and the herd effect

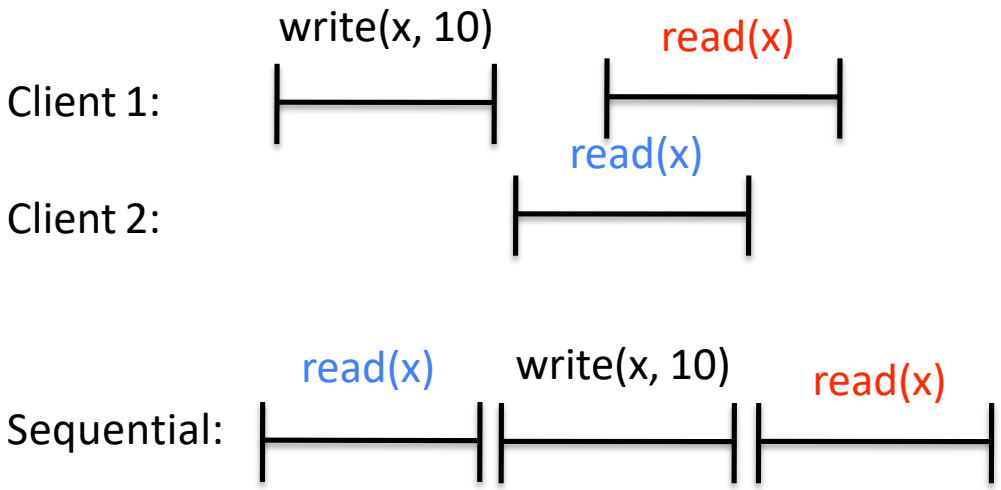


Watches, Locks, and the herd effect

- 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

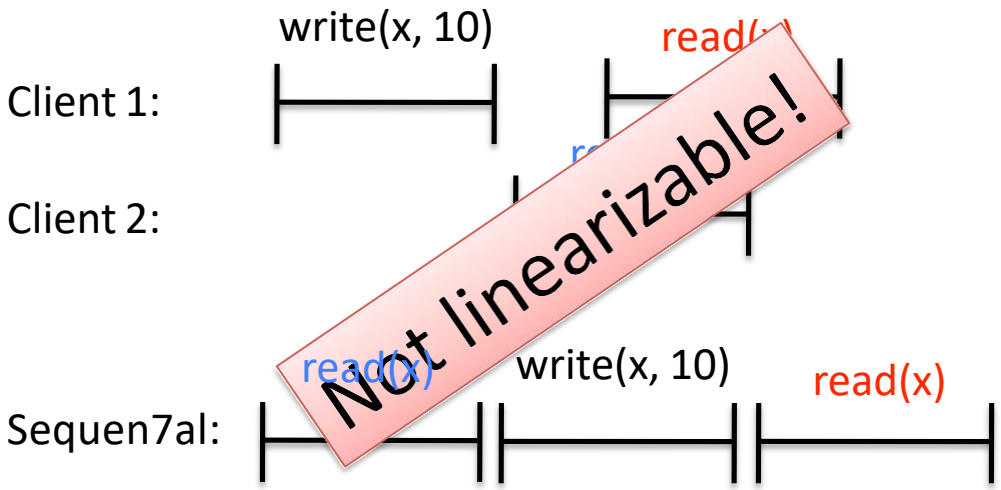
Linearizability

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



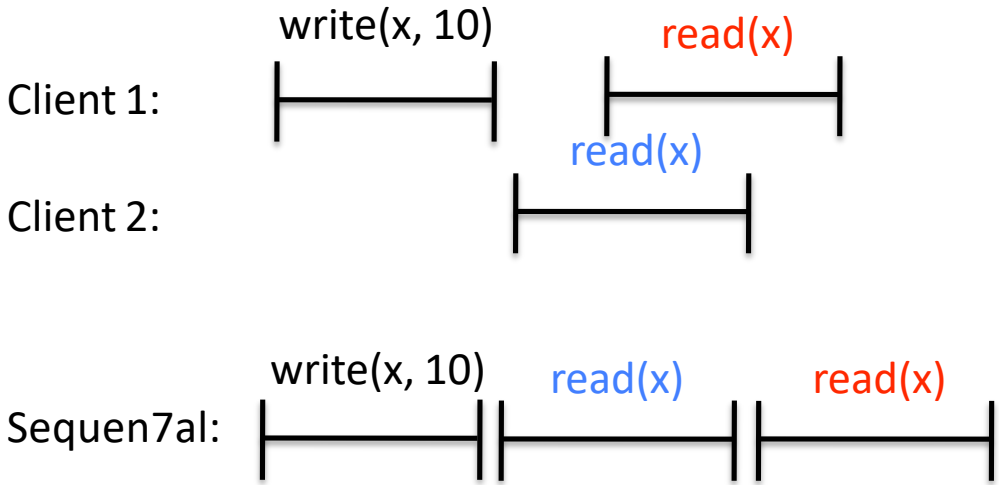
Linearizability

- 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

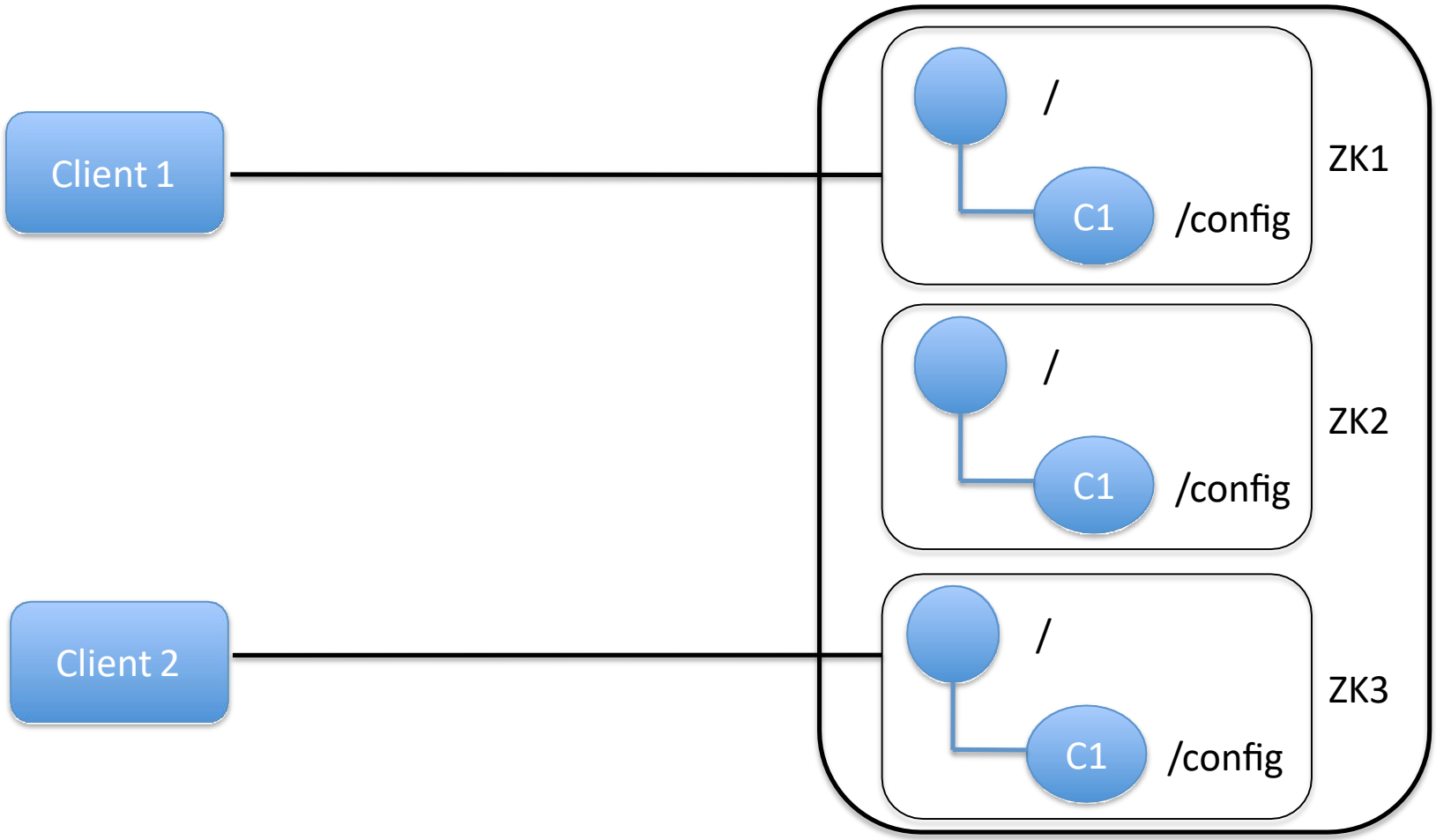
Implementing consensus

- 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'`

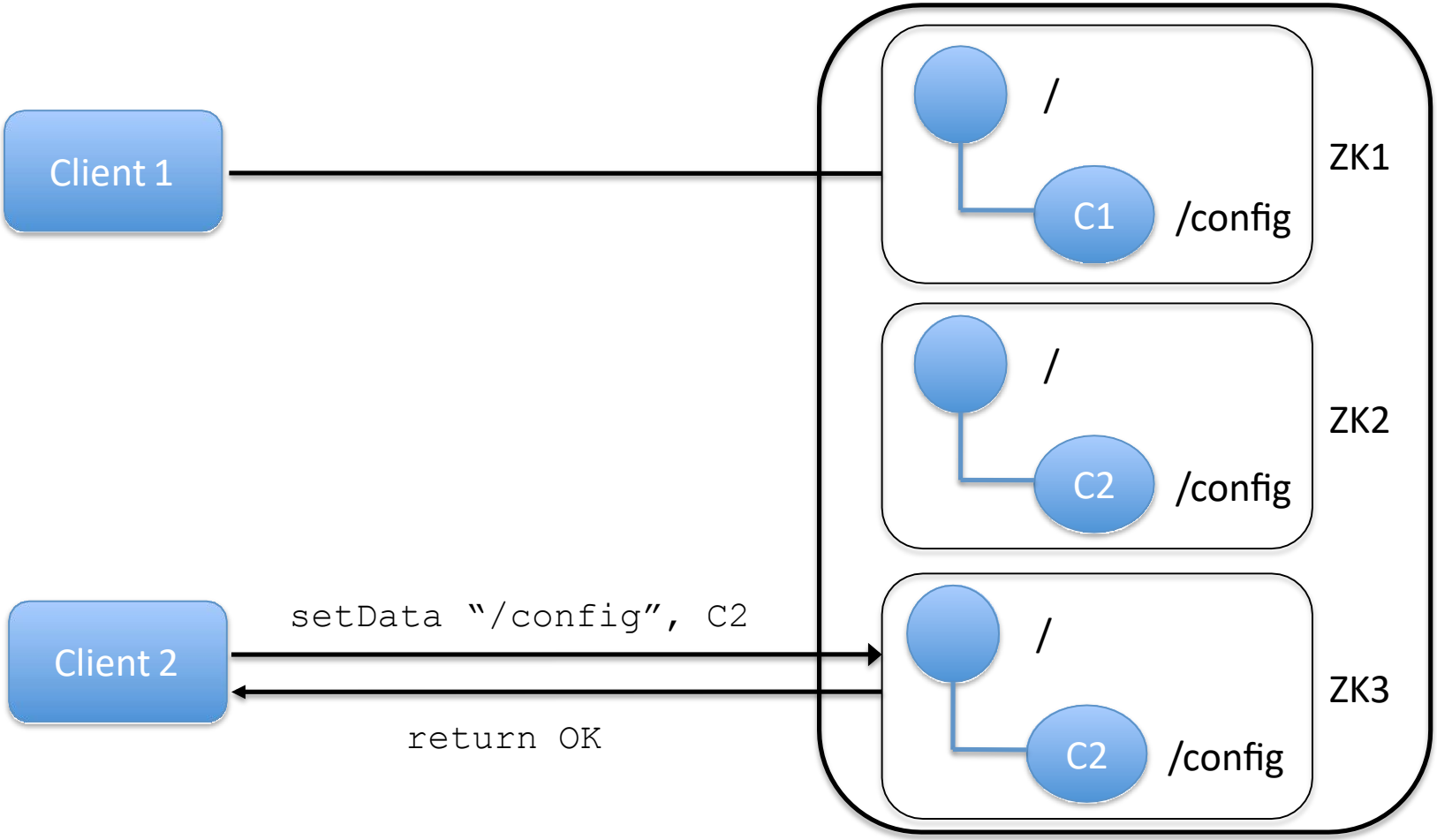
Linearizability

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

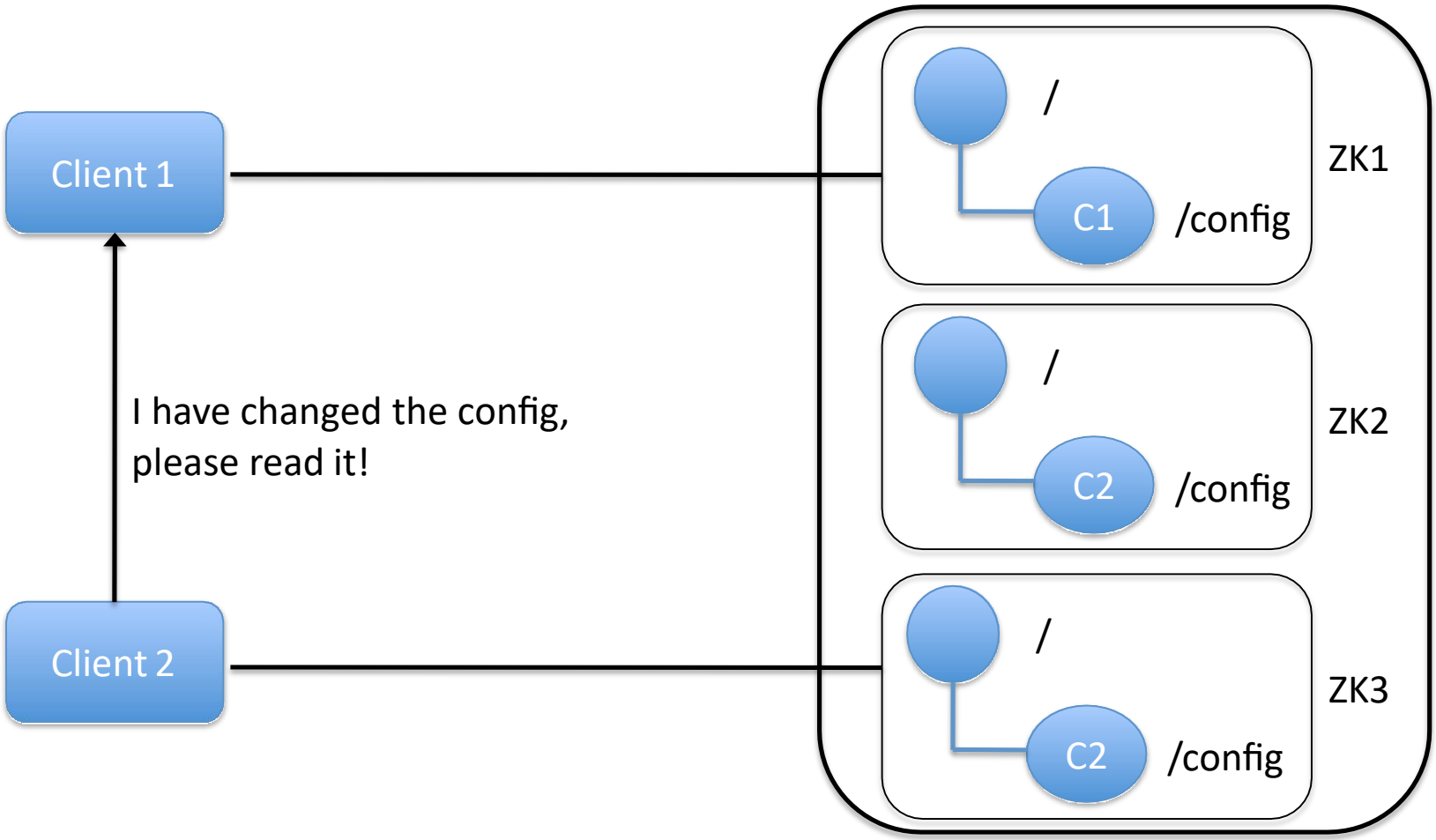
Hidden channels



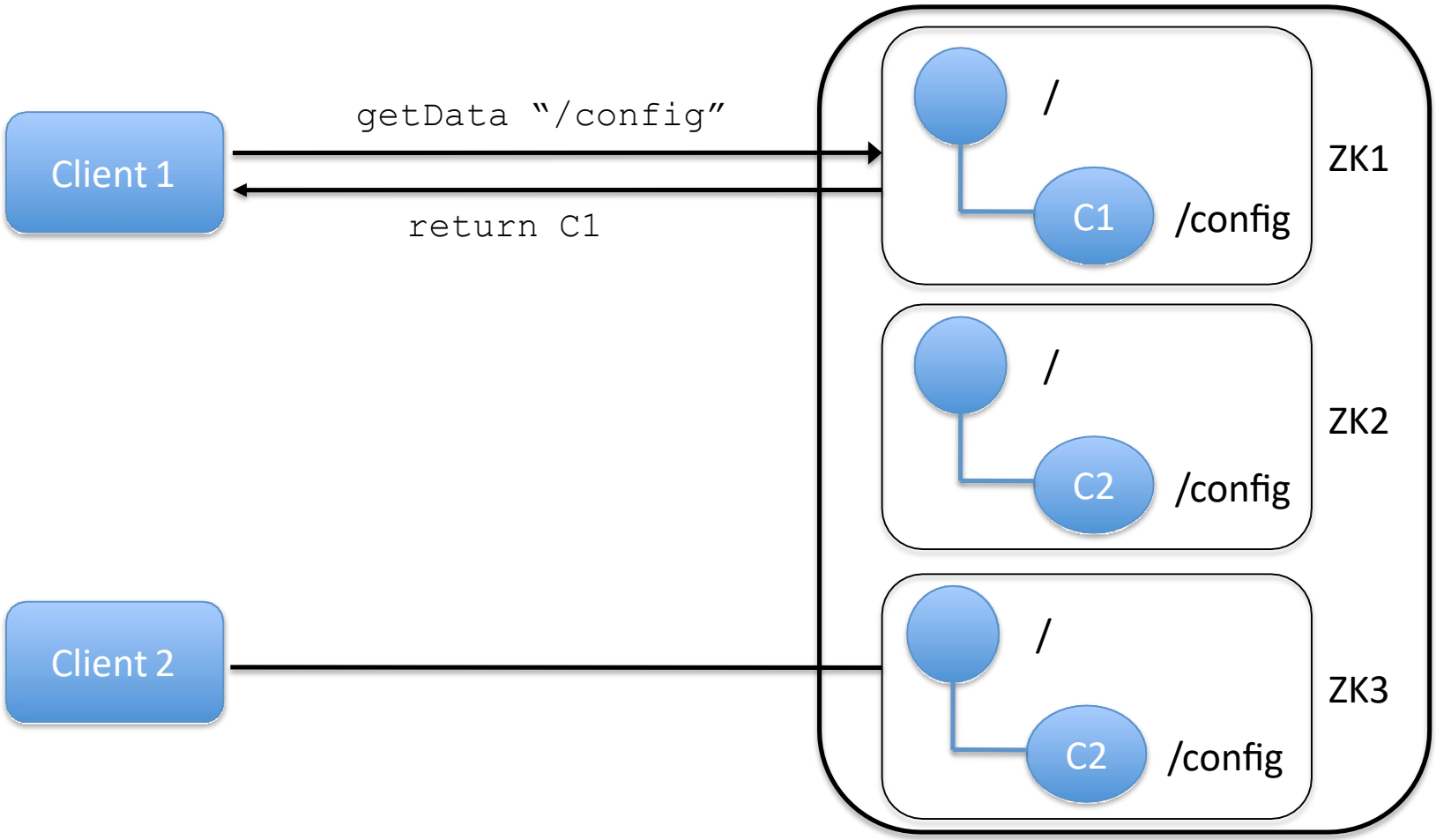
Hidden channels



Hidden channels

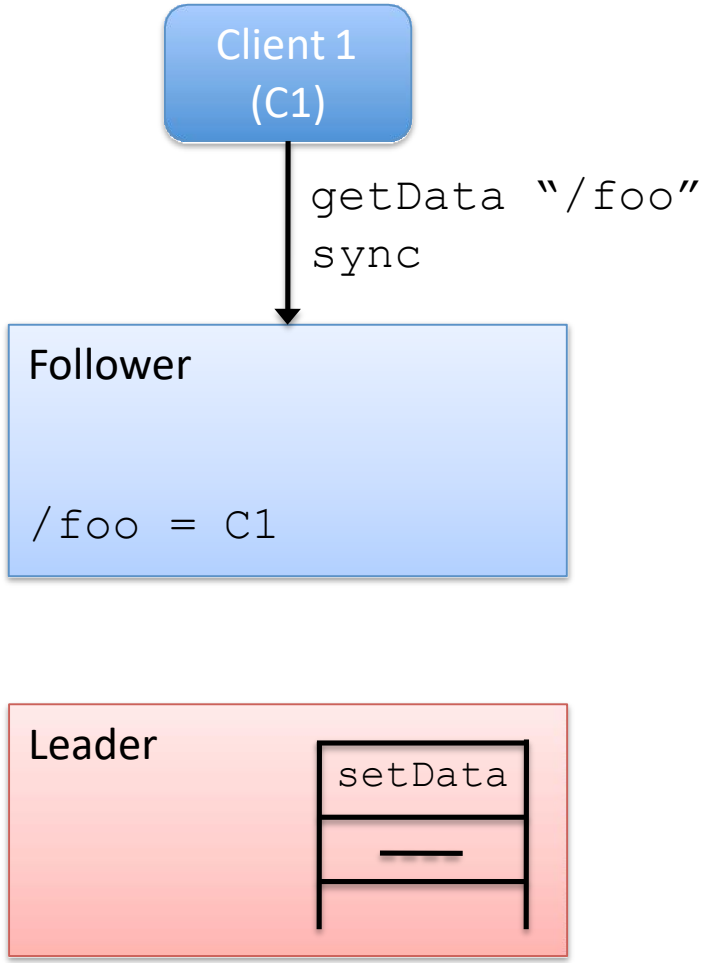


Hidden channels



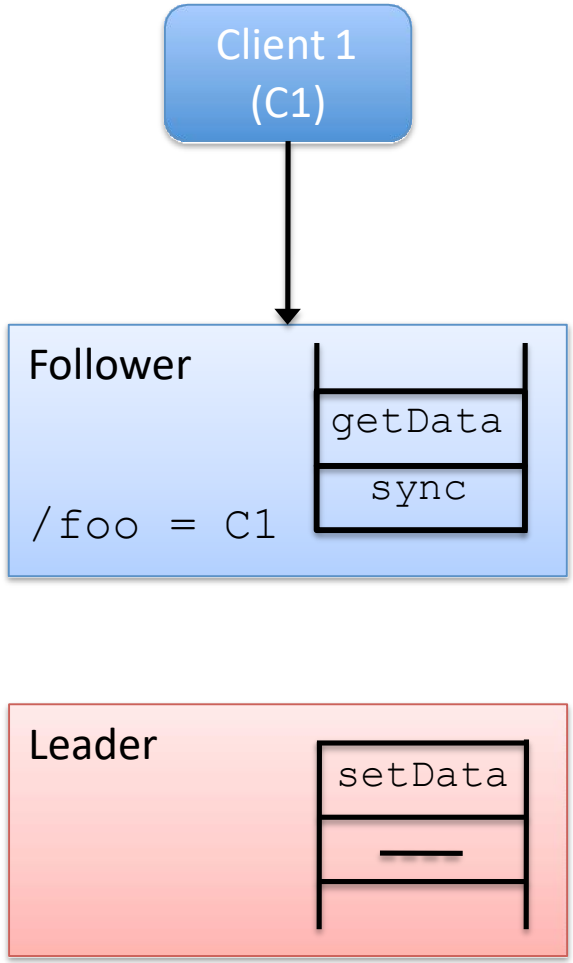
A hat trick…

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



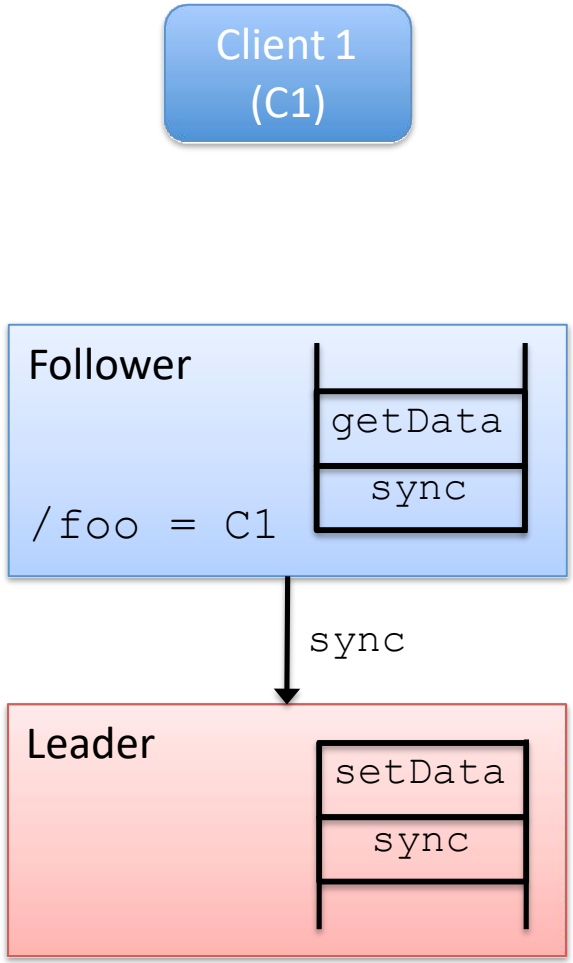
A hat trick…

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



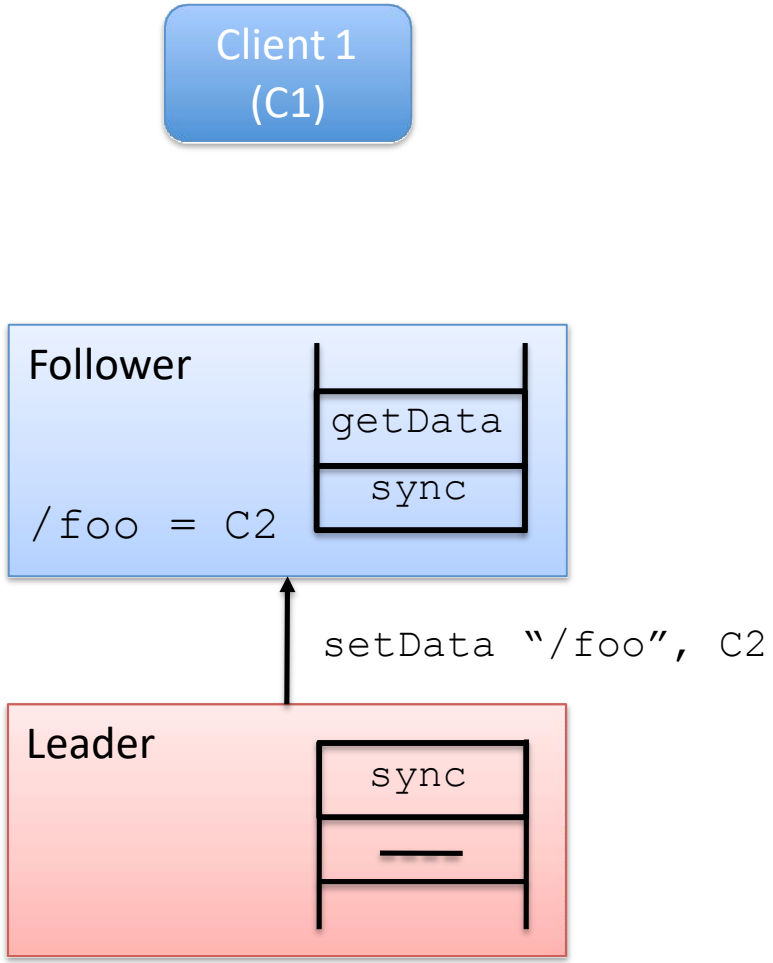
A hat trick…

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



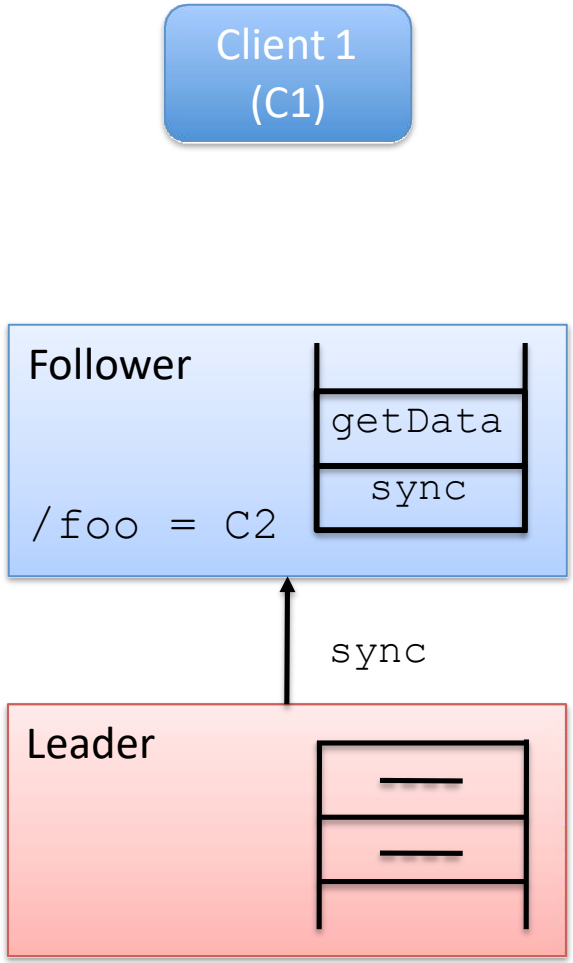
A hat trick…

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



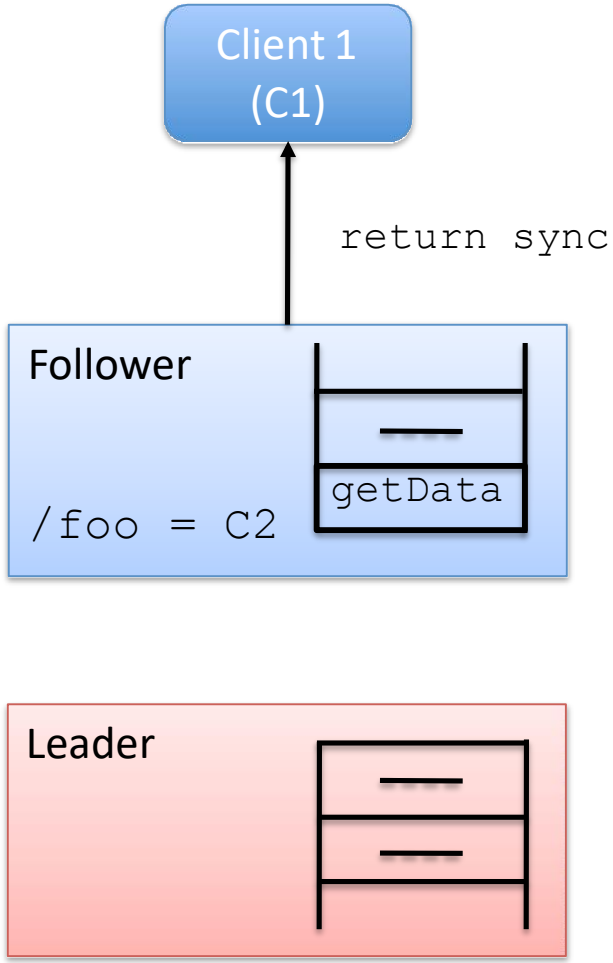
A hat trick…

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



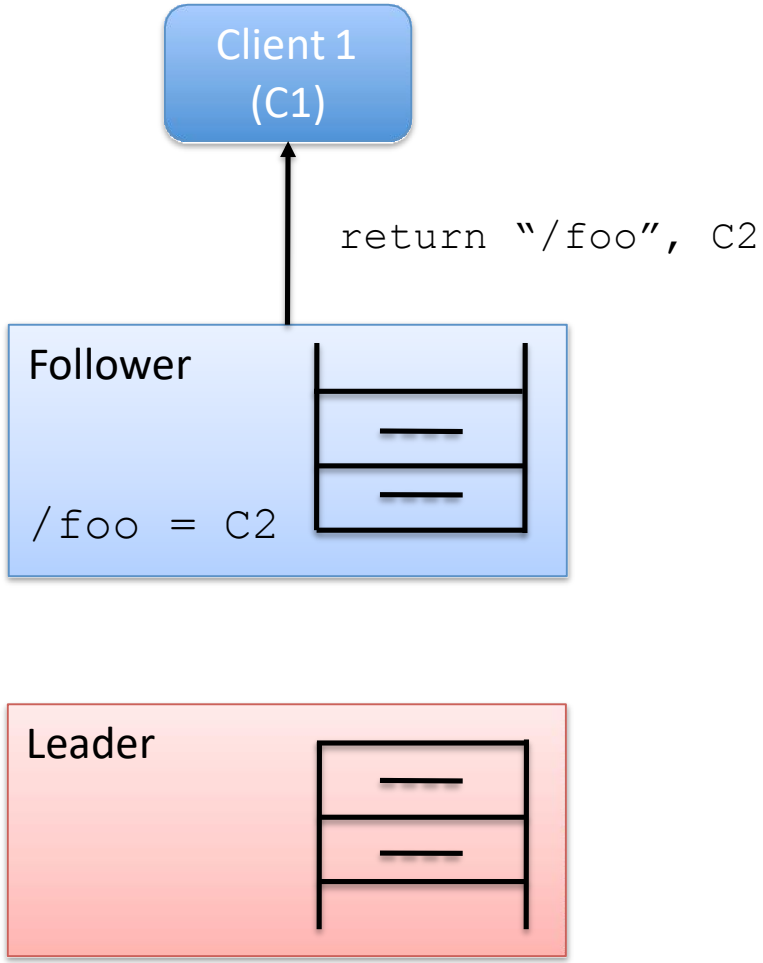
A hat trick…

- `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

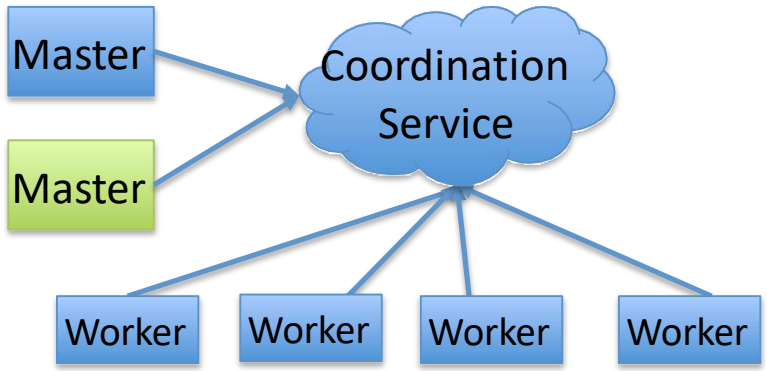
ZooKeeper Tutorial

Part 3

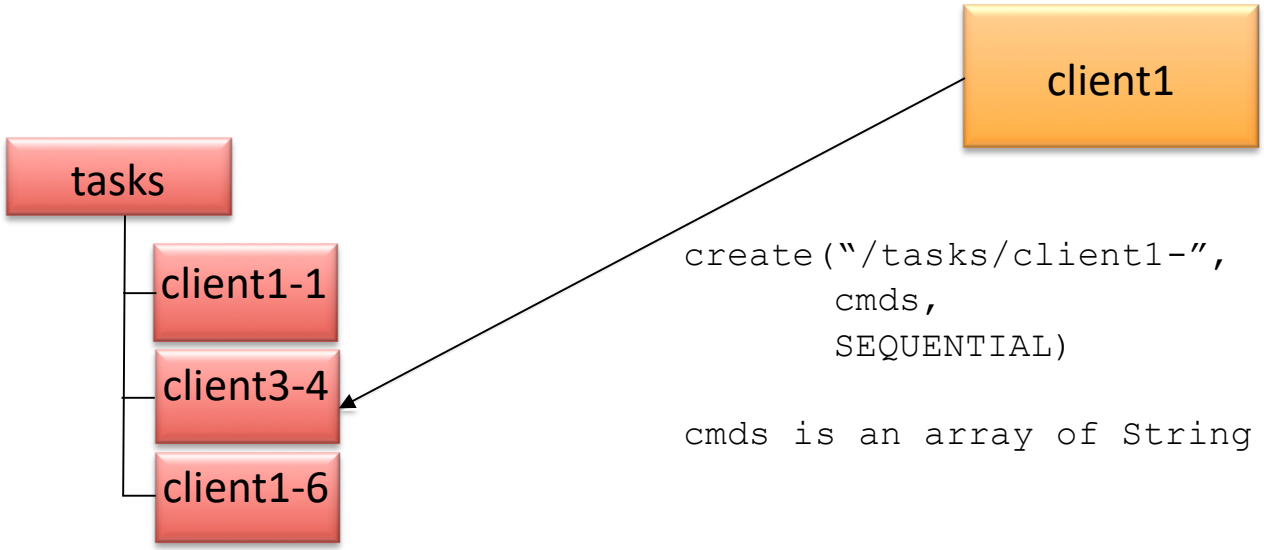
How it really works

Master/Worker System

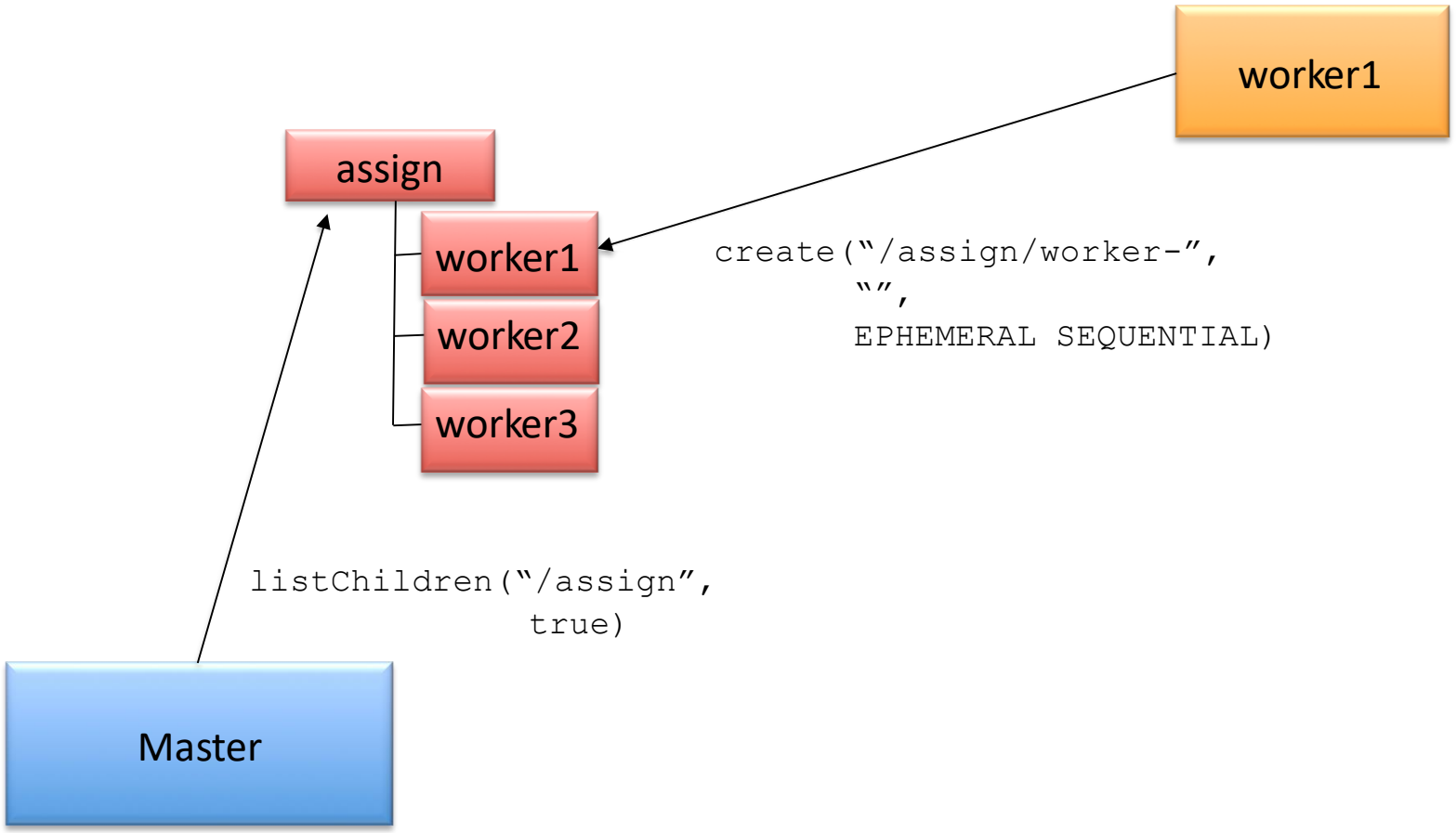
- Clients
  - Monitor the tasks
  - Queue tasks to be executed
- Masters
  - Assign tasks to workers
- Workers
  - Get tasks from the master
  - Execute tasks



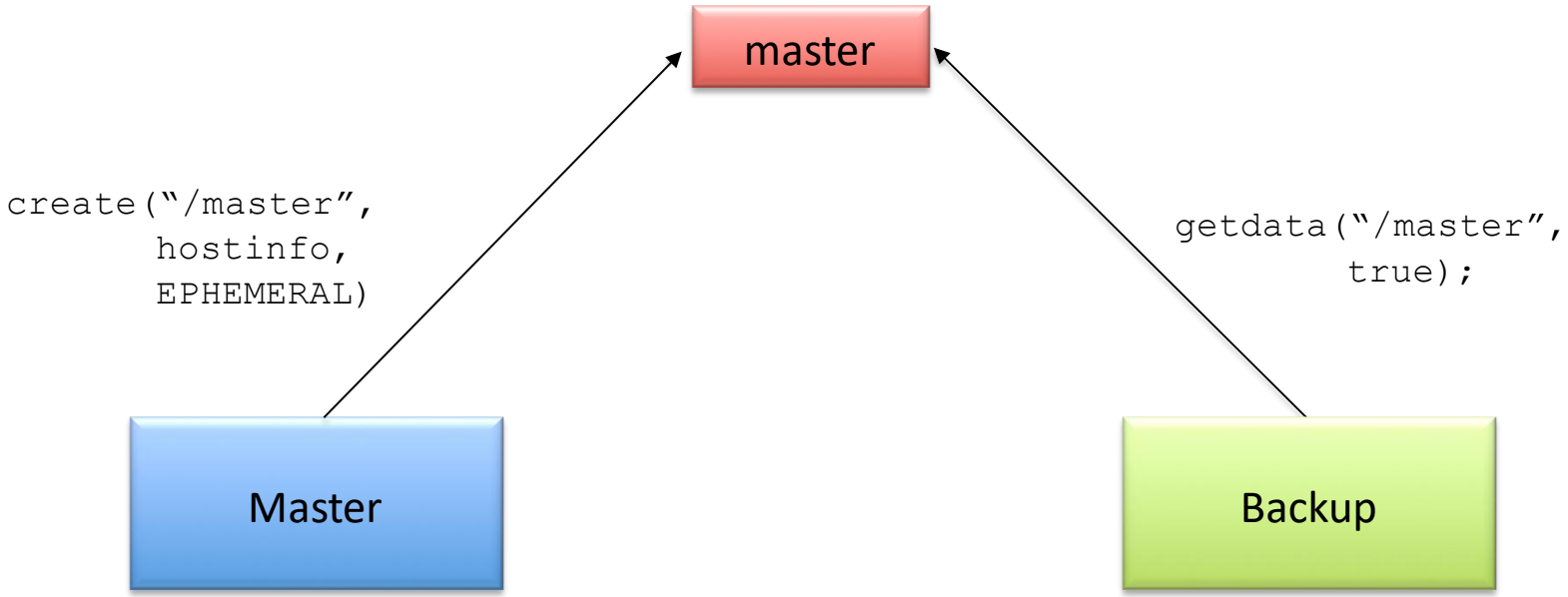
Task Queue


client1
tasks
client1-1
client3-4
client1-6
create("/tasks/client1-",
cmds,
SEQUENTIAL)
cmds is an array of String

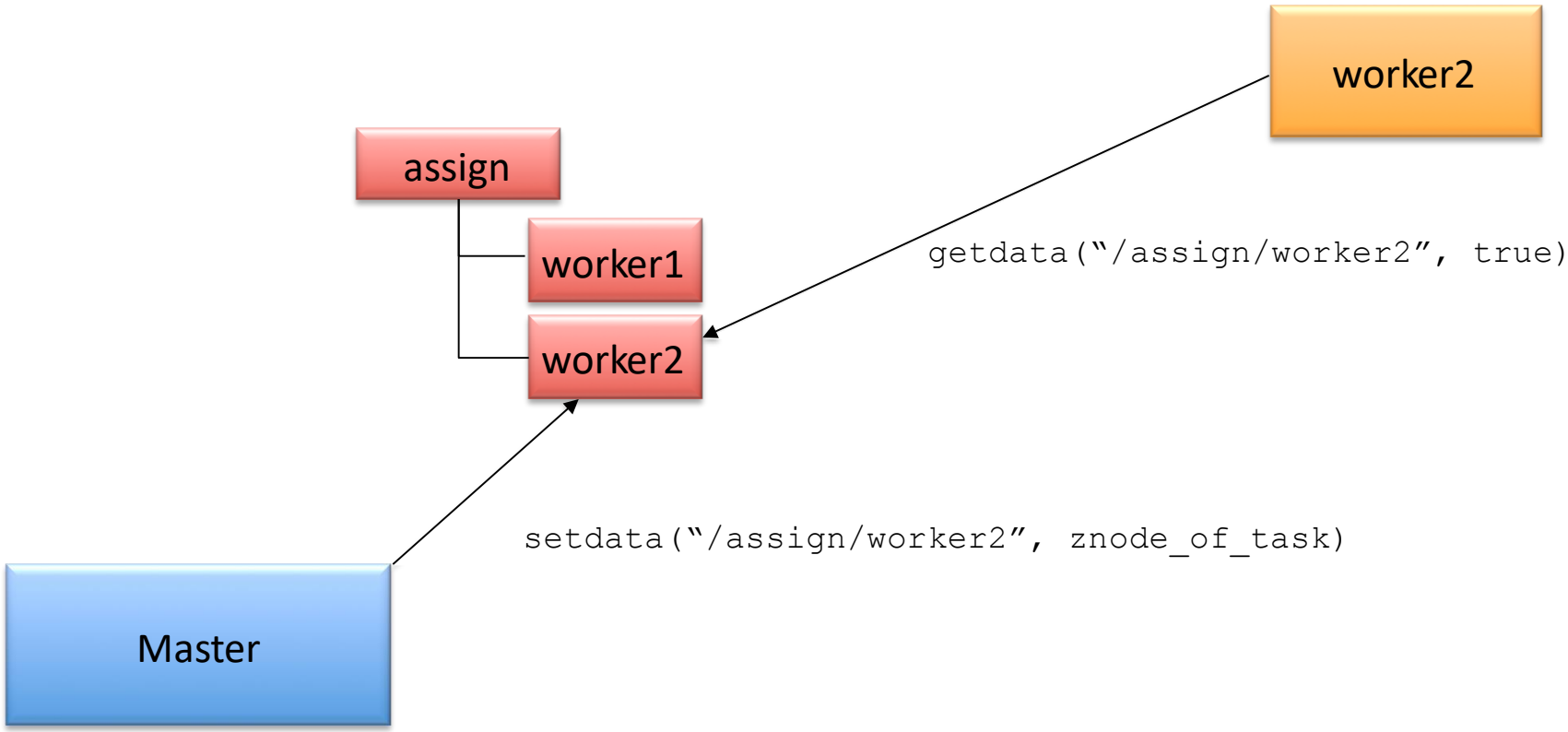
Group Membership



Leader ElecKon



Configuration



Connecting to ZooKeeper

- Everyone has their own ZooKeeper address and auth info.
- Try connecting to ZooKeeper with the CLI.

java -jar zookeeper-3.3.2-fatjar.jar client zkaddr

- Use addAuth command to authenKcate
- Try out some commands
  - Create znodes for `/servers`, `/tasks`, `/assign`

Worker Processing

- Create a session
- Create the “worker” ephemeral znode
- Watch for the assign znode
- Deal with the watches
  - Processing the assignment
    - Update status in the task
    - Delete assignment znode when finished
  - What do to with SessionExpired

Code on your own or follow together

Client Processing

- Create a session
- Create a task as a child of the `/tasks` znode
- Watch the status child of the `/tasks` znode

Code on your own or follow together

Master Processing

- Create a session
- Do leader election using master znode
- Watch the worker list
- Watch the task queue
- Watch the assignment queue
- Deal with the watches
  - Deal with workers coming and going
  - Assign new tasks
  - Watch for compleKons

Code on your own or follow together

Give it a try…

- Start up the master
- Start up a worker
- Try submitting a command
- Queue up a bunch of sleep 100
- Add more workers
- Try killing a worker
- Try killing the master. Did take over work?

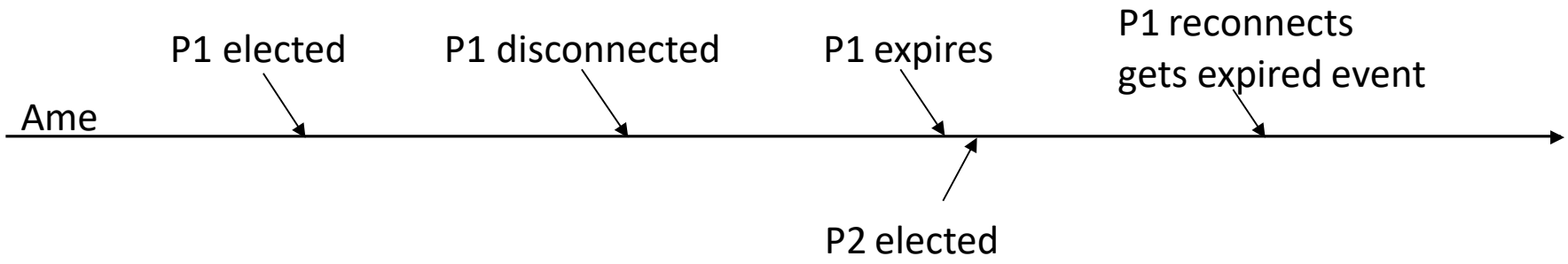
ZooKeeper Tutorial

Part 4

Caveat Emptor

Revisit FLP and CAP

- What should a master do when disconnected?
  - What is the consequence of acAng as a master while disconnected?



Revisit FLP and CAP

- What happens if master elecAon gets a "`ConnectionLossException`" aLer the create?
  - How do you fix it?
  - How do you test it?

Guidelines to `ConnectionLoss`

- A process will not see state changes while disconnected
- Masters should act very conservaAvely, they should not assume that they sAll have mastership
- Don't treat as if it's the end of the world. The client library will try to recover the session

Other issues

- Watch out for `SEQUENTIAL` | `EPHEMERAL`!
- Problems resetting the ZooKeeper state
  - What happens when you clear server state while clients are running?
  - What happens when you clear some servers but not others?

WriAng a test

- Use JUnit
- Use QuorumBase
  - In setup call `QuorumBase.setup()`
  - In tearDown call `QuorumBase.tearDown()`
- Write a simple test
  - Use `QuorumBase.hostPort` to iniAalize the ZooKeeper object in the tests
  - Startup a master and a backup.
  - Kill the master and make sure backup takes over

Guidelines for `SessionExpiration`

- It is the end of the world!
- Should be rare.
- The session handle is dead, so you need a new one.
- It is dangerous to try to transparently recover by creating a new session. Usually there is some cleanup and setup that needs to be done

Code on your own or follow together

Summary

- When used properly ZooKeeper can make it easy to build distributed applicaAons.
- ZooKeeper is a tool to help you deal with the chaos of distributed systems. It isn't magic.
  - Don't try to shortcut the API
  - Think about the consequences of `ConnectionLoss` and `SessionExpiration`
  - Make sure you test
- Checkout the developer resources

  http://zookeeper.apache.org