Background on Microkernels

Prof. Pamod Bhatotia
https://dse.in.tum.de/bhatotia/

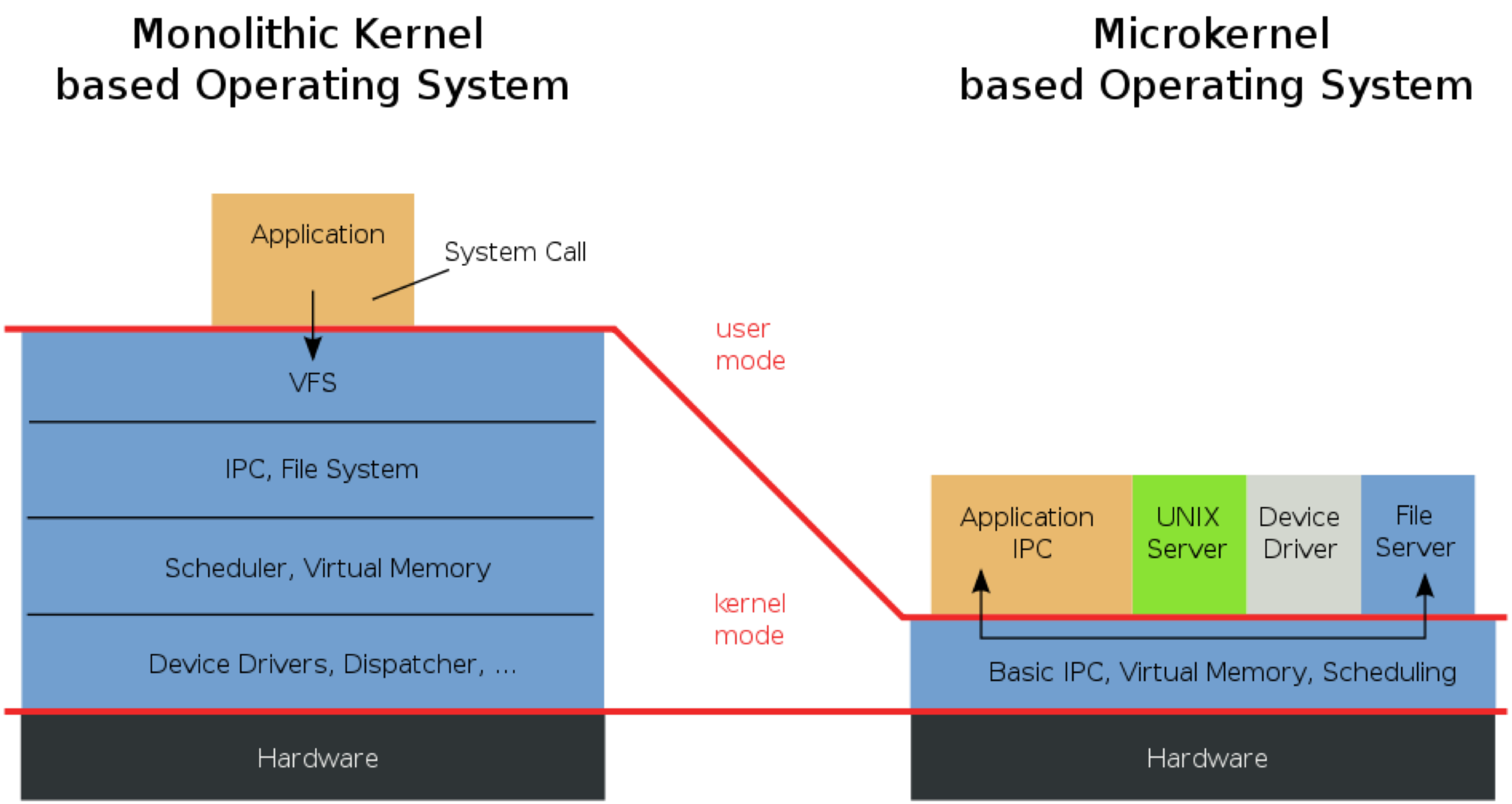
Monolithic Kernels

- All OS services operate in kernel space
- Good performance
- Disadvantages
  - Dependencies between system component
  - Complex & huge (millions(!) of lines of code)
  - Larger size makes it hard to maintain
- E.g. Multics, Unix, BSD, Linux

Microkernels

- Minimalist approach
  - IPC, virtual memory, thread scheduling
- Put the rest into user space
  - Device drivers, networking, file system, user interface
- More stable with less services in kernel space
- Disadvantages
  - Lots of system calls and context switches
- E.g. Mach, L4, AmigaOS, Minix, K42

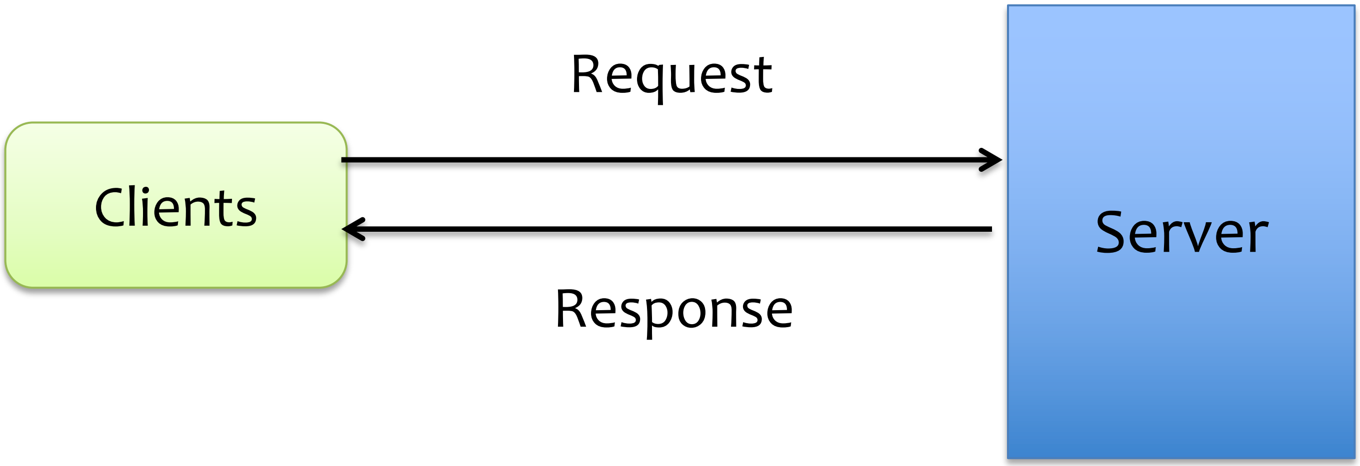
Microkernel Vs Monolithic Kernels



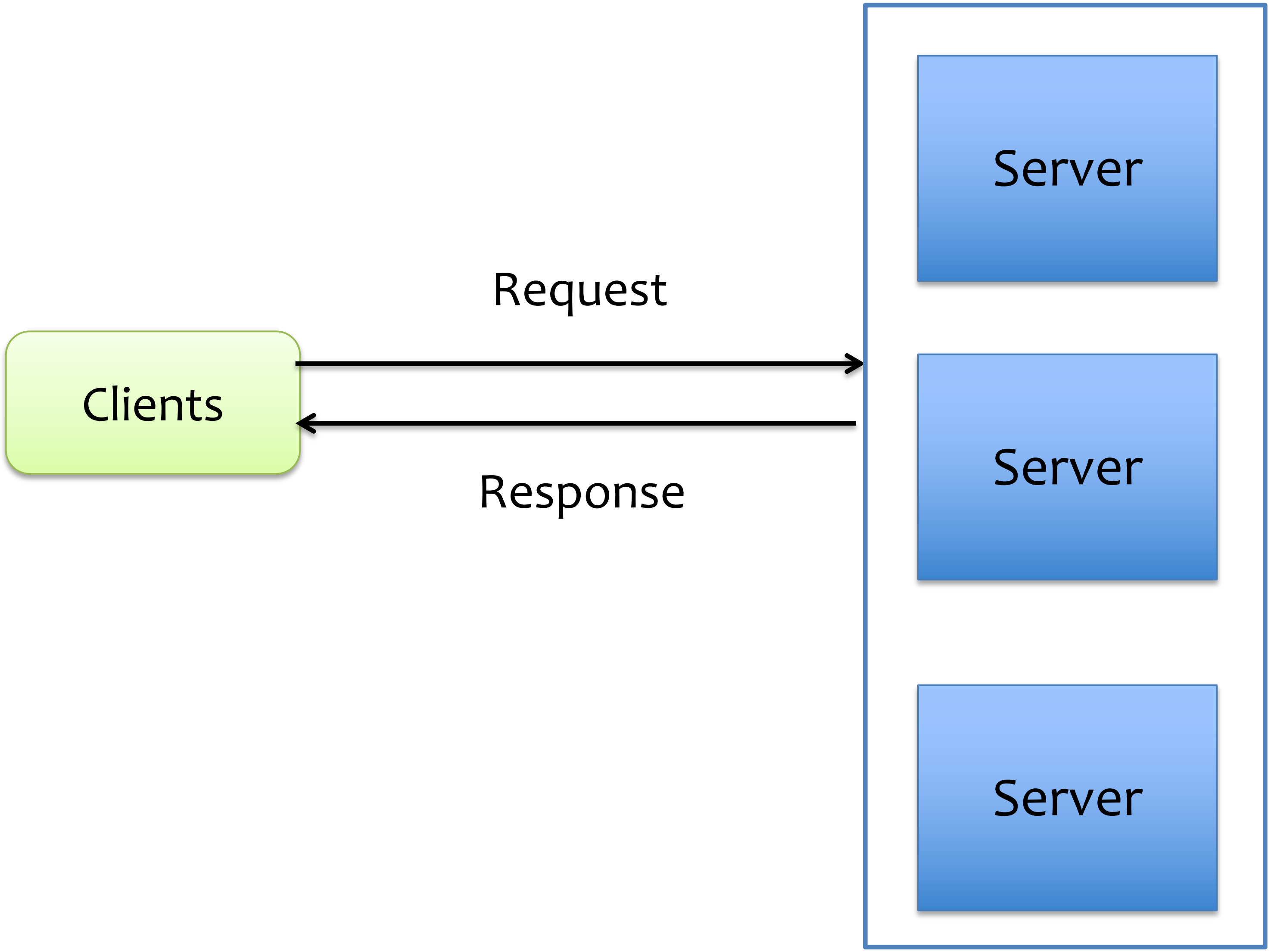
Background on
State Machine Replication

Prof. Pamod Bhatotia
https://dse.in.tum.de/bhatotia/

Fault tolerance



Replication



State machine replication

1. Implement a service as a deterministic state machine
2. Replicate server
3. Provide all replicas with the same input

Guarantees: all correct replicas will produce the same output

Consensus algorithm

Distributed Synchronization via Zookeeper

Prof. Pamod Bhatotia
https://dse.in.tum.de/bhatotia/

Co-ordination in distributed systems

- (Dynamic) configuration
- Synchronization
- Leader election
- Group membership
- Barriers
- Locks
- …

Challenges in distributed systems

- The network is unreliable
- Process may crash/fail in arbitrary ways
- The network messages may arrive arbitrarily

Zookeeper

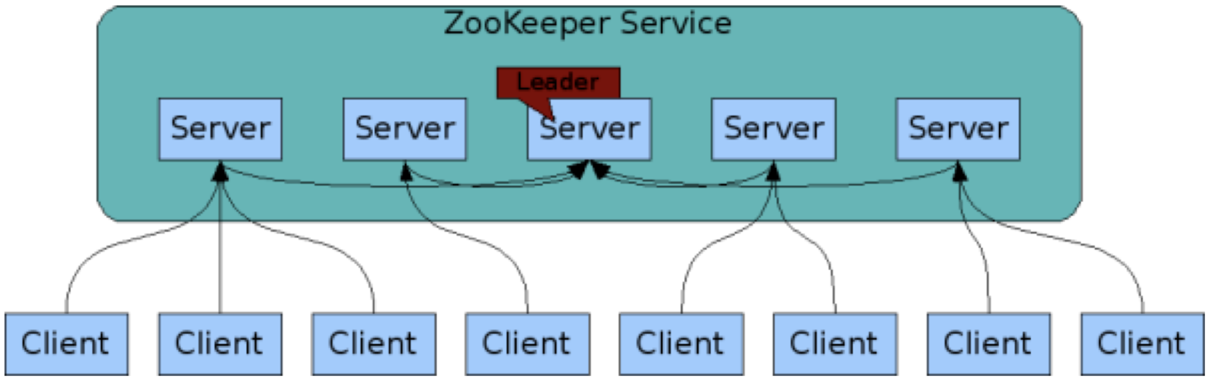
- A co-ordination (micro-)kernel
  - Minimalistic APIs that can be used to build a wide-range of co-ordination primitives
- APIs are wait-free
  - No blocking primitives in ZooKeeper
  - Blocking can be implemented by a client
  - Deadlock free!

Zookeeper design principles

- Zookeeper = FIFO ordering for clients + Linearzible writes + Wait-free APIs

- Guarantees
  - Client requests are processed in FIFO order
  - Writes to ZooKeeper are linearizable
  - Clients receive notifications of changes before the changed data becomes visible

Zookeeper architecture

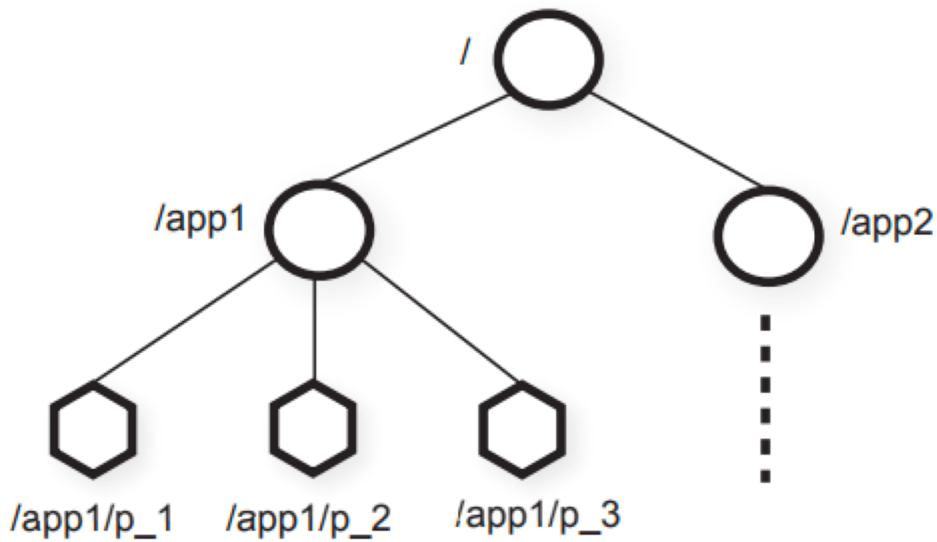


Terminology:

- **Clients:** users of the Zookeeper service
- **Server:** process providing the Zookeeper service
- **Session:** Clients establish a session when connecting to Zookeeper

Zookeeper data model

Abstraction: A set of data nodes (znodes) organized in a hierarchal namespace



Znodes can store data

znodes

- Znodes are accessed similar to UNIX filesystem namespace
- Znodes can be classified as:
  - **Regular:** created and deleted by clients explicitly
  - **Ephemeral:** can be deleted explicitly or by the system itself when the session terminates (the client that created it)
- Flags:
  - **Sequential:** montonically increasing counters
  - **Watch flag:** allows client to receive timely notification of changes without polling

Zookeeper APIs

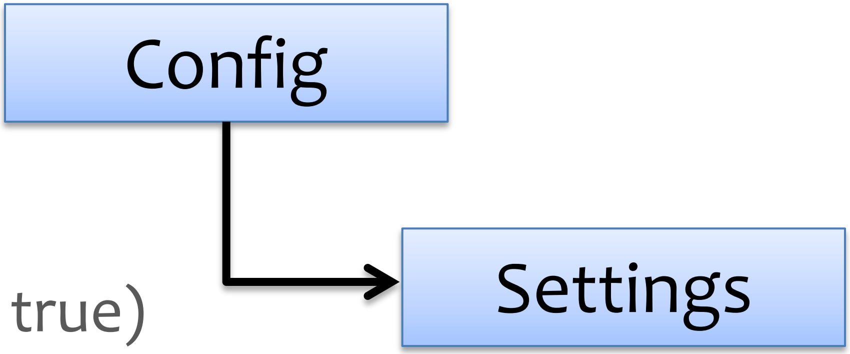
- create(path, data, flags)
- delete(path, version)
- exists(path, watch)
- getData(path, watch)
- setData(path, data, version)
- getChildren(path, watch)
- Sync()

> - Version is used for conditional update
> - Synchronous and asynchronous APIs are available for clients!

Zookeeper use-cases

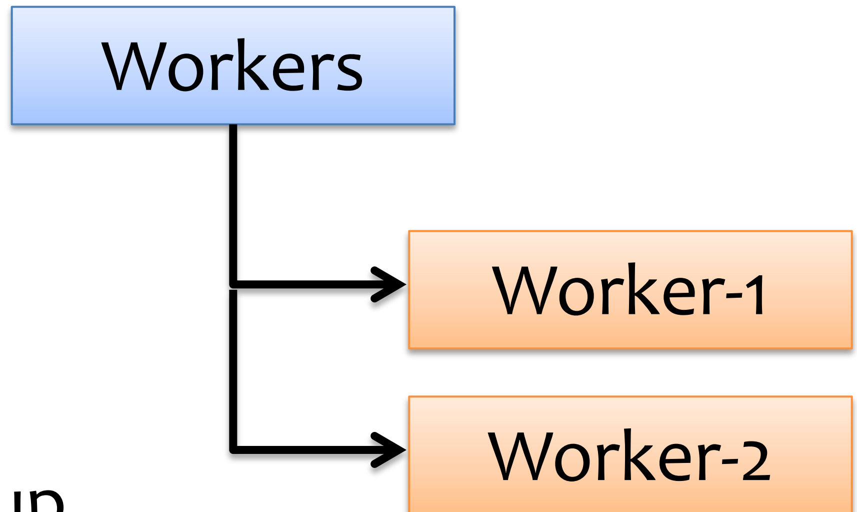
How to use Zookeeper to implement distributed co-ordination protocols?

Example 1: Configuration



1. Workers get configuration
   - getData("…/config/settings", true)
2. Admin change the config
   - setData("…/config/settings", newConf-1)
3. Workers notified of change and get the new settings
   - getData("…/config/settings", true)

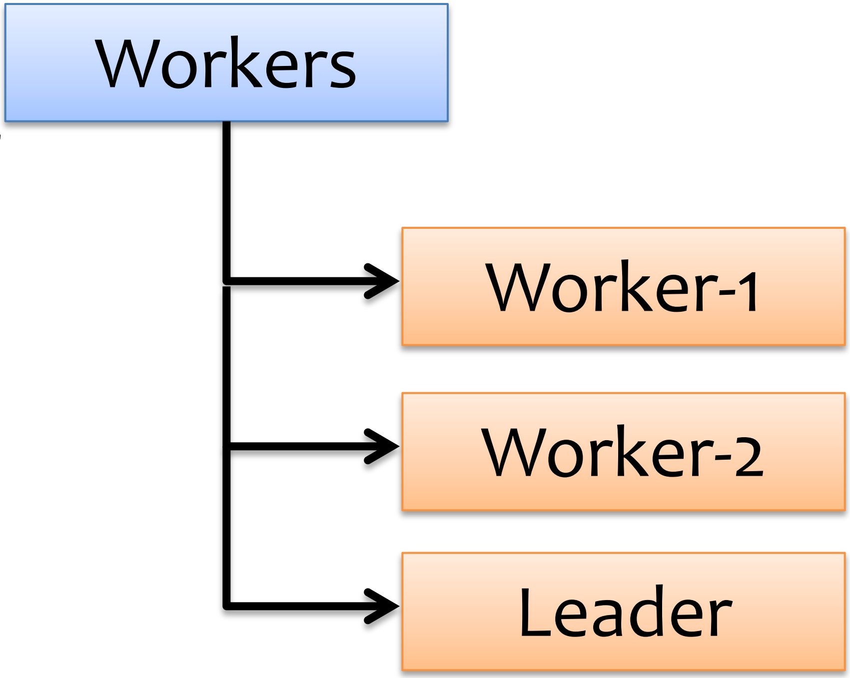
Example 2: Group membership



1. Register serverName in group
   - Create("…/workers/workerName", hostInfo, EPHEMERAL)
2. List group members
   - getChildren("…/workers", true)

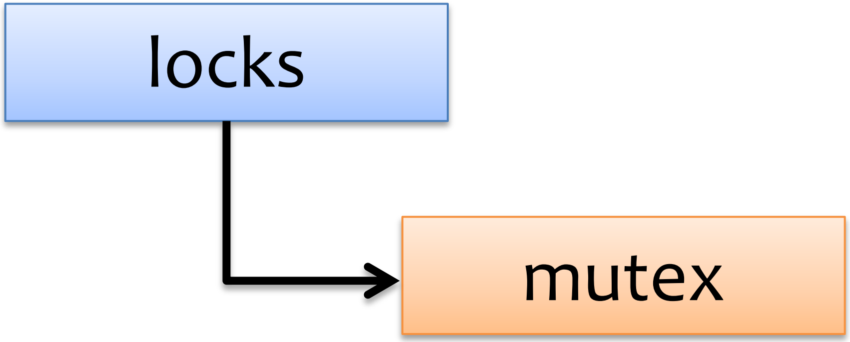
Example 3: Leader Election

1. getData("…/workers/leader", true)
2. If successful follow the leader described in the data and exit
3. create("…/workers/leader", hostname, EPHERMERAL)
4. If successful lead and exit
5. Goto step 1



Example 4: Locks

1. create("…/locks/mutex", EPHEMERAL)
2. If succeed then lock acquired
3. Else, getData("…/locks/mutex", true)
4. Goto step 1



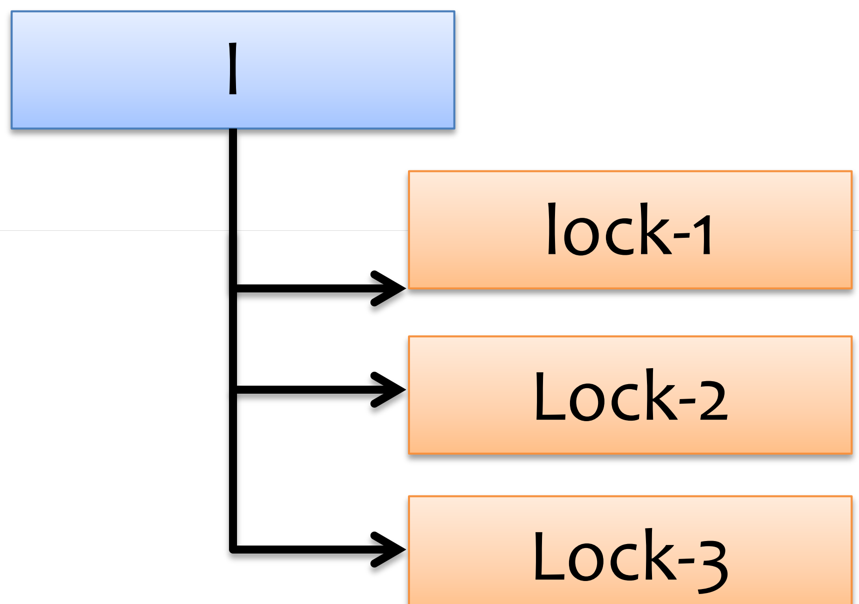
Example 5: Locks without herding

Lock

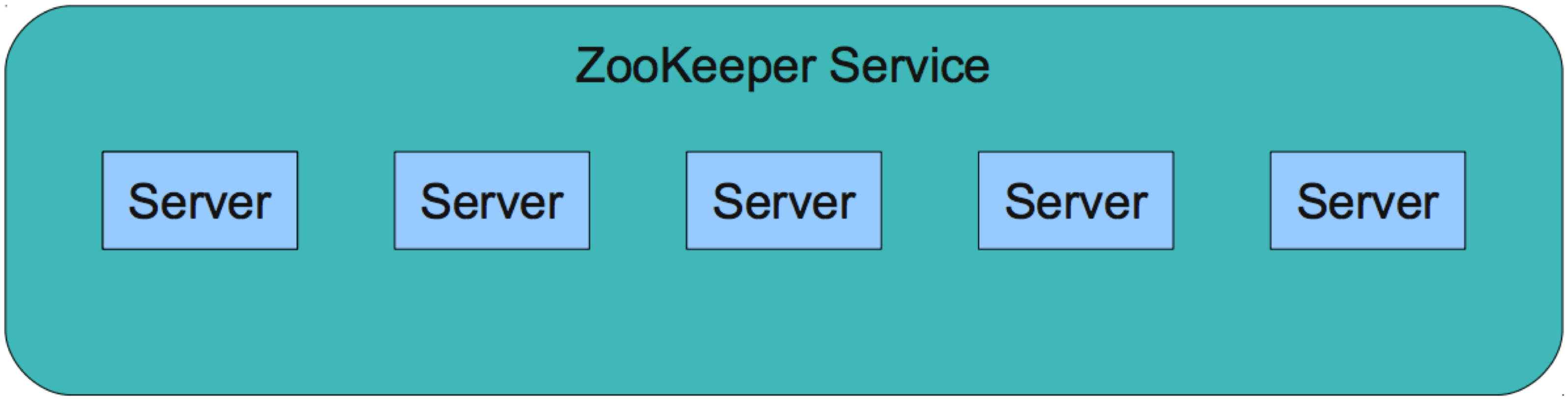
```
1 n = create(l + "/lock-", EPHEMERAL|SEQUENTIAL)
2 C = getChildren(l, false)
3 if n is lowest znode in C, exit
4 p = znode in C ordered just before n
5 if exists(p, true) wait for watch event
6 goto 2
```

Unlock

```
1 delete(n)
```



System Implementation



- All servers have a copy of the state in memory
- A leader is elected at startup
- Followers service clients, all updates go through leader
- Update responses are sent when a majority of servers have persisted the change
  - We need 2f+1 machines to tolerate f failures

Summary

- Apache Zookeeper
  - Co-ordination in distributed systems
  - A distributed co-ordination kernel
  - Usage to build powerful primitives

- Resources:
  - Compulsory reading: Zookeeper [ATC'10]:
    - Website: https://zookeeper.apache.org
    - Paper: https://www.usenix.org/legacy/event/atc10/tech/full_papers/Hunt.pdf
  - Recommended reading:
    - Chubby [OSDI'06]: https://research.google.com/archive/chubby.html
    - Zab [DSN'11]: https://dl.acm.org/citation.cfm?id=2056409
    - Wait-free synchronization [TOPLAS'91]: https://dl.acm.org/citation.cfm?id=102808

Back up slides