



---

**ZooKeeper**  
Yahoo! Research  
NFIC 2010



## Cloud Computing

---

- Elastic – size changes dynamically
- Scale – large number of servers



# Coordination is important







# Coordination is important





# Internet-scale Challenges

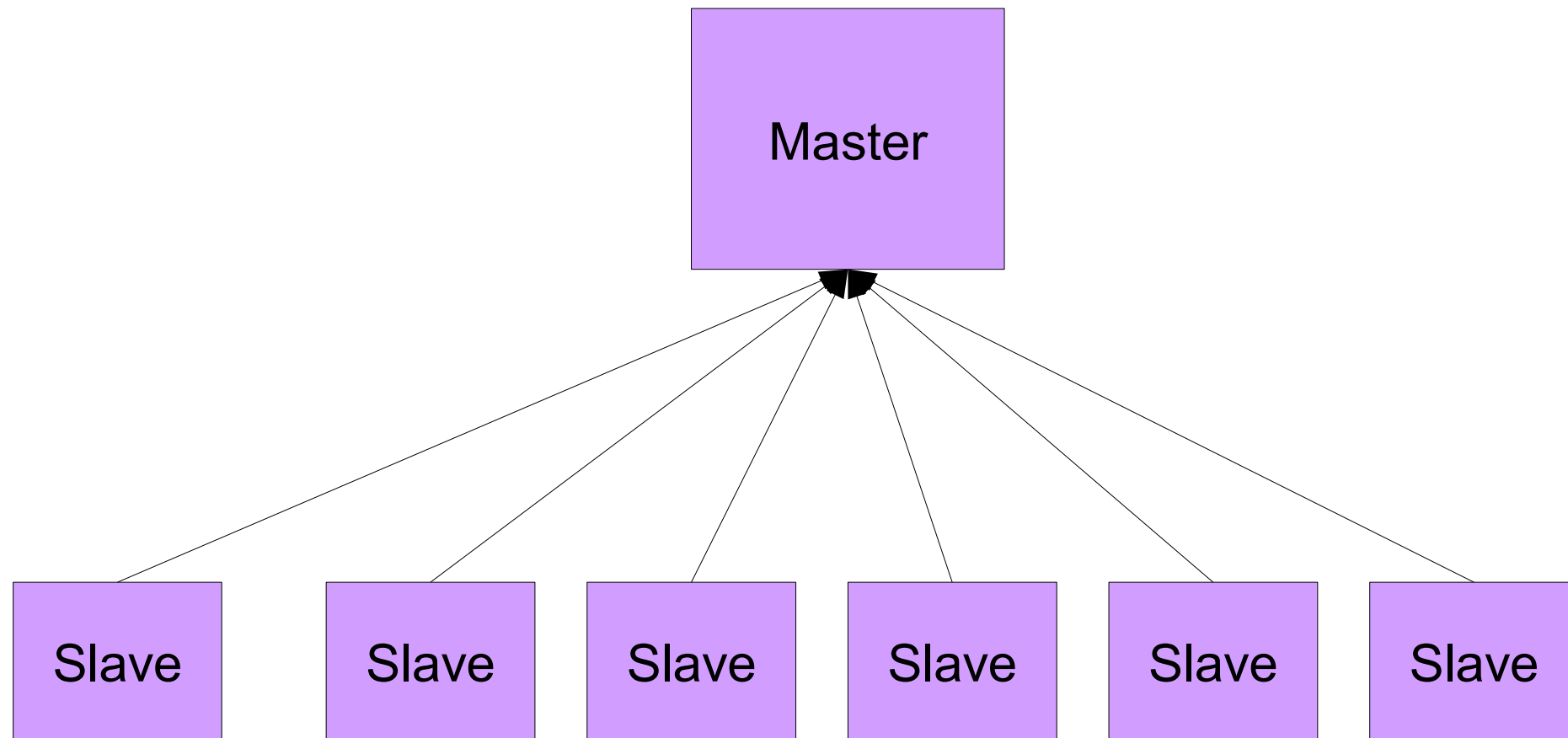


- Lots of servers, users, data
- FLP, CAP
- Mere mortal programmers



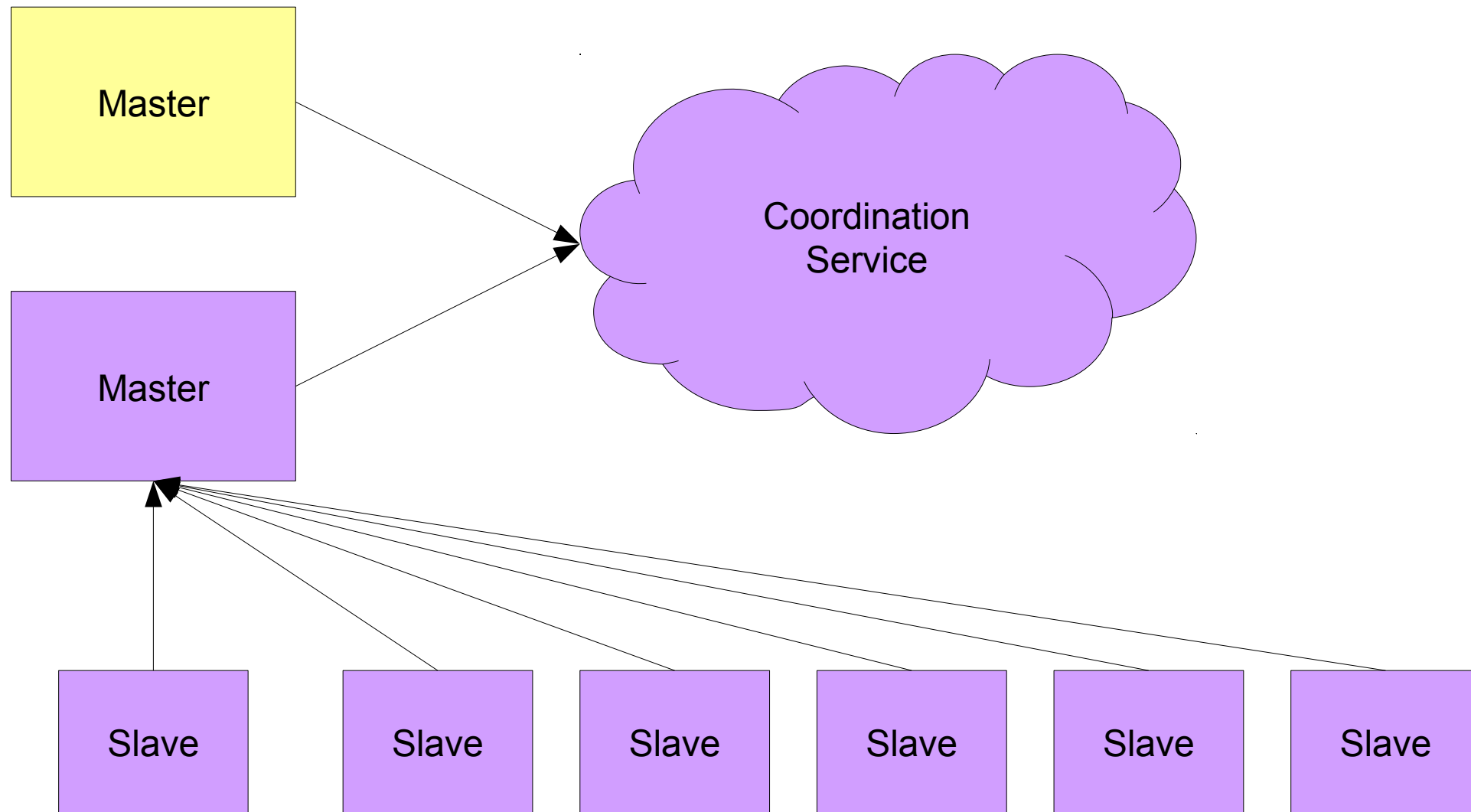


# Classic Distributed System



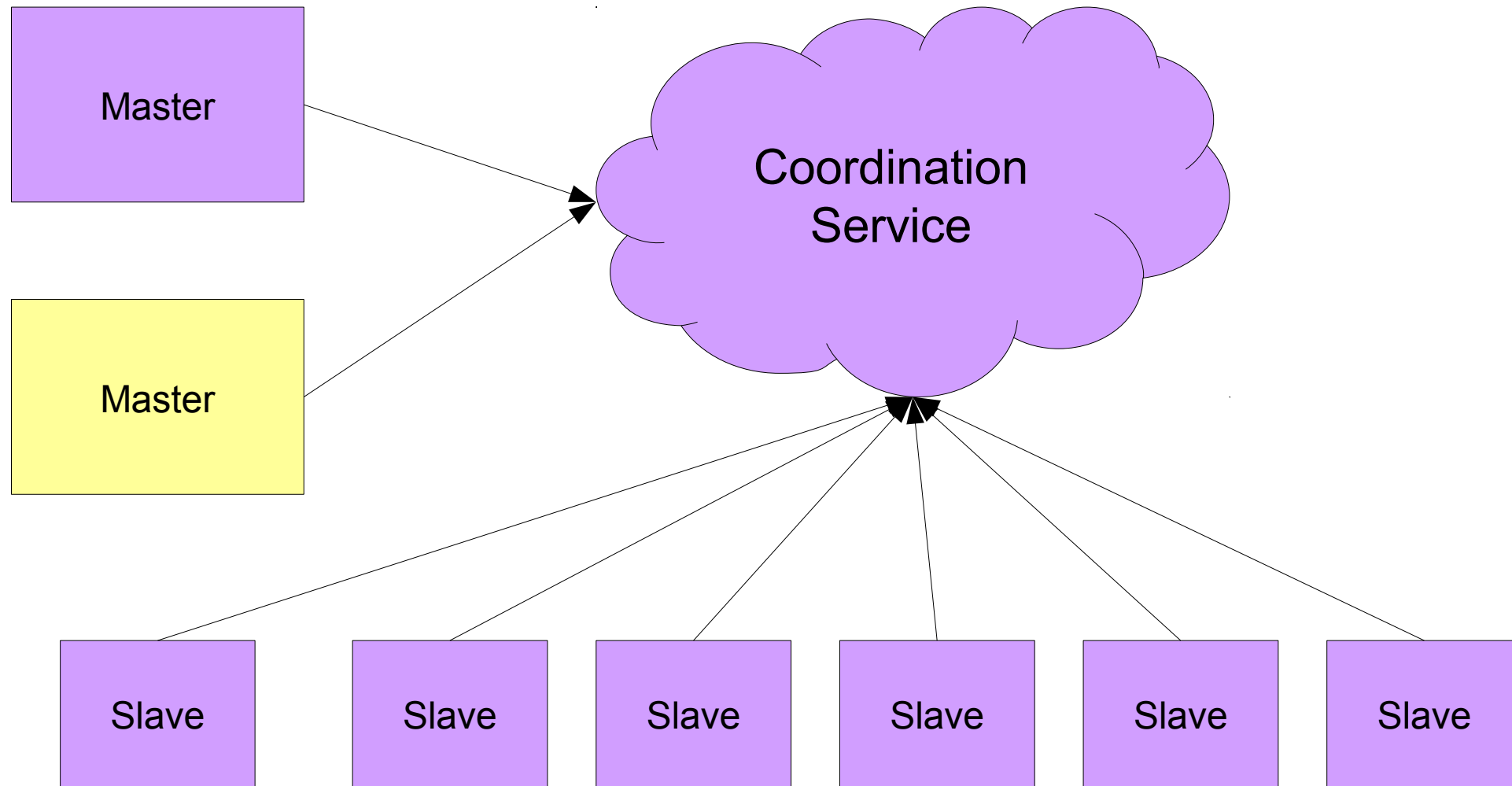


# Fault Tolerant Distributed System





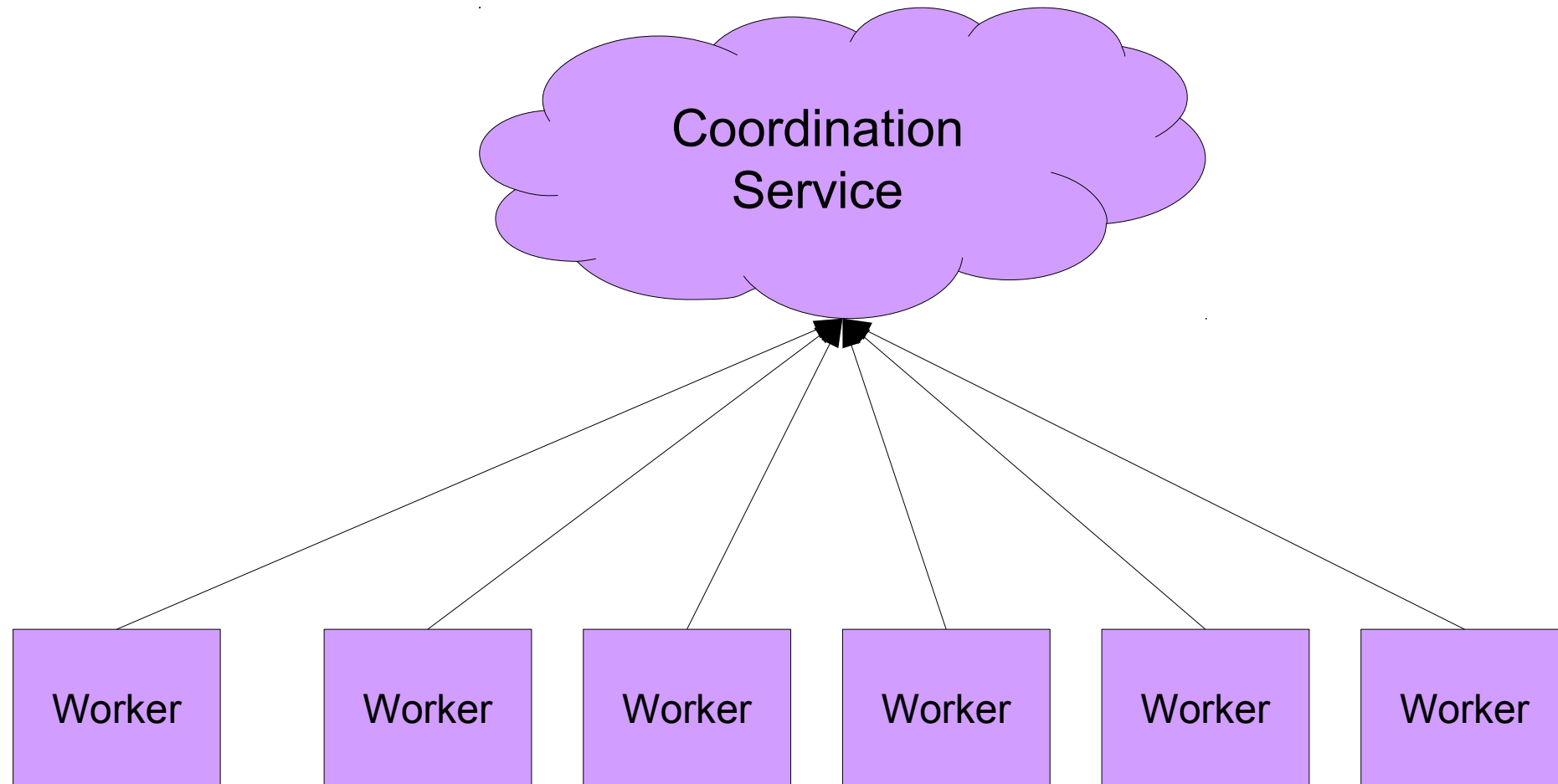
# Fault Tolerant Distributed System







# Fully Distributed System





# What is coordination?

- Group membership
- Leader election
- Dynamic Configuration
- Status monitoring
- Queuing
- Barriers
- Critical sections



## Goals

- Been done in the past
  - ISIS, distributed locks (Chubby, VMS)
- High Performance
  - Multiple outstanding ops
  - Read dominant
- General (Coordination Kernel)
- Reliable
- Easy to use



wait-free

- Pros
  - Slow processes cannot slow down fast ones
  - No deadlocks
  - No blocking in the implementations
- Cons
  - Some coordination primitives are blocking
  - Need to be able to efficiently wait for conditions



# Serializable vs Linearizability

- Linearizable writes
- Serializable read (may be stale)
- Client FIFO ordering





## Change Events

---

- Clients request change notifications
- Service does timely notifications
- Do not block write requests
- Clients get notification of a change before they see the result of a change



**Solution**

---

Order + wait-free + change events = coordination



## ZooKeeper API

String create(path, data, acl, flags)

void delete(path, expectedVersion)

Stat setData(path, data, expectedVersion)

(data, Stat) getData(path, watch)

Stat exists(path, watch)

String[] getChildren(path, watch)

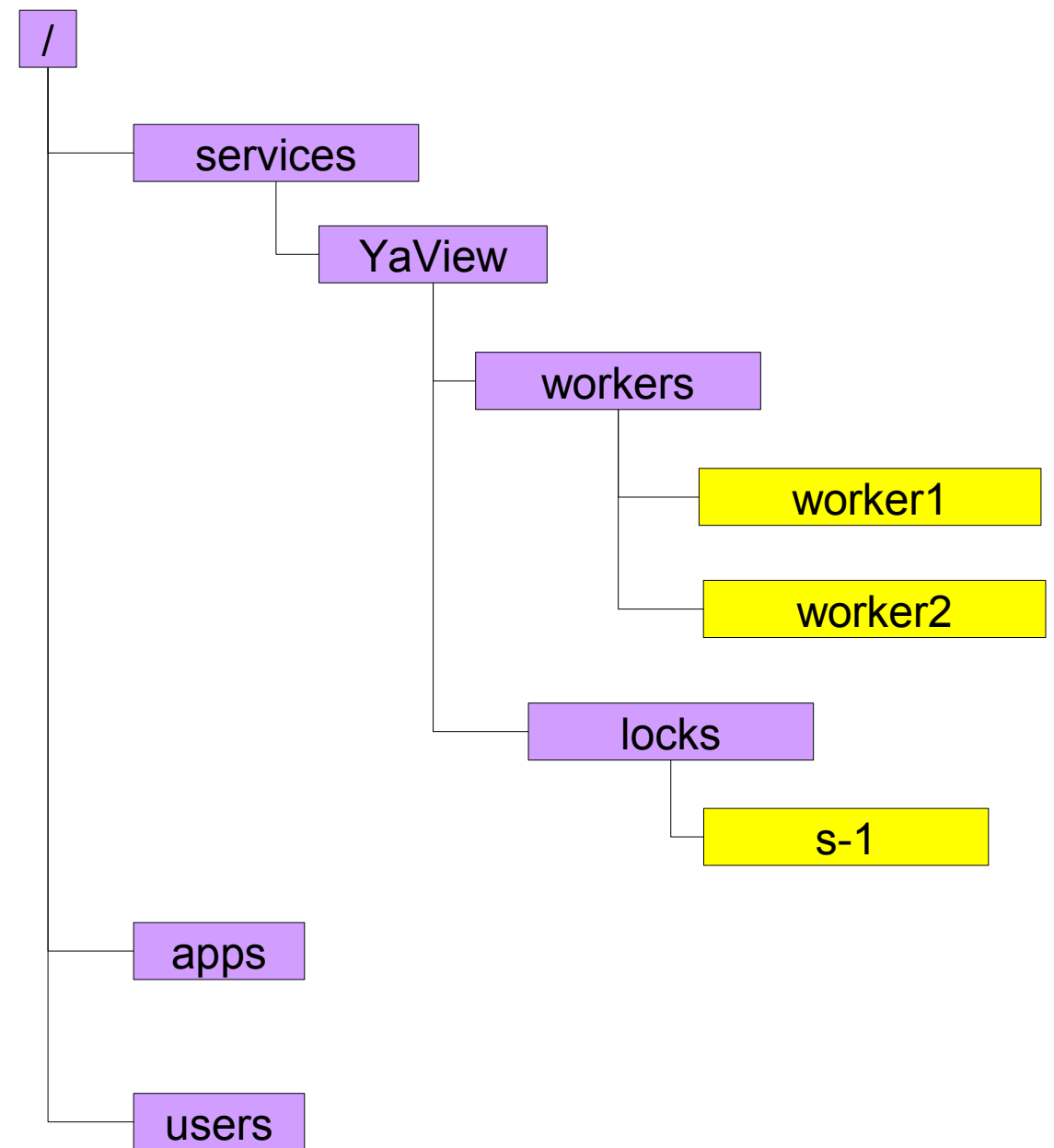
void sync()

Stat setACL(path, acl, expectedVersion)

(acl, Stat) getACL(path)

# Data Model

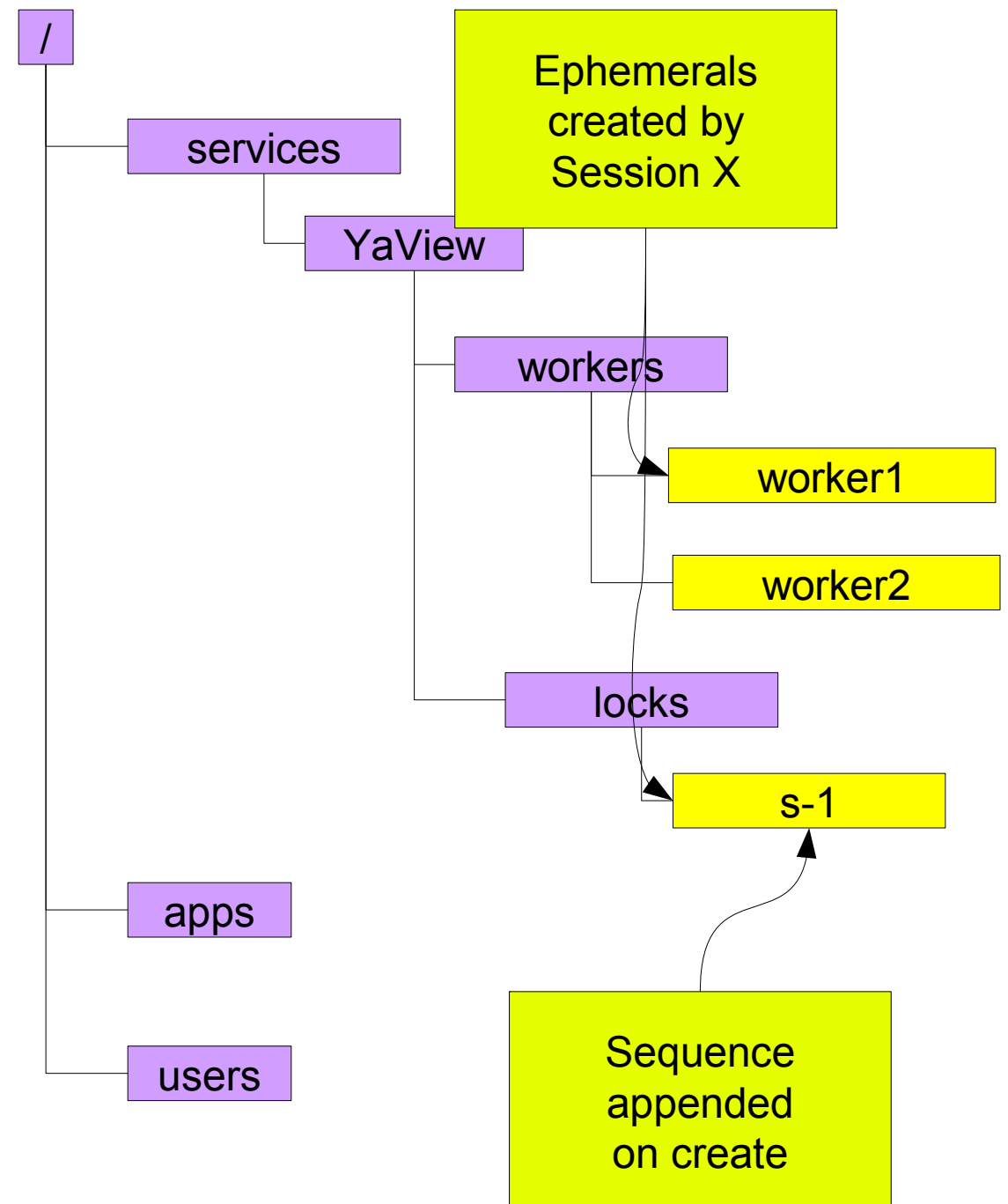
- Hierarchical namespace (like a file system)
- Each znode has data and children
- data is read and written in its entirety





# Create Flags

- Ephemeral: znode deleted when creator fails or explicitly deleted
- Sequence: append a monotonically increasing counter

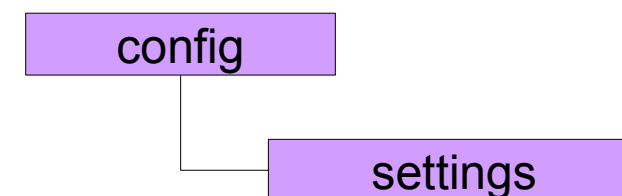






# Configuration

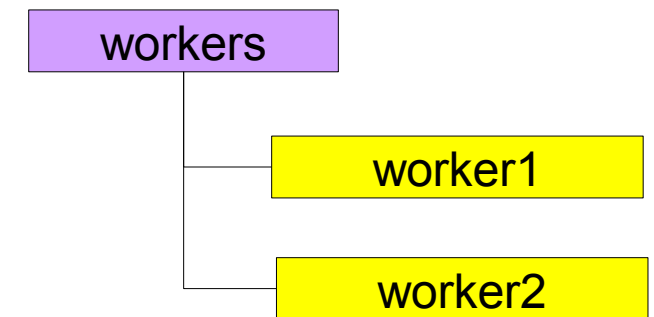
- Workers get configuration
  - `getData("../config/settings", true)`
- Administrators change the configuration
  - `setData("../config/settings", newConf, -1)`
- Workers notified of change and get the new settings
  - `getData("../config/settings", true)`





## Group Membership

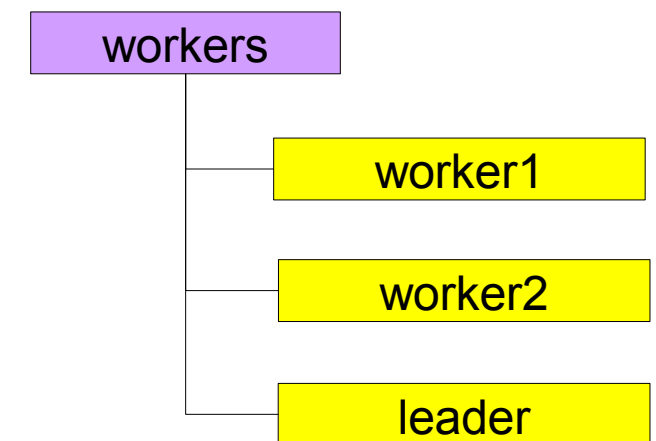
- Register serverName in group
  - create(“.../workers/workerName”, hostInfo, EPHEMERAL)
- List group members
  - listChildren(“.../workers”, true)





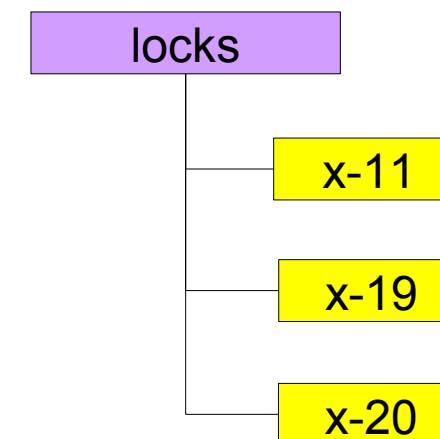
# Leader Election

- 1 `getData(.../workers/leader", true)`
- 2 if successful follow the leader described in the data and exit
- 3 `create(.../workers/leader", hostname, EPHEMERAL)`
- 4 if successful lead and exit
- 5 goto step 1



If a watch is triggered for `.../workers/leader`, followers will restart the leader election process

- 1 id = create("../locks/x-", SEQUENCE|EPHEMERAL)
- 2 getChildren("../locks"/, false)
- 3 if id is the 1<sup>st</sup> child, exit
- 4 exists(name of last child before id, true)
- 5 if does not exist, goto 2)
- 6 wait for event
- 7 goto 2)

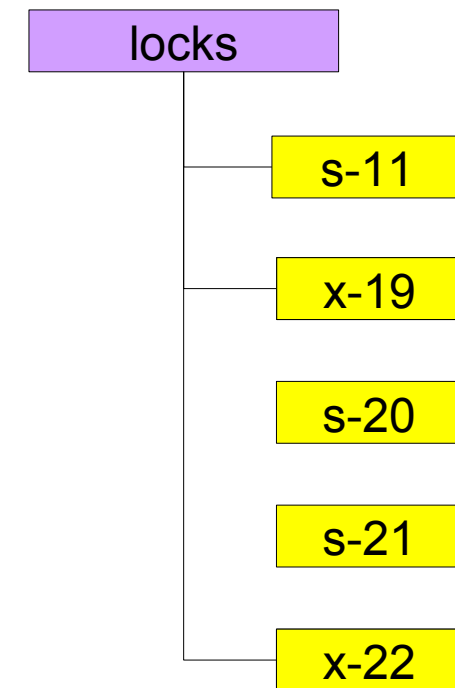


Each znode watches one other.  
No herd effect.



# Shared Locks

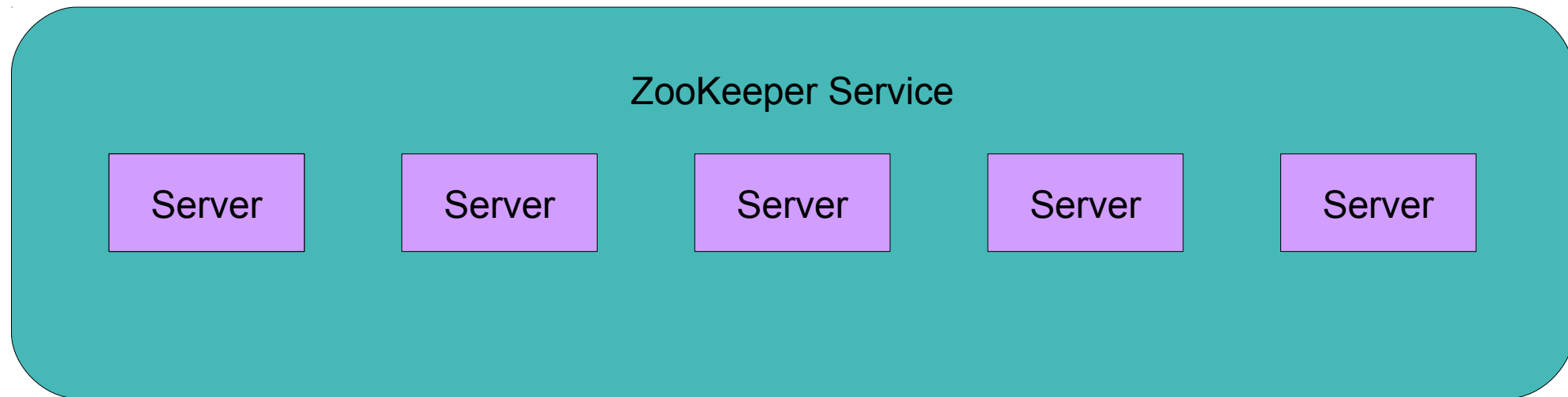
- 1 id = create("../locks/s-", SEQUENCE|EPHEMERAL)
- 2 getChildren("../locks"/, false)
- 3 if no children that start with x- before id, exit
- 4 exists(name of the last x- before id, true)
- 5 if does not exist, goto 2)
- 6 wait for event
- 7 goto 2)







# ZooKeeper Servers

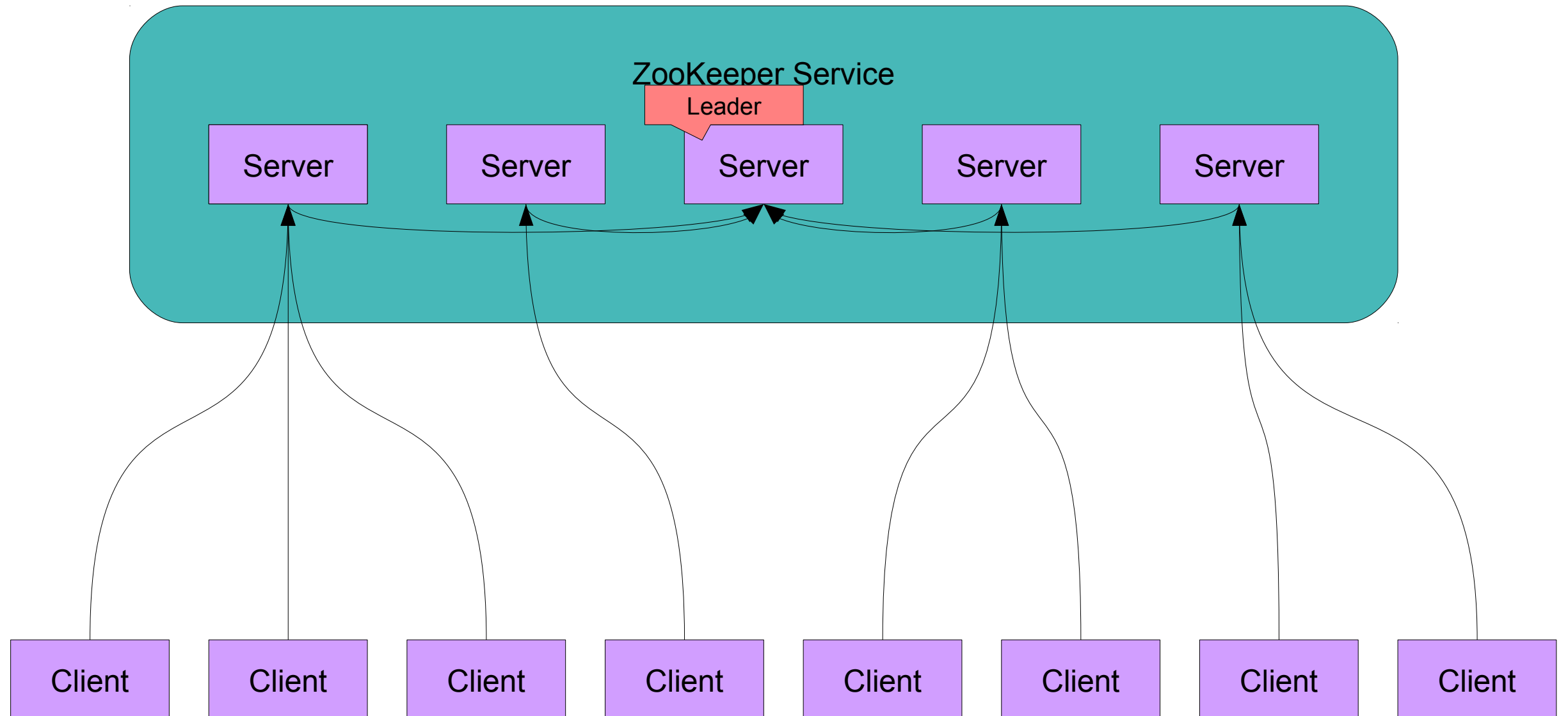


- All servers have a copy of the state in memory
- A leader is elected at startup
- Followers service clients, all updates go through leader
- Update responses are sent when a majority of servers have persisted the change

We need  $2f+1$  machines to tolerate  $f$  failures



# ZooKeeper Servers





## Evaluation & Experience



# Evaluation

Cluster of 50 servers

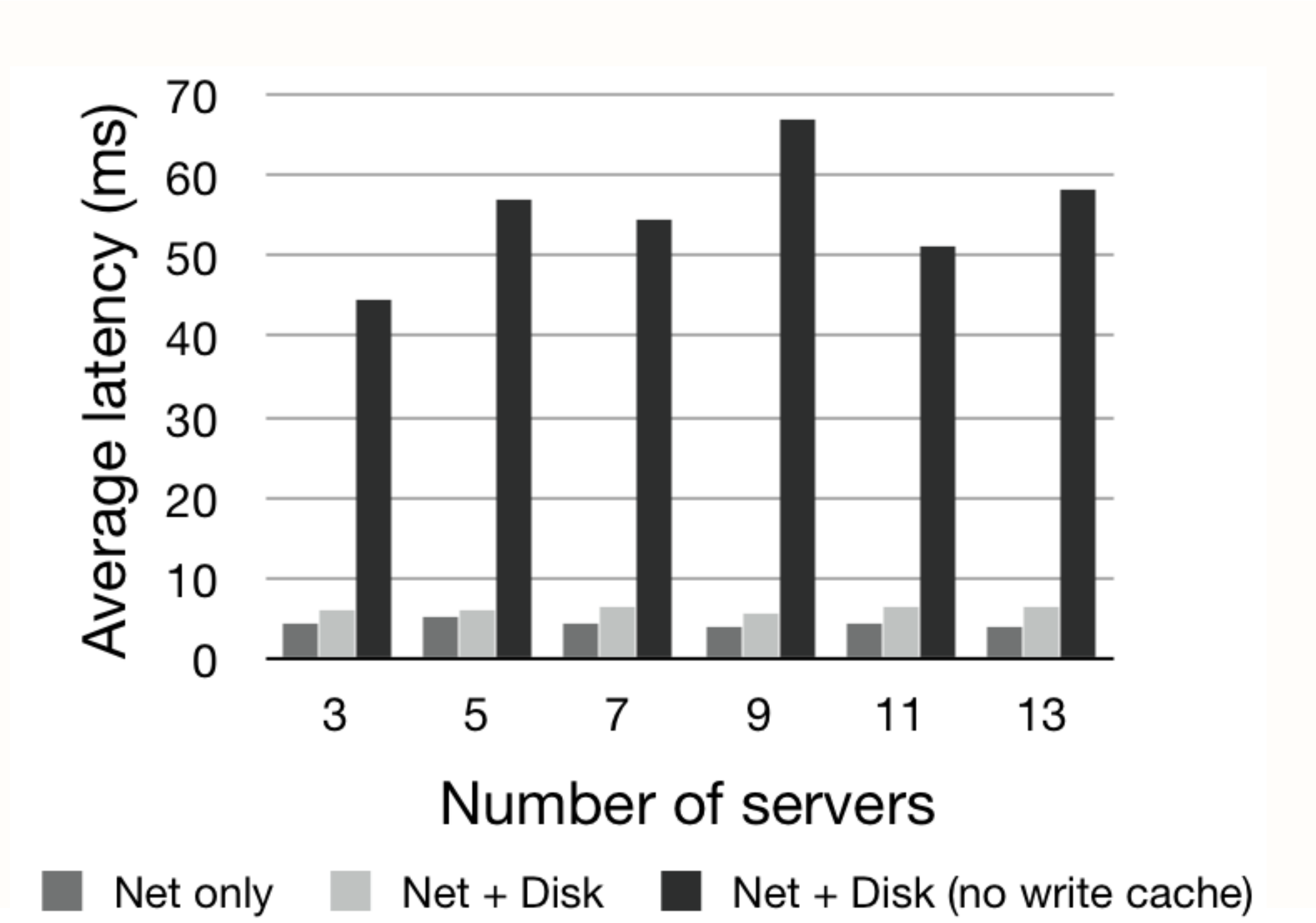
Xeon dual-core 2.1 GHz

4 GB of RAM

Two SATA disks



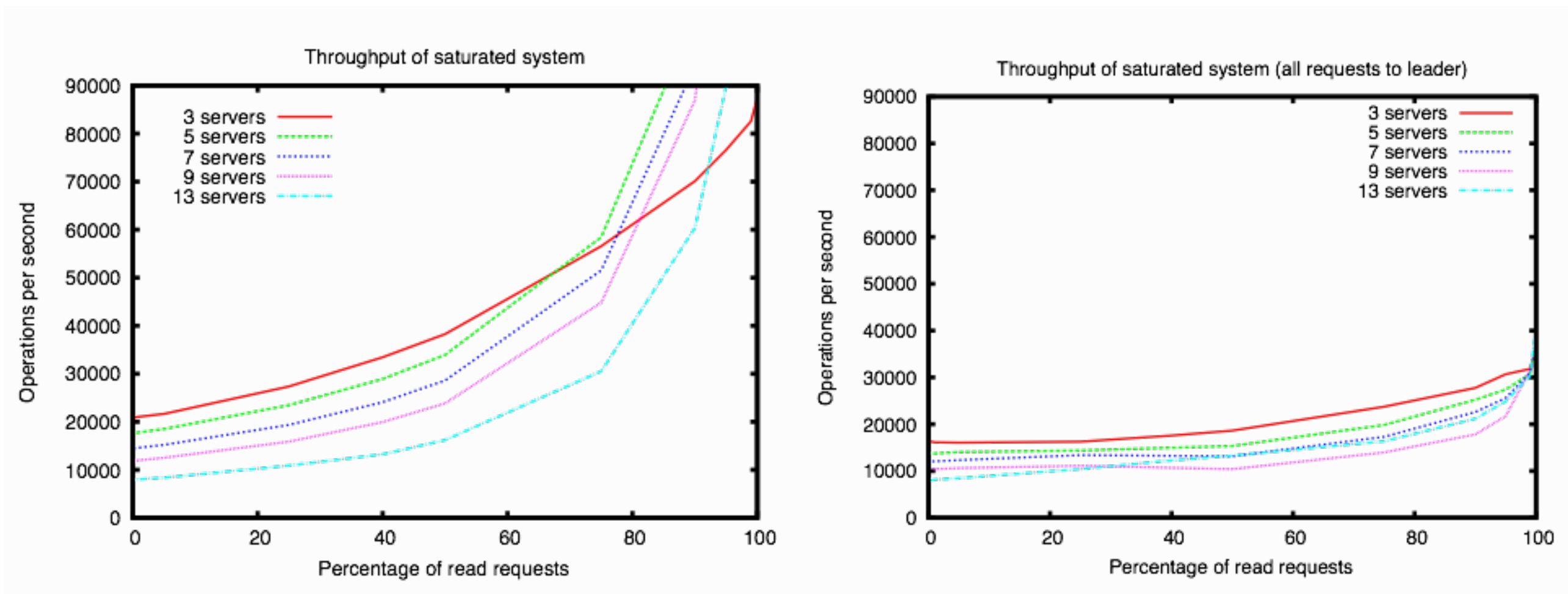
# Latency





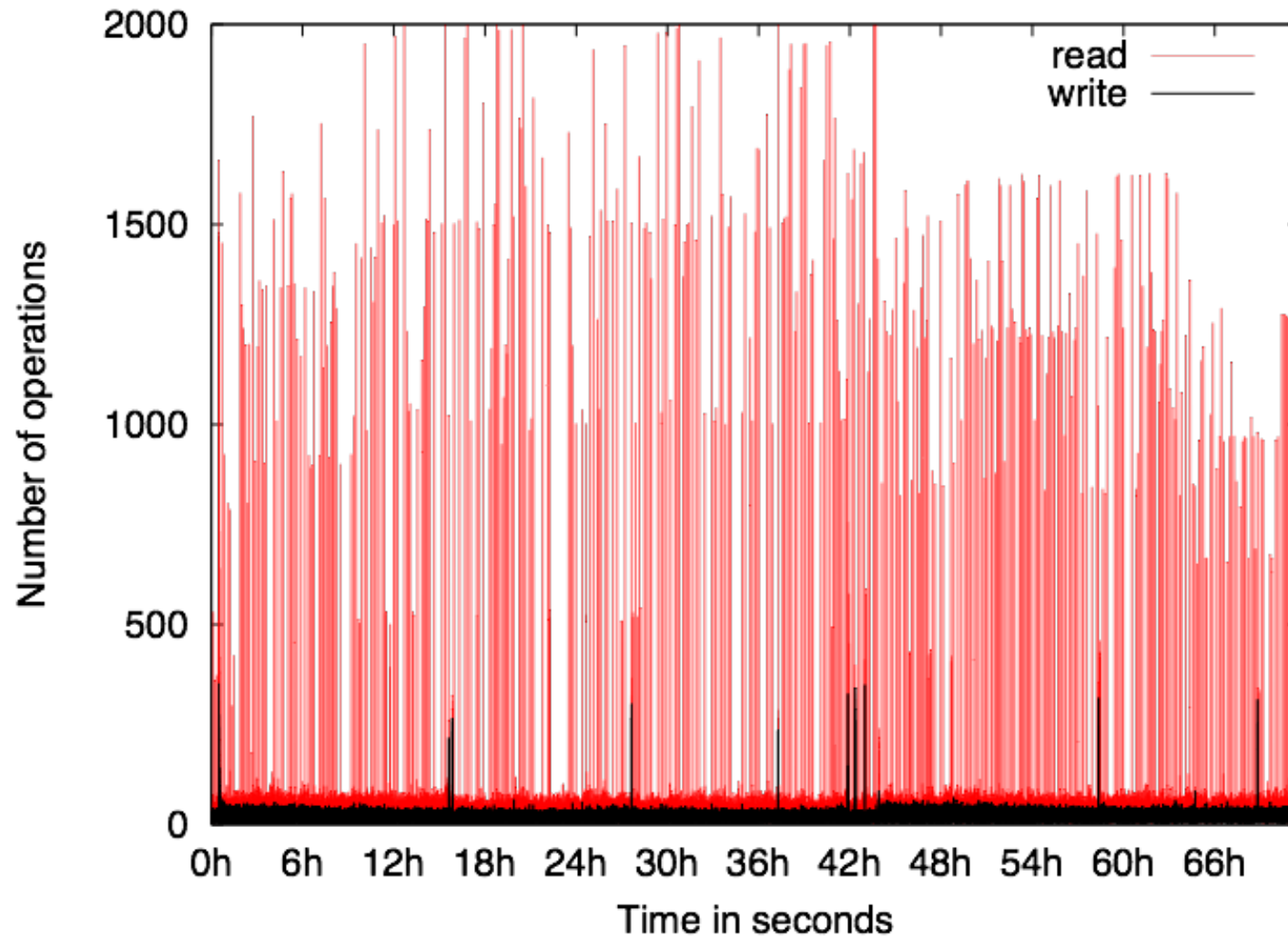


# Throughput





# Load in production (Fetching Service)





## At Yahoo!...

- Has been used for:
  - Cross-datacenter locks (wide-area locking)
  - Web crawling
  - Large-scale publish-subscribe (Hedwig: ZOOKEEPER-775)
  - Portal front-end
- Largest cluster I'm aware of
  - Thousands of clients



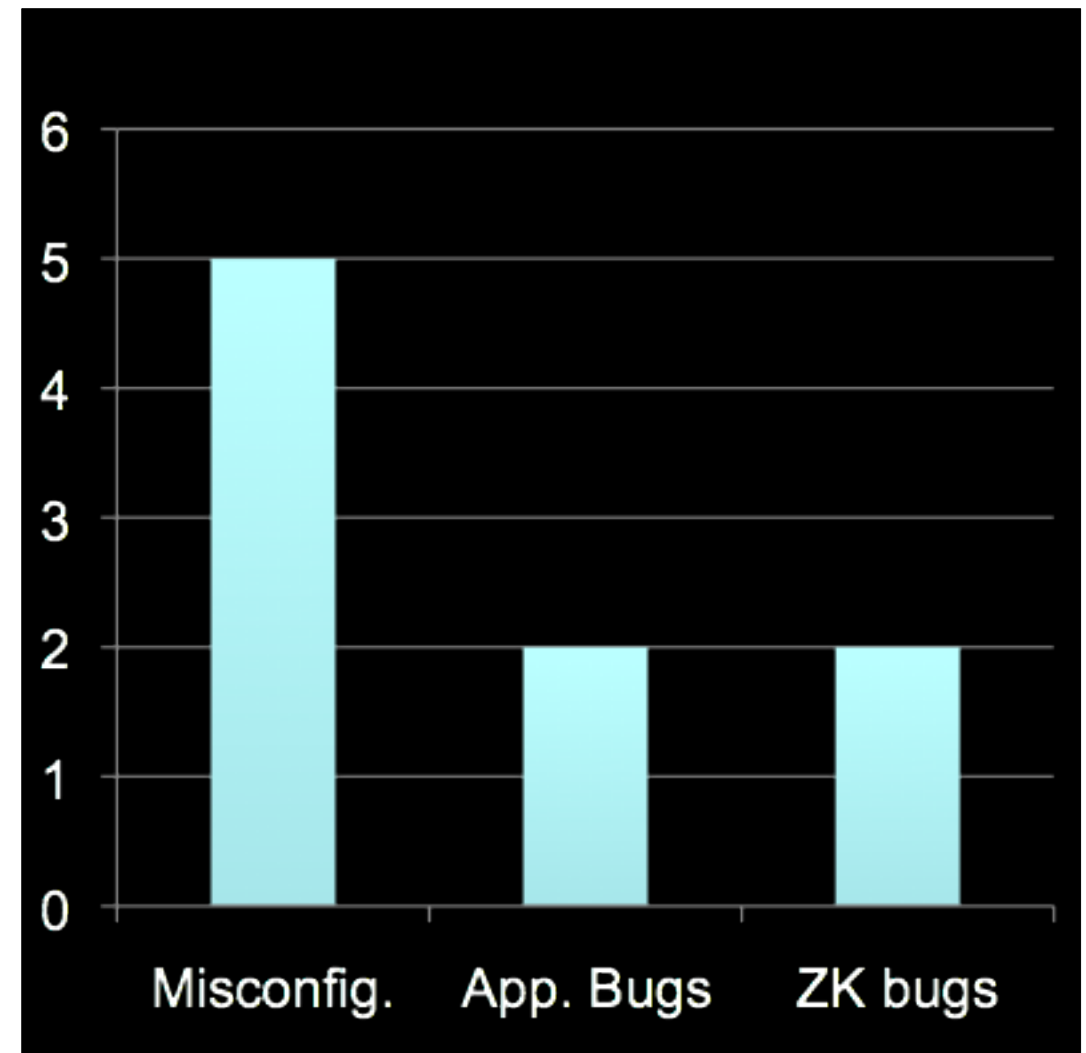
## Faults in practice

- Bugzilla
  - Ticket system for software defects, improvements, etc
- Fetching service queue
  - Over 2 years running
  - 9 tickets reporting issues with ZooKeeper



# Faults in practice

- **Misconfiguration: 5 issues**
  - System configuration, not ZK
  - *E.g.*, misconfigured net cards, DNS clash
- **Application bugs: 2 issues**
  - Misunderstanding of the API semantics
  - *E.g.*, race condition using async api
- **ZK bugs: 2 issues**
  - Really our fault
  - API and server (affected all)





## Summary

---

- Easy to use
- High Performance
- General
- Reliable
- Release 3.3 on Apache
  - See <http://hadoop.apache.org/zookeeper>
  - Committers from Yahoo! and Cloudera