Design of Zookeeper

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

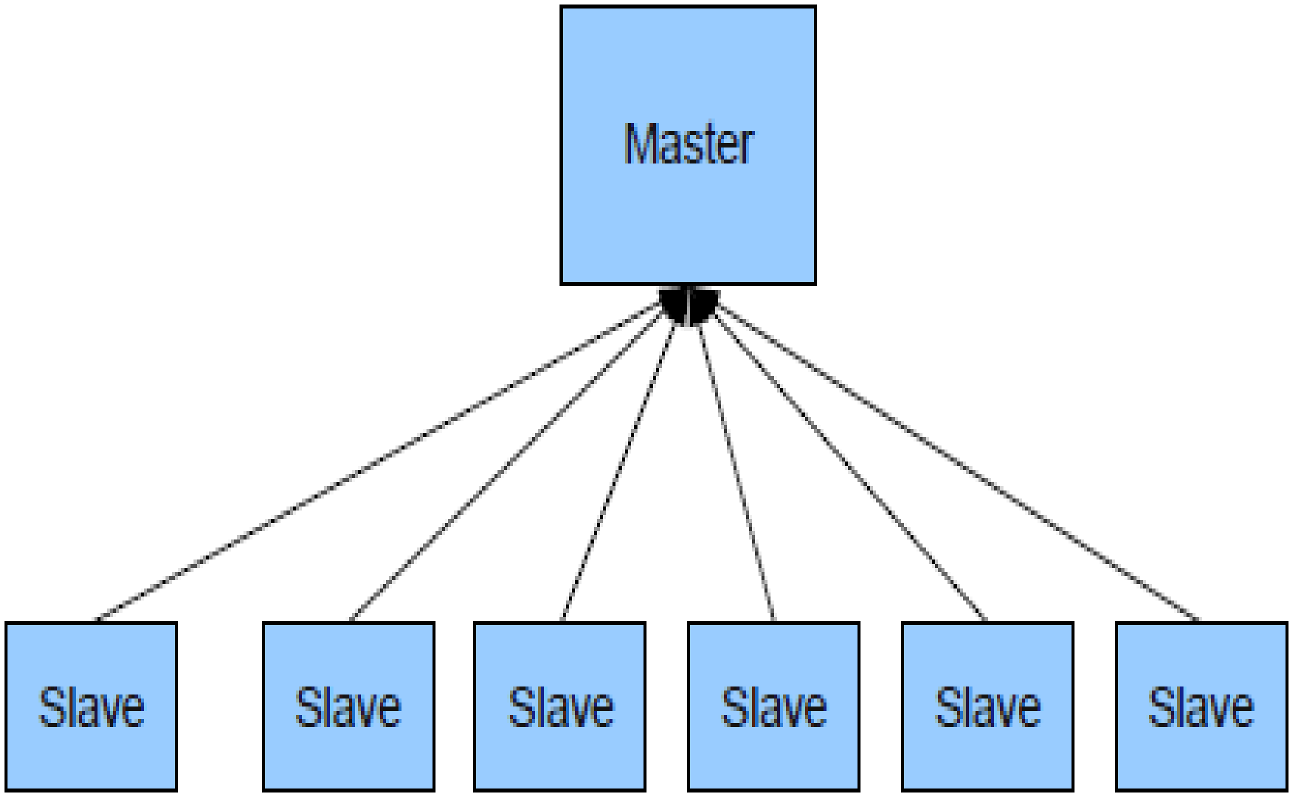
- In this lecture, we will discuss the '**design of ZooKeeper'**, which is a service for coordinating processes of distributed applications.
- We will discuss its basic fundamentals, design goals, architecture and applications.

https://zookeeper.apache.org/

ZooKeeper, why do we need it?

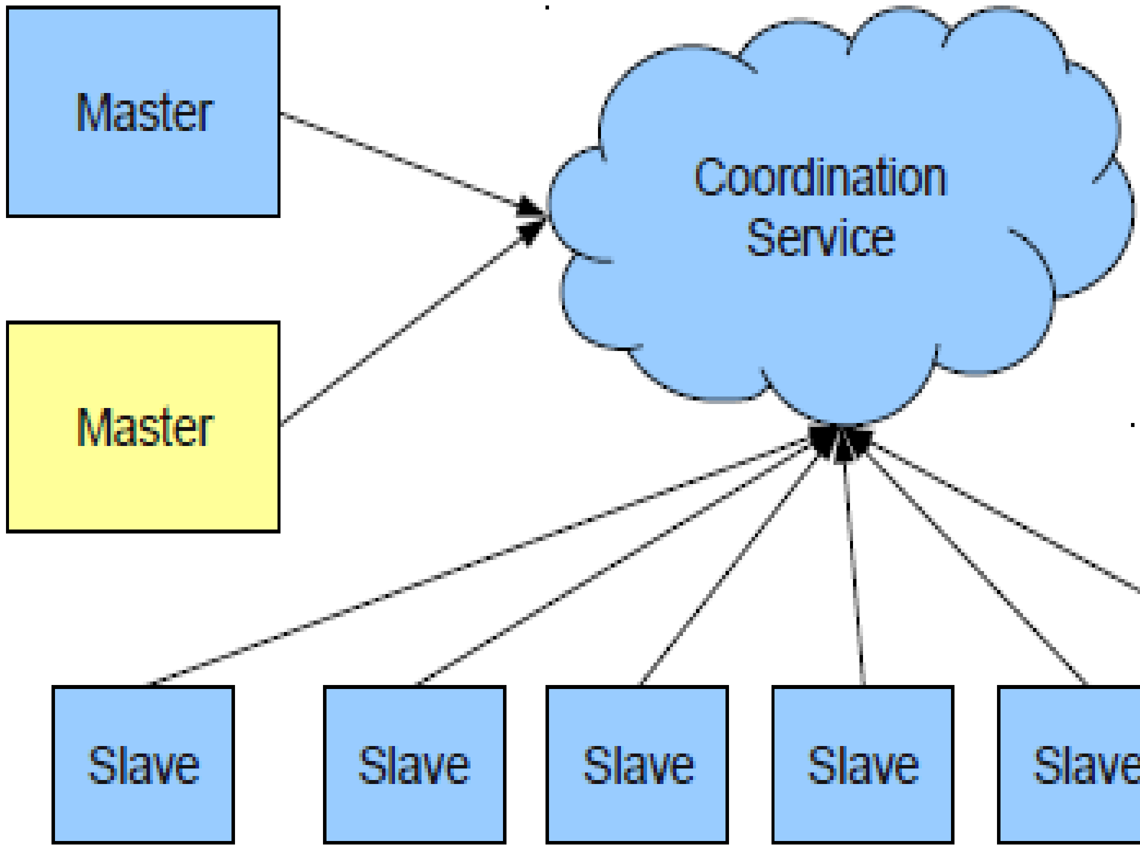
- Coordination is important

Classic Distributed System



- Most of the system like HDFS have one Master and couple of slave nodes and these slave nodes report to the master.

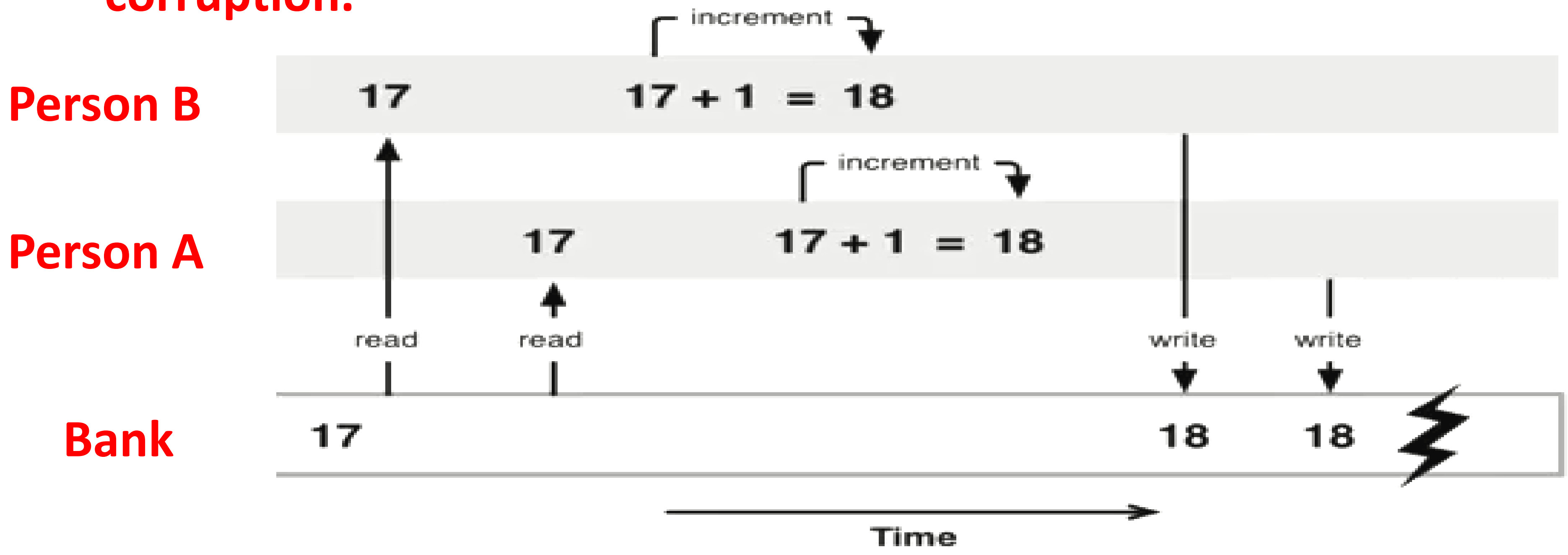
Fault Tolerant Distributed System



- Real distributed fault tolerant system have Coordination service, Master and backup master.
- If primary failed then backup works for it.

What is a Race Condition?

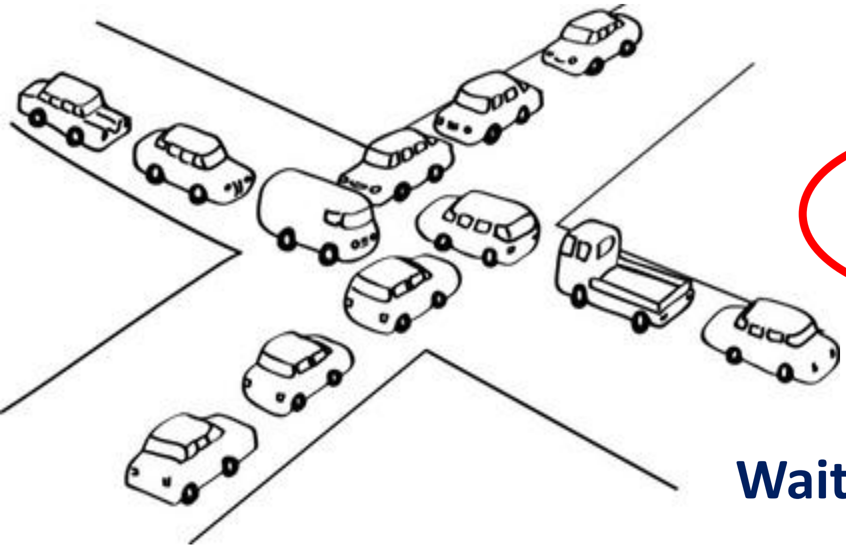
- When two processes are competing with each other causing **data corruption.**

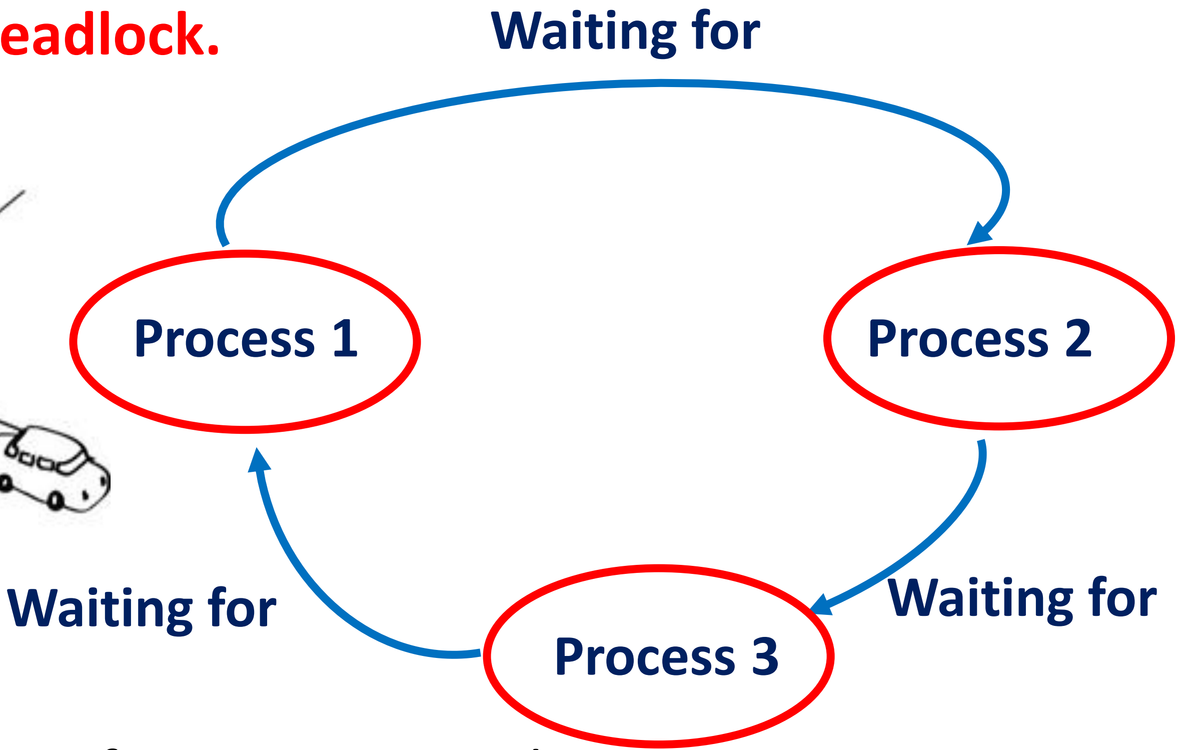


As shown in the diagram, two persons are trying to deposit 1 rs. online into the same bank account. The initial amount is 17 rs. Due to race conditions, the final amount in the bank is 18 rs. instead of 19.

What is a Deadlock?

- When two processes are waiting for each other directly or indirectly, it is called **deadlock.**

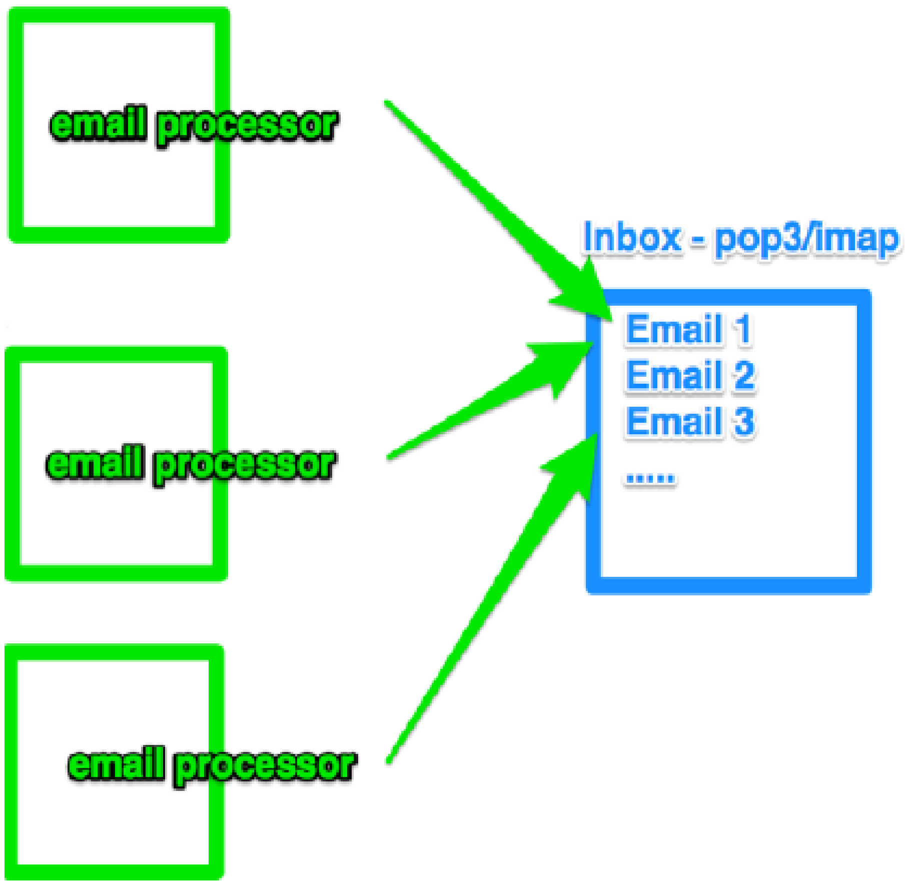




- Here, Process 1 is waiting for process 2 and process 2 is waiting for process 3 to finish and process 3 is waiting for process 1 to finish. All these three processes would keep waiting and will never end. This is called dead lock.

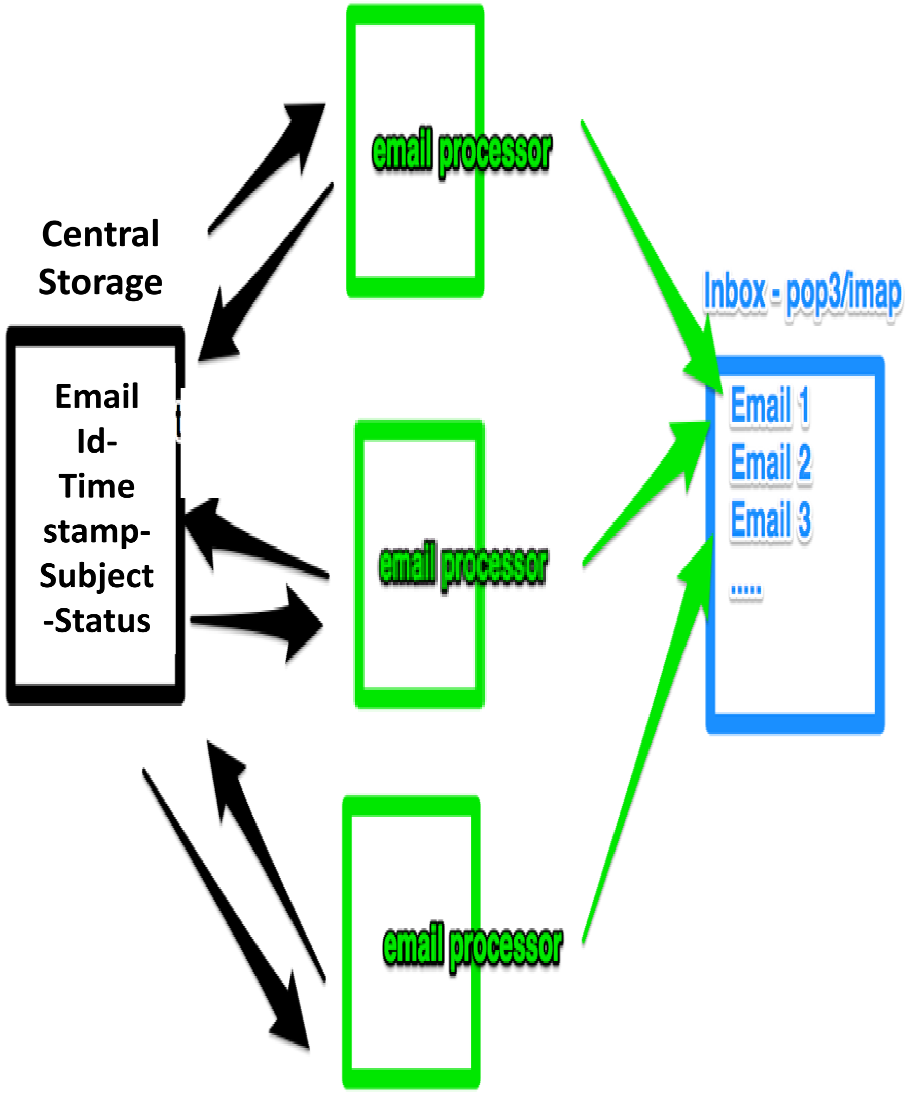
What is Coordination ?

- **How would Email Processors avoid reading same emails?**
- Suppose, there is an inbox from which we need to index emails.
- Indexing is a heavy process and might take a lot of time.
- Here, we have multiple machine which are indexing the emails. Every email has an id. You can not delete any email. You can only read an email and mark it read or unread.
- Now how would you handle the coordination between multiple indexer processes so that every email is indexed?



What is Coordination ?

- If indexers were running as multiple threads of a single process, it was easier by the way of using synchronization constructs of programming language.
- But since there are multiple processes running on multiple machines which need to coordinate, we need a central storage.
- This central storage should be safe from all concurrency related problems.
- **This central storage is exactly the role of Zookeeper.**



What is Coordination ?

- **Group membership:** Set of datanodes (tasks) belong to same group
- **Leader election:** Electing a leader between primary and backup
- **Dynamic Configuration:** Multiple services are joining, communicating and leaving (Service lookup registry)
- **Status monitoring:** Monitoring various processes and services in a cluster
- **Queuing:** One process is embedding and other is using
- **Barriers:** All the processes showing the barrier and leaving the barrier.
- **Critical sections:** Which process will go to the critical section and when?

What is ZooKeeper ?

- **ZooKeeper is a highly reliable distributed coordination kernel**, which can be used for distributed locking, configuration management, leadership election, work queues,….
- Zookeeper is a replicated service that holds the metadata of distributed applications.
- **Key attributed of such data**
  - **Small size**
  - **Performance sensitive**
  - **Dynamic**
  - **Critical**
- **In very simple words,** it is a central store of key-value using which distributed systems can coordinate. Since it needs to be able to handle the load, Zookeeper itself runs on many machines.

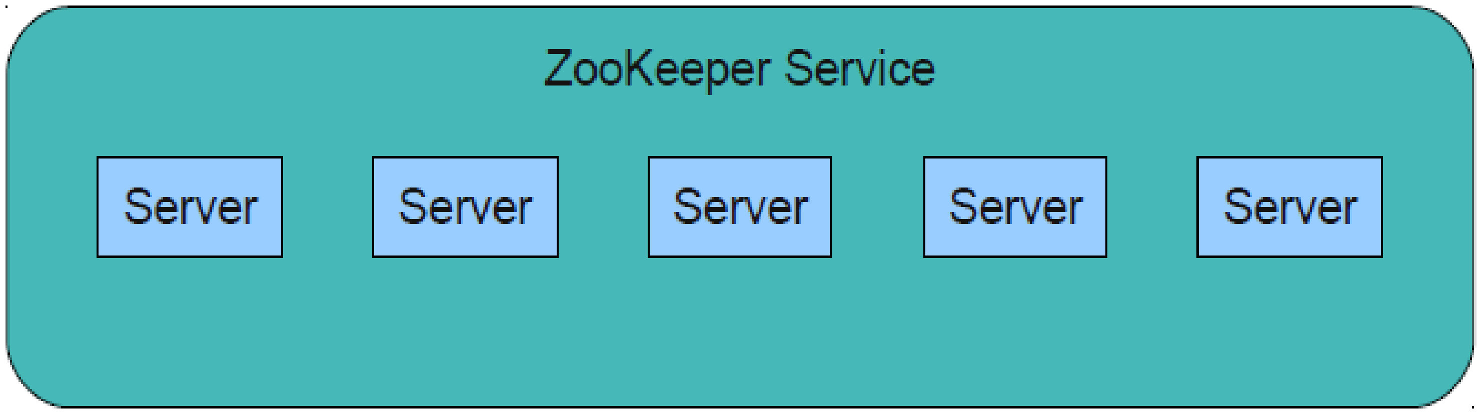
What is ZooKeeper ?

- Exposes a simple set of primitives
- Very easy to program
- Uses a data model like directory tree
- Used for
  - Synchronisation
  - Locking
  - Maintaining Configuration
- Coordination service that does not suffer from
  - Race Conditions
  - Dead Locks

Design Goals: 1. Simple

- A shared hierarchal namespace looks like standard file system
- The namespace has data nodes - znodes (similar to files/dirs)
- Data is kept in-memory
- Achieve high throughput and low latency numbers.
- **High performance**
  - Used in large, distributed systems
- **Highly available**
  - No single point of failure
- **Strictly ordered access**
  - Synchronisation

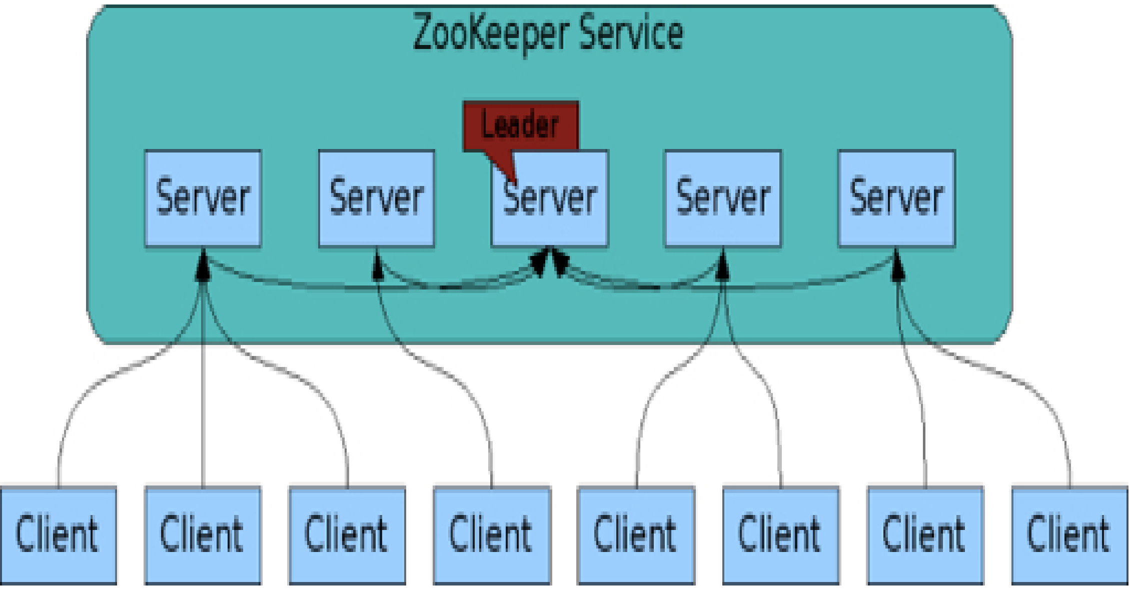
Design Goals: 2. Replicated



- 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 **2*f*+1** machines to tolerate ***f*** failures

Design Goals: 2. Replicated



The client

- Keeps a TCP connection
- Gets watch events
- Sends heart beats.
- If connection breaks,
    - Connect to different server.

The servers

- Know each other
- Keep in-memory image of State
- Transaction Logs & Snapshots - persistent

Design Goals: 3. Ordered

- ZooKeeper stamps each update with a number
- **The number:**
  - Reflects the order of transactions.
  - used implement higher-level abstractions, such as synchronization primitives.

Design Goals: 4. Fast

Performs best where reads are more common than writes, at ratios of around 10:1.

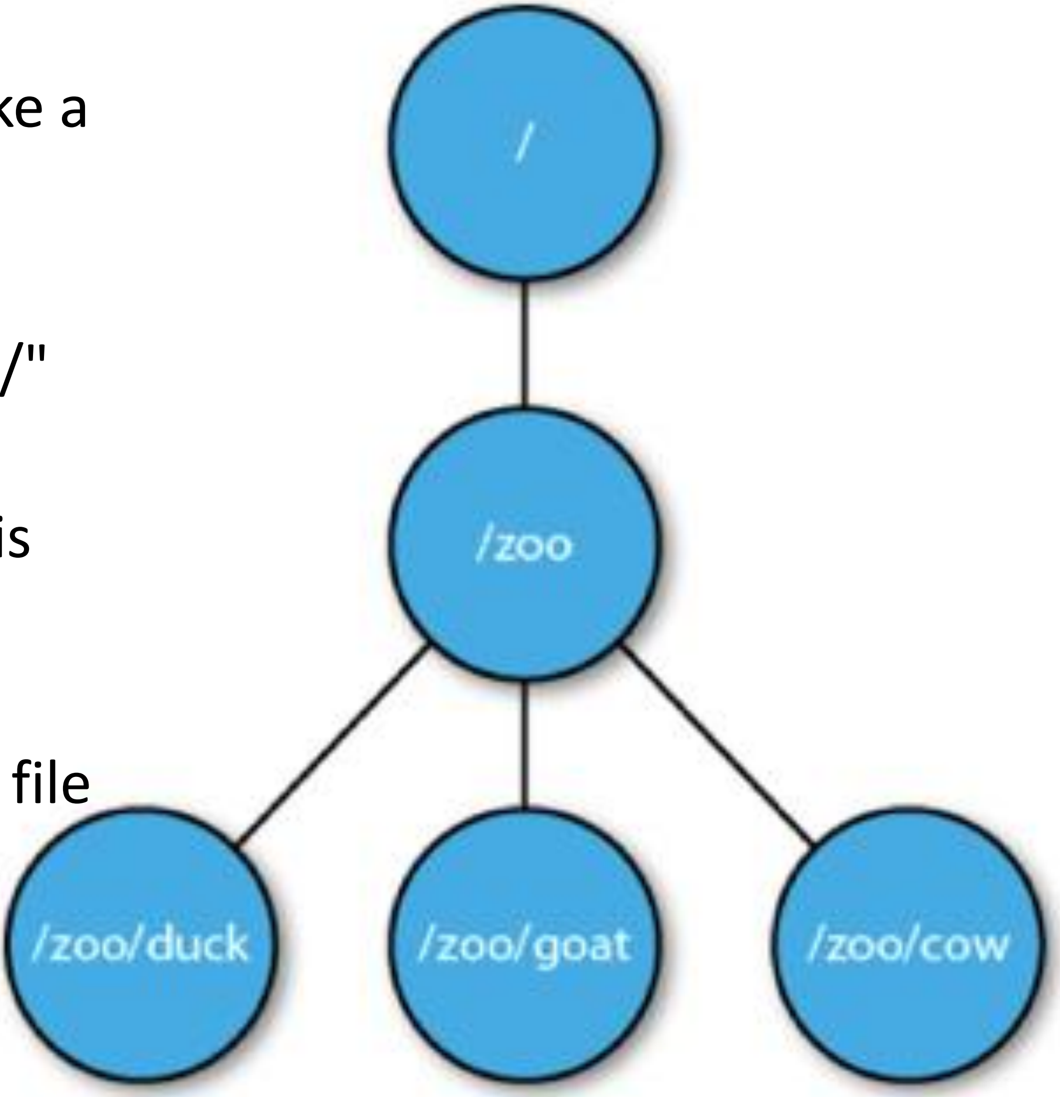
At Yahoo!, where it was created, the throughput for a ZooKeeper cluster has been benchmarked at over 10,000 operations per second for write-dominant workloads generated by hundreds of clients

Data Model

- The way you store data in any store is called **data model.**
- **Think of it as highly available fileSystem:** In case of zookeeper, think of data model as if it is a highly available file system with little differences.
- **Znode:** We store data in an entity called znode.
- **JSON data:** The data that we store should be in JSON format which Java script object notation.
- **No Append Operation:** The znode can only be updated. It does not support append operations.
- **Data access (read/write) is atomic:** The read or write is atomic operation meaning either it will be full or would throw an error if failed. There is no intermediate state like half written.
- **Znode:** Can have children

Data Model Contd...

- So, znodes inside znodes make a tree like heirarchy.
- The top level znode is "/".
- The znode "/zoo" is child of "/" which top level znode.
- duck is child znode of zoo. It is denoted as /zoo/duck
- Though "." or ".." are invalid characters as opposed to the file system.



Data Model – Znode - Types

- **Persistent**
  - Such kind of znodes remain in zookeeper untill deleted. This is the default type of znode. To create such node you can use the command: create /name_of_myznode "mydata"
- **Ephemeral**
  - Ephermal node gets deleted if the session in which the node was created has disconnected. Though it is tied to client's session but it is visible to the other users.
  - An ephermal node can not have children not even ephermal children.

Data Model – Znode - Types

- **Sequential**
  - Creates a node with a sequence number in the name
  - The number is automatically appended.

```
create -s /zoo v
Created /zoo0000000008
```

```
create -s /zoo/ v
Created /zoo/0000000003
```

```
create -s /xyz v
Created /xyz0000000009
```

```
create -s /zoo/ v
Created /zoo/0000000004
```

- The counter keeps increasing monotonically
- Each node keeps a counter

Architecture

- **Zookeeper can run in two modes: (i) Standalone and (ii) Replicated.**

(i) Standalone:

- In standalone mode, it is just running on one machine and for practical purposes we do not use stanalone mode.
- This is only for testing purposes.
- It doesn't have high availability.

(ii) Replicated:

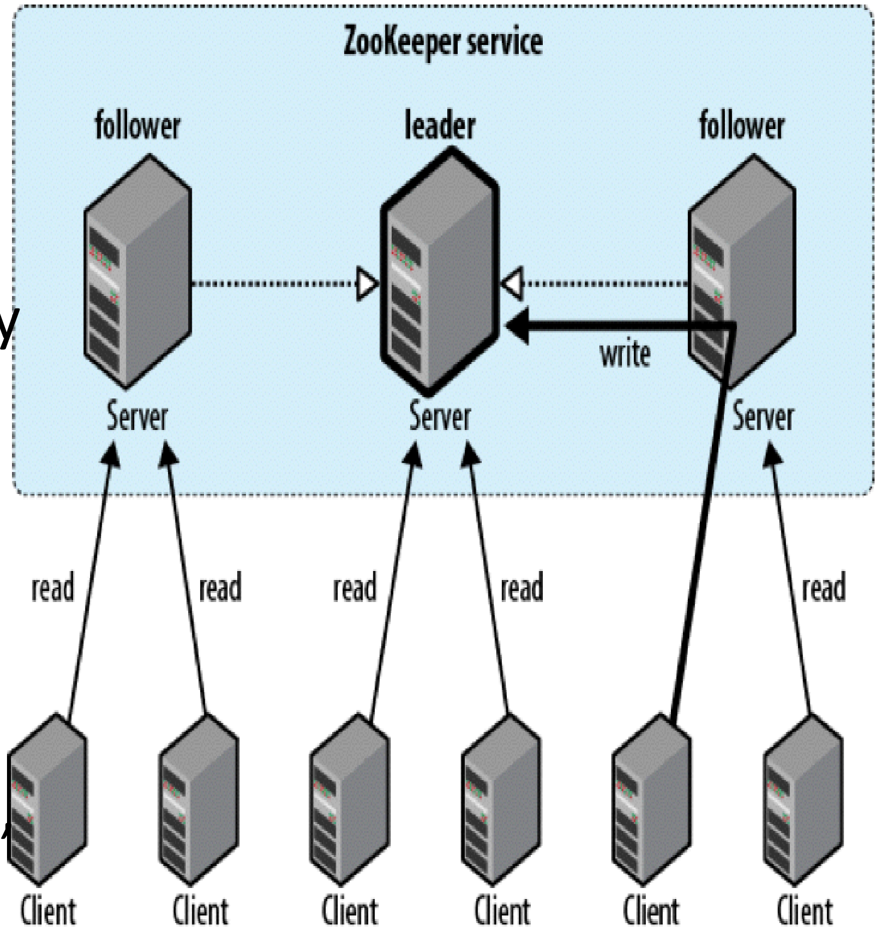
- Run on a cluster of machines called an ensemble.
- High availability
- Tolerates as long as majority.

Architecture: Phase 1

Phase 1: Leader election (Paxos Algorithm)

- The machines elect a distinguished member - leader.
- The others are termed followers.
- This phase is finished when majority sync their state with leader.
- If leader fails, the remaining machines hold election. takes 200ms.
- If the majority of the machines aren't available at any point of time, the leader automatically steps down.

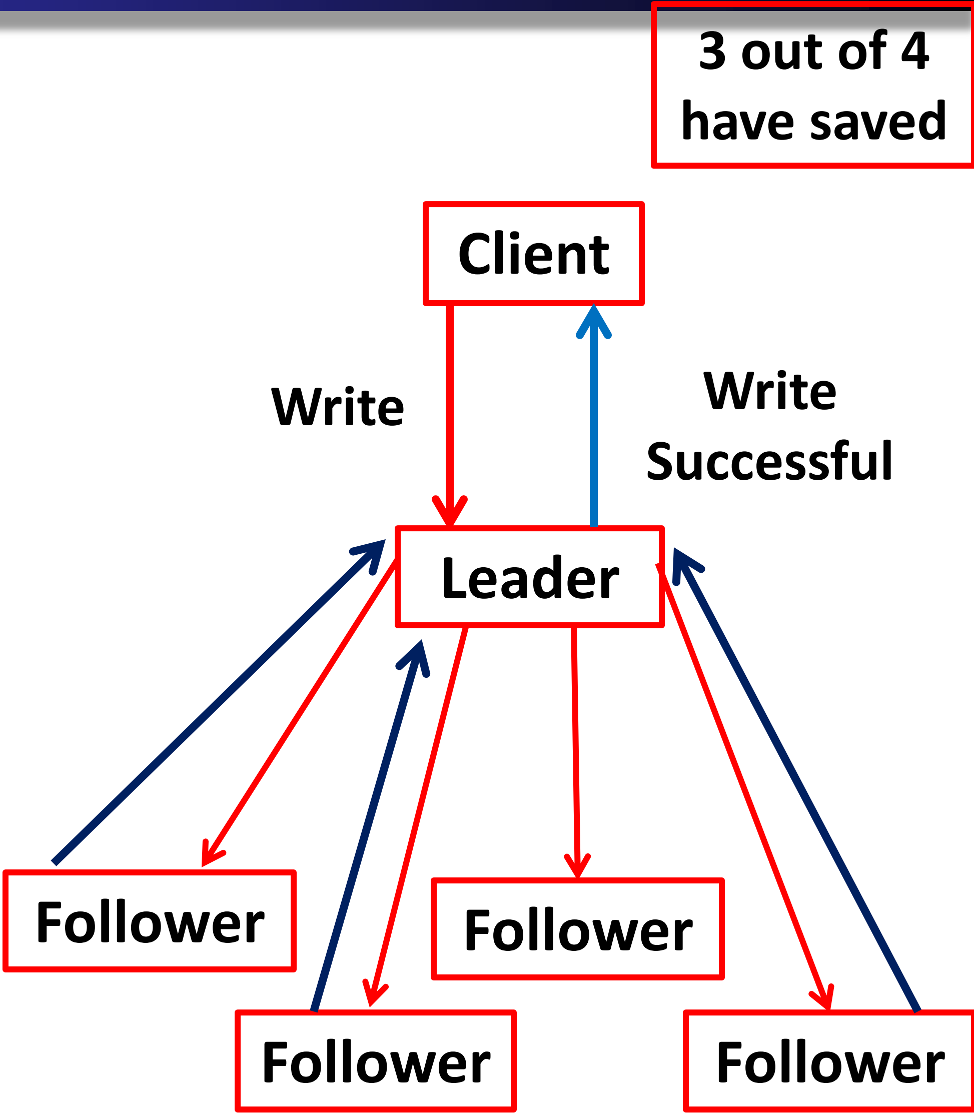
Ensemble



Architecture: Phase 2

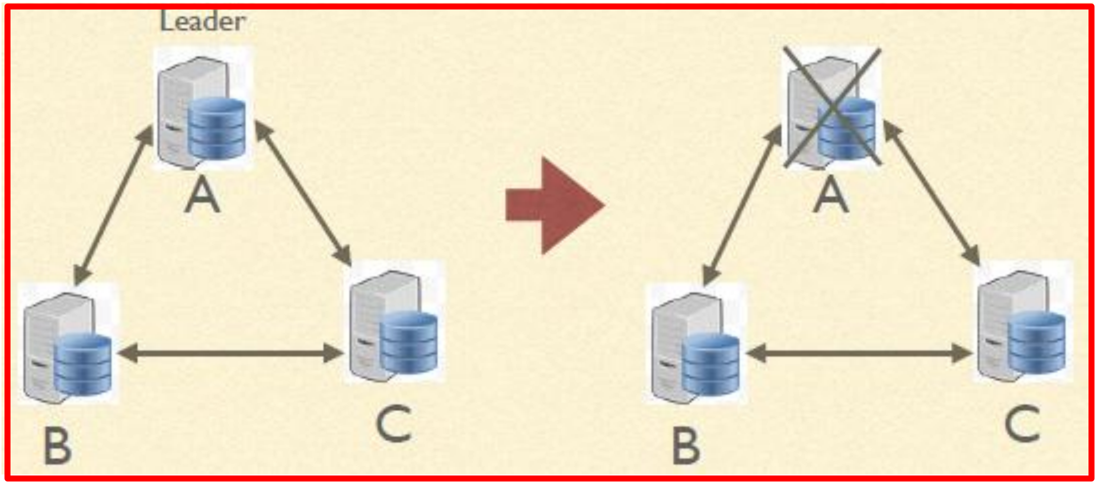
Phase 2: Atomic broadcast

- All write requests are forwarded to the leader,
- Leader broadcasts the update to the followers
- When a majority have persisted the change:
    - The leader commits the up-date
    - The client gets success response.
- The protocol for achieving consensus is atomic like two-phase commit.
- Machines write to disk before in-memory



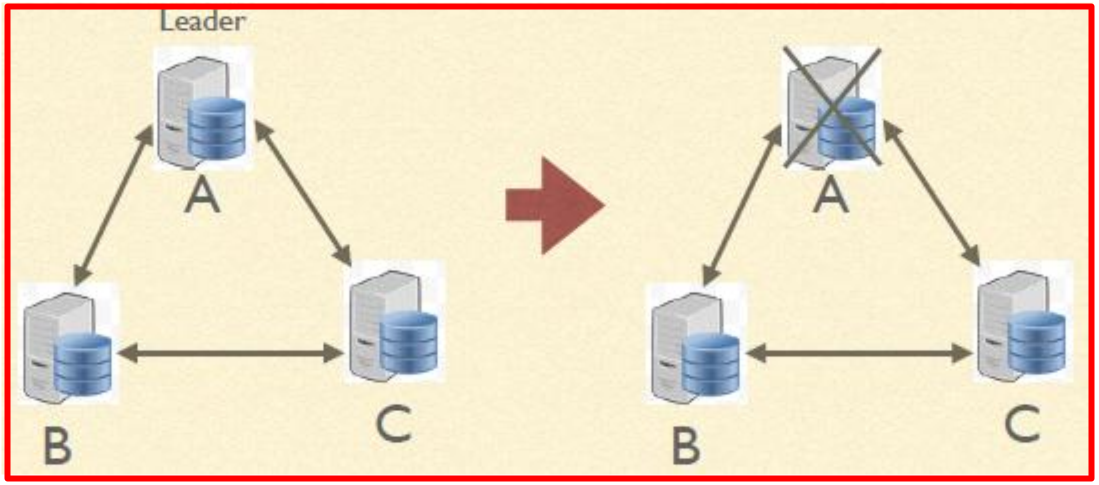
Election Demo

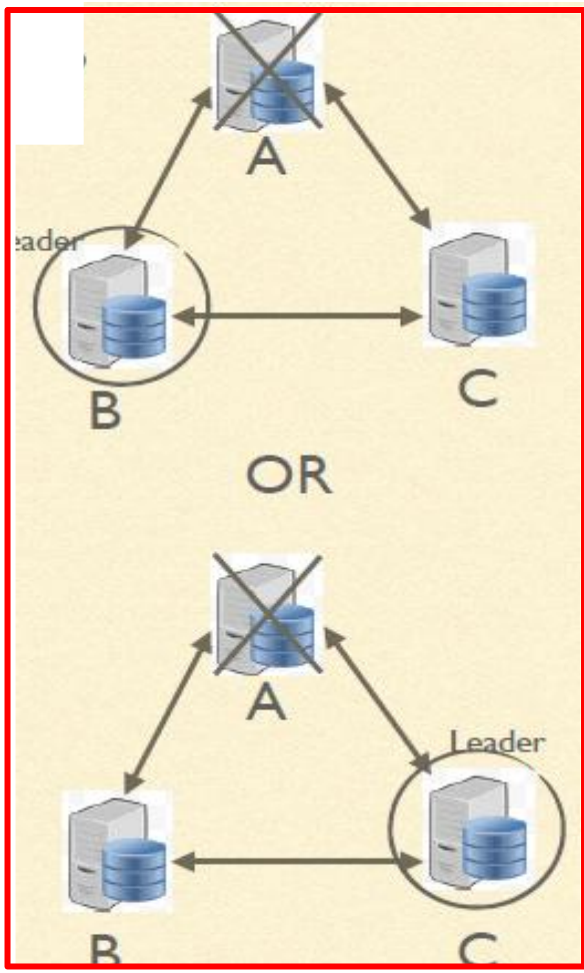
- If you have three nodes A, B, C with A as Leader. And A dies. Will someone become leader?



Election Demo

- If you have three nodes A, B, C with A as Leader. And A dies. Will someone become leader?

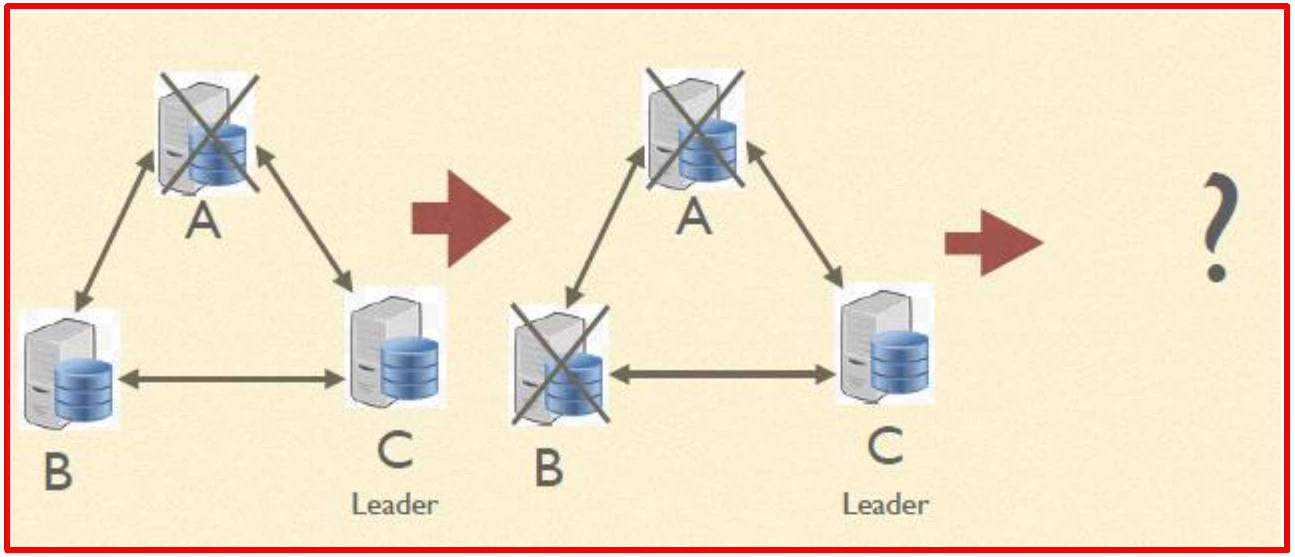




Yes. Either B or C.

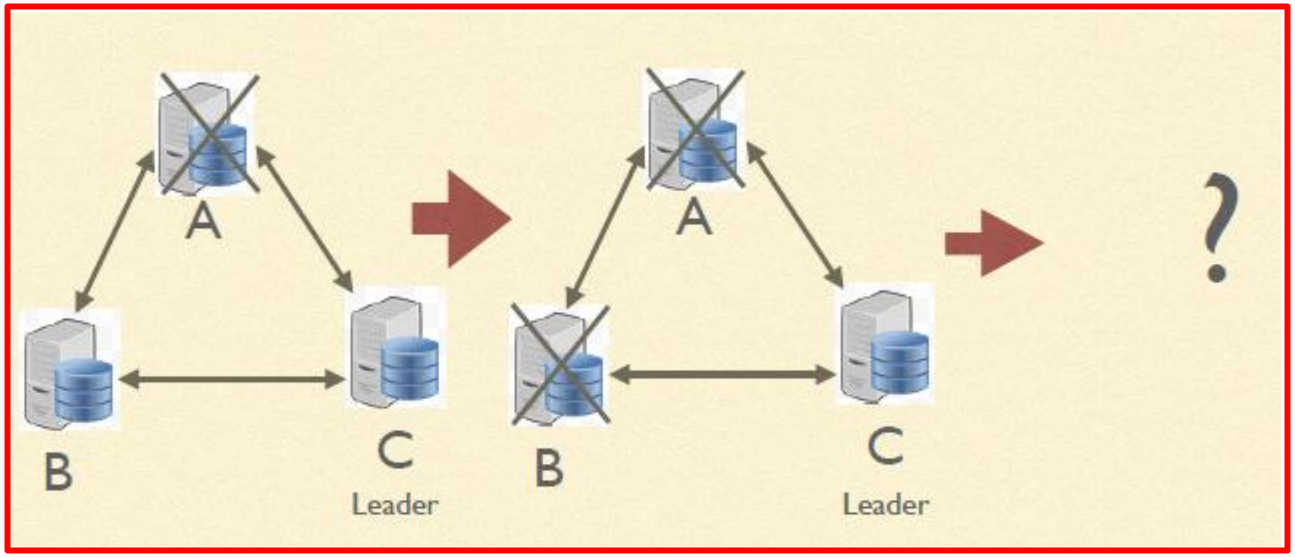
Election Demo

- If you have three nodes A, B, C And A and B die. Will C become Leader?



Election Demo

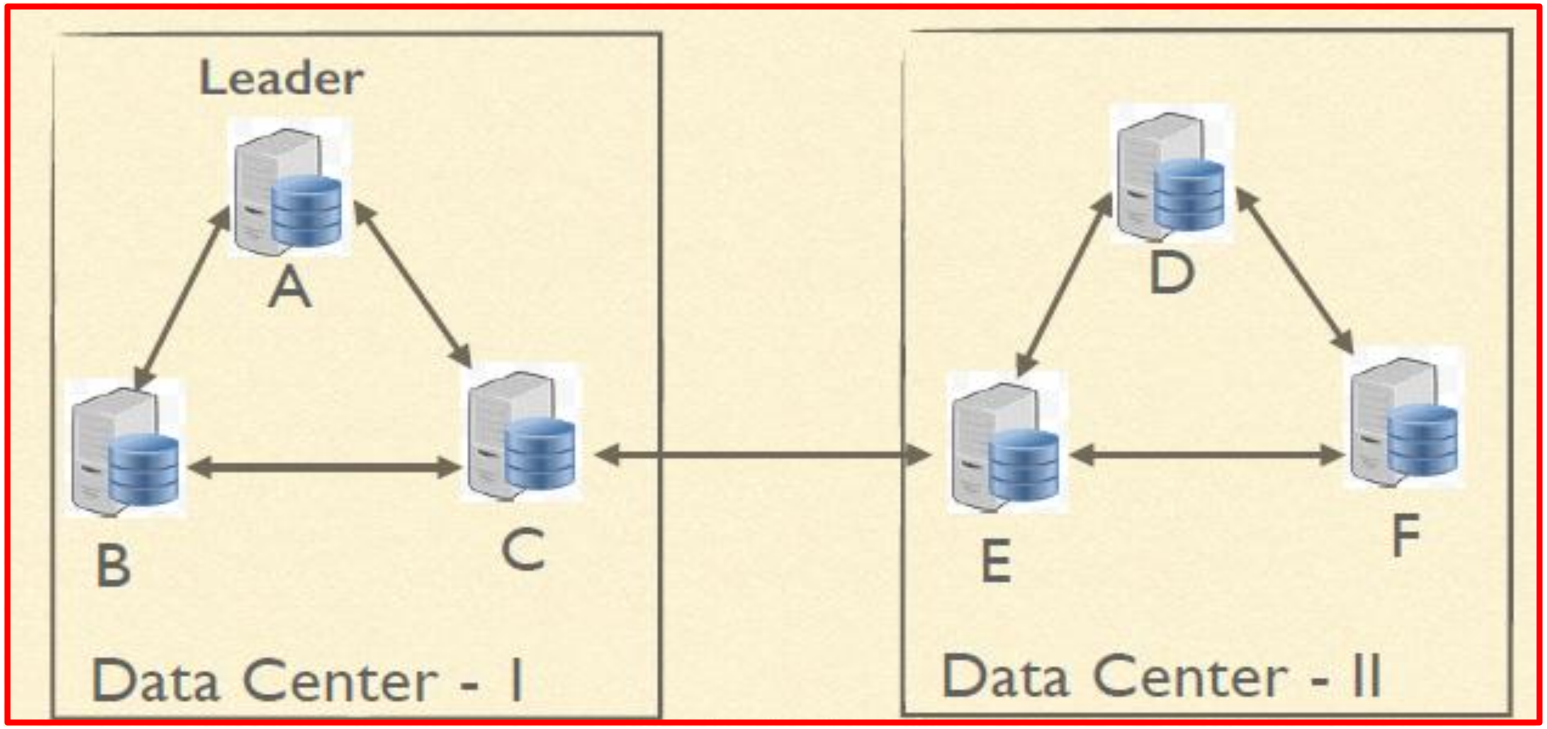
- If you have three nodes A, B, C And A and B die. Will C become Leader?



No one will become Leader.
C will become Follower.
Reason: Majority is not available.

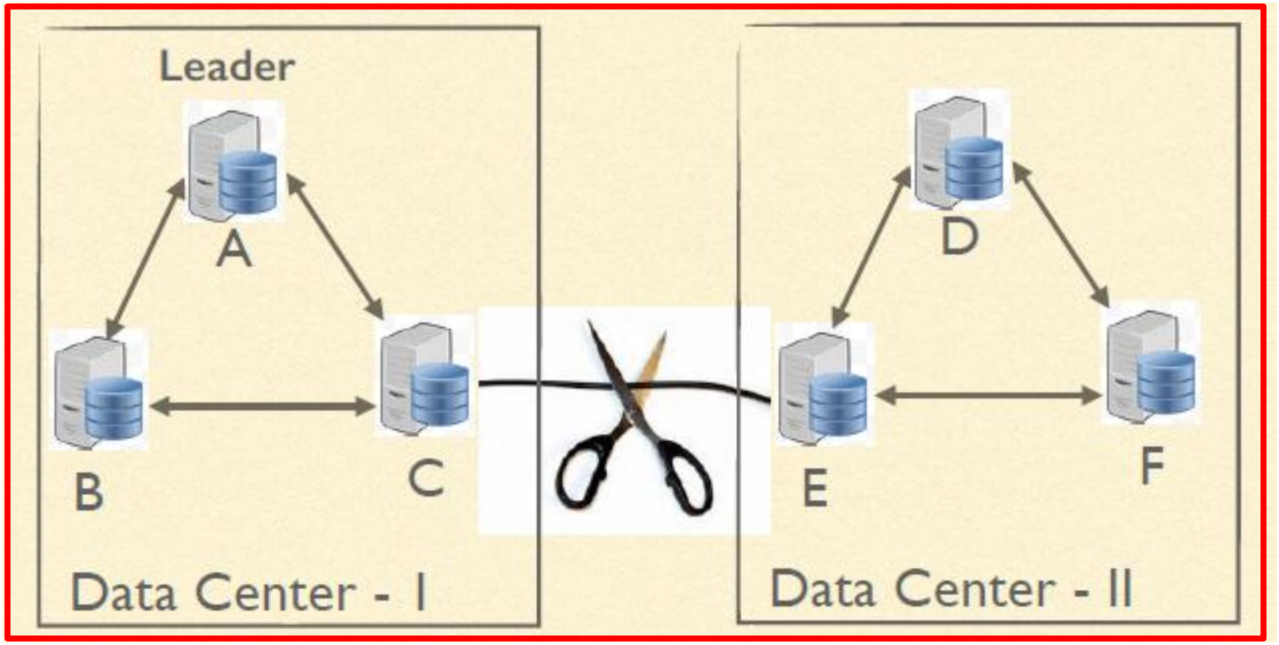
Why do we need majority?

- **Imagine:** We have an ensemble spread over two data centres.



Why do we need majority?

- **Imagine:** The network between data centres got disconnected. If we did not need majority for electing Leader,
- **What will happen?**

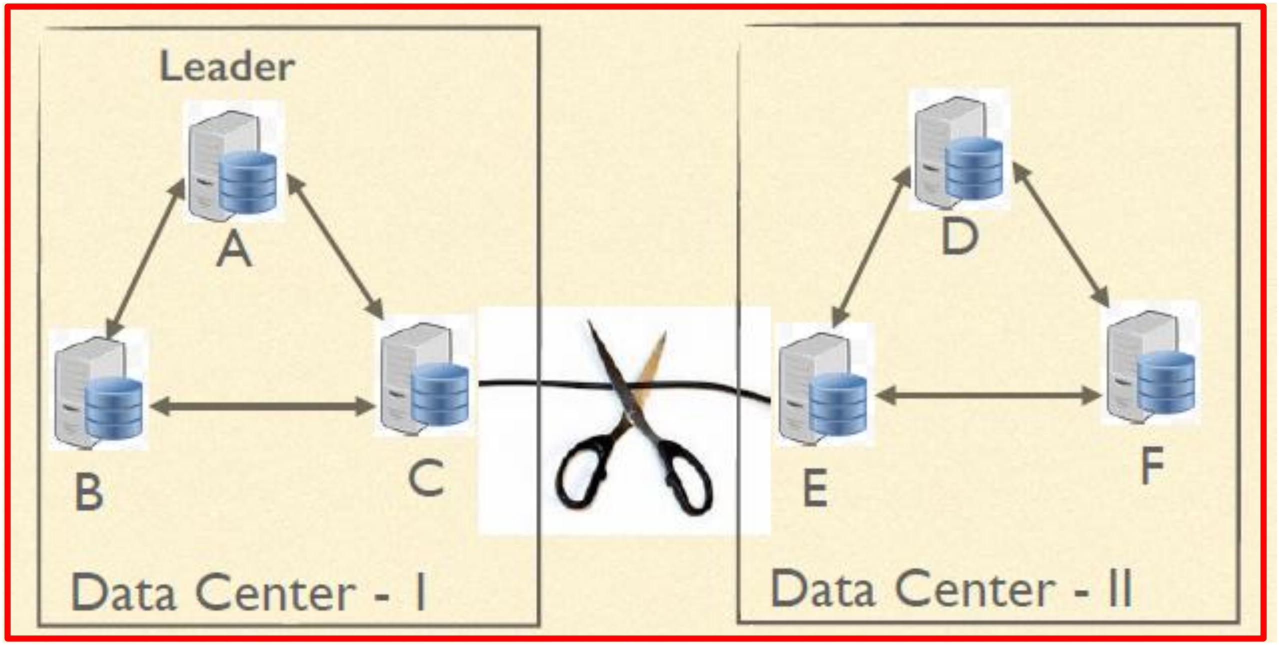


Why do we need majority?

Each data centre will have their own Leader.

No Consistency and utter Chaos.

That is why it requires majority.



Sessions

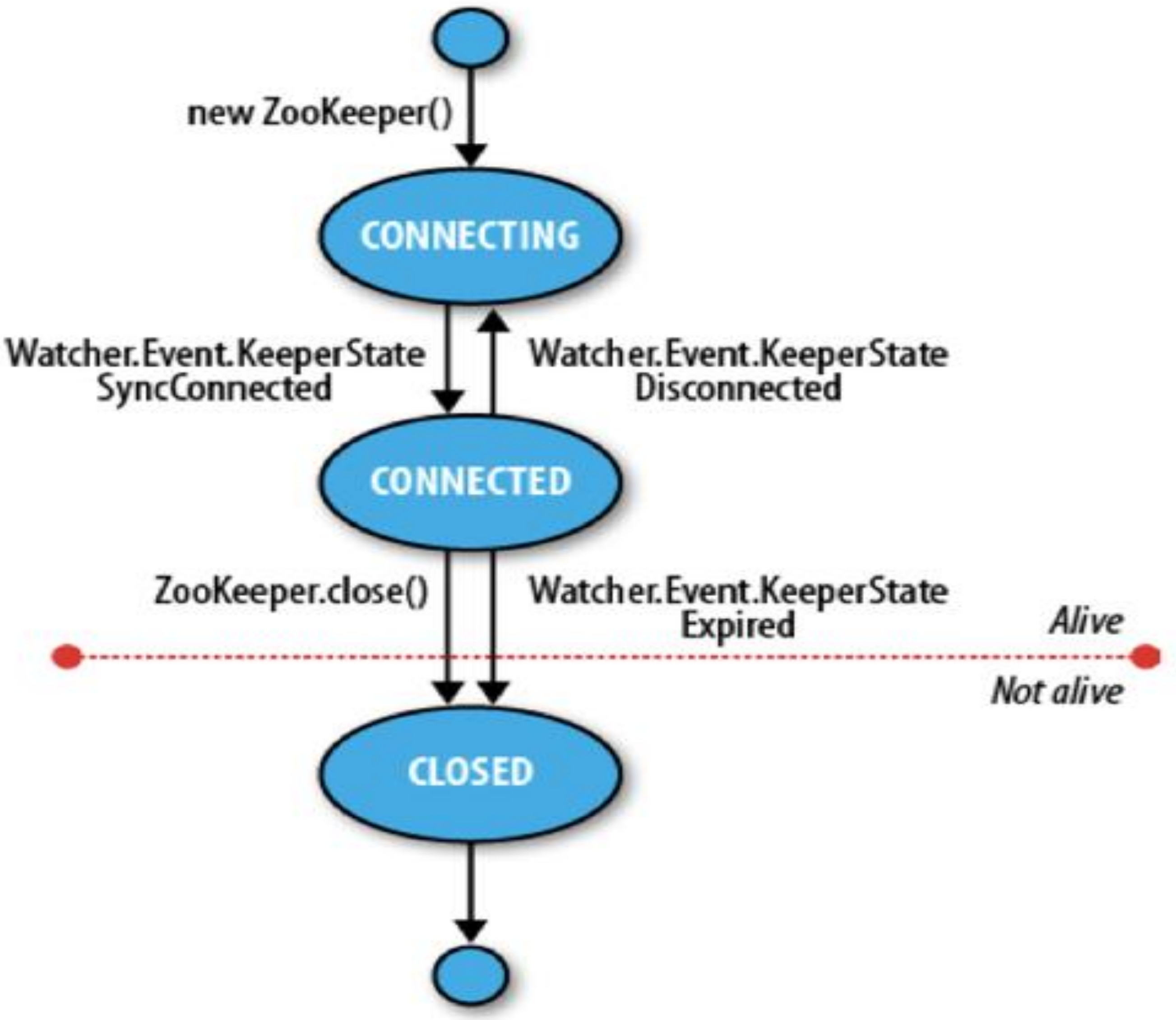
- Lets try to understand **how do the zookeeper decides to delete ephermals nodes** and takes care of session management.

- A client has list of servers in the ensemble
- It tries each until successful.
- Server creates a new session for the client.
- A session has a timeout period - decided by caller

Contd...

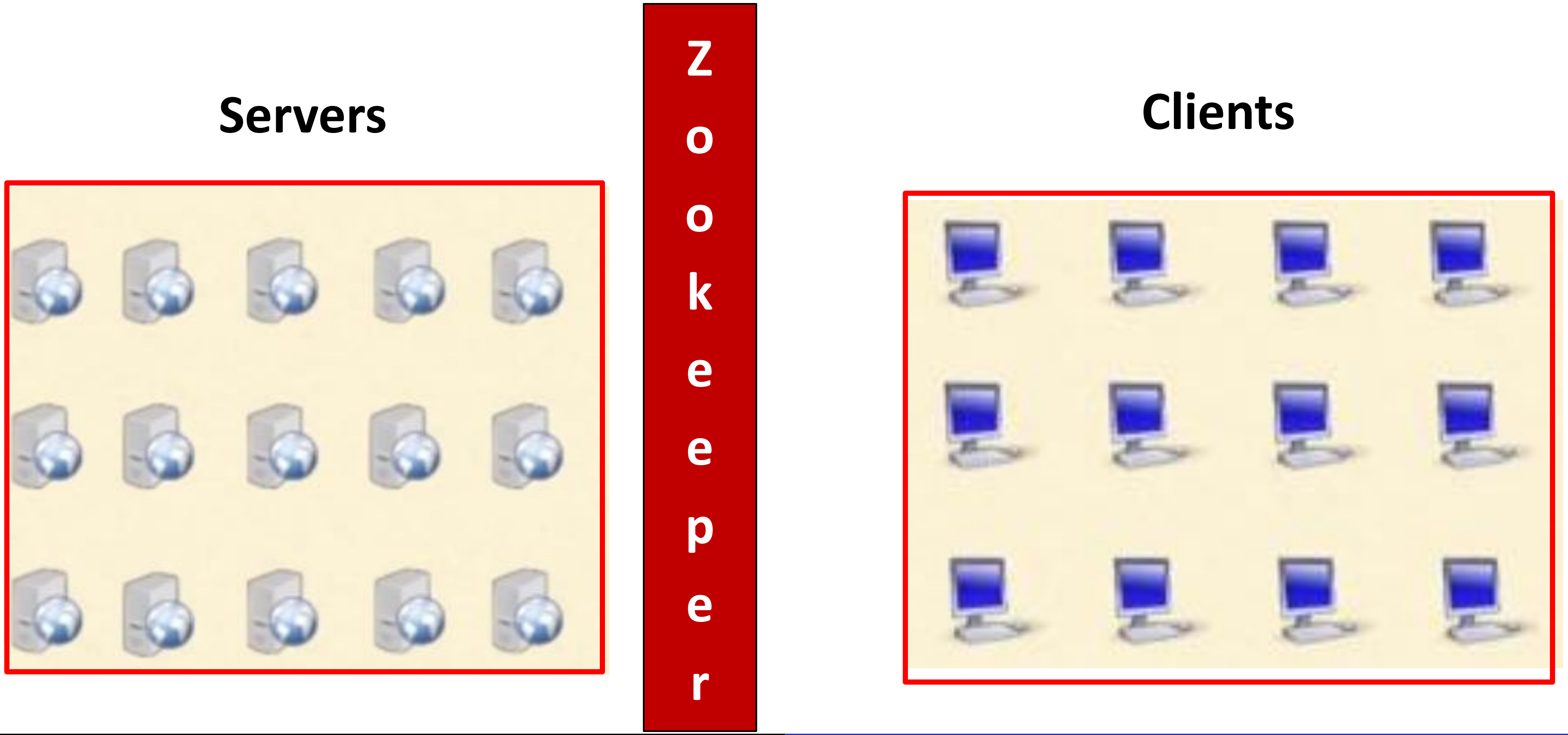
- If the server hasn’t received a request within the timeout period, it may expire the session.
- On session expire, ephermal nodes are lost
- To keep sessions alive client sends pings (heartbeats)
- Client library takes care of heartbeats
- Sessions are still valid on switching to another server
- Failover is handled automatically by the client
- Application can't remain agnostic of server reconnections - because the operations will fail during disconnection.

States


new ZooKeeper()
CONNECTING
Watcher.Event.KeeperState SyncConnected
Watcher.Event.KeeperState Disconnected
CONNECTED
ZooKeeper.close()
Watcher.Event.KeeperState Expired
Alive
Not alive
CLOSED

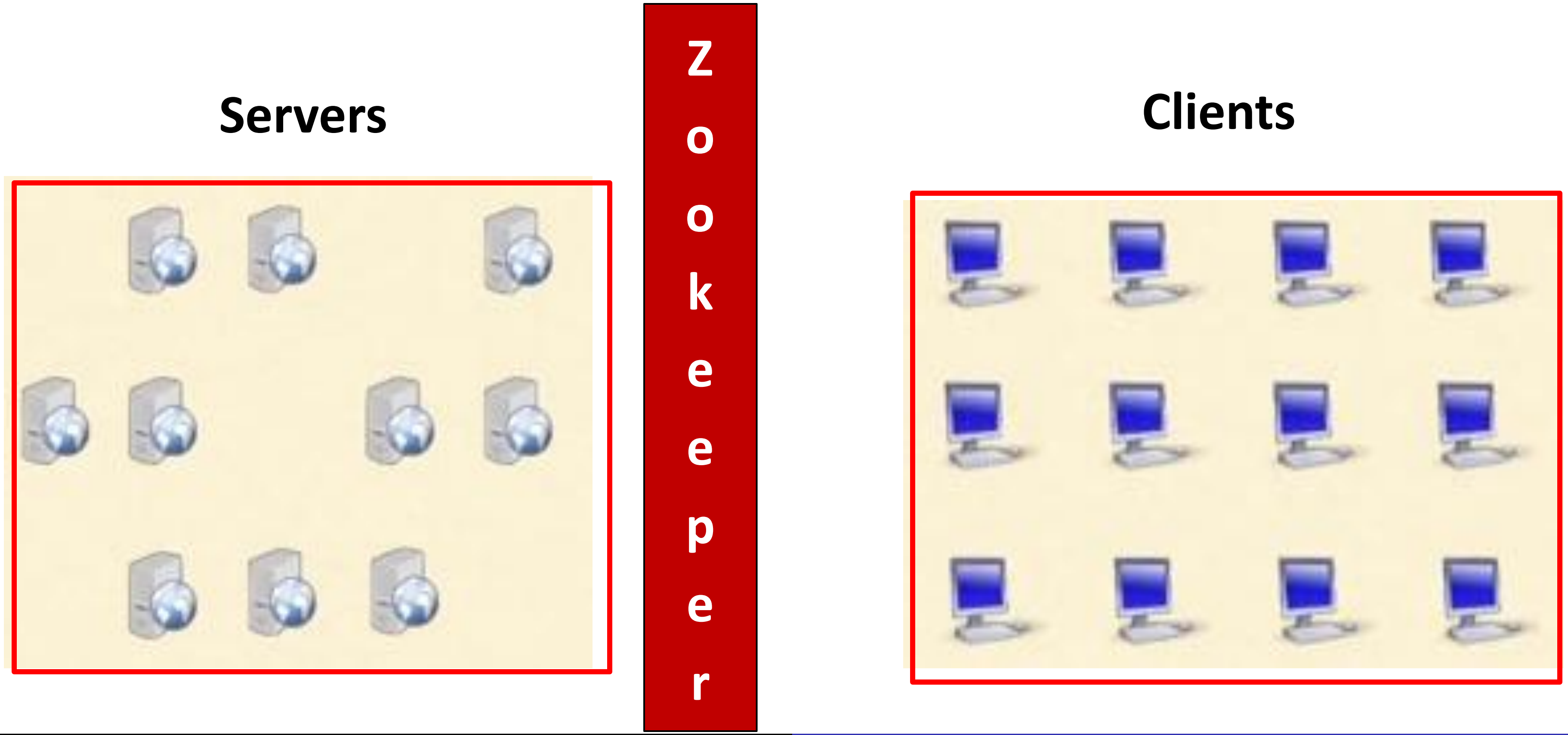
Use Case: Many Servers How do they Coordinate?

- Let us say there are many servers which can respond to your request and there are many clients which might want the service.



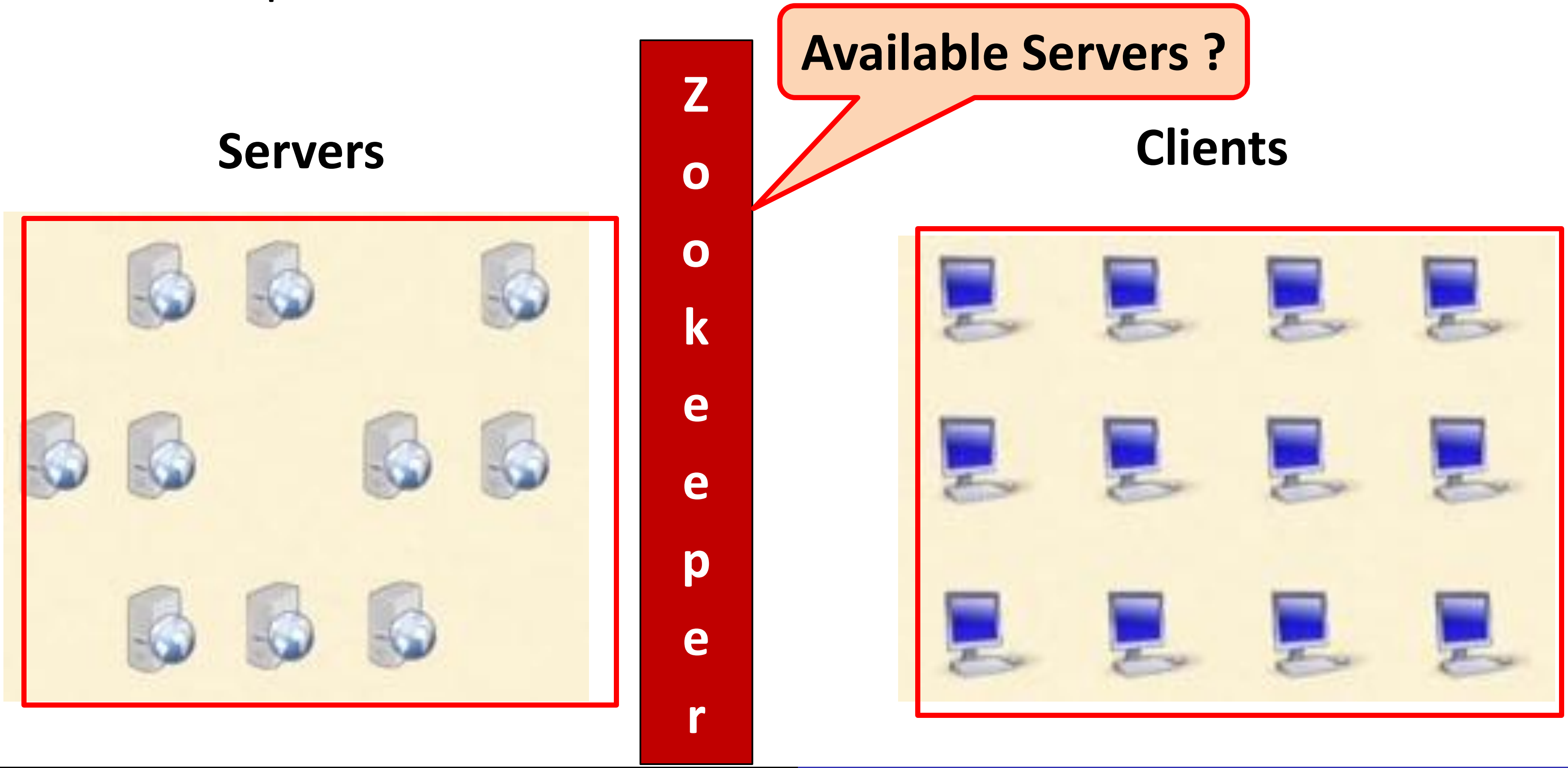
Use Case: Many Servers How do they Coordinate?

- From time to time some of the servers will keep going down. How can all of the clients can keep track of the available servers?



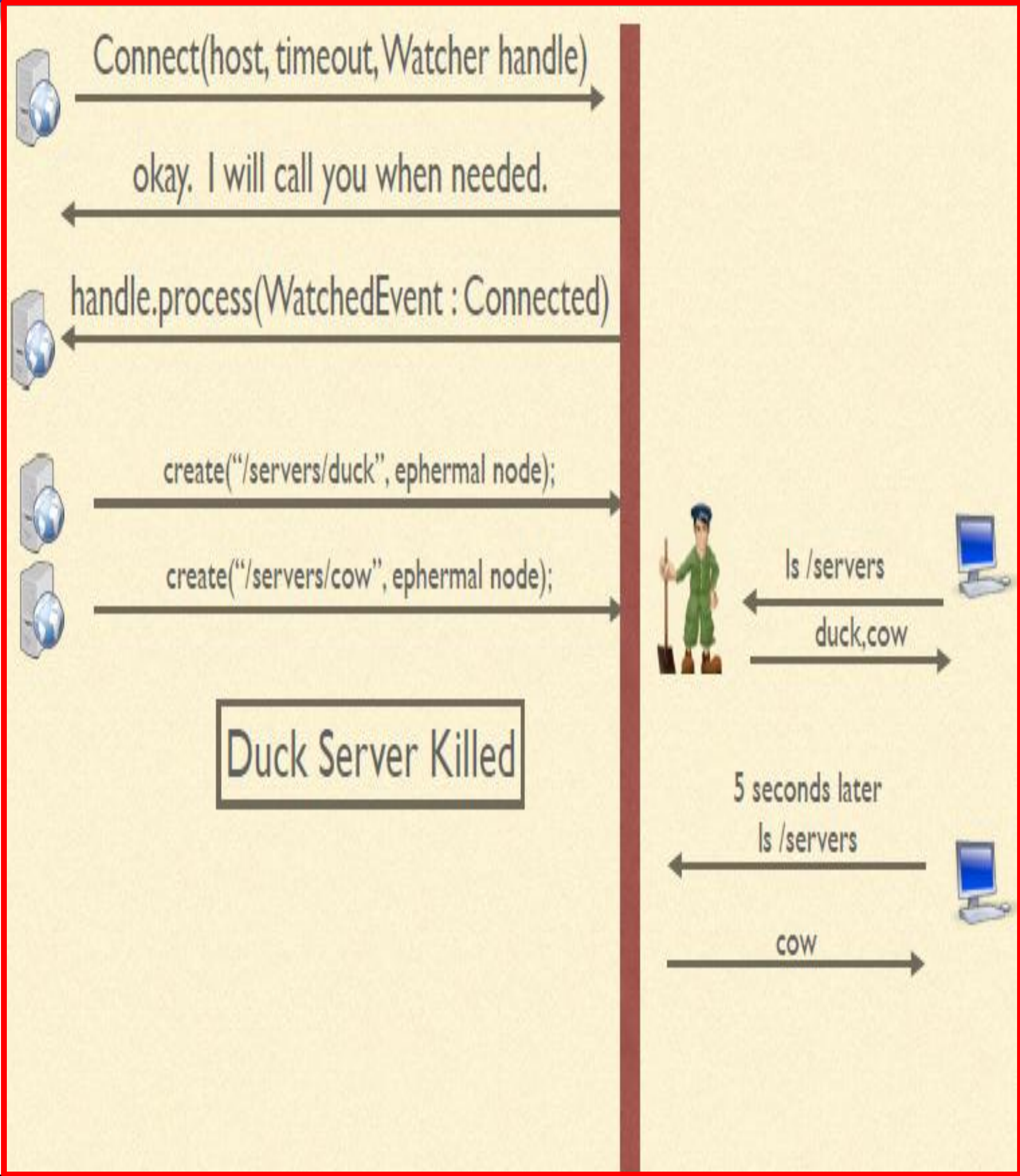
Use Case: Many Servers How do they Coordinate?

- It is very easy using zookeeper as a centeral agency. Each server will create their own ephermal znode under a particular znode say "/servers". The clients would simply query zookeeper for the most recent list of servers.



Use Case: Many Servers How do they Coordinate?

- Lets take a case of two servers and a client. The two server duck and cow created their ephermal nodes under "/servers" znode. The client would simply discover the alive servers cow and duck using command ls /servers.
- Say, a server called "duck" is down, the ephermal node will disappear from /servers znode and hence next time the client comes and queries it would only get "cow". So, the coordinations has been made heavily simplified and made efficient because of ZooKeeper.



Connect(host, timeout, Watcher handle)
okay. I will call you when needed.
handle.process(WatchedEvent : Connected)
create("/servers/duck", ephermal node);
create("/servers/cow", ephermal node);
ls /servers
duck,cow
Duck Server Killed
5 seconds later
ls /servers
cow

Guarantees

- **Sequential consistency**
  - Updates from any particular client are applied in the order
- **Atomicity**
  - Updates either succeed or fail.
- **Single system image**
  - A client will see the same view of the system, The new server will not accept the connection until it has caught up.
- **Durability**
  - Once an update has succeeded, it will persist and will not be undone.
- **Timeliness**
  - Rather than allow a client to see very stale data, a server will shut down,

Operations

OPERATION	DESCRIPTION
create	Creates a znode (parent znode must exist)
delete	Deletes a znode (mustn’t have children)
exists/ls	Tests whether a znode exists & gets metadata
getACL, setACL	Gets/sets the ACL for a znode getChildren/ls Gets a list of the children of a znode
getData/get, setData	Gets/sets the data associated with a znode
sync	Synchronizes a client’s view of a znode with ZooKeeper

Multi Update

- Batches together multiple operations together
- Either all fail or succeed in entirety
- Possible to implement transactions
- Others never observe any inconsistent state

APIs

- Two core: Java & C
- contrib: perl, python, REST
- For each binding, sync and async available

Synch:

Public Stat exists (String path, Watcher watcher) throws KeeperException, InterruptedException

Asynch:

Public void exists (String path, Watcher watcher, StatCallback cb, Object ctx

Watches

- Clients to get notifications when a znode changes in some way
- Watchers are triggered only once
- For multiple notifications, re-register

Watch Triggers

- The read operations exists, getChildren, getData may have watches
- Watches are triggered by write ops: create, delete, setData
- **ACL (Access Control List)** operations do not participate in watches

WATCH OF …ARE TRIGGERED	WHEN ZNODE IS…
exists	created, deleted, or its data updated.
getData	deleted or has its data updated.
getChildren	deleted, or its any of the child is created or deleted

ACLs - Access Control Lists

ACL Determines who can perform certain operations on it.

- **ACL is the combination**
  - authentication scheme,
  - an identity for that scheme,
  - and a set of permissions
- **Authentication Scheme**
  - **digest -** The client is authenticated by a username & password.
  - **sasl -** The client is authenticated using Kerberos.
  - **ip -** The client is authenticated by its IP address.

Use Cases

Building a reliable configuration service

- A Distributed lock service

    Only single process may hold the lock

When Not to Use?

1. To store big data because:
   - The number of copies == number of nodes
   - All data is loaded in RAM too
   - Network load of transferring all data to all Nodes

2. Extremely strong consistency

ZooKeeper Applications: The Fetching Service

- **The Fetching Service:** Crawling is an important part of a search engine, and Yahoo! crawls billions of Web documents. The Fetching Service (FS) is part of the Yahoo! crawler and it is currently in production. Essentially, it has master processes that command page-fetching processes.

- The master provides the fetchers with configuration, and the fetchers write back informing of their status and health. The main advantages of using ZooKeeper for FS are recovering from failures of masters, guaranteeing availability despite failures, and decoupling the clients from the servers, allowing them to direct their request to healthy servers by just reading their status from ZooKeeper.

- Thus, FS uses ZooKeeper mainly to manage configuration metadata, although it also uses Zoo- Keeper to elect masters (leader election).

ZooKeeper Applications: Katta

- **Katta:** It is a distributed indexer that uses Zoo- Keeper for coordination, and it is an example of a non- Yahoo! application. Katta divides the work of indexing using shards.
- A master server assigns shards to slaves and tracks progress. Slaves can fail, so the master must redistribute load as slaves come and go.
- The master can also fail, so other servers must be ready to take over in case of failure. Katta uses ZooKeeper to track the status of slave servers and the master (**group membership**), and to handle master failover (**leader election**).
- Katta also uses ZooKeeper to track and propagate the assignments of shards to slaves (**configuration management**).

ZooKeeper Applications: Yahoo! Message Broker

- **Yahoo! Message Broker: (YMB)** is a distributed publish-subscribe system. The system manages thousands of topics that clients can publish messages to and receive messages from. The topics are distributed among a set of servers to provide scalability.
- Each topic is replicated using a primary-backup scheme that ensures messages are replicated to two machines to ensure reliable message delivery. The servers that makeup YMB use a shared-nothing distributed architecture which makes coordination essential for correct operation.
- YMB uses ZooKeeper to manage the distribution of topics (configuration metadata), deal with failures of machines in the system (failure detection and group membership), and control system operation.

ZooKeeper Applications: Yahoo! Message Broker

- Figure, shows part of the znode data layout for YMB.
- Each broker domain has a znode called nodes that has an ephemeral znode for each of the active servers that compose the YMB service.
- Each YMB server creates an ephemeral znode under nodes with load and status information providing both group membership and status information through ZooKeeper. of YMB.

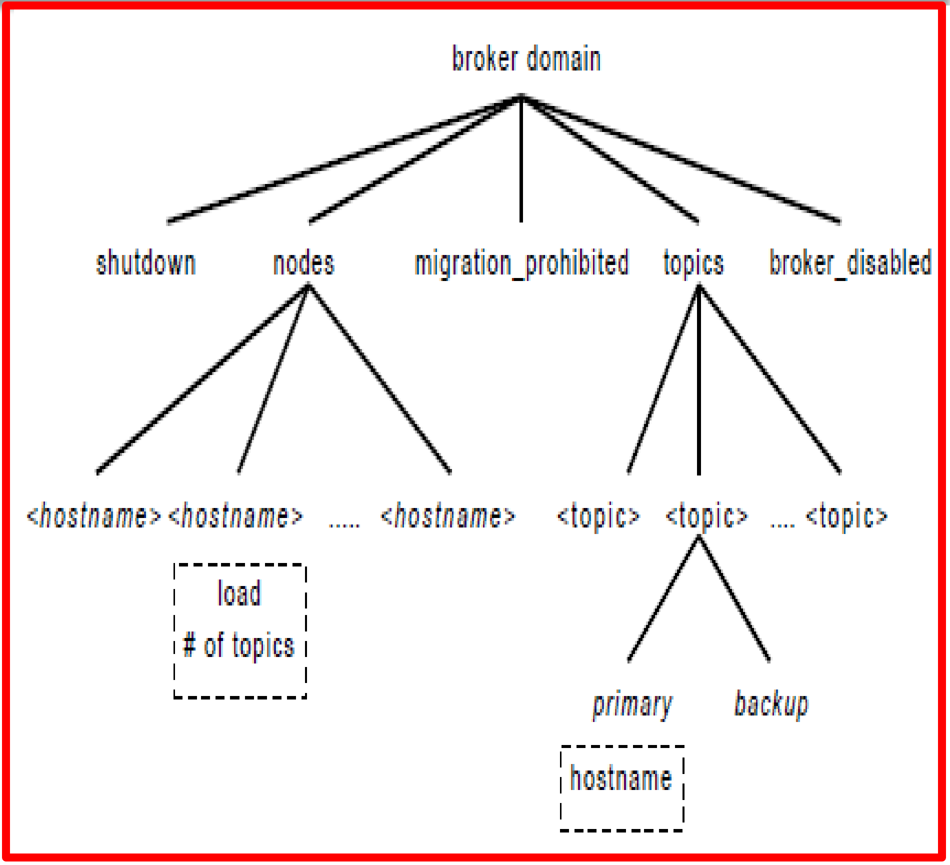


Figure: The layout of Yahoo! Message Broker (YMB) structures in ZooKeeper

ZooKeeper Applications: Yahoo! Message Broker

- The topics directory has a child znode for each topic managed by YMB.
- These topic znodes have child znodes that indicate the primary and backup server for each topic along with the subscribers of that topic.
- The primary and backup server znodes not only allow servers to discover the servers in charge of a topic, but they also manage leader election and server crashes.

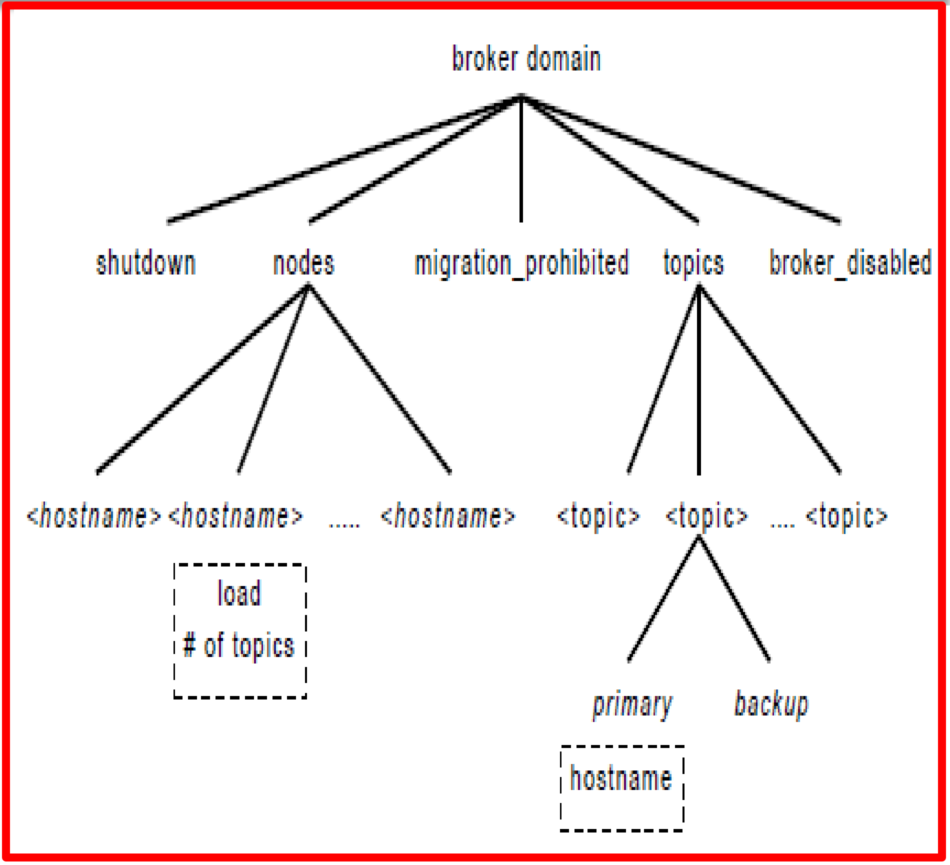
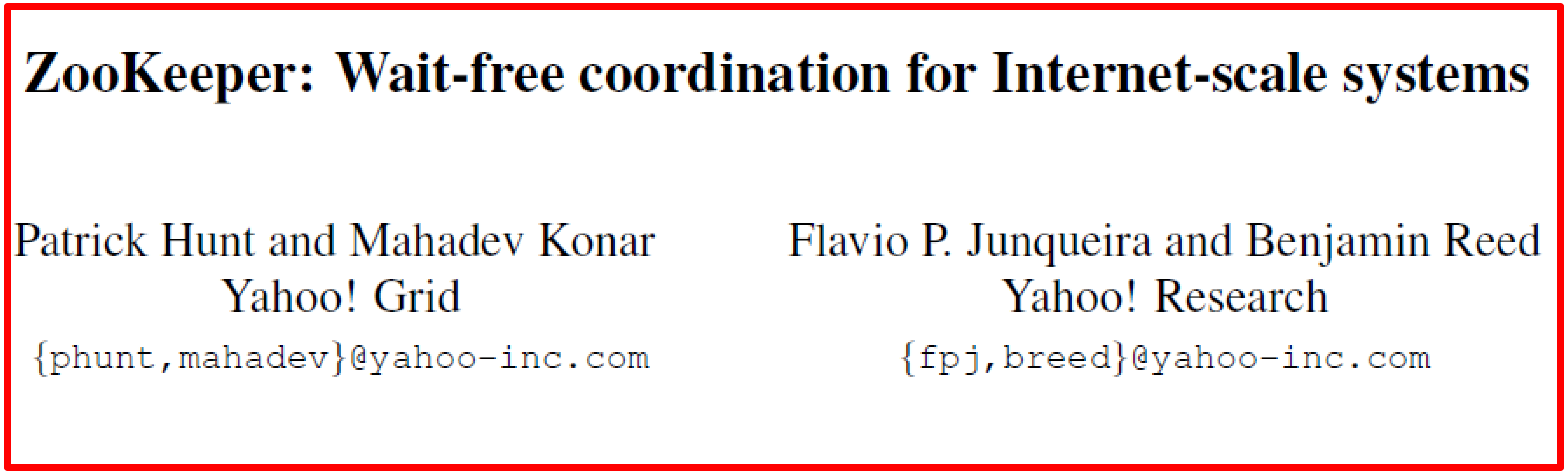


Figure: The layout of Yahoo! Message Broker (YMB) structures in ZooKeeper

More Details



ZooKeeper: Wait-free coordination for Internet-scale systems

Patrick Hunt and Mahadev Konar
Yahoo! Grid
{phunt,mahadev}@yahoo-inc.com

Flavio P. Junqueira and Benjamin Reed
Yahoo! Research
{fpj,breed}@yahoo-inc.com

See: https://zookeeper.apache.org/

Conclusion

- **ZooKeeper** takes a wait-free approach to the problem of coordinating processes in distributed systems, by exposing wait-free objects to clients.

- **ZooKeeper** achieves throughput values of hundreds of thousands of operations per second for read-dominant workloads by using fast reads with watches, both of which served by local replicas.

- In this lecture, the basic fundamentals, design goals, architecture and applications of ZooKeeper.