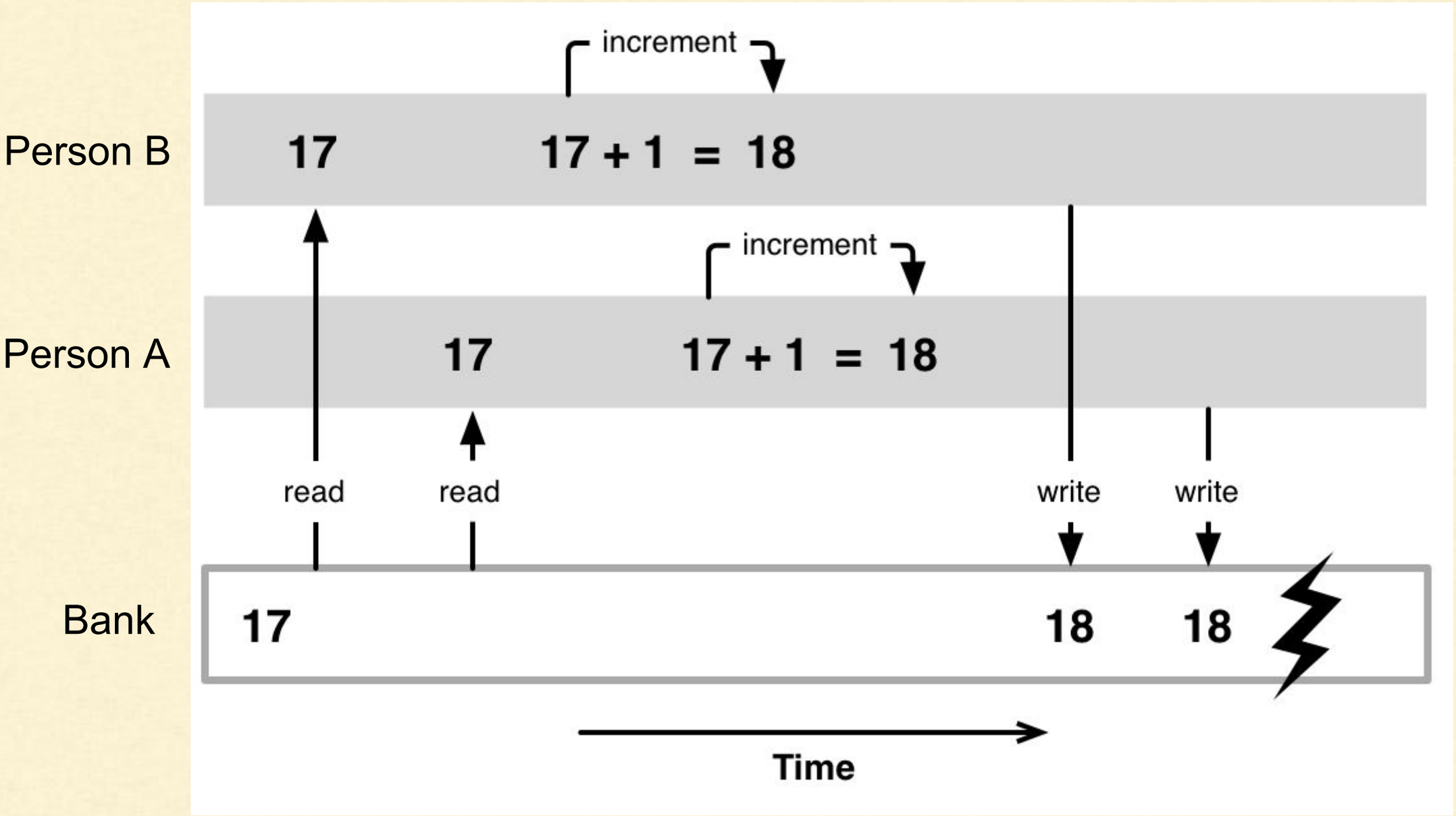
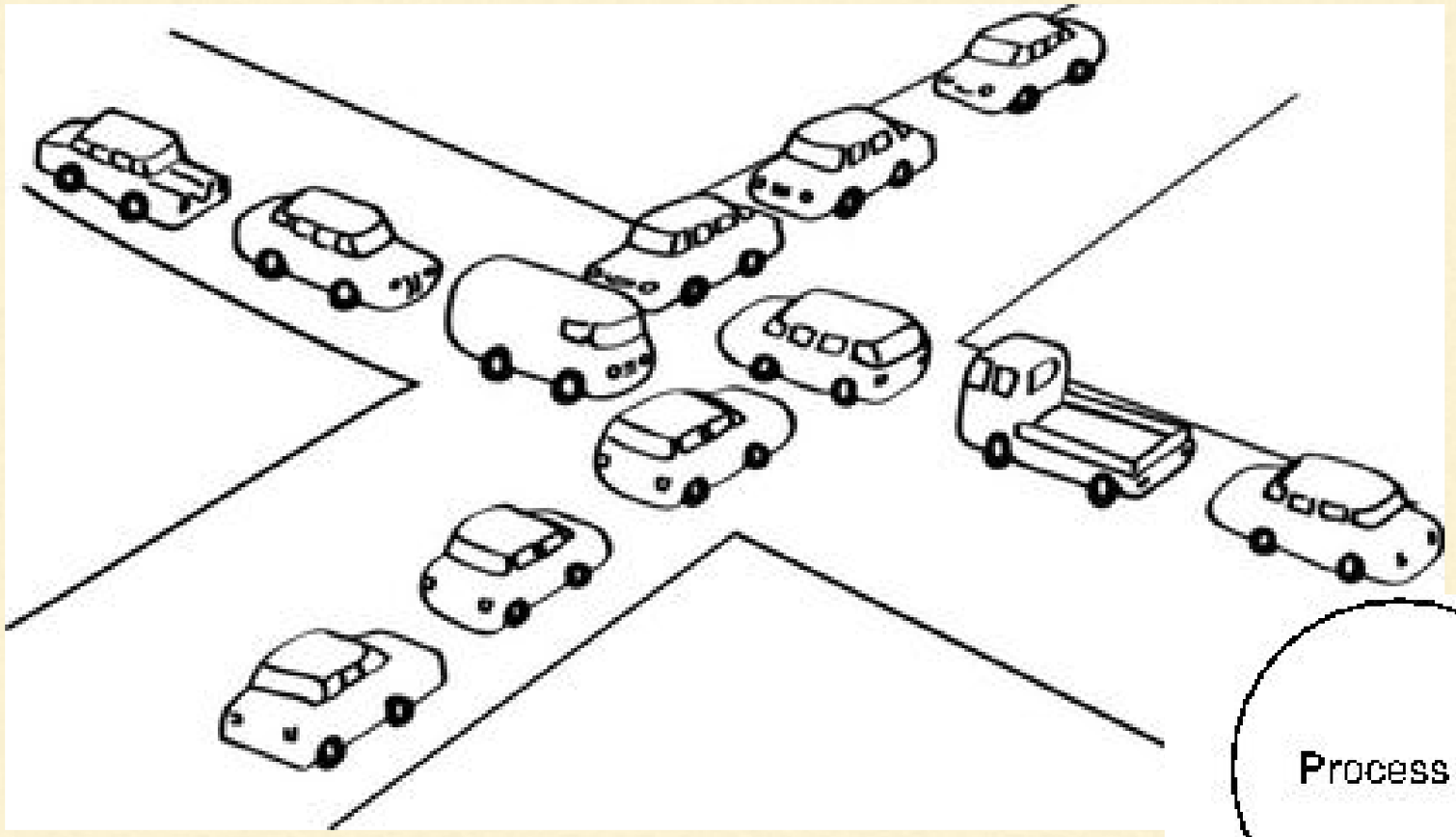
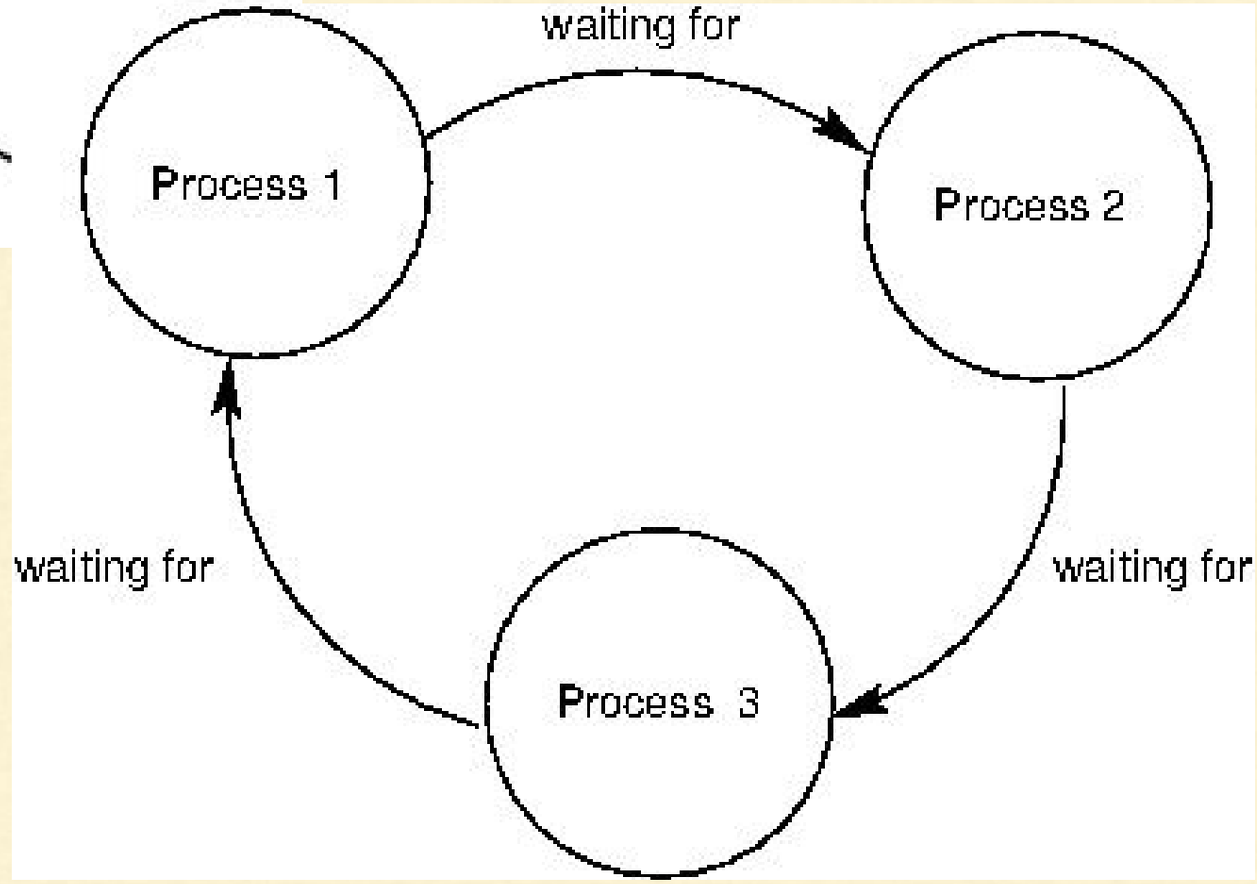
Welcome to ZooKeeper

What is Race Condition?



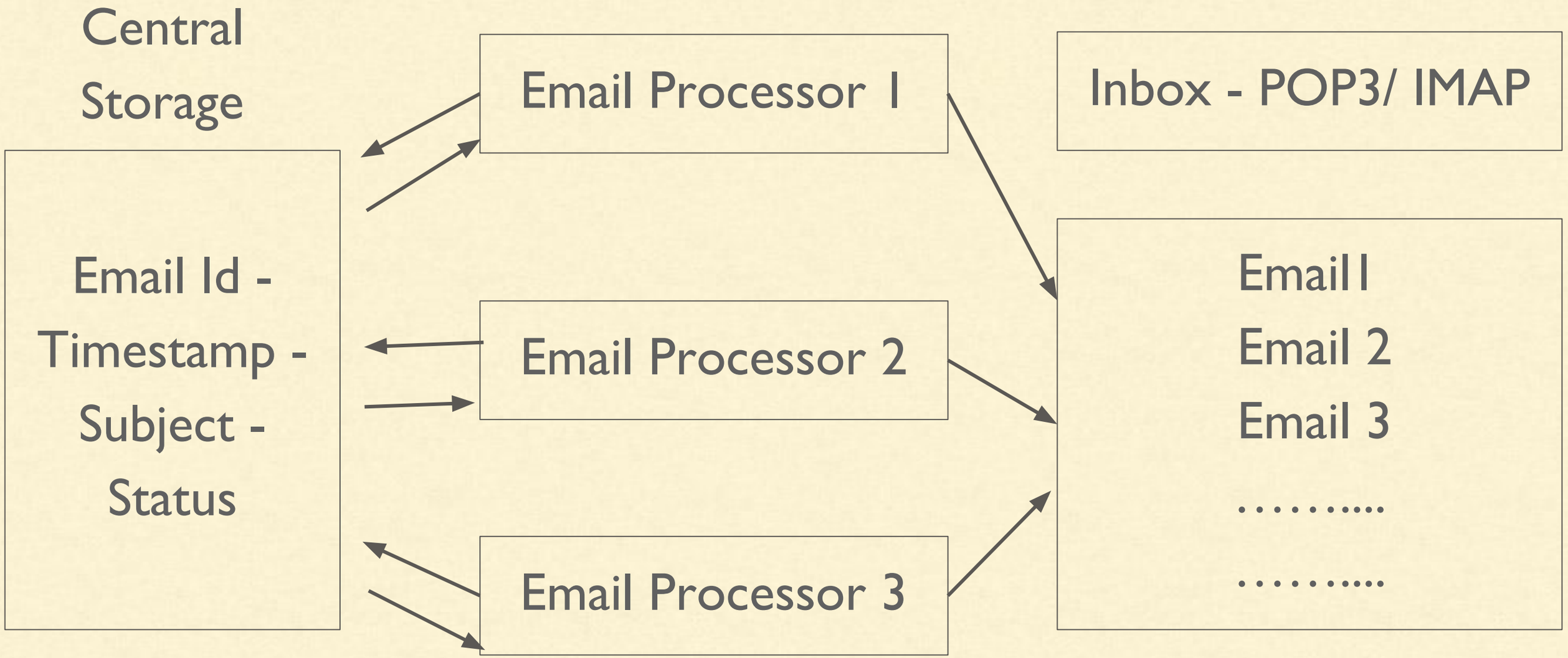
What is a Deadlock?





Coordination?

How would email processors avoid reading same emails?



ZooKeeper Introduction

A Distributed Coordination Service for Distributed Applications

- Exposes a simple set of primitives
- Very easy to program to
- Uses a data model like directory tree

ZooKeeper Introduction - Contd.

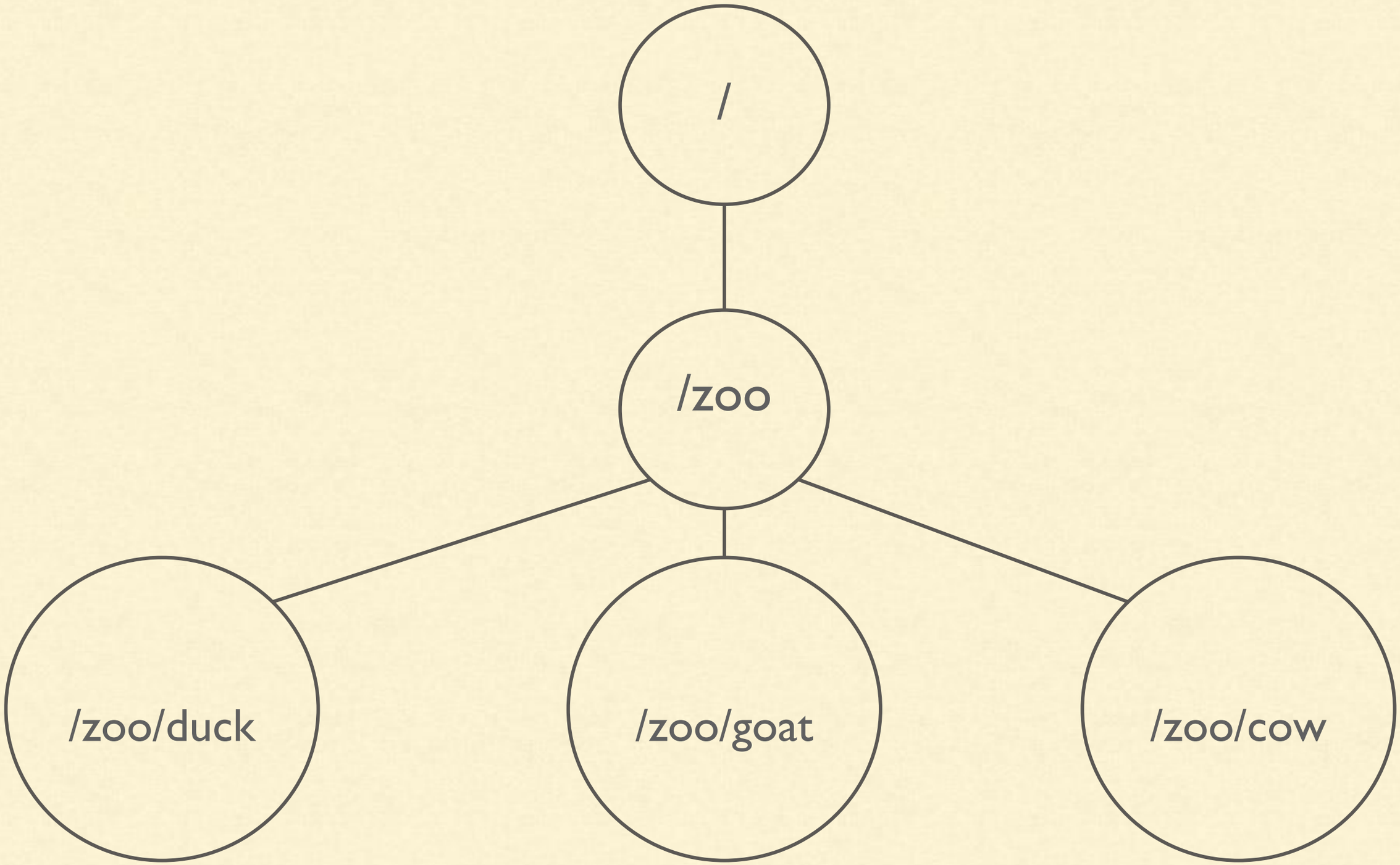
A Distributed Coordination Service for Distributed Applications

- Used for
  - Synchronisation
  - Locking
  - Maintaining configuration
  - Failover management
- Coordination service that does not suffer from
  - Race conditions
  - Dead locks

Data Model

- Think of it as highly available file system
- znode - can have data
- JSON data
- No append operation
- Data access (read/write) is atomic - either full or error
- znode - can have children
- znodes form a hierarchical namespace

Data Model - Contd.



Data Model - Znode - Types

- Persistent
- Ephemeral
- Sequential

Data Model - Persistent Znode

- Remains in ZooKeeper until deleted
- Create /mynode mydata

Data Model - Ephemeral Znode

- Deleted by Zookeeper as session ends or timeout
- Though tied to client's session but visible to everyone
- Can not have children, not even ephemeral ones
- create -e /apr9/myeph this-will-disappear

Data Model - Sequential Znode

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

create -s /zoo v Created /zoo0000000004	create -s /zoo/ v Created /zoo/0000000005
create -s /xyz v Created /xyz0000000006	create -s /zoo/ v Created /zoo/0000000007

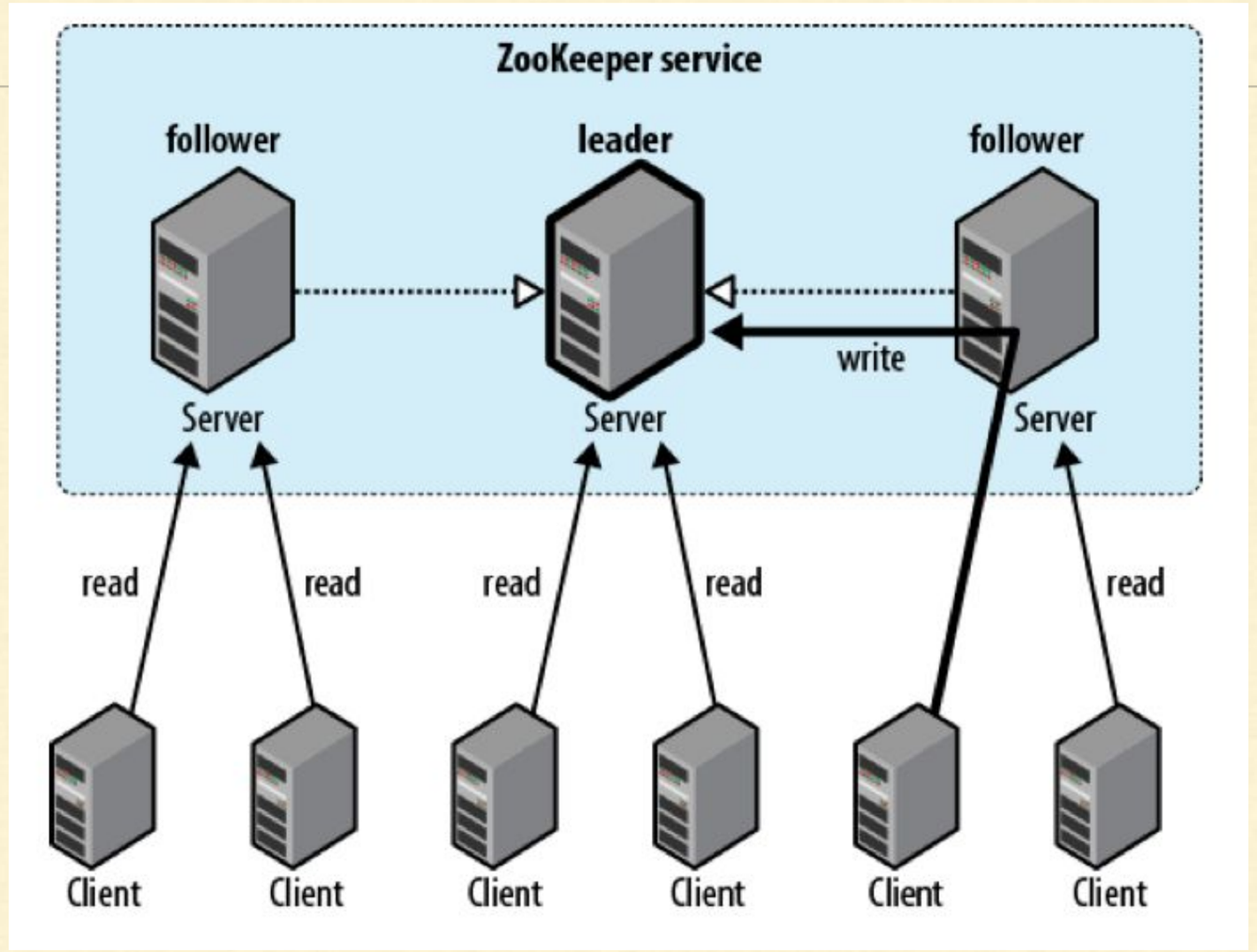
Architecture

Runs in two modes

- Standalone:
  - There is single server
  - For Testing
  - No High Availability
- Replicated:
  - Run on a cluster of machines called an ensemble
  - Uses Paxos Algorithm
  - HA
  - Tolerates as long as majority

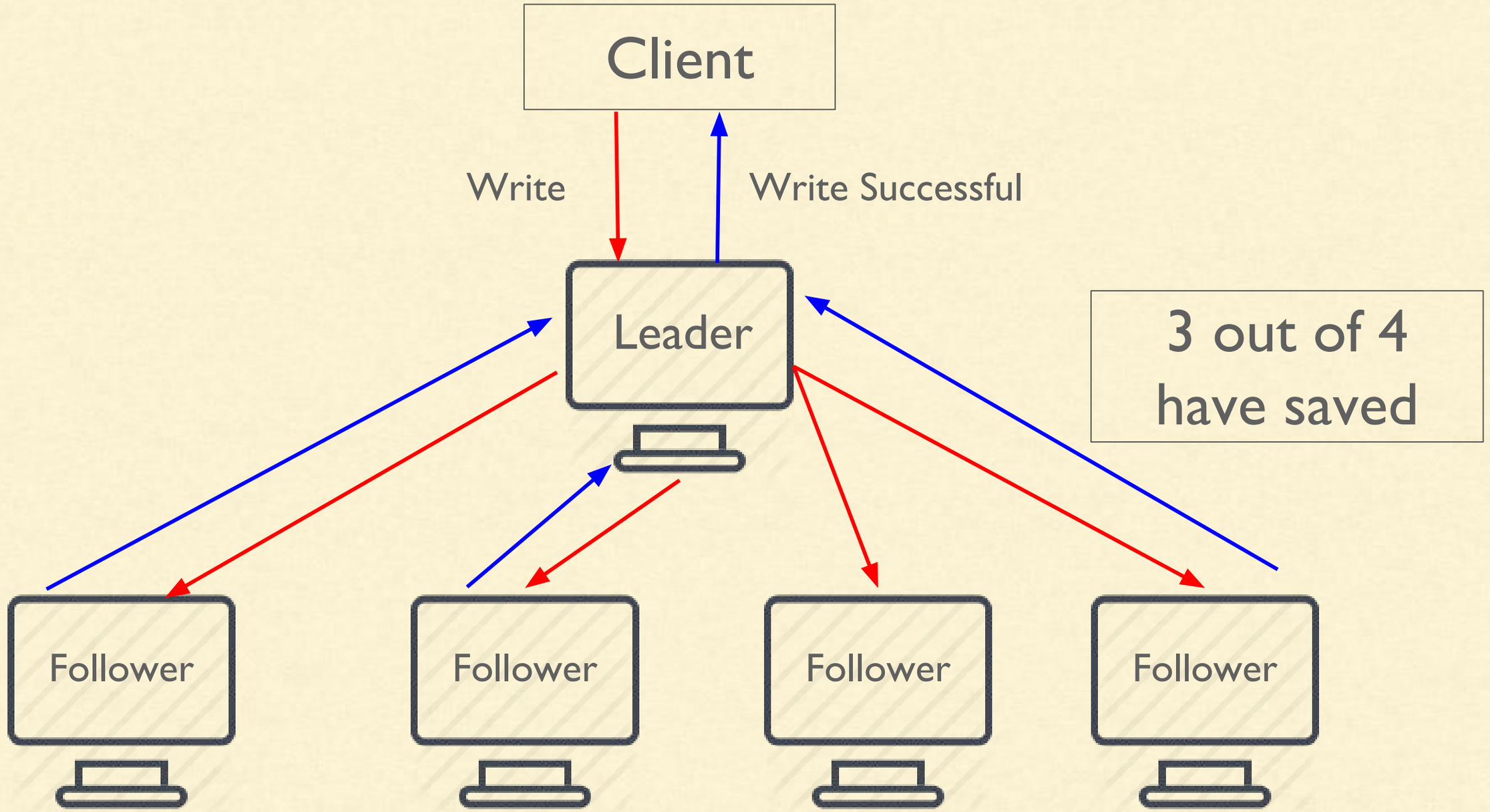
Architecture

Ensemble



- 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 within 200ms
  - If the majority is not available at any point of time, the leader steps down

Architecture - Phase 2

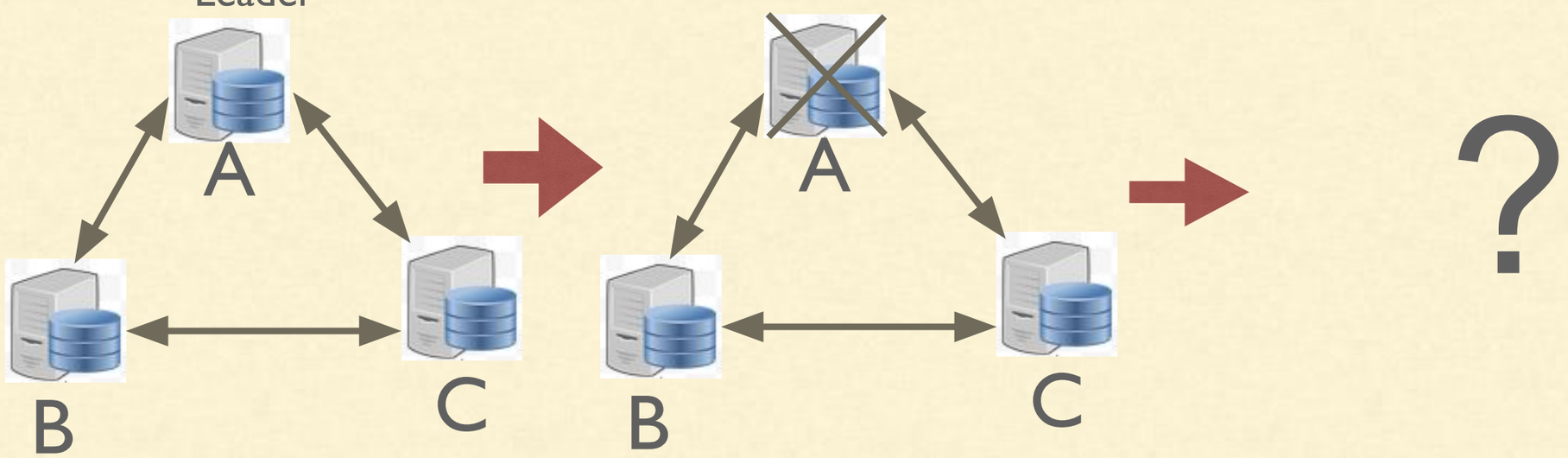


Architecture - Phase 2 - Contd.

- The protocol for achieving consensus is atomic like two-phase commit
- Machines write to disk before in-memory

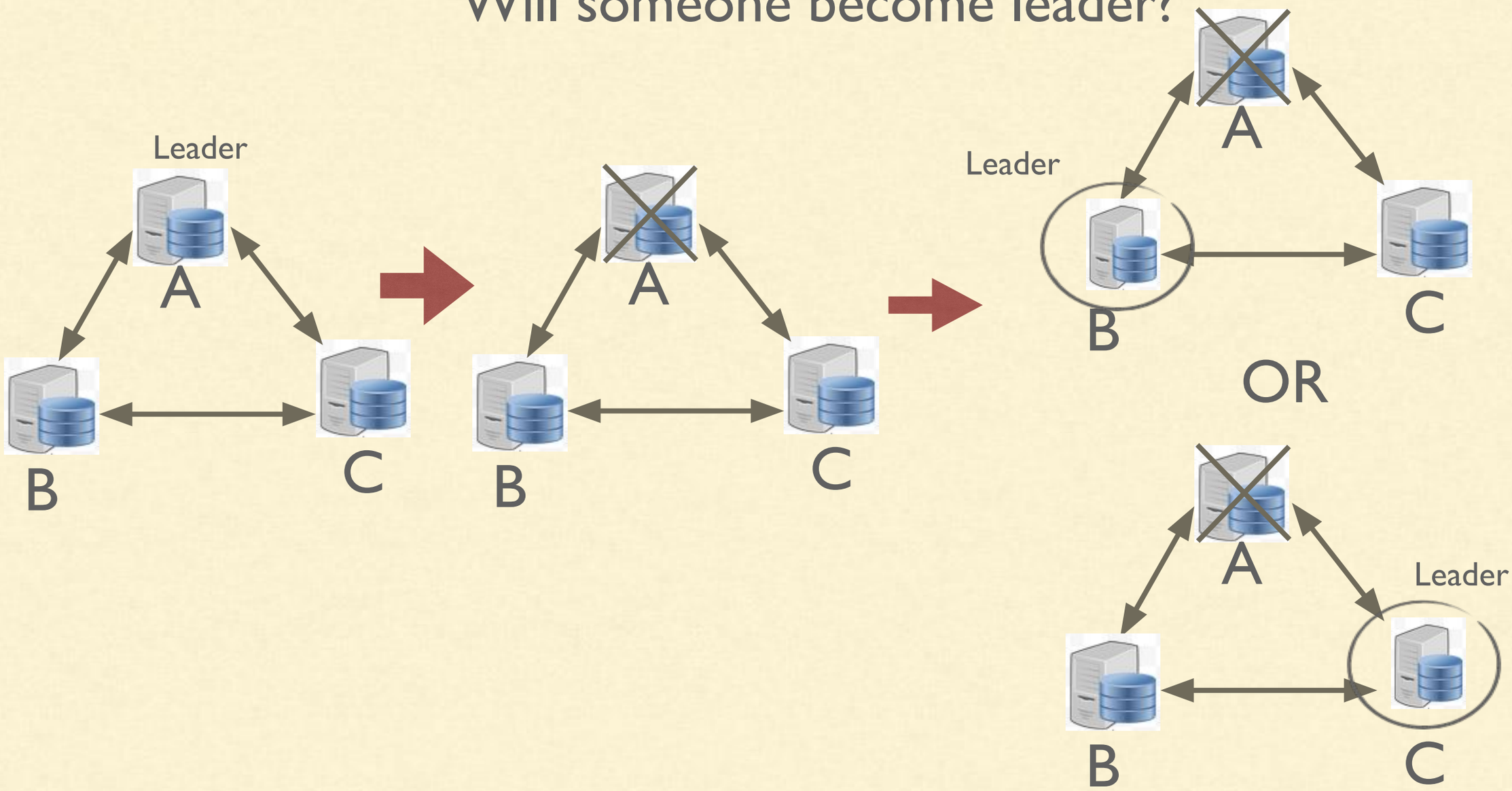
Election Demo

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?

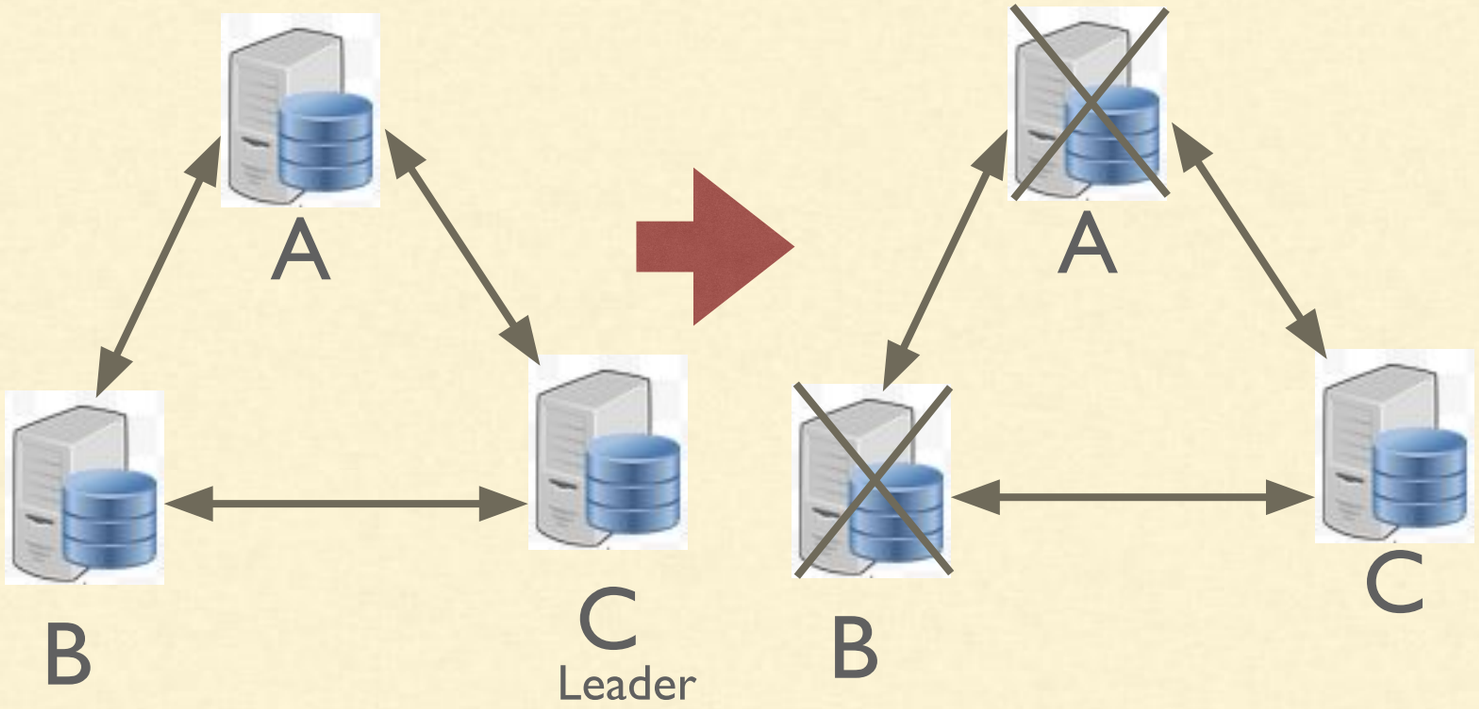


Majority Demo

If you have three nodes A, B, C with C as a leader

And A and B die.

Will C remain leader?

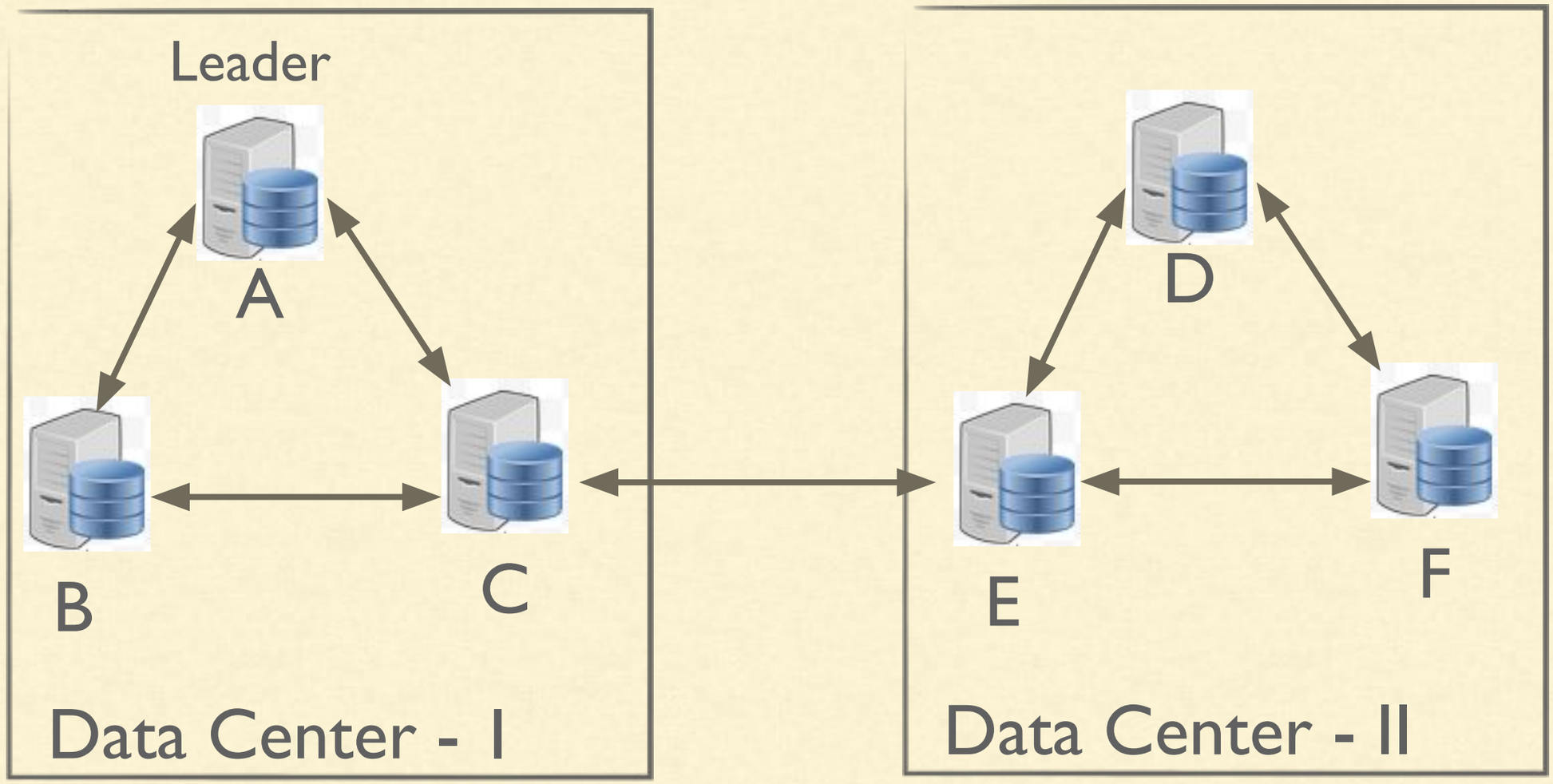


C will step down. No one will be the Leader as majority is not available.

Why Do We Need Majority?

Imagine

We have an ensemble spread over two data centres.



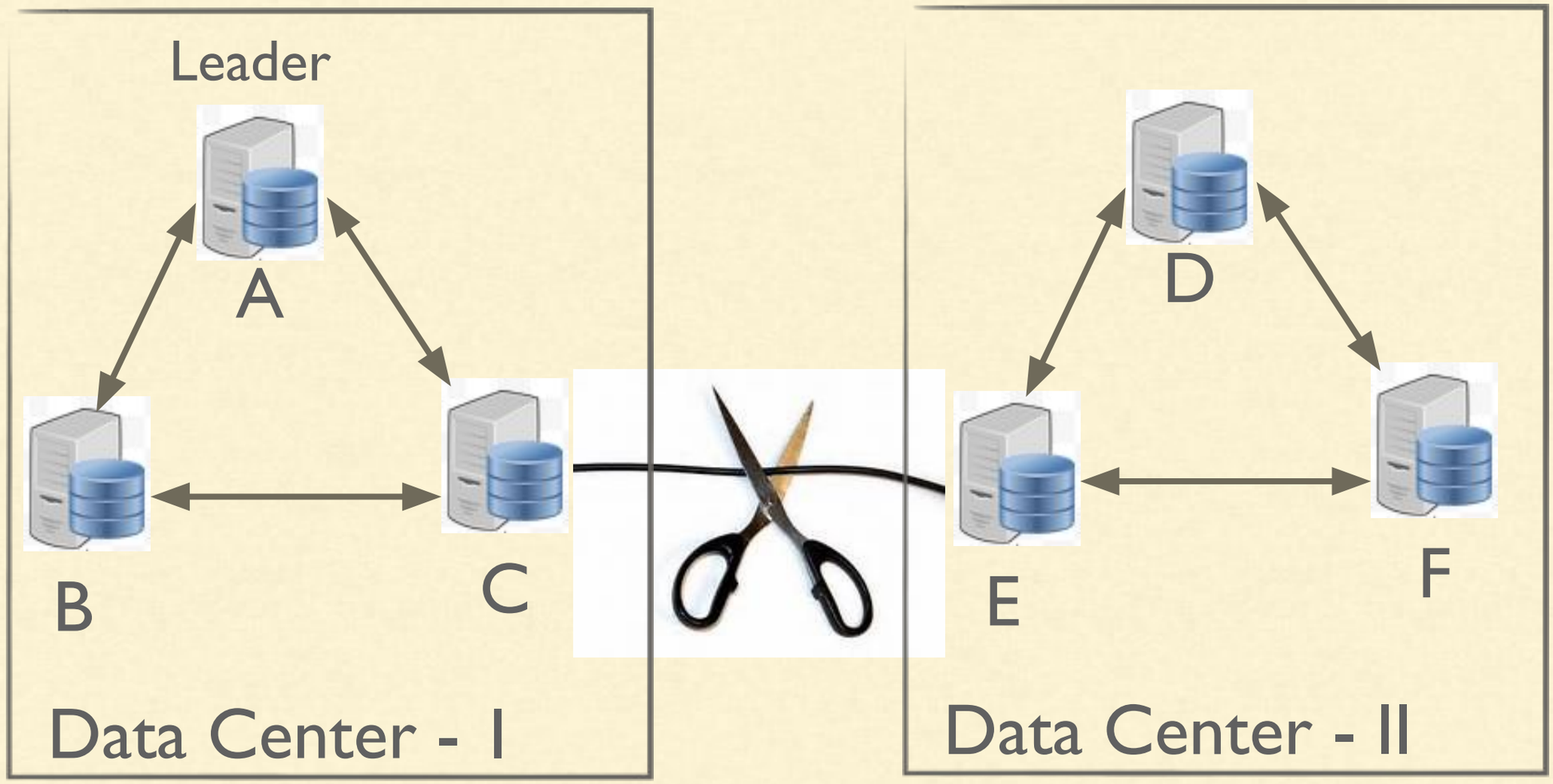
Why Do We Need Majority?

Imagine

The network between data centres got disconnected.

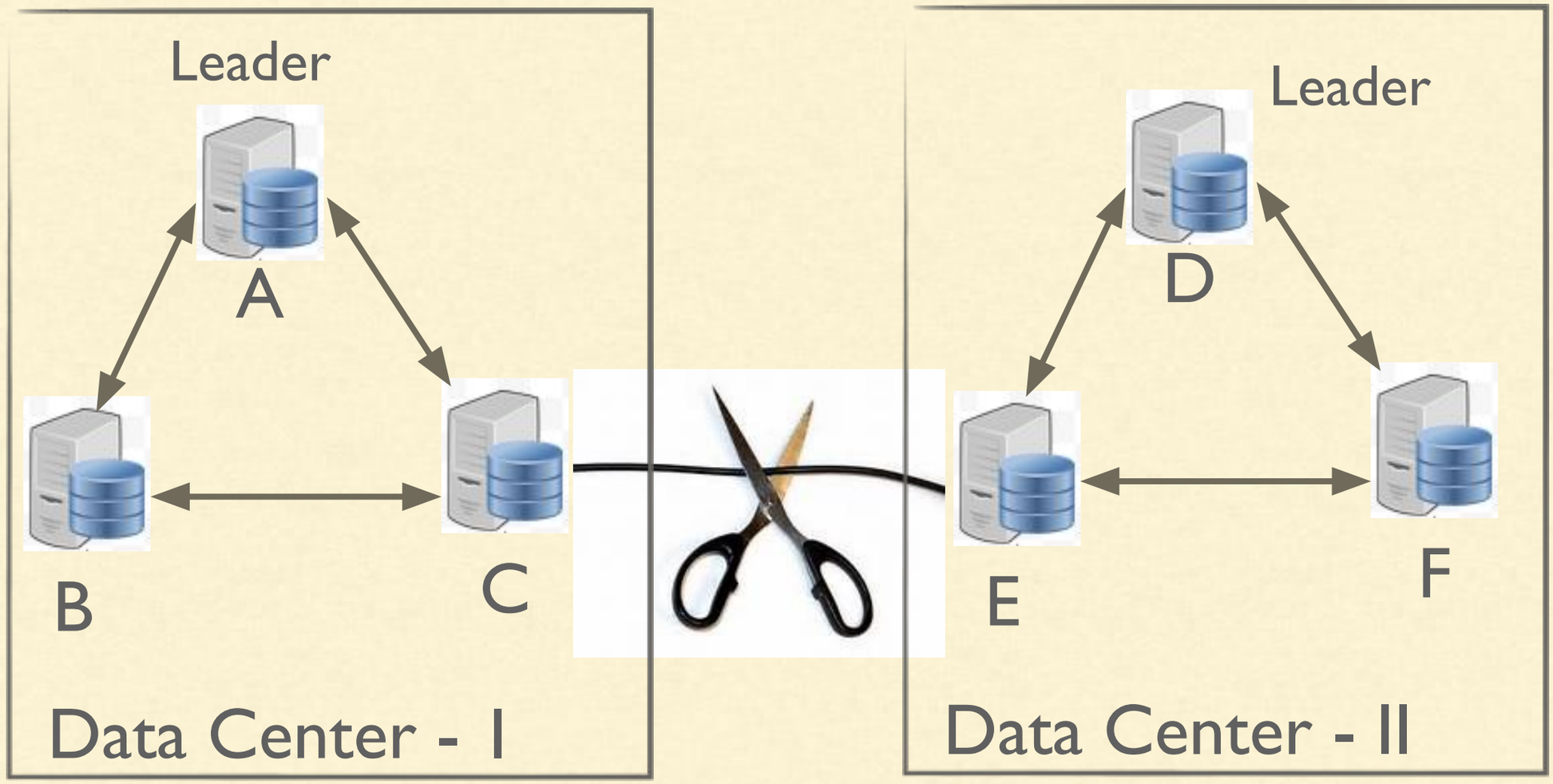
If we did not need majority for electing Leader,

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.



Question

An ensemble of 10 nodes can tolerate a shutdown of how many nodes?

4

Sessions

- 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

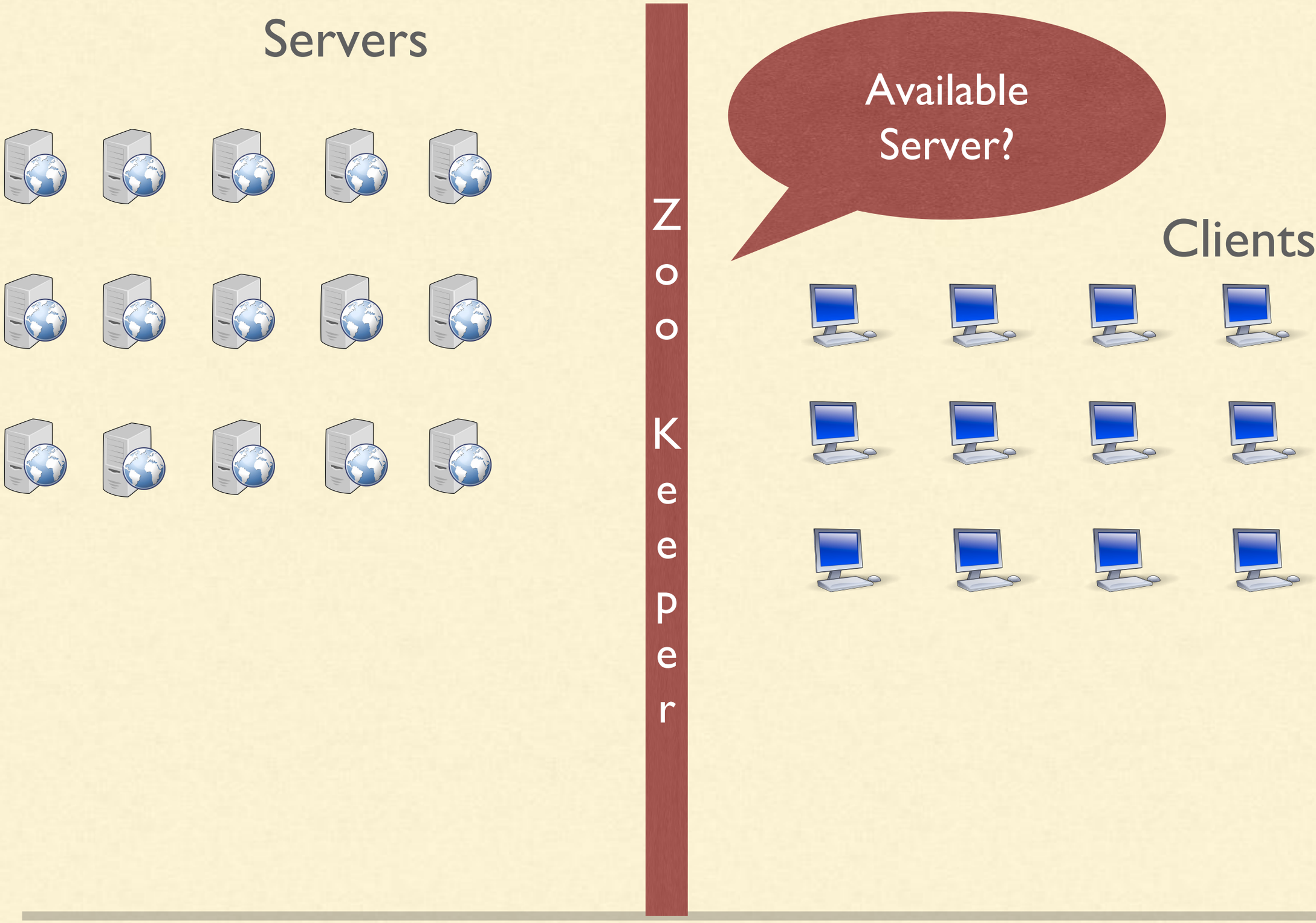
Sessions - Continued

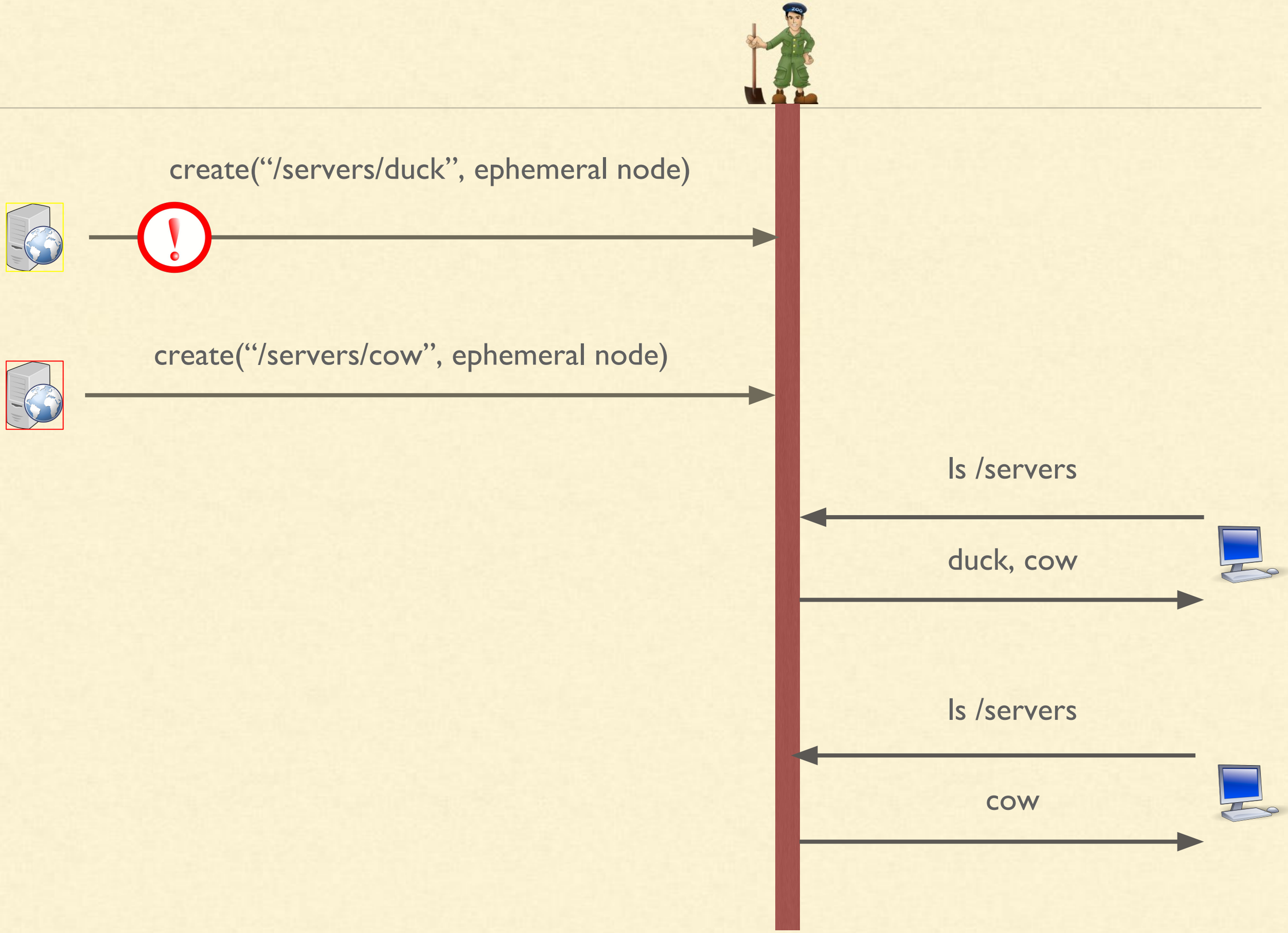
- If the server hasn't received a request within the timeout period, it may expire the session
- On session expiry, ephemeral nodes are deleted
- To keep sessions alive client sends pings (heartbeats)
- Client library takes care of heartbeats

Sessions - Continued

- 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 ops will fail during disconnection

Use Case - Many Servers - How Do They Coordinate?




create("/servers/duck", ephemeral node)
create("/servers/cow", ephemeral node)
ls /servers
duck, cow
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

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

Sync:

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

Async:

```
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 ops *exists*, *getChildren*, *getData* may have watches
- Watches are triggered by write ops: *create, delete, setData*
- ACL 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

Determines who can perform certain operations on it.

- ACL is the combination
  - authentication scheme,
  - an identity for that scheme,
  - and a set of permissions

- Depends on authentication:
  - 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

Big Data & Hadoop

Thank you.

+1 419 665 3276 (US)
+91 803 959 1464 (IN)

hadoop@knowbigdata.com

Subscribe to our Youtube channel for latest videos -
https://www.youtube.com/channel/UCxugRFe5wETYA7nMH6VGyEA

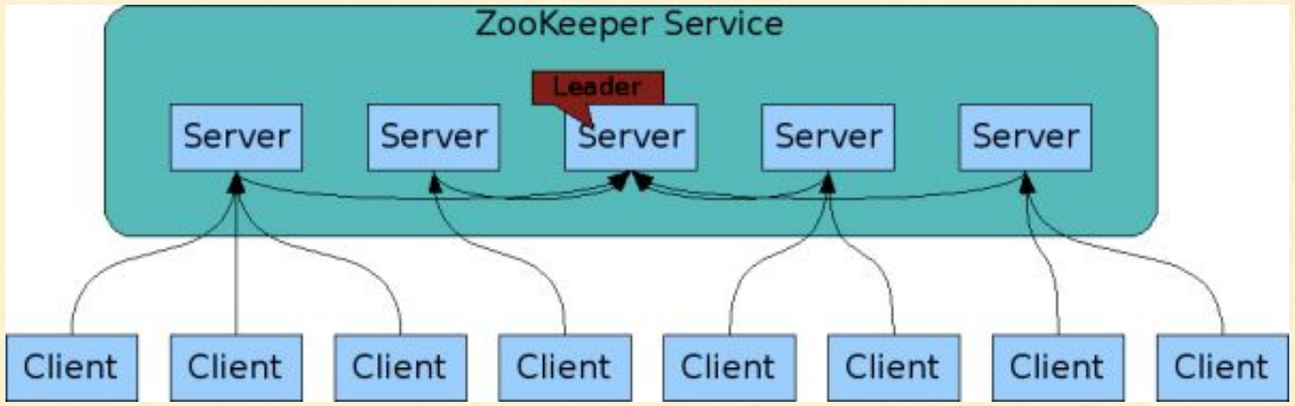
Design Goals

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 - HA



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

r u ok?

- Check connect to port 2181
  - echo ruok | nc localhost 2181

ruok	Check server state.
conf	server configuration (from zoo.cfg).
envi	server environment, versions and other system properties.
srvr	statistics,znodes, mode (standalone, leader or follower).
stat	server statistics and connected clients
srst	Resets server statistics.
isro	is it in read-only (ro) mode?

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);

Duck Server Killed

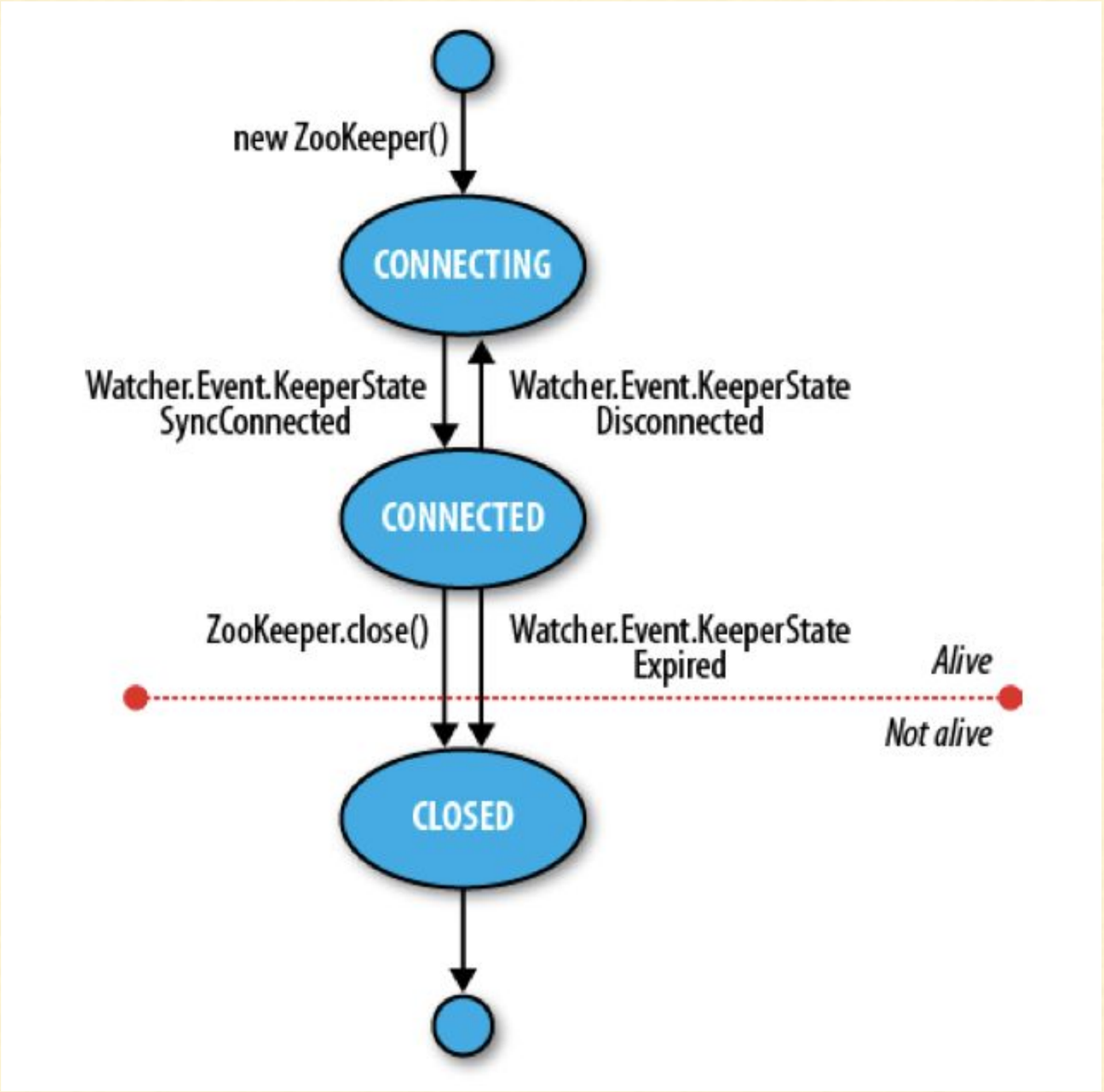
ls /servers

duck,cow

5 seconds later
ls /servers

cow

States

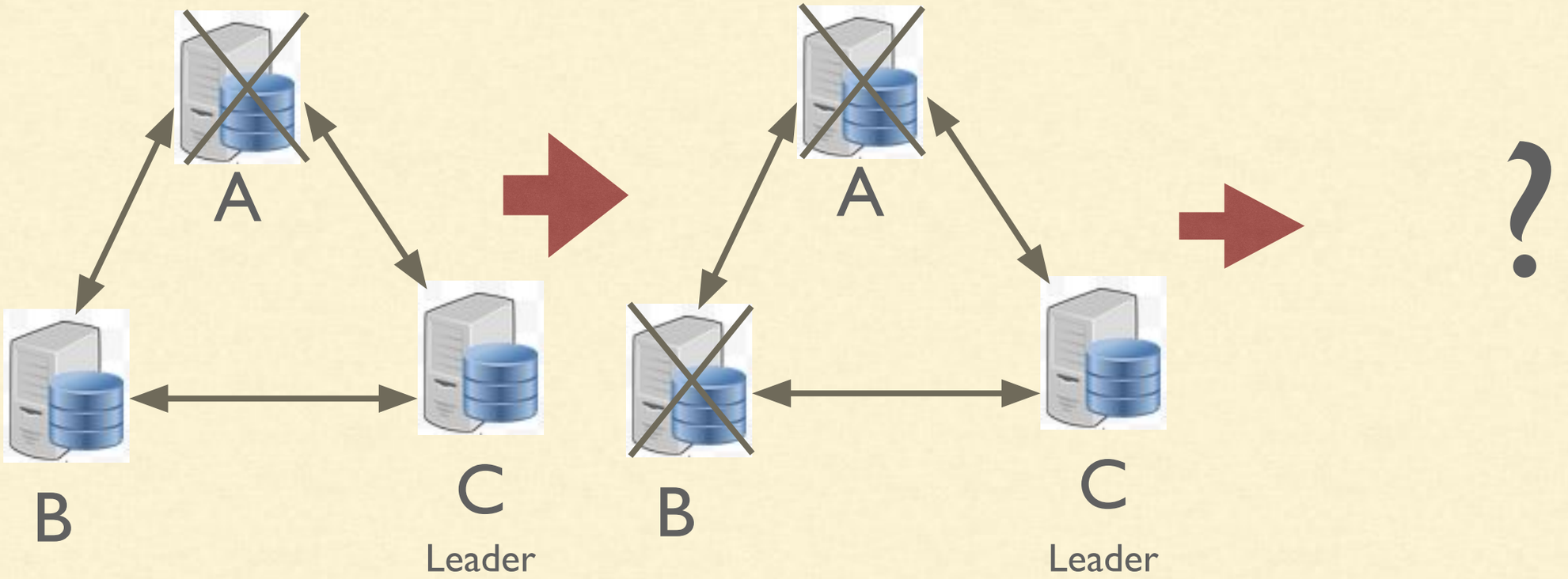

new ZooKeeper()
CONNECTING
Watcher.Event.KeeperState
SyncConnected
Watcher.Event.KeeperState
Disconnected
CONNECTED
ZooKeeper.close()
Watcher.Event.KeeperState
Expired
Alive
Not alive
CLOSED

Election Demo

If you have three nodes A, B, C

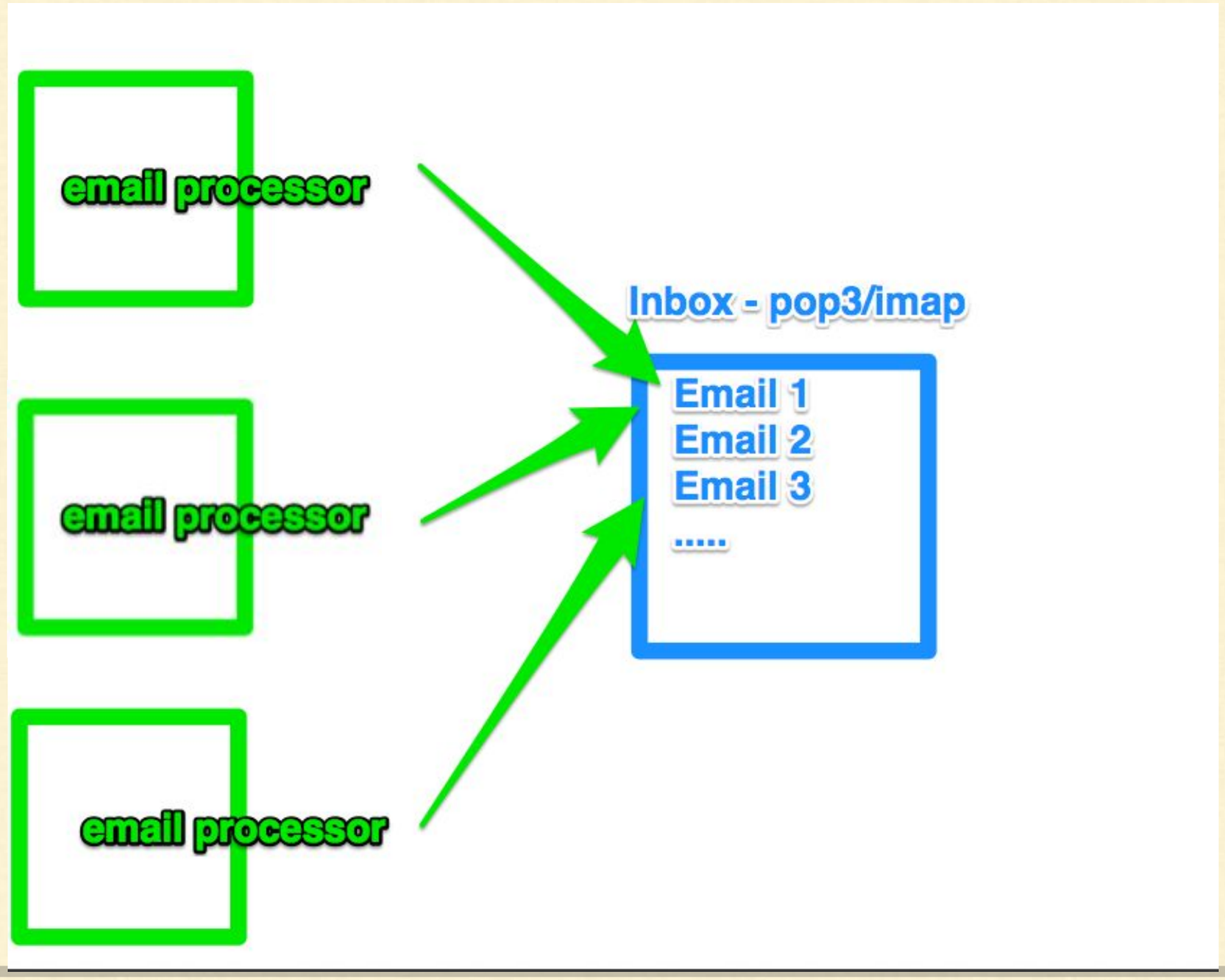
And A and B die.

Will C remain Leader?

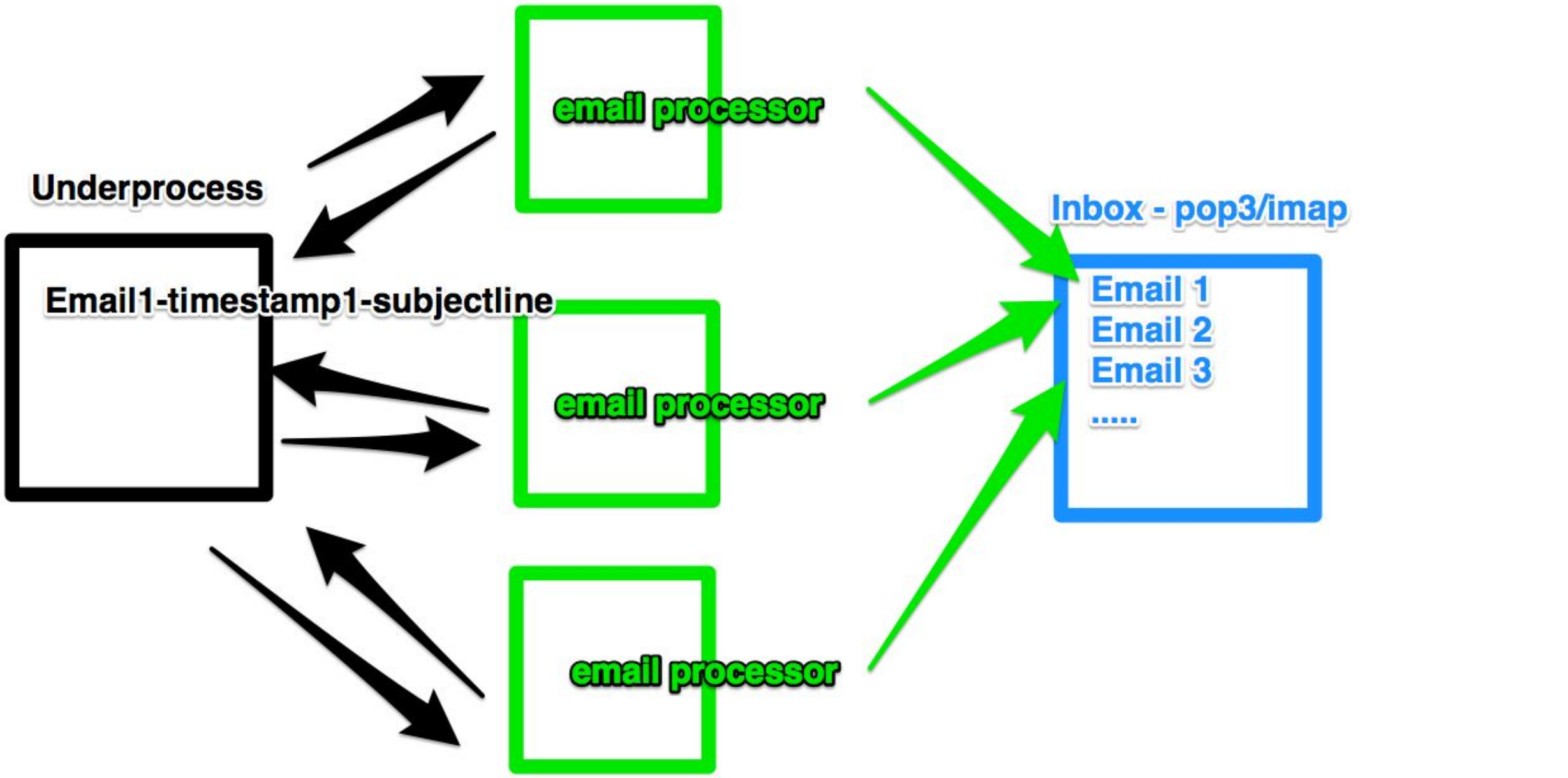


Coordination?

How would Email Processors avoid reading same emails?



Coordination?

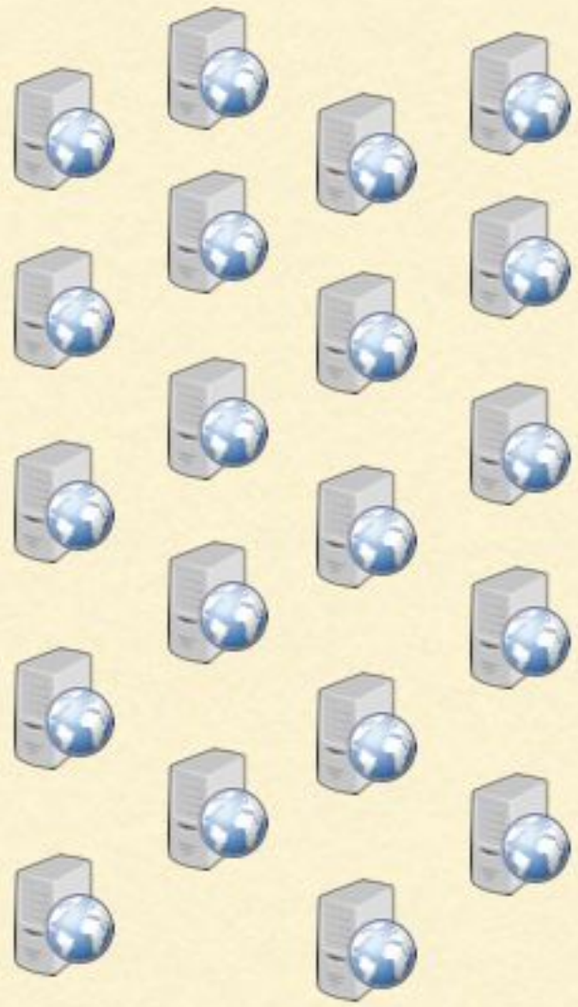


Getting Started

- List the znodes at the top level: ***ls /***
- List the children of a znode *brokers*: ***ls /brokers***
- Get the details about the znode: ***get /brokers***

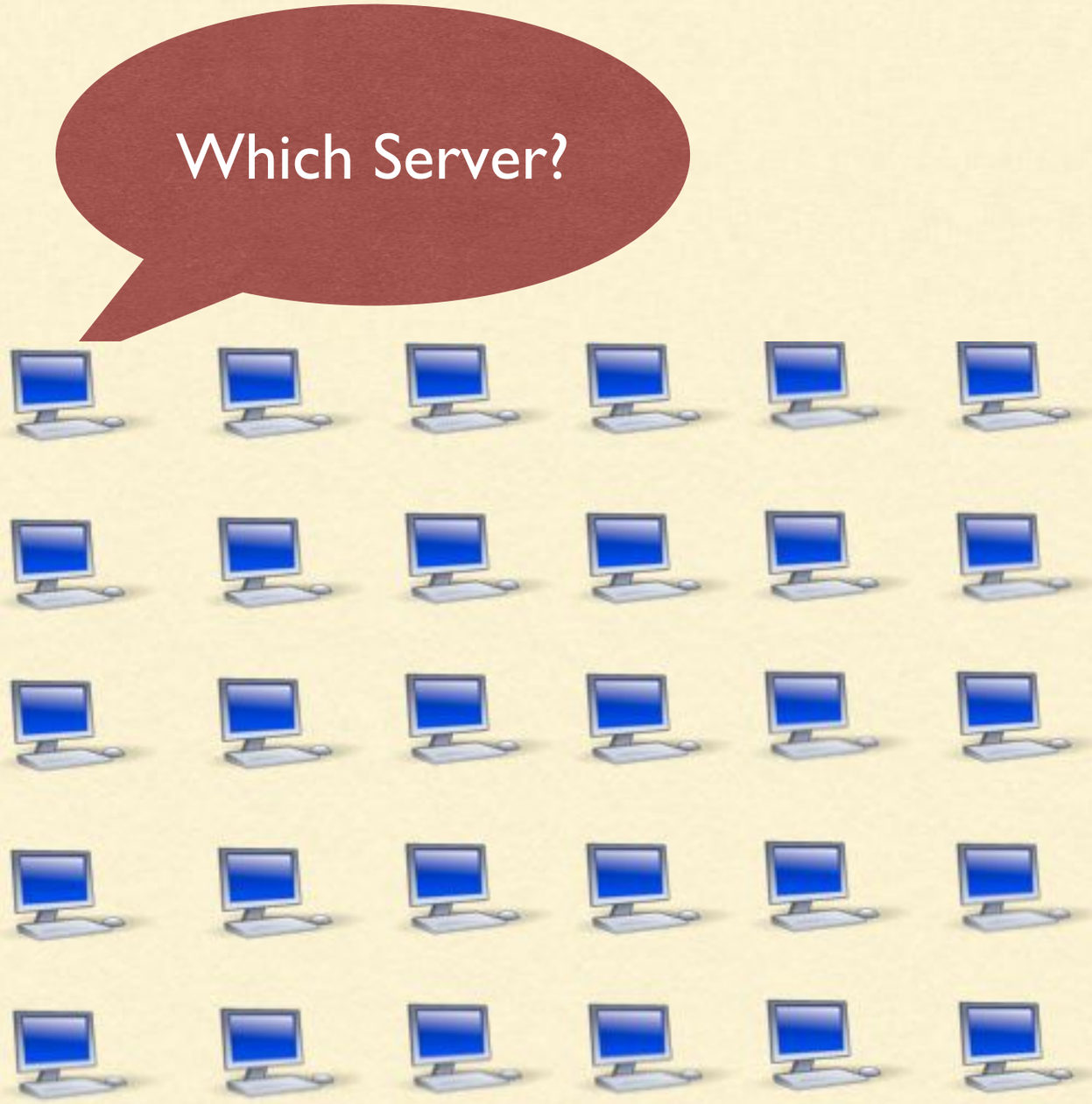
Use Case - Many Servers - How Do They Coordinate?

Servers



Can’t keep list on single node how to remove a failed server?

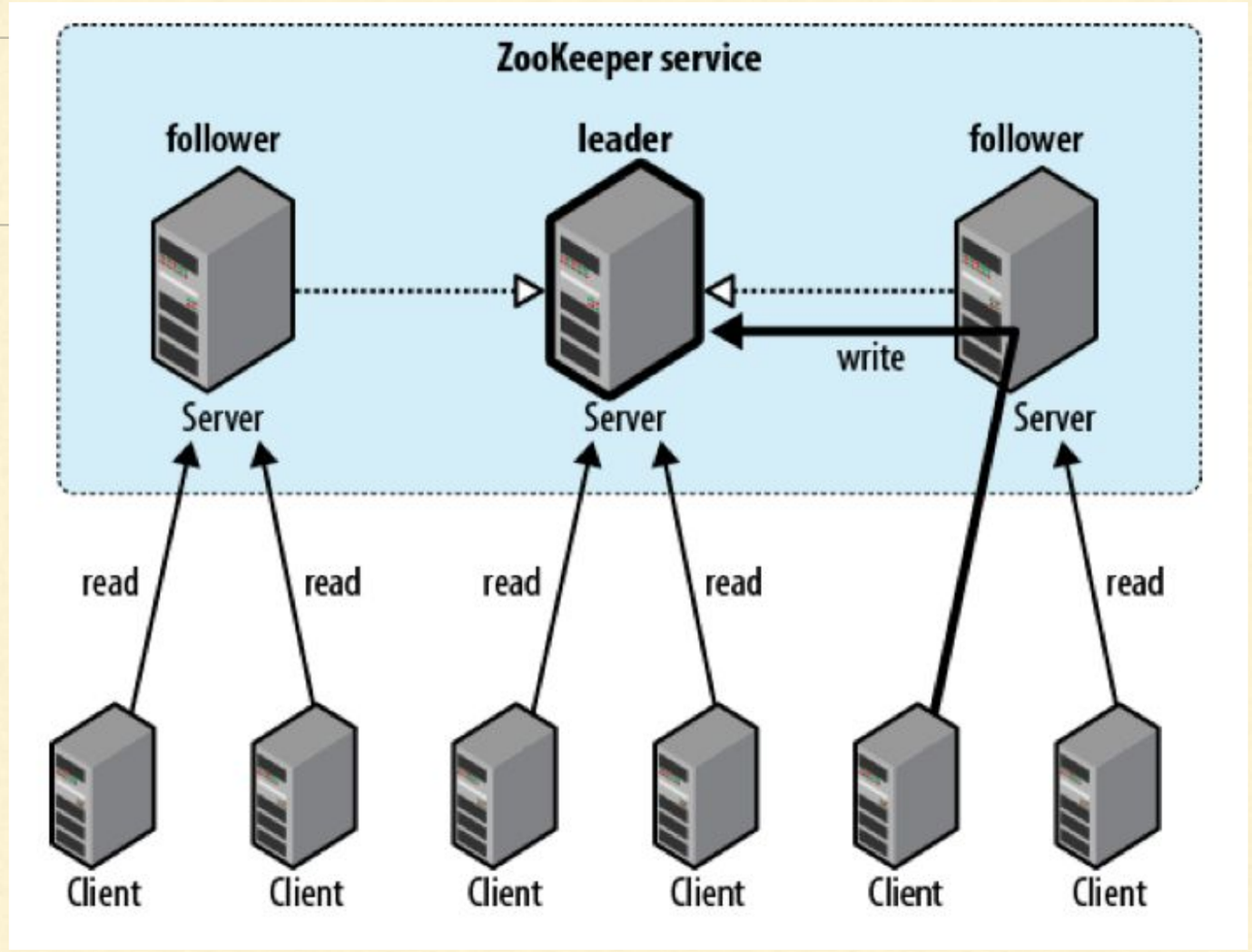
zookeeper



Data Model - Continued

- /a/./b != /a/b
- /a/./b, /a/../b, /a//b is invalid

Architecture



- Phase 2: Atomic broadcast
  - All write requests are forwarded to the leader,
  - Leader broadcasts the update to the followers
  - When a majority have saved (persisted) the change:
    - The leader commits the update
    - The client gets success response