What Are Some Characteristics of Cloud-Native Applications?

A Partial List

- They scale
- They are fault tolerant
  - Crashes of a component don’t bring down the entire system
- They can grow and shrink dynamically
  - You don’t need to restart the whole system to add, update, or remove individual parts

They operate continuously and evolve without downtime over a wide variety of operating conditions.

Cloud Native Application Design Implies Asynchronicty

Highly available, tolerant of weak global state consistency

CAP Theorem

- It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees
  - ***Consistency***: Every read receives the most recent write or an error
  - ***Availability***: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
  - ***Partition tolerance***: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

https://en.wikipedia.org/wiki/CAP_theorem

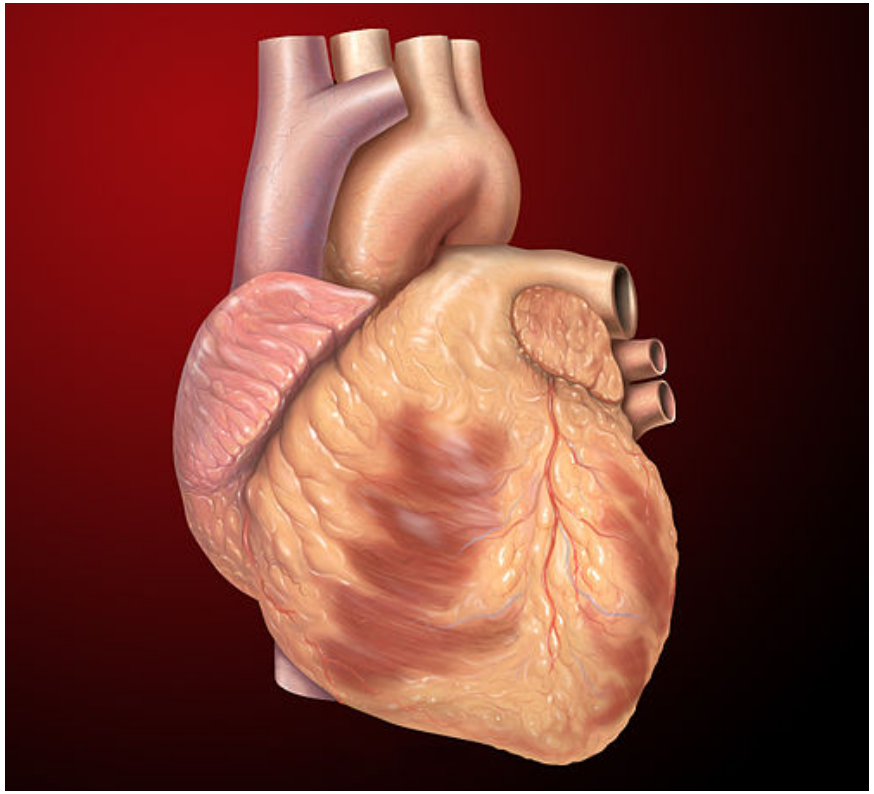
Distributed State and Coordination Management: Consul, ETCD, Zookeeper

HashiCorp Consul

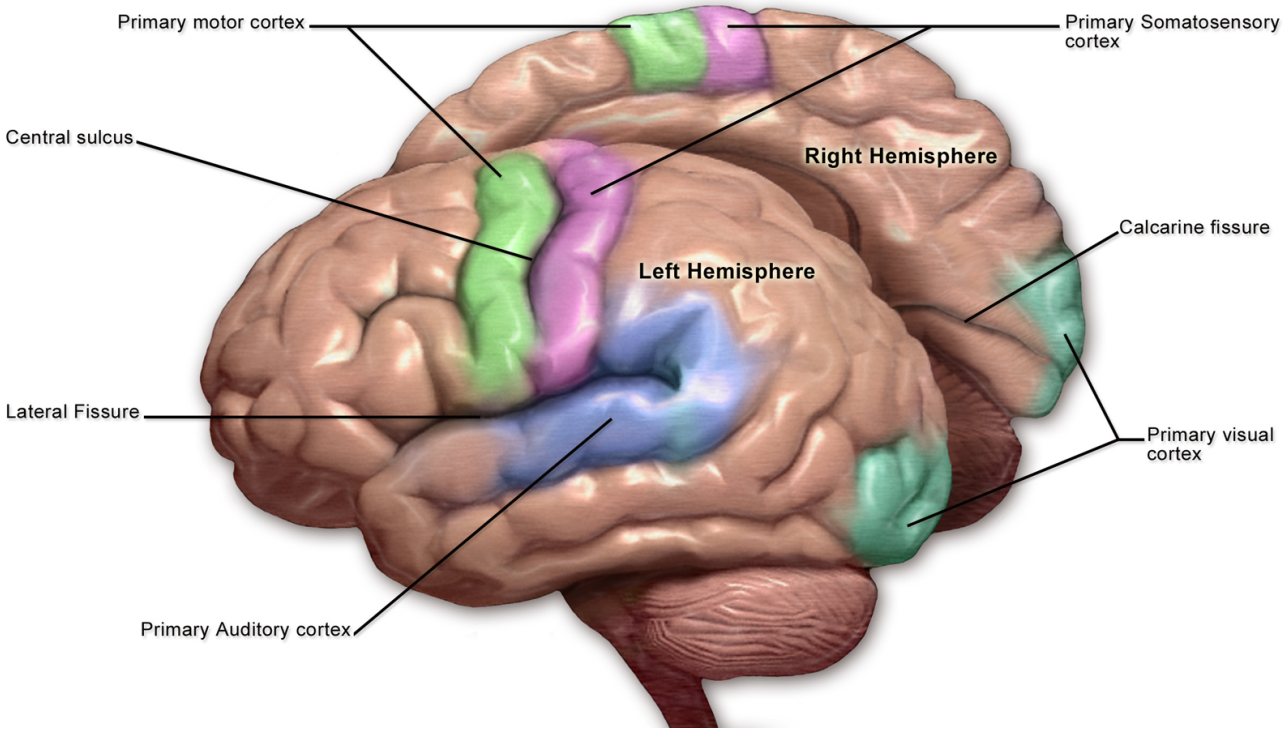
Key Takeaways

- All these systems use hierarchically arranged key-value pairs for storing small bits of useful information
  - Tree node and leaf structure (like a file system)
- Consul, ETCD, and Zookeeper are internally interesting distributed computing systems in their own right
- Zookeeper is the best documented, so we'll base lecture on it.
- But Consul and ETCD are more relevant these days
  - As you might have guessed from the logos

Messaging, Data Plane



Information, Control Plane

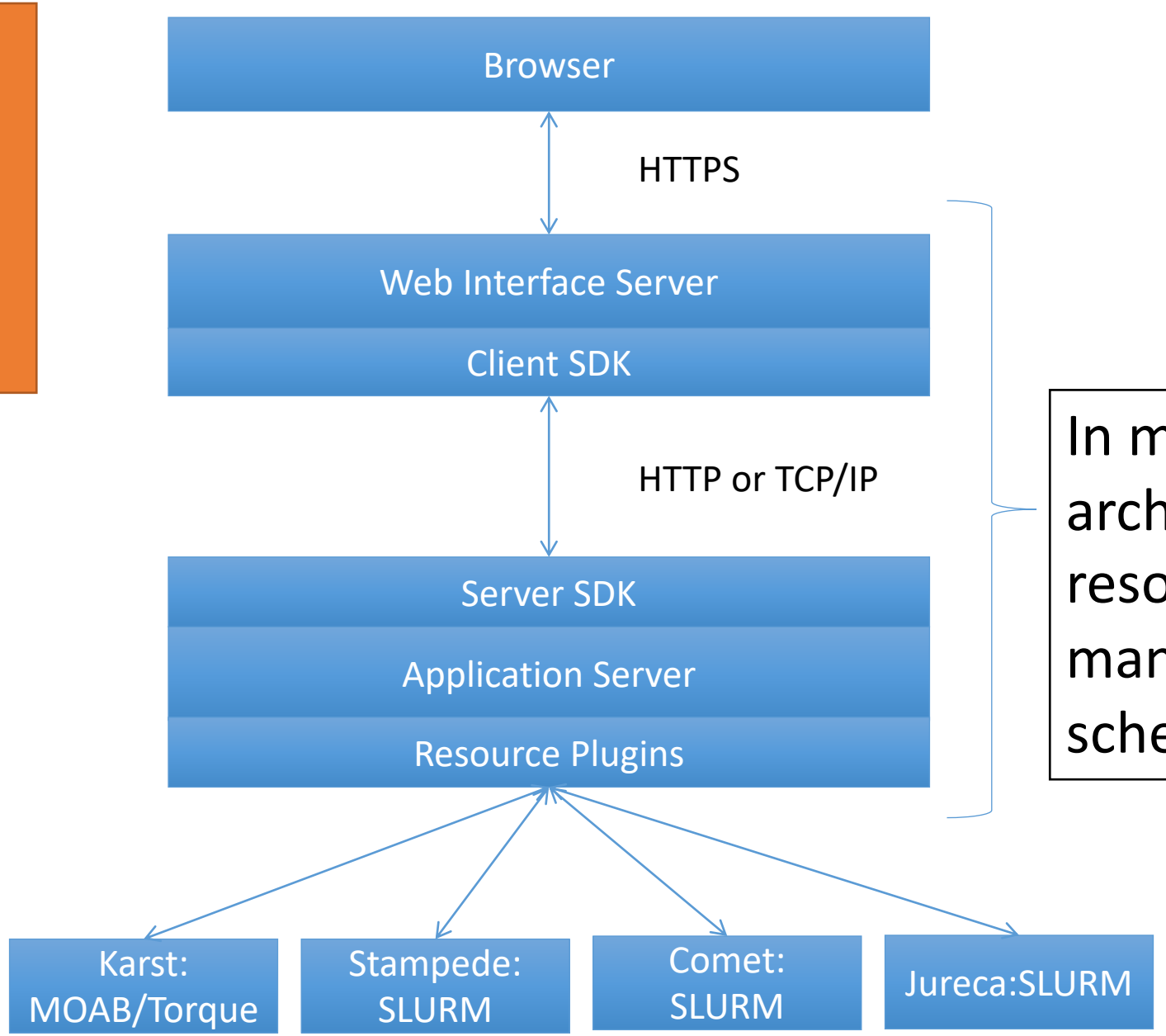


Apache Zookeeper

Hunt, P., Konar, M., Junqueira, F.P. and Reed, B., 2010, June. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In USENIX Annual Technical Conference (Vol. 8, p. 9).

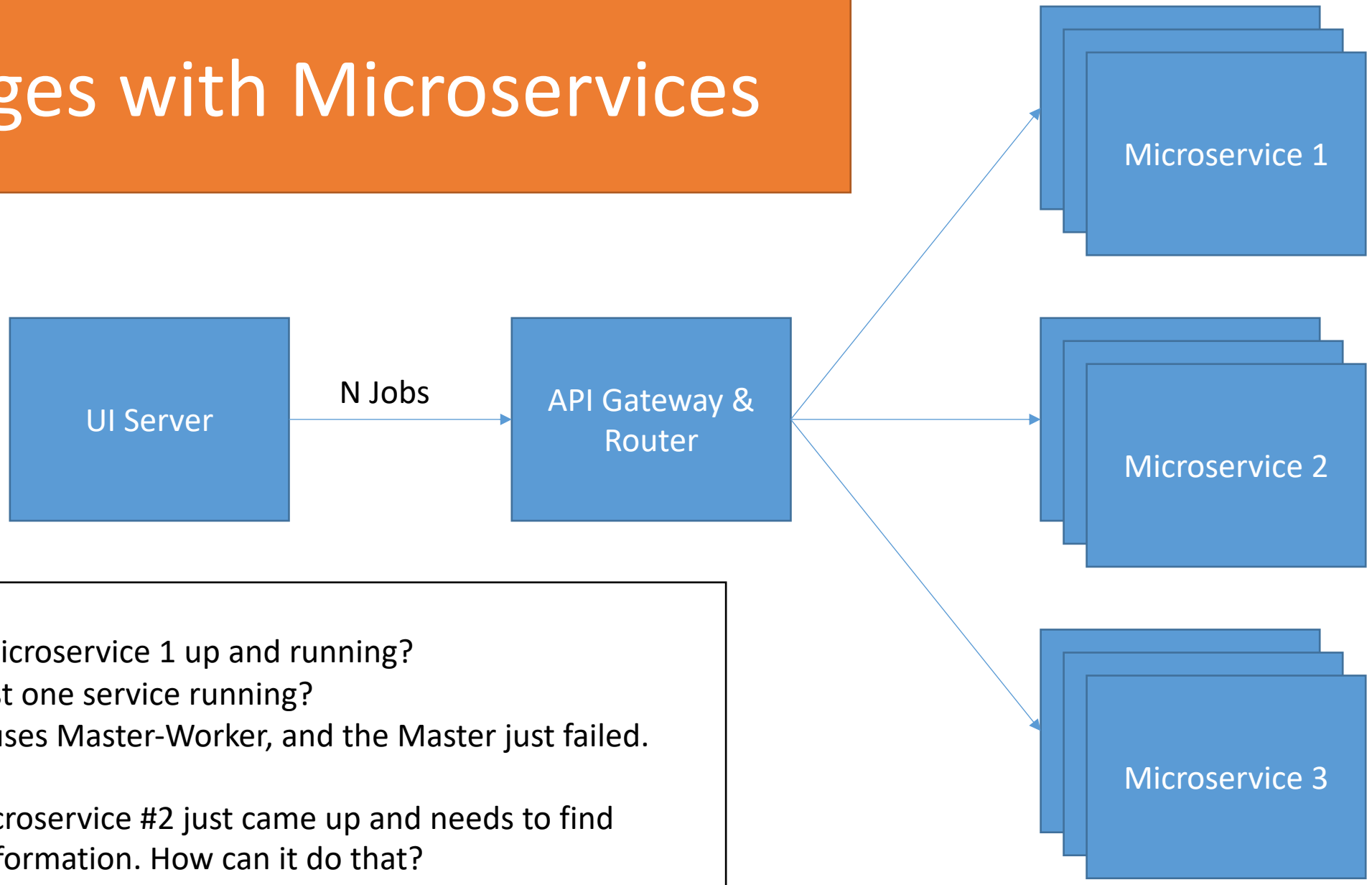
And other papers from http://scholar.google.com. ZK is a great example of successful open source software with strong underlying CS principles documented in scholarly articles.

Recall the Gateway Octopus Diagram: this is a “north-south” monolithic application.



In micro-service arch, these need resource management and scheduling

Challenges with Microservices



Some questions:

- Is Replica 2 of microservice 1 up and running?
- Do I have at least one service running?
- Microservice 3 uses Master-Worker, and the Master just failed. What do I do?
- Replica 2 for microservice #2 just came up and needs to find configuration information. How can it do that?

Apache Zookeeper and Microservices

- ZK can manage information in your system
  - IP addresses, version numbers, and other configuration information of your microservices.
  - This is stuff that is dynamic or otherwise doesn't fit into a static configuration file.
- The health of the microservices.
- The state of a calculation.
- Group membership

Service Discovery

A new microservice has joined. How do other services learn about it? What do they need to know?

System Health

What microservices are down, unreachable, or acting strangely? What do you do about it?

Runtime Server Configuration

What type of information does a new microservice need to know when it comes up?

Rendezvous Problem

What happens if essential configuration information isn’t available to a server when it starts? For example, who is the leader?

Configuration Changes

A configuration has been changed. How do you notify every service that needs to update its settings?

Group Membership for Services

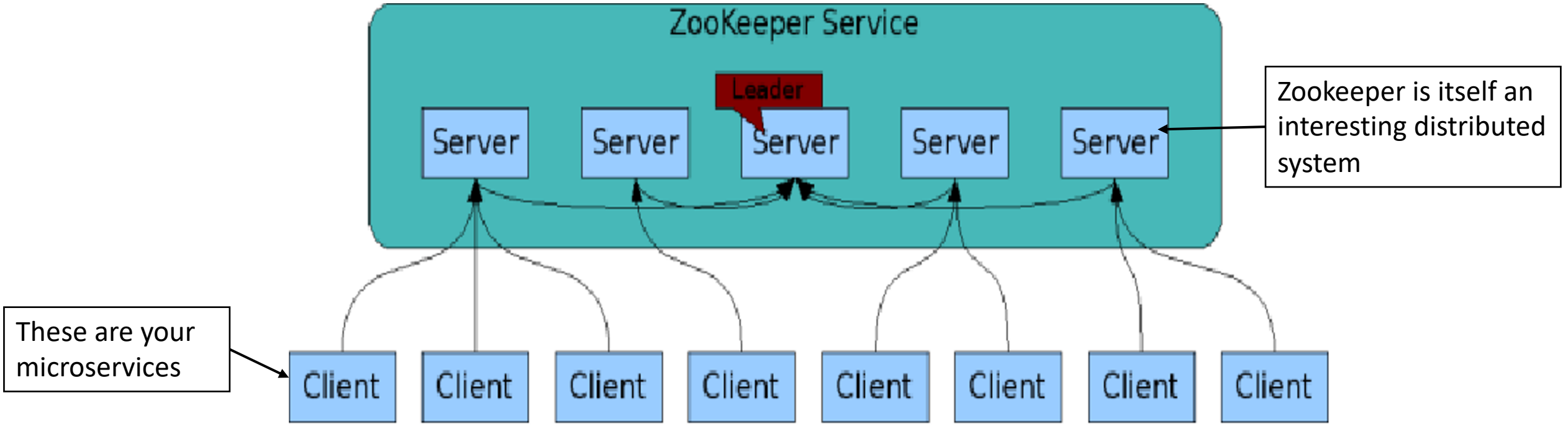
How can I find all the members of a specific type of service?

Locks

Locks allow multiple service instances and service types to modify shared resources, like files or configuration information. How do I do this?

Leader Election

If a master in a master-worker configuration goes down, how do I choose a new leader?



- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data in memory (!)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server & maintains a TCP connection.
- Client can read from any Zookeeper server.
- Writes go through the leader & need majority consensus.

https://cwiki.apache.org/confluence/display/ZOOKEEPER/ProjectDescription

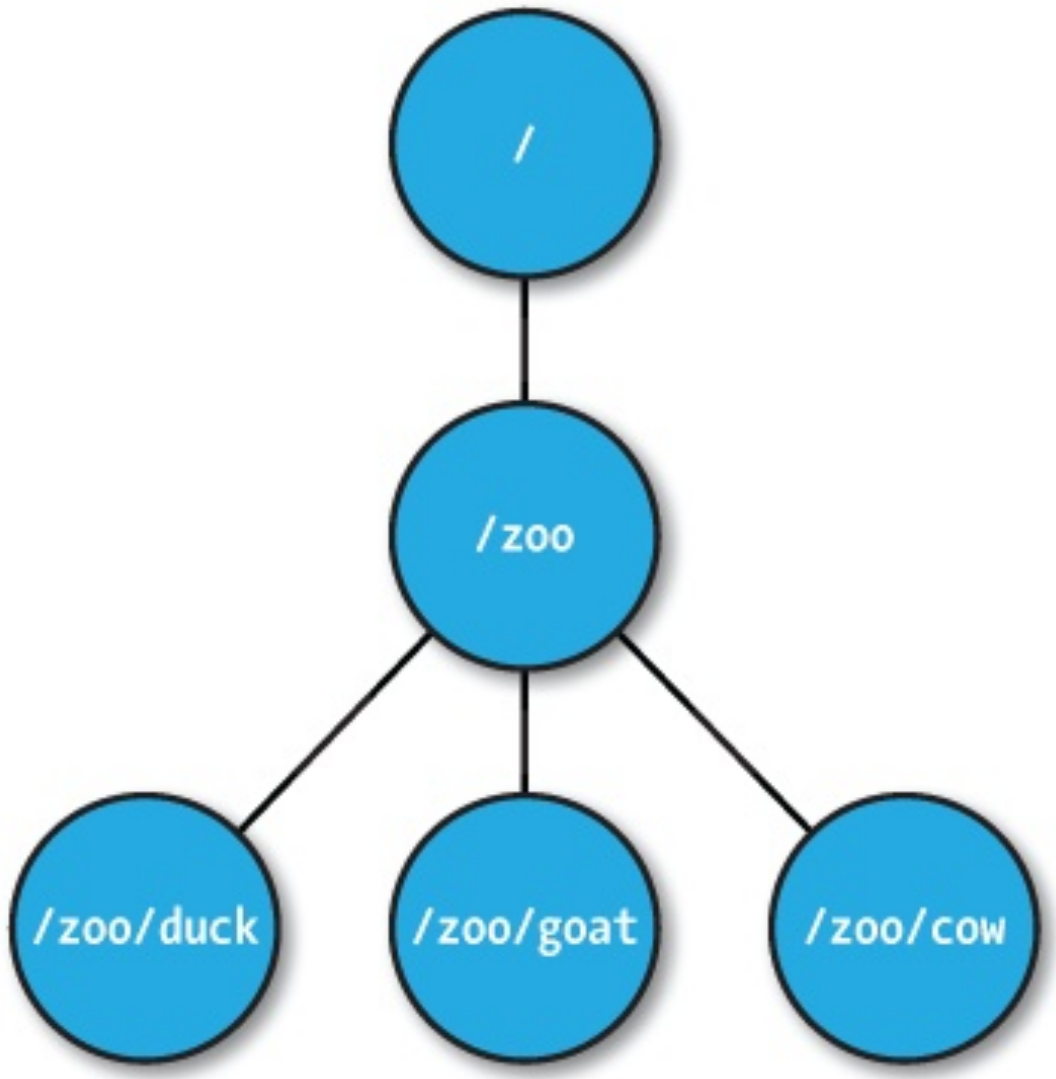
Zookeeper, Briefly

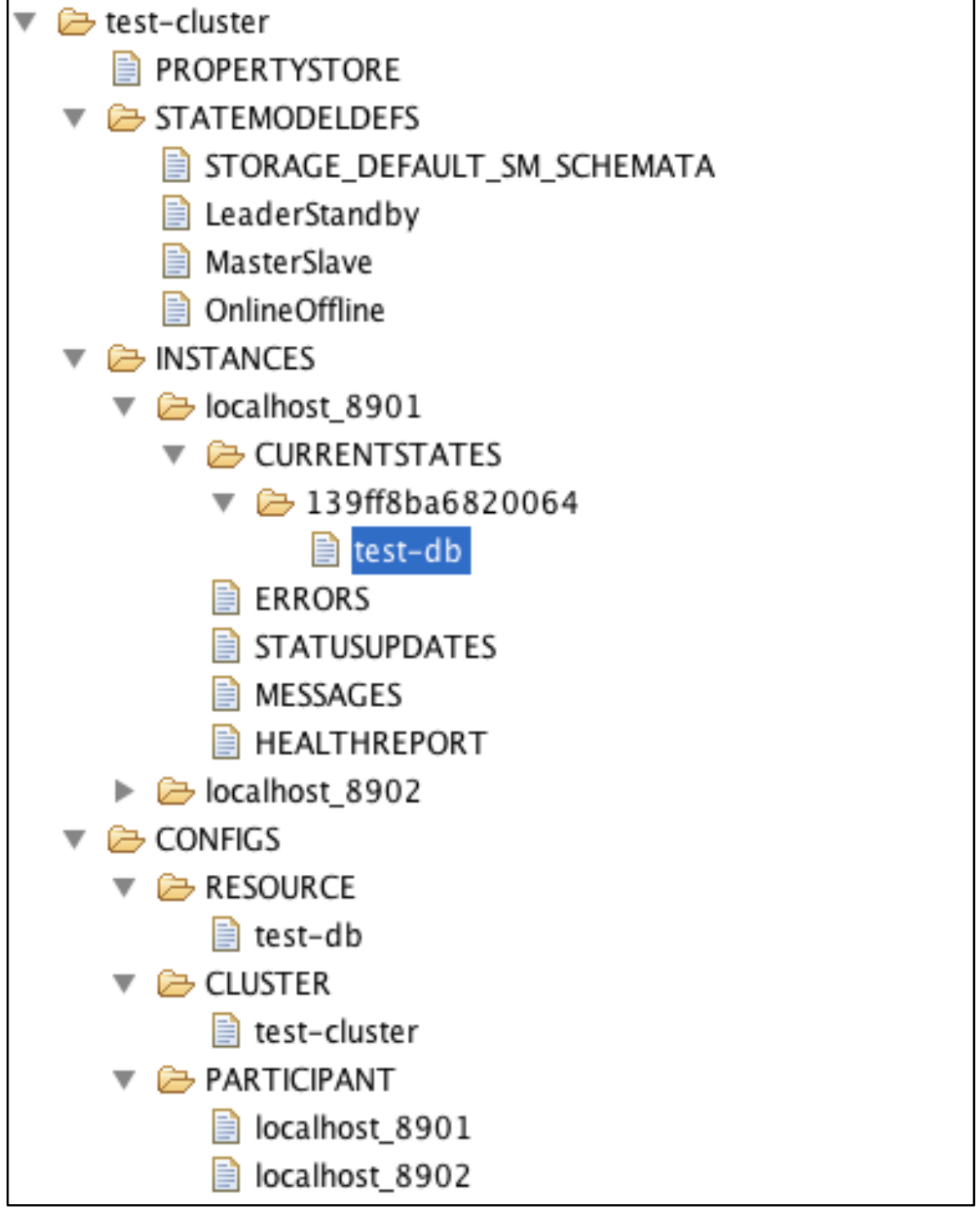
- Zookeeper Clients (that is, your applications) can create and discover nodes on Zookeeper trees
- Clients can put small pieces of data into the nodes and get small pieces out.
  - 1 MB max for all data per server by default
  - Each node also has built-in metadata like its version number.
- Even the existence/non-existence of nodes can be useful information
- You could build a small DNS or an LDAP server with Zookeeper.
- It can support several other interesting uses

Can you think of ways that this would be useful?

Zookeeper Trees Consist of ZNodes

- Znodes maintain data with version numbers and timestamps.
- Version numbers increases with changes
- Data in a node are read and written in its entirety




test-cluster
PROPERTYSTORE
STATEMODELDEFS
STORAGE_DEFAULT_SM_SCHEMATA
LeaderStandby
MasterSlave
OnlineOffline
INSTANCES
localhost_8901
CURRENTSTATES
139ff8ba6820064
test-db
ERRORS
STATUSUPDATES
MESSAGES
HEALTHREPORT
localhost_8902
CONFIGS
RESOURCE
test-db
CLUSTER
test-cluster
PARTICIPANT
localhost_8901
localhost_8902

ZNode types

Regular

- Clients create and delete explicitly

Ephemeral

- Like regular znodes associated with sessions
- Deleted when session expires

Both Regular and Ephemeral Nodes can be **Sequential:** the name name includes a universal, monotonically increasing counter

Zookeeper API (1/2)

- **create(path, data, flags)**: Creates a znode with path name path, stores data[] in it, and returns the name of the new znode.
  - *flags* enables a client to select the type of znode: regular, ephemeral, and set the sequential flag;
- **delete(path, version)**: Deletes the znode path if that znode is at the expected version
- **exists(path, watch)**: Returns true if the znode with path name path exists and returns false otherwise.
  - Note the *watch* flag

Zookeeper API (2/2)

- **getData(path, watch)**: Returns the data and meta-data, such as version information, associated with the znode.
- **setData(path, data, version)**: Writes data[] to znode path if the version number is the current version of the znode
- **getChildren(path, watch)**: Returns the set of names of the children of a znode
- **sync(path)**: Waits for all updates pending at the start of the operation to propagate to the server that the client is connected to.

Service Discovery

- Have each new instance create an ephemeral node when it joins
  - /root/services/*newService*
- The content of the node can contain useful information about the service
  - Its IP address
- Querying the children of /root/services will get a service list.
- You should also group them.
  - /root/services/dataStaging/*newService*

Group Membership for Services

- We just saw this: /root/services/dataStaging/*newService*
- Or whatever works for you
  - /root/services/dataStaging/members/*newService*
- Path names are just conventions
- You may want to define a few bootstrapping paths in static configuration files.
- Need unique names? Create new nodes with sequential flag
  - /root/services/dataStaging/members/*newService_3*

System Health

- Builds on Service Discovery
- Ephemeral nodes get deleted if the client that creates them fails.
  - A service can also delete these nodes by choice
- Other clients can put *watches* on these nodes.
  - If the node is deleted, a watch notification is fired

Runtime Server Configuration

- Assume services need run time configuration that they read after they come up
  - For example, which queues do I read from or write to?
  - Other examples?
- Put this info into Zookeeper as a standard node
  - /root/services/dataStaging/config
- Services in /root/services/dataStaging read from config
- If config is empty or doesn’t exist, they watch() for it.
  - Receive notices if the config is created or gets populated

Rendezvous Problem

- Related to the runtime configuration problem
- Sometimes the whole configuration may not be known until all the pieces come up
- Create a node /root/dataStaging/rendezvous
- All the members of /root/dataStaging/services put a watch on the rendezvous node
- When the node is created and filled in, the watch is fired and everyone reads the information in the rendezvous node.

Configuration Changes

- Just like the previous example, except the configuration information changes.
  - For example, everyone in the dataStaging group should now start publishing to a different topic.

Locks

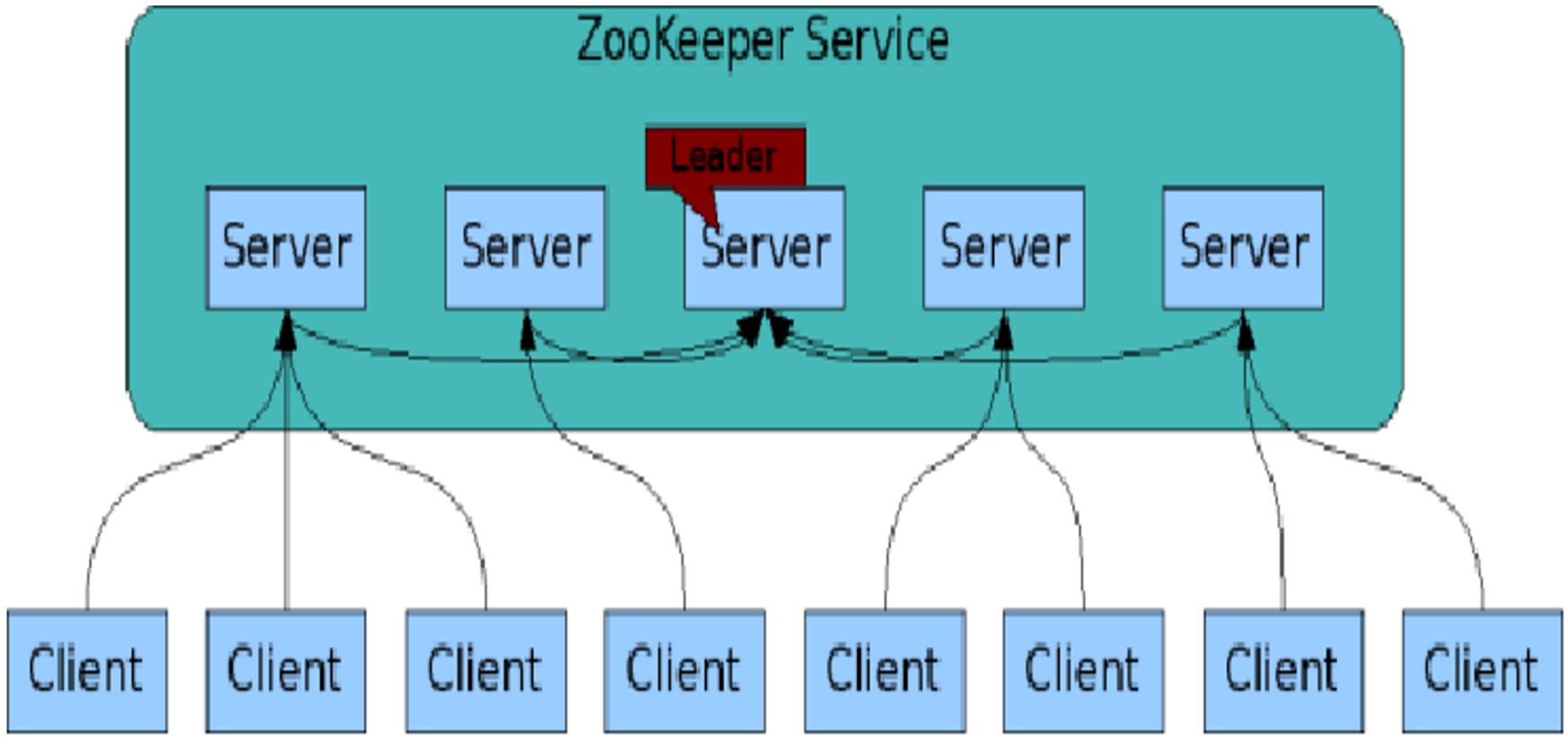
- Assume a client wants a lock on a resource
- Create a sequential, ephemeral node
  - /root/dataStaging/resources/lock_1
- If you want a lock but someone else has it, also put a watch on the lock entry preceding yours
  - Client #2 creates /root/dataStaging/resources/lock_2 and puts a watch on lock_1.
  - When lock_1 is released, Client #2 gets the lock

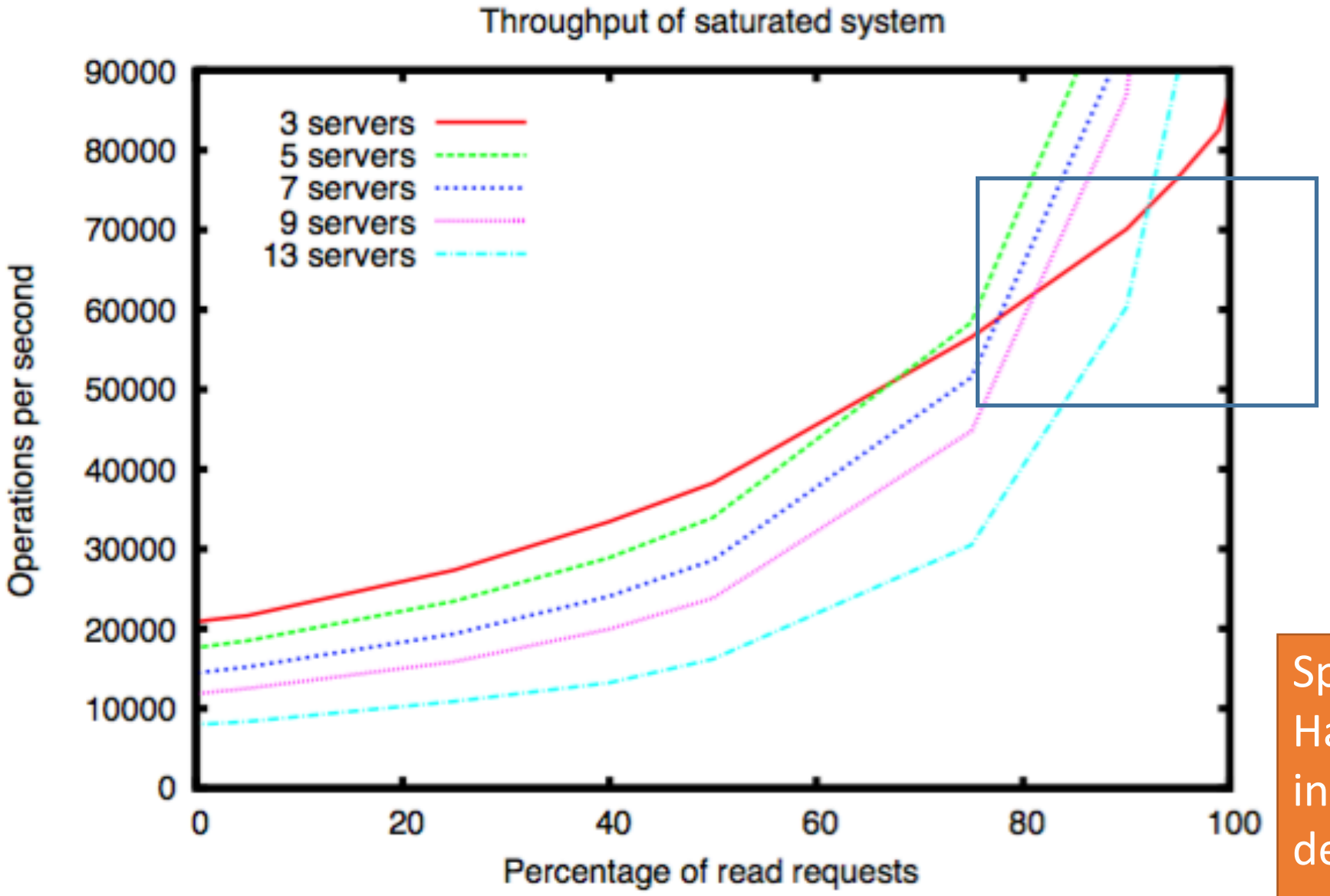
Leader Election

- We'll talk about this in detail in future lectures about RAFT
- You can use Zookeeper to manage leader elections
- Zookeeper itself manages leadership changes

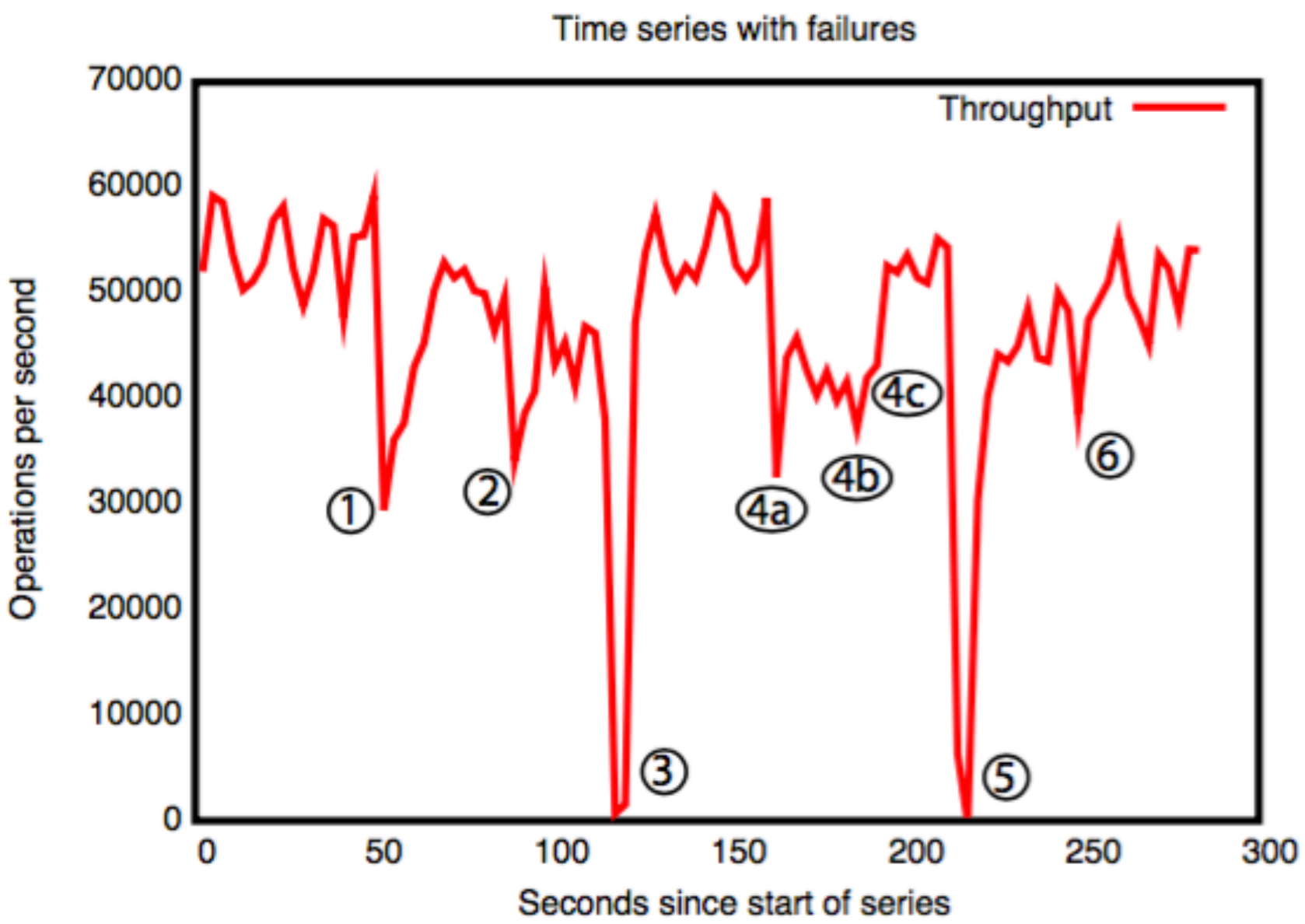
Reality Check

- Zookeeper's API is low level and can be hard to use.
- Apache Curator (http://curator.apache.org/) provides a higher level API and recipes for using Zookeeper.
  - Easier to use and less error prone
  - But Java only
- HashiCorp's Consul is also a popular choice
  - "Next generation" after Zookeeper
  - Enterprise considerations like security built in


ZooKeeper Service
Leader
Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client



Speed isn't everything. Having many servers increases reliability but decreases throughput as # of writes increases.



1. Failure and recovery of follower.
2. Failure and recovery of follower.
3. Failure of leader (200 ms to recover).
4. Failure of two followers (4a and 4b), recovery at 4c.
5. Failure of leader
6. Recovery of leader (?)

A cluster of 5 zookeeper instances responds to manually injected failures.

Final Thoughts and Cautions

- Zookeeper is powerful but it is only one possible solution.
- Messaging is also very powerful distributed computing concept
  - You could build a queuing system with Zookeeper, but you shouldn't
  - https://cwiki.apache.org/confluence/display/CURATOR/TN
  - There are high quality queuing systems already
- Highly available versus elastic, recoverable components
  - Zookeeper is better in the latter case
- Where is the state of your system? Make one choice. Don't have shadow states.

Zookeeper: Wait-Free Data Objects

Key design choice: wait-free data objects	• Locks are not a Zookeeper primitive • You can use Zookeeper to build lock-based systems
Resembles distributed file systems	• Smaller data
FIFO client ordering of messages	• Asynchronous messaging • Assumes you can order messages globally
Basic idea: idempotent state changes	• Components can figure out the state by looking at the change long • Operations incompatible with state throw exceptions
A good approach when systems can tolerate inconsistent data	• DNS for example • But not E-Commerce, which needs stronger guarantees

Coordination Examples in Distributed Systems

Configuration

Basic systems just need lists of operational parameters for the system processes: IP addresses of other members

Sophisticated systems have dynamic configuration parameters.

Group Membership, Leader Election

- Processes need to know which other processes are alive
- Which processes are definitive sources for data (leaders)?
- What happens if the leader is faulty?

Locks

- Implement mutually exclusive access to critical resources.

Zookeeper Caches and Watches

Zookeeper clients cache data	• Reads go to the cache
Watches: notify listening clients when cache has changed	• Watches don’t convey the content of the change. • Only work once; clients decide on the action
Why?	• Networks aren’t uniform or reliable (fallacy) • Centralized, top-down management doesn’t scale
Suitable for read-dominated systems	• If you can tolerate inconsistencies