Apache Zookeeper

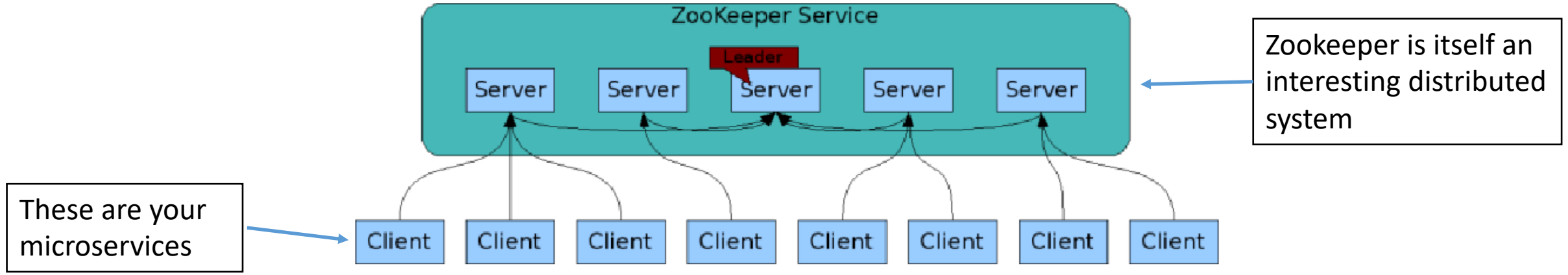
Hunt, P., Konar, M., Junqueira, F.P. and Reed, B., 2010, June. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In USENIX Annual Technical Conference (Vol. 8, p. 9).

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.

What Is Zookeeper?

- In short, Zookeeper provides a way for microservices to put and get small pieces of information that they need to share.
- Zookeeper allows clients to organize information into tree structures.
  - Like a file system, DNS, or LDAP
- Zookeeper lets you keep the information ordered across a distributed set of ZK servers.
  - Each message sent by a client is considered to be a state change.

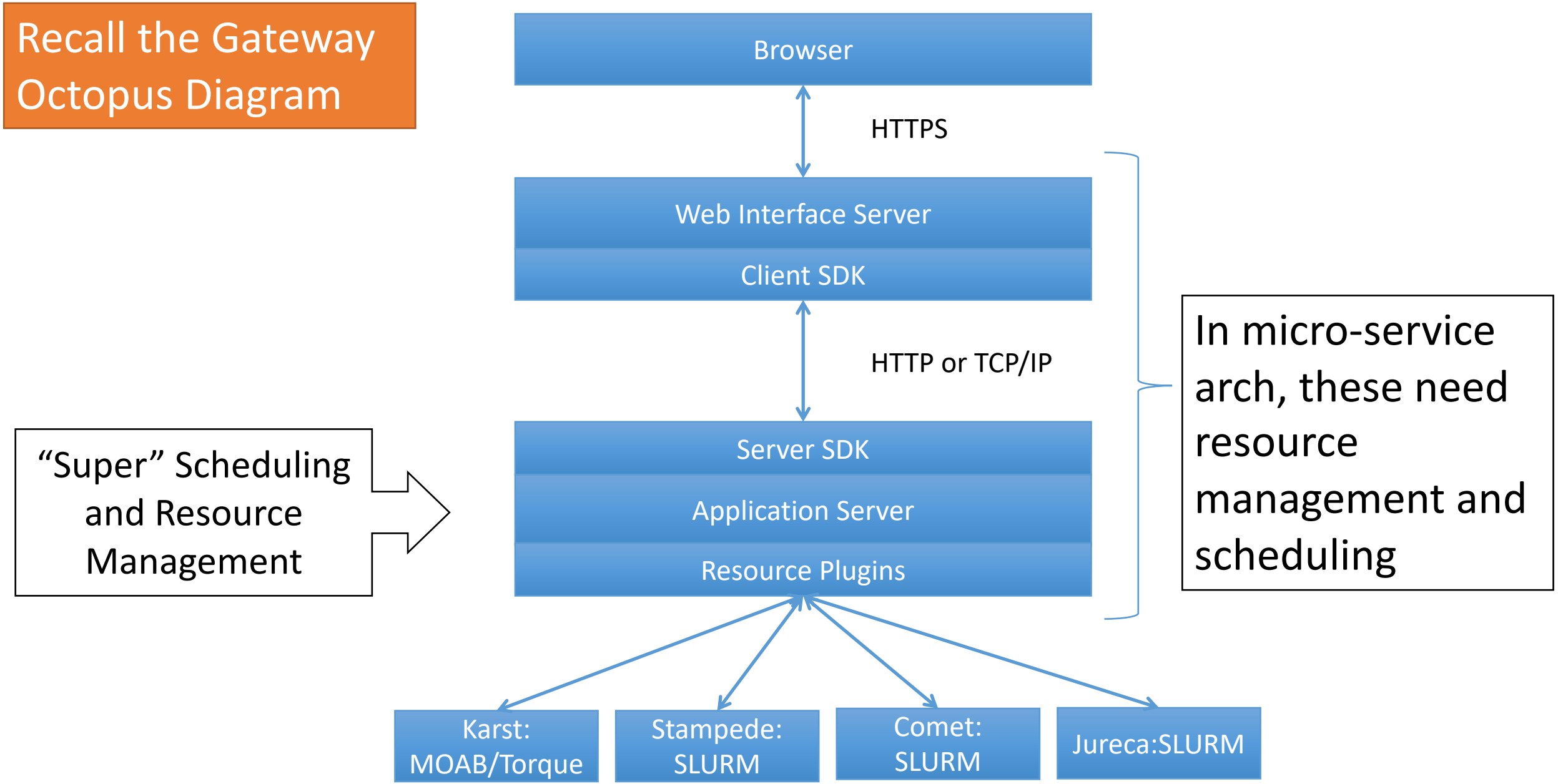
The ZooKeeper Service



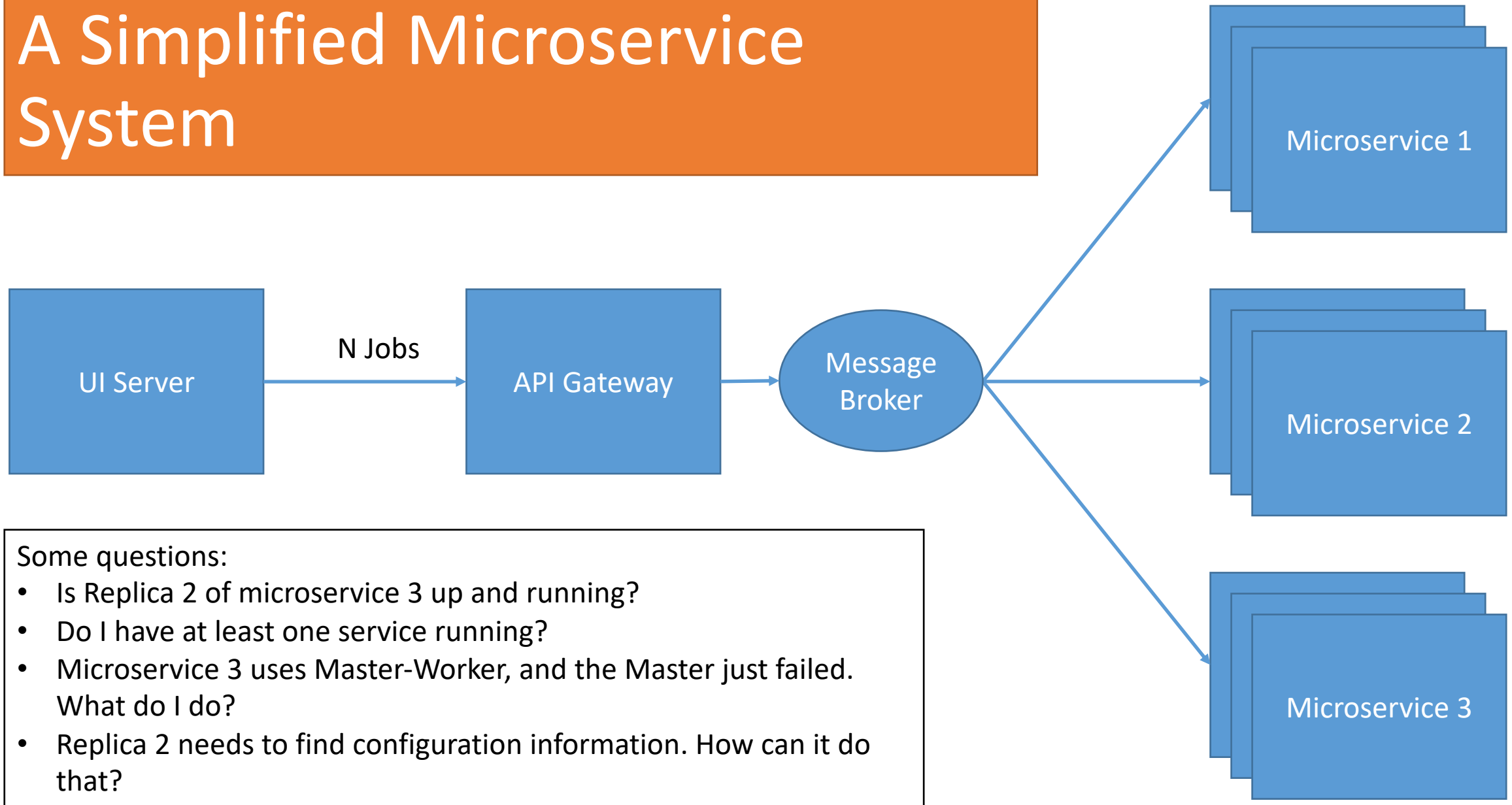
- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data in memory (!)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server & maintains a TCP connection.
- Client can read from any Zookeeper server.
- Writes go through the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

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

Recall the Gateway Octopus Diagram



A Simplified Microservice System



Some questions:

- Is Replica 2 of microservice 3 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 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.
- The health of the microservices.
- The state of a particular calculation.
- Group membership

Apache Zookeeper Is...

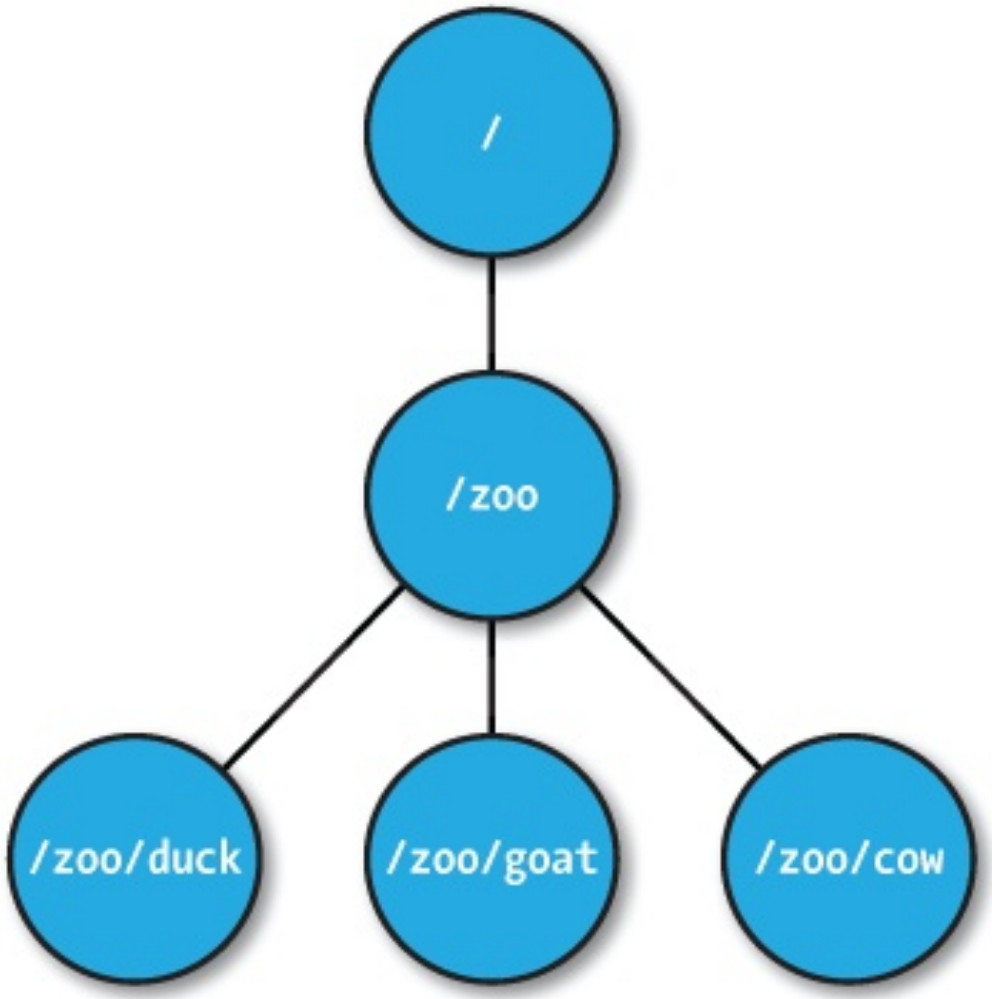
- A system for solving **distributed coordination problems** for multiple cooperating clients.
- A lot like a distributed file system...
  - As long as the files are tiny.
  - And you could get notified when the file changes
  - And the full file pathname is meaningful to applications
- A way to solve microservice management problems.
- An interesting implementation of a distributed system itself.
  - Look under the hood AND read the papers

Zookeeper, More Briefly

- Zookeeper Clients (that is, your applications) can create and discover nodes on ZK 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.
- You could build a small DNS with Zookeeper.
- Some simple analogies
  - Lock files and .pid files on Linux systems.

Let’s next look at ZK’s data structure and its API

ZNodes



- 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

Sequential

- Property of regular and ephemeral znodes
- Has a universal, monotonically increasing counter appended to the name

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.

What Can You Do with this Simple API?

Recall classic distributed systems: DNS, Git, NFS, etc.

Configuration Management

- All clients get their configuration information from a named znode
  - /root/config-me
- Example: you can build a public key store with Zookeeper
- Clients set watches to see if configurations change
- Zookeeper doesn’t explicitly decide which clients are allowed to update the configuration.
  - That would be an implementation choice
  - Zookeeper uses leader-follower model internally, so you could model your own implementation after this.

The Rendezvous Problem

- Classic distributed computing algorithm
- Consider master-worker
  - Specific configurations may not be known until runtime
  - EX: IP addresses, port numbers
  - Workers and master may start in any order
- Zookeeper implementation:
  - Create a rendezvous node: /root/rendezvous
  - Workers read /root/rendezvous and set a watch
    - If empty, use watch to detect when master posts its configuration information
  - Master fills in its configuration information (host, port)
  - Workers are notified of content change and get the configuration information

Locks

- Familiar analogy: lock files used by Apache HTTPD and MySQL processes
- Zookeeper example: who is the leader with primary copy of data?
- Implementation:
  - Leader creates an ephemeral file: /root/leader/lockfile
  - Other would-be leaders place watches on the lock file
  - If the leader client dies or doesn’t renew the lease, clients can attempt to create a replacement lock file
- Use SEQUENTIAL to solve the herd effect problem.
  - Create a sequence of ephemeral child nodes
  - Clients only watch the node immediately ahead of them in the sequence

Configuration Management

- Problem: gateway components in a distributed system need to get the correct configuration file.
- Solution: Components contact Zookeeper to get configuration metadata.
- Comments: this includes both the component's own configuration file as well as configurations for other components
  - Rendezvous problem

Service Discovery

- Problem: Component A needs to find instances of Component B
- Solution: Use Zookeeper to find available group members instances of Component B
  - More: get useful metadata about Component B instances like version, domain name, port #, flavor
- Comments
  - Useful for components that need to directly communicate but not for asynchronous communication (message queues)

Group Membership

- Problem: a job needs to go to a specific flavor of application manager. How can this be located?
- Solution: have application managers join the appropriate Zookeeper managed group when they come up.
- Comments: This is useful to support scheduling

System State for Distributed Systems

- Which servers are up and running? What versions?
- Services that run for long periods could use ZK to indicate if they are busy (or under heavy load) or not.
- Note overlap with our Registry
  - What state does the Registry manage? What state would be more appropriate for ZK?

Leader Election

- Problem: metadata servers are replicated for read access but only the master has write privileges. The master crashes.
- Solution: Use Zookeeper to elect a new metadata server leader.
- Comment: this is not necessarily the best way to do this