Project Midterms: March 22nd: No Extensions

- Team Presentations
- 10 minute presentations by each team member
- Demo of Gateway
- System Design
  - What choices did you make for state management, data storage, API implementation, metadata modelings, etc
- Team recommendations
  - How can the team work more effectively?
  - How can the class be more effective?

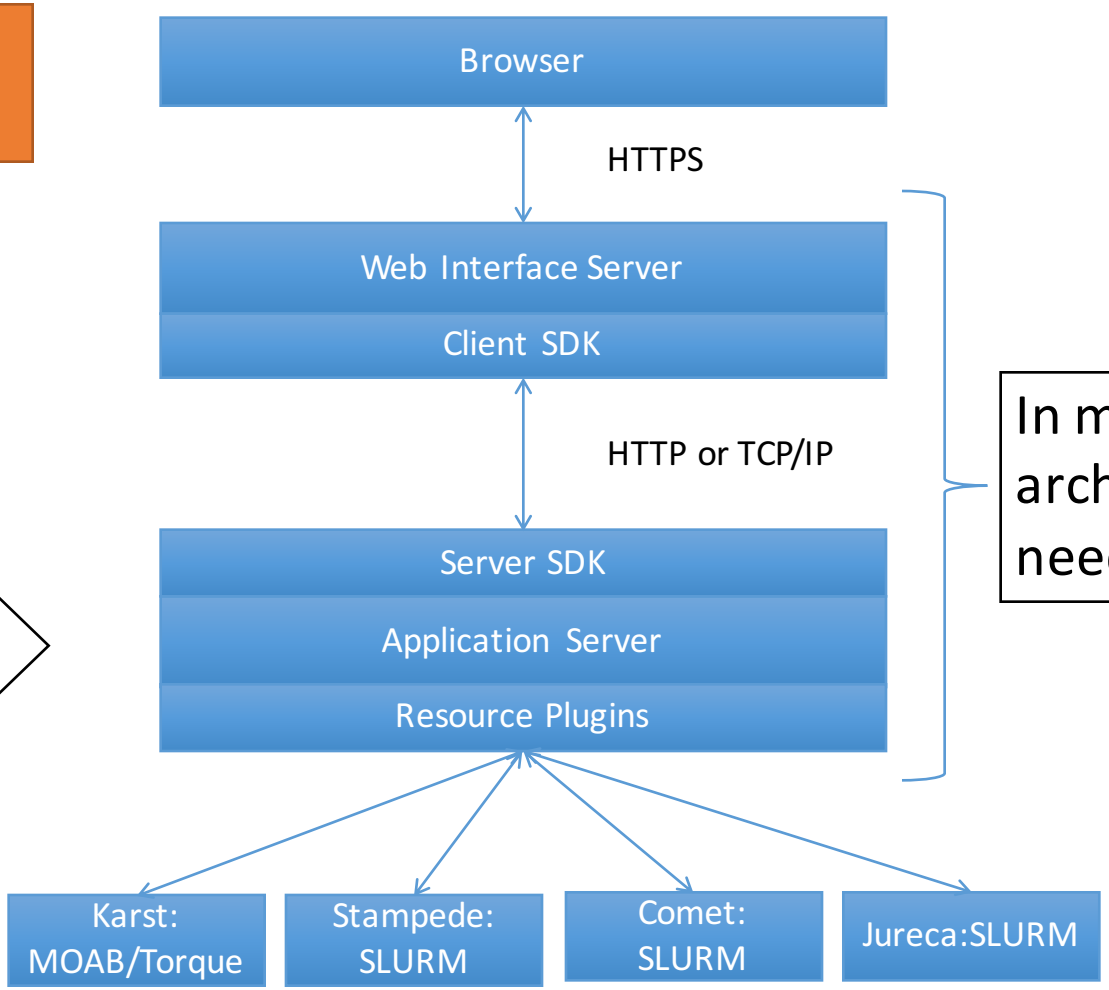
Introduction to Distributed Systems

Overview of Key Concepts

Some Definitions

Distributed Systems	• Multiple software components that work together as a single composite entity.
Distributed Computing	• Distributed components work together to perform large computations by breaking them into smaller parts. • Examples: HTCondor, Apache Hadoop, Apache Spark, MPI, etc
Parallel Computing	• A special case of distributed computing • Typically on a specialized resource (Big Red II, Blue Waters, etc) • Can assume tighter coupling
Science Gateways	• At scale are distributed systems. • Theme for this lecture

Recall the Gateway Octopus Diagram



"Super" Scheduling and Resource Management

In micro-service arch, these also need scheduling

Some General Advice

- Get a book on distributed systems and read it.
  - Smart people have thought about these problems for a long time.
  - Jim Gray invented two-phase commit c. 1978, for example
  - CS problems often map to non-CS analogies
- There many algorithms and design patterns for most problems that you will face.
- The challenge: knowing the right prior solution for the problem at hand
- The anti-pattern: reinventing something in a text book out of ignorance

Classic Problems in Distributed Systems

- Global Naming
- Identity Management, Authentication, Authorization
- Distributed Transactions
- Distributed State Machines
- Leader Election
- Messaging, publish-subscribe, notification

Classic Distributed Systems Examples

Git	• Versioning
Distributed File Systems	• Write, not just read at scale • Global identities, groups,
Replicated Databases	• Primary copies, leaders, transactions
Resource Managers	• Torque, SLURM, etc • Fault tolerance, process coordination

More Examples

Domain Name Servers	• Naming, caching • Eventual consistency
REST Systems	• Idempotent state • Error messages
Queuing Systems	• Message order, replay, guaranteed delivery
Domain Name Servers	• Scaling power of hierarchies and name spaces

Apache Zookeeper as a Case Study

Apache Zookeeper	• Illustrates several distributed systems concepts • Some interesting design choices and assumptions
Illustrative Only	• Lectures are not an endorsement
Many Others to Study	• Apache Storm, Kafka, Spark, Hadoop, etc
Interesting Times	• Open source software that is small enough to understand and experiment with • But galactically scalable, used to run many companies

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).

Apache Zookeeper Is…

- A distributed system for solving distributed coordination problems.
  - It looks 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 distributed system itself.

Zookeeper summary

- ZooKeeper provides a simple and high performance kernel for building more complex coordination primitives.
  - Helps distributed components coordinate
- Clients contact zookeeper services to read and write metadata.
  - Read from cache but writes are more complicated
- Tree model for data.
  - Node names may be all you need
  - Lightly structured metadata stored in the nodes.
- **Wait-free** aspects of shared registers with an **event-driven** mechanism similar to cache invalidations of distributed file systems
- Targets simple metadata systems that read more than they write.
  - Small total storage


Engineering
Blog
Engineering life
Open source
Tech talks
Find roles
https://engineering.pinterest.com/blog/building-follower-model-scratch
Pinterest engineering blog
BACK TO ALL POSTS
JUL 12, 2013

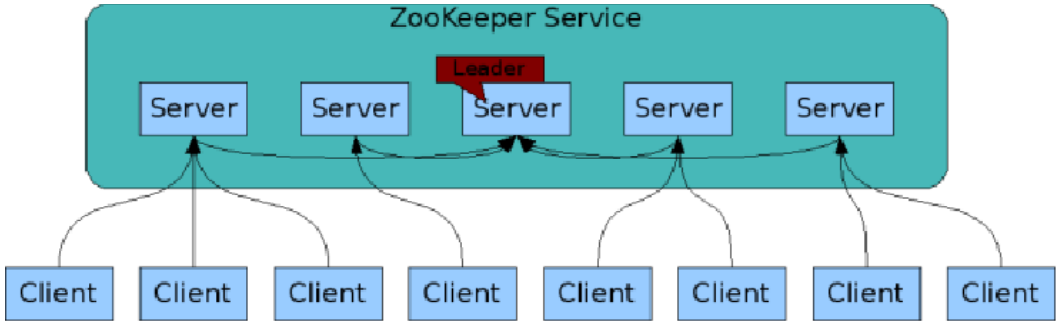
Coordination Examples in Distributed Systems

Configuration	• Basic systems just need lists of operational parameters for the system processes • 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?
Locks	• Implement mutually exclusive access to critical resources.

Zookeeper Design

- Doesn’t prescribe specific coordination primitives.
- Lower level than that.
- Allow clients to create their own coordination applications

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 & needs majority consensus.

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

Zookeeper: Wait-Free Data Objects

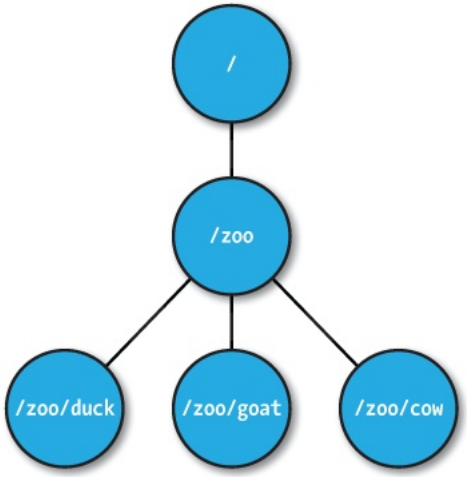
Key design choice: wait-free data objects	• Locks are not a Zookeeper primitive • You can use Zookeeper to build lock-based systems
Resembles distributed file systems	• No transactional guarantees
FIFO client ordering of messages	• Asynchronous messaging • Assumes you can order messages globally
Basic 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 errors	• DNS for example • But not E-Commerce, which needs stronger guarantees

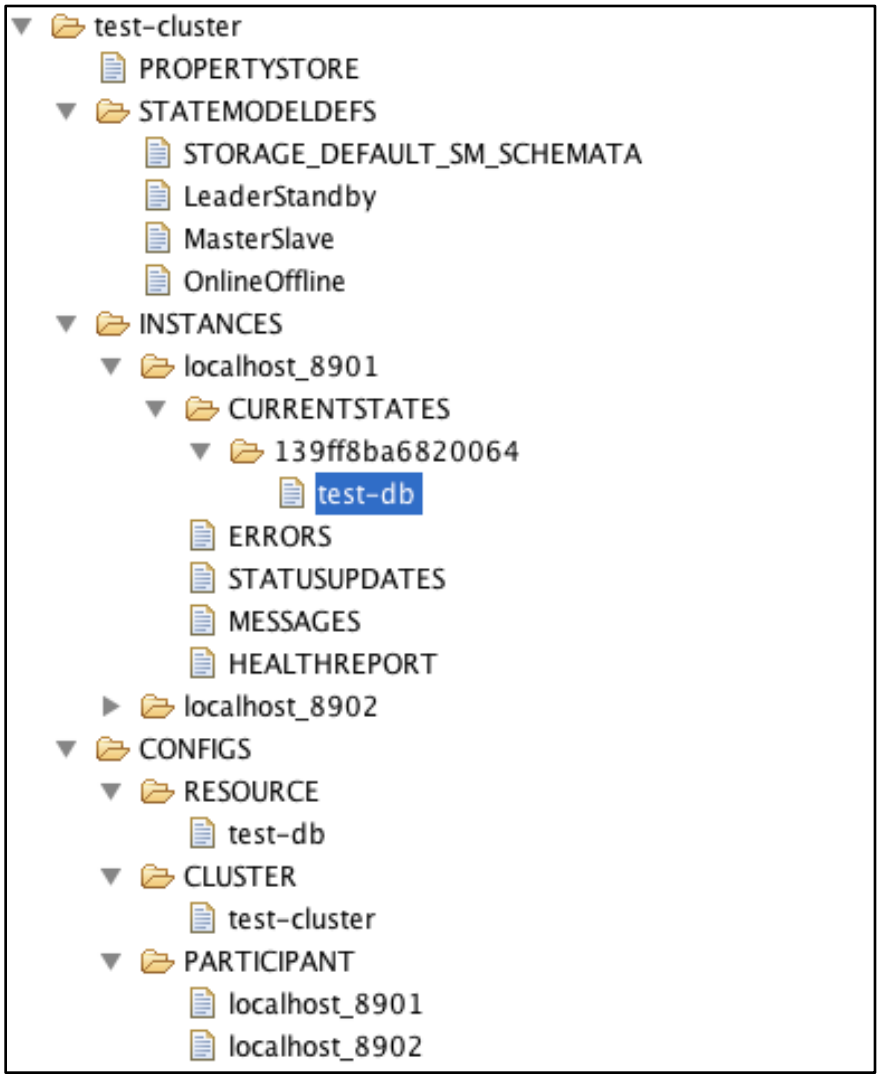
Zookeeper Caches and Watches

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

ZNodes

- Maintain a stat structure with version numbers for data changes, ACL changes and timestamps.
- Version numbers increases with changes
- Data is read and written in its entirety





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?

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.

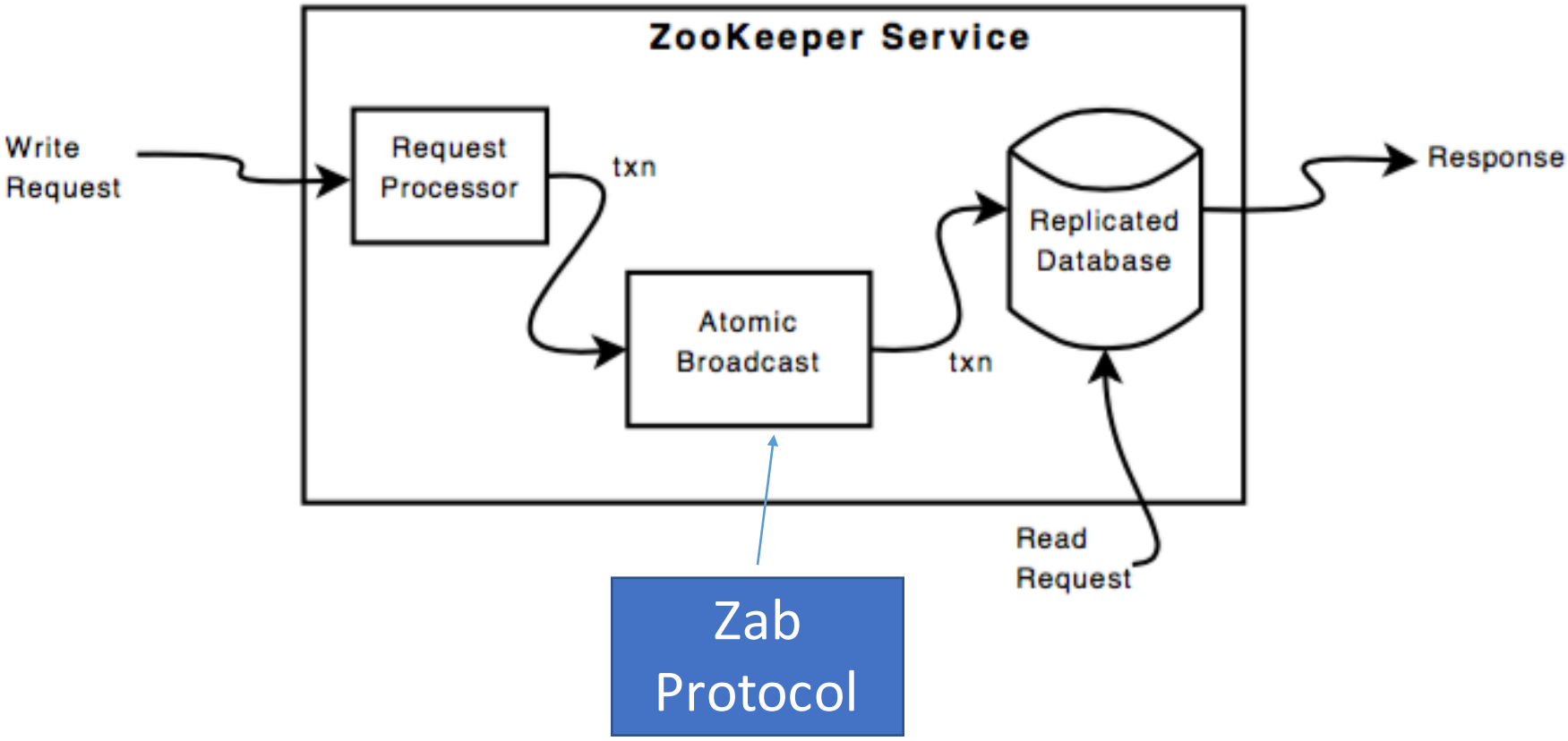
The Rendezvous Problem

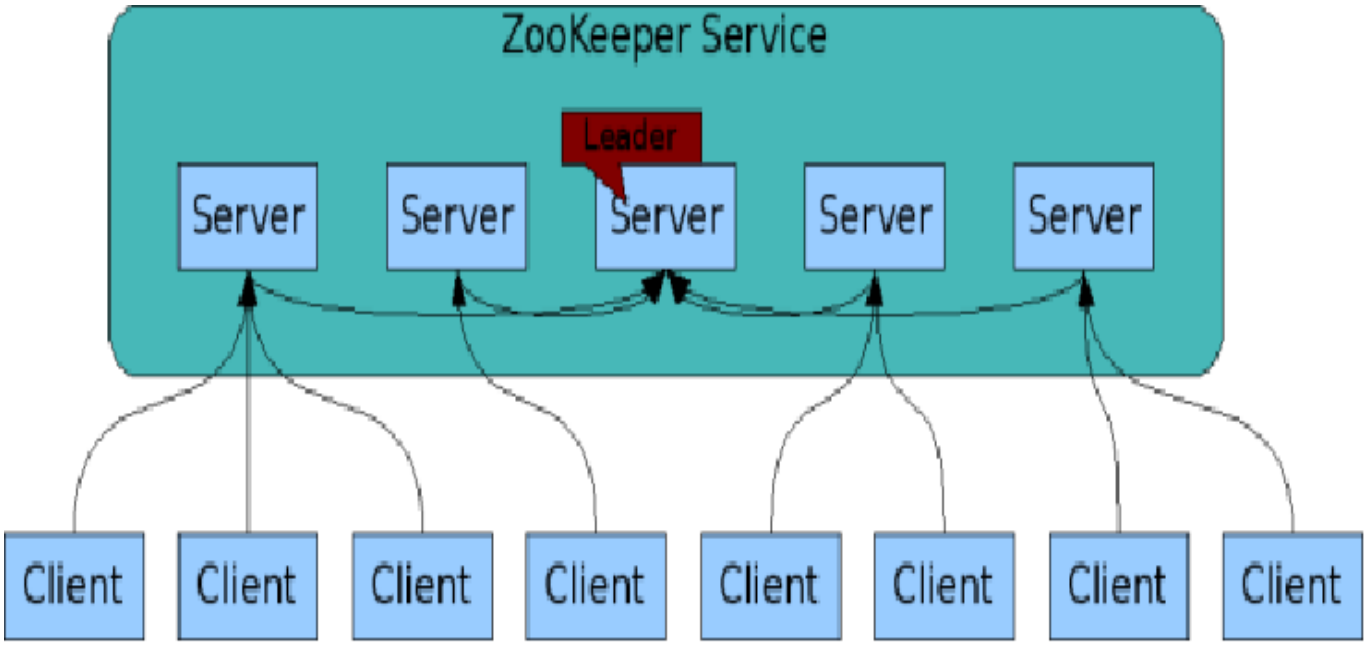
- Classic distributed computing algorithm
- Consider master-worker
  - Specific configurations may not be known until runtime
  - EX: IP addresses, port numbers
  - Workers and master may start in any order
- Zookeeper implementation:
  - Create a rendezvous node: /root/rendezvous
  - Workers read /root/rendezvous 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

Under the Zookeeper Hood


ZooKeeper Service
Write Request
Request Processor
txn
Atomic Broadcast
txn
Replicated Database
Response
Read Request
Zab Protocol


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

Zookeeper Handling of Writes

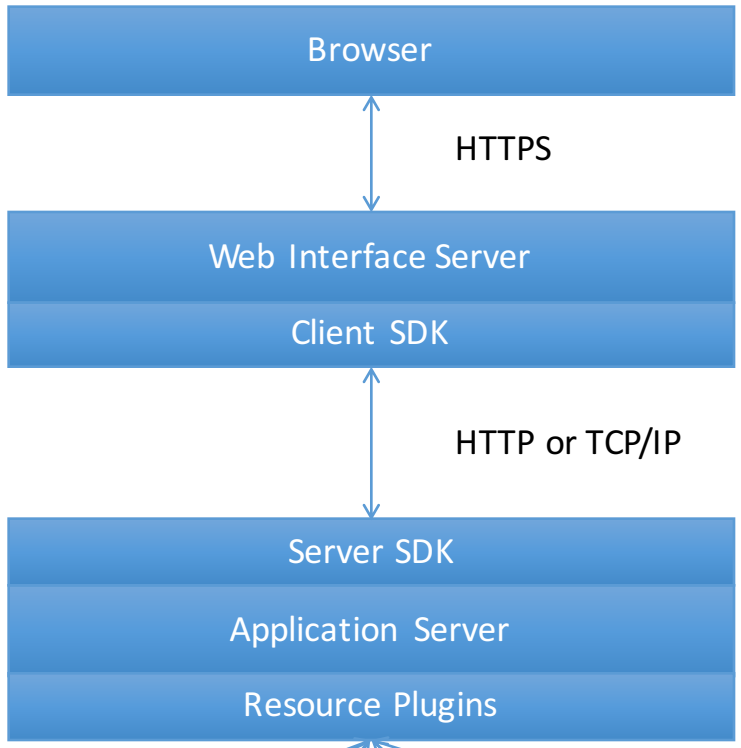
- READ requests are served by any Zookeeper server
  - Scales linearly, although information can be stale
- WRITE requests change state so are handed differently
- One Zookeeper server acts as the leader
- The leader executes all write requests forwarded by followers
- The leader then broadcasts the changes
- The update is successful if a majority of Zookeeper servers have correct state at the end of the process
- Zab protocol

Some Zookeeper Implementation Simplifications

- Uses TCP for its transport layer.
  - Message order is maintained by the network
  - The network is reliable?
- Assumes reliable file system
  - Logging and DB checkpointing
- Does write-ahead logging
  - Requests are first written to the log
  - The ZK DB is updated from the log
- ZK servers can acquire correct state by reading the logs from the file system
  - Checkpoint reading means you don't have to reread the entire history
- Assumes a single administrator so no deep security

Zookeeper and Science Gateways

Recall the Gateway Octopus Diagram



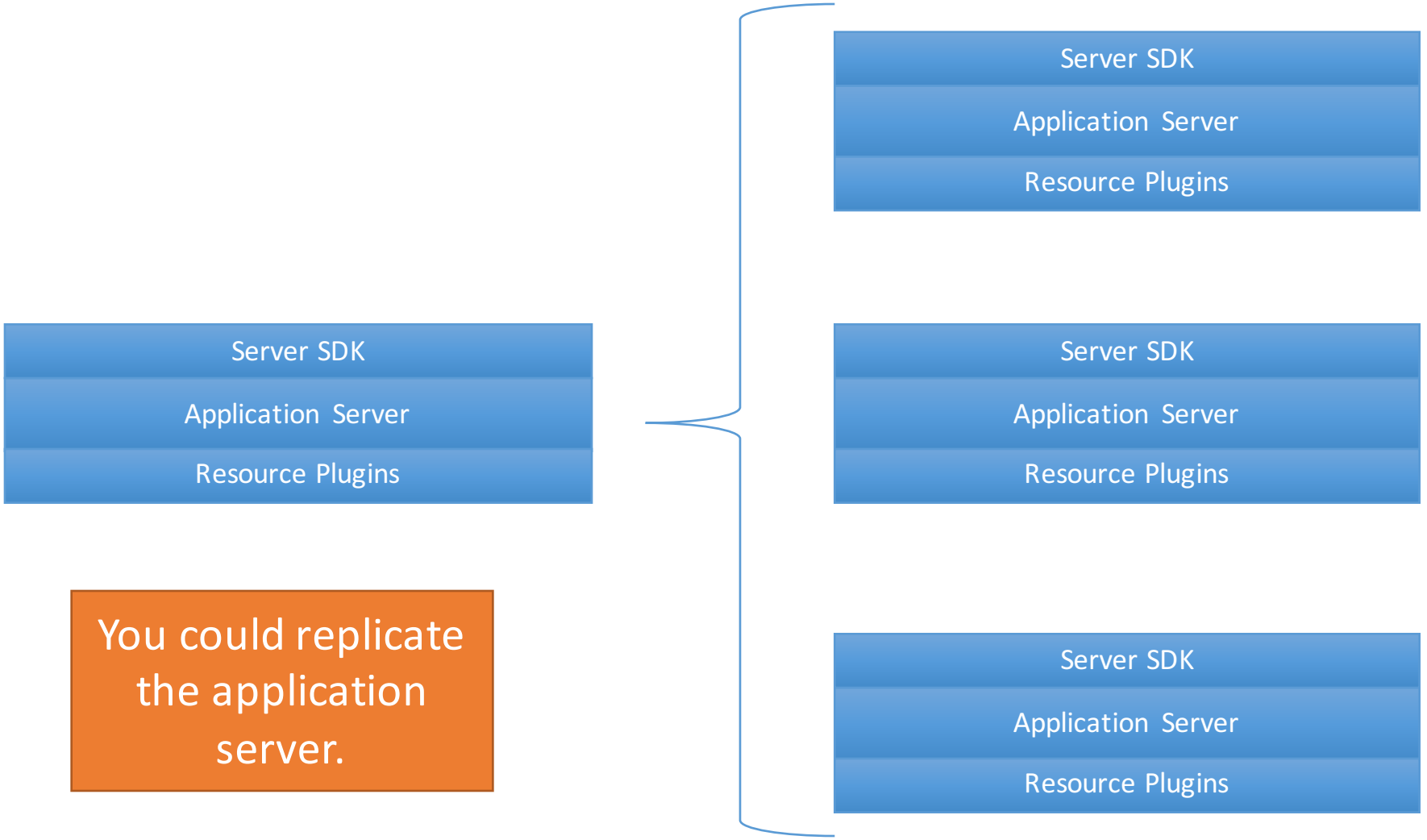
In micro-service arch, these also need scheduling

"Super" Scheduling and Resource Management

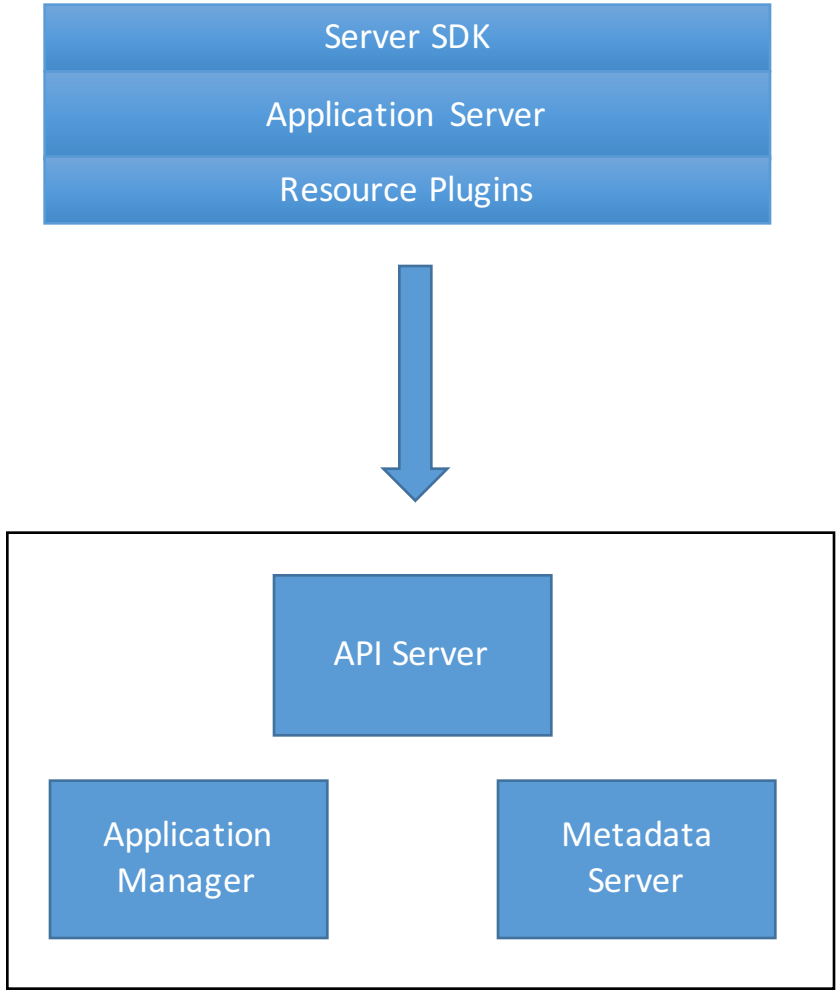
Different archs, schedulers, admin domains, …

How Can Gateways Use Zookeeper?

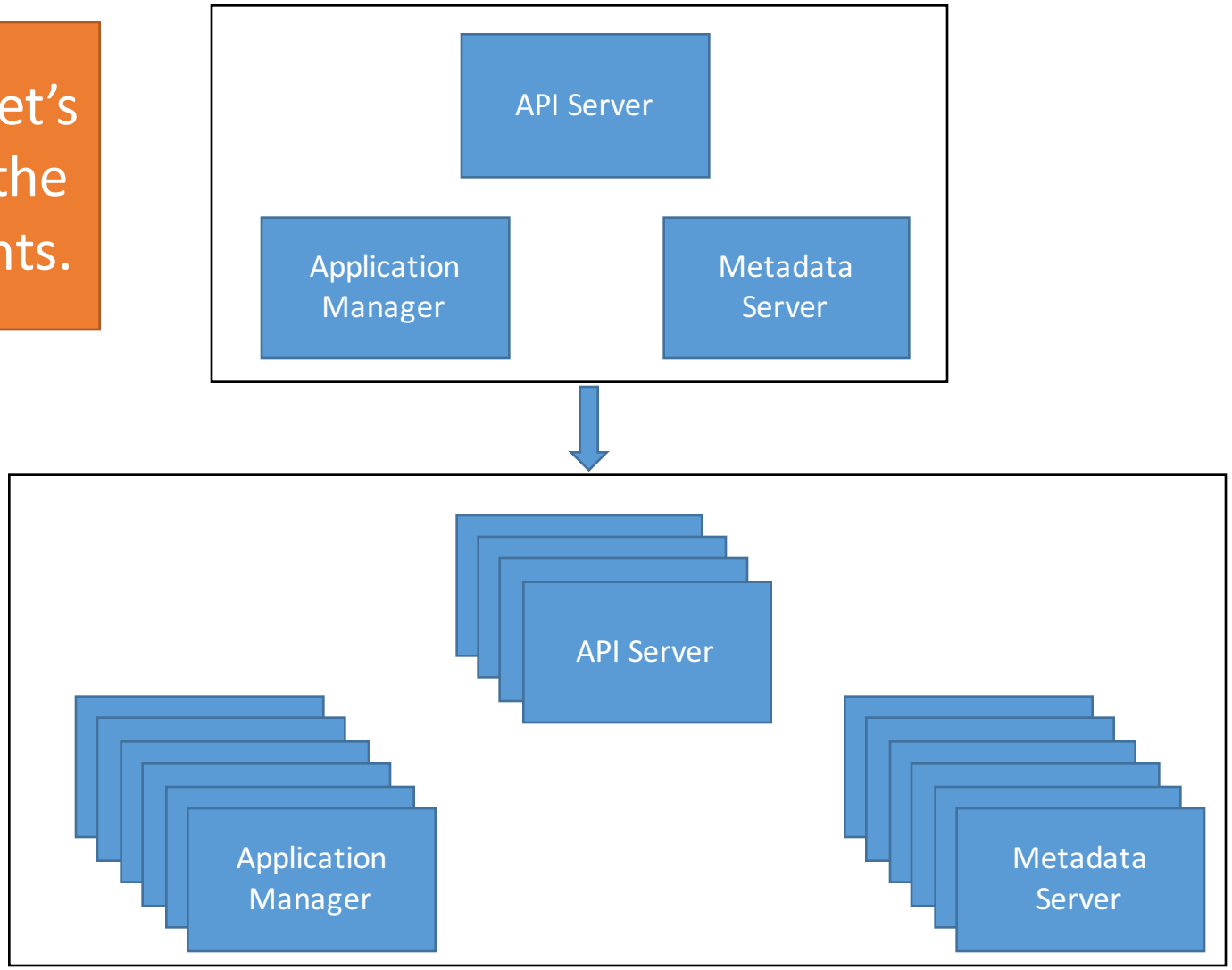
You tell me


Server SDK
Application Server
Resource Plugins
Server SDK
Application Server
Resource Plugins
Server SDK
Application Server
Resource Plugins
Server SDK
Application Server
Resource Plugins
You could replicate the application server.

But let's think first of decoupling the application server into smaller components.



And now let's replicate the components.



Why Do This?

- Fault tolerance
- Increased throughput, load balancing
- Component versions
  - Not all components of the same type need to be on the same version
  - Backward compatibility checking
- Component flavors
  - Application managers can serve different types of resources
  - Useful to separate them into separate processes if libraries conflict.

How Can Gateways Use Zookeeper?

I'll ask again

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
  - See Pinterest blog earlier for a failover example
  - 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.

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

Final Thoughts and Cautions

- Zookeeper is powerful but it is only one possible solution.
- A message queue 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.

The Fallacies of Distributed Computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.