ZooKeeper

A highly available, scalable, distributed, configuration, consensus, group membership, leader election, naming, and coordination service

Observations

1) Distributed systems always need some form of coordination
2) Programmers cannot use locks correctly
   - distributed deadlocks are the worst!
3) Group messaging can be hard to use in some applications

What “works”

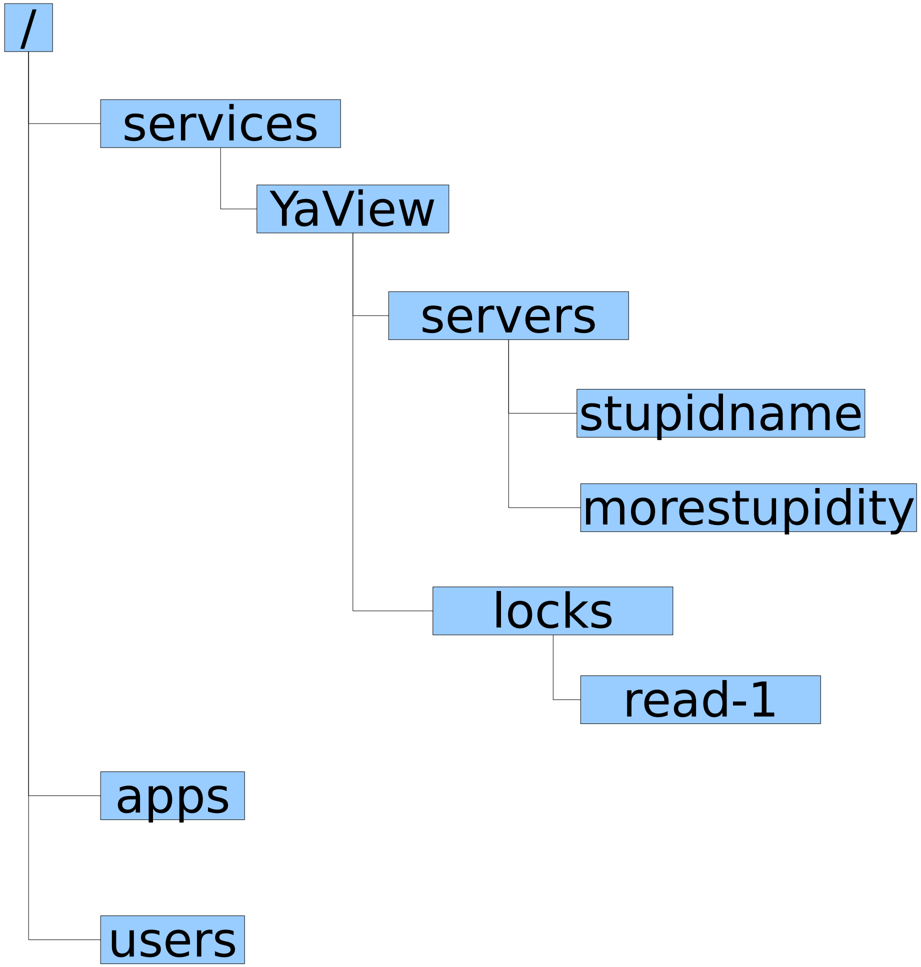
1) Programmers use shared file systems
   - Programmers are comfortable with file API
   - file servers are generic infrastructure components
   - It mostly works
2) File API and servers lack some needed semantics
   - Reasonable handling of concurrent writes
   - Change notifications

Making things really work

1) Conditional updates (to deal with concurrent clients)
2) Ordered updates and strong persistence guarantees
3) Watches for data changes
4) Ephemeral nodes
5) Generated file names

Data Model

1) Hierarchal namespace (like a file system)
2) Each znode has data and children
3) data is read and written in its entirety



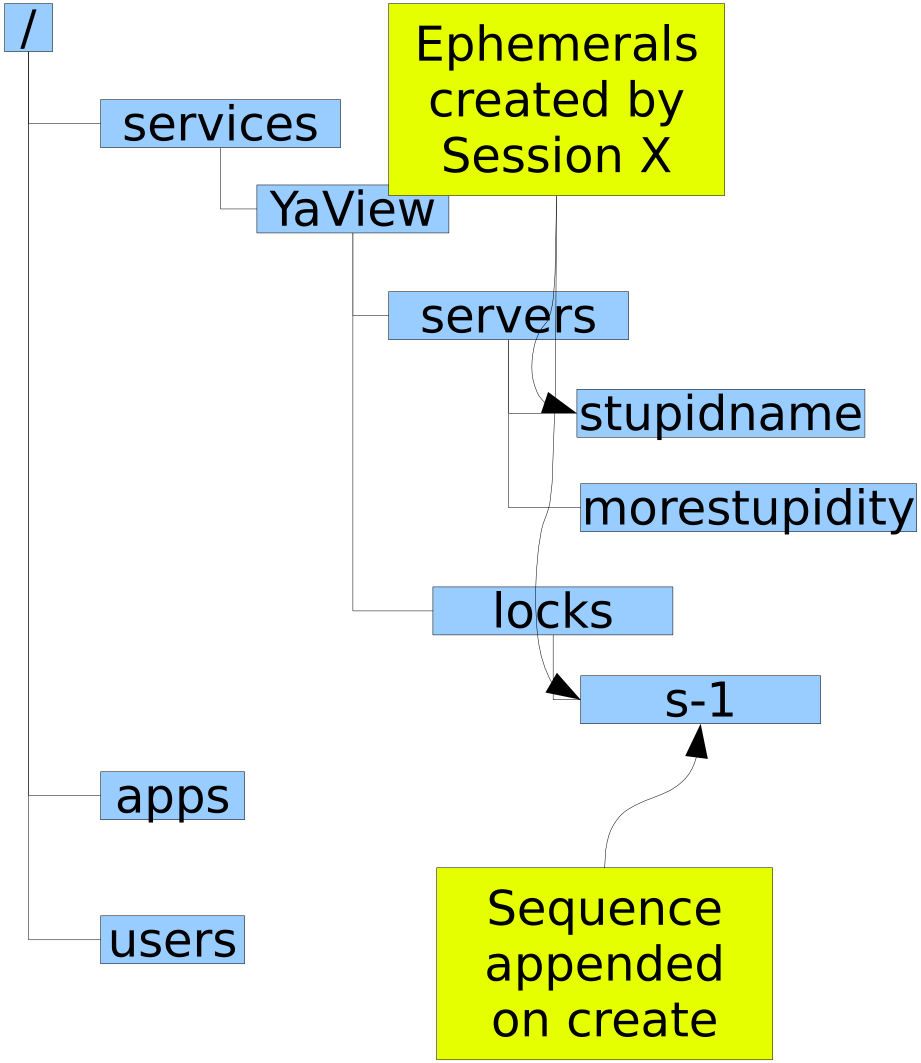
ZooKeeper API

```
String create(path, data, acl, flags)

void delete(path, expectedVersion)

Stat setData(path, data, expectedVersion)

(data, Stat) getData(path, watch)

Stat exists(path, watch)

String[] getChildren(path, watch)

void sync(path)


Stat setACL(path, acl, expectedVersion)

(acl, Stat) getACL(path)
```

Create Flags

1) Ephemeral: the znode will be deleted when the session that created it times out or it is explicitly deleted

2) Sequence: the the path name will have a monotonically increasing counter relative to the parent appended



ZooKeeper Guarantees

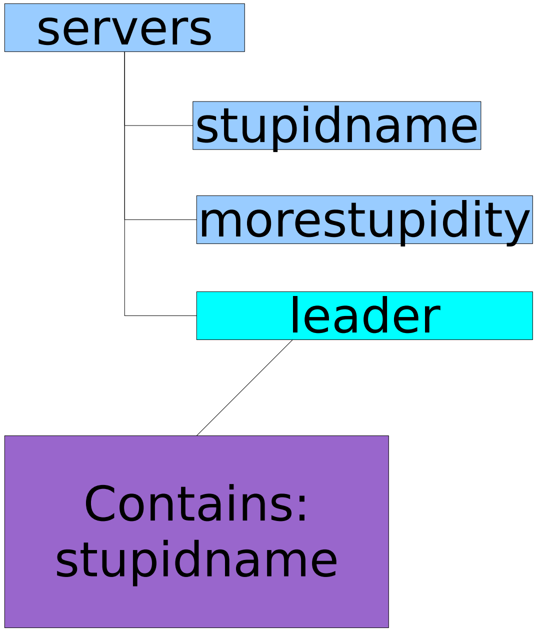
1) Clients will never detect old data.
2) Clients will get notified of a change to data they are watching within a bounded period of time.
3) All requests from a client will be processed in order.
4) All results received by a client will be consistent with results received by all other clients.

... is used to simplify examples, real usage requires full path names

Leader Election

1) getData(“.../servers/leader”, true)
2) if successful follow the leader described in the data and exit
3) create(“.../servers/leader”, hostname, EPHEMERAL)
4) if successful lead and exit
5) goto step 1

If a watch is triggered for “.../servers/leader”, followers will restart the leader election process



servers

stupidname

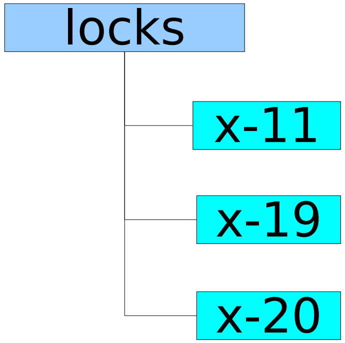
morestupidity

leader

Contains: stupidname

Locks

1) id = create("…/locks/x-", SEQUENCE|EPHEMERAL)
2) getChildren("…/locks"/, false)
3) if id is the 1st child, exit
4) exists(name of last child before id, true)
5) if does not exist, goto 2)
6) wait for event
7) goto 2)



Each znode watches one other.
No herd effect.

Shared Locks

1)id = create("…/locks/s-", SEQUENCE|EPHEMERAL)

2)getChildren("…/locks"/, false)

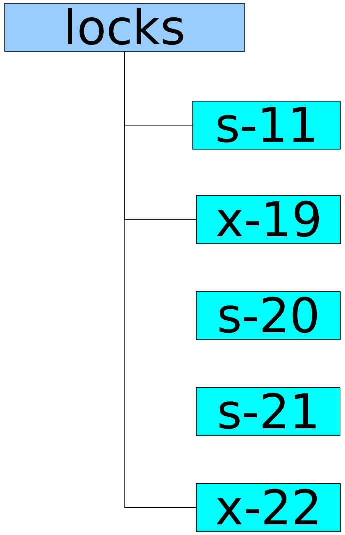
3)if no children that start with x- before id, exit

4)exists(name of the last x- before id, true)

5)if does not exist, goto 2)

6)wait for event

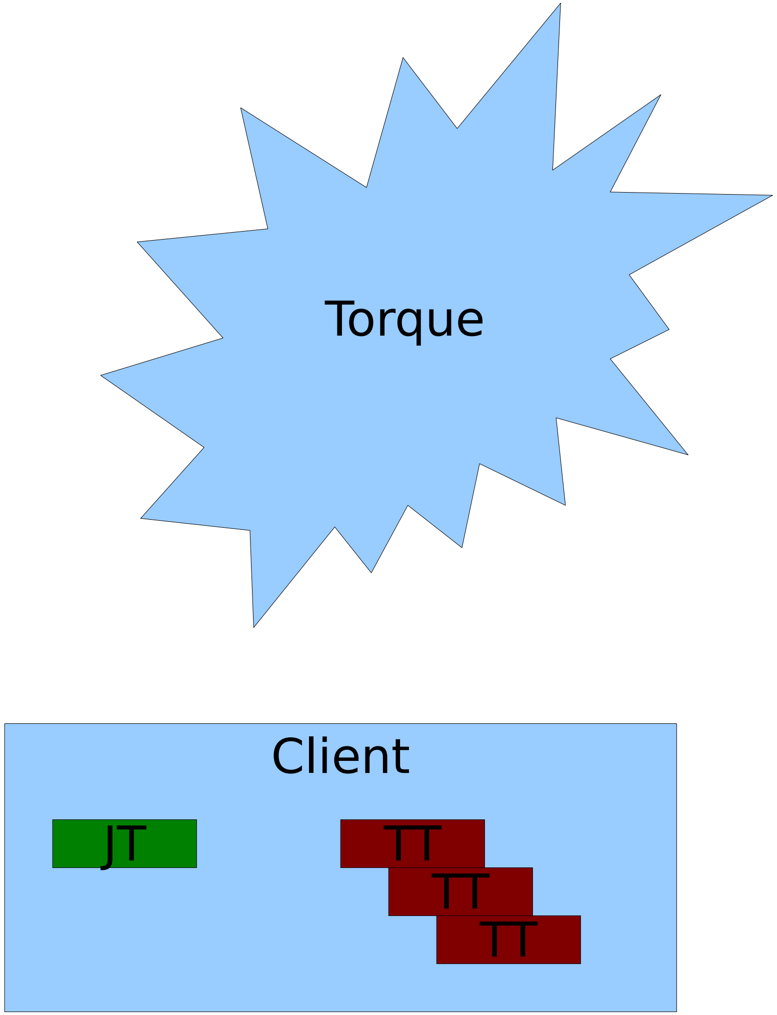
7)goto 2)



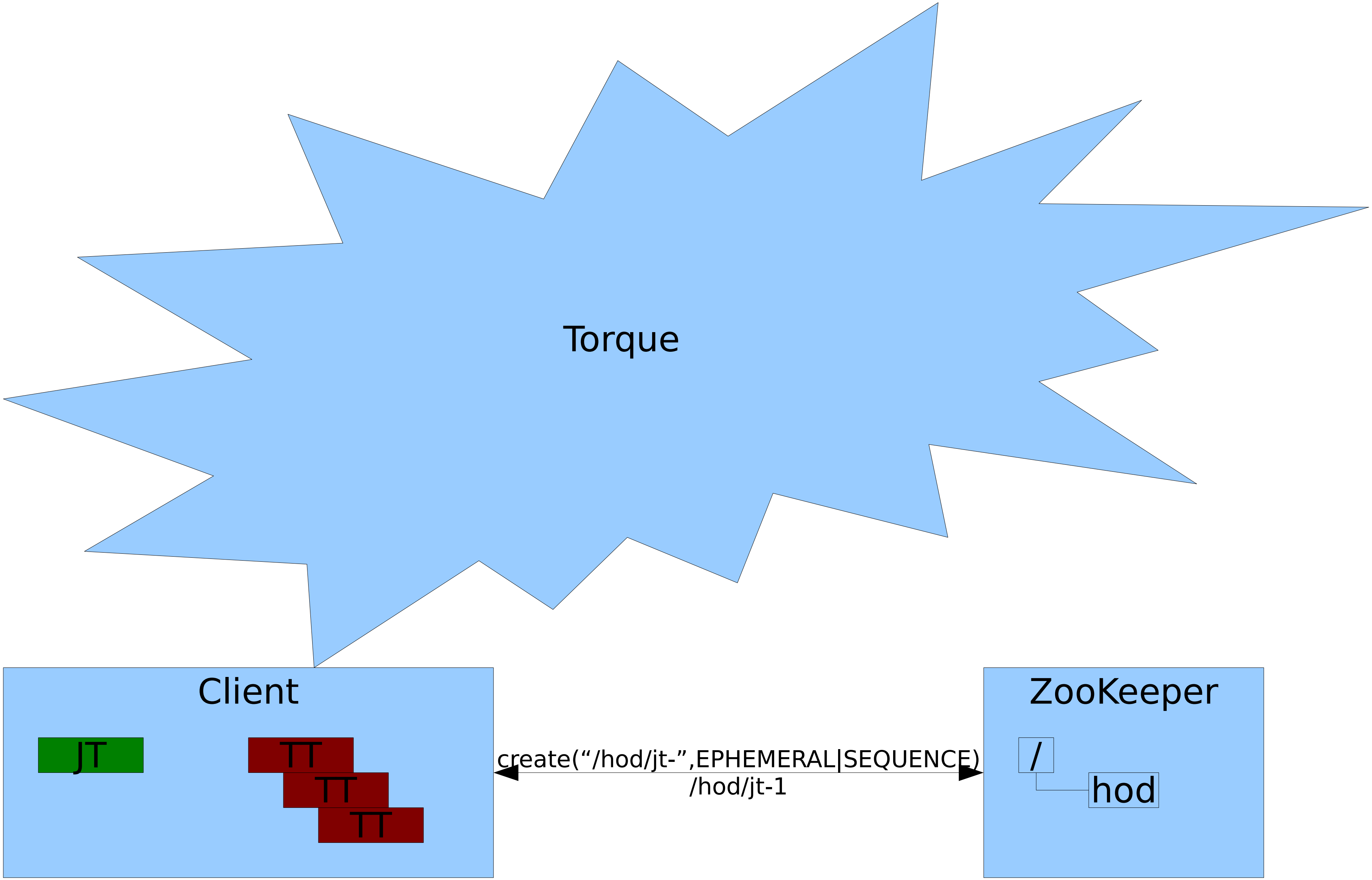
Each znode watches one other.
No herd effect.

HOD

1) A client submits a request to start jobtracker and a set of tasktrackers to torque
2) The ip address and the ports that the jobtracker will bind to is not known apriori
3) The tasktrackers need to find the jobtracker
4) The client needs to find the jobtracker



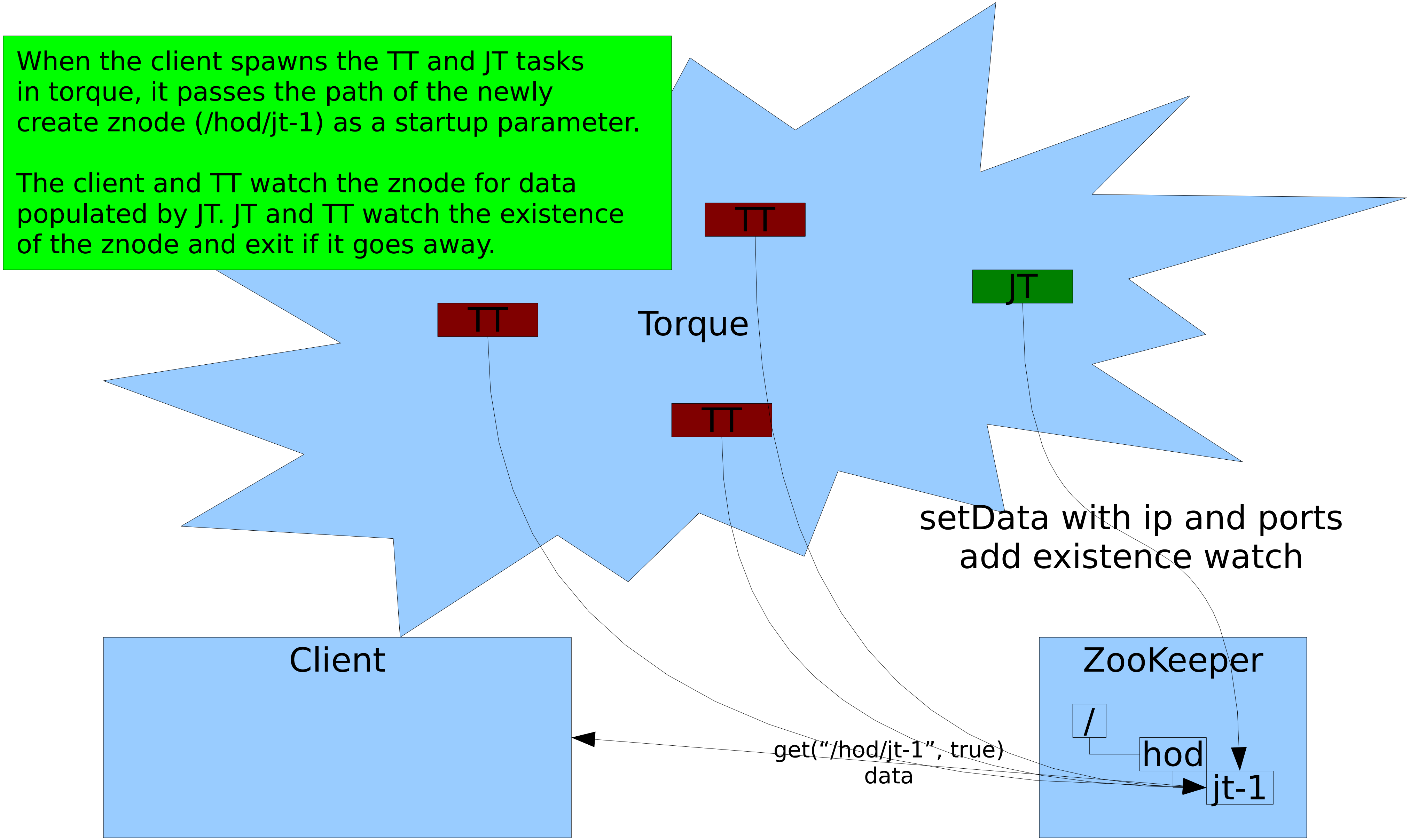
HOD with ZooKeeper



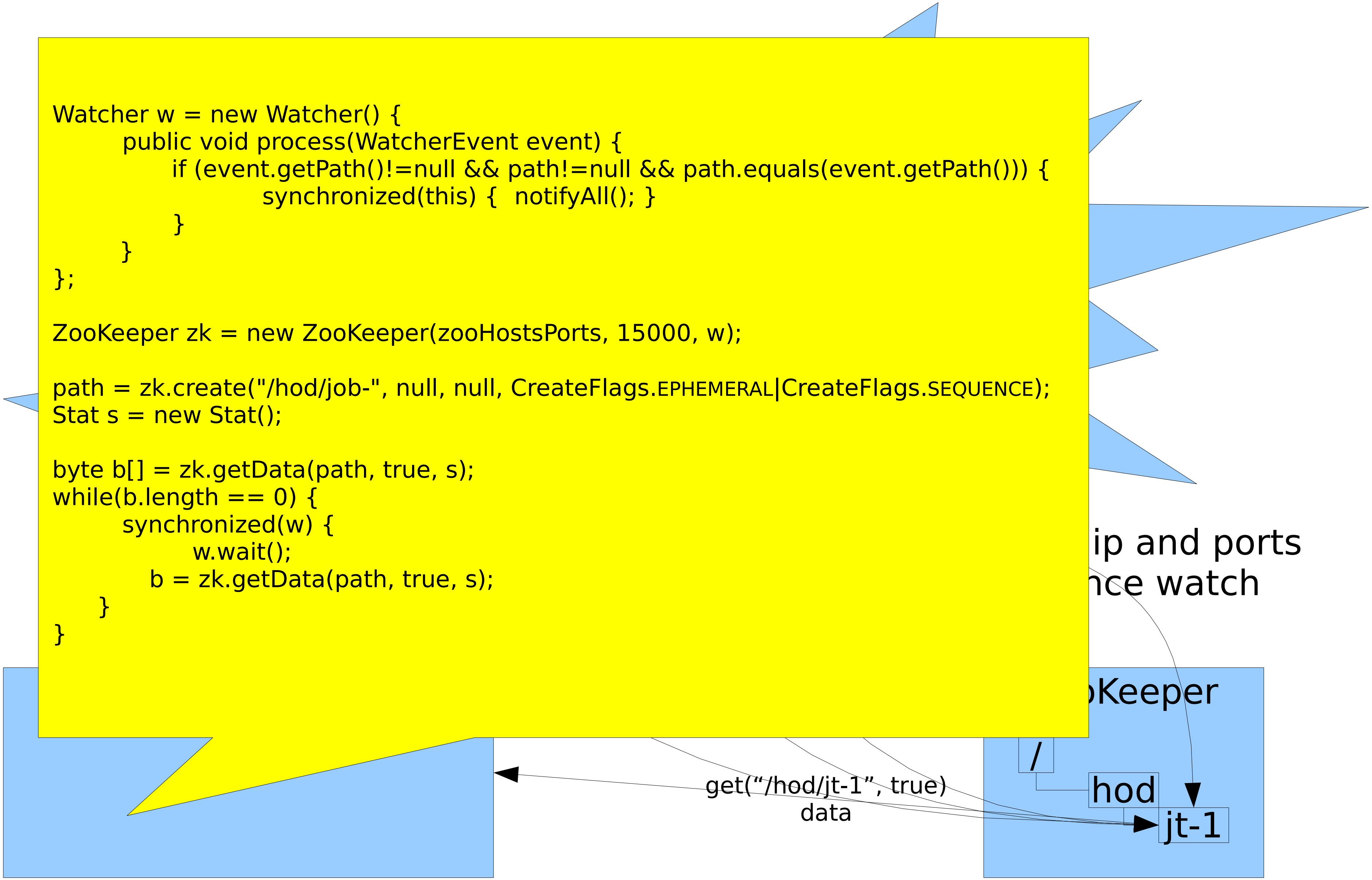
HOD with ZooKeeper

When the client spawns the TT and JT tasks in torque, it passes the path of the newly create znode (/hod/jt-1) as a startup parameter.

The client and TT watch the znode for data populated by JT. JT and TT watch the existence of the znode and exit if it goes away.

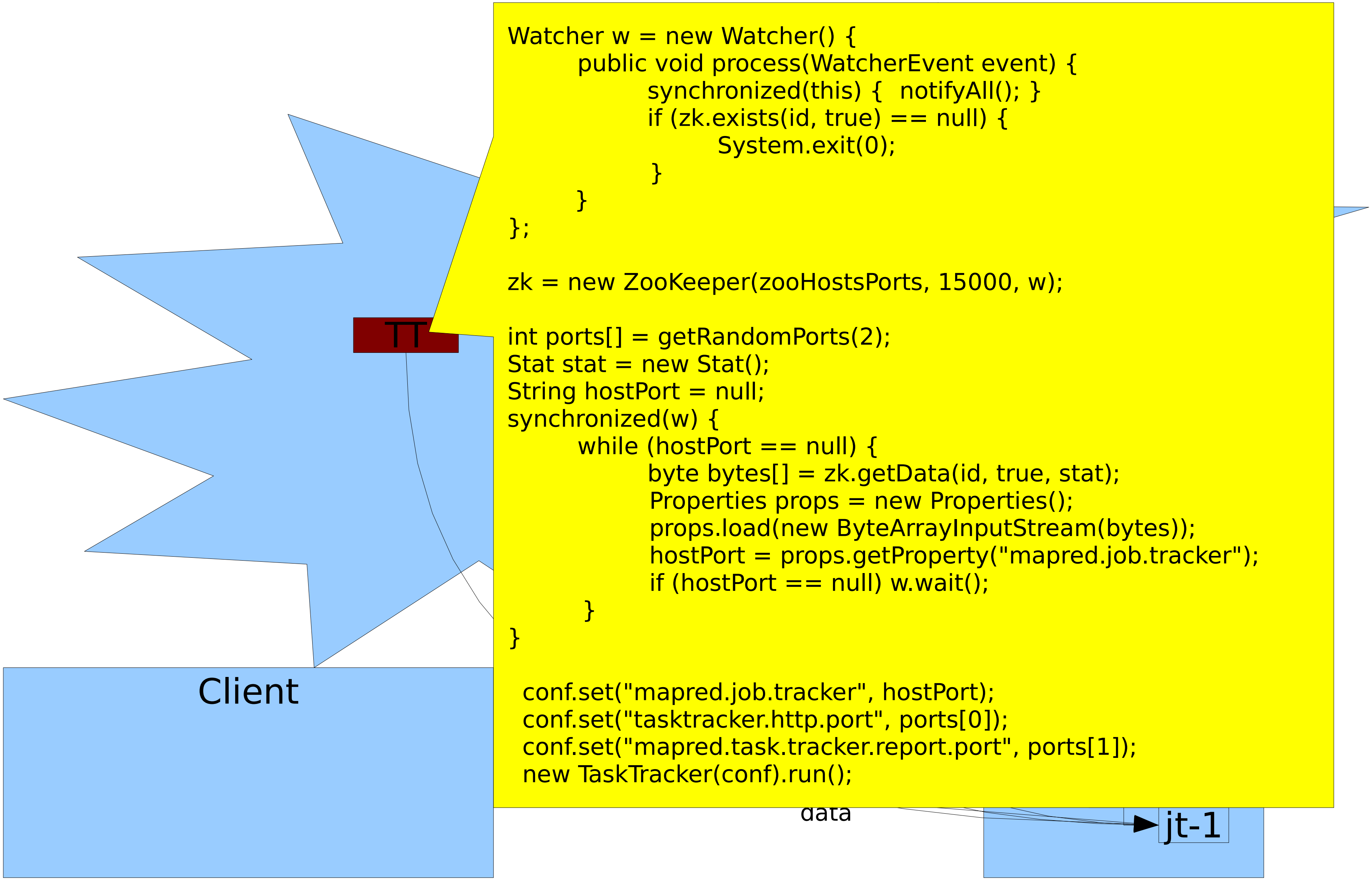


HOD with ZooKeeper



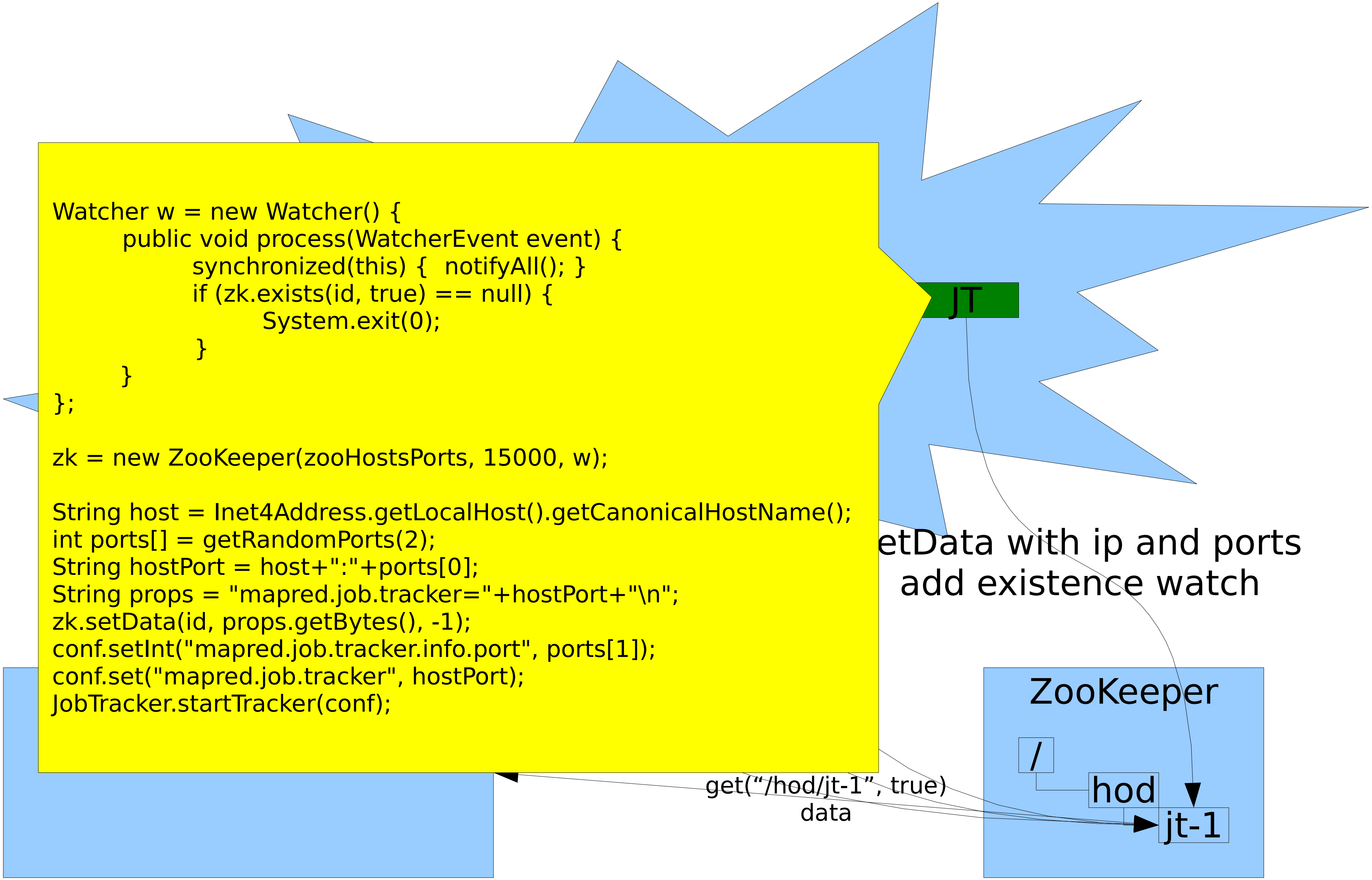
```
Watcher w = new Watcher() {
        public void process(WatcherEvent event) {
              if (event.getPath()!=null && path!=null && path.equals(event.getPath())) {
                        synchronized(this) {  notifyAll(); }
              }
        }
};

ZooKeeper zk = new ZooKeeper(zooHostsPorts, 15000, w);

path = zk.create("/hod/job-", null, null, CreateFlags.EPHEMERAL|CreateFlags.SEQUENCE);
Stat s = new Stat();

byte b[] = zk.getData(path, true, s);
while(b.length == 0) {
        synchronized(w) {
                w.wait();
           b = zk.getData(path, true, s);
     }
}
```

HOD with ZooKeeper



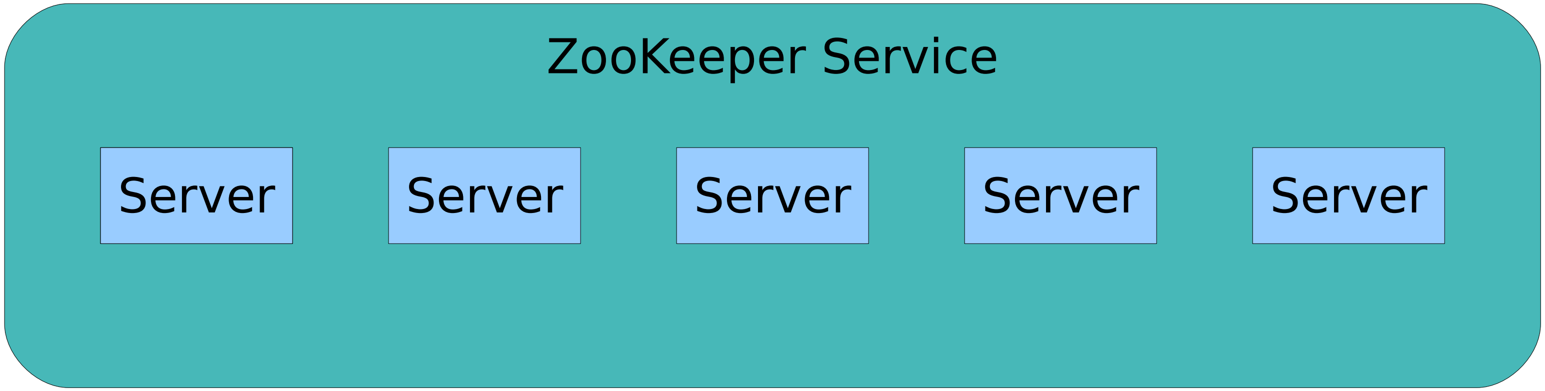
```
Watcher w = new Watcher() {
        public void process(WatcherEvent event) {
                synchronized(this) {  notifyAll(); }
                if (zk.exists(id, true) == null) {
                        System.exit(0);
                }
        }
};

zk = new ZooKeeper(zooHostsPorts, 15000, w);

int ports[] = getRandomPorts(2);
Stat stat = new Stat();
String hostPort = null;
synchronized(w) {
        while (hostPort == null) {
                byte bytes[] = zk.getData(id, true, stat);
                Properties props = new Properties();
                props.load(new ByteArrayInputStream(bytes));
                hostPort = props.getProperty("mapred.job.tracker");
                if (hostPort == null) w.wait();
         }
}

  conf.set("mapred.job.tracker", hostPort);
  conf.set("tasktracker.http.port", ports[0]);
  conf.set("mapred.task.tracker.report.port", ports[1]);
  new TaskTracker(conf).run();
```

HOD with ZooKeeper



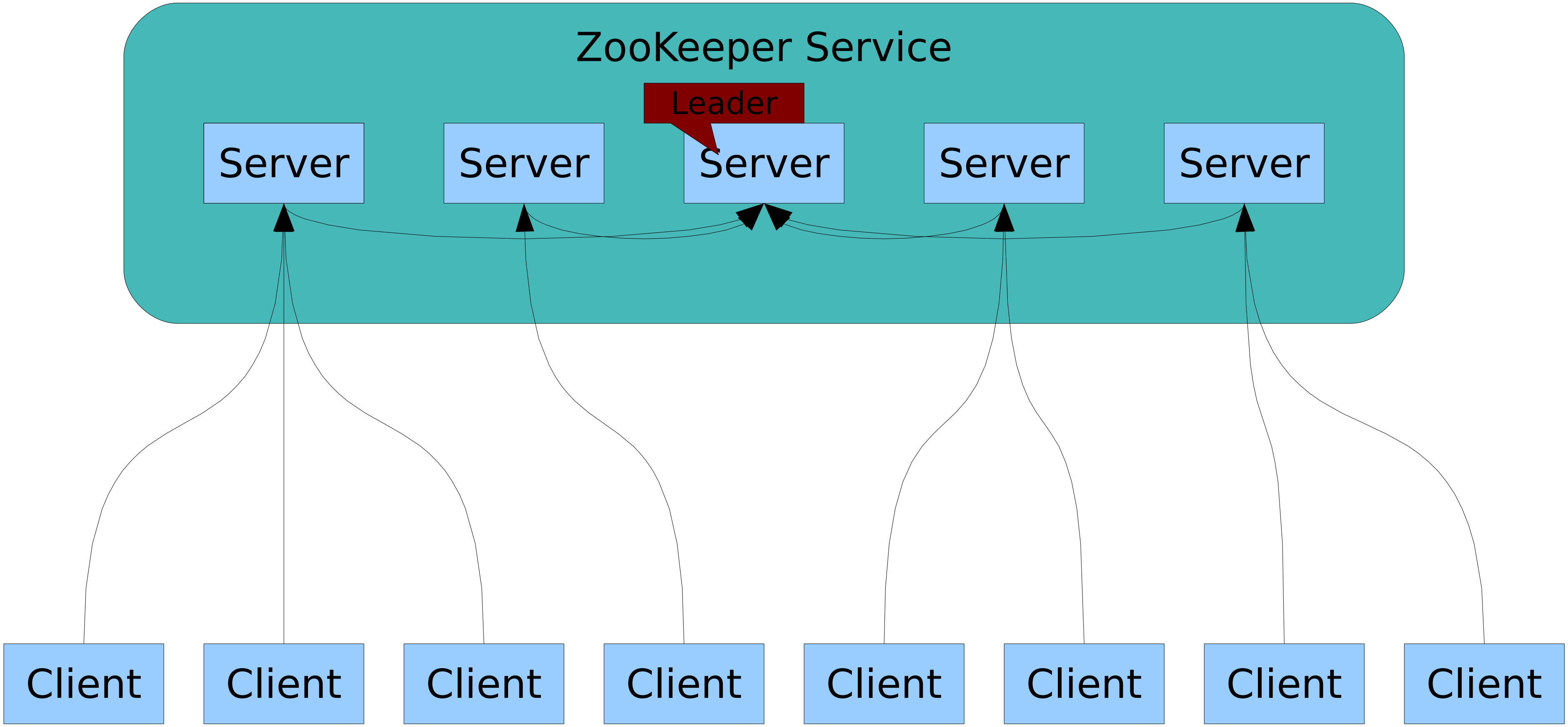
```
Watcher w = new Watcher() {
        public void process(WatcherEvent event) {
                synchronized(this) {  notifyAll(); }
                if (zk.exists(id, true) == null) {
                        System.exit(0);
                }
        }
};

zk = new ZooKeeper(zooHostsPorts, 15000, w);

String host = Inet4Address.getLocalHost().getCanonicalHostName();
int ports[] = getRandomPorts(2);
String hostPort = host+":"+ports[0];
String props = "mapred.job.tracker="+hostPort+"\n";
zk.setData(id, props.getBytes(), -1);
conf.setInt("mapred.job.tracker.info.port", ports[1]);
conf.set("mapred.job.tracker", hostPort);
JobTracker.startTracker(conf);
```

ZooKeeper Servers



1) All servers store a copy of the data
2) A leader is elected at startup
3) Followers service clients, all updates go through leader
4) Update responses are sent when a majority of servers have persisted the change

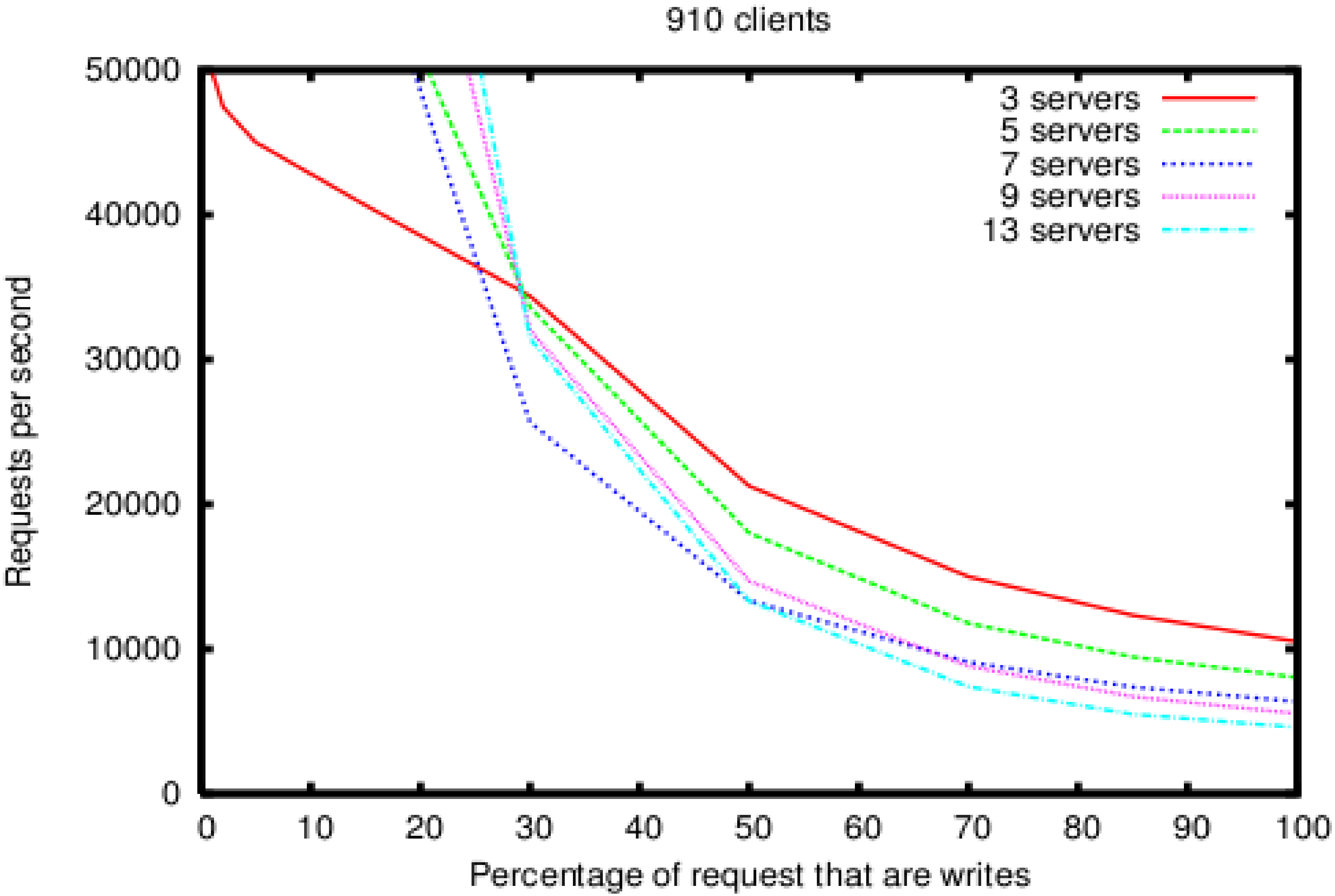
ZooKeeper Servers



Performance at Extremes

Servers	1% Writes	100% Writes
13	265115	4592
9	195178	5550
7	147810	6371
5	75308	8048
3	49827	10519

Performance



Cool Related Projects

- Client libraries for higher level primitives (Avery Ching and Jacob Levy)
- ZooKeeper FUSE (Swee Lim)

Status

- Code on zookeeper.sf.net
- Quorum and Standalone servers working
- Java and C clients available
- Working on cross colo ZooKeeper
- Starting design of distributed ZooKeeper