ZooKeeper

Scott Leberknight

Your mission...

(should you choose to accept it)

Build a distributed lock service

Only one process may own the lock

Must preserve ordering of requests

Ensure proper lock release

...this message will self destruct in 5 seconds

Mission Training

"Distributed coordination service"

Distributed, hierarchical filesystem

High availability, fault tolerant

Performant (i.e. it's fast)

Facilitates loose coupling

Fallacies of Distributed computing...

1 The network is reliable.

http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing

Partial **Failure**

Did my message get through?

Did the operation complete?

Or, did it fail???

Follow the Leader

What problems can it solve?

Group membership

Distributed data structures
(locks, queues, barriers, etc.)

Reliable configuration service

Distributed workflow

Training exercise:

Group Membership

Get connected...

```
public ZooKeeper connect(String hosts, int timeout)
      throws IOException, InterruptedException {

  final CountDownLatch signal = new CountDownLatch(1);
  ZooKeeper zk = new ZooKeeper(hosts, timeout, new Watcher() {
    @Override
    public void process(WatchedEvent event) {
      if (event.getState() ==
          Watcher.Event.KeeperState.SyncConnected) {
        signal.countDown();
      }
    }
  });
  signal.await();
  return zk;
}
```

ZooKeeper sessions

Tick time

Session timeout

Automatic failover

znode

Persistent or ephemeral

Optional sequential numbering

Can hold data (like a file)

Persistent ones can have children (like a directory)

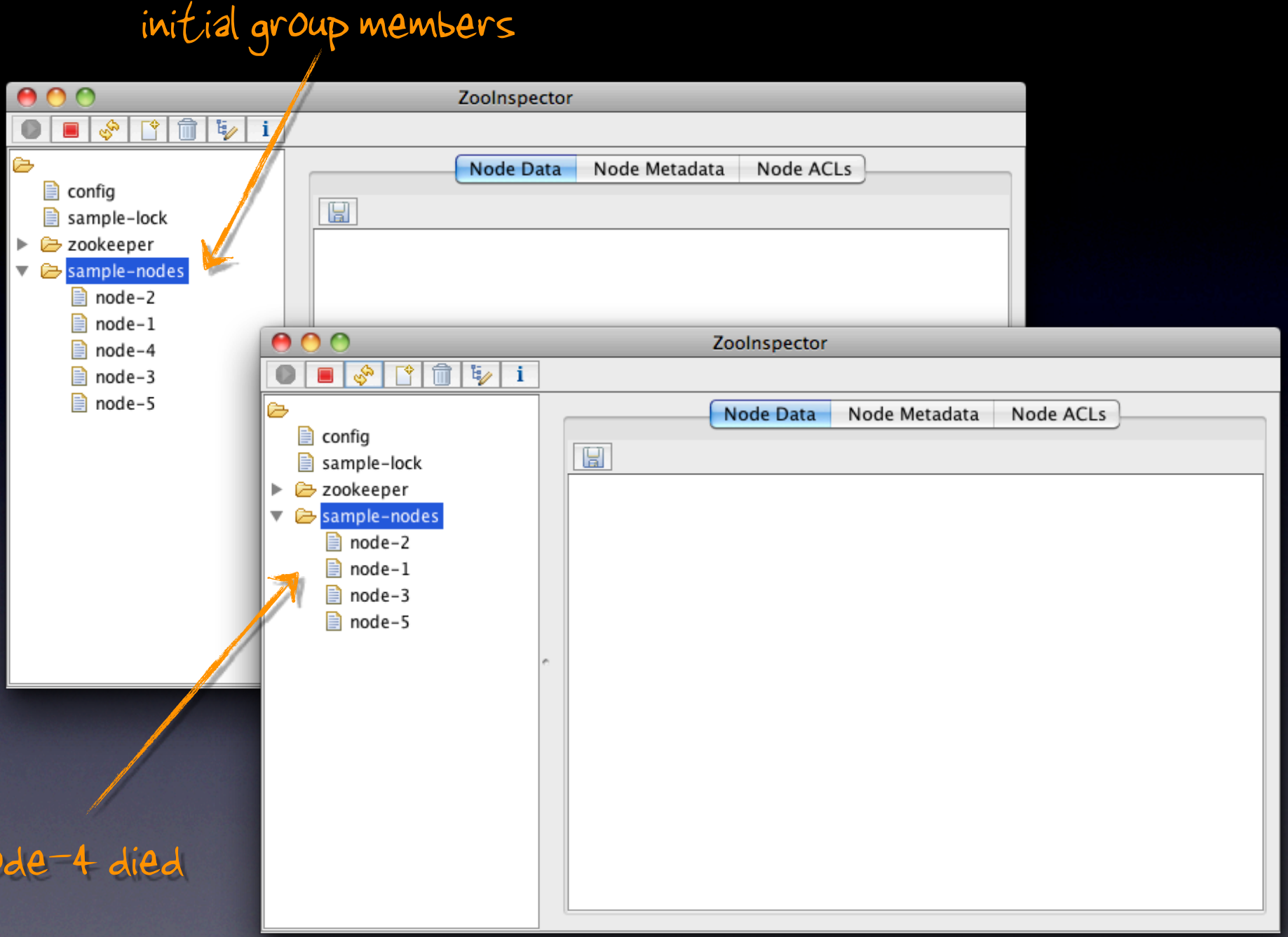
```
public void create(String groupName)
    throws KeeperException, InterruptedException {

  String path = "/" + groupName;
  String createdPath = zk.create(path,
    null /*data*/,
    ZooDefs.Ids.OPEN_ACL_UNSAFE,
    CreateMode.PERSISTENT);
  System.out.println("Created " + createdPath);
}
```

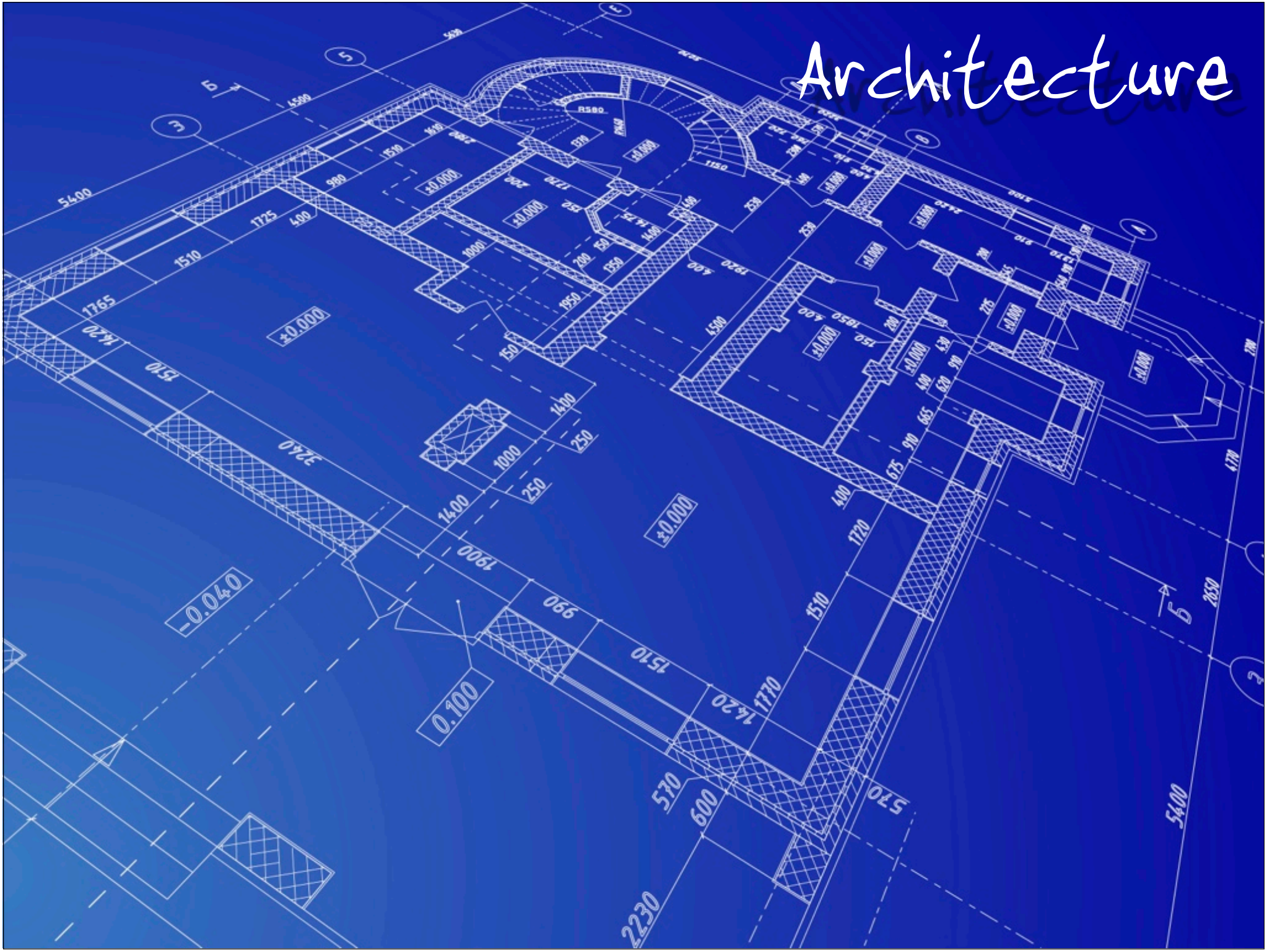
```
public void join(String groupName, String memberName)
    throws KeeperException, InterruptedException {

  String path = "/" + groupName + "/" + memberName;
  String createdPath = zk.create(path,
    null /*data*/,
    ZooDefs.Ids.OPEN_ACL_UNSAFE,
    CreateMode.EPHEMERAL);
  System.out.println("Created " + createdPath);
}
```

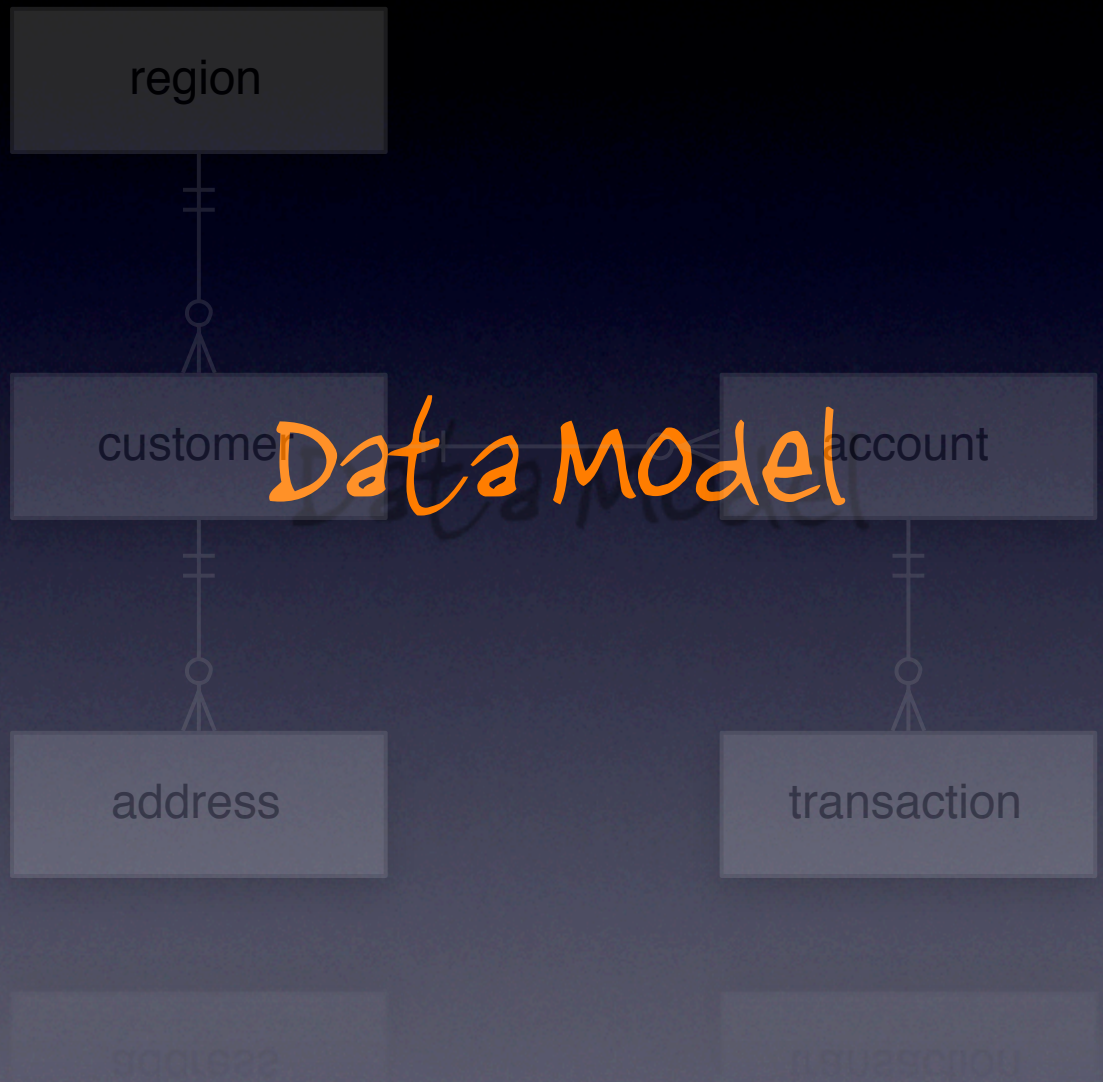
```
public void list(String groupName)
    throws KeeperException, InterruptedException {

  String path = "/" + groupName;
  try {
    List<String> children = zk.getChildren(path, false);
    for (String child : children) {
      System.out.println(child);
    }
  } catch (KeeperException.NoNodeException e) {
    System.out.printf("Group %s does not exist\n", groupName);
  }
}
```

```
public void delete(String groupName)
    throws KeeperException, InterruptedException {

  String path = "/" + groupName;
  try {
    List<String> children = zk.getChildren(path, false);
    for (String child : children) {
      zk.delete(path + "/" + child, -1);
    }
    zk.delete(path, -1);  // parent
  } catch (KeeperException.NoNodeException e) {
    System.out.printf("%s does not exist\n", groupName);
  }
}
```

-1 deletes unconditionally, or specify version


initial group members
ZooInspector
config
sample-lock
zookeeper
sample-nodes
node-2
node-1
node-4
node-3
node-5
Node Data
Node Metadata
Node ACLs
ZooInspector
config
sample-lock
zookeeper
sample-nodes
node-2
node-1
node-3
node-5
Node Data
Node Metadata
Node ACLs
node-4 died


Architecture



Data Model

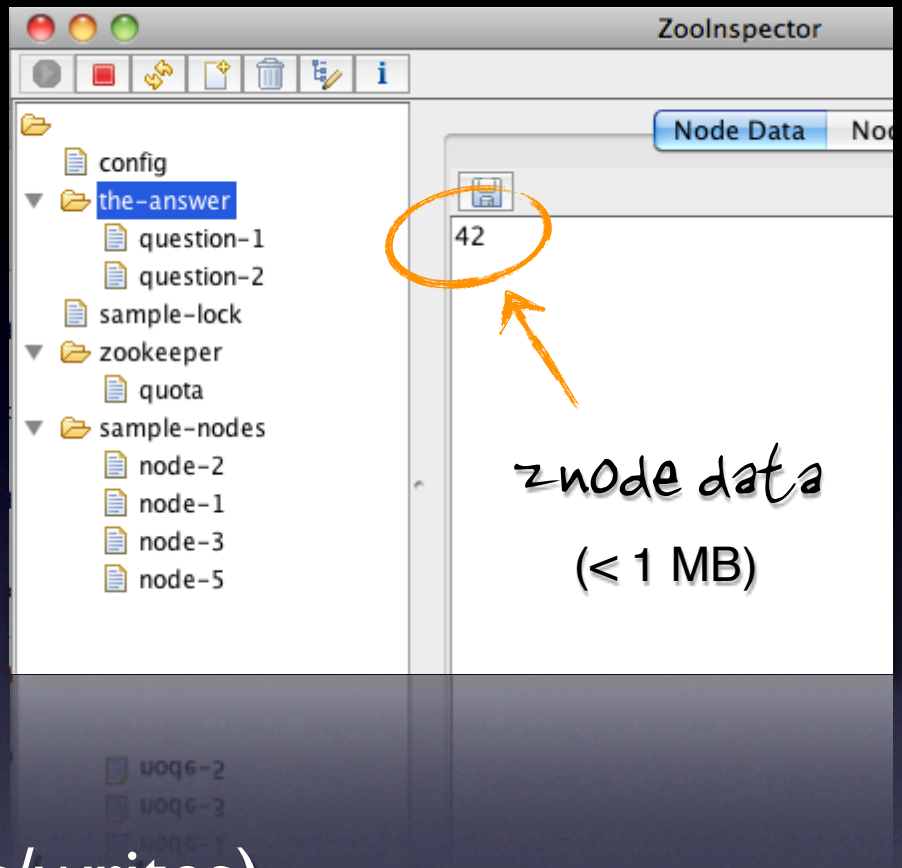
Hierarchical filesystem

Comprised of **`znodes`**

Watchers

Atomic **`znode`** access (reads/writes)

Security
(via authentication, ACLs)



znode - types

Persistent

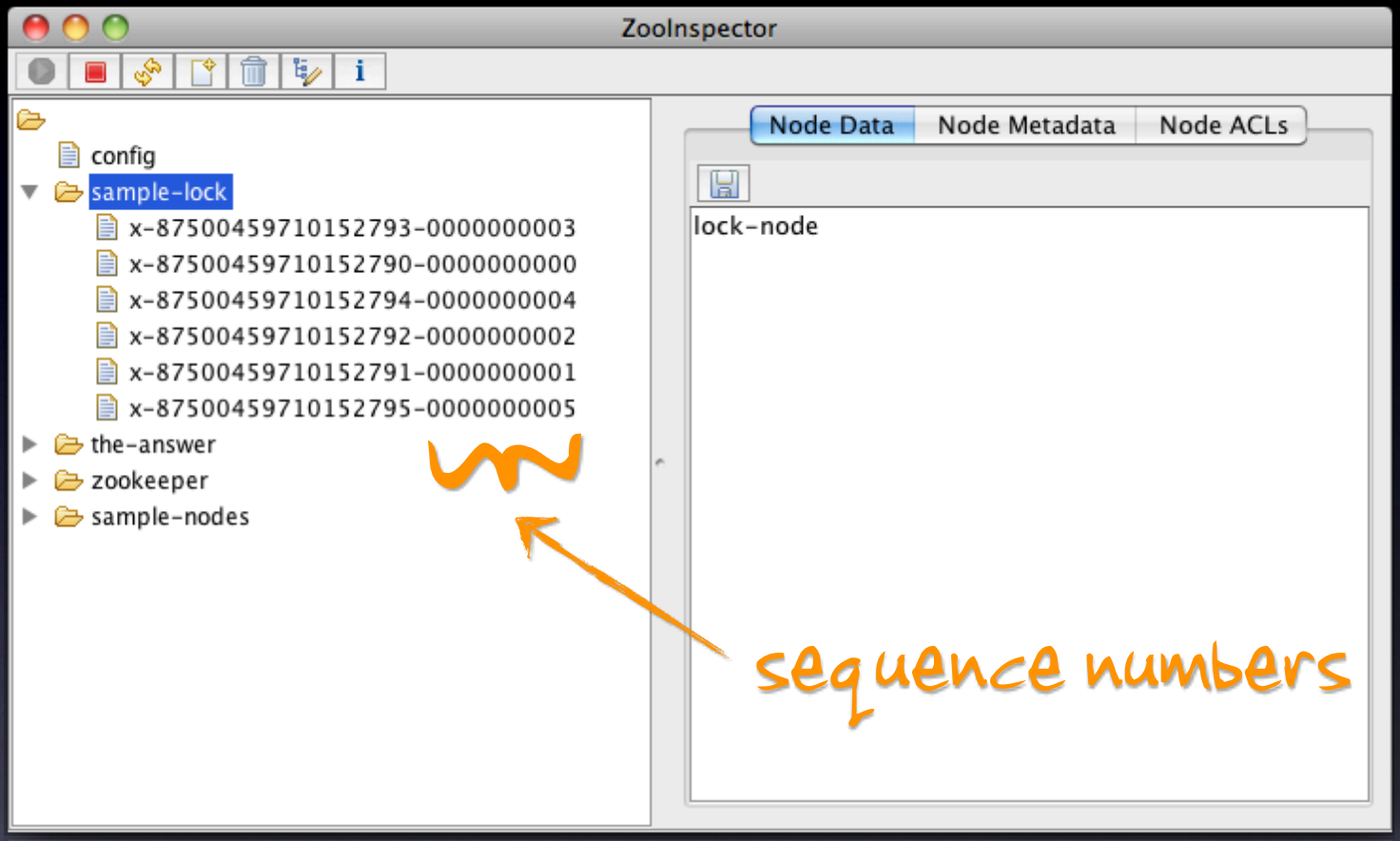
Persistent Sequential

Ephemeral

Ephemeral Sequential

die when session expires

znode - sequential



znode - operations

Operation	Type
create	write
delete	write
exists	read
getChildren	read
getData	read
setData	write
getACL	read
setACL	write
sync	read

APIs

synchronous

asynchronous

Synchronous

```
public void list(final String groupName)
    throws KeeperException, InterruptedException {

  String path = "/" + groupName;
  try {
    List<String> children = zk.getChildren(path, false);
    // process children...
  }
  catch (KeeperException.NoNodeException e) {
    // handle non-existent path...
  }
}
```

aSync

```
public void list(String groupName)
    throws KeeperException, InterruptedException {

  String path = "/" + groupName;
  zk.getChildren(path, false,
    new AsyncCallback.ChildrenCallback() {
      @Override
      public void processResult(int rc, String path,
          Object ctx, List<String> children) {
        // process results when get called back later...
      }
    }, null /* optional context object */);
}
```

znode - Watchers

Set (one-time) watches on read operations

Write operations trigger watches
on affected znodes

Re-register watches on events (optional)

```
public interface Watcher {

  void process(WatchedEvent event);

  // other details elided...
}
```

```
public class ChildZNodeWatcher implements Watcher {
  private Semaphore semaphore = new Semaphore(1);

  public void watchChildren()
        throws InterruptedException, KeeperException {
    semaphore.acquire();
    while (true) {
      List<String> children = zk.getChildren(lockPath, this);
      display(children);
      semaphore.acquire();
    }
  }
  @Override public void process(WatchedEvent event) {
    if (event.getType() == Event.EventType.NodeChildrenChanged) {
      semaphore.release();
    }
  }
  // other details elided...
}
```

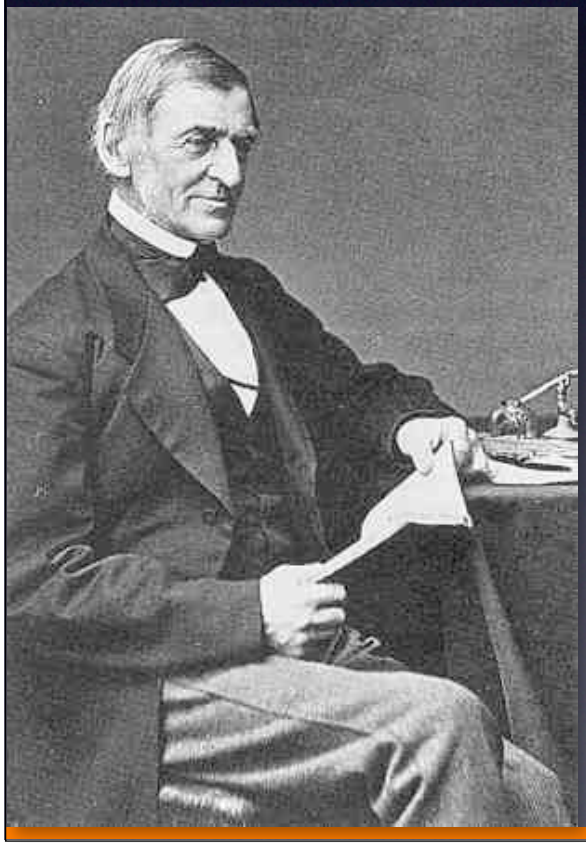
When using watchers...

...updates can be missed!

(not seen by client)

Data consistency

"A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines. With consistency a great soul has simply nothing to do."



- Ralph Waldo Emerson
(Self-Reliance)

Data consistency in ZK

Sequential updates

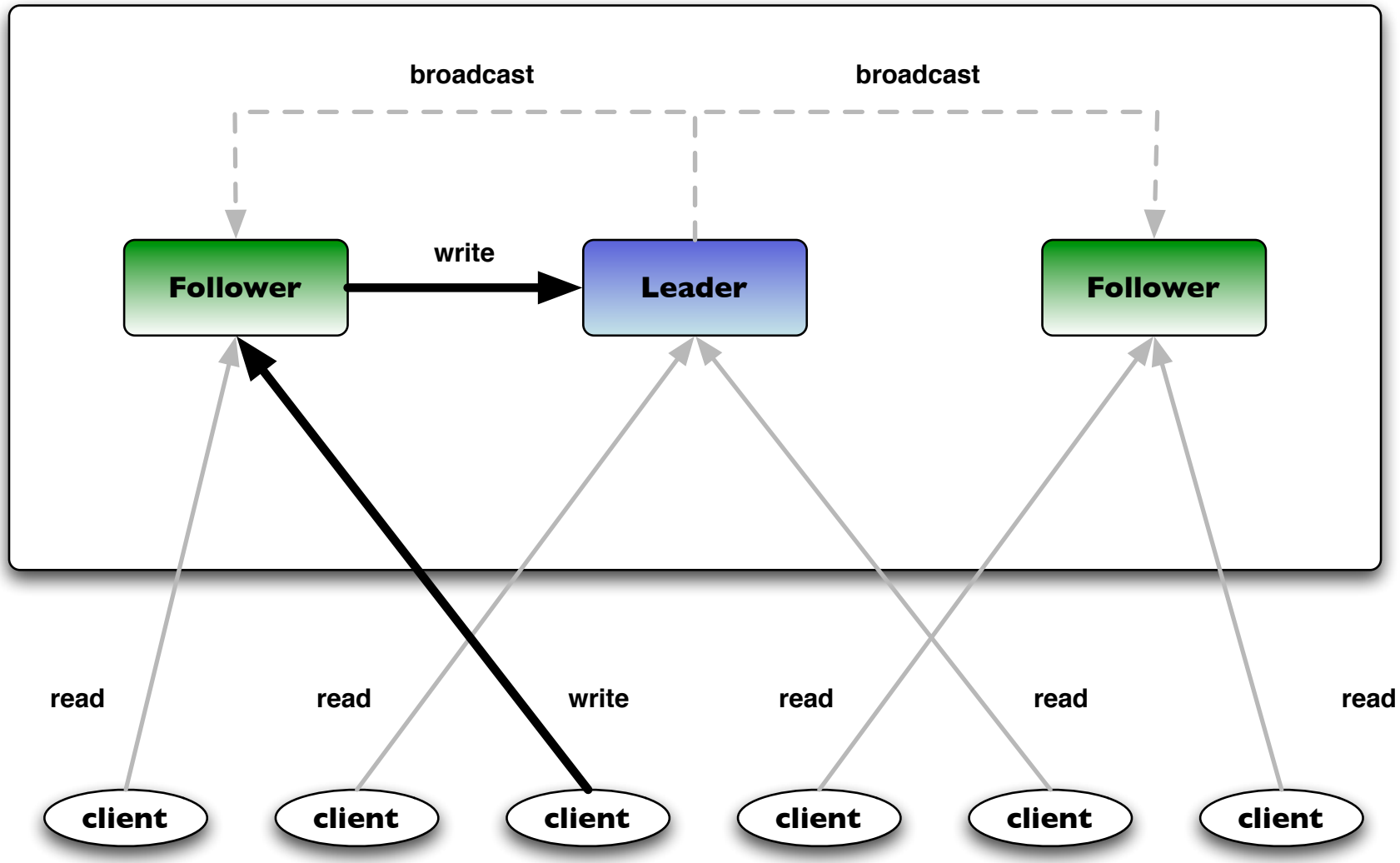
Atomicity (all or nothin')

Consistent client view
(across all ZooKeeper servers)

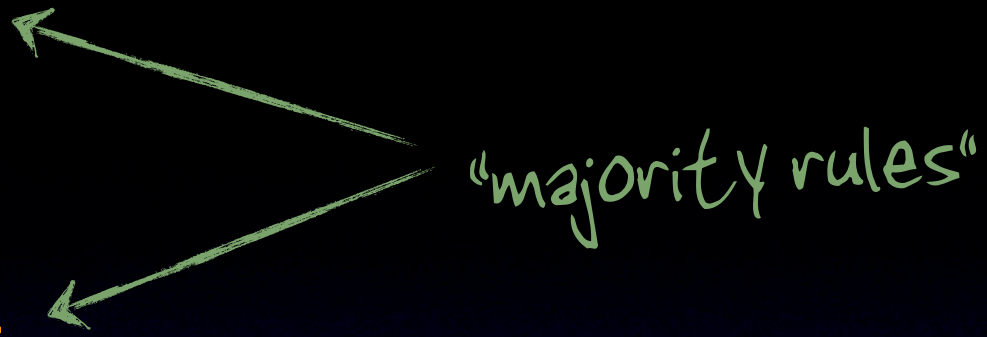
Durability (of writes)

Bounded lag time
(eventual consistency)

Ensemble


broadcast
broadcast
Follower
write
Leader
Follower
read
read
write
read
read
read
client
client
client
client
client
client

Leader election



Atomic broadcast

Clients have session on one server

Writes routed through leader

Reads from server memory

Sessions

Client-requested timeout period

Automatic keep-alive (heartbeats)

Automatic/transparent failover

Writes

All go through leader

Broadcast to followers

Global ordering → every update has unique

`zxid` (ZooKeeper transaction id)

Reads

"Follow-the-leader"

Eventual consistency

Can lag leader

In-memory (*fast!*)

Training Complete

Final Mission:

Distributed Lock

Objectives

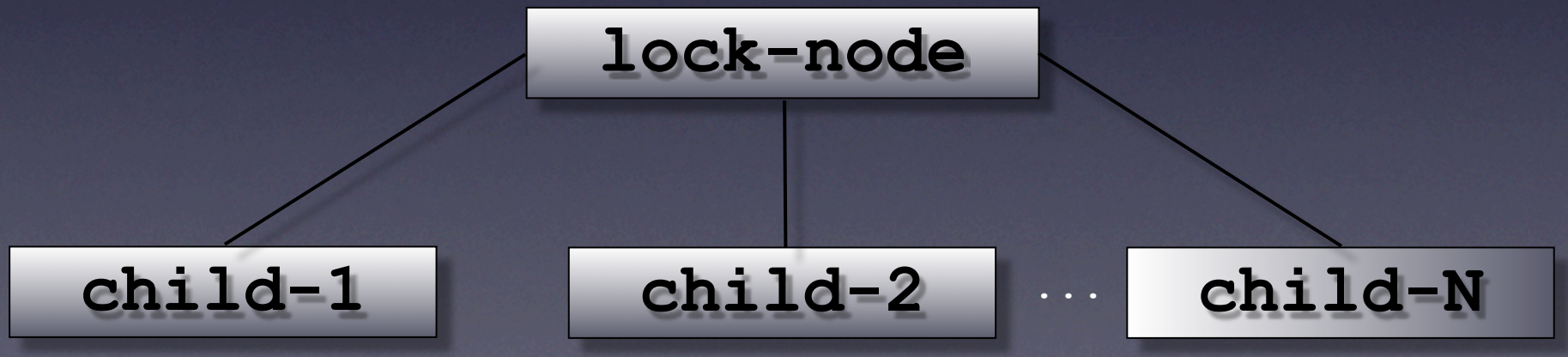
Mutual exclusion between processes

Decouple lock users

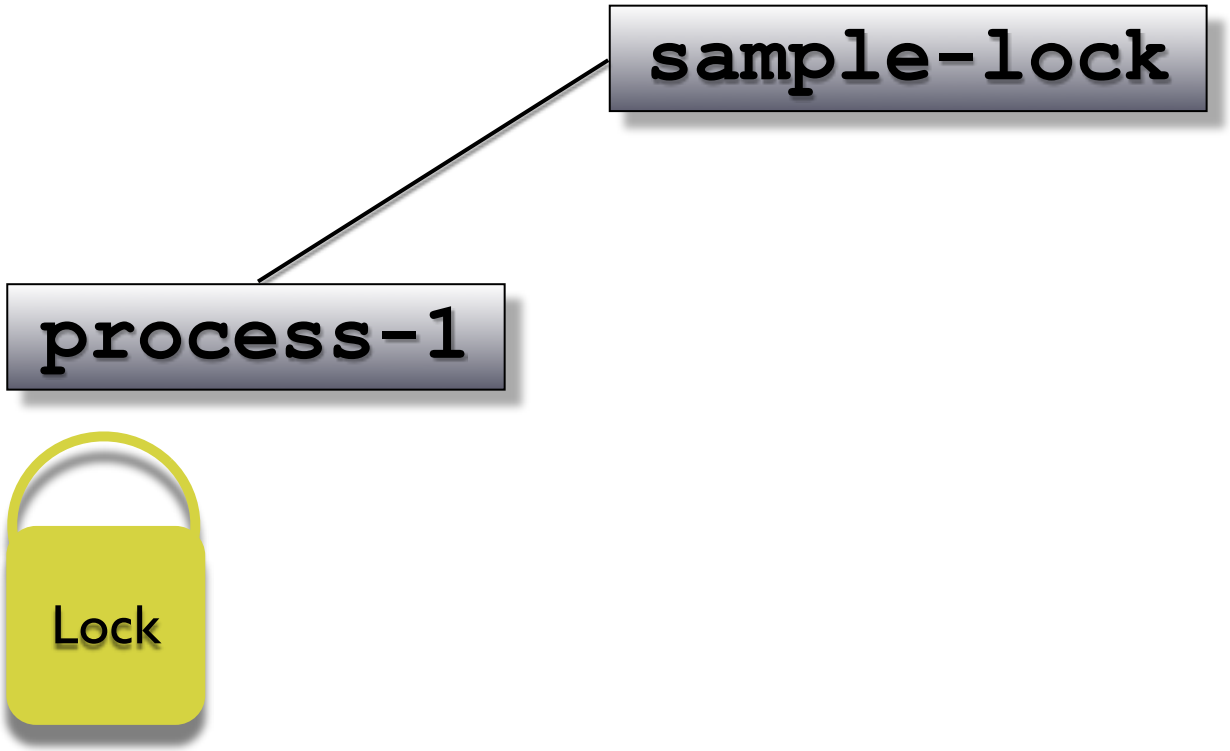
Building Blocks

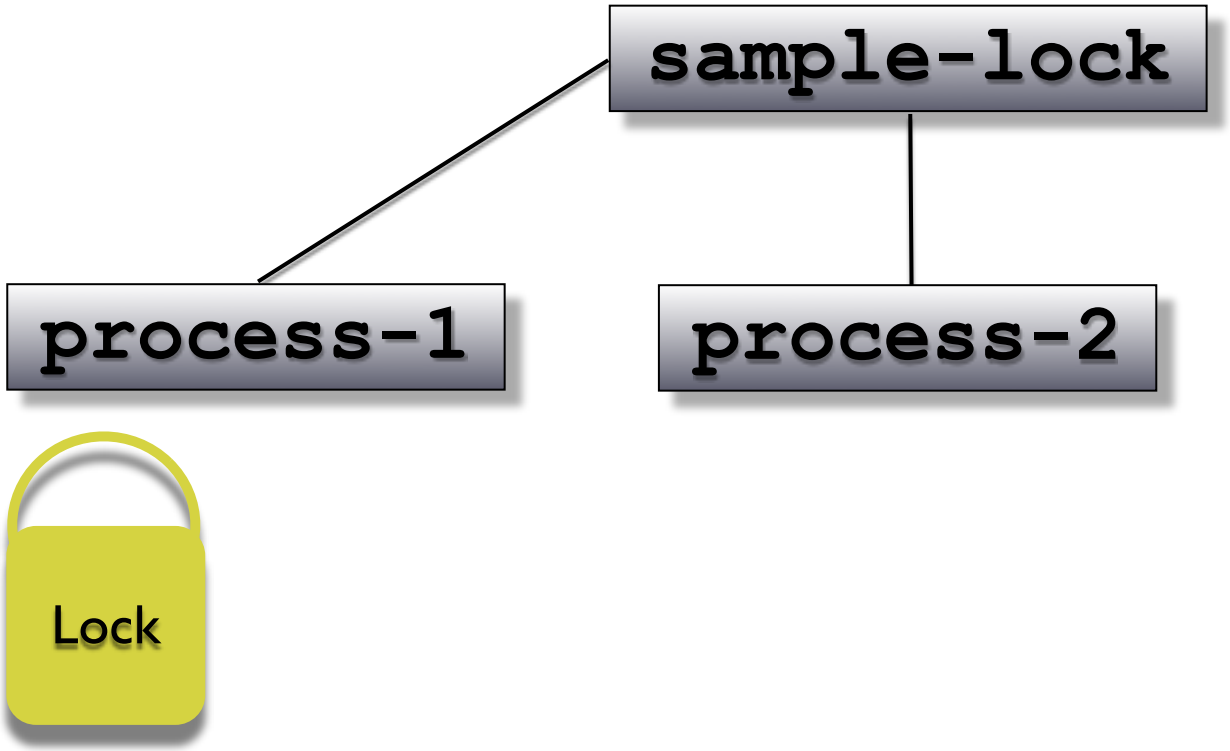
parent lock **znode**

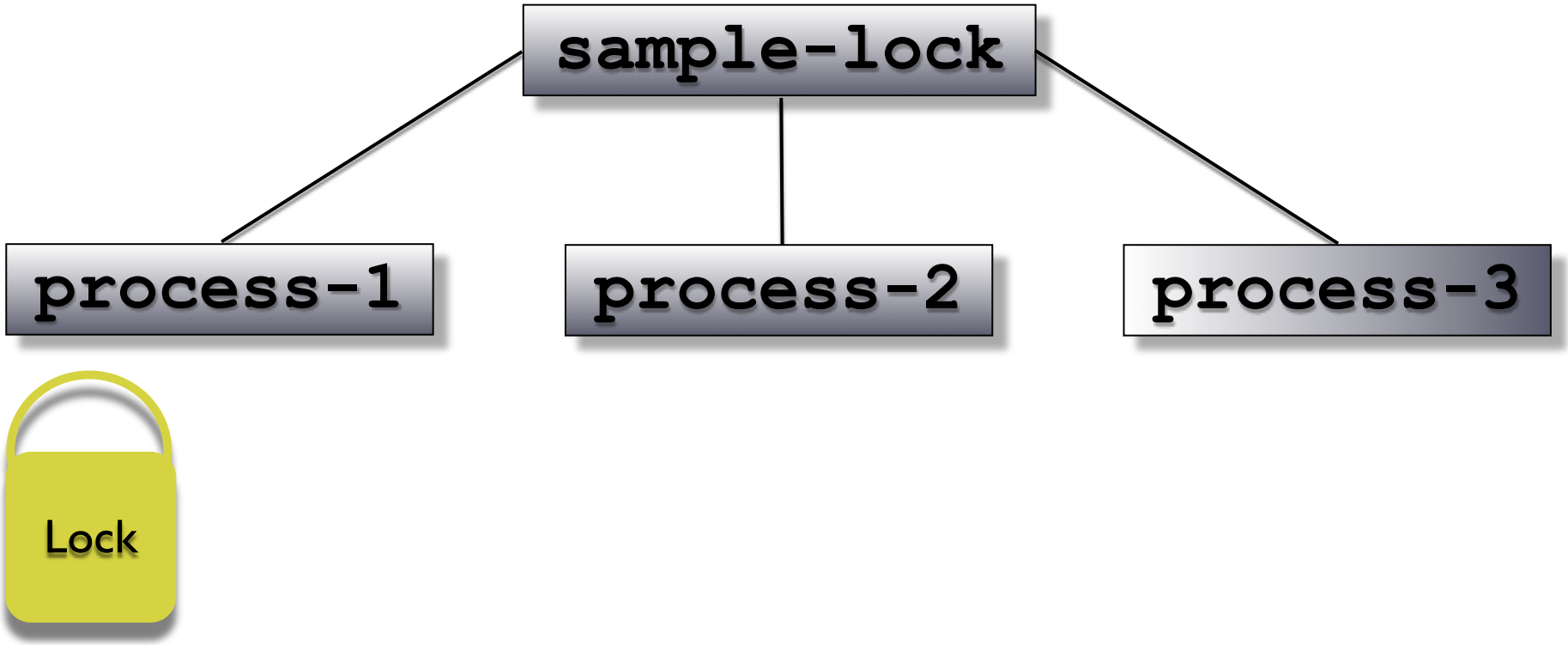
ephemeral, sequential child **znodes**

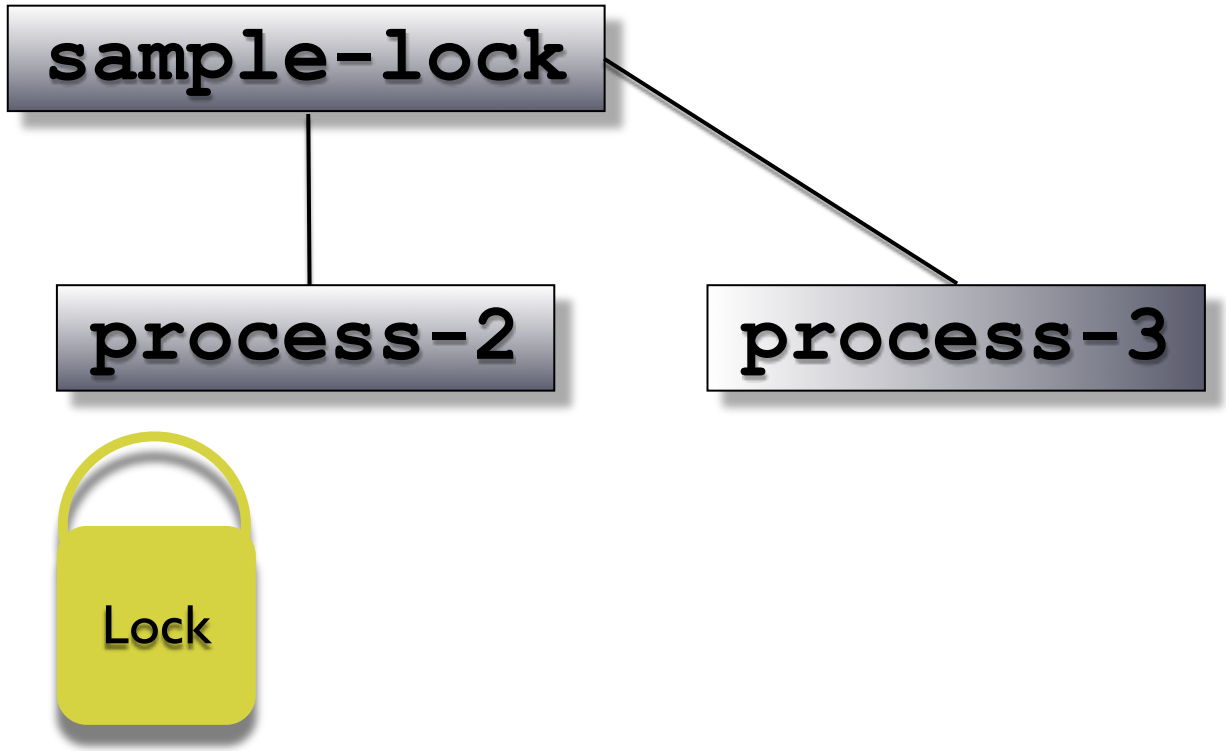


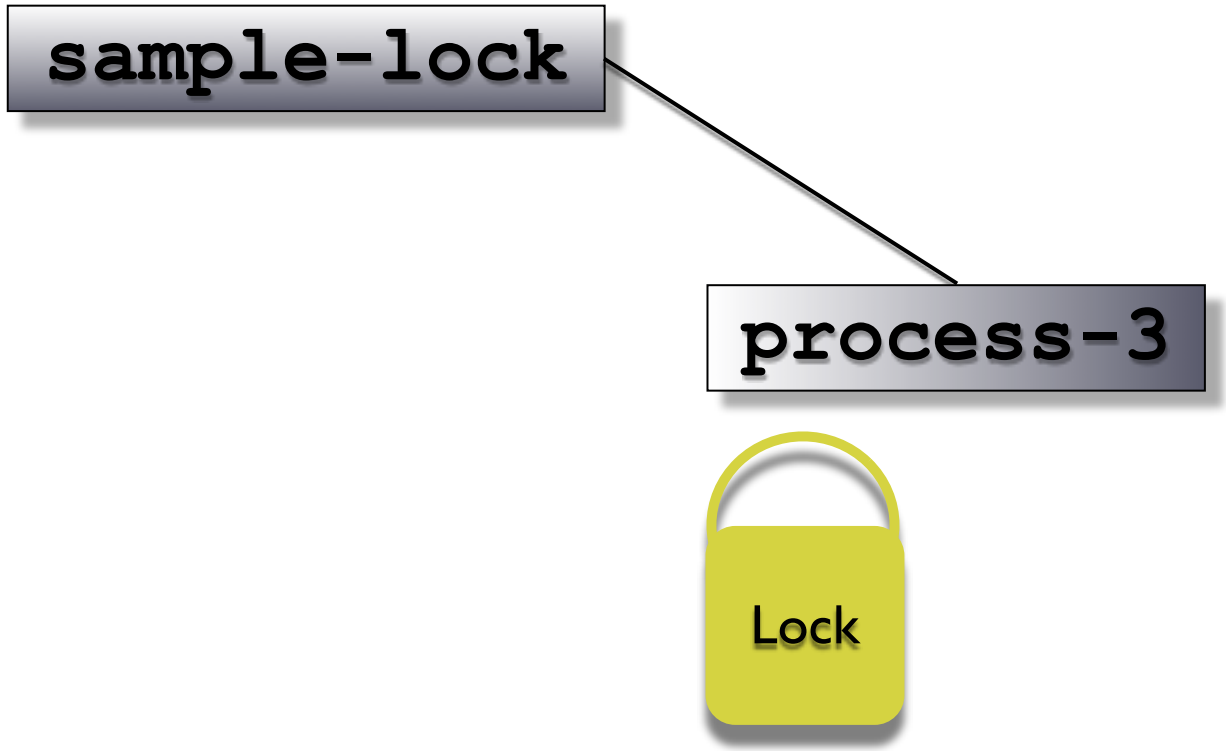
sample-lock


sample-lock
process-1
Lock


sample-lock
process-1
process-2
Lock


sample-lock
process-1
process-2
process-3
Lock


sample-lock
process-2
process-3
Lock


sample-lock
process-3
Lock

sample-lock

pseudocode (remember that stuff?)

1. create child ephemeral, sequential znode

2. list children of lock node and set a watch

3. if znode from step 1 has lowest number,
   then lock is acquired. done.

4. else wait for watch event from step 2,
   and then go back to step 2 on receipt

This is OK, but there are problems...

(perhaps a main B bus undervolt?)

Problem #1 – connection Loss

If partial failure on **znode** creation...

...then how do we know if **znode** was created?

Solution #1 (Connection Loss)

Embed session **`id`** in child **`znode`** names

`lock-<sessionId>-`

Failed-over client checks for child w/ **`sessionId`**


Problem #2 – The Herd Effect

Many clients watching lock node.

Notification sent to all watchers on lock release...

...possible large spike in network traffic

Solution #2 (The Herd Effect)

Set watch only on child **`znode`**
with preceding sequence number

e.g. client holding **`lock-9`** watches only **`lock-8`**

Note, **`lock-9`** could watch **`lock-7`**
(e.g. lock-8 client died)

Implementation

Do it yourself?

...or be lazy and use existing code?

```
org.apache.zookeeper.recipes.lock.WriteLock
org.apache.zookeeper.recipes.lock.LockListener
```

Implementation, part deux

`WriteLock` calls back when lock acquired (async)

Maybe you want a synchronous client model...

Let's do some decoration...

```
public class BlockingWriteLock {
  private CountDownLatch signal = new CountDownLatch(1);
  // other instance variable definitions elided...

  public BlockingWriteLock(String name, ZooKeeper zookeeper,
      String path, List<ACL> acls) {
    this.name = name;
    this.path = path;
    this.writeLock = new WriteLock(zookeeper, path,
                                   acls, new SyncLockListener());
  }

  public void lock() throws InterruptedException, KeeperException {
    writeLock.lock();
    signal.await();
  }

  public void unlock() { writeLock.unlock(); }

  class SyncLockListener implements LockListener { /* next slide */ }
}
```

```
class SyncLockListener implements LockListener {

  @Override public void lockAcquired() {
    signal.countDown();
  }

  @Override public void lockReleased() {
  }

}
```

```
BlockingWriteLock lock =
  new BlockingWriteLock(myName, zooKeeper, path,
                        ZooDefs.Ids.OPEN_ACL_UNSAFE);
try {
  lock.lock();
  // do something while we have the lock
}
catch (Exception ex) {
  // handle appropriately...
}
finally {
  lock.unlock();
}
```

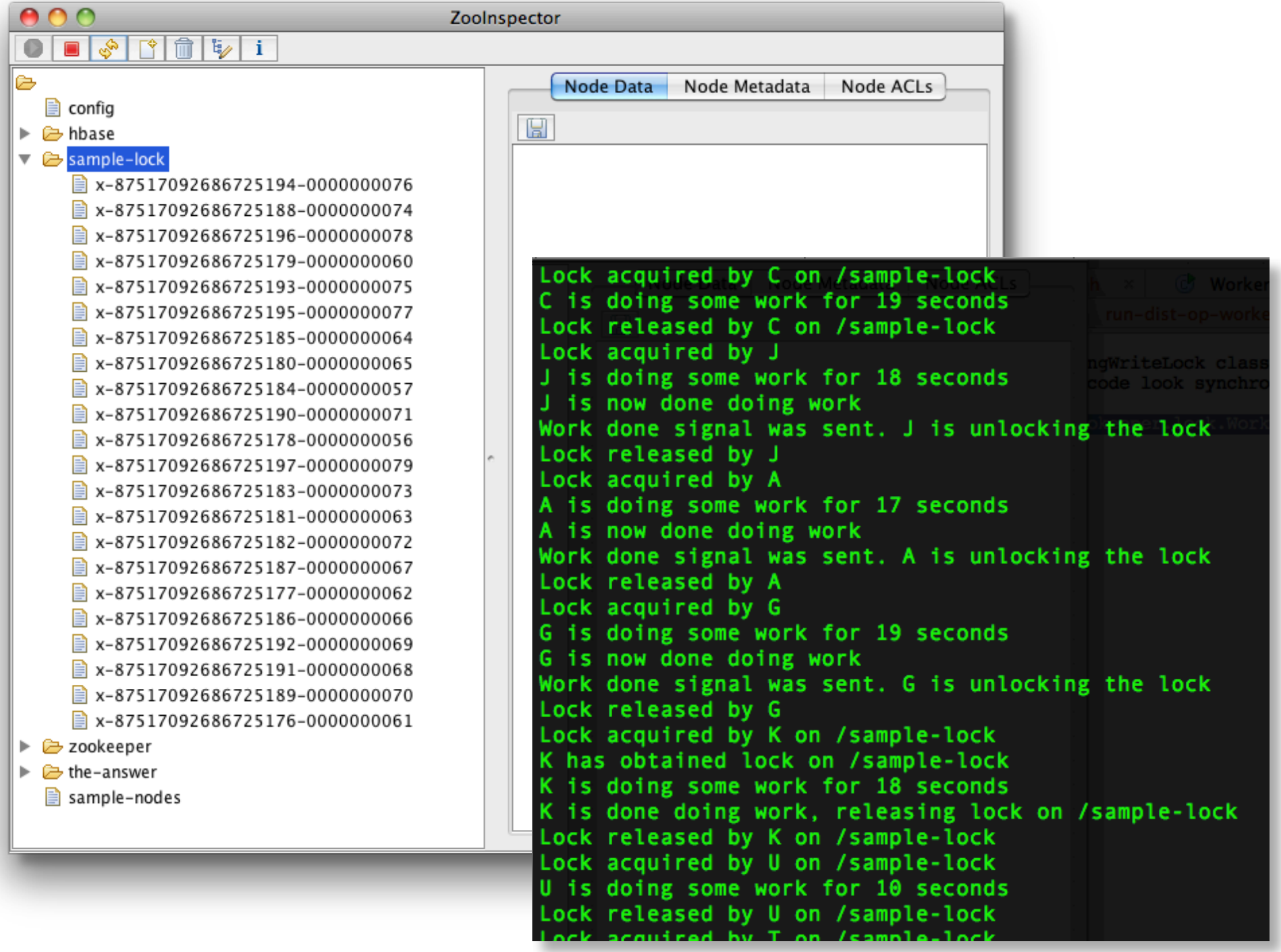
Easy to forget!

(we can do a little better, right?)

```
public class DistributedOperationExecutor {
  private final ZooKeeper _zk;
  public DistributedOperationExecutor(ZooKeeper zk) { _zk = zk; }

  public Object withLock(String name, String lockPath,
                         List<ACL> acls,
                         DistributedOperation op)
      throws InterruptedException, KeeperException {
    BlockingWriteLock writeLock =
      new BlockingWriteLock(name, _zk, lockPath, acl);
    try {
      writeLock.lock();
      return op.execute();
    } finally {
      writeLock.unlock();
    }
  }
}
```

```
executor.withLock(myName, path, ZooDefs.Ids.OPEN_ACL_UNSAFE,
  new DistributedOperation() {
    @Override public Object execute() {
      // do something while we have the lock
      return whateverTheResultIs;
    }
  }
);
```


Run it!


ZooInspector
Node Data
Node Metadata
Node ACLs
config
hbase
sample-lock
x-87517092686725194-0000000076
x-87517092686725188-0000000074
x-87517092686725196-0000000078
x-87517092686725179-0000000060
x-87517092686725193-0000000075
x-87517092686725195-0000000077
x-87517092686725185-0000000064
x-87517092686725180-0000000065
x-87517092686725184-0000000057
x-87517092686725190-0000000071
x-87517092686725178-0000000056
x-87517092686725197-0000000079
x-87517092686725183-0000000073
x-87517092686725181-0000000063
x-87517092686725182-0000000072
x-87517092686725187-0000000067
x-87517092686725177-0000000062
x-87517092686725186-0000000066
x-87517092686725192-0000000069
x-87517092686725191-0000000068
x-87517092686725189-0000000070
x-87517092686725176-0000000061
zookeeper
the-answer
sample-nodes
Lock acquired by C on /sample-lock
C is doing some work for 19 seconds
Lock released by C on /sample-lock
Lock acquired by J
J is doing some work for 18 seconds
J is now done doing work
Work done signal was sent. J is unlocking the lock
Lock released by J
Lock acquired by A
A is doing some work for 17 seconds
A is now done doing work
Work done signal was sent. A is unlocking the lock
Lock released by A
Lock acquired by G
G is doing some work for 19 seconds
G is now done doing work
Work done signal was sent. G is unlocking the lock
Lock released by G
Lock acquired by K on /sample-lock
K has obtained lock on /sample-lock
K is doing some work for 18 seconds
K is done doing work, releasing lock on /sample-lock
Lock released by K on /sample-lock
Lock acquired by U on /sample-lock
U is doing some work for 10 seconds
Lock released by U on /sample-lock

Mission:

Review...

let's crowdsource

(for free beer, of course)

Like a filesystem, except distributed & replicated

Build distributed coordination, data structures, etc.

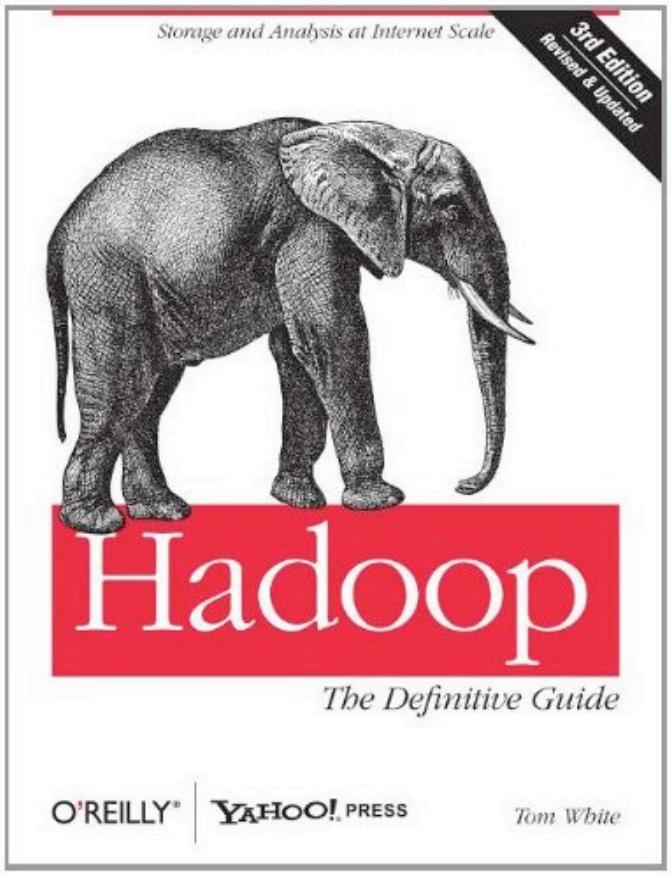
High-availability, reliability

Automatic session failover, keep-alive

Writes via leader, in-memory reads (*fast*)

Refs

zookeeper.apache.org/



shop.oreilly.com/product/0636920021773.do

(3rd edition pub date is **May 29, 2012**)

photo attributions

Follow the Leader - http://www.flickr.com/photos/davidspinks/4211977680/

Antique Clock - http://www.flickr.com/photos/cncphotos/3828163139/

Skull & Crossbones - http://www.flickr.com/photos/halfanacre/2841434212/

Ralph Waldo Emerson - http://www.americanpoems.com/poets/emerson

1921 Jazzing Orchestra - http://en.wikipedia.org/wiki/File:Jazzing_orchestra_1921.png

Apollo 13 Patch - http://science.ksc.nasa.gov/history/apollo/apollo-13/apollo-13.html

Herd of Sheep - http://www.flickr.com/photos/freefoto/639294974/

Running on Beach - http://www.flickr.com/photos/kcdale99/3148108922/

Crowd - http://www.flickr.com/photos/laubarnes/5449810523/

(others from iStockPhoto...)

(my info)

scott.leberknight@nearinfinity.com

www.nearinfinity.com/blogs/

twitter: sleberknight