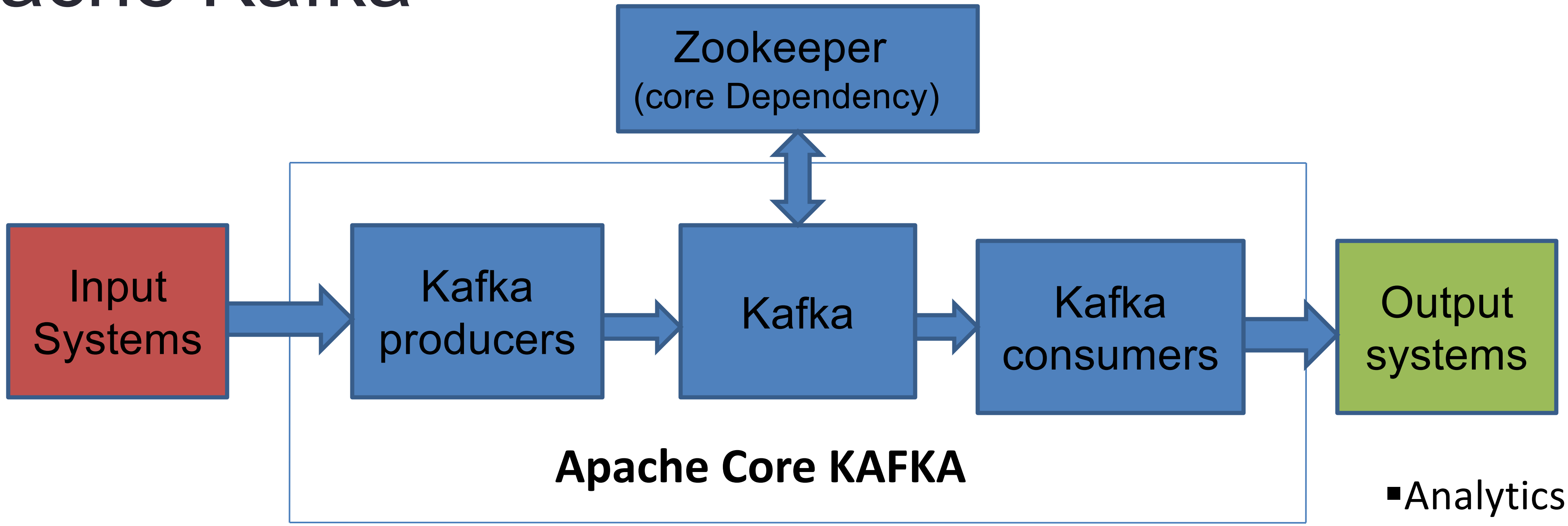
APACHE ZOOKEEPER

Apache Kafka - Core Kafka

- Kafka gets conflated with Kafka ecosystem
- Apache Core Kafka consists of Kafka Broker, start up scripts for Zoo Keeper, and client APIs for Kafka
- Apache Core Kafka does ***not*** include
  - Confluent Schema Registry (not an Apache project)
  - Kafka REST Proxy (not an Apache project)
  - Kafka Connect
  - Kafka Streams

Apache Kafka



- Log aggregation
- Metrics
- KPIs
- Batch imports
- Audit trial
- User activity logs
- Web logs

Not part of core

- Scheme Registry
- Avro
- Kafka REST Proxy
- Kafka Connect
- Kafka Streams

Apache Kafka Core

- Server Broker
- Scripts to start libs
- Script to start up Zookeeper
- Utils to create topics
- Utils to monitor stats

- Analytics
- Databases
- Machine learning
- Dashboards
- Indexed for search
- Business Intelligene

Kafka needs Zookeeper

- Zookeeper helps with leadership election of Kafka Broker and Topic Partition pairs
- Zookeeper manages service discovery for Kafka Brokers that form the cluster
- Zookeeper sends changes to Kafka
  - New Broker join, Broker died, etc.
  - Topic removed, Topic added, etc.
- Zookeeper provides in-sync view of Kafka Cluster configure

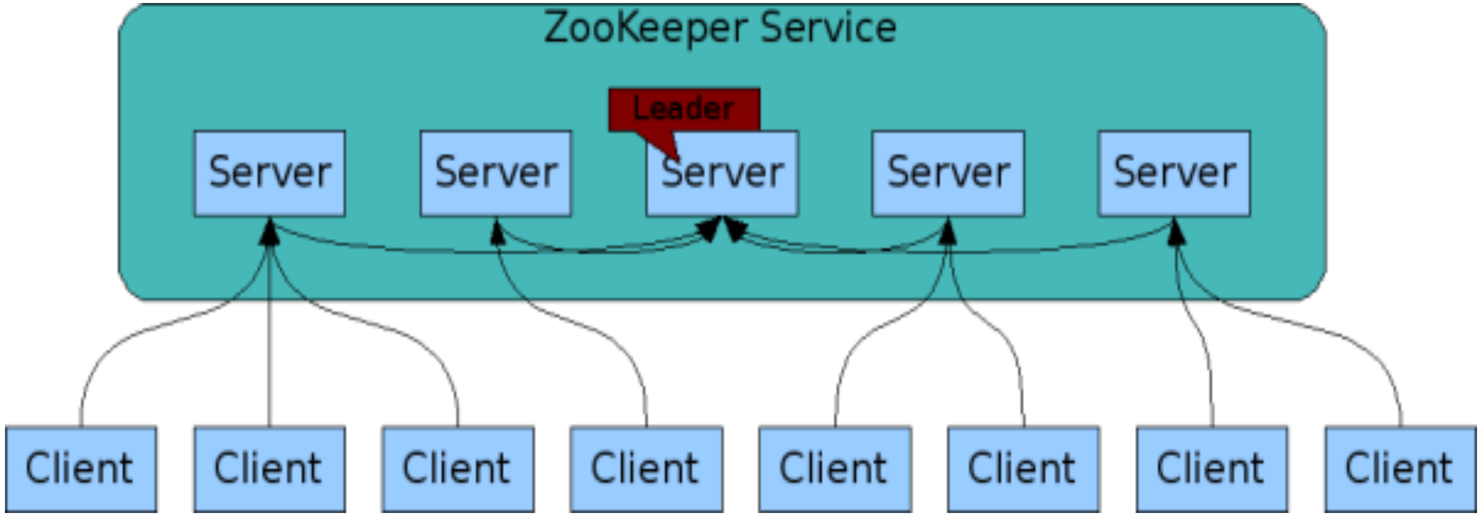
Overview – What is ZooKeeper?

- An open source, high-performance coordination service for distributed application.
- Exposes common services in simple interface:
  - Naming
  - Configuration management
  - Locks & synchronization
  - Groups services
- Build your own on it for specific needs

Overview – ZooKeeper Use Cases

- Configuration Management
  - Cluster member nodes bootstrapping configuration from a centralized source in unattended way
- Distributed Cluster Management
  - Node join / leave
  - Node statuses in real time
- Naming service – e.g. DNS
- Distributed synchronization – locks, barriers, queues
- Leader election in a distributed system

The ZooKeeper Service (ZKS)



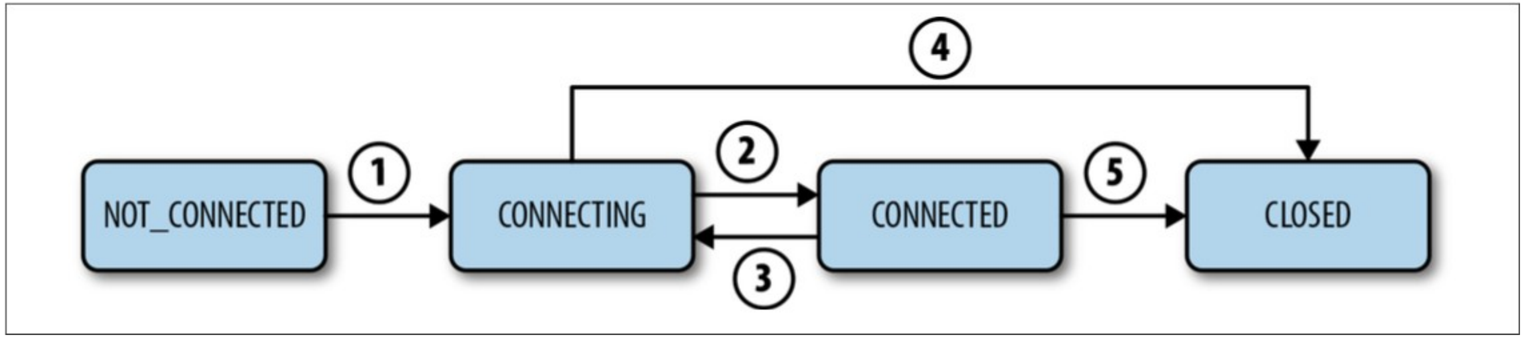
- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data (in-memory)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server and maintain a TCP connection

The ZKS - Sessions

- Before executing any request, client must establish a session with service
- All operations client summits to service are associated to a session
- Client initially connects to any server in ensemble, and only to single server.
- Session offer *order guarantees* – requests in session are executed in FIFO order

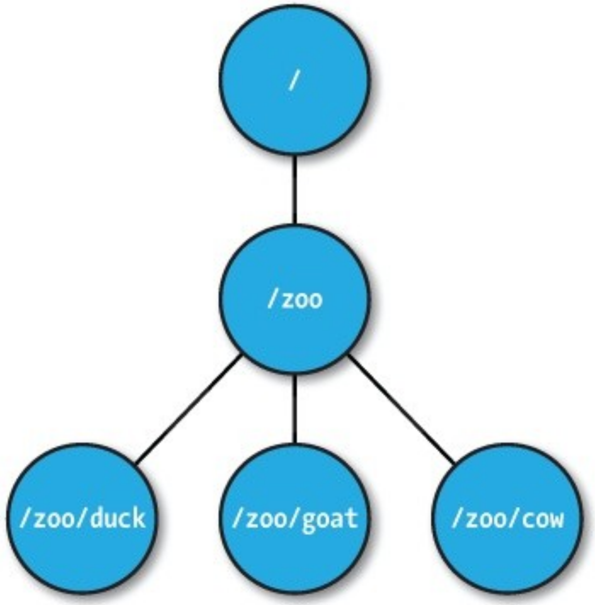
The ZKS – Session States and Lifetime

- Main possible states: CONNECTING, CONNECTED, CLOSED, NOT_CONNECTED



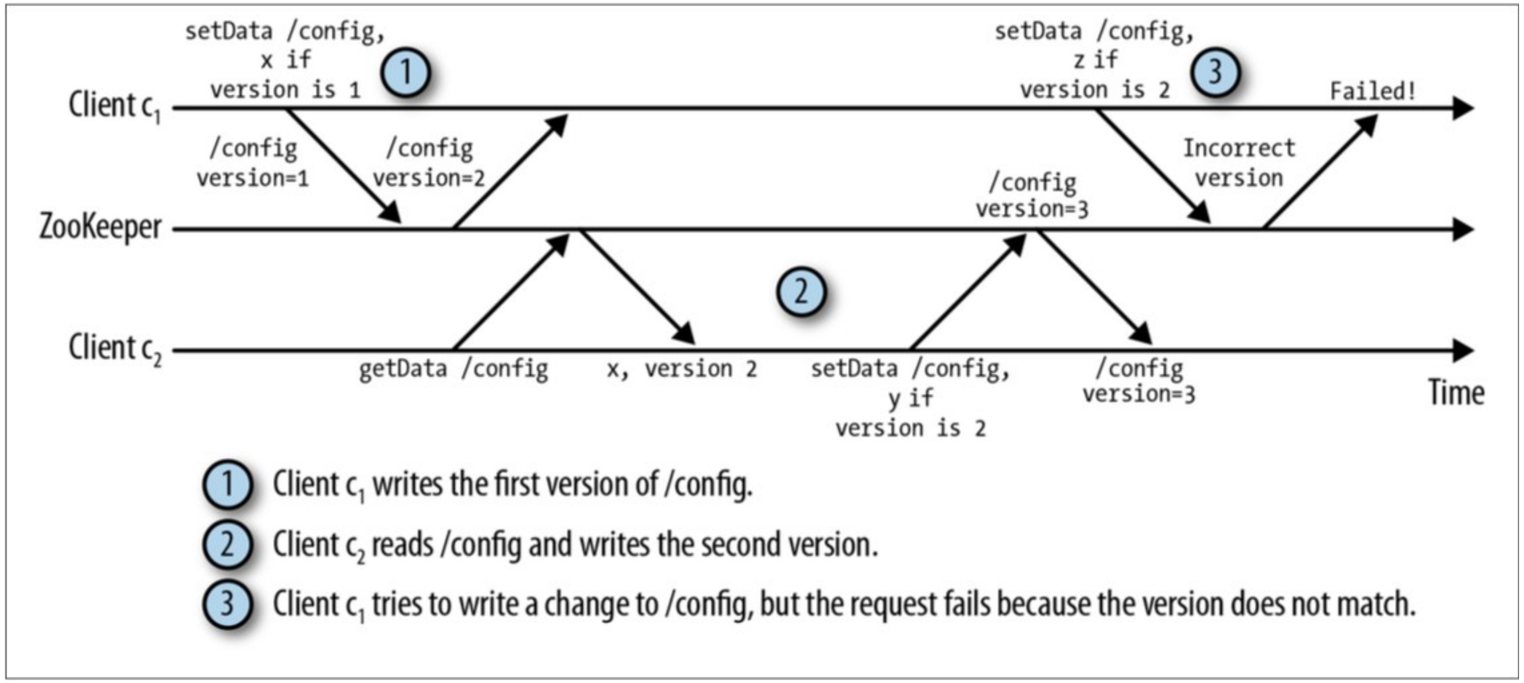
The ZooKeeper Data Model (ZDM)

- Hierarchal name space
- Each node is called as a ZNode
- Every ZNode has data (given as byte[]) and can optionally have children
- ZNode paths:
  - Canonical, absolute, slash-separated
  - No relative references
  - Names can have Unicode characters
- ZNode maintain **stat structure**



ZDM - Versions

- Eash Znode has version number, is incremented every time its data changes
- *setData* and *delete* take version as input, operation succeeds only if client's version is equal to server's one

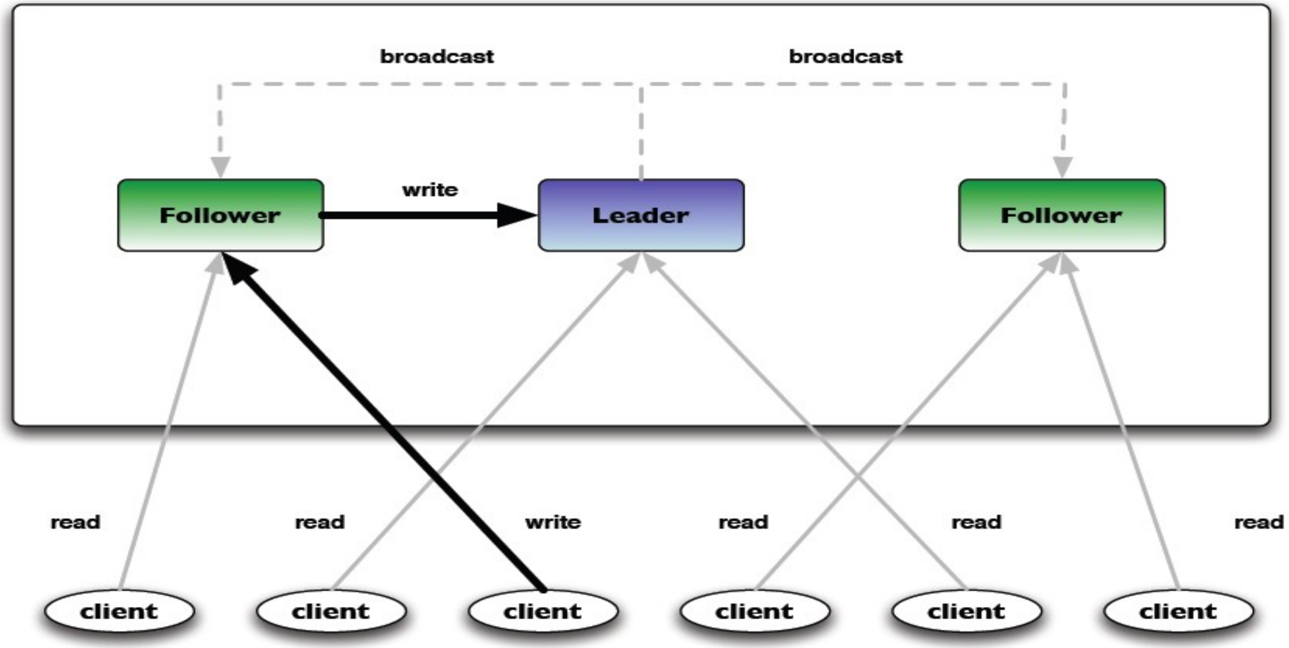


1 Client c_1 writes the first version of /config.

2 Client c_2 reads /config and writes the second version.

3 Client c_1 tries to write a change to /config, but the request fails because the version does not match.

ZDM – Znode – Reads & Writes



- Read requests are processed locally at the ZooKeeper server to which client is currently connected
- Write requests are forwarded to leader and go through majority consensus before a response is generated

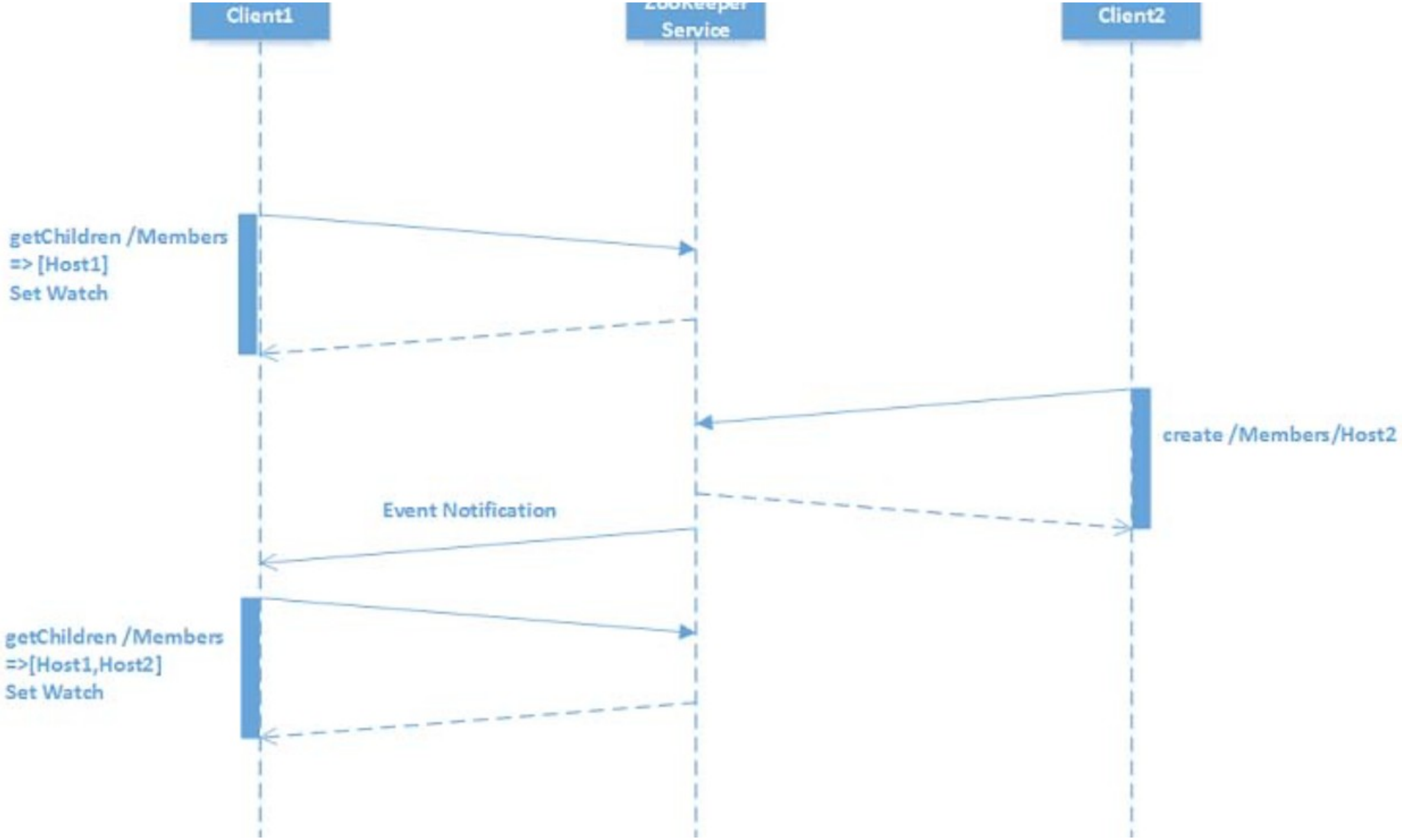
ZDM – Consistency Guarantees

- Sequential Consistency
- Atomicity
- Single System Image
- Reliability
- Timeliness (Eventual Consistency)

ZDM - Watches

- A watch event is one-time trigger, sent to client that set watch, which occurs when data for which watch was set changes.
- Watches allow clients to get notifications when a znode changes in any way (NodeChildrenChanged, NodeCreated, NodeDataChanged,NodeDeleted)
- All of read operations – **getData()**, **getChildren()**, **exists()** – have option of setting watch
- ZooKeeper Guarantees about Watches:
  - Watches are ordered, order of watch events corresponds to the order of the updates
  - A client will see a watch event for znode it is watching before seeing the new data that corresponds to that znode

ZDM – Watches (cont)



Leader Election – SID-ZXID(epoch+count)

```
2020-08-31 21:24:35,972 [myid:2] - INFO
[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):QuorumPeer@1371] – LOOKING
2020-08-31 21:24:35,975 [myid:2] - INFO
[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):FastLeaderElection@944] - New election. My id = 2,
proposed zxid=0x0
2020-08-31 21:24:35,999 [myid:2] - INFO  [WorkerReceiver[myid=2]:FastLeaderElection$Messenger$WorkerReceiver@389] -
Notification: my state:LOOKING; n.sid:2, n.state:LOOKING, n.leader:2, n.round:0x1, n.peerEpoch:0x0, n.zxid:0x0, message
format version:0x2, n.config version:0x0
2020-08-31 21:24:36,004 [myid:2] - INFO  [QuorumConnectionThread-[myid=2]-2:QuorumCnxManager@513] - Have smaller
server identifier, so dropping the connection: (myId:2 --> sid:3)
2020-08-31 21:24:36,009 [myid:2] - INFO  [WorkerReceiver[myid=2]:FastLeaderElection$Messenger$WorkerReceiver@389] -
Notification: my state:LOOKING; n.sid:3, n.state:LOOKING, n.leader:3, n.round:0x1, n.peerEpoch:0x0, n.zxid:0x0, message
format version:0x2, n.config version:0x0
2020-08-31 21:24:36,017 [myid:2] - INFO  [WorkerReceiver[myid=2]:FastLeaderElection$Messenger$WorkerReceiver@389] -
Notification: my state:LOOKING; n.sid:1, n.state:FOLLOWING, n.leader:3, n.round:0x1, n.peerEpoch:0x1, n.zxid:0x0,
message format version:0x2, n.config version:0x0
2020-08-31 21:24:36,018 [myid:2] - INFO
[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):QuorumPeer@857] - Peer state changed: following
2020-08-31 21:24:36,018 [myid:2] - INFO
[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):QuorumPeer@1453] - FOLLOWING
```

Lab : Zookeeper