Building distributed systems using Helix

http://helix.incubator.apache.org
Apache Incubation Oct, 2012
@apachehelix

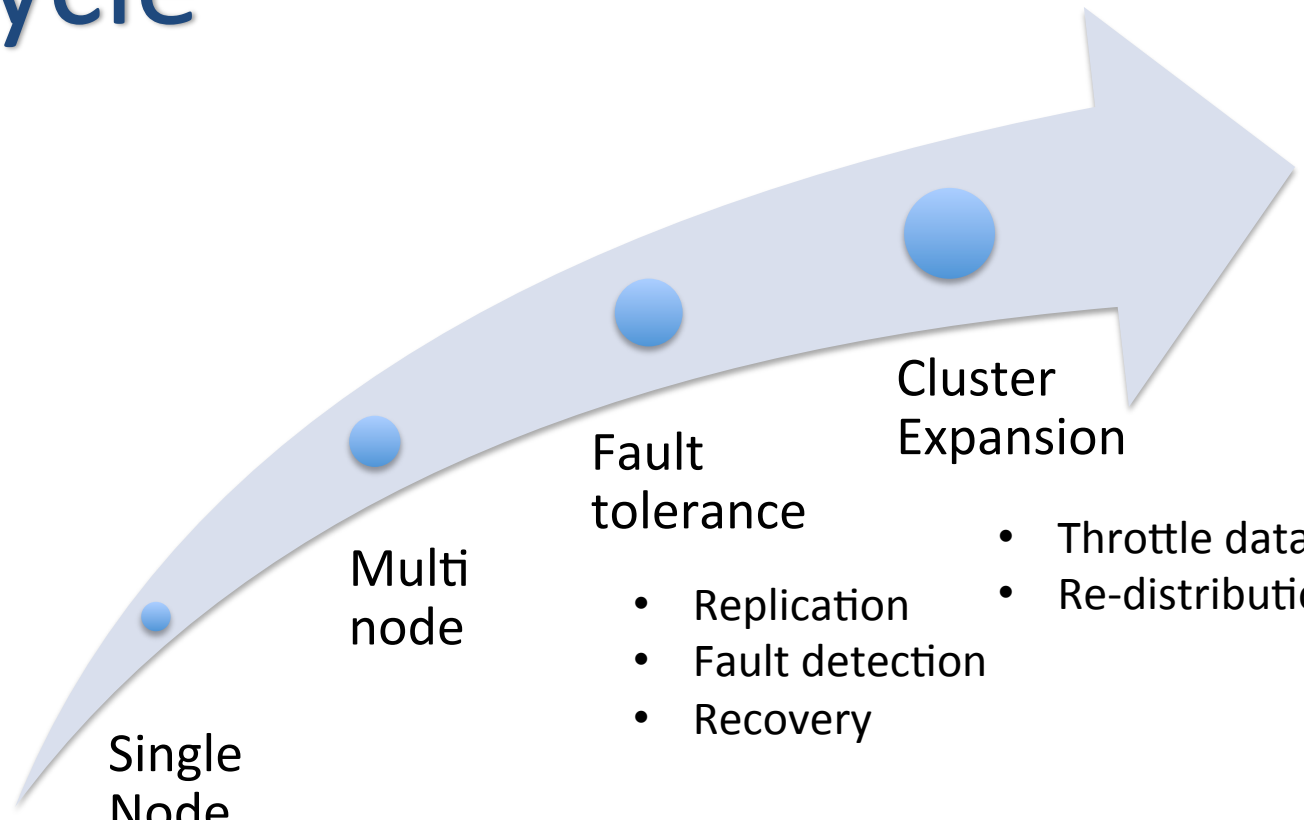
Kishore Gopalakrishna, @kishoreg1980
http://www.linkedin.com/in/kgopalak

Outline

- **Introduction**
- Architecture
- How to use Helix
- Tools
- Helix usage

Examples of distributed data systems

Lifecycle

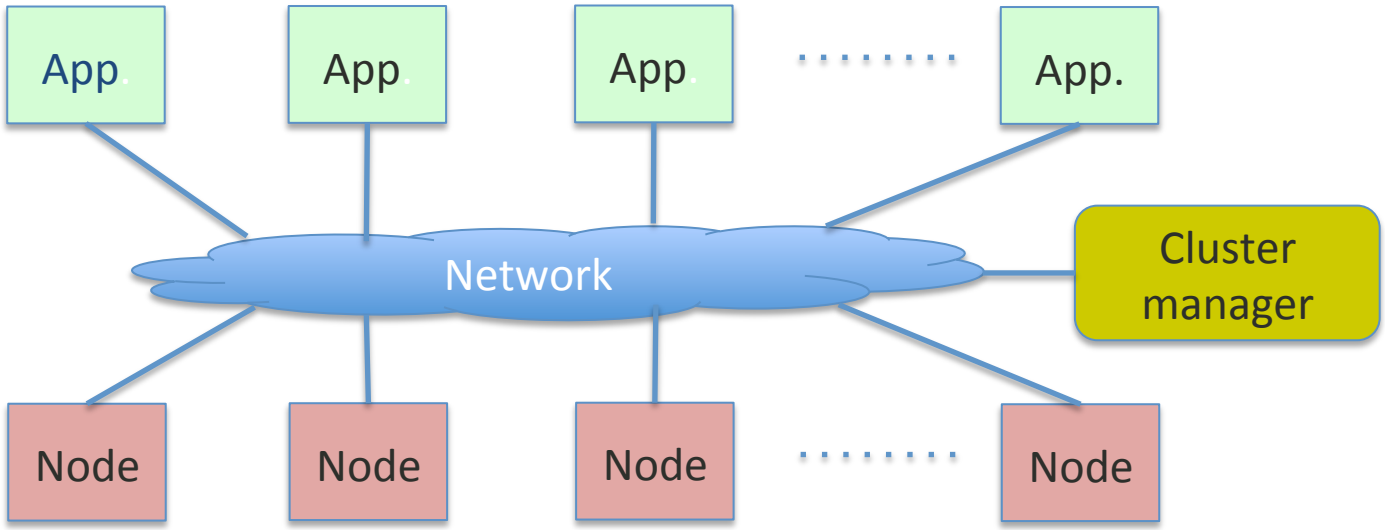


- Partitioning
- Discovery
- Co-location

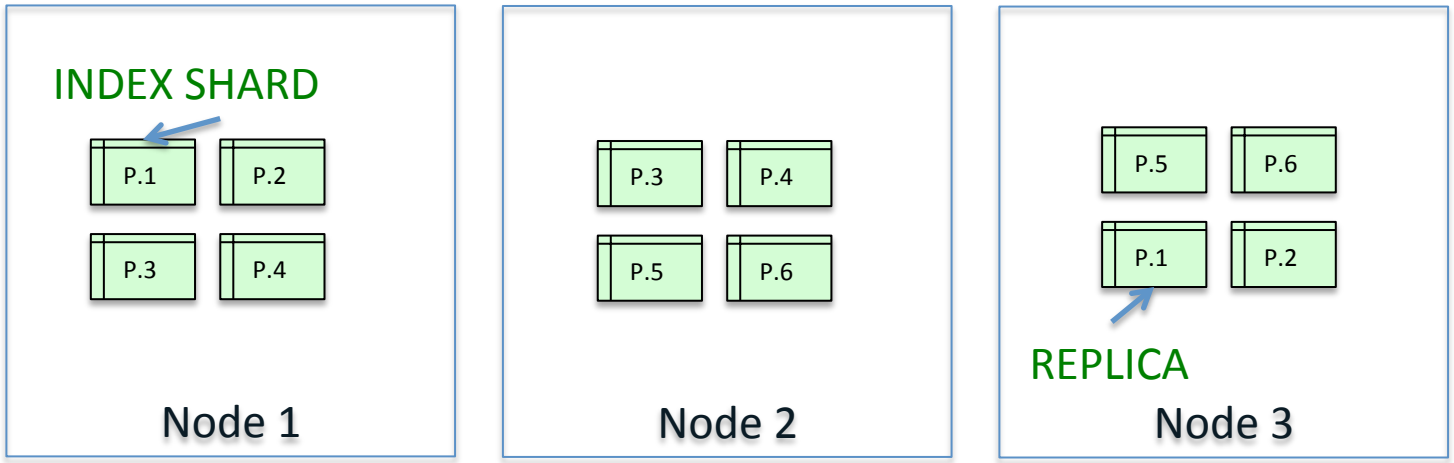
- Replication
- Fault detection
- Recovery

- Throttle data movement
- Re-distribution

Typical Architecture



Distributed search service



Partition management

- Multiple replicas
- Even distribution
- Rack aware placement

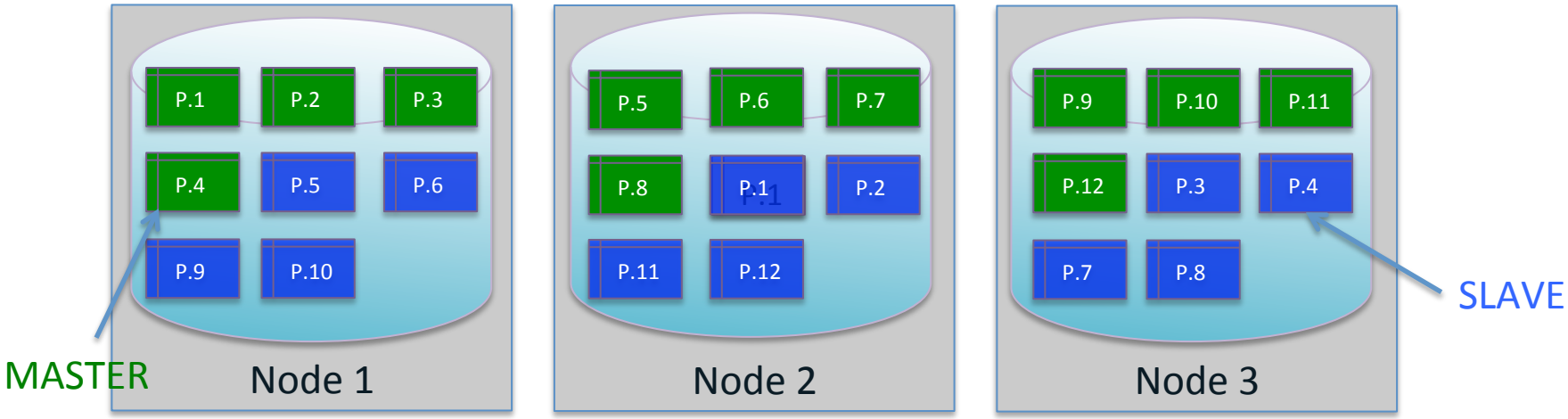
Fault tolerance

- Fault detection
- Auto create replicas
- Controlled creation of replicas

Elasticity

- re-distribute partitions
- Minimize movement
- Throttle data movement

Distributed data store



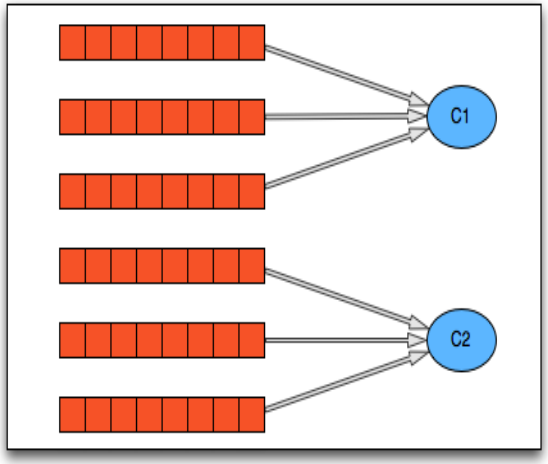
Partition management	Fault tolerance	Elasticity
• Multiple replicas • 1 designated master • Even distribution	• Fault detection • Promote slave to master • Even distribution • No SPOF	• Minimize downtime • Minimize data movement • Throttle data movement

Message consumer group

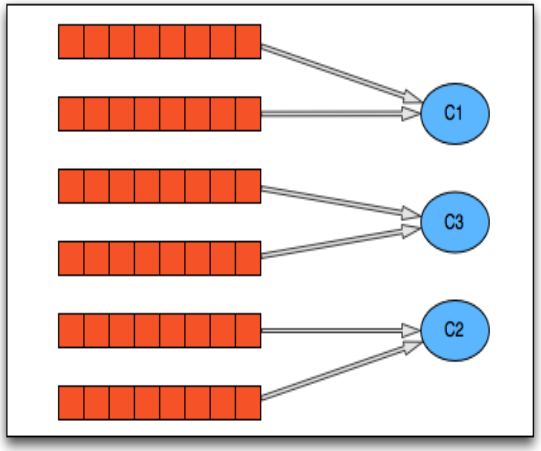
- Similar to Message groups in ActiveMQ
  - guaranteed ordering of the processing of related messages across a single queue
  - load balancing of the processing of messages across multiple consumers
  - high availability / auto-failover to other consumers if a JVM goes down
- Applicable to many messaging pub/sub systems like kafka, rabbitmq etc

Message consumer group

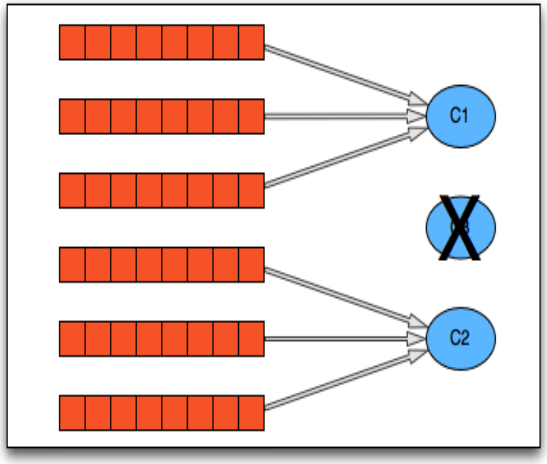
ASSIGNMENT



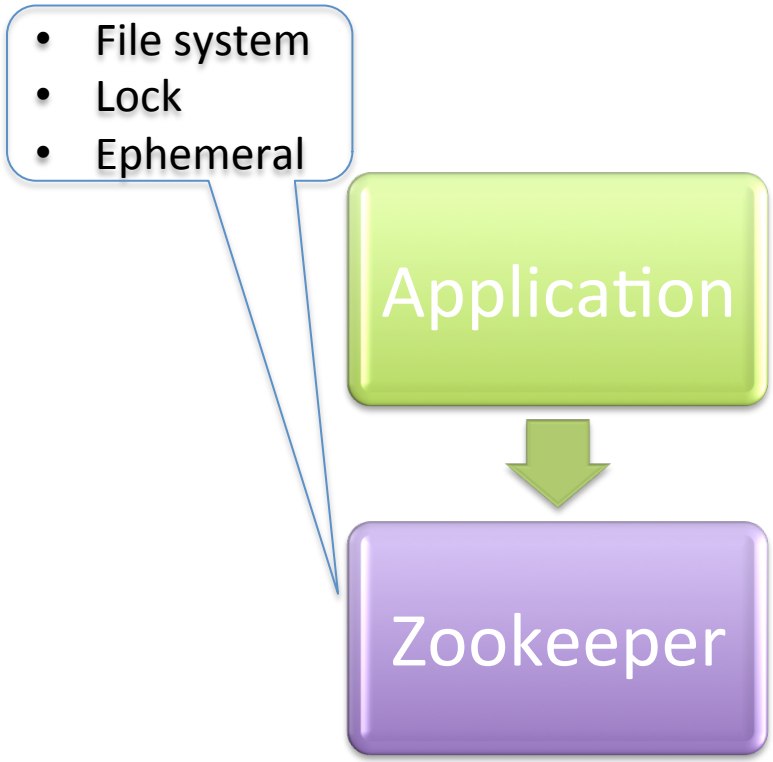
SCALING



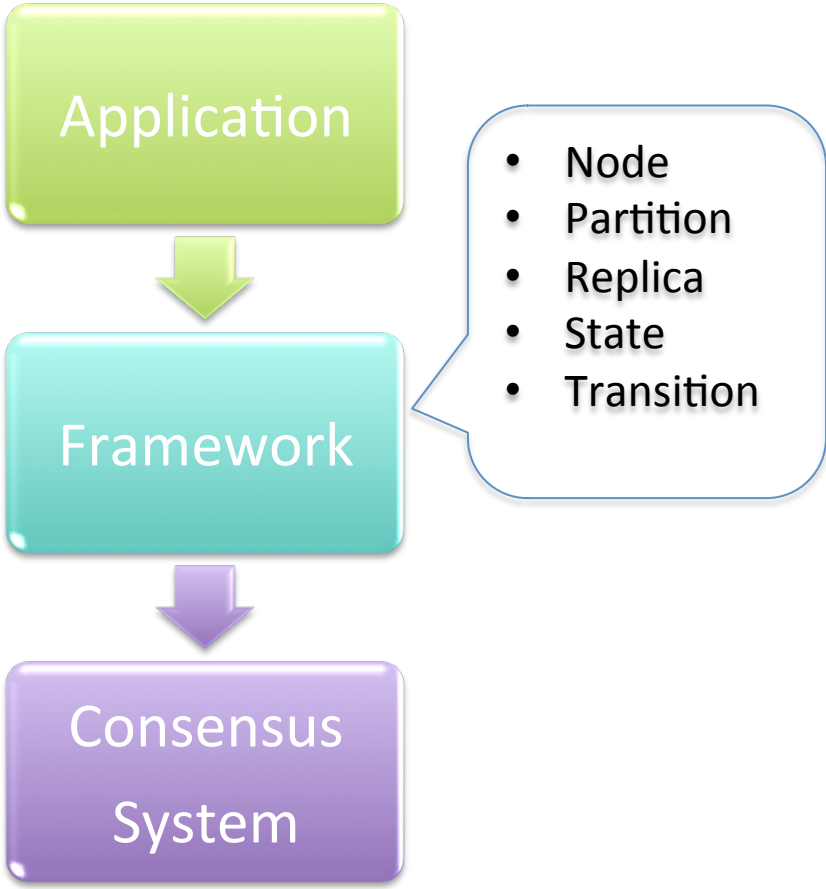
FAULT TOLERANCE



Zookeeper provides low level primitives.



We need high level primitives.




HELIX

Outline

- Introduction
- **Architecture**
- How to use Helix
- Tools
- Helix usage

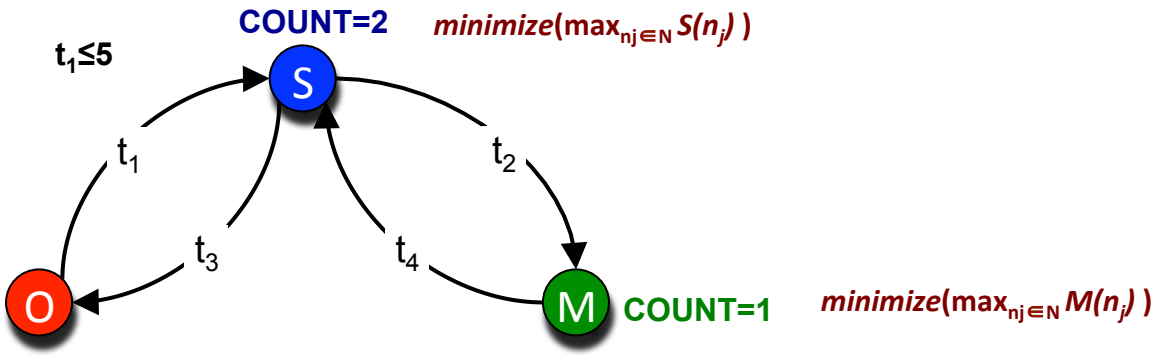
Terminologies

Node	A single machine
Cluster	Set of Nodes
Resource	***A logical entity e.g. database, index, task***
Partition	Subset of the resource.
Replica	Copy of a partition
State	Status of a partition replica, e.g Master, Slave
Transition	Action that lets replicas change status e.g Slave -> Master

Core concept

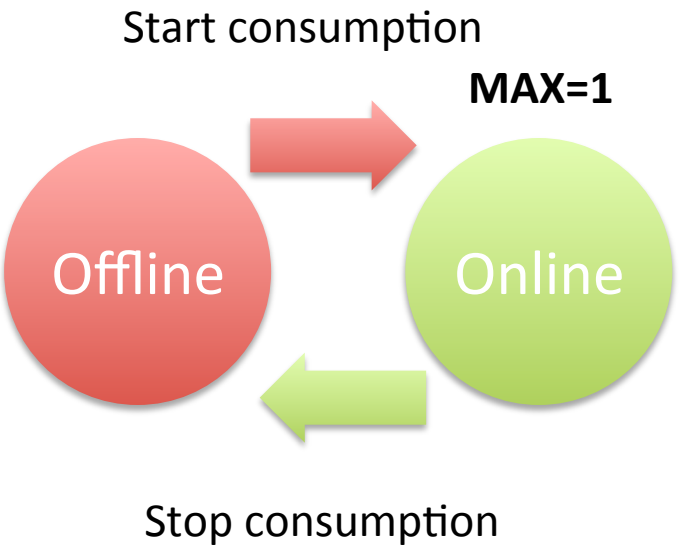
State Machine	Constraints	Objectives

• States
  • Offline, Slave, Master
• Transition
  • O->S, S->M,S->M, M->S
	• States
	  • M=1, S=2
	• Transitions
	  • concurrent(0->S) < 5
		• Partition Placement
		• Failure semantics

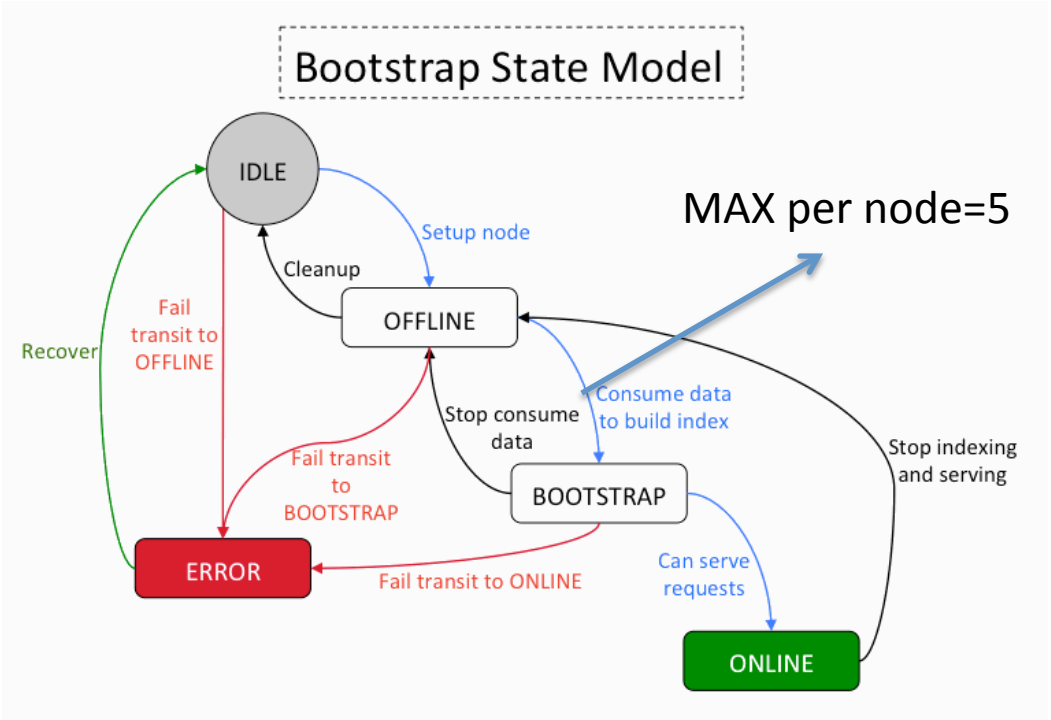


Helix solution

Message consumer group

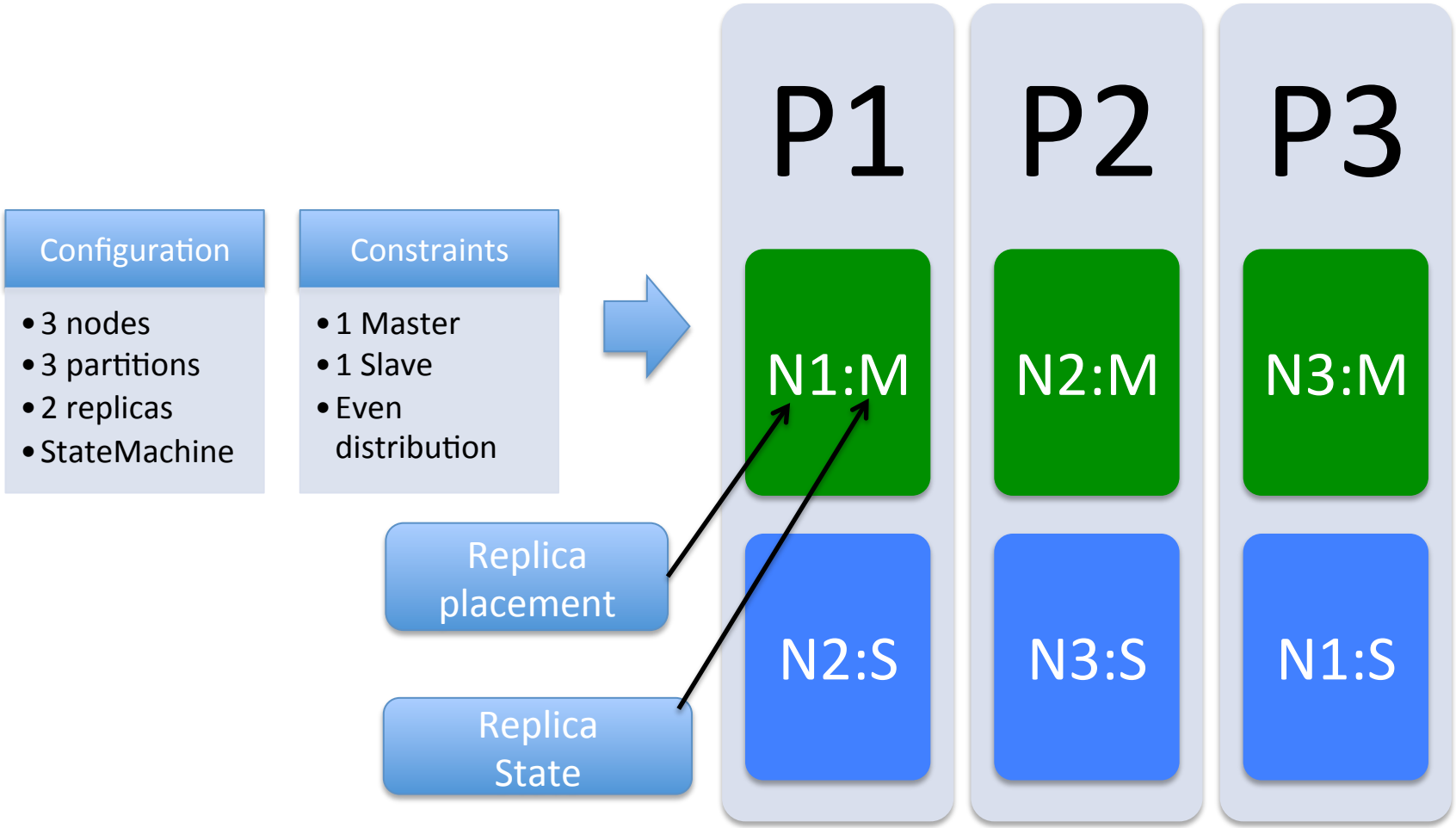


Distributed search

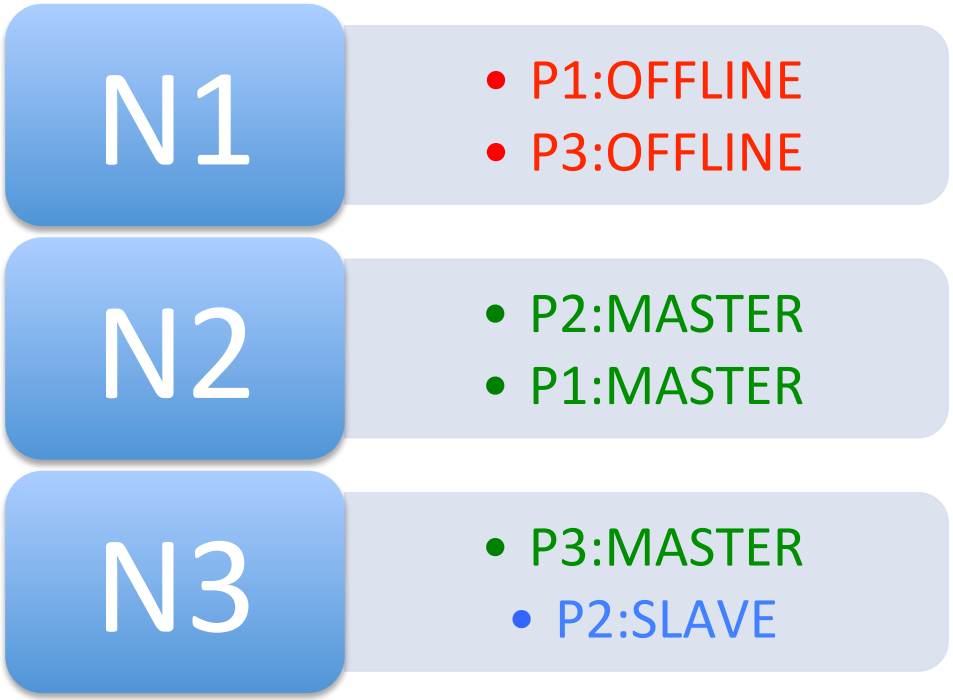


MAX=3
(number of replicas)

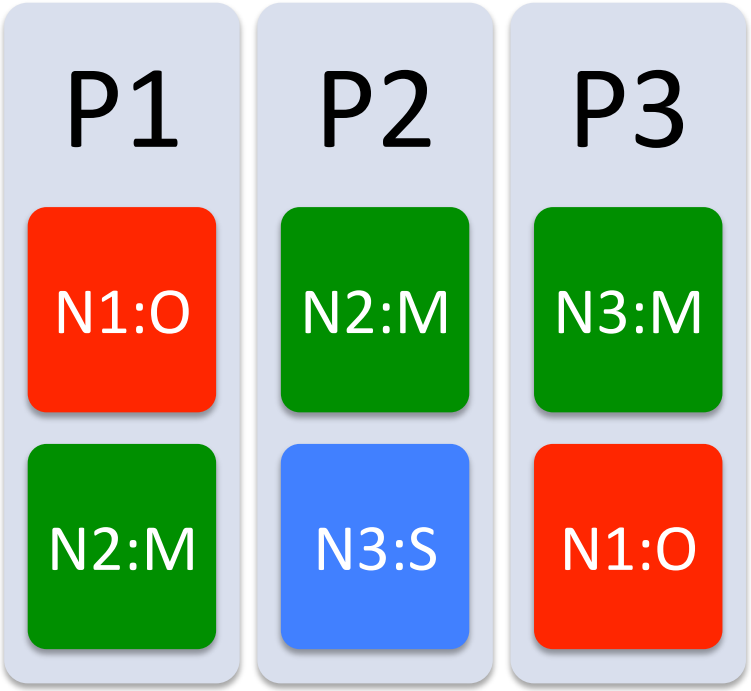
IDEALSTATE


Configuration
• 3 nodes
• 3 partitions
• 2 replicas
• StateMachine
Constraints
• 1 Master
• 1 Slave
• Even distribution
Replica placement
Replica State
P1
P2
P3
N1:M
N2:M
N3:M
N2:S
N3:S
N1:S

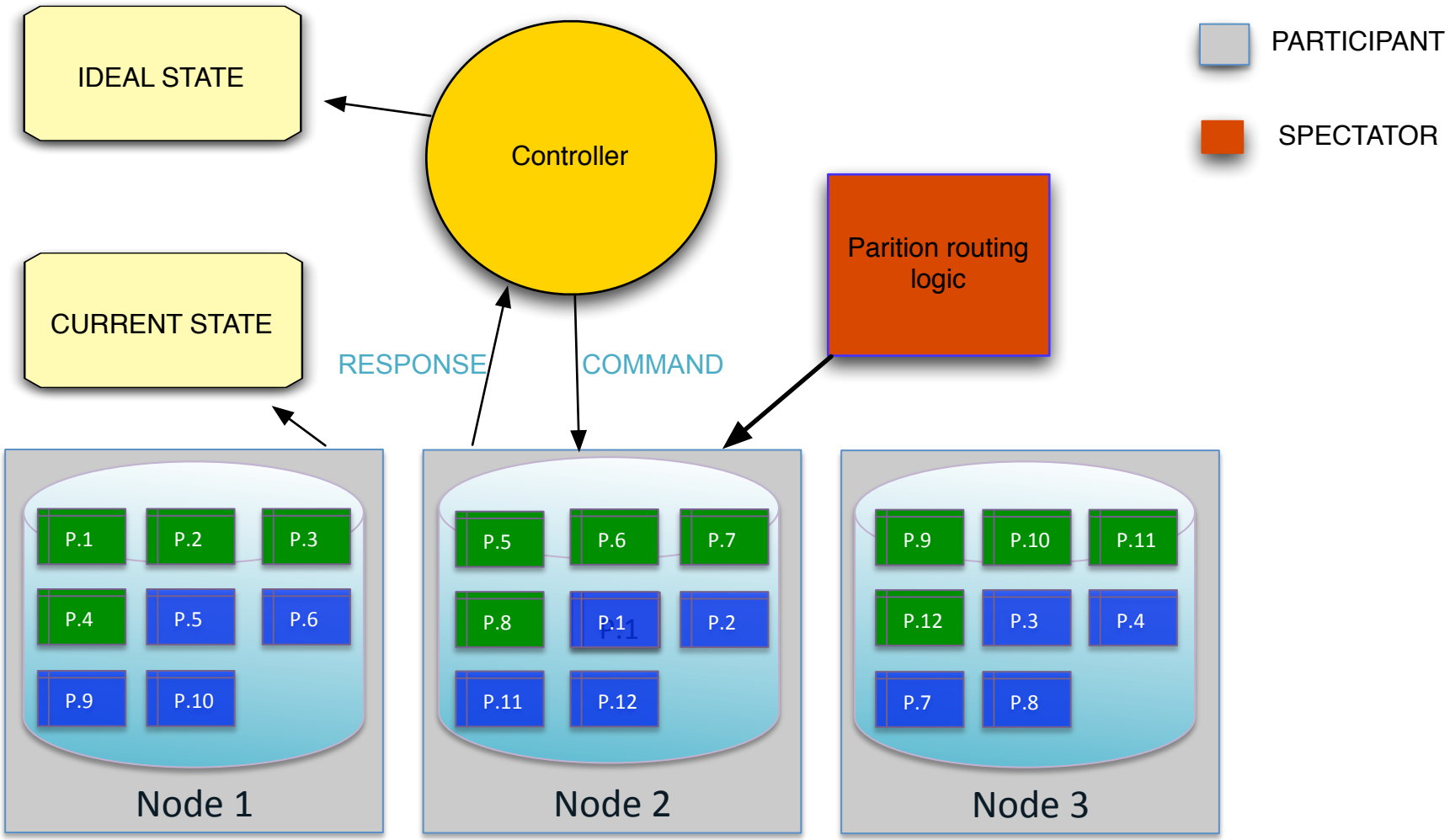
CURRENT STATE



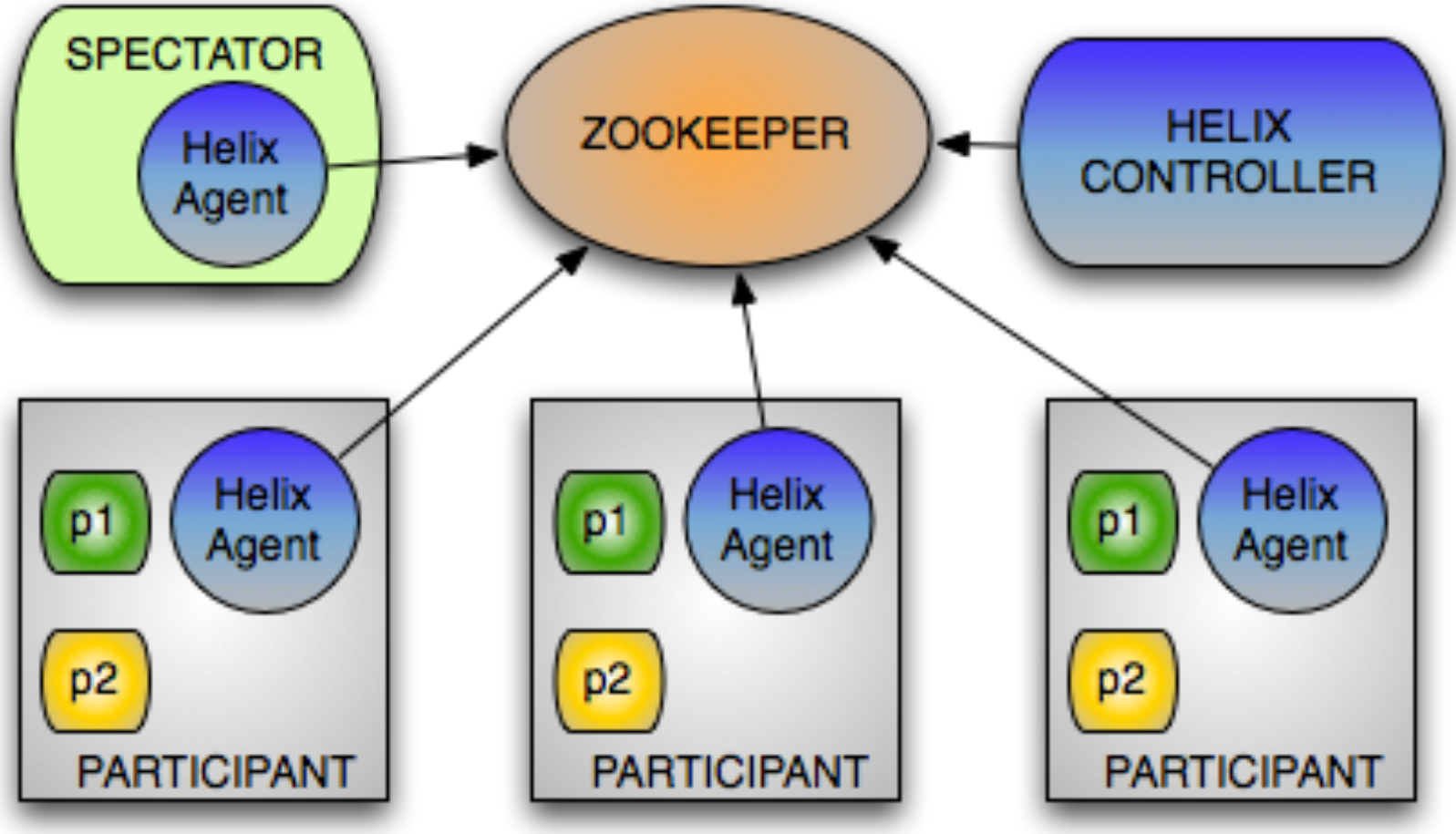
EXTERNAL VIEW



Helix Based System Roles



Logical deployment



Outline

- Introduction
- Architecture
- **How to use Helix**
- Tools
- Helix usage

Helix based solution

1. Define

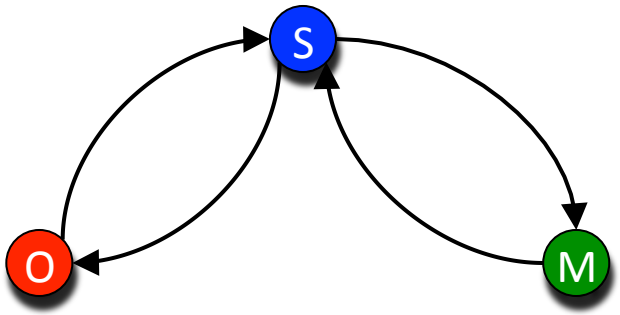
2. Configure

3. Run

Define: State model definition

- States
  - All possible states
  - Priority
- Transitions
  - Legal transitions
  - Priority
- Applicable to each partition of a resource

- e.g. MasterSlave



Define: state model

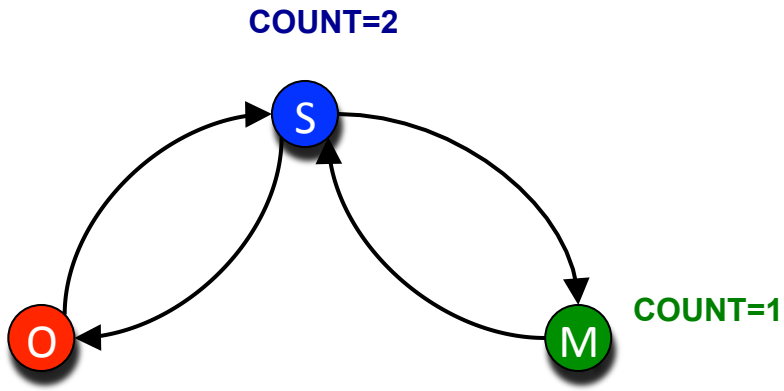
```
Builder = new StateModelDefinition.Builder("MASTERSLAVE");
// Add states and their rank to indicate priority.
builder.addState(MASTER, 1);
builder.addState(SLAVE, 2);
builder.addState(OFFLINE);

//Set the initial state when the node starts
builder.initialState(OFFLINE);
```

```
//Add transitions between the states.
builder.addTransition(OFFLINE, SLAVE);
builder.addTransition(SLAVE, OFFLINE);
builder.addTransition(SLAVE, MASTER);
builder.addTransition(MASTER, SLAVE);
```

Define: constraints

	State	Transition
Partition	Y	Y
Resource	-	Y
Node	Y	Y
Cluster	-	Y

	State	Transition
Partition	M=1,S=2	-



Define:constraints

```
// static constraint
builder.upperBound(MASTER, 1);


// dynamic constraint
builder.dynamicUpperBound(SLAVE, "R");



// Unconstrained
 builder.upperBound(OFFLINE, -1;
```

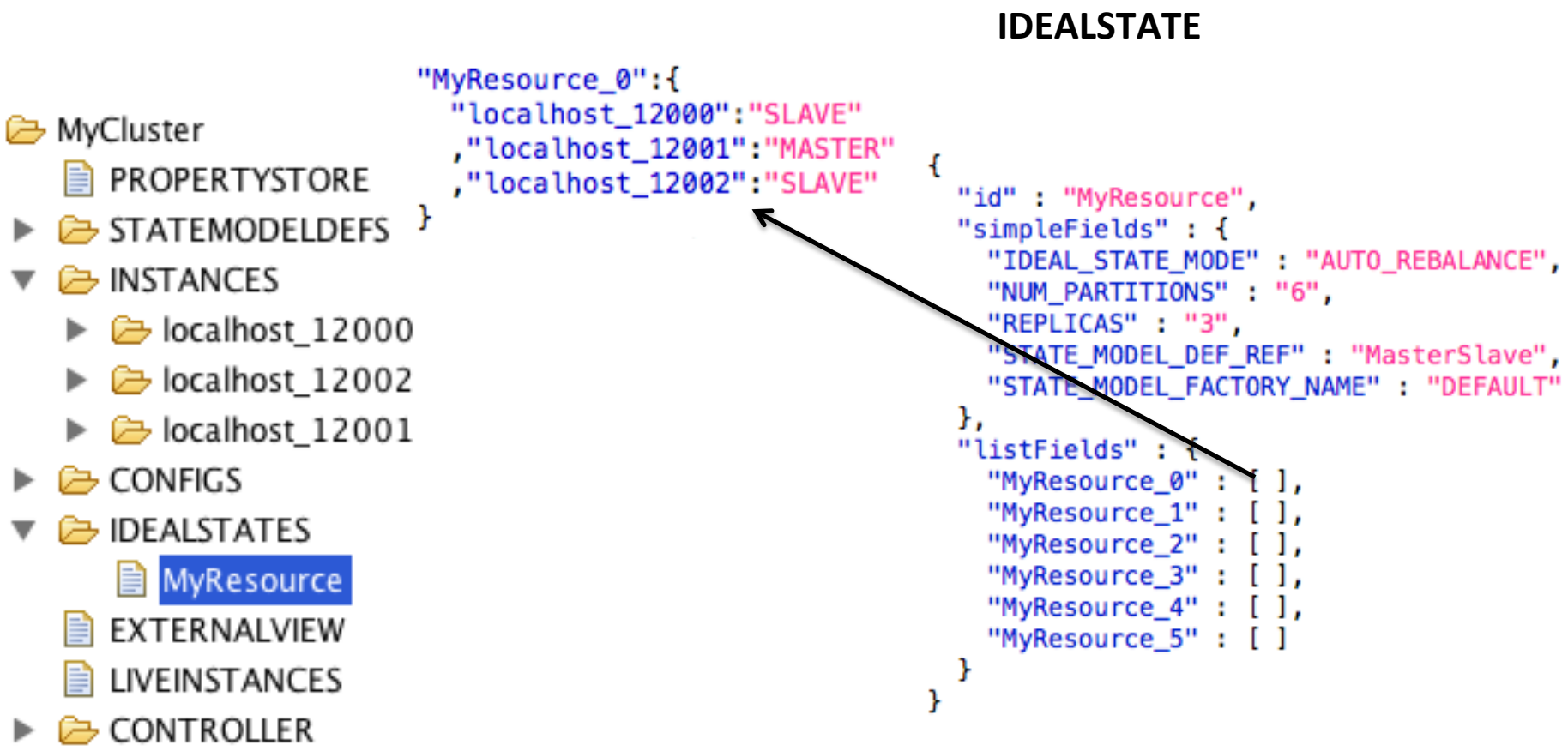
Define: participant plug-in code

```
@StateModelInfo(initialState = "OFFLINE", states = { "OFFLINE", "SLAVE", "MASTER" })
ExampleStateModel extends StateModel{
  @Transition(from = "OFFLINE", to = "SLAVE")
  public void fromOfflineToSlave(Message m, NotificationContext context)
  {
    log("Transitioning "+ m.partition + " from:"+ m.from +" to:"+ m.to);
    //Application specific code goes here
  }

  @Transition(from = "SLAVE", to = "MASTER")
  public void fromSlaveToMaster(Message m, NotificationContext context)
  {}

  @Transition(from = "SLAVE", to = "OFFLINE")
  public void fromSlaveToOffine(Message m, NotificationContext context)
  {}

  @Transition(from = "MASTER", to = "SLAVE")
  public void fromMasterToSlave(Message m, NotificationContext context)
  {}
}
```

Step 2: configure

helix-admin –zkSvr <zkAddress>
CREATE CLUSTER
--addCluster <clusterName>
ADD NODE
--addNode <clusterName instanceId(host:port)>
CONFIGURE RESOURCE
--addResource <clusterName resourceName partitions statemodel>
REBALANCE ➔SET IDEALSTATE
--rebalance <clusterName resourceName replicas>

zookeeper view



Step 3: Run

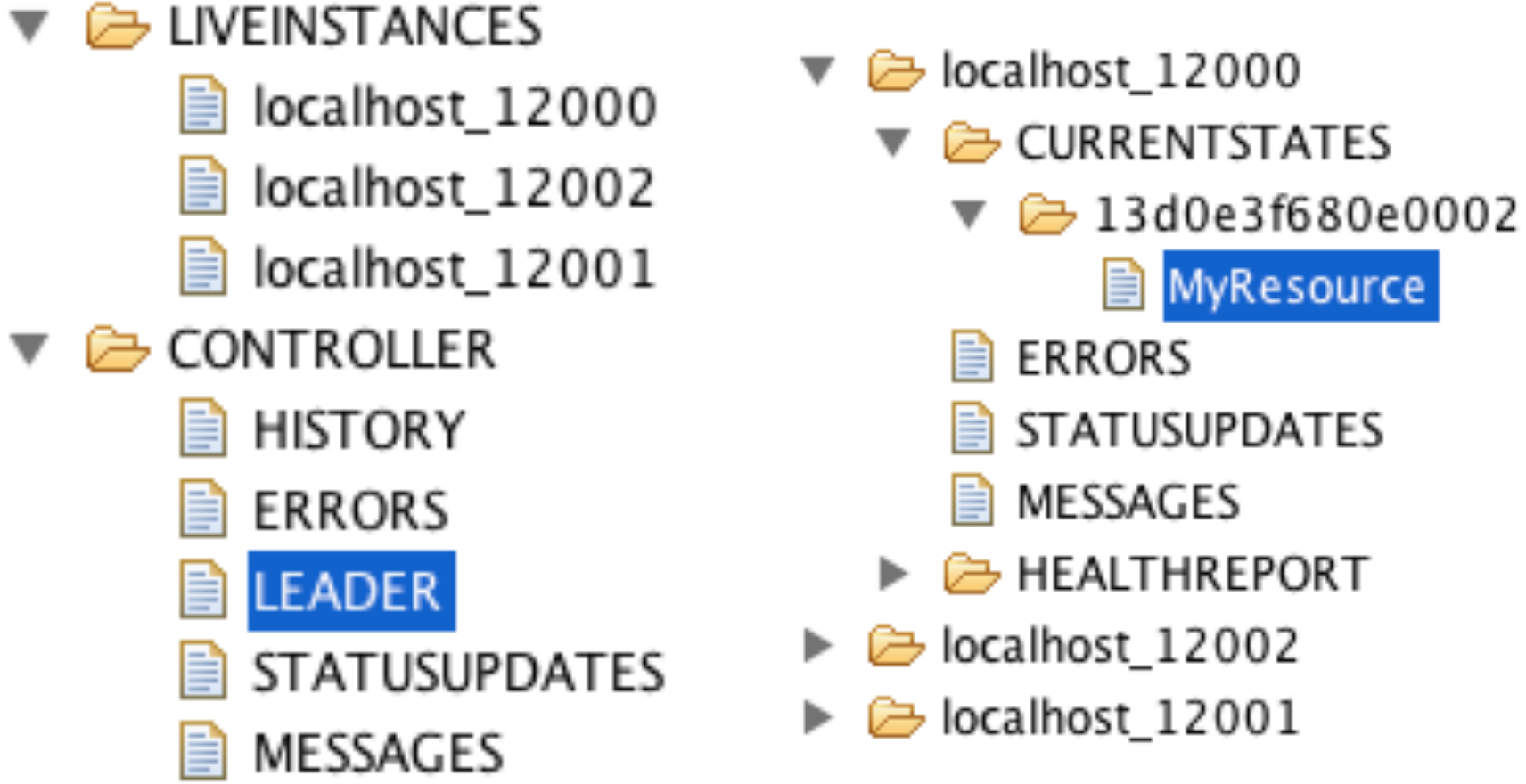
START CONTROLLER

run-helix-controller -zkSvr localhost:2181 –cluster MyCluster

START PARTICIPANT

```
manager = HelixManagerFactory.getZKHelixManager(clusterName, instanceName,
    InstanceType.PARTICIPANT, zkConnectString);
stateModelFactory = new MasterSlaveStateModelFactory();
StateMachineEngine stateMachine = manager.getStateMachineEngine();
stateMachine.registerStateModelFactory("MasterSlave", stateModelFactory);
manager.connect();
```

zookeeper view



Znode content

CURRENT STATE


```
{
  "id":"MyResource"
  ,"simpleFields":{
    ,"SESSION_ID":"13d0e34675e0002"
    ,"INSTANCE_NAME":"localhost:12000"
    ,"STATE_MODEL_DEF":"MasterSlave"
  }
  ,"mapFields":{
    "MyResource_0":{
      "CURRENT_STATE":"SLAVE"
    }
    ,"MyResource_1":{
      "CURRENT_STATE":"MASTER"
    }
    ,"MyResource_2":{
      "CURRENT_STATE":"MASTER"
    }
  }
}
```


EXTERNAL VIEW

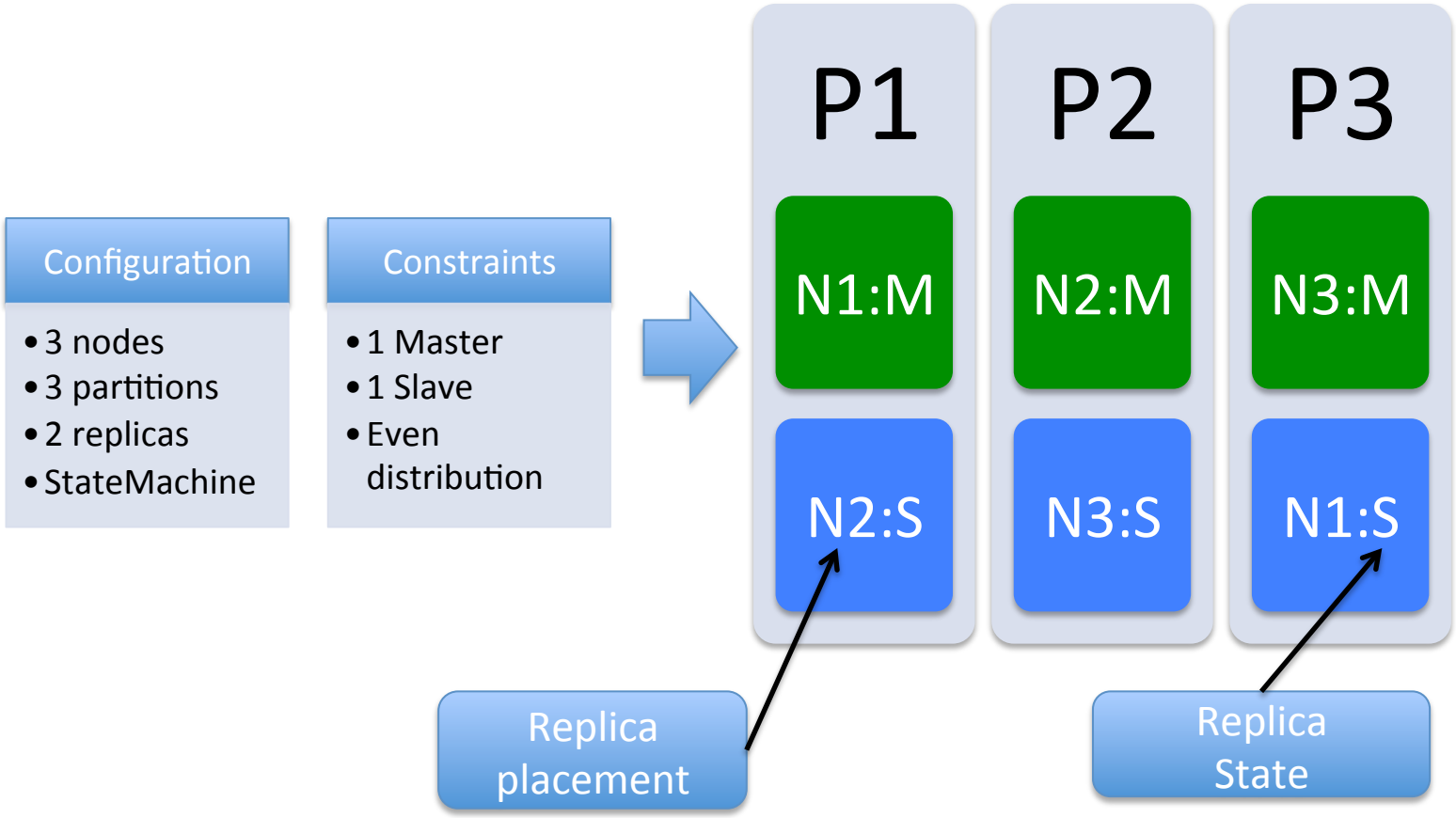

```
{
  "id":"MyResource"
  ,"simpleFields":{
  }
  ,"listFields":{
  }
  ,"mapFields":{
    "MyResource_0":{
      "localhost_12000":"SLAVE"
      ,"localhost_12001":"MASTER"
      ,"localhost_12002":"SLAVE"
    }
    ,"MyResource_1":{
      "localhost_12000":"MASTER"
      ,"localhost_12001":"SLAVE"
      ,"localhost_12002":"SLAVE"
    }
    ,"MyResource_2":{
      "localhost_12000":"MASTER"
      ,"localhost_12001":"SLAVE"
      ,"localhost_12002":"SLAVE"
    }
  }
}
```

Spectator Plug-in code

```
class RoutingLogic{
   public void write(Request request){

     partition = getPartition(request.key);

     List<Node> nodes = routingTableProvider.getInstances(partition, "Master");

     nodes.get(0).write(request);

   }

  public void read(Request request){

    partition = getPartition(request.key);

     List<Node> nodes = routingTableProvider.getInstances(partition);

     random(nodes).read(request);
   }
}
```

Helix Execution modes

IDEALSTATE


Configuration
• 3 nodes
• 3 partitions
• 2 replicas
• StateMachine
Constraints
• 1 Master
• 1 Slave
• Even distribution
P1
P2
P3
N1:M
N2:M
N3:M
N2:S
N3:S
N1:S
Replica placement
Replica State

Execution modes

- Who controls what

	AUTO REBALANCE	AUTO	CUSTOM
Replica placement	Helix	App	App
Replica State	Helix	Helix	App

Auto rebalance v/s Auto

AUTO REBALANCE

```
{
  "id" : "MyResource",
  "simpleFields" : {
    "IDEAL_STATE_MODE" : "AUTO_REBALANCE",
    "NUM_PARTITIONS" : "3",
    "REPLICAS" : "1",
    "STATE_MODEL_DEF_REF" : "OnlineOffline",
  }
  "listFields" : {
    "MyResource_0" : [],
    "MyResource_1" : [],
    "MyResource_2" : []
  },
  "mapFields" : {
  }
}
```

AUTO

```
{
  "id" : "MyResource",
  "simpleFields" : {
    "IDEAL_STATE_MODE" : "AUTO",
    "NUM_PARTITIONS" : "3",
    "REPLICAS" : "2",
    "STATE_MODEL_DEF_REF" : "MasterSlave",
  }
  "listFields" : {
    "MyResource_0" : [node1, node2],
    "MyResource_1" : [node2, node3],
    "MyResource_2" : [node3, node1]
  },
  "mapFields" : {
  }
}
```

In action

Auto rebalance
MasterSlave p=3 r=2 N=3

Node1	Node2	Node3
P1:M	P2:M	P3:M
P2:S	P3:S	P1:S

On failure: Auto create replica and assign state

Node 1	Node 2	Node 3
P1:O	P2:M	P3:M
P2:O	P3:S	P1:S
	P1:M	P2:S

Auto
MasterSlave p=3 r=2 N=3

Node 1	Node 2	Node 3
P1:M	P2:M	P3:M
P2:S	P3:S	P1:S

On failure: Only change states to satisfy constraint

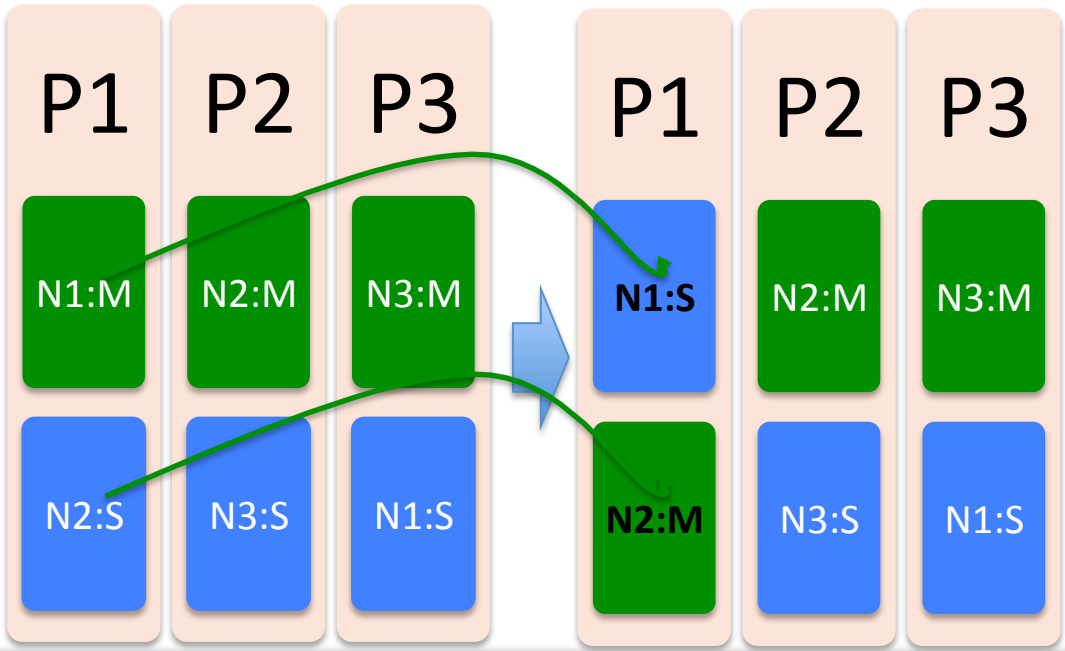
Node 1	Node 2	Node 3
P1:M	P2:M	P3:M
P2:S	P3:S	P1:**M**

Custom mode: example

```
{
  "id" : "MyResource",
  "simpleFields" : {
    "IDEAL_STATE_MODE" : "CUSTOM",
    "NUM_PARTITIONS" : "3",
    "REPLICAS" : "3",
    "STATE_MODEL_DEF_REF" : "MasterSlave",
  }
 "mapFields" : {
    "MyResource_0" : {
      "node1" : "MASTER",
      "node2" : "SLAVE",
      "node3" : "SLAVE",
    },
    "MyResource_1" : {
      "node1" : "MASTER",
      "node2" : "SLAVE",
      "node3" : "SLAVE",
    },
    "MyResource_2" : {
      "node1" : "MASTER",
      "node2" : "SLAVE",
      "node3" : "SLAVE",
    }
  }
}
```

Custom mode: handling failure

- Custom code invoker
  - Code that lives on all nodes, but active in one place
  - Invoked when node joins/leaves the cluster
  - Computes new idealstate
  - Helix controller fires the transition **without violating constraints**



Transitions		
1	N1	M→S
2	N2	S→ M

1 & 2 in parallel violate single master constraint

Helix sends 2 after 1 is finished

Outline

- Introduction
- Architecture
- How to use Helix
- **Tools**
- Helix usage

Tools

- Chaos monkey
- Data driven testing and debugging
- Rolling upgrade
- On demand task scheduling and intra-cluster messaging
- Health monitoring and alerts

Data driven testing

- Instrument –
  - Zookeeper, controller, participant logs
- Simulate – Chaos monkey
- Analyze – Invariants are
  - Respect state transition constraints
  - Respect state count constraints
  - And so on
- Debugging made easy
  - Reproduce exact sequence of events

Structured Log File - sample

timestamp	partition	instanceName	sessionId	state
1323312236368	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236426	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236530	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236530	TestDB_91	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236561	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	SLAVE
1323312236561	TestDB_91	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236685	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	SLAVE
1323312236685	TestDB_91	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236685	TestDB_60	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236719	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	SLAVE
1323312236719	TestDB_91	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	SLAVE
1323312236719	TestDB_60	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	OFFLINE
1323312236814	TestDB_123	express1-md_16918	ef172fe9-09ca-4d77b05e-15a414478ccc	SLAVE

No more than R=2 slaves

Time	State	Number Slaves	Instance
42632	OFFLINE	0	10.117.58.247_12918
42796	SLAVE	1	10.117.58.247_12918
43124	OFFLINE	1	10.202.187.155_12918
43131	OFFLINE	1	10.220.225.153_12918
43275	SLAVE	2	10.220.225.153_12918
43323	SLAVE	**3**	10.202.187.155_12918
85795	MASTER	2	10.220.225.153_12918

How long was it out of whack?

Number of Slaves	Time	Percentage
0	1082319	0.5
1	35578388	16.46
2	179417802	82.99
3	118863	0.05

83% of the time, there were 2 slaves to a partition
93% of the time, there was 1 master to a partition

Number of Masters	Time	Percentage
0	15490456	7.164960359
1	200706916	92.83503964

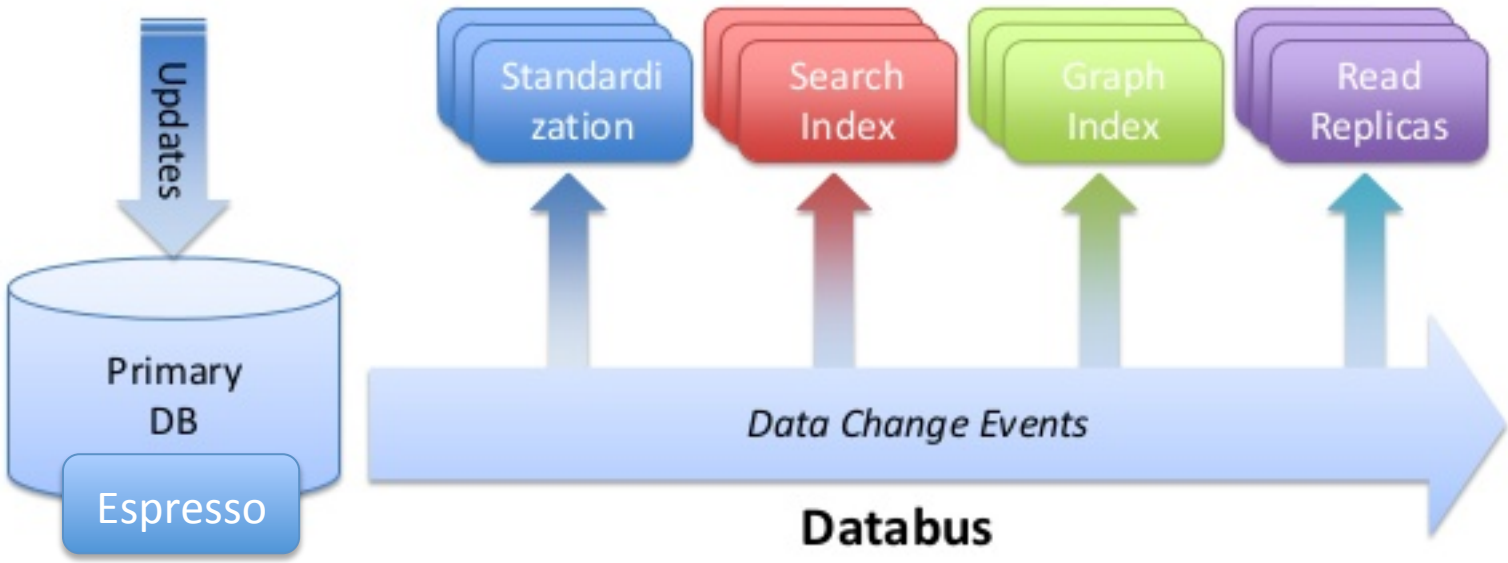
Invariant 2: State Transitions

FROM	TO	COUNT
MASTER	SLAVE	55
OFFLINE	DROPPED	0
OFFLINE	SLAVE	298
SLAVE	MASTER	155
SLAVE	OFFLINE	0

Outline

- Introduction
- Architecture
- How to use Helix
- Tools
- **Helix usage**

Helix usage at LinkedIn

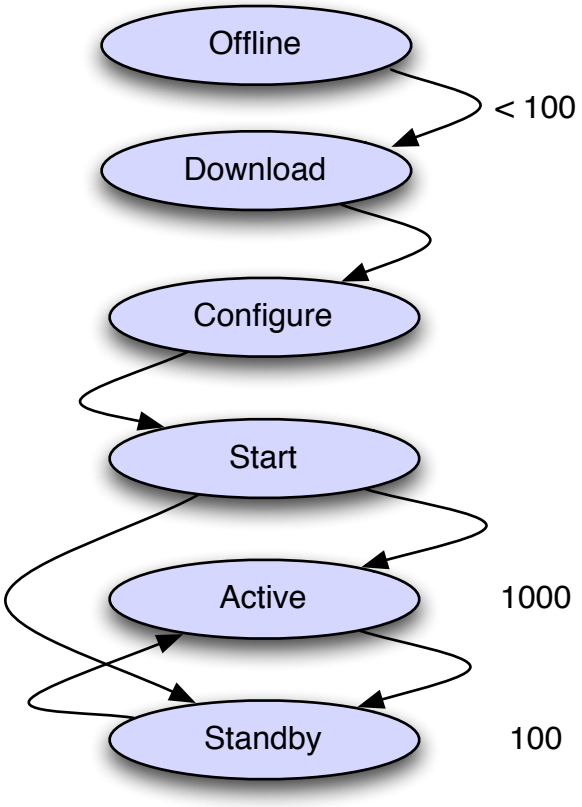


In flight

- Apache S4
  - Partitioning, co-location
  - Dynamic cluster expansion
- Archiva
  - Partitioned replicated file store
  - Rsync based replication
- Others in evaluation
  - Bigtop

Auto scaling software deployment tool

- States
  - Download, Configure, Start
  - Active, Standby
- Constraint for each state
  - Download < 100
  - Active 1000
  - Standby 100



Summary

- Helix: A Generic framework for building distributed systems
- Modifying/enhancing system behavior is easy
  - Abstraction and modularity is key
- Simple programming model: declarative state machine

Roadmap

- Features
  - Span multiple data centers
  - Automatic Load balancing
  - Distributed health monitoring
  - YARN Generic Application master for real time Apps
  - Stand alone Helix agent

website	http://helix.incubator.apache.org
user	user@helix.incubator.apache.org
dev	dev@helix.incubator.apache.org
twitter	@apachehelix, @kishoreg1980