Above the Clouds:
Clustered Akka

Jonas Bonér
CTO Typesafe
Twitter: @jboner

The problem

It is way too hard to build:

1. correct highly concurrent systems
2. truly scalable systems
3. fault-tolerant systems that self-heals

...using "state-of-the-art" tools

Introducing

akka

Vision

Simpler

- Concurrency
- Scalability
- Fault-tolerance

Vision

...with a single unified

- Programming model
- Runtime service

Manage system overload

Scale up & Scale out

Replicate and distribute for fault-tolerance

Transparent load balancing

So far...

Remote Actors

Remote Server

```
// use host & port in config
Actor.remote.start()

Actor.remote.start("localhost", 2552)
```

Scalable implementation based on
NIO (Netty) & Protobuf

Remote actor

```
import Actor._

remote register (“service:id”, actorOf[MyService])
```

server part

Remote actor

```
val service = remote actorFor (
  “service:id”,
  “darkstar”,
  9999)

service ! message
```

client part

So...what's the problem?

Does not meet the vision

Deployment (local vs remote) is a dev decision
We get a fixed and hard-coded topology
Can't change it dynamically and adaptively

Needs to be a
deployment & runtime decision

Introducing...

Clustered Actors

Address

```
val actor = actorOf[MyActor]
```

Bind the actor to a virtual address

Address

```
val actor = actorOf[MyActor](“my-service”)
```

Bind the actor to a virtual address

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Type of load-balancing

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Type of load-balancing

Nr of replicas in cluster

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Type of load-balancing

Stateless

Nr of replicas in cluster

Deployment

Deployment

- Actor address is virtual and decoupled from how it is deployed

Deployment

- Actor address is virtual and decoupled from how it is deployed
- If no deployment configuration exists then actor is deployed as local

Deployment

- Actor address is virtual and decoupled from how it is deployed
- If no deployment configuration exists then actor is deployed as local
- The same system can be configured as distributed without code change (even change at runtime)

Deployment

- Actor address is virtual and decoupled from how it is deployed
- If no deployment configuration exists then actor is deployed as local
- The same system can be configured as distributed without code change (even change at runtime)
- Write as local but deploy as distributed in the cloud without code change

Deployment

- Actor address is virtual and decoupled from how it is deployed
- If no deployment configuration exists then actor is deployed as local
- The same system can be configured as distributed without code change (even change at runtime)
- Write as local but deploy as distributed in the cloud without code change
- Allows runtime to dynamically and adaptively change topology

The runtime provides

The runtime provides

- Subscription-based cluster membership service

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing
- Leader election

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing
- Leader election
- Durable mailboxes - guaranteed delivery

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing
- Leader election
- Durable mailboxes - guaranteed delivery
- Highly available centralized configuration service

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing
- Leader election
- Durable mailboxes - guaranteed delivery
- Highly available centralized configuration service
- ...and more

Clustering of Stateless Actor

Akka Node

Akka Node

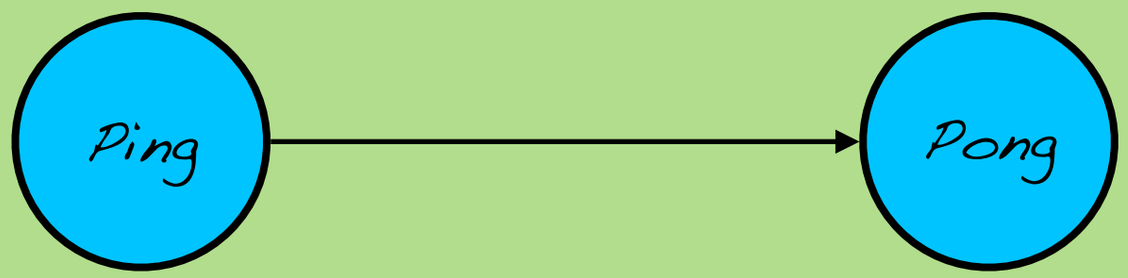
```
val ping = actorOf[Ping](“ping”)
val pong = actorOf[Pong](“pong”)

ping ! Ball(pong)
```

Akka Node

```
val ping = actorOf[Ping]("ping")
val pong = actorOf[Pong]("pong")

ping ! Ball(pong)
```



Akka
Cluster Node

Ping

Pong

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Ping

Pong

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Ping

Pong

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

```
akka {
  actor {
    deployment {
      ping {}
      pong {
        router = "round-robin"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

Ping

Pong

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Ping

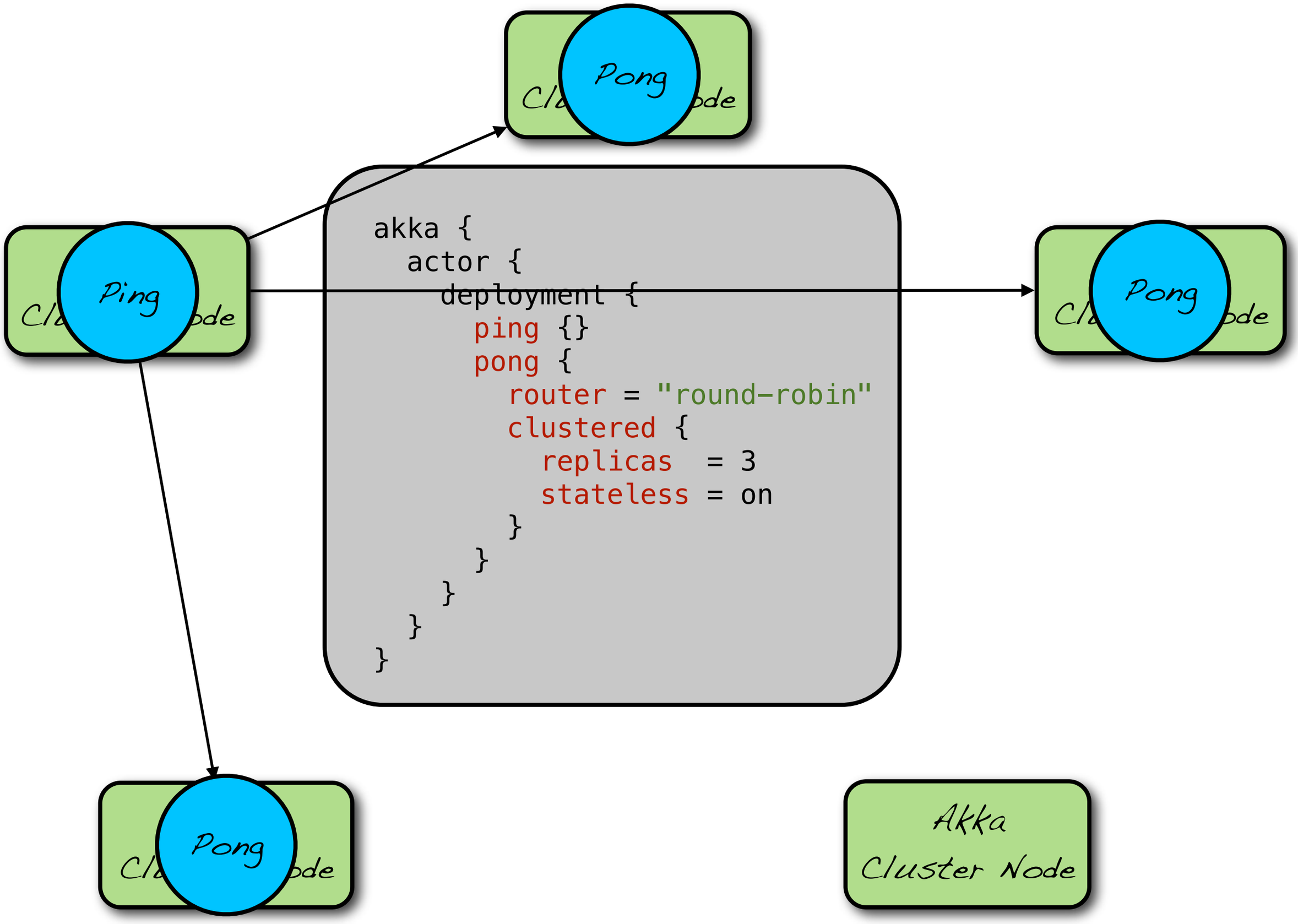
```
akka {
  actor {
    deployment {
      ping {}
      pong {
        router = "round-robin"
        clustered {
          replicas  = 3
          stateless = on
        }
      }
    }
  }
}
```

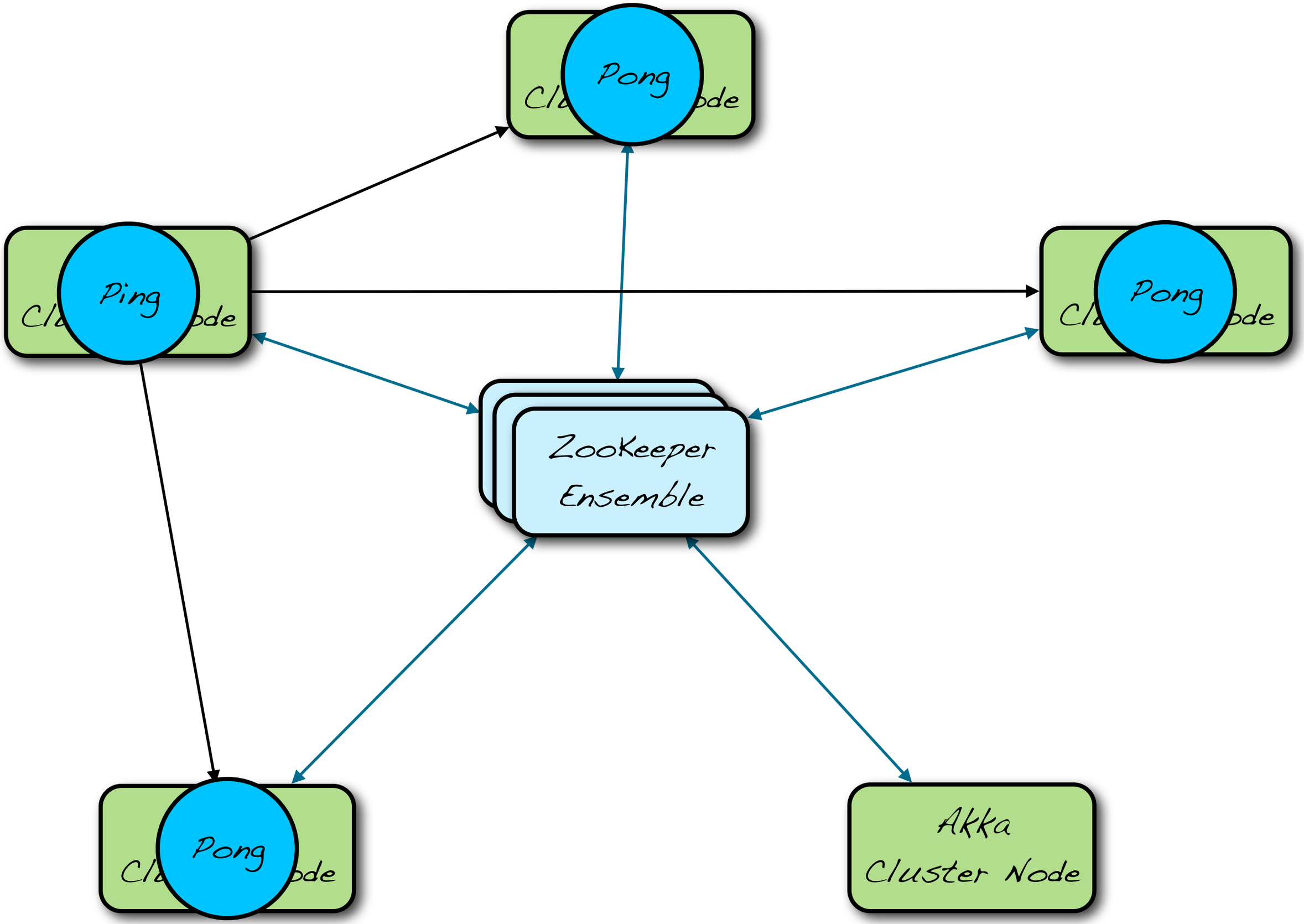
Pong

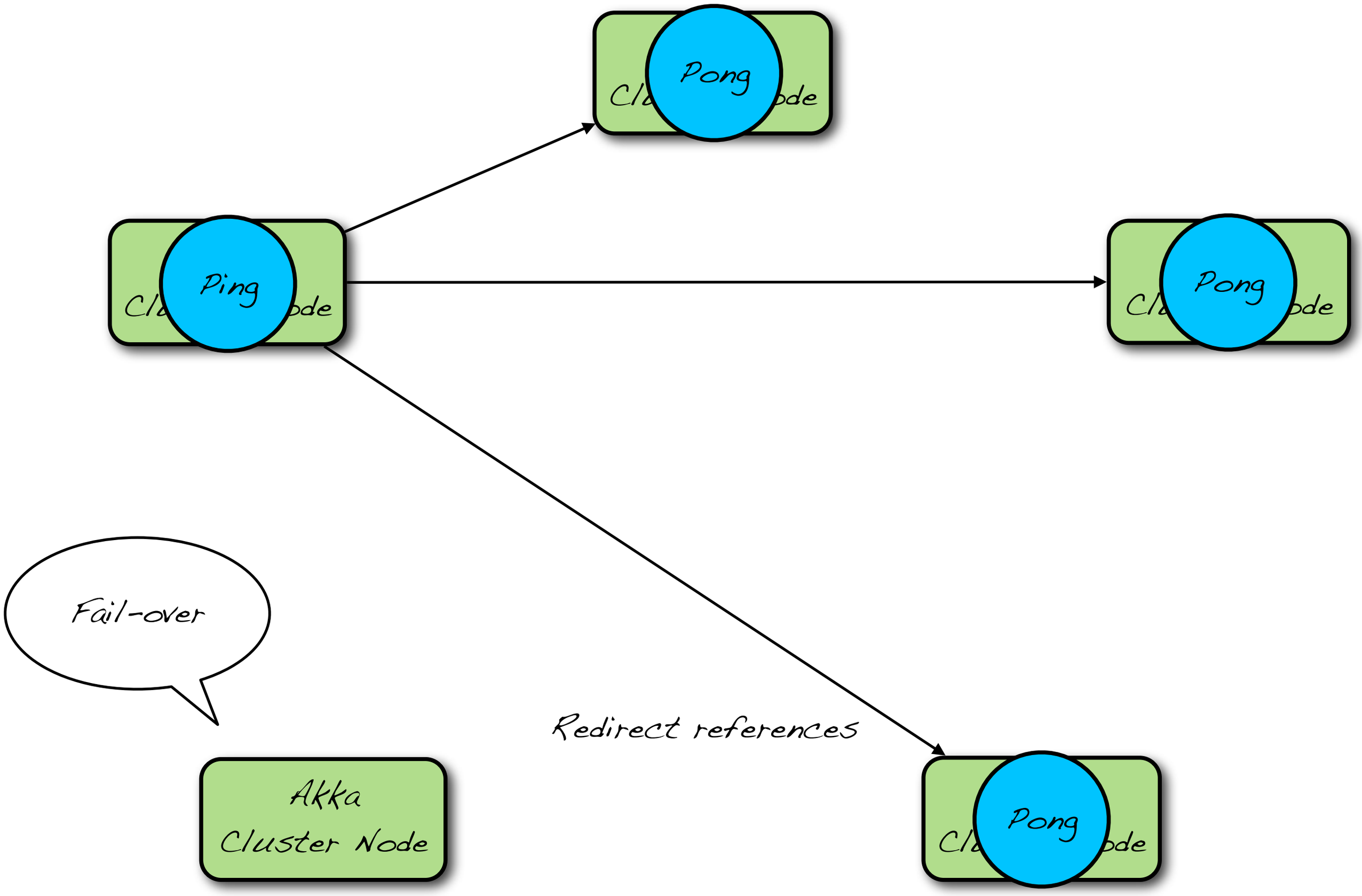
Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node


Pong
Ping
Pong
Pong
akka {
actor {
deployment {
ping {}
pong {
router = "round-robin"
clustered {
replicas = 3
stateless = on
}
}
}
}
}
Akka
Cluster Node


Pong
Ping
Pong
ZooKeeper
Ensemble
Pong
Akka
Cluster Node


Pong
Ping
Pong
Fail-over
Redirect references
Akka
Cluster Node
Pong

Clustering of Stateful Actor

Replication

Transaction log

Deployment configuration

```
akka {
  actor {
    deployment {
      carts {
        clustered {
          home        = "node:test-node-1"
          stateless = off
        }
      }
    }
  }
}
```

Deployment configuration

```
akka {
  actor {
    deployment {
      carts {
        clustered {
          home       = "node:test-node-1"
          stateless = off
        }
      }
    }
  }
}
```

Home node

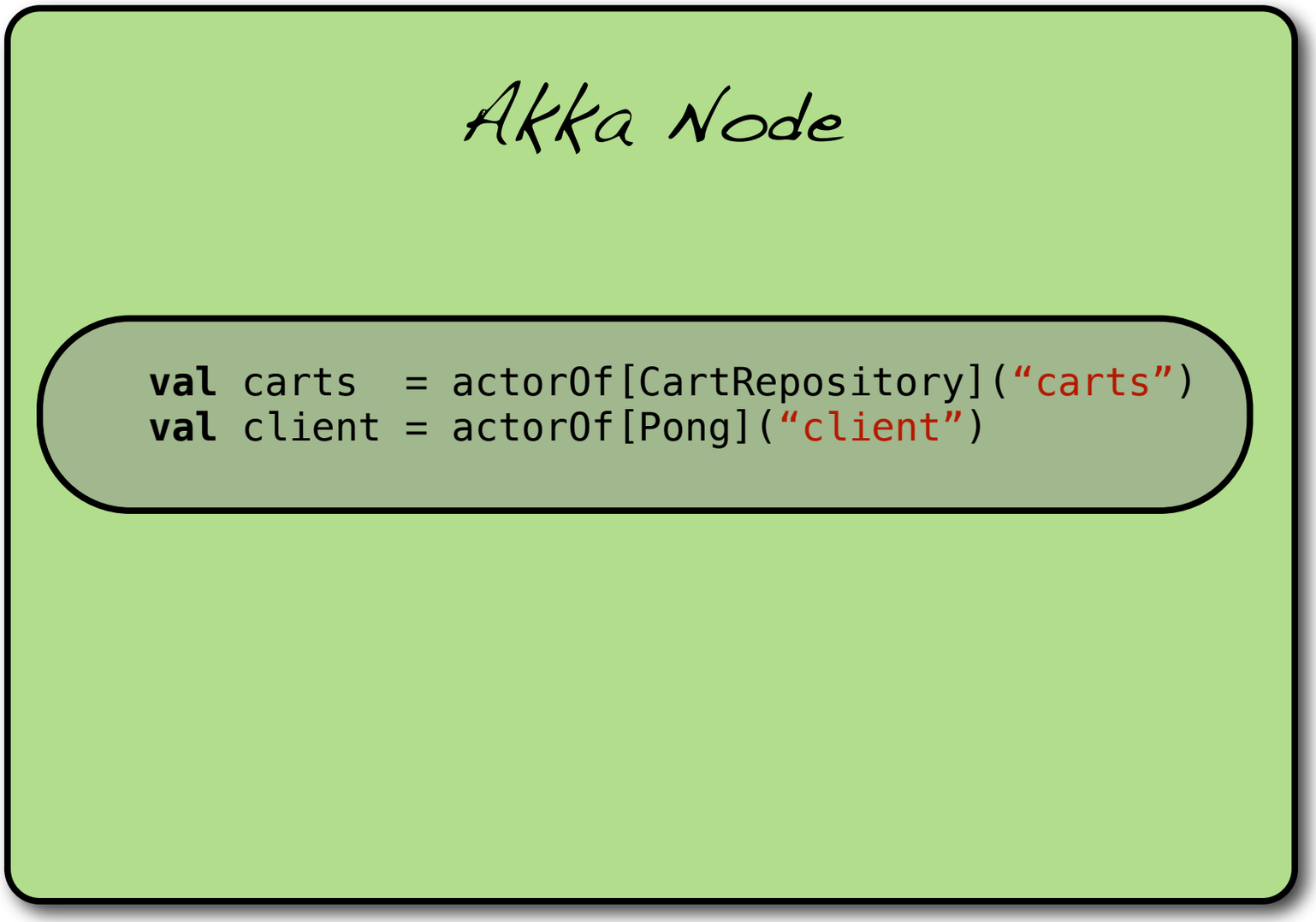
Deployment configuration

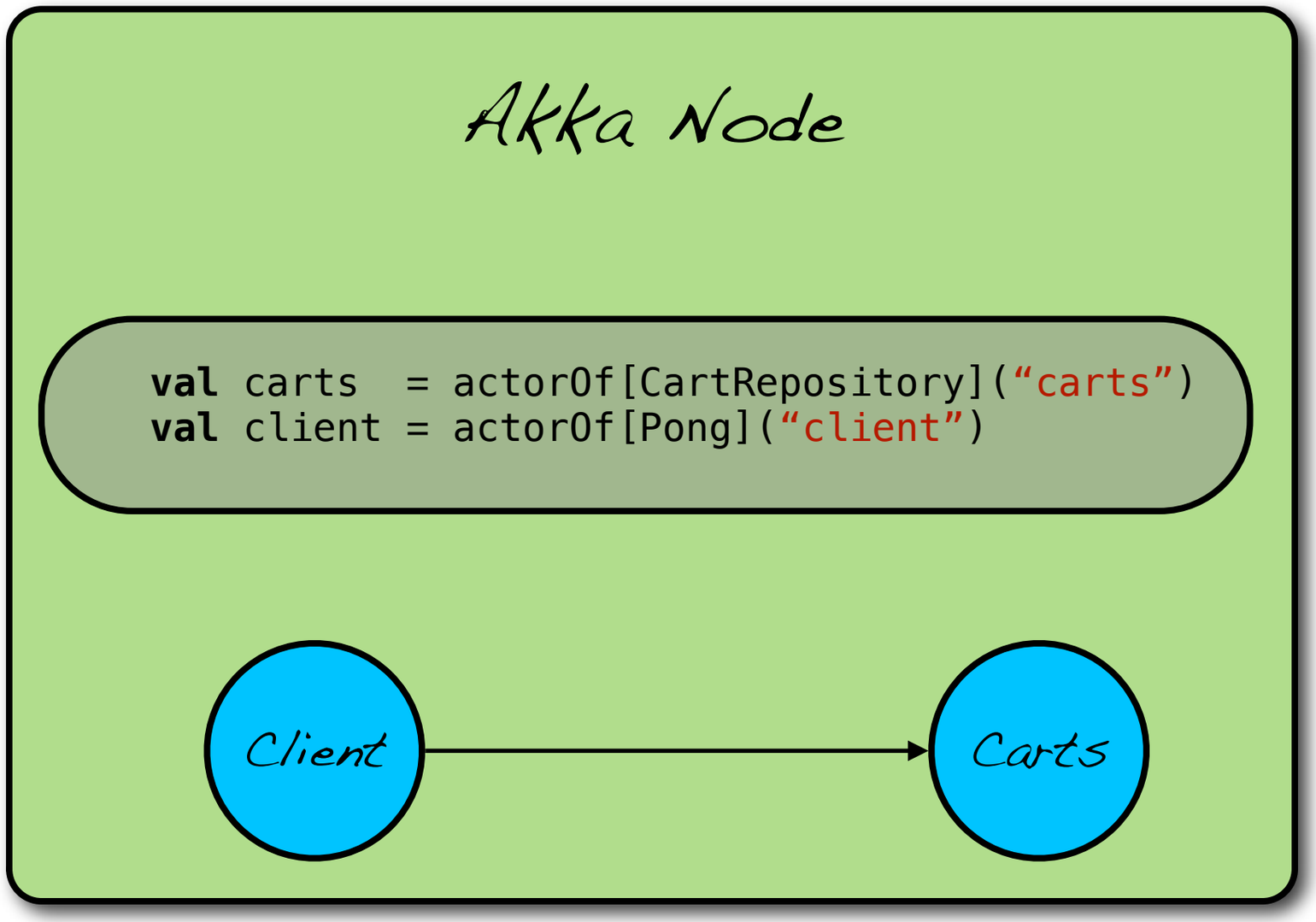
```
akka {
  actor {
    deployment {
      carts {
        clustered {
          home      = "node:test-node-1"
          stateless = off
        }
      }
    }
  }
}
```

Home node

Stateful

Akka Node


Akka Node
val carts = actorOf[CartRepository](“carts”)
val client = actorOf[Pong](“client”)


Akka Node
val carts = actorOf[CartRepository]("carts")
val client = actorOf[Pong]("client")
Client
Carts

Akka
Cluster Node

Client

Carts

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Client

Carts

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Client

Carts

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

```
akka {
  actor {
    deployment {
      carts {
        clustered {
          stateless = off
        }
      }
    }
  }
}
```

Client

Carts

Akka
Cluster Node

Akka
Cluster Node

Carts

Client

Akka
Cluster Node

Transaction Log

Akka
Cluster Node

Akka
Cluster Node

Write behind replication

Carts

Client

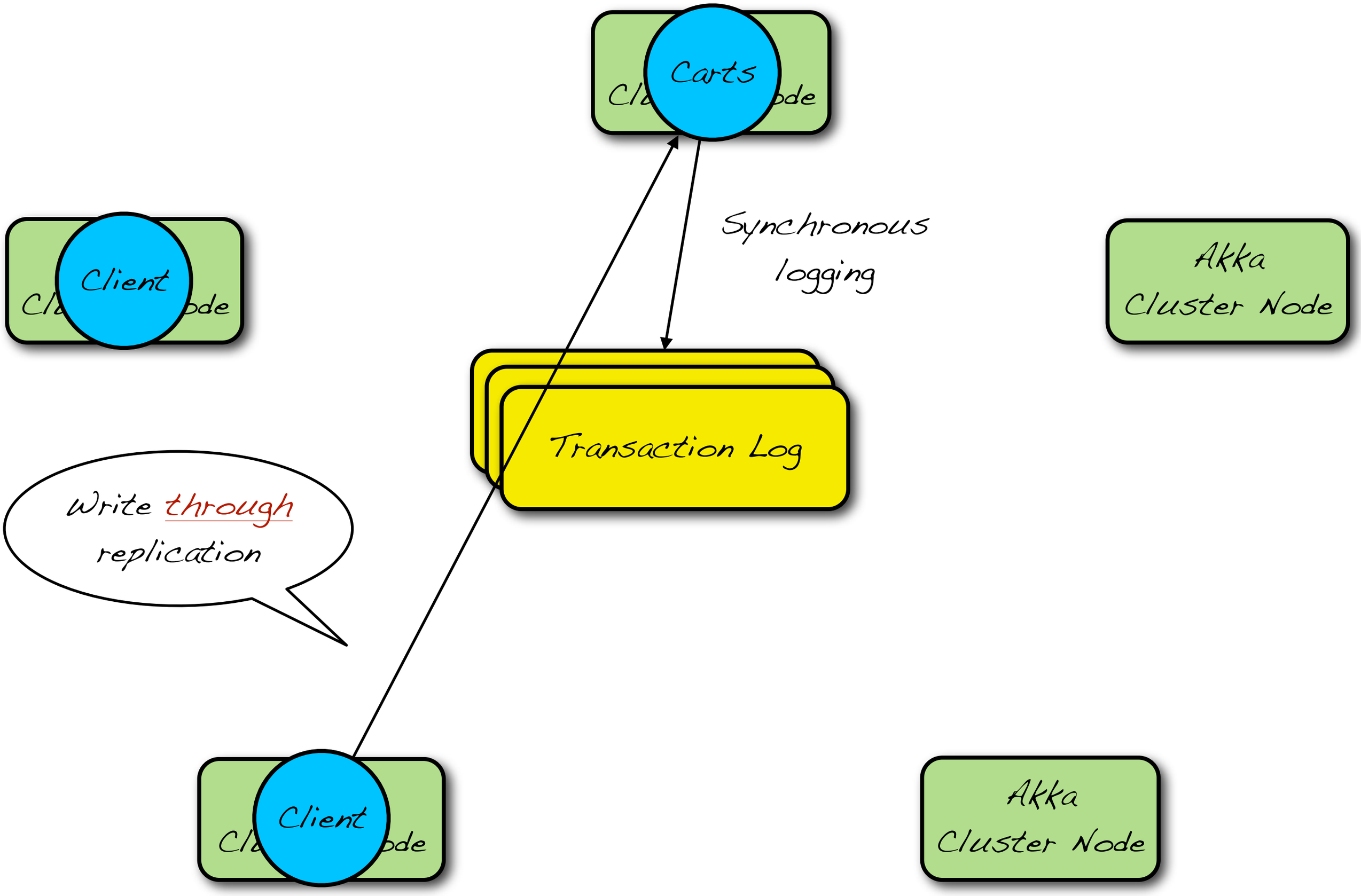
Asynchronous logging

Akka Cluster Node

Transaction Log

Akka Cluster Node

Akka Cluster Node


Carts
Client
Synchronous
logging
Akka
Cluster Node
Transaction Log
Write through
replication
Client
Akka
Cluster Node

Carts

Client

Akka
Cluster Node

Transaction Log

Write through
replication

Client

Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Carts

Transaction Log

Client

Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Carts

Transaction Log

Replay transaction log

Client

Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Carts

Transaction Log

Client

Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Carts

Redirect references

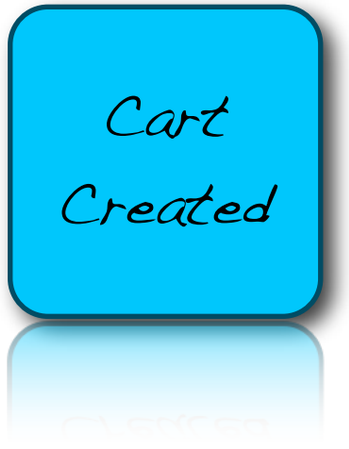
Transaction Log

Client

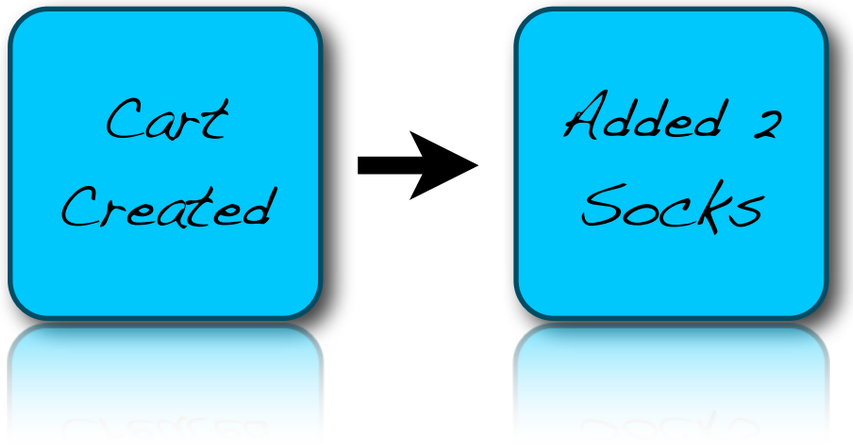
Akka
Cluster Node

Transaction log
Store messages

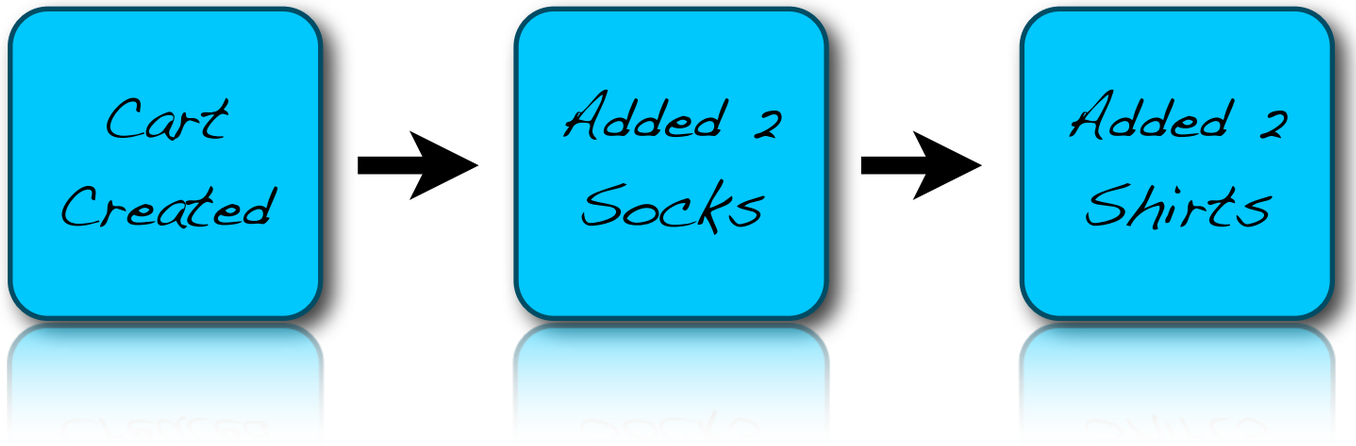
Transaction log
Store messages



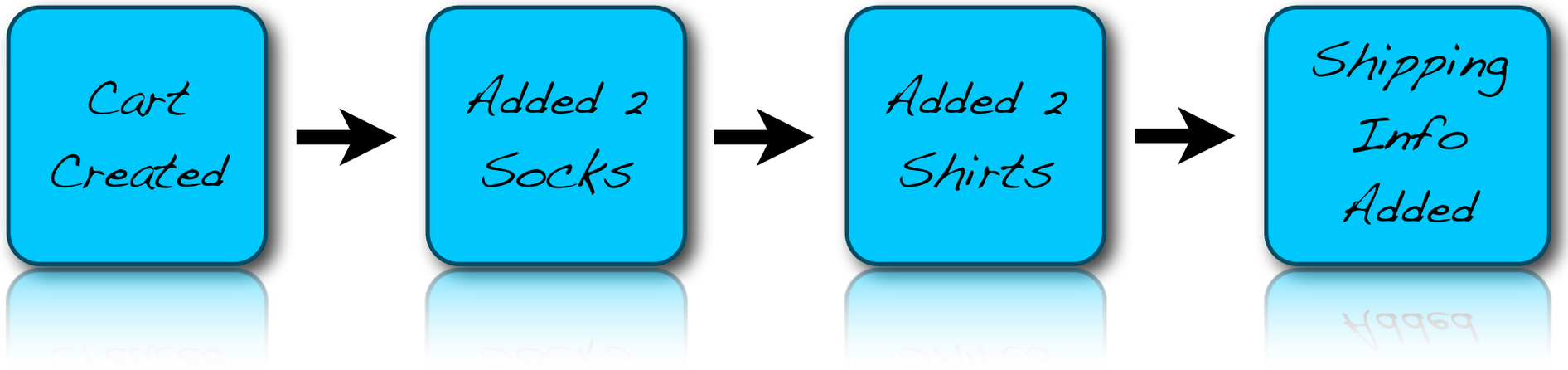
Transaction log
Store messages



Transaction log
Store messages

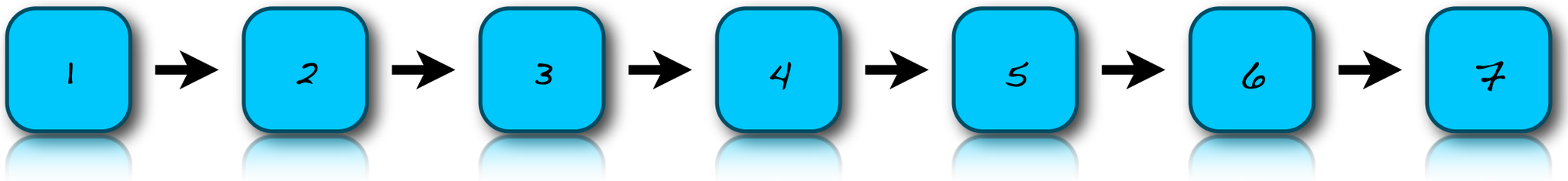


Transaction log
Store messages



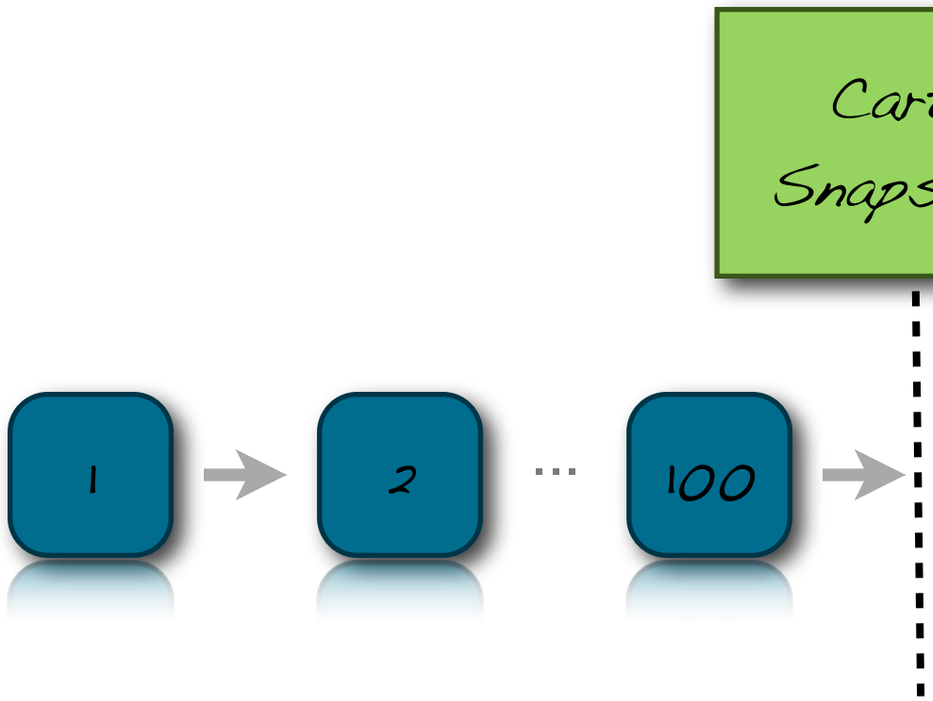
Transaction log
Replaying messages

Transaction log
Replaying messages

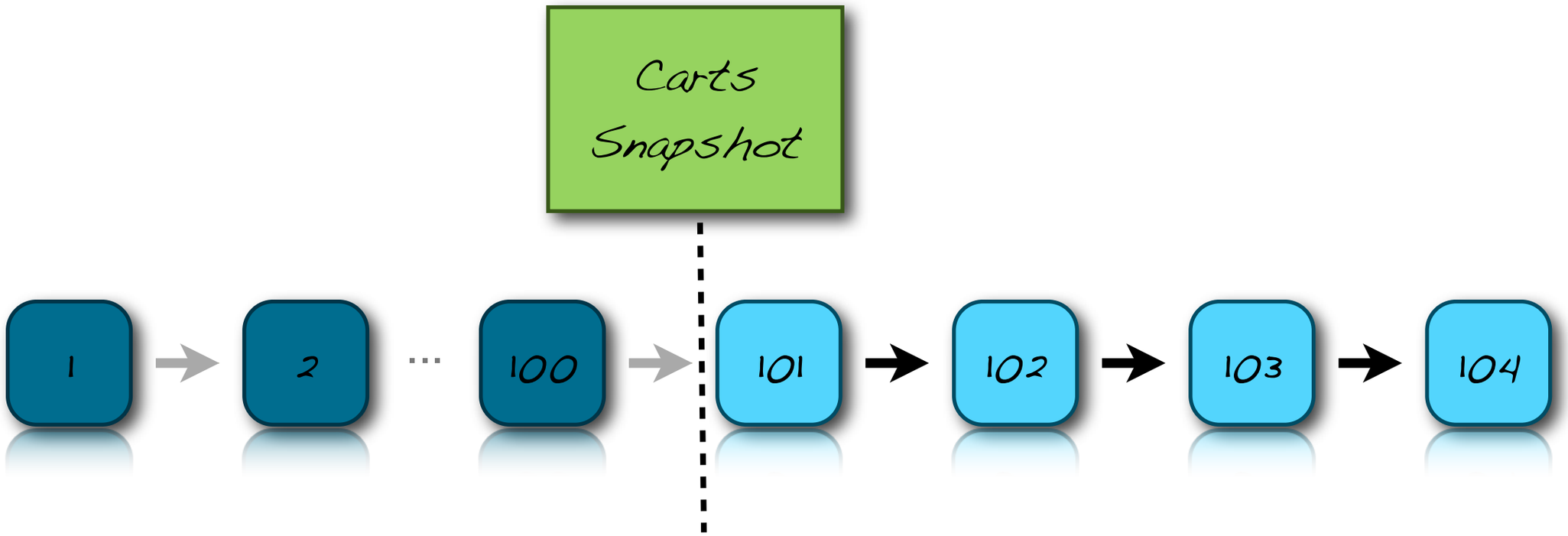


Transaction log
Rolling Snapshot

Transaction log
Rolling Snapshot



Transaction log
Rolling Snapshot



Replication

2

Data Grid

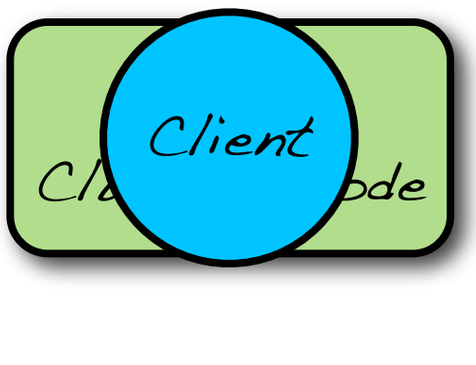
Data Grid

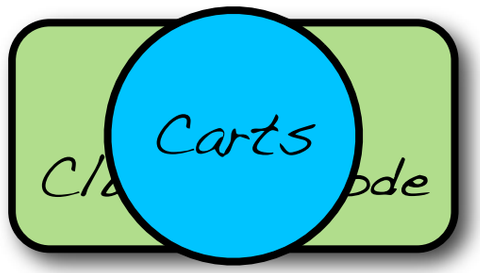
Actor state

- Stored in "external" Data Grid
- Transactional (distributed STM)
- Versioned
- Replicated
- Queries

Implementations

- Custom Akka Data Grid
- SPI for third-party Data Grids


Client


Carts


Akka
Cluster Node


Akka
Cluster Node


Akka
Cluster Node

Carts

Client

```
akka {
  actor {
    deployment {
      carts {
        clustered {
          stateless = off
          replicas = 3
        }
      }
    }
  }
}
```

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Carts

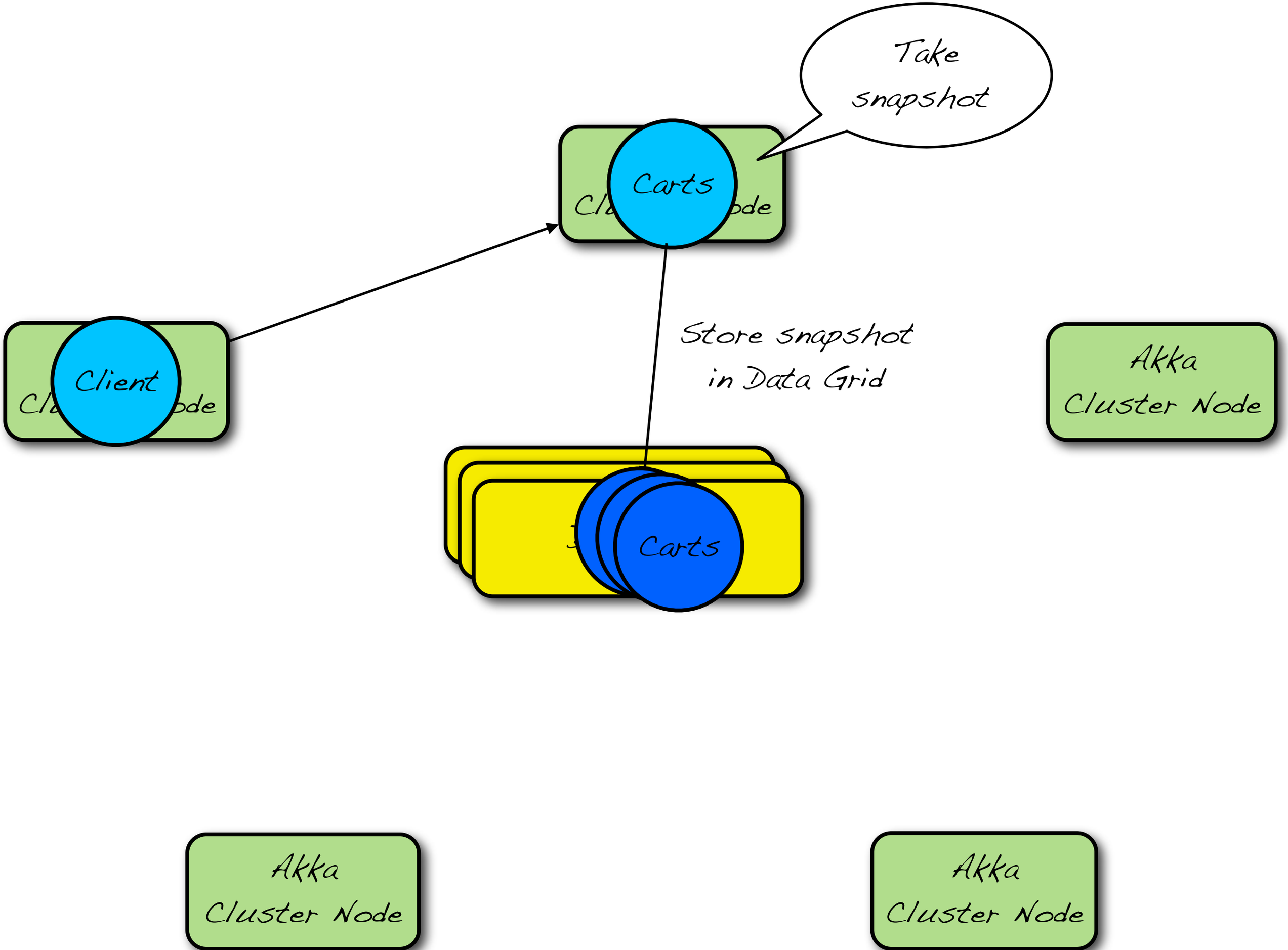
Client

Akka
Cluster Node

Data Grid

Akka
Cluster Node

Akka
Cluster Node


Take
snapshot
Carts
Client
Store snapshot
in Data Grid
Akka
Cluster Node
Carts
Akka
Cluster Node
Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Carts

Carts

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Carts

Carts

Read in snapshot
from Data Grid

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Fail-over

Client

Redirect references

Carts

Carts

Akka
Cluster Node

Akka
Cluster Node

For power users: Cluster API

For power users: Cluster API

```
import Actor.cluster
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})

cluster store actorRef
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})

cluster store actorRef
cluster remove actorAddress
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})

cluster store actorRef
cluster remove actorAddress

val actorRef = cluster use actorAddress
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})

cluster store actorRef
cluster remove actorAddress

val actorRef = cluster use actorAddress
val actorRef = cluster ref (actorAddress, router)
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})

cluster store actorRef
cluster remove actorAddress

val actorRef = cluster use actorAddress
val actorRef = cluster ref (actorAddress, router)

cluster migrate (fromNode, toNode, actorAddress)
```

For power users: Cluster API

```
import Actor.cluster

cluster.start()
cluster.shutdown()

cluster register (new ChangeListener {
  def nodeConnected(node: String, client: ClusterNode) { ... }
  ...
})

cluster store actorRef
cluster remove actorAddress

val actorRef = cluster use actorAddress
val actorRef = cluster ref (actorAddress, router)

cluster migrate (fromNode, toNode, actorAddress)

cluster send (() => { ... }, nrReplicas) map (_.result)
```

Routers

- Direct
- Random
- Round robin
- Least CPU (soon)
- Least RAM (soon)
- Least messages (soon)
- Custom

Durable Mailboxes

- File-based
- Redis-based
- Beanstalk-based
- MongoDB-based
- ZooKeeper-based
- Cassandra-based (soon)
- AMQP-based (soon)
- JMS-based (soon)

Part of Akka 2.0

ETA mid fall

http://akka.io

EOF