Above the Clouds:
Introducing Akka

Jonas Bonér
CTO Typesafe
Twitter: @jboner
http://akka.io
http://typesafe.com

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

Heavily influenced by

Erlang OTP

Concurrency

Concurrency

- Actors

Concurrency

- Actors
- Transactional References &
  Data structures

Concurrency

- Actors
- Transactional References & Data structures
- Agents

Concurrency

- Actors
- Transactional References & Data structures
- Agents
- Dataflow Concurrency

Remoting

Remoting

- Erlang-style supervisor hierarchies

Remoting

- Erlang-style supervisor hierarchies
- Actor cluster replication

Remoting

- Erlang-style supervisor hierarchies
- Actor cluster replication
- Actor node migration upon failure

Fault-tolerance

Fault-tolerance

- Clustered Actors

Fault-tolerance

- Clustered Actors
- Data Grid

Fault-tolerance

- Clustered Actors
- Data Grid
- Compute Grid

WHERE IS AKKA USED?

SOME EXAMPLES:

FINANCE

- Stock trend Analysis & Simulation
- Event-driven messaging systems

BETTING & GAMING

- Massive multiplayer online gaming
- High throughput and transactional betting

TELECOM

- Streaming media network gateways

SIMULATION

- 2D/3D simulation engines

E-COMMERCE

- Batch processing
- Social media community sites

Akka Actors

one tool in the toolbox

Actors

```
case class Tick(value: Int)

class Counter extends Actor {
  var counter = 0

  def receive = {
    case Tick(value) =>
      counter += value
      println(counter)
  }
}
```

Create

```
val counter = actorOf[Counter]
```

`counter` is an `ActorRef`

Send:!

```
counter ! Tick(1)
```

fire-forget

Send: ?

```
// returns a future
val future = actor ? message

future.await

val result = future.result
```

returns the Future directly

Reply

```
class SomeActor extends Actor {
  def receive = {
    case User(name) =>
      // use reply
      self.reply(“Hi ” + name)
  }
}
```

Future

```
val future1, future2, future3 =
  Future.empty[String]

future1.onComplete(f => ...)
future2.map(f => ...)
future3.filter(f => ...)

future1.completeWithResult(...)
future2.completeWithException(...)
future3.completeWith(future2)
```

Future

```
val f1 = Futures.future(callable)
val f2 = Futures.reduce(futures)((x, y) => ..)
val f3 = Futures.fold(zero)(futures)((x, y) => ...)
```

HotSwap

```
self become {
  // new message handler
  case NewMessage =>
    ...
}
```

HotSwap

```
actor ! HotSwap {
   // new message handler
   case NewMessage =>
      ...
}
```

HotSwap

```
self.unbecome()
```

Set dispatcher

```
class MyActor extends Actor {
  self.dispatcher =
    Dispatchers.newPinnedDispatcher(self)

  ...
}

actor.dispatcher = dispatcher // before started
```

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
        }
      }
    }
  }
}
```

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

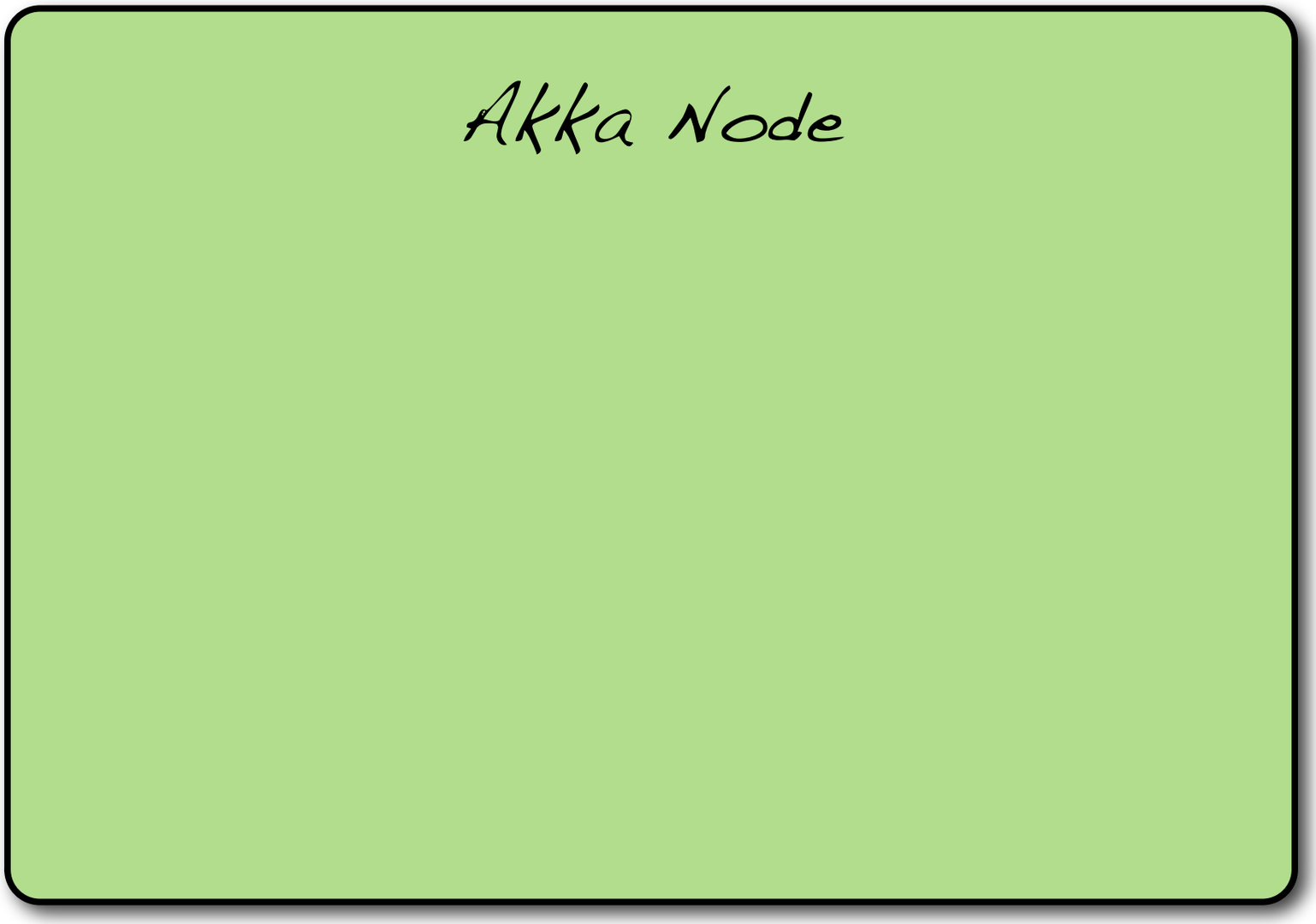
The runtime provides

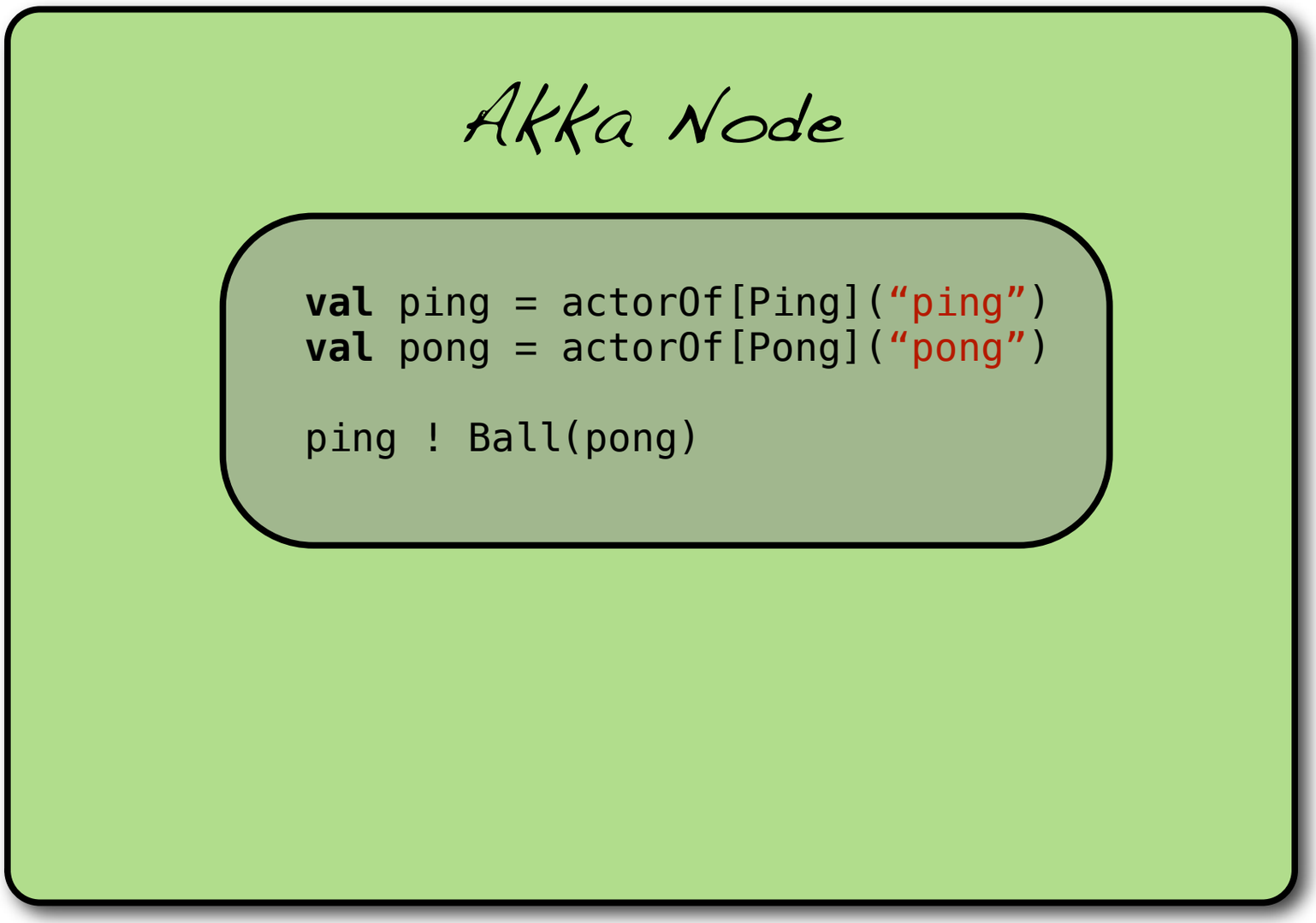
- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing
- Leader election
- Durable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

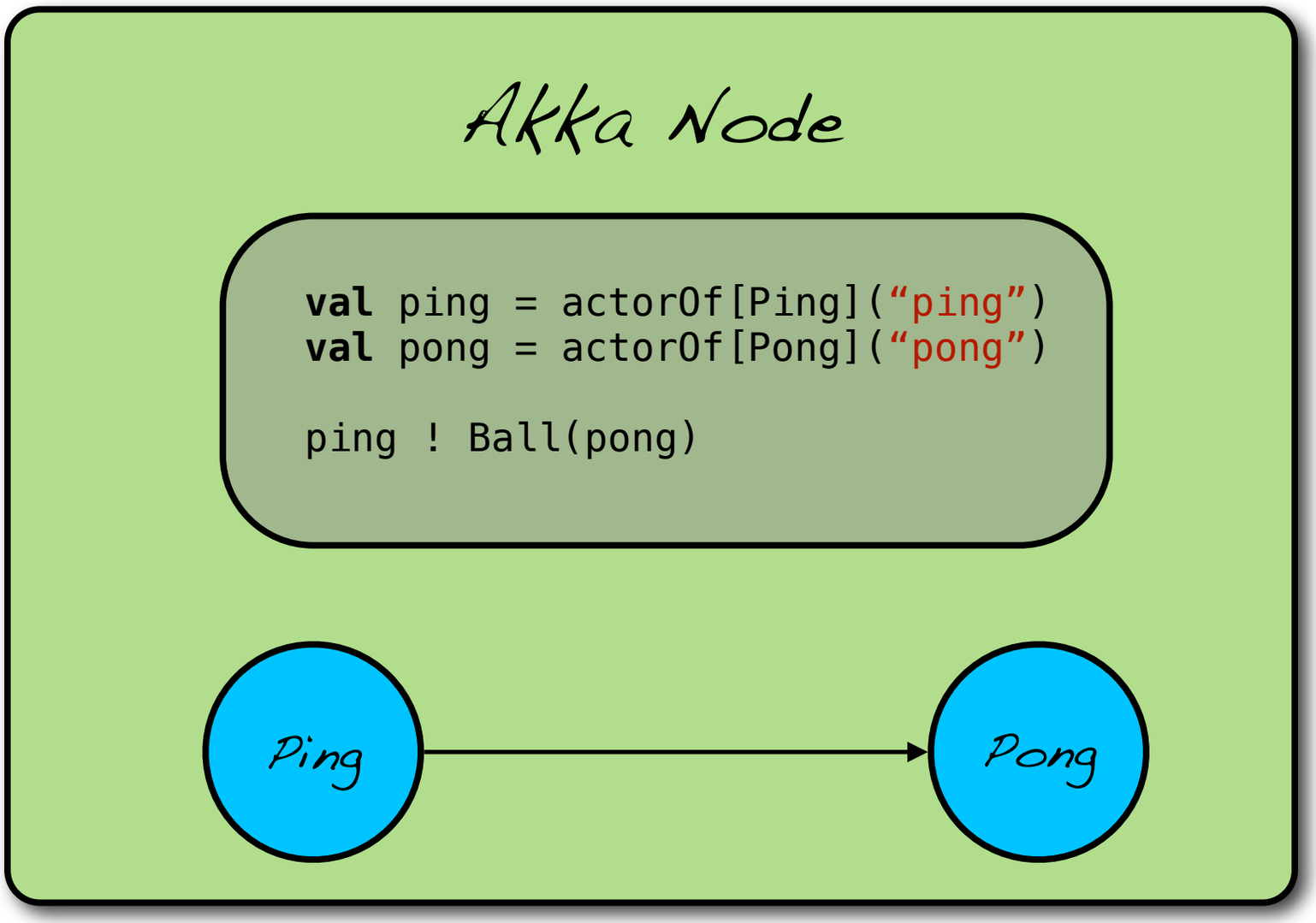
The runtime provides

- 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)
Ping
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 {
  actor {
    deployment {
      ping {}
      pong {
        router = "round-robin"
        clustered {
          replicas  = 3
        }
      }
    }
}
```

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
        }
      }
    }
  }
}
```

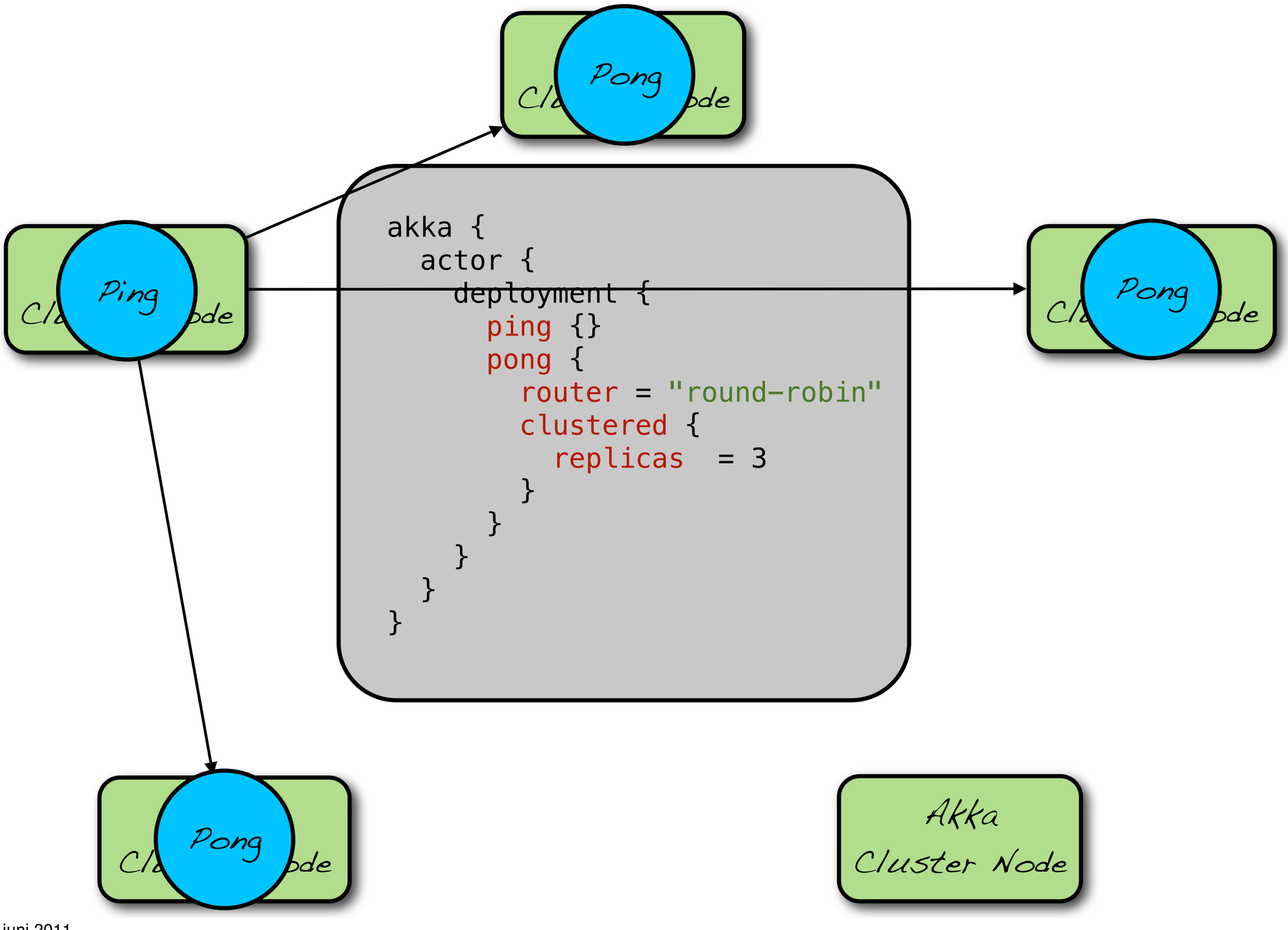
Ping

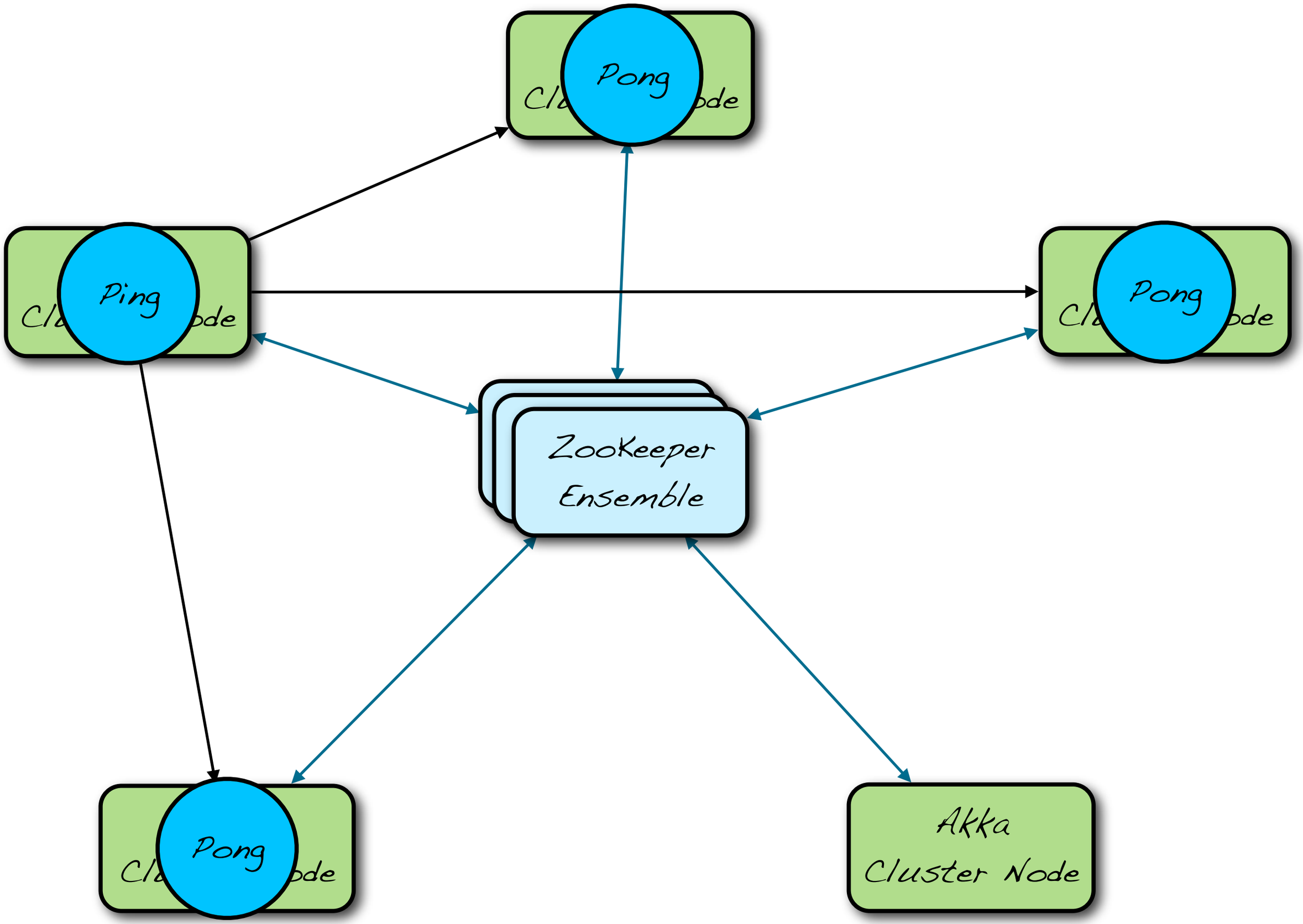
Pong

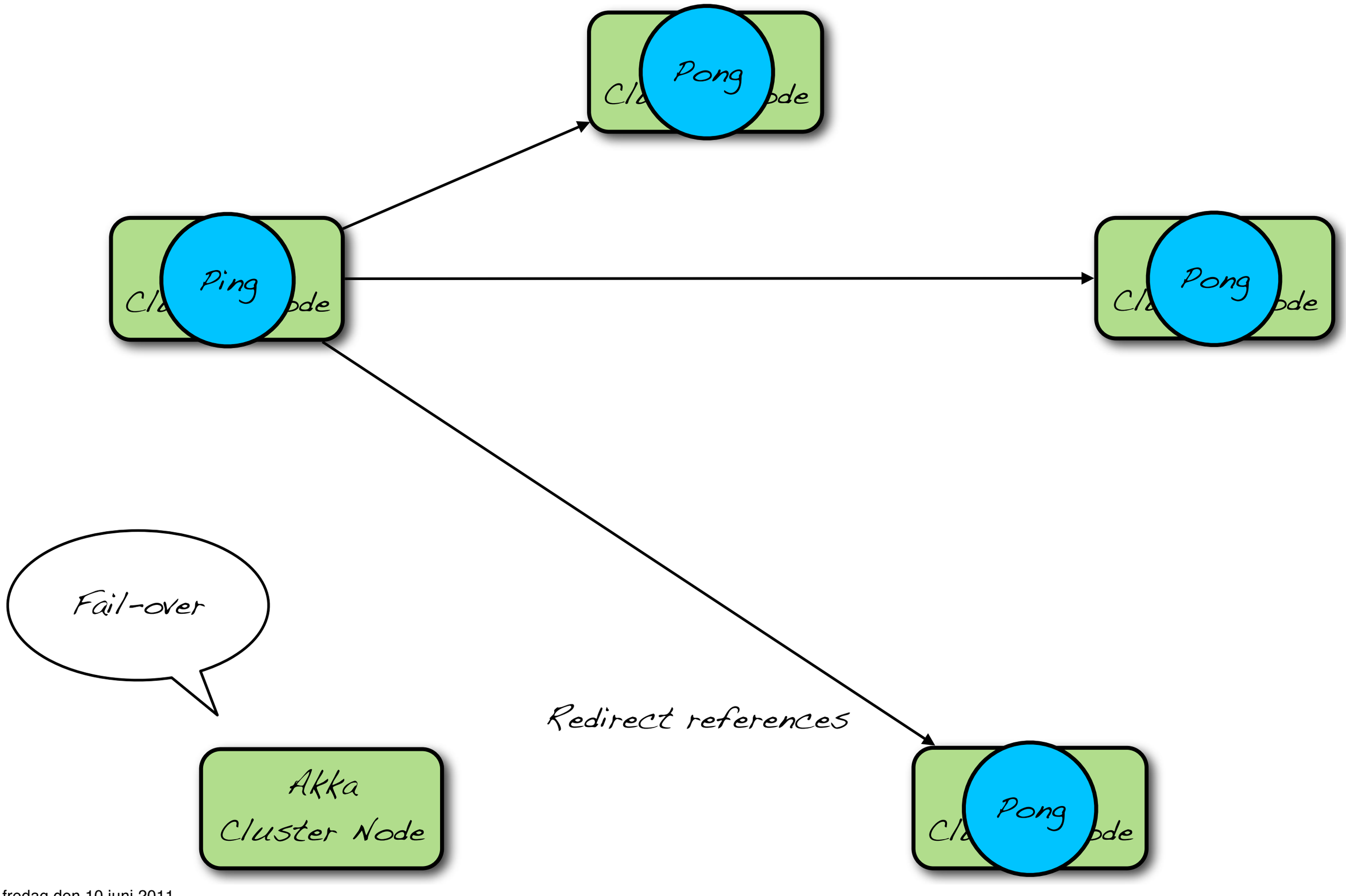
Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node


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


Pong
Ping
Pong
ZooKeeper
Ensemble
Pong
Akka
Cluster Node


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

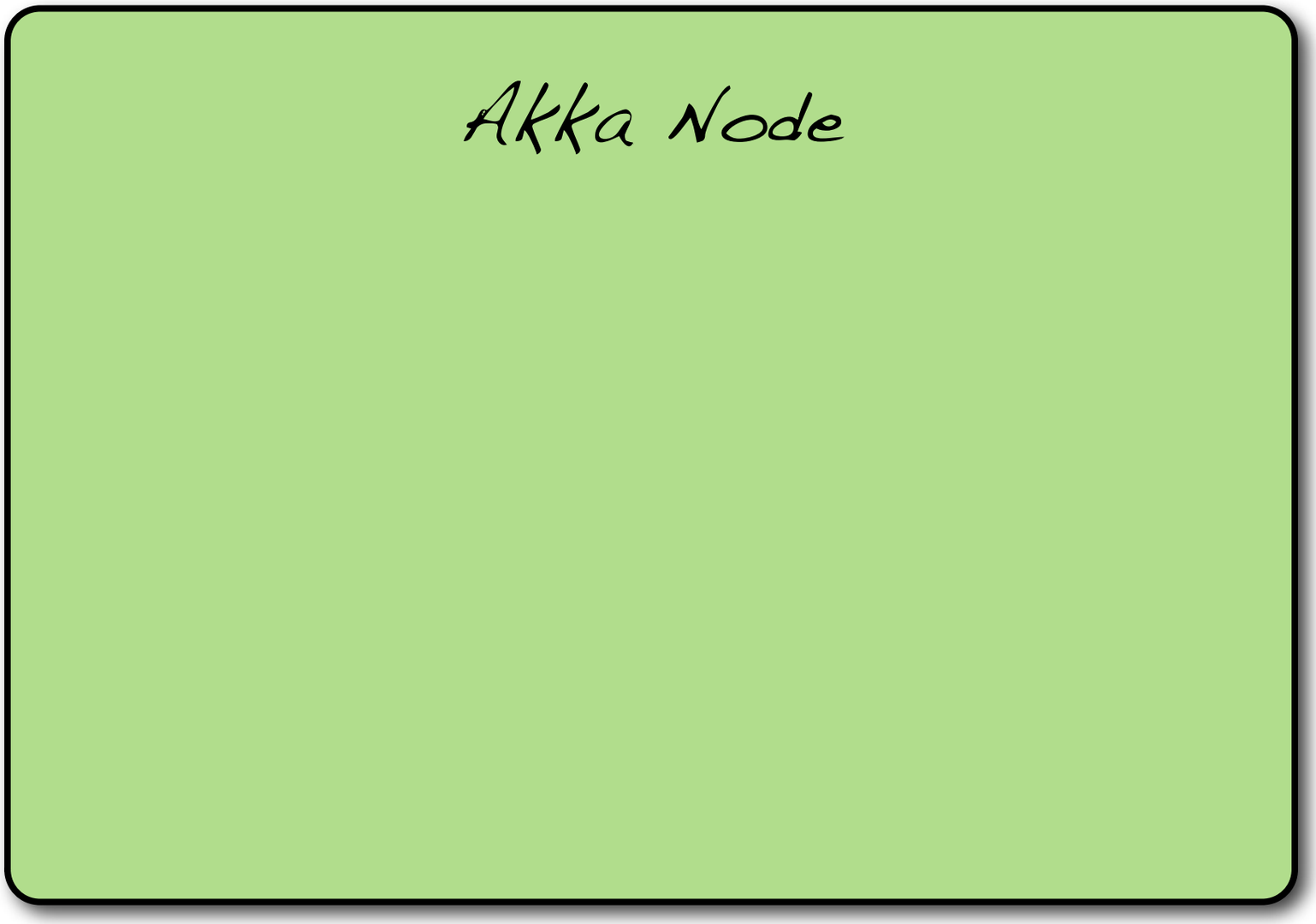
Clustering of Stateful Actor

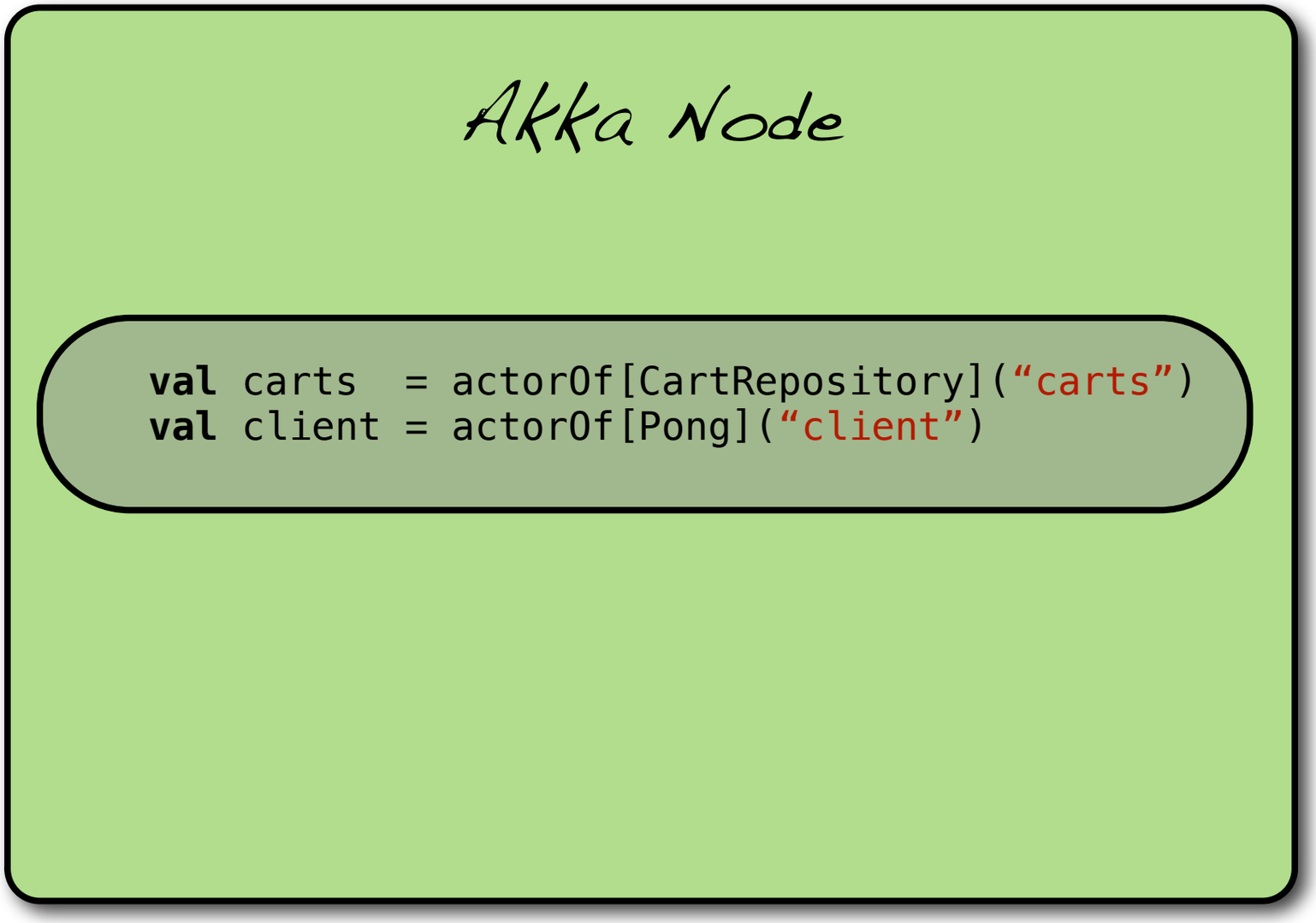
Replication

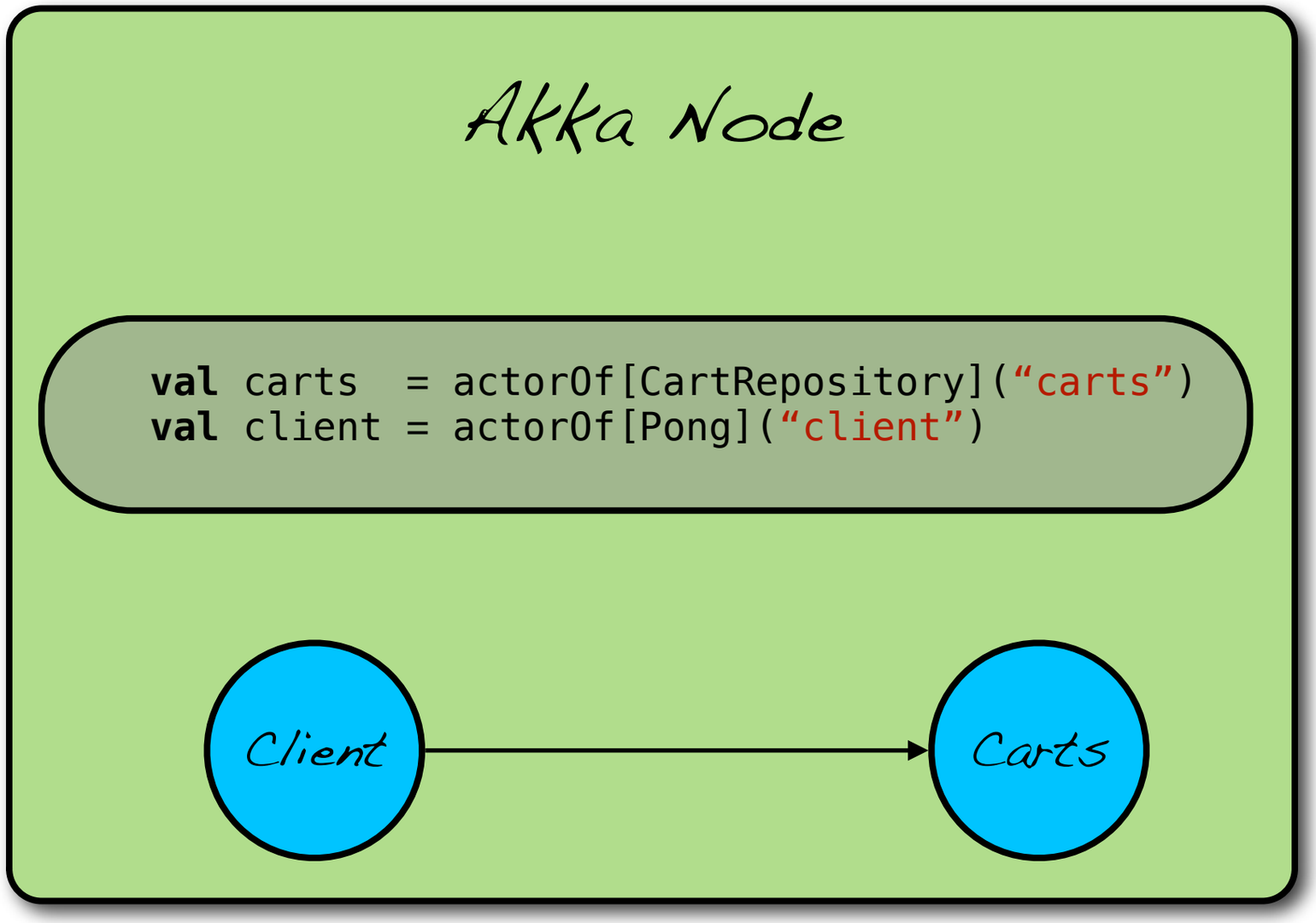
Transaction log

Deployment configuration

```
akka {
  actor {
    deployment {
      carts {
        clustered {
          home = "node:test-node-1"
          replication {
            storage = "transaction-log"
            strategy = "write-through"
          }
        }
      }
    }
  }
}
```


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 {
          replication {
            storage = "transaction-log"
            strategy = "write-through"
          }
        }
      }
    }
  }
}
```

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

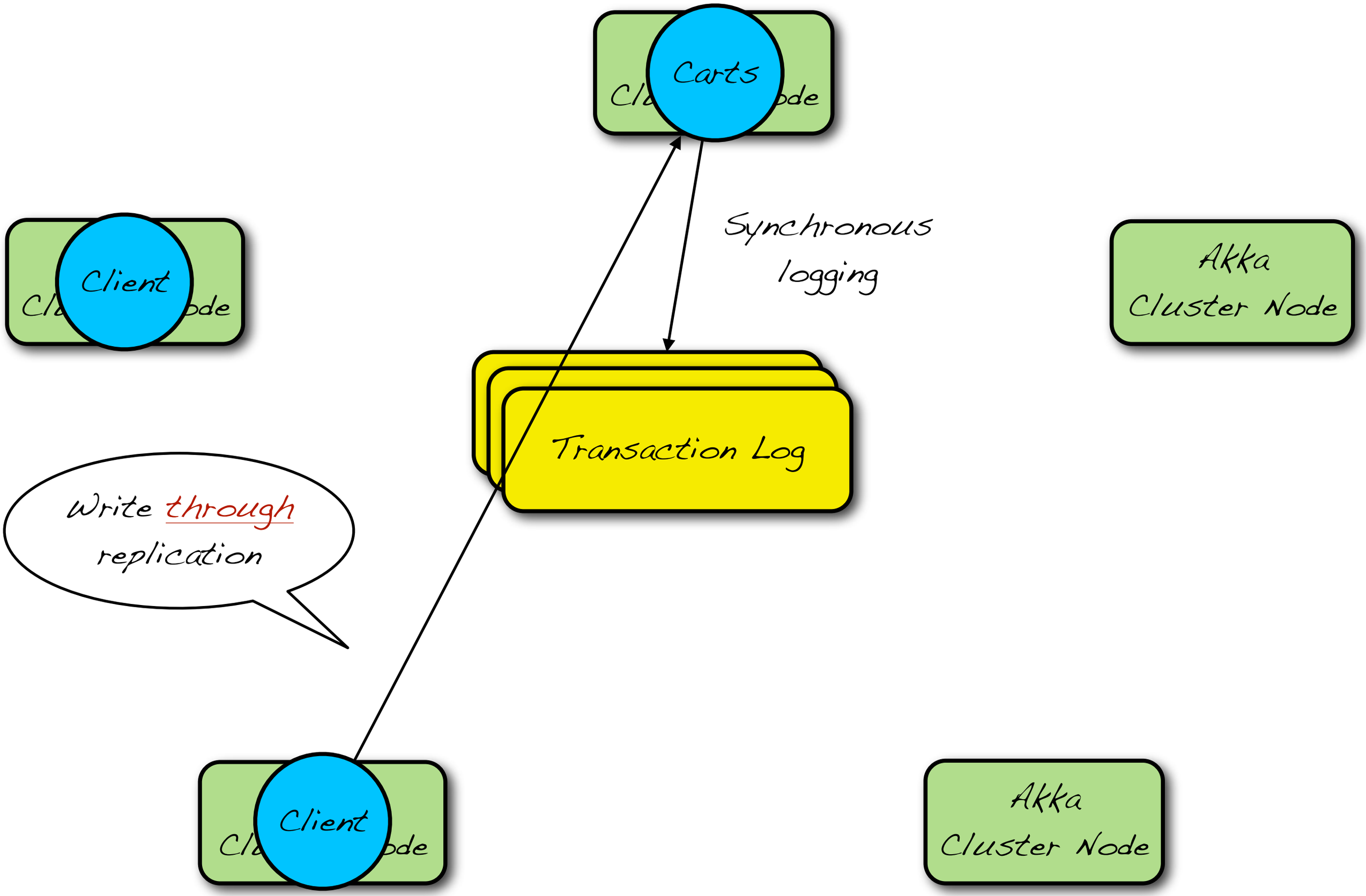
Asynchronous logging

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

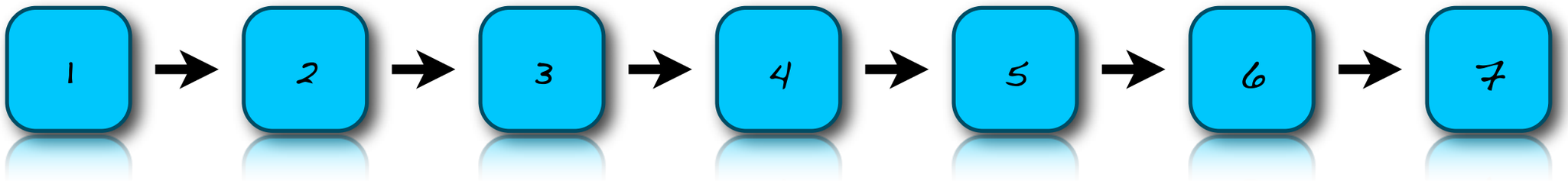
Transaction Log

Client

Akka
Cluster Node

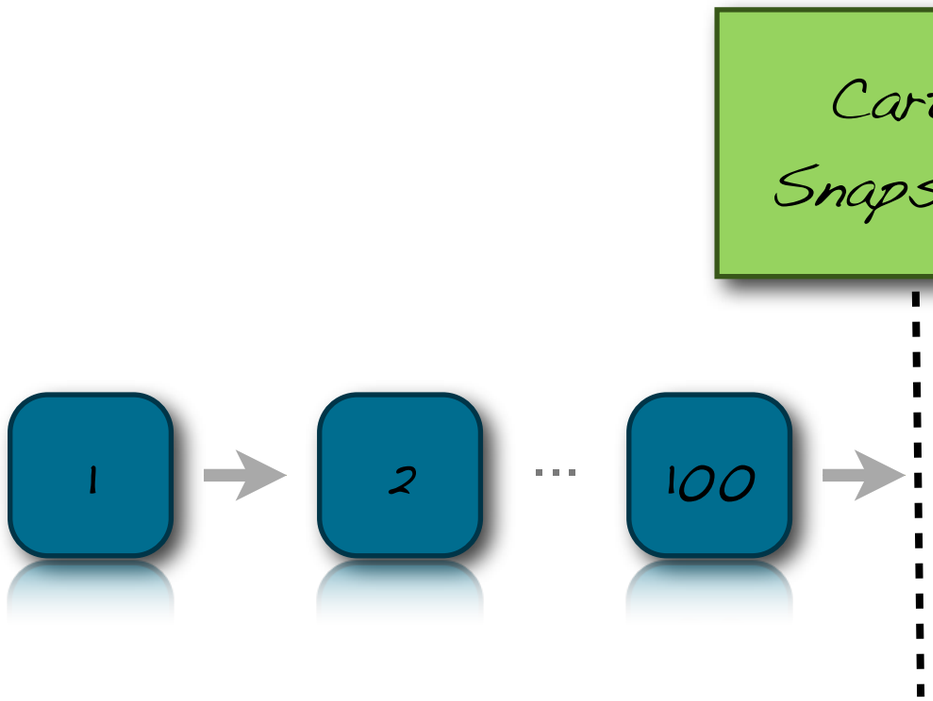
Transaction log
Replaying messages

Transaction log
Replaying messages

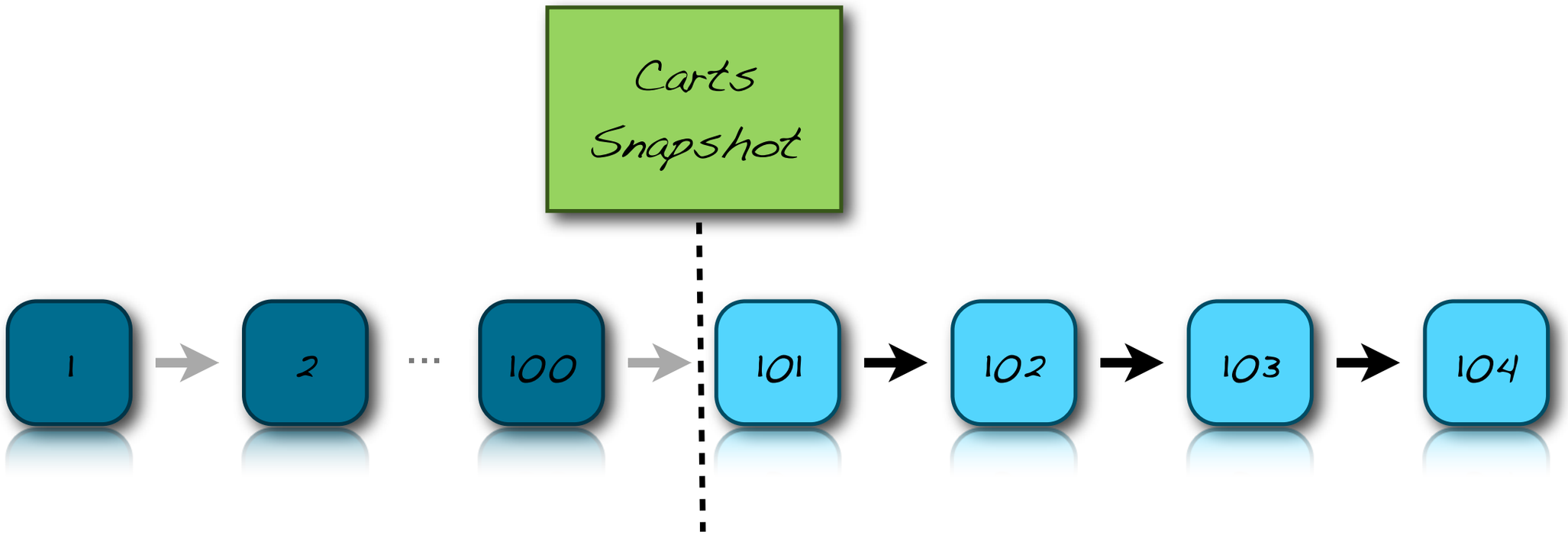


Transaction log
Rolling Snapshot

Transaction log
Rolling Snapshot



Transaction log
Rolling Snapshot



Replication

2

Data Grid

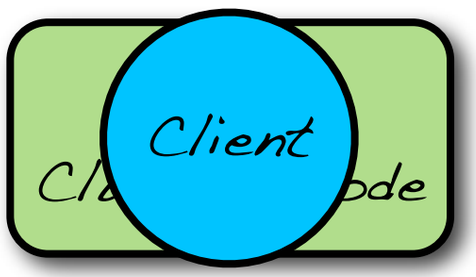
Data Grid

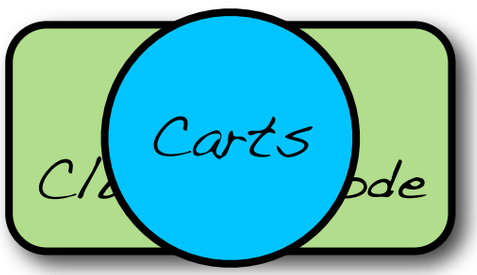
Actor state

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

Implementations

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

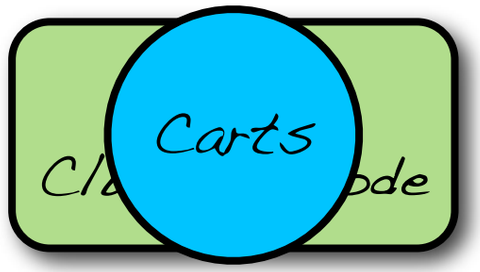

Client

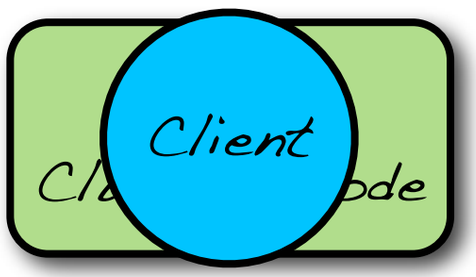

Carts


Akka
Cluster Node


Akka
Cluster Node


Akka
Cluster Node






```
akka {
  actor {
    deployment {
      carts {
        clustered {
          replicas = 3
          replication {
            storage = "data-grid"
            strategy = "write-behind"
          }
        }
      }
    }
  }
}
```

Akka
Cluster Node

Akka
Cluster Node

Akka
Cluster Node

Carts

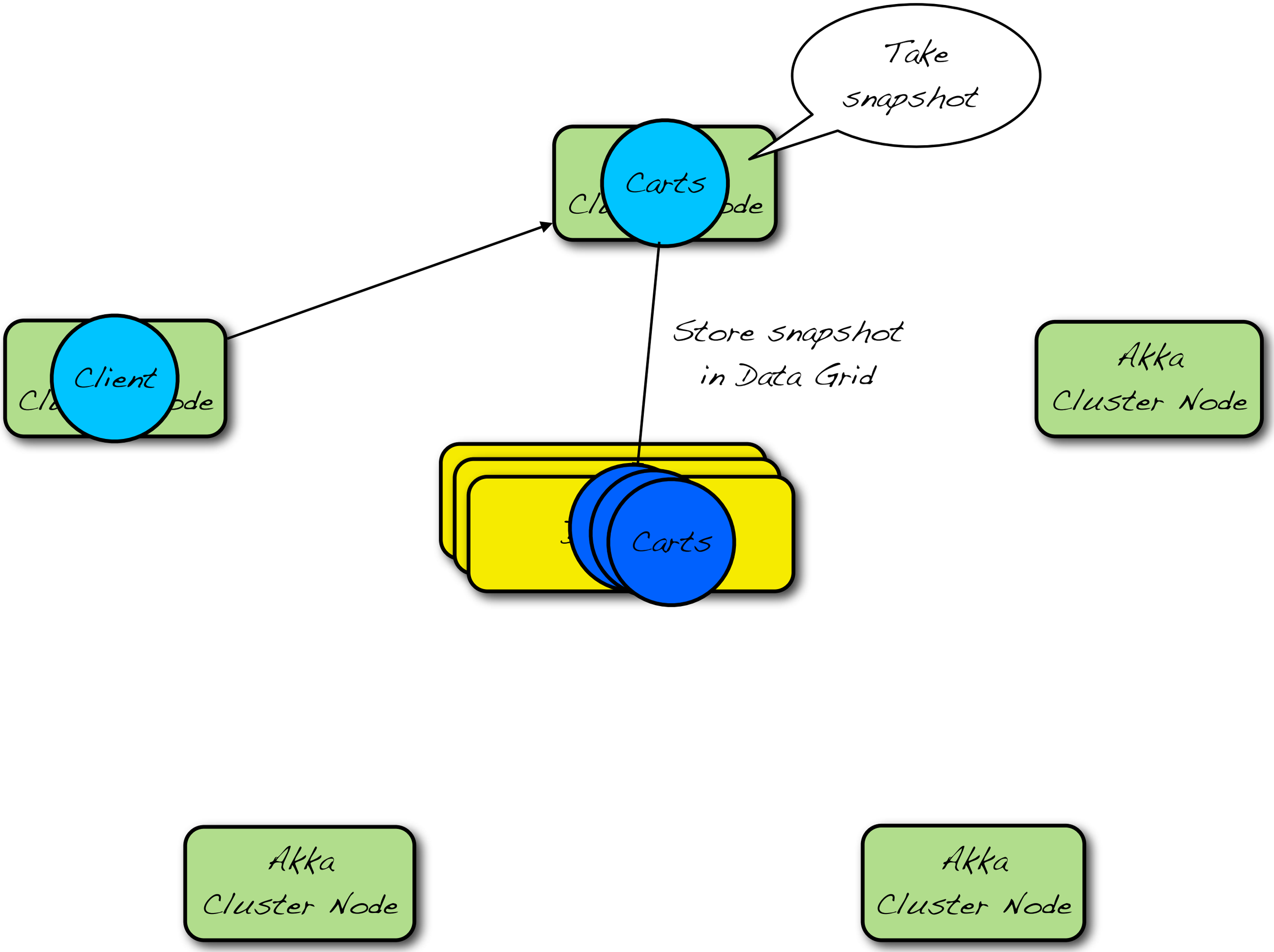
Client

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

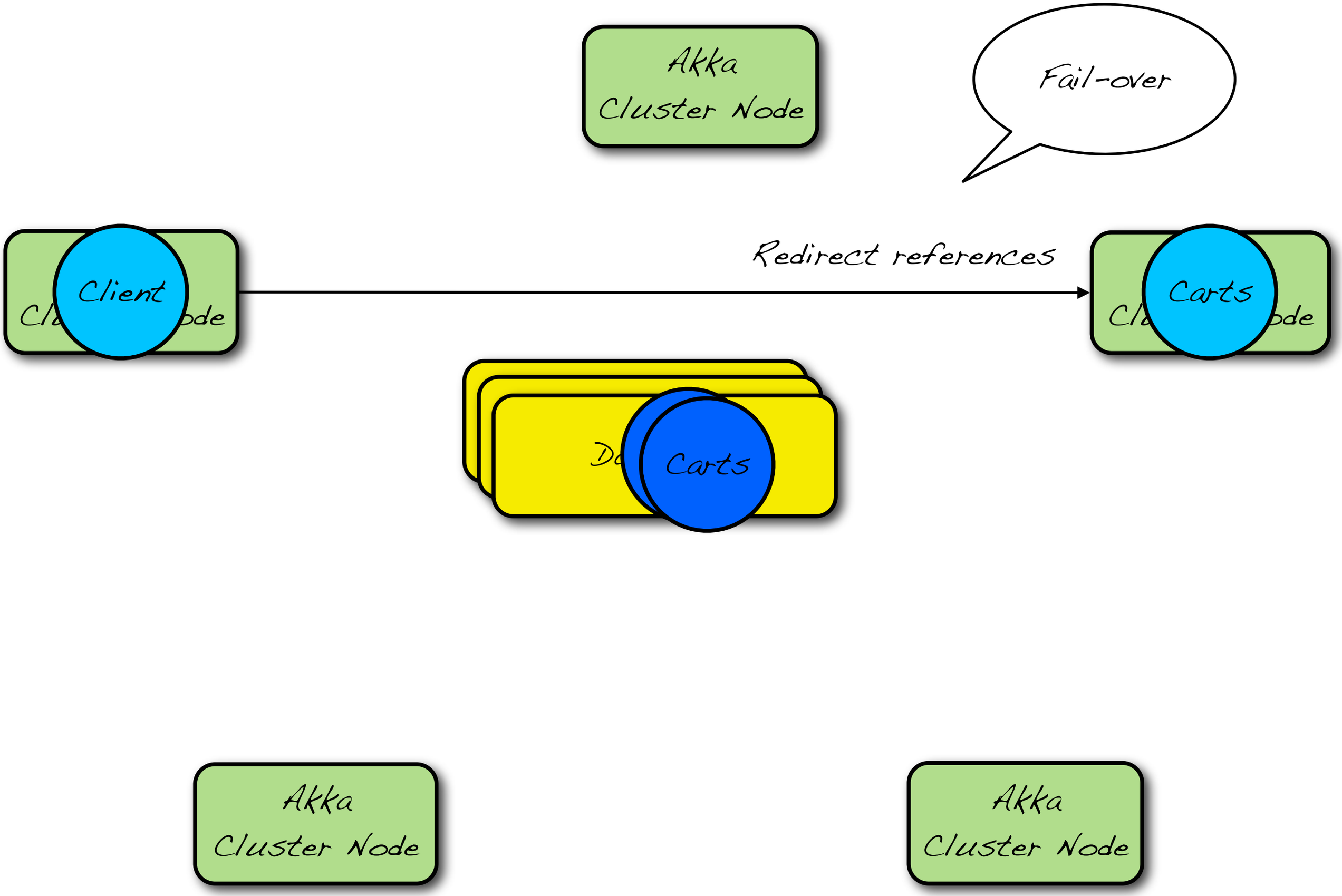
Carts

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)

Clustering of Akka 2.0

ETA mid fall

Learn more

http://akka.io

http://typesafe.com

EOF

Let it crash

fault-tolerance

Linking

```
link(actor)
unlink(actor)

startLink(actor)
spawnLink[MyActor]
```

Fault handlers

```
AllForOneStrategy(
  errors,
  maxNrOfRetries,
  withinTimeRange)

OneForOneStrategy(
  errors,
  maxNrOfRetries,
  withinTimeRange)
```

Supervision

```
class Supervisor extends Actor {
  faultHandler = AllForOneStrategy(
    List(classOf[IllegalStateException]),
    5, 5000))

  def receive = {
    case Register(actor) => link(actor)
  }
}
```

Manage failure

```
class FaultTolerantService extends Actor {
  ...
  override def preRestart(reason: Throwable) = {
    ... // clean up before restart
  }
  override def postRestart(reason: Throwable) = {
    ... // init after restart
  }
}
```