akka : Reloaded

Josh Suereth
@jsuereth

Typesafe

The Problem:

Programming is HARD....

- correct highly concurrent systems
- truly scalable systems
- fault tolerant systems that self-heal

... Let's go shopping!

Simpler

- Concurrency
- Scalability
- Fault-Tolereance

Vision

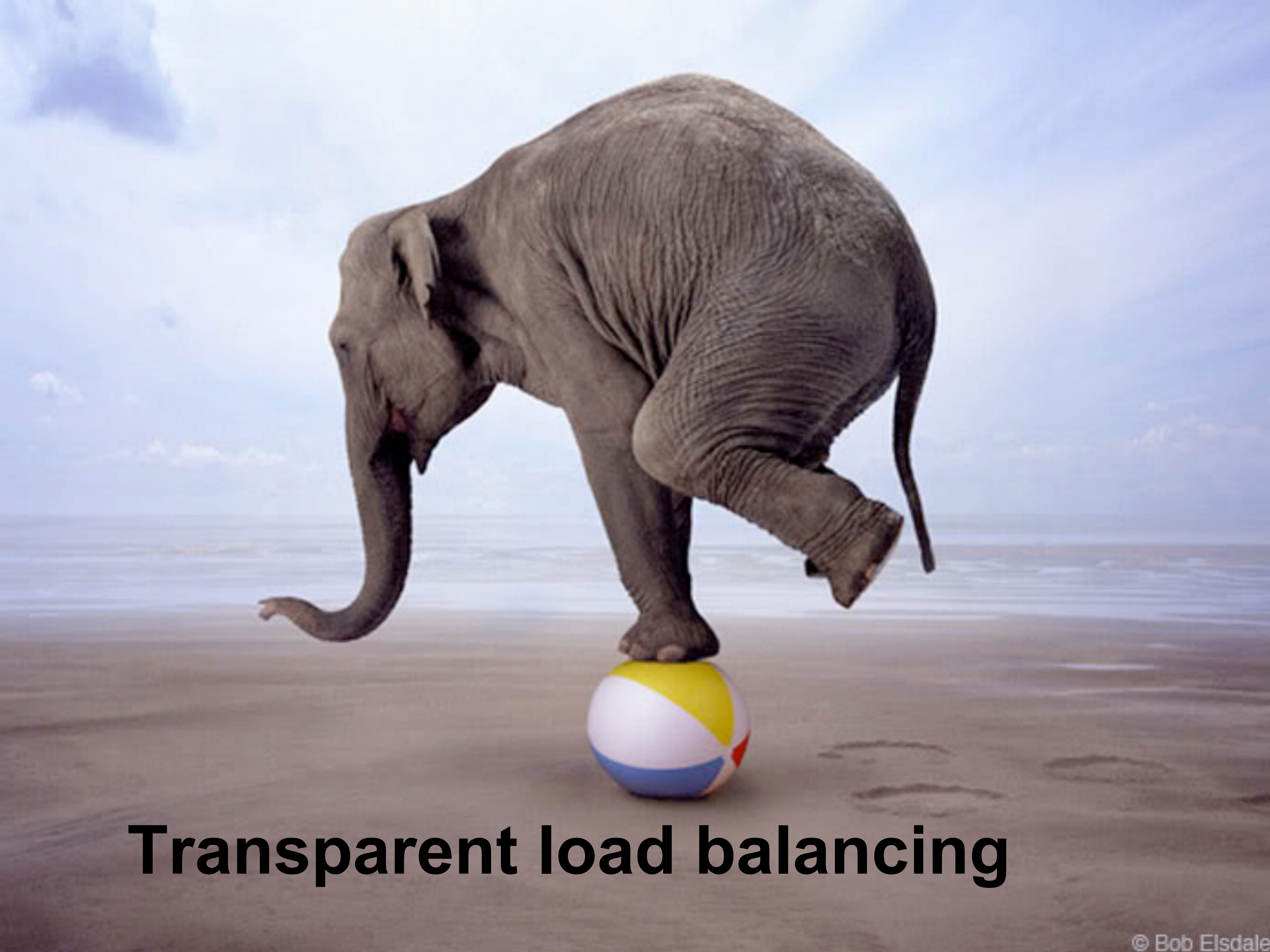
A single unified....

- programming model
- runtime service

Manage System Overload

Scale up & Scale out

Replicate and Distribute for fault tolerance



Transparent load balancing

What was...

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-side

Remote Actor

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

service ! message
```

client-side

We can do better!

Does not meet the vision

- Deployment (local vs remote) is a development 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
        }
      }
    }
  }
}
```

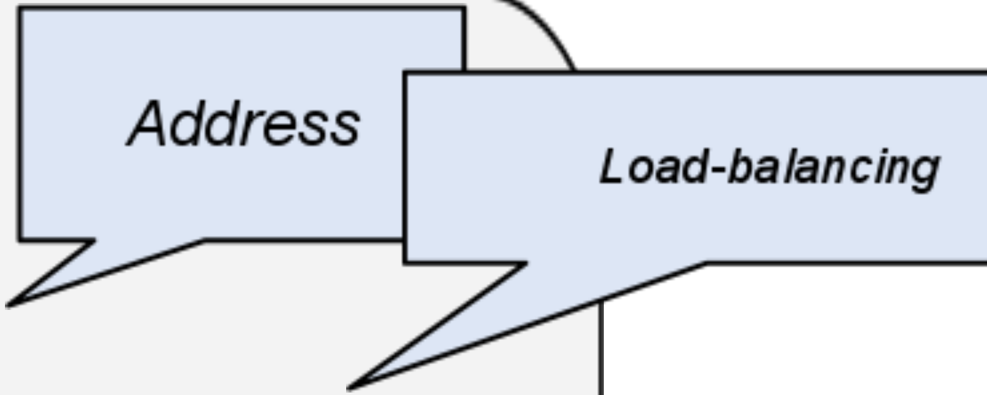
Deployment Configuration

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

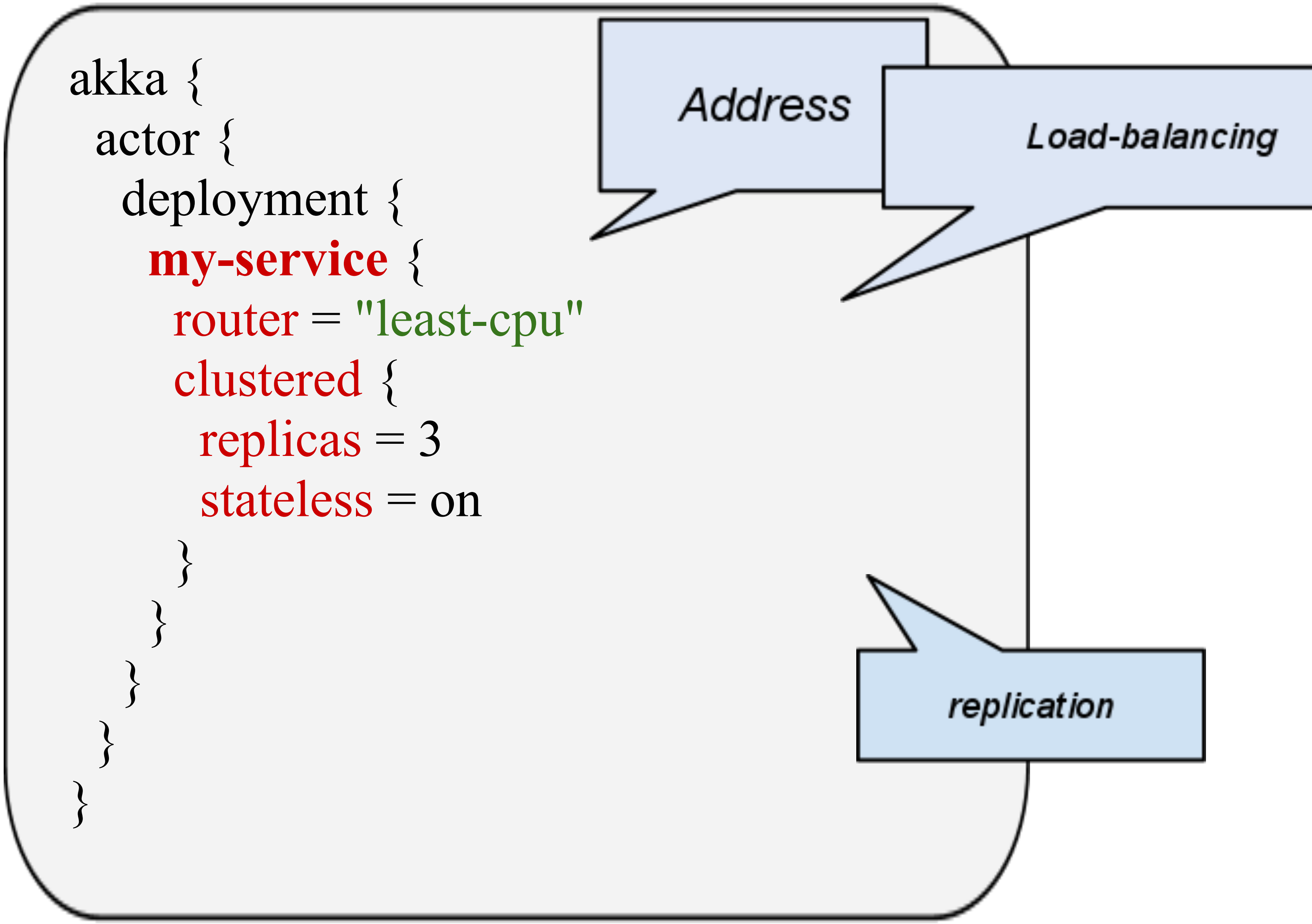
Address

Deployment Configuration

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



Deployment Configuration



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

Deployment

- Actor address is *decoupled* from location and deployment.

Deployment

- Actor address is *decoupled* from location and deployment.
- If no configuration is found, actor is deployed locally

Deployment

- Actor address is *decoupled* from location and deployment.
- If no configuration is found, actor is deployed locally
- The same system can be configured for distribution *without code change*

Deployment

- Actor address is *decoupled* from location and deployment.
- If no configuration is found, actor is deployed locally
- The same system can be configured for distribution *without code change.*
- Write and test locally. Test and deploy in the cloud with confidence.

Deployment

- Actor address is *decoupled* from location and deployment.
- If no configuration is found, actor is deployed locally
- The same system can be configured for distribution *without code change.*
- Write and test locally. Test and deploy in the cloud with confidence.
- Modify distribution at runtime.

Deployment

- Actor address is *decoupled* from location and deployment.
- If no configuration is found, actor is deployed locally
- The same system can be configured for distribution *without code change.*
- Write and test locally. Test and deploy in the cloud with confidence.
- Modify distribution at runtime.
- Runtime will dynamically and adaptively change topology

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

The runtime provides...

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

The runtime provides...

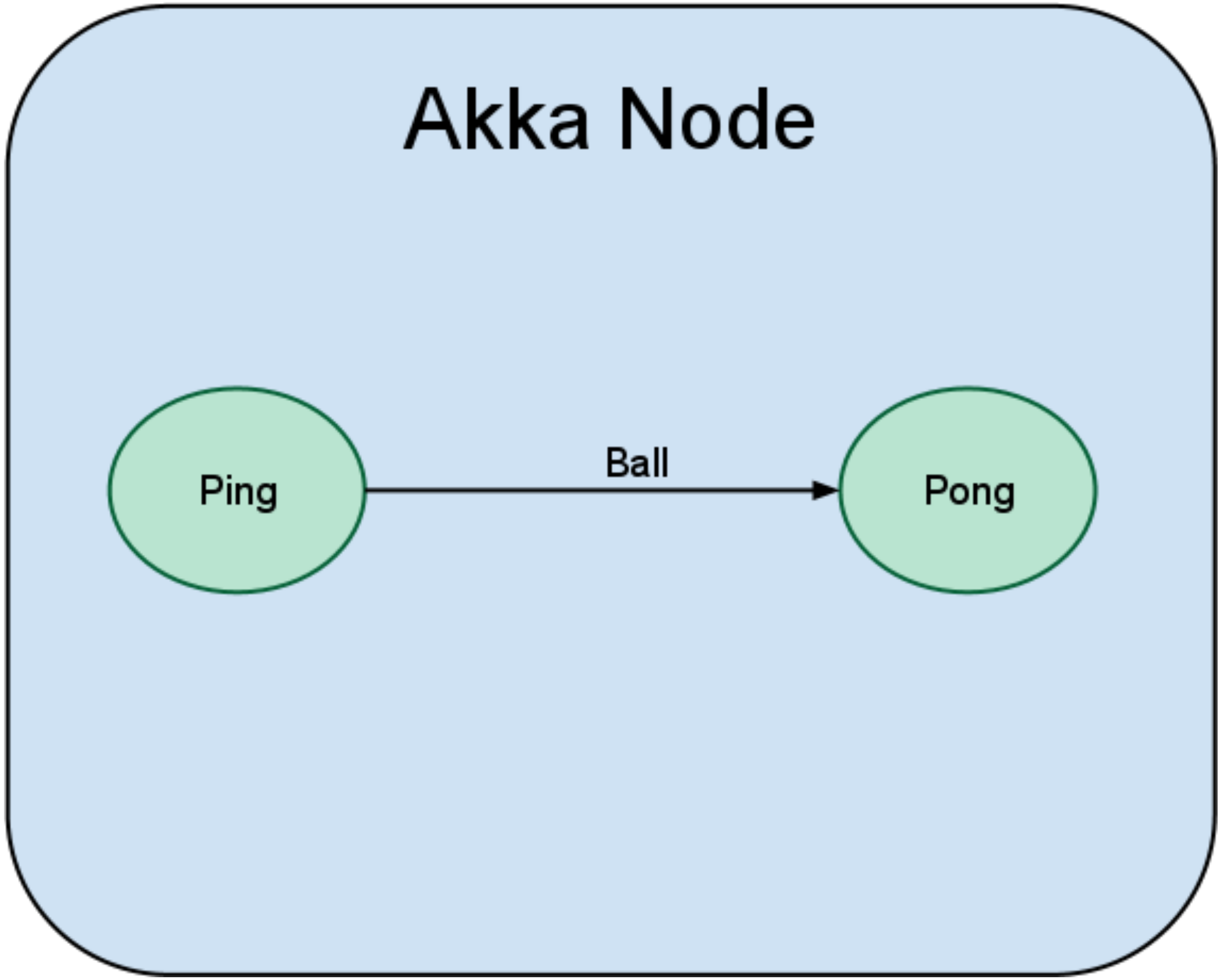
- Subscription based cluster membership service
- Highly available cluster registry for actors
- Automatic cluster-wide deployment
- Automatic replication with fail-over
- Transparent and user configurable load balancing
- Transparent adaptive cluster rebalancing
- Leader election
- Durable Mailboxes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

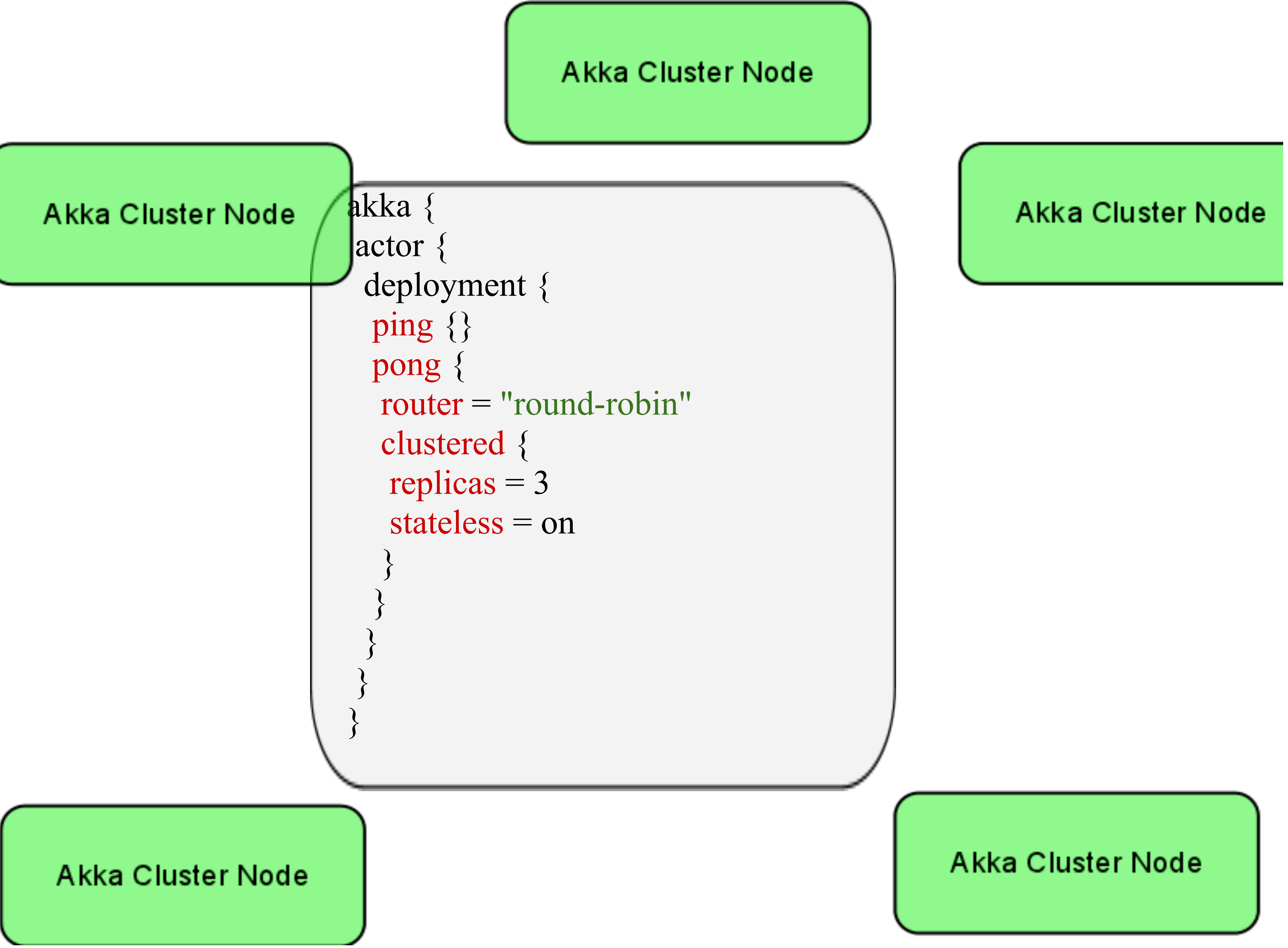
Clustering of Stateless Actors

Classic Example

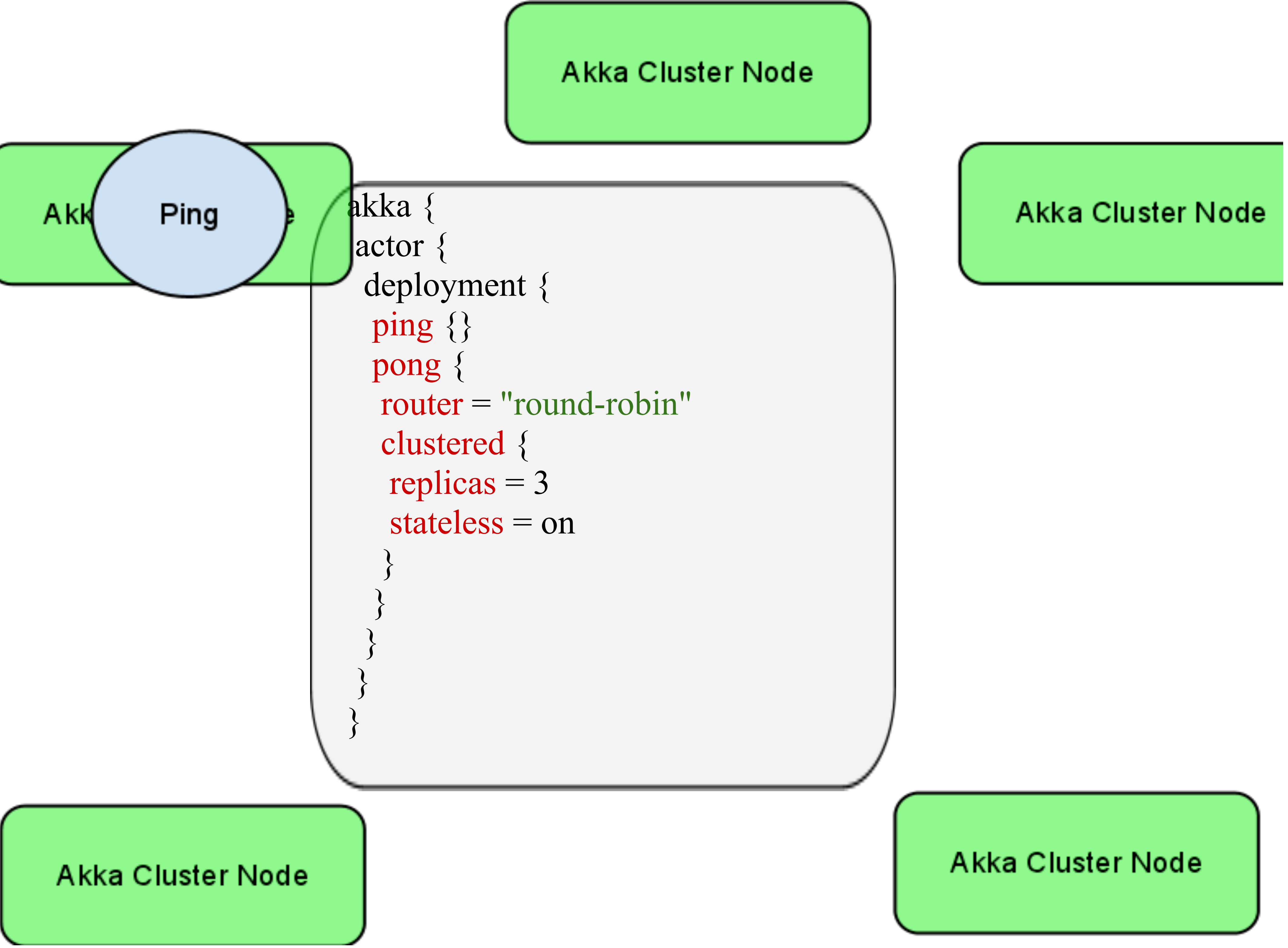
```
val ping = actorOf[Ping]("ping")
val pong = actorOf[Pong]("ping")

ping ! Ball(pong)
```

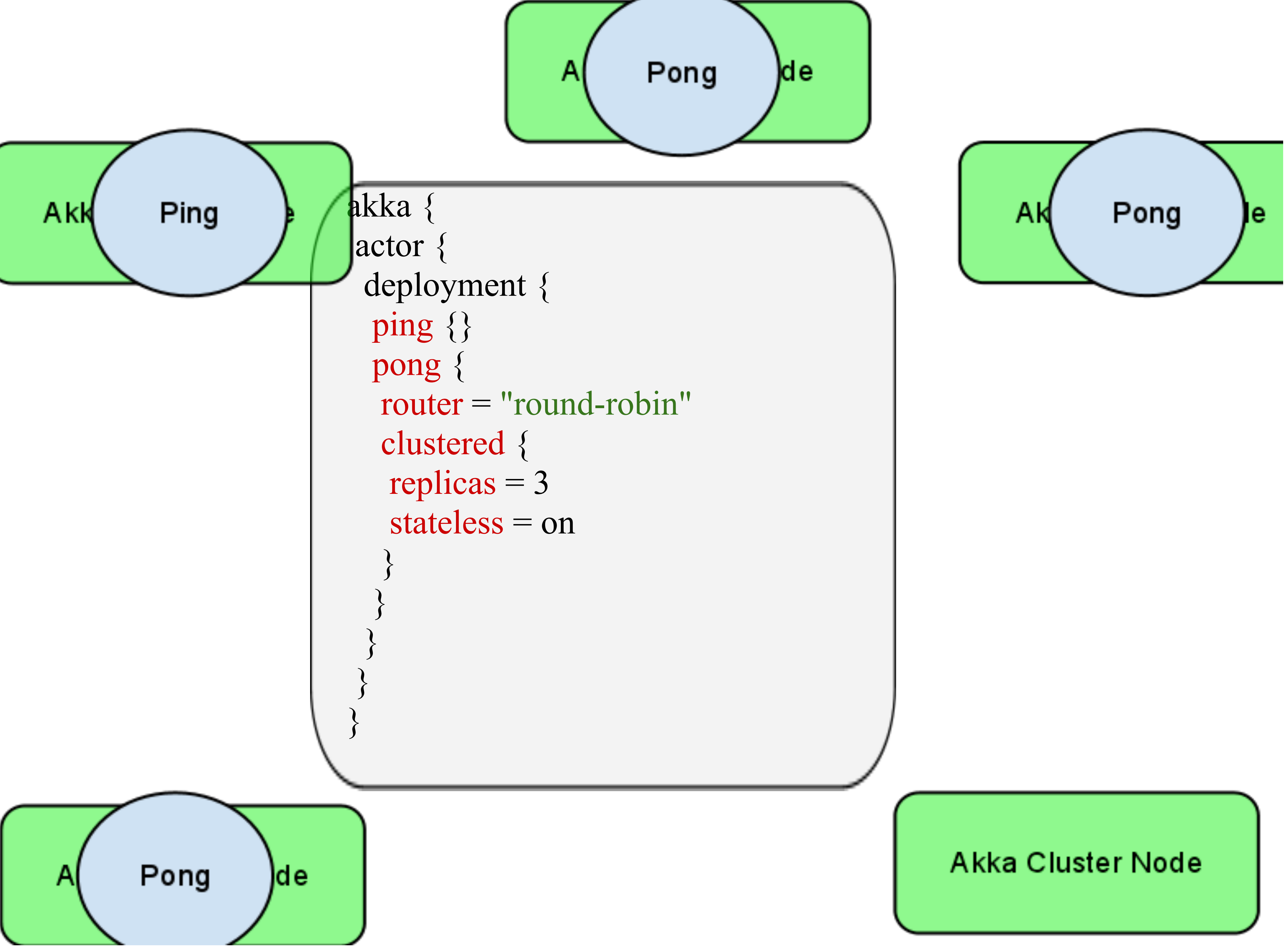

Akka Node
Ping
Ball
Pong



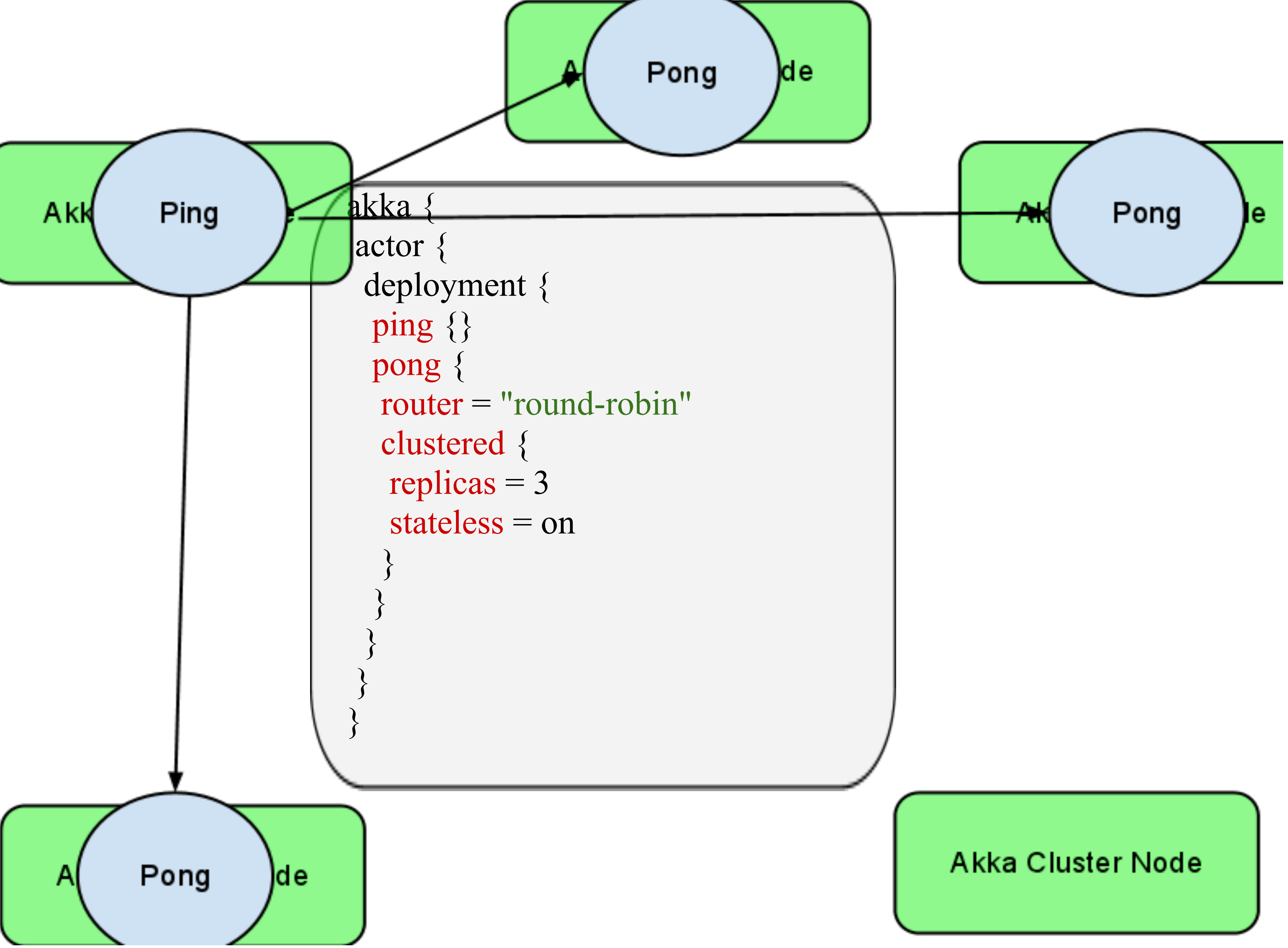
```
akka {
 actor {
  deployment {
   ping {}
   pong {
    router = "round-robin"
    clustered {
     replicas = 3
     stateless = on
    }
   }
  }
 }
}
```



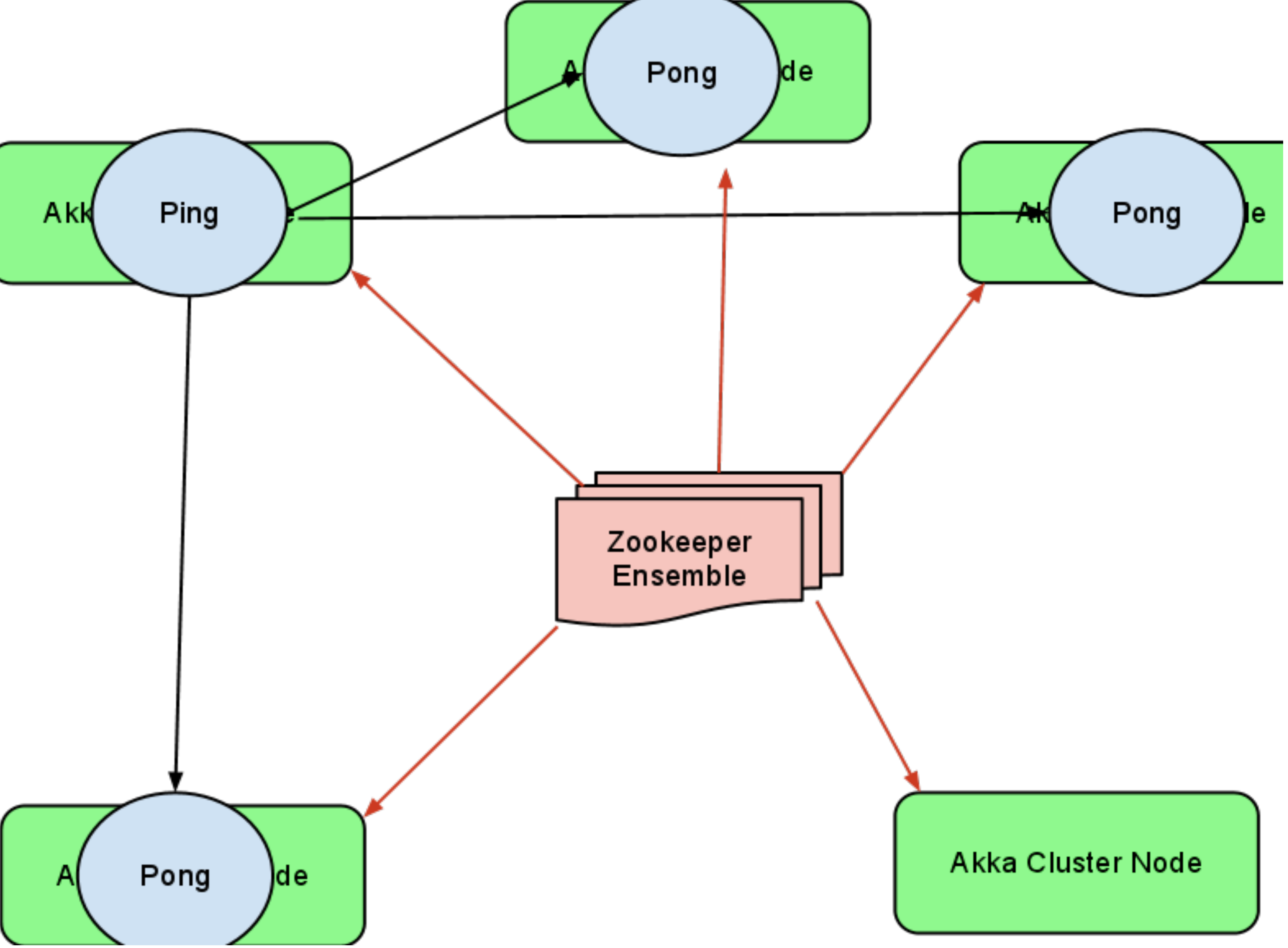
```
akka {
 actor {
  deployment {
   ping {}
   pong {
    router = "round-robin"
    clustered {
     replicas = 3
     stateless = on
    }
   }
  }
 }
}
```

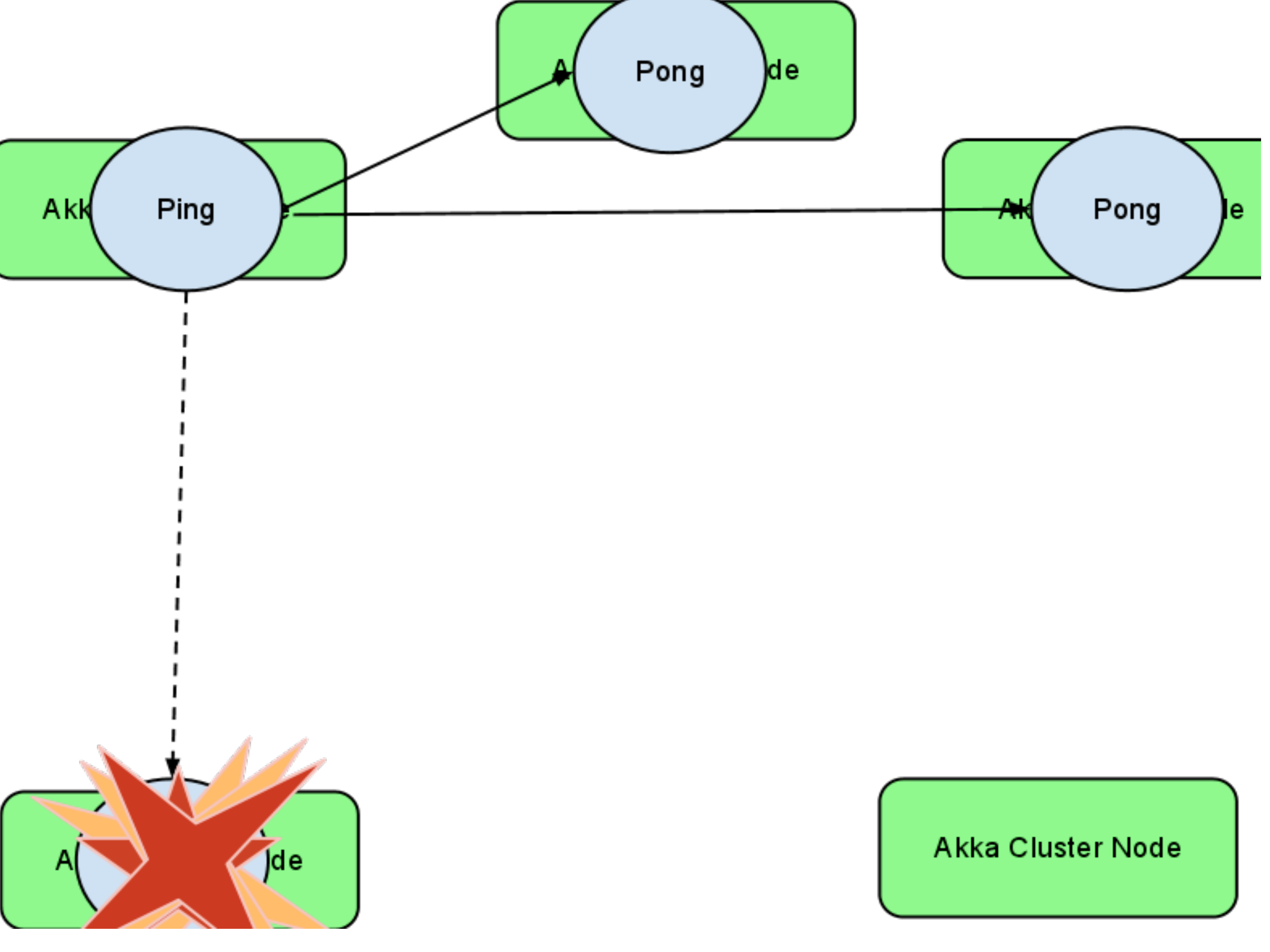


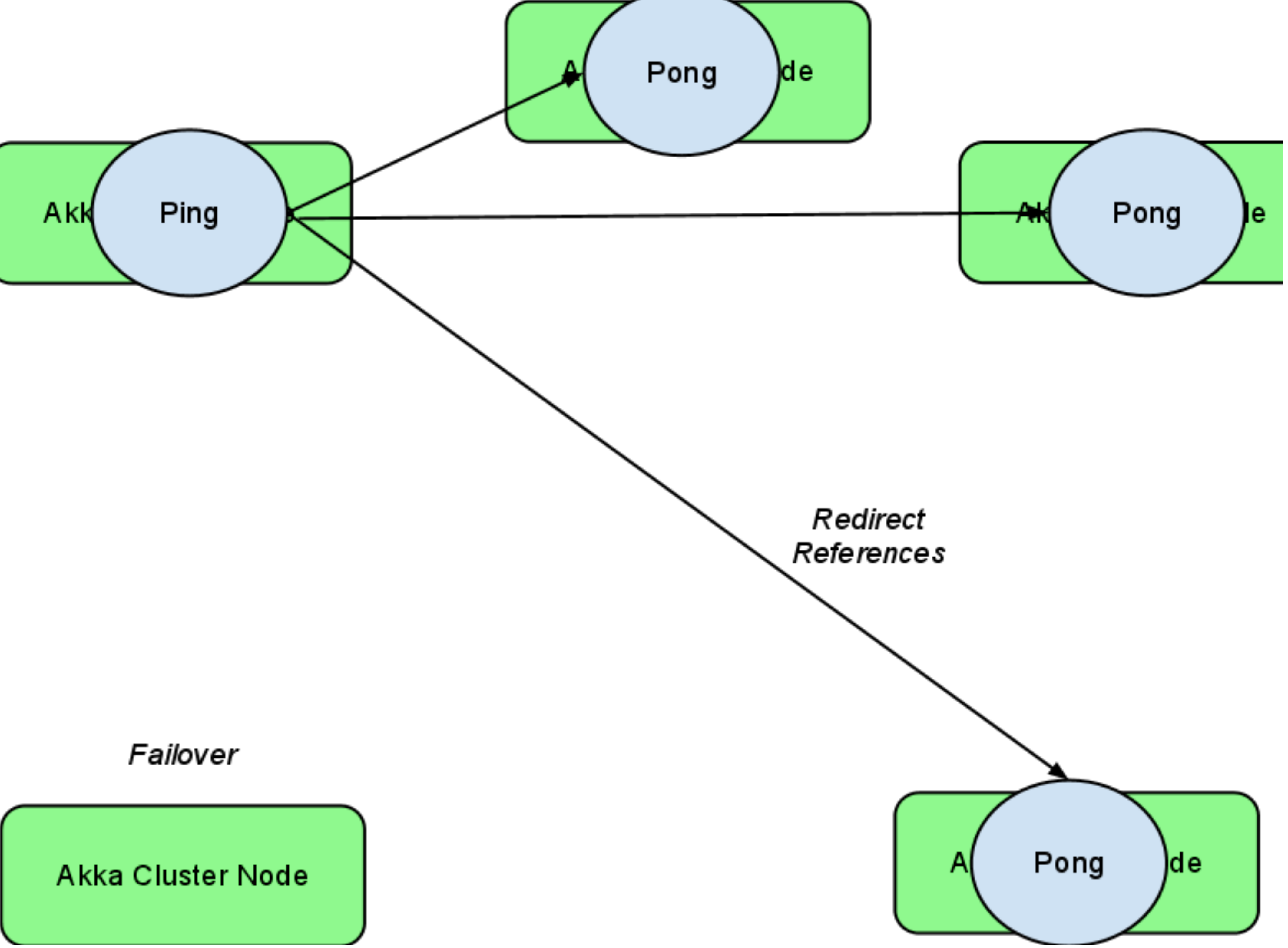
```
akka {
 actor {
  deployment {
   ping {}
   pong {
    router = "round-robin"
    clustered {
     replicas = 3
     stateless = on
    }
   }
  }
 }
}
```



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


Pong
Ping
Pong
Zookeeper Ensemble
Pong
Akka Cluster Node


Pong
Ping
Pong
Akka Cluster Node


Pong
Akk
Ping
Pong
Redirect References
Failover
Akka Cluster Node
Pong

Clustering of Stateful Actors

Replication

1

Transaction Log

Deployment Configuration

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

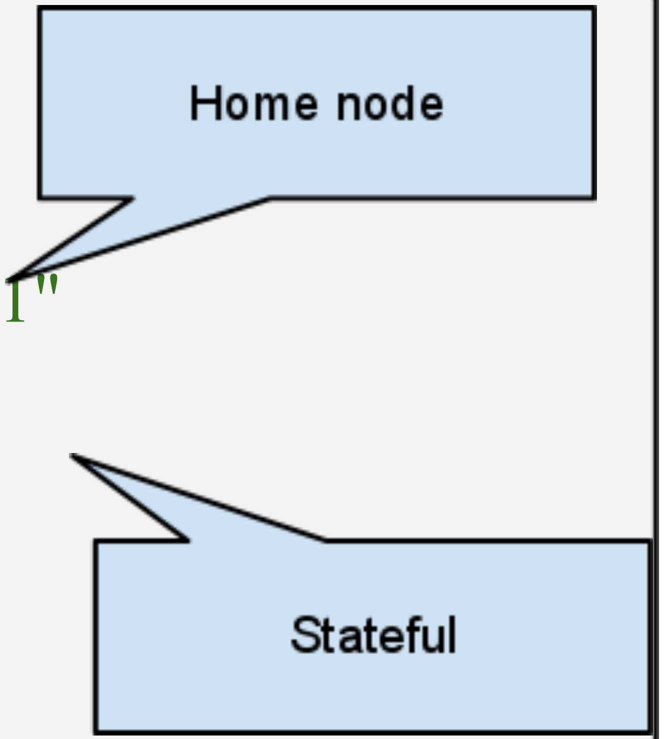
Deployment Configuration

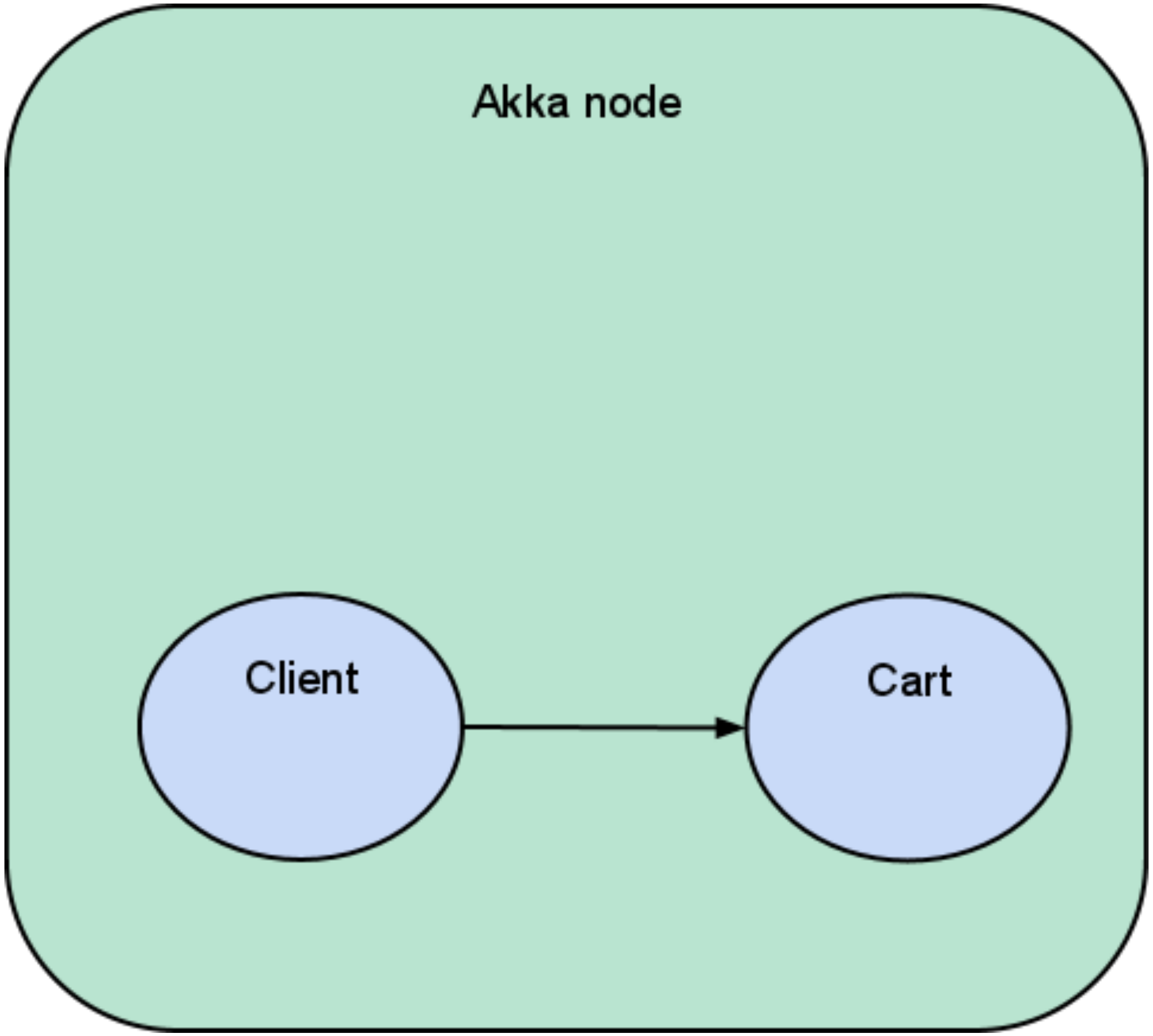
```
akka {
 actor {
  deployment {
   carts {
    clustered {
     home = "node:test-node-1"
     stateless = off
    }
   }
  }
 }
}
```

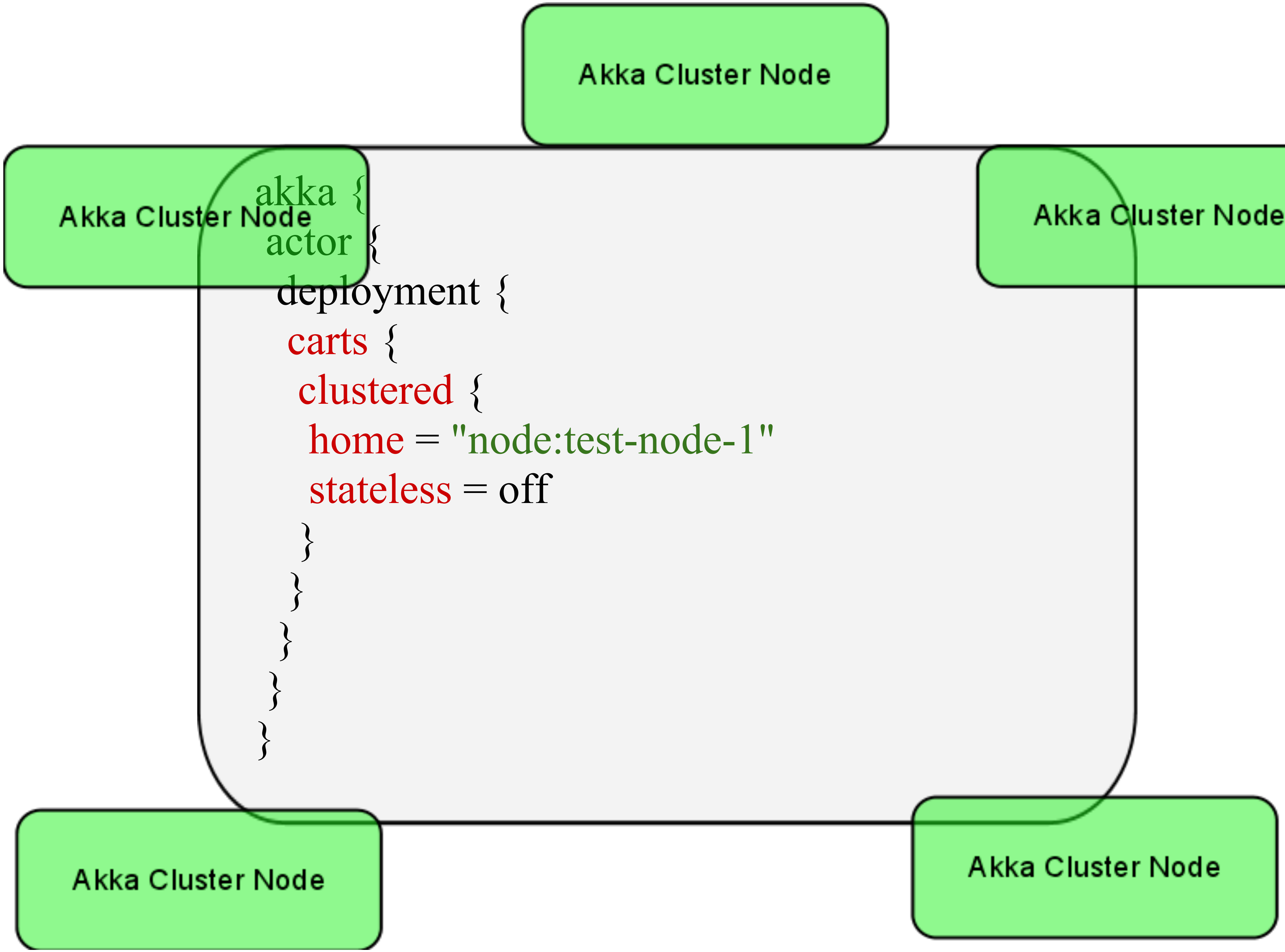
Home node

Deployment Configuration

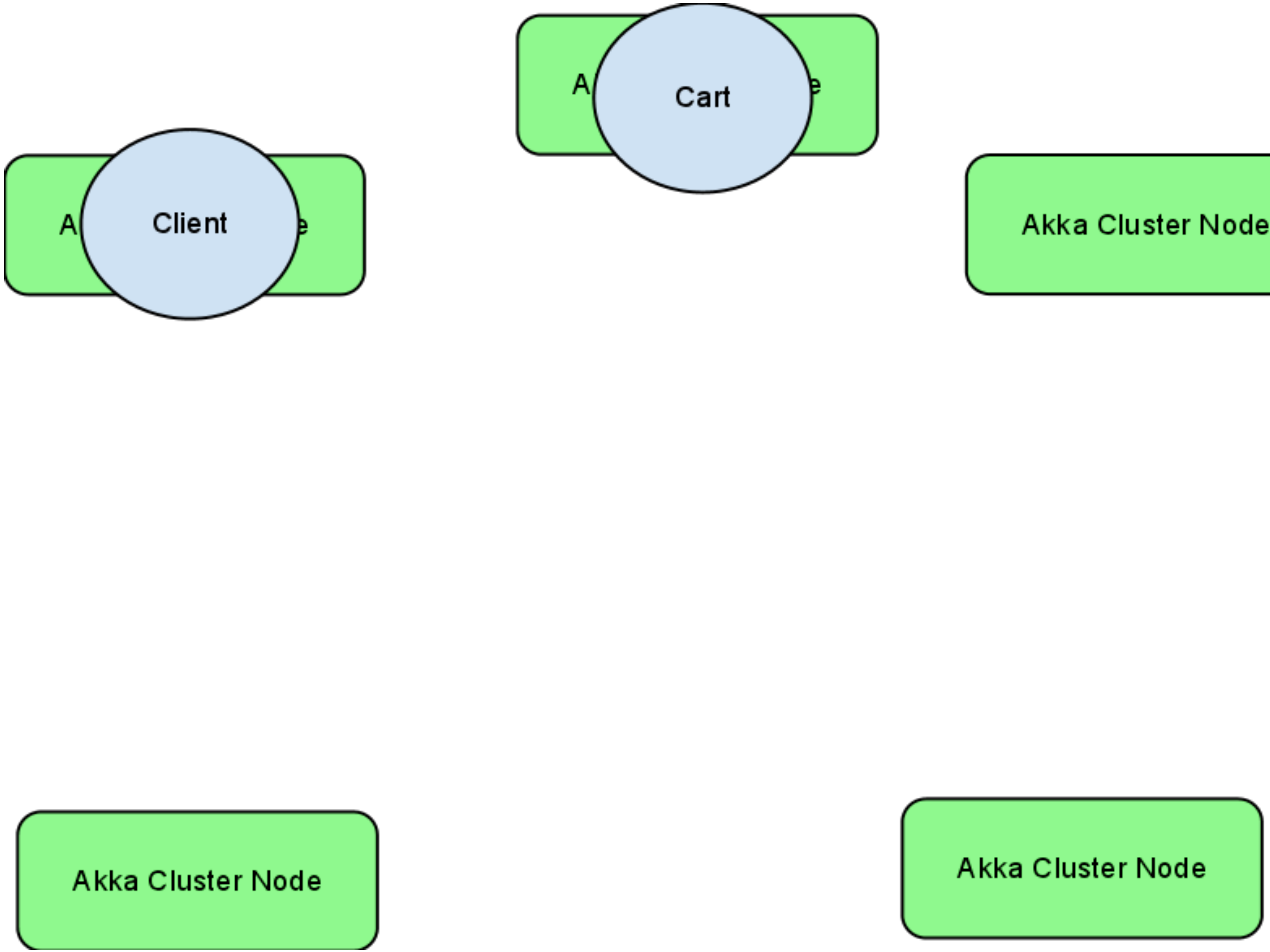
```
akka {
 actor {
  deployment {
   carts {
    clustered {
     home = "node:test-node-1"
     stateless = off
    }
   }
  }
 }
}
```

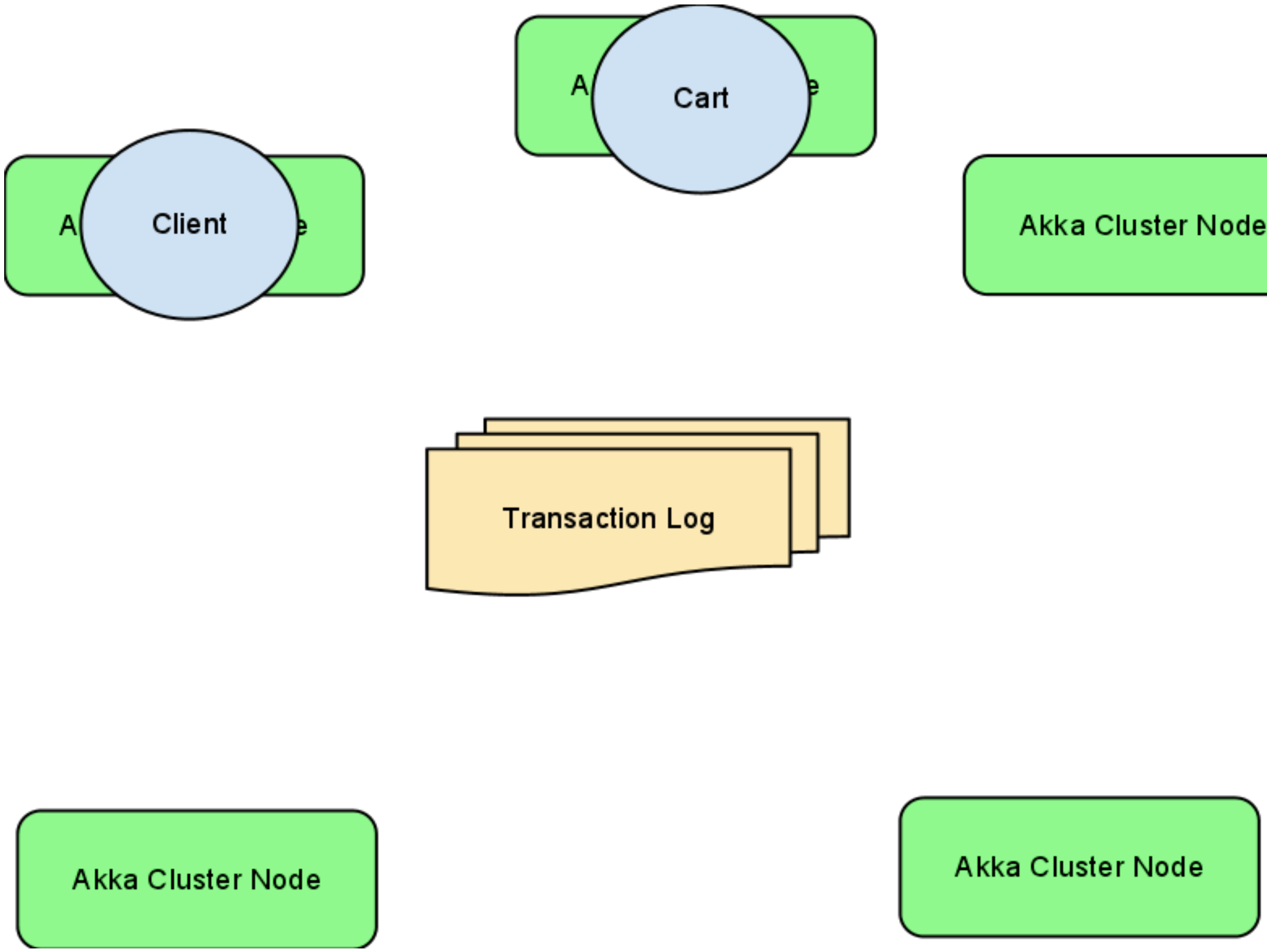


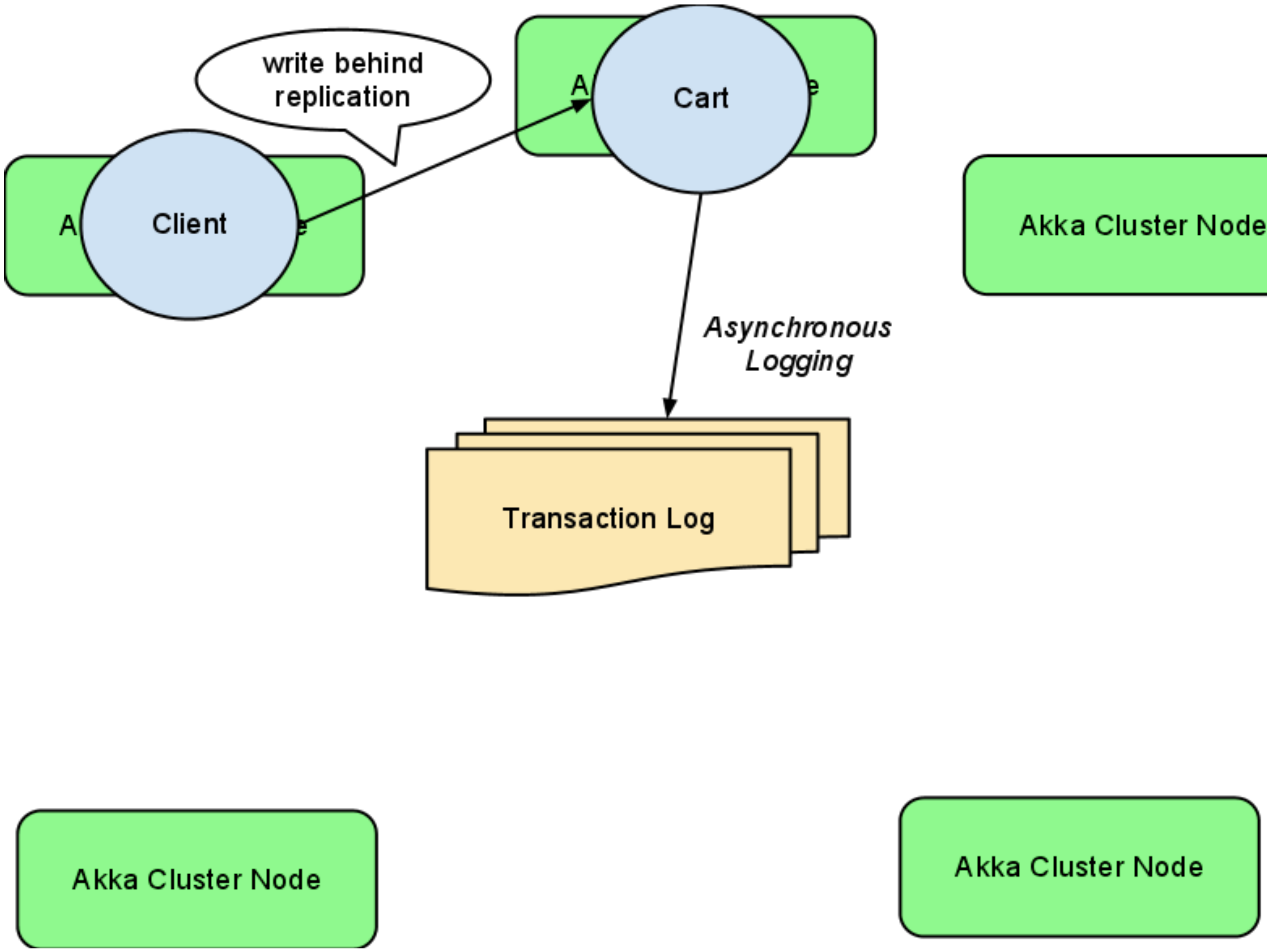

Akka node
Client
Cart

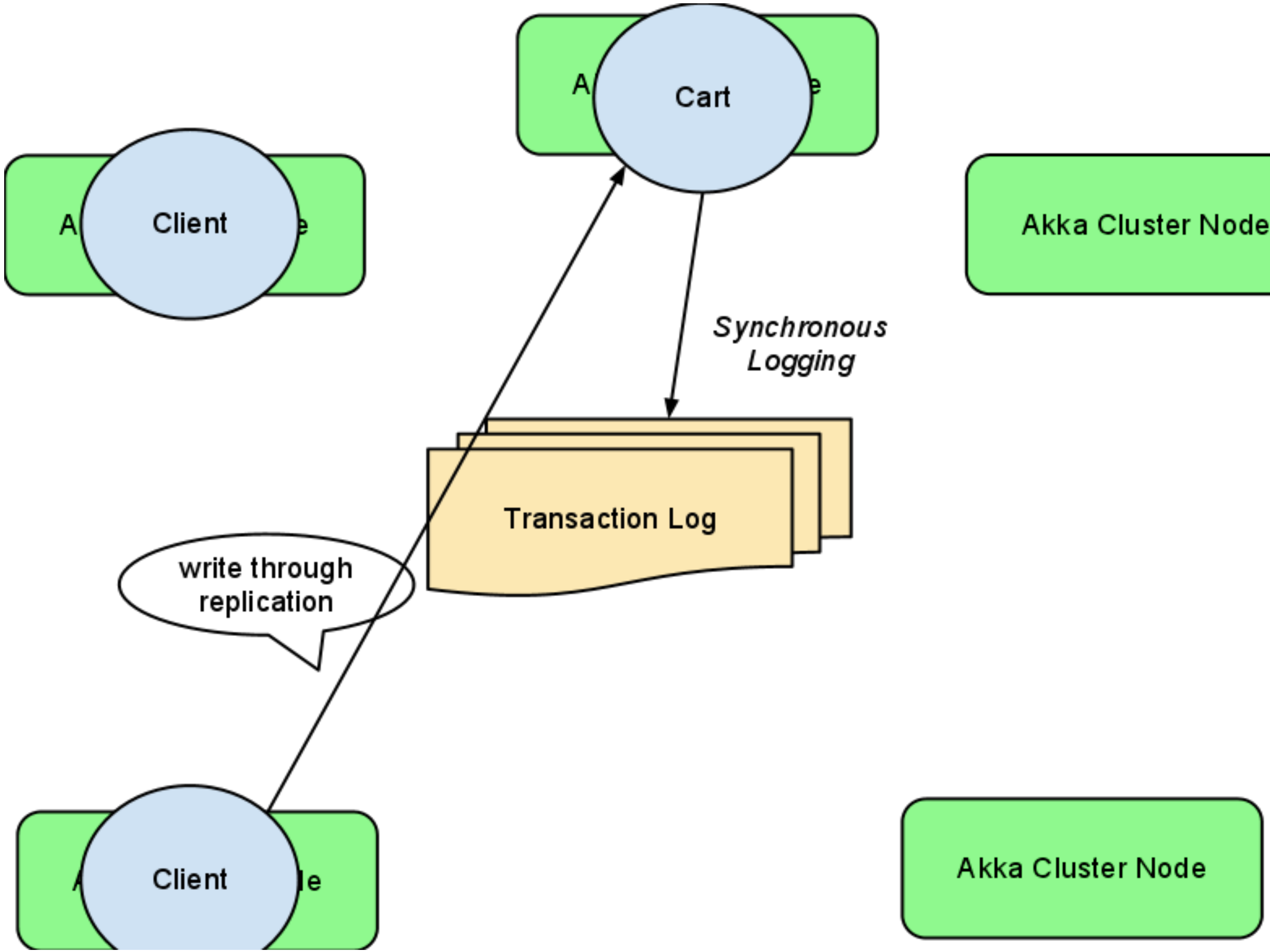


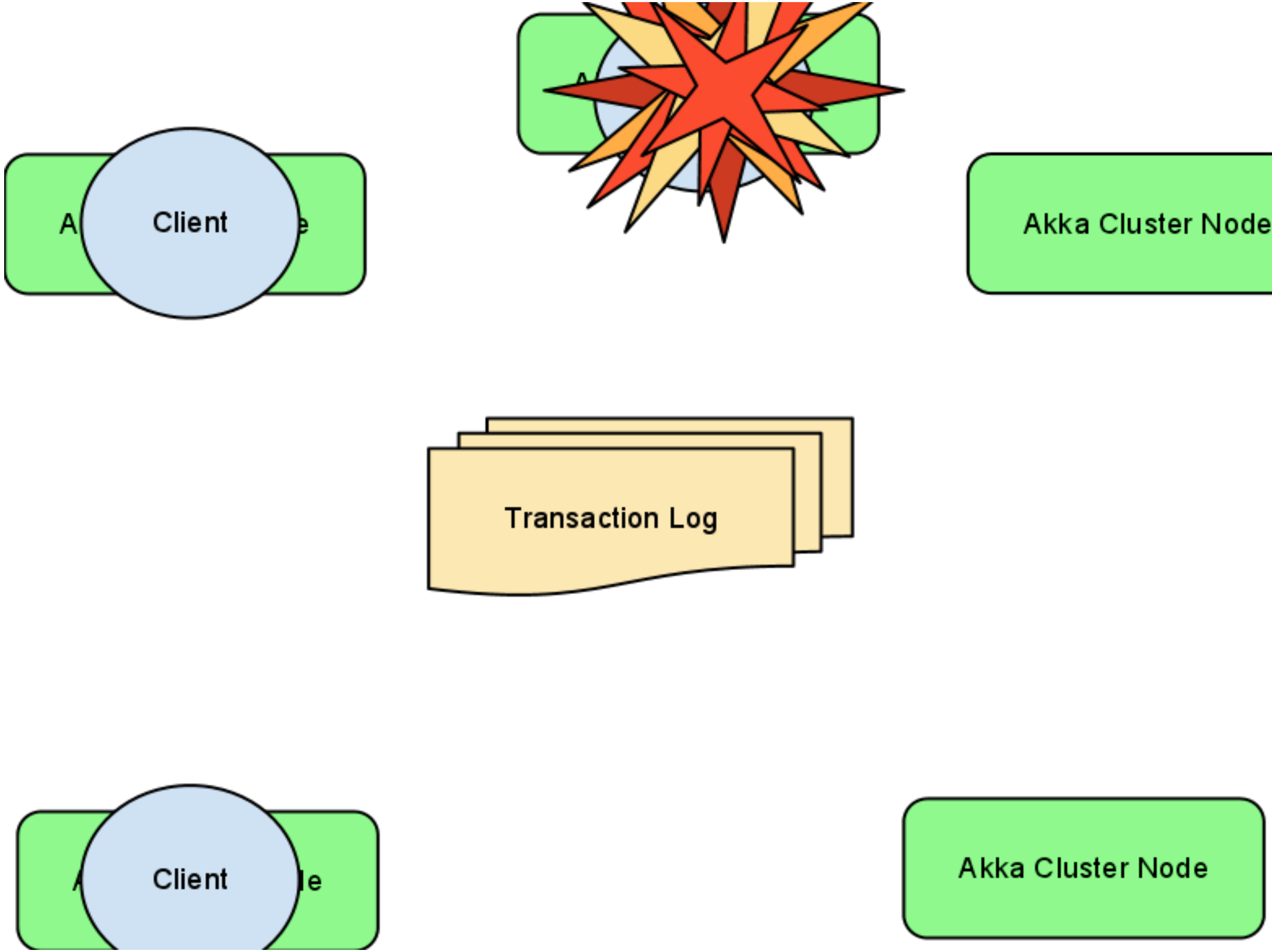
```
akka {
 actor {
  deployment {
   carts {
    clustered {
     home = "node:test-node-1"
     stateless = off
    }
   }
  }
 }
}
```

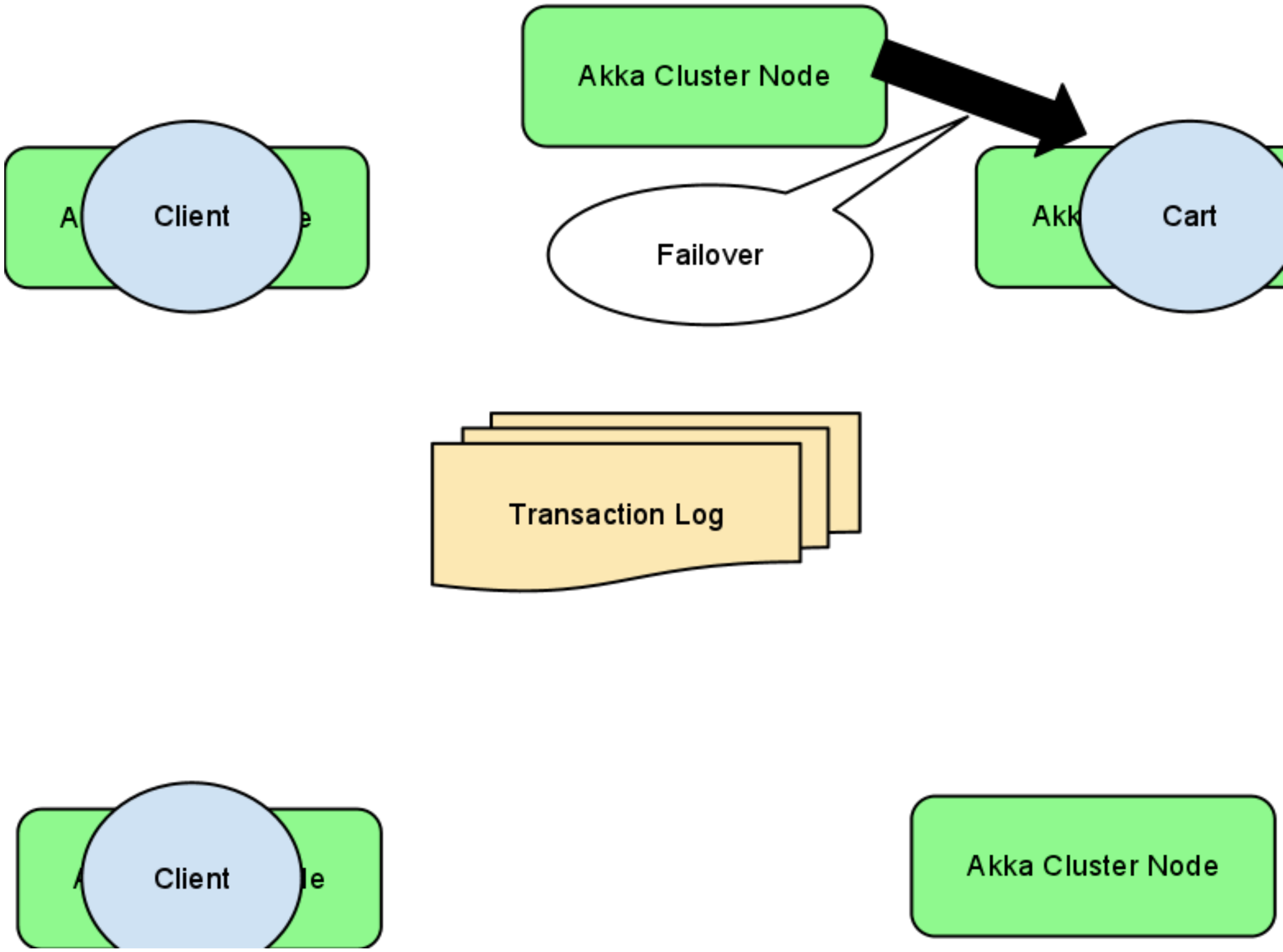

Cart
Client
Akka Cluster Node
Akka Cluster Node
Akka Cluster Node

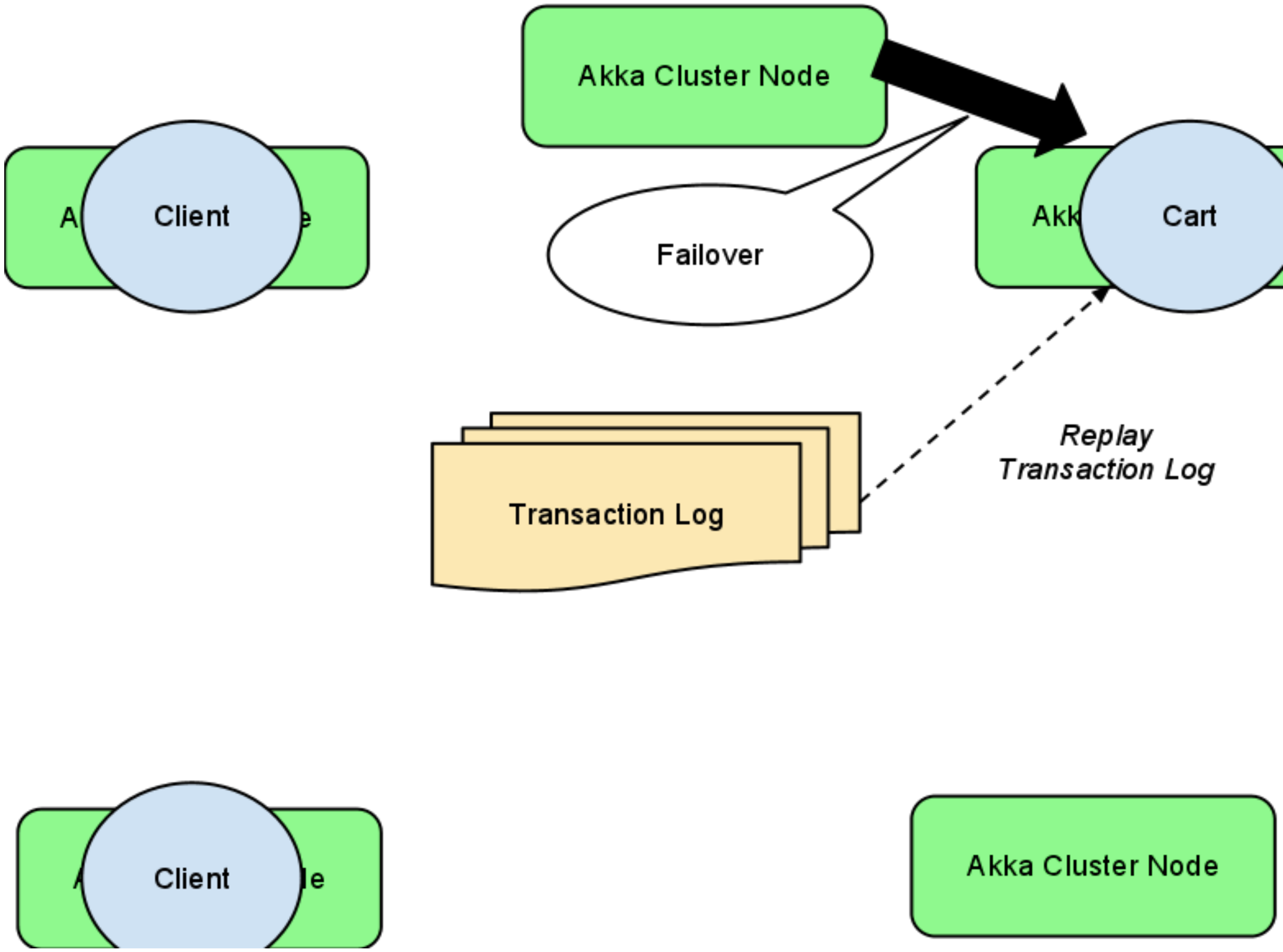

Cart
Client
Akka Cluster Node
Transaction Log
Akka Cluster Node
Akka Cluster Node

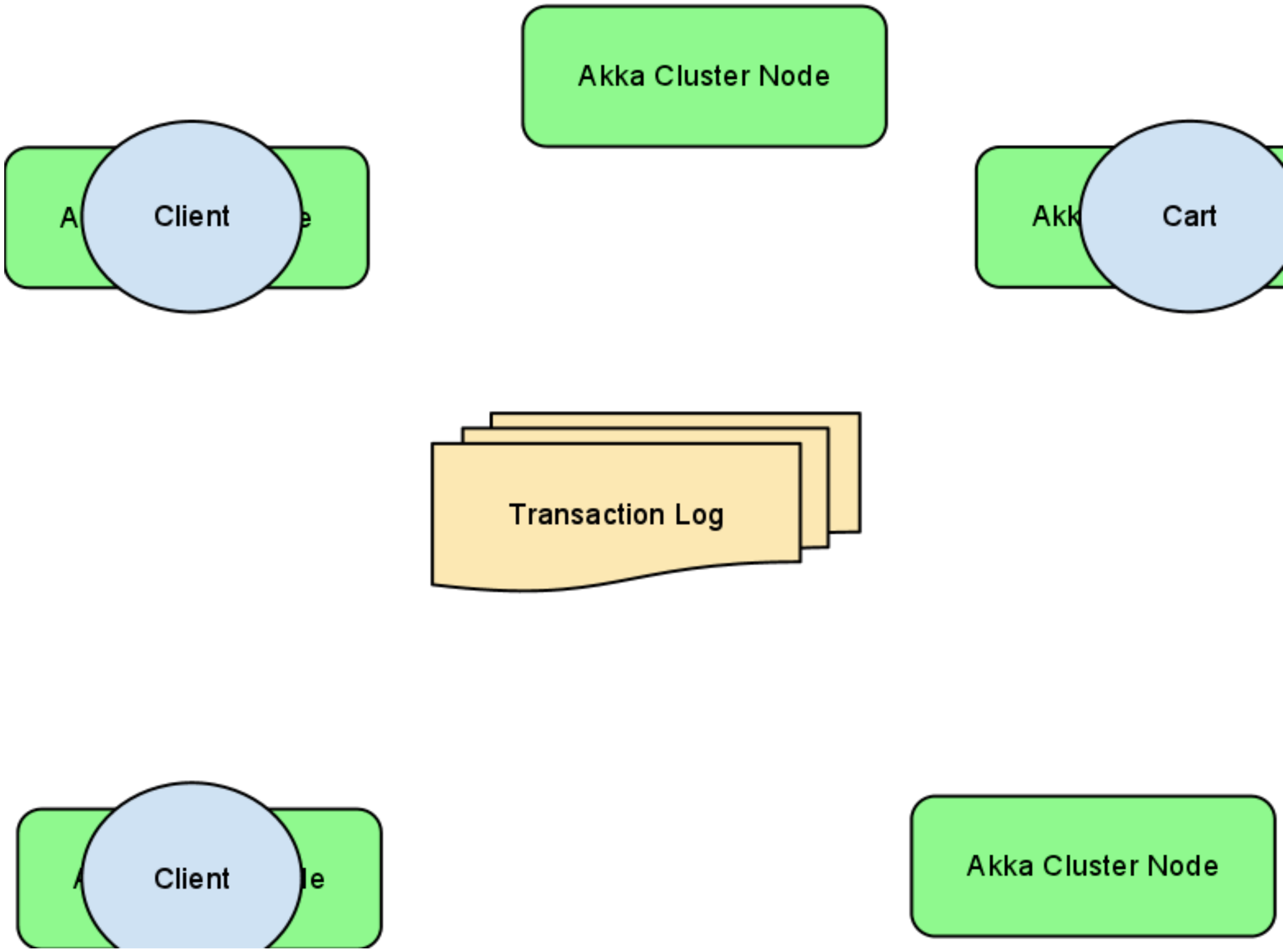

write behind replication
Cart
Client
Akka Cluster Node
Asynchronous Logging
Transaction Log
Akka Cluster Node
Akka Cluster Node

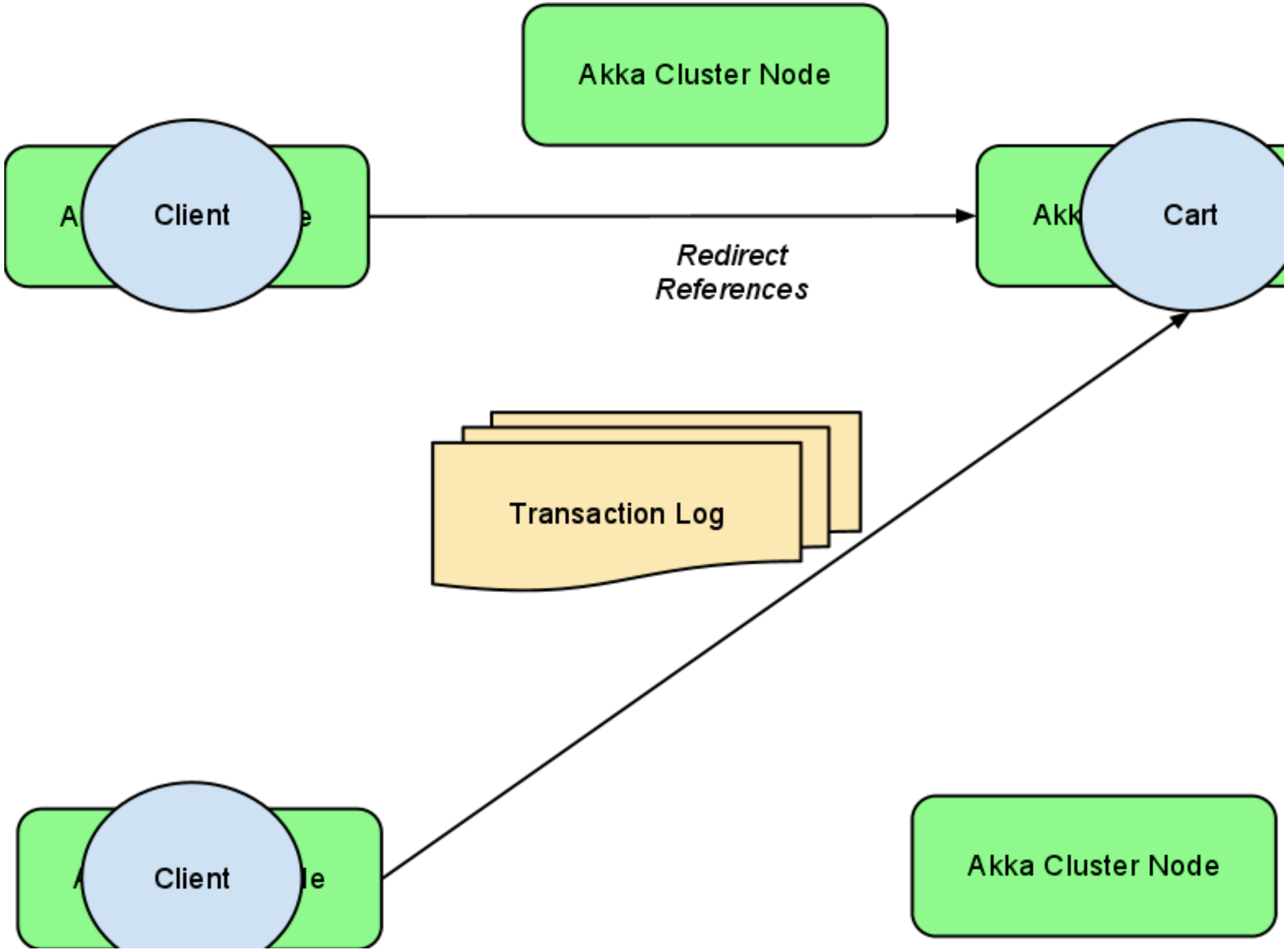

Cart
A
e
Client
A
e
Akka Cluster Node
Synchronous
Logging
Transaction Log
write through
replication
Client
A
le
Akka Cluster Node


Client
Akka Cluster Node
Transaction Log
Client
Akka Cluster Node


Akka Cluster Node
Client
Failover
Cart
Transaction Log
Client
Akka Cluster Node

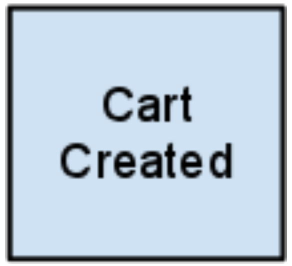

Akka Cluster Node
Client
Failover
Cart
Akk
Transaction Log
Replay Transaction Log
Client
Akka Cluster Node


Akka Cluster Node
Client
Cart
Transaction Log
Client
Akka Cluster Node

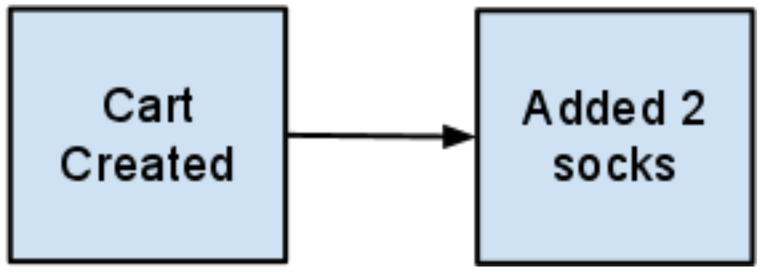

Akka Cluster Node
Client
Akk
Cart
Redirect References
Transaction Log
Client
Akka Cluster Node

Transaction Log: Storing Messages

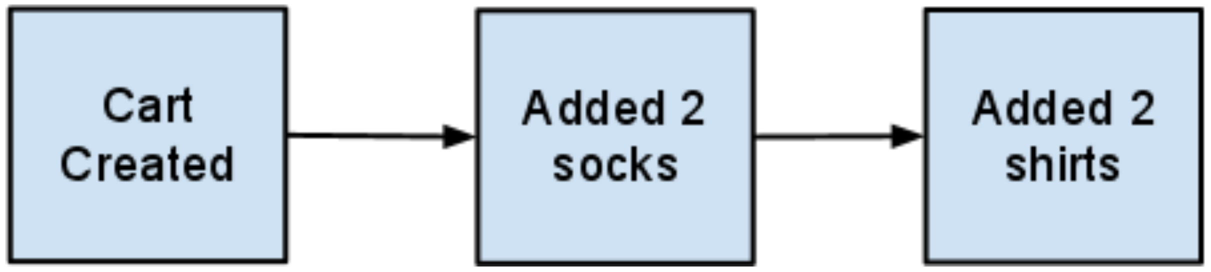
Transaction Log: Storing Messages



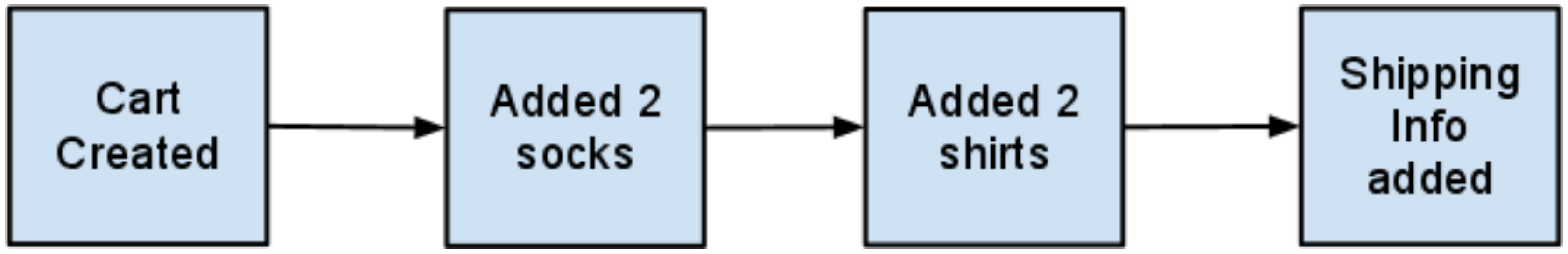
Transaction Log: Storing Messages



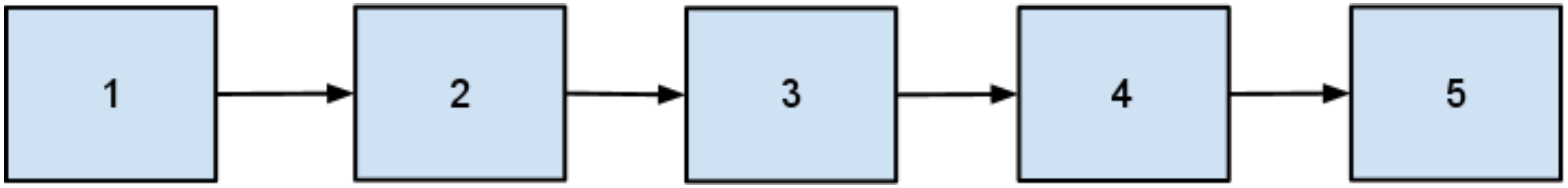
Transaction Log: Storing Messages



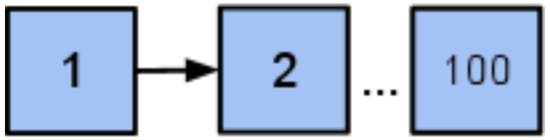
Transaction Log: Storing Messages



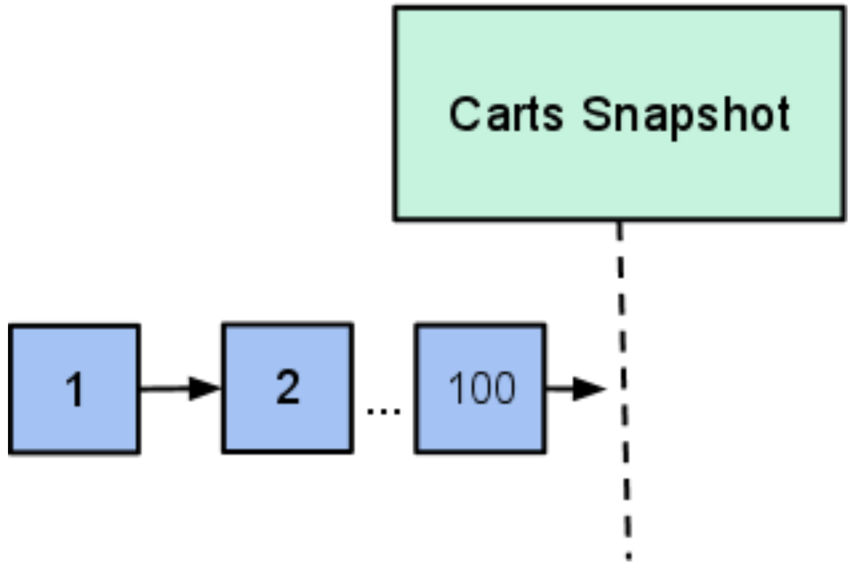
Transaction Log: Replaying



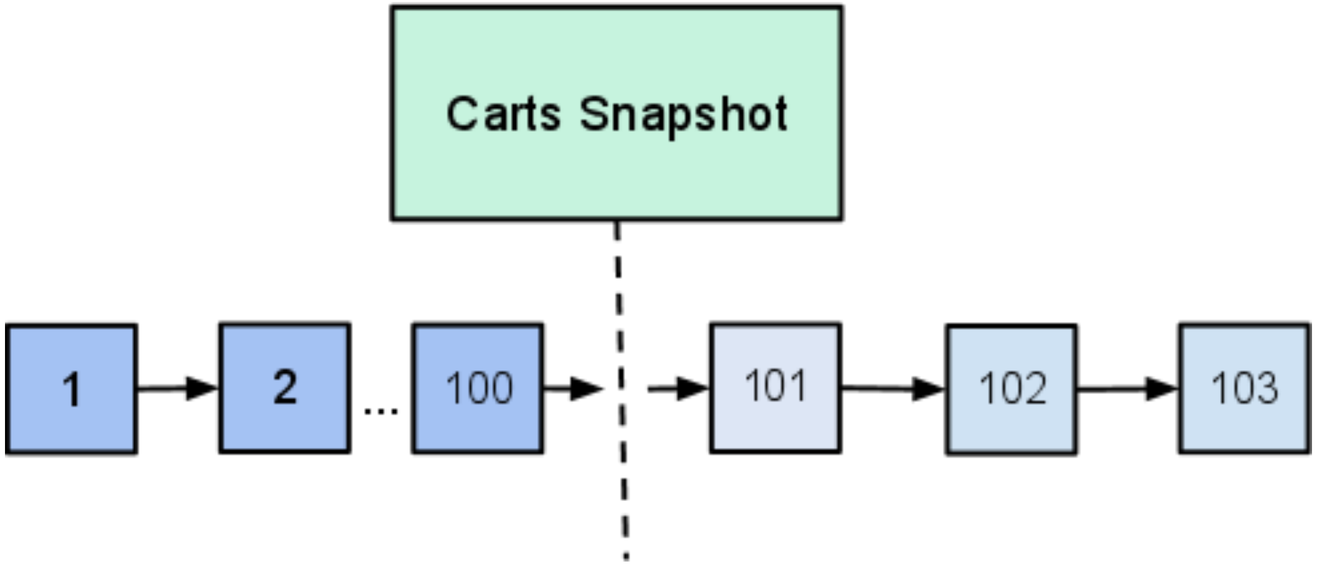
Transaction Log: Rolling Snapshot



Transaction Log: Rolling Snapshot



Transaction Log: Rolling Snapshot



Replication

2

Data Grid

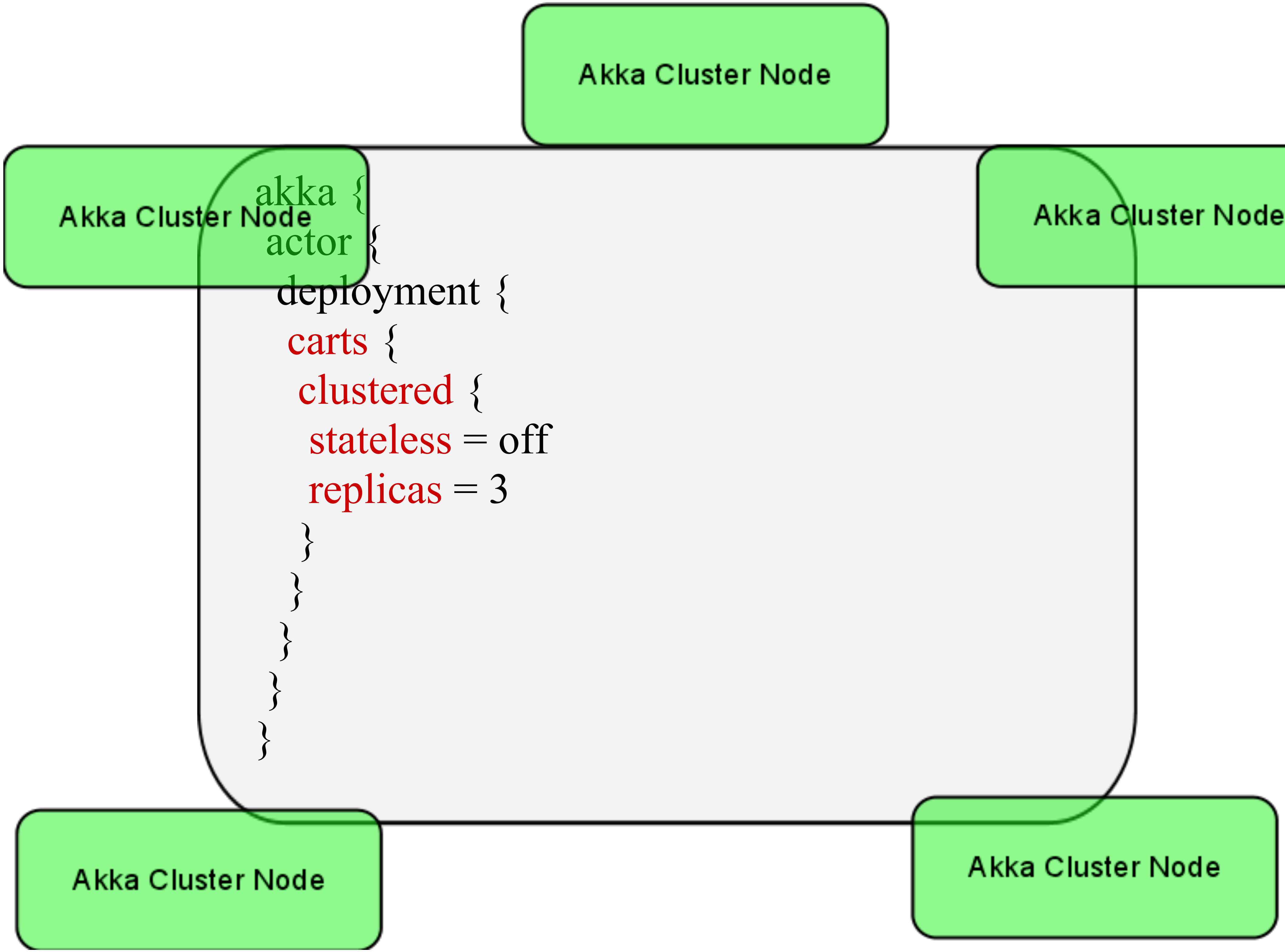
Data Grid

Actor State

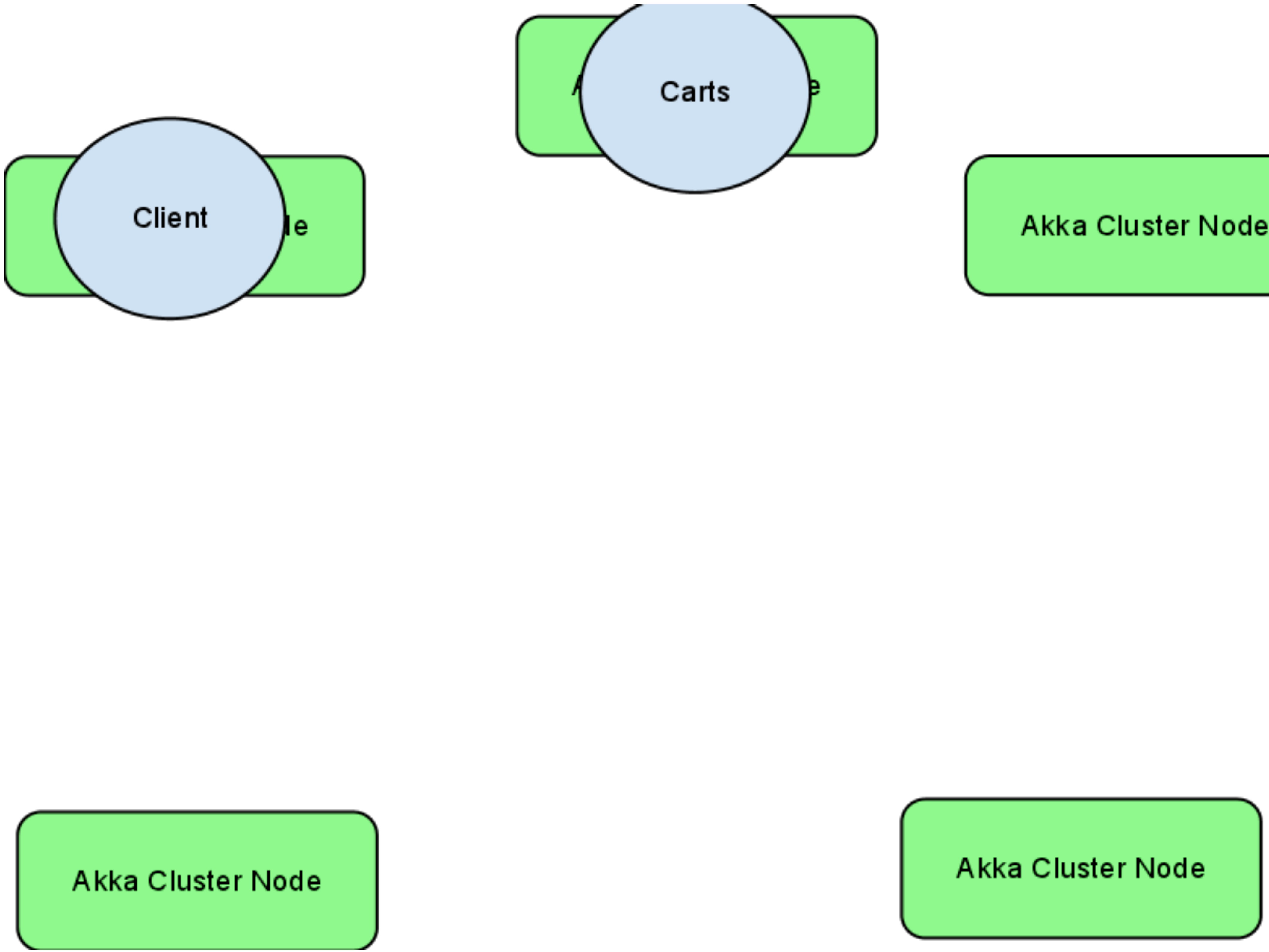
- Stored in 'external' Data Grid
- Transactional (distributed STM)
- Versioned
- Replicated
- Query-able

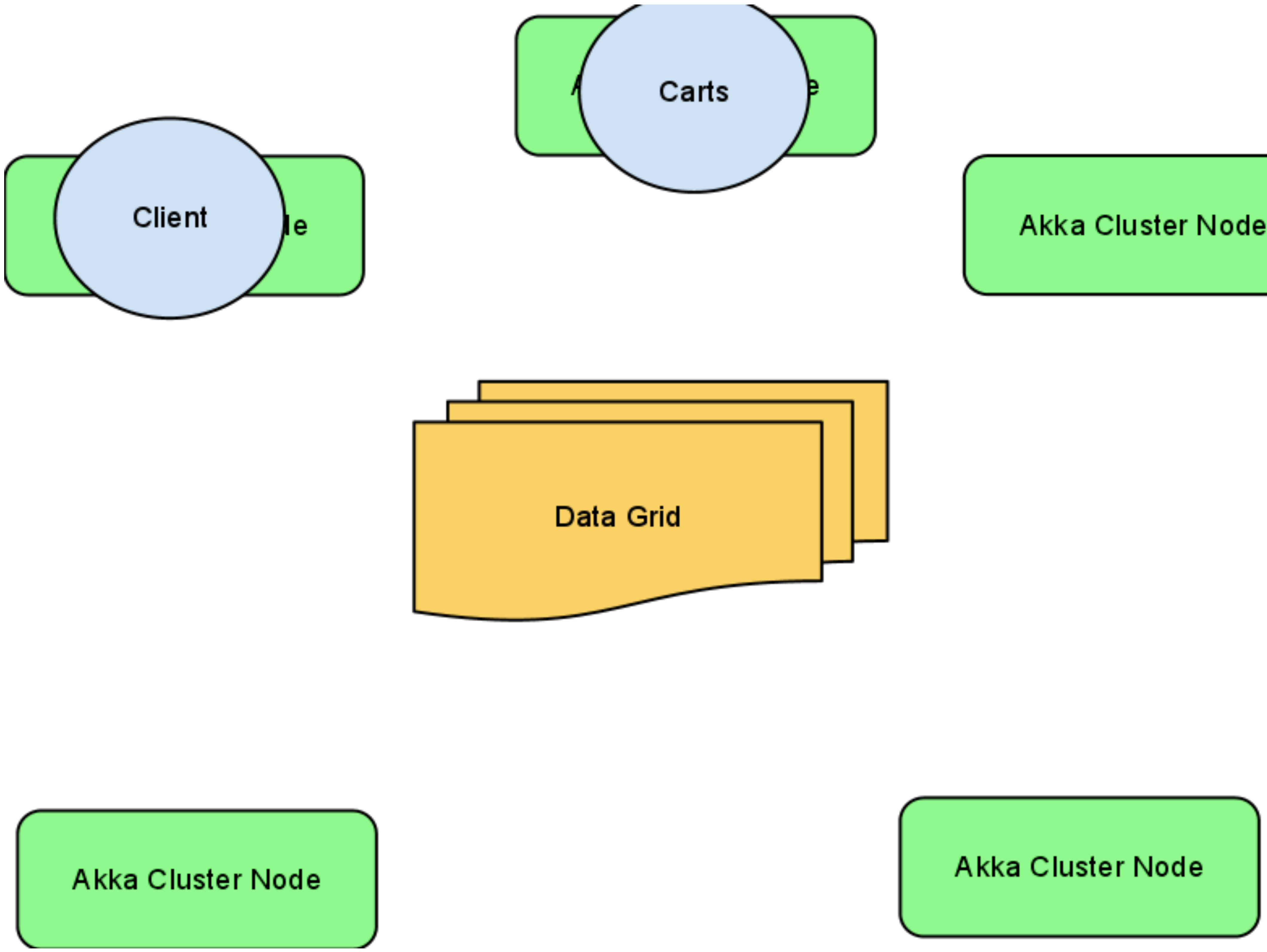
Implementations

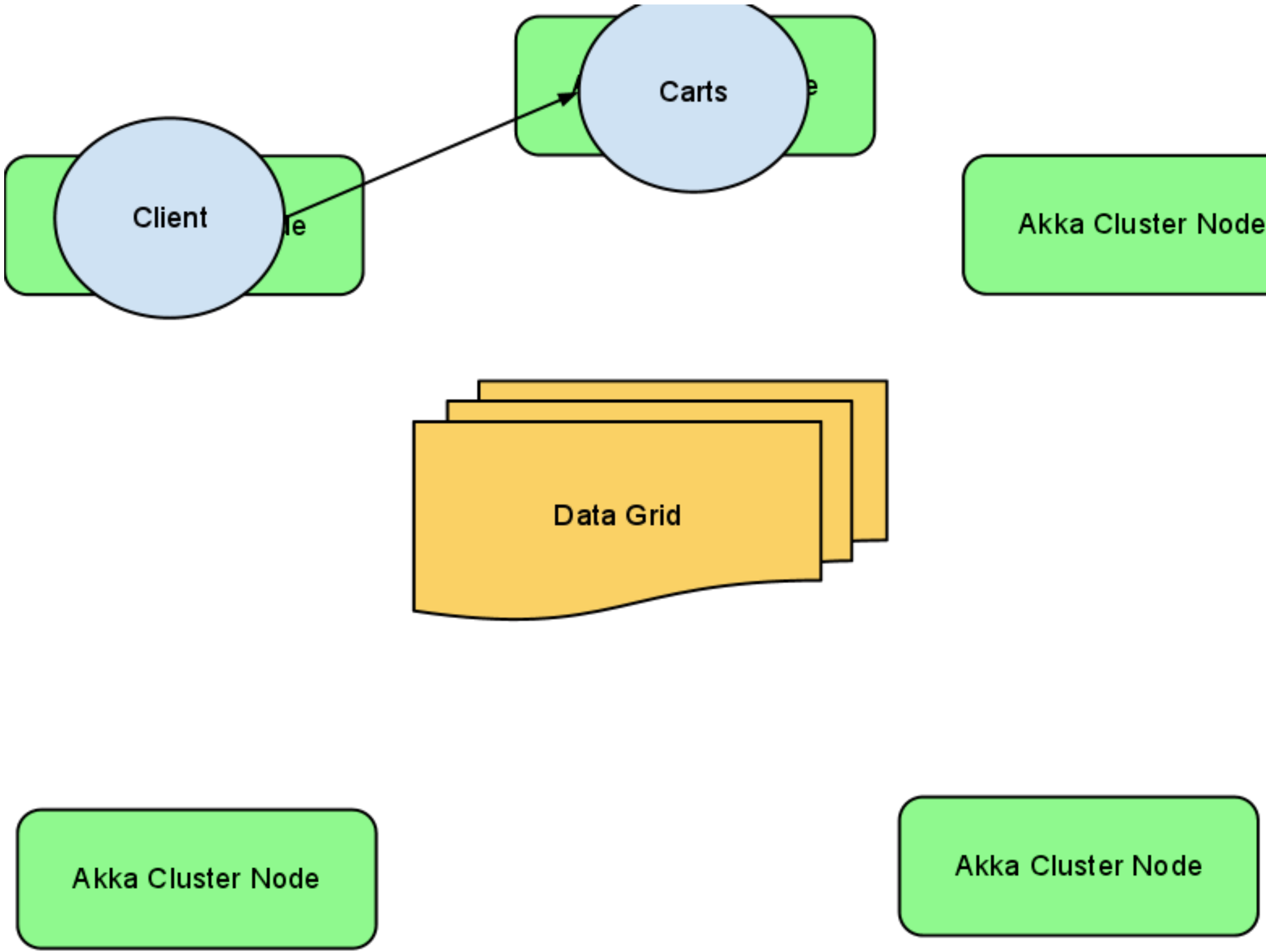
- Custom Akka Data Grid
- SPI for third parties

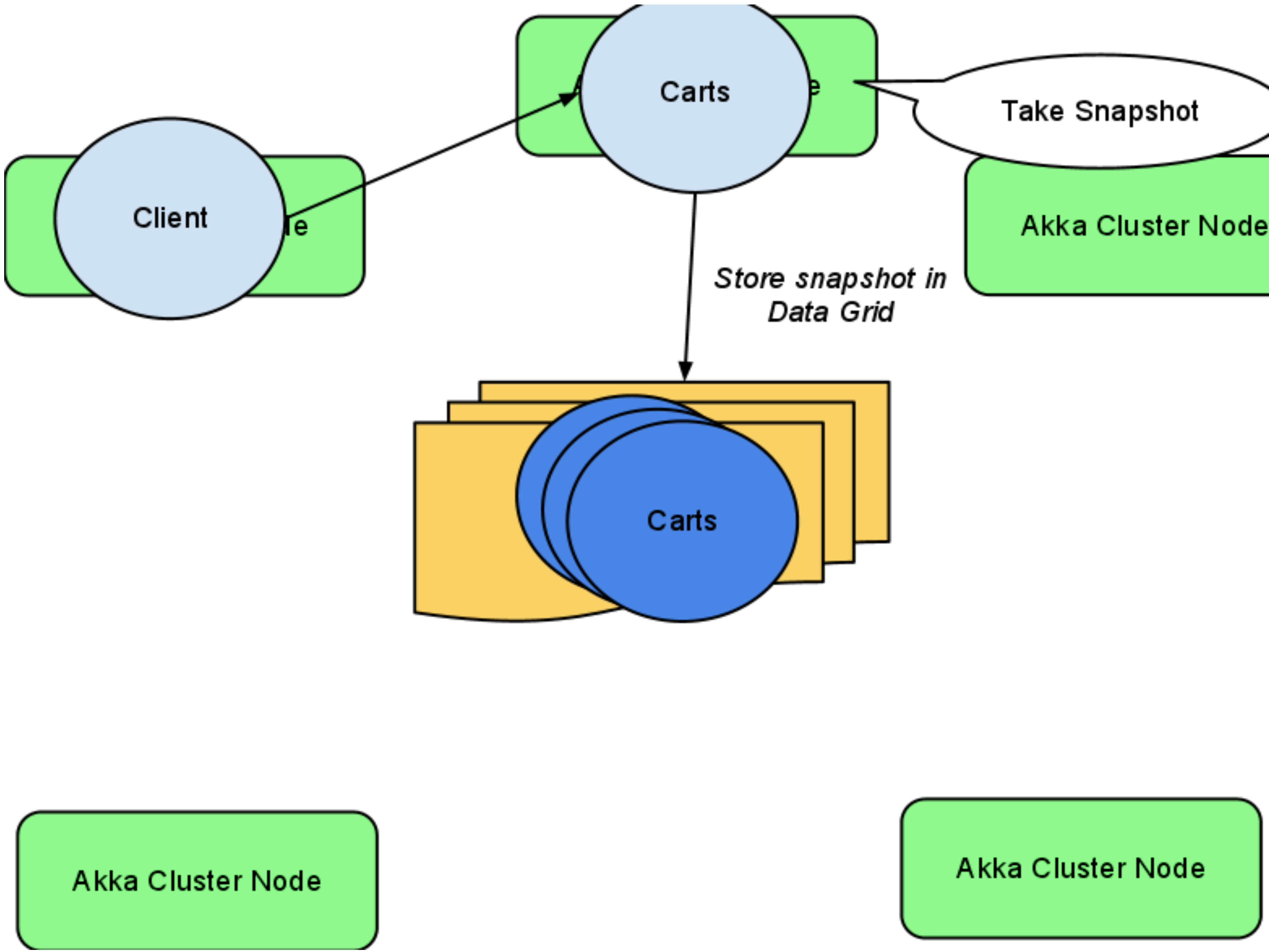


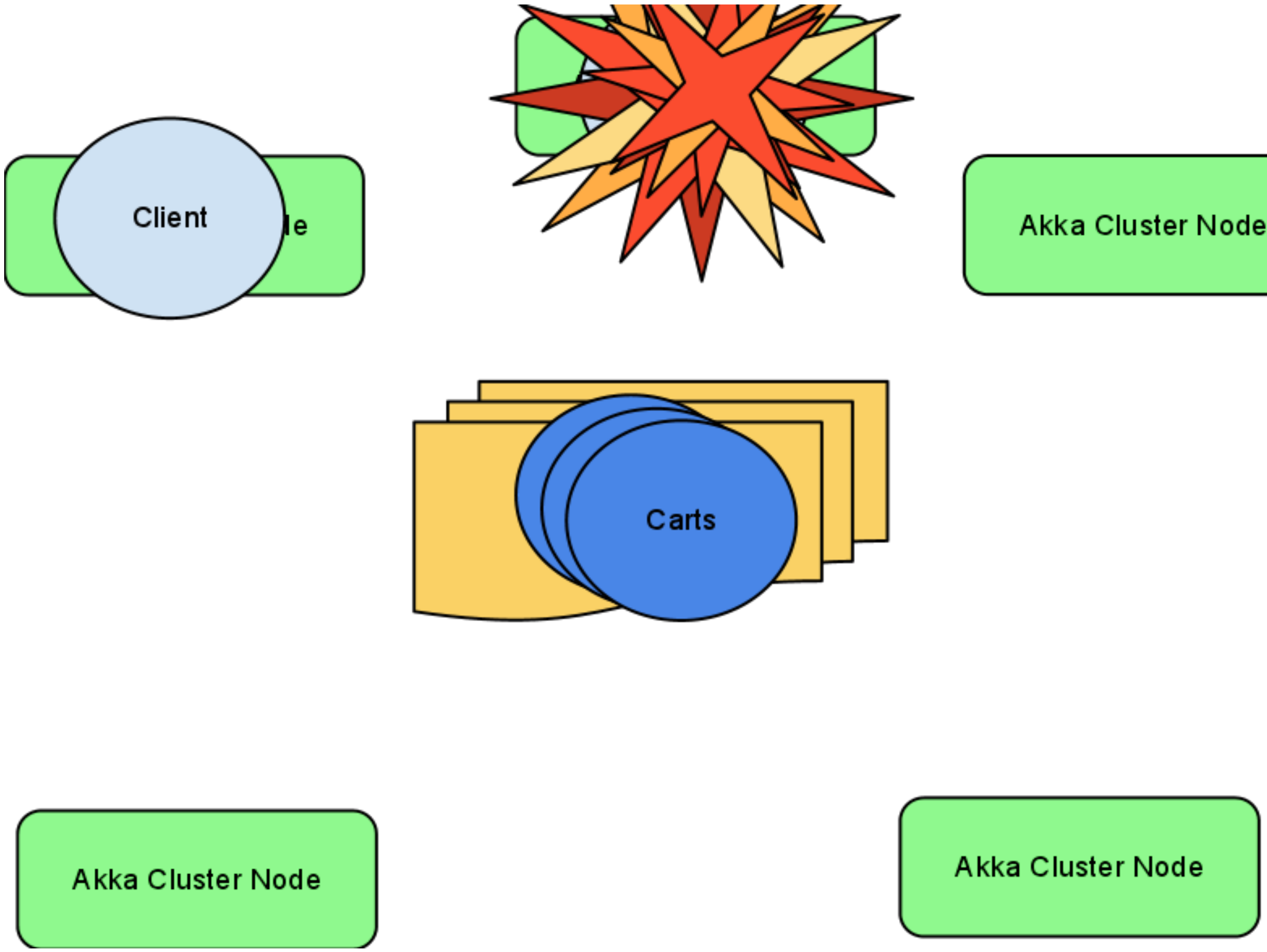
```
akka {
 actor {
  deployment {
   carts {
    clustered {
     stateless = off
     replicas = 3
    }
   }
  }
 }
}
```

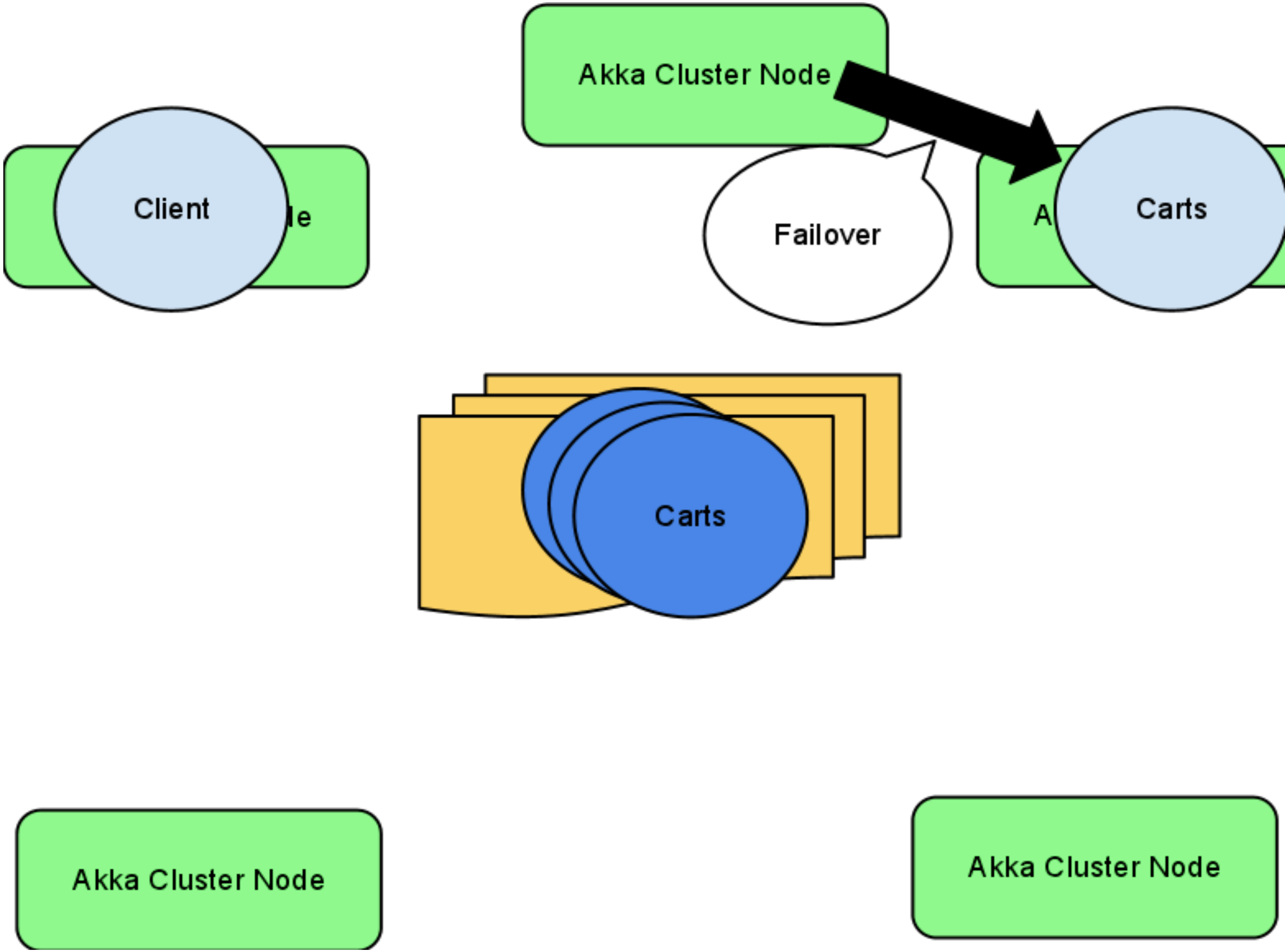

Carts
Client
Akka Cluster Node
Akka Cluster Node
Akka Cluster Node

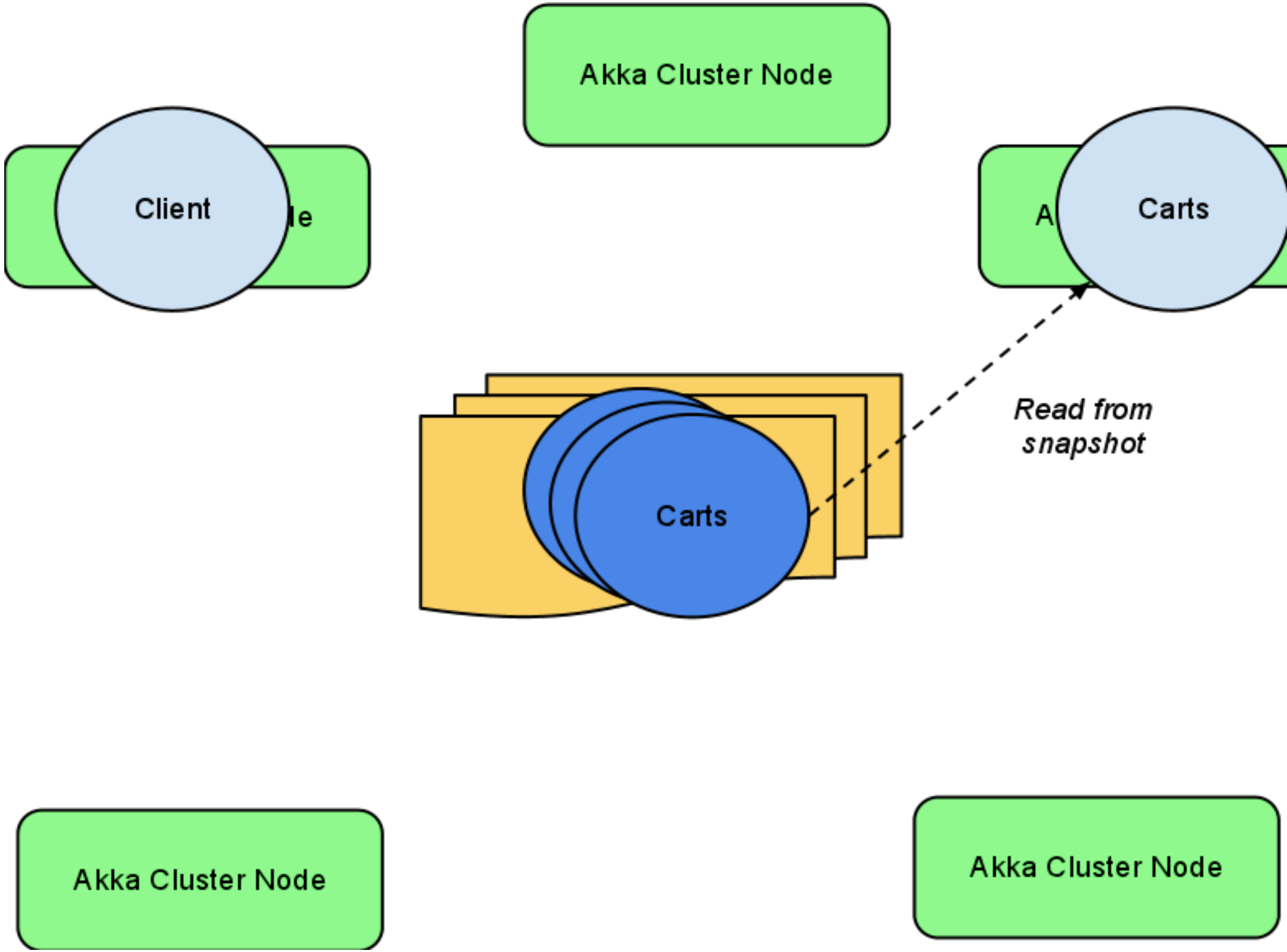

Carts
Client
Akka Cluster Node
Data Grid
Akka Cluster Node
Akka Cluster Node

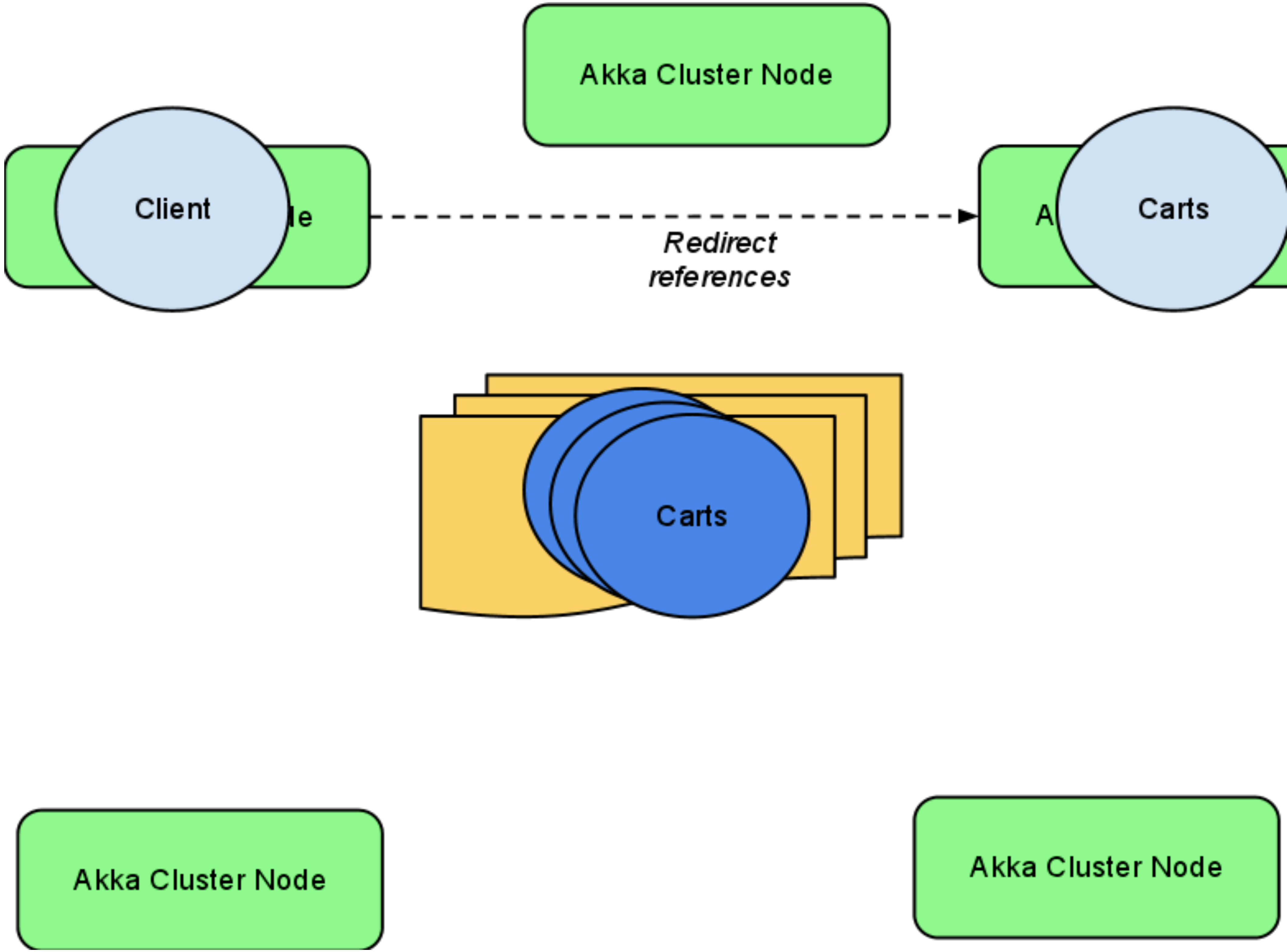

Carts
Client
Akka Cluster Node
Data Grid
Akka Cluster Node
Akka Cluster Node


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


Client
Akka Cluster Node
Carts
Akka Cluster Node
Akka Cluster Node


Akka Cluster Node
Client
Failover
Carts
Carts
Akka Cluster Node
Akka Cluster Node


Akka Cluster Node
Client
Carts
Read from snapshot
Carts
Akka Cluster Node
Akka Cluster Node


Akka Cluster Node
Client
Carts
Redirect references
Carts
Akka Cluster Node
Akka Cluster Node

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 {
  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
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
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)

AKKA 2.x

Hakka's Paradise

http://akka.io

Roadmap:

- 2.0
  - Location Transparency
  - API Cleanup
  - Configuration-based deployment
  - Failure Detection
  - Improved Supervisors + Lifecycle management
- 2.1
  - Clustered Elastic Akka
  - TO THE CLOUD!
- 2.2
  - Clustered management, replication, migration of stateful actors

EOF