Above the Clouds: Introducing Akka

Fredrik Ekholdt
Typesafe

Email: fredrik.ekholdt@typesafe.com
Twitter: @ekhfre

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

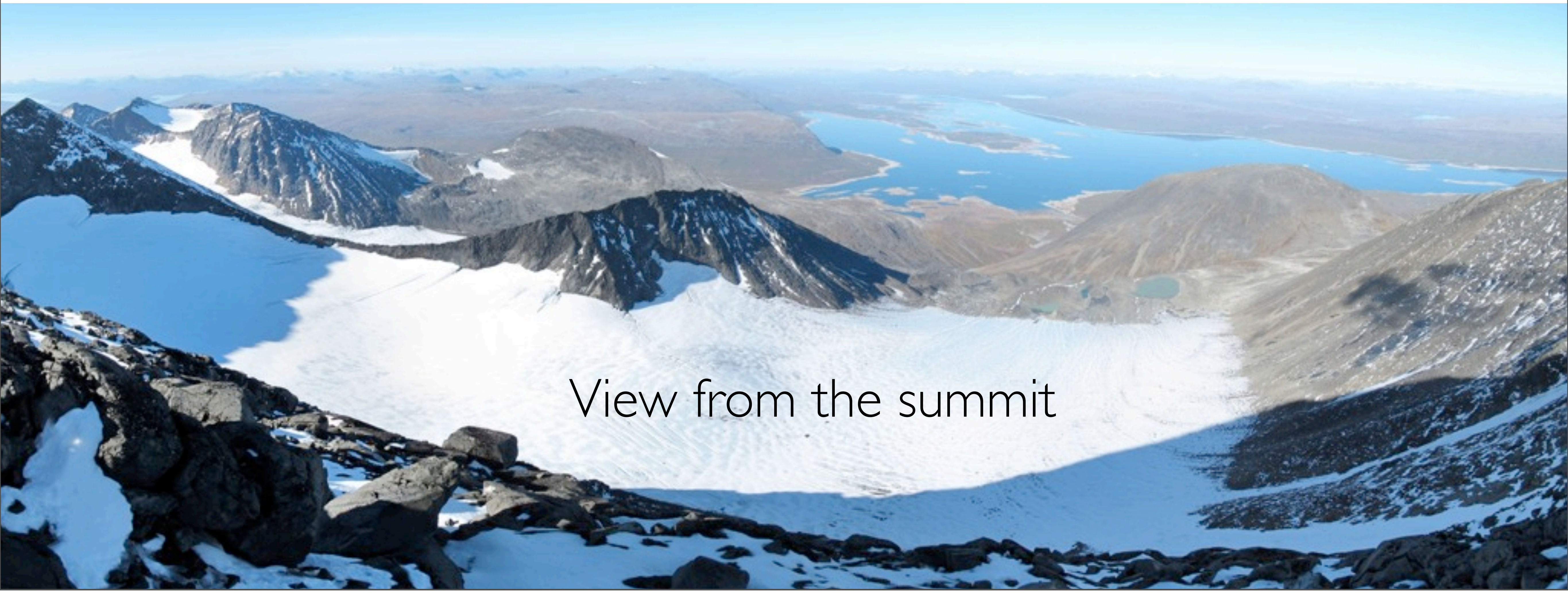
Introducing akka

Akka (Áhkká):

The name comes from the goddess in the Sami mythology that represented all the wisdom and beauty in the world.

It is also the name of a beautiful mountain in Laponia in the north part of Sweden

Introducing akka

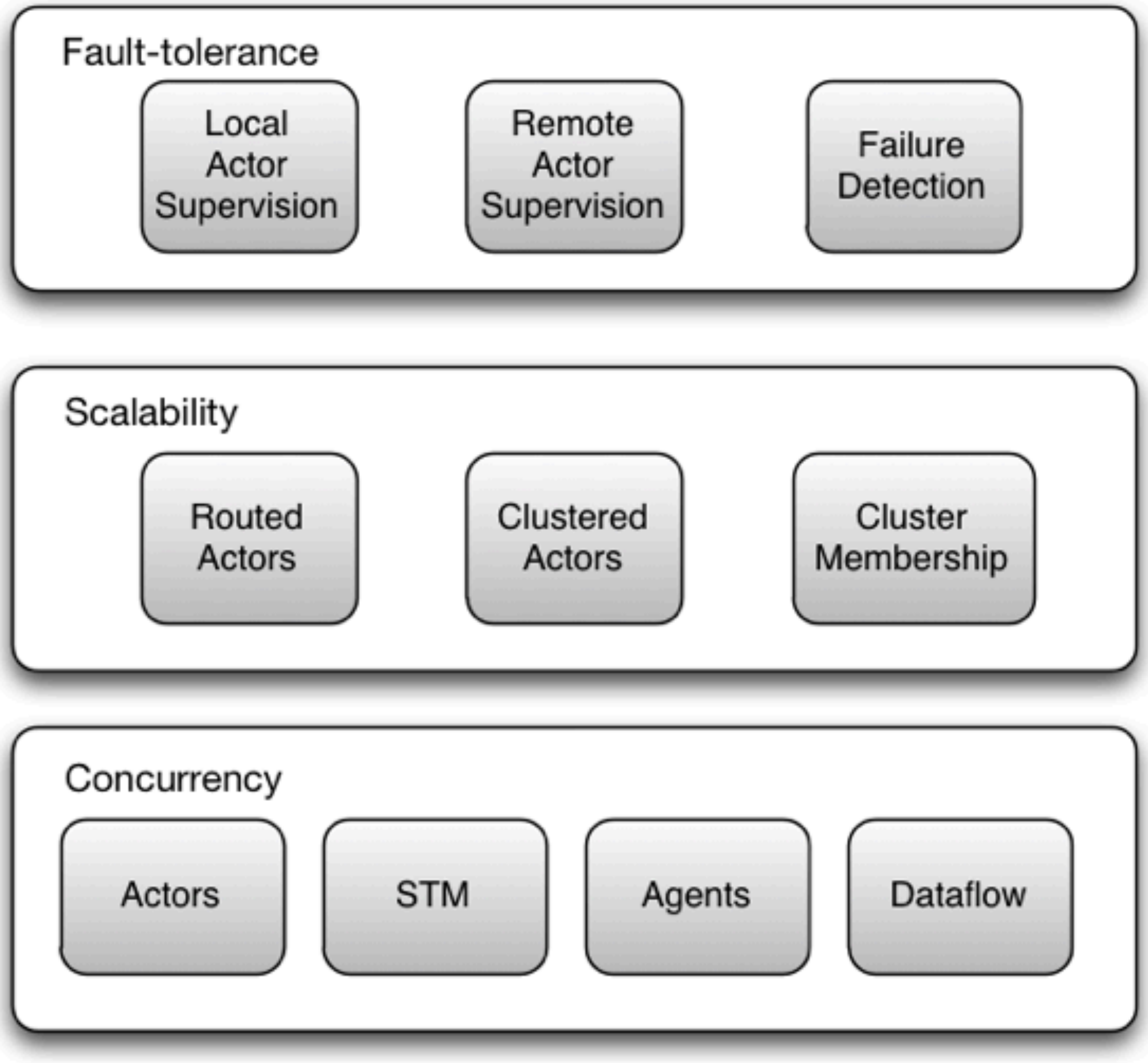


Vision

Simpler

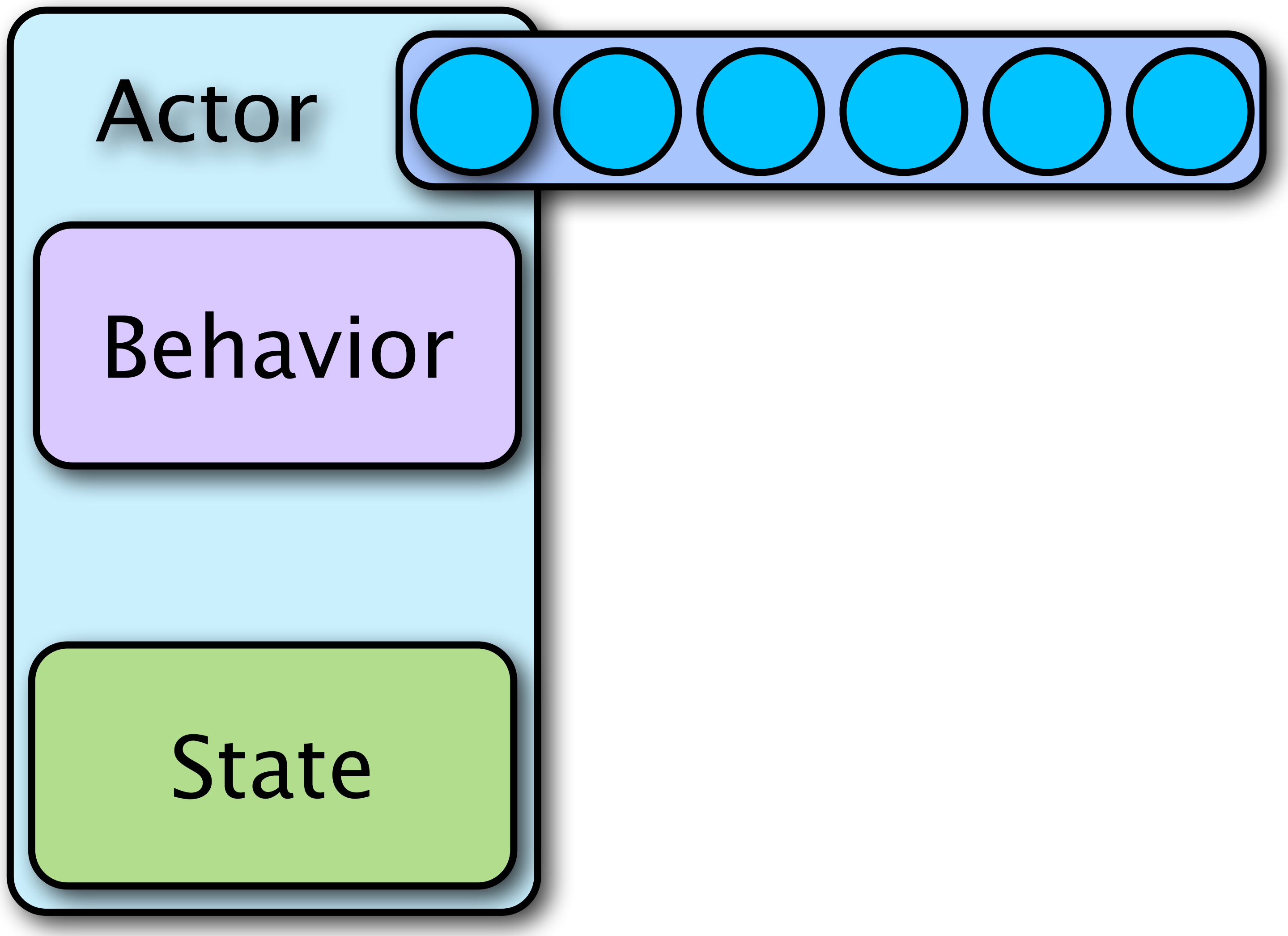
- Concurrency
- Scalability
- Fault-tolerance

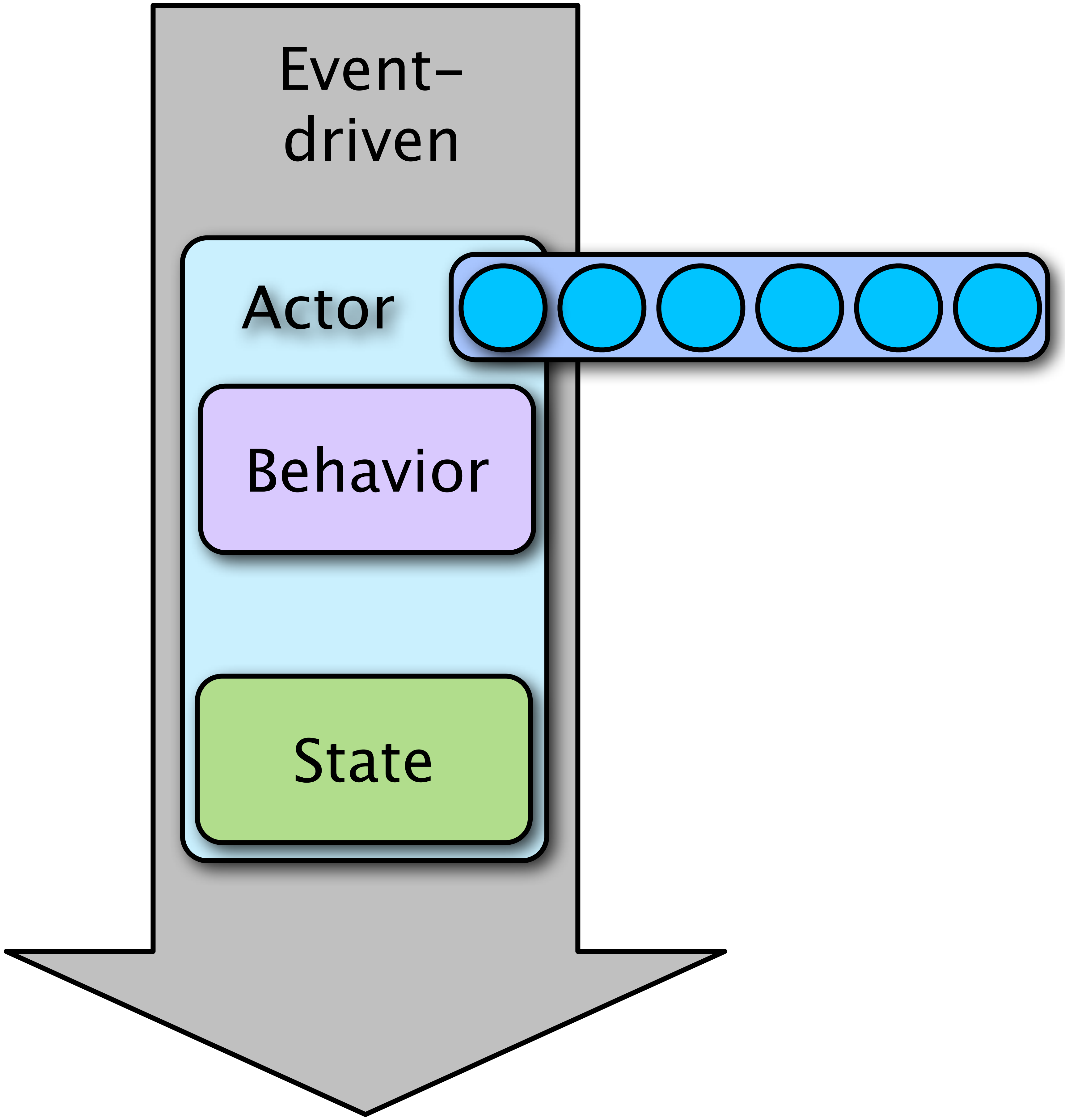
THE TOOLBOX...

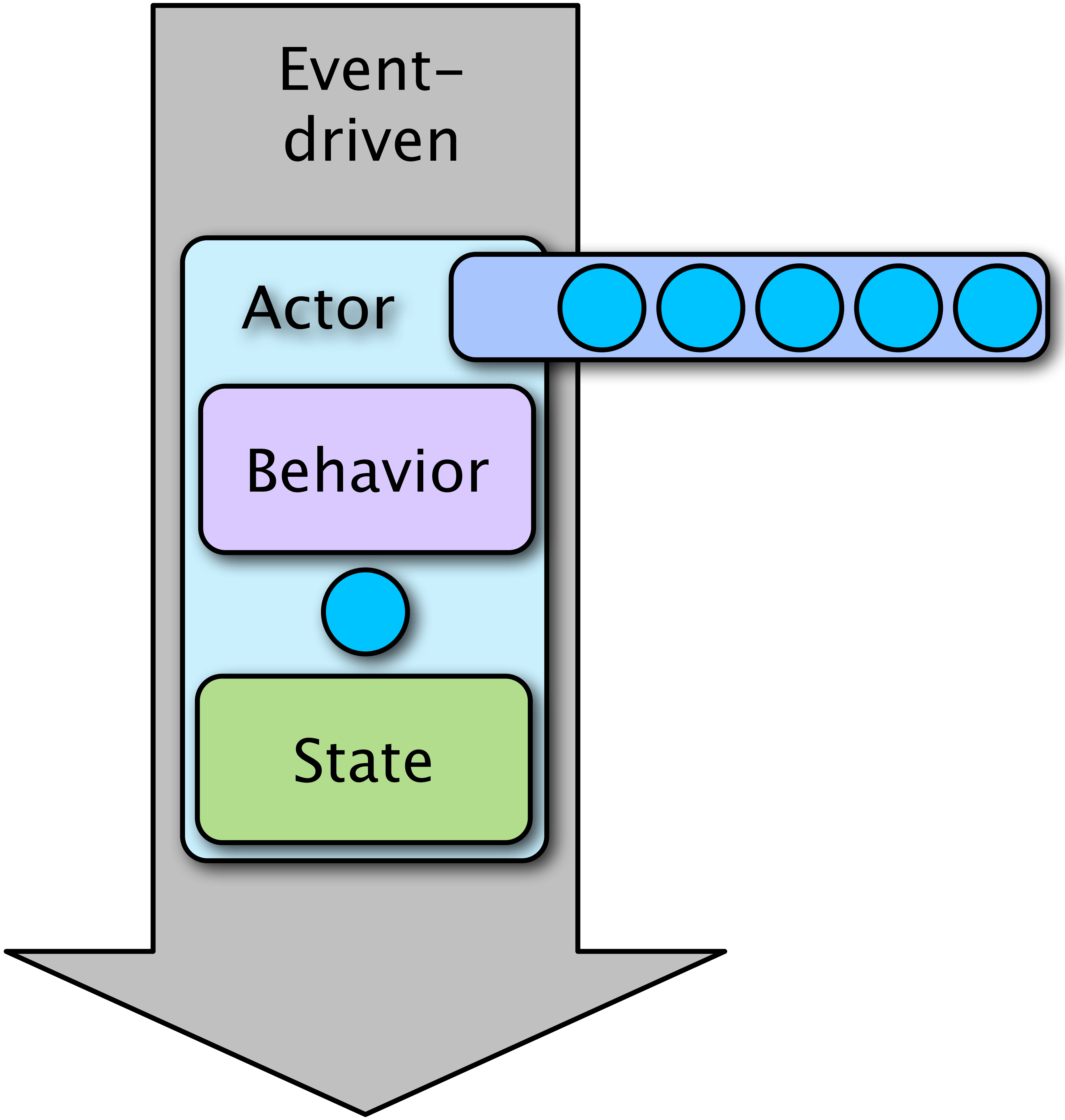


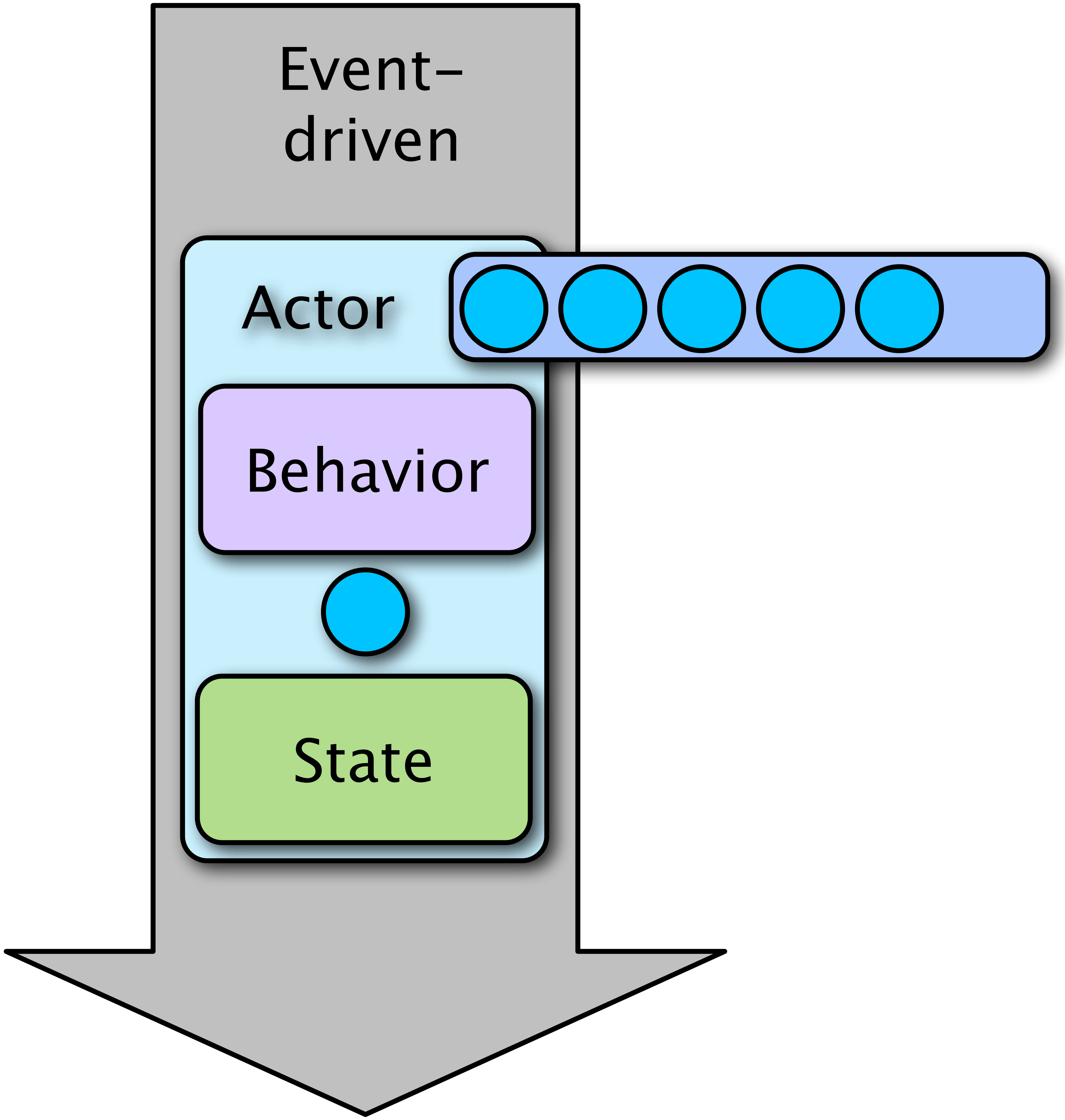
CORE SERVICES

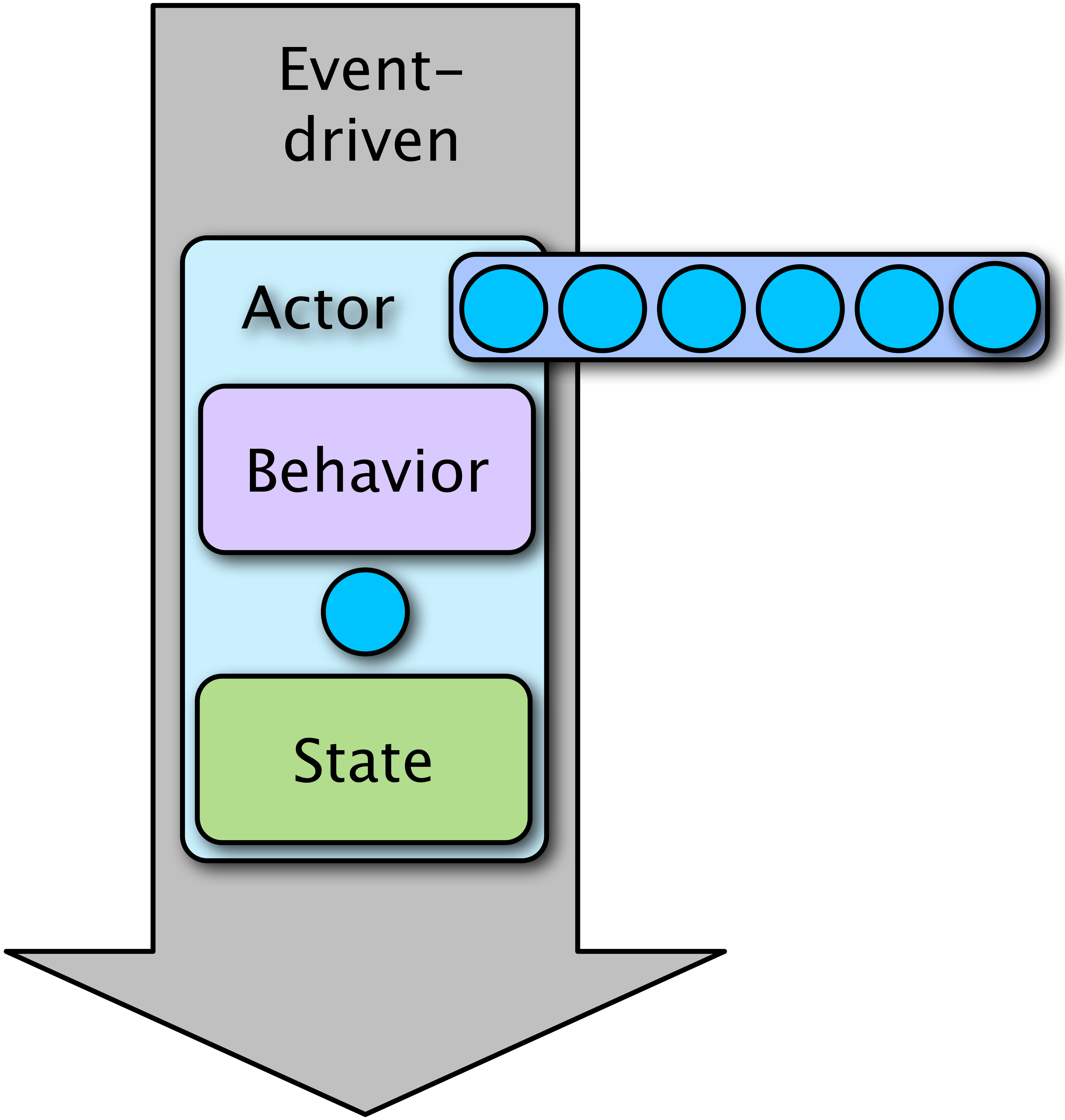
The Actor...

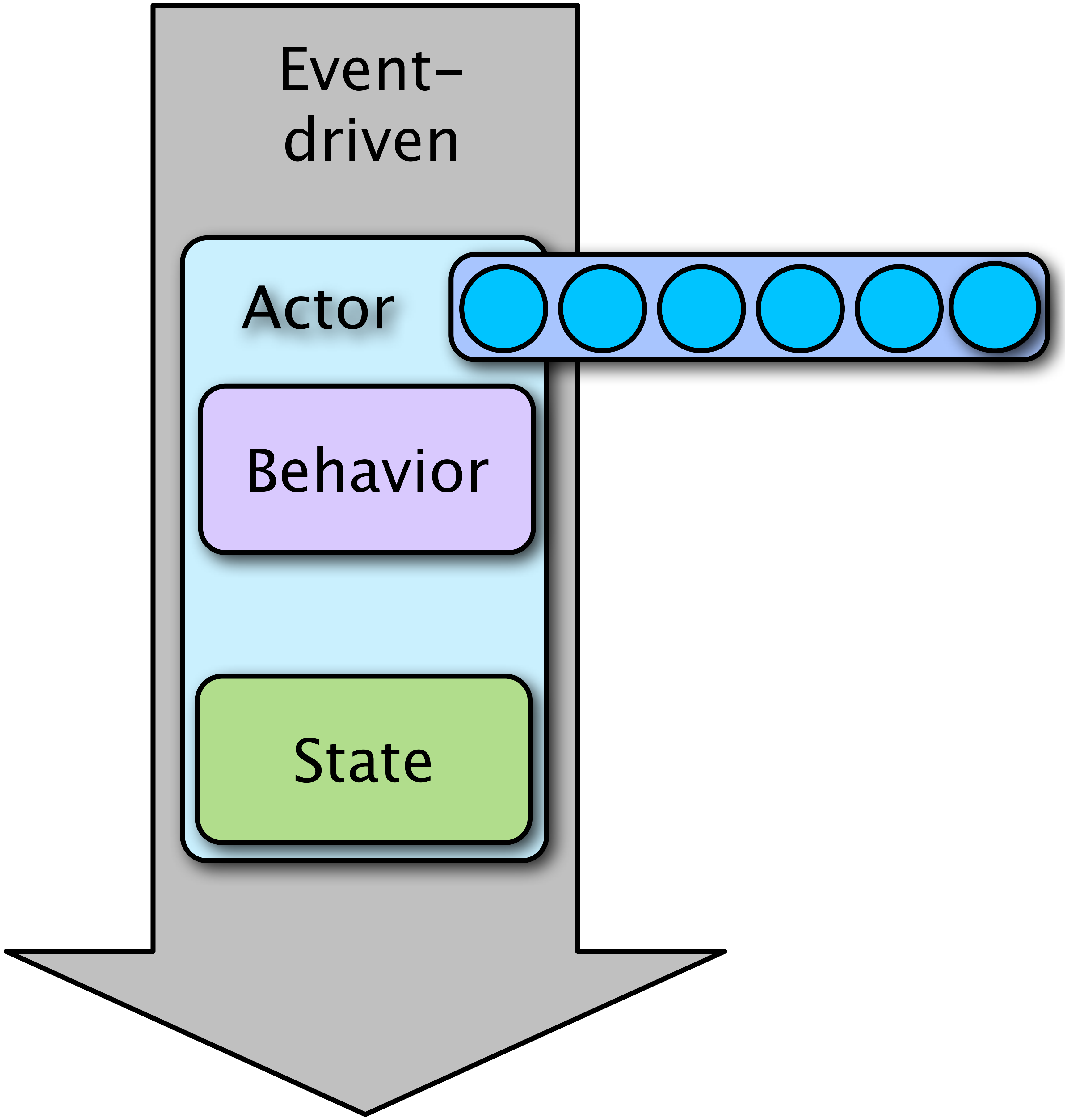

Actor
Behavior
State

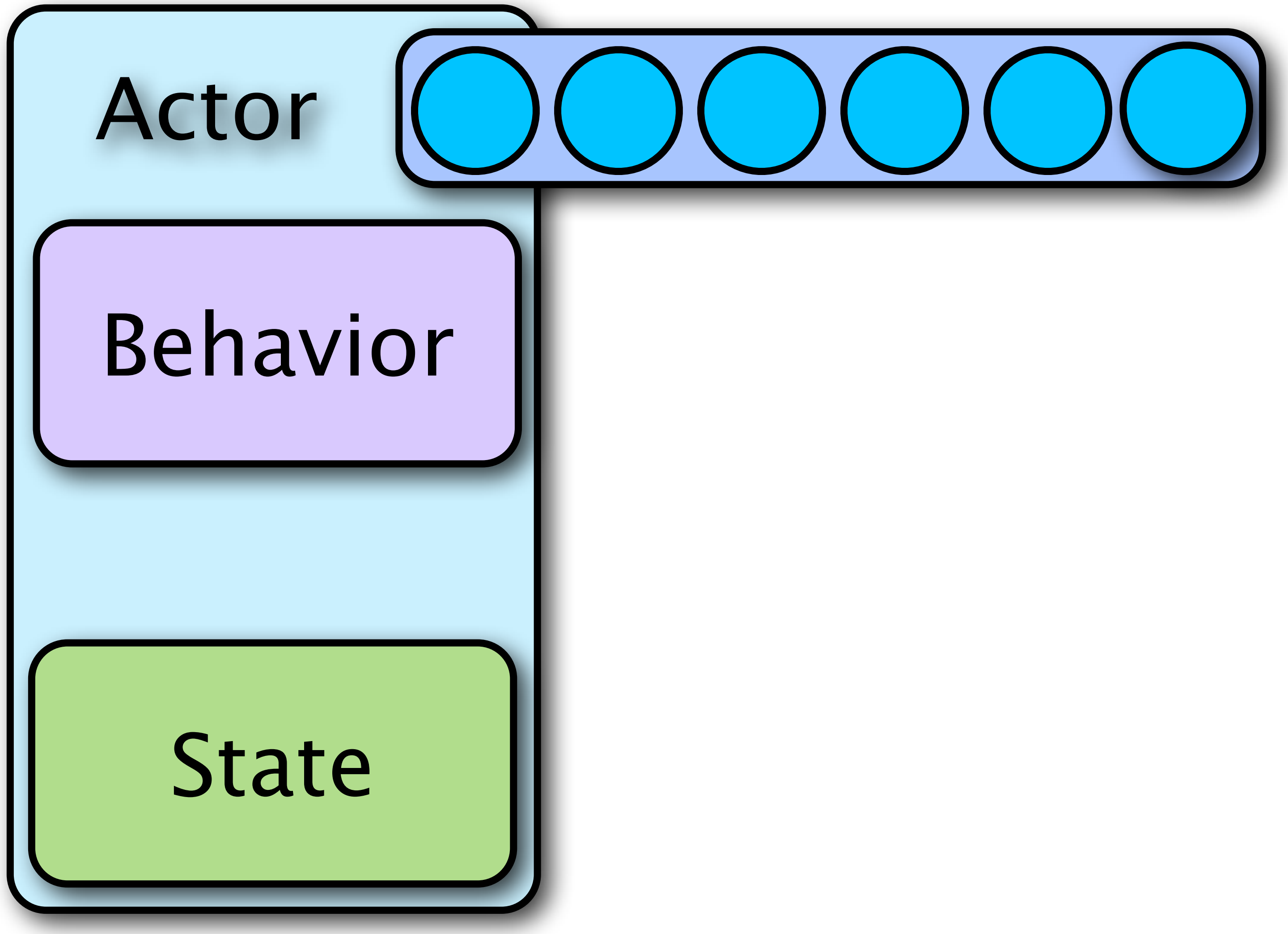

Event-
driven
Actor
Behavior
State

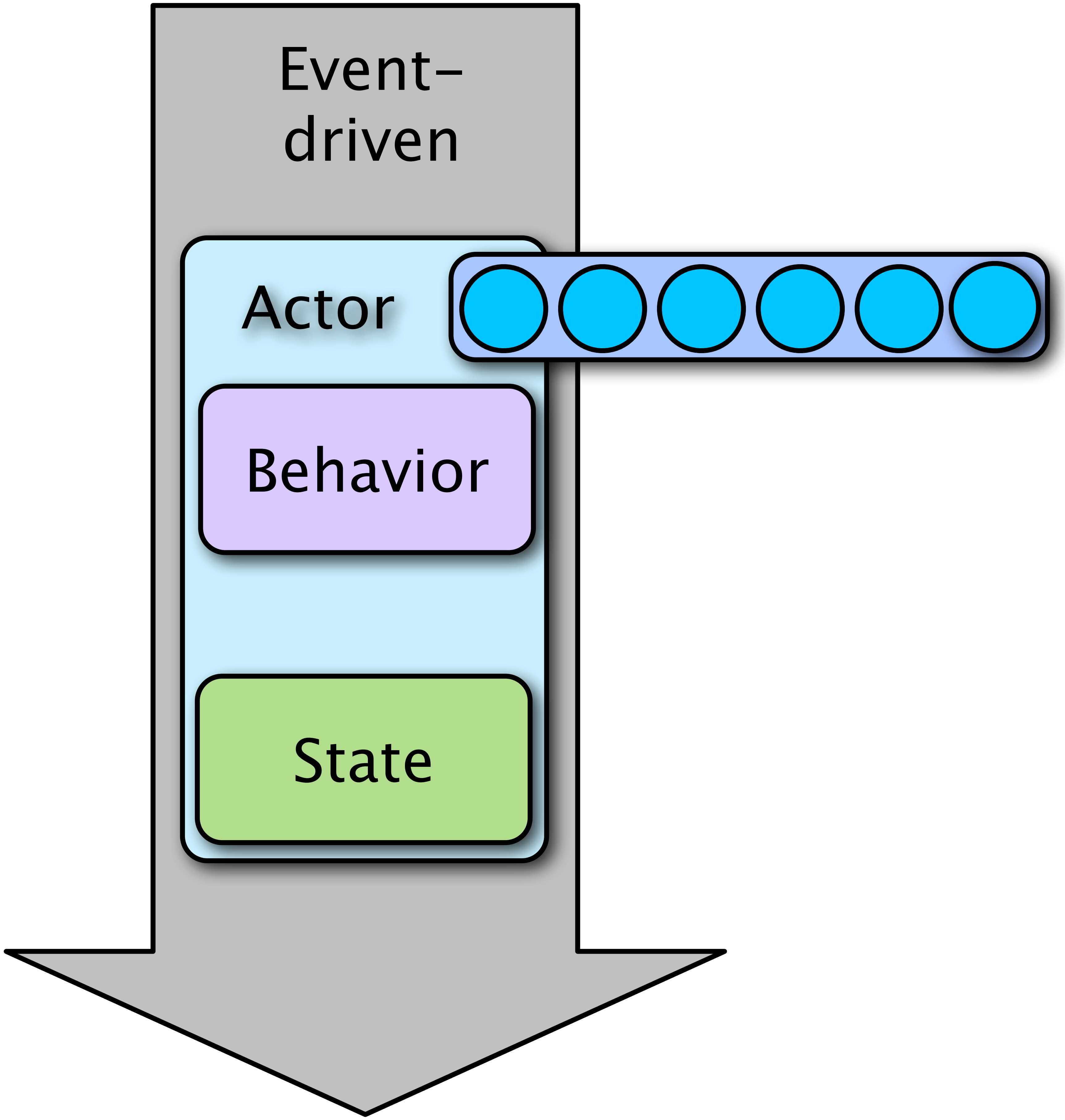

Event-
driven
Actor
Behavior
State


Event-
driven
Actor
Behavior
State


Event-
driven
Actor
Behavior
State


Event-
driven
Actor
Behavior
State


Actor
Behavior
State


Event-
driven
Actor
Behavior
State

Akka Actors

Let the hakking begin!

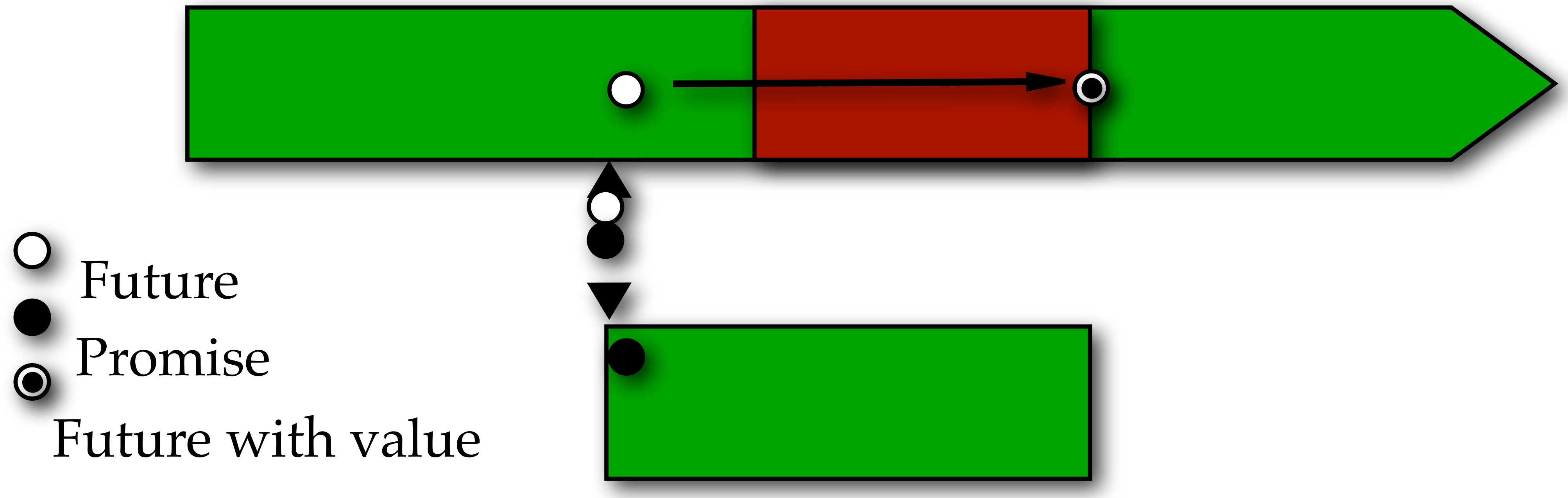
Take aways

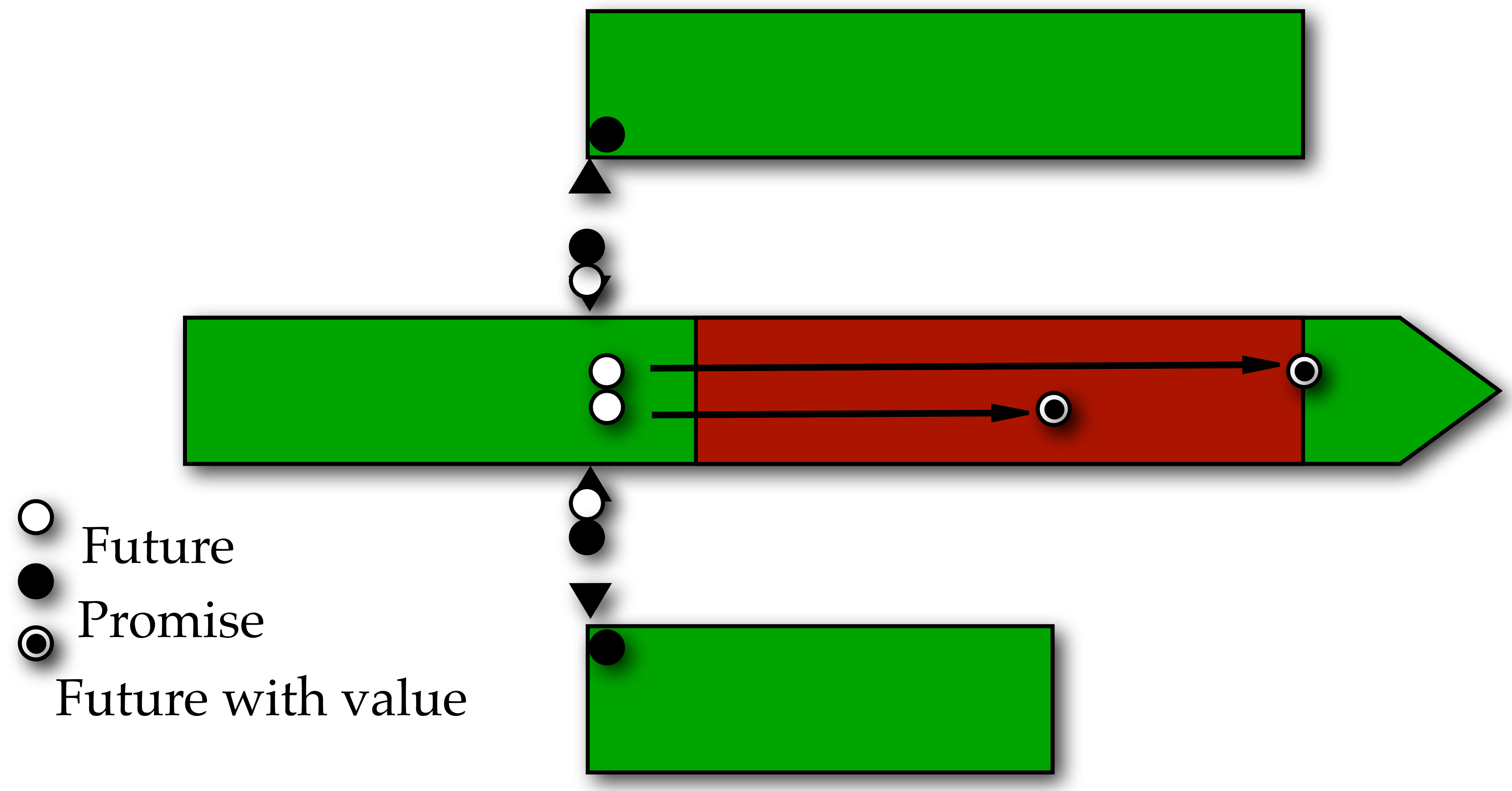
- Create an Actor: class AnActor extends Actor
- Start an Actor: actorOf[AnActor].start()
- Safe
  - Encapsulated state
  - One message at a time
- Asynchronous:
  - bang (!)
- Reply:
  - self.reply (m)
  - actor ? “message”

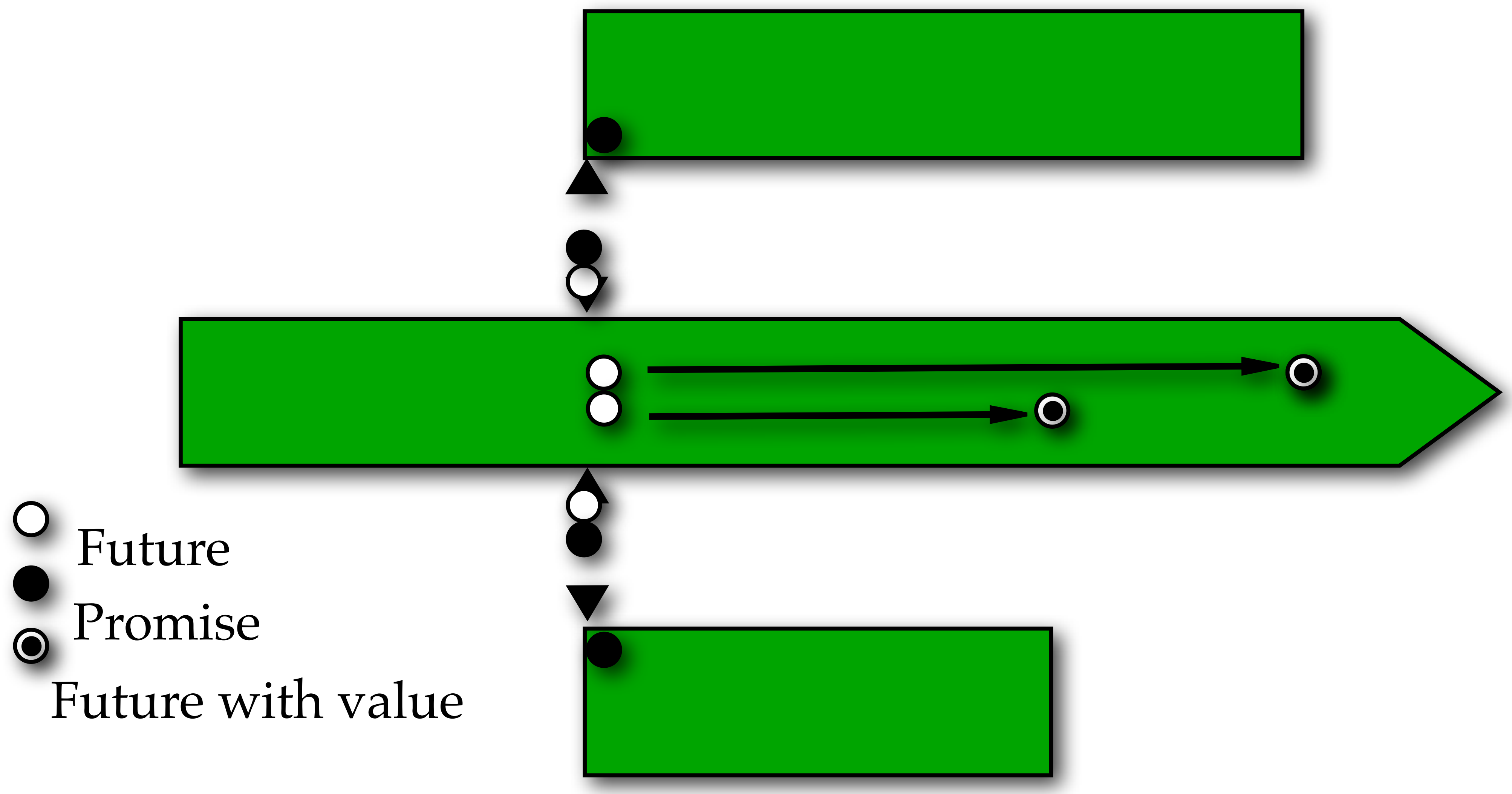
using

Akka Futures

without blocking


○ Future
● Promise
◉ Future with value


Future
Promise
Future with value


Future
Promise
Future with value

Take aways

- Use Futures when you need something in the Future without blocking
- Use onComplete
- Compose Futures
  - for - comprehensions
  - Check out the Futures utilities (firstCompletedOf and much, much more)

Remote Actors

Another snippet of code...

New Remote Actors

Name

```
appContext.actorOf[MyActor](“my-service”)
```

Bind the actor to a name

Deployment

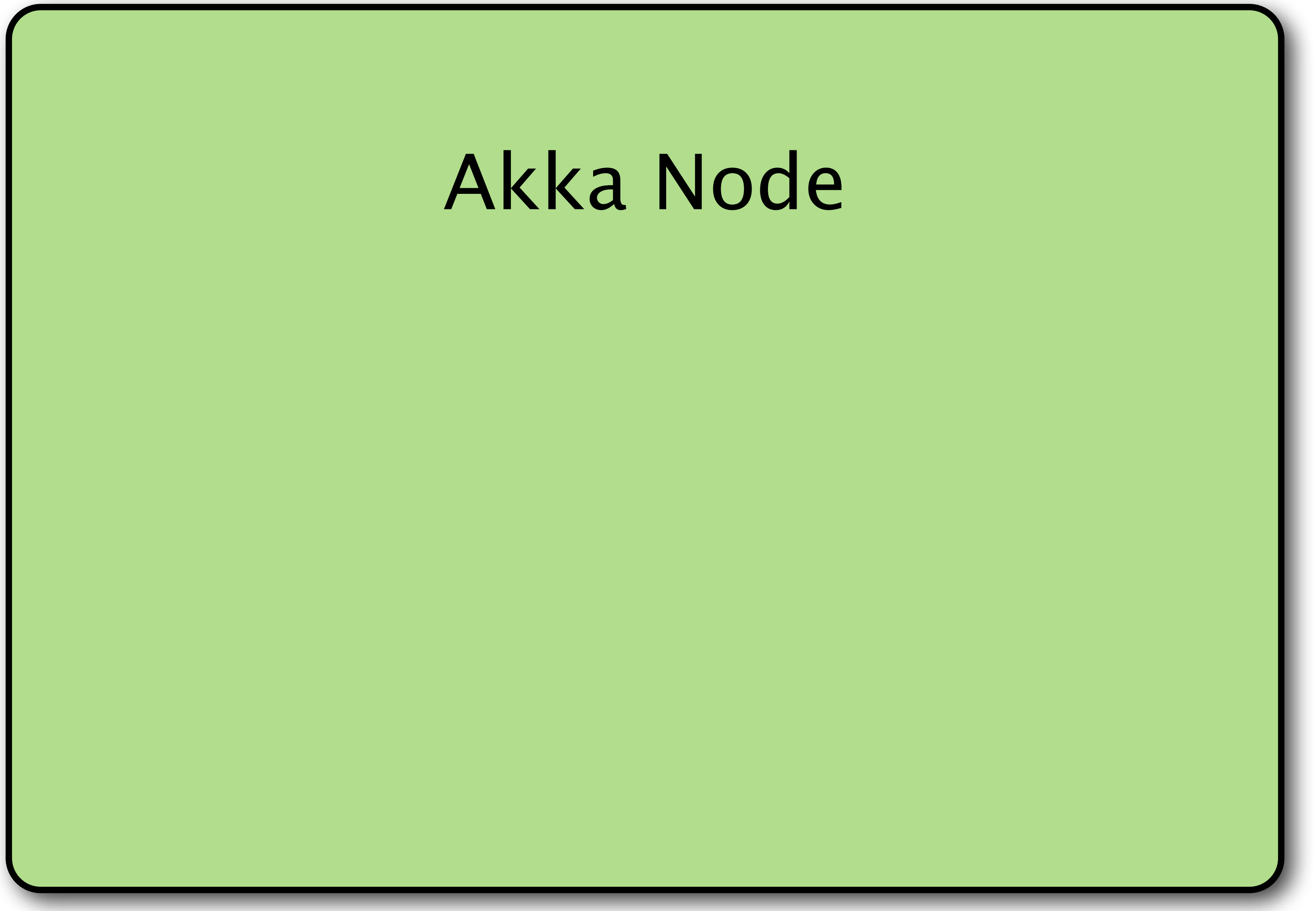
- Actor name is virtual and decoupled from how it is deployed
- If no deployment configuration exists then actor is deployed as local
- The same system can be configured as distributed without code change (even change at runtime)
- Write as local but deploy as distributed in the cloud without code change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

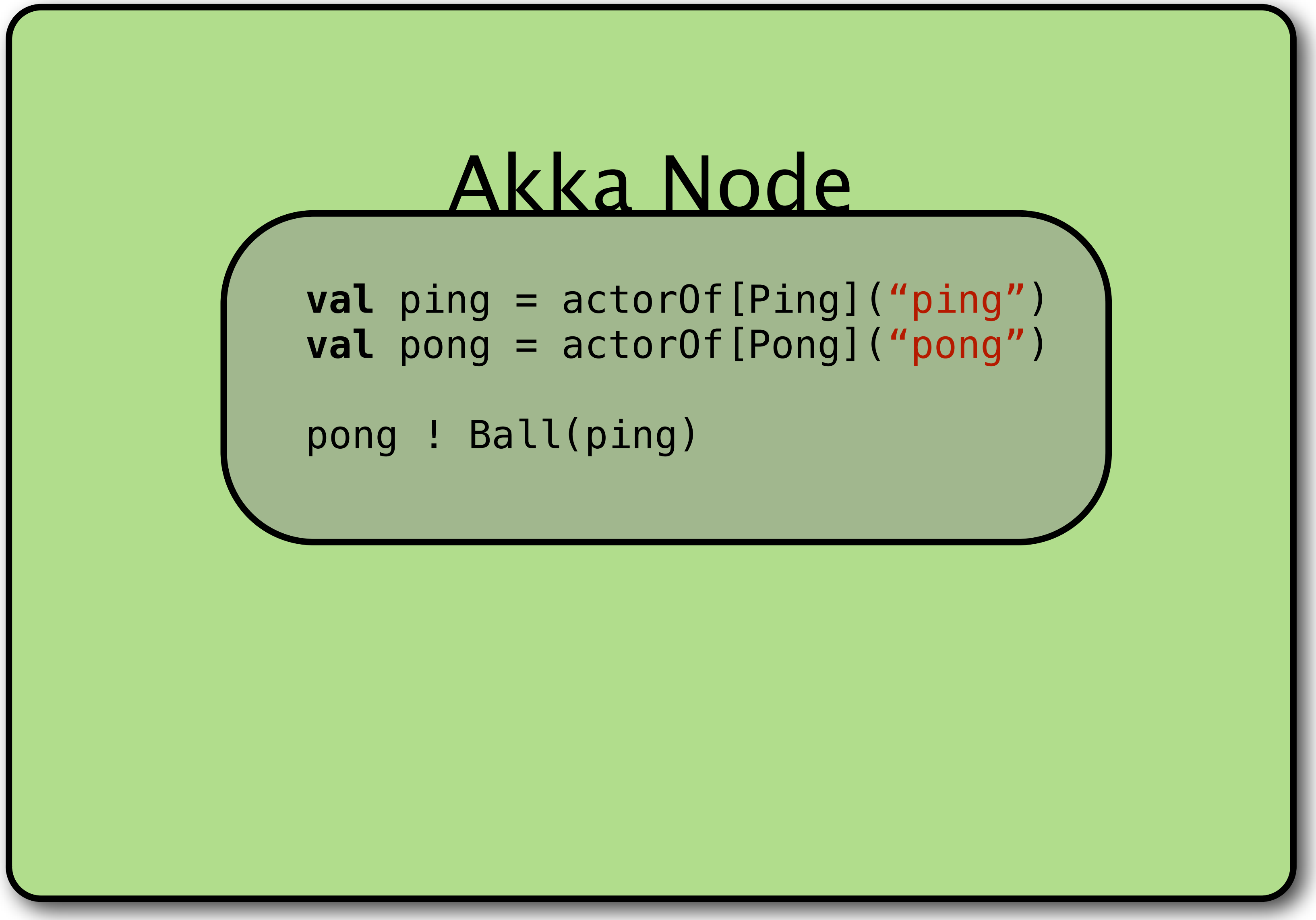
Deployment configuration

```
akka {
  actor {
    deployment {
      /path/to/my-service {
        router = "round-robin"
        nr-of-instances = 3
        remote {
          nodes = ["wallace:2552", "gromit:2552"]
        }
      }
    }
  }
}
```

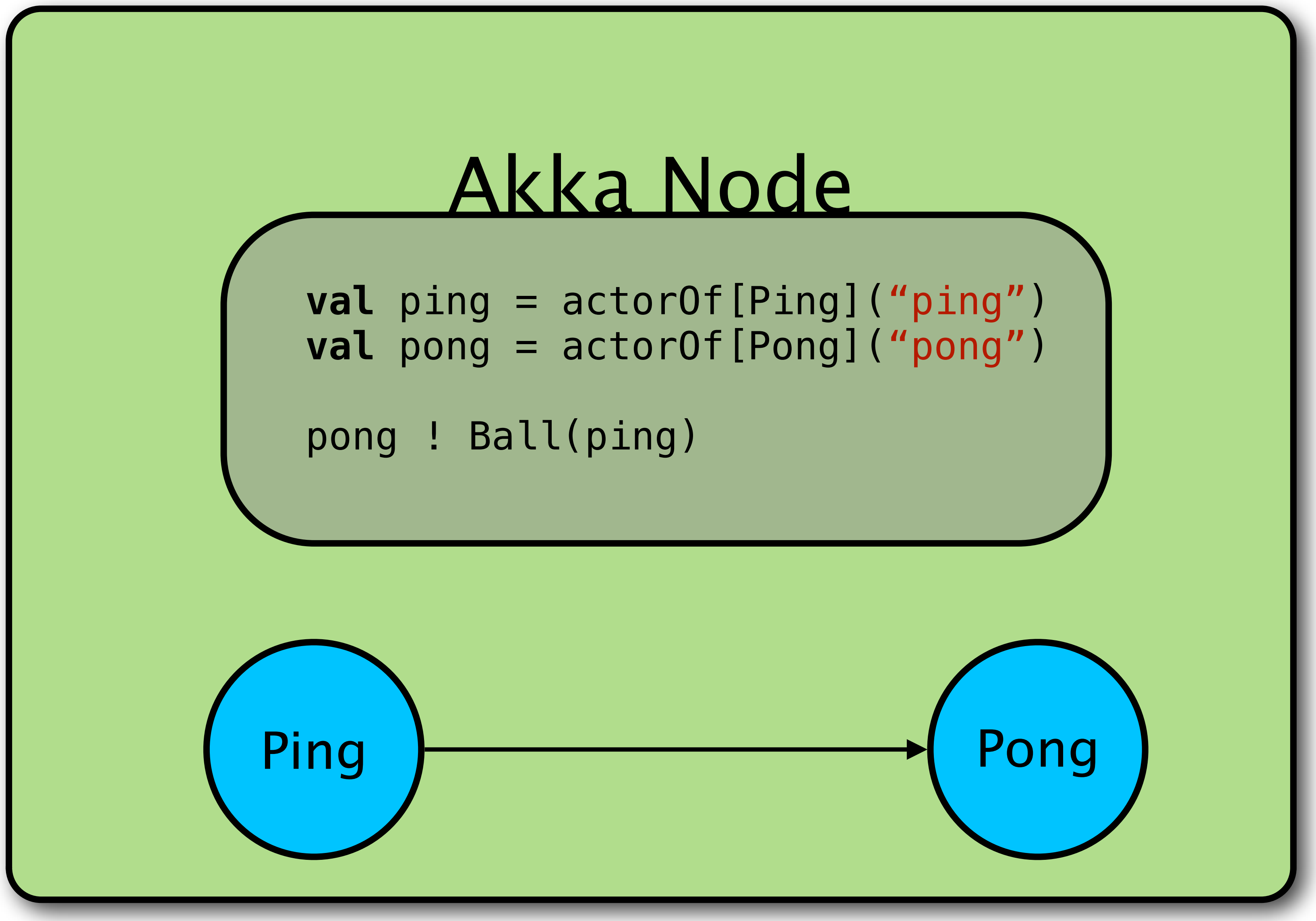
The runtime provides

- Decentralized P2P gossip-based cluster membership (dynamo-style w/ vector clocks, hand-off on fail-over etc.)
- Automatic adaptive cluster rebalancing
- Automatic cluster-wide deployment
- Highly available configuration service
- Automatic replication with automatic fail-over upon node crash
- Transparent and user-configurable load-balancing
- ...and much more

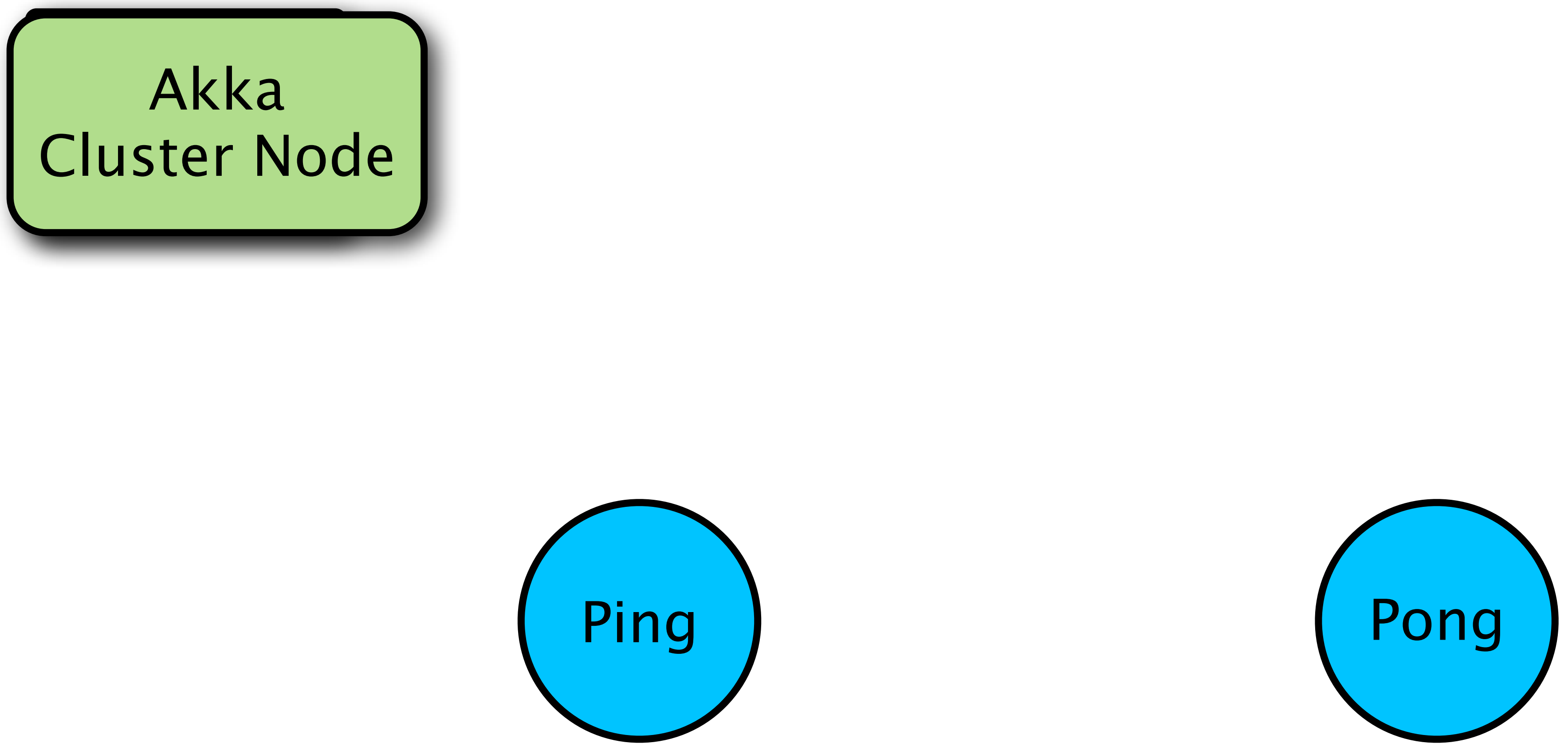

Akka Node

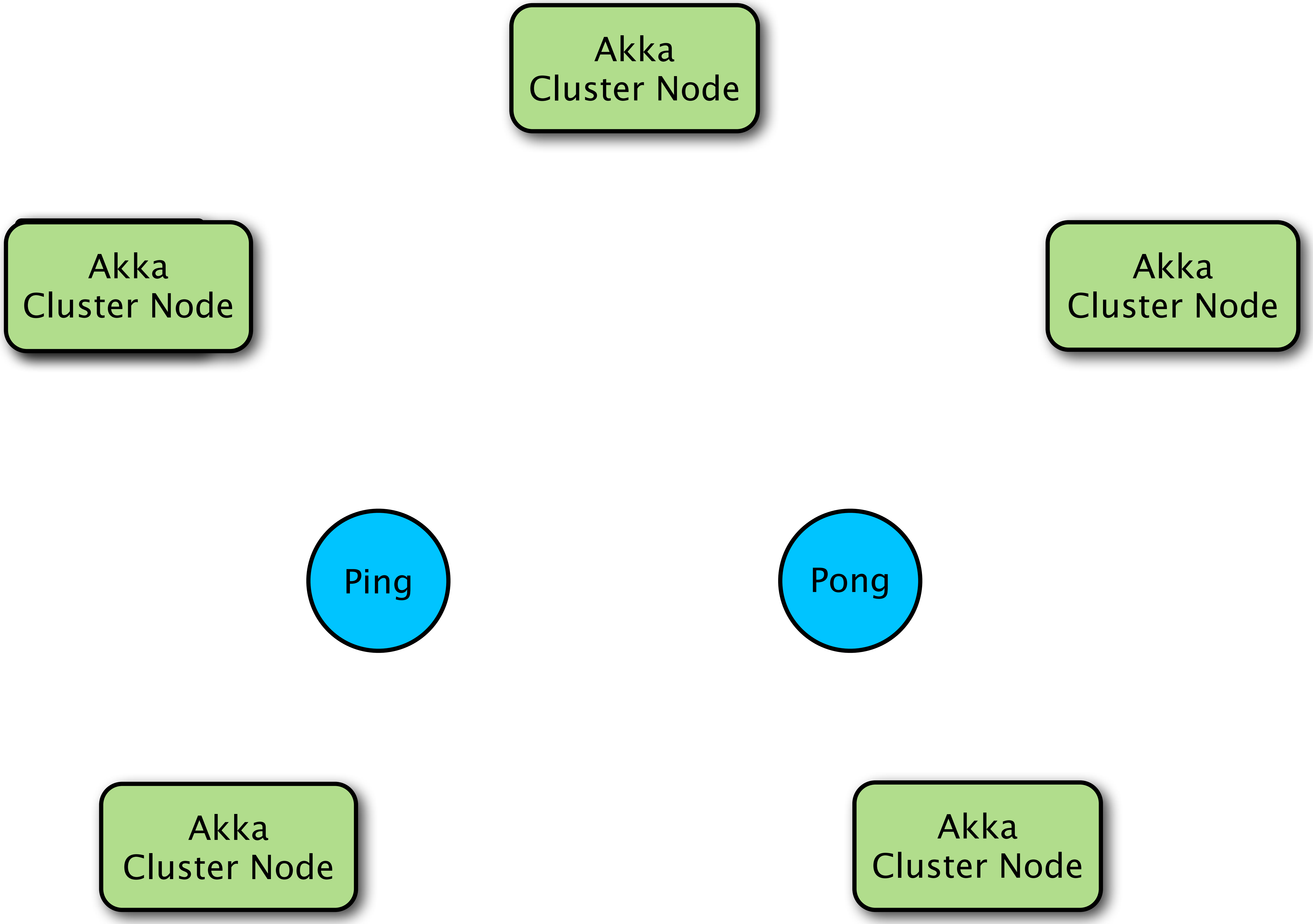


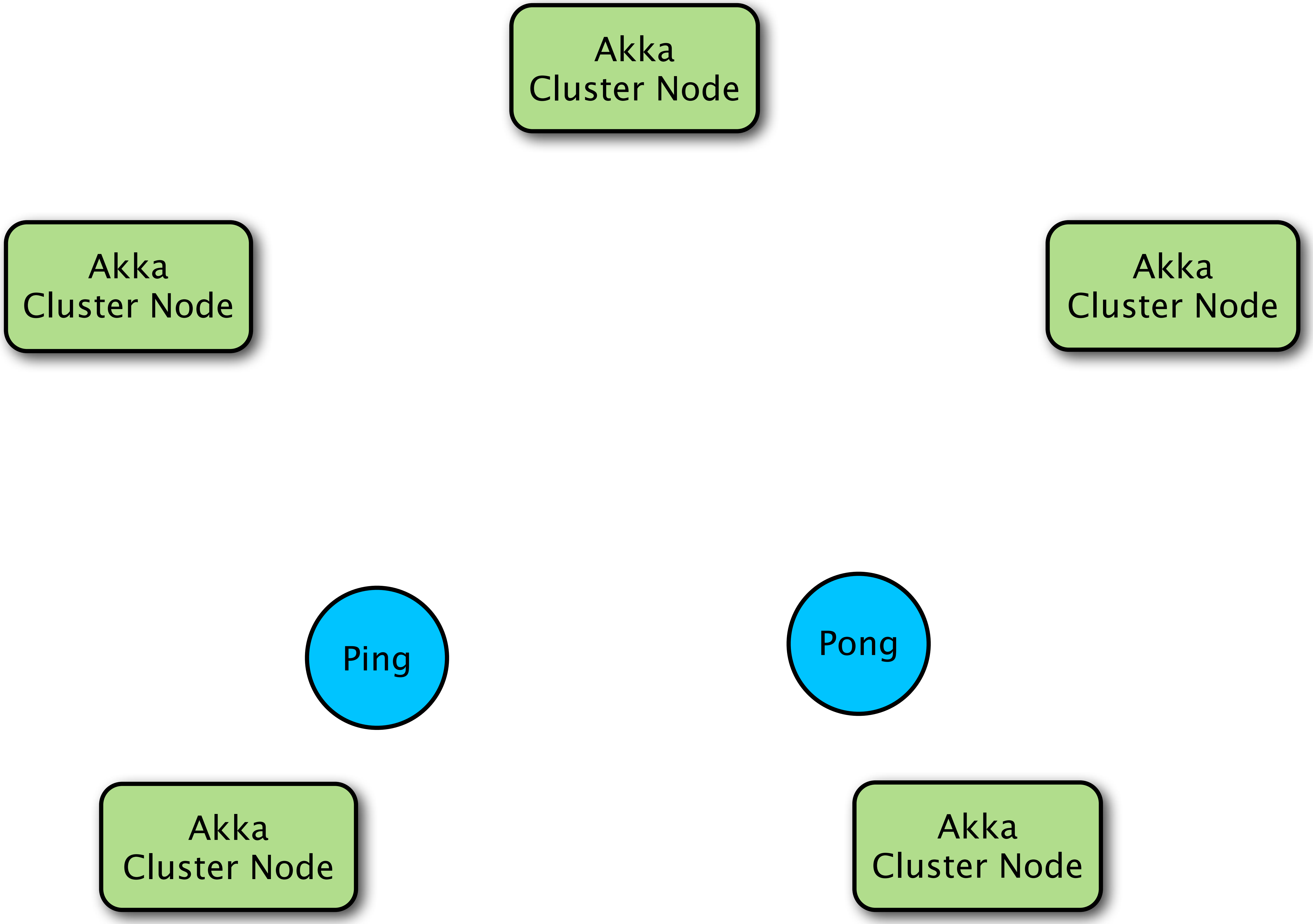
```
val ping = actorOf[Ping](“ping”)
val pong = actorOf[Pong](“pong”)

pong ! Ball(ping)
```

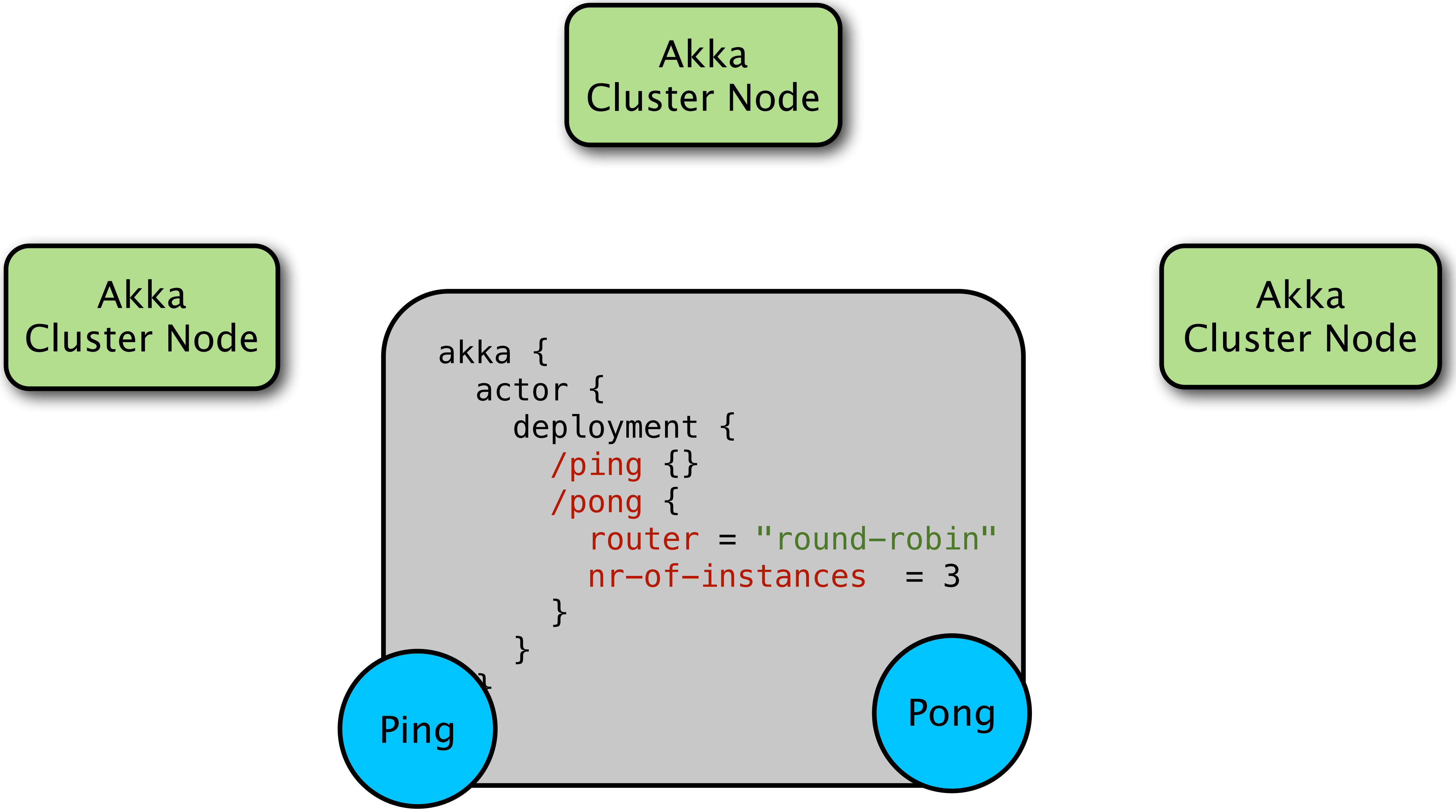


```
val ping = actorOf[Ping]("ping")
val pong = actorOf[Pong]("pong")

pong ! Ball(ping)
```

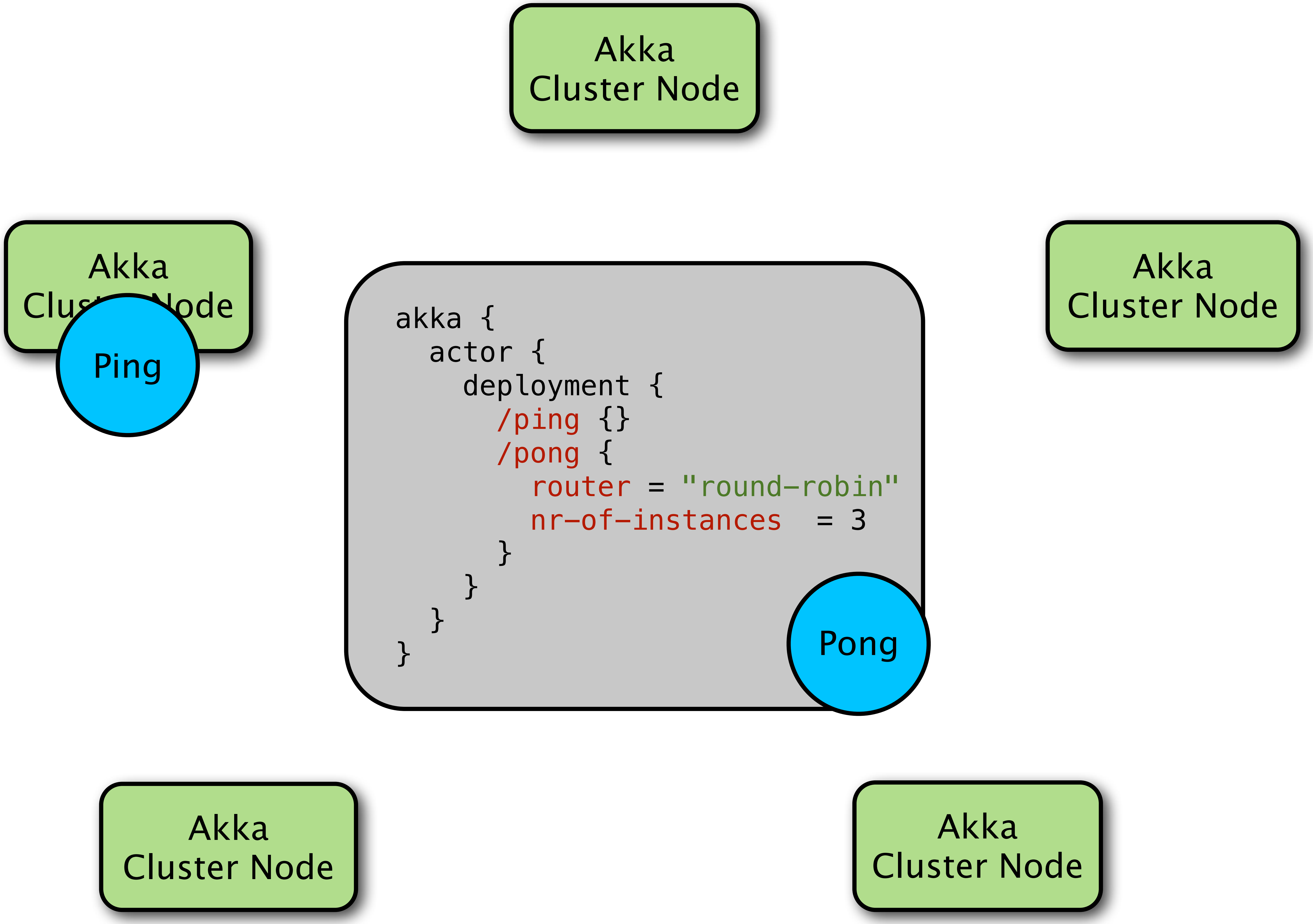

Akka
Cluster Node
Ping
Pong


Akka Cluster Node
Akka Cluster Node
Akka Cluster Node
Ping
Pong
Akka Cluster Node
Akka Cluster Node

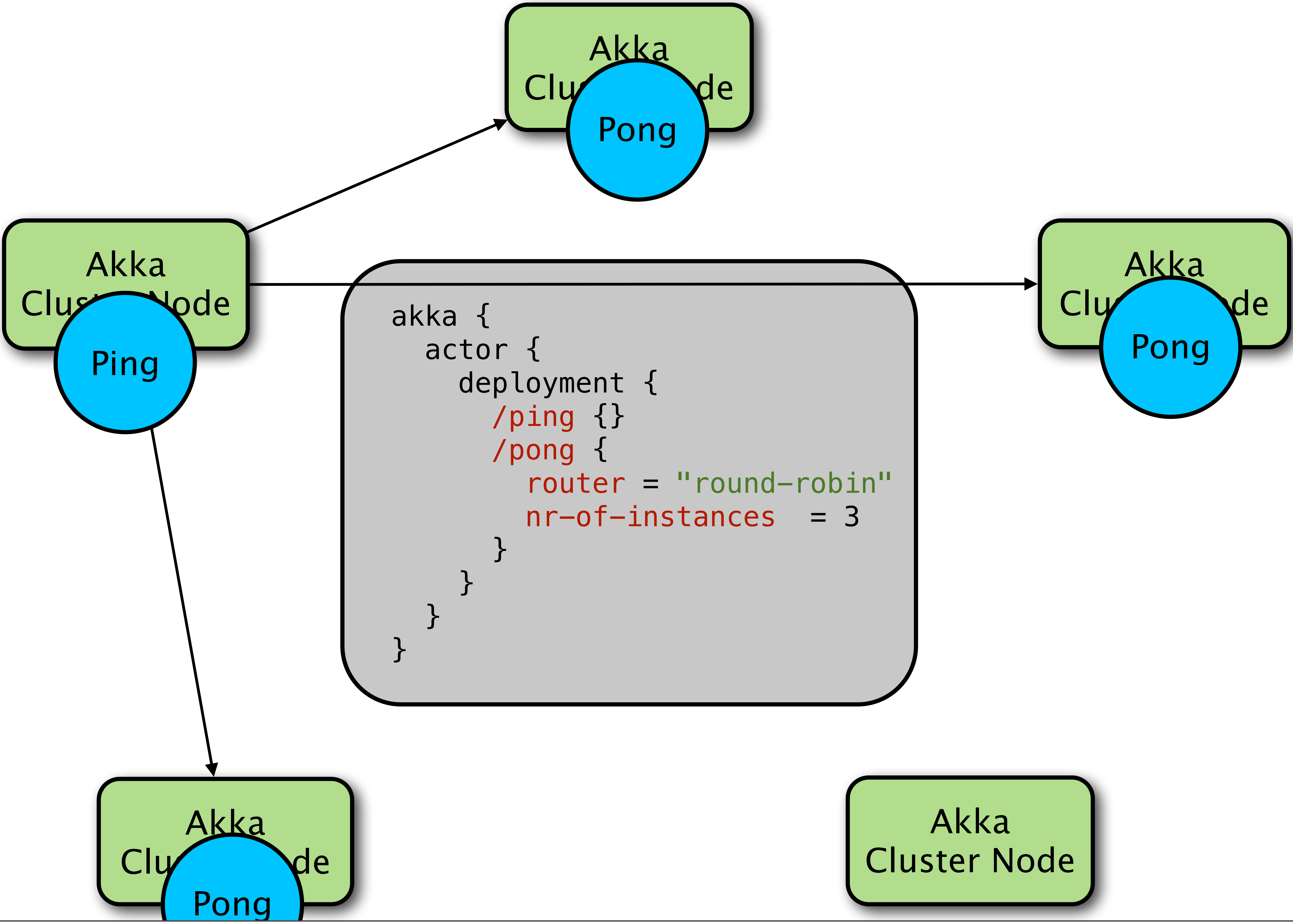

Akka Cluster Node
Akka Cluster Node
Akka Cluster Node
Ping
Pong
Akka Cluster Node
Akka Cluster Node



```
akka {
  actor {
    deployment {
      /ping {}
      /pong {
        router = "round-robin"
        nr-of-instances  = 3
      }
    }
  }
```



```
akka {
  actor {
    deployment {
      /ping {}
      /pong {
        router = "round-robin"
        nr-of-instances  = 3
      }
    }
  }
}
```

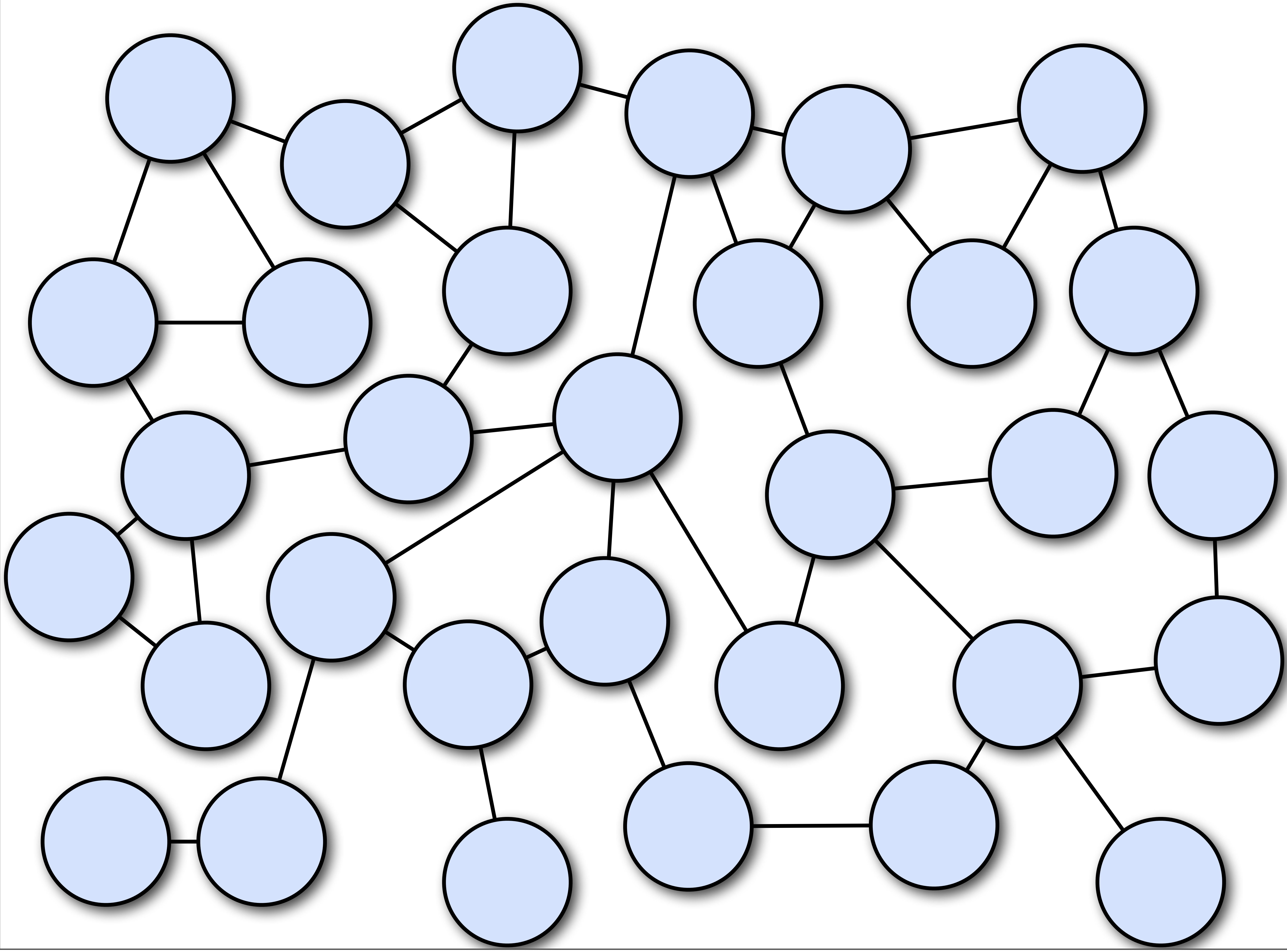


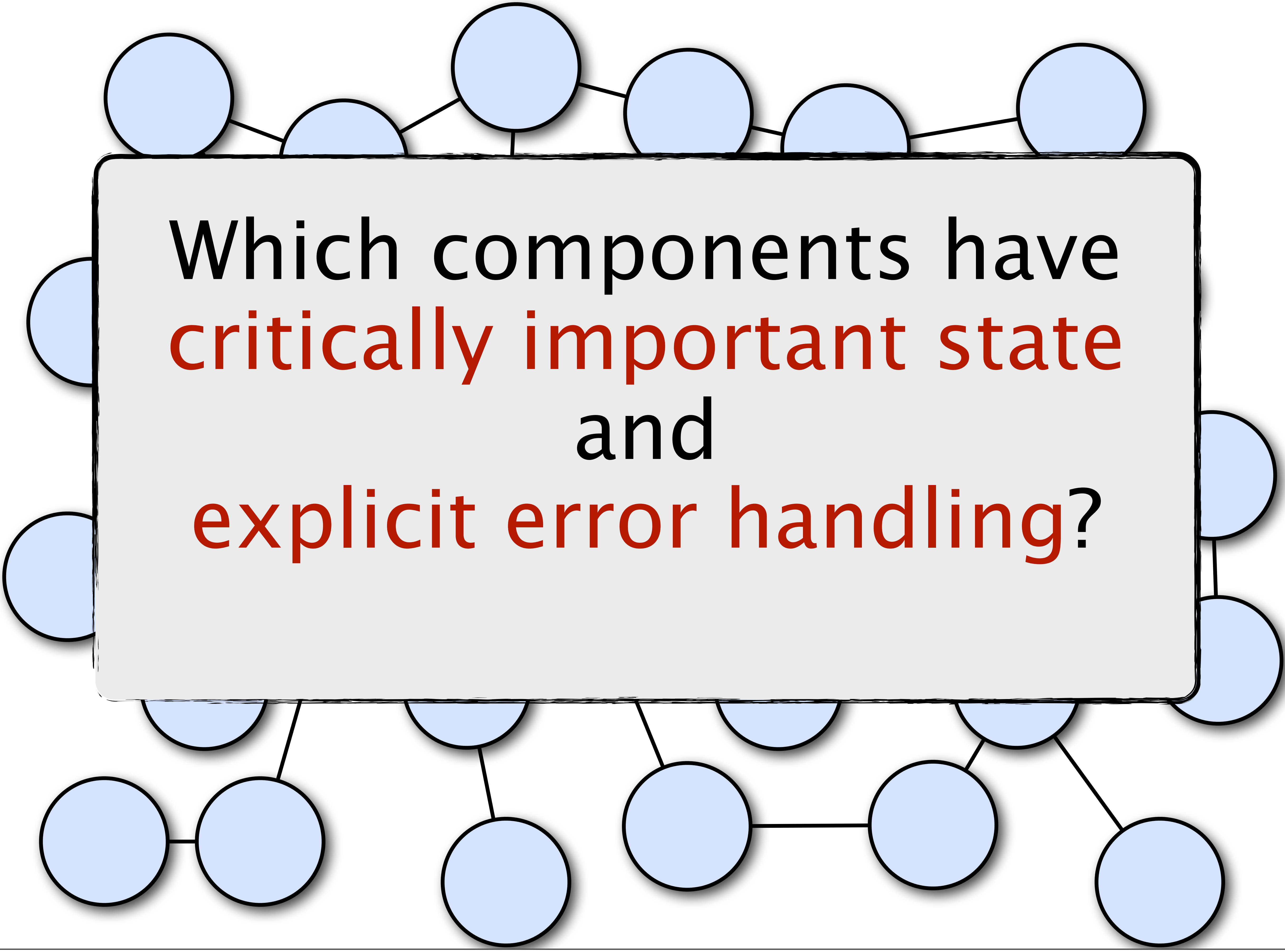
```
akka {
  actor {
    deployment {
      /ping {}
      /pong {
        router = "round-robin"
        nr-of-instances  = 3
      }
    }
  }
}
```

Summing up

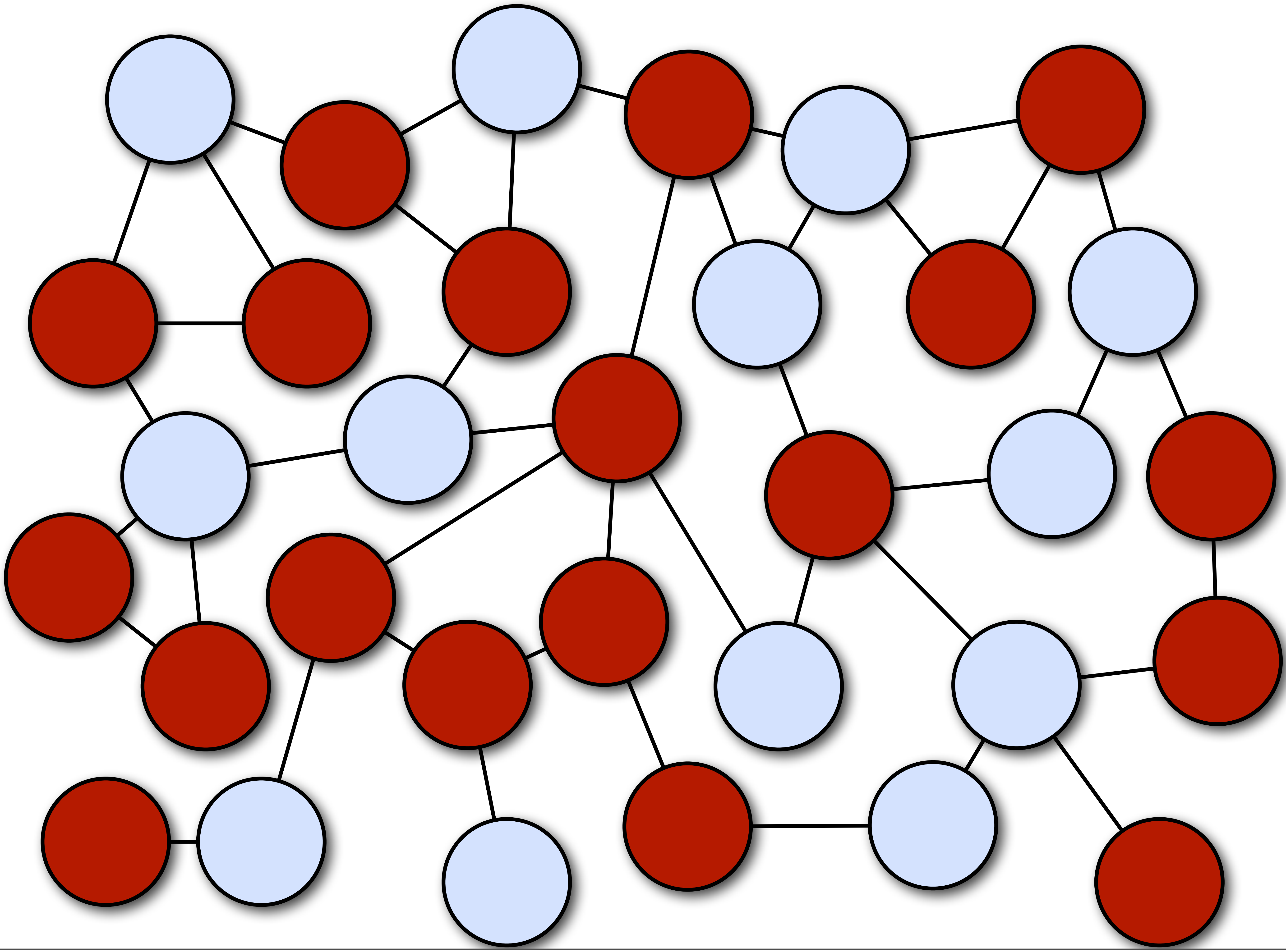
Let it crash
fault-tolerance

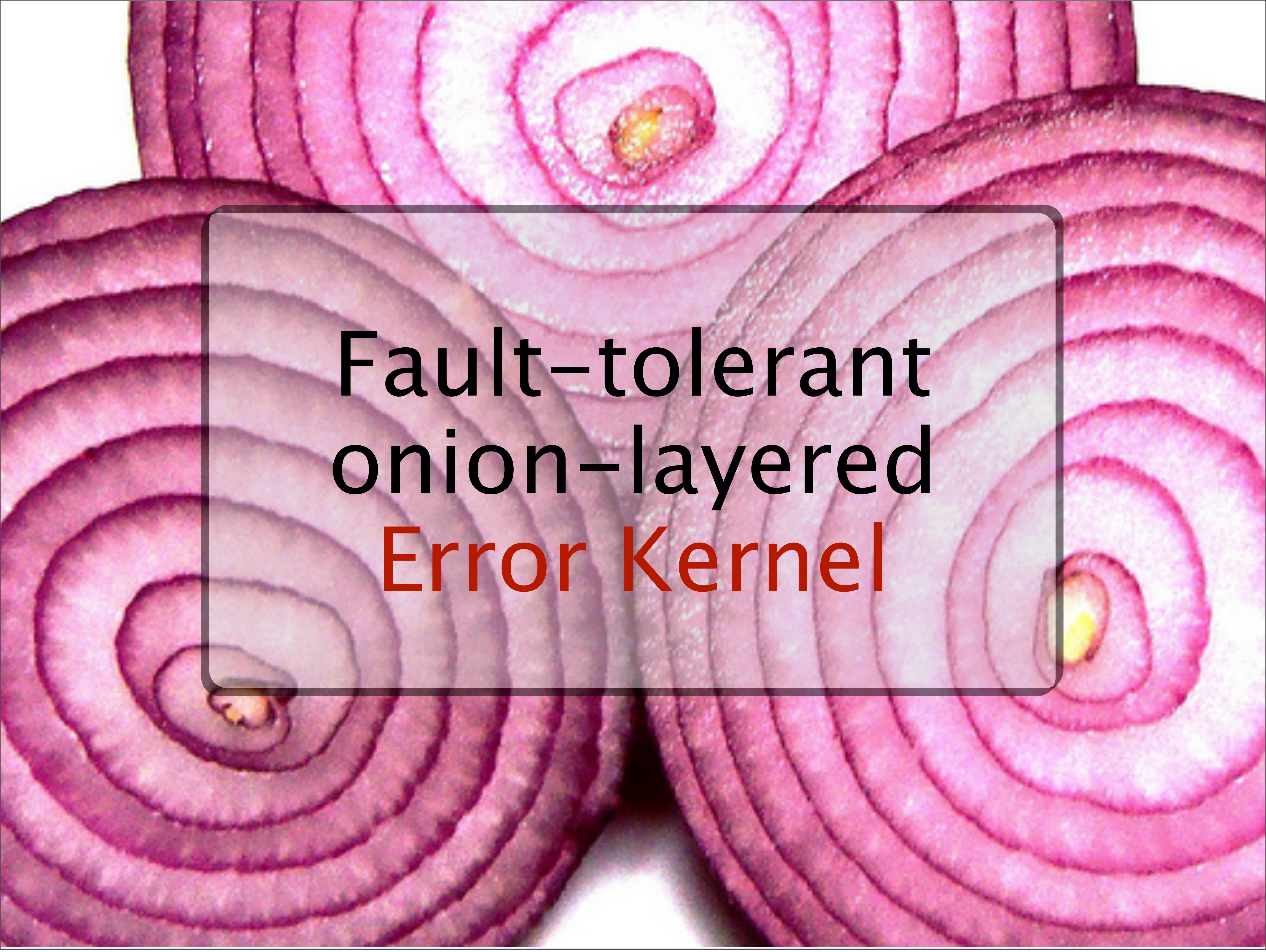
...let’s take a standard OO application



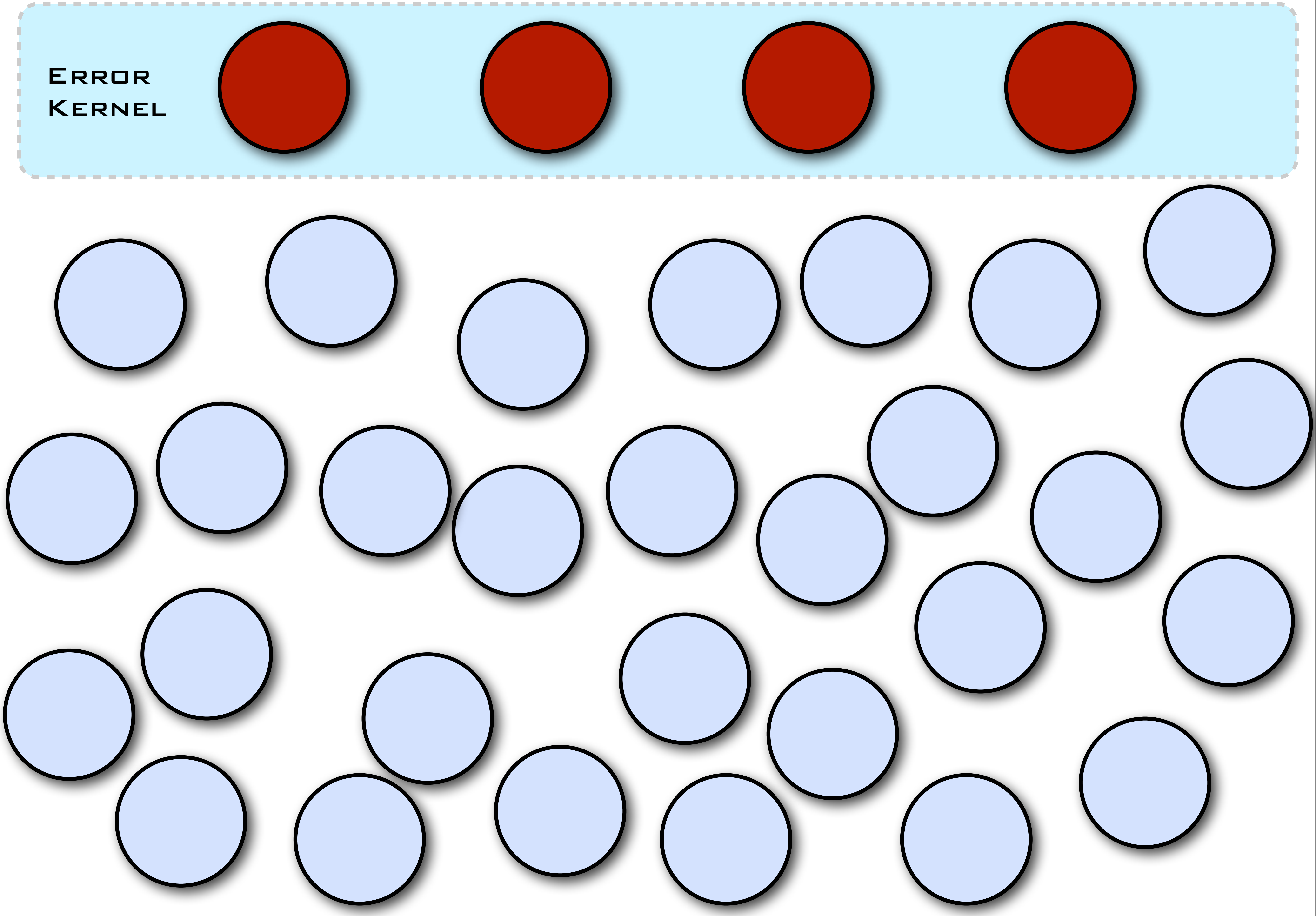


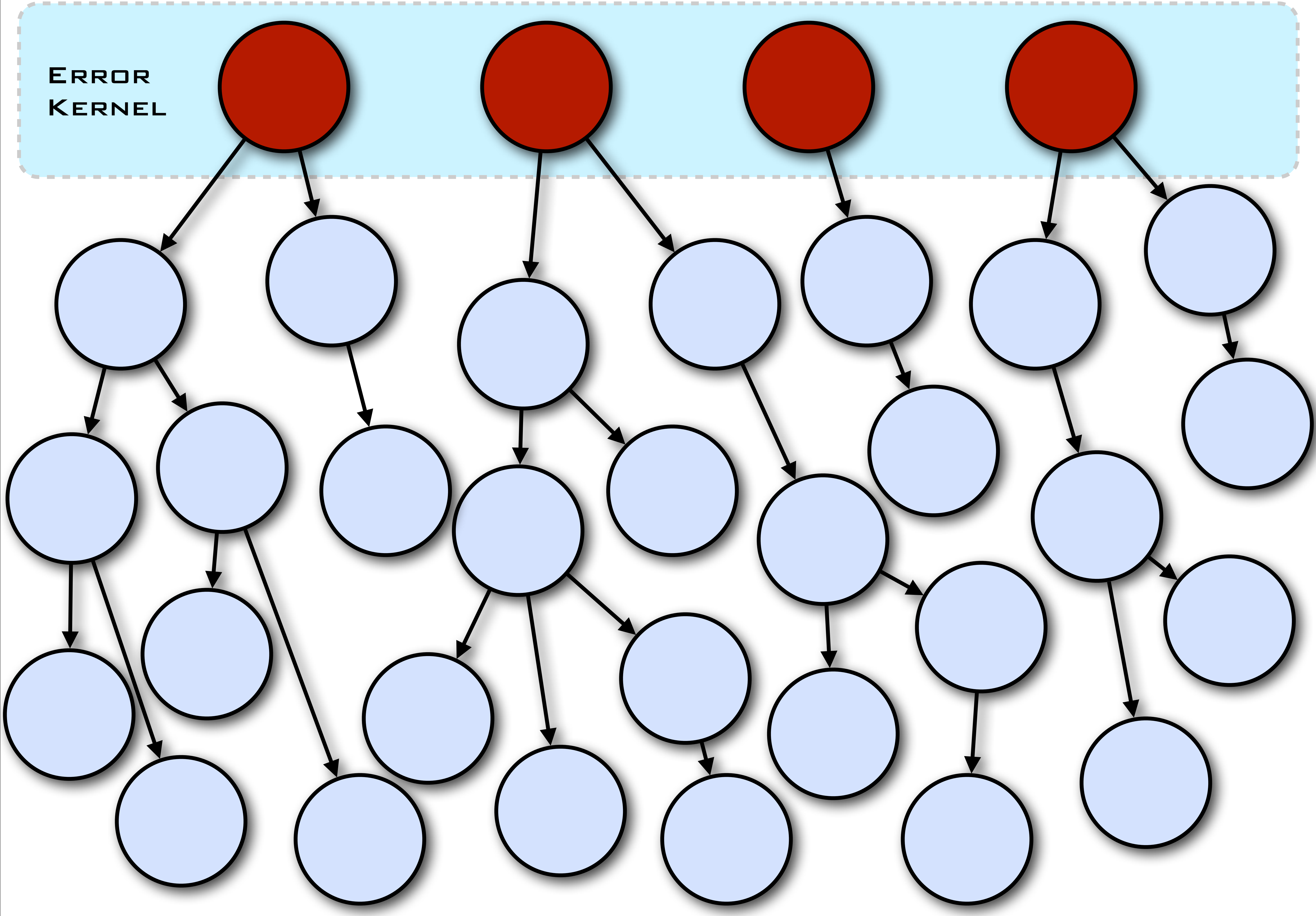
Which components have critically important state and explicit error handling?

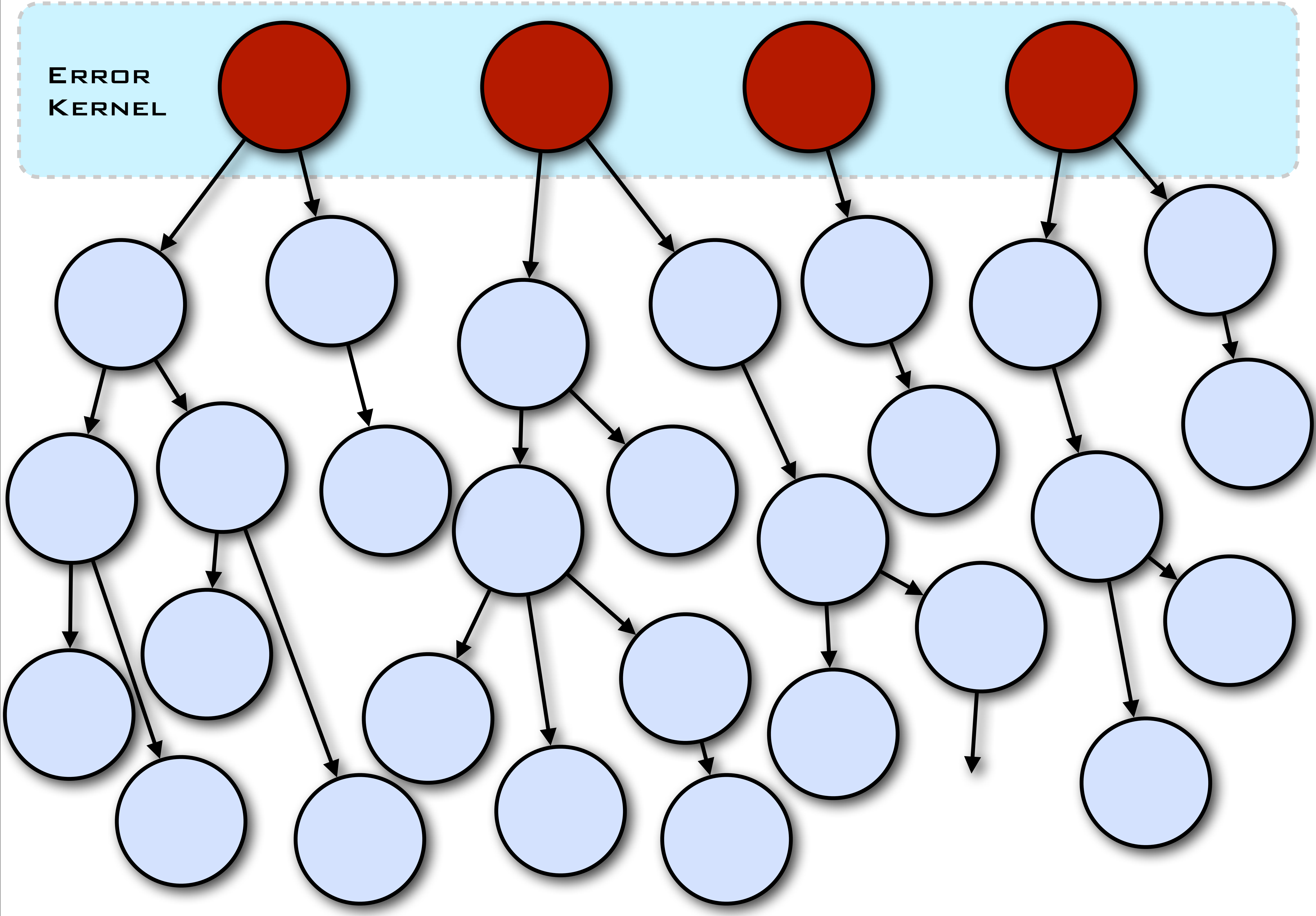


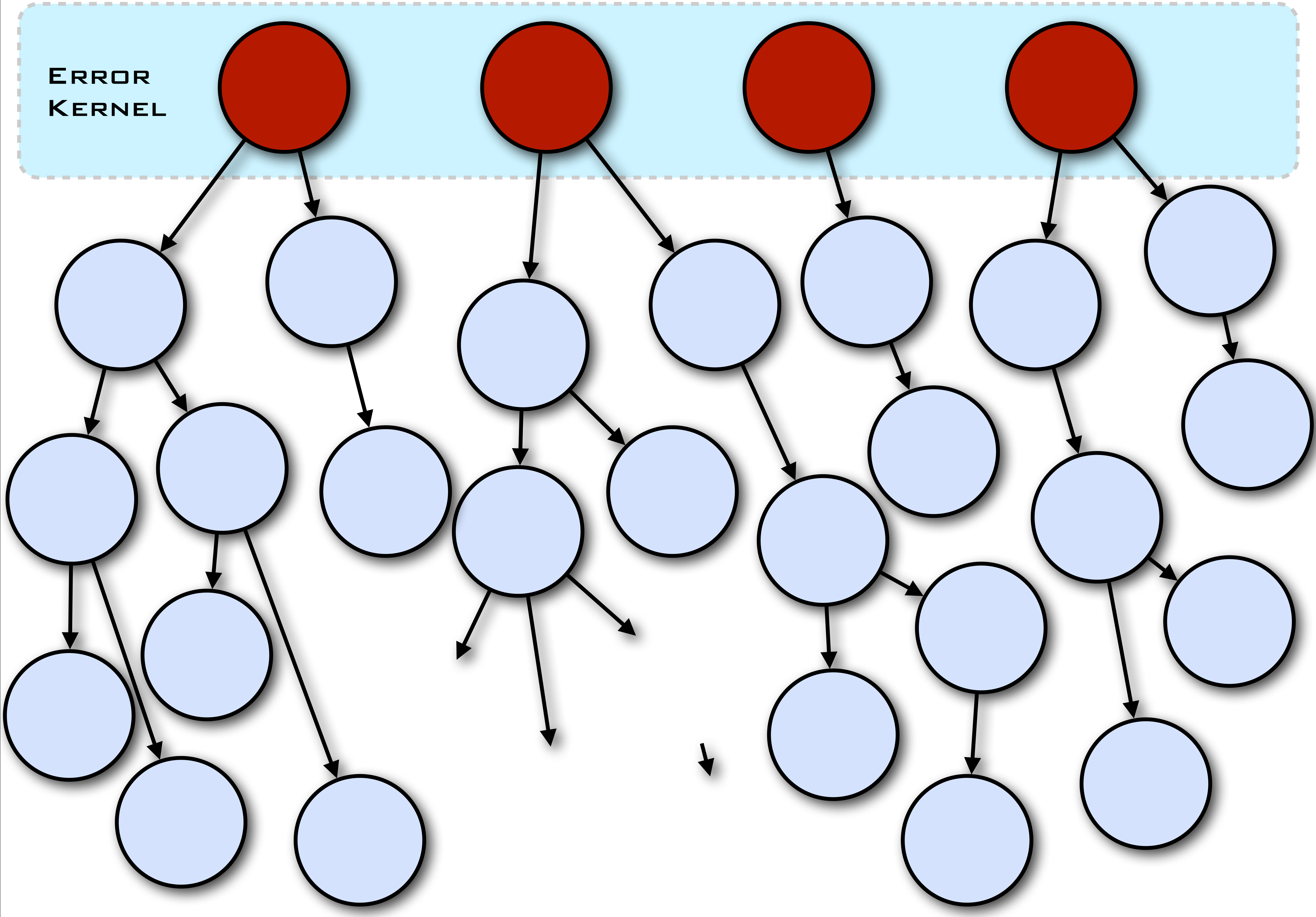


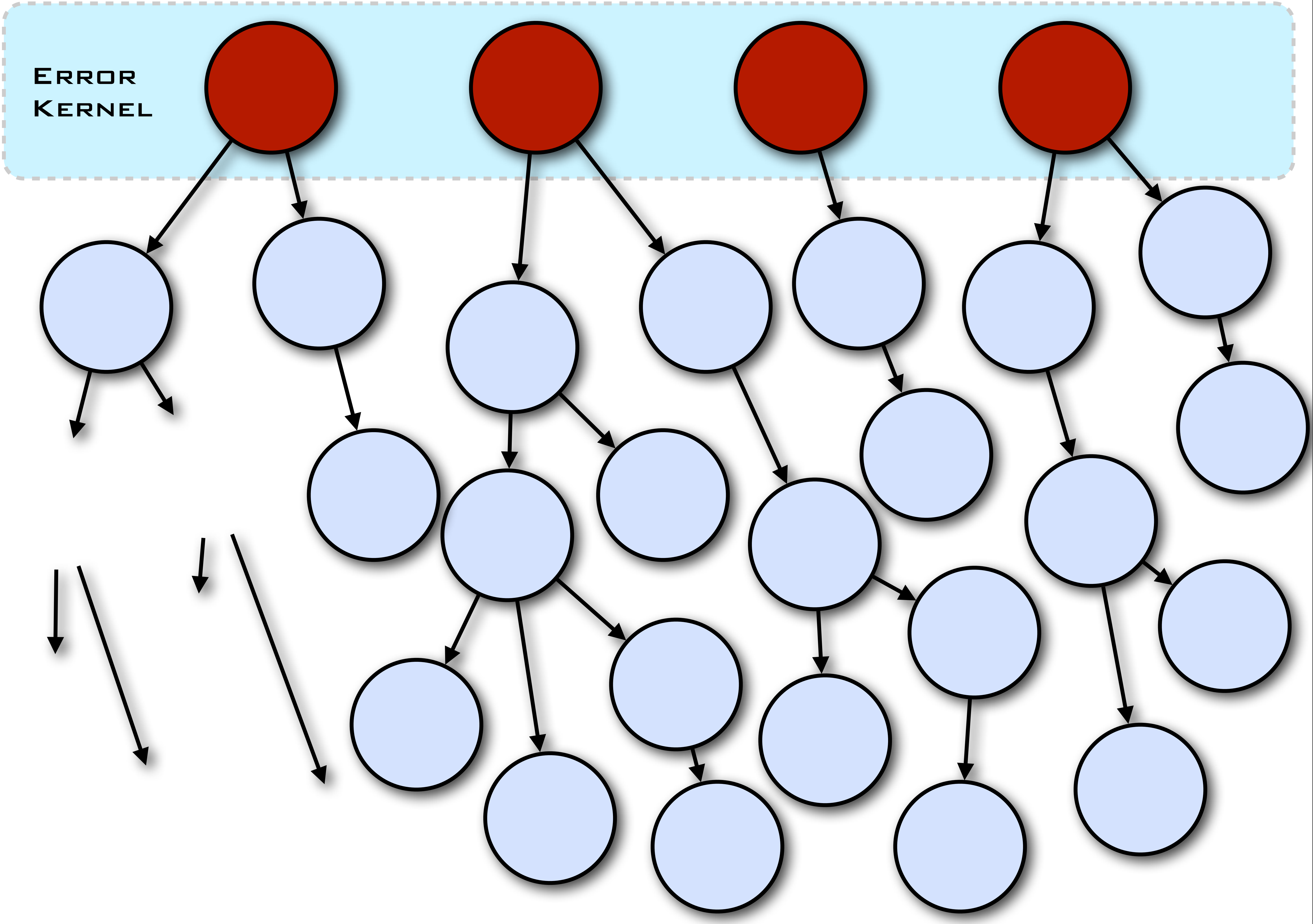
Fault-tolerant onion-layered Error Kernel

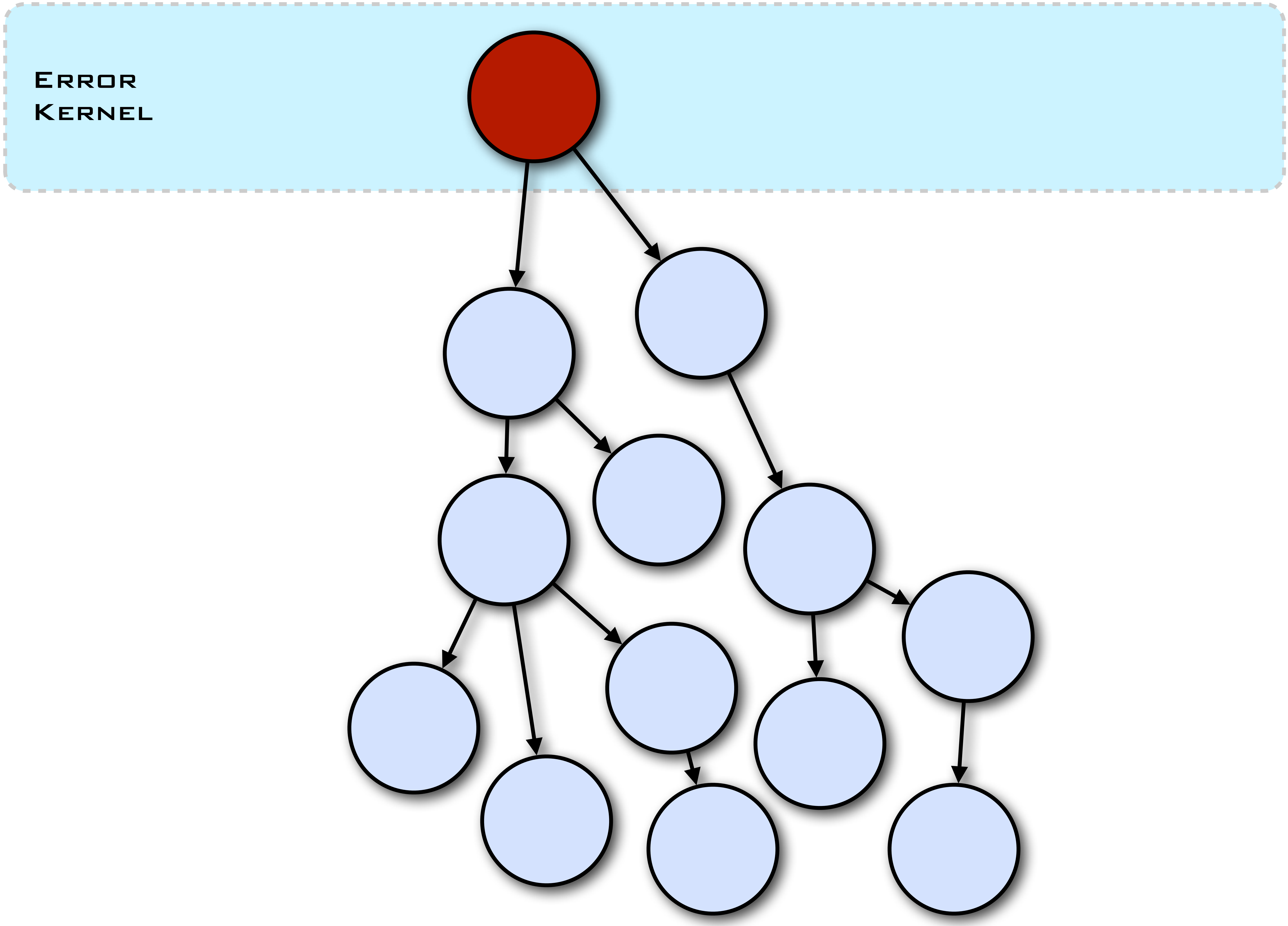

Error
Kernel

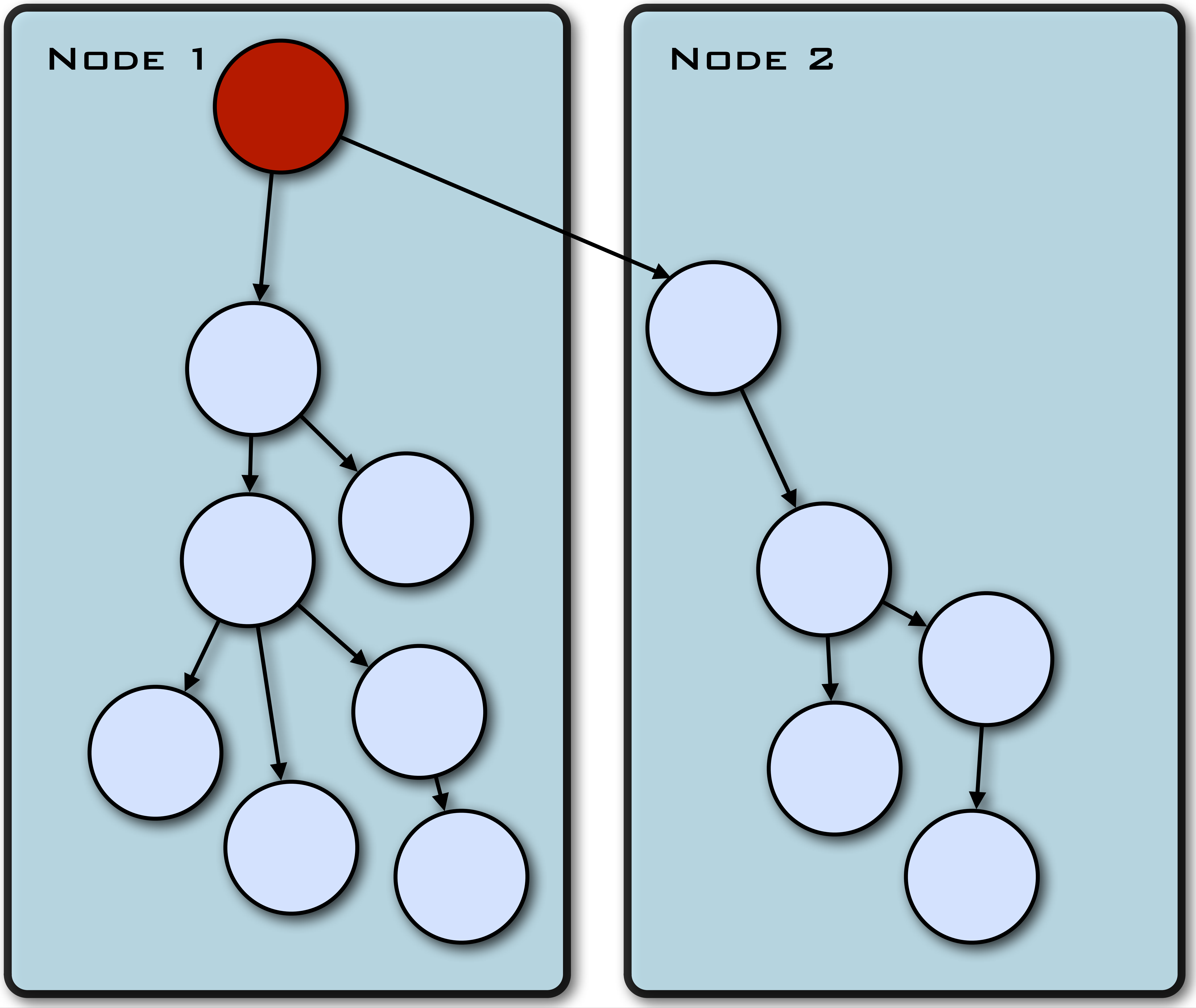

Error Kernel


Error
Kernel


Error
Kernel


Error
Kernel


Error
Kernel

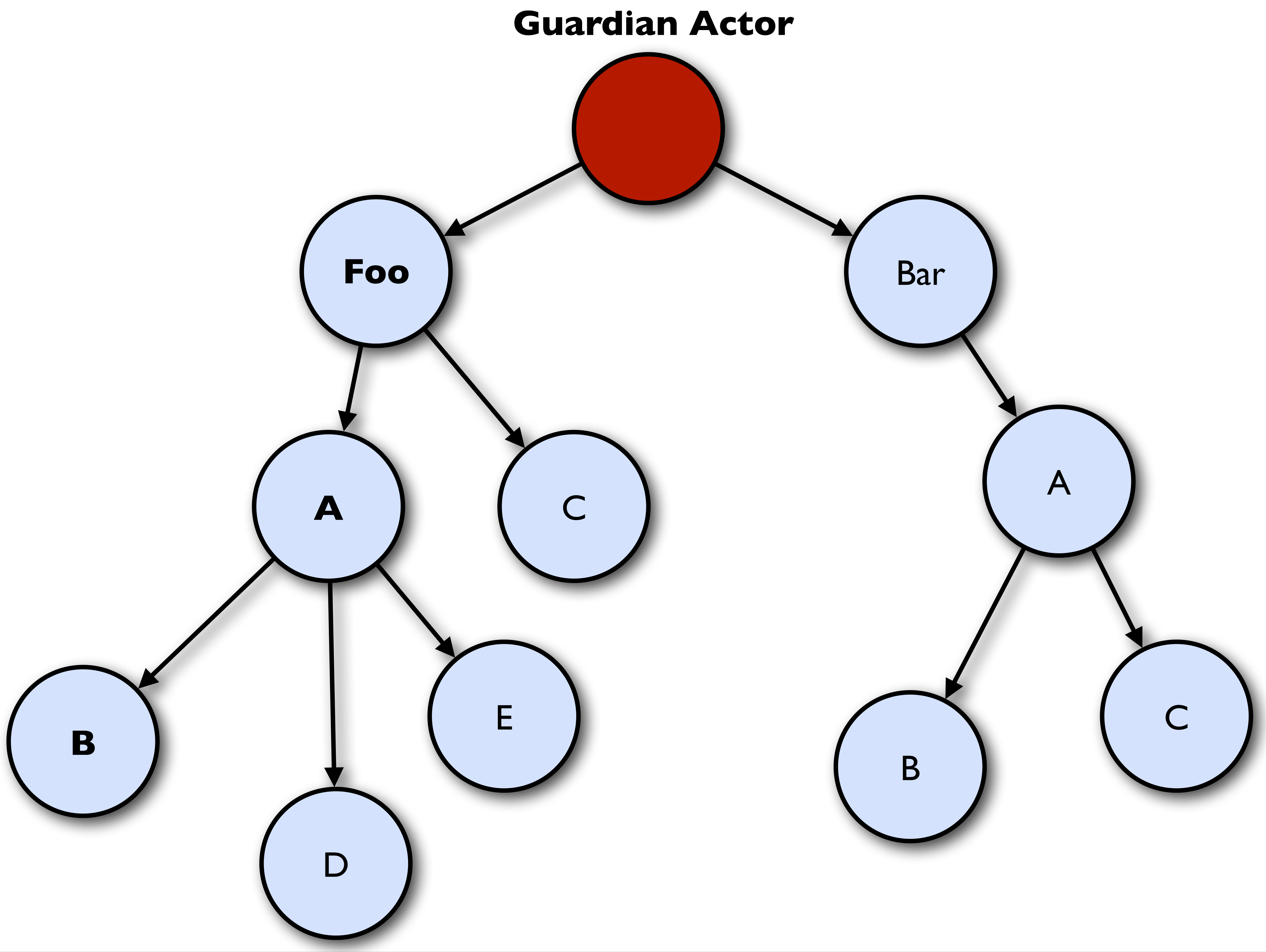

Node 1
Node 2

Parental automatic supervision

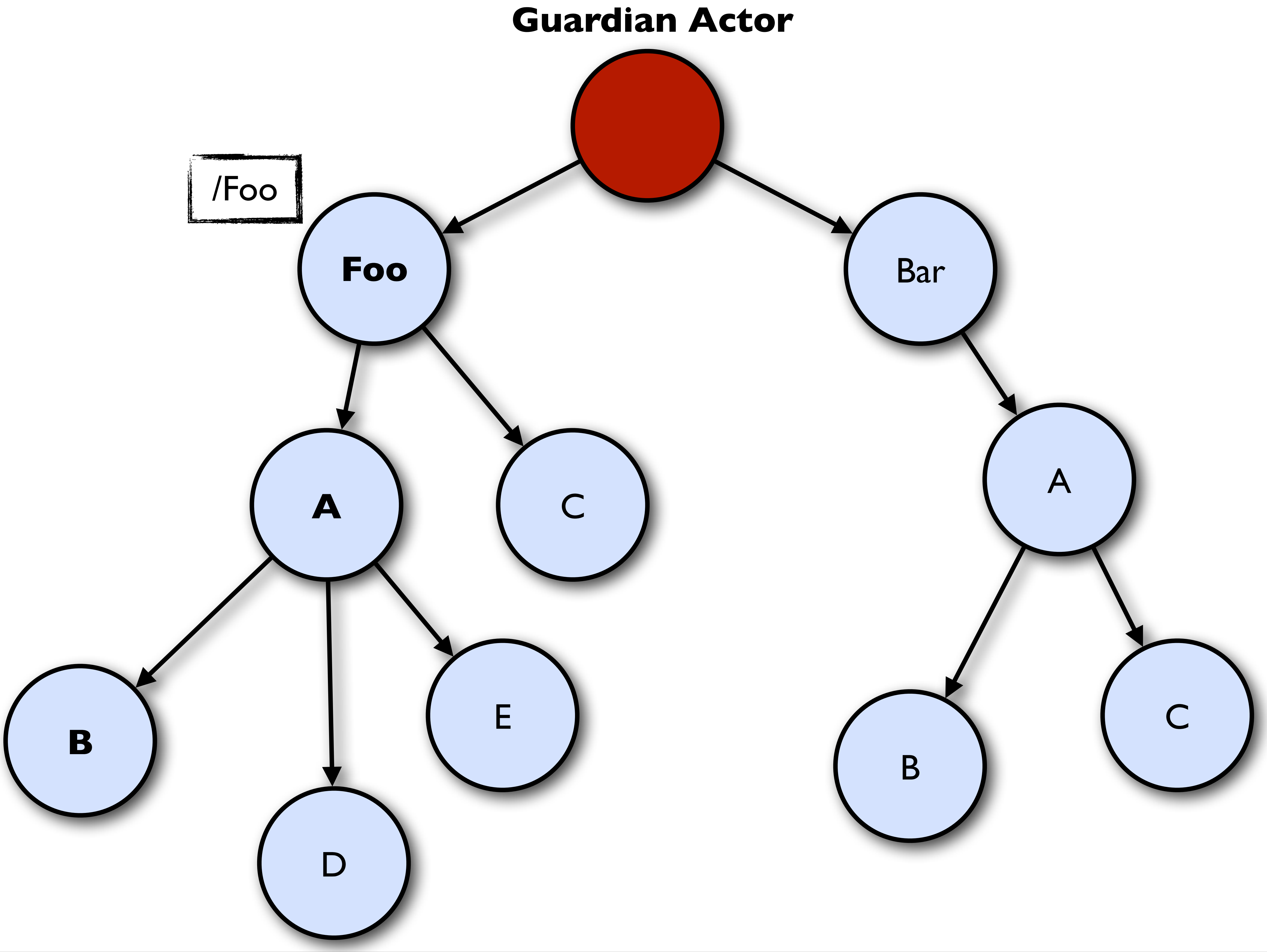
```
// from within an actor
context.actorOf[MyActor]
```

transparent and automatic fault handling by design

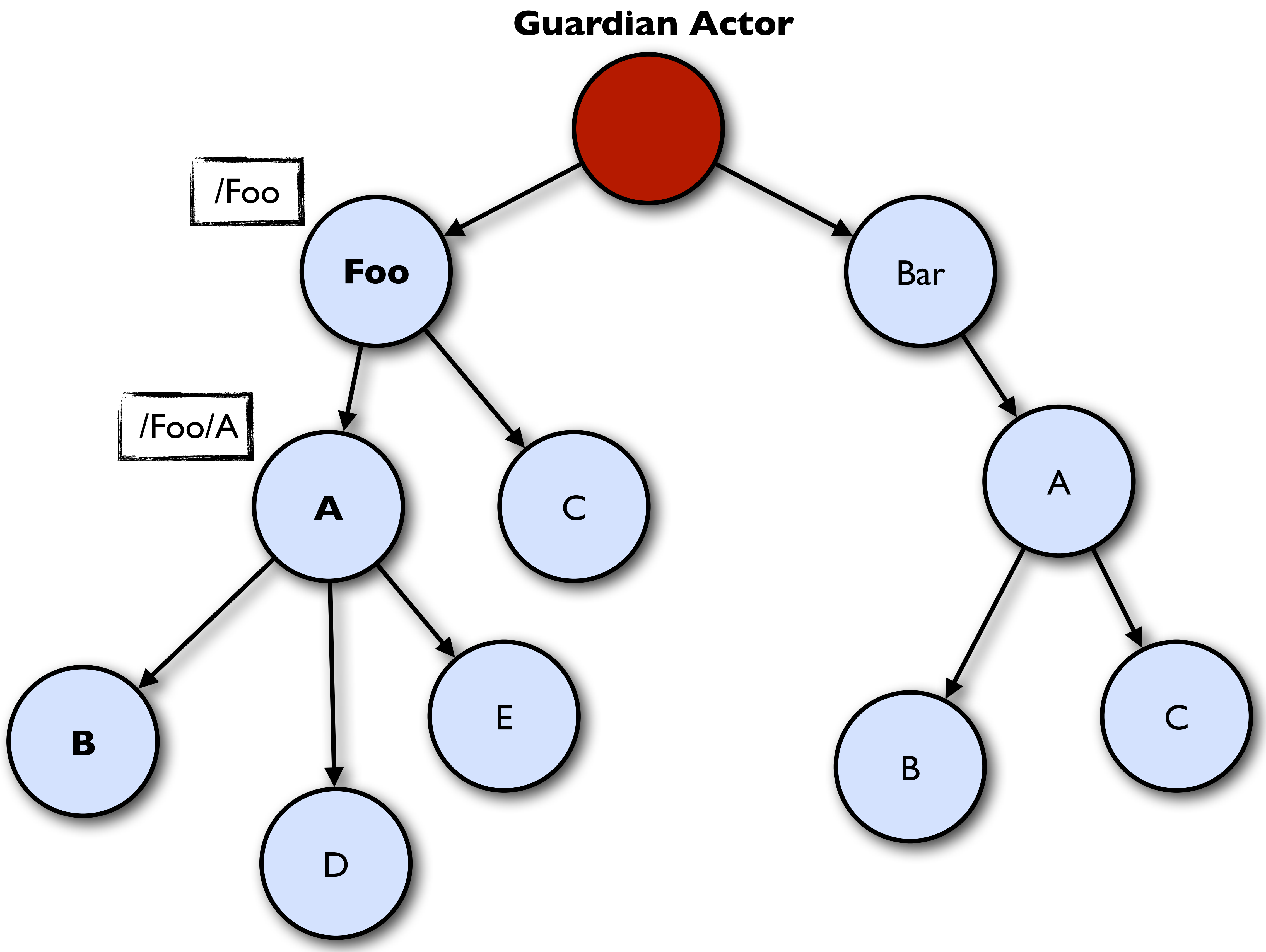
Parental automatic supervision



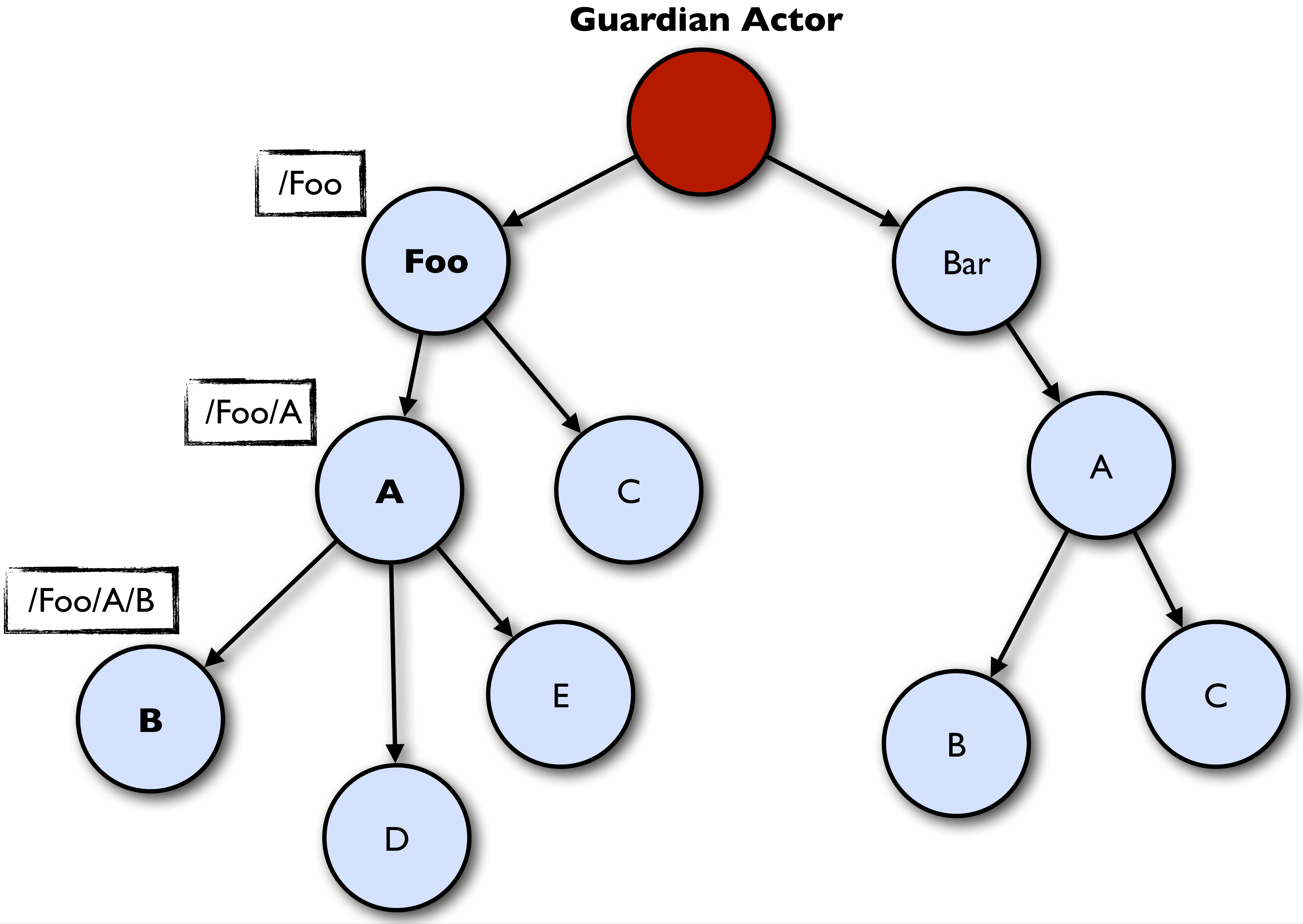
Parental automatic supervision



Parental automatic supervision



Parental automatic supervision



Supervision

```
class MySupervisor extends Actor {
  faultHandler = OneForOneStrategy({
   case _: ActorKilledException => Stop
   case _: ArithmeticException  => Resume
   case _: Exception            => Restart
   case _                       => Escalate
  },
  maxNrOfRetries  = None,
  withinTimeRange = None)

  def receive = {
    case NewUser(name) =>
      ... = context.actorOf[User](name)
  }
}
```

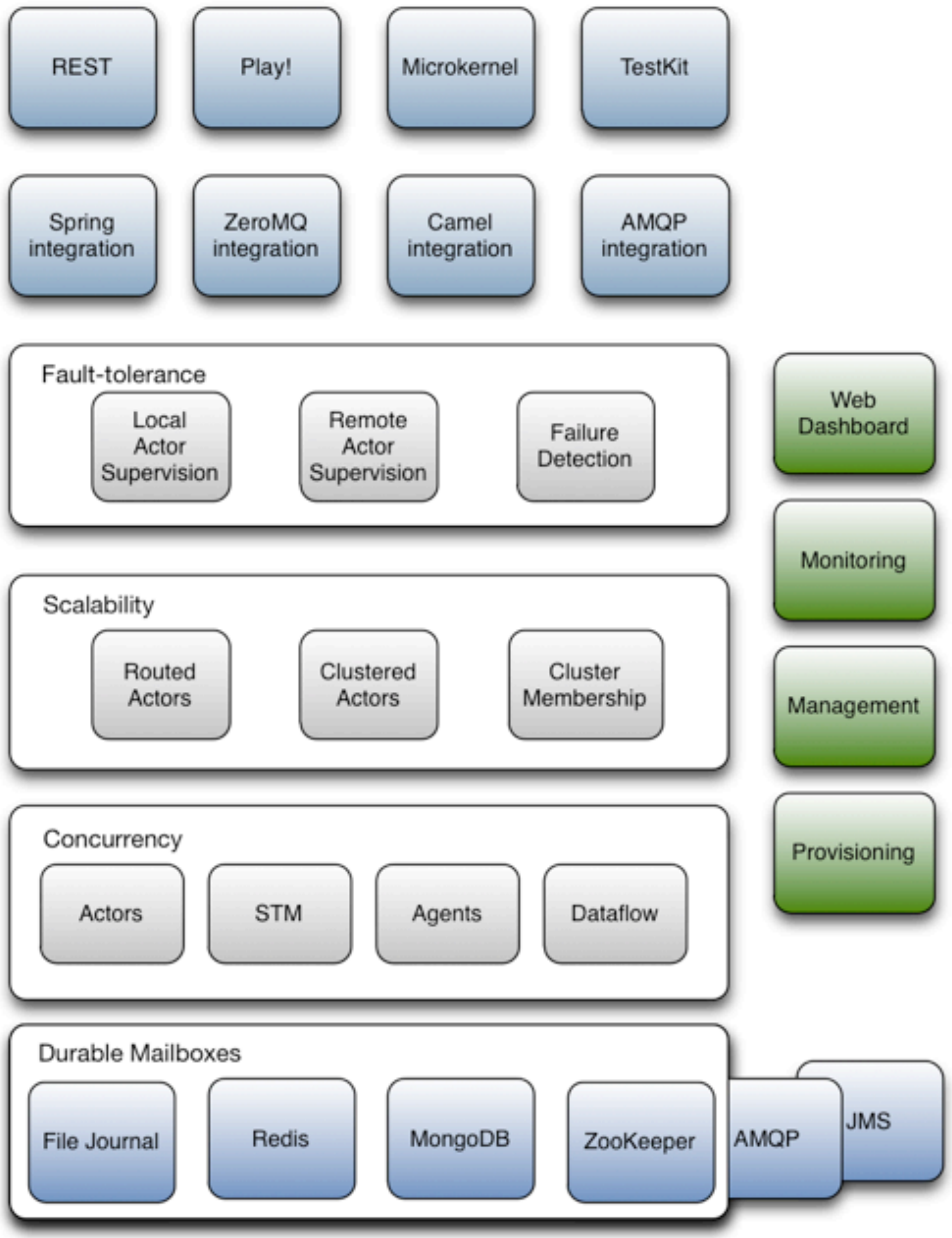
Supervision

```
class MySupervisor extends Actor {
  faultHandler = AllForOneStrategy({
   case _: ActorKilledException => Stop
   case _: ArithmeticException  => Resume
   case _: Exception            => Restart
   case _                       => Escalate
  },
  maxNrOfRetries  = None,
  withinTimeRange = None)

  def receive = {
    case NewUser(name) =>
      ... = context.actorOf[User](name)
  }
}
```

Manage failure

```scala
class FaultTolerantService extends Actor {
  ...
  override def preRestart(
    reason: Throwable, message: Option[Any]) = {
    ... // clean up before restart
  }
  override def postRestart(reason: Throwable) = {
    ... // init after restart
  }
}
```

ARCHITECTURE



TYPESAFE STACK ADD-ONS

Get it and learn more

http://akka.io

http://typesafe.com

https://gist.github.com/1388237

EOF