Typesafe

Scaling Up & Out with Actors: Introducing Akka

√

Akka Tech Lead

Email: viktor.klang@typesafe.com
Twitter: @viktorklang

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

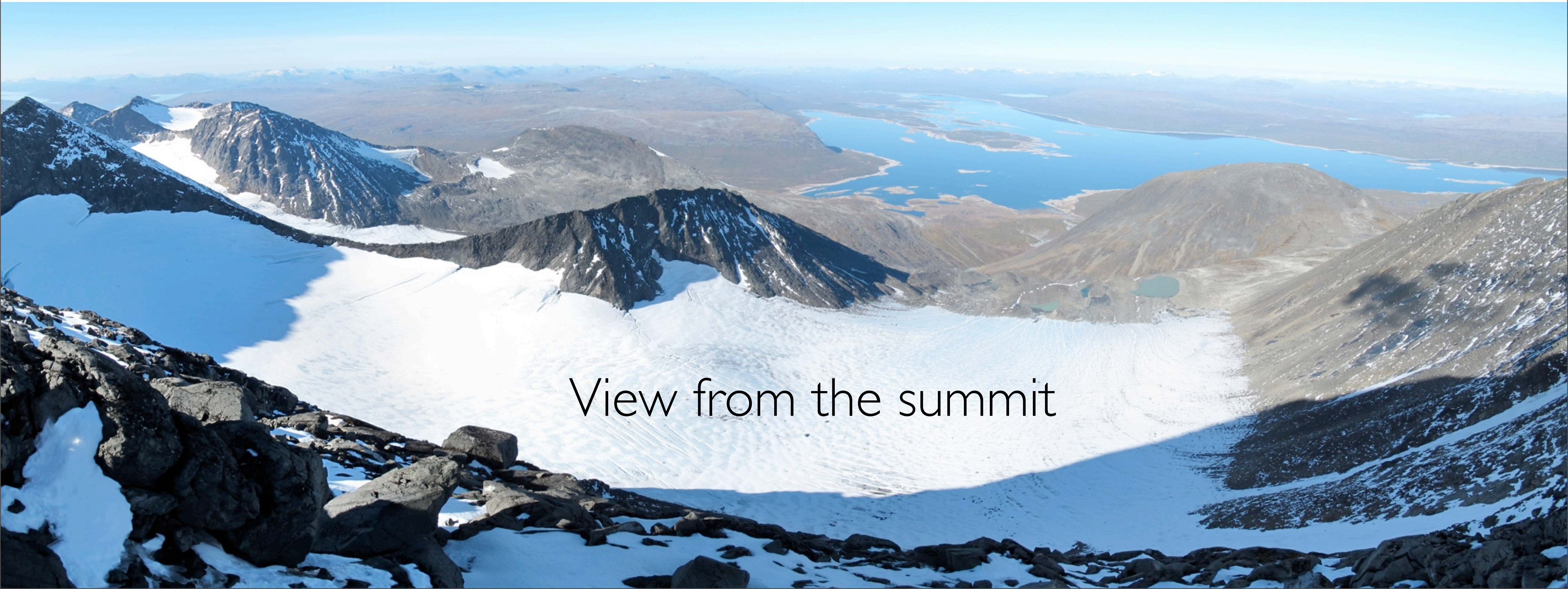
Introducing akka

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


View from the summit

Vision

Simpler

- Concurrency
- Scalability
- Fault-tolerance

Vision

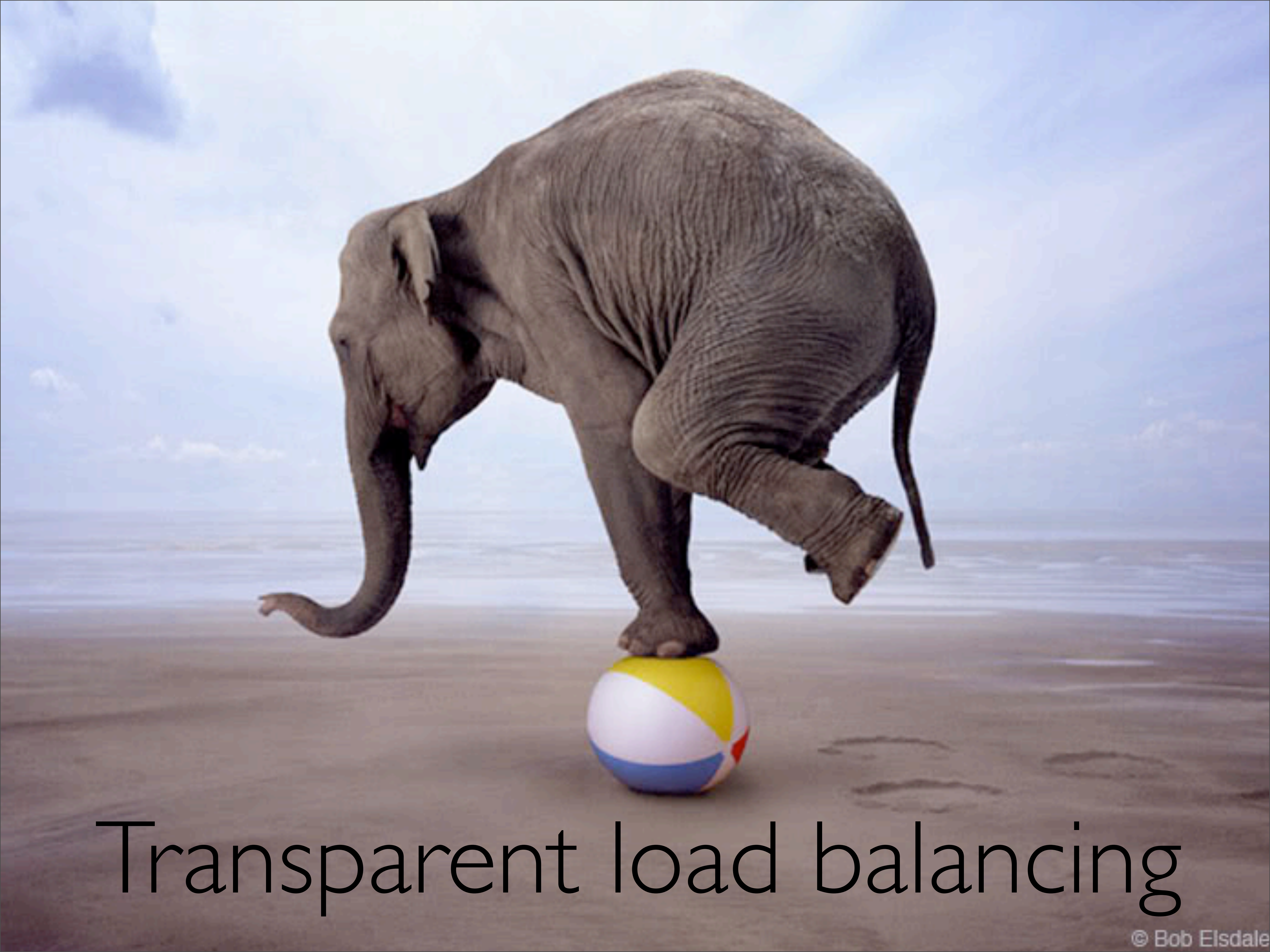
...with a single unified

- Programming Model
- Managed Runtime
- Open Source Distribution

Manage system overload

Scale up & Scale out

Replicate and distribute for fault-tolerance



Transparent load balancing

INTRODUCING

Akka **2.0**

Akka **2.0-RC1**

636 tickets **closed**

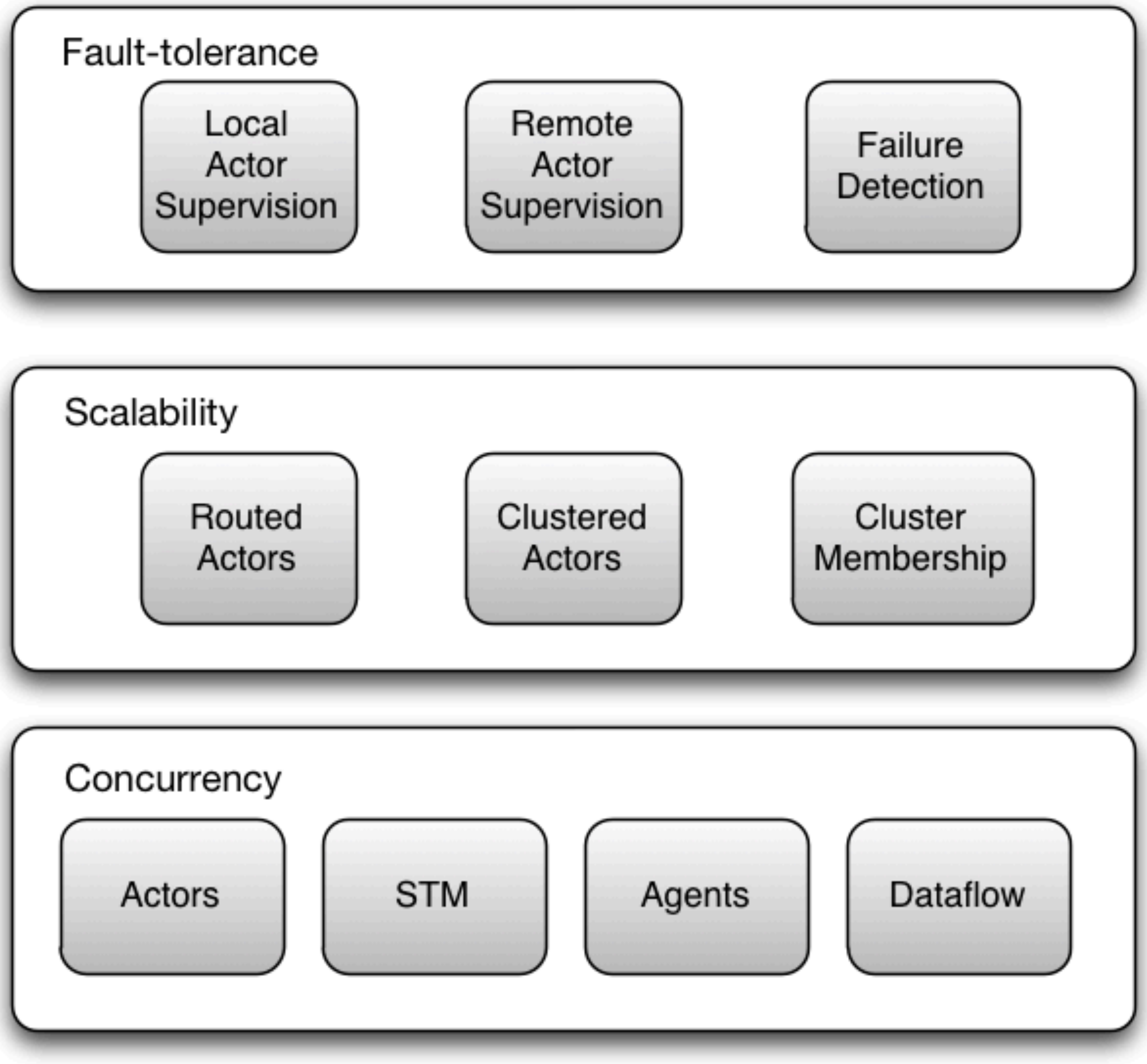
1011 files **changed**

96261 lines **added**

56733 lines **removed**

56733 lines **removed**

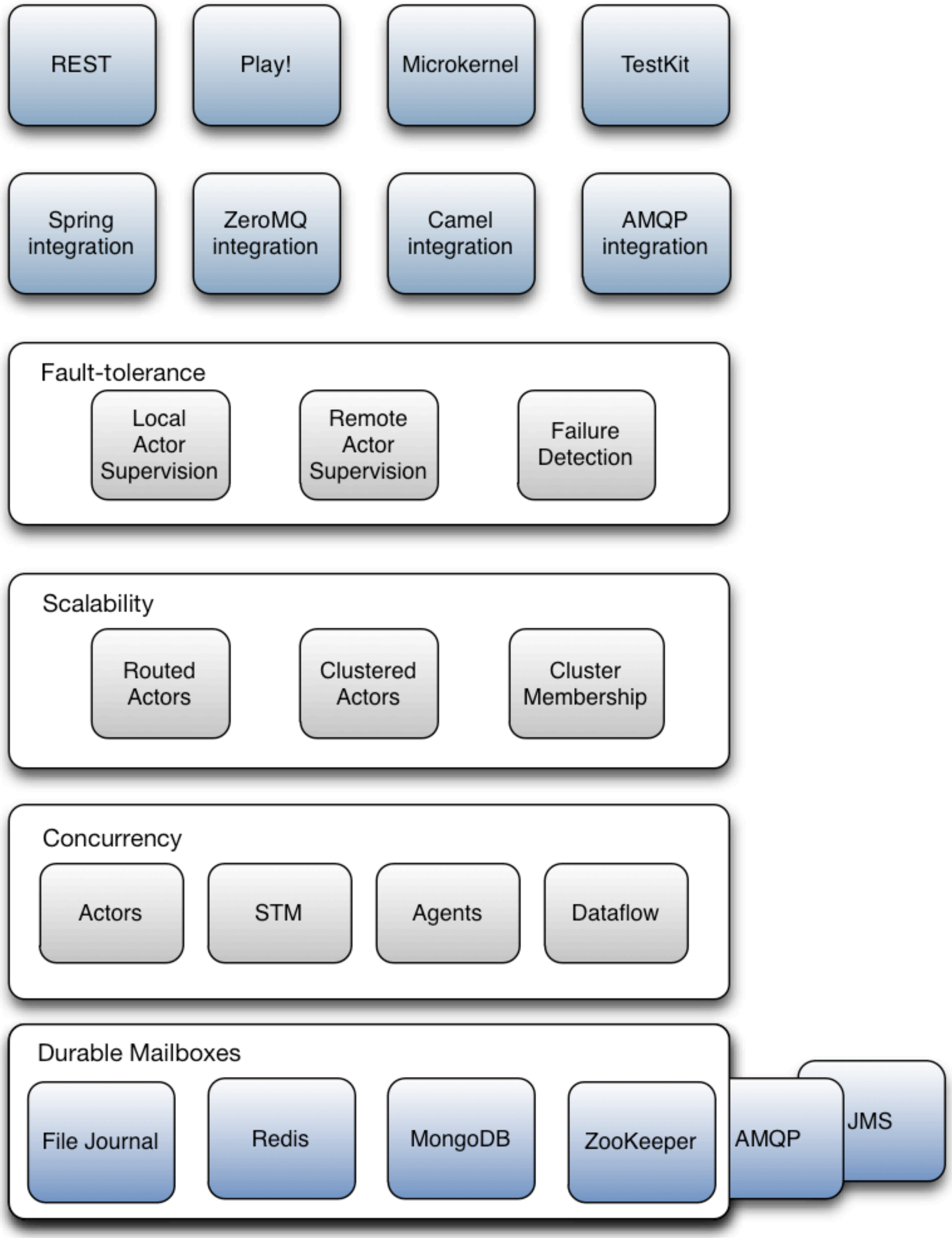
ARCHITECTURE



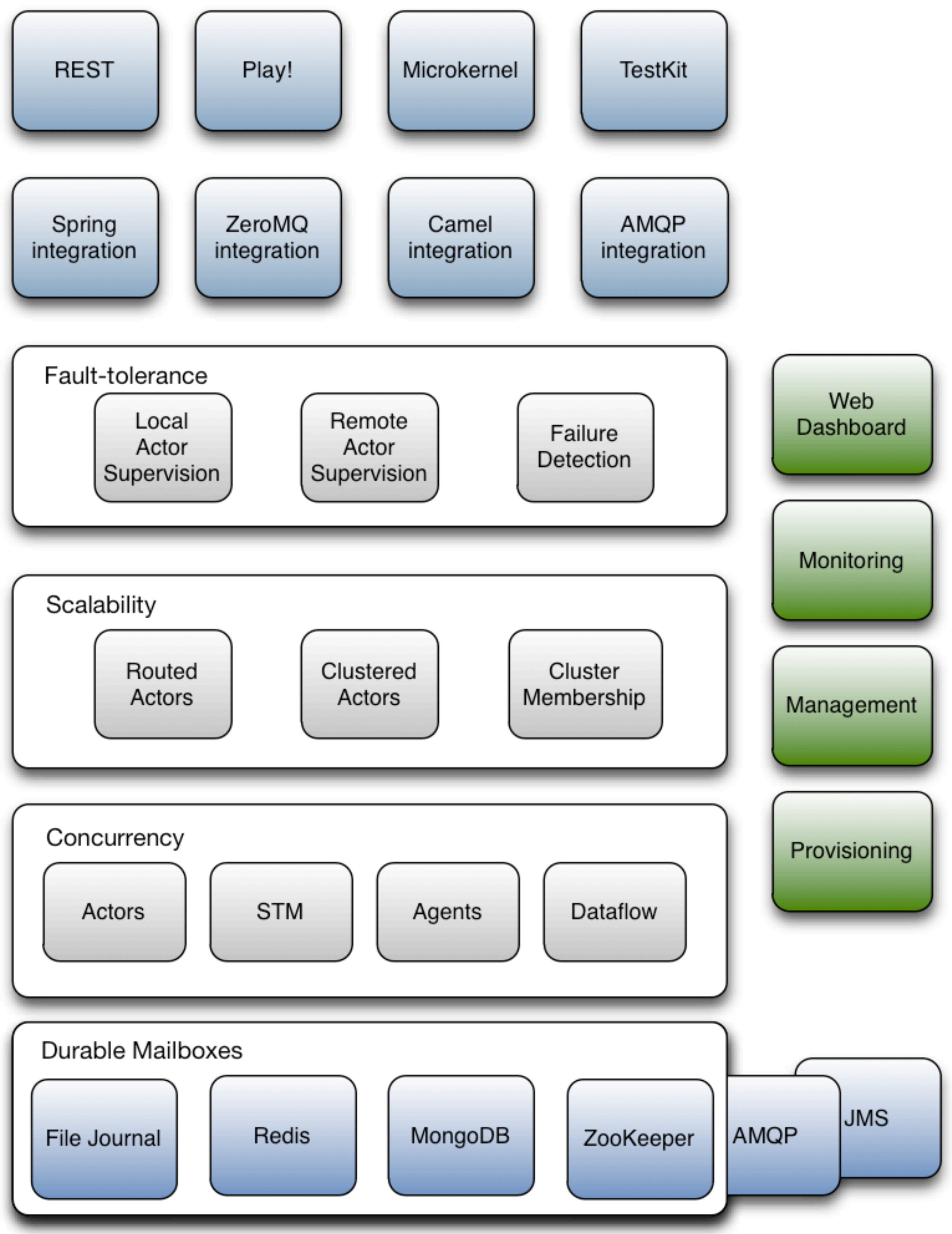
CORE SERVICES

ARCHITECTURE

ADD-ON MODULES



ARCHITECTURE



TYPESAFE STACK ADD-ONS

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

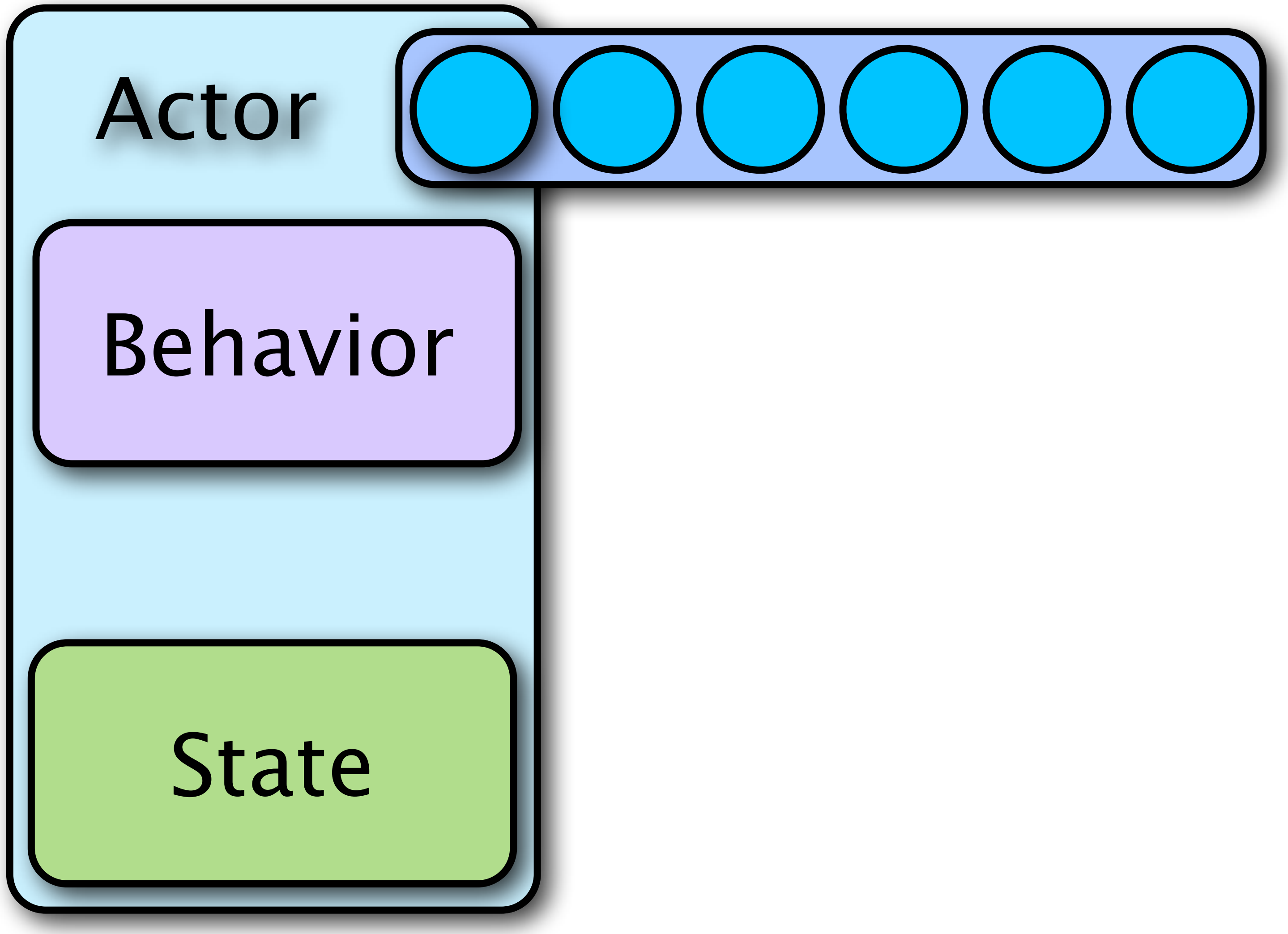
- 3D simulation engines

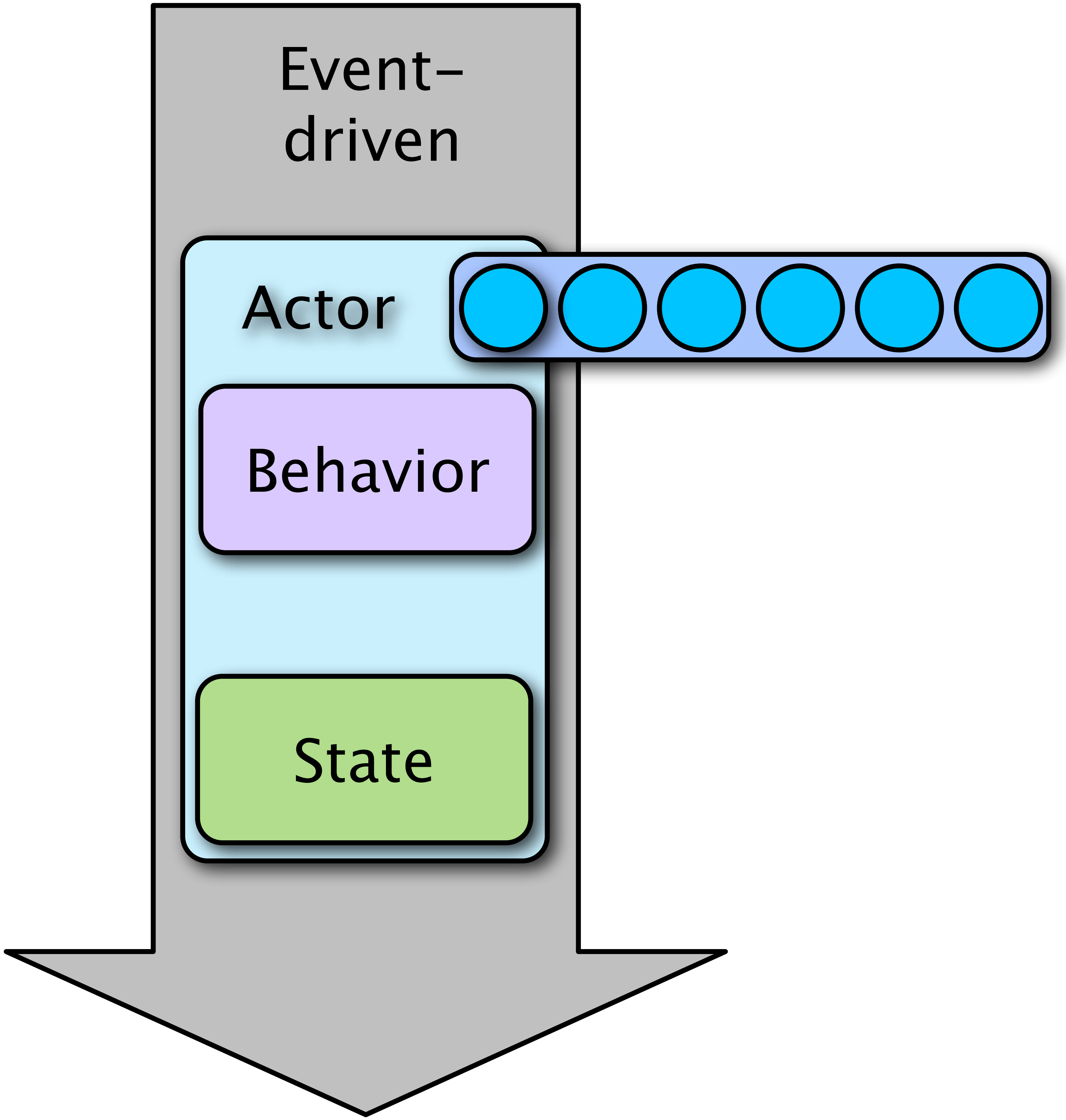
E-COMMERCE

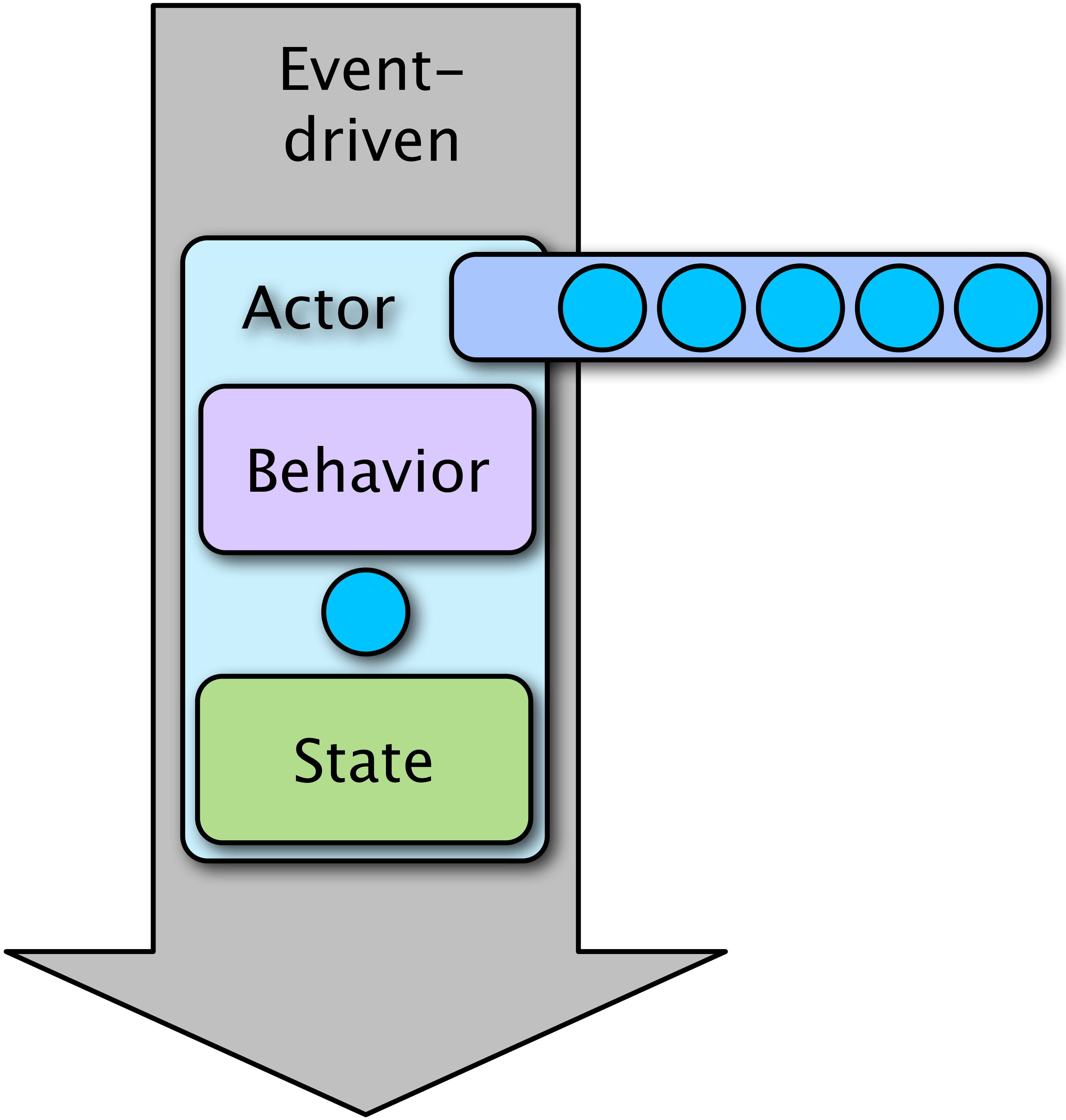
- Social media community sites

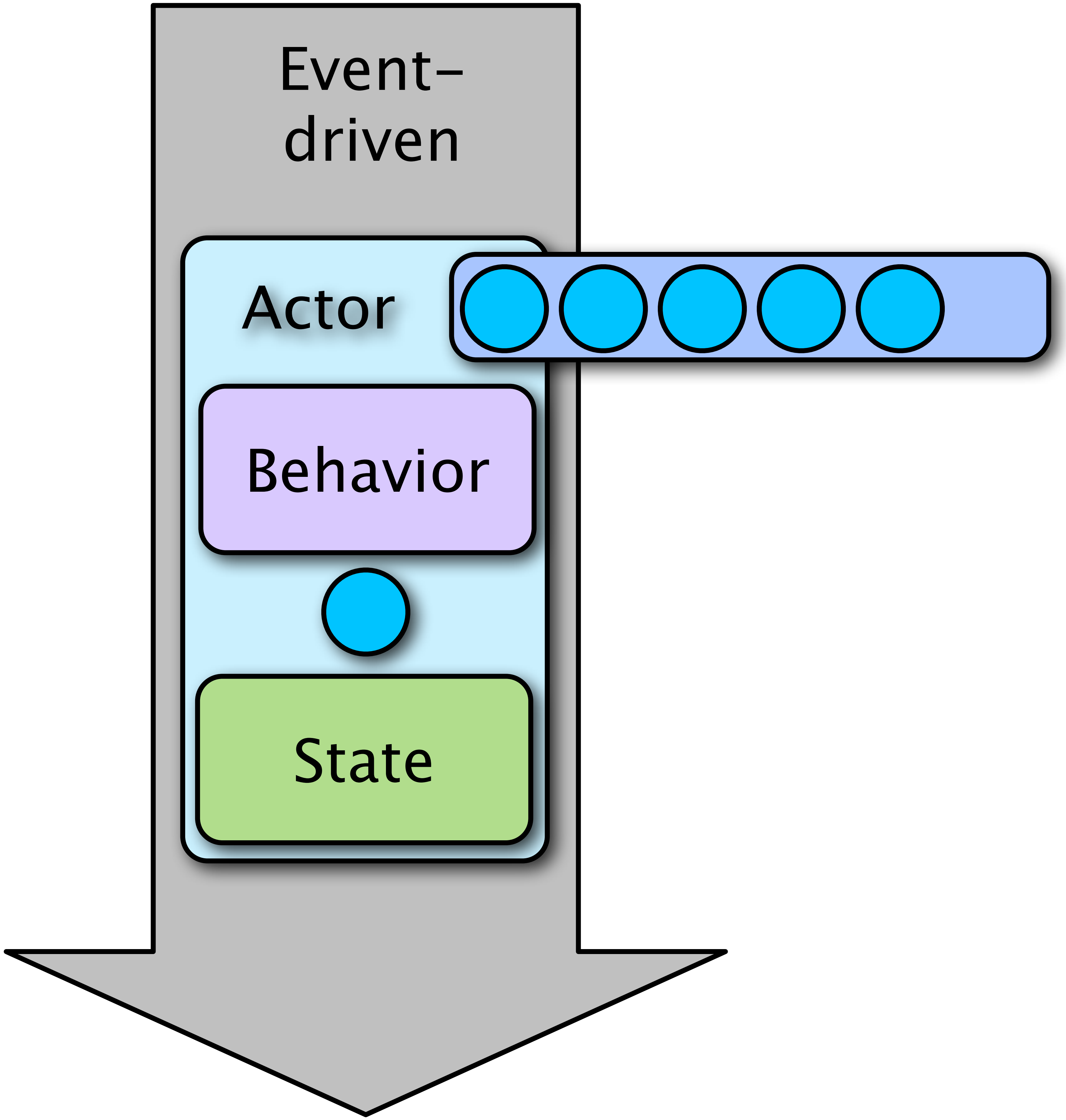
Scale **up**

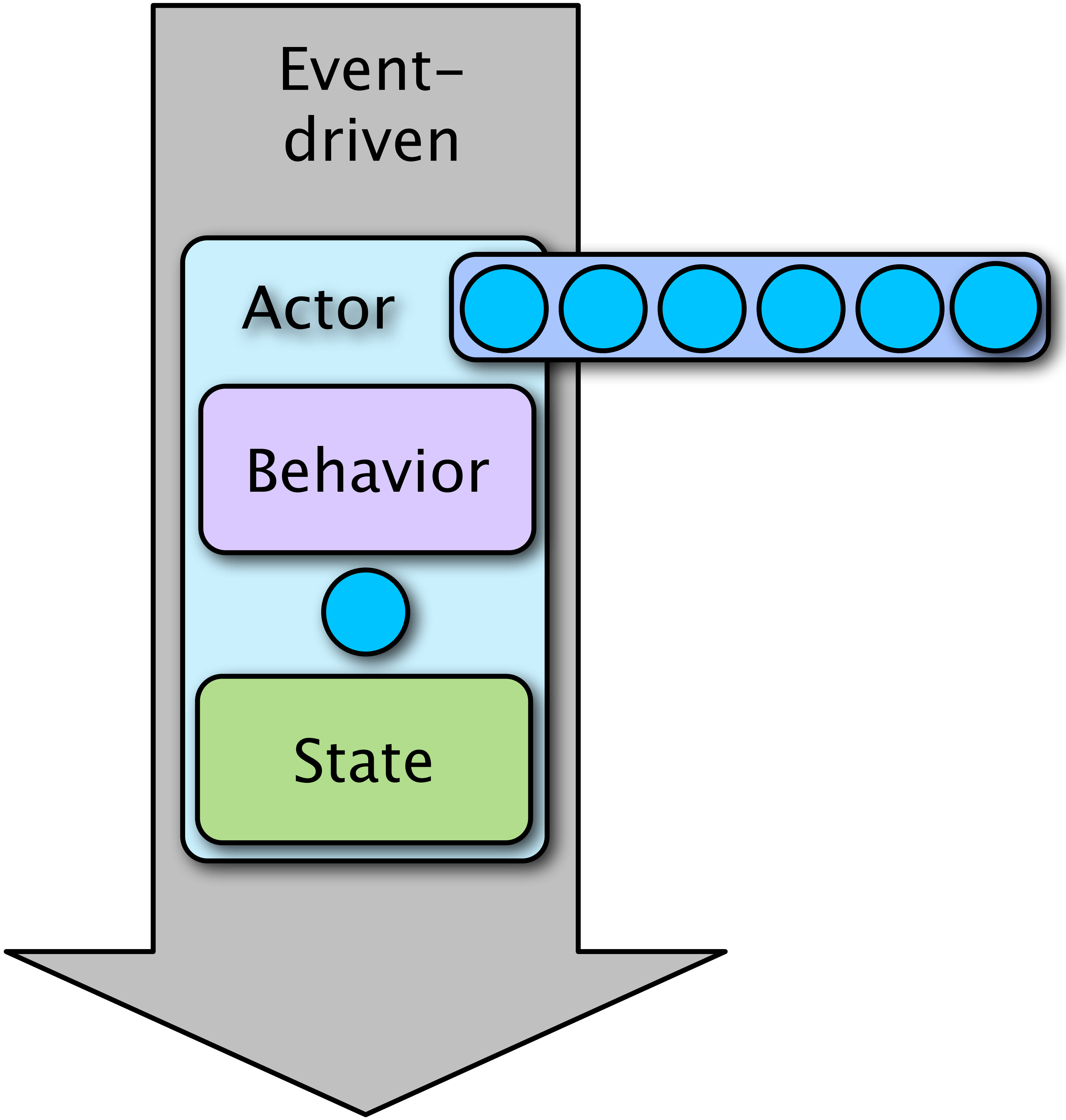
What is an Actor?

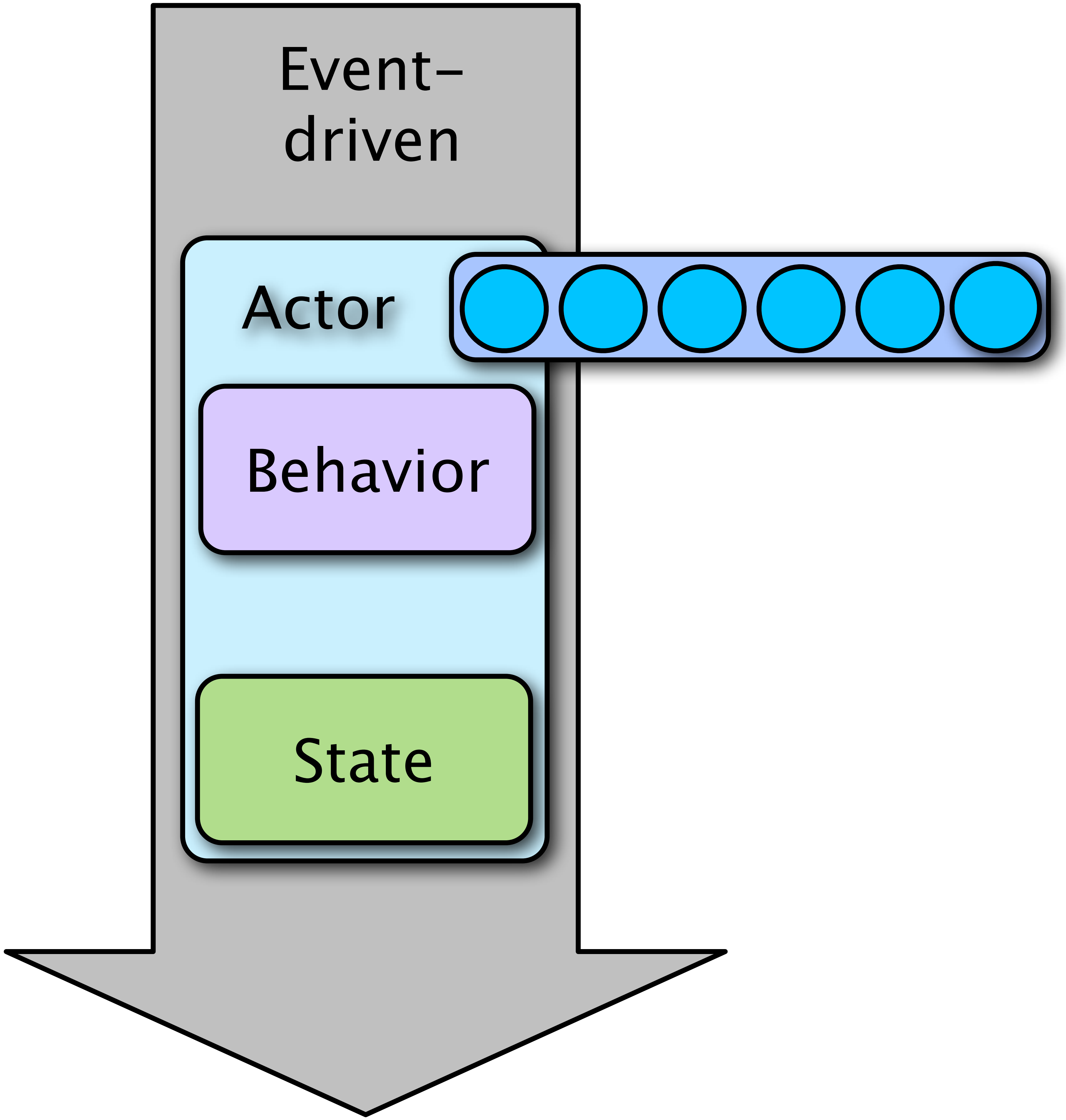

Actor
Behavior
State

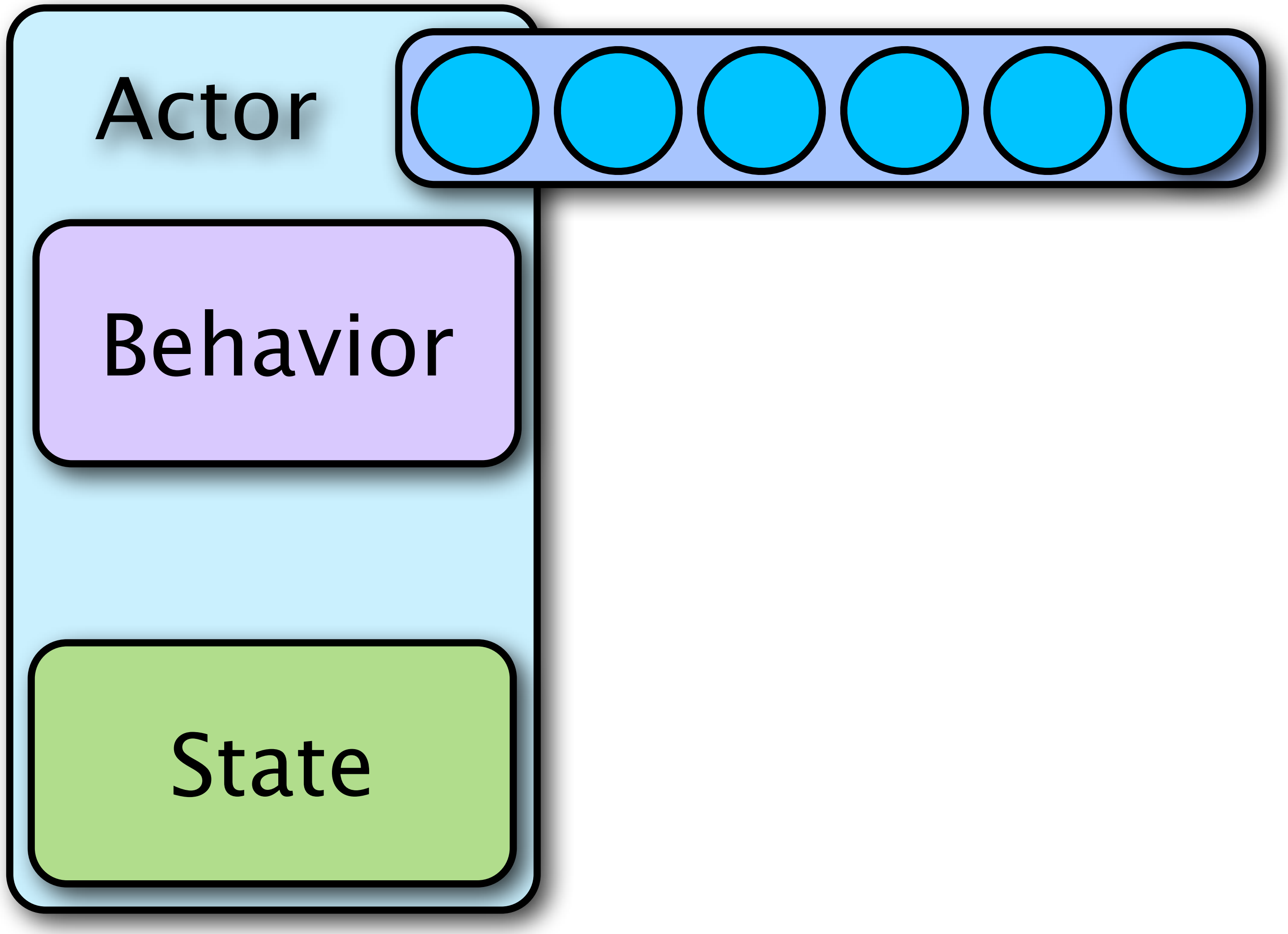

Event-
driven
Actor
Behavior
State

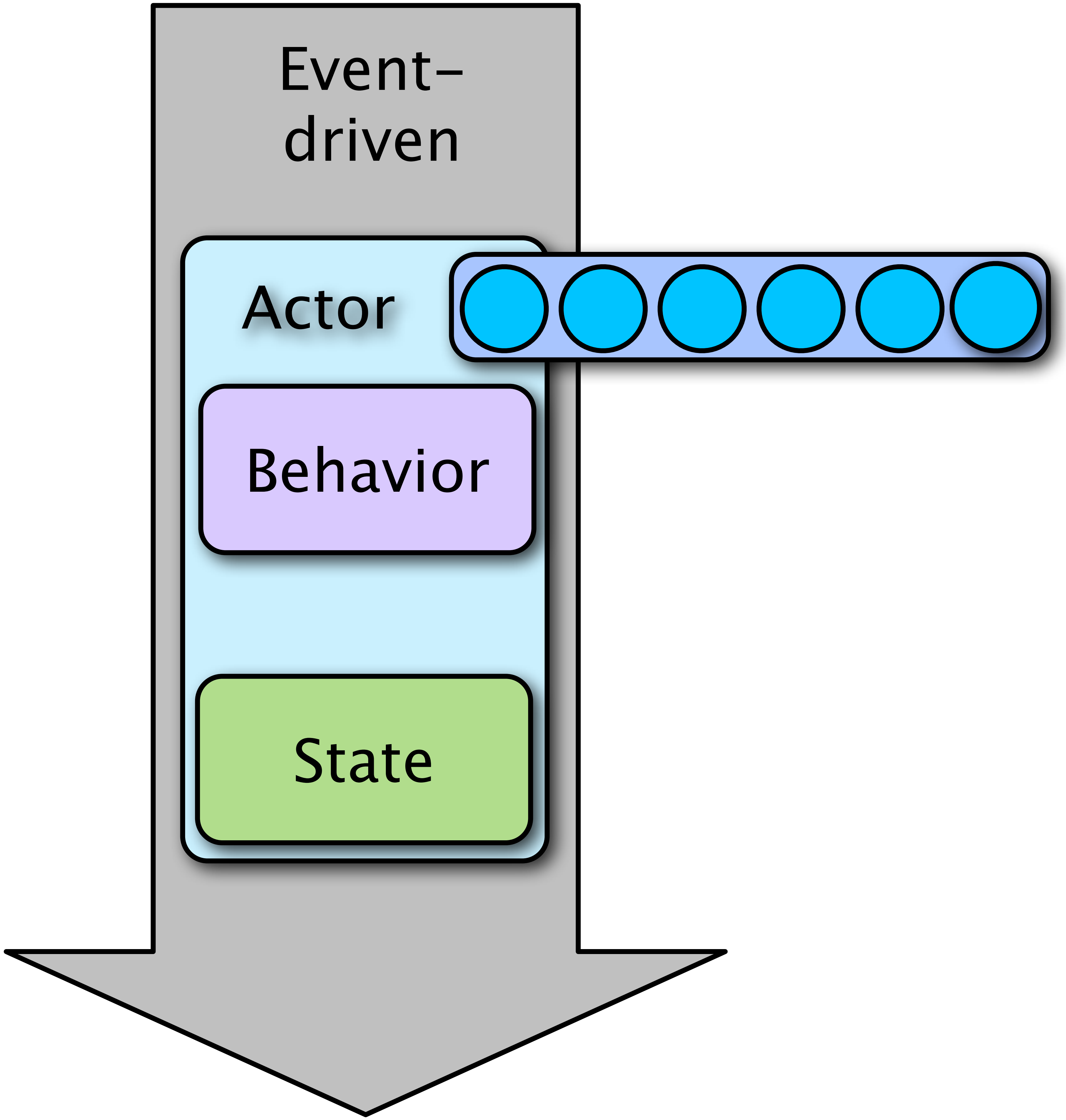

Event-
driven
Actor
Behavior
State


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

one tool in the toolbox

Actors

```scala
case object Tick

class Counter extends Actor {
  var counter = 0

  def receive = {
    case Tick =>
      counter += 1
      println(counter)
  }
}
```

Scala API

Actors

```
class Counter extends UntypedActor {
  int counter = 0;

  void onReceive(Object msg) {
    if (msg.equals(“Tick”)) {
      counter += 1;
      System.out.println(counter);
    }
  }
}
```

Java API

Create Application

```
val conf = ConfigFactory.load(“application”)

val system = ActorSystem(“my-app”, conf)
```

Scala API

Create Application

```
Config conf =
          ConfigFactory.load("application");

ActorSystem system =
         ActorSystem.create("my-app", conf);
```

Java API

Create Actors

```
val counter = system.actorOf(Props[Counter])
```

counter is an **ActorRef**

Creates a top-level actor

Scala API

Create Actors

```
ActorRef counter =
  system.actorOf(new Props(Counter.class));
```

Creates a top-level actor

Java API

Stop actors

```
system.stop(counter)
```

...also stops all actors in the hierarchy below

Send: !

```
counter ! Tick
```

fire-forget

Scala API

...or use tell

```
counter tell Tick
```

fire-forget

Scala API

...or use tell

```
counter.tell(tick);
```

fire-forget

Java API

Send: ?

```
import akka.patterns.ask

// returns a future
val future = actor ? message

future onSuccess {
  case x => println(x)
}
```

returns the Future directly

Scala API

...or use ask

```
import akka.patterns.ask

// returns a future
val future = actor ask message

future onSuccess {
  case x => println(x)
}
```

returns the Future directly

Scala API

Reply

```
class SomeActor extends Actor {
  def receive = {
    case User(name) =>
      // reply to sender
      sender ! ("Hi " + name)
  }
}
```

Scala API

Reply

```
class SomeActor extends UntypedActor {
  void onReceive(Object msg) {
    if (msg instanceof User) {
      User user = (User) msg;
      // reply to sender
      getSender().tell("Hi " + user.name);
    }
  }
}
```

Java API

...or use ask

```
import akka.patterns.ask

// returns a future
val future = actor ask message

future onSuccess {
  case x => println(x)
}
```

Scala API

...or use ask

```
import static akka.patterns.Patterns.ask;

Future<Object> future = ask(actor, message, timeout);

future.onSuccess(new OnSuccess<Object>() {
  public void onSuccess(String result) {
    System.out.println(result);
  }
});
```

Java API

Future

```
val f = Promise[String]()

f onComplete { ... }
  onSuccess { ... }
  onFailure { ... }
f foreach { ... }
f map { ... }
f flatMap { ... }
f filter { ... }
f zip otherF
f fallbackTo otherF

Await.result(f, 5 seconds)
```

Scala API

Future

```
firstCompletedOf
                  fold
                                       reduce
find
                traverse
sequence
```

Combinators for collections of Futures

become

```
context become {
  // new body
  case NewMessage =>
    ...
}
```

Scala API

become

```
context.become(new Procedure[Object]() {
  void apply(Object msg) {
    // new body
    if (msg instanceof NewMessage) {
      NewMessage newMsg = (NewMessage)msg;
      ...
    }
  }
});
```

Java API

unbecome

```
context.unbecome()
```

Routers

```
val router =
 system.actorOf(
   Props[SomeActor].withRouter(
    RoundRobinRouter(nrOfInstances = 5)))
```

Scala API

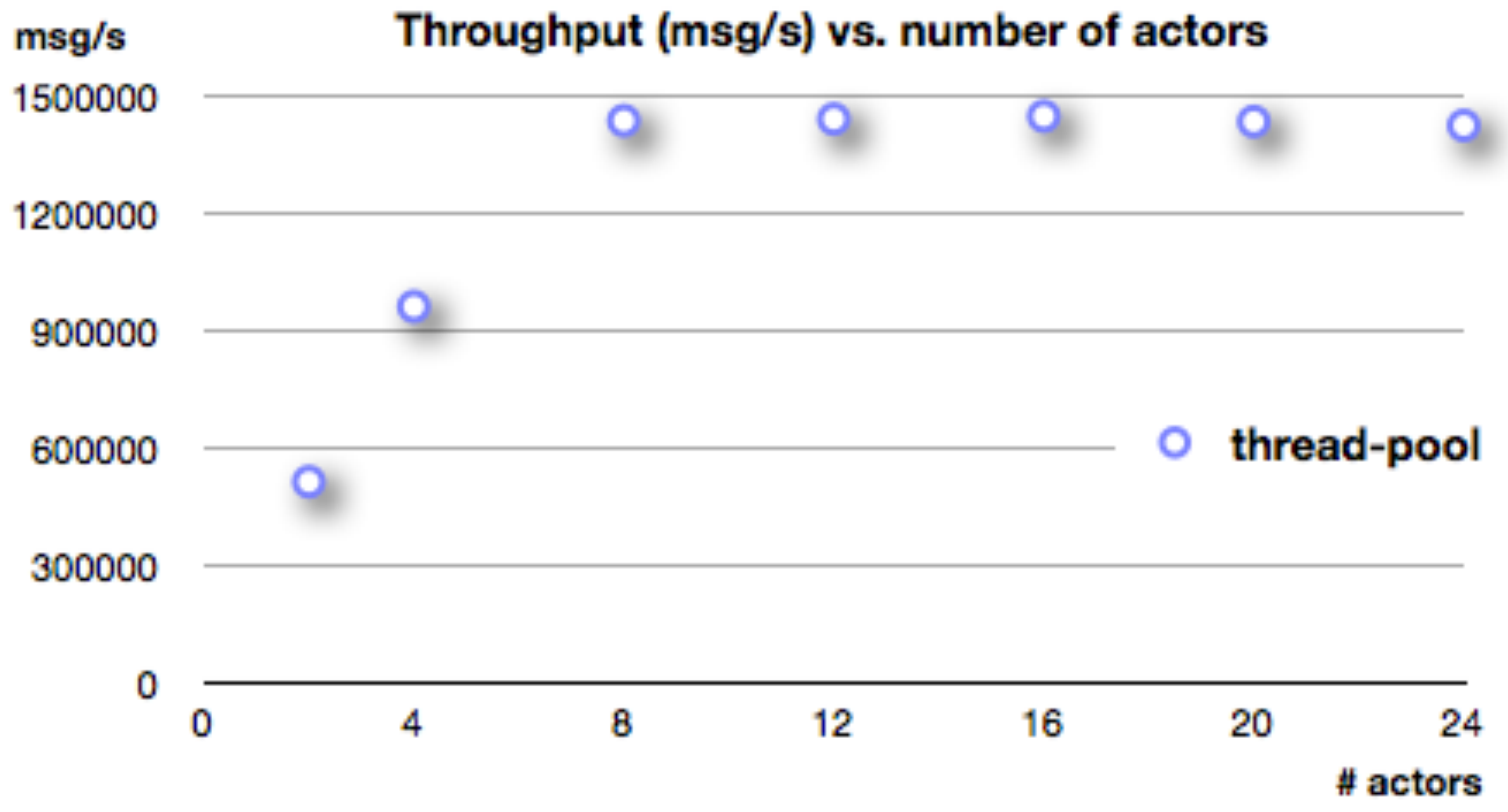
Router + Resizer

```
val resizer =
  DefaultResizer(lowerBound = 2,
                 upperBound = 15)

val router =
  system.actorOf(
    Props[ExampleActor1].withRouter(
      RoundRobinRouter(resizer = Some(resizer))
    )
  )
```

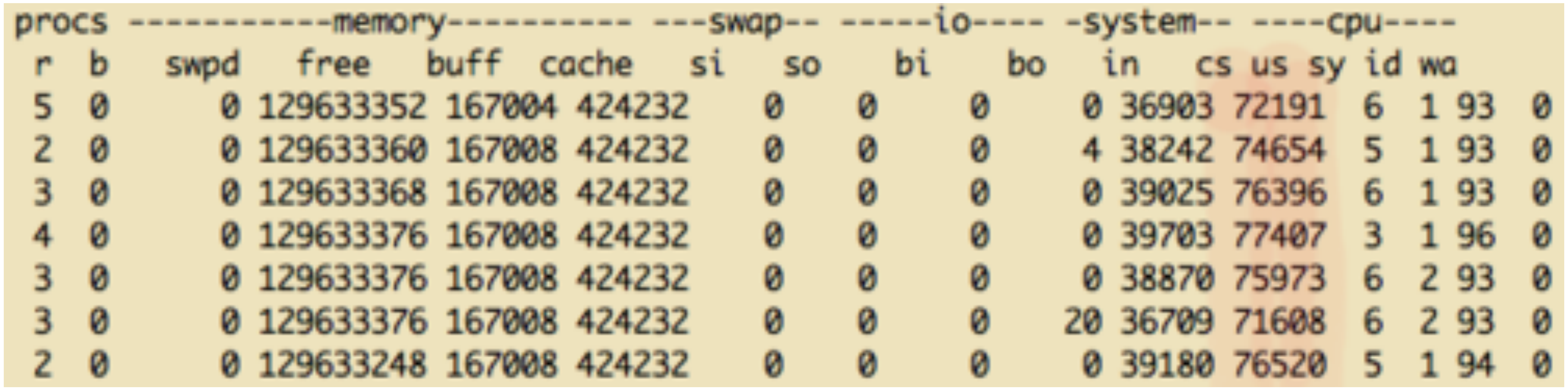
Scala API

Scale **up?**

ThreadPoolExecutor

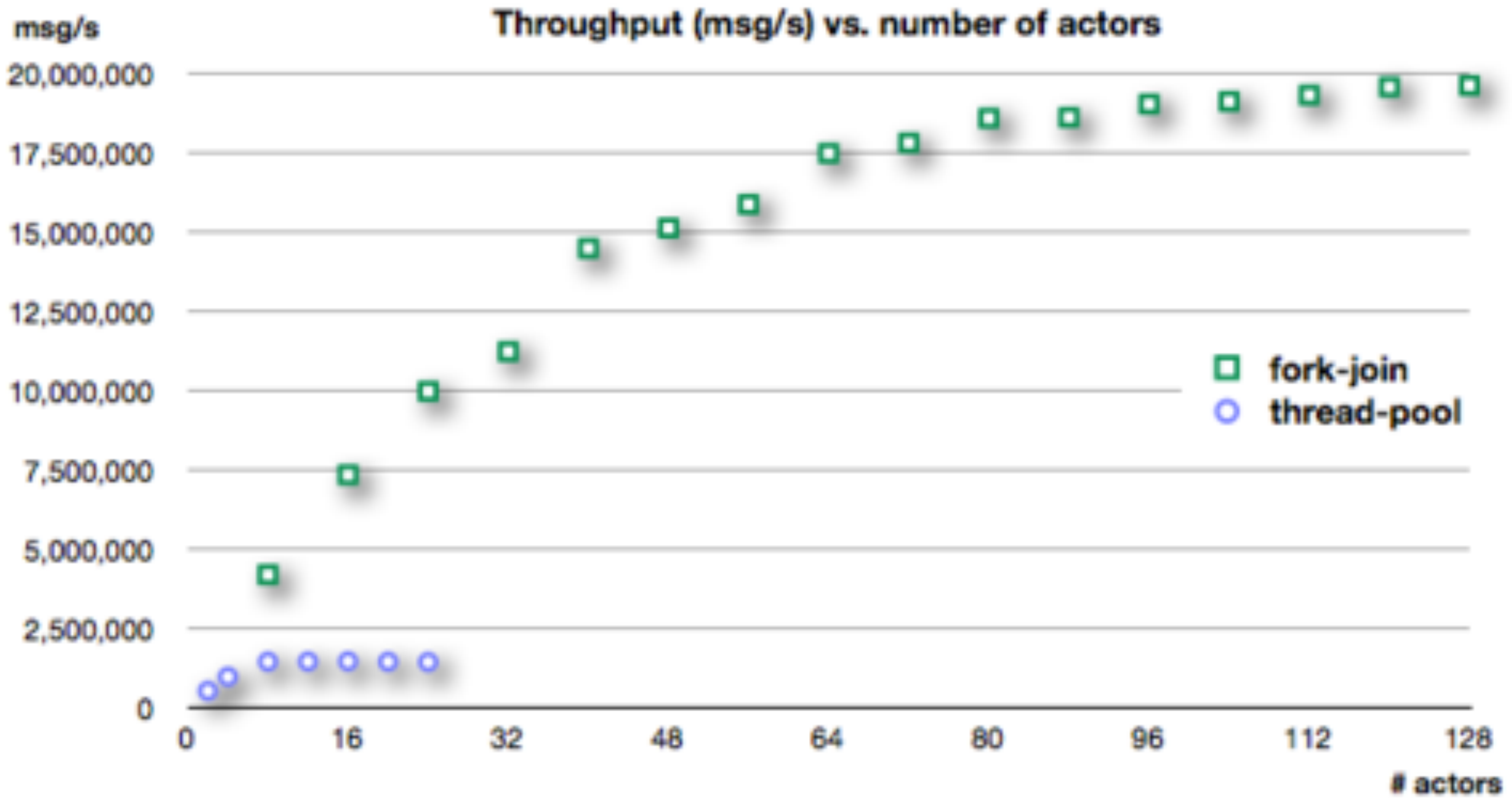


ThreadPoolExecutor

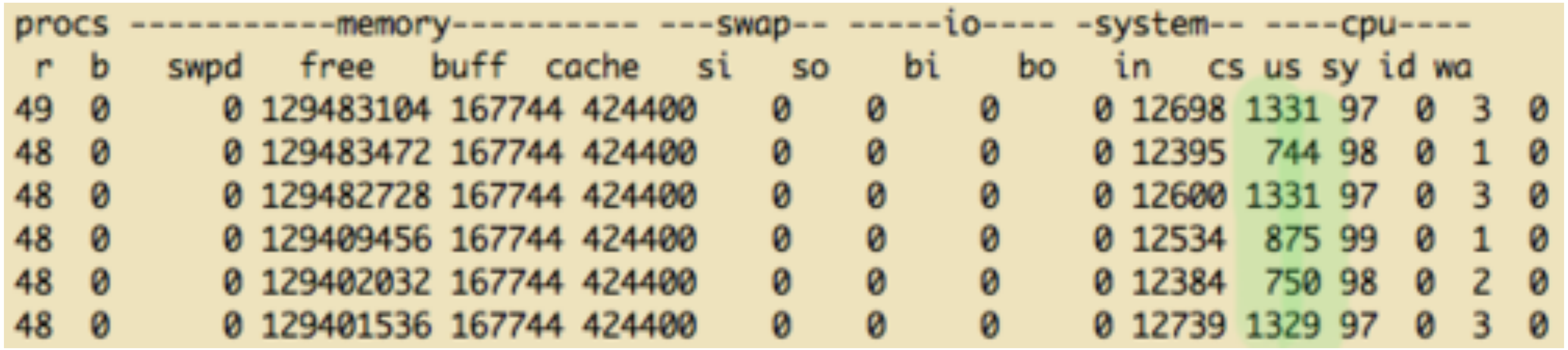


```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 5  0      0 129633352 167004 424232    0    0     0     0 36903 72191  6  1 93  0
 2  0      0 129633360 167008 424232    0    0     0     4 38242 74654  5  1 93  0
 3  0      0 129633368 167008 424232    0    0     0     0 39025 76396  6  1 93  0
 4  0      0 129633376 167008 424232    0    0     0     0 39703 77407  3  1 96  0
 3  0      0 129633376 167008 424232    0    0     0     0 38870 75973  6  2 93  0
 3  0      0 129633376 167008 424232    0    0     0    20 36709 71608  6  2 93  0
 2  0      0 129633248 167008 424232    0    0     0     0 39180 76520  5  1 94  0
```

new ForkJoinPool



new ForkJoinPool



```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
49  0      0 129483104 167744 424400    0    0     0     0 12698 1331 97  0  3  0
48  0      0 129483472 167744 424400    0    0     0     0 12395  744 98  0  1  0
48  0      0 129482728 167744 424400    0    0     0     0 12600 1331 97  0  3  0
48  0      0 129409456 167744 424400    0    0     0     0 12534  875 99  0  1  0
48  0      0 129402032 167744 424400    0    0     0     0 12384  750 98  0  2  0
48  0      0 129401536 167744 424400    0    0     0     0 12739 1329 97  0  3  0
```

Scale **out**

New Remote Actors

Name

```scala
val actor = system.actorOf(Props[MyActor], “my-service”)
```

Bind the actor to a name

Scala API

Name

```
ActorRef actor = system.actorOf(
  new Props(MyActor.class), “my-service”)
```

Bind the actor to a name

Java API

Deployment

Actor name is virtual and decoupled from how it is deployed

Deployment

If no deployment configuration exists then actor is deployed as local

Deployment

The same system can be configured as distributed without code change (even change at runtime)

Deployment

Write as local but deploy as distributed in the cloud without code change

Deployment

Allows runtime to dynamically and adaptively change topology

Deployment configuration

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

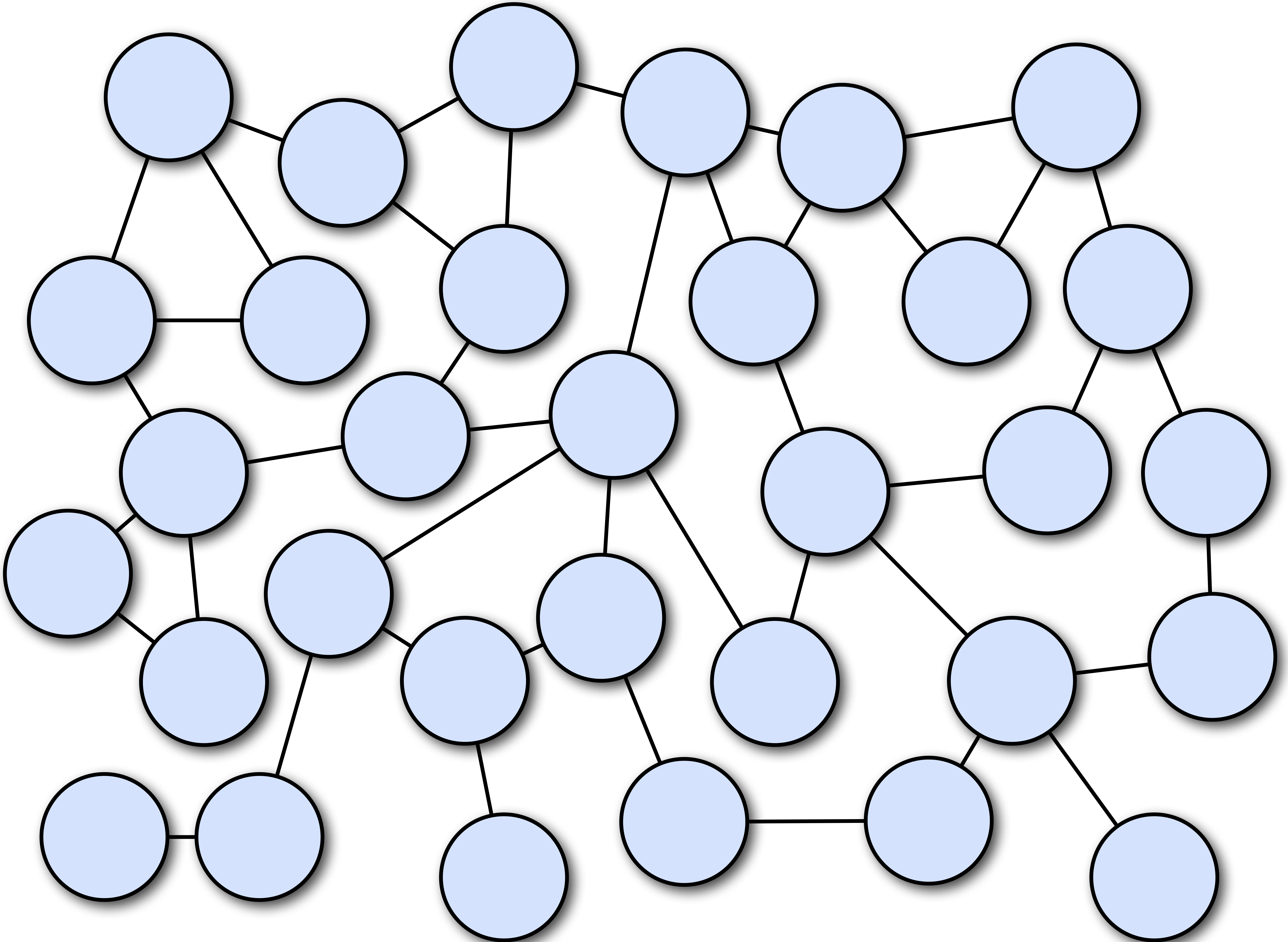
Let it crash

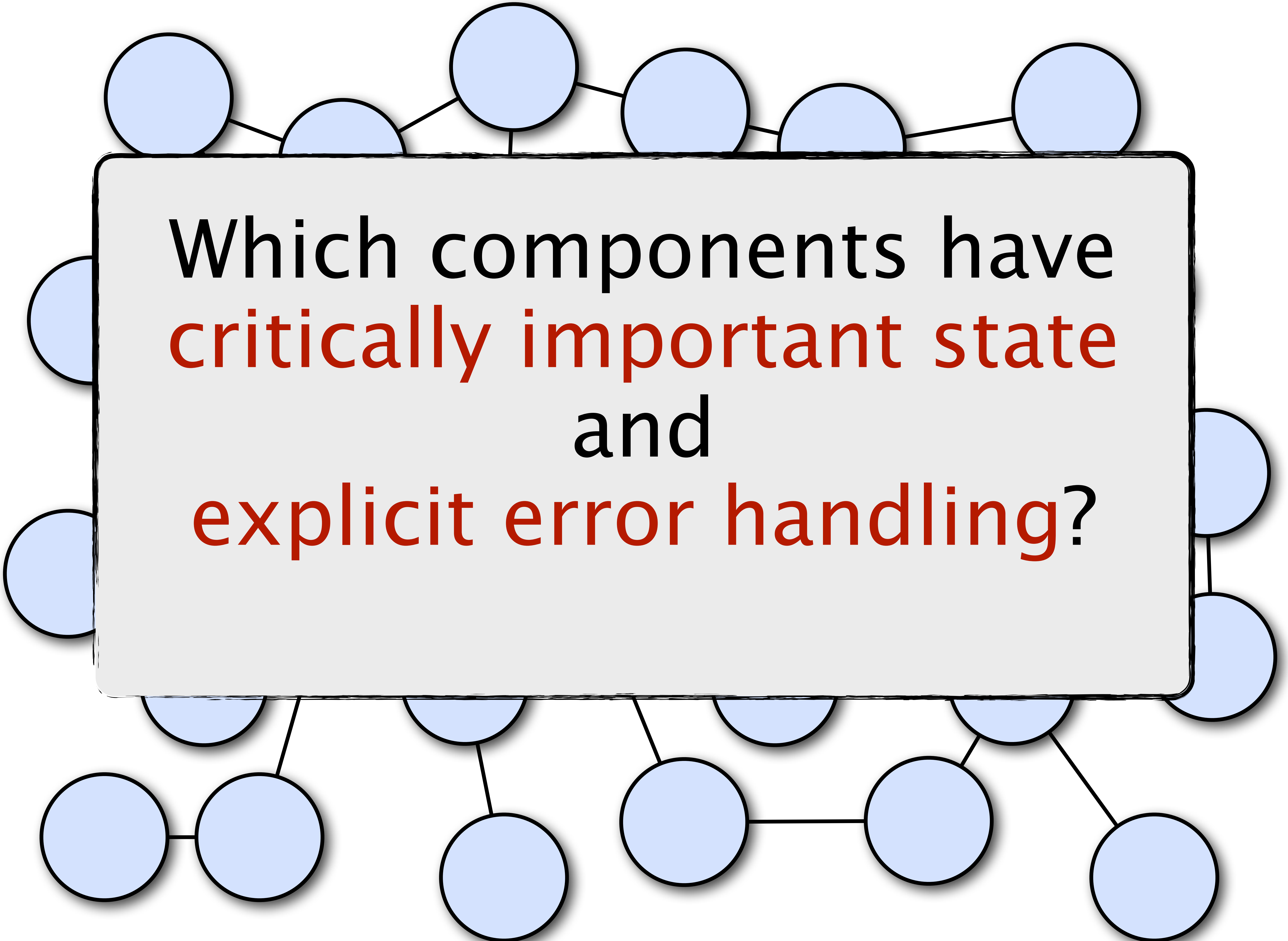
fault-tolerance

The Erlang model

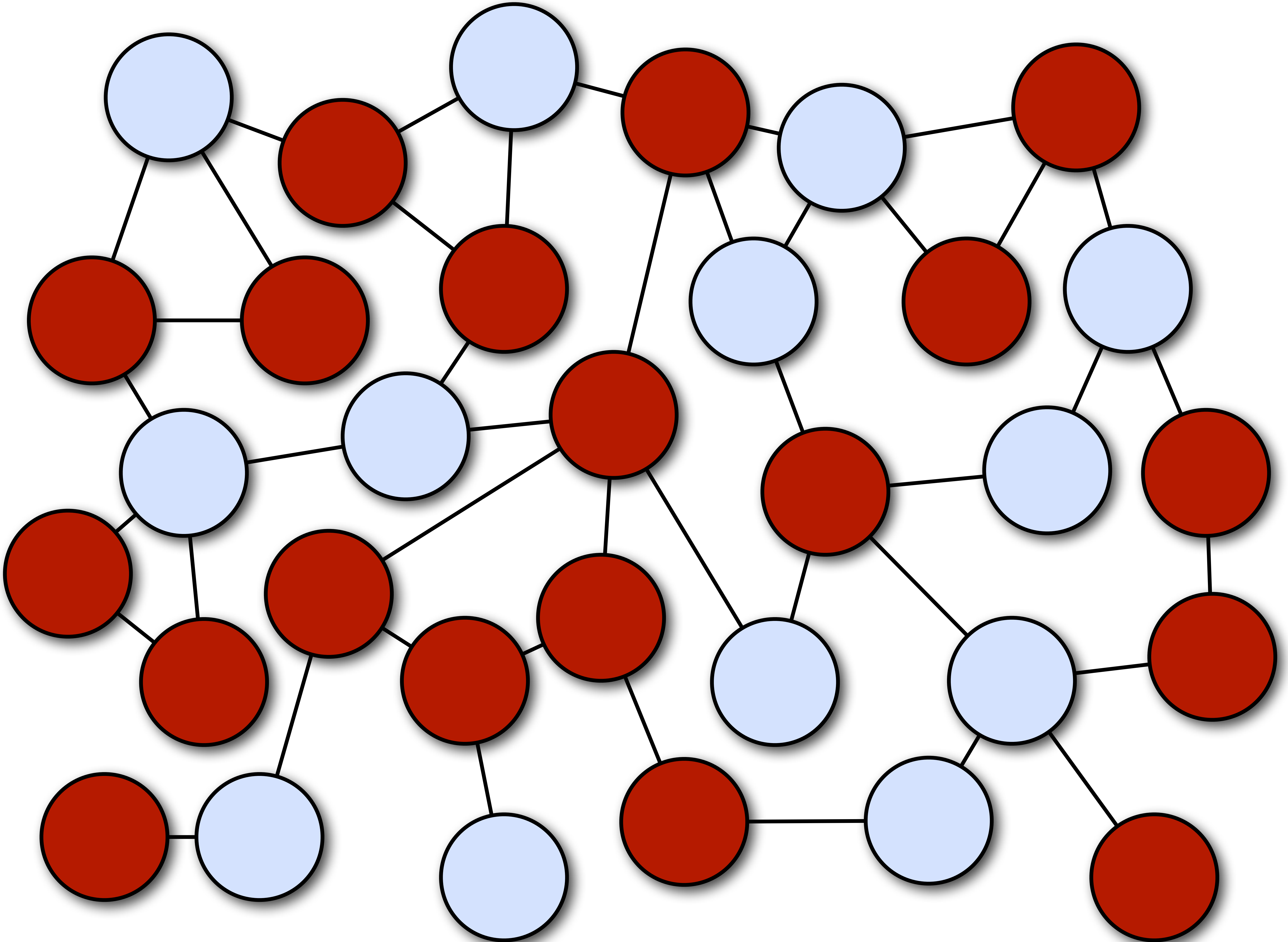
9 nines

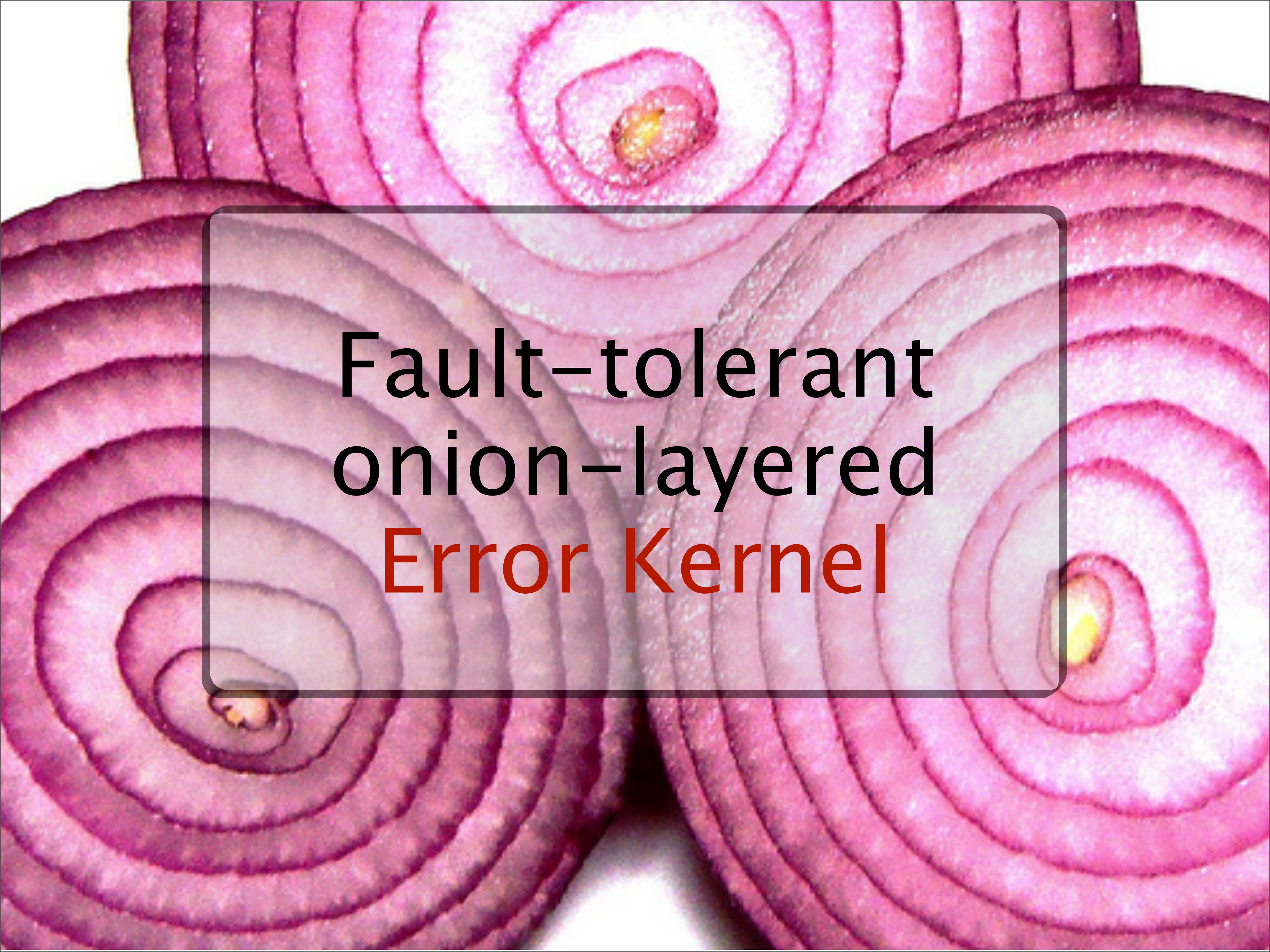
...let’s take a standard OO application



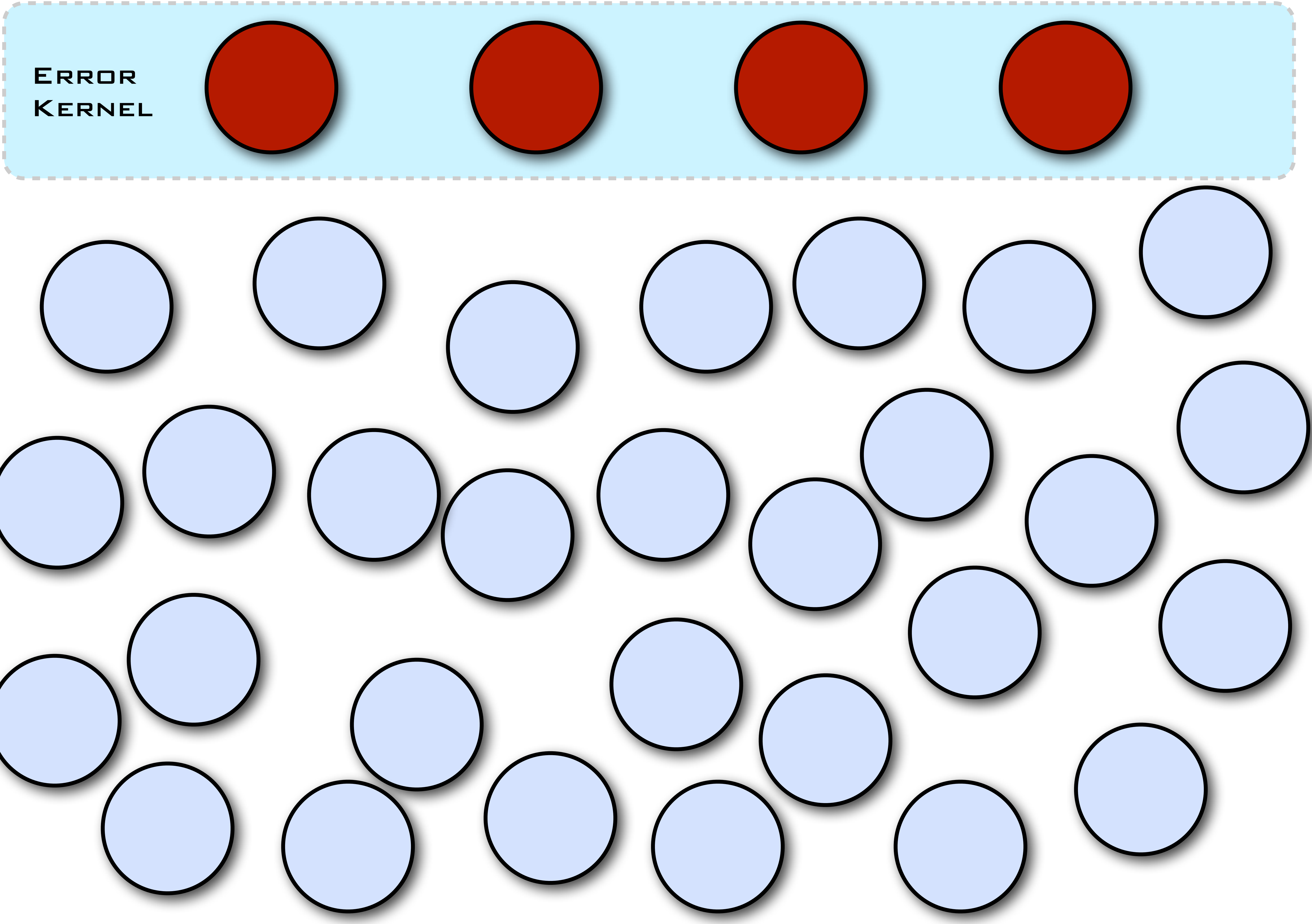


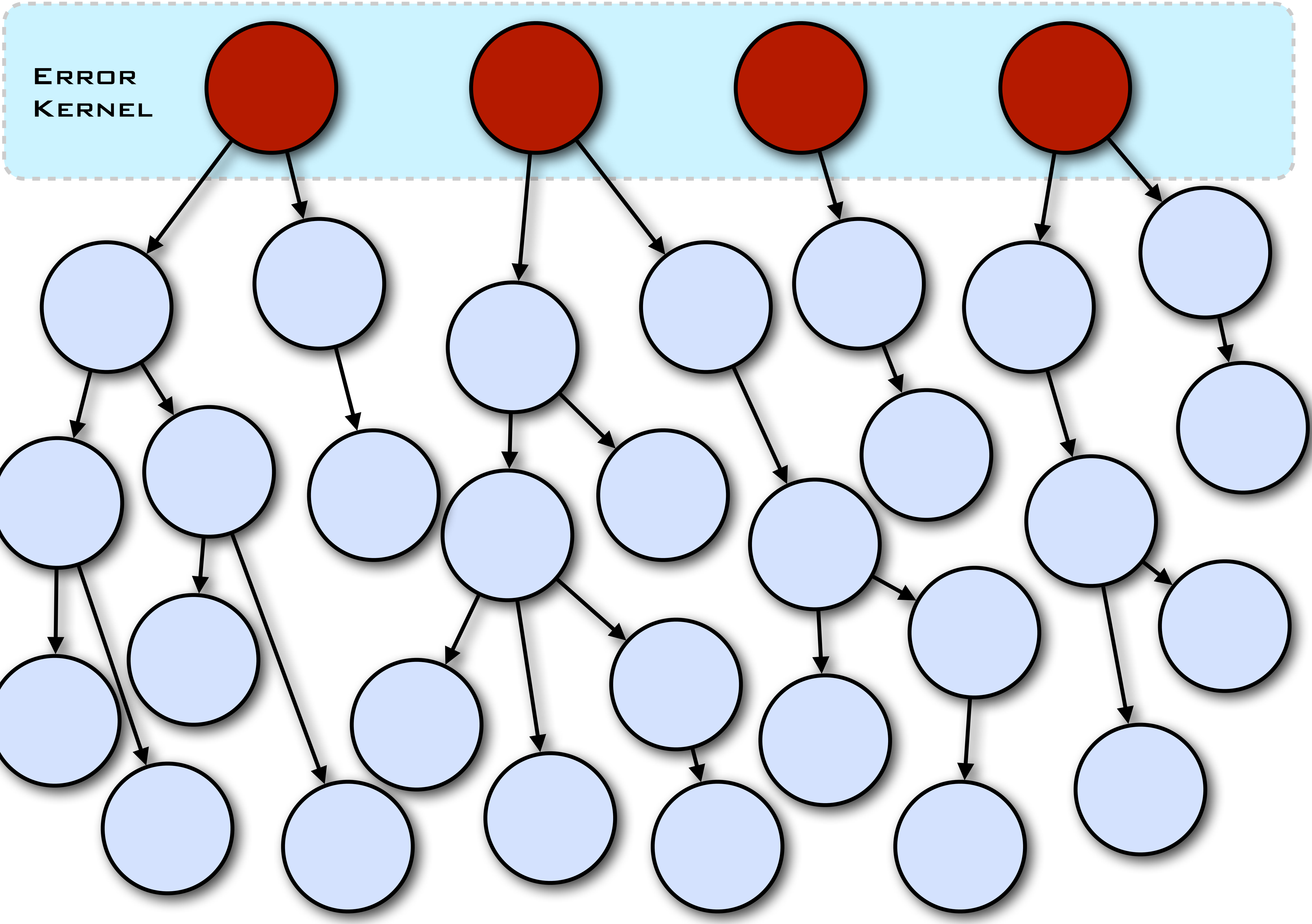
Which components have critically important state and explicit error handling?

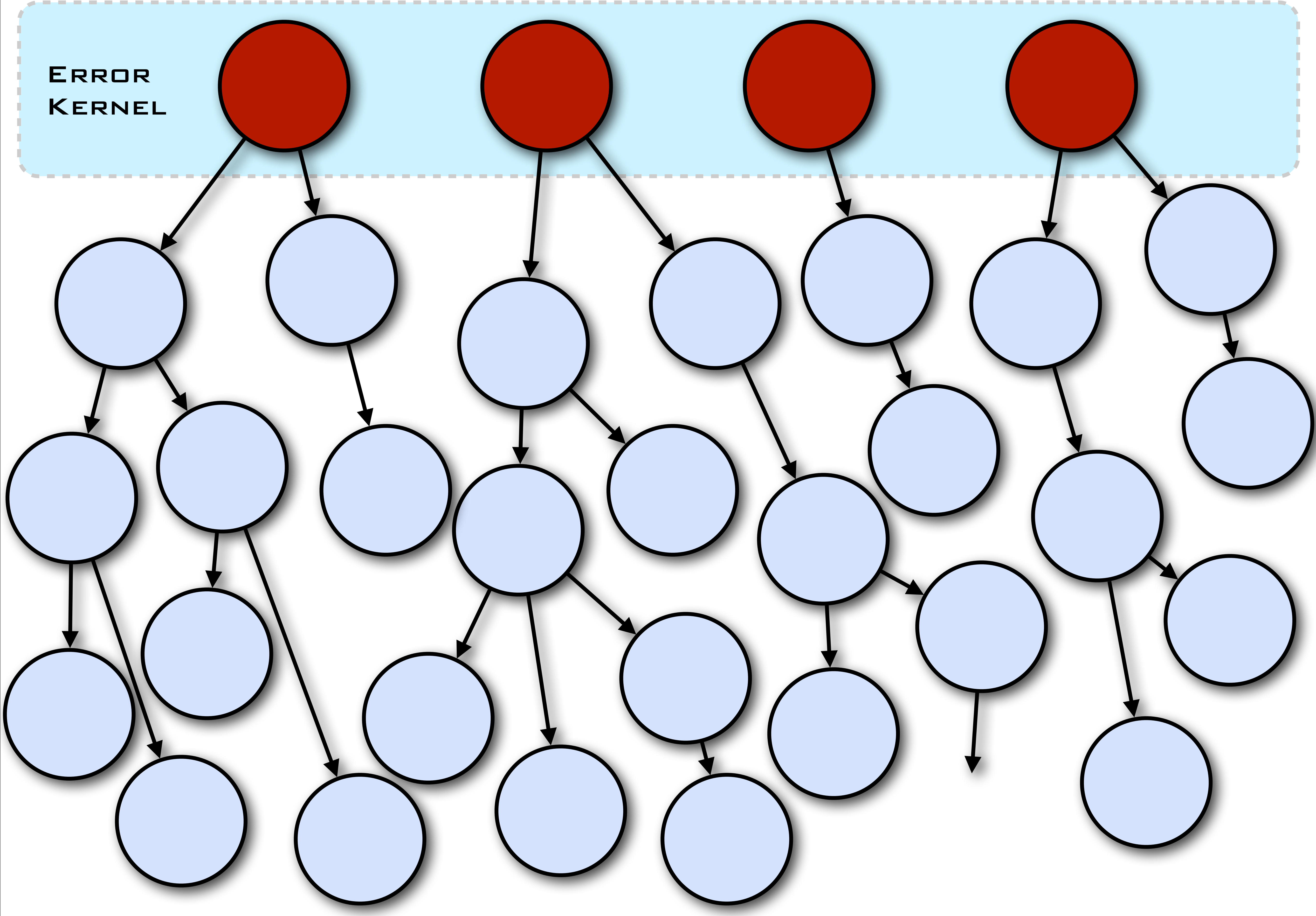


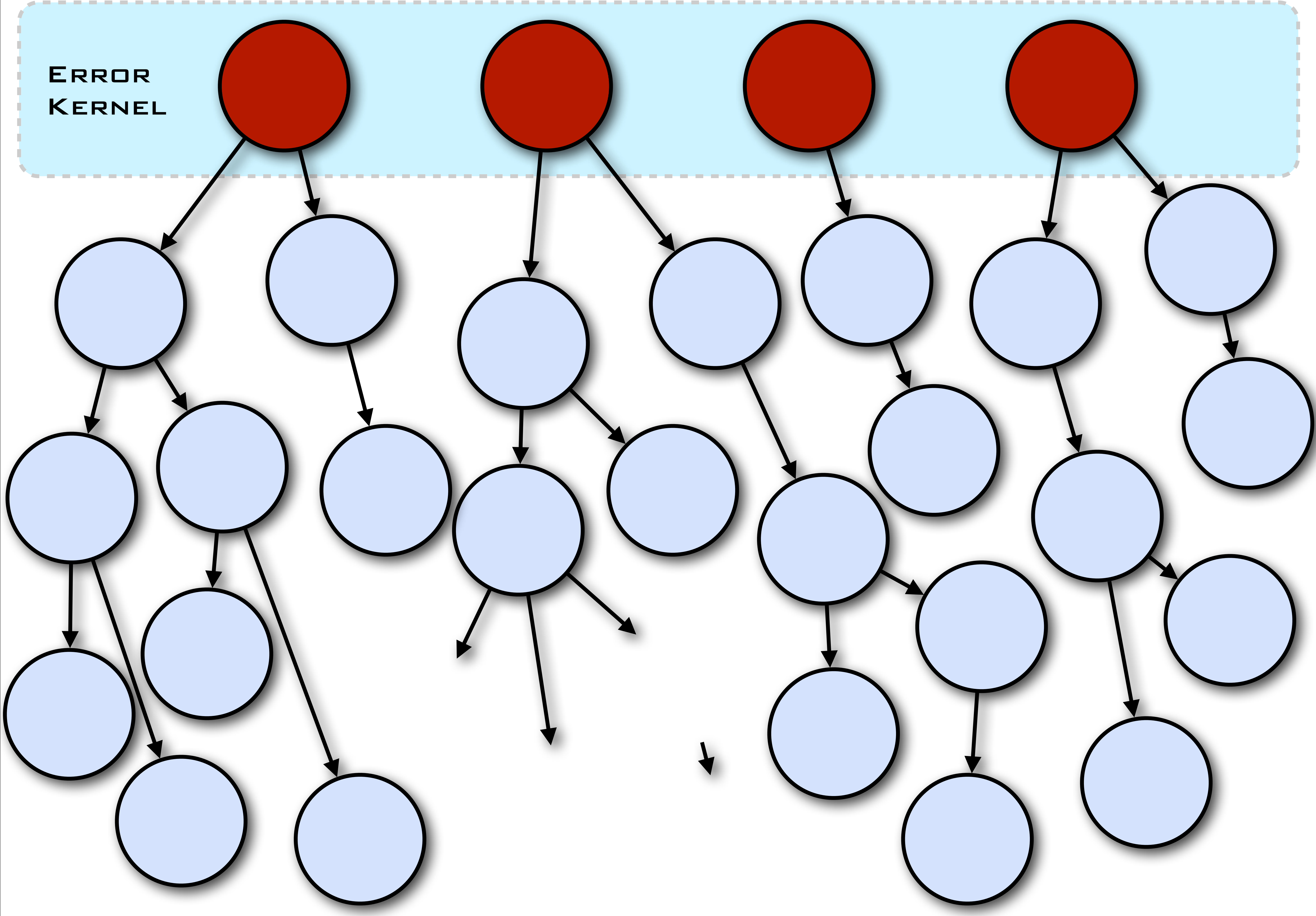


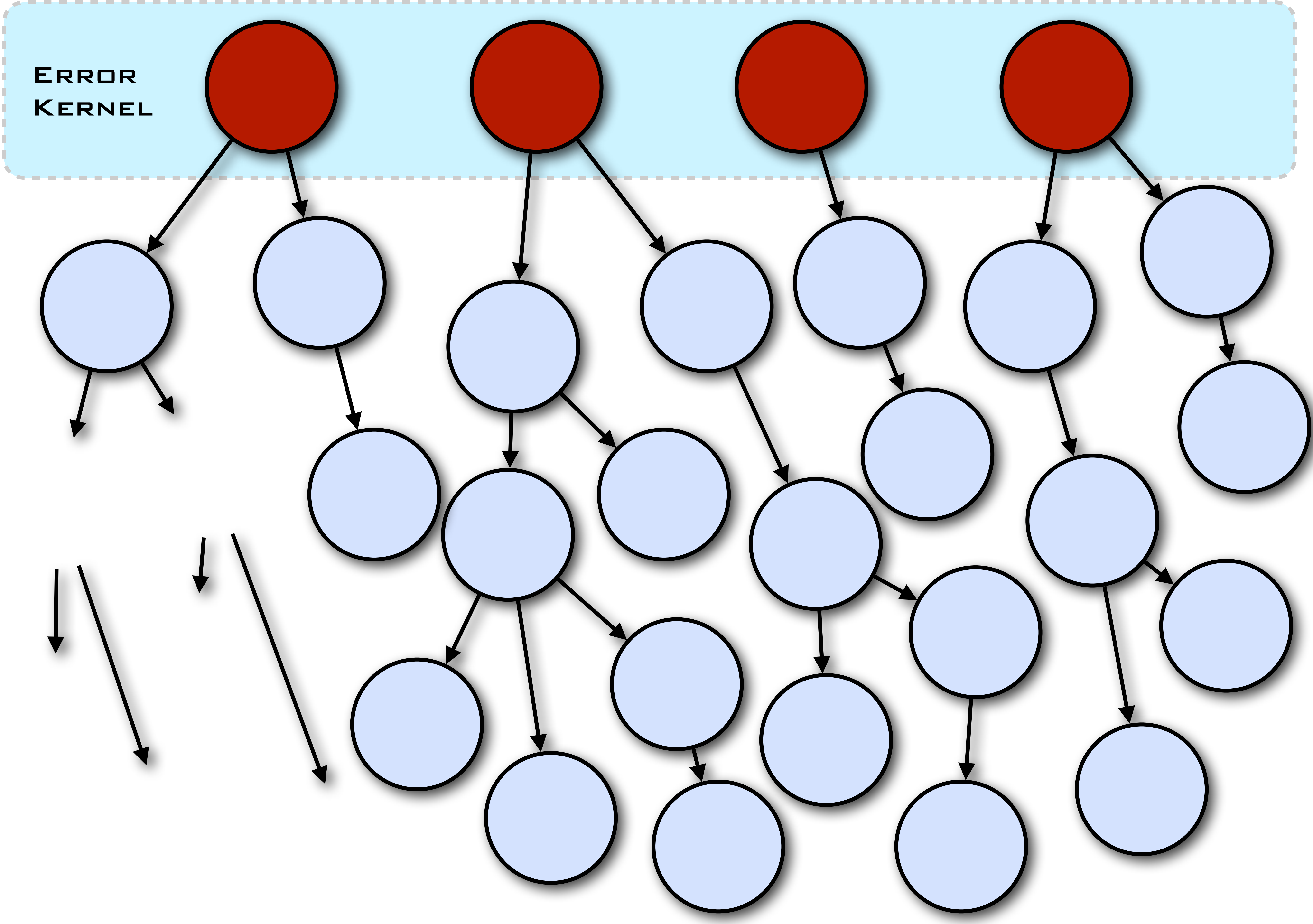
Fault-tolerant onion-layered Error Kernel

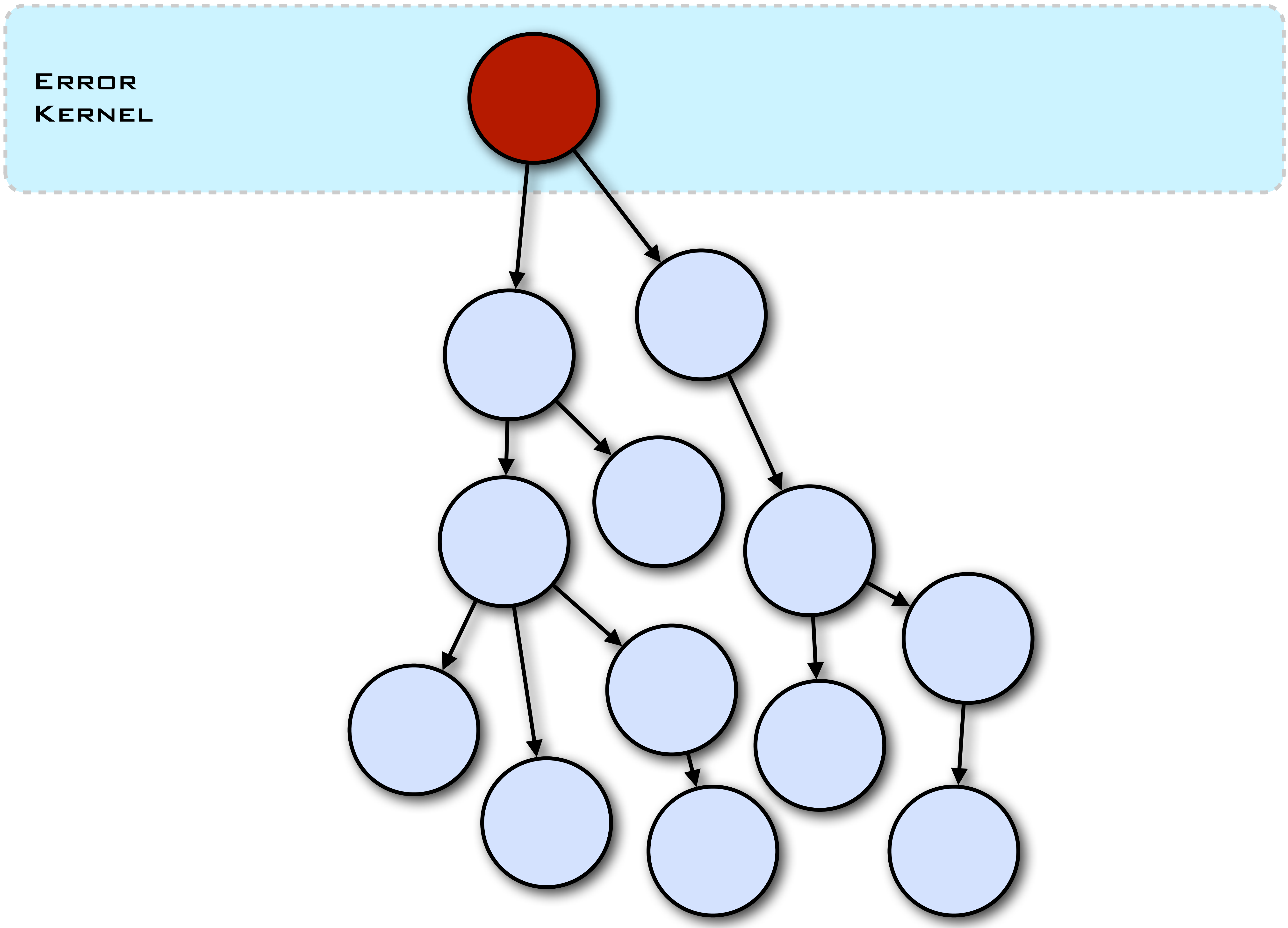

Error
Kernel

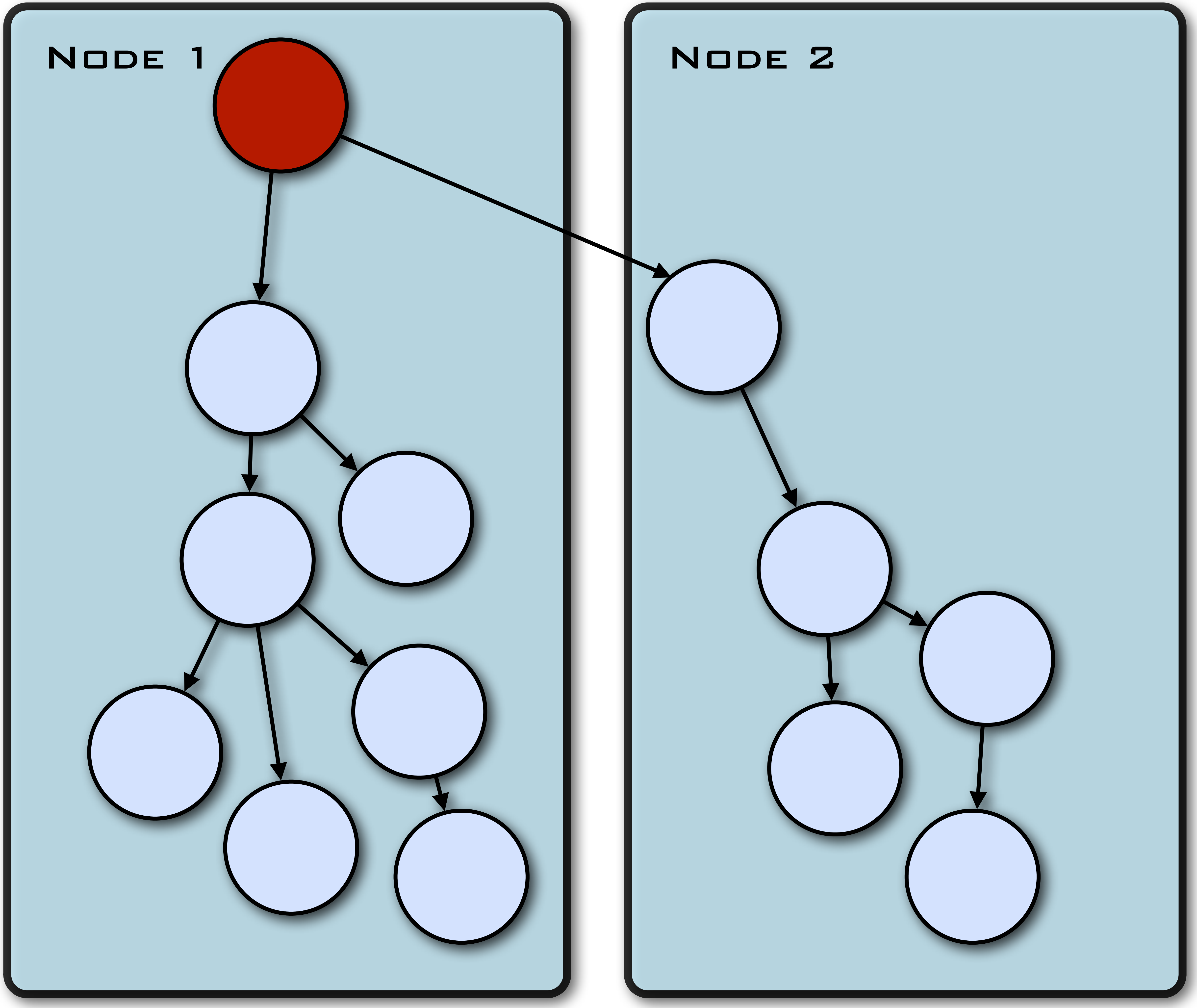

Error Kernel


Error
Kernel


Error
Kernel


Error
Kernel


Error
Kernel


Node 1
Node 2

Parental automatic supervision

```
// from within an actor
val child = context.actorOf(Props[MyActor], “my-actor”)
```

transparent and automatic fault handling by design

Scala API

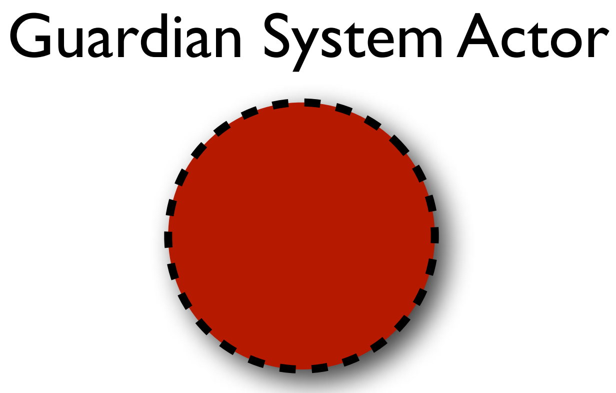
Parental automatic supervision

```
// from within an actor
ActorRef child = getContext().actorOf(
  new Props(MyActor.class), “my–actor”);
```

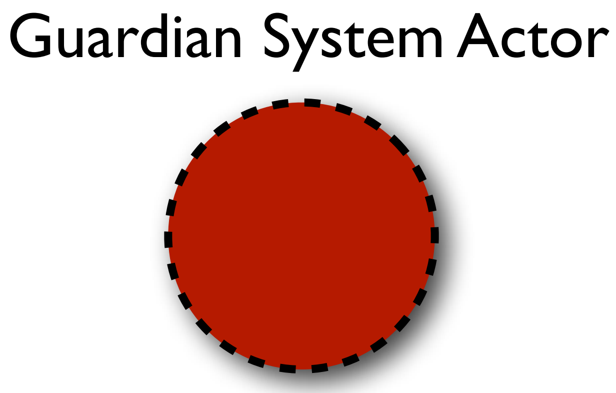
transparent and automatic fault handling by design

Java API

Parental automatic supervision

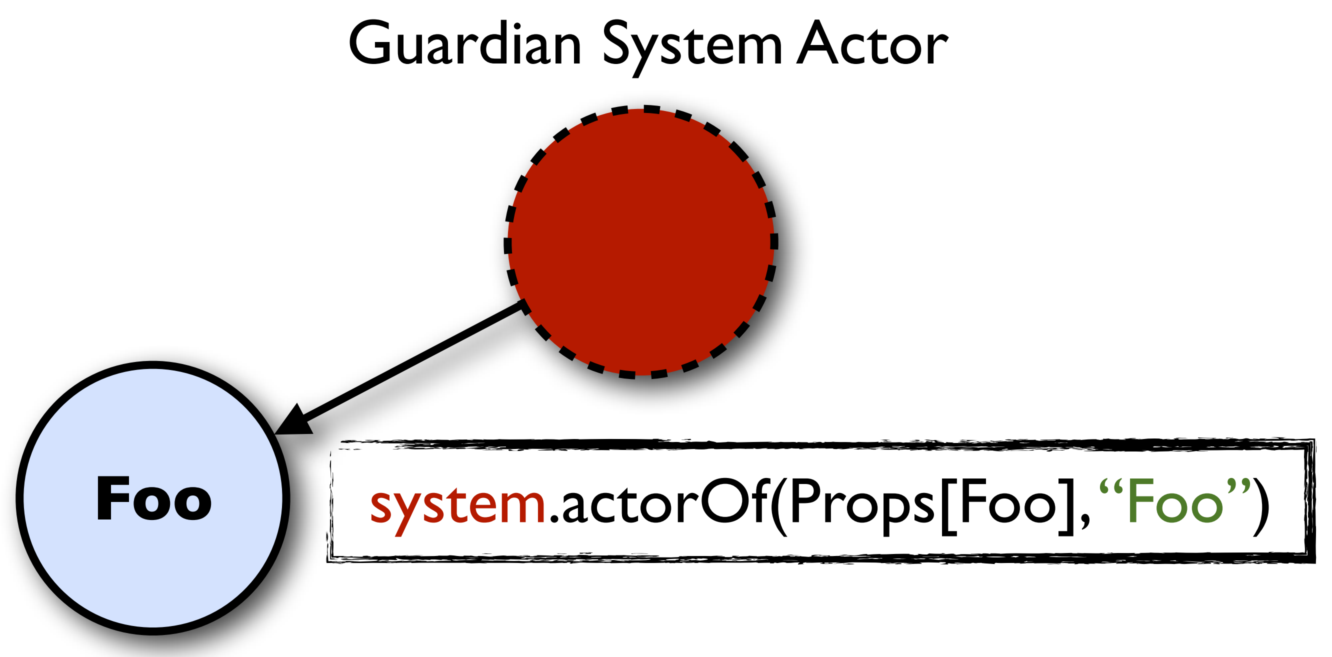


Parental automatic supervision



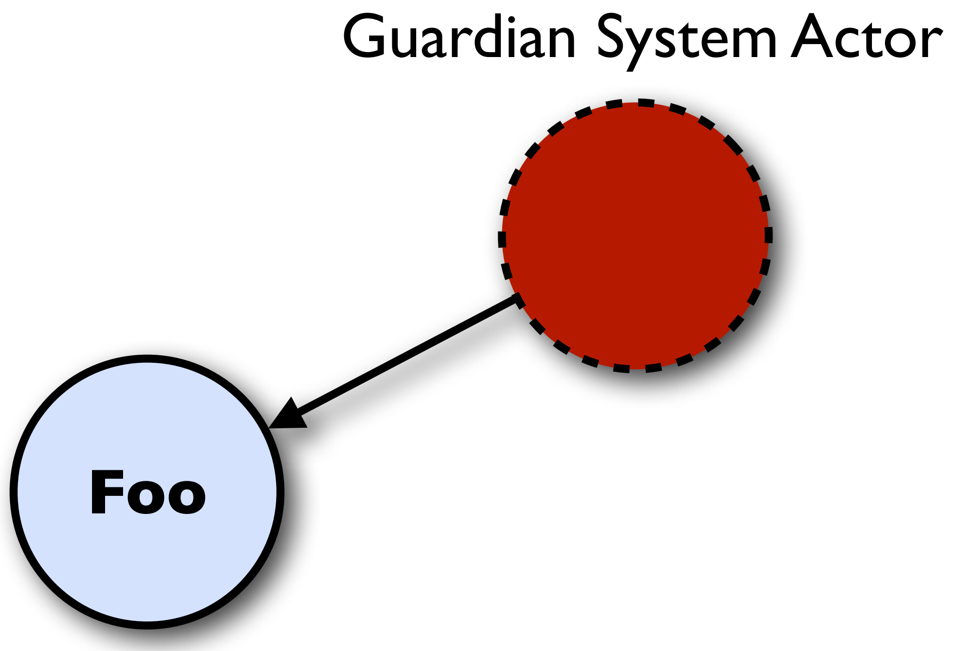
```
system.actorOf(Props[Foo], "Foo")
```

Parental automatic supervision



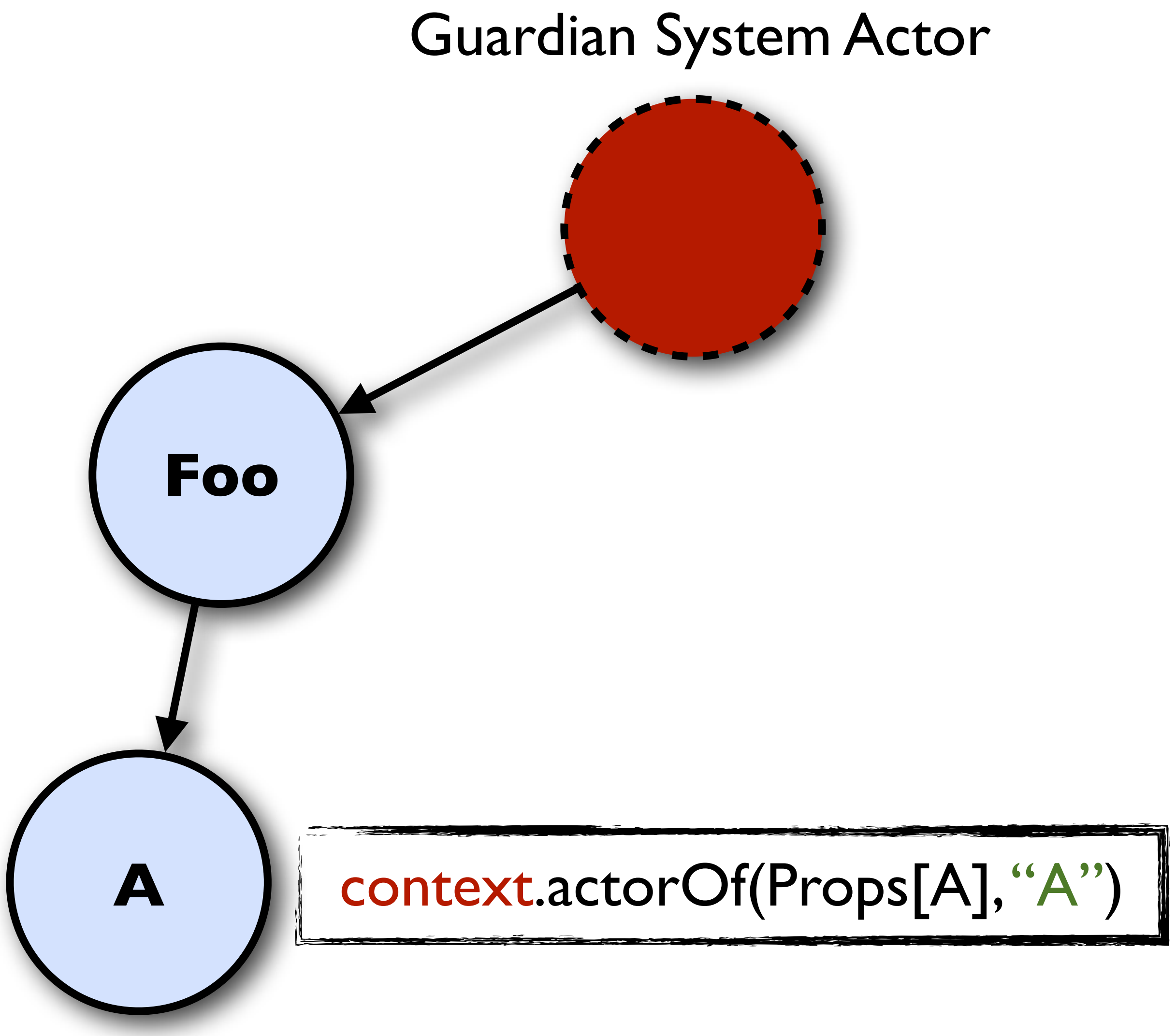
```
system.actorOf(Props[Foo],“Foo”)
```

Parental automatic supervision



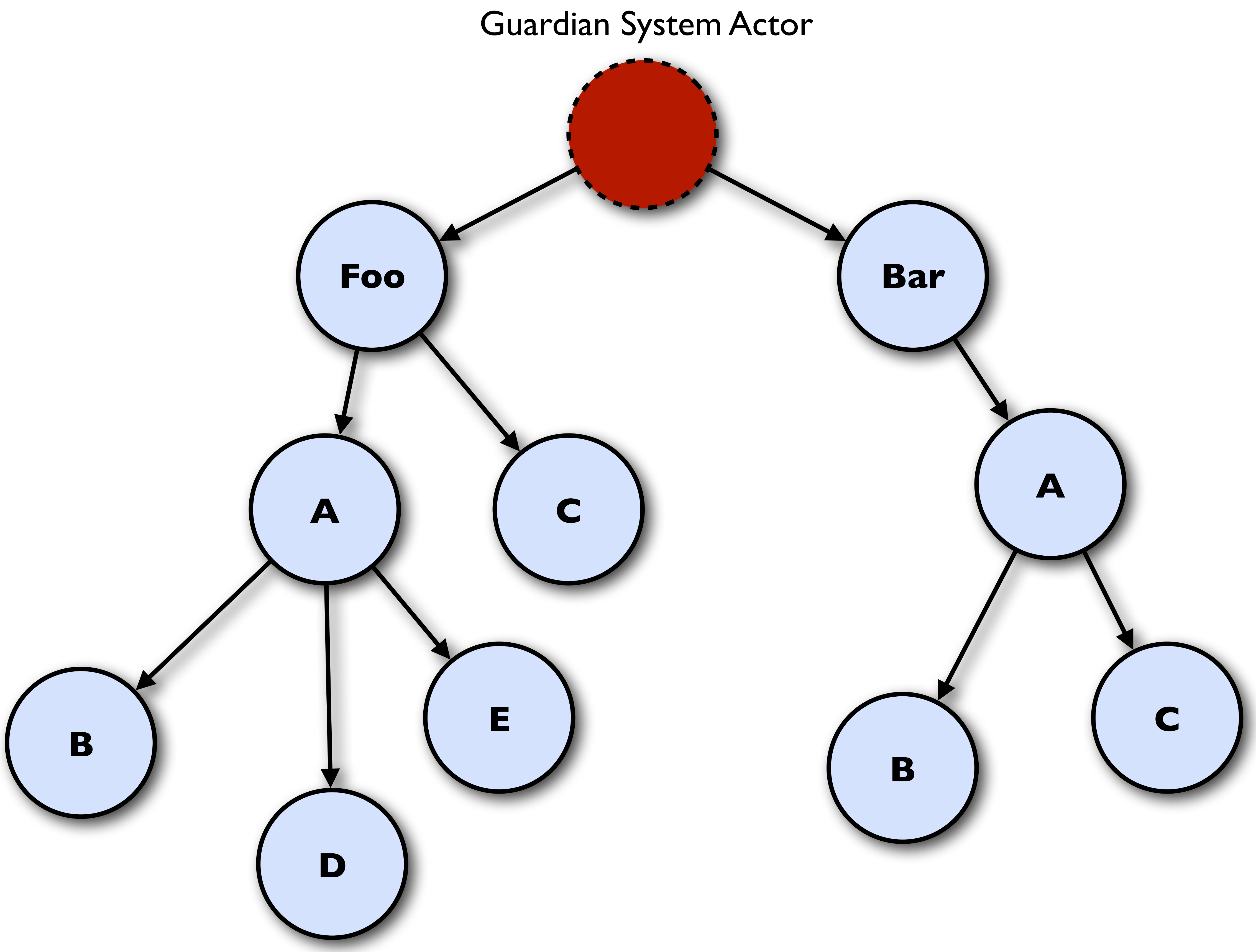
```
context.actorOf(Props[A], "A")
```

Parental automatic supervision

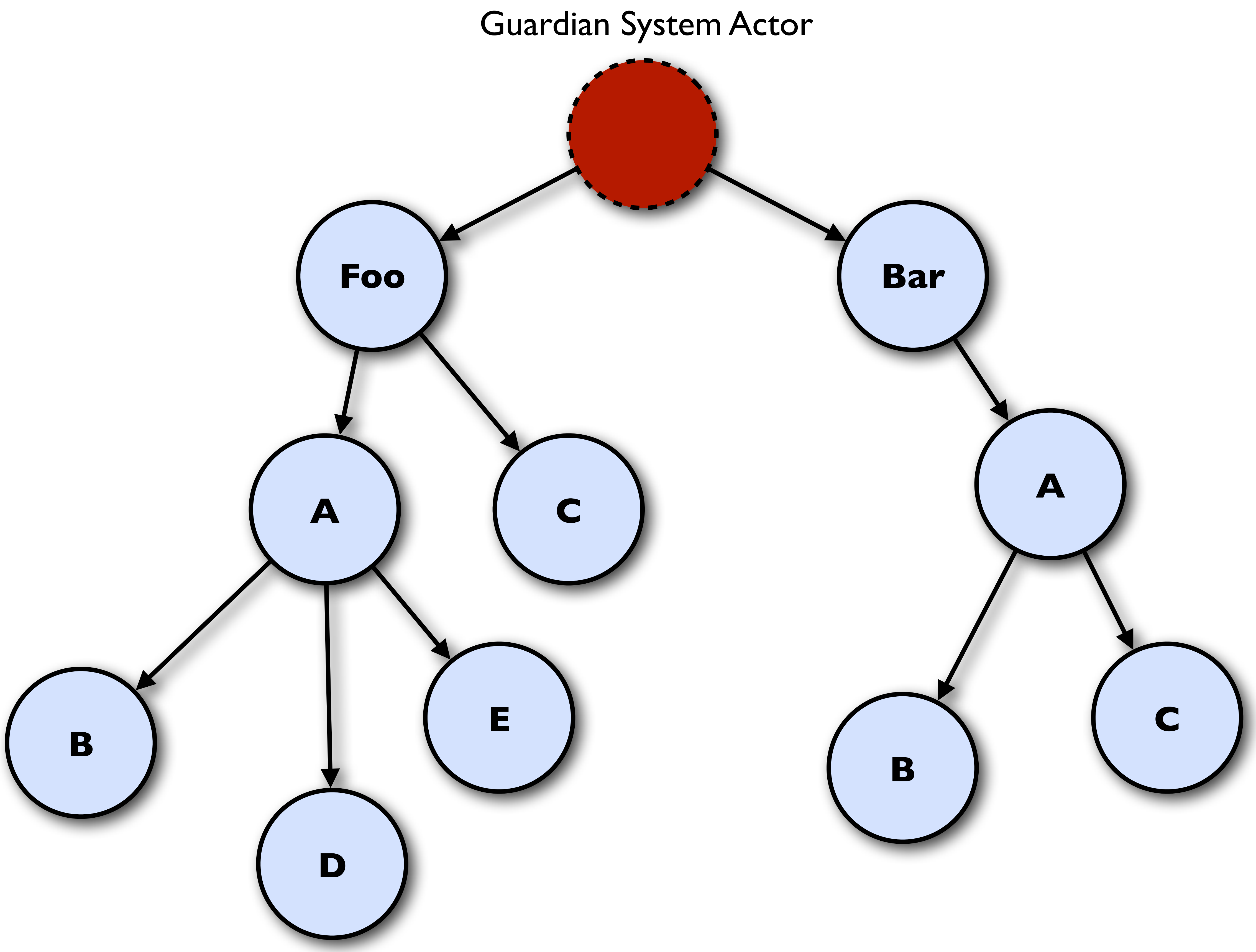


```
context.actorOf(Props[A], "A")
```

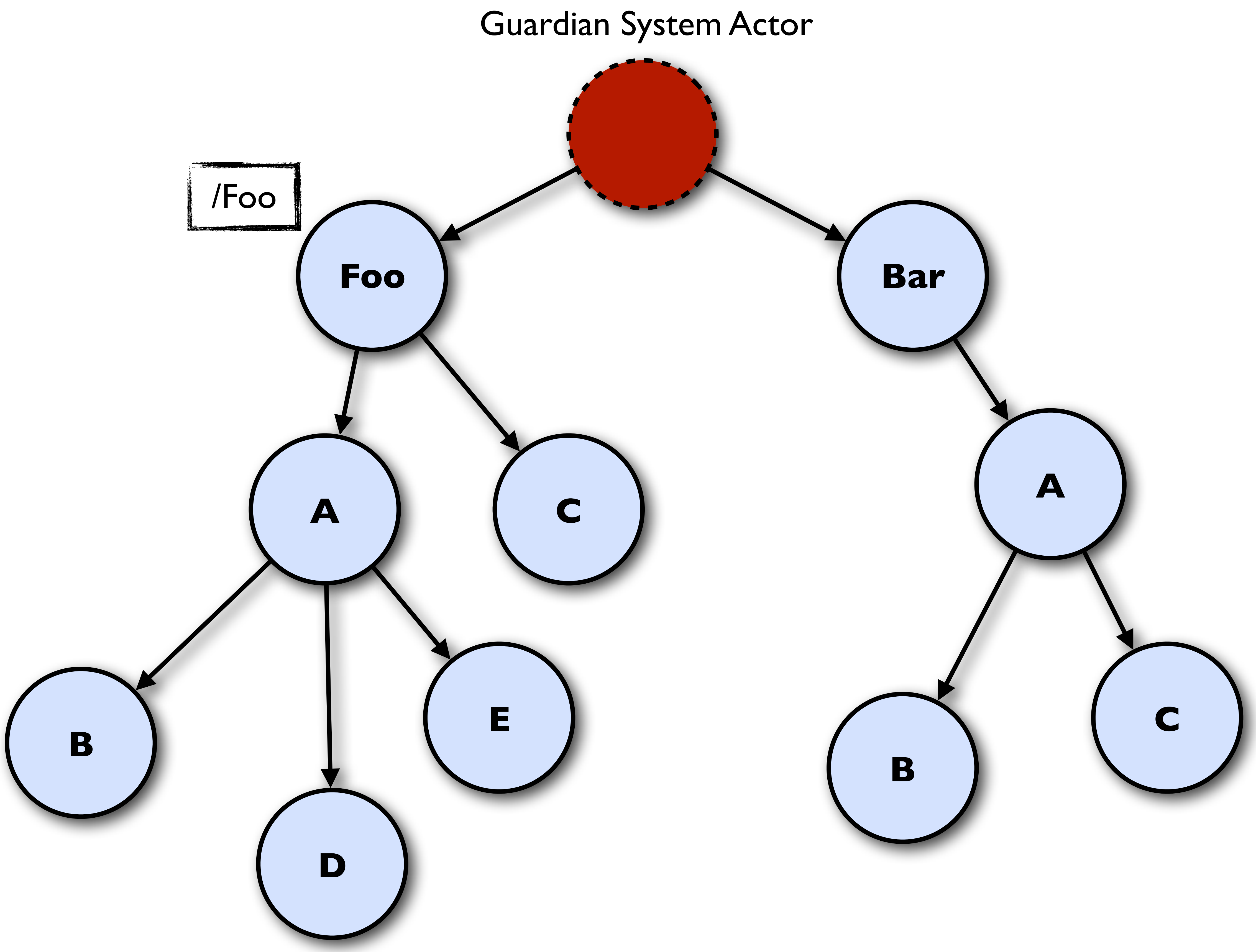
Parental automatic supervision



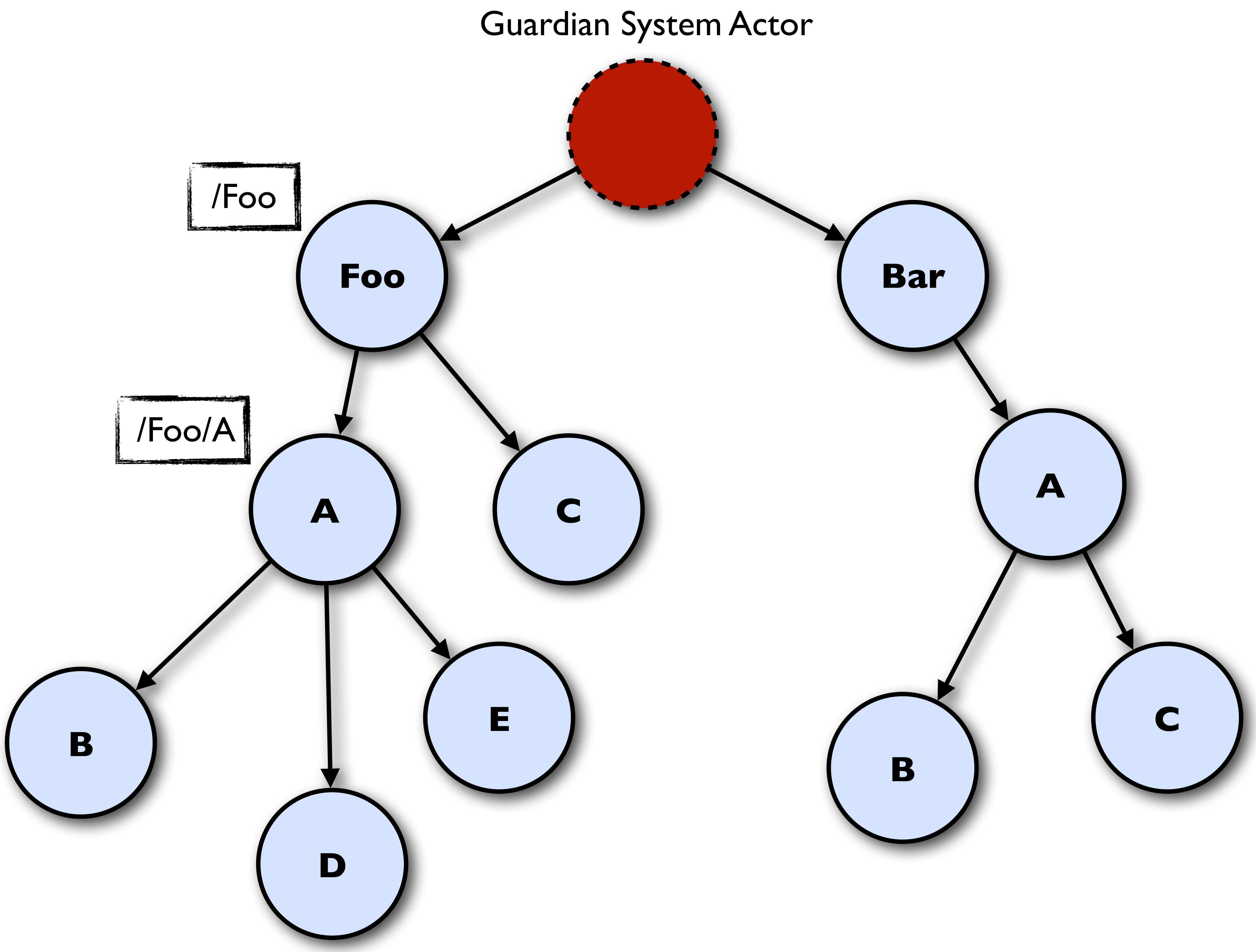
Name resolution



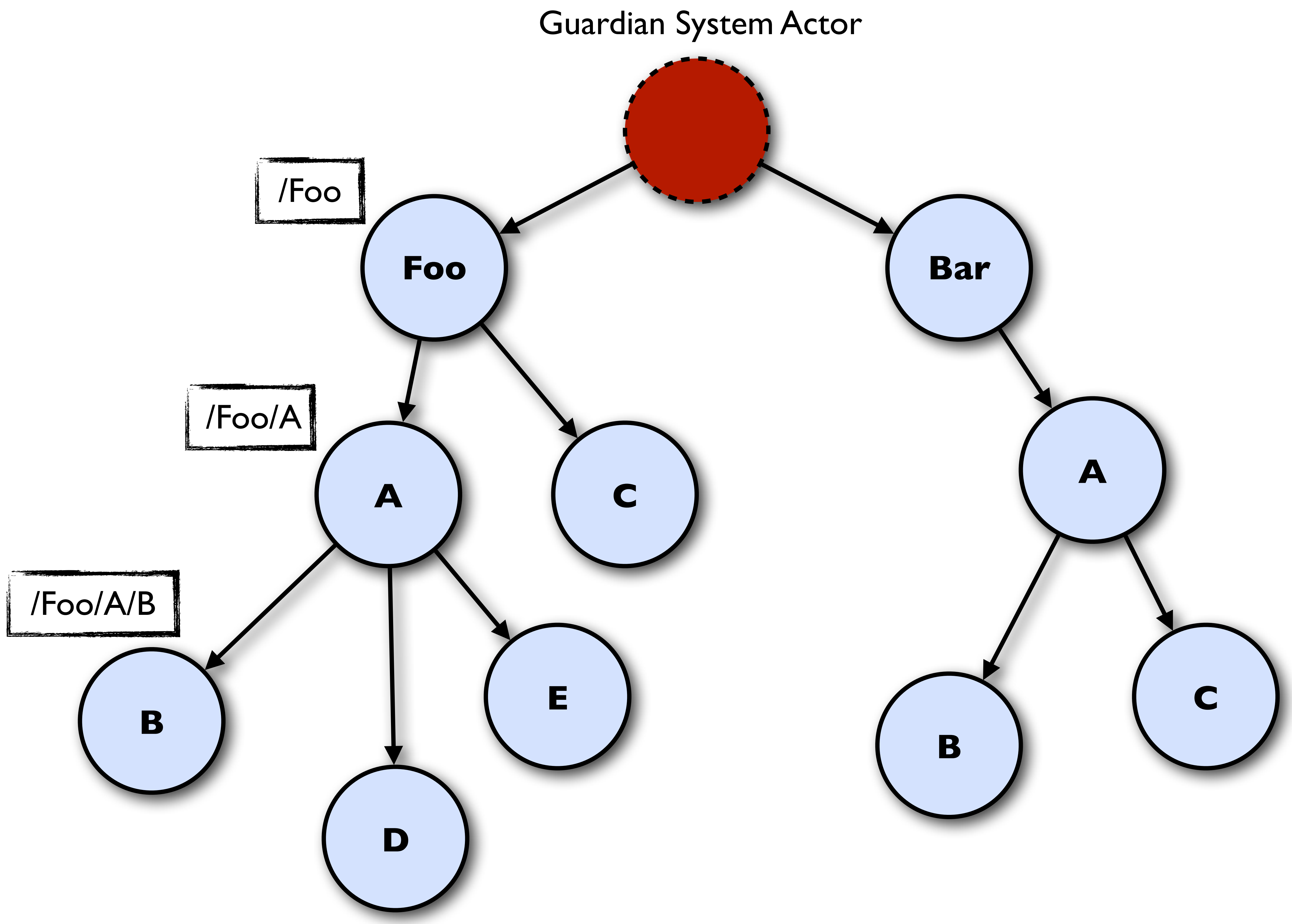
Name resolution



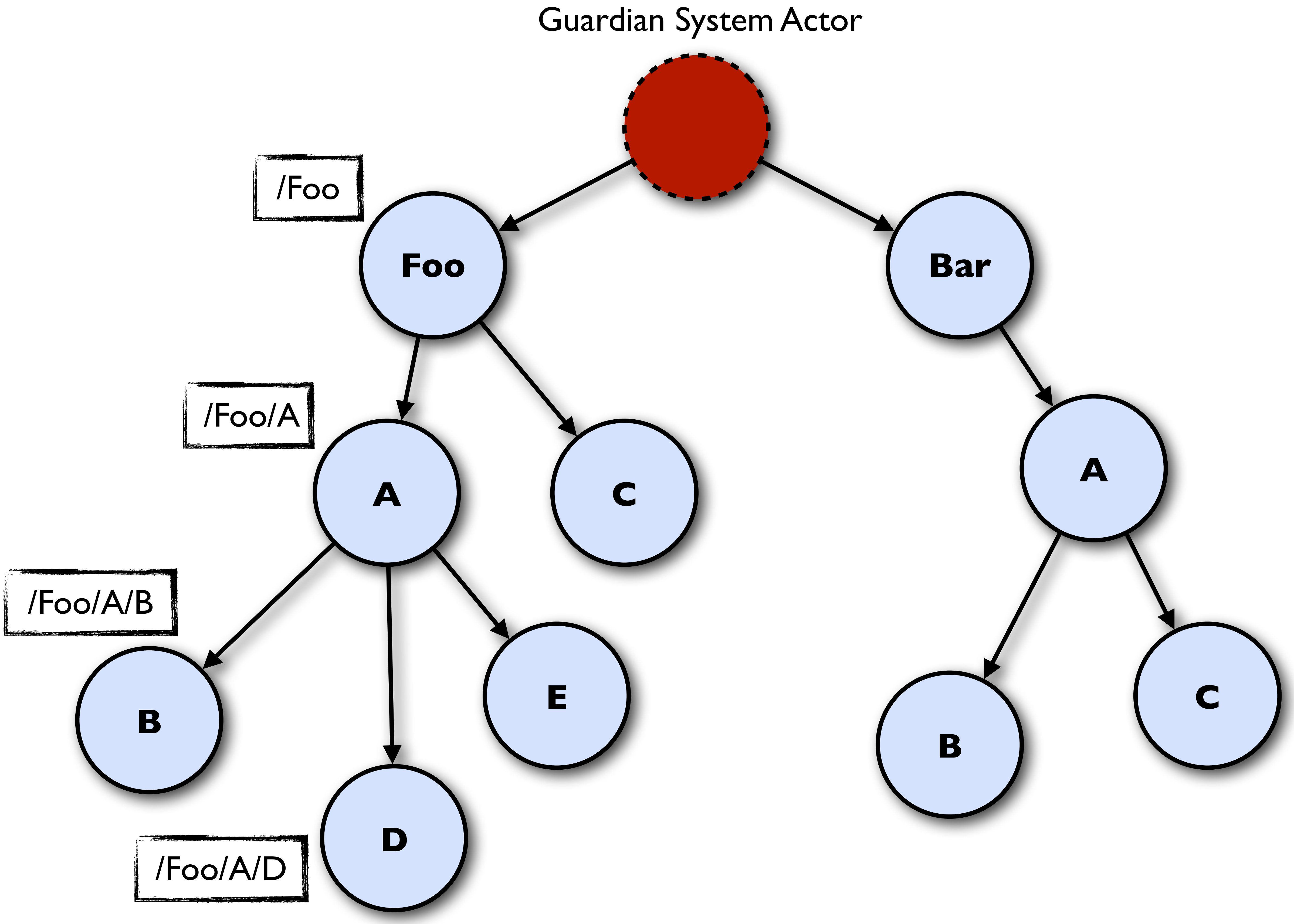
Name resolution



Name resolution



Name resolution



Supervision

```
class MySupervisor extends Actor {
  def supervisorStrategy = 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)
  }
}
```

Scala API

Supervision

```
class MySupervisor extends Actor {
  def supervisorStrategy = 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)
  }
}
```

Scala API

Manage failure

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

Scala API

watch/unwatch

```
val buddy: ActorRef = ...

val watcher = system.actorOf(Props(
  new Actor {
    context.watch(buddy)

    def receive = {
      case t: Terminated => ...
    }
  }
))
```

Scala API

Akka **2.1+**

The runtime provides

Decentralized P2P gossip-based cluster membership (dynamo-style w/ vector clocks, hand-off on fail-over etc.)

The runtime provides

Automatic adaptive cluster rebalancing

The runtime provides

Automatic cluster-wide deployment

The runtime provides

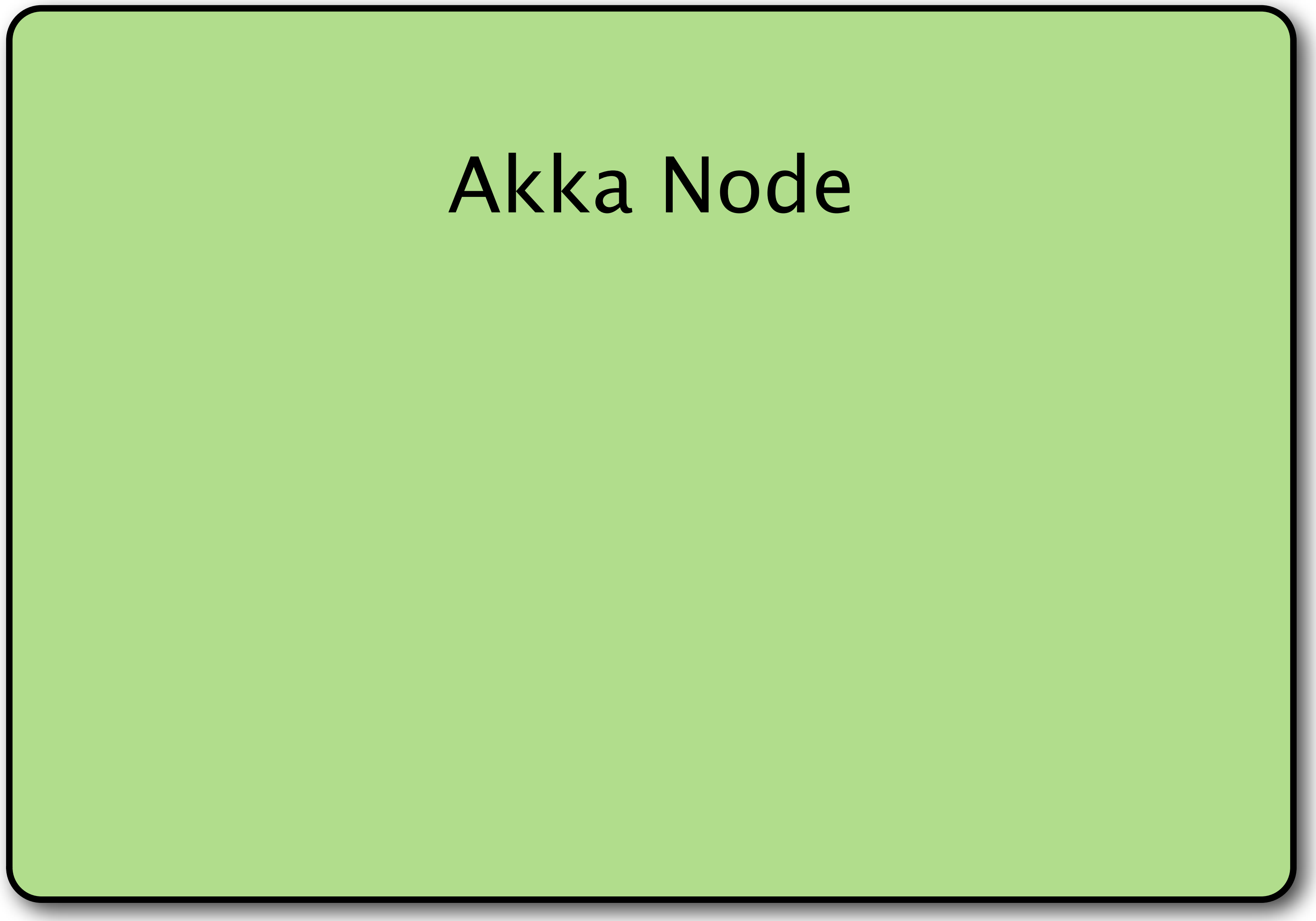
Highly available configuration service

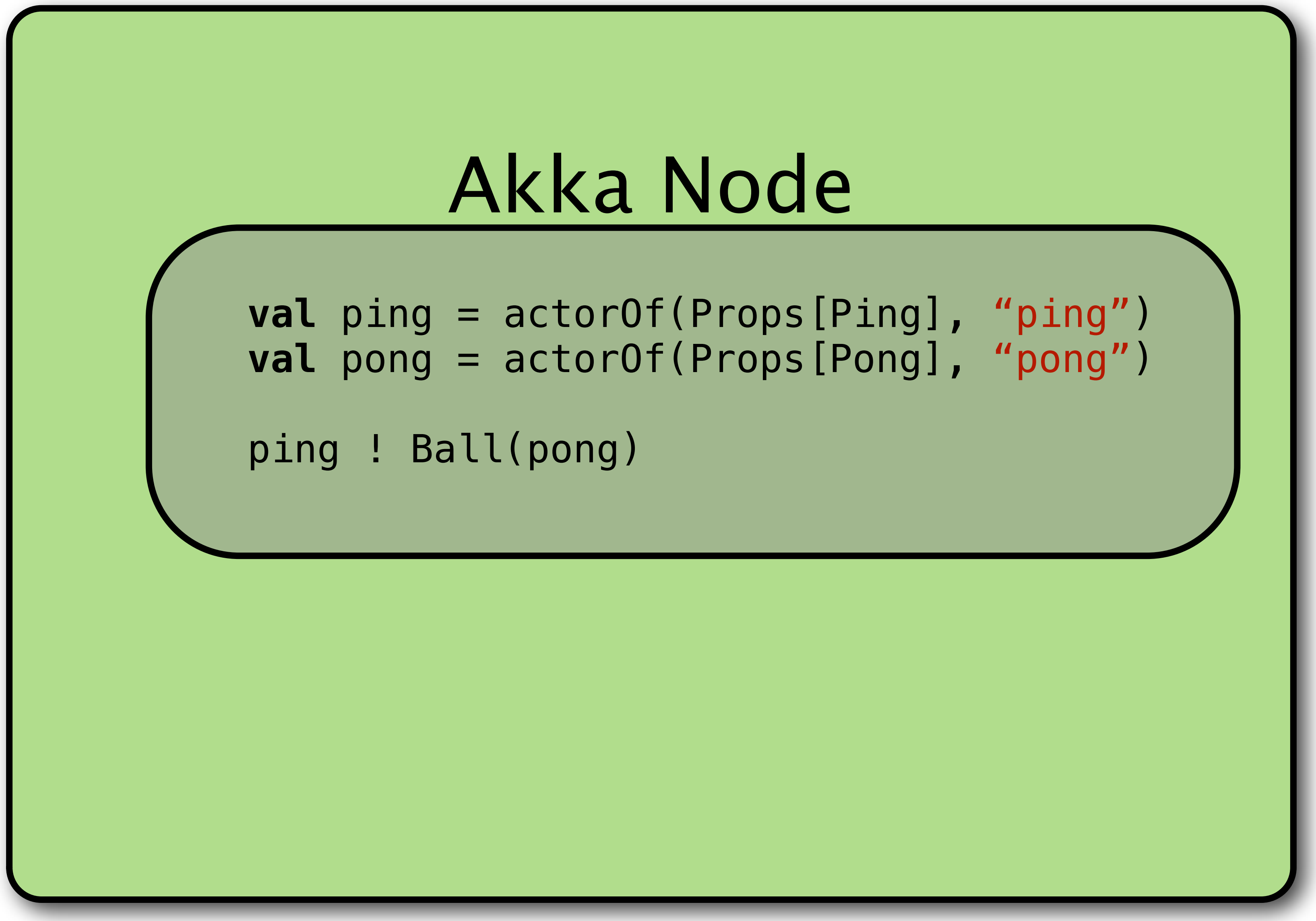
The runtime provides

Automatic replication with automatic fail-over upon node crash

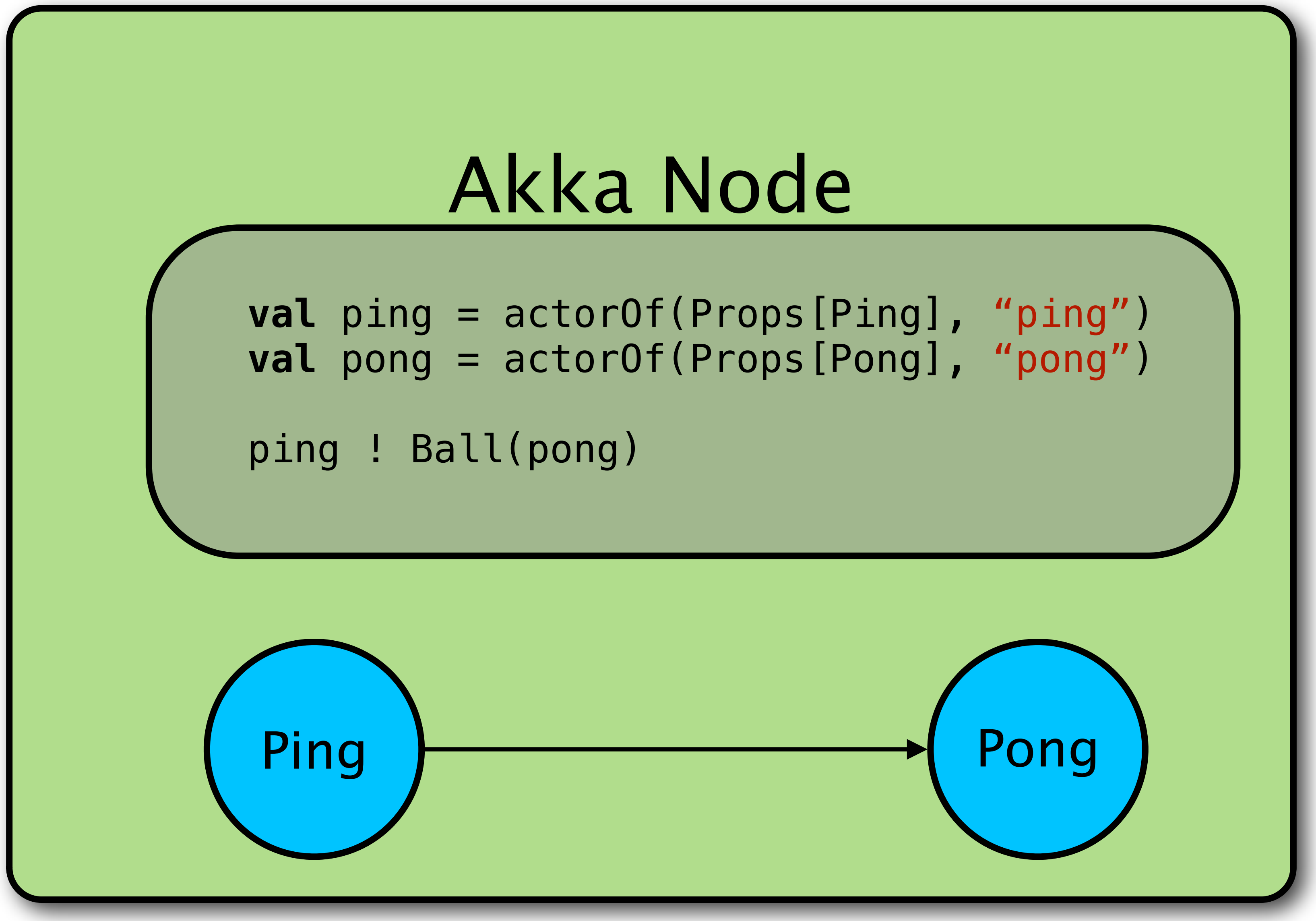
The runtime provides

Transparent and user-configurable load-balancing

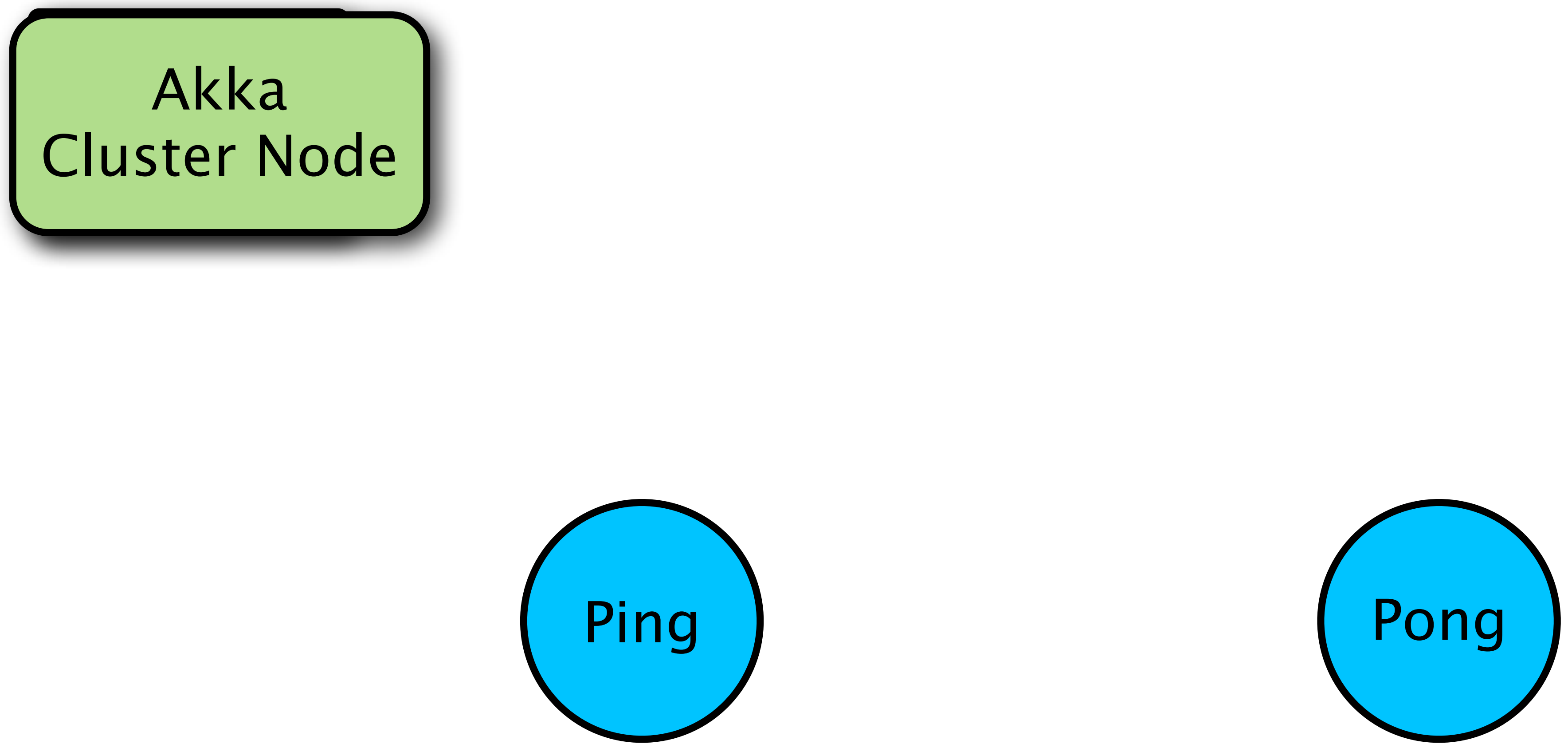

Akka Node

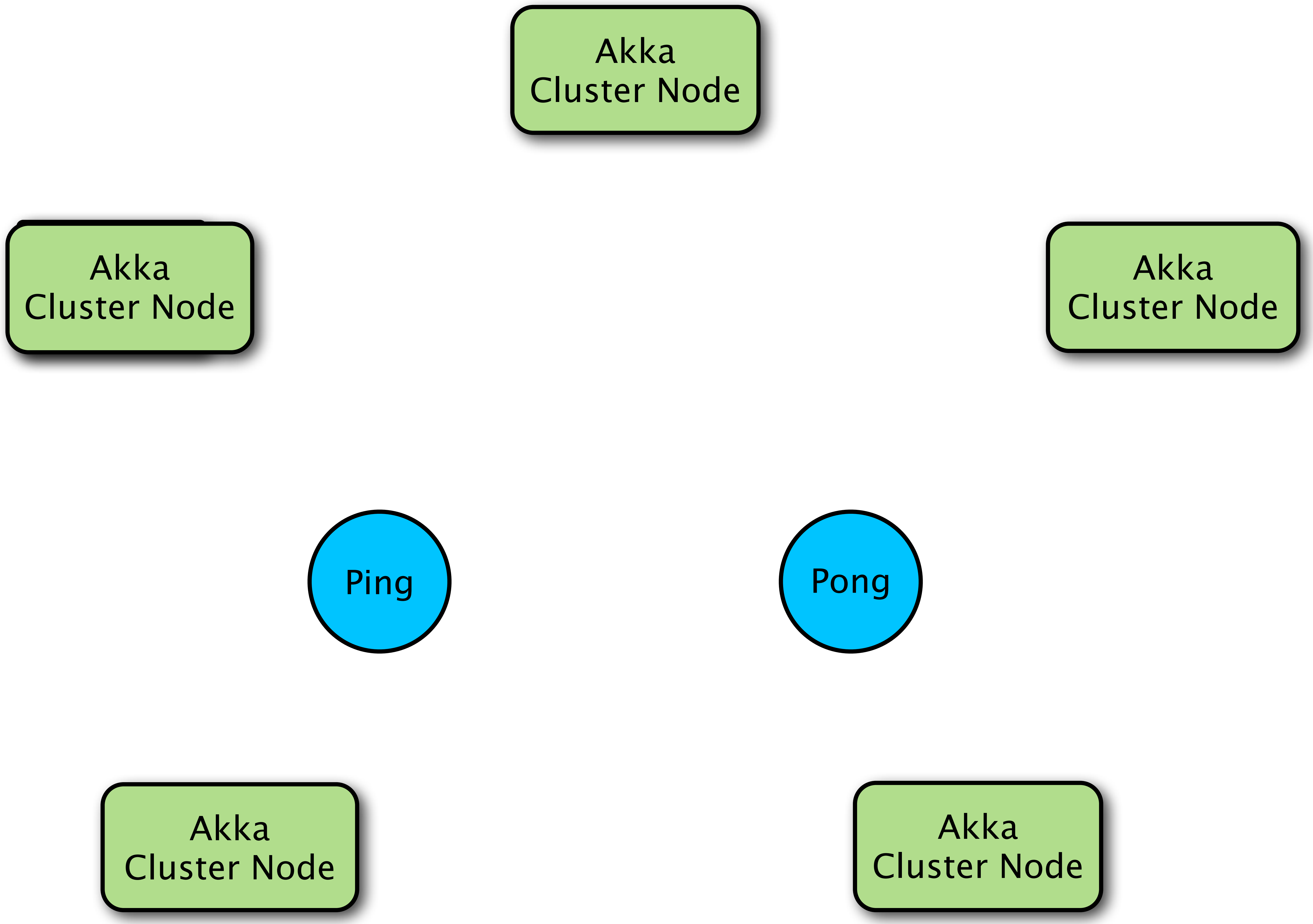


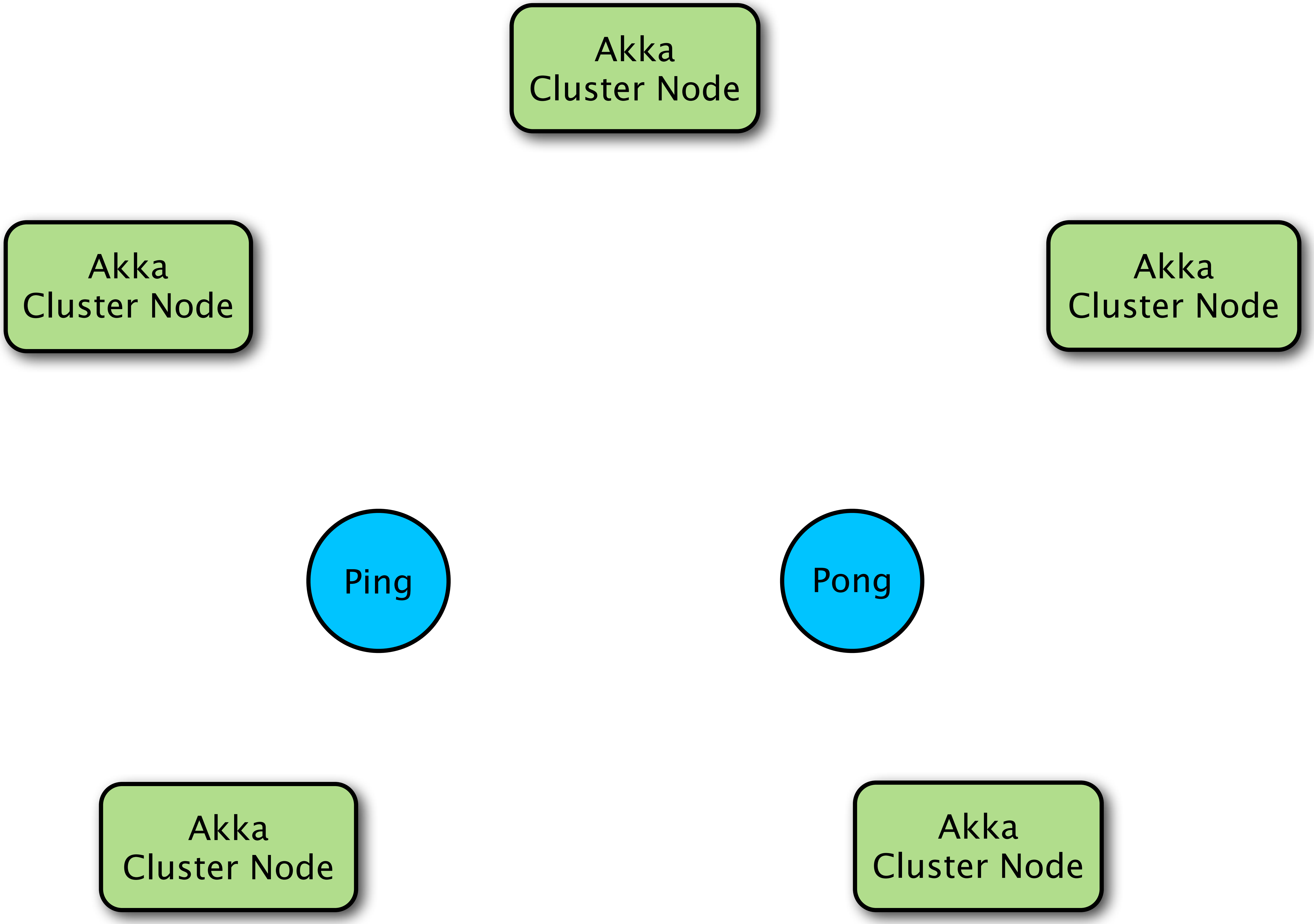
```
val ping = actorOf(Props[Ping], "ping")
val pong = actorOf(Props[Pong], "pong")

ping ! Ball(pong)
```

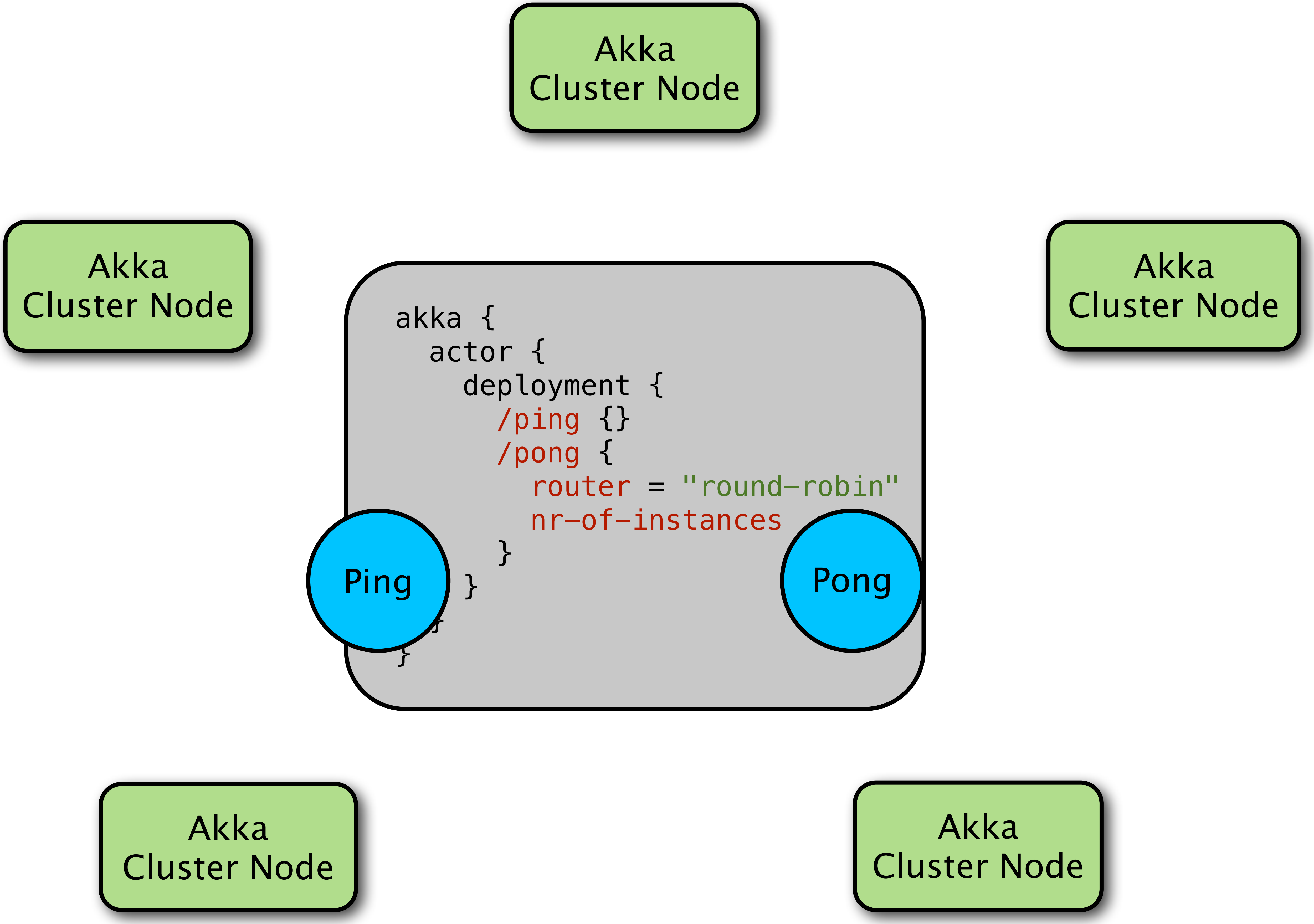


```
val ping = actorOf(Props[Ping], “ping”)
val pong = actorOf(Props[Pong], “pong”)

ping ! Ball(pong)
```

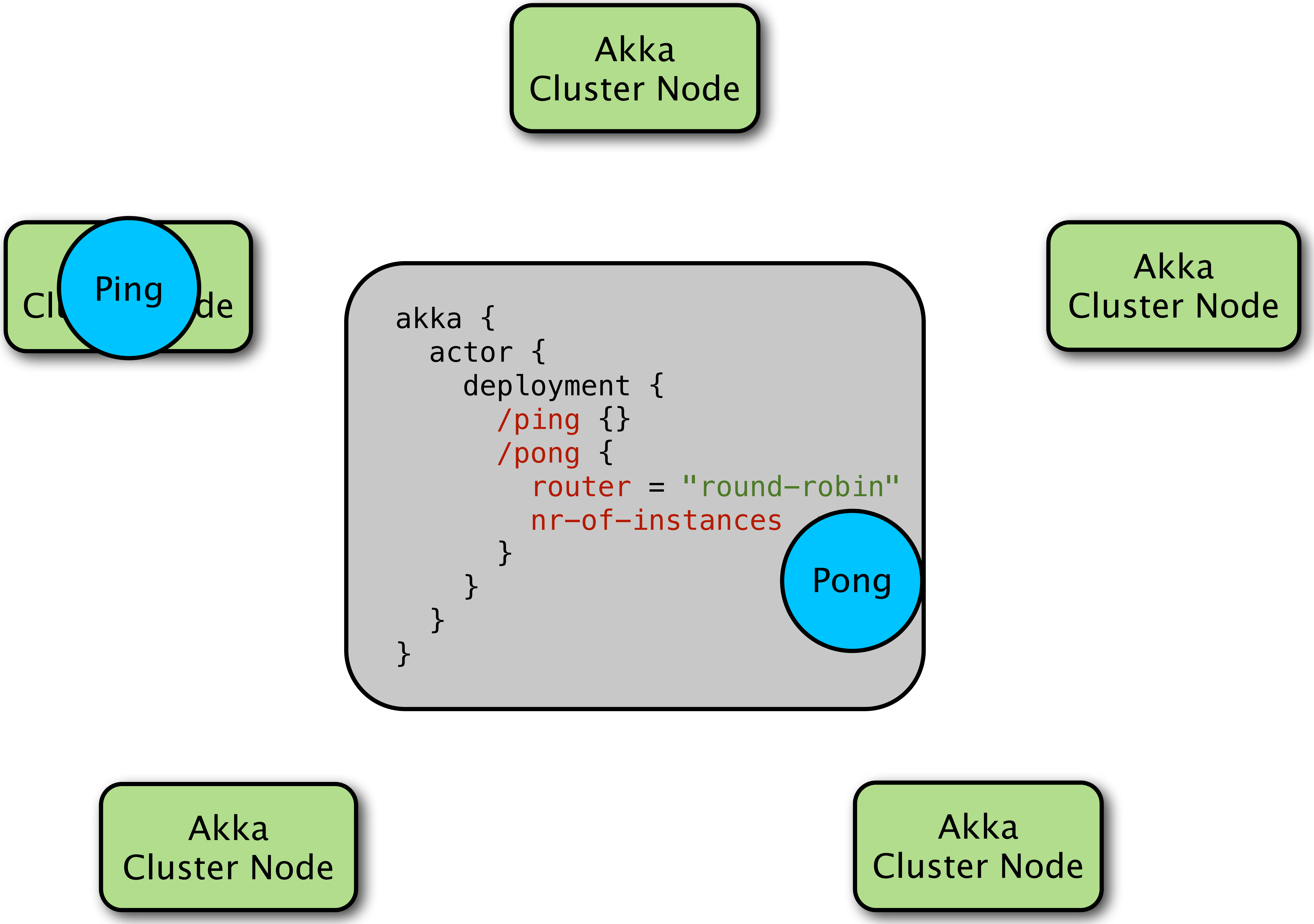

Akka
Cluster Node
Ping
Pong


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

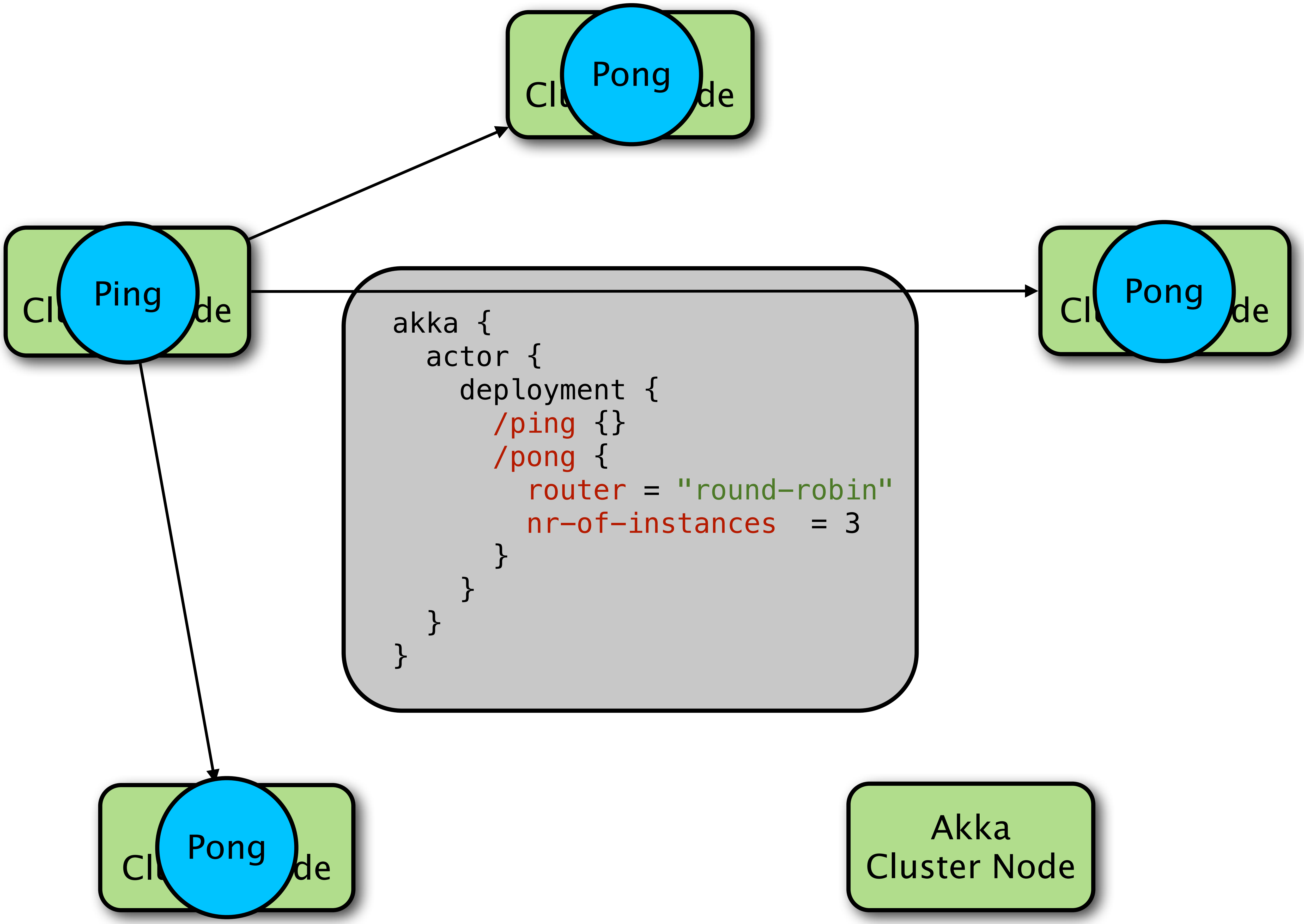

Akka Cluster Node
Akka Cluster Node
Akka Cluster Node
Ping
Pong
Akka Cluster Node
Akka Cluster Node



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



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



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

...and much much more

HTTP

Transactors

FSM

Durable Mailboxes

Camel

Microkernel

SLF4J

NIO

ZeroMQ

Dataflow

AMQP

Agents

Spring

TestKit

Get it and learn more

http://akka.io

http://typesafe.com

EOF