Scaling
software with

Henrik Engström

Software Engineer at Typesafe

@h3nk3

Scaling
software with


Copyright Ingeborg van Leeuwen

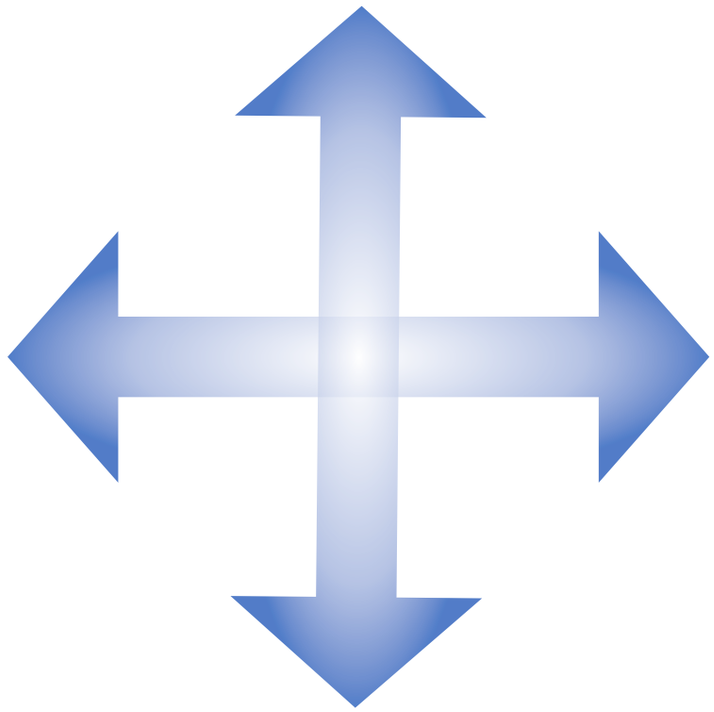
Leverage what you already know

All this is working with the Java investment; software, skills, tools and techniques, you have today:

- USE Akka from Java or Scala today
- INTEGRATE with, and deploy into, your current infrastructure/environment

Selection of Akka Production Users

TDC
Telefónica
CSC
htc quietly brilliant
hp
BLIZZARD ENTERTAINMENT
UBS
SIEMENS
amazon.com
KLOUT
W3C®
HSBC
CISCO
HUAWEI
JUNIPER NETWORKS
svt
Autodesk
CREDIT SUISSE
IGN
Atos
O2
vmware
dialog Smart Stream Platform
SEVEN Networks
xerox
DRW TRADING GROUP
OOYALA
novus
abiquo
foursquare
T8 Webware
CARTOMAPIC
moshi monsters
ROVIO
BBC
navirec
Maritime Poker
PRECOG
Answers.com The world's leading Q&A site
azavea
banksimple
zeebox The best thing to happen to TV since TV



Typesafe

akka

Manage System Overload

Typesafe

akka

Program at a Higher Level

Program at a Higher Level

Program at a Higher Level

- Never think in terms of shared state, state visibility, threads, locks, concurrent collections, thread notifications etc.

Program at a Higher Level

- Never think in terms of shared state, state visibility, threads, locks, concurrent collections, thread notifications etc.
- Low level concurrency plumbing BECOMES SIMPLE WORKFLOW - you only think about how messages flow in the system

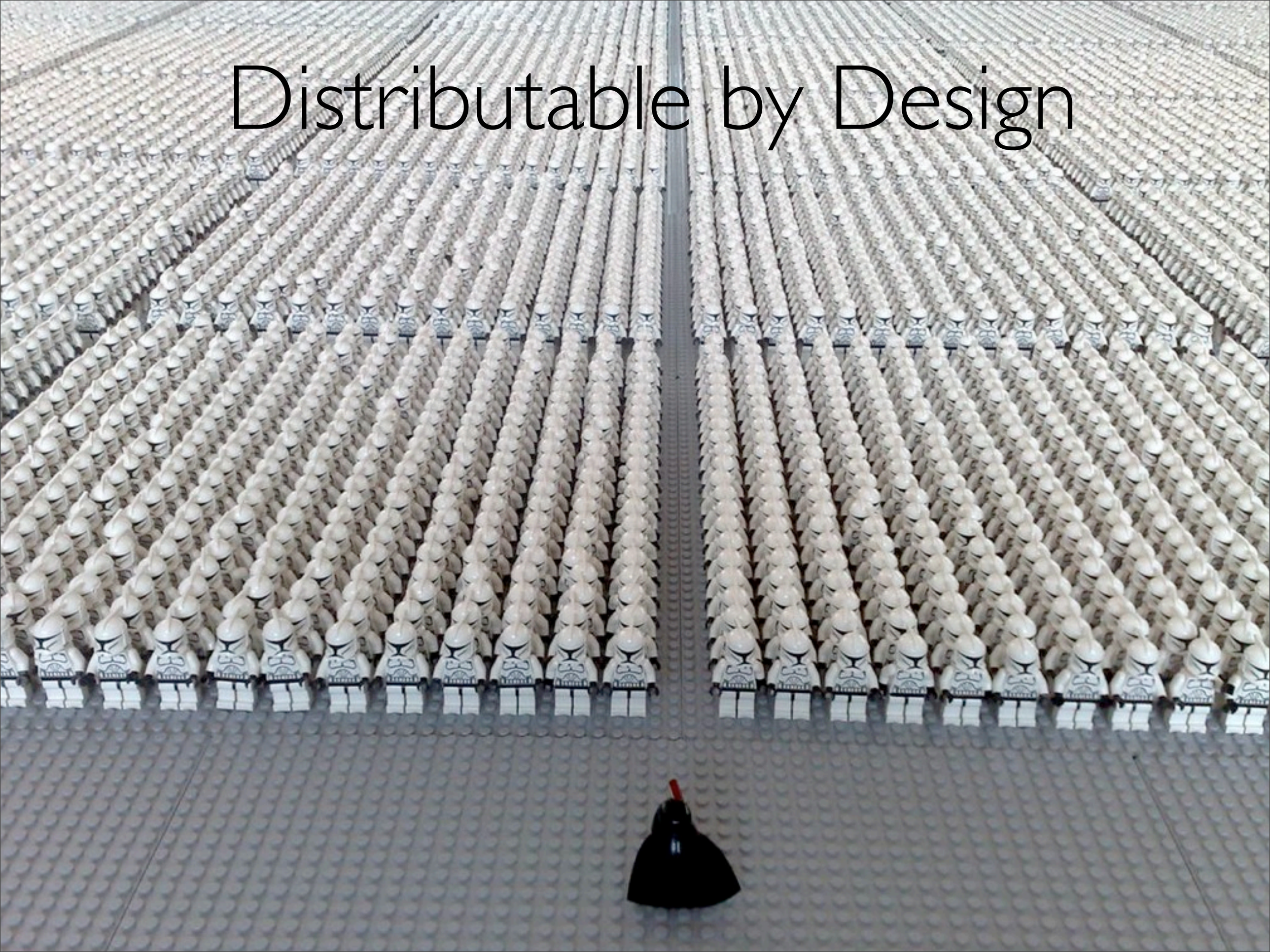
Program at a Higher Level

- Never think in terms of shared state, state visibility, threads, locks, concurrent collections, thread notifications etc.
- Low level concurrency plumbing BECOMES SIMPLE WORKFLOW - you only think about how messages flow in the system
- You get high CPU utilization, low latency, high throughput and scalability - FOR FREE as part of the model

Program at a Higher Level

- Never think in terms of shared state, state visibility, threads, locks, concurrent collections, thread notifications etc.
- Low level concurrency plumbing BECOMES SIMPLE WORKFLOW - you only think about how messages flow in the system
- You get high CPU utilization, low latency, high throughput and scalability - FOR FREE as part of the model
- Proven and superior model for detecting and recovering from errors

Distributable by Design



Distributable by Design

Typesafe

akka

Distributable by Design

- Actors are location transparent & distributable by design

Typesafe

akka

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model
- You get the PERFECT FABRIC for the CLOUD

Typesafe

akka

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model
- You get the PERFECT FABRIC for the CLOUD
  - elastic & dynamic

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model
- You get the PERFECT FABRIC for the CLOUD
  - elastic & dynamic
  - fault-tolerant & self-healing

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model
- You get the PERFECT FABRIC for the CLOUD
  - elastic & dynamic
  - fault-tolerant & self-healing
  - adaptive load-balancing, cluster rebalancing & actor migration

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model
- You get the PERFECT FABRIC for the CLOUD
  - elastic & dynamic
  - fault-tolerant & self-healing
  - adaptive load-balancing, cluster rebalancing & actor migration
  - build extremely loosely coupled and dynamic systems that can change and adapt at runtime

Business benefits

Typesafe

akka

Lower DEVELOPMENT COST

Lower DEVELOPMENT COST

- More functionality and scalability with less lines of code

Lower DEVELOPMENT COST

- More functionality and scalability with less lines of code
- Faster to BUILD the system

Lower DEVELOPMENT COST

- More functionality and scalability with less lines of code
- Faster to BUILD the system
- Easier to UNDERSTAND the system

Lower DEVELOPMENT COST

- More functionality and scalability with less lines of code
- Faster to BUILD the system
- Easier to UNDERSTAND the system
- Easier to MAINTAIN the system

Lower DEVELOPMENT COST

- More functionality and scalability with less lines of code
- Faster to BUILD the system
- Easier to UNDERSTAND the system
- Easier to MAINTAIN the system
- Easier to EVOLVE the system

Lower DEVELOPMENT COST

- More functionality and scalability with less lines of code
- Faster to BUILD the system
- Easier to UNDERSTAND the system
- Easier to MAINTAIN the system
- Easier to EVOLVE the system
- You will encounter LESS BUGS

Lower OPERATIONS COST

Lower OPERATIONS COST

- Open Source - Apache 2 license

Typesafe

akka

Lower OPERATIONS COST

- Open Source - Apache 2 license
- Lower cost on HARDWARE since

Lower OPERATIONS COST

- Open Source - Apache 2 license
- Lower cost on HARDWARE since
  - resource utilization is increased - less money on hardware

Lower OPERATIONS COST

- Open Source - Apache 2 license
- Lower cost on HARDWARE since
  - resource utilization is increased - less money on hardware
  - elasticity allows you to grow and shrink the system on demand - perfect in virtualized environments

Lower OPERATIONS COST

- Open Source - Apache 2 license
- Lower cost on HARDWARE since
  - resource utilization is increased - less money on hardware
  - elasticity allows you to grow and shrink the system on demand - perfect in virtualized environments
- Lower cost caused by unplanned DOWNTIME

Lower OPERATIONS COST

- Open Source - Apache 2 license
- Lower cost on HARDWARE since
  - resource utilization is increased - less money on hardware
  - elasticity allows you to grow and shrink the system on demand - perfect in virtualized environments
- Lower cost caused by unplanned DOWNTIME
- Allows operations team to be more EFFICIENT

Lower OPERATIONS COST

- Open Source - Apache 2 license
- Lower cost on HARDWARE since
  - resource utilization is increased - less money on hardware
  - elasticity allows you to grow and shrink the system on demand - perfect in virtualized environments
- Lower cost caused by unplanned DOWNTIME
- Allows operations team to be more EFFICIENT
  - get control of the system with the Typesafe Console

Address NEW business requirements

Tackle large scale business IT challenges you otherwise wouldn't be able to address, e.g:

- go from batch analytics to real time analytics
- scale your systems along with financial markets or social media volume

How

can we achieve this?

Typesafe

akka

Let's use Actors

What is an Actor?

What is an Actor?

- Akka's unit of code organization is called an Actor

What is an Actor?

- Akka's unit of code organization is called an Actor
- Actors helps you create concurrent, scalable and fault-tolerant applications

What is an Actor?

- Akka's unit of code organization is called an Actor
- Actors helps you create concurrent, scalable and fault-tolerant applications
- Actors is a model for organizing your code that keeps many "policy decisions" separate from the business logic

What is an Actor?

- Akka's unit of code organization is called an Actor
- Actors helps you create concurrent, scalable and fault-tolerant applications
- Actors is a model for organizing your code that keeps many "policy decisions" separate from the business logic
- Actors may be new to many in the Java community, but they are a tried-and-true concept (Hewitt 1973) used for many years in telecom systems with 9 nines uptime

What can I use Actors for?

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread
- an object instance or component

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread
- an object instance or component
- a callback or listener

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread
- an object instance or component
- a callback or listener
- a singleton or service

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread
- an object instance or component
- a callback or listener
- a singleton or service
- a router, load-balancer or pool

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread
- an object instance or component
- a callback or listener
- a singleton or service
- a router, load-balancer or pool
- an out-of-process service

What can I use Actors for?

In different scenarios, an Actor may be an alternative to:

- a thread
- an object instance or component
- a callback or listener
- a singleton or service
- a router, load-balancer or pool
- an out-of-process service
- a Finite State Machine (FSM)

So, what is the

Actor Model?

Carl Hewitt's definition

http://bit.ly/hewitt-on-actors

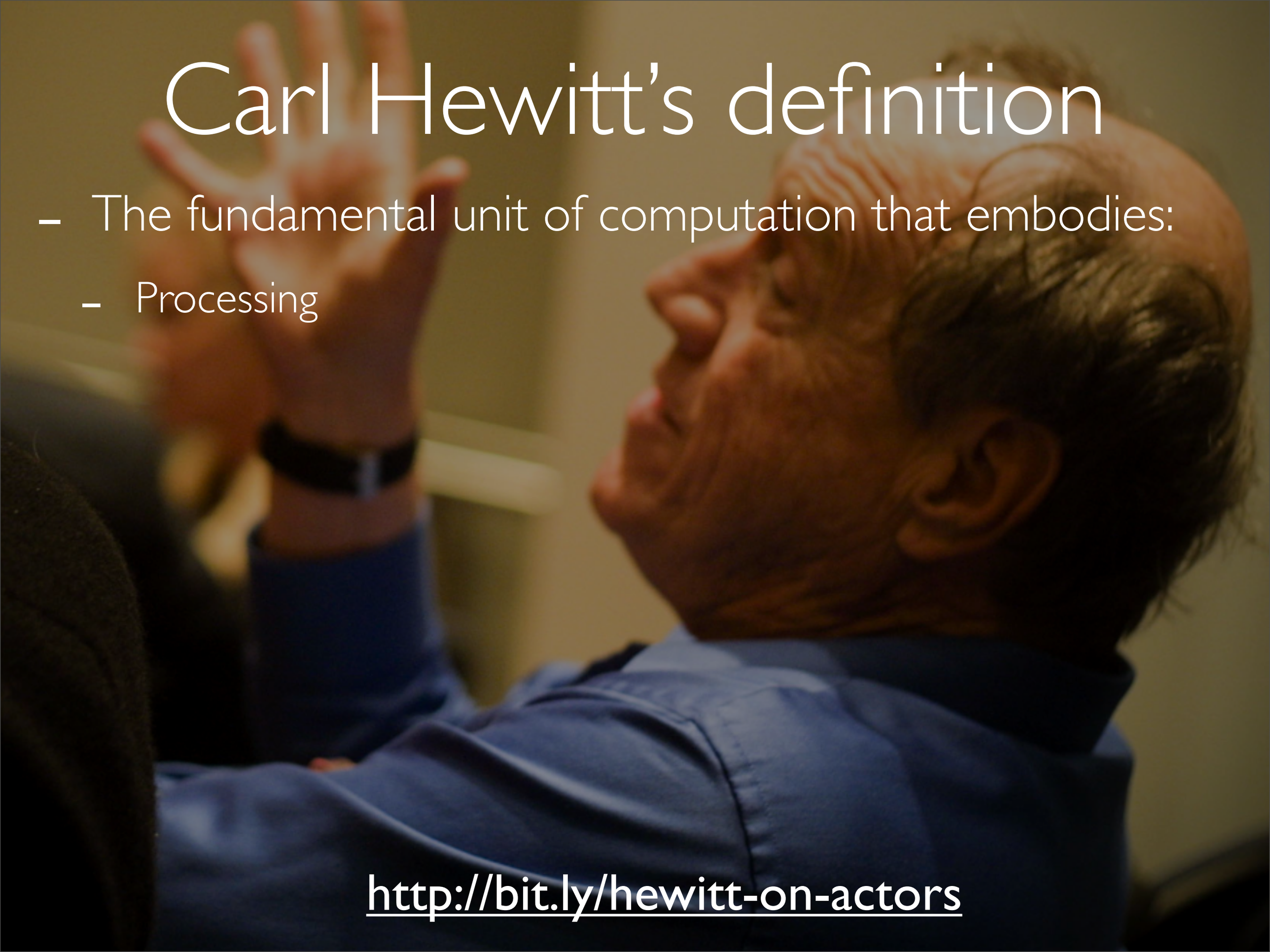
Carl Hewitt's definition

- The fundamental unit of computation that embodies:

http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing



http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage

http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage
  - Communication

http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage
  - Communication
- 3 axioms - When an Actor receives a message it can:

http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage
  - Communication
- 3 axioms - When an Actor receives a message it can:
  - Create new Actors

http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage
  - Communication
- 3 axioms - When an Actor receives a message it can:
  - Create new Actors
  - Send messages to Actors it knows

http://bit.ly/hewitt-on-actors

Carl Hewitt's definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage
  - Communication
- 3 axioms - When an Actor receives a message it can:
  - Create new Actors
  - Send messages to Actors it knows
  - Designate how it should handle the next message it receives

http://bit.ly/hewitt-on-actors

4 core Actor operations

0. DEFINE
1. CREATE
2. SEND
3. BECOME
4. SUPERVISE

0. DEFINE

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}
```

0. DEFINE

Define the message(s) the Actor should be able to respond to

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}
```

0. DEFINE

Define the message(s) the Actor should be able to respond to

Define the Actor class

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}
```

0. DEFINE

Define the message(s) the Actor should be able to respond to

Define the Actor class

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}
```

Define the Actor's behavior

1. CREATE

- CREATE - creates a new instance of an Actor
- Extremely lightweight (2.7 Million per GB RAM)
- Very strong encapsulation - encapsulates:
  - state
  - behavior
  - message queue
- State & behavior is indistinguishable from each other
  - Only way to observe state is by sending an actor a message and see how it reacts

CREATE Actor

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
```

Typesafe

akka

CREATE Actor

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(w                    o " + who)
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
```

Create an Actor system

Typesafe

akka

CREATE Actor

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(w
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
```

Create an Actor system

Actor configuration

CREATE Actor

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(w
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
```

Create an Actor system

Actor configuration

Give it a name

Typesafe

akka

CREATE Actor

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(w
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
```

Create an Actor system

Actor configuration

Create the Actor

Give it a name

Typesafe

akka

CREATE Actor

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(w
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
```

Create an Actor system

Actor configuration

You get an ActorRef back

Create the Actor

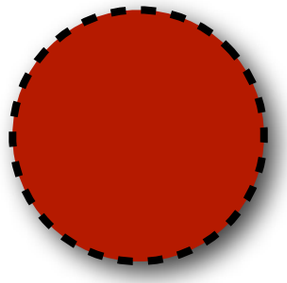
Give it a name

Typesafe

akka

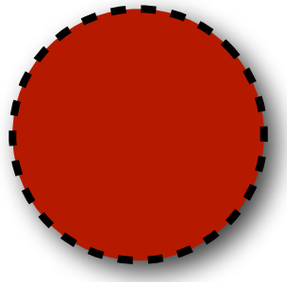
Actors can form hierarchies

Guardian Actor



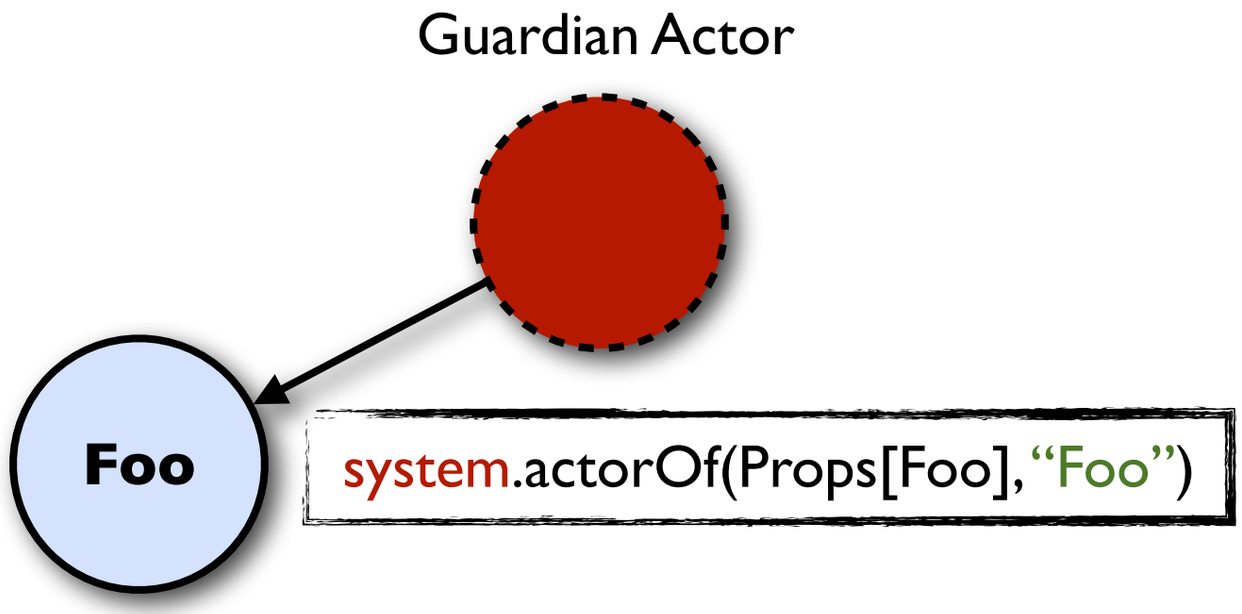
Actors can form hierarchies

Guardian Actor

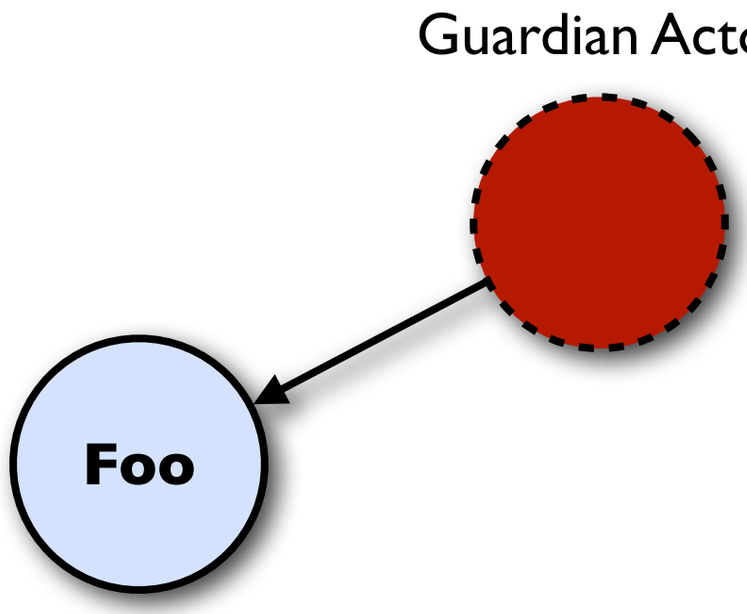


system.actorOf(Props[Foo], "Foo")

Actors can form hierarchies

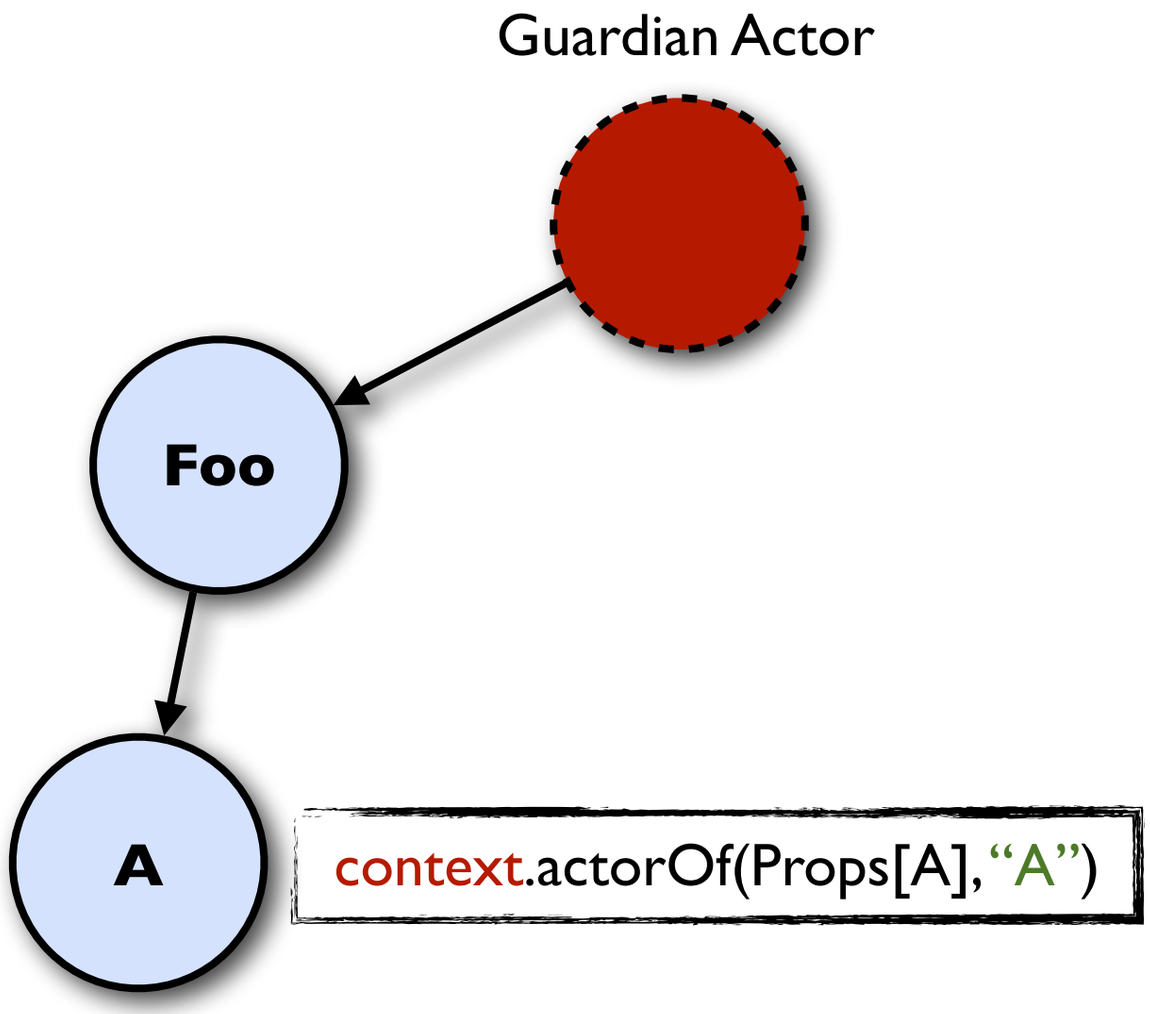


Actors can form hierarchies

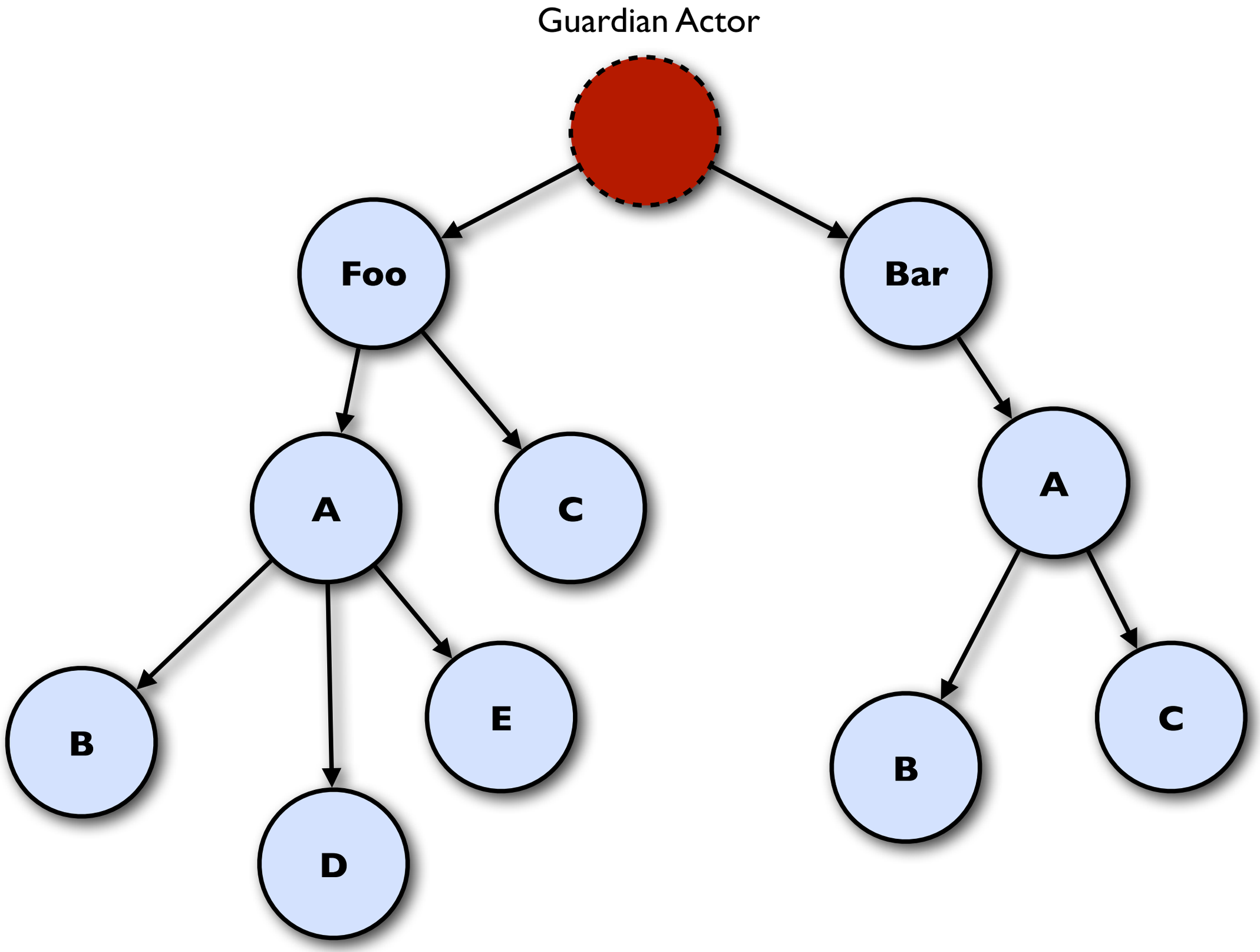



```
context.actorOf(Props[A], "A")
```

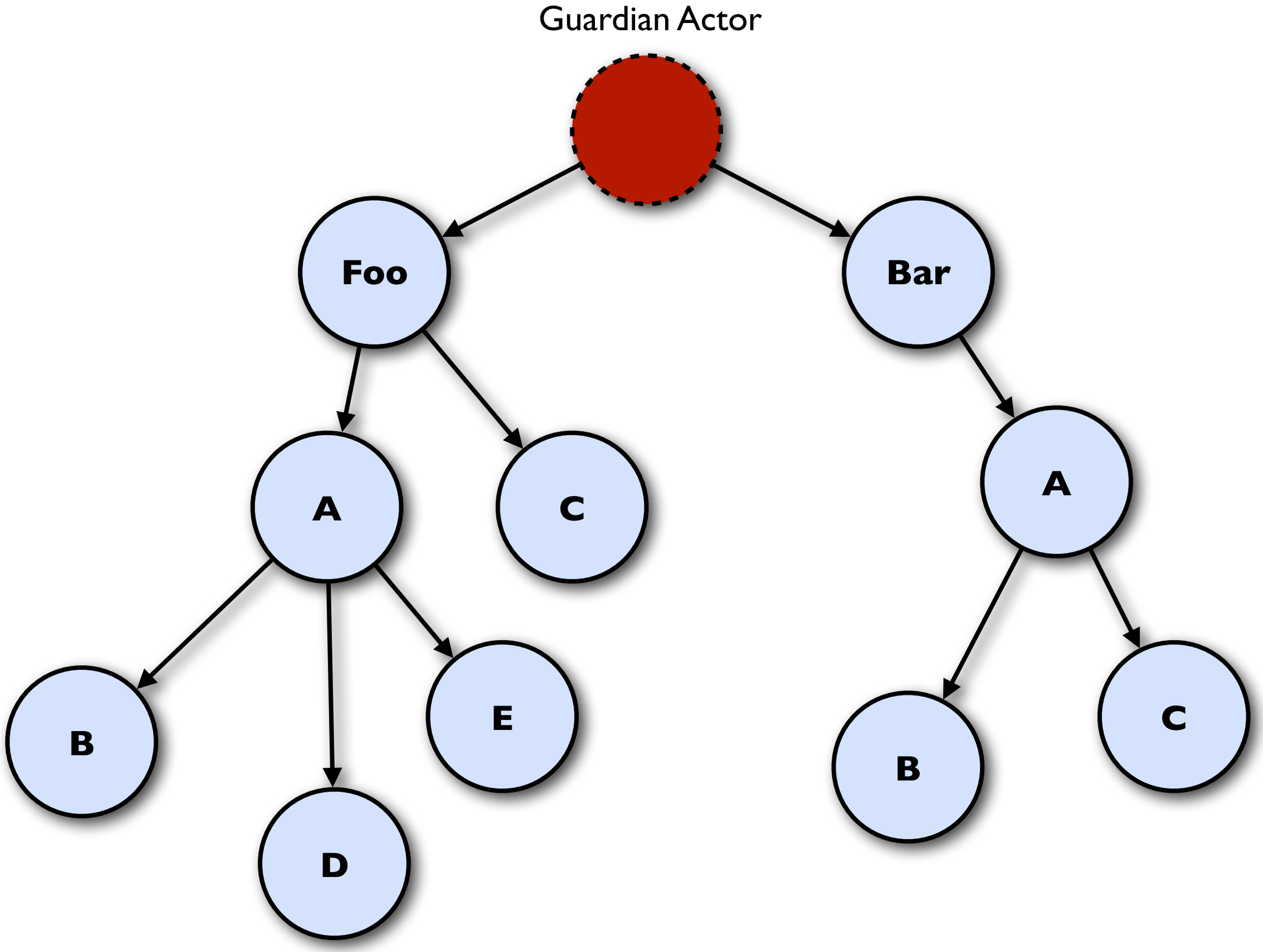
Actors can form hierarchies



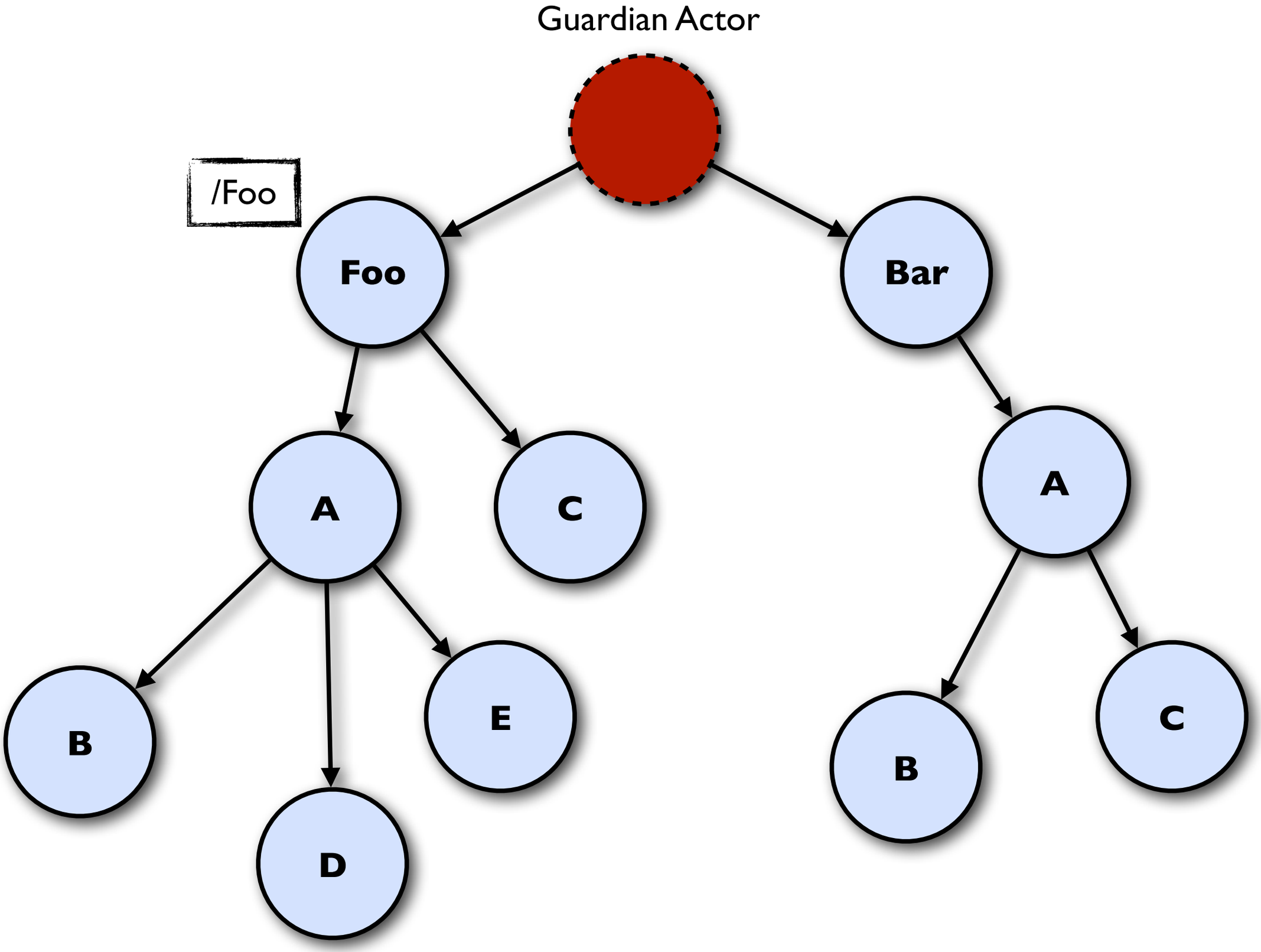
Actors can form hierarchies



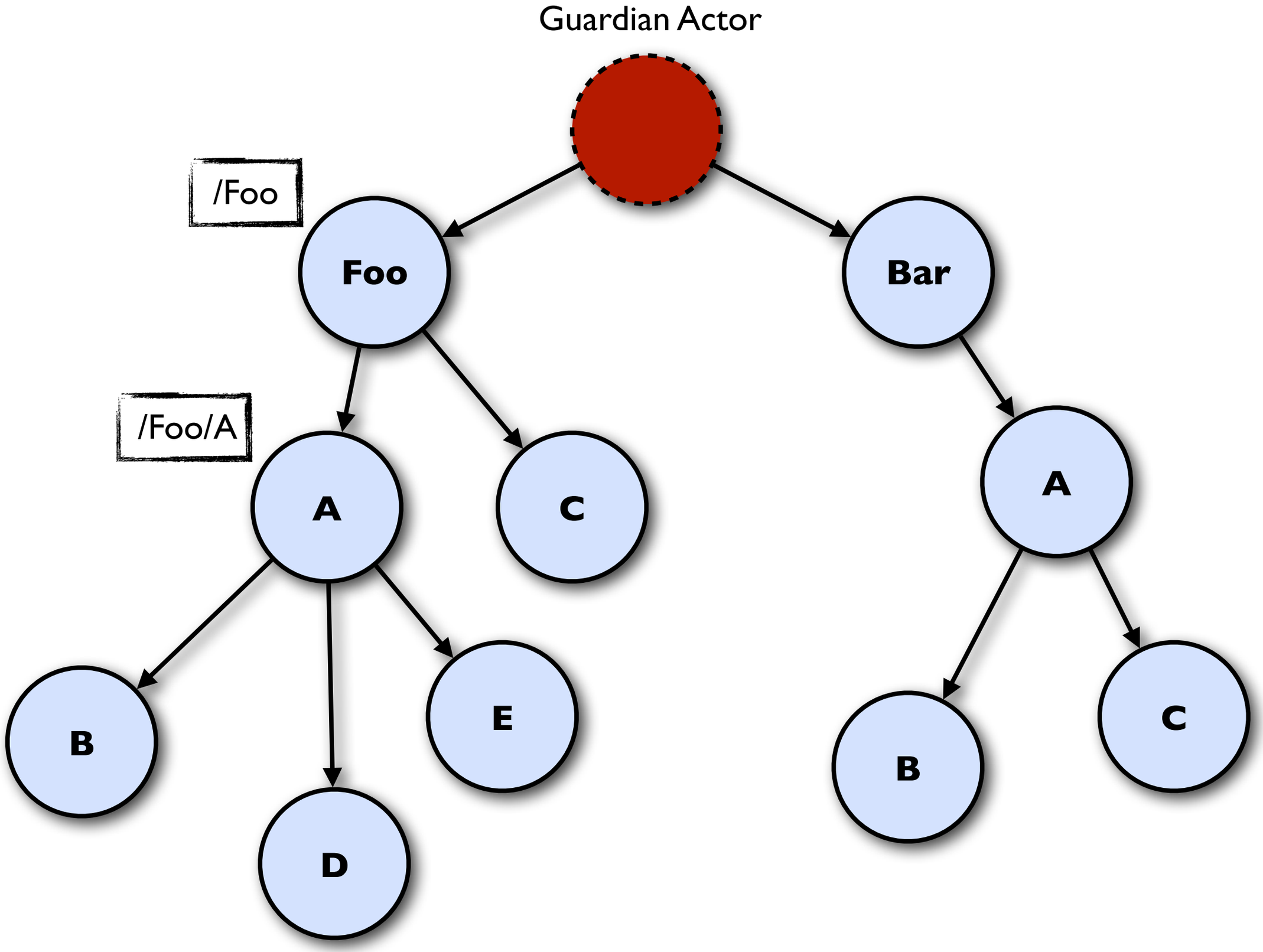
Name resolution - like a file-system



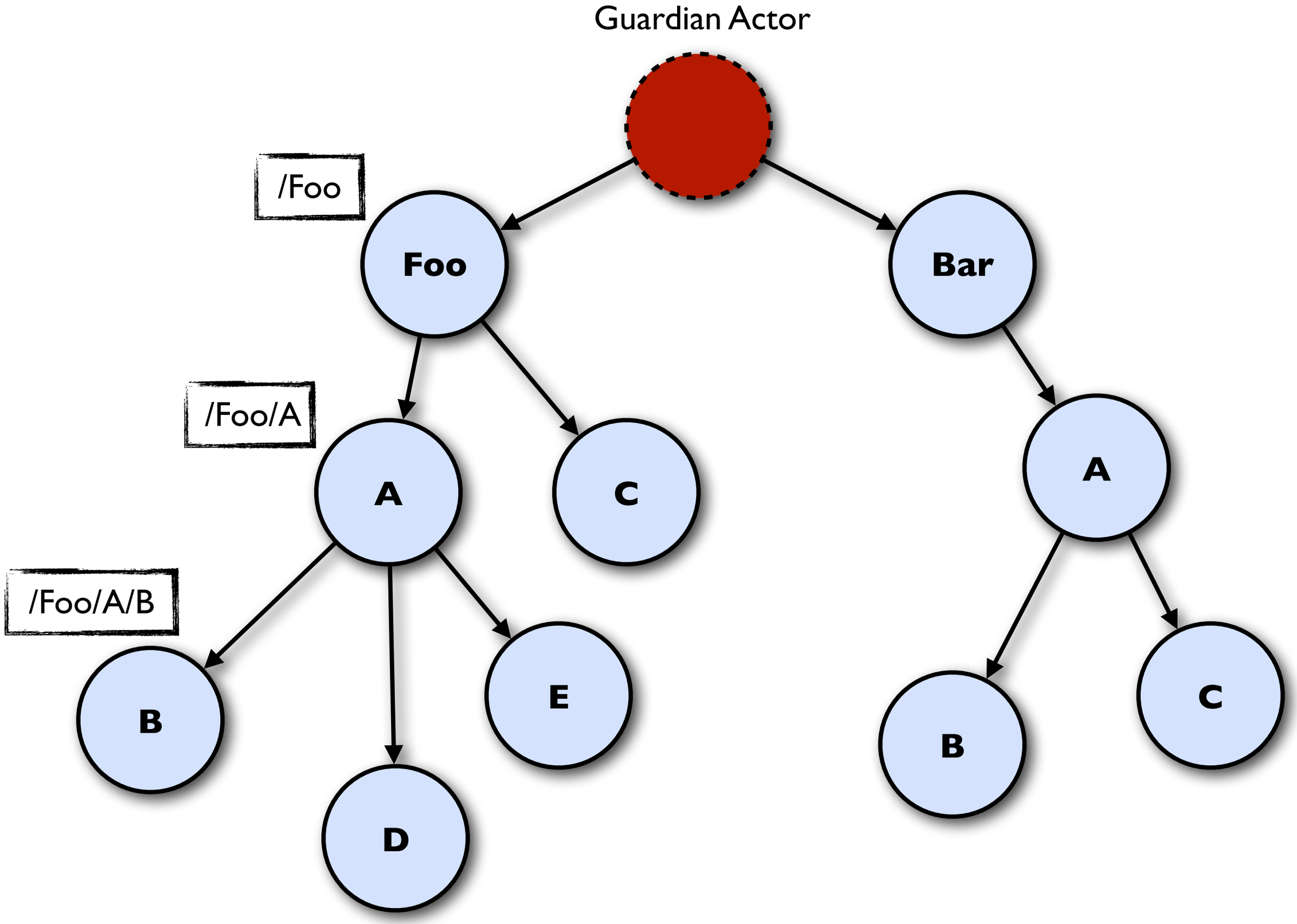
Name resolution - like a file-system



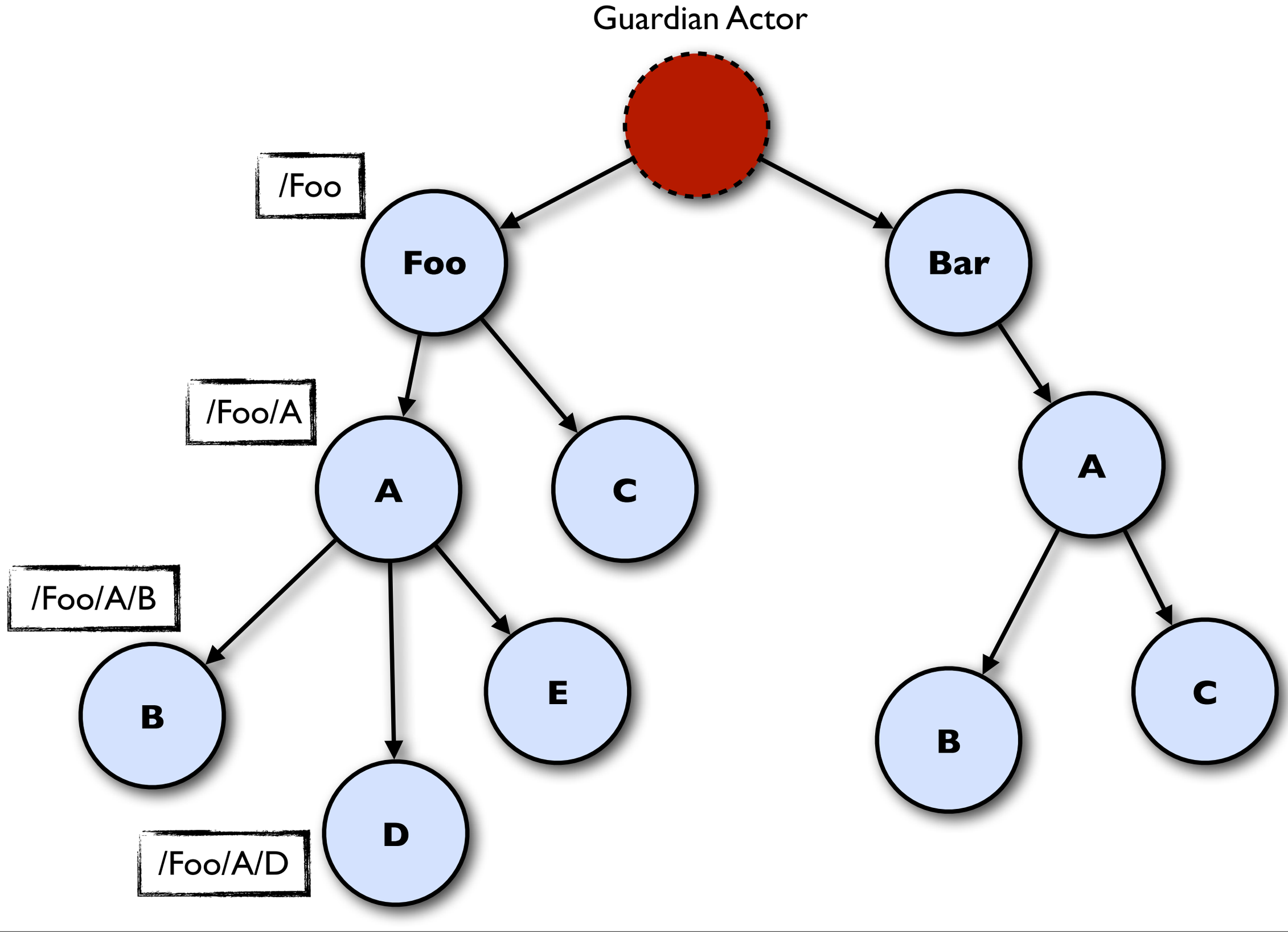
Name resolution - like a file-system



Name resolution - like a file-system



Name resolution - like a file-system



2. SEND

2. SEND

- SEND - sends a message to an Actor

2. SEND

- SEND - sends a message to an Actor
- Asynchronous and Non-blocking - Fire-forget

2. SEND

- SEND - sends a message to an Actor
- Asynchronous and Non-blocking - Fire-forget
- EVERYTHING is asynchronous and lockless

2. SEND

- SEND - sends a message to an Actor
- Asynchronous and Non-blocking - Fire-forget
- EVERYTHING is asynchronous and lockless
- Everything happens *reactively*

2. SEND

- SEND - sends a message to an Actor
- Asynchronous and Non-blocking - Fire-forget
- EVERYTHING is asynchronous and lockless
- Everything happens *reactively*
  - An Actor is passive until a message is sent to it, which triggers something within the Actor

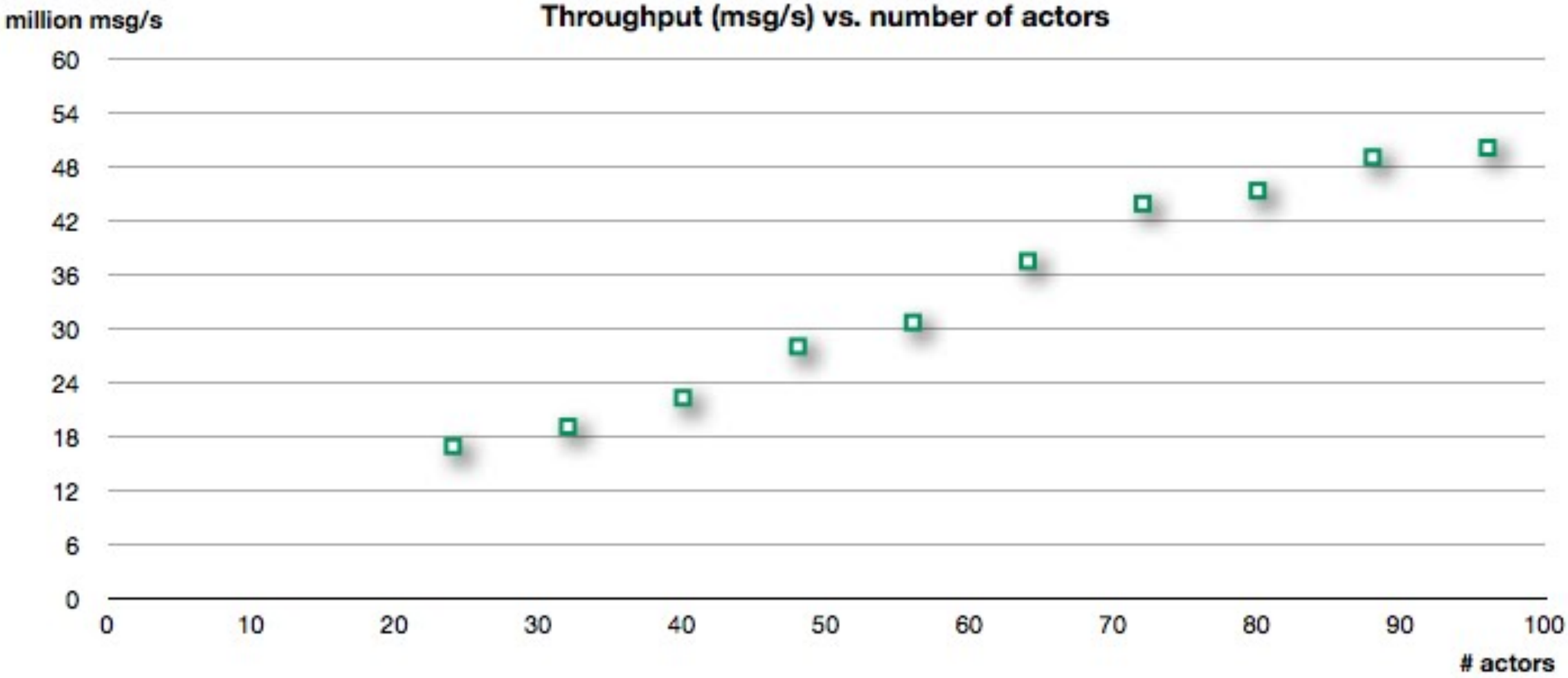
2. SEND

- SEND - sends a message to an Actor
- Asynchronous and Non-blocking - Fire-forget
- EVERYTHING is asynchronous and lockless
- Everything happens *reactively*
  - An Actor is passive until a message is sent to it, which triggers something within the Actor
  - Messages is the *kinetic energy* in an Actor system

2. SEND

- SEND - sends a message to an Actor
- Asynchronous and Non-blocking - Fire-forget
- EVERYTHING is asynchronous and lockless
- Everything happens *reactively*
  - An Actor is passive until a message is sent to it, which triggers something within the Actor
  - Messages is the *kinetic energy* in an Actor system
  - Actors can have lots of buffered *potential energy* but can't do anything with it until it is triggered by a message

Throughput on a single box @ 48 cores



+50 million messages per second

Typesafe

akka

SEND message

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
greeter ! Greeting("Scala Camp Krakow")
```

SEND message

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
greeter ! Greeting("Scala Camp Krakow")
```

Send the message

Full example

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) => log.info("Hello " + who)
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
greeter ! Greeting("Scala Camp Krakow")
```

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote =
      }
    }
  }
}
```

Remote deployment

Just feed the ActorSystem with this configuration

Configure a Remote Provider

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote =
      }
    }
  }
}
```

Typesafe

akka

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote =
      }
    }
  }
}
```

For the Greeter actor

Configure a Remote Provider

Typesafe

akka

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote =
      }
    }
  }
```

For the Greeter actor

Configure a Remote Provider

Define Remote Path

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote = akka://
      }
    }
  }
```

For the Greeter actor

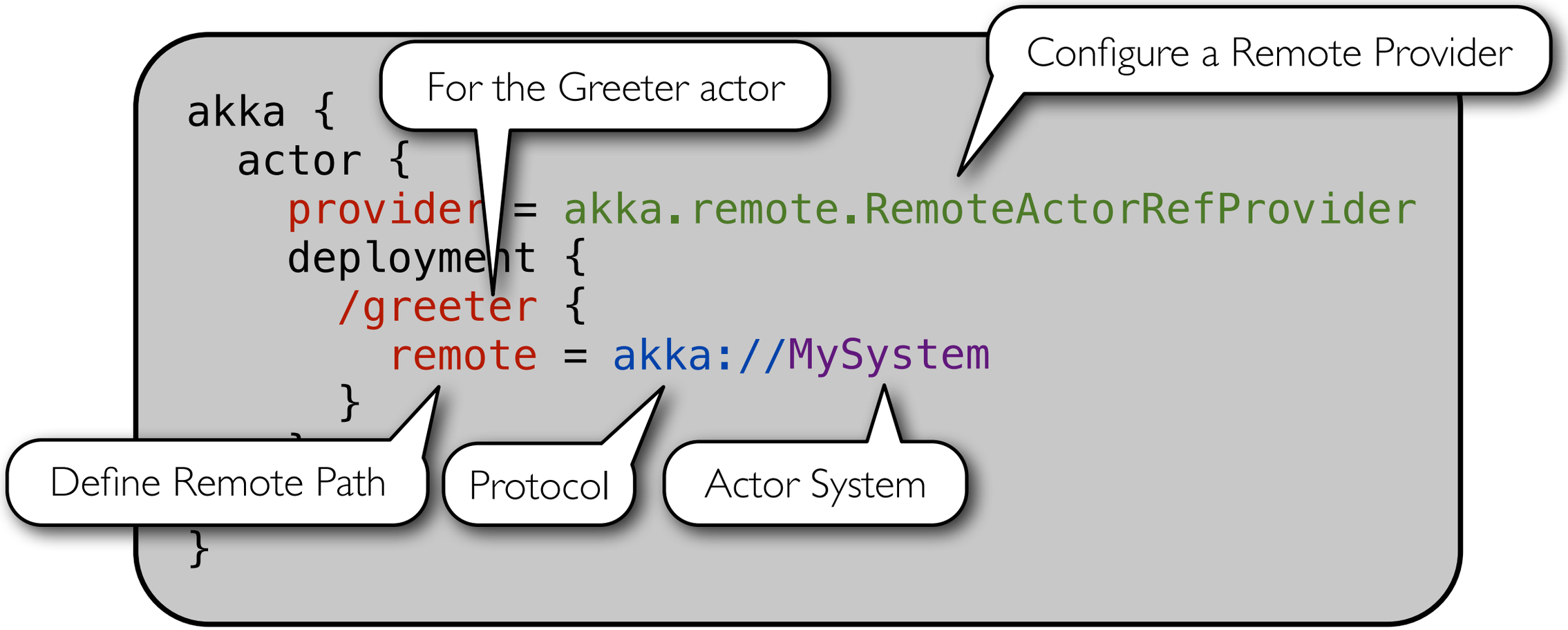
Configure a Remote Provider

Define Remote Path

Protocol

Remote deployment

Just feed the ActorSystem with this configuration



Typesafe

akka

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote = akka://MySystem@machine1
      }
    }
  }
}
```

For the Greeter actor

Configure a Remote Provider

Define Remote Path

Protocol

Actor System

Hostname

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote = akka://MySystem@machine1:2552
      }
    }
  }
}
```

For the Greeter actor

Configure a Remote Provider

Define Remote Path

Protocol

Actor System

Hostname

Port

Remote deployment

Just feed the ActorSystem with this configuration

```
akka {
  actor {
    provider = akka.remote.RemoteActorRefProvider
    deployment {
      /greeter {
        remote = akka://MySystem@machine1:2552
      }
    }
  }
}
```

For the Greeter actor

Configure a Remote Provider

Define Remote Path

Protocol

Actor System

Hostname

Port

Zero code changes

Typesafe

akka

Routers

Load Balancing

Routers

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

Router + Resizer

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

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

...or from config

```
akka.actor.deployment {
  /greeter {
    router = round-robin
    nr-of-instances = 5
  }
}
```

…or from config

```
akka.actor.deployment {
  /greeter {
    router = round-robin
    resizer {
      lower-bound = 2
      upper-bound = 15
    }
  }
}
```

3. BECOME

3. BECOME

- BECOME - dynamically redefines Actor's behavior

3. BECOME

- BECOME - dynamically redefines Actor's behavior
- Triggered reactively by receive of message

3. BECOME

- BECOME - dynamically redefines Actor's behavior
- Triggered reactively by receive of message
- In a type system analogy it is as if the object changed type - changed interface, protocol & implementation

3. BECOME

- BECOME - dynamically redefines Actor's behavior
- Triggered reactively by receive of message
- In a type system analogy it is as if the object changed type - changed interface, protocol & implementation
- Will now react differently to the messages it receives

3. BECOME

- BECOME - dynamically redefines Actor's behavior
- Triggered reactively by receive of message
- In a type system analogy it is as if the object changed type - changed interface, protocol & implementation
- Will now react differently to the messages it receives
- Behaviors are stacked & can be pushed and popped

Why would I want to do that?

Why would I want to do that?

- Let a highly contended Actor adaptively transform itself into an Actor Pool or a Router

Why would I want to do that?

- Let a highly contended Actor adaptively transform itself into an Actor Pool or a Router
- Implement an FSM (Finite State Machine)

Why would I want to do that?

- Let a highly contended Actor adaptively transform itself into an Actor Pool or a Router
- Implement an FSM (Finite State Machine)
- Implement graceful degradation

Why would I want to do that?

- Let a highly contended Actor adaptively transform itself into an Actor Pool or a Router
- Implement an FSM (Finite State Machine)
- Implement graceful degradation
- Spawn up (empty) generic Worker processes that can become whatever the Master currently needs

Why would I want to do that?

- Let a highly contended Actor adaptively transform itself into an Actor Pool or a Router
- Implement an FSM (Finite State Machine)
- Implement graceful degradation
- Spawn up (empty) generic Worker processes that can become whatever the Master currently needs
- Other: Use your imagination!

Why would I want to do that?

- Let a highly contended Actor adaptively transform itself into an Actor Pool or a Router
- Implement an FSM (Finite State Machine)
- Implement graceful degradation
- Spawn up (empty) generic Worker processes that can become whatever the Master currently needs
- Other: Use your imagination!
- Very useful once you get the used to it

become

```
def expectHello: Receive = {
  case Hello(name) =>
    sender ! “Hello there ” + name
    context.become(expectGoodbye)
  case Goodbye(_) => sender ! “What?”
}

def expectGoodbye: Receive = {
  case Hello(_) => sender ! “Not now”
  case Goodbye(_) =>
    sender ! “See ya later dude”
    context.become(expectHello)
}

def receive = expectHello
```

Failure Recovery in Java/C/C# etc.

Failure Recovery in Java/C/C# etc.

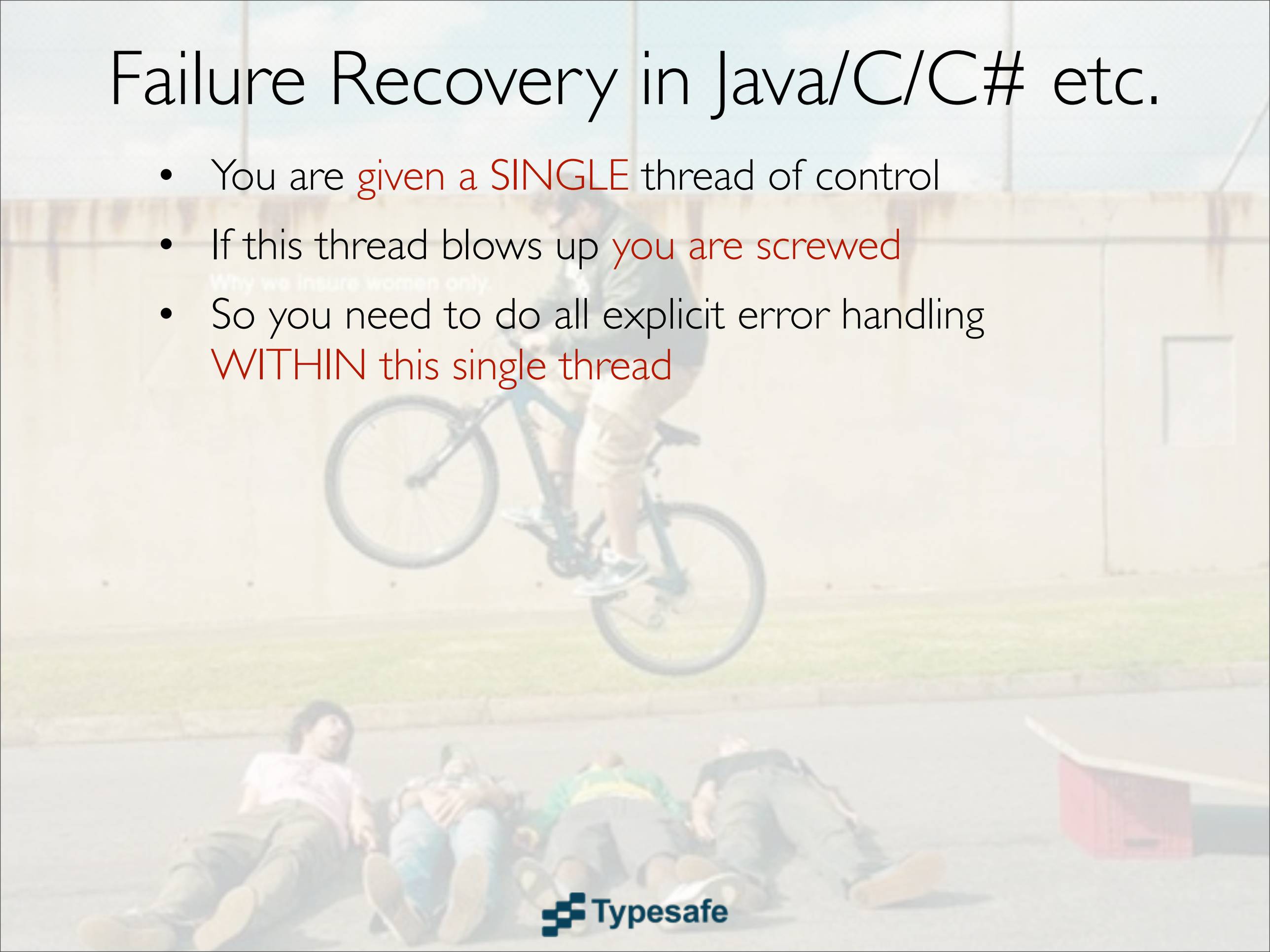
- You are given a SINGLE thread of control

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread



Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread
- To make things worse - errors do not propagate between threads so there is NO WAY OF EVEN FINDING OUT that something have failed

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread
- To make things worse - errors do not propagate between threads so there is NO WAY OF EVEN FINDING OUT that something have failed
- This leads to DEFENSIVE programming with:

Typesafe

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread
- To make things worse - errors do not propagate between threads so there is NO WAY OF EVEN FINDING OUT that something have failed
- This leads to DEFENSIVE programming with:
  - Error handling TANGLED with business logic

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread
- To make things worse - errors do not propagate between threads so there is NO WAY OF EVEN FINDING OUT that something have failed
- This leads to DEFENSIVE programming with:
  - Error handling TANGLED with business logic
  - SCATTERED all over the code base

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread
- To make things worse - errors do not propagate between threads so there is NO WAY OF EVEN FINDING OUT that something have failed
- This leads to DEFENSIVE programming with:
  - Error handling TANGLED with business logic
  - SCATTERED all over the code base

We can do better than this!!!

Typesafe

Just

LET IT CRASH

Typesafe

akka

Typesafe

akka

4. SUPERVISE

4. SUPERVISE

- SUPERVISE - manage another Actor's failures

4. SUPERVISE

- SUPERVISE - manage another Actor's failures
- Error handling in actors is handle by letting Actors monitor (supervise) each other for failure

4. SUPERVISE

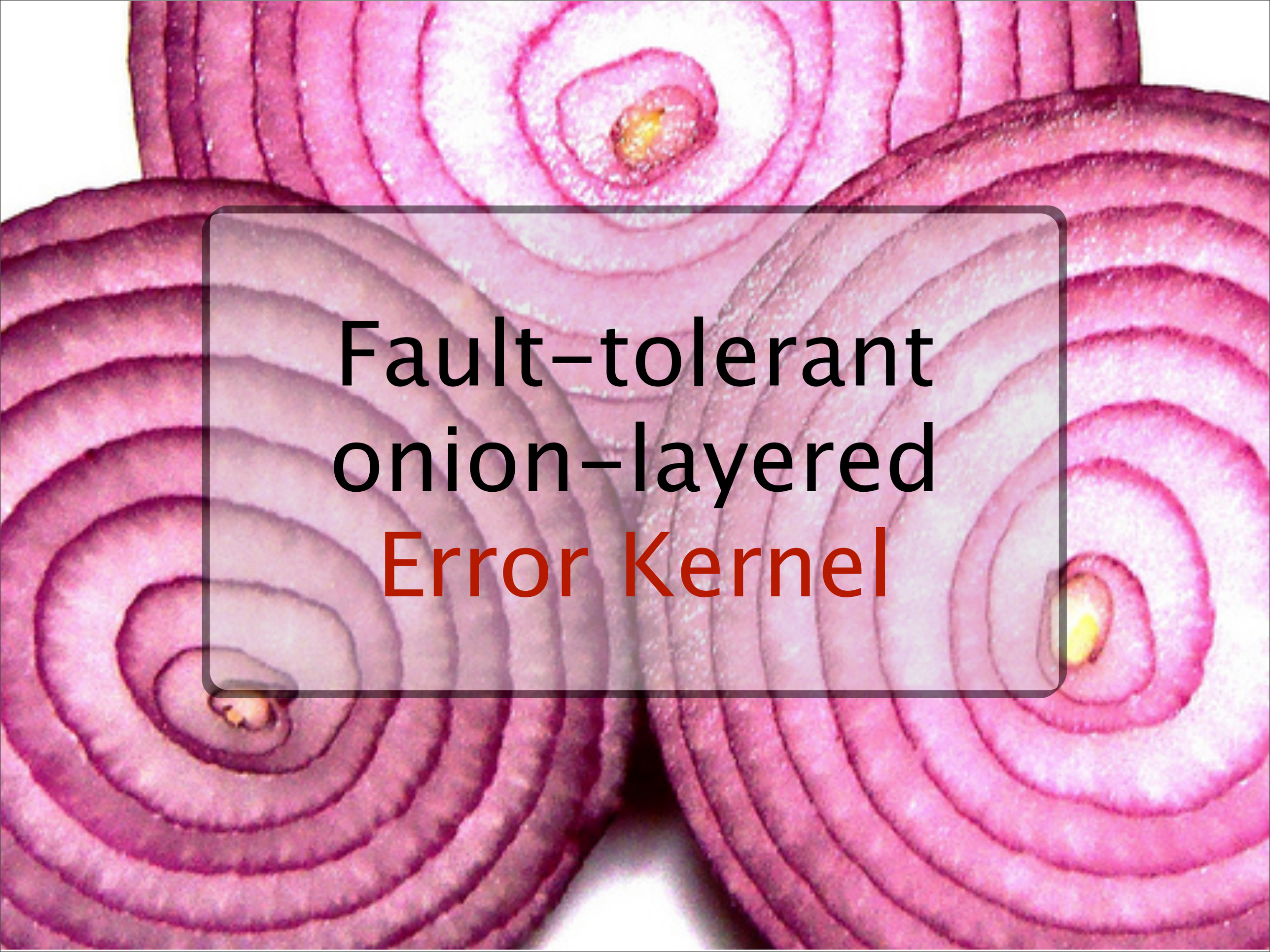
- SUPERVISE - manage another Actor's failures
- Error handling in actors is handle by letting Actors monitor (supervise) each other for failure

- This means that if an Actor crashes, a notification will be sent to his supervisor, who can react upon the failure

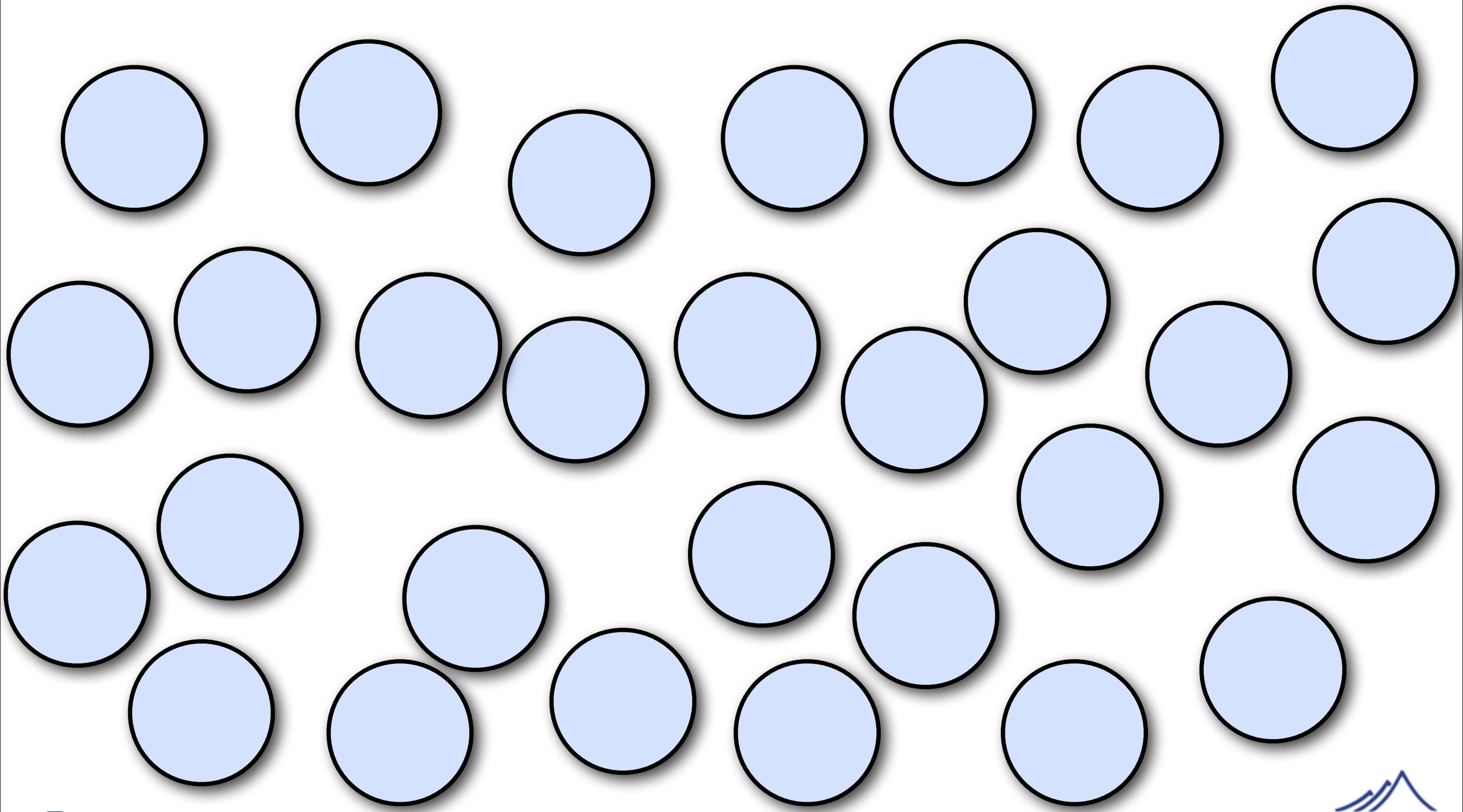
4. SUPERVISE

- SUPERVISE - manage another Actor's failures
- Error handling in actors is handle by letting Actors monitor (supervise) each other for failure

- This means that if an Actor crashes, a notification will be sent to his supervisor, who can react upon the failure
- This provides clean separation of processing and error handling


Fault-tolerant
onion-layered
Error Kernel

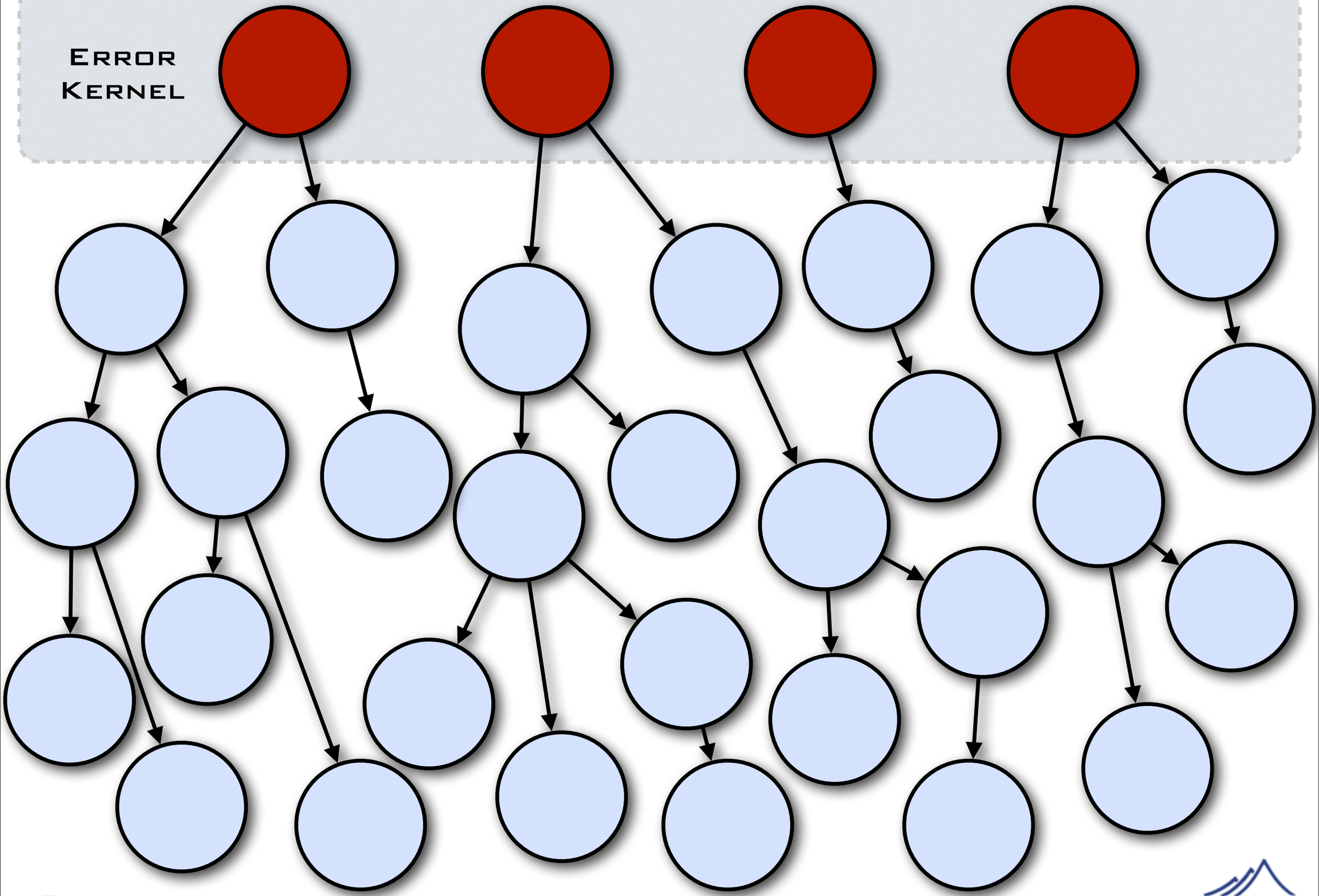
Error Kernel



Typesafe

akka

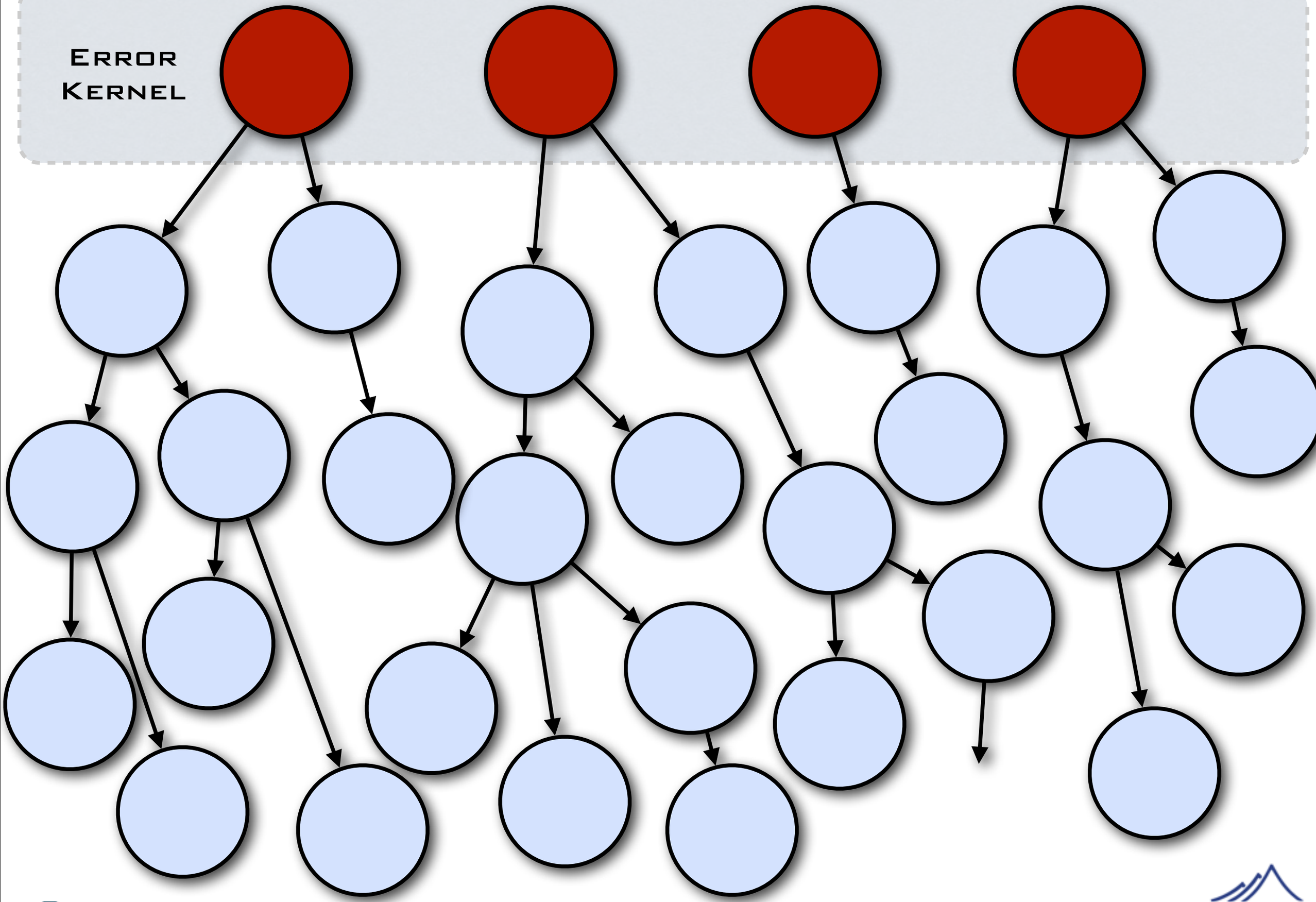
Error Kernel



Typesafe

akka

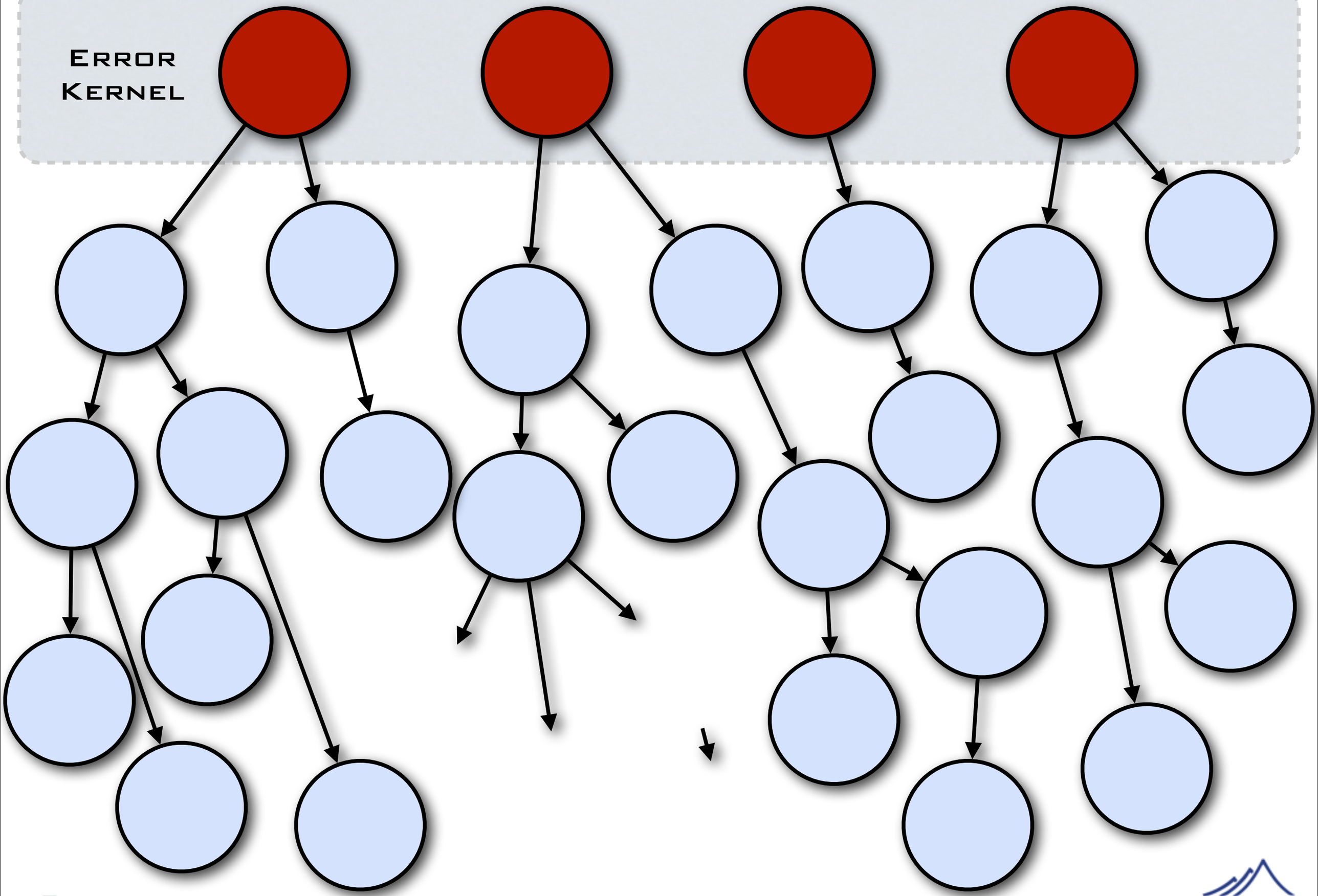
Error Kernel



Typesafe

akka

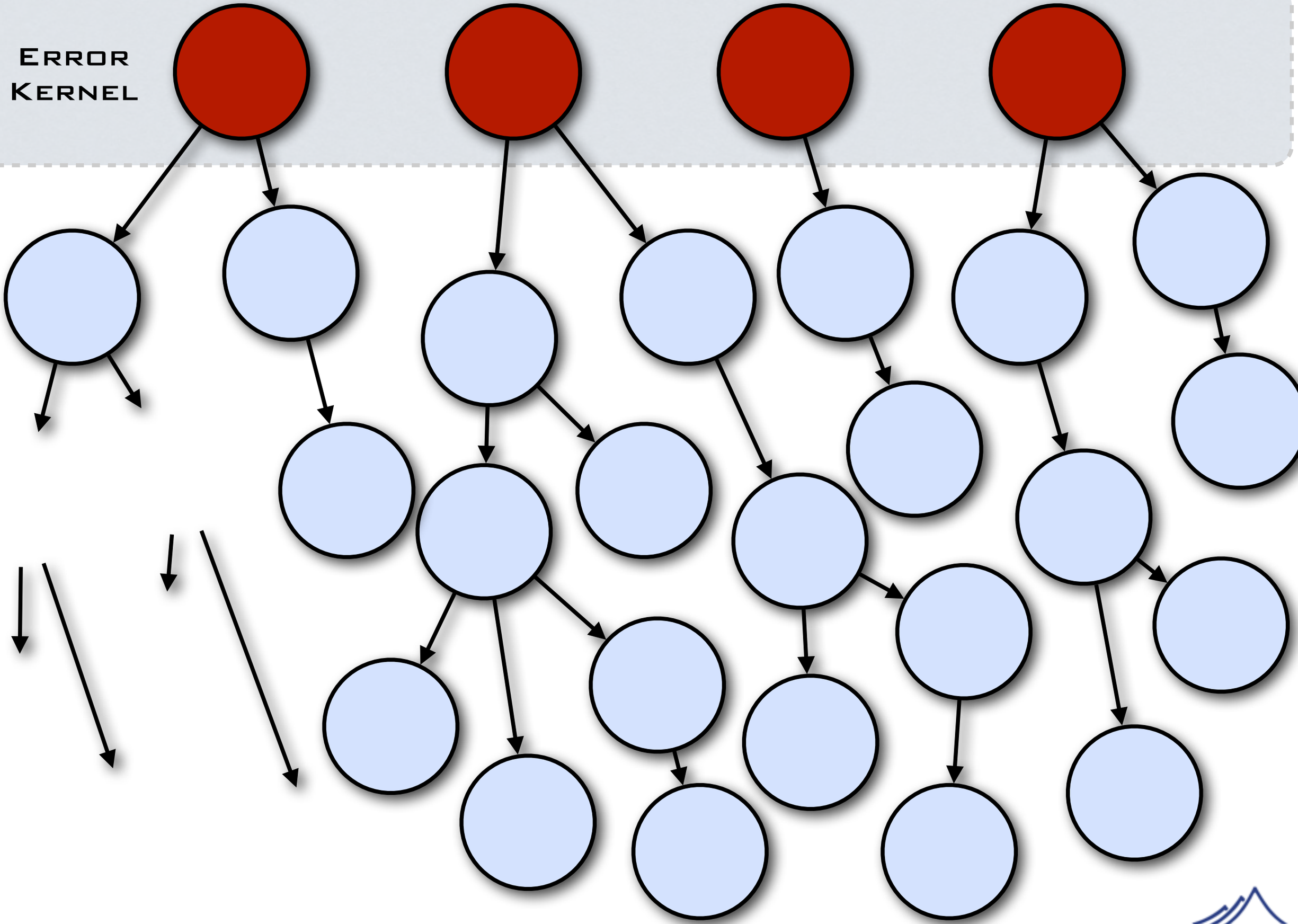
Error Kernel



Typesafe

akka

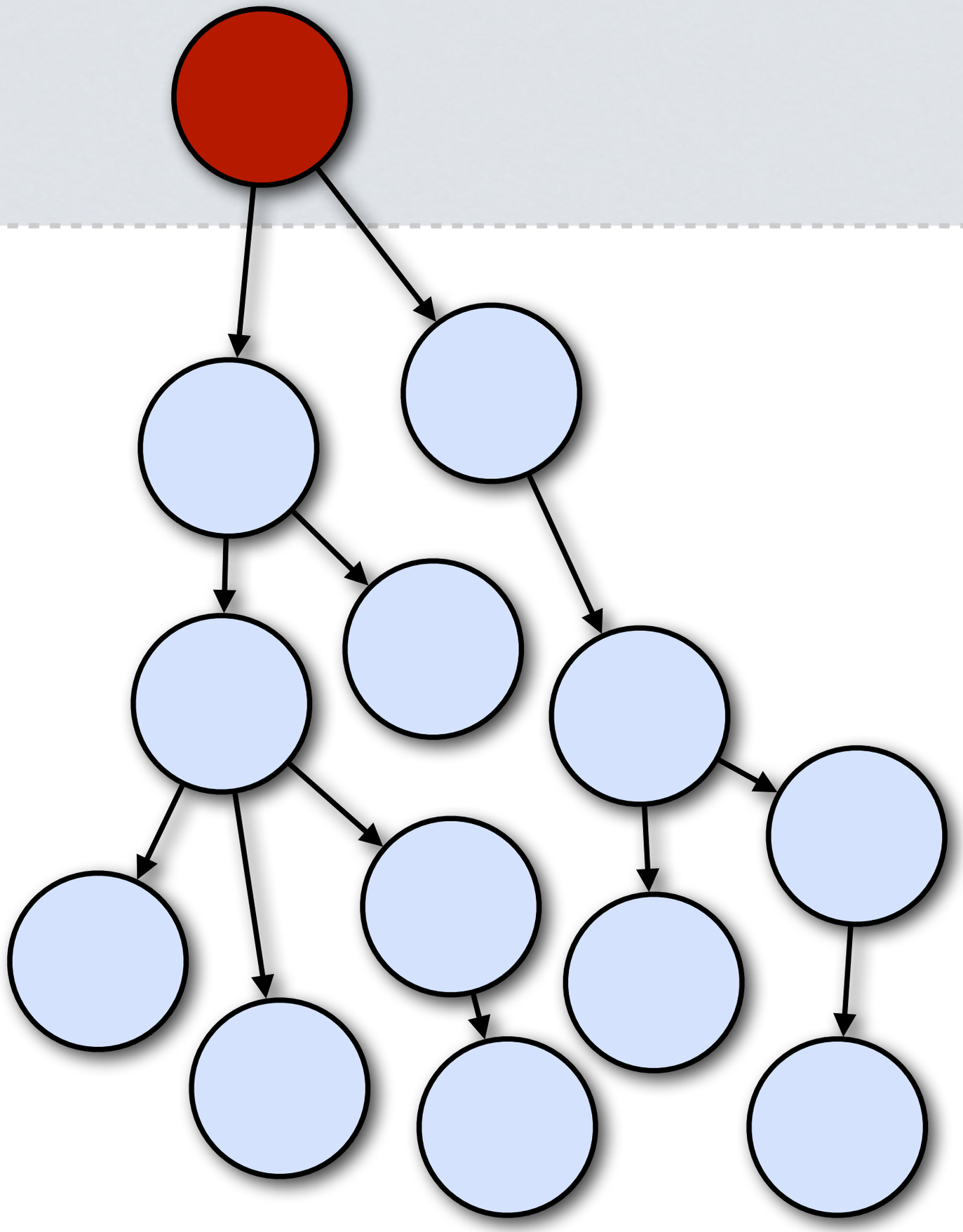
Error Kernel



Typesafe

akka

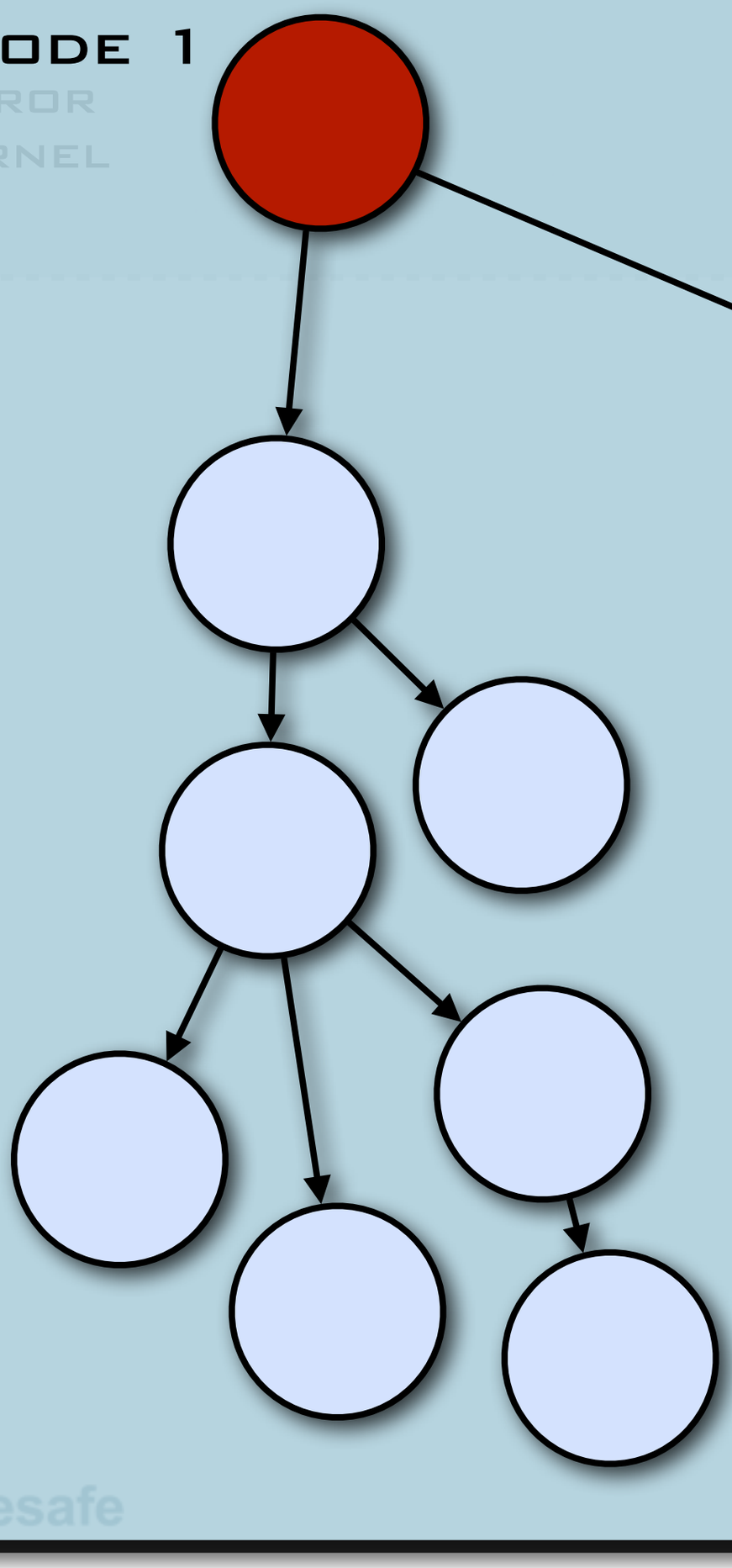
Error Kernel

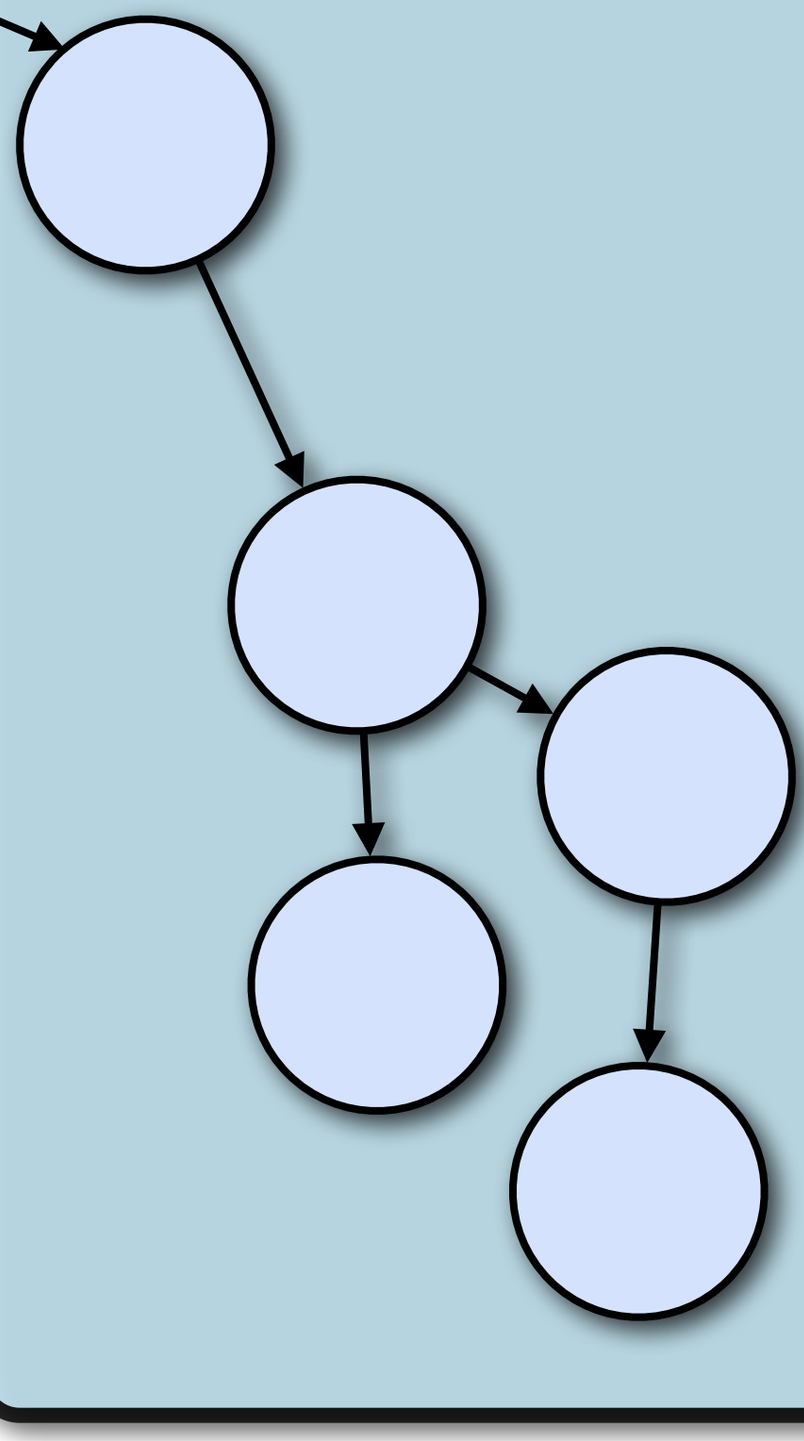


Node 1

Error Kernel

Node 2





Typesafe

akka

SUPERVISE Actor

Every single actor has a
default supervisor strategy.
Which is usually sufficient.
But it can be overridden.

SUPERVISE Actor

Every single actor has a
default supervisor strategy.
Which is usually sufficient.
But it can be overridden.

```
class Supervisor extends Actor {
  override val supervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1 minute) {
      case _: ArithmeticException  => Resume
      case _: NullPointerException => Restart
      case _: Exception            => Escalate
  }
```

Typesafe

akka

SUPERVISE Actor

```
class Supervisor extends Actor {
  override val supervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1 minute) {
      case _: ArithmeticException  => Resume
      case _: NullPointerException => Restart
      case _: Exception            => Escalate
  }

  val worker = context.actorOf(Props[Worker])

  def receive = {
    case n: Int => worker forward n
  }
}
```

Manage failure

```
class Worker extends Actor {
  ...

  override def preRestart(
    reason: Throwable, message: Option[Any]) {
    ... // clean up before restart
  }

  override def postRestart(reason: Throwable) {
    ... // init after restart
  }
}
```

Akka Camel

Typesafe

akka

Consumer

```
class Consumer1 extends Consumer {
  def endpointUri =

  def receive = {
    case msg: CamelMessage ⇒
      println("received %s" format msg.bodyAs[String])
  }
}
```

Consumer

```
class Consumer1 extends Consumer {
  def endpointUri = “file:data/input/actor”

  def receive = {
    case msg: CamelMessage ⇒
      println("received %s" format msg.bodyAs[String])
  }
}
```

Consumer

```
class Consumer1 extends Consumer {
  def endpointUri =

  def receive = {
    case msg: CamelMessage ⇒
      println("received %s" format msg.bodyAs[String])
  }
}
```

Consumer

```
class Consumer1 extends Consumer {
  def endpointUri = “jetty:http://localhost:8877/camel/default”

  def receive = {
    case msg: CamelMessage ⇒
      println("received %s" format msg.bodyAs[String])
  }
}
```

Producer

```
class OrderActor extends Actor with Producer with Oneway {
  def endpointUri = "jms:queue:Orders"
}

val system = ActorSystem("some-system")
val orderActor = system.actorOf(Props[OrderActor])

orderActor ! <order amount="42" currency="EUR" itemId="9"/>
```

Akka Cluster

Experimental module in 2.1

What's New?

- Gossip-based Cluster Membership
- Accrual Failure Detector
- Cluster DeathWatch
- Cluster-Aware Routers

Gossiping Protocol

Gossiping Protocol

Used for:

Gossiping Protocol

Used for:

- Cluster Membership

Gossiping Protocol

Used for:

- Cluster Membership
- Configuration data

Gossiping Protocol

Used for:

- Cluster Membership
- Configuration data
- Leader Determination

Gossiping Protocol

Used for:

- Cluster Membership
- Configuration data
- Leader Determination
- *Partitioning data*

Gossiping Protocol

Used for:

- Cluster Membership
- Configuration data
- Leader Determination
- *Partitioning data*
- *Naming Service*

Enable clustering

```
akka {
  actor {
    provider = "akka.cluster.ClusterActorRefProvider"
    ...
  }

  extensions = ["akka.cluster.Cluster"]

  cluster {
    seed-nodes = [
      "akka://ClusterSystem@127.0.0.1:2551",
      "akka://ClusterSystem@127.0.0.1:2552"
    ]

    auto-down = on
  }
}
```

Configure a clustered router

```
akka.actor.deployment {
  /statsService/workerRouter {
    router = consistent-hashing
    nr-of-instances = 15

    cluster {
      enabled = on
      max-nr-of-instances-per-node = 3
      allow-local-routees = on
    }
  }
}
```

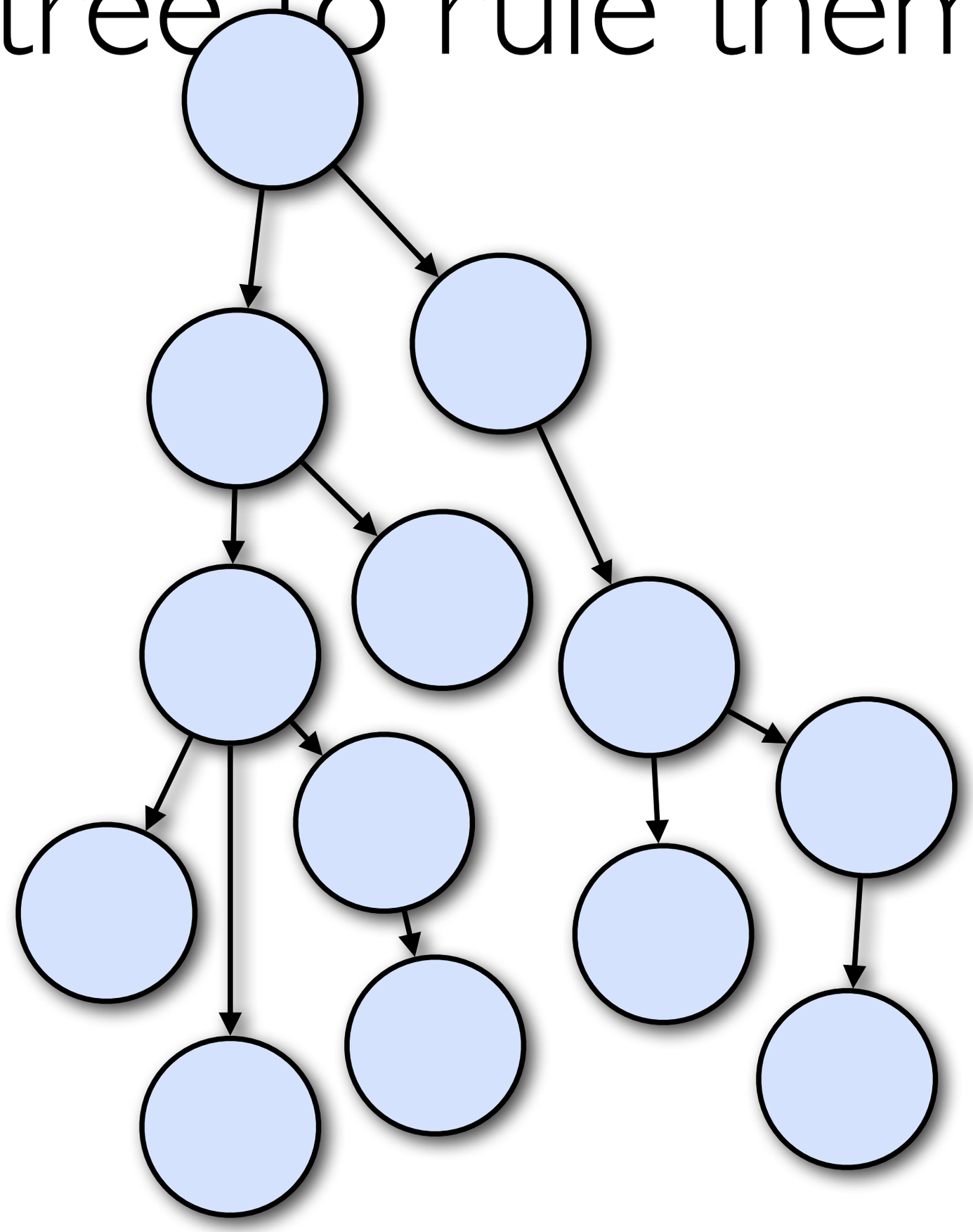
Rollins

... when the Cluster grows up

One tree to rule them all

- Coltrane has one Actor tree per node
- Rollins still has this as physical substrate
- Cluster tree is mapped to local sub-trees

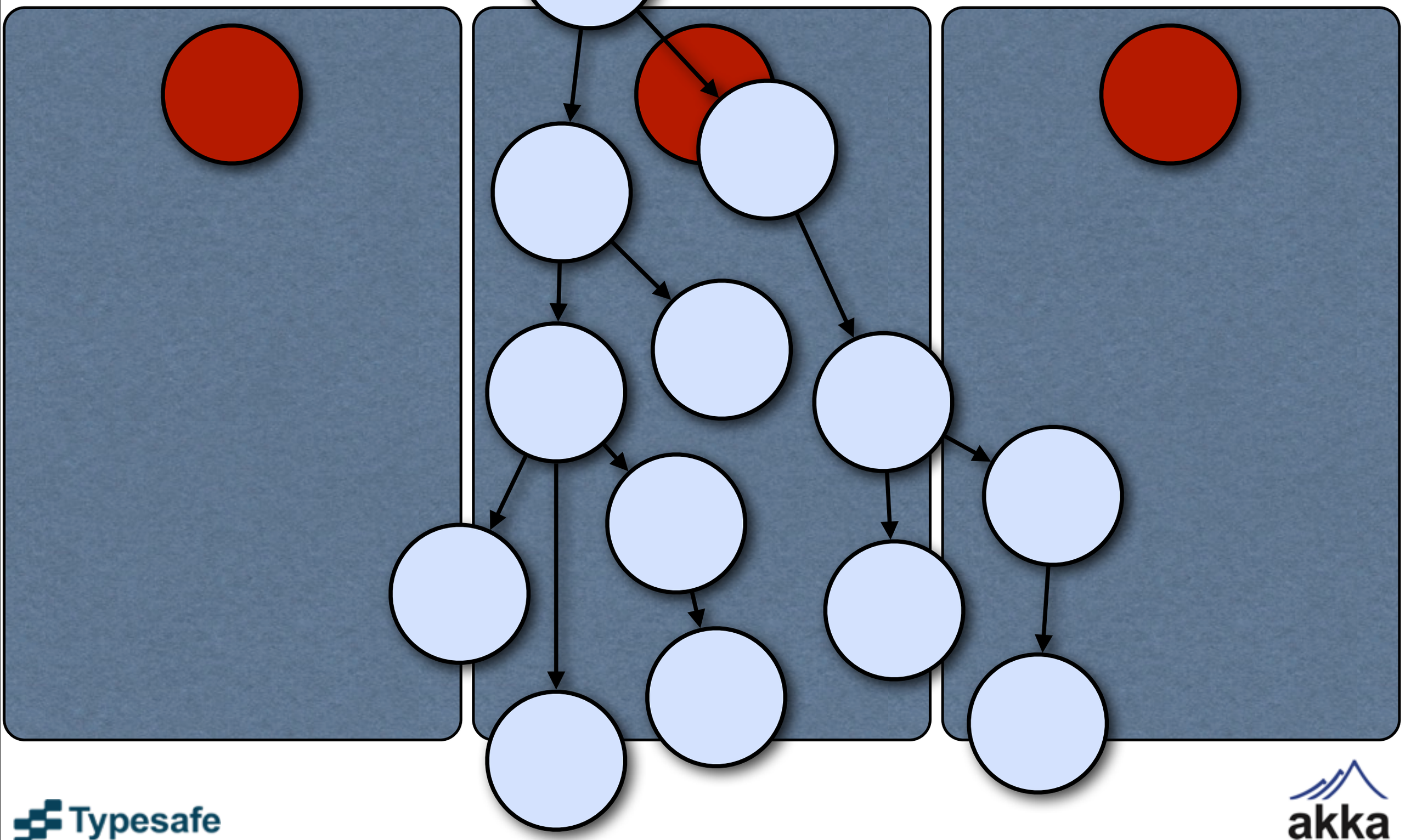
One tree to rule them all



Typesafe

akka

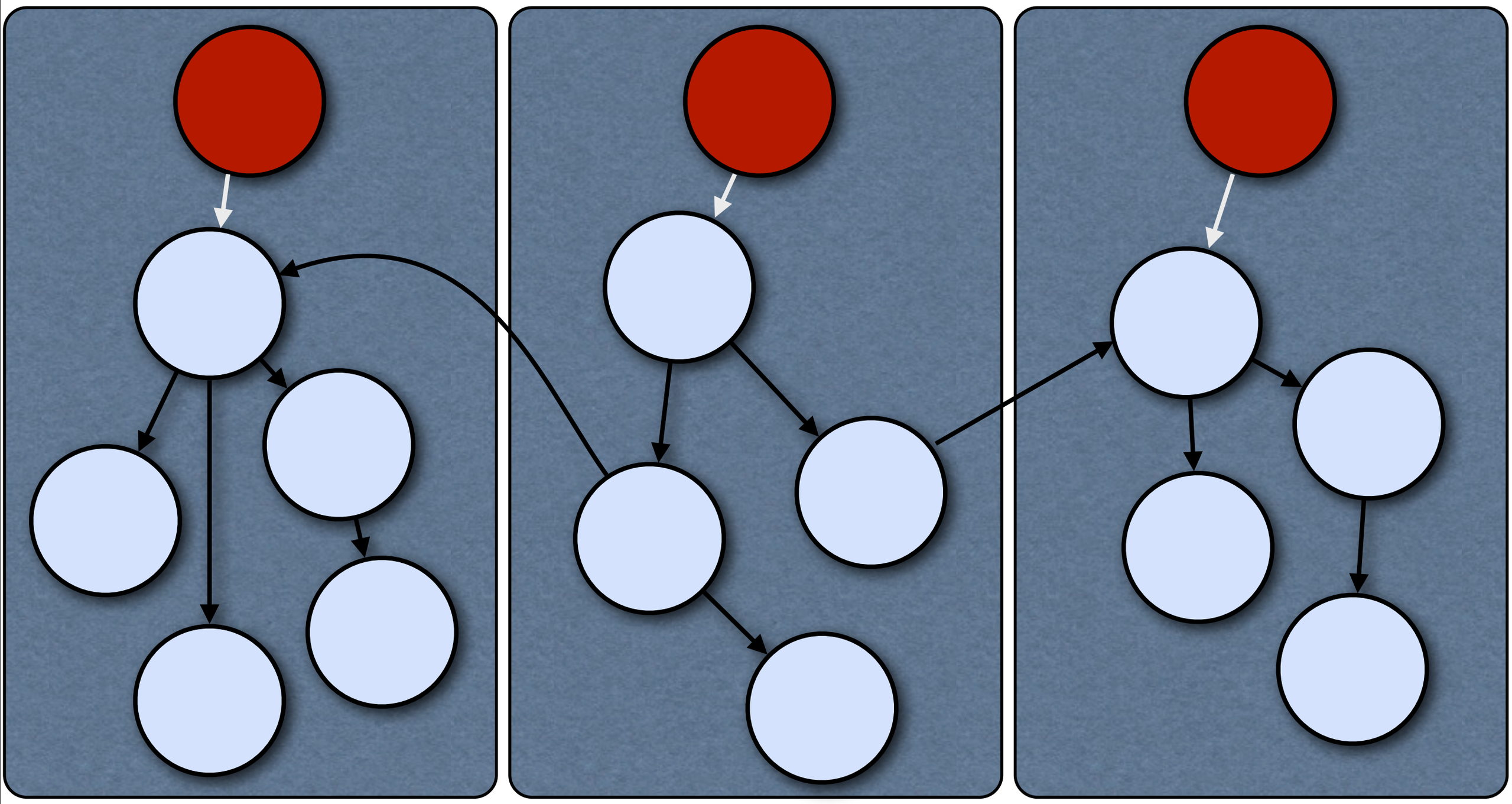
One tree to rule them all



Typesafe

akka

One tree to rule them all



Typesafe

akka

The Magic Sauce

- User code only sees `cluster://...` names
- ActorRef becomes repointable
  - local (current ActorCell)
  - remote (new RemoteActorCell)
- Can now move actors around transparently
  - Actor encapsulation makes it possible

What does this enable?

- Actor migration
- Actor replication
- Automatic cluster partitioning
  - later also based on runtime metrics
- Node fail-over
  - first for stateless actors
  - later for stateful actors using event sourcing

⇛ *Fault Tolerance & Distribution*

Cluster Specification

doc.akka.io/docs/akka/snapshot/cluster/cluster.html

Cluster User Guide

doc.akka.io/docs/akka/snapshot/cluster/cluster-usage.html

Cluster Code

github.com/akka/akka/tree/master/akka-cluster

Typesafe

akka

...we have much **much** more

...we have much **much** more

Durable Mailboxes

FSM

TestKit

Pub/Sub

EventBus

TypedActor

IO

Pooling

Microkernel

Dataflow

ZeroMQ

SLF4J

Transactors

Agents

Extensions

get it and learn more

http://akka.io

http://letitcrash.com

http://typesafe.com

EØF

Akka

Typesafe

akka

Case Studies

- Royal Netherlands Marechaussee
- Gilt

Typesafe

akka

RNM

- Main function is to combat
  - illegal residence, smuggling, trafficking, etc.
  - cross-border crimes
- Challenges
  - collect data for traffic profiling
  - observe vehicles + select ones to be stopped
  - respond to quick alerts

RNM

- Solution
  - Akka because
    - ease of use
    - leverages existing Java investments
    - scalability and performance
- Been in production since August 1st 2012
- Further information:
  - http://www.typesafe.com/public/case-studies/DBP-case-study.pdf

Typesafe

akka

Gilt

- Online shopping of top designer labels
- Uses so called “flash sales”
- Gilt live

Gilt

- RoR => Java => Scala
  - scalability issues with RoR
  - uses modern web technology: web sockets => Play! web framework

- "Akka is key to handling traffic" - a huge % of Gilt's business is done between 12:00 PM and 12:10 PM. Gilt uses both actors and futures to handle the load.
- Read more: http://www.typesafe.com/public/case-studies/Gilt%20Live%20Case%20Study

Typesafe

akka

E0F

Case Studies

Typesafe

akka

The Typesafe Console

Typesafe

akka

About the Console

- Monitors Akka applications
- Implemented in Scala, Akka and Play
- Comes as part of the Typesafe subscription
- Aim: It should be possible to have running at *all time* in production

Architecture

Console

Your Monitored Application

Instrumented Akka

Analyze

Query

Mongo DB

Typesafe

akka

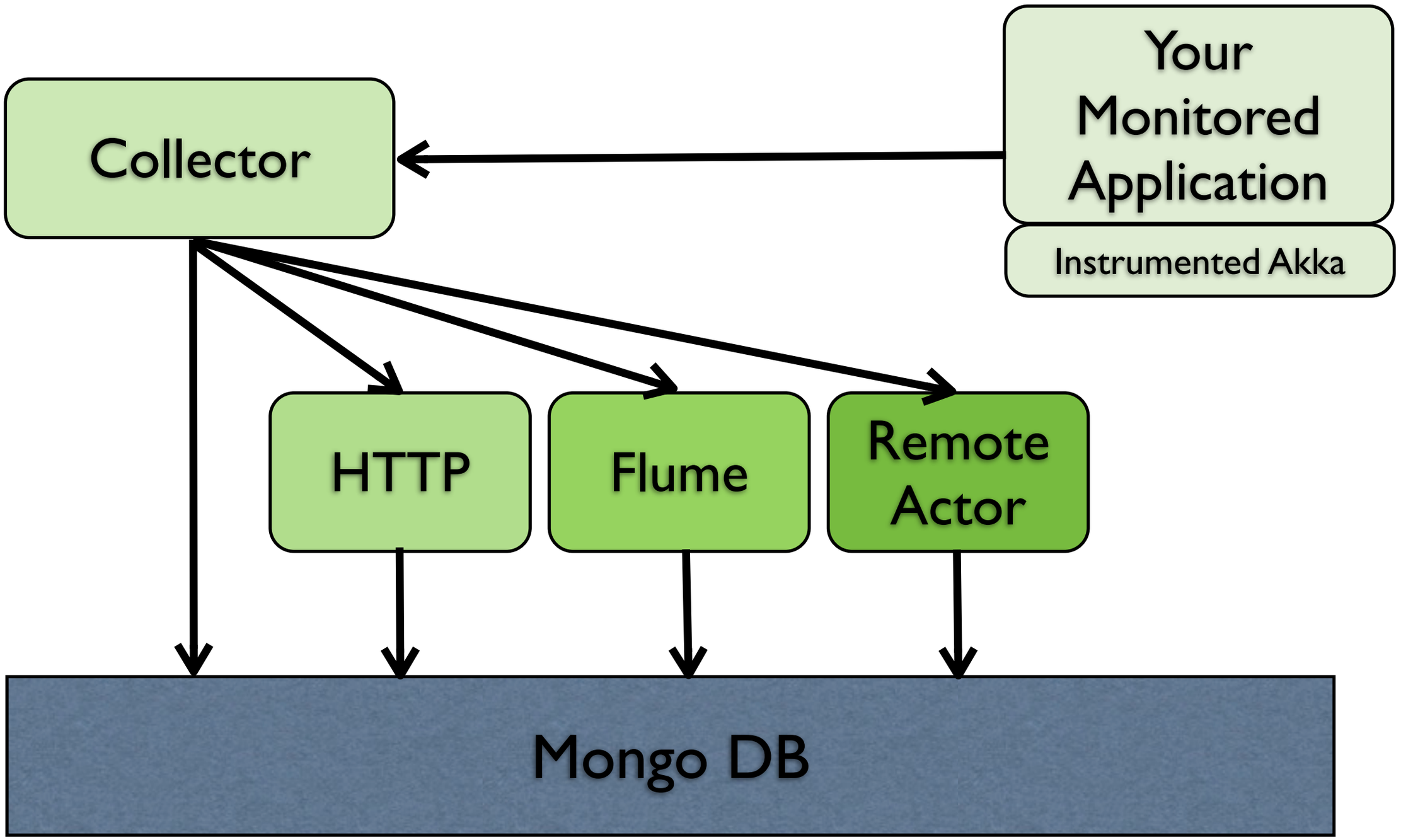
Trace Module

- Aspect Load Time Weaving
- Stores data in ThreadLocal
- Batch store via socket into offline storage
- Tracing works for
  - Within one ActorSystem and between ActorSystems
  - Between remote ActorSystems
  - for all operations in Akka
- Available for Akka 1.3, 2.0.x, 2.1 (soon)

Trace Event Example

```
// JSON PAYLOAD
{
"_id" : BinData(3,"4RHh8WFt3T6JkhKx9rj2qQ=="),
"annotation" :
{ "type" : "received", "actor" : "akka://MyAS/user/ActorA"}
"trace" : BinData(3,"4RHh8ZFx2z6JkhKx9rj2qQ=="),
"parent" : BinData(3,"4RHh8UAf3T6JkhKx9rj2qQ=="),
"sampled" : 1,
"node" : "MyNode",
"actorSystem" : "MyActorSystem",
"host" : "10.0.1.234",
"timestamp" : NumberLong("1346248552246"),
"nanoTime" : NumberLong("1346248552246712000")
 // ...
}
```

Under the hood



Trace API

- Collect data in "hot path"
  - Tracer(system).markStart("myPath")
  - Tracer(system).markEnd("myPath")
- Collect application specific data
  - Tracer(system).record("login","user1")

Deployment

- Text based configuration = DevOps :-)
- Simple Setup
  - JVM1 : Your application
  - JVM2 : Query and Analyze modules
  - JVM3 : Console
  - One MongoDB instance

Deployment

- Advanced setup
  - JVM 1-m : Your application
  - JVM m-n : Analyze
  - JVM n1 : Query
  - JVM n2 : Console
  - 2 Mongo DB instances: Data + Statistics

Demo

Thx

@h3nk3

Typesafe

akka