Above the Clouds: Introducing Akka

Jonas Bonér - Scalable Solutions
Garrick Evans - Autodesk

The problem

It is way too hard to build:

1. correct highly concurrent systems
2. truly scalable systems
3. fault-tolerant systems that self-heals

...using “state-of-the-art” tools

Introducing akka

Vision

Simpler

- Concurrency
- Scalability
- Fault-tolerance

Vision

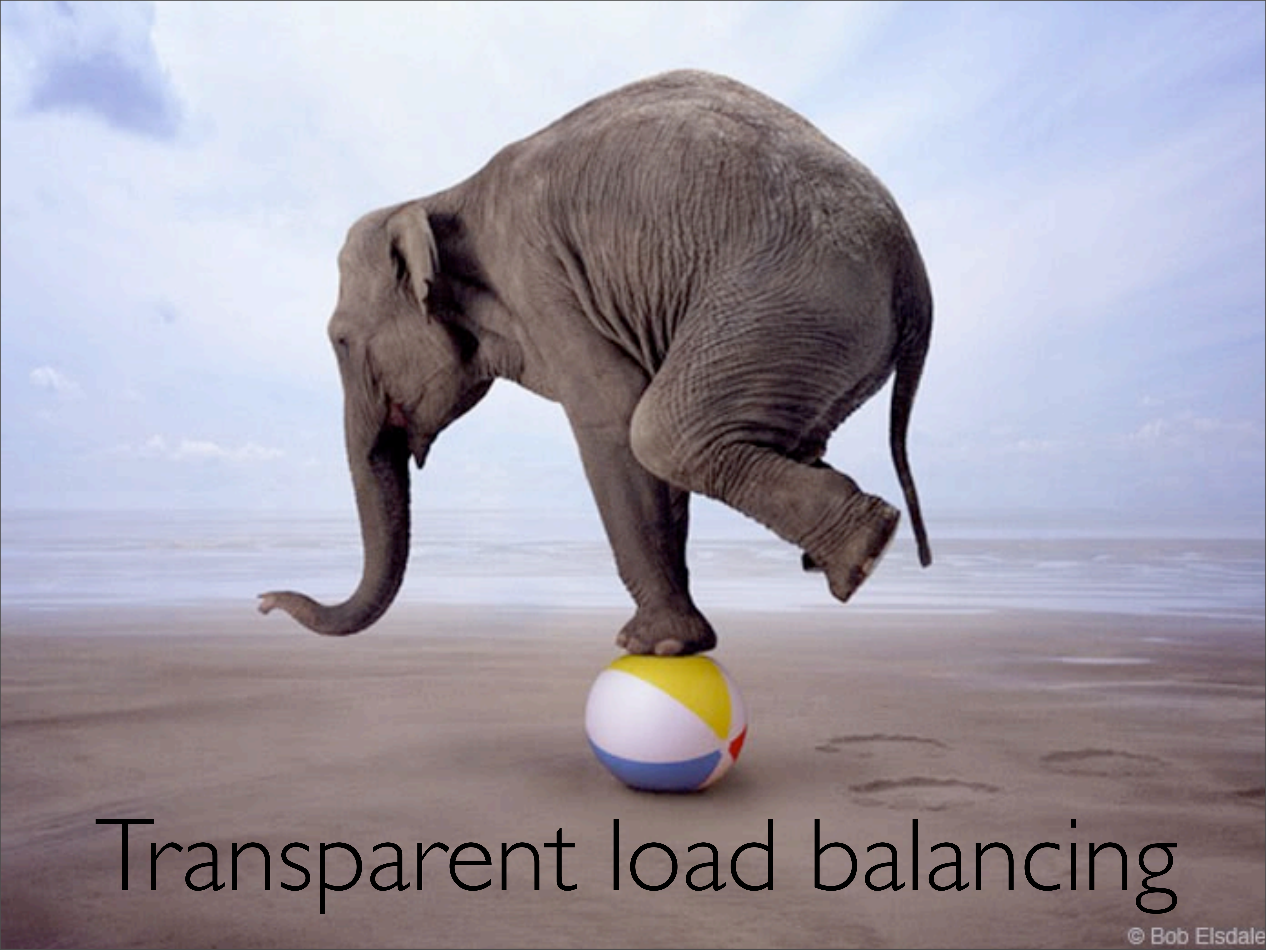
...with a single unified

- Programming model
- Runtime service

Manage system overload

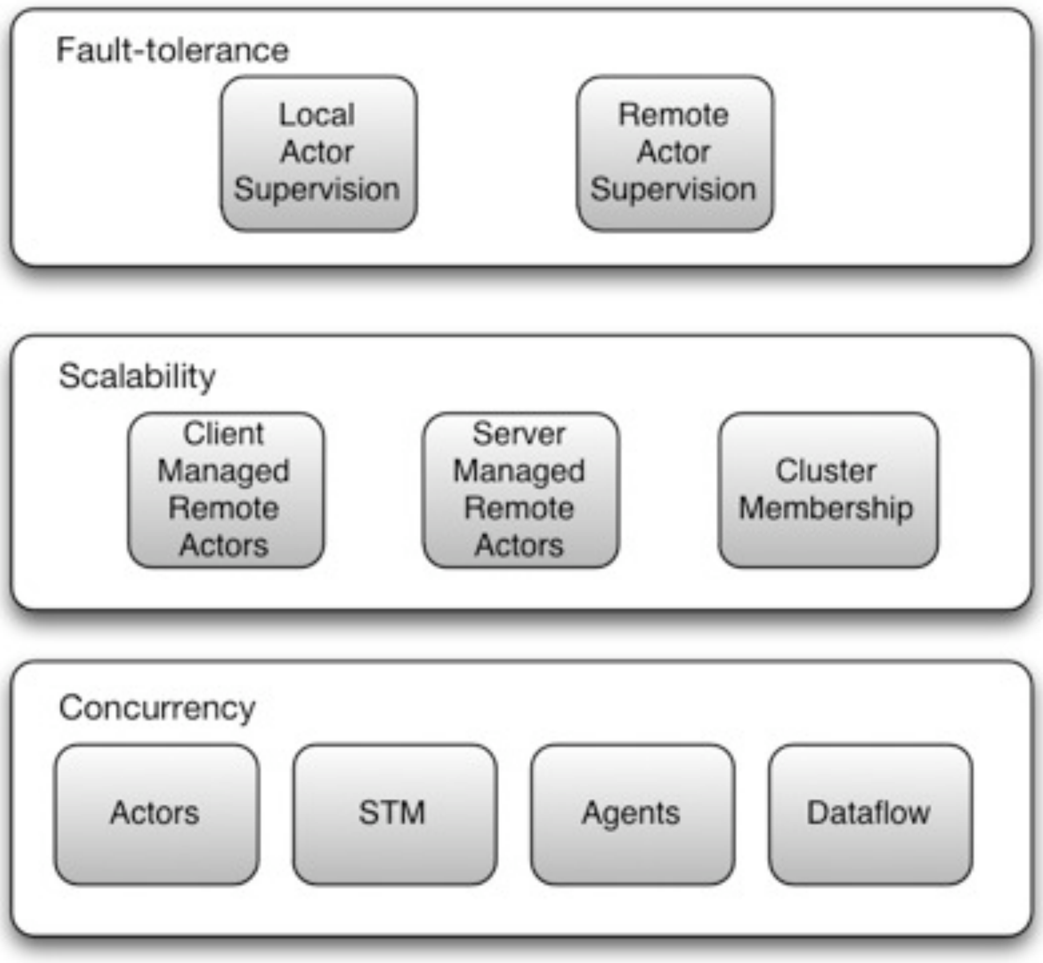
Scale up & Scale out

Replicate and distribute for fault-tolerance



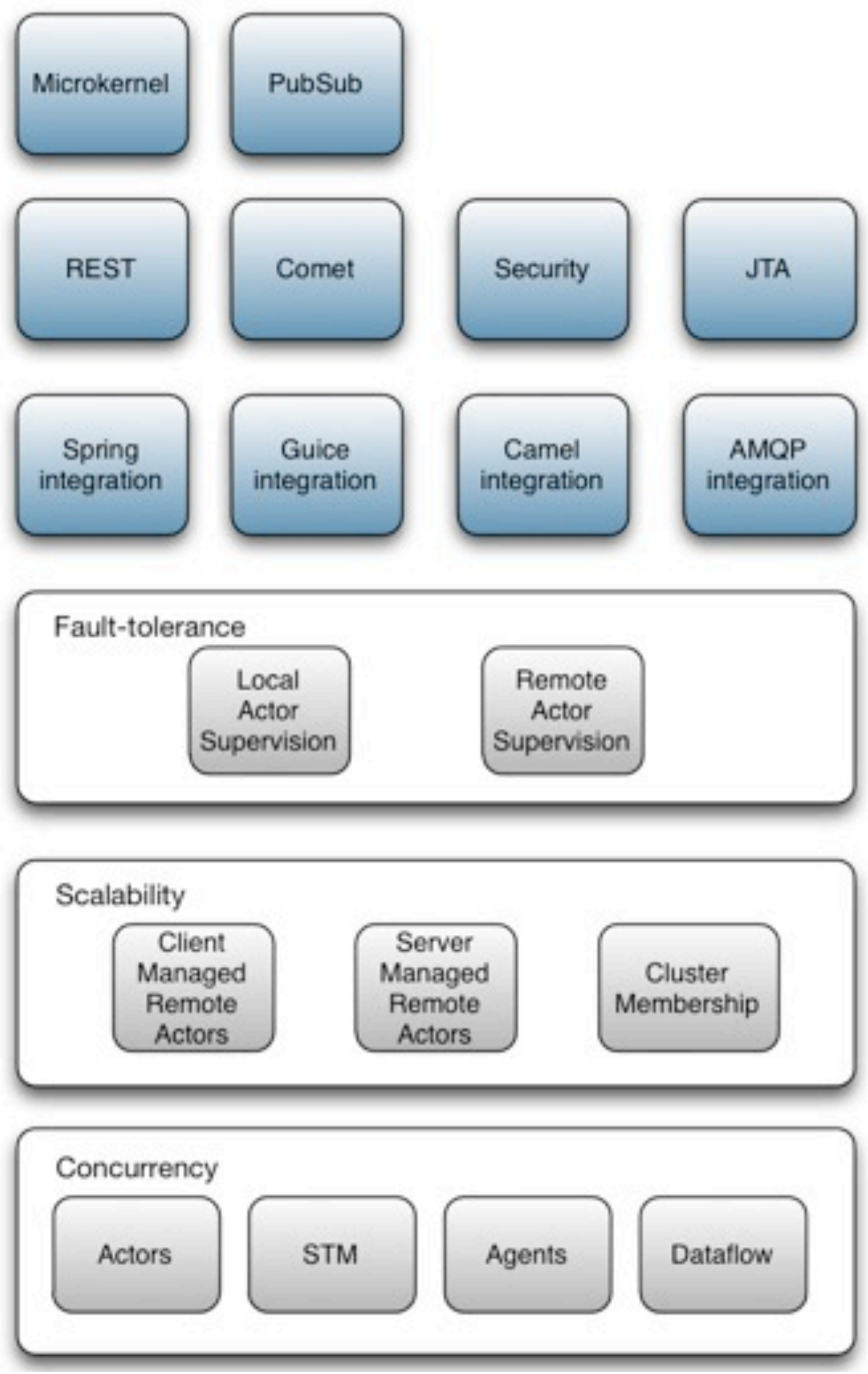
Transparent load balancing

ARCHITECTURE



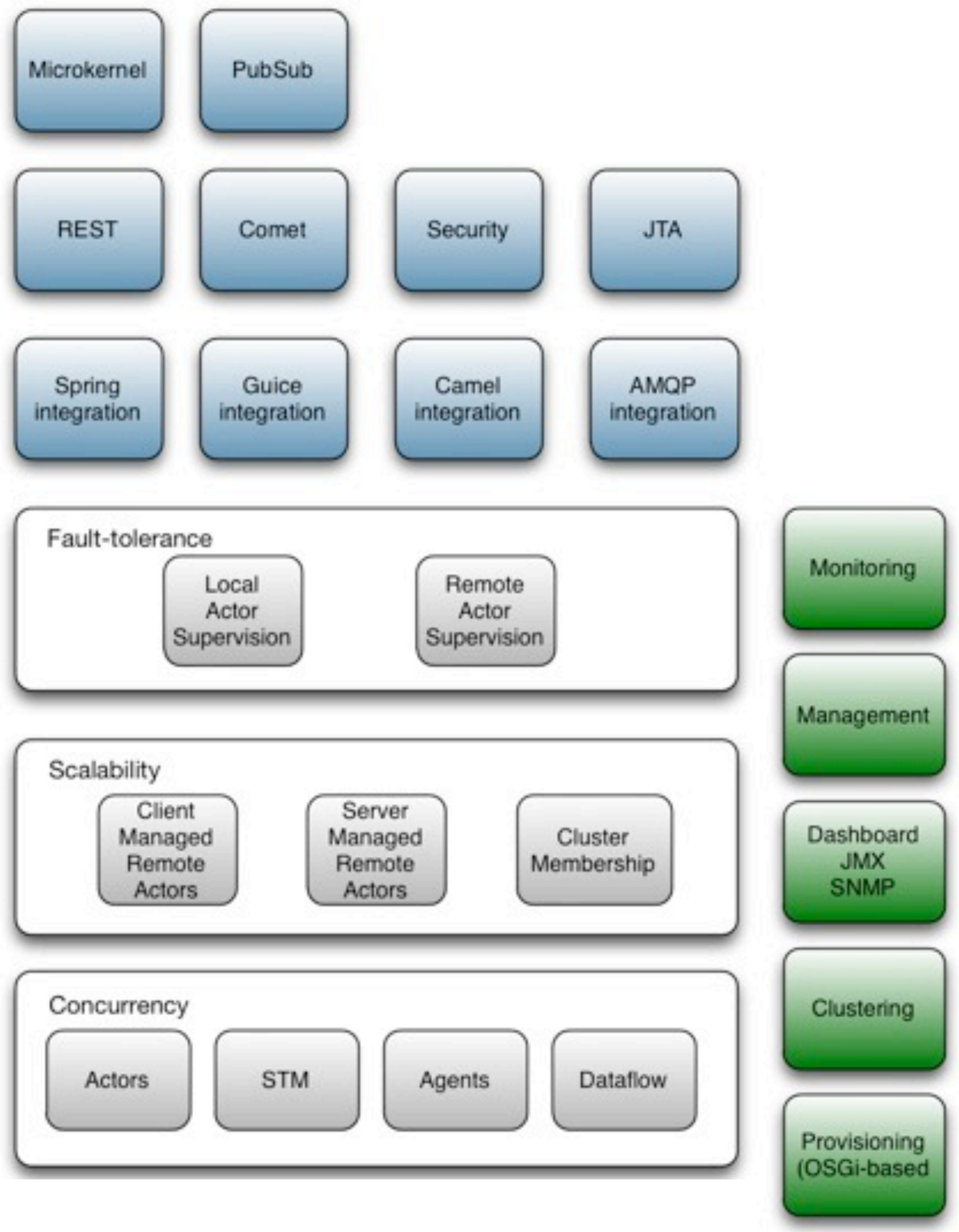
CORE SERVICES

ARCHITECTURE



ADD-ON MODULES

ARCHITECTURE



CLOUDY
AKKA

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

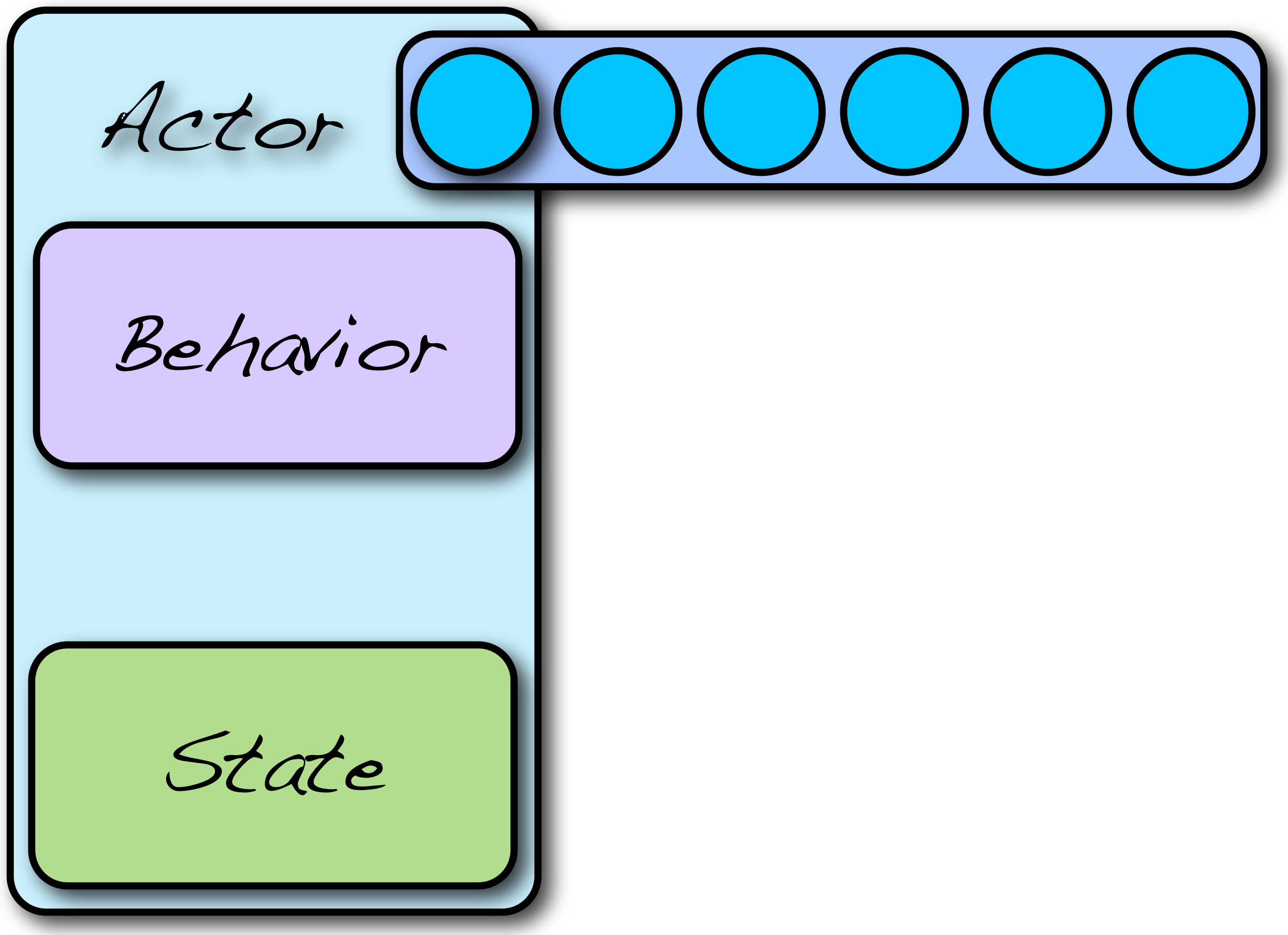
SIMULATION

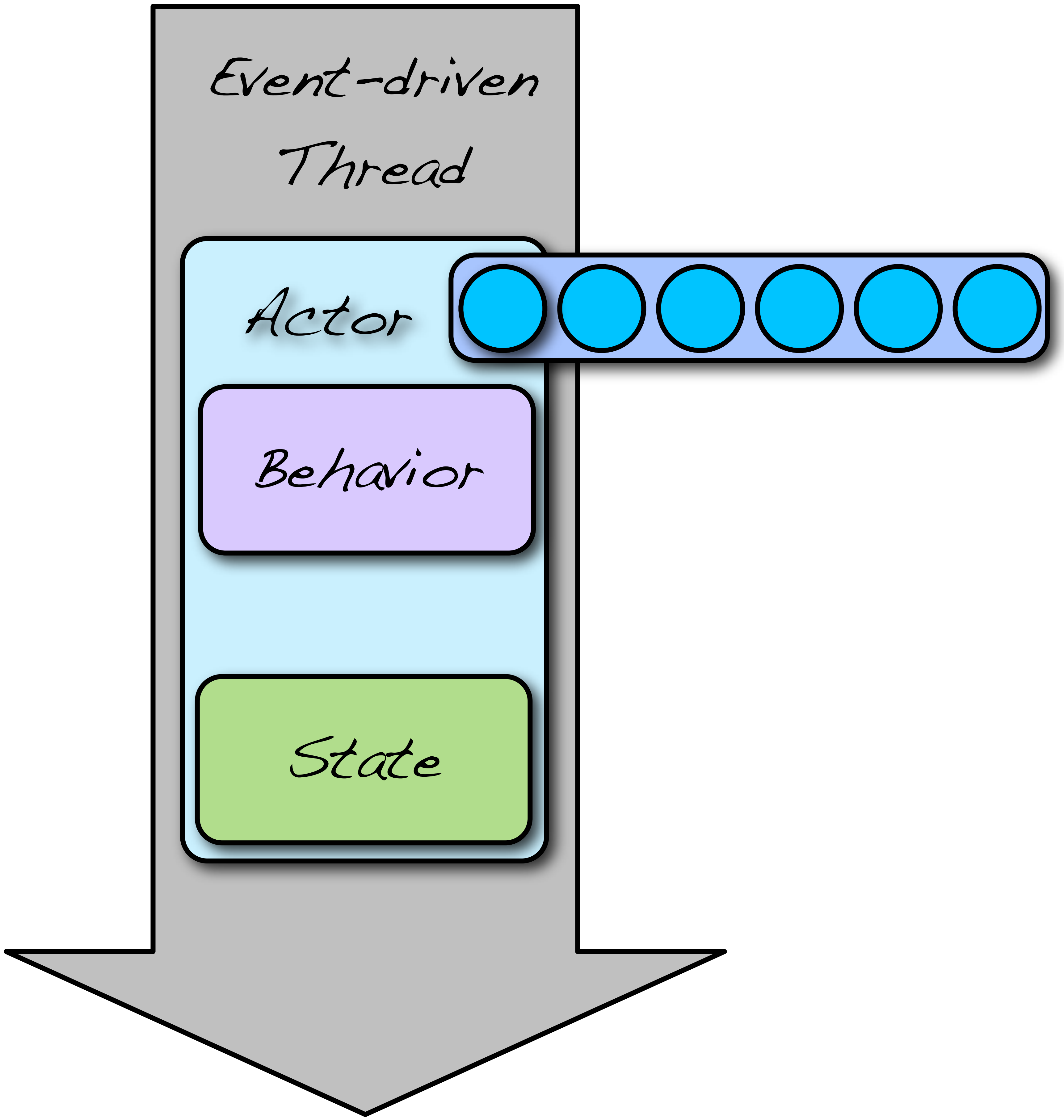
- 3D simulation engines

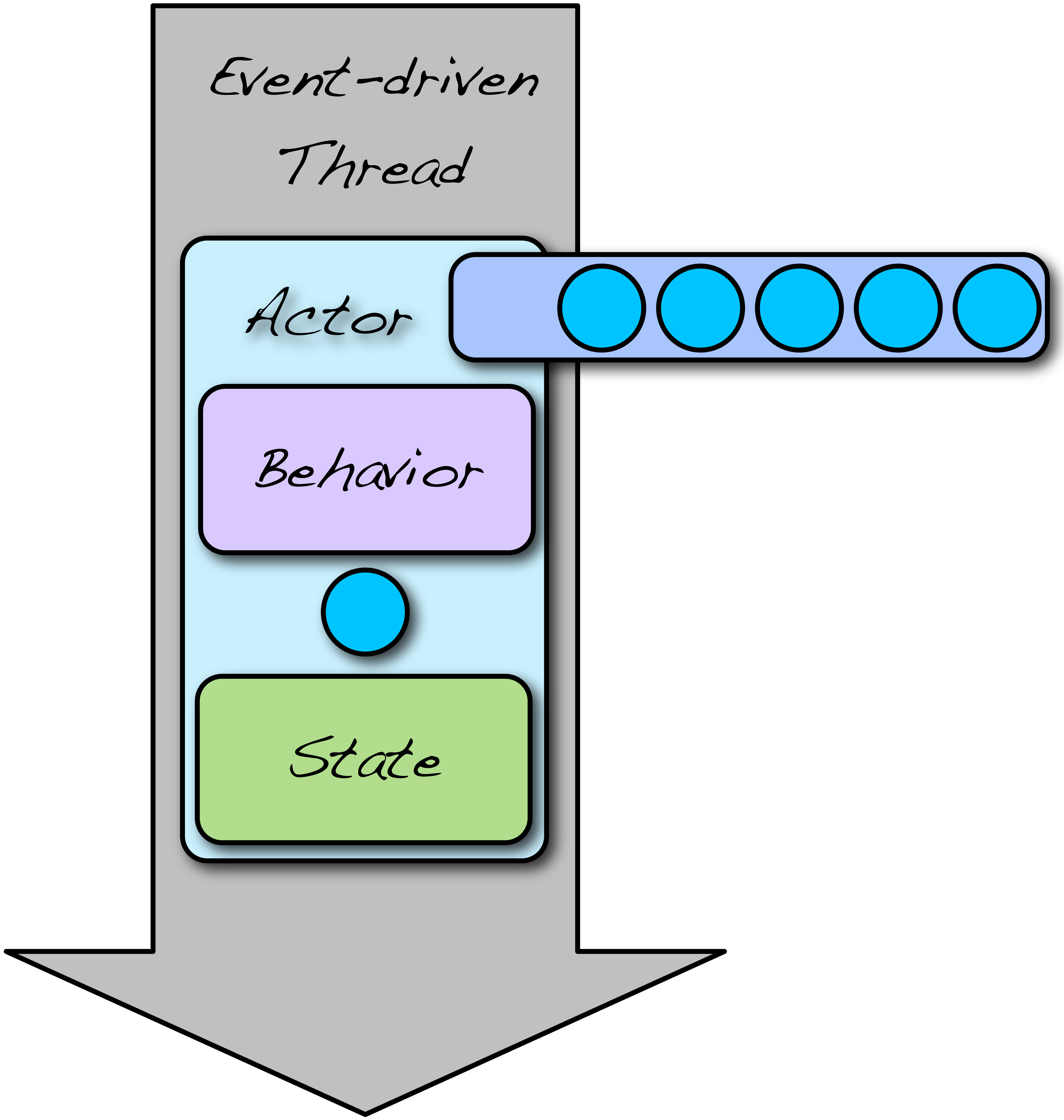
E-COMMERCE

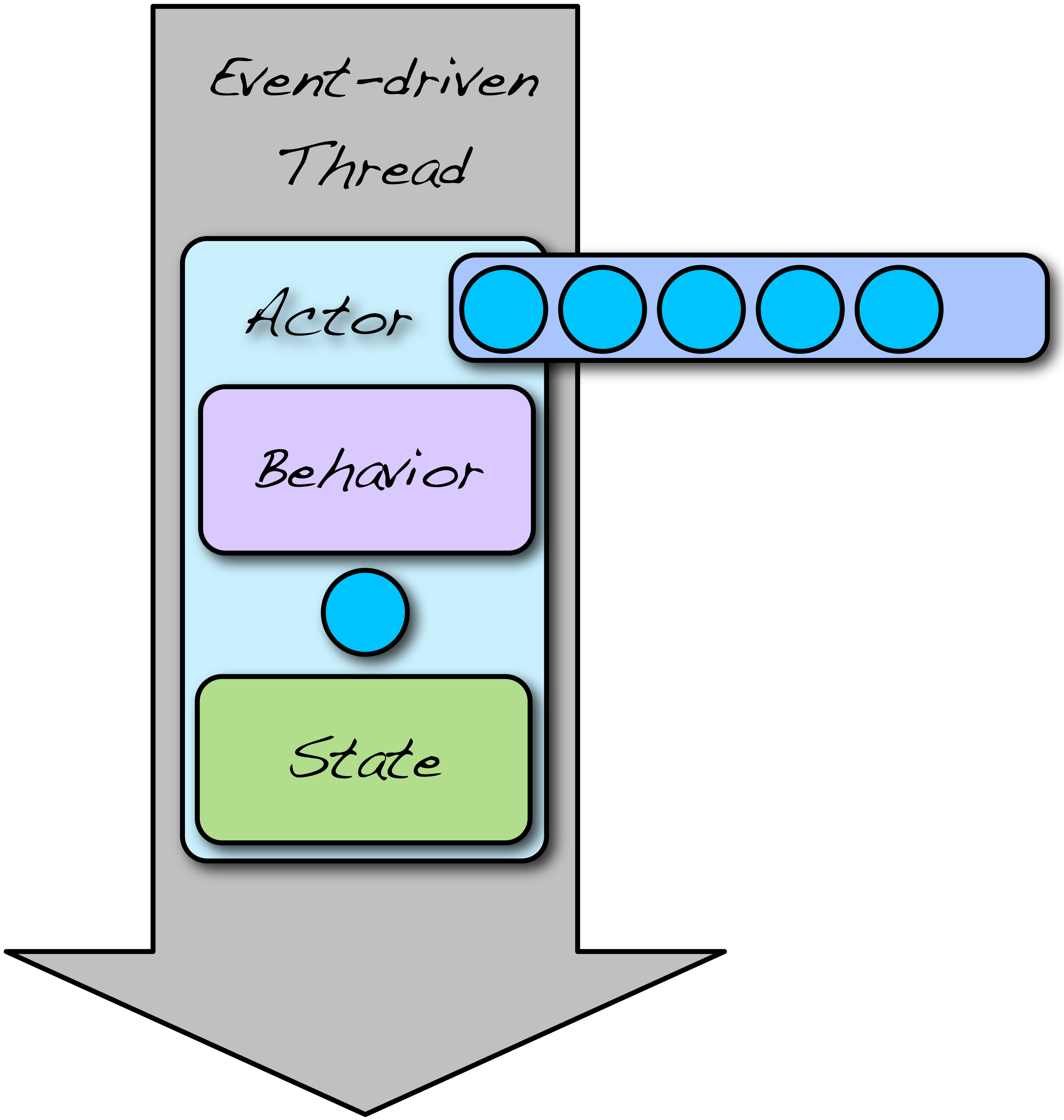
- Social media community sites

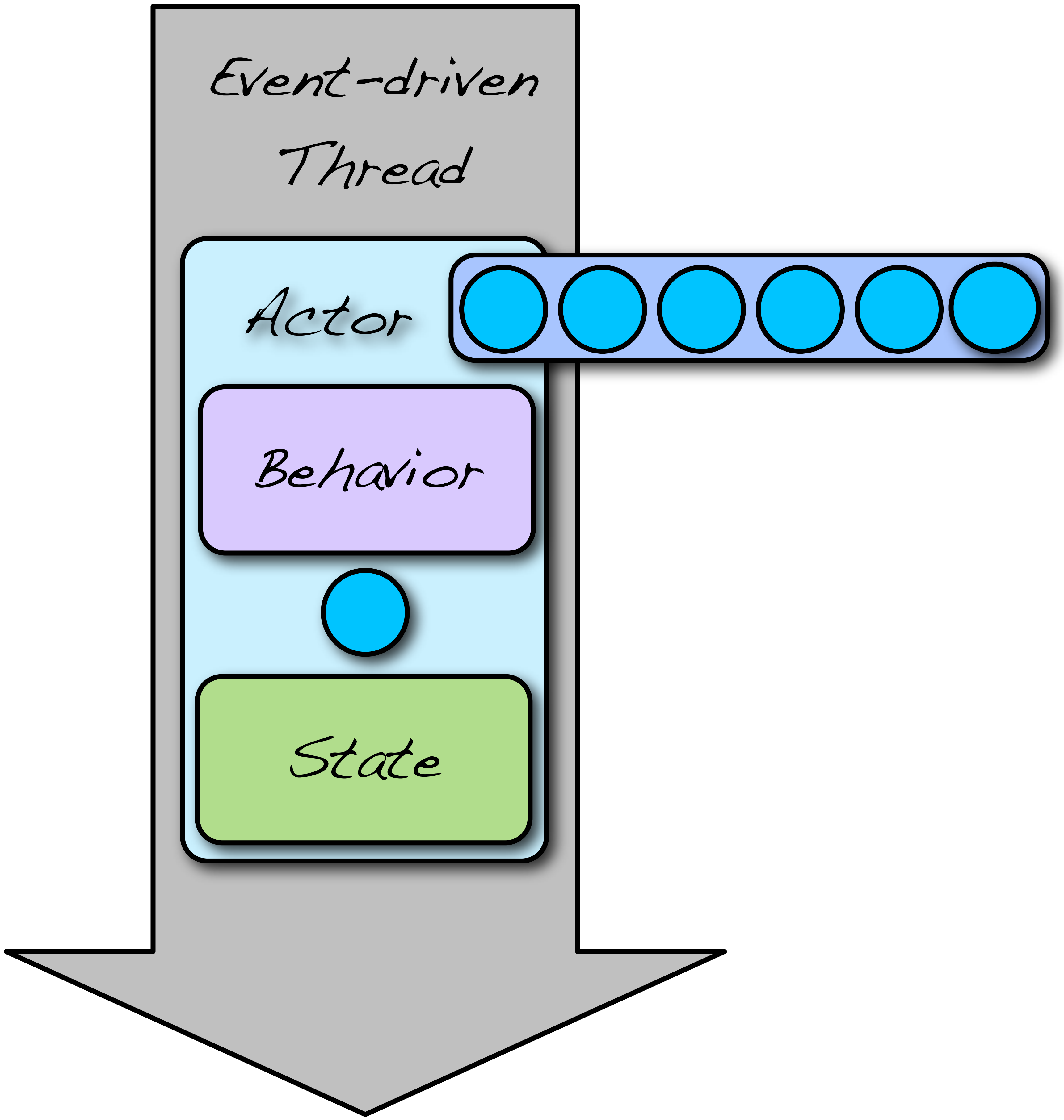
What is an Actor?

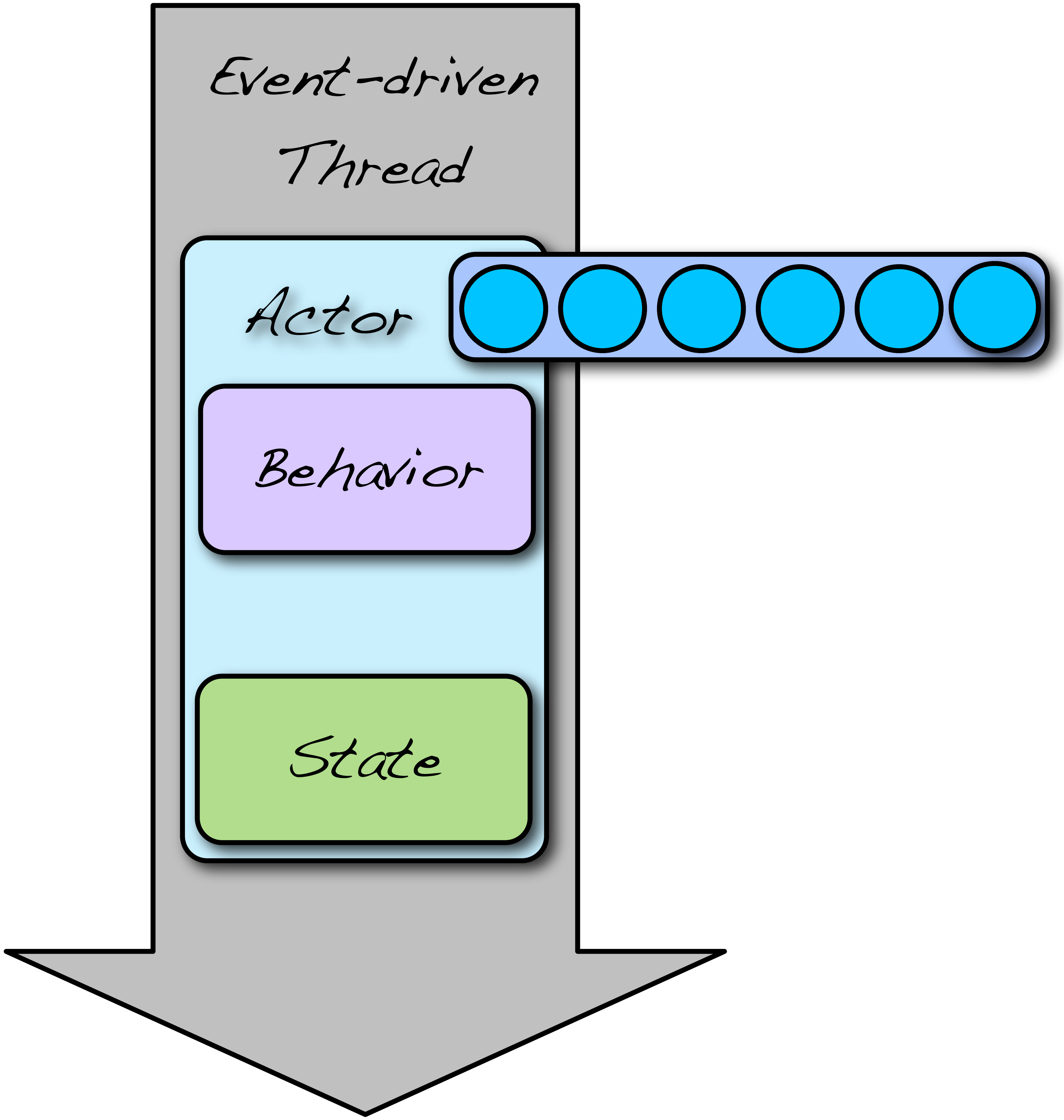

Actor
Behavior
State

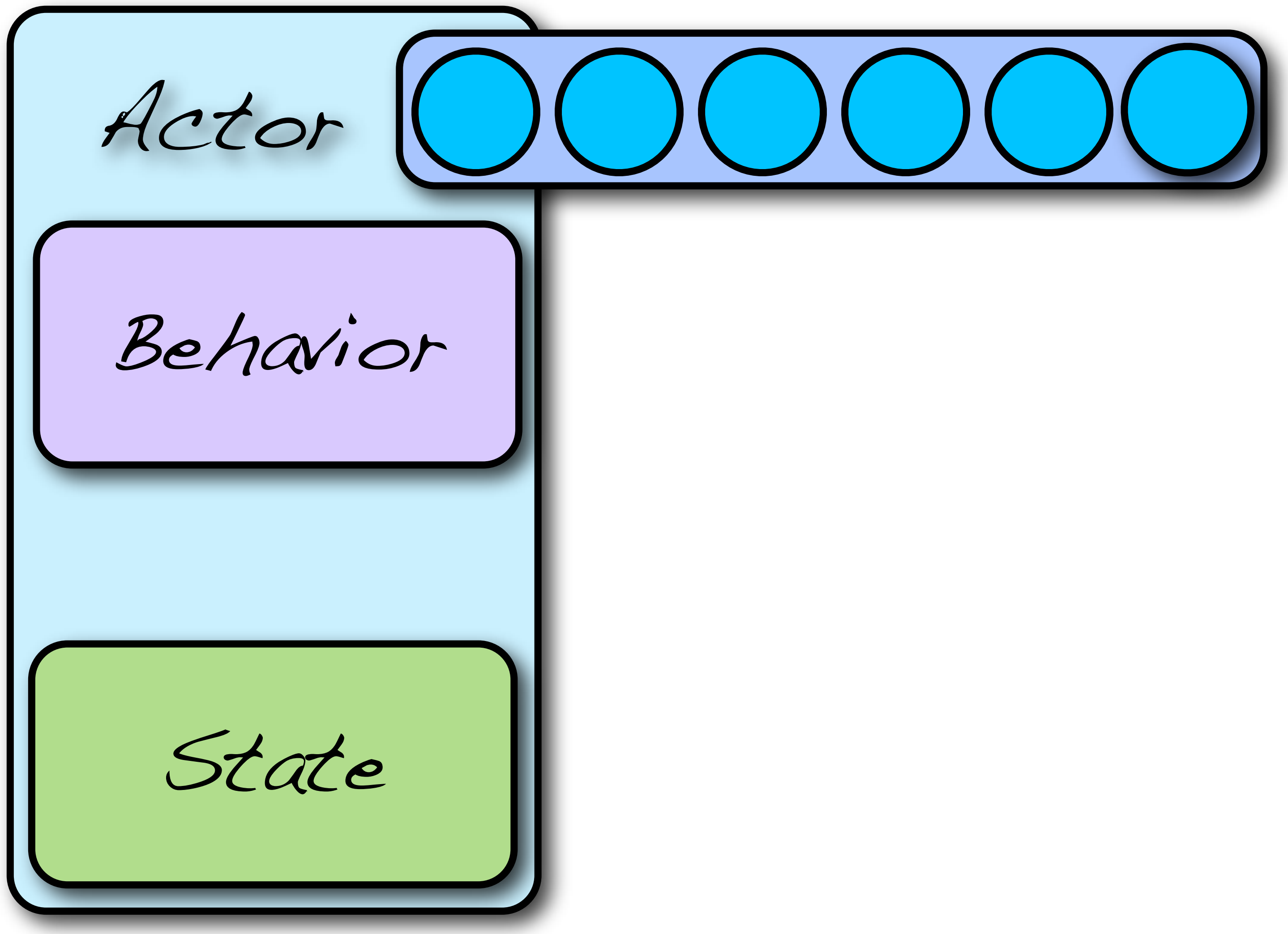

Event-driven Thread
Actor
Behavior
State

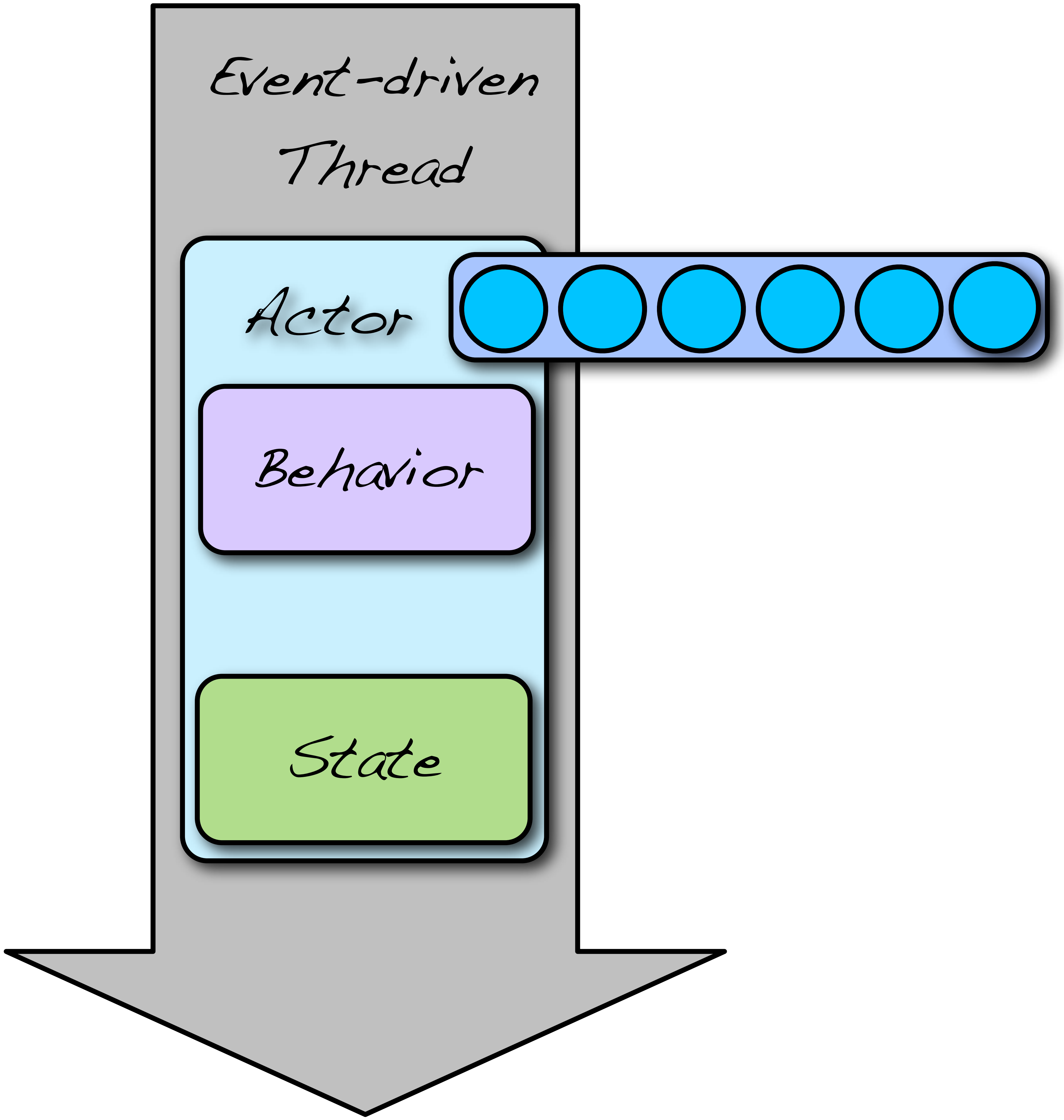

Event-driven
Thread
Actor
Behavior
State


Event-driven
Thread
Actor
Behavior
State


Event-driven
Thread
Actor
Behavior
State


Event-driven
Thread
Actor
Behavior
State


Actor
Behavior
State


Event-driven
Thread
Actor
Behavior
State

Akka Actors

one tool in the toolbox

Actors

```
case object Tick

class Counter extends Actor {
  var counter = 0

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

Create Actors

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

`counter` is an `ActorRef`

Start actors

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

Stop actors

```
val counter = actorOf[Counter].start
counter.stop
```

Send: !

```
counter ! Tick
```

fire-forget

Send: !!!

```scala
// returns a future
val future = actor !!! Message
future.await
val result = future.result
```

returns the Future directly

Future

```
val future1, future2, future3 =
  new DefaultCompletableFuture(1000)

future1.await
future2.onComplete(f => ...)

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

Future

```
// Blocking
Futures.awaitOne(futures)
Futures.awaitAll(futures)

// Non-blocking
val f = Futures.firstCompletedOf(futures)
val f = Futures.reduce(futures)((x, y) => ..)
val f = Futures.fold(zero)(futures)((x, y) => ..)
```

Send: !!

```
val result = (actor !! Message).as[String]
```

uses Future under the hood and blocks until timeout or completion

Reply

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

HotSwap

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

HotSwap

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

HotSwap

```
self.unbecome()
```

Set dispatcher

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

  ...
}

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

Remote Actors

Remoting in Akka 1.0

Remote Actors

Client-managed

Server-managed

Problem

Deployment (local vs remote) is a dev decision

We get a fixed and hard-coded topology

Can’t change it dynamically and adaptively

Needs to be a

deployment & runtime decision

Clustered Actors

(in development for upcoming Akka 2.0)

Address

```
val actor = actorOf[MyActor](“my-service”)
```

Bind the actor to a virtual address

Deployment

- Actor address is virtual and decoupled from how it is deployed
- If no deployment configuration exists then actor is deployed as local
- The same system can be configured as distributed without code change (even change at runtime)

- Write as local but deploy as distributed in the cloud without code change
- Allows runtime to dynamically and adaptively change topology

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
    }
  }
}
```

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Type of load-balancing

Tuesday, May 3, 2011

Deployment configuration

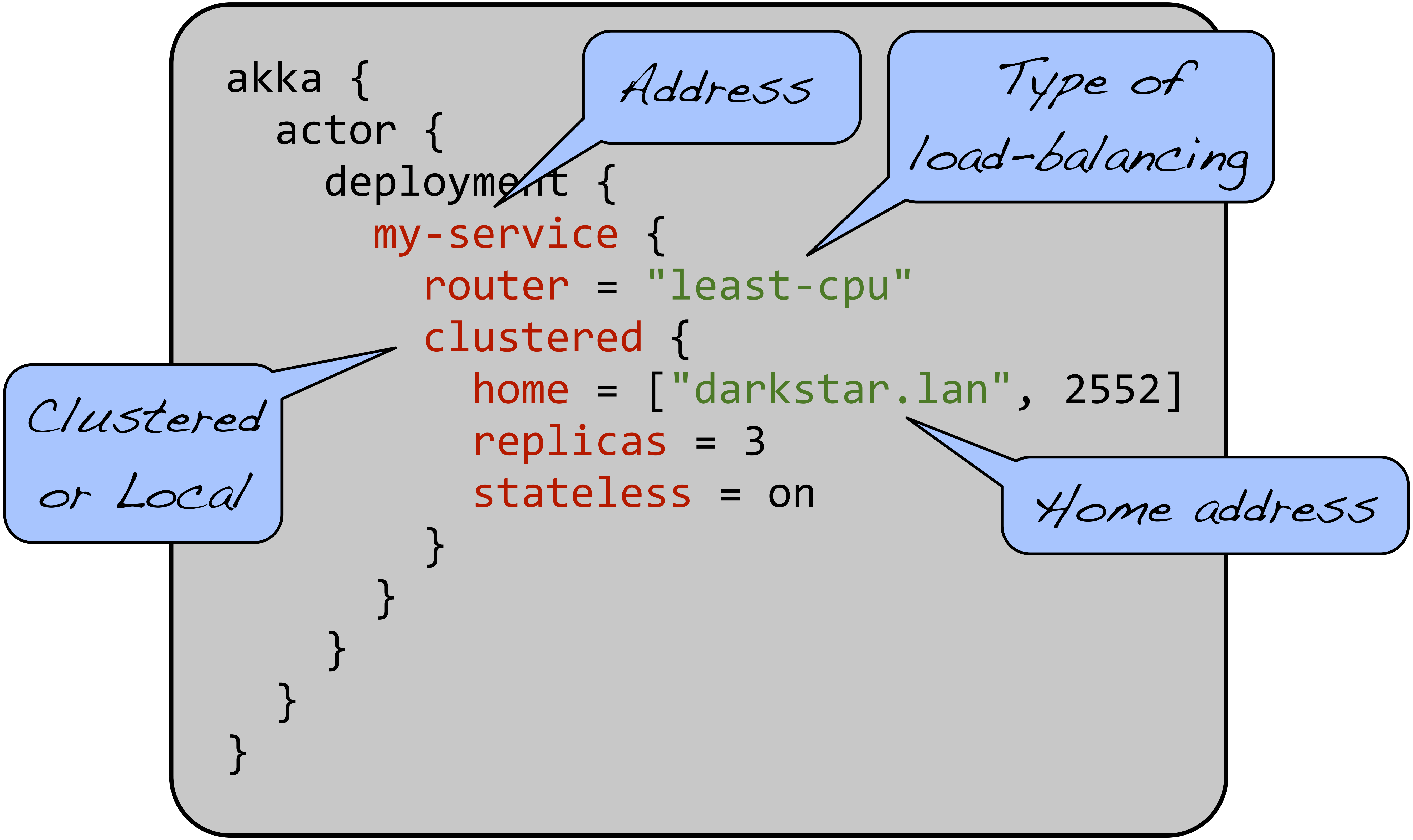
```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

Type of load-balancing

Clustered or Local

Deployment configuration



```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
    }
  }
}
```

Deployment configuration

```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
    }
  }
}
```

Address

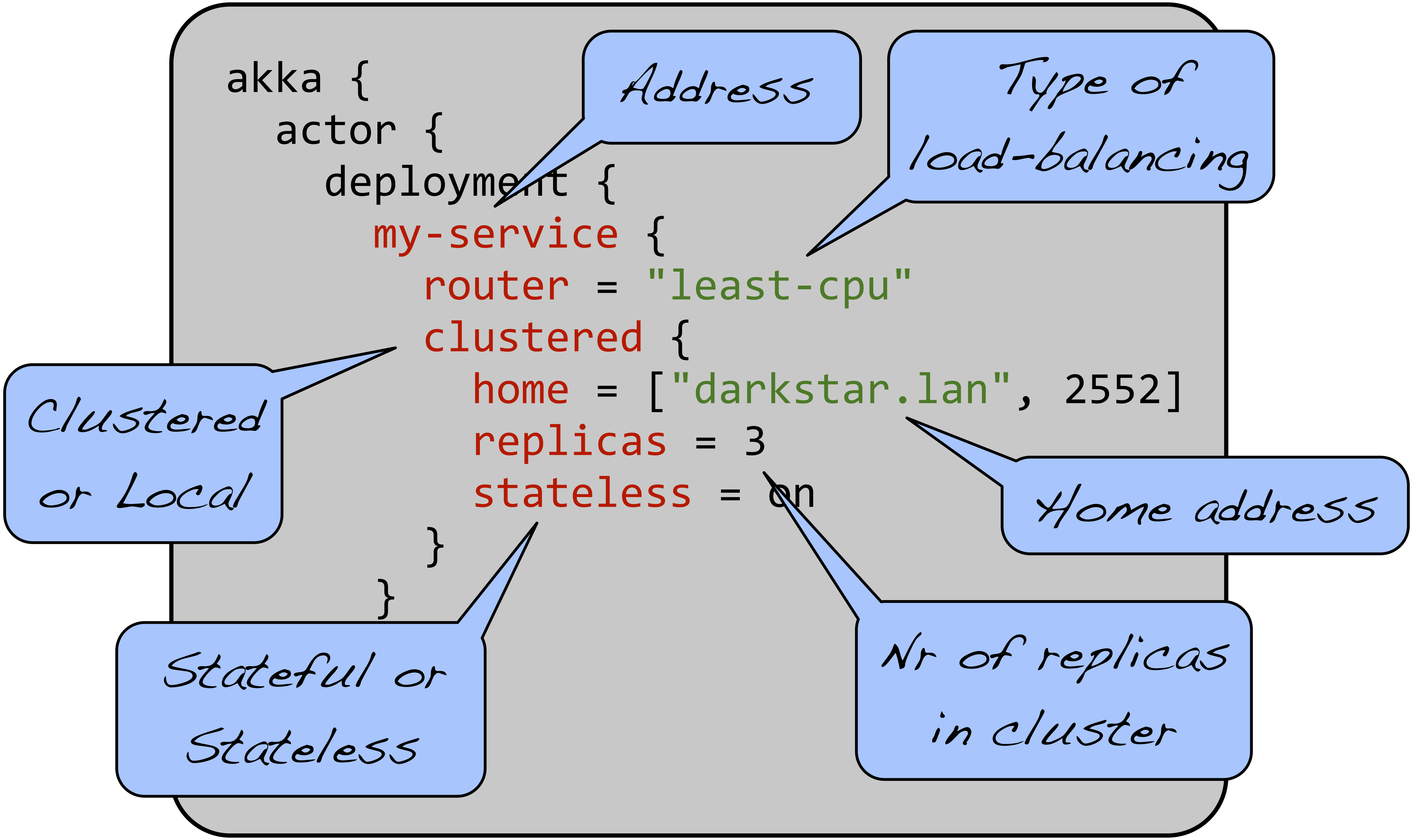
Type of load-balancing

Clustered or Local

Home address

Nr of replicas in cluster

Deployment configuration



```
akka {
  actor {
    deployment {
      my-service {
        router = "least-cpu"
        clustered {
          home = ["darkstar.lan", 2552]
          replicas = 3
          stateless = on
        }
      }
```

The runtime provides

- Subscription-based cluster membership service
- Highly available cluster registry for actors
- Highly available centralized configuration service
- Automatic replication with automatic fail-over upon node crash
- Transparent and user-configurable load-balancing
- Transparent adaptive cluster rebalancing
- Leader election
- Compute grid facilities
- Event Sourcing

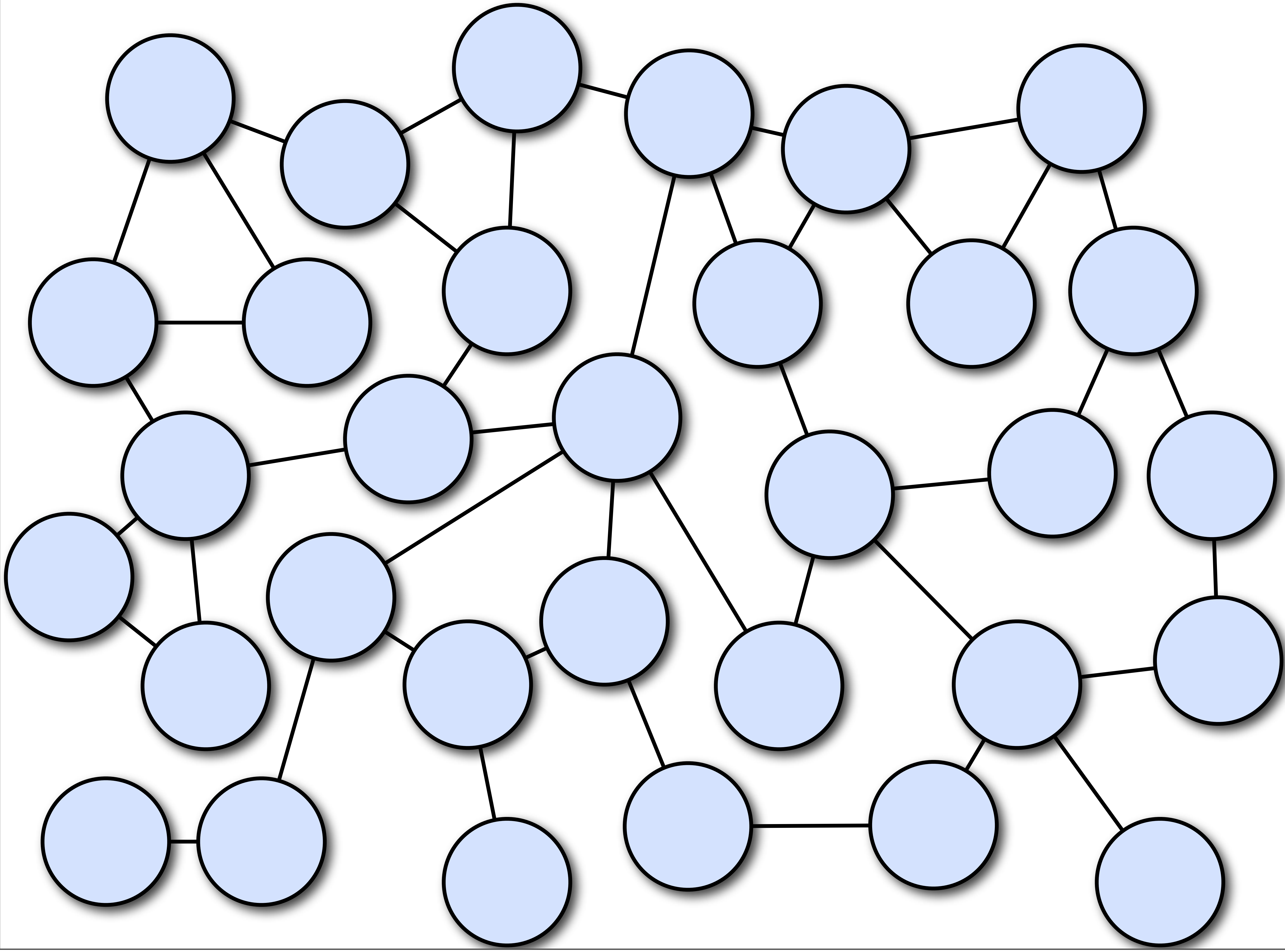
Let it crash

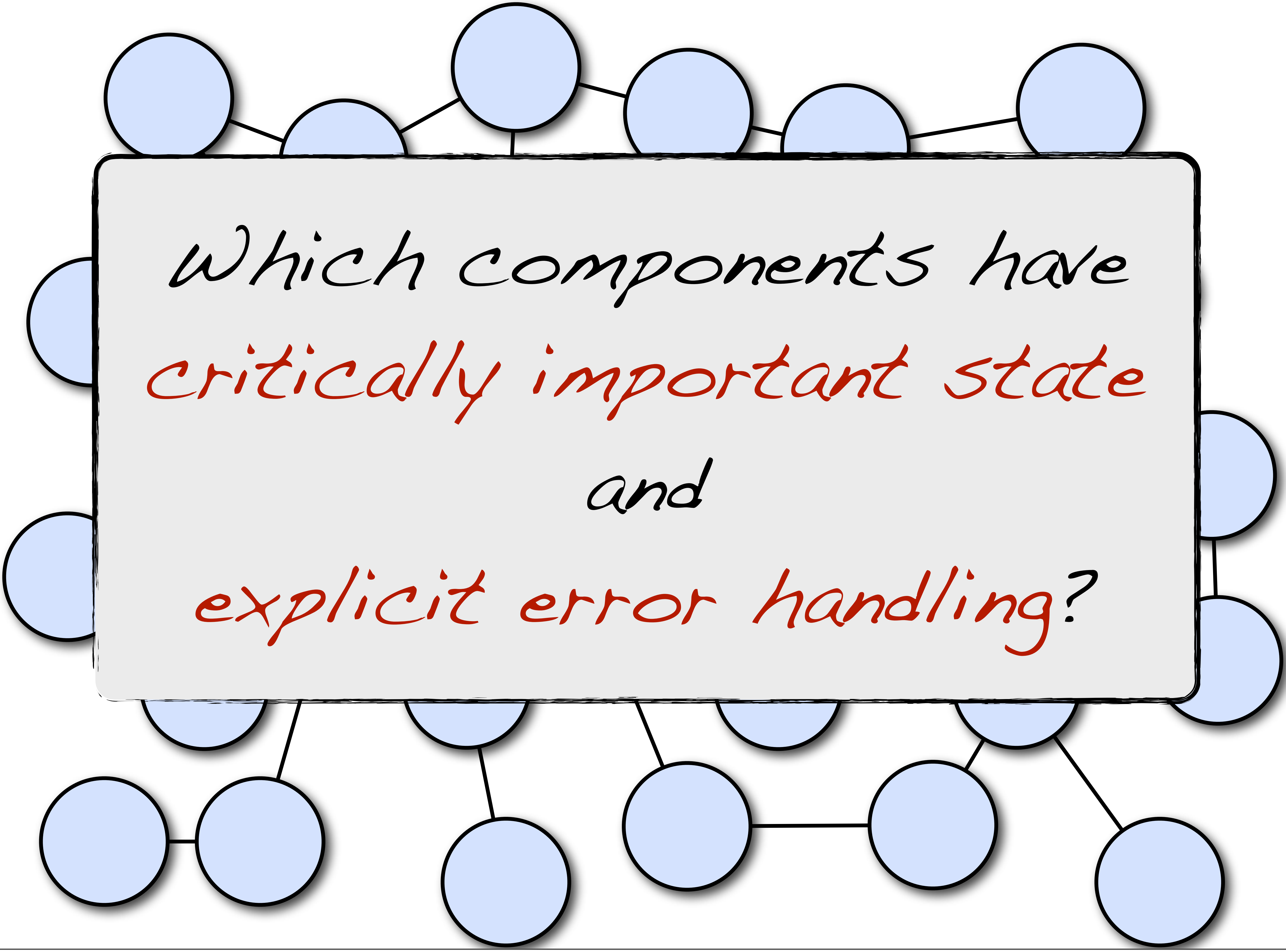
fault-tolerance

The Erlang model

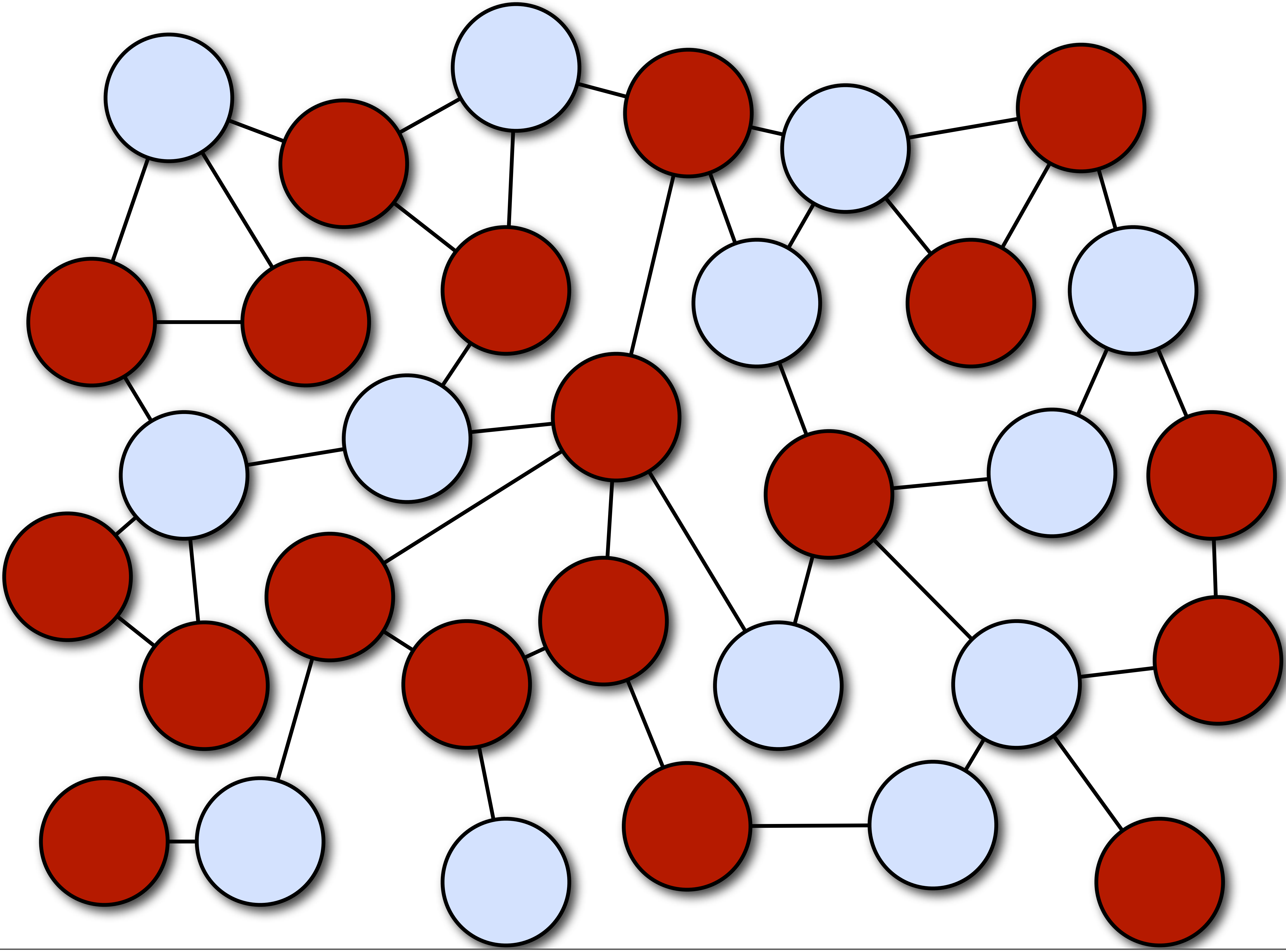
9 nines

...let's take a standard OO application





Which components have critically important state and explicit error handling?



Classification of State

- Scratch data
- Static data
  - Supplied at boot time
  - Supplied by other components
- Dynamic data
  - Data possible to recompute
  - Input from other sources; data that is impossible to recompute

Classification of State

- Scratch data
- Static data
  - Supplied at boot time
  - Supplied by other components
- Dynamic data
  - Data possible to recompute
  - Input from other sources; data that is impossible to recompute

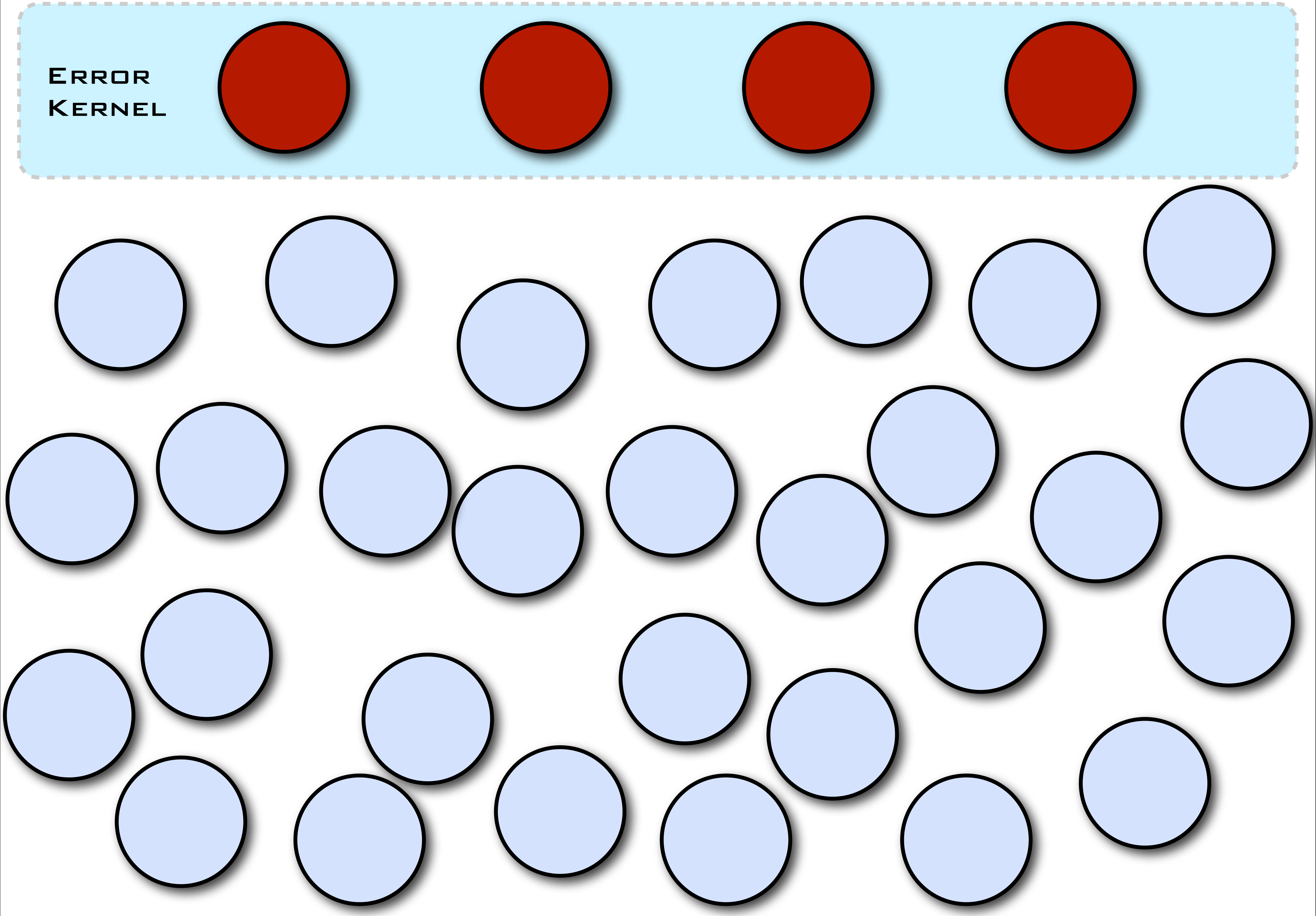
Classification of State

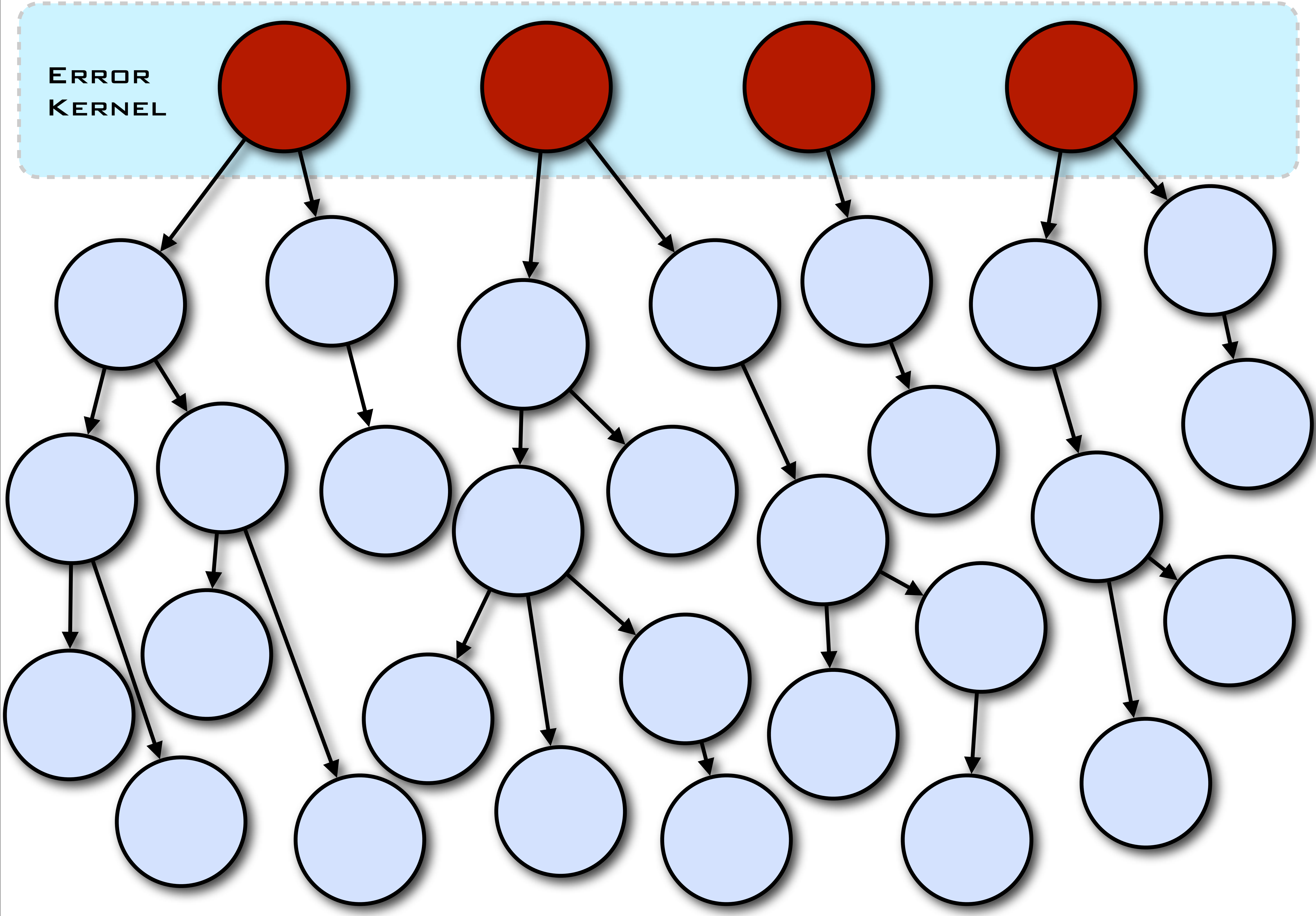
- Scra
- Stati
  - Su
  - Su nts
- Dyna
  - Da e

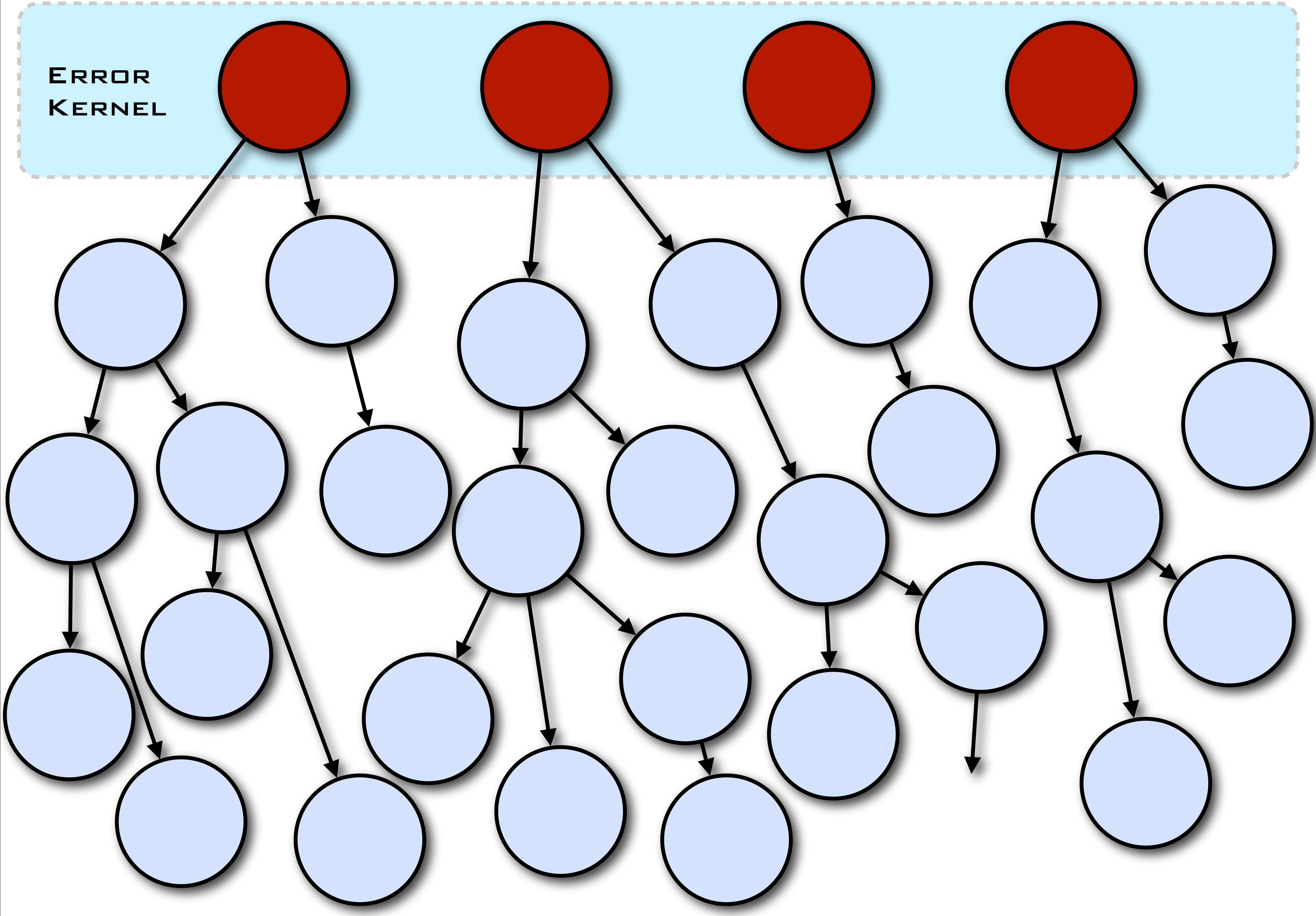
Must be
protected
by any means

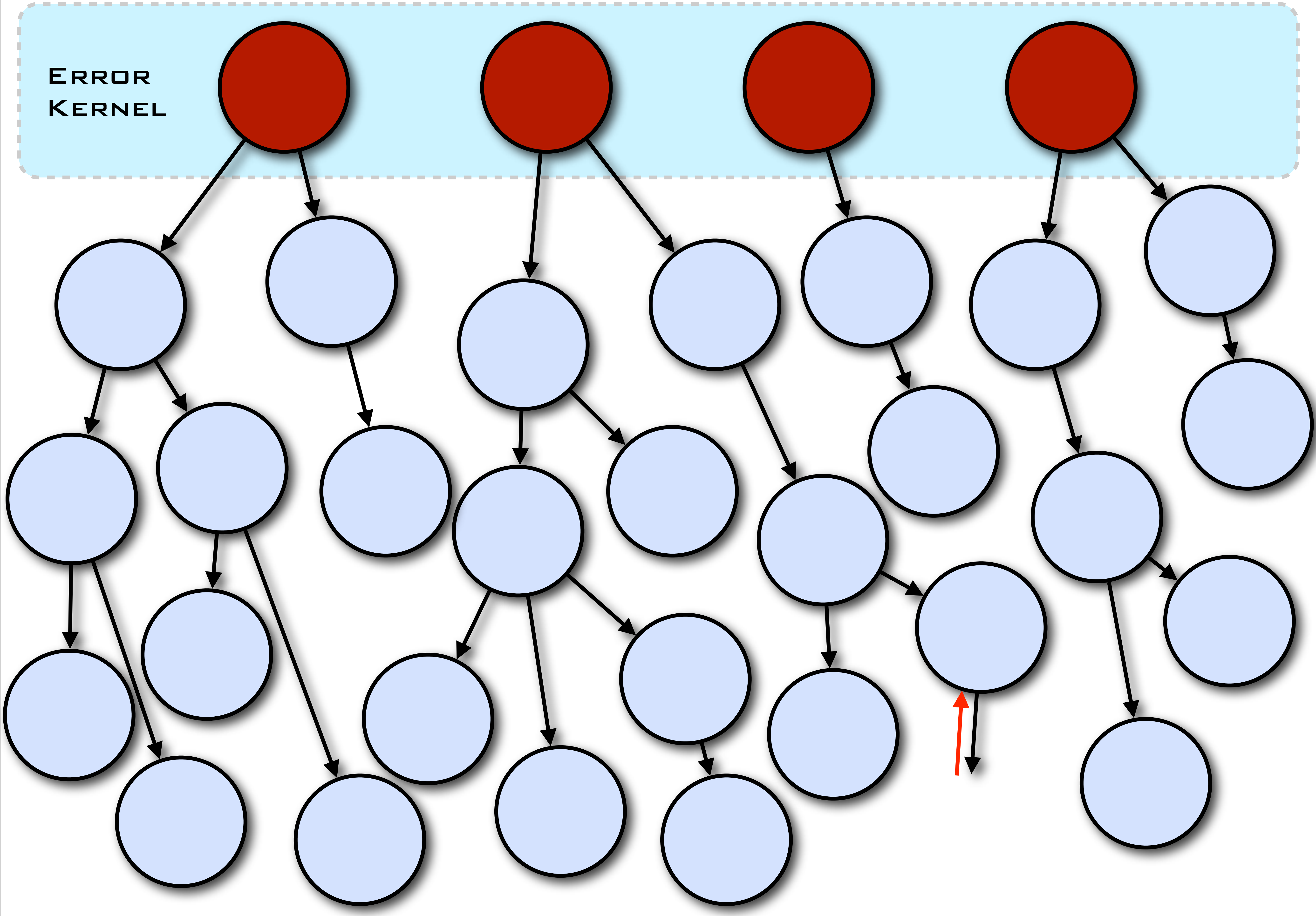
- Input from other sources; data that is impossible to recompute

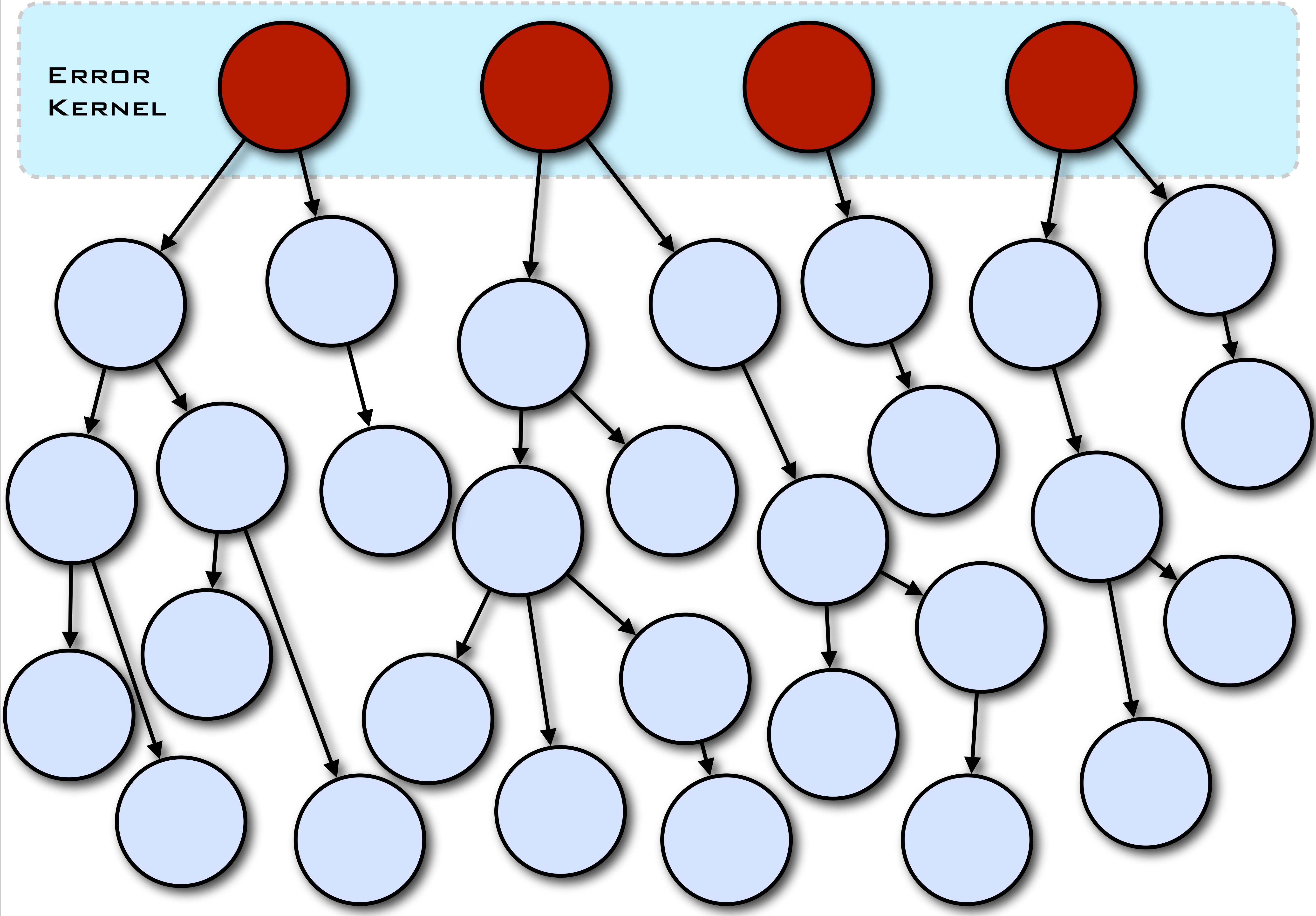
Fault-tolerant onion-layered Error Kernel

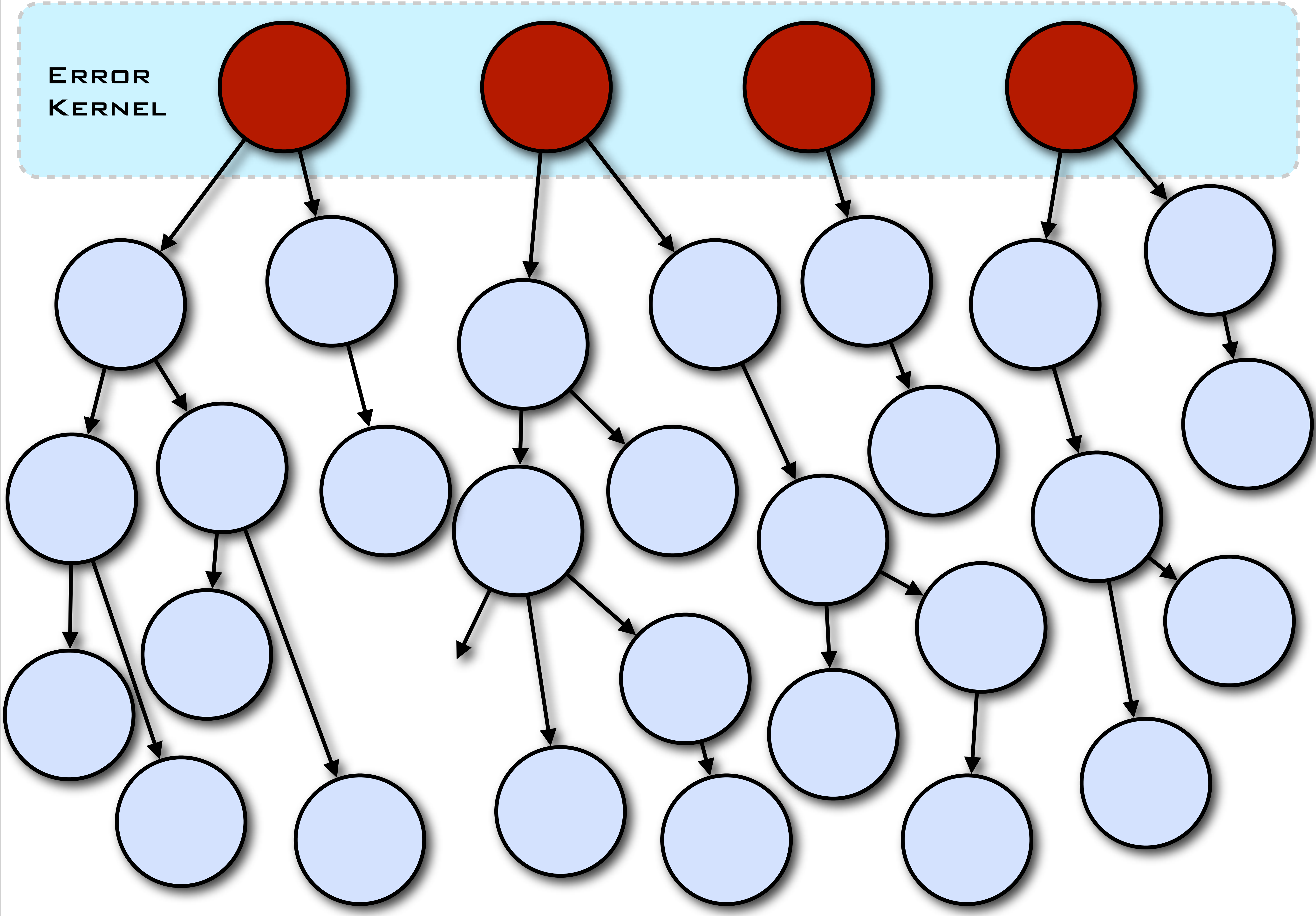

Error
Kernel

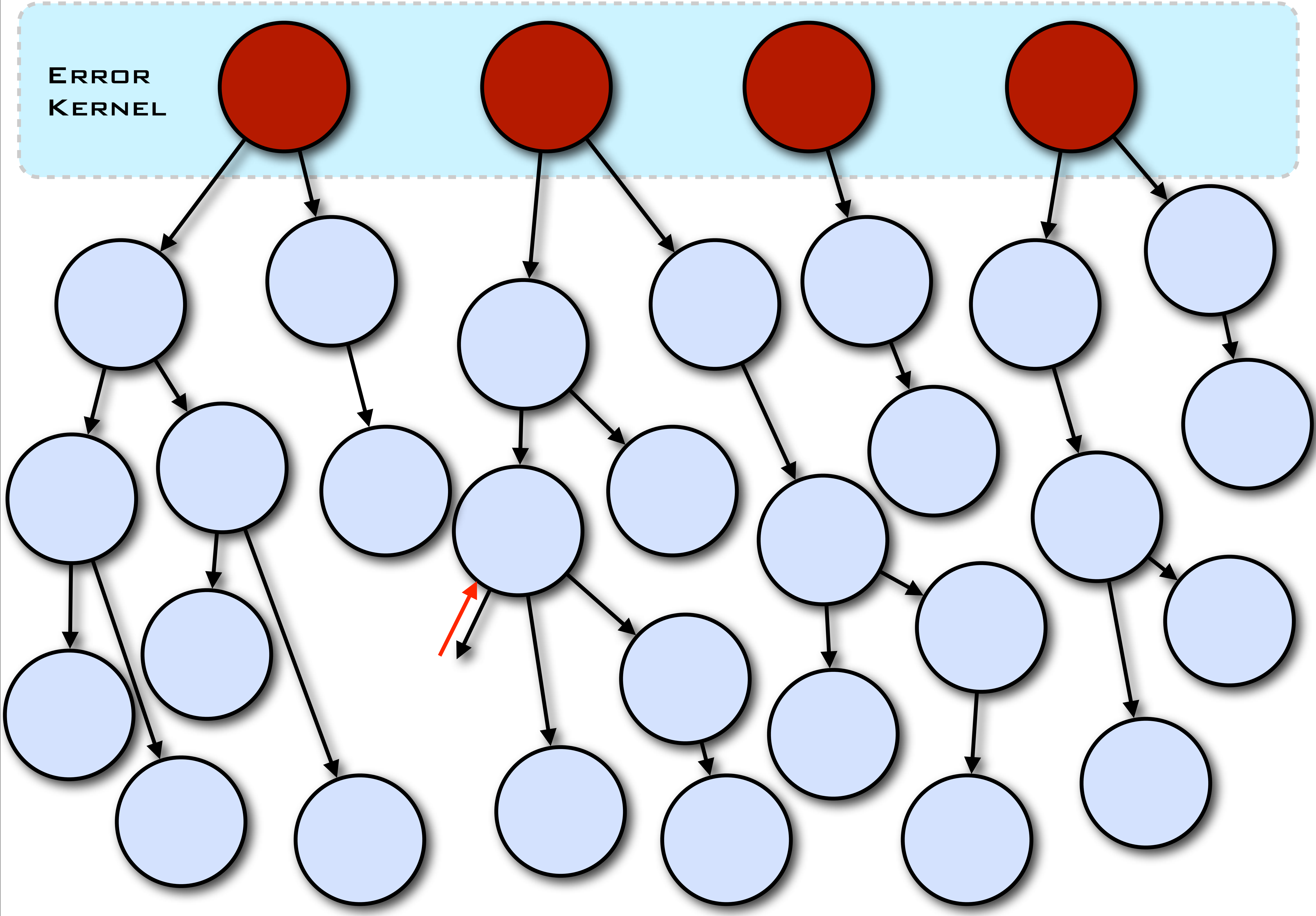

Error
Kernel

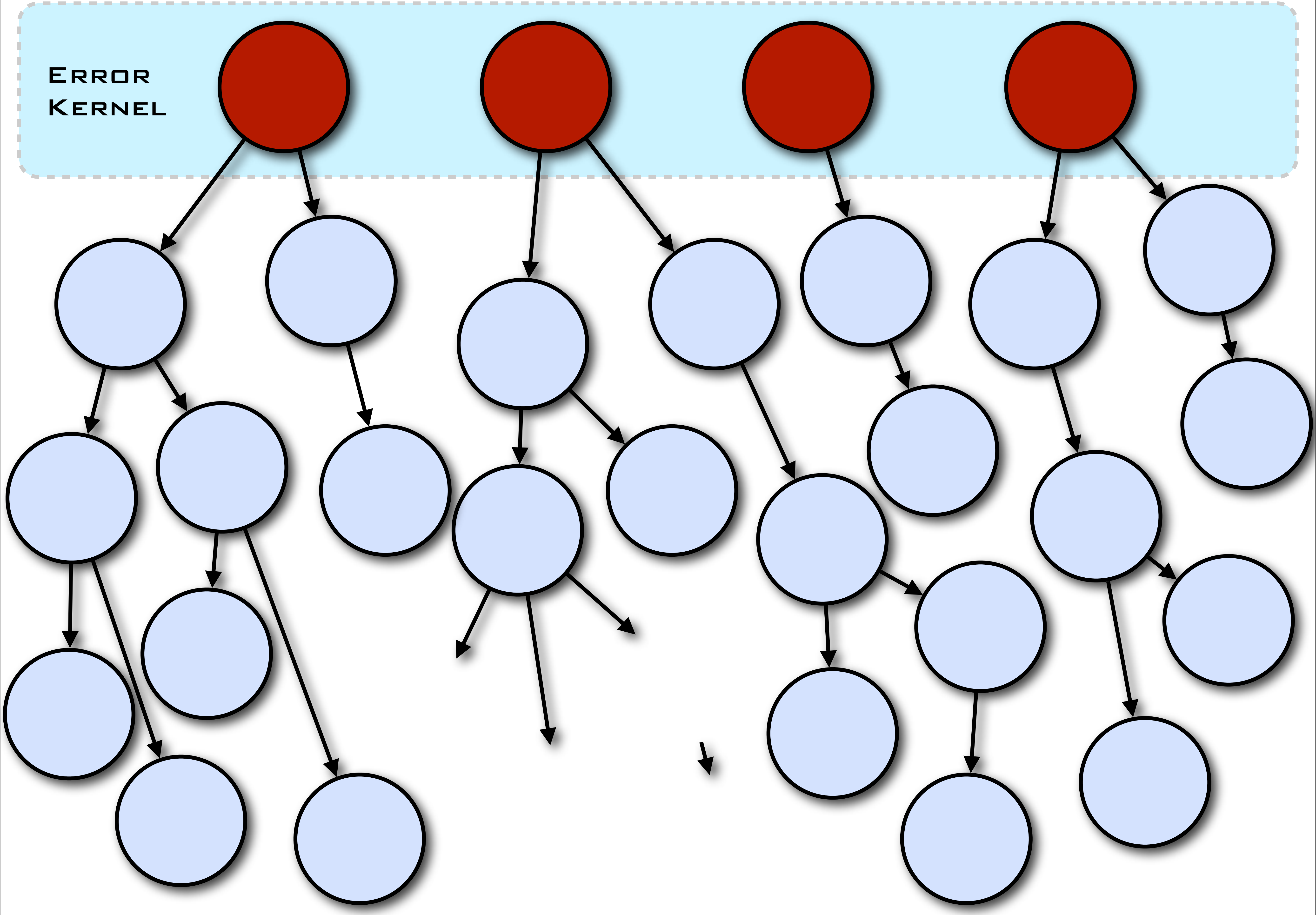

Error Kernel

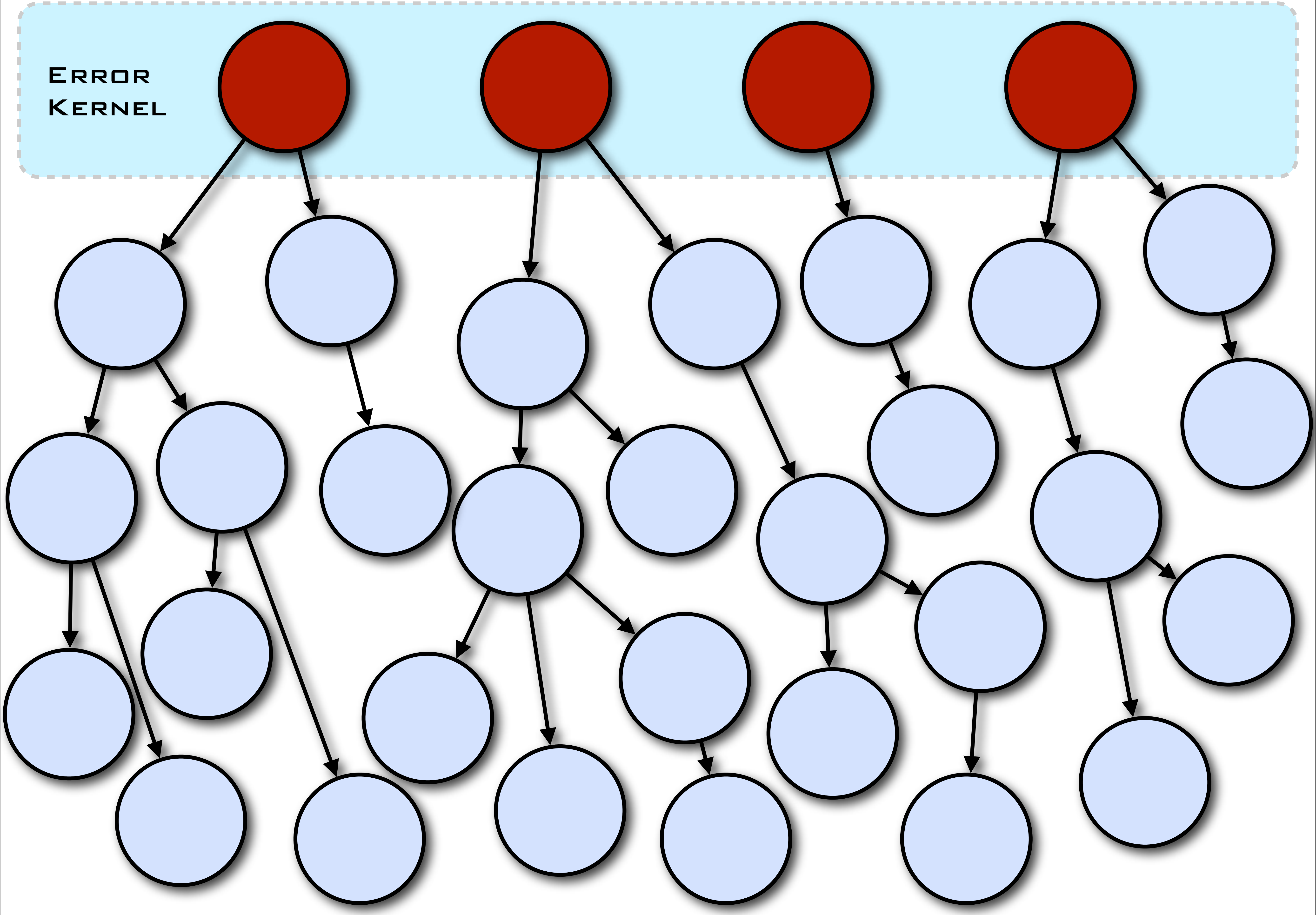

Error Kernel

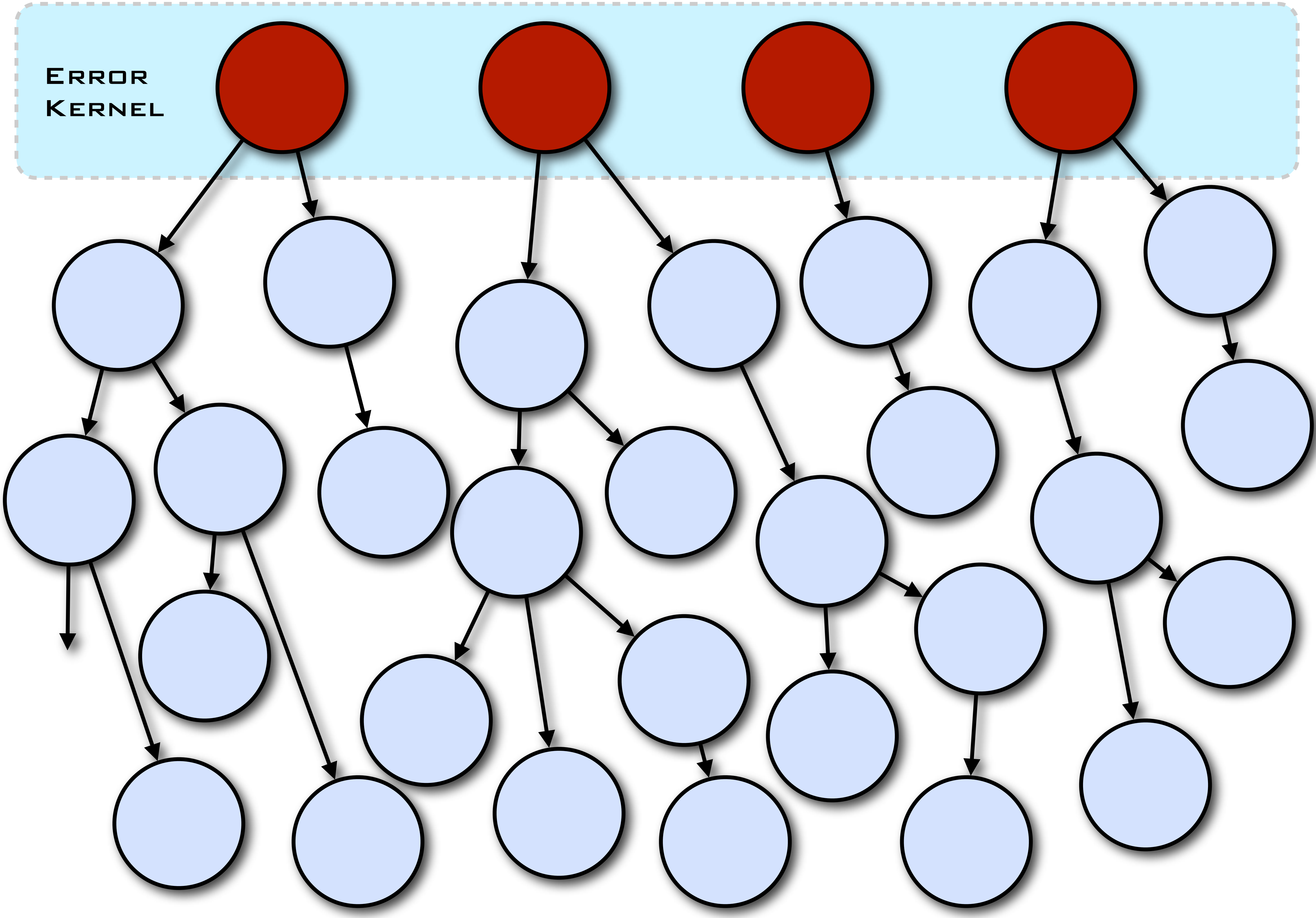

Error Kernel

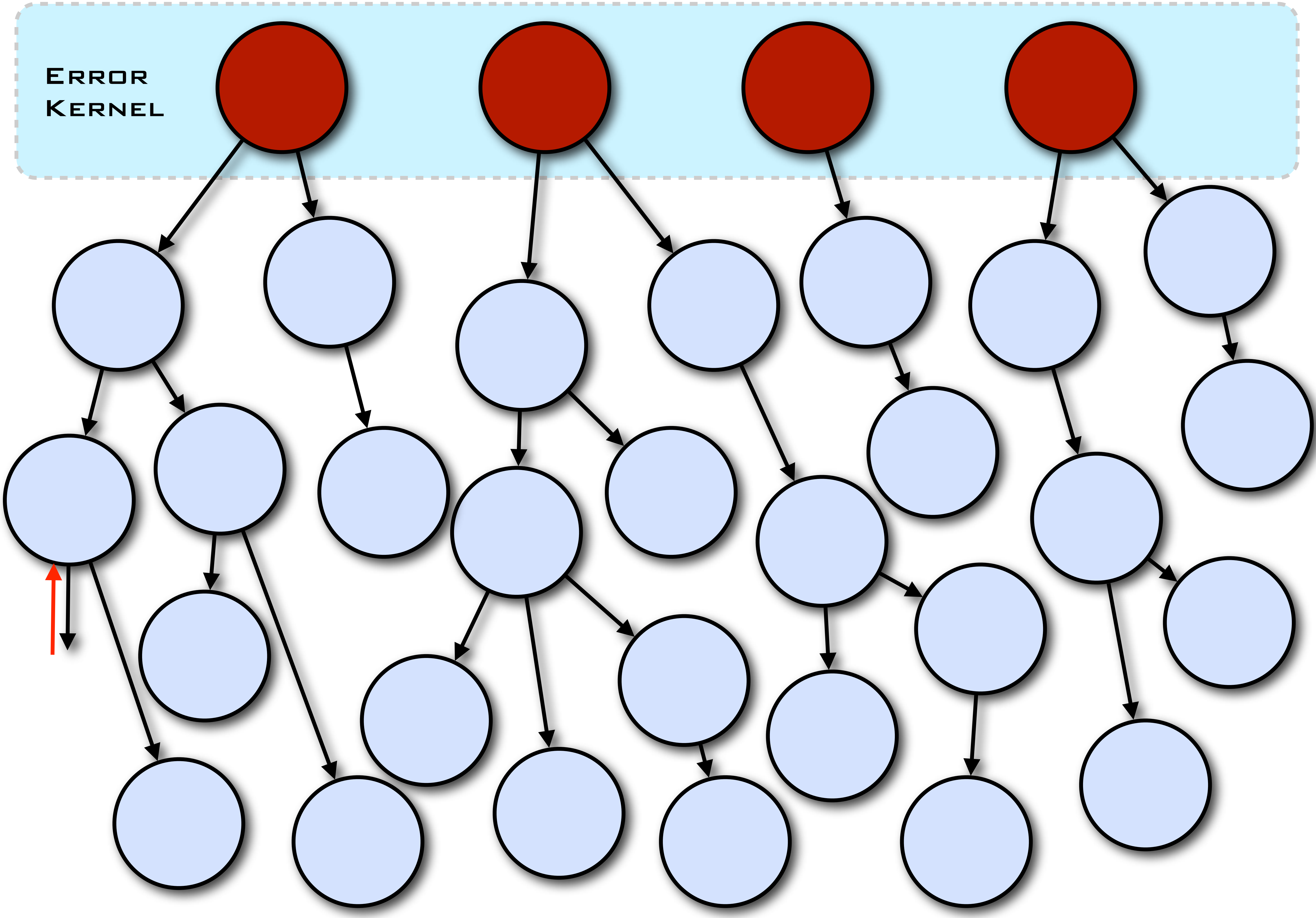

Error
Kernel

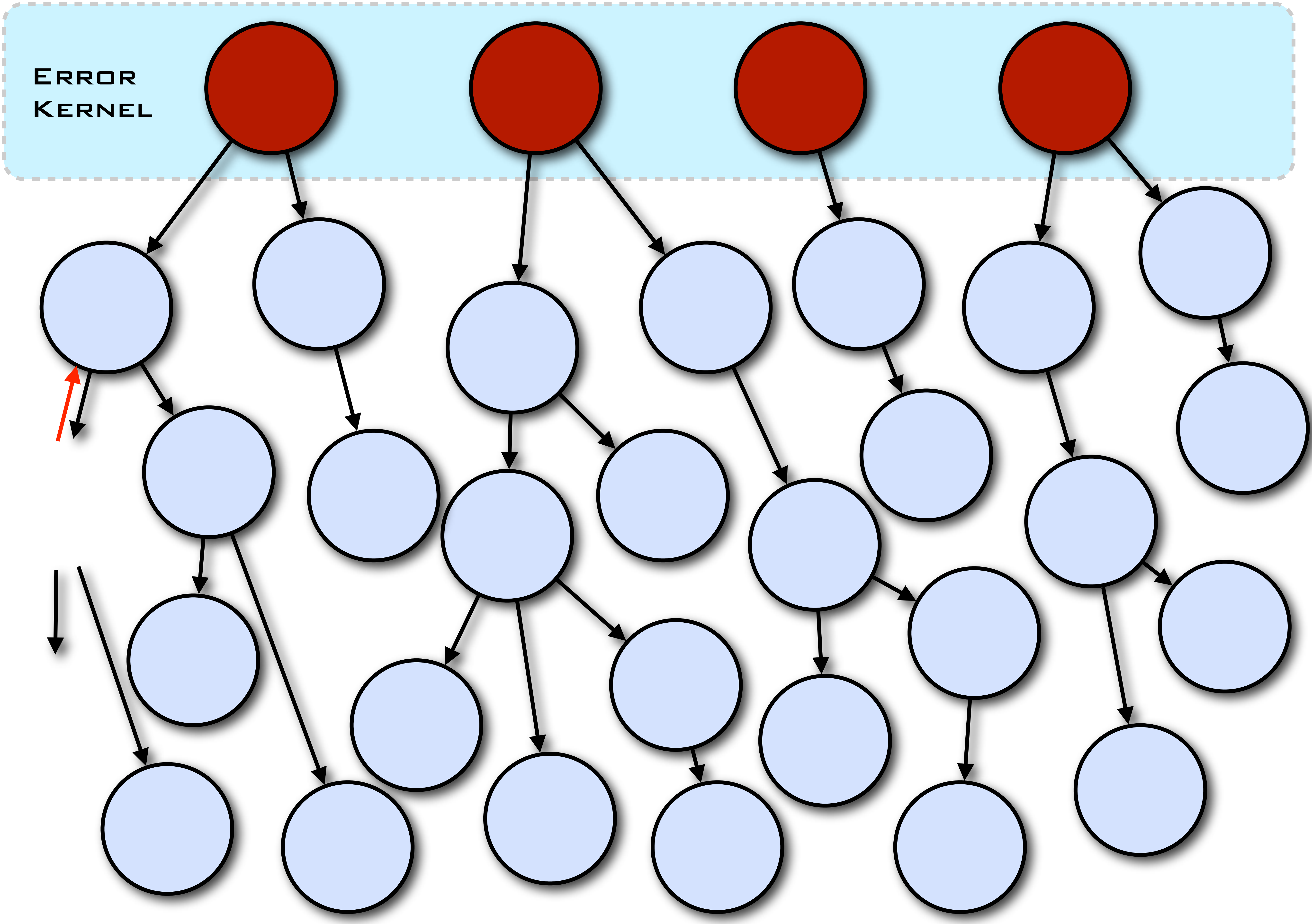

Error Kernel

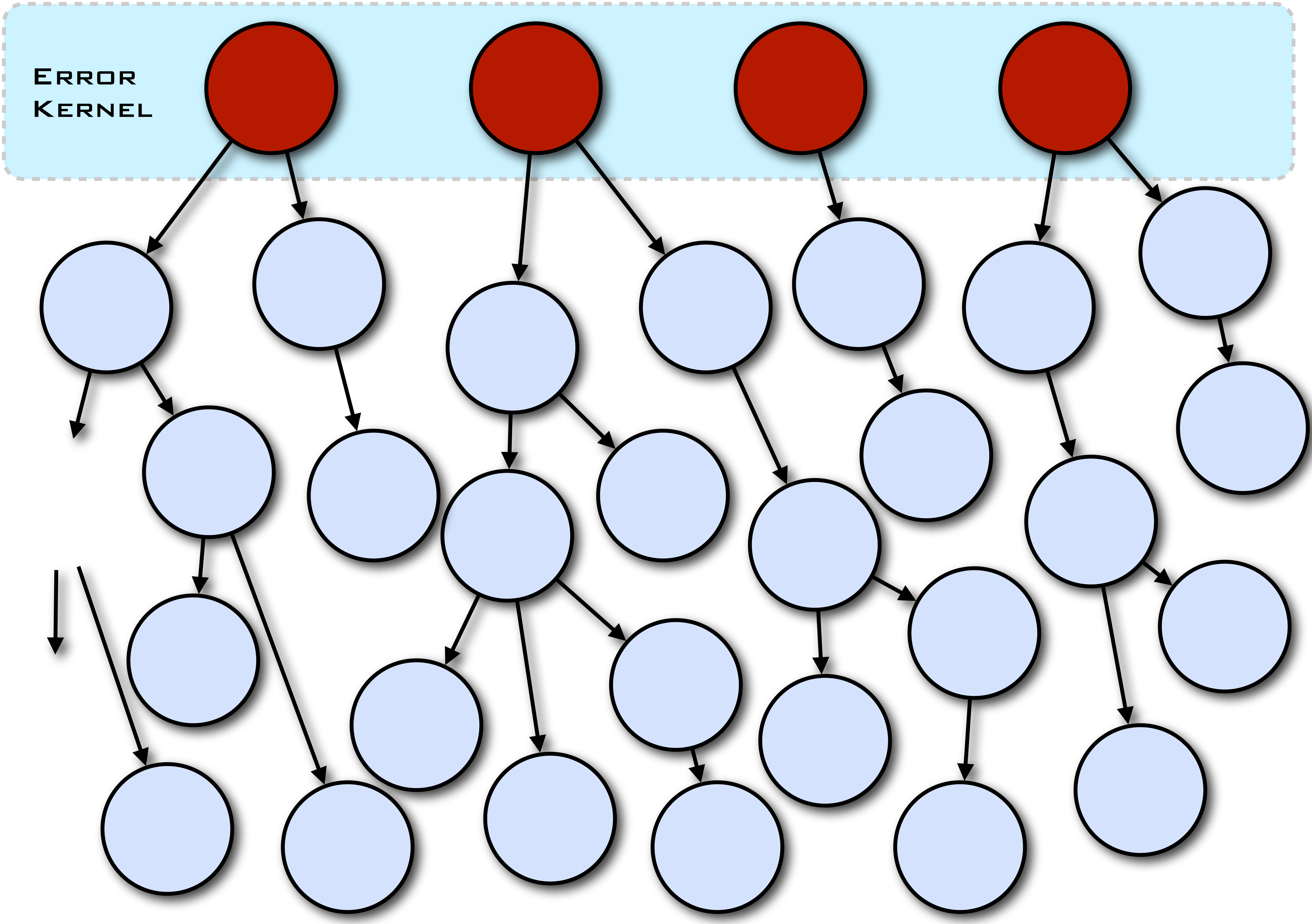

Error Kernel

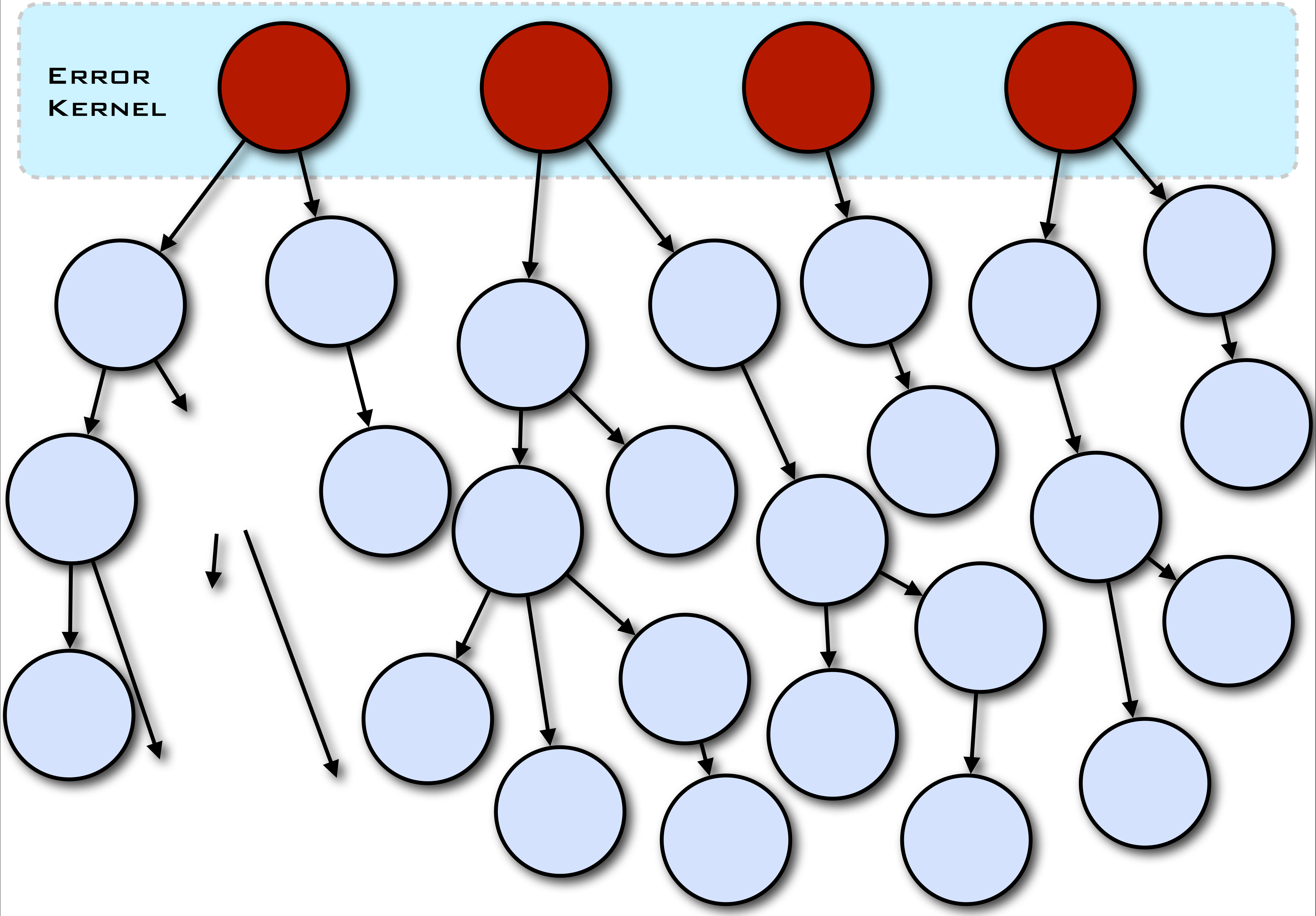

Error Kernel

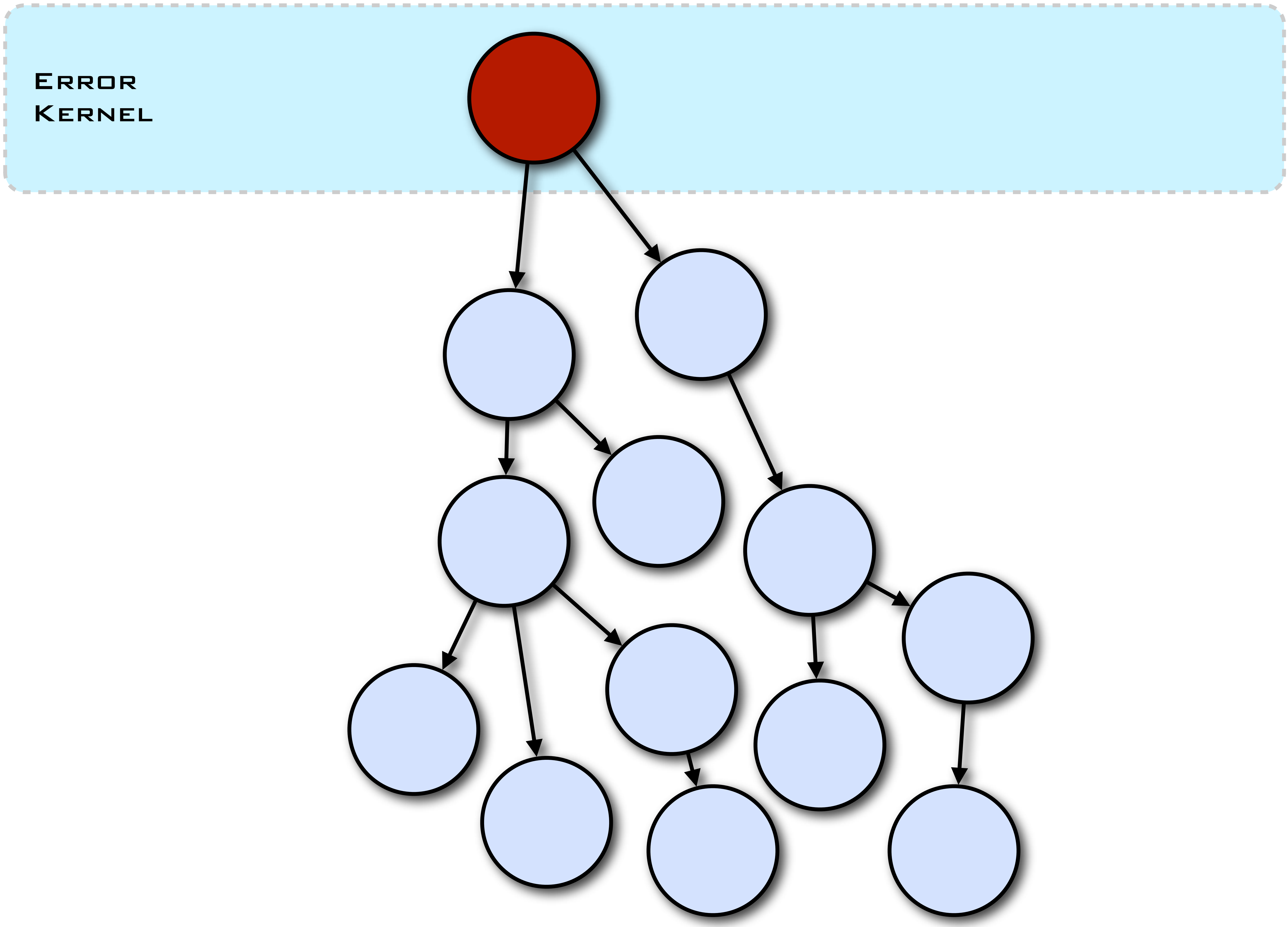

Error Kernel

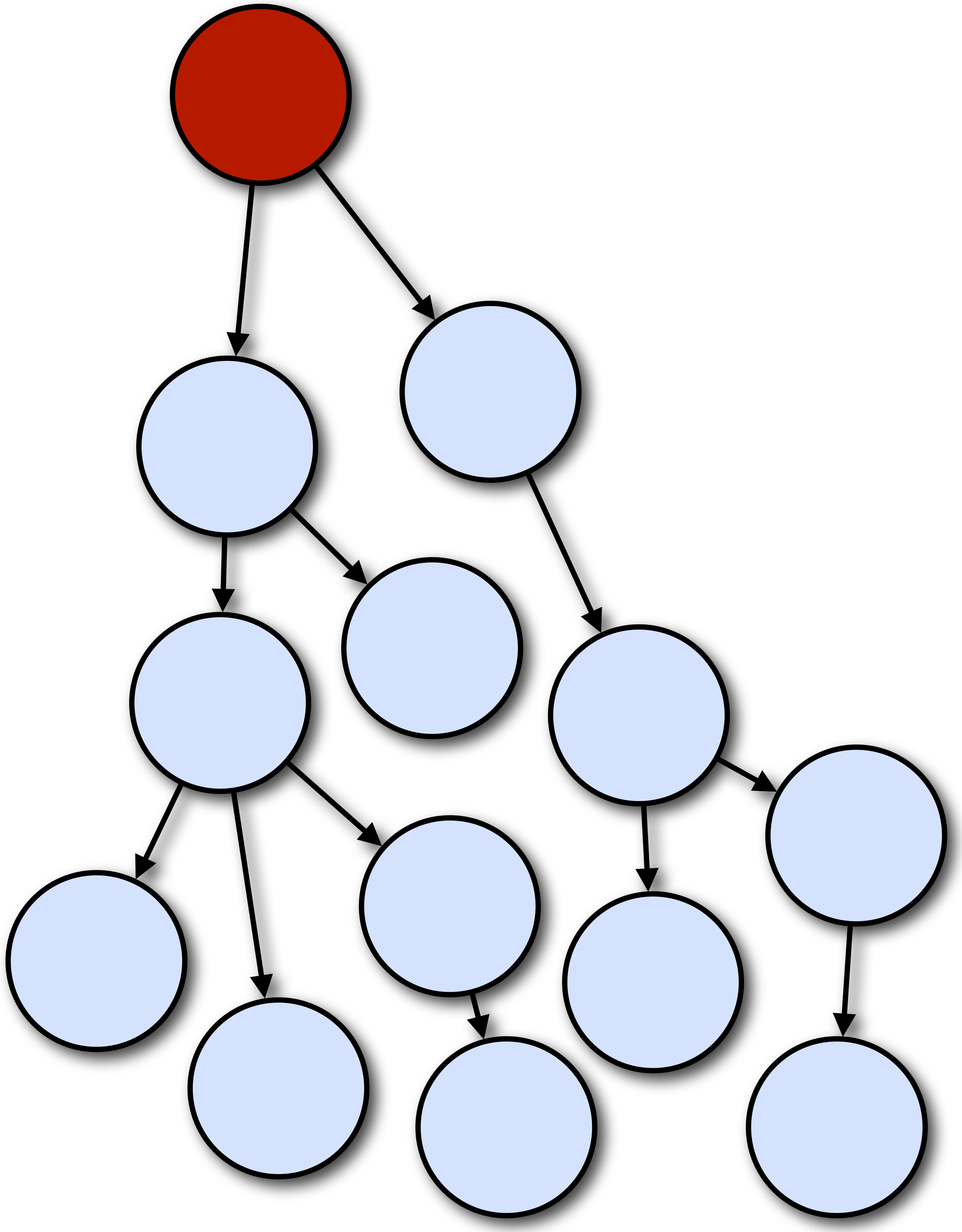

Error Kernel

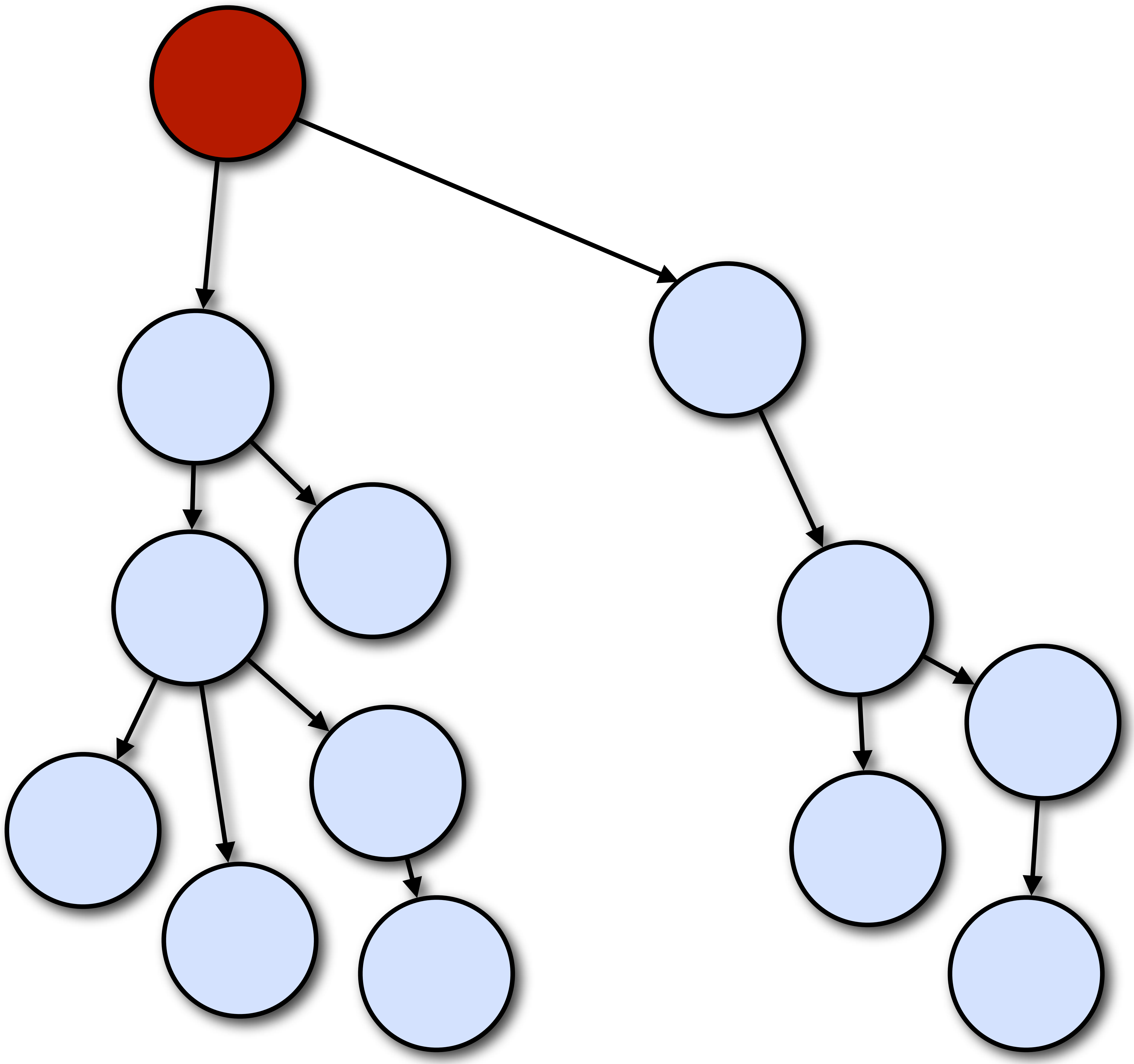

Error
Kernel

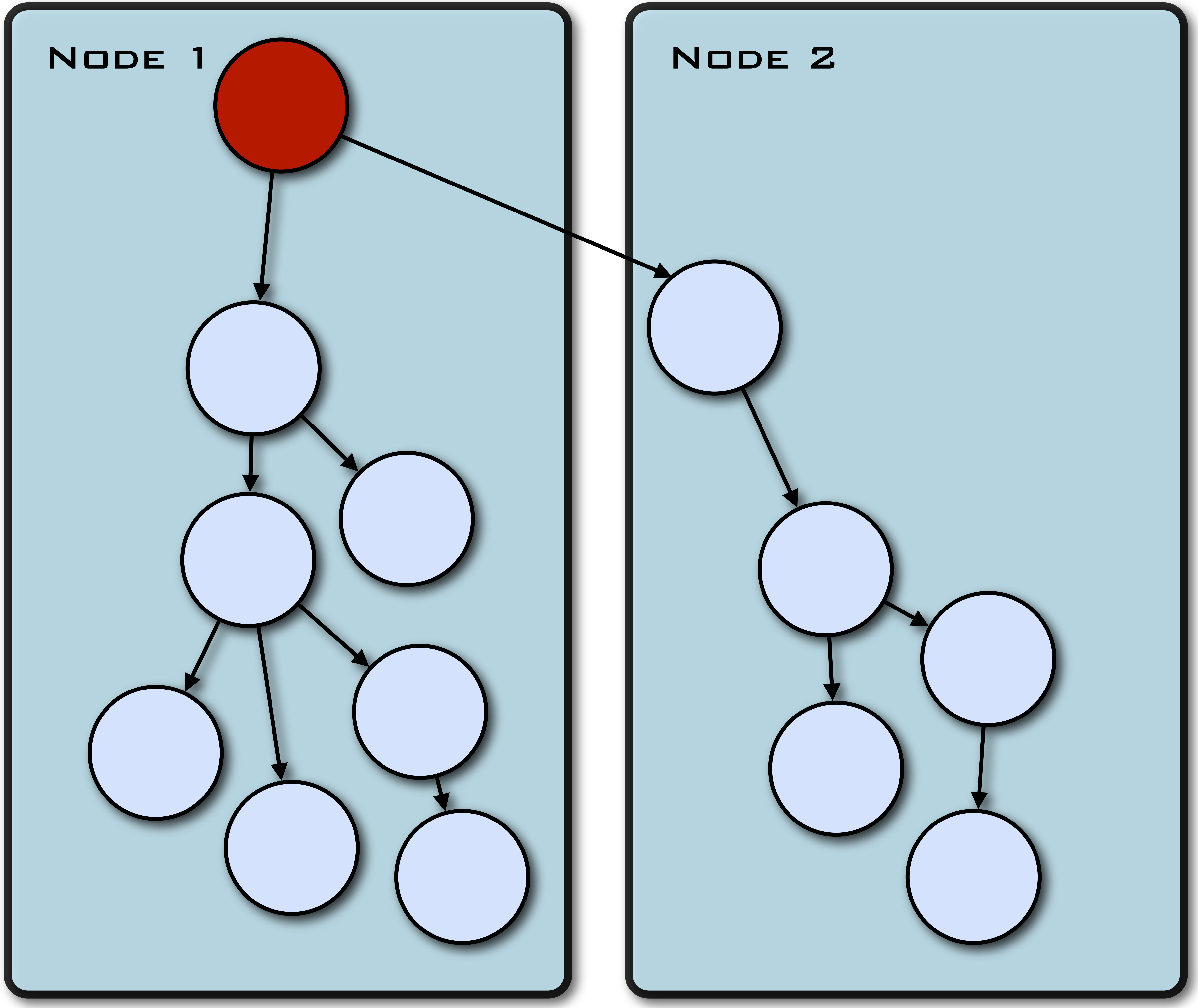

Error
Kernel


Error
Kernel


Error
Kernel






Node 1
Node 2

...and much much more

STM

FSM

HTTP

Camel

Microkernel

Guice

JTA

Dataflow

OSGi

AMQP

scalaz

Spring

Security

Project Hydrogen:

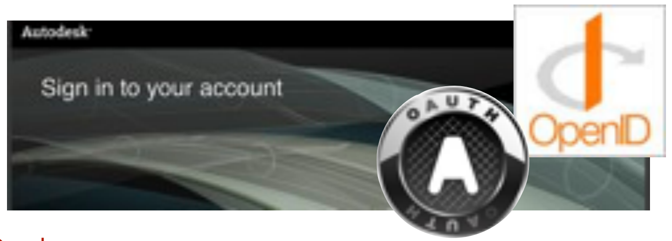
Building a distributed compute platform for design engineering with Akka

Garrick Evans

Autodesk, Inc

The Big Picture

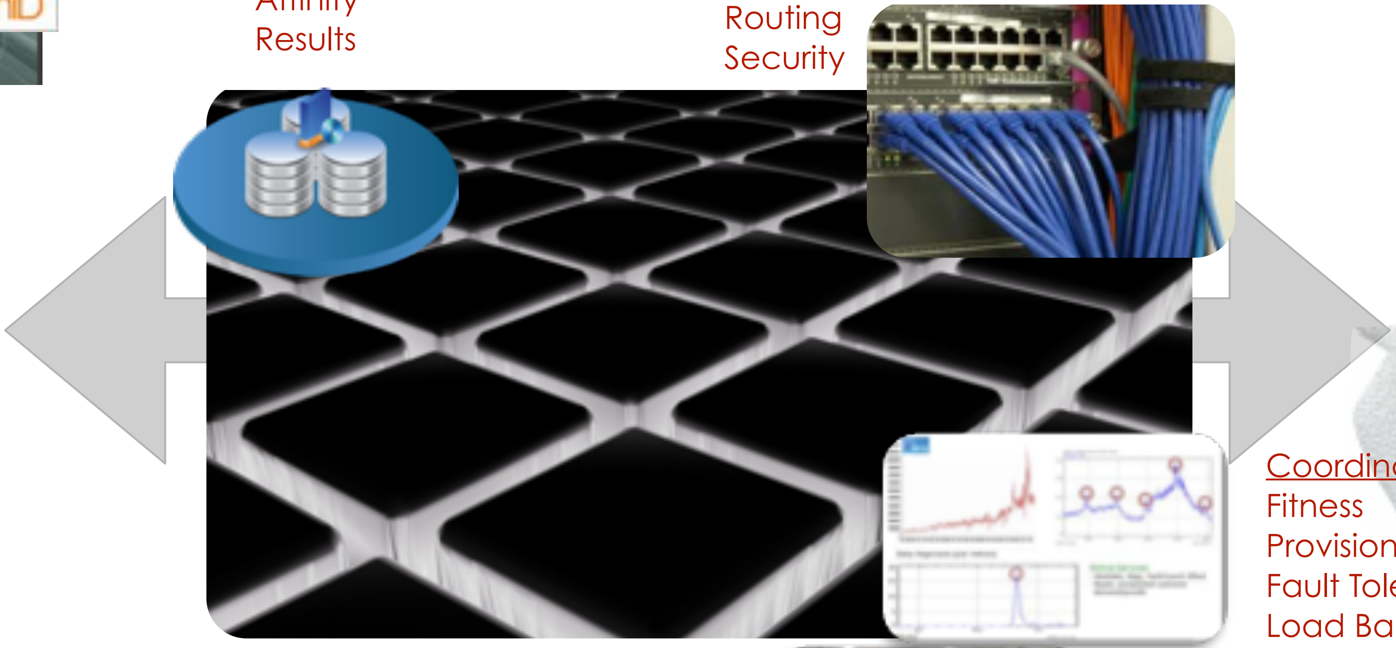
Products & Services



Customers
Customizable Offerings
Powerful On-Demand Technology
Flexibility in Access

Data
Caching
Affinity
Results

Channels
Partitioning (Functional, QoS)
Scheduling
Routing
Security

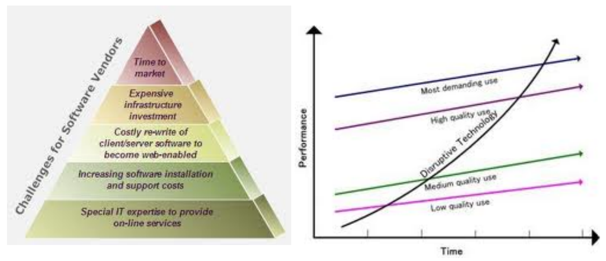


Meter
Quotas
Capacity

Events
Log
Semantics
Scrutinize
Process

Coordination
Fitness
Provisioning
Fault Tolerance
Load Balancing

Business
Offering
Tiering
Pricing



Hybrid Environment
Managed/Virtual
Public/Private

Audit
Operations
Analytics
Profiles

Some examples

- Clustered Physically-Correct Rendering
- Manufactured Part Design Optimization and Digital Simulation
- 3D Model Reconstruction from Photo Scenes

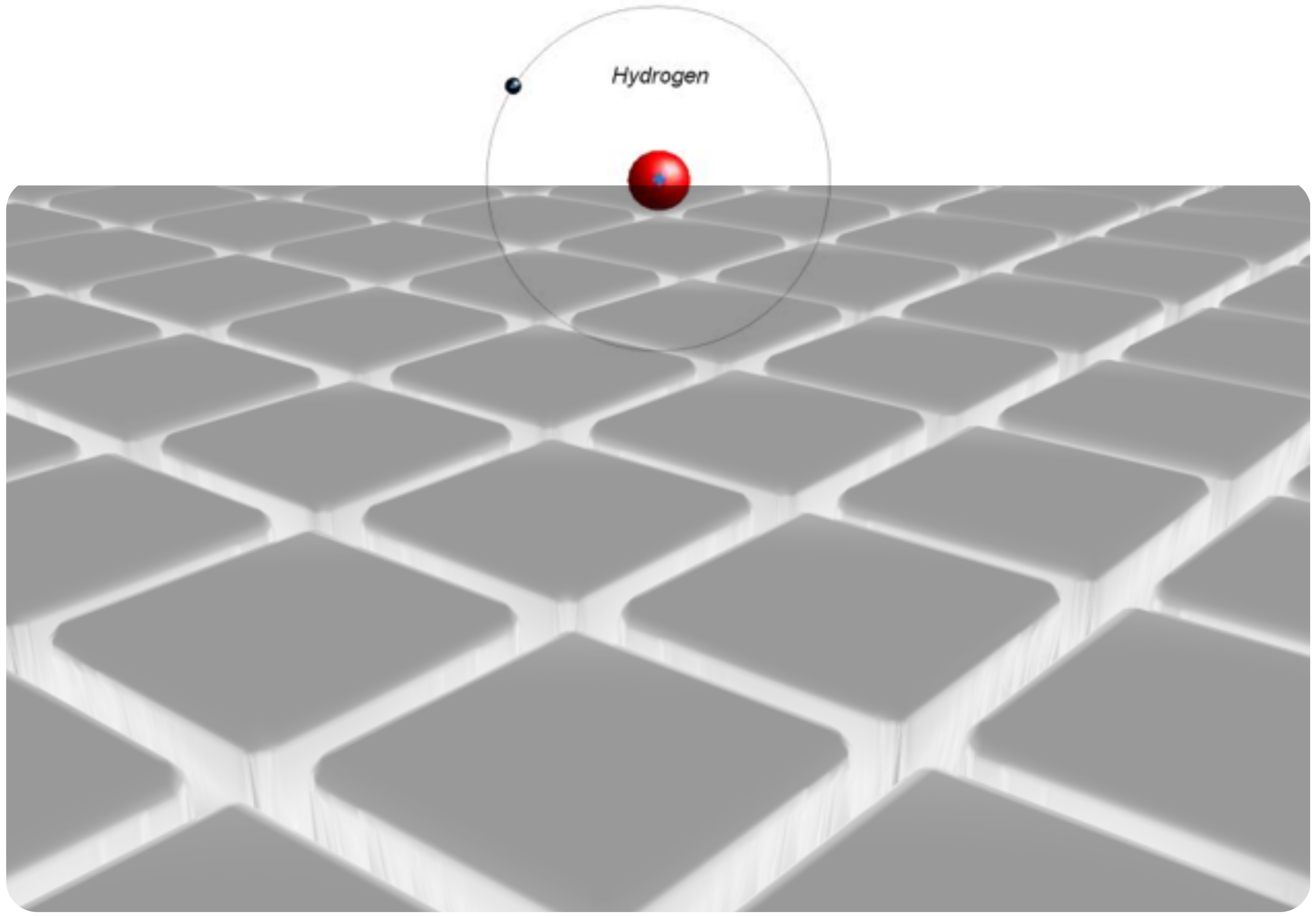
Visit //Autodesk Labs for more information and trials of

Project Neon

Project Centaur

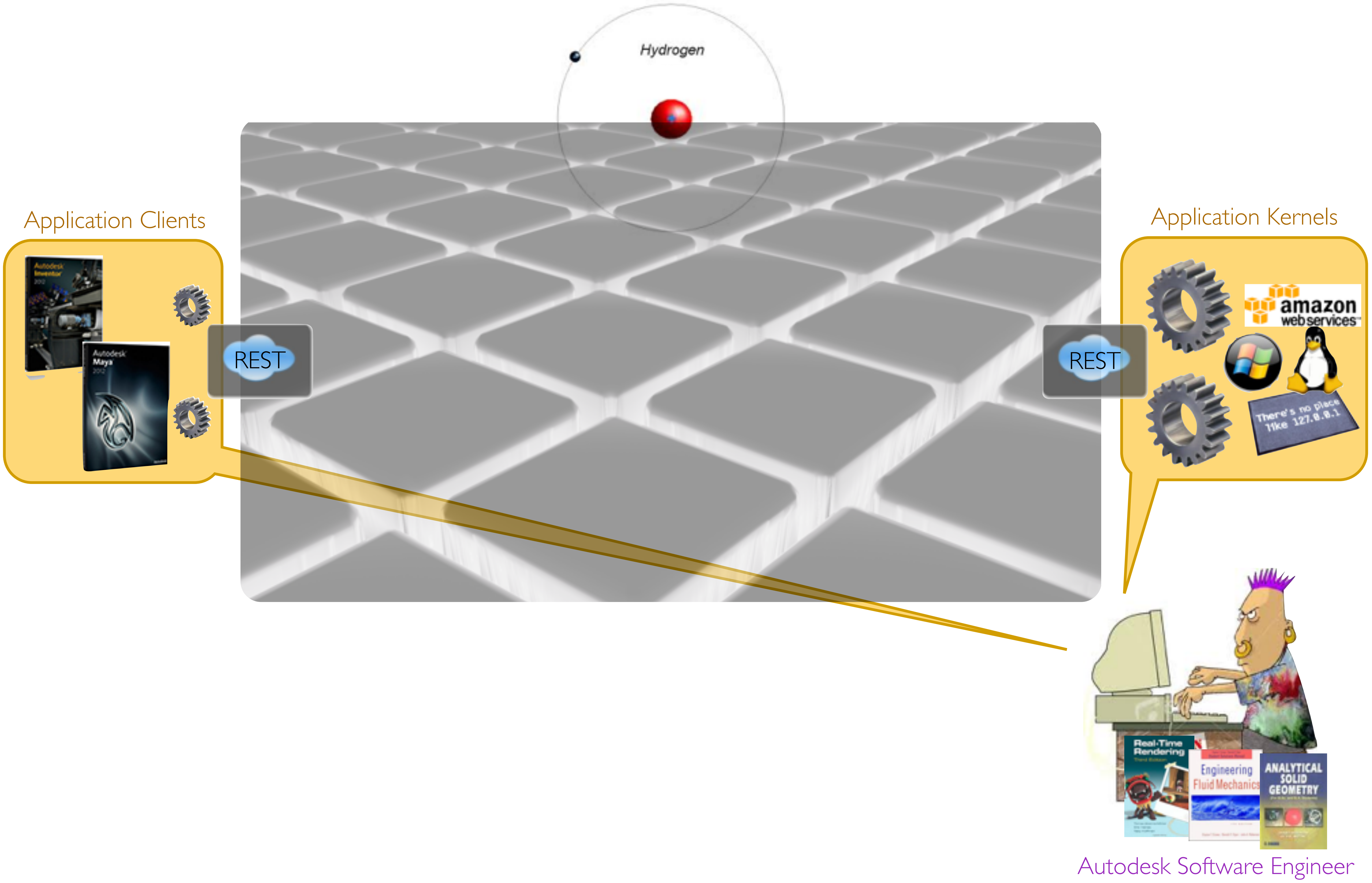
Project Photofly

A Smaller Picture

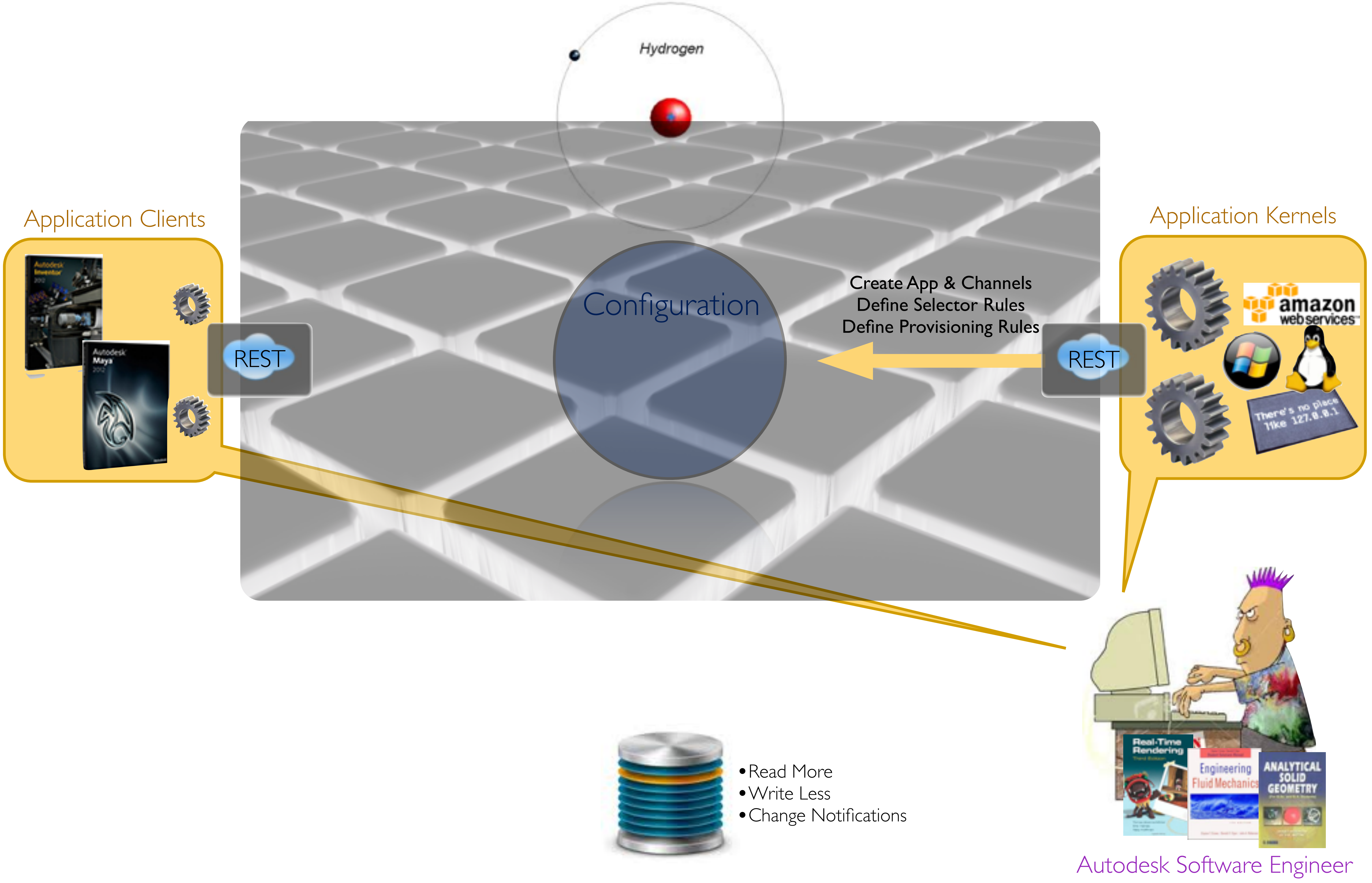


Autodesk Software Engineer

A Smaller Picture

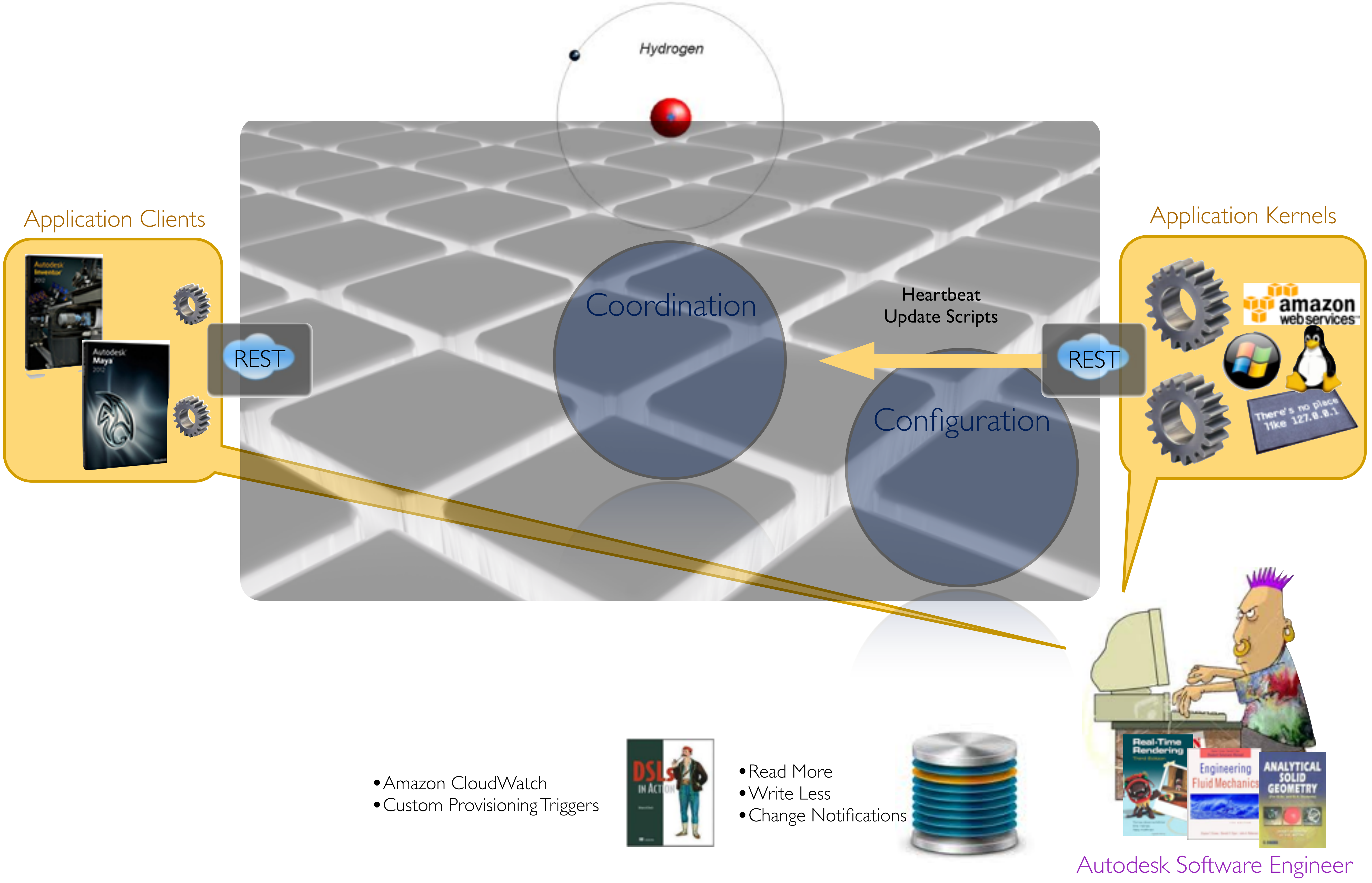


A Smaller Picture



- Read More
- Write Less
- Change Notifications

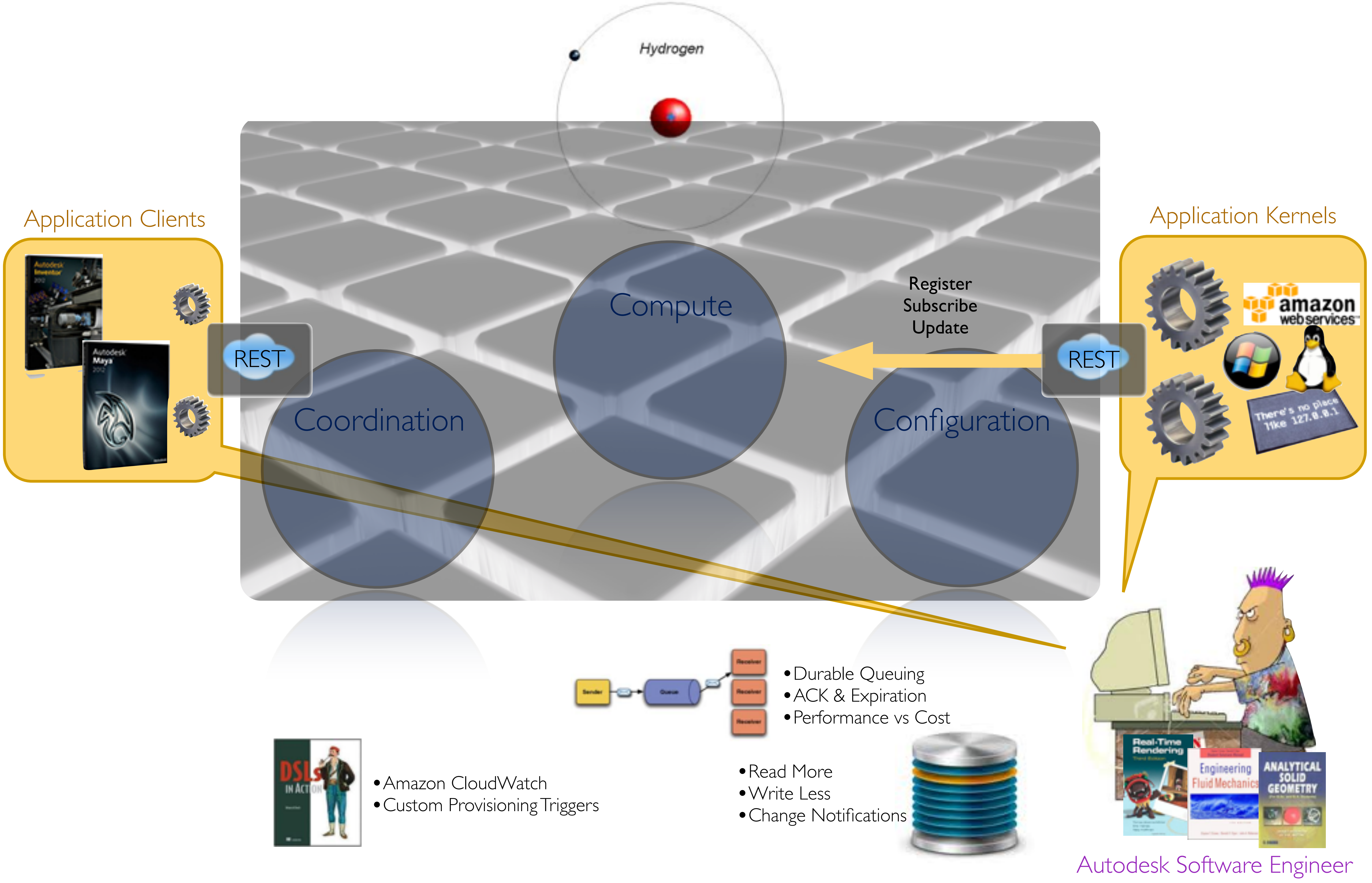
A Smaller Picture



- Amazon CloudWatch
- Custom Provisioning Triggers

- Read More
- Write Less
- Change Notifications

A Smaller Picture

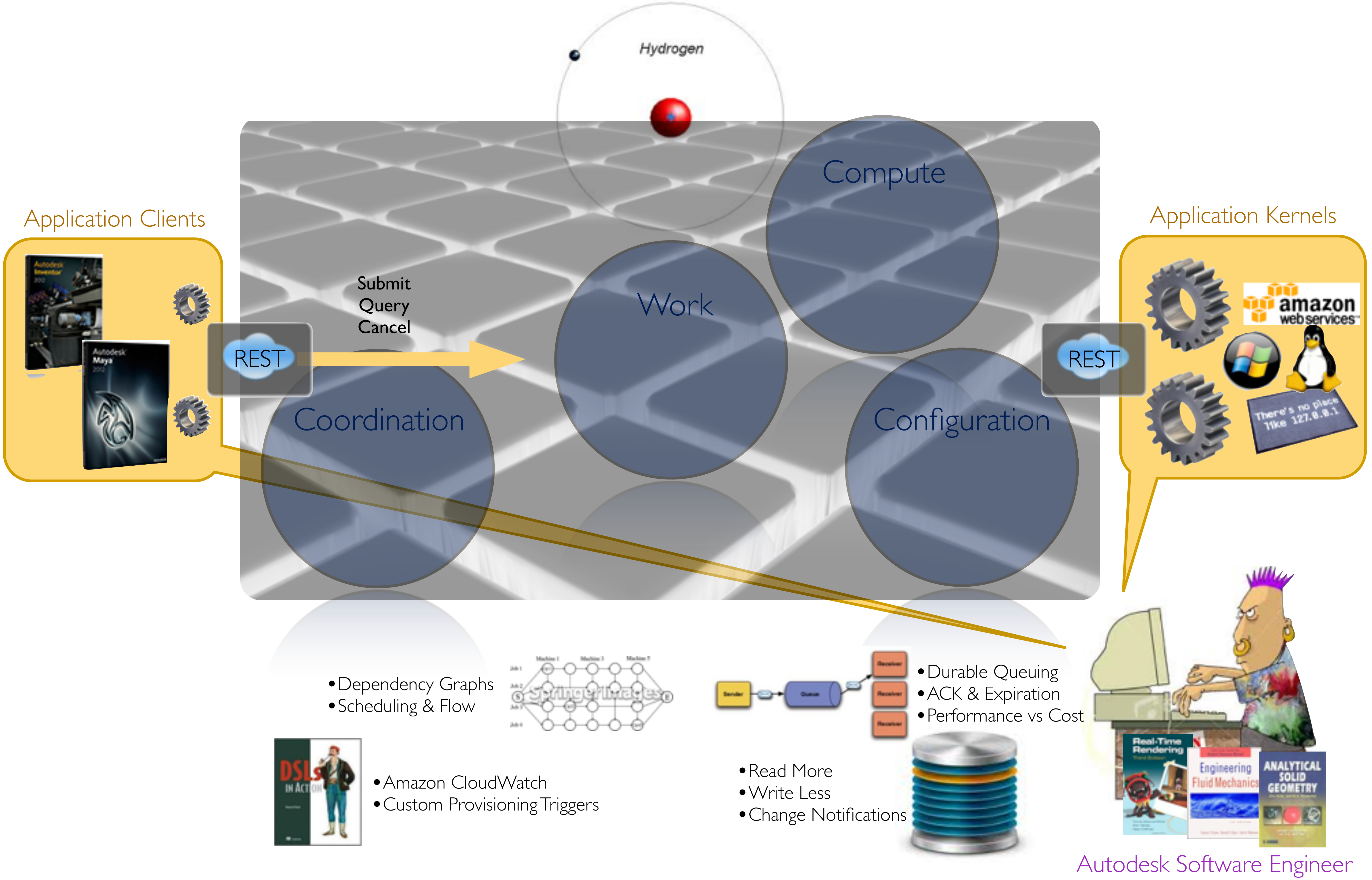


- Durable Queuing
- ACK & Expiration
- Performance vs Cost

- Amazon CloudWatch
- Custom Provisioning Triggers

- Read More
- Write Less
- Change Notifications

A Smaller Picture



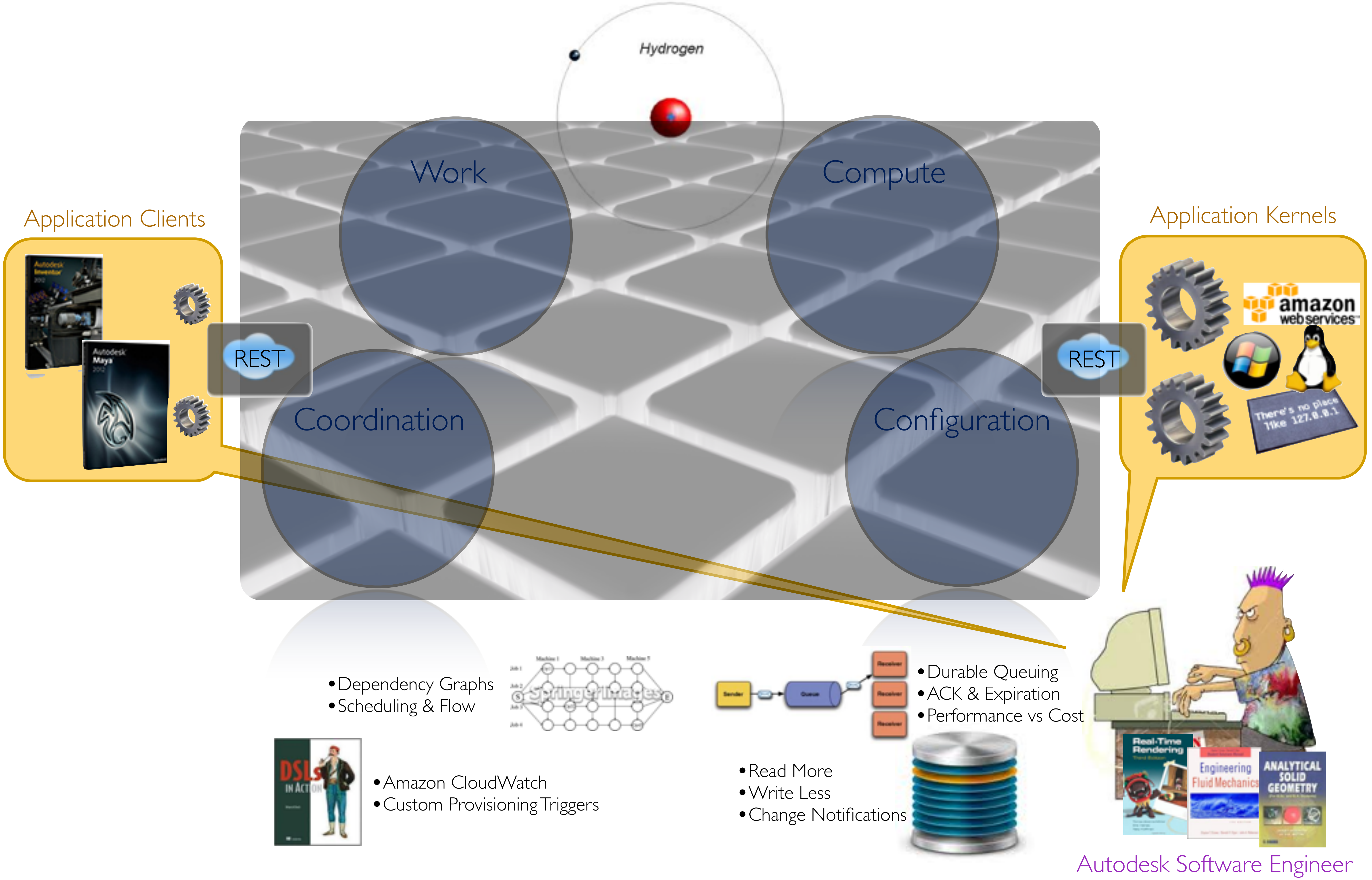
- Dependency Graphs
- Scheduling & Flow

- Durable Queuing
- ACK & Expiration
- Performance vs Cost

- Amazon CloudWatch
- Custom Provisioning Triggers

- Read More
- Write Less
- Change Notifications

A Smaller Picture



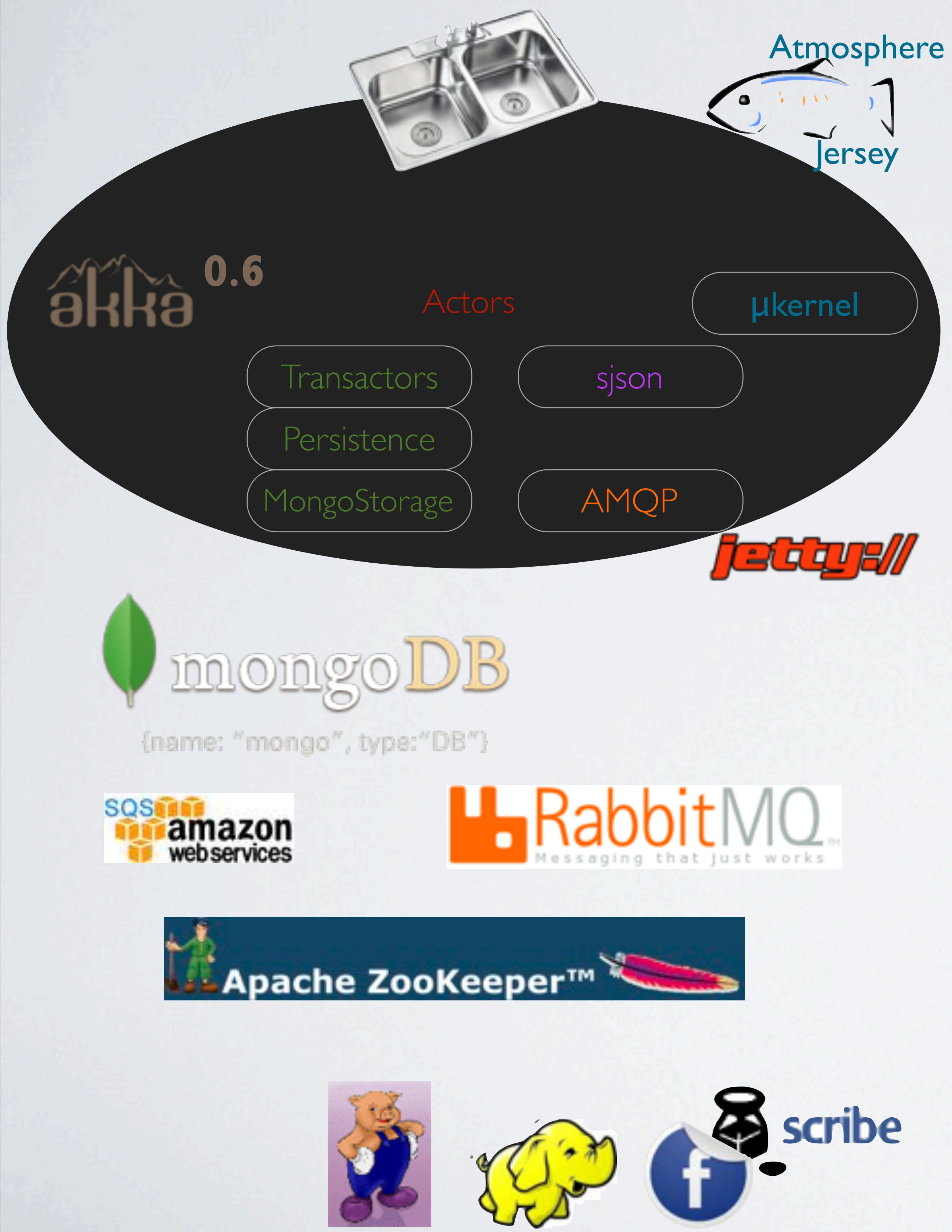
- Dependency Graphs
- Scheduling & Flow

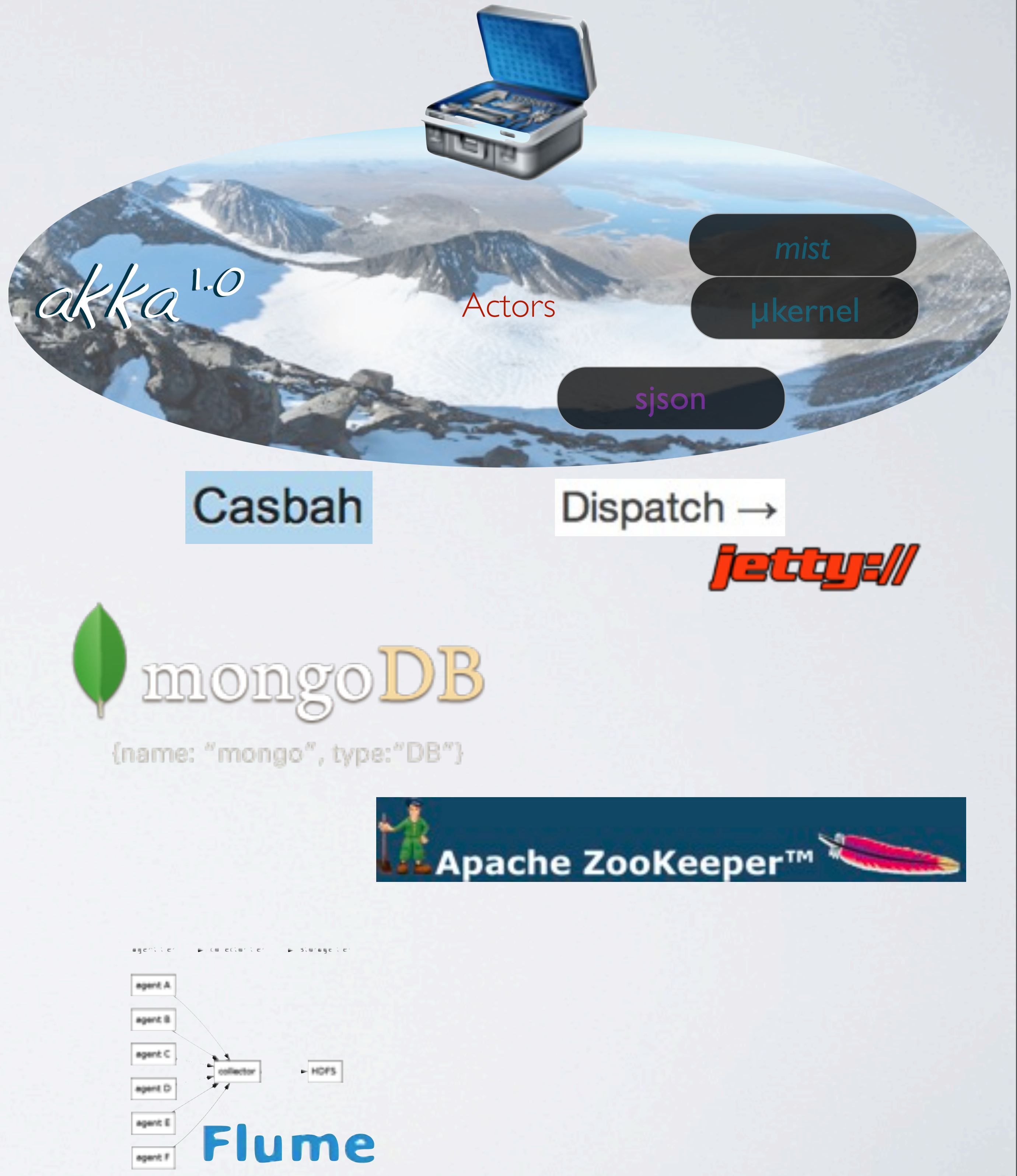
- Amazon CloudWatch
- Custom Provisioning Triggers

- Durable Queuing
- ACK & Expiration
- Performance vs Cost

- Read More
- Write Less
- Change Notifications

Saturation Simple Things





So how does Akka help?

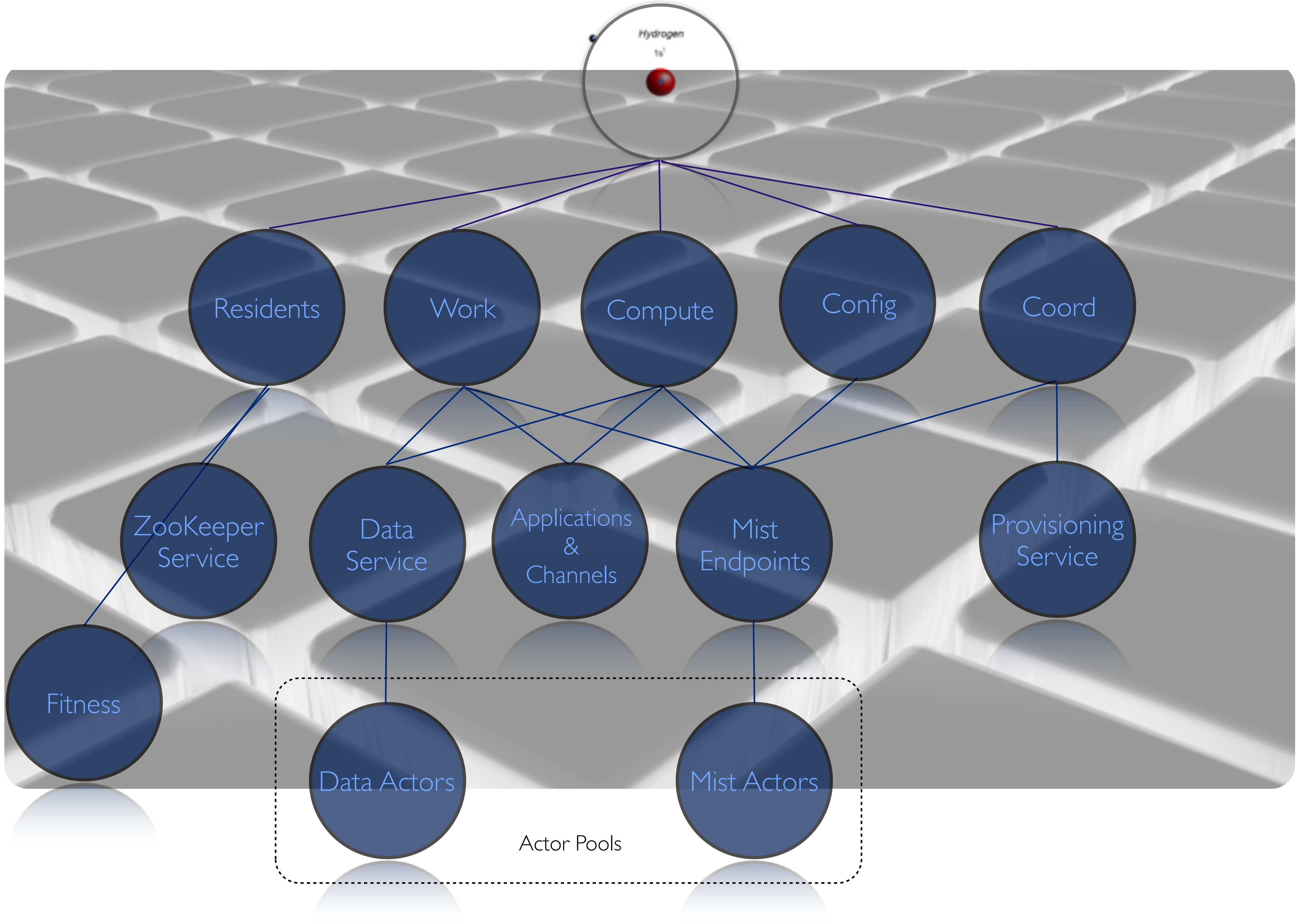
- Actors make it easy to reason about concurrency
- Supervisors make it easy to compose fault-tolerant services
- Both it make easy to distribute functionality at the right scope

- The code, the team and the community are rock solid
- Zero production issues with the core offering

- Native Scala API to leverage power of the language

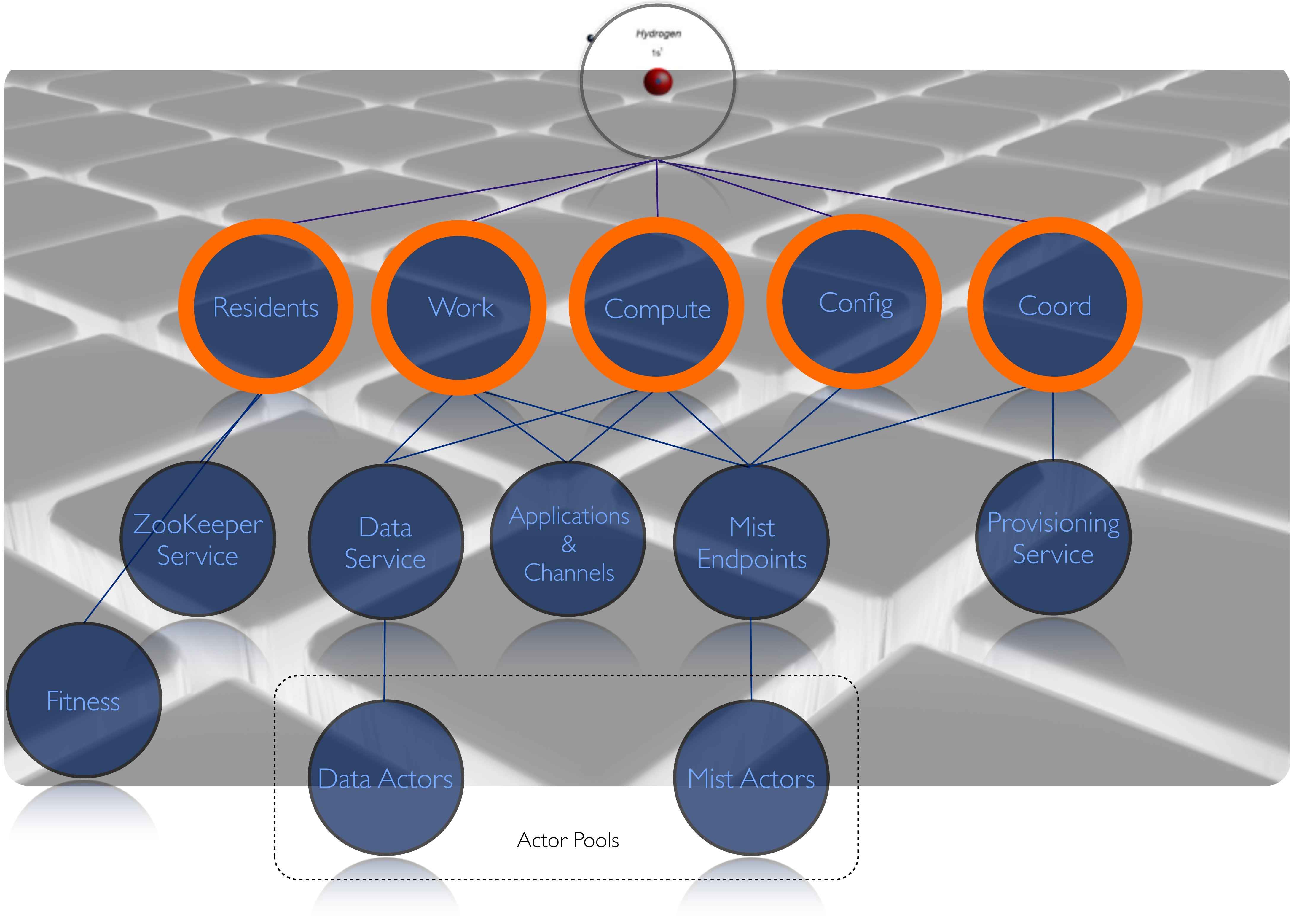
- 2 Examples...

Supervised Services



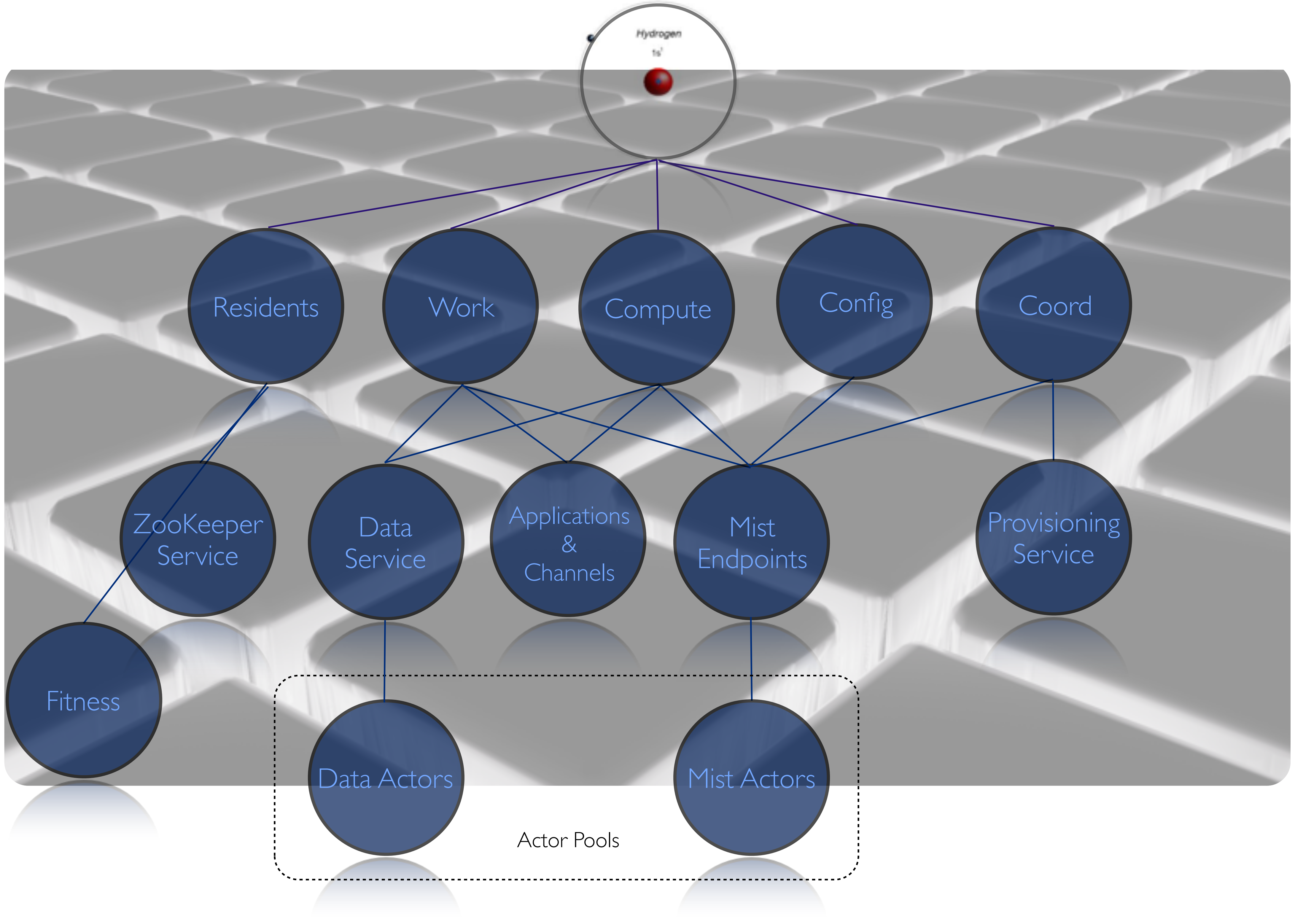
Supervised Services

Dev Local



Supervised Services

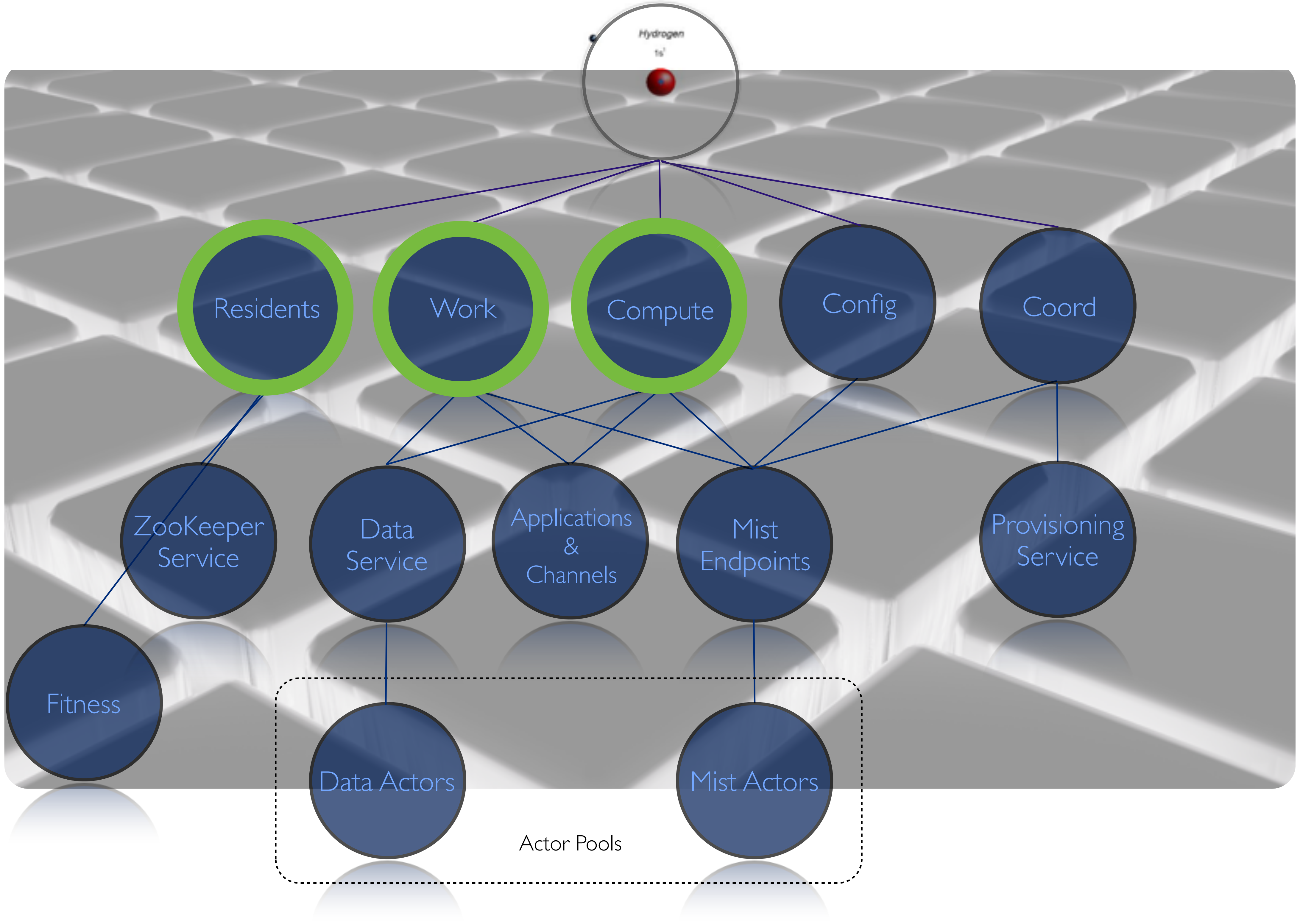
Dev Local



Supervised Services

Dev Local

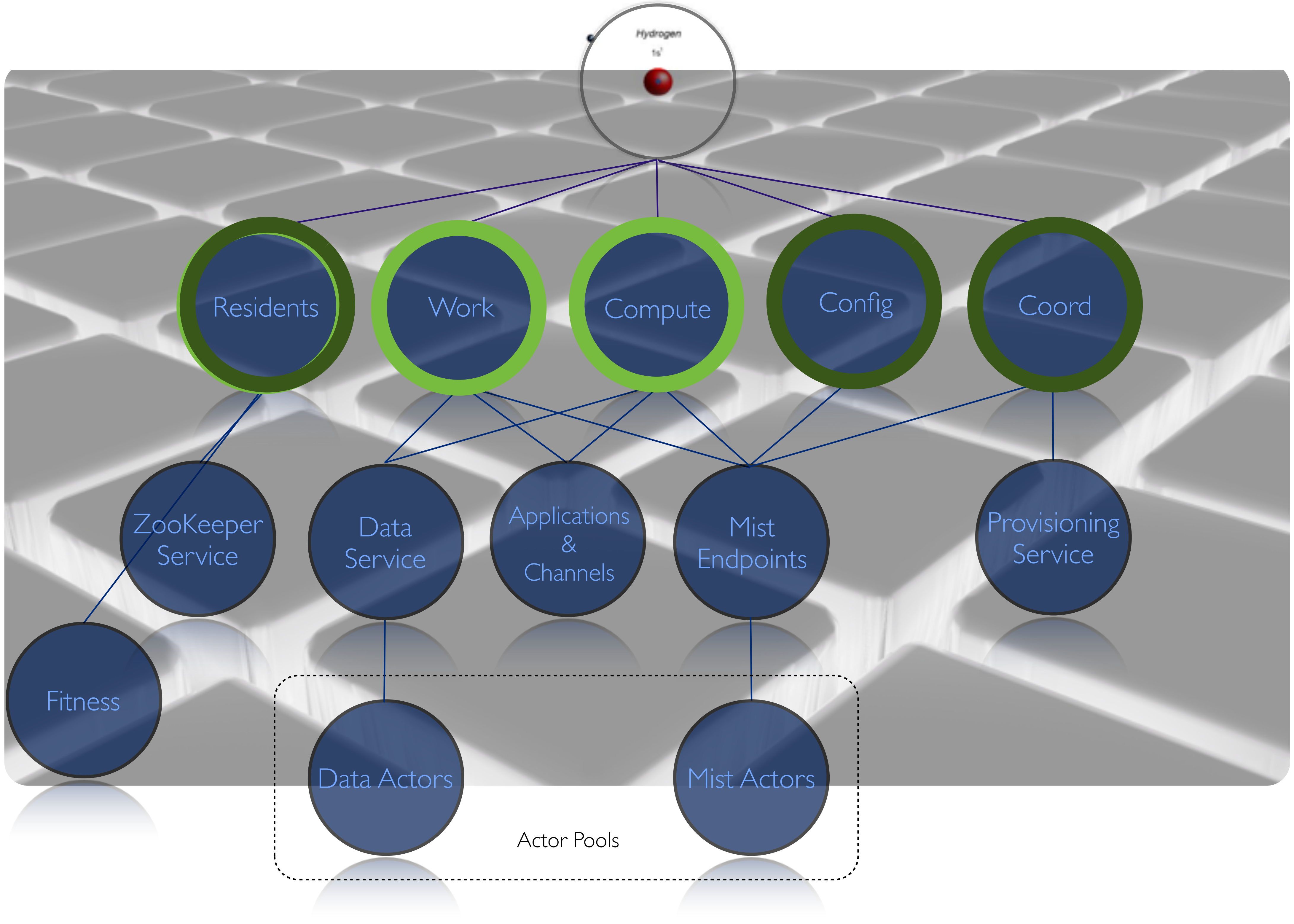
Dev EC2 Apps



Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

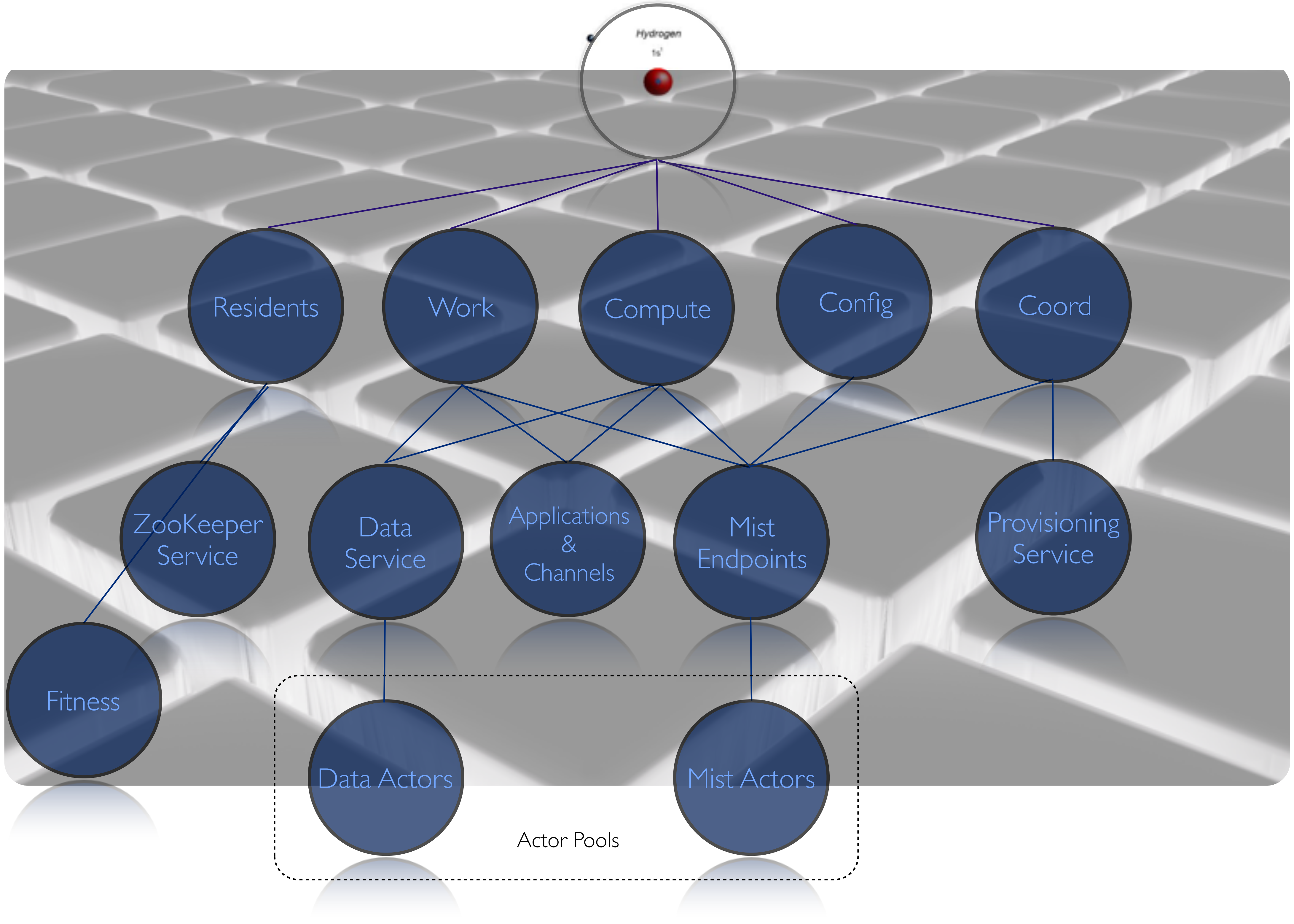


Supervised Services

Dev Local

Dev EC2 Apps

Dev EC2 Control

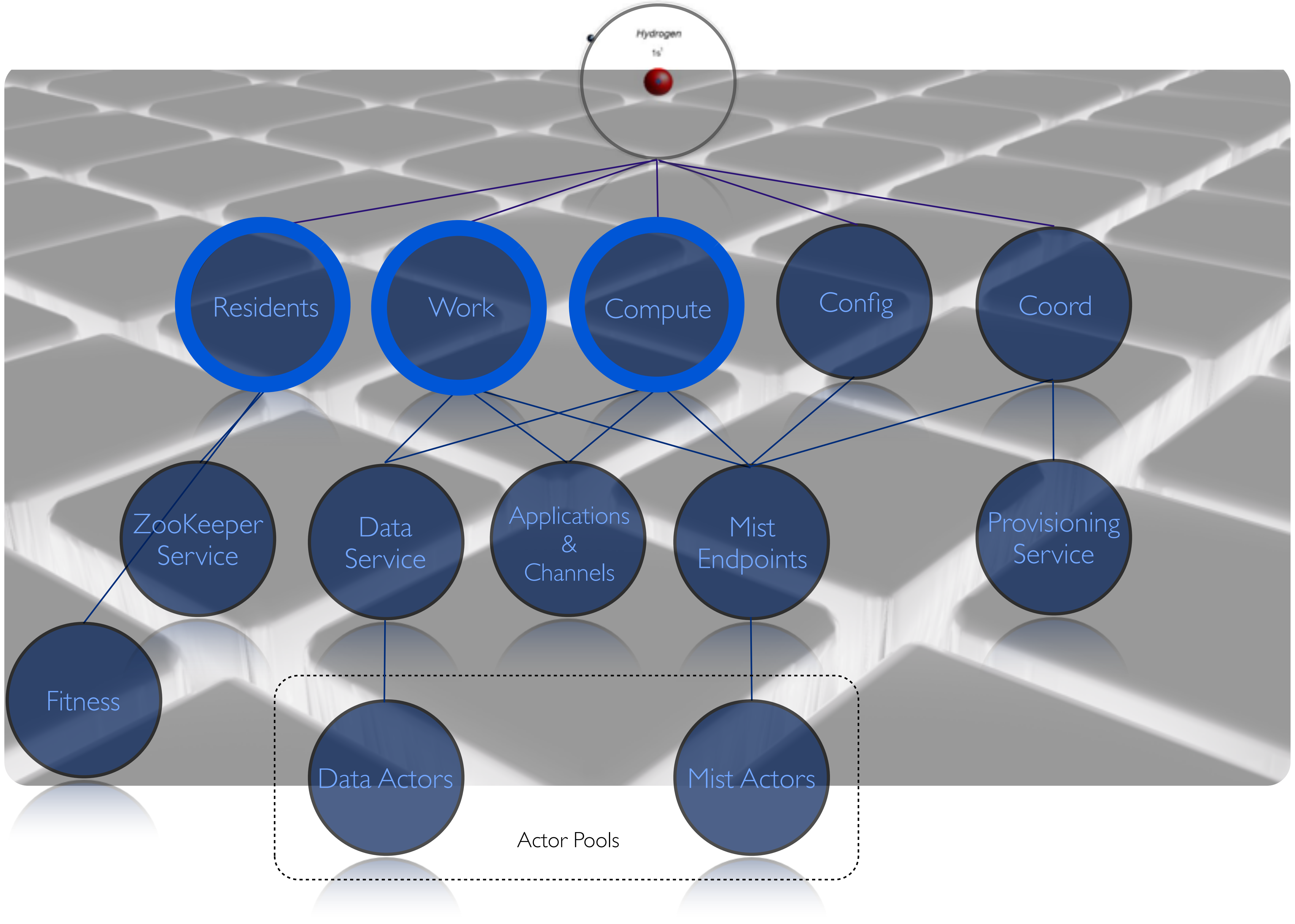


Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps

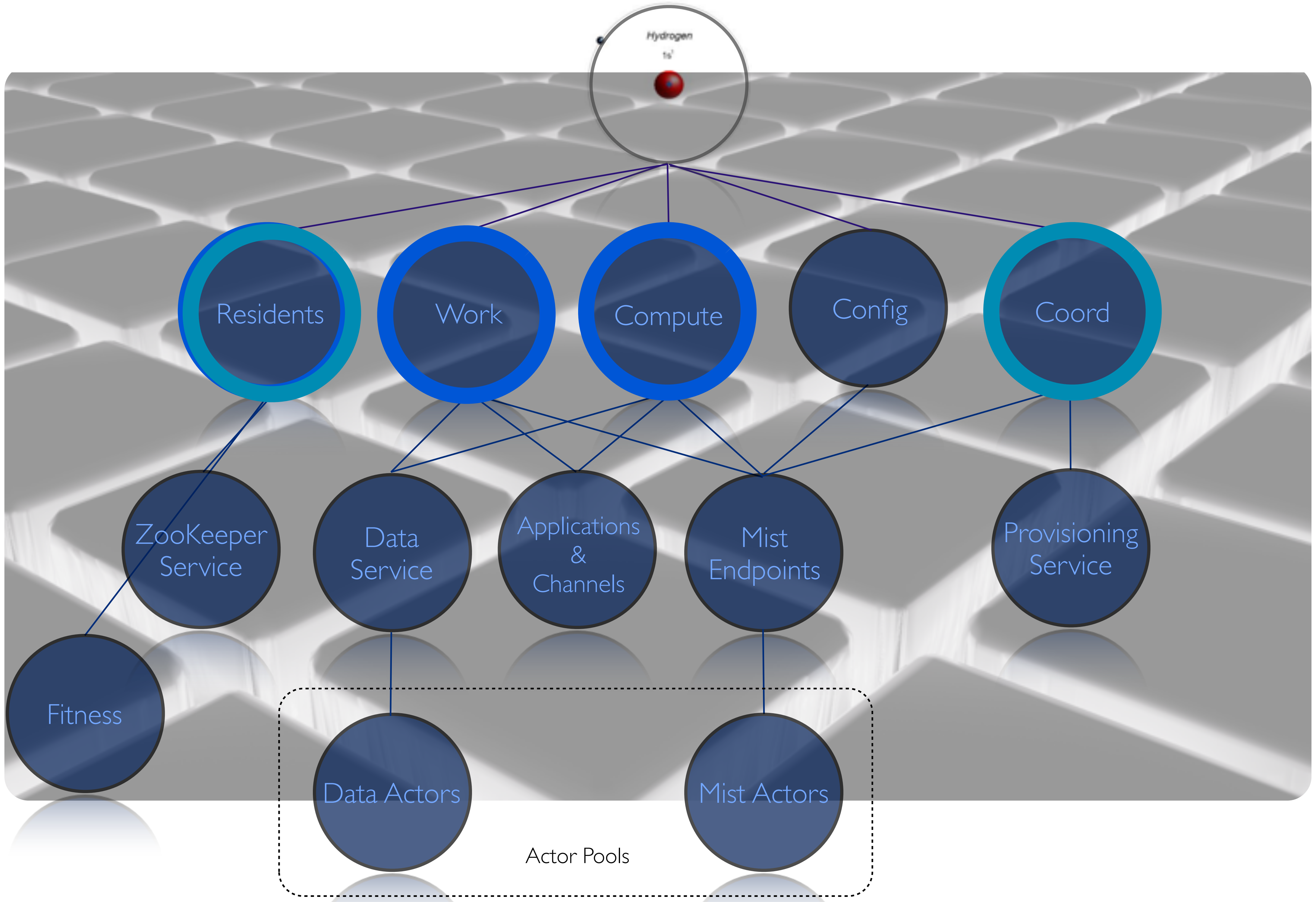


Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps
Staging Coord

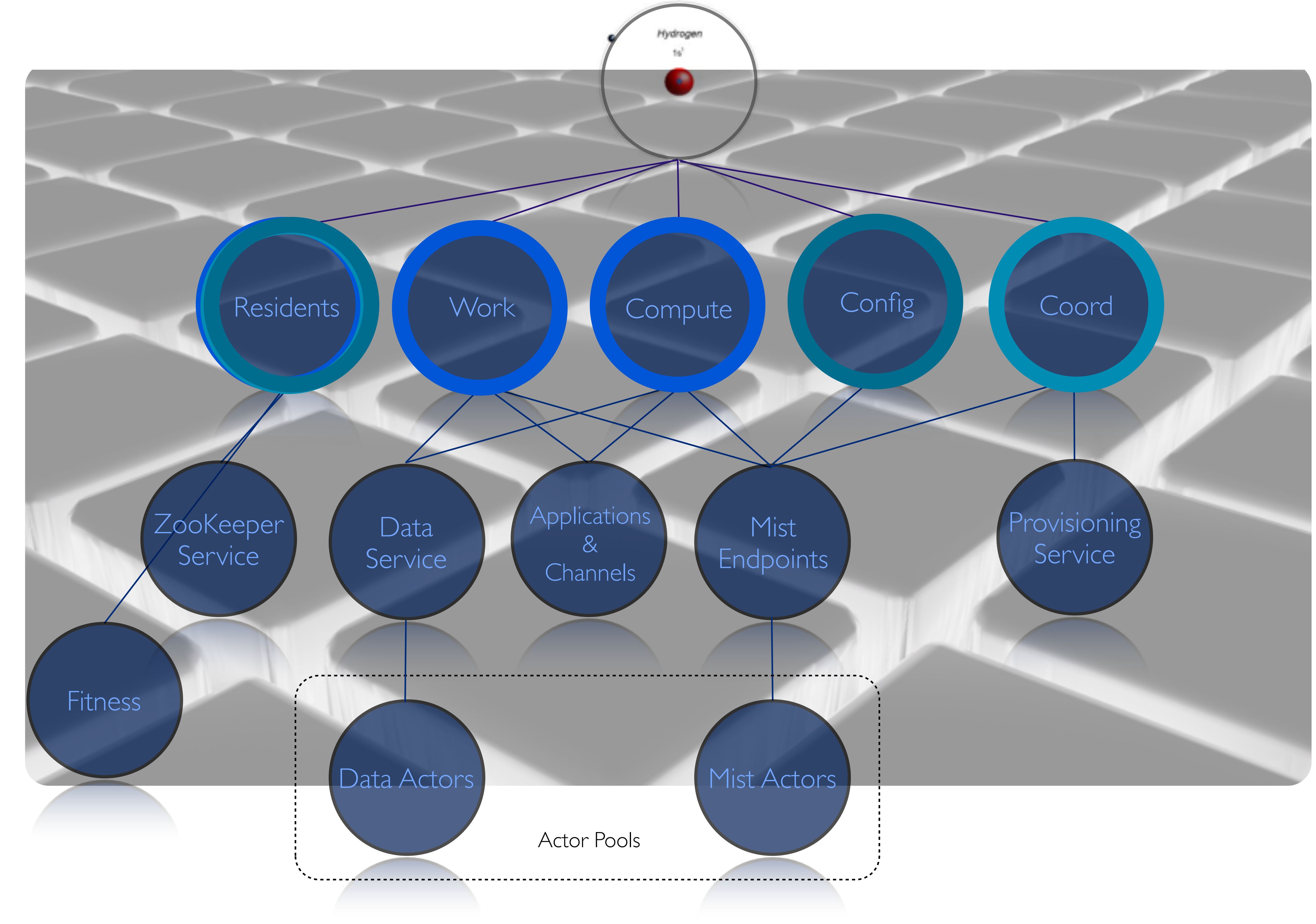


Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps
Staging Coord
Staging Config

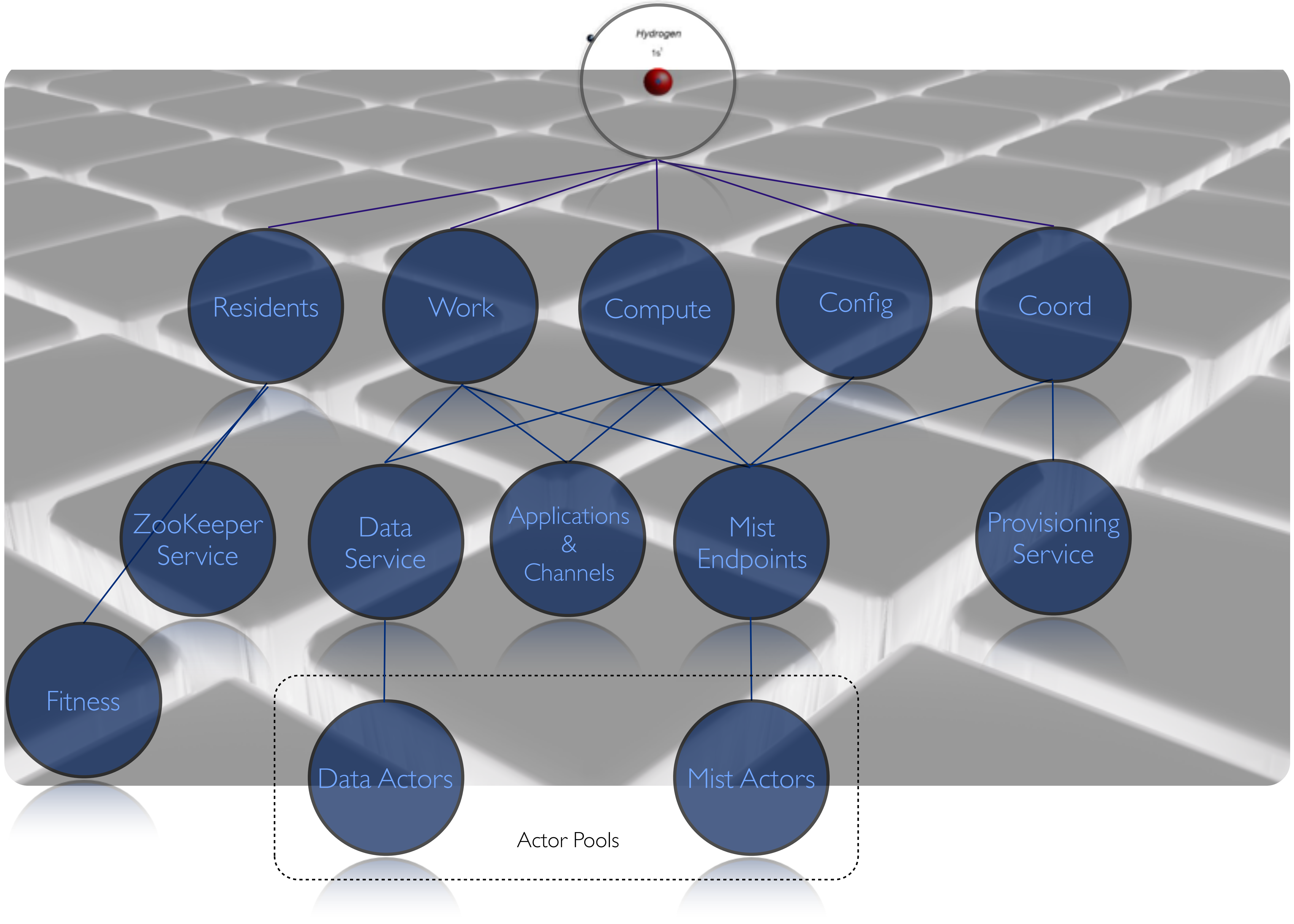


Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps
Staging Coord
Staging Config



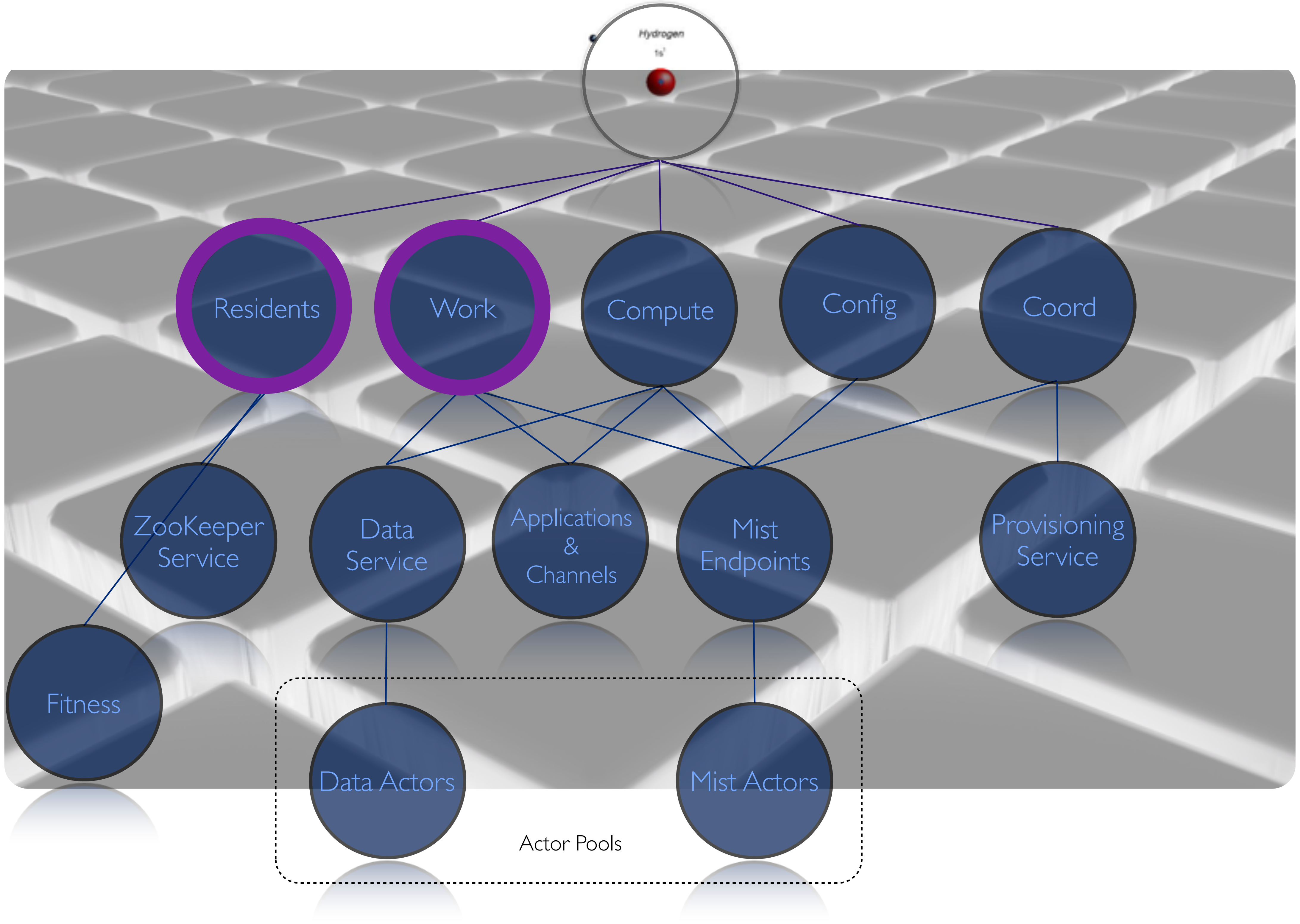
Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps
Staging Coord
Staging Config

Production Front End
(Client Apps)



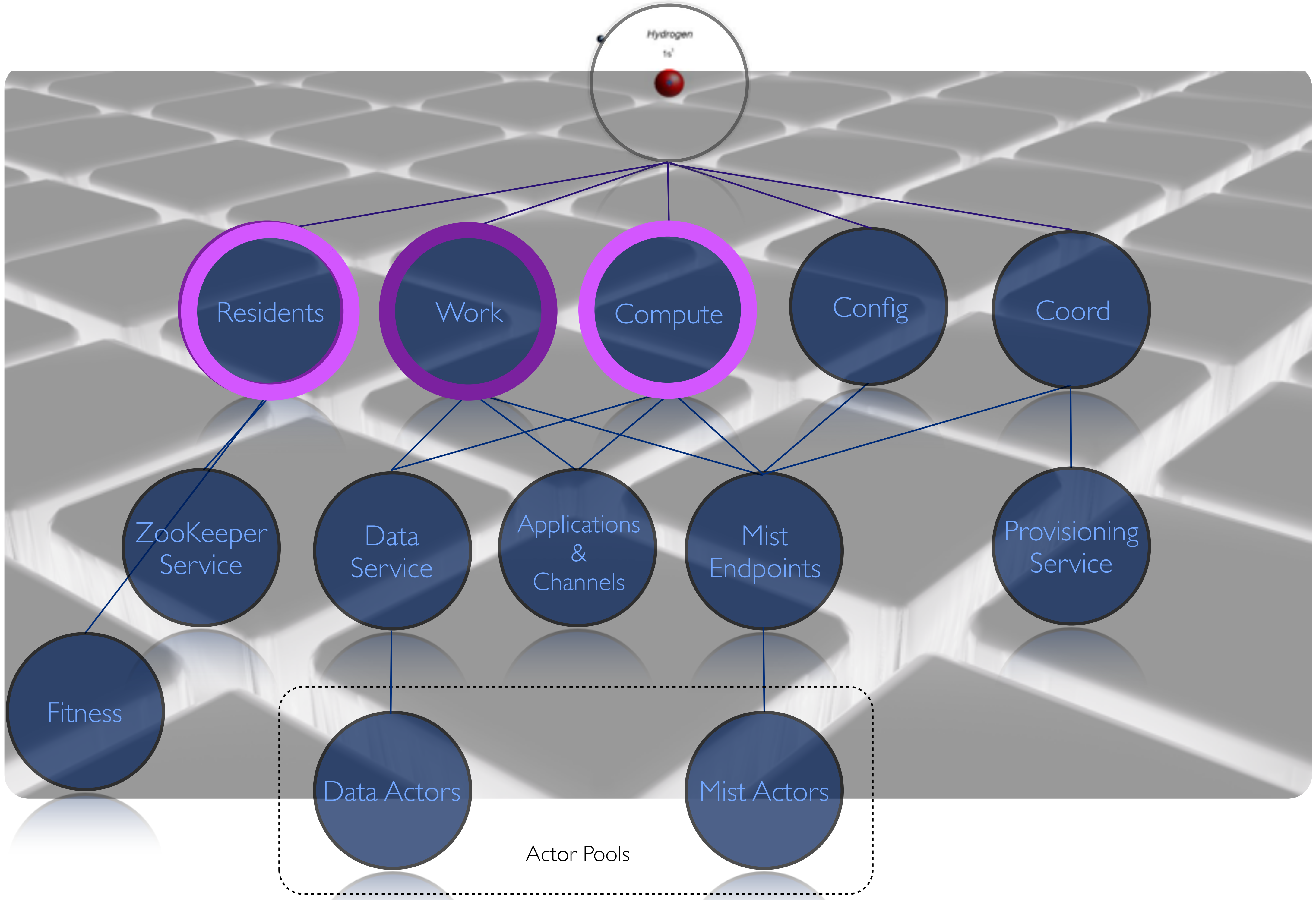
Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps
Staging Coord
Staging Config

Production Front End
(Client Apps)
Production Back End
(Worker Apps)



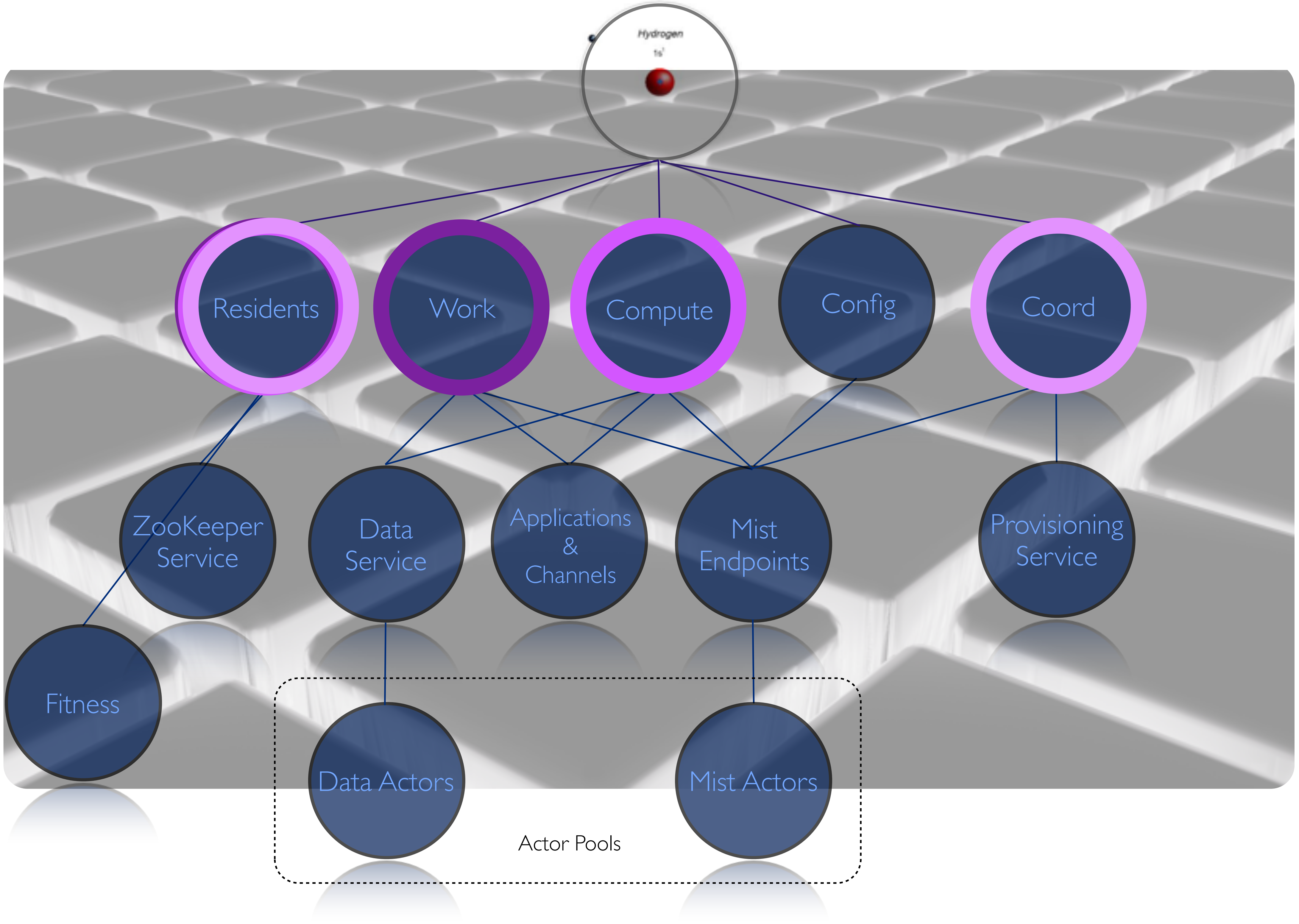
Supervised Services

Dev Local

Dev EC2 Apps
Dev EC2 Control

Staging Apps
Staging Coord
Staging Config

Production Front End
(Client Apps)
Production Back End
(Worker Apps)
Production Back End
(Worker Control)



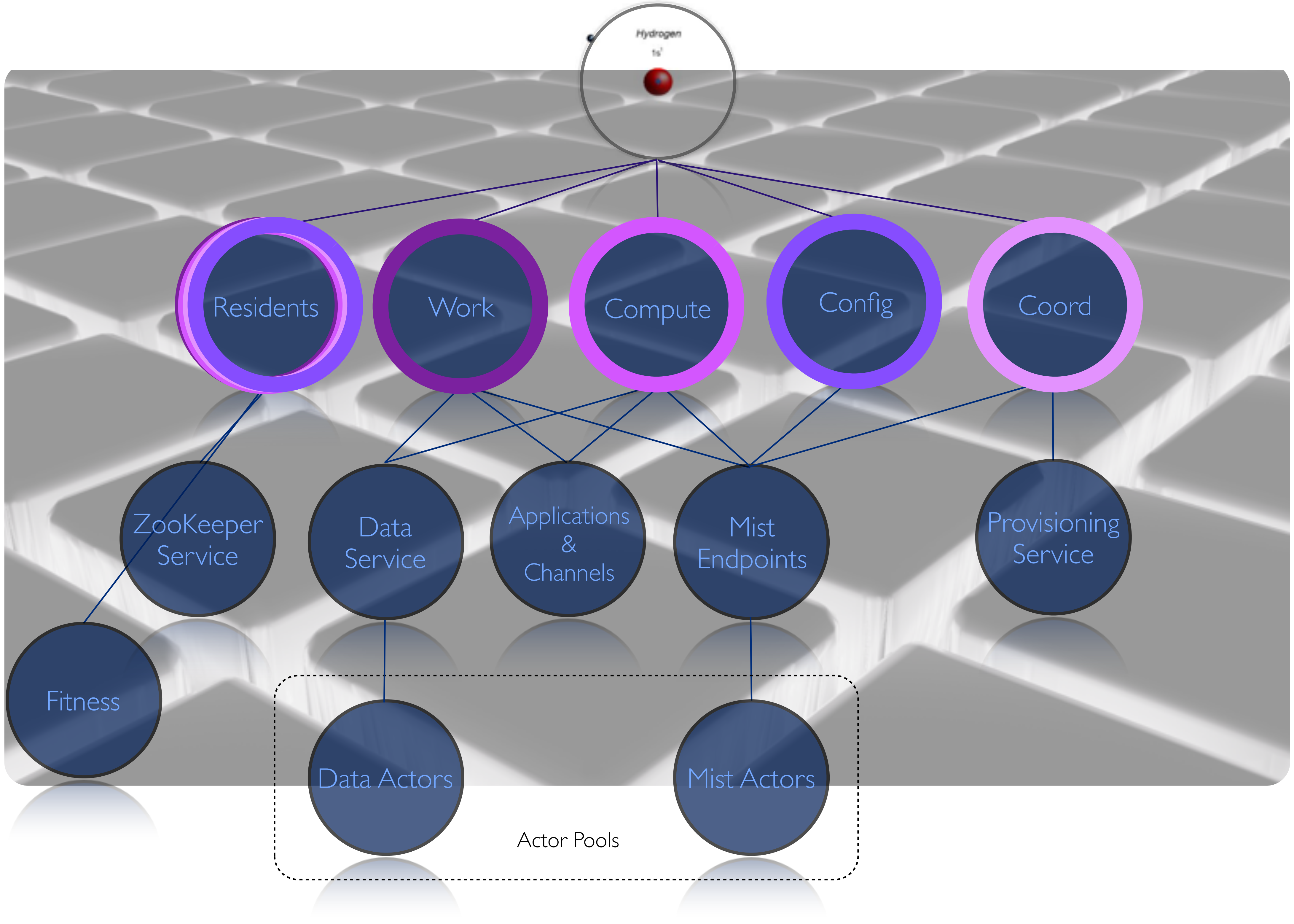
Supervised Services

Dev Local

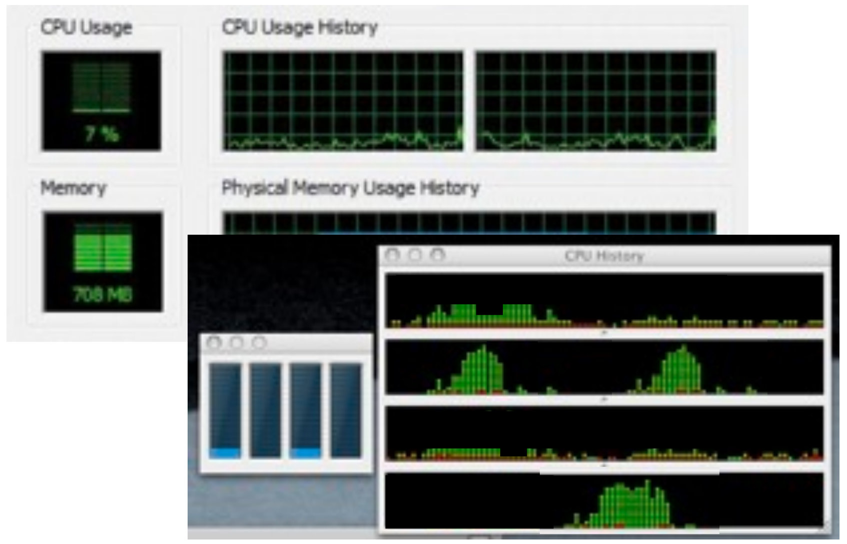
Dev EC2 Apps
Dev EC2 Control

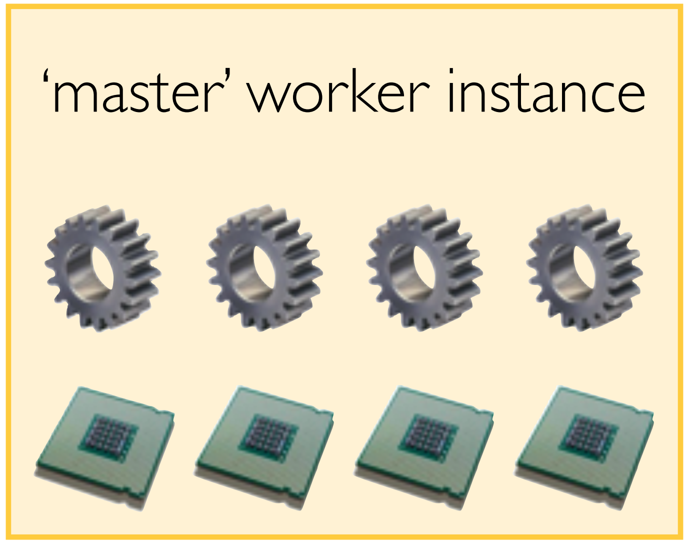
Staging Apps
Staging Coord
Staging Config

Production Front End
(Client Apps)
Production Back End
(Worker Apps)
Production Back End
(Worker Control)
Production Admin
(App Config)



Custom Provisioning





- core affinity
- moderate-long execution times
- little sustained system pressure

- local & cluster data ops
- access additional services
- cross-worker communications

Custom Provisioning

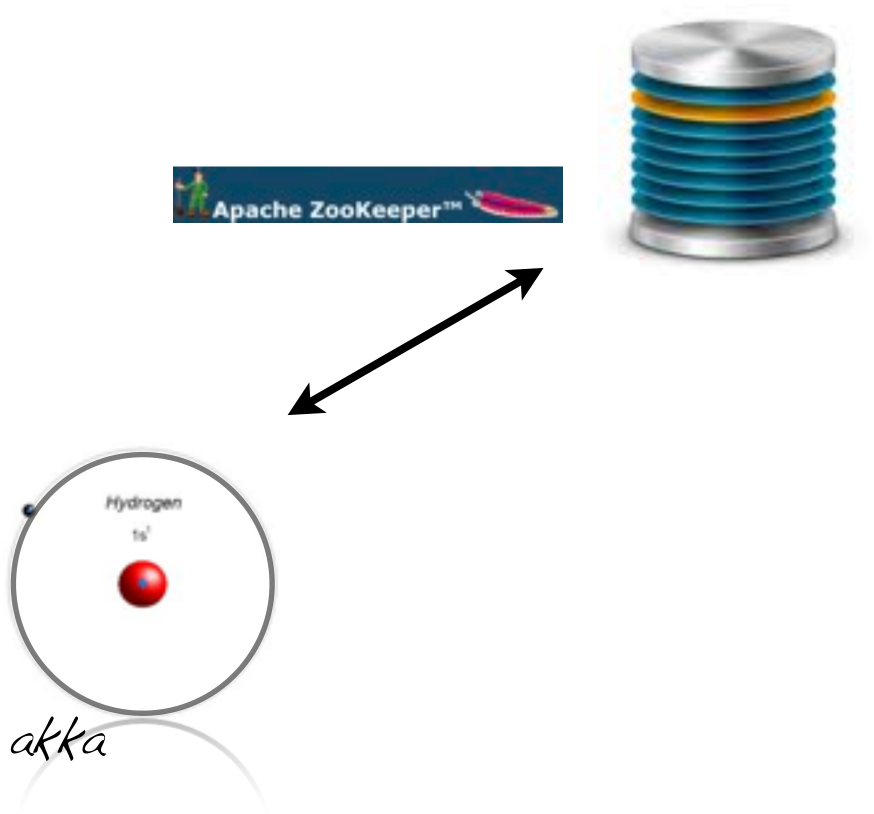
Custom Provisioning

Apache ZooKeeper™

Channel Rules

- Program
- Configurations
    - AMIs
    - Regions & AZs

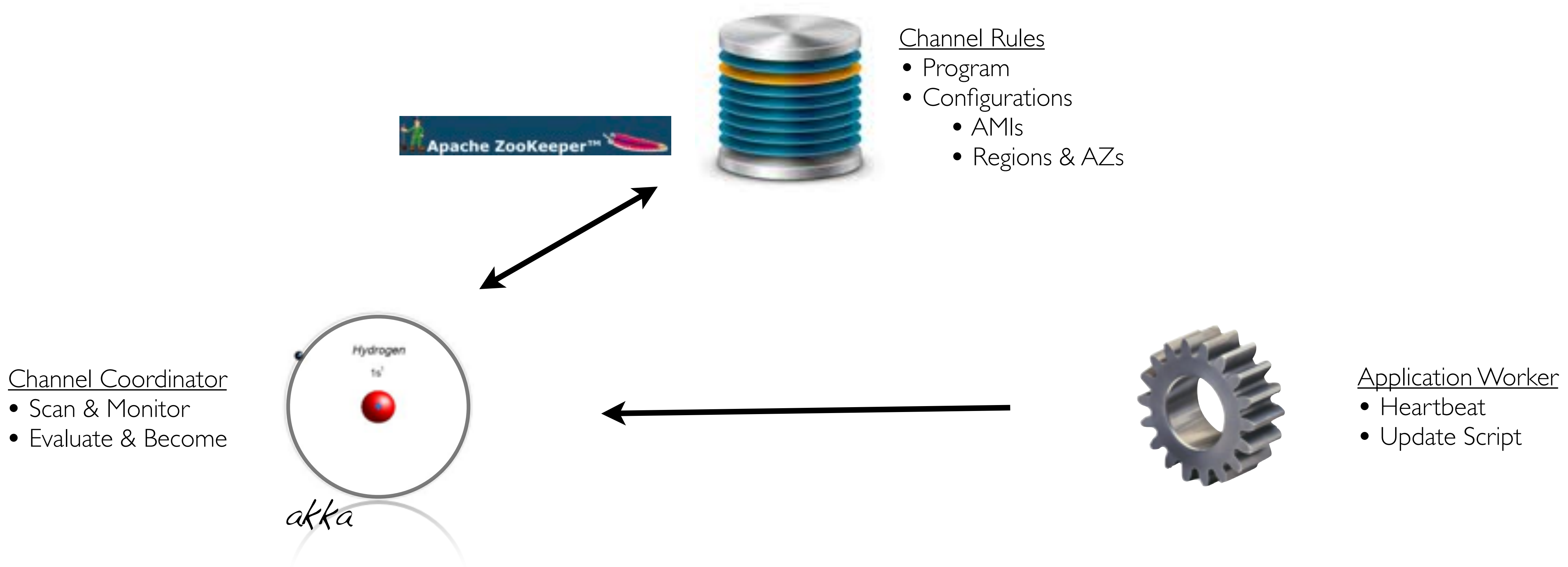
Custom Provisioning



Channel Rules
- Program
- Configurations
    - AMIs
    - Regions & AZs

Channel Coordinator
- Scan & Monitor
- Evaluate & Become

Custom Provisioning

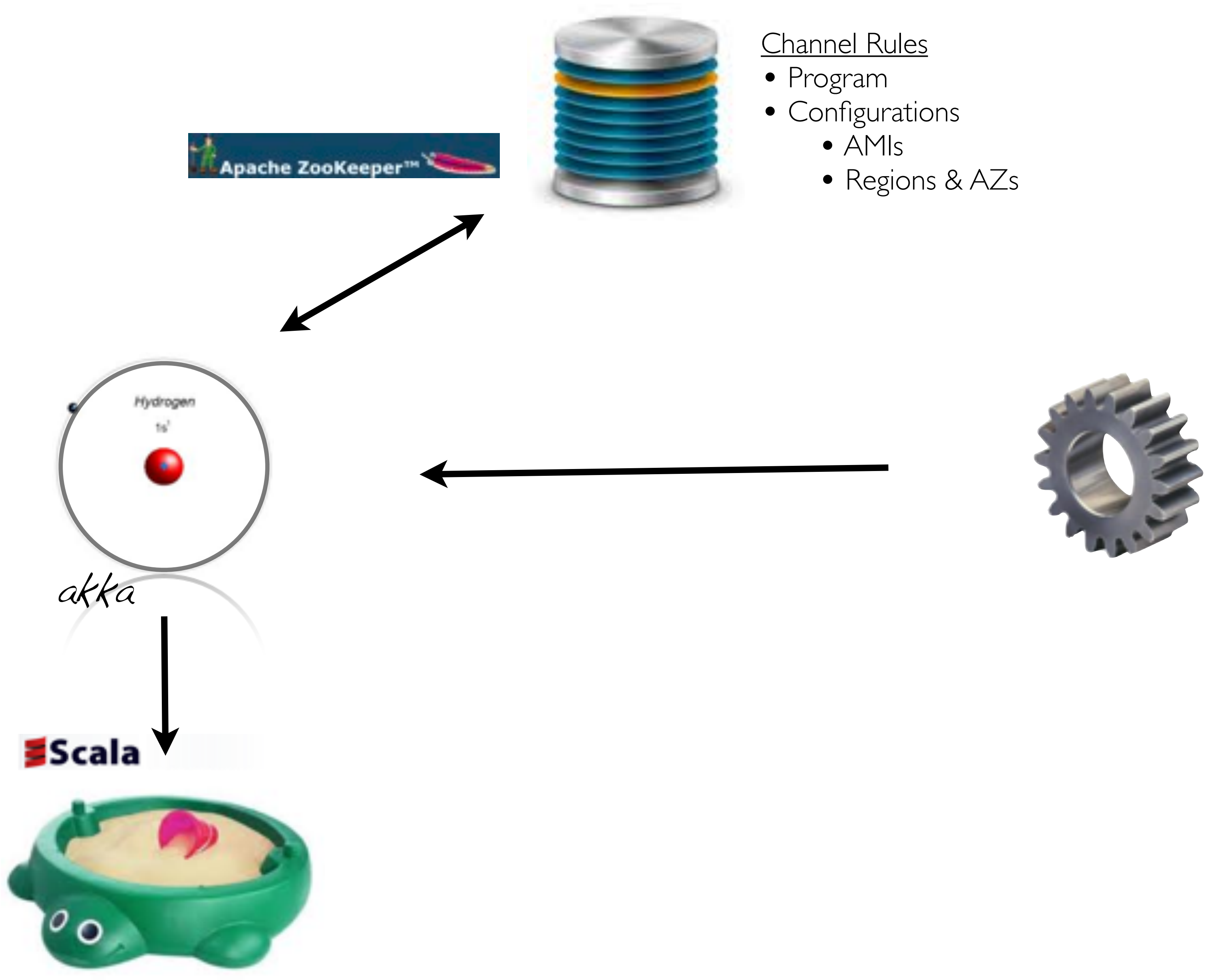


Channel Rules
- Program
- Configurations
  - AMIs
  - Regions & AZs

Channel Coordinator
- Scan & Monitor
- Evaluate & Become

Application Worker
- Heartbeat
- Update Script

Custom Provisioning



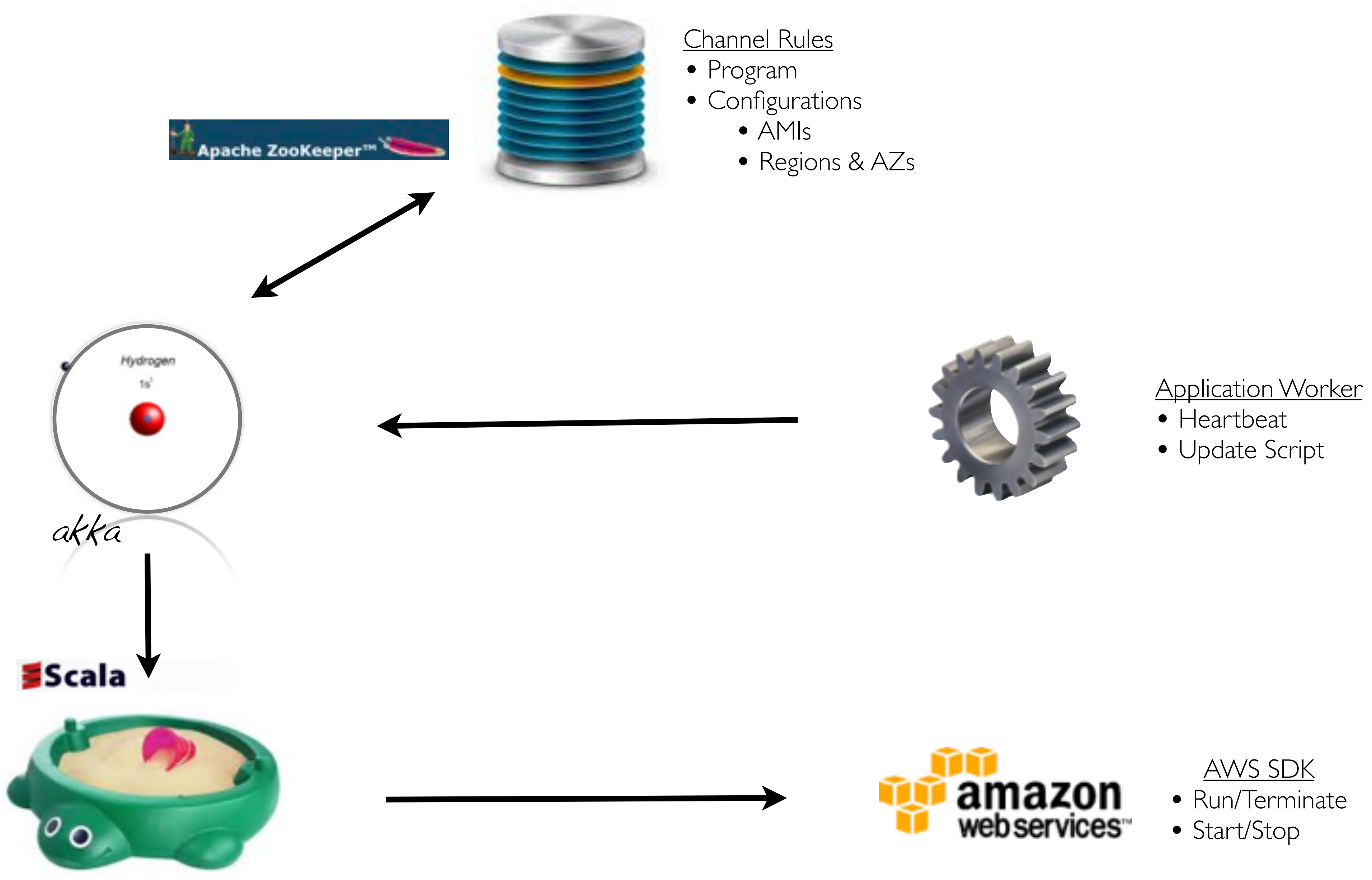
Channel Rules
- Program
- Configurations
  - AMIs
  - Regions & AZs

Channel Coordinator
- Scan & Monitor
- Evaluate & Become

Application Worker
- Heartbeat
- Update Script

Channel Coordinator
- Run Script
- Run Program

Custom Provisioning



Channel Rules
- Program
- Configurations
  - AMIs
  - Regions & AZs

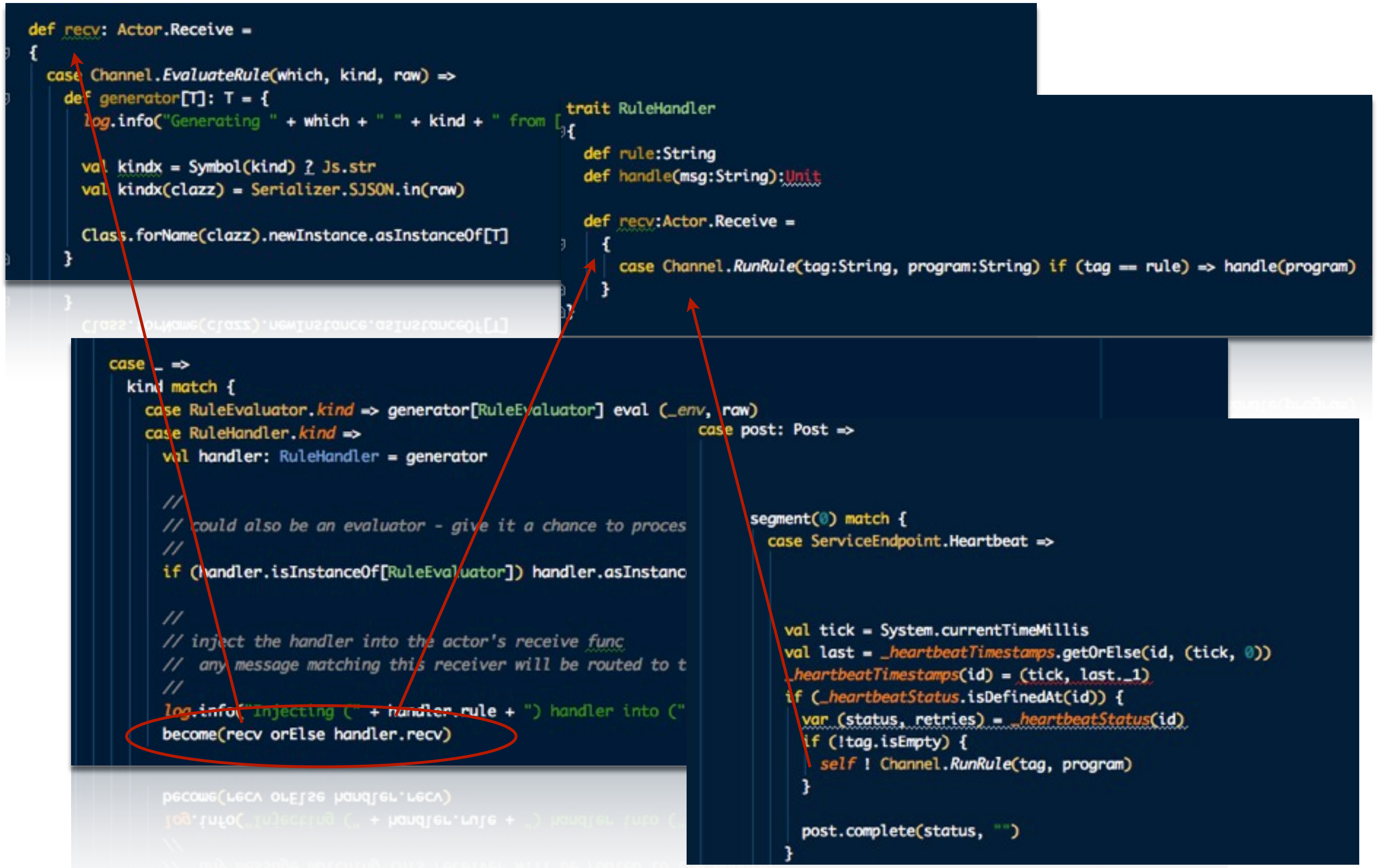
Channel Coordinator
- Scan & Monitor
- Evaluate & Become

Application Worker
- Heartbeat
- Update Script

Channel Coordinator
- Run Script
- Run Program

AWS SDK
- Run/Terminate
- Start/Stop

Custom Provisioning



```
def recv: Actor.Receive =
{
  case Channel.EvaluateRule(which, kind, raw) =>
    def generator[T]: T = {
      log.info("Generating " + which + " " + kind + " from [

      val kindx = Symbol(kind) ? Js.str
      val kindx(clazz) = Serializer.SJSON.in(raw)

      Class.forName(clazz).newInstance.asInstanceOf[T]
    }
```

```
trait RuleHandler
{
  def rule:String
  def handle(msg:String):Unit

  def recv:Actor.Receive =
  {
    case Channel.RunRule(tag:String, program:String) if (tag == rule) => handle(program)
  }
}
```

```
  case _ =>
    kind match {
      case RuleEvaluator.kind => generator[RuleEvaluator] eval (_env, raw)
      case RuleHandler.kind =>
        val handler: RuleHandler = generator

        //
        // could also be an evaluator - give it a chance to proces
        //
        if (handler.isInstanceOf[RuleEvaluator]) handler.asInstanc

        //
        // inject the handler into the actor's receive func
        //  any message matching this receiver will be routed to t
        //
        log.info("Injecting (" + handler.rule + ") handler into ("
        become(recv orElse handler.recv)
```

```
case post: Post =>


  segment(0) match {
    case ServiceEndpoint.Heartbeat =>


      val tick = System.currentTimeMillis
      val last = _heartbeatTimestamps.getOrElse(id, (tick, 0))
      _heartbeatTimestamps(id) = (tick, last._1)
      if (_heartbeatStatus.isDefinedAt(id)) {
        var (status, retries) = _heartbeatStatus(id)
        if (!tag.isEmpty) {
          self ! Channel.RunRule(tag, program)
        }

        post.complete(status, "")
      }
```

So what is this Mist anyway?

Sept 2010

viktor: "... so how's it going with Atmo?"

me: "i'm actually rolling atmo out, i don't really need comet, i just need to delay responding. jonas offered the insight that explicitly creating a completable future and passing that around instead of the broadcaster would accomplish the same. so far it works beautifully. i've got one more service to replace and then take it for a spin. re: atmo, i just don't have the cycles any longer to try to bend it to my will."

viktor: "Ah, nice, however, that means that you're hogging a thread while waiting for the completion? ... yeah, I have spent too much time trying to bend it to my will as well, will consider dropping it in favor of either Jetty Cont/WebSockets or Netty WebSockets..."

me: "That is true and I am seeing some time outs under stress testing. Still, I'd rather have a well understood set of blocking i/o threads to worry about than a seemingly unbounded set hosing my jvm."

viktor: "I absolutely agree"

So what is this Mist anyway?

Sept 2010

viktor: "... so how's it going with Atmo?"

me: "i'm actually rolling atmo out, i don't really need comet, i just need to delay responding. jonas offered the insight that explicitly creating a completable future and passing that around instead of the broadcaster would accomplish the same. so far it works beautifully. i've got one more service to replace and then take it for a spin. re: atmo, i just don't have the cycles any longer to try to bend it to my will."

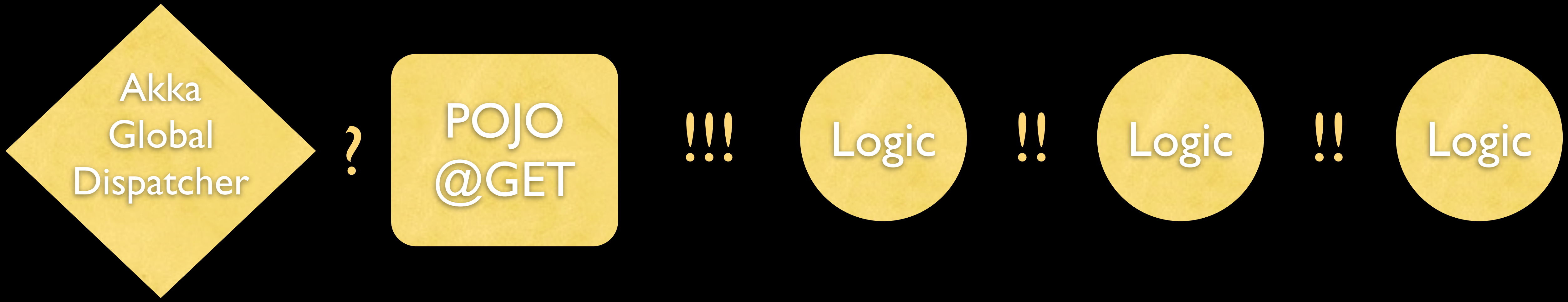
viktor: "Ah, nice, however, that means that you're hogging a thread while waiting for the completion? ... yeah, I have spent too much time trying to bend it to my will as well, will consider dropping it in favor of either Jetty Cont/WebSockets or Netty WebSockets..."

me: "That is true and I am seeing some time outs under stress testing. Still, have a well understood set of blocking i/o threads to worry about than a seemingly unbounded set hosing my jvm."

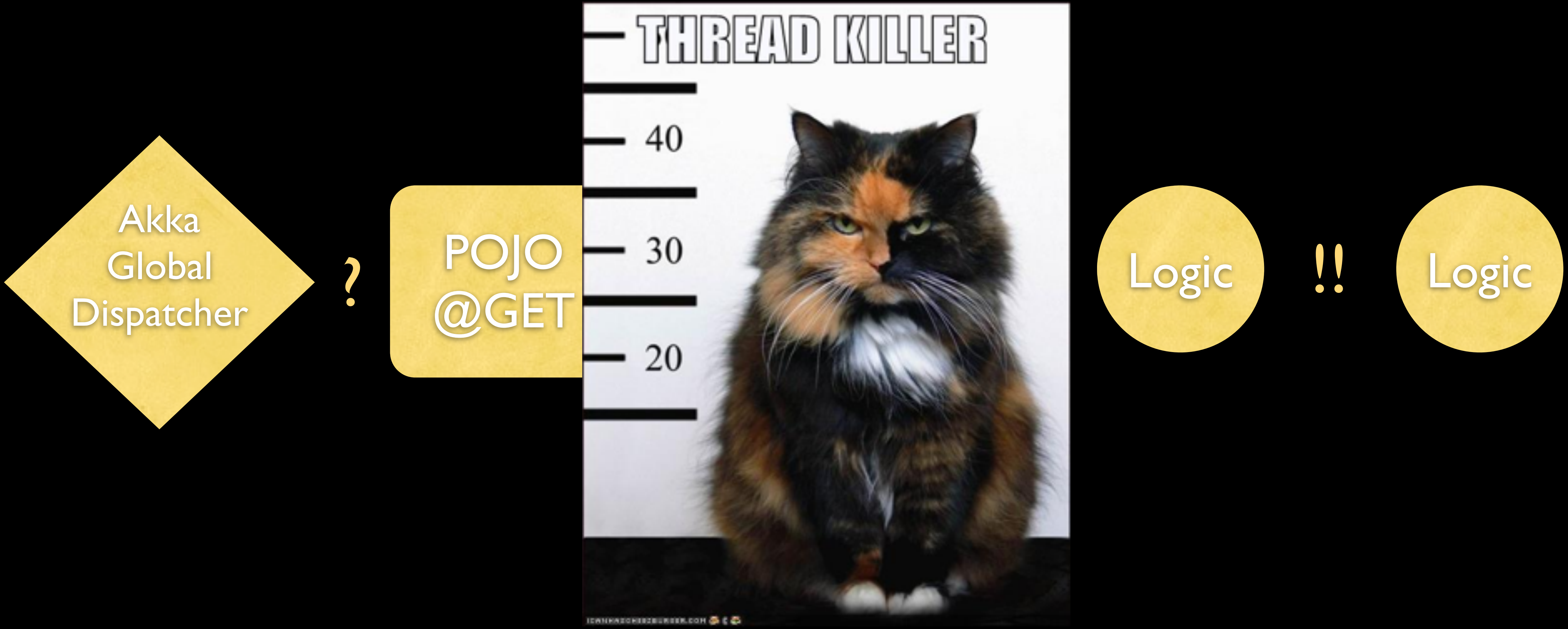
viktor: "I absolutely agree"

Mist

Mist

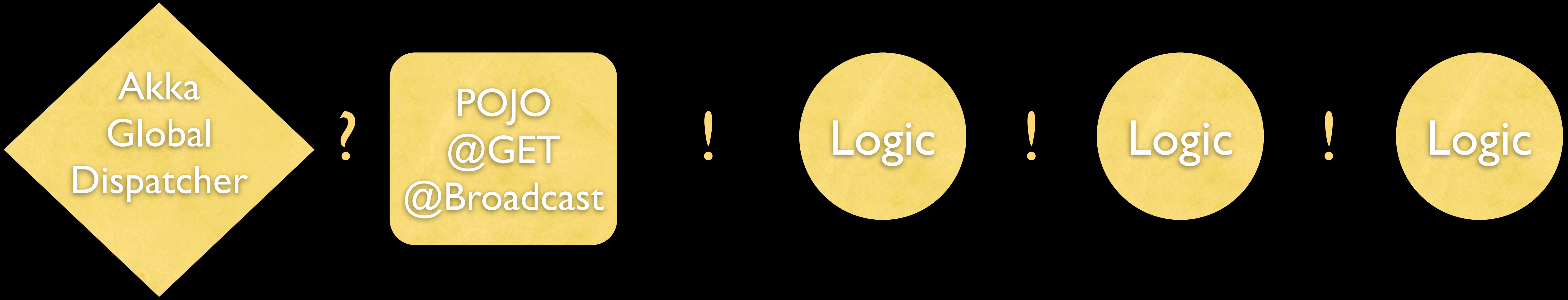


Mist

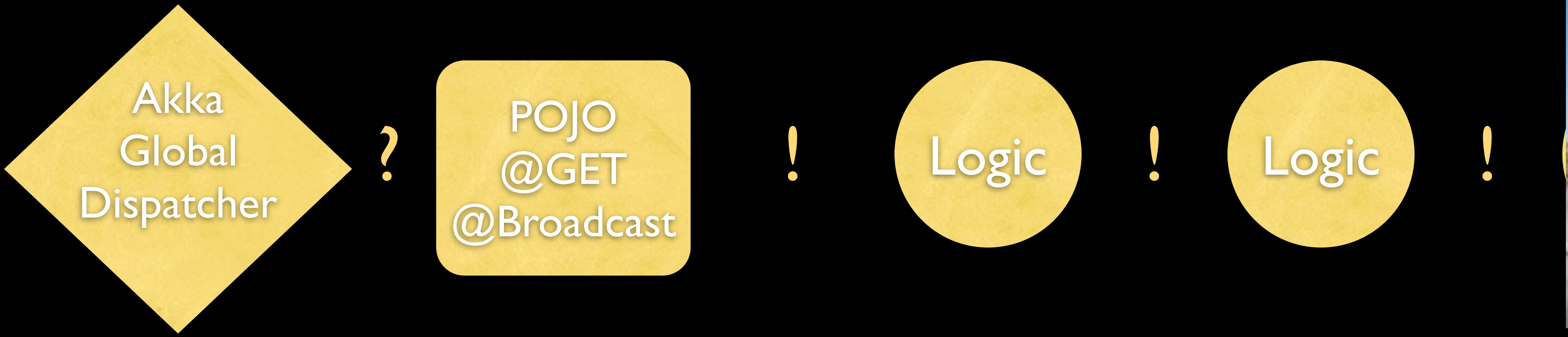

Akka Global Dispatcher
?
POJO @GET
THREAD KILLER
40
30
20
Logic
!!
Logic

Mist

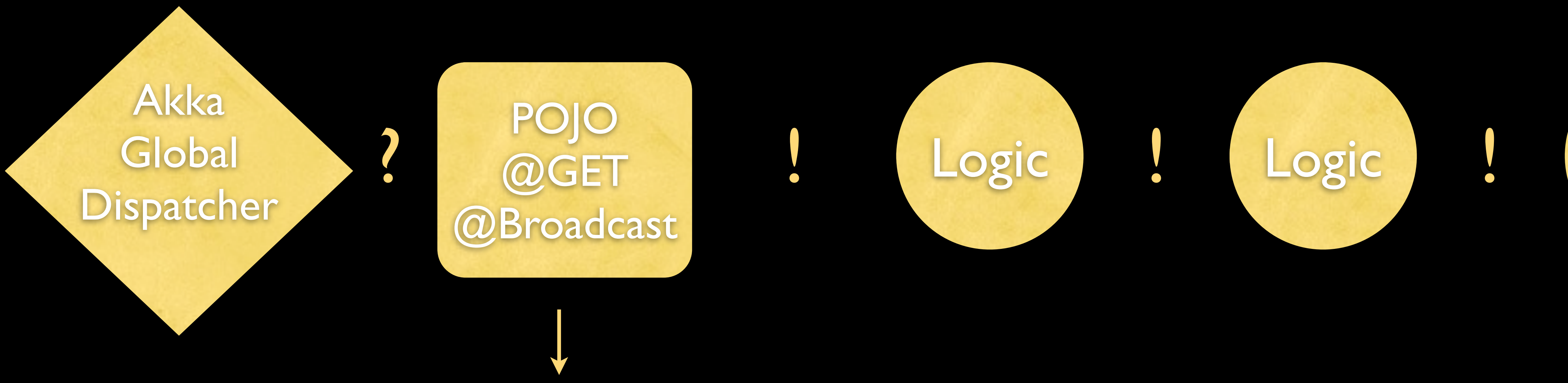
Mist


Akka Global Dispatcher
?
POJO @GET @Broadcast
!
Logic
!
Logic
!
Logic

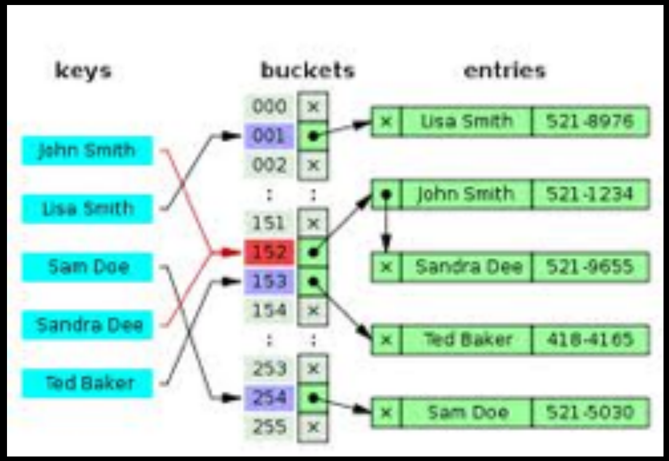
Mist



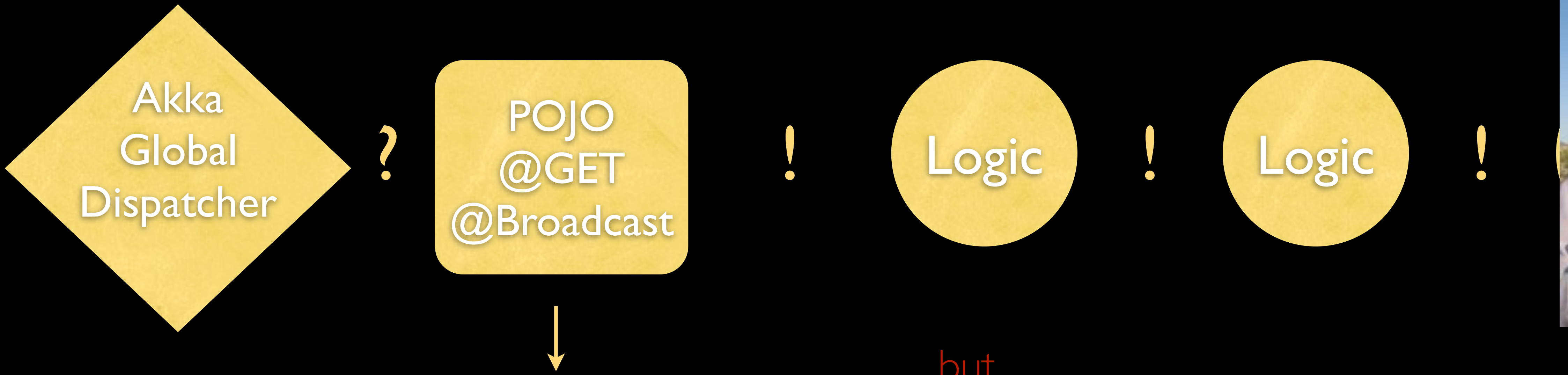
Mist



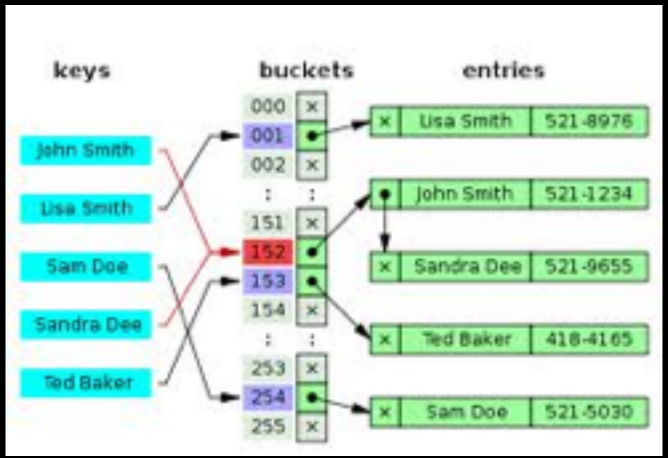
too much shared state



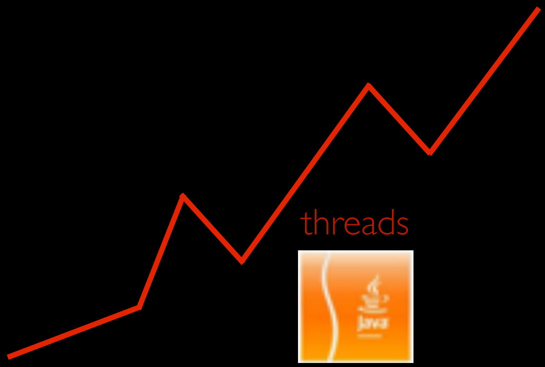
Mist



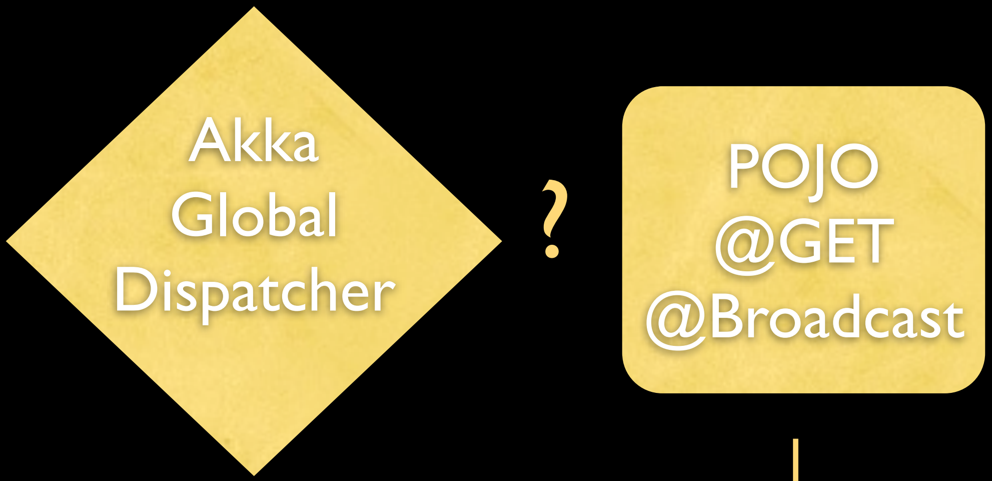
too much shared state



but...

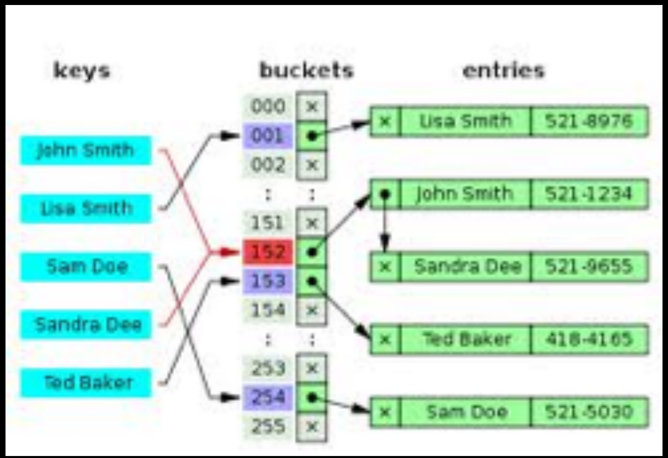


Mist



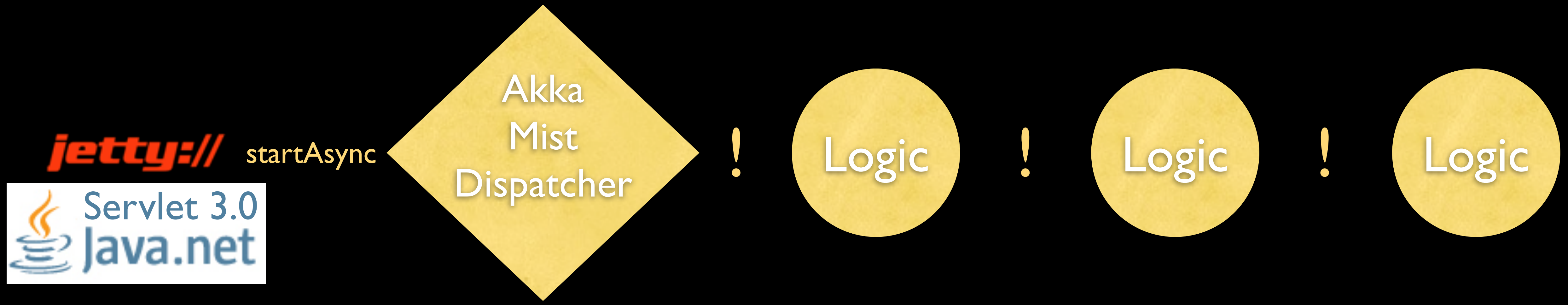
but...

too much shared state

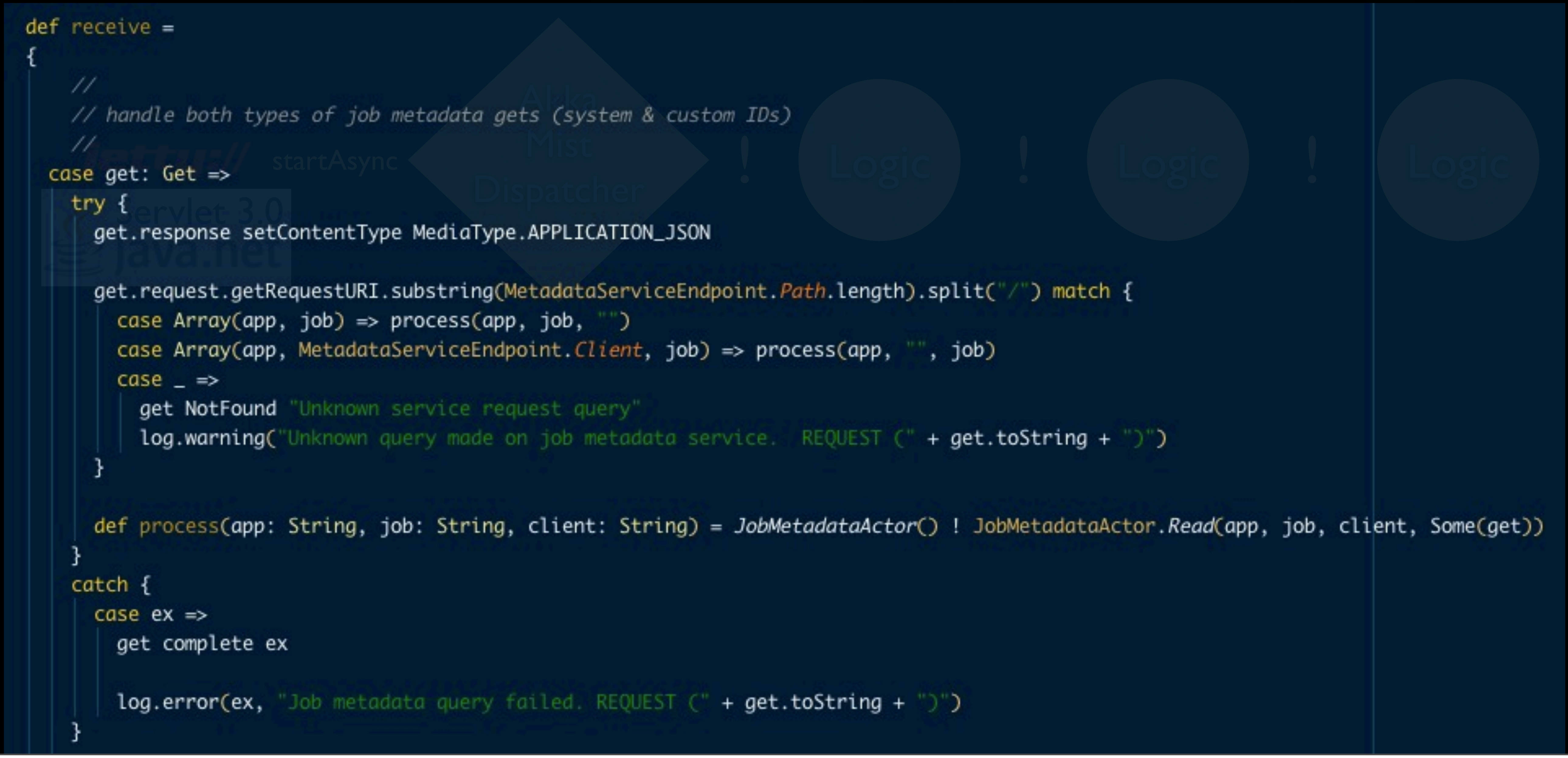


Mist

Mist

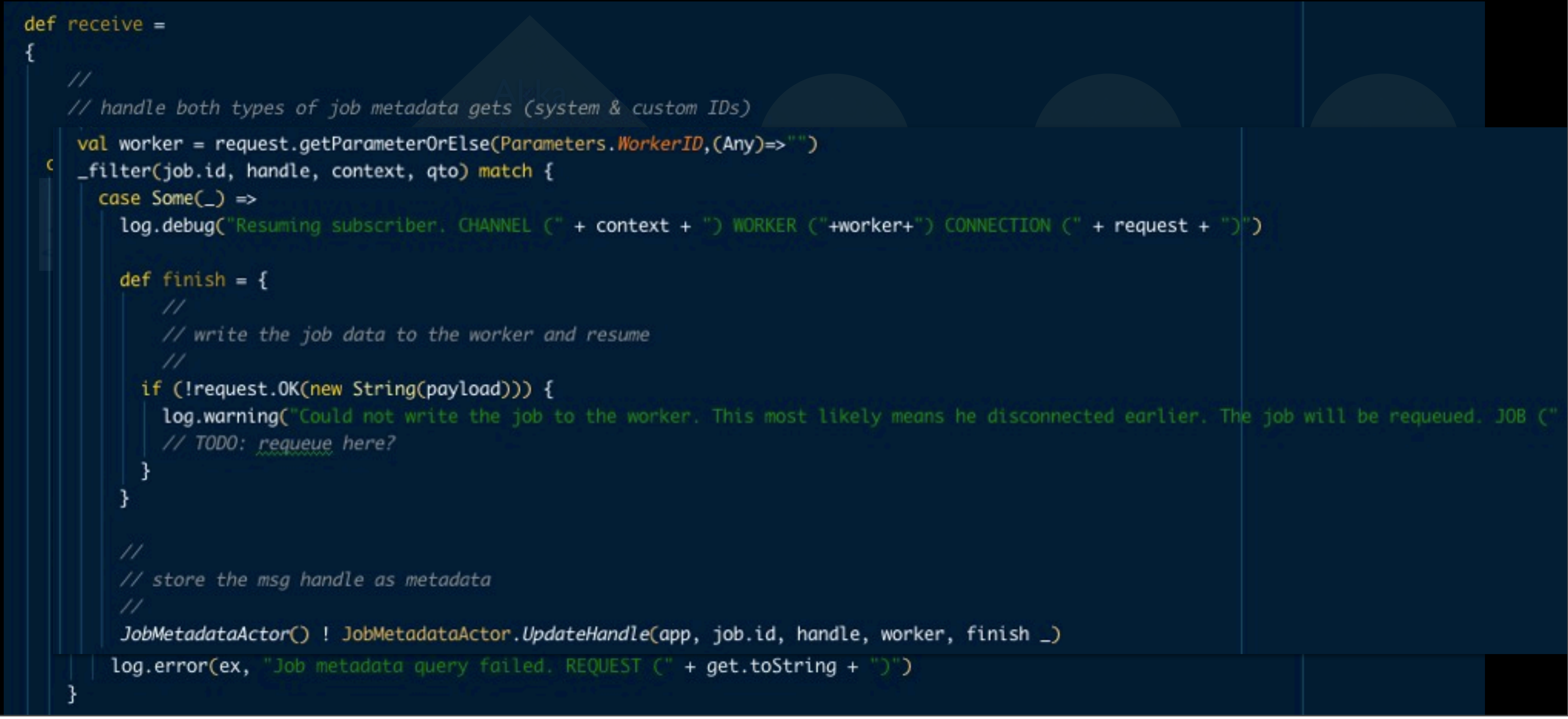

jetty://
Servlet 3.0
Java.net
startAsync
Akka
Mist
Dispatcher
!
Logic
!
Logic
!
Logic

Mist

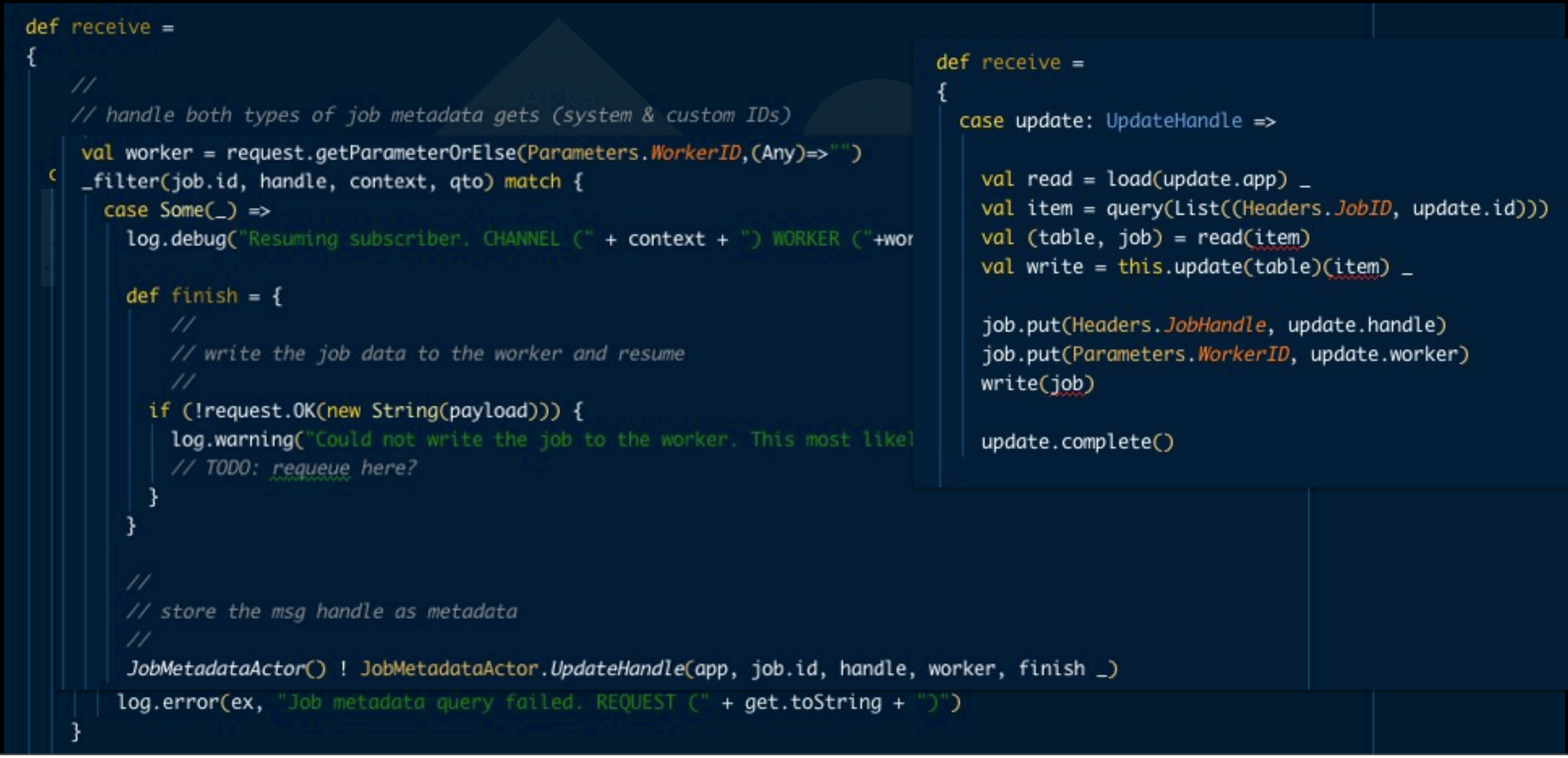


```
def receive =
{
    //
    // handle both types of job metadata gets (system & custom IDs)
    //
  case get: Get =>
    try {
      get.response setContentType MediaType.APPLICATION_JSON

      get.request.getRequestURI.substring(MetadataServiceEndpoint.Path.length).split("/") match {
        case Array(app, job) => process(app, job, "")
        case Array(app, MetadataServiceEndpoint.Client, job) => process(app, "", job)
        case _ =>
          get NotFound "Unknown service request query"
          log.warning("Unknown query made on job metadata service.  REQUEST (" + get.toString + ")")
      }

      def process(app: String, job: String, client: String) = JobMetadataActor() ! JobMetadataActor.Read(app, job, client, Some(get))
    }
    catch {
      case ex =>
        get complete ex

        log.error(ex, "Job metadata query failed. REQUEST (" + get.toString + ")")
    }
```

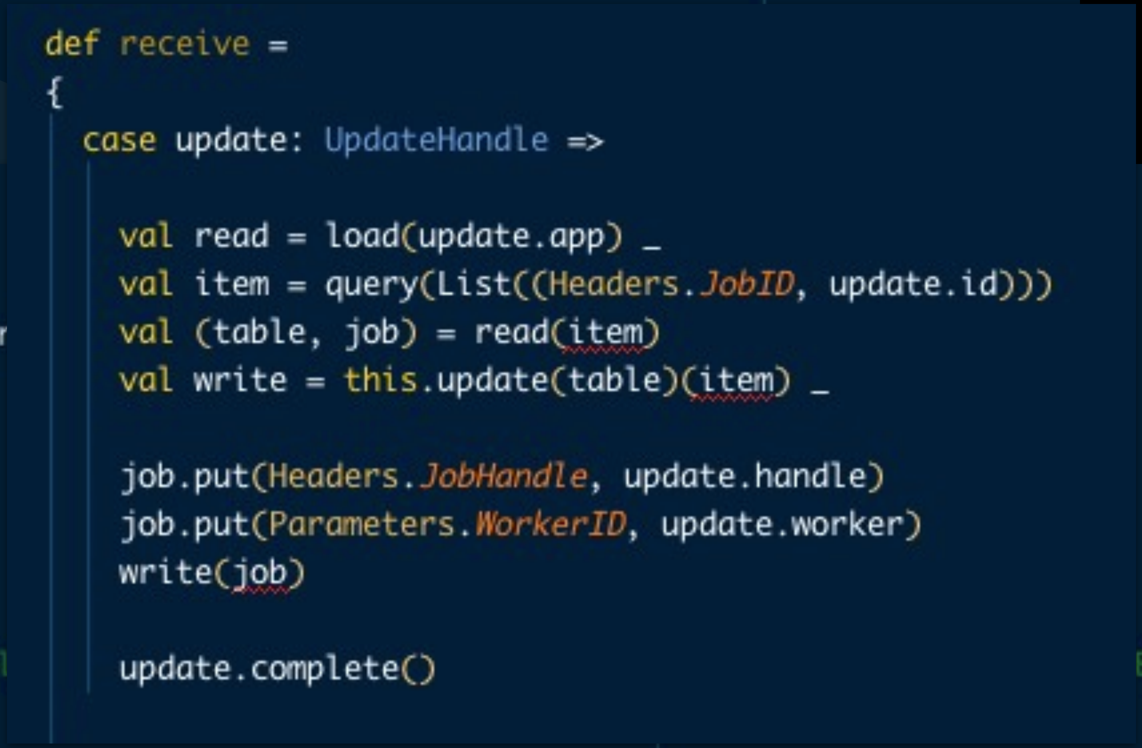
Mist



```
def receive =
{
    //
    // handle both types of job metadata gets (system & custom IDs)
    val worker = request.getParameterOrElse(Parameters.WorkerID,(Any)=>"")
    _filter(job.id, handle, context, qto) match {
      case Some(_) =>
        log.debug("Resuming subscriber. CHANNEL (" + context + ") WORKER ("+worker+") CONNECTION (" + request + ")")

        def finish = {
          //
          // write the job data to the worker and resume
          //
          if (!request.OK(new String(payload))) {
            log.warning("Could not write the job to the worker. This most likely means he disconnected earlier. The job will be requeued. JOB ("
            // TODO: requeue here?
          }
        }

        //
        // store the msg handle as metadata
        //
        JobMetadataActor() ! JobMetadataActor.UpdateHandle(app, job.id, handle, worker, finish _)
      log.error(ex, "Job metadata query failed. REQUEST (" + get.toString + ")")
}
```

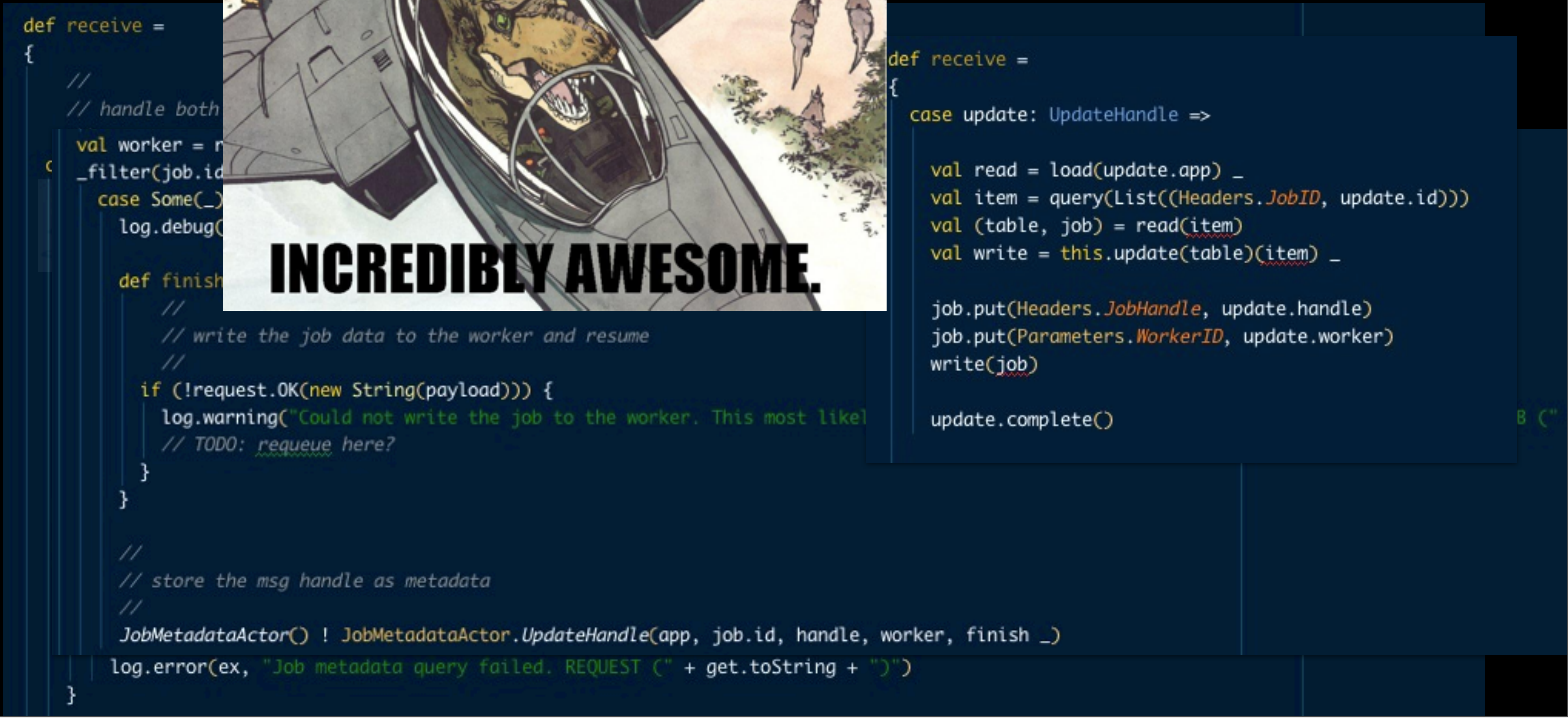


```
def receive =
{
    //
    // handle both types of job metadata gets (system & custom IDs)
     val worker = request.getParameterOrElse(Parameters.WorkerID,(Any)=>"")
     _filter(job.id, handle, context, qto) match {
       case Some(_) =>
         log.debug("Resuming subscriber. CHANNEL (" + context + ") WORKER ("+wor

         def finish = {
             //
             // write the job data to the worker and resume
             //
           if (!request.OK(new String(payload))) {
             log.warning("Could not write the job to the worker. This most likel
             // TODO: requeue here?
           }
         }

         //
         // store the msg handle as metadata
         //
         JobMetadataActor() ! JobMetadataActor.UpdateHandle(app, job.id, handle, worker, finish _)
        log.error(ex, "Job metadata query failed. REQUEST (" + get.toString + ")")
    }
```



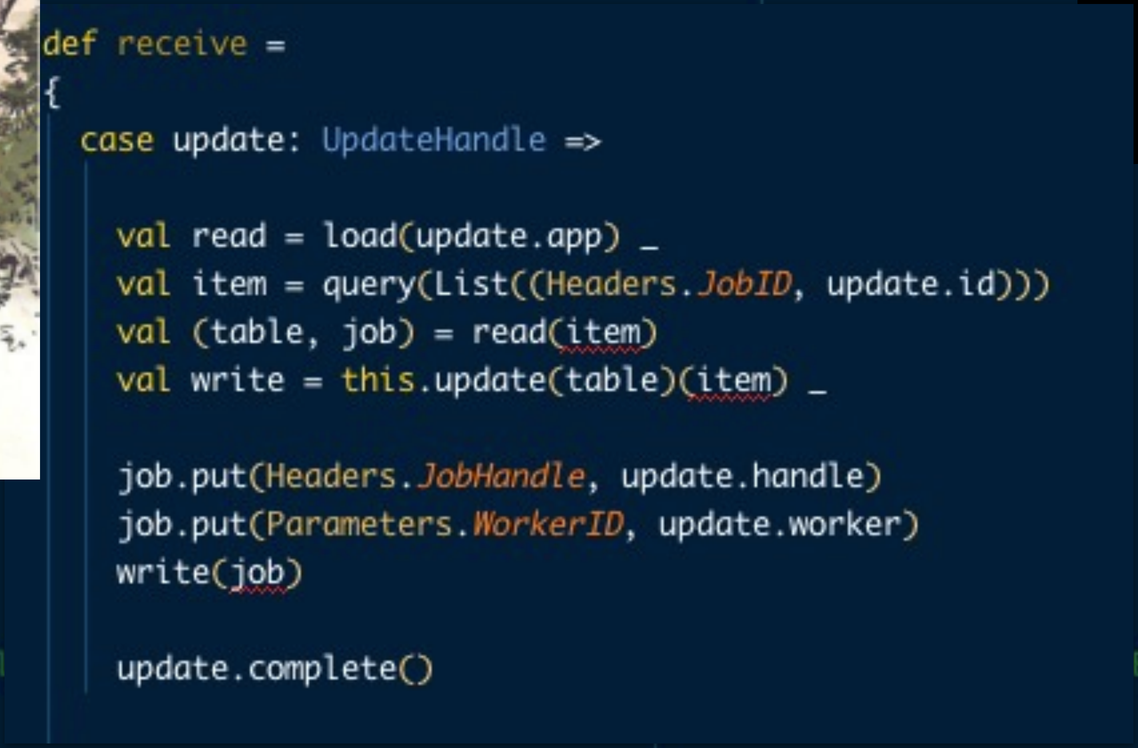
```
def receive =
{
  case update: UpdateHandle =>

    val read = load(update.app) _
    val item = query(List((Headers.JobID, update.id)))
    val (table, job) = read(item)
    val write = this.update(table)(item) _

    job.put(Headers.JobHandle, update.handle)
    job.put(Parameters.WorkerID, update.worker)
    write(job)

    update.complete()
```

Mist



```
def receive =
{
    //
    // handle both
      val worker = r
 (    _filter(job.id
        case Some(_)
          log.debug(

          def finish
            //
            // write the job data to the worker and resume
            //
           if (!request.OK(new String(payload))) {
             log.warning("Could not write the job to the worker. This most likel
             // TODO: requeue here?
           }
          }

          //
          // store the msg handle as metadata
          //
          JobMetadataActor() ! JobMetadataActor.UpdateHandle(app, job.id, handle, worker, finish _)
        log.error(ex, "Job metadata query failed. REQUEST (" + get.toString + ")")
      }
```

```
B ("
```



```
def receive =
{
  case update: UpdateHandle =>

    val read = load(update.app) _
    val item = query(List((Headers.JobID, update.id)))
    val (table, job) = read(item)
    val write = this.update(table)(item) _

    job.put(Headers.JobHandle, update.handle)
    job.put(Parameters.WorkerID, update.worker)
    write(job)

    update.complete()
```

akka

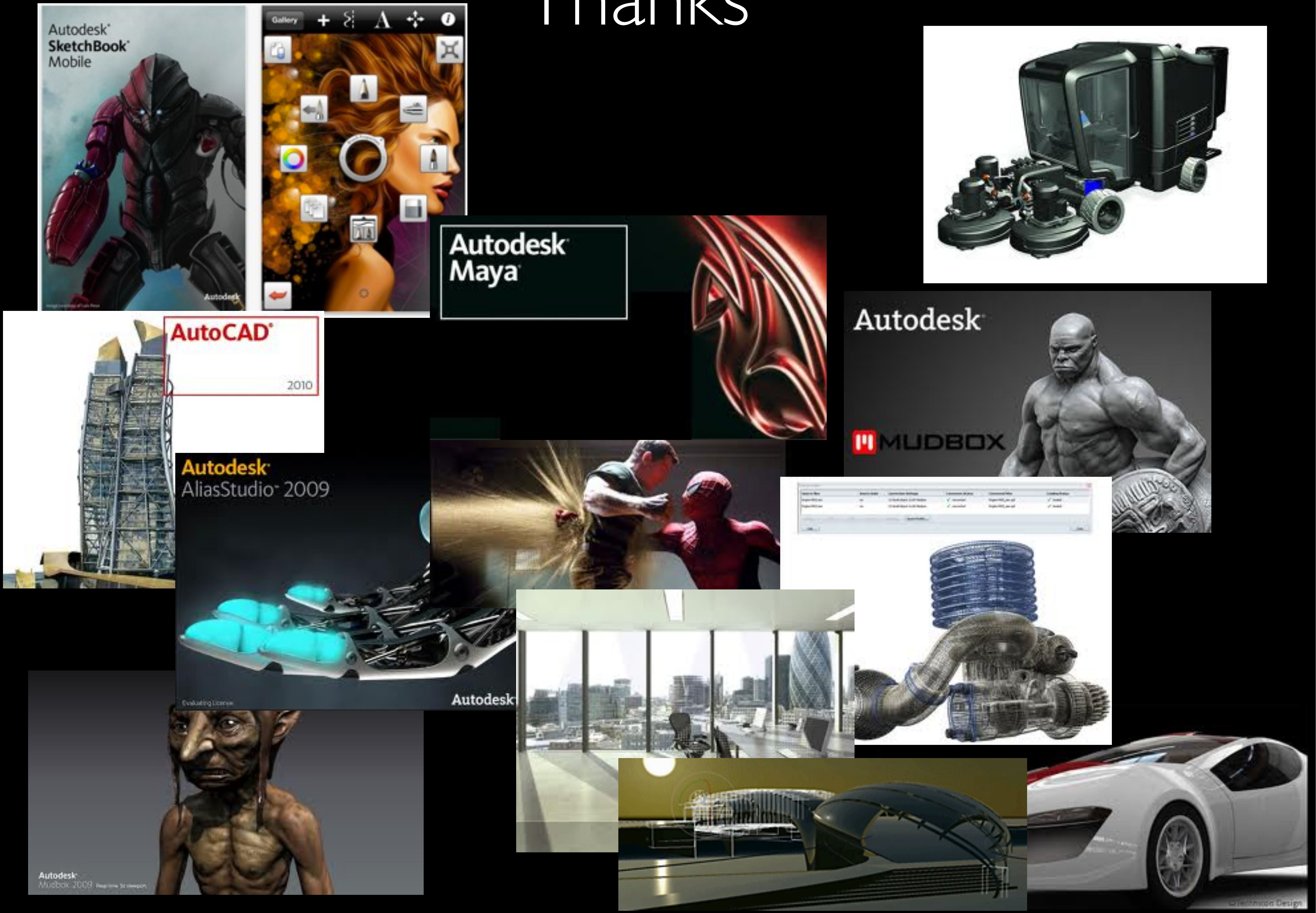
Mist

- Mist developed as Hydrogen component
- Autodesk becomes a contributor to Akka
- Releases Akka-Mist in 1.0
- Experimental extensions to Mist for Jetty Websockets (git branch)
- Contributes ActorPool in 1.1

- Looking forward to more...

Thanks

Thanks


Autodesk SketchBook Mobile
Gallery
Autodesk Maya
AutoCAD 2010
Autodesk
MUDBOX
Autodesk AliasStudio 2009
Autodesk
Autodesk Mudbox 2009

Get it and learn more

http://akka.io

EOF