Scaling out with Akka Actors

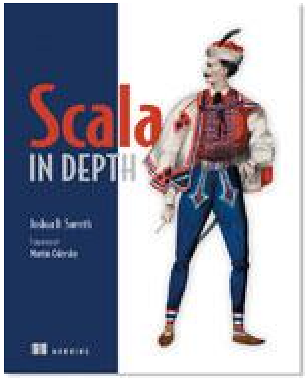
J. Suereth

Agenda

- The problem
- Recap on what we have
- Setting up a Cluster
-
- Advanced Techniques

Who am I?

- Author Scala In Depth, sbt in Action
- Typesafe Employee
- Big Nerd



I IS TEN NINJAS

The new web

- EVENT DRIVEN
- ASYNCHRONOUS
- DATA-DRIVEN
- BIG DATA
- SINGLE PAGE DESIGN
- COMPOSITION OF SERVICES
- DISTRIBUTED
- REACTIVE

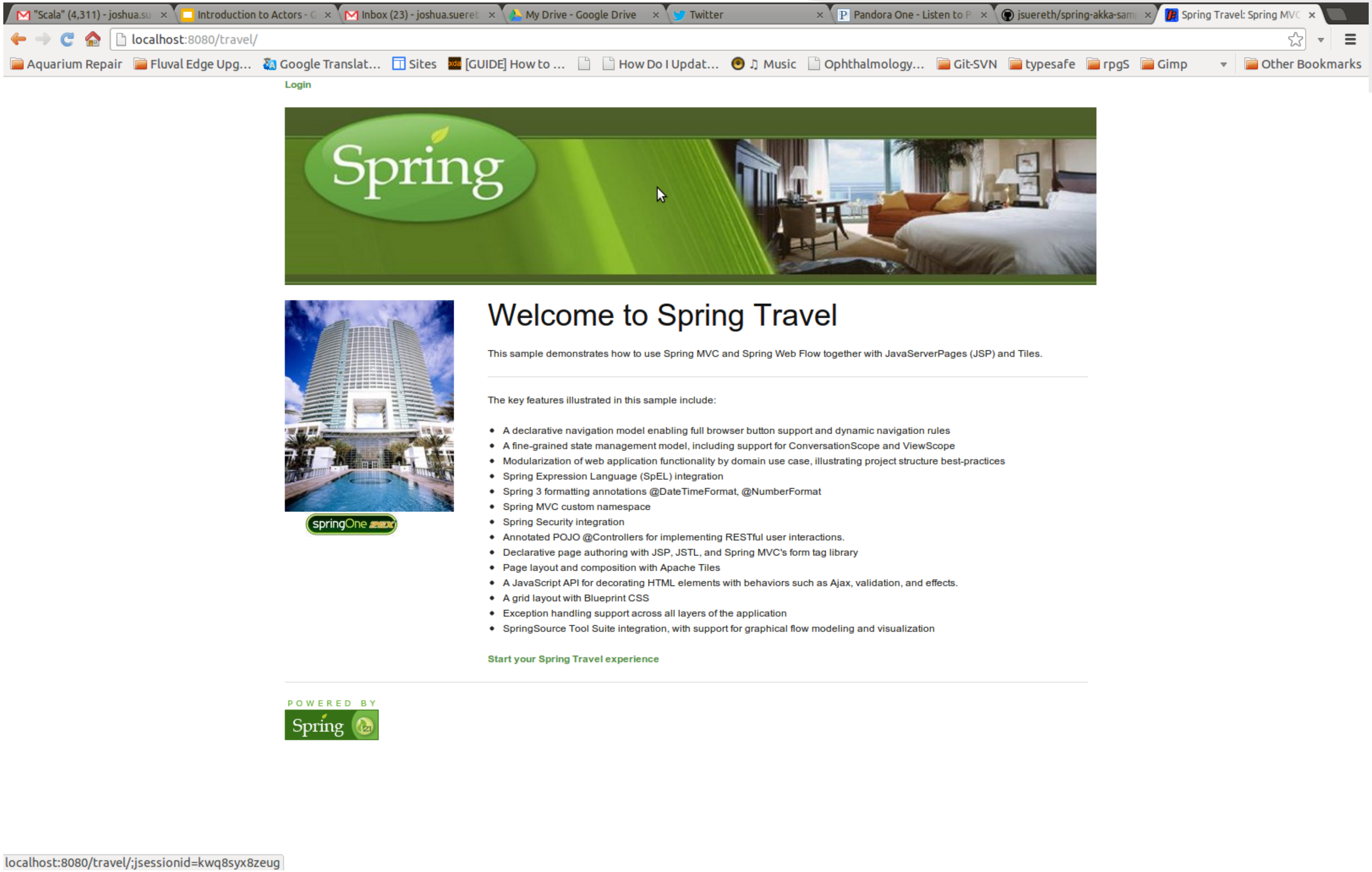
The new web

- EVENT DRIVEN
- ASYNCHRONOUS
- DATA-DRIVEN
- BIG DATA
- SINGLE PAGE DESIGN
- COMPOSITION OF SERVICES
- DISTRIBUTED
- REACTIVE

The problem

I can't scale my website

The Hotel Search Business



Welcome to Spring Travel

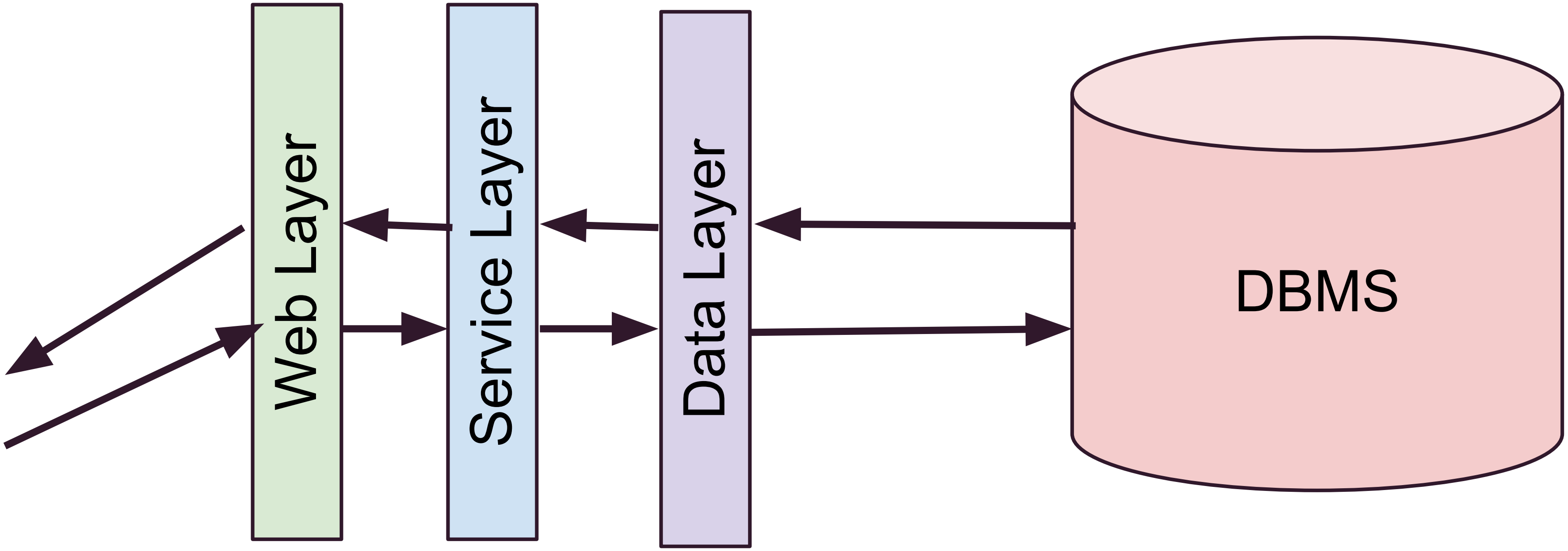
This sample demonstrates how to use Spring MVC and Spring Web Flow together with JavaServerPages (JSP) and Tiles.

The key features illustrated in this sample include:

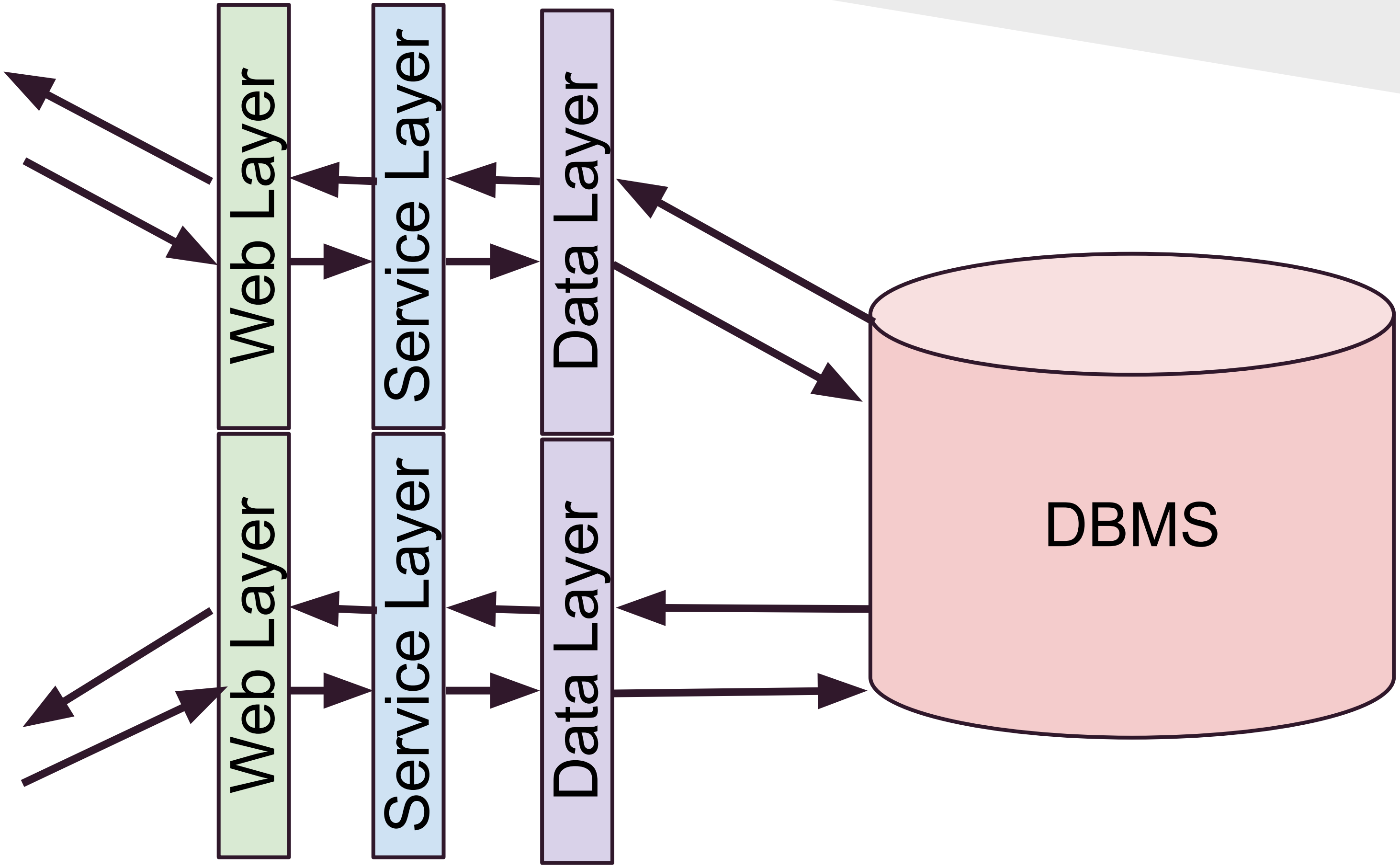
- A declarative navigation model enabling full browser button support and dynamic navigation rules
- A fine-grained state management model, including support for ConversationScope and ViewScope
- Modularization of web application functionality by domain use case, illustrating project structure best-practices
- Spring Expression Language (SpEL) integration
- Spring 3 formatting annotations @DateTimeFormat, @NumberFormat
- Spring MVC custom namespace
- Spring Security integration
- Annotated POJO @Controllers for implementing RESTful user interactions.
- Declarative page authoring with JSP, JSTL, and Spring MVC's form tag library
- Page layout and composition with Apache Tiles
- A JavaScript API for decorating HTML elements with behaviors such as Ajax, validation, and effects.
- A grid layout with Blueprint CSS
- Exception handling support across all layers of the application
- SpringSource Tool Suite integration, with support for graphical flow modeling and visualization

Start your Spring Travel experience

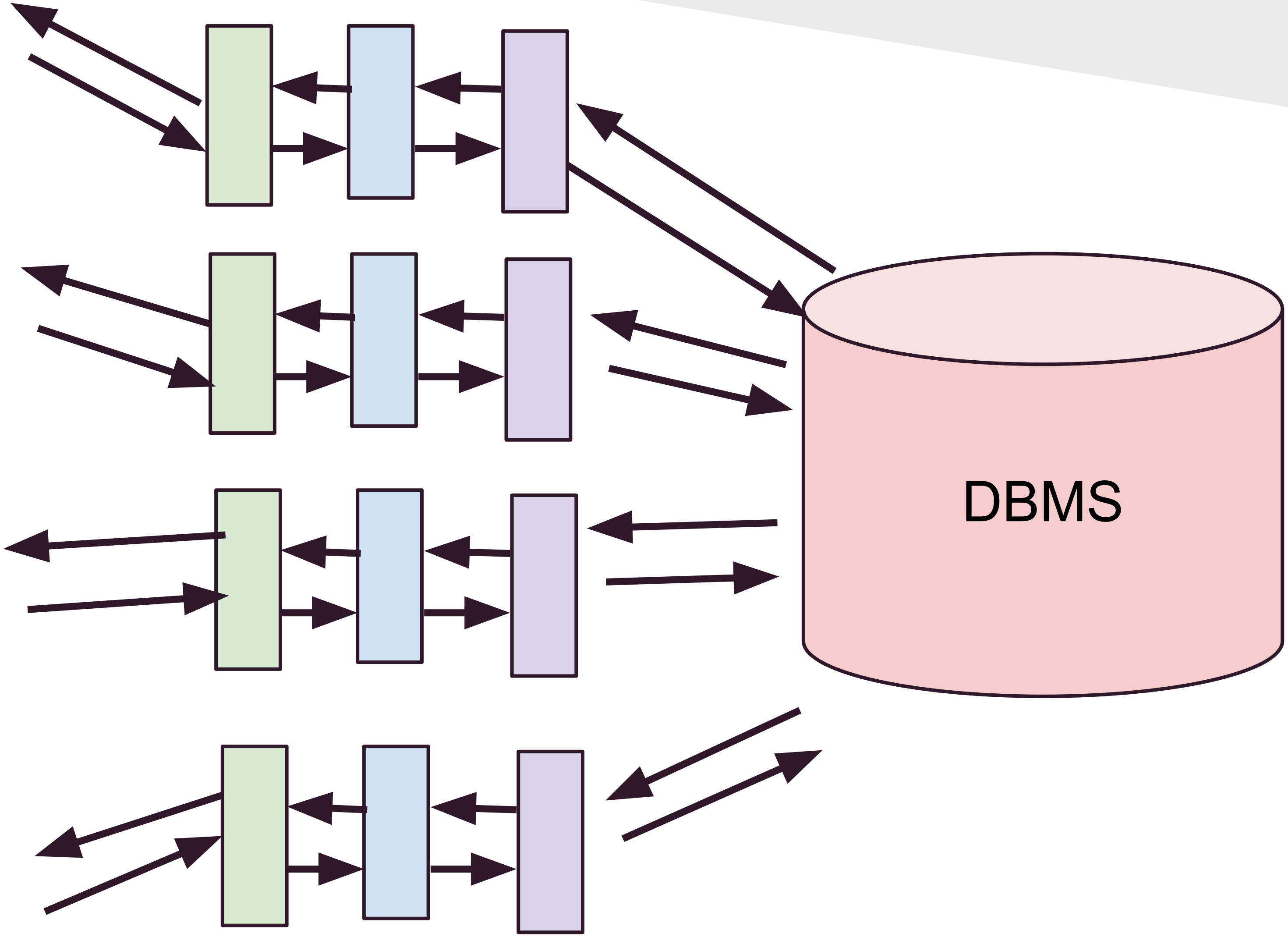
Architecture



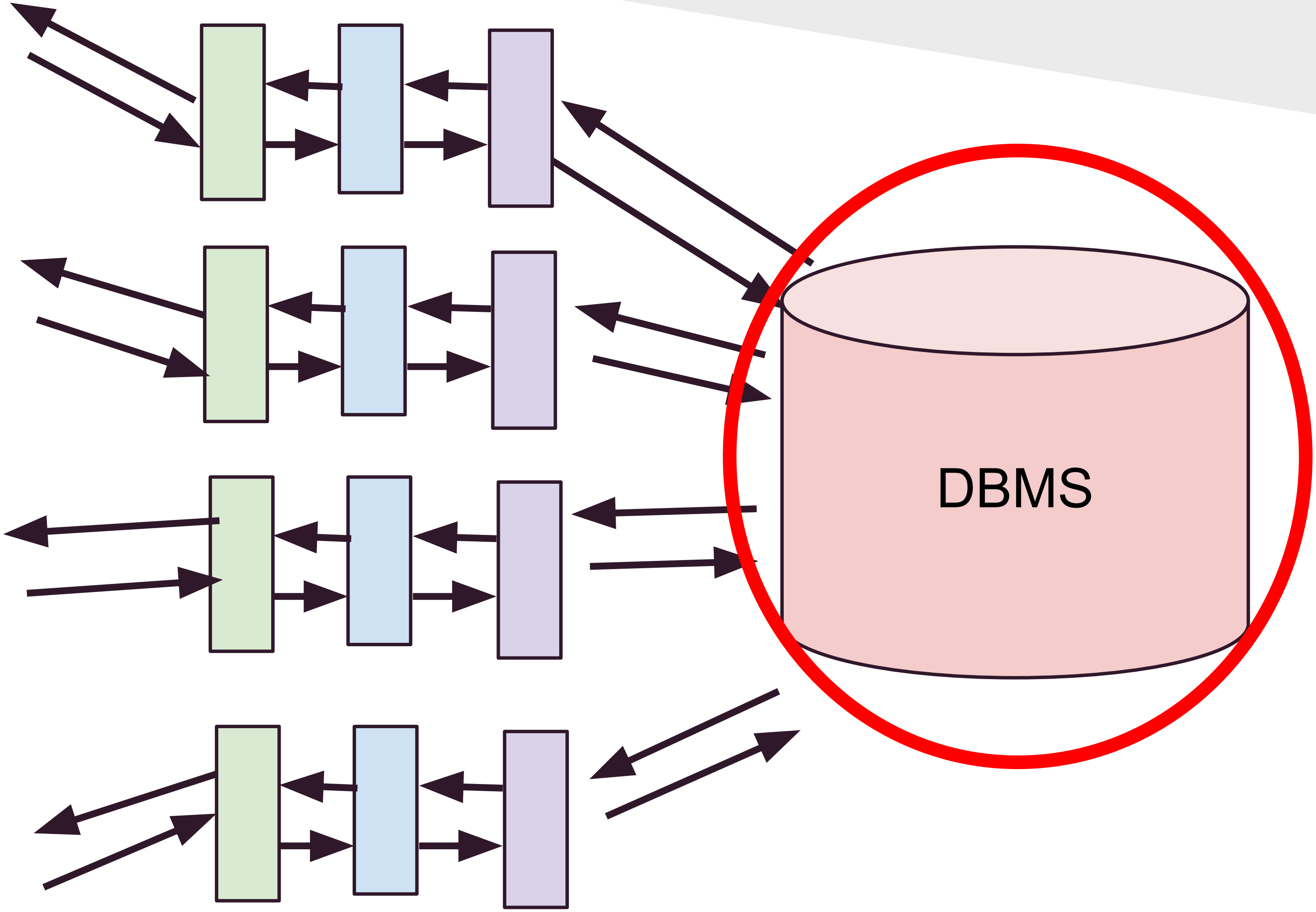
Architecture



Architecture



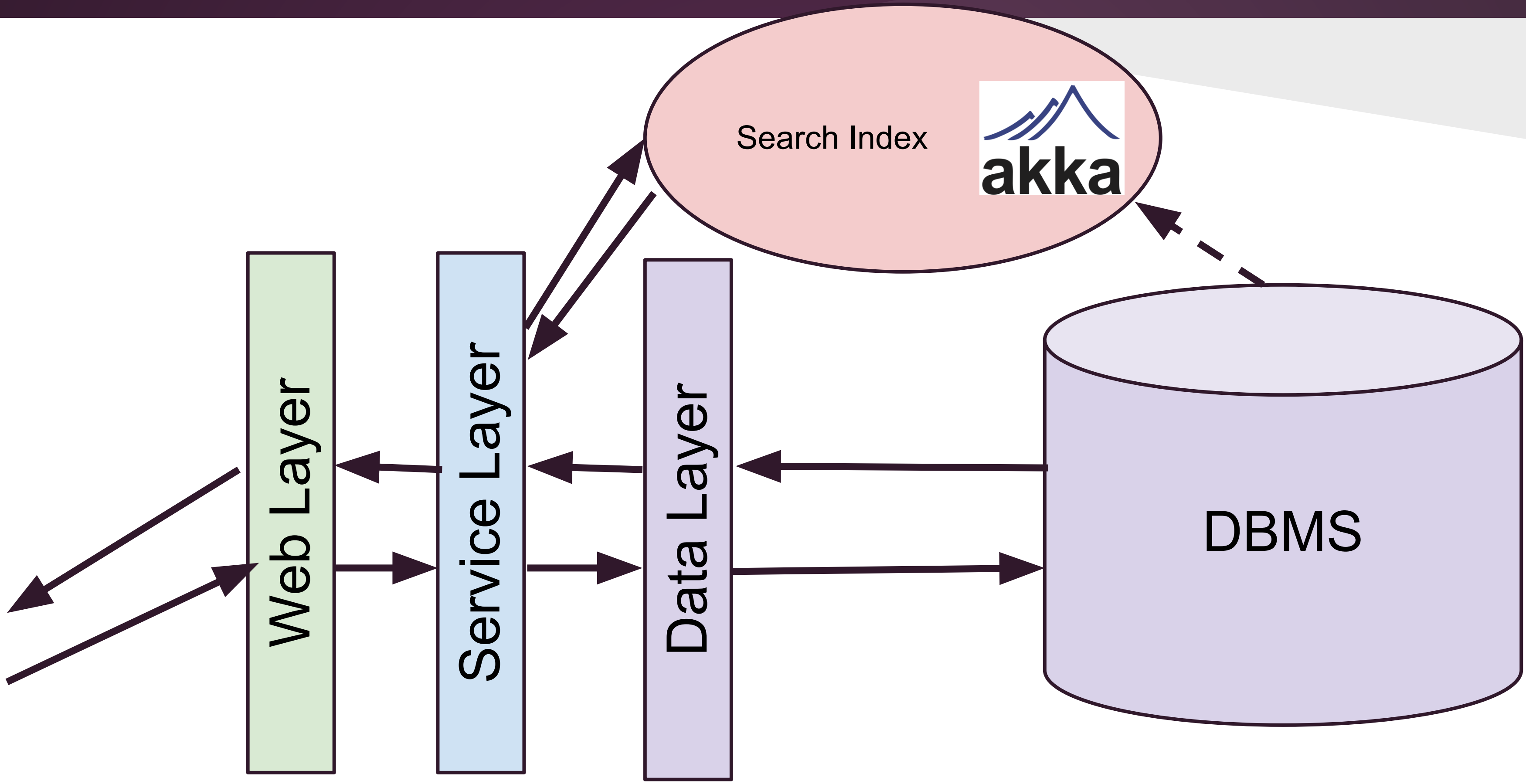
Architecture



So…
we built a Search system that

- Finds hotels
- Dynamically grows the search index
- Caches previous query results for some time
- Detects system overload and returns a cute animal drawing

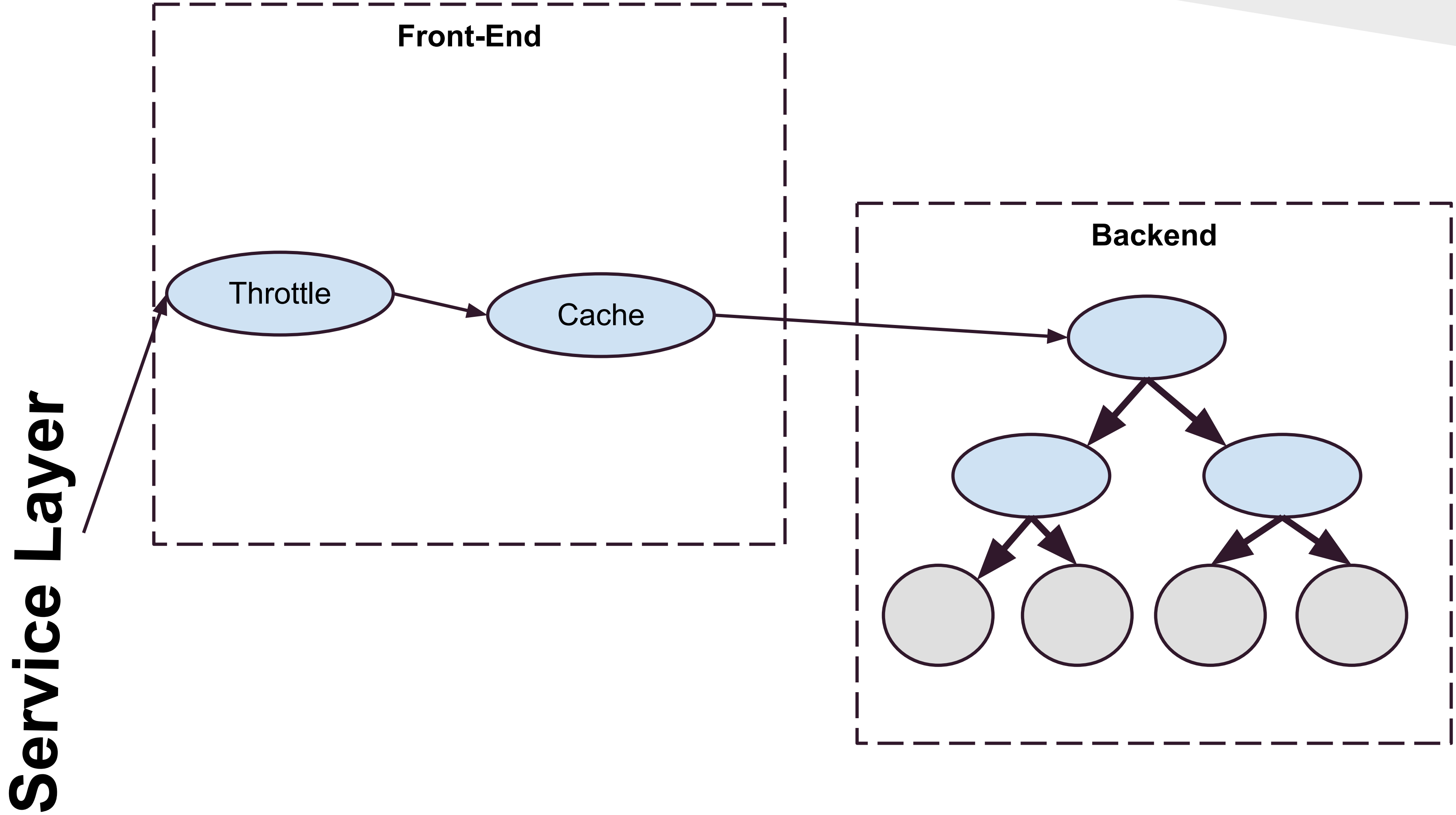
Our Current Architecture



Let's dig into

the Search Index

Current Actor Layout



Scatter Gather Search Index

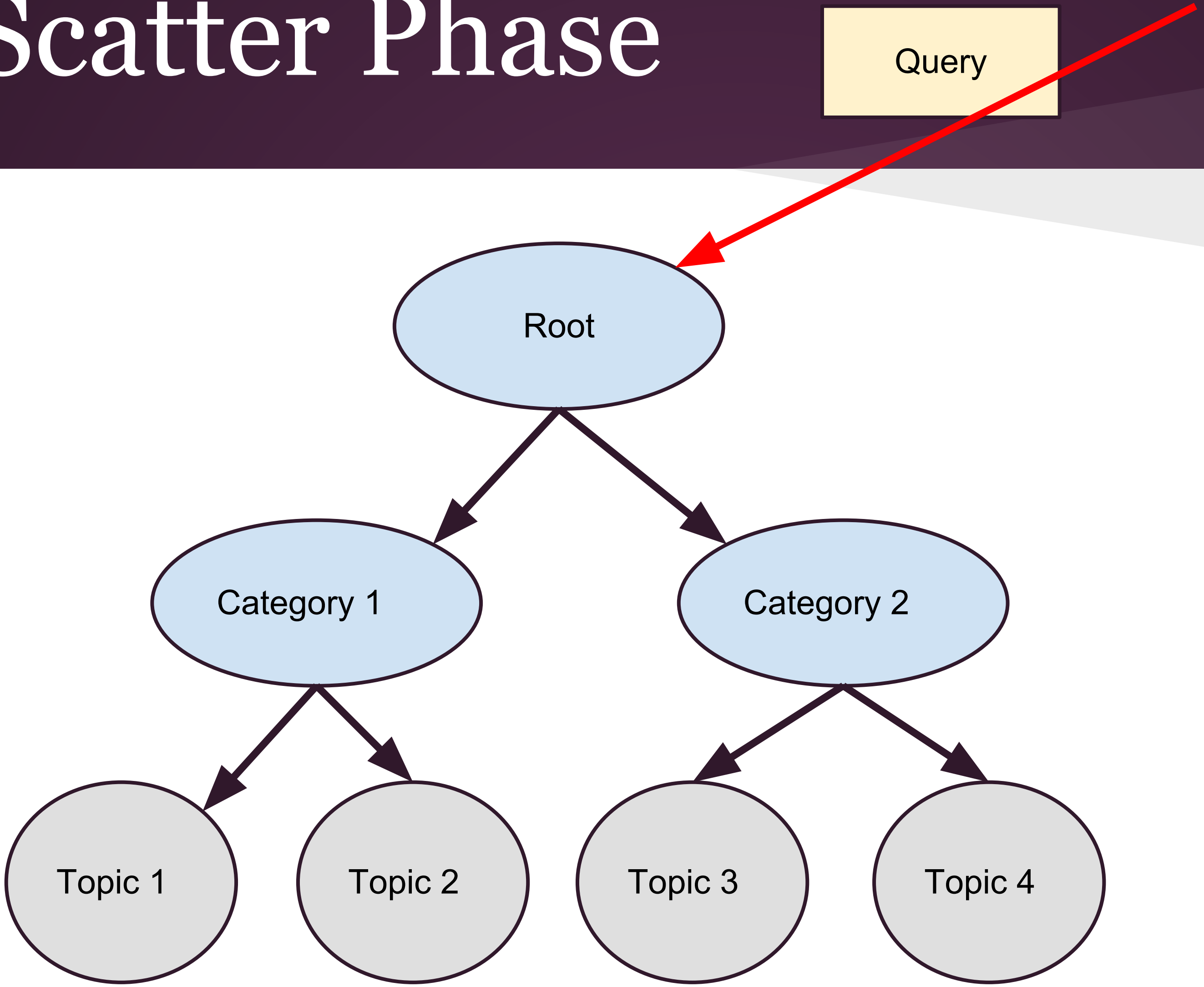
- Split documents into Topics
  - Create a "leaf" actor for each topic.
  - Topic actors have local index
- Categories
  - Group topics into categories
  - Group categories into more categories
  - one root
  - delegate queries to topics
  - aggregate results
- Dynamically Expands
  - Topics can decide to split into categories and sub-topics

Front End

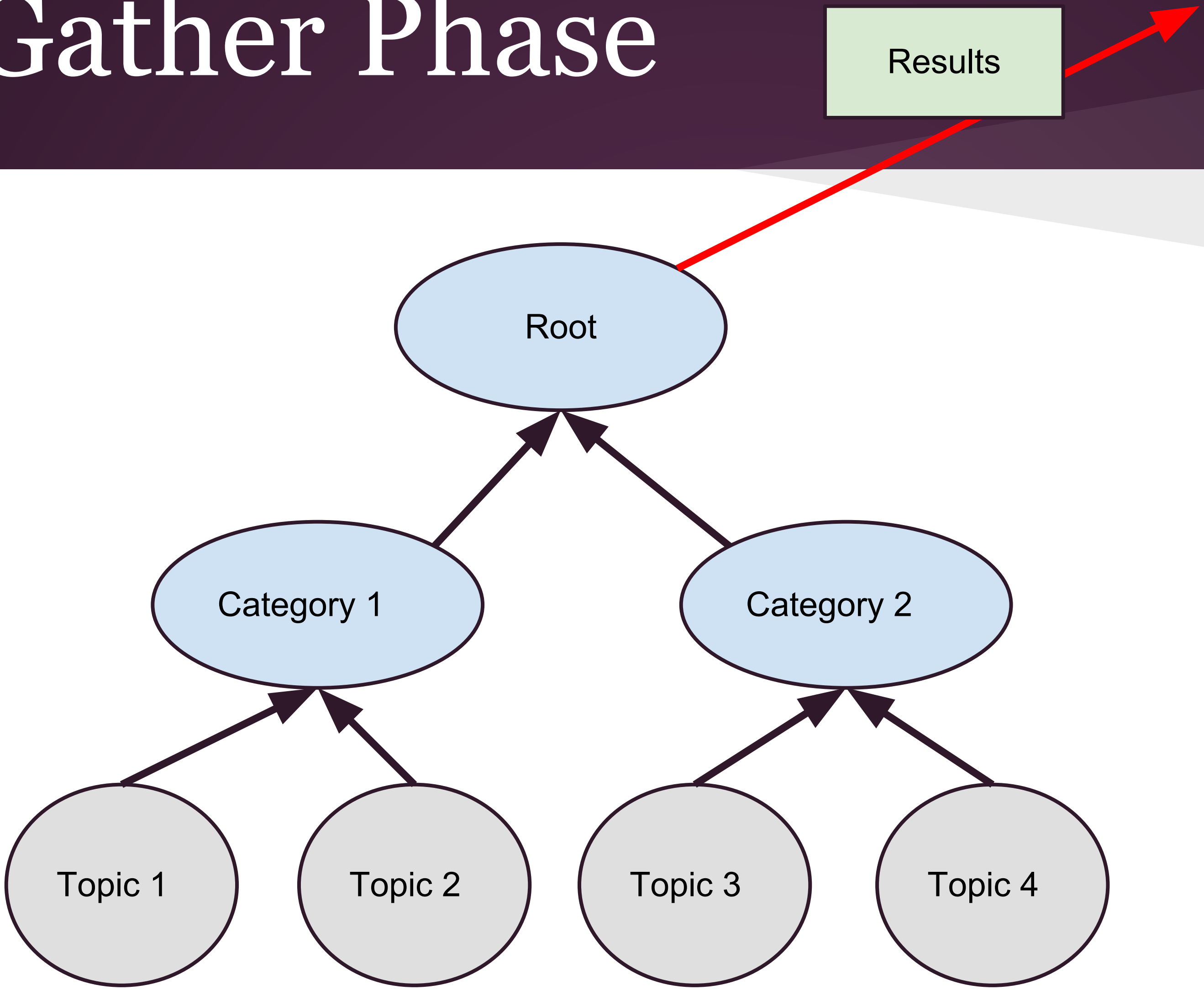
- Query Cache
  - Caches top N query results
  - (Not in sample code) Evicts stale cache
  - Primary source of speedup!
- Throttler
  - Records average query-response-time
  - When in "failure" mode, prevents queries from hitting the system and returns 'failure' response.

Let's remember....

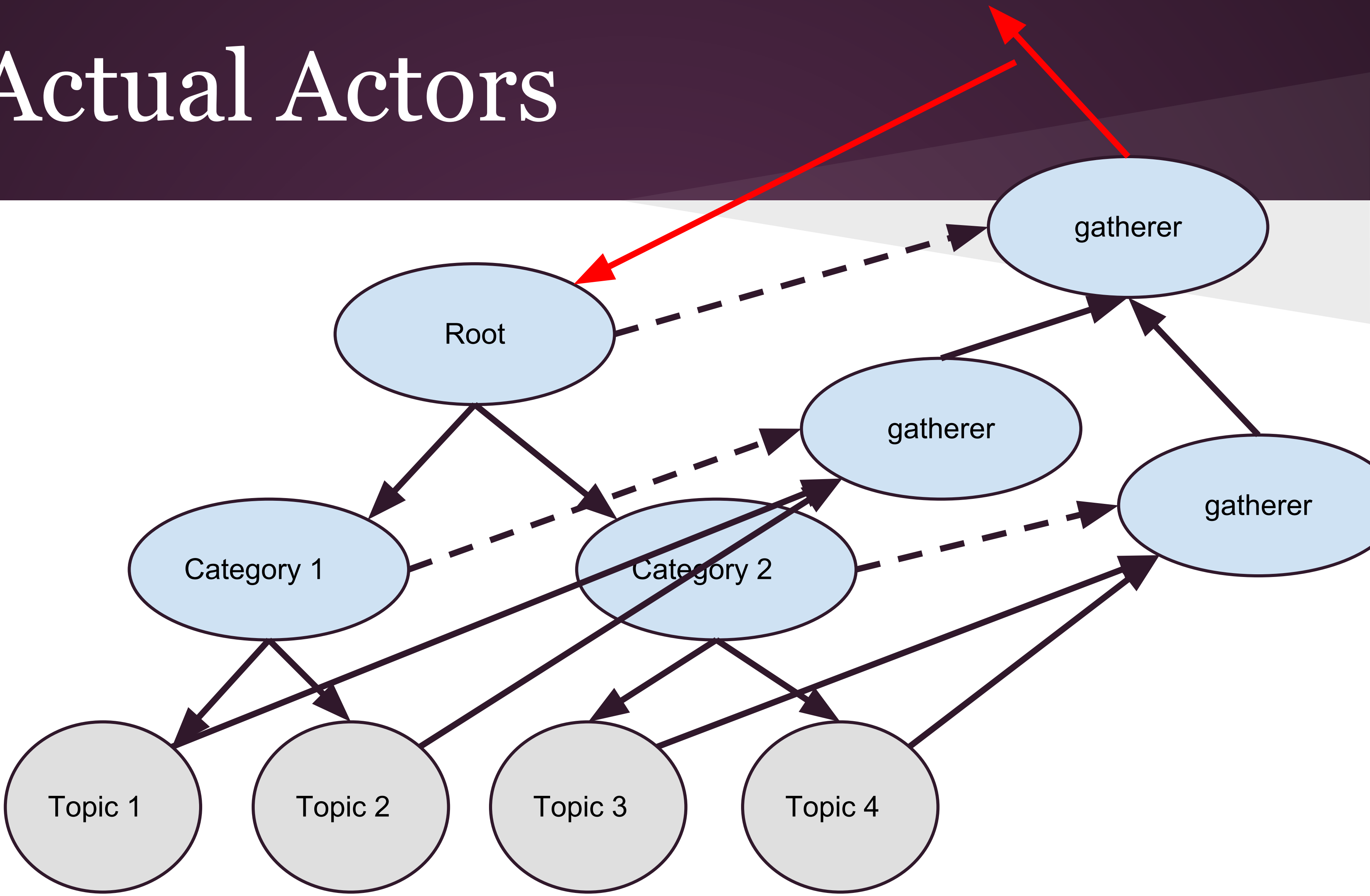
Scatter Phase



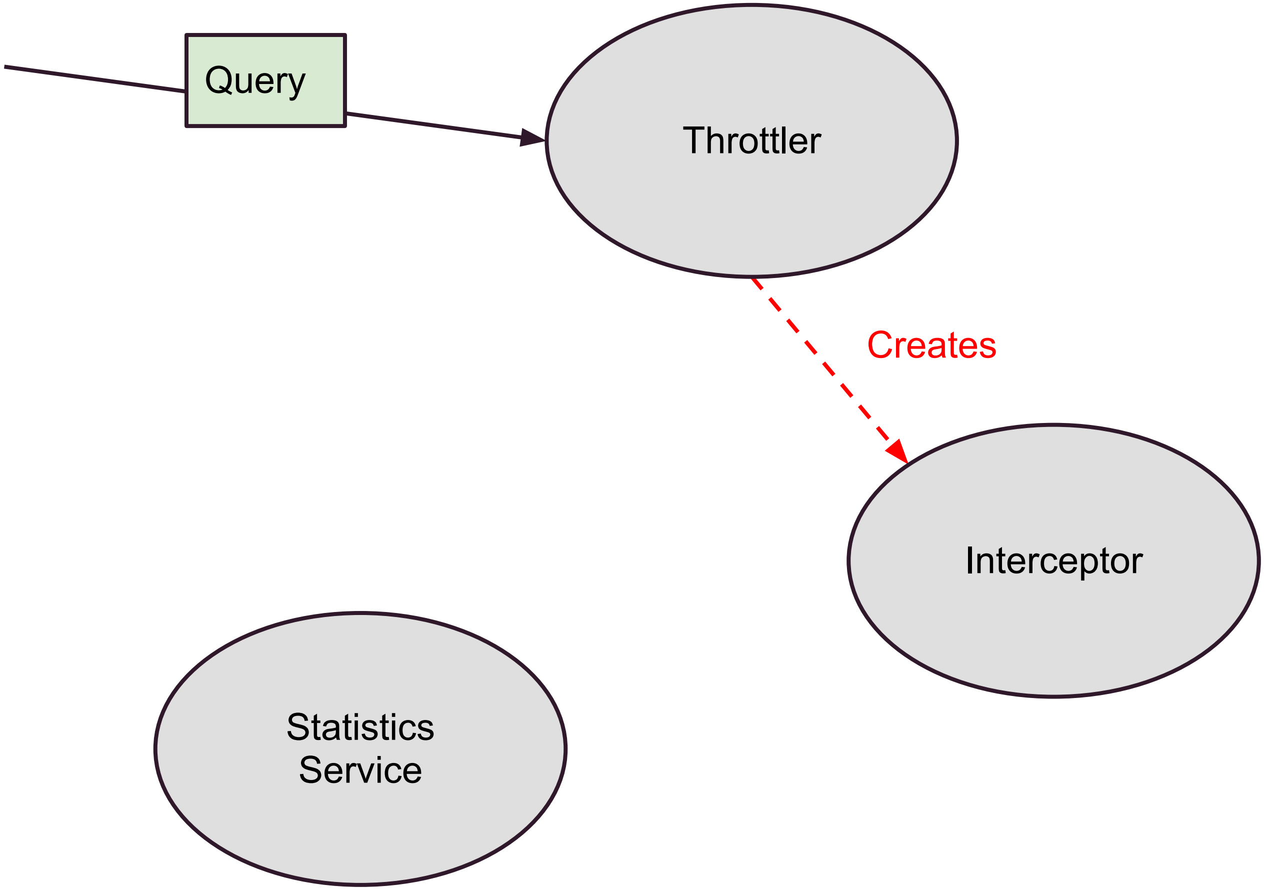
Gather Phase



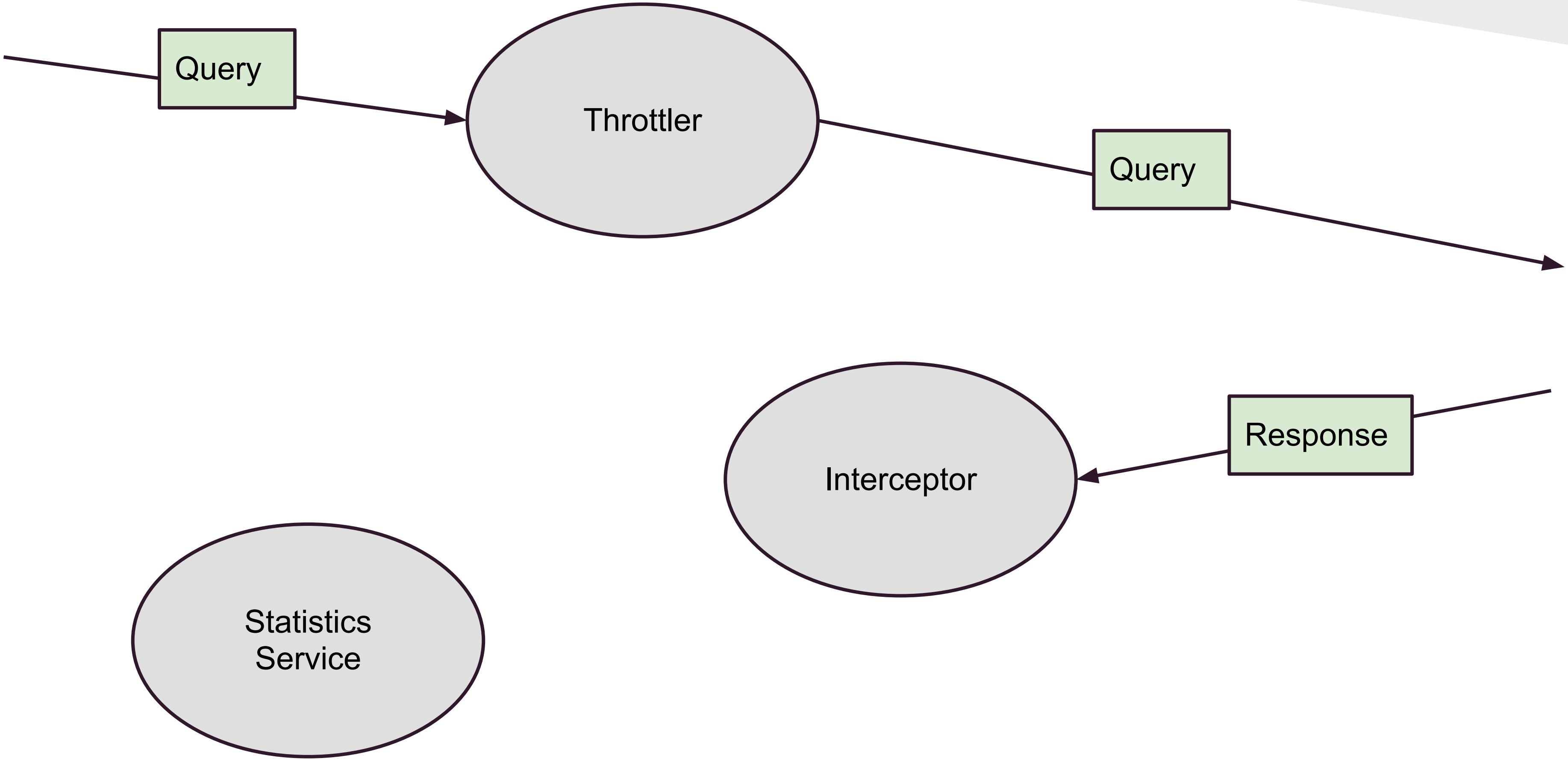
Actual Actors



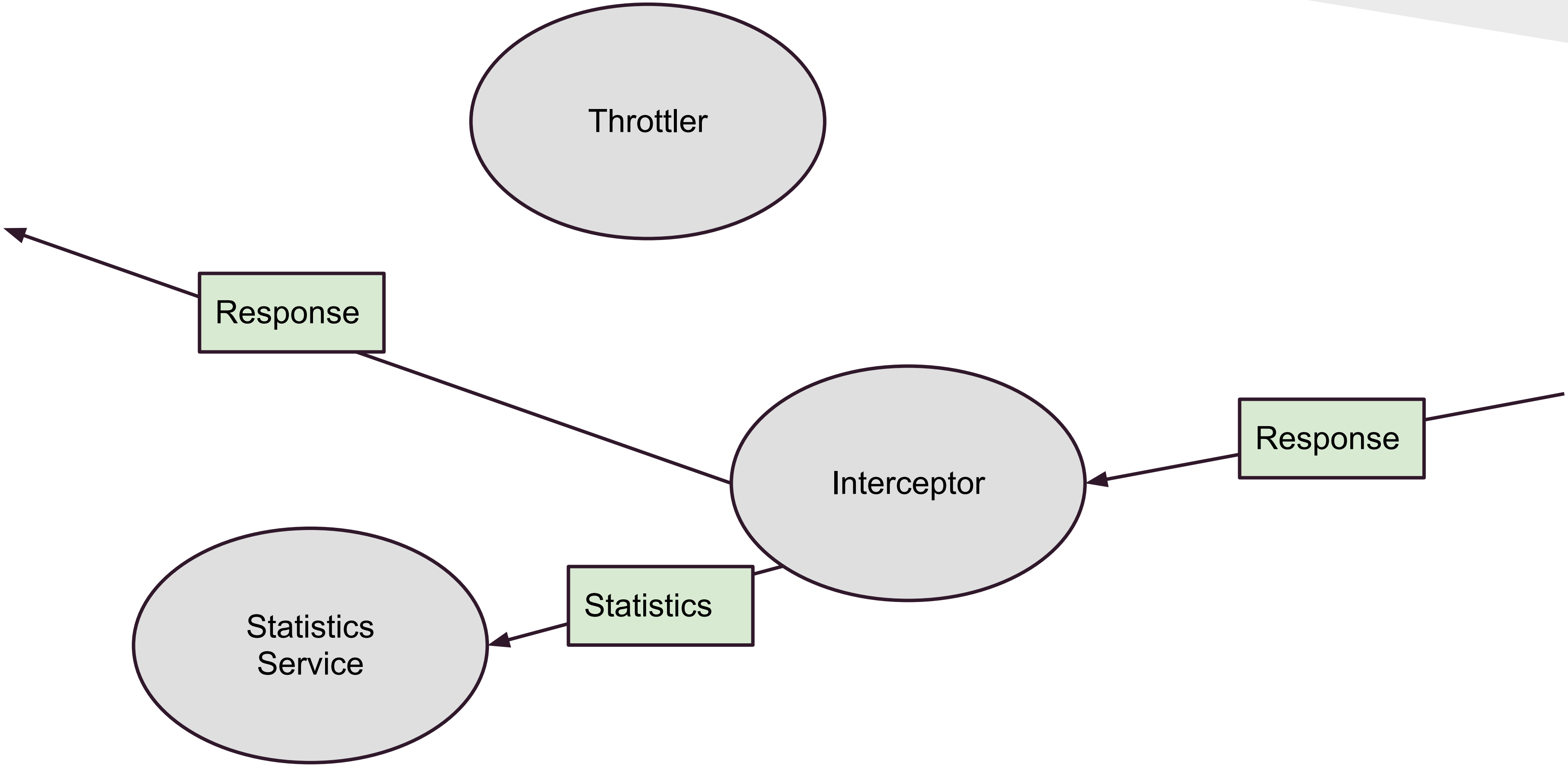
Throttling



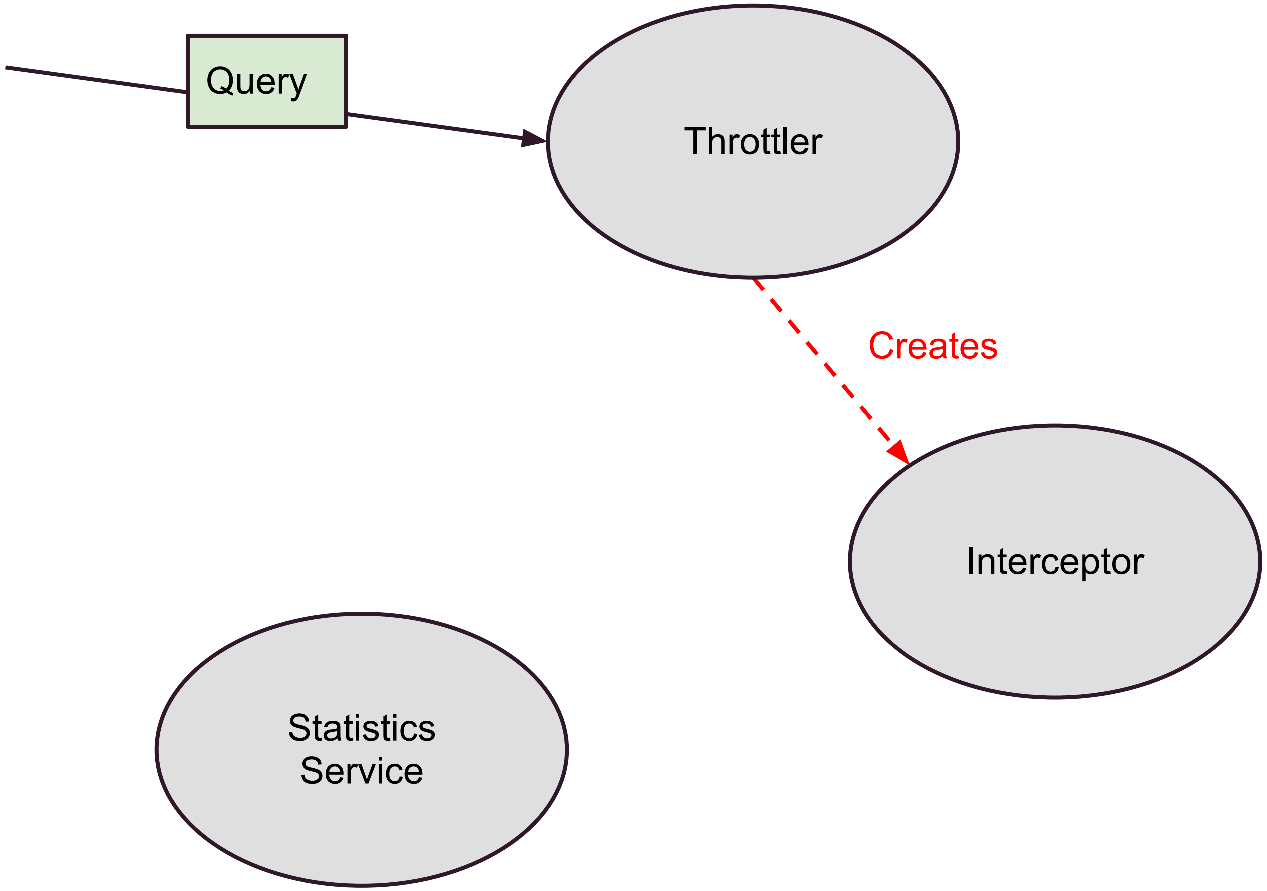
Throttling


Query
Throttler
Query
Interceptor
Response
Statistics Service

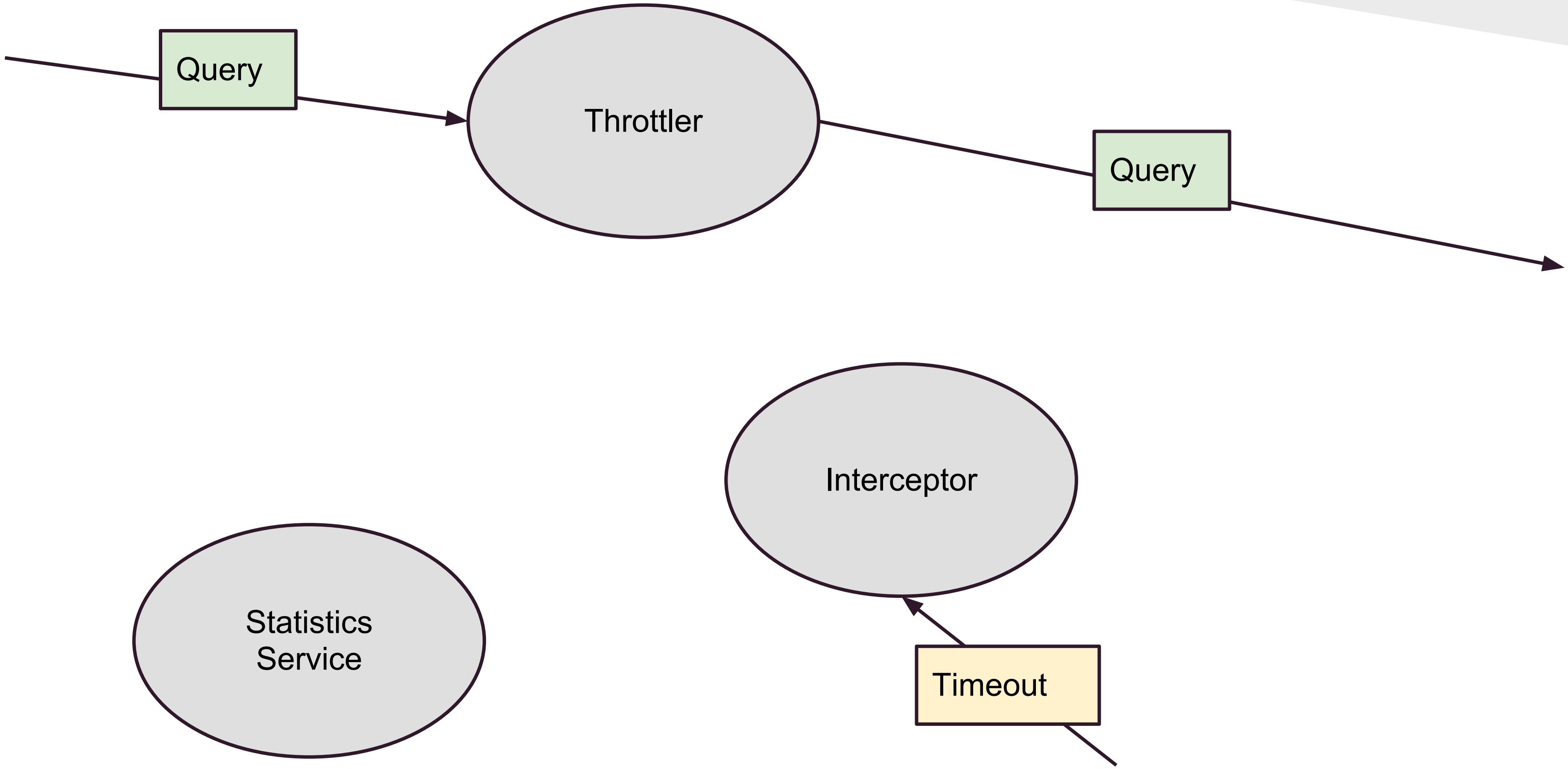
Throttling


Throttler
Response
Interceptor
Response
Statistics
Statistics
Service

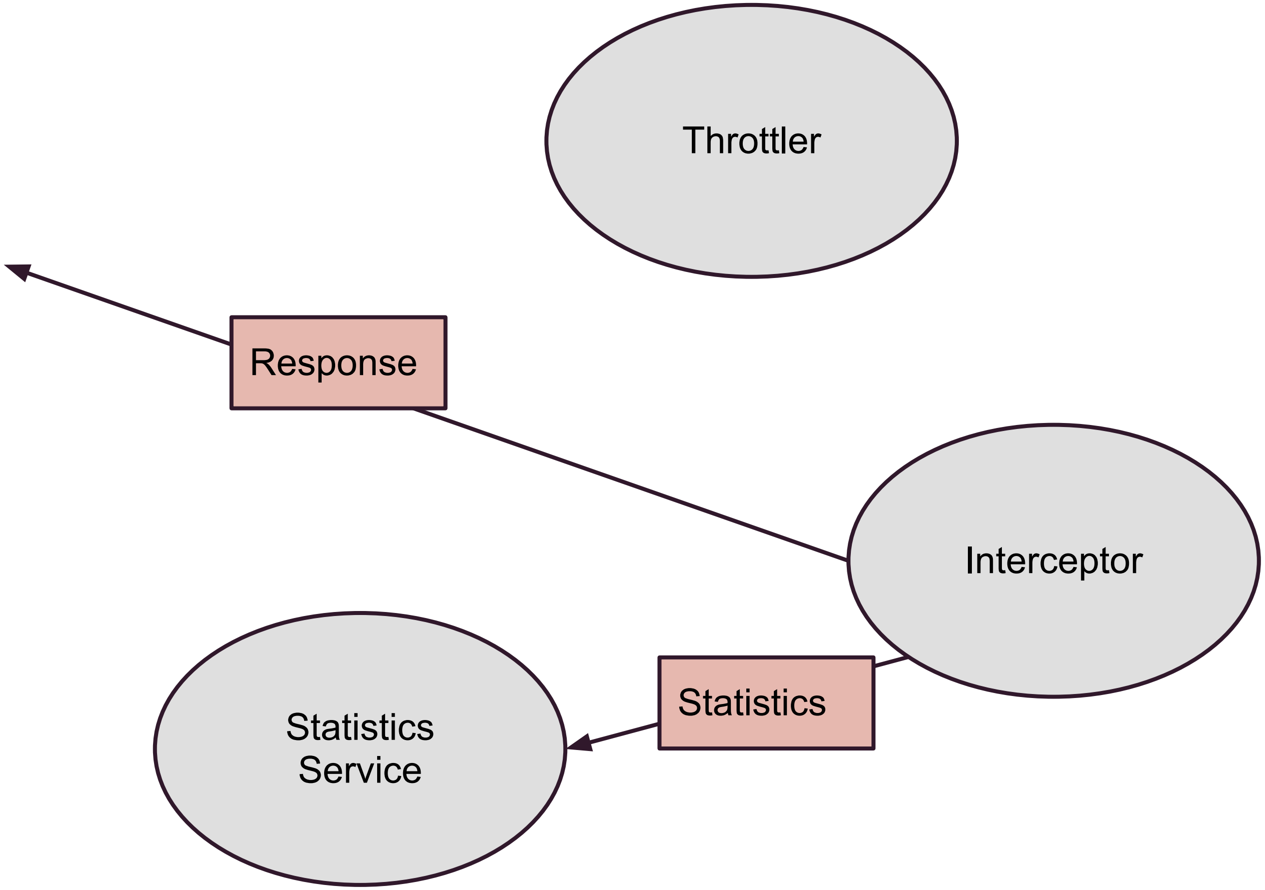
Throttling - Timeouts



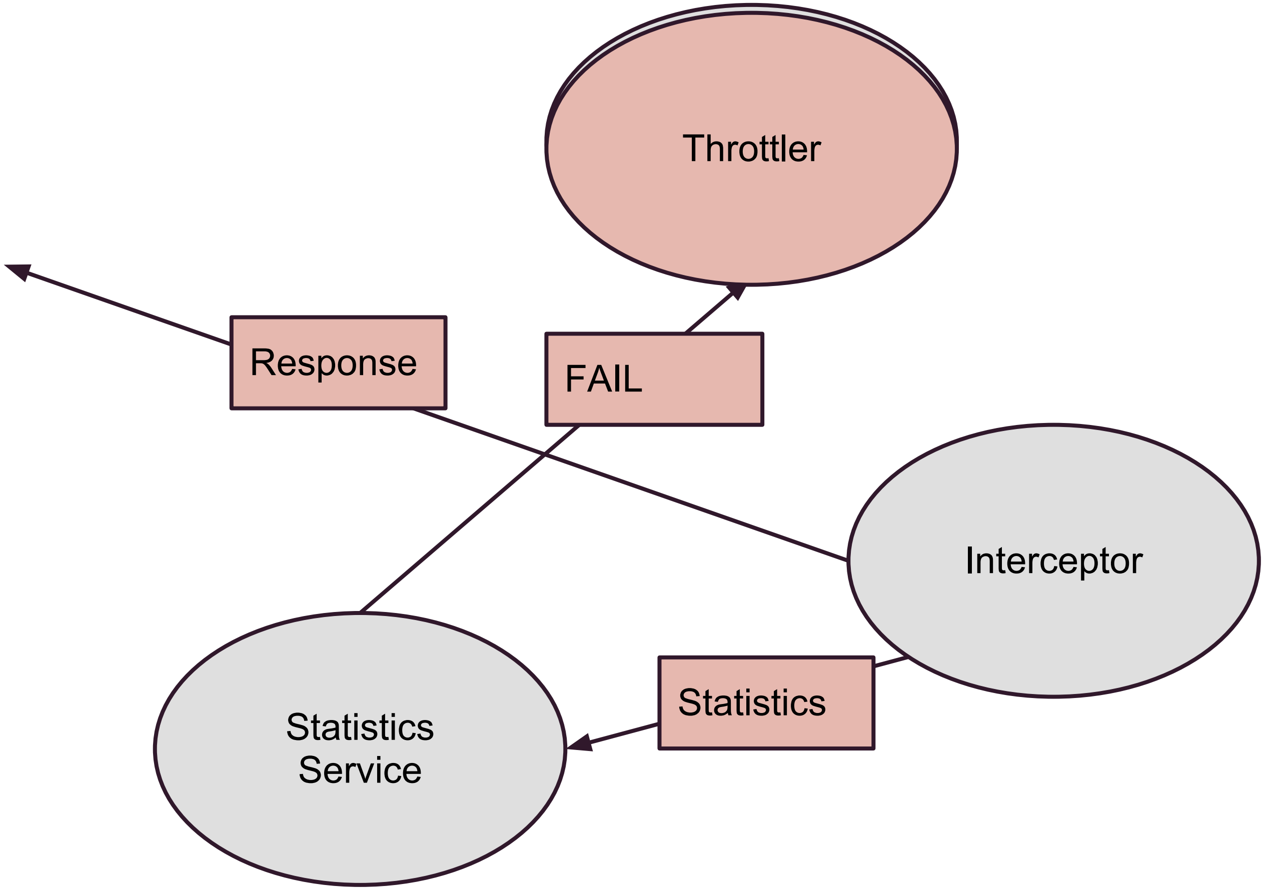
Throttling - Timeouts



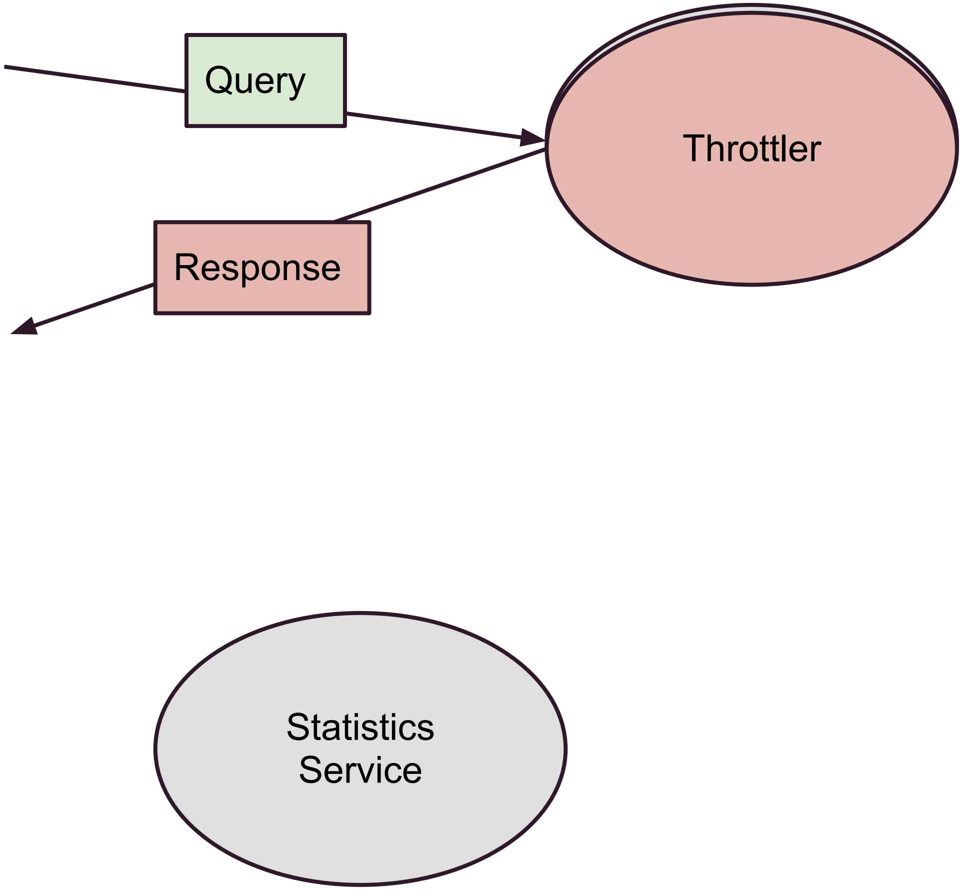
Throttling - Timeouts



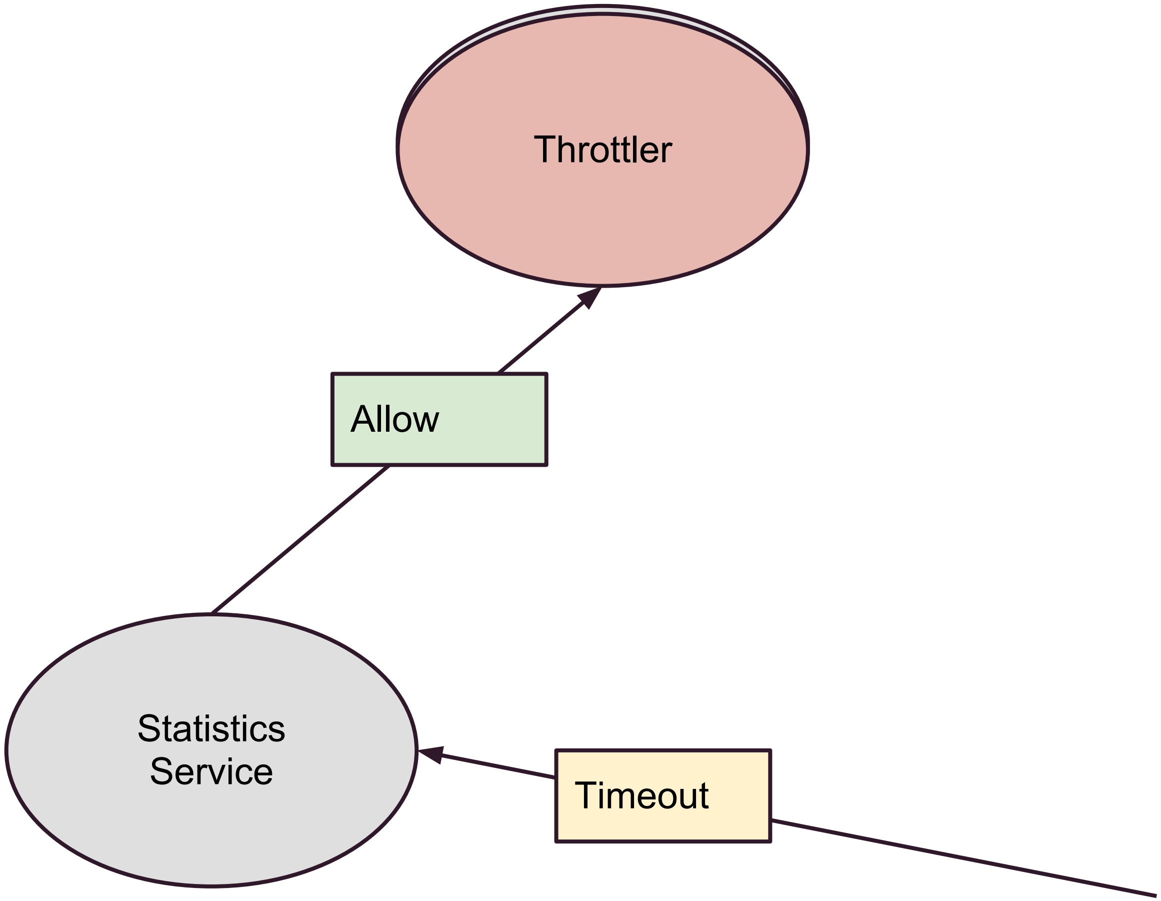
Throttling - Timeouts



Throttling - Dropping Queries



Throttling - Recovery



Throttler

Allow

Statistics Service

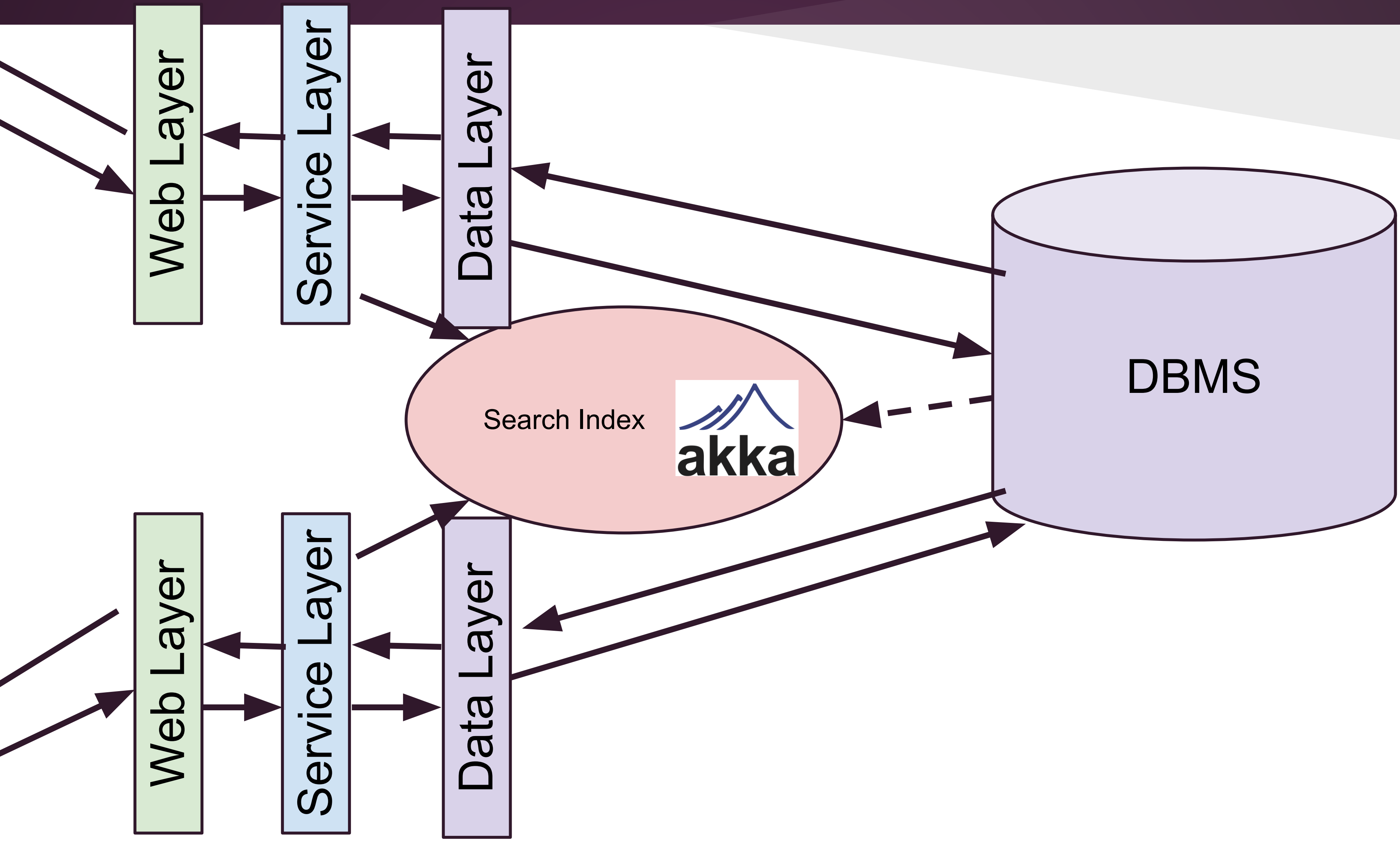
Timeout

What now?

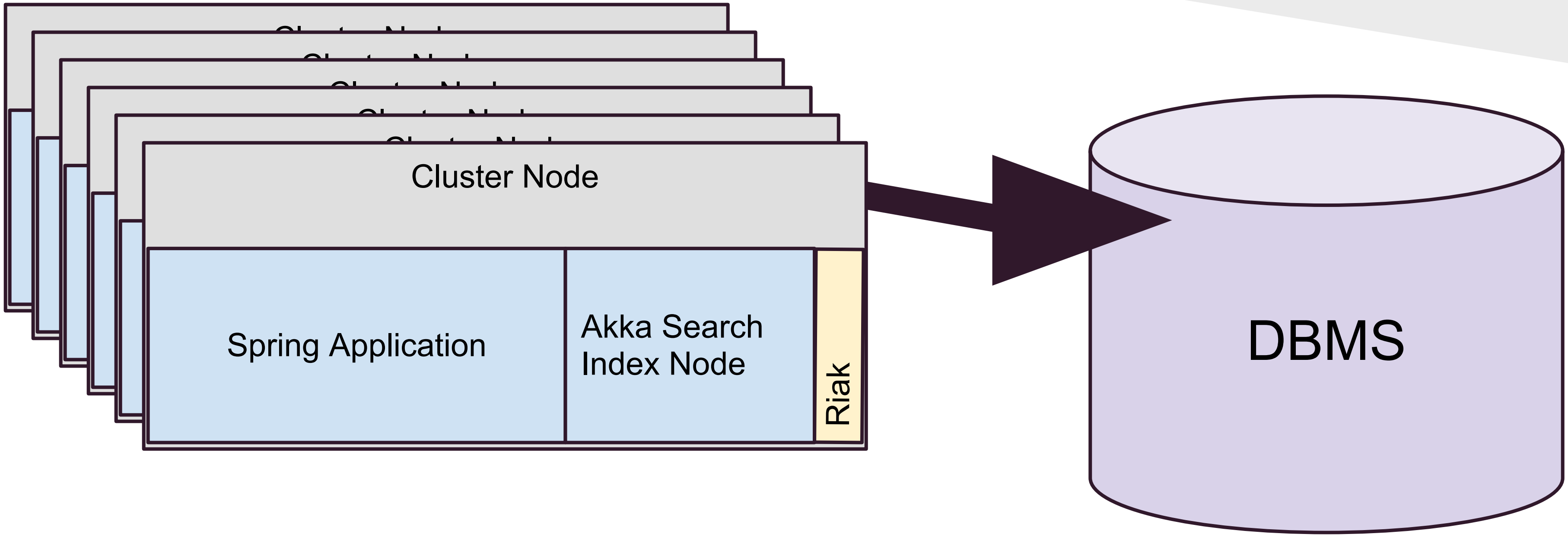
We installed our Search Tree on a huge-mongous server, and it's sucking up all 128GB RAM, and all 24 cores!

.... It's time to scale out

Remember we tried...



Now we want


Cluster Node
Spring Application
Akka Search Index Node
Riak
DBMS

Using Akka Clustering

- Akka now supports automatic cluster membership and notification
  - Considered experimental in 2.1
  - We're using 2.2-M2 for this talks
- Let's identify portions of our application and how we can scale them out

Setting up an Akka Cluster

Your Build

```
libraryDependencies ++= Seq(
  "com.typesafe.akka" %% "akka-actor" % "2.2-M2",
  "com.typesafe.akka" %% "akka-cluster-experimental" %
"2.2-M2")
```

```
<dependency>
  <groupId>com.typesafe.akka</groupId>
  <artifactId>akka-actor-${scala.version}</artifactId>
  <version>2.2-M2</version>
</dependency>
<dependency>
  <groupId>com.typesafe.akka</groupId>
  <artifactId>akka-cluster-experimental-${scala.version}</artifactId>
  <version>2.2-M2</version>
</dependency>
```

Application Configuration

```
akka {
  actor {
    provider = "akka.cluster.ClusterActorRefProvider"
  }
  remote {
    log-remote-lifecycle-events = off
    netty.tcp {
      hostname = "127.0.0.1"
      port = 0
    }
  }
  cluster {
    seed-nodes = [
      "akka.tcp://ClusterSystem@127.0.0.1:2551" ,
      "akka.tcp://ClusterSystem@127.0.0.1:2552" ]

    auto-down = on
  }
}
```

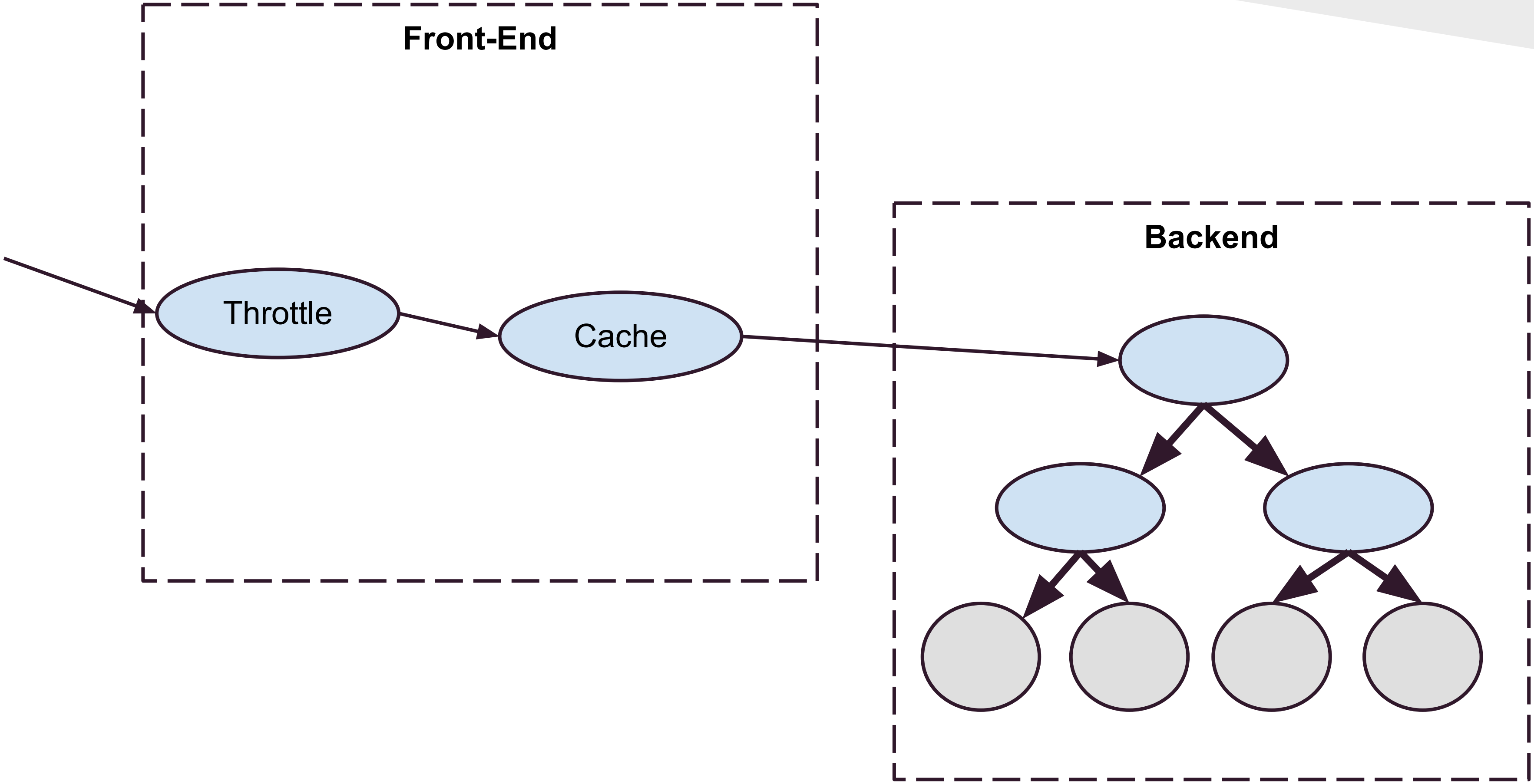
Actor references become cluster-ified

Nodes we look for to join the cluster

Code

```
val system =
  ActorSystem("ClusterSystem")
```

Remember the Actor Layout



Step #1

Let's automatically generate throttle and cache on every cluster node.

Creation code unchanged

```
system.actorOf(Props[FrontEnd],
                      "search-front-end")
```

This runs on every cluster node where we want a frontend

Registration on the FrontEnd

```
case class RegisterSearchTree(tree: ActorRef)

class FrontEnd extends Actor with ...{
  ...
  def receive: Receive = {
    case RegisterSearchTree(tree) =>
      // Now we create the cache + throttler
  }
}
```

The backend will now tell the frontend where it is, as each frontend cluster member registers.

Create Cluster-Aware Backend

```
class TreeTop .. extends Actor {

  val searchTree: ActorRef = createSearchTree()

  val cluster = Cluster(context.system)

  override def preStart(): Unit =
     cluster.subscribe(self, classOf[MemberUp])


  override def postStop(): Unit =
    cluster.unsubscribe(self)

  ...
}
```

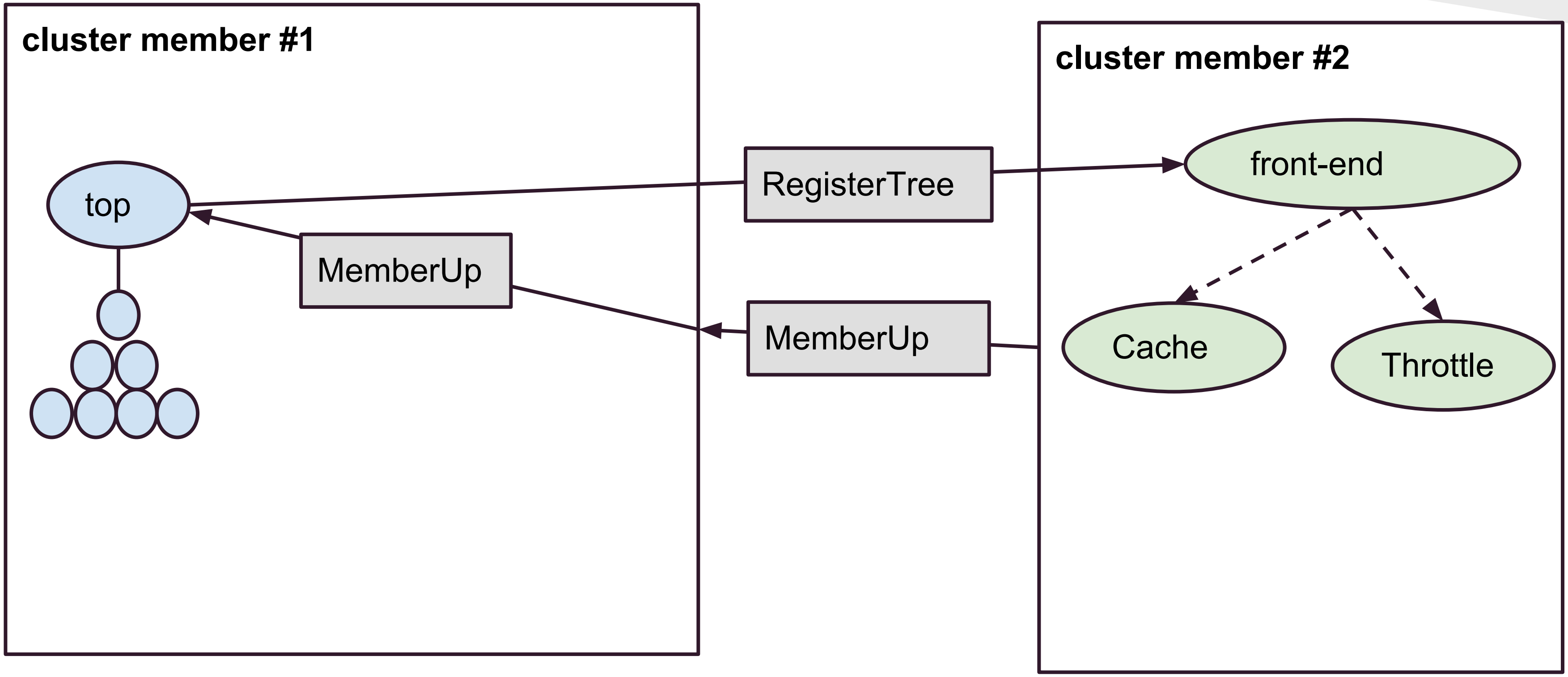
A new "top" on the scatter-gather tree registers for cluster membership events

Create Cluster-Aware Backend

```
def receive: Receive = {

  case q: SearchQuery => searchTree.tell(q, sender)

  case h: AddHotel => searchTree.tell(h, sender)

  case MemberUp(member) =>
    val memberFrontEnd =
      context.actorFor(
        RootActorPath(member.address) /
                      "user" / "search-front-end")
    memberFrontEnd ! RegisterTree(self)
}
```

Notify the local "search-front-end" when a member joins the cluster

What we have now



Just one node?

MemberUp message is still fired, so front end still finds the back end.

Recap #1

Can use Cluster membership notifications to register important services with each other.

Step #2

Ensure the Search Tree can survive node failure

Cluster Singleton Pattern

- Construct a Manager on every cluster node
- Managers communicate and elect a "leader"
- On leader failure, a new leader is chosen
- Create local proxy actor who keeps track of where the leader is.
- *Issues*
  - *Bottleneck*
  - *Delay in failure recovery (single point of failure)*

See: http://doc.akka.io/docs/akka/snapshot/contrib/cluster-singleton.html

Creating the Singleton

```
import akka.contrib.pattern.ClusterSingletonManager

system.actorOf(Props(
  new ClusterSingletonManager(
    singletonProps = _ => Props(new NodeManager("top",
db)),
    singletonName = "search-tree",
    terminationMessage = PoisonPill,
    role = None)),
  name = "singleton")
```

Creating the Singleton

```
    singletonProps = _ => Props(new NodeManager("top",
db)),
```

Creating the Singleton

```
ab));
    singletonName = "search-tree",
    terminationMessage = PoisonPill,
    role = None)),
```

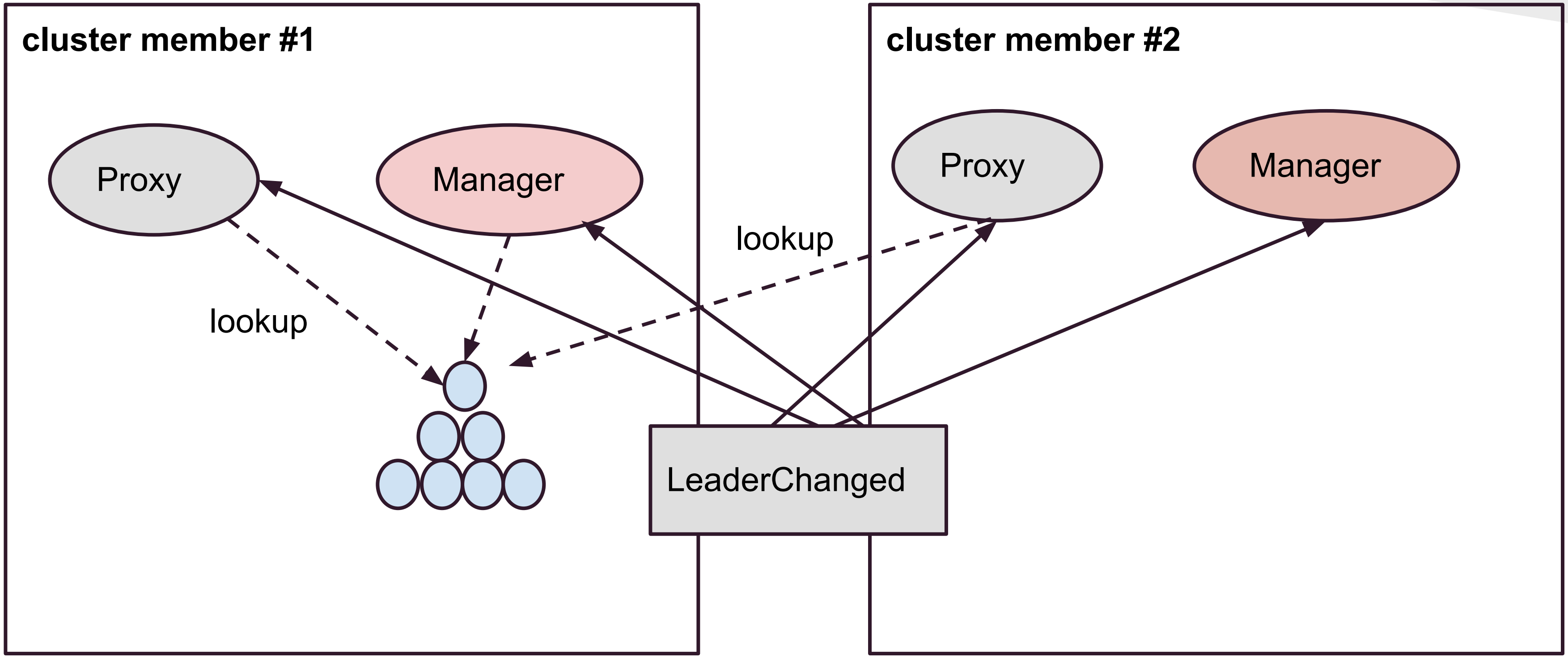
Creating the Proxy

```
class TreeTopProxy extends Actor {
  val cluster = Cluster(context.system)

  override def preStart(): Unit =
    cluster.subscribe(self, classOf[LeaderChanged])

  override def postStop(): Unit =
    cluster.unsubscribe(self)


  var leaderAddress: Option[Address] = None
  ...
```

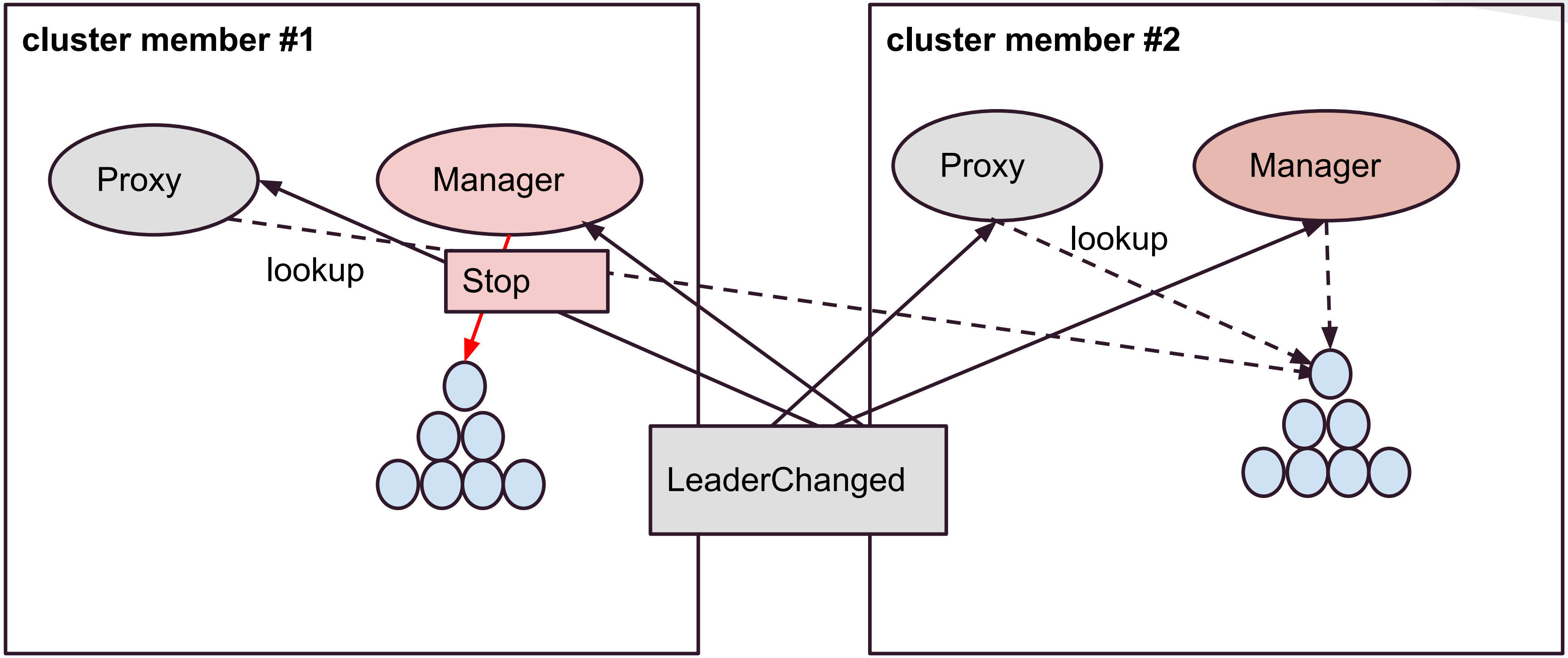
Creating the Proxy (part 2)

```
  ...
  def receive = {
    case state: CurrentClusterState =>
      leaderAddress = state.leader
    case LeaderChanged(leader) =>
      leaderAddress = leader
    case msg => singleton foreach { _ forward msg }
  }
  def singleton: Option[ActorRef] =
    leaderAddress map (a =>
       context.actorFor(RootActorPath(a) /
        "user" / "singleton" / "search-tree"))
}
```

Visualizing



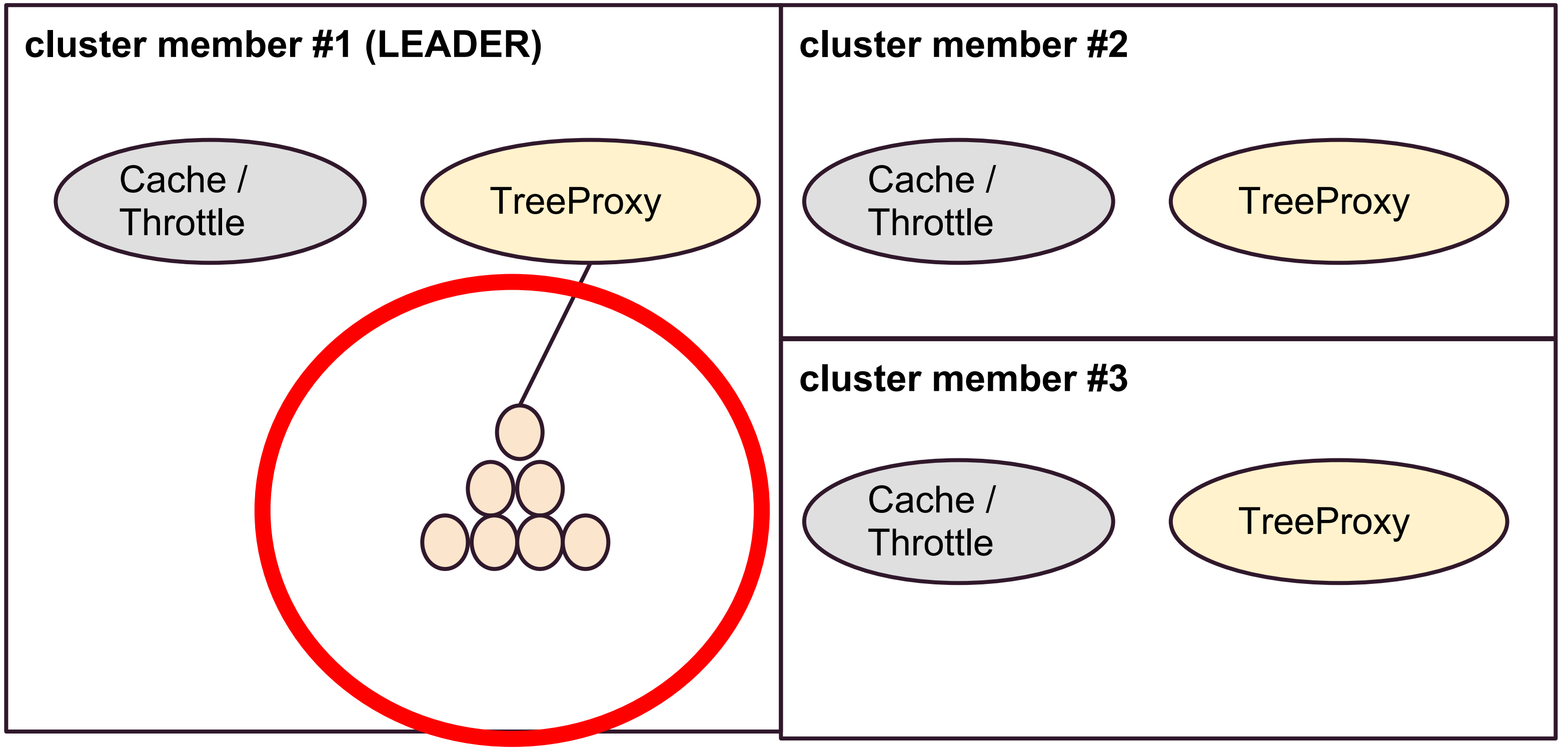
Visualizing



Step #3

Fragment the Search Tree

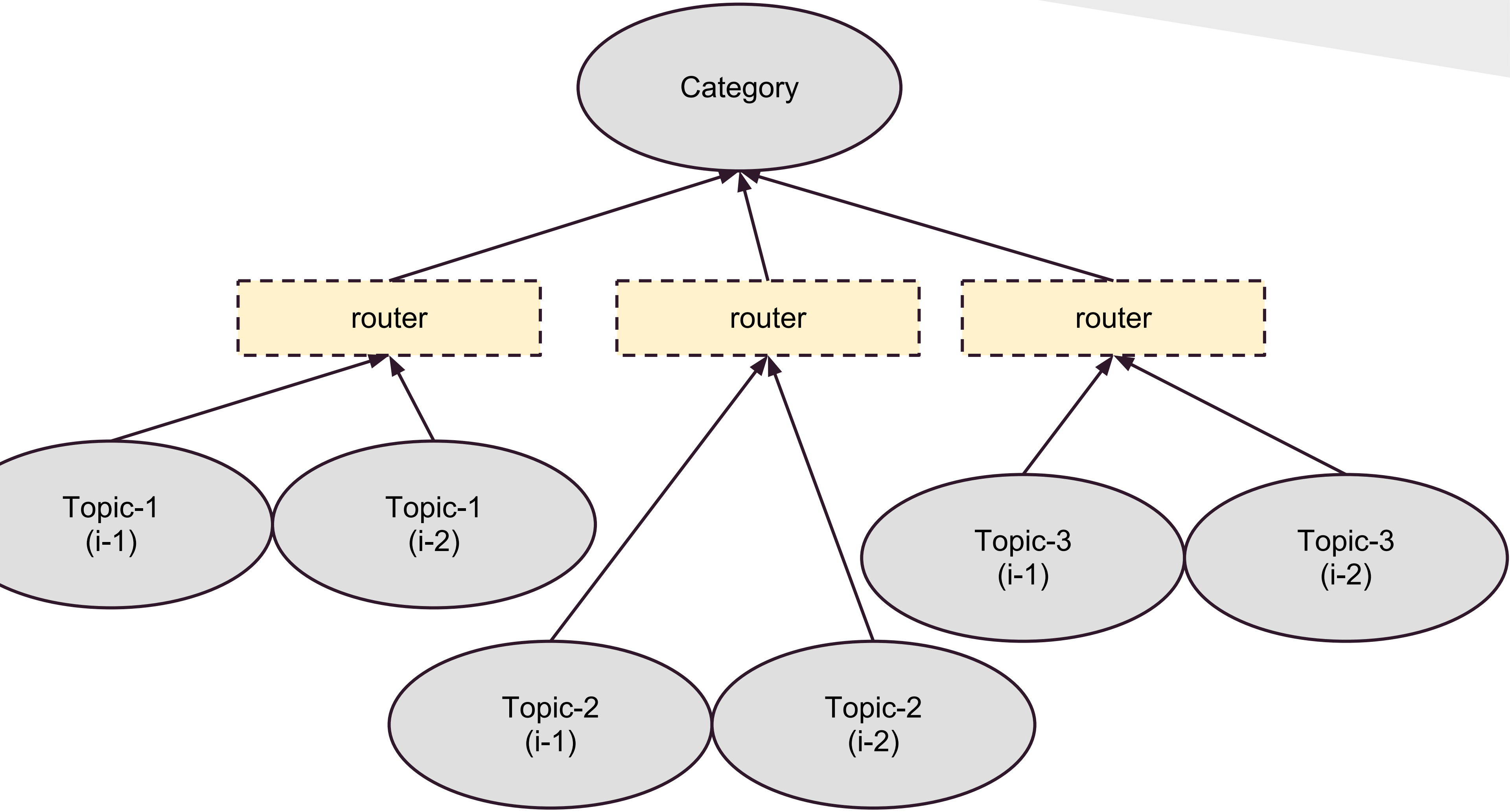
We still have scaling issues



What are routers?

- Layer between ActorRef / Actors
- Route messages to underlying actors
- Non-Cluster Examples:
  - Round Robin
  - Scatter Gather (first-found)
  - Consistent Hashing
  - Random
  - Broadcast

Tree with local routers



Clustered Router

Like local routers, but actor instances may be on other nodes.

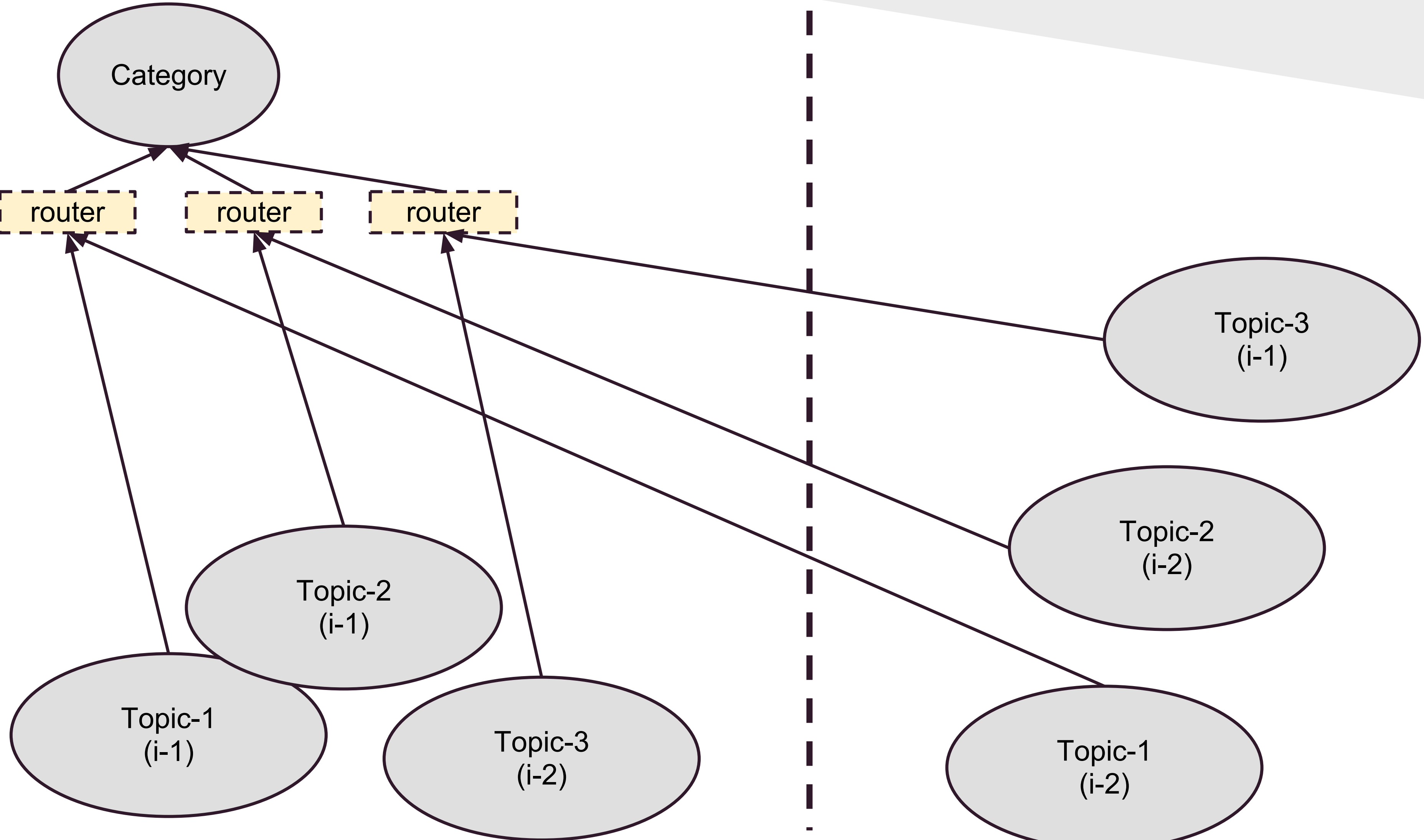
Clustered Router

```
props.withRouter(
   ClusterRouterConfig(
         BroadcastRouter(1),
         ClusterRouterSettings(
               totalInstances = 3,
               maxInstancesPerNode = 1,
               allowLocalRoutees = true,
               useRole = None
         )
   )
)
```

Local Router

Cluster Router

Tree with remote routers



Metrics based Routing

- Requires "sigar" dependency to enable
- Examples:
    - AdaptiveLoadBalancingRouter
        - heap
        - cpu
        - load
        - mix

Recap

Clustered system design with Actors

Actor Systems

- Partition state into small pieces
- Communicate with immutable messages
- Spawn new actors to track temporary state
- Design as a Topology
- Partition threads on the topology
- Bubble errors on the topology

Clustered Actor Systems

- Partition Topology on nodes in the cluster
  - Limit instances with routers
  - Register with other clusters using cluster listeners
  - Use roles to fragment actors across the cluster
  - Keep "singleton" actors on the leader or role leader
- Avoid excessive inter-node messaging
  - Use statistics based routing
  - Fragment in 'large pieces'
- Allow time for cluster convergence and fault detection

Key Point

Ensure your system can recover from failure

Resources

- http://github.com/jsuereth/intro-to-actors
  Example code (clusters branch)
- http://akka.io
  Akka concurrency framework for the JVM

Questions?