Mathieu ANCELIN
Alexandre DELEGUE

Mathieu ANCELIN

- Développeur @SERLI
- Scala, Java, web & OSS
  - ReactiveCouchbase, Weld-OSGi, etc ...
  - Poitou-Charentes JUG
- Membre de l'expert group MVC 1.0
- @TrevorReznik

SERLI

Alexandre DELEGUE

- Développeur @ SERLI
  - Java
  - Scala
  - Web
  - spring, play, …
- @chanksleroux

SERLI

SERLI

- Société de conseil et d'ingénierie du SI
- 75 personnes
- 80% de business Java
- Contribution à des projets OSS
- 10% de la force de travail sur l'OSS
- Membre de l'EG JSR-346
- Membre de l'OSGi Alliance
- www.serli.com @SerliFr

SERLI

Les technologies présentées sont inspirées de technologies réelles
Les applications qui en découlent sont fictives
Toute ressemblance avec des applications existantes n’est que fortuite

SERLi

www.amazing.com

Amazing Store

Hello test | 0 items | Logout

Search a product

Search

Mitraillette années 30 canon court

Mitraillette années 30. S ...

3599.0 €

Add to cart

Mitraillette années 30

Mitraillette années 30. P ...

3599.0 €

Add to cart

Arbalète et accessoires

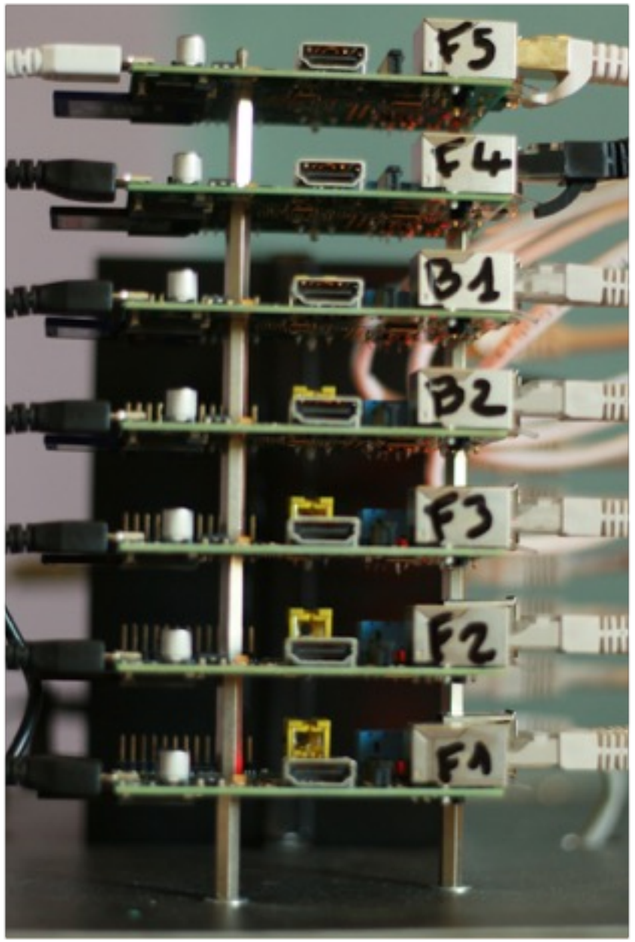
Magnifique arbalète pouli ...

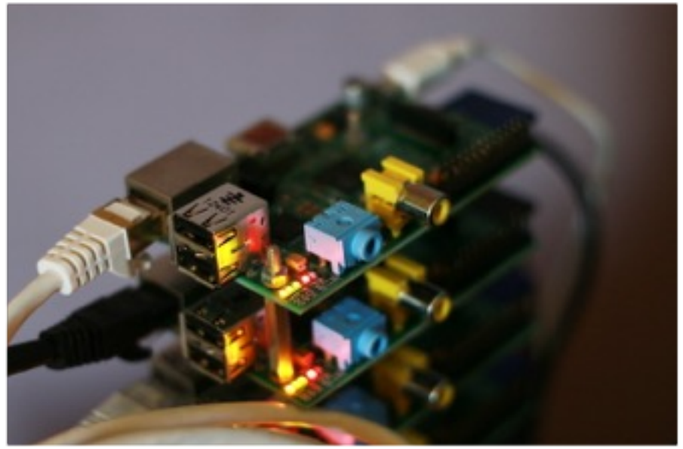
200.99 €

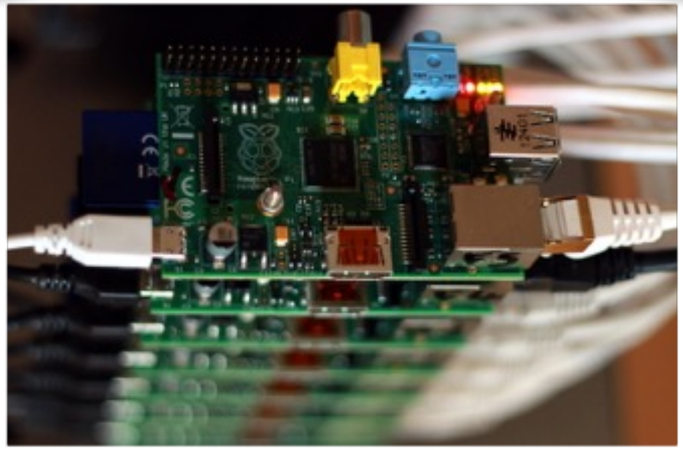
Add to cart

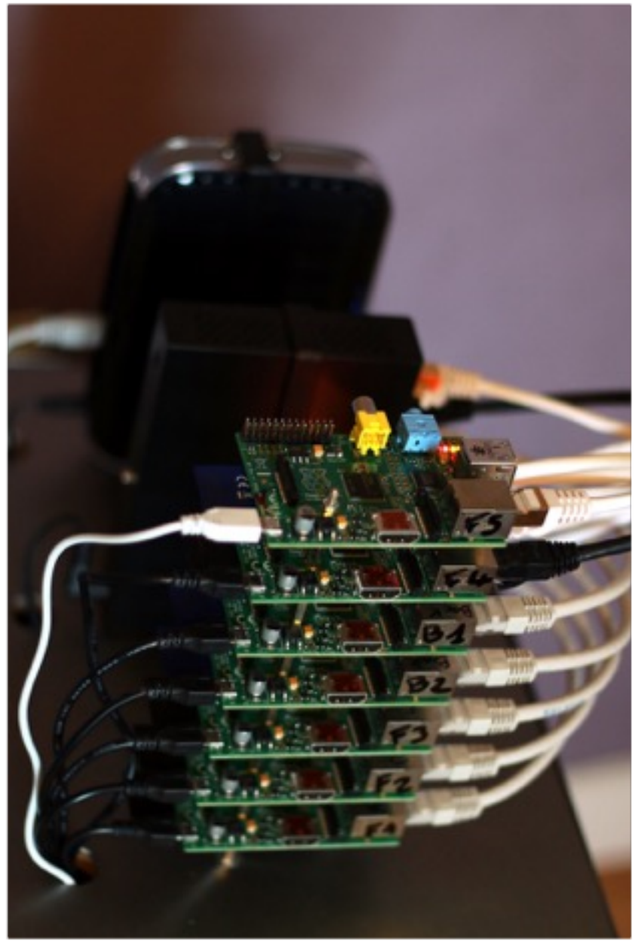
SERLI

Mobile infrastructure





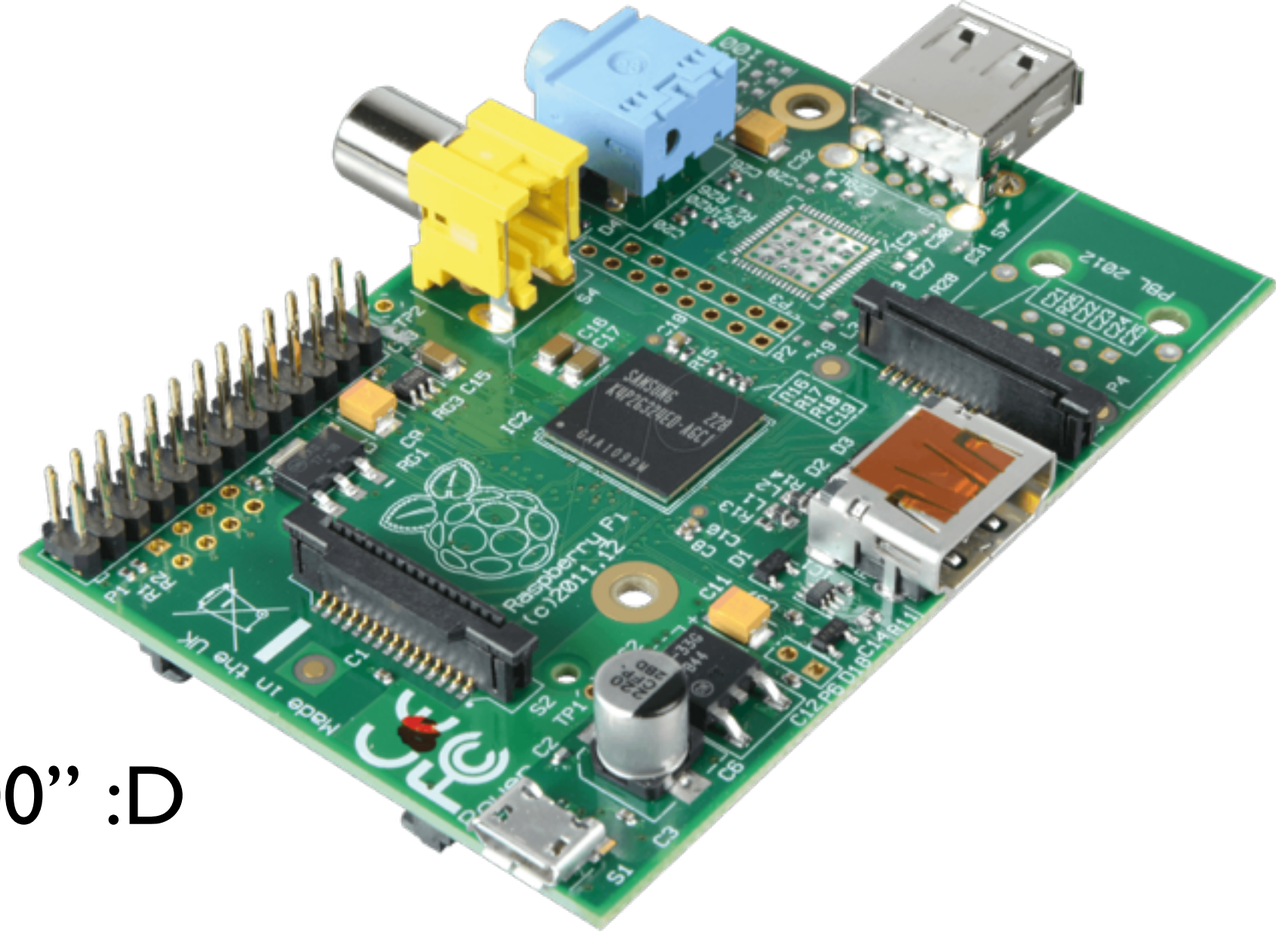




SERLI

Raspberry Pi

- CPU 700 MHz
- 512 Mo de RAM
- 2 ports USB
- Carte SD
- Port ethernet



- “Un ordinateur comme en 2000” :D
- Le tout pour ~35$

Mobile infrastructure

Raspberry pi
x 7

Routeur wifi

Switch

Demo

Résister à la charge

Résister aux pannes

Absorber une hausse de trafic

SERLI

Approche classique

Monolithique

SERLI

Latence

Bloquant

Charge

SERLI

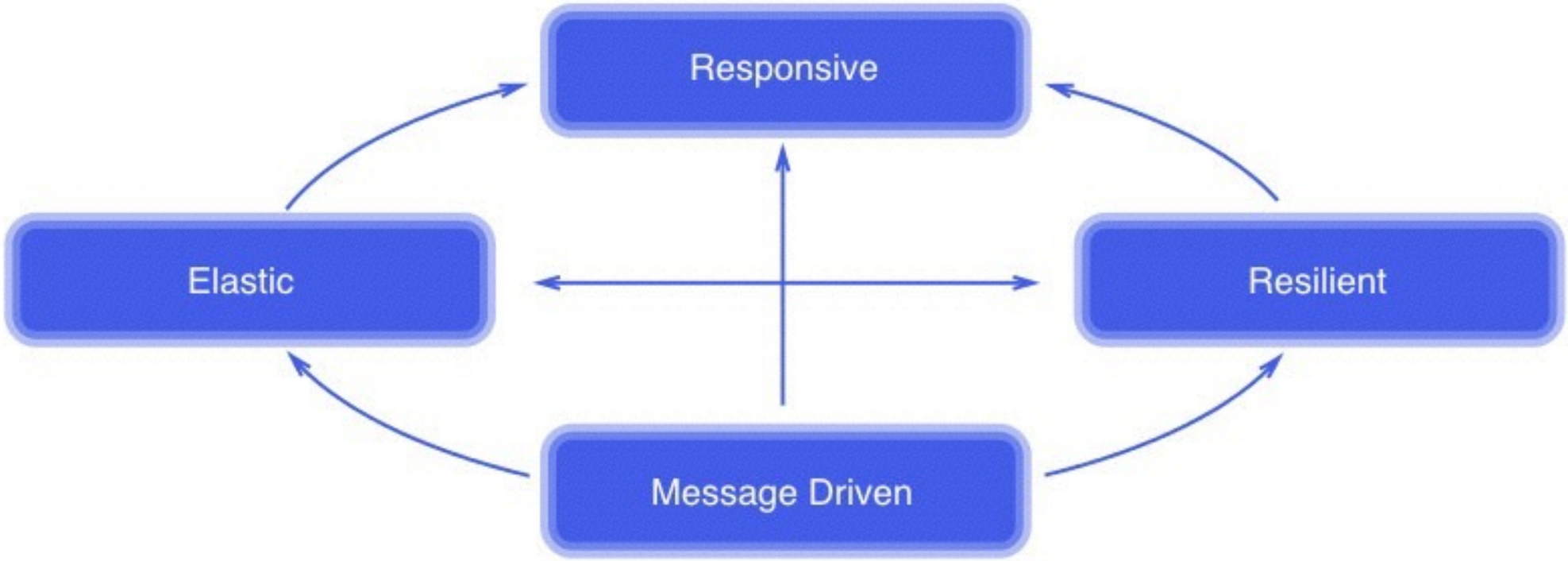
Scalabilité

Comment faire ?

SERLI

Reactive Manifesto

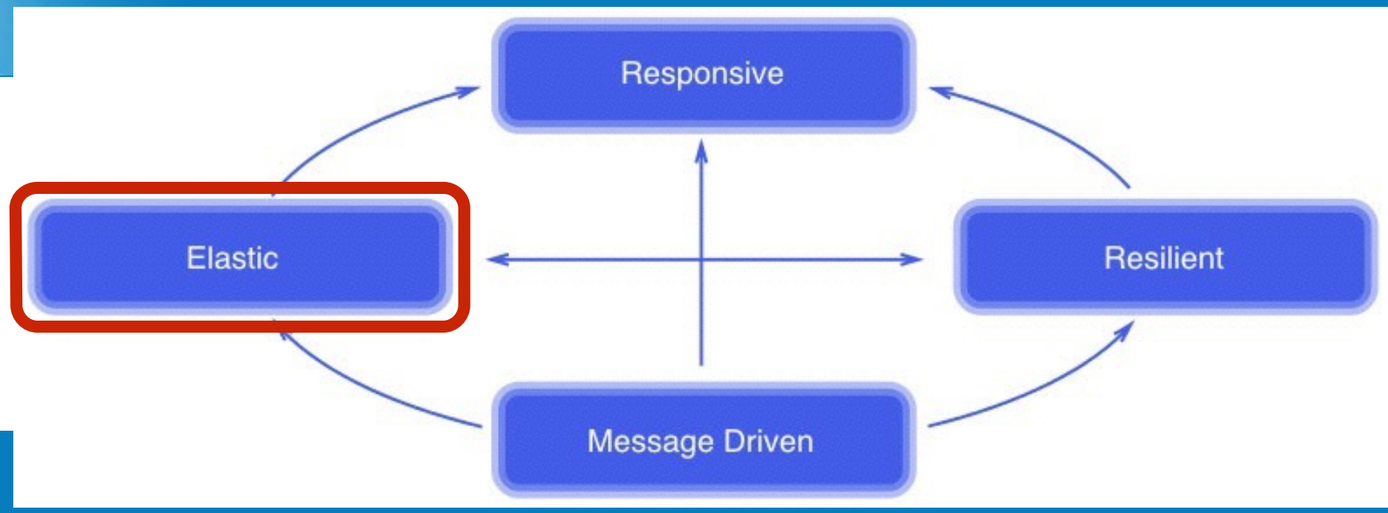
Reactive Manifesto



Reactive manifesto

- **react to event :** the event-driven nature enables the following qualities
- **react to load :** focus on scalability by avoiding contention on shared resources
- **react to failure :** build resilient systems with the ability to recover at all levels
- **react to user :** honor response time guarantees regardless of load

Scalable / React to load



Overview

Frontend

Cart

Identity

elasticsearch.

Cassandra

Cassandra

Cassandra

Backend

SERLI

Datas

elasticsearch.

Cassandra

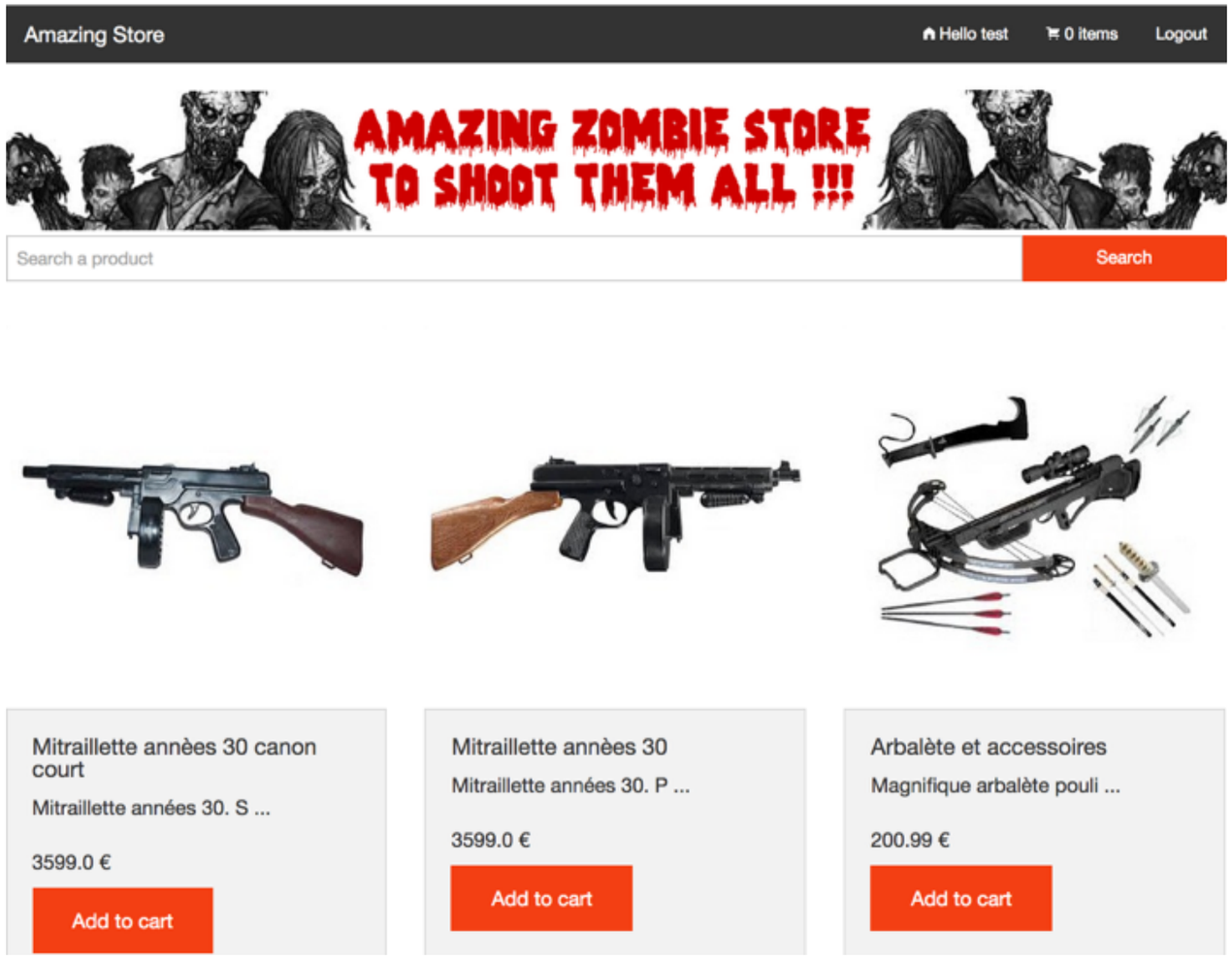
Mini / Micro Services

- Une application par domaine fonctionnel
  - store-frontend : présentation du contenu
  - store-identity : authentification / gestion de compte
  - store-cart : panier
  - store-backend : administration du site

Stateless

- Chaque application est stateless
  - aucune donnée n'est stockée dans l'application (pas de cache, pas de fichier …)
- Chaque application peut être clonée

Frontend

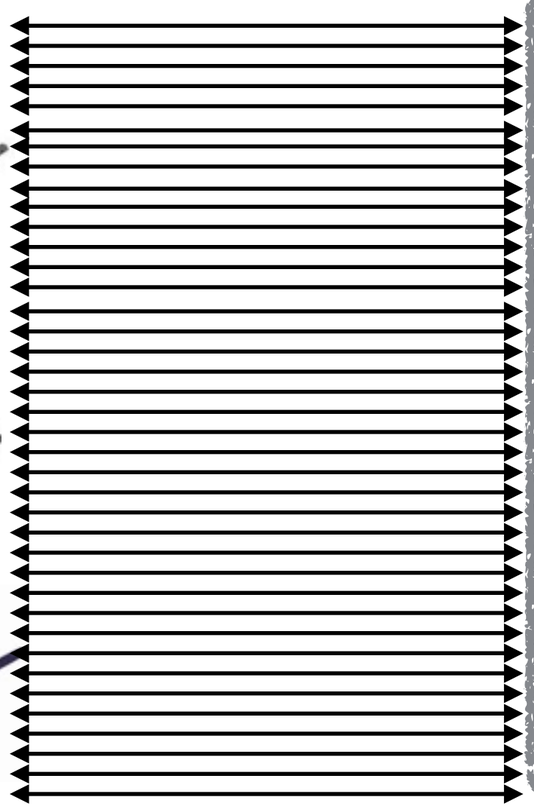


- 100 % html
- Indexation par les moteurs de recherche
- stateless
- une url == un contenu

SERLI

Limiter la charge

Cache



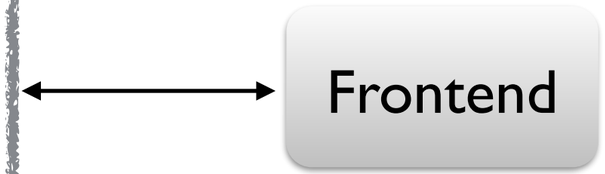
Cache

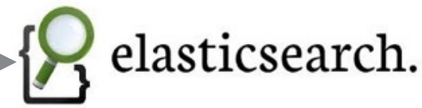
Browser

CDN

nginx/http

varnish





Optimisations

elasticsearch.

Base de données ?

Cache
+
recherche full text

Modèle de données

```
{
    id: "04abe480-2521-11e4-acde-f7b0d99b8321",
    label: "Product number 1",
    description: "A description …",
    image: "image.jpeg",
    price: 1.5,
    fragments: [{
            type: "search",
            html: " <div >...</div>"
        },{
            type: "cart",
            html: " <tr >...</tr>"
        }
    ]
}
```

Données indexées pour la recherche

HTML pré-généré

SERLI

Recherche

GET /products?q=some%20text

Frontend

- Concat fragments
- Include into main template

Search some text

SERLI

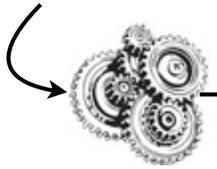
Créer / mettre à jour

UpdateProduct

Backend

ProductUpdated

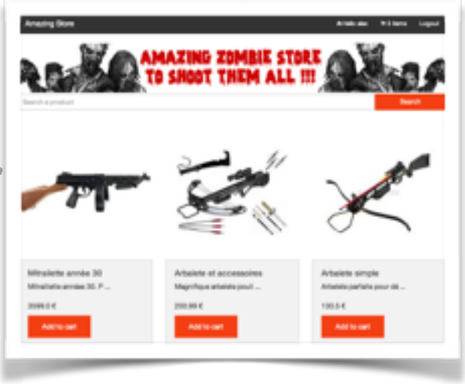
Frontend



HTML

EventSource

HTML



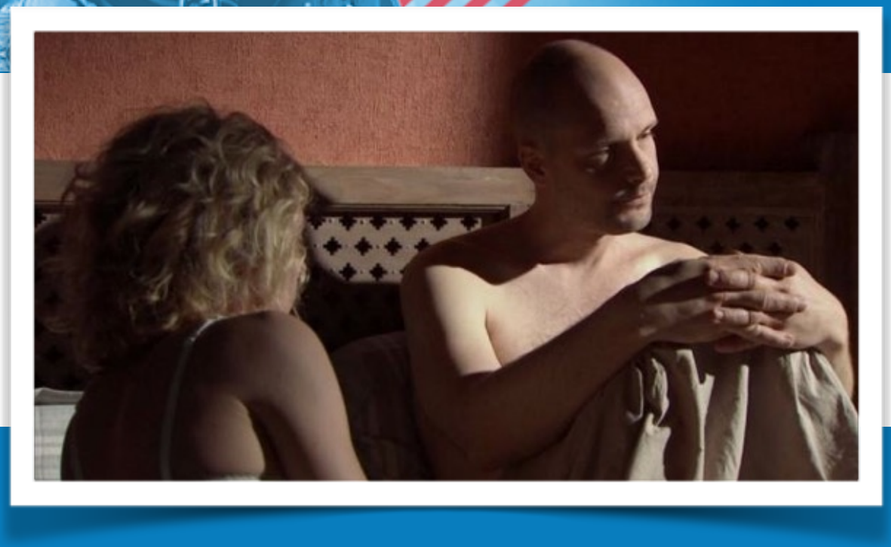
ProductUpdated

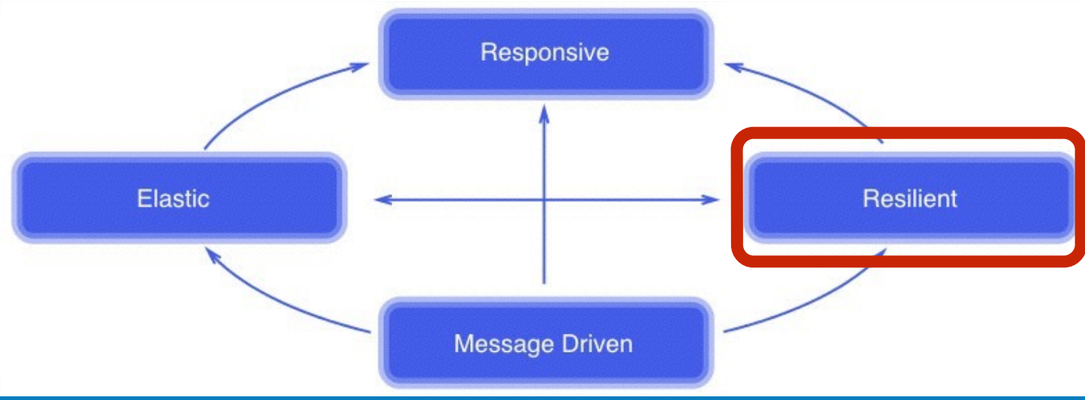
Cassandra

elasticsearch.

SERLi

Resilient / react to failure





SERLI

Infrastructure

NGINX

Identity

Frontend

Identity

Cart

Backend

Frontend

Cart

Backend

Identity

Cassandra

elasticsearch.

SERLI

Infrastructure

NGINX

Identity

Frontend

Identity

Cart

Backend

Frontend

Cart

Backend

Identity

Cassandra

elasticsearch.

SERLI

Infrastructure

NGINX

Identity

Frontend

Identity

Cart

Backend

~~Frontend~~

~~Cart~~

Backend

Identity

Cassandra

elasticsearch.

Infrastructure

NGINX

Identity

Frontend

Identity

Cart

Backend

~~Frontend~~

~~Cart~~

~~Backend~~

~~Identity~~

Cassandra

elasticsearch.

Infrastructure

NGINX

Identity

Frontend

Identity

~~Cart~~

~~Backend~~

~~Frontend~~

~~Cart~~

~~Backend~~

~~Identity~~

Cassandra

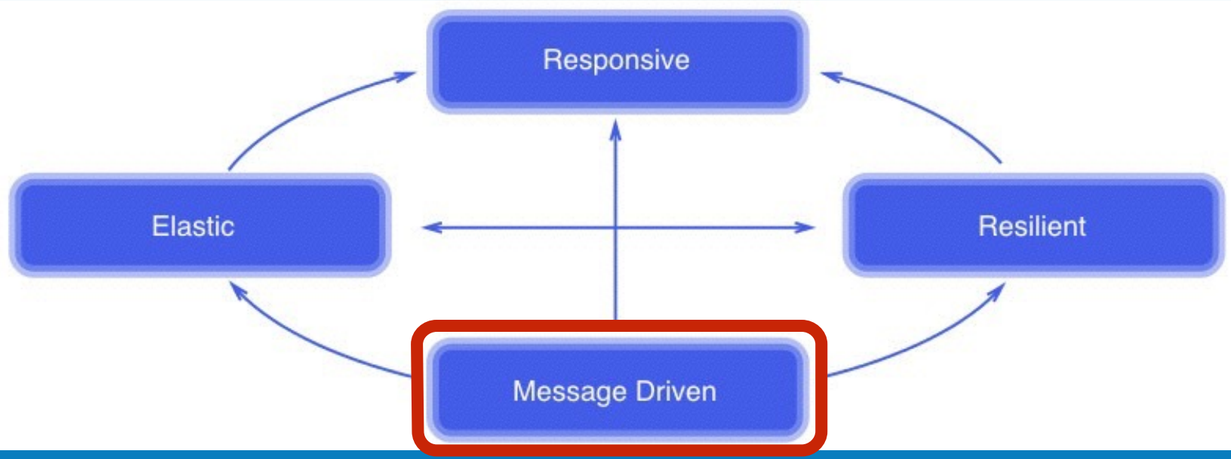
elasticsearch.

Demo

Let It Crash !!!

SERLi

Event-driven / react to event



Akka ?

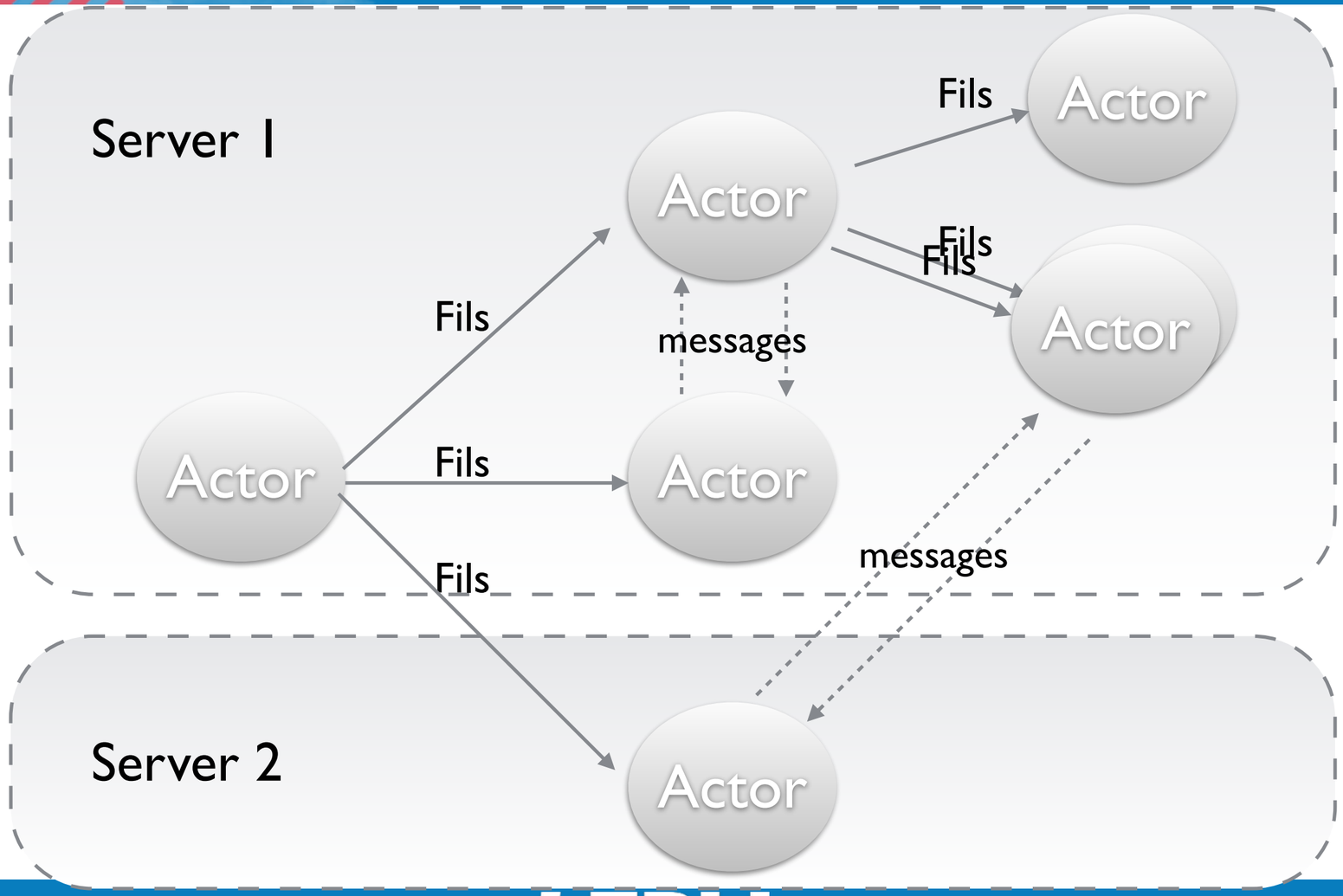
Akka ??

SERLI

Akka

- Akka
  - Modèle acteur
  - Un acteur = Une entité statefull
  - Communication entre acteurs par messages (même à distance)
  - Un acteur peut créer/détruire des enfants
  - Un acteur peut surveiller d'autres acteurs
  - Plus de problèmes de concurrence, asynchrone par nature
  - Résistant aux pannes
  - Java or Scala

Akka


Server 1
Actor
Fils
Actor
Fils
Fils
Actor
messages
Actor
Fils
Actor
Fils
messages
Server 2
Actor

Akka messages

```
import akka.actor.{Props, ActorSystem, ActorLogging, 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")
greeter ! Greeting("Charlie Parker")
```

Struts 2

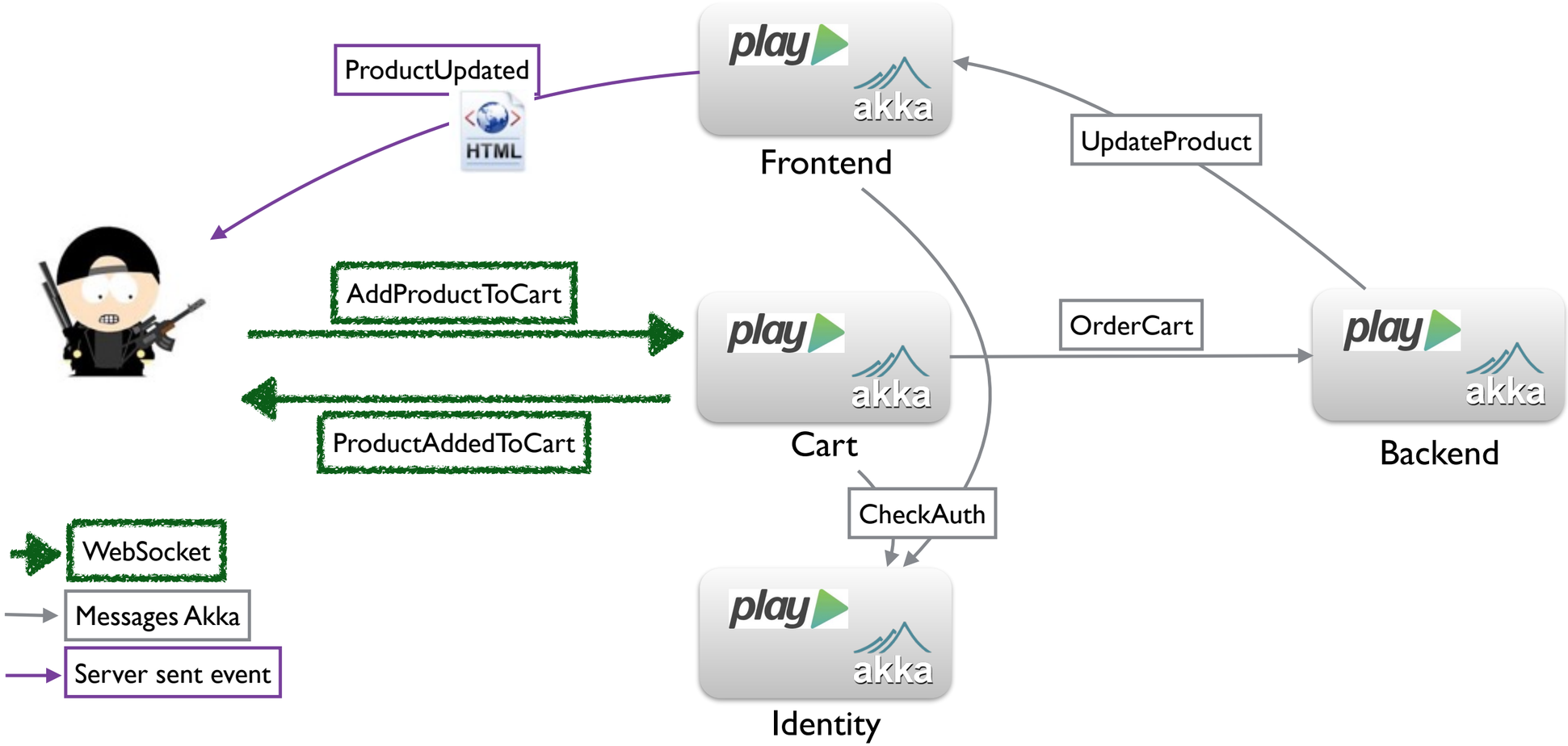
Play 2

- Framework web
- java or scala
- Support pour les websocket et le server sent event
- Asynchrone
  - pré-requis pour une application orientée événements

Play async

```
case class ListProductsQuery()

class ProductView extends Actor {
  override def receive: Receive = {
    case ListProductsQuery() => models.Product.list() pipeTo sender()
  }
}

class ProductsController  extends Controller {
  private def listProducts(): Future[List[models.Product]] = {
    (Actors.productView() ? ListProductsQuery()).mapTo[List[models.Product]]
  }
  def index() = Action.async {
    listProducts().map(products => Ok(views.html.index(products)))
  }
}
```

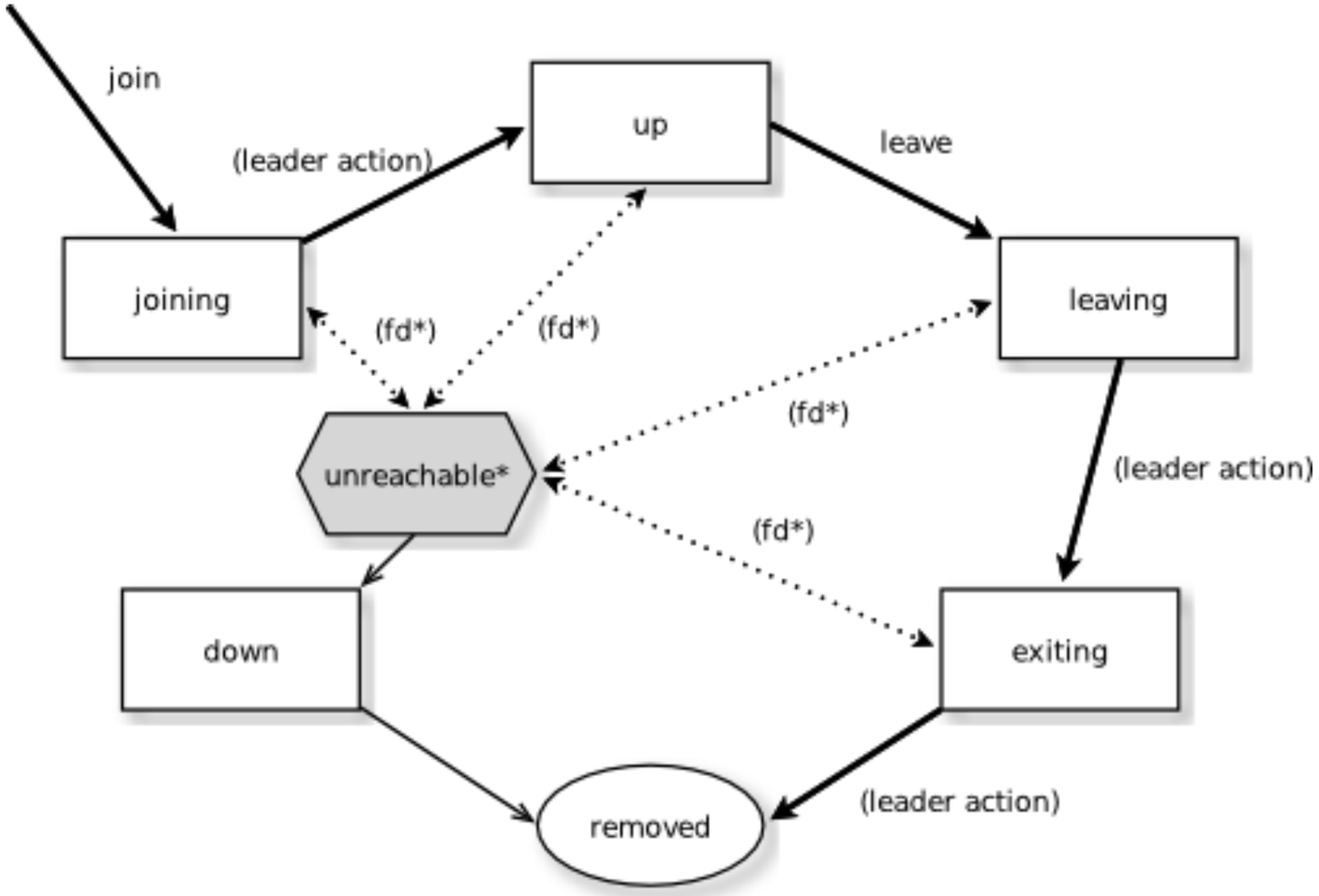
Messages


ProductUpdated
HTML
play
akka
Frontend
UpdateProduct
AddProductToCart
ProductAddedToCart
play
akka
Cart
OrderCart
play
akka
Backend
CheckAuth
play
akka
Identity
WebSocket
Messages Akka
Server sent event

Cluster

- Utilisation de Akka-cluster
- Librairie permettant de former un cluster de systèmes d'acteurs
  - simple service d'adhésion
  - tolérant aux pannes
  - décentralisé (P2P, gossip)
  - pas de SPOF
  - pas de SPOB
  - détection des pannes

Akka cluster



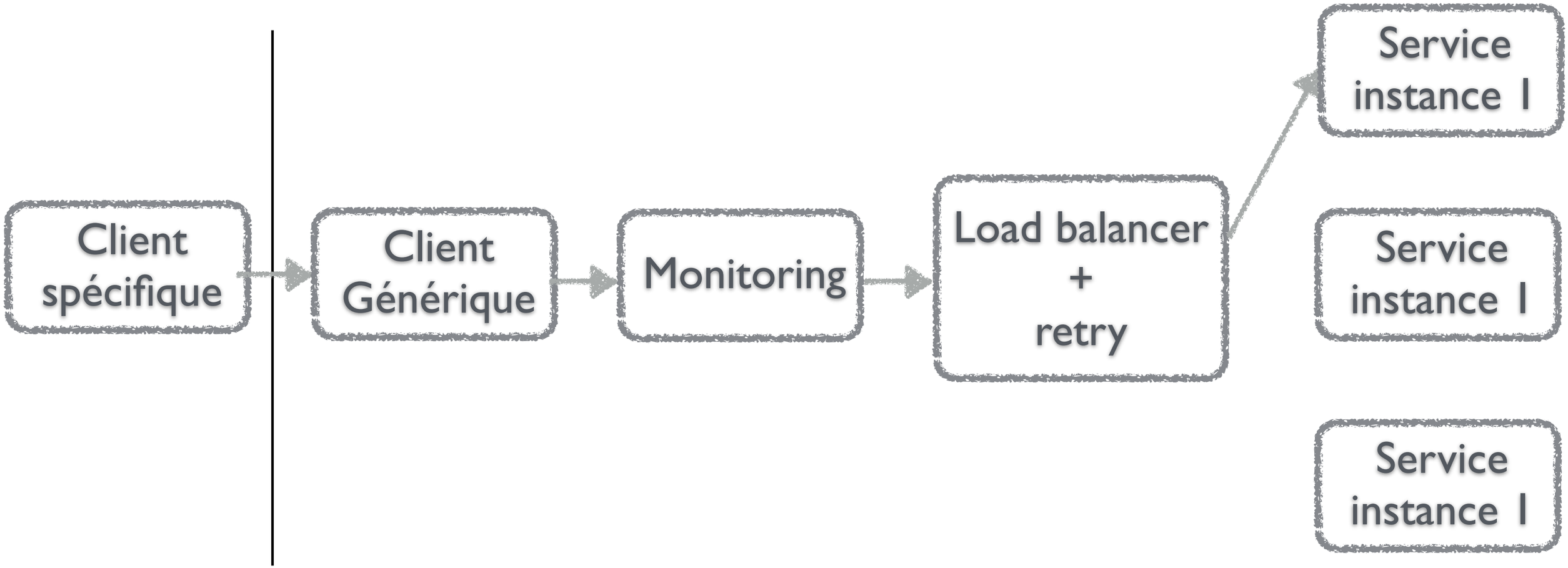
Cluster services

- Librairie de découverte de services distribués
- Exposition descripteurs de services (URL, protocole, version, nom)
- Repose sur les memberships du cluster Akka
- Clients de services
  - HTTP, Akka, Thrift, Memcached …
- Petits plus
  - Circuit breaker
  - Monitoring
  - Load balancing (pas très intelligent)
  - Retry with exponential back off

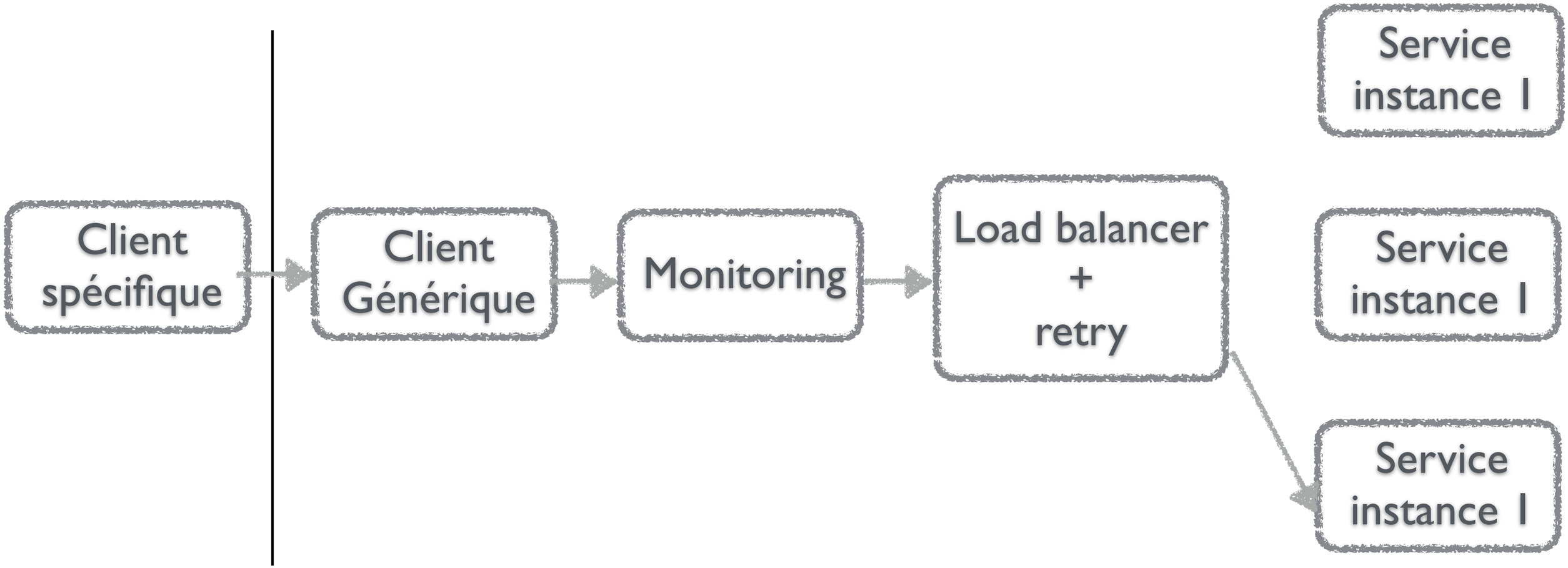
finagle

+ Eureka

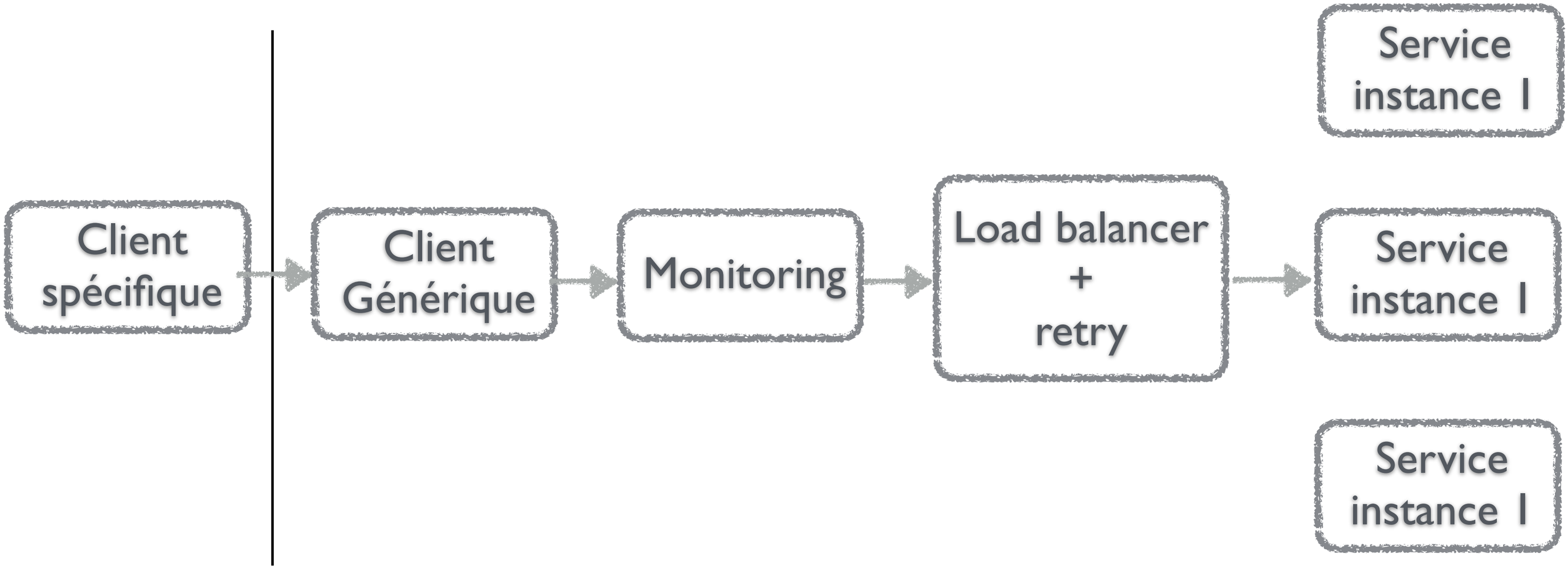
Cluster services



Cluster services



Cluster services



Cluster service

```
services {
  boot {
    host = "frontal2"
    port = 2551
    role = "APP"
  }
  seed = ["frontal1:2551","frontal3:2551"]
}
```

SERLI

Cluster service

```
object Env {

  implicit val ec = ExecutionContext.fromExecutor(Executors.newCachedThreadPool())

  val configuration = Configuration.load()
  val APP_NAME = configuration.getString("app.name").getOrElse("IDENTITY")
  val MACHINE_NAME = configuration.getString("services.boot.host").getOrElse("127.0.0.1")

  val registry = Services(s"$MACHINE_NAME-$APP_NAME").startFromConfig()

  val service1 = Service(name = "USERINFO", url = s"http://$MACHINE_NAME:$8000/user", version = Some("1.2.0"))
  val service2 = Service(name = "USERCHECK", url = s"akka.tcp://mysystem@$MACHINE_NAME:$2251/user/service2")

  val registration1 = registry.registerService(service1)
  val registration2 = registry.registerService(service2)

  def shutdown() = {
    registration1.unregister()
    registration2.unregister()
    registry.stop()
  }
}
```

Cluster service

```scala
case class User(id: UUID, email: String, name: String)

object User {
  val userFormat = Json.format[User]
}

trait Service1 {
  def findUser(email: String): Future[Option[User]]
}

trait Service2 {
  def isValidUser(email: String, password: String): Future[Option[(Boolean, User)]]
}
```

Cluster service

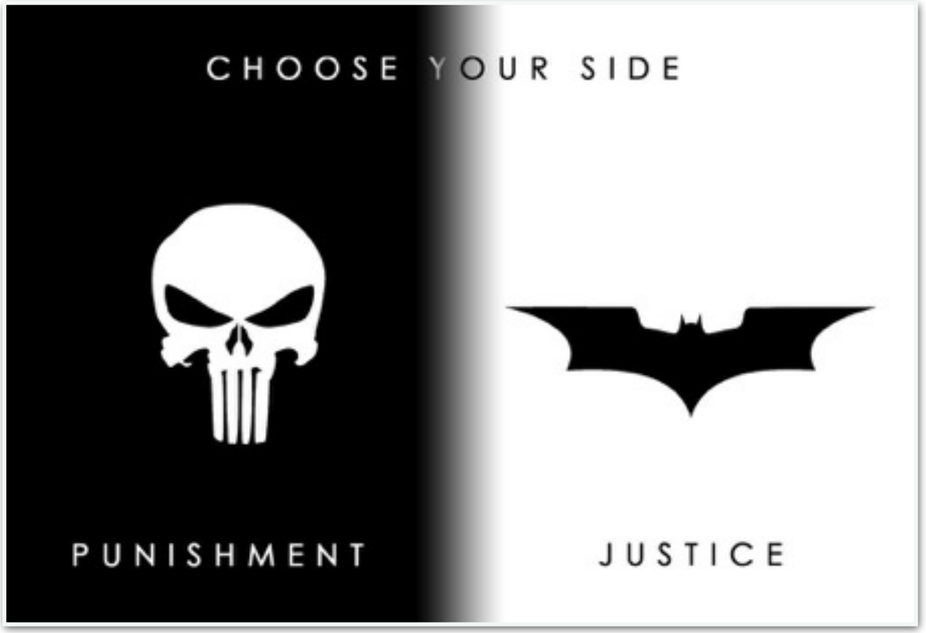
```
object Service1Client extends Service1 {

  import Env.ec

  val client = Env.registry.httpClient(name = "USERINFO",
                                       version = Some("1.2.0"),
                                       retry = 5)

  def findUser(email: String): Future[Option[User]] = {
    client.withParams("email" -> email).get()
      .map(r => Json.parse(r.body().string()).asOpt(User.userFormat))
  }
}
```

Cluster service

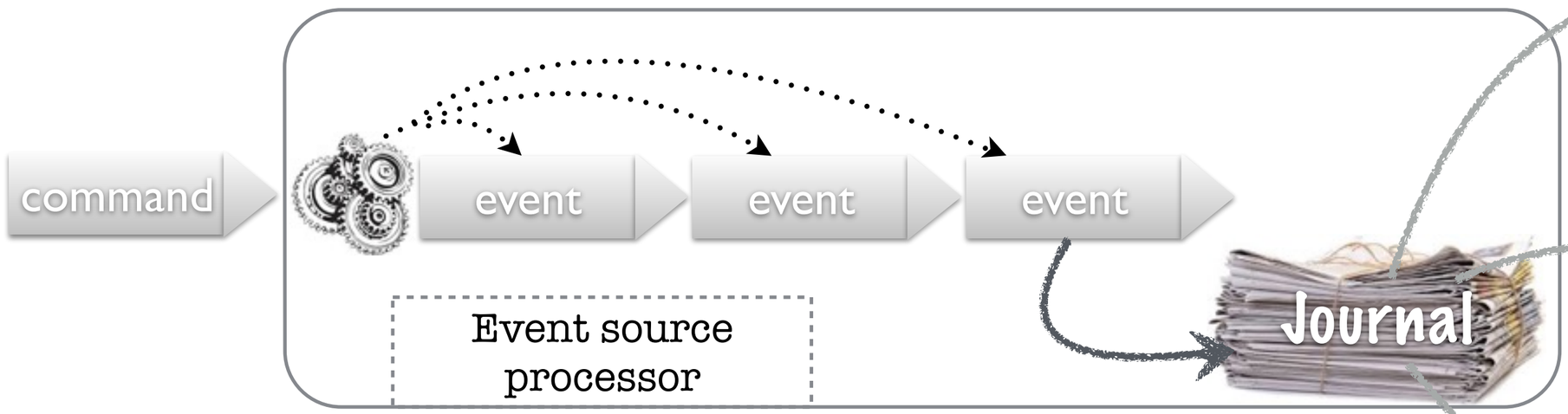
```
object Service2Client extends Service2 {
  import Env.ec

  case class UserValidRequest(email: String, password: String)
  case class UserValidResponse(valid: Boolean, user: Option[User])

  val client = Env.registry.akkaClient(name = "USERCHECK", retry = 5)

  def isValidUser(email: String, password: String): Future[Option[(Boolean, User)]] = {
    client.ask[UserValidResponse](UserValidRequest(email, password)).map {
      case UserValidResponse(_, None) => None
      case UserValidResponse(valid, Some(user)) => Some((valid, user))
    }
  }
}
```

CQRS & EventSourcing

- Command Query Responsibility Segregation
  - Command : Enregistrement
  - Query : Lecture
  - 2 modèles distincts
  - Séparation des services
- Event sourcing
  - Stockage des événements

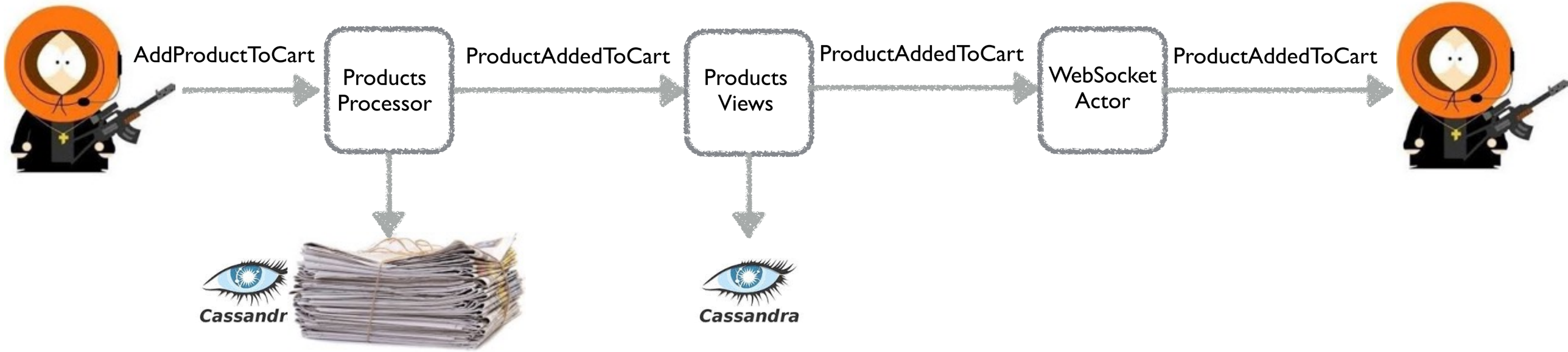


Persistence



SERLi

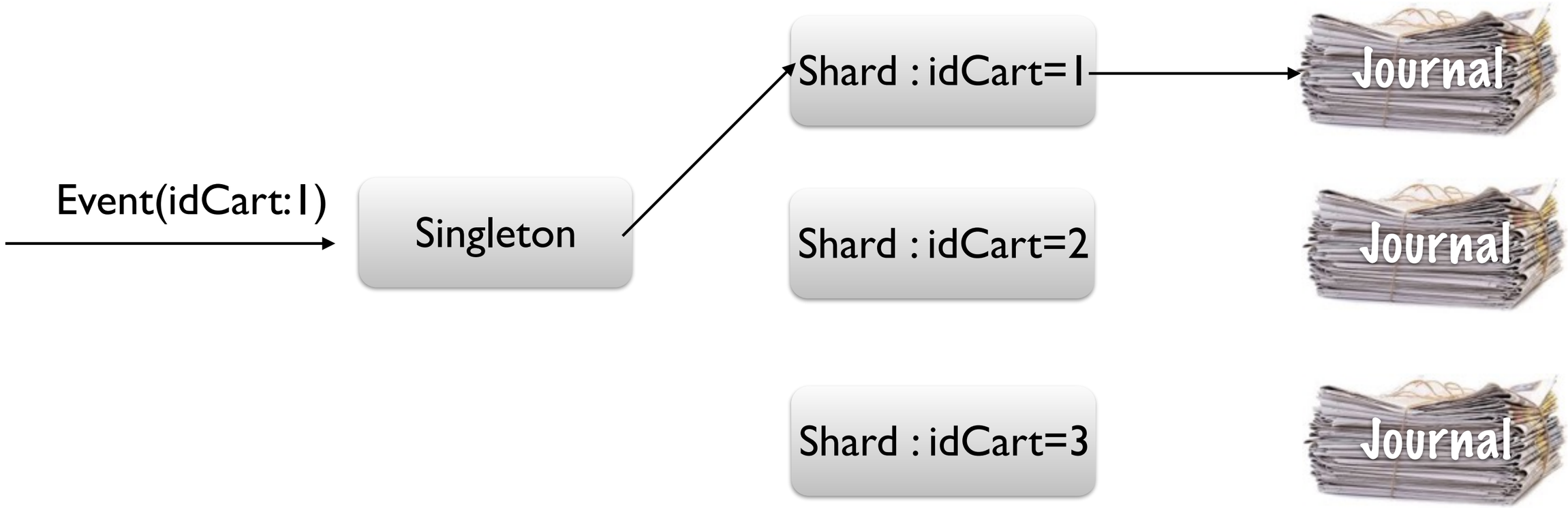
Exemple


AddProductToCart
Products Processor
ProductAddedToCart
Products Views
ProductAddedToCart
WebSocket Actor
ProductAddedToCart
Cassandr
Cassandra

Akka persistence ?

- L'event processor doit être un singleton ! (pas d'écritures concurrentes dans un journal)
- Le journal doit être shardé pour distribuer les écritures
- Akka propose le cluster sharding
  - Un singleton route les écritures par shard (par exemple une partition de journal par user)
  - Pour lire l'état courant, il faut que chaque shard rejoue son journal et stocke l'état en mémoire
  - Possibilité d'utiliser passivate pour « killer » des shard et libérer de la mémoire

Akka persistence



Oui mais ...

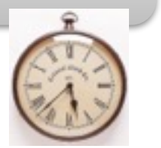
- Singleton == point de contention
- Plein de petits journaux
  - comment faire des stats sur tous les articles commandés sur tous le paniers à partir des journaux
- Un acteur par shard avec son état courant en mémoire
  - conso mémoire
  - passivate ?

Distribuer les écritures ?

Comment conserver
l'ordre des événements ?

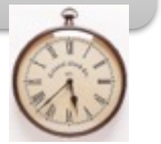
La solution implémentée

CartProcessor



CartProcessor

CartProcessor



Event(20140212114610123, payload)

Event(20140212114711224, payload)

Event(20140212114611156, payload)

Oui mais ...

- Synchronisation de l'horloge sur les serveurs
- La milli-seconde n'est pas assez précise, nano-seconde ?
- Vector clocks
  - Implémentation par Martin Krasser
  - http://krasserm.github.io/2015/01/13/event-sourcing-at-global-scale

Messages

```
object Messages {
  sealed trait Event
  case class ProductAddedToCart(id: String, qte: Int) extends Event

  sealed trait Command
  case class AddProductToCart(id: String, qte: Int) extends Command

  sealed trait Query
  case class ReadCart(id: String) extends Query
}
```

Event source processor

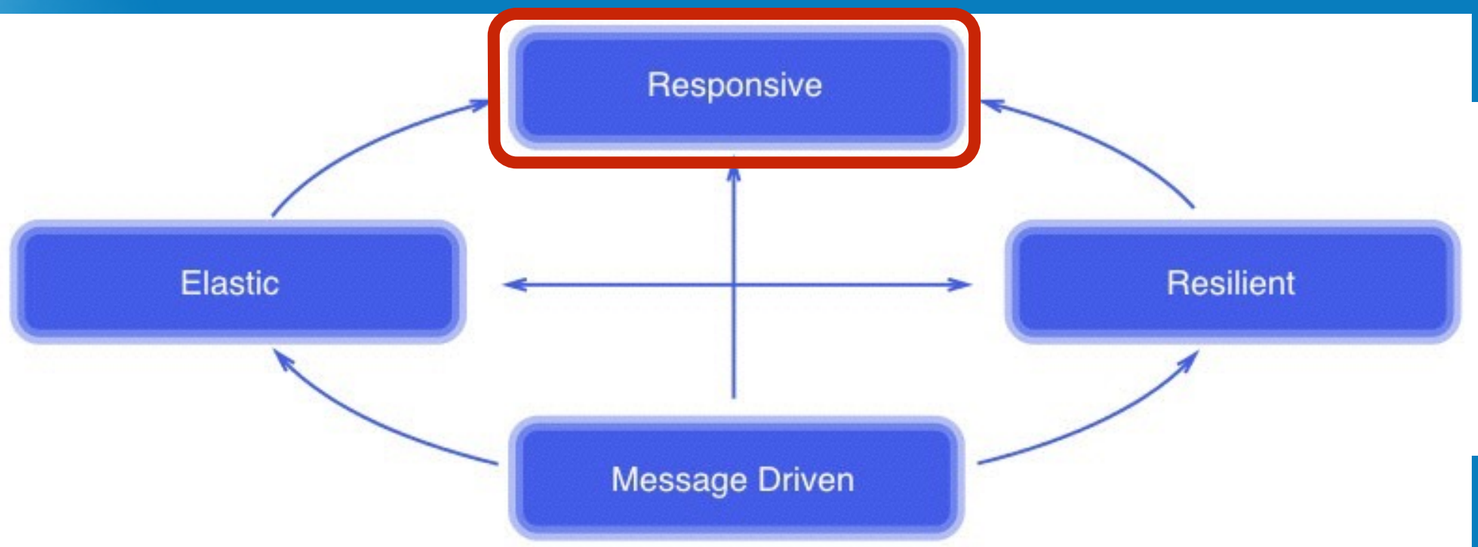
```
class CartProcessor extends EventsourceProcessor {

  override def persistentId: String = "cartProcessor"

  override def receiveCommand: Receive = {
    case Messages.AddProductToCart(id, qte) => //...
      persist(Messages.ProductAddedToCart(id, qte))
  }

  override def views: Views = Views {
    Seq(context.actorOf(CartView.props()))
  }
}
```

View

```
class CartView extends views.View {

  val store = CartStore

  override def receiveEvent: Receive = {
    case e:Messages.Event => store(e)
  }

  override def receiveRecover: Receive = {
    case e:Messages.Event => store(e)
  }

  def store(event: Messages.Event) = {
    event match {
      case Messages.ProductAddedToCart(id, qte) => store.addToCart(id, qte)
    }
  }

  override def receiveQuery: Receive = {
    case Messages.ReadCart(id) => store.read(id) pipeTo sender()
  }
}
```

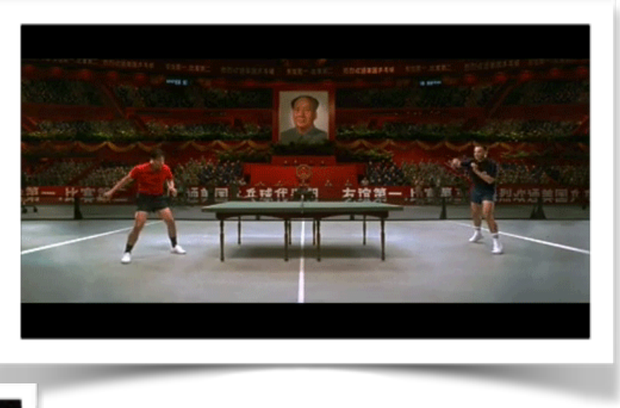
SERLi

Responsive

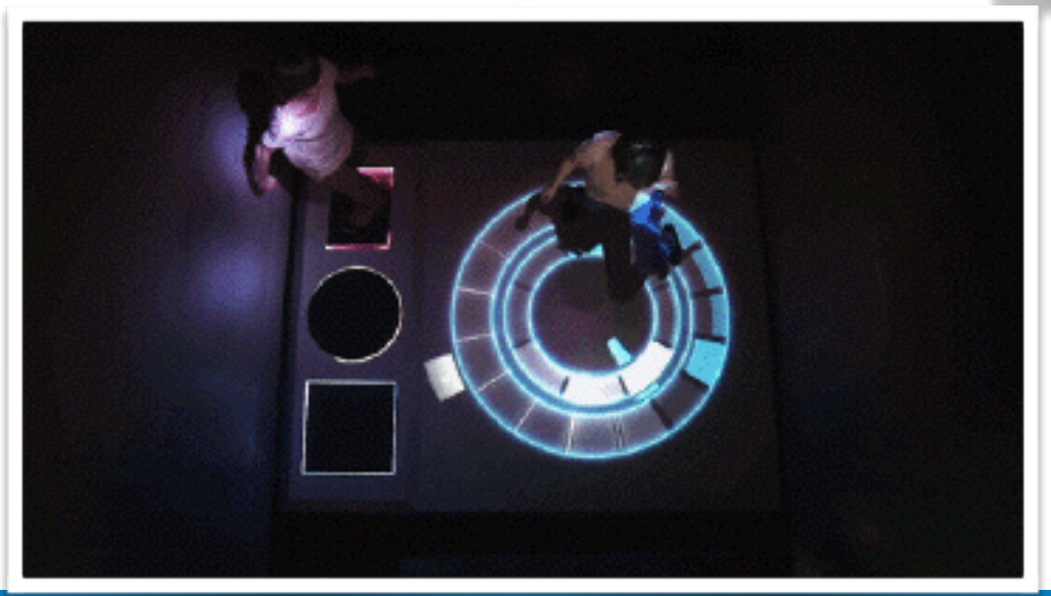


Frontend réactif

+



=



SERLI

Demo

Realtime web!

Et les perfs dans tout ca ?

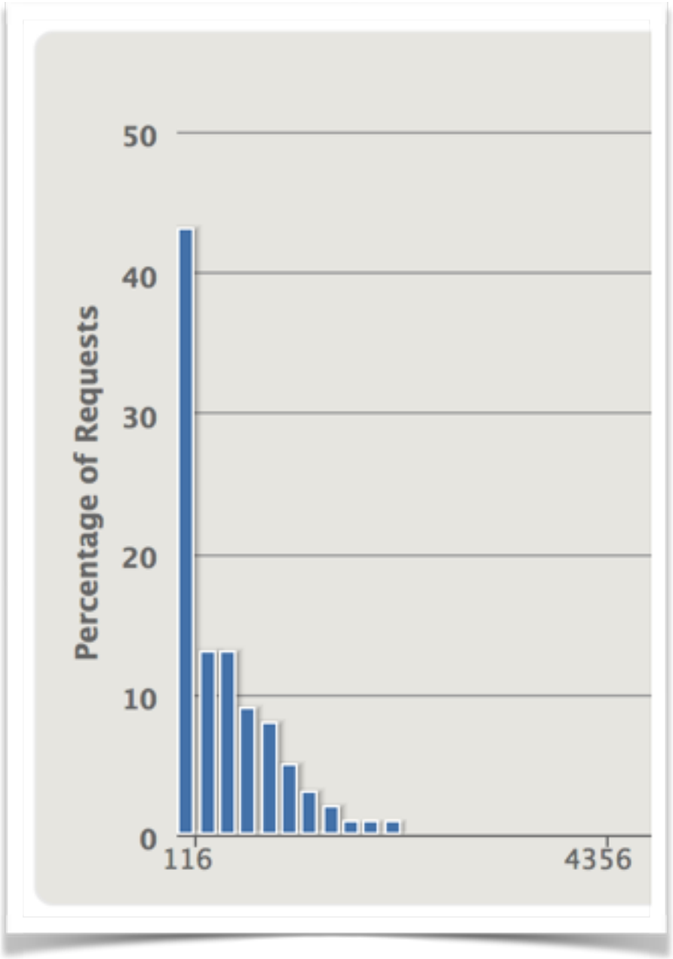
Les chiffres

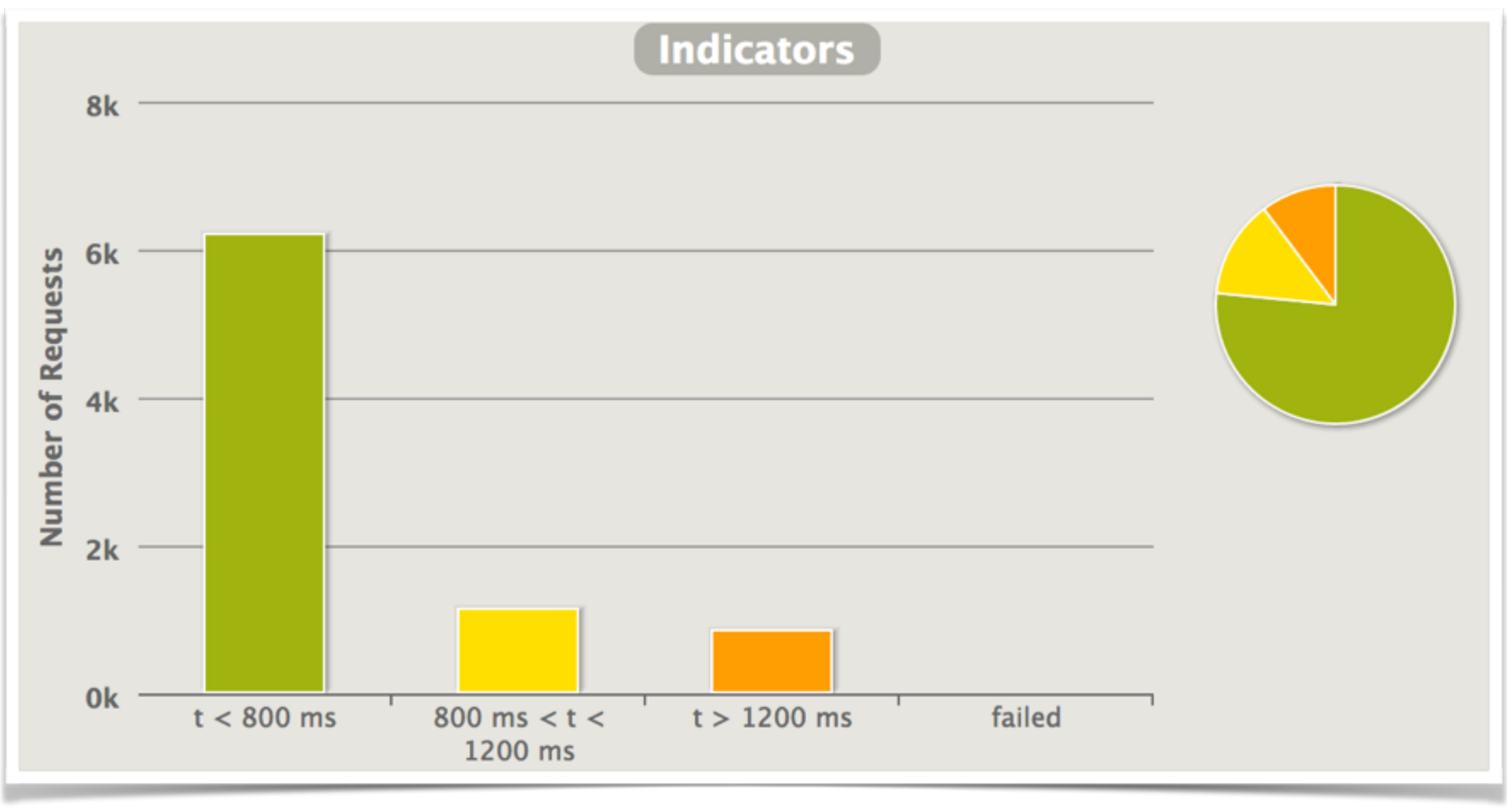
367 998 456 756

Les chiffres

367 998 456 756 €

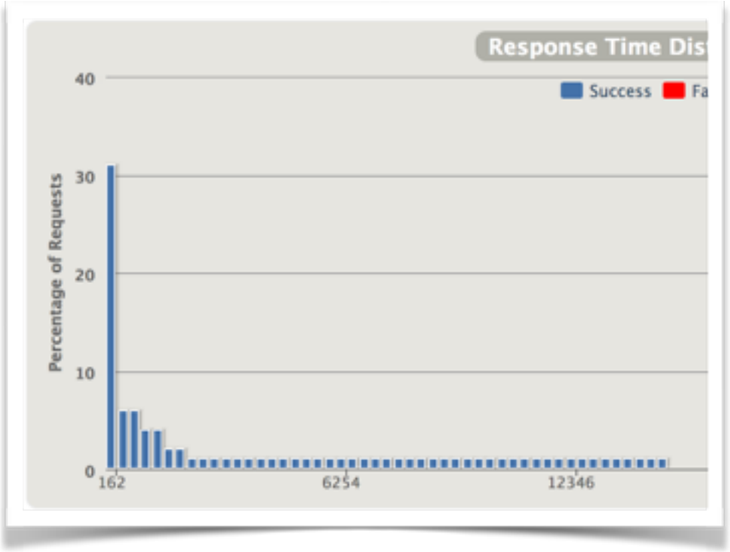
200 clients

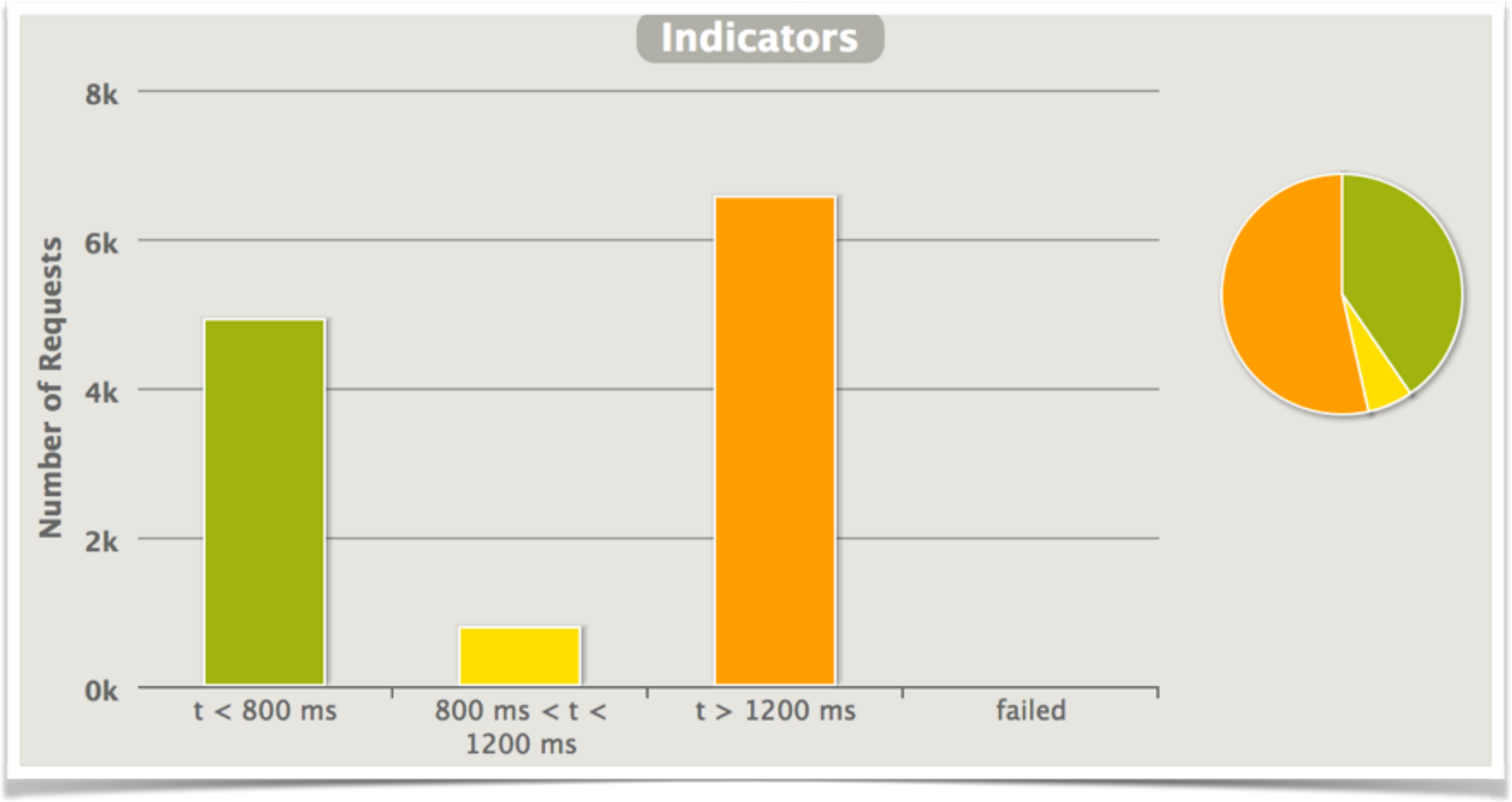




SERLi

400 clients





SERLi

Metrics everywhere !!!

Cassandra

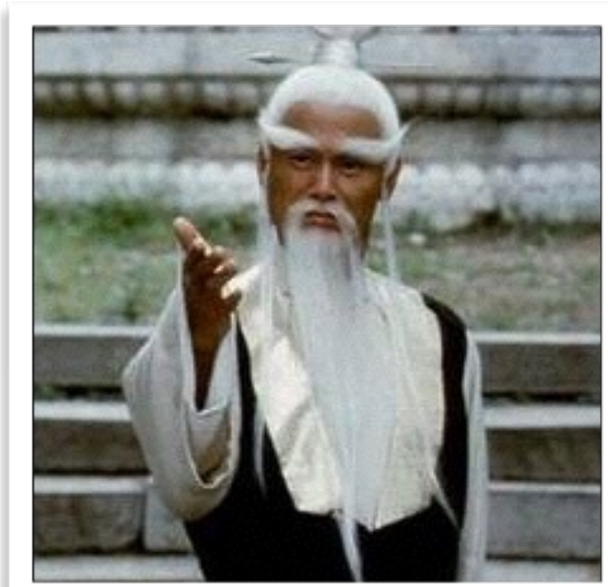
Messages akka

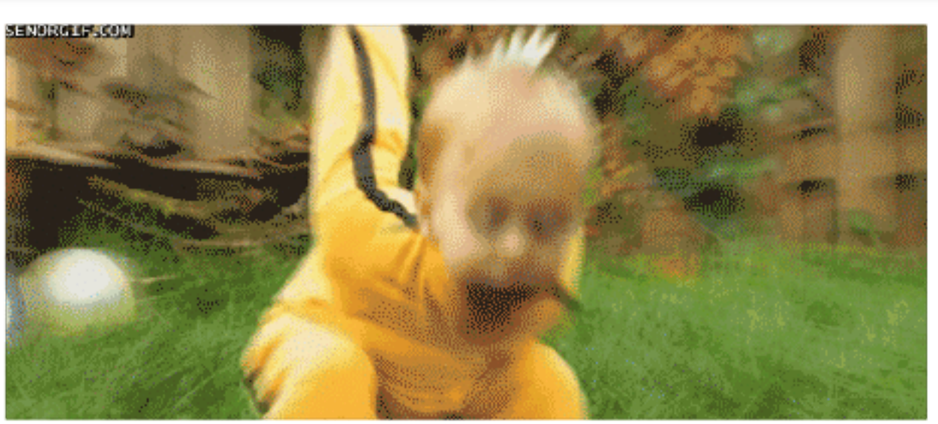
Journal

Elasticsearch

Les problèmes rencontrés

Nginx / mongo ...



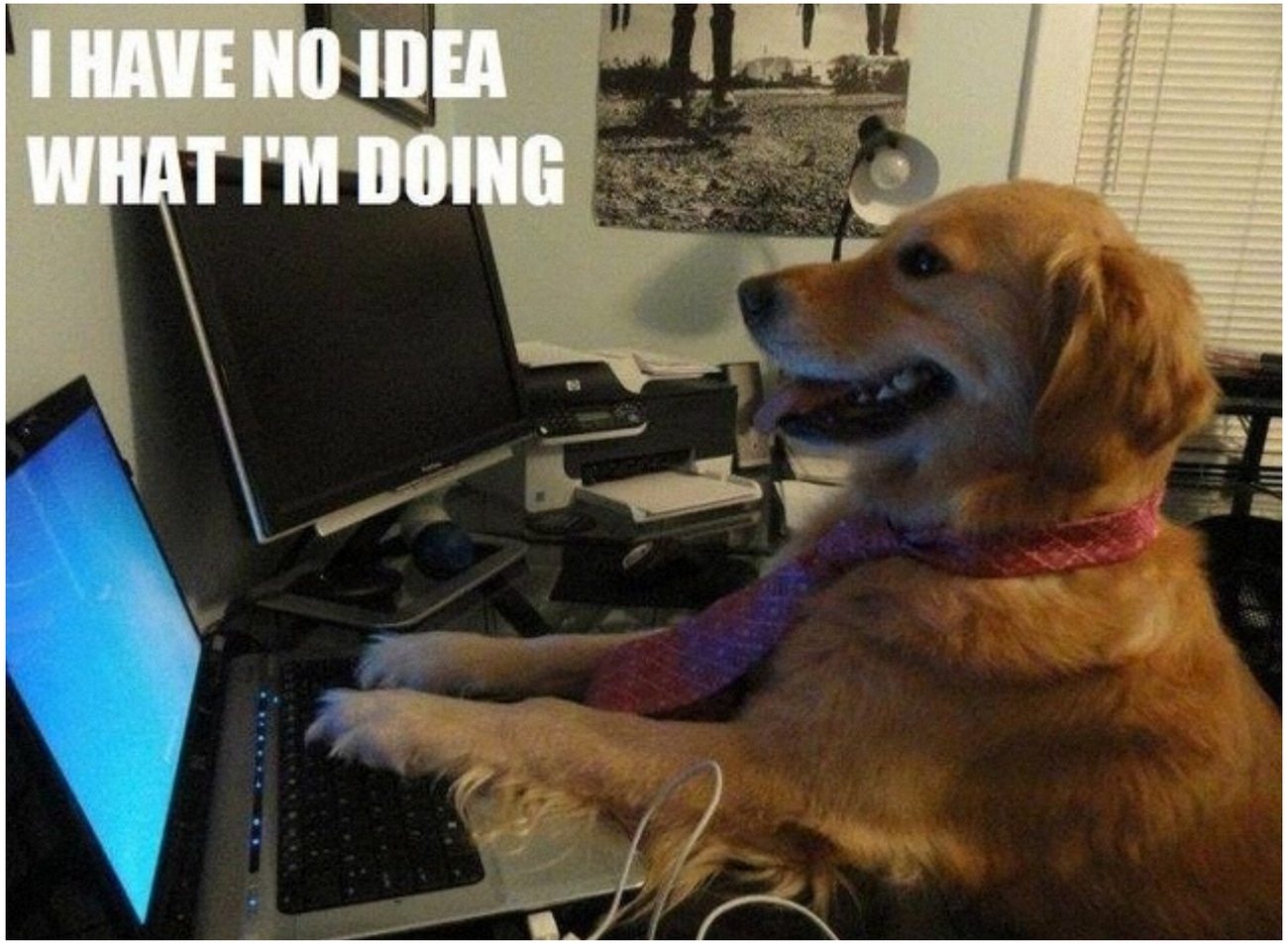


Cassandra

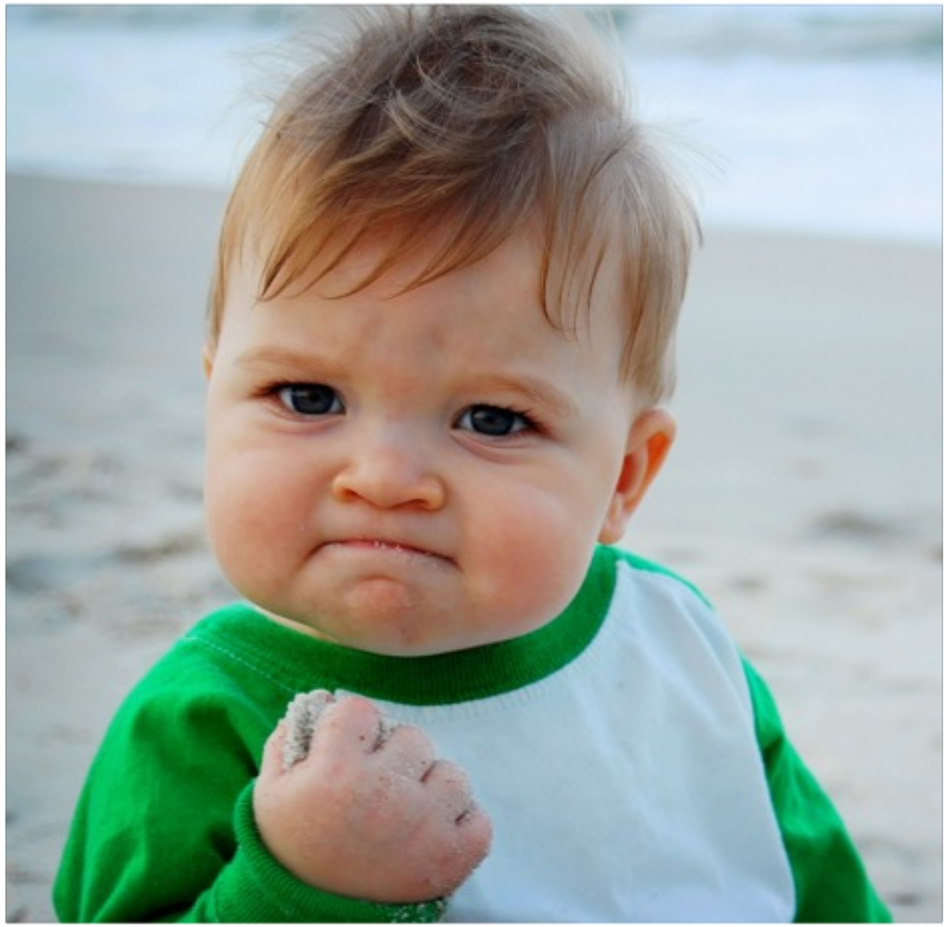
Bench web socket

SERLI

Configuration akka-cluster



Conclusion



scala

akka

Scala

play

SERLI

This is the end ...