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

Société de conseil et d'ingénierie

Développement, expertise, R&D, formation

70 personnes

Contribution à des projets OSS

Membre du JCP

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

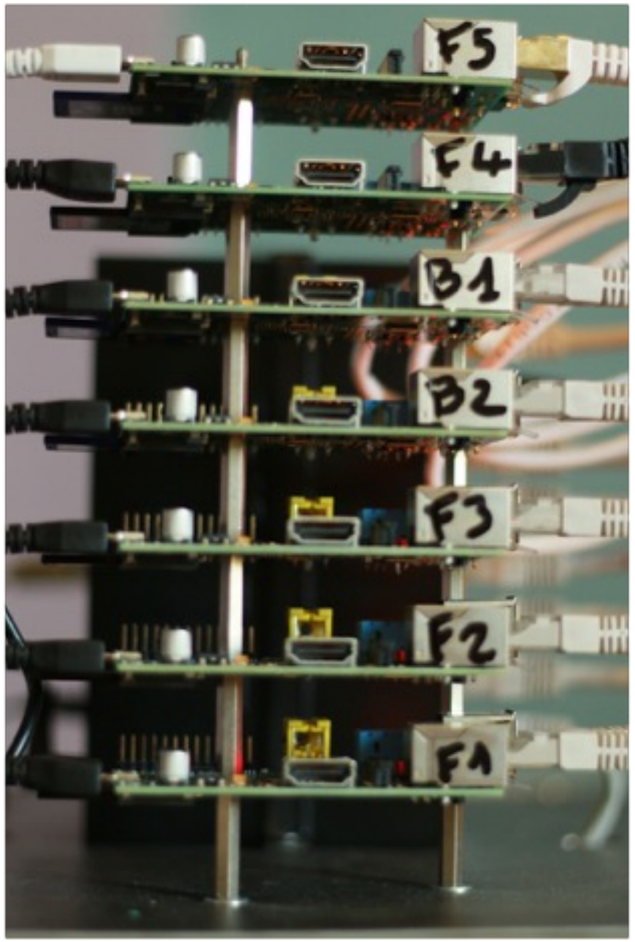
Magnifique arbalète pouli ...

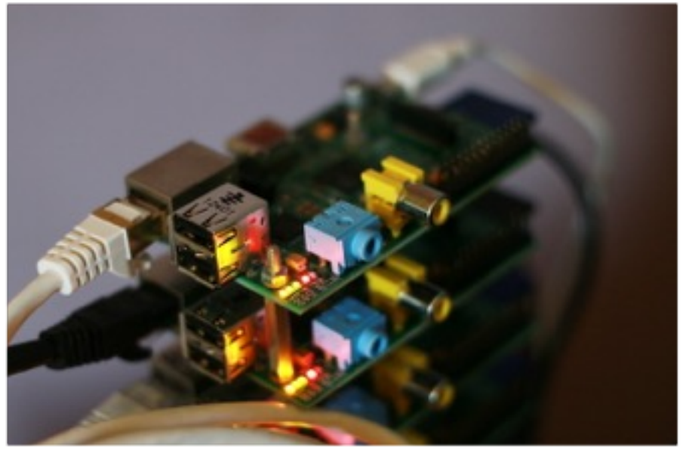
200.99 €

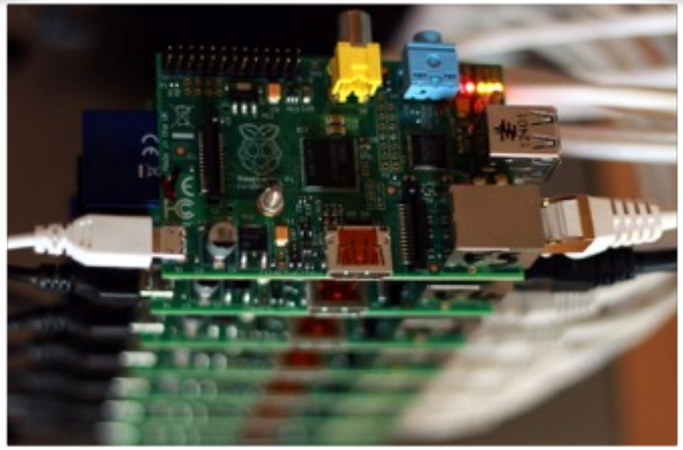
Add to cart

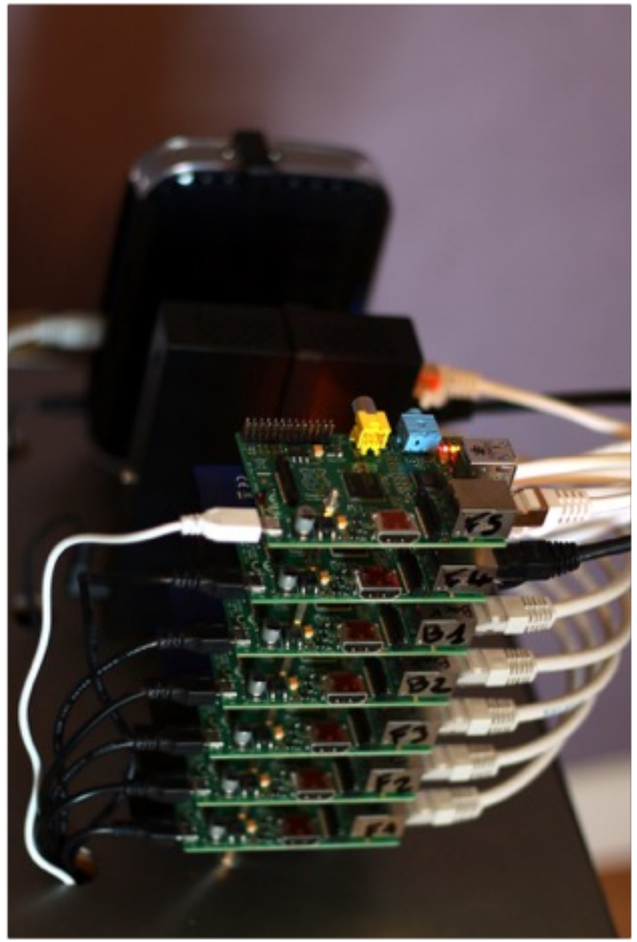
SERLI

Mobile infrastructure





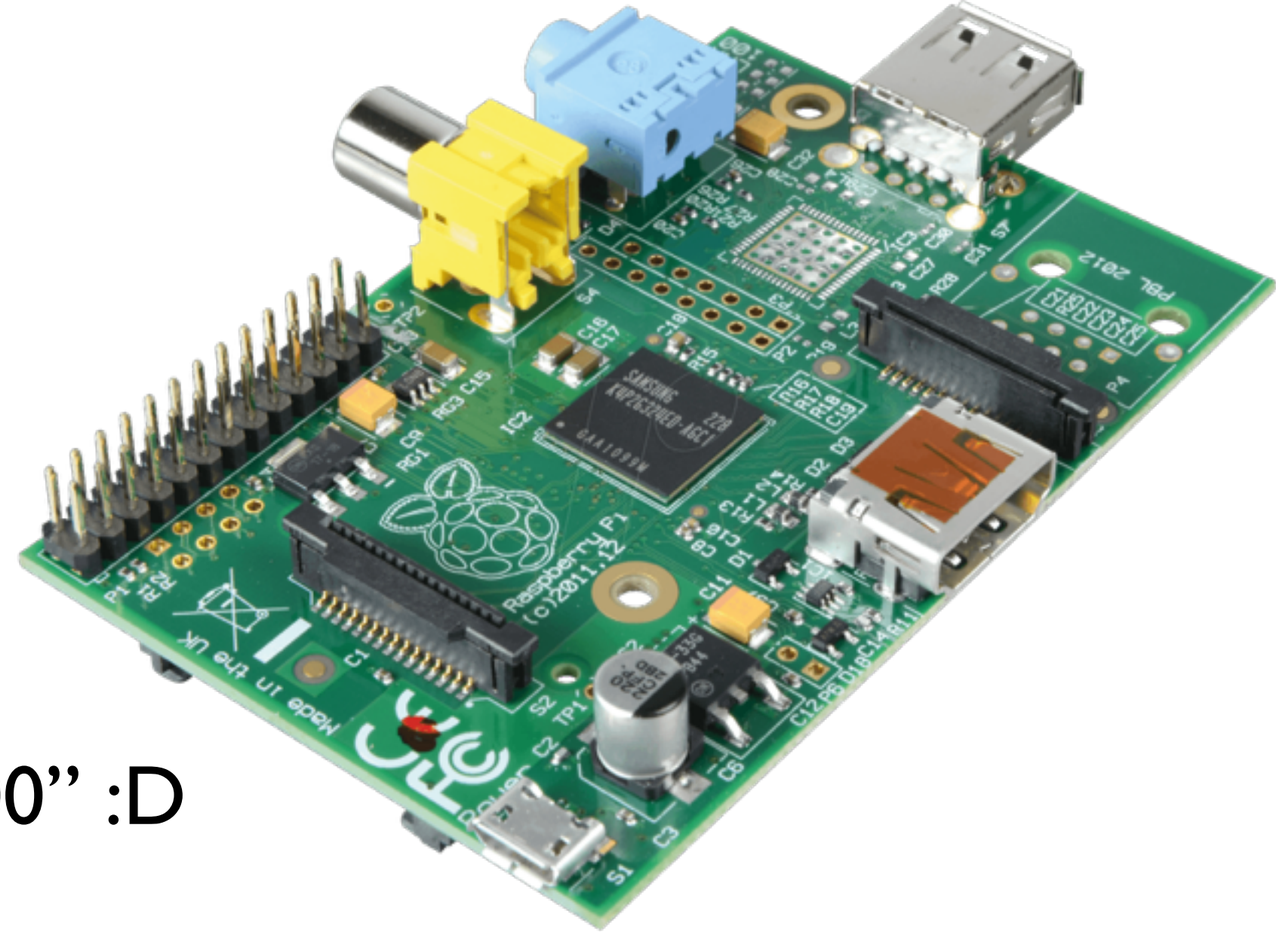




SERLI

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

SERLI

Absorber une hausse de trafic

SERLI

Approche classique

Monolithique

SERLI

Latence

Bloquant

Charge

SERLI

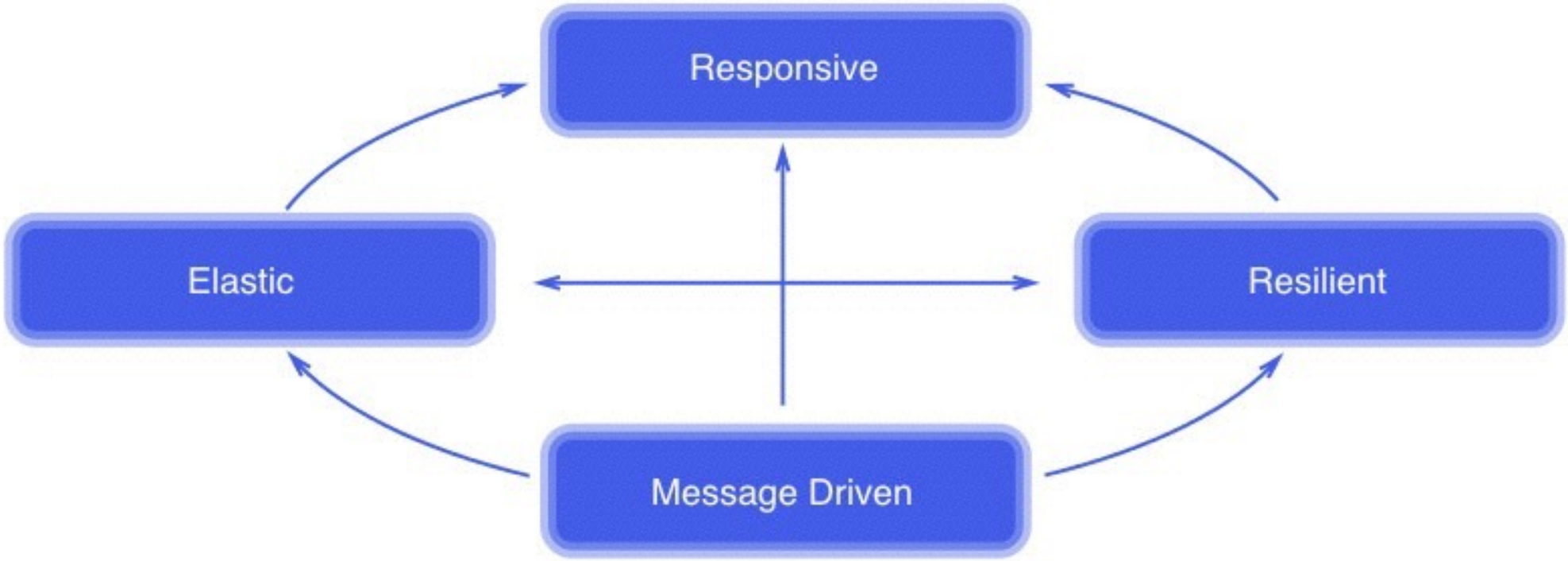
Scalabilité

Comment faire ?

SERLI

Reactive Manifesto

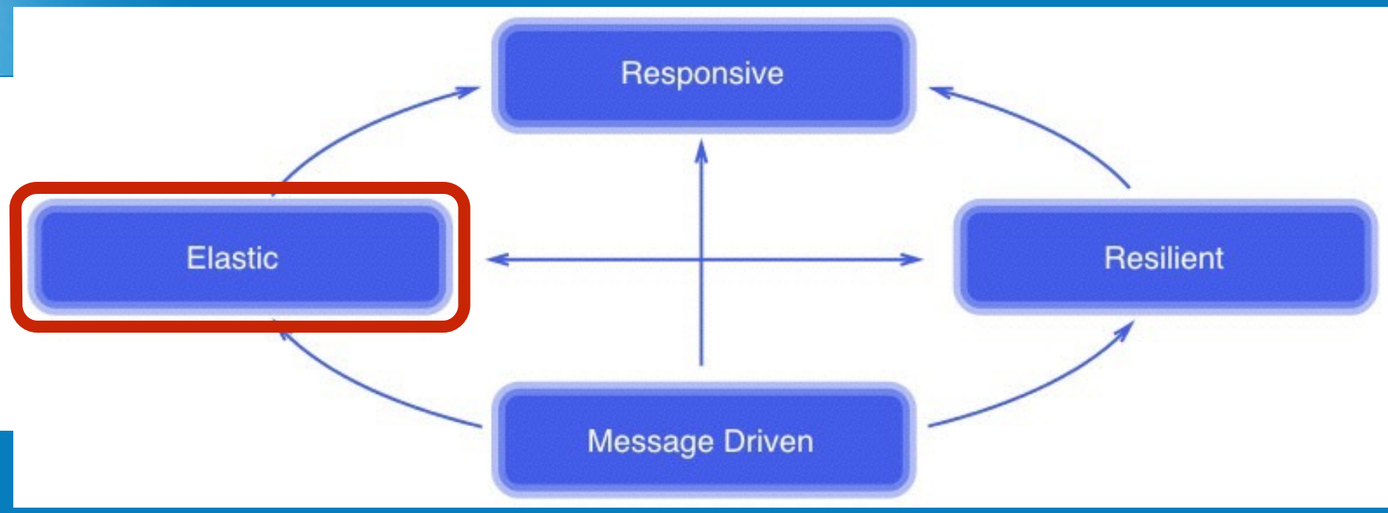
Reactive Manifesto



Reactive manifesto

- **responsive :** le système répond dans un temps acceptable et linéaire
- **resilient :** le système répond même en cas de panne.
- **elastic :** le sytème réponds malgré une charge variable. On doit pouvoir augmenter ou en diminuer les ressources allouées aux services.
- **message driven :** le système repose sur des envois de messages asynchrones. Ceci assure un couplage lâche entre les composants de l'application

Scalable / React to load



Overview

Frontend

Cart

Identity

elasticsearch.

Cassandra

Cassandra

Cassandra

Backend

SERLI

Datas

elasticsearch.

Cassandra

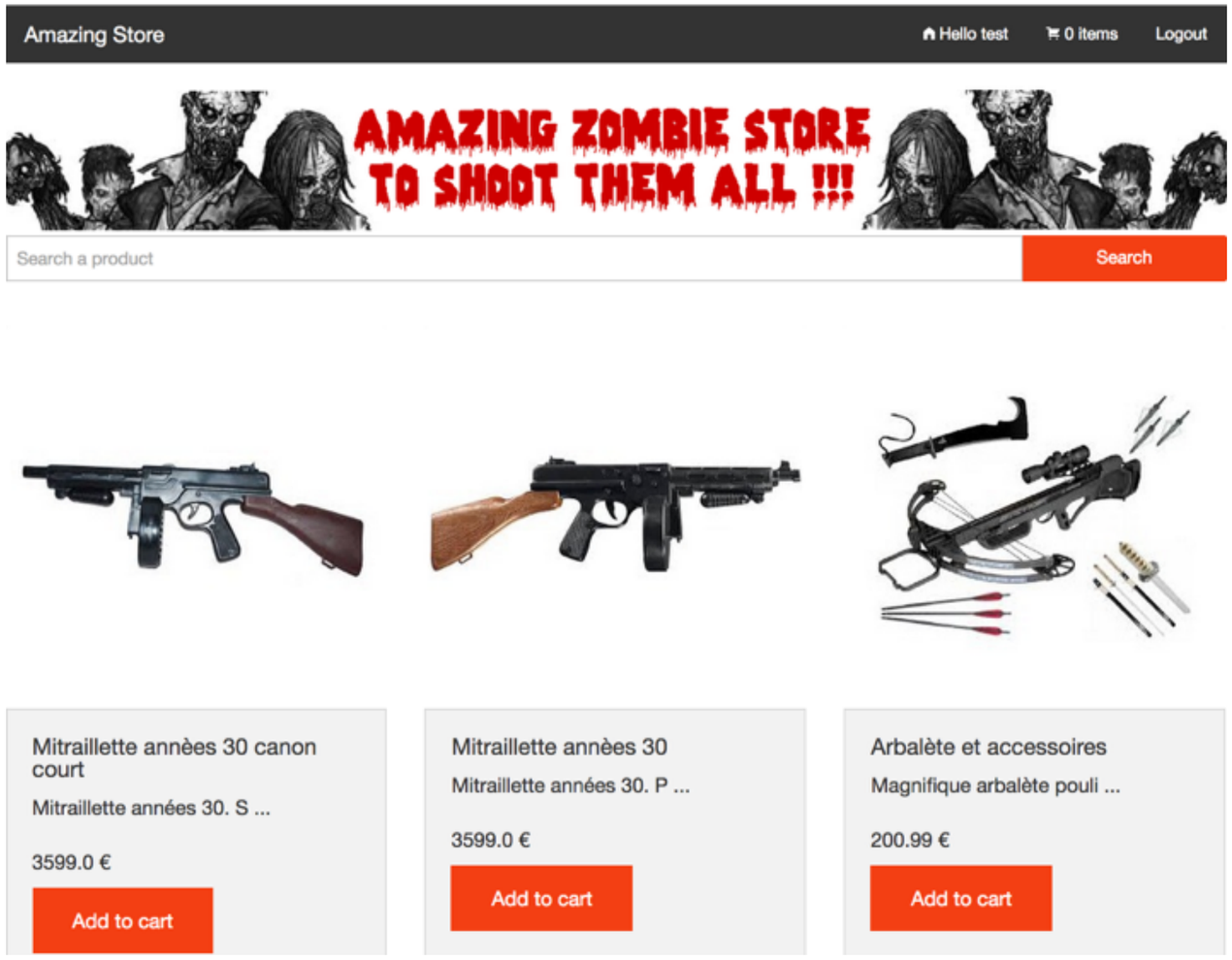
Mini / Micro Services

- 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

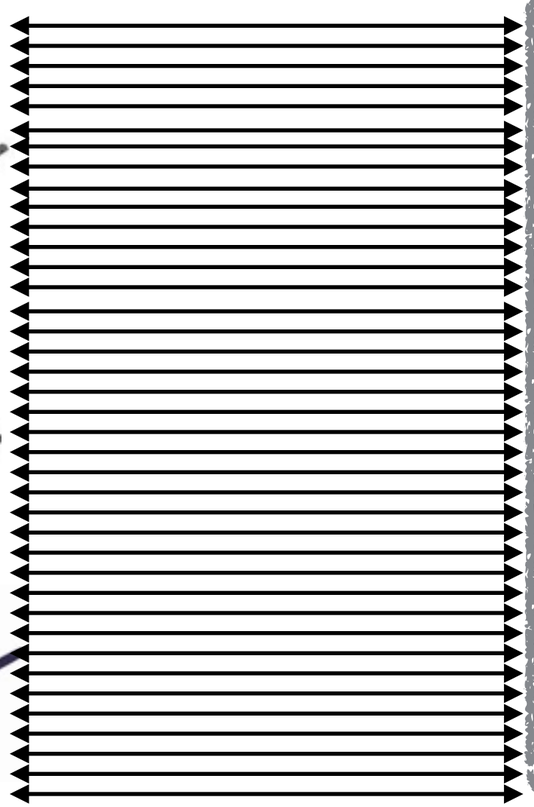


- 100 % html
- Indexation par les moteurs de recherche
- stateless
- une url == un contenu

SERLI

Limiter la charge

Cache



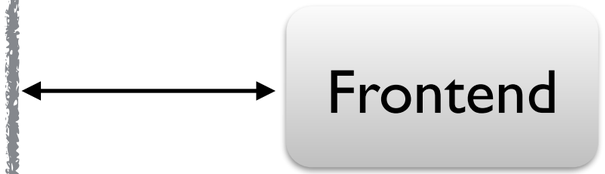
Cache

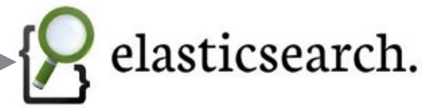
Browser

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

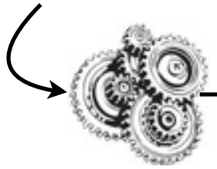
Créer / mettre à jour

UpdateProduct

Backend

ProductUpdated

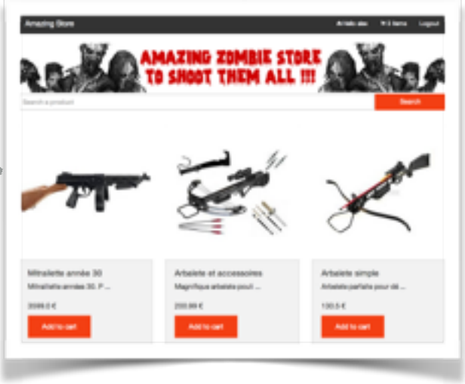
Frontend



HTML

EventSource

HTML



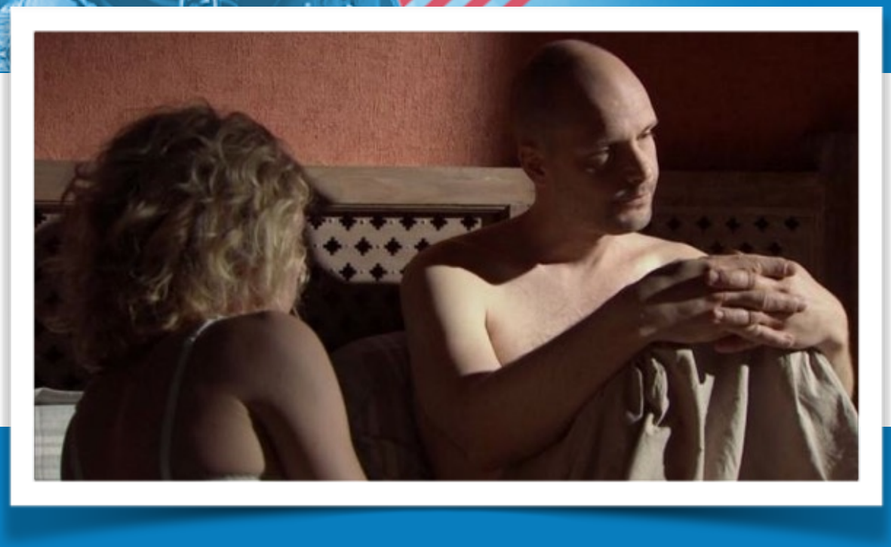
ProductUpdated

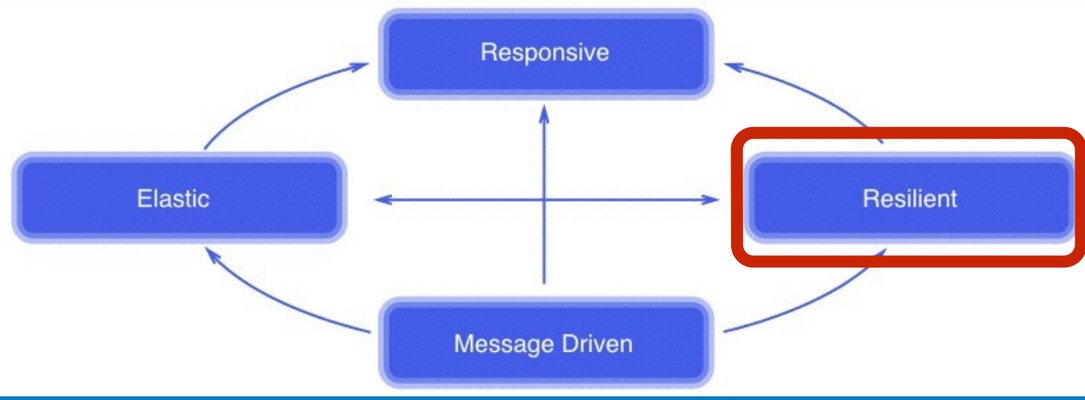
Cassandra

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

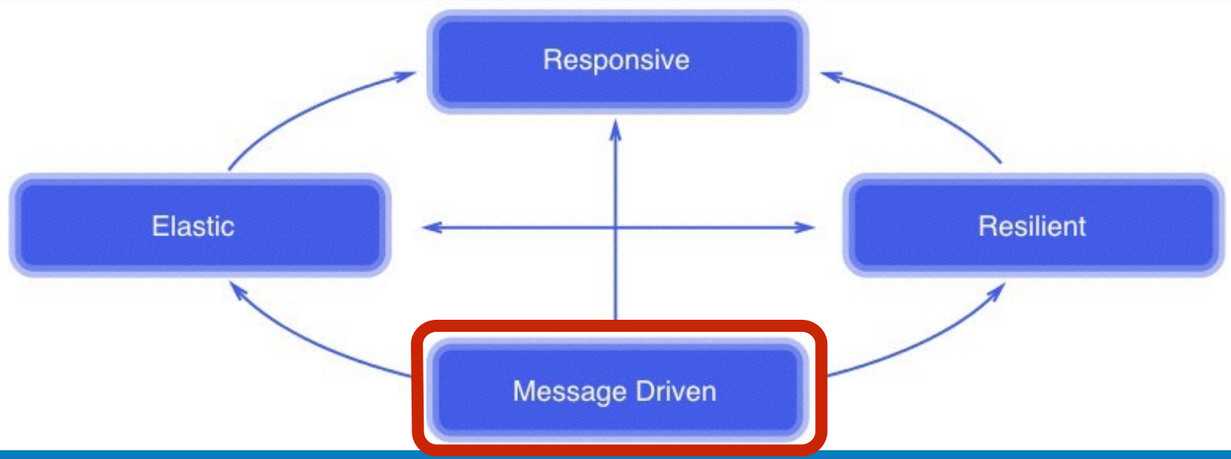
elasticsearch.

Demo

Let It Crash !!!

SERLi

Event-driven / react to event



Akka ?

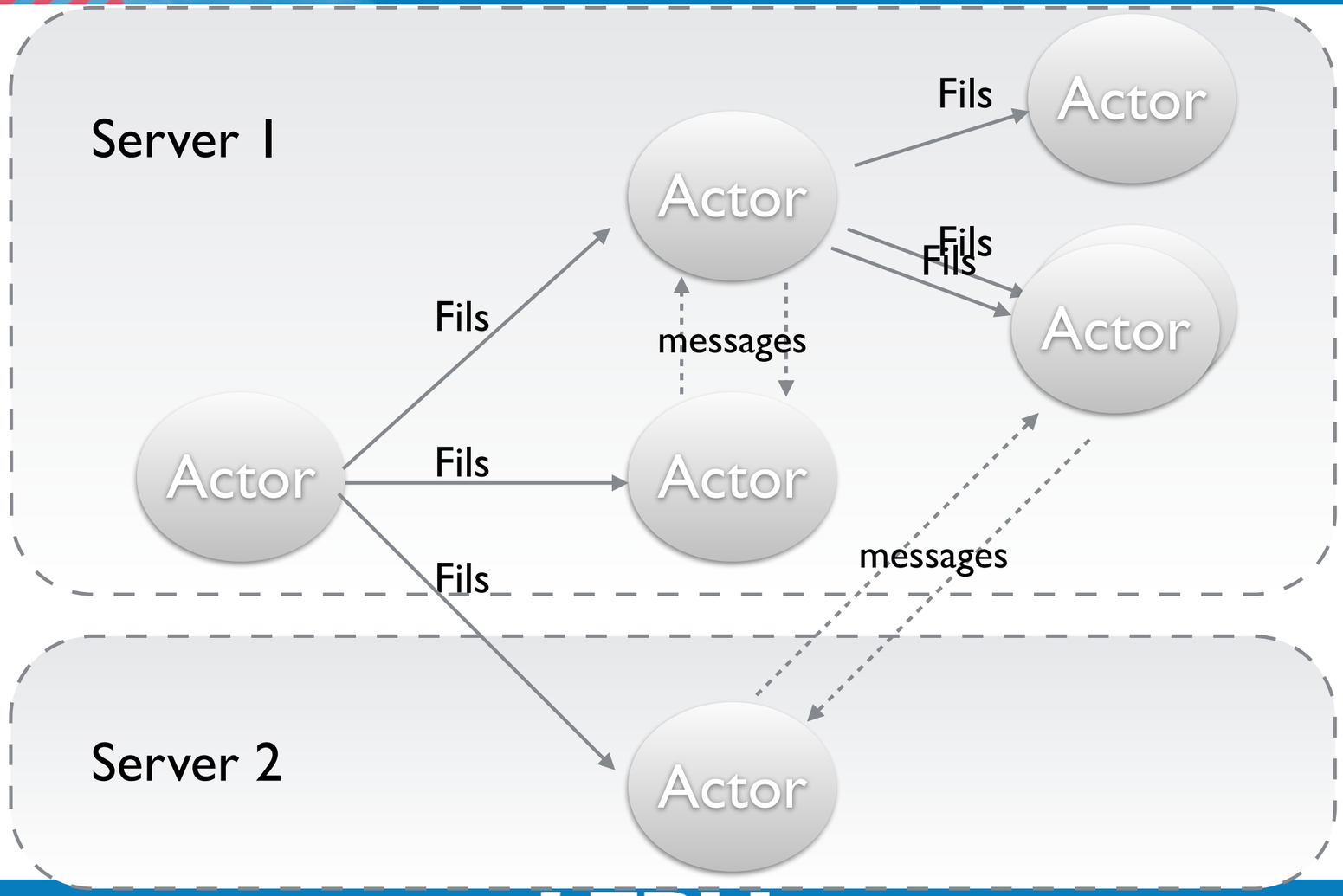
Akka ??

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 fullstack :
  - json
  - validation
  - templating
  - …
- java or scala
- Support pour les websocket et le server sent event
- Asynchrone
  - pré-requis pour une application orientée événements

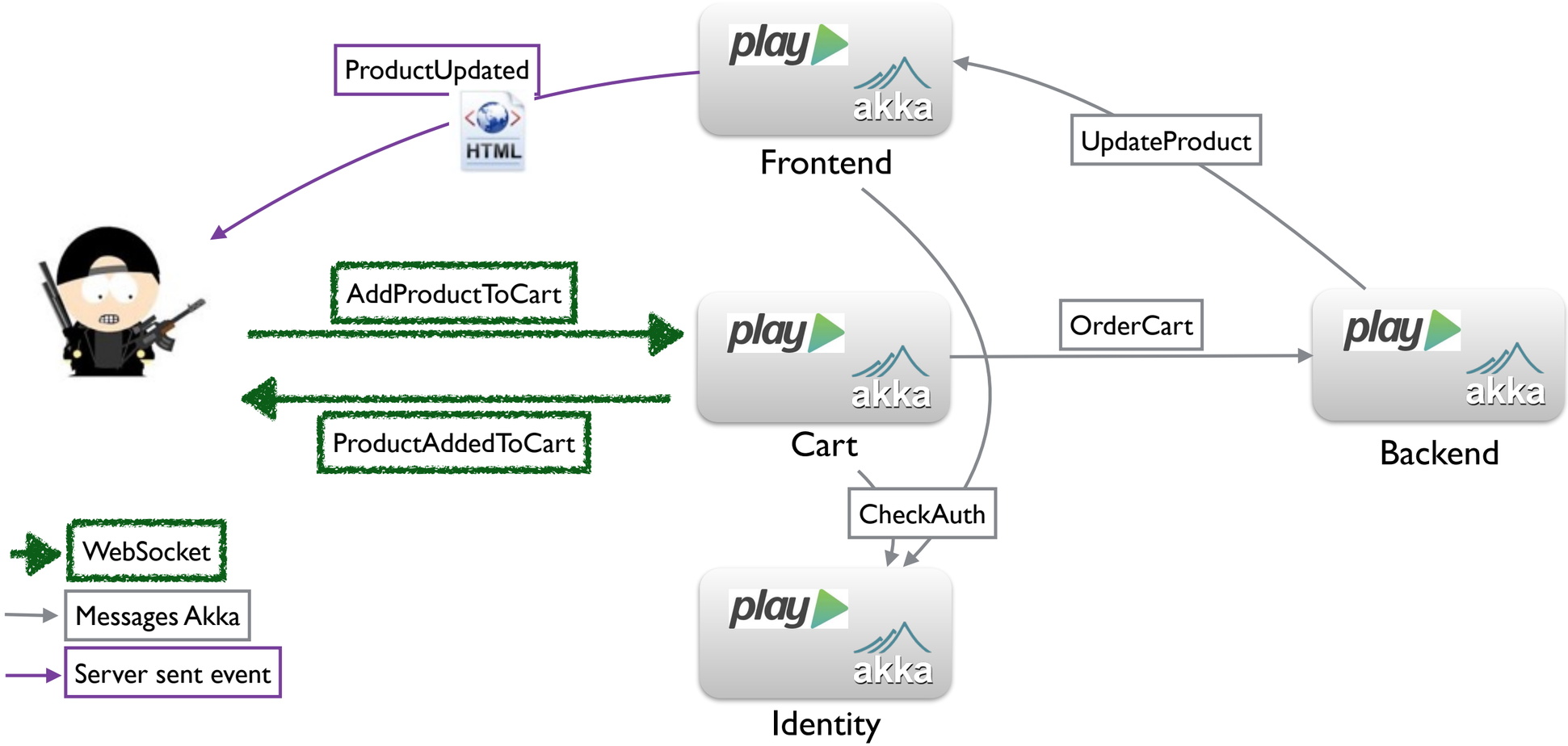
Play async

```
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)))
  }
}
```

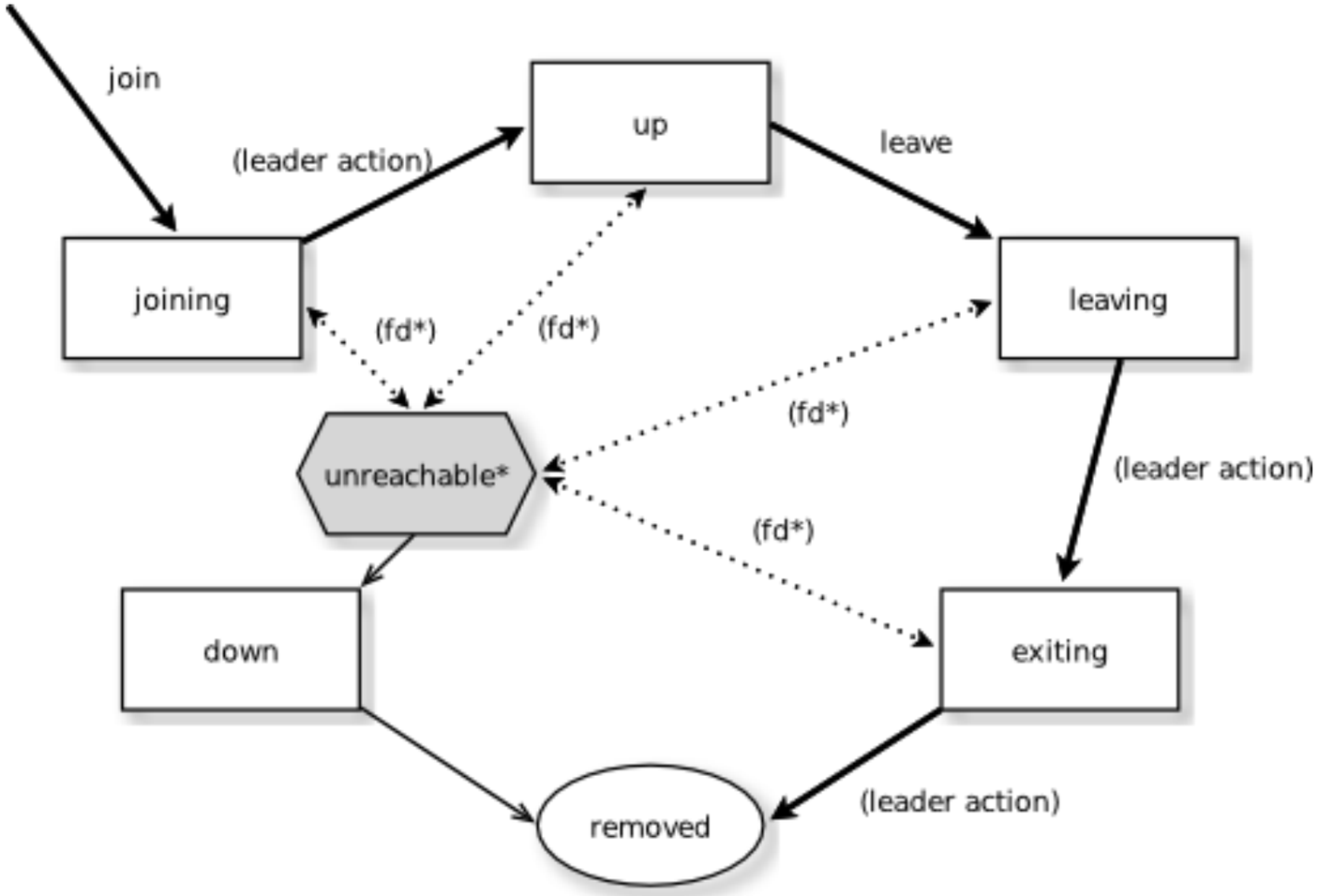
Messages


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



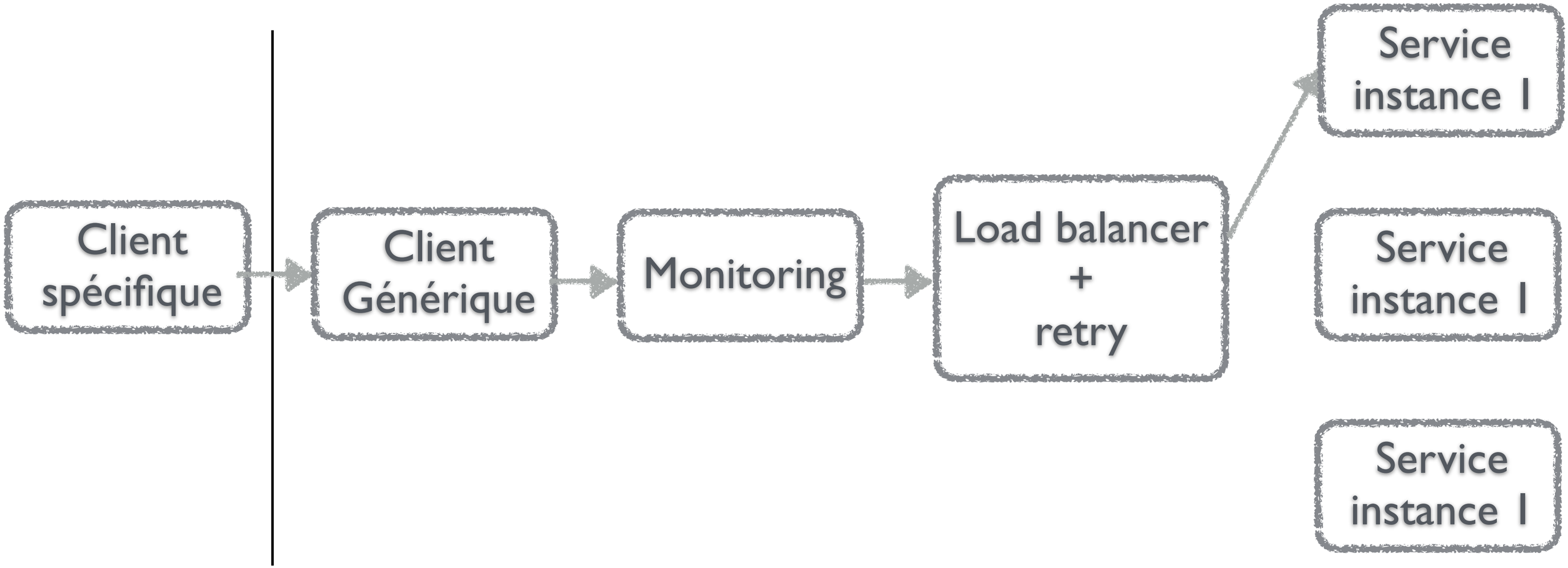
Cluster services

- 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

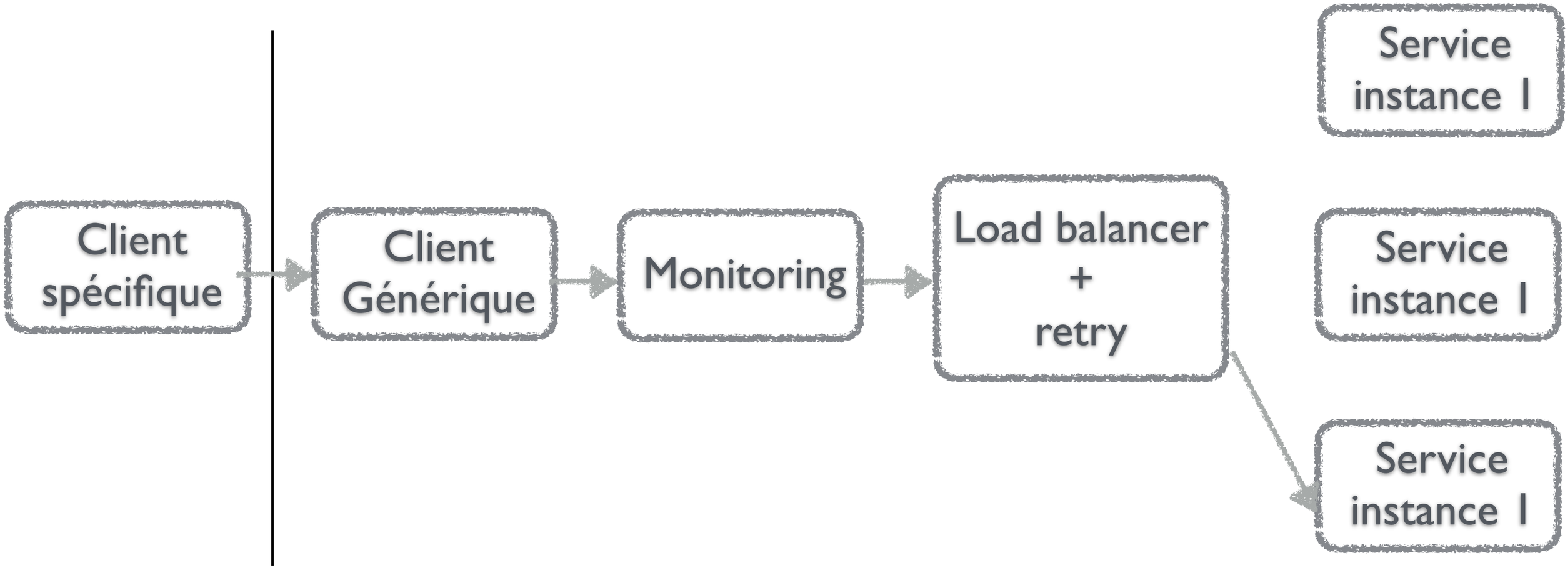
finagle

+ Eureka

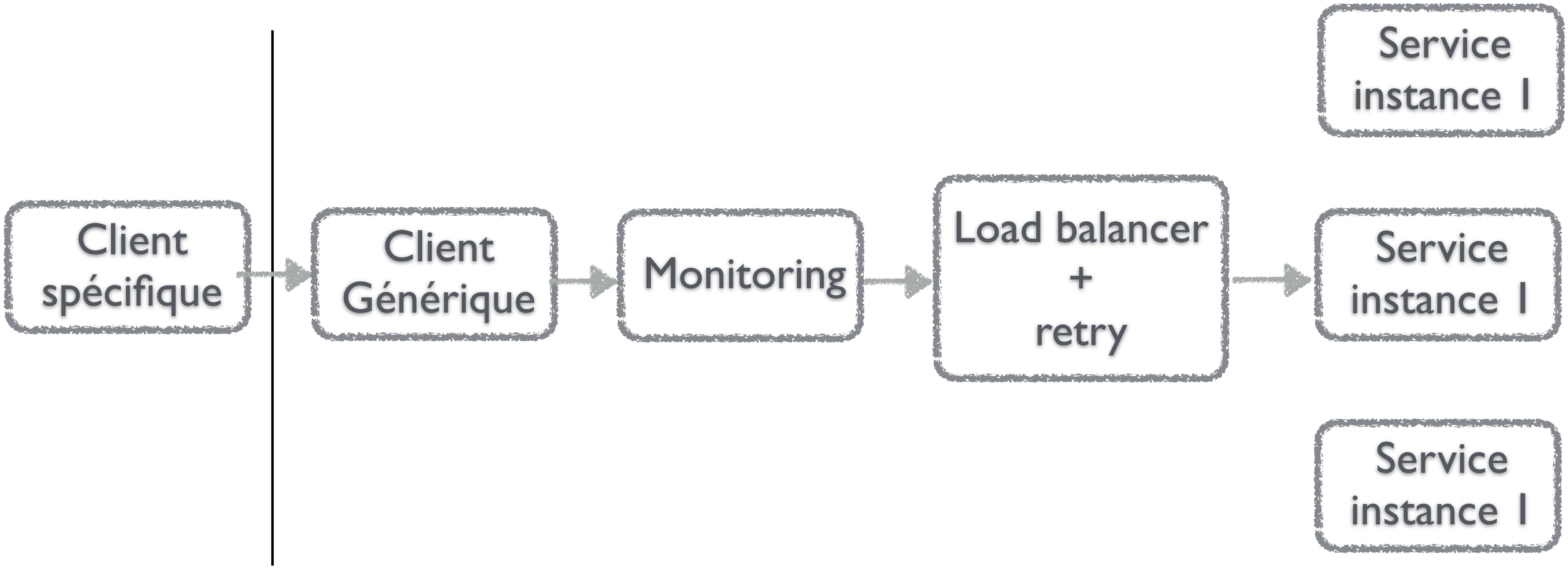
Cluster services



Cluster services

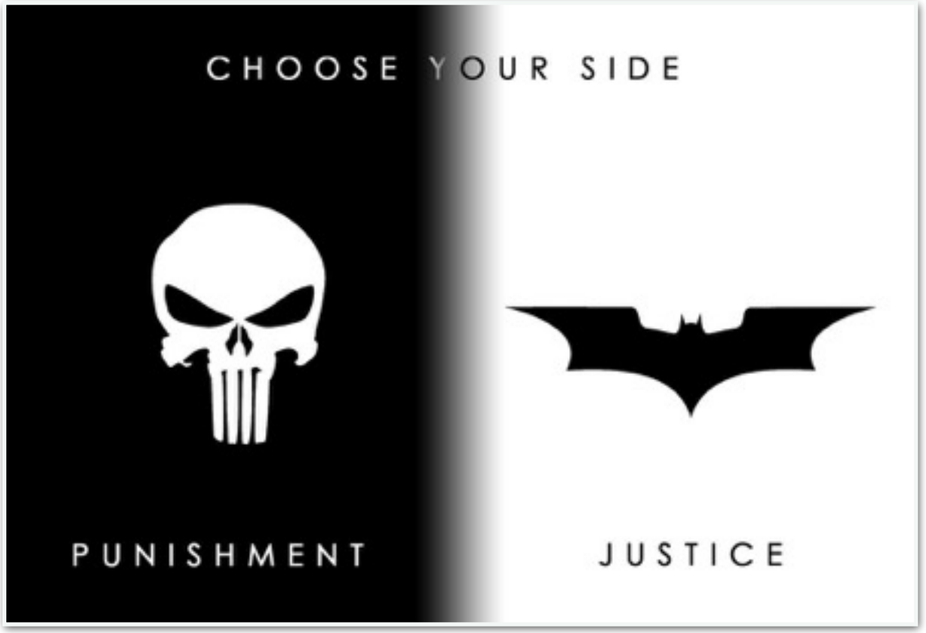


Cluster services

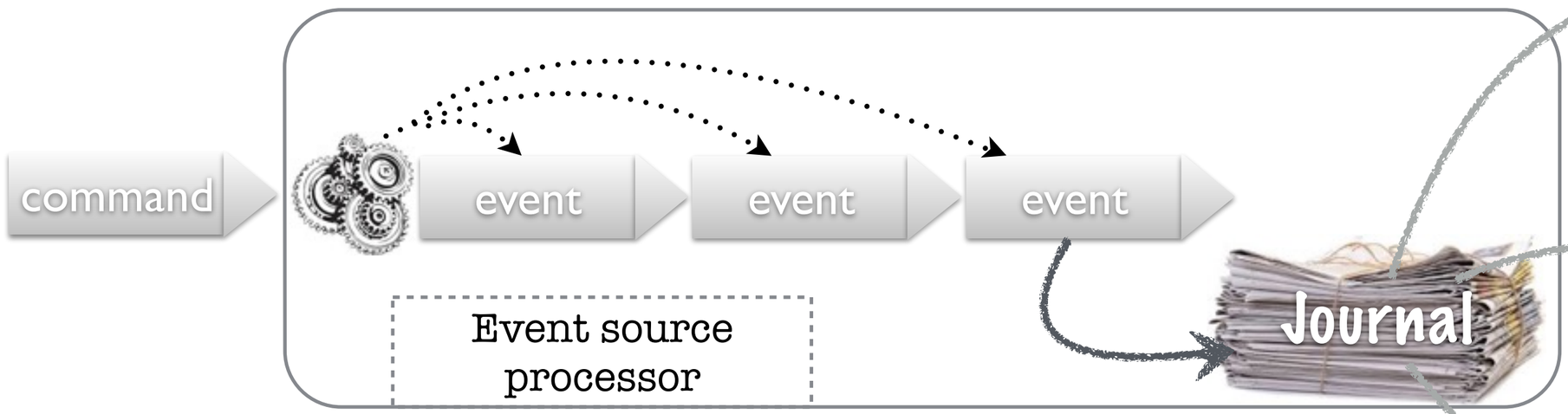


CQRS & EventSourcing

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

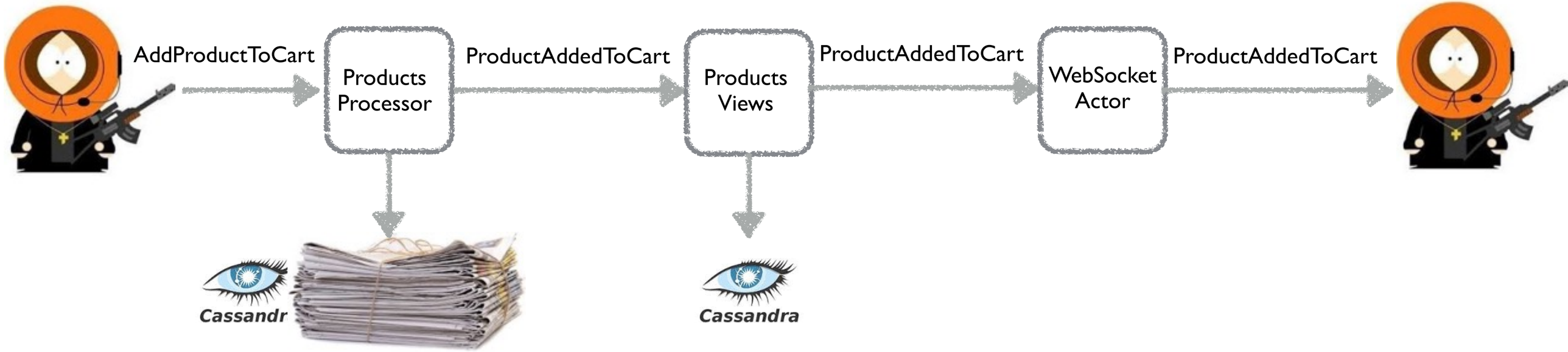


Persistence



SERLi

Exemple

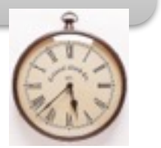

AddProductToCart
Products Processor
ProductAddedToCart
Products Views
ProductAddedToCart
WebSocket Actor
ProductAddedToCart
Cassandr
Cassandra

Distribuer les écritures ?

Comment conserver
l'ordre des événements ?

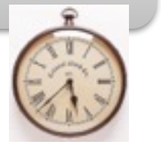
La solution implémentée

CartProcessor



CartProcessor

CartProcessor



Event(20140212114610123, payload)

Event(20140212114711224, payload)

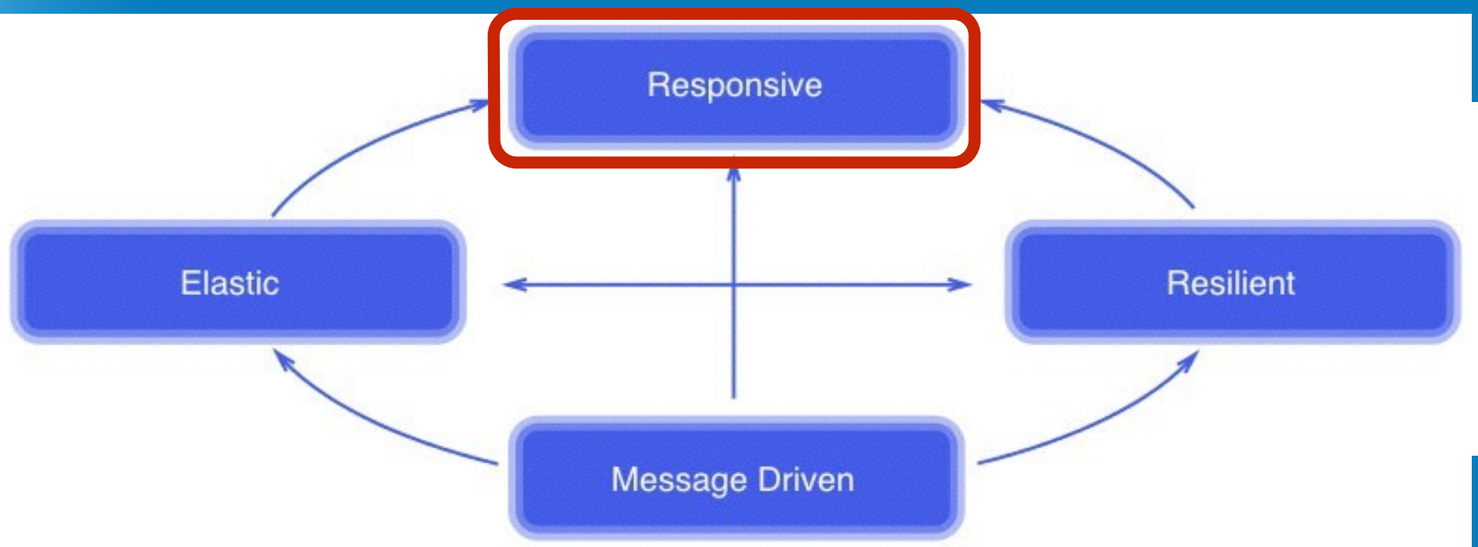
Event(20140212114611156, payload)

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

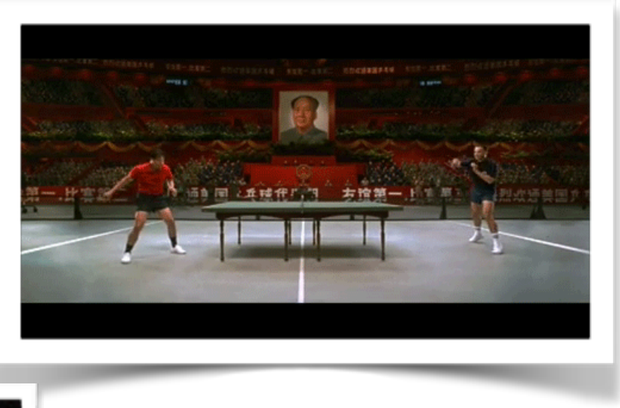
SERLi

Responsive

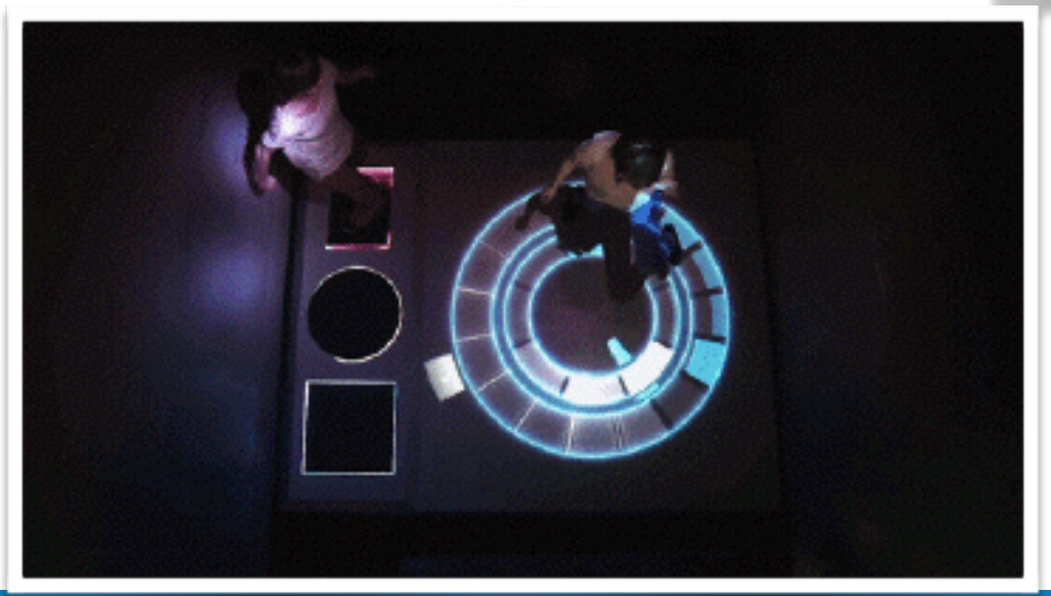


Frontend réactif

+



=



SERLI

Et les perfs dans tout ca ?

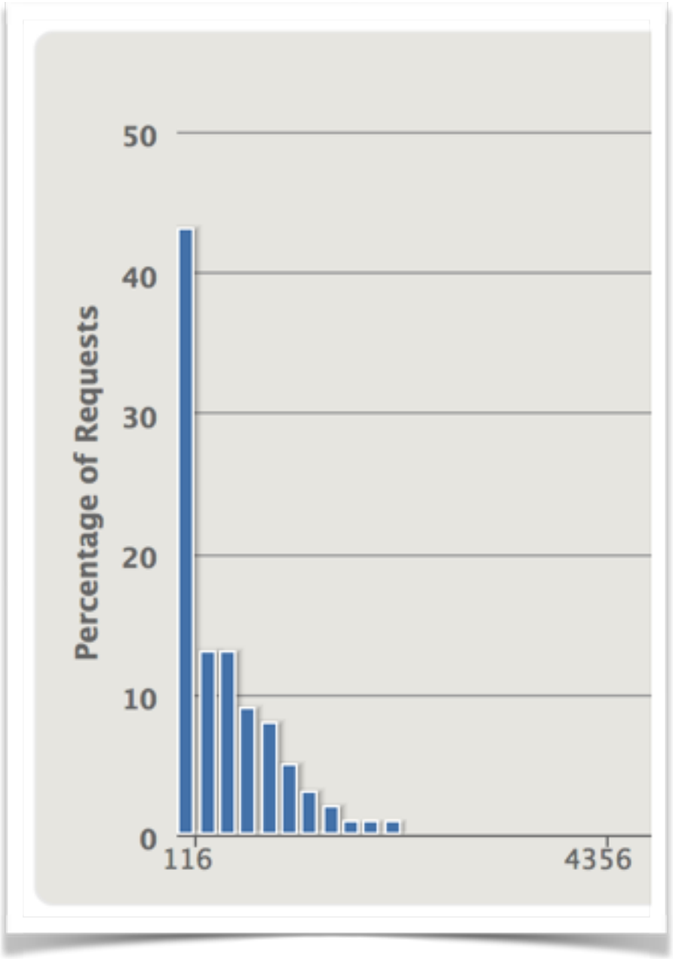
Les chiffres

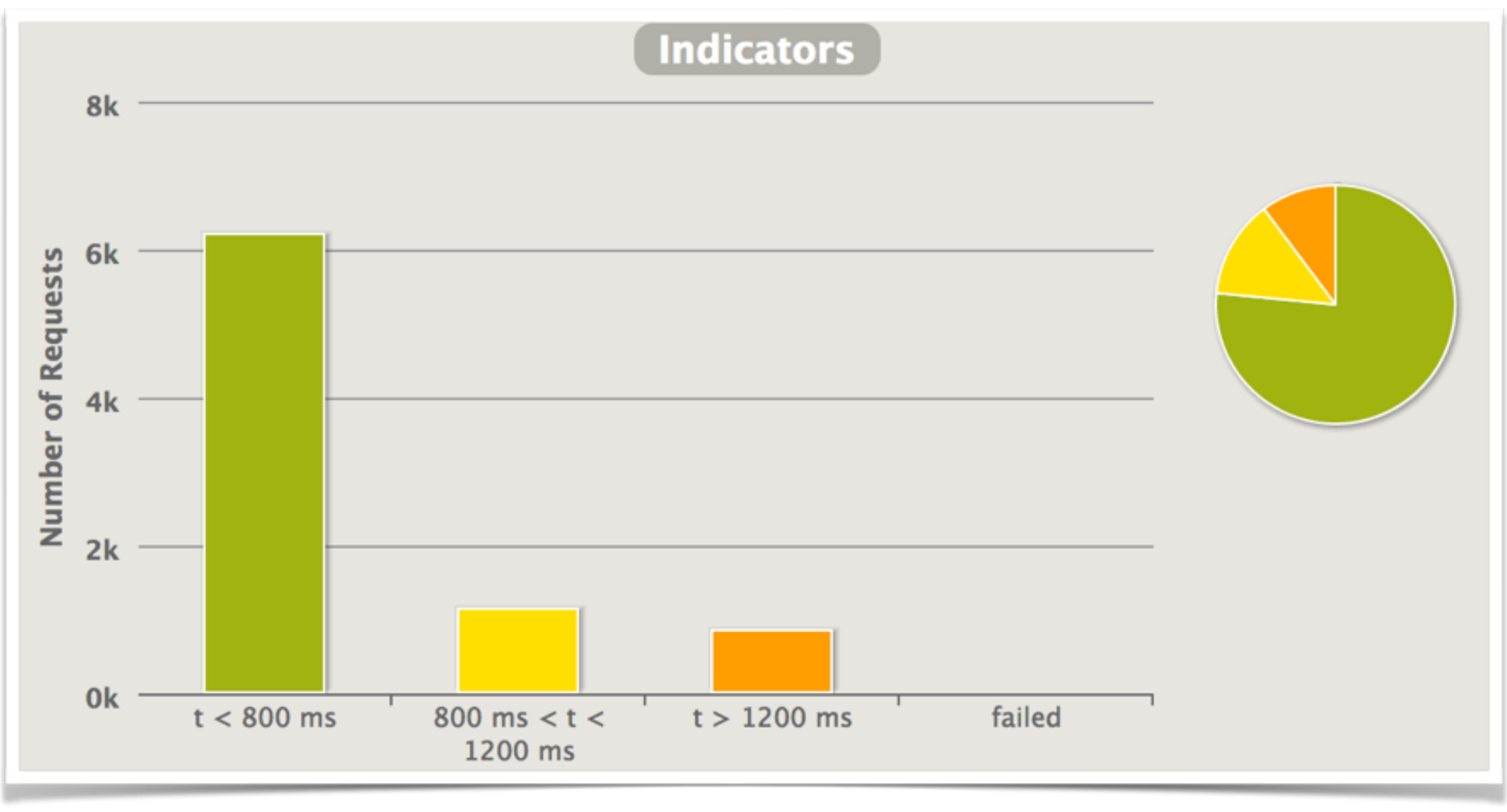
367 998 456 756

Les chiffres

367 998 456 756 €

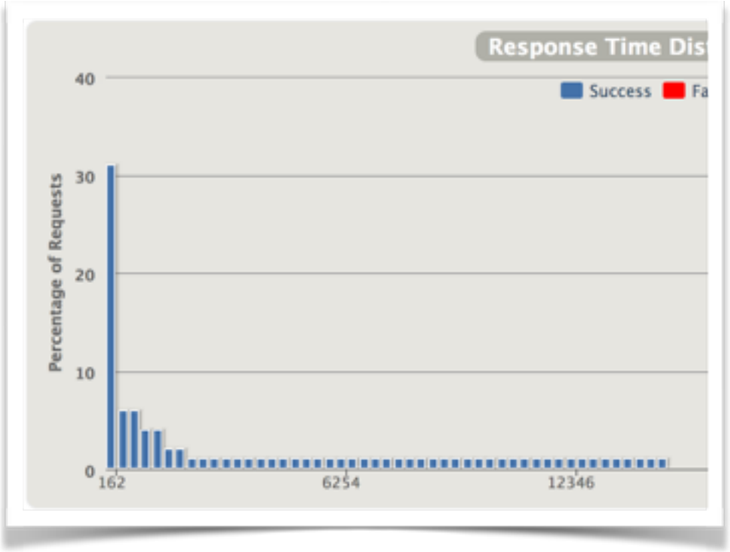
200 clients

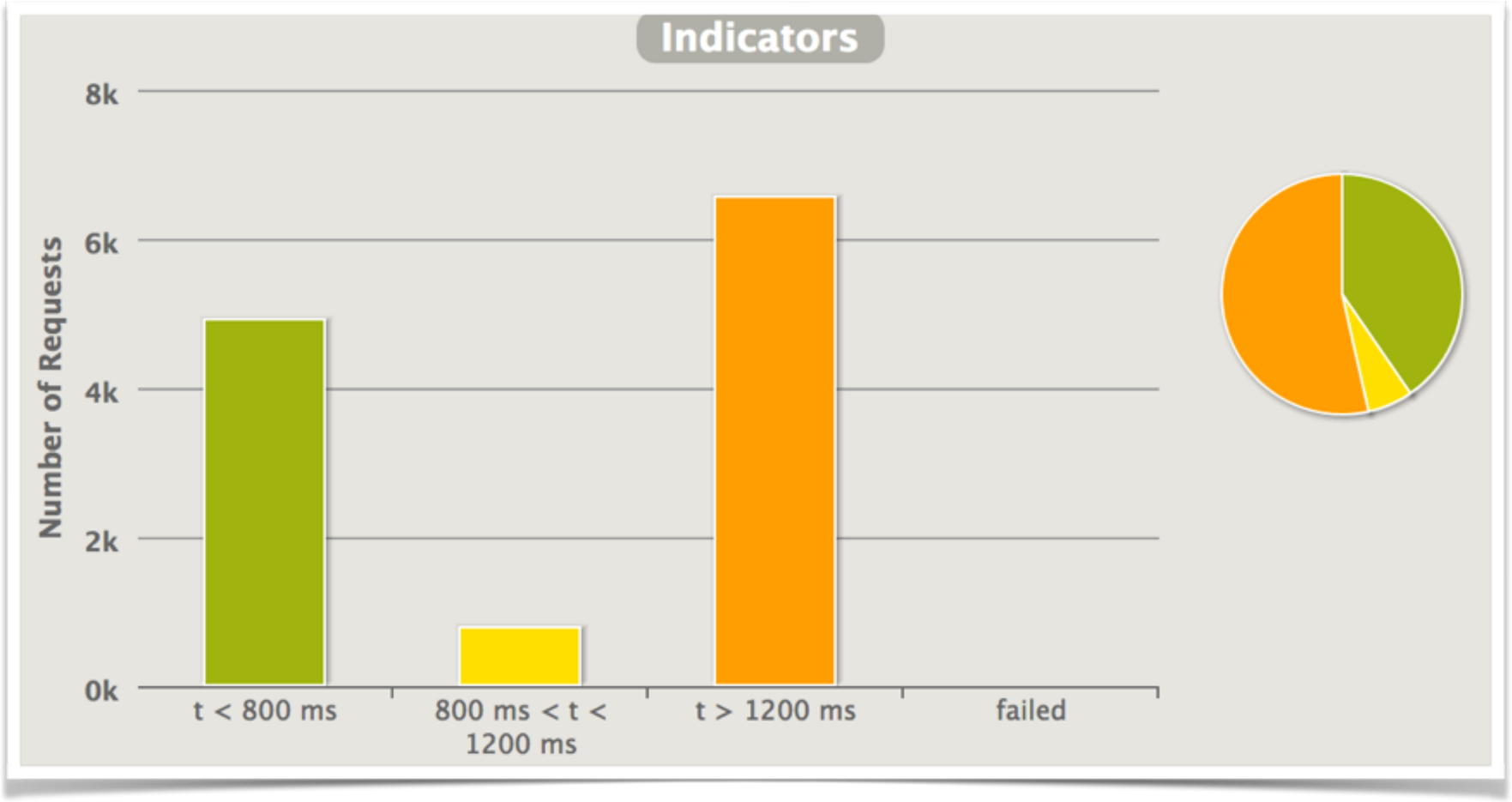




SERLi

400 clients





SERLi

Metrics everywhere !!!

Cassandra

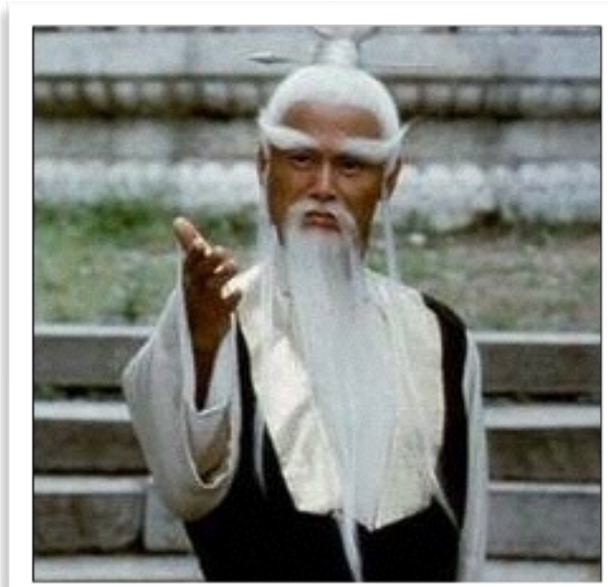
Messages akka

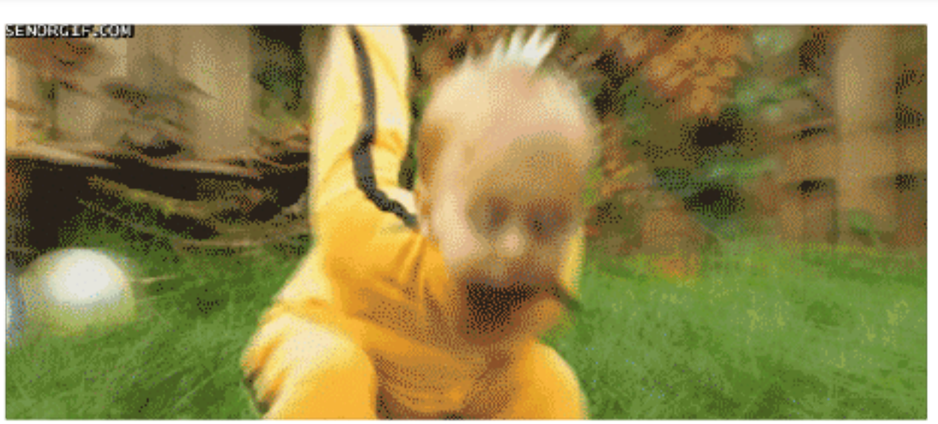
Journal

Elasticsearch

Les problèmes rencontrés

Nginx / mongo ...



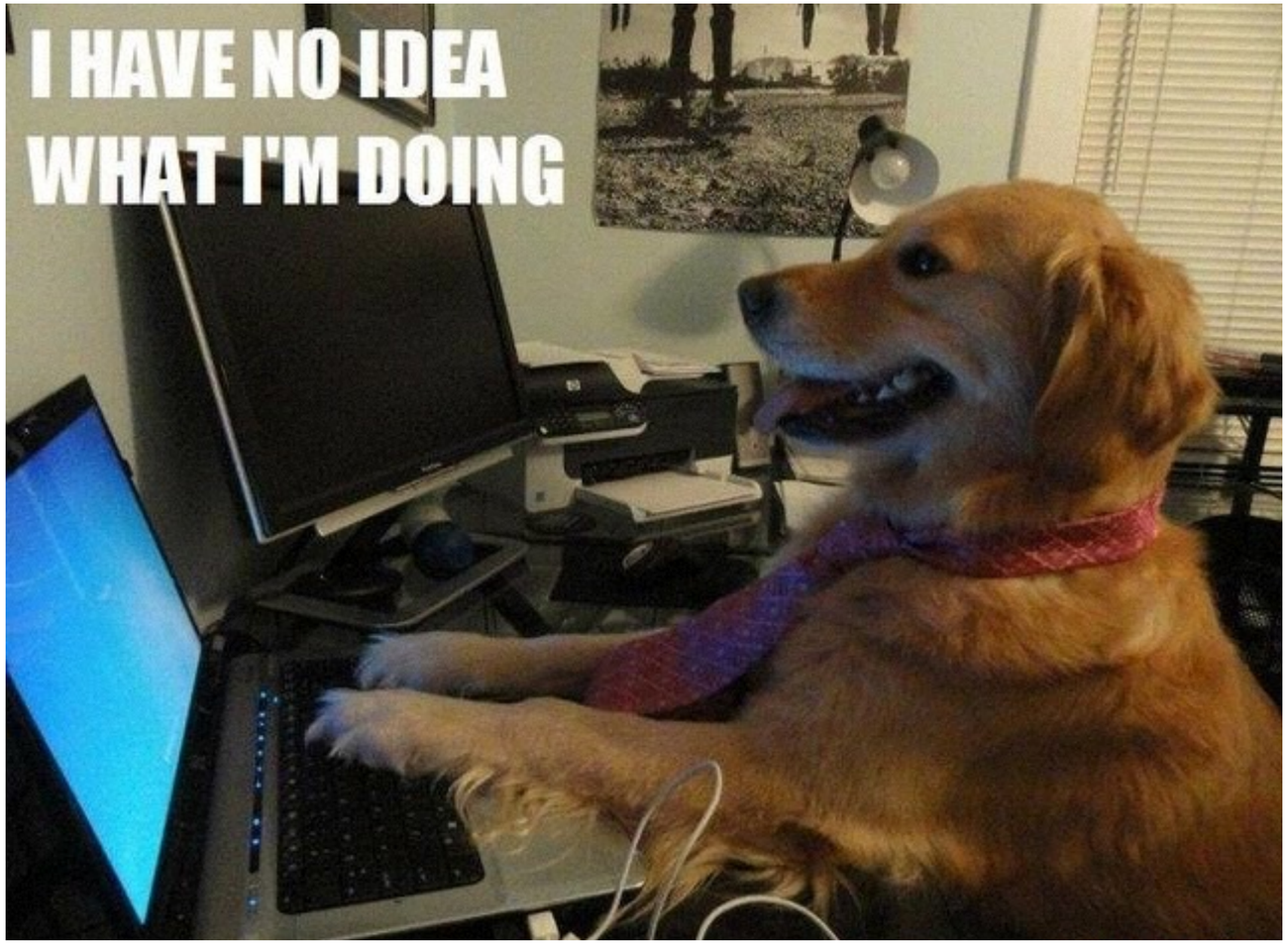


Cassandra

Bench web socket

SERLI

Configuration akka-cluster



En java

- RX java : helper pour l'async, implémentation des reactive-streams
- Ratpack : framework http async pour java 8, implémentation des reactive-streams
- reactor : lib de messages sur la ivm, implémentation des reactive-streams
- vert.x : plateforme polyglote à la node js
- play et akka, implémentation des reactive-streams

Avec servlet et java 8

- Servlet asynchrone

```
public class AsyncServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        AsyncContext asyncContext = req.startAsync();
        CompletableFuture<String> futureString = CompletableFuture.supplyAsync(() -> "Reponse");
        futureString
            .thenAcceptAsync((string) -> {
                try {
                    asyncContext.getResponse().getWriter().print(string);
                } catch (IOException e) {
                    //Gérer l'erreur
                }
            })
            //Terminer :
            .whenCompleteAsync((aVoid, throwable) -> {
                asyncContext.complete();
            });
        ;
    }
}
```

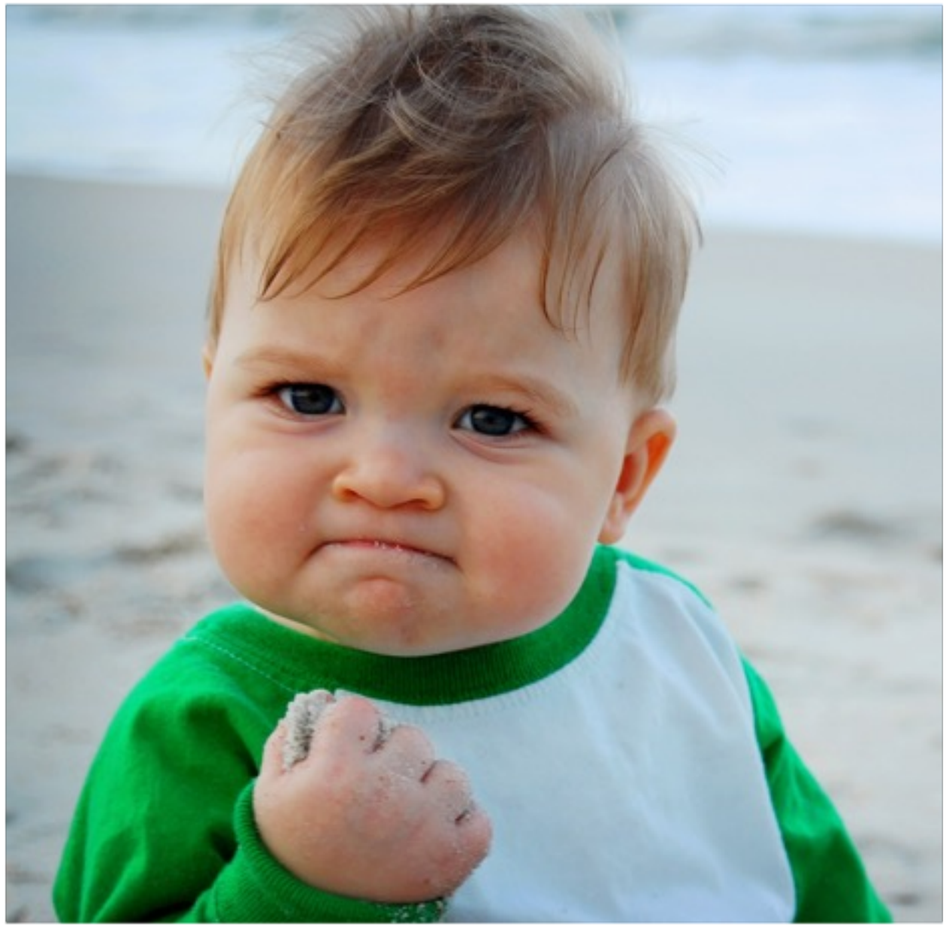
SERLI

Avec spring mvc et rx java

```
@SpringBootApplication
public class Application {

    @RequestMapping("asyncGet")
    public DeferredResult<String> asyncGet(){
        DeferredResult<String> deferredResult = new DeferredResult<>();
        Observable
            .from(CompletableFuture.supplyAsync(() -> "Reponse"))
            .map(String::toUpperCase)
            .doOnNext(deferredResult::setResult)
            .doOnError(deferredResult::setErrorResult);
        return deferredResult;
    }
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Conclusion



scala

akka

Scala

play

SERLI

Liens

- Le projet
  - https://github.com/mathieuancelin/reactive-raspberry-pi-cluster
- Le modèle réactif
  - http://www.reactivemanifesto.org/
  - http://reactive-streams.org
  - https://www.playframework.com
  - http://akka.io/

Liens

- Nosql
  - https://www.elastic.co/
  - http://cassandra.apache.org/
- Systèmes distribués
  - http://the-paper-trail.org/blog/distributed-systems-theory-for-the-distributed-systems-engineer
- Exemple app reactives
  - http://fr.lichess.org/
  - http://fr.lichess.org/monitor

This is the end ...