Play! framework 2

Mathieu ANCELIN
SERLI

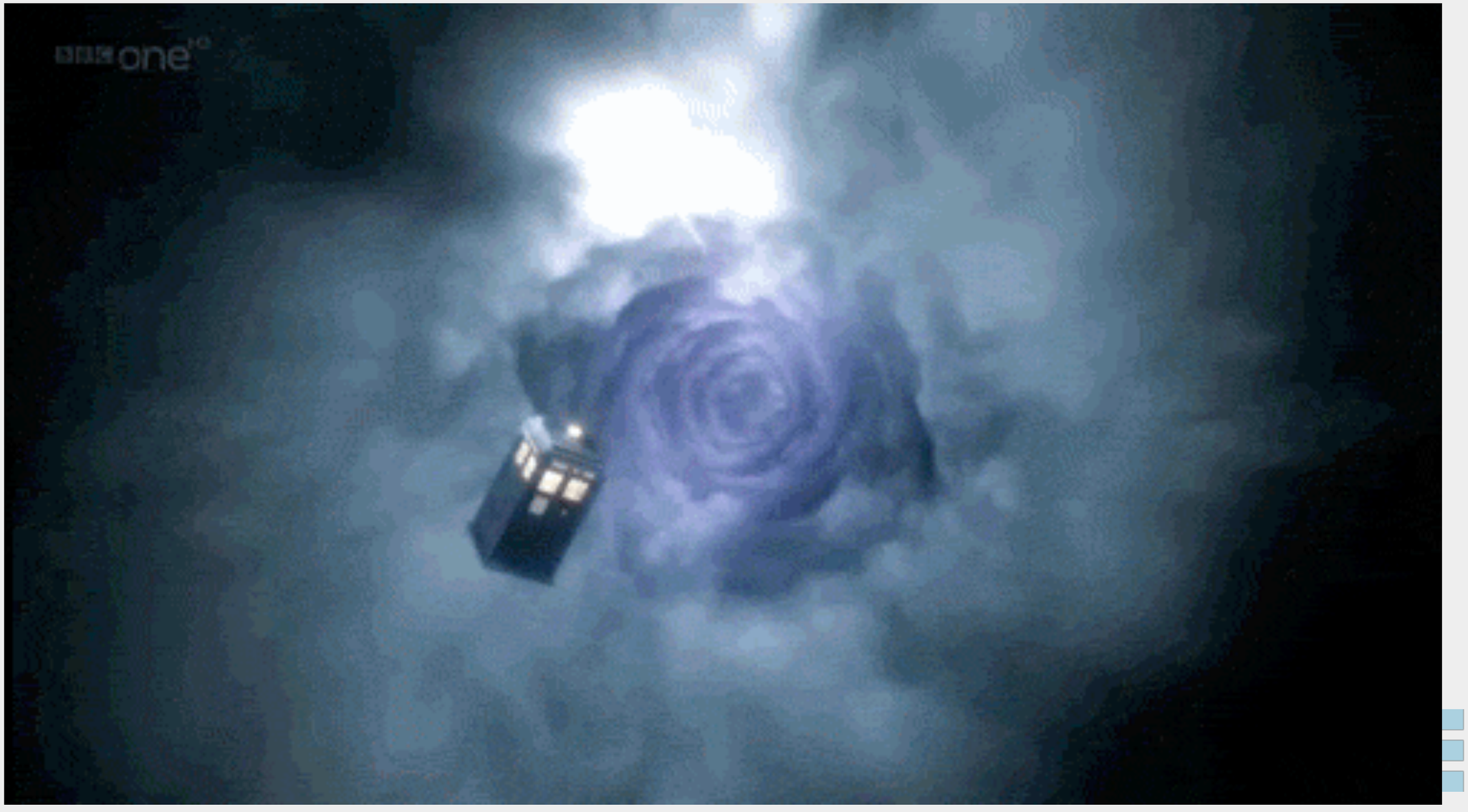
@TrevorReznik

Mathieu ANCELIN

- Ingénieur d'étude @SERLI
- Java, Scala, Web & OSS
  - ReactiveCouchbase, Weld, etc ...
  - Poitou-Charentes JUG
- Membre de l'expert group CDI 1.1 (JSR-346)
- Membre de l'expert group OSGi Enterprise
- @TrevorReznik

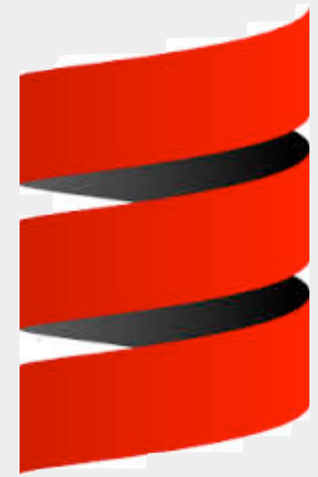
SERLI

Un peu d'histoire



Un peu d'histoire

- Réécriture from scratch
  - En Scala !!!
- Annoncé à Devoxx 2011
  - Guillaume Bort rejoinds le board Typesafe
  - Play Framework devient une brique de la stack Typesafe
  - Devient la stack web
- Version 2.0 en Avril 2012
- Le lead du framework revient à Typesafe



What's new ?

- Moteur Scala
- Fournit également une API Java
  - moins complète que la version Scala
- Datastore agnostic
- Cible les applications hautement scalables, peu consommatrices de ressources, au fil d l'eau, realtime data manipulation
- Composition d'applications
  - WOA

What's new ?

- Construit pour le web temps réel
  - entièrement asynchrone et non-bloquant
  - framework réactif (http://www.reactivemanifesto.org/)
  - Utilisation massive de Akka, les Futures et Promises
  - APIs utilisant les Futures
  - gestion très fine de pools d'exécution pour les traitements bloquants
- Introduit l'API des Iteratees pour faire face aux besoins du temps réel dans des applications web (reactive programming)
  - pour l'instant en Scala
  - peut-être bientôt en Java (lambda) ?

SBT

- Simple Build Tool
  - Outil de build standard de facto pour Scala
  - Gère le versioning des librairies
  - Gère les dépendencies
  - Gère la compilation et le packaging

Akka

- Librairie Scala très populaire
  - Paradigme de programmation orienté ‘acteurs’
  - Bien plus poussé que les acteurs Scala fournis par défaut
  - devient le standard dans Scala 2.10
- Enormes capacités pour traitements distribués et concurrents

Router

```
GET    /clients/all              controllers.Clients.list()

GET    /clients/:id              controllers.Clients.show(id: Long)

GET    /files/*name              controllers.Application.download(name)

GET    /clients/$id<[0-9]+>      controllers.Clients.show(id: Long)
```

```
GET    /                         controllers.Application.show(page)

GET    /                         controllers.Application.show(page= home")

GET    /:page                    controllers.Application.show(page)

GET    /clients                  controllers.Clients.list(page:Int?=1)
```

Contrôleurs

```
package controllers

import play.api.mvc._

object Application extends Controller {

  def index = Action {
    Ok("It works!")
  }

  def hello(name: String) = Action { request =>
    Ok("Hello" + name + "!")
  }
}
```

Vues

- Vues également écrites en Scala
- Vues typesafe
  - il faut déclarer les paramètres de la vue
- la vue est compilée

```
views/Application/index.scala.html
```

```
views.html.Application.index()
```

Vues

```
@(customer: Customer, orders: Seq[Order])

<h1>Welcome @customer.name!</h1>

<ul>
  @orders.map { order =>
    <li>@order.title</li>
  }
</ul>
<ul>
@for(order <- orders) {
    <li>@order.title</li>
}
</ul>
```

Appel depuis les contrôleurs

```
package controllers

import play.api.mvc._

object Application extends Controller {

  def index = Action {
    val bob = Customer("Bob")
    Ok(views.html.index(bob, bob.orders()))
  }
}
```

Formulaires (tuples)

```
val userFormTuple = Form(
  tuple(
    "name" -> text,
    "age" -> number
  )
)
```

Formulaires (case classes)

```
case class UserData(name: String, age: Int)

val userForm = Form(
  mapping(
    "name" -> text,
    "age" -> number
  )(UserData.apply)(UserData.unapply)
)
```

Formulaire (binding)

```
userForm.bindFromRequest.fold(
  formWithErrors => {
    BadRequest(views.html.user(formWithErrors))
  },
  userData => {
    val newUser = models.User(userData.name, userData.age)
    val id = models.User.create(newUser)
    Redirect(routes.Application.home(id))
  }
)
```

Anorm

- JDBC pour Scala !!!

```
case class Song( id: Long = -1L, path: String, name: String, artist: String,
        album: String, likeit: Long, dontlikeit: Long, played: Long )

object Song {

    val simple = {
        get[Long]( "song.id" ) ~ get[String]( "song.filepath" ) ~
        get[String]( "song.name" ) ~ get[String]( "song.artist" ) ~
        get[String]( "song.album" ) ~ get[Long]( "song.likeit" ) ~
        get[Long]( "song.dontlikeit" ) ~ get[Long]( "song.played" ) map {
            case id ~ path ~ name ~ artist ~ album ~ likeit ~ dontlikeit ~ played =>
              Song( id, path, name, artist, album, likeit, dontlikeit, played )
        }
    }
}
```

Anorm

```
def findAll() = DB.withConnection { implicit connection =>
    SQL( "select * from song" ).as( Song.simple * )
}

def findById( id:Long ) = DB.withConnection { implicit connection =>
    SQL( "select * from song s where s.id = {id}" ).
         on( "id" -> id ).as( Song.simple.singleOpt )
}
```

Cache

```
Cache.set("item.key", connectedUser)

val maybeUser: Option[User] =
                  Cache.getAs[User]("item.key")

val user: User = Cache.getOrElse[User]("item.key") {
  User.findById(connectedUser)
}

Cache.remove("item.key")
```

Json

```
val json: JsValue = Json.parse("""
{
  "user": {
    "name" : "toto",
    "age" : 25,
    "email" : "toto@jmail.com",
    "isAlive" : true,
    "friend" : {
      "name" : "tata",
      "age" : 20,
      "email" : "tata@coldmail.com"
    }
  }
}
""")
```

Json

```
val json = Json.obj(
  "users" -> Json.arr(
    Json.obj(
      "name" -> "Bob",
      "age" -> 31,
      "email" -> "bob@gmail.com"
    ),
    Json.obj(
      "name" -> "Kiki",
      "age" -> 25,
      "email" -> JsNull
    )
  )
)
```

```
val jsonString: String =
  Json.stringify(json)
```

Json : JsPath

```
val name: String = (json \ "user" \ "name").as[String]

name === "toto"

val maybeName: Option[String] =
        (json \ "user" \ "name").asOpt[String]

maybeName === Some("toto")
```

Json : Reader / Writer

```
implicit val creatureReads = (
  (__ \ "name").read[String] and
  (__ \ "isDead").read[Boolean] and
  (__ \ "weight").read[Float]
)(Creature)
```

```
implicit val creatureWrites = (
  (__ \ "name").write[String] and
  (__ \ "isDead").write[Boolean] and
  (__ \ "weight").write[Float]
)(unlift(Creature.unapply))
```

```
val js = Json.obj(
  "name" -> "gremlins",
  "isDead" -> false,
  "weight" -> 1.0F
)
val c = js.as[Creature]
val js2 = Json.toJson(c)
```

Asynchrone

- Le framework est complètement asynchrone by design
  - Utilisation de Akka pour traiter les requêtes
- Possibilité de renvoyer des résultats asynchrones depuis les contrôleurs
  - utile pour les traitements long
  - ne bloque pas les ressources

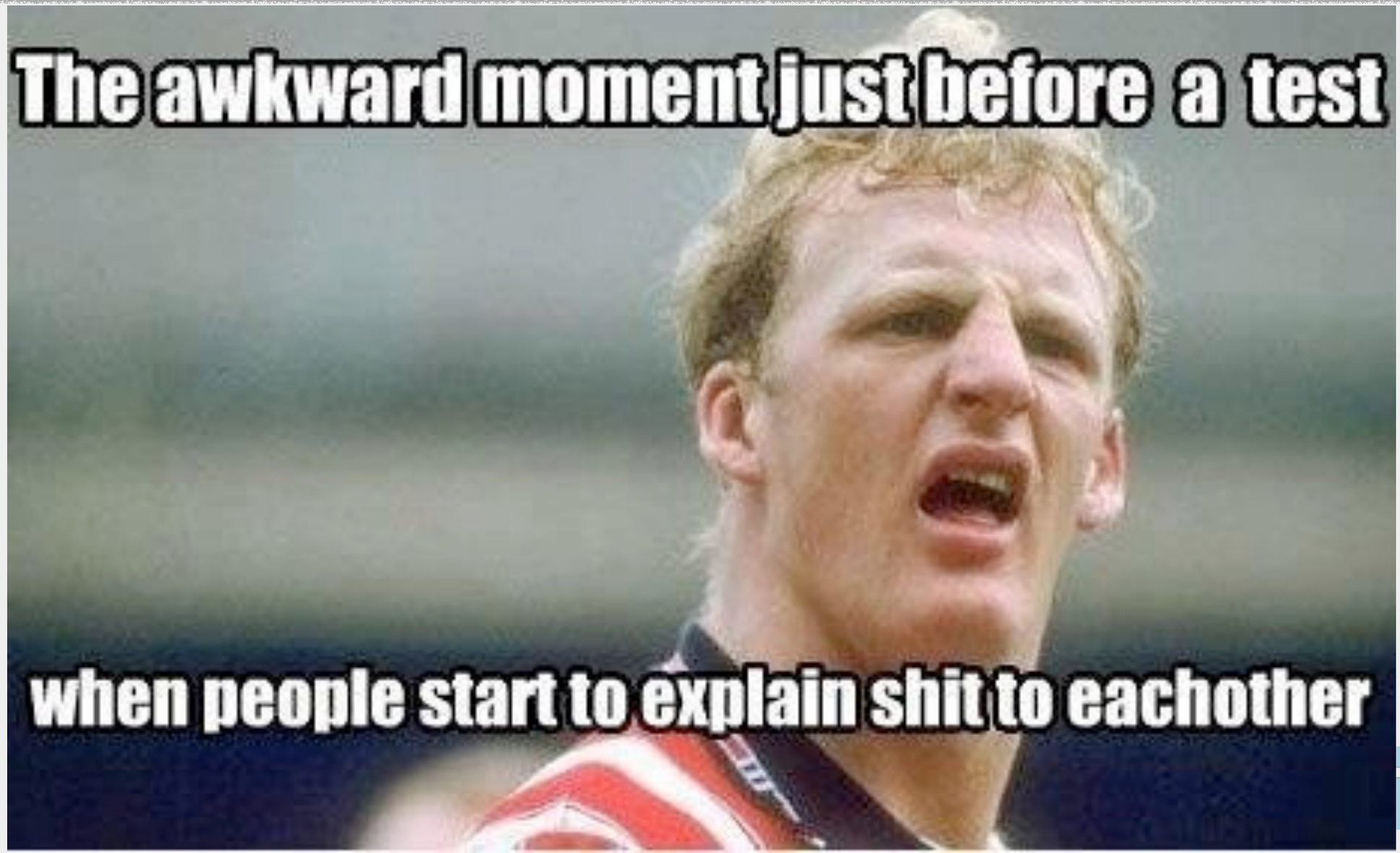
Asynchrone

```
package controllers

import play.api.mvc._
import play.api.libs.concurrent.Execution.Implicits._

object Application extends Controller {

  def index = Action.async {
    val bob = Customer("Bob")
    val promiseOfOrders = Future { bob.orders() }
    promiseOfOrders.map( orders =>
      Ok(views.html.index(bob, orders))
    )
  }
}
```

ExecutionContext

- La plupart des méthodes asynchrones on besoin d'un ThreadPool pour fonctionner
  - en scala l'ExecutionContext
- Créé à partir d'un ExecutorService Java standard
  - Créé dans chaque système Akka
- La plupart du temps pour éviter d'alourdir la syntaxe d'appel des méthodes, l'ExecutionContext est passé implicitement
  - DAFUQ ???

WTF are Iteratees ?



WTF are Iteratees ?

- API de traitement et de manipulation de flux de données réactive et progressive
  - modélisation fonctionnelle de producteurs, transformateurs et consommateurs de flux de données composables à la manière d'un pipeline
  - modèle de traitement commun à n'importe quel flux de données
  - n'est pas spécifiquement liée aux I/O et/ou à des fonctionnalités web temps réel
- Dans les faits, les Iteratees sont utilisées au coeur du framework Play 2

Trois composants essentiels

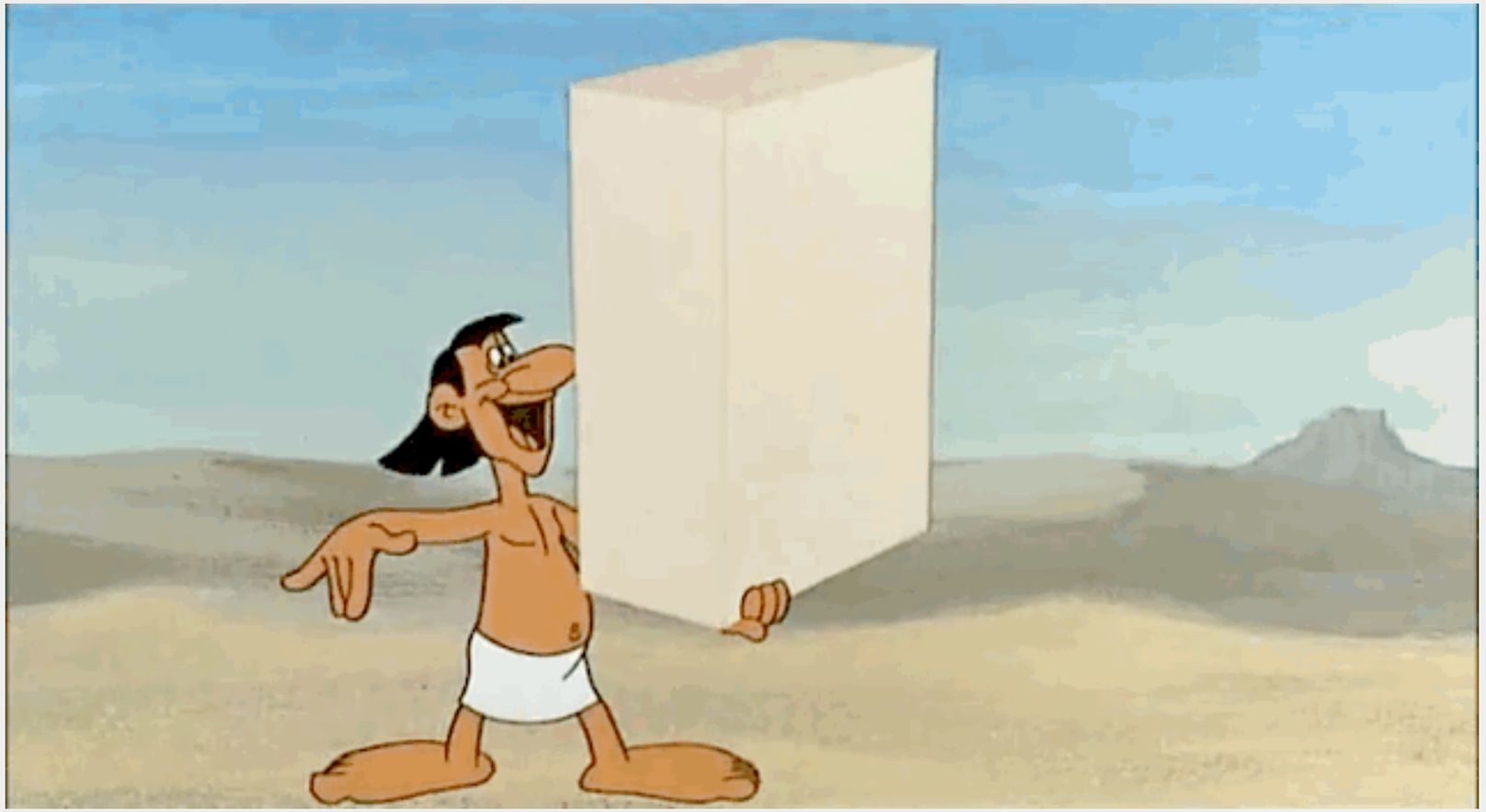
Enumerator

```
Enumerator[Bloc]
```

Iteratee

`Iteratee[Bloc, Colonne]`

Enumeratee

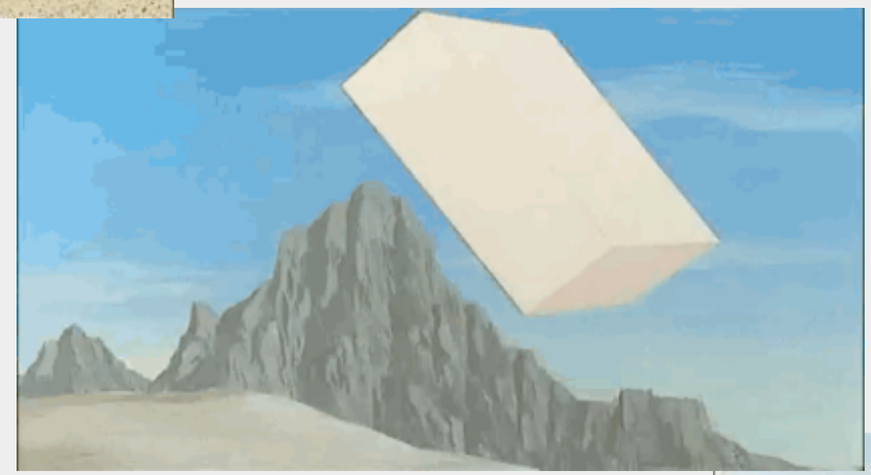


`Enumeratee[BlocALEnvers, BlocALEndroit]`

```
Enumerator[BlocALEnvers]
```

```
Iteratee
    [BlocALEndroit, Pyramide]
```

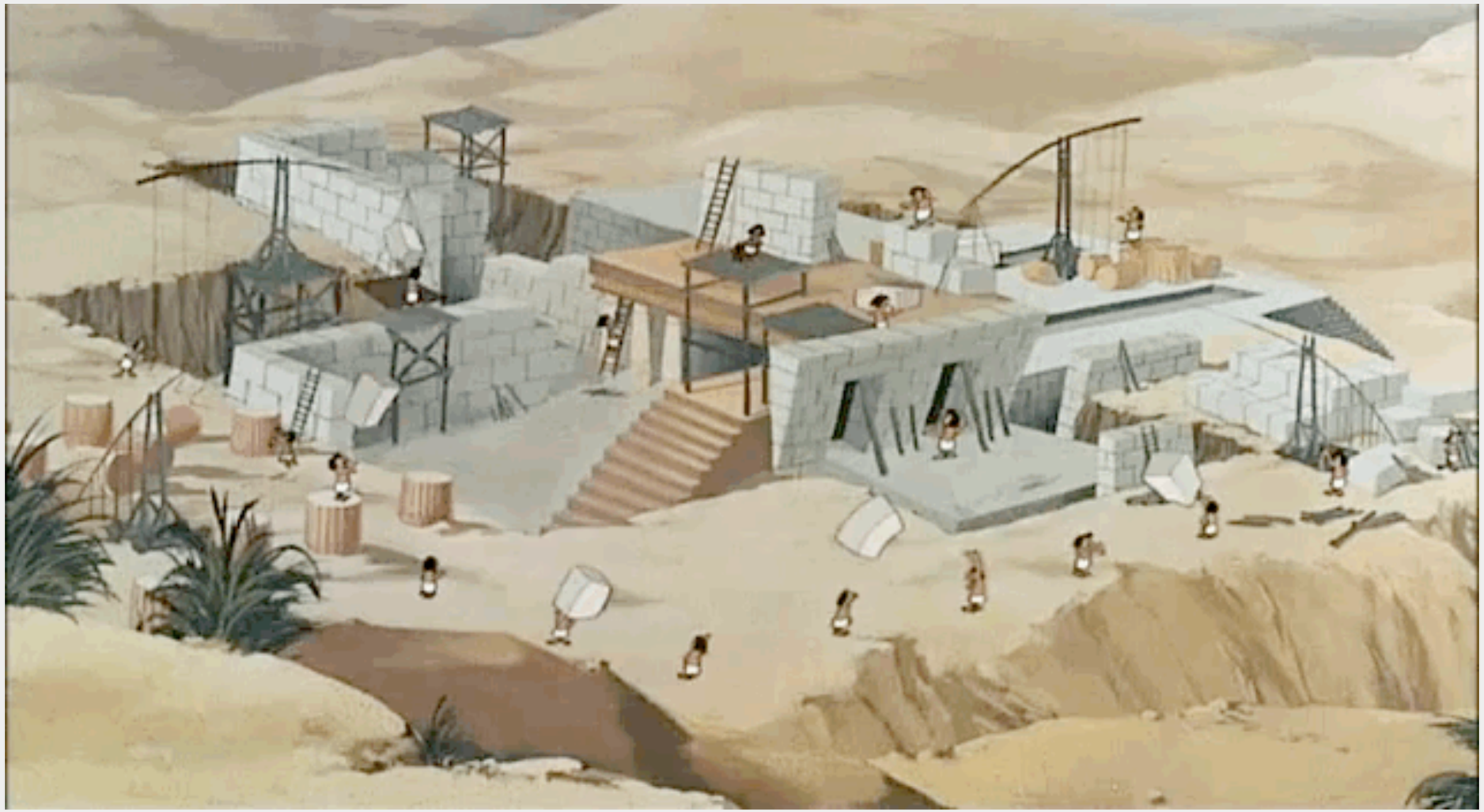
Enumerator[BlocALEnvers]



Enumeratee
[BlocALEnvers, BlocALEndroit]

Iteratee
[BlocALEndroit, Pyramide]

Votre application temps réel



avec play

SERLI

Utilisation

- Stream de données
  - Zound, Deezer like, VOD, etc ...
- Mise à jour de l'UI, Collaboration temps réel
  - http://fr.lichess.org
- Récupération de contenu paginé
  - web services Klout
- Stream et consommation depuis des process systèmes
  - PlayCLI
- Tout ce que vous voulez ...

Manipulation

```
case class BlocALEnvers(id: String)
case class BlocALEndroit(id: String)
case class Pyramide(blocs: Seq[BlocALEndroit])

val enumerator: Enumerator[BlocALEnvers] = Enumerator(
  BlocALEnvers("1"), BlocALEnvers("2"),
  BlocALEnvers("3"), BlocALEnvers("4")
)

val enumeratee: Enumeratee[BlocALEnvers, BlocALEndroit] =
                    Enumeratee.map { bloc => BlocALEndroit(bloc.id) }

val iteratee: Iteratee[BlocALEndroit, Pyramide] =
     Iteratee.fold(Pyramide(Seq[BlocALEndroit]()))
       ((pyra, bloc) => Pyramide(pyra.blocs :+ bloc))

(enumerator &> enumeratee).apply(iteratee).
                      flatMap(_.run).map(pyra => println(pyra))

// Pyramide(BlocALEndroit(1), BlocALEndroit(2), BlocALEndroit(3), BlocALEndroit(4))
```

WebSockets

```
def index =  WebSocket.using[String] { request =>

    val (out,channel) = Concurrent.broadcast[String]
    val in = Iteratee.foreach[String] {
      msg =>
      println(msg)
        channel push("RESPONSE: " + msg)
    }
    (in,out)
}
```

Streams

```
def index = Action {
  val enumerator = Enumerator(
    Json.obj("foo" -> "bar"),
    Json.obj("foo" -> "bar"),
    Json.obj("foo" -> "bar")
  )
  Ok.chunked(enumerator).andThen(Enumerator.eof)
}
```

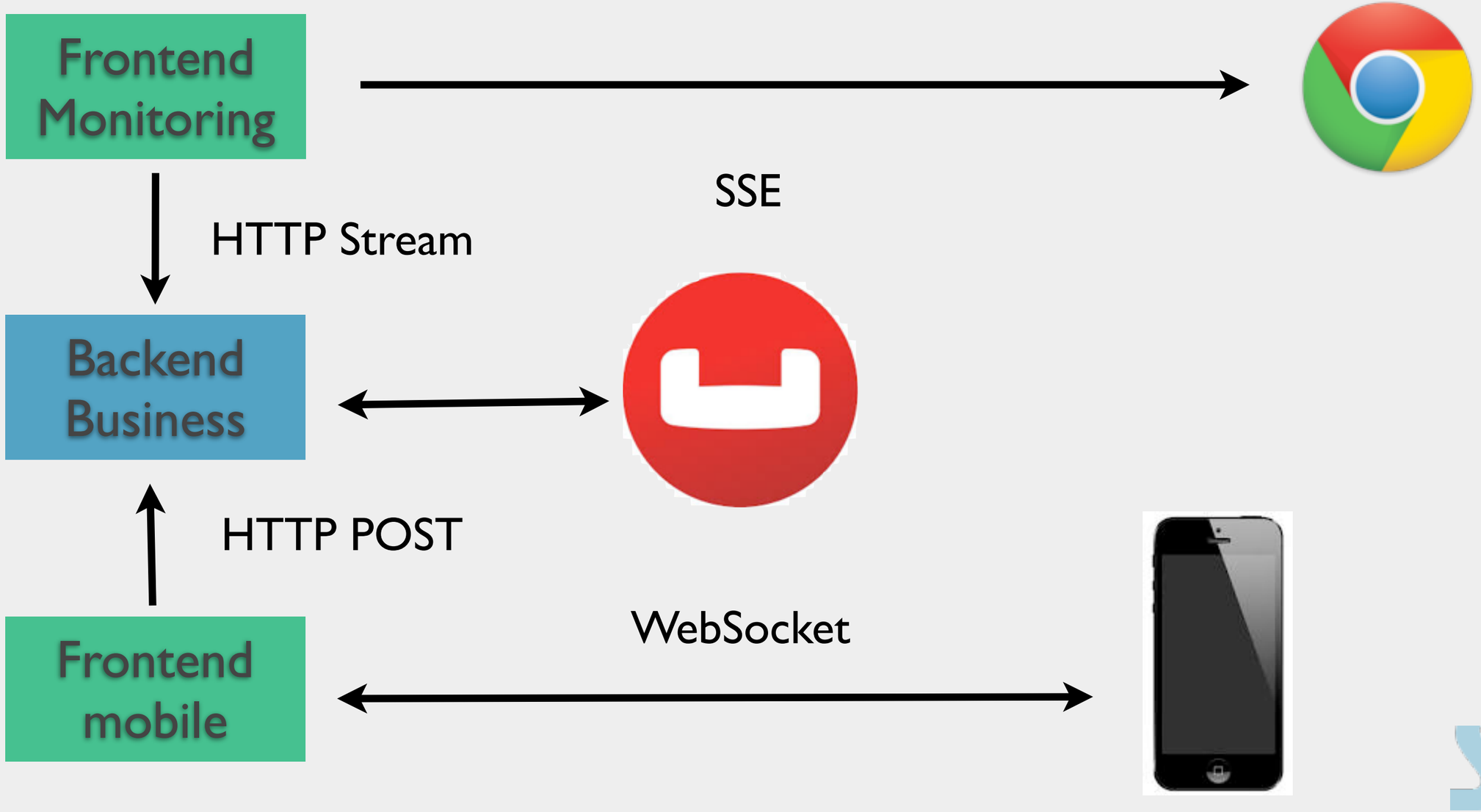
SSE

```
def index = Action {
  val enumerator = Enumerator(
    Json.obj("foo" -> "bar"),
    Json.obj("foo" -> "bar"),
    Json.obj("foo" -> "bar")
  )
  Ok.chunked(
    enumerator.through( EventSource() )
  ).as("text/event-stream")
}
```

Who ???

- LinkedIn
- Gilt
- The guardian
- Valtech
- Klout
- Apple
- Intel
- ...

TP


Frontend Monitoring
SSE
HTTP Stream
Backend Business
HTTP POST
Frontend mobile
WebSocket