Actor Model - Akka

Mind7 Consulting – Ismail Hrimech

Page de publicité

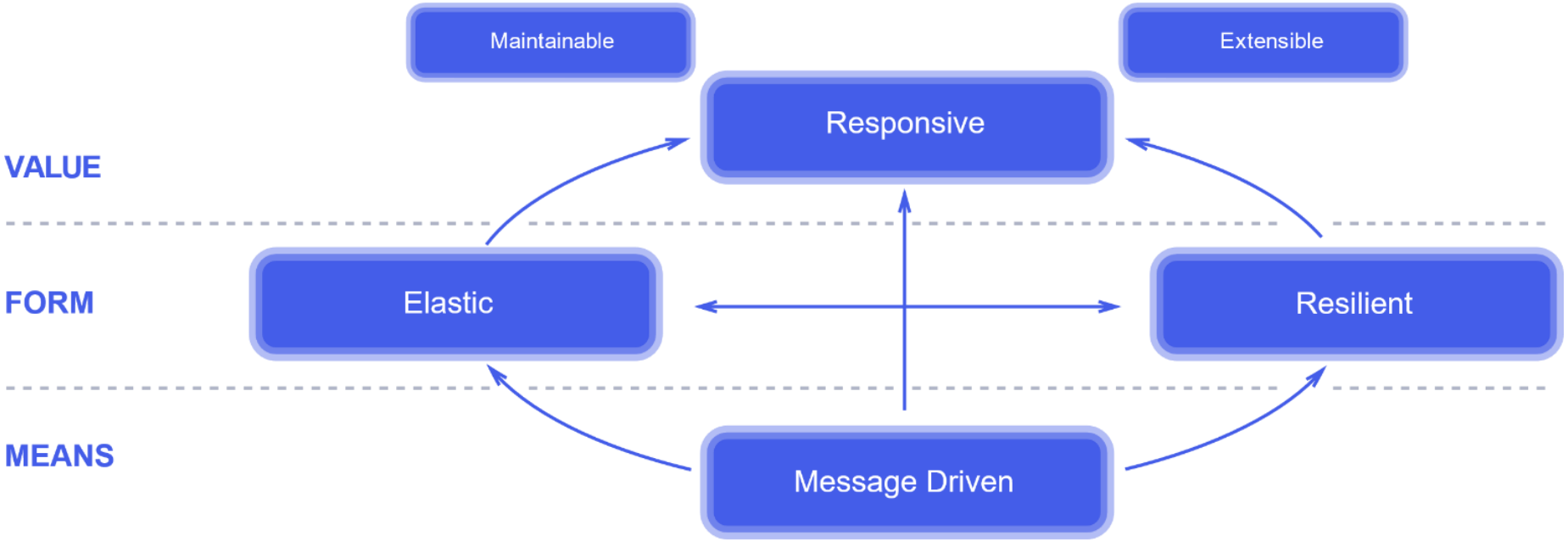
- Mind7 consulting, partenaire de Lightbend

- Ismail Hrimech : Développeur couteau suisse (Scala, Akka, Spark, Go, Java, Angular, NodeJS, PHP ...)

Sommaire

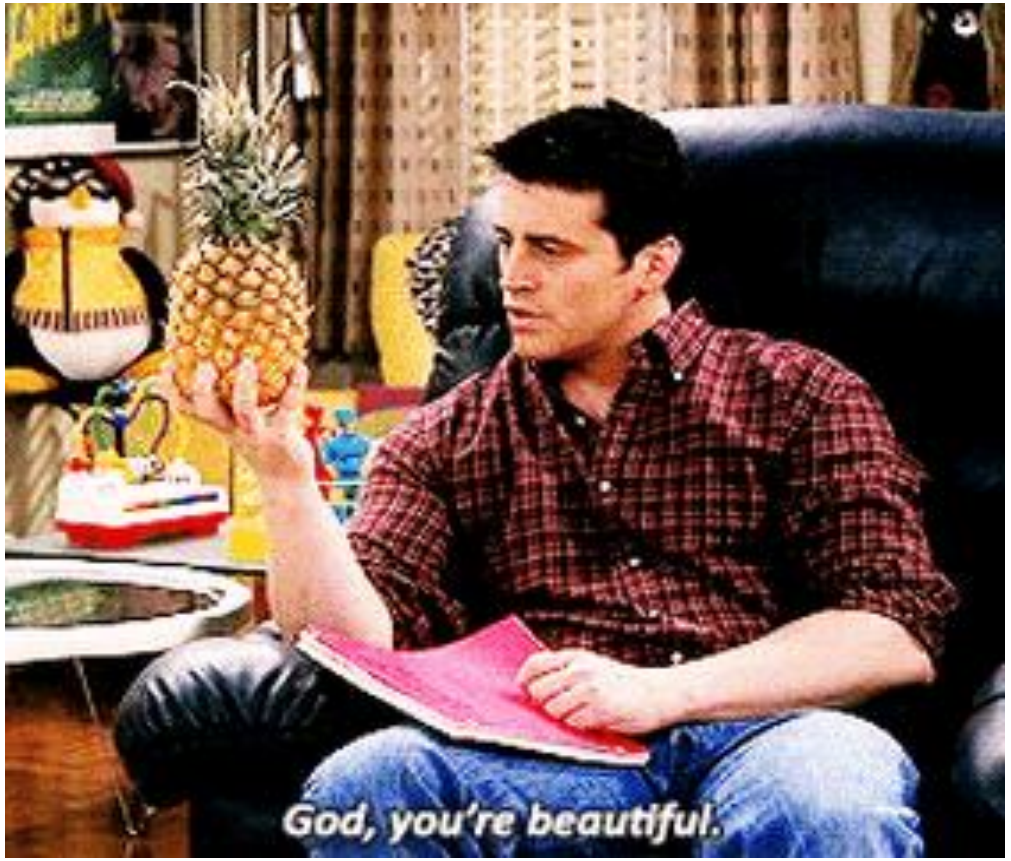
- Introduction
- Notre bible : The Reactive Manifesto
- Actor model (théorie)
- Scala : Guide de survie en milieu hostile
- Actor model (pratique : modélisation d'une problématique)
- Akka : Théorie
- Akka (Pratique) : Création de deux acteurs liés
- Akka http : Théorie
- Akka http : Pratique – Endpoint requêtant sur une BDD

Il était une fois, le manifeste réactif (reactive manifesto)



Vous pouvez exprimer votre adhésion en signant la page officielle : https://www.reactivemanifesto.org/fr

ET LE MODÈLE ACTEUR (ACTOR MODEL) ?



L’actor model en quelques mots

Actor model = Modèle de programmation concurrentielle.

Chaque acteur est responsable d’une partie du traitement (SRP).

Chaque acteur possède une adresse unique et communique avec les autres acteurs de manière asynchrone.

Chaque acteur peut créer de nouveaux acteurs, envoyer/recevoir des messages d’autres acteurs et modifier son comportement.

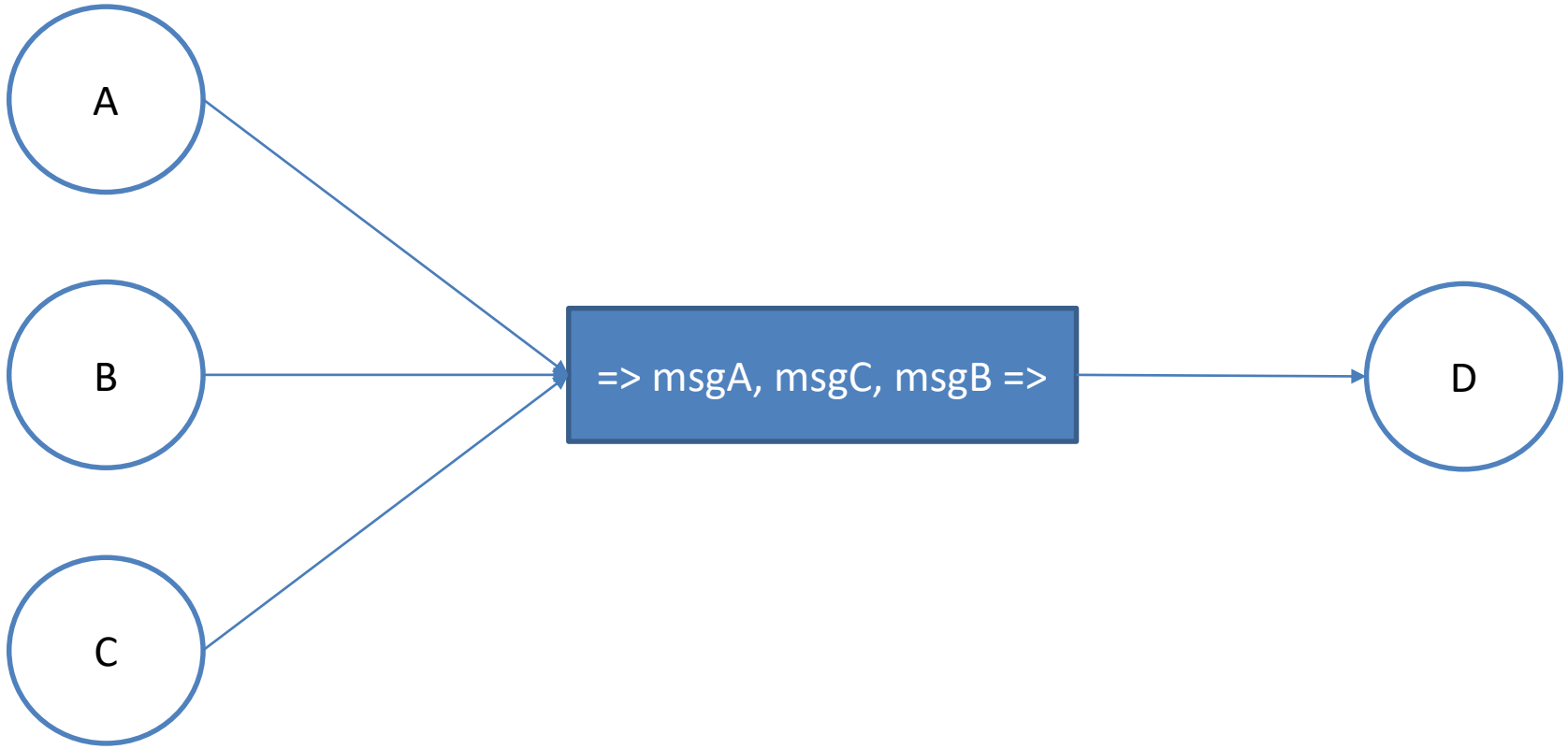
Chaque acteur possède une mailbox dans laquelle arrivent ses message. Ils sont ensuite traités en FIFO.

Bah, c’est quoi la différence avec ma POO en Java ?

D’accord, petit exemple alors

- Supposons que les objets A, B et C veulent communiquer simultanément avec D pour changer son état. Comment fait on en Java (ou ailleurs en POO) ?

Avec l'Actor Model



Qu’est ce qu’un Acteur ?

Actor Model : Les acteurs

- L’acteur est la fonction primitive du modèle acteur => Plus petite unité de conception
- Un acteur peut:

- Créer un nouvel Acteur,
- Changer son état ou son comportement (décisions internes),
- Réagir à un message :
  - En changeant son état,
  - En modifiant son comportement,
  - En envoyant des messages (à de nouveaux acteur ou à l’acteur expéditeur),
  - En créant de nouveaux acteurs.

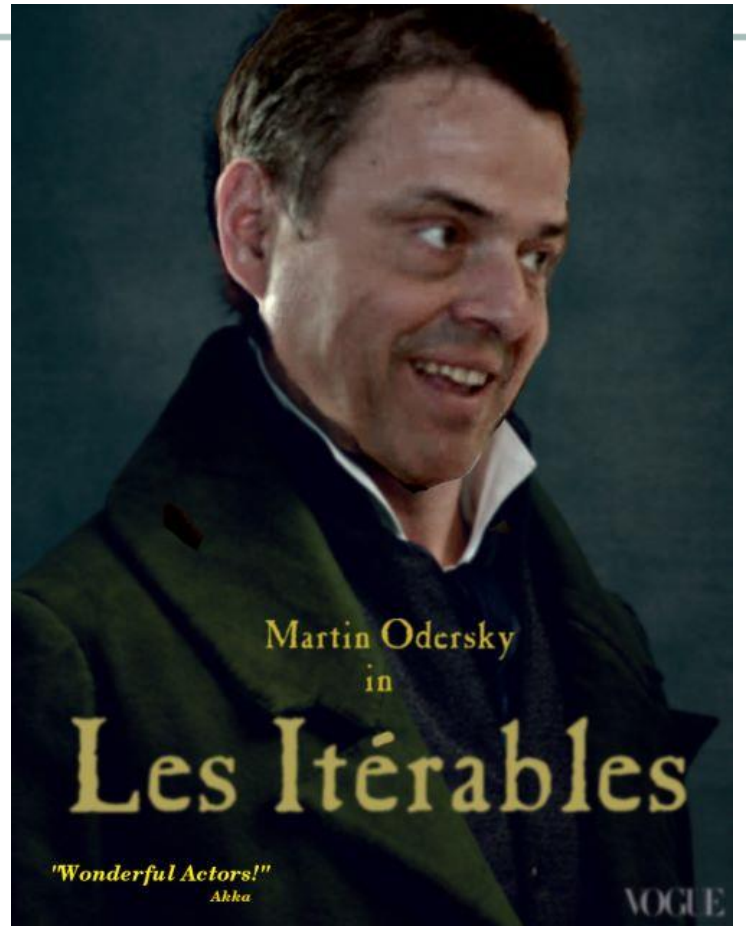
Comment communiquent les acteurs ?

Interaction des acteurs

- Communication via échange de message exclusivement,
- At most once delivery (best effort),
- Chaque acteur possède une adresse (référence, path)
- L’acteur réagit à un message (en changeant son comportement, son état persisté)

Deuxième page de pub: Scala : guide de survie*

- Un mélange de POO et de fonctionnel:
  - Toutes les valeurs sont des objets
  - Toutes les fonctions sont des valeurs
- Statiquement typé (avec inférence de type),
- Utilisation des librairies Java possible,
- Traits => interfaces qui peuvent implémenter des méthodes
- Case class => Pratique pour faire du pattern matching



Afin de respecter le temps imparti, sont pris quelques raccourcis pour accélérer la prise en main

Scala, guide de survie, cette fois avec du code

```
// Constante (immutables)
val jeSuisUneConstante = "Un string, un int, un array, etc..."
// variables (mutables)
var jeSuisUneVariable = "Un string, un int, un array, etc..."

// un trait, comme une interface, en mieux
trait JeSuisUnTrait

// une classe
class JeSuisUneClass() extends JeSuisUnTrait

// un objet (singleton) => object compagnon de la class dans ce cas particulier (même nom)
object JeSuisUneClass
```

Scala, guide de survie, cette fois encore avec du code, ce titre est beaucoup trop long

```scala
// case class/object => utiles pour le pattern matching
case class JeSuisUneCaseClass(unParametre: String)
case object JeSuisUnCaseObject

val stuff: Any
stuff match {
  case JeSuisUnCaseObject => println("ok")
  case JeSuisUneCaseClass(param) => println(s"ok $param")
  case cestJusteUneChaine: String => println(s"en fait c'est juste : $cestJusteUneChaine")
  case _: Int => println("c'est le numéro gagnat du loto")
  case _ => println("tout le reste")
}
```

Exercice de pratique – Scala (Exercice 01)

CheatSheet : https://docs.scala-lang.org/cheatsheets/

Akka : Implémentation de l'Actor Model

- Implémentation en Scala et Java du modèle acteur dans la JVM
- Dans Akka, tout est acteur
- Création d'acteur : En étendant la class « Actor »
- Implémentation d'acteur :
  - Top level : « system.actorOf(props: Props, name: String) »
  - Enfants : « context.actorOf(props: Props, name: String) »

  Où :
    - Props: Propriétés de l'acteur
    - Name: (optionnel) Un nom donné à l'acteur. U

Akka: Communication des acteurs

- Les acteurs communiquent en s'envoyant des messages via la méthode « ! » appelé bang ou « ? » (ask)
- Exemple :

```
val referenceVersActeur: ActorRef = system.actorOf(Props[MonActeur], "nom-acteur")
referenceVersActeur ! message
```

- Les communications sont asynchrones
- L'acteur reçoit les messages dans la mailbox et les traite en mode FIFO
- Lorsque l'acteur ne traite pas de message il est en suspend et ne consomme pas de mémoire

Un exemple ?

```
class ActeurExemple extends Actor {
  val actorEnfantRef = context.actorOf(Props[ActeurEnfant])

  override def preStart(): Unit = {
    println("Actor Example started")
  }

  override def postStop(): Unit = {
    println("Acteur Example arrêté")
  }

  override def receive: Receive = {
    // self est une référence de type actorRef qui renvoie l'adresse de l'acteur
    case "stop" => context.stop(self)
    case x => logMessageReceived(x)
  }

  def logMessageReceived(message: Any): Unit = {
    println(s"message: $message")
    actorEnfantRef ! message
  }
}
```

Un autre exemple ?

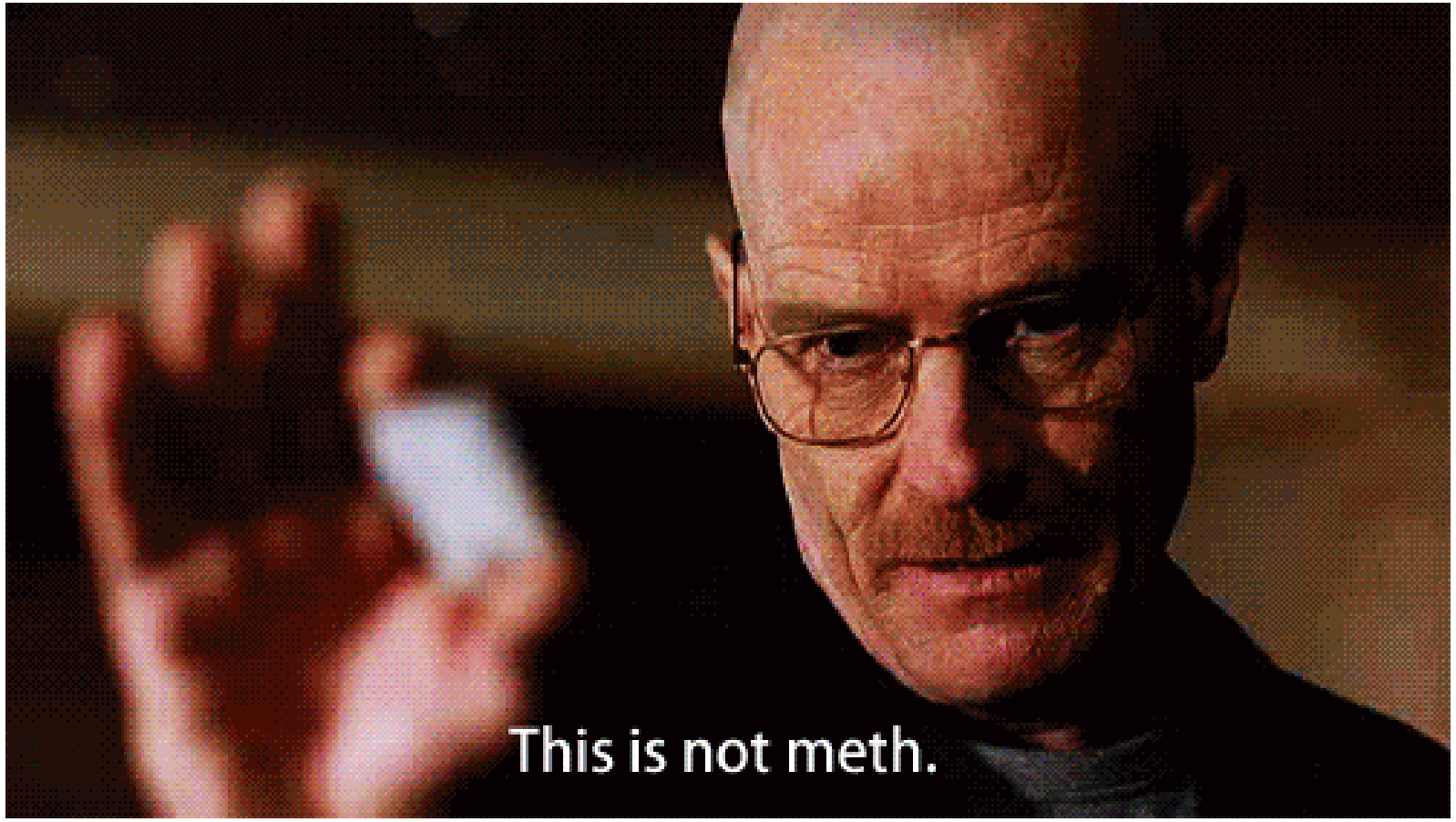
```
object MainObject extends App {

 val  message =  "Message simple"
  // Création d'un systeme d'acteur
  val system = ActorSystem("MeetupAkka")

  // Création d'une instance de ActeurSimple (elle doit renvoyer une référence)
  val acteurSimple = system.actorOf(Props[ActeurSimple], "acteur-simple")

  // TODO 04 Envoyer à la référence de acteur simple un message urgent et un message simple
  // TODO executer ensuite pour vérifier le résultat
  acteurSimple ! message
}
```

Practice Time : Créez votre propre Acteur (Exercice 02)

Akka (suite)

- Arrêt d’un acteur:
  - Envoyer un message pour qu’il s’arrête seul (via « context.stop(self) »)
  - L’arrêter si on possède sa référence « context.stop(ActorRef) »
  - Envoyer un poisonPill

Exercice 2bis: PoisonPill ? (si on a du temps…)



C’est sympa tout ça, mais comment je ramène ma donnée ?

Akka http

- Akka http est une librairie (et non un framework) basée sur Akka Streams,
- Fournit des outils pour effectuer des requêtes http (client) et en recevoir (serveur),
- Implémente l'Actor Model mais peut-être intégré à un autre modèle de programmation.

Akka http (pratique) Exercice 03

Pourquoi utiliser le modèle acteur ?

- Correspond aux besoins de calcul concurrentiels,
- Peu couteux en ressources,
- Simplifie les interaction et simple de conception => Un workflow se traduit naturellement dans un modèle acteur,
- Évite les « locks »,
- Learning curve => apprentissage rapide,

Use cases : Logistique (Client Mind7), Paiement en ligne (Paypal), Pari en ligne (Betfair), IoT (Samsung), Retail (Walmart) etc …
Cas clients => https://www.lightbend.com/case-studies

Autres use cases => https://stackoverflow.com/questions/4493001/good-use-case-for-akka

Bravo, vous êtes arrivés à la fin de la présentation !