# Retour d'expériences
# Spark appliqué à la génomique

**BIOINFO@LIPME**

# Architecture

**Très satisfaisante à l'échelle de notre équipe**

- **RAM/CPU: cluster Spark est un sous ensemble de notre cluster grid-engine (SGE)**
  - + Pas de risques (financiers/techniques) lié au choix
  - − Schedulers (GE/YARN) indépendants ➔ compétition sur les ressources CPU/RAM
- **STOCKAGE**
  - + Jobs Spark ➔ hadoopfs
  - + Jobs SGE ➔ beegfs
- **Spark**
  - + v3 vs v2: Meilleure utilisation de la mémoire
  - − Les ressources sont réservées au moment de l'exécution

    ➔Une tâche qui (re)échoue (ex: partitionnement non adapté, « data skew ») n'est pas rejouée avec plus de ressources.

# Apprentissages (point de vue développeur)

**Investissement important  nécessaire pour maitriser suffisamment afin de bien exploiter l'architecture distribuée**

## Langages

+ Scala est le langage natif de Spark, on accède à toutes les fonctionnalités
+ Scala est orienté objet, fonctionnel, exploite la JVM, concis, la parallélisation multithreads très simple
- Pénible d'optimiser la consommation mémoire (cf JVM)

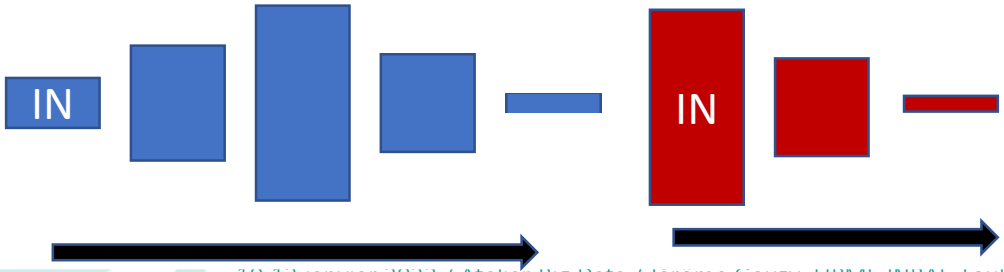
## Architecture distribuée

+ Transparent de distribuer une structure de données
- L'utilisation de structures de données immutables consomme beaucoup de RAM (cf  JVM)
± Pour exploiter les données, le programmeur doit bien comprendre à quel niveau il se trouve (driver ou worker) pour savoir s'il accède à toutes les données (driver) ou seulement à celles d'une partition (worker)
± L'objectif doit être d'éviter de récupérer trop de données au niveau du driver
- L'optimiseur interne fait parfois sur un échantillon des évaluations plus couteuses que l'exécution brutale sur tout le jeu de données (fonction des données)
- Difficile de contrôler /comprendre l'optimiseur de code interne
± Il faut accepter que la séquence de code écrite n'est pas nécessairement celle qui sera exécutée

# Adéquation à l'analyse des séquences

**Tout est possible mais il faut maitriser**

- **Utilisation d'outils d'analyse de séquences**
  - Ce n'était pas notre objectif
  - – Peu d'outils, quelques outils bioinfo sont publiés par an
- **Implémentation d'outils d'analyse de séquences**
  - Notre objectif
  - \+ Binding scala/spark et C/C++ fonctionne bien dans les « User Defined Function » ➔ on peut étendre Spark à bas niveau.
  - API Génomique de bas niveau (redéveloppement scala ou binding) toujours en développement
  - – Big Genome Data != Big Business Data ➔ Pb RAM (cf JVM)




*Schémas classiques de transformation/filtrage des données dans les deux domaines*

# Adéquation au Machine Learning

**Tout a été possible assez simplement**

- **API ml scala**
  - + Nombreuses méthodes implémentées
  - + Relativement stable … désormais
  - + Simple à utiliser
  - + Une implémentation par type de méthode (peut-être limitant pour le chercheur en ML vs R mais pas pour l'utilisateur de méthodes)
  - + La scalabilité de notre programme (s'il est bien écrit) est gérée au niveau de l'architecture
- **Deep Learning**
  - API BigDL  non encore évalué
- **(Visualisation)**
  - – Bien moins développé qu'en R ou python

# Portabilité du code distribué que nous développons

**Bien, comme prévu**

- **Développement sur station de travail**
  - + IntelliJ (IDE) gère très bien les projets scala/sbt
  - + 1er niveau de débuggage très bien géré par IntelliJ
- **Exécution sur notre cluster Spark**
  - + Quasiment transparent d'un point de vue du code
  - - Mais le programme doit gérer correctement les accès aux différents filesystems
  - - Debuggage possible mais plus difficile qu'en local
- **Exécution dans un cloud privé**
  - Toujours pas testé

# Développement Architecture couplant intimement Données (DataLake) et Analyse (Spark)

**Pas encore mais projet financé à partir de 2023**

# Contributeurs: BIOINFO@LIPME

**2016-2022**

- **Axel Verdier**
- **Léo Géré**
- **Maël Chiotti**
- **Javad Razavi**

- **Sébastien Carrère**
- **Jérôme Gouzy**
- **Ludovic Legrand**