Architecture Spark / Hadoop

Ludovic Legrand

Big Data

3 V

Volume

- Volume de données en To et Po

Variété

- Diversité des formats Texte, Image, Video, Voix
- Données plus ou moins structurées

Vélocité

- Vitesse du flux de données

L'intersection de ces 3 propriétés sont à l'origine des technologies Big Data pour le stockage et le traitement

Big Data

3 V en génomique

Volume

- En génomique les données pour un organisme ou pour un projet peuvent dépasser les 10To

Variété

- Principalement du texte plus ou moins structuré
- Images et vidéos sont de plus en plus utilisés

Vélocité

- Flux de données entrant faible et essentiellement en batch
- Le phénotypage haut débit et l'IoT vont vers des flux en stream plus important
- Temps de traitement en heures ou en jours

Big Data

3 V en génomique

Encore loin des besoins de la physique ou des GAFAM

Evolution des projets et des analyses

- Plus de données *omique et de phénotypage
- Etudes à des échelles plus importantes (espèce, écosystème…)
- Actualisation plus fréquente des analyses
  - Prendre en compte les nouvelles données
  - Rejouer avec des nouveaux algorithmes

Big Data

Caractéristiques des technologies Big Data

Matériel informatique accessible

- Maitrise des couts

Rapprocher le traitement de la donnée

- Limiter les goulots d'étranglement comme le réseau et les I/O
- Exploiter au mieux les ressources du cluster

Passage à l'échelle

- Exécuter le même code sur une machine, un cluster ou dans le cloud
- Etendre facilement les capacités de traitement et de stockage

Hadoop

Hadoop = MapReduce + HDFS + YARN

Hadoop

- Initié par Google et Yahoo! En 2006
- Projet Apache depuis 2009
- Evolution lente
  - Hadoop 1.0 en 2011
  - Hadoop 3.0 en 2016

MapReduce

- Framework de calcul distribué
- Traitement batch de type MapReduce
- Ecriture sur disque entre les etapes

HDFS

- Système de fichiers distribué
- Tolérant aux pannes et extensible

YARN

- Gestionnaire de ressources

Hadoop

HDFS

Performance

- Distribution des données
- **Co-localisation des données et des traitements**

Robustesse

- Réplication des fichiers x3
- Haute disponibilité des métadonnées
- Write once Read many

Passage à l'échelle

- Ajouter des nœuds (volume + traitement)
- Ajouter des disques (volume uniquement)
- **Pas de limites de volume ou de taille de fichier**

Hadoop

Architecture

NameNode

- Métadonnées des fichiers et des blocs
- Surveillance de la réplication
- Surveillance des DataNode

DataNode

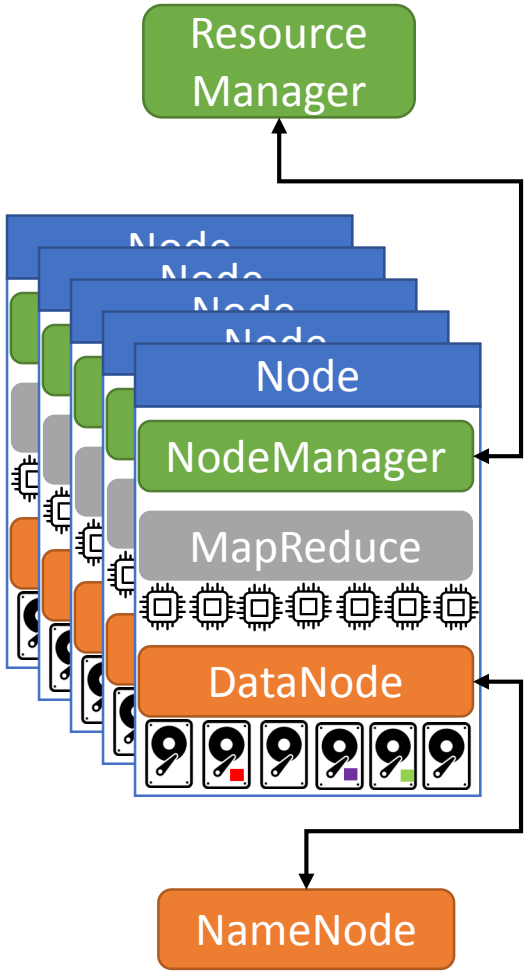
- Stockage des blocs
- Reporting des blocs

ResourceManager (YARN)

- Surveille les NodeManager
- Gère les ressources

NodeManager

- Provisionne et surveille des exécuteurs
- Gestion des ressources du serveur



Spark

Présentation

Histoire

- Création en 2009 par Matei Zaharia à Berkeley
- Le projet rejoint la fondation Apache en 2013
- Matei Zaharia co-fonde l'entreprise Databricks en 2019

Développement

- 1.0 en 2014
- 3.3 (2022) est la plus récente
- 1 à 2 versions par an
- 1000+ contributeurs

Spark

Présentation

Polyvalent

- Traitements batch, interactif et « temps réel »
- API riche : SQL, Graph, Machine Learning…

Performant

- Utilisation de la RAM entre les étapes
- Performances jusqu'à x100 comparées à Hadoop MapReduce
- Réorganisation des instructions et optimisation du code

Polyglotte

- Scala, Java
- Python, R, SQL

Intégration avec de nombreuses sources de données

- SQL, NoSQL, CSV, JSON, XML
- Parquet, ORC, Avro

Spark

Composants

Spark SQL

- Requêtes SQL
- Nombreuses fonctions optimisées

Spark ML

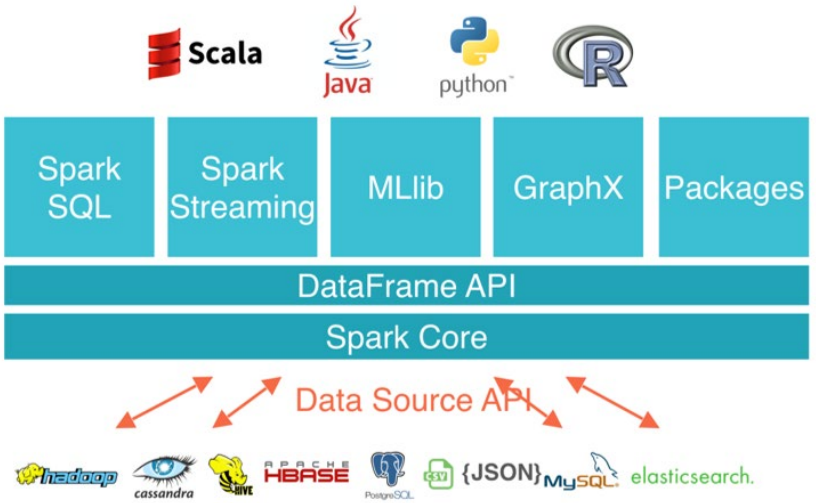
- Machine Learning
- Moins riche que R/Python
- Algorithmes distribués

Spark Streaming

- Traitement de flux de données
- Traitement en micro-batch

GraphX

- Traitement sur les graphes
- N'est pas une base de données graphe



Spark

Glossaire

Driver

- Orchestre les tâches
  - Construction et optimisation du graphes des tâches
  - Réorganisation du code
  - Distribution des tâches
- Surveille les exécuteurs et les tâches
  - Relance les tâches en erreur
- Négocie avec le Cluster Manager pour les ressources
- Single Point Of Failure de Spark

Direct Acyclic Graph (DAG)

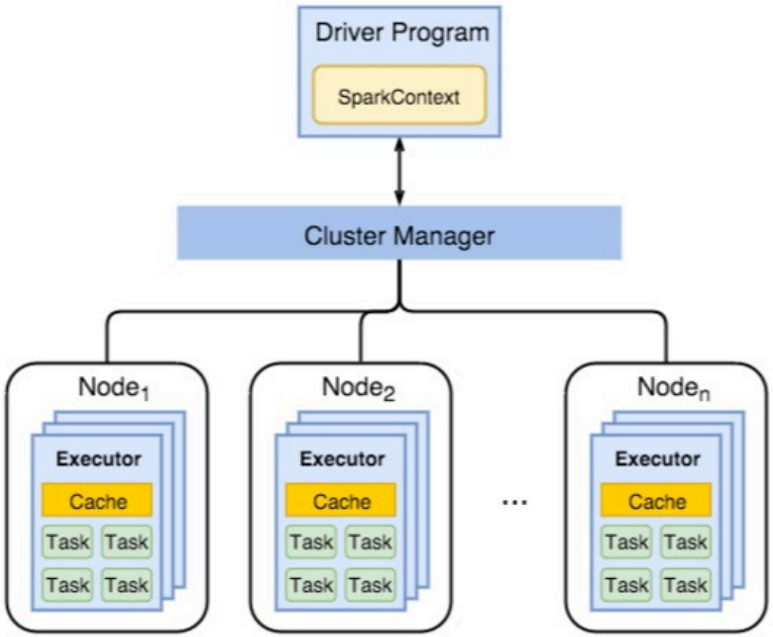
- Graphe de l'ensemble des opérations sur un jeu de données

Exécuteur

- Conteneur avec des ressources fixes (CPU, RAM)
- Exécute les tâches

Spark

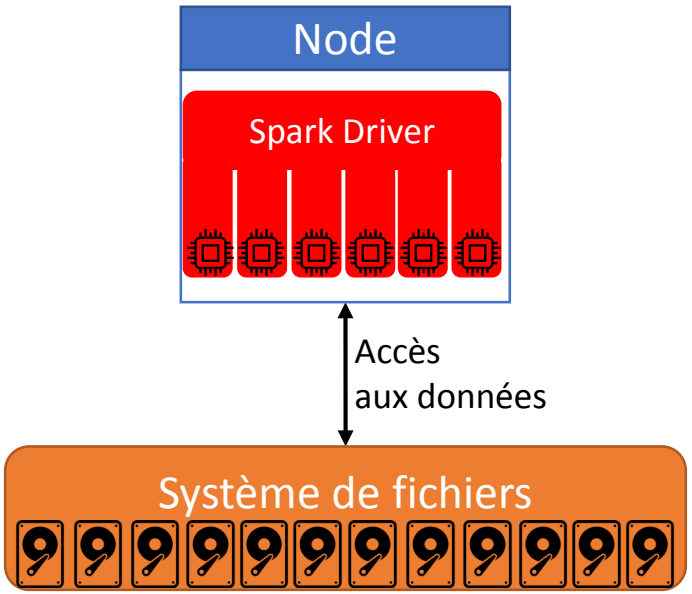
Briques logicielles



1. Instanciation du driver
   - Soit en local (mode client)
   - Soit sur le cluster (mode cluster)
2. Demande des ressources au Cluster Manager
   - N conteneurs avec X cœurs et Y RAM
3. Instanciation des conteneurs
4. Création/optimisation du DAG par le Driver
5. Distribution des tâches sur les exécuteurs par le Driver

Spark local

Architecture



Architecture

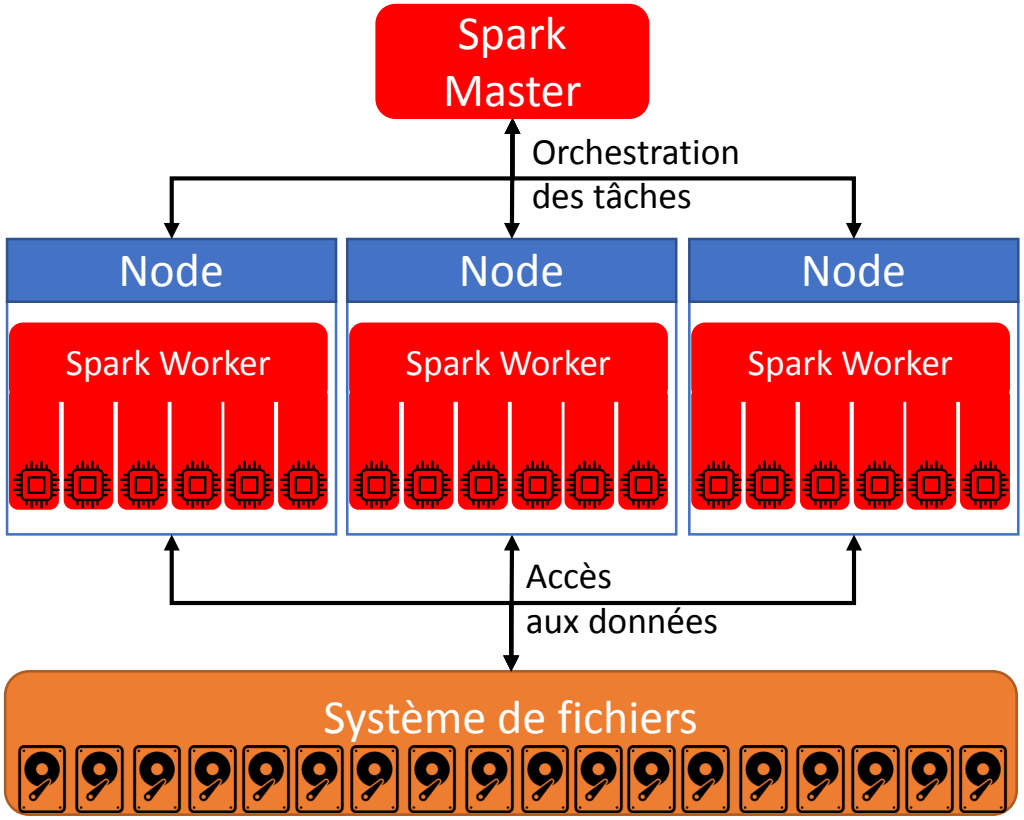
- Uniquement Spark
- Une seule machine
- Le driver fait tout

Performance

- Nécessite un réseau et un système de fichiers assez performant
- Perte de la colocalisation données/traitement
- Ressources limitées à une machine

Spark « standalone »

Architecture



Architecture simple

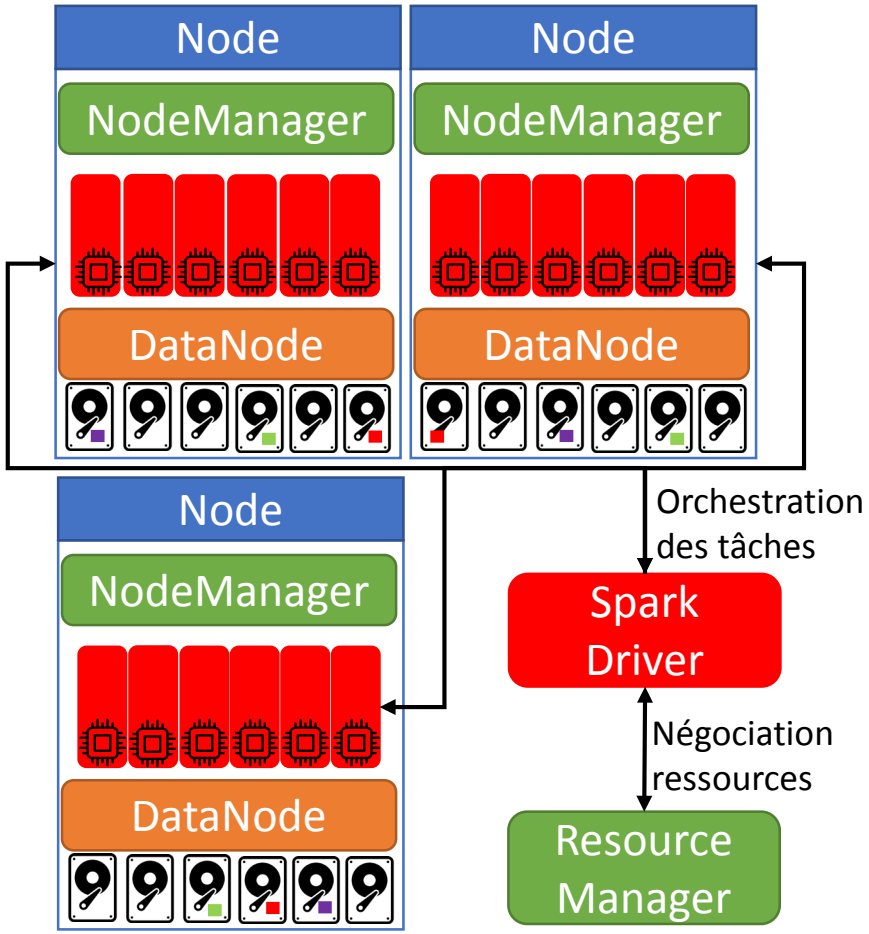
- Uniquement Spark
  - 1 Spark Master
  - N Spark Worker
- Infrastructure informatique type HPC

Performances

- Réseau et système de fichiers doivent être très performant
- Perte de la colocalisation données/traitement
- Passage à l'échelle facile

Spark + Hadoop

Architecture



Architecture complexe

- Nombreux services
- Infrastructure informatique à anticiper

Performance

- Colocalisation traitement/calcul
- Passage à l'échelle assez facile

Spark

Glossaire

RDD/DataFrame/DataSet

- Structure de données utilisées par Spark
- Structure de données distribuées et immutables

Transformation

- Modification du jeu de données
- Filtre, Map …
- Accumulées dans le DAG mais pas exécutées de suite (Lazy)

Action

- comptage, écriture …
- provoque l'exécution des transformations précédentes

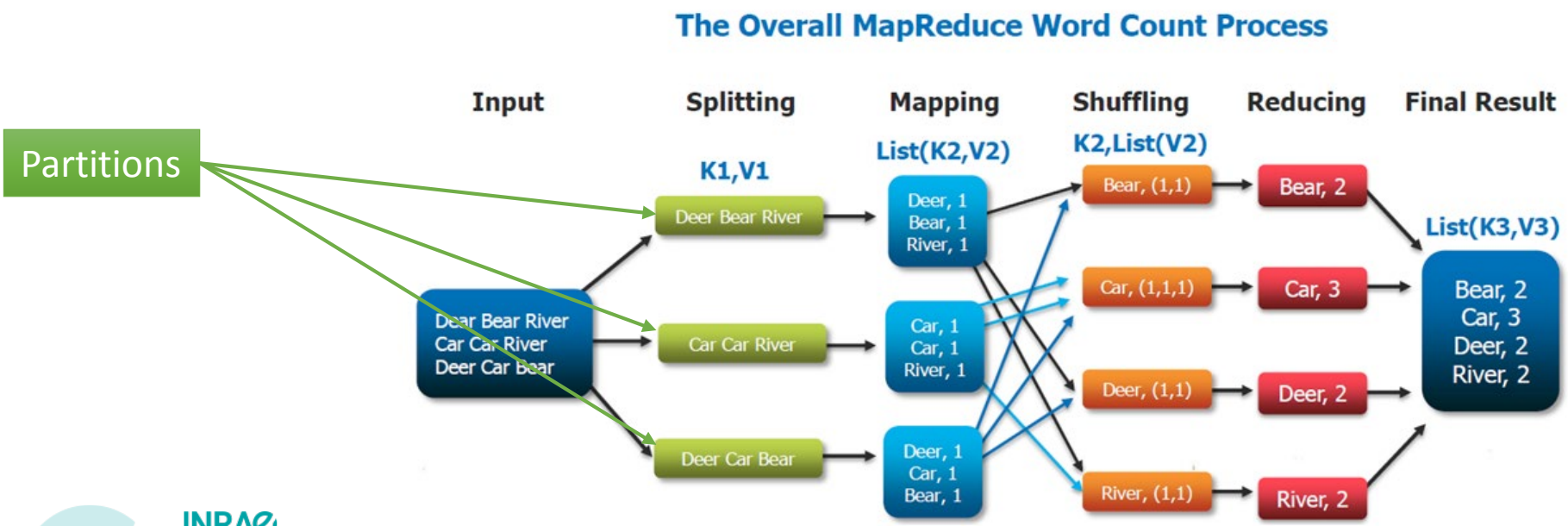
Spark

Anatomie d'un job map-reduce

```
val conf = new SparkConf().setAppName("Word Count Application")
val sc = new SparkContext(conf)

val count = sc.textFile(s"maprfs://spark-ics/user/llegrand/word_
    .flatMap(_.split(" "))
    .map(w => (w, 1))
    .reduceByKey(_ + _)

count.saveAsTextFile(s"maprfs://spark-ics/user/llegrand/word_cou
```



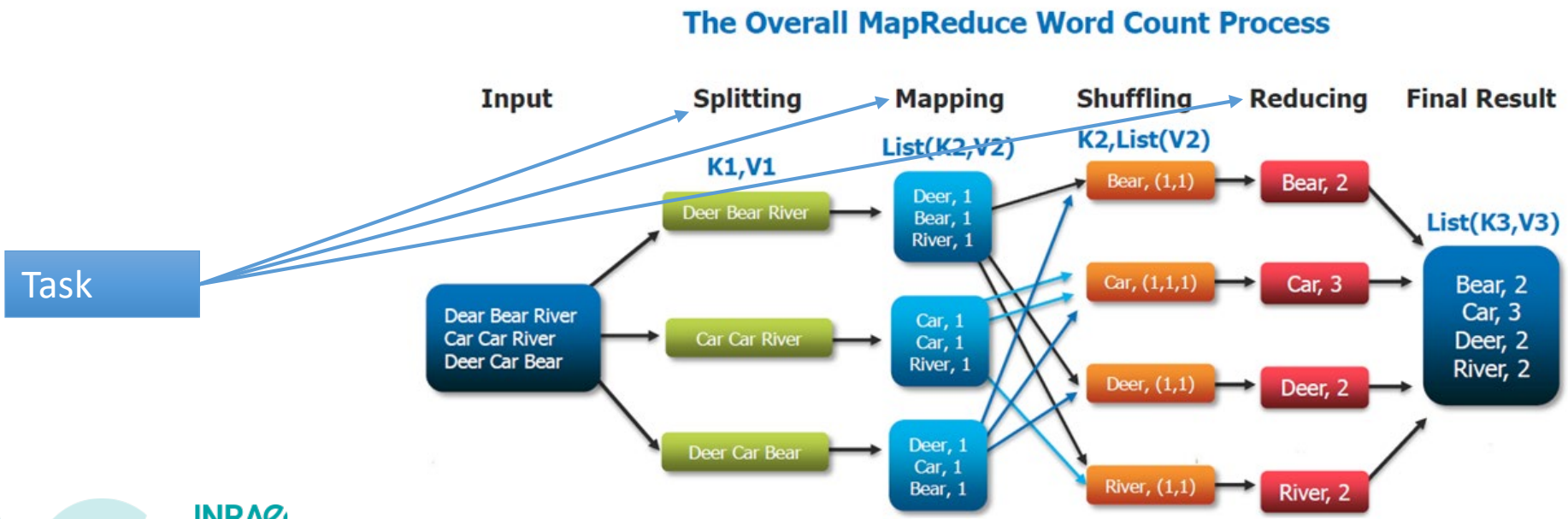
Spark

Anatomie d'un job map-reduce

```
val conf = new SparkConf().setAppName("Word Count Application")
val sc = new SparkContext(conf)

val count = sc.textFile(s"maprfs://spark-ics/user/llegrand/word_
    .flatMap(_.split(" "))
    .map(w => (w, 1))
    .reduceByKey(_ + _)

count.saveAsTextFile(s"maprfs://spark-ics/user/llegrand/word_cou
```



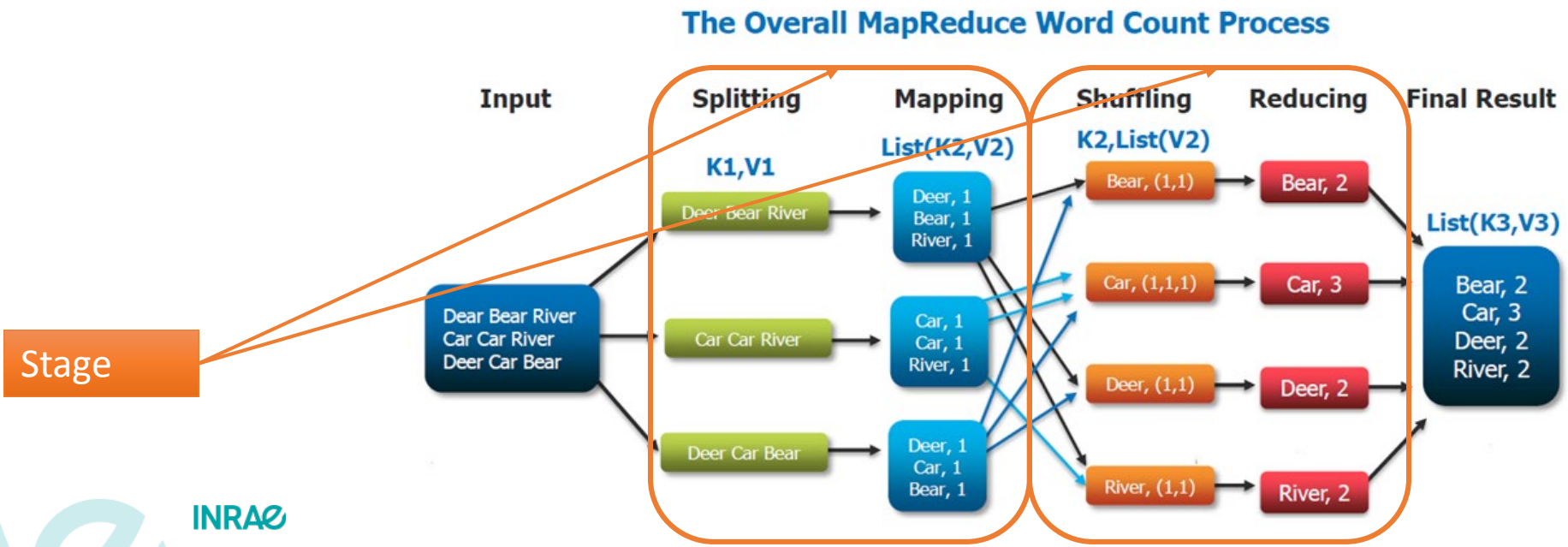
Spark

Anatomie d'un job map-reduce

```scala
val conf = new SparkConf().setAppName("Word Count Application")
val sc = new SparkContext(conf)

val count = sc.textFile(s"maprfs://spark-ics/user/llegrand/word_
    .flatMap(_.split(" "))
    .map(w => (w, 1))
    .reduceByKey(_ + _)

count.saveAsTextFile(s"maprfs://spark-ics/user/llegrand/word_cou
```



Spark

Anatomie d'un job map-reduce

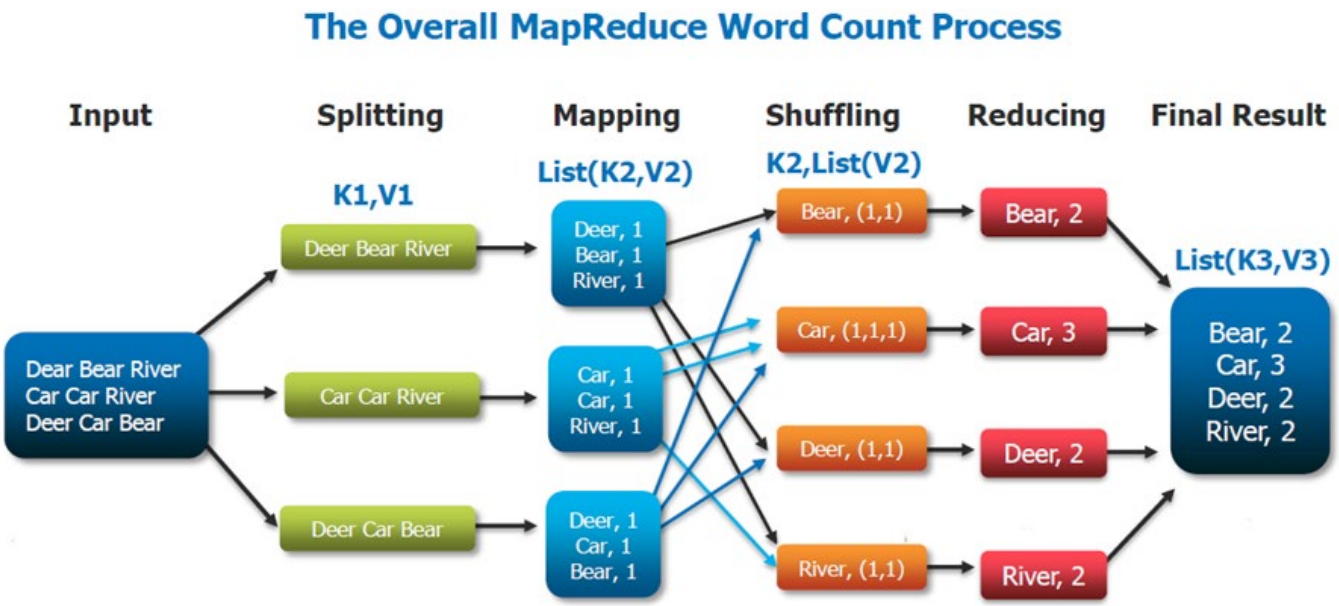
```
val conf = new SparkConf().setAppName("Word Count Application")
val sc = new SparkContext(conf)

val count = sc.textFile(s"maprfs://spark-ics/user/llegrand/word_
    .flatMap(_.split(" "))
    .map(w => (w, 1))
    .reduceByKey(_ + _)

count.saveAsTextFile(s"maprfs://spark-ics/user/llegrand/word_cou
```

Transformations

Action



Merci pour votre attention

Des Question ?