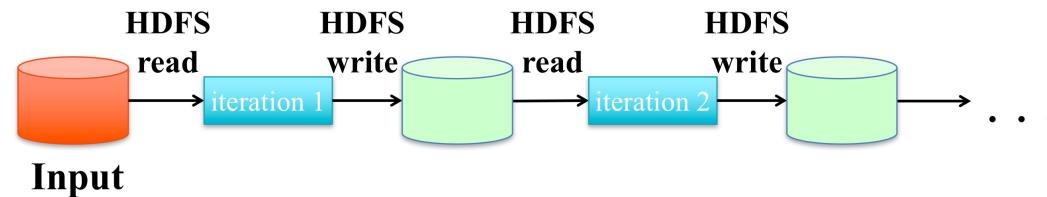




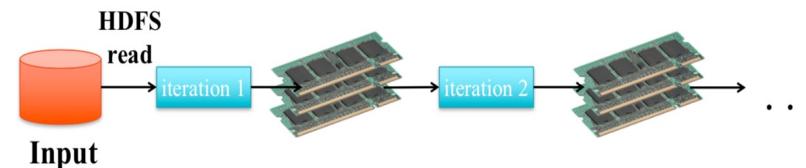
# spark

# Introduction à Spark

- Le moteur d'exécution MapReduce est de moins en moins utilisé du fait de sa lenteur. En effet, les résultats des phases intermédiaires sont écrites et relues à partir de HDFS. Des accès aux données qui coûtent et pénalisent les temps de traitement.



- En 2009, une équipe de recherche de Berkley va travailler sur le développement d'un gestionnaire de ressources appelé Mesos. Pour la validation de Mesos, ils vont implémenter un moteur d'exécution sous le nom de Spark. Celui-ci contrairement au MapReduce de HADOOP va conserver les résultats intermédiaires en mémoire ce qui en fera quelques années plus tard la référence en terme de moteur d'exécution.



- De nombreux projets de l'éco-système HADOOP vont abandonner le moteur MapReduce de HADOOP au profit de Spark comme Hive, Mahout et bien d'autres.
- Spark est un système distribué d'analyse de données « in-memory » et extensible.
- Il supporte plusieurs langages de programmation Java, Scala, Python.

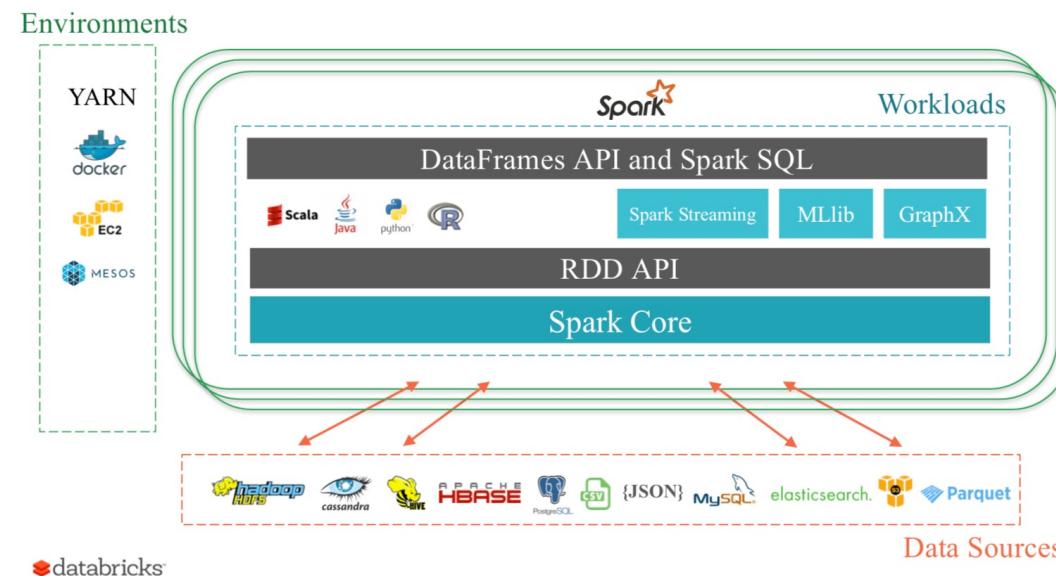
## Spark introduction



- Spark s'inspire fortement de HADOOP
  - L'utilisation de matériel commun pour le traitement **mais ne gère pas le stockage des données**
  - Les pannes matérielles sont gérées au niveau logiciel
  - Exécuter le code applicatif au plus près des données
  - Séparer la logique du traitement des données de la logique de distribution du calcul
  - Intègre son propre cluster manager 'Spark standalone' lequel est interchangeable avec YARN / MESOS
- Les améliorations apportées par Spark
  - Développé en SCALA, langage basé sur une JVM.
    - 100 lignes de code JAVA peut être traduit en une dizaine de lignes en SCALA
  - Hérite des avantages de la programmation fonctionnelle
    - Le calcul est évalué comme une fonction mathématique
    - Le développement d'applications parallélisées est plus facile
  - Concept de Resilient Distributed Dataset
- Spark est adapté pour les traitements batch (par lot) mais ne convient pour les systèmes en temps réel. On peut classifier les applications en temps réel comme des applications qui traitent les données en flux tendu avec un temps de traitement inférieur à la dizaine de milli-seconde. Vous pouvez aussi entendre de traitements pseudo temps-réel où le temps de traitement est compris entre 10 ms et 50 ms. Ce type de traitement peut être géré par Spark avec ses fonctionnalités de streaming.

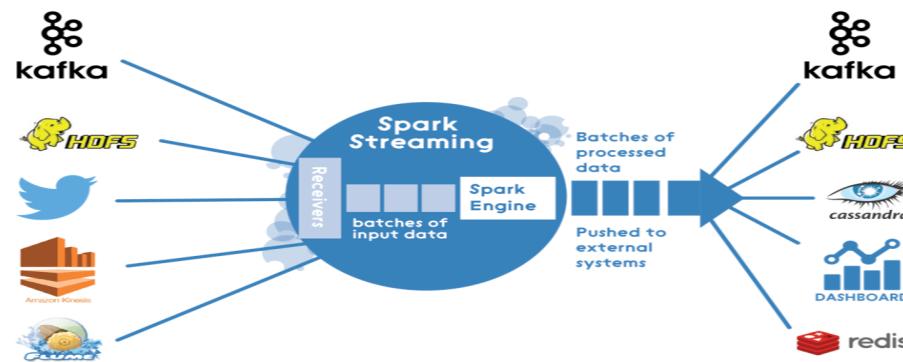
# Spark introduction

- L'idée de Spark est d'appliquer les mêmes opérations à tous les éléments d'un jeu de données.
- Bien entendu, au même titre que MapReduce, Spark gère la tolérance aux pannes et peut resoumettre une tâche qui échoue.
- Spark couvre une large variété de traitement :
  - Batch programs
  - Interactive SQL
  - Streaming
  - Graph processing



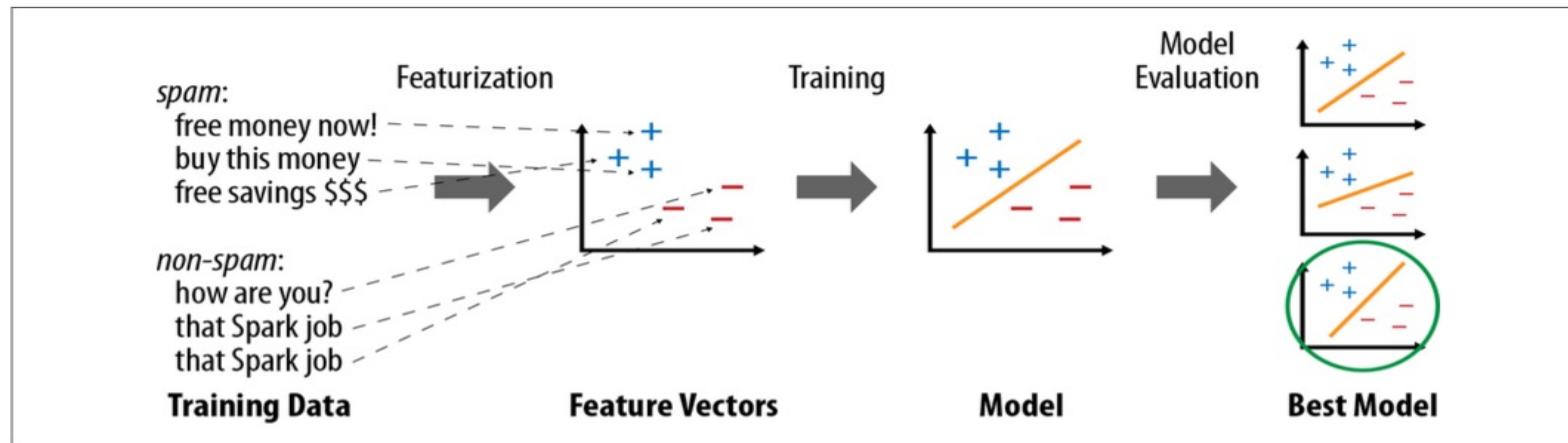
## Fonctionnalités Spark : Streaming

- Les traitements par flux permettent de traiter les données par un intervalle régulier. Par exemple, Spark va collecter les données provenant de sources de données comme Kafka , une base NoSQL, des réseaux sociaux toutes les 5s et appliquer un traitement qui peut être redirigé dans une source externe comme HDFS, une base de données NoSQL ...
- Le traitement est continu. En cas de panne ou d'arrêt, il est possible de configurer Spark pour mettre en place des points de contrôle.



# Fonctionnalités Spark : Machine learning

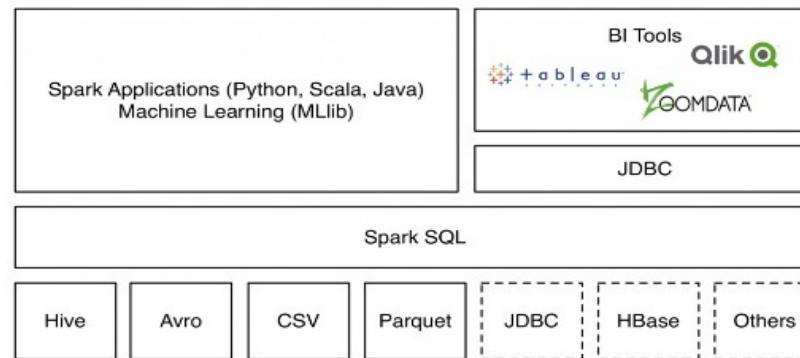
- Le module Spark MLlib offre des fonctionnalités d'apprentissage dans plusieurs domaines. La documentation disponible sur le site Web de Spark présente les types de données utilisés (par exemple, les vecteurs et la structure LabeledPoint). Ce module offre des fonctionnalités qui incluent:
  - Statistics
  - Classification
  - Regression
  - Collaborative Filtering
  - Clustering
  - Dimensionality Reduction
- Notez qu'une librairie spécifique au machine learning a été développé sous le nom d'Apache Mahout laquelle est compatible avec Spark.



## Fonctionnalités Spark : SQL interactif

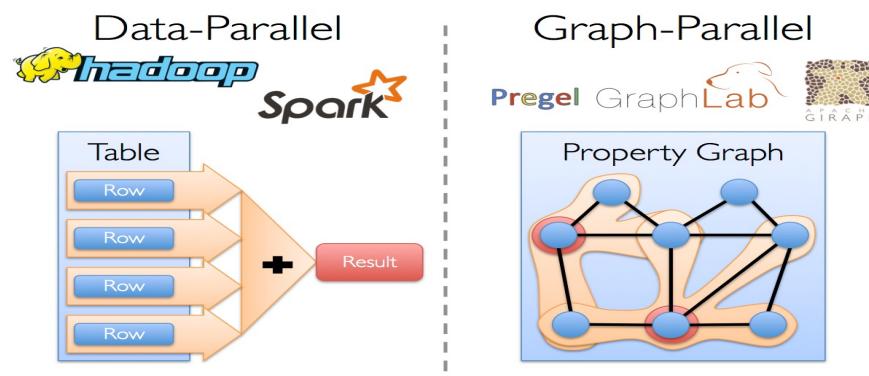
- Depuis la version 1.3, Spark intègre la notion de datafram afin que les données puissent être traitées sous forme tabulaire et que des fonctions tabulaires (telles que select, filter, groupBy) puissent être utilisées pour traiter les données.
- Le module Spark SQL est compatible avec une large variété de formats Parquet, Avro, ORC, JSON, CSV ...
- Spark SQL est une interface qui permet d'exécuter du SQL quelque soit le format du fichier.
- La diversité des formats permet à Spark d'être flexible mais surtout de faciliter l'exploitation des données.
- Spark SQL permet aussi de s'interfacer avec Hive et des bases de données

### Data source API

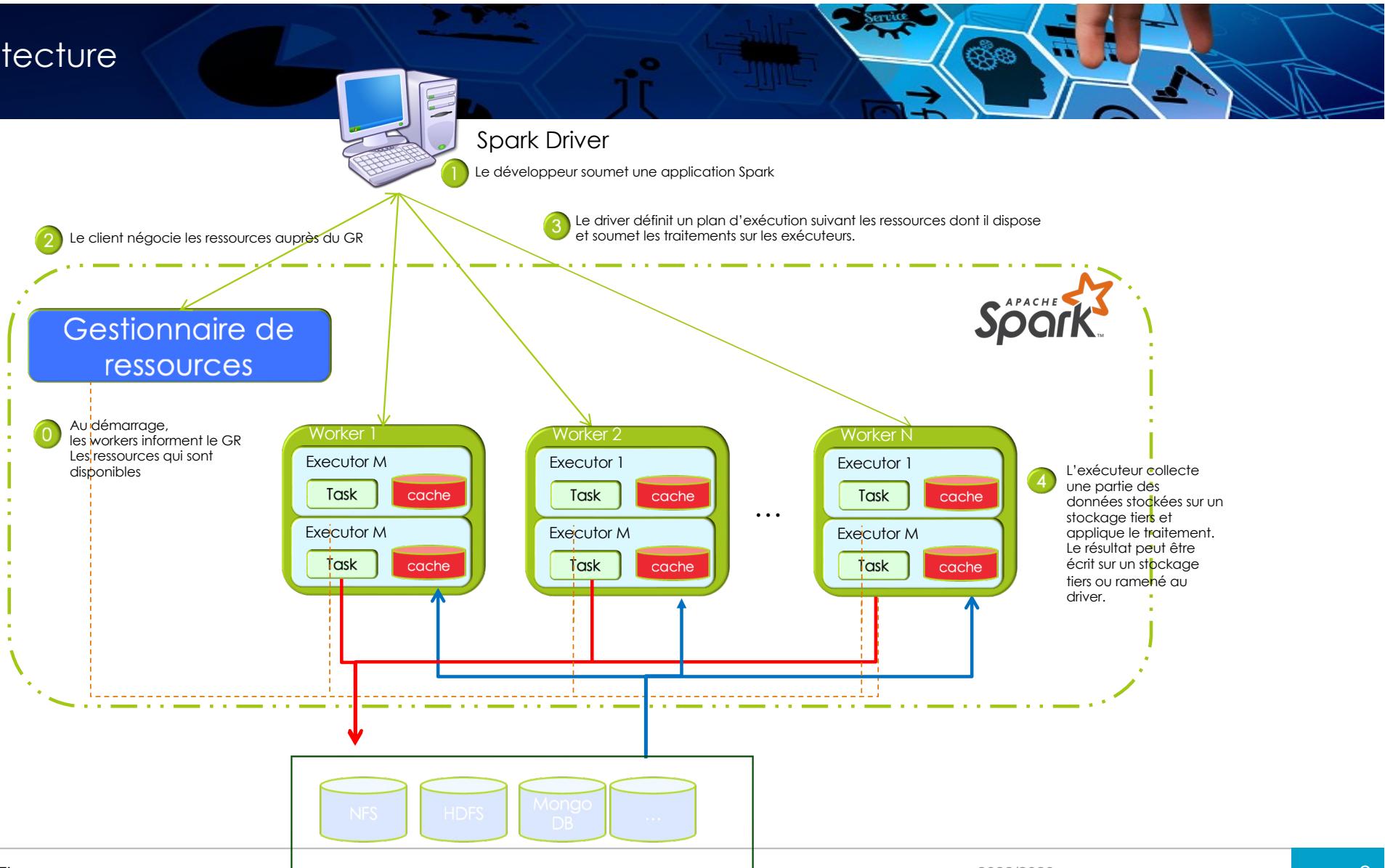


## Fonctionnalités Spark : SQL interactif

- Spark intègre un module GraphX qui vous permet de traiter vos données sous forme de graphes en représentant celles-ci sous forme d'arcs et de sommets comme avec Neo4J.
- GraphX est décrit comme une solution permettant de traiter de gros volume de données mais cela reste à confirmer en pratique. En effet, les données ne sont pas structurées comme dans Neo4J sous forme de graphe ce qui rend les performances moindres.

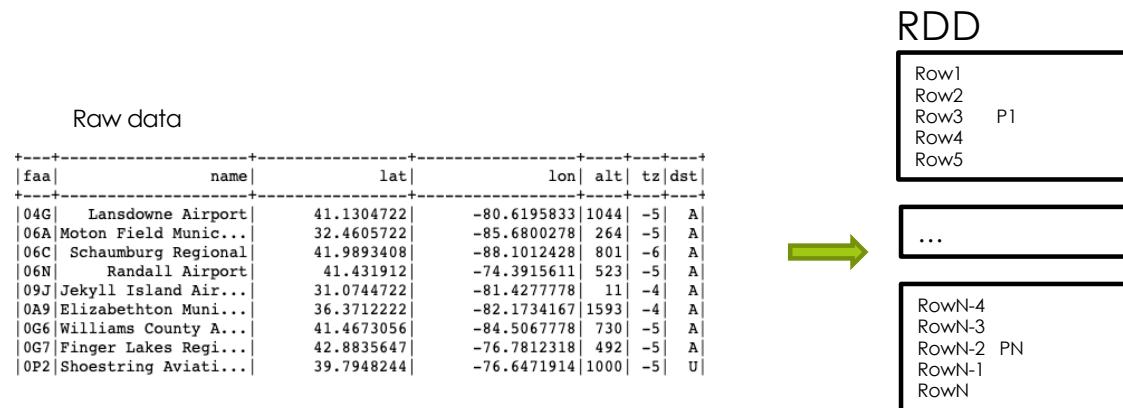


# Spark architecture



# Resilient Distributed Dataset RDD

- Spark est conçu autour de la notion de Resilient Distributed Dataset RDD (jeux de données résilient et distribués).
- Le RDD représente vos données comme une collection qui peut être opérée en parallèle.
- Un RDD est une structure de données immutable. Une fois créé, il ne peut pas être modifié. Une opération qui modifie un RDD retourne un nouvel RDD.
- Spark s'appuie sur la notion de RDD pour distribuer les traitements. Le RDD est décomposé en partition ou chacune d'entre elle sera traitée indépendamment et en parallèle sur les workers.
- Notez qu'il est possible d'avoir une correspondance entre les partitions du RDD et les blocs HDFS. Par exemple, HDFS stock les données en block de 128Mo et Spark va s'appuyer de ses échanges avec le Namenode pour établir une correspondance 1-1 entre les partitions du RDD et la tailles des blocs HDFS.



## Resilient Distributed Dataset : RDD

- Les opérations sur les RDD sont classées en 2 catégories
  - Les transformations
    - <https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations>
  - Les actions
    - <https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions>
- Une transformation exécute un traitement sur un RDD et retourne un nouveau RDD.
  - Dans l'exemple ci-dessous le RDD dataSrc représente le fichier source.
  - L'application de la fonction « len » dans la phase map va retourner un nouveau RDD contenant uniquement la longueur de chaque ligne.
  - La phase reduce quand à elle va appliquer une fonction d'addition sur l'ensemble des éléments retournés par la phase map.
  - Notez que Spark est paresseux (lazyness), c'est à dire qu'aucun traitement n'est exécuté lors des transformations.
  - Il existe de nombreuses transformations : map(), flatMap(), filter(), sample(), intersection() ...
- Une action est une méthode du RDD qui retourne une valeur.
  - L'appel de l'action reduce() sera l'évènement qui lancera les traitements
  - Toutes les transformations dont héritent le RDD à l'origine de l'action seront planifiées par l'optimiseur de Spark (le driver).
  - Un plan d'exécution sera alors établi sous forme de tâches lesquels seront distribuées sur l'ensemble des exécuteurs.
  - Il existe de nombreuses actions : reduce(), collect(), count(), take() ...

## Resilient Distributed Dataset : RDD

Voici, un exemple d'un traitement Map et Reduce avec Spark.

```
dataSrc = sc.textFile("mydata.txt")
lineLengths = dataSrc.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

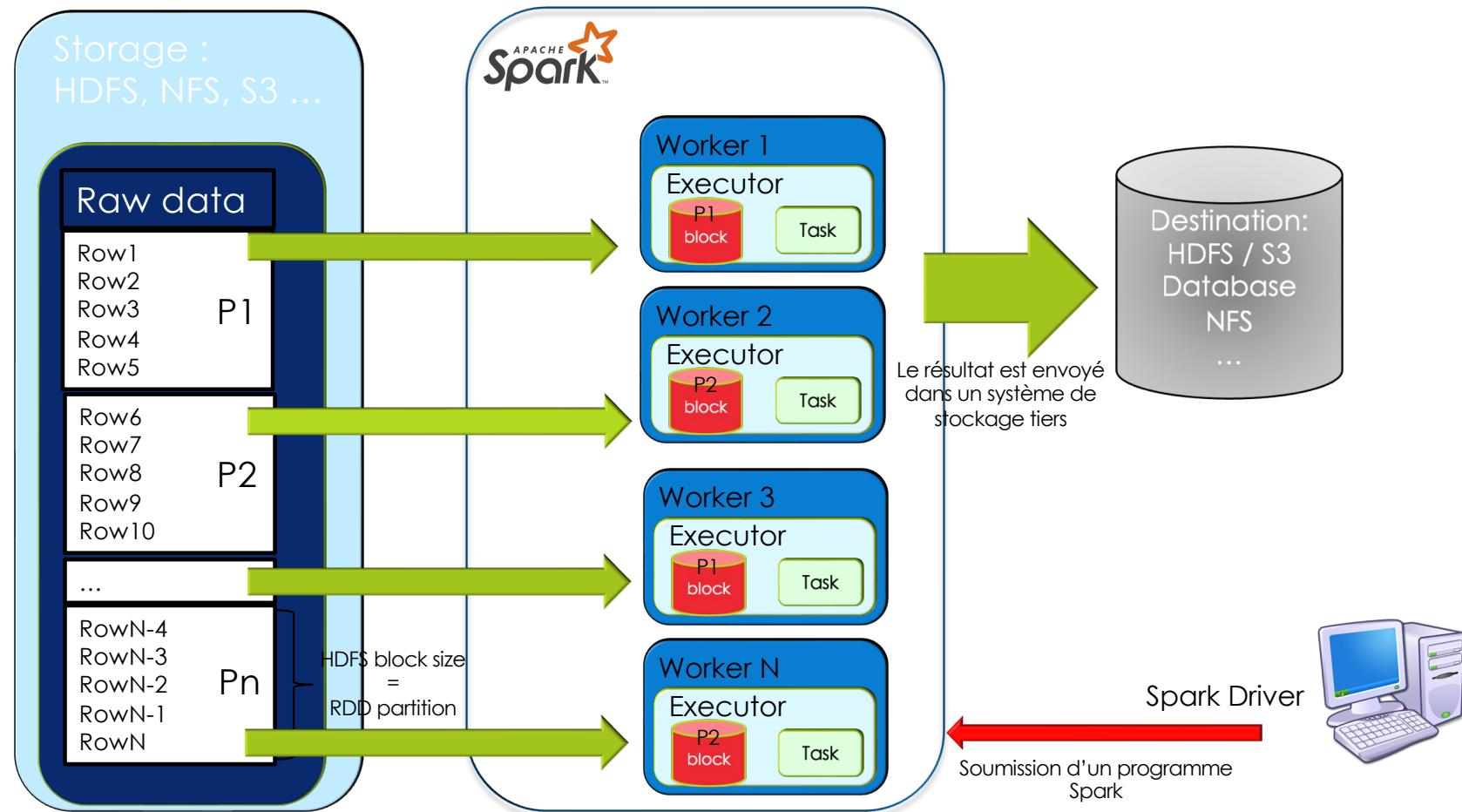
Que réalise ce programme ?

Le programme suivant compte le nombre de caractères dans un fichier.  
Bien entendu, ce programme est décorrélé de la taille des données à traiter.  
Ce programme fonctionnera aussi bien sur un cluster de 3 machines que sur N machines.

## Resilient Distributed Dataset : Pairs RDD

- Avec le moteur MapReduce, nous avons vu qu'il pouvait être pratique de travailler avec des tuples.
- Spark permet de travailler avec des « pairs RDD » , lesquels sont des RDD mais sous forme de tuple (clé, valeur). Les « pairs RDD » sont couramment utilisés pour effectuer des agrégations.
- Spark fournit des opérations spéciales sur les RDD contenant des paires clé / valeur.
- Par exemple, les « pairs RDD » ont une méthode `ReduceByKey()` qui peut agréger les données séparément pour chaque clé, et une méthode `join()` qui peut fusionner deux RDD ensemble en regroupant les éléments avec la même clé.
- Il est courant d'extraire des champs d'un RDD (représentant, par exemple, une heure d'événement, un ID client ou un autre identifiant) et d'utiliser ces champs comme clés dans des opérations d'agrégation.
- Pour les actions et transformations liées au pairs RDD, reportez vous à la documentation :
  - <https://spark.apache.org/docs/latest/rdd-programming-guide.html#working-with-key-value-pairs>
- Exemple :
  - `pairRDD1 : [(1, 2), (3, 4), (3, 6)]`
  - `pairRDD2 : [(3, 9)]`
  - transformation de pair RDD avec `groupByKey` :
    - `RDD1.groupByKey() -> [(1, [2]), (3, [4, 6])]`
  - transformation de pairs RDD avec `join` :
    - `RDD1.join(RDD2) -> [(3, (4, 9)), (3, (6, 9))]`

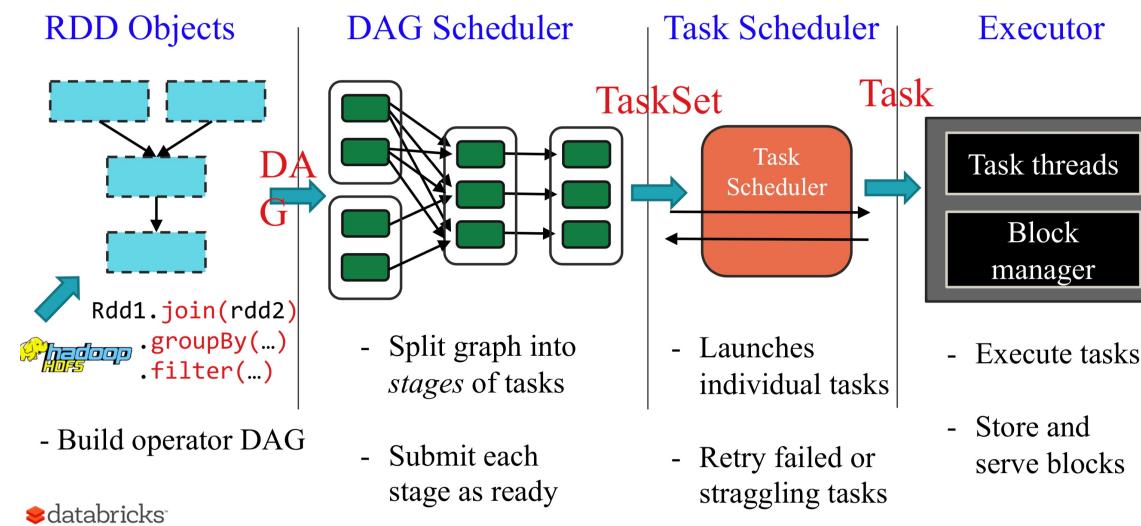
# Resilient Distributed Dataset RDD



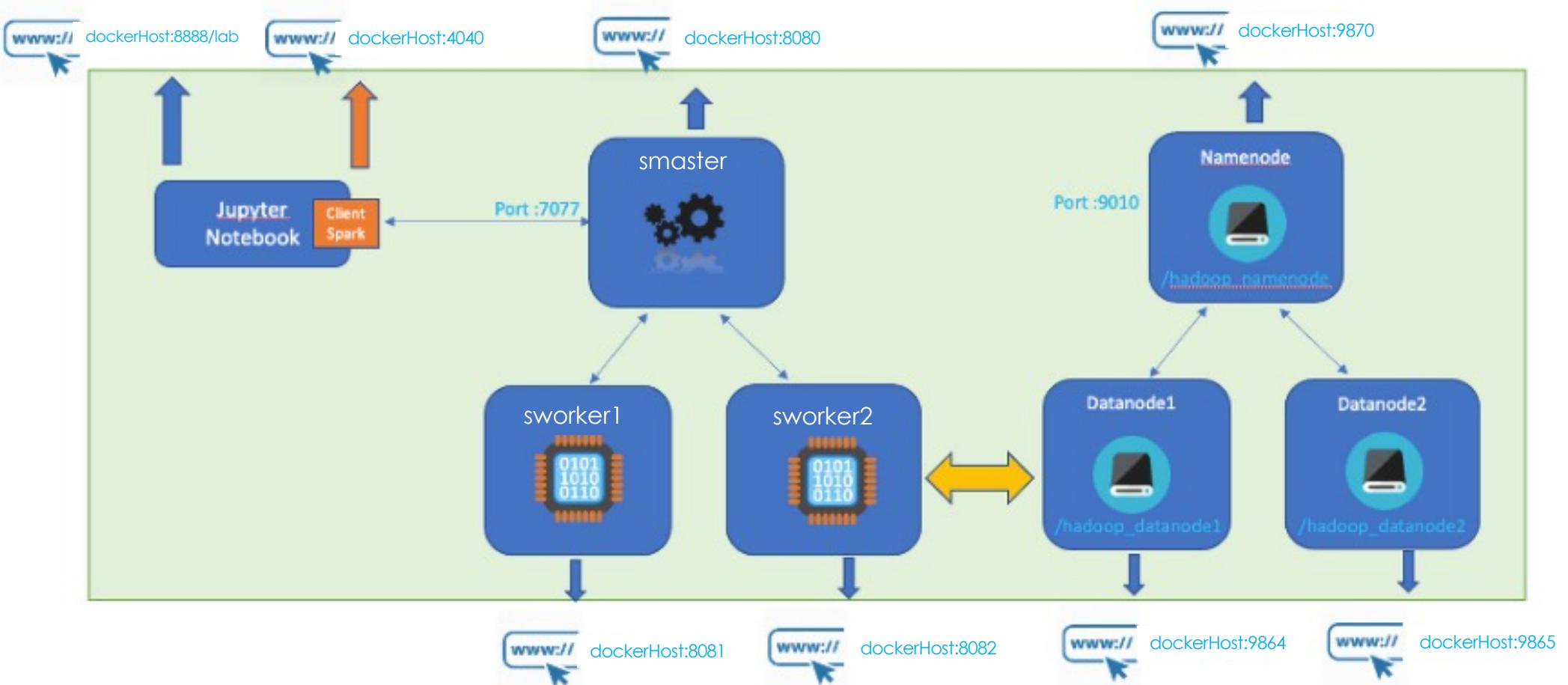
# Resilient Distributed Dataset RDD

- Voici comment Spark va interpréter votre programme et décharger le développeur de tous les aspects liés à la distribution du calcul.
  - Le plan d'exécution est appelé le Directed Acyclic Graph.
  - DAG est un graphe où les sommets représentent le RDD et les arcs les opérations qui doivent être appliquées.
  - C'est à partir du DAG que l'ensemble des tâches seront réparties sur les exécuteurs.

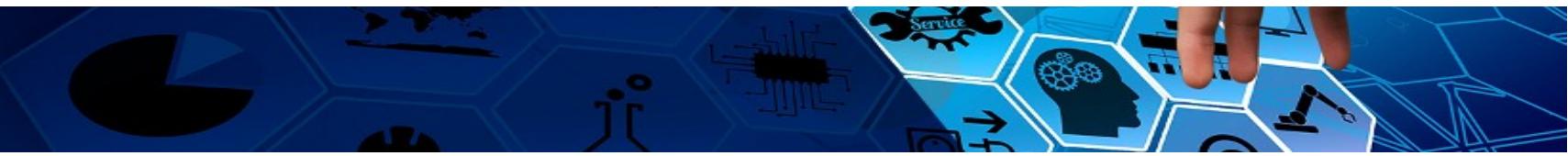
## Scheduling Process



# TP : chap6\_Spark\_introduction



# Les Data Frames



- En 2015, Spark va s'inspirer de la notion de Data Frame en R et Python (Panda) pour fournir une nouvelle API plus simple à utiliser que les RDD.
- Les Data Frames offrent un haut niveau d'abstraction qui facilitent la programmation et améliorent les performances d'exécution quelque soit le langage de programmation.
  - L'utilisation des RDD en JAVA, Python ou Scala oblige Spark à convertir le code en java byte code. Dans le cas où vous programmez en python, Spark va utiliser la librairie Py4J pour convertir votre code en JAVA et ensuite générer le byte code. Ces phases de conversions ne sont pas optimales et se traduisent par des pertes de performances.
- On écrit de moins en moins de code.

**hadoop**

```
private IntWritable one = new IntWritable(1);
private IntWritable output =new IntWritable();
protected void map(LongWritable key,
                   Text value,
                   Context context) {
    String[] fields = value.split("\t");
    output.set(Integer.parseInt(fields[1]));
    context.write(one, output);
}

-----
IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable();

protected void reduce(IntWritable key,
                      Iterable<IntWritable> values,
                      Context context) {
    int sum = 0;
    int count = 0;
    for (IntWritable value: values) {
        sum += value.get();
        count++;
    }
    average.set(sum / (double) count);
    context.write(key, average);
}
```

**Spark Using RDD**

```
rdd = sc.textfile(...).map(_.split(" "))
rdd.map { x => (x(0), (x(1).toFloat, 1)) }.
reduceByKey { case ((num1, count1), (num2, count2)) =>
  (num1 + num2, count1 + count2)
}.
map { case (key, (num, count)) => (key, num / count) } .
collect()

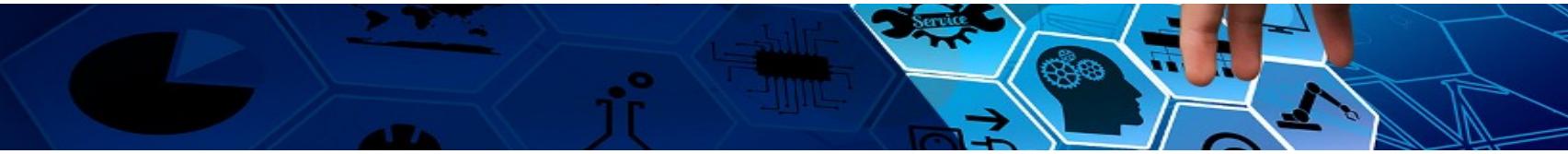
rdd = sc.textfile(...).map(lambda s: s.split())
rdd.map(lambda x: (x[0], (float(x[1]), 1))).\
reduceByKey(lambda t1, t2: (t1[0] + t2[0], t1[1] + t2[1])).\
map(lambda t: (t[0], t[1][0] / t[1][1])).\
collect()
```

**Using DataFrames**

```
import org.apache.spark.sql.functions._

val df = rdd.map(a => (a(0), a(1))).toDF("key", "value")
df.groupBy("key")
  .agg(avg("value"))
.collect()
```

## Les Data Frames



- Comme les RDD, les DataFrames sont paresseux. Les transformations contribuent au plan de requête, mais elles n'exécutent rien.
- Les actions provoquent l'exécution de la requête.

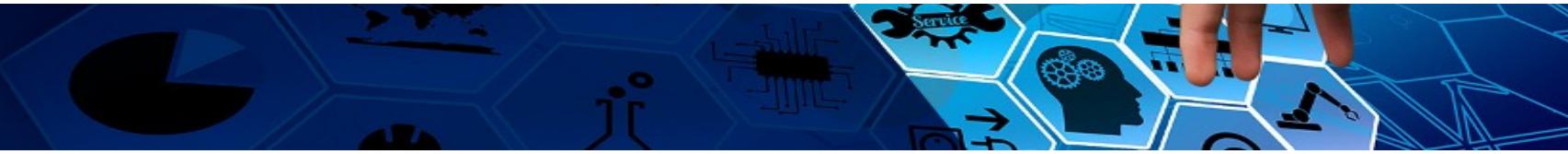
Transformation	Action
filter	Count
select	Collect
drop	Show
intersect	Head
join	take
...	..

## Les Data Frames : exemple d'utilisation

- Création d'un DataFrame qui contient seulement les utilisateurs de moins de 18 ans.
  - `young = users.filter(users["age"] < 21)`
- Un syntaxe alternative similaire à la syntaxe Pandas en python
  - `young = users[users.age < 21]`
- Ajouter un an à tous les utilisateurs :
  - `young.select(young["name"], young["age"] + 1)`
- Compter le nombre d'utilisateur par genre
  - `young.groupBy("gender").count()`
- Joindre les utilisateurs de moins 18 ans à un autre dataframe logs via le champs userId:
  - `young.join(log, logs["userId"] == users["userId"], "left_outer")`

## Les Data Frames et Spark SQL

- Les DataFrames sont fondamentalement liés à Spark SQL.
- Spark SQL fournit une interface *SQL-like*.
- Tout ce que vous pouvez faire avec Spark SQL, vous pouvez le faire dans DataFrames et réciproquement.
- Spark SQL vous permet de manipuler des données distribuées avec des requêtes SQL. Actuellement, deux dialectes SQL sont pris en charge.
  - Si vous utilisez un Spark SQLContext, le seul dialecte pris en charge est «sql», un riche sous-ensemble de SQL 92.
  - Si vous utilisez un contexte Hive, le dialecte par défaut est "Hiveql", correspondant au dialecte SQL de Hive. Notez que « Hiveql » est un dialecte plus riche que «sql».
- Les requêtes SQL sont exécutées via un SQLContext ou HiveContext, en utilisant la méthode sql () .
- Vous pouvez mélanger des méthodes DataFrame et des requêtes SQL dans le même code.
- Pour utiliser SQL, il faut définir le nom de la table :
  - Soit en utilisant les tables persistantes définies dans Hive.
  - Soit en créant un nom de table qui sera un alias vers un Data Frame avec la méthode registerTempTable()
- Que signifie exactement «exécution de la requête»? Ça veut dire:
  - Spark lance une lecture distribuée de la source de données.
  - Les données transitent par les transformations (les RDD résultant du plan d'exécution).
  - Le résultat de l'action est renvoyé au driver.



## ■ Création d'un DataFrame avec un objet SQLContext

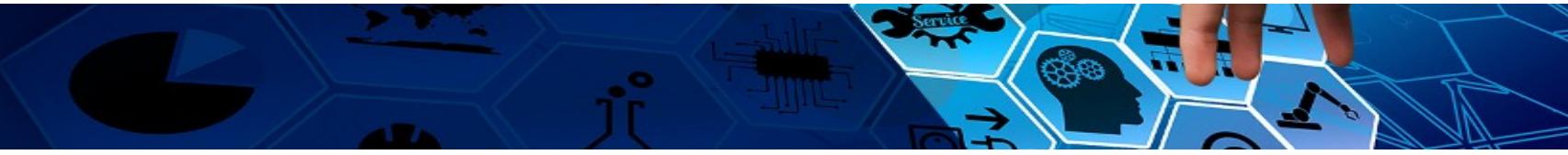
- Dans le shell Spark Scala ou pySpark, vous avez un SQLContext disponible automatiquement, sous la forme sqlContext.
- Dans une application, vous pouvez facilement en créer un vous-même à partir d'un SparkContext.
- L'API de source de données DataFrame est cohérent, quel que soit le format de données. Ouvrir une source de données fonctionne à peu près de la même manière, quoi qu'il arrive.

```
# Les 4 lignes suivantes indiquent la configuration du cluster Spark lesquels ne
# sont pas nécessaires avec le shell pySpark.
```

```
from pyspark import SparkContext, SparkConf
conf = SparkConf().setAppName(appName).setMaster(master) sc =
SparkContext(conf=conf)
sqlContext = SQLContext(sc)
```

```
# Création d'un DF à partir de données JSON et un autre à partir de données
# parquet.
```

```
df = sqlContext.read.parquet("/path/to/data.parquet")
df2 = sqlContext.read.json("/path/to/data.json")
```



- Les Data Frames ont une structure de table qui est décrite par un schéma.
- Spark peut inférer un schema à partir d'un fichier JSON ou Parquet.
- Pour voir le schema d'un DataFrame utilisez la méthode printSchema()

```
df.printSchema() root
| -- firstName: string (nullable = true)
| -- lastName: string (nullable = true)
| -- gender: string (nullable = true)
| -- age: integer (nullable = false)
```

- Bien entendu, il est possible de spécifier le schéma manuellement

```
Claire,McBride,F,23
Abigail,Cottrell,F,75
José,Rivera,M,59
Ravi,Dasgupta,M,25
```

```
from pyspark.sql import Row
rdd = sc.textFile("people.csv")
Person = Row('first_name', 'last_name', 'gender', 'age')
def line_to_person(line):
    cells = line.split(",")
    cells[3] = int(cells[3])
    return Person(*cells)

peopleRDD = rdd.map(line_to_person)
df = peopleRDD.toDF()
```

TP : SPARK\_TP01\_DF, SPARK\_TP02\_DF, SPARK\_TP03\_DF, SPARK\_TP04\_DF

# TP DATAFRAME

The screenshot shows the JupyterLab interface. On the left, there is a file browser with a sidebar containing icons for running, editing, viewing, and sharing files. The main area displays a list of Jupyter notebooks in a folder named 'HADOOP/SPARK'. The notebook 'SPARK\_DF\_01.ipynb' is currently selected. On the right, there is a notebook editor window titled 'SPARK\_DF\_01.ipynb' which contains the following content:

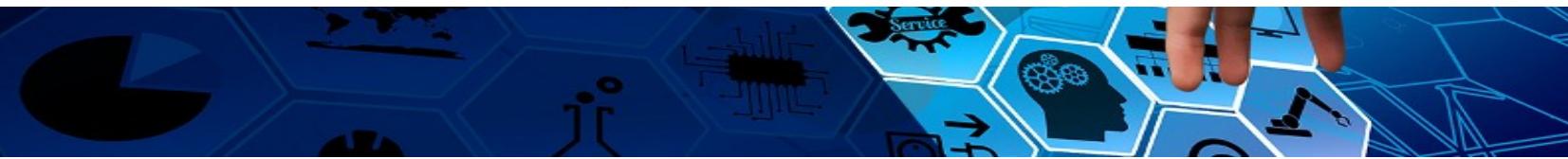
## Introduction au Dataframe Spark

Les Dataframes sont apparus à partir de la version 2.x. Les DataFrames est une couche d'abstraction des RDD, qui présentent les données comme des tables de base de données sans se préoccuper de la taille des données. Les DataFrames offrent de nombreux avantages:

- Une syntaxe beaucoup plus simple.
- Possibilité d'utiliser SQL directement dans la trame de données.
- Les opérations sont automatiquement réparties via les RDD.

Si vous avez utilisé R ou même la bibliothèque pandas avec Python, vous connaissez probablement déjà le concept des DataFrames. Même si Spark supporte plusieurs

## HADOOP Architecture



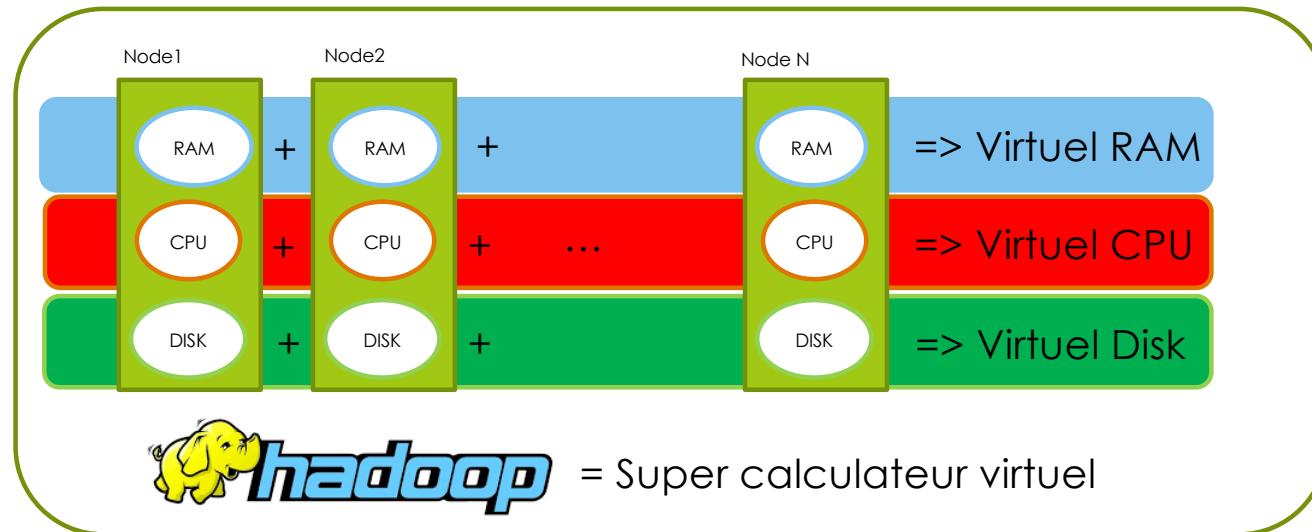
- HADOOP est une plateforme open-source basée sur une architecture lambda offrant des fonctionnalités de calcul distribué, d'extensibilité et de tolérance aux pannes.
- Il est écrit en java mais il supporte différents langages C, C++, Python ...
- Il a été conçu pour gérer de gros volumes de données ( Peta octets et même Hexa octets).
- HADOOP est très économique, de gros volumes de données peuvent être traités à partir de matériel populaire.
- Selon la loi de Moore, n'importe quel matériel peut tomber en panne et si il le peut alors il tombera en panne. HADOOP intègre des mécanismes de haute disponibilité pour garantir le service.
- HADOOP est un « cluster » de machines où le nombre de machines n'a pas de limite.
- HADOOP a séduit de nombreuses entreprises et communautés.
- De nombreux logiciels vont naître autour de cette plateforme et diversifier les traitements possibles.

## Les distributions HADOOP

- Officiellement, HADOOP est la distribution offerte par la fondation Apache.
- Plusieurs compagnies et organisations vendent leur propre distribution basée sur la distribution Apache
  - Amazon Web Services : Amazon utilise HADOOP avec leur solution de stockage EC2.
  - Cloudera et Hortonworks : Des compagnies qui participent activement au développement de l'éco-système HADOOP.
  - BigInsights Enterprise Edition, solution fournie par IBM
  - Pivotal HD, distribution implémentée par Pivotal pour les architectures Kappa.
  - Serengeti, solution développé par VMWare
  - ...

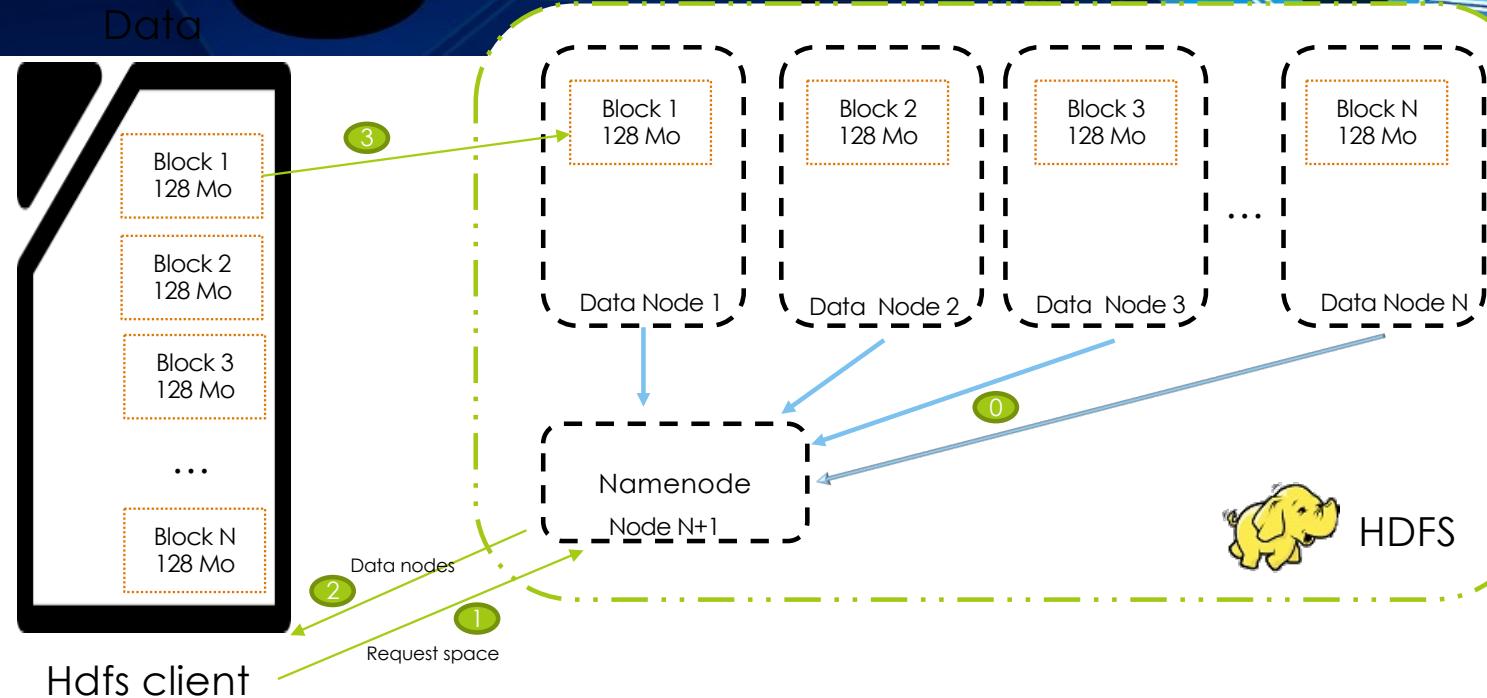
## HADOOP architecture

- Pour traiter de gros volumes de données, HADOOP virtualise l'ensemble des ressources de calcul, de mémoire et de stockage de plusieurs machines en un super calculateur virtuel.



- A partir de 2.0, HADOOP s'appuie sur 3 composants.
  - Un stockage distribué HDFS
  - Un moteur de calcul distribué
  - Un gestionnaire de ressource

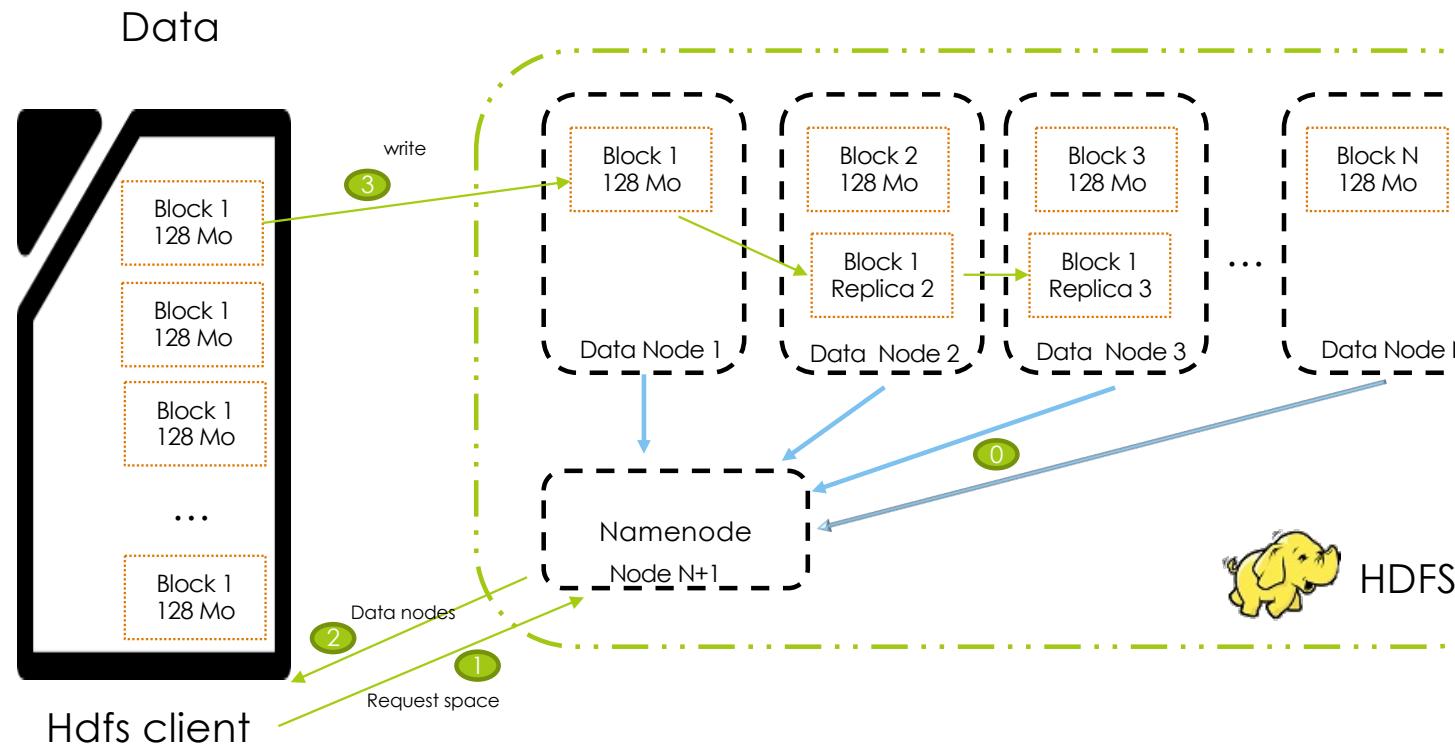
## HADOOP Distributed File System HDFS : exemple écriture d'un fichier



- 0 Au démarrage, tous les Datanodes notifient le Namenode de la liste des blocks qu'ils contiennent.
- 1 Pour charger des données dans HDFS, le client contacte le Namenode et lui demande de créer un fichier dans l'arborescence du système de fichiers.
- 2 Le Namenode alloue un bloc de données (ex 128 Mo) et retourne au client une liste de blocs disponibles sur lesquels les données peuvent être stockées.
- 3 Le client contact le Datanode recommandé par le Namenode et lui envoie ses données.
  - Si le fichier de données est plus grand que 128Mo alors le client HDFS renouvellera une nouvelle demande de bloc au Namenode et procédera à l'écriture comme décrit précédemment.

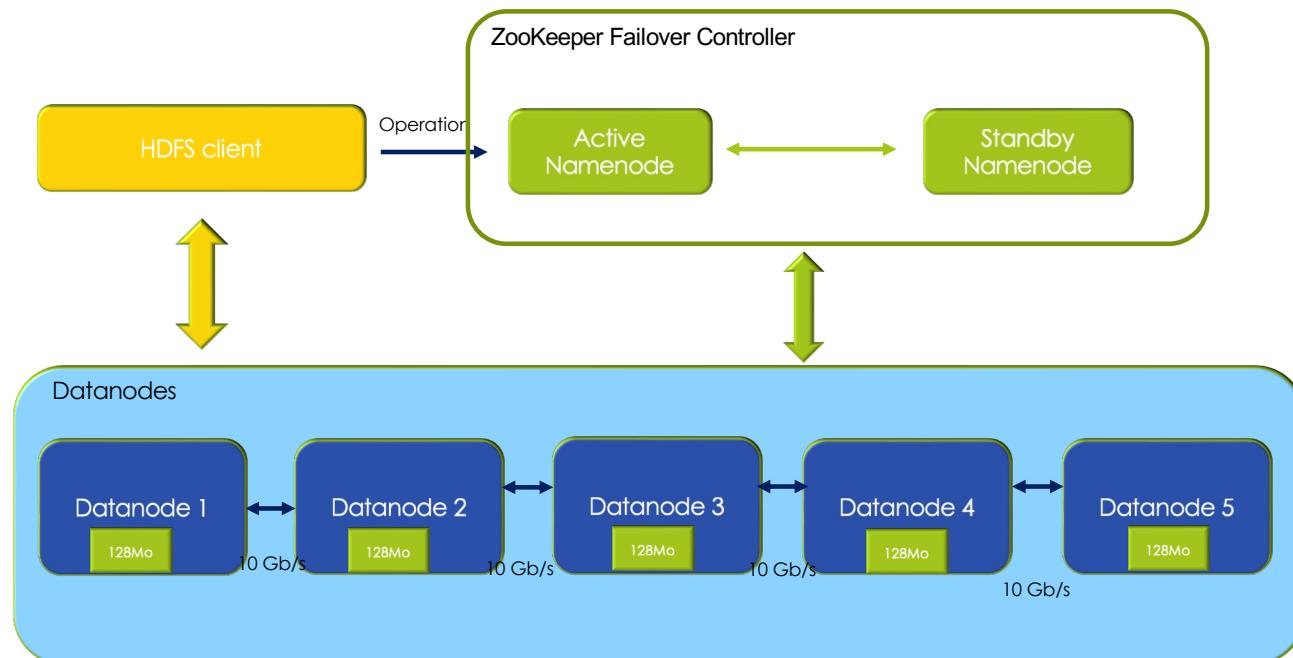
## HADOOP Distributed File System HDFS : exemple écriture d'un fichier

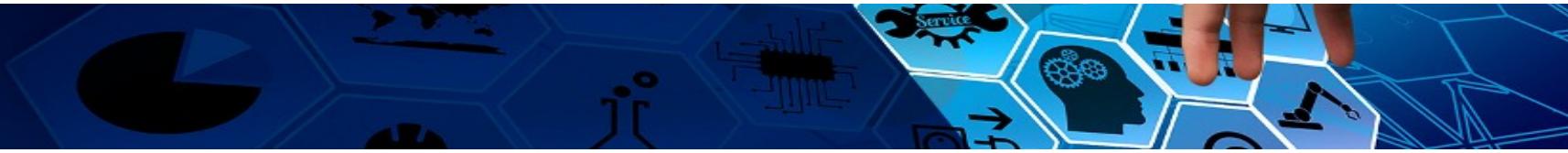
- Pour assurer la disponibilité des données HDFS permet de répliquer les blocs de données entre les Datanodes (par défaut 3).
- La réPLICATION des données s'opèrent lorsque que la première copie du bloc de données est envoyé sur le premier Datanode. Chaque Datanode qui stockeront un réPLICA, liste définie par le Namenode, recevront le bloc à partir du Datanode précédent.



# HADOOP Distributed File System HDFS : Haute disponibilité

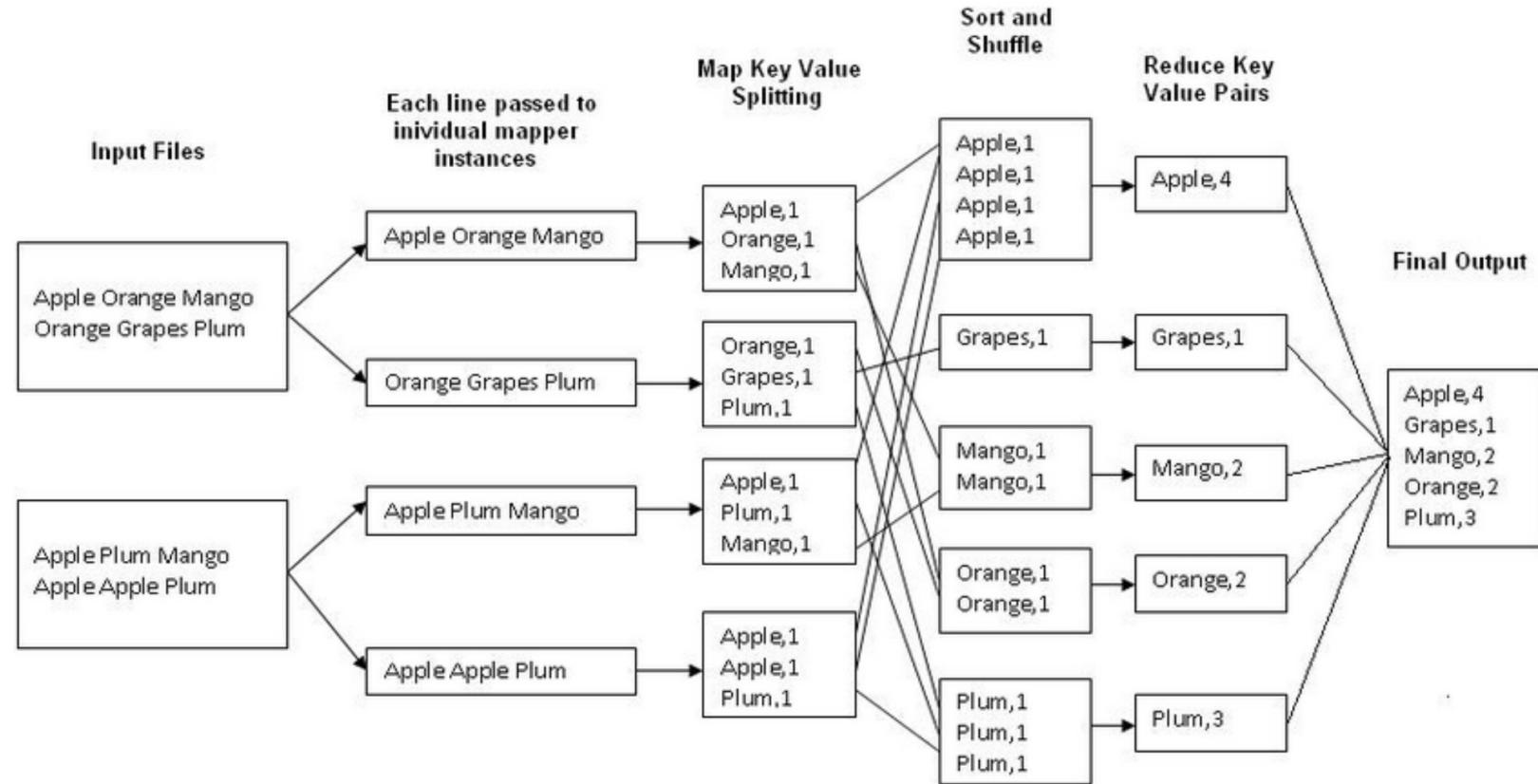
- Toutes les écritures dans HDFS passent par le Namenode.
- Si le Namenode tombe en panne, nous perdons l'accès aux données aussi bien en écriture qu'en lecture.
- Dans HADOOP 2.0, il est possible d'exécuter 2 Namenodes simultanément pour fournir la haute disponibilité, avec un actif et le second en passif.
- La solution de haute disponibilité repose sur Apache ZooKeeper. Zookeeper a le rôle d'un coordinateur qui supervise l'état du Namenode actif et gère en cas de panne le mécanisme de basculement vers le Namenode passif.





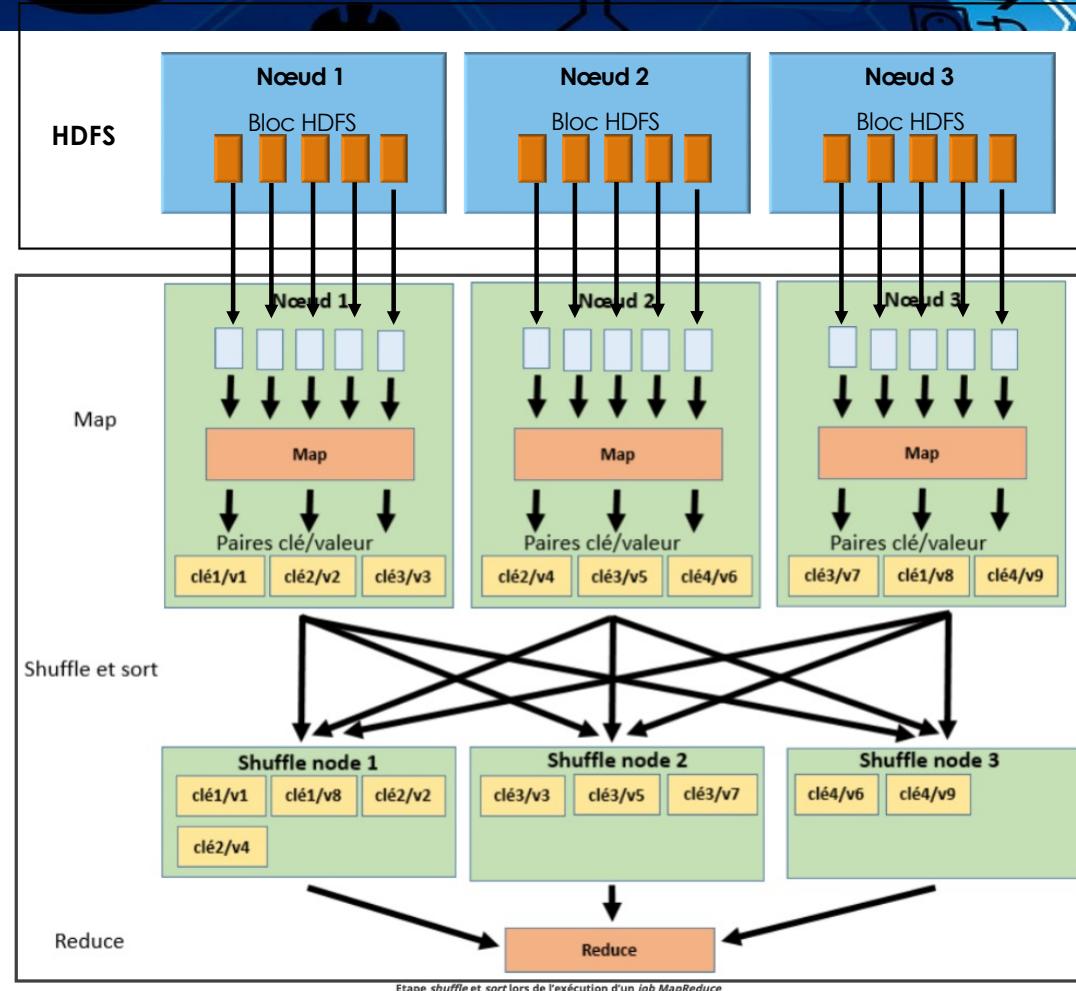
500 MB

# Moteur d'exécution : MapReduce

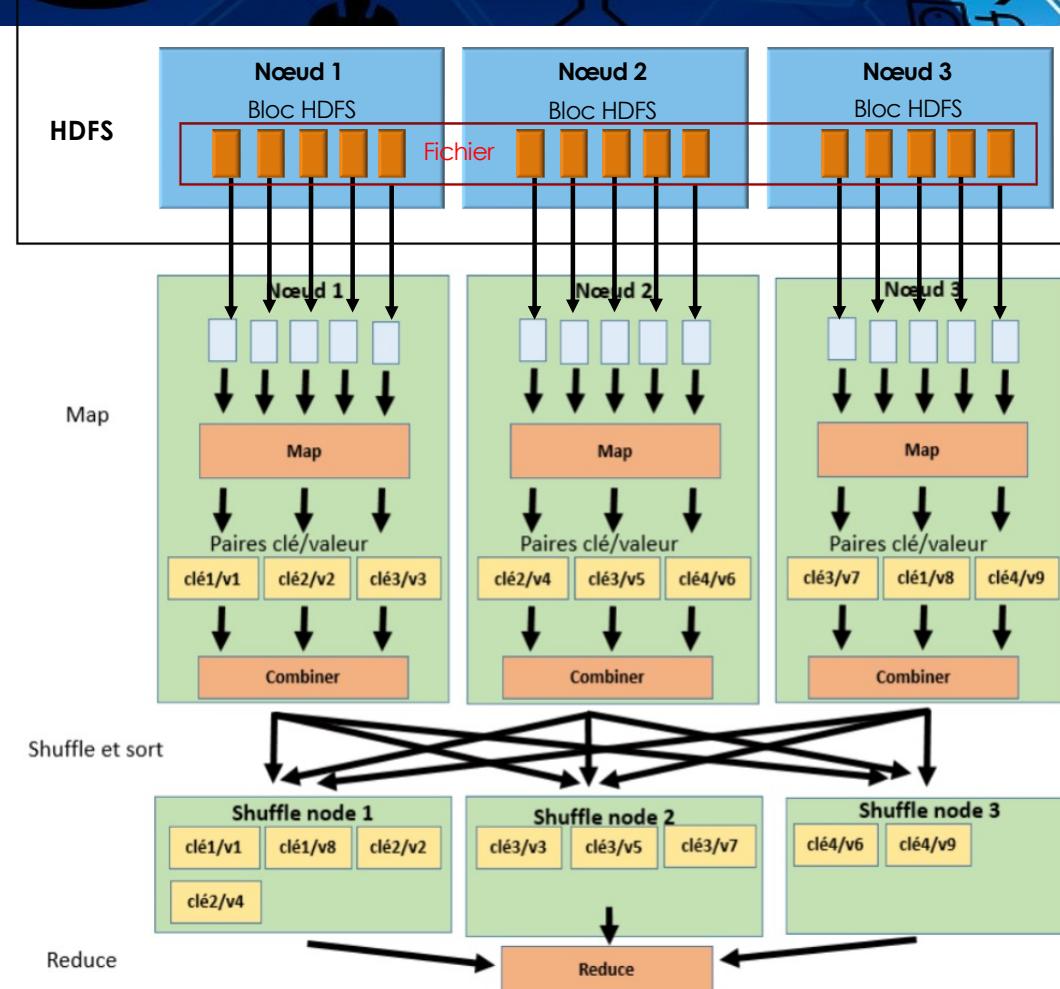


Source : <http://javax4u.blogspot.fr/2012/11/hadoop.html>

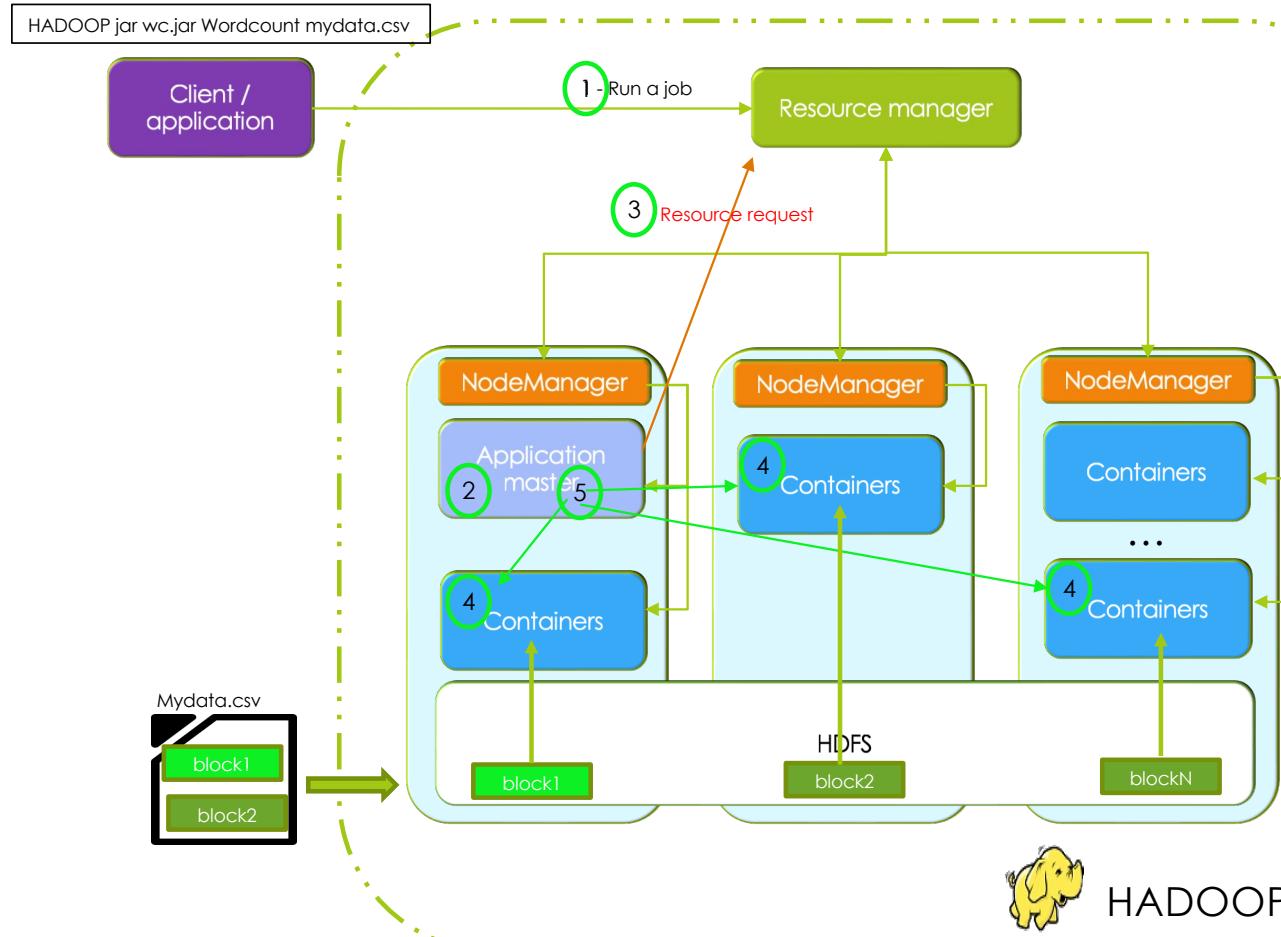
# Moteur d'exécution : MapReduce v1



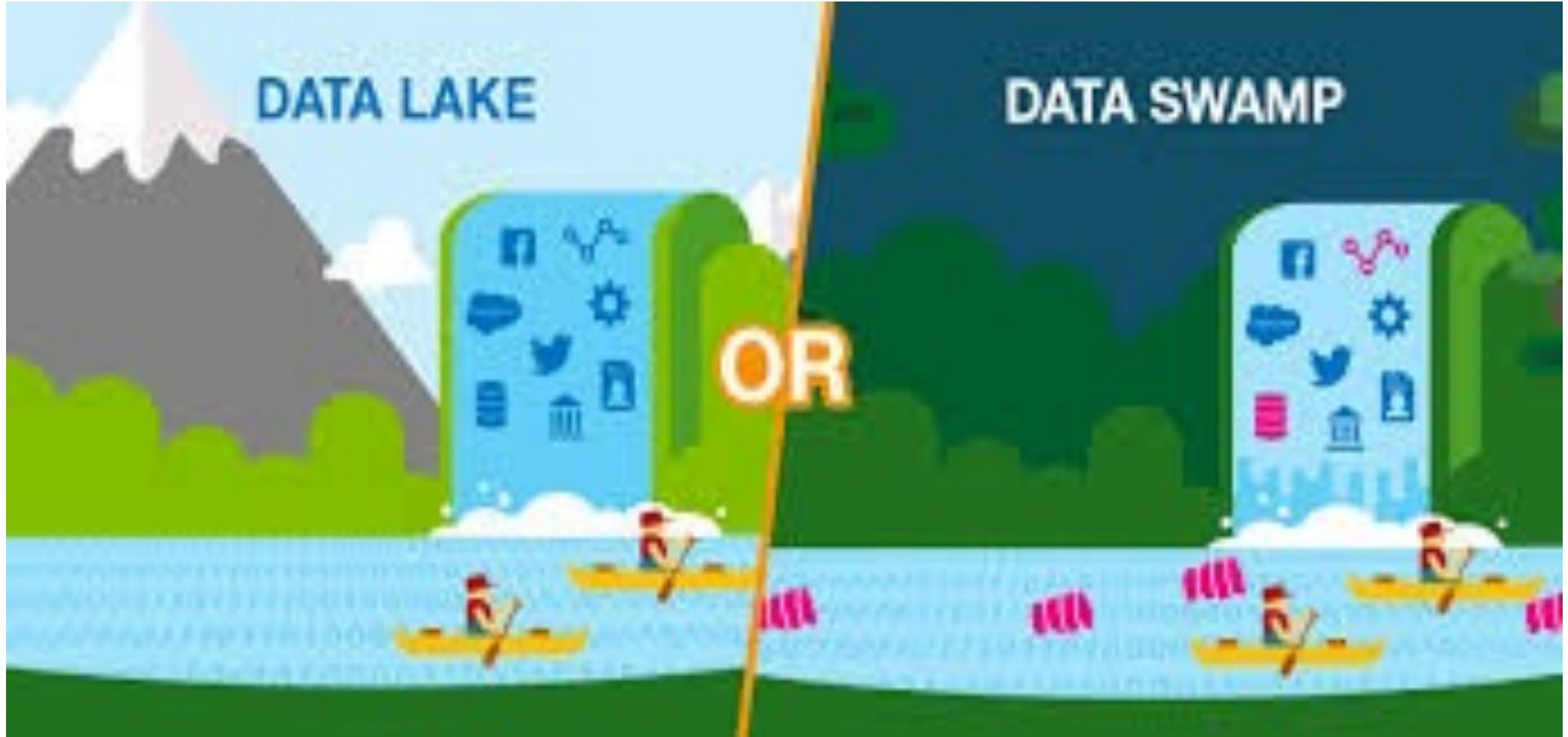
# Moteur d'exécution : MapReduce v2



# Gestionnaire de ressources: YARN

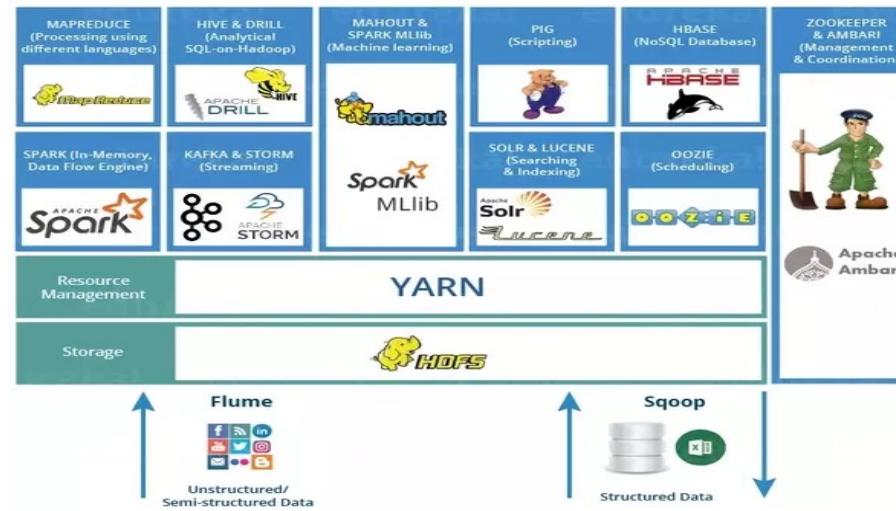


## L'écosystème HADOOP

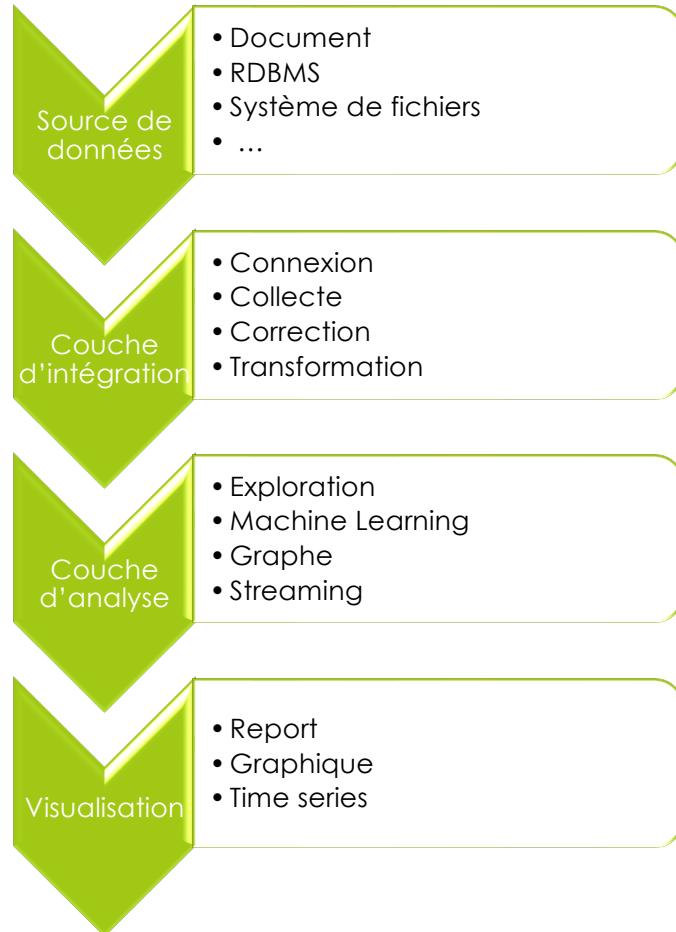


## L'écosystème HADOOP

- Grâce à l'architecture flexible de HADOOP, il est possible de supporter une large variété d'application via la containerisation.
- De nombreux projets naissent autour de HADOOP et font partie de l'écosystème HADOOP.
- Vous trouverez ci-dessous une architecture typique que l'on peut retrouver dans de nombreuses entreprises ( Hortonworks / Cloudera, mapR).

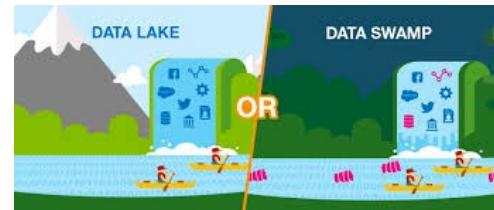


# Traitement de données



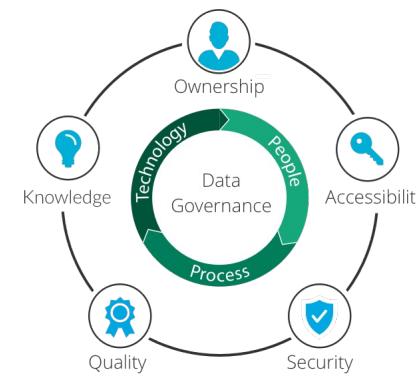
# La gouvernance de données

- Les données sont difficilement exploitables sans organisation.



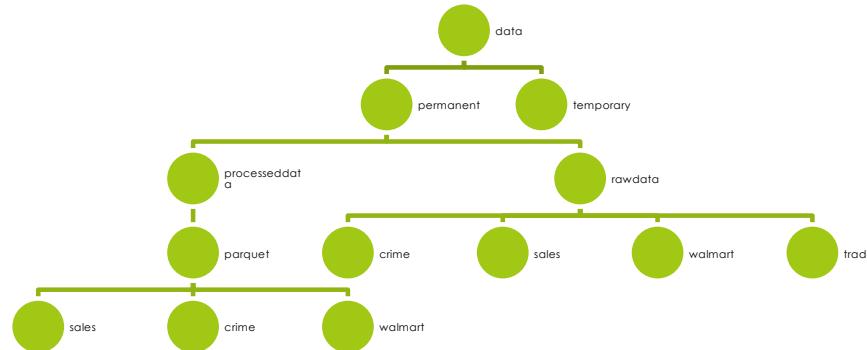
- Mise en place d'une gouvernance de données :

- normes, principes et règles régissant divers types de données.
  - Médicale
  - bancaire
- une stratégie d'entreprise pour gérer
  - les données
  - les flux
  - les accès
  - le stockage
  - les mises à jour



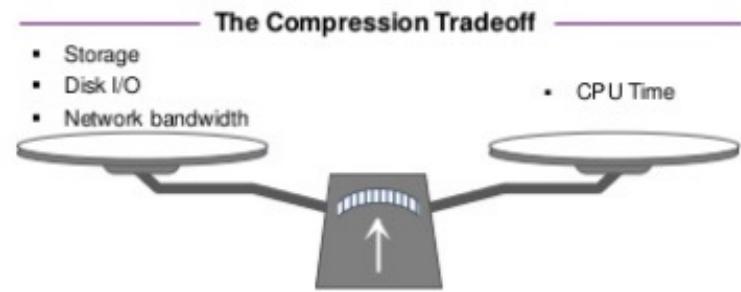
# Comment organiser vos données ?

- Où se trouvent vos données ?
- Quels types de données possédez-vous ?
- Qu'arrive-t-il à vos données ?
- Vos données sont-elles exactes et sûres ?
- Exemple d'organisation pour les TP



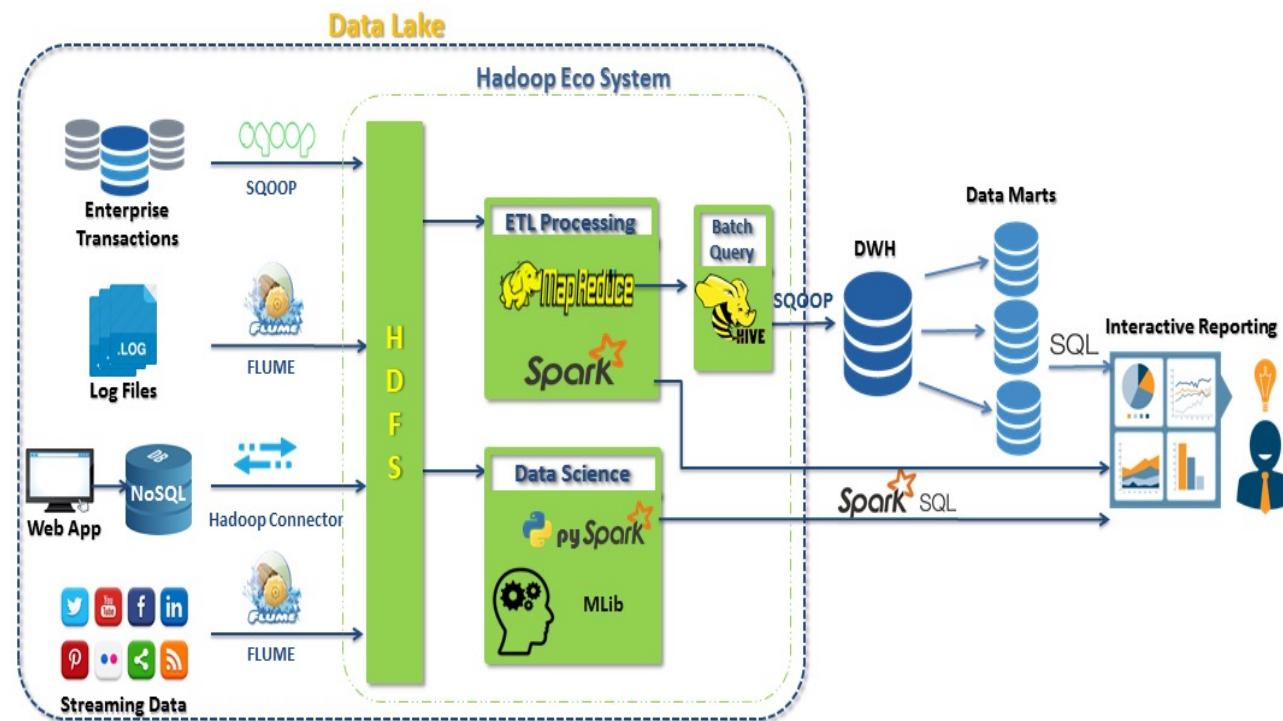
## Les formats de fichiers

- Fichier non compressé : csv, texte, json
  - Volumineux en espace
  - Sécable en partition
- Fichier compressé
  - Gain en espace
  - Pas forcément sécable
- Fichier binaire
  - Produit d'une sérialisation et sécable : avro parquet



# DATA LAKE by HADOOP

- Voici un exemple, d'une architecture Lambda avec un Data Lake basée sur l'eco-système HADOOP.



## Conclusion

