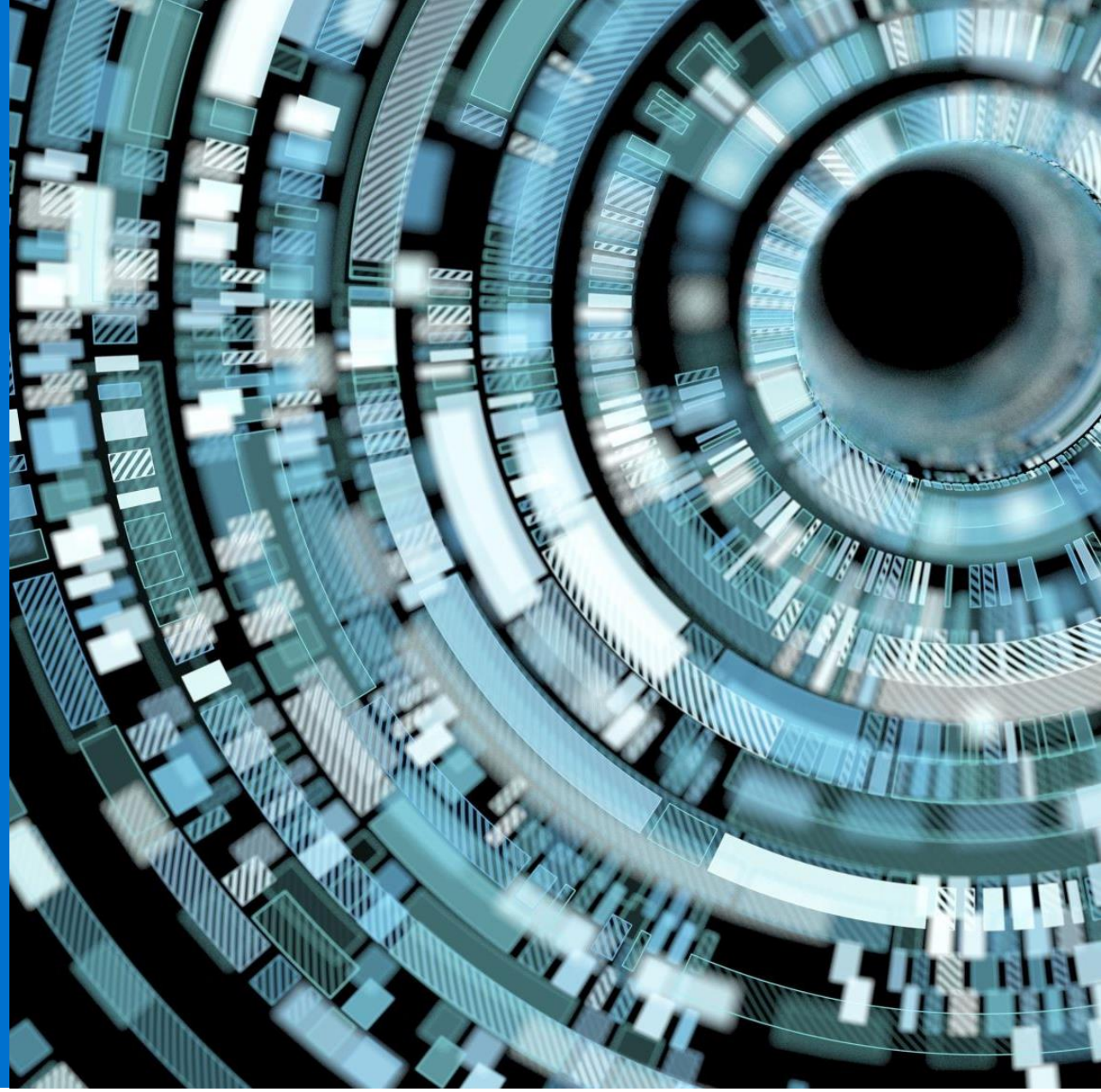
Traiter les données avec l'écosystème Spark

Lorenzo Faccioli, 2023



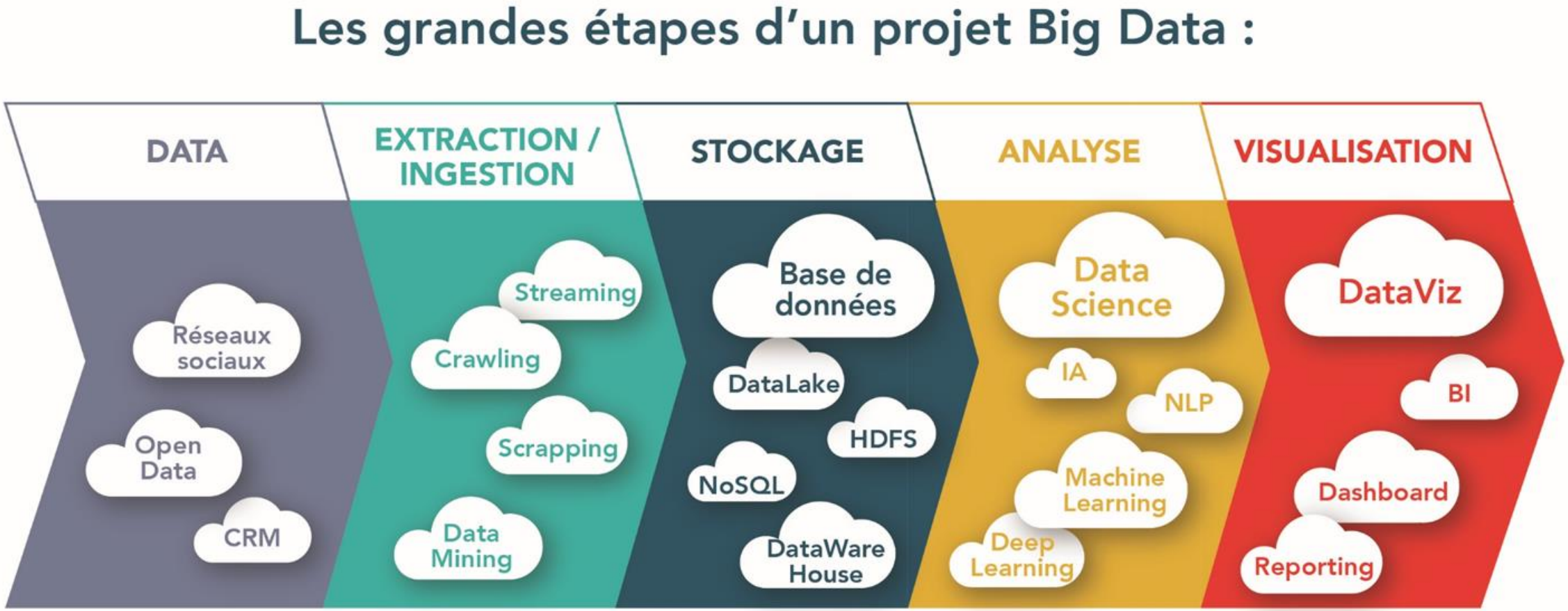
mazars

01

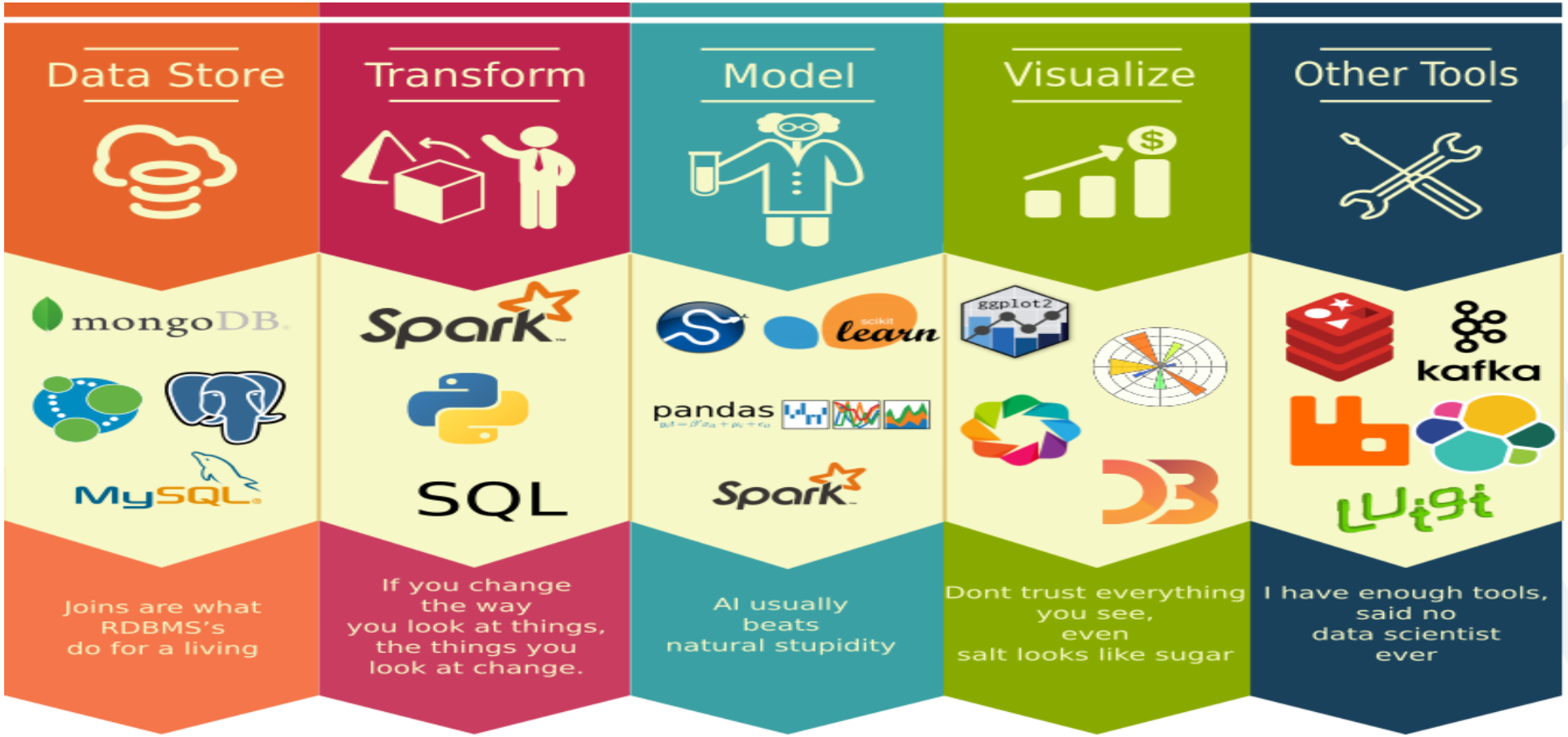
Section 01

Le monde Big Data

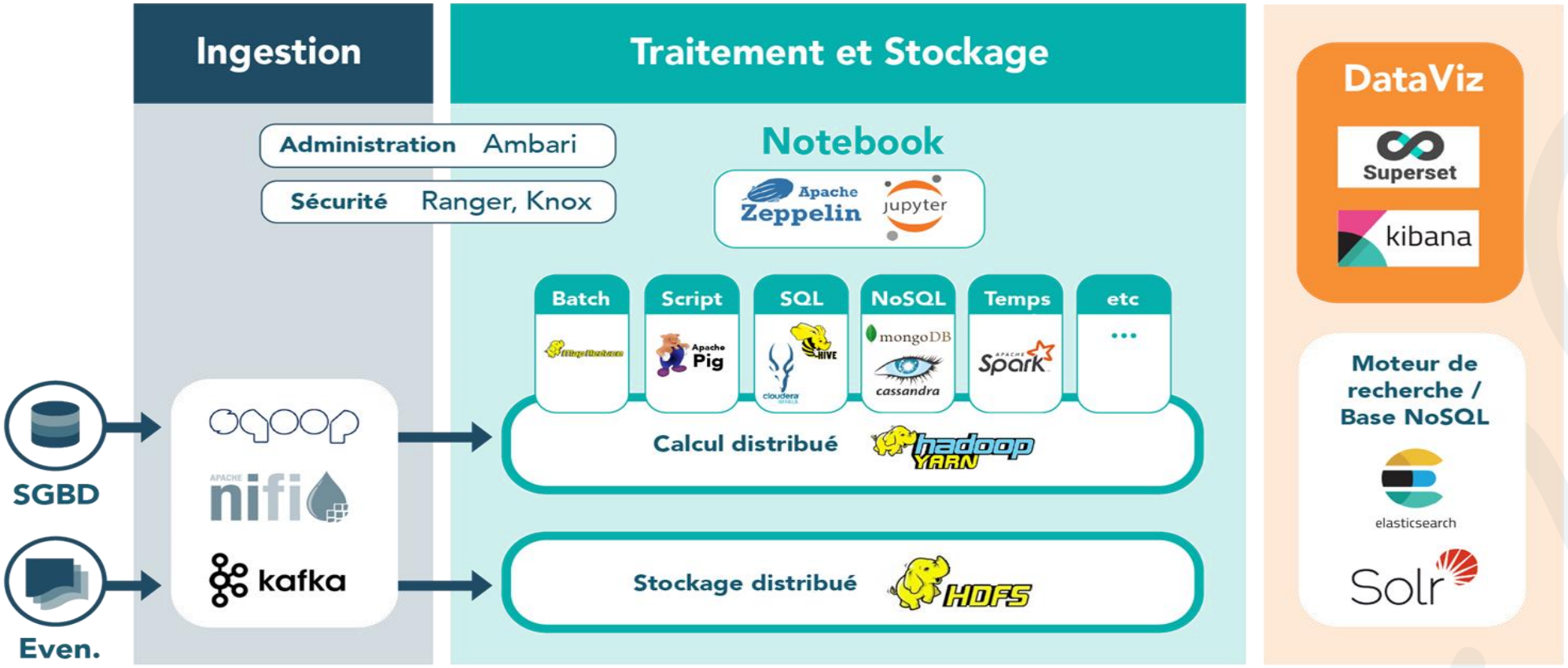
Projet Big Data



Projet Big Data



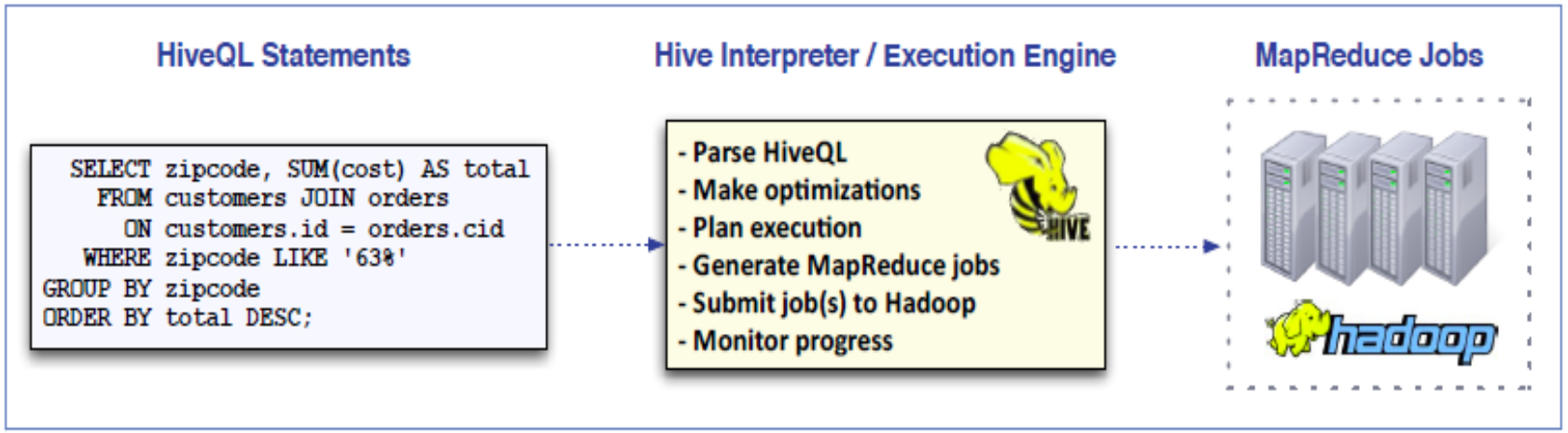
Architecture & Ecosystème Hadoop – La pile technologique de Hadoop 2



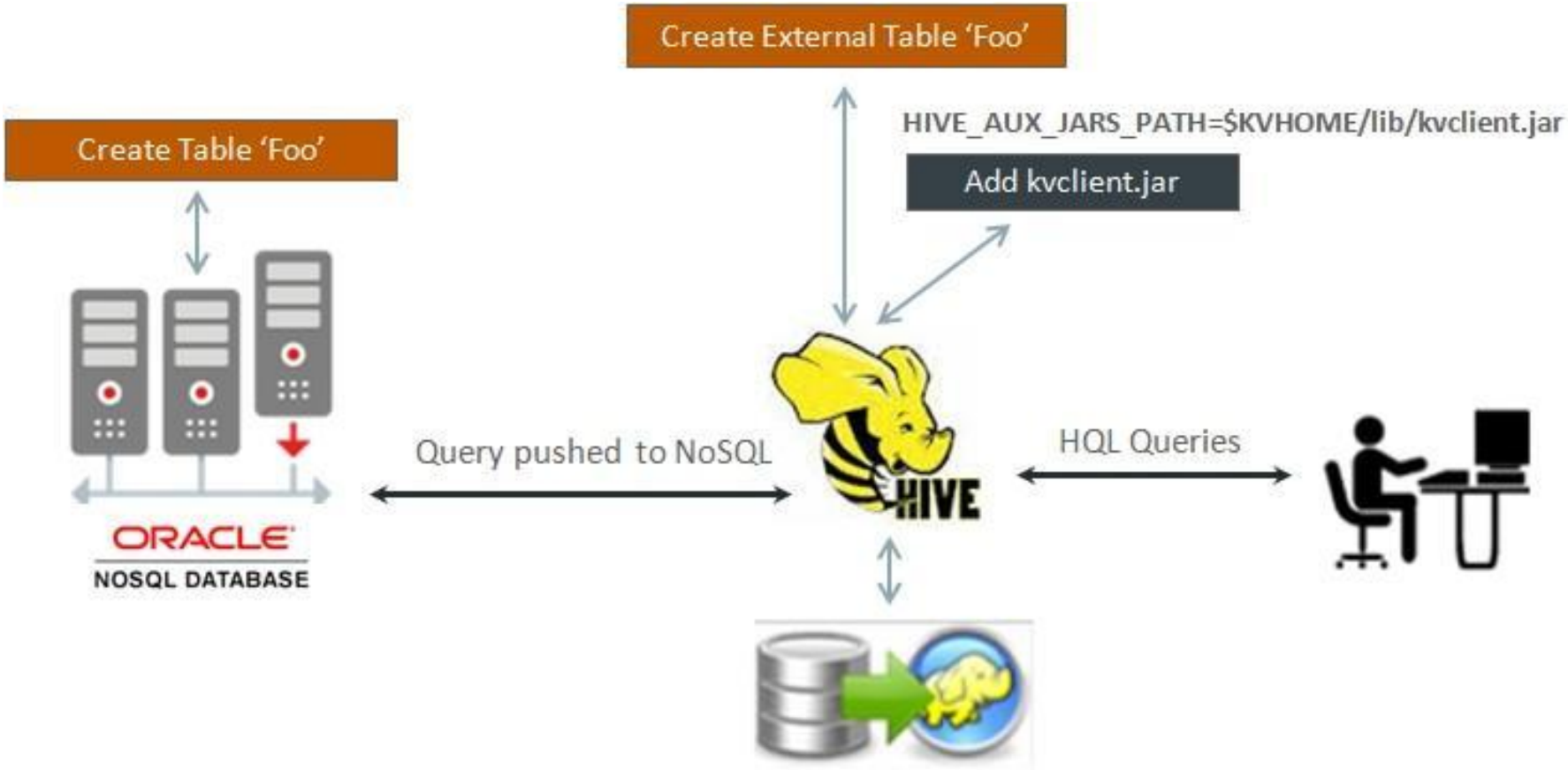
Plateforme Big Data - Hadoop

Hive, pourquoi ?

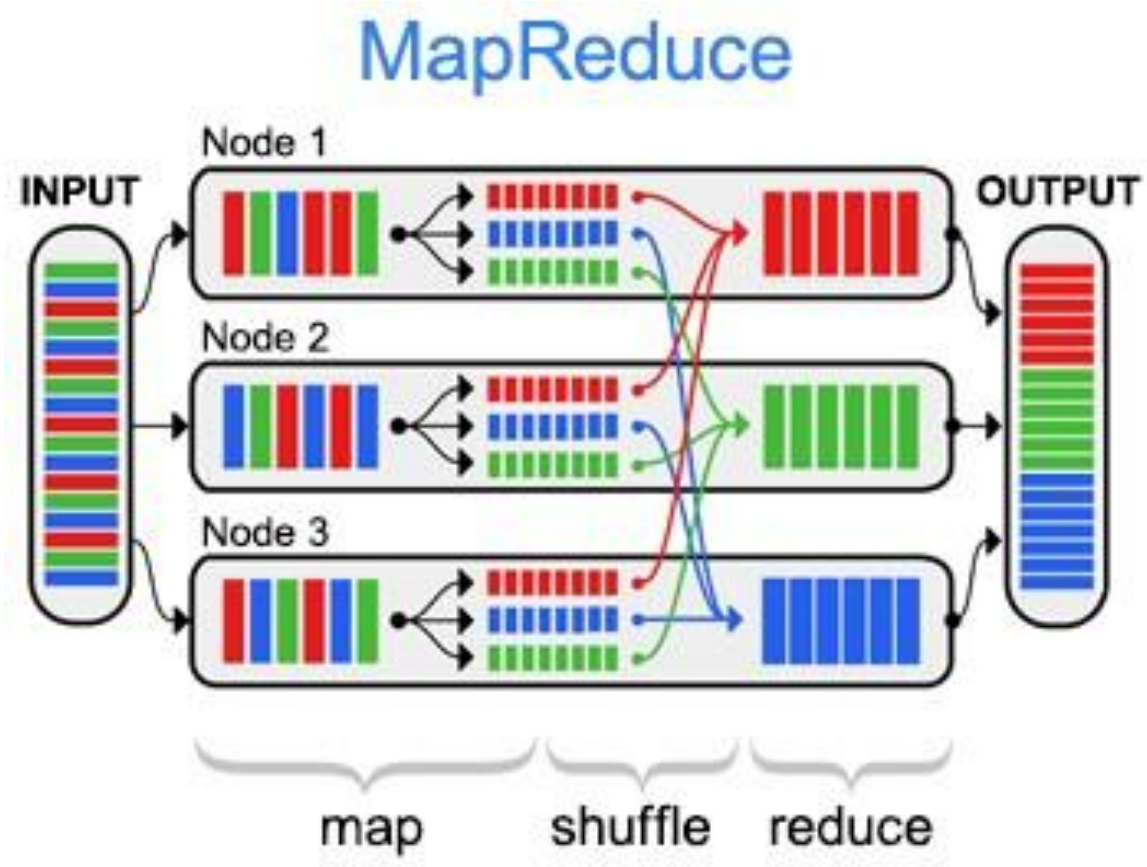
1.Planification de la requête : la requête est reçue par le driver (pilote). Elle est compilée, optimisée et planifiée comme un job
2.Exécution du job : le job est exécuté sur le cluster Hadoop



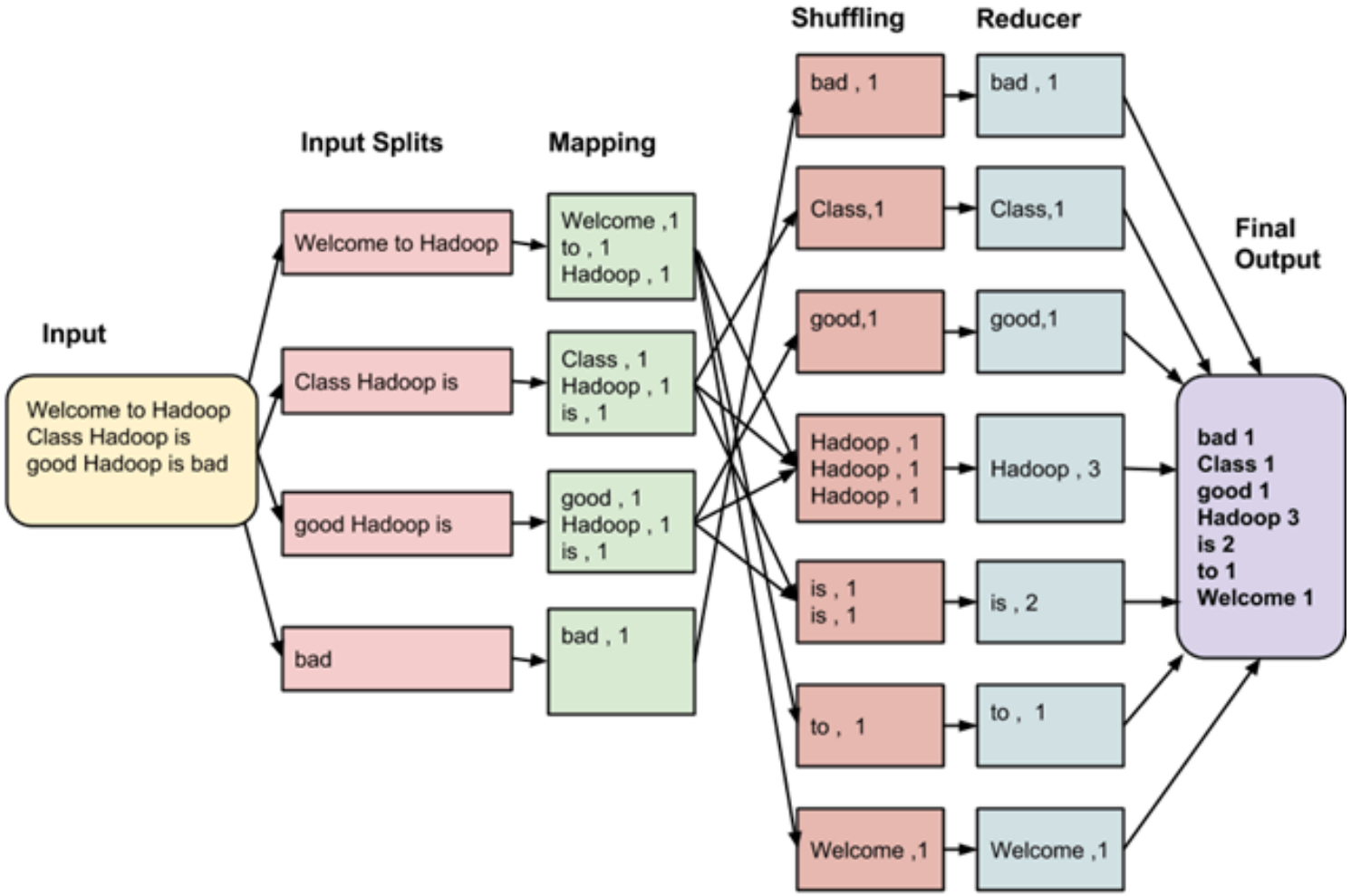
Hive, pourquoi ?



MapReduce, pourquoi ?



MapReduce, pourquoi ?



02

Section 02

HDFS

Analyse du débit de données

- Critique pour l'estimation des performances
  - Un peu d'arithmétique suffit
- La vitesse de lecture d'un disque type est d'environ 100 Mo/s (SATA 1)
  - La simple lecture d'1 To demande au moins trois heures !
  - La compression de donnée peut aider, mais nécessite du temps CPU
- Problématique : comment stocker un gros volume de données et y accéder efficacement ?

Comment augmenter le débit ?

- Problématique : comment stocker un tel volume de données et y accéder efficacement ?
- Répartir les données sur plusieurs disques afin de pouvoir y accéder en parallèle. Il faut plus de disques et de nœuds pour traiter rapidement des données massives.

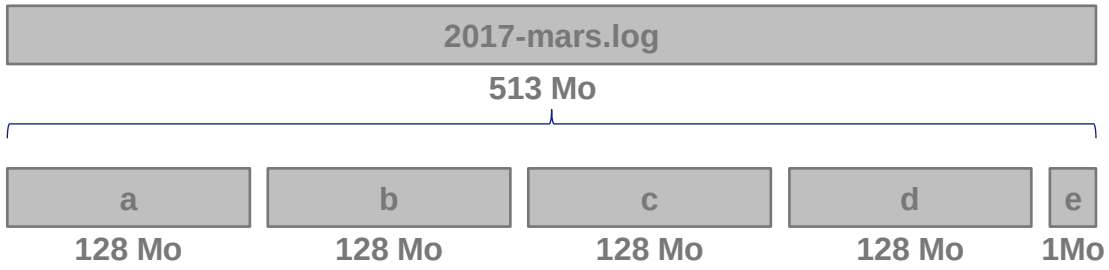
Ressources nécessaires pour traiter ces données en deux minutes

Volume de données	Nombre de disques	Nombre de nœuds (4 disques par nœud)
1 To	80	20
10 To	800	200
100 To	8000	2000
1000 To = 1 Po	80000	20000

Concepts de base de HDFS

Lorsque vous stockez un fichier dans HDFS, le système le décompose en un ensemble de blocs individuels et stocke ces blocs dans différents nœuds (serveurs) du cluster Hadoop.

- Découper les fichiers en blocs afin de les répartir sur les nœuds pour un accès parallèle.
- Dupliquer les blocs sur différents nœuds afin de se prémunir contre les pannes matérielles.



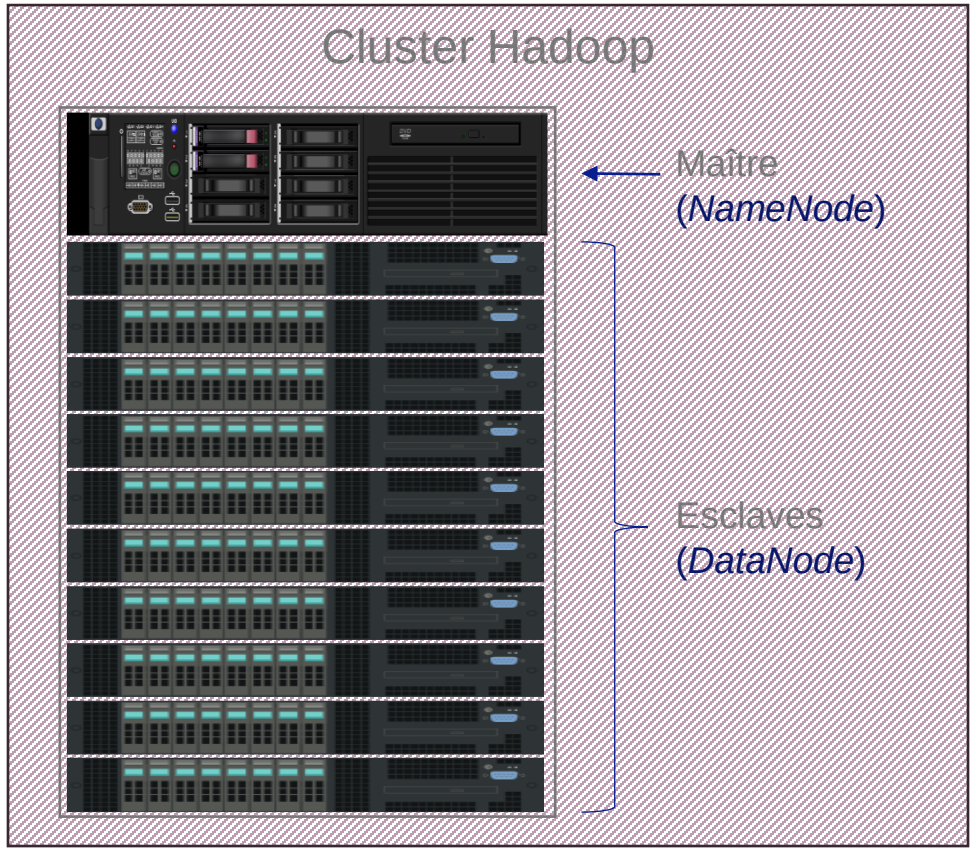
HDFS – Caractéristques

- HDFS – Hadoop Distributed File System
- Système de stockage de très gros volumes de données à bas coût et fiable.
- Optimisé pour l'accès séquentiel sur un nombre relativement petit de gros fichiers
  - Fichiers en térabytes
  - Système de fichiers en pétabytes
- Ressemble beaucoup aux système de fichier Unix
  - Hiérarchie avec le style de chemin Unix (ex: /sales/region/asia.txt)
  - Permissions et droits des fichiers à la Unix
- Incompatibilités avec Unix sur certains points (POSIX)
  - Pas de concept de dossier courant
  - Impossibilité de modifier un fichier une fois écrit (mode ajout uniquement)
  - Utilisation d'utilitaires Hadoop ou de codes spécifiques pour l'accès à HDFS.

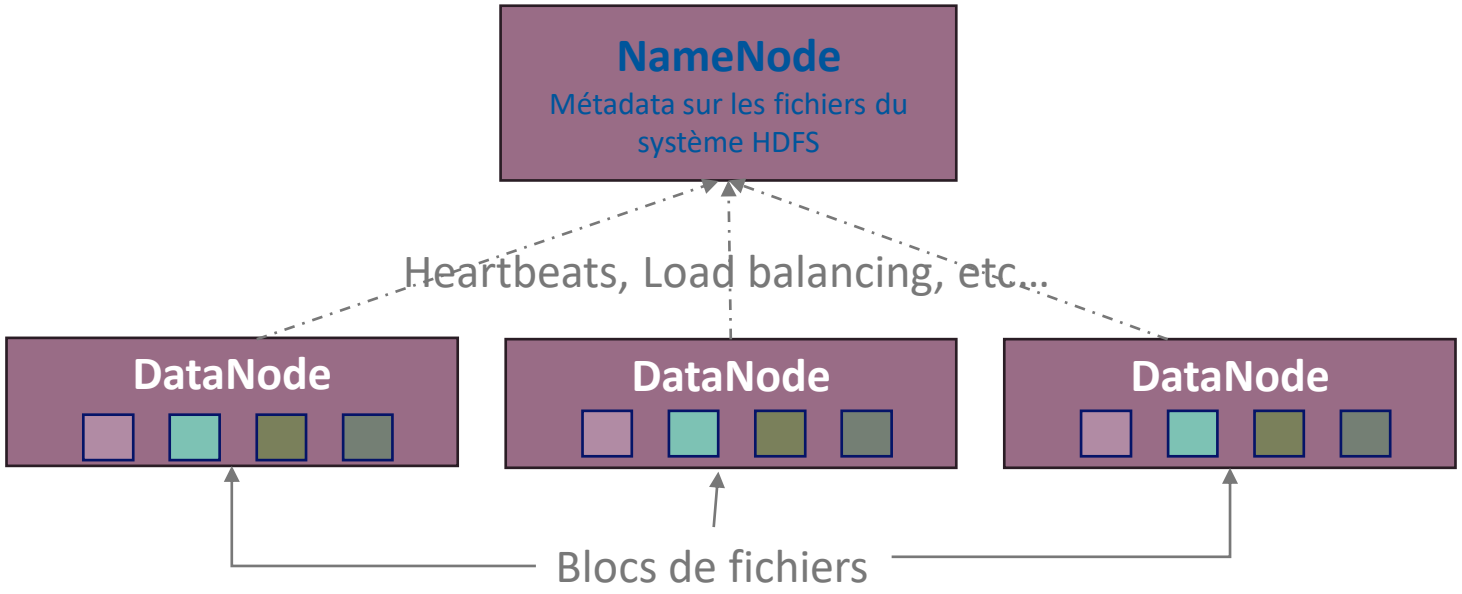
HDFS - Architecture

- Hadoop est basé sur une architecture Maitre/Esclave

- Service HDFS Maitre : NameNode
  - Gèrer les namespaces et les métadata des fichiers
  - Monitorer les nœuds esclaves
- Service HDFS Esclave : DataNode
  - Lecture/écriture des données des fichiers

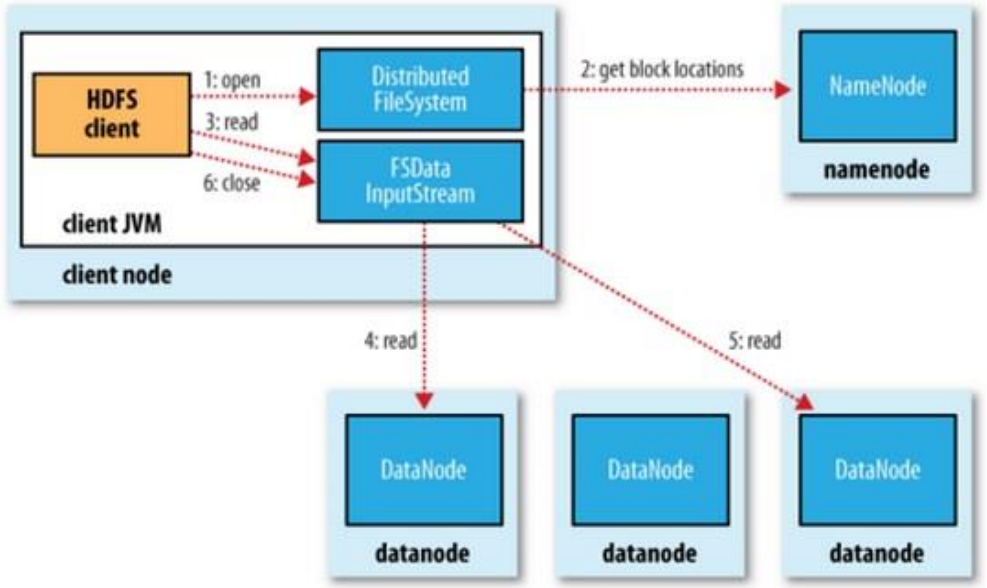


HDFS - Architecture



HDFS - Lecture des Données

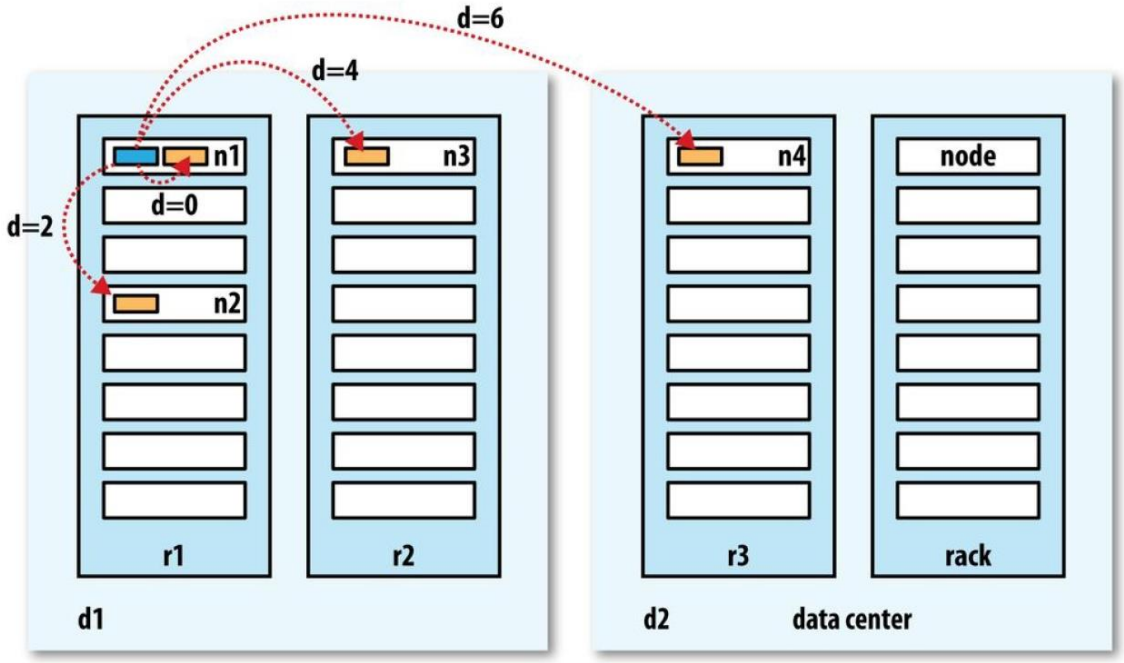
1. Ouverture d'un fichier via la fonction *open()* de l'objet *FileSystem*
2. Appel au NameNode pour récupérer la liste des DataNodes pour le premier lot de blocs du fichier
3. Connexion au premier DataNode de la liste
4. Stream des données du DataNode
5. Fermeture de la connexion au DataNode
6. Répétition des étapes 3-5 pour le bloc suivant
   ou des étapes 2-5 pour les prochains lots de blocs
7. Fermeture du fichier



Préférences de lecture des données

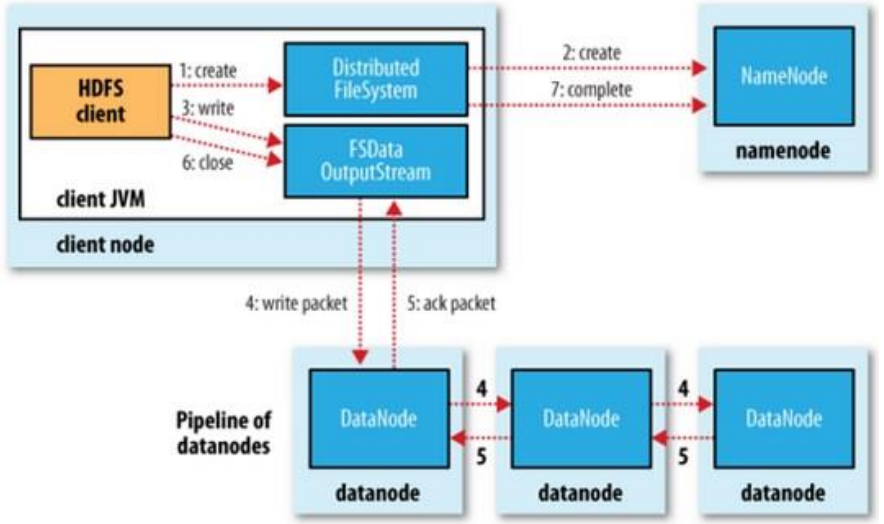
- Trouver le DataNode le plus proche hébergeant un bloc de données
- Distance calculé en fonction de la proximité réseau :

0 – Même nœud
2 – Même Rack
4 – Même Datacenter
6 – Datacenters différents



Ecriture des données

1) Création du fichier via l'appel *create()* de l'objet *DistributedFileSystem*
2) Appel au NameNode pour créer le fichier sans blocks dans l'espace de nom du filesystem
3) Appel du NameNode pour récupérer la liste des DataNodes pour stocker les "répliques" des lots de blocks de données.
4) Connexion au premier DataNode de la liste et chainage des streams de blocks de données
5) Les DataNodes accusent réception des blocks
6) Répétitions des étapes 4-5 pour le prochain block ou les étapes 3-5 pour le prochain lots de blocks
7) Fermeture de la file de données
8) Notification au NameNode de la fin de l'écriture dans le fichier



Préférences d’Écriture des Données

1er copie - coté client (si présence d’un démon DataNode) ou sélection aléatoire d’un DataNode non surchargé
2e copie - sur DataNode dans un rack différent du même datacenter
3e copie – sur DataNode aléatoire non surchargé sur le même rack
4e et copies suivantes – sur un DataNode aléatoire et non surchargé

HDFS – Services (daemons)

- DataNode
  - Responsable du stockage des données
  - Plusieurs par cluster

- NameNode
  - Responsable du stockage des méta-données
  - Maintient la base de données d'allocation des blocs
  - 1 actif par cluster

- NameNode secondaire
  - Responsable du traitement des méta-données
  - 1 actif par cluster

Données

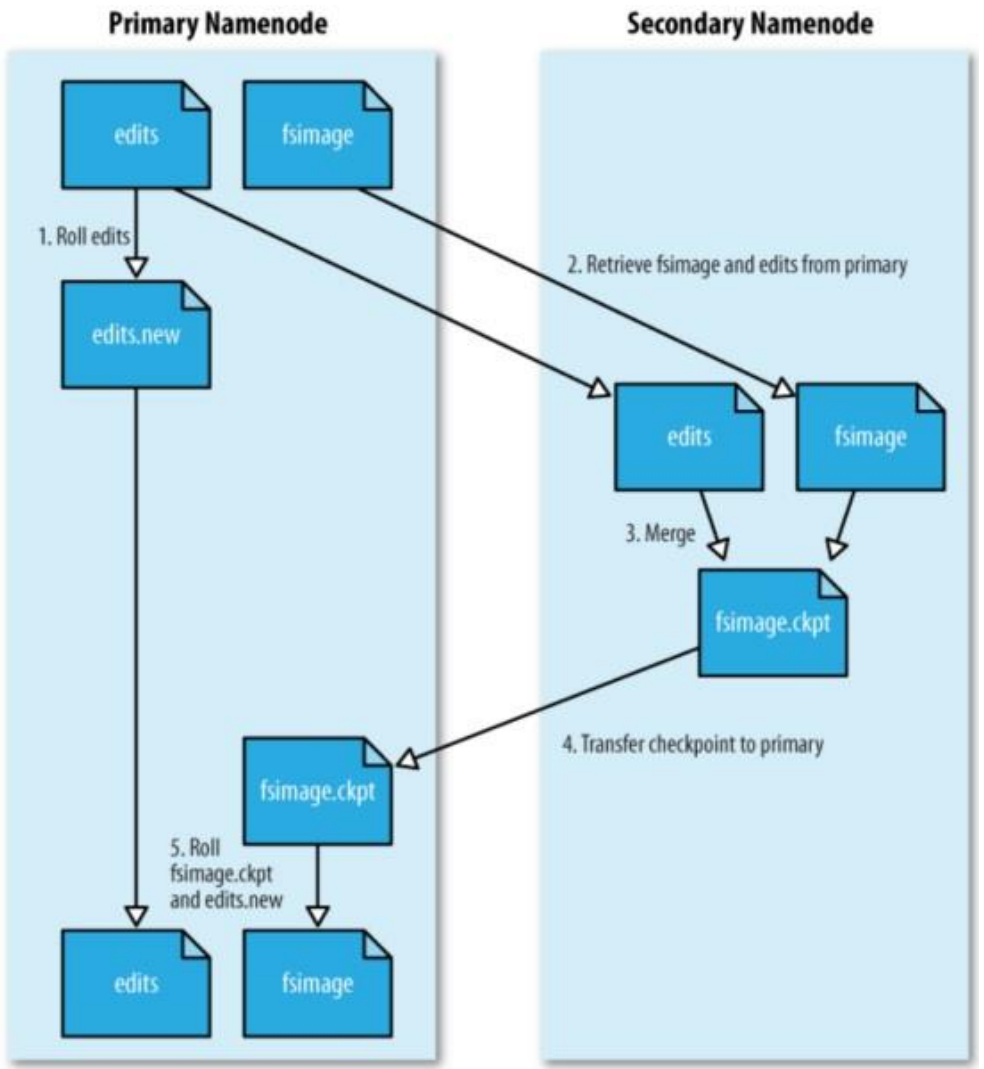
- Stockées sous forme de blocks sur le disque
- Stockées et récupérées localement par le DataNode
- Journal périodique des modifications NameNode
- Tolérance aux pannes via la réplication
- Traitements parallèles via la réplication

HDFS – Méta-Données

- Stockées sous forme de fichiers dans le NameNode:
  - fsimage_[transaction ID] : snapshot des méta-données du filesystem jusqu'à la transaction spécifiée
  - edits_inprogress_[transaction ID] : modifications incrémentales des méta-données depuis la transaction spécifiée.
- Contient des informations sur les propriétés du filesystem
- Copiées et servies à partir du NameNode en RAM
- NameNode secondaire
- Ne pas confondre méta-données avec la base des localisations des données.

Gestion des méta-données

- **Gestion périodique des méta-données**
  - Chaque heure
  - *éditions* des fichiers dépassant les 64 Mo



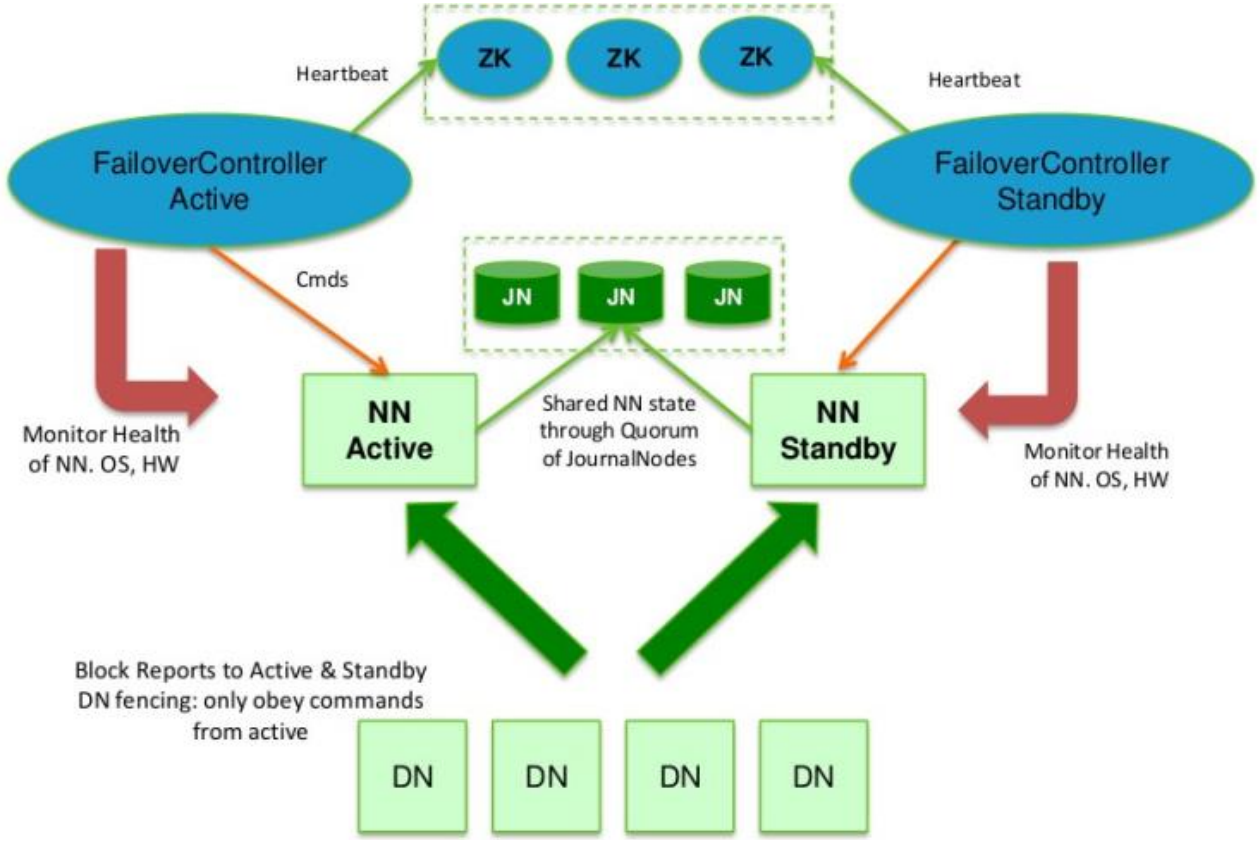
HDFS – Méta-données

```
data/dfs/name
├── current
│   ├── VERSION
│   ├── edits_0000000000000000001-0000000000000000007
│   ├── edits_0000000000000000008-0000000000000000015
│   ├── edits_0000000000000000016-0000000000000000022
│   ├── edits_0000000000000000023-0000000000000000029
│   ├── edits_0000000000000000030-0000000000000000030
│   ├── edits_0000000000000000031-0000000000000000031
│   ├── edits_inprogress_0000000000000000032
│   ├── fsimage_0000000000000000030
│   ├── fsimage_0000000000000000030.md5
│   ├── fsimage_0000000000000000031
│   ├── fsimage_0000000000000000031.md5
│   └── seen_txid
└── in_use.lock
```

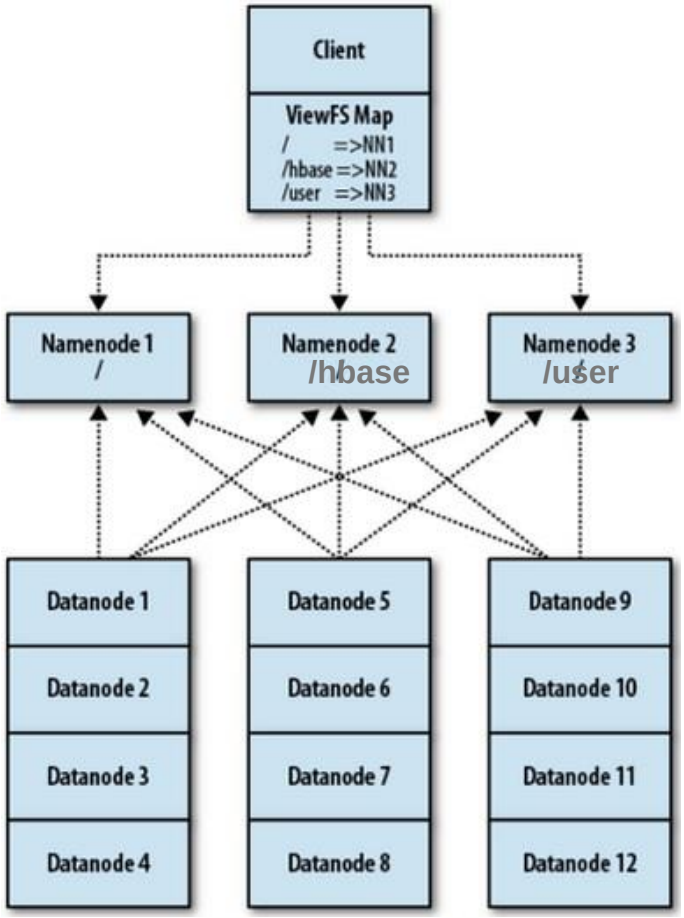
HDFS – Méta-données

```
data/dfs/data/
├── current
│ ├── BP-1079595417-192.168.2.45-1412613236271
│ │ ├── current
│ │ │ ├── VERSION
│ │ │ ├── finalized
│ │ │ │ └── subdir0
│ │ │ │ └── subdir1
│ │ │ │ ├── blk_1073741825
│ │ │ │ └── blk_1073741825_1001.meta
│ │ │ |── lazyPersist
│ │ │ └── rbw
│ │ ├── dncp_block_verification.log.curr
│ │ ├── dncp_block_verification.log.prev
│ │ └── tmp
│ └── VERSION
└── in_use.lock
```

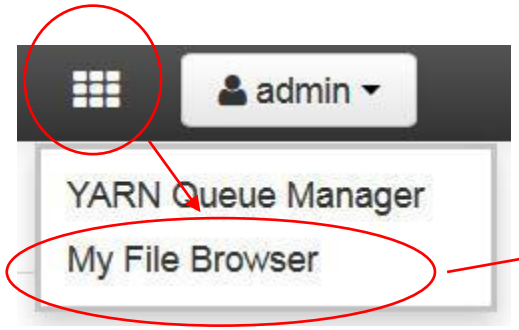
HDFS – Haute Disponibilité (NameNode)

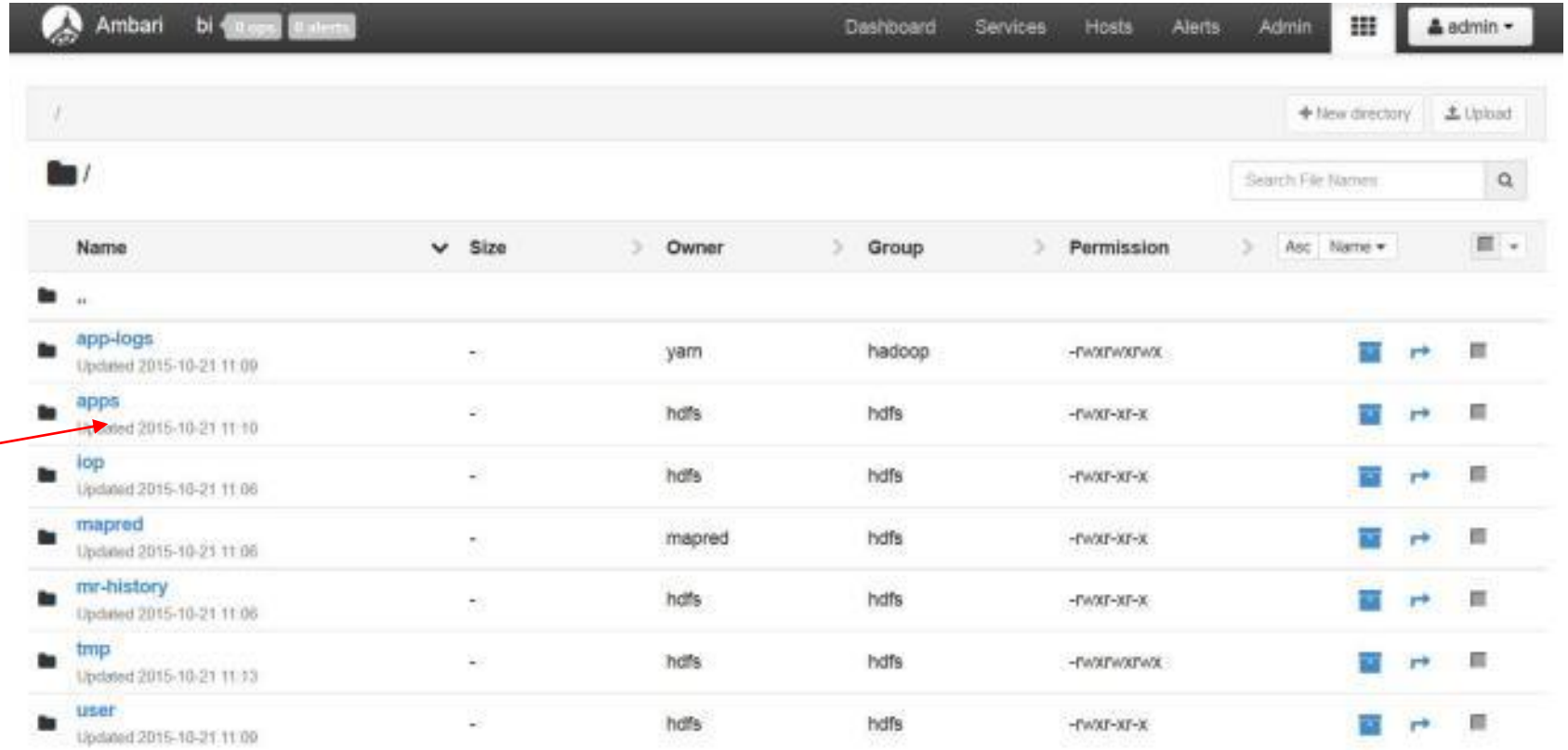


HDFS – Fédération de NameNode



HDFS – Navigation via Ambari





HDFS – CLI – Accès via la Ligne de Commande

- **HDFS n'est pas un système de fichier à but généraliste**
  - Pas intégré aux systèmes d'exploitation, recours à des outils spécialisés pour y accéder
  - L'utilisateur final y accède typiquement via la commande en ligne « hadoop fs »

- **Exemple : affichage du contenu du fichier « /user/fred/sales.txt »**

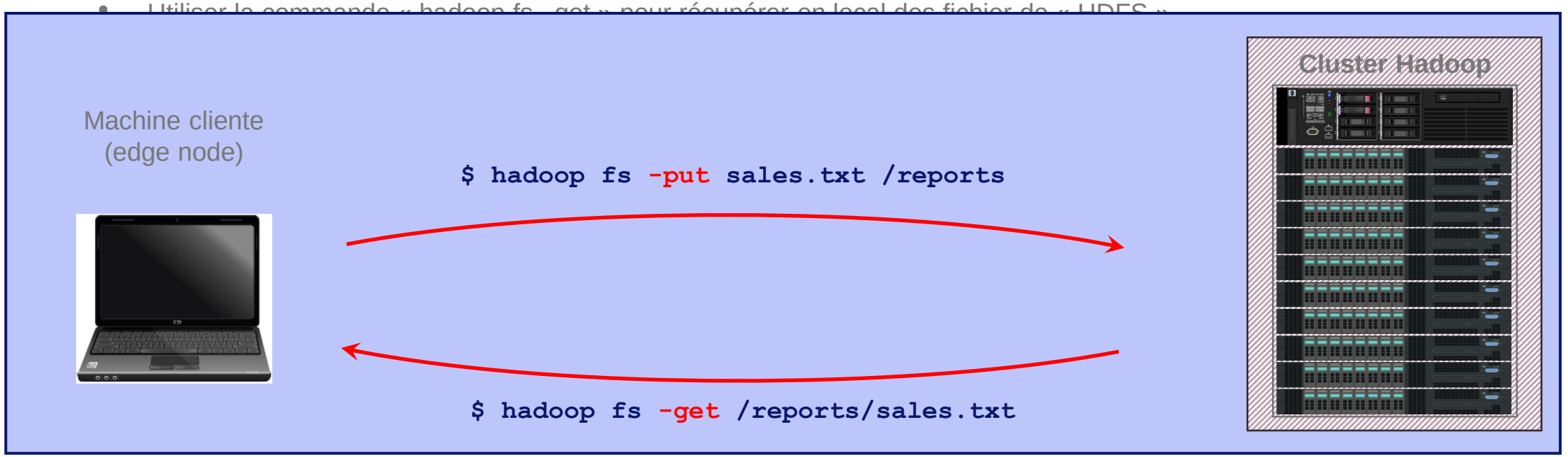
```
$ hadoop fs -cat /user/fred/sales.txt
```

- **Exemple : création d'un dossier (sous la racine) appelé « reports »**

```
$ hadoop fs -mkdir /reports
```

HDFS – Exemple

- **Rappel : le système HDFS est distinct de votre système de fichier local**
  - Utiliser la commande « hadoop fs –put » pour copier des fichier locaux vers « HDFS »
  - Utiliser la commande « hadoop fs –get » pour récupérer en local des fichier de « HDFS »



HDFS – Ligne de Commande

HDFS offre un ensemble de commandes qui peuvent être utilisées sur la console

Les commandes se présentent sous la forme générale :

```
$ hadoop fs -cmd <args>
```

Où **cmd** est une commande spécifique et <**args**> est un nombre variable d'arguments

Les commandes sont très similaires aux commandes Unix; par ex, pour dresser la liste des fichiers de HDFS

```
$ hadoop fs -ls /user
```

HDFS – Commandes de Base

- La commande « get » de HDFS place les fichiers de HDFS sur le système de fichiers

```
hadoop fs -get  NomFichier.txt .
```

Répertoire en cours sur le système de fichiers local

- Hadoop prend également en charge les commandes « cat », « tail » et « head » pour l'examen d'un fichier.

```
hadoop fs -cat  NomFichier.txt

hadoop fs -cat NomFichier.txt | head

hadoop fs -tail NomFichier.txt
```

Affiche les 10 premières lignes du fichier

Affiche le dernier kilooctect de données

- La commande pour supprimer des fichiers est « rm ».

```
hadoop fs -rm NomFichier.txt
```

HDFS – Liste des Commande Shell

1.	**ls <path>**	Lists the contents of the directory specified by path, showing the names, permissions, owner, size and modification date for each entry.
2.	**lsr <path>**	Behaves like -ls, but recursively displays entries in all subdirectories of path.
3.	**du <path>**	Shows disk usage, in bytes, for all the files which match path; filenames are reported with the full HDFS protocol prefix.
4.	**dus <path>**	Like -du, but prints a summary of disk usage of all files/directories in the path.
5.	**mv <src><dest>**	Moves the file or directory indicated by src to dest, within HDFS.
6.	**cp <src> <dest>**	Copies the file or directory identified by src to dest, within HDFS.
7.	**rm <path>**	Removes the file or empty directory identified by path.
8.	**rmr <path>**	Removes the file or directory identified by path. Recursively deletes any child entries (i.e., files or subdirectories of path).
9.	**put <localSrc> <dest>**	Copies the file or directory from the local file system identified by localSrc to dest within the DFS.
10.	**copyFromLocal <localSrc> <dest>**	Identical to -put
11.	**moveFromLocal <localSrc> <dest>**	Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.
12.	**get [-crc] <src> <localDest>**	Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.
13.	**getmerge <src> <localDest>**	Retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.

14.	**cat <filen-ame>**	Displays the contents of filename on stdout.
15.	**copyToLocal <src> <localDest>**	Identical to -get
16.	**moveToLocal <src> <localDest>**	Works like -get, but deletes the HDFS copy on success.
17.	**mkdir <path>**	Creates a directory named path in HDFS. Creates any parent directories in path that are missing (e.g., mkdir -p in Linux).
18.	**setrep [-R] [-w] rep <path>**	Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)
19.	**touchz <path>**	Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.
20.	**test -[ezd] <path>**	Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.
21.	**stat [format] <path>**	Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y).
22.	**tail [-f] <file2name>**	Shows the last 1KB of file on stdout.
23.	**chmod [-R] mode,mode,... <path>...**	Changes the file permissions associated with one or more objects identified by path.... Performs changes recursively with R. mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask.
24.	**chown [-R] [owner][:[group]] <path>...**	Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified.
25.	**chgrp [-R] group <path>...**	Sets the owning group for files or directories identified by path.... Sets group recursively if -R is specified.
26.	**help <cmd-name>**	Returns usage information for one of the commands listed above. You must omit the leading '-' character in cmd.

HDFS – Gestion des Snapshots

- HDFS SNAPSHOTS
  - Permet de protéger les données critiques de l'entreprise d'erreurs de manipulation des utilisateurs ou des applications
  - Crée un point de restauration du système de fichier, ou d'un sous dossier à un instant donné.
  - Le « snapshot » est en mode lecture-seul
  - La création d'un « snapshot » est une opération atomique et instantanée, quelque soit la tailles des données du système de fichier.
  - Mise à échelle : il n'y a pas création de copies des blocks de données.

HDFS – Gestion des Snapshots

- **HDFS SNAPSHOTS**
  - Autoriser les « snapshots » au niveau d'un dossier
    ```
    hdfs dfsadmin -allowSnapshot  /tmp/test
    ```
  - Création d'un snapshot du dossier : /tmp/test
    ```
    hdfs dfs -createSnapshot /tmp/test
    ```
  - Création d'un sous-dossier pour le snapshot :
    ```
    /tmp/test/.snapshot/s20160728-000941.091
    ```
  - Afficher le contenu du snapshot
    ```
    hdfs dfs -ls /tmp/test/.snapshot/s20160728-000941.091
    ```

HDFS – Gestion des Snapshots

- **HDFS SNAPSHOTS**
  - Quelques commandes admin utiles :

```
hdfs dfsadmin -allowSnapshot <path>
hdfs dfsadmin -disallowSnapshot <path>
```

  - Quelques commandes utilisateurs utiles

```
hdfs dfs -createSnapshot <path> [<snapshotName>]
hdfs dfs -deleteSnapshot <path> <snapshotName>
hdfs dfs -renameSnapshot <path> <oldName> <newName>
hdfs lsSnapshottableDir
hdfs snapshotDiff <path> <fromSnapshot> <toSnapshot>
```

HDFS - Quotas

- **« Count quotas » versus « space quotas »** :
  - count quotas : limite du nombre de fichiers dans un répertoire HDFS
  - space quotas : limite l'utilisation de l'espace disque d'un répertoire HDFS
- **Fixer des quotas**:

```
hadoop dfsadmin -setQuota [count] [path] hadoop dfsadmin -setSpaceQuota [size] [path]
```

- **Supprimer les quotas**:

```
hadoop dfsadmin -clrQuota [path] hadoop dfsadmin -clrSpaceQuota [path]
```

- **Visualiser les quotas**:

```
hadoop fs -count -q [path]
```

HDFS – Gestion des Accès avec ACL

- **Utilisation des ACLs (Access Control Lists) pour gérer les droits**
  - Activer le

```
<property>
          <name>dfs.namenode.acls.enabled</name>
          <value>true</value>
</property>
```

  - Utiliser les commandes ACL via le shell unix

```
$ hadoop dfs -setfacl  ....
$ hadoop dfs -getfacl  ….
```

HDFS – Gestion des Accès avec ACL

- ACL pour l'ajout de droits automatiques aux sous-dossier
- ents

```
$ hdfs dfs -setfacl -m default:group:execs:r-x
/monthly-sales-data
```

- Créer des sous dossiers

```
$ hdfs dfs -mkdir /monthly-sales-data/JAN
$ hdfs dfs -mkdir /monthly-sales-data/FEB
```

- Vérifier les ACL des sous dossiers

```
$ hdfs dfs -getfacl -R /monthly-sales-data
# file: /monthly-sales-data
# owner: bruce
# group: sales
user::rwx
group::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other::---
# file: /monthly-sales-data/FEB
# owner: bruce
# group: sales
user::rwx
group::r-x
group:execs:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other::---
# file: /monthly-sales-data/JAN
# owner: bruce
# group: sales
user::rwx
group::r-x
group:execs:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other::---
```

03

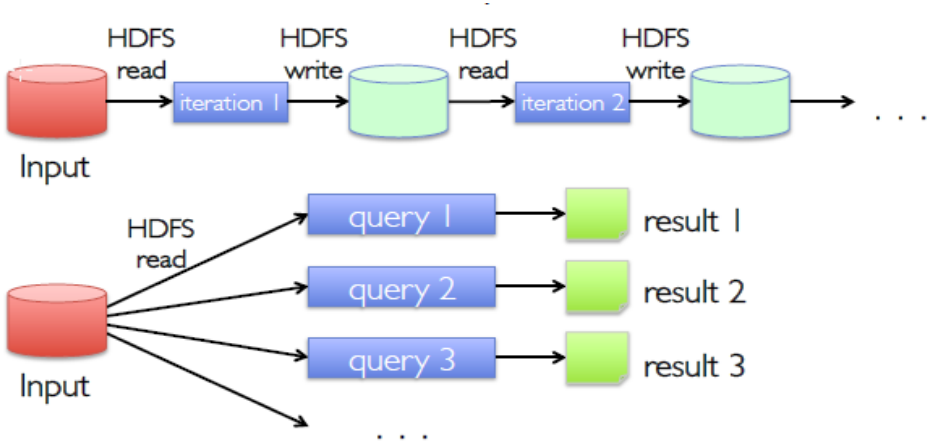
Section 03

Spark

- Apparition - Principe

Hadoop ➡ Stockage distribué + Traitement distribué de grands volumes de données

- **Stockage de données:** Hadoop est la plateforme distribuée la plus utilisée pour stocker de grands volumes de données
- **Traitement de Données:** paradigme naif
  - **Lecture sur HDFS (disques)** - à chaque fois, même si le traitement est itératif et nécessite plusieurs lectures
  - **Ecriture sur HDFS (disques)** – toutes tailles de résultats finaux et résultats intermédiaires



- Problématique : comment stocker un tel volume de données et y accéder efficacement ?
- Répartir les données sur plusieurs disques afin de pouvoir y accéder en parallèle. Il faut plus de disques et de nœuds pour traiter rapidement des données massives.

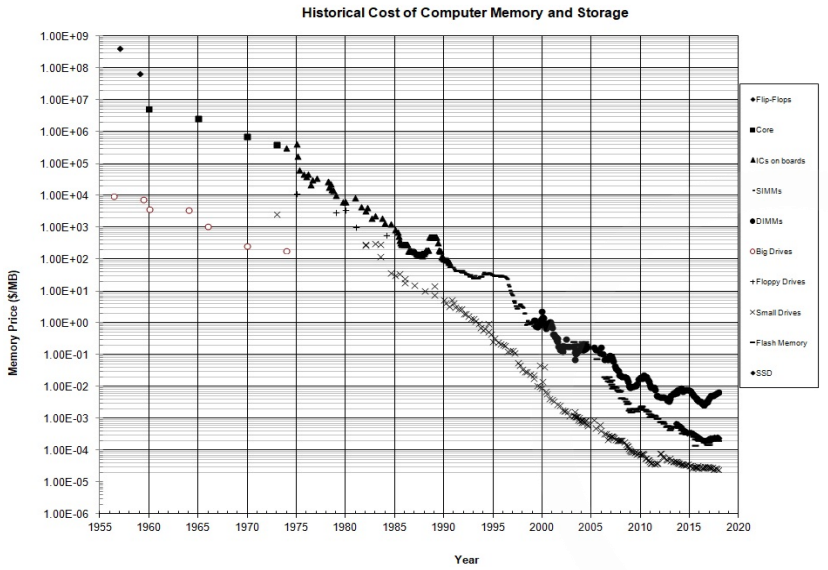
Ressources nécessaires pour traiter ces données en deux minutes

Volume de données	Nombre de disques	Nombre de nœuds (4 disques par nœud)
1 To	80	20
10 To	800	200
100 To	8000	2000
1000 To = 1 Po	80000	20000

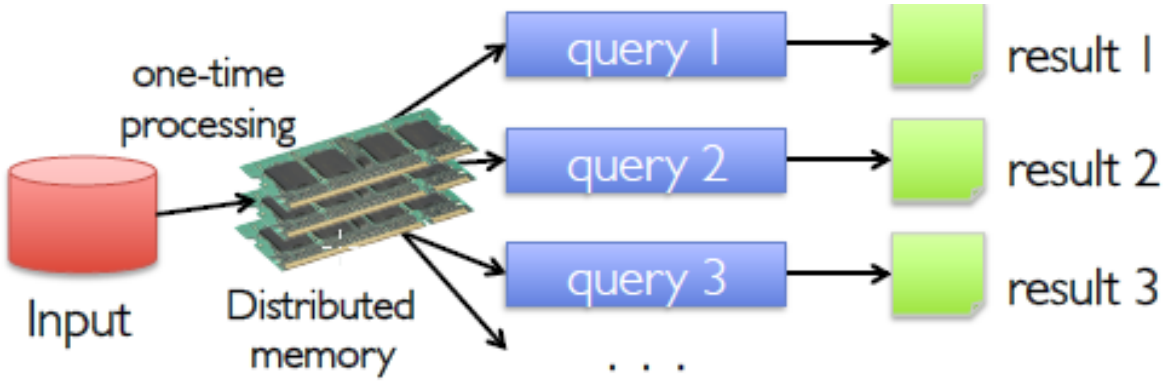
Avancée technologique: baisse du coût de la RAM

Amélioration 1: Exploiter au maximum la RAM disponible

- L'IO de la RAM est 100 fois plus rapide que l'IO des disques
- La RAM est un composant fortement démocratisé
- Les nœuds de cluster Hadoop sont souvent bien équipés en RAM (32, 64, 128…)

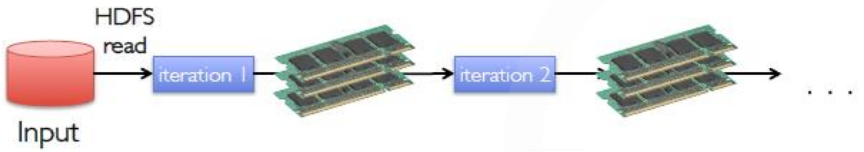


http://jcmit.net/mem2015.htm



Amélioration 2: Rendre intelligents les plans d'exécutions des requêtes

- Construire un plan de stratégie d'exécution avant l'exécution de la requêtes
- Identifier les jeux de données qui seront lus plusieurs fois pour les laisser en RAM



Apparition de Spark qui cherche à accélérer le traitement des systèmes Hadoop

Important : Spark n'est pas un remplaçant d'Hadoop, il est plutot une alternative au MapReduce d'Hadoop.

SPARK: Pourquoi?

Celui qui possède le plus de données domine le marché

Banques ?

Assurances ?

04

Section 04

Spark

- Spark vs MapReduce

Hadoop et Spark deux framework Big Data mais n'ont pas le même usage

- Hadoop est un framework de stockage et de traitement distribué des données en batch
- Spark est un framework de traitement des données temps rééel mais il n'assure pas le stockage

Hadoop et Spark peuvent s'utiliser indépendamment

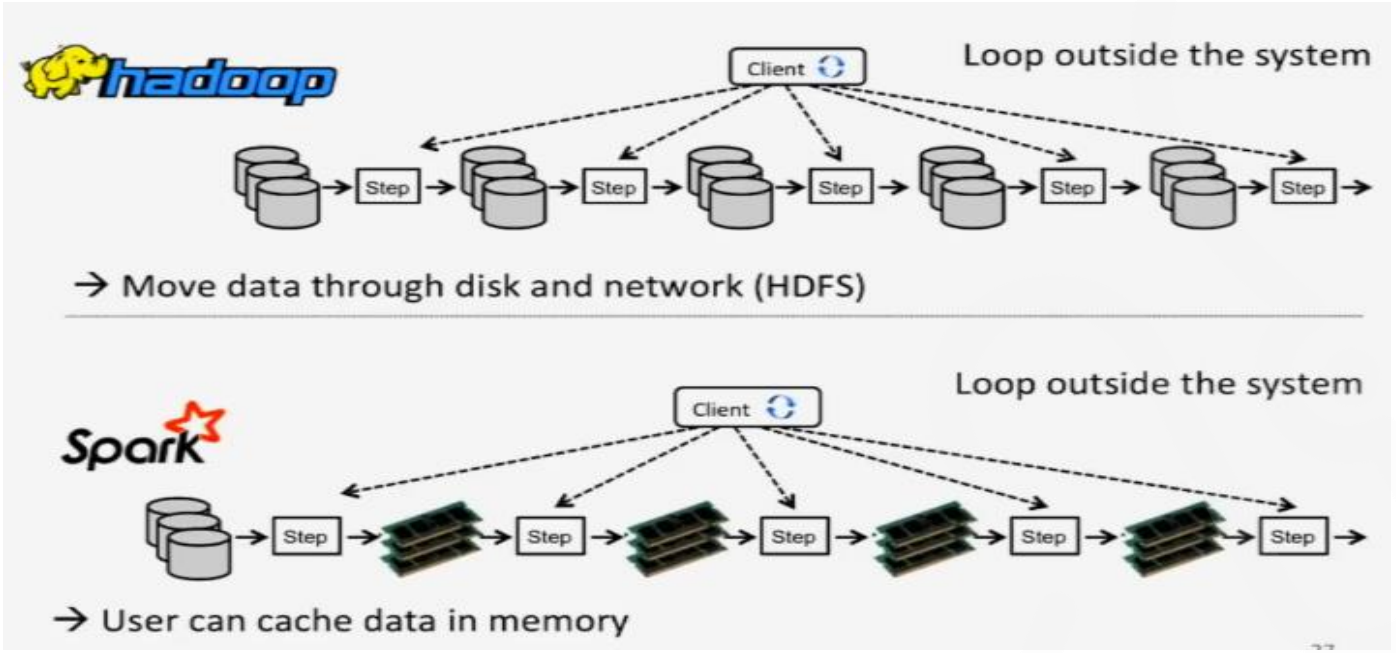
Les différences entre Hadoop et Spark pour le stockage et le traitement des données sont:

	Hadoop MR	Spark
Stockage	Disque	Disque et Mémoire
Traitements	Batch	Batch, stream, traitements interactifs
Langage d'API	Java	Scala, Java, Python

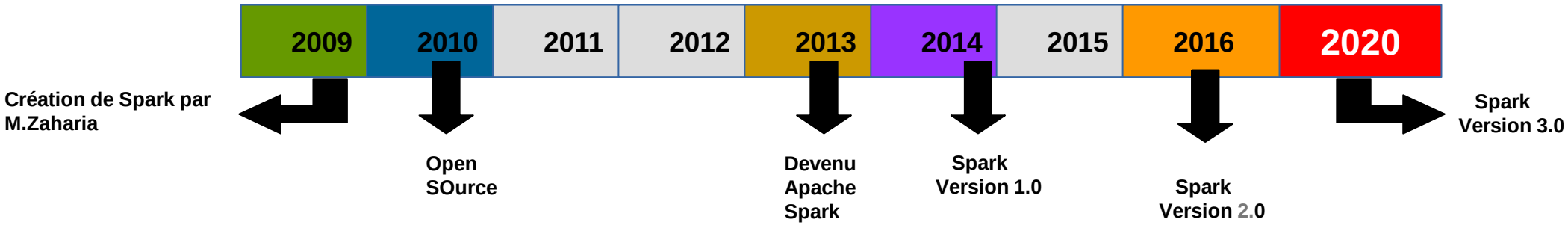
Spark est plus rapide que Hadoop: moins d'accès disque

- **100 fois** plus rapidement que Hadoop MapReduce **in-memory** et **10 fois** plus vite **sur disque**
- En 2014 record Spark (Daytona GraySort Contest)
- Tri de 100 TO de données (**Hadoop** a fait **72 min** avec **2100 cores**, **Spark** a fait **23 min** avec **206 cores)**

Spark est plus simple que Hadoop



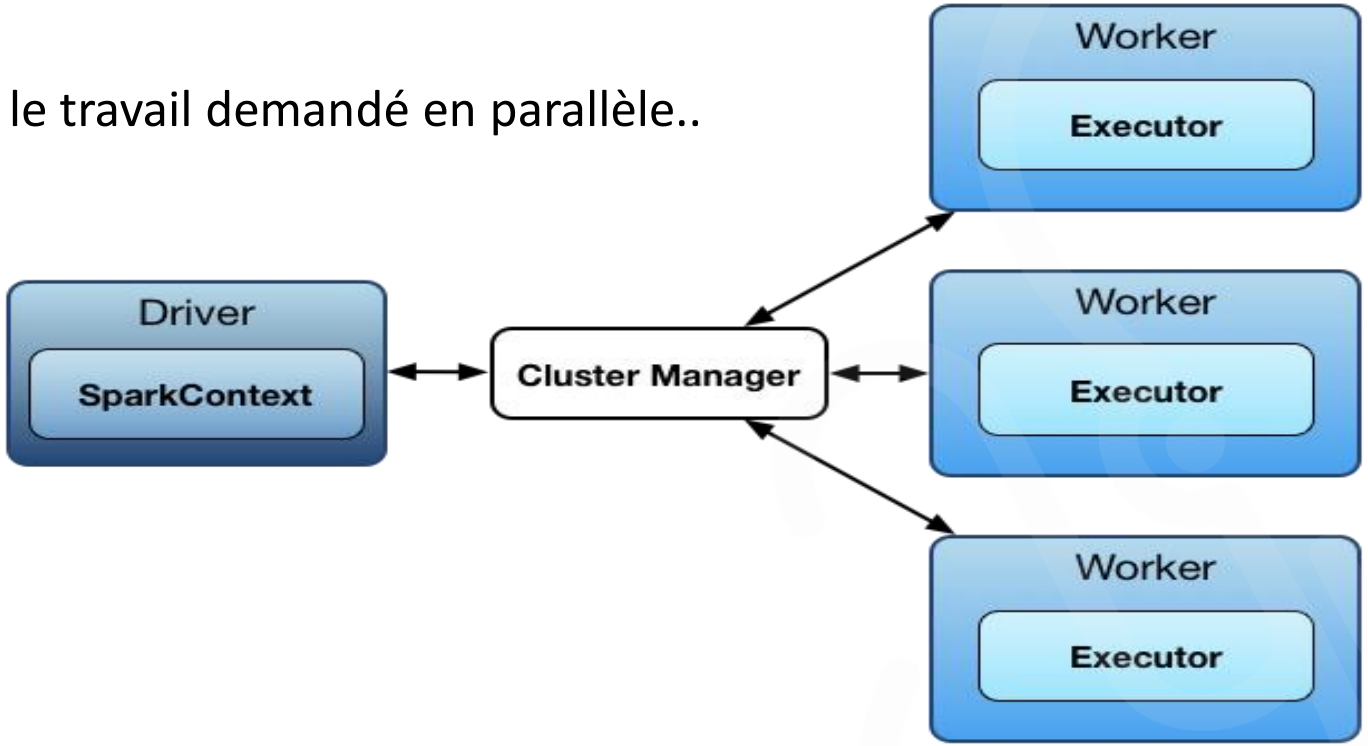
- Spark est un framework open source de calcul distribué.
- Spark fut conçu par Matei Zaharia en 2009 dans le cadre de sa thèse de doctorat à l'université de Californie à Berkeley.
- Spark est un projet de la fondation Apache depuis 2013.
- Spark est dévelopé en Scala
- La version 1.0 fut lancée en 2014
- En Juillet 2016, Apache Spark est passé en version 2.0.
- En Juin 2020 Apache Spark est passé en version 3.0



Spark se base sur une **architecture Maître / Esclave**.

Une **machine maître** (**driver**) écoute les demandes de traitement clientes, découpe chaque traitement et, délègue sa réalisation à des machines esclaves.

Les **machines esclaves** (**workers**) vont accomplir le travail demandé en parallèle..

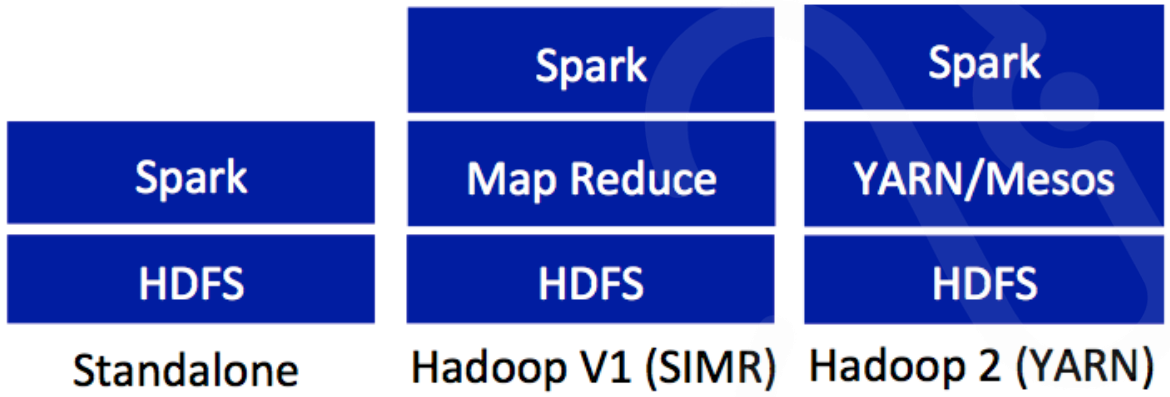


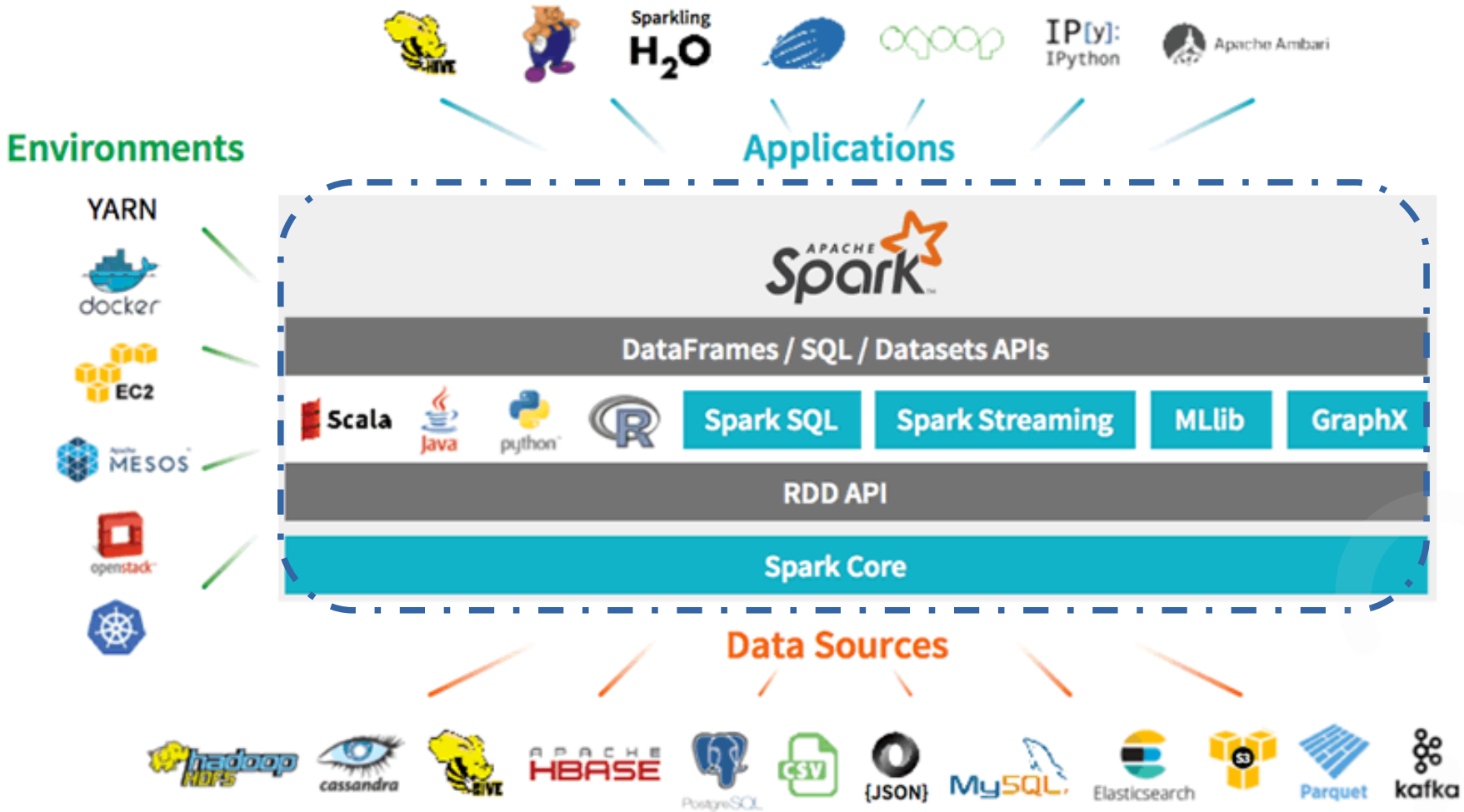
Le gestionnaire de cluster (**cluster manager**) permet de:

- exécuter des traitements lourds sur un cluster
- piloter les noeuds esclaves
- distribuer les tâches équitablement sur noeuds esclaves
- arbitrer la quantité de CPU et de mémoire qui sera allouée à chacun des traitements

Spark offre pour l'instant trois solutions de gestionnaire de cluster :

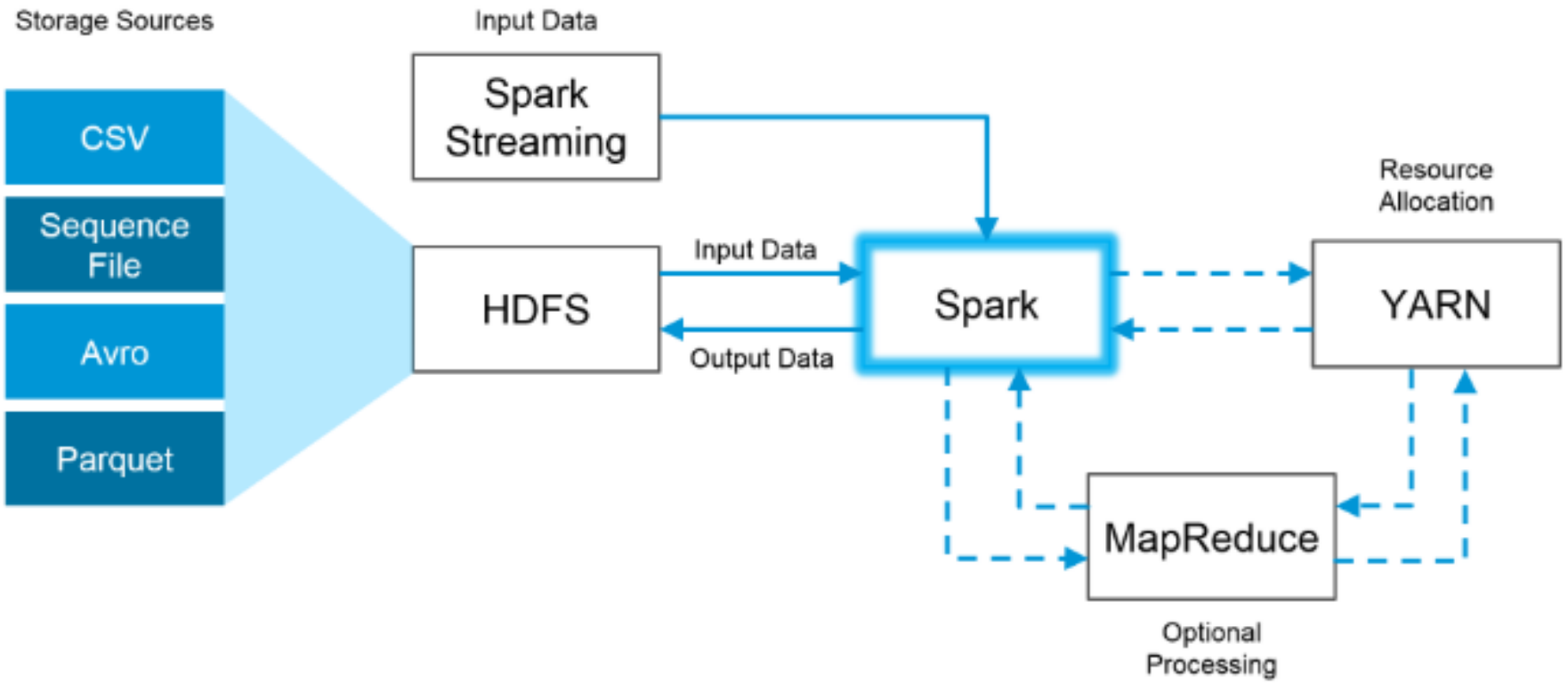
- Stand Alone
- Hadoop YARN
- Apache Mesos
- **Apache Mesos**
- **support is deprecated as of Spark 3.2.0**
- It will be removed in
- a future version





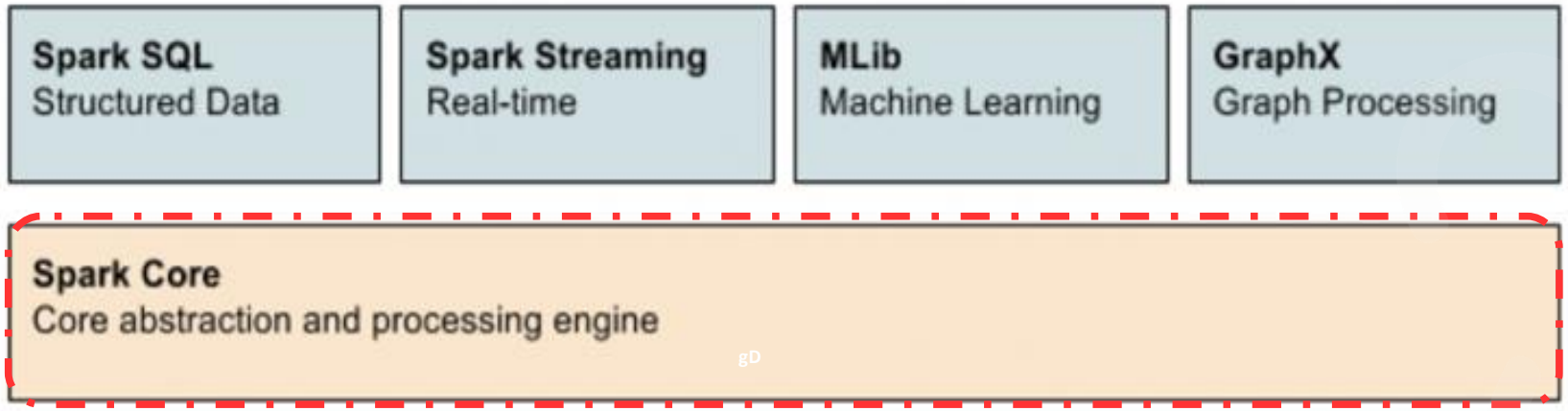
Source : Databricks

Spark –Complementarité avec Hadoop



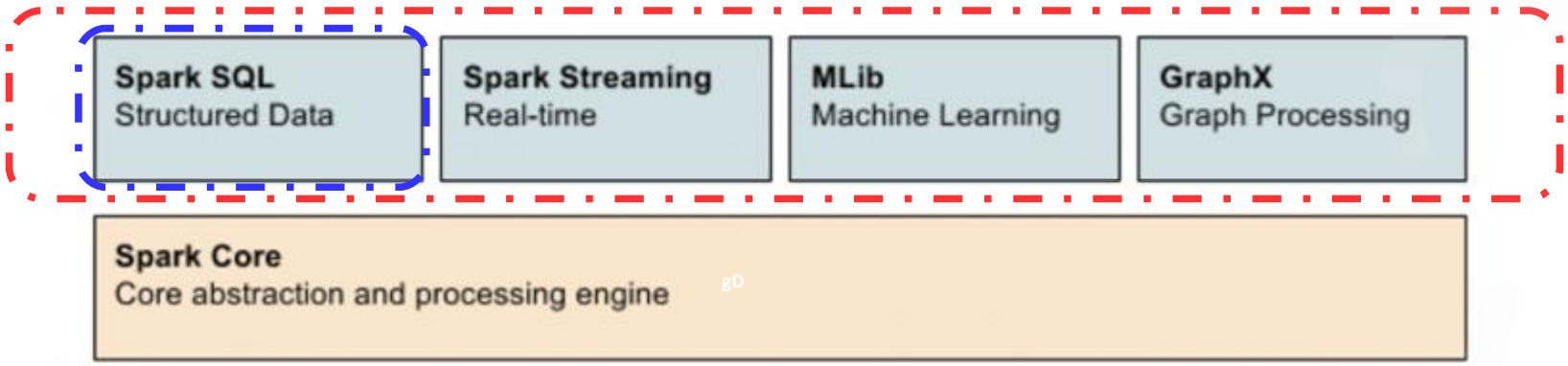
Spark core contient les fonctionnalités cœurs :

- Task Scheduling
- Memory Management
- Fault Recovery
- Interaction avec les différents types de stockages et types de données
- les API de manipulation de RDD en Scala / Python / Java
- Etc.



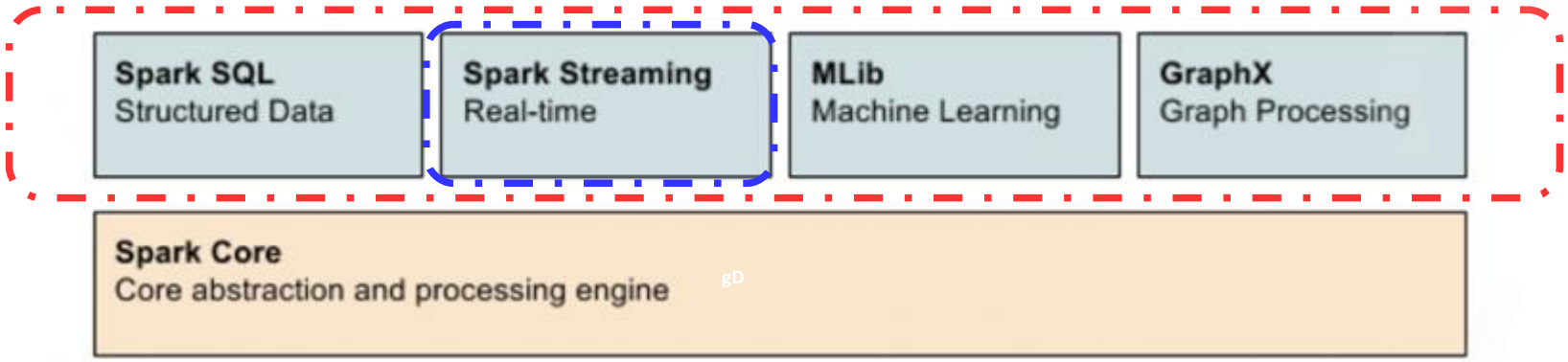
Spark SQL

- permet d'exécuter des requêtes de type SQL en utilisant les outils BI et de visualisation traditionnels
- permet d'extraire, transformer et charger des données sous différents formats (JSON, Parquet, base de données) et les exposer pour des requêtes.



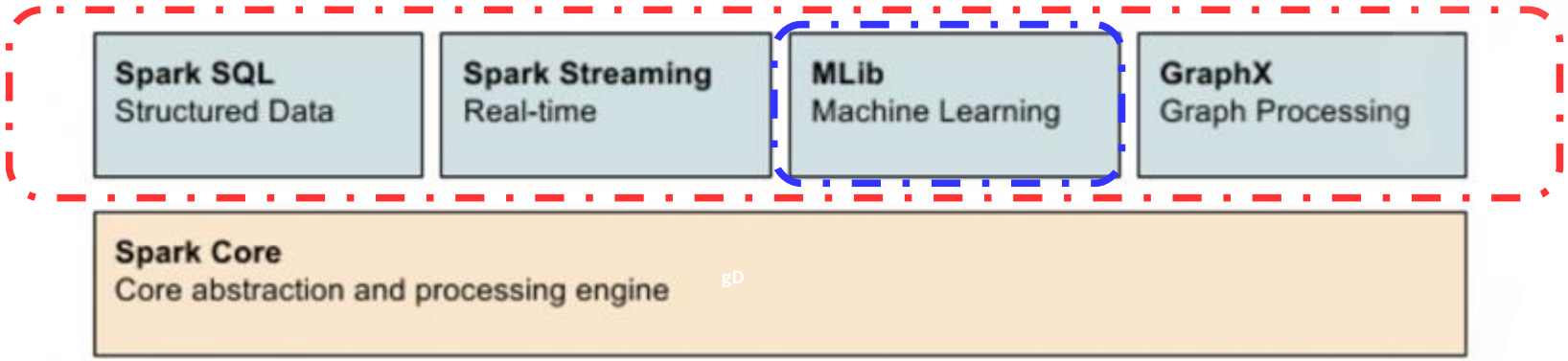
Spark Steaming

- utilisé pour le traitement temps-réel des données en flux.
- s’appuie sur un mode de traitement en "micro batch“
- utilisé pour les données temps-réel DStream, c’est-à-dire une série de RDD (Resilient Distributed Dataset).



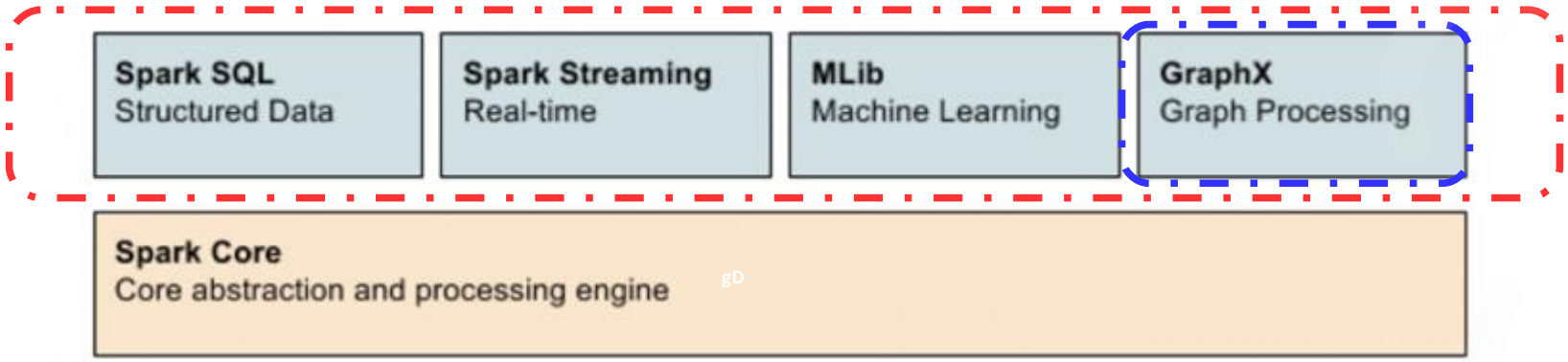
Mllib

Une librarie de machine learning qui contient tous les algorithmes et les utilitaires d'apprentissage classiques, comme la classification, la régression, le clustering, la réduction de dimensions…



GraphX

- Est une API pour les traitements de graphes et de parallélisation de graphes
- Inclut une collection toujours plus importante d'algorithmes et de builders pour simplifier les tâches d'analyse de graphes



05

Section 05

Spark

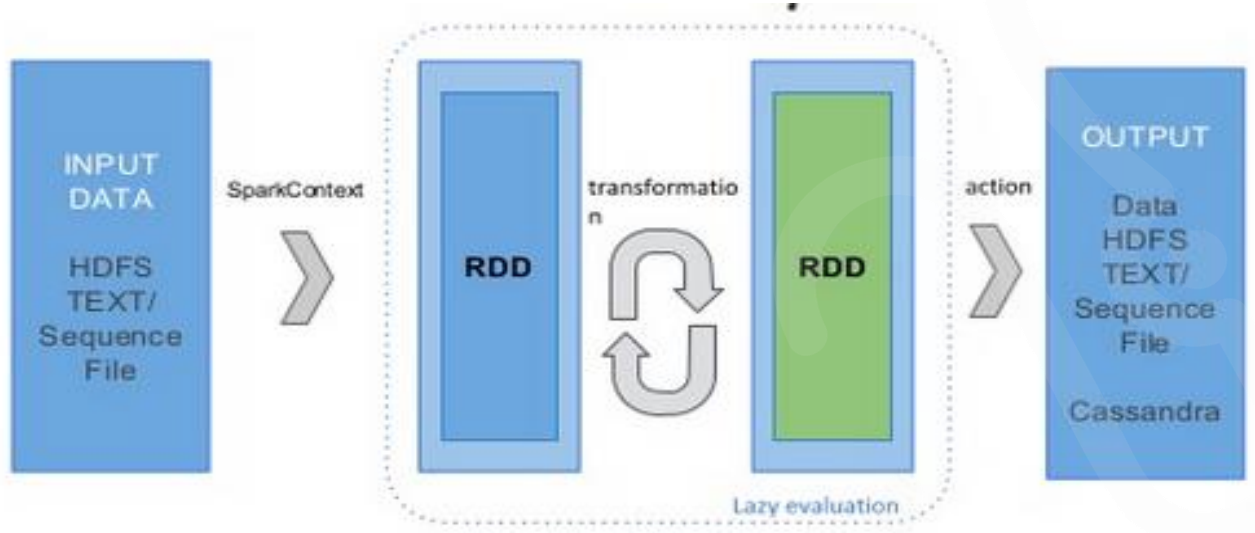
- RDD, DataFrame

RDD (Resilient Distributed Dataset) est le concept central du framework Spark

- Dataset: jeu de données qui se parcourt comme une **collection**
- Distributed: jeu de données **partitionné** et **chacune des partitions traitée sur un noeud** du cluster
- Resilient: En cas de **perte d'un noeud**, le **sous-traitement sera automatiquement relancé** sur un autre noeud

La RDD supporte 3 types d'operations

- La création
- Les transformations
- Les actions



Pour créer une RDD, on peut charger les données à partir:

- Une collection (List), transformée en RDD avec l'opérateur 'parallelize'
- Un fichier local ou distribué (HDFS) dont le format est configurable: texte brut, SequenceFile Hadoop, JSON, etc.
- Une base de données: JDBC, Cassandra, Hbase, etc.
- Un autre RDD auquel on aura appliqué une transformation

Le chemin inverse, exporter un RDD dans un fichier, dans une base de données ou une collection est aussi possible

Les transformations: elles retournent un nouveau RDD.

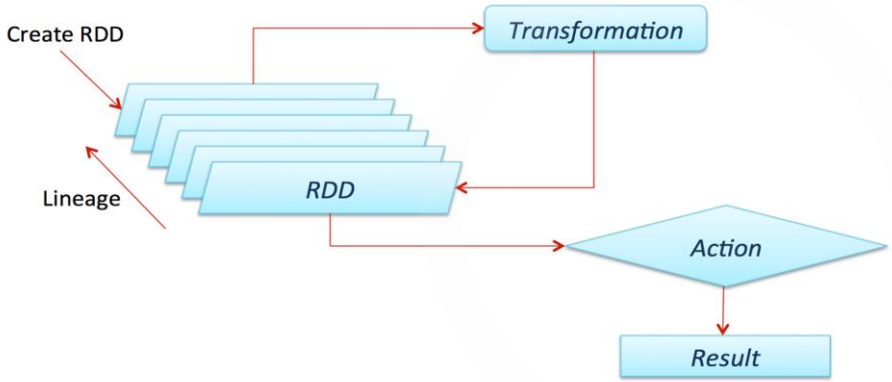
- Rien n'est évalué lorsque l'on fait appel à une fonction de transformation. Cette fonction prend juste un RDD et retourne un nouveau RDD.

Les principales fonctions de transformation sont:

- **map()** permet de transformer un élément en un autre élément
- **flatMap()** permet de découper un élément en 1 ou plusieurs autres éléments
- **mapToPair()** permet de transformer un élément en un tuple clé-valeur
- **filter()** permet de filtrer les éléments en ne conservant que ceux qui correspondent à une expression
- **reduceByKey()** permet d'agréger des éléments entre eux

Les actions: évaluent et retournent une nouvelle valeur.

- Au moment où une fonction d'action est appelée sur un objet RDD, les transformations nécessaires à l'évaluation de l'action sont calculées…



Les principales actions sont

- **count()** et **countByKey()** permet de compter les éléments
- **collect()** permet de récupérer les éléments dans une collection
- **saveAsTextFile()** permet de sauver le résultat dans des fichiers texte
- **reduce()** permet d'agréger des éléments entre eux

Un **résultat intermédiaire** peut être conservé temporairement grâce aux méthodes

- **cache()** (stockage en mémoire)
- **persist()** (stockage en mémoire ou sur disque)

Dataframe = RDD + colonnes nommées

- **Dataframe:** collection de données distribuées, organisées en colonnes (Similaire à une table)

- Peut être converties en RDD en appelant la méthode rdd, qui retourne le contenu de la DataFrame sous forme de RDD de lignes

- L'API Dataframe est disponible pour Scala, Java, Python et R.

- Peut être crée à partir de différentes sources de données :
  - des RDD existants
  - des fichiers de données structurés (csv)
  - des jeux de données JSON
  - des bases de données externes
  - des tables HIVE

Dataset: des RDDs structurés + type des colonnes est connu, Dataframe sinon.

Dataset est un dataframe dont le schéma précis est connu.

Dataset = dataframe + schéma

L'API Dataset a été introduit dans Spark à partir de la version 1.6

L'API Dataset est disponible pour Scala et Java mais pas pour Python.

06

Section 06

Spark

- Interagir avec Spark

- Spark est développé en Scala (langage de programmation fonctionnelle basé sur jvm) **« Développer »**
- Spark consiste à interagir avec Spark core (Scala) et avec des API

Les trois principales API sont écrites pour **Scala, Java, et Python** (**API R** depuis Spark 1.4)

Documentations : https://spark.apache.org/docs/

Deux façons pour exploiter ses API

- Utiliser un CLI
- Soumettre un programme en Stand Alone

	Scala	Java	Python
CLI	spark-shell		pyspark
Stand Alone	maven	maven	spark-submit

CLI : Utilisation interactive des API Spark

Examples

spark-shell (Scala)

pyspark (Python)

- Toute application Spark a besoin d'un **Spark Context:**
  - Le Spark Context est le point d'entrée
  - Spark Shell intégre un SparkContext préconfiguré « **sc** »

Utiliser le mode Standlone* revient à éxécuter un code **scala**, **python** ou **java** avec **les librairies de Spark** et définir le SparkContext manuellement.

```java
// en Java
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
SparkConf conf = new SparkConf().setMaster("local").setAppName("My App");
```

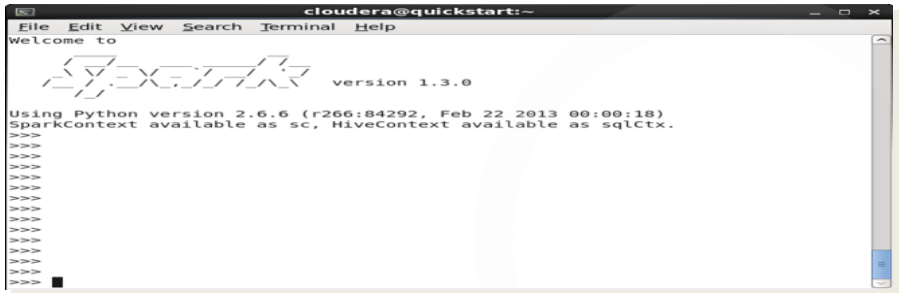
```scala
// en Scala
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
val conf = new SparkConf().setMaster("local").setAppName("My App")
```

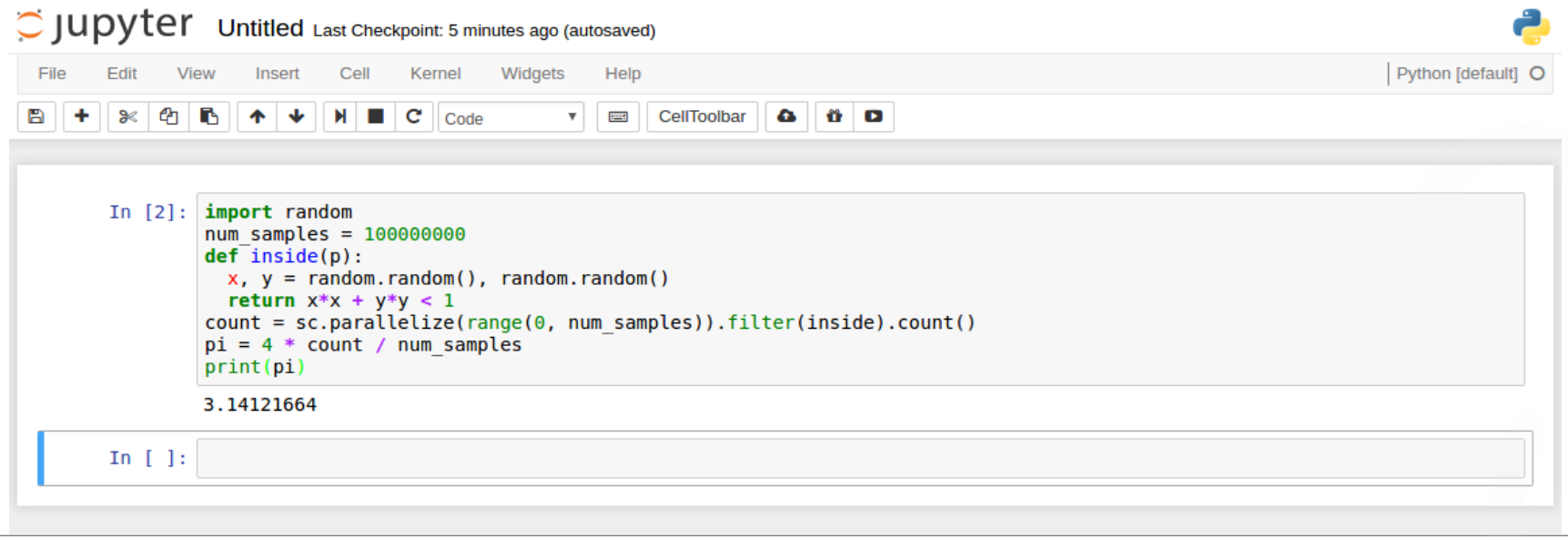
```python
// en Python
from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("My App")
sc = SparkContext(conf = conf)
```

Spark Python API est une librairie qui permet de manipuler les objets et les méthodes Spark en programmation **Python**

L'API PySpark propose une utilisation interactive via Spark Shell:

- Il permet d'établir le lien entre l'API python et Spark Core
- Il permet d'initialiser le Spark Context sous le nom "**sc**"





- Pour pouvoir utiliser PySpark, on faire l'installation sur plusieurs plate-forme

Sur windows:

https://changhsinlee.com/install-pyspark-windows-jupyter/
http://deelesh.github.io/pyspark-windows.html

http://jupyterhub.octopeek.com:8001/tree?# pour se connecter au Jupyter hub et verifier l'installation

Installation 01 : Windows ou Mac

Sur **machine virtuelle en utilisant**:

- Ubuntu
- VirtualBox

Allez sur : https://www.virtualbox.org/wiki/Downloads

Cliquez sur “Windows Host” :
https://download.virtualbox.org/virtualbox/5.2.6/VirtualBox-5.2.6-120293-Win.exe
https://download.virtualbox.org/virtualbox/6.0.8/VirtualBox-6.0.8-130520-Win.exe

Ensuite il faut aller sur le site de Ubuntu (Linux) : Ubuntu Desktop
https://www.ubuntu.com/download

Une fois VirtualBox installé, vous pouvez des à présent installer Ubuntu sur votre machine virtuelle

Configuration de la VM :
- Nom : Spark
- Type : Linux
- Version : Ubuntu (64 bits)
- Mémoire : ¼ voir 1/2 de votre mémoire machine
- Disque : VDI (VirtualBox Disk Image) > Fixed Size
  10 GB minimum

Setting :
- CPU (System) : 1/2
- Vidéo Memory (Display) : 2/3

Ensuite lancez votre VM

Installation 01 : Linux sur Windows ou Mac

Installation de Linux :
- Lancez la VM
- Chargez l'image .ISO de Ubuntu déjà téléchargée
- Install Ubuntu
- Erase disk and install Ubuntu

Dans cette partie nous allons installer ensemble environnement de développement PySpark.
Note :

- Les versions Spark < 2.2 ne sont pas compatibles avec python 3.6

Installation :

- Python : Sur Ubuntu vous allez trouver les versions 2.7 et 3.5 déjà installées

- Jupyter :

```
$ sudo apt install python3-pip
$ pip3 install jupyter
$ jupyter notebook
```

Installer Java / Scala :
$ sudo apt-get update
$ sudo apt-get install default-jre
$ sudo apt-get install scala
$ java -version

Installer la librairie Py4J qui connecte Java et Scala avec Python :
$ pip3 install py4J

Installer Spark :
Direct Download > Spark-2.4.3-bin-hadoop2.7.tgz

Mettez le fichier télécharger sur le répertoire home
$ sudo tar -zxvf spark-2.2.1…

Ajouter le PATH à ce que python retrouve spark :

```
$ export SPARK_HOME='home/ubuntu/spark-2.2.1-bin-hadoop2.7'
$ export PATH=$SPARK_HOME:$PATH
$ export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
$ export PYSPARK_DRIVER_PYTHON="jupyter"
$ export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
$ export PYSPARK_PYTHON=python3
```

Éviter les problème de permissions :

```
$ sudo chmod 777 spark-2.2.0-bin-hadoop2.7
$ sudo chmod 777 python
$ sudo chmod 777 pyspark
$ sudo chmod 777 spark-2.2.1-bin-hadoop2.7
```

Terminal : Aller sur le path python3> import pyspark

Jupyter : Aller sur le path python > jupyter notebook

Python, Spark et Jupyter (suite)

On a vu dans le slide précédant que pour utiliser PySpark il faut à chaque fois aller sur le path « python » et lancer jupyter. Hors y a moyen d'importer spark de n'importe quel path via “findspark”

Configuration :

```
$pip3 install findspark
python3> import findspark
                    findspark.init('path to spark-2…')
                    import pyspark

jupyter notebook> import findspark
                findspark.init('path to spark-2…')
                        import pyspark
```

Exercice

Lancez une session Spark (Spark Context)

```
from pyspark import SparkContext
```

07

Section 07

Spark et la donnée

- Spark python
- Spark DataFrame opérations basiques
- Spark DataFrame opérations avancéés

Rappel Python

- Opérations
- Structure de données
- Conditions et boucles
- Les fonctions
- Exercices

Rappel Python : Opérations

- Typage des données : type(1.0)
- Addition : 1+1
- Multiplication : 3*1
- Division : 1 / 2
- Puissance : 2**4
- Modulo : 4 % 2
- Caractère : 'char' ou "char"

Rappel Python : Affichage

```
print("hello")

print("hello {}".format('Alain’))

print("hello {}, mon nom est {}".format('Alain', 'Sarah’))

print("First: {x}, Second {y}".format(x='XXX', y='YYY'))
```

Liste : `Mylist = ['a','b','c’]` (Attention : le mot « list » est reservé)

indexation : *`mylist[2] = 'c'`*
`mylist[-1] = 'c’`
`mylist[0] = 'NEW’` #Remplacer
`mylist.append('d’)` #Ajout à la fin

Liste imbriquée : `mylist = [1,2,[100,200]]`

indexation : `mylist[2]`
`mylist[2][1]`

dictionnaire : `d = {'key1' : 'value', 'key2' : 'value2’}` `(d['key2’] )`

Tuple : `t = (1,2,3)`

Comparaisons : `'a' == 'a'` (sensible à la casse)

```
1 != 2
('a' == 'a') and/or (1 == 2)
```

Conditions :

```
if 1 == 2:
        print('yea!')
elif 3==3:
        print("deuxieme condition juste")
else :
        print('nop’)
```

Boucles :

```
seq = [1,2,3,4,5]
for i in seq:
        print(i)
```

While :

```
i=1
while i<5:
        print('i is {}'.format(i))
        ii = i+1
```

Python3 :

```
x = [1,2,3,4]
list(range(5))
list(range(2, 5))

out = []
for i in x:
      out.append(i**2)
print(out)
```

Rappel Python : Fonction

1.
```
def my_func():
        print('hello’)

Call : my_func()
```

2.
```
def my_func('vous'):
      print('hello'+vous)

  Call : my_func('Alain')
```

3.
```
def square(var):
      retrun var**2

result = square(2)
```

```
st = 'hello my name is Sam'
st.lower()
st.upper()
st.split()          # on met la caractère de la séparaton entre parenthèse (ici c'est espace)
```

dictionnaire :

```
d = {'key1' : 'value', 'key2' : 'value2'}
d.keys()
d.items()
```

08

Section 08

Spark et la donnée

- Spark DataFrame opérations basiques

Spark DataFrame

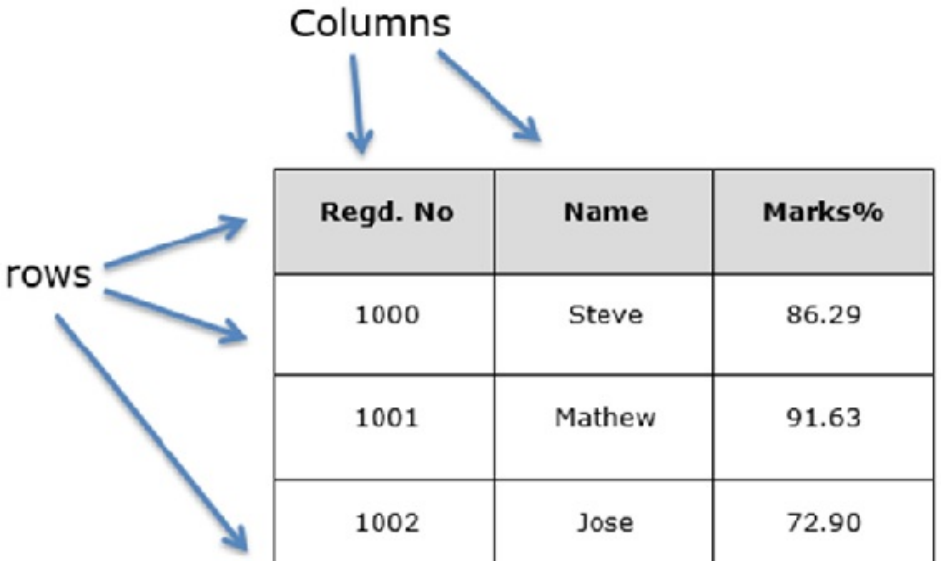
- Spark a commencé avec les RDD dont la syntaxe est un peu difficile !

- Spark.2 est allé vers le syntaxe dataframe qui est beaucoup plus facile pour la manipulation de données

- Spark DF sont capables d'importer et exporter la donnée provenant d'une grande variété de sources.

- Nous pouvons ensuite utiliser ces DataFrames pour appliquer diverses transformations sur les données.

- Spark DataFrames contiennent des données dans un format de colonne et de ligne.
- Chaque colonne représente une caractéristique ou une variable ou un feature.
- Chaque ligne représente un point de données individuel ou une observation.

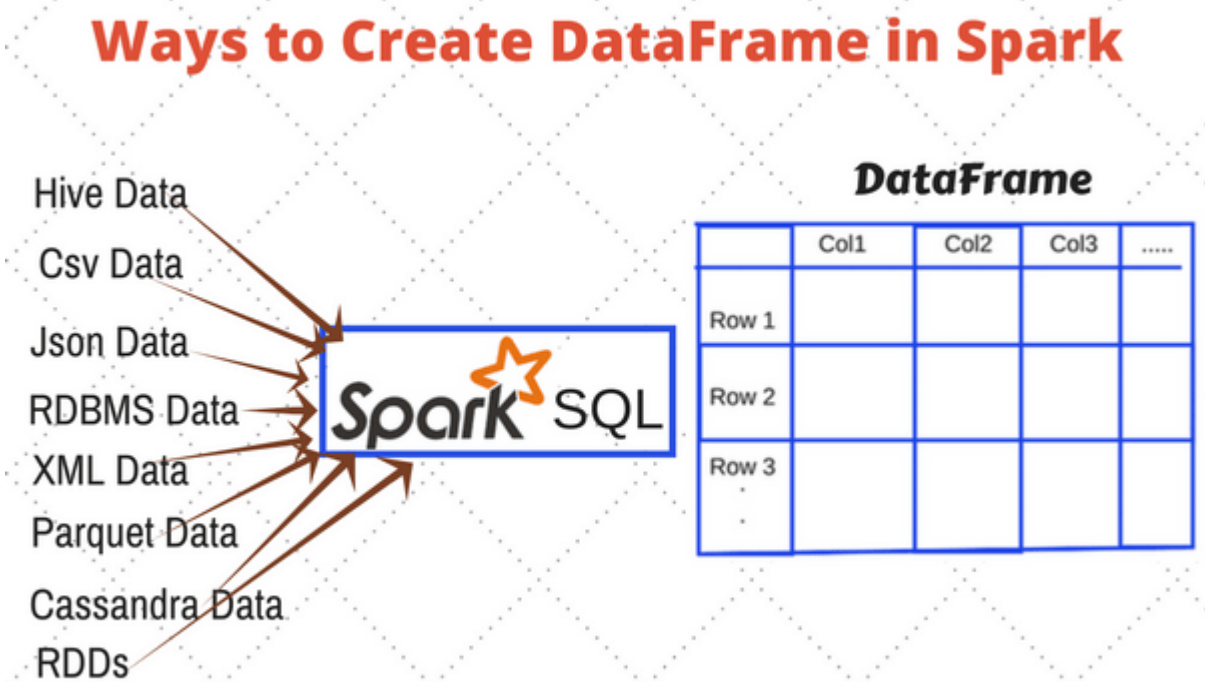


Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

Comment créer un DataFrame?

Un DataFrame dans Apache Spark peut être créé de plusieurs manières:

- Il peut être créé en utilisant différents formats de données. Par exemple, chargement des données à partir de JSON, CSV.
- Chargement de données depuis un RDD existant.
- Spécification par programme du schéma



Spark DataFrame : Fichier JSON

- TP avec les fichiers sur ‘Data’

- Pour utiliser Spark on a besoin de passer par « Spark Session »

```
from pyspark.sql import SparkSession
```

Detecter Path pour pyspark

```
import findspark
findspark.init()
```

Créer un context:

```
spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext
```

- Lecture du DF de type JSON:

```
df = spark.read.json(‘path/people.json’)
```

Spark DataFrame : Fichier JSON

```
from pyspark.sql.types import StructField,StringType, IntegerType, StructType

data_schema = [StructField('age', IntegerType(), True),
               StructField('name', StringType(), True)]
# Changer de type
final_struc = StructType(fields=data_schema)
df = spark.read.json('people.json', schema=final_struc)
df.printSchema()
```

- Afficher le DF et les noms de colonne :

```
df.show()
df.columns
```

Connaitre le schéma du DF :

```
df.printSchema()
```

- Description de la données :

```
df.describe().show()
```

- Créer un echantillon

```
df_sample = df.sample(False, 0.2, 42)
```

- Nature de la colonne :

```
df.select('age').show()
```

- Afficher n lignes :

```
df.head(1)
```

- Créer une colonne en copiant la première :

```
df.withColumn('newage', df['age']).show()
```

- Renommer la colonne :

```
df.withColumnRenamed('age', 'my_new_age').show()
```

09

Section 09

Spark et la donnée

- Spark DataFrame opérations avancéés

Spark DataFrame : Aggrégation & GroupBy

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('aggs').getOrCreate()

df = spark.read.csv('/path/sales_info.csv', inferSchema=True, header=True)
df.show()
df.printSchema()
```

- GroupBy : Une variable

```
df.groupBy('Company').mean().show()     #Pareil pour count(), max(), min(), sum()
```

- Si on veut des select sur une colonne sans passer par GroupBy --> Aggregate

```
df.agg({'Sales':'sum'}).show()          #somme d'une colonne
```

- Importer des fonctions :

```
from pyspark.sql.functions import countDistinct,avg,stddev
```

- Count distinct :

```
df.select(countDistinct('Sales')).show()
```

- modifier le nom de la variable après une opération :

```
df.select(avg('Sales').alias('AVG')).show()
```

- formatage (num):

écart type : `df.select(stddev('Sales')).show()`

```
from pyspark.sql.functions import format_number
sales_std= df.select(stddev("Sales").alias('std'))
sales_std.select(format_number('std',2)).show()
```

- orderBy:

```
df.orderBy("Sales").show()
df.orderBy(df['Sales'].desc()).show()
```

Spark DataFrame : Filtre

- Filtre :

```
df.filter("Close < 500").show()                # un select sur tout les article < 500 euros   |ou|
df.filter(df['Close'] < 500).show()
df.filter("Close < 500").select("Open").show()          #même chose, mais en sélectionnant que l
```

- colonne Open

```
df.filter("Close < 300").select(["Open", "Close"]).show()  #Multiple colonne
```

- Filtre avec double condition :

```
df.filter((df['Close'] <200) & (df['Open'] > 200)).show()
```

- Filtre avec equal :

```
df.filter(df['Low']==197.16).show()
```

- En collect :

```
result = df.filter(df['Low']==197.16).collect()
```

- Affichage :

```
row = result[0]
row.asDict()
row.asDict()['Volume']
```

Trois options :

Les garder
Les supprimer
Les remplacer

```
df = spark.read.csv('/path/ContainsNull.csv', inferSchema = True, header=True)
df.show()
df.na.drop()
```

- Supprimer les lignes avec x nulls

```
df.na.drop(thresh=x).show()
```

- Supprimer les lignes avec toutes les valeurs en null :

```
df.na.drop(how='all').show()
```

- Supprimer des ligne avec une colonne précise avec Null :

```
df.na.drop(subset=['Sales']).show()
```

- Remplacer des null avec des string sur les colonnes de nature String :

```
df.na.fill('Fill Value').show()
```

- Remplacer des null avec des numéro sur les colonnes de nature num :

```
df.na.fill(0).show()
```

-Remplacer des null avec des string sur les colonnes de nature String (avec la précision de la colonne) :

```
df.na.fill('Fill Value', subset=['Name']).show()
```

- Moyenne:

```
from pyspark.sql.functions import mean
mean_val = df.select(mean(df['Sales'])).collect()
mean_val
```

- Remplacer les null par une moyenne :

```
mean_sales = mean_val[0][0]
df.na.fill(mean_sales,['Sales']).show()
```

```
f = spark.read.csv('/Path/appl_stock.csv', inferSchema = True, header=True)

from pyspark.sql.functions
import (dayofmonth, hour, dayofyear, month, year, weekofyear, format_number, date_format)
```

- Avoir le num du jour :

```
df.select(dayofmonth(df['Date'])).show()
```

- Avoir le l'heure du jour / mois / année:

```
df.select(hour(df['Date'])).show()
df.select(month(df['Date'])).show()
df.select(year(df['Date'])).show()
```

- Récupérer l'année d'un timestramp et le mettre dans une nouvelle colonne :

```
df.withColumn("YYYY", year(df['Date'])).show()
```

- Moyenne d'une collone de l'année :

```
newdf = df.withColumn("YYYY", year(df['Date']))
newdf.groupBy("YYYY").mean().show()
newdf.groupBy("YYYY").mean().select(['YYYY', "avg(Close)"]).show()
result = newdf.groupBy("YYYY").mean().select(['YYYY', "avg(Close)"])
```

- Renommer :

```
result.withColumnRenamed("avg(Close)", "AVG").show()
```

- Formater

```
new_result.select(format_number('AVG', 2)).show()
```

Dataframe : Exercices

Fichier : Walmart Stock (2012-2017) : File calmart_stock.csv
Questions :
1. Créez une spark session
2. Lire le fichier csv associé
3. Affichez les noms des colonnes
4. Affichez le schéma du fichier
5. Affichez les 5 premières lignes
6. Utilisez describe () pour en savoir plus sur le DataFrame
7. Calculez mean et stddev dans le dataframe describe (). Formatez les nombres pour qu'ils affichent seulement deux décimales.
   nb. Attentions aux types de données que .describe () renvoie

10

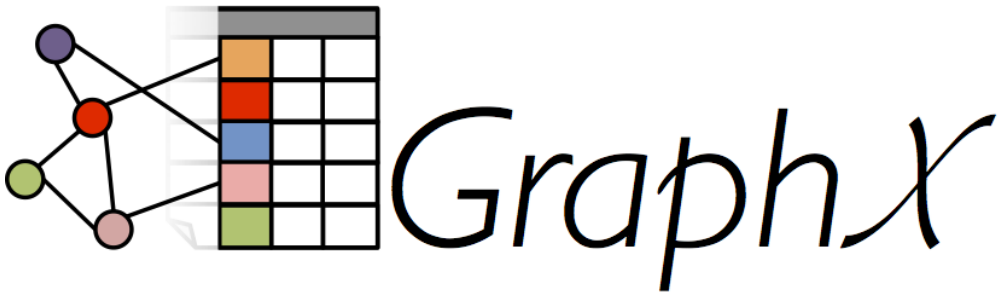
Section 10

Spark et les Graphes

GraphX est l'API d'Apache Spark pour les graphes et le calcul parallèle.

GraphX unifie les processus ETL (Extract, Transform & Load), l'analyse exploratoire et le calcul de graphes itératif au sein d'un même système.

L'utilisation des graphiques est visible dans les amis de Facebook, les connexions LinkedIn, les routeurs Internet, les relations entre les galaxies et les étoiles en astrophysique et dans les cartes de Google. Même si le concept de calcul de graphe semble très simple, les applications de graphes sont littéralement illimitées



GraphX est l'API Spark pour les graphes et le calcul parallèle de graphes. Il comprend une collection croissante d'algorithmes de graphes et de générateurs pour simplifier les tâches d'analyse de graphes.

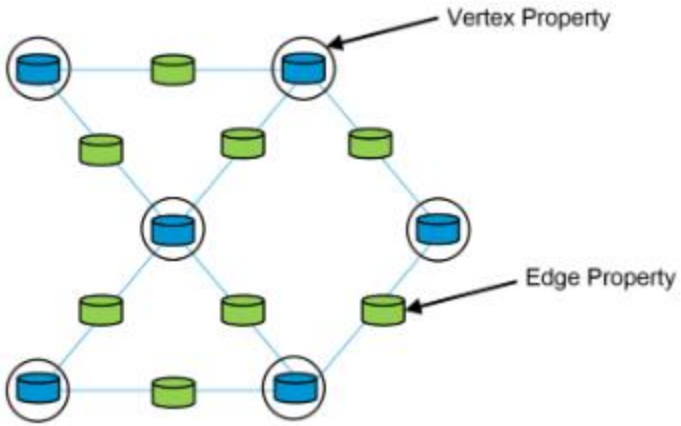


Figure: *Property Graph*

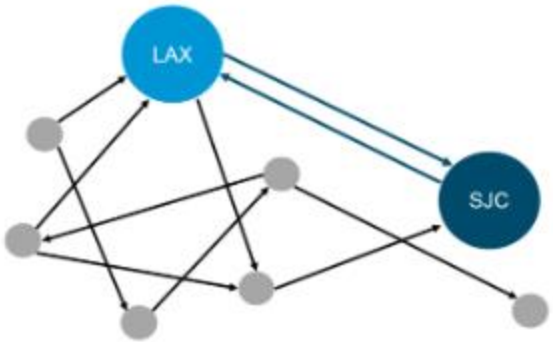


Figure: *An example of property graph*

GraphX étend le RDD Spark avec un graphe de propriétés distribuées résilient. Le graphe de propriétés est un multigraphe dirigé qui peut avoir plusieurs arêtes en parallèle. Chaque arête et chaque sommet sont associés à des propriétés définies par l'utilisateur. Les arêtes parallèles permettent plusieurs relations entre les mêmes sommets.

Un graphe est une structure mathématique constituant un ensemble d'objets dans lequel certaines paires d'objets sont liées dans un sens. Ces relations peuvent être représentées à l'aide d'arêtes et de sommets formant un graphique. Les sommets représentent les objets et les bords représentent les différentes relations entre ces objets.

Figure: *Spark GraphX Tutorial – Vertices, edges and triplets in graphs*

Concepts de graphes - Sommets, arêtes et triplets dans les graphiques

En informatique, un graphe est un type de données abstrait destiné à implémenter les concepts de graphe non orienté et orienté à partir des mathématiques, en particulier le domaine de la théorie des graphes. Une structure de données de graphique peut également associer à chaque bord une valeur de bord, telle qu'une étiquette symbolique ou un attribut numérique (coût, capacité, longueur, etc.).

Un graphe ?

- On appelle **graphe** tout ensemble de points, dont certaines paires sont directement reliées par un (ou plusieurs) liens(s).
- Un graphe sert à manipuler des concepts, et à établir un lien entre ces concepts.

- Un problème comportant des objets avec des relations entre ces objets peut être **modéliser par un graphe.**

- Les graphes sont des outils très puissants et largement répandus qui se prêtent bien à la résolution de nombreux problèmes.

- **Dans un graphe, on appelle :**
-
  - les points des **nœuds** ou des **sommets**
  - les liens sont nommés des **arêtes**

Use cases graphes

- **Réseaux sociaux**
  - Les sommets sont des individus
  - Les arêtes sont les relations entre les individus
- **Ordonnancement de tâches**
- **Minimisation de l'usage de ressources avec les arbres couvrants minimaux**
- **Recherche de plus court chemin dans un graphe**

- **GraphX facilite l'analyse des données avec les opérateurs et les algorithmes intégrés. Il permet également de mettre en cache les donnée pour éviter le recalcul lorsqu'il faut appeler un graphique plusieurs fois.**

- **GraphX**

  - est une API pour les traitements de graphes et de parallélisation de graphes

  - étend les RDD de Spark en introduisant le Resilient Distributed Dataset Graph, un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arrêtes.

  - Pour le support de ces traitements, GraphX expose un jeu d'**opérateurs** de base (par exemple subgraph, joinVertices, aggregateMessages), ainsi qu'une variante optimisée de l'API Pregel.

  - inclut une collection toujours plus importante d'**algorithmes** et de builders pour simplifier les tâches d'analyse de graphes.

GraphX – Gestion du graphe

- **Le graphe de propriétés**
  - Est un multi-graphe dirigé avec des objets définis par l'utilisateur attachés à chaque arête et à chaque sommet
  - Multi-graphe signifie qu'il peut y avoir plusieurs arêtes partageant la même source et destination
  - Chaque sommet a une clé unique VertexID
  - Chaque arête a l'ID du sommet source et destination
- Comme les RDD, les graphes de propriétés sont :
  - **Non modifiables** : les changements de valeurs ou de la structure du graphe se font en produisant un nouveau graphe.
  - **Distribués** : le graphe est partitionné en utilisant un ensemble d'heuristiques pour le partitionnement des nœuds.
  - **Résistant aux pannes** : comme avec RDD, chaque partition sur le graphe peut etre recrée sur une autre machine pour la tolérance aux pannes

Opérateurs de graphe

- Les graphes de propriété sont une collection d'**opérateurs de base** produisant un nouveau graphe avec des propriétés et structures différentes

- Les **opérateurs de base** sont définis dans *Graph* et les opérateurs composés dans *GraphOps*

- Grâce à *implicit en Scala,* les opérateurs de *GraphOps* sont disponibles dans *Graph*

- On trouve les oparateurs d'information, les opérateurs de transformation, les opérateurs de structure, les opérateurs de jointure et les opérateurs d'agrégation

Opérateurs de graphe

- **Opérateurs d'information** → numEdges, numVertices, inDegrees, OutDegrees, degree
- **Opérateurs de transformation** : utilisés pour initialiser le graphe pour un calcul
  - → partitionBy, mapVertices, mapEdges, mapTriplets
- **Opérateurs de structure**
  - *reverse* → retourne un nouveau graphe en inversant la direction des arêtes
  - *subgraph* → prend des prédicats sur sommets et arêtes et retourne le graphe contenant les sommets faisant les prédicats et reliés par les arêtes
  - mask → retourne un sous-graphe correspondant à l'intersection d'un graphe donné et d'un graphe-masque
  - *GroupEdges* → pour un multi-graphe, fusionne les différentes arêtes entre 2 sommets en une seule
- **Opérateurs de jointure**
  - *joinVertices* → joint les sommets avec le RDD en entrée et retourne un nouveau graphe avec les propriétés obtenues en appliquant la fonction map aux sommets joignant
- **Opérateurs d'agrégation**
  - *aggregateMessage* → applique une fonction sendMsg (map) définie par l'utilisateur à chaque triplet arête puis utilise mergeMsg(reduce) pour agréger ces messages pour le sommet destination

Algorithmes

- **Le Spark GraphX est livré avec un ensemble d'algorithmes de graphes pré-construits pour aider au traitement et à l'analyse des données graphiques.**

- Ces algorithmes sont disponibles dans le package *org.apache.spark.graphx.lib*. C'est aussi simple que d'appeler ces algorithmes comme méthodes dans la classe Graph.

- Les algorithmes graphiques aident à exécuter les analyses sur des ensembles de données graphiques sans avoir à écrire des propres implémentations de ces a.

- Liste de divers algorithmes qui aident à trouver des modèles dans les graphiques.
  - PageRank
  - Connected component
  - Label propagation
  - Strongly connected components
  - Triangle count
  - Single-Source-Shortest-Paths
  - Community Detection

PageRank

•L'algorithme PageRank est utilisé pour déterminer l'importance relative d'un objet dans un ensemble de données de graphe. Il mesure l'importance de chaque nœud dans un graphique

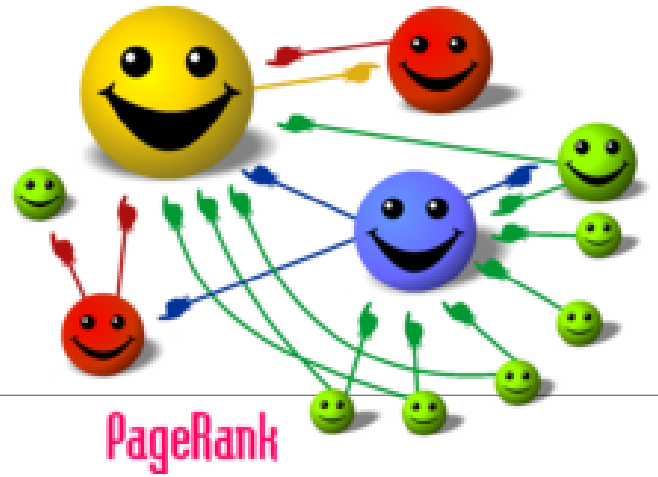
•

•Le moteur de recherche de Google est un exemple classique de PageRank.→ Google utilise PageRank comme l'une des mesures permettant de déterminer l'importance d'une page Web en fonction du nombre d'autres pages Web référencées.

•

•Un autre exemple est le site Web d'un réseau social comme Twitter → Si un utilisateur de Twitter est suivi par beaucoup d'autres

• utilisateurs, alors cet utilisateur a une plus grande influence sur le réseau.



•Cette métrique peut être utilisée pour la sélection / le placement des annonces

• auprès des utilisateurs qui suivent le premier utilisateur

PageRank

- Pour estimer le rand d'une page, la formule suivante est utilisée.

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

An equation from an early PageRank paper.

- GraphX fournit deux implémentations de PageRank:
- **Statique et Dynamique**.

- **PageRank statique**: Cet algorithme s'exécute pour un nombre fixe
- d'itérations pour générer des valeurs PageRank pour un ensemble
- donné de noeuds
- dans un ensemble de données graphique.

- **PageRank dynamique**: D'autre part, l'algorithme PageRank dynamique fonctionne jusqu'à ce que les valeurs PageRank convergent en fonction d'une valeur de tolérance prédéfinie

11

Section 11

Spark et les Streaming

Spark Streaming est une extension de l'API Spark qui permet un traitement évolutif, à haut débit et tolérant aux pannes, des flux de données en direct.

Les données peuvent être ingérées à partir de nombreuses sources comme les sockets Kafka, Flume, Kinesis ou TCP, et peuvent être traitées en utilisant des algorithmes complexes exprimés avec des fonctions de haut niveau comme map, reduce, join et window.


Kafka
Flume
HDFS/S3
Kinesis
Twitter
Spark Streaming
HDFS
Databases
Dashboards

En interne, Spark Streaming reçoit des flux de données d'entrée en direct et divise les données en lots, qui sont ensuite traités par le moteur Spark pour générer le flux final de résultats par lots.

Spark Streaming a une vue des données différente de celle de Spark. Dans Spark non diffusé en continu, toutes les données sont placées dans un jeu de données distribué résilient, ou RDD.

Cela ne suffit pas pour le streaming. Dans Spark Streaming, le nom principal est DStream - Discretized Stream. C'est essentiellement la séquence des RDD.

Les verbes sont à peu près la même chose - de la même manière que nous avons des actions et des transformations avec les RDD, nous avons également des actions et des transformations avec DStream.

Un RDD est un conteneur qui contient toutes vos données, comme une sorte de tableur.

Un D-Stream est une feuille de calcul, puis une autre, puis une autre à mesure qu'elles se déplacent dans le temps. Nous obtenons un flux de données, nous le discrétisons, nous le gelons à un moment, puis à un autre, puis à un autre. En un sens, il n'est pas nécessaire d'attendre que toutes les données s'empilent pour former un seul RDD géant, nous prenons une partie du RDD, puis nous effectuons rapidement le traitement, puis nous donnons un peu plus de données et le traitement se poursuit au petit partie des données.

De cette façon, nous n'avons pas besoin d'attendre que toutes les données donnent des réponses, nous pouvons arriver à une réponse sur la base des données dont nous disposons.

Exemple Spark Streaming :

Parce que nous utilisons Spark Streaming et non le streaming structuré (encore expérimental) nous devons utiliser la syntaxe "RDD".

Cela provient de l'utilisation d'un SparkContext au lieu d'une SparkSession.

Nous allons construire une application très simple qui se connecte à un flux de données local (un terminal ouvert) via une connexion socket.

Il comptera ensuite les mots pour chaque ligne que nous tapons.

Spark Streaming Exemple :

Les étapes pour le streaming seront:

- Créer un SparkContext
- Créer un StreamingContext
- Créer un flux de données
- Lire dans les lignes comme un « Dstream » et appliquer l'action souhaitée