
Panorama de l'écosystème Hadoop

DRIO 5101A : Big Data Analytics avec Spark
Thibaud Vienne - ESIEE Paris

Fonctionnement du cours

- 11 heures de cours :
 - Big Data : Une introduction (2h)
 - Panorama de l'écosystème Hadoop (3h)
 - Le framework Spark (3h)
 - Machine learning avec Spark (3h)

- 15 heures de TP :
 - TP 1 : Tutorial + Comptage de mots avec Spark.
 - TP 2 : Analyse de logs Apache.
 - TP 3 : Prédiction de dates de sorties de chansons.
 - TP 4 : Classification de clics internet.

- Evaluation finale

Sommaire

1. Hadoop: présentation et généralités
2. HDFS
3. Hadoop MapReduce
4. Panorama des projets Hadoop
5. Conclusion

Hadoop : présentation et généralités

Hadoop : généralités

- Hadoop est un Framework Big Data open source écrit en Java.
- Il est créé en 2005 par Doug Cutting, ingénieur chez Yahoo.
- C'est un framework créé pour effectuer du stockage distribué et des traitements distribués sur de très grands jeux de données.
- Il est actuellement le framework phare pour effectuer des projets Big Data.



Hadoop : historique

2004: Publication par Google de 3 « White papers » portant sur le stockage distribué et le calcul distribué.

2005: Doug Cutting, travaillant chez Yahoo, s'inspire de ces white papers pour commencer à développer le framework « Hadoop ».

2009: Le code source de Hadoop est rendu public et devient un projet Open-Source sous fondation Apache.

2011: Première version stable, Hadoop 1.0.0

2014: Hadoop version 2.0.0

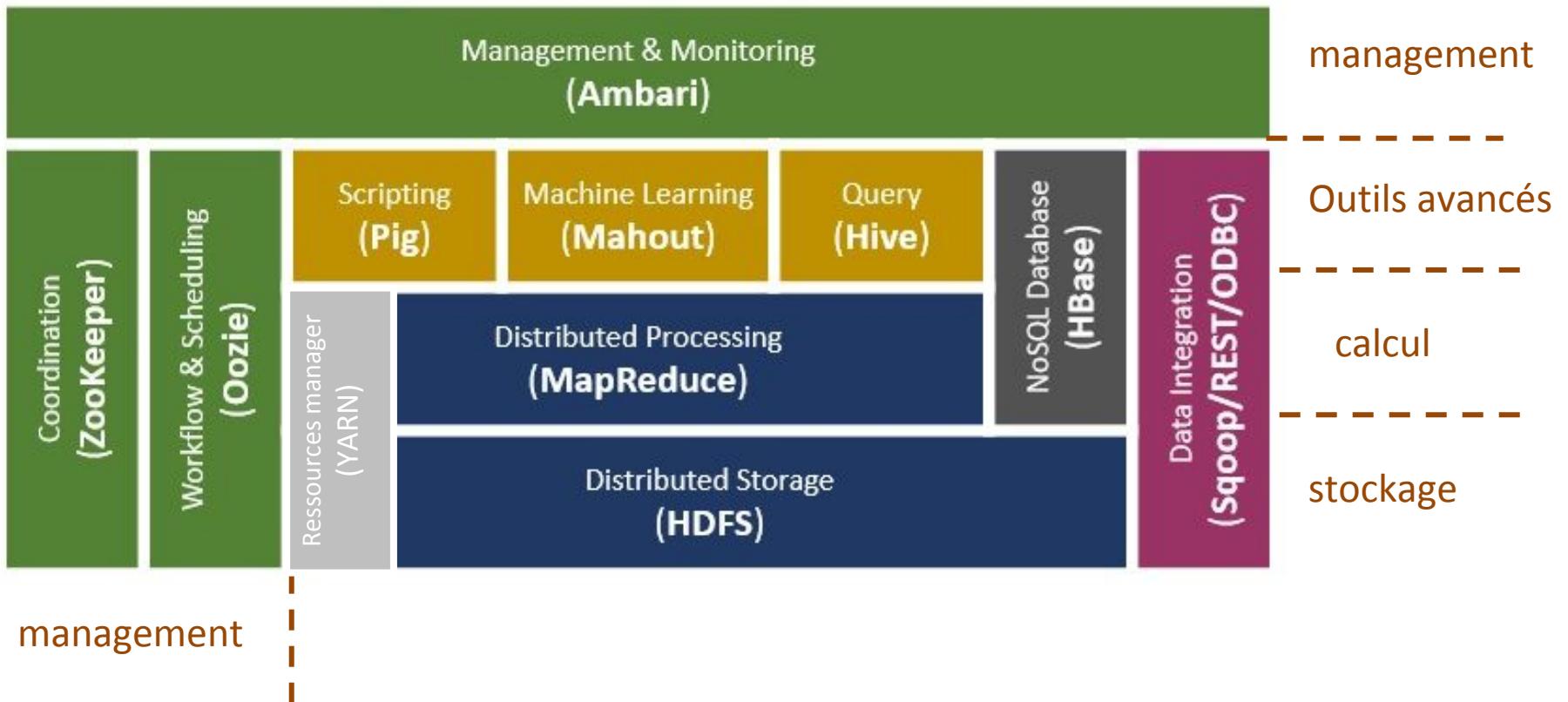


Hadoop : objectifs

- Faciliter le stockage de larges jeux de données structurés / non structurés.
- Pouvoir effectuer des calculs rapidement sur ces données grâce à des algorithmes de calculs distribués (Mapreduce).
- Être efficace quelque que soit le nombre de machines (scalabilité).
- Forte Transparence et abstraction pour l'utilisateur.
- Interopérabilité des différentes briques projets.
- Tolérance aux pannes.

Hadoop : un framework en briques

Apache Hadoop Ecosystem



Hadoop : composants principaux



Stockage de
données massives



Traitement (calcul)
sur ces données



HDFS

Architecture HDFS

Stockage des données

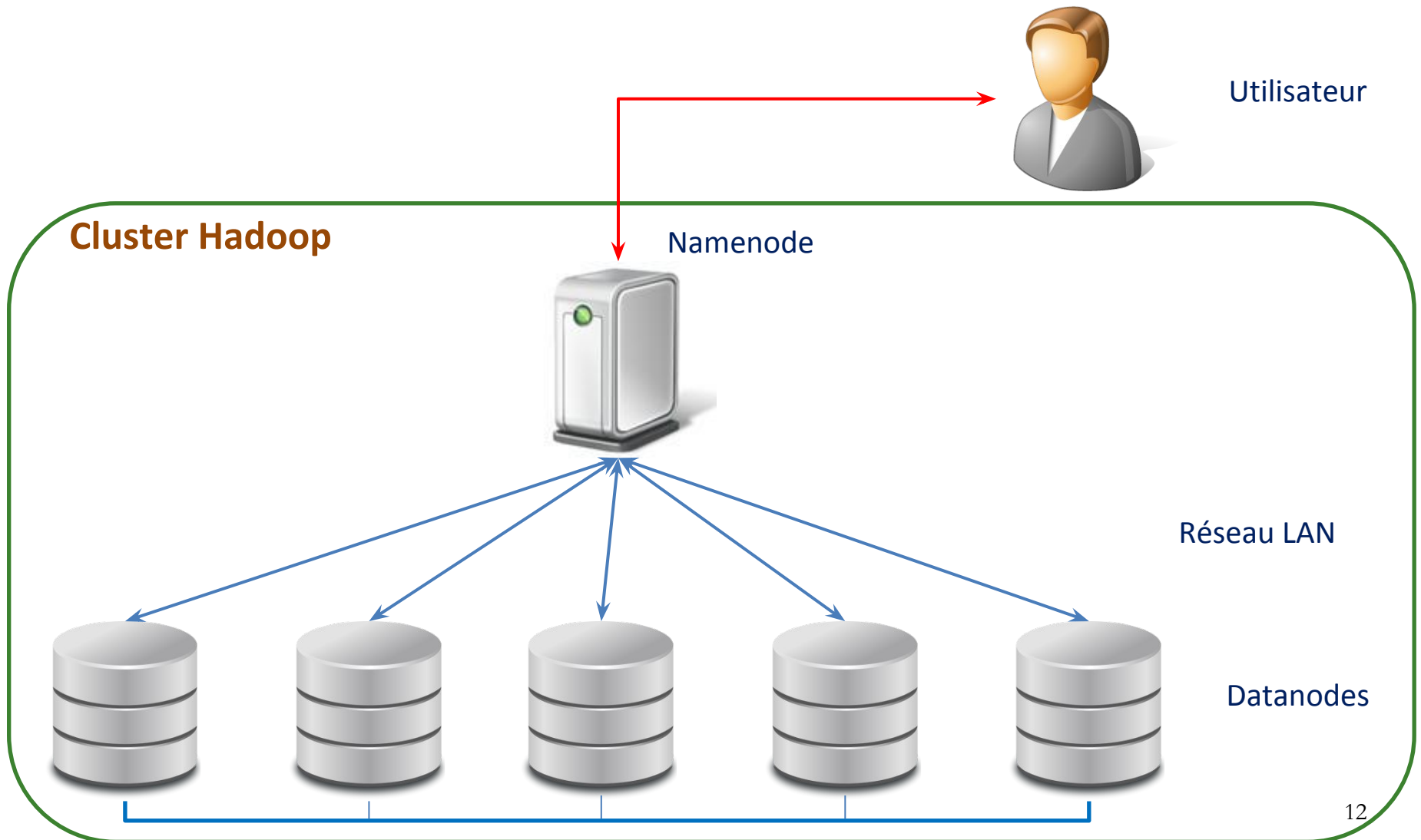
Ligne de commande

Architecture HDFS : présentation

- HDFS (*Hadoop Distributed Files System*) est le module de stockage des données sur Hadoop. C'est la couche la plus « basse » dans l'architecture Hadoop.
- Pour l'utilisateur, ce système est transparent et très proche en fonctionnement au noyau UNIX.
- Un ensemble de machines/serveurs mises en relation par HDFS s'appelle « **un cluster** ». On y retrouve deux types de machines / serveurs:
 - Les **namenodes** (nœuds maîtres)
 - Les **datanodes** (nœuds esclaves)
- Le namenode et les datanodes communiquent entre eux au travers d'un réseau LAN/WAN afin de pouvoir effectuer des opérations dans HDFS.



Architecture HDFS : le cluster (1)



Architecture HDFS : Les Datanodes

- Les datanodes sont les composants de bases de HDFS.
Un cluster contient généralement entre 3 et 2000 datanodes.
- Ils sont parfois appelés « nœuds esclaves ».
- Les datanodes stockent de la donnée.
- En cas de panne d'un d'un datanode, les données restent quand même accessibles au travers d'un autre datanode (*mécanisme de réplication de la donnée – slide 19*).
- Les datanodes peuvent communiquer entre eux ainsi qu'avec le namenode au travers du réseau LAN/WAN. Cependant, ils ne communiquent pas avec l'utilisateur.



Architecture HDFS : Le Namenode

- Le namenode est le composant central de HDFS. Il est la machine qui manage HDFS.
- Il est parfois appelé le « master node ».
- Il ne contient que des métadonnées (la localisation des fichiers sur le cluster ainsi que l'arborescence du système de fichiers).
- Il est essentiel au fonctionnement de HDFS. En cas de panne, les données deviennent alors inaccessibles, voire perdues à jamais. Pour éviter cela, on associe souvent au cluster un namenode secondaire.
- L'utilisateur communique exclusivement avec le namenode.



Architecture HDFS : Les metadata

- Les metadata contiennent la structure des fichiers et dossiers dans HDFS. C'est le système de checkpoint de HDFS.
- Les données metadata contiennent essentiellement deux types de fichiers:
- fichiers « **fsimage** »: ils indiquent l'état du fichier à un instant « t ».
- fichiers « **edits** »: fichiers logs qui contiennent les modifications faites depuis le fsimage le plus récent.
- Note: le Fichier « VERSION » contient des informations relatives à HDFS (numéro de version, identifiants cluster, type de stockage...).

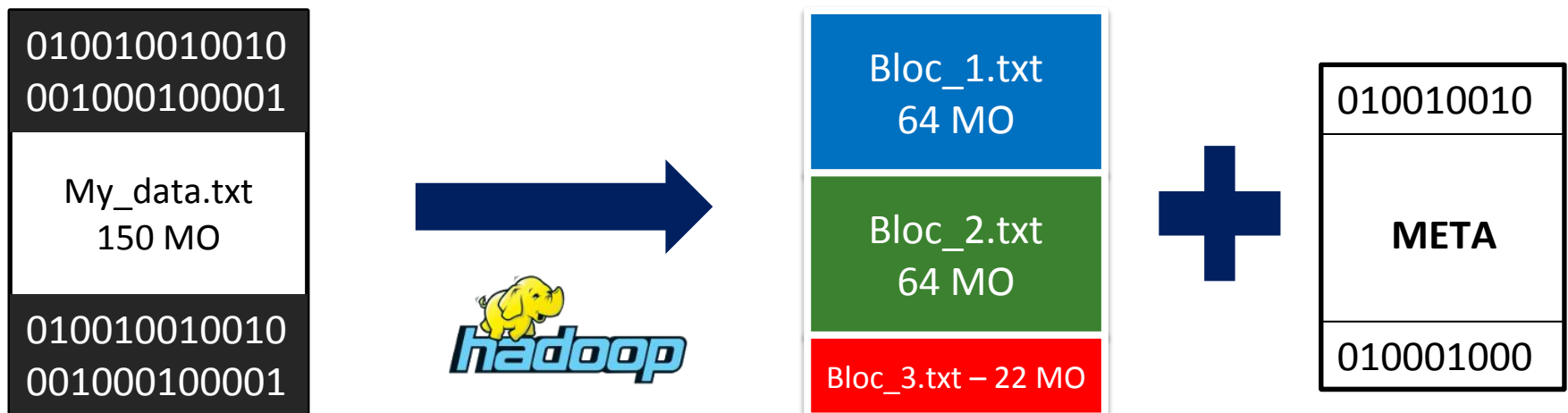
```
data/dfs/name
├── current
│   ├── VERSION
│   ├── edits_00000000000000000001-00000000000000000007
│   ├── edits_00000000000000000008-00000000000000000015
│   ├── edits_00000000000000000016-00000000000000000022
│   ├── edits_00000000000000000023-00000000000000000029
│   ├── edits_00000000000000000030-00000000000000000030
│   ├── edits_00000000000000000031-00000000000000000031
│   ├── edits_inprogress_0000000000000000000032
│   ├── fsimage_0000000000000000000030
│   ├── fsimage_0000000000000000000030.md5
│   ├── fsimage_0000000000000000000031
│   ├── fsimage_0000000000000000000031.md5
│   ├── seen_txid
└── in_use.lock
```

Architecture HDFS : Protection

- Hadoop HDFS prévoit deux mécanismes de protection pour protéger le namenode en cas de panne / casse.
- « **la réplication des métadonnées** »: première solution historiquement mise en œuvre dans HDFS, elle consiste à synchroniser périodiquement les métadonnées avec un système de stockage extérieur au cluster Hadoop. En cas de panne/casse du namenode, on possède ainsi une copie récente de nos métadonnées.
- « **L'utilisation d'un Secondary Namenode** ». Solution plus récente et généralement utilisée aujourd'hui. Elle consiste à utiliser un namenode de standby, qui prendra le relais du namenode principal en cas de panne/casse. De même les metadata sont synchronisées périodiquement pour éviter toute perte de données.

Stockage des données

- Dans HDFS, les données sont par défaut découpées en blocs de 64 MO avant d'être stockées sur le cluster.
- La taille des blocs influe sur les performances de requêtage de données. Plus la taille de bloc est élevée, plus le chargement total des données sera rapide.
- On associe à chaque fichier des métadonnées.
- Exemple, supposons un fichier my_data.txt d'une taille de 150 MO.



Stockage des données

- Les données sont ensuite envoyées sur les différents datanodes. Celles-ci sont également répliquées un certain nombre de fois égal à la valeur du « facteur de réplication » (ici égal à 3).



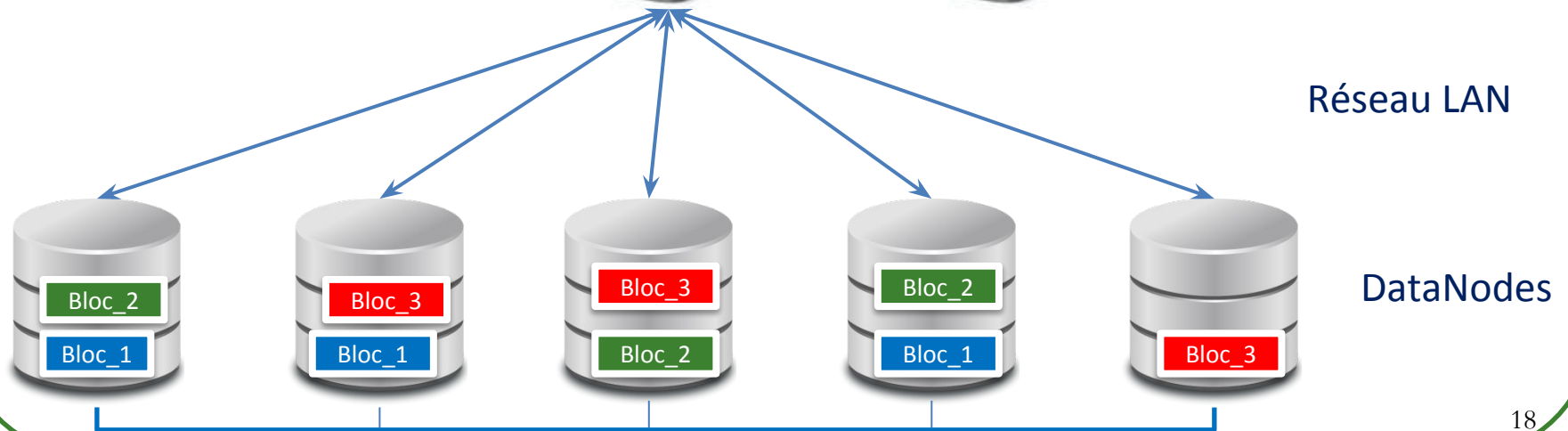
Utilisateur

Cluster Hadoop

NameNode

Secondary
NameNode

010010010
META
010001000



Ligne de commande

- Pour interagir avec Hadoop, on utilise une ligne de commande (terminal) depuis l'interface utilisateur.
- Pour se référer à l'interface utilisateur, la documentation officielle se réfère au terme « **en local** ».
- La ligne de commande de HDFS est très proche des fonctionnalités proposées sur une ligne de commande type UNIX. Elle est également la façon la plus simple d'interagir avec HDFS.

```
[root@sandbox ~]# hadoop fs -mkdir /user/hadoop  
[root@sandbox ~]# hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2 /user/hadoop/dir3
```


Ligne de commande

Fonctionnalité	Commande (effectuée sur un terminal en local)
Création d'un nouveau répertoire dans l'arborescence HDFS.	# <code>hadoop fs -mkdir <paths></code>
Afficher les fichiers présents dans un répertoire HDFS.	# <code>hadoop fs -ls <args></code>
Uploader un fichier du système local au cluster HDFS	# <code>hadoop fs -put <local-src> <HDFS_dest_path></code>
Afficher la taille des fichiers et répertoires dans un répertoire HDFS	# <code>hadoop fs -du <repertoire></code>
Obtenir l'aide	# <code>hadoop fs -help</code>

```
[root@sandbox ~]# hadoop fs -ls /user/hadoop
Found 3 items
drwxr-xr-x  - root hdfs      0 2016-03-01 07:40 /user/hadoop/dir1
drwxr-xr-x  - root hdfs      0 2016-03-01 06:11 /user/hadoop/dir2
drwxr-xr-x  - root hdfs      0 2016-03-01 06:11 /user/hadoop/dir3
[root@sandbox ~]# hadoop fs -ls /user/hadoop/dir1
Found 1 items
-rw-r--r--   3 root hdfs    157 2016-03-01 07:40 /user/hadoop/dir1/popularNames.txt
[root@sandbox ~]# hadoop fs -ls /user/hadoop/dir1/popularNames.txt
-rw-r--r--   3 root hdfs    157 2016-03-01 07:40 /user/hadoop/dir1/popularNames.txt
[root@sandbox ~]#
```

Ligne de commande

- La liste de commande complète est disponible sur la documentation officielle HDFS.
- **Note:** il est également possible d'interagir avec HDFS avec de la programmation Java.
- **Note:** Il existe également des interfaces graphiques permettant d'interagir avec HDFS (ex: Ambari). Celles-ci s'appuient bien sûr sur la ligne de commande HDFS.



The screenshot shows a web browser window with the URL <https://hadoop.apache.org/docs/r2.4.1/>. The page features the Hadoop logo (a yellow elephant) and the word "hadoop" in a stylized blue font. Below the logo, there is a breadcrumb trail: "Apache > Hadoop > Apache Hadoop Project Dist POM > Apa". The main content area is divided into two columns. The left column contains a list of links under three categories: "General", "Common", and "HDFS". The right column contains a list of command names. The "General" category includes links to Overview, Single Node Setup, Cluster Setup, Hadoop Commands, Reference, File System Shell, and Hadoop Compatibility. The "Common" category includes links to CLI Mini Cluster, Native Libraries, Superusers, Secure Mode, Service Level, Authorization, and HTTP Authentication. The "HDFS" category includes links to HDFS User Guide, High Availability With QJM, High Availability With NFS, Federation, ViewFs Guide, HDFS Snapshots, HDFS Architecture, Edits Viewer, Image Viewer, Permissions and HDFS, Quotas and HDFS, and HFTP. The right column lists the following commands: Overview, appendToFile, cat, chgrp, chmod, chown, copyFromLocal, copyToLocal, count, cp, du, dus, expunge, get, getfacl, getmerge, ls, lsr, mkdir, moveFromLocal, moveToLocal, mv, put, rm, rmr, setfacl, setrep, stat, tail, test, text, and touchz.

→ **General**

- Overview
- Single Node Setup
- Cluster Setup
- Hadoop Commands
- Reference
- File System Shell
- Hadoop Compatibility

→ **Common**

- CLI Mini Cluster
- Native Libraries
- Superusers
- Secure Mode
- Service Level
- Authorization
- HTTP Authentication

→ **HDFS**

- HDFS User Guide
- High Availability With QJM
- High Availability With NFS
- Federation
- ViewFs Guide
- HDFS Snapshots
- HDFS Architecture
- Edits Viewer
- Image Viewer
- Permissions and HDFS
- Quotas and HDFS
- HFTP

- Overview
- appendToFile
- cat
- chgrp
- chmod
- chown
- copyFromLocal
- copyToLocal
- count
- cp
- du
- dus
- expunge
- get
- getfacl
- getmerge
- ls
- lsr
- mkdir
- moveFromLocal
- moveToLocal
- mv
- put
- rm
- rmr
- setfacl
- setrep
- stat
- tail
- test
- text
- touchz

Hadoop MapReduce

Algorithmes MapReduce

Hadoop MapReduce

Hadoop MapReduce en pratique

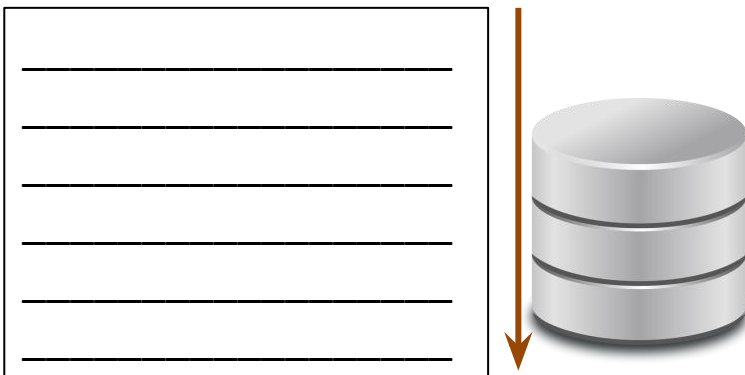
Hadoop version 2.0

Algorithmes MapReduce

- Dans le cas de grands volumes de données, le traitement du dataset ne peut être effectué par une seule machine. Les calculs seraient bien trop lents et en inadéquation avec de nombreux points de la règle des 5Vs.
- Il est donc nécessaire que les calculs soient effectués par plusieurs machines simultanément afin de réduire les temps de latence.
- Cependant, le traitement distribué des données devient rapidement très complexe. Fort heureusement, des solutions existent.
- C'est ainsi qu'en 2005, plusieurs whitepapers sont sortis sous l'impulsion de Google et ont alors ouvert la voie à un nouveau paradigme de calcul : les algorithmes dit "MapReduce".

Algorithmes MapReduce

- Un algorithme de type MapReduce consiste à séparer les données et à les traiter simultanément afin de gagner en performance.
- Cela est rendu possible en informatique grâce à l'utilisation de plusieurs machines pouvant fonctionner en parallèle.
- La parallélisation est rendue transparente pour l'utilisateur afin qu'il puisse se concentrer sur le traitement des données.



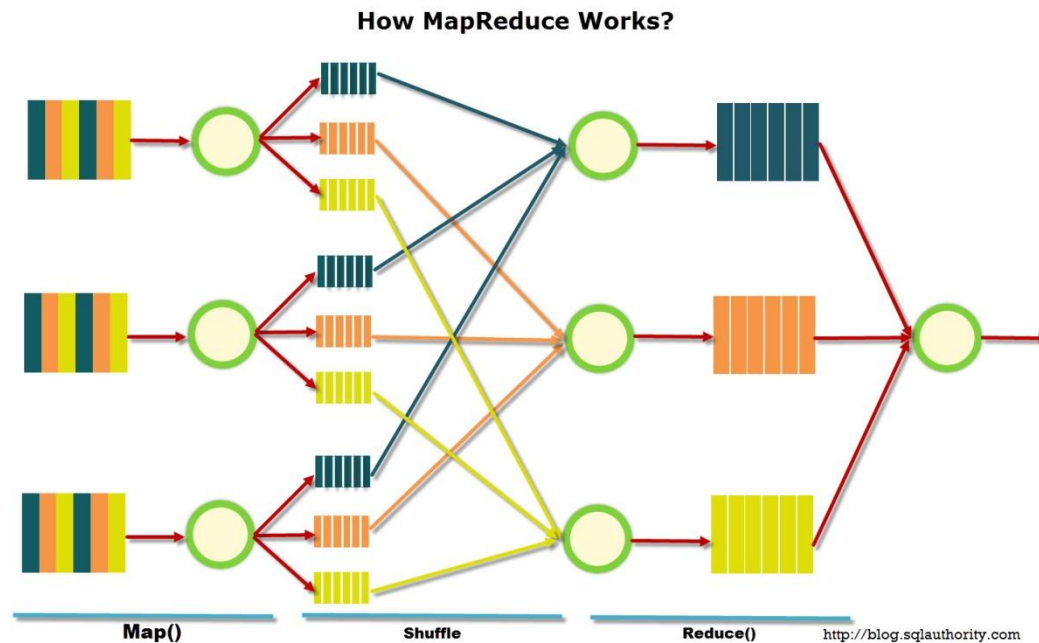
Algorithme classique



Algorithme type MapReduce

Algorithmes MapReduce : Présentation

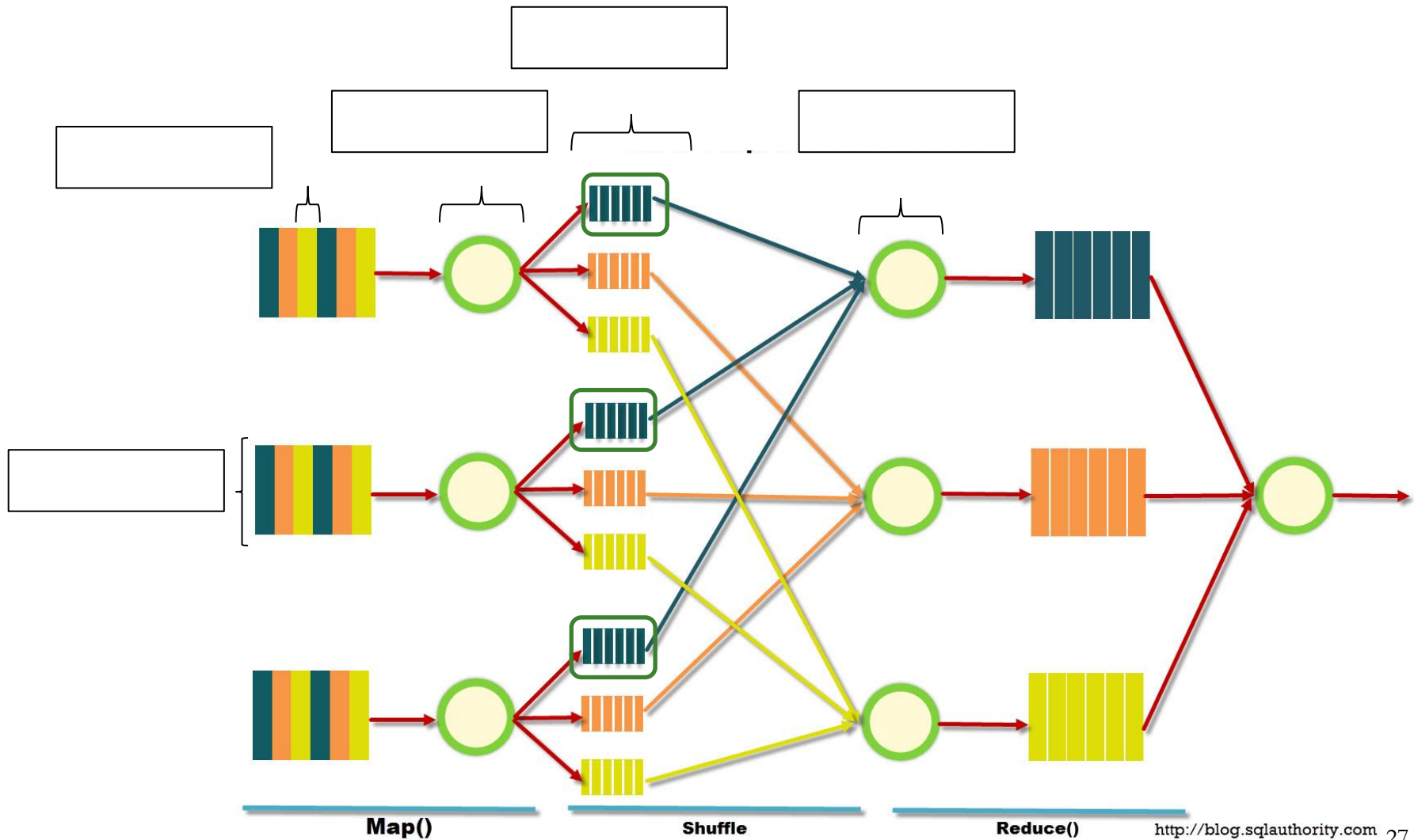
- L'algorithme MapReduce est composé de deux parties consécutives et opérées par deux entités distinctes:
 - Le « mapping » qui est effectué par des objets Java de type Mappers sur Hadoop MapReduce.
 - Le « reducing » qui est effectué par des objets Java de type Reducers sur Hadoop MapReduce.
- Bien écrites, ces deux fonctions « mapping » et « reducing » permettent de produire des traitements complexes (traitement de données, fouille de données, algorithmes de machine learning...)



Algorithmes MapReduce : vocabulaire

Terme	Définition
Job	Un job est le processus global qu'on exécute. Il comprend les données d'entrées, le mapping, le reducing et la sortie.
Task	Chaque job est découpé en plusieurs tâches sur les mappers et les reducers. Une fois que toutes les tâches sont finies, le job est fini.
Split / bloc	Les données d'entrée sont divisées en plusieurs splits (généralement, un split est équivalent à un bloc HDFS et a une taille de 64 MO).
Record / observation	Chaque bloc contient un certain nombre d'observations / lignes. Par défaut, le mapper lit et traite une ligne à la fois.
Partition	C'est l'entrée passée à un reducer.

Quiz



Algorithmes MapReduce : Application

- Afin de pouvoir mieux comprendre le principe du MapReduce, nous allons illustrer la théorie par un exemple concret.
- On suppose une chaîne de supermarchés possédant de nombreux magasins avec les données suivantes:

2016-08-28	Paris	Alimentation	12,00€
2016-08-29	Marseille	Vêtements	34,00€
2016-08-29	Lyon	Médiathèque	24,00€



2016-08-28	Marseille	Alimentation	8,00€
2016-08-28	Marseille	Médiathèque	26,00€
2016-08-29	Lyon	Alimentation	18,00€



2016-08-28	Paris	Médiathèque	3,00€
2016-08-29	Paris	Vêtements	56,00€
2016-08-28	Lyon	Alimentation	16,00€

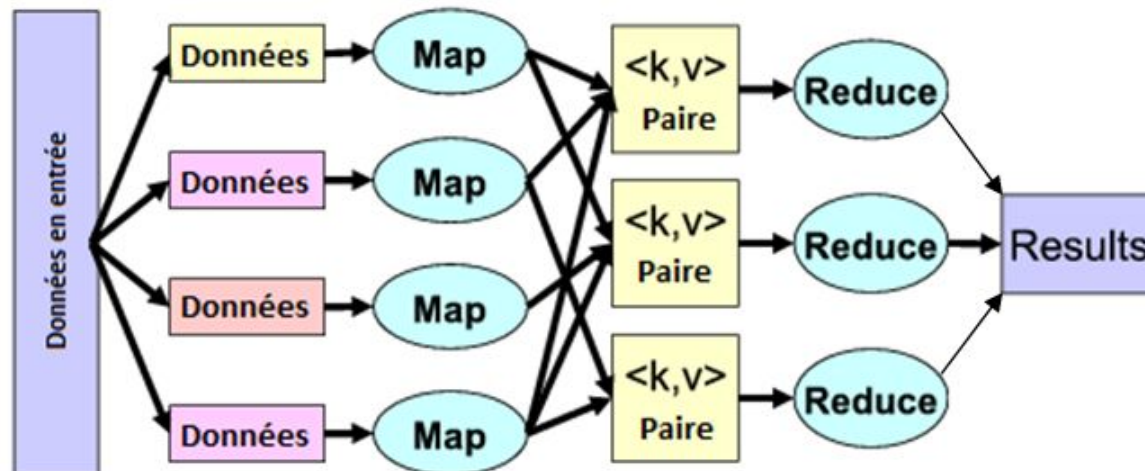


Question:

Quels ont été les profits générés pour chaque ville?

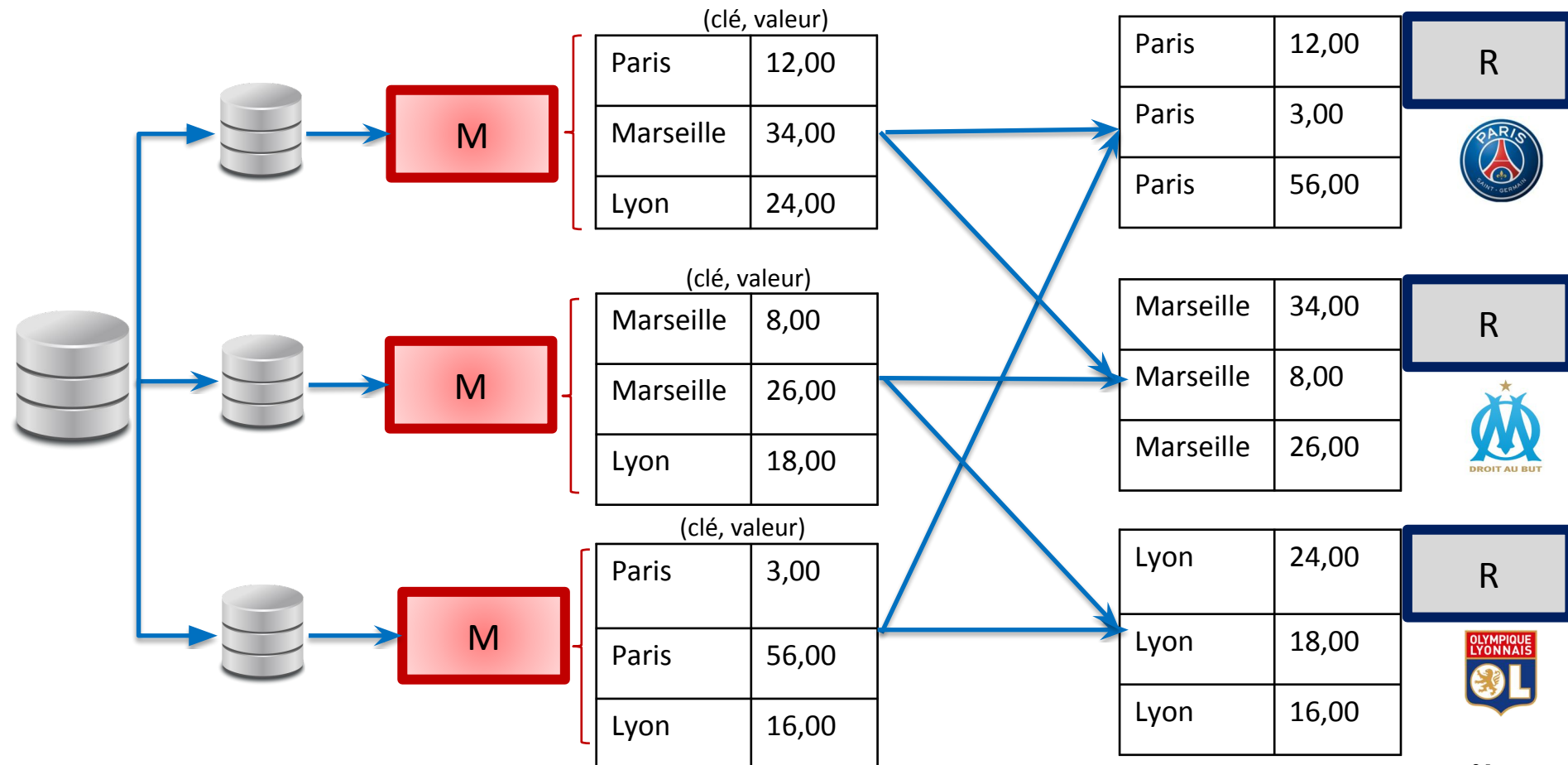
Algorithmes MapReduce : Mapping

- La phase « **mapping** » est un petit programme effectué par plusieurs machines simultanément sur des petits blocs de données. Cette phase est effectuée par les mappers.
 - Une opération dite de « mapping » consiste à transformer une collections d'objets « A » en objets « B ».
- « **Tokenization** »: Premièrement, le mapping transforme chaque observation du dataset sous forme d'un tuple au format « (clé, valeur) ».
 - « **shuffling-sort** »: les tuples (clés, valeur) sont regroupés au sein des reducers selon la valeur de leur clés.



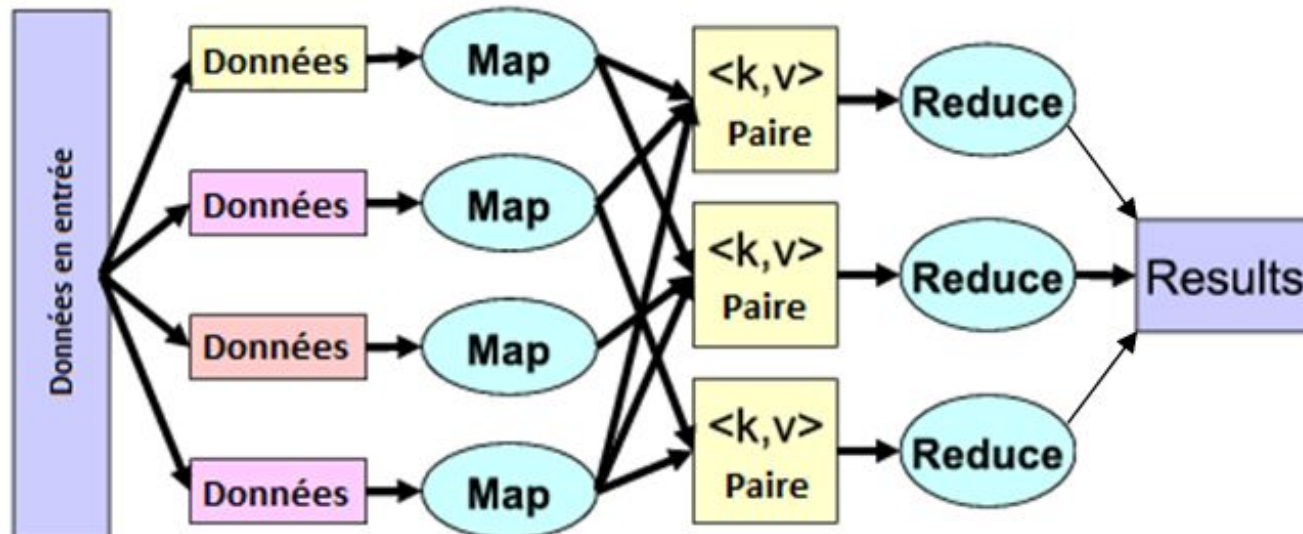
Algorithmes MapReduce : Mapping

- Pour répondre à la question posée, on décide de faire un mapping (*ville* , *prix*) :

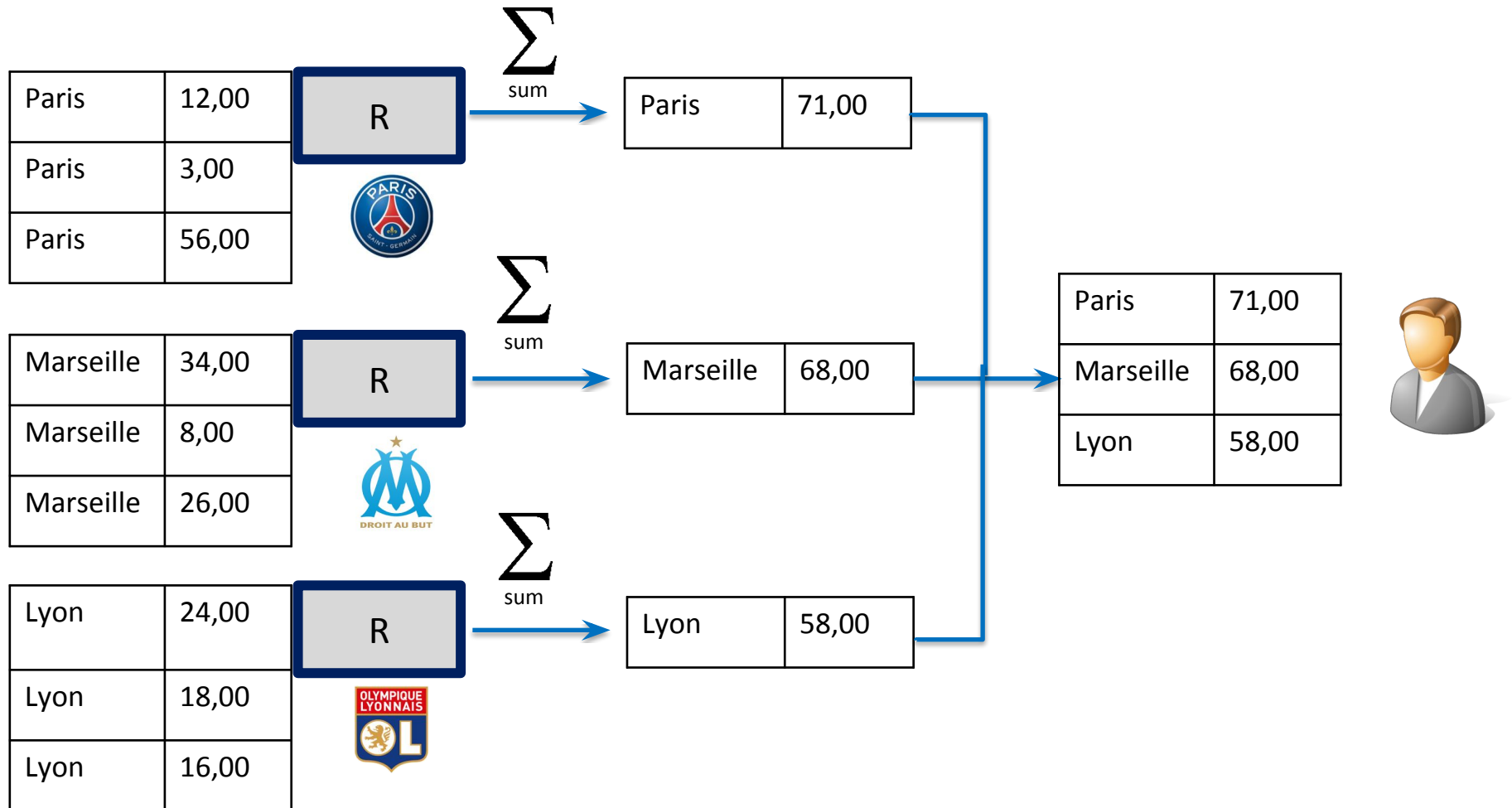


Algorithmes MapReduce : Reducing

- La phase « **Reducing** » est un programme exécuté sur les données transmises par les mappers (*les partitions*). Cette phase est effectuée par les reducers.
- Cette phase consiste à déduire le résultat attendu par le programme à partir des données (*clés, valeurs*) issues des mappers.
- Il y a autant de reducers qu'il y a de clés différentes.
- Enfin, les données finales sont assemblées et transférées à l'utilisateur. C'est « l'output ».

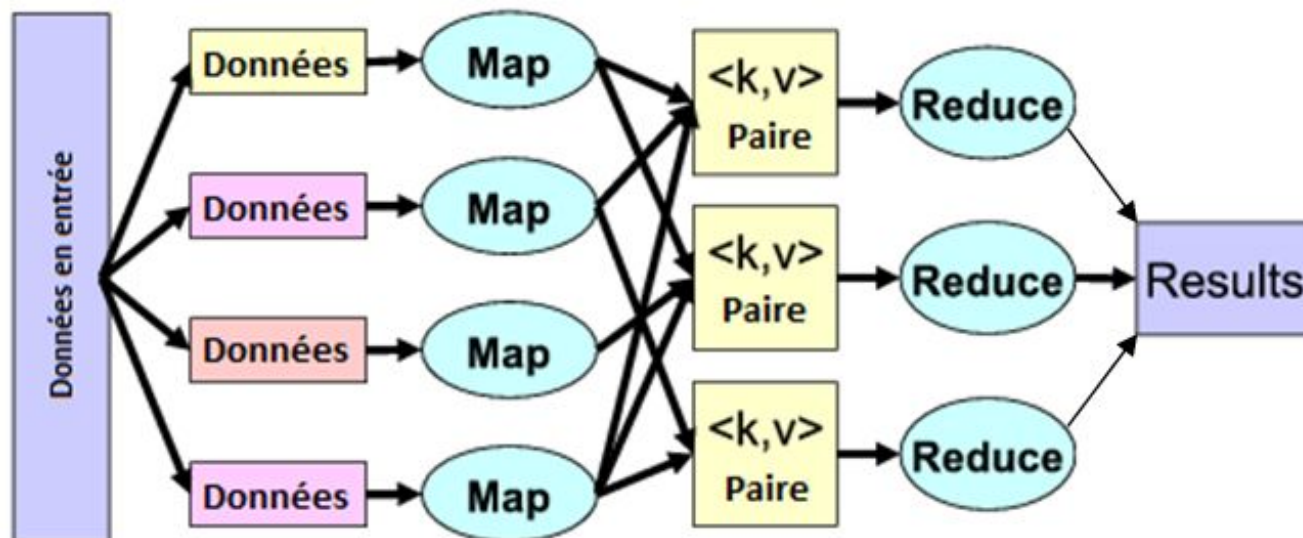


Algorithmes MapReduce : Reducing



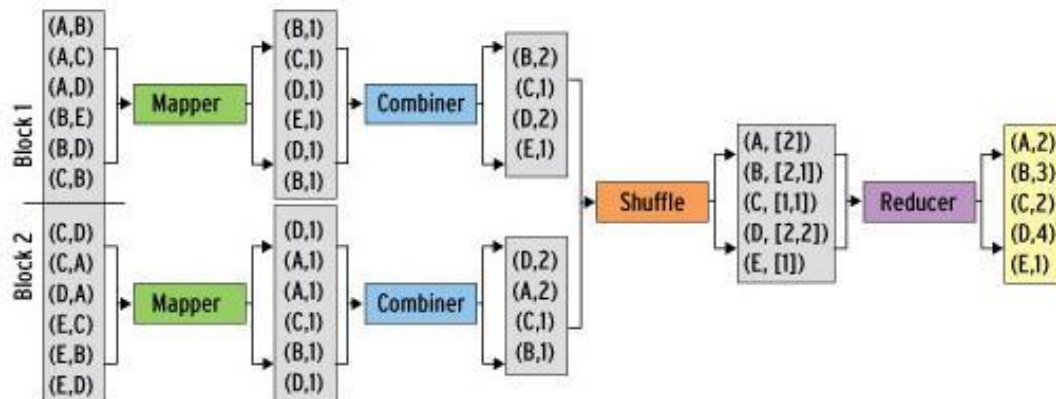
Algorithmes MapReduce : Optimisation

- L'algorithme présenté précédemment est très peu optimisé. En effet, après chaque phase de mapping, les données sont “shufflées” et envoyées à chaque “reducer”.
- Cependant, ces coûts de transfert sont très lourds et nécessitent évidemment des optimisations afin de rester viable en présence de grands volumes de données.

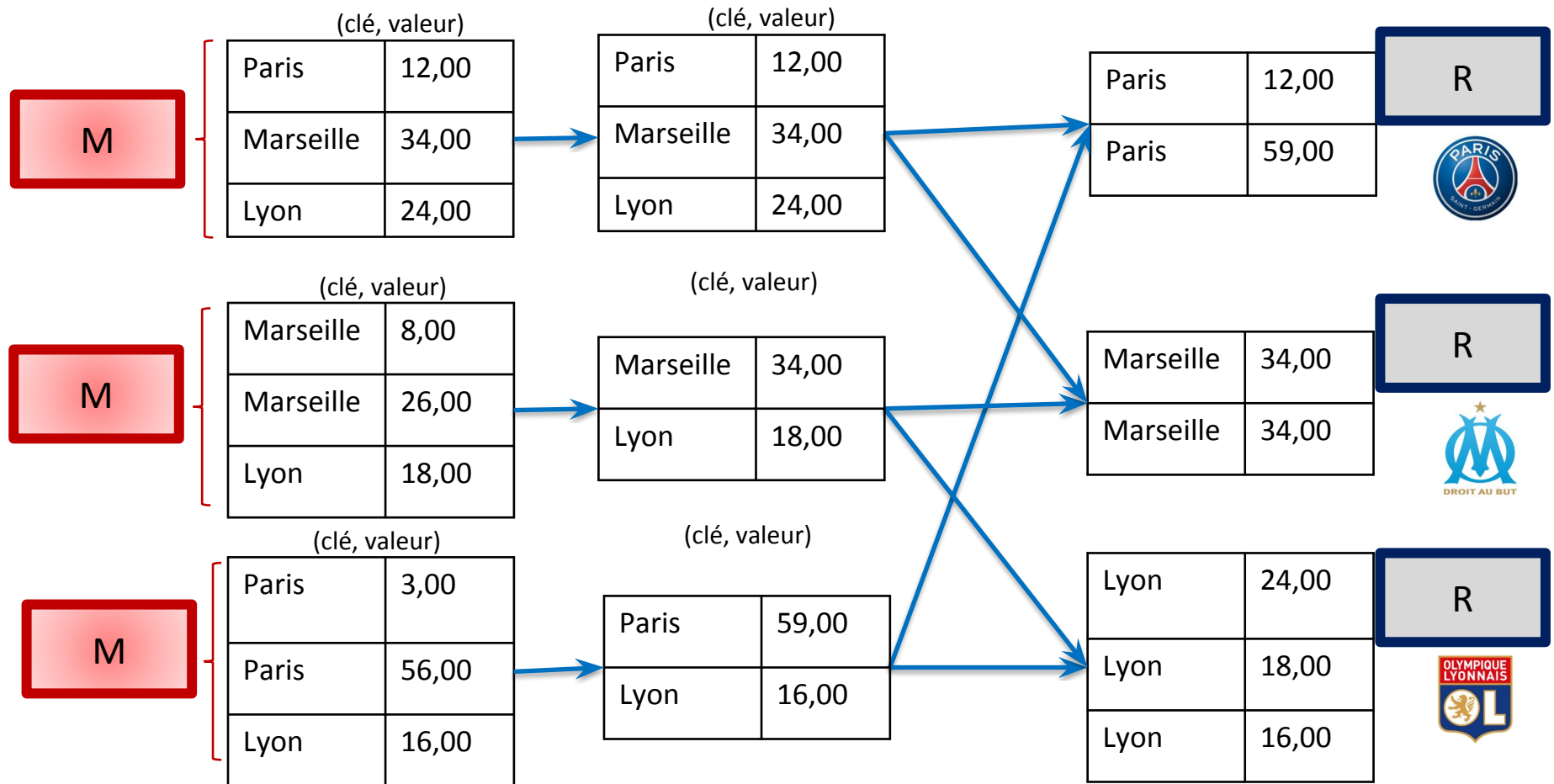


Algorithmes MapReduce : Optimisation

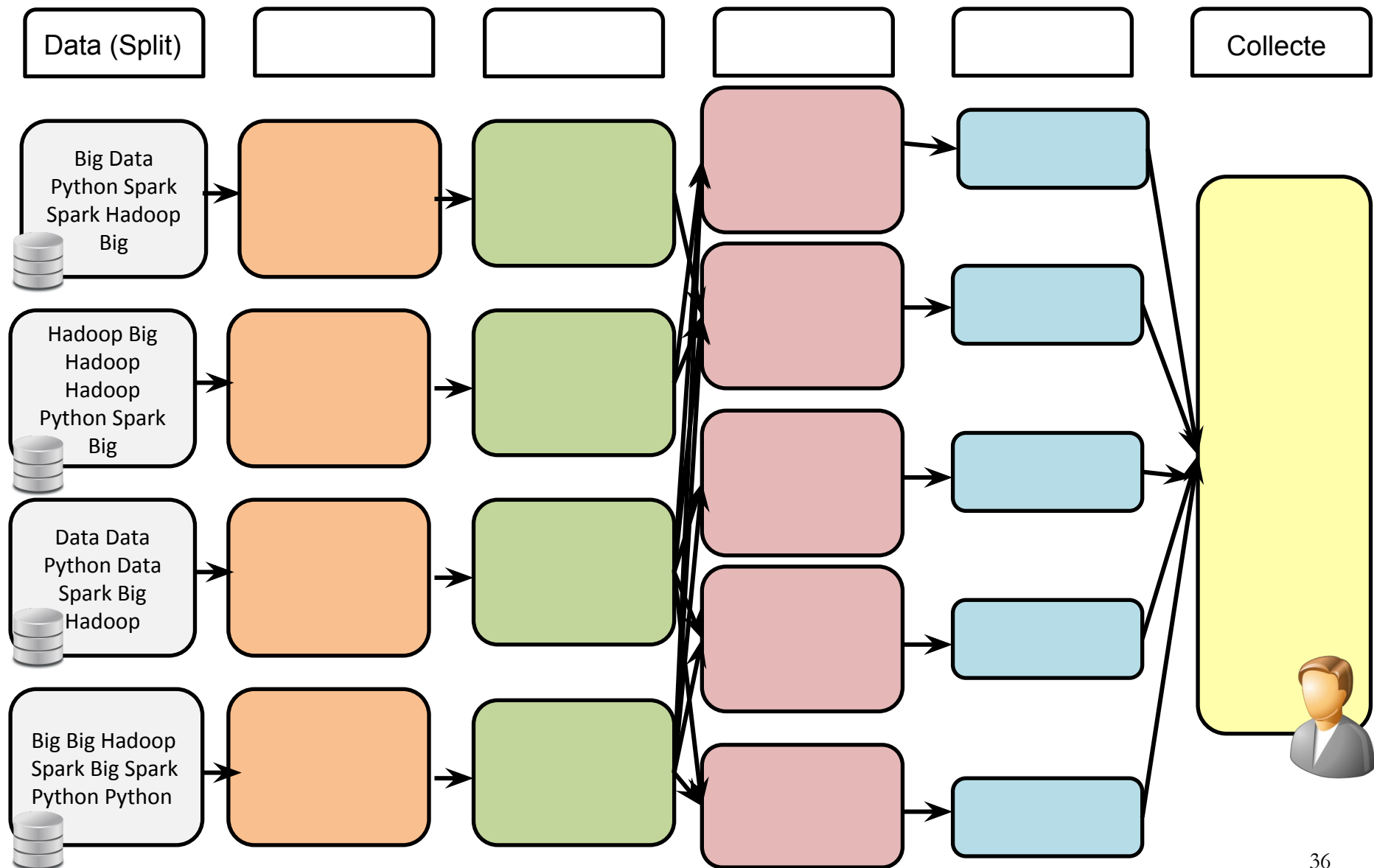
- Un combiner, aussi appelé “semi-reducer” est généralement ajouté à l’algorithme MapReduce afin d’améliorer les performances.
- Un *combiner* est utilisé pour condenser les données à la sortie d’un mapper avant de les envoyer aux reducers.
- **Note** : Un *combiner* peut parfois être de même nature que le reducer.
- **Note** : Ainsi, on limite le nombre d'observations lors du shuffle.



Algorithmes MapReduce : Optimisation



Quiz : Word Count avec Combining



Hadoop MapReduce

- Le projet Hadoop MapReduce est la brique de traitement distribué dans le framework Hadoop.
- Elle s'appuie sur les algorithmes dit « MapReduce » vus précédemment.
- Hadoop MapReduce permet d'effectuer des calculs parallélisables sur un énorme volume de données (dataset stocké sur HDFS ou dans une base de données) en utilisant un grand nombre de machines.



Hadoop MapReduce : Composants

- Placée au-dessus d'un système de stockage distribué (ex: HDFS ou autre), la brique Hadoop MapReduce connaît un niveau d'abstraction supplémentaire.
- A ce niveau d'abstraction, un traitement MapReduce est réalisé par quatre entités distinctes.

L'utilisateur



Les TaskTrackers



Un système de
stockage distribué



Le JobTracker



Hadoop MapReduce : JobTracker

- Le jobTracker et les Tasktrackers sont des entités essentielles pour pouvoir effectuer un job MapReduce. Ces deux entités sont des JVM (*Java Virtual Machines*) auxquelles on a attribué des ressources (mémoires, heap, cpu...)
- Le **jobTracker** tourne généralement sur le NameNode.
- Il reçoit la demande de job de la part du client et est capable de communiquer avec le namenode afin de connaître la localisation des données.
- Il est responsable de la création du job et de son management tout au long de son exécution. Il découpe le job en tâches et le distribue aux différents Tasktrackers.
- En cas de panne du jobTracker, HDFS reste fonctionnel mais les processus Hadoop MapReduce sont stoppés.

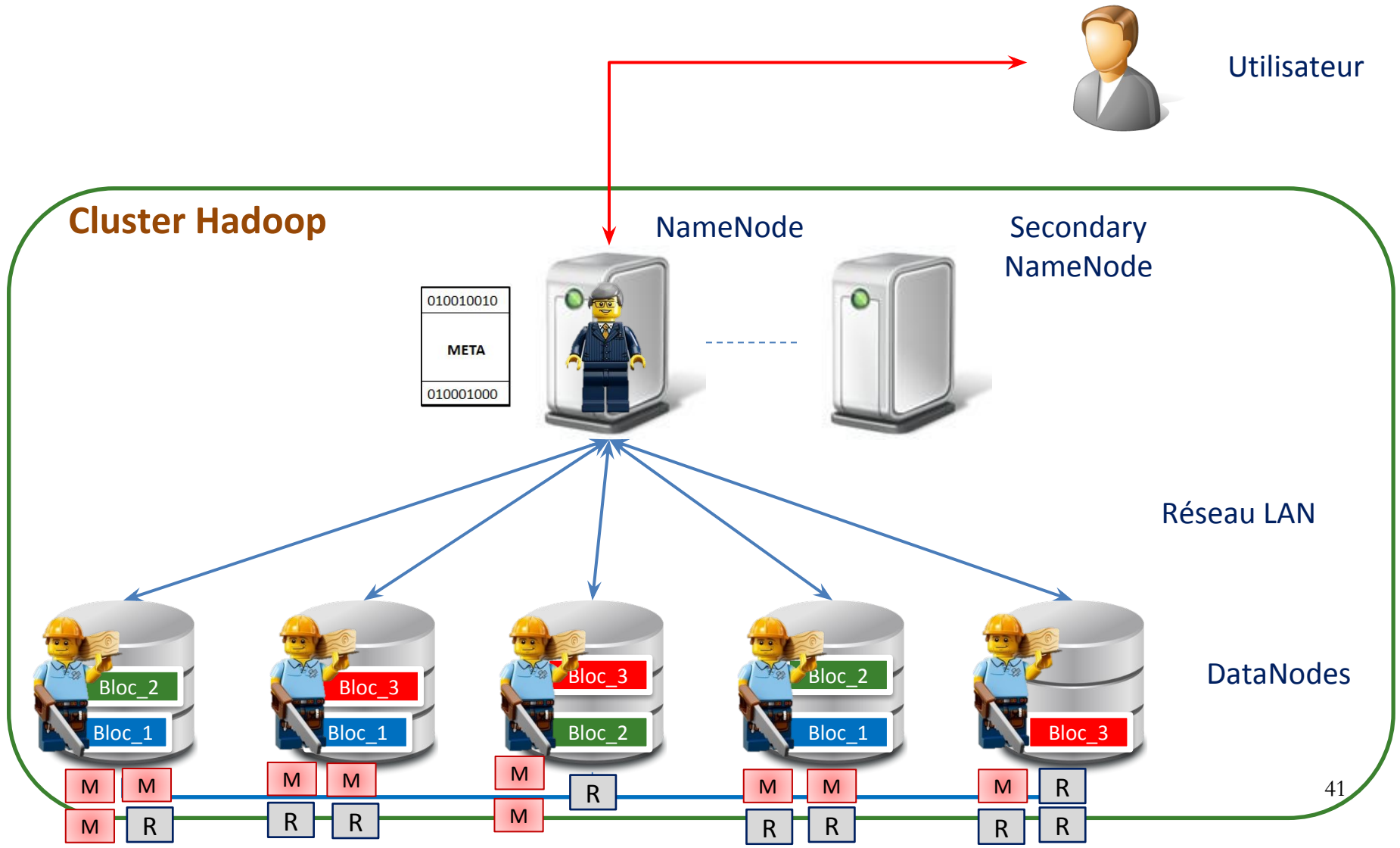


Hadoop MapReduce : TaskTrackers

- Les TaskTrackers sont également des JVM (*Java Virtual Machines*) auxquelles on a attribué des ressources (mémoires, heap, cpu...).
- Les **taskTrackers** tournent sur les datanodes. Ils sont chargés d'effectuer les calculs et de reporter leurs statuts au Jobtracker.
- Un taskTracker est polyvalent. Tant que les ressources le permettent, il joue simultanément le rôle de plusieurs mappers et reducers.
- Les taskTrackers sont en communication constante avec le Jobtracker auquel ils précisent leur avancement.
- En cas de panne d'un taskTracker, le Jobtracker s'occupe de façon transparente de transférer les tâches de ce taskTracker vers un autre.



Hadoop MapReduce : architecture



En pratique

- Les programmes Hadoop mapReduce étant complexes et parfois longs dans leur exécution, il est conseillé d'adopter la démarche de développement suivante:
 1. Au départ, test du script en local sur un seul nœud - petites données.
 2. Ensuite, test du script sur un cluster Hadoop – petites données.
 3. Enfin, test du script sur un cluster Hadoop – grandes données.
- Les programmes Hadoop MapReduce sont usuellement écrits en Java. Il est cependant possible d'utiliser d'autres APIs pour développer, notamment Python, Perl, Ruby. Cela est rendu possible grâce à l'outil Hadoop streaming introduit dans la version 0.14 de Hadoop.
- Pour les besoins de la classe, nous allons nous intéresser à développer et exécuter un programme MapReduce avec Python sur un exemple.

En pratique

- Il est possible de développer un « semblant d'algorithme MapReduce » en local et en python en utilisant un pipeline unix bash. Cela permet de développer et tester facilement son algorithme en local.

```
cat data.txt | python mapper.py | sort | python reducer.py
```

- Pour cela, nous allons avoir besoin de construire quelques outils :
 - un jeu de données (txt, csv...)
 - Un programme python de mapping.
 - Un programme python de reducing.
- **Attention « Input & output »:** Pour pouvoir mettre en place ce python Trick, il est nécessaire de faire les liaisons entre les pipelines dans notre programme python en ajoutant quelques lignes à nos script:
 - Les données seront chargées depuis STDIN (standard input)
 - Les données seront exportées depuis STDOUT (standard output)

En pratique : Avec python en local

cat data.txt

```
2016-01-01,12:01,Paris,Musique,14.00,CB
2016-01-02,14:08,Lyon,Alimentation,15.25,cash
2016-01-02,15:03,Marseille,Musique,26.02,CB
2016-01-03,15:18,Paris,Alimentation,8.36,CB
```

Total des ventes par magasin?

cat data.txt | python mapper.py

```
def mapper():
    """
    mapping function
    """
    for line in sys.stdin:
        row = line.strip().split(",")
        date, time, store, item, cost, payment = row
        print("%s,%s" % (store, cost))

mapper()
```

```
Paris,14.00
Lyon,15.25
Marseille,26.02
Paris,8.36
```

En pratique : Avec python en local

cat data.txt | python mapper.py | sort

```
Lyon,15.25  
Marseille,26.02  
Paris,14.00  
Paris,8.36
```

cat data.txt | mapper.py | sort | reducer.py

```
def reducer():  
    """  
    Reducing Function  
    """  
    sales_total = 0  
    old_key = None  
    for line in sys.stdin:  
        row = line.strip().split(",")  
        current_key, sale = row  
  
        if old_key is None:  
            old_key = current_key  
  
        if old_key != current_key:  
            print("%s,%s" % (old_key, sales_total))  
            sales_total = 0  
            old_key = current_key  
  
        sales_total += float(sale)  
  
    print("%s,%s" % (old_key, sales_total))  
  
reducer()
```

```
Lyon,15.25  
Marseille,26.02  
Paris,22.36
```

En pratique : Exécution sur un cluster

- Depuis la version 0.14, Hadoop Streaming a été intégré à la distribution de base d'Hadoop. Cela permet de pouvoir construire des programmes Hadoop MapReduce avec différents langages de programmation.
- Les programmes MapReduce peuvent alors s'exécuter avec la commande Hadoop streaming:

```
$HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-streaming-X.Y.Z.jar \  
-input input_dirs \  
-output output_dir \  
-mapper $path/mapper.py \  
-reducer $path/reducer.py
```

- Cette commande prend les arguments suivants:
 - « Input » : les données d'entrées sous forme d'un chemin HDFS.
 - « output » : Répertoire HDFS où seront stockées les données résultats
 - « mapper » : le chemin local du programme python chargé du mapping
 - « reducer » : le chemin local du programme python chargé du reducing

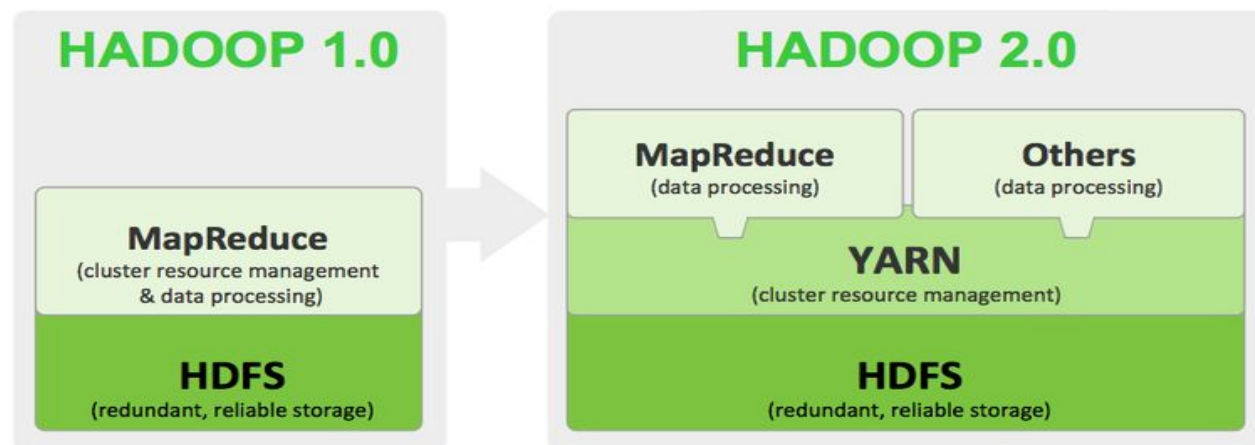
En pratique : Exécution sur un cluster

- **Remarque:** il est possible de passer d'autres arguments à la commande Hadoop streaming. (verbose, combiner, etc...)
- Il est également possible de spécifier le nombre de mappers voulu afin d'augmenter les performances de notre MapReduce. Pour cela on utilise l'argument « **-D mapred.map.tasks=X** ».
- Si cet argument n'est pas spécifié, Hadoop MapReduce créera un nombre de mappers par défaut tel que défini dans la configuration Hadoop MapReduce.

Parameters	Description
-input directory/file-name	Input location for mapper. (Required)
-output directory-name	Output location for reducer. (Required)
-mapper executable or script or JavaClassName	Mapper executable. (Required)
-reducer executable or script or JavaClassName	Reducer executable. (Required)
-file file-name	Makes the mapper, reducer, or combiner executable available locally on the compute nodes.
-inputformat JavaClassName	Class you supply should return key/value pairs of Text class. If not specified, TextInputFormat is used as the default.
-outputformat JavaClassName	Class you supply should take key/value pairs of Text class. If not specified, TextOutputFormat is used as the default.
-partitioner JavaClassName	Class that determines which reduce a key is sent to.
-combiner streamingCommand or JavaClassName	Combiner executable for map output.
-cmdenv name=value	Passes the environment variable to streaming commands.
-inputreader	For backwards-compatibility: specifies a record reader class (instead of an input format class).
-verbose	Verbose output.
-lazyOutput	Creates output lazily. For example, if the output format is based on FileOutputFormat, the output file is created only on the first call to output.collect (or Context.write).
-numReduceTasks	Specifies the number of reducers.
-mapdebug	Script to call when map task fails.
-reducededbug	Script to call when reduce task fails.

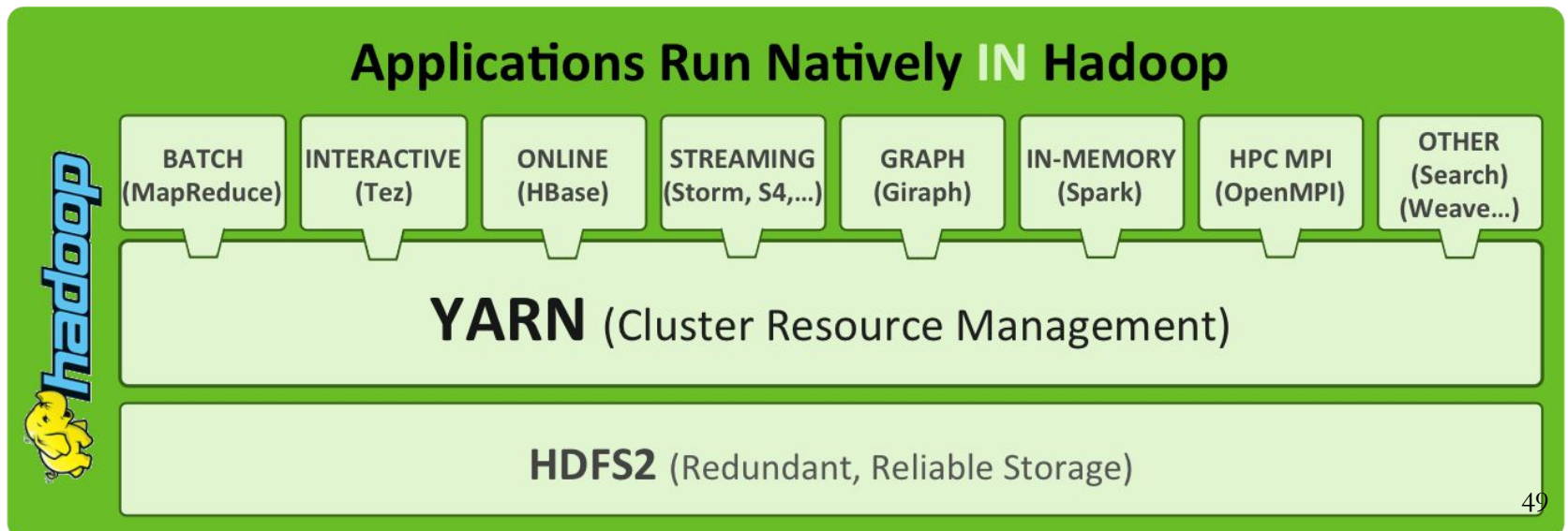
Hadoop version 2.0 : YARN

- La brique Hadoop MapReduce a connu quelque changements entre Hadoop 1.0 et Hadoop 2.0.
- Dans Hadoop 1.0, l'allocation de ressources se faisait par MapReduce. Dans Hadoop 2.0, l'allocation des ressources se fait par une nouvelle brique, « YARN » (*Yet Another Resource Negotiator*).
- Attention : Hadoop MapReduce 1 met en jeu les Tasktrackers et jobTracker comme vus précédemment. Hadoop MapReduce 2 les remplace par de nouvelles entités appelées Ressources managers et Node managers.



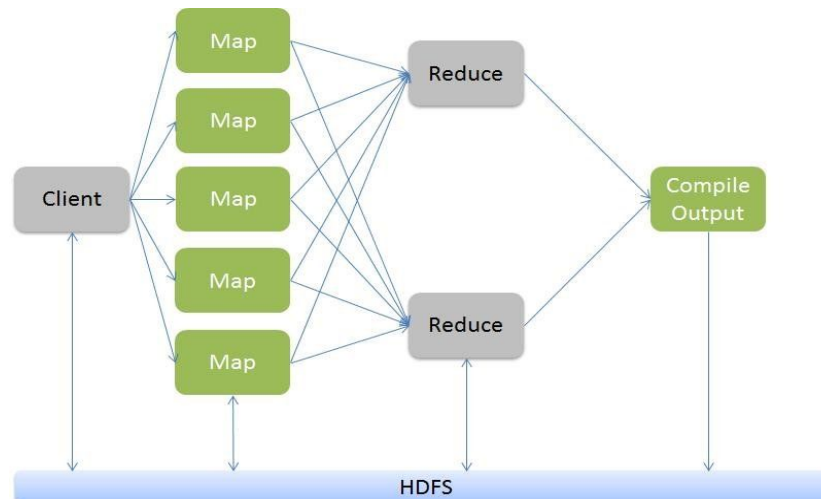
Hadoop version 2.0 : YARN

- YARN (« Yet Another Resource Negotiator ») est un module Hadoop dédié à l'allocation des ressources pour les briques au-dessus de HDFS. YARN est une brique nouvelle qui est venue se greffer à partir de la version Hadoop 2.0.
- L'allocation des ressources par YARN ne s'applique pas seulement à MapReduce et permet ainsi à de nombreuses autres briques de tourner à la place de MapReduce.
- Le fonctionnement de YARN est plus complexe et ne sera pas vu lors de cette unité!



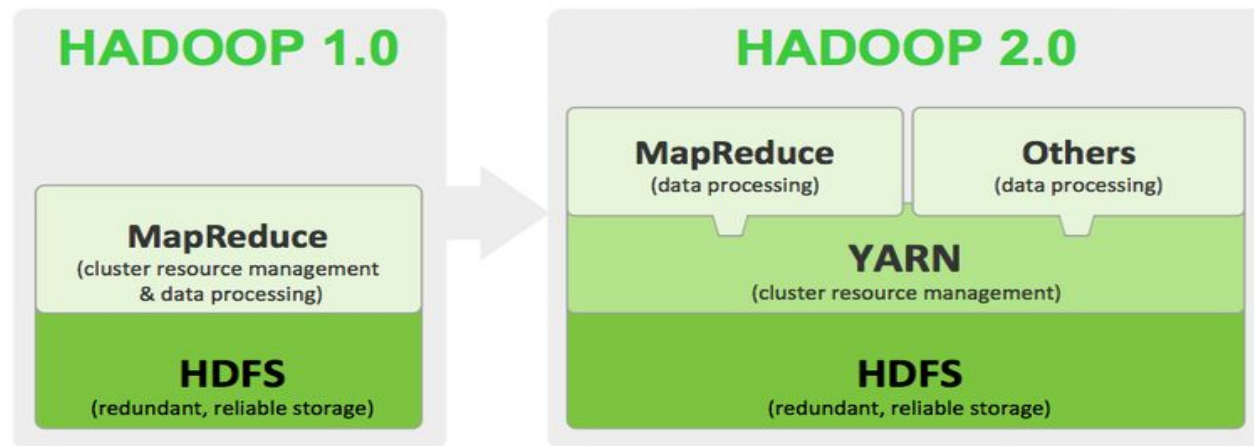
Résumé

- Les algorithmes MapReduce permettent de traiter rapidement de gros volumes de données. En effet, le travail est découpé en petits morceaux qui sont alors traités par plusieurs machines simultanément.
- La première étape est le « **Mapping** ». Elle est effectuée par les mappers. Elle consiste à transformer les données sous forme d'un tuple (clé ; valeur).
- La deuxième étape est le « **Reducing** ». Elle est effectuée par les reducers (un reducer par clé). Elle consiste à effectuer une fonction sur les données afin d'obtenir le résultat voulu.
- Les données calculées sont ensuite regroupées et transmises en tant qu'output à l'utilisateur.



Résumé

- Dans MapReduce version 1, Hadoop fait intervenir des JVMs appelées les Tasktrackers et le jobTracker.
 - Le jobTracker est chargé de manager le travail sur les Tasktrackers et de communiquer avec l'utilisateur.
 - Les Tasktrackers sont chargé de faire les mapping et reducing.
- Dans Hadoop MapReduce version 2, l'allocation des ressources a été séparée de MapReduce. La brique YARN fait son apparition. Elle met en jeu des ressources managers et des nodes managers.
- Avec l'introduction de YARN, il est maintenant possible de substituer la brique Hadoop MapReduce avec d'autres briques comme Spark que nous verrons la prochaine fois.



Résumé

- Bien que traditionnellement écrits en Java, il est possible depuis Hadoop 0.14 de développer des applications Hadoop MapReduce dans bien d'autres langages grâce à l'outil `hadoop streaming`.
- Pour créer un programme MapReduce, on commence toujours par le créer en local avec un petit jeu de données et le « python trick »:

```
cat data.txt | mapper.py | sort | reducer.py
```

- On passe ensuite à l'exécution sur un cluster et un vrai jeu de données grâce à l'outil `Hadoop streaming` tel que:

```
$HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-streaming-X.Y.Z.jar \  
-input input_dirs \  
-output output_dir \  
-mapper $path/mapper.py \  
-reducer $path/reducer.py
```

Panorama des projets Hadoop

Les distributions Hadoop

Les projets Hadoop

Une mission réalisée chez Natixis

Les distributions Hadoop

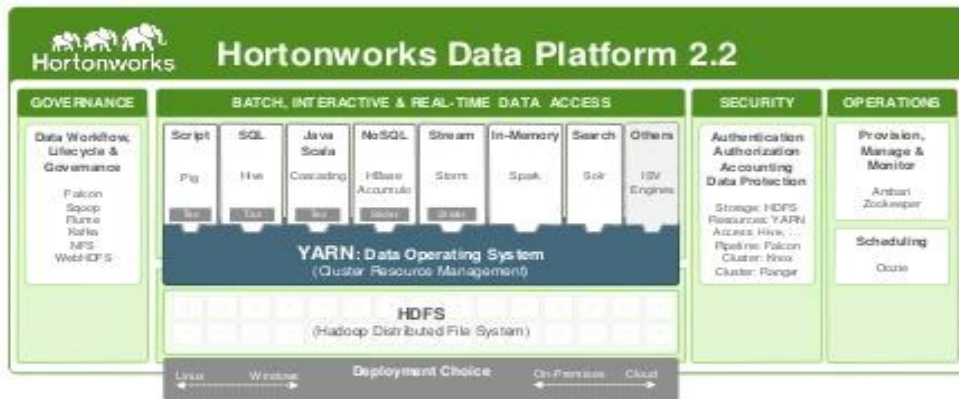
- Pour obtenir Hadoop, il est évidemment possible d'installer soi-même la base du framework (HDFS + Hadoop MapReduce + YARN) et de l'enrichir avec différents projets open-source.
- Cela étant, on peut adopter une stratégie plus rapide et plus industrielle en passant par une distribution.
- Les distributions sont distribuées par des entreprises éditrices et sont des solutions Hadoop prêtes à l'emploi. L'éditeur peut également assurer le support commercial et le service client.
- Selon la distribution, on retrouve parfois des produits différents, dont certains sont parfois développés par l'éditeur lui-même.



Les distributions Hadoop

- Les distributions connaissent des différences entre elles. Elles ne répondent parfois pas au même besoin:
- **MapR:** pile logicielle très fonctionnelle, elle possède une grande simplicité de mise en œuvre. MapR propose également de nombreux outils d'intégration de données (sqoop, hue...). En revanche, la distribution possède de nombreux outils fait-maisons qui s'éloignent de l'esprit Apache.
- **Cloudera:** Une distribution robuste et puissante boostée par l'outil fait-maison « Cloudera Manager ». La distribution possède également Spark et Impala qui sont parmi les briques les plus performantes du moment.
- **Hortonworks:** spin-off de « yahoo! », tous ses outils sont open-sources. A privilégié Spark à Impala. Equipe commerciales et techniques présentes en France.

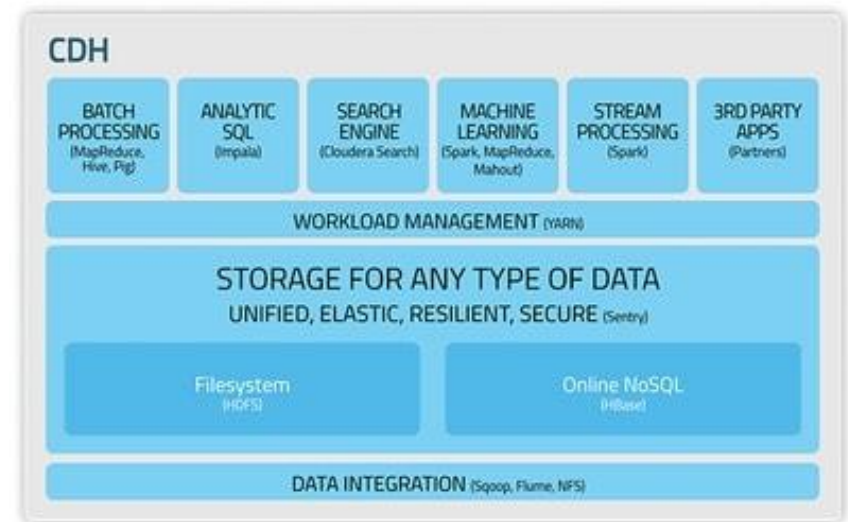
Les distributions Hadoop



Hortonworks

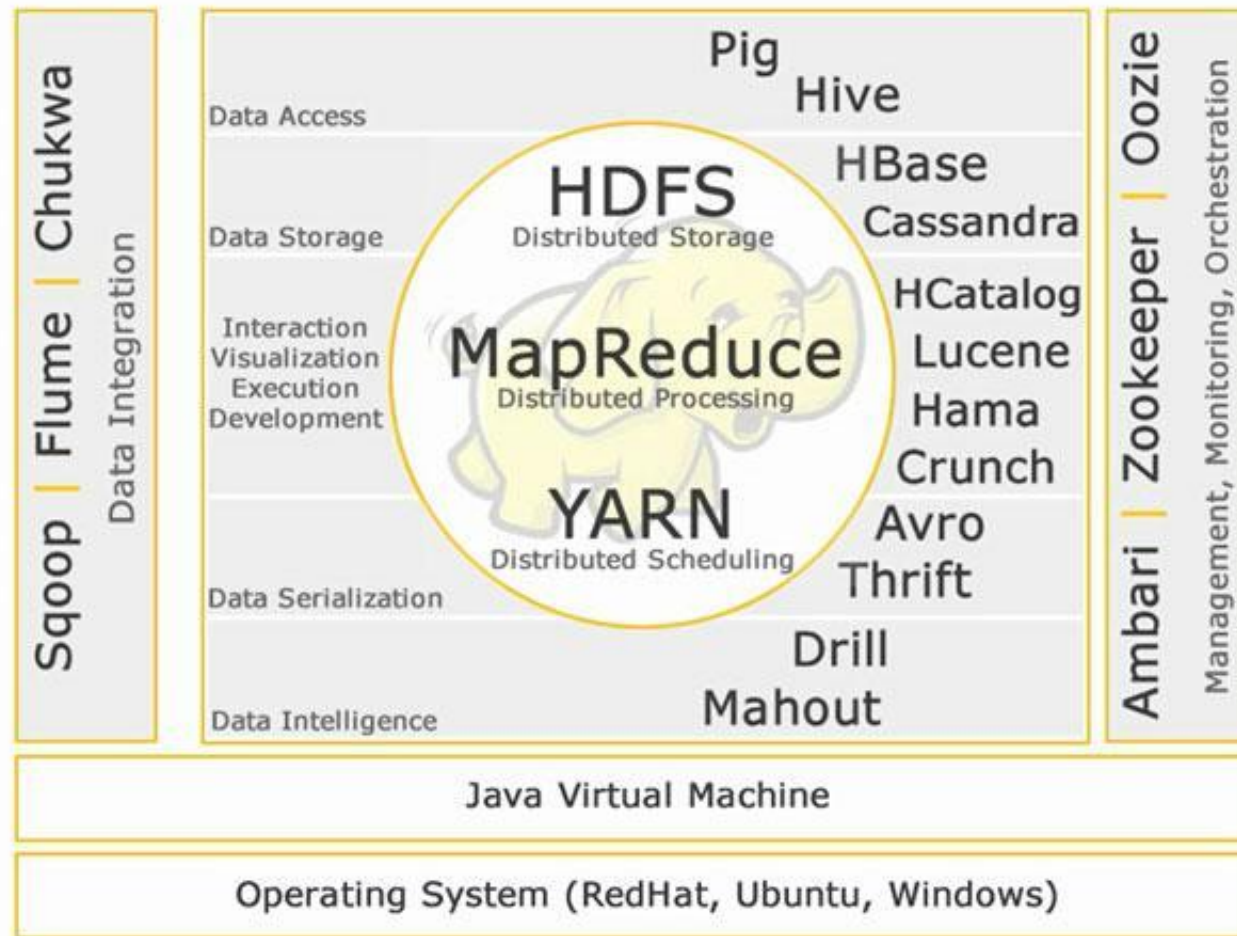


MapR



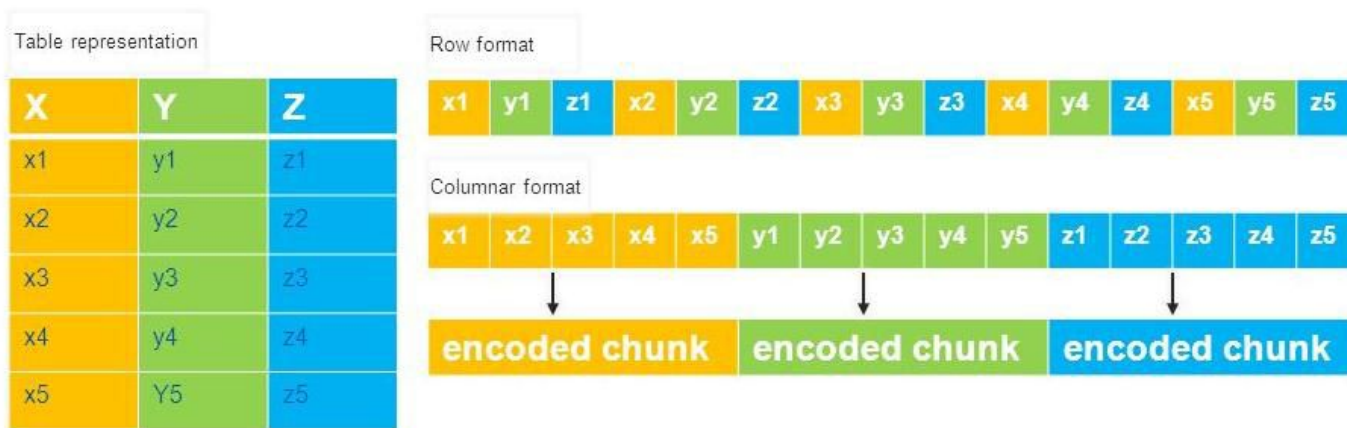
Cloudera

Les projets Hadoop : Overview



Les projets Hadoop : HBase

- Hbase est un outil développé sous la fondation Apache. C'est un système de stockage et de gestion de bases de données.
- Concurrent de HDFS, il peut efficacement le remplacer comme brique de stockage.
- Hbase est une base de données dite orientée colonne.
 - Concept avantageux lorsque de nombreuses lignes mais peu de colonnes.
 - particulièrement adapté pour stocker de gros volumes mais perte de compréhension.



Les projets Hadoop : Pig

- Pig est une plate-forme de programmation haut-niveau développée par Yahoo! En 2006.
- Pig permet de développer plus facilement des programmes MapReduce en évitant notamment d'écrire explicitement les fonctions Map et Reduce.
- il utilise un langage appelé le « Pig Latin ». Celui-ci possède un fort niveau d'abstraction sur la brique MapReduce.

```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```



Les projets Hadoop : Hive

- Hive est un entrepôt de données / base de données initialement développé par facebook entre 2007 et 2008. Il est capable de structurer la donnée en dataframe.
- Il utilise un langage appelé le HiveQL, très proche en fonctionnalités du SQL-92.
- Les requêtes effectuées avec Hive sont implicitement converties en jobs MapReduce.



Query

Function	MySQL	HiveQL
Retrieving information	SELECT from_columns FROM table WHERE conditions;	SELECT from_columns FROM table WHERE conditions;
All values	SELECT * FROM table;	SELECT * FROM table;
Some values	SELECT * FROM table WHERE rec_name = "value";	SELECT * FROM table WHERE rec_name = "value";
Multiple criteria	SELECT * FROM table WHERE rec1="value1" AND rec2="value2";	SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";
Selecting specific columns	SELECT column_name FROM table;	SELECT column_name FROM table;
Retrieving unique output records	SELECT DISTINCT column_name FROM table;	SELECT DISTINCT column_name FROM table;
Sorting	SELECT col1, col2 FROM table ORDER BY col2;	SELECT col1, col2 FROM table ORDER BY col2;
Sorting backward	SELECT col1, col2 FROM table ORDER BY col2 DESC;	SELECT col1, col2 FROM table ORDER BY col2 DESC;
Counting rows	SELECT COUNT(*) FROM table;	SELECT COUNT(*) FROM table;
Grouping with counting	SELECT owner, COUNT(*) FROM table GROUP BY owner;	SELECT owner, COUNT(*) FROM table GROUP BY owner;
Maximum value	SELECT MAX(col_name) AS label FROM table;	SELECT MAX(col_name) AS label FROM table;
Selecting from multiple tables (Join same table using alias w/"AS")	SELECT pet.name, comment FROM pet, event WHERE pet.name = event.name;	SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name);

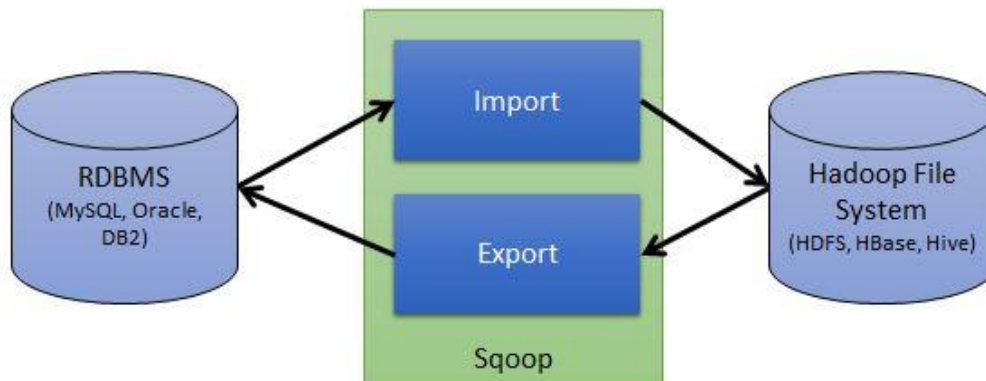
Les projets Hadoop : Mahout

- Mahout est un projet Hadoop qui permet de faire du machine Learning au-dessus des briques HDFS, MapReduce et Spark Core.
- Mahout possède les fonctionnalités suivantes:
 - Classification (logistic regression, Naïve Bayes...)
 - Clustering (k-means, fuzzy k-means...)
 - Dimension reduction (PCA, Singular Value Decomposition...)
- Mahout est parfois cité comme un projet vieillissant comparé à certaines librairies comme Spark MLlib / Spark ML.



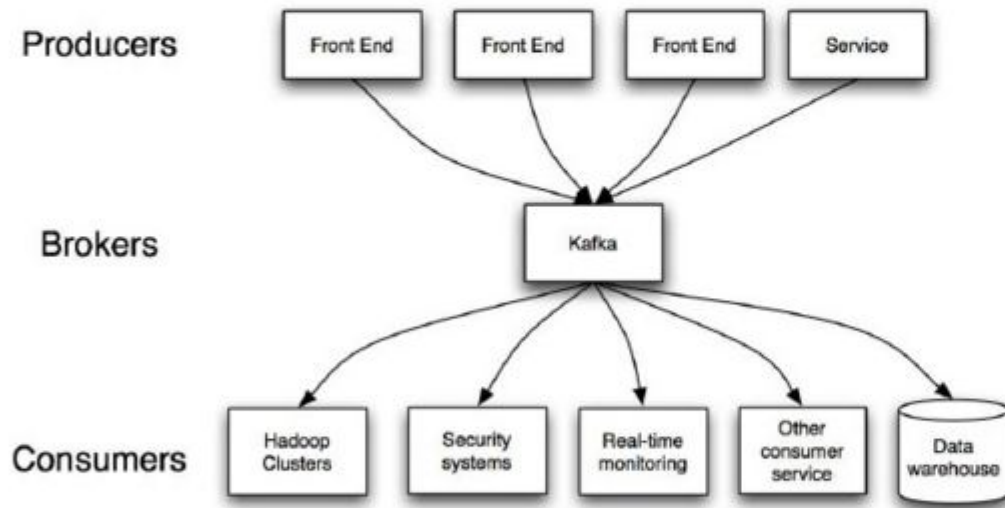
Les projets Hadoop : Sqoop

- Sqoop (contraction de SQL et Hadoop) est un projet Apache qui a pour but de réaliser des importations / exportations entre:
 - Une base de données SQL (MySQL, Oracle, PostgreSQL).
 - Le système de stockage distribué HDFS, Hbase...
- Les imports/exports réalisés par sqoop sont des traitements batchs (données chargées d'un seul coup)



Les projets Hadoop : Kafka

- Outil popularisé par linkedin qui l'a développé afin de pouvoir construire ses fils d'actualités.
- Il est destiné à l'ingestion et la manipulation de données en temps réel.
- L'ingestion réalisée par Kafka est un traitement en streaming (temps réel + données traitées en continues)



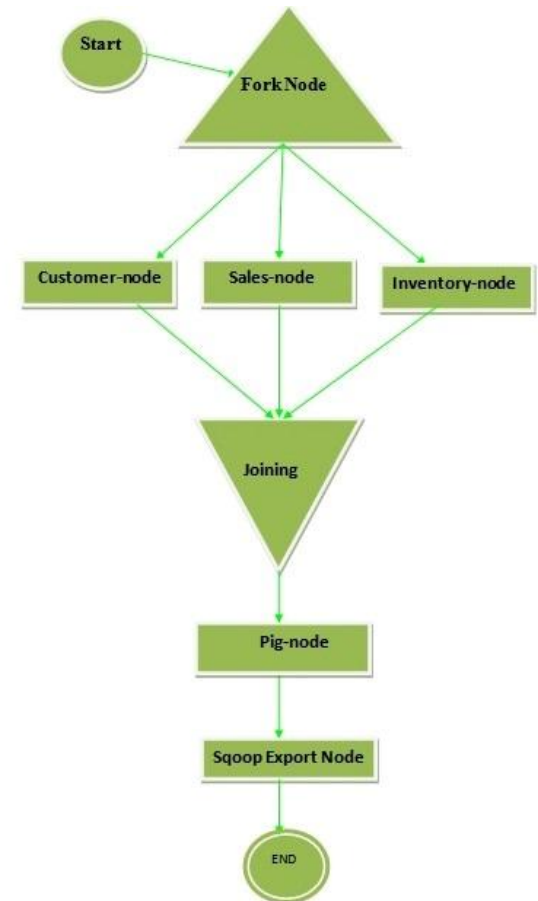
Les projets Hadoop : Ambari

- Présent principalement dans la distribution HortonWorks, il permet de manager le cluster Hadoop de façon graphique et facilement compréhensible.
- Le serveur Ambari est accessible par interface web pour plus de facilité.



Les projets Hadoop : Oozie

- C'est le scheduler de l'écosystème Hadoop.
- Il permet d'automatiser le déclenchement de jobs (applications) issues de l'écosystème Hadoop.
- Ces applications (appelées dans le jargon oozie « **workflow** ») peuvent être déclenchées à heures fixes, de façon périodique ou sur des conditions liées à l'environnement HDFS (apparition d'un fichier, déplacement d'un dossier etc...)
- Il permet également de pouvoir construire des workflows complexes consistant en l'enchaînement de plusieurs job Hadoop.

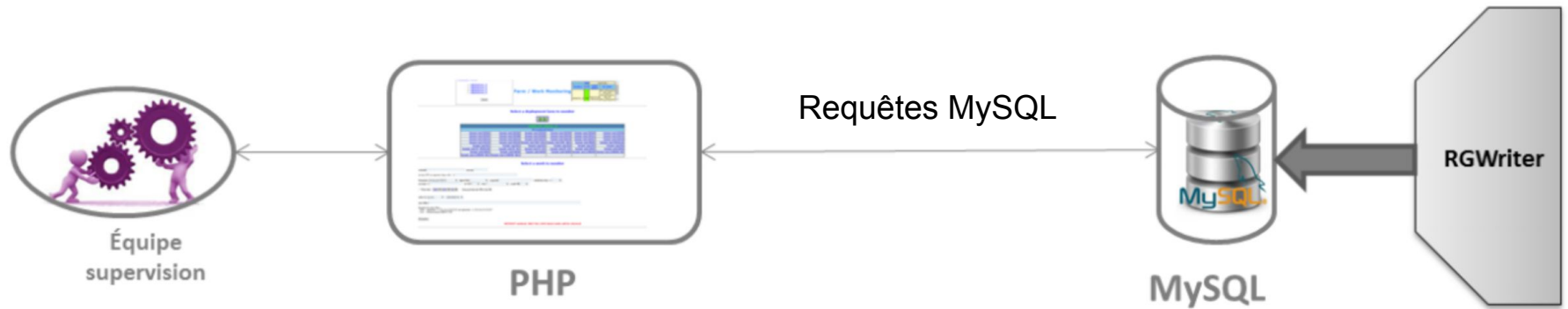


Les projets Hadoop : Spark

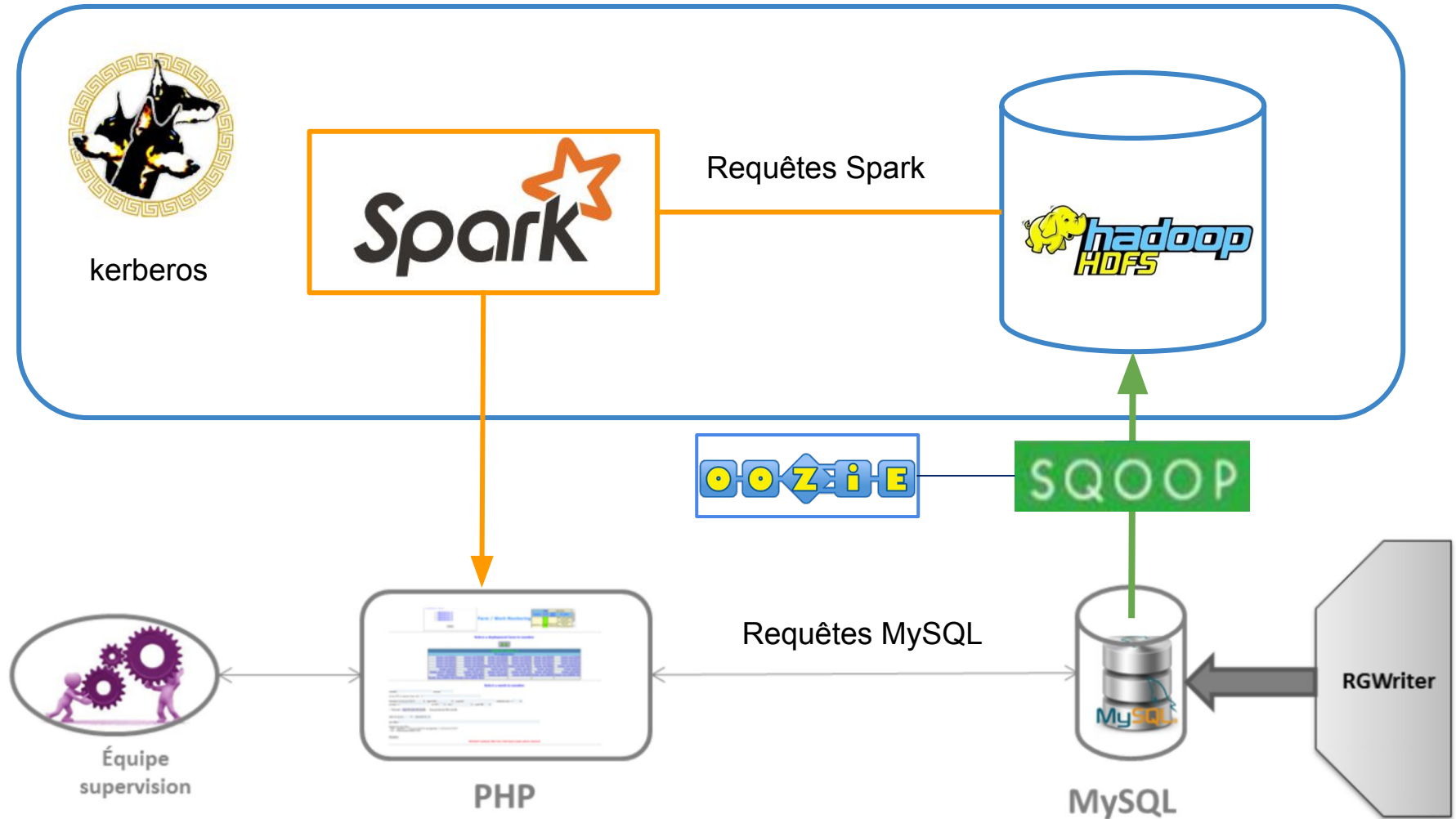


- C'est un outil de data Processing alternatif à Hadoop MapReduce.
- Spark possède de bien meilleures performances en terme de temps de latence et a maintenant complètement dépassé Hadoop MapReduce.
- Il permet une véritable programmation distribuée avec plusieurs API (Scala, Python, Java et R).
- Il permet également de pouvoir exécuter des tâches de type SQL, Machine learning, calcul de graphe en distribué.

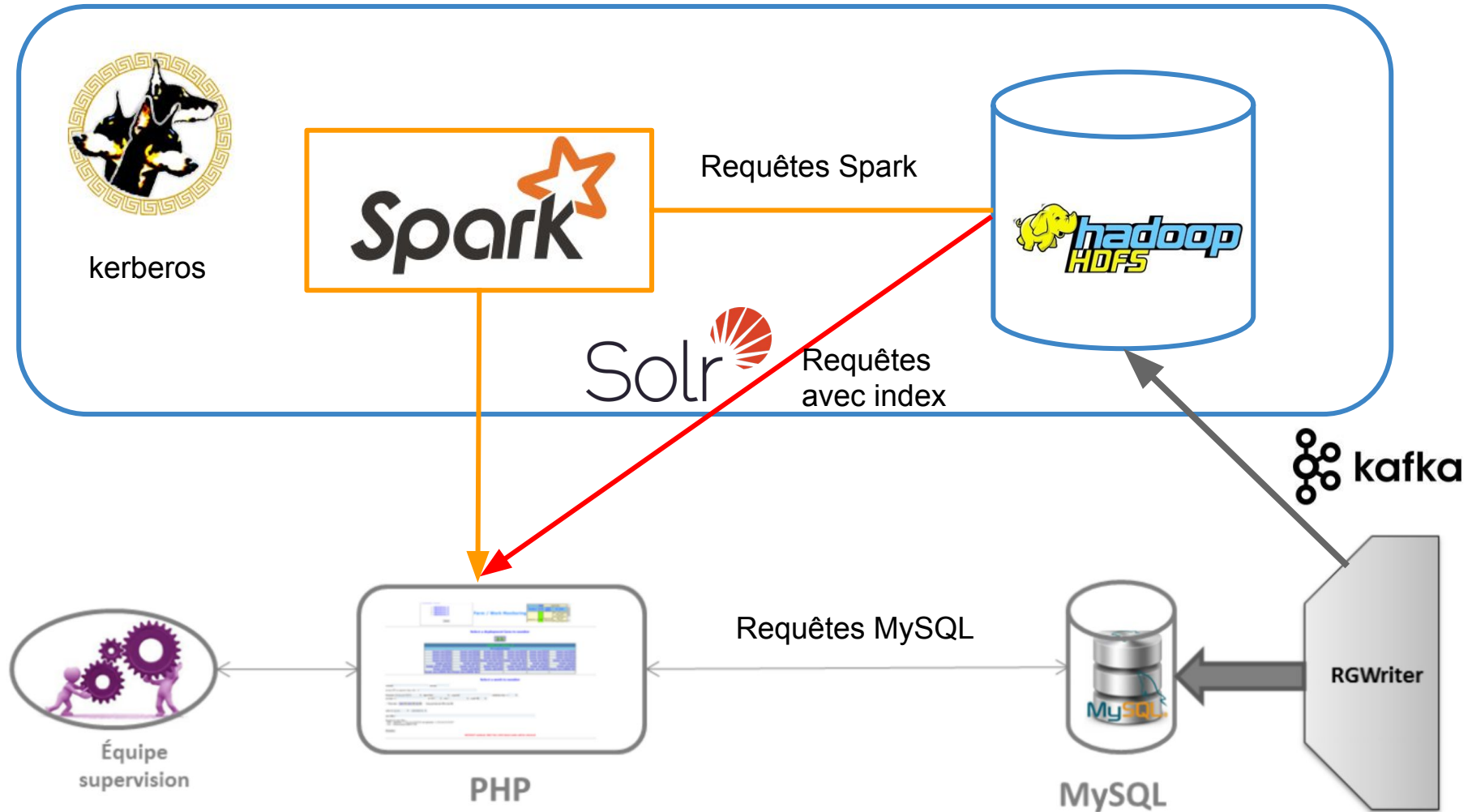
Une mission réalisée chez Natixis : Avant



Une mission réalisée chez Natixis : Maintenant



Une mission réalisée chez Natixis : A venir



Conclusion

- Hadoop est un framework composé de briques projets (stockage, calcul, intégration, management..) interchangeables les une avec les autres.
- Les briques principales du framework Hadoop sont:
 - Hadoop HDFS pour le stockage distribué des données
 - Hadoop MapReduce pour le calcul distribué sur les données
- La programmation en MapReduce reste cependant complexe et demande de fortes compétences en programmation. Heureusement, il existe des modules Hadoop préprogrammés utilisant HDFS et MapReduce et permettant de faire des traitement beaucoup plus complexes. Et ce, de façon transparente pour l'utilisateur!
- De nombreux projets s'appuient sur ces briques de base, notamment le fameux Framework Spark, utilisé de plus en plus par les data scientists pour effectuer du Big data Analytics !

