

# Chp1 : Introduction Approche et Vocabulaire

# Faits

## Introduction aux Big Data

---

- Chaque jour, nous générerons 2,5 trillions d'octets de données
- 90% des données dans le monde ont été créées au cours des **deux dernières années**
- 90% des données générées sont **non structurées**
- Source:
  - Capteurs utilisés pour collecter les informations climatiques
  - Messages sur les médias sociaux
  - Images numériques et vidéos publiées en ligne
  - Enregistrements transactionnels d'achat en ligne
  - Signaux GPS de téléphones mobiles
  - ...
- Données appelées **Big Data** ou **Données Massives**

# Intérêts

## Introduction aux Big Data

1 / 3

- Chefs d'entreprise prennent fréquemment des décisions basées sur des informations en lesquelles ils n'ont pas confiance, ou qu'ils n'ont pas

1 / 2

- Chefs d'entreprise disent qu'ils n'ont pas accès aux informations dont ils ont besoin pour faire leur travail

83 %

- Des DSI (Directeurs des SI) citent : « L'informatique décisionnelle et analytique » comme faisant partie de leurs plans pour améliorer leur compétitivité

60 %

- Des PDG ont besoin d'améliorer la capture et la compréhension des informations pour prendre des décisions plus rapidement

# Sources

## Introduction aux Big Data

---

- Sources multiples: sites, bases de données, téléphones, serveurs:
  - Déetecter les sentiments et réactions des clients
  - Déetecter les conditions critiques ou potentiellement mortelles dans les hôpitaux , et à temps pour intervenir
  - Prédire des modèles météorologiques pour planifier l'usage optimal des éoliennes
  - Prendre des décisions risquées basées sur des données transactionnelles en temps réel
  - Identifier les criminels et les menaces à partir de vidéos, sons et flux de données
  - Étudier les réactions des étudiants pendant un cours, prédire ceux qui vont réussir, d'après les statistiques et modèles réunis au long des années (domaine *Big Data in Education*)

# Challenges

## Introduction aux Big Data

---

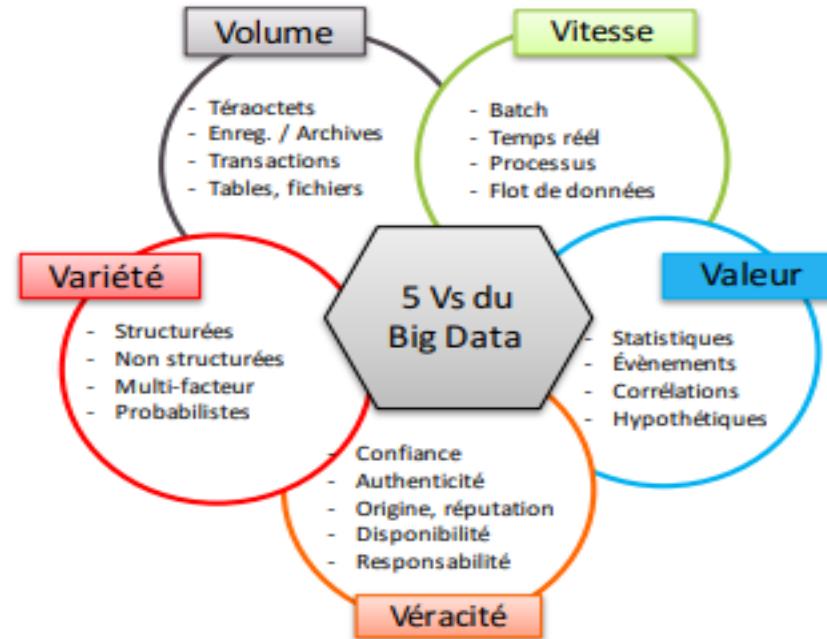
- Réunir un grand volume de données variées pour trouver de nouvelles idées
- Capturer des données créées rapidement
- Sauvegarder toutes ces données
- Traiter ces données et les utiliser

# Les 5 V

## Introduction aux Big Data

- Extraction d'informations et décisions à partir de données, caractérisées par les 5 V:

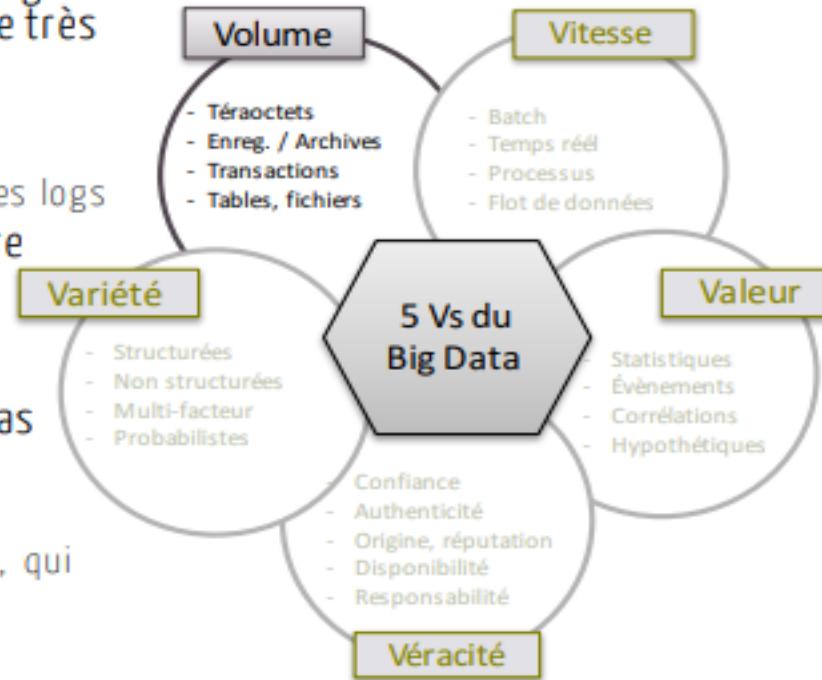
- Volume (*Volume*)
- Variété (*Variety*)
- Vitesse (*Velocity*)
- Véracité (*Veracity*)
- Valeur (*Value*)



# Volume

## Introduction aux Big Data

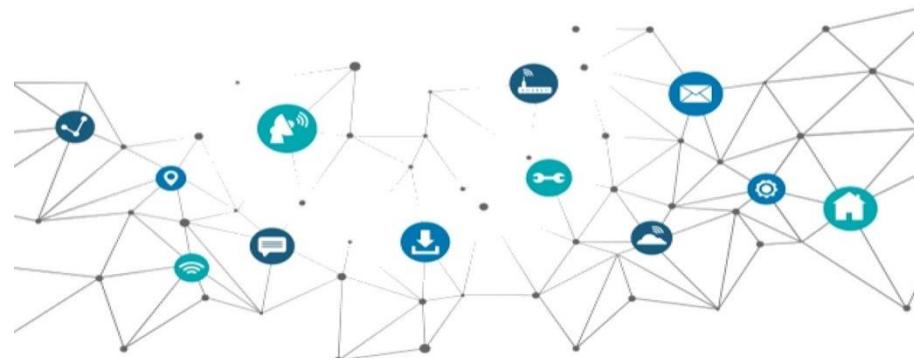
- Le prix de stockage des données a beaucoup diminué ces 30 dernières années:
  - De \$100,000 / Go (1980)
  - À \$0.10 / Go (2013)
- Les lieux de stockage fiables (comme des SAN: Storage Area Network) ou réseaux de stockage peuvent être très coûteux
  - Choisir de ne stocker que certaines données, jugées sensibles
  - Perte de données, pouvant être très utiles, comme les logs
- Comment déterminer les données qui méritent d'être stockées?
  - Transactions? Logs? Métier? Utilisateur? Capteurs? Médicales? Sociales?
- ➔ **Aucune donnée n'est inutile.** Certaines n'ont juste pas encore servi.
- Problèmes:
  - Comment stocker les données dans un endroit fiable, qui soit moins cher
  - Comment parcourir ces données et en extraire des informations facilement et rapidement?



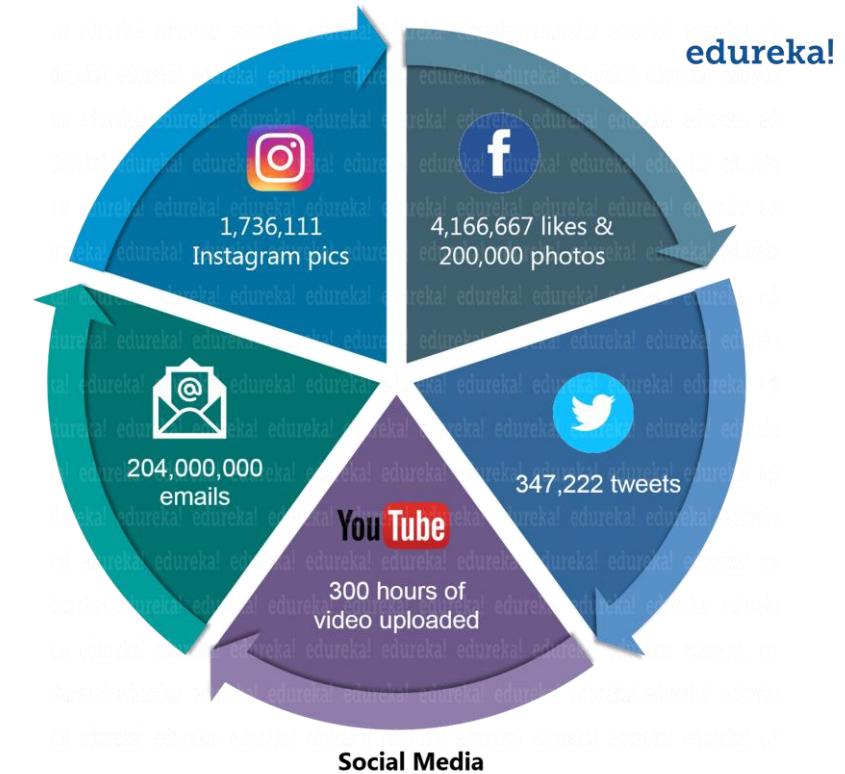
La caractéristique de volume, qui implique que la taille des données augmente de façon régulière, fait qu'on ne peut plus se contenter d'un système centralisé classique.



## INTERNET OF THINGS



Volume => Scalabilité Horizontale



## Scale UP



Augmentation des capacités du serveur de stockage en rajoutant des processeurs, de la RAM ou des disques.

## Scale OUT

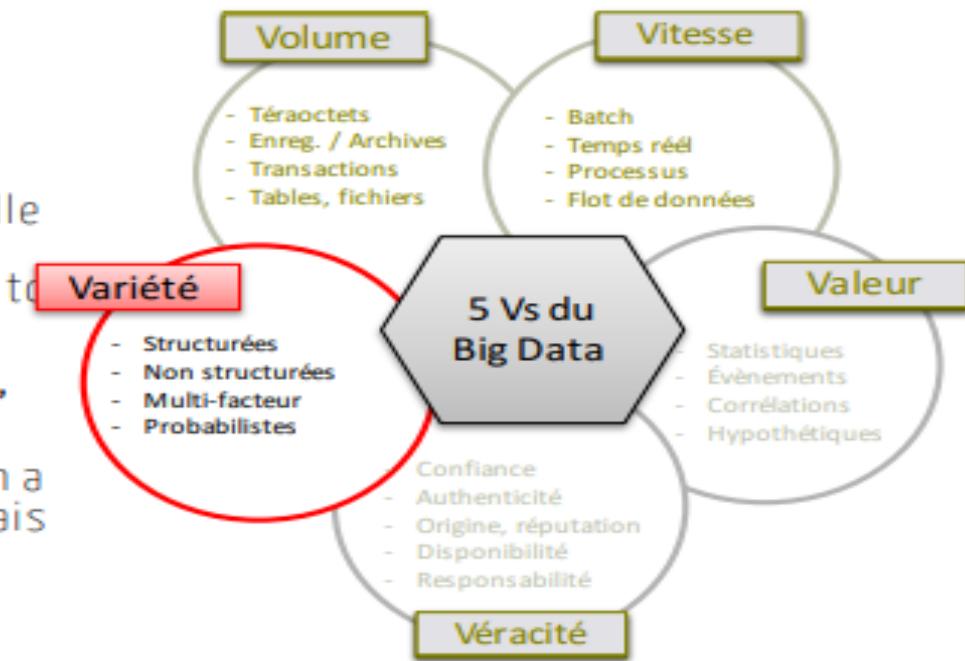


Un volume de données toujours en augmentation, il serait plus judicieux de rajouter des machines au besoin, créant ainsi un cluster de machines interconnectées, ou système réparti, dont la taille et la capacité sont virtuellement illimitées

# Variété

## Introduction aux Big Data

- Pour un stockage dans des bases de données ou dans des entrepôts de données, les données doivent respecter un format prédéfini.
- La plupart des données existantes sont non-structurées ou semi-structurées
- Données sous plusieurs formats et types
- On veut tout stocker:
  - Exemple:** pour une discussion dans un centre d'appel, on peut la stocker sous forme textuelle pour son contenu, comme on peut stocker l'enregistrement en entier, pour interpréter le ton de voix du client
- Certaines données peuvent paraître obsolètes, mais sont utiles pour certaines décisions:
  - Exemple:** Pour le transport de marchandise, on a tendance à choisir le camion le plus proche. Mais parfois, ce n'est pas la meilleure solution. D'autres problèmes peuvent intervenir.
  - Besoin de:** Données GPS, Plan de livraison du camion, Circulation, Chargement du camion, Niveau d'essence...

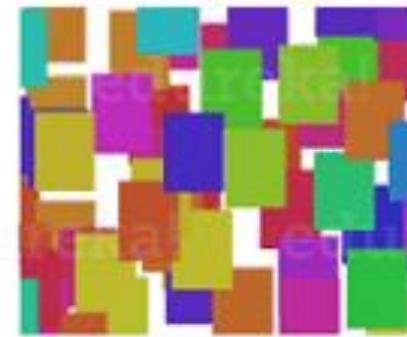




**Structured Data**



**Semi Structured Data**



**Unstructured Data**

**Variété** La variété de données implique non seulement que nous sommes en présence de données structurées, semi-structurées et non structurées, mais également que ces données peuvent parvenir de sources différentes, avec des formats différents, et que même à partir d'une même source, ce format peut changer d'un moment à un autre.

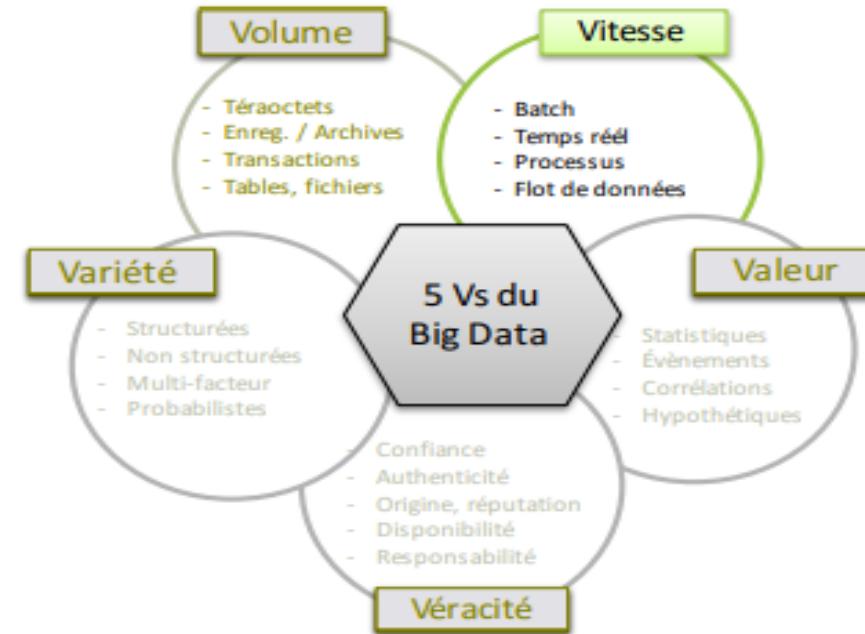
C'est pour ces raisons qu'un système Big Data se doit de supporter des types de données changeants, sans pour autant requérir à des subterfuges qui alourdissent ou contournent le système de stockage.

**Variété => Flexibilité**

# Vitesse

## Introduction aux Big Data

- Rapidité d'arrivée des données
- Vitesse de traitement
- Les données doivent être stockées à l'arrivée, parfois même des Teraoctets par jour
  - Sinon, risque de perte d'informations
- Exemple
  - Il ne suffit pas de savoir quel article un client a acheté ou réservé
  - Si si on sait que vous avez passé plus de 5mn à consulter un article dans une boutique d'achat en ligne, il est possible de vous envoyer un email dès que cet article est soldé.



Vitesse = vélocité est une propriété qui, couplée au volume, gérer des données en continue arrivée implique qu'il y'a un risque énorme de perte de données, si elles ne sont pas manipulées à temps. C'est pour cette raison qu'un système Big Data se doit d'être continuellement disponible

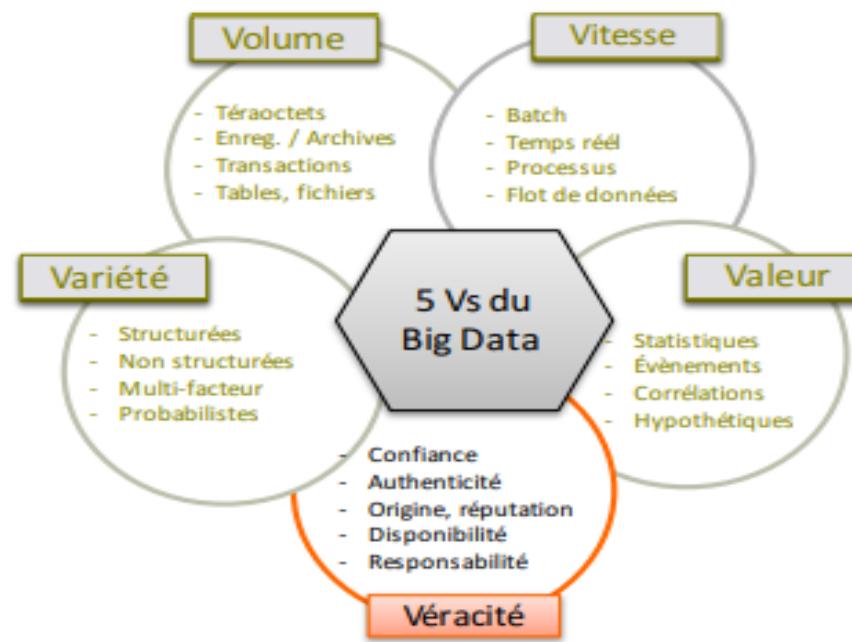


Vélocité => Disponibilité

# Véracité

## Introduction aux Big Data

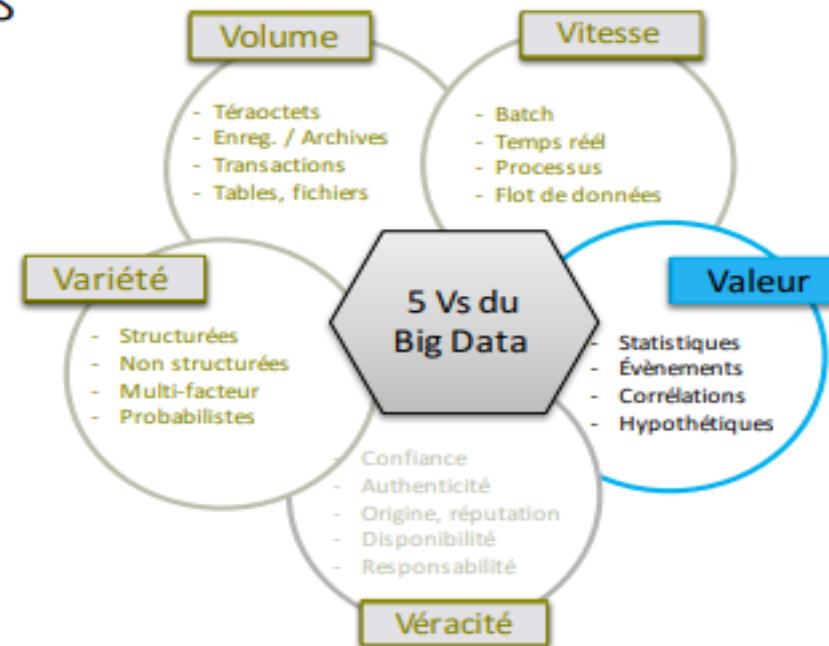
- Cela fait référence au désordre ou la fiabilité des données. Avec l'augmentation de la quantité, la qualité et précision se perdent (abréviations, typos, déformations, source peu fiable...)
- Les solutions Big Data doivent remédier à cela en se référant au **volume** des données existantes
- Nécessité d'une (très) grande rigueur dans l'organisation de la **collecte** et le **recouplement, croisement, enrichissement** des données pour lever l'incertitude et la nature imprévisible des données introduites dans les modèles mais aussi pour respecter le cadre légal pour créer la **confiance** et garantir la **sécurité** et **l'intégrité** des données.



# Valeur

## Introduction aux Big Data

- Le V le plus important
- Il faut transformer toutes les données en valeurs exploitables: les données sans valeur sont inutiles
- Atteindre des objectifs stratégiques de création de valeur pour les clients et pour l'entreprise dans tous les domaines d'activité



# Théorème CAP

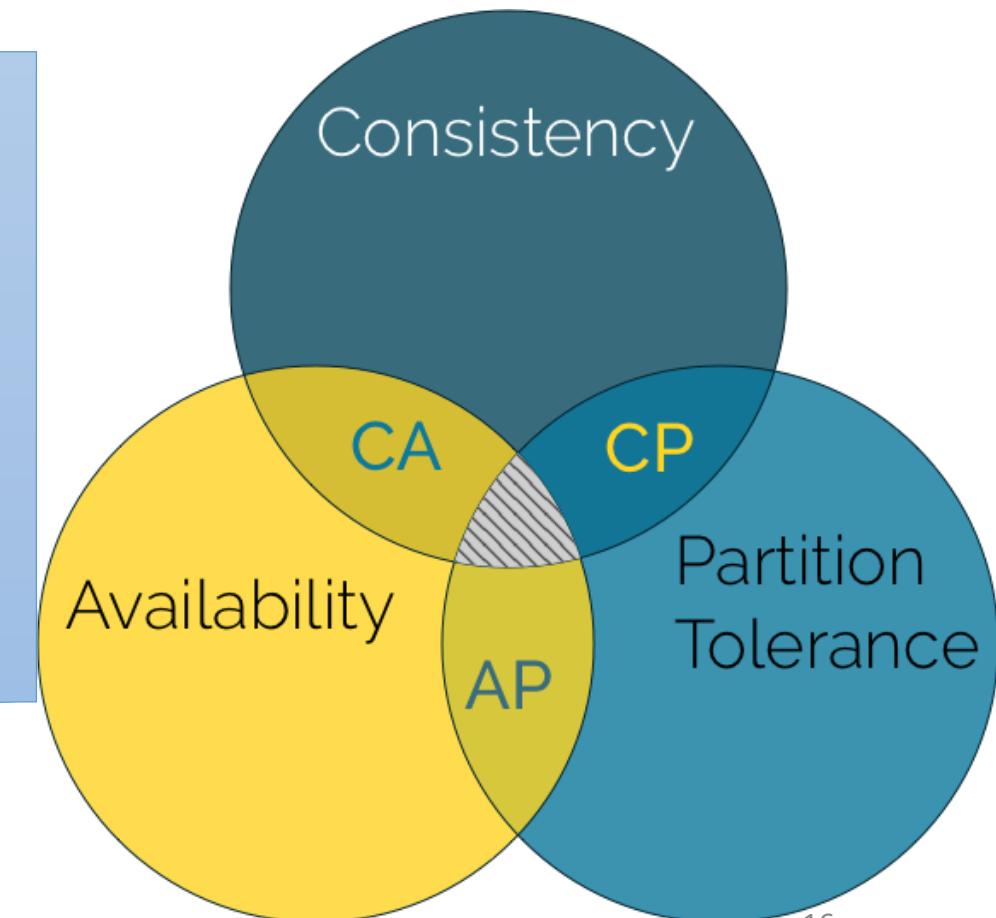
Le théorème de CAP dit :

Dans toute base de données, vous ne pouvez respecter au plus que 2 propriétés parmi la *cohérence*, la *disponibilité* et la *distribution*.

1. *Consistency* (Cohérence) : Une donnée n'a qu'un seul état visible quel que soit le nombre de réplicas

2. *Availability* (Disponibilité) : Tant que le système tourne (distribué ou non), la donnée doit être disponible

3. *Partition Tolerance* (Distribution) : Quel que soit le nombre de serveurs, toute requête doit fournir un résultat correct



# Principes de base du Domaine des Big Data

**Stocker d'abord, réfléchir ensuite**



À cause de la vitesse, il est important de considérer qu'il nous sera parfois difficile, voire impossible, de nettoyer les données ou de faire un traitement quelconque dessus, avant de les stocker. Cela risque dans bien des cas de nous faire perdre des données, le cauchemar de tout scientifique des données!

Nous devons donc envisager la possibilité de définir des systèmes de stockage qui contiennent des données non nettoyées, en vrac (appelées *raw data*), pour ensuite lancer des traitements dessus.. L'horreur pour un gestionnaire de bases des données! 😱

**Absolument TOUTES les données sont importantes!**

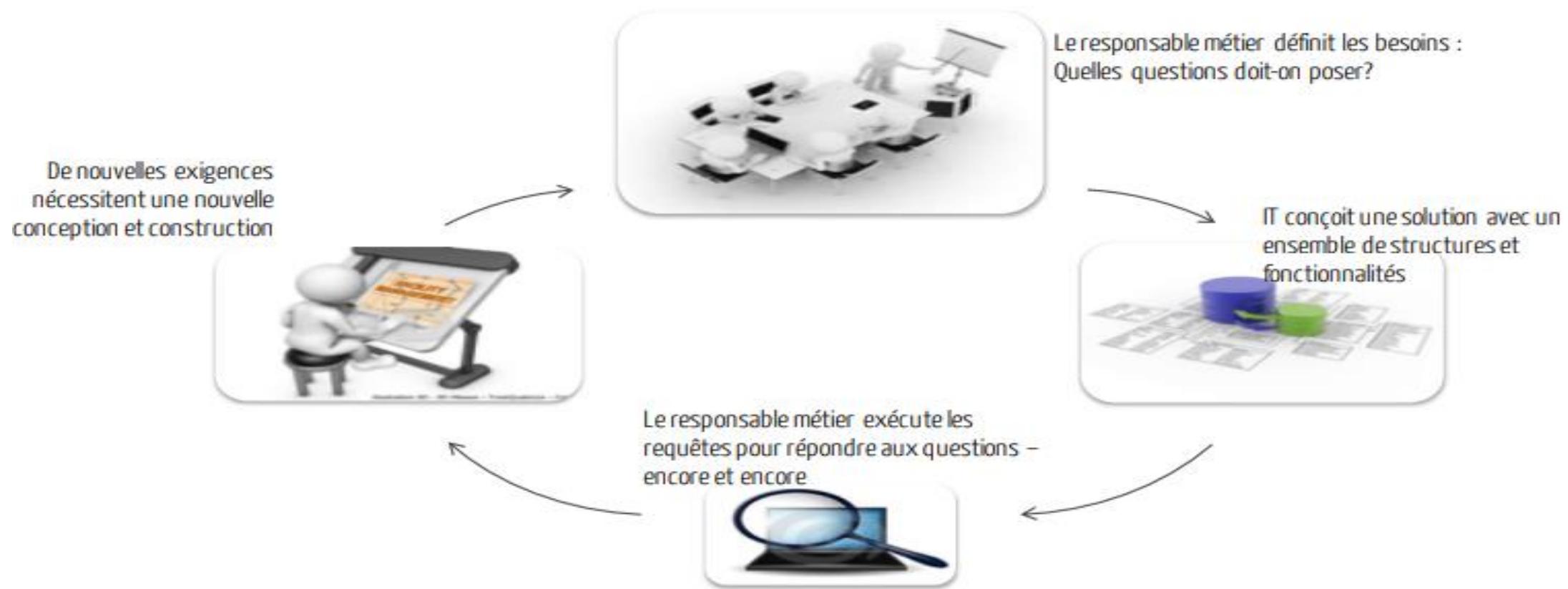
**Ce sont les données qui pilotent le traitement**

**La redondance, c'est bien**

# Approche Traditionnelle

## Introduction aux Big Data

Les besoins métier guident la conception de la solution



# Approche Traditionnelle

## Introduction aux Big Data

---

- Appropriée pour:
  - Des données structurées
  - Opérations et processus répétitifs
  - Sources relativement stables
  - Besoins bien compris et bien cadrés

# Approche Big Data

## Introduction aux Big Data

Les sources d'information guident la découverte créative

De nouvelles idées conduisent à l'intégration de technologies traditionnelles



Le responsable métier et IT identifient les sources de données disponibles



Le responsable métier détermine les questions à poser en explorant les données et relations entre elles



IT fournit une plateforme qui permet une exploration créative de toutes les données disponibles



# Approche Big Data vs Approche Traditionnelle

## Introduction aux Big Data

- La question **n'est pas**:
  - Dois-je choisir entre l'approche classique et l'approche Big Data?
- Mais plutôt:
  - Comment les faire fonctionner ensemble?





# Big Data

Chp2 - Hadoop et MapReduce



# Présentation

« High-Availability Distributed Object-Oriented Platform » est un framework libre et open source écrit en Java destiné à stocker et traiter les données volumineuses sur un cluster.

Il est utilisé par un grand nombre de contributeurs et utilisateurs.

- Hadoop a été conçu par Doug Cutting en 2004 et a été inspiré par les publications Map Reduce, GoogleFS, BigTable de Google.
- Utilisé par les géants du web comme Yahoo!, Twitter, LinkedIn, eBay et Amazon.
- S'occupe de toutes les problématiques liées aux calcul distribué:
  - ❖ La tolérance aux pannes
  - ❖ L'accès et le partage des données.
  - ❖ La répartition des taches aux machines membres du cluster

# Pourquoi Hadoop?

- ✓ Il offre une évolutivité pour stocker de gros volumes de données sur du matériel de base.
- ✓ l'écosystème Hadoop permet de gérer ces différents types de données (texte, csv, image, vidéo, son, signal...).
- ✓ L'écosystème Hadoop a la capacité à faciliter un environnement partagé.(Étant donné que même les clusters de taille modeste peuvent avoir de nombreux cœurs, il est important d'autoriser l'exécution simultanée de plusieurs tâches.)

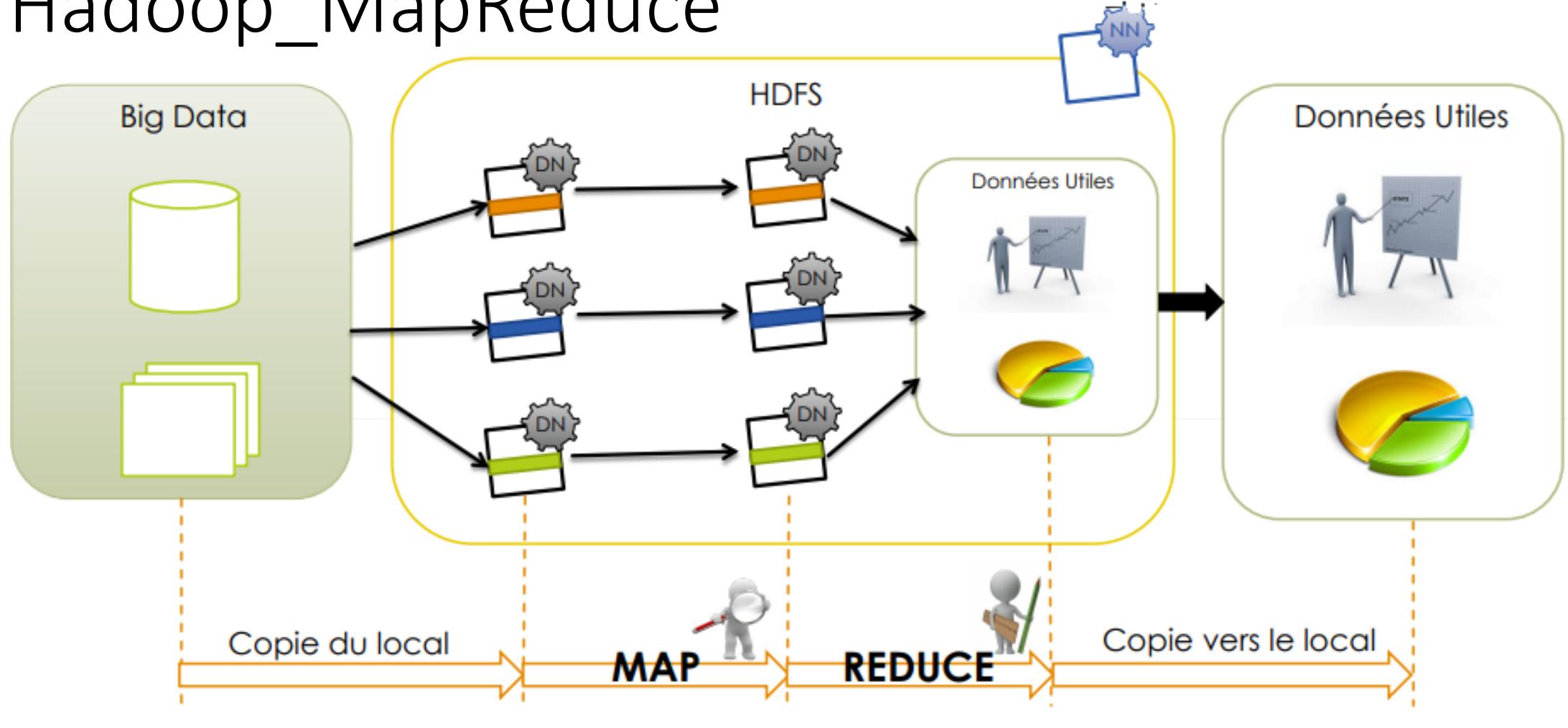
Hadoop assure les critères des solutions du BigData:

- ❖ **Performance**: réaliser d'une façon distribuée des traitements sur des volumes de données massives, de l'ordre de plusieurs péta-octets d'une manière distribuée et en parallèle.
- ❖ **Economie**: contrôle des coûts en utilisant de matériels de calcul de type standard.
- ❖ **Evolutivité** (scalabilité): un plus grand cluster devrait donner une meilleure performance.
- ❖ **Tolérance aux pannes**: la défaillance d'un nœud ne provoque pas l'échec de calcul.
- ❖ **Parallélisme de données**: le même calcul effectué sur toutes les données.

# Hadoop\_MapReduce

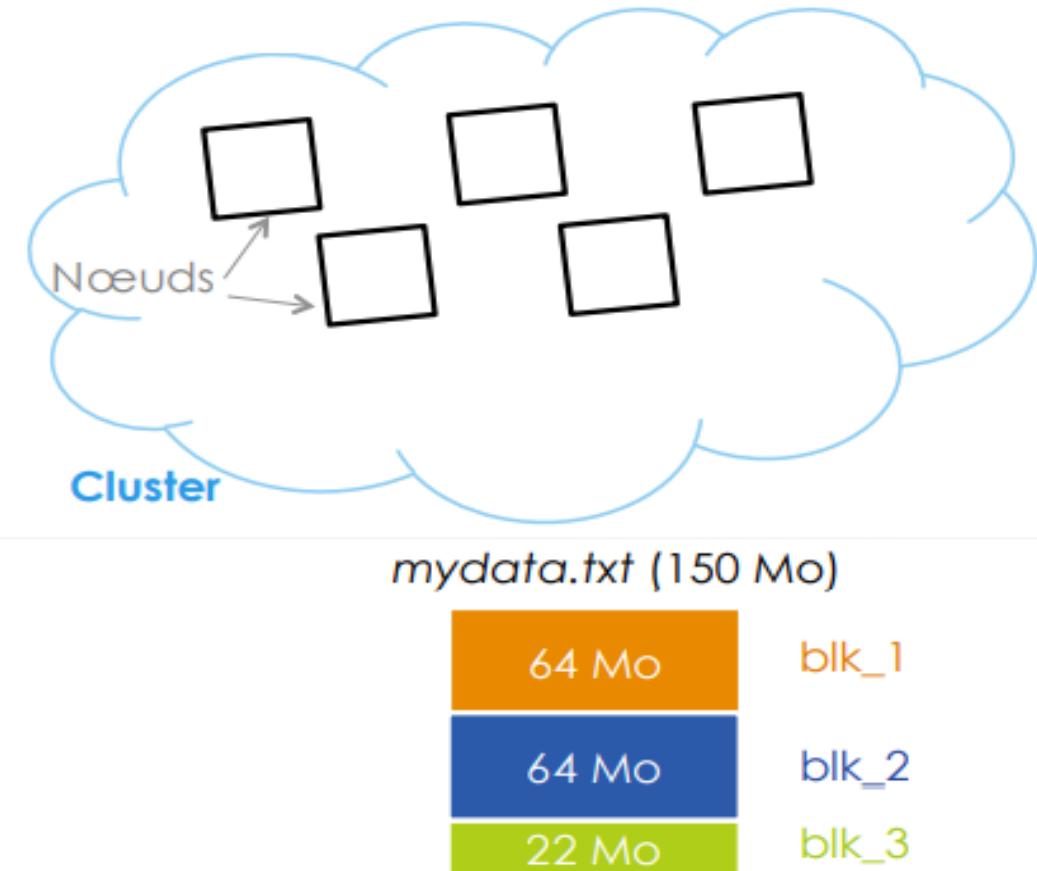
- Le projet Hadoop consiste en **deux** grandes parties:
  - Stockage des données : **HDFS** (*Hadoop Distributed File System*)
  - Traitement des données : **MapReduce**
- Principe :
  - Diviser les données
  - Les sauvegarder sur une collection de machines, appelées *cluster*
  - Traiter les données directement là où elles sont stockées, plutôt que de les copier à partir d'un serveur distribué
- Il est possible d'ajouter des machines à votre cluster, au fur et à mesure que les données augmentent

# Hadoop\_MapReduce



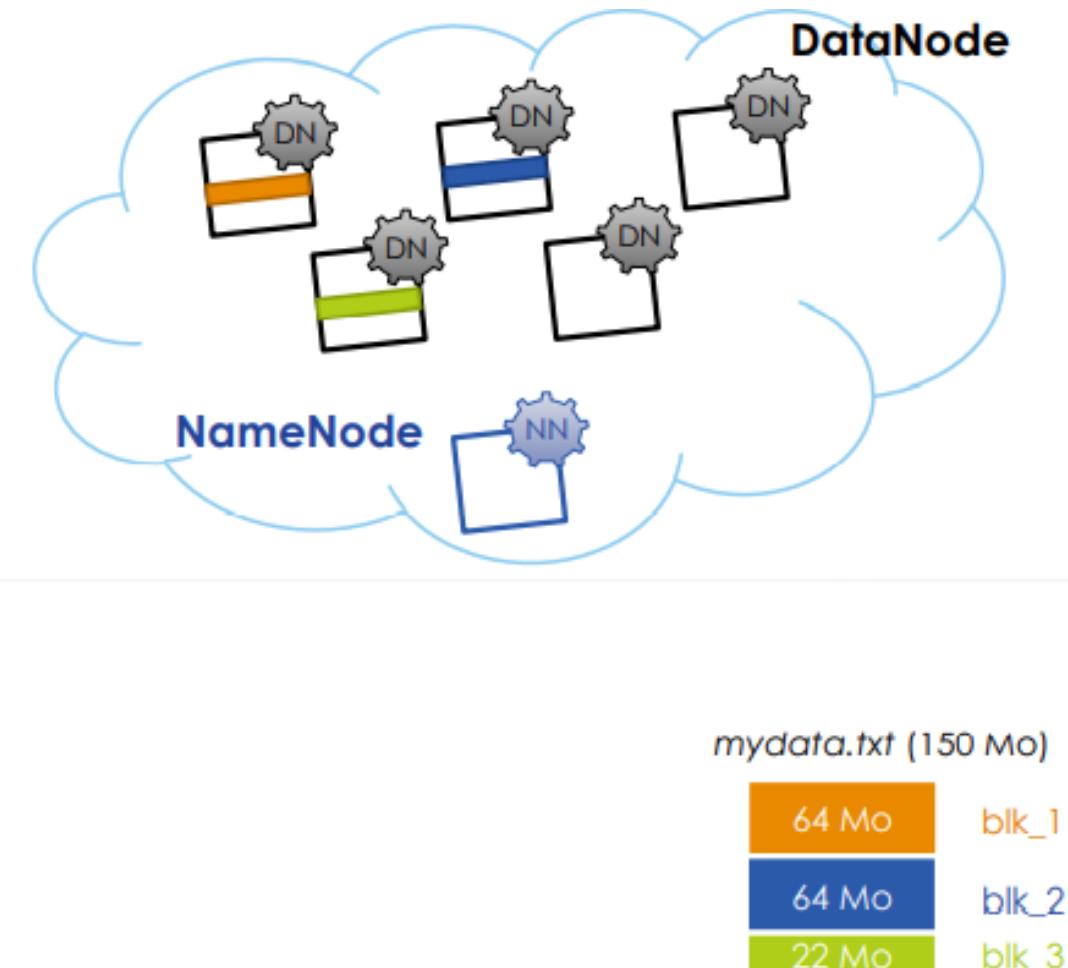
# HDFS : Hadoop Distributed File System

- HDFS est un système de fichiers distribué, extensible et portable
- Ecrit en Java
- Permet de stocker de très gros volumes de données sur un grand nombre de machines (nœuds) équipées de disques durs banalisés ➔ Cluster
- Quand un fichier *mydata.txt* est enregistré dans HDFS, il est décomposé en grands blocs (par défaut 64Mo), chaque bloc ayant un nom unique: *blk\_1*, *blk\_2*...



# HDFS : Hadoop Distributed File System

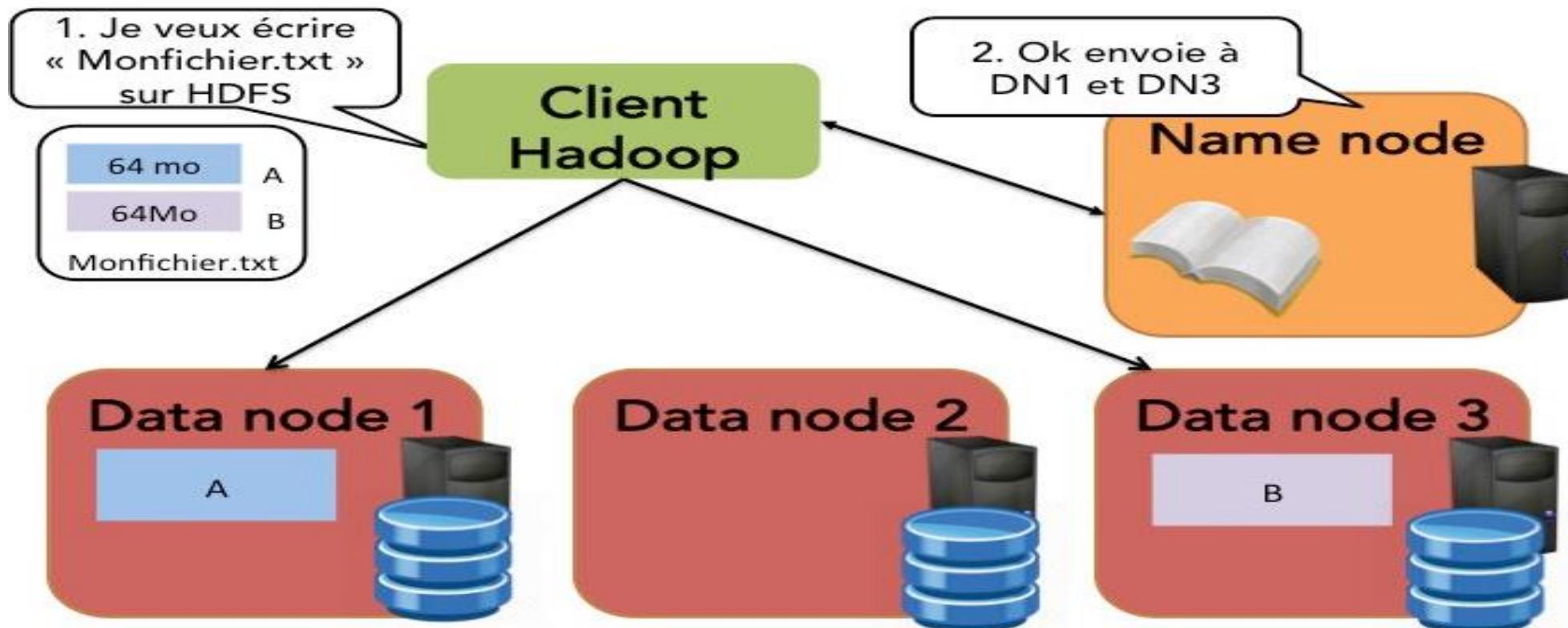
- Chaque bloc est enregistré dans un nœud différent du cluster
- **DataNode** : démon sur chaque nœud du cluster
- **NameNode** :
  - Démon s'exécutant sur une machine séparée
  - Contient des métadonnées
  - Permet de retrouver les nœuds qui exécutent les blocs d'un fichier



# HDFS : Hadoop Distributed File System

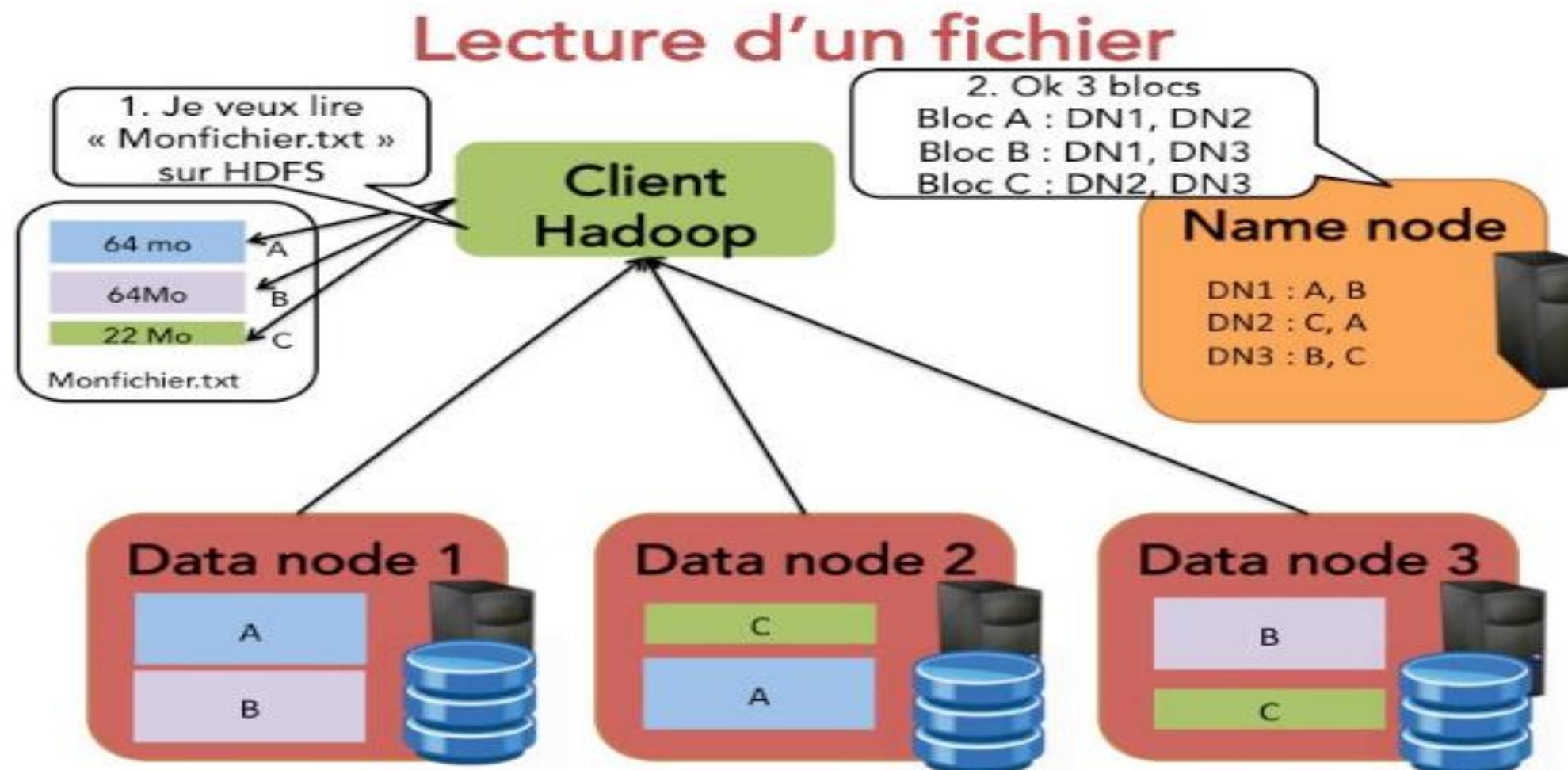
1. Le client indique au name node qu'il souhaite écrire un bloc.
2. Le name node indique le data node à contacter.
3. Le client envoie le bloc au data node.
4. Les data nodes répliquent les blocs entre eux.
5. Le cycle se répète pour le bloc suivant.

## Ecriture d'un fichier



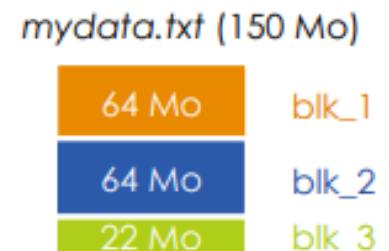
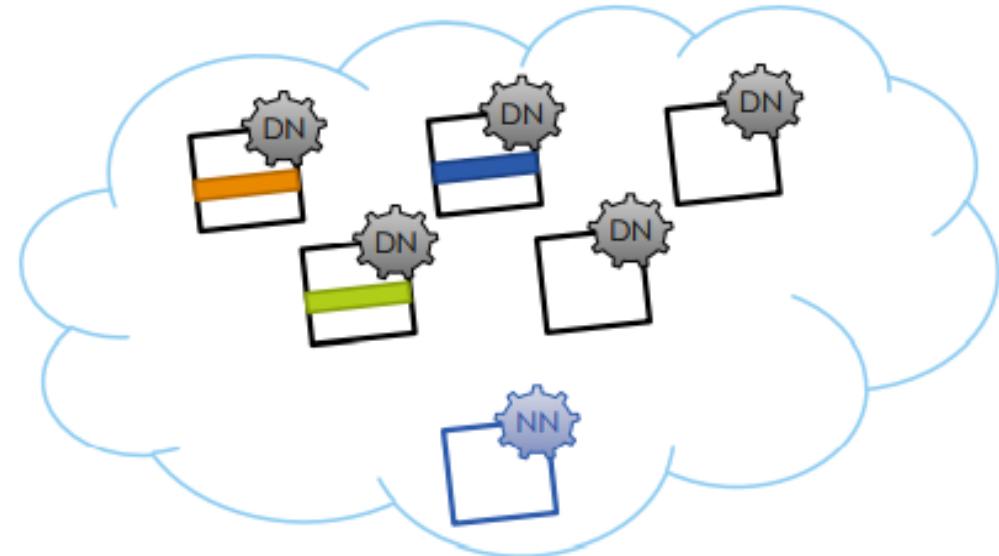
Si on souhaite lire un fichier dans HDFS, c'est également assez simple.

1. Le client indique au name node qu'il souhaite lire un fichier.
2. Le name node indique sa taille ainsi que les différents data nodes contenant les blocs.
3. Le client récupère chacun des blocs sur l'un des data nodes.
4. Si le data node est indisponible, le client en contacte un autre.



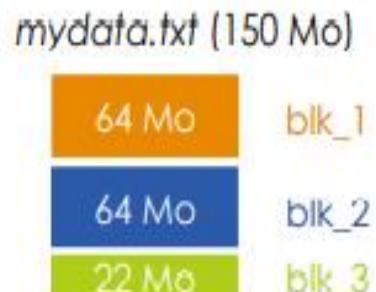
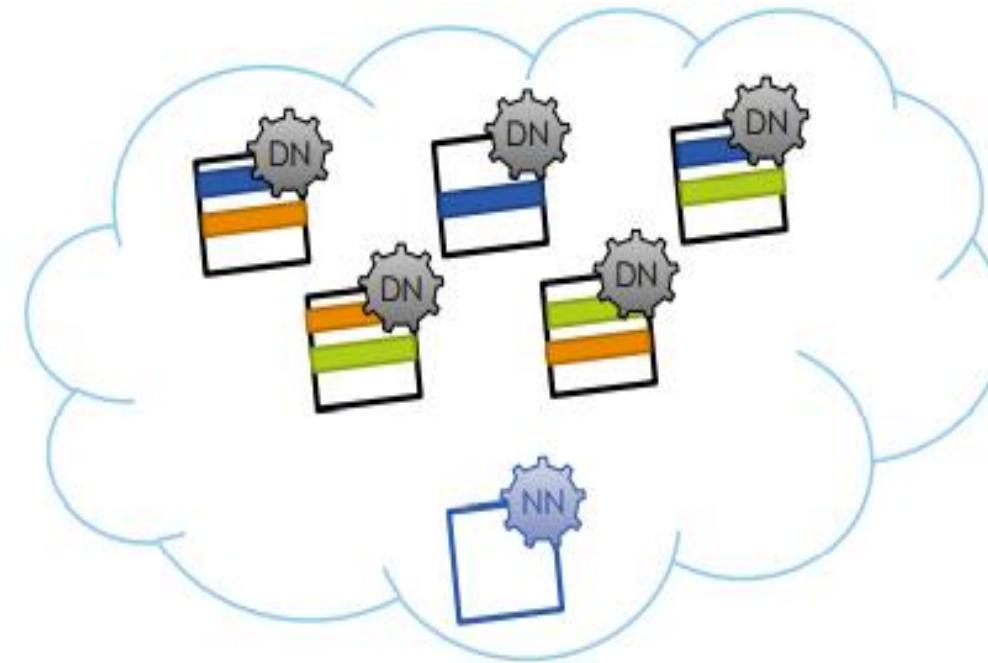
# HDFS : Hadoop Distributed File System

- Quels sont les problèmes possibles?
  - Panne de réseau ?
  - Panne de disque sur les DataNodes ?
  - Pas tous les DN sont utilisés ?
  - Les tailles des blocks sont différentes ?
  - Panne de disque sur les NameNodes ?



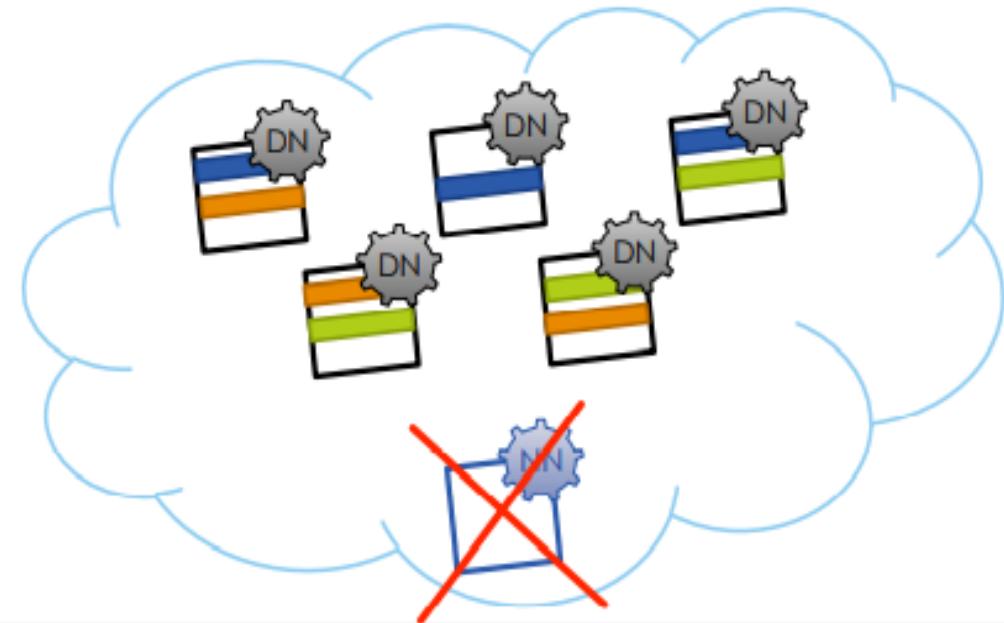
# HDFS : Hadoop Distributed File System

- Si l'un des nœuds a un problème, les données seront perdues
  - Hadoop réplique chaque bloc 3 fois
  - Il choisit 3 nœuds au hasard, et place une copie du bloc dans chacun d'eux
  - Si le nœud est en panne, le NN le détecte, et s'occupe de répliquer encore les blocs qui y étaient hébergés pour avoir toujours 3 copies stockées
  - Concept de **Rack Awareness** (rack = baie de stockage)
- Si le NameNode a un problème ?



# HDFS : Hadoop Distributed File System

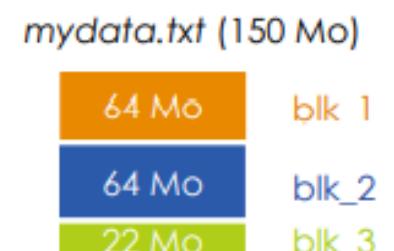
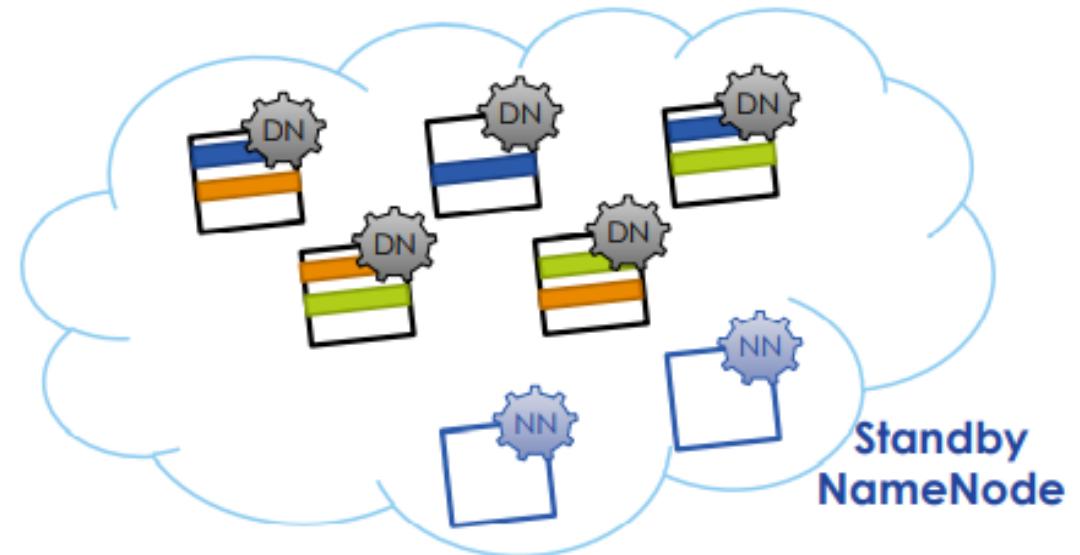
- Si le NameNode a un problème :
  - Données inaccessibles?
  - Données perdues à jamais
  - Pas de problème
- Si c'est un problème d'accès (réseau), les données sont temporairement inaccessibles
- Si le disque du NN est défaillant, les données seront perdues à jamais



mydata.txt (150 Mo)	
64 Mo	blk_1
64 Mo	blk_2
22 Mo	blk_3

# HDFS : Hadoop Distributed File System

- Pour éviter cela, le NameNode sera dupliqué, non seulement sur son propre disque, mais également quelque part sur le système de fichiers du réseau
- Définition d'un autre NN (*standby namenode*) pour reprendre le travail si le NameNode actif est défaillant



# Récapitulation

## NameNode :

- Il peut être considéré comme le maître du système.
- Le NameNode est le nœud (ou machine) principal et il ne stocke pas les données réelles. Vu que dans Hadoop les données ne sont pas au même endroit et elles sont même découpées en morceaux et dupliquées, le travail du NameNode est de savoir à tout instant où se trouvent les données. On peut dire qu'il stocke des métadonnées sur les machines (ou noeud) où se trouvent véritablement les données.

- Il enregistre chaque modification apportée aux métadonnées du système de fichiers
- Si un fichier est supprimé dans HDFS, le NameNode l'enregistrera immédiatement dans EditLog
- Il reçoit régulièrement un Heartbeat et un rapport de blocage de tous les DataNodes du cluster pour garantir que les DataNodes sont actifs
- Il conserve un enregistrement de tous les blocs dans le HDFS et le DataNode dans lesquels ils sont stockés
- Il dispose de fonctionnalités de haute disponibilité,

# Récapitulation

## DataNodes :

- Les DataNodes sont les nœuds esclaves dans HDFS.
- Les données réelles sont stockées sur DataNodes.
- Un fichier est divisé en un ou plusieurs blocs qui sont stockés dans un ensemble de DataNodes.
- Les DataNodes sont chargés de répondre aux demandes de lecture et d'écriture des clients du système de fichiers.
- Les DataNodes effectuent également la création, la suppression et la réPLICATION de blocs sur instruction du NameNode.

# Récapitulation

## NameNode :

- FsImage est un fichier stocké sur le système de fichiers du système d'exploitation qui contient la structure de répertoire complète (espace de noms) du HDFS avec des détails sur l'emplacement des données sur les blocs de données et quels sont les blocs stockés dans chaque nœud.
- Ce fichier est utilisé par le NameNode lors de son démarrage.

# Récapitulation

## NameNode :

- EditLogs est un journal des transactions qui enregistre les modifications apportées au système de fichiers HDFS ou toute action effectuée sur le cluster HDFS
- Ajout d'un nouveau bloc,
- RéPLICATION,
- Suppression,

En bref, il enregistre les modifications depuis la création de la dernière FsImage.

# Récapitulation

## NameNode :

- Chaque fois que le NameNode redémarre, les EditLogs sont appliqués à FsImage pour obtenir le dernier instantané du système de fichiers.
- les redémarrages de NameNode sont rares dans les clusters de production

# Récapitulation

Pour cette raison, vous pouvez rencontrer les problèmes suivants :

- EditLog devient de plus en plus volumineux, en particulier lorsque le NameNode s'exécute pendant une longue période sans redémarrage ;
- Le redémarrage de NameNode prend plus de temps, car trop de modifications doivent maintenant être fusionnées ;
- Si le NameNode ne redémarre pas (c'est-à-dire qu'il plante), il y aura une perte de données importantes, car le FsImage utilisé au moment du redémarrage est très ancien

# Récapitulation

## NameNode secondaire (2NN) :

- Le Namenode secondaire est un nœud d'assistance pour NameNode et le but précis du 2NN est d'avoir un point de contrôle (checkpoint) dans HDFS, ce qui aide le NameNode à fonctionner efficacement.
  1. Il obtient les EditLogs du NameNode à intervalles réguliers et s'applique à la FSImage.
  2. Une fois qu'il a une nouvelle FSImage, il la recopie dans le NameNode.
  3. Le NameNode utilisera cette FSImage pour le prochain redémarrage, ce qui réduira le temps de démarrage.

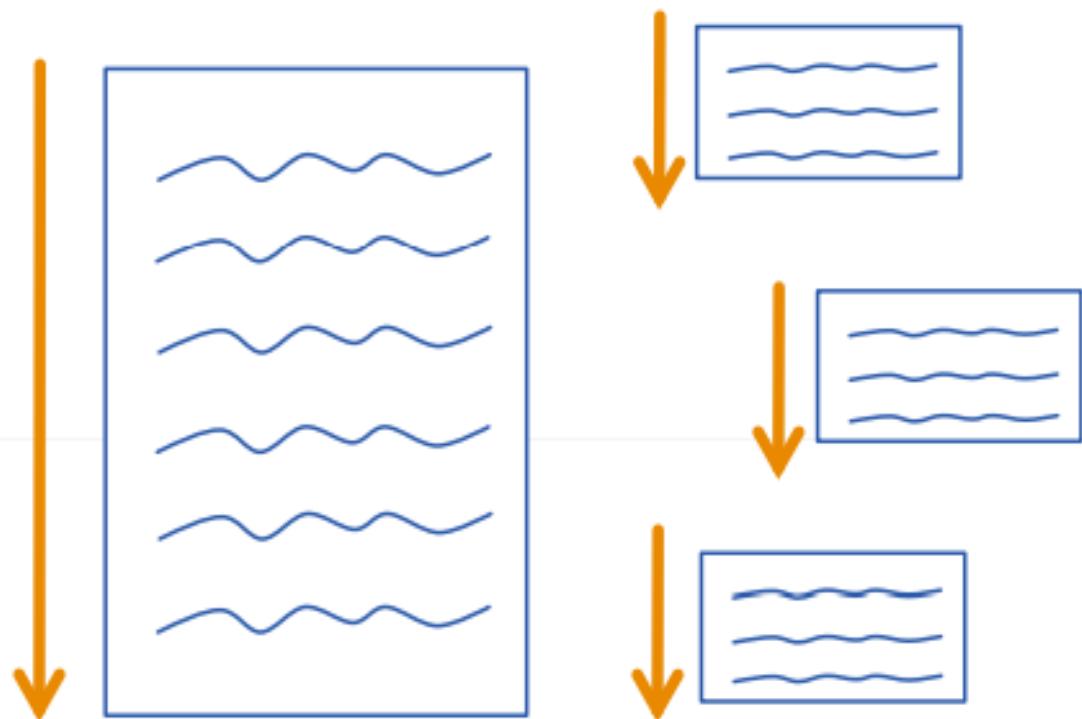
# Récapitulation

NameNode secondaire (2NN) :

- Cependant, ce processus n'est pas sans problèmes. Des retards dans le processus mentionné peuvent entraîner le démarrage d'un NameNode
- Des retards peuvent survenir si :
  - Le 2NN prend trop de temps pour télécharger les EditLogs à partir du NameNode;
  - Le NameNode est lent à télécharger les FsImages sur le 2NN et/ou à télécharger les FsImages mises à jour à partir du NameNode secondaire.

# Map-Reduce

- Patron d'architecture de développement permettant de traiter des données volumineuses de manière parallèle et distribuée
- A la base, le langage Java est utilisé, mais grâce à une caractéristique de Hadoop appelée *Hadoop Streaming*, il est possible d'utiliser d'autres langages comme Python ou Ruby
- Au lieu de parcourir le fichier séquentiellement (bcp de temps), il est divisé en morceaux qui sont parcourus en parallèle.



# Map-Reduce : Exemple

- Imaginons que vous ayez plusieurs magasins que vous gérez à travers le monde
- Un très grand livre de comptes contenant TOUTES les ventes
- **Objectif** : Calculer le total des ventes par magasin pour l'année en cours
- Supposons que les lignes du livres aient la forme suivante:
  - Jour      Ville      produit      Prix



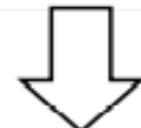
2012-01-01	London	Clothes	25.99
2012-01-01	Miami	Music	12.15
2012-01-02	NYC	Toys	3.10
2012-01-02	Miami	Clothes	50.00

# Map-Reduce : Exemple

- Si on utilise les hashtable sur 1To, Problèmes ?
  - Ça ne marchera pas ?
  - Problème de mémoire ?
  - Temps de traitement long ?
  - Réponses erronées ?
- Le traitement séquentiel de toutes les données peut s'avérer très long
- Plus on a de magasins, plus l'ajout des valeurs à la table est long
- Il est possible de tomber à court de mémoire pour enregistrer cette table
- Mais cela peut marcher, et le résultat sera correct



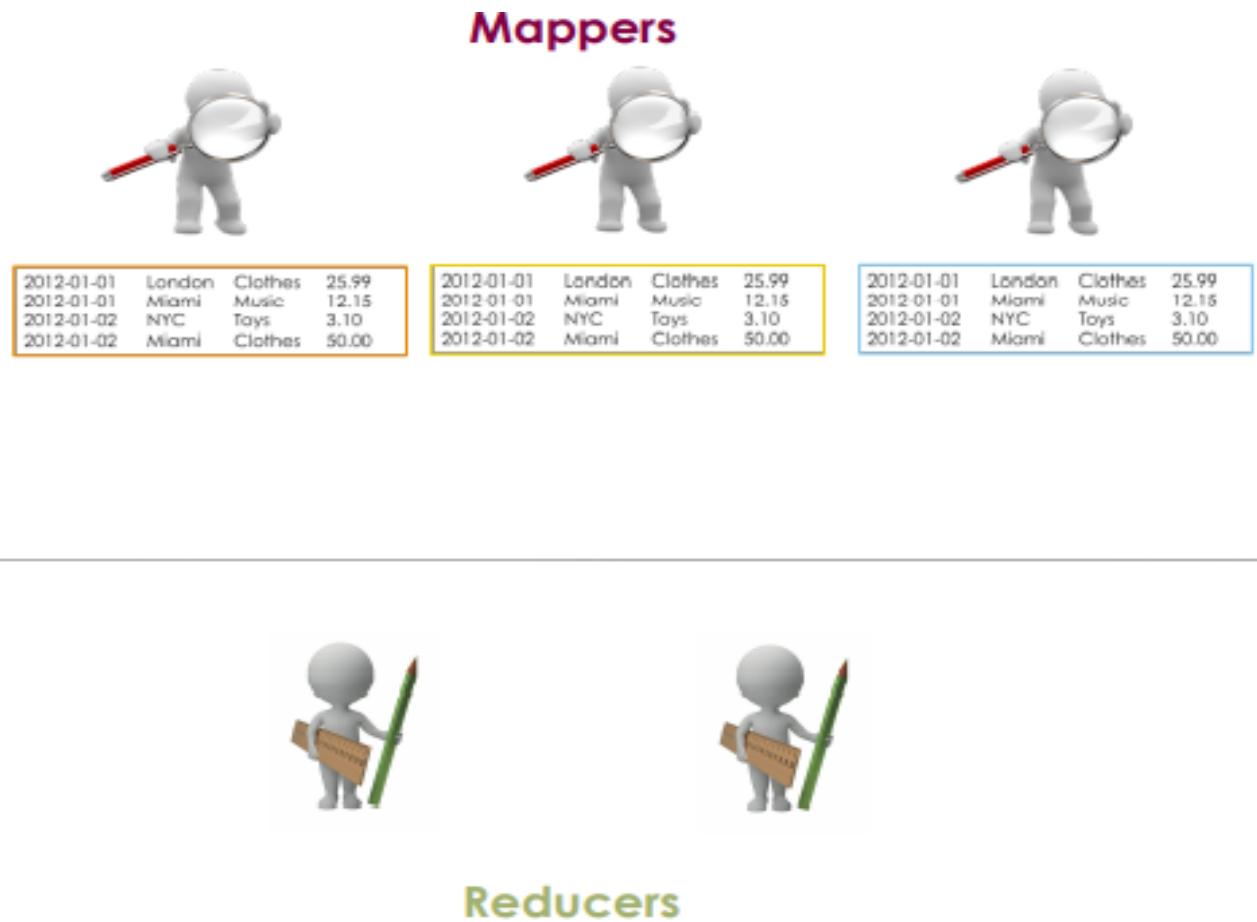
2012-01-01	London	Clothes	25.99
2012-01-01	Miami	Music	12.15
2012-01-02	NYC	Toys	3.10
2012-01-02	Miami	Clothes	50.00



Clef	Valeur
London	25.99
Miami	62.15
NYC	3.10

# Map-Reduce : Exemple

- **Map-Reduce** : Moyen plus efficace et rapide de traiter ces données
- Au lieu d'avoir une seule personne qui parcourt le livre, si on en recrutait plusieurs?
- Appeler un premier groupe les *Mappers* et un autre les *Reducers*
- Diviser le livre en plusieurs parties, et en donner une à chaque Mapper
  - Les Mappers peuvent travailler en même temps, chacun sur une partie des données



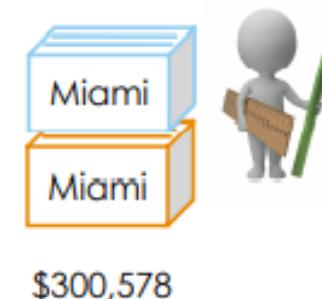
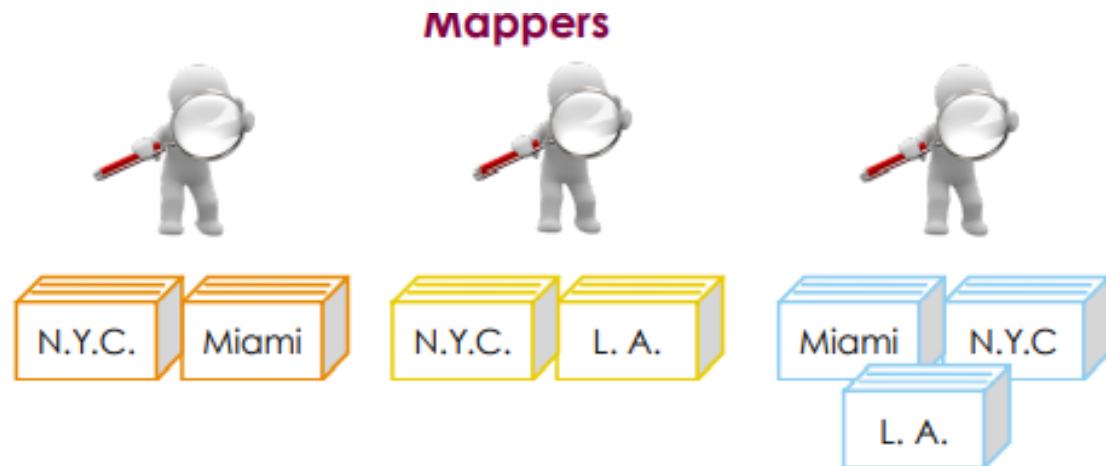
# Map-Reduce : Exemple

## Mappers :

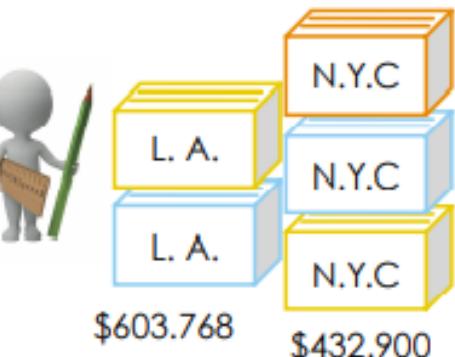
- Pour chaque entrée, saisir la ville, et le total des ventes et les enregistrer dans une fiche
- Rassembler les fiches du même magasin dans une même pile

## Reducers :

- Chaque Reducer sera responsable d'un ensemble de magasins
- Ils collectent les fiches qui leur sont associées des différents Mappers
- Ils regroupent les petites piles d'une même ville en une seule
- Ils parcourent ensuite chaque pile par ordre alphabétique des villes (L.A avant Miami), et font la somme de l'ensemble des enregistrements



Reducers



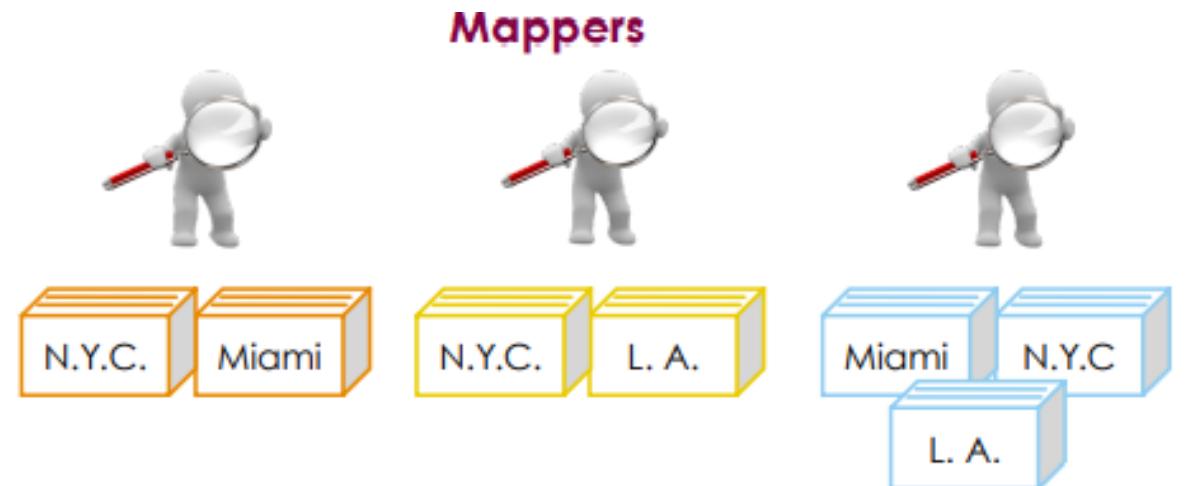
# Map-Reduce : Exemple

- Le Reducer reçoit des données comme suit :

Miami	12.34
Miami	99.07
Miami	3.14
NYC	99.77
NYC	88.99

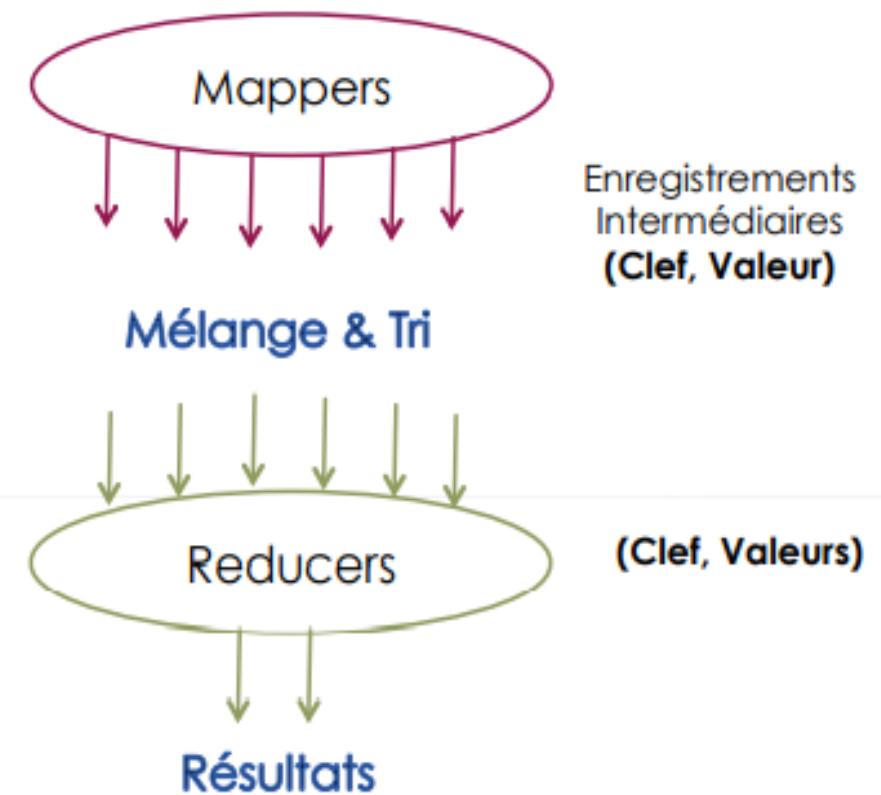
- Pour chaque entrée, de quoi avons-nous besoin pour calculer la totalité des ventes pour chaque magasin?

- Coût précédent
- Coût en cours
- Ventes totales par magasin
- Magasin précédent
- Magasin en cours



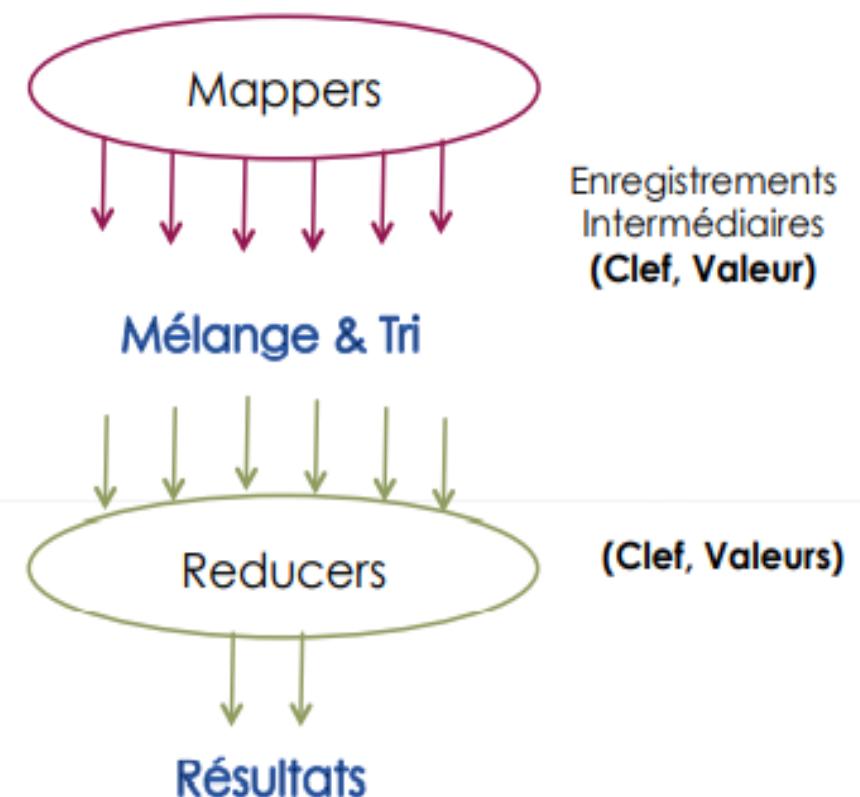
# Map-Reduce : Fonctionnement

- Les Mappers sont de petits programmes qui commencent par traiter chacun une petite partie des données
- Ils fonctionnent en parallèle
- Leurs sorties représentent les *enregistrements intermédiaires*: sous forme d'un couple (*clef, valeur*)
- Une étape de *Mélange et Tri* s'ensuit
  - *Mélange* : Sélection des piles de fiches à partir des Mappers
  - *Tri* : Rangement des piles par ordre au niveau de chaque Reducer
- Chaque Reducer traite un ensemble d'enregistrements à la fois, pour générer les résultats finaux

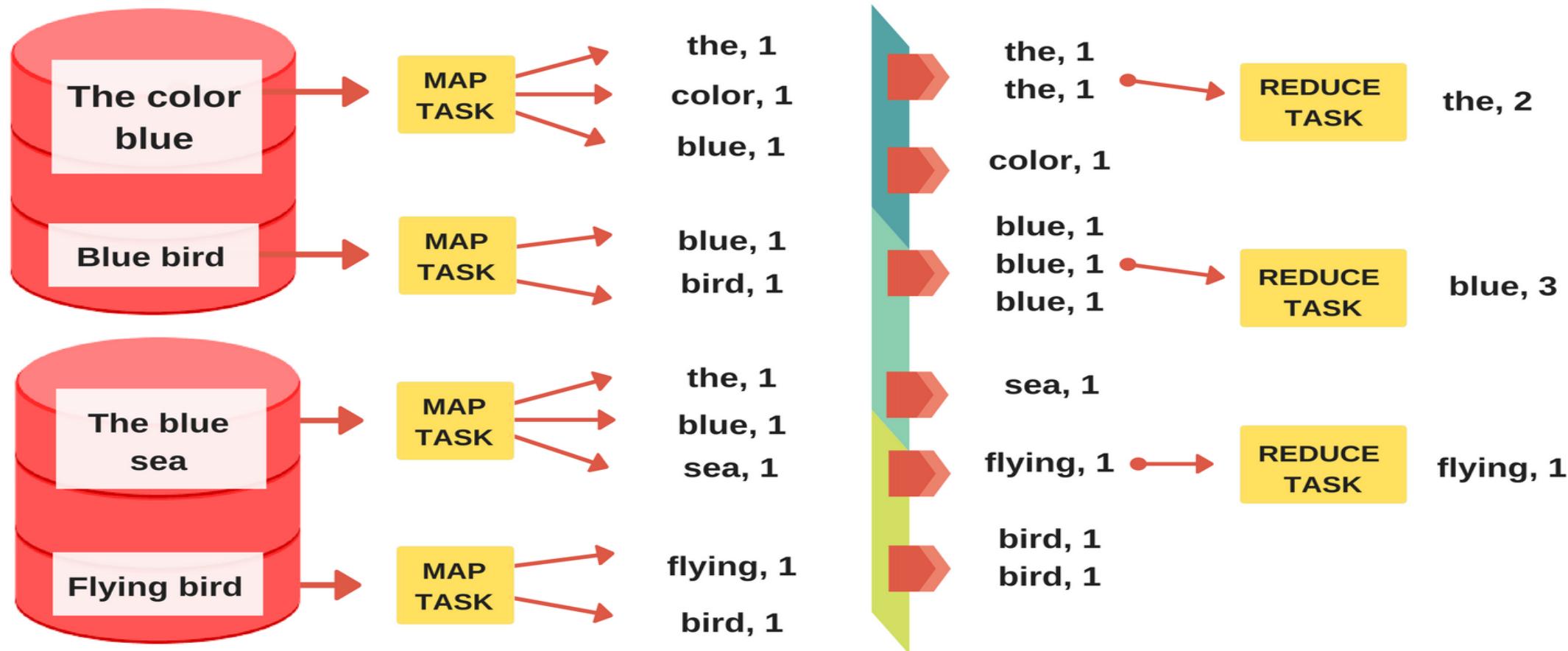


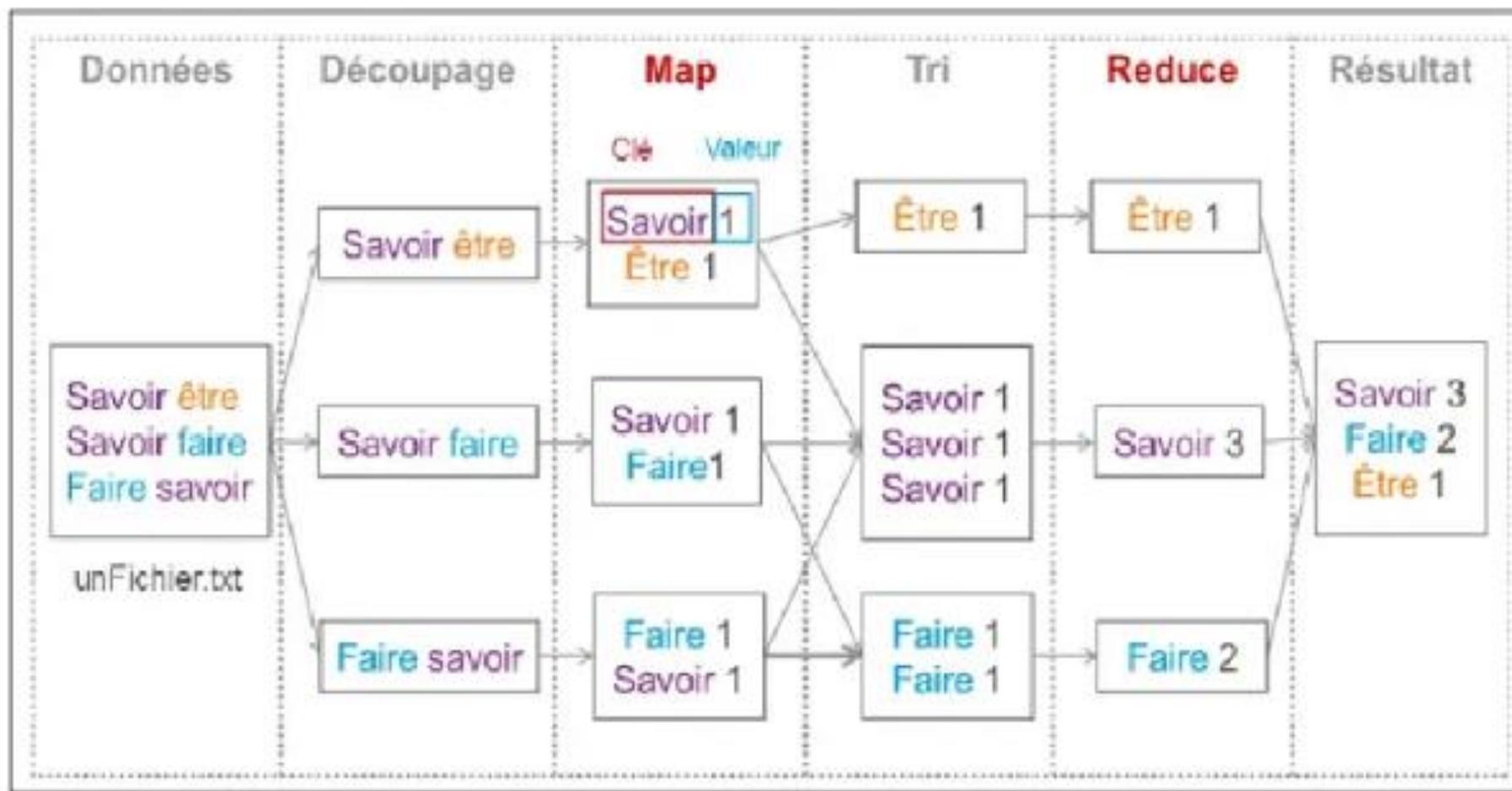
# Map-Reduce : Fonctionnement

- Pour avoir un résultat trié par ordre, on doit:
  - Soit avoir un seul Reducer, mais ça ne se met pas bien à l'échelle
  - Soit ajouter une autre étape permettant de faire le tri final
- Si on a plusieurs Reducers, on ne peut pas savoir lesquels traitent quelles clefs: le partitionnement est aléatoire.



# SORT and SHUFFLE





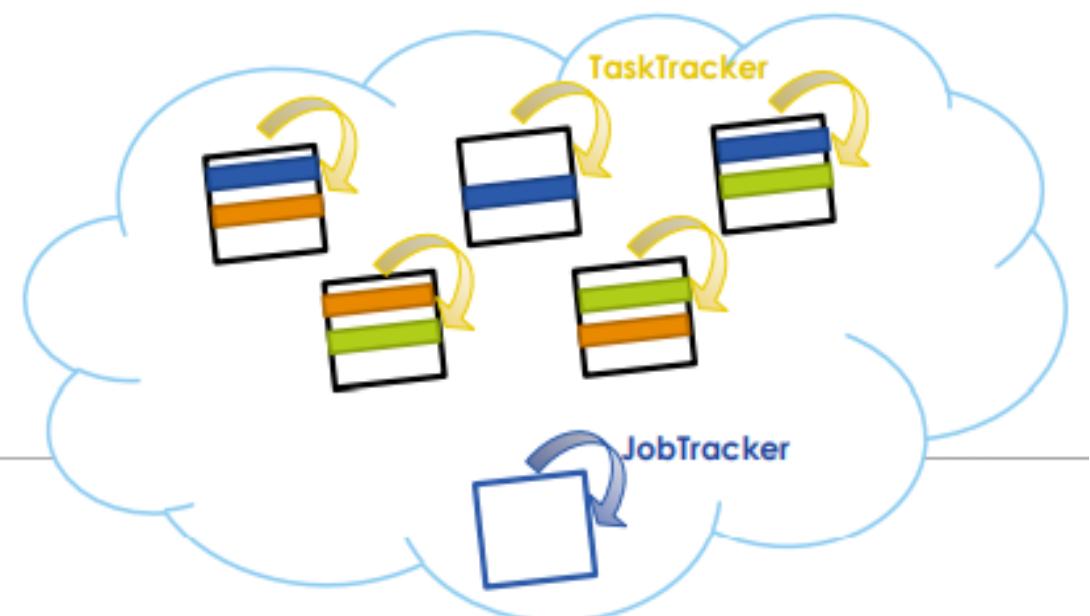
# Démons de MapReduce

## ➤ JobTracker

- Divise le travail sur les Mappers et Reducers, s'exécutant sur les différents nœuds

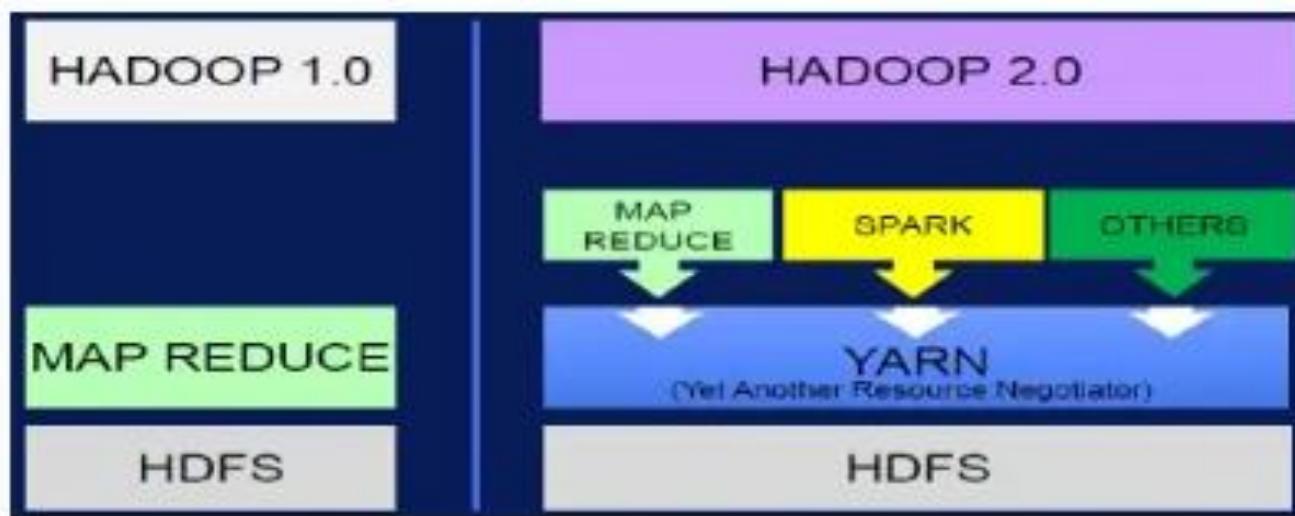
## ➤ TaskTracker

- S'exécute sur chacun des nœuds pour exécuter les vraies tâches de MapReduce
- Choisit en général de traiter (Map ou Reduce) un bloc sur la même machine que lui
- S'il est déjà occupé, la tâche revient à un autre tracker, qui utilisera le réseau (rare)



# Hadoop v2

Utilisation plus élevée des ressources → Coût inférieur



La description de Hadoop comme possédant 2 couches (MapReduce et HDFS) est correcte pour la version 1 de Hadoop.

Depuis la version 2, Hadoop a adopté une troisième couche : YARN ("Yet Another Resource Negotiator"), un outil de gestion des ressources distribuée.(puissance de calcul et et la répartition de la charge entre les machines)



### Yet Another Resource Negotiator

- La technologie est devenue un sous-projet Apache Hadoop au sein de l'Apache Software Foundation (ASF) en 2012 et était l'une des principales fonctionnalités ajoutées dans Hadoop 2.0, qui a été publiée pour test cette année-là et est devenue disponible en Octobre 2013.

## YARN



- YARN est un gestionnaire de ressources au sein d'un cluster Hadoop. Il a pour but de planifier et allouer les ressources au sein des clusters Hadoop.
- Un peu comme HDFS qui se charge de la gestion du stockage, YARN se charge de la puissance de calcul.
- YARN est responsable de :
  - L'allocation des ressources système aux différentes applications exécutées dans un cluster Hadoop,
  - La planification des tâches à exécuter sur différents nœuds de cluster.

## YARN



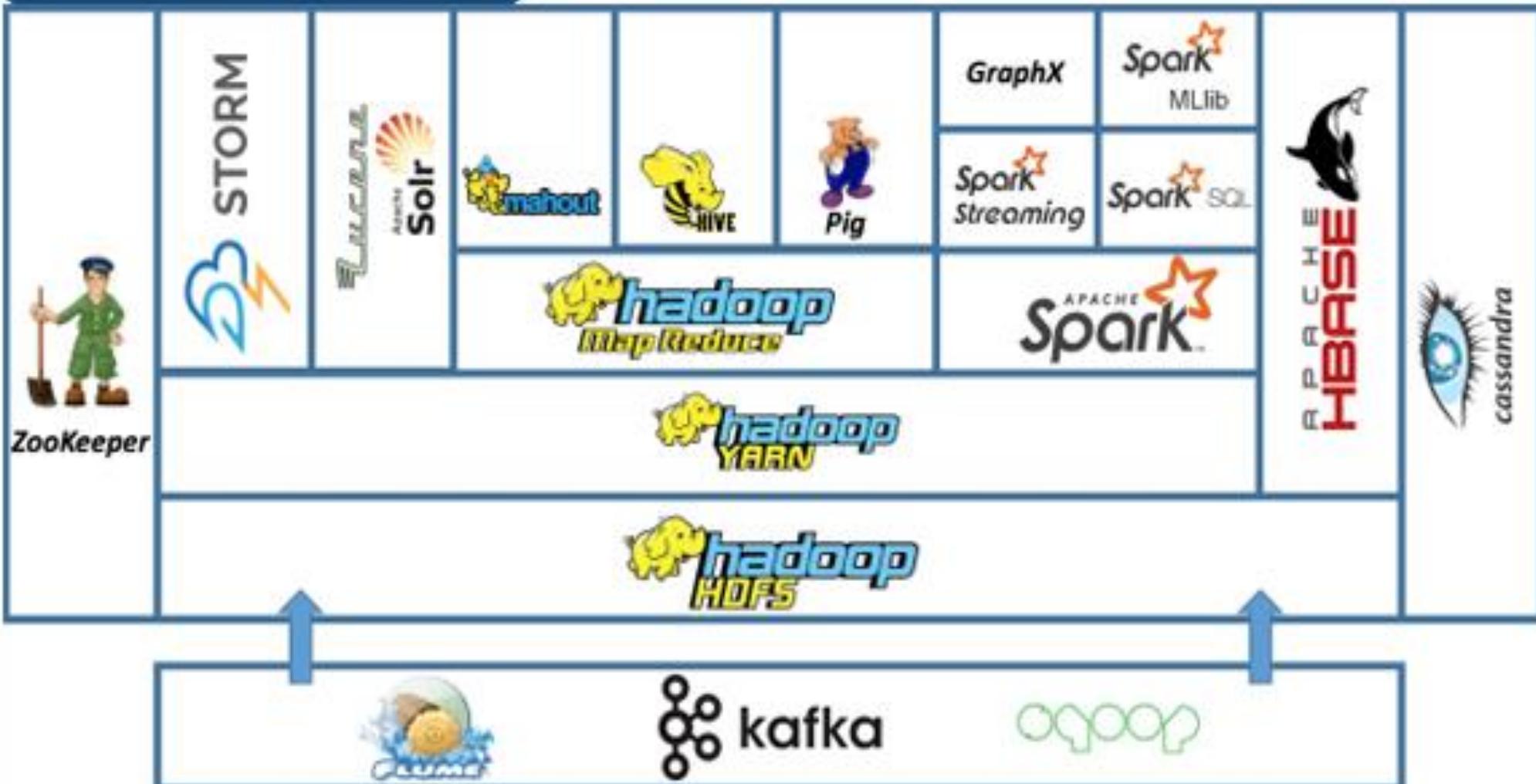
- Le gestionnaire de ressources global (**ResourceManager**) a pour rôle d'accepter les tâches soumises par les utilisateurs, de programmer les tâches et de leur allouer des ressources.
- Sur chaque noeud, on retrouve un **NodeManager** dont le rôle de surveiller et de rapporter au ResourceManager. On retrouve par ailleurs un **ApplicationMaster**, créé pour chaque application, chargé de négocier les ressources et de travailler conjointement avec le NodeManager pour exécuter et surveiller les tâches.

## YARN



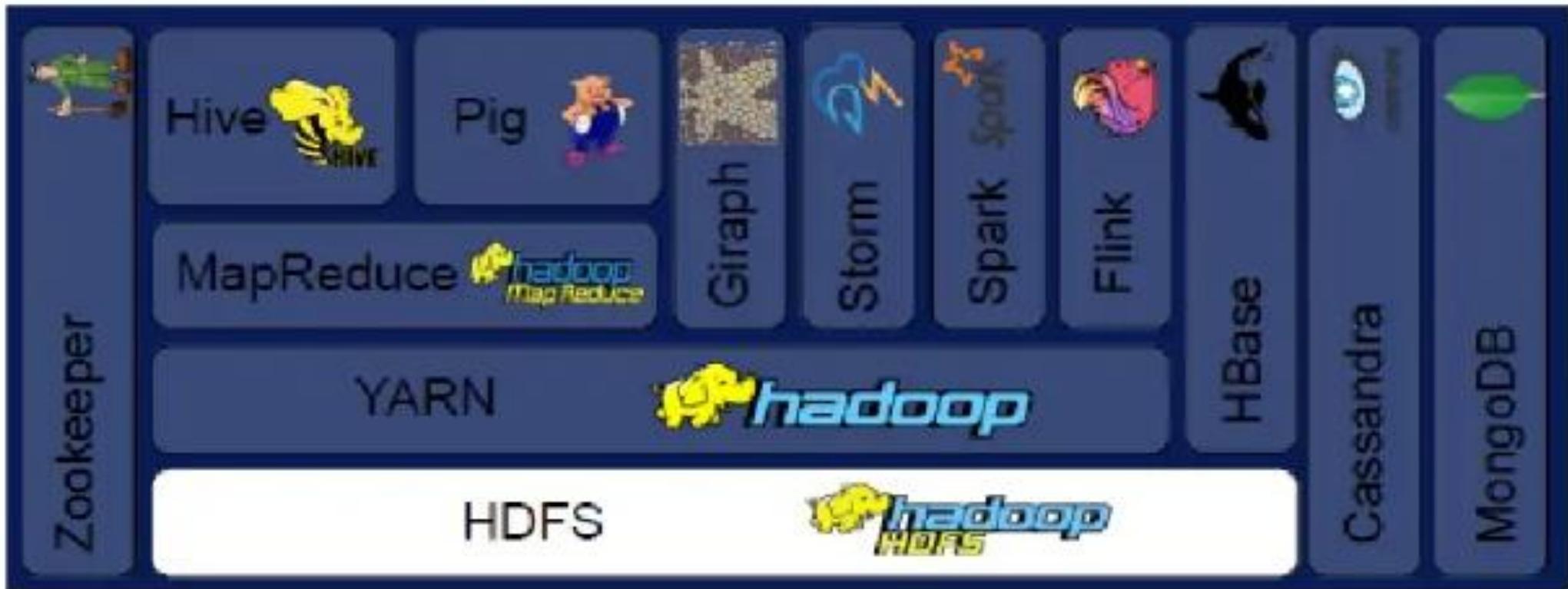
- Enfin, les **containers** de ressources sont contrôlés par les NodeManagers et assignent les ressources allouées aux applications individuelles.
- Généralement, les containers YARN sont organisés en nœuds et programmés pour exécuter des tâches uniquement si des ressources sont disponibles pour se faire.
- Sous Hadoop 3.0, il est toutefois possible de créer des « containers opportunistes » pouvant être placés en attente jusqu'à ce que des ressources soient libérées. Ce concept permet d'optimiser l'usage des ressources.

## Écosystème Hadoop

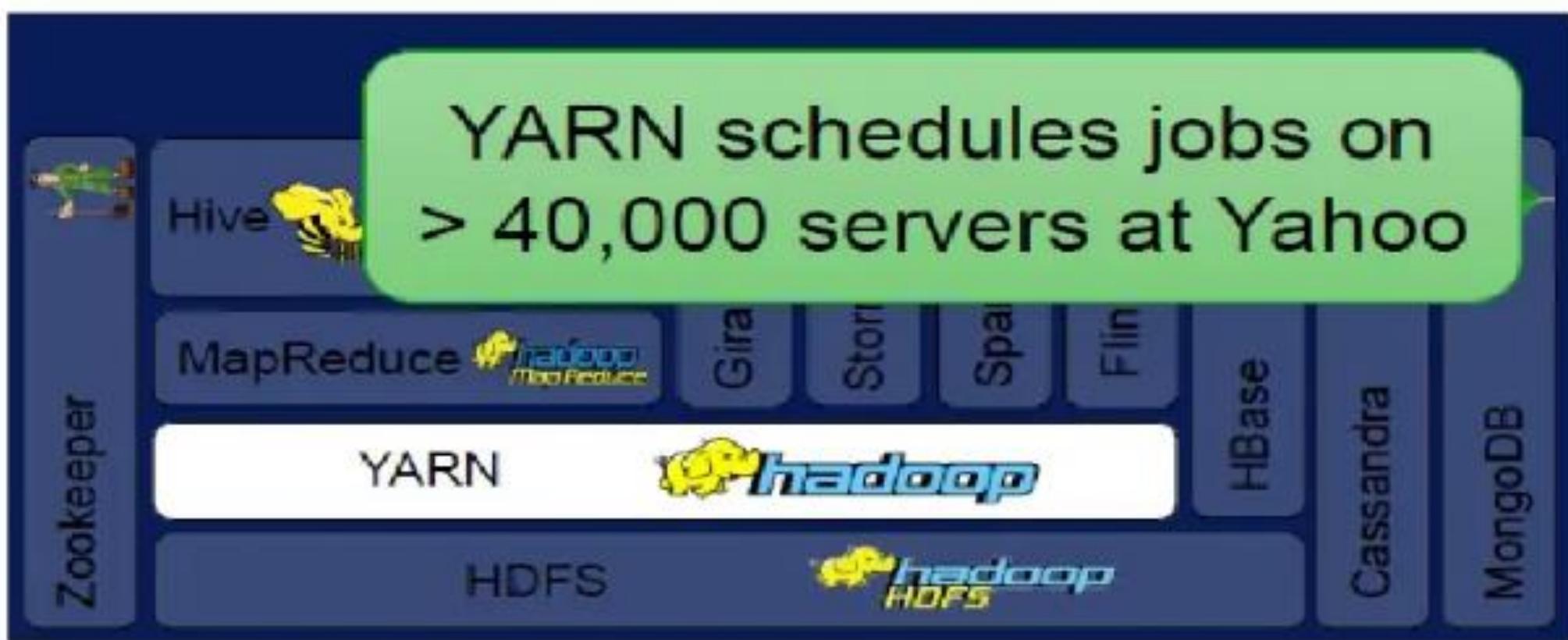


# HDFS

- ❑ Stockage évolutif
- ❑ Tolérance aux pannes



- ❑ Planification flexible
- ❑ gestion des ressources
- ❑ YARN planifie des travaux sur plus de 40 000 serveurs chez Yahoo



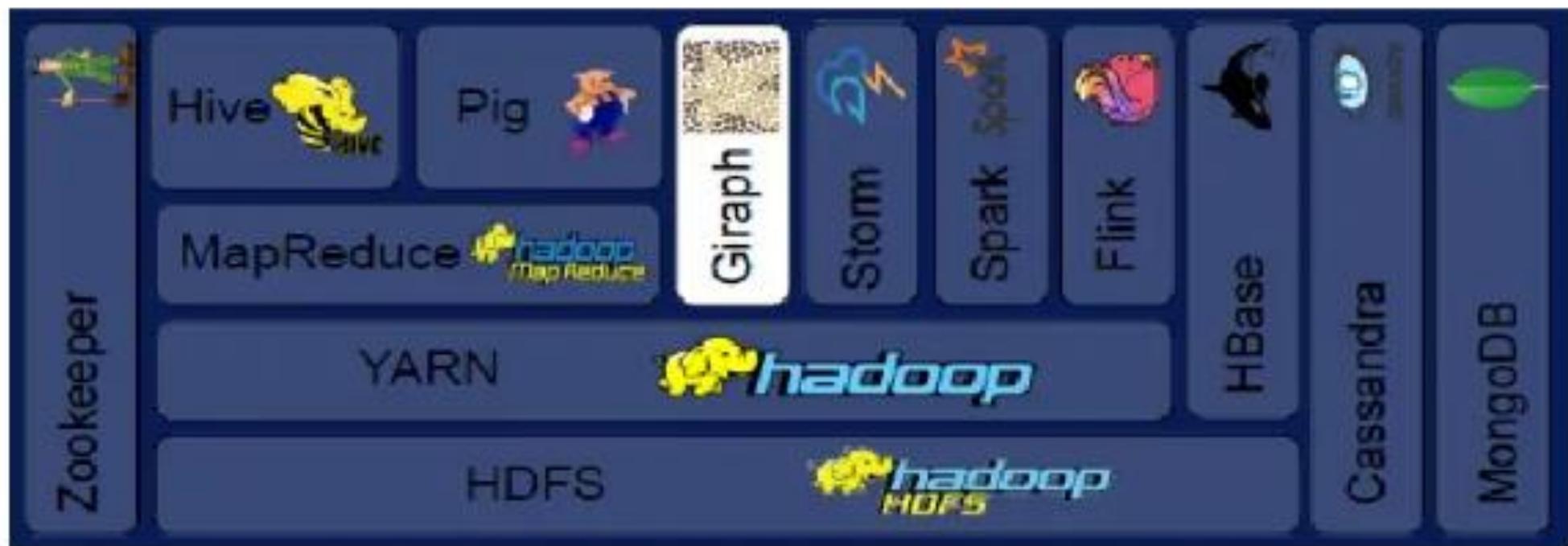
- Modèle de programmation simplifié
- Map → appliquer ()
- Reduce → résumer ()



- Modèles de programmation de niveau supérieur
- Pig = script de flux de données pour la création de programme MapReduce
- Hive = Traduit les requêtes SQL en MapReduce



- ❑ Modèles spécialisés pour le traitement de graphes
- ❑ Giraph utilisé par Facebook pour analyser les graphiques sociaux



- ❑ Traitement en temps réel et en mémoire
- ❑ En mémoire -> 100 fois plus rapide pour certaines tâches



- ❑ BDs No SQL pour les données non structurées



- ❑ Utilisé pour la gestion
- ❑ Synchronisation
- ❑ La haute disponibilité
- ❑ Configuration
- ❑ Il garde essentiellement une trace des informations qui doivent être synchronisées dans le cluster



## Écosystème Hadoop

- En plus des briques de base HDFS, YARN et MapReduce, plusieurs outils existent pour permettre :
  - L'extraction et le stockage des données de/sur HDFS
  - La simplification de traitement sur ces données
  - La gestion et coordination de la plateforme
  - La monitoring du cluster

## Écosystème Hadoop

L'écosystème Hadoop est composé des éléments suivants :

- HDFS : le système de fichier distribué d'Apache Hadoop
- YARN : le gestionnaire de ressources et des tâches dans un cluster Hadoop
- MapReduce : un framework pour le traitement de grands ensembles de données en parallèle
- Spark : un moteur d'analyse unifié pour le traitement de données à grande échelle
- PIG, HIVE : Services de traitement de données à l'aide de requêtes du type SQL
- HBase : système de gestion de base de données non-relationnelles distribué

## Écosystème Hadoop

L'écosystème Hadoop est composé des éléments suivants :

- Mahout, Spark MLlib : des services pour faire du Machine Learning
- Apache Drill : Moteur de requête SQL sur Hadoop
- Zookeeper : Pour la gestion de Cluster
- Oozie : Système de planification de workflow pour gérer les jobs Hadoop
- Sqoop : Lecture et écriture des données à partir de bases de données externes
- Flume : collecte de logs et stockage dans HDFS
- Solr & Lucene : Pour la recherche et l'indexation
- Ambari : Mise à disposition, monitoring et maintenance de cluster

### Avantages de Hadoop

- La gestion des défaillances : que ce soit au niveau du stockage ou traitement, les nœuds responsables de ces opérations sont automatiquement gérés en cas de défaillances.  
→ Forte tolérance aux pannes.
- La complexité réduite : capacité d'analyse et de traitement des données à grande échelle.
- Le coût réduit : Hadoop est open source, et malgré leur massivité et complexité, les données sont traitées efficacement et à très faible coût.

### Inconvénients de Hadoop

- Difficulté d'intégration avec d'autres systèmes informatiques : Le transfert de données d'une structure Hadoop vers des bases de données traditionnelles est loin d'être trivial.
- Administration complexe : Hadoop utilise son propre langage. L'entreprise doit donc développer une expertise spécifique Hadoop.
- Traitements de données différés et temps de latence important : Hadoop n'est pas fait pour l'analyse temps réel des données.