La Rochelle Université

D'ici, on voit + loin !

Mise en œuvre des systèmes big data

2024 - 2025

Ahmed Hamdi

La Rochelle Université

SOMMAIRE

- Introduction au Big Data
- L'écosystème Hadoop
  - HDFS
  - Map-reduce
- Traitement de données
  - Batch Processing
  - Streaming Processing
- Bases de données NoSQL
  - Cassandra
  - MongoDB

La Rochelle Université

Introduction aux Big Data

Big Data

Faits et intérêts

Le **Big Data** englobe de grand volumes de données structurées et non structurées au-delà des capacités de traitement traditionnelles

+ Volume de données quotidien

On estime que le **volume** de données générées chaque jour dans le monde atteindra **463 exaoctets** (463Md Go) en 2024.

+ Données non structurées

En 2025, la proportion de données **non structurées** est estimée à **80%.**

+ Multiplicités de sources

Assistants vocaux, capteurs, médias sociaux, journaux d'exécution…

Source : https://www.sales-hacking.com/fr/post/statistiques-big-data

Big Data

Faits et intérêts

Le **Big Data** englobe de grand volumes de données structurées et non structurées au-delà des capacités de traitement traditionnelles

+ Confiance des dirigeants

95 % des sociétés dans le monde considèrent que le principal défi est de traiter les données non structurées

+ Accès aux informations

La moitié des dirigeants considèrent que les solutions data actuelles manquent de flexibilité, ce qui peut limiter leur accès aux données

+ Nécessité d'améliorer le traitement

60 % des chefs d'entreprise estiment qu'ils doivent améliorer leur traitement des données pour prendre des décisions plus rapidement

Source : https://www.sales-hacking.com/fr/post/statistiques-big-data

Big Data

Volume, Vélocité, Variété

+ Volume

Énorme quantité de données générées chaque jour (Go, To, Po, Eo)

+ Vélocité

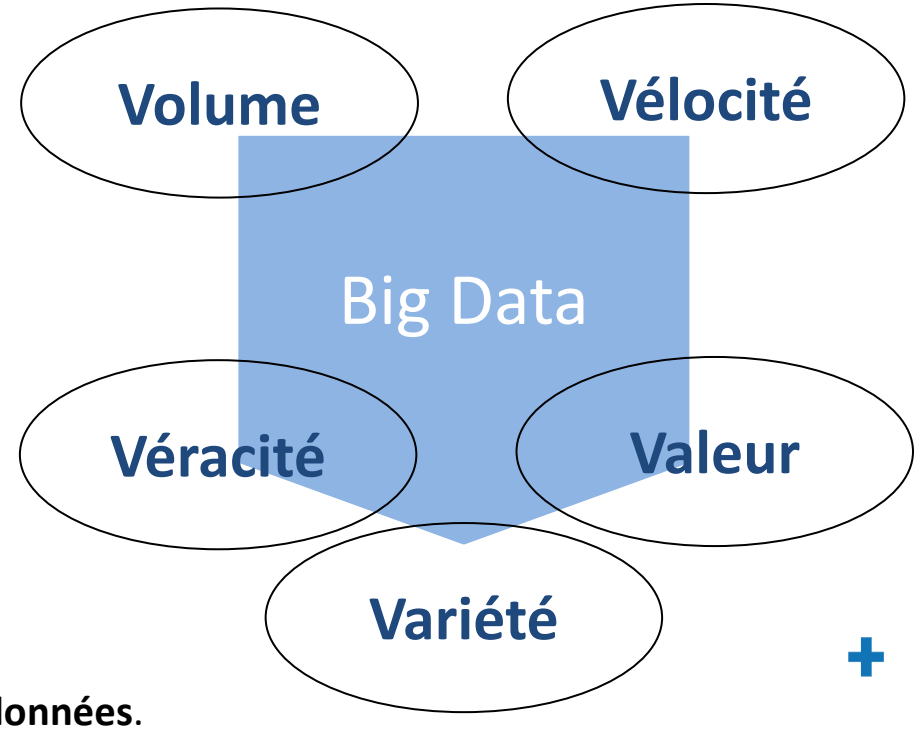
La rapidité avec laquelle les données sont créées et traitées (flux rapide)

+ Variété

Diversité des types de données (structurées, semi-structurées, non-structurées)

Big Data

Modèle 5V



+ Véracité

La **qualité** et la **fiabilité** des **données**. Les informations collectées peuvent être incomplètes, imprécises ou biaisées !

+ Valeur

L'utilité que l'on peut tirer des **données**. Une donnée brute n'a de sens que si elle peut fournir des analyses exploitables

Big Data

Encore des caractéristiques V…

- **Variabilité :** données au fil de temps
- **Visualisation :** capacité à représenter visuellement des données
- **Vulnérabilité :** sécurité liés au stockage et la gestion de données
- **Viabilité :** pertinence de données pour répondre à un besoin métier spécifique
- **Vulnérabilité :** conformité aux normes de données
- **Volatilité :** durée de vie de données

Le modèle 3V

Volume (1/3)

- Le prix de stockage de données a beaucoup diminué
  - de 100,000$ / Go (1980)
  - à 0.1$ / Go (2013)
- Lieux de stockage fiables
  - ne stocker que certaines données, jugées sensibles
  - perte de données, éventuellement, utiles

➔ Comment déterminer les données qui méritent d'être stockées

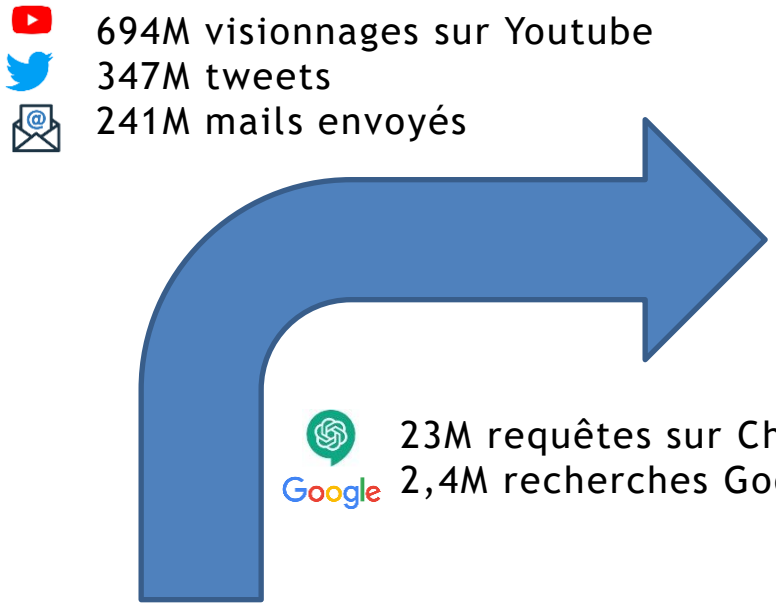
- Transactions, logs…

- Problèmes
  - Comment stocker les données dans un endroit fiable et pas cher ?
  - Comment parcourir ces données et en extraire des informations d'une manière simple et rapide?

Le modèle 3V

Vélocité (2/3)

- **Rapidité d'arrivée de données**
  - Les données doivent être stockées à l'arrivée, parfois même des Téraoctets par jour
- **Vitesse de traitement de données**
- **Exemple : pour recommander un article**
  - Il ne suffit pas de savoir quel article un client a acheté ou réservé auparavant
  - Un article consulté plus de 5 minutes, peut être recommandé par mail ou par SMS



Source : https://datareportal.com

Le modèle 3V

Variété (3/3)

- La plupart des données existantes sont semi- ou non-structurées
- Pas de format prédéfini :
  - les données sont sous plusieurs formats et types (texte, image, son…)
- Même les données obsolètes peuvent être utiles pour certaines décisions

Le modèle 5V

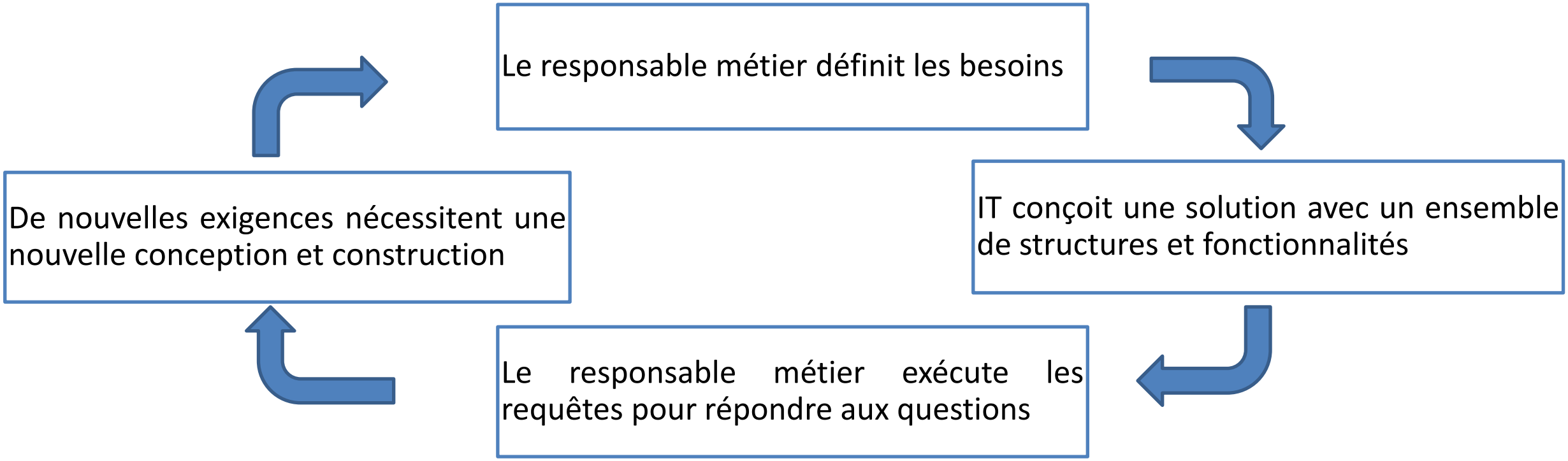
Véracité et Valeur

+ Les données proviennent de **sources multiples et hétérogènes**, ce qui peut entraîner **des problèmes de précision, de cohérence et de crédibilité**. La véracité implique de s'assurer que les données sont exactes, pertinentes et exploitables pour éviter des analyses biaisées ou erronées.
  - ➔ Gestion du bruit, des erreurs, des doublons et des données incomplètes

+ Il ne s'agit **pas seulement** de collecter des données massives, mais de **les transformer en informations exploitables** pour prendre des décisions éclairées ou créer de nouveaux services. La valeur représente l'utilité des données pour l'entreprise ou l'organisation.
  - ➔ Identifier les données qui ont un impact réel sur les objectifs stratégiques.

Big Data vs Approche classique

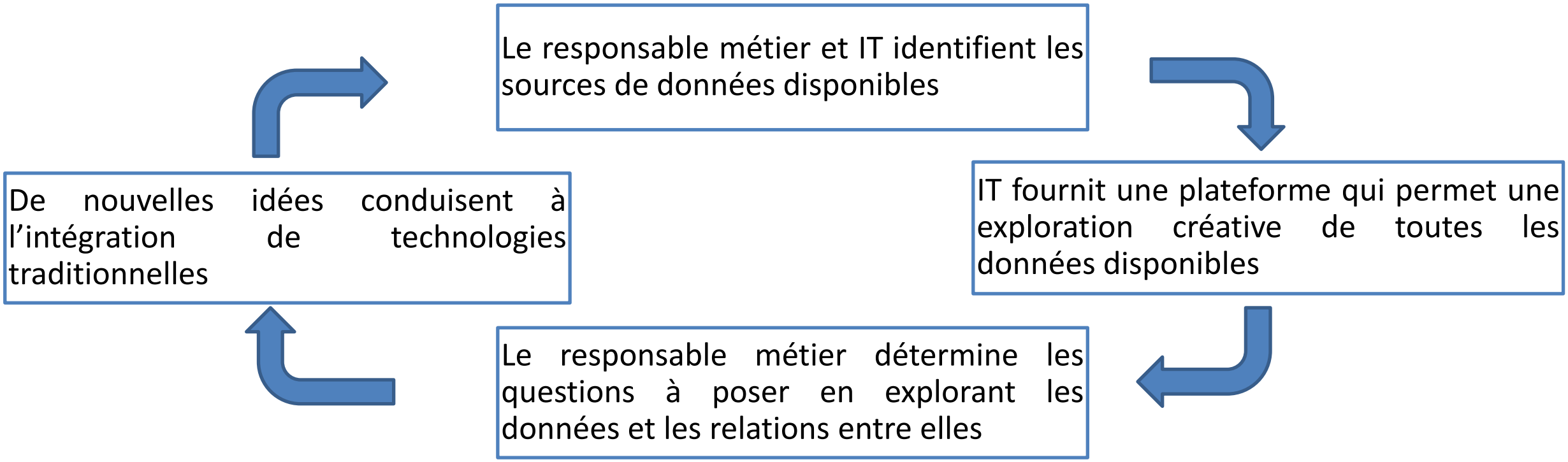
Approche classique



➔ C'est une solution appropriée pour des données structurées, les opérations et les processus répétitifs, les sources relativement stables, les besoins bien compris et bien définis

Big Data vs Approche classique

Approche big data



➔ La question n'est pas de choisir entre l'approche classique et l'approche big data **mais plutôt** comment les faire fonctionner ensemble?

La Rochelle Université

L’écosystème HADOOP

HADOOP

Présentation du framework

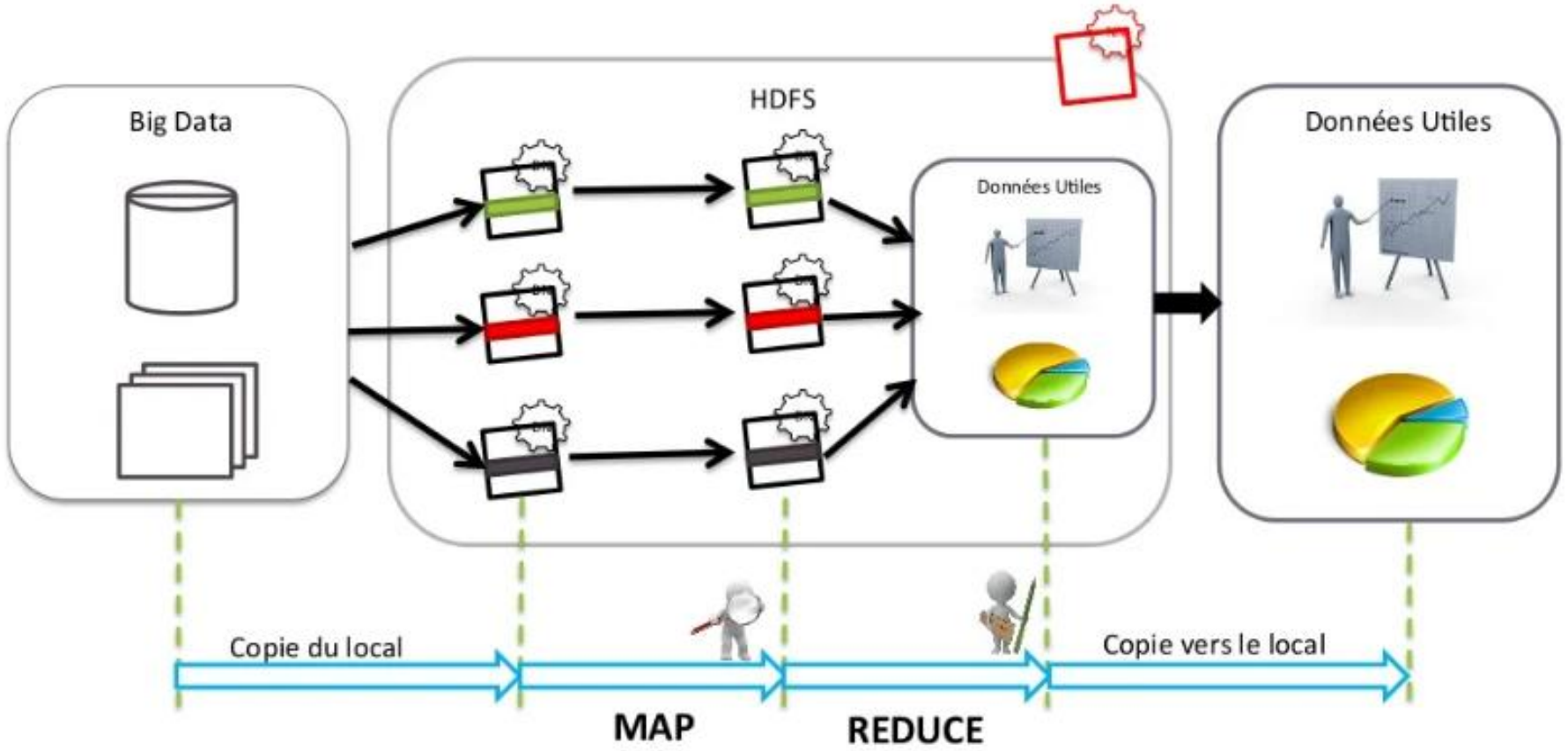
- Le système Hadoop consiste en deux grandes parties
  - Stockage des données : HDFS (Hadoop Distributed File System)
  - Traitement de données : MapReduce / YARN

- Principe :
  - Diviser les données
  - Les sauvegarder sur une collection de machines, appelées clusters
  - Traiter les donnés 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

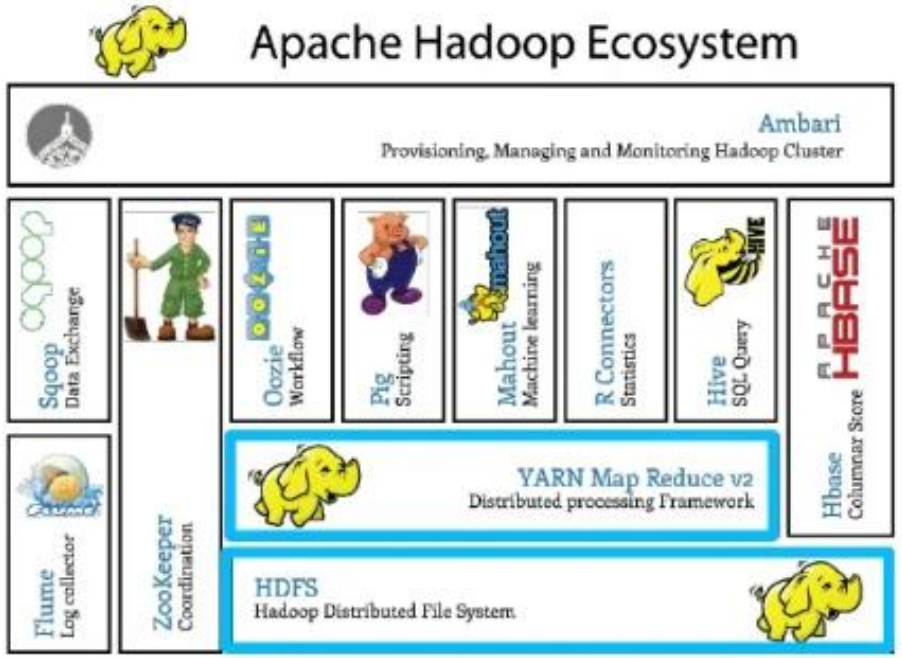
Présentation du framework



HADOOP

Écosystème

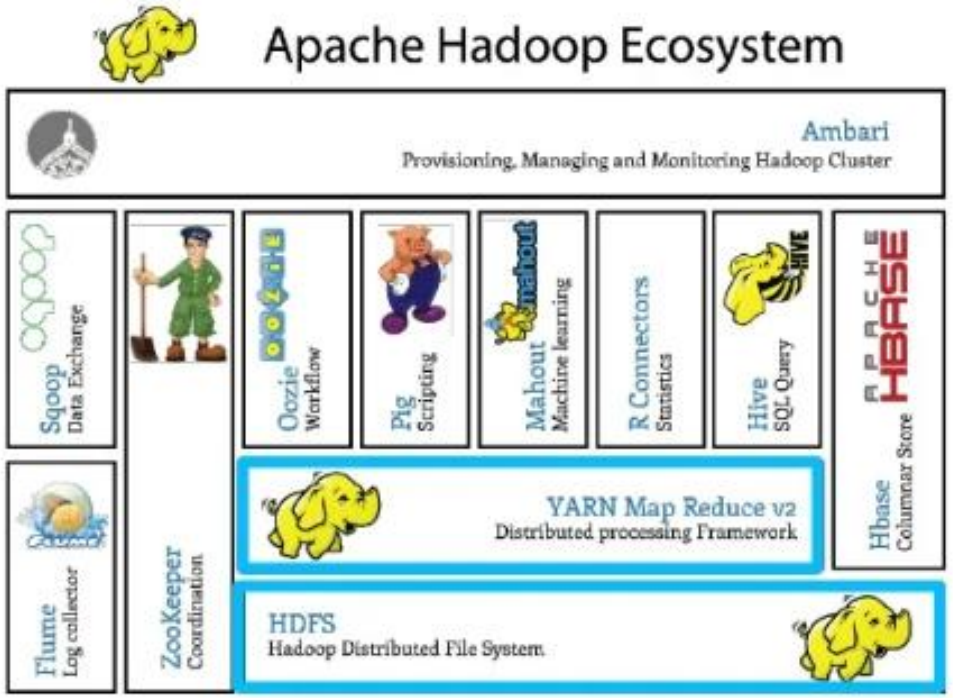
- En plus des briques de base HDFS/Map Reduce et Yarn, plusieurs outils existent pour permettre:
  - L'extraction et le stockage de données de/sur HDFS
  - La simplification des opérations de traitement sur ces données
  - La gestion et coordination de la plateforme
  - Le monitoring du cluster



HADOOP

Ecosystème

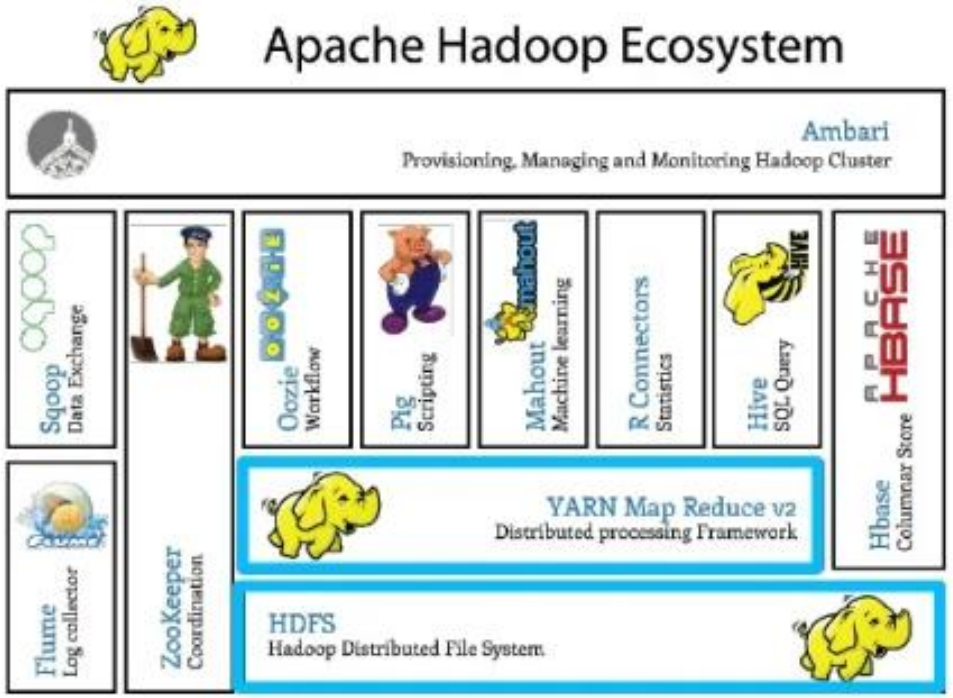
- Parmi les outils
  - **Pig** : langage de script
  - **Hive** : langage proche de SQL
  - **R Connectors** : accès à HDFS et exécution de requêtes Map/Reduce à partir du langage R
  - **Mahout** : bibliothèque de machine learning
  - **Oozie**: ordonnancer les jobs Map Reduce, en définissant des workflows
- D'autres outils sont au dessus du HDFS
  - **Hbase** : base de données NoSQL orientée colonne
  - **Impala** : requêtage de données directement à partir de HDFS en utilisant des requêtes Hive SQL



HADOOP

Ecosystème

- Certains outils permettent de connecter HDFS aux sources externes
  - **Sqoop** : lecture et écriture des données à partir de bases de données externes
  - **Flume** : collecte des logs et stockage dans HDFS
- Enfin, d'autres outils permettent la gestion et l'administration de Haddop
  - **Ambari** : provisionnement, gestion et monitoring des clusters
  - **Zokeeper** : service centralisé pour maintenir les informations de configuration, de nommage et de synchronisation distribuée

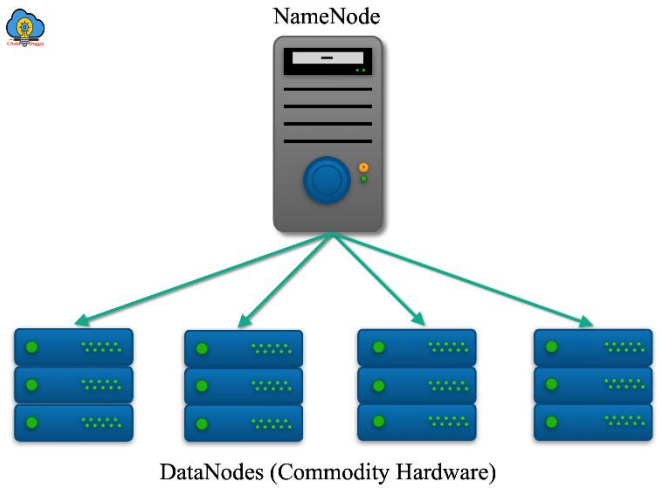


HDFS: Hadoop Distributed File System

Architecture

- HDFS est un système de fichiers distribué, extensible et portable (développé en Java)

- Permet de stocker de très gros volumes de données sur un grand nombre de machines (nœuds) équipées de disques durs banalisés ➔ cluster



NameNode:

- démon s'exécutant sur une machine séparée, il contient des métadonnées
- permet de retrouver les nœuds qui exécutent les blocs d'un fichier

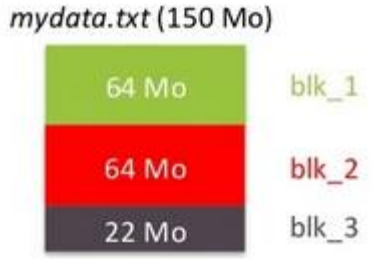
DataNode

- démon sur chaque nœud du cluster
- Ils stockent réellement les données dans HDFS et exécutent les instructions du NameNode

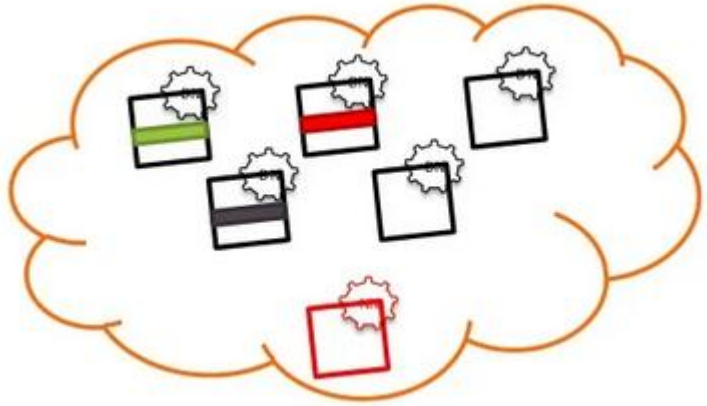
HDFS: Hadoop Distributed File System

Architecture

- Quand un fichier de données enregistré dans HDFS, il est décomposé en grands blocs (par défaut 64Mo), chaque bloc ayant un nom unique: blk_1, blk_2…
- Chaque bloc est enregistré sans un nœud différent du cluster



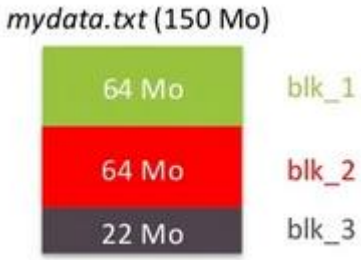
- **Quels sont les problèmes possibles:**
  - Panne de réseau
  - Panne de disque sur les DataNodes
  - Pas tous les DN sont utilisés
  - Les tailles des blocs sont différentes
  - Panne de disque sur les NameNodes



HDFS: Hadoop Distributed File System

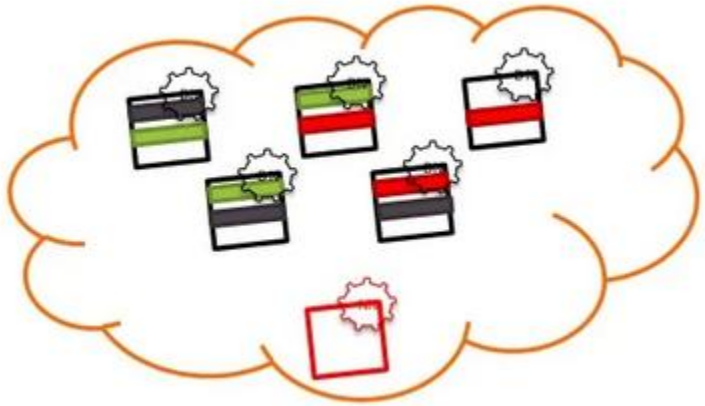
Architecture

+ **Si l'un des nœuds a un problème, les données seront perdues**
  - Hadoop réplique chaque bloc 3 fois (par défaut)
  - 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 trois copies stockées
  - **Et si le NameNode a un problème?**



+ **Si c'est un problème d'accès (réseau), les données sont temporairement inaccessibles**

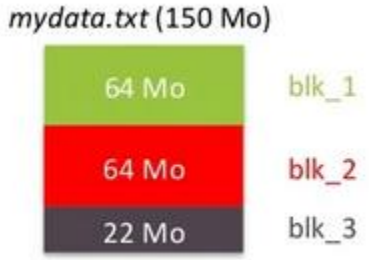
+ **Si le disque du NN est défaillant, les données sont perdues à jamais**



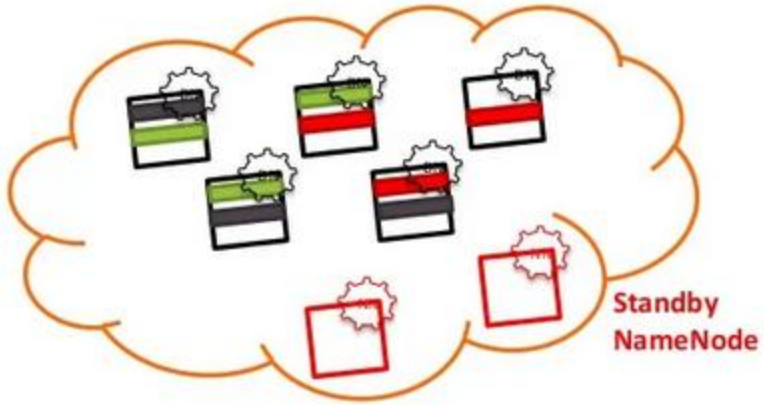
HDFS: Hadoop Distributed File System

Architecture

+ Pour éviter cela, le NameNode sera dupliqué, non seulement sur son propre disque, mais également quelque part sur le système de fichier du réseau



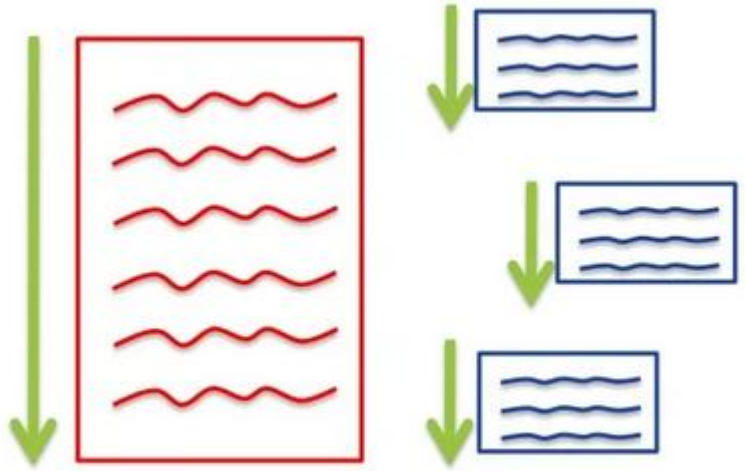
+ Définition d'un autre NameNode (standby NameNode) pour reprendre le travail si le NameNode actif est défaillant



HDFS: Hadoop Distributed File System

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 Hadoop Streaming, il est possible d'utiliser d'autres langages comme Python ou Ruby
- Au lieu de parcourir le fichier séquentiellement (beaucoup de temps), il est divisé en morceaux qui sont parcourus en parallèle



Map-Reduce

Exemple

- Imaginons que vous avez une chaine de 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 des livres suivent le format :

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

- Possibilité
  - Pour chaque entrée, saisir la ville et le prix
  - Si on trouve une entrée avec une ville déjà saisie, on les regroupe en faisant la somme des ventes

- Dans un environnement de calcul traditionnel, on utilise généralement les tables de hachage sous forme de (clef, valeur)

- Dans notre cas, la clef serait l'adresse du magasin et la valeur le total des ventes

Map-Reduce

Exemple

- Si on utilise les tables de hachage sur 1To.
  - Le traitement séquentiel de toutes les données peut s'avérer très long
  - Plus on a 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

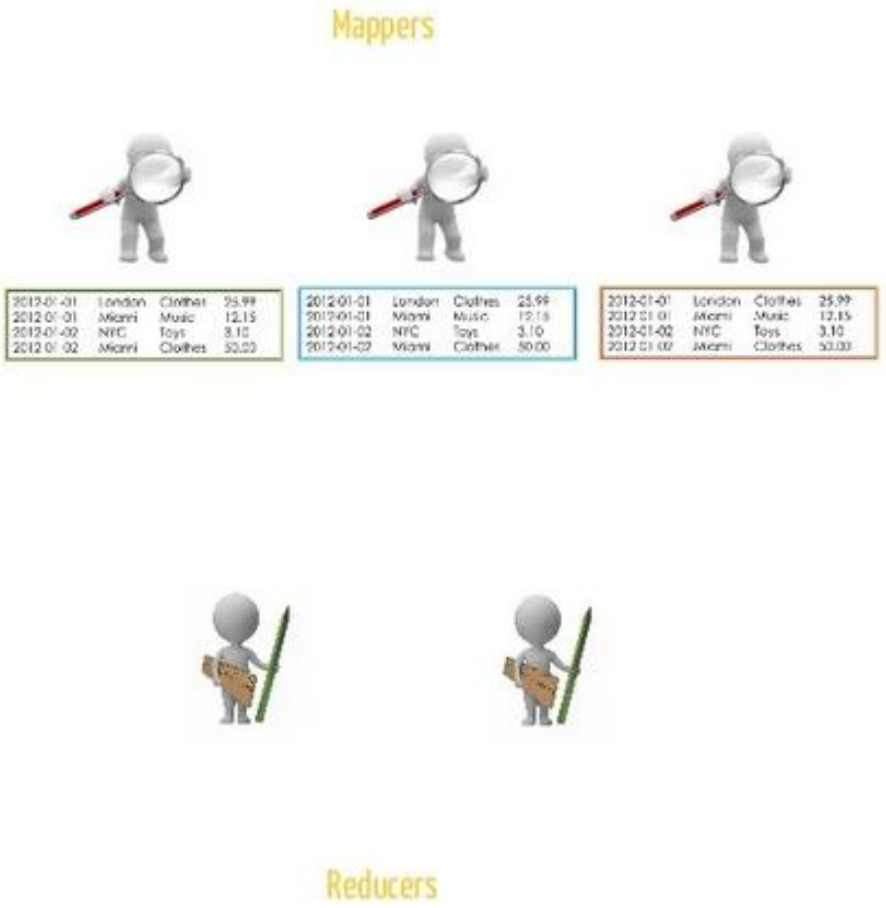
2012-01-01	London	Clothes	25.99
2012-01-01	Miami	Music	12.15
2012-01-02	NYC	Toys	3.10
2012-01-02	Miami	Clothes	50.00

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 pour 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

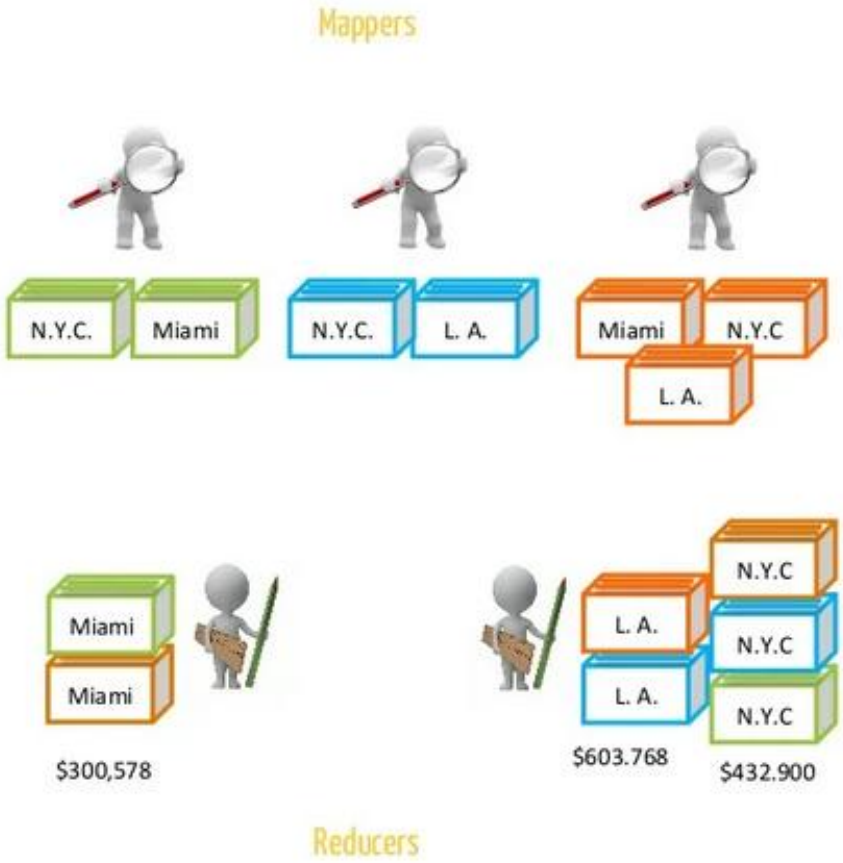
Exemple

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



Map-Reduce

Exemple

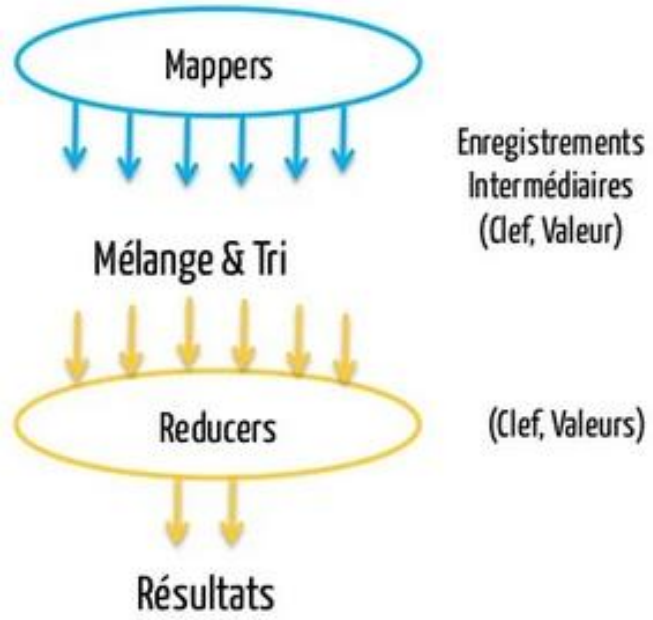
- **Le reducer reçoit des données comme suit**
  - Miami 12.34
  - Miami 99.07
  - Miami 3.14
  - NYC 99.77
  - NYC 88.99

- **Pour chaque entrée, de quoi avons-nous besoin pour calculer la totalité des ventes pour chaque magasin?**
  - Coût en cours
  - Ventes totales en magasin
  - Magasin précédent
  - Magasin en cours

Map-Reduce

Fonctionnement

- Les mappers sont de petits programmes qui commencent par traiter chacun une petite partie de 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

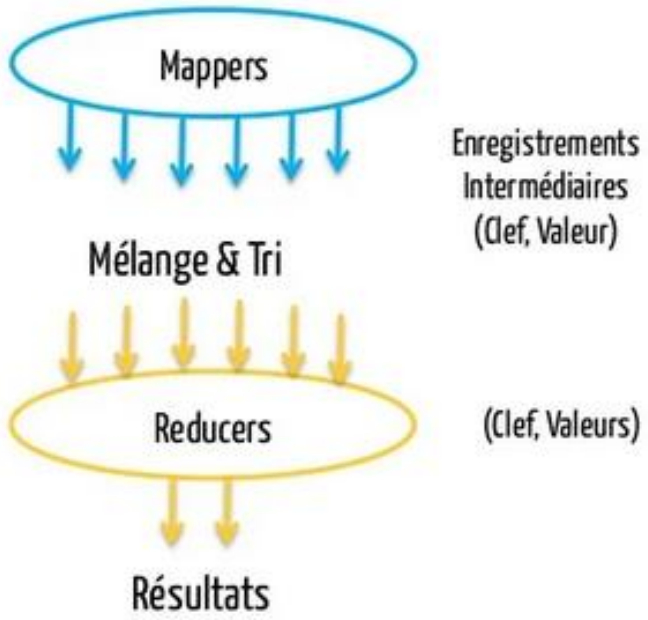


Map-Reduce

Résultats

- 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

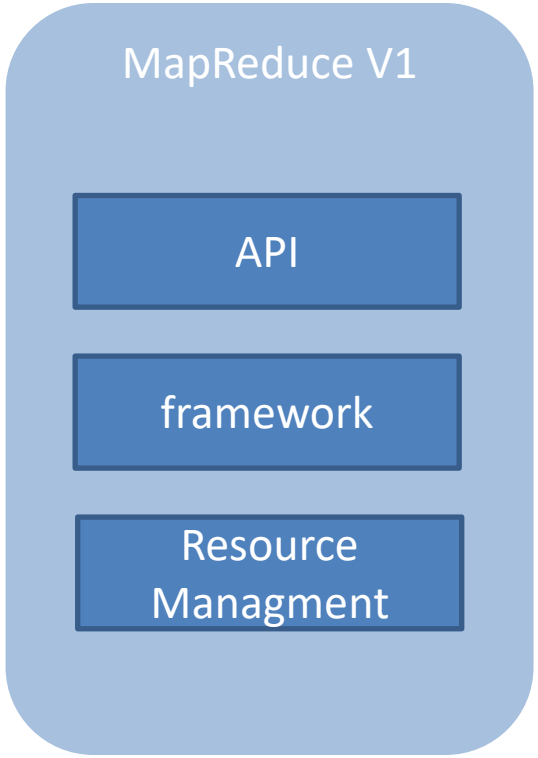


Map-Reduce V1 (MRv1)

Composants

- MapReduce v1 intègre trois composants

- **API**
  - Permettre au programmeur l'écriture d'application MapReduce
- **Framework**
  - Permettre l'exécution des jobs MapReduce, le Shuffle/Sort…
- **Resource Management**
  - Infrastructure pour gérer les nœuds du cluster, allouer des ressources et ordonnancer les jobs



Map-Reduce V1 (MRv1)

Démons

JobTracker

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

TaskTracker

- S'exécuter sur chacun des nœuds pour exécuter les vraies tâches de Map-Reduce
- Choisir 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)

Map-Reduce V1 (MRv1)

Fonctionnement

- Un job Map-Reduce est divisé sur plusieurs tâches appelées mappers et reducers
- Chaque tâche est exécutée sur un nœud du cluster
- Chaque nœud a un certain nombre de slots prédéfinis:
  - Map Slots
  - Reduce slots
- Un slo est une unité d'exécution qui représente la capacité du task tracker à exécuter une tâche (map ou reduce) individuellement, à un moment donné
- Le Job Tracker se charge à la fois:
  - D'allouer les ressources (mémoire, CPU…) aux différents tâches
  - De coordonner l'exécution des jobs Map-Reduce
  - De réserver et ordonnancer les slots, et de gérer les fautes en réallouant les slots au besoin

Map-Reduce V1 (MRv1)

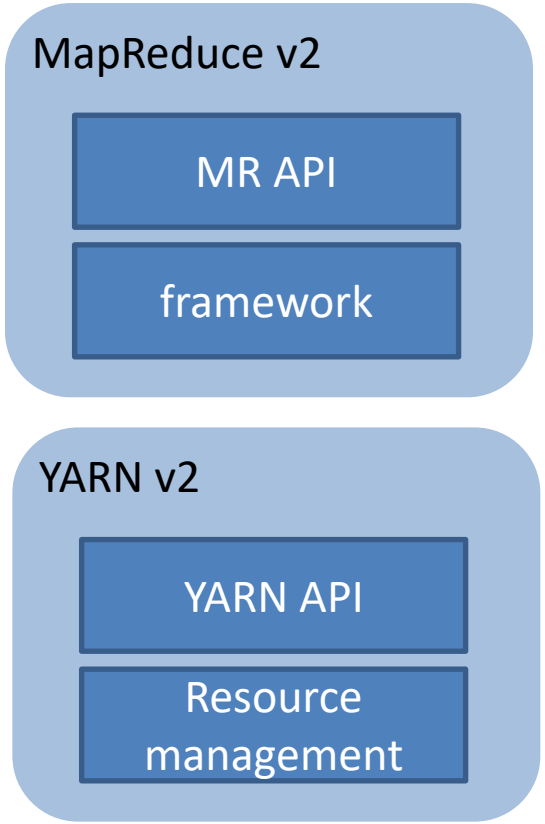
Problèmes

- Le Job Tracker s'exécute sur une seule machine, et fait plusieurs tâches (gestion de ressources, ordonnancement et monitoring des tâches…)
  - **Problème de scalabilité**: les nombreux datanodes existants ne sont pas exploités, et le nombre de nœuds par cluster limité à 4000
- Si le Job Tracker tombe en panne, tous les jobs doivent redémarrer
  - Problème de disponibilité: SPoF
- Le nombre de map slots et de reduce slots est prédéfini
  - Problème d'exploitation: si on a plusieurs map jobs à exécuter, et que les map slots sont pleins, les reduce slots ne peuvent pas être utilisés et vice-versa.
- Le Job Tracker est fortement intégré à Map Reduce
  - Problème d'interoperabilité: impossible d'exécuter des applications MapReduce sur HDFS

Map-Reduce V2 (MRv2)

Composants

- MapReduce v2 sépare la gestion des ressources de celle des tâches MR
- **Pas de notion de slots**
  - Les nœuds ont des ressources (CPU, mémoire…) allouées aux applications à la demande
- **Définition de nouveaux démons**
  - La plupart des fonctionnalités du Job Tracker sont déplacées vers le Application Master
  - Un cluster peut avoir plusieurs Application Masters
- **Support des application MR et non-MR**



Map-Reduce V2 (MRv2)

Démons

Resource Manager (RM)

- ✓ Tourne le nœud master et ordonnancer les ressources
- ✓ Arbitrer les ressources entre plusieurs applications

Node Manager (NM)

- ✓ S'exécuter sur les nœuds esclaves
- ✓ Communiquer avec RM

Resource Manager (RM)

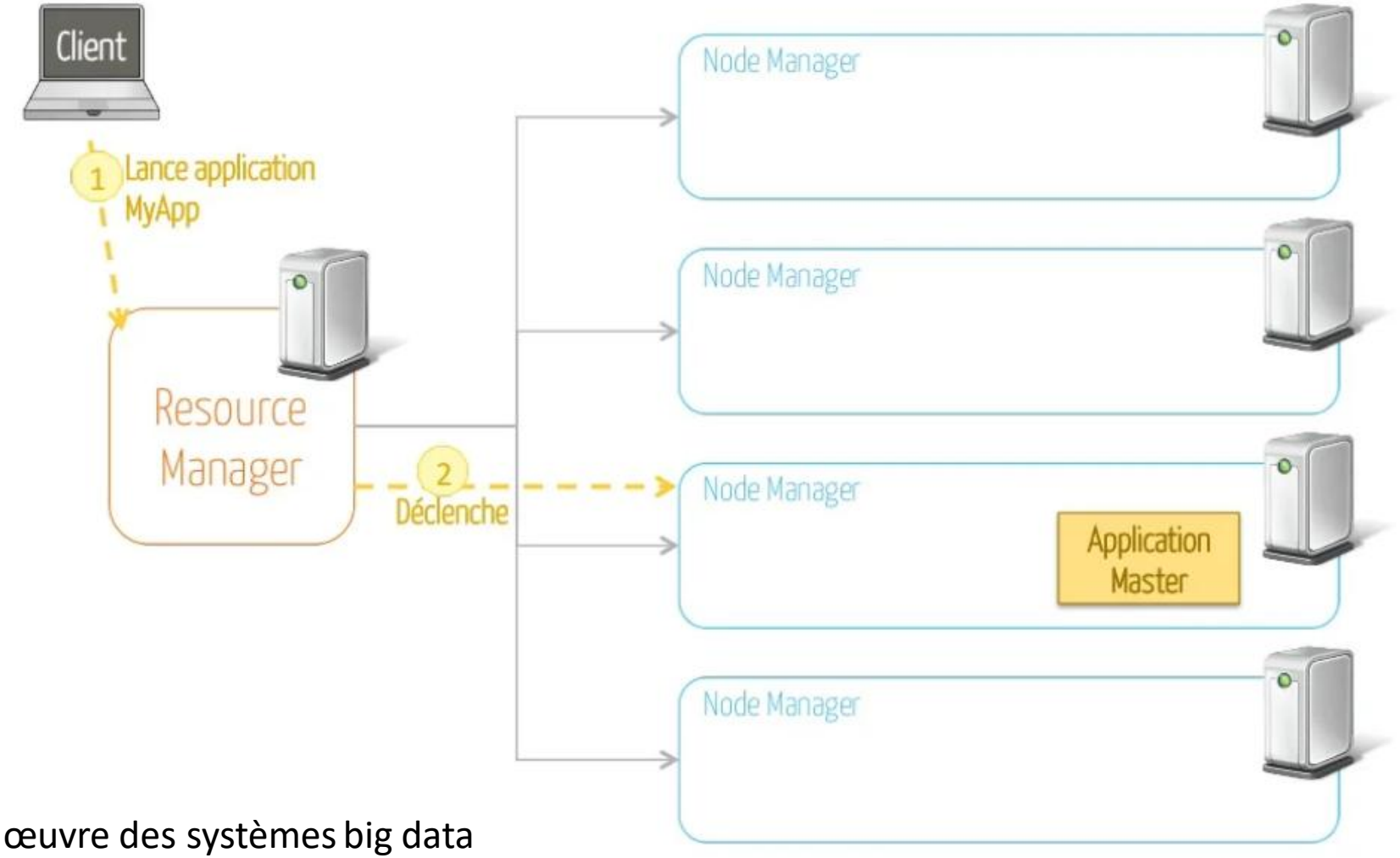
- ✓ Créés par RM à la demande
- ✓ se voit allouer des ressources sur le nœud esclave

Resource Manager (AM)

- ✓ Un seul par application et s'exécute sur un container
- ✓ Demande plusieurs containers pour exécuter les tâches de l'application

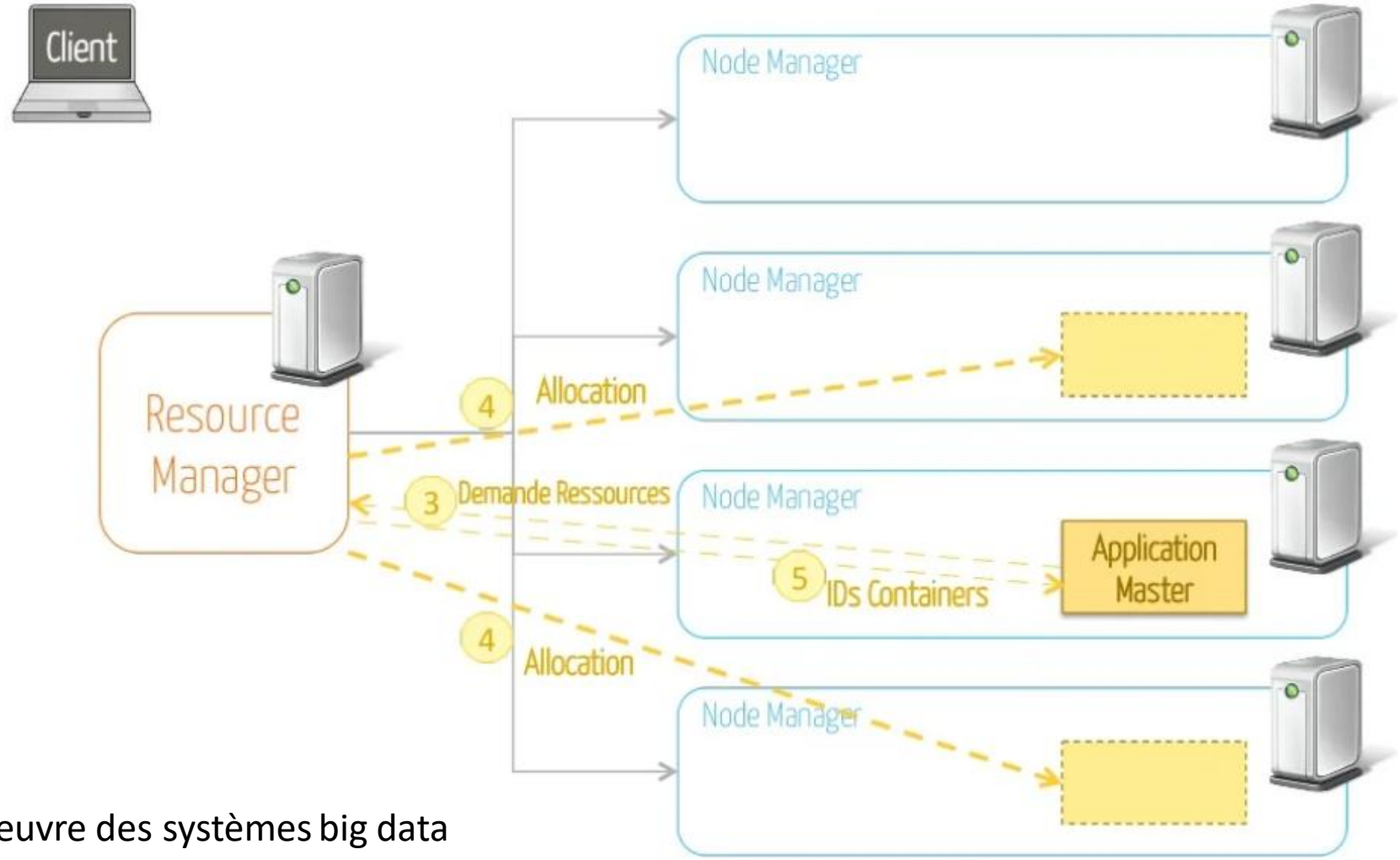
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



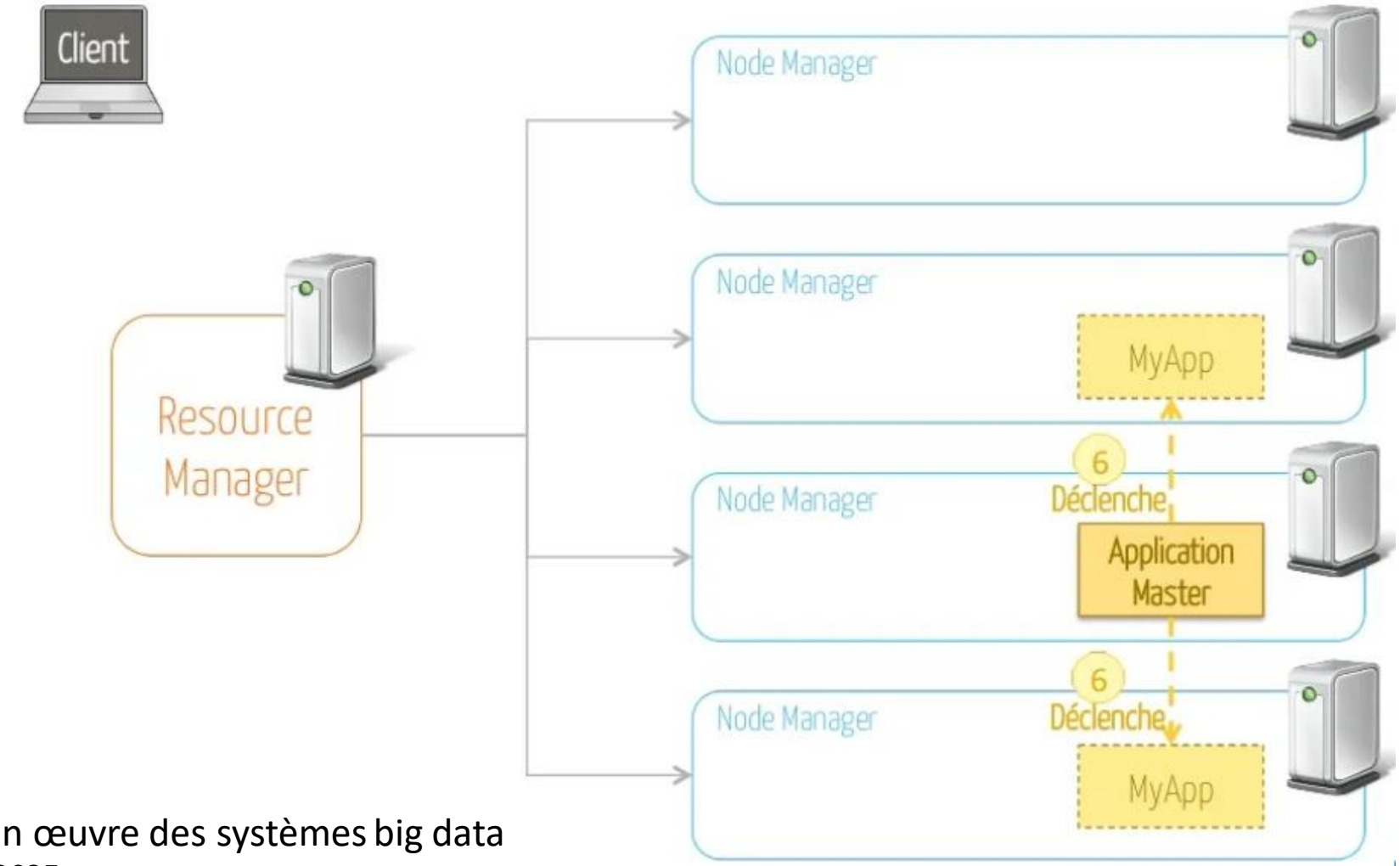
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



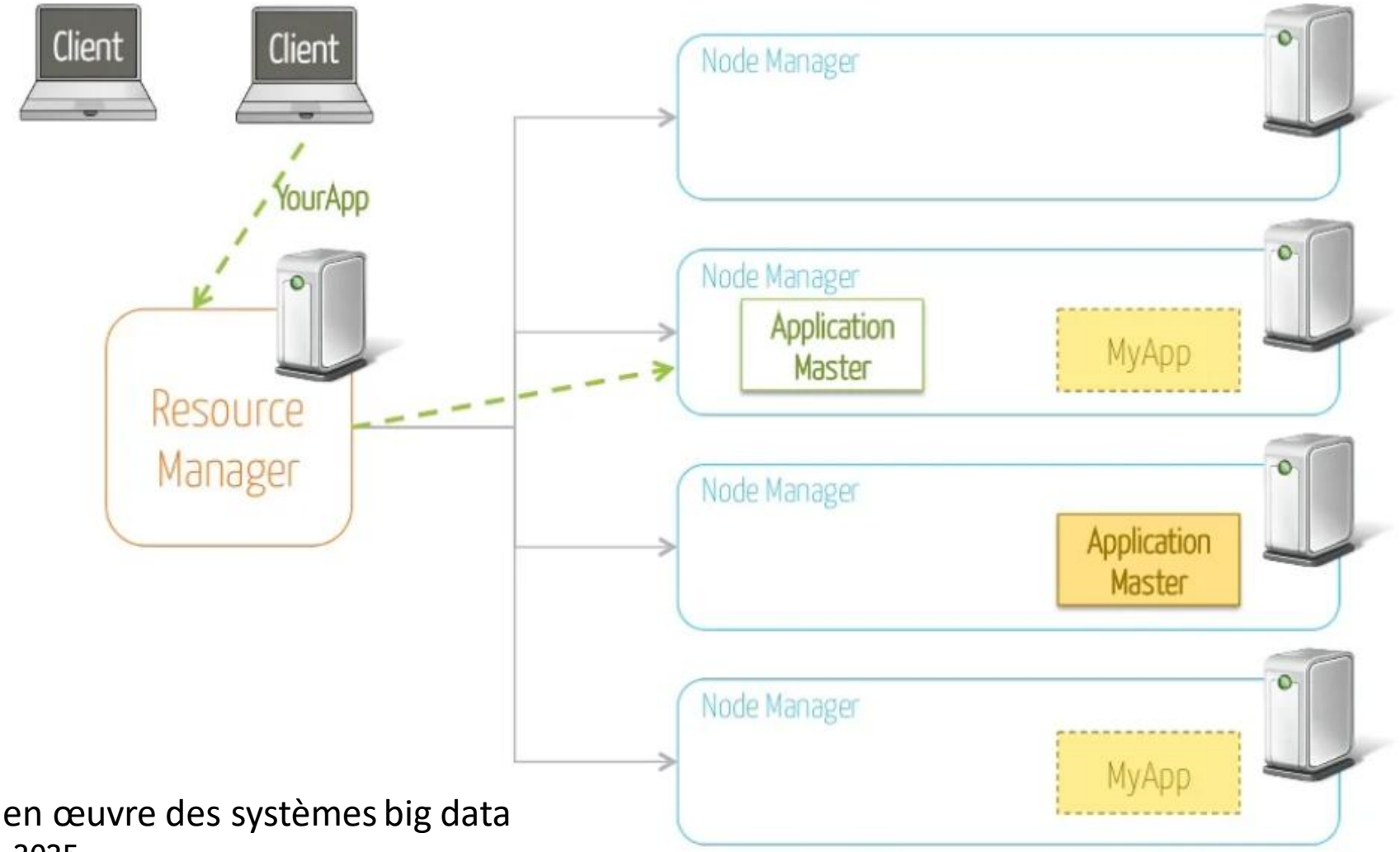
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



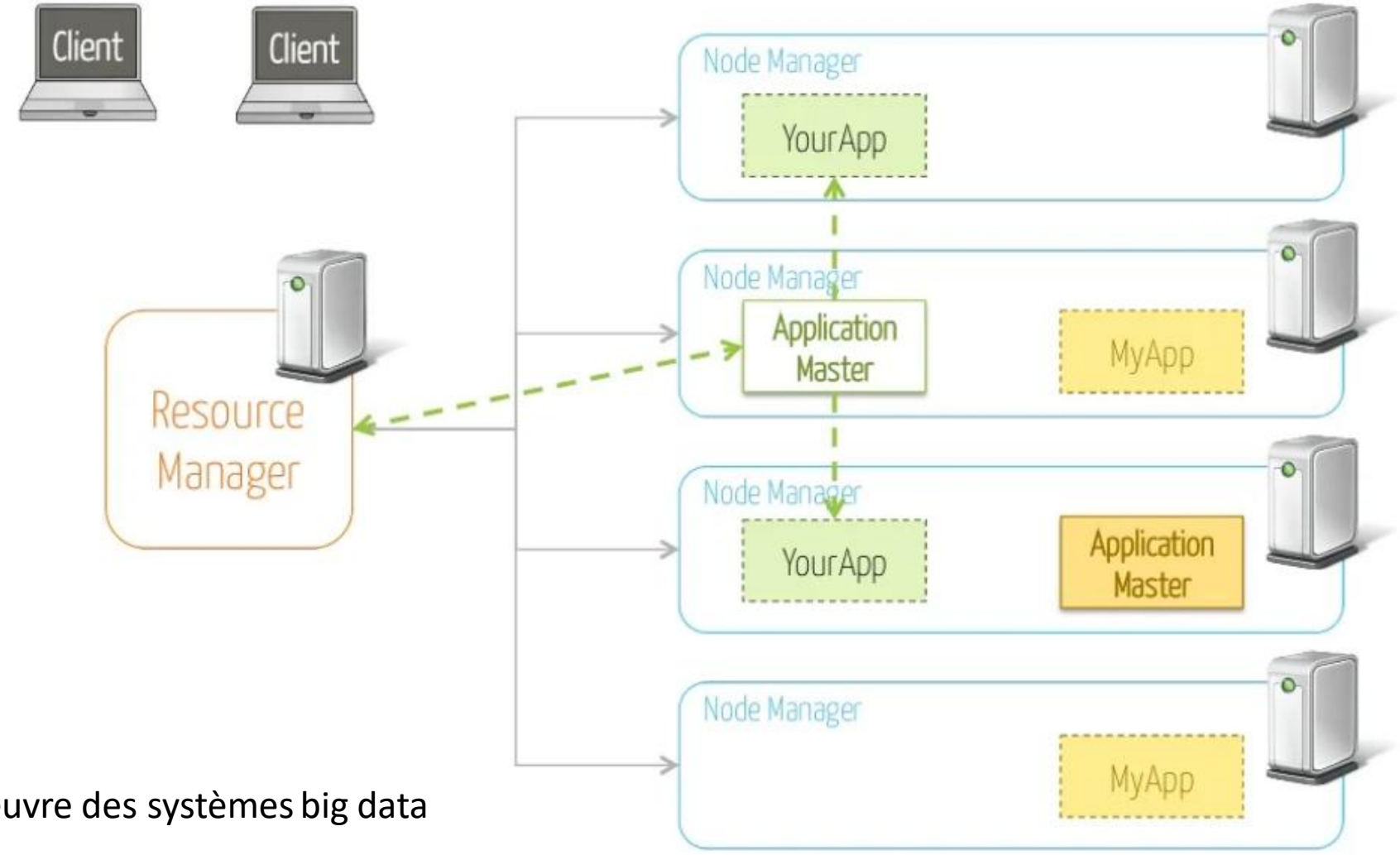
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



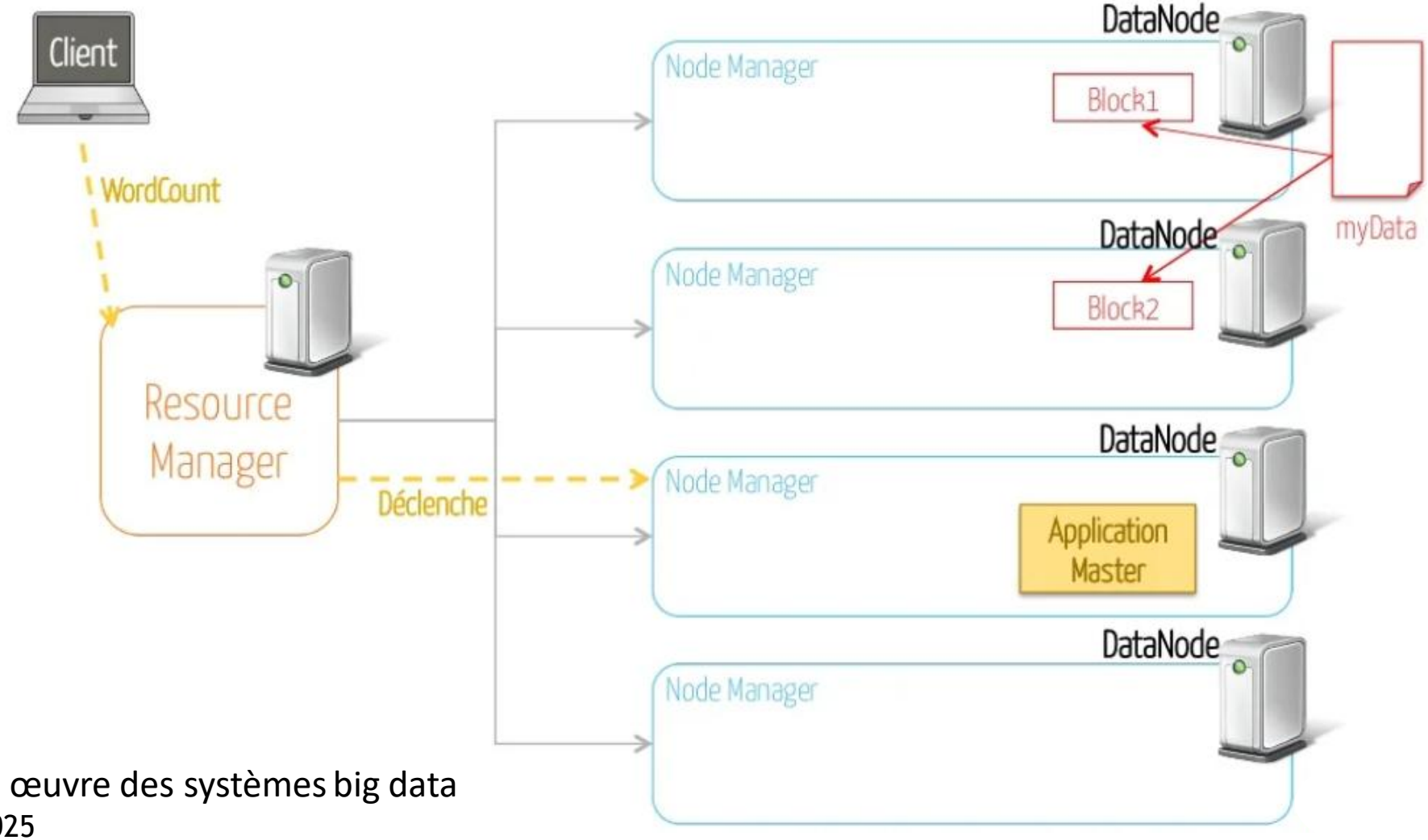
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



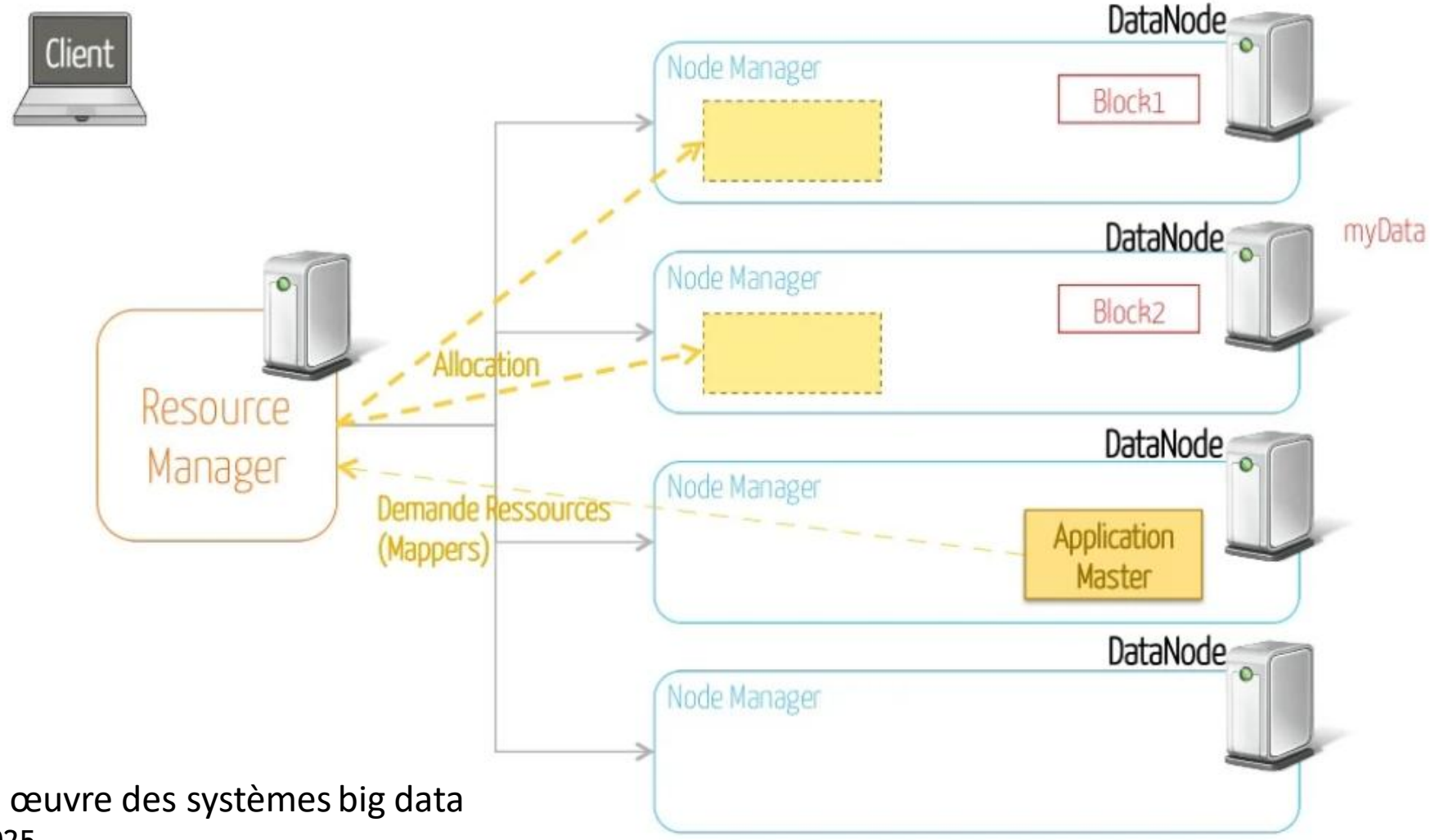
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



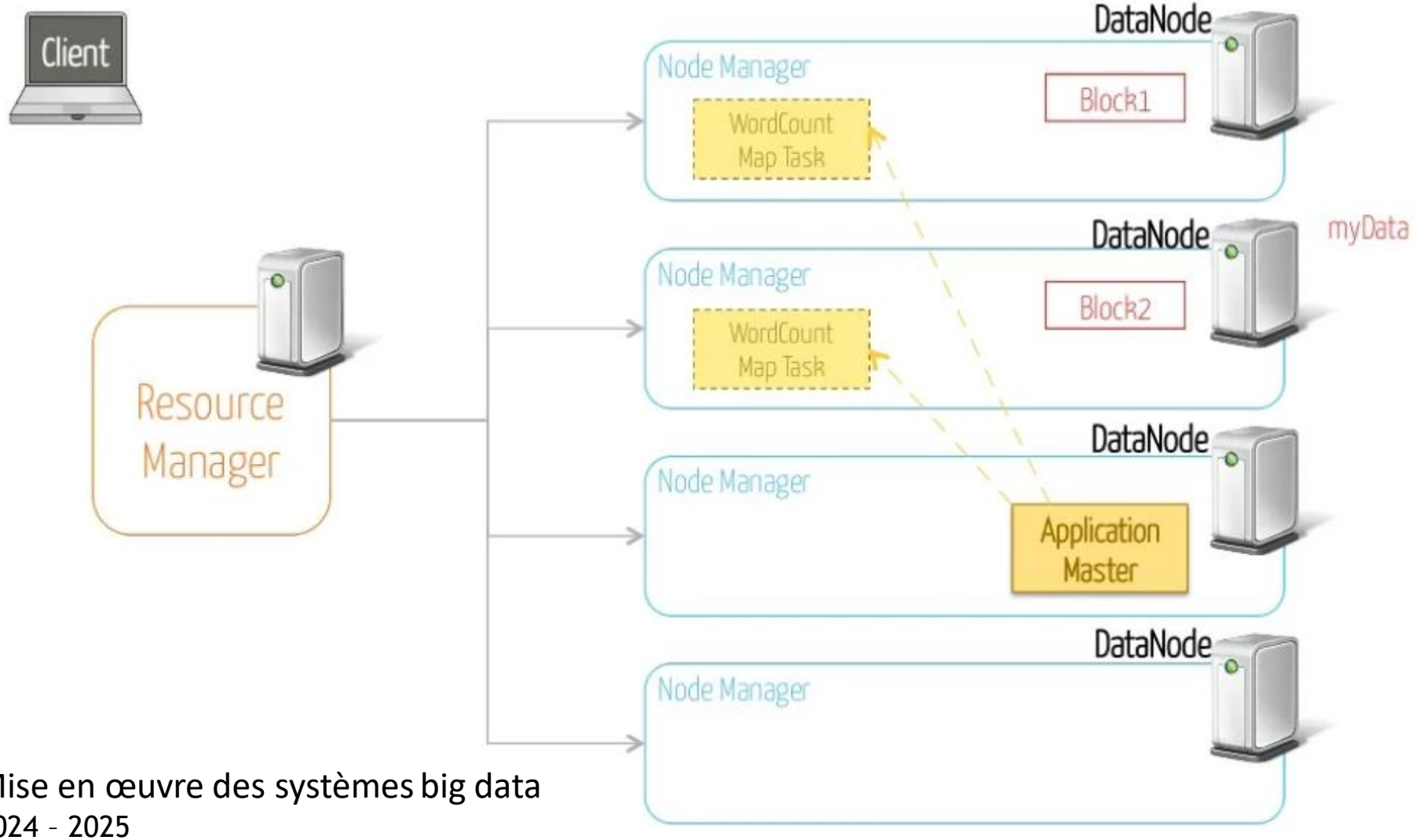
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



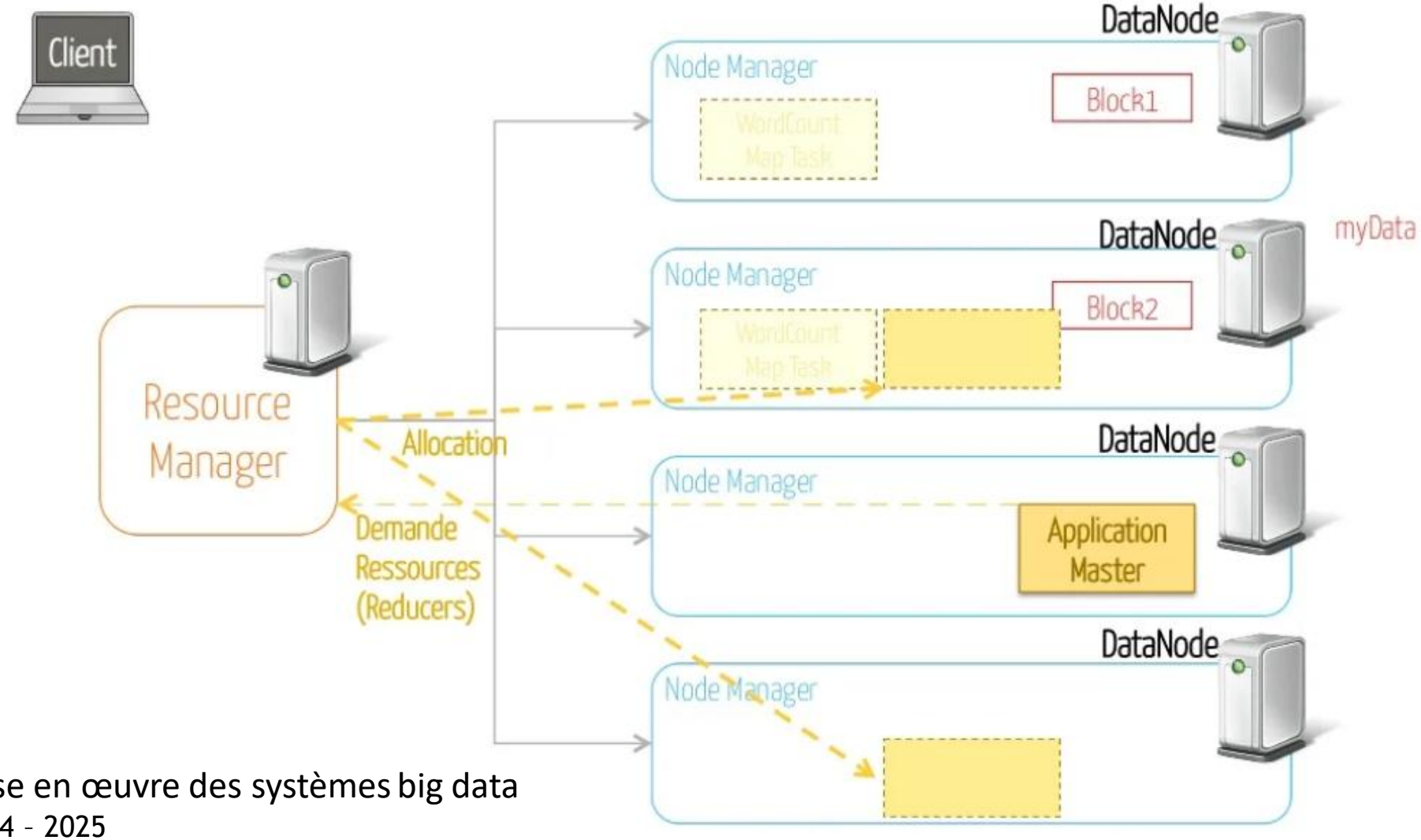
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



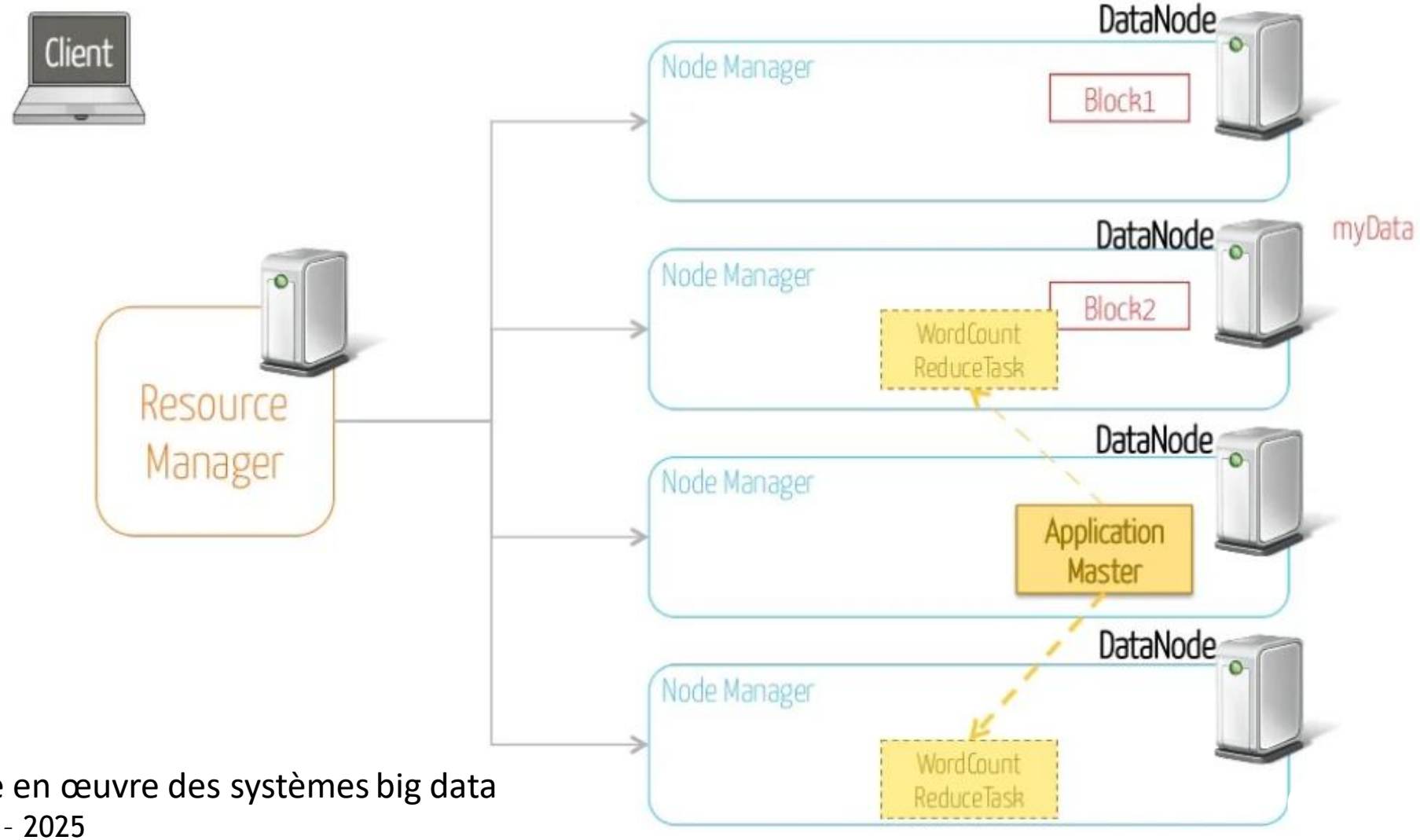
Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



Map-Reduce V2 (MRv2)

Lancement d'une application dans un cluster YARN



Map-Reduce Design Patterns

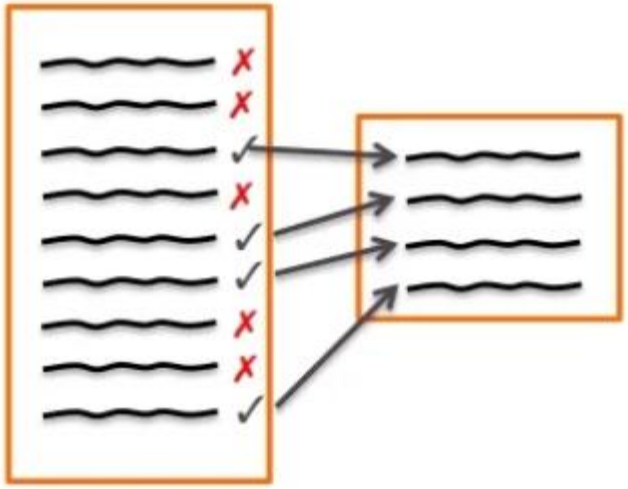
Présentation

- Les design patterns (patrons de conception) représentent les types de traitements Map-Reduce les plus utilisés avec Haddop
- Classés en trois catégories:
  - Patrons de filtrage (filtering patterns)
    - Echantillonage de données
    - Listes des top-n
  - Patrons de filtrage (filtering patterns)
    - Comptage des enregistrements
    - Trouver les min et les max
    - Statistiques
    - indexes
  - Patrons structurels
    - Combinaison de données relationnelles

Patrons de filtrage

Présentation

- Ne modifient pas les données
- Trient, parmi les données présentes, lesquelles garder et lesquelles enlever
- On peut obtenir
  - **Des filtres simples**: définition d'une fonction indiquant le critère de filtrage
  - **L'échantiollonage** (sampling): création d'un petit ensemble d'enregistrements à partir d'un grand ensemble, en retenant des échantillons (comme la valeur la plus élevée d'un champs particulier)
  - **L'échantiollonage aléatoire**: retenir un échantillon représentatif des données initiales
  - **Les listes Top-n**



Patrons de filtrage

Exemple

+ Exemple de filtrage simple

- **Cas d'étude**: fichier contenant tous les posts des utilisateurs sur un forum
- **Filtre**: retenir les posts les plus courts, contenant une seule phrase
  - Une phrase est un post qui ne contient aucune ponctuation ou alors une à la fin

```
def mapper():
    reader = csv.reader(sys.stdin, delimiter='\t')
    writer = csv.writer(sys.stdout, delimiter='\t', quotechar='"'
                                                , quoting=csv.QUOTE_ALL)
    for line in reader:

        for i in line:
            #print('-',i)
            if len(i) == 0:
                continue
            if "!" in i[:-1] :
                continue
            if "." in i[:-1]:
                continue
            if "?" in i[:-1]:
                continue
            else:
                writer.writerow(line)
```

Hadoop

Récapitulatif

Avantages

- Gestion de défaillances
- Sécurité et persistance des données
- Montée en charge
- Complexité réduite
- Coût réduit

Inconvénients

- Difficulté d'intégration avec d'autres systèmes
- Administration complexe
- Temps de latence important
- Produit en développement continu

La Rochelle Université

Traitement de données

Batch vs. Streaming processing

Batch processing (traitement par lot)

Présentation

- Moyen efficace de traiter de grands volumes de données
- Les données sont collectées, stockées, traitées, puis les résultats sont fournis
- Les systèmes de batch processing ont besoin de programmes différents pour l'entrée, le traitement et la génération des données
- Le traitement est réalisé sur l'ensemble de données
- Traitement de données complexes (CEP: Complex Event Provessing)
  - Concept de traitement des évènements pour identifier les évènements significatifs
  - Détection de schémas complexes, corrélation, abstraction et hiérarchie entre évènements
- Hadoop Map-Reduce est un exemple de système utilisant le traitement par lot

Batch processing (traitement par lot)

Caractéristiques

- A accès à toutes les données
- Peut réaliser des traitements lourds et complexes
- Est en général plus concerné par le débit (nombre d'actions réalisées en une unité de temps) que par la latence (temps requis pour réaliser l'action) des différents composants du traitement
- Sa latence est calculée en minutes (voire plus)
- Cible les caractéristiques volume et variété des big data

Batch processing (traitement par lot)

Inconvénients

- Il n'est pas possible d'exécuter des travaux récursifs ou itératifs de manière inhérente
- Toutes les données doivent être prêtes avant le début du job
  - N'est pas appropriée pour des traitements en ligne ou temps réel
- Latence d'exécution élevée
  - Produit des résultats sur des données relativement anciennes

Batch processing (traitement par lot)

Cas d'utilisation

- Les chèques de dépôt dans une banque accumulés et traités chaque jour
- Les statistiques par jour/mois/année
- Factures générées pour les cartes de crédit (en général mensuelles)

Traitement par streaming

Caractéristiques

- Les traitements se font sur un élément ou un petit nombre de données récentes
- Le traitement est relativement simple
- Doit compléter chaque traitement en un temps proche du temps réel
- Les traitements sont généralement indépendants
- Asynchrone: les sources de données n'interagissent pas directement avec l'unité de traitement en streaming, en attendant une réponse par exemple
- La latence de traitement est estimée en secondes

Traitement par streaming

Inconvénients

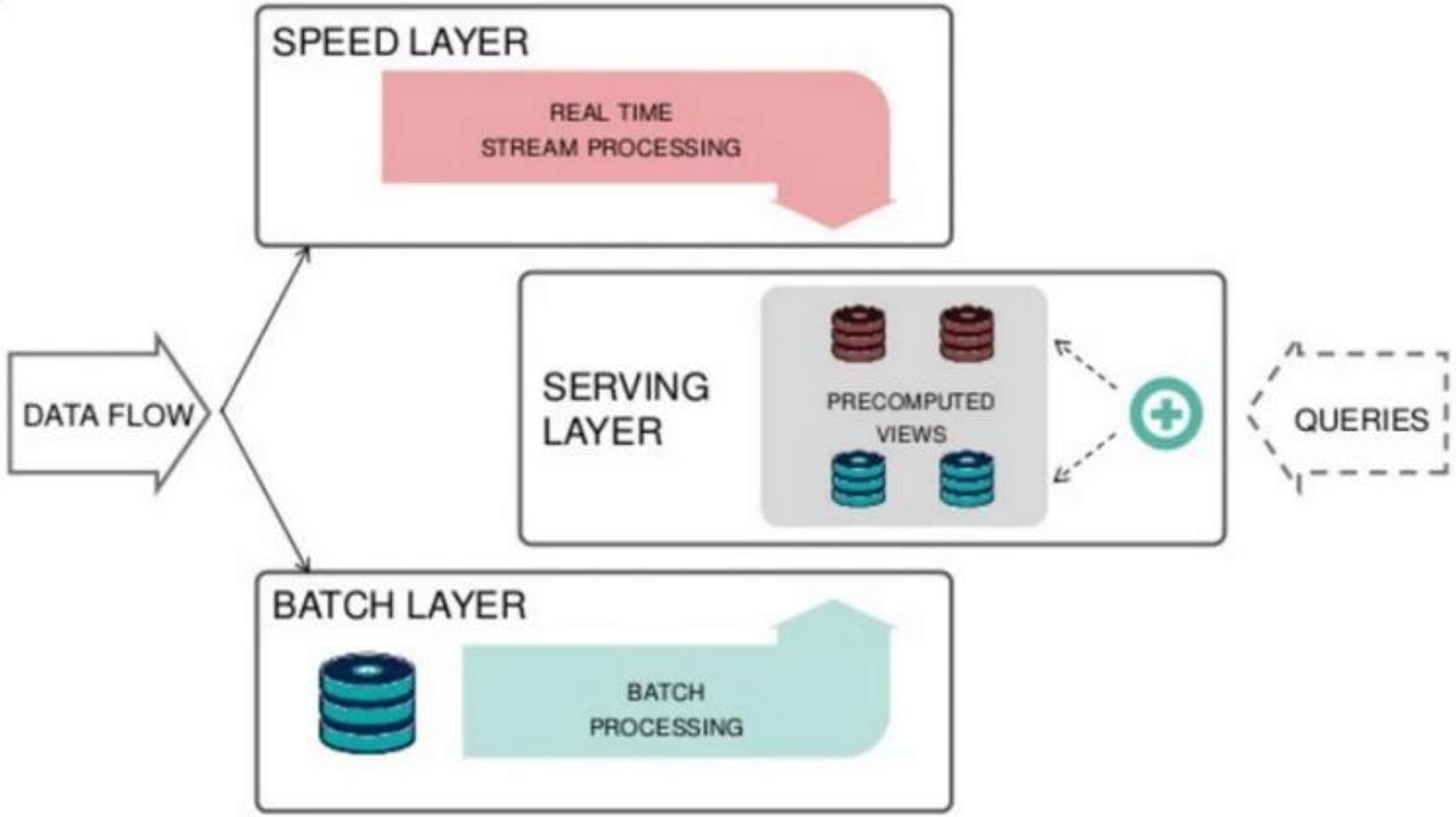
- Pas de visibilité sur l'ensemble de données
  - Certains types de traitements ne peuvent pas être réalisés
- Complexité opérationnelle élevée
  - Plus complexes à maintenir que les traitements batch
  - Le système doit être toujours connecté, toujours prêt, avoir des temps de réponses courts et gérer les données à l'entrée
- Risque de pertes de données

Traitement par streaming

Cas d'utilisation

- Recommandation en temps réel
  - Prise en compte de navigation récente, géolocalisation
  - Publicité en ligne, re-marketing
- Surveillance de larges infrastructures
- Agrégation de données financières à l'échelle d'une banque
- Internet des objets

Lambda Architecture



Lambda Architecture

Couches

+ Batch Layer

- Gérer l'unité de stockage principale: des données brutes, immuables et complètes
- Précalculer et conserver les résultats de requêtes appelées batch views

+ Serving Layer

- Indexer les batch views pour qu'elles soient requêtées au besoin avec une faible latence

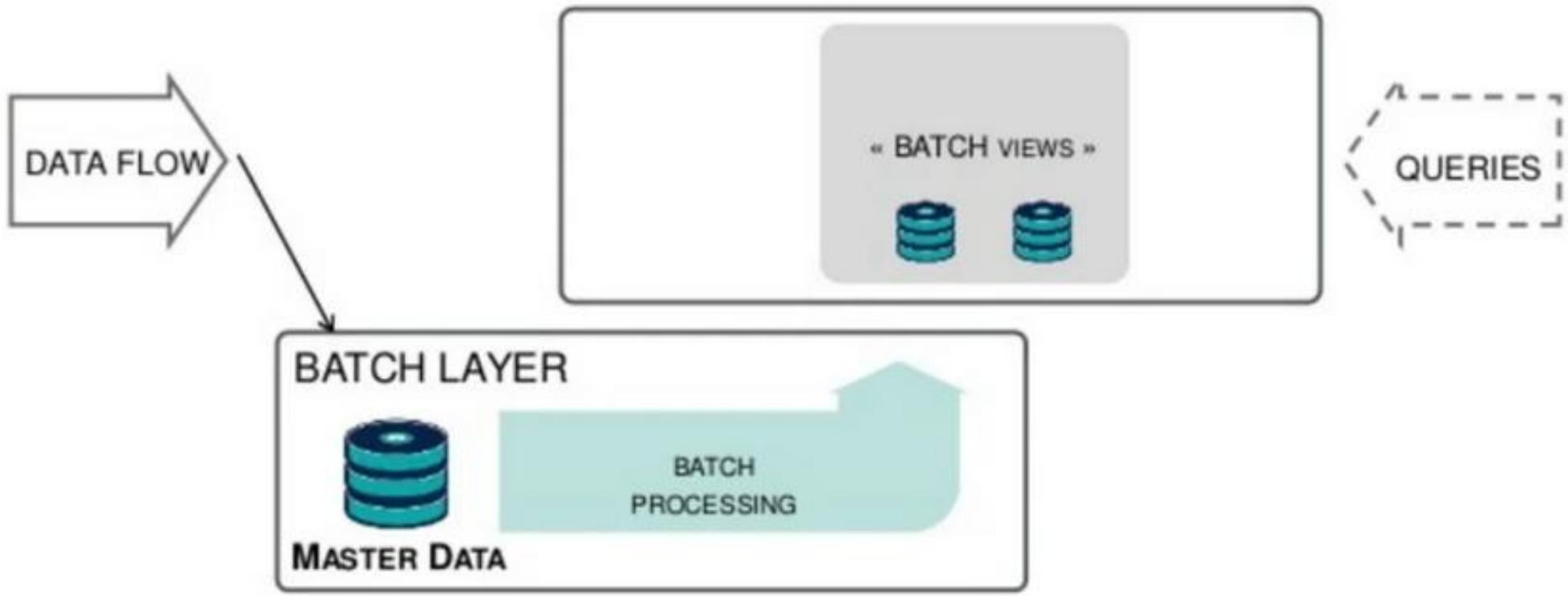
+ Speed Layer

- Accommoder toutes les requêtes sujettes à des besoins de faible latence
- Utilisation d'algorithmes rapides et incrémentaux
- Gérer les données récentes uniquement

Lambda Architecture

Batch Layer

- **Stockage maître + Traitements Batch**



Lambda Architecture

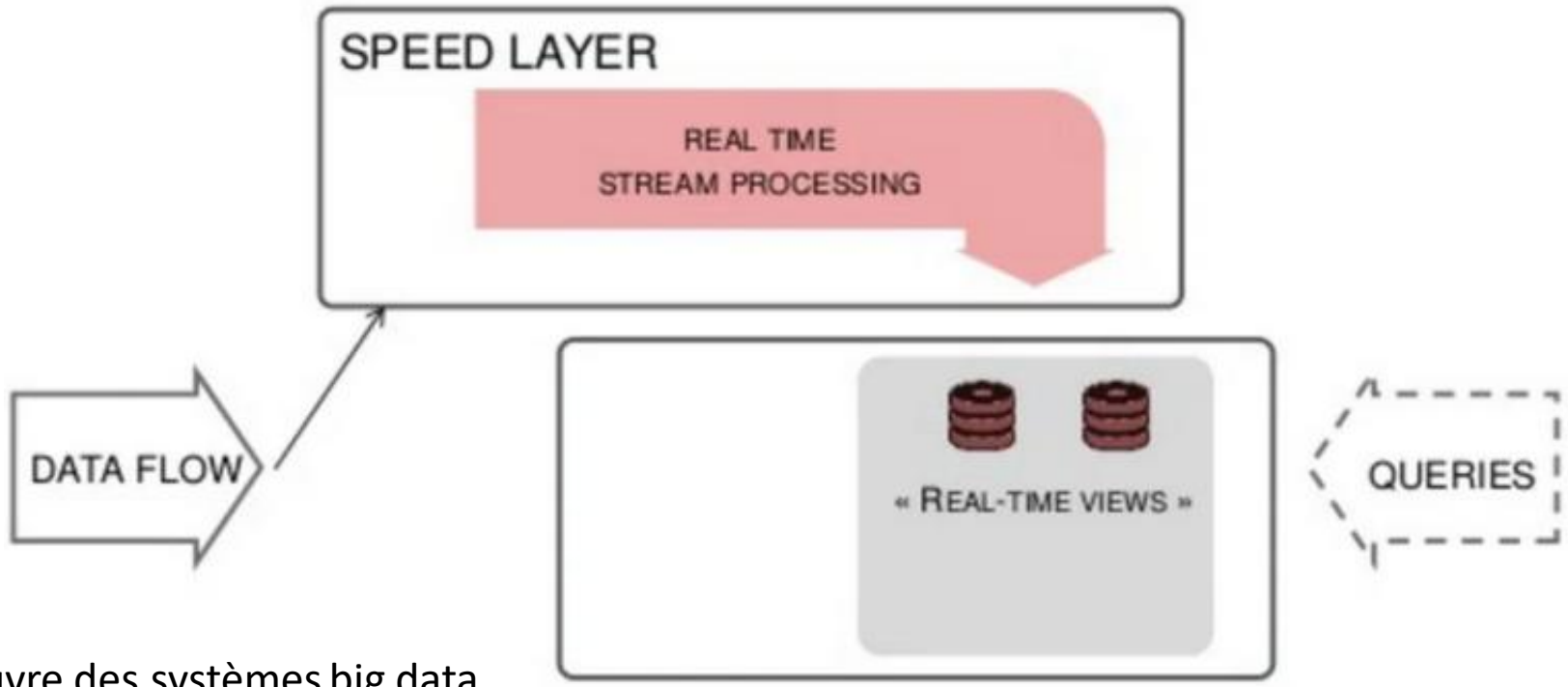
Batch Layer

- **Besoins**
  - Stockage scalable
    - ✓ Haute distribution
    - ✓ Support de la charge et du volume
  - Tolérance aux pannes
    - ✓ Système de réplication et distribution des données
  - Robustesse
    - ✓ Surtout concernant les évolutions du schéma
  - Permettant tout type de traitement
- **Technologies**
  - Hadoop
  - Avro : système de sérialisation de données
  - Hive …

Lambda Architecture

Speed Layer

+ Mise à jour des vues en continu, de manière incrémentale
+ Latence : ~10ms ➔ qlq secondes



Lambda Architecture

Speed Layer

+ Besoins

- Traitement en continu (stream processing)
- Architecture asynchrone, distribué et scalable
- Tolérance aux pannes
- Garanties de traitement si possible
  - ✓ Rejeu possible des messages en cas de perte d'un nœud

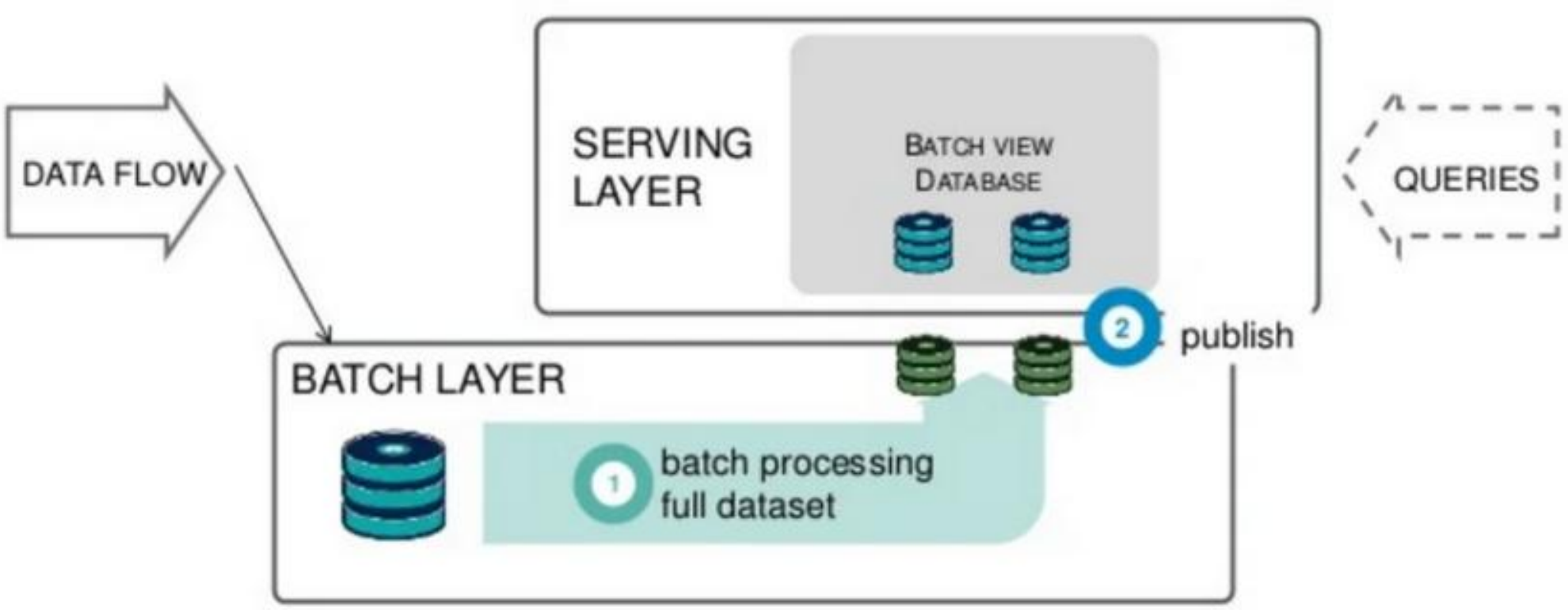
+ Technologies

- **Kafka** : collecte temps réel
- **Akka** : message-driven applications
- **Storm** : streaming processing
- **Spark** : micro-batch processing

Lambda Architecture

Serving Layer : batch views

+ Vues précalculées



Lambda Architecture

Serving Layer : batch views

Besoins

- ❑ Écritures massives
- ❑ Lectures indexées, accès aléatoire à faible temps de réponse
- ❑ Scalabilité et tolérance aux pannes

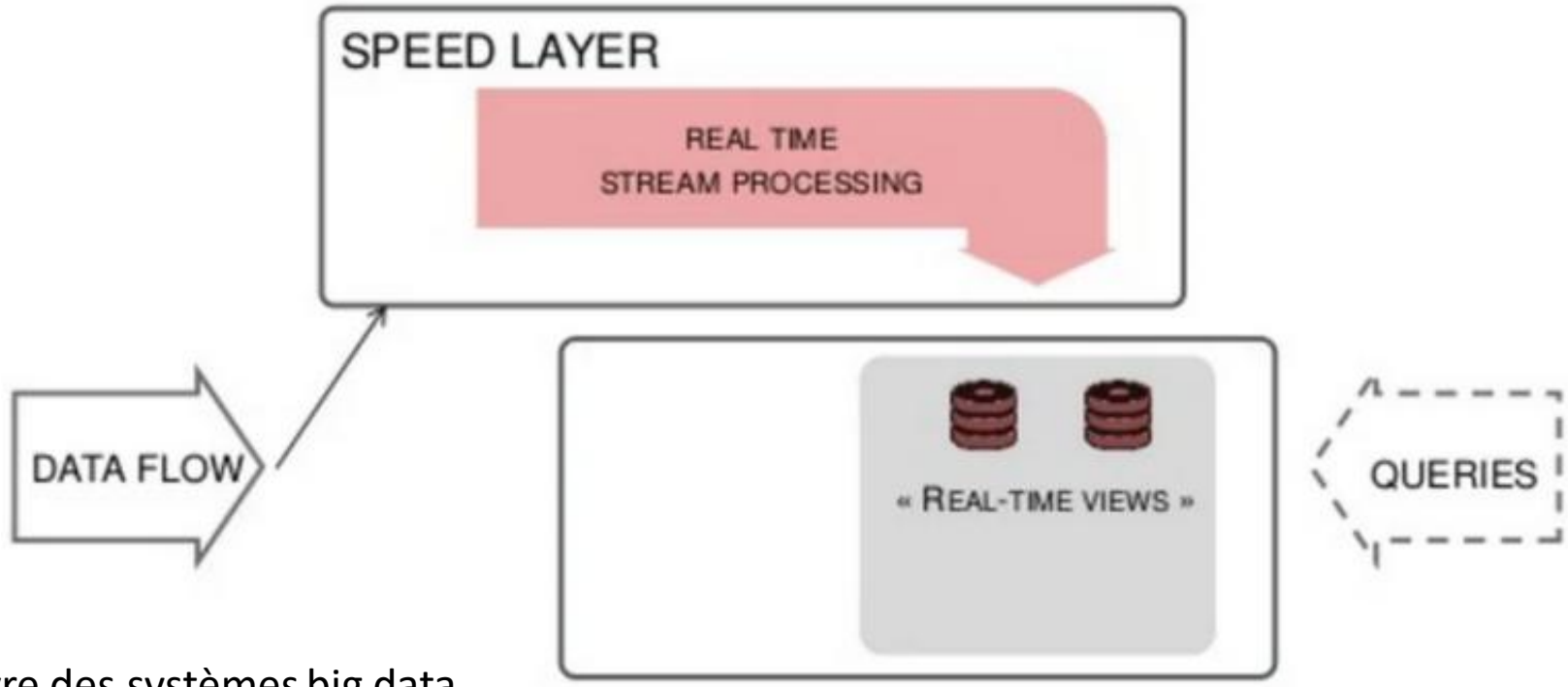
Technologies

- ❑ Cassandra
- ❑ Hbase
- ❑ SploutSQL : requêtage SQL à faible latence
- ❑ …

Lambda Architecture

Serving Layer : realtime views

+ Vues requêtées de façon intensive et performante
+ Temps de réponse court, fort débit de requête supporté



Lambda Architecture

Serving Layer : realtime views

Besoins

- Support de fortes sollicitations en lecture et écriture (mise à jour incrémentale)
- Scalabilité et tolérance aux pannes

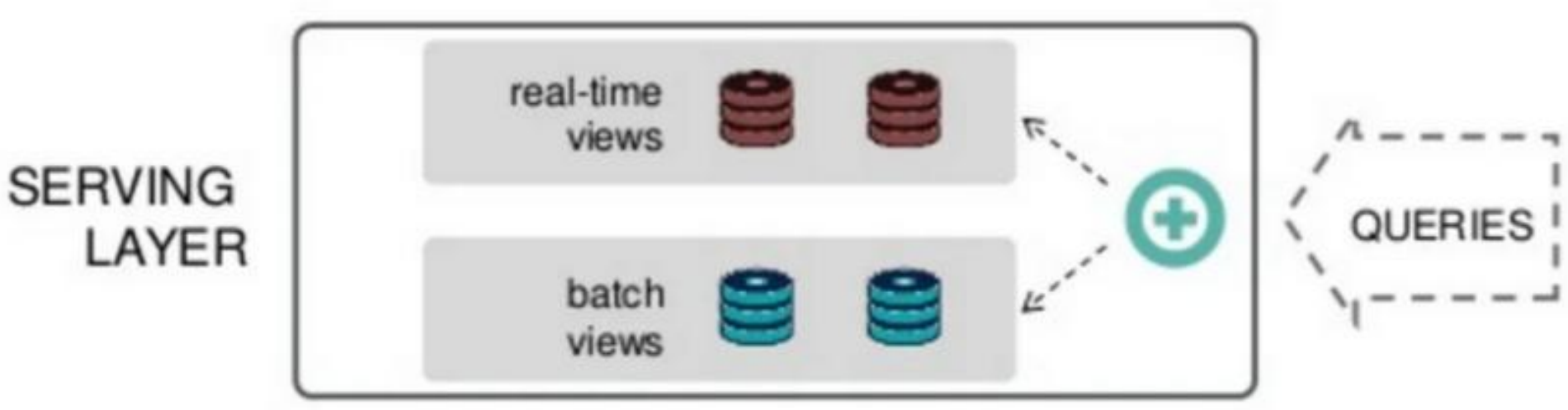
Technologies

- Cassandra
- Hbase
- Redis
- ElasticSearch

Lambda Architecture

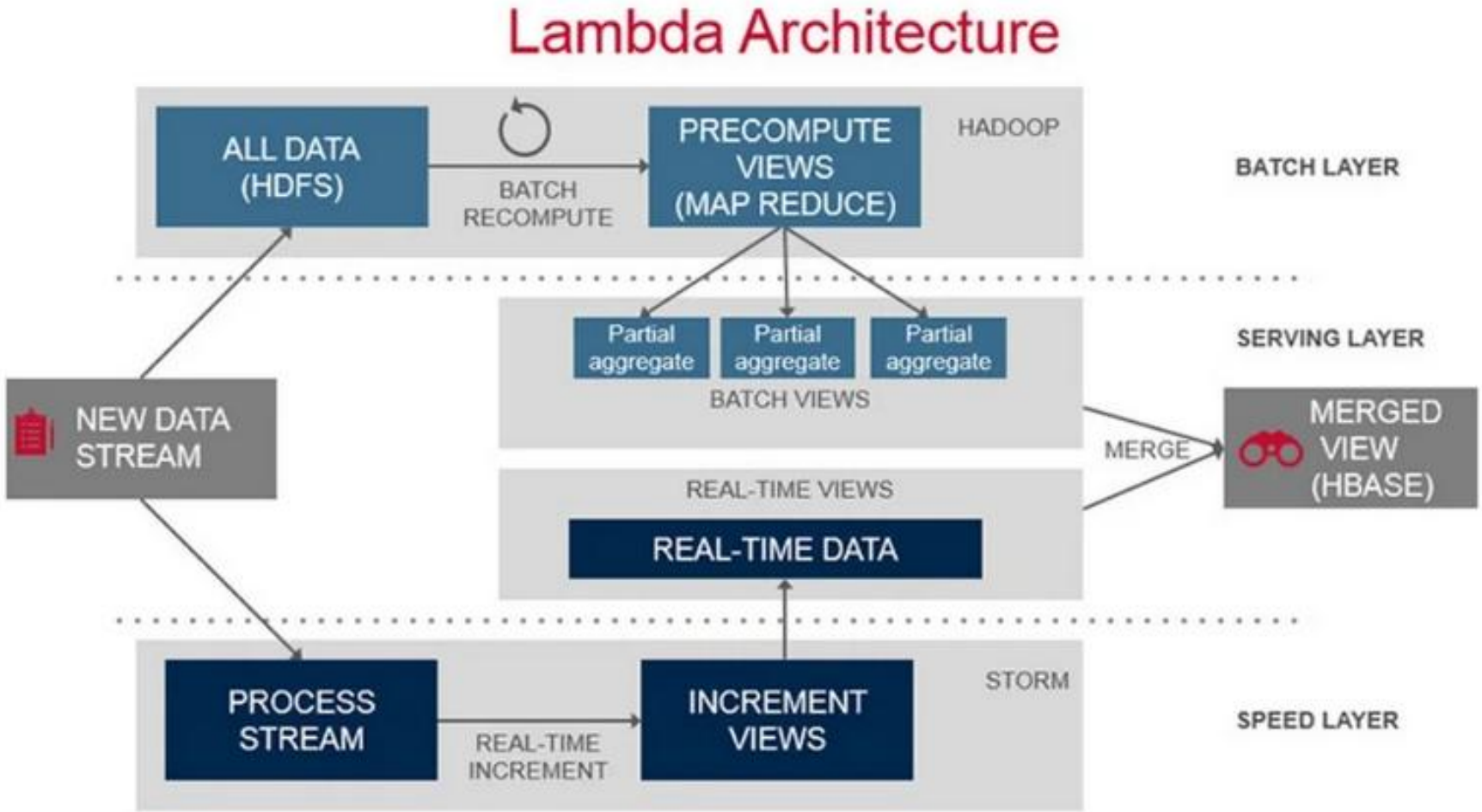
Serving Layer : fusion

- **Logique de fusion développée selon les vues et leur modélisation**
- **Peut s'avérer difficile :**
  - Expiration des vues
  - Recouvrement (intersection) possible entre les données batch et temps réel



Lambda Architecture

Proposition d'implémentation (MapR)



Lambda Architecture

Caractéristiques

+ Avantages

- Données de bases immuables et conservées en entier
- Grande tolérance aux fautes contre les pannes matérielles et les erreurs humaines
- Supporter une variété de cas d'utilisation qui incluent un requêtage à faible latence tout autant que des mises à jour
- Capacité de « scaling out » linéaire
- Facilement extensible

+ Inconvénients

- Coûteuse en terme de mise en place et expertise

Autres architectures

✚ Encore des lettres grecques …

- **Kappa**

https://www.kai-waehner.de/blog/2021/09/23/real-time-kappa-architecture-mainstream-replacing-batch-lambda

- **Zeta**

https://www.oreilly.com/content/zeta-architecture-hexagon-is-the-new-circle

La Rochelle Université

Bases de données NOSQL

Bases de données NOSQL

Définition

- Bases de données NOSQL (Not Only SQL)
  - Ce n'est pas No SQL (comme le nom laisse supposer)
- Bases de données non-relationnelles et largement distribuées
- Analyse et organisation rapides et ad-hoc des données de très grands volumes et de types de données disparates
- Appelées également
  - Cloud databases
  - Non-relational databases
  - Big data databases
  - …
- Développées en réponse à l'augmentation exponentielle des données générées, enregistrées et analysées par les utilisateurs modernes et leurs applications

Bases de données NOSQL

Atouts

Principaux atouts

- Évolutivité
- Disponibilité
- Tolérance de fautes

Caractéristiques

- Modèle de données sans schéma
- Architecture distribuée
- Utilisation de langages et interfaces qui ne sont pas uniquement du SQL

NOSQL et Big Data

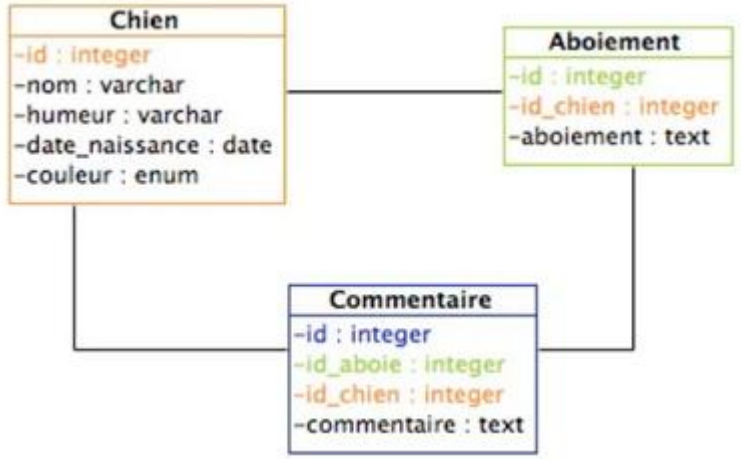
- De point de vue métier, utiliser un environnement BigData et NOSQL fournit un avantage compétitif certain

Bases de données NOSQL

Types

Types des BD NOSQL

- Clef/valeur
- Orientées documents
- Orientées colonnes
- Orientées graphes



Propriétés des BDR

- ACID : *atomicity, consistency, isolation and durability*
- Utilisation de SQL

Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL

Bases de données NOSQL

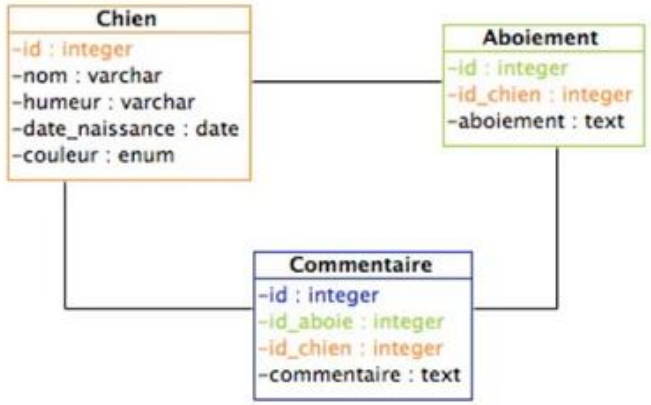
Clef/valeur (key/value store)

Principales opérations

- PUT, GET et DELETE

Exemples

- DynamoDB (Amazon), Azure Table Storage(ATS), Redis, Voldemort (Linkedin)

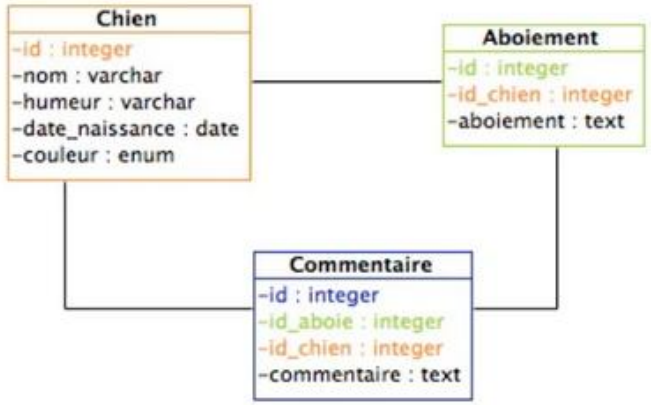


Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL

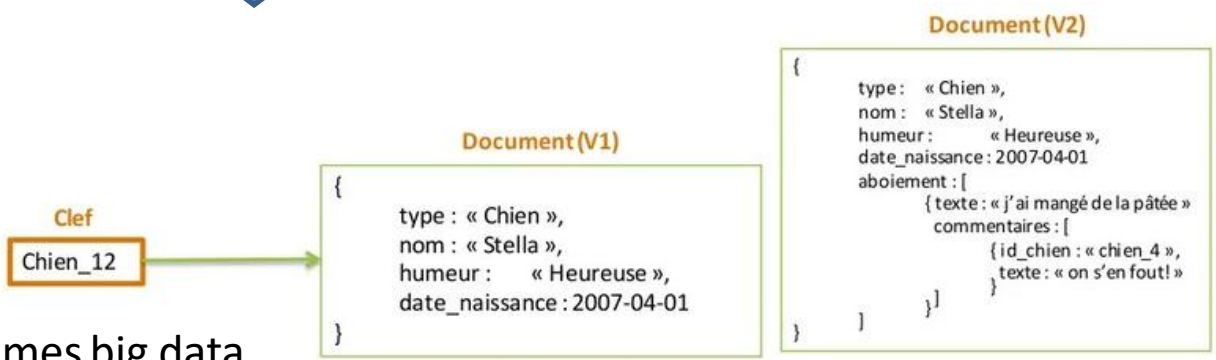
Bases de données NOSQL

Orientées documents

- Types de documents
  - JSON et XML
- Exemples
  - MongoDB (SourceForge), CouchDB (Apache) RavenDB (.NET)



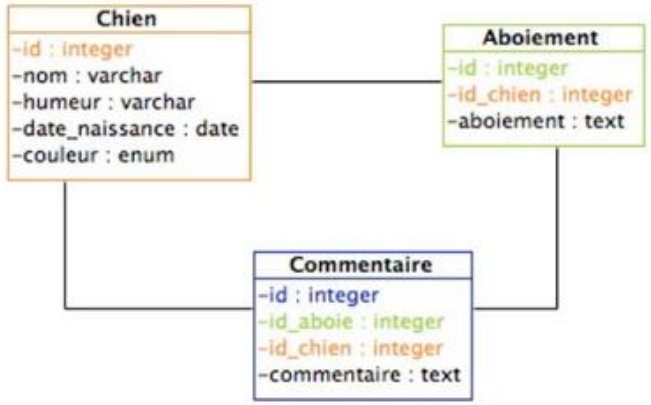
Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL



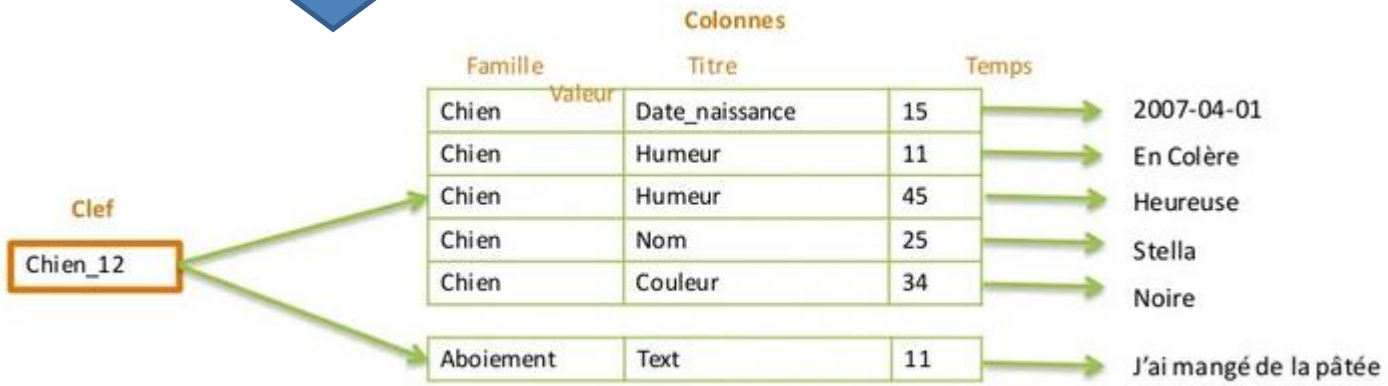
Bases de données NOSQL

Orientées colonnes

- Clef/Valeur évoluée
- Exemples
  - Hbase (Hadoop), Cassandra (Facebook, Twitter), BigTable (Google)

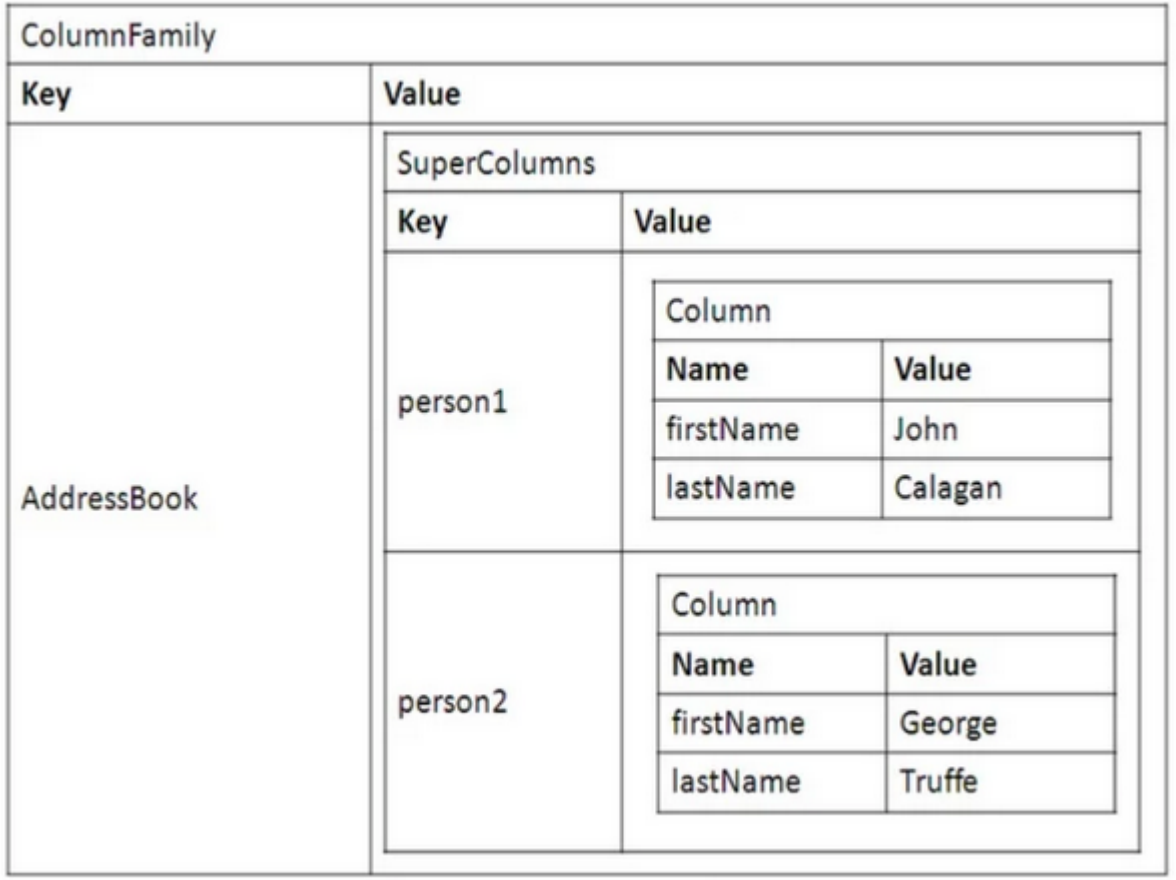


Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL



Bases de données NOSQL

Orientées colonnes



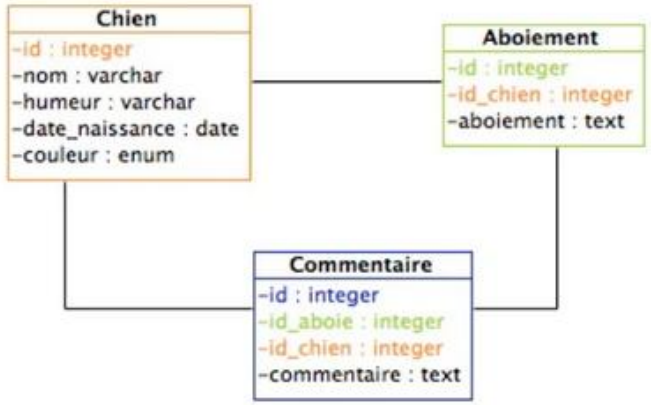
Exemple Cassandra : Colonne / Super Colonne / Famille de Colonnes

Bases de données NOSQL

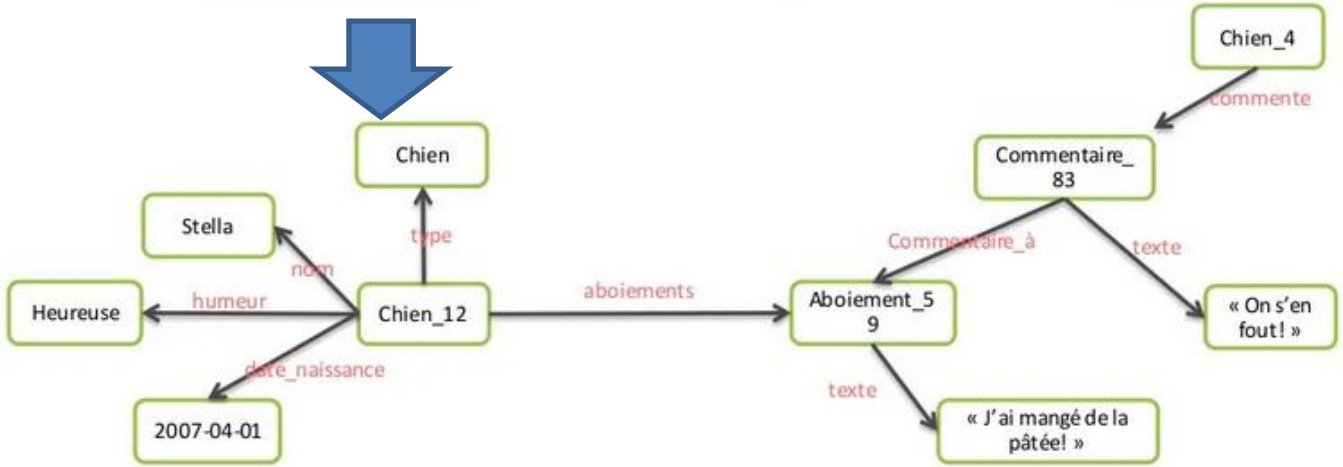
Orientées graphes

- Ensemble d'éléments interconnectées (nœuds) avec un nombre indéterminée de relations.

- Exemples
  - OrientDB



Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL



Bases de données NOSQL

Comparaison

Type BD	Performance	Évolutivité	Flexibilité	Complexité	Fonctionnalité
Clef/valeur	☺	☺	☺	☺	Variable
Orientée colonnes	☺	☺	☹	☹	Minimale
Orientée document	☺	☹	☺	☹	Variable
Orientée graphes	☹	☹	☺	☹ ☹	Théorie des graphes

Bases de données NOSQL

Propriétés ACID

- Propriétés ACID : atomicité, cohérence, isolation, durabilité
  - **Atomicité** : soit toutes les instructions sont exécutées, soit aucune
  - **Consistance** : toute transaction amène la BD d'un état valide à un autre
  - **Isolation** : une transaction ne devrait pas voir les effets des autres transactions concurrentes
  - **Durabilité** : les changements sont persistents

- Toutes les BDR supportent les transactions ACID

- MAIS :
  - Données de plus en plus volumineuses
  - Besoin de systèmes évolutifs
  - Besoin de BD distribuées

Bases de données NOSQL

Théorème CAP

- Destiné à évalué les systèmes de stockage distribués
- Propriétés CAP : consistency, availibility, partition tolerance
  - **Cohérence** : si j'écris une donnée dans un nœud et que je la lis à partir d'un autre nœud dans un système distribué, je retrouve ce que j'ai écris sur le premier
  - **Disponibilité** : à tout moment, pour chaque requête, la réponse est garantie. Même en cas de panne, les données restent accessibles
  - **Tolérance au partitionnement** : les données peuvent être partitionnées sur différents supports sans souci de localisation. Les activités continuent sans interruption lors de la modification du système (en cas d'ajout/suppression d'un nœud et en cas de chute du réseau)
- **Théorème CAP** : il est impossible de satisfaire les trois propriétés CAP en même temps

Bases de données NOSQL

Théorème CAP

- Soit un système distribué et on est entrain de modifier une donnée sur le nœud N1 et d'essayer de la lire à partir du nœud N2
  - N2 peut retourner la dernière bonne valeur dont il dispose ➔ **pas de cohérence**
  - N2 attend que la nouvelle valeur lui parvienne. La probabilité d'échec de transmission est élevée dans les systèmes distribués ce qui conduit éventuellement à une attente infinie ➔ **pas de disponibilité**
  - Satisfaire la cohérence et la disponibilité n'est pas possible avec un système de stockage partitionné ➔ **pas de tolérance au partitionnement**

- Pas de jointure dans les bases NOSQL, il n'y a plus de jointures

  ➔ La cohérence n'est plus assurée de la même manière

- Cohérence de NOSQL: cohérence immédiate et éventuelle des données à travers les nœuds de la base de la base distribué

Bases de données NOSQL

Propriétés BASE

- BASE : Basically Available, Soft-state, Eventually consistency
  - **Basically available** : le système garantie la disponibilité
  - **Soft-state** : l'état du système peut changer dans le temps, même sans nouvelles entrées
  - **Eventual consistency** : les modifications arriveront éventuellement à tous les serveurs, si on leur donne suffisamment de temps

- BASE est plus flexible que ACID, accepte certaines erreurs dont celles avec des occurrences rares

Bases de données NOSQL

Récapitulatif

BDR	NOSQL
Forte cohérence	Éventuelle cohérence
Grande quantité de données	Énorme quantité de données
Possible évolutivité	Facile évolutivité
Bonne disponibilité	Très haute disponibilité
SQL	MAP-Reduce

La Rochelle Université

CASSANDRA

Cassandra

présentation

- Base de données :
  - Distribuée
  - À haute performance
  - Extrêmement évolutive
  - Tolérante aux fautes
  - Non-relationnelle

- A la base, c'est une combinaison de BigTable de Google et DynamoDB de Amazon, incubée dans Facebook, et hébergée comme solution open source dans Apache

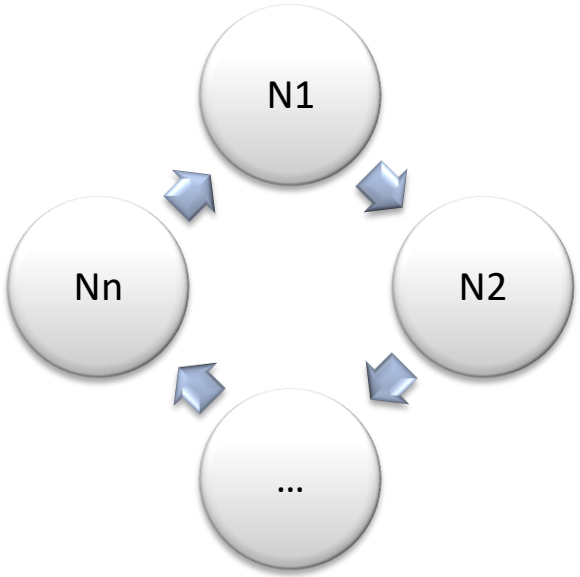
Cassandra

Architecture

- Système distribué composé de plusieurs nœuds identiques
  - Pas de nœud maître (NameNode)
  - Les données sont partitionnées et répliquées sur les différents nœuds
  - L'utilisateur contrôle le nombre de répliques qu'il désire avoir pour ces données

- Lecture et écriture à partir de n'importe quel nœud.

- Utilisation du protocole **Gossip** pour la communication
  - Échange de données chaque seconde



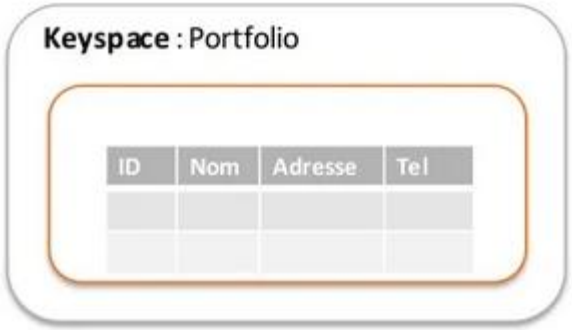
Cassandra

Architecture

+ Base orientée colonne

Vocabulaire :

- Keyspace (~database)
- Column family (~table)
  - Appelée table dans la nouvelle version du langage CQL
  - Schéma plus flexible et dynamique qu'une table
- Colonne (~enregistrement)
  - Indexée par une clé
  - D'autre champs peuvent aussi être indexée à la demande



Cassandra

Partitionnement

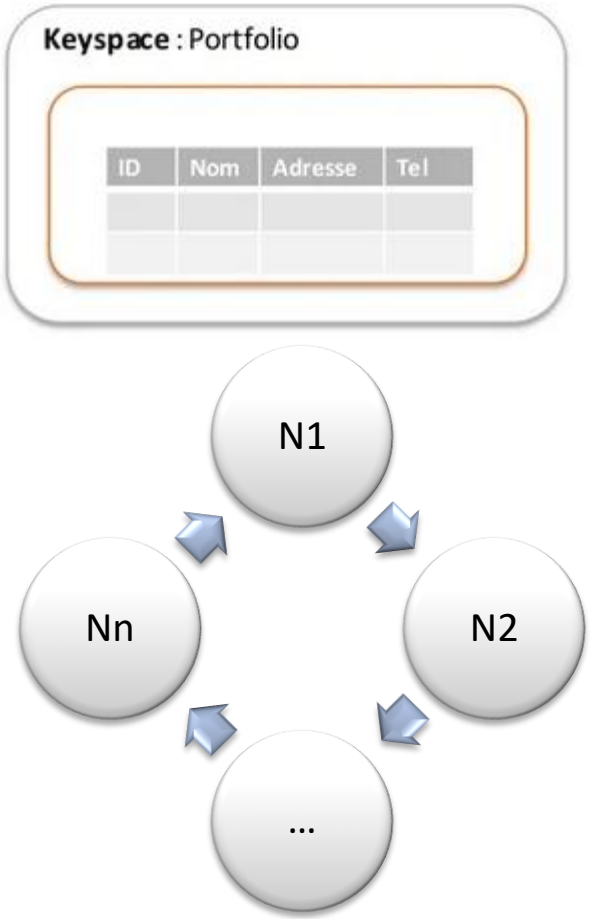
- Chaque nœud est responsable d'une partie de la base
- Les données sont insérées par l'utilsateur
- Elles sont ensuite placées dans un nœud

- Stratégie de partitionnement :
  - ❖ Aléatoire (par défaut) : recommandé

    Les données sont partitionnées le plus équitablement possible
  - ❖ Ordonnée :

    Les données sont stockées, selon les clefs de familles de colonnes, par ordre à travers les nœuds.

- Stratégie spécifiée dans un fichier cassandra.yaml
- En cas de modification de stratégie, il faut recharger toutes les données

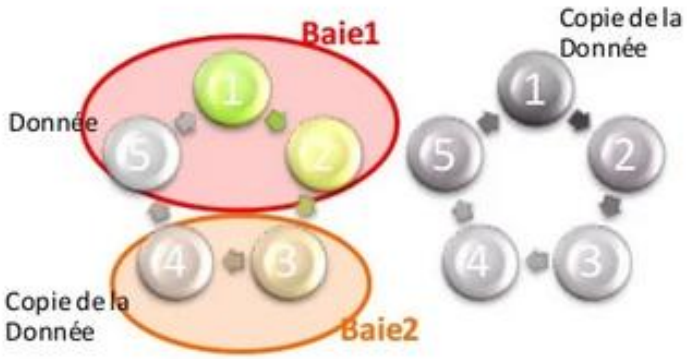


Cassandra

Réplication

- Pour assurer la tolérance aux fautes
- L'utilisateur spécifie le facteur de réplication

- Stratégie de réplication :
  - ❖ **Simple :**
    - La colonne est placée sur un nœud
    - La réplique est placé sur le nœud suivant dans l'anneau (clockwise)
  - ❖ **Par topologie de réseau :**
    - Plus sophistiquée
    - Plus de contrôle d'emplacement
    - Parcours du cluster (clockwise) à la recherche d'un nœud dans une baie différente. Si introuvable, placer la colonne dans un nœud de la même baie



Cassandra

Cohérence

- Architecture *Read and Write Anywhere*
- L'utilisateur peut se connecter à n'importe quel nœud et lire /écrire les données qu'il veut
- Les données sont automatiquement partitionnées et répliquées à travers le cluster
- Cassandra est la base de données NOSQL la plus rapide en écriture
- Extension du concept de cohérence éventuelle à une cohérence ajustable
- Choix possible entre cohérence forte ou éventuelle selon les besoins
- Ce choix est fait par opération

  *SELECT * FROM .. USING CONSISTENCY ANY WHERE ..*

Cassandra

Stratégies d'écriture

- Niveau de cohérence : combien de répliques doivent être écrites avant de retourner un acquittement au client?

- **Any** : une écriture doit réussir sur n'importe quel nœud ➔ haute disponibilité avec basse cohérence
- **One** (par défaut dans CQL) : une écriture doit réussir sur le commit log avec au moins une réplique
- **Quorum** : une écriture doit réussir sur un certain pourcentage de nœuds répliques (pourcentage = (facteur de réplication / 2) + 1) ➔ meilleure compromis de disponibilité et cohérence
- ***All*** : une écriture doit réussir sur tous les nœuds répliques d'une colonne ➔ plus haute cohérence mais la plus basse disponibilité

Cassandra

Stratégies d'écriture

- Cassandra tente de modifier une colonne sur toutes les répliques
- Si certains nœuds répliques ne sont pas disponible, un indice (hint) est sauvegardé sur l'un des nœuds répliques en marche, pour mettre à jour tous les nœuds répliques en panne une fois disponible
- Si aucun nœud réplique n'est disponible, l'utilisation de la stratégie ANY permettra au nœud coordinateur de stocker cet indice. La donnée ne sera lisible que quand l'un des nœuds est disponible de nouveau

Cassandra

Stratégies de lecture

- Niveau de cohérence : combien de répliques doivent répondre avant de retourner le résultat au client?

- **One** (par défaut dans CQL) : obtention du résultat à partir de la réplique la plus proche

➔ La plus faible cohérence et la plus haute disponibilité

- **Quorum** : obtention du résultat à partir d'un certain pourcentage de nœuds répliques (pourcentage = (facteur de réplication / 2) + 1)

➔ meilleure compromis de disponibilité et cohérence

- ***All*** : obtention du résultat le plus récent à partir de tous les nœuds répliques

➔ La plus haute cohérence et la plus faible disponibilité

Cassandra

Gestion de données/objets

- Deux interfaces pour gérer les objets/données
  - ❑ *Cassandra CLI (command Line Interface)*
  - ❑ *CQL (Cassandra Query Language)* : langage proche de SQL

- CQL
  - ❑ Les objets (keysapces, familles de colonnes et index) sont créés, modifiés et supprimés avec les requêtes respectives CREATE, ALTER et DROP.
  - ❑ Les données sont insérées, modifiées et supprimées avec les requêtes respectives INSERT, UPDATE et DELETE.
  - ❑ Les données sont lues avec SELECT
  - ❑ Le type de cohérence est défini avec USING CONSISTENCY

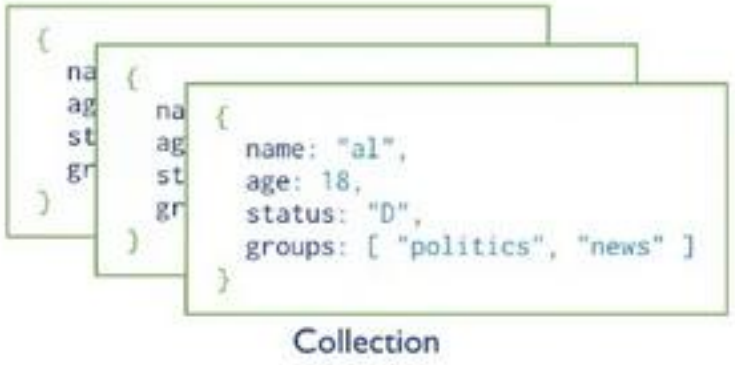
La Rochelle Université

MongoDB

MongoDB

Présentation

- SGBD NOSQL développé en 2007 et écrit en C++
- SGBD orienté document, à schéma flexible et distribuable
- La structure du document n'est pas fixé à sa création
- Les documents sont organisés sous forme de collection
- Une collection est un groupe de document reliés par des indexes en commun



MongoDB

Terminologie

MongoDB	SQL
Base de données	Base de données
Collection	Table
Document	Ligne
Champ	Colonne
Index	Index
Imbrication ou référence	Jointure
Clef primaire	Clef primaire

MongoDB

Document

- Structure de donnée JSON-like, composée de paires clef/valeur
- Stocké sur le disque sous forme de document BSON (binary json)

```
var mydoc = {
                _id: ObjectId("5099803df3f4948bd2f98391"),
                name: { first: "Alan", last: "Turing" },
                birth: new Date('Jun 23, 1912'),
                death: new Date('Jun 07, 1954'),
                contribs: [ "Turing machine", "Turing test", "Turingery" ],
                views : NumberLong(1250000)
              }
```

MongoDB

Document

- Le champ _id : seul champ obligatoire, utilisé comme clef primaire dans une collection
- Le champ _id peut être de tout type autre que tableau

```
var mydoc = {
                _id: ObjectId("5099803df3f4948bd2f98391"),
                name: { first: "Alan", last: "Turing" },
                birth: new Date('Jun 23, 1912'),
                death: new Date('Jun 07, 1954'),
                contribs: [ "Turing machine", "Turing test", "Turingery" ],
                views : NumberLong(1250000)
              }
```

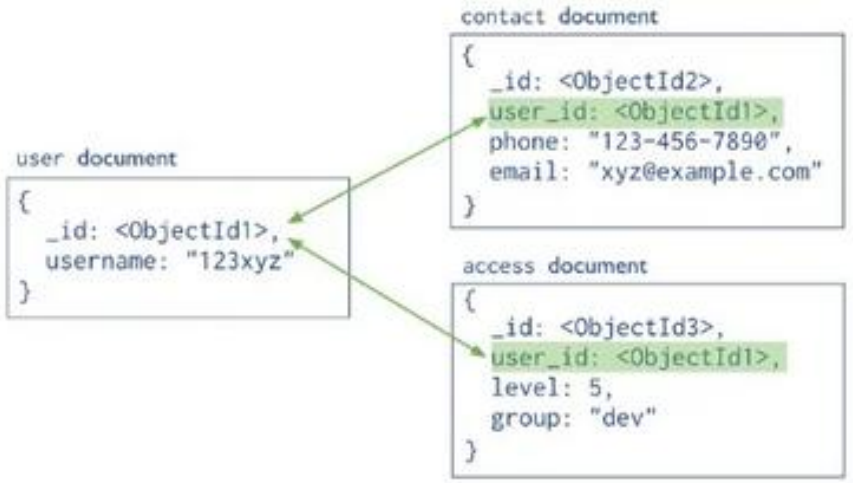
MongoDB

Structure de document

- Deux manières de relations entre les données

Références

Inclusion de liens de références d'un document à un autre



Imbrication

Inclure des documents dans un champ ou un tableau

MongoDB

Gestion de données/objets

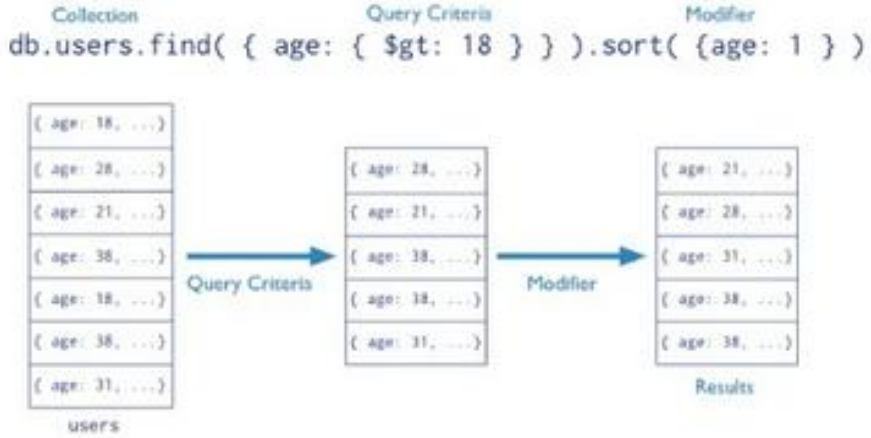
- On choisit les références quand :
  - L'imbrication va produire des données dupliquées sans grands avantages en terme de performance de lecture
  - On veut représenter des relations many-to-many complexes
  - On veut modéliser de larges ensembles de données hiérarchiques

- On choisit les données imbriquées quand :
  - Présence de relations **contains** entre les éléments
  - Présence des relations one-to-many où les documents fils (many) apparaissent toujours dans le contexte de documents parents (one)

MongoDB

Opération de lecture

- Cible une unique collection (spécifique) de documents



- Fournit une méthode db.collection.find() pour l'extraction de données. La méthode db.collection.findOne() retourne un seul document

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

Équivalent SQL

```
SELECT _id, name, address   ← projection
FROM   users                ← table
WHERE  age > 18             ← select criteria
LIMIT  5                    ← cursor modifier
```

MongoDB

Opération d'écriture

- **L'insertion** de document sans champ _id est possible, le système l'ajoute en générant le champ de type ObjectId

```
db.users.insert (          ← collection
   {
     name: "sue",          ← field: value
     age: 26,              ← field: value
     status: "A"           ← field: value
   }
)
```

document

- **La mise à jour** d'un document s'effectue par la méthode update. Plusieurs documents peuvent être modifiés si multi:true.

```
db.users.update(                  ← collection
   { age: { $gt: 18 } },          ← update criteria
   { $set: { status: "A" } },     ← update action
   { multi: true }                ← update option
)
```

- La méthode remove **supprime** des documents répondant à des critères

```
db.users.remove(          ← collection
   { status: "D" }        ← remove criteria
)
```

D’ici, on voit + loin !

La Rochelle Université

univ-larochelle.fr