Master 1

FRAMEWORK LOGICIEL POUR LE BIG DATA

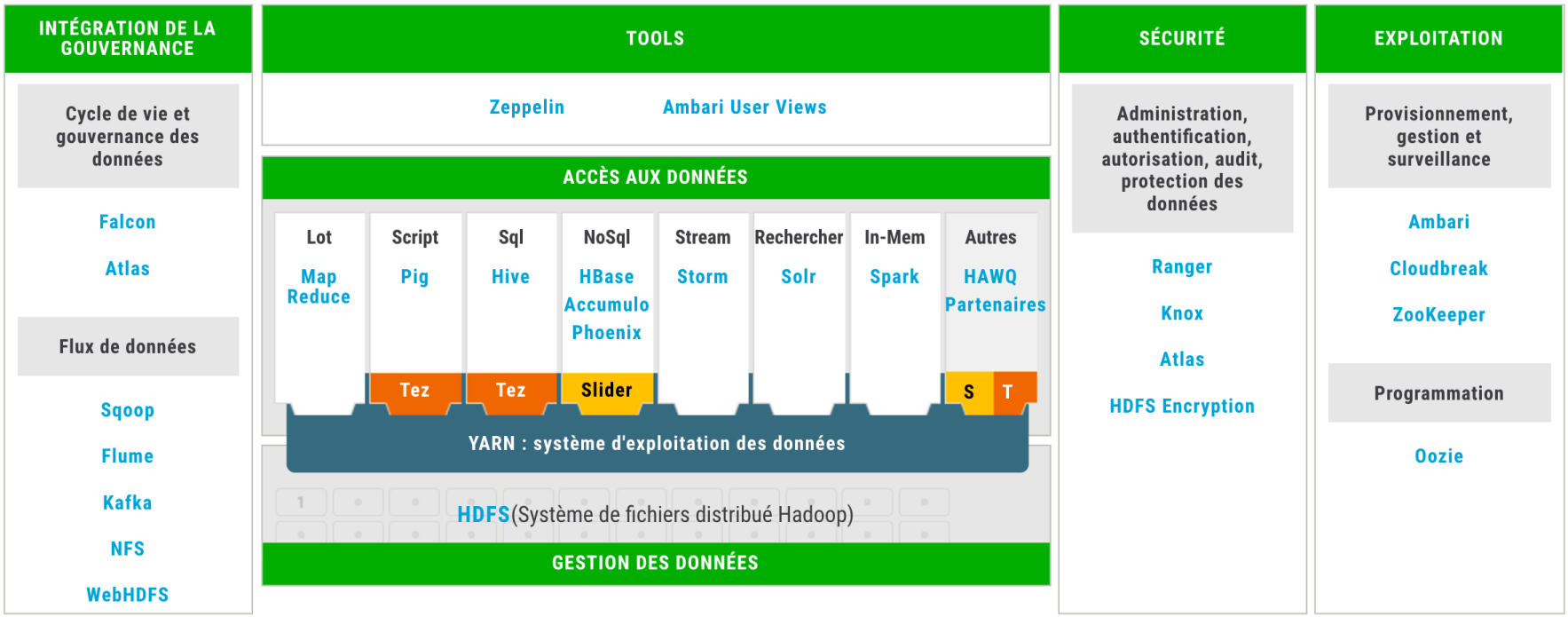
COURS 3

R.JAZIRI

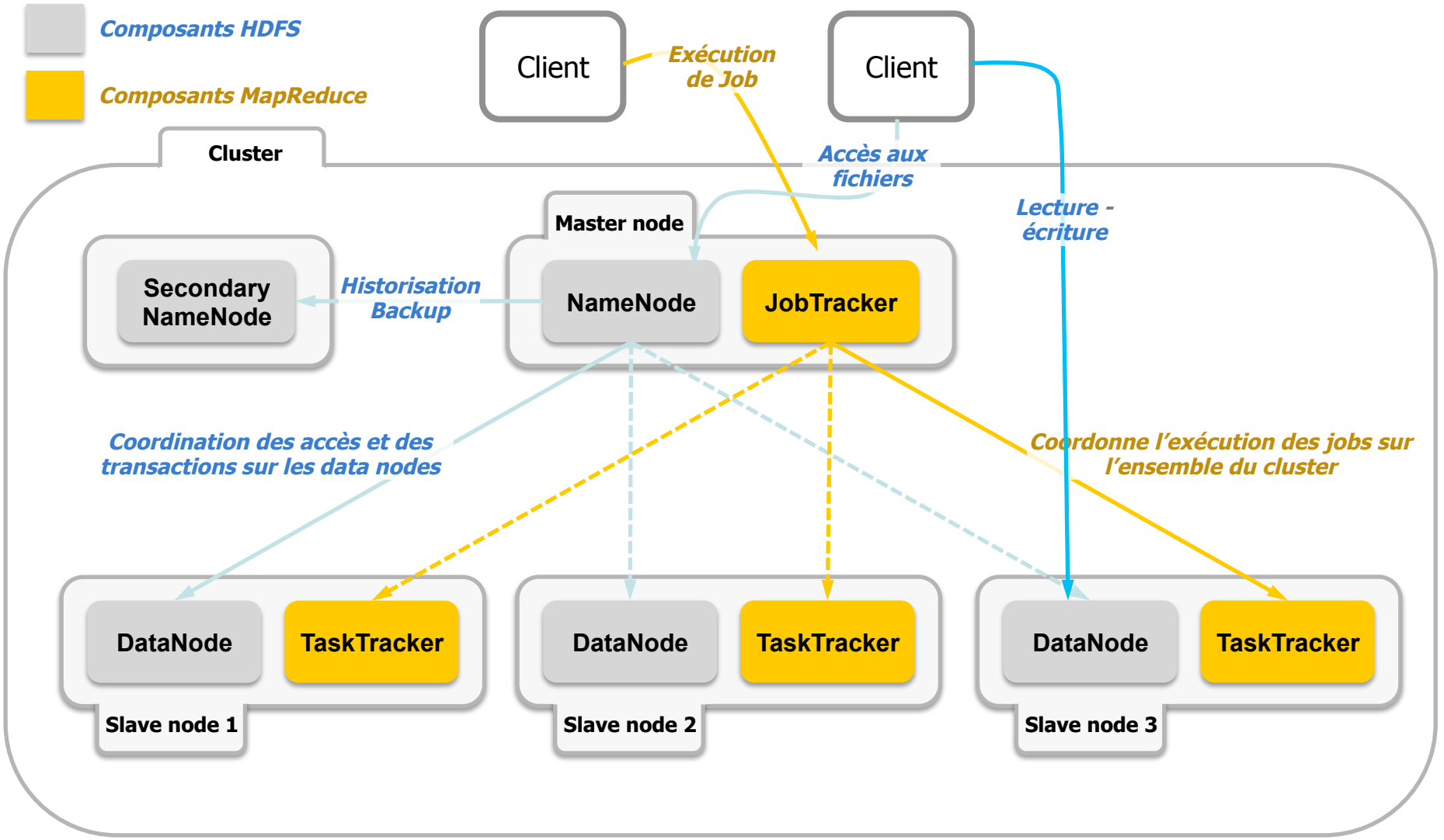
PLAN DU COURS

- Big data, les fondamentaux
- Introduction aux fondements technologiques (Hadoop, HDFS, MapReduce, …),
- Yarn
- …

ECOSYSTÈME HADOOP



Parallèlisation via architecture distribuée
Organisation de la plateforme



HDFS + MapReduce

- L'entrée de MapReduce doit être fait à partir de fichiers dans HDFS
  - Chaque bloc contient une liste de pairs clé-valeur
- Les tâches **Map** sont préférentiellement assignées aux nœuds contenant un block
  - L'entrée des tâches Map sont des fichiers locaux, tout comme la sortie
  - Les résultats des maps seront groupés : un groupe par reducer
  - Chaque groupe est ordonnée (sorted)
- Les tâches **Reduce** sont assignées à un ou plusieurs nœuds
  - Les tâches Reduce récupèrent ses entrées à partir de tous les nœuds Map associées à son groupe
  - Le résultat est calculé et stocké dans un fichier HDFS
- La sortie est un ensemble de fichiers dans HDFS (un fichier par reducer)

La partie Map

- L'entrée de Map est un ensemble de mots {w} d'une partition du texte
  - Clé=w Valeur=null
- La fonction Map calcule
  - Le nombre de fois qu'une clé w apparaît dans la partition
- La sortie de la fonction map (pour une machine) est une liste sous la forme
  - <w, nombre de mots>
- Si nous avons plusieurs machines calculant Map, la sortie "finale" est plutôt une liste d'entrées
  - {<w, {valeur1, valeur2, ...}>}

La partie Reduce

- L'entrée de reduce correspond à la sortie de la fonction map
  - {<clé, {valeur}>}, où
    - cle= “mot”
    - Chaque valeur est un entier
- La fonction Reduce calcule
  - Le nombre total d'occurrences d'un mot k :
    - La somme de toutes les valeurs avec la clé k
- La sortie de la fonction Reduce
  - <clé, N>

L'API MapReduce en Java

- Chaque programme MapReduce doit spécifier un Mapper et un Reducer (pas obligatoire)
  - Le Mapper a une méthode **map** qui transforme une entrée (clé, valeur) en un nombre arbitraire de paires intermédiaires (clé', valeur')
    - Par défaut, une instance Mapper sera lancée par fichier d'entrée
  - Le Reducer a une méthode **reduce** qui transforme les clés intermédiaires (clé', valeur'*) et les agrège dans un nombre arbitraire de paires de sortie (clé", valeur")
    - Par défaut, un reducer par noeud

L'exemple du WordCount

- Presque tous les tutoriaux Hadoop utilisent l'exemple du WordCount
  - Simple pour comprendre
  - Pas vraiment intéressant (et très lent)
- Entrée : un grand fichier texte
- Sortie : le nombre d'occurences de chaque mot dans le fichier
- Exemple :
  - Pour le texte "Robur the Coqueror" de Jules Verne, nous avons

11 fois le mot "corpuscles", une fois le mot "susceptible", 5 fois "clear", etc.

WordCount – comment penser en MR

- Dans un code séquentiel, WordCount peut être implémenté avec un parseur String et un HashList <k, V> où :
  - La clé est le "mot"
  - La valeur est un entier incrémenté à chaque fois que ce mo apparaît
- Même dans un environnement distribué, la procédure est presque la même :
  - Etape 1 : chaque machine parse une partie du fichier et enregistre le résultat partiel
  - Etape 2 : à la fin, nous faisons la somme des valeurs associées à chaque clé

LES DISTRIBUTIONS

- Le seul fournisseur qui utilise 100% du projet open source Apache Hadoop sans ses propres (non-open) modifications.
- Il est le premier vendeur à utiliser les fonctionnalités Apache HCatalog pour des services de méta-données.
- Par ailleurs, leur initiative Stinger optimises massivement le projet Hive.
- Il offre un très bon bac à sable, facile à utiliser pour commencer.
- Il a développé et committé des améliorations au niveau du coeur qui rend Apache Hadoop exécutable nativement sur les plate-formes Microsoft Windows incluant Windows Server et Windows Azure.

LES DISTRIBUTIONS

- La distribution de loin la plus installée avec le plus grand nombre de déploiements référencés.
- Un outillage puissant pour le déploiement, la gestion et le suivi est disponible.
- Impala est développée par Cloudera pour offrir des traitements temps réel de big

LES DISTRIBUTIONS

- Utilise quelques concepts différents de ses concurrents, en particulier du support pour un système de fichier Unix natif au lieu de HDFS (avec des composants non open source) pour une meilleure performance et une facilité d'utilisation.
- Les commandes natives Unix peuvent être utilisées à la place des commandes Hadoop.
- Il se différencie de ses concurrents avec des fonctionnalités de haute disponibilité comme les snapshots, la réplication ou encore le basculement avec état ("stateful failover").
- L'entreprise est aussi à la tête du projet Apache Drill, un projet open source réinventé à partir de Dremel de Google pour des requêtes de type SQL sur des données Hadoop afin d'offrir des traitements temps réels.

LES DISTRIBUTIONS

- Il diffère des autres en tant que solution hébergée sur l'infrastructure web scalable d'Amazon Elastic Compute Cloud (Amazon EC2) et d'Amazon Simple Storage Service (Amazon S3).
- Outre la distribution d'Amazon, vous pouvez aussi utiliser MapR sur EMR. Un use case majeur est les clusters éphémères.
- Si vous avez besoin de traitements de big data ponctuels ou peu fréquents, ERM peut vous éviter de perdre beaucoup d'argent.
- De l'écosystème Hadoop seul Pig et Hive sont inclus.
- Il est mis au point pour travailler avec les données dans S3, qui a une latence plus élevée et ne place pas les données sur vos noeuds de calcul.
- Les Entrées/Sorties de fichiers sur ERM sont plus lents et plus latents que des E/S sur votre propre cluster Hadoop ou EC2

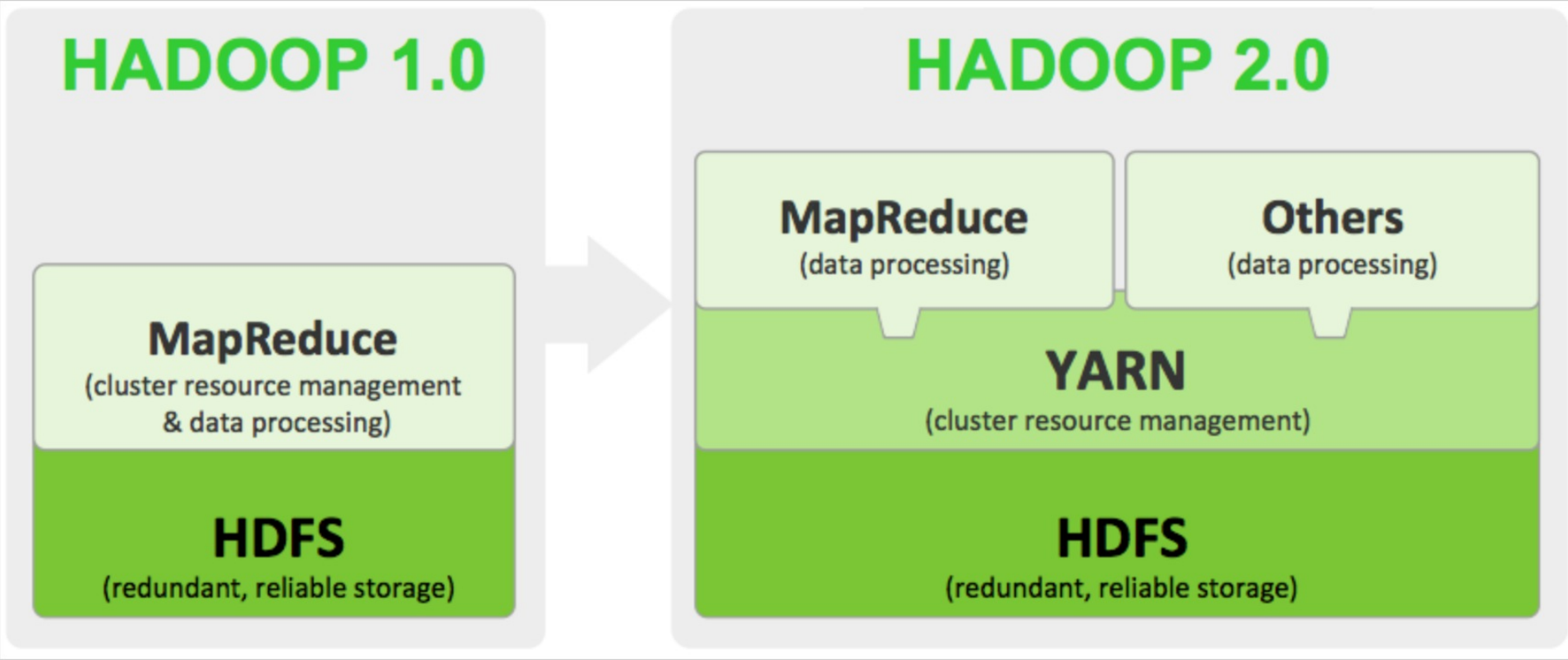
Modes d'Exécution Hadoop

- Hadoop propose trois modes d'exécution :
  - Local (standalone)
    - Bon pour des petits tests et debugage
  - Local (pseudo-distributed)
    - Les tâches sont lancées dans différentes VM Java
    - La configuration est presque comme celle d'un cluster
  - Fully-distributed (cluster)
    - Toutes les machines du cluster sont configurées avec Hadoop
    - Une topologie maître-esclave est mise en place

Hadoop 1 vs Hadoop 2

- Jusqu'à 2012, Hadoop reposait entièrement sur le couplage MapReduce-HDFS
  - Ce sont les versions 0.2x et 1.x
- EN 2012 on a vu l'arrivée de Hadoop 2.x, qui déconnecte HDFS de MapReduce
  - Possibilité d'exécuter des applications autres que MapReduce
  - Possible grâce au nouveau mécanisme d'ordonnancement YARN
- Ce qui change
  - Une redistribution des rôles (pas très compliqué)
  - Une gestion des ressources plus structurée
  - Une API légèrement plus souple pour la programmation MapReduce
    - On garde l'ancienne API quand même
  - La réplication des services clés (Haute Disponibilité)

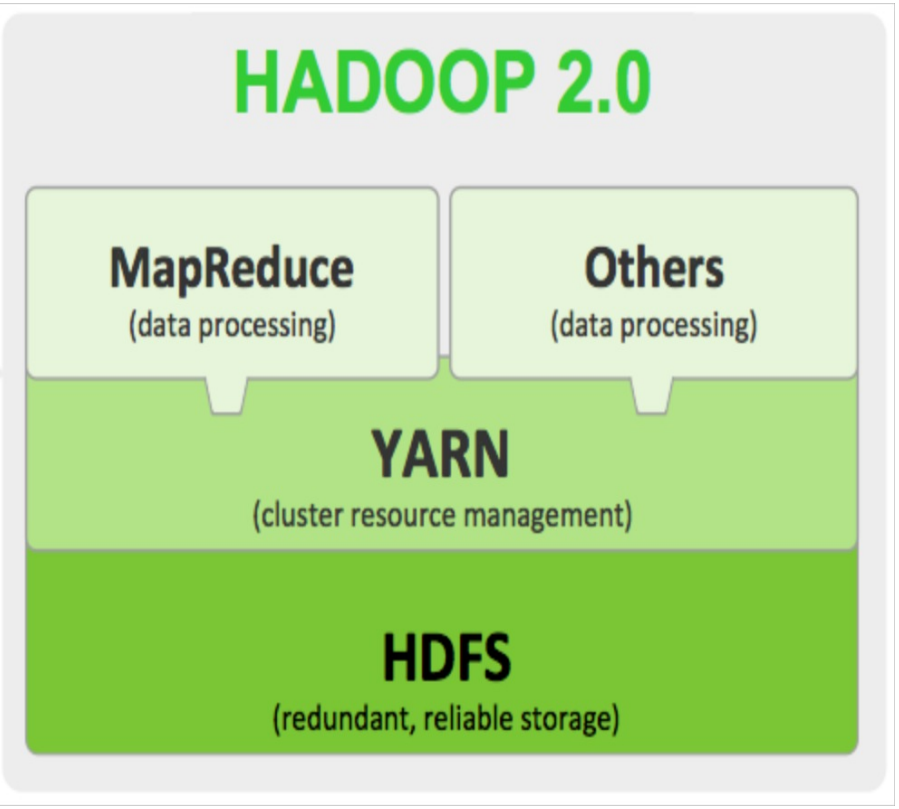
Projet



Yarn apporte une séparation claire entre les problématiques suivantes :

- Gestion de l'état du cluster et des ressources.
- Gestion de l'exécution des jobs.

HADOOP 2.x



La nouvelle version d'Hadoop apporte donc une évolution architecturale majeure : **YARN**

YARN permet de décharger ***le JobTracker*** qui avait tendance à cumuler trop de rôles et devenait donc complexe à maintenir et à faire scaler.

Cette remise à plat des rôles à permis aussi de découpler Hadoop de Map Reduce et de permettre à des frameworks alternatifs d'être portés directement sur Hadoop et HDFS **et non au dessus de Map Reduce.**

Cela va donc permettre à Hadoop, outre une meilleure scalabilité, de s'enrichir de nouveaux frameworks couvrant des besoins peu ou pas couverts avec Map Reduce.

HADOOP 2.x_MAP REDUCE

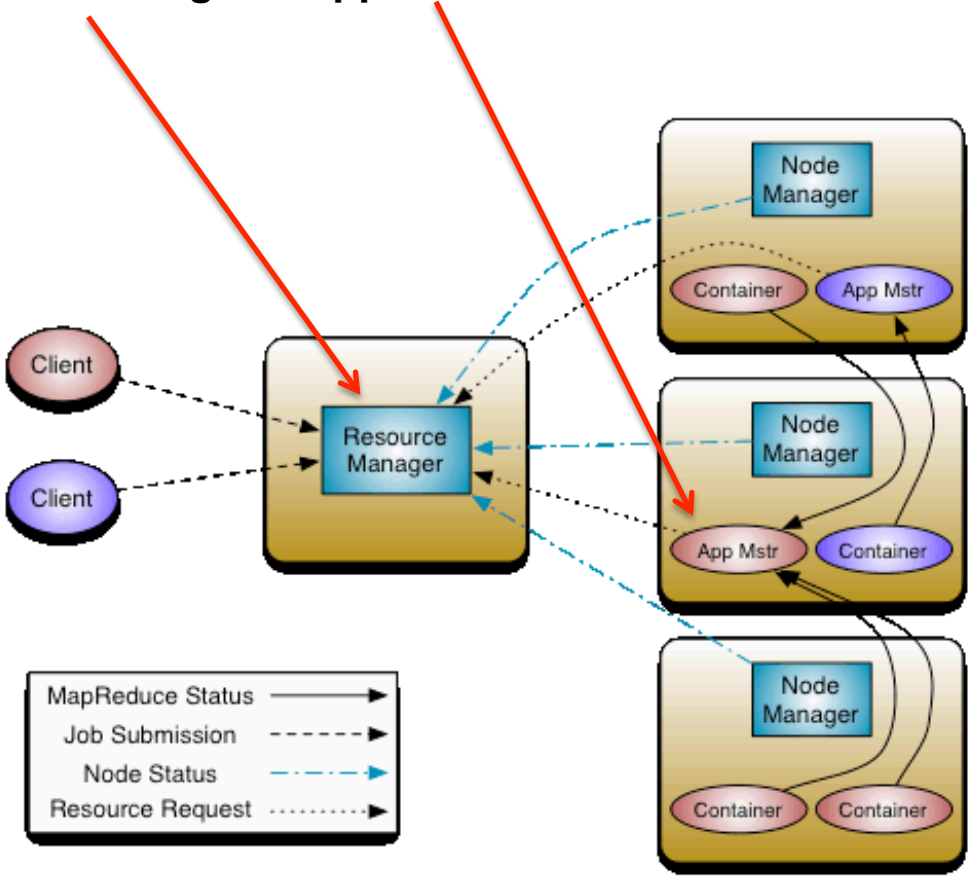
JobTracker = ResourceManager + ApplicationMaster.

Le ***jobtracker*** a été disparu de l'architecture ou plutôt ses rôles ont été répartis différemment.

L'architecture est maintenant organisée comme suit :

1. ***ResourceManager*** dont le périmètre d'action est global au cluster
2. ***ApplicationMaster*** locaux dont le périmètre est celui d'un job ou d'un groupe de jobs.

La différence, de part le découplage, se trouve dans la multiplicité. En effet, Un ResourceManager gère n ApplicationMaster, lesquels gèrent chacun n jobs.

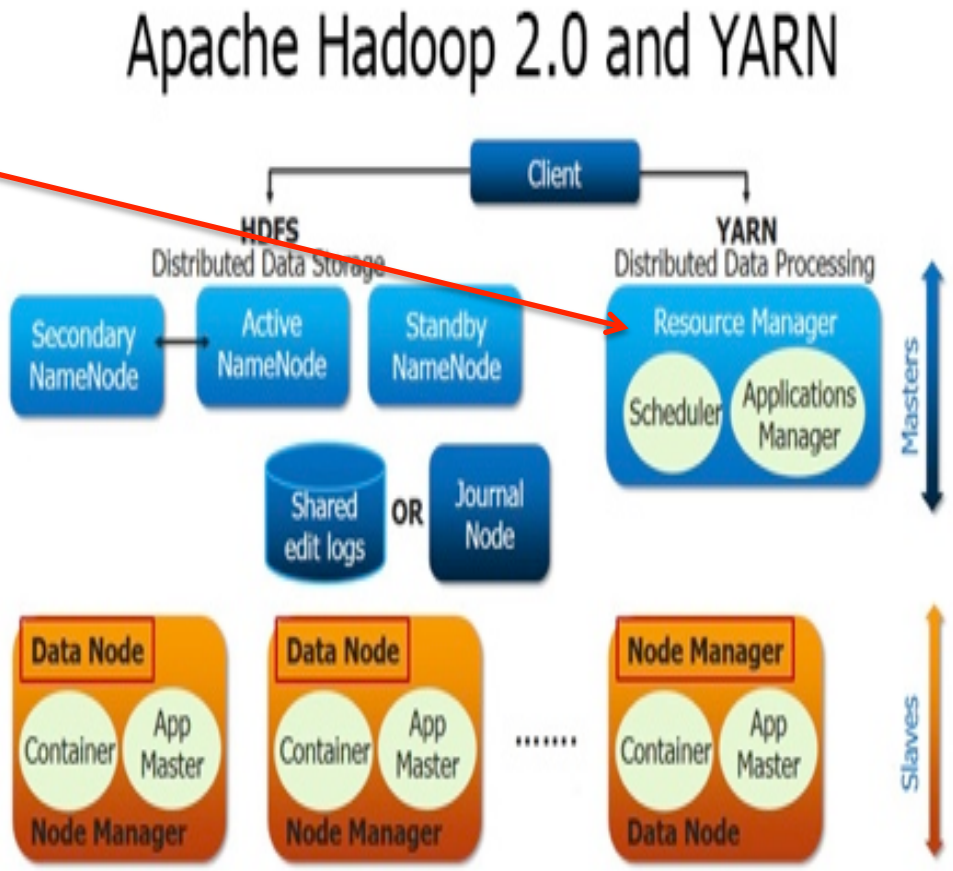


HADOOP 2.x_MAP REDUCE

1. Le ***ResourceManager*** est le remplaçant du ***JobTracker*** du point de vue du client qui soumet des jobs à un cluster Hadoop.

Il n'a maintenant plus que deux tâches à accomplir :

   a. **Scheduler**
   b. **ApplicationsManager**



HADOOP 2.x_MAP REDUCE

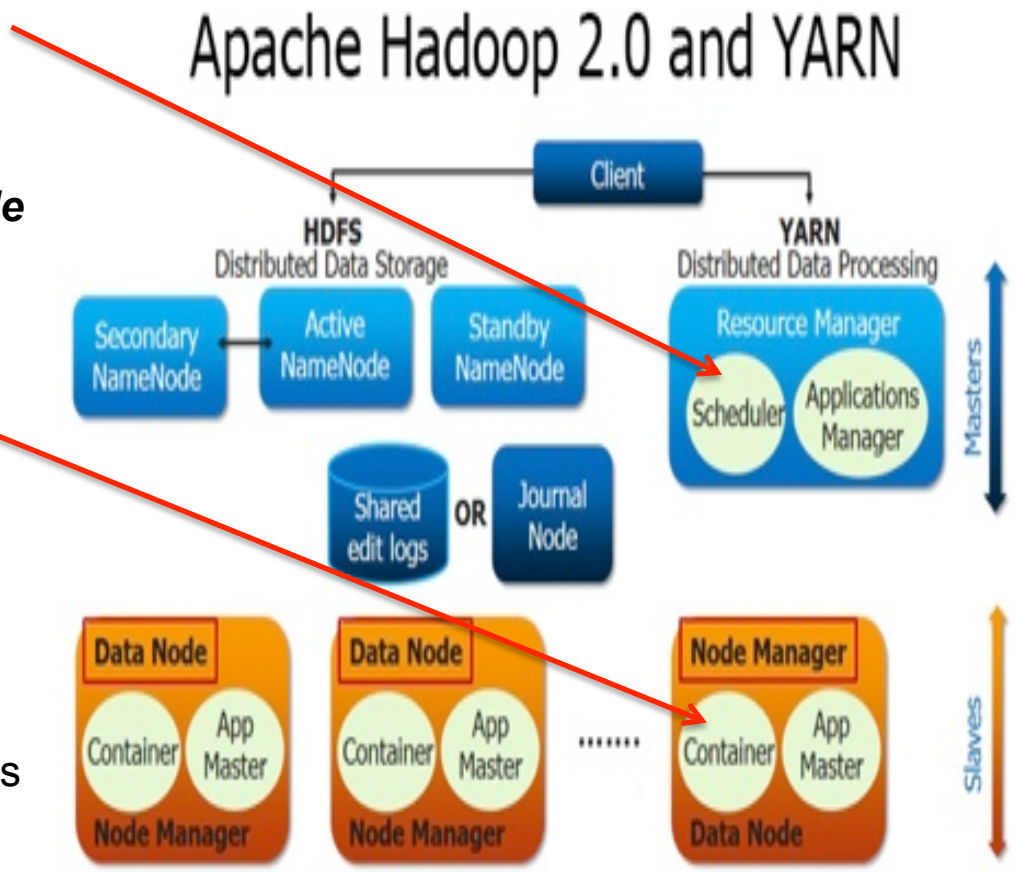
a. **Le *Scheduler*** est responsable de l'allocation des ressources des applications tournant sur le cluster.

Les resources allouées aux applications par ***le Scheduler*** pour leur permettre de s'exécuter sont appelées des ***Containers***.

Un ***Container*** désigne un regroupement de mémoire, de cpu, d'espace disque, de bande passante réseau, …

Ces informations sont remontées du cluster par les ***NodeManager***, qui sont des agents tournant sur chaque noeud et tenant le ***Scheduler*** au fait de l'évolution des resources disponibles.

Scheduler peut ainsi prendre ses décisions d'allocation des ***Containers*** en prenant en compte des demandes de ressources cpu, disque, réseau, mémoire, …

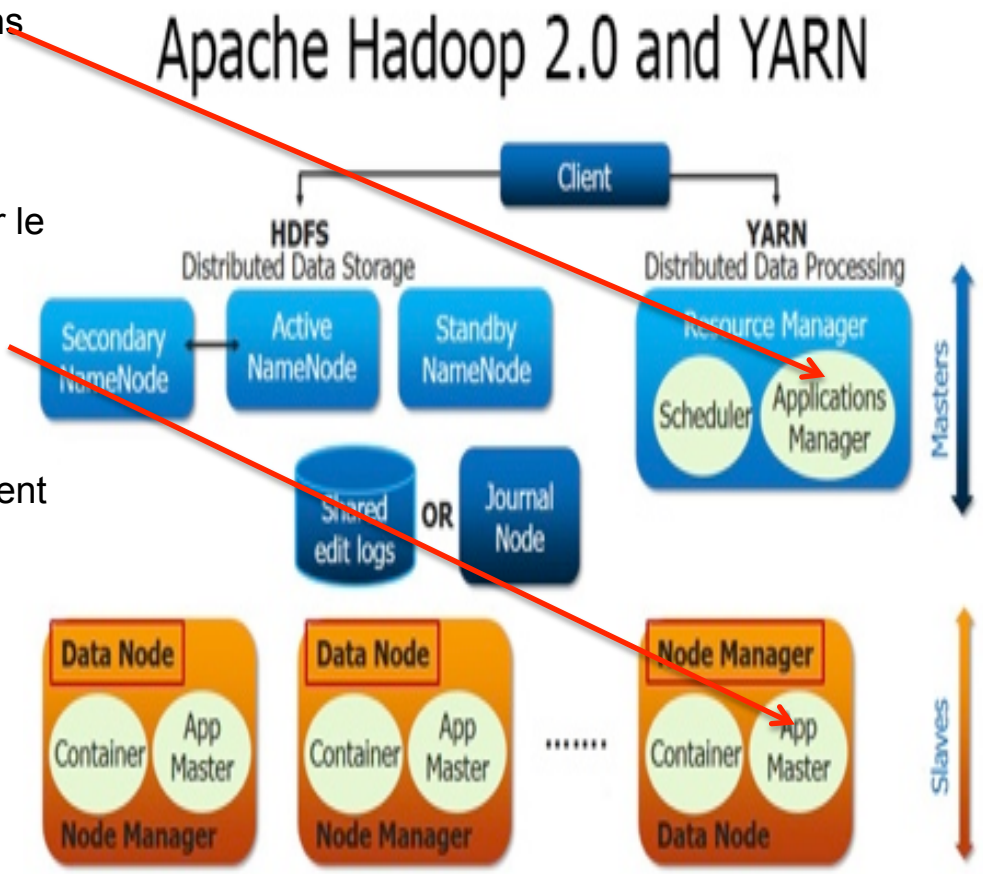


HADOOP 2.x_MAP REDUCE

b. L'ApplicationsManager: accepte les soumissions d'applications.

L'***ApplicationsManager*** ne s'occupe que de négocier le premier *Container* que le Scheduler allouera sur un noeud du cluster.

La particularité de ce premier *Container* est qu'il contient **l'*ApplicationMaster*.**

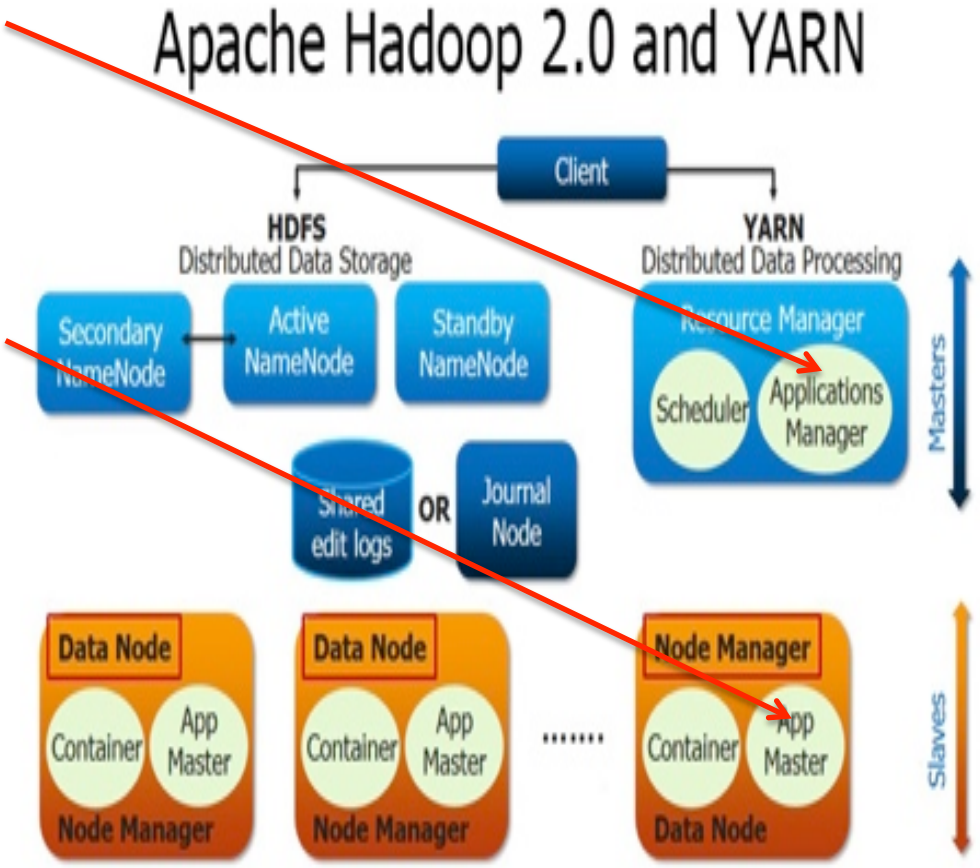


HADOOP 2.x_MAP REDUCE

2. L'***ApplicationMaster*** est le composant spécifique à chaque application qui est en charge des jobs qui y sont associés.

- Lancer et au besoin relancer des jobs
- Négocier les *Containers* nécessaires auprès du *Scheduler*
- Superviser l'état et la progression des jobs.

Un ***ApplicationMaster*** gère donc un ou plusieurs jobs tournant sur un framework donné.



TaskTracker = ApplicationMaster.

HADOOP 2.x - HDFS

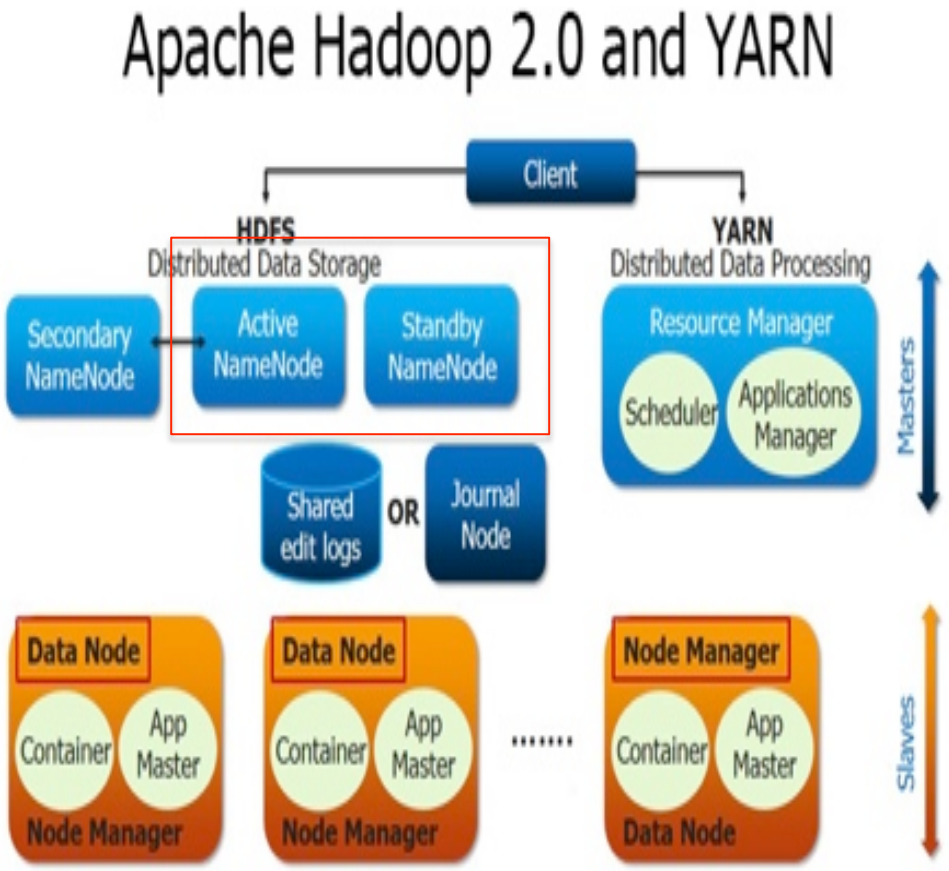
Dans un cluster HDFS, les données sont répliquées sur plusieurs nœuds (**datanode**).

Les métadonnées sont stockées dans le **namenode**. **si le namenode tombe, on perd les données**.

Des mécanismes de type **secondary** et **backup *namenode*** ont été mis en place mais le temps de down est trop important pour un gros cluster.

Dans Hadoop 2, **on peut mettre deux namenodes en mode actif/attente.**

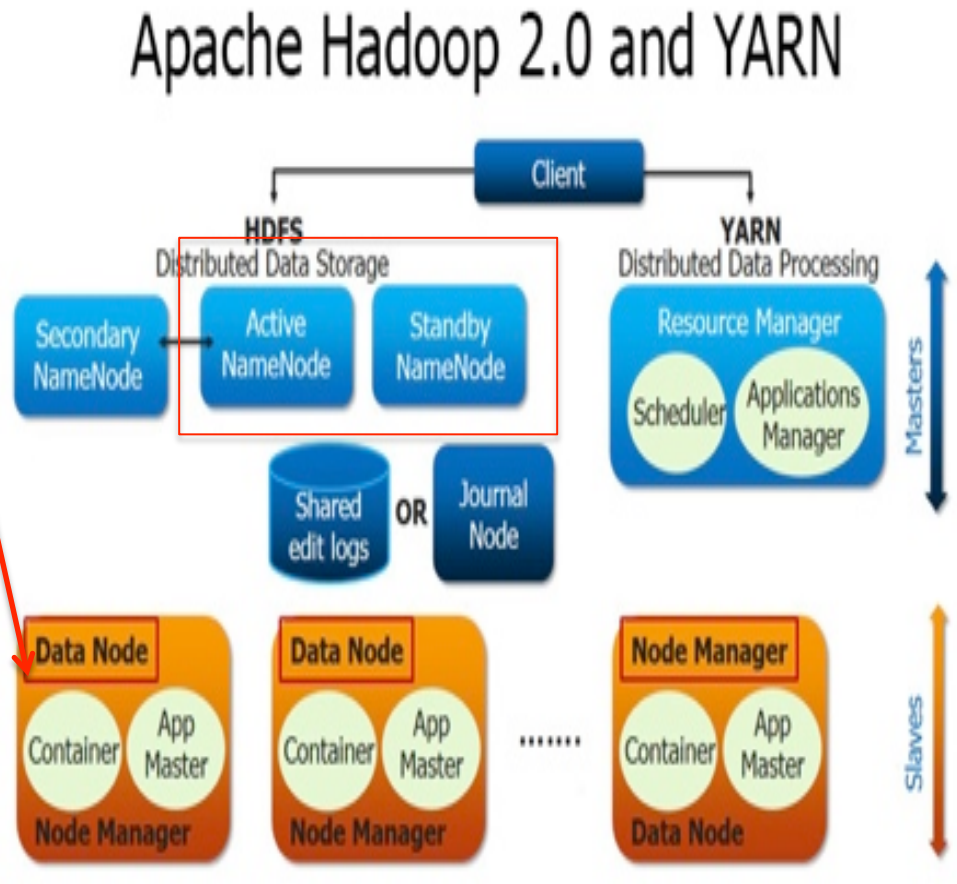
Cela signifie que le namenode actif est celui qui est utilisé par le cluster tandis que celui en attente se contente d'écouter ce qui se passe.



HADOOP 2.x - HDFS

Chaque datanode envoi ses informations aux deux namenodes (actif et celui en attente).

La bascule peut donc se faire plus rapidement lorsque le noeud actif tombe.



HADOOP 2.x - Avantage

- Les changements architecturaux d'Hadoop 2.0.0 sont, à minima, intéressants à deux titres :

  - Du point de vue ops, haute disponibilité et la capacité de mutualiser l'infrastructure HDFS;
  - Du point de vue dev et métier, le découplage par rapport à Map Reduce va permettre de faire émerger d'autres framework. Hadoop n'est donc plus un moteur MapReduce, mais bien un moteur générique pour du calcul distribué.

TP

1. **Installation de Hadoop : voir moodle**
2. **Manipulation de fichiers sur HDFS**
3. **Exemple wordCount EN JAVA**