TEK-UP
Private Higher School of Technologies & Engineering

BIG DATA

HADOOP – HDFS – MAP/REDUCE - YARN

Dr. Aïcha BEN JRAD
Maître Assistante, Tek-Up
aicha.benjrad@tek-up.tn

HADOOP

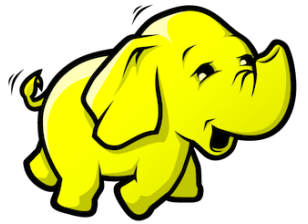
- Hadoop est une plateforme (framework) open source conçue pour réaliser d'une façon distribuée des traitement sur des volumes de données massives, de l'ordre de plusieurs pétaoctets. Ainsi, il est destiné à faciliter la création d'applications distribuées et scalables.
- Projet de la fondation Apache
  - Open source
- Modèle simple pour les développeurs: il suffit de développer des tâches Map-Reduce, depuis des interfaces simples accessibles via des librairies dans des langages multiple (Java, Python, C/C++, etc.)
- S'occupe de toutes les problématique liées au calcul distribué, comme l'accès et la partage des données, la tolérance aux pannes, ou encore la répartition de tâches aux machines membres du cluster: le programmeur a simplement à s'occuper du développement logiciel pour l'exécution de la tâche.

HADOOP

- Hadoop assure les critères des solution Big Data :
  - **Performance :** support du traitement d'énormes data sets (millions de fichiers , Go à To de données totales) en exploitant le parallélisme et la mémoire au sein des clusters de calcul.
  - **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 : HISTORIQUE

- 2003/2004 :
  - Deux whitepapers par Google
    - GFS (un système de fichier distribué)
    - MapReduce (calcul distribué)
- 2004 :
  - Première implémentation par Doug Cutting (directeur archive.org) et Mike Cafarella (étudiant) développent un framework.
- 2006 Doug Cutting, chez Yahoo
  - Améliore l'indexation du moteur de recherche de Yahoo
  - Créé une nouvelle version du framework en tant que projet Open Source de la fondation Apache : Hadoop (le nom d'un éléphant en peluche de son fils)
- 2015
  - Hadoop est utilisé dans nombreuses entreprises et universités
- 2021
  - Version 3.3.x : https://hadoop.apache.org/



QUI UTILISE HADOOP ?

ECO-SYSTÈME HADOOP

Hadoop User Experience (HUE)

Data Exchange

Sqoop

Zoo Keeper

Coordination

Pig

Scripting

Hive

SQL

Mahout

ML

Oozie

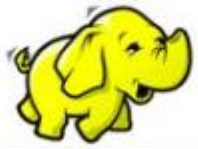
Workflow

Hbase

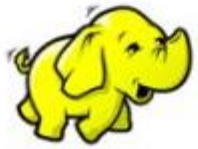
Columnar data store

Flume

Log Control



YARN/Map Reduce V2



Hadoop Distributed File System

MODULES DE APACHE HADOOP

- Module de base
  - HDFS : Système de fichiers distribué
  - MapReduce : Framework de traitement parallélisé
  - YARN (Yes Another Resource Negotiator)/MapReduce 2.0 : Système de gestion de ressources. (à partir de la version 2.3)
- Autres modules :
  - Flume : un service qui a été créé pour permettre la collecte, l'agrégation et la mobilité de données log.
  - Sqoop : Transfert des données entre Hadoop et un entrepôt de données structuré tel qu'une BD relationnelle.
  - Hive : exécution de requêtes SQL sur un cluster Hadoop (développé par Facebook)
  - Pig : Requêtage des données Hadoop à partir d'un langage de script (développé par Yahoo)

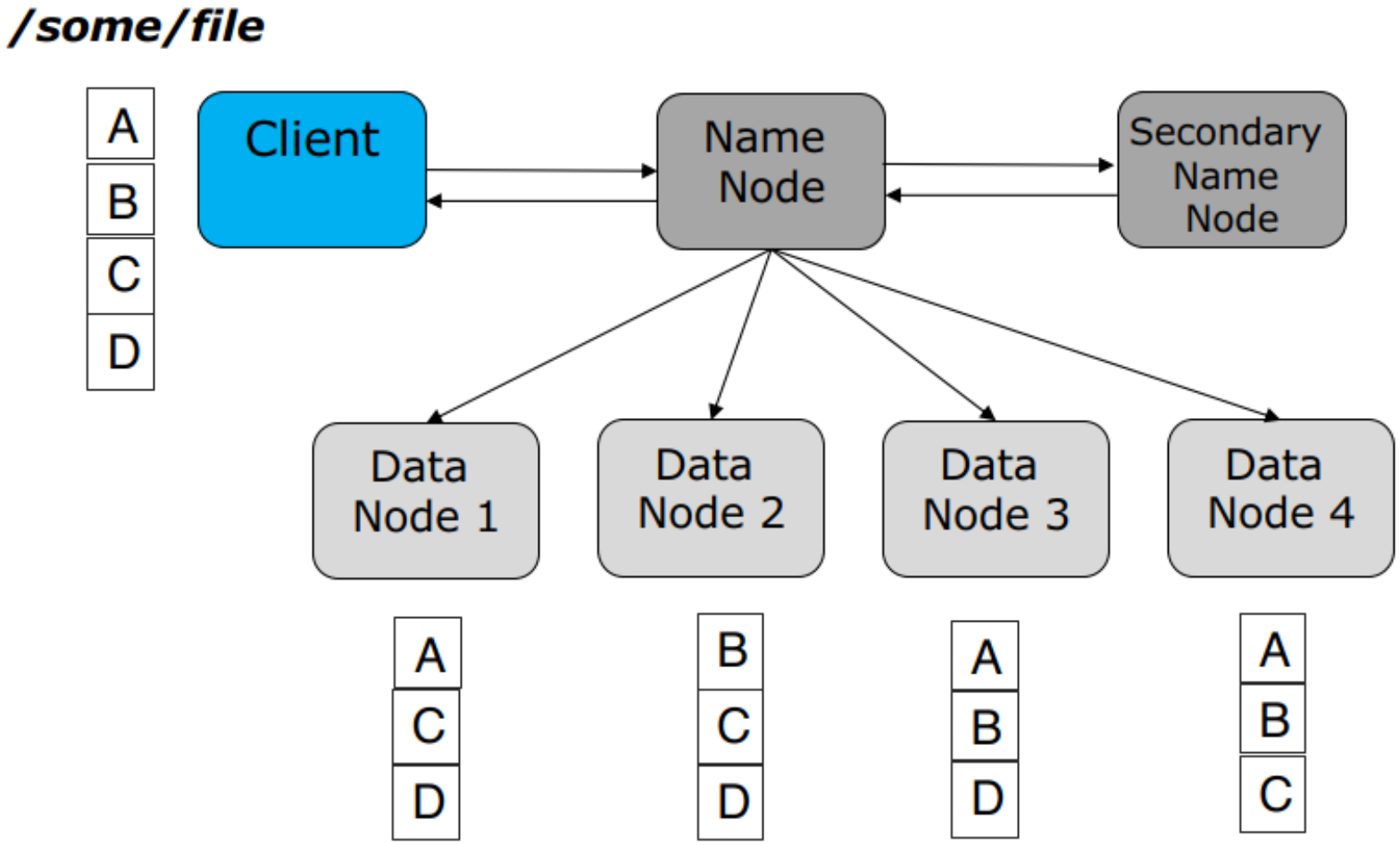
MODULES DE APACHE HADOOP

- Spark : Exécution dans la mémoire des traitements parallèles
- Mahout : Algorithmes ML se basent sur MapReduce.
- Hbase : Base de données NoSQL pour les données non structurées
- Oozie : planificateur et organisateur de workflow Hadoop.
- ZooKeeper : coordinateur de services (jobs distribués) – la synchronisation de l'exécution des tâches distribuées dans le cluster.
- Hue : une interface Web d'analyse de données avec Apache Hadoop. Il fournit un navigateur pour HDFS et Hbase et des éditeurs pour Hive, Pig et Sqoop.
- Etc.

HDFS

- HDFS: Hadoop Distributed File System
- Système pour stocker les données
- Inspiré de GFS (Google File System)
  - Reference: whitepaper du département de recherche de Google (The Google File System, 2003)
- Mêmes fonctionnalités que les systèmes de fichiers FAT32, NTFS ou encore Ext3FS
- Particularité de HDFS
  - Notion de blocs de données gérés par le FileSystem
  - Le stockage des données structurées ou non-structurées

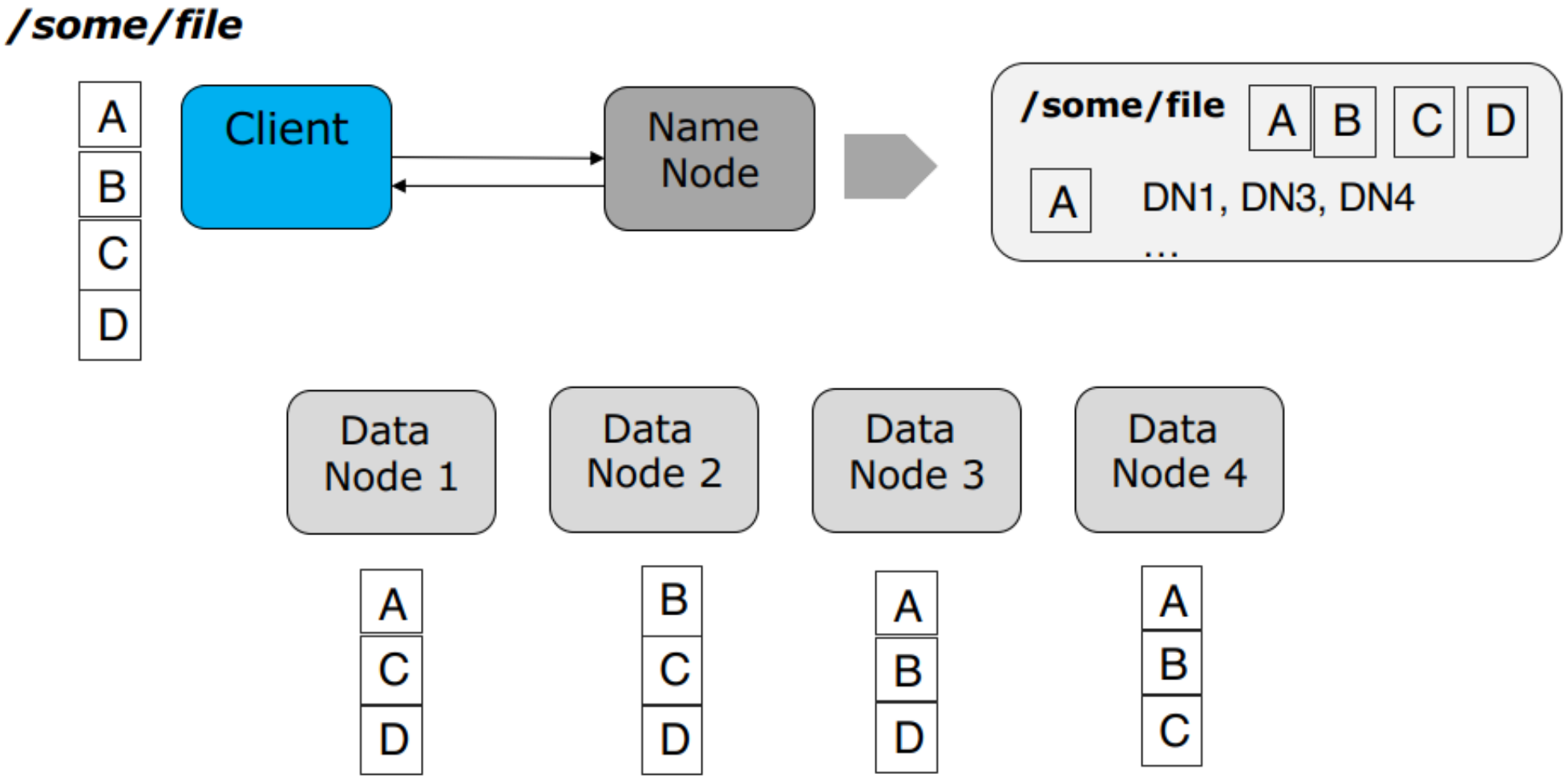
HDFS : ARCHITECTURE



COMPOSANTS DE HDFS

- Name Node (NN)
  - Un service central qui s'occupe de gérer l'état du système de fichiers. Il maintient l'arborescence du système de fichiers et les métadonnées de l'ensemble des fichiers et répertoires d'un système Hadoop. Le NN a une connaissance des Data nodes dans lesquels les blocs sont stockés
- Data Node (DN)
  - Contient les blocs de données. Il stocke les blocs de données lui-même. Il y a un Data Node pour chaque machine au sein du cluster. Les DNss sont sous les ordres du NN et sont surnommés les Workers. Ils sont donc sollicités pour les NN lors des opérations de lecture et d'écriture.
- BlockSize :
  - Taille unitaire de stockage (par défaut 128Mo)
  - Ex: 1 fichier de 1 Go, BlockSize 128Mo => 8 blocks
- Replication Factor :
  - C'est le nombre de copies d'une donnée devant être réparties sur les différents nœuds du cluster.
  - Par défaut 3 : une primaire et deux secondaires.

HDFS : RÉPARTITION ET DUPLICATION



HDFS : ECRITURE ET LECTURE

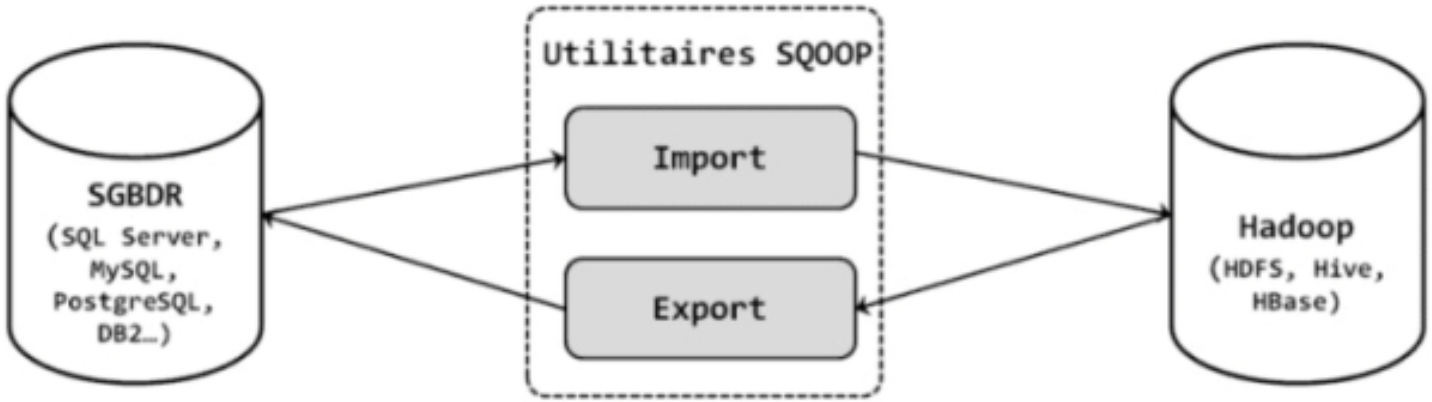
- Lecture de HDFS
  - Le client interroge le NN pour localiser les adresses de DN hébergeant les blocs à lire
  - Le NN donne la liste des DN pour chaque bloc
  - Le client choisit un DN pour récupérer le bloc, s'il ne répond pas il demande à un autre
- Ecriture dans HDFS
  - Le client indique au NN qu'il souhaite écrire un bloc
  - Le NN indique le DN à qui envoyer
  - Le client envoie le bloc au DN
  - Le DN communique ses blocs au NN
  - Les DN répliquent le bloc entre eux
  - Le cycle se répète pour le bloc suivant

SHELL HDFS

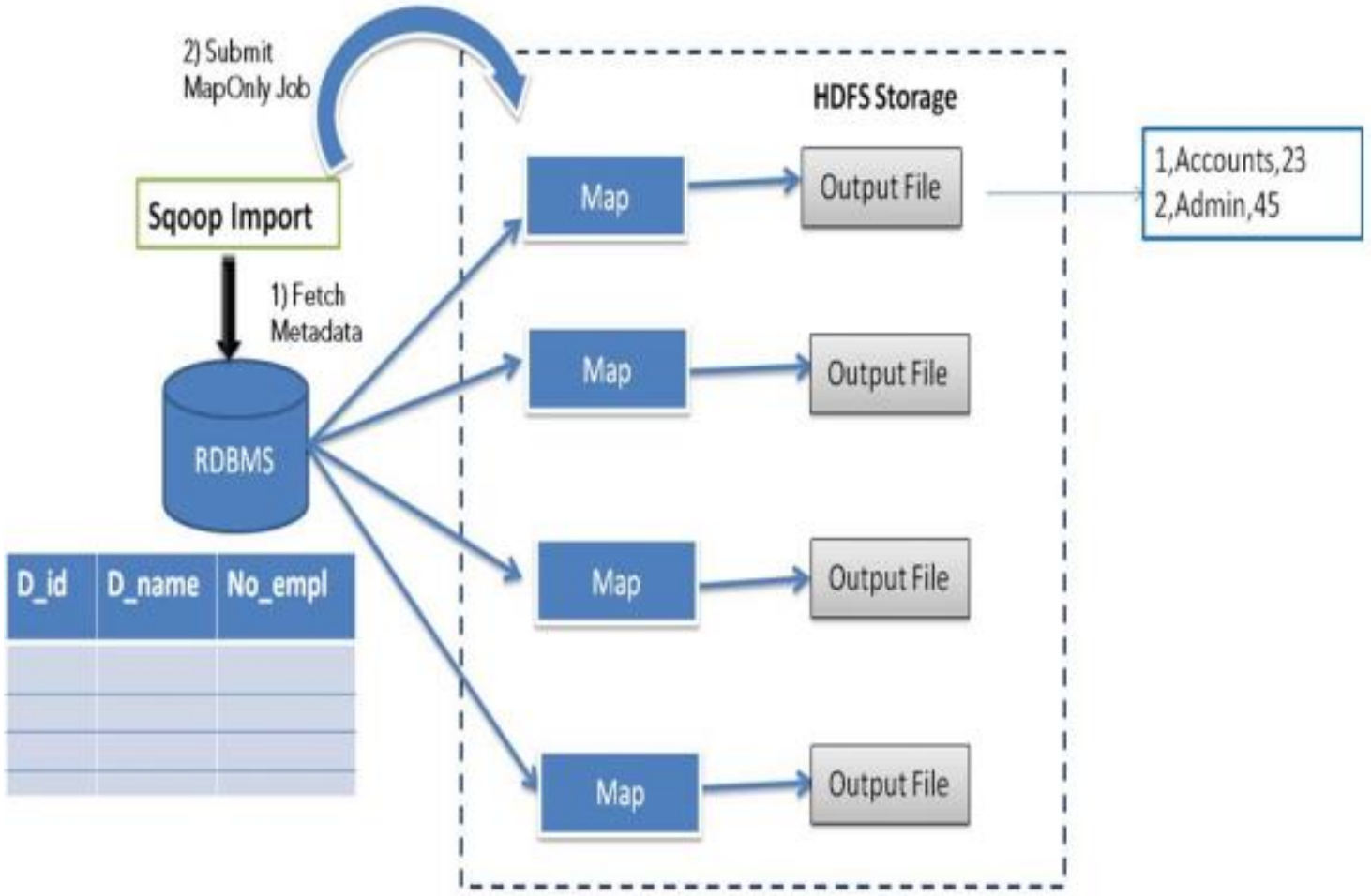
Instruction	Fonctionnalité
`hadoop fs -ls`	Afficher le contenu du répertoire racine
`hadoop fs -put file.txt`	Upload un fichier dans hadoop (à partir du répertoire courant linux)
`hadoop fs -get file.txt`	Download un fichier à partir de hadoop sur votre disque local
`hadoop fs -tail file.txt`	Lire les dernières lignes du fichier
`hadoop fs -cat file.txt`	Affiche tout le contenu du fichier
`hadoop fs -mv file.txt newfile.txt`	Renommer le fichier
`hadoop fs -rm newfile.txt`	Supprimer le fichier
`hadoop fs -mkdir myinput`	Créer un répertoire
`hadoop fs -cat file.txt | less`	Lire le fichier page par page

TRANSFERT DES DONNÉES ENTRE SGBD ET HDFS

- **Sqoop**
  - Outil conçu pour transférer efficacement des données entre des sources de données structurées, semi-structurées et non-structurées
  - Assure le transfert des données vers et depuis HDFS
  - Supporte plusieurs type de BDR
  - Conserve les protocole de sécurité de BDR
  - Utilisation de connecteur JDBC (mais pénalise les performance)



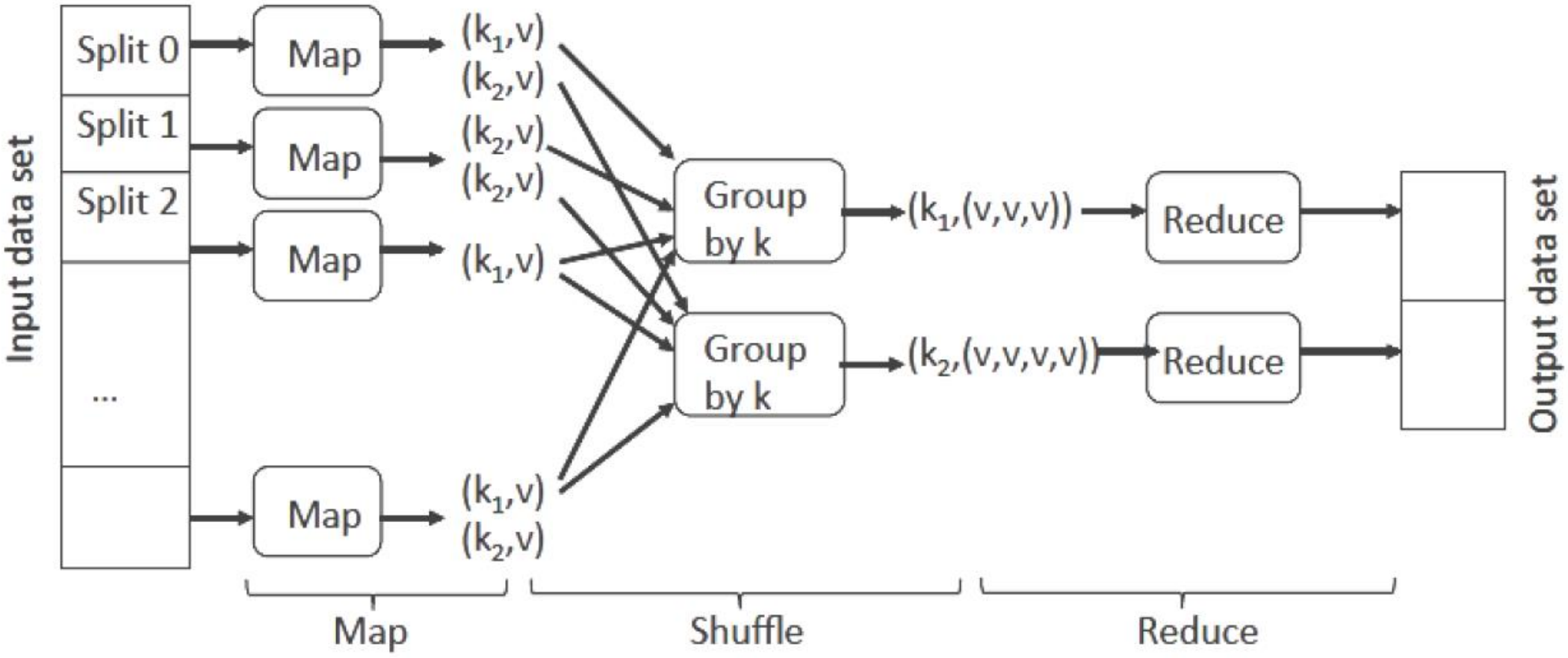
ÉTAPES DE TRANSFERT DES DONNÉES



MAP-REDUCE

- Publication issue du département de recherche de Google en 2004
  - MapReduce : Simplifier Data Processing on Large Clusters
- Principe
  - Découper le programme en plusieurs sous-problèmes de taille réduite
  - Exécution parallèle sur un cluster
- Opérations
  - Map transforme les données d'entrée en une série de couples (key, value)
  - Reduce applique un traitement sur les données d'une même clé

FONCTIONNEMENT DE MAP-REDUCE



ETAPES DE MAP-REDUCE

- Découpage (**Split**)
  - Les données d'entrée sont découpés en plusieurs blocs (splits)
- Mapping (**Map**)
  - A chaque entrée on associe le couple (clé, valeur)
- Groupage (**Shuffle**)
  - Les valeurs de même clé sont regroupés (clé, (valeur1, valeur2, …))
- Reduction (**Reduce**)
  - Pour chaque groupe indexé par un clé, on associe un couple final (clé résultat) après des calculs sur l'ensemble (valeur1, valeur2, …)

EXEMPLE WORD COUNT

- Objectif
  - Compter le nombre d'occurrences des différents mots composants un grand fichier

am stram gram
pic et pic et colégram
bour et bour et ratatam
am stram gram

EXEMPLE WORD COUNT : MAP

- Principe de Map
  - A chaque mot on associe le couple (mot, 1)
    - Clé: mot, valeur: 1
  - Le programme doit être écrit par l'utilisateur

- Pseudo-code:
  - Map (key, value)
    - POUR CHAQUE word w IN value

      Emettre(w,1)

am,1
stram,1
gram,1
pic,1
et,1
pic,1
et,1
colégram,1
bour,1
et,1
bour,1
et,1
ratatam,1
am,1
stram,1
gram,1

EXEMPLE WORD COUNT : SHUFFLE & SORT

- Les valeurs de chaque mot sont regroupées par clé
- Ces programmes sont fournis et exécutés par Hadoop d'une manière optimale

am,[1,1]
bour,[1,1]
colégram,[1]
et,[1,1,1,1]
gram,[1,1]
pic,[1,1]
ratatam,[1]
stram,[1,1]

EXEMPLE WORD COUNT : REDUCE

- Principe
  - Calculer la somme de toutes les valeurs (1) pour chaque clé (mot)

- Pseudo-code
  - Reduce (key, values)

```
// key: un mot; values : une liste de 1
Results = 0
POUR CHAQUE value v IN values
    resultat += v ;
Emettre (key, resultat)
```

```
am,[1,1]
bour,[1,1]
colégram,[1]
et,[1,1,1,1]
gram,[1,1]
pic,[1,1]
ratatam,[1]
stram,[1,1]
```

EXEMPLE WORD COUNT : PROGRAMME DE MAP EN PYTHON

- Mapper.py

```
!/usr/bin/env python
"""mapper.py"""
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    for word in words:
        # write the results to STDOUT (standard output); what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        # tab-delimited; the trivial word count is 1
        print ('%s\t%s' % (word, 1))
```

EXEMPLE WORD COUNT : PROGRAMME DE REDUCE EN PYTHON

- Reducer.py

```
#!/usr/bin/env python
import sys
# maps words to their counts
word2count = {}
# input comes from STDIN
for line in sys.stdin:
        # remove leading and trailing whitespace
        line = line.strip()
        # parse the input we got from mapper.py
        word, count = line.split('\t', 1)
        # convert count (currently a string) to int
        try:
                count = int(count)
        except ValueError:
                continue
        try:
                word2count[word] = word2count[word]+count
        except:
                word2count[word] = count
# write the tuples to stdout (they are unsorted )
for word in word2count.keys():
        print '%s\t%s'% ( word, word2count[word] )
```

EXÉCUTION SUR HADOOP

- Sauvegarder dans un dossier input_dirs les fichiers
  - reducer.py et mapper.py
- Attribuer les permissions d'execution
  - chmod +x mapper.py
  - chmod +x reducer.py
- Tester map/reduce sans hadoop
  - Echo « Hello Word Hello Hello Word World » | python mapper.py | python reducer.py
- Exécution sur Hadoop
  - $HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-streaming-1.2.1.jar \
    -input input_dirs \
    -output output_dir \
    -mapper <path>/mapper.py \
    -reducer <path>/reducer.py

EXERCICE

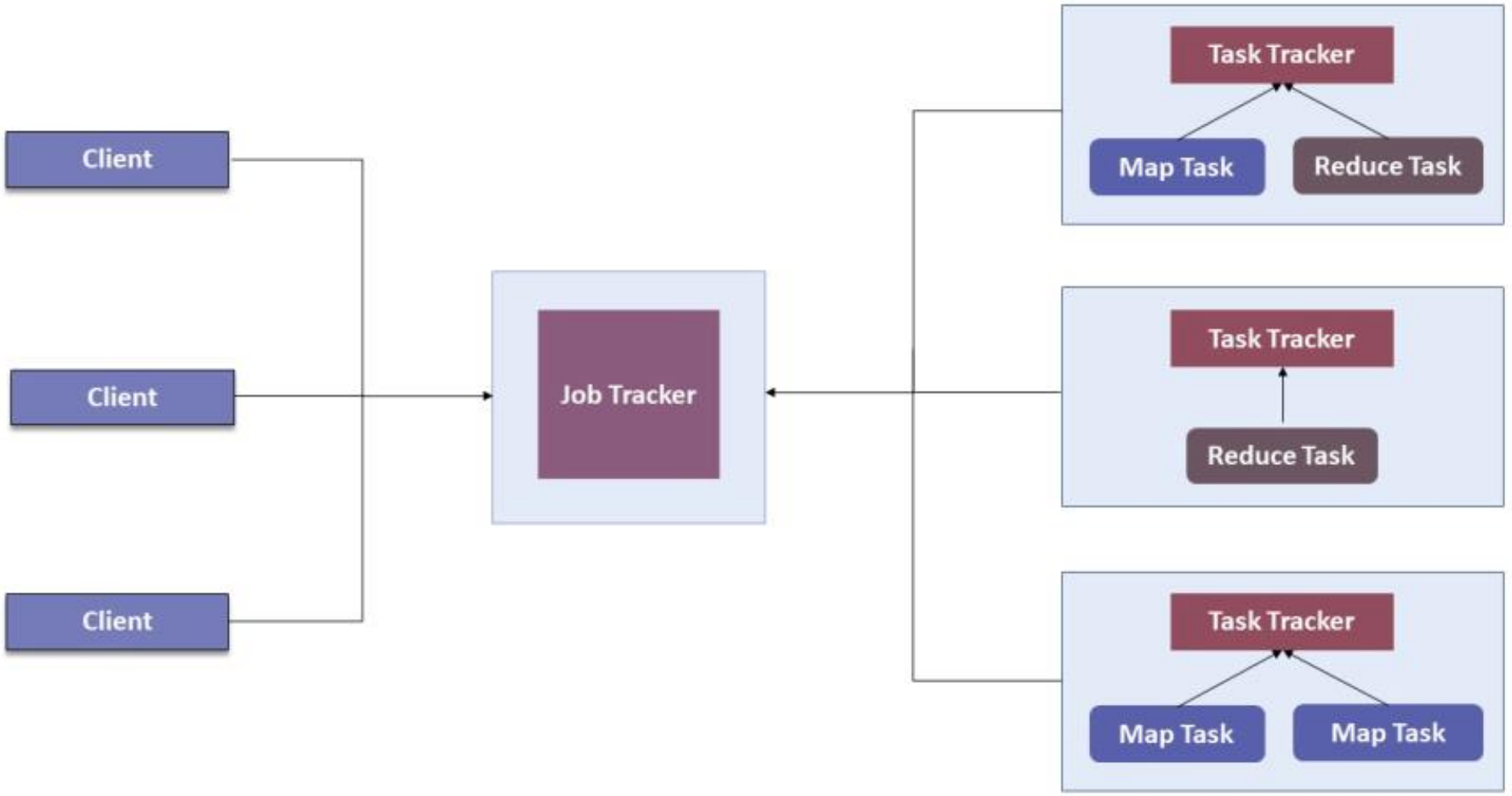
- Une chaine de magasins dispose d'un ensemble de fichiers logs sous la forme:
  - Date Cust_key OrderID Payment
- On veut connaître le nombre de commandes et la commande la plus récente par client
- Proposer une solution utilisant Map-Reduce en décrivant le principe des programmes, leur entrées et leurs sorties

SOLUTION

- Map
  - Extrait les couples (cust_key, (OrderID, Date)) de chaque entrée du fichier des logs

- Reduce
  - Compte le nombre de valeurs associé à cust_key ainsi que la commande la plus récente de l'ensemble des commandees

ARCHITECTURE HADOOP V1



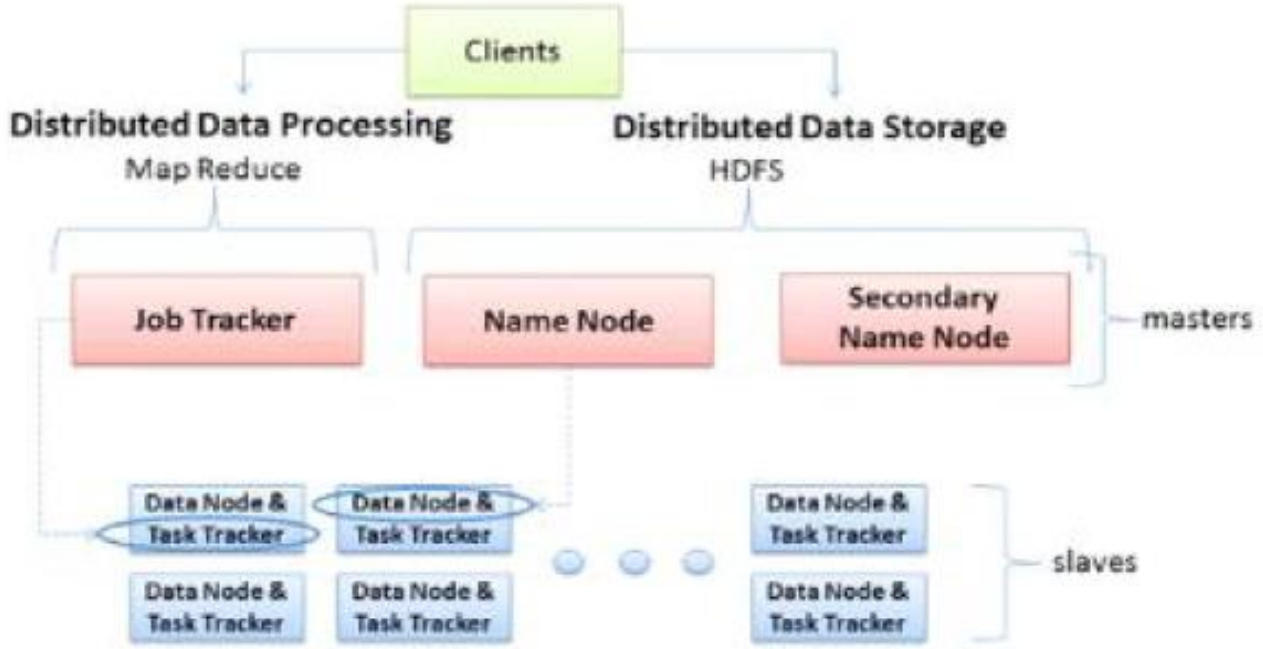
ARCHITECTURE HADOOP V1

- Comme pour HDFS, la gestion des tâches de Hadoop se base sur deux serveurs (des daemons):
- **Le JobTracker:**
  - Il y a un seul JobTracker sur une seule machine du cluster Hadoop.
  - Le JobTracker est en communication avec le NameNode de HDFS et sait donc où sont les données.
  - Il va directement recevoir la tâche à exécuter (un .jar Java), ainsi que les données d'entrées (nom des fichiers stockés sur HDFS) et le répertoire où stocker les données de sortie (toujours sur HDFS).
- **Le TaskTracker:**
  - Il est en communication constante avec le JobTracker et va recevoir les opérations simples à effectuer (MAP/REDUCE) ainsi que les blocs de données correspondants (stockés sur HDFS).
  - Il y a un TaskTracker sur chaque machine du cluster.
  - Comme le JobTracker est conscient de la position des données (grâce au NameNode), il peut facilement déterminer les meilleures machines auxquelles attribuer les sous-tâches (celles où les blocs de données correspondants sont stockés).

ARCHITECTURE HADOOP V1

- Pour effectuer un traitement Hadoop:
  - stocker nos données d'entrée sur HDFS,
  - créer un répertoire où Hadoop stockera les résultats sur HDFS,
  - compiler nos programmes MAP et REDUCE au sein d'un .jar Java.
  - On soumettra alors le nom des fichiers d'entrée, le nom du répertoire des résultats, et le .jar lui-même au JobTracker: il s'occupera du reste (et notamment de transmettre les programmes MAP et REDUCE aux serveurs TaskTracker des machines du cluster)

ARCHITECTURE HADOOP V1 : ARCHITECTURE GÉNÉRALE

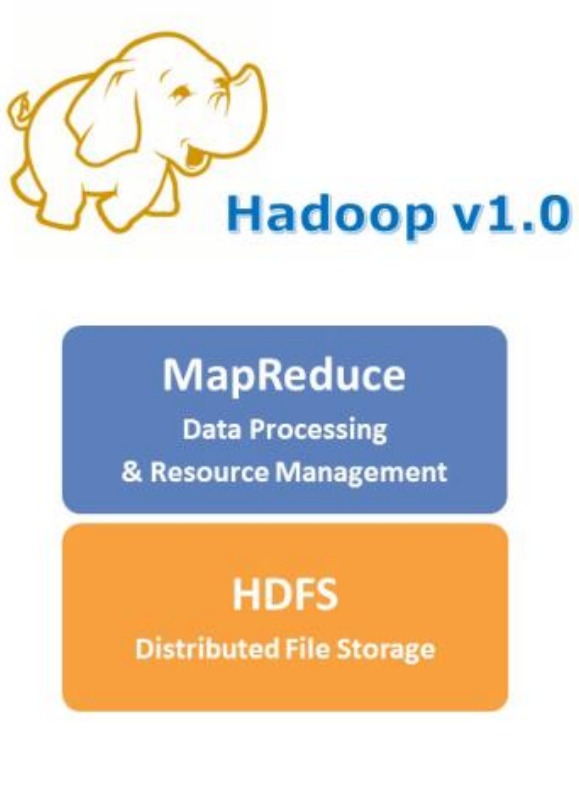


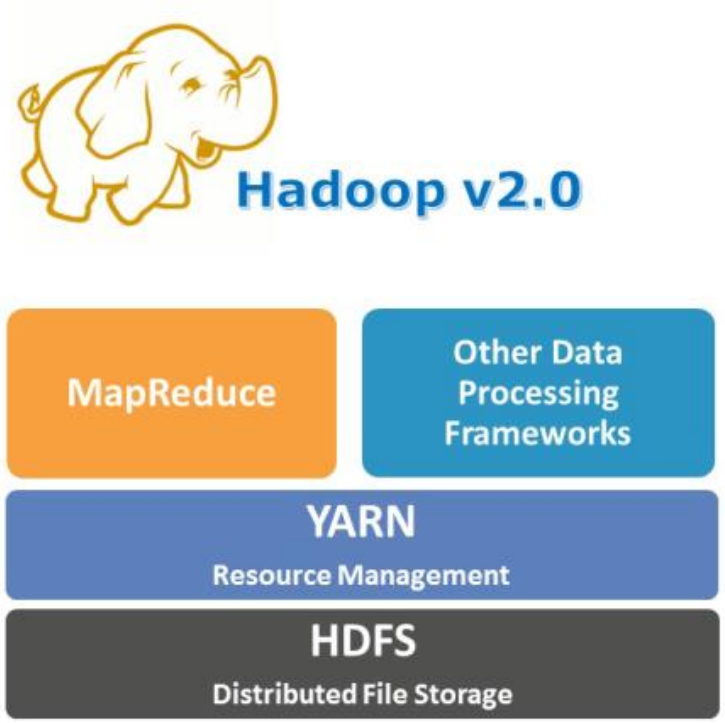
ARCHITECTURE HADOOP V1

- Les machines maîtres ont trois principaux rôles qui leur sont associées :
  - **JobTracker :** c'est le rôle qui permet à la machine maître de lancer des tâches distribuées, en coordonnant les esclaves. Il planifie les exécutions, gère l'état des machines esclaves et agrège les résultats des calculs.
  - **NameNode :** ce rôle assure la répartition des données sur les machines esclaves et la gestion de l'espace de nom du cluster. La machine qui joue ce rôle contient des métadonnées qui lui permettent de savoir sur quelle machine chaque fichier est hébergé.
  - **SecondaryNameNode :** ce rôle intervient pour la redondance du NameNode. Normalement, il doit être assuré par une autre machine physique autre que le NameNode car il permet en cas de panne de ce dernier, d'assurer la continuité de fonctionnement du cluster.
- Deux rôles sont associés aux machines esclaves :
  - **TaskTracker :** ce rôle permet à un esclave d'exécuter une tâche MapReduce sur les données qu'elle héberge. Le TaskTracker est piloté par JobTracker d'une machine maître qui lui envoie la tâche à exécuter.
  - **DataNode :** dans le cluster, c'est une machine qui héberge une partie des données. Les noeuds de données sont généralement répliqués dans le cadre d'une architecture Hadoop dans l'optique d'assurer la haute disponibilité des données.

HADOOP V1 VS HADOOP V2

- Hadoop v1 :
  - Uniquement l'utilisation de MapReduce
  - Mauvaise utilisation des ressources
  - Autres applications non prises en charge
- Hadoop V2
  - une troisième couche : YARN ("Yet Another Resource Negociator"), un outil de gestion des ressources distribuée. (puissance de calcul et la répartition de la charge entre les machines)





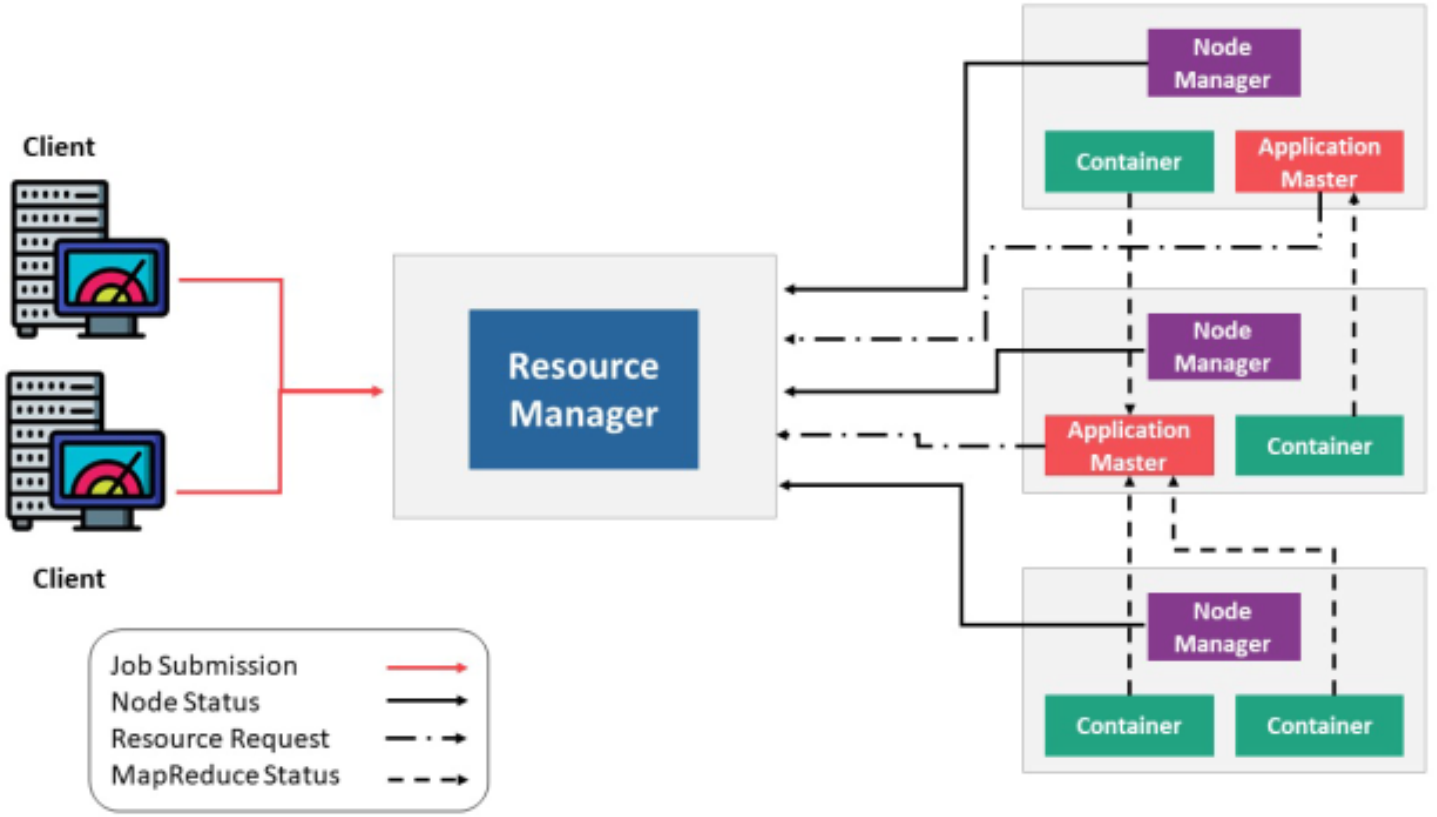
YARN: YET ANOTHER RESOURCE NEGOTIATION

- Gestion de ressources et scheduling séparés
- DataNode existent toujours
- JobTracker et TaskTracker n'existent plus
- Compatibilité : Hadoop v2 supporte MapReduce V1

YARN (MAP-REDUCE V2)

- YARN répond aux problématiques suivantes du Map Reduce :
  - Problème de limite de "Scalability" notamment par une meilleure séparation de la gestion de l'état du cluster et des ressources. (~ 4000 Noeuds, 40 000 Tâches concourantes.)
  - Problème d'allocation des ressources.(dans Hadoop V1, on a parfois des tâches map sont en attentes alors que des slots de tâche reduce sont libres)

ARCHITECTURE HADOOP V2



LES COMPOSANTS DE YARN

- Outre la gestion des ressources, YARN effectue également la planification des tâches. L'architecture Apache Hadoop YARN comprend les principaux composants suivants:
  - ResourceManager
    - agit en tant que planificateur et alloue des ressources entre toutes les applications du système (s'exécute sure un démon maître).
  - NodeManager
    - prend la navigation à partir du ResourceManager et s'exécute sur chaque nœud du cluster (les DataNodes). Les ressources disponibles sur un seul nœud sont gérées par NodeManager.
  - ApplicationMaster :
    - gère le cycle de vie du travail utilisateur et les besoins en ressources des applications individuelles. Il fonctionne avec le NodeManager et surveille l'exécution des tâches par les conteneurs.

LES COMPOSANTS DE YARN

- ***Resource Manager :***
  - remplace le JobTracker (gérer les ressources et les jobs) et ne gère que les ressources du Cluster.
  - Il a deux composantes principales:
    - ***a) Scheduler :***
      - Le Scheduler est responsable de l'allocation des ressources des applications tournant sur le cluster.
      - Il s'agit uniquement d'ordonnancement et d'allocation de ressources.
      - Les ressources allouées aux applications par le Scheduler pour leur permettre de s'exécuter sont appelées des Containers
    - ***b) Application Manager***
      - L'ApplicationsManager accepte les soumissions d'applications.
      - Une application n'étant pas gérée par le ResourceManager, la partie 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

LES COMPOSANTS DE YARN

- **Node Manager:**
  - Les NodeManager sont des agents tournant sur chaque nœud et tenant le Scheduler au fait de l'évolution des ressources disponibles. Ce dernier peut ainsi prendre ses décisions d'allocation des Containers en prenant en compte des demandes de ressources cpu, disque, réseau, mémoire
  - Permet d'exécuter plus de tâches qui ont du sens pour l'Application Master, pas seulement du Map et du Reduce.
  - La taille des ressources est variable (RAM, CPU, network….). Il y aura plus de valeurs codées en dur qui nécessitent un redémarrage.
- **ApplicationMaster :**
  - L'ApplicationMaster est le composant spécifique à chaque application, il est en charge des jobs qui y sont associés.
  - Lancer et au besoin relancer des jobs
  - Négocier les Containers nécessaires auprès du Scheduler

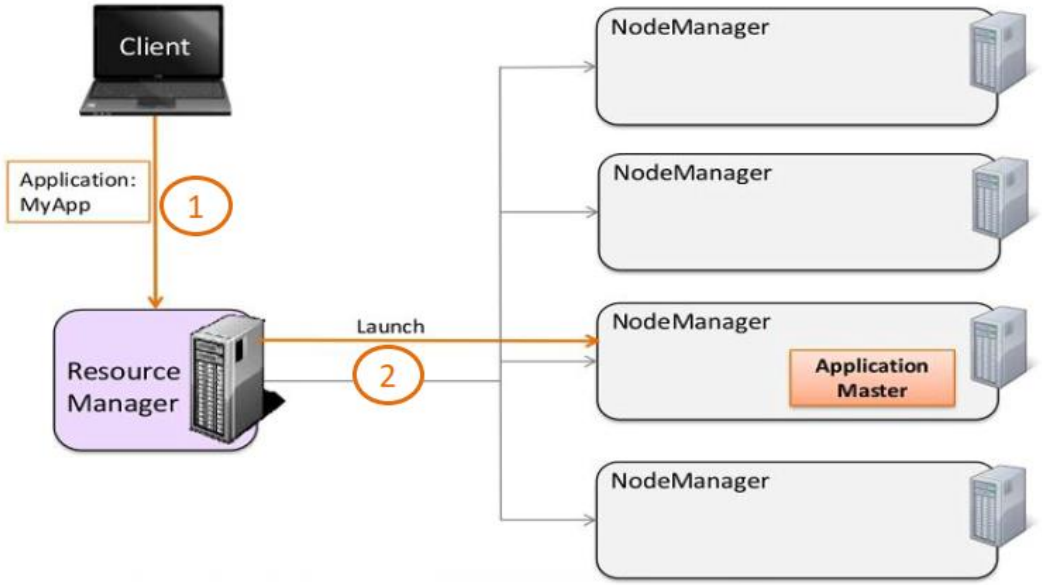
LES COMPOSANTS DE YARN

- Superviser l'état et la progression des jobs.
- Un ApplicationMaster gère donc un ou plusieurs jobs tournant sur un framework donné. Dans le cas de base, c'est donc un ApplicationMaster qui lance un job MapReduce. De ce point de vue, il remplit un rôle de TaskTracker.
- L'ApplicationsManager est l'autorité qui gère les ApplicationMaster du cluster. A ce titre, c'est donc via l'ApplicationsManager que l'on peut
  - Superviser l'état des ApplicationMaster
  - Relancer des ApplicationMaster

HADOOP V2 : LANCEMENT D'UNE APPLICATION

1) Le client (un outil Hadoop console) va soumettre le travail à effectuer au ResourceManager: une archive java .jar implémentant les opérations Map et Reduce, et également une classe driver (qu'on peut considérer comme le « main » du programme)

2) Le ResourceManager alloue un container, « Application Master », sur le cluster et y lance la classe « driver » du programme.

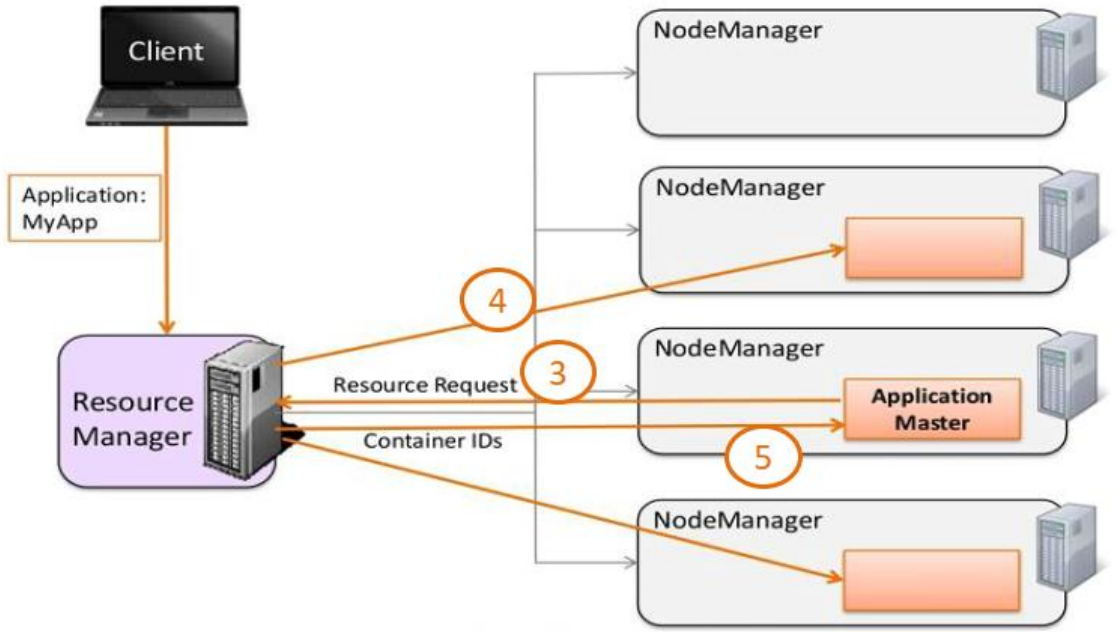


HADOOP V2 : LANCEMENT D'UNE APPLICATION

3) Cet « application master » va se lancer et confirmer au ResourceManager qu'il tourne correctement.

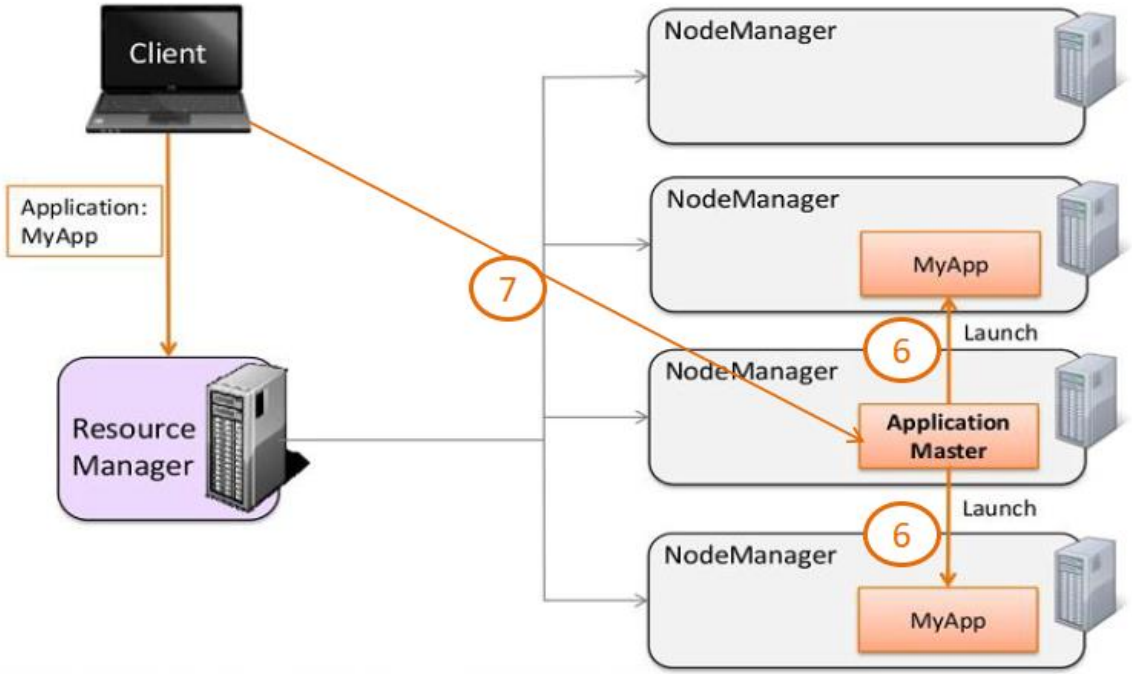
4) Pour chacun des fragments des données d'entrée sur lesquelles travailler, l'Application Master va demander au Resource Manager d'allouer un container en lui indiquant les données sur lesquelles celui-ci va travailler et le code à exécuter.

5) L'Application Master va alors lancer le code en question (map ou reduce) sur le container alloué. Il communiquera avec la tâche directement sans passer par le ResourceManager.

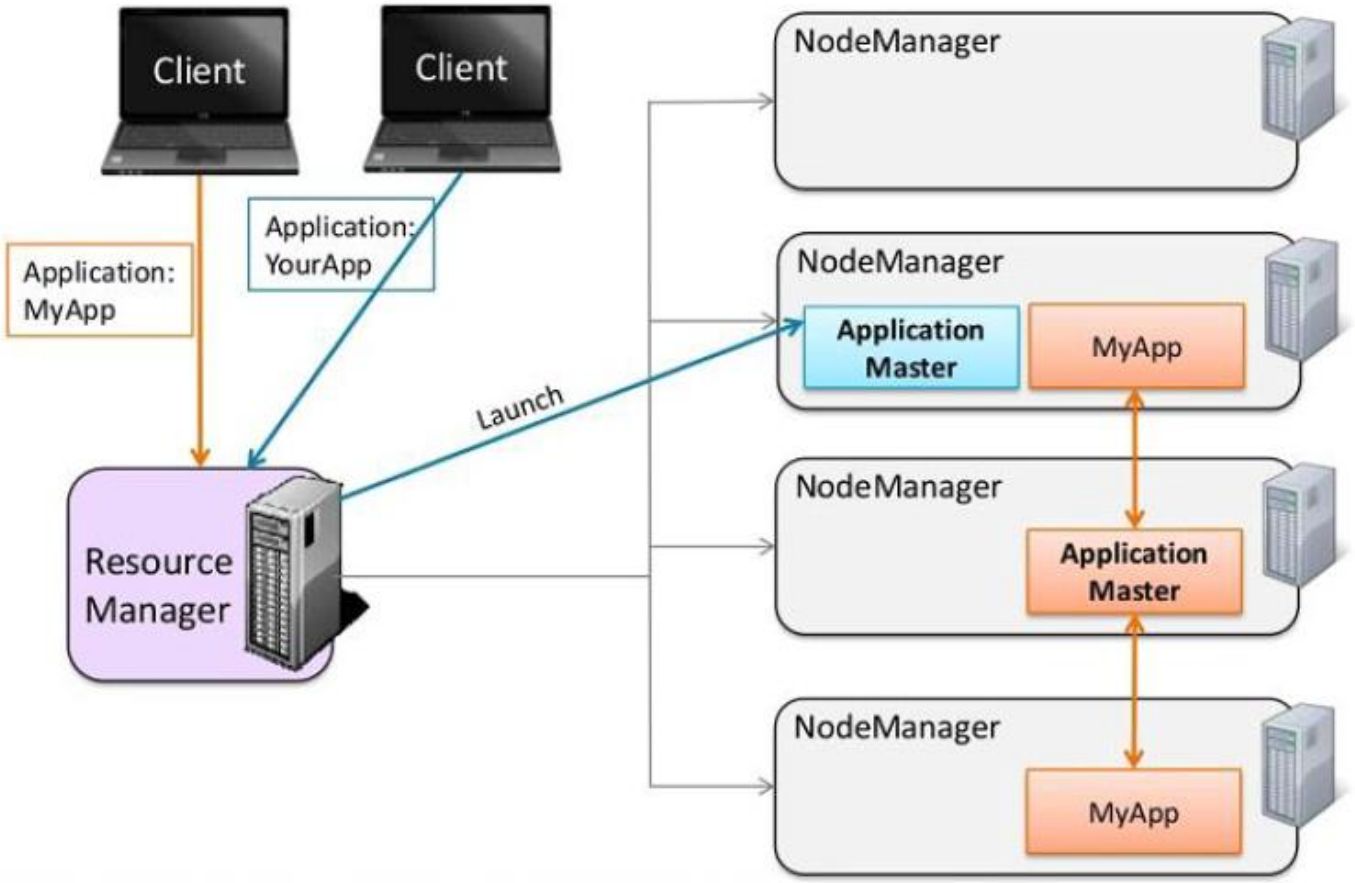


HADOOP V2 : LANCEMENT D'UNE APPLICATION

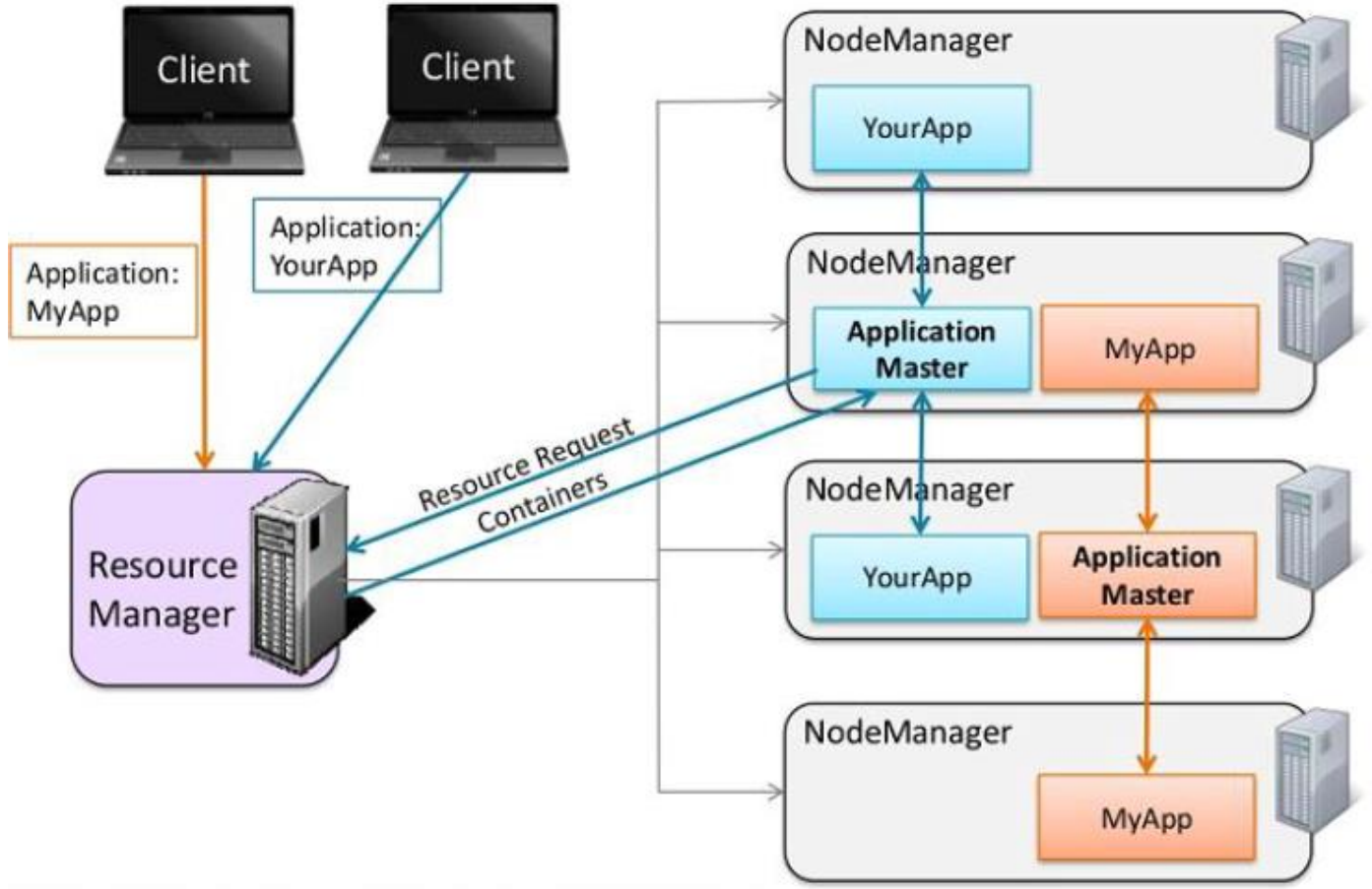
6) Ces tâches vont régulièrement contacter l'Application Master du programme pour transmettre des informations de progression, de statut, etc. parallèlement, chacun des NodeManager communique en permanence avec le ResourceManager pour lui indiquer son statut en terme de ressources (containers lancés, RAM, CPU), mais sans information spécifique aux tâches exécutées.
7) Pendant l'exécution du programme, le client peut à tout moment contacter le programme en cours d'exécution; pour ce faire, il communique directement avec l'Application Master du programme en contactant le container correspondant sans passer par le ResourceManager du cluster.
8) A l'issue de l'exécution du programme, l'Application Master s'arrête; son container est libéré et est à nouveau disponible pour de futures tâches



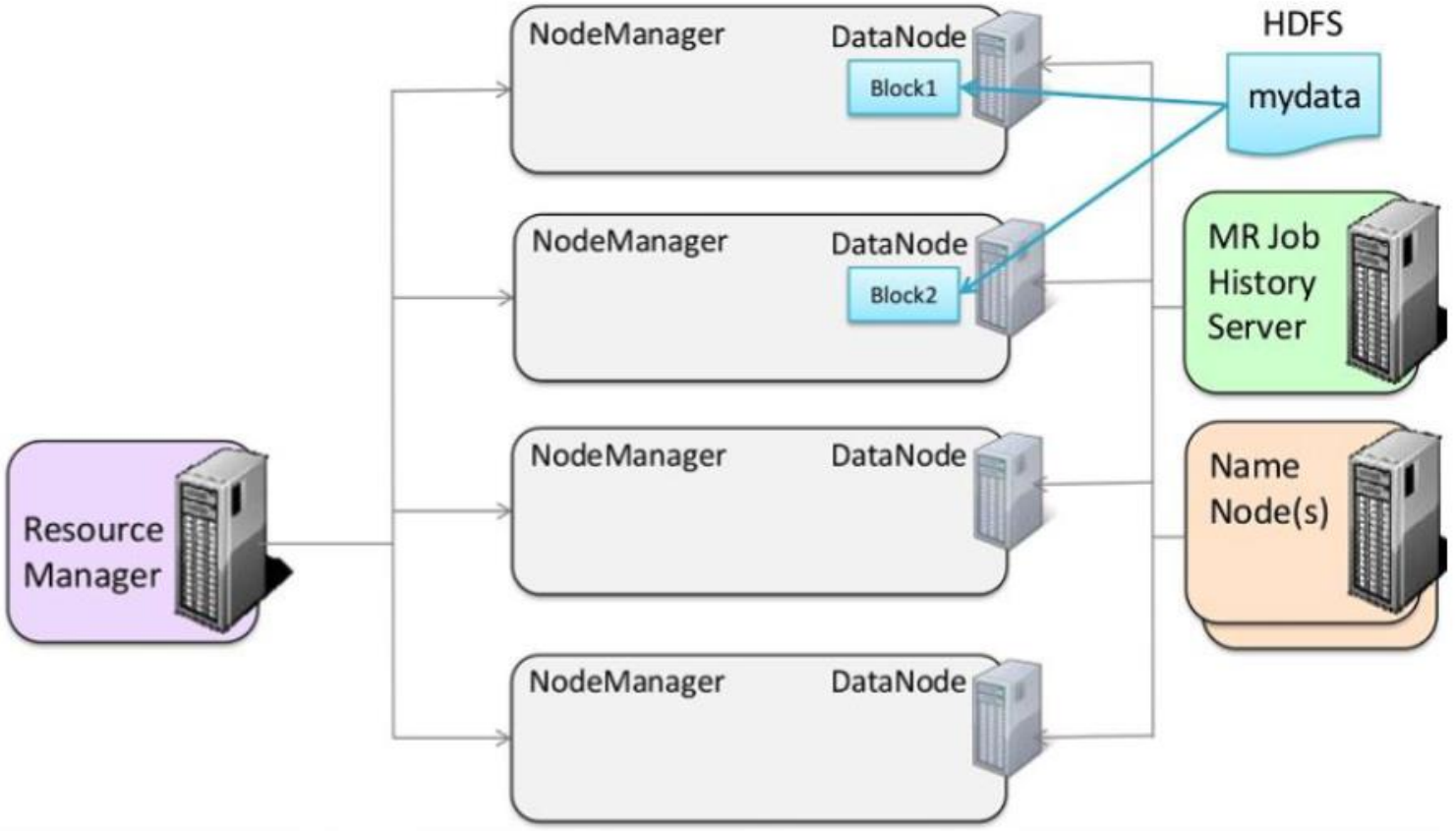
HADOOP V2 : LANCEMENT D'UNE APPLICATION



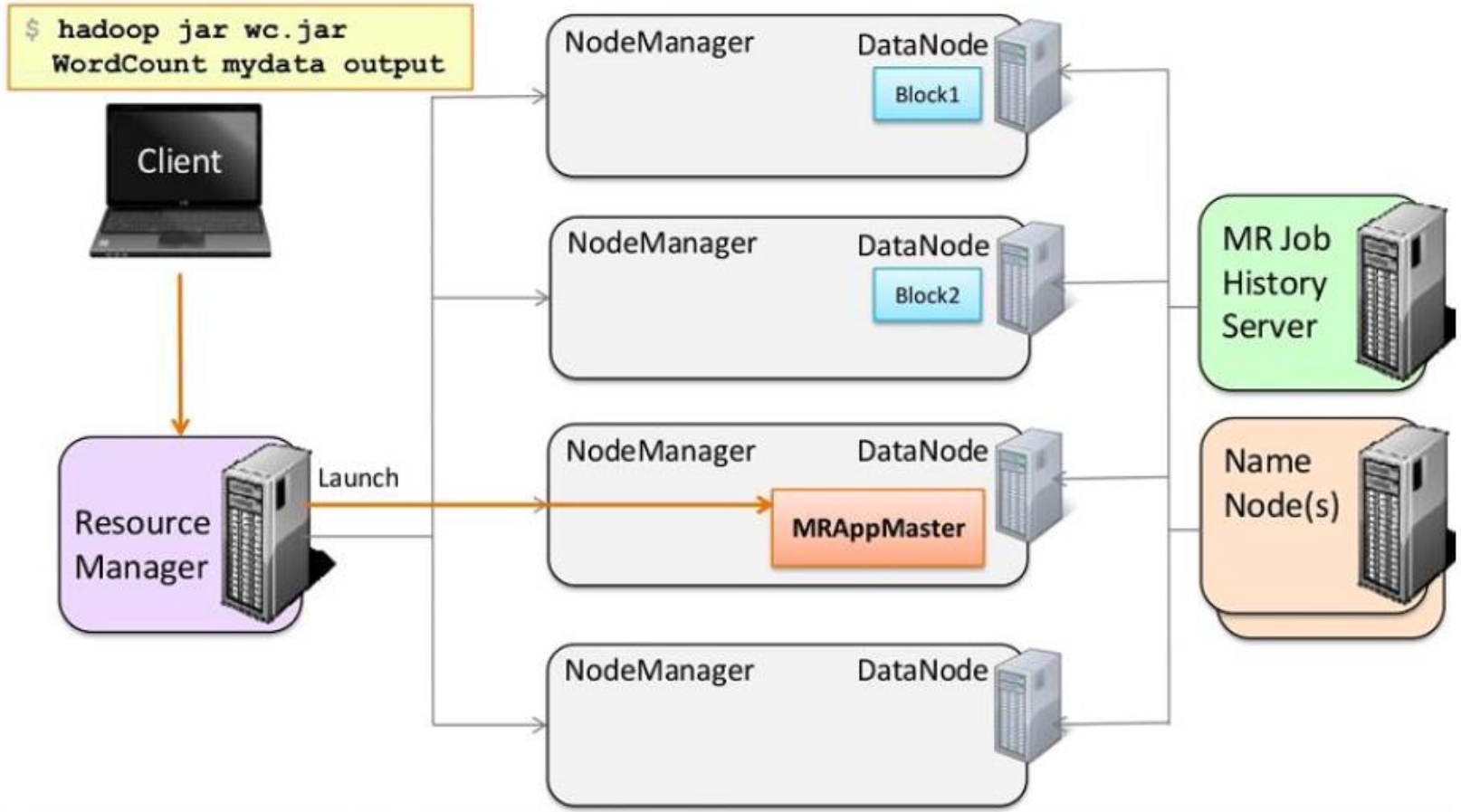
HADOOP V2 : LANCEMENT D'UNE APPLICATION



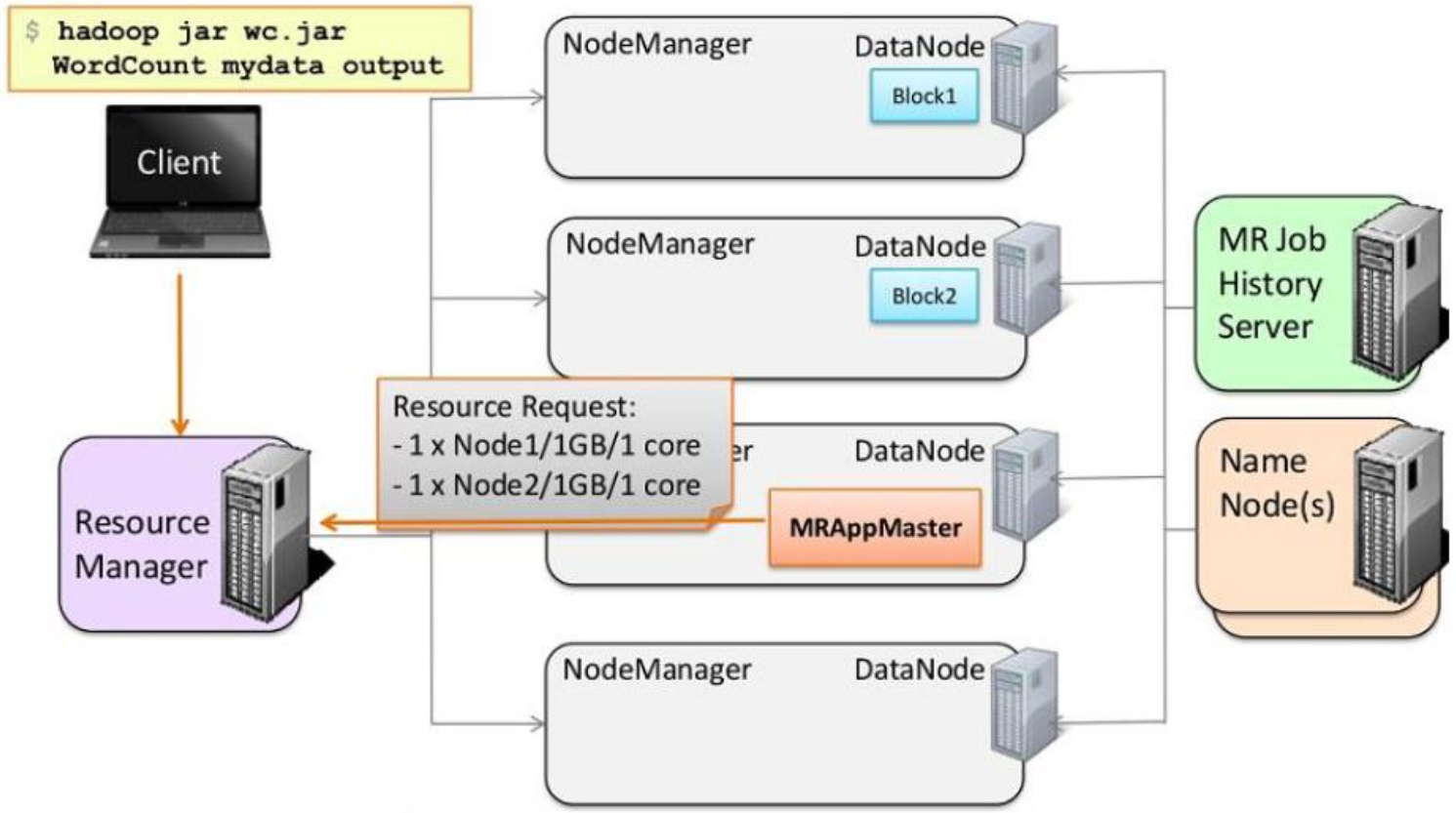
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



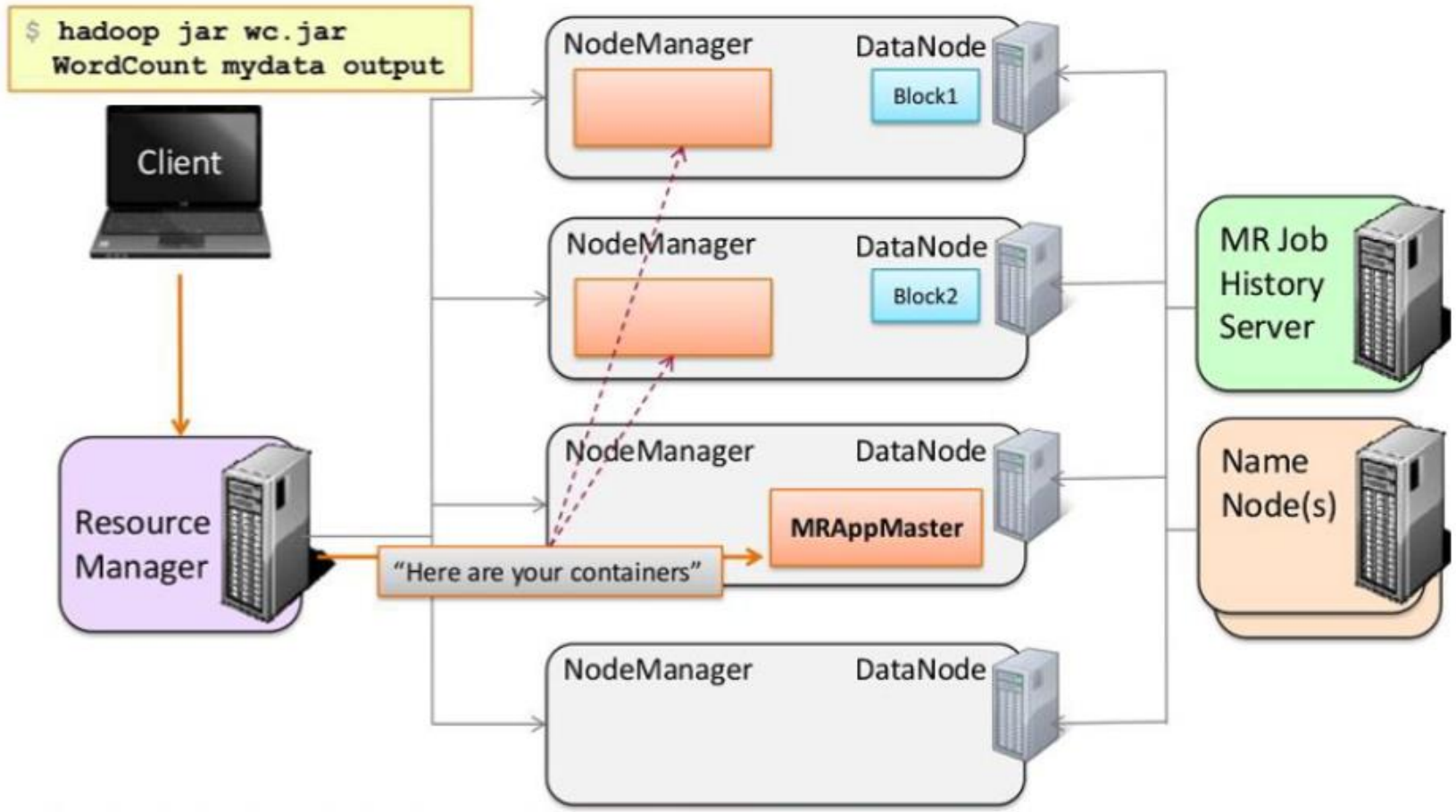
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



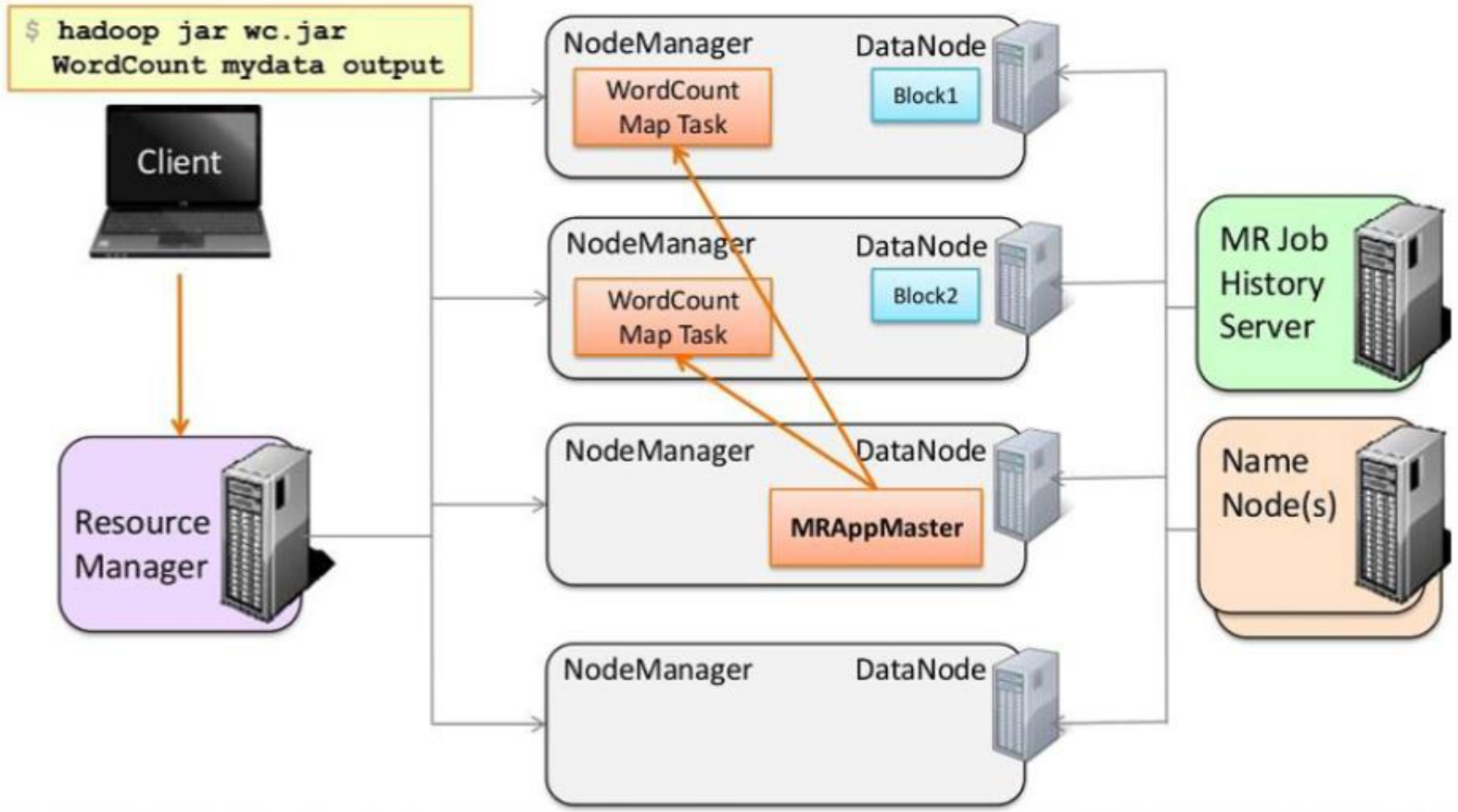
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



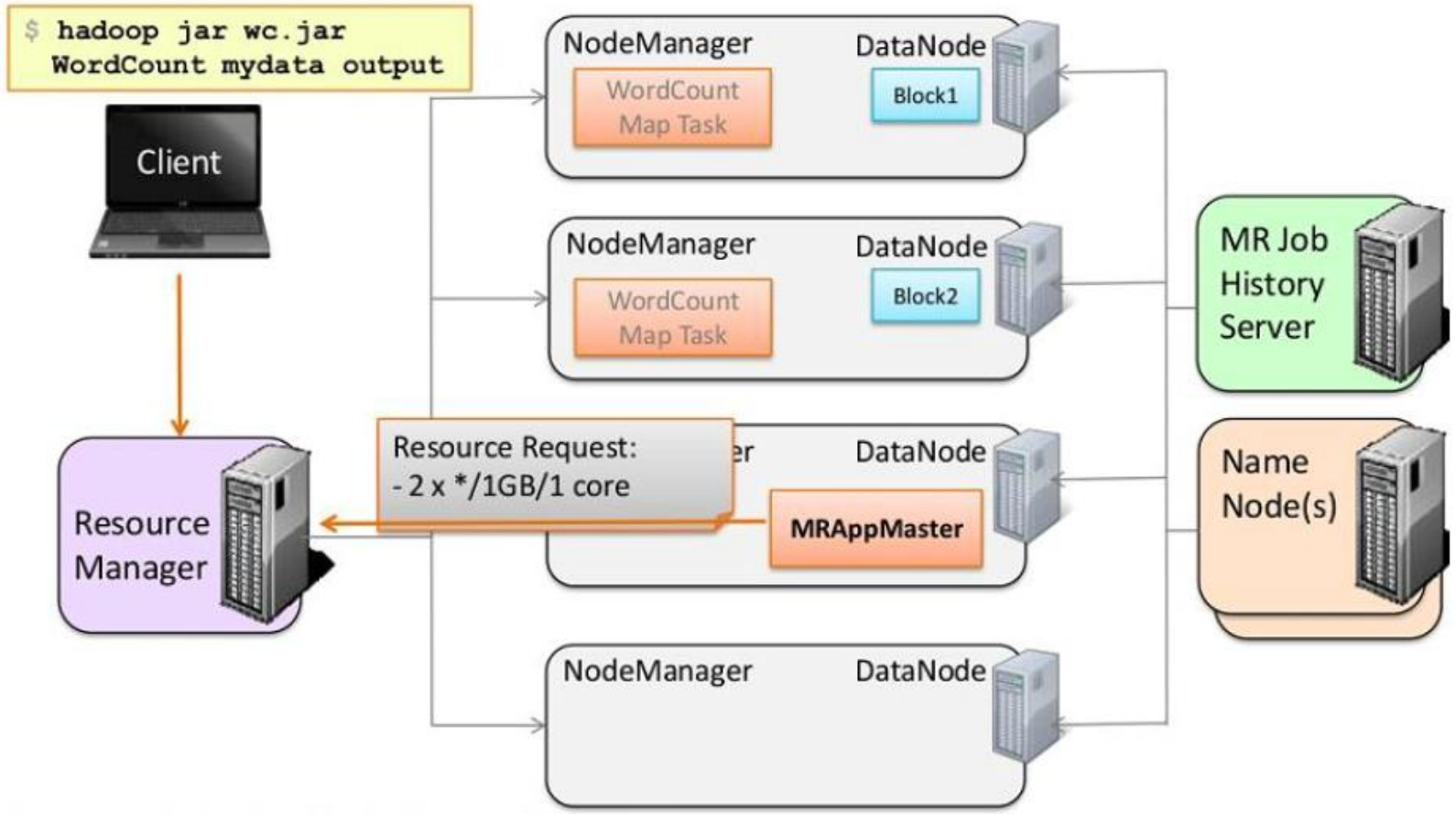
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



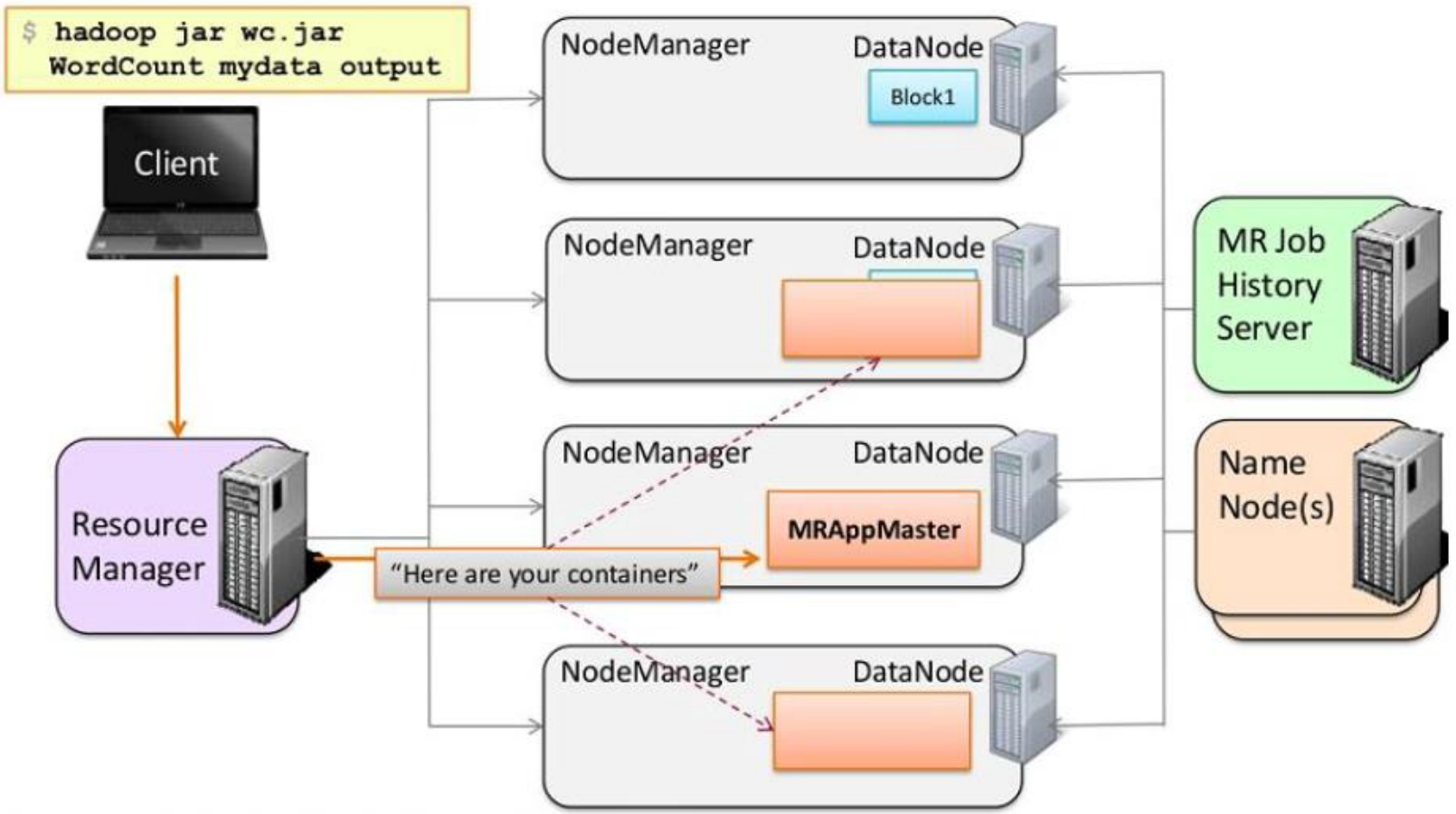
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



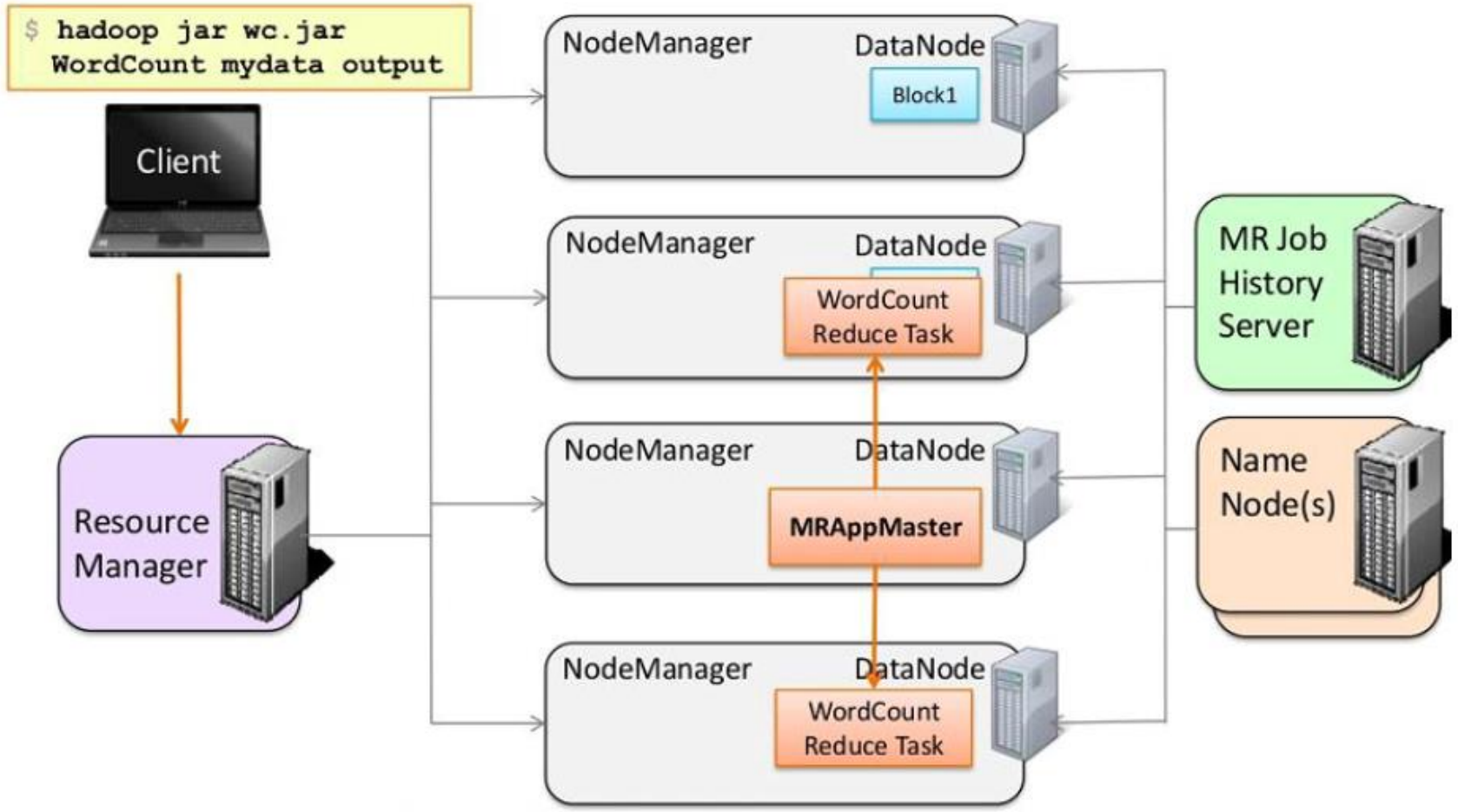
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



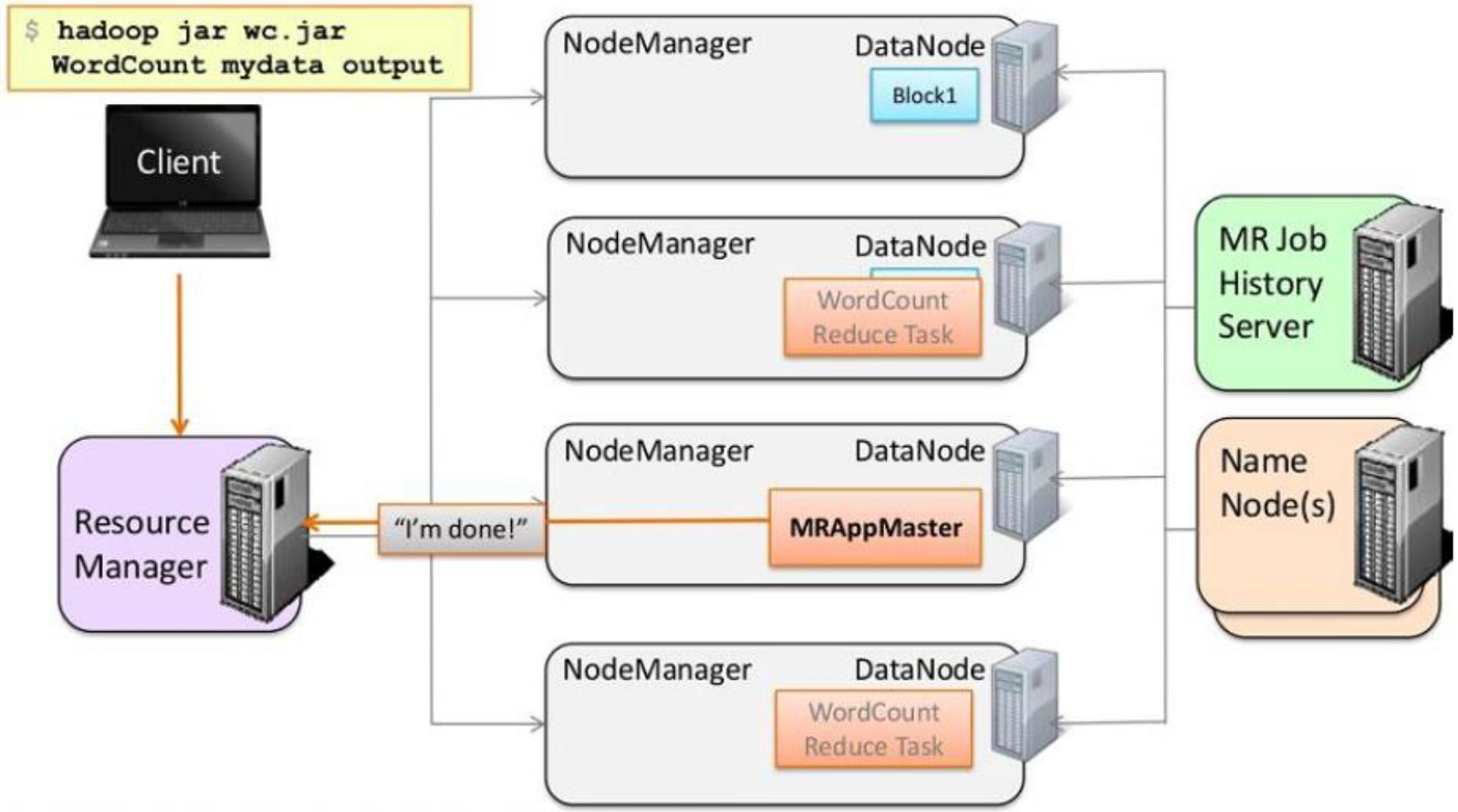
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



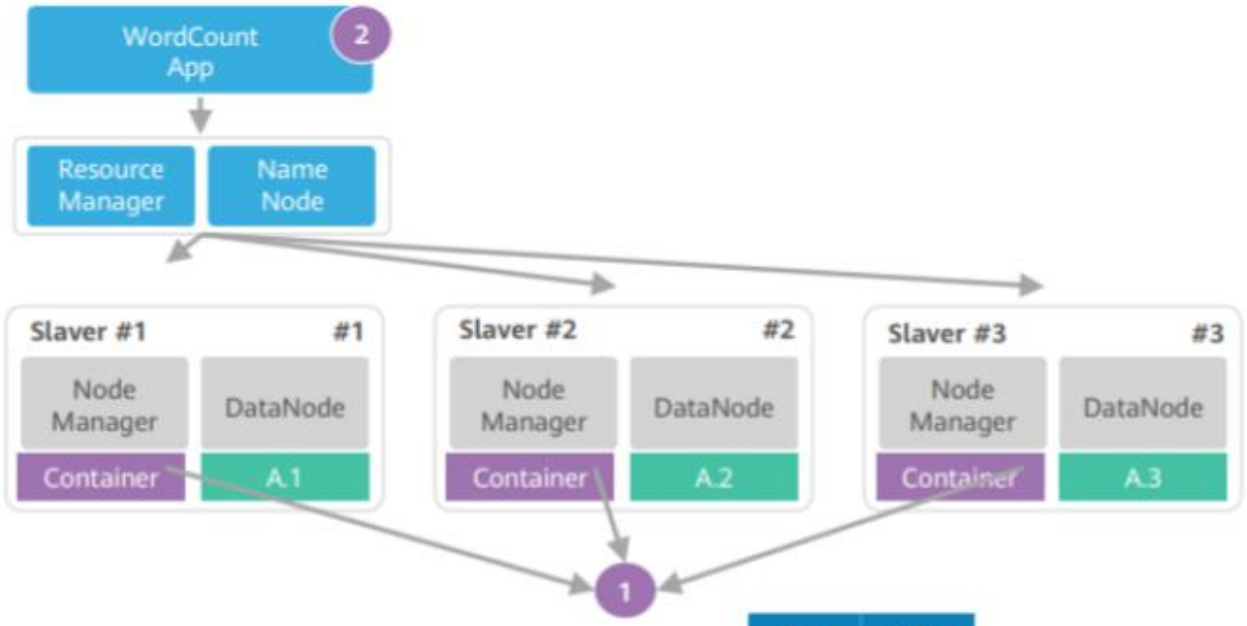
HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



HADOOP V2 : EXÉCUTION D'UN JOB MAP-REDUCE



Key	Value
a	3
are	2
hi	2
hello	3

3

Key	Value
a	1
a	1
a	1
are	1
are	1
hi	1
hi	1
hello	1
hello	1
hello	1