Hadoop HDFS

- Draft -

Khaled Jouini
j.khaled@gmail.com

Institut Supérieur d'Informatique et des Technologies de Communication

2017-2018

Hadoop HDFS (Hadoop Distributed File System)

Hadoop : **Framework** open source pour le **stockage** et le **traitement** (MapReduce) **distribués**.

Écrit en Java/C [1], inspiré de Google FS et son MapReduce

Principaux supports/contributeurs : Yahoo, FaceBook, IBM, Hortonworks, etc.

Composantes clés

- **HDFS pour le stockage distribué**
- **MapReduce pour le traitement distribué**

Écosystème

- HBASE : BD Distribuées
- ZooKeeper : Orchestrateur, Configuration/synchronisation
- Pig/Hive : Langages de haut niveau
- YARN (Hadoop 2.0) : Traitements autres que MapReduce
- Sqoop (chargement de données relationnelles), Oozie, Storm (flux de données), etc.

[1] Mais utilisable avec d'autres langages

Hadoop HDFS (Hadoop Distributed File System)

Fonctionnalités d'un système de fichiers, entre autres :

- **Manipulation des fichiers** : des opérations sont définies pour permettre la manipulation des fichiers par les programmes d'application, à savoir : créer/détruire des fichiers, insérer, supprimer et modifier un enregistrement dans un fichier.

- **Allocation de la place sur mémoires secondaires** : les fichiers étant de taille différente et cette taille pouvant être dynamique, le SGF alloue à chaque fichier un nombre variable de granules de mémoire secondaire de taille fixe (blocs).

- **Localisation des fichiers** : il est nécessaire de pouvoir identifier et retrouver les données ; pour cela, chaque fichier possède un ensemble d'informations descriptives (nom, adresse. . .) regroupées dans un inode.

- etc.

Hadoop HDFS (Hadoop Distributed File System)

Hadoop Distributed File System (HDFS) principles

- Distributed, scalable, fault tolerant, high throughput
- Data access through MapReduce
- Files split into blocks (aka splits)
- 3 replicas for each piece of data by default
- Can create, delete, and copy, but cannot update
- Designed for streaming reads, not random access
- Data locality is an important concept: processing data on or near the physical storage to decrease transmission of data

Hadoop HDFS (Hadoop Distributed File System)

HDFS : repose sur 2 types de noeuds, les NameNodes et les DataNodes

DataNode (serveur de données) : **stocke** et **restitue** les **blocs de données** (les chunks)

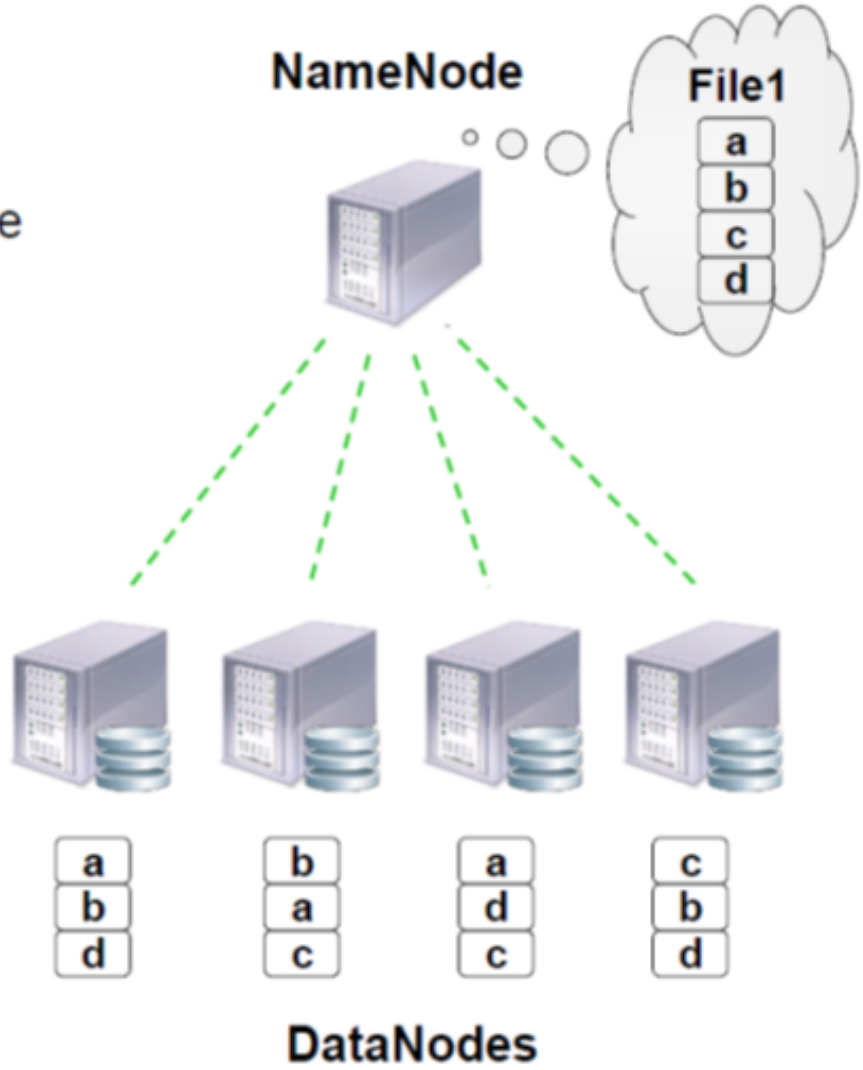
NameNode (serveur de métadonnées)

- **Stocke le répertoire**, l'espace des noms, l'arborescence et les métadonnées des fichiers.
- **Centralise la localisation des blocs** dans le cluster HDFS
- Lors de la lecture des blocs d'un fichier, le NameNode est interrogé. Il renvoie pour chaque bloc, l'adresse du DataNode le plus accessible (qui a la plus grande bande passante)
- Les DataNodes envoient périodiquement au NameNode la liste des blocs que chacun héberge.
- Si le NameNode constate qu'un bloc n'est pas suffisamment répliqué, il **initie une réplication sur d'autres DataNodes**
- Le NameNode capture périodiquement des **snapshots** (**fsImage**) des métadonnées et enregistre les modifications intermédiaires dans un fichier edit Log (*e.g.* renommage d'un fichier, migration d'un chunk, etc.).
- Il est unique mais dispose d'un noeud secondaire
- Hadoop v1 : le noeud secondaire actualise le fsImage à partir de l'edit log et ne sert qu'à une reprise plus rapide du NameNode (pas de *High Availability* et pas de *failover* automatique).

Hadoop HDFS (Hadoop Distributed File System)

HDFS architecture

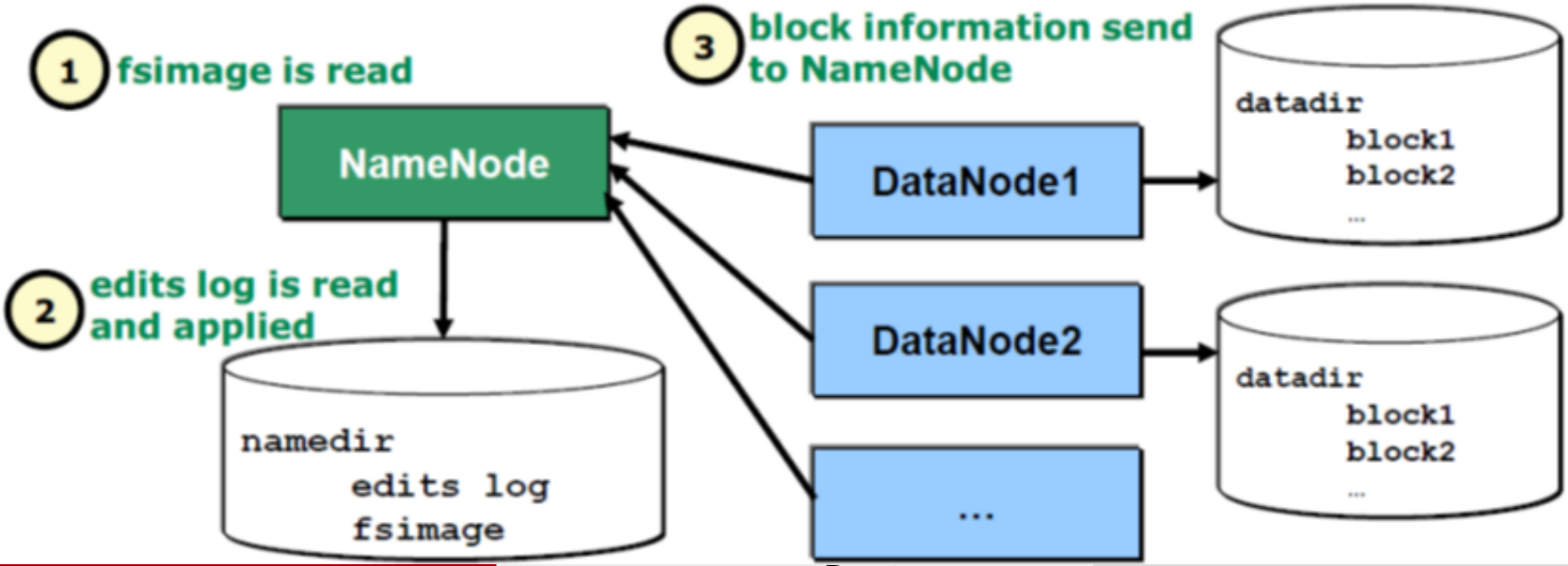
- Master/Slave architecture
- Master: **NameNode**
  - manages the file system namespace and metadata
    - FsImage
    - Edits Log
  - regulates client access to files
- Slave: **DataNode**
  - many per cluster
  - manages storage attached to the nodes
  - periodically reports status to NameNode



Hadoop HDFS (Hadoop Distributed File System)

NameNode startup

1. NameNode reads fsimage in memory
2. NameNode applies editlog changes
3. NameNode waits for block data from data nodes
   - NameNode does not store the physical-location information of the blocks
   - NameNode exits SafeMode when 99.9% of blocks have at least one copy accounted for

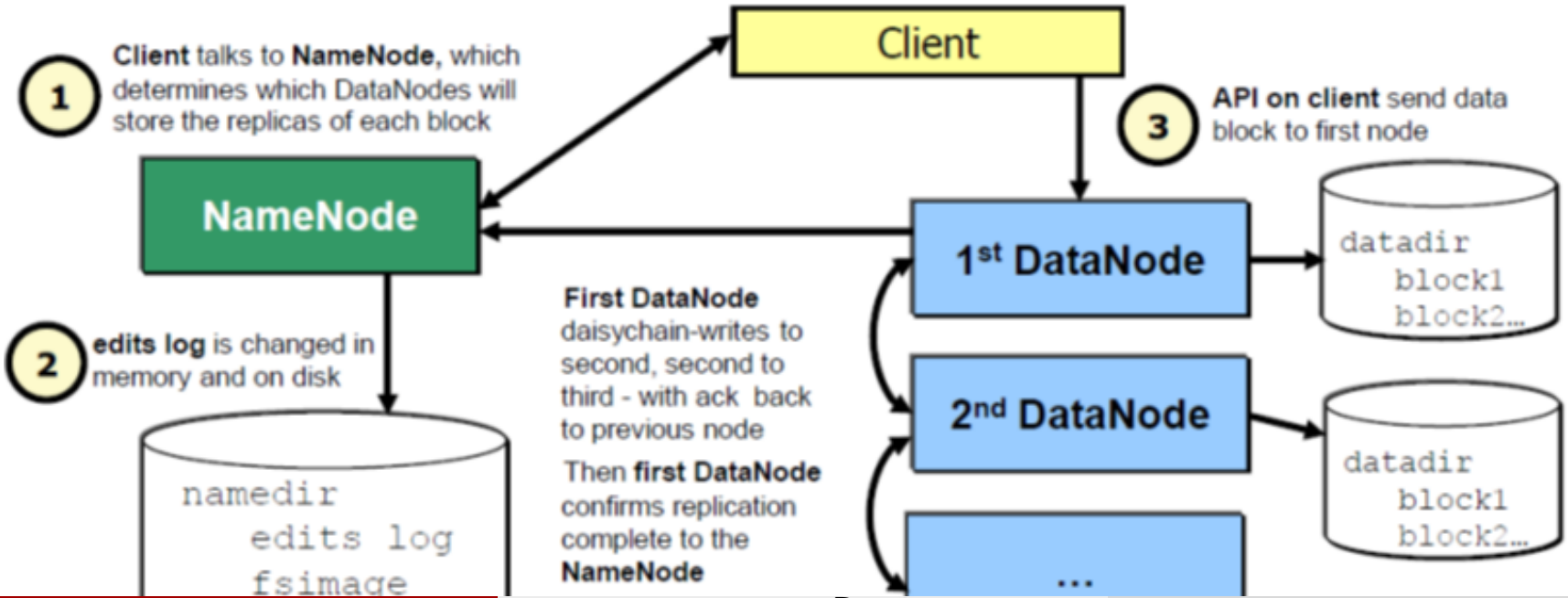


Hadoop HDFS (Hadoop Distributed File System)

Réplication asynchrone

Adding a file to HDFS: replication pipelining

1. File is added to NameNode memory by persisting info in edits log
2. Data is written in blocks to DataNodes
   - DataNode starts chained copy to two other DataNodes
   - if at least one write for each block succeeds, the write is successful



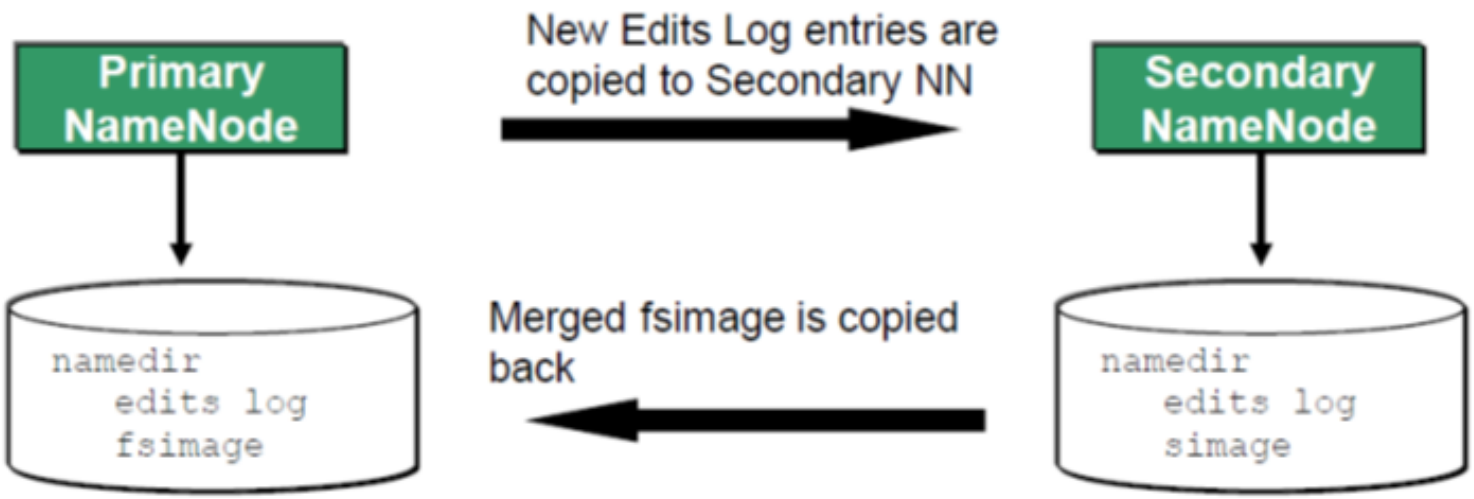
Le sharding de Hadoop HDFS (Hadoop Distributed File System)

- Hadoop v2 : entre autres, amélioration de la haute disponibilité
- **StandBy NameNode** : permet une reprise sur panne automatisée et une haute disponibilité.
- Pour pouvoir récupérer l'editlog même en cas d'indisponibilité du NameNode, celui-ci est stocké sur des noeuds tiers (JournalNodes)
- **JournalNodes** : le NameNode stocke l'editlog dans les JournalNodes et le standby le récupère des JournalNodes.

Hadoop HDFS (Hadoop Distributed File System)

Secondary NameNode

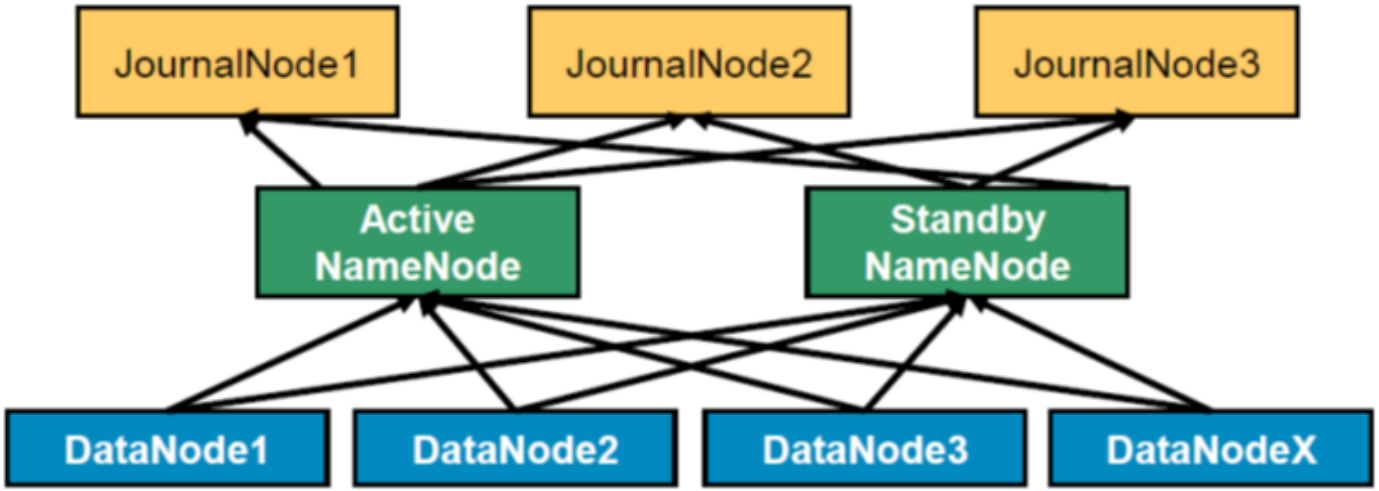
- During operation primary NameNode cannot merge fsImage and edits log
- This is done on the secondary NameNode
  - Every couple minutes, secondary NameNode copies new edit log from primary NN
  - Merges edits log into fsimage
  - Copies the new merged fsImage back to primary NameNode
- Not HA but faster startup time
  - Secondary NN does not have complete image. In-flight transactions would be lost
  - Primary NameNode needs to merge less during startup
- Was temporarily deprecated because of NameNode HA but has some advantages
  - (No need for Quorum nodes, less network traffic, less moving parts)



Hadoop HDFS (Hadoop Distributed File System)

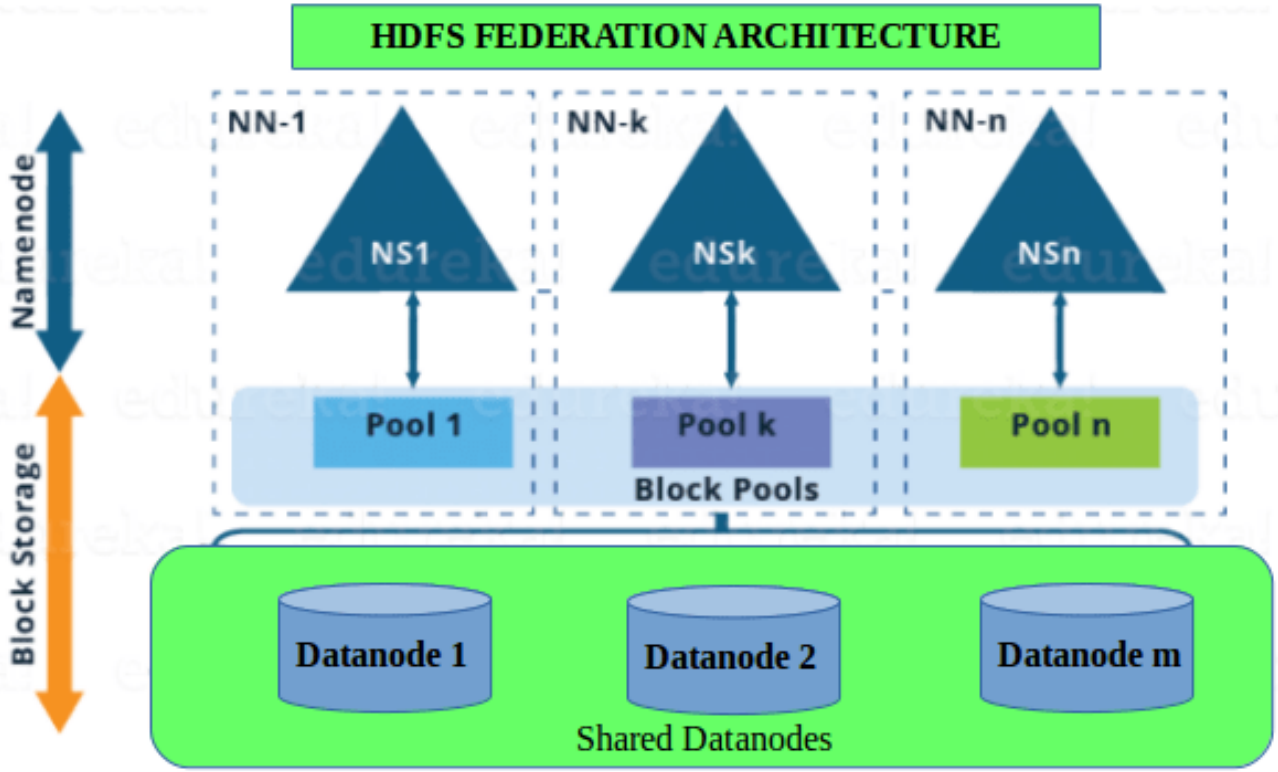
HDFS-2 NameNode HA (High Availability)

- HDFS-2 adds NameNode High Availability
- Standby NameNode needs filesystem transactions and block locations for fast failover
- Every filesystem modification is logged to at least 3 quorum journal nodes by active NameNode
  - Standby Node applies changes from journal nodes as they occur
  - Majority of journal nodes define reality
  - Split Brain is avoided by JournalNodes (They will only allow one NameNode to write to them)
- DataNodes send block locations and heartbeats to both NameNodes
- Memory state of Standby NameNode is very close to Active NameNode
  - Much faster failover than cold start



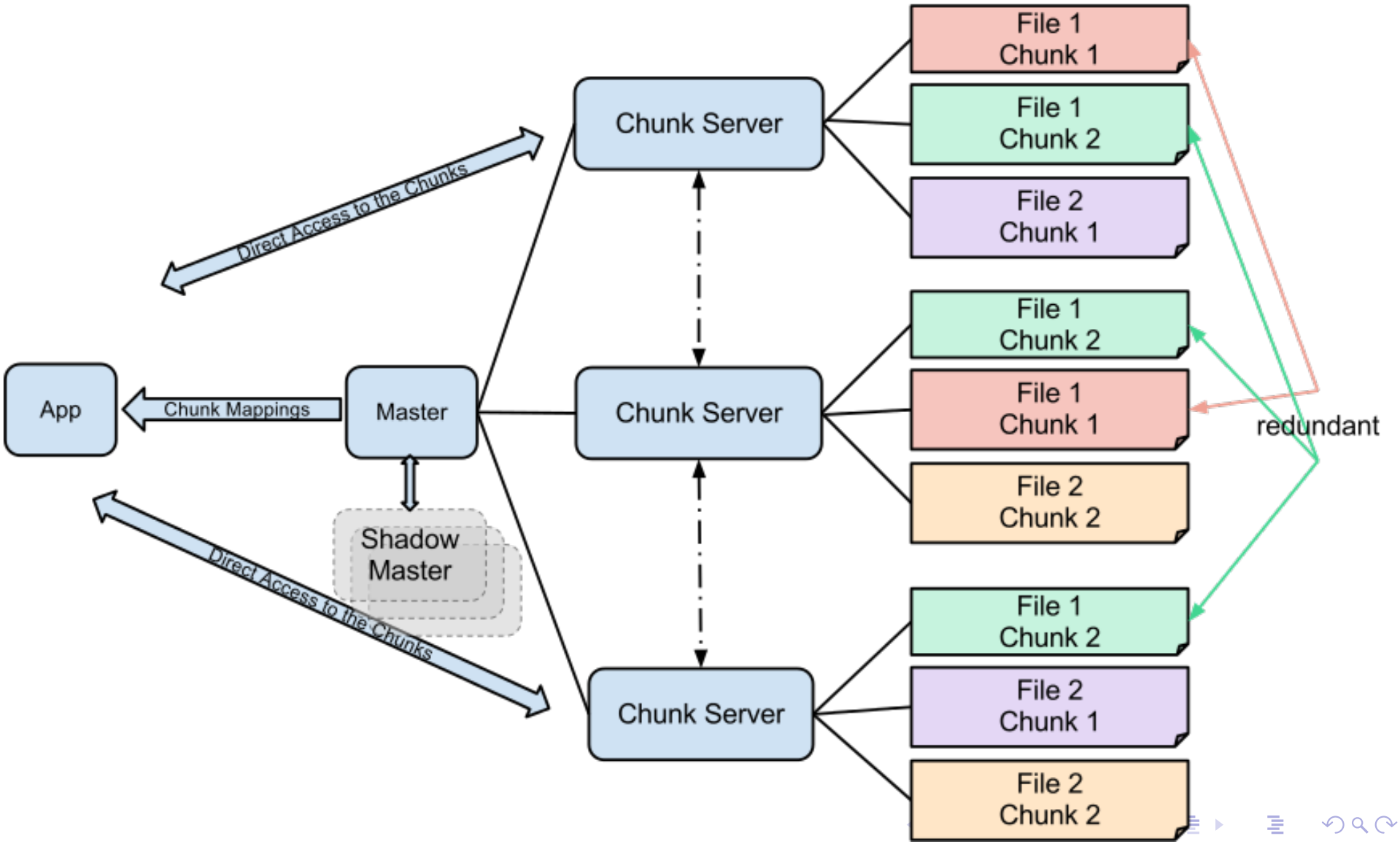
Hadoop HDFS (Hadoop Distributed File System)

- Les JournalNodes ne requièrent pas beaucoup de ressources et peuvent tourner sur des machines existantes du cluster.
- Hadoop peut tolérer la perte de (N+1)/2 JournalNodes
- Hadoop v3 : possibilité d'utiliser plusieurs namenodes/namespaces (fédération)



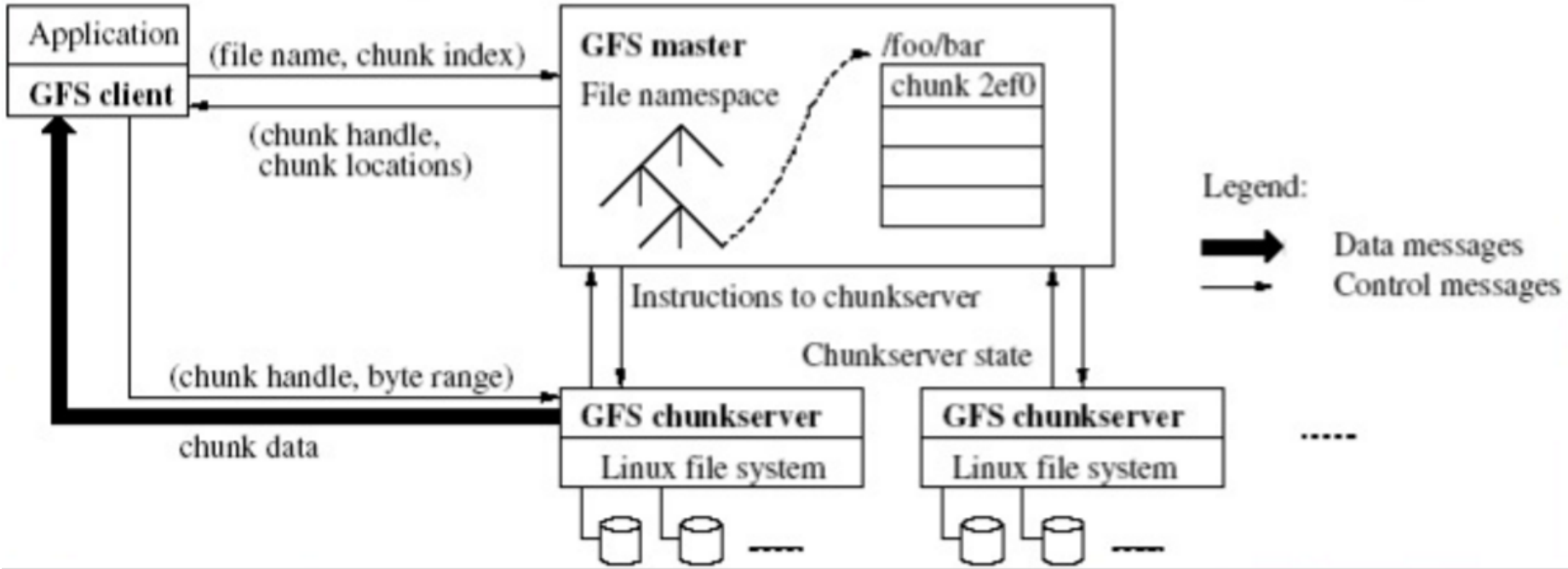
Le sharding de Google File System (GFS)

Architecture un peu différente, mais les grands principes sont là!



Google File System (GFS)

Architecture un peu différente, mais les grands principes sont là!

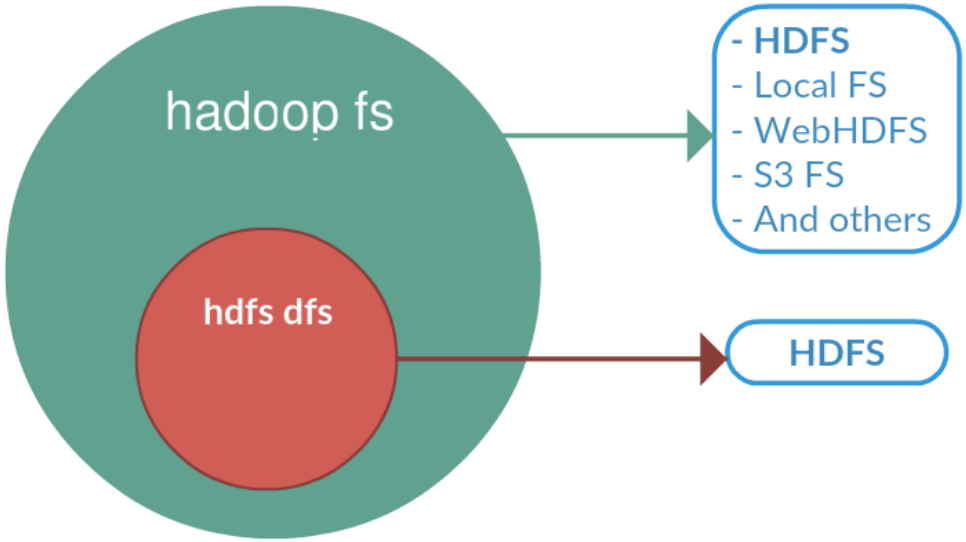


Formats de fichiers supportés par HDFS

- **Fichiers text usuels** (y compris csv, tsv, etc.).
- **Sequence files** : fichiers binaires, spécialement conçus pour Map/Reduce. Chaque donnée doit avoir un champ key et un champ value.
- **Avro** : plus qu'un format de fichiers, Avro est un framework de sérialisation/déserialisation d'objets complexes. Décrit les données qu'il stocke en JSON. Les données elles-mêmes peuvent être au format Json ou dans un format binaire plus compact.
- **Parquet et RCFile** : Formats de fichier oreinté colonnes (efficace pour le calcul d'agrégats).

Commandes Hadoop

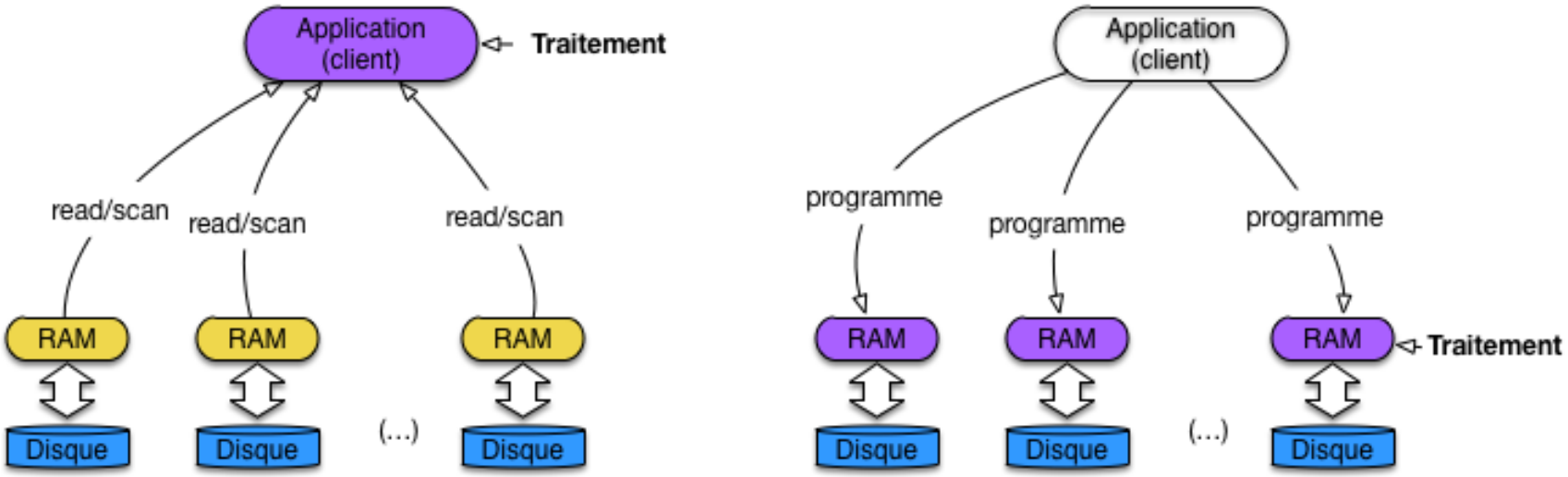
- Essentiellement des commandes POSIX (mkdir, ls, rm, cp, etc.)
- Mais également, des commandes propre à HDFS, copyFromLocal, copyToLocal, etc.
- Exemple : `hadoop fs -copyFromLocal file:///myfile.txt hdfs://localhost:9000/user/khaled/myfile.txt`
- `hdfs dfs...` uniquement pour les fichiers HDFS.
- Apache Ambari : interface web d'administration de Hadoop



Chapitre 3 - Systèmes NoSQL : calcul distribué avec MapReduce

Les programmes vers les données, pas l'inverse!

Client serveur pour traiter des TOs de données ? **Ne marche pas**.



Il faut placer les traitements au plus près des données.

Pourquoi un Framework

Un framework **pilote** un traitement, conduit selon un processus générique (notion d'inversion de contrôle).

L'application "injecte" des fonctions dans le framework.
Exemples : Applications MVC, XSLT, Couches ORM, **MapReduce**.

Le framework applique les fonctions pendant l'exécution du processus.

- Une fonction *map()* à appliquer pendant la phase de Map
- Une fonction *reduce()* à appliquer pendant la phase de Reduce.

Rôle d'un *framework* MapReduce

Prendre en charge la distribution, et gérer les reprises sur panne.

Comptons les mots : la fonction de Map

Phase de Map : on extrait les termes, on les compte localement, on transmet au framework.

```
function mapTF($id, $contenu)
{
  // $id: identifiant du document
  // $contenu: contenu textuel du document

  // On boucle sur tous les termes du contenu
  foreach ($t in $contenu) {
    // Comptons le nb d'occurrences de $t dans $contenu
    $count = nbOcc ($t, $contenu);
    // Emission du terme et de son nombre d'occurrences
    emit ($t, $count);
  }
}
```

NB : rien n'indique le contexte de distribution.

Comptons les mots : la fonction de Reduce

Phase de Reduce : on reçoit, pour chaque terme, tous les compteurs, et on les additionne.

```
function reduceTF($t, $compteurs)
{
  // $t: un terme
  // $compteurs: les nombres d'occurrences, un pour chaque doc.
  $total = 0;

  // Boucles sur les compteurs et calcul du total
  foreach ($c in $compteurs) {
   $total = $total + $c;
  }

  // Et on produit le total
  return $total;
}
```

Même remarque : rien n'indique le contexte de distribution.

L'exécution par le Framework

URL	Document
u_1	the jaguar is a new world mammal of the felidae family.
u_2	for jaguar, atari was keen to use a 68k family device.
u_3	mac os x jaguar is available at a price of us $199 for apple's new "family pack".
u_4	one such ruling family to incorporate the jaguar into their name is jaguar paw.
u_5	it is a big cat.

Déroulons le processus

term	count
jaguar	1
mammal	1
family	1
jaguar	1
available	1
jaguar	1
family	1
family	1
jaguar	2
...	

Sortie du *map*
Entrée du *shuffle*

Déroulons le processus

term	count
jaguar	1
mammal	1
family	1
jaguar	1
available	1
jaguar	1
family	1
family	1
jaguar	2
...	

Sortie du *map*
Entrée du *shuffle*

term	count
jaguar	1,1,1,2
mammal	1
family	1,1,1
available	1
...	

Sortie du *shuffle*
Entrée du *reduce*

Déroulons le processus

term	count
jaguar	1
mammal	1
family	1
jaguar	1
available	1
jaguar	1
family	1
family	1
jaguar	2
...	

Sortie du *map*
Entrée du *shuffle*

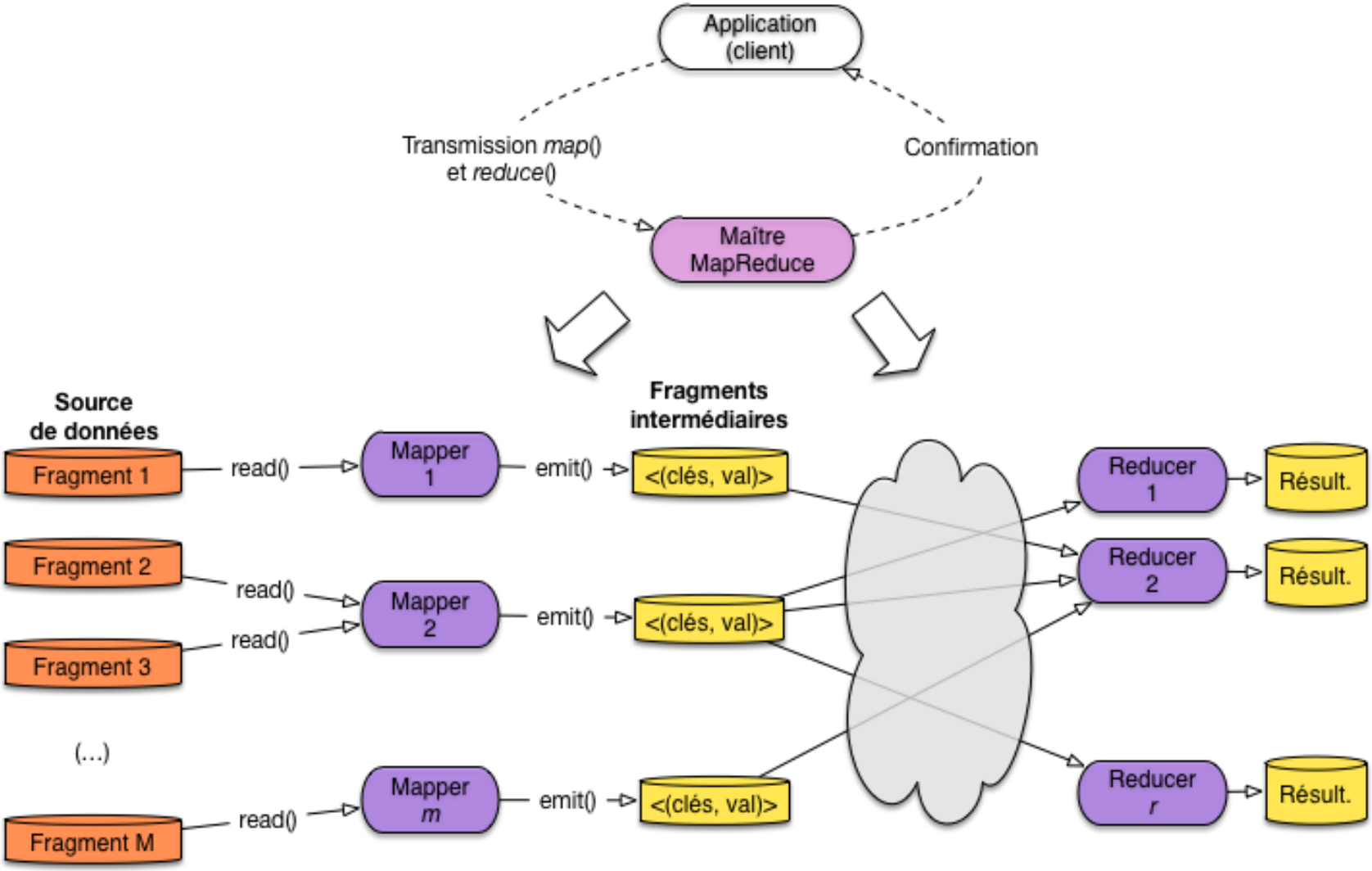
term	count
jaguar	1,1,1,2
mammal	1
family	1,1,1
available	1
...	

Sortie du *shuffle*
Entrée du *reduce*

term	count
jaguar	5
mammal	1
family	3
available	1
...	

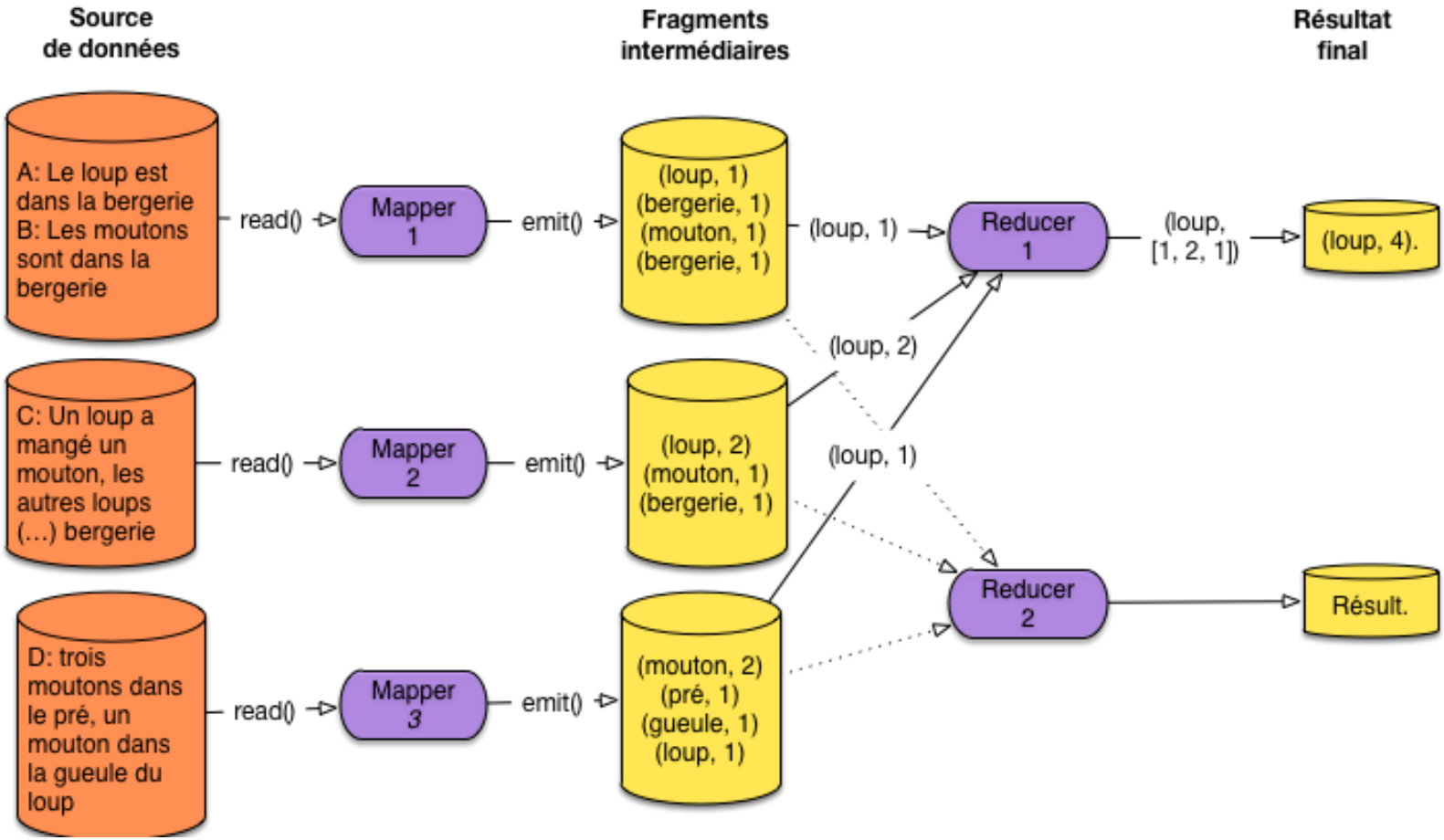
Résultat final

La grande vision



Un tout petit exemple

Quatre documents qui parlent de loups, de moutons, de bergerie.



Chapitre 3 - Systèmes NoSQL : calcul distribué avec MapReduce

Hadoop v1 JobTracker et TaskTracker

- Hadoop Map/reduce utilise 2 types de processus :
  - **JobTracker** : pour l'**orchestration** (surveillance/reprise sur panne) et l'**injection** des Map()/ Reduce() aux noeuds du cluster
  - **TaskTracker** : pour l'**exécution d'une tâche** (Map ou Reduce) **sur un noeud**
- Déroulement typique
  1. Les applications clientes soumettent les traitements Map/Reduce au JobTracker
  2. Le **JobTracker consulte le NameNode** et décide à quels noeuds **injecter les Map()**.
  3. Idéalement, les Map() sont affectés aux noeuds DataNodes stockant les données interrogées. Si un noeud DataNode est occupé, le Map() est affecté à un noeud se trouvant dans la même rack
  4. Chaque **TaskTracker exécute localement la fonction Map()** qui lui a été soumise
  5. Le TaskTracker **notifie périodiquement** (par heartbeat) le JobTracker de sa présence
  6. Si un TaskTracker échoue (n'envoie pas de notification), la tâche est affectée à un autre noeud
  7. Une fois la phase Map est terminé, le JobTracker **injecte les Reduce()** aux noeuds disponibles. Chaque noeud a un nombre de slots fixe dans lesuels sont mises les tâches en attente.
- **Le JobTracker est un SPOF** du Map/Reduce de Hadoop v1. Cette limitation a été résolue dans Hadoop v2.0 (YARN)

Chapitre 3 - Systèmes NoSQL : calcul distribué avec MapReduce

Gestion des pannes

Un traitement MapReduce peut durer des jours, sur des centaines de machines : la probabilité de panne est très grande.

Ré-exécuter le traitement complètement en cas de panne ⇒ on est à peu près sûr de ne jamais terminer.

Dans Hadoop, le Maître (*JobTracker*) surveille l'ensemble du processus (tâches de Map, tâches de Reduce).

1. Panne d'un Reducer : on peut reprendre à partir du résultat des Mappers, **stocké sur disque**.
2. Panne d'un Mapper : on recommence la tâche sur le même fragment (**ou sur un réplica**)
3. Si le Maître tombe en panne : on recommence tout, mais la probablité est très faible.

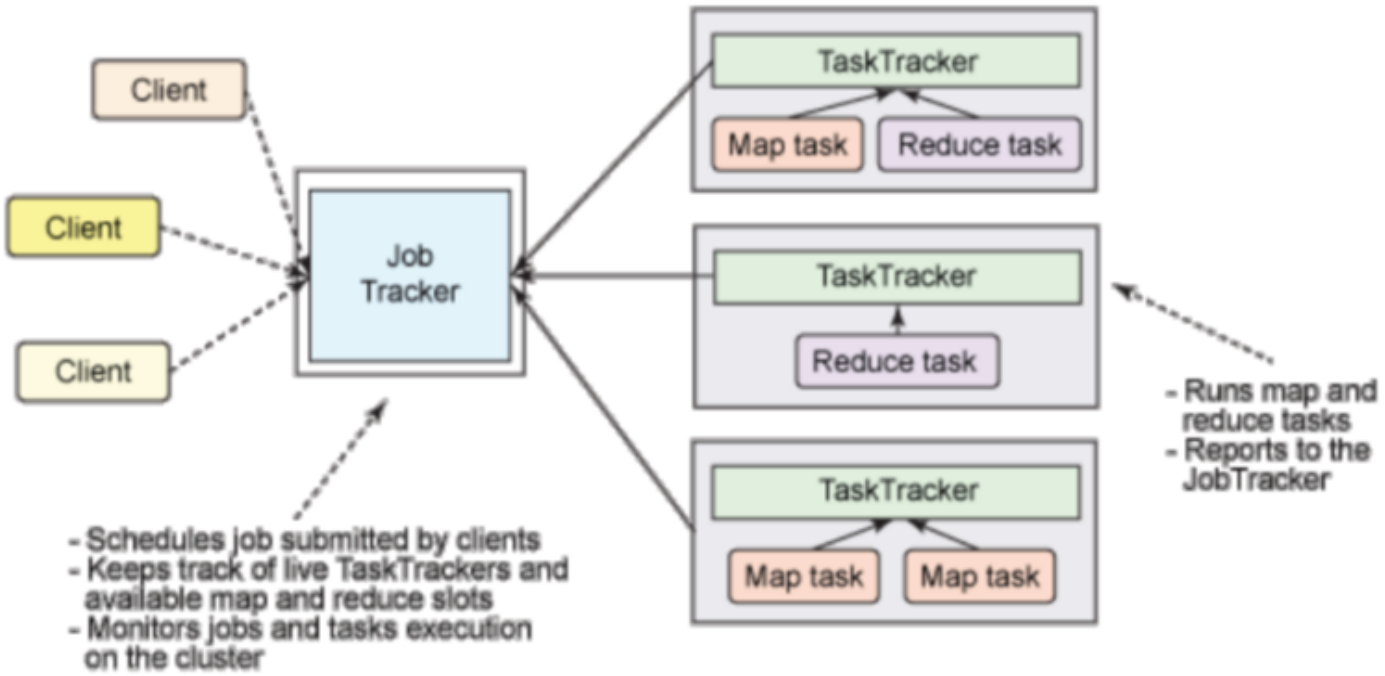
Essentiel

Tout repose sur une **sérialisation** (écriture sur disque) aux différentes étapes. **Robuste mais très lent**.

Hadoop v1 JobTracker et TaskTracker

Architecture of MRv1

- Classic version of MapReduce (MRv1)



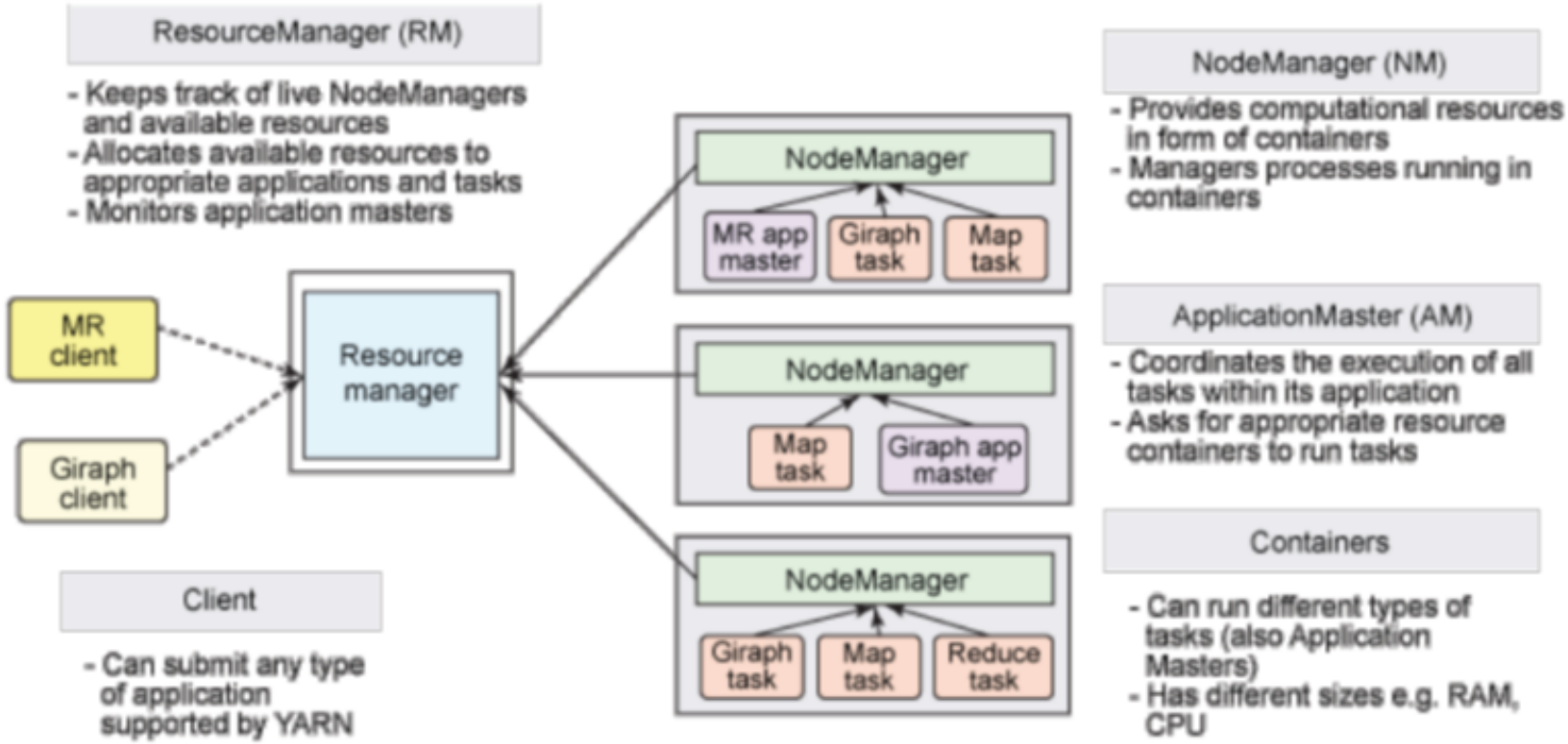
Hadoop v2 YARN

- JobTracker et TaskTrakers sont remplacés par : (JobTracker) ResourceManager, ApplicationMaster et (TaskManagers) NodeManagers
- **ResourceManager** : le noeud maître de YARN (un par cluster)
  - Responsable de la collecte des informations sur les ressources (RAM et CPU) disponibles. Exécute plusieurs services dont le plus important est le scheduler
  - **Scheduler** : le scheduler YARN est respondable de l'allocation des ressources aux applications tournant sur Hadoop. Plusieurs stratégies (FIFO, Fair, Capacity, etc.)
- **ApplicationMaster** (un par application)
  - Chaque application a un AM qui lui est propre, tournant sur dans un conteneur indépendant.
  - l'AM envoie périodiquement des heartbeats au ResourceManager et lui réclame des ressources supplémentaires au besoin.
- **NodeManagers** (un par noeud)
  - Gère les ressources au niveau d'un noeud (il en existe par noeud).
  - Créé les containers au niveau d'un noeud et notifie périodiquement le RM d'informations sur l'utilisation des ressources au niveau du noeud.
  - Remplace les TaskTtrackers. Alors qu'un tasktracker gère un nombre fixe de slots, un NM gère des containers dynamiquement créés. Contrairement aux slots, les containers peuvent être utilisés pour des tâches map, des tâches reduce et des tâches d'autres frameworks que Map/Reduce.

Hadoop v2 YARN : déroulement typique de l'exécution d'une application

YARN architecture

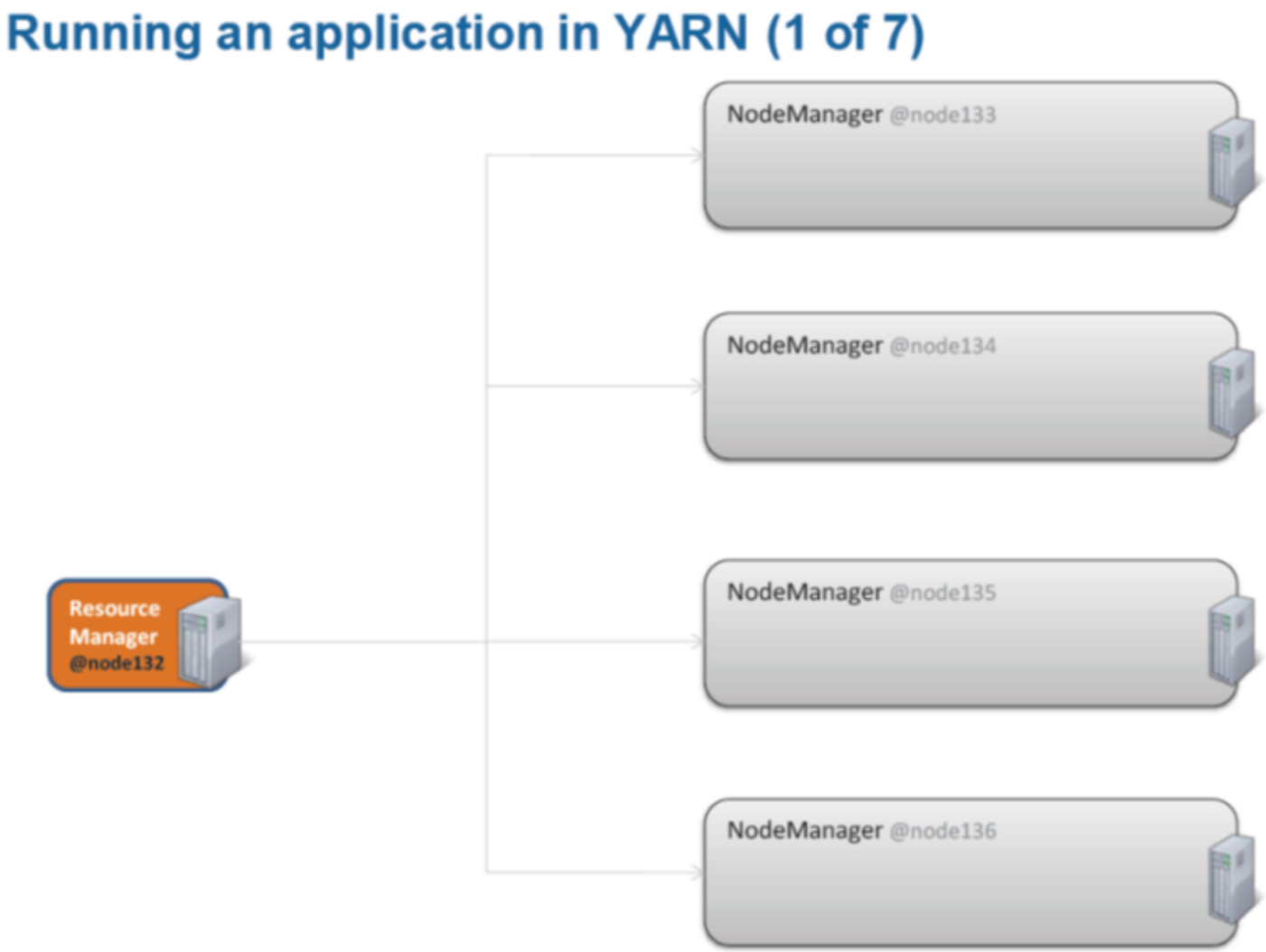
- High level architecture of YARN



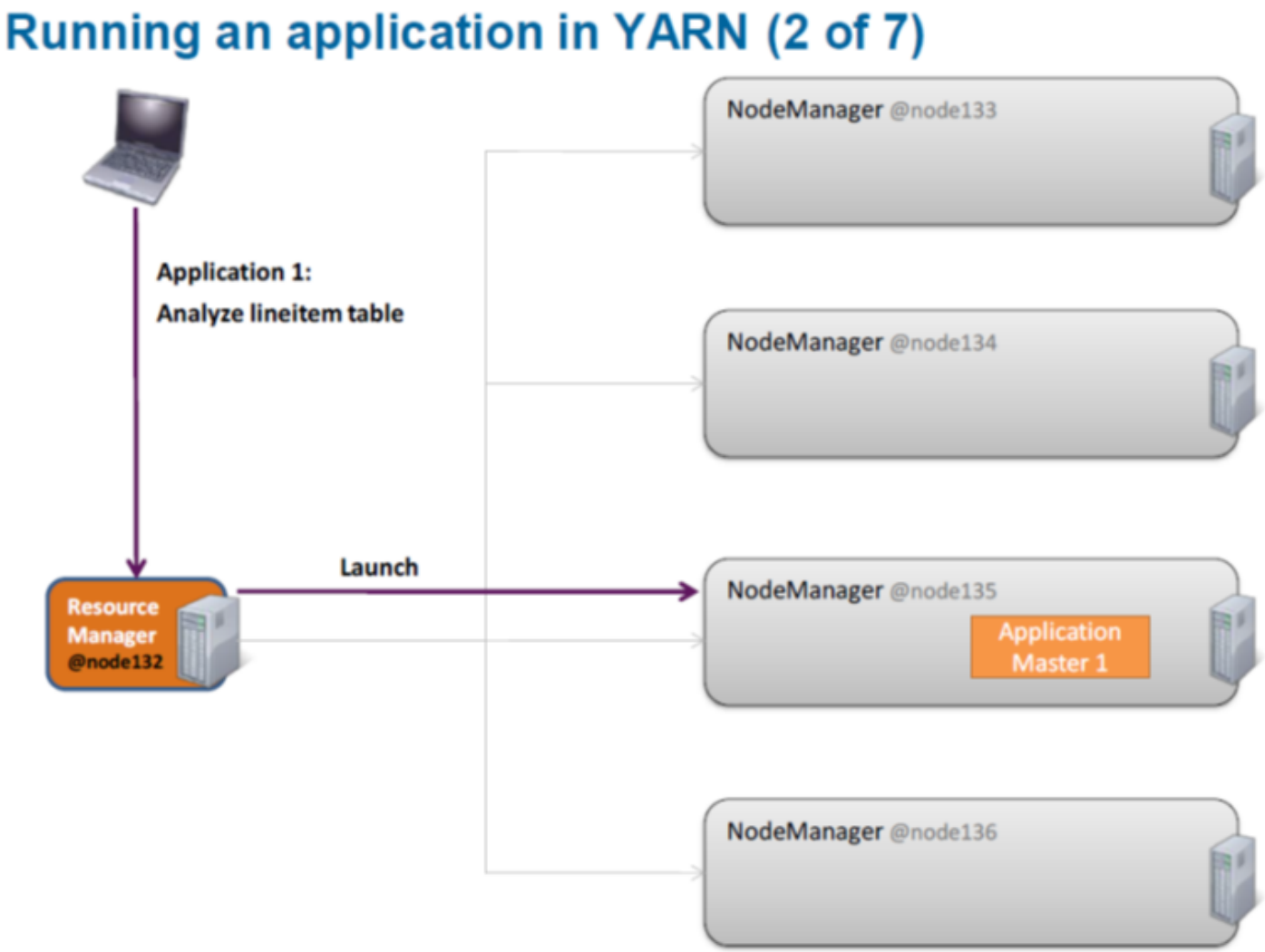
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

1. Le client soumet une application MapReduce au ResourceManager
2. Le ResourceManager négocie un container pour l'ApplicationMaster et lance l'ApplicationMaster.
3. L'ApplicationMaster démarre et s'enregistre auprès du ResourceManager.
4. L'ApplicationMaster négocie des ressources (*resource containers*) pour l'application cliente.
5. Le NodeManager lance des containers pour l'application.
6. Lors de l'exécution, le client notifie l'ApplicationMaster de la progression de l'exécution. Une fois l'exécution terminée, l'ApplicationMaster s'arrête et libère les containers associés à l'application.

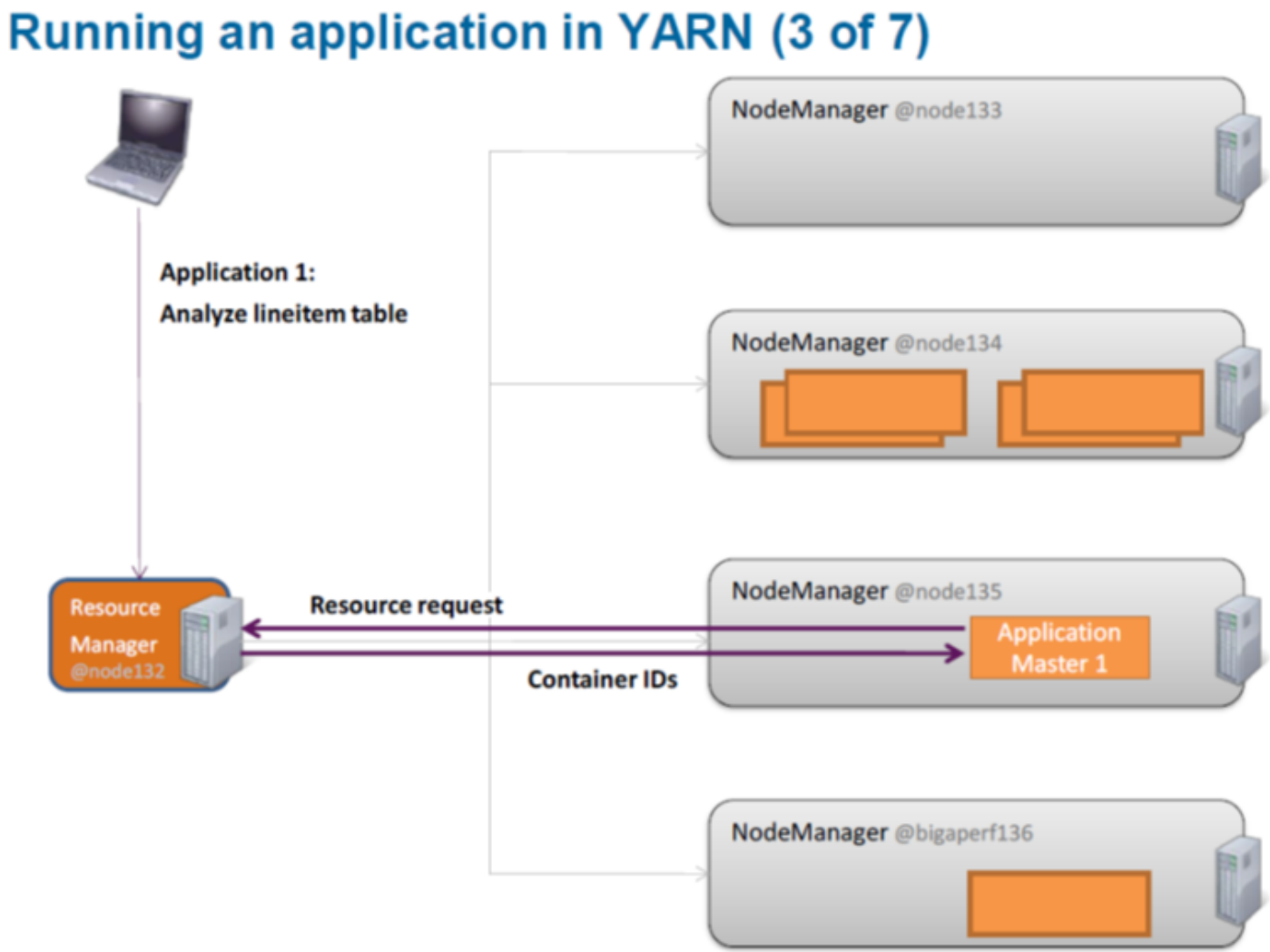
Hadoop v2 YARN : déroulement typique de l'exécution d'une application



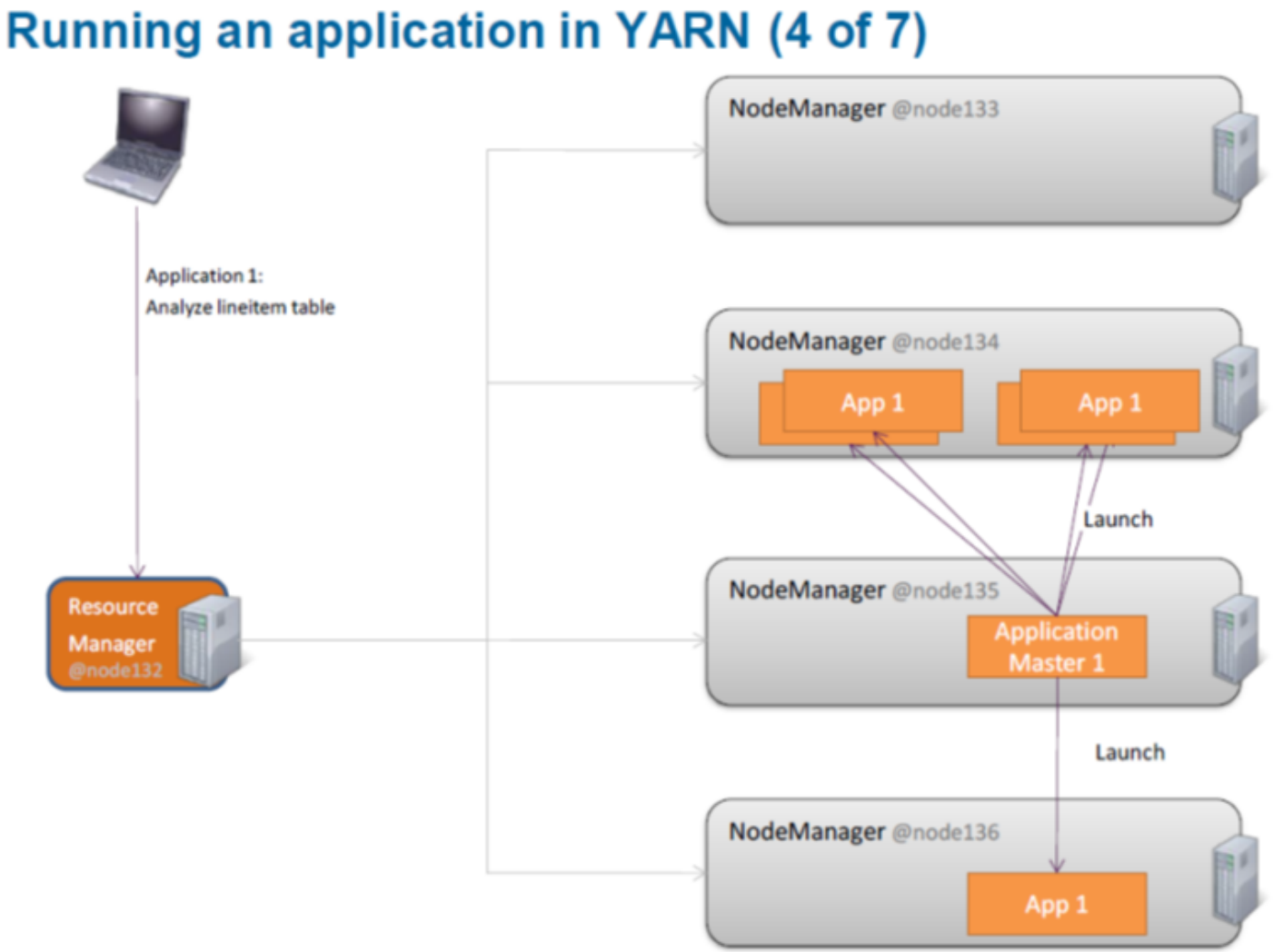
Hadoop v2 YARN : déroulement typique de l'exécution d'une application



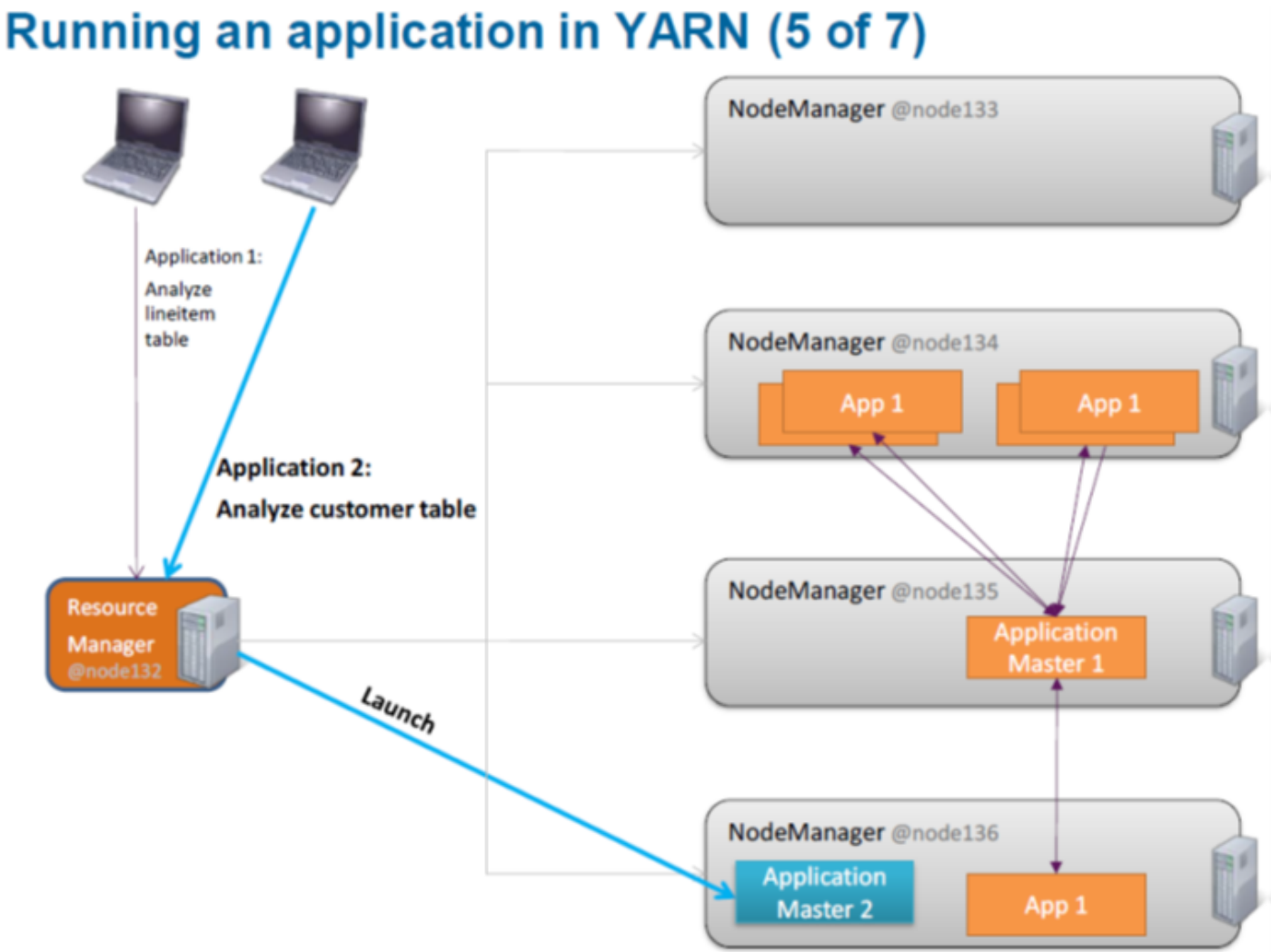
Hadoop v2 YARN : déroulement typique de l'exécution d'une application



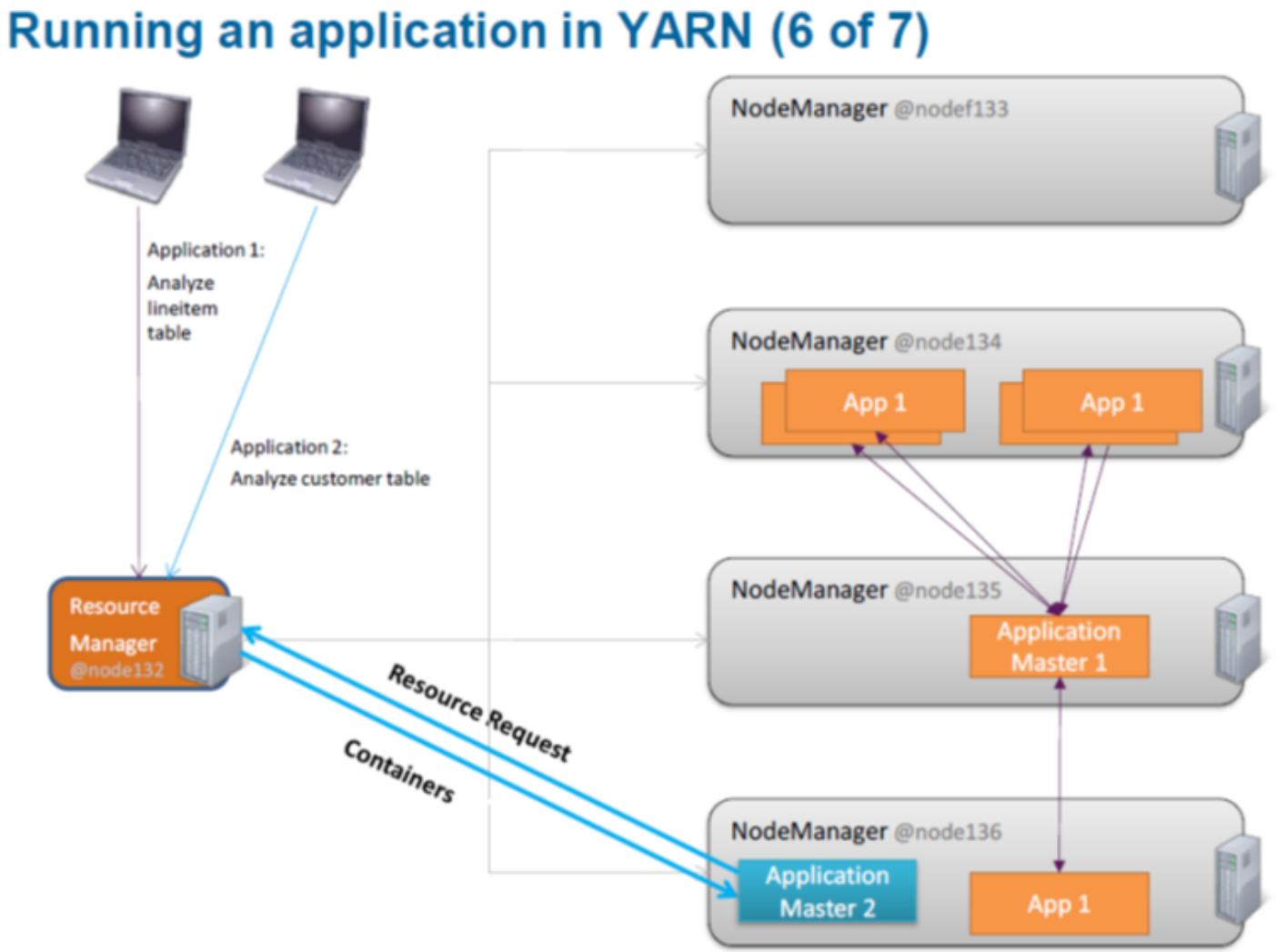
Hadoop v2 YARN : déroulement typique de l'exécution d'une application



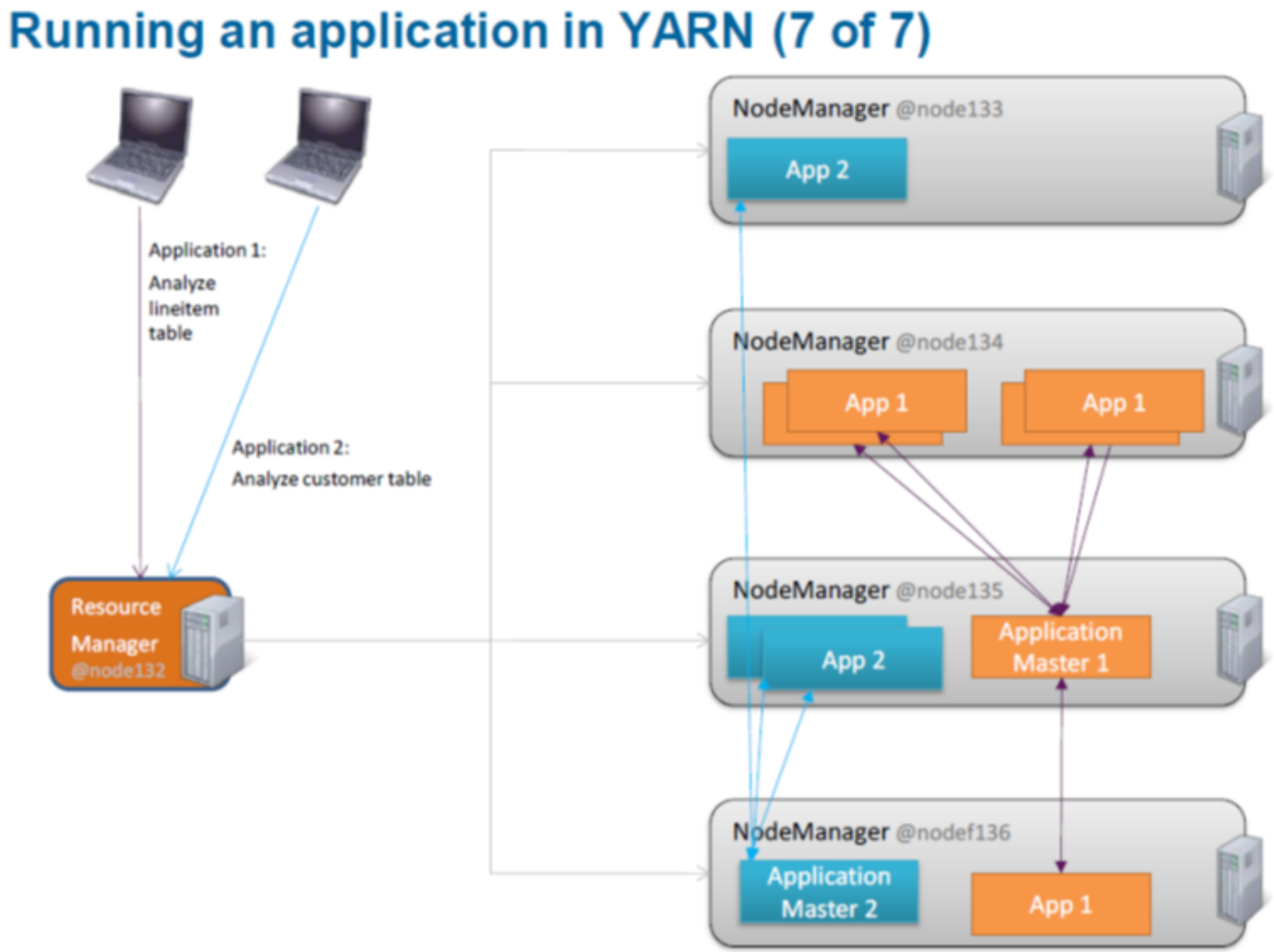
Hadoop v2 YARN : déroulement typique de l'exécution d'une application



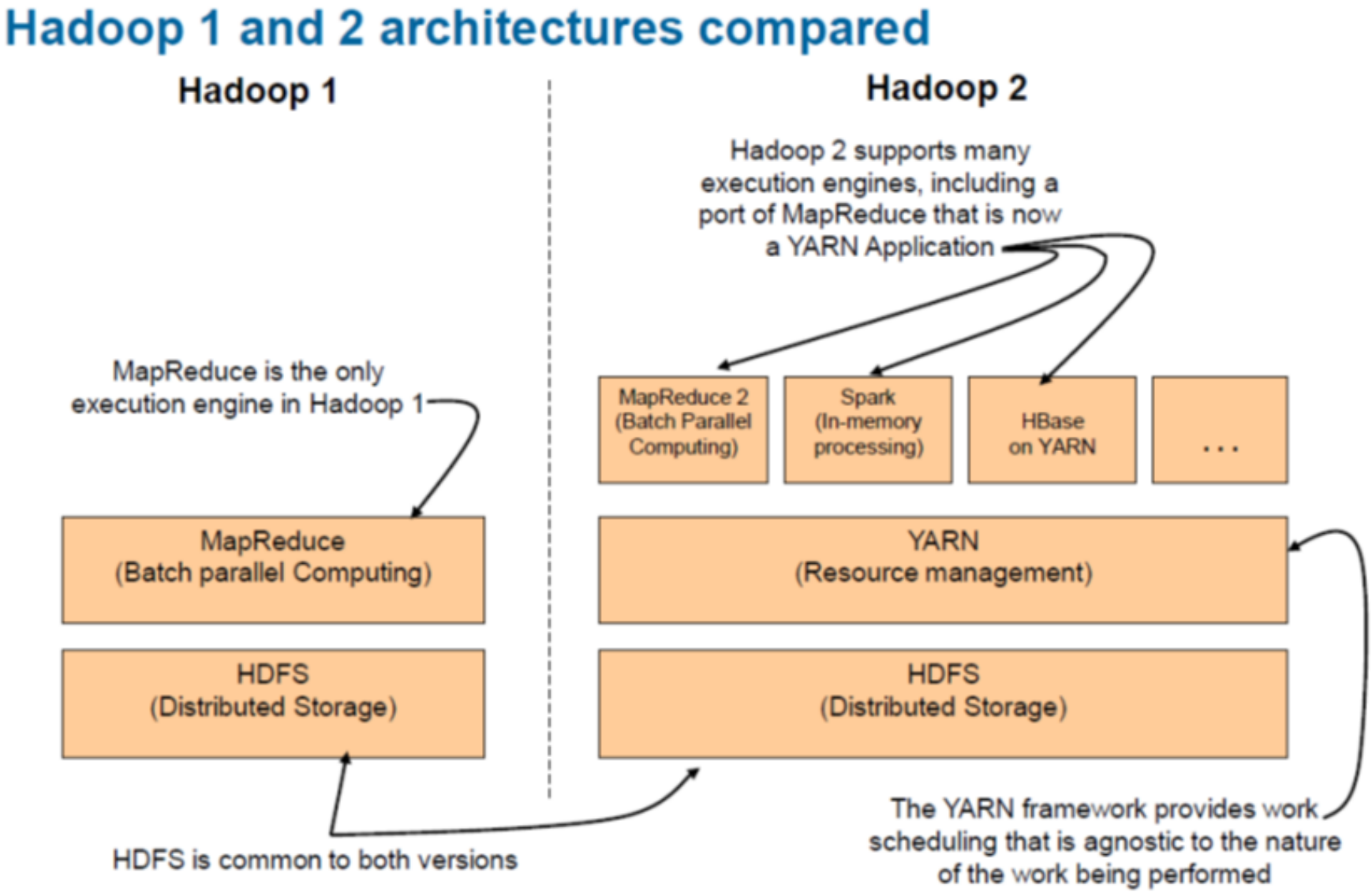
Hadoop v2 YARN : déroulement typique de l'exécution d'une application



Hadoop v2 YARN : déroulement typique de l'exécution d'une application



YARN vs MapReduce



Chapitre 4 - Systèmes NoSQL et transactions

- Pig et Hive, des langages de haut niveau