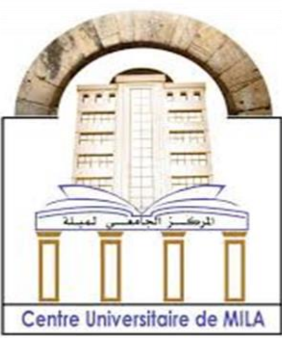


REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Département d'informatique

Master 2 I2A

CHAPITRE 02: HADOOP

Année Universitaire:2024/2025

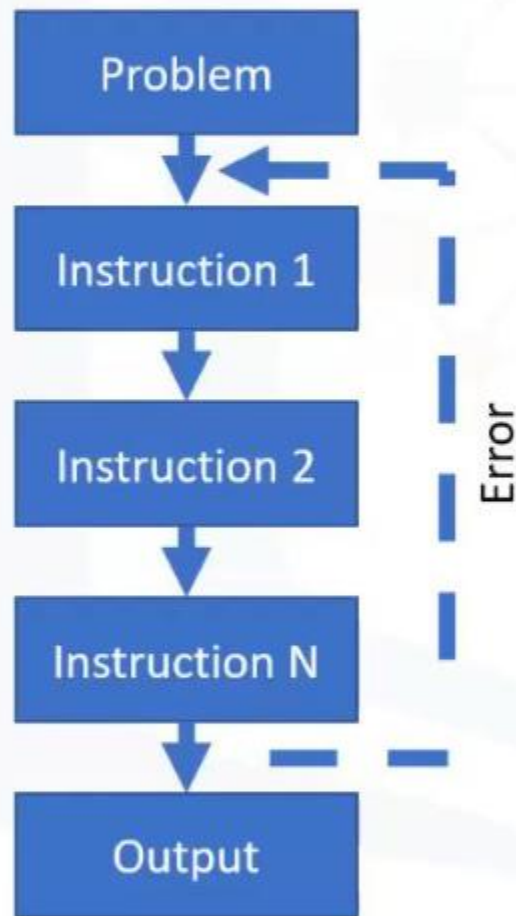
Plan

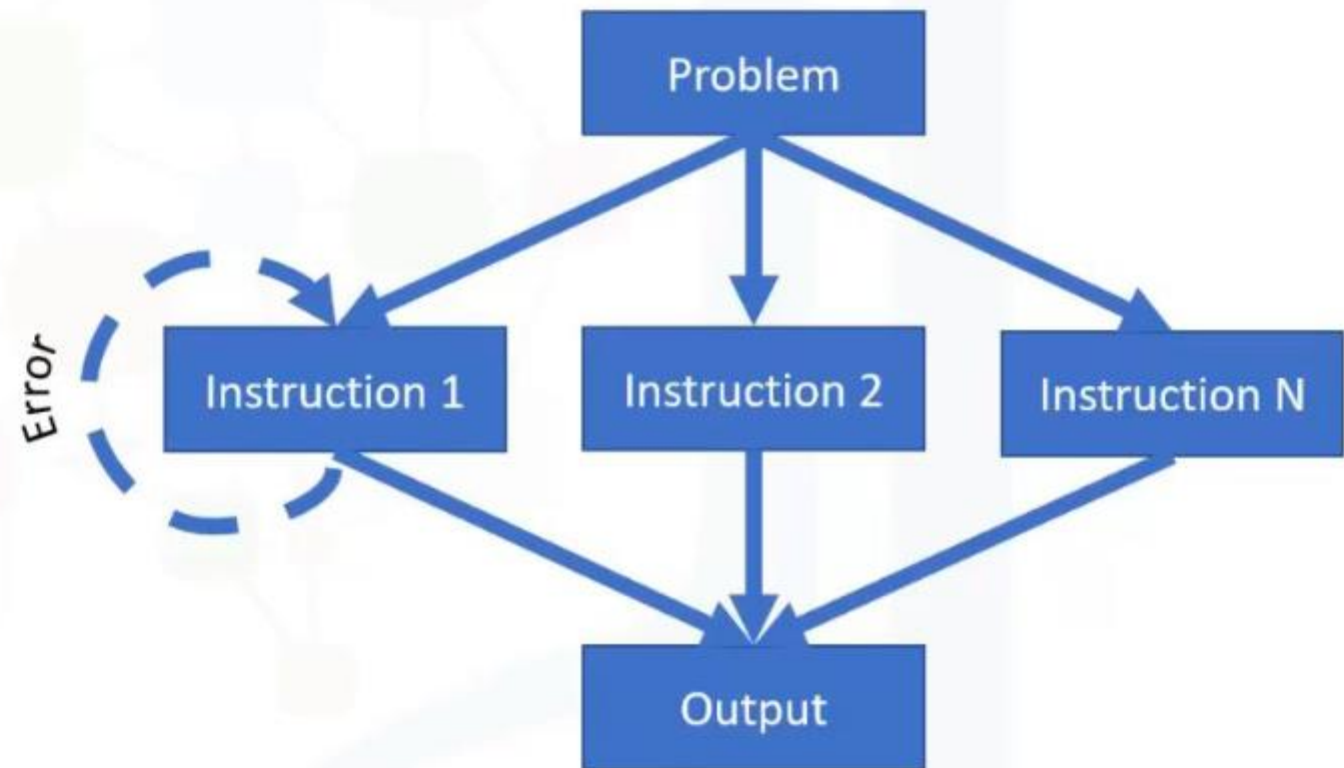
- Traitement à grand échelle
- Présentation Hadoop
- MapReduce
- Exemples d'applications du modèle MapReduce
- HDFS
- Architecture Hadoop
- Programmation Hadoop

Histoire du traitement à grande échelle

- Matériel spécialisé
- Extrêmement coûteux
- Créé pour résoudre des problèmes spécifiques

Linear vs. parallel processing





Pourquoi le traitement parallèle ?

● **Réduire le temps de traitement** : l'approche de traitement parallèle peut traiter de **grands** ensembles de **données** en une **fraction** de temps.

● **Faible besoin** : moins de besoins en **mémoire** et en **calcul**, car les ensembles d'instructions sont **distribués** à des nœuds d'exécution plus petits.

● **Flexibilité** : davantage de **nœuds** d'exécution peuvent être **ajoutés** ou **supprimés** du réseau de traitement en fonction de la **complexité** du problème.

Hadoop: Problèmatique ?

● Avoir un **framework** déjà disponible, **facile à déployer**, et qui permette l'exécution de **tâches parallélisables** – et le **support** et le **suivi** de ces **tâches** – de manière **rapide** et **simple** à mettre en œuvre.

● L'idée étant d'avoir **un outil** qui puisse être **installé** et **configuré rapidement** au sein d'une **entreprise**/d'une **université** et qui permettre à des **développeurs** d'**exécuter** des tâches distribuées avec un **minimum** de **formation requise**.

● L'outil en **question** devant être facile à **déployer**, **simple** à supporter, et pouvant permettre la **création** de **clusters** de taille variables **extensibles** à tout moment.

Hadoop

- **Hadoop** est un framework **open source développé** en **Java** utilisé pour traiter d'***énormes*** ensembles de données.

- Ensemble de **programmes** et **d'outils open source**.

- Utile pour **traiter de grands ensembles de données**

- Gère les **tâches** ou les **processus parallèles**

- **Les composantes principales de la plateforme sont: HDFS et MapReduce**

- **Hadoop est un éco-système avec 100+ de sous-projets dont: Hbase, Hive, Pig; Sqoop, Zookeeper, Impala, ….**

Pourquoi Hadoop ?

- **Traitement des données structurées/non structurées**
- **Traitement des données volumineuses**
- **La tolérance aux pannes est garantie.**
- **Aucune perte de données**
- **Duplication des données**
- **Modèle simple pour les développeurs: il suffit de développer des tâches map-reduce, depuis des interfaces simples accessibles via des librairies dans des langages multiples (Java, Python, C/C++...).**

Qui utilise Hadoop ?

YAHOO!

ebay

facebook

amazon.com

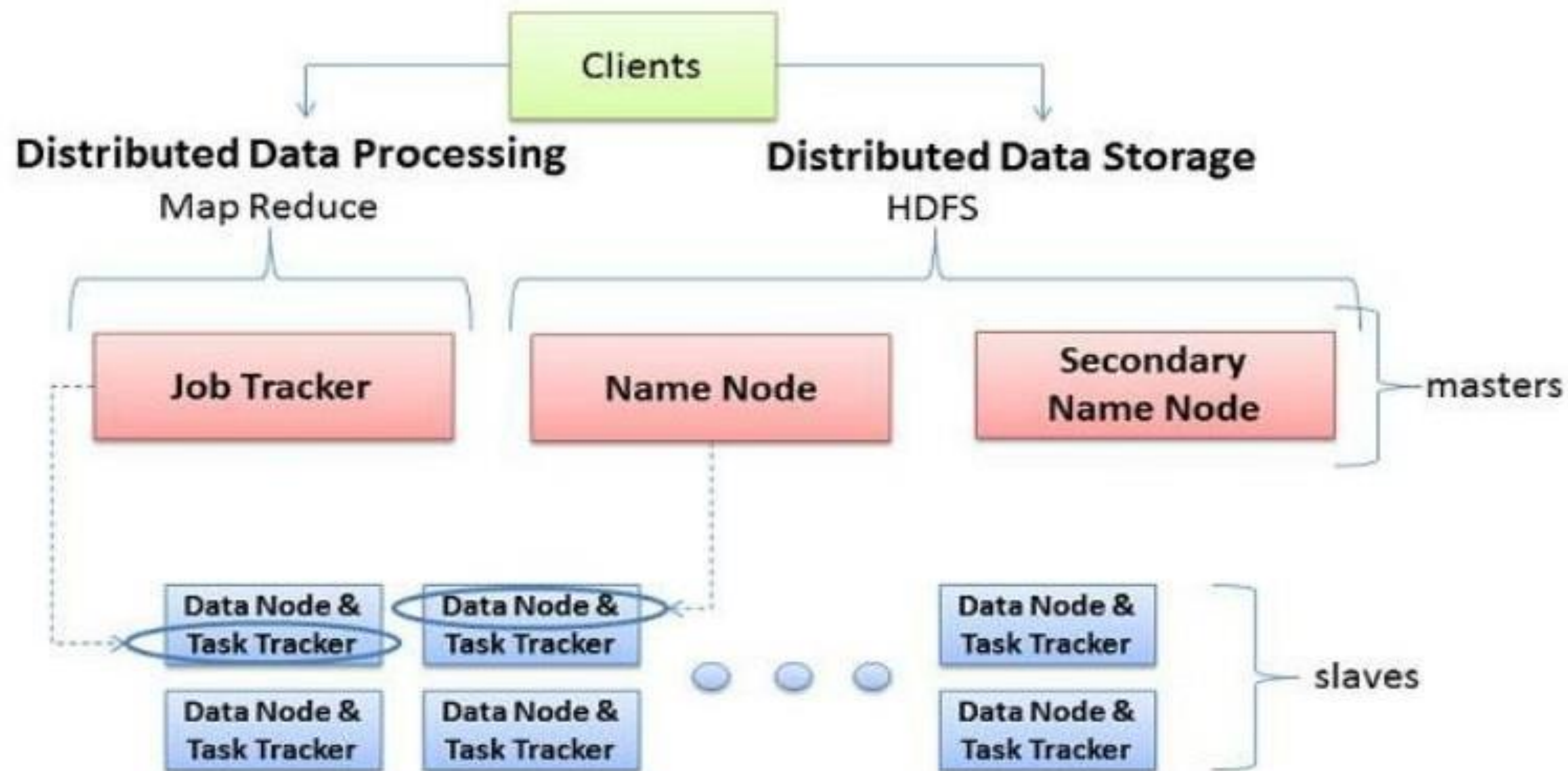
Google

Berkeley
UNIVERSITY OF CALIFORNIA

Linkedin

… et des centaines d'entreprises et universités à travers le monde.

Architecture globale



MapReduce (1)

- **Pour exécuter un problème large de manière distribué:**
  - ❑ **il faut pouvoir découper le problème en plusieurs problèmes de taille réduite à exécuter sur chaque machine du cluster.**

- **Modèle de programmation** utilisé dans **Hadoop** pour le **traitement du BigData**

- **Technique de traitement** pour **l'informatique distribuée**

- **Comprend les tâches : Map et Reduce**

- **Peut être codé dans de nombreux langages comme Java, C++, Python, Ruby et R.**

MapReduce (2)

On distingue donc 4 étapes distinctes dans un traitement MapReduce:

- **Découper (split) les données d'entrée en plusieurs fragments.**
- **Mapper chacun de ces fragments pour obtenir des couples (clef ; valeur).**
- **Grouper (shuffle) ces couples (clef ; valeur) par clef.**
- **Réduire (reduce) les groupes indexés par clef en une forme finale, avec une valeur pour chacune des clefs distinctes.**

MapReduce (3)

Pour résoudre un problème via la méthodologie MapReduce avec Hadoop, on devra donc:

● Choisir une manière de découper les données d'entrée de telle sorte que l'opération MAP soit parallélisable.

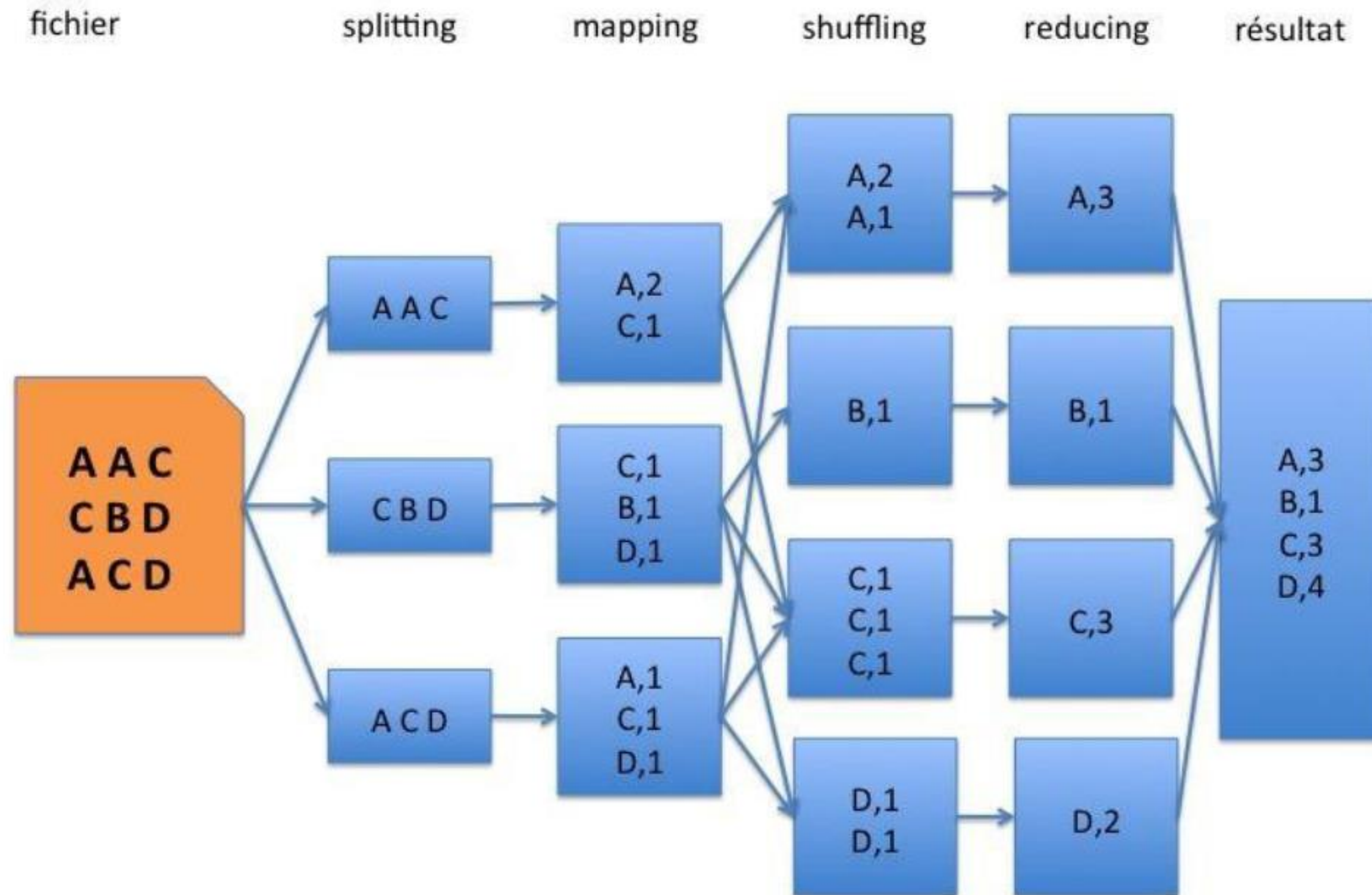
● Définir quelle CLEF utiliser pour notre problème.

● Écrire le programme pour l'opération MAP.

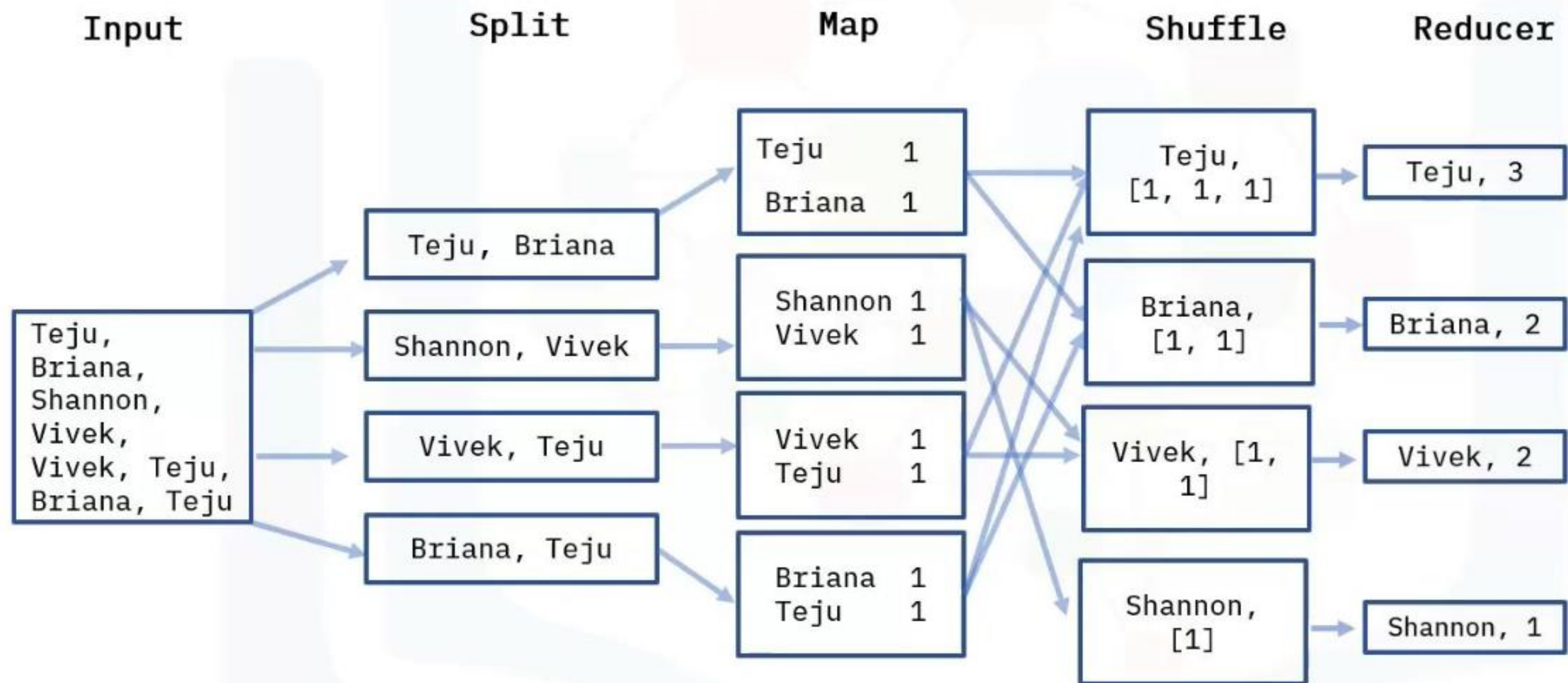
● Ecrire le programme pour l'opération REDUCE.

… et Hadoop se chargera du reste (problématiques calcul distribué, groupement par clef distincte entre MAP et REDUCE, etc.).

Exemple compteur des lettres en MapReduce



Exemple compteur des mots en MapReduce



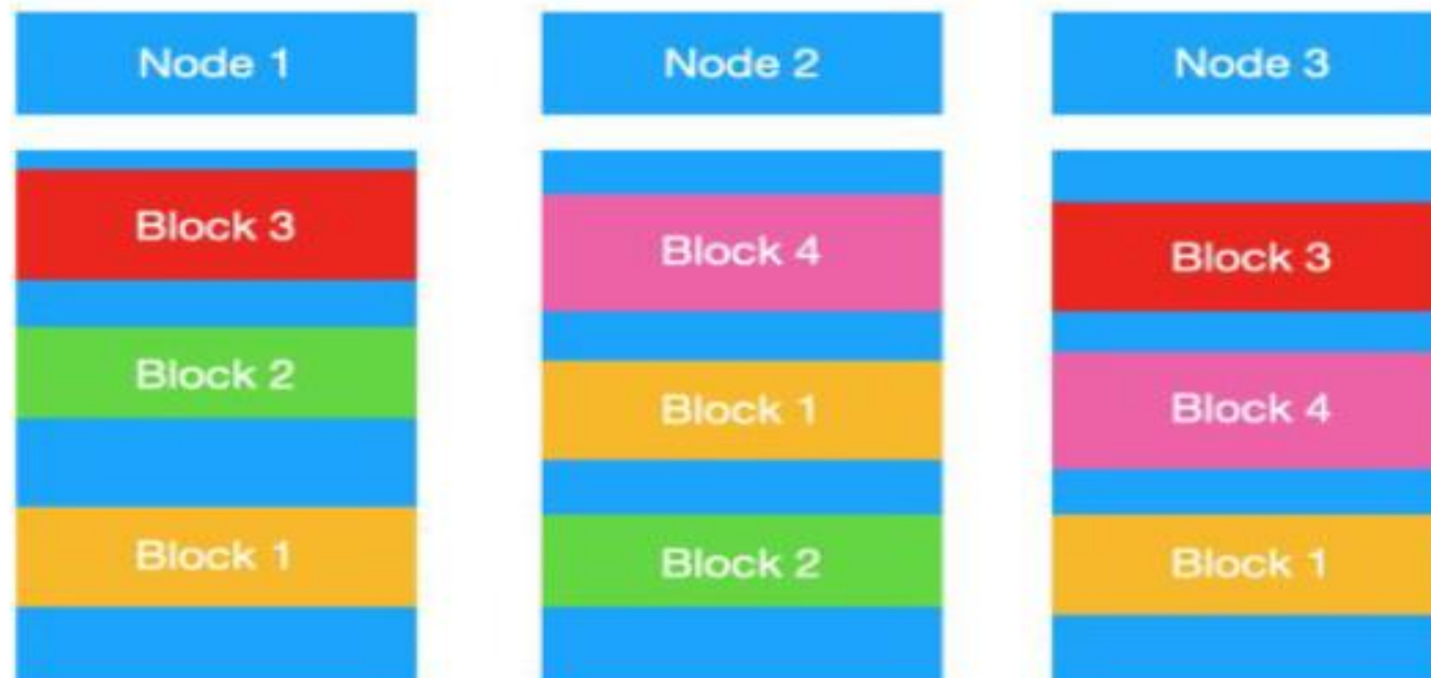
HDFS

- Stockage de données distribué
- Inspiré par le Google FileSystem (GFS)
- Haute disponibilité / réplication des données
- Blocs de 64Mo (chunks)
- Utilise TCP/IP et RPC

Hôtes HDFS

- Architecture Maître / Esclave
- NameNode : Gère l'espace de noms du système de fichiers (serveurs esclaves) et l'accès aux fichiers par les clients
- DataNode : Gère le stockage des fichiers sur un noeud (création / suppression de blocs de données, réplication)
- Réplication configurable

HDFS: Distribution de morceaux des données



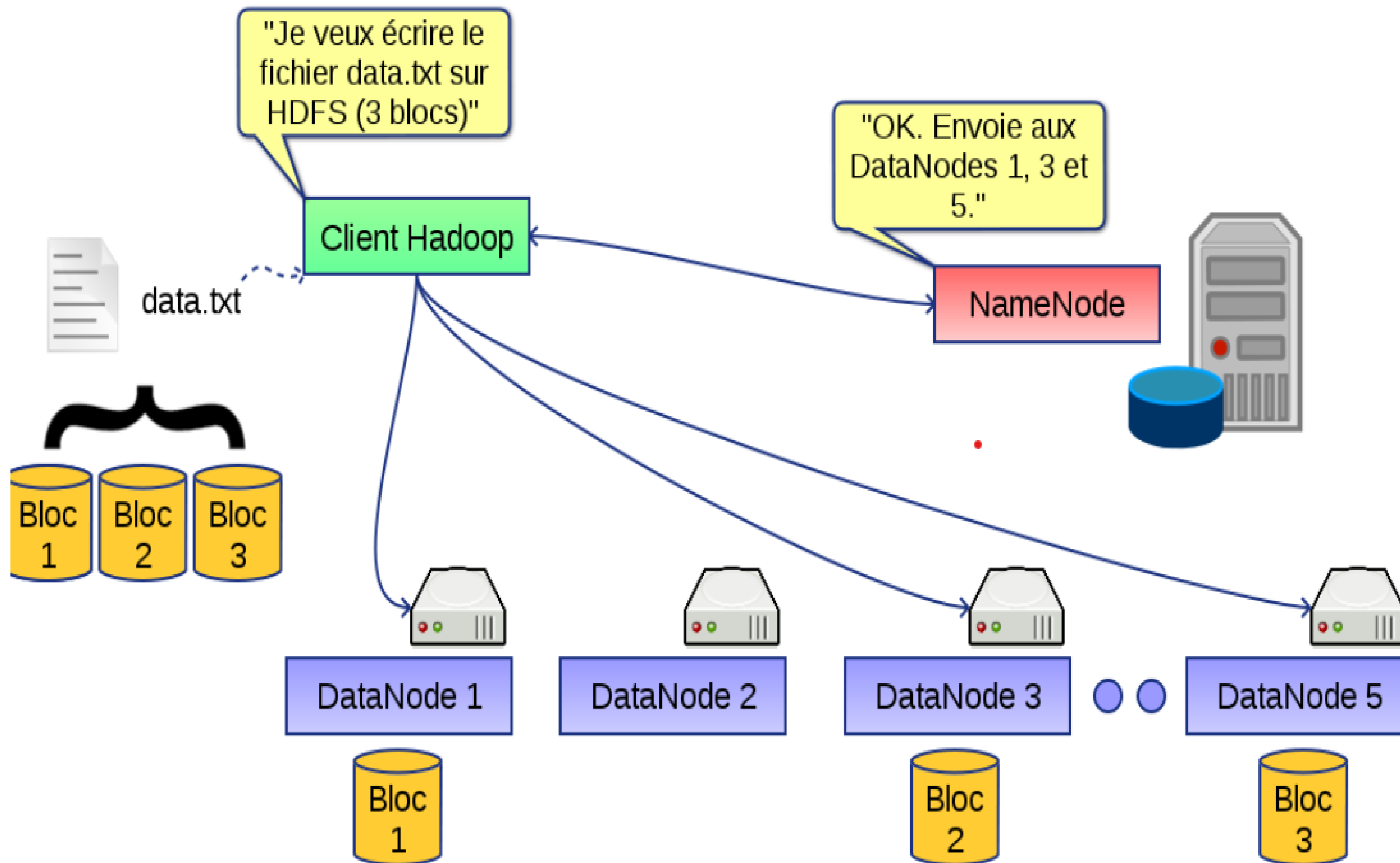
HDFS

HDFS repose sur deux serveurs (des daemons):

- **Le NameNode, qui stocke les informations relatives aux noms de fichiers. C'est ce serveur qui, par exemple, va savoir que le fichier « livre_5321 » dans le répertoire « data_input » comporte 58 blocs de données, et qui sait où ils se trouvent. Il y a un seul NameNode dans tout le cluster Hadoop.**

- **Le DataNode, qui stocke les blocs de données eux-mêmes. Il y a un DataNode pour chaque machine au sein du cluster, et ils sont en communication constante avec le NameNode pour recevoir de nouveaux blocs, indiquer quels blocs sont contenus sur le DataNode, signaler des erreurs, etc...**

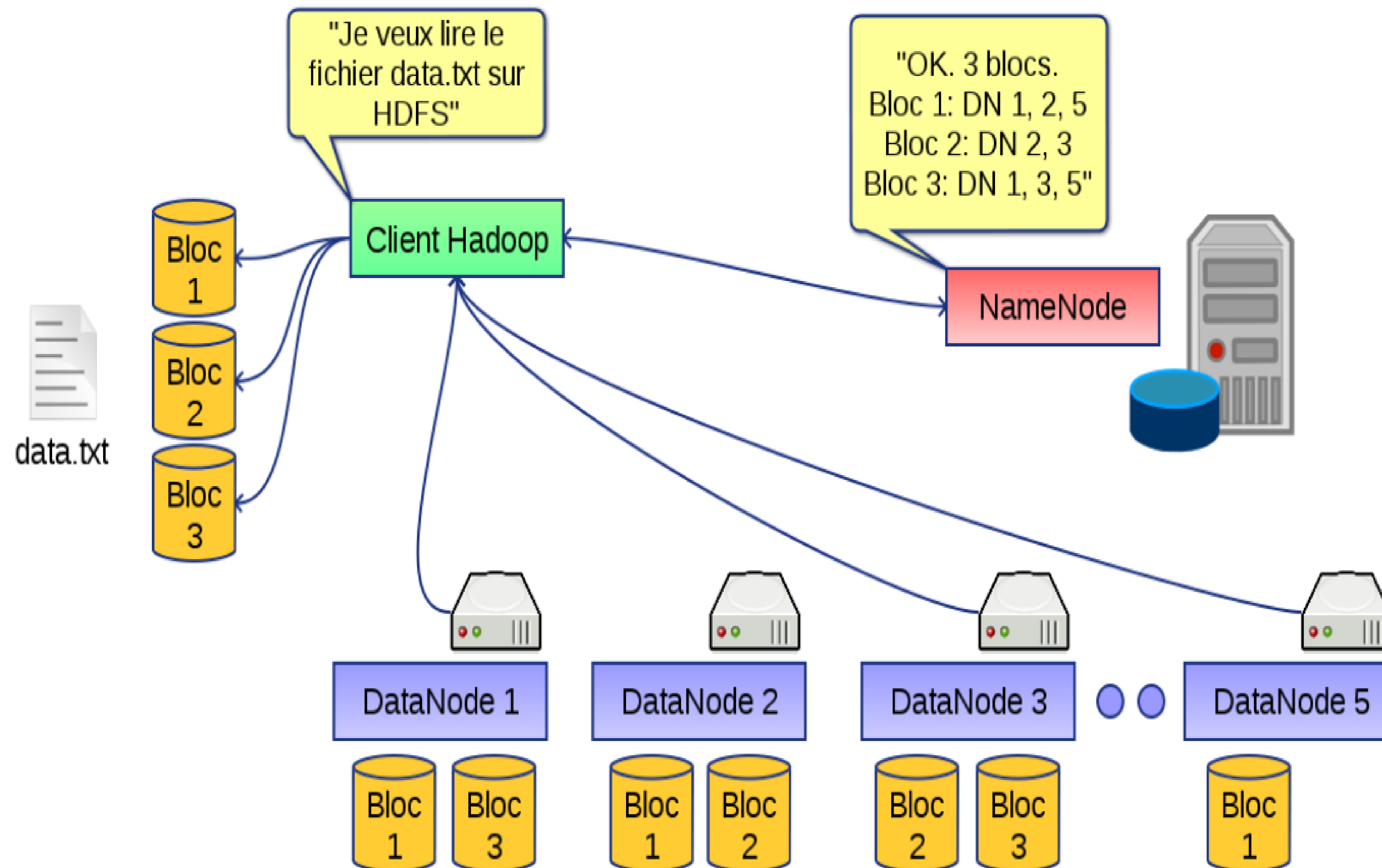
Ecriture d’un fichier sur HDFS



- Le client indique au NameNode qu'il souhaite écrire un bloc.
- Celui-ci lui indique le DataNode à contacter.
- Le client envoie le bloc au Datanode.
- Les DataNodes répliquent le bloc entre eux.
- Le cycle se répète pour le bloc suivant.

Lecture d’un fichier HDFS

Lecture HDFS



- Le client indique au NameNode qu'il souhaite lire un fichier.
- Celui-ci lui indique sa taille et les différents DataNode contenant les N blocs.
- Le client récupère chacun des blocs à un des DataNodes.
- Si un DataNode est indisponible le client le demande à un autre.

Architecture Hadoop (1)

On distingue dans les slides qui suivent deux architectures:

● La première, constitue du JobTracker et du TaskTracker, correspond à la première version du moteur d'exécution map/reduce (« MRv1 », Hadoop 1.x) et est présentée à titre informatif et pour des raisons historiques.

● A partir de la version 2.x, Hadoop intègre un nouveau moteur d'exécution: Yarn (« MRv2 »); cette architecture d'exécution, l'actuelle, est également présentée dans un second temps.

Architecture Hadoop (2)

Comme pour HDFS, la gestion des tâches de Hadoop se basait jusqu'en version 2 sur deux serveurs (des daemons):

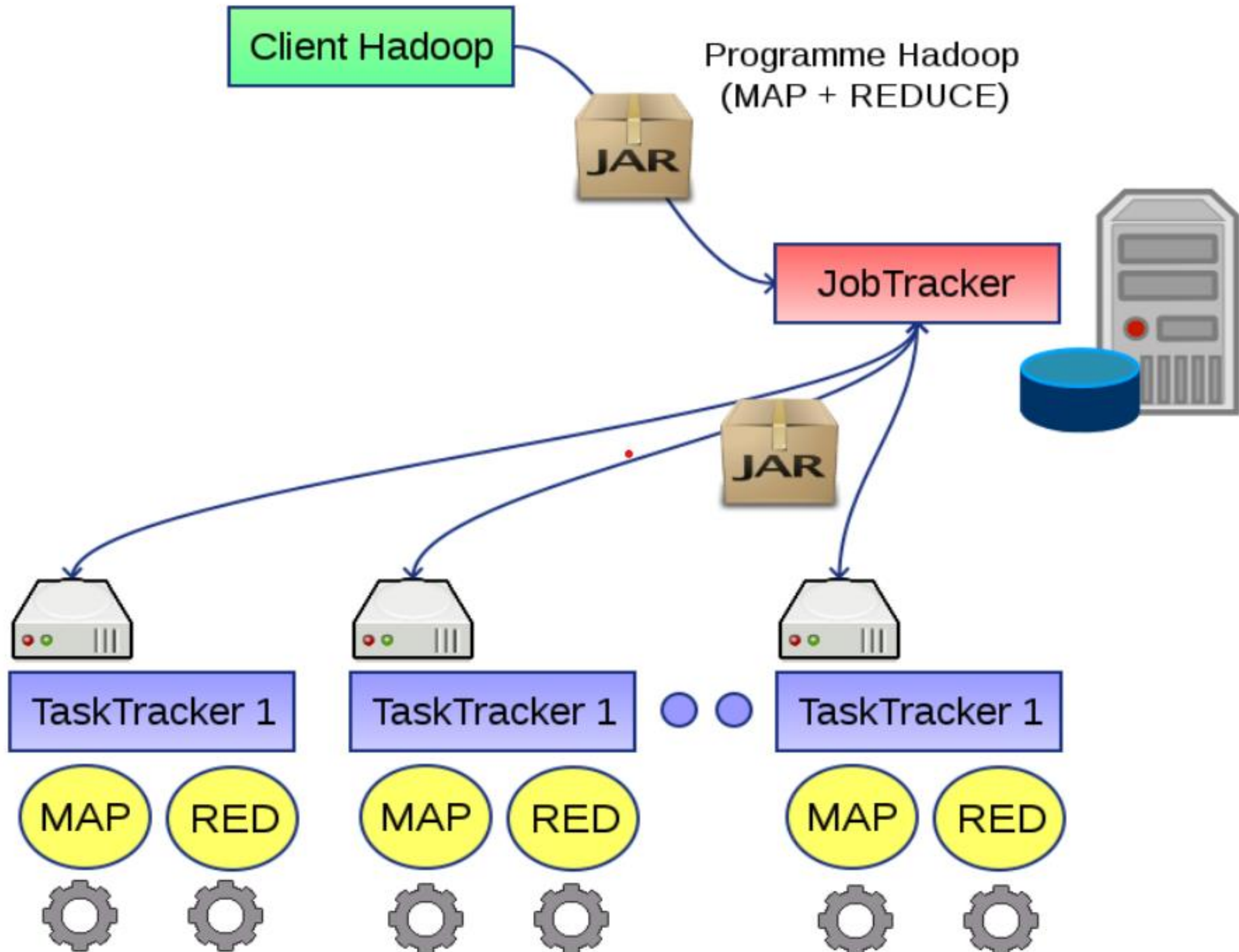
● Le JobTracker, qui va directement recevoir la tâche à exécuter (un .jar Java), ainsi que les données d'entrées (nom des fichiers stockés sur HDFS) et le répertoire où stocker les données de sortie (toujours sur HDFS).

- 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.

● Le TaskTracker, qui 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.

Architecture Hadoop (3): MRv1



Architecture Hadoop (4): Limitation du MRv1

Le JobTracker doit gérer tous les jobs en cours d'exécution, en plus de choisir les ressources des nouveaux jobs soumis. Au final, le JobTracker constitue un goulot d'étranglement quand la taille du cluster et le nombre de jobs augmentent.

- **Une seconde version d'Hadoop a donc été développée avec une couche plus évoluée de gestion des applications distribuées,**

- En **2010 Yahoo** ! a démarré le développement d'une nouvelle couche d'exécution et de contrôle des applications **Hadoop** au dessus d'HDFS. Cette version fut appelé **YARN** : ***Yet Another Resource Negotiator***.
- Elle distingue clairement :
  - les fonctionnalités de gestion et **d'allocation de ressources** de calculs, c'est-à-dire l'identification des **noeuds** de données qui hébergeront les **traitements**,
  - les fonctionnalités de lancement et de **monitoring** des tâches d'une application distribuée, sur les ressources allouées.
- Comparé à la version *1*, le **JobTracker** et les **TaskTrackers** disparaissent, au profit d'un **ResourceManager** et de plusieurs **NodeManager** (si on considère des applications Map-Reduce).

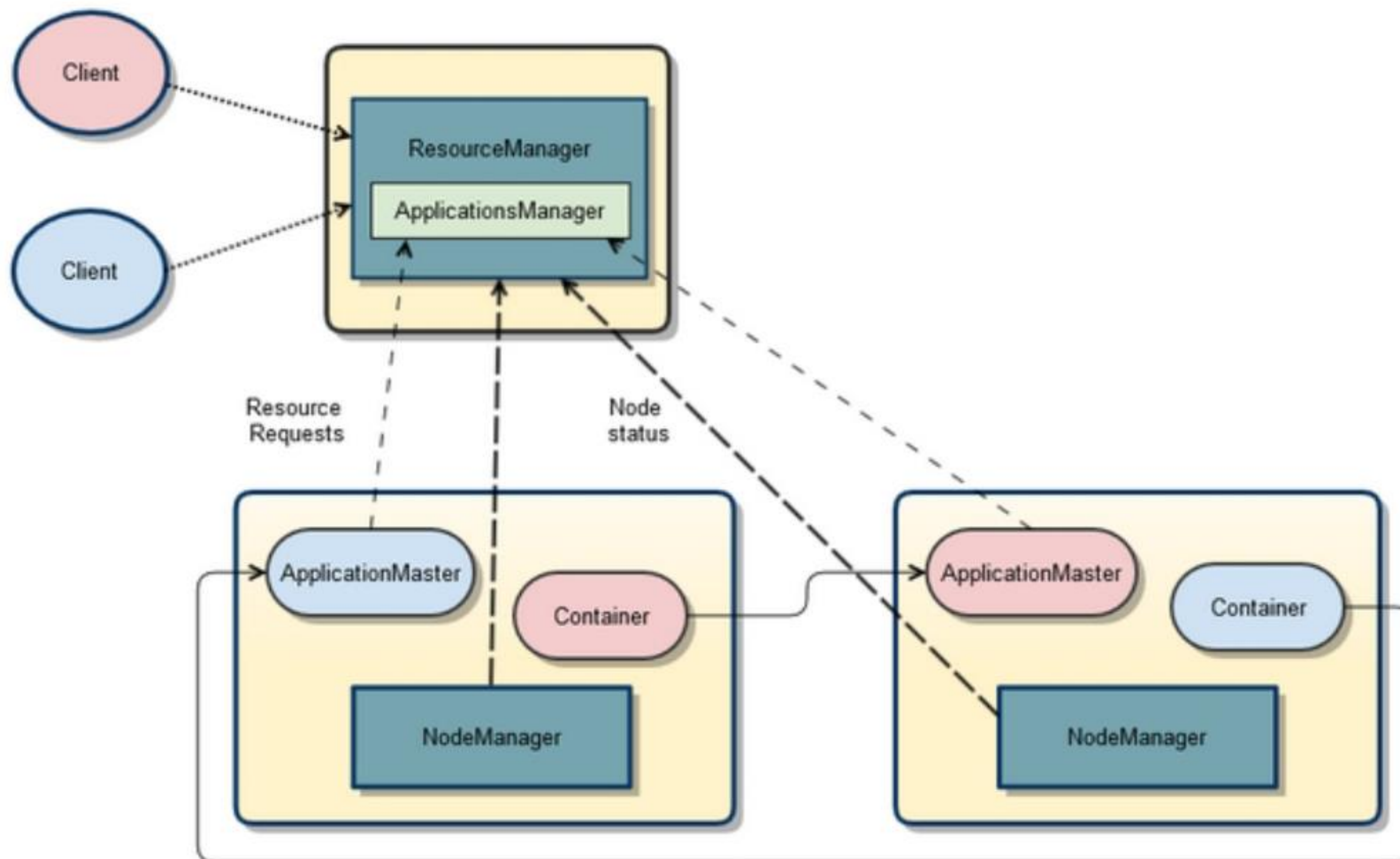
Architecture Hadoop (5): Yarn

Le nouveau moteur d'exécution introduit dans Hadoop 2 est Yarn. Ce nouveau moteur d'exécution est:

● Plus générique (il impose moins une logique map/reduce inflexible, et d'une manière plus générale l'interdépendance avec HDFS est moins forte).

● Vise à décoréler la gestion des resources (CPU, mémoire, slots d'exécution, etc.) du scheduling des applications dans deux démons séparés (là où JobTracker assurait ces deux tâches en MRv1)

Architecture Hadoop (5): Yarn



Yarn: Resource Manager

Le Resource Manager maintient une table de « resources » disponibles sur le cluster: machines, slots d'exécution, mémoire/CPU sur les machines, etc.
Il est composé de deux principaux composants:

● Le scheduler, en charge de distribuer des travaux de manière générique aux différents slots d'exécution du cluster.

● L'Applications Manager, à l'écoute de nouveaux programmes à exécuter proposés par des clients, et qui lancera une première tâche lors du début de l'exécution:
la tâche Application Master, qui constituera le « chef d'orcheste » de l'exécution du programme et soumettra par elle-même de nouvelles tâches à effectuer au scheduler du ResourceManager directement

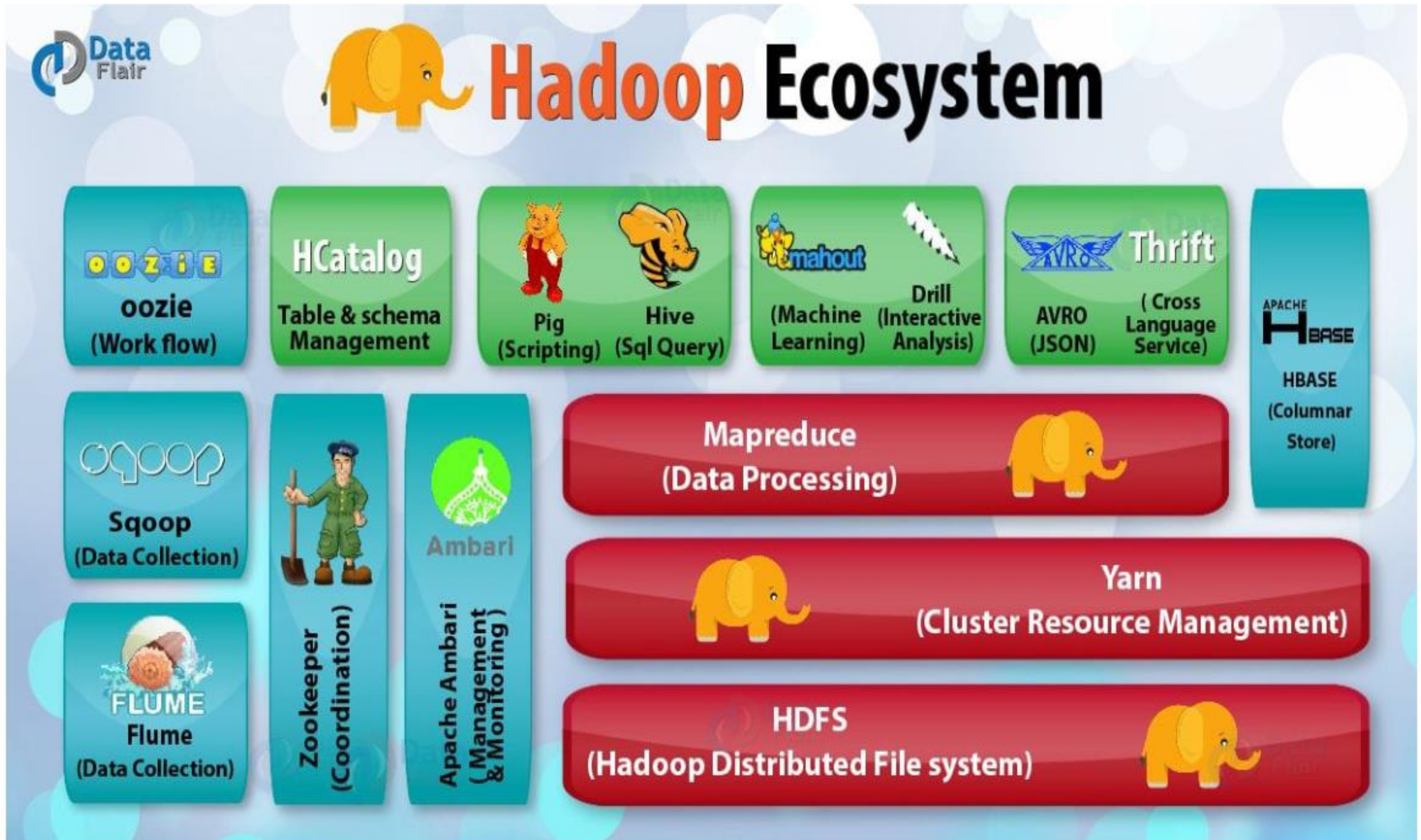
Yarn: NodeManager

● Du coté des nœuds, c'est un second démon, NodeManager, qui tourne sur chaque machine et qui est comparable au TaskTracker du modèle v1.

● Ce démon est en charge de maintenir les slots d'exécution locaux (désignés sous le terme générique de container d'exécution), et de recevoir et exécuter des tâches dans ces containers attribués aux applications par le scheduler.

● Ces tâches à exécuter peuvent être de simples opérations map ou reduce, mais aussi le coeur de l'application lui-même : la classe driver, main du programme, qui serait alors lancée au sein d'une tâche spéciale ApplicationMaster, lancée par l'ApplicationManager du ResourceManager.

Architecture plus complexe



Programmation Hadoop

Pour développer un programme Hadoop, on va créer trois classes distinctes:

- **Une classe dite « Driver » qui contient la fonction main du programme. Cette classe se chargera d'informer Hadoop des types de données clef/valeur utilisées, des classes se chargeant des opérations MAP et REDUCE, et des fichiers HDFS à utiliser pour les *entrées/sorties*.**

- **Une classe MAP (qui effectuera l'opération MAP).**

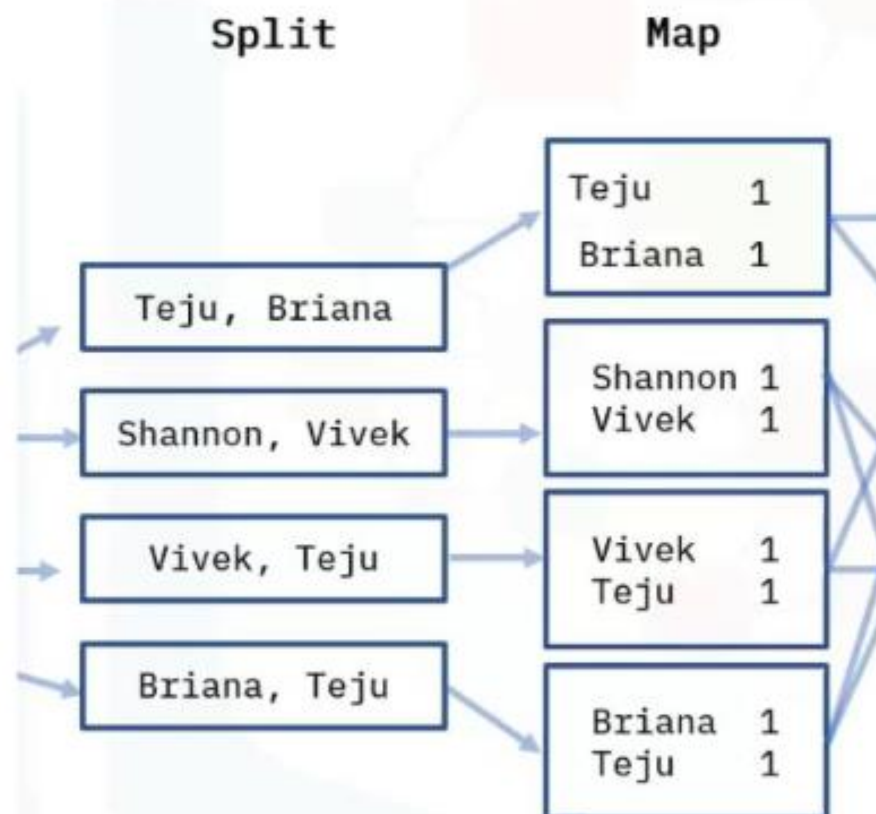
- **Une classe REDUCE (qui effectuera l'opération REDUCE).**

Programmation Hadoop
Datatype

1. **Integer** –> **IntWritable**: It is the Hadoop variant of *Integer*. It is used to pass integer numbers as key or value.
2. **Float** –> **FloatWritable**: Hadoop variant of *Float* used to pass floating point numbers as key or value.
3. **Long** –> **LongWritable**: Hadoop variant of *Long* data type to store long values.
4. **Short** –> **ShortWritable**: Hadoop variant of *Short* data type to store short values.
5. **Double** –> **DoubleWritable**: Hadoop variant of *Double* to store double values.
6. **String** –> **Text**: Hadoop variant of *String to pass string characters as key or value.*
7. **Byte** –> **ByteWritable**: Hadoop variant of *byte* to store sequence of bytes.
8. **null** –> **NullWritable**: Hadoop variant of *null* to pass null as a key or value. Usually *NullWritable* is used as data type for output key of the reducer, when the output key is not important in the final result.

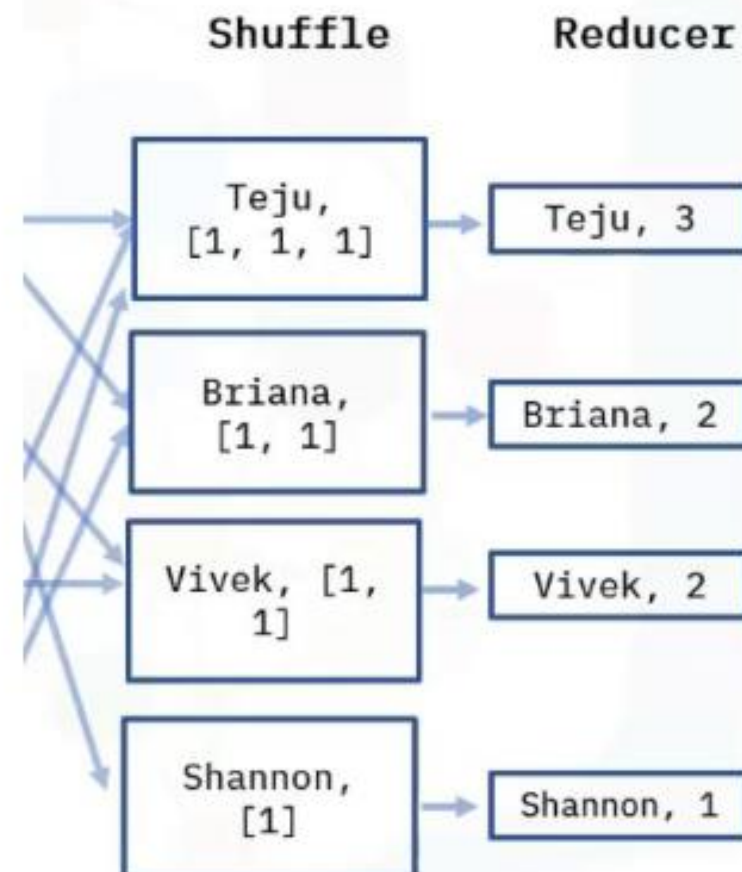
Programmation Hadoop
MAP

```java
public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
  StringTokenizer itr = new StringTokenizer(value.toString());
  while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
  }
 }
}
```



Programmation Hadoop
Reduce

```
public void reduce(Text key, Iterable<IntWritable> values,
                   Context context
                   ) throws IOException, InterruptedException {
  int sum = 0;
  for (IntWritable val : values) {
    sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
 }
}
```



Conclusion

- Base intéressante pour pouvoir gérer de gros volumes de données
- Combinaison de HDFS et de MapReduce
- Implémenté dans de nombreux outils
- Puissant mais difficile à implémenter