

Big Data

Chapitre 2: Hadoop

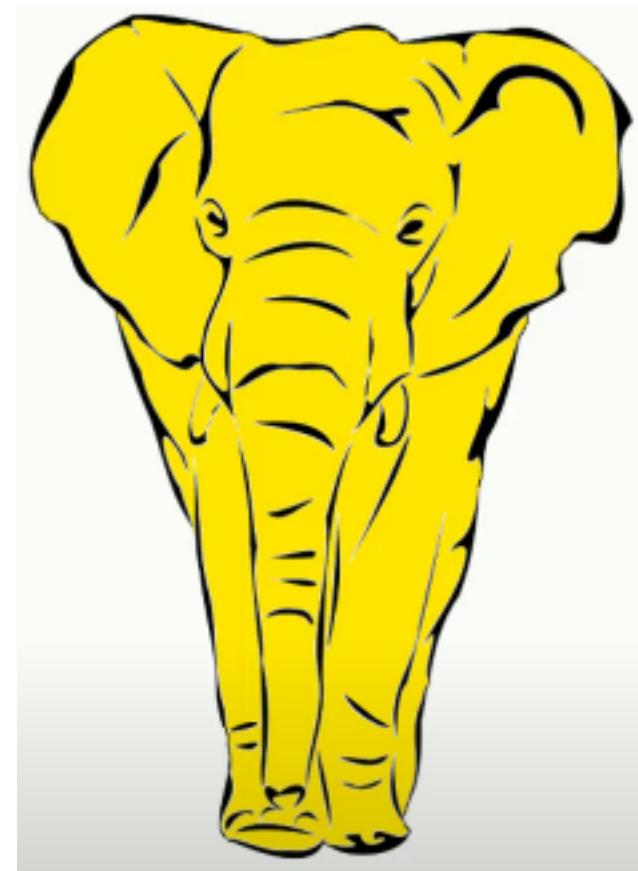
Présentation du framework

HDFS

Map Reduce

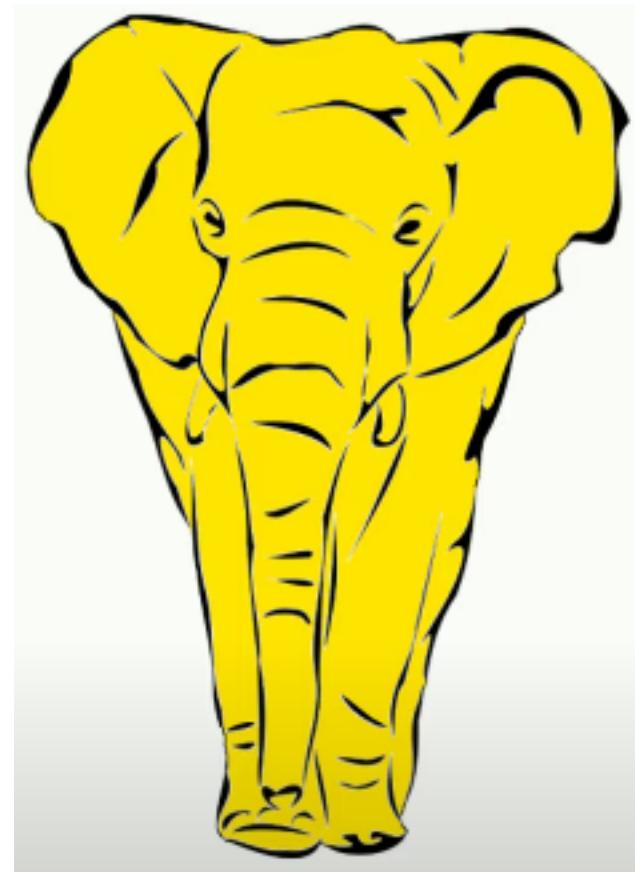
Map Reduce

- C'est la partie la plus compliquée de Hadoop
- Deux parties à voir:
 - Le framework Map Reduce
 - La technique de programmation Map Reduce



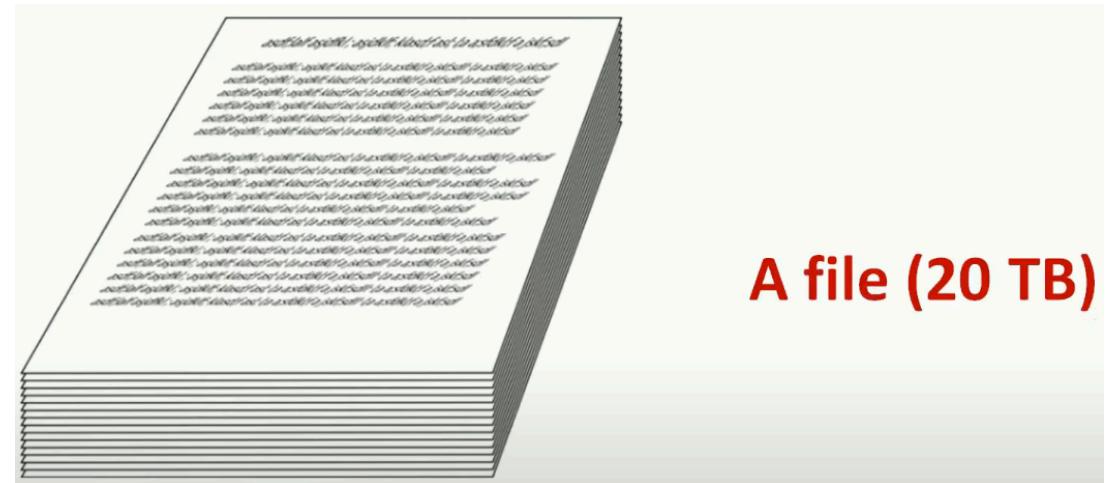
Map Reduce

- C'est la partie la plus compliquée de Hadoop
- Deux parties à voir:
 - **Le framework Map Reduce**
 - Fournit des API Java; classes et interfaces pour la création de programmes Map Reduce
 - Gère des tâches internes au moment de l'exécution de programmes
 - **Map Reduce execution engine**
 - **La technique de programmation Map Reduce**
 - Méthode de raisonnement pour la résolution de problèmes Big Data
 - Ce n'est pas une approche universelle pour la résolution de tous les types de problèmes Big Data



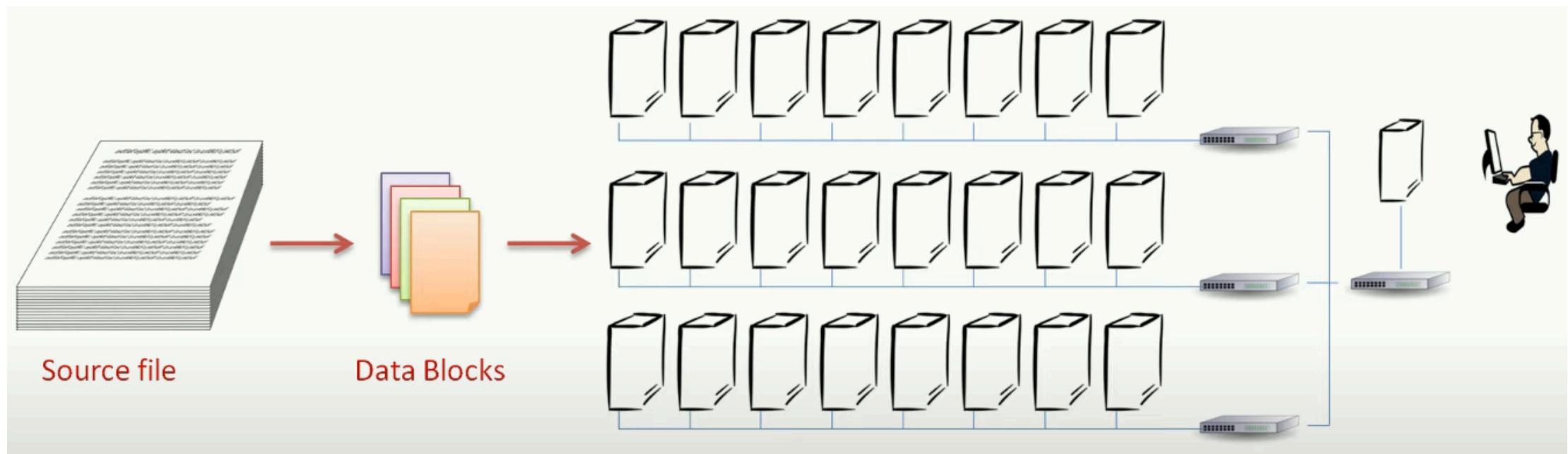
Map Reduce

- Commençons par un exemple de problème simple
- Nous disposons d'un fichier de 20 TB
- Au lieu de le stocker sur une seule machine, on va le stocker sur un cluster Hadoop



A file (20 TB)

Map Reduce



Map Reduce

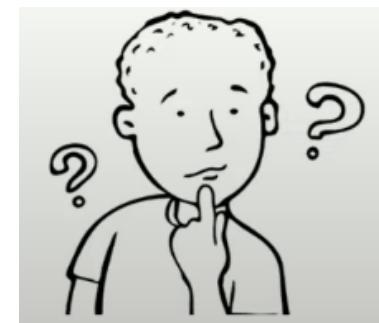


Map Reduce

On souhaite compter le nombre de lignes dans ce fichier!

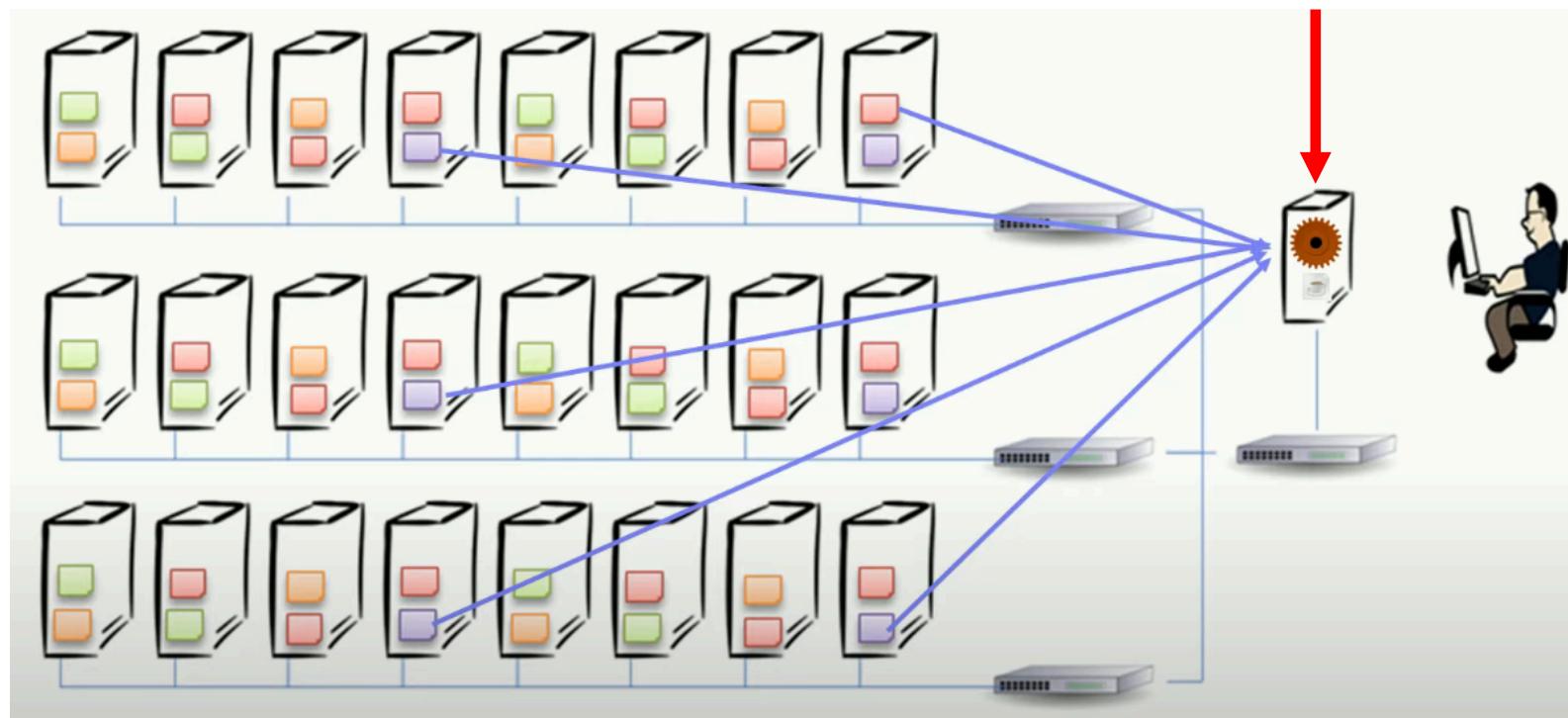


- Lire le fichier ligne par ligne
- Incrémenter un compteur
- Si fin de fichier envoyer le résultat



Map Reduce

Lire le fichier en entier
à partir du cluster
HDFS puis compter le
nombre de lignes

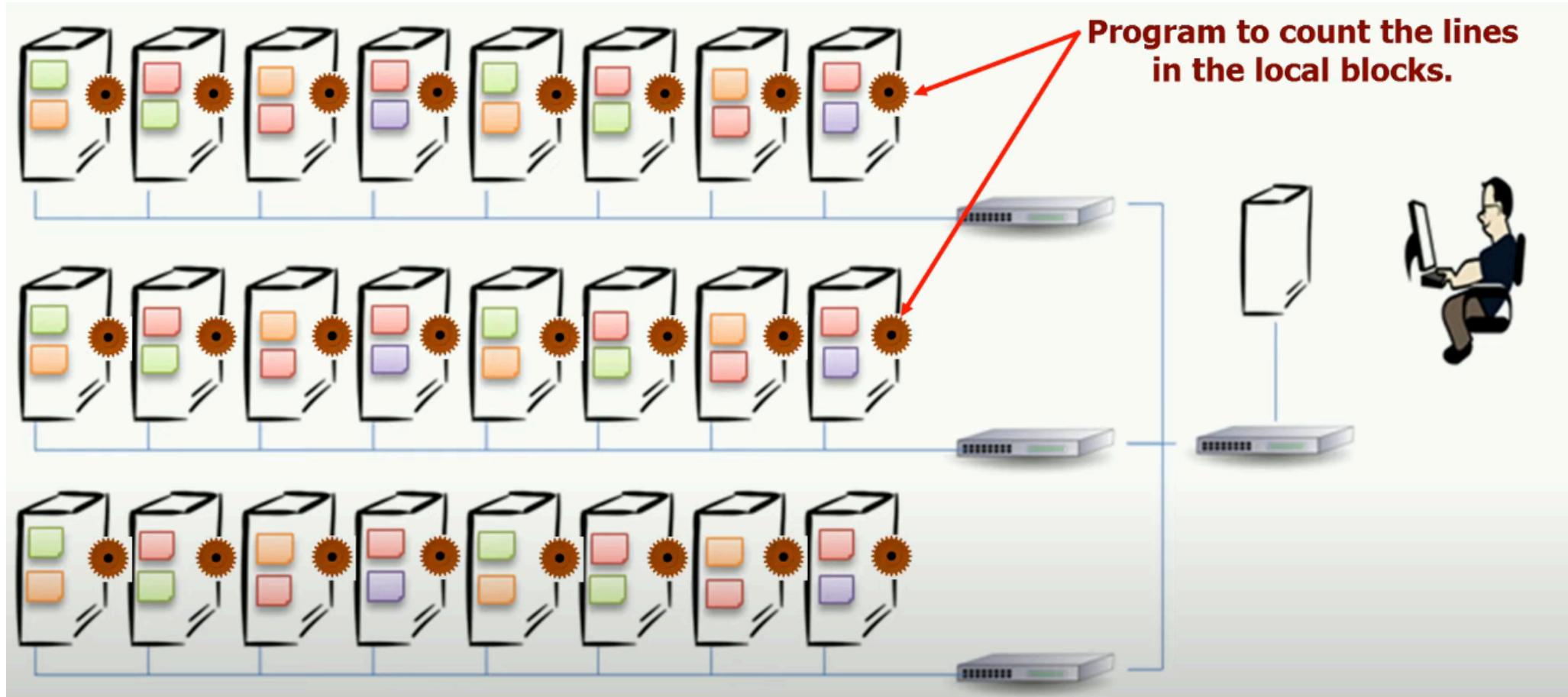


Programme Java
pour compter le
nombre de lignes

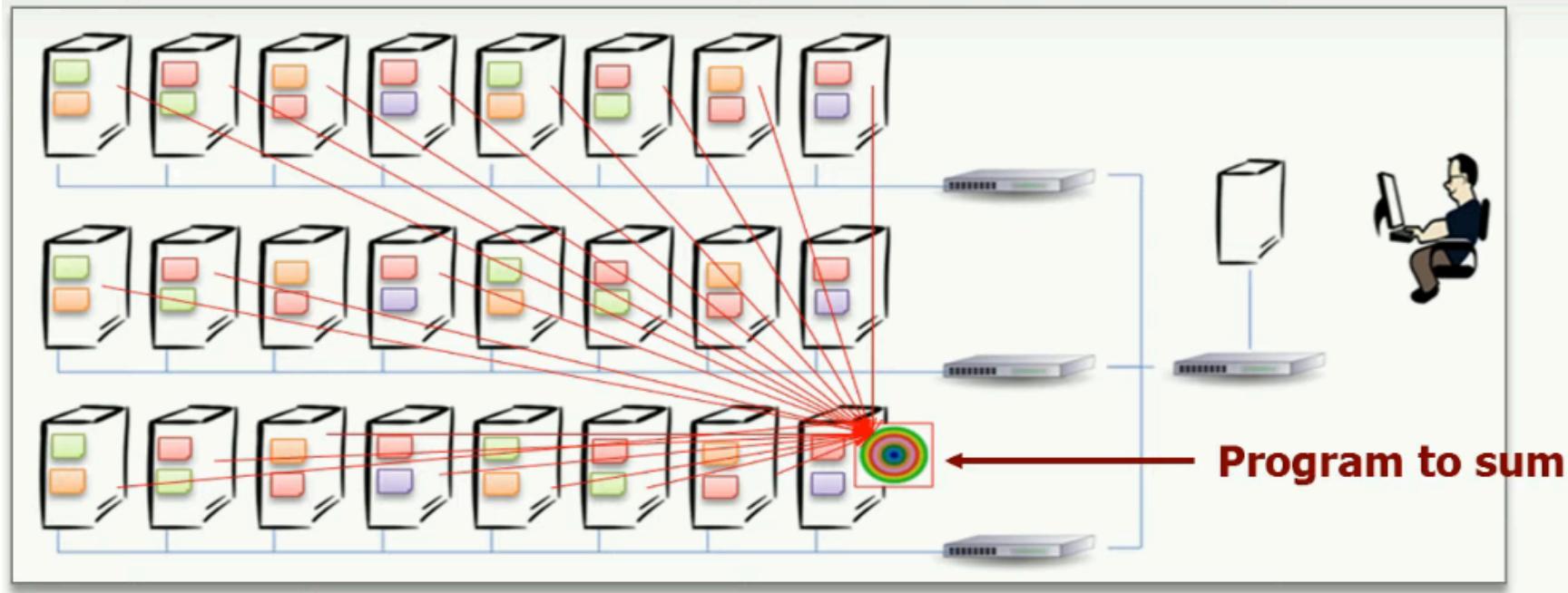
Map Reduce

- Le programme est exécuté sur une machine locale
- Tous les blocks (20 TB) sont copiés dans la machine locale
 - Le déplacement des données est très couteux en terme de temps
- Ceci peut prendre des heures, voire des jours pour finir ce calcul
- La solution est de ne pas déplacer les données
- Faire le traitement sur chaque block en local
- Exploiter les nœuds (machines) pour faire un calcul parallèle
 - Réduire le temps de calcul

Map Reduce

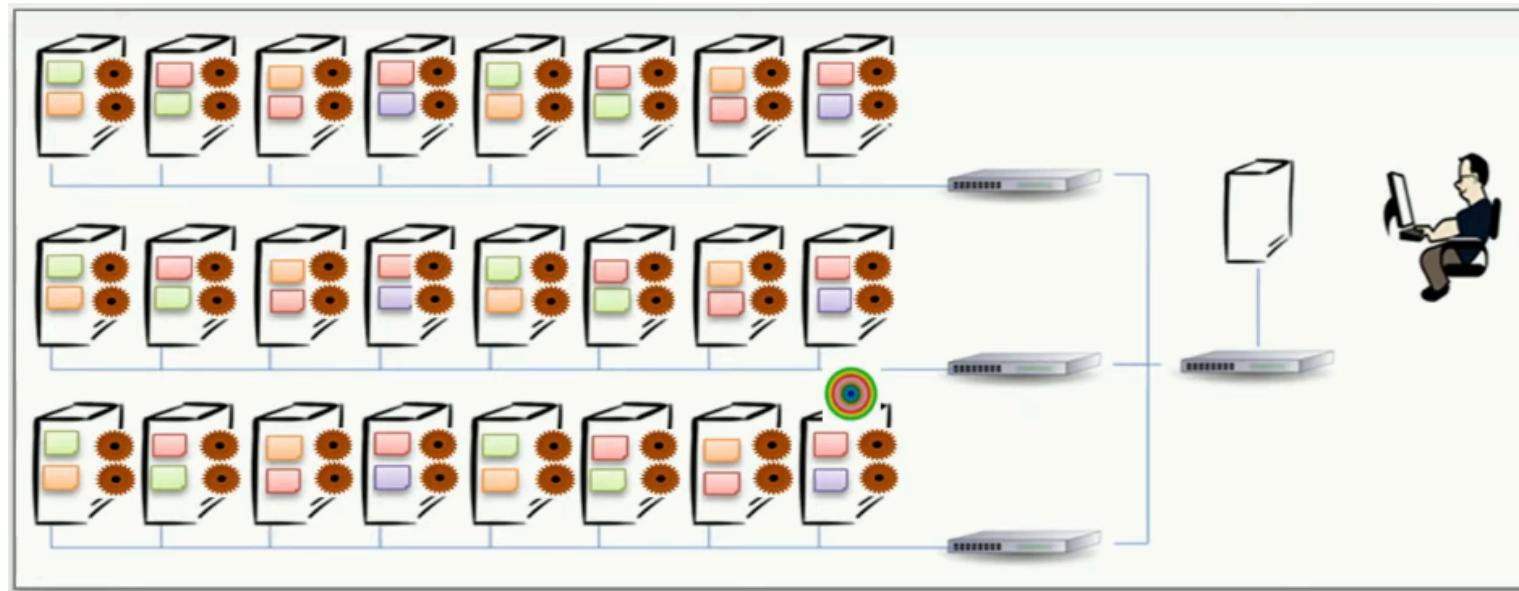


Map Reduce



- Collecter tous les résultats sur une seule machine
- Cette tâche n'est pas couteuse en terme de temps d'execution
 - Quelques chiffres à copier
- Faire la somme pour avoir le résultat final

Map Reduce



- La fonction Map est exécutée sur chaque machine, en particulier **sur chaque block de donnée**.
- La fonction Reduce est exécutée sur une seule machine

- **C'est quoi Map reduce?**

 **Map Function**
 **Reduce Function**

Map Reduce

- Comment la fonction Map va lire les lignes dans chaque block?
- Qui va collecter les résultats de chaque fonction Map dans chaque machine et les envoyer ensuite à la machine exécutant la fonction Reduce?



C'est le moteur d'exécution (execution engine) qui se charge de tout ça

Map Reduce

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class LineCount {

    public static class LineCountMapper
        extends Mapper<Object, Text, Text, IntWritable> {

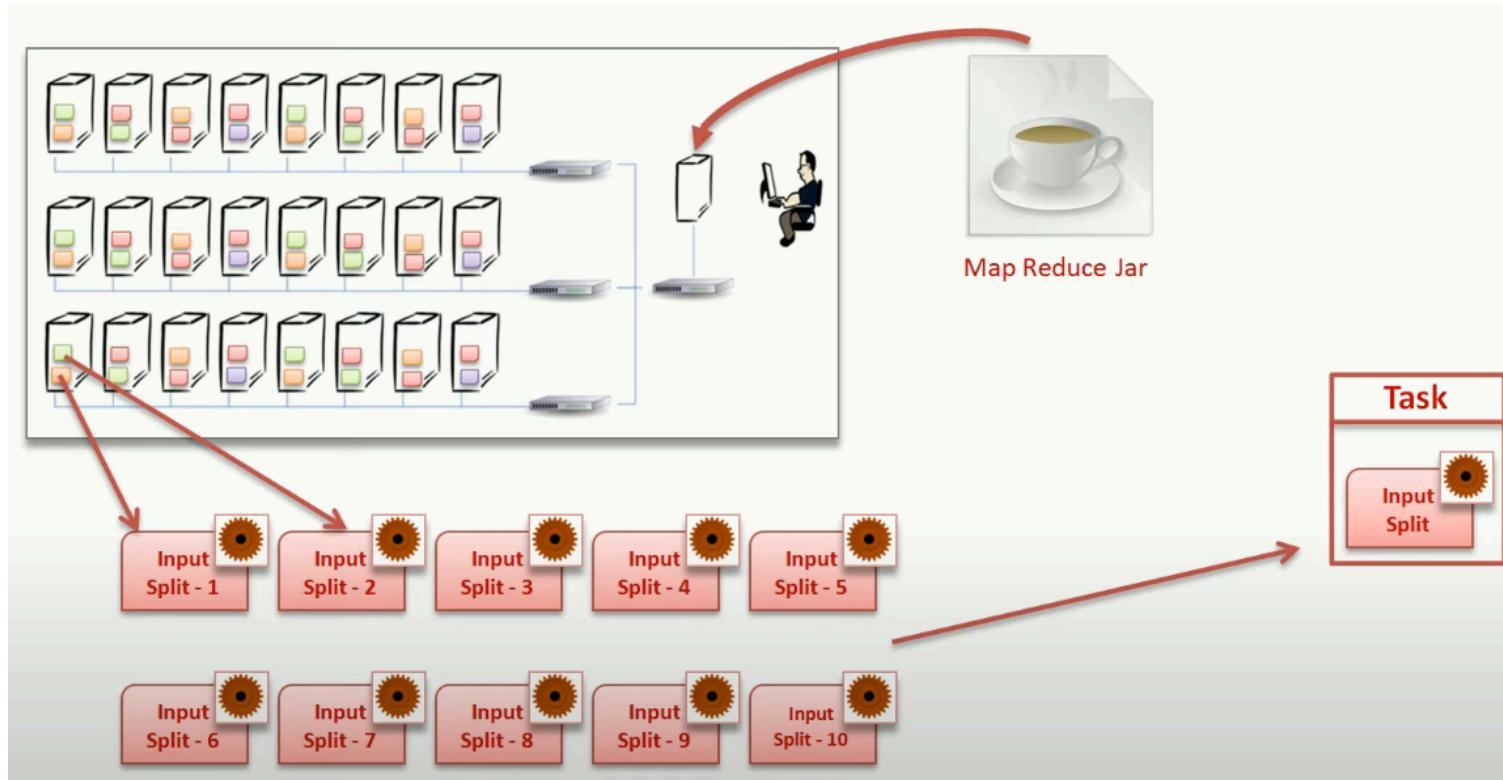
        public void map(Object key, Text value, Context context
                        ) throws IOException, InterruptedException {
            String line = value.toString();
            if (line != null && !line.isEmpty())
                context.write(new Text("No of Lines"), new IntWritable(1));
        }
    }

    public static class LineCountReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        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);
        }
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "line count");
    job.setJarByClass(LineCount.class);
    job.setMapperClass(LineCountMapper.class);
    job.setCombinerClass(LineCountReducer.class);
    job.setReducerClass(LineCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

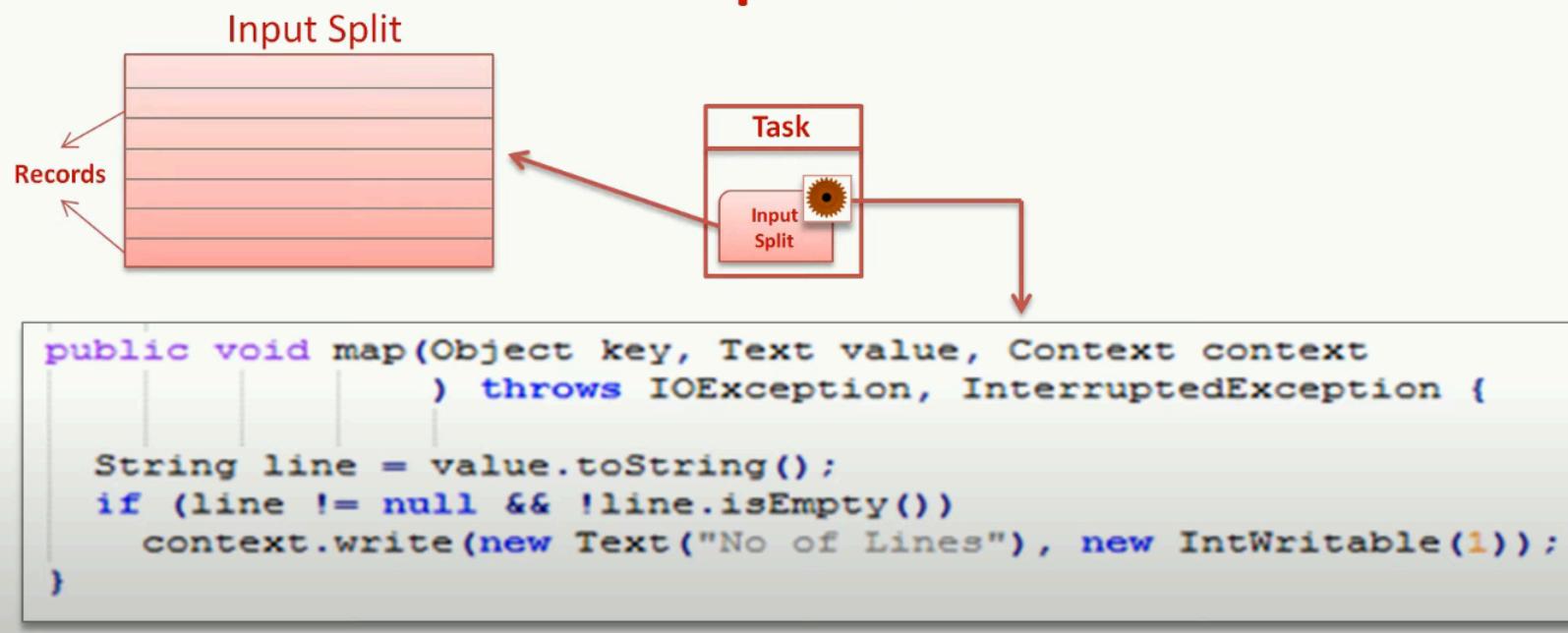
Map Reduce



- Le fichier jar est déployé au niveau du framework
- Le moteur d'exécution crée des input split un pour chaque bloc
- Exécuter une instance de la fonction map dans chaque input split (un processus pour chaque bloc)

Map Reduce

The Map Function

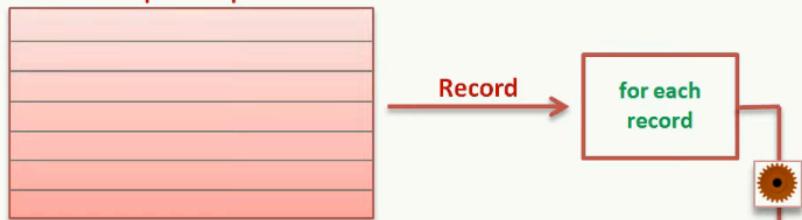


- Le framework devra exécuter la fonction map en lui passant les données à partir du input split
- Au lieu de fournir tout le bloc à la fonction map, le moteur d'exécution divise le bloc en records (enregistrements)
- Dans cet exemple, un record est une ligne de texte

Map Reduce

The Map Function

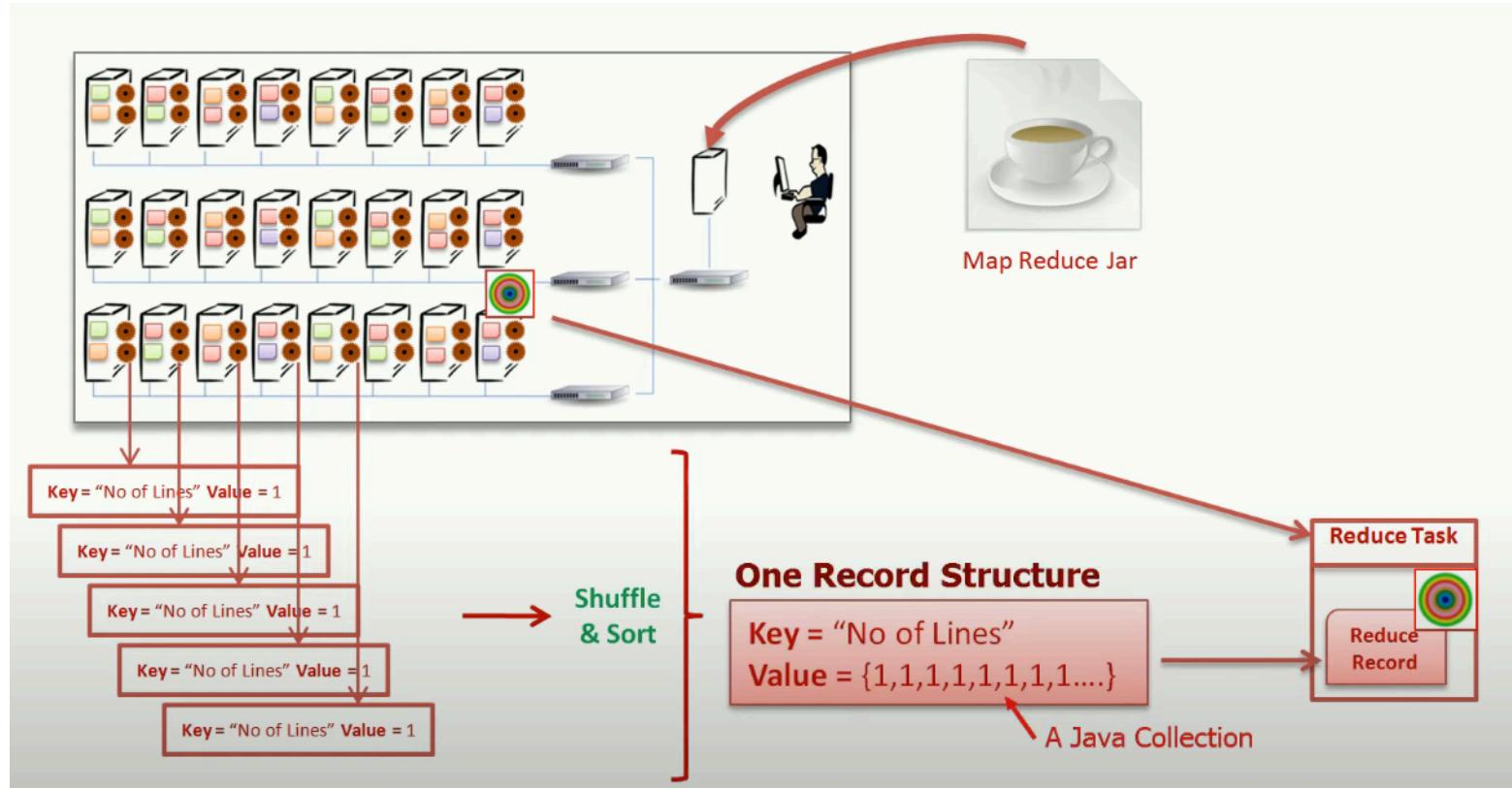
Input Split



```
public void map(Object key, Text value, Context context  
    ) throws IOException, InterruptedException {  
  
    String line = value.toString();           Key  
    if (line != null && !line.isEmpty())  
        context.write(new Text("No of Lines"), new IntWritable(1));  
}
```

- Le framework met la fonction map dans une boucle
- Dans chaque itération, la fonction map est appliquée sur chaque record (ligne)

Map Reduce



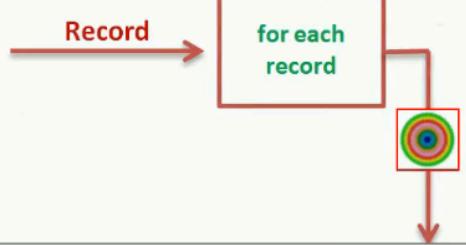
- Par défaut, un seul processus de la fonction reduce est créé
- Ici, un seul suffira pour calculer la somme de nombre de lignes
- Le framework collecte tous les résultats de chaque mapper
- Un tri est effectué par clé
- Chaque clé lui correspond un ensemble de valeurs

Map Reduce

The Reduce Function

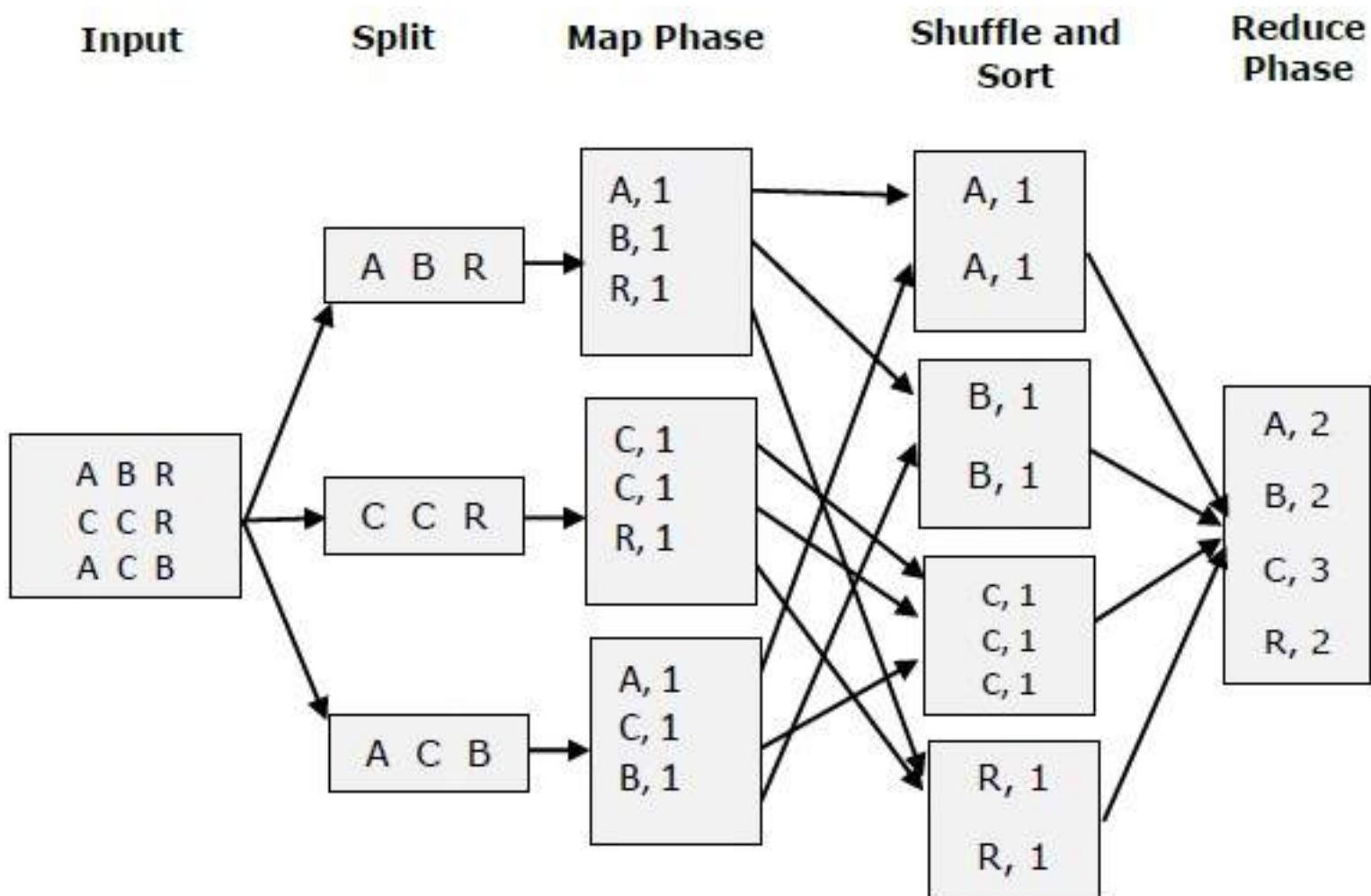
Reducer Records

Key = "No of Lines"
Value = {1,1,1,1,1,1,1,1....}



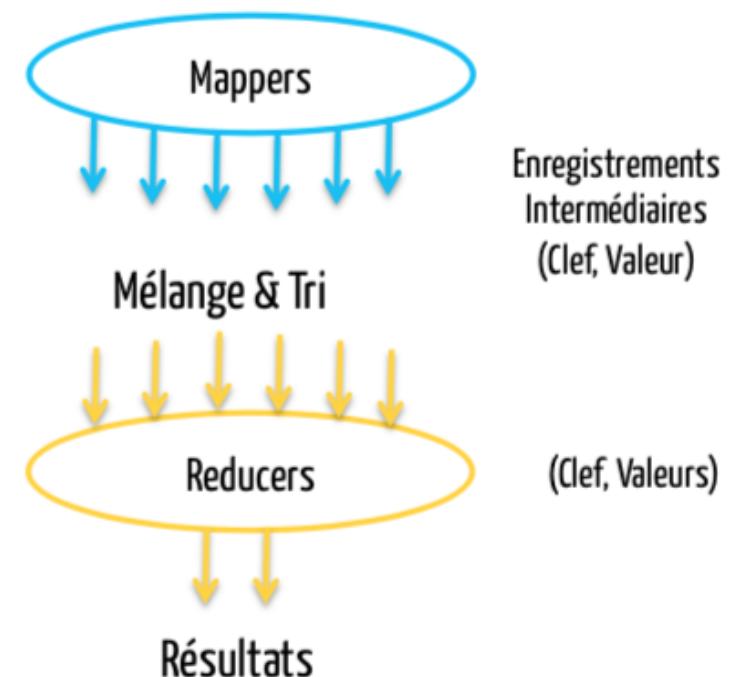
```
public void reduce(Text key, Iterable<IntWritable> values,
                  Context context
) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
}
```

- Le framework exécute la fonction reduce dans une boucle
- Une itération pour chaque clef/valeurs
- Ici une seule itération
- Le résultat est renvoyé au framework sous forme de clef/valeur
- Puis enregistré dans un fichier HDFS



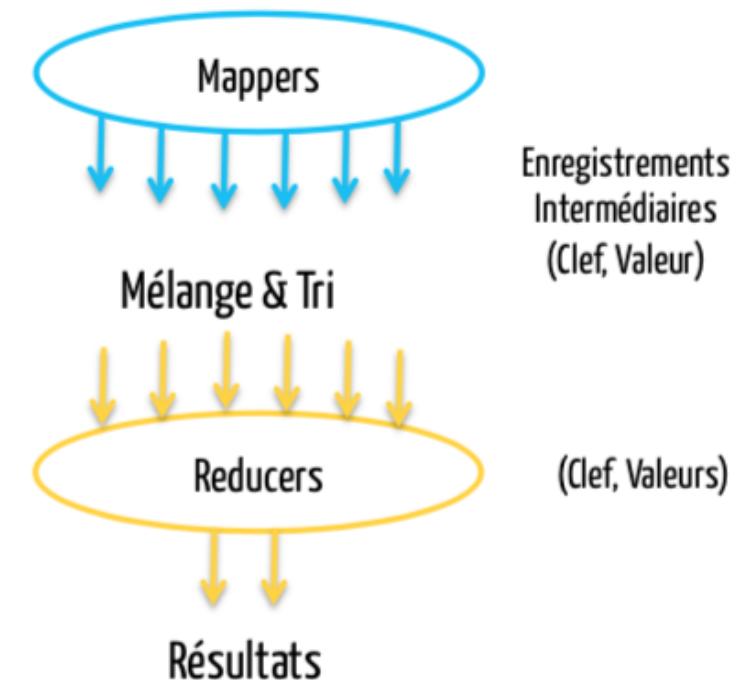
Map Reduce: fonctionnement

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



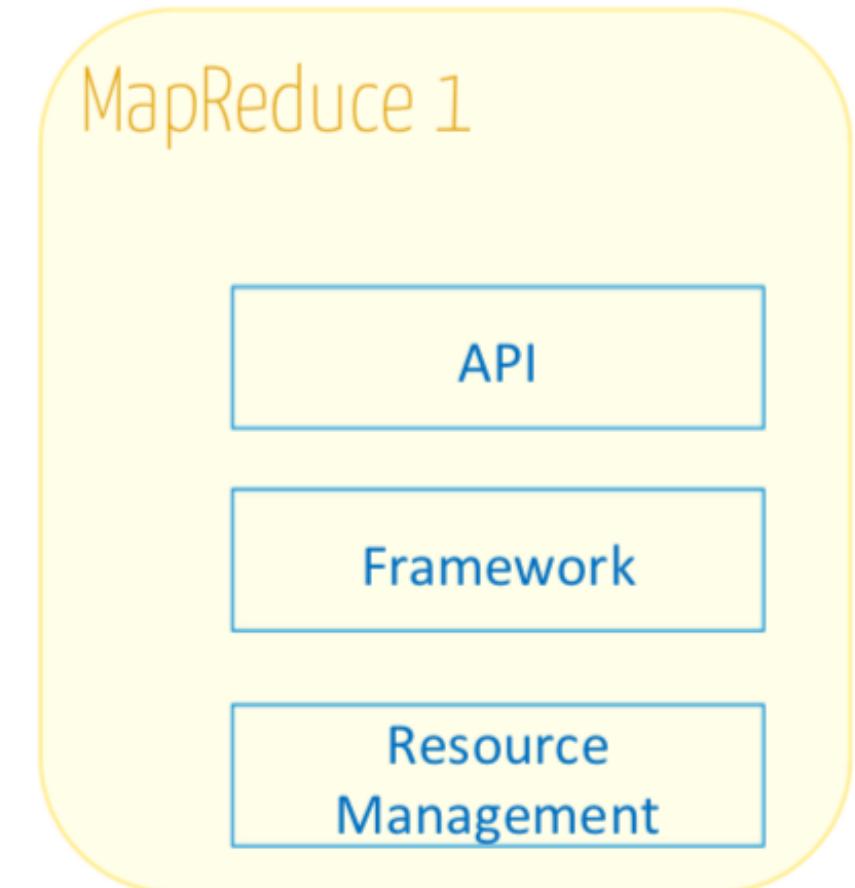
Map Reduce: fonctionnement

- 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: composants

- MapReduce v1 intègre trois composants
 - API
 - Pour permettre au programmeur l'écriture d'applications MapReduce
 - Framework
 - Services permettant l'exécution des Jobs MapReduce, le Shuffle/Sort...
 - Resource Management
 - Infrastructure pour gérer les noeuds du cluster, allouer des ressources et ordonner les jobs



Map Reduce v1: démons

- **JobTracker**
 - Divise le travail sur les Mappers et Reducers, s'exécutant sur les différents nœuds
- **TaskTracker**
 - S'exécute sur chacun des nœuds pour exécuter les vraies tâches de Map-Reduce
 - Choisit 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: fonctionnement

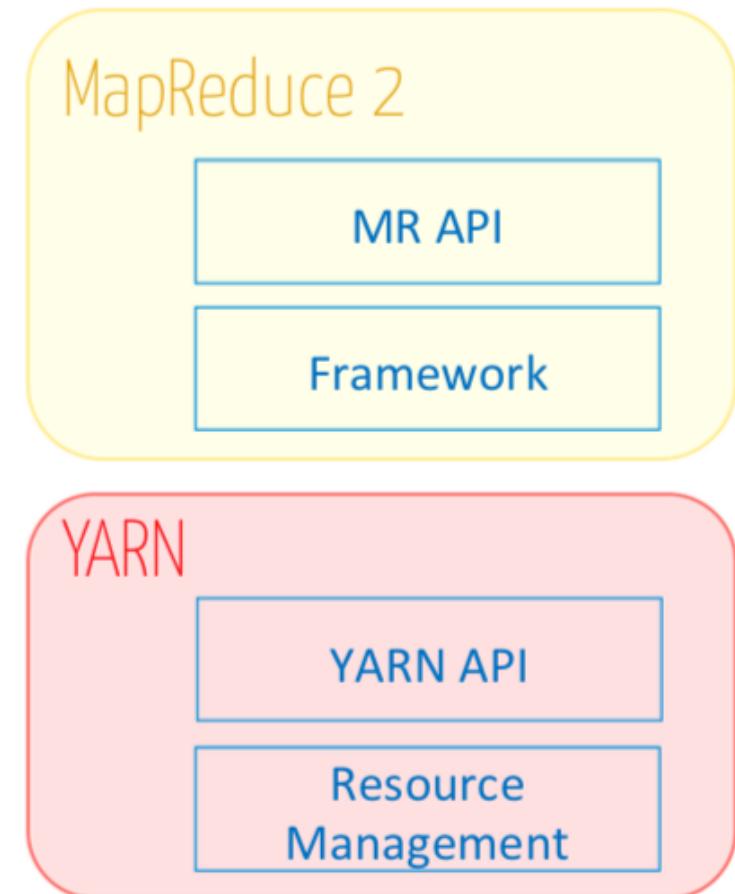
- Un job Map-Reduce (ou une Application Map-Reduce) est divisé sur plusieurs tâches appelées mappers et reducers
- Chaque tâche est exécutée sur un nœud du cluster
- Chaque nœud a un certain nombre de slots prédéfinis:
 - Map Slots
 - Reduce Slots
- Un slot est une unité d'exécution qui représente la 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 différentes tâches
 - De coordonner l'exécution des jobs Map-Reduce
 - De réserver et ordonner les slots, et de gérer les fautes en réallouant les slots au besoin

Map Reduce v1: 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 noeuds 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'interopérabilité: impossible d'exécuter des applications non-MapReduce sur HDFS

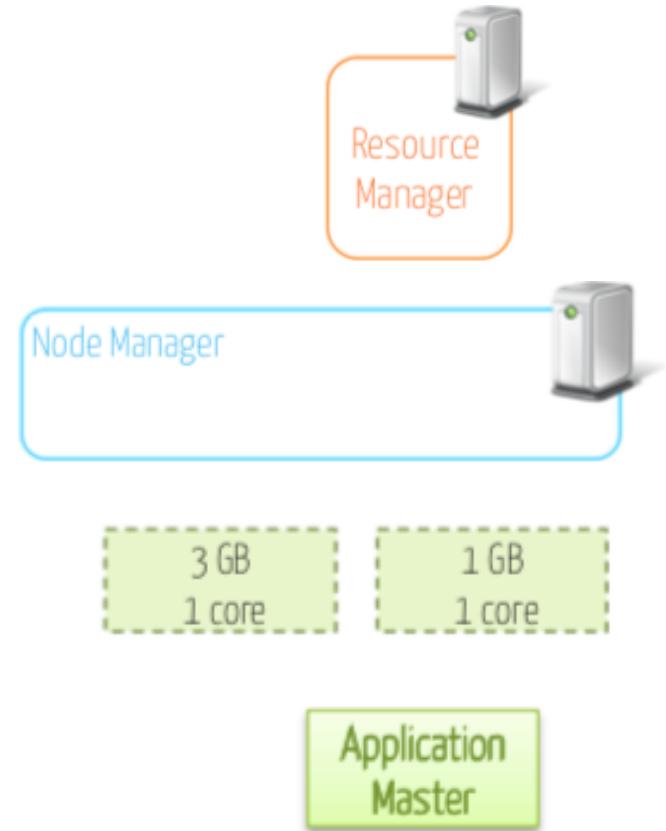
Map Reduce v2: composants

- 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
- Supporte les applications MR et non-MR

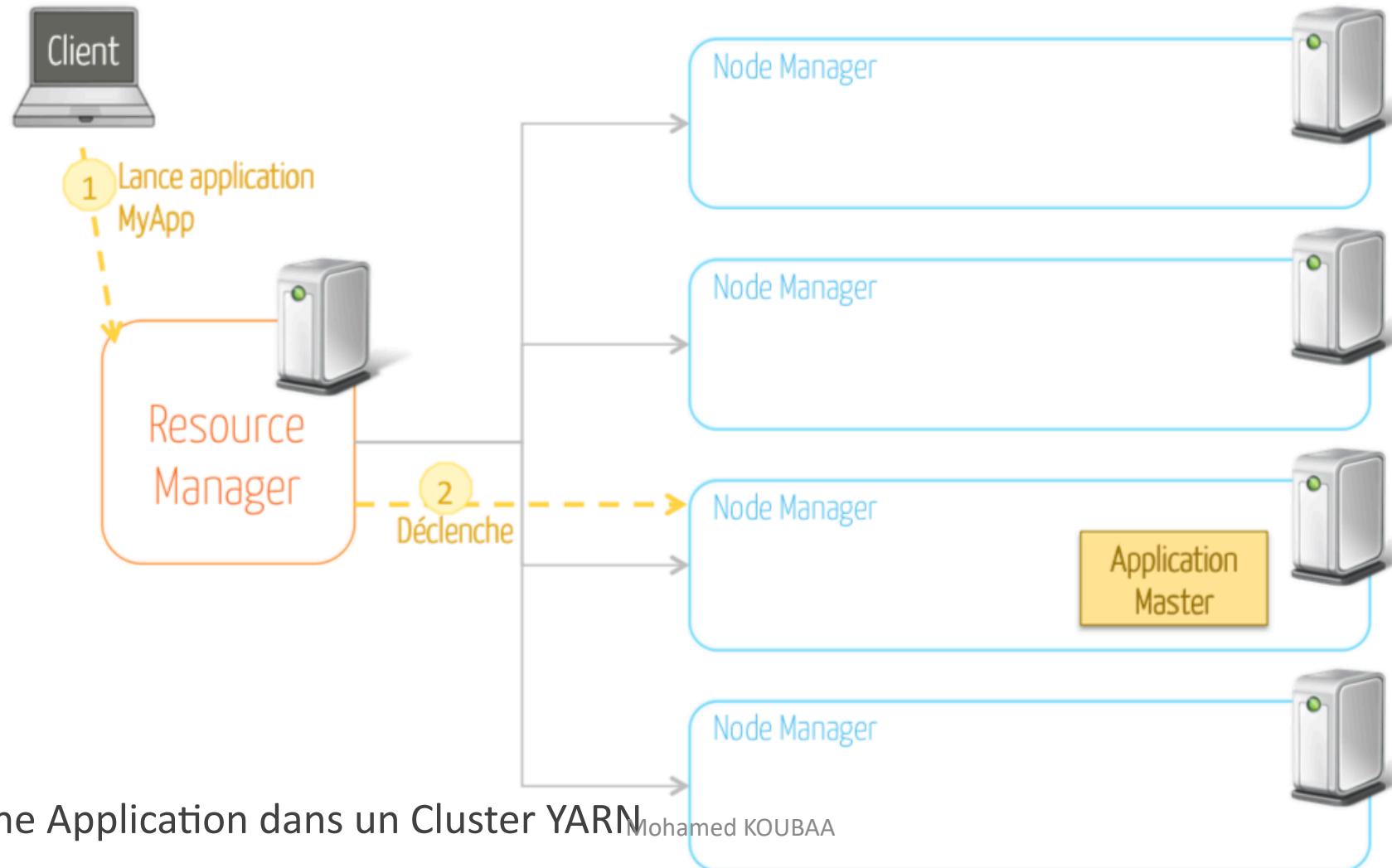


Map Reduce v2: démons

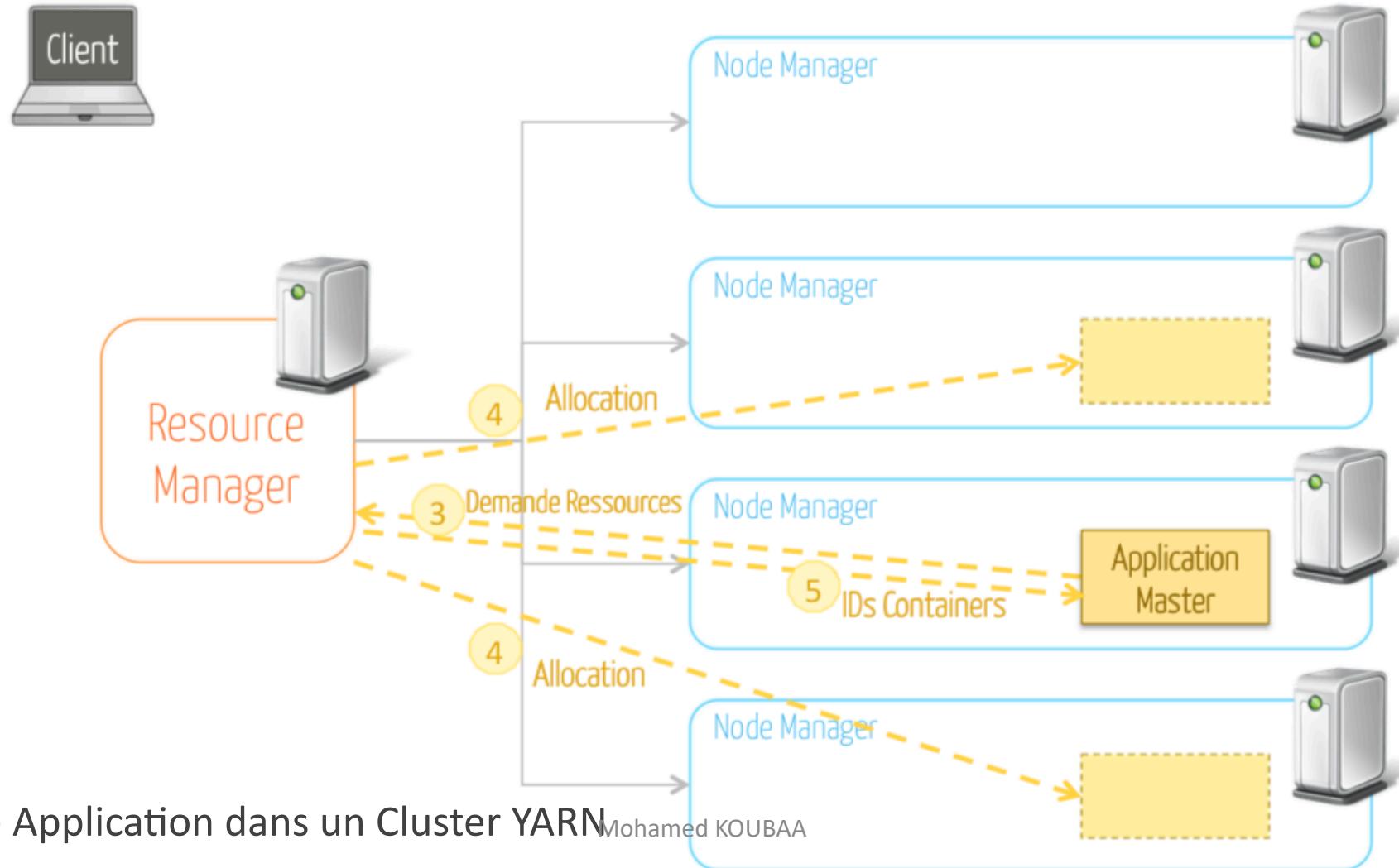
- **Resource Manager (RM)**
 - Tourne sur le nœud master
 - Ordonnanceur de ressources global
 - Permet l'arbitrage des ressources entre plusieurs applications
- **Node Manager (NM)**
 - S'exécute sur les nœuds esclaves
 - Communique avec RM
- **Containers**
 - Créés par RM à la demande
 - Se voit allouer des ressources sur le nœud esclave
- **Application Master (AM)**
 - Un seul par application
 - S'exécute sur un container
 - Demande plusieurs containers pour exécuter les tâches de l'application



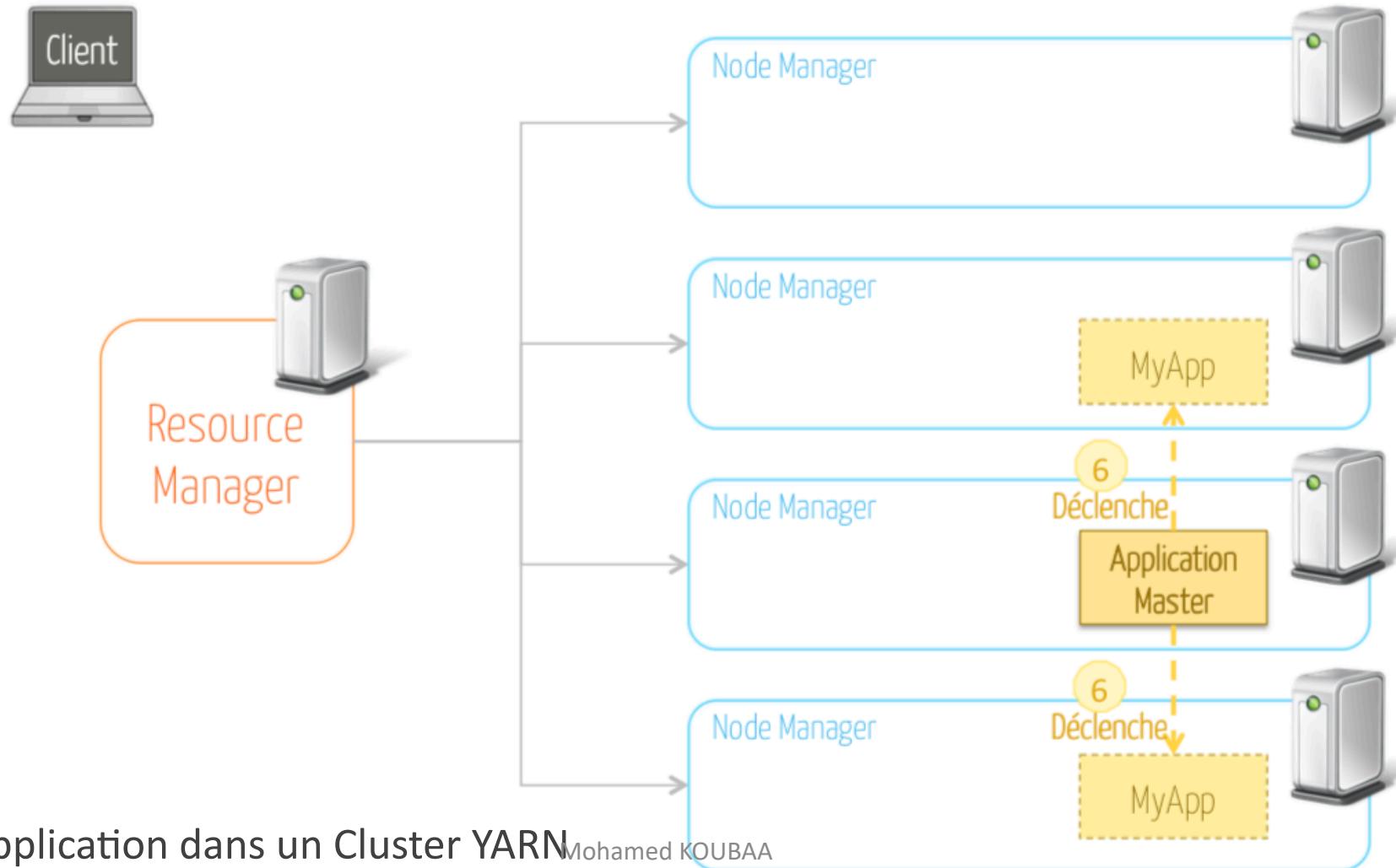
Map Reduce v2



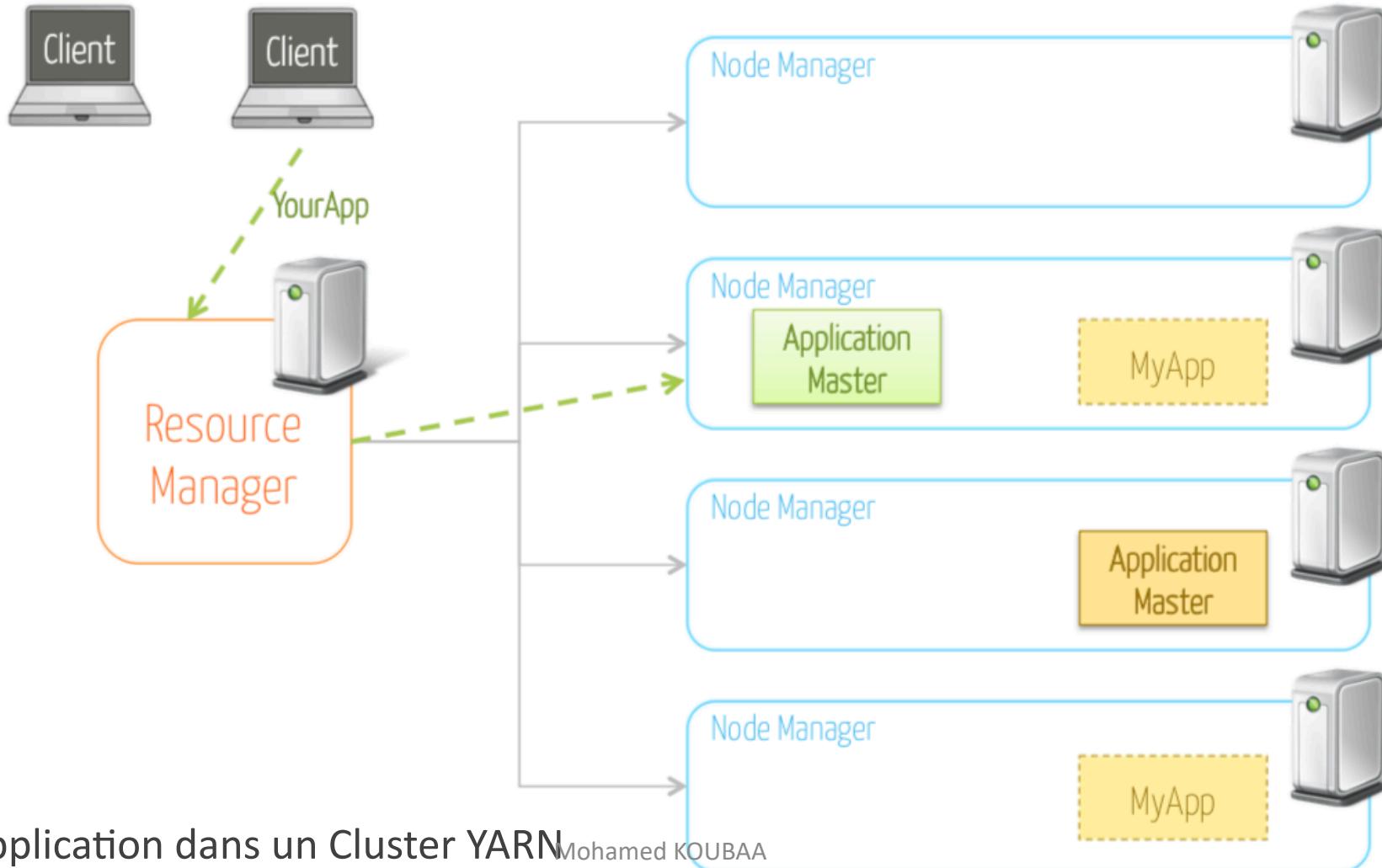
Map Reduce v2



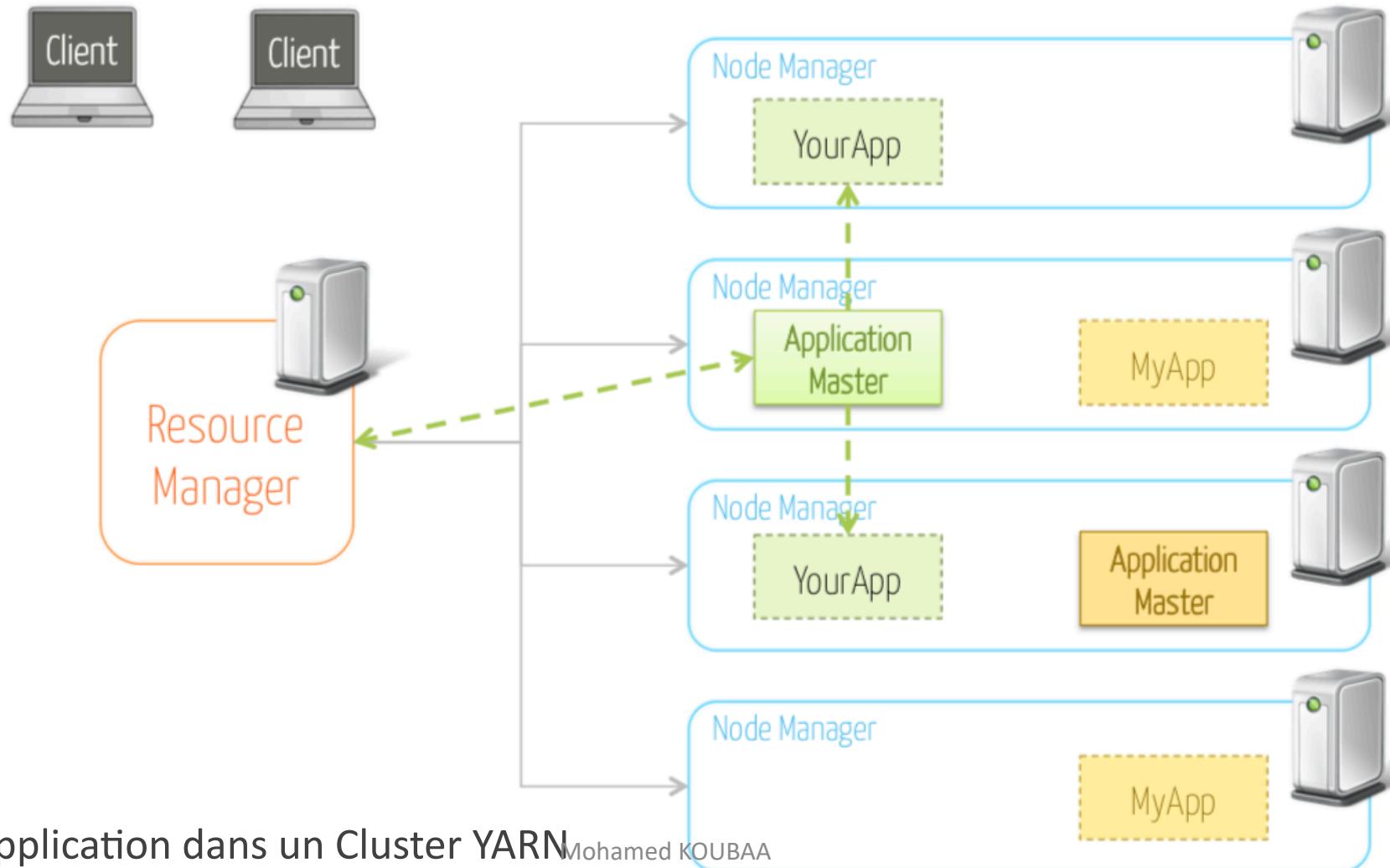
Map Reduce v2



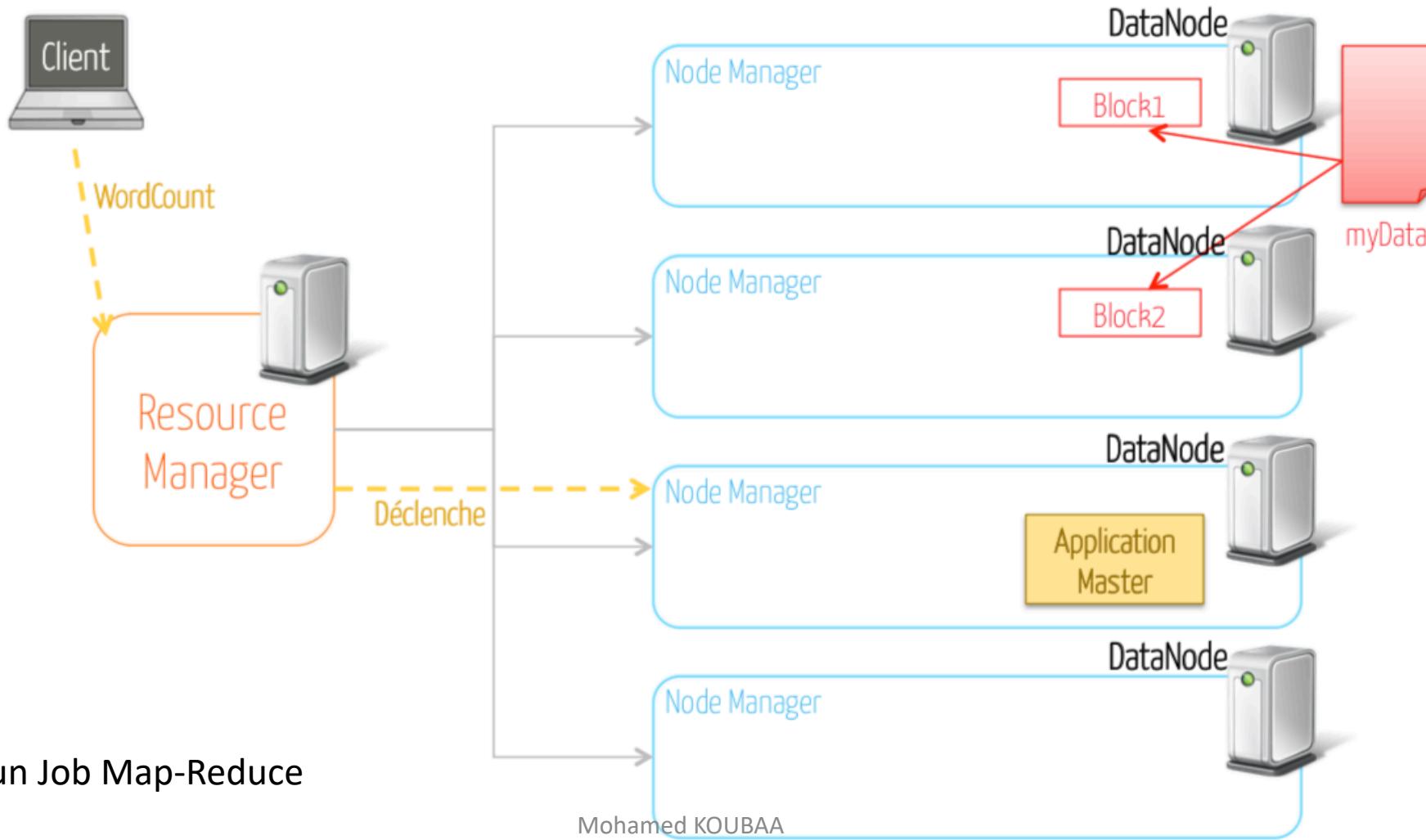
Map Reduce v2



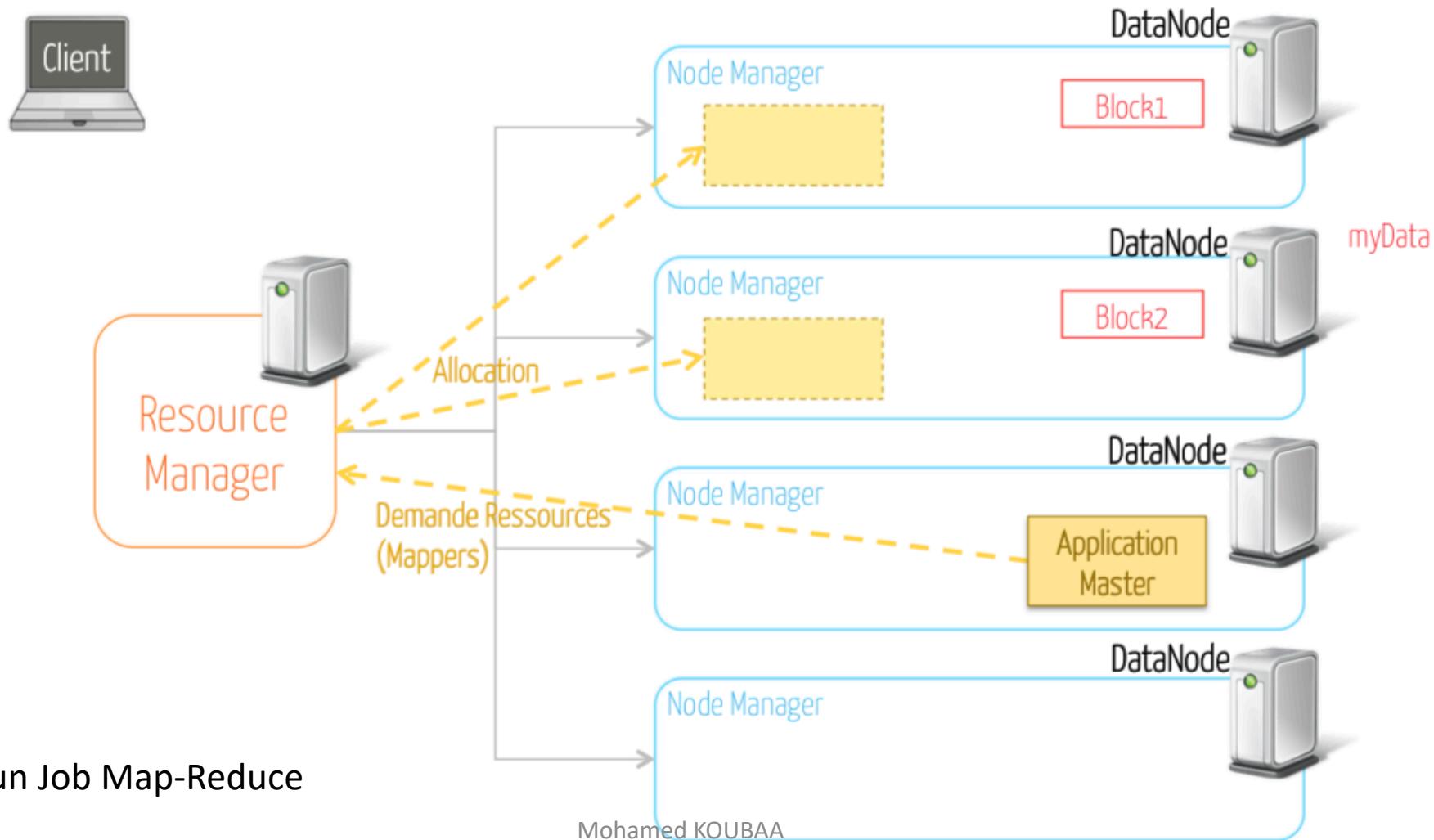
Map Reduce v2



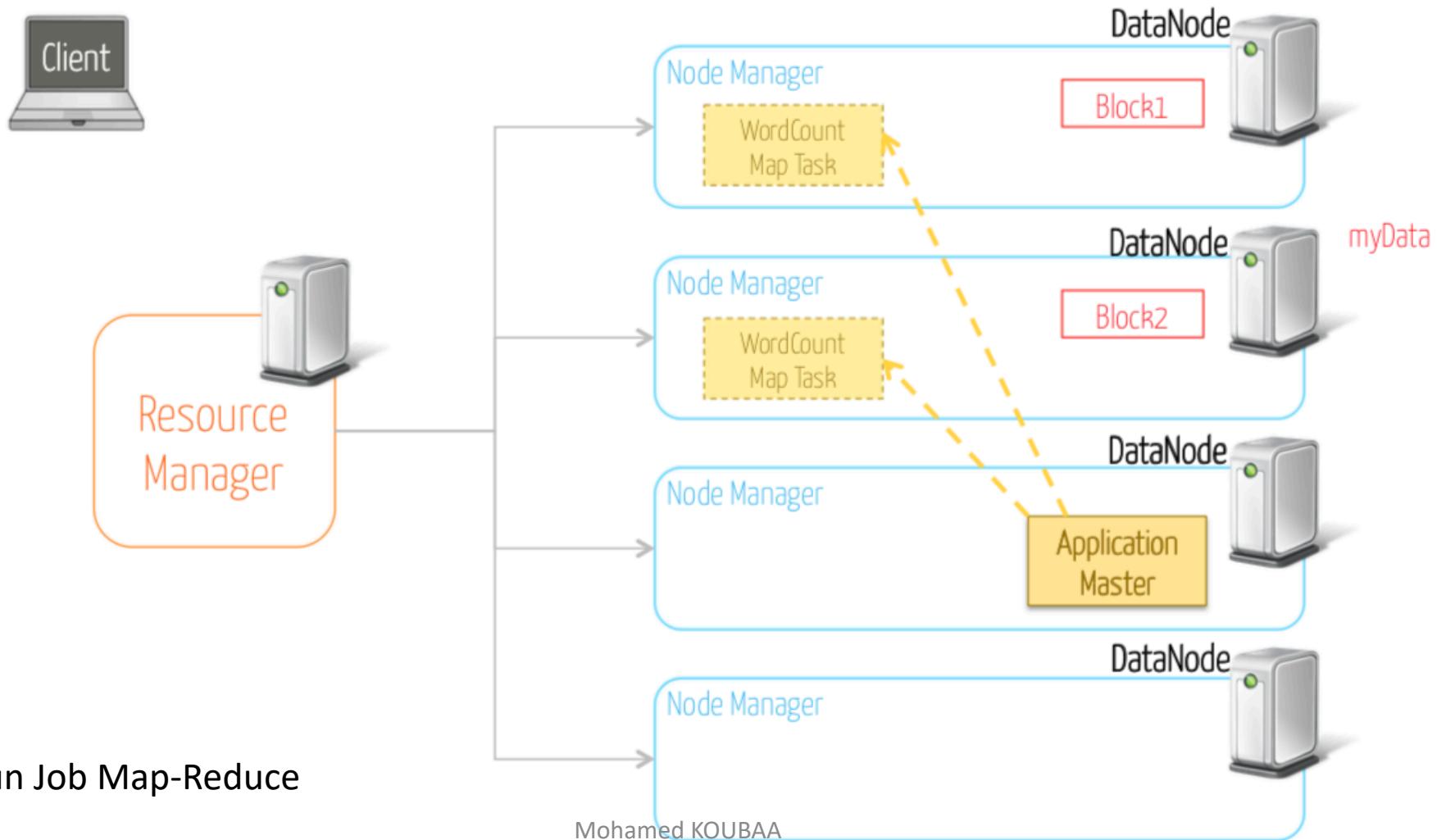
Map Reduce v2



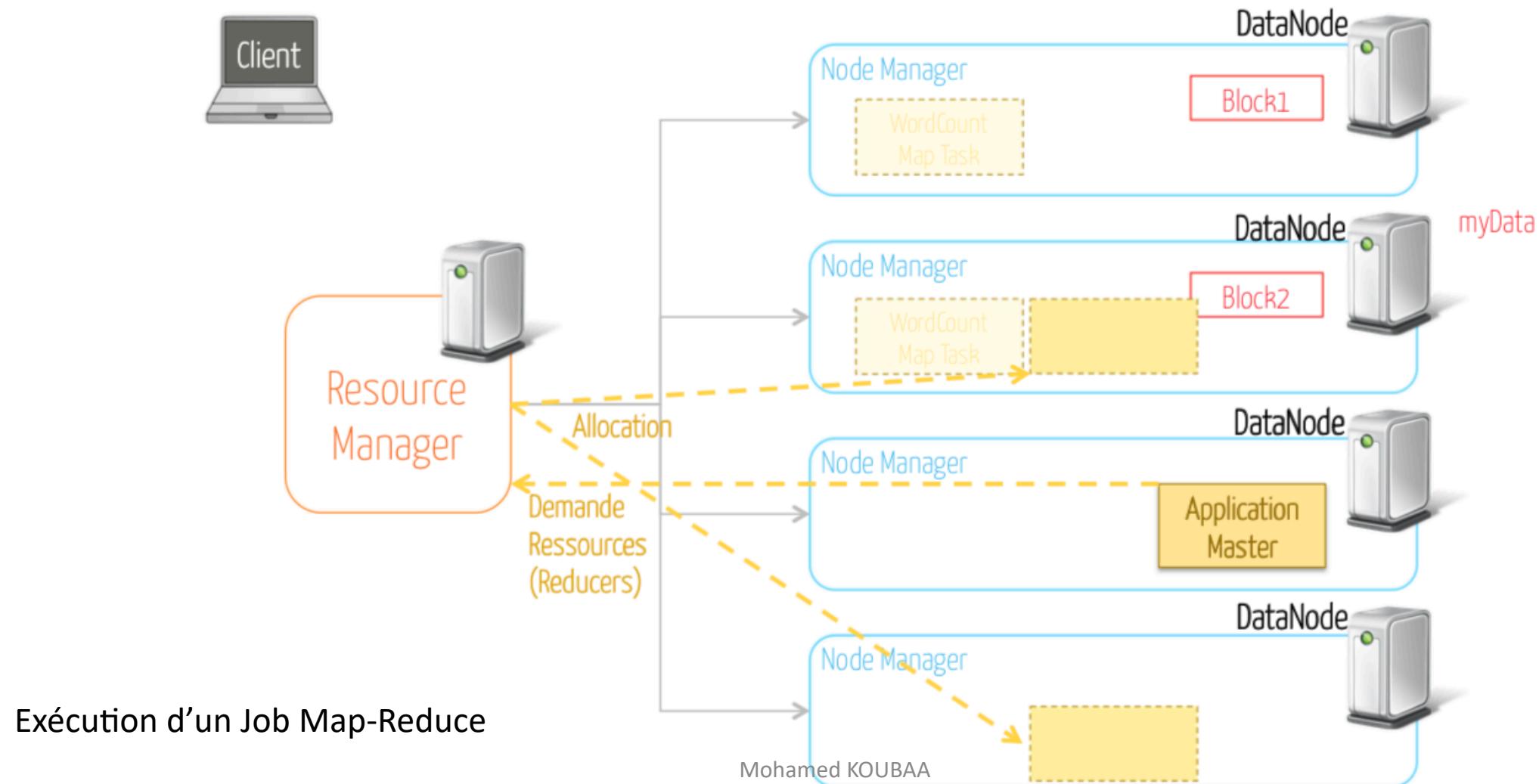
Map Reduce v2



Map Reduce v2



Map Reduce v2



Map Reduce v2

