INSTITUT SUPERIEUR D'INFORMATIQUE DU KEF

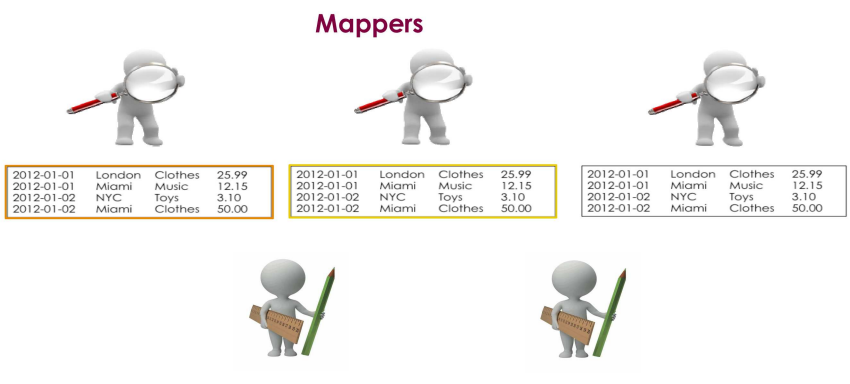
RÉPUBLIQUE TUNISIENNE MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ DE JENDOUBA
INSTITUT SUPÉRIEUR D'INFORMATIQUE DU KEF

CHAPITRE 4
MAPREDUCE

Hafedh Fechichi
ferchichi.hafedh@gmail.com
ISI Kef
Février 2021
Mastère SIW

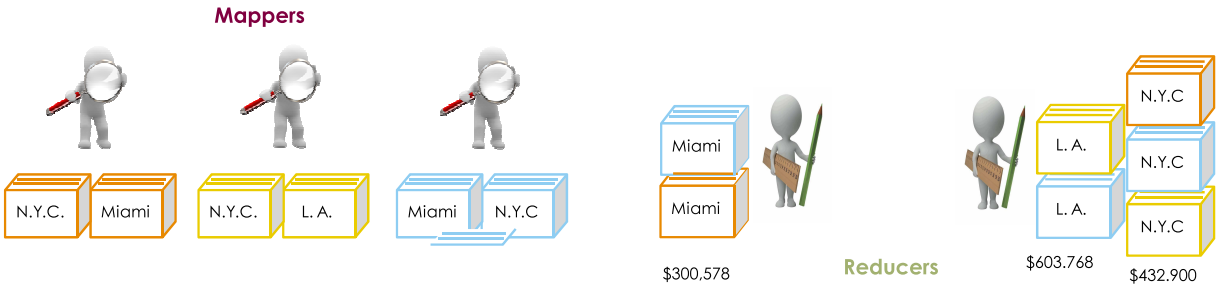
Exemple (1) de motivation

- **Map-Reduce : Moyen plus efficace et rapide de traitement des données massives.**
- **Au lieu d'avoir une seule personne qui parcourt le livre, si on en recrutait plusieurs?**
  - Appeler un premier groupe les **Mappers** et un autre les **Reducers**
  - Diviser le livre en plusieurs parties, et en donner une à chaque Mapper
- **Les Mappers peuvent travailler en même temps, chacun sur une partie des données**



Exemple (2) de motivation

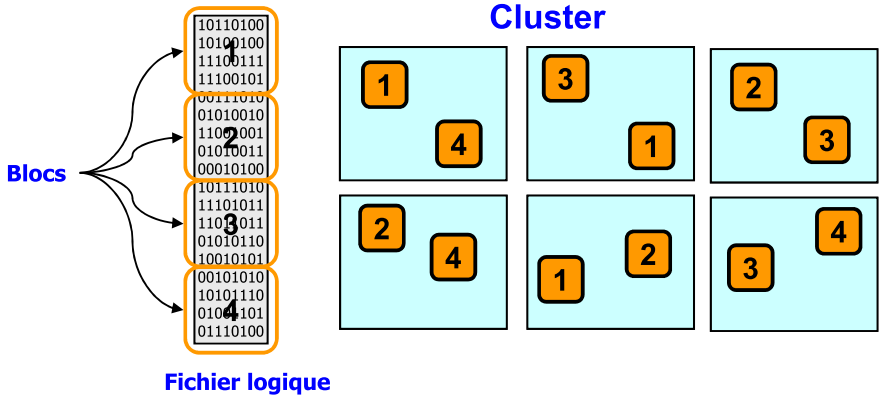
- **Mappers**
  - Pour chaque entrée, saisir la ville et le total de ventes dans une fiche
  - Rassembler les fiches d'un même magasin dans une pile
- **Reducers**
  - Chaque **reducer** sera responsable d'un ensemble de magasins
  - Collectent les fiches associées des différents **mappers**
  - Pour chaque ville, ils parcourent les piles en ordre alphabétique (Los Angeles avant Miami) et font la somme des enregistrements



Introduction à MapReduce

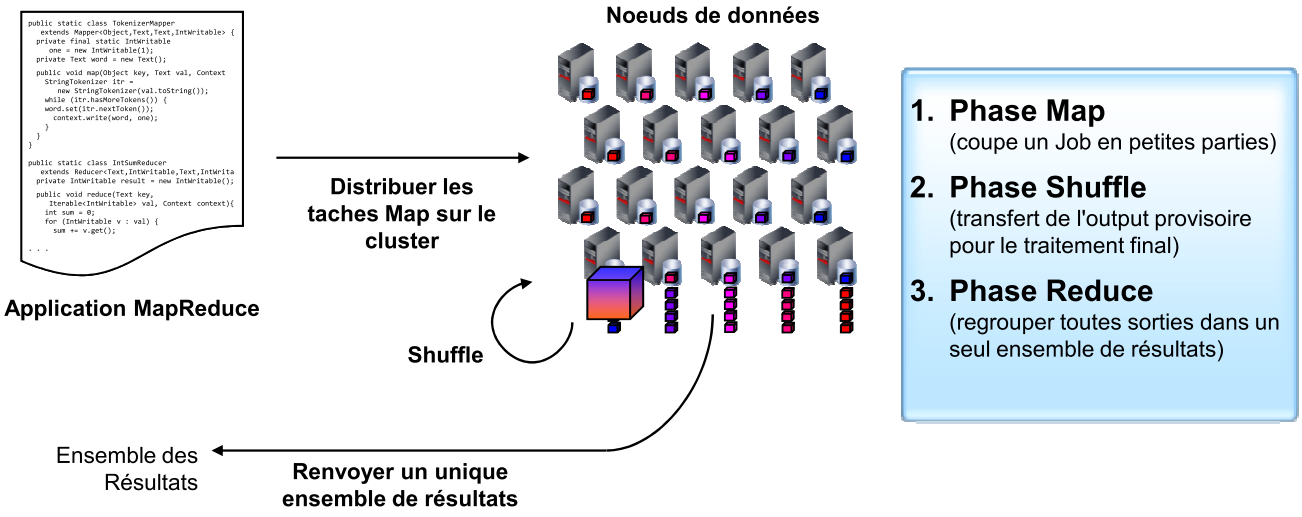
- **Principes de base**
  - Les données sont stockées à travers tout le cluster
  - Les programmes sont offerts aux données (pas les données aux programmes)

- **Les données sont stockées à travers tout le cluster (le DFS)**
  - Le cluster entier participe au système de fichier
  - Les blocs d'un seul fichier sont distribués à travers le cluster
  - Un bloc est répliqué pour la possibilité de réparation



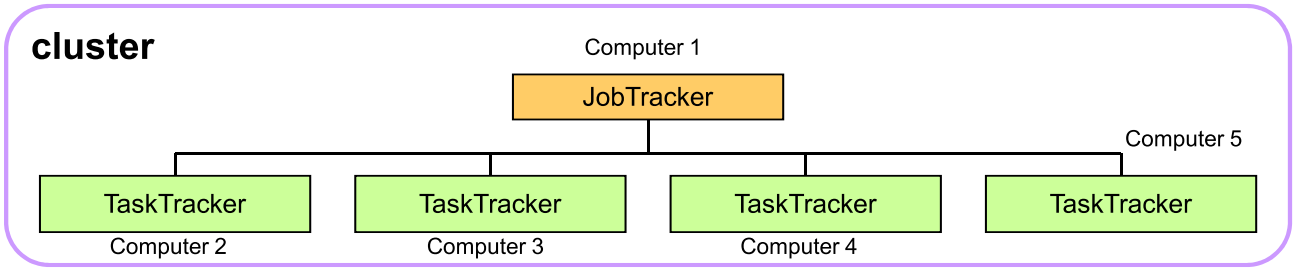
Introduction à MapReduce

- Modèle de calcul de Hadoop
  - Les données sont stockées dans un système de fichiers distribué traversant plusieurs noeuds machines.
  - Apporter les fonctions aux données
  - Distribuer l'application sur les ressources de calcul où les données résident
- Evolutif à des milliers de nœuds et des peta-bits de données



Moteur MapReduce

- Architecture Master / Slave
  - Un seul master (JobTracker) contrôle l'exécution des tâches sur plusieurs slaves (TaskTrackers).
- **JobTracker**
  - Accepte les tâches MapReduce soumises par les clients
  - Envoie les tâches Map et Reduce aux (nœuds) TaskTrackers
  - Conserve les tâches les plus proches (sur les machines physiques) possible des données
  - Surveille les tâches et le statut du TaskTracker
- **TaskTracker**
  - Exécute les tâches Map et Reduce
  - Envoie les rapports de monitoring au JobTracker
  - Gère le stockage et la transmission des résultats (output) intermédiaires

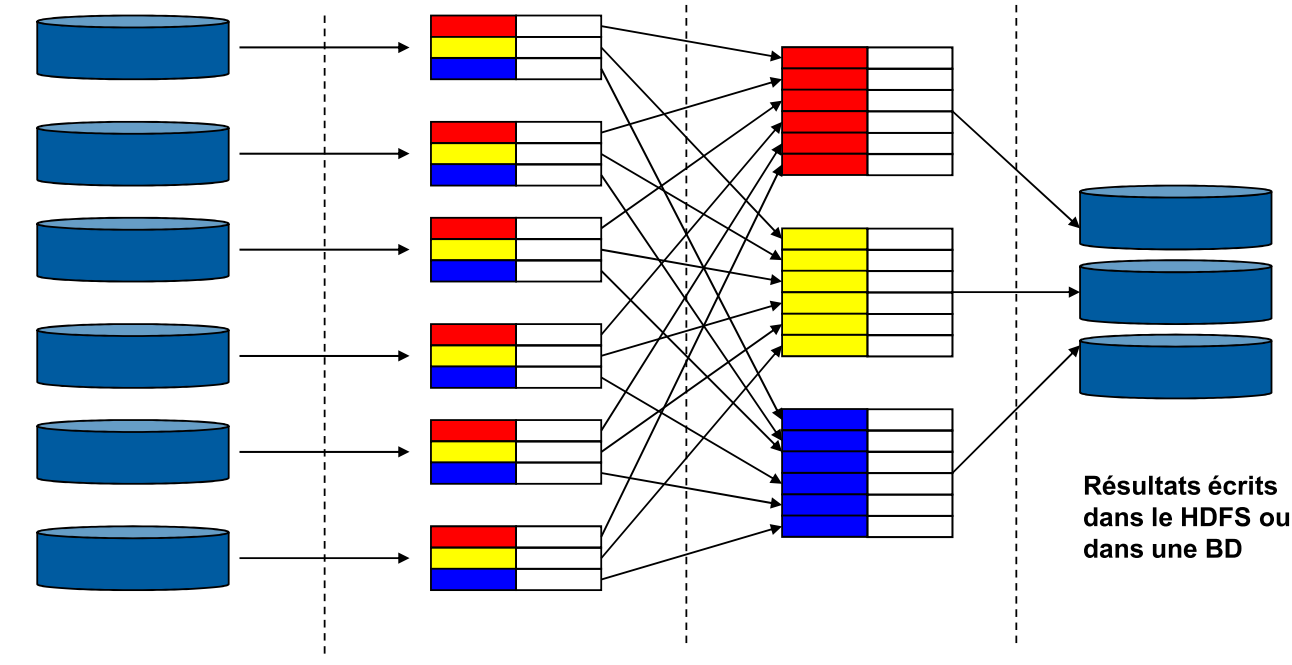


Le modèle de programmation MapReduce

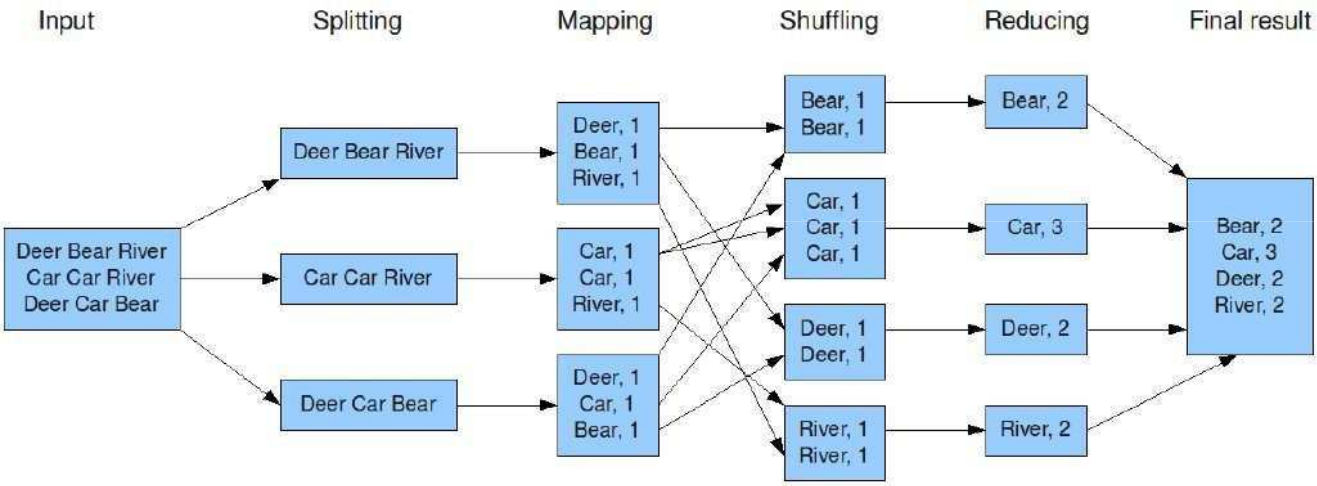
- **Phase "Map":**
  - Les données en entrée sont découpées en pièces/morceaux.
  - Les nœuds ***Workers*** traitent les pièces de données individuelles en parallèle (sous le contrôle global du **Job Tracker**)
  - Chaque nœud ***Worker*** stocke ses résultats dans son système de fichier local où un ***Reducer*** est capable d'accéder à ce dernier.

- **Phase "Reduce":**
  - Les données sont agrégées ('reduced" à partir de la phase *Map*) par les *workers* (nœuds) (sous contrôle du Job Tracker)
  - Plusieurs tâches *Reduce* peuvent paralléliser l'agrégation

Vue générale MapReduce

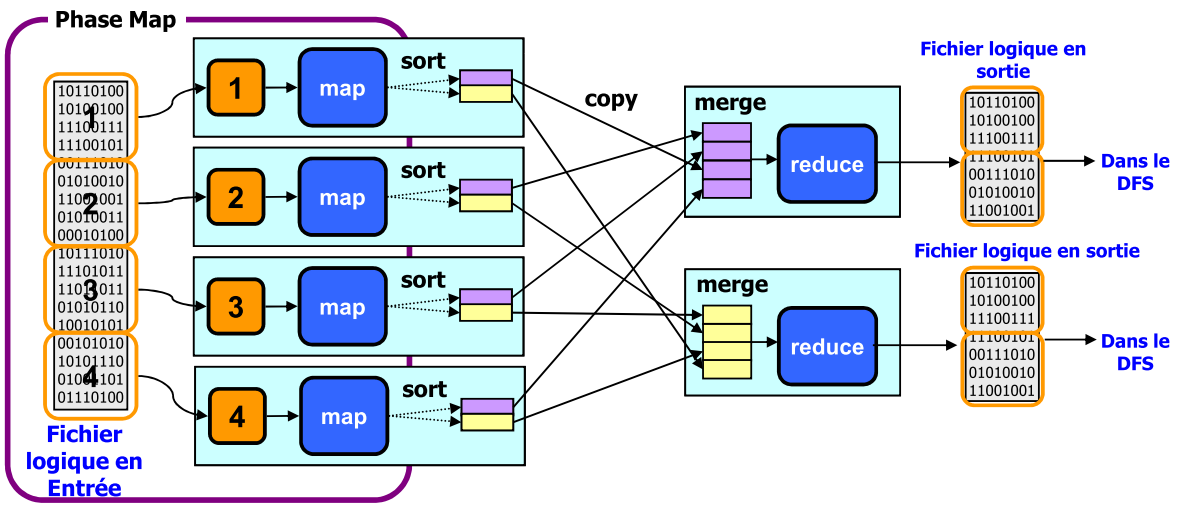


Exemple MAP REDUCE



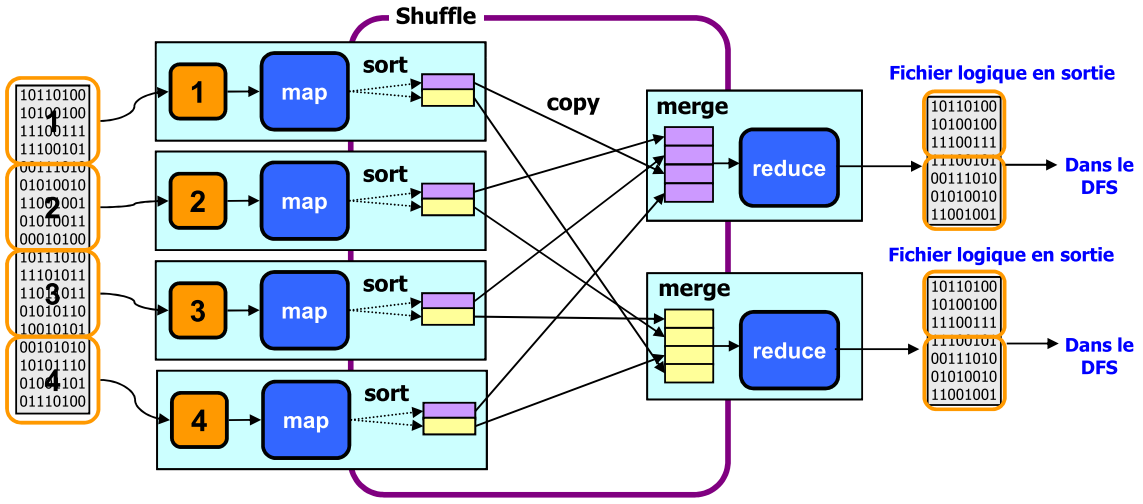
Phase Map

- **Mappers**
  - Petits programmes distribués à travers le cluster d'une manière locale aux données
  - Traitent une *portion* des données en entrée (appelée aussi *split*).
  - Chaque *Mapper* parcourt, filtre ou transforme son input
  - Produisent des pairs groupées `<key,value>`



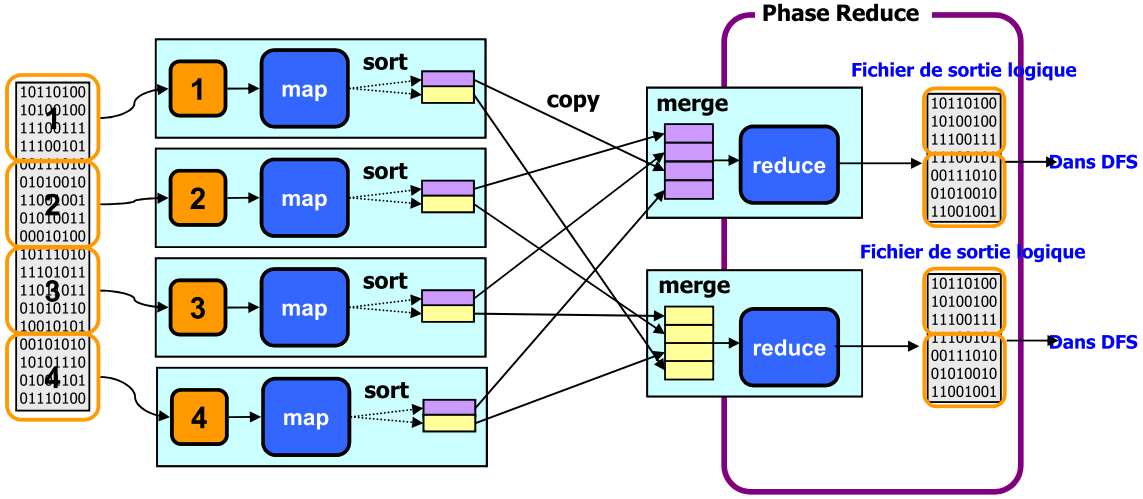
Phase Shuffle

- **Les sorties des Mapers sont groupés localement par clé.**
- **Pour chaque *clé*, un nœud est sélectionné pour traiter les données.**
- **Tout ce mouvement de données (shuffle) est orchestré par MapReduce d'une manière transparente.**



Phase Reduce

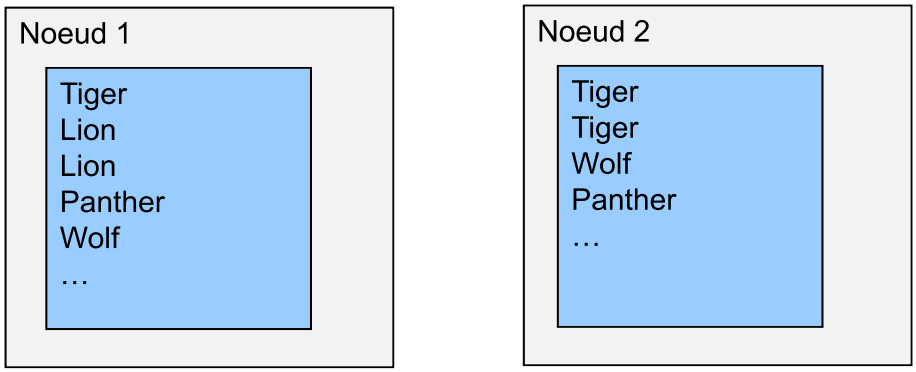
- **Reducers**
  - Petits programmes responsables de l'agrégation de toutes les *valeurs (*dans les pairs) correspondantes à la *clé* en question.
  - Chaque Reducer écrit la sortie/résultat du traitement dans son propre système de fichier.



Exemple : Word Count

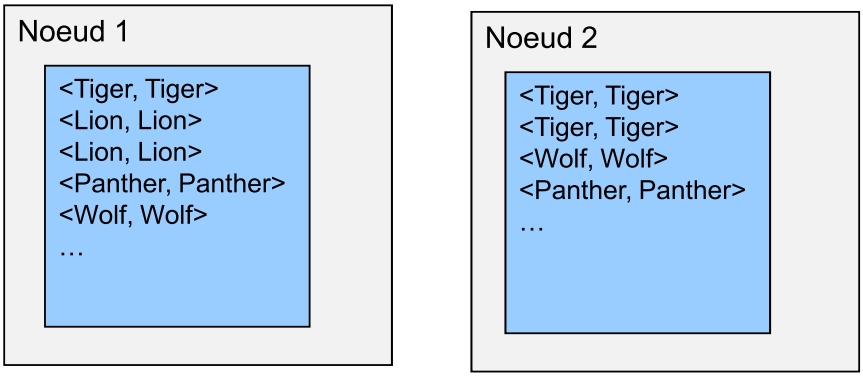
• Dans cet exemple, nous avons une liste de noms d'animaux
  - MapReduce peut découper les fichiers automatiquement selon les sauts de ligne.
  - Le fichier est découpé en deux blocs sur deux nœuds.
  - L'objectif est de connaitre le nombre d'occurences de chaque animal. Le traitement SQL est comme suit:

```
SELECT COUNT(NAME) FROM ANIMALS
       WHERE NAME IN (Tiger, Lion …)
       GROUP BY NAME;
```



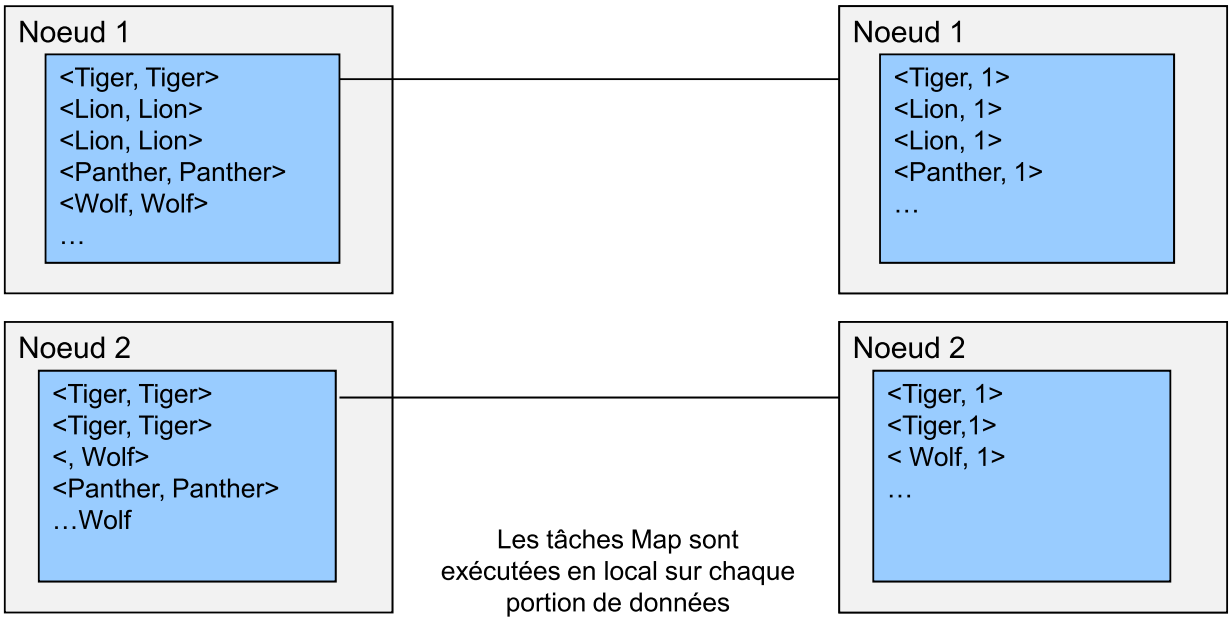
Exemple : Entrée de la phase Map

• Les tâches Map nécessites des pairs <***clé/valeur*** > comme entrées
• Si aucune clé n'est disponible, on doit la fabriquer
• Le mapping des données en entrée (fichiers, liens web, etc.) aux pairs <**clé, valeur**> est réalisé dans la classe **`InputFormat`**



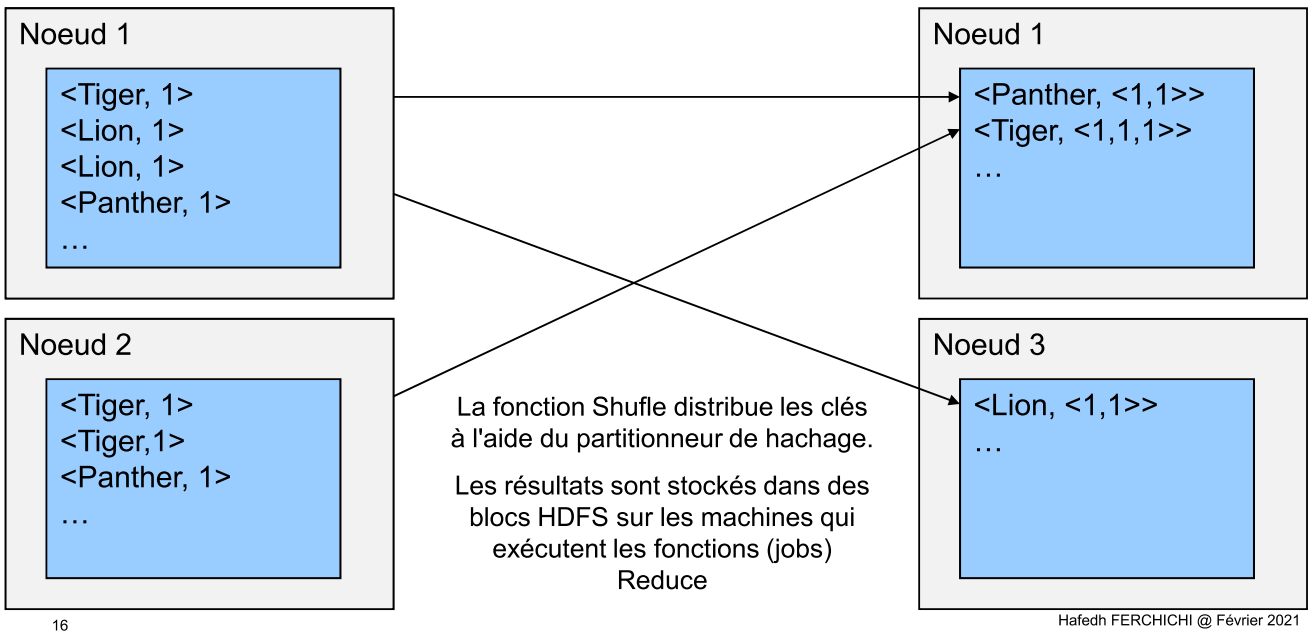
Exemple : Tâche Map

- Dans notre fonction Map
  - Préparer le compte par la transformation en <*Text(nom), Integer(1)*>



Exemple : Phase Shuffle

- Déplacer toutes les valeurs correspondantes à la même clé dans le même nœud cible.
- La distribution est effectuée en utilisant la classe **Partitioner**
- Les fonctions Reduce peuvent s'exécuter sur des nœuds aléatoires (dans notre exemple Nœud 1 et 3).
  - Les nombres de fonctions ***Map*** et ***Reduce*** ne sont pas nécessairement identiques

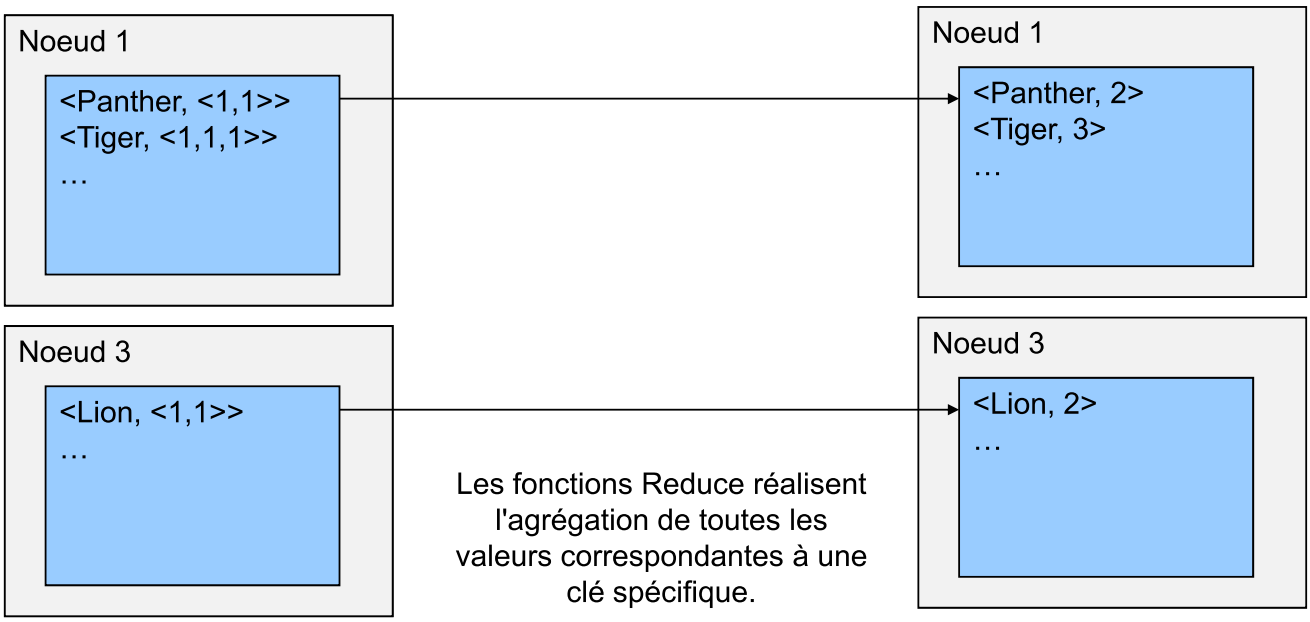


Persistance des données dans Shuffle

- Les données sont persistés dans le DFS pendant la phase ***Shuffle***.
  - Permet aux fonction ***Map*** et ***Reduce*** de redémarrer.
- Plusieurs sorties d'une fonction ***Map*** doivent être fusionnées en une seule entrée pour la fonction ***Reducer***
  - La fusion est effectuée en mémoire

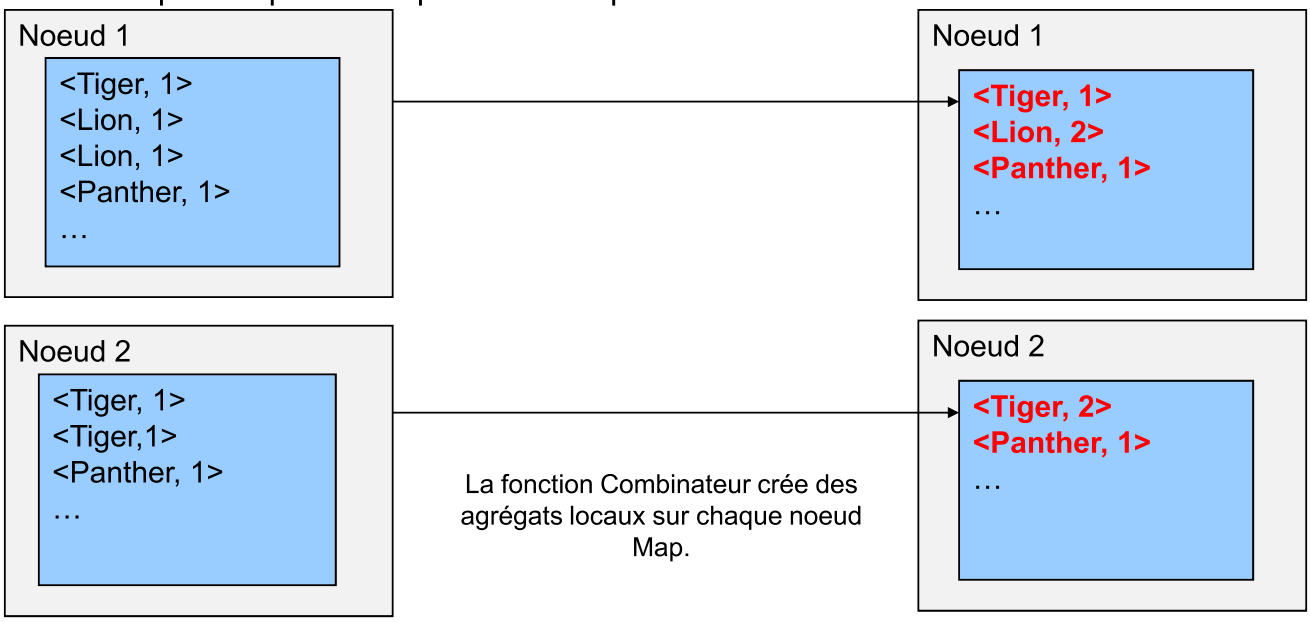
Exemple : Phase Reduce

La fonction Reduce calcule, pour chaque clé, les valeurs agrégée
La sortie de la fonction Reduce est écrite dans le DFS



Exemple : Combiner (optionnel)

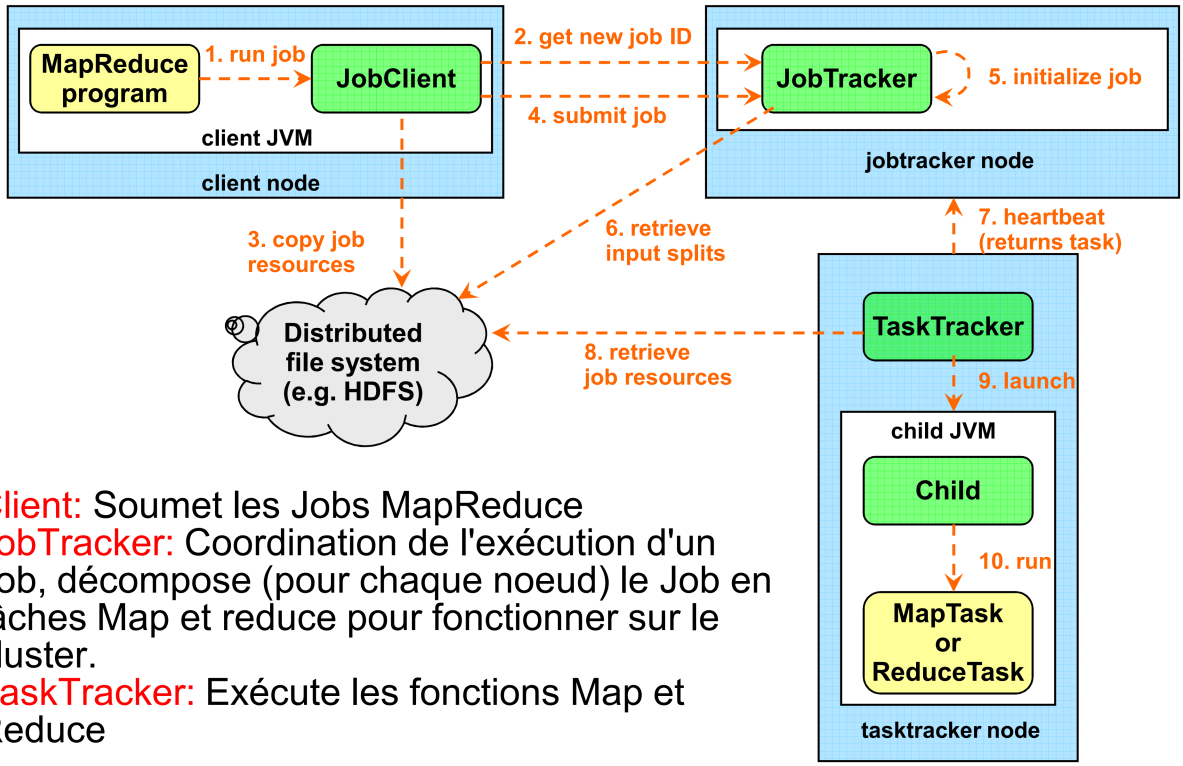
• Pour des raisons de performance, un agrégat local dans la fonction Map peut être utile.
• Réduit le taux de données qui doivent être copiées à travers le réseau et minimise l'effort de fusion.
• Lancé après la phase Map et avant la phase Shuffle.



Tâches Map/Reduce

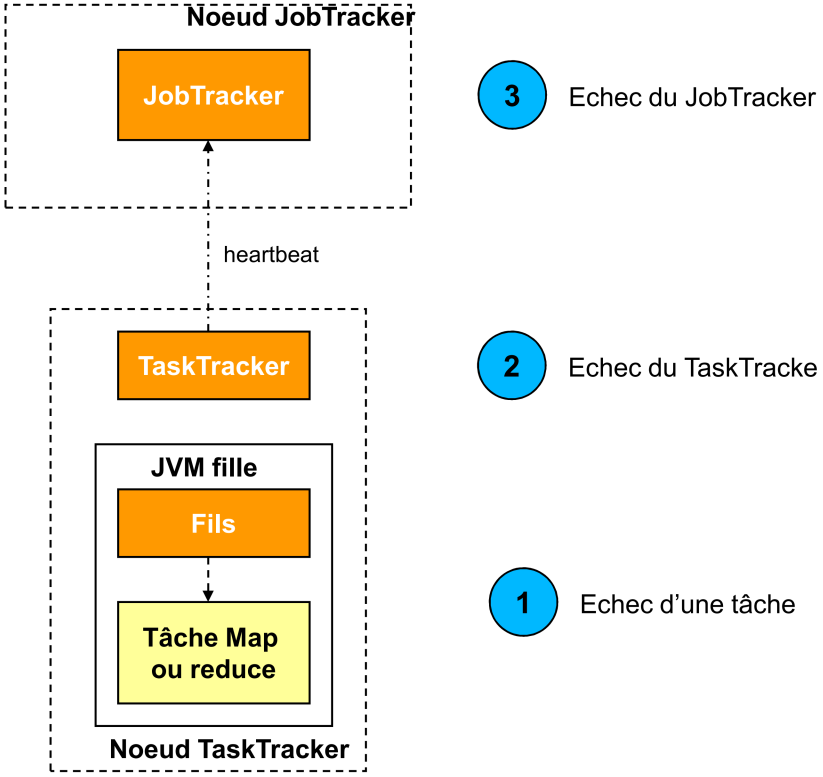
- **Exécution locale**
  - Hadoop essaie d'exécuter les splits en local
  - Si aucun slot local n'est disponible le split sera déplacer à la fonction Map
- **Nombre de tâches Map**
  - Il est possible de configurer le nombre fonctions Map et Reduce
  - Si un fichier n'est pas découpable, il y aura une unique fonction Map
- **Nombre de tâches Reduce**
  - En général, le nombre de tâches Reduce est inférieur à celui des tâches Map
  - La sortie d'une fonction Reduce est écrit en local sur le HDFS
- **Exécution redondante**
  - Il est possible de configurer deux tâches Map ou plus qui seront lancées pour chaque split
    - La première fonction Map qui se termine sera gagnante
    - Dans les systèmes avec un grand nombre de machines (pas chers), ceci permet d'augmenter la performance.
    - Dans les systèmes avec un nombre limité de machines, ceci peut réduire la performance.

Exécution des Jobs mapReduce dans Hadoop



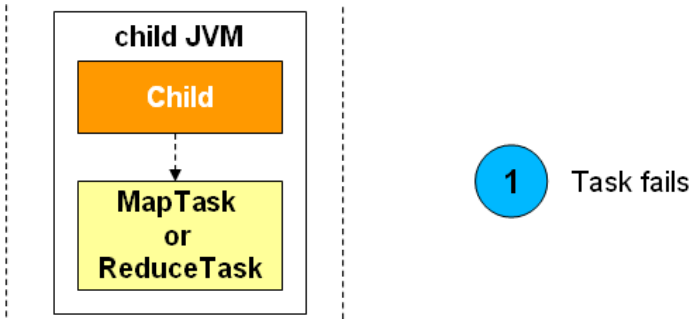
- Client: Soumet les Jobs MapReduce
- JobTracker: Coordination de l'exécution d'un Job, décompose (pour chaque noeud) le Job en tâches Map et reduce pour fonctionner sur le cluster.
- TaskTracker: Exécute les fonctions Map et Reduce

Tolérance aux fautes



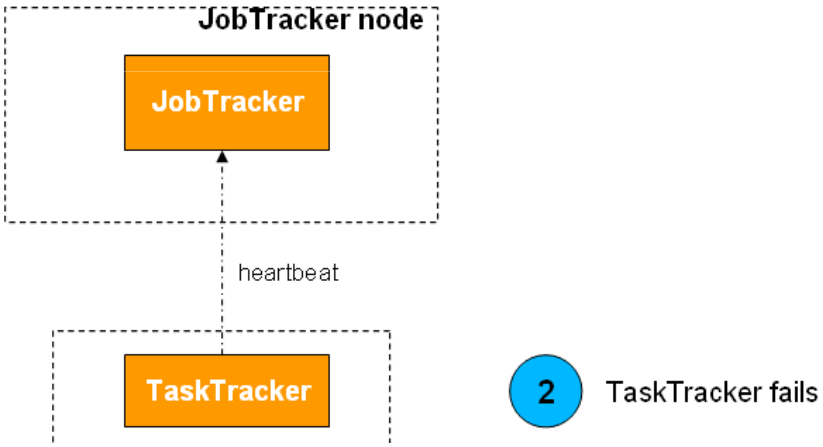
Tolérance aux fautes

- **Echec d'une fonction**
  - Si une fonction fille échoue, la JVM fille envoies un rapport au TaskTracker avant qu'elle quitte et une autre fonction s'exécute.
  - Si la fonction fille se bloque, elle sera détruite. Le JobTracker re-planifie la tâche sur une autre machine..
  - Si l'échec de la fonction persiste la tâche va échouer.



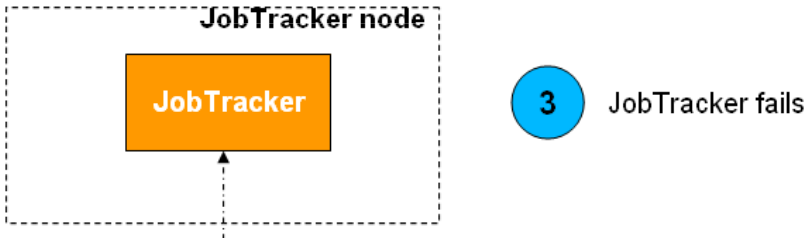
Tolérance aux fautes

- **Echec du TaskTracker**
  - JobTracker ne reçoit aucun signe du TaskTracker
  - Supprime le TaskTracker du groupe de TaskTrackers pour planifier des tâches.



Tolérance aux fautes

- **Echec du JobTracker**
  - Point de défaillance unique (problème d'exécution d'un Job)



Conclusion

- **Un job MAP-REDUCE se décompose en 7 tâches**
  1. Configuration du **Job MAP-REDUCE**
  2. Découpage du fichier d'entrée en **BLOC de taille fixe** par le **HDFS**
  3. Création de « **M** » **Task Tracker** correspondant aux « **M** » **BLOC de fichier**
  4. Application de la fonction **MAP** et sérialisation sur le **disque dur local du nœud**
  5. **Tri** des paires de (clés, valeurs) et passage à la fonction **REDUCE** pour traitement
  6. Application de la fonction **REDUCE** et sérialisation le **disque dur du nœud**
  7. Sérialisation des « **R** » fichiers **REDUCE** dans un SGBD par exemple

Exemple d'application 1

La classe Mapper

```java
package Test;
import java.io.IOException;

public class MapperWC extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private static final Pattern WORD_BOUNDARY = Pattern.compile("\\s*\\b\\s*");
    public void map(LongWritable offset, Text lineText, Context context) throws IOException, InterruptedException {
        String line = lineText.toString();
        Text currentWord = new Text();
        //tableau de mots (String [])
        for (String word : WORD_BOUNDARY.split(line)) {
            if (word.isEmpty()) {
                continue;
            }
            currentWord = new Text(word);
            context.write(currentWord, one);
        }
    }
}
```


Exemple d'application 1

La classe Reducer

```java
package Test;
import java.io.IOException;


public class ReducerWC extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable count : counts) {
            sum += count.get();
        }
        context.write(word, new IntWritable(sum));
    }
}
```

Exemple d'application 1

La classe Main

```
1  package Test;
2
3  import org.apache.hadoop.conf.Configured;
15
16 public class MainWC extends Configured implements Tool {
17
18     private static final Logger LOG = Logger.getLogger(MainWC.class);
19
20     public static void main(String[] args) throws Exception {
21         int res = ToolRunner.run(new MainWC(), args);
22         LOG.debug("Job is Finished");
23         System.exit(res);
24     }
25
```

```
public int run(String[] args) throws Exception {
    Job job = Job.getInstance(getConf(), "wordcount");
    Path inputFilePath = new Path(args[0]);
    Path outputFilePath = new Path(args[1]);
    job.setJarByClass(this.getClass());
    // Use TextInputFormat, the default unless job.setInputFormatClass is used
    FileInputFormat.addInputPath(job, inputFilePath);
    FileOutputFormat.setOutputPath(job, outputFilePath);
    job.setMapperClass(MapperWC.class);
    job.setReducerClass(ReducerWC.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileSystem fs = FileSystem.newInstance(getConf());
    if (fs.exists(outputFilePath)) {
        fs.delete(outputFilePath, true);   }

    return job.waitForCompletion(true) ? 0 : 1;
} }
```

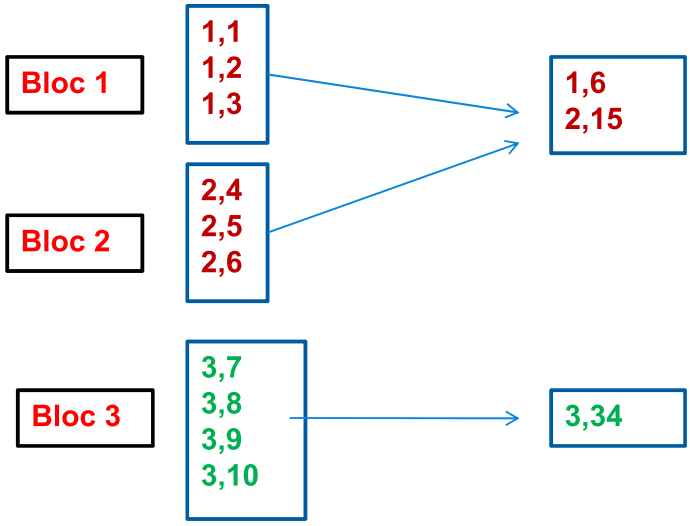

Exemple d'application 2

- **Écrire un algorithme qui permet de calculer la somme des dix nombres consécutifs entre 1 et 10.**
  - Comment faire pour transformer dix nombre en paires de clés/valeurs et agréger les valeurs de façon à obtenir leurs somme consécutive ?
  - Qu'est ce que vous désignez comme clé et qu'est ce que vous désignez comme valeur ?

```
Algorithme Somme
Var  x, i : entier
Début
        X=0
        Pour i de 1 à 10 faire
                X= x+i
        Fin pour
        Ecrire (x)
Fin
```

Exemple d'application 2

- Supposons que vous décidiez de diviser les 10 nombres en trois blocs (1,2 et 3)
- La clé de chaque paire est le numéro du bloc dans lequel le nombre apparait et la valeur est le nombre en question



Exemple d'application 2

- L'avantage c'est que vous n'avez pas à spécifier dans votre code la façon dont le fichier doit être divisé en bloc
- La façon dont les blocs doivent être traités en parallèle
- Les opérations intermédiaire entre le Map et le reduce

```
Fonction Map (Fichier _des_10 _nombres)
        Colonne « clé » = ID-Bloc
        Colonne « Valeur » = Fichier _des_10 _nombres (nombre)
```

```
Fonction Reduce (Clés_Map, Liste_des_nombres)
        Total = 0
        Pour val In Liste_des_nombres
                Total = Total + val
        Emettre (clé, Total)
```

Exemple d'application 3

Employés

Nom	Dép_ID	Fonction
Ali	111	Cadre
Sami	999	Cadre Sup
Amira	999	Cadre Sup

Département

Dép_ID	Département
111	Comptabilité
999	Finances

On souhaite obtenir pour chaque employé la liste des département auxquels il est assigné. En d'autre terme, on souhaite obtenir la table suivante :

Nom	Dép_ID	Département	Fonction
Ali	111	Comptabilité	Cadre
Sami	999	Finances	Cadre Sup
Amira	999	Finances	Cadre Sup

Exemple d'application 3 - Map-Reduce

- **Ajouter dans chaque table la colonne du nom de la table et faire une concaténation verticale des tables**
- **Passer le fichier obtenu à la fonction Map( ). La clé de la paire sera la colonne qui lie les 2 tables.**
- **Le reduce obtient en entrée les clés avec leur liste de valeurs ensuite il les regroupe par clé de façon à obtenir les lignes des 2 tables correspondant à la même clé (la jointure)**

Exemple d'application 3 - Map-Reduce

Etape 1 : Tables concaténées verticalement

Employé s	Ali	111	Cadre
Employé s	Sami	999	Cadre Sup
Employé s	Amira	999	Cadre Sup
Département	111	Comptabilité	
Département	999	Finances	

Etape 2 : Phase MAP , la clé est la colonne que les 2 tables possèdent en commun et la valeur la liste des autres colonnes

Clé =111 , valeur = (Employés, Ali, Cadre)
Clé =999 , valeur = (Employés, Sami, Cadre Sup)
Clé =999 , valeur = (Employés, Amira, Cadre Sup)
Clé =111 , valeur = (Département, Comptabilité)
Clé =999 , valeur = (Département, Finance)

Etape 3 : phase Reduce, Regroupement des valeurs par clé

Clé =111 , valeur = [(Employés, Ali, Cadre) , (Département, Comptabilité)]
Clé =999 , valeur = [(Employés, Sami, Cadre Sup),
(Employés, Amira, Cadre Sup),
(Département, Finance)]