Programmation MAP REDUCE

Map-Reduce et programmation fonctionnelle

Map

- Applique une fonction sur chaque élément d'une liste
- Retourne une liste de résultats
- Map(f(x),X[1:n]) à [f(X[1]),...,f(X[n])]

Exemple :

Map (X^2,[0,1,2,3,4,5])=[0,1,4,9,16,25]

Reduce/fold

- Fait l'itération d'une fonction sur une liste d'éléments
- Applique la fonction sur les résultats précédents et l'élément courant
- Retourne un et un seul résultat,

Exemple :

Reduce (**x+y**,[0,1,2,3,4,5])=(((((0+1)+2)+3)+4)+5)=15

Map-Reduce: Exemple 1

-Calcul des ventes-

Imaginons que vous ayez plusieurs magasins que vous gérez à travers le monde

2018-01-01	Miami	Clothes	25.99
2018-01-01	Miami	Music	12.15
2018-01-02	NYC	Toys	3.10
2018-01-02	Miami	Clothes	50.00
2017-05-16	London	Toys	25.19

Objectif

Calculer le total des ventes par magasin pour l'année 2018

Clef	Valeur
Miami	88.14
NYC	3.10

Map-Reduce: Exemple 1

-Calcul des ventes-

❑ Approche traditionnelle

- Si on utilise les hashtables sur 1To, Problèmes ?
  - ❑ Ça ne marchera pas ?
  - ☑ Problème de mémoire ?
  - ☑ Temps de traitement long ?
  - ❑ Réponses erronées ?
- Le traitement séquentiel de toutes les données peut s'avérer très long
- Plus on a de magasins, plus l'ajout des valeurs à la table est long
- Il est possible de tomber à court de mémoire pour enregistrer cette table
- Mais cela peut marcher, et le résultat sera correct

2018-01-01	Miami	Clothes	25.99
2018-01-01	Miami	Music	12.15
2018-01-02	NYC	Toys	3.10
2018-01-02	Miami	Clothes	50.00
2017-05-16	London	Toys	25.19

Map-Reduce: Exemple 1

-Calcul des ventes-

❑ Approche Big Data

On suppose que le livre sur HDFS est découpé en deux parties:

2018-01-01	Miami	Clothes	25.99
2018-01-01	Miami	Music	12.15
2018-01-02	NYC	Toys	3.10
2018-01-02	Miami	Clothes	50.00
2017-05-16	London	Toys	25.19

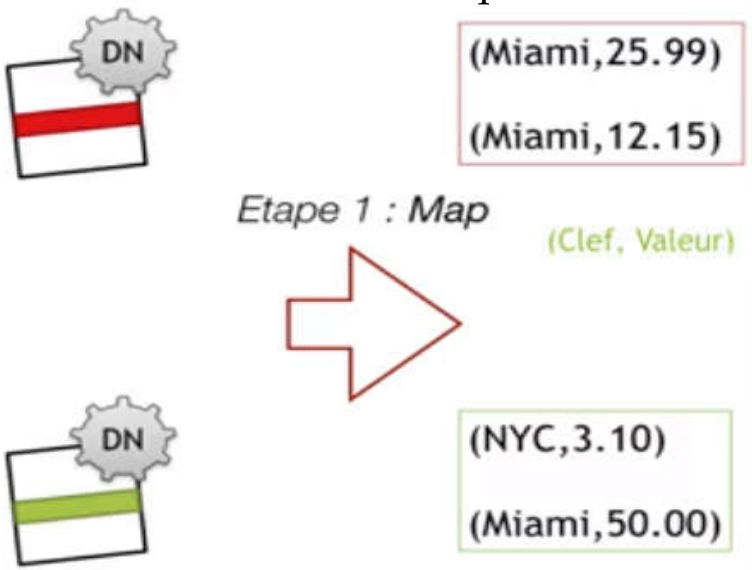
Map-Reduce: Exemple 1

-Calcul des ventes-

❑ Etape 1: MAP

- ❑ Faire des petits traitements en parallèle ligne par ligne: Pas d'opérations de calculs entre plusieurs lignes.
- ❑ 2 Types de filtrage dans cet exemple:
  - **Filtrage vertical :** diminuer le nombre de colonnes: nom magasin, valeur des ventes .
  - **Filtrage horizontal**: supprimer les lignes non concernées (année autre que 2018).

2018-01-01	Miami	Clothes	25.99
2018-01-01	Miami	Music	12.15
2018-01-02	NYC	Toys	3.10
2018-01-02	Miami	Clothes	50.00
2017-05-16	London	Toys	25.19

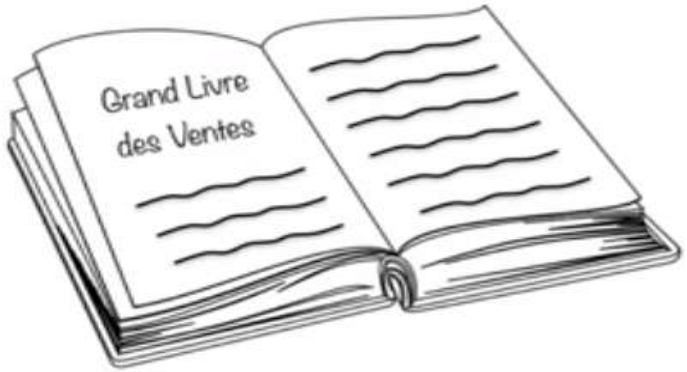


Map-Reduce: Exemple 1

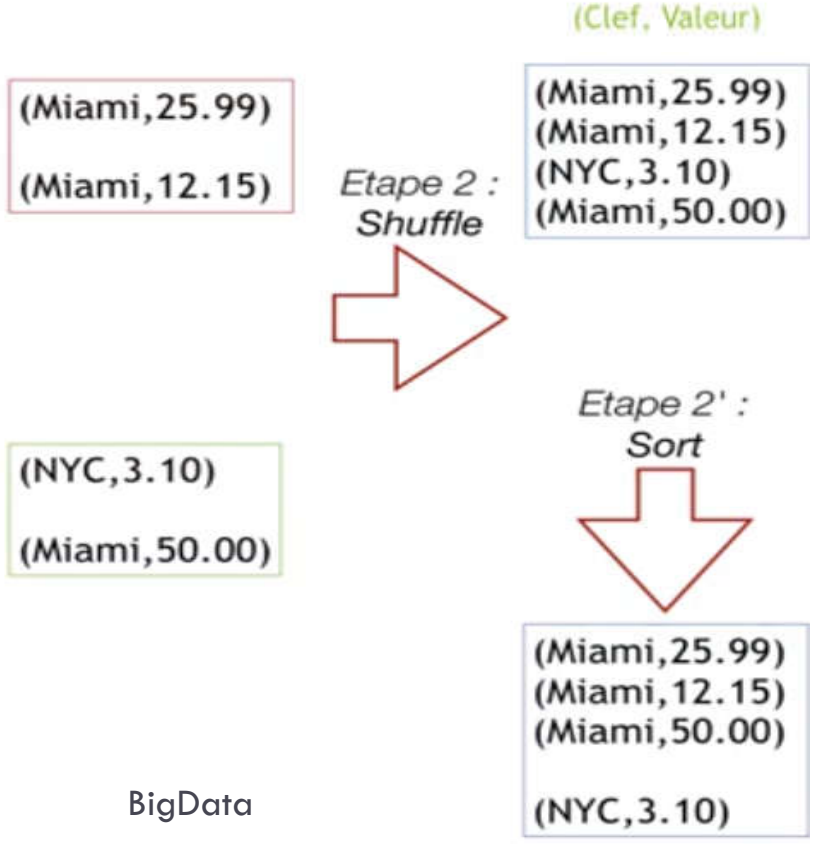
-Calcul des ventes-

❑ Etape 2: Sort and Shuffle

- ❑ Etape faite automatiquement par Hadoop, composée de deux sous étapes:
  - **Shuffle:** Rassemblement des données dans la même machine
  - **Sort:** Tri par clé pour regrouper les ventes de même magasin successivement



2018-01-01	Miami	Clothes	25.99
2018-01-01	Miami	Music	12.15
2018-01-02	NYC	Toys	3.10
2018-01-02	Miami	Clothes	50.00
2017-05-16	London	Toys	25.19



Map-Reduce: Exemple 1

-Calcul des ventes-

❑ Etape 3: Reduce

- ❑ Comme le mapper, le code de reducer doit être écrit par l'utilisateur
- ❑ Ici; faire la somme des ventes par magasin.

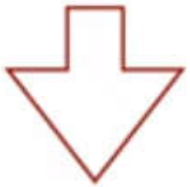
(Clef, Valeur)

(Miami,25.99)
(Miami,12.15)
(Miami,50.00)

(NYC,3.10)

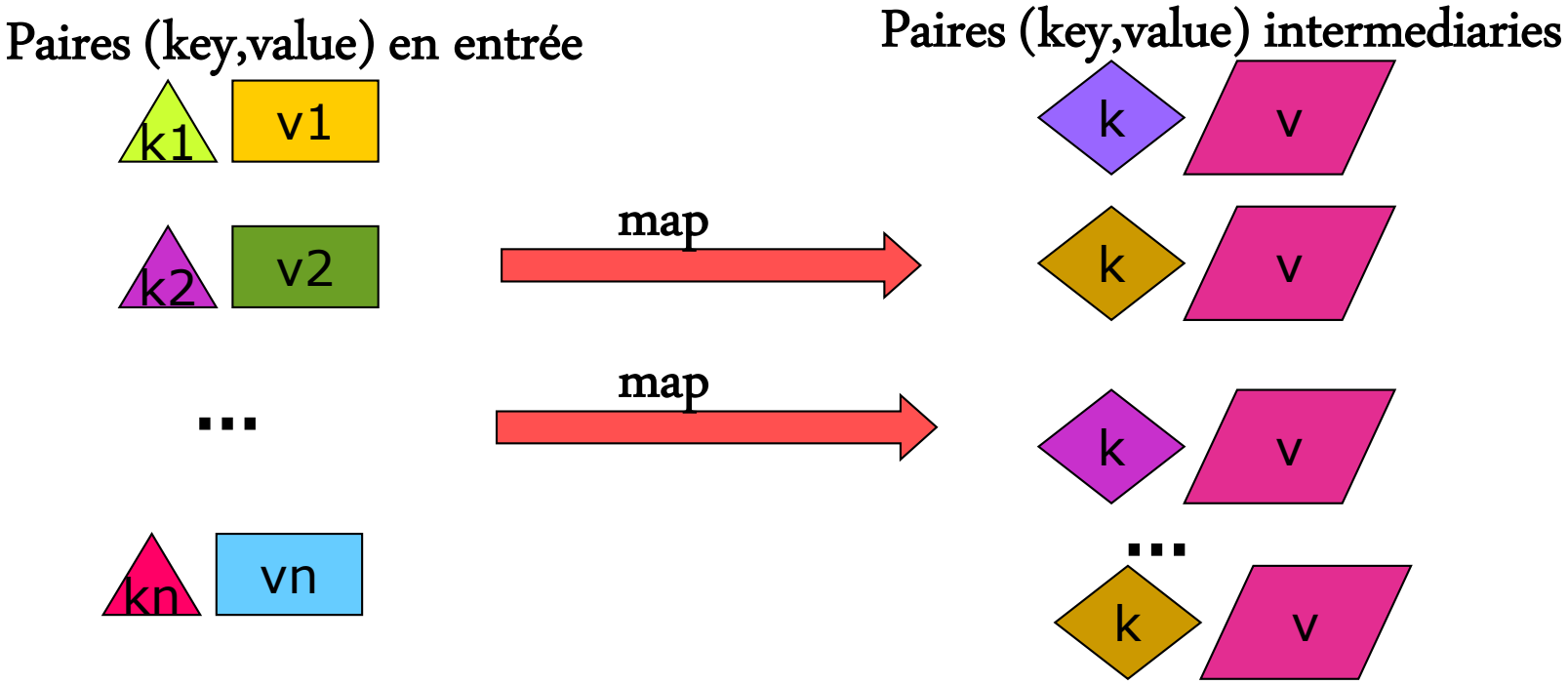
2018-01-01	Miami	Clothes	25.99
2018-01-01	Miami	Music	12.15
2018-01-02	NYC	Toys	3.10
2018-01-02	Miami	Clothes	50.00
2017-05-16	London	Toys	25.19

Etape 3 : Reduce



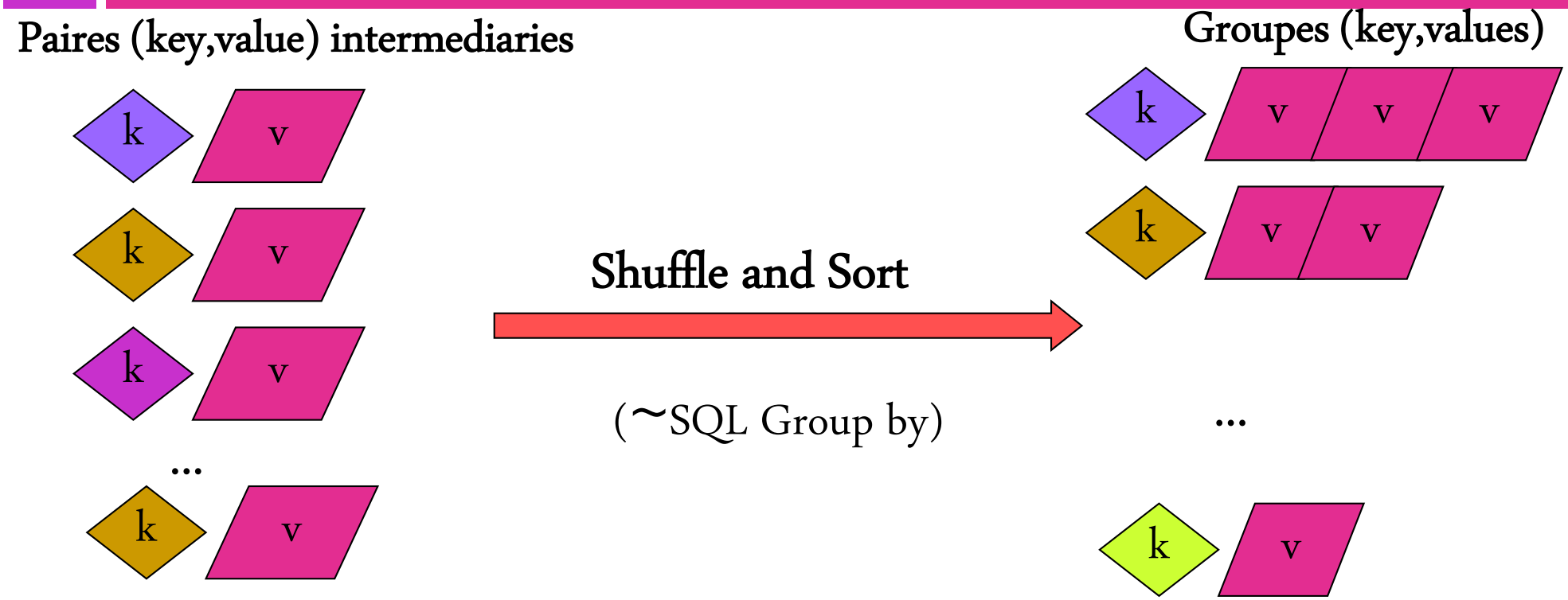
(Miami , 88.14)
(NYC , 3.10)

A retenir: L'étape MAP



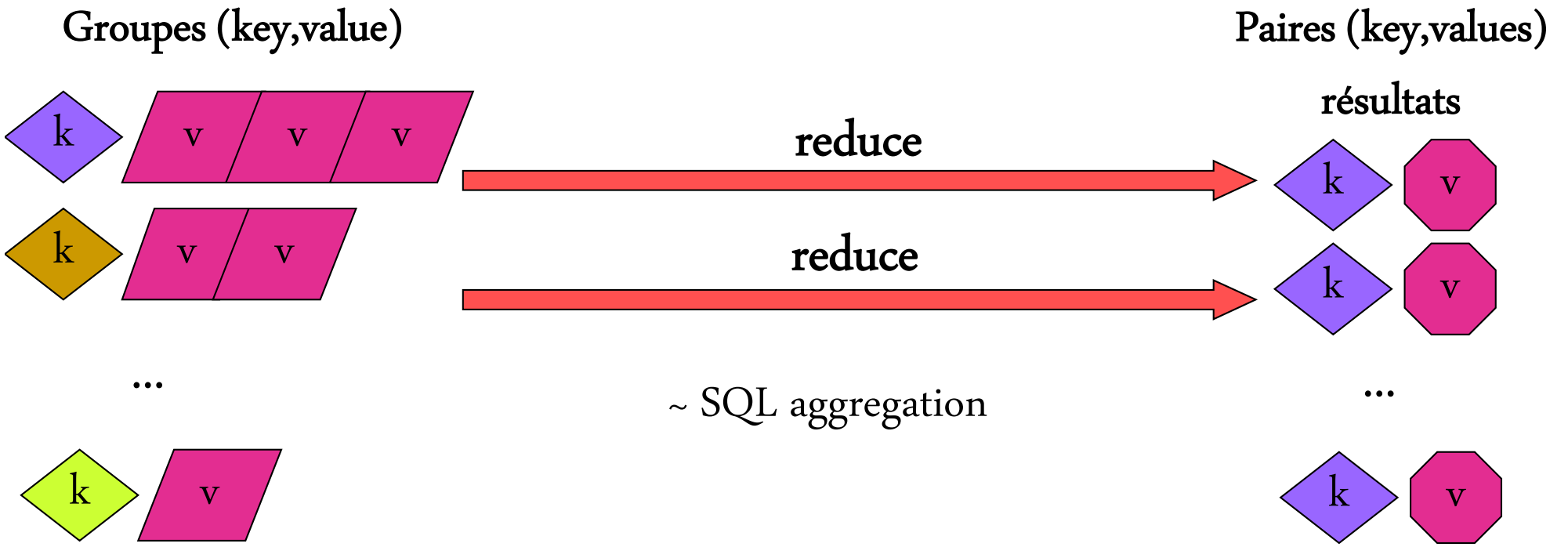
- Dans l'étape **Map,** le but est de partir d'un couple <clé, valeur> et d'y associé de nouveaux couples <clé, valeur>. Les clés en entrée sont différentes des clés produites par le Map.
- Le nombre de tâches Map ne dépend pas du nombre de nœuds, mais du nombre de blocs de données en entrée. Chaque bloc se fait assigner une seule tâche Map.

A retenir : L'étape Shuffle and Sort



- C'est un étape **automatique de regroupement et tri** s'intègre entre MAP et REDUCE pour la redistribution des données afin que les paires produites par Map ayant les mêmes clés soient sur les mêmes machines.

A retenir : L'étape REDUCE

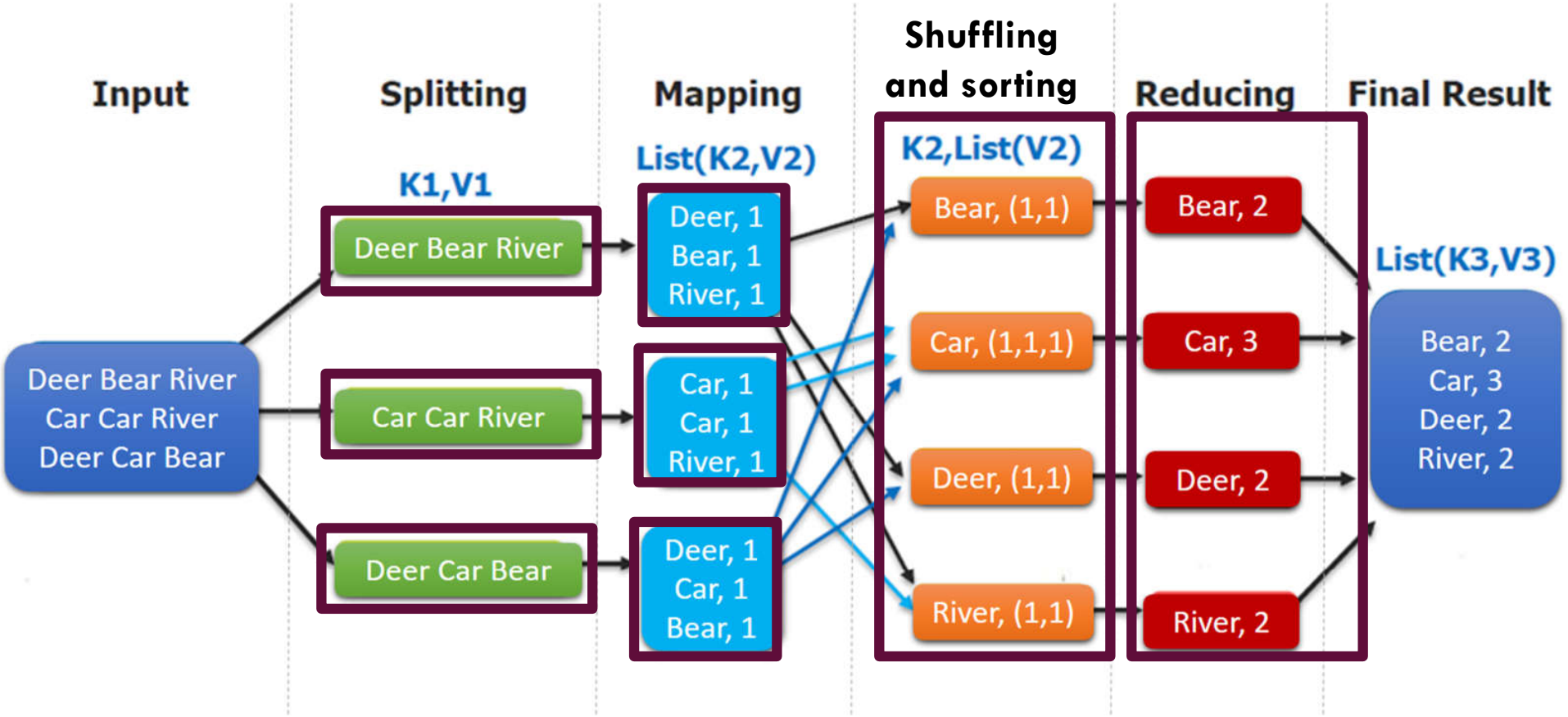


- Dans l'étape **Reduce** le but est d'associer toutes les valeurs correspondantes à la même clé. On souhaite donc rassembler tous les couples <clé, valeur>.
- Pour le traitement, les taches Reduce suivent le même schéma que les taches Map.
- Elles n'ont pas à s'exécuter parallèlement et dès qu'un nœud fini son traitement un autre lui est aussitôt assigné.

Map-Reduce: Exemple 2

-Word Count-

- Word Count: Comptage des occurrences des mots



Map-Reduce: Exemple 2

-Word Count-

☐ Pseudocodes du map et du reduce

```
map(key, text): # input: key=position, text=line
  for each word in text:
    Emit(word,1) # outputs: key/value

reduce(key, list of values): # input: key == word, our mapper output
  count = 0
  for each v in values:
    count += v
  Emit(key, count) # it is possible to emit multiple (key, value) pairs here
```

Map-Reduce: Exemple 3

-Graphe Social-

- On administre un réseau social comportant des millions d'utilisateurs.
- Pour chaque utilisateur, on a dans notre BD la liste des utilisateurs qui sont ses amis sur le réseau (via une requête SQL).
- On souhaite afficher quand un utilisateur va sur la page d'un autre utilisateur une indication *Vous avez N amis en commun* ;
- On ne peut pas se permettre d'effectuer une série de requêtes SQL à chaque fois que la page est accédée (trop lourd en traitement).

$\Rightarrow$ On va donc développer des programmes Map et Reduce pour cette opération et exécuter le traitement toutes les nuits sur notre BD, en stockant le résultat dans une nouvelle table.

Map-Reduce: Exemple 3

-Graphe Social-

Ici, nos données d'entrée sont sous la forme Utilisateur ⇒ Amis :

$A \Rightarrow B, C, D$
$B \Rightarrow A, C, D, E$
$C \Rightarrow A, B, D, E$
$D \Rightarrow A, B, C, E$
$E \Rightarrow B, C, D$

- Puisqu'on est intéressé par l'information *amis en commun entre deux utilisateurs* et qu'on aura à terme une valeur par clé, on va choisir pour clé la concaténation entre deux utilisateurs.
- Par exemple, la clé $A-B$ désignera *les amis en communs des utilisateurs A et B*.
- On peut segmenter les données d'entrée là aussi par ligne.

Map-Reduce: Exemple 3

-Graphe Social-

❑La phase MAP

- Notre opération Map va se contenter de prendre la liste des amis fournie en entrée, et va générer toutes les clés distinctes possibles à partir de cette liste.
- La valeur sera simplement la liste d'amis, telle quelle.
- On fait également en sorte que la clé soit toujours triée par ordre alphabétique (clé $B - A$ sera exprimée sous la forme $A - B$).
- Ce traitement peut paraître contre-intuitif, mais il va à terme nous permettre d'obtenir, pour chaque clé distincte, deux couples (key, value) : les deux listes d'amis de chacun des utilisateurs qui composent la clé.

Map-Reduce: Exemple 3

-Graphe Social-

La phase MAP

Le pseudo code de notre opération Map est le suivant :

```
UTILISATEUR = [PREMIERE PARTIE DE LA LIGNE]
POUR AMI dans [RESTE DE LA LIGNE], FAIRE:
  SI UTILISATEUR < AMI:
    CLEF = UTILISATEUR+    +AMI
  SINON:
    CLEF = AMI+    +UTILISATEUR
  GENERER COUPLE (CLEF; [RESTE DE LA LIGNE])
```

Par exemple, pour la première ligne :

```
A => B, C, D
```

On obtiendra les couples (key, value) :

```
("A-B"; "B C D")   ("A-C"; "B C D")   ("A-D"; "B C D")
```

Map-Reduce: Exemple 3

-Graphe Social-

La phase MAP

Pour la seconde ligne :

```
B => A, C, D, E
```

On obtiendra ainsi :

```
("A-B"; "A C D E")  ("B-C"; "A C D E")  ("B-D"; "A C D E")
("B-E"; "A C D E")
```

Pour la troisième ligne :

```
C => A, B, D, E
```

On aura :

```
("A-C"; "A B D E")  ("B-C"; "A B D E")  ("C-D"; "A B D E")
("C-E"; "A B D E")
```

...et ainsi de suite pour nos 5 lignes d'entrée

Map-Reduce: Exemple 3

-Graphe Social-

Entre la phase MAP et la phase REDUCE

Une fois l'opération Map effectuée, Hadoop va récupérer les couples (key, valeur) de tous les fragments et les grouper par clé distincte. Le résultat sur la base de nos données d'entrée :

```
Pour la clef "A-B": valeurs "A C D E" et "B C D"
Pour la clef "A-C": valeurs "A B D E" et "B C D"
Pour la clef "A-D": valeurs "A B C E" et "B C D"
Pour la clef "B-C": valeurs "A B D E" et "A C D E"
Pour la clef "B-D": valeurs "A B C E" et "A C D E"
Pour la clef "B-E": valeurs "A C D E" et "B C D"
Pour la clef "C-D": valeurs "A B C E" et "A B D E"
Pour la clef "C-E": valeurs "A B D E" et "B C D"
Pour la clef "D-E": valeurs "A B C E" et "B C D"
```

...on obtient bien, pour chaque clé *USER*1 – *USER*2, deux listes d'amis : les amis de *USER*1 et ceux de *USER*2.

Map-Reduce: Exemple 3

-Graphe Social-

La phase REDUCE

Il nous faut enfin écrire notre programme Reduce. Il va recevoir en entrée toutes les valeurs associées à une clé. Son rôle va être très simple : déterminer quels sont les amis qui apparaissent dans les listes (les valeurs) qui nous sont fournies.

```
LISTE_AMIS_COMMUNS=[]    // Liste vide au départ.
SI LONGUEUR(VALEURS)!=2, ALORS:  // Ne devrait pas se produire.
  RENVOYER ERREUR
SINON:
  POUR AMI DANS VALEURS[0], FAIRE:
    SI AMI EGALEMENT PRESENT DANS VALEURS[1], ALORS:
      // Présent dans les deux listes d'amis, on l'ajoute.
      LISTE_AMIS_COMMUNS+=AMI
RENVOYER LISTE_AMIS_COMMUNS
```

Map-Reduce: Exemple 3

-Graphe Social-

La phase REDUCE

Après exécution de l'opération Reduce pour les valeurs de chaque clé unique, on obtiendra donc, pour une clé $A - B$, les utilisateurs qui apparaissent dans la liste des amis de A et dans la liste des amis de B. Autrement dit, on obtiendra la liste des amis en commun des utilisateurs A et B. Le résultat est :

```
"A-B": "C, D"
"A-C": "B, D"
"A-D": "B, C"
"B-C": "A, D, E"
"B-D": "A, C, E"
"B-E": "C, D"
"C-D": "A, B, E"
"C-E": "B, D"
"D-E": "B, C"
```

On sait ainsi que A et B ont pour amis communs les utilisateurs C et D, ou encore que B et C ont pour amis communs les utilisateurs A, D et E.

Map-Reduce: Exemple 3

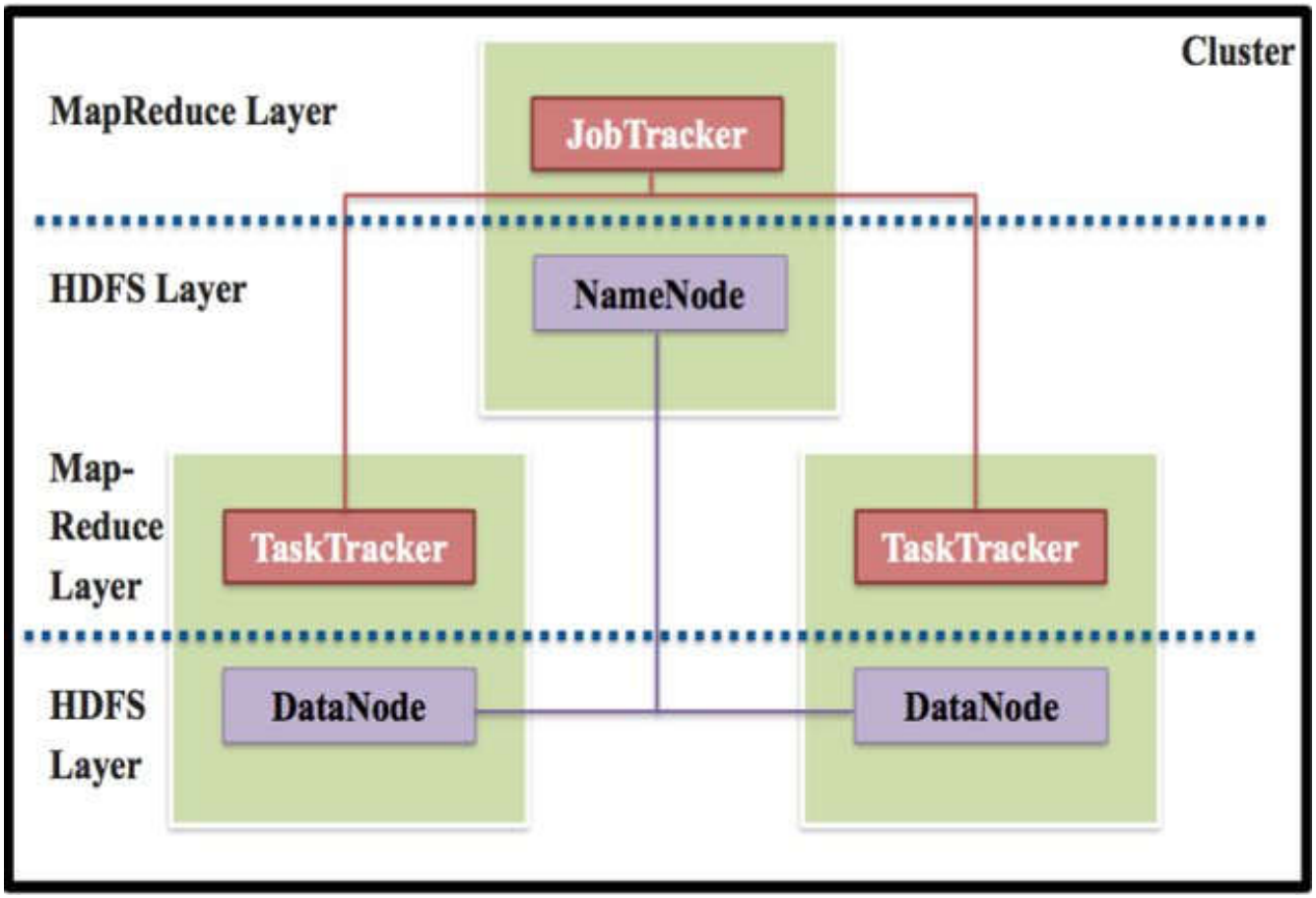
-Graphe Social-

Synthèse

- En utilisant le modèle MapReduce, on a ainsi pu créer deux programmes très simples (nos programmes Map et Reduce) de quelques lignes de code seulement, qui permettent d'effectuer un traitement somme toute assez complexe.
- Mieux encore, notre traitement est parallélisable : même avec des dizaines de millions d'utilisateurs, du moment qu'on a assez de machines au sein du cluster Hadoop, le traitement sera effectué rapidement. Pour aller plus vite, il nous suffit de rajouter plus de machines.
- Pour notre réseau social, il suffira d'effectuer ce traitement toutes les nuits à heure fixe, et de stocker les résultats dans une table.
  Ainsi, lorsqu'un utilisateur visitera la page d'un autre utilisateur, un seul SELECT dans la BD suffira pour obtenir la liste des amis en commun - avec un poids en traitement très faible pour le serveur.

Fonctionnement de Map Reduce

Hadoop MapReduce: Fonctionnement



Hadoop MapReduce: Fonctionnement

- Deux démons: Job Tracker et Task Tacker
- Job Tracker
  - S'exécute sur la même machine que le Namenode (machine master)
  - Divise le travail sur les Mappers et Reducers s'exécutant sur les différents nœuds.
- Task Tacker
  - S'exécute sur chacun des nœuds pour exécuter les vraies tâches de Map-Reduce.
  - Choisit de traiter (map et reduce) un bloc sur la même machine que lui affecté
  - S'il est déjà occupé, la tâche revient à un autre tracker qui utilisera le réseau (rare).

Hadoop MapReduce: Fonctionnement

- Un job Map-Reduce est divisé en plusieurs tâches **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 ordonnancer les slots, et de gérer les fautes en réallouant les slots au besoin.

Hadoop MapReduce: Fonctionnement