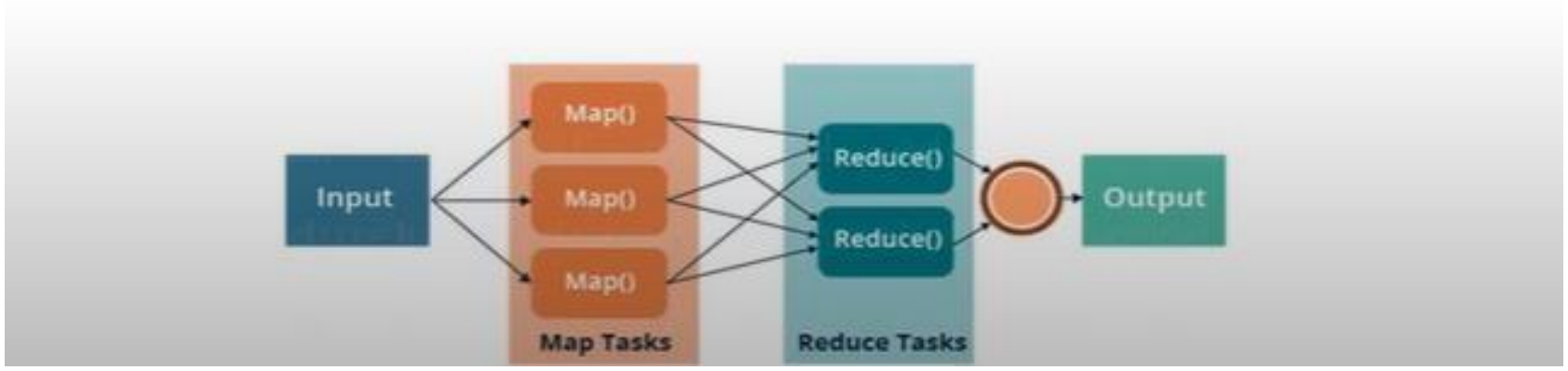
Big data et Docker

Framework map/reduce

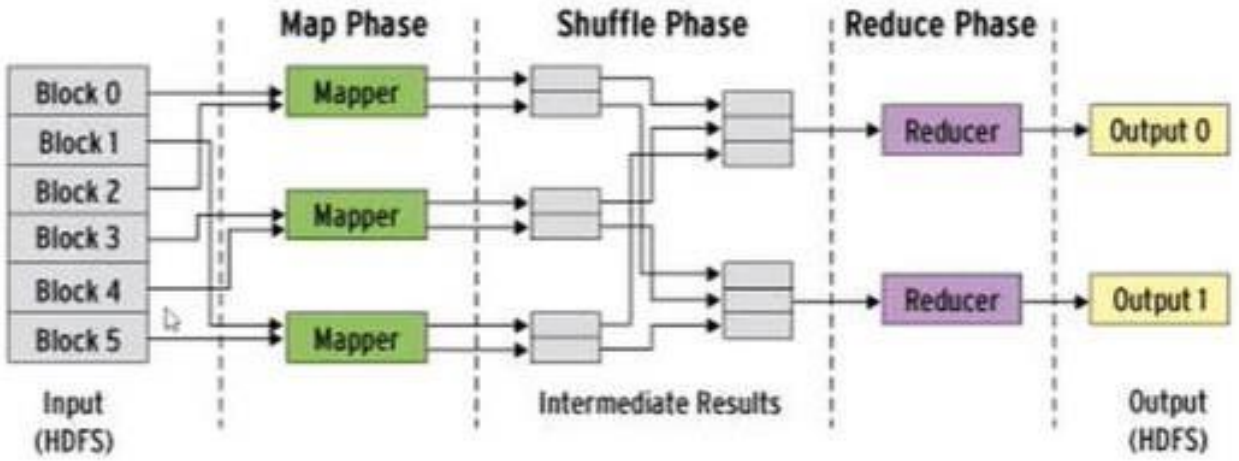
- **FRAMEWORK MAPREDUCE**
  - MapReduce est conçu pour traiter une grande quantité de données structurées et non structurées stockées dans le HDFS en utilisant un grand nombre de nœuds.
  - MapReduce exécute les tâhes en parallèle.
  - Le Framework se décompose en deux phases:
    - La phase **Map** permet aux différents nœuds du cluster de distribuer leur travail.
    - La phase **Reduce** permet de réduire la forme finale des résultats des clusters en un seul résultat.
  - Chaque phase génère des paires « clé/valeur » en entrée et en sortie.



Big data et Docker

Framework map/reduce

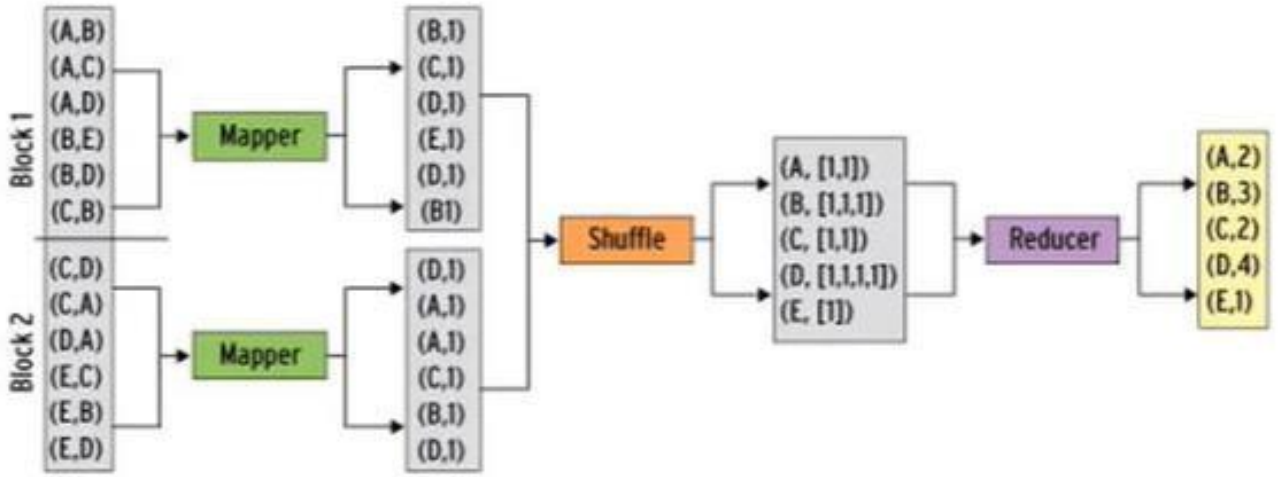
- **FRAMEWORK MAPREDUCE**
  - La première phase est le mappage, dans lequel un bloc de données est lu et traité pour produire des paires « clé-valeur » en tant que **sorties intermédiaires.**
  - Les sorties intermédiaires du mappage sont les entrées de la phase Reduce.
  - Le phase « Reduce » reçoit la paire « clé-valeur » de plusieurs tâches du Map.
  - Ensuite, le réducteur agrège ces données intermédiaires en un ensemble plus petit de tuples ou de paires clé-valeur qui constitue la sortie finale.



Big data et Docker

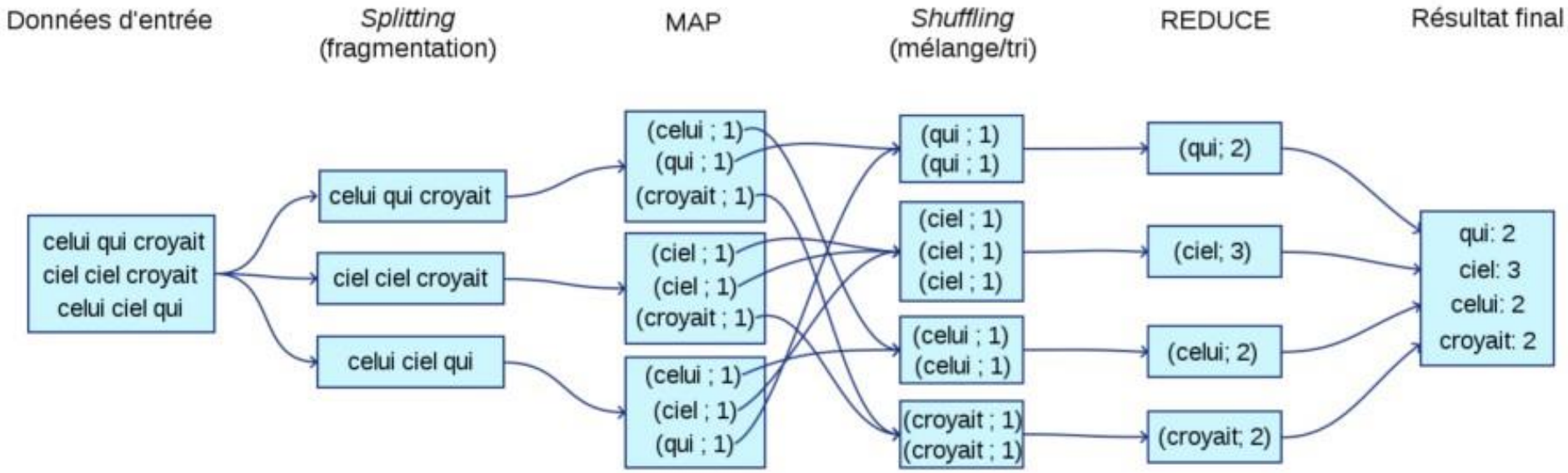
Framework map/reduce

- **FRAMEWORK MAPREDUCE**
  - La première phase est le mappage, dans lequel un bloc de données est lu et traité pour produire des paires « clé-valeur » en tant que **sorties intermédiaires**.
  - Les sorties intermédiaires du mappage sont les entrées de la phase Reduce.
  - Le phase « Reduce » reçoit la paire « clé-valeur » de plusieurs tâches du Map.
  - Ensuite, le réducteur agrège ces données intermédiaires en un ensemble plus petit de tuples ou de paires clé-valeur qui constitue la sortie finale.



Fréquence d'apparition d'un mot

Les 4 étapes de l'algorithme MR sont illustrées à l'aide du schéma suivant extrait des slides cités plus haut :



Les 4 étapes algorithmiques sont :

— split : est une fonction qui divise l'ensemble des données en blocs de données,

— map : est une fonction classique en programmation fonctionnelle qui applique une fonction sur chaque bloc de données. Le résultat de la fonction appliquée à chaque bloc est un ensemble de couples (clé, valeur),

— shuffle : est l'opération qui consiste à regrouper les couples (clé ; valeur) qui ont des clés identiques,

— reduce : est la fonction classique de programmation fonctionnelle (aussi appelée fold) qui réduit un ensemble de données en une donnée finale.

Big data et Docker

Framework map/reduce

L'étape principale d'interaction entre les données, shuffle, est toujours la même. Elle est donc définie dans l'algorithme MR. Par contre, pour utiliser un algorithme MR, l'utilisateur doit fournir les éléments et fonctions suivantes :

- le couple (clé, valeur)
- la fonction split : donnée $\rightarrow$ [donnée, . . . , donnée],
- la fonction map : donnée $\rightarrow$ [{clé, valeur}, . . . , {clé, valeur}],
- la fonction reduce : [valeur, . . . , valeur] $\rightarrow$ valeur.

Exemple 1 : Multiplication d'une matrice par un vecteur

Le premier problème auquel nous allons nous intéresser est celui qui consister à multiplier une matrice `A` de grande taille (n×n) par un vecteur `v` de taille `n` . Il s'agit donc de calculer

$$Av = x$$

avec

$$x = (x_1, \ldots, x_n)$$

et

$$x_i = \sum_{j=1}^{n} a_{ij} v_j$$

c'est un joli problème mathématique mais bien loin de nos préoccupations !
En grande partie pour ce problème que MapReduce a été conçu chez Google car c'est une opération nécessaire au calcul du fameux PageRank, utilisé pour ordonnancer les résultats d'une recherche Web.
Dans ce cas, n est le nombre de pages web indexées...

Un vrai problème big data ! De plus, c'est une opération très commune, que l'on retrouve dans de nombreux problème et notamment dans les algorithmes du data scientist.

Exemple 1 : Multiplication d'une matrice par un vecteur

Pour ce problème, la vraie question est la manière avec laquelle représenter la matrice A et donc la forme de l'entrée donnée à MapReduce?

Très souvent, pour ce type de problèmes, nous sommes en présence de matrices creuses et on évite donc de représenter les zéros.

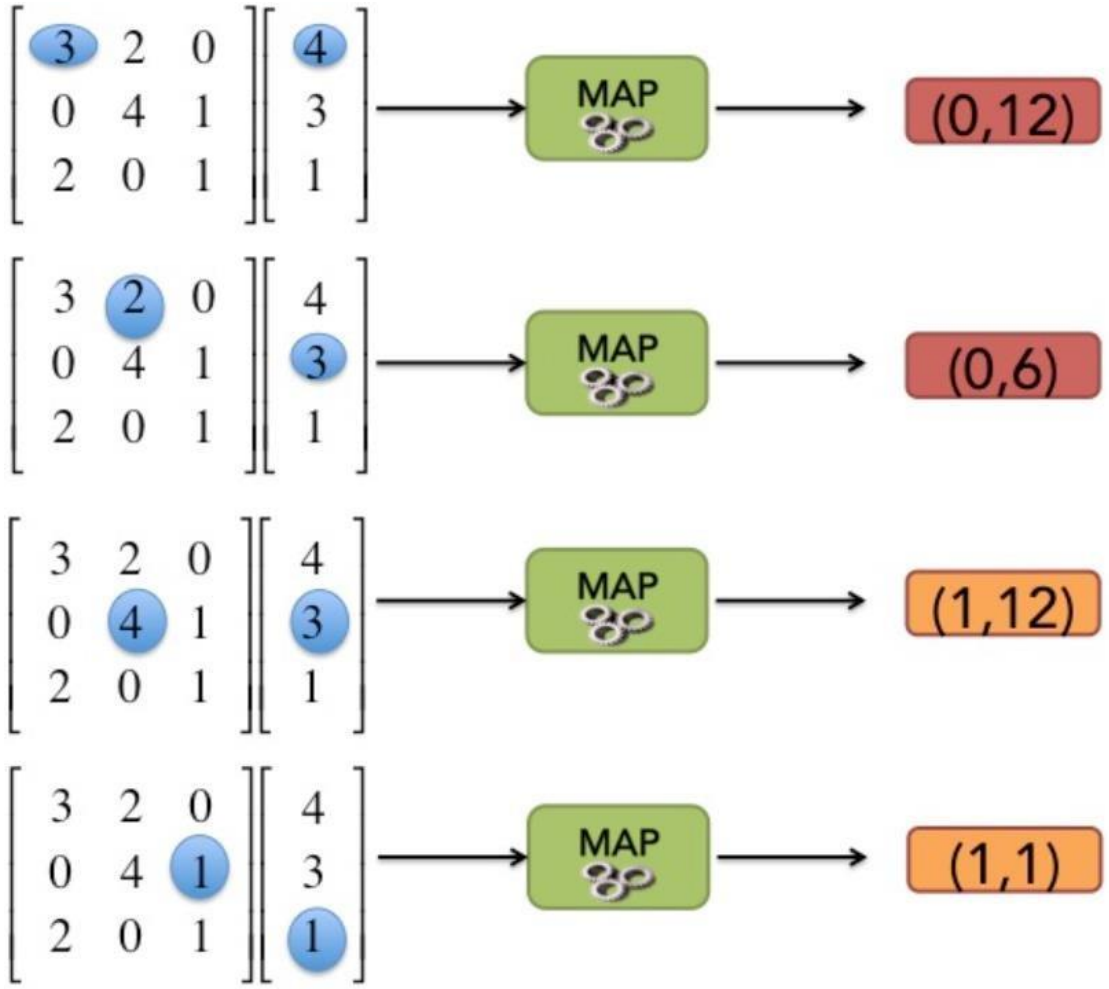
Ici, nous allons donc considérer que la matrice A est stockée sous la forme de triplets (i,j,aij) (les coordonnées sont explicites).

De même, le vecteur v est stocké sous la forme de paires (j,vj) .

Big data et Docker

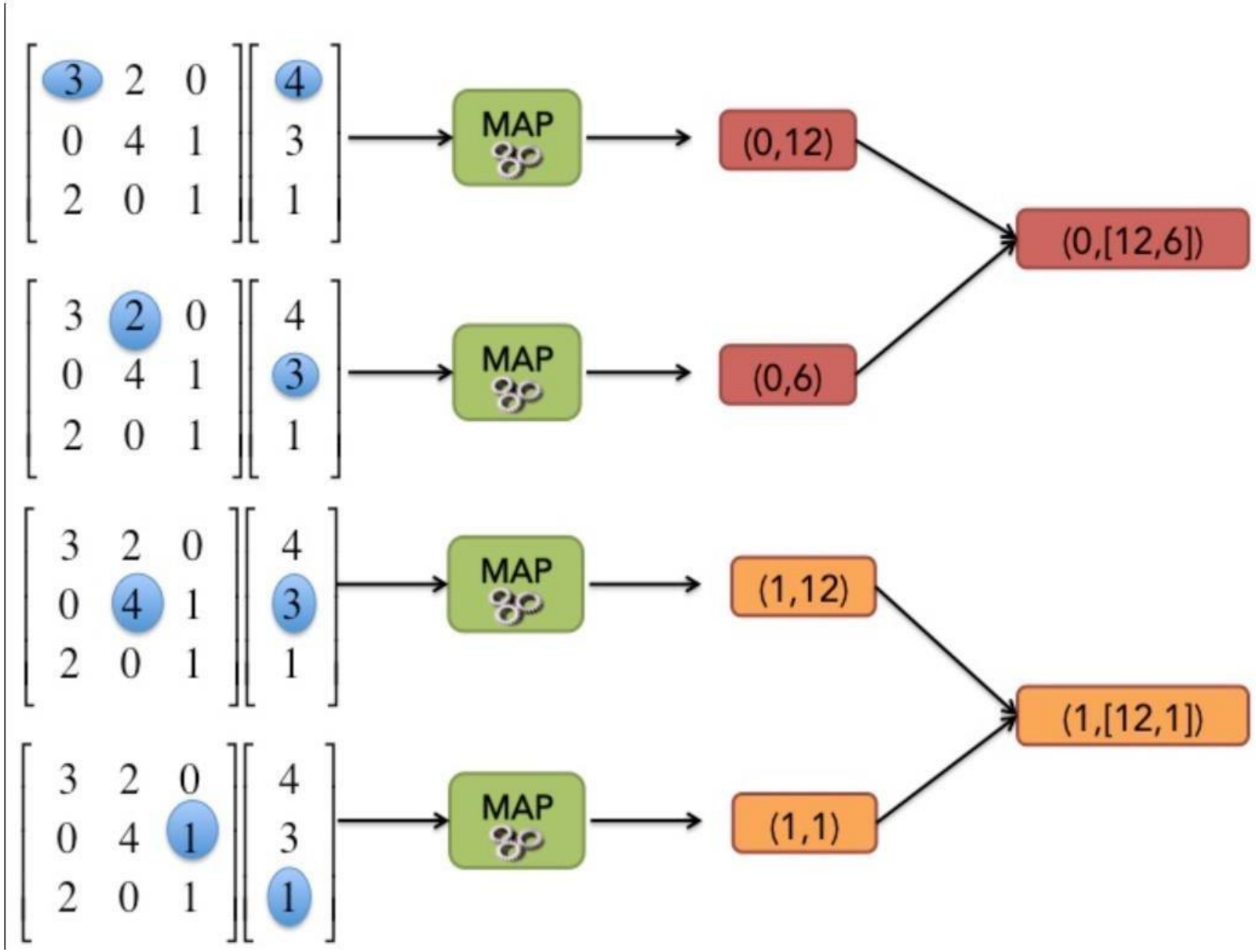
Framework map/reduce

Exemple 1 : Multiplication d'une matrice par un vecteur

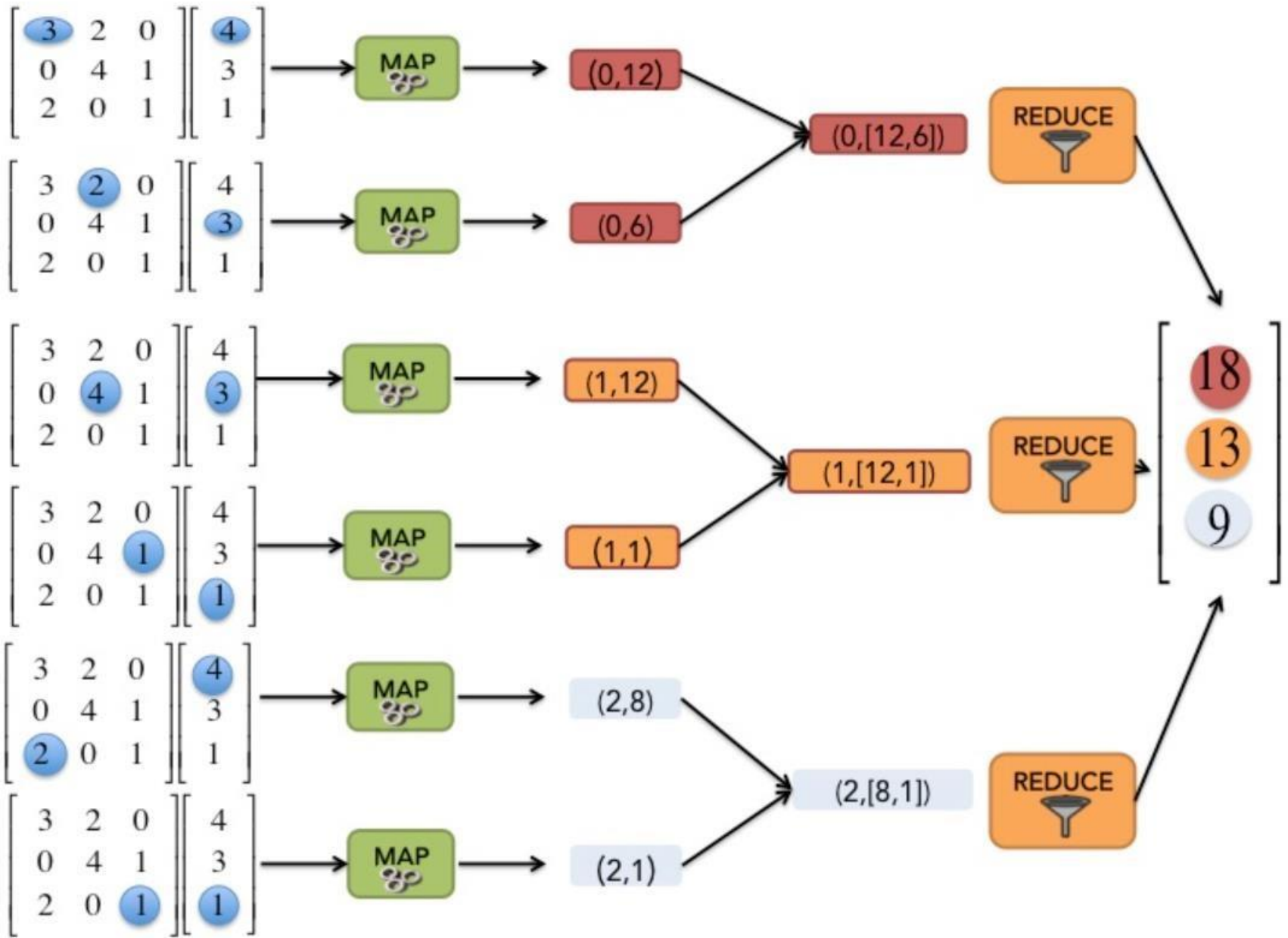


pour chaque élément de la matrice, l'opération MAP va juste générer la paire (i,aijvj)

SHUFFLE and SORT regroupe toutes les valeurs associées à la même clé i dans une paire $(i, [a_{i1}v_1, \ldots, a_{in}v_n])$.



L'opération REDUCE est donc aussi très évidente, il suffit de faire la somme de toutes les valeurs associées à une clé donnée.



Exemple 2 : Jointure de deux tables de données

```
1 SELECT *
2 FROM Films F JOIN Realisateurs R
3 ON F.ID_realisateur=R.ID_realisateur
```

Entreprise de vente de films en ligne avec un gigantesque catalogue.

une stratégie qui s'appelle *Reduce-Side Join*, c'est-à-dire que l'opération de jointure en tant que telle sera effectuée dans la phase REDUCE.

ID_film	Titre	ID_realisateur	ID_acteur	...
1111	Pulp Fiction	123	23	
1112	Le pianiste	4567	678	
1113	La leçon de piano	234	567	
...	...	...	...	

Films

jointure

ID_réalisateur	Nom
123	Quentin Tarentino
4567	Roman Polanski
234	Jane Campion
...	...

Réalisateurs

Big data et Docker

Framework map/reduce

Exemple 2 : Jointure de deux tables de données

ID_realisateur	Titre
123	Pulp Fiction
4567	Le pianiste
234	La leçon de piano
123	Reservoir dogs
..	...

ID_réalisateur	Nom
123	Quentin Tarentino
4567	Roman Polanski
234	Jane Campion
...	...

Regrouper les enregistrements des deux tables en une seule longue liste d'enregistrements en ajoutant à chaque enregistrement le nom de la table dont il est issu.

Films

Réalisateurs

Films	123	Pulp Fiction
Films	4567	Le pianiste
Films	234	La leçon de piano
Films	123	Reservoir dogs
Réalisateurs	123	Q. Tarentino
Réalisateurs	4567	R. Polanski
Réalisateurs	234	J. Campion
...	...	...

Big data et Docker

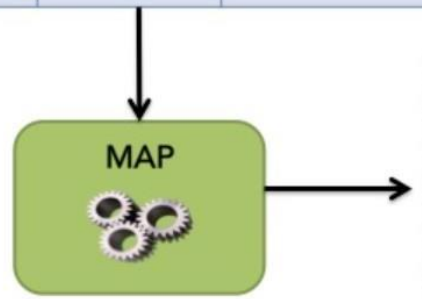
Framework map/reduce

Exemple 2 : Jointure de deux tables de données

Il suffit de renvoyer pour chaque enregistrement la paire(clé, valeur)où la clé est la clé de jointure (ID_realisateur) et la valeur est le contenu de l'enregistrement.

```python
def map(key,value):
    intermediate = []
    for i in value:
        intermediate.append((i[1], (i[0], i[1:])))
    return intermediate
```

Films	123	Pulp Fiction
Films	4567	Le pianiste
Films	234	La leçon de piano
Films	123	Reservoir dogs
Réalisateurs	123	Q. Tarentino
Réalisateurs	4567	R. Polanski
Réalisateurs	234	J. Campion
...	...	...



(123, (Films, Pulp Fiction))
(4567, (Films,Le pianiste))
(234, (Films, La leçon de piano))
(123, (Films, Reservoir dogs))
(123, (Réalisateurs, Q. Tarentino))
(4567, (Réalisateurs, R. Polanski))
(234, (Réalisateurs, J. Campion))

Big data et Docker

Framework map/reduce

Exemple 2 : Jointure de deux tables de données

(123, (Films, Pulp Fiction))
(123, (Films, Reservoir dogs))

→ (123, [(Films, Pulp Fiction), (Films, Reservoir dogs), (Réalisateurs, Q. Tarentino)]

SHUFFLE and SORT retourne les résultats intermédiaires qui sont alors regroupés par clé de jointure commune et chaque paire regroupée constitue donc une entrée idéale à une opération REDUCE.

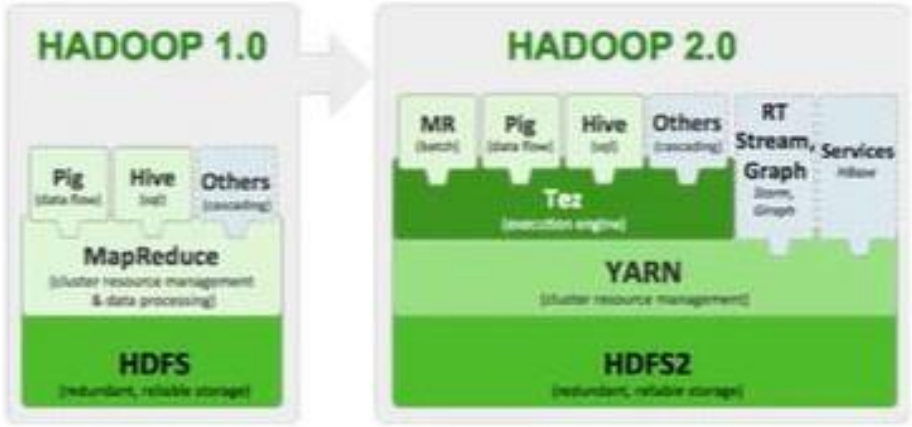
L'opération REDUCE est aussi facile à concevoir. Elle concatène les enregistrements des tables Films et Réalisateurs associées à une même clé de jointure. Et au final, nous avons donc le schéma d'exécution suivant de MapReduce pour notre problème de jointure:

ID_réalisateur	Nom	Titre Films
123	Quentin Tarentino	Pulp Fiction
123	Quentin Tarentino	Reservoir dogs

l'écosystème Hadoop

Framework yarn

- **YARN** (Yet Another Resource Negotiator)
  - Planificateur et gestionnaire des ressources du cluster.
- Il y a deux idées principales avec YARN.
  - Pour éviter que chaque nouveau Framework ne dispose pas de son propre gestionnaire de ressources et de son propre planificateur, et qu'il soit en concurrence avec les autres gestionnaires de ressources et planificateurs.
  - Fournir une planification générique et une gestion des ressources. De cette façon, Hadoop peut supporter plus que **MapReduce**.

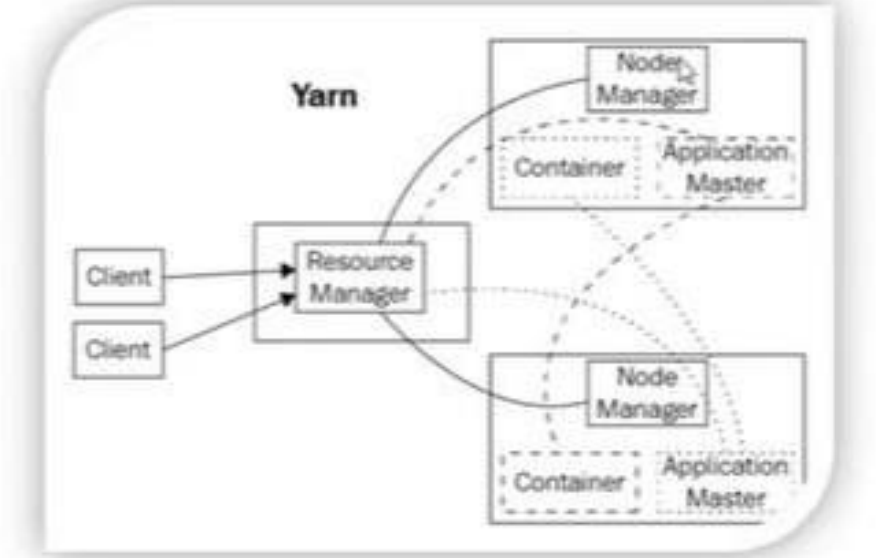


- Mémoire libre
- Nombre de cœurs …..

ARCHITECTURE YARN

- YARN comporte deux composants : « **RESOURCE MANAGER** » et « **NODE MANAGER** »
- Le « **RESOURCE MANAGER** » gestionnaire de ressources connait les capacités de chaque nœud via la communication avec « **NODE MANAGER** » s'exécutant sur chaque nœud.
- Lorsqu'une application souhaite s'exécuter, le client lance « **APPLICATION MASTER** », qui négocie ensuite avec « **RESOURCE MANAGER** » pour obtenir des ressources dans le cluster sous la forme de conteneurs « **CONTAINER** ». Ces ressources sont affectées à des conteneurs sur chaque nœud.
- Les tâches sont ensuite exécutées dans des conteneurs.

Conteneur= Cœurs + processeurs + RAM …..



CONTENEUR

- Un conteneur représente les CPU (cœurs) et la mémoire allouée sur un seul nœud à utiliser pour exécuter des tâches et des processus. Les conteneurs sont supervisés par NodeManager et planifiés par ResourceManager.
- Exemples de conteneurs:
  - Un cœur et 4 Go de RAM
  - Deux coeurs et 6 Go de RAM
  - Quatre coeurs et 20 Go de RAM
- Certains conteneurs sont affectés à « **map** » et d'autres à « **Reduce** », tout cela est coordonné par ApplicationMaster en collaboration avec ResourceManager.