# HADOOP

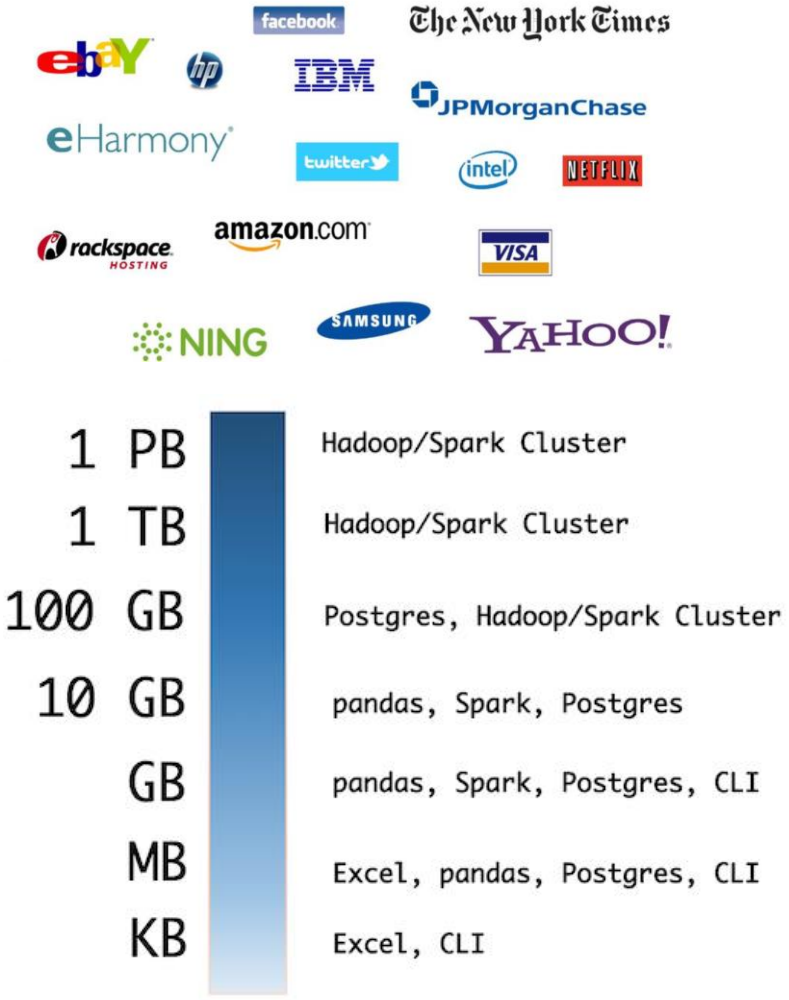

M. Ben & R. Tavenard

# L’écosystème big data Hadoop

- Framework permettant la création d’applications big data distribuées
  - 3 V du big data
    - **V**olume de données très important
    - **V**itesse de lecture et d’exécution
    - **V**ariété des données

- En particulier, Hadoop fournit :
  - HDFS (*Hadoop Distributed File System*)
  - YARN (*Yet Another Ressource Negotiator*)
  - Map/Reduce distribué

- Outils écrits en Java



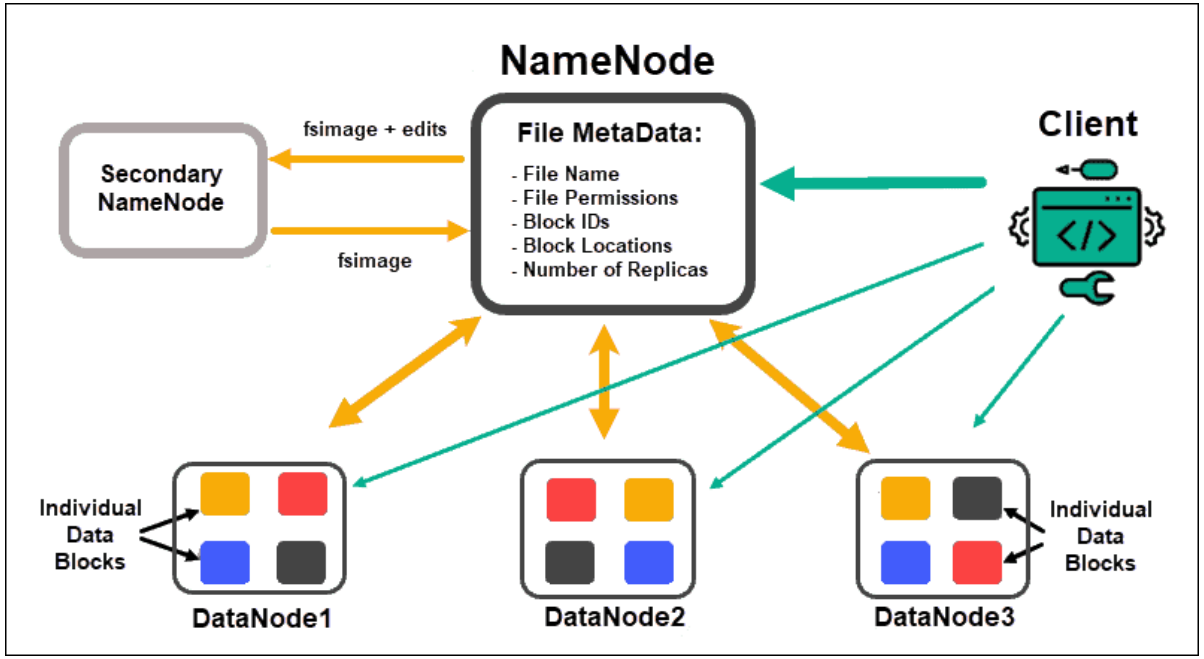
# HDFS

- Système de fichiers distribué
  - Facilement extensible
    - Cluster de données basé sur des machines bon marché
  - Tolérant aux défaillances matérielles
    - Procédé de réplication et de réallocation des données
  - Efficace pour les calculs sur gros volumes de données
    - Déplace les programmes au niveau des données plutôt que de rassembler les données au niveau des programmes

      "*Moving Computation is Cheaper than Moving Data*"
    - Gain de latence et de bande passante

# HDFS

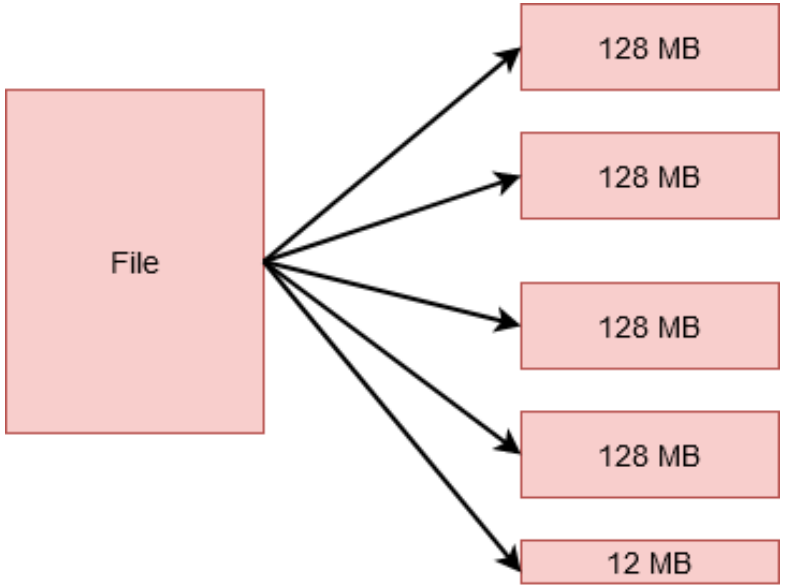
- Architecture
  - NameNode
    - Serveur principal chargé de gérer le HDFS et les échanges de données avec les « clients » (i.e. utilisateurs)
    - Contient les métadonnées du HDFS
  - DataNodes
    - Serveurs de données du cluster HDFS
  - Secondary NameNode
    - Assiste le NameNode principal pour générer les nouvelles images du système de fichiers




*source : https://phoenixnap.com*

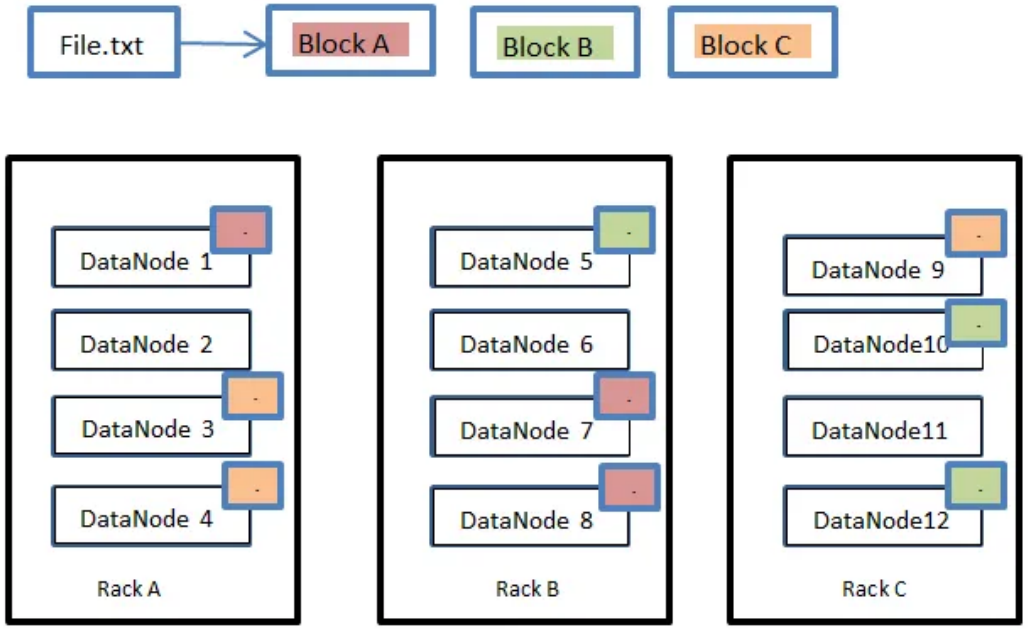
# HDFS

- Découpage des fichiers en blocs
  - Blocs de grande taille : 128 Mo par défaut
    - Par comparaison : qq Ko dans Windows
  - Permet de limiter la quantité de métadonnées
    - La quantité de données gérée peut être gigantesque (plusieurs pétaoctets)
    - Le NameNode doit répertorier l'emplacement des blocs de chaque fichier et de leurs répliques



# HDFS

- Réplication des blocs
  - Facteur de réplication
    - 3 par défaut
  - Algorithme de réplication
    - Basé sur la distribution des DataNodes au sein des racks du cluster
    - rack = ensemble de machines
      - proches physiquement
      - connectées au même switch réseau
  - Gestion par le NameNode
    - Choix des DataNodes de réplication
    - Equilibrage du cluster
    - Accès en lecture : détermination de la réplique du bloc la plus proche du client




*source : https://medium.com/@sharayushinde158*

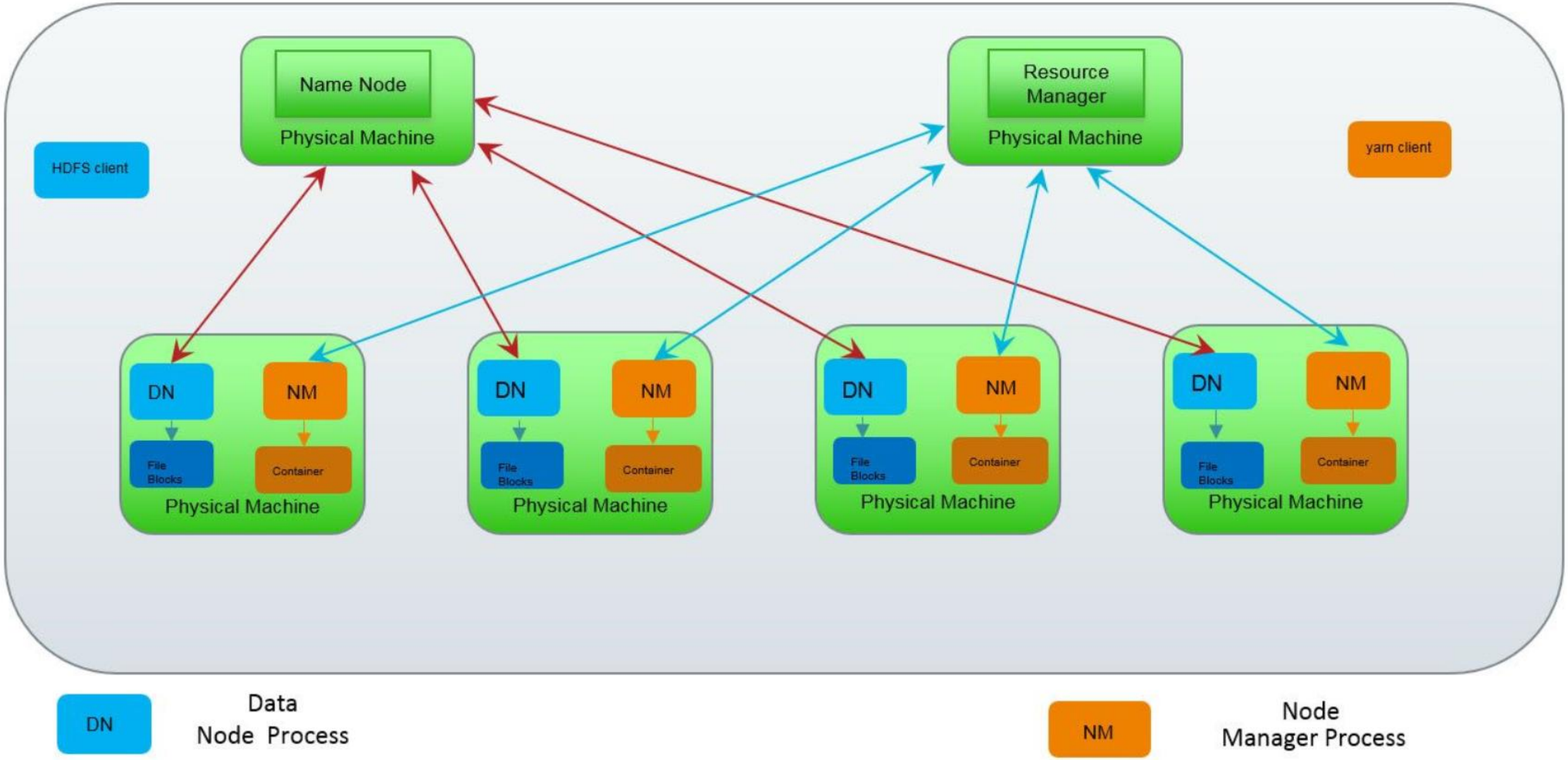
# HDFS : commandes shell

- Permettent d'interagir avec le système de fichiers distribué
- Rôles similaires aux commandes shell Linux
- Forme générale :
  - `hadoop fs -`*`commandName`* `<args>`
  - Les arguments sont des URI sur le système local ou distribué :
    - Exemple sur le système local :
      `file://local_dir/local_file`
    - Exemple sur le système distribué :
      `hdfs://namenodehost/hdfs_dir/hdfs_file`
    - Possibilité d'utiliser des URI relatifs
- Mise en œuvre : voir la documentation

# YARN

- *Yet Another Ressource Negociator*
- Système de gestion d'applications distribuées (ex : Spark)
- Gère et distribue les ressources de calcul du cluster
  - CPUs
  - Mémoire
- Basé sur une architecture maitre/esclave:
  - Un RessourceManager sur une machine dédiée
    - Gère les ressources du cluster globalement et supervise les NodeManager
  - Un NodeManager sur chaque nœud
    - Supervisé par le RessourceManager, gère les ressources du nœud

# Architecture HDFS+YARN



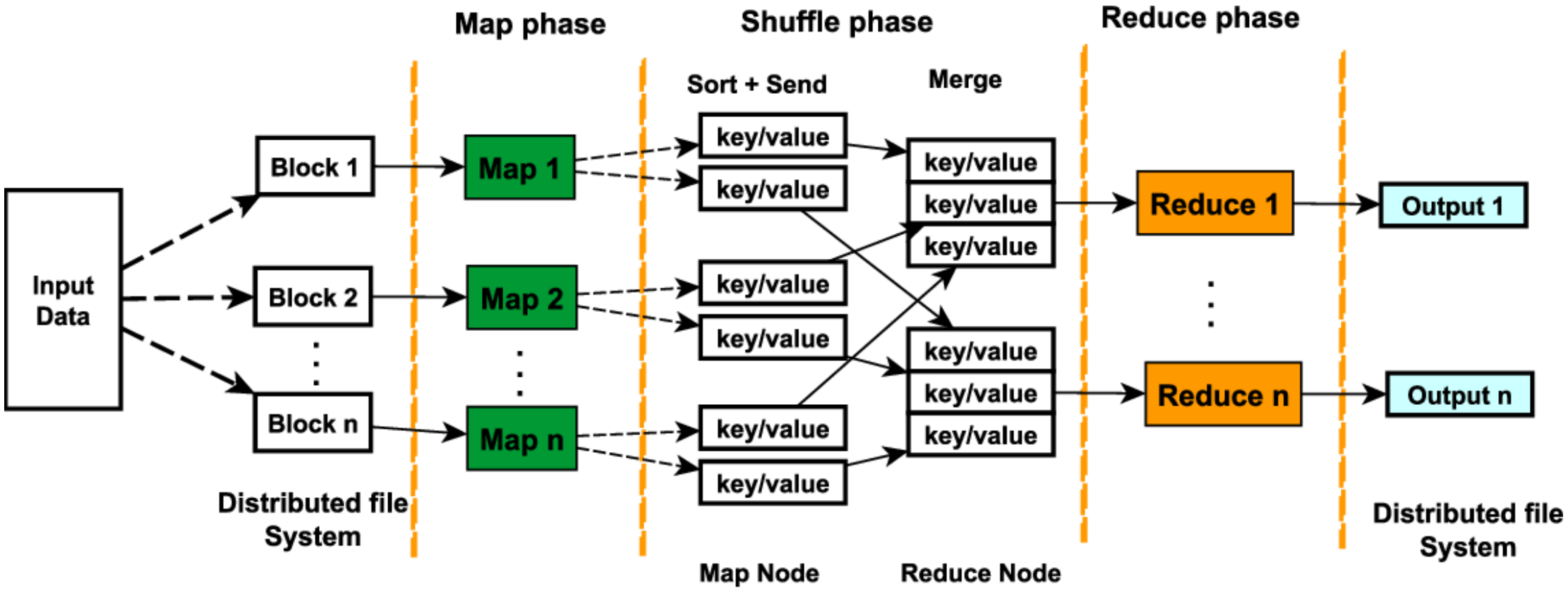

*source : https://community.cloudera.com*

# Hadoop MapReduce

- Implémentation distribuée du modèle MapReduce
  - Les entrées/sorties et les échanges de données reposent sur des fichiers du HDFS

- Étapes du traitement :
  1. Découpage des données de travail
     → Correspond au découpage naturel en bloc de HDFS
  2. Calculs parallèles indépendants : processus Map
     → Chaque nœud de calcul Map (*MapNode*) traite lui-même les données qu'il héberge
        → Limite les transferts de données et accélère le processus
  3. Tri et regroupements intermédiaires par clé
     → Les résultats intermédiaires associés à une même clé sont transférés sur un même nœud de calcul
  4. Agrégation des résultats par clé : processus Reduce
     → Un nœud de calcul Reduce (*ReduceNode*) est chargé d'agréger les valeurs intermédiaires d'une ou plusieurs clés intermédiaires
     → Les résultats finaux sont écrits dans des fichiers du HDFS (`part-00000`, `part-00001`, …)

# Hadoop MapReduce : flux de traitement




*source : https://www.codingninjas.com*

# Hadoop MapReduce : mise en oeuvre

- En Java (langage natif)
  - Écrire les `mapper` et `reducer` dans des classes Java
  - Compiler les classes Java
  - Exécuter le programme Java compilé
    - `hadoop jar mon_programme_java.jar …`

- En Python :
  - 1ère solution :
    - écrire les `mapper` et `reducer` en Python et les compiler en Java à l'aide de l'outil `Jython` (support limité des fonctionnalités de Python)
  - 2ème solution :
    - Utiliser l'outil Hadoop Streaming
      - Se sert des entrée/sortie standard (`stdin`, `stdout`) pour lire/écrire les données
      - Les `mapper` et `reducer` sont des programmes (Python ou autre) qui lisent/écrivent sur les entrée/sortie standard

# MapReduce avec Hadoop Streaming

- Ligne de commande

```
mapred streaming \
  -input myInputDirs \
  -output myOutputDir \
  -mapper myMapper.py \
  -reducer myReducer.py \
  -file myMapper.py \
  -file myReducer.py \
```