Map-Reduce : un cadre de programmation parallèlle pour l'analyse de grandes données

Master ILC, Université de Strasbourg
Stéphane Genaud

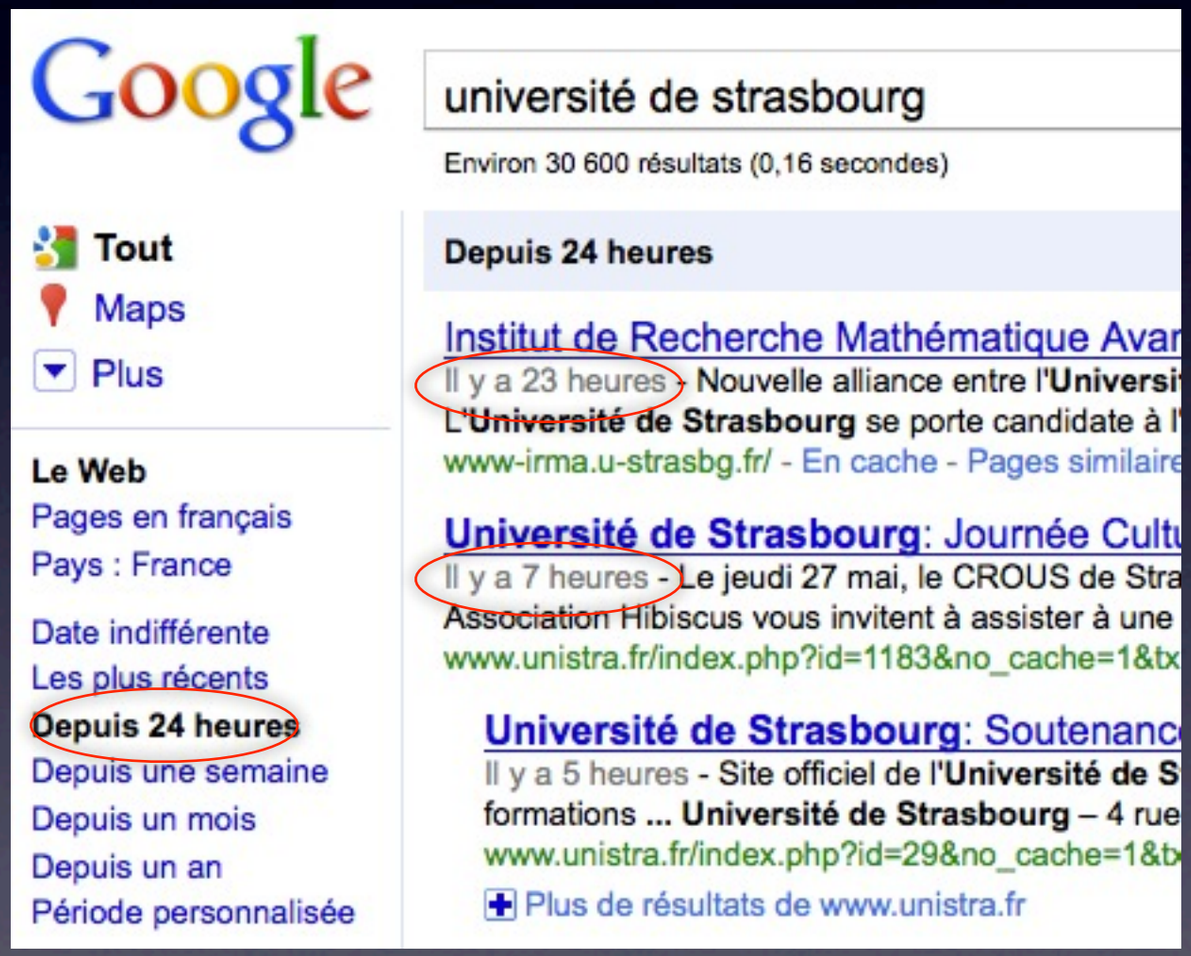
Traitement de données distribuées

- Problème particulier mais de plus en plus fréquent: traitement de fichiers textes de très grandes tailles.
- Google confrontés les premiers au problème:
  - Exemple: 20 milliards pages web x 20KB = 4E14 KB (400 TB)
  - vitesse lecture disque 30-35 MB/sec
  - => 4 mois de lecture
  - Nécessite de l'ordre de 1000 disques
  - Traitement de ces données encore plus long et fastidieux

Traitement de données distribuées

- Google a introduit Map-Reduce [Dean and Ghemawat 2004]
- Ils s'en servent de manière intensive:
  - utilisation en sept 2007: 400+ TB de données traitées, temps et nombre moyen de machine par traitement Map-Reduce : 6 min et 436
  - indexation web traite 20+ TB de données brutes
  - analyse d'images satellites
  - calculs statistiques pour Google translate
  - clustering des Google news
  - ....

Traitement de données



27 mai 2010, 14:00

- analyser les données toujours plus vite (avant 2007, 2 jours d'ancienneté)
- ... alors que le volume des données augmente toujours

Traitement de données distribuées

- Autres applications possibles:
  - caractériser les activités des utilisateurs Facebook
  - analyser les fichiers de logs d'un serveur web
  - fouille de données sur logs des caisses enregistreuses super-marché
  - recherche de séquences d'ADN dans un génôme
  - le New York Times a utilisé 100 machines sur Amazon EC2 avec Hadoop pour traiter 4TB d'images TIFF (stockées sur S3) pour produire 11 millions de PDFs (durée 24h, coût 240 $)

Map-Reduce

- Un modèle de programmation ...
- ... avec un schéma très contraint.
- Mais permet
  - parallélisation automatique
  - de l'équilibrage de charge
  - des optimisations sur les transferts disques et réseaux
  - de la tolérance aux pannes

Schéma général

1. Lecture des données
2. Map : pour chaque élément de données, construire un couple (clé,valeur)
3. Trier selon les clés
4. Reduce: agréger, résumer, filtrer ou transformer les données
5. Écrire les données

Exemple: word count

Problème: *parmi un ensemble de textes, compter le nombre d'occurrences de chaque mot.*

- Données: un ensemble de fichiers textes
- Map : sur chaque texte, décomposer en mots, et à chaque mot *m*, ajouter à la liste [(*m*, 1),...]
- Sort : le système trie/regroupe les paires selon la clé *m, dans une liste [(m,[1, 1, ...]),...]*
- Reduce : les valeurs 1 sont additionnées pour chaque mot : *[(m,2),...]*

Schéma général

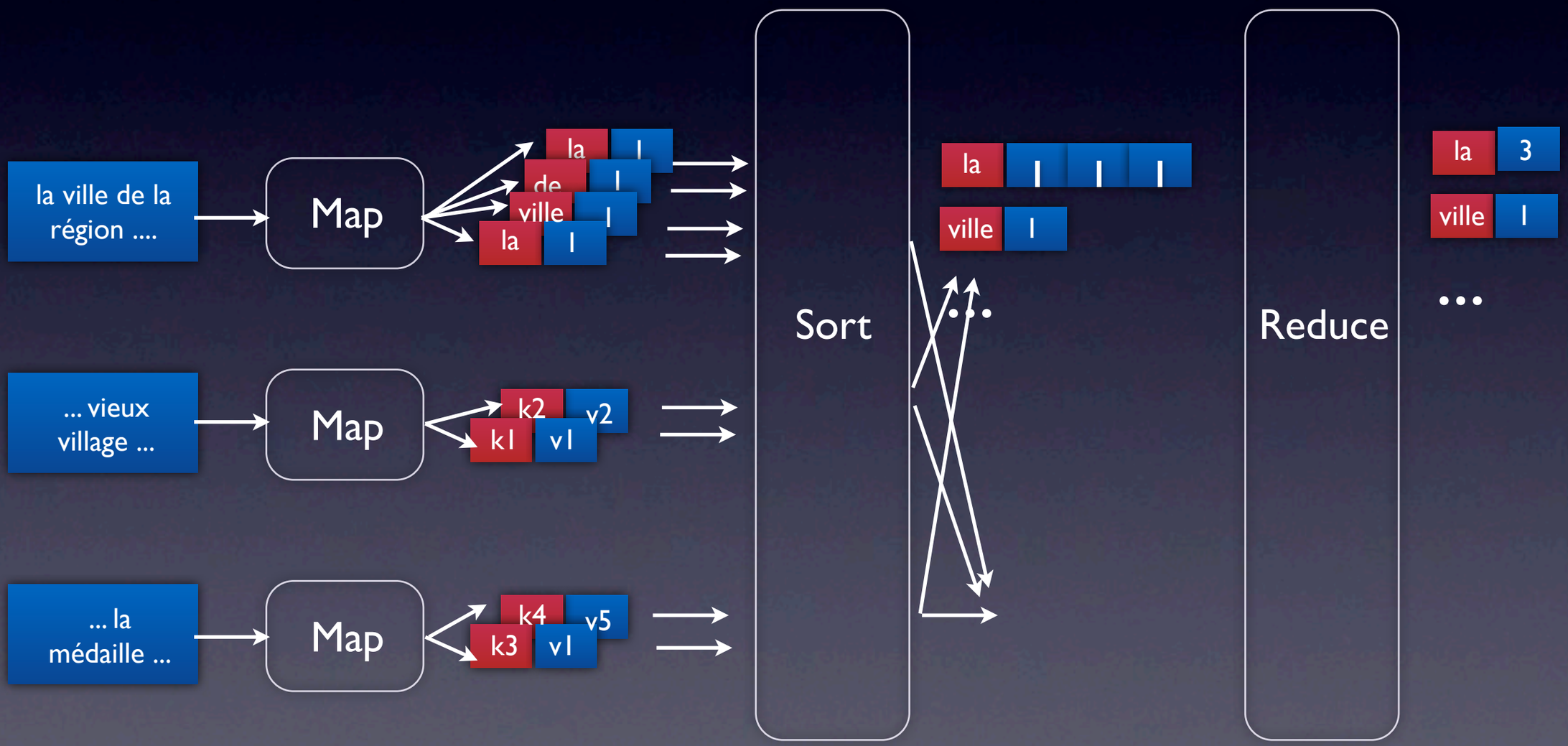
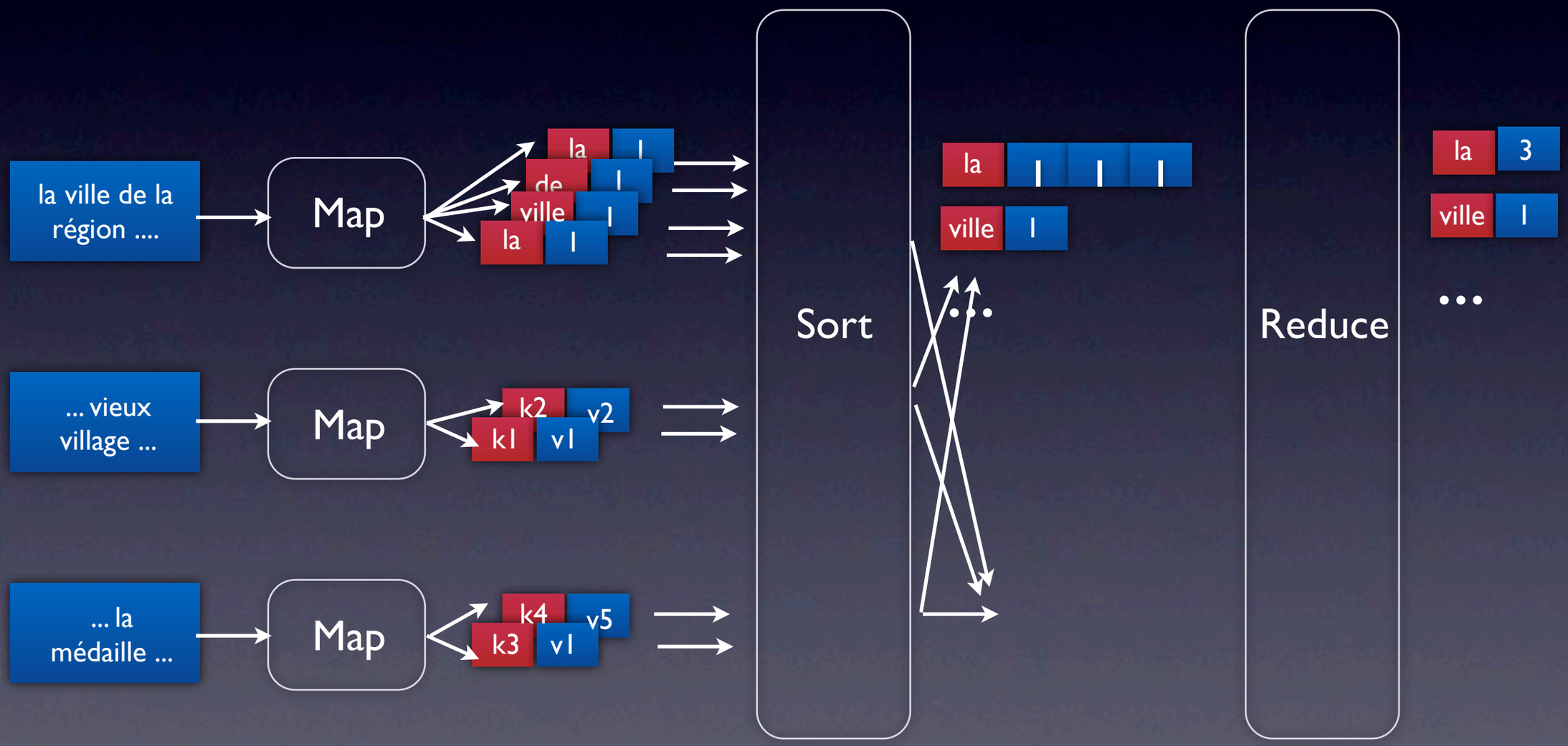


Schéma général



...devrait s'appeler Map-Sort-Reduce ...

Pseudo-code word count

```
Map(String input_key, String input_values) :
    foreach word w in input_values:
      EmitIntermediate( w, «1»);
```

```
Reduce (String key, Iterator intermediate_values):
    int result=0;
    foreach v in intermediate_values:
      result += ParseInt( v );
    Emit( key, String( result ));
```

Exemple: index renversé

Problème: *soit un ensemble de fichiers contenant des mots. Etablir une liste des mots, et pour chaque mot une liste des fichiers ou il apparait.*

```
F1 : Strasbourg Nantes Paris
F2 : Mulhouse Colmar Strasbourg
```

```
Colmar : F2
Nantes : F1
Mulhouse: F2
Paris : F1
Strasbourg : F1 F2
```

Pseudo-code
index renversé

```
Map(String filename, String words) :
    foreach word w in words:
       EmitIntermediate( w, filename );
```

```
Reduce (String key, Iterator intermediate_values):
// key=word, intermediate_values=filenames

    foreach f in intermediate_values:
       result += f + ‘   ‘;

    Emit( key, result ));
```

Implémentations Map-Reduce

- Des librairies MapReduce existent pour C++, C#, Erlang, Java, Python, Ruby, R
- La plus connue (hors celle propriétaire de Google) est Hadoop (projet Apache).

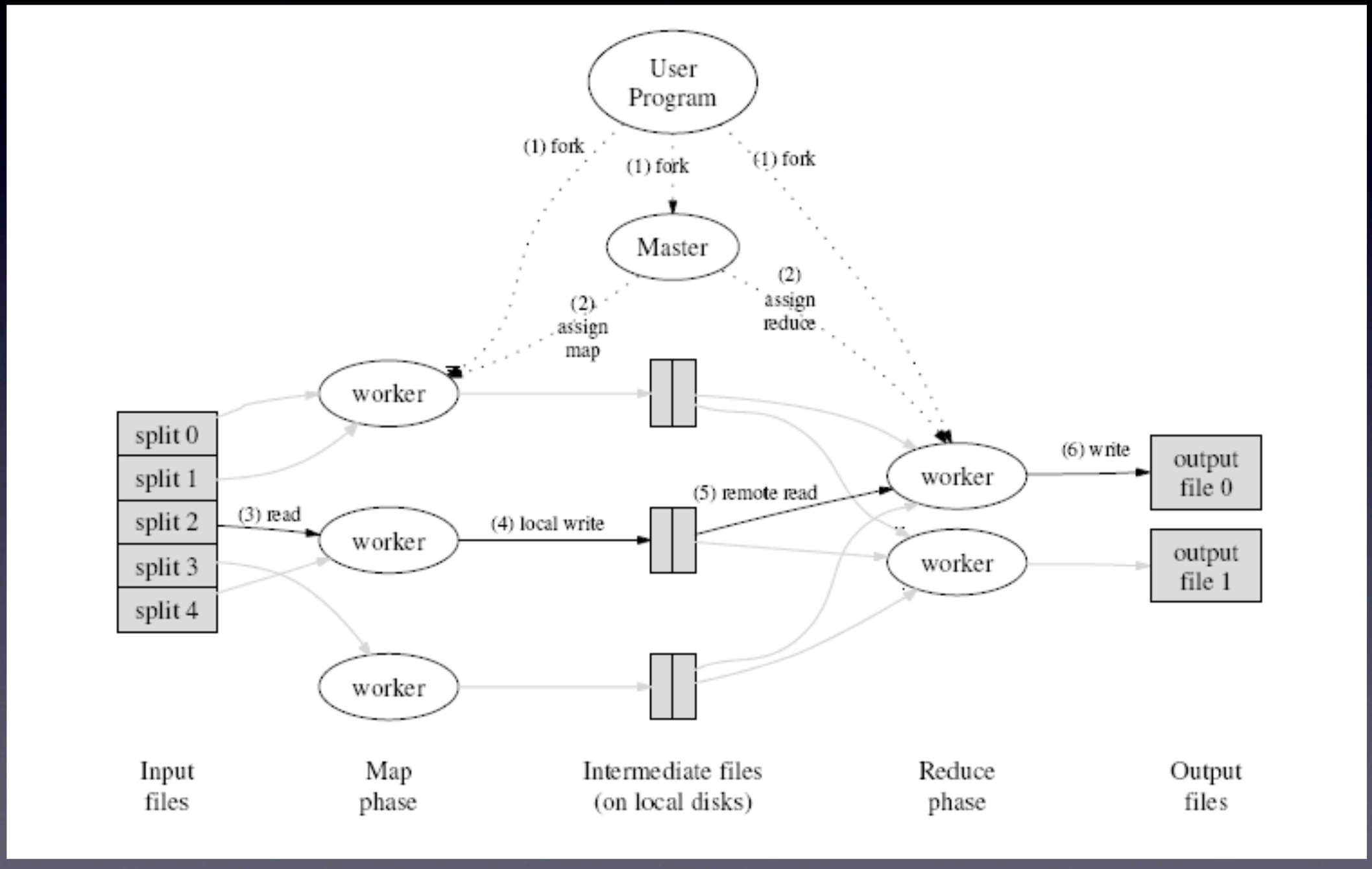
wc avec Hadoop

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

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

wc avec Hadoop

```
public static class IntSumReducer
    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);
  }
}
```

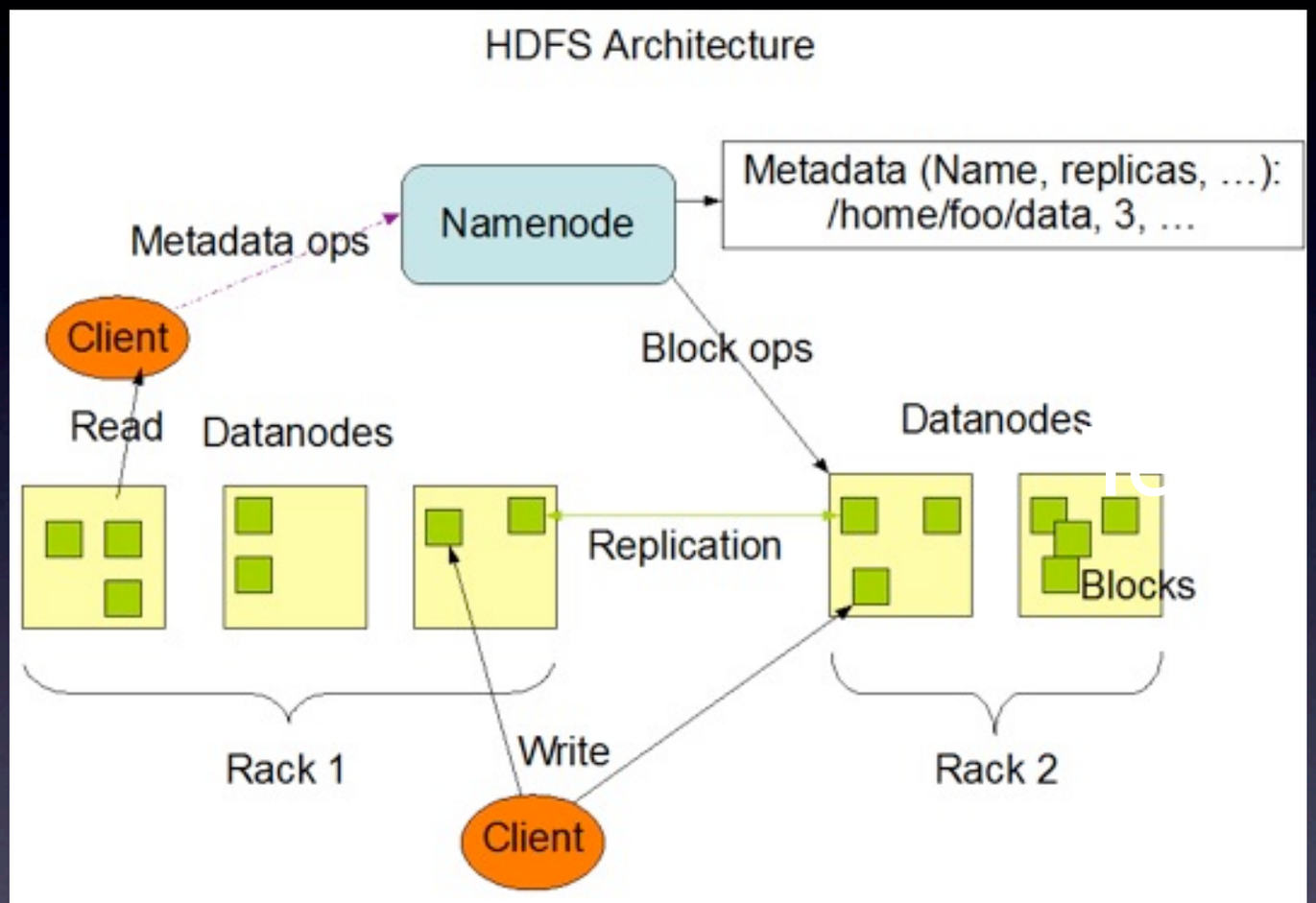
Schéma général d'exécution



Système de fichiers

- Les lectures et les communications entre processus (une JVM par process) se font par écriture de fichiers.
- Hadoop propose le pendant du Google File System (GFS): le Hadoop File System (HDFS)
- Caractéristiques:
  - non-posix,
  - write-once, read-many
  - block-size : par défaut 16 MB

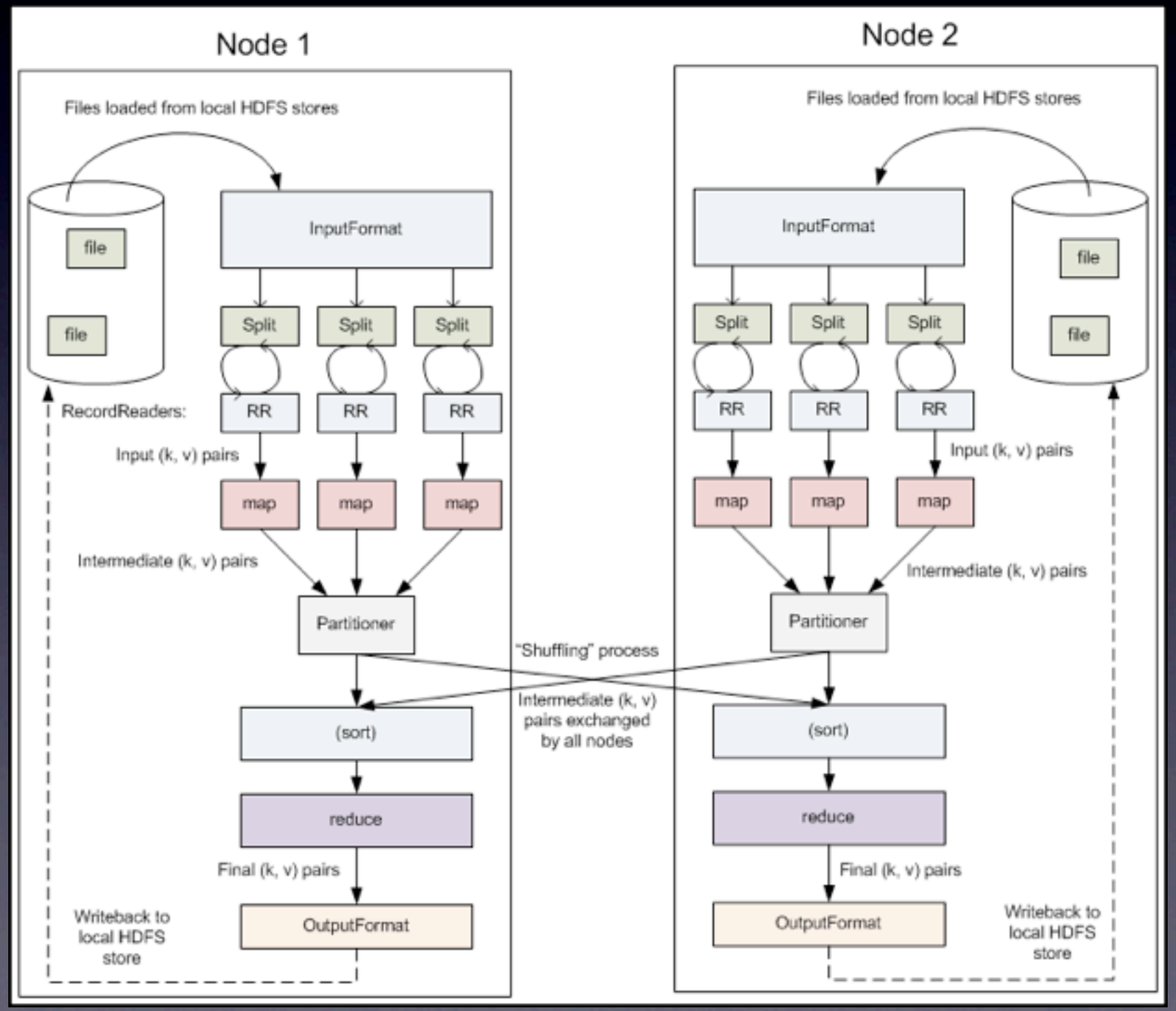
Système de fichiers



Réplication des blocs de données, pour plus de disponibilité et fiabilité.

Namenode centralise les meta-données, et surveille vitalité des blocs par battement de coeurs.

Flot des données



Autres algorithmes utiles pour l'analyse de données

- Algorithmes de graphes
- Classification

- Map-Reduce semble ne pas être applicables à de nombreux problèmes.
- Par exemple, les algorithmes de graphes ne semblent pas adaptés.
- Première étape: trouver une représentation adéquate

- Représentation naturelle d'un graphe comme liste chaînée de noeuds. Inadaptée au parallélisme car nécessite mémoire partagée.

```
class GraphNode
{
  Object data;
  Vector<GraphNode>out_edges;
  GraphNode  iter_next;
}
```

- Représentation plus simplement «partageable»: représentation matricielle :
- Matrice d'adjacence : M[i][j]= '1' quand il y a un lien du sommet *i* à *j*.
- On peut bien sûr mettre la distance sur l'arrête (à la place de 1).

- Pour les grands graphes représentant la plupart des phénomènes réels, les matrice d'adjacence sont creuses (presque que des 0, lignes inutilement longues).

- Utiliser une représentation matrice creuse.
- Ex: sommet: (voisin,distance)

A: (B, 4), (C, 3), (D, 1)

B: (E,2)

C: (G,5)

...

Exemple : degrés des sommets

Problème: *soit la liste des arrêtes d'un graphe exprimées par des relation binaires: (A,B) pour une relation non-orientée entre sommets A et B.*

Etablir pour chaque arrête, le degré de chacun des deux sommets de la relation.

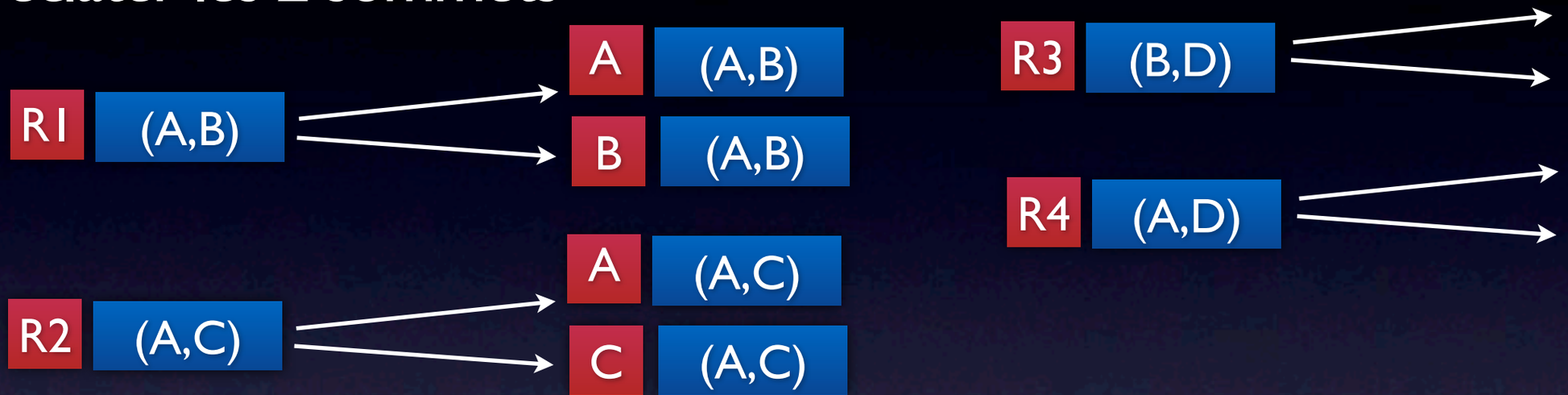
```
(A,B)
(A,C)
(B,D)
(A,D)
```

```
(A,B),d(A)=3,d(B)=2
(A,C),d(A)=3,d(C)=1
(B,D),d(B)=2,d(D)=2
(A,D),d(A)=3,d(D)=2
```

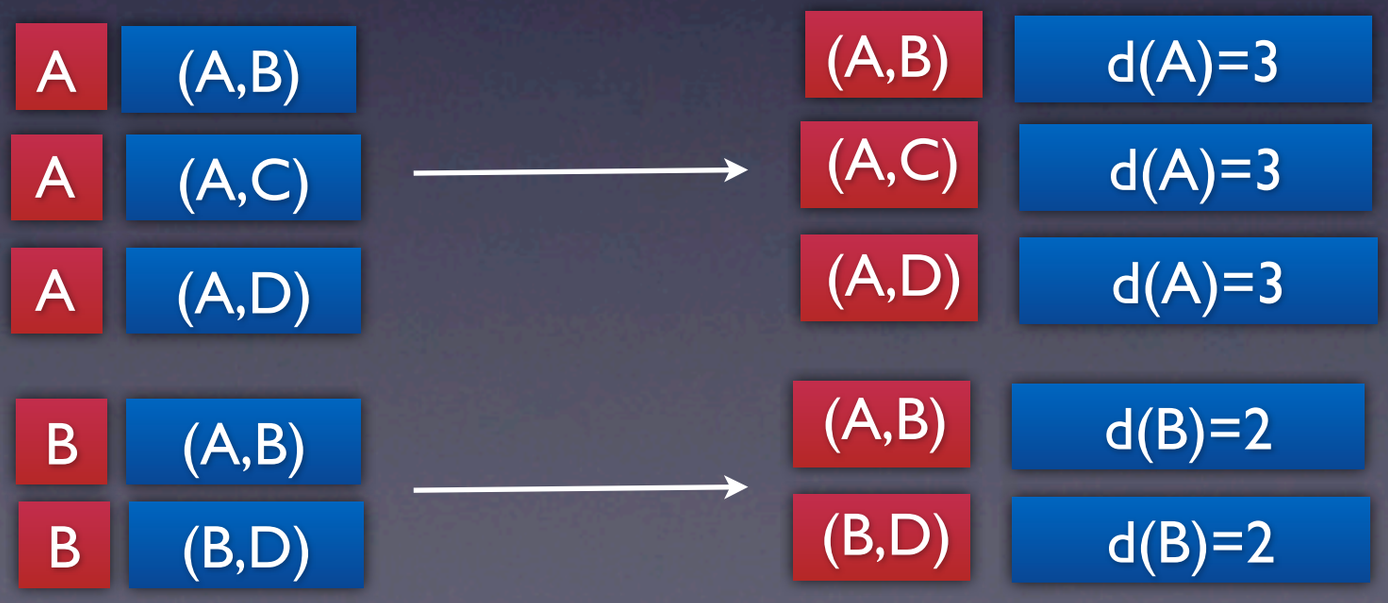
Exemple : degrés des sommets

Nécessite 2 passes:

Map1 : éclater les 2 sommets



Reduce1 : compte le nombre d'enregistrements avec même clé, et ré-émet la relation comme clé avec le degré comme valeur.

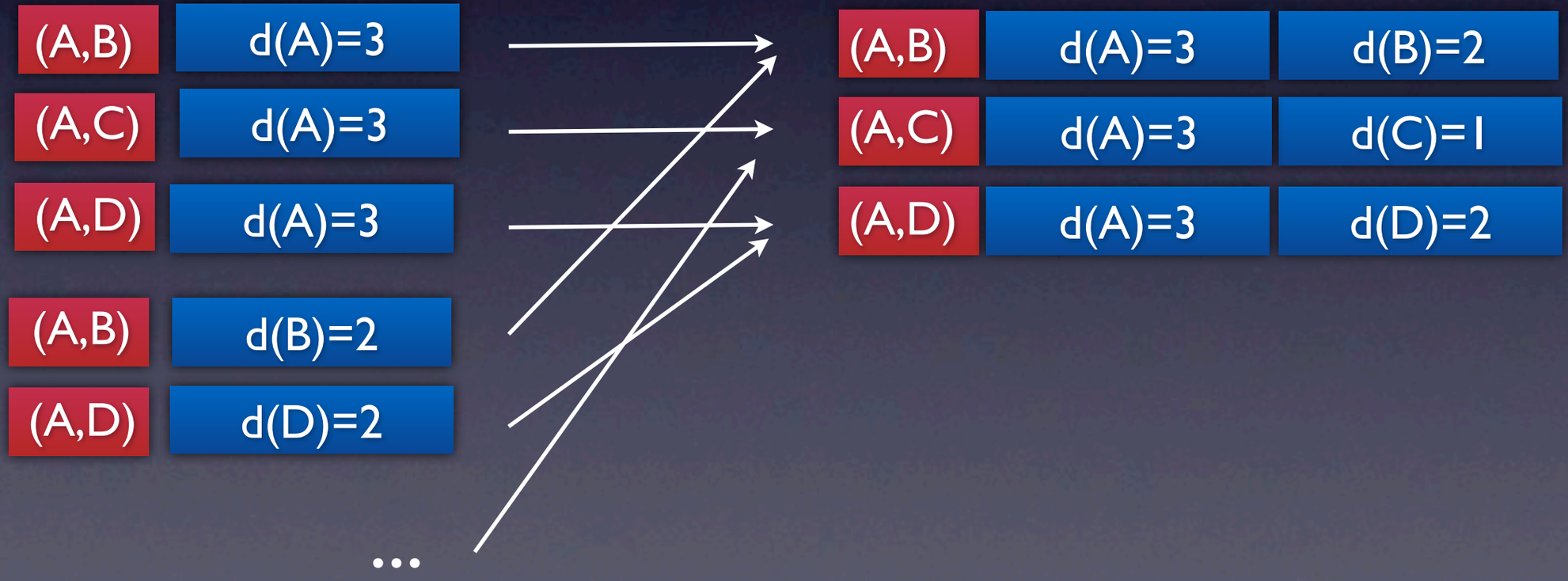


Exemple : degrés des sommets

Passe 2:

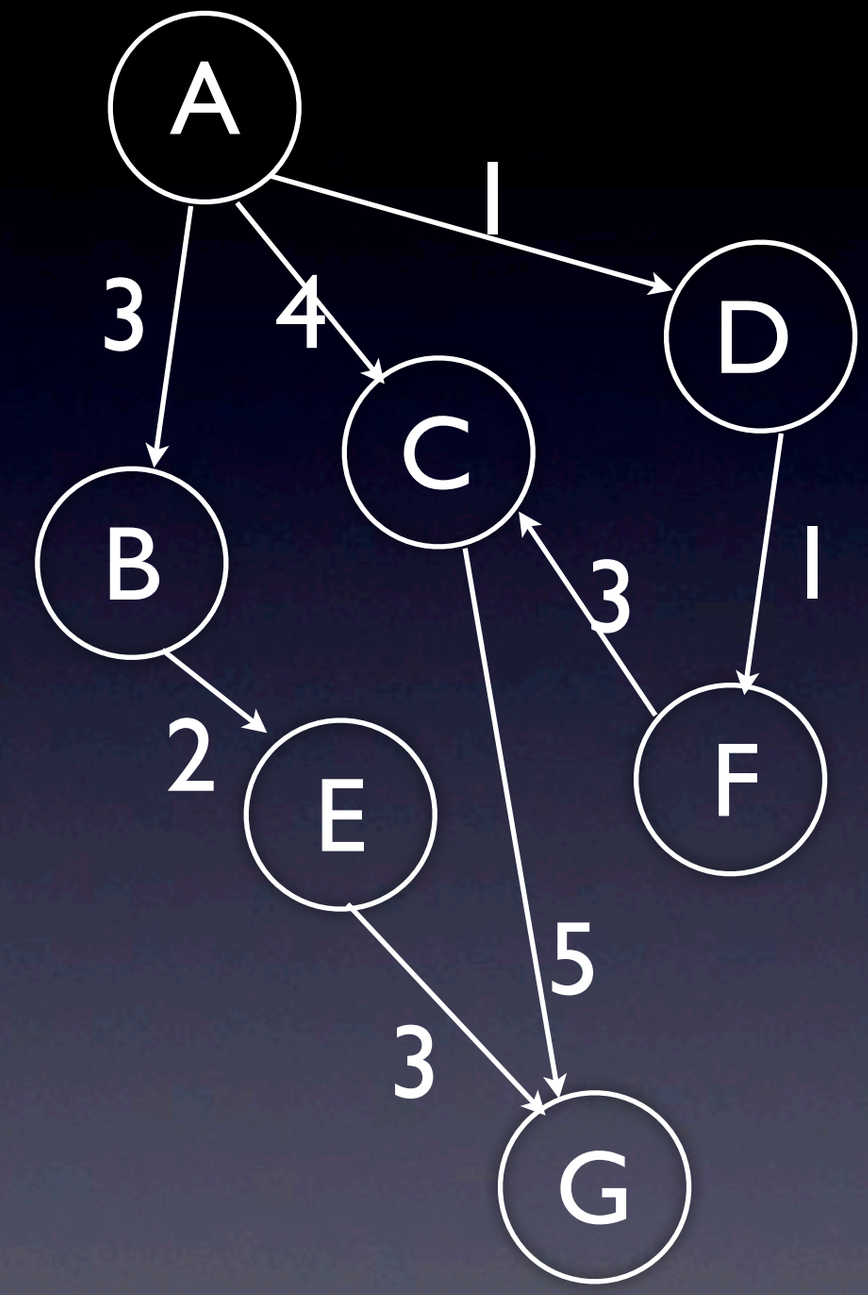
Map2 : identité (quand appariement map et reduce obligatoire comme avec Hadoop)

Reduce2 : agrège les degrés trouvés pour une même relation

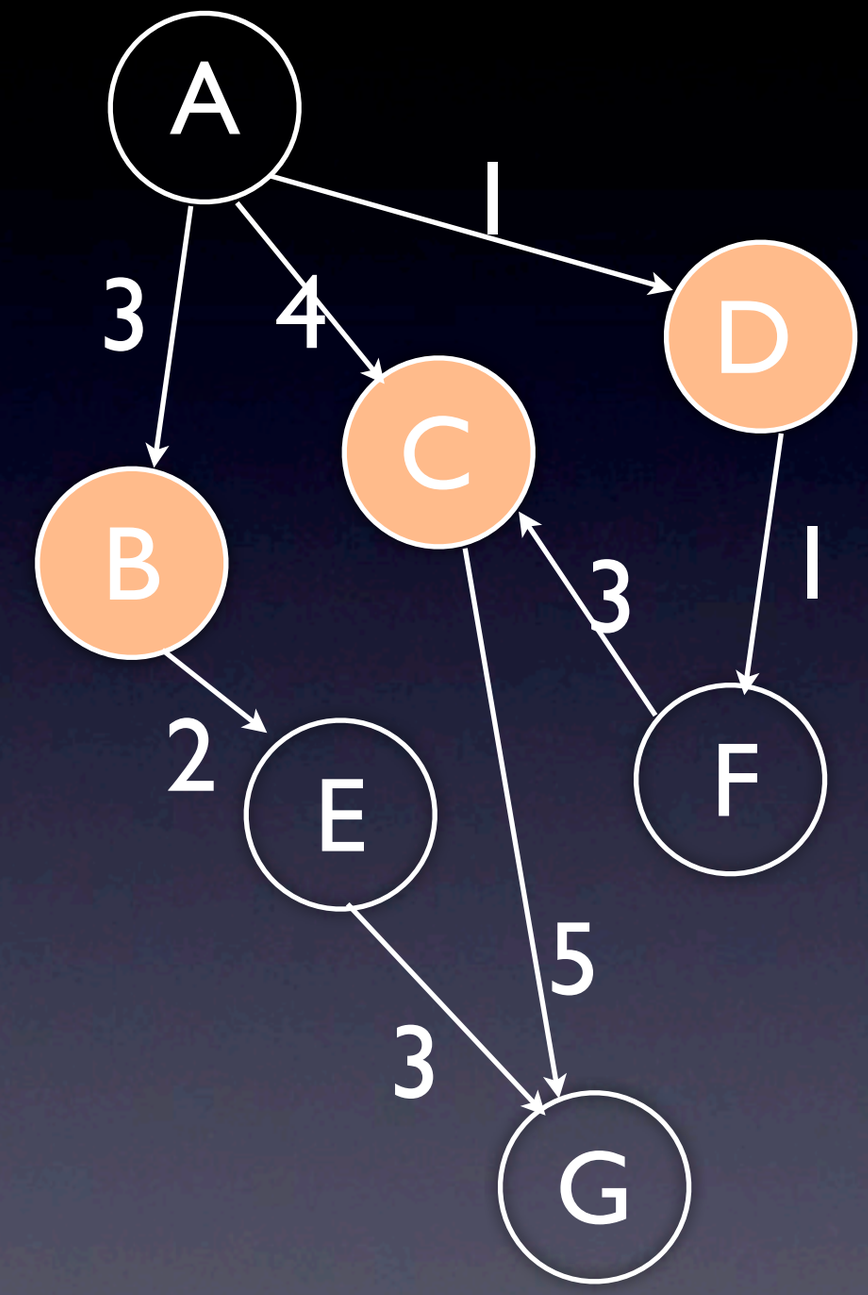


Exemple du plus court chemin

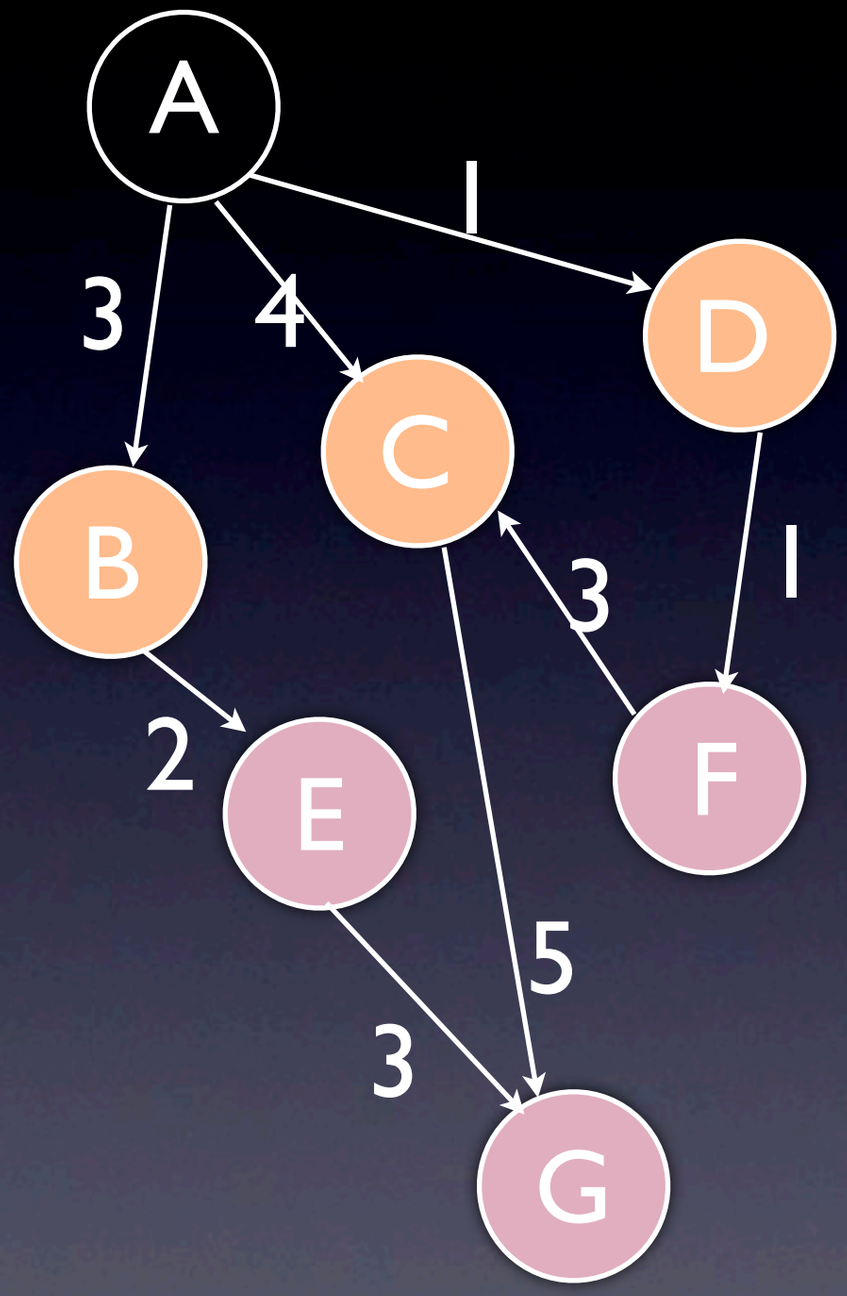
- Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- Dans le cas d'un graphe, nécessité de colorer les sommets pour l'arrêt.



- Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- Dans le cas d'un graphe, nécessité de colorer les sommets pour l'arrêt.



- Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- Dans le cas d'un graphe, nécessité de colorer les sommets pour l'arrêt.



Plus court chemin

On peut définir la solution de manière inductive: soit le sommet de départ n_0

- DistanceTo(n_0) = 0

- Pour tous les sommets *n* accessibles directement de n_0, DistanceTo(n) = 1

- Pour tous les sommets *n* accessibles à partir d'un autre ensemble de sommets *S*,

$$\text{DistanceTo}(n) = 1 + \min(\text{DistanceTo}(m), m \in S)$$

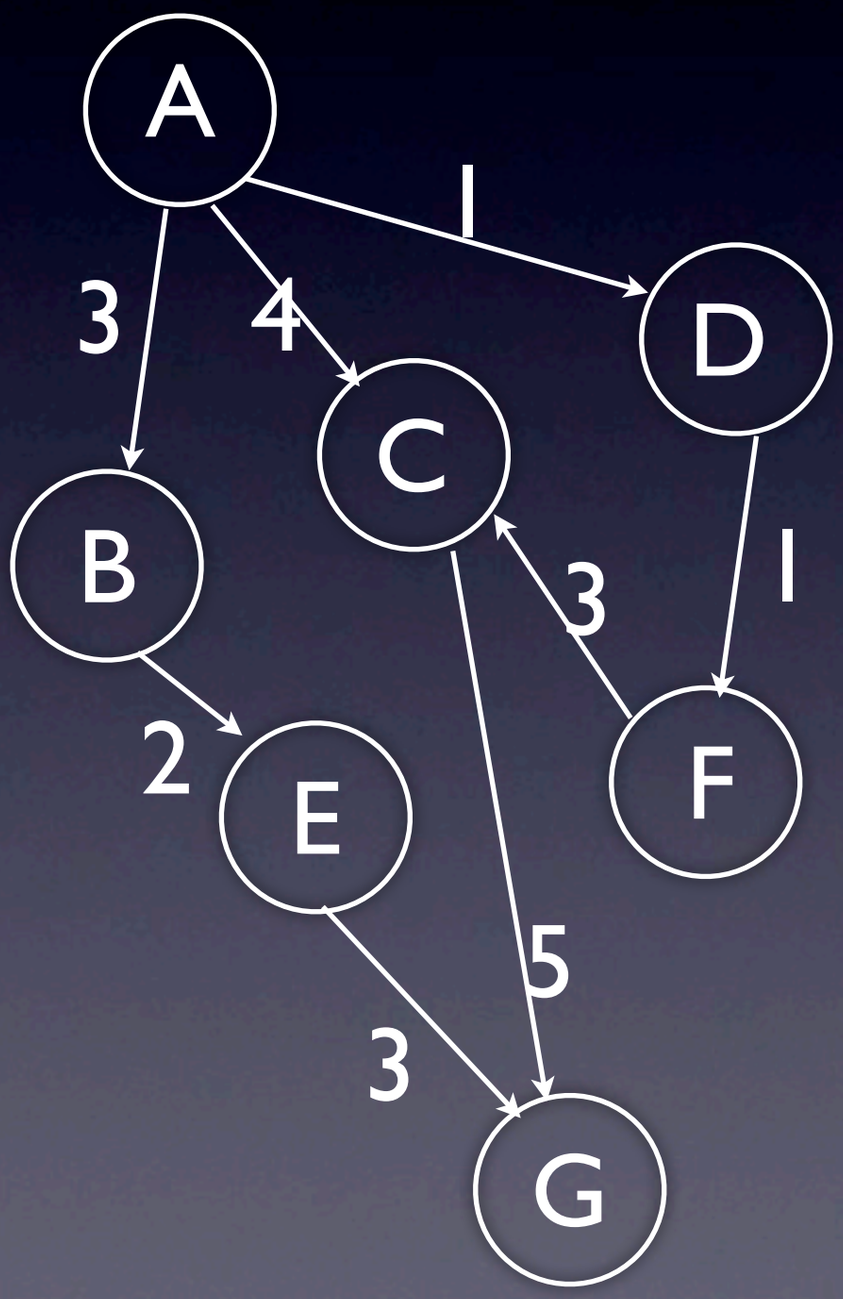
Algorithme Map-Reduce

- La fonction map reçoit :
- clé : un sommet *n*,
- valeur : *(D, points-to)*
  - *D* : distance du sommet à partir du début
  - *points-to* : liste des sommets accessibles depuis *n*
- ∀p ∈ points-to, emet (p, D+1)

- La fonction Reduce traite toutes les distances possibles vers un *p* donné et sélectionne le plus petit.

Algorithme Map-Reduce

On part de A: la distance du début est 0. Pour les autres, initialement infinie.



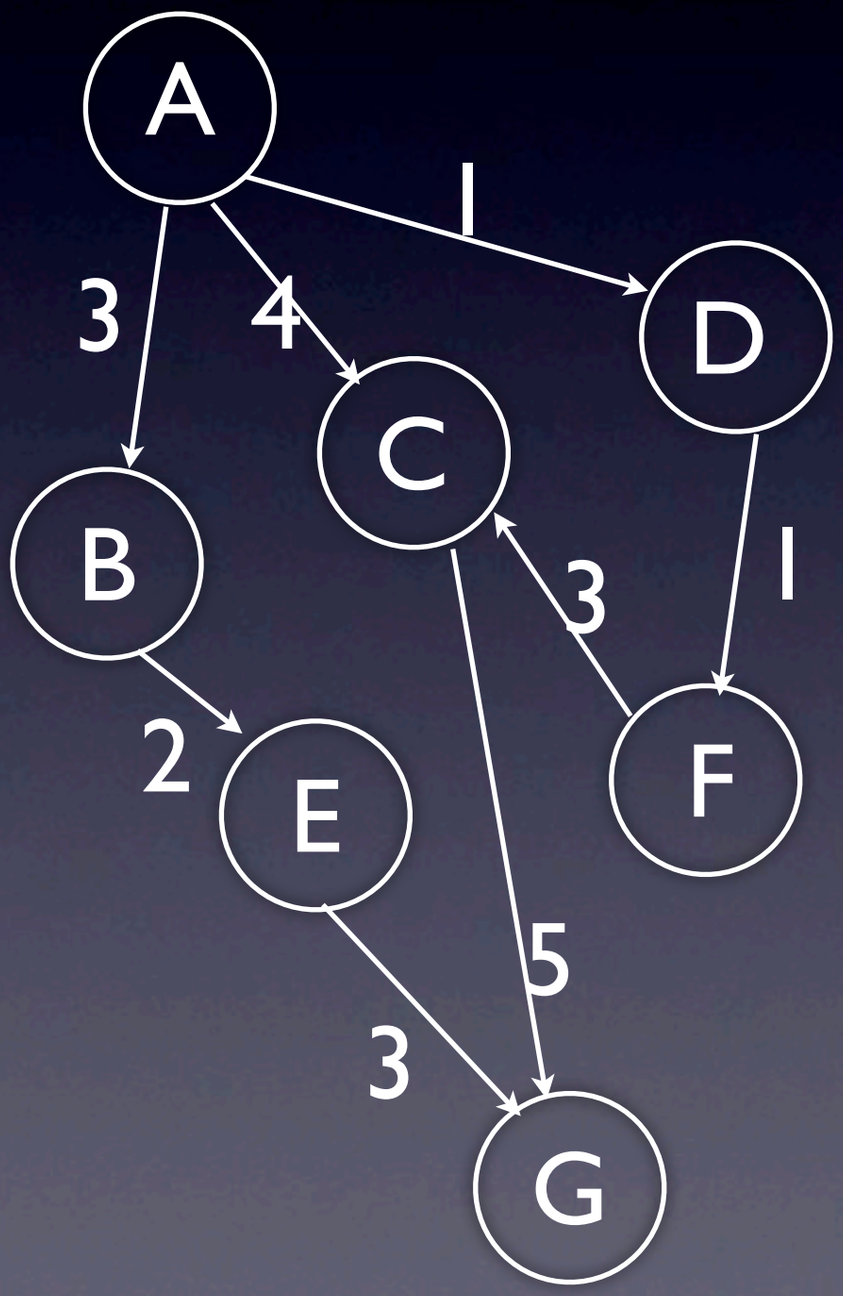
A	0 , {B,C,D}
B	inf, {E}
C	inf, {G}
D	inf, {F}
E	inf, {G}
F	inf, {C}

map

B	3 , {}
C	4 , {}
D	1 , {}
B	inf, {E}
C	inf, {G}
D	inf, {F}
E	inf, {G}
F	inf, {C}

Ce map génère des paires, comme (B,(3,{}), que l'on peut interpréter comme : à partir de A, on atteint B, en ayant parcouru une distance de 3.

On réduit en prenant pour une même clé, la distance minimum.



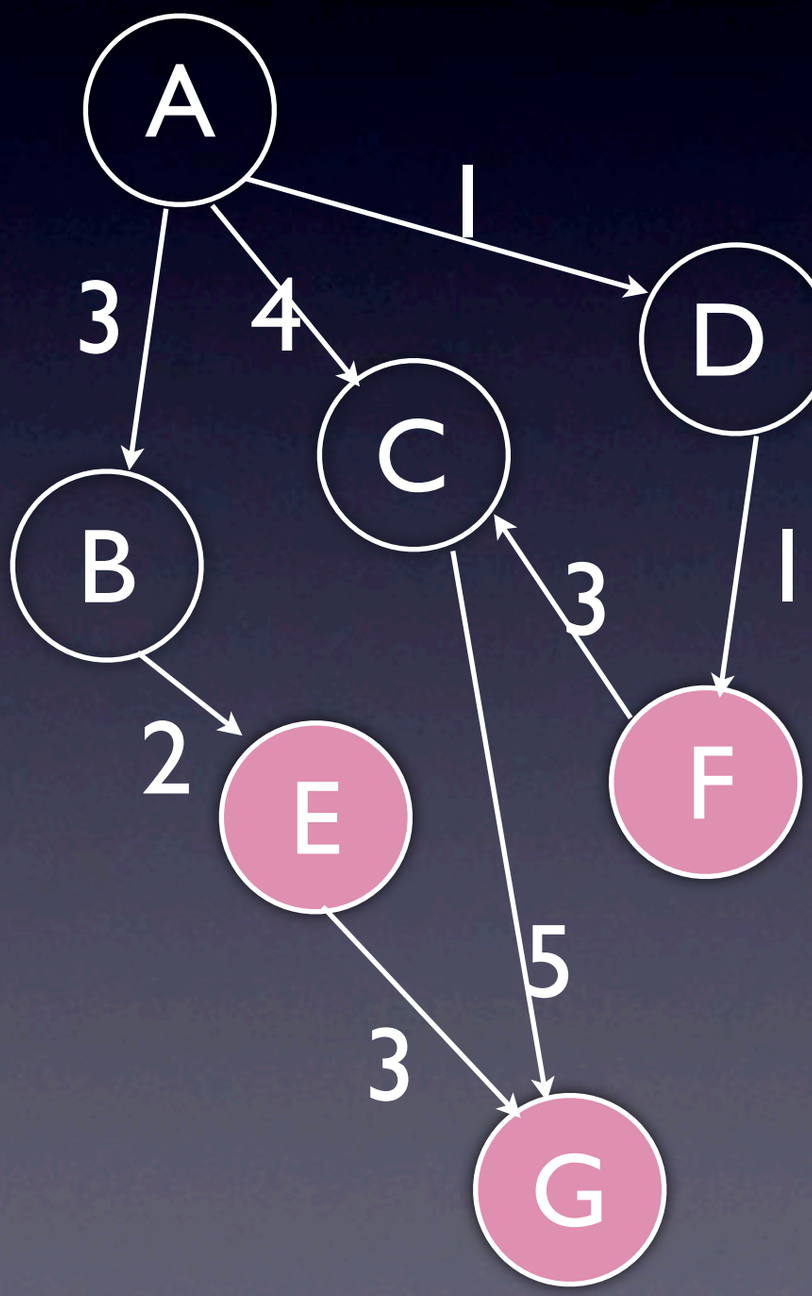
B	3 , {}
C	4 , {}
D	1, {}
B	inf, {E}
C	inf, {G}
D	inf, {F}
E	inf, {G}
F	inf, {C}

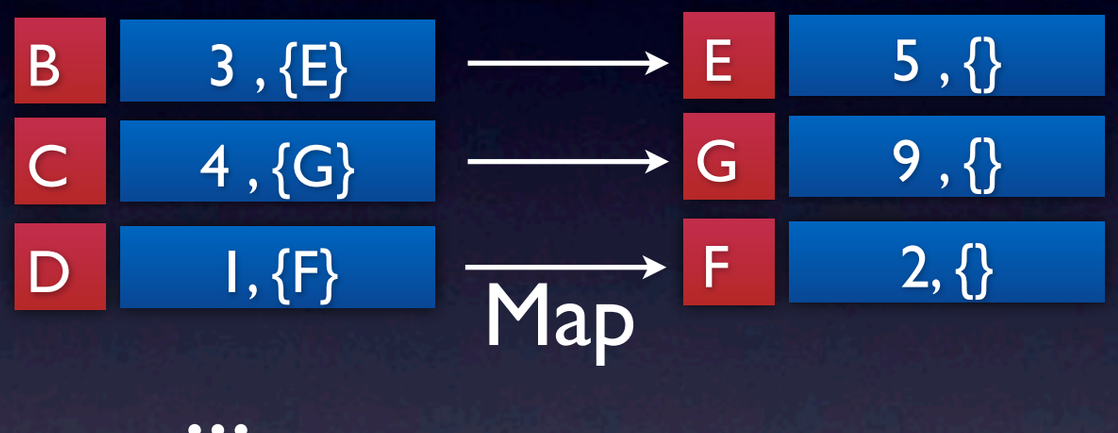
Reduce

B	3 , {E}
C	4 , {G}
D	1, {F}
E	inf, {G}
F	inf, {C}

passe 1

Le deuxième map génère de nouvelles clés ‘sommet’ en substituant
(clé,(d,{succ1,succ2,...}))
par
[(succ1,(d+dist(clé,succ1),{});
 (succ2,(d+dist(clé,succ2),{}); ...]

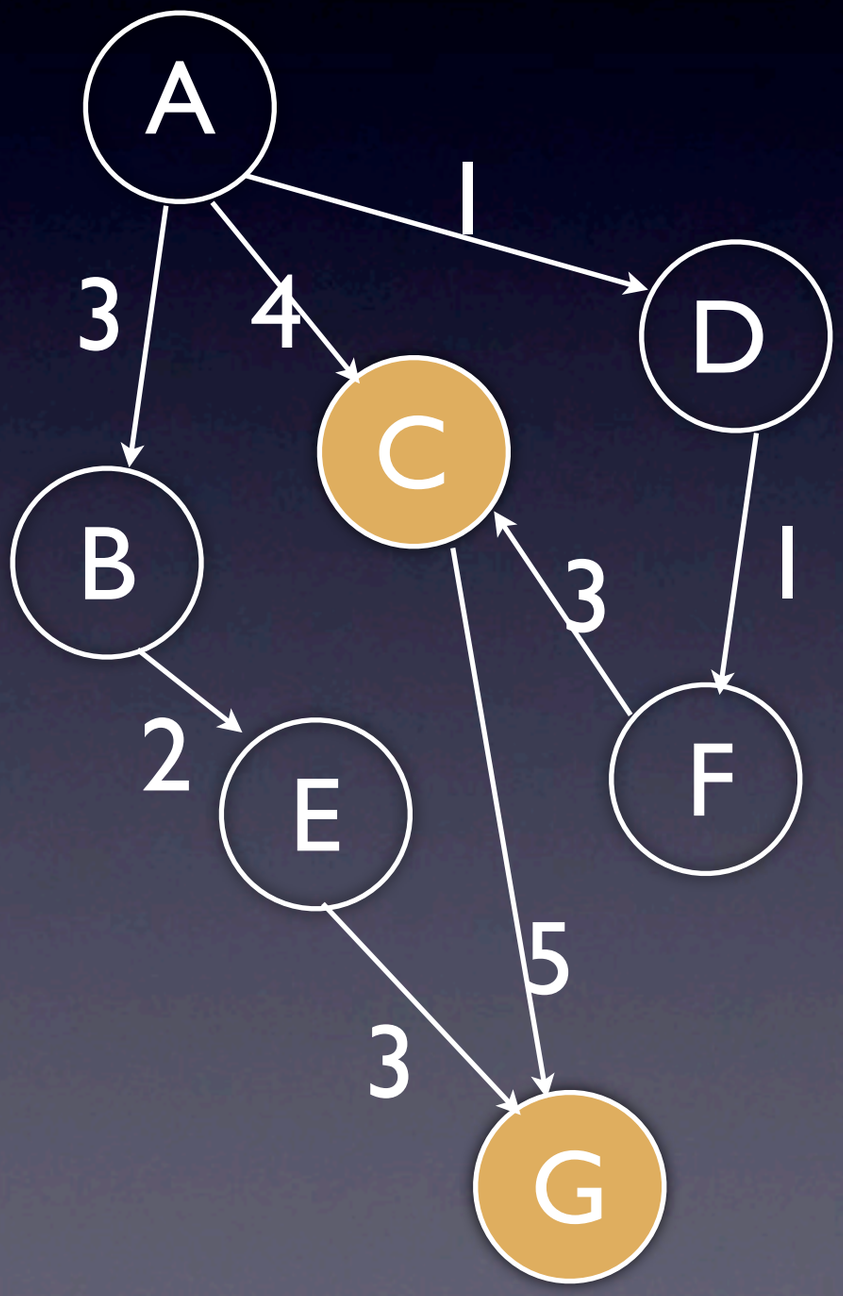




...

passe 2

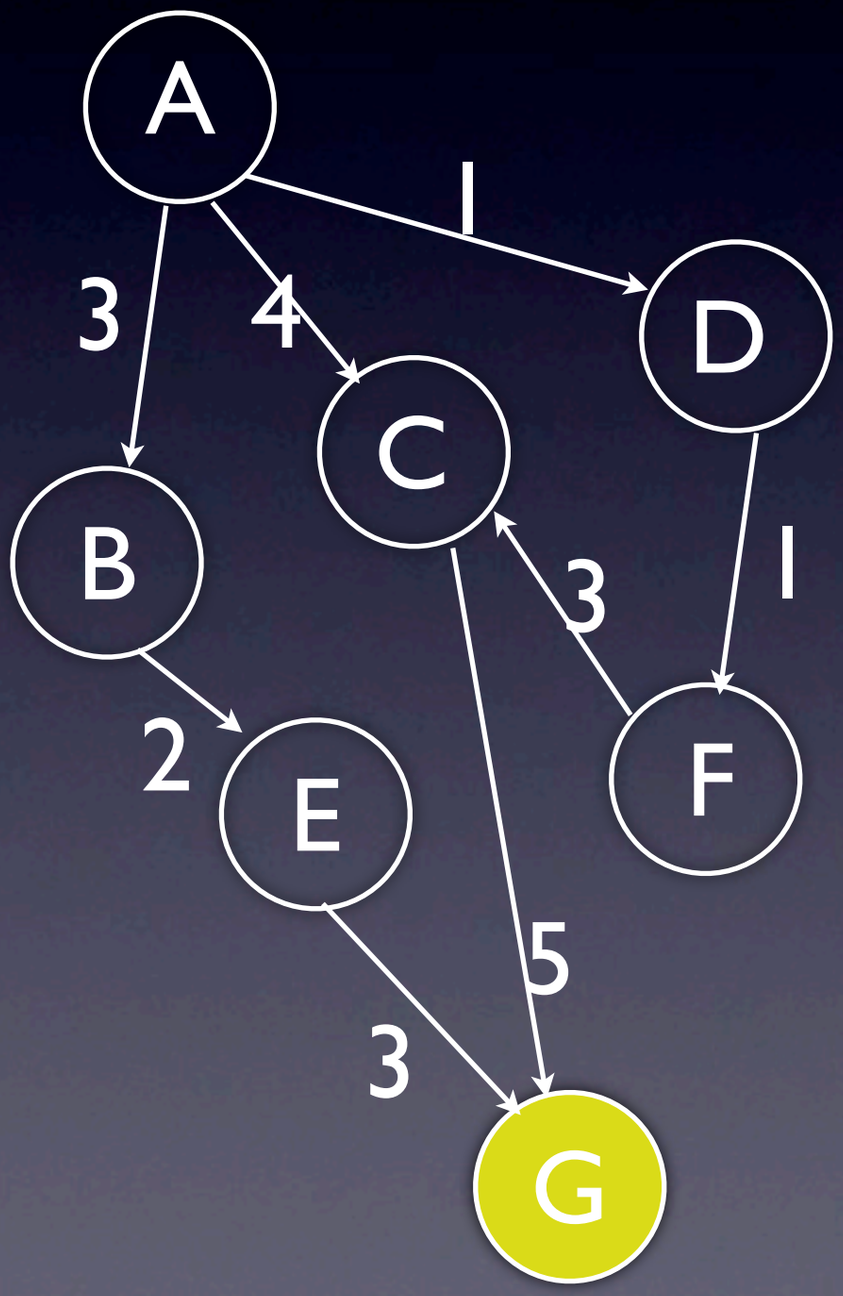
On réduit en prenant pour une même clé, la distance minimum

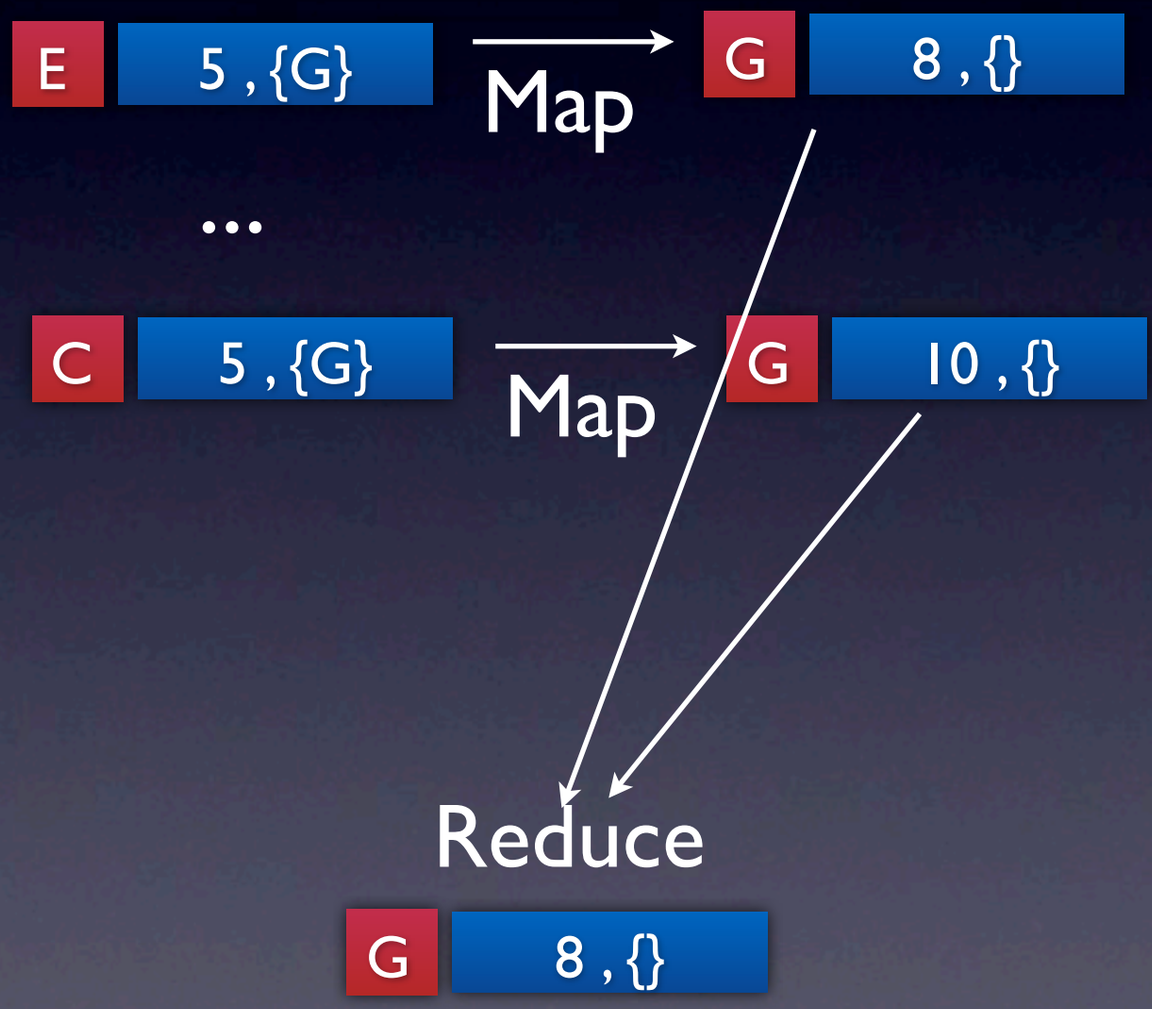


...

passe 3

On réduit en prenant pour une même clé, la distance minimum





passe 4

Clustering

- Classification non-supervisée: trouver *k* groupes de points tels que les points dans chaque groupe soient plus «proches» de ceux de leur groupe que des autres groupes.
- K-means :

1. prendre au hasard k points initiaux, désignés comme «centres».
2. pour chaque point, définir quel est le centre le plus proche. Le point rejoint le groupe associé au centre.
3. pour chaque groupe, définir un nouveau centre, barycentre des points appartenant au groupe.
4. Recommencer l'étape 2. Si aucun point ne change de groupe, fin de l'algorithme, sinon recommencer à l'étape 3.

Complexité k-means

- complexité K-means : pour *n* points, *k* centres, complexité distance *d, i itérations : k.n.d.i*