

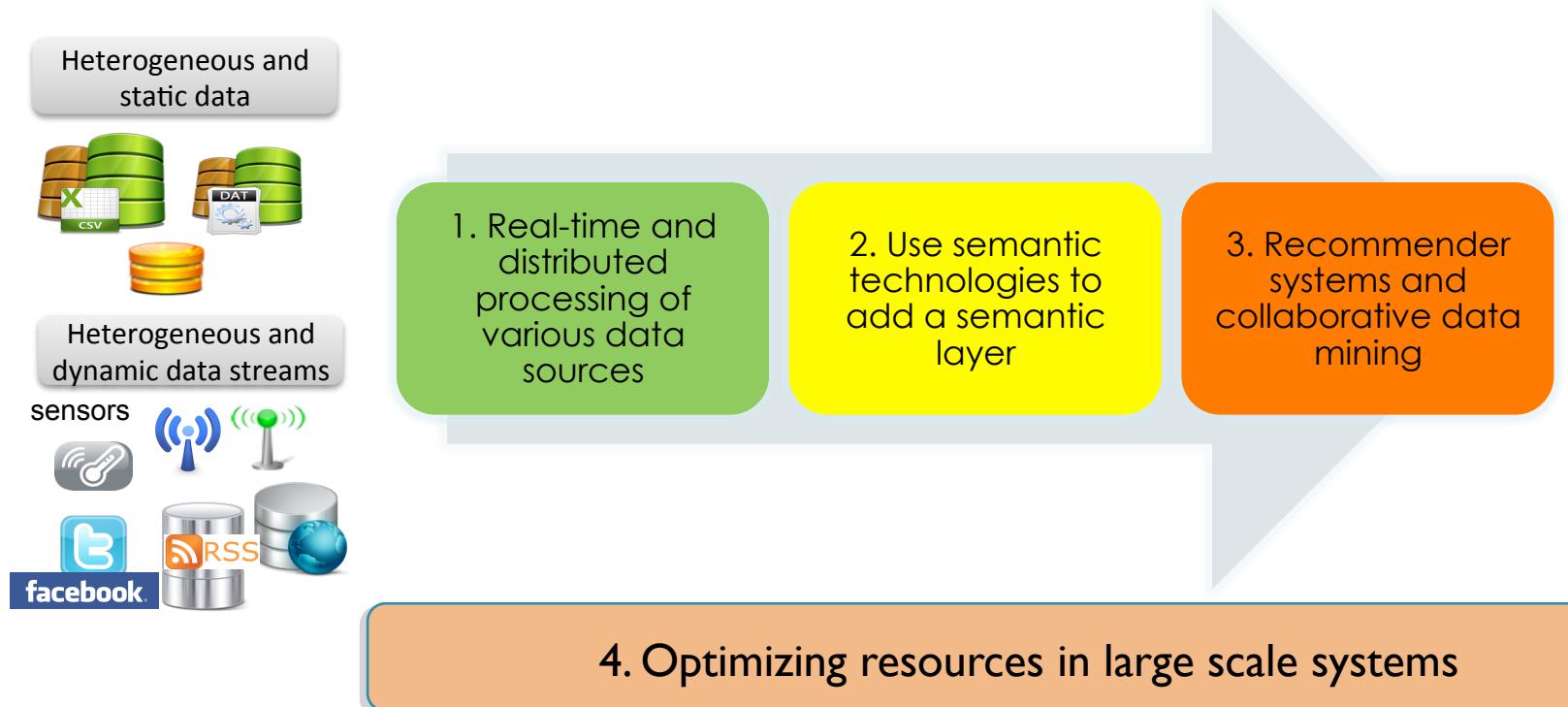
# Bases de données NoSQL

Raja CHIKY  
[raja.chiky@isep.fr](mailto:raja.chiky@isep.fr)  
2014-2015



## About me

- Associate professor in Computer Science – head of LISITE-RDI Research team
- Research interest: Data stream mining, scalability and resource optimization in distributed architectures (e.g cloud architectures), recommender systems
- Lecturer in CNAM, Telecom Sud (INT), ECP, Institut Mines-Télécom, etc.
- Research field: Large scale data management





# Plan

- 
- **Introduction**
  - **Concepts de base autour des BDs distribués**
  - **MapReduce**
  - **Bases de données clé-valeur**
  - **Bases de données orientées colonne**
  - **Bases de données orientées document**
  - **Bases de données orientées graphe**



# Introduction



# Big Data: Buzzword!



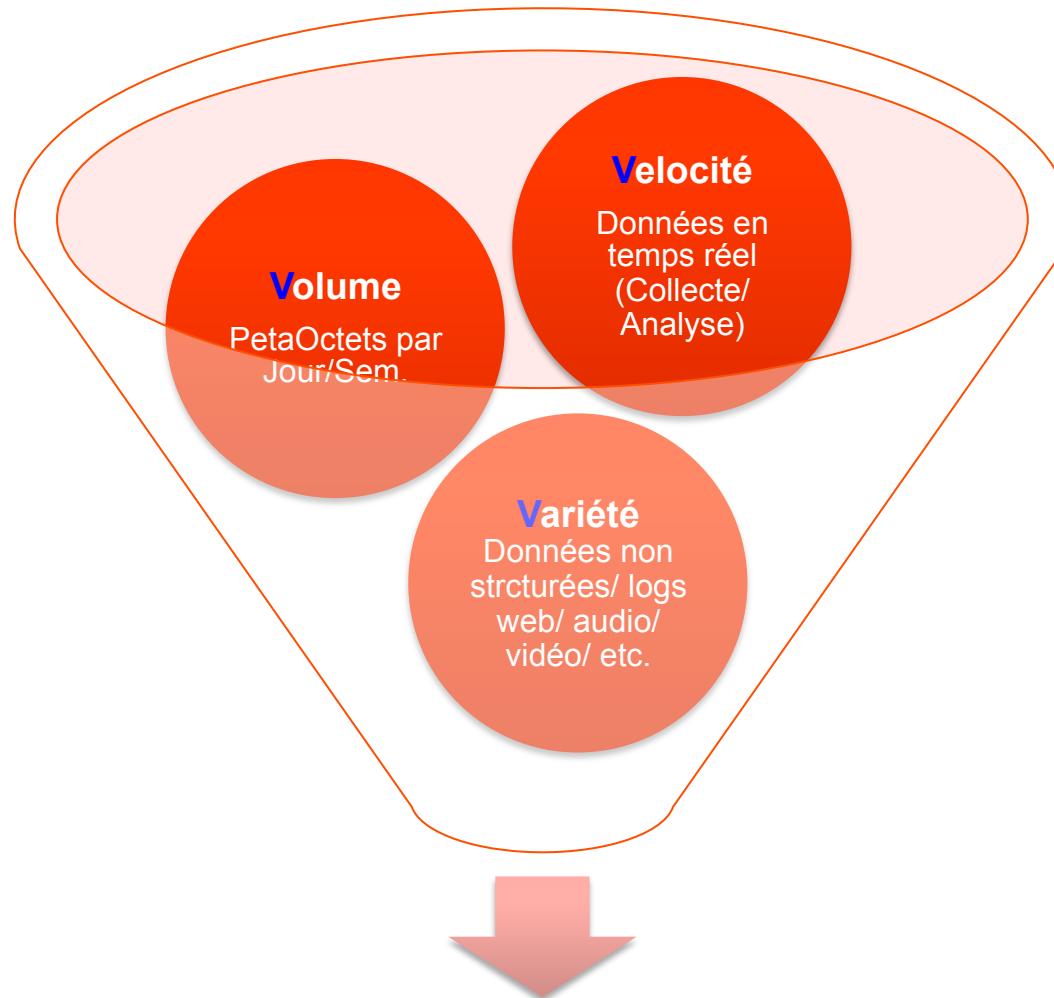
The collage includes the following items:

- PM NETWORK**: BIG DATA: FUELING GOVERNMENT EFFICIENCY
- Forbes**: Big Data, Big Payoff: Delivering Moneyball Results To Business
- nature**: THE BIGGEST DATA SCIENCE CHALLENGE IS BIG DATA ITSELF
- Enjeux et usages du Big Data**: technologies, méthodes
- THE FLOOD OF BIG DATA**: Infographic showing the growth of big data from 1960 to 2014.
- Big Data Technologies for Near-Real-Time Results**: White paper by Intel.
- The Economist**: The data deluge AND HOW TO HANDLE IT: A 14-PAGE SPECIAL REPORT
- MANAGEMENT ET INFORMATIQUE**: collection dirigée par Nicolas Monier
- Marketing Data**: The Path to Personalization
- Economist Intelligence Unit**: Big data and the democratisation of decisions
- BUSINESS TECHNOLOGY**: Big Data Broadens Its Range
- SPDR Paper No. 13**: Climate Risk, Big Data and the Weather Market
- The Age of Big Data**: By STEVE LOWE
- RÉFÉRENTIEL DE PRATIQUES**: Meilleures approches pour tirer parti du BIG DATA
- ISEP**: Institut Mines-Télécom

# Introduction

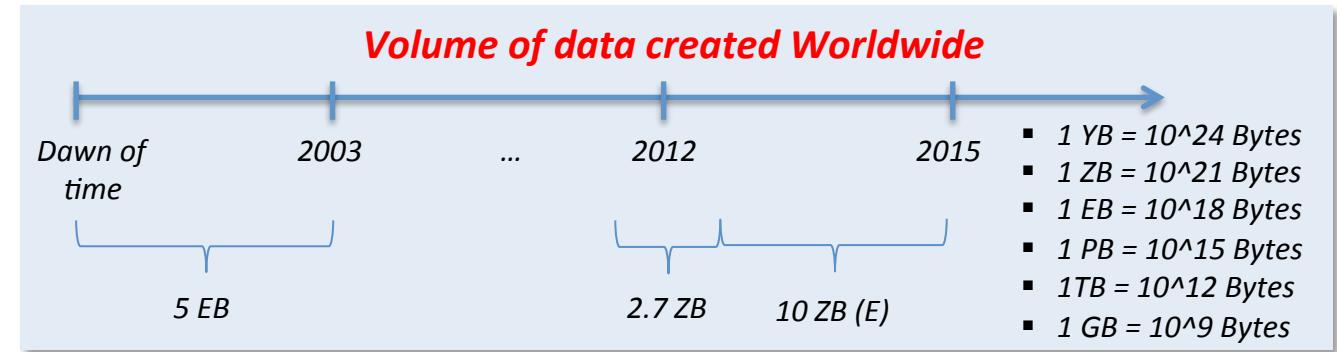
- **Des entreprises face à de nouveaux usages**
    - Une augmentation exponentielle des volumes de données à traiter
    - La prise en compte des données en tant que ressources stratégiques
  - **Des technologies en constante évolution**
    - Accroissement des capacités de calculs et d'analyse
    - Coût de stockage moins onéreux
- 
- L'émergence du Big Data
    - Comment mettre à profit des ressources jusqu'alors inexploitées?
    - Comment mettre à profit des sources disponibles ailleurs?

# Big Data: 3V

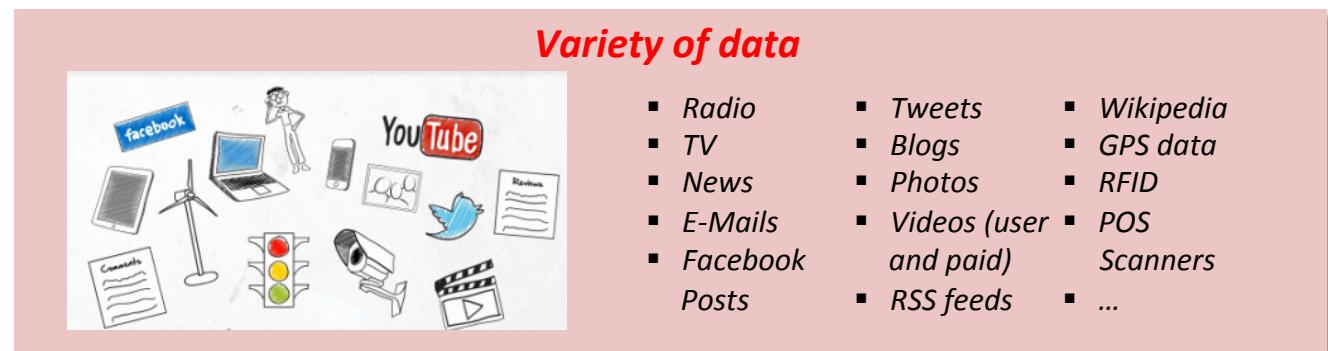
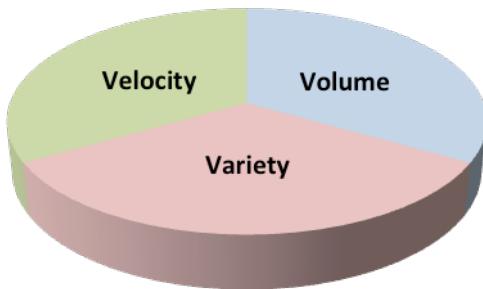


En extraire la **Valeur**

# So, what is Big Data?



## Big Data Elements



+ Veracity (IBM) -  
information uncertainty

Source: Big Data & Analytics - Why Should We Care?

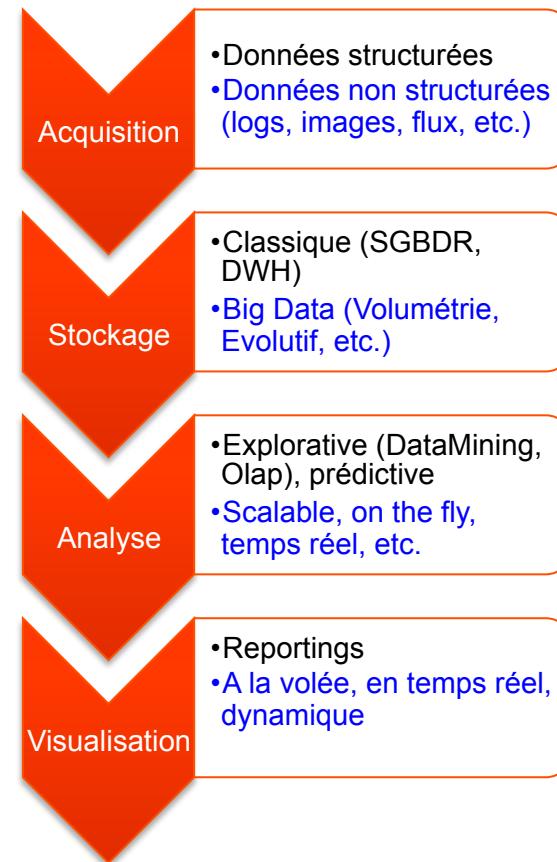
- Walmart handles 1M transactions per hour
- Google processes 24PB of data per day
- AT&T transfers 30 PB of data per day
- 90 trillion emails are sent per year
- World of Warcraft uses 1.3 PB of storage

## Velocity of data

- Facebook when had a user base of 900 M users, had 25 PB of compressed data
- 400M tweets per day in June '12
- 72 hours of video is uploaded to Youtube every minute

# Défis du Big Data

- **Appréhender une croissance exponentielle du volume de données**
- **Maîtriser la variété des données et en extraire l'information pertinente (la valeur)**
- **Traitements temps réel des informations**
- **Rendre l'information accessible au plus tôt**
- **Anticiper et faciliter la prise de décision**



# Exemple simple: Google-Grippe

[Page d'accueil de Google.org](#) (en anglais)

[Suivi de la dengue](#)

[Suivi de la grippe](#)

[Accueil](#)

[Sélectionnez un pays/territoire](#)

[Comment ça marche ?](#)

[FAQ](#)

[Propagation du virus](#)

Très élevée

Élevée

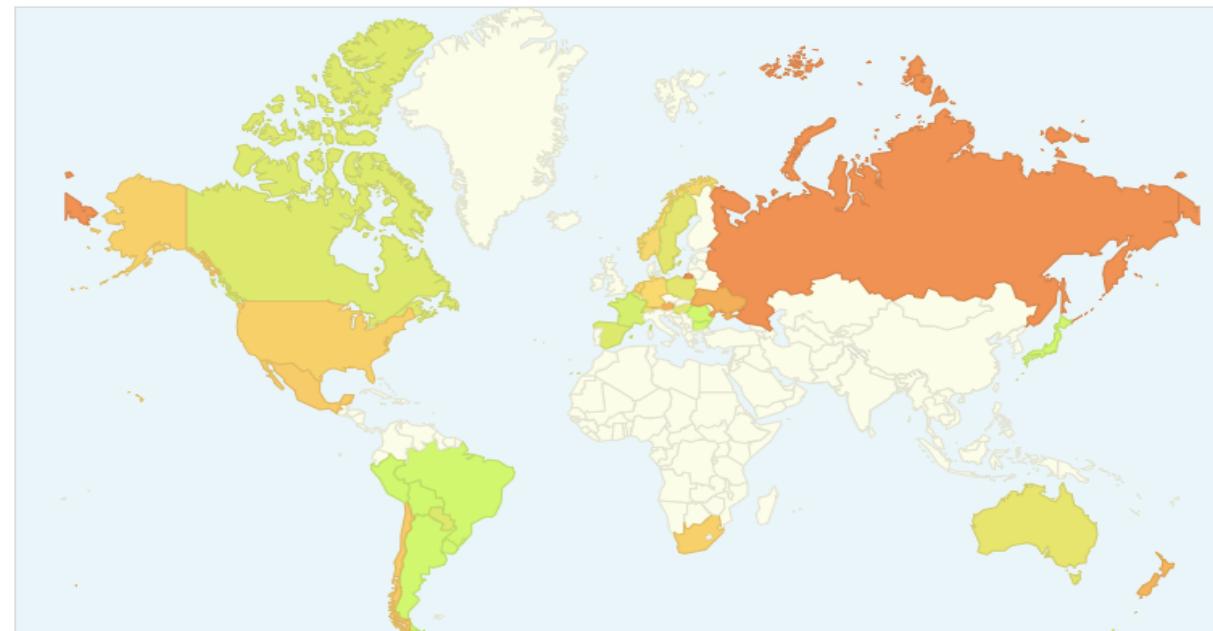
Modérée

Basse

Minimale

## Suivez l'évolution de la grippe dans le monde entier

Certains termes de recherche semblent être de bons indicateurs de la propagation de la grippe. Afin de vous fournir une estimation de la propagation du virus, ce site rassemble donc des données relatives aux recherches lancées sur Google. [En savoir plus »](#)

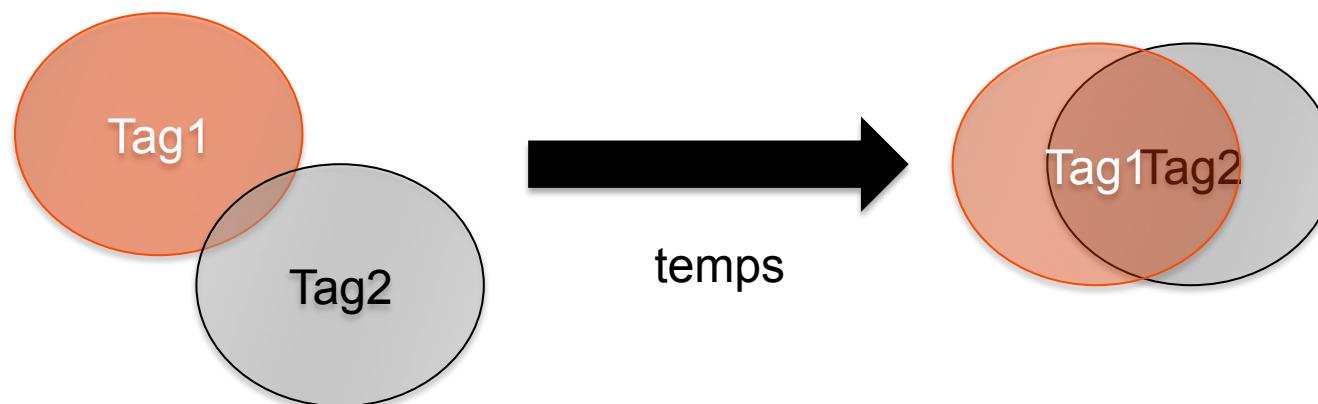


## Exemple 2: extraction de tendances

- **Extraire les tendances dans les flux textuels**
- **Un grand volume de données bruitées et non structurées**
- **Exemple de tendances**

#benoitXVI #demanion

#armstrong #dopage







## **Concepts de base autour des BD's distribuées**

Extensibilité – Sharding -  
MapReduce

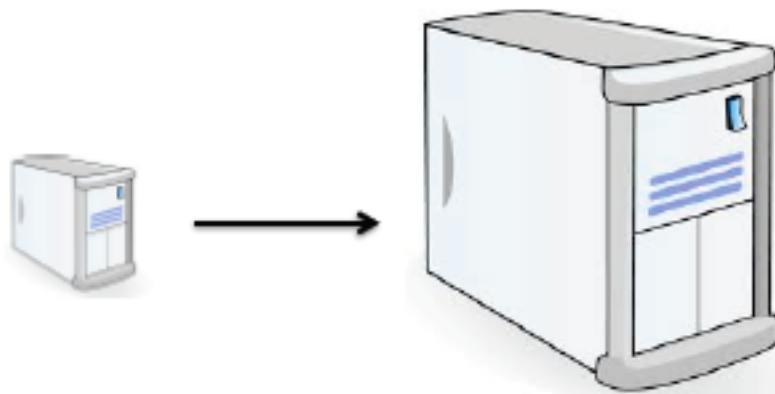


## Extensibilité (scalability)

- L'extensibilité est la propriété d'un système, d'un réseau ou d'un processus qui témoigne de sa capacité à gérer des charges de travail importantes en toute souplesse ou à être agrandi sans difficulté
- Deux types:
  - Verticale
  - Horizontale

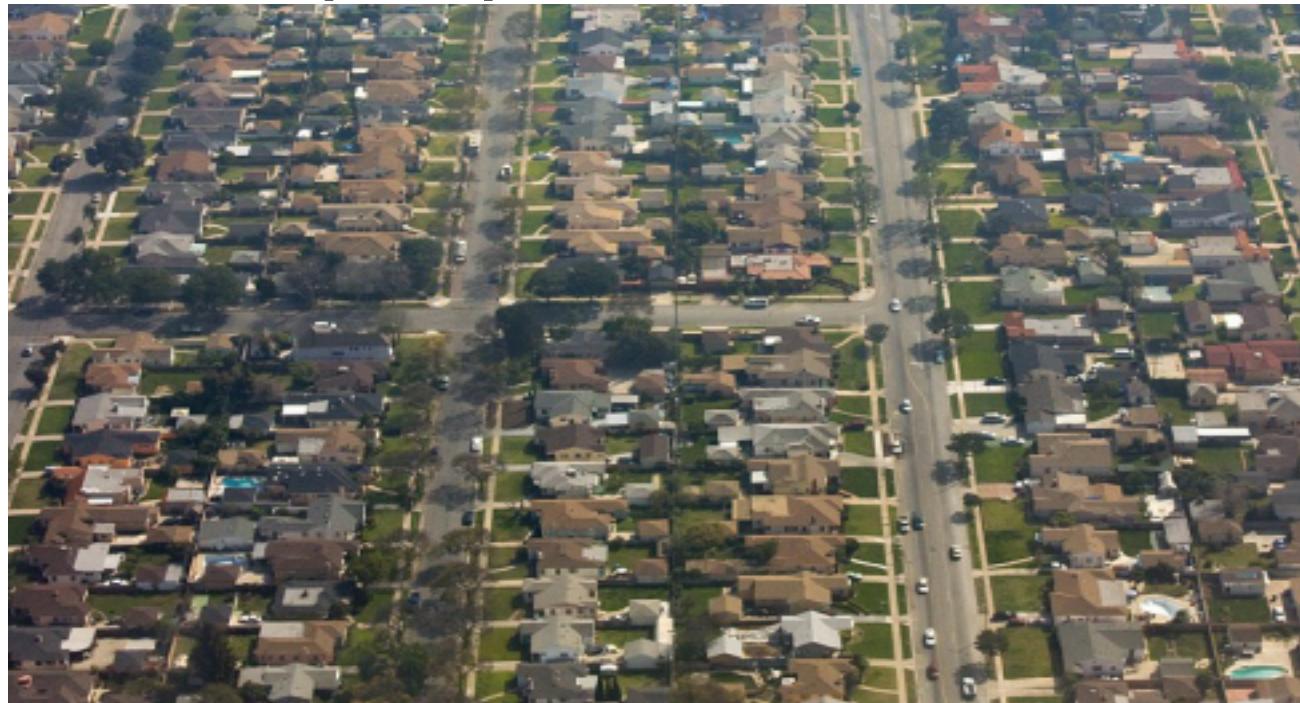
## Extensibilité verticale

- **En pratique**
  - Remplacer les serveurs par des machines plus puissantes
- **Coût important**
- **Mise en place facile**



# Extensibilité horizontale

- **En pratique:**
  - Ajout de machines ordinaires
- **Coût moins important**
- **Mise en place plus délicate**



## Monter en charge un SGBDR

- **Problèmes de passage à l'échelle quand le jeu de données est trop volumineux**
- **SGBDR n'ont pas été conçues pour être distribués**
- **Solutions de bases de données distribuées (multi-nœuds)**  
=>«passage à l'échelle horizontale»
- **Différentes approches:**
  - Maître-esclave
  - Partitionnement (sharding)

# Monter en charge un SGBDR

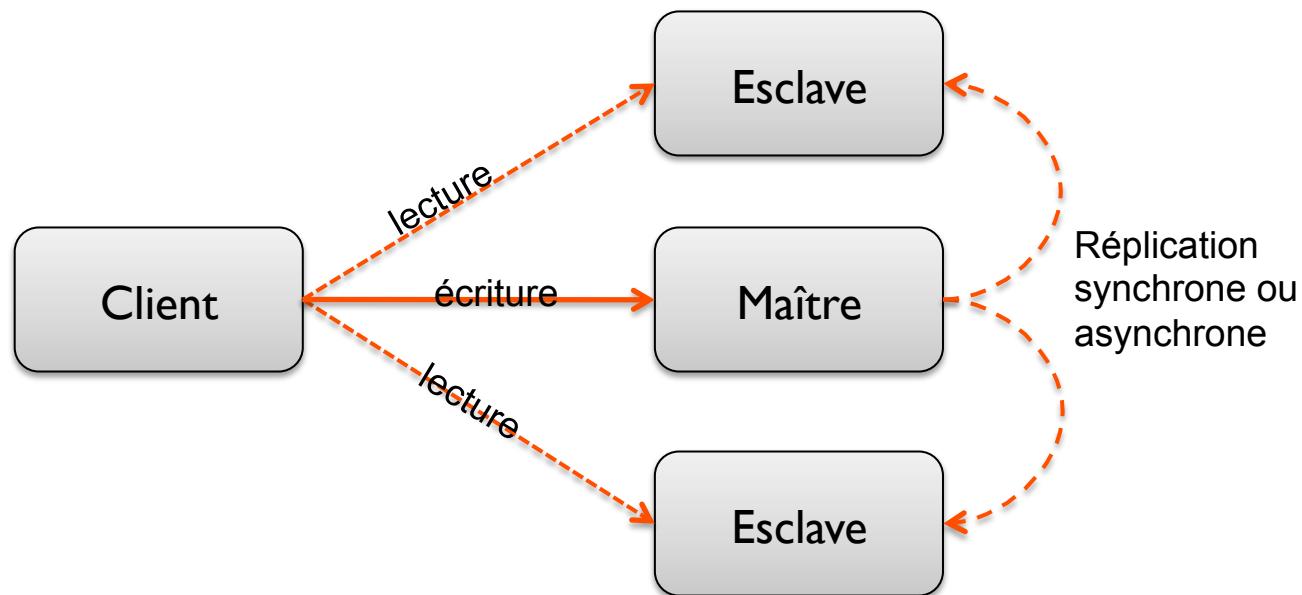
## ■ Maître/esclave

- Toutes les écritures sont envoyées au maître
- Les lectures peuvent être incorrectes tant que les écritures n'ont pas été propagées
- La volumétrie des données peut poser des problèmes si le maître doit dupliquer les données aux esclaves

## ■ Partitionnement

- Efficaces pour lectures et écritures
- Manque de transparence, les applications doivent s'adapter au partitionnement
- Perte des contraintes référentielles (relations) et jointure entre les partitions

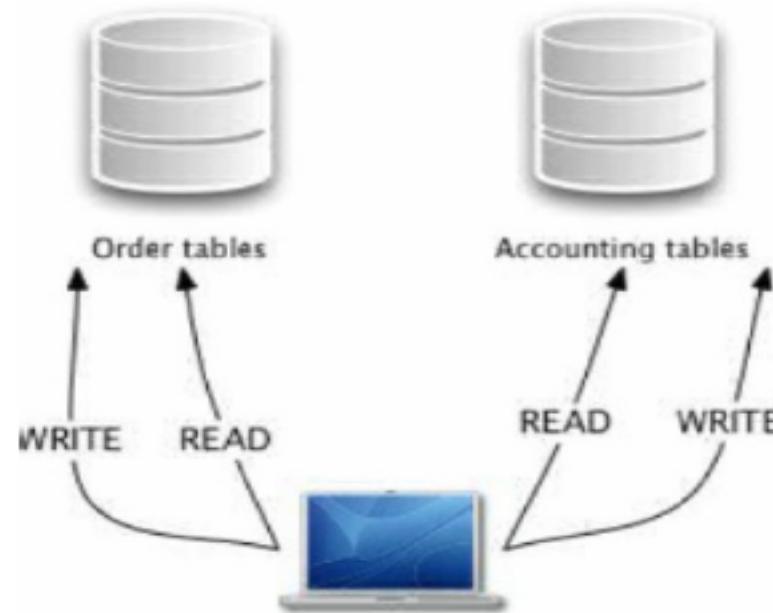
# Maître/esclave



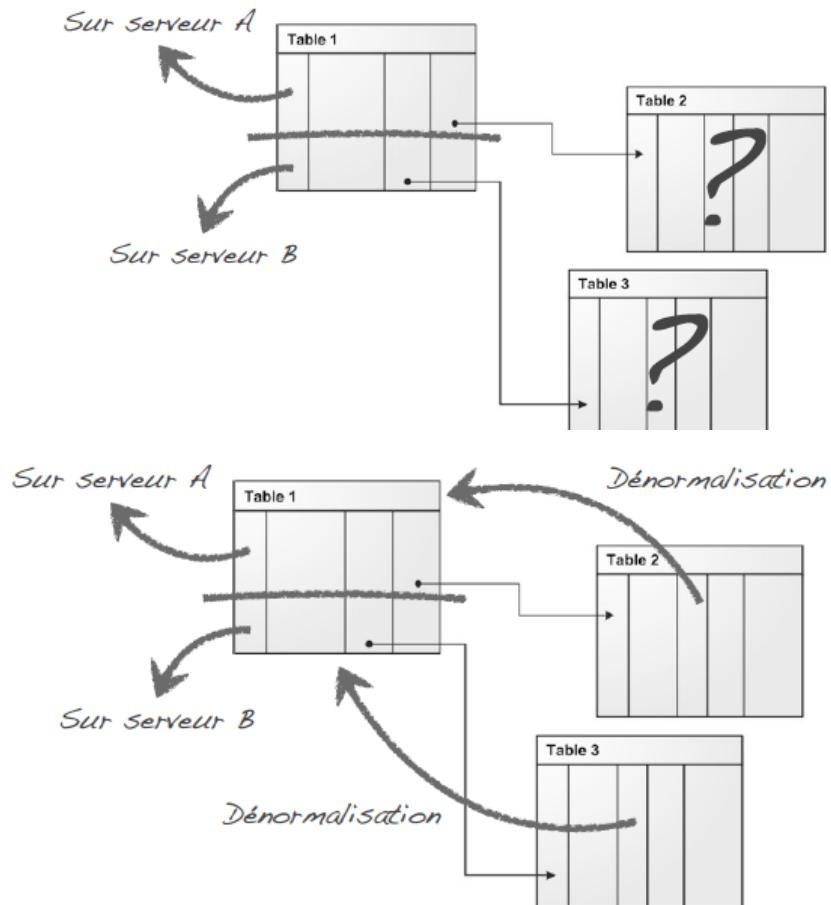
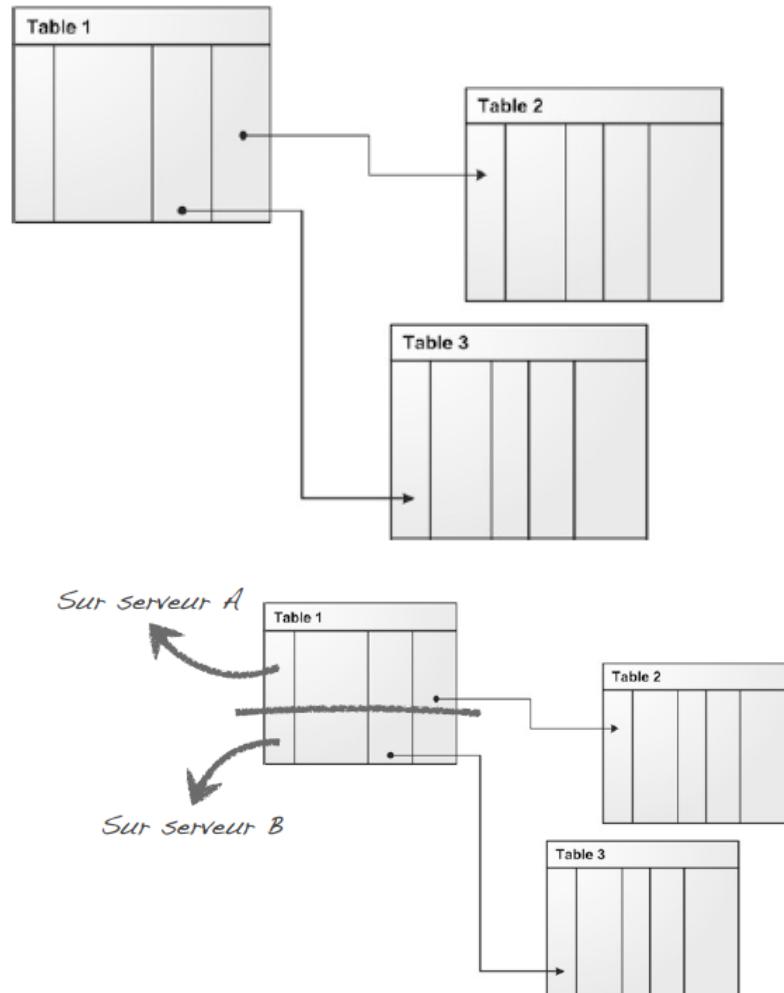
# Partitionnement vertical

## ▪ Vertical

- Les serveurs stockent différentes tables d'une base de données



# Partitionnement horizontal



On perd alors beaucoup de l'intérêt du relationnel !

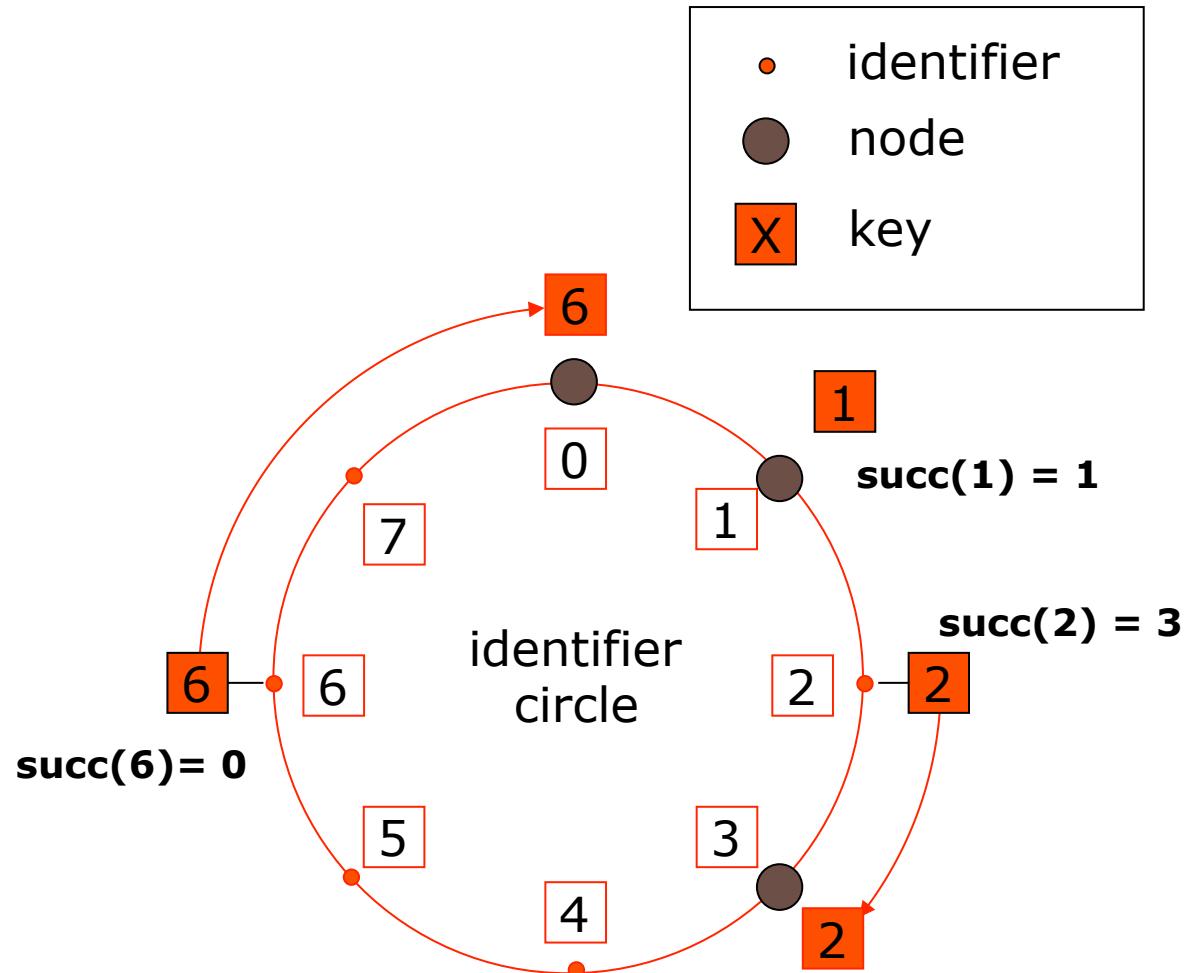
# Partitionnement horizontal

- **Chaque serveur est responsable d'un sous ensemble de données (identifié par un intervalle de clés)**
- **Ajout d'une nouvelle machine**
  - comment répartir la charge (load balancing) ?
- **Localisation des données**
  - Distributed Hash Tables (**DHT**)

# DHT

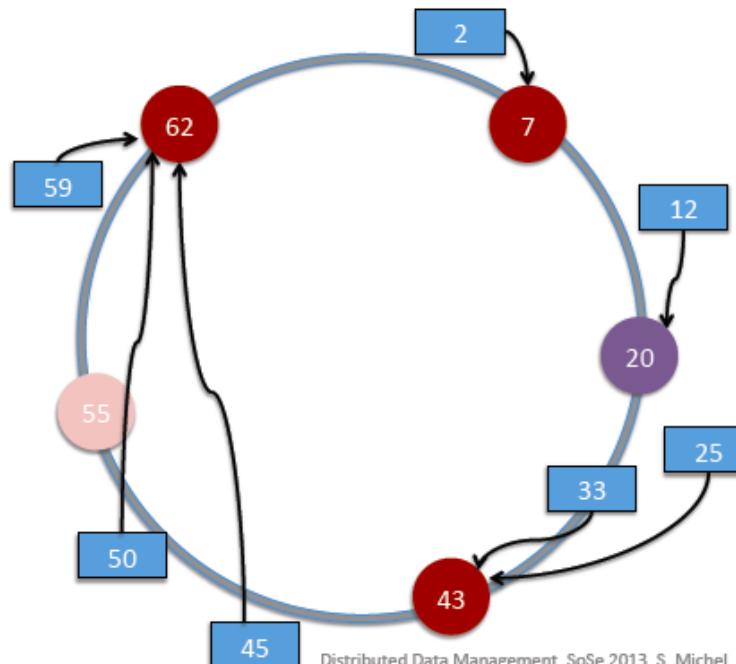
- **Mécanisme distribué et décentralisé permettant d' associer des valeurs de clés à un contenu par hachage de la clé**
  - Chaque participant gère une partie de la table de hachage.
  - Performance en recherche de l' ordre de  $\log(N)$
  - Facilement reconfigurable lorsqu' un site se connecte ou se déconnecte
- **Mécanisme général pour la localisation de ressources distribuées identifiées par des clés**
  - Les clés et les noeuds (adresse IP) sont hachés sur le même anneau ( cercle)
  - Clés et pairs sont affectés à un identifiant ds  $[0..m]$
  - Succ(kid) : le plus petit identifiant de noeud qui soit supérieur ou égal à  $k \bmod m$
  - Pred(kid) : le plus grand identifiant de noeud inférieur à  $k \bmod m$
- **Chaque clé d' identifiant k est gérée par le noeud suivant ou égal, i.e., d' identifiant supérieur ou égal**
  - $\text{Pred}(\text{Succ}(k)) < \text{kid} \leq \text{Succ}(\text{kid})$
- **Fonction, de hachage consistante comme MD5, SHA1 pour éviter les collisions et bien équilibrer la charge**

## Exemple (1)



## Exemple (2)

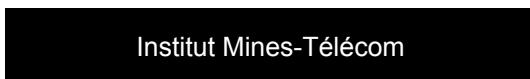
Removed Server (id 55)



Distributed Data Management, SoSe 2013, S. Michel



# MapReduce





# Motivation: traitement de données à grande échelle



- Traiter de grands volumes de données ( $> 1 \text{ To}$ )
- Paralléliser les traitements sur des centaines / milliers de processeurs

... Et tout cela de façon très simplifiée

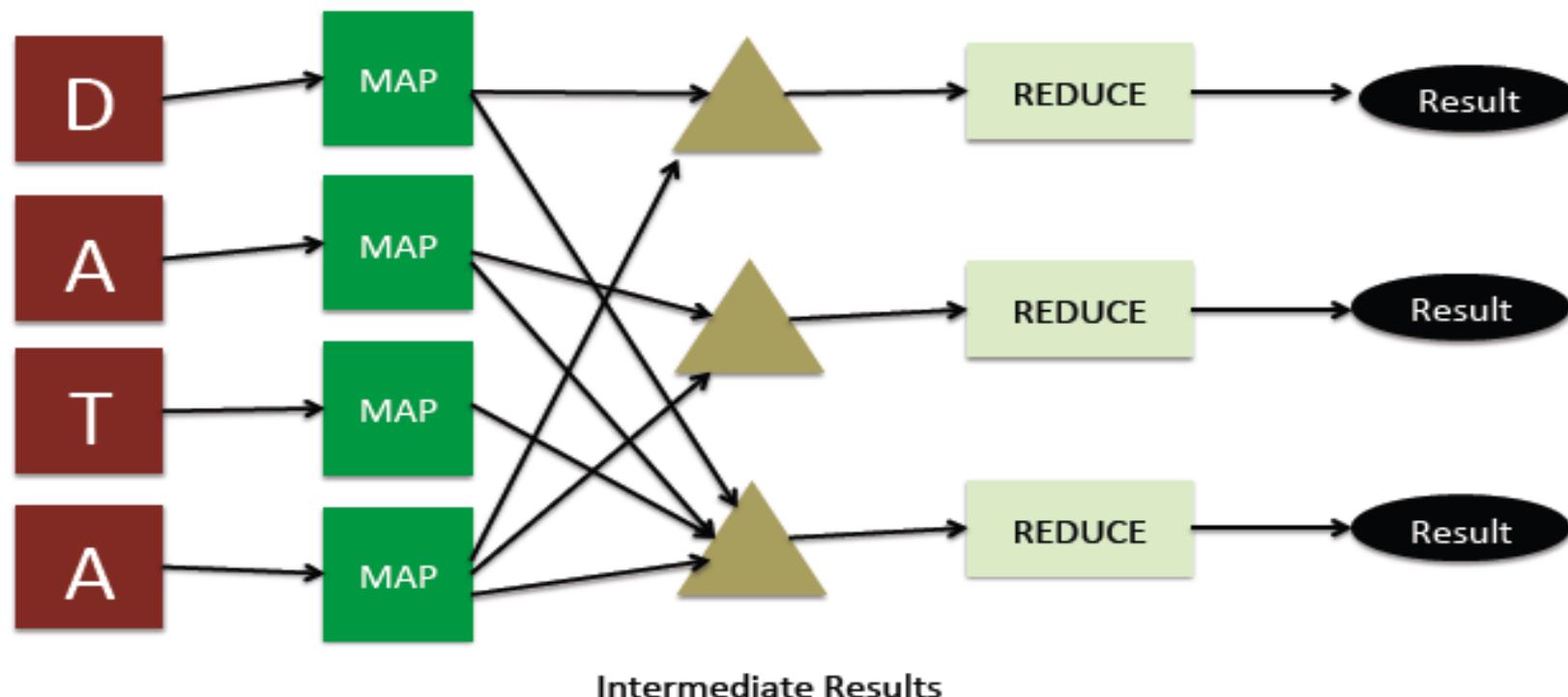
# MapReduce

- **Introduit par Google en 2004**
- **But: faire des traitements parallèles (indexation, datamining, ...)**
- **Déplacer les traitements vers l'infrastructure de stockage**
- **MapReduce Offre:**
  - **Parallélisation et distribution automatique des traitements**
  - **Tolérance aux pannes**
  - **Equilibrage de charge**
  - **Abstraction propre pour les programmeurs**

# Modèle de programmation

- **Emprunté à la programmation fonctionnelle**
- **Les programmeurs implémentent deux méthodes:**
  - **Map**: est une étape de transformation des données sous la forme de paires clé/valeur
  - **Reduce**: est une étape de fusion des enregistrements par clé pour former le résultat final

# MapReduce (architecture globale)



Distributed Data Management, SoSe 2013, S. Michel

# Map

- **map (in\_key, in\_value) -> (out\_key, intermediate\_value) list**
- **Exemples:**

- Majuscules

```
let map(k, v) = emit(k.toUpperCase(), v.toUpperCase())
("foo", "bar") -> ("FOO", "BAR")
("Foo", "other") -> ("FOO", "OTHER")
("key2", "data") -> ("KEY2", "DATA")
```

- Changement de clé

```
let map(k, v) = emit(v.length(), v)
("hi", "test") -> (4, "test")
("x", "quux") -> (4, "quux")
("y", "abracadabra") -> (10, "abracadabra")
```

## Reduce

- **reduce (out\_key, intermediate\_value list) -> out\_value list**
- **Après la fin de map(), toutes les valeurs intermédiaires pour une clé de sortie sont regroupées dans une liste**
- **reduce() combine les valeurs intermédiaires dans une ou plusieurs valeurs finales pour chaque clé de sortie**
- **Exemple:**

```
let reduce(k, vals) =  
    sum = 0  
    foreach int v in vals:  
        sum += v  
    emit(k, sum)
```

(“A”, [42, 100, 312]) -> (“A”, 454)  
(“B”, [12, 6, -2]) -> (“B”, 16)

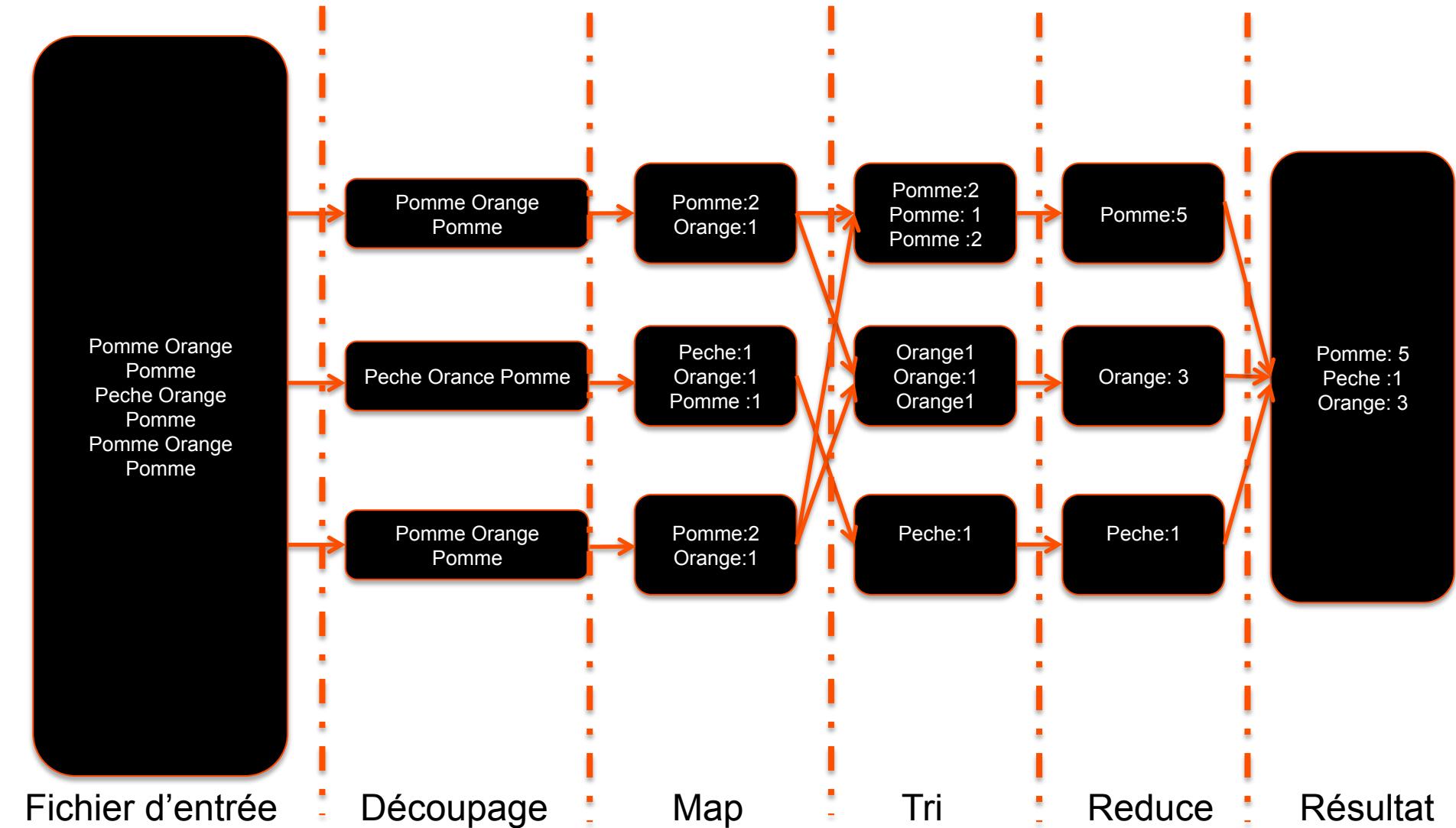
# Map and Reduce

- **Map (k1,v1) → list(k2,v2)**
- **Reduce(k2,list(v2)) → list(k3,v3)**

Les clés permettent de regrouper les données et les machines/tâches

- **Par exemple:**
  - k1= identifiant d'un document
  - v1= contenu du document
  - k2= mot
  - v2= occurrence
  - k3=mot
  - v3= occurrence finale

# Cas d'utilisation – Word Count



## Exemple -- Word Count

- **Programme Map et reduce**

map (key, value):

// key: document name; value: text of document

for each word w in value: emit(w, 1)

reduce (key, values):

// key: a word; values: the count of the word

result = 0

for each v in values: result += v

emit (key, result)

# Traitement vers les données

- **Données stockées dans des systèmes de fichier distribués**
  - Ex. Google File System, Hadoop Distributed File System
- **Plusieurs blocks (chunks généralement de 64Mo)**
- **Noeud maître (master node) connaît les localités des données**
  - Reçoit les jobs
  - Calcule le nombre de map et reduce nécessaires
  - Sélectionne et active les nœuds esclaves
- **Noeud maître envoie les traitements vers les nœuds esclaves (worker nodes)**

# MapReduce

- **Applications**

- Indexation de document dans un moteur de recherches, grep distribué, tri distribué, statistiques d'accès au web, construction d'index inversé, clustering de documents, etc.

- **Plusieurs implémentations: C#, C++, Erlang, Java, Python, etc.**

- **La plus connue:**

- Apache Hadoop Map Reduce



# Hadoop



- **Projet Open Source (Apache Software Foundation)**
- **Environnement d'exécution distribué**
- **Java**
- **Sous-projets: MapReduce, HDFS, Hbase, Pig, Hive, Mahout, etc.**
- **Grande volumétrie de données**
  - Exemple de Yahoo: 500GB de données triées en 59s avec un cluster de 1400 nœuds.
- **Applications:**
  - LastFm: statistiques hebdomadaires (Top artistes, Top Titres)
  - Facebook: reporting et statistiques internes (croissance du nombre d'utilisateurs, consultation des pages, etc.)
- **<http://wiki.apache.org/hadoop/PoweredBy>**
  - Amazon, Apple, Ebay, IBM, Google, Microsoft, SAP, Twitter, etc

## Use cases

- **Facebook** has multiple Hadoop clusters deployed now - with the biggest having about 2500 cpu cores and 1 PetaByte of disk space. We are loading over 250 gigabytes of compressed data (over 2 terabytes uncompressed) into the Hadoop file system every day and have hundreds of jobs running each day against these data sets. (source: [https://www.facebook.com/note.php?note\\_id=16121578919](https://www.facebook.com/note.php?note_id=16121578919))
- **Google** : the size of one phase of the computation [of the index] dropped from approximately 3800 line of C++ code to approximately 700 lines when expressed using MapReduce. (source: The art of rails, Edward Benson, p38)
- **The Yahoo ! Search Webmap** is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo ! Web search query. (source: [en.wikipedia.org](http://en.wikipedia.org))
- **Other users:** Amazon, ebay, Netflix, Spotify, IBM, LinkedIn, etc.

# Implémentation des opérations d'algèbre relationnel en MapReduce



## ■ Sélection:

- Map retourne l'enregistrement s'il correspond aux critères de sélection
- Pas de Reduce

## ■ Jointure:

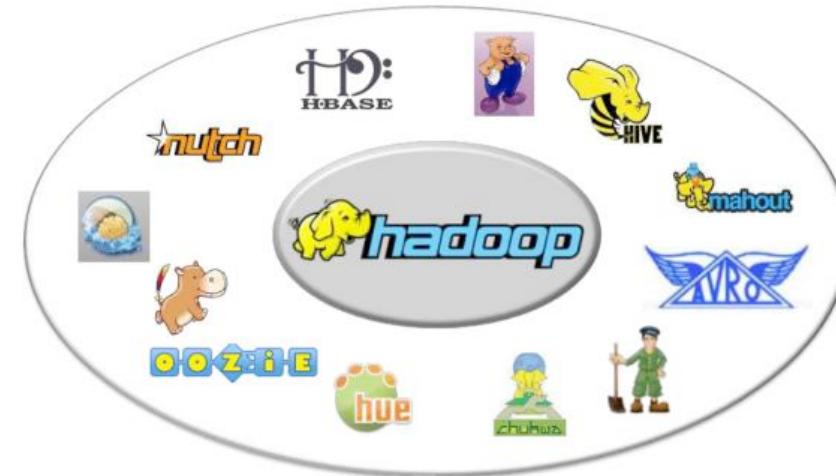
- Map renvoie une paire intermédiaire composée du champ de jointure en clé et l'enregistrement en valeur
- Reduce reçoit pour chaque clef de jointure la liste des enregistrements correspondants et se charge de faire le produit cartésien de ces enregistrements

## MapReduce: conclusions

- **MapReduce s'est avéré être une abstraction très utile**
- **Simplifie grandement les calculs à grande échelle chez les grands acteurs du web (Google, facebook, etc.)**
- **Paradigme de programmation fonctionnelle peut être appliqué à des applications à grande échelle**
- **Fun & Simple: on ne se concentre que sur le problème et on laisse la librairie gérer le « sale boulot » (réPLICATION, tolérance aux fautes, répartition de charge, etc.)**

# Ecosystème Hadoop

- Avro: Serialisation données
- Chukwa: supervision de clusters
- Flume: collecte de données log en temps réel
- Hbase: BD Colonnes
- Hive: Requêtage sur les données (proche de SQL)
- Pig: Langage de requêtage (scripting)
- Sqoop: transfert de données entre SGDBDR et hadoop
- Oozie: Workflow et orchestration de jobs
- Hue: Interface graphique pour Hadoop
- ...



# Problèmes de MapReduce

- **La simplicité est le principe du MapReduce, tant que l'application ne nécessite pas des requêtes complexes SQL**
- **Opérations classiques de bases de données doivent être codées à la main**
  - Join, selection, projection, etc.
- **Solution: Utilisation d'un langage de haut niveau**
  - Permet de traduire le langage de haut niveau en mapReduce de façon automatique
  - Pig, Hive, etc.

- **Implémentation commencée par Yahoo! Research**
- **Execute plus de 30% des jobs Yahoo!**
- **Caractéristiques**
  - Exprime des jobs mapReduce
  - Fournit des opérateurs relationnels (SQL)
    - (JOIN, GROUP BY, etc.)
  - Facilement intégrables avec des programmes en java

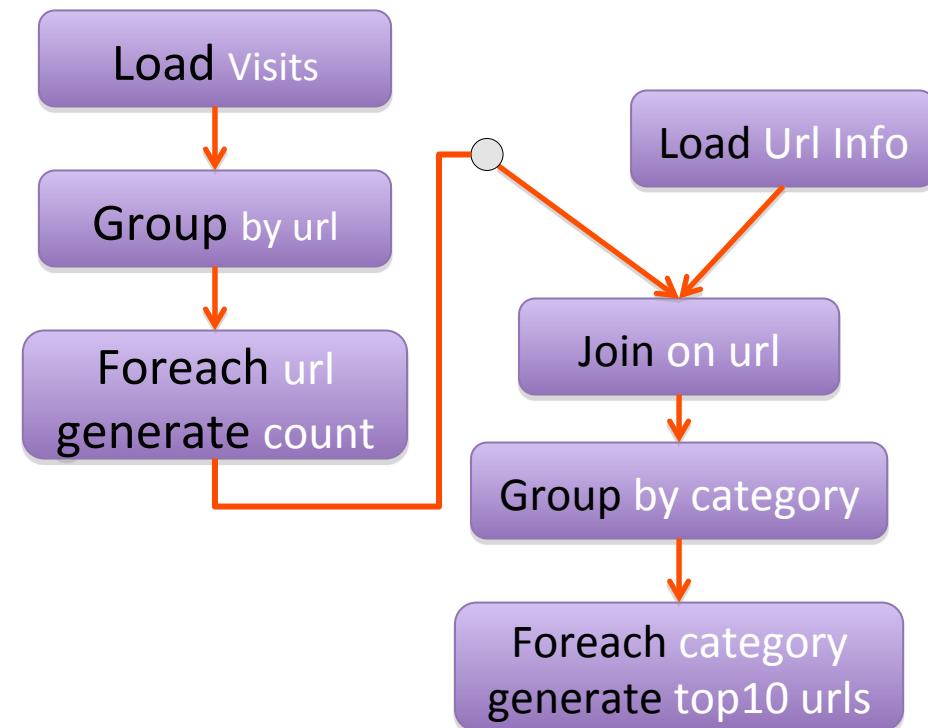


## Pig - exemple

- Soient deux tables: Visits, url-Info
- Trouver les top 10 des pages les plus visitées dans chaque catégorie

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9



# En MapReduce

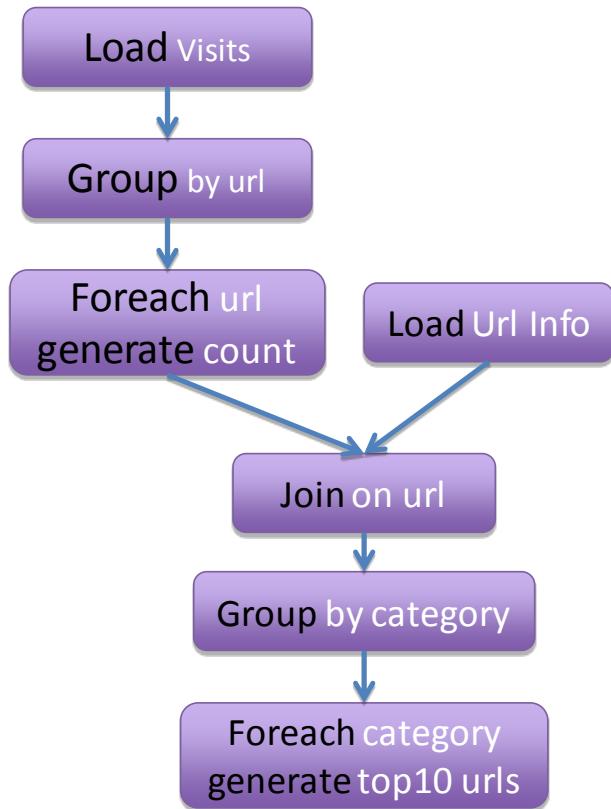
```

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.mapred.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;
public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {
        public void map(LongWritable k, Text val,
                       OutputCollector<Text, Text> oc,
                       Reporter reporter) throws IOException {
            // Pull the key out
            String line = Val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Friend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {
        public void map(LongWritable k, Text val,
                       OutputCollector<Text, Text> oc,
                       Reporter reporter) throws IOException {
            // Pull the key out
            String line = Val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Friend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text> {
        public void reduce(Text key,
                           Iterator<Text> iter,
                           OutputCollector<Text, Text> oc,
                           Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
            store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();
            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
                    first.add(value.substring(1));
                else second.add(value.substring(1));
            }
            first.add(val.toString());
            second.add(val.toString());
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {
        public void map(
                        Text k,
                        Text val,
                        OutputCollector<Text, LongWritable> oc,
                        Reporter reporter) throws IOException {
            // Find the url
            String line = Val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            String value = line.substring(secondComma + 1);
            Text outKey = new Text(key);
            // Add up all for the combiner/reducer to sum instead.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {
        public void reduce(
                        Text key,
                        Iterator<LongWritable> iter,
                        OutputCollector<WritableComparable, Writable> oc,
                        Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
            }
            reporter.setStatus("OK");
        }
        oc.collect(key, new LongWritable(sum));
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {
        public void map(
                        WritableComparable key,
                        Writable val,
                        OutputCollector<LongWritable, Text> oc,
                        Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {
        int count = 0;
        public void reduce(
                        LongWritable key,
                        Iterator<Text> iter,
                        OutputCollector<LongWritable, Text> oc,
                        Reporter reporter) throws IOException {
            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(MapReduceBase.class);
        lp.setCombinerClass(LoadAndFilterUsers.class);
        lp.setReducerClass(Join.class);
        lp.setOutputFormat(TextOutputFormat.class);
        Path("/user/gates/pages");
        FileInputFormat.addInputPath(lp, new
        Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
        new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);
        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setMapperClass(MapReduceBase.class);
        lfu.setCombinerClass(LoadAndFilterUsers.class);
        lfu.setReducerClass(Join.class);
        lfu.setOutputFormat(TextOutputFormat.class);
        Path("/user/gates/users");
        FileOutputFormat.setOutputPath(lfu,
        new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);
        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setMapperClass(MapReduceBase.class);
        join.setCombinerClass(Join.class);
        join.setReducerClass(IdentityMapper.class);
        join.setOutputFormat(TextOutputFormat.class);
        join.setOutputFormat(TextOutputFormat.class);
        Path("/user/gates/tmp/joined");
        FileInputFormat.addInputPath(join, new
        Path("/user/gates/users"));
        FileInputFormat.addInputPath(join, new
        Path("/user/gates/tmp/indexed_pages"));
        FileOutputFormat.setOutputPath(join, new
        Path("/user/gates/tmp/joined"));
        join.setMapperClass(MapReduceBase.class);
        join.setCombinerClass(ReduceUrls.class);
        join.setReducerClass(ReduceUrls.class);
        join.setOutputFormat(TextOutputFormat.class);
        join.setOutputFormat(TextOutputFormat.class);
        Path("/user/gates/tmp/grouped");
        FileOutputFormat.setOutputPath(group, new
        Path("/user/gates/tmp/grouped"));
        Path("/user/gates/tmp/grouped");
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);
        Job top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 10 sites");
        top100.setMapperClass(MapReduceBase.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        top100.setOutputFormat(TextOutputFormat.class);
        top100.setOutputFormat(TextOutputFormat.class);
        top100.setMapperClass(LimitClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
        Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
        Path("/user/gates/top10sitesforusers1to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);
        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}

```

# En Pig

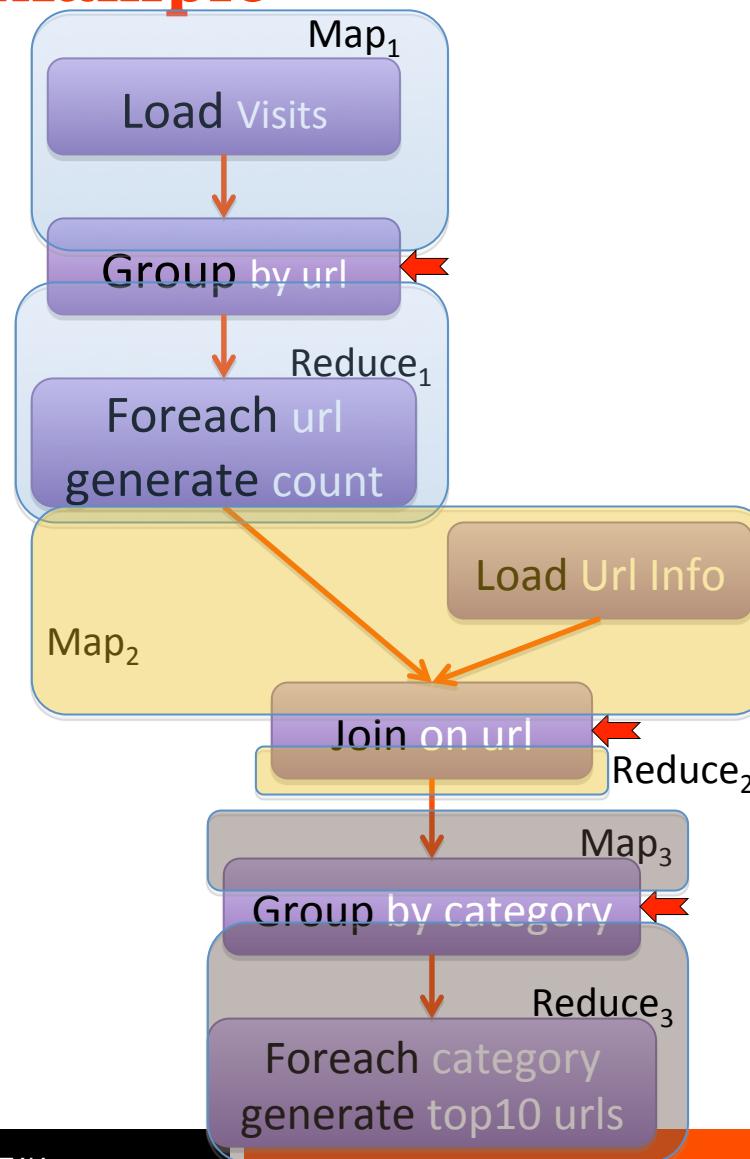


```

visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits
              generate url, count(visits);
urlInfo = load '/data/urlInfo' as
          (url, category, pRank);
visitCounts = join visitCounts by url,
                  urlInfo by url;
gCategories = group visitCounts by
               category;
topUrls = foreach gCategories
            generate top(visitCounts,10);
store topUrls into '/data/topUrls';
  
```

\* top, count sont des fonctions définies par l'utilisateur

# Pig by Example

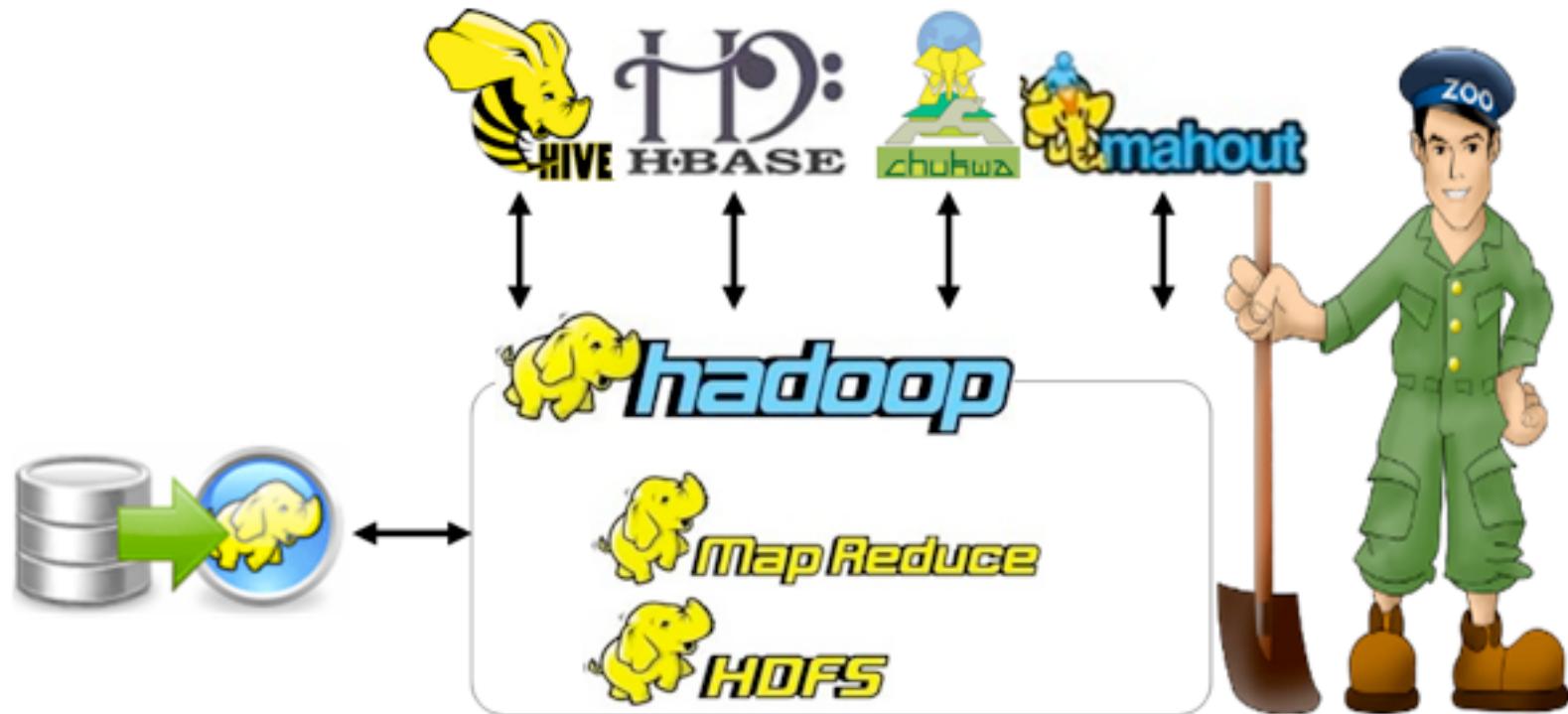


## Hive

- **Un entrepôt de données pour Hadoop**
- **Open-Source**
- **Écrit en java**
- **Les métadonnées sont stockées dans des bases de données relationnelles**
- **Offre un langage de requêtage proche de SQL:  
HiveQL**
- **Offre la possibilité de définir des fonctions**
- **Offre une indexation**



# Architecture - Hadoop



## Architecture - Hive

- **Au dessus de HDFS et Cluster Hadoop**
- **Utiliser Flume et Sqoop pour migrer les données dans Hive**
- **Utiliser la commande Hive « LOAD DATA »**
- **Utiliser ODBC pour se connecter à l'architecture BI**

# Langages - LDD

- **Creation d'une table**

```
hive> CREATE TABLE customer (age INT, address STRING)
```

- **Voir les tables**

```
hive> SHOW TABLES;
```

- **Décrire les tables**

```
hive> DESCRIBE customer;
```

- **Modification d'une table**

```
hive> ALTER TABLE customer ADD COLUMNS (age INT);
```

- **Décrire les tables**

```
hive> DROP TABLE customer;
```

## Langages – LMD

- **Charger un fichier dans Hive**

```
hive> LOAD DATA LOCAL INPATH '/data/home/test.txt' OVERWRITE  
INTO TABLE customer;
```

- **HIVEQL**

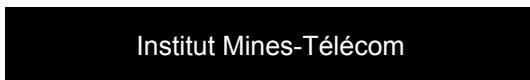
```
hive> SELECT c.age FROM customer c WHERE c.sdate='2008-08-15'  
hive>INSERT OVERWRITE DIRECTORY '/data/hdfs_file' SELECT c.*  
FROM customer c WHERE c.sdate='2008-08-15';
```

- **Ecrit tout le contenu de la table customer dans un répertoire hdfs**

- **Peut s'utiliser avec ODBC pour s'interfacer avec des outils de Business Intelligence**

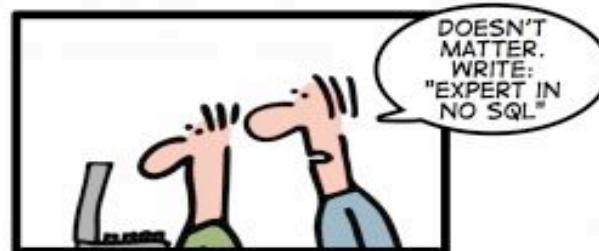


# NoSQL





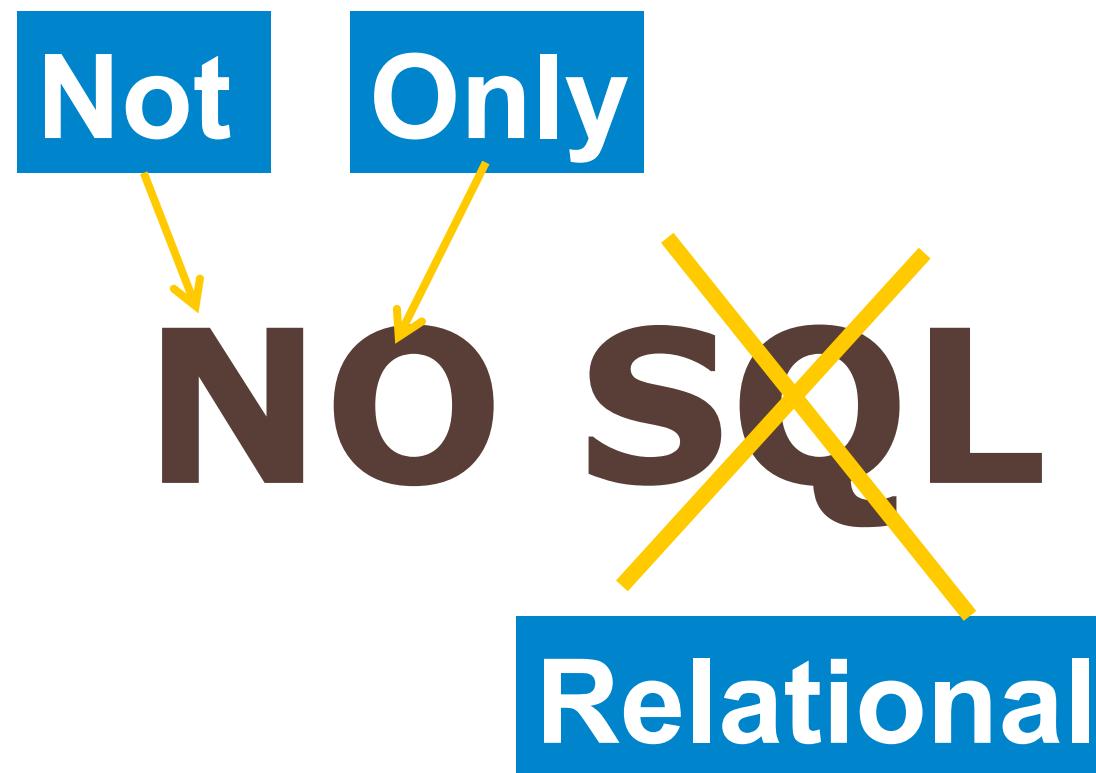
## HOW TO WRITE A CV



Leverage the NoSQL boom



# Fausses et vraies idées autour du NoSQL



# NoSQL?

- **No SQL => Not Only SQL**
- **SQL ne doit pas mourir mais des solutions de stockage doivent être envisagées pour des applications particulières (applications web en particulier)**
- **Nom exact: Bases de données non relationnelles**
- **Le modèle ACID ne permet pas un passage à l'échelle dans un environnement distribué, limitant par exemple les débits en écriture (les plus coûteux)**
  - Atomicity
  - Consistency
  - Isolation
  - Durability



# RDBMS

- **MySQL, PostgreSQL, SQLite, Oracle, etc.**
- **Besoins:**

- Schémas
- Cohérence forte
- Transactions
- Matures et opérationnelles
- Expertise disponible

# Quand utiliser NoSQL

- **Grand volume de données**
  - Besoin d'utiliser une architecture fortement distribuée
  - Google, Amazon, Yahoo, Facebook -10-100K serveurs
- **Requêtes massives**
  - Eviter les jointures car très gourmandes en temps de traitement
- **Schéma évolutif**
  - Flexibilité et évolutivité du schéma à grande échelle

## Nouveaux besoins

- **Applications web: explosion des données et des requêtes**
- **Latence: temps de réponse faible et prévisible**
- **Passage à l'échelle et élasticité**
- **Haute disponibilité**
- **Schémas flexibles / données semi-structurées**
- **Distribution géographique (multitude de datacenters)**
- **Pas besoin de : transaction/ forte cohérence/ intégrité/ requêtes complexes**

## Exemples d'applications

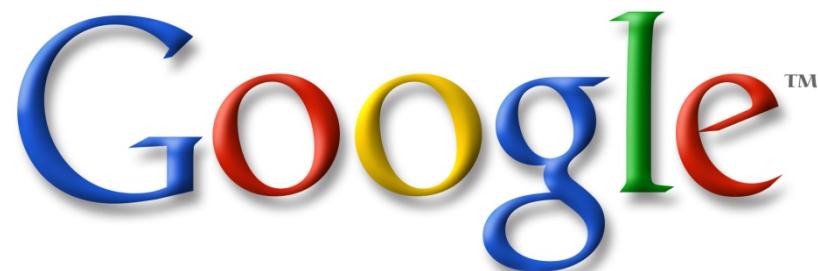
- **Recherche des utilisateurs: stockage des mots clés**
- **Alertes (ex: actualités, changement de prix, etc.)**
- **Activités dans les réseaux sociaux (connexion/déconnexion, messages postés, chargement de photos, tags, etc.)**
- **Traces des utilisateurs: clics (articles, publicités, etc.)**



YAHOO!®

TELECOM  
Evolution  


source  
forge

The logo for Facebook consists of the word "facebook" in a white, rounded sans-serif font, centered on a solid blue rectangular background.The Google logo features the word "Google" in its signature multi-colored, rounded font. The letters are colored blue, red, yellow, and green, with a trademark symbol (TM) at the top right.The LinkedIn logo consists of the word "Linked" in a large, bold, black sans-serif font, followed by the word "in" in a white sans-serif font inside a blue rounded square.The Twitter logo is the word "twitter" in a light blue, rounded font, with each letter having a thin white outline.The Ubuntu One logo features the word "ubuntu" in a black sans-serif font above the word "one" in a smaller, orange sans-serif font.The Amazon.com logo consists of the word "amazon.com" in a large, bold, black sans-serif font, with a yellow curved arrow underneath the "a". Below the main text is the tagline "and you're done." in a smaller, black sans-serif font.The S-Télécom logo features the letters "s-Télécom" in a white sans-serif font inside a black rectangular box.The ISEP logo consists of the letters "isep" in a blue sans-serif font next to a small blue square icon with a white graphic.

# NoSQL

## ■ Avantages

- Passage à l'échelle
- Haute disponibilité
- Elasticité
- Flexibilité du schéma
- Gestion des données éparses (NULL) et semi-structurées
- Evolutivité horizontale

## ■ Inconvénients

- Pas de standards
- Contrôle d'accès insuffisant
- Requête limité

• Applications client compliquées à cause de « Eventual Consistency »

# Théorème du CAP (E.Brewer, N. Lynch 2000)



- **Trois propriétés d'un système: la cohérence (Consistency), la disponibilité (Availability) et la partition (Partition tolerance)**
  - Consistency : l'ensemble des clients voient la même donnée, même après des mises à jour
  - Availability : tous les clients peuvent trouver de la donnée répliquée, même lors d'un crash
  - Partition tolerance : la base de données est tolérante au partitionnement
- **Au maximum 2 propriétés respectées par n'importe quel système de données distribué**
- **Pour passer à l'échelle, on utilise le partitionnement. Ce qui implique un choix entre la cohérence ou la disponibilité**



**Merci pour votre attention**

Raja.chiky@isep.fr

