

Bases de données NoSQL

Raja CHIKY
raja.chiky@isep.fr
2015-2016

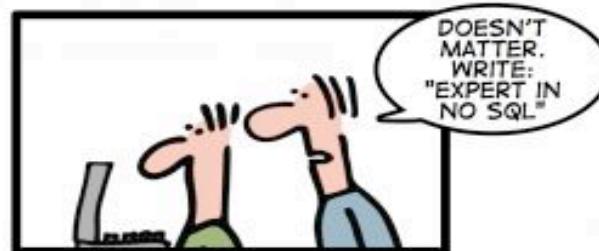


Plan

- **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**



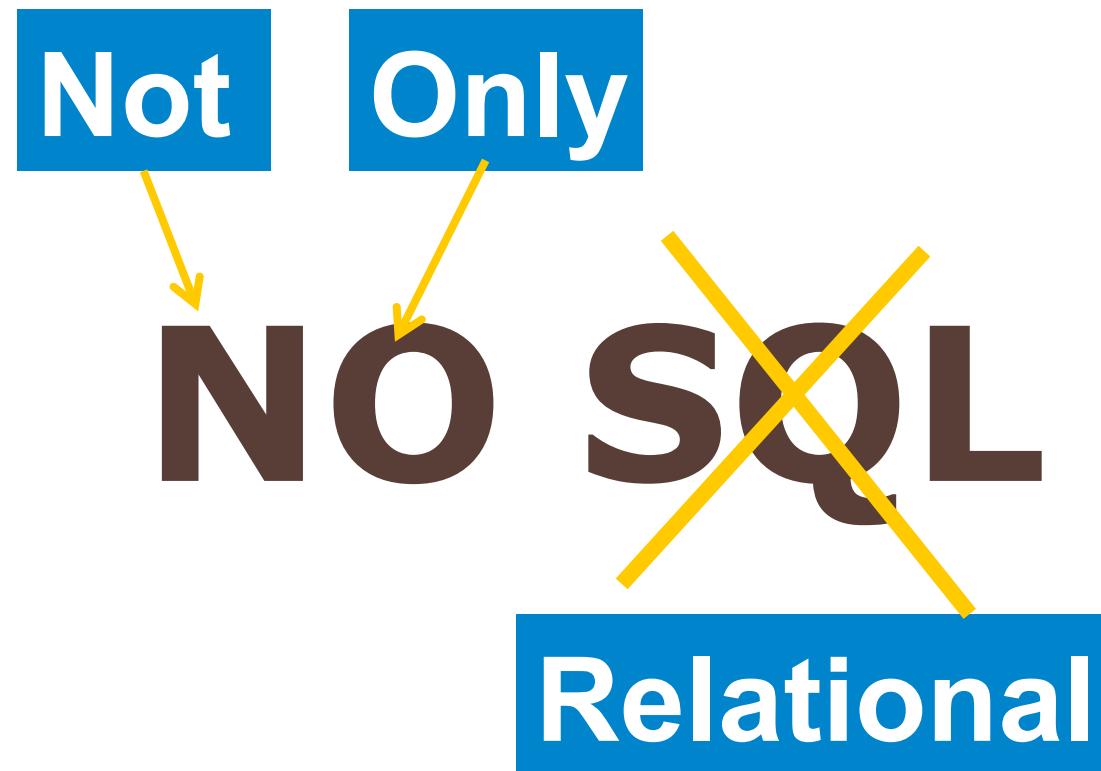
HOW TO WRITE A CV



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**

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 »

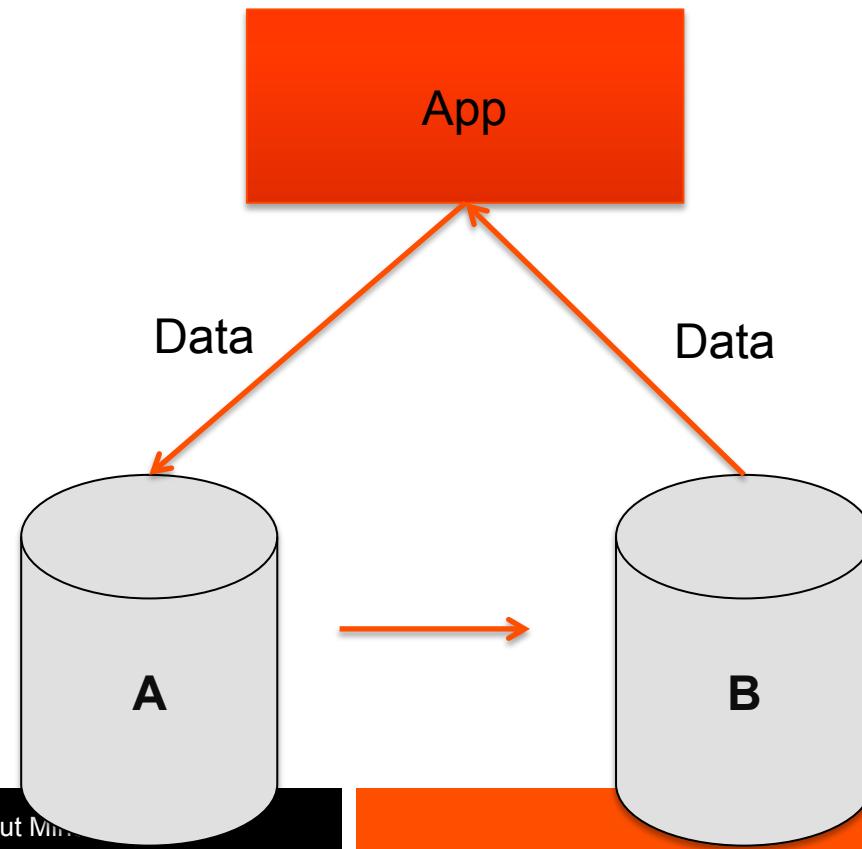
Théorème du CAP (E.Brewer, N. Lynch 2000)



- **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**

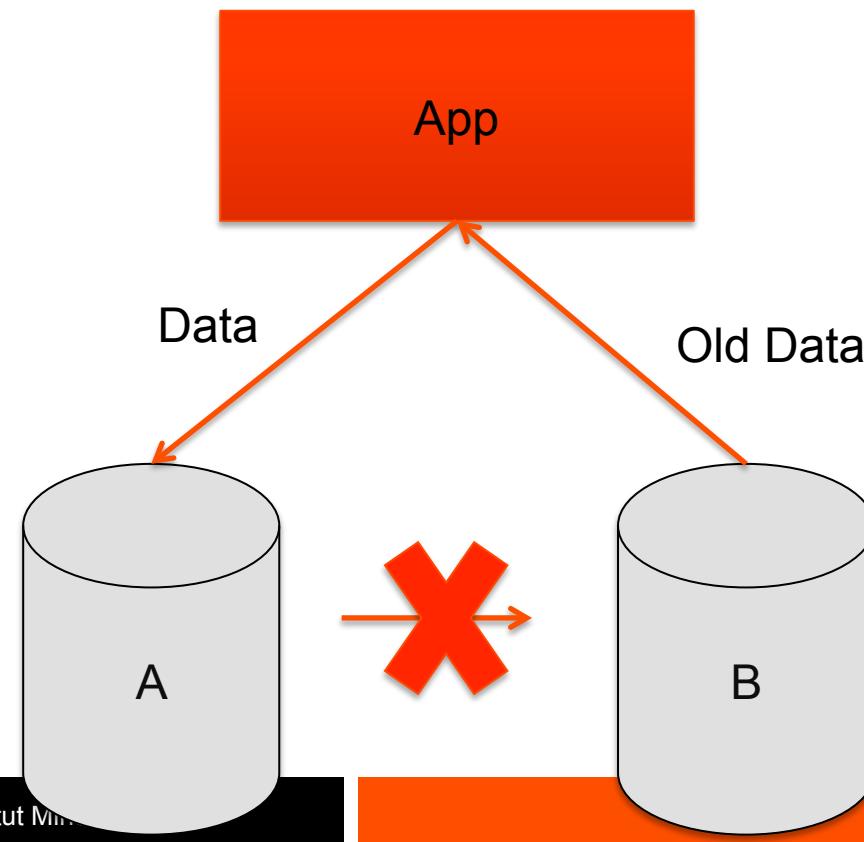
Une preuve simple (1)

Pas de partitionnement,
Cohérence et disponibilité



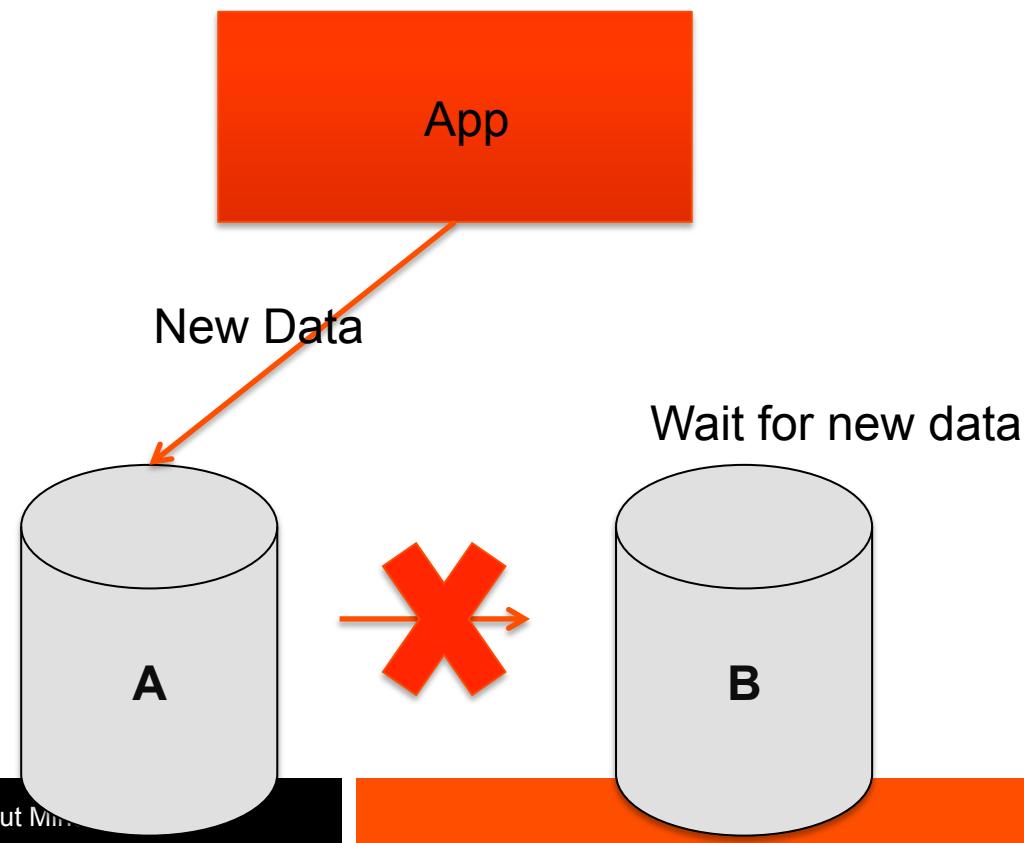
Une preuve simple (2)

Partitionnement et disponibilité
Incohérence



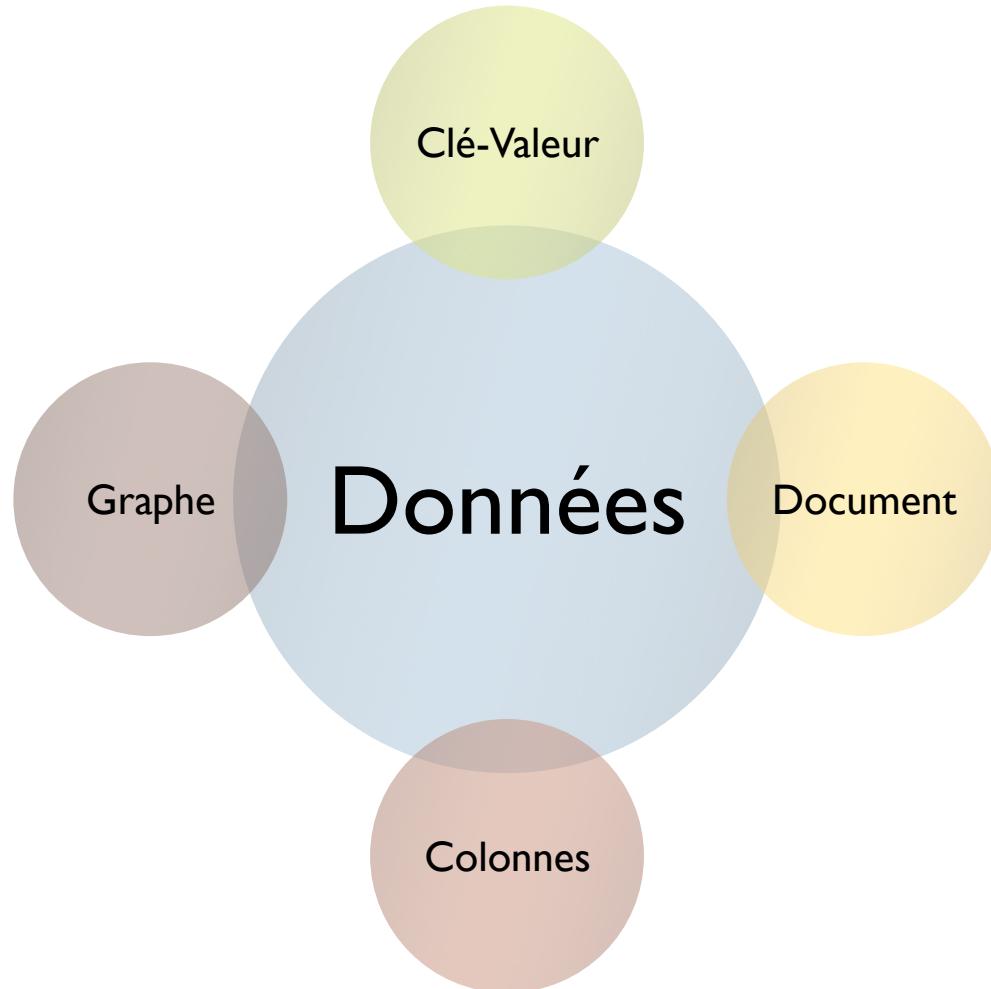
Une preuve simple (3)

Partitionnement, Cohérence
Pas disponible

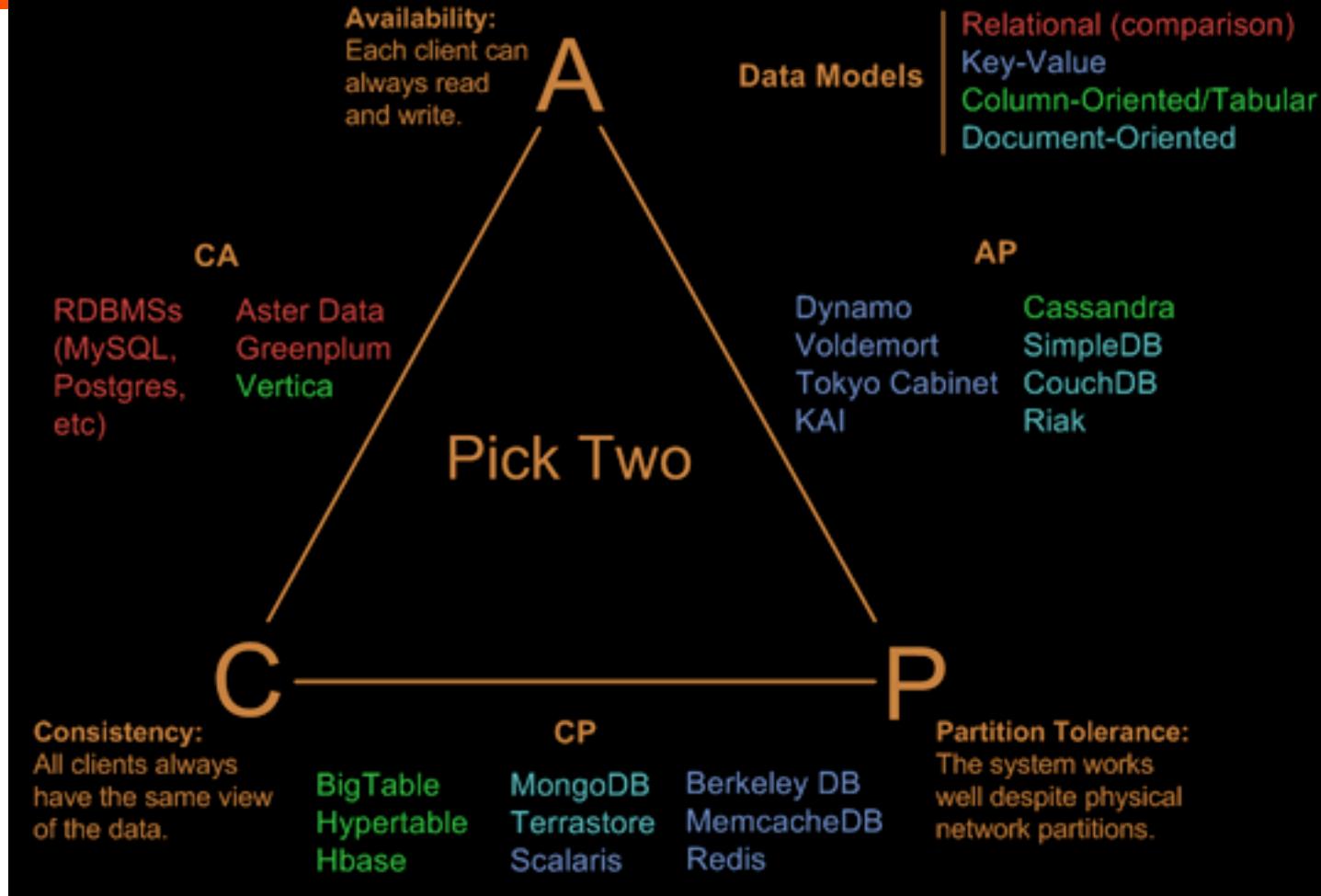




Taxonomie NoSQL



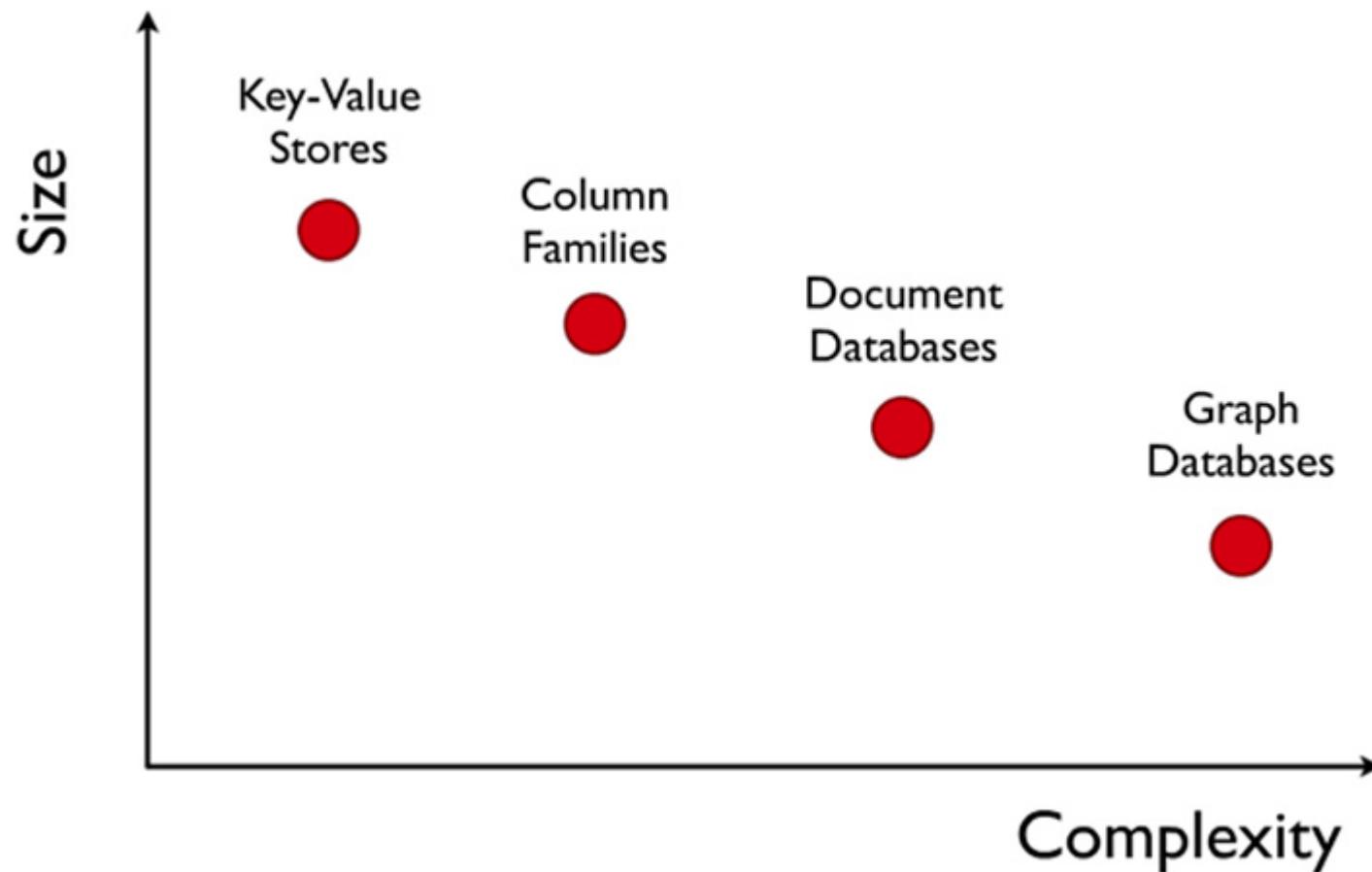
Visual Guide to NoSQL Systems



Visual Guide to NoSQL Systems - Nathan Hurst's Blog
<http://blog.nahurst.com/visual-guide-to-nosql-systems>



Complexité



Comment peut-on faire des requêtes dans une BD NoSQL?



- **Interfaces REST**
 - HTTP via une API
- **Langages de requêtes basées sur SQL**
 - GQL – SQL-Like QL pour Google BigTable
 - CQL – Cassandra Query Language
 - SPARQL – langage de requête pour le Web sémantique
 - Gremlin, Cypher – langage de parcours d'un graphe
 - Sones Graph Query Language
- **API's de requêtage de données**
 - GAE (Google Application Engine) API
 - Thrift API
 - Etc.
- **Map reduce**
 - Pig, Hive, etc.

Rappels

- **SQL**

- Universalité « One size fits all »
- Logique : schémas, contraintes
- Cohérence forte
- Ressources limitées
- Langage standard: SQL
- Traitements centralisés

- ▶ **NoSQL**

- ▶ Systèmes sur mesure
- ▶ Pas de schémas ni contraintes
- ▶ Cohérence « à terme »
- ▶ Ressources « illimitées »
- ▶ Langages spécialisés, API
- ▶ Traitements distribués (mapReduce)

Limites du SGBDR



“ If the only tool you have is a hammer, you tend to see every problem as a nail.”

Abraham Maslow

Limites du SGBDR



© H. Blumhardt.com

If the only tool you have is a relational database,
everything looks like a table.

A Walk in Graph Databases - 2012



Bases de données clé-valeur

Dynamo, Redis, Voldemort, ...



Clé/Valeur



■ Avantages

- Très rapide
- Passage à l'échelle
- Modèle simple
- Distribution horizontale très facile



■ Inconvénients

- Plusieurs données ne peuvent être modélisées comme clé/valeur

Dynamo

- **Amazon, 2007**
- **Architecture P2P: DHT (Distributed Hash Table)**
 - Versioning des objets
 - Algorithme de « Gossiping » (détection des échecs et des adhésions)
 - Hachage cohérent
- **Exigences et hypothèses**
 - Modèle de requêtes simple (clés unique, blobs, pas de schéma)
 - Architecture décentralisée (P2P) et hétérogène
 - Faible latence / haut débit
 - Service interne (pas de modèle de sécurité)
 - Haute disponibilité
- **Dynamo est utilisé pour:**
 - Le panier des achats, préférences des utilisateurs, gestion des sessions, gestion des ventes, catalogue de produits

Résumé des techniques utilisées dans Dynamo et leurs avantages



Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

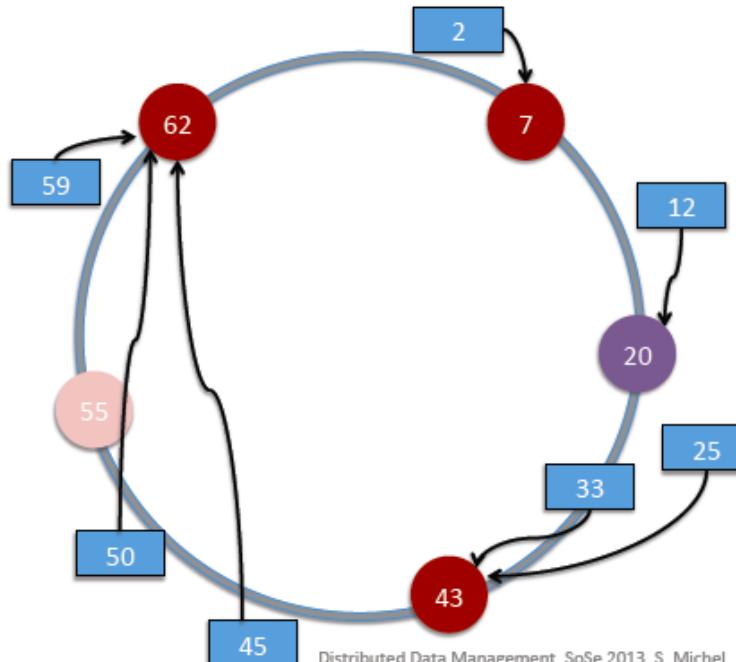
Partitionnement

- **Nœuds en cercle: position affectée aléatoirement aux nœuds**
- **Connexion/déconnexion d'un nœud n'affecte que ses 2 voisins**
- **RéPLICATION: N répliques (successeurs dans le cercle)**
- **Versioning:**
 - Stockage des mises à jour
 - Eventual consistency
 - Réconciliation des conflits effectuée par les clients (vector clock)



Exemple

Removed Server (id 55)



Distributed Data Management, SoSe 2013, S. Michel



Noeuds virtuels

- **Le problème avec les DHT**
 - Noeuds placés aléatoirement autour du cercle
 - Données non uniformément distribuées sur les noeuds
- **Dans Dynamo**
 - Utilise des noeuds virtuels
 - Chaque noeud physique dispose d'un certain nombre de noeuds virtuels, le nombre dépend de sa capacité
 - Les noeuds virtuels sont distribués autour du cercle



Interface de requête Dynamo

- **Seulement 2 opérations implémentées**
- **put (key, context, object)**
 - key: clé associée à un objet
 - context: vector clocks (historique)
 - object: l'objet à stocker
- **get (key)**

Autre système: REDIS

- **Remote Dictionary Server**
- **Projet open-source**
- **Redis stocke les données en mémoire (In memory DB)**
- **Possibilité de stocker des données sur disque (snapshot)**
- **Au démarrage du service, dernier snapshot mis en mémoire**
- **Très performant**
- **Utilisé par Twitter, Instagram, Stackoverflow, ask.fm, etc**
- **Cas d'utilisation: Préférences des utilisateurs, stockage des sessions, Statistiques et scores dans un jeu par exemple, ...**

Installation

```
>wget http://download.redis.io/redis-stable.tar.gz
>tar xvzf redis-stable.tar.gz
>cd redis-stable
>make

>redis-server
>redis-cli redis 127.0.0.1:6379
> ping
      PONG
>redis 127.0.0.1:6379> set mykey somevalue
      OK
>redis 127.0.0.1:6379> get mykey
      "somevalue"
```

Commandes utiles

- **Toutes les clés**
>KEYS *

- **Effacer toutes les clés**
>FLUSHALL

- **Sauvergarder les données sur disque**
>SAVE

- **Expiration**
>set a 1
>expire a 5
>get a
(...5secondes après)

- **Multi-get**
>set a 1
>set b 12
>mget a b

▶ Incrémentation

>SET MyVar 10
>GET MyVar
>INCR MyVar
>INCRBY MyVar 10

▶ Sauvegarde

>SAVE 30 1
- -sauvegarder après 30 secondes si au moins 1 clé est modifiée

Types de données: Strings

- Max 512 MO
-
- >SET counter 10
 - >INCRBY counter 100
 - >DECR counter
 - >APPEND counter 01
 - >GET counter
 - >INCR counter

Types de données: LIST

- Liste de « Strings » triés par ordre d'insertion
- Longueur max: $2^{32}-1$
- Quelques commandes:
**LPUSH, LRANGE,
RPUSH, LLEN**

```
>DEL myList  
>LPUSH myList un  
>LPUSH myList deux  
>LPUSH myList trois  
>LPUSH myList quatre  
>LRANGE myList 0 1
```

Types de données : SET

```
>SADD myset Hello
>SADD myset World
>SADD myset World
>SMEMBERS myset
>SISMEMBER myset World
>SISMEMBER myset Bonjour
```

Quelques commandes: SCARD, SDIFF,
SDIFFSTORE, SINTER, SINTERSTORE,
SMOVE,

Type de données: Hash

- Map entre une chaîne de caractère et sa valeur
- Par exemple: on veut stocker un utilisateur (son id, login, mdp, age)

```
>HSET user1 ID "001"
>HSET user1 login "pseudo1"
>HSET user1 mdp "azerty"
>HSET user1 age "35"
>HGETALL user1
>HMSET user2 ID "002" login "pseudo2"
>HGET user2 login
```

Types de données: Sorted Set

- Ensemble de données triées selon un score pré-défini

```
>ZADD myzset 1 "one"  
>ZADD myzset 1 "uno "  
>ZADD myzset 2 "two" 3 "three "  
>ZRANGE myzset 0 -1 WITHSCORES  
>ZINCRBY myzset 3 "one"  
>ZRANGE myzset 0 -1
```

Autres caractéristiques

▪ Publish/Subscribe

※ Client 1

```
redis> publish public-notice  
'test message'  
(integer) 1
```

```
redis> publish public-alert  
'server will shutdown'  
(integer) 1
```

※ Client 2

```
redis> psubscribe public*  
Reading messages... (press Ctrl-C  
to quit)  
1) "psubscribe"  
2) "public*"  
3) (integer) 1  
  
1) "pmESSAGE"  
2) "public*"  
3) "public-notice"  
4) "test message"  
1) "pmESSAGE"  
2) "public*"  
3) "public-alert"  
4) "server will shutdown"
```

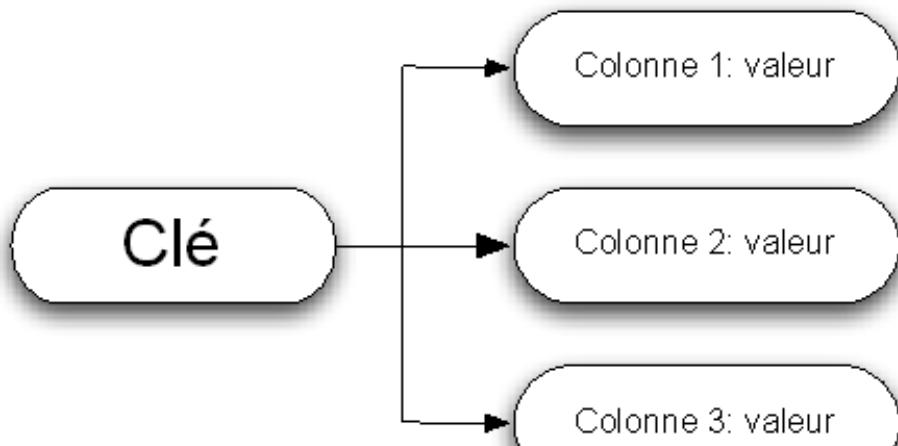
From: Kris Jeong, REDIS Intro

Autres systèmes

- **Voldemort:** <http://project-voldemort.com/>
 - Crée en 2008 par LinkedIn – Open-source
 - Implémenté en java, fournit des plugins pour Berkeley DB et MySQL
 - Méthodes d'accès: Thrift, Avro et protobuf. Utilisable avec Hadoop
- **Kyoto Cabinet:** <http://fallabs.com/kyotocabinet/>.
 - Implémenté en C++, Licence GNU-GPL
 - Système de fichiers avec des paires (clé-valeur)
 - Méthodes d'accès: API pour C, C++, Java, C#, Python, Ruby, Perl, Erlang, Ocaml,...
- **Et aussi Riak, Kai, Membase, ...**



Bases de données orientées colonne



Cassandra, Hbase,Bigtable, ...

DB orientée colonnes

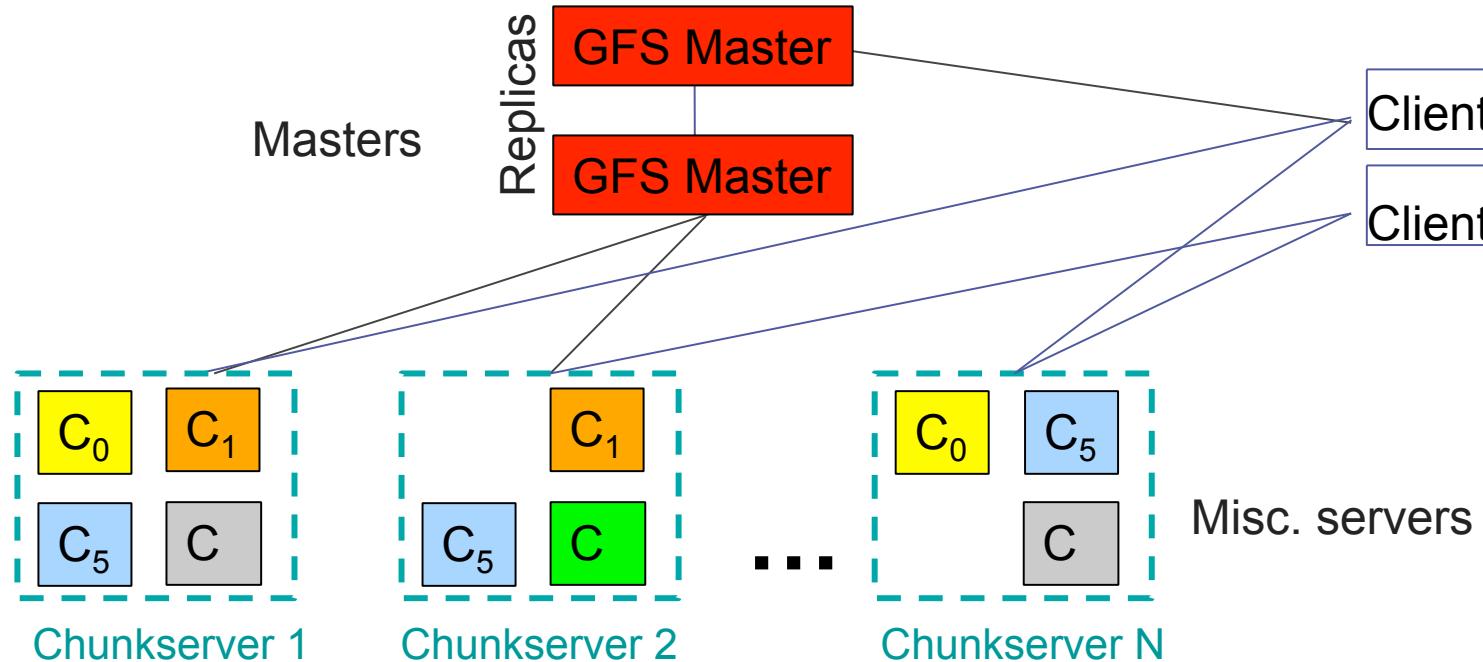
- **Ressemble logiquement aux bases de données relationnelles (tables)**
- **Organisée physiquement en fonction des colonnes**
- **Utile pour les tâches d'analyses sur des colonnes**
- **Schéma dynamique**
- **Utile pour les données éparses**

BigTable

- **Base de données propriétaire gérée en interne chez Google**
- **Accessible au public uniquement via**
 - Google App Engine
- **Écrite en C++**
- **Cohérence forte (strong consistency)**
 - L'ensemble des clients voient la même valeur d'une donnée après les mises à jour
- **Stockage de données**
 - Basé sur le système de fichiers distribués **GFS** (Google File Systems)



Google File System (GFS)



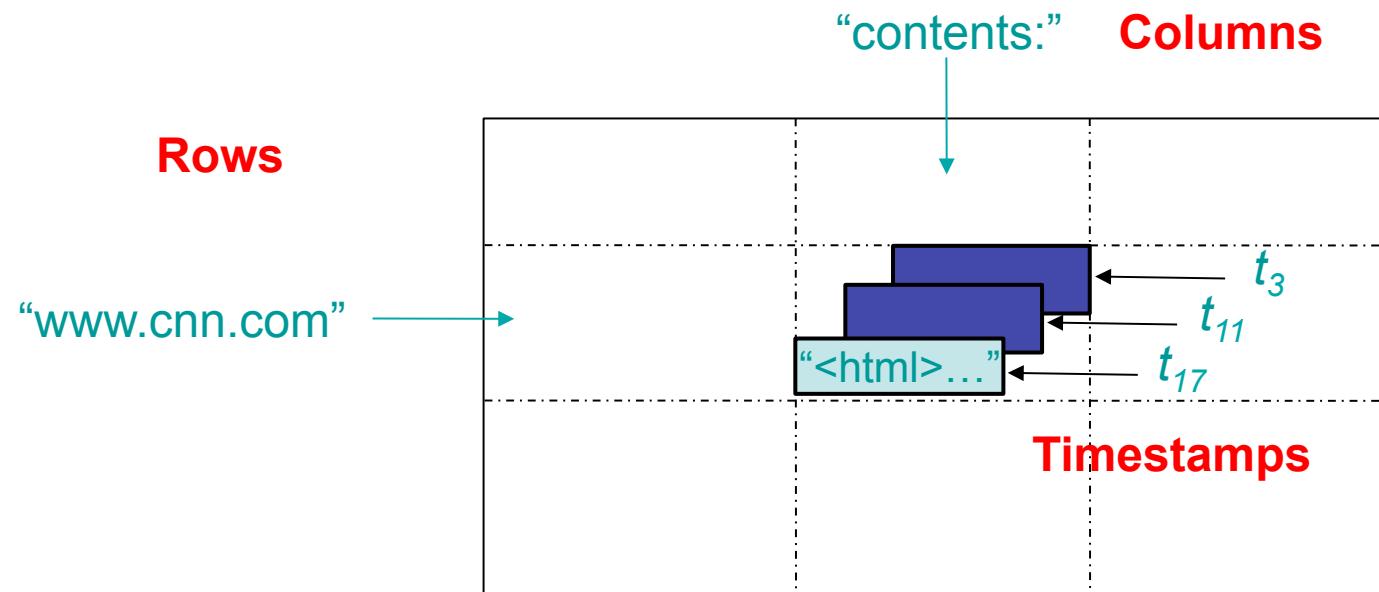
- Master manages metadata
- Data transfers happen directly between clients/chunkservers
- Files broken into chunks (typically 64 MB)
- Chunks triplicated across three machines for safety

BigTable-Modèle de données

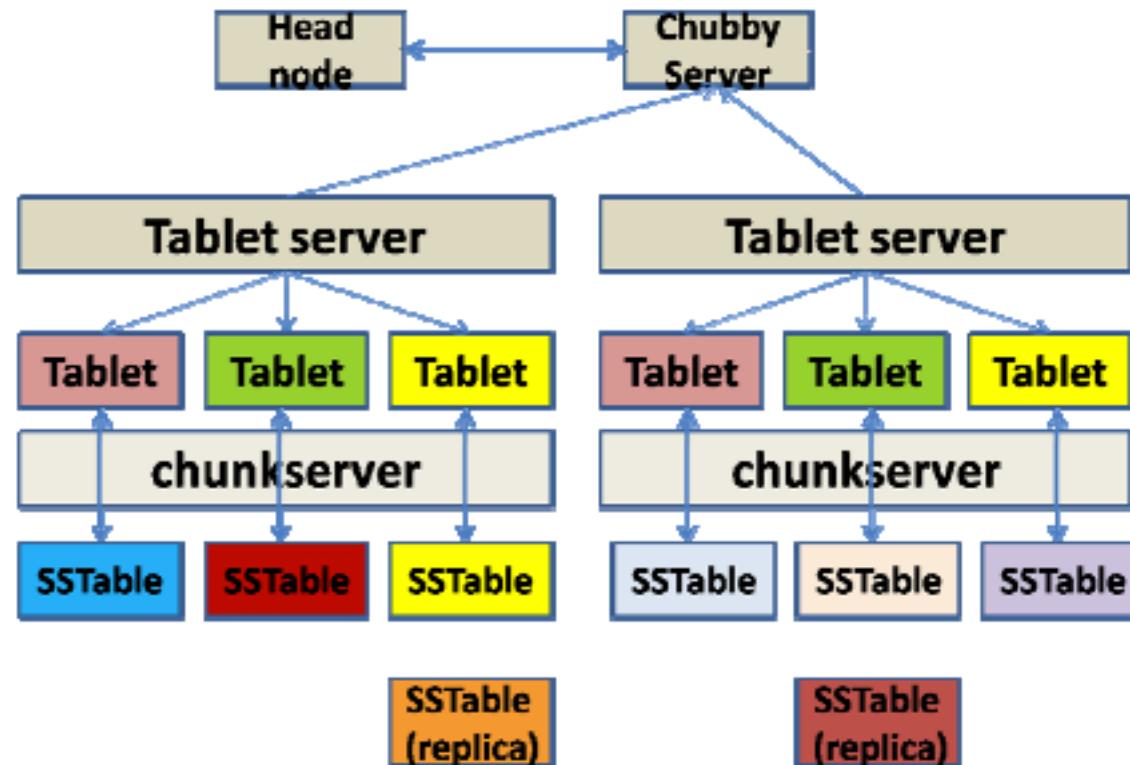
- **Map multi-dimensionnel**
- **Row key**
 - Clé unique d'une entité (string de 64KB)
- **Column family**
 - Représente un groupe d'attributs (colonnes) reliés
- **Column**
 - Comporte différentes versions de la donnée (ordonnées par timestamp décroissant)
- **L'accès à une donnée se fait par:**
 - (Row key → column family → column → timestamp)

BigTable-Modèle de données

$(row, column, timestamp) \rightarrow cell\ contents$



BigTable - Architecture



BigTable- Architecture

- **La tablet est l'unité de distribution de données**
 - Intervalle de ligne triées par ordre lexicographique des clés [start key-end key]
- **Tablet server**
 - Gère un ensemble de tablets (généralement quelques centaines)
- **Chubby server**
 - Maintient des connexions avec les Tablet Servers à l'aide de fichiers de verrouillage
 - Attribution dynamique des Tablets aux Tablet Servers
 - Responsable de la répartition de charge et de la tolérance aux fautes

Ecriture/lecture de données

- **Opérations d'écriture sont stockées dans une table en mémoire memtable**
- **Quand la taille de la memtable atteint un seuil prédéfini**
 - La memtable est figée et ensuite convertie en SSTable par le GFS
 - Une nouvelle memtable est créée
- **Opérations de lecture sont effectuées sur une combinaison de memtables et SSTables**

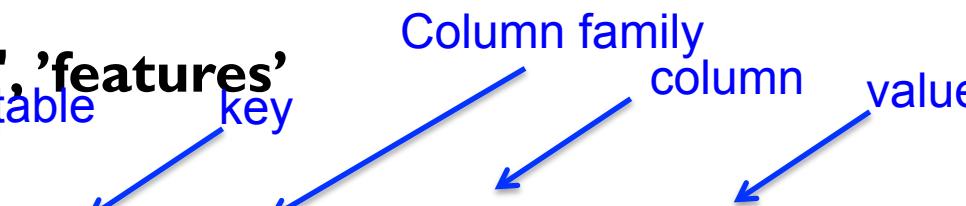
BigTable-Utilisateurs

- **Google analytics**
- **Google finance**
- **Orkut (réseau social tenu par Google)**
- **Personalized search**
- **Writely (éditeur de texte on-line racheté par Google)**
- **Google earth**

Hbase

- **Basé sur BigTable**
 - *HDFS (GFS), ZooKeeper (Chubby)*
 - *Master Node (Master Server), Region Servers (Tablet Servers)*
 - *HStore (tablet), memcache (memtable), HFile (SSTable)*
- **Écrit en Java**
- **Utilise HDFS**
- **Licence Apache 2.0**
- **Sponsorisé par :**
 - Yahoo!, Microsoft, HP, Facebook, Covalent, IONA, AirPlus International, BlueNog, Intuit, Joost, Matt Mullenweg, Two Sigma Investments.

Hbase : exemple d'usage

- **create 'cars', 'features'**

 - table → 'cars'
 - key → 'row1'
 - Column family → 'features'
 - column → 'make', 'model', 'year'
 - value → 'bmw', '5 series', '2012'
- **put 'cars', 'row1', 'features:make', 'bmw'**
- **put 'cars', 'row1', 'features:model', '5 series'**
- **put 'cars', 'row1', 'features:year', '2012'**

- **put 'cars', 'row2', 'features:make', 'mercedes'**
- **put 'cars', 'row2', 'features:model', 'e class'**
- **put 'cars', 'row2', 'features:year', '2012'**



HBase

■ Hbase propose:

- Modèle de données similaire à celui de BigTable
- Des classes natives pour la connexion avec MapReduce
- Accès REST (JSON, XML, etc.)
- Hive/Pig pour les analyses
- Java API
- Web Usage Interface
 - Tâches administratives, monitoring, ajout/suppression de noeuds, etc.
- Interface Thrift avec support pour plus de 10 langages
 - Python, PHP, Perl, Ruby, C++, Erlang, etc.

Hbase- architecture maître/esclave

- **Master**

- Responsable de l'allocation des régions aux RegionServers

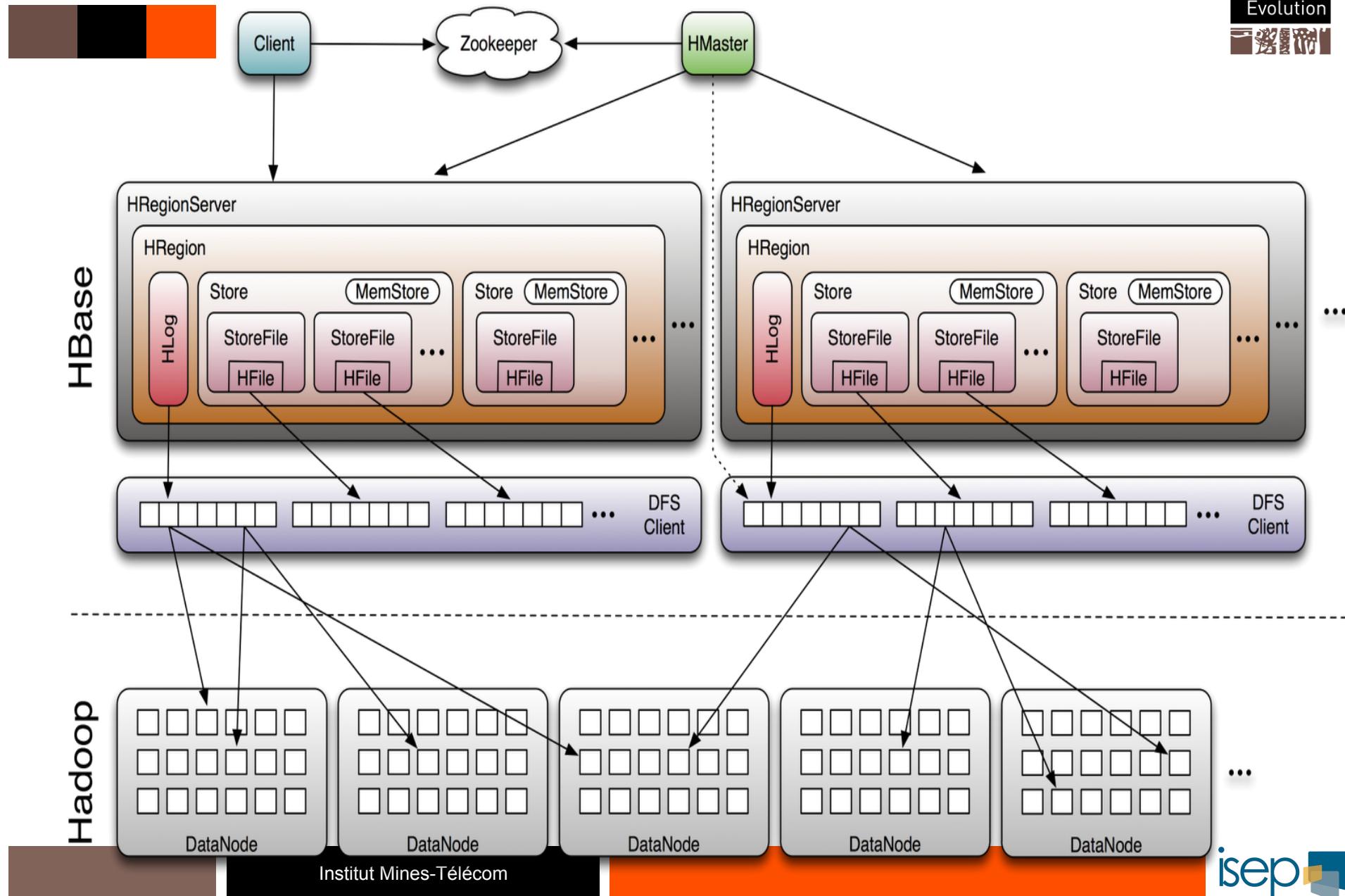
- **RegionServer**

- Responsable d'un ensemble de régions
 - Chaque région est un ensemble de rangées ordonnées

- **Hbase client**

- Interroge le master pour savoir sur quel RegionServer se retrouve la région recherchée
 - Ensuite communique directement avec la RegionServer pour obtenir les données

Hbase-Architecture



Hadoop

DataNode

DataNode

DataNode

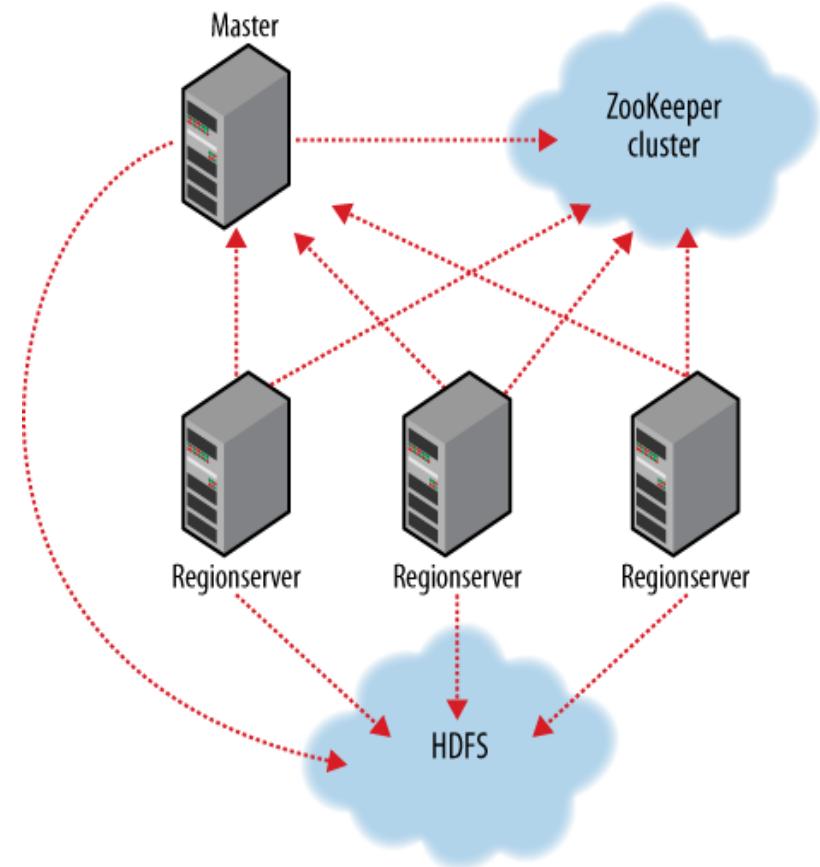
DataNode

DataNode

Institut Mines-Télécom

ZooKeeper

- basé sur *Chubby Server de Google*) est utilisé pour gérer différents serveurs (afin de résoudre le problème de *Single Point of Failure*).



Hbase-Utilisateurs

- <http://www.yahoo.com/>
- <http://www.adobe.com/>
- <http://www.stumbleupon.com/>
- <http://www.bedrock.com/>
- <http://www.filmweb.pl/>
- <http://www.flurry.com/>
- <http://www.drawtoscaleconsulting.com/>
- <http://www.kalooga.com/>
- <http://www.mahalo.com/>
- <http://www.meetup.com/>
- <http://ning.com/>
- <http://www.openplaces.org/>
- <http://www.powerset.com/>
- <http://www.readpath.com/>
- <http://www.runa.com/>
- <http://www.socialmedia.com/>
- <http://www.streamy.com/>
- <http://www.stumbleupon.com/>
- <http://www.subrecord.org/>
- etc.

Cassandra

- **Initiateur**
 - Facebook en 2007 (solution pour le problème « inbox search »)
- **Licence Apache 2.0**
- **Écrit en Java**
- **Modèle de données**
 - Basé sur BigTable (modèle de données) et Dynamo (partitionnement et cohérence)
 - Pas de versioning
 - Définition de **super-colonnes**
- **Interface Thrift**
 - Ruby, Perl, Python, Scala et Java, ...
 - CQL (Cassandra Query Language) : ressemble au SQL au niveau des commandes.

Cassandra

- **Décentralisation**
 - Pas d'architecture maître/esclave
 - Chaque nœud dans le cluster est identique
- **Tolérance aux fautes**
 - Les données sont répliquées sur N (facteur de réPLICATION) nœuds
 - 'Always writable': accepte l'écriture de données même en cas de défaillance
- **Extensibilité**
 - Ajout de nouveaux nœuds à l'aide d'un protocole 'gossip'
 - Le débit de lecture/écriture augmente de manière linéaire avec l'augmentation du nombre de machines
- **Cohérence « tunable »**
 - Les écritures et les lectures offrent un niveau de cohérence configurable

Niveaux de cohérence (Consistency)

■ Pour une écriture

- Le niveau ONE
 - Assure que l'écriture a été effectuée dans au moins un nœud
- Le niveau ALL
 - Assure que l'écriture a été effectuée vers tous les N nœuds (avec $N=ReplicationFactor$)
- Le niveau QUORUM
 - Assure que l'écriture a été effectuée sur les M nœuds (avec $M=ReplicationFactor/2+1$)

■ Pour une lecture

- Le niveau ONE
 - Envoie l'enregistrement retourné par le premier nœud qui a répondu
- Le niveau ALL
 - Requête tous les nœuds et retourne l'enregistrement avec le timestamp le plus récent
 - Si un nœud ne répond pas, l'opération échoue
- Le niveau QUORUM
 - Requête tous les nœuds et retourne l'enregistrement avec le timestamp le plus récent retourné par la majorité des noeuds

Cassandra- Modèle de données

- **L'accès à une donnée se fait par:**
 - Keyspace → column family → row → optional super column → column
- **Exemple de super column**
 - Supposons que l'utilisateur identifié par la clé '123456' a plusieurs pages dans son marque-pages (bookmarks). Une requête de type (:UserBookmarks,'123456') pourrait retourner:

```
{"work" =>  
  { 'Google' => 'http://google.com',  
   'IBM' => 'http://ibm.com'},  
  'todo': {...},  
  'cooking': {...}  
}
```

Dans cet exemple, work, todo et cooking sont des 'super columns'
et :UserBookmarks est le nom de la 'column family'

Colonne

- **La plus petite entité**
- **Triplet: nom, valeur, timestamp**

```
//this is a column in JSON notation
    name: ''emailAddress'',
    value:'rchiky@isep.fr'
    timestamp:123456789
}
```

Super-colonne

```
{ // this is a SuperColumn
  name: "homeAddress",
  // with an infinite list of Columns
  value: {
    // note the keys is the name of the Column
    street: {name: "street", value: "1234 x street", timestamp:
123456789},
    city: {name: "city", value: '' Paris", timestamp: 123456789},
    zip: {name: "zip", value: " 75013", timestamp: 123456789},
  }
}
```

Famille de colonnes (~Table)

- **Contient des lignes (rows) et chaque ligne contient des colonnes**

```
UserProfile = { // this is a ColumnFamily
    pseudo1: { // this is the key to this Row inside the CF
        // now we have an infinite # of columns in this row
        username: "pseudo1",
        email: "pseudo1@example.com",
        phone: "(+33) 6976-6666"
    }, // end row
    pseudo2: { // this is the key to another row in the CF
        // now we have another infinite # of columns in this row
        username: "pseudo2",
        email: "pseudo2@example.com",
        phone: "(+33) 6555-1212",
        age: "66",
        gender: "undecided"
    },
}
```

Cassandra- Utilisateurs (2011 vs.

TELECOM



Cassandra : Installation (1 seul nœud)



- Télécharger la dernière version à partir de <http://cassandra.apache.org/download/>
- Décompresser (xxx étant le numéro de version)
 - tar zxvf apache-cassandra-xxx-bin.tar.gz
- Un seul nœud (très facile pour un cluster, répéter l'opération pour chaque nœud)
- Configuration : 1 seul fichier de configuration
 - Conf/cassandra.yaml

directories where Cassandra should store data on disk. (ligne 69)

data_file_directories:

- /var/lib/cassandra/data

commit log

commitlog_directory: /var/lib/cassandra/commitlog (ligne 74)

saved caches

saved_caches_directory: /var/lib/cassandra/saved_caches (ligne 141)

- sudo mkdir /var/lib/cassandra.

Configuration de log4j pour Cassandra



- **Fichier de configuration: log4j-server.properties**
log4j.append.R.File=/var/log/cassandra/system.log
- **sudo mkdir /var/log/cassandra**
- **Lancement du serveur**
 - ./bin/cassandra -f
- **Lancement du client**
 - ./bin/cassandra-cli [-host localhost –port 9160]

```
porta-info-4:cassandra rchiky$ bin/cassandra-cli
Welcome to the Cassandra CLI.

Type 'help;' or '?' for help.
Type 'quit;' or 'exit;' to quit.

Cassandra CLI version 1.0.2
[default@unknown] connect localhost/9160;
Connected to: "Test Cluster" on localhost/9160
[default@unknown]
```

Keyspace

- ~ = Base de données
- généralement un keyspace par application

```
[default@unknown] show keyspaces;
Base de données
Keyspace: system;
  Replication Strategy: org.apache.cassandra.locator.LocalStrategy
  Durable Writes: false
  Options: [replication_factor:1]
Column Families:
  ColumnFamily: HintsColumnFamily (Super)
    "hinted handoff data"
      Key Validation Class: org.apache.cassandra.db.marshal.BytesType
      Default column value validator: org.apache.cassandra.db.marshal.BytesType
      Columns sorted by: org.apache.cassandra.db.marshal.BytesType/org.apache.cassandra.db.marshal.BytesType
      Row cache size / save period in seconds / keys to save : 0.0/0/all
      Row Cache Provider: org.apache.cassandra.cache.ConcurrentLinkedHashCacheProvider
      Key cache size / save period in seconds: 0.01/0
      GC grace seconds: 0
      Compaction min/max thresholds: 4/32
      Read repair chance: 0.0
      Replicate on write: true
      Built indexes: []
        Compaction Strategy: org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy
      ColumnFamily: IndexInfo
        "indexes that have been completed"
          Key Validation Class: org.apache.cassandra.db.marshal.BytesType
          Default column value validator: org.apache.cassandra.db.marshal.BytesType
          Columns sorted by: org.apache.cassandra.db.marshal.UTF8Type
          Row cache size / save period in seconds / keys to save : 0.0/0/all
          Row Cache Provider: org.apache.cassandra.cache.ConcurrentLinkedHashCacheProvider
          Key cache size / save period in seconds: 0.01/0
          GC grace seconds: 0
          Compaction min/max thresholds: 4/32
```

Création d'un keySpace

```
>help create keyspace;
```

```
>create keyspace CarDataStore  
    with placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy' and  
    strategy_options = {replication_factor:1};
```

```
>show keyspaces;
```

```
>use CarDataStore;
```

```
>help create column family;
```

```
>create column family Cars  
    with comparator =AsciiType  
    and gc_grace = 0  
    and min_compaction_threshold = 5 and max_compaction_threshold = 31;
```

```
bin/cassandra-cli -host localhost --file \Users\rchiky\NoSQL\examples\cardatastore-cassandra.txt
```

Ajouter des données

```
set Cars['Prius']['make'] = 'toyota';
set Cars['Prius']['model'] = 'prius 3';
set Cars['Corolla']['make'] = 'toyota';
set Cars['Corolla']['model'] = 'le';
set Cars['fit']['make'] = 'honda';
set Cars['fit']['model'] = 'fit sport';
set Cars['focus']['make'] = 'ford';
set Cars['focus']['model'] = 'sel';
```

« Requêter » les données

- **get Cars['Prius'];**

```
[default@CarDataStore] get Cars['Prius'];
=> (column=make, value=746f796f7461, timestamp=1322128161917000)
=> (column=model, value=70726975732033, timestamp=1322128280285000)
Returned 2 results.
```

- **get Cars['Prius']['make'];**

```
[default@CarDataStore] get Cars['Prius']['make'];
=> (column=make, value=746f796f7461, timestamp=1322128161917000)
```

Cassandra : Système de requêtes



- **Requêtes simples**
- **Requêtes par clé ou ensemble de clés**
- **Possibilité d'ajouter un attribut colonne**
- **Commandes : set/get/del/count/list/truncate/drop/create/update**

>help list;

>help del;

- **CQL:**

>./bin/cqlsh

>USE CarDataStore;

Cassandra: tri

- Les données sont triées dans cassandra
- Option **CompareWith** : **BytesType**, **UTF8Type**, **LexicalUUIDType**, **TimeUUIDType**, **AsciiType**, and **LongType**
- **Exemple: LongType**

//Exemple non trié mais jamais ce sera le cas dans cassandra

// ignorer les valeurs

```
{name: 123, value: "hello there"},  
{name: 832416, value: "kjjkbcjkcbbd"},  
{name: 3, value: "101010101010"},  
{name: 976, value: "kjjkbcjkcbbd"}
```

Cassandra: tri

- **Exemple: LongType**

// Chaque colonne est considérée comme un long 64bit

```
{name: 3, value: "101010101010"},  
{name: 123, value: "hello there"},  
{name: 976, value: "kjjkbcjkcbbd"},  
{name: 832416, value: "kjjkbcjkcbbd"}
```

Cassandra: tri

- **Exemple: UTF8Type**

```
{name: 123, value: "hello there"},  
{name: 3, value: "101010101010"},  
{name: 832416, value: "kjjkbcjkcbbd"},  
{name: 976, value: "kjjkbcjkcbbd"}
```

Cassandra:tri des super-colonnes

// 2 enregistrements avec 2 super-colonnes.

// Dans un ordre aléatoire

```
{ // first SuperColumn from a Row
    name: "workAddress",
    // and the columns within it
    value: {
        street: {name: "street", value: "1234 x street"},
        city: {name: "city", value: "san francisco"},
        zip: {name: "zip", value: "94107"}
    }
},
{ // another SuperColumn from same Row
    name: "homeAddress",
    // and the columns within it
    value: {
        street: {name: "street", value: "1234 x street"},
        city: {name: "city", value: "san francisco"},
        zip: {name: "zip", value: "94107"}
    }
}
```

Cassandra:tri des super-colonnes

- CompareSubcolumnsWith=UTF8Type
- CompareWith=UTF8Type

```
{  
  {  
    name: "homeAddress",  
  
    value: {  
      // see, these are sorted by Column name too  
      city: {name: "city", value: "san francisco"},  
      street: {name: "street", value: "1234 x street"},  
      zip: {name: "zip", value: "94107"}  
    }  
  },  
  name: "workAddress",  
  value: {  
  
    city: {name: "city", value: "san francisco"},  
    street: {name: "street", value: "1234 x street"},  
    zip: {name: "zip", value: "94107"}  
  }  
}
```

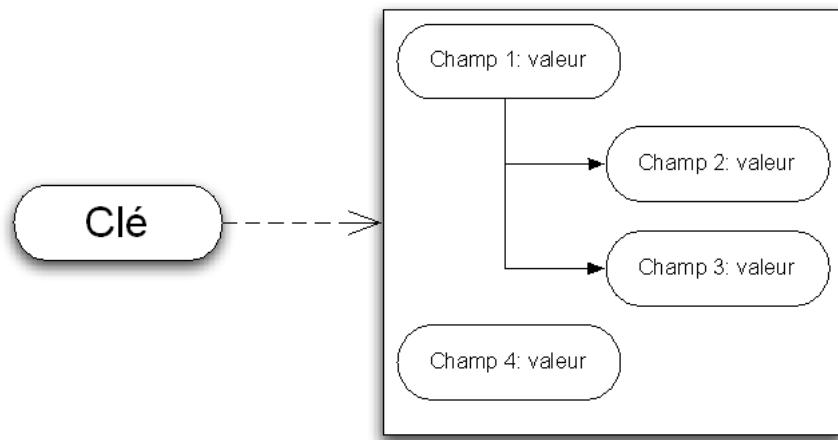
BD orientées colonne

- **Adapté pour les données semi-structurées**
- **Indexation simple**
- **Extensibilité horizontale**
- **Utilisation de Mapreduce pour les requêtes**
- **Ajout facile de colonnes**



Bases de données orientées documents

CouchDB, MongoDB



Modèle document

- **Collection de « documents »**
- **Modèle « clé/valeur », la valeur est un document semi-structuré hiérarchique de type JSON ou XML**
- **Pas de schéma pour les documents**
- **Structure arborescente : une liste de champs, un champ a une valeur qui peut être une liste de champs, ...**
- **CRUD**
- **BD les plus connues:**
 - CouchDB, MongoDB, ...

CouchDB

- **<http://couchdb.apache.org/> , ~2005**
- **Pas de schéma, base de données orientée document**
 - Documents stockées en format JSON (XML dans les vieilles versions)
 - Stockage suivant B-tree
 - Pas de verrouillage
 - Pas de jointure, pas de clé primaire ou clé secondaire
 - Implémenté en Erlang
 - 1^{ère} version en C++ puis en Erlang
 - RéPLICATION
 - Versioning

CouchDB – Format JSON

- Similaire au XML
- Plus facile à lire et écrire

```
{  
    "Type": "Auteur",  
    "Prenom": "Julien",  
    "Nom": "Dupont",  
    "Sexe": "Femme",  
    "date_naissance": "19021989",  
    "date_inscription": "28062011"  
}
```



JSON

```
{"menu": {  
    "id": "file",  
    "value": "File",  
    "popup": {  
        "menuitem": [  
            {"value": "New", "onclick":  
"CreateNewDoc()"},  
            {"value": "Open", "onclick":  
"OpenDoc()"},  
            {"value": "Close", "onclick":  
"CloseDoc()"}  
        ]  
    }  
}}
```

XML

```
<menu id="file" value="File">  
  <popup>  
    <menuitem value="New"  
    onclick="CreateNewDoc()" />  
    <menuitem value="Open"  
    onclick="OpenDoc()" />  
    <menuitem value="Close"  
    onclick="CloseDoc()" />  
  </popup>  
</menu>
```



CouchDB REST API

- **Create** **HTTP PUT** **/db/docid**
- **Read** **HTTP GET** **/db/docid**
- **Update** **HTTP POST** **/db/docid**
- **Delete** **HTTP DELETE** **/db/docid**
- **Vues**
 - Filter, sort, “join”, aggregate, report
 - Basé sur Map/Reduce
 - Construite sur demande
 - Peuvent être matérialisées et mises à jour incrémentalement

CouchDB- Utilisateurs

- **Apple**
- **BBC**
- **Canonical**
- **Cern**



MongoDB

- **SGBD open source, orienté documents. Il est scalable, sans schéma, et non relationnel**
- **Modèle de requêtage très riche**
- **Système d'indexation des documents**
- **Requêtes géo-spatiales**

MongoDB- utilisateurs

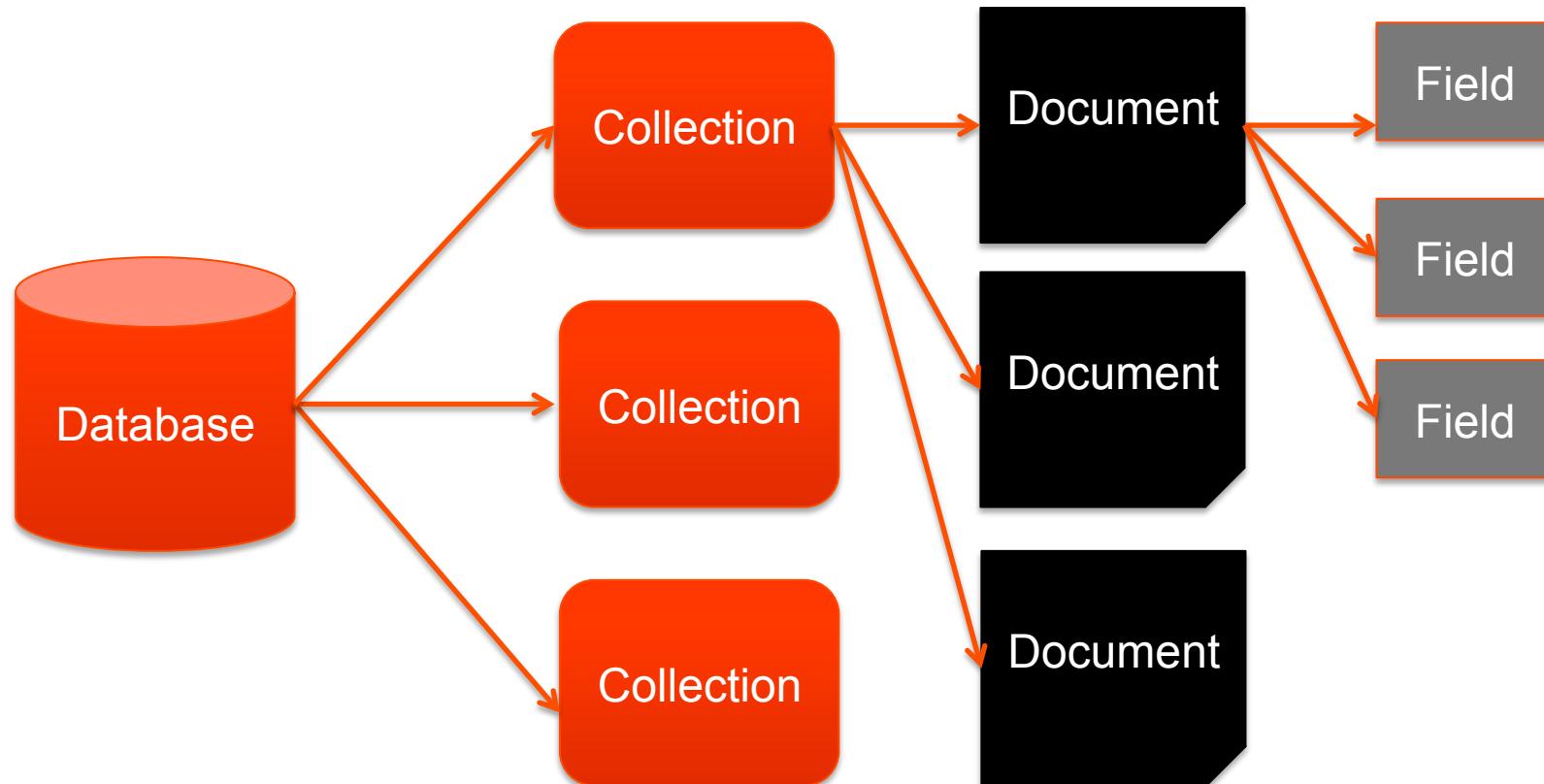


MongoDB- Stockage des données

- **Stockage au format BSON (“binary JSON”)**
- **Types supportés:**
 - String, Integer, Double, Date, Byte Array, Booléen, Null, BSON object et BSON Array
- **Exemple dans une collection "users" :**

```
{  
  "_id": "c9167a15625045fb439c7078 ",  
  "username": "rchiky",  
  "firstname": "Raja",  
  "lastname": "Chiky"  
}
```
- **Drivers existant dans la plupart des langages de programmation**
C, C++, JavaScript, Python, Perl, C# / .NET, Java, PHP, ...
- **Flexibilité:**
 - Authentification facultative
 - Génération automatique des collections
 - Accès à des fonctionnalités avancées

Modèle de données



MongoDB vs. SGBDR

SGBDR	MongoDB
Database	Database
Tables	Collections
Rows	Documents
Columns	Champs

- Les documents peuvent avoir un nombre de clés différents même s'ils font partie de la même collection

`{name:'Chiky', city:'Paris', hobby:['Tennis','Movies']}`

`{name:'Chiky', city:'Paris', profession:'Associate professor', tel:'0149545234'}`

MongoDB- MapReduce

- MongoDB implémente le framework Map/Reduce
- Exemple: (en JavaScript)

```
map = function(){
    emit(this.age, this.city);
}

reduce = function(key, values){
    var total = 0, nb = 0;
    values.forEach(function(value) {
        total += value; // Ajout de l'âge
        nb++; // Compteur pour faire la moyenne
    });
    return nb == 0 ? 0 : total / nb;
}
```

- Pour obtenir la liste des moyennes d'âge par ville:

```
>db.users.mapReduce(map, reduce, {out: { inline : 1 }});
```

(Le 3^e argument permet de ne pas stocker les résultats dans une nouvelle collection et donc de tout faire en mémoire).

MongoDB: Modèle de données

Relational

PERSON				
Pers_ID	Surname	First_Name	City	
0	Miller	Paul	London	
1	Ortega	Alvaro	Valencia	
2	Huber	Urs	Zurich	
3	Blanc	Gaston	Paris	
4	Bertolini	Fabrizio	Rome	

CAR				
Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

NO RELATION



MongoDB

```
{
  first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: [45.123, 47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, ... },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, ... }
  ]
}
```

MongoDB - Requête

Rich Queries

- Find Paul's cars
- Find everybody in London with a car built between 1970 and 1980

Geospatial

- Find all of the car owners within 5km of Trafalgar Sq.

Text Search

- Find all the cars described as having leather seats

Aggregation

- Calculate the average value of Paul's car collection

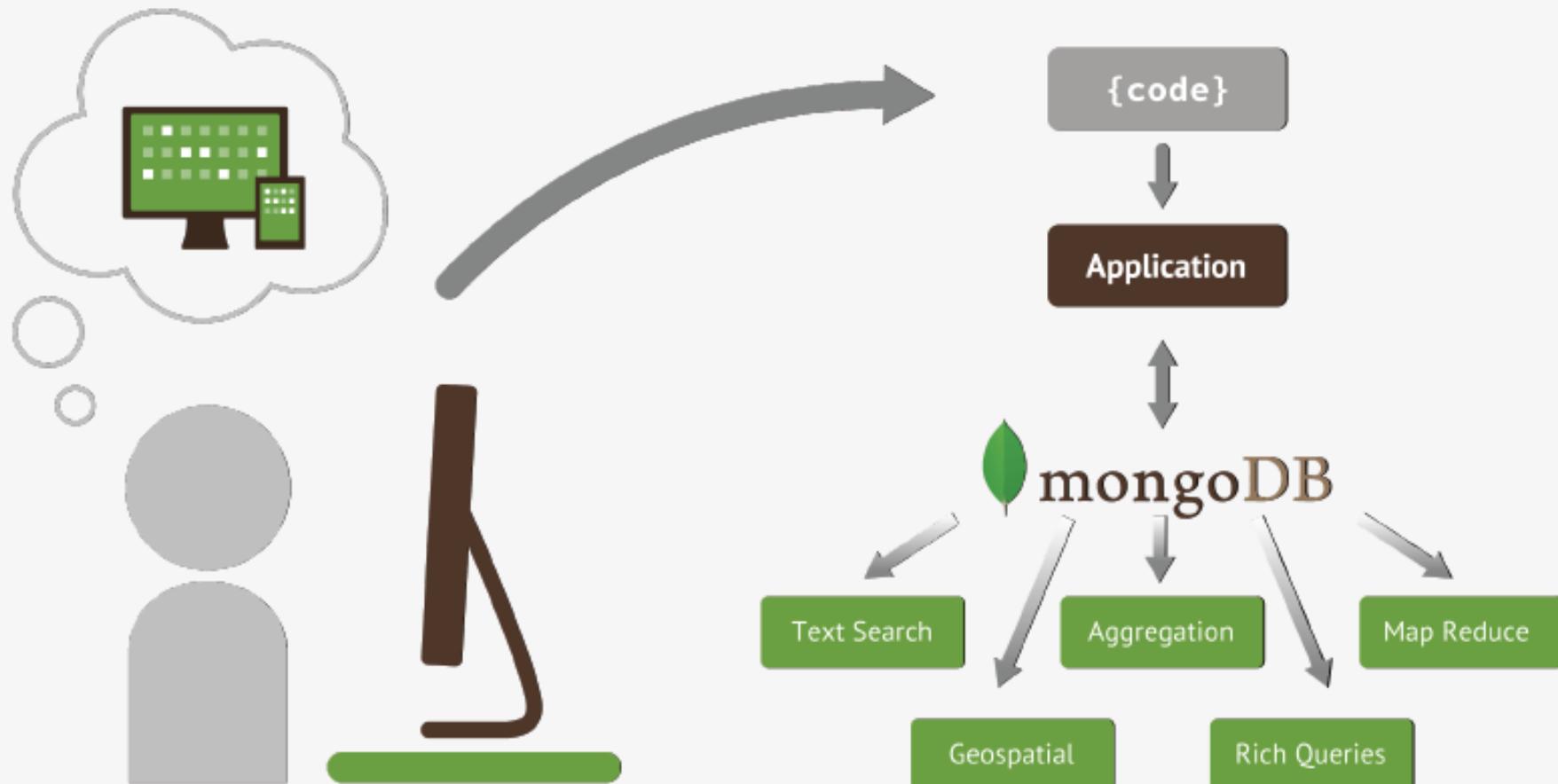
Map Reduce

- What is the ownership pattern of colors by geography over time? (is purple trending up in China?)

MongoDB

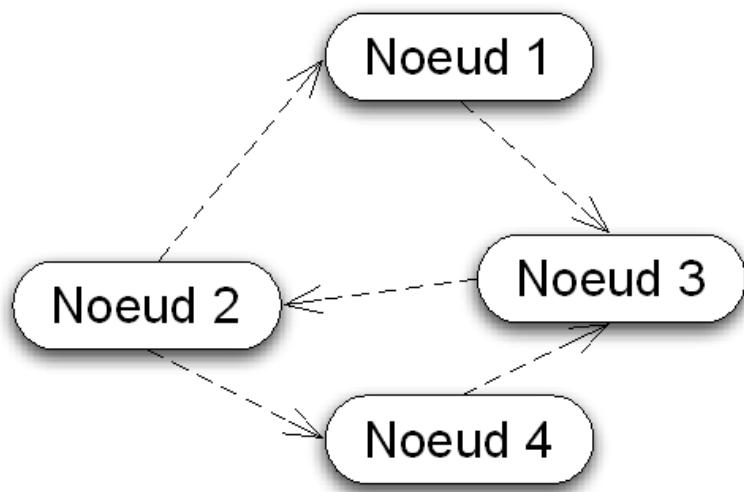
```
{
  first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location:
  [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, ... },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, ... }
  ]
}
```

Developers are more productive



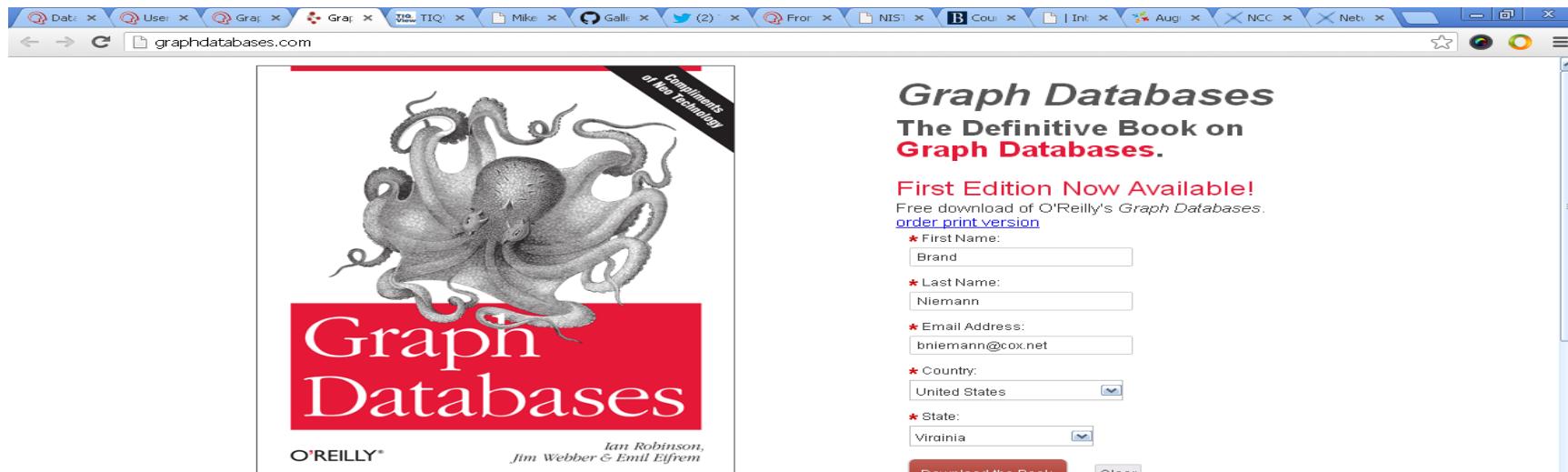


Bases de données orientées graph



Neo4J, Infinite Graph,
HyperGraphDB,...

Graph Databases: Livre



Graph Databases introduces graphs and graph databases to technology enthusiasts, developers, and database architects.

Graph Databases, published by O'Reilly Media, discusses the problems that are well aligned with graph databases, with examples drawn from practical, real-world use cases. This book also looks at the ecosystem of complementary technologies, highlighting what differentiates graph databases from other database technologies, both relational and NOSQL.

Graph Databases is written by Ian Robinson, Jim Webber, and Emil Eifrem, graph experts and enthusiasts at [Neo Technology](#), creators of [Neo4j](#), the world's leading graph database.

Table of Contents

- 1. Introduction

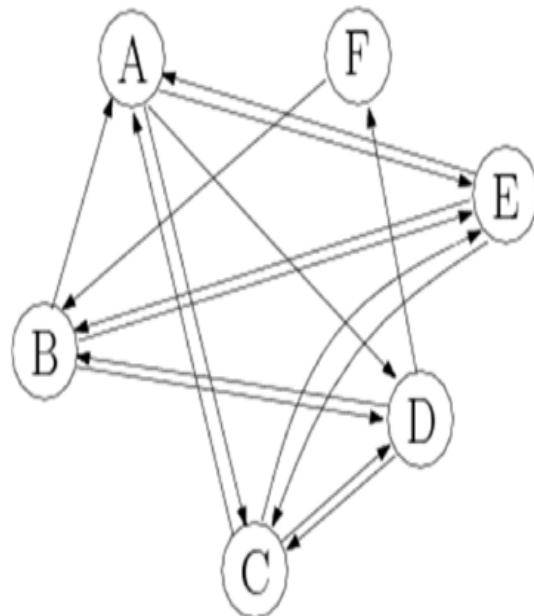
+

Ian Robinson's Blog:
[Running the Graph Databases Book Examples](#)

The current version of Neo4j has two different types of index: named indexes and automatic indexes. The Cypher query examples throughout the [Graph Databases](#) book use named indexes. There's a problem, however: data created using a Cypher CREATE statement won't be indexed in a named index. This has

<http://graphdatabases.com/>

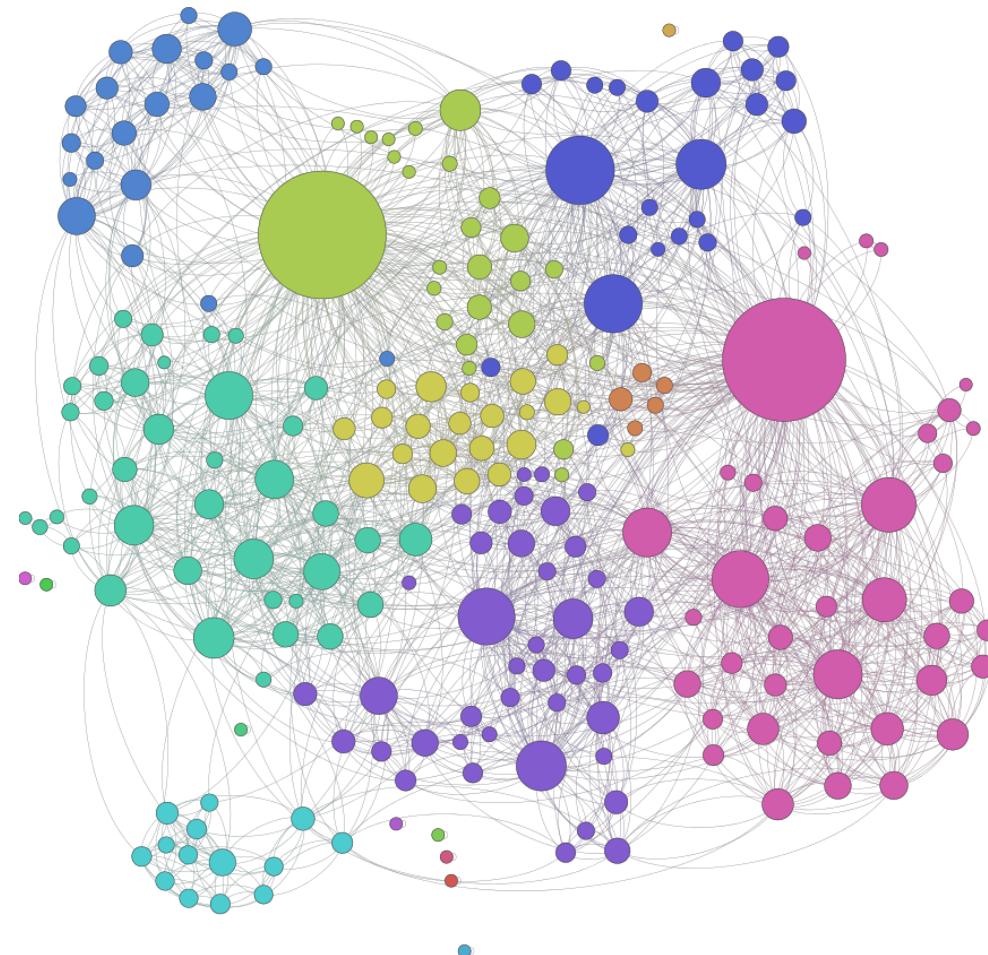
Graphes



- Représente un réseau
- Il s'agit d'un ensemble d'éléments (appelés noeuds ou sommets) reliés par des liens pouvant posséder une direction.
- Un graphe est donc une structure de données générique et très flexible



Graphes partout



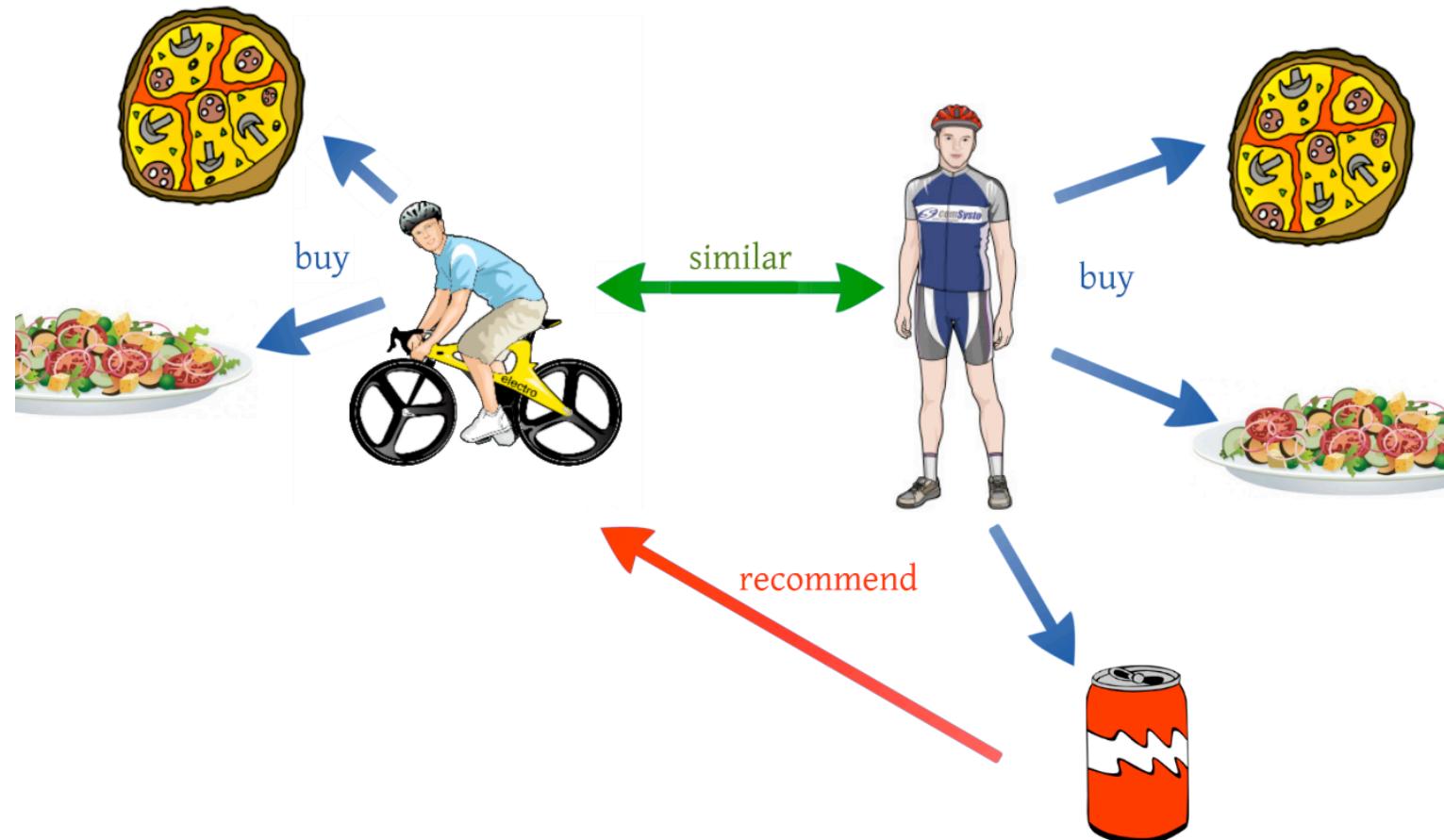
Réseaux sociaux



Itinéraire



Systèmes de recommandation



Base de données orientée graphe

- **Optimisée pour les connexions entre les enregistrements**
- **Rapidité pour l'interrogation des enregistrements**
- **Une base de données transactionnelle avec les opérations habituelles**

- **Une base de données relationnelle peut vous dire combien de livres vous avez vendu au dernier trimestre**
- **Mais une base de données orienté graphe peut également vous dire quel livre votre client peut acheter à sa prochaine visite**

SGBDR vs BDGraphes

Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

Bob's friend?



PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

```
SELECT Person.person
FROM Person JOIN PersonFriend
  ON Person.person = PersonFriend.person
WHERE PersonFriend.friend = 'Bob'
```

→ Alice

SGBDR vs BDGraphes

Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

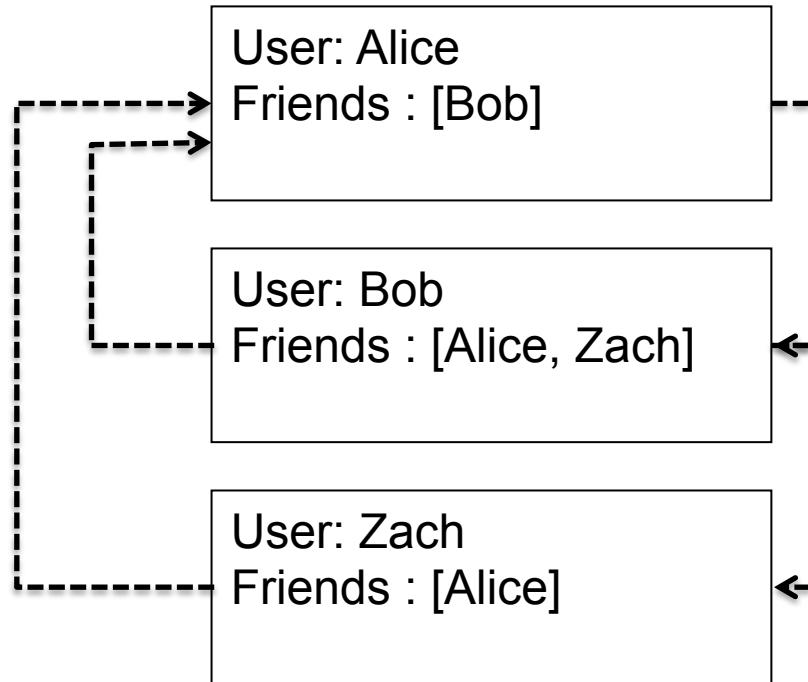


PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Alice's friends-of-friends

```
SELECT pf1.person as PERSON, pf3.person as FRIEND_OF_FRIEND
FROM PersonFriend pf1 INNER JOIN Person
    ON pf1.person = Person.person
INNER JOIN PersonFriend pf2
    ON pf1.friend = pf2.person
INNER JOIN PersonFriend pf3
    ON pf2.friend = pf3.person
WHERE pf1.person = 'Alice' AND pf3.person <> 'Alice'
```

BD NoSQL



Who are Bob's friends?
Who is friend with Bob?

BD Graphes

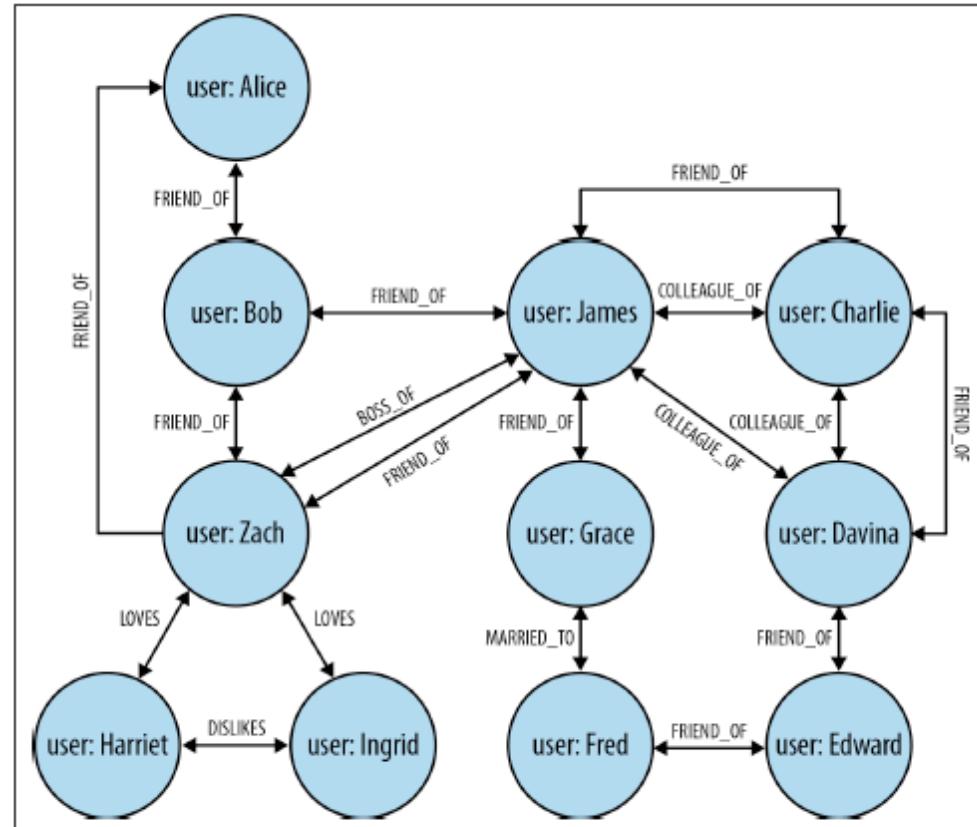


Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph



Comparaison

- ~1000000 personnes
- Moyenne 50 friends par personne

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Base de données orientée graphe

- **Les noeuds possèdent des propriétés qui peuvent être des variables de plusieurs types**
- **Les liens sont appelés des relations et possèdent un type, un sens et des propriétés**
- **On peut traverser le graphe suivant des conditions déterminées à partir d'un noeud de départ et en suivant ses relations**
- **Les propriétés des noeuds et des relations peuvent être indexées afin de rechercher parmi leurs valeurs**
- **Exemple: Neo4j. Populaire, open-source avec support commercial, bonne documentation**
- **Clients: Adobe, Viadeo, Cisco...**

En résumé

- **Avantages :**

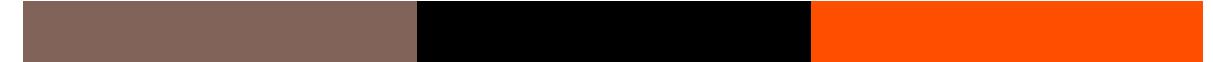
- Représentation visuelle sous forme de graphe
- Modélisation de données et relations complexes
- Algorithmes de la théorie des graphes, par exemple le plus court chemin, le moins coûteux (Dijkstra ou A*)...

- **Inconvénients :**

- Doit traverser l'entièreté du graphe avant d'obtenir une réponse définitive
- Concept relativement récent

- **Utilisations classiques :**

- Données sociales (réseaux sociaux)
- Recommandations
- Données spatiales



Merci pour votre attention

Raja.chiky@isep.fr

