

Big Data

Emergence et principes des BdD NoSQL

Stéphane Vialle

&

Gianluca Quercini



ÉCOLE DOCTORALE
Sciences et technologies
de l'information
et de la communication (STIC)



Emergence et principes des BdD NoSQL

- 1. Motivations et émergence des BdD NoSQL**
2. Principes des BdD NoSQL
3. Aperçu des différentes BdD NoSQL
4. Le retour de SQL...
5. Format de données JSON

Motivation et émergence

1969-1970 : arrivée du modèle relationnel

Rappel des principes :

- Repose sur des relations entre les valeurs des données
(indépendamment de leur emplacement en mémoire)
- Manipulation à travers une algèbre et un langage de haut niveau
- Dissocie représentation-et-interrogation du stockage, et sera quand même efficace!
(les moteurs de SGBD finiront même par être plus efficace que les solutions *ad hoc*)

Mais des contraintes :

- Toutes les lignes d'une Relation ont les mêmes colonnes
(valeur *NULL* si absence de données)
- Modifications par séquences atomiques pour que la BdD soit toujours totalement cohérente
- Conception d'un schéma de base qui s'impose à toute la BdD
- Interrogation par *jointures* de nombreuses (petites) Relations

Motivation et émergence

1969-1970 : arrivée du modèle relationnel

On ne peut pas mettre n'importe quoi dans une BdD Relationnelle !

- Toutes les données doivent respecter le schéma initial...
...difficile de faire entrer des données imprévues !

- Le langage Relationnel SQL est adapté pour extraire des informations selon des conditions sur leurs valeurs
→ requêtes OLTP (OnLine Transaction Processing) : **OK**

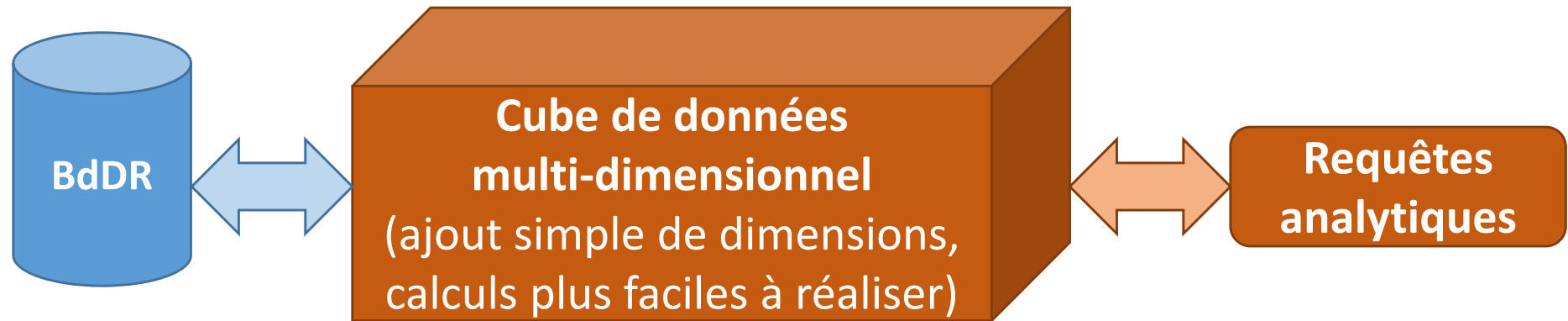
...mais n'est pas adapté pour faire des calculs de statistiques complexes sur ces valeurs, ni sur des données volumineuses!

- requêtes OLAP (OnLine Analytical Processing) : **problème...**

Motivation et émergence

1969-1970 : arrivée du modèle relationnel

De nouvelles solutions apparaissent pour les besoins en *analytics*

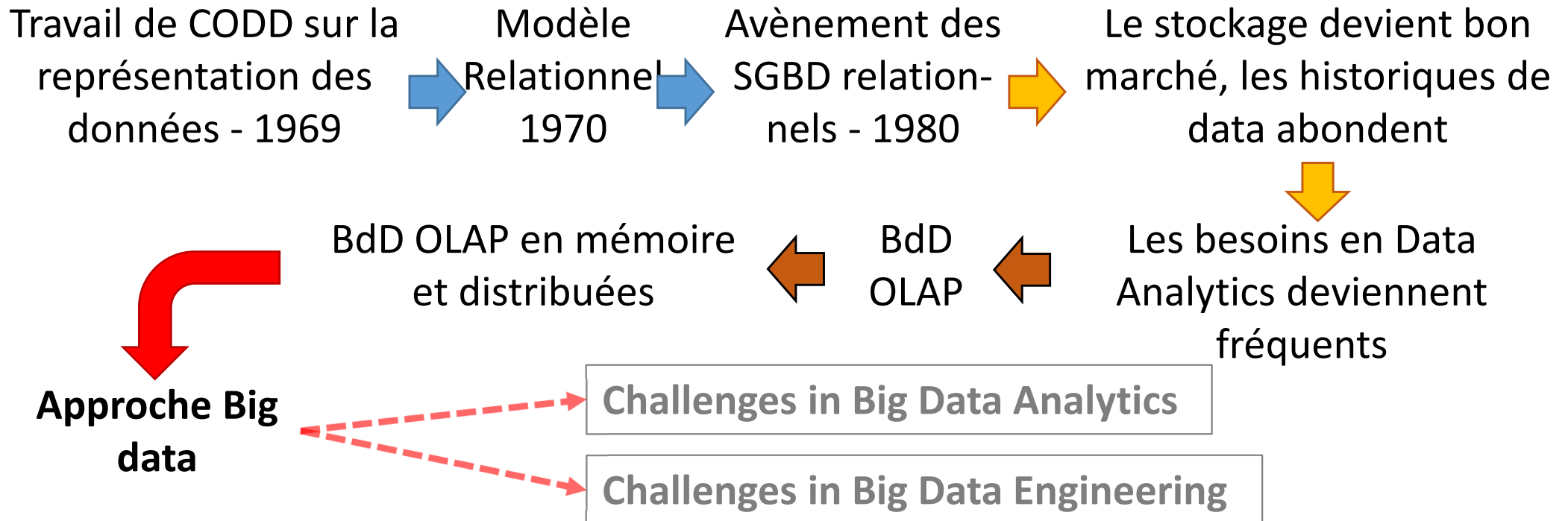


- OLAP, puis In-memory OLAP (plus rapide)
- CEP (*Complex Event Processing*) pour supporter des flux de mises à jour, et réagir automatiquement aux changements des données
- ... la BdD SQL ne servait presque plus qu'à du stockage...

Mais besoin d'outils encore plus innovants : **prémises du Big Data analytics (Data Science) et du Big Data Engineering**


Motivation et émergence

Evolution des technologies de BdD traditionnelles




Deux types de requêtes / d'utilisation :

Requêtes OLTP :
 approche transactionnelle classique des BdD Relationnelles



Requêtes OLAP :
 besoin en analyse de données, peu favorable aux BdD Relationnelles

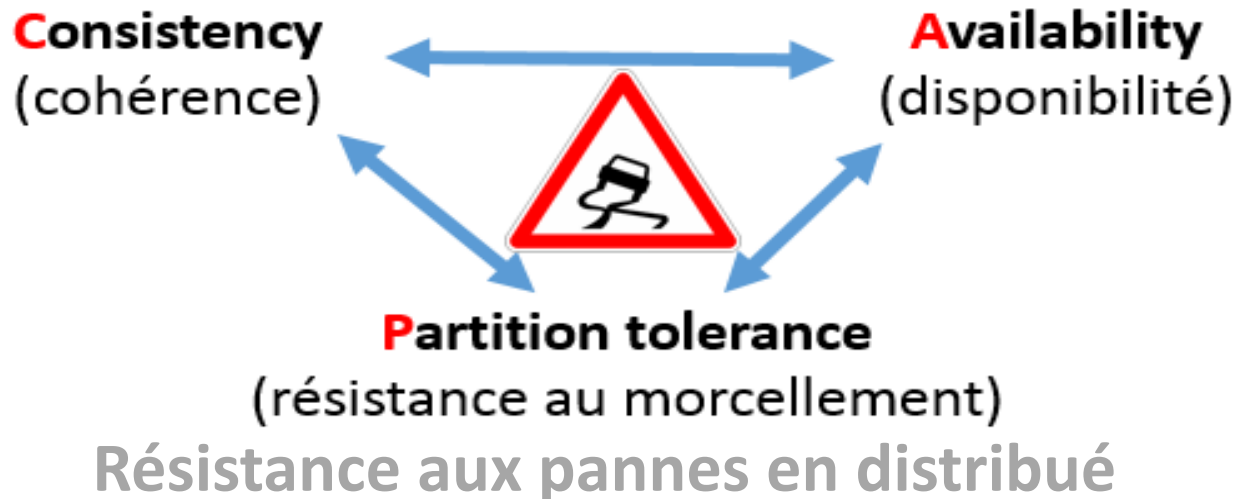


Motivation et émergence

Théorème / problème « CAP »

Base toujours perçue cohérente,
même pendant les mises-à-jour

Disponibilité garantie en
l'absence de pannes



En mode distribué à large échelle :

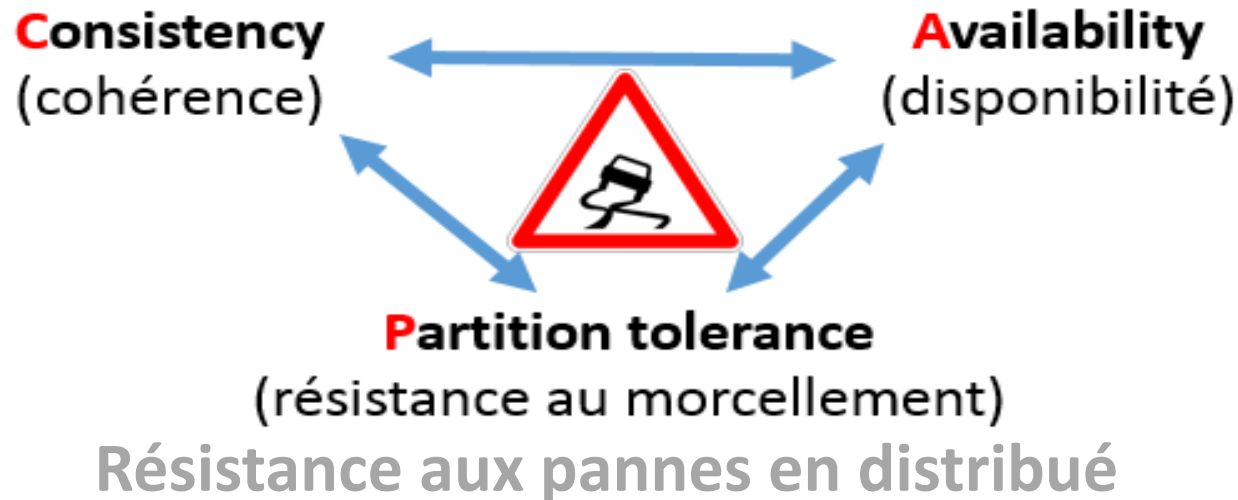
- on n'a jamais toutes les data à jour en même temps
- on ne peut pas différer des requêtes chaque fois qu'on fait une maj (on ne traiterai jamais de requêtes!)
- on ne peut pas arrêter de fonctionner dès que des parties de la BdD sont en pannes

Motivation et émergence

Théorème / problème « CAP »

Base toujours perçue cohérente,
même pendant les mises-à-jour

Disponibilité garantie en
l'absence de pannes

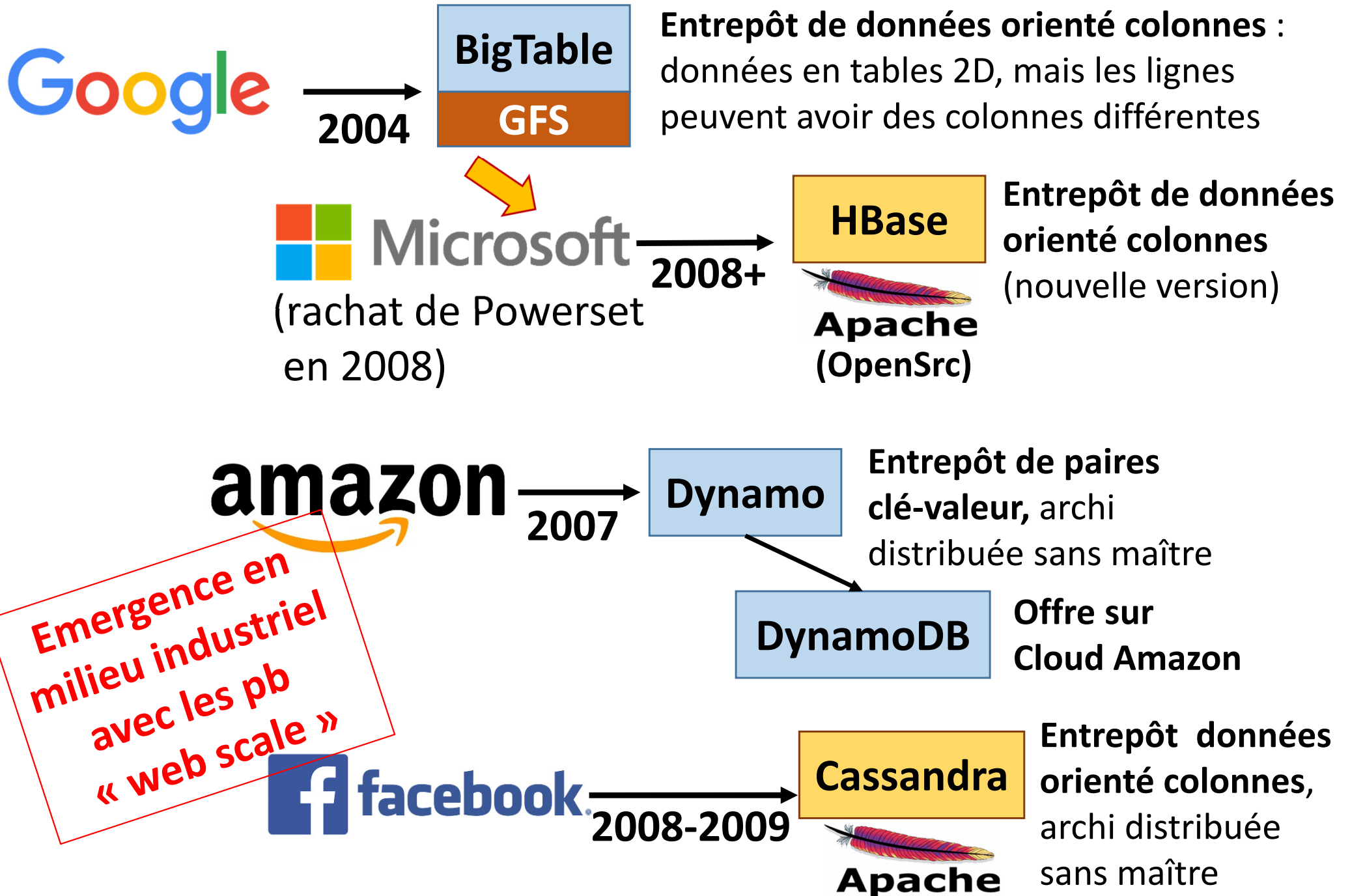


En mode distribué à large échelle :

- **on ne peut vérifier que 2 propriétés sur 3**
(théorème CAP, E. Brewer 2000-2002)
- **le Big Data et le NoSQL renoncent surtout à la garantie de *consistency***

**Mais on repousse
sans cesse la limite**

Motivation et émergence




Emergence et principes des BdD NoSQL


1. Motivations et émergence des BdD NoSQL
- 2. Principes des BdD NoSQL**
3. Aperçu des différentes BdD NoSQL
4. Le retour de SQL...
5. Format de données JSON

Principes du NoSQL

« NoSQL vs SQL » :

- **Grande souplesse dans le format des données stockées** (en résumé : pas de schéma!) 
- **Exploitation de très grosses volumétries en temps raisonnable** grâce au relâchement des contraintes d'intégrités et de cohérence
- **Augmentation des performances par la distribution massive** du stockage et des traitements : s'appuie sur un mécanisme *Map-Reduce* et un système de fichiers distribué

Mais beaucoup de critiques sur :

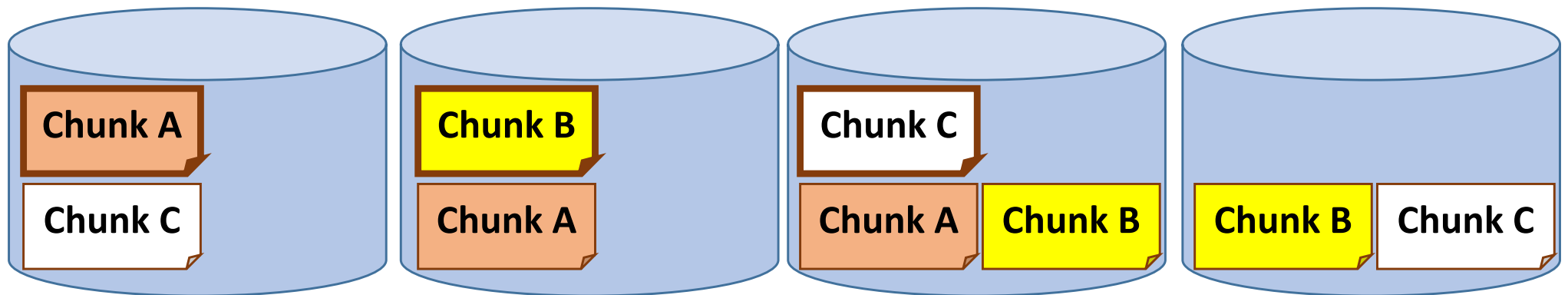
- **La faiblesse de performance** d'Hadoop et des premières architectures BigData...
- **La difficulté d'exploiter des données hétérogènes** (pas de schéma → complexifie la couche applicative) 

Sharding et réplication

Distribution (*sharding*) et réplication :

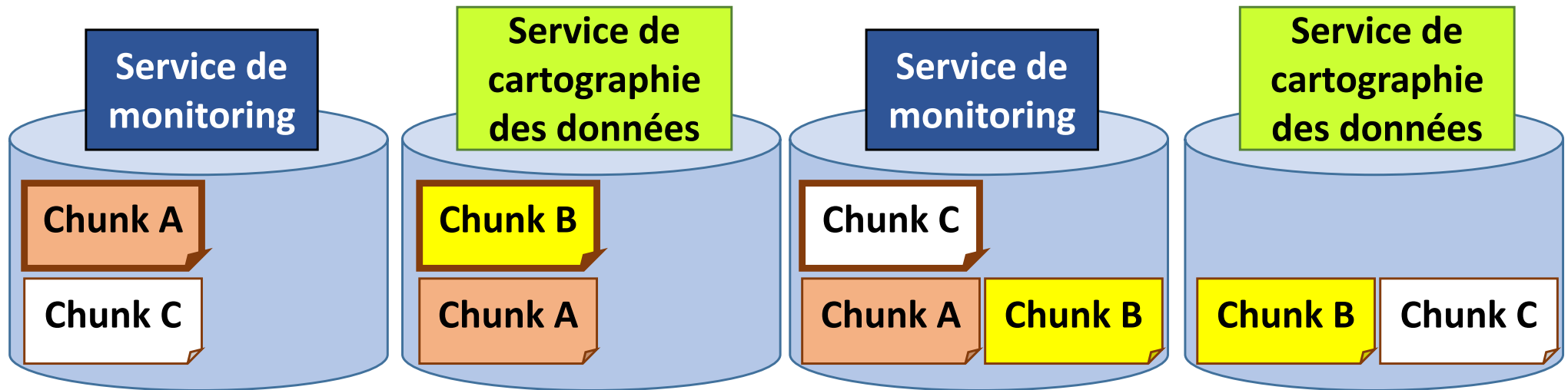
- Un fichier/une table est découpé en morceaux (*chunks*) distribués pour permettre:
 - des accès parallèles plus rapides (répartir la charge)
 - des stockages plus volumineux (et extensibles)
- Les *chunks* sont répliqués sur des nœuds différents pour la tolérance aux pannes

Concepts
d'HDFS



→ Le *sharding* est un découpage des différentes lignes d'une table par tranches dans différents *chunks* (au lieu de distribuer les colonnes)

Sharding et réplication



Des services de :

- cartographie du stockage
- *monitoring & heartbeat*

Réplication des services :

- pour la tolérance aux pannes

Concepts
d'HDFS

Stockage de données hétérogènes

Stockage beaucoup plus libre (qu'avec une BdD relationnelle) :

- Absence de schéma de base (!)
- Possibilité d'entrer des données :
 - hétérogènes
 - opaques ou structurées
- Possibilité de **modifier dynamiquement** la structures de certaines données

Plutôt facile de remplir une BdD NoSQL...
... beaucoup plus difficile de l'interroger !

Requêtes en *Map-Reduce*

Reproduction de certaines requêtes en *Map-Reduce* :

```
SELECT g(liste d'attr1), attr2
```

```
FROM relation
```

```
WHERE f(lignes de la relation)
```

```
GROUP BY attr2;;
```

3: **reduce**(g(), liste d'attr1)

0 : Data stockée en fichier HDFS

1: **map**(f(), liste des lignes de la relation)

2: **Shuffle & Sort**, groupement des lignes retenues selon attr2 (key)

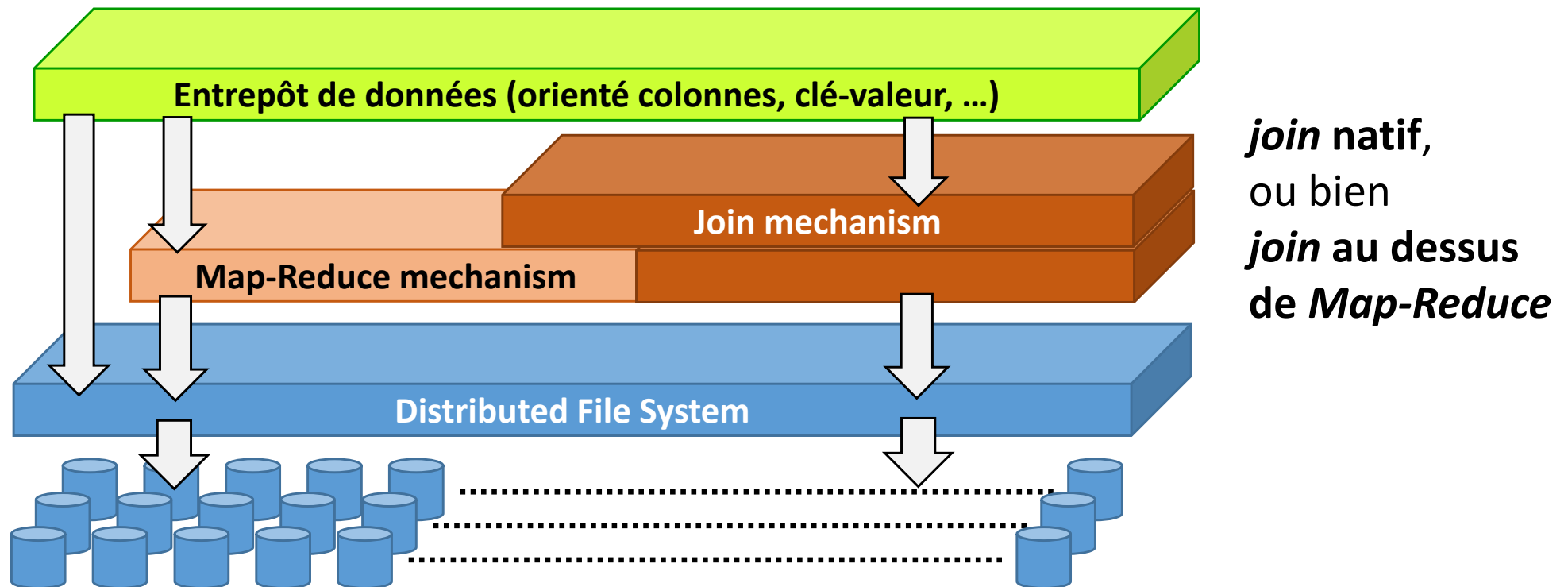
La solution « *Map-Reduce* » permet d'implanter facilement des requêtes « *Select-From-Where-GroupBy* » :

- sur un système distribué à grande échelle
- sur des données structurées complexes et/ou hétérogènes

Passe à l'échelle
Plus générique...
...plus compliqué!

Principes d'architecture NoSQL

Architecture de principe d'une BdD NoSQL :



Certaines BdD NoSQL sont bâties :

- Au dessus d'Hadoop : HDFS et *Map-Reduce* d'Hadoop
- Au dessus d'HDFS mais ajoutent leur propre couche *Map-Reduce*
- Au dessus d'une architecture complètement propre (**MongoDB**)

Variété des BdD NoSQL

Classification des BdD NoSQL :

- Stockage / Entrepôt de paires clé-valeur
([Redis](#), [Riak](#))
- BdD orientés documents
([MongoDB](#) → voir TD)
- BdD orientés colonnes
([BigTable](#), [HBase](#), [Cassandra](#))
- BdD créées pour des index inversés
([Elasticsearch](#))
- BdD orientés graphes
([Neo4J](#))

Emergence et principes des BdD NoSQL

1. Motivations et émergence des BdD NoSQL
2. Principes des BdD NoSQL
- 3. Aperçu des différentes BdD NoSQL**
4. Le retour de SQL...
5. Format de données JSON

Entrepôts (NoSQL) de paires clé-valeur

Particularités des entrepôts de paires clé-valeur :

La solution la plus **extensible** (« *scalable* »), mais **simple/pauvre** :

- Statistiquement, la plupart des applications demandent à lire des données à partir de leurs identifiants
→ engendre le besoin de BdD stockant des **paire clé-valeur**
- Le composant clé de ces bases est leur **fonction de hachage**
→ distribution et recherche des données dans le système distribué
- Mécanisme final très efficace mais avec **peu de fonctionnalités**
→ développements dans la couche applicative, en *Map-Reduce*

Rmq : Initialement des *valeurs* binaires et opaques, puis des valeurs structurées hiérarchiques analysables (ex : format JSON)

Entrepôts clé-valeur► **BdD orientées documents**

Particularités des BdD NoSQL orientées documents :

On associe des **clés** à des **documents à structure hiérarchique**

- Les valeurs ne sont plus opaques
- On peut manipuler les champs des données
- Manipulation de doc web au format HTML/XML, ou doc JSON

On stocke des données prêtes à être interrogées sans jointure :

- Exploitation rapide
- Mais la jointure doit être faite lors de l'écriture 
- Si besoin de croiser des informations : **plus complexe et lent**

Rmq : L'utilisation de documents structurés non opaque permet de les analyser et de produire des index inversés

Rmq : BdD devenues de très grandes tailles, proches des entrepôts de paires clé-valeur

Entrepôts clé-valeur ←... BdD orientées documents

BdD NoSQL orientées colonnes

Particularités des BdD NoSQL orientées colonnes :

Stockent des **tables 2D de clé - ensemble de valeurs**

Ressemblent à des tables relationnelles, mais bcp plus souples

- Les lignes peuvent avoir des colonnes différentes et en nombres différents
- Les colonnes d'une ligne peuvent évoluer dynamiquement en nombre et en nom
- Pas de champ « NULL » contrairement à une table relationnelle (pas de colonne inutile dans une ligne)

Des requêtes simples (minimalistes) *comparé à une* BdD SQL
Ou bien des traitements complexes en *Map-Reduce*...



Rmq : BdD NoSQL conçues pour stocker des associations
one-to-many comme on en trouve très fréquemment
sur le **web** !



BdD NoSQL orientées colonnes

Particularités des BdD NoSQL orientées colonnes :

| | | | | | | |
|------------|------------|--------|------------|--------|------------|--------|
| 100 | vente-2010 | 100000 | vente-2011 | 150000 | vente-2014 | 180000 |
| 200 | vente-2012 | 1000 | | | | |
| 211 | vente-2010 | 500000 | achat-2016 | 10000 | | |



Mise à jour : renommage et ajout de colonnes

| | | | | | | |
|------------|------------|--------|-------------------|-------------|------------|--------|
| 100 | vente-2010 | 100000 | vente-2012 | 150000 | vente-2014 | 180000 |
| 200 | vente-2012 | 1000 | | | | |
| 211 | vente-2010 | 500000 | achat-2014 | 3000 | achat-2016 | 10000 |

Chaque ligne peut avoir des colonnes différentes

Les colonnes d'une lignes peuvent évoluer dynamiquement

BdD NoSQL d'index inversés

Particularités des BdD NoSQL conçues pour des index inversés :

Un index inversé est indispensable pour traiter rapidement les requêtes de recherche de documents par mots clés

Mais l'index inversé peut devenir TRES volumineux (plus que les documents analysés) :

- il faut le compresser
- mais pas trop pour que la décompression à la volée soit rapide
- et/ou trouver un format de compression permettant de travailler dans le format compressé

Les BdD NoSQL spécialisées en index inversés apportent:

- des algorithmes optimisés pour construire ces index
- des algorithmes de compression/décompression adaptés

BdD NoSQL de graphes

Particularités des BdD NoSQL orientées graphes :

Spécialement adaptées pour fouiller le web et les réseaux sociaux

- Apportent des stockage de graphes efficaces (par références)
- Apportent des algorithmes d'analyse de graphes optimisés et adaptés au stockage réalisé

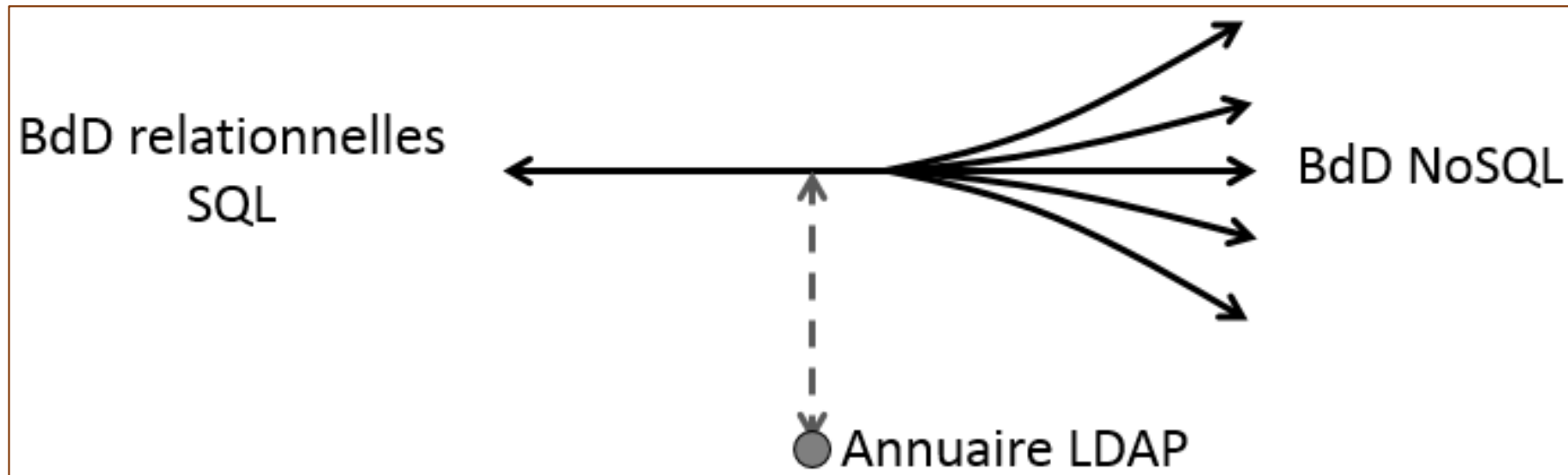
Les autres bases NoSQL pourraient stocker des graphes mais seraient moins efficaces pour les analyser

Rmq : **Neo4j** est un cas extrême de technologie très complète et très efficace pour stocker/fouiller/analyser des graphes

- Rapide (codage par "pointeurs")
- Stockage compact
- Répliquée sur cluster, mais pas distribuée...

Positionnement des annuaires LDAP

Annuaire LDAP vs NoSQL :



- Stockage hiérarchique des données (très utile pour décrire des SI)
→ non SQL, non relationnel
- Plus aboutis en sécurité/contrôle des accès que les BdD NoSQL
- Implantations anciennes, moins performantes que les BdD NoSQL
- Impose un schéma hiérarchique en arbre (avec possibilité de pontage)
→ contraignant comparé aux BdD NoSQL

Les annuaires LDAP sont « à mi-chemin » et « à part »

Emergence et principes des BdD NoSQL

1. Motivations et émergence des BdD NoSQL
2. Principes des BdD NoSQL
3. Aperçu des différentes BdD NoSQL
4. **Le retour de SQL...**
5. Format de données JSON

Le besoin de retrouver SQL

Très vite des critiques sur Map-Reduce et NoSQL

- Faiblesse des performances d'Hadoop et des premières architectures BigData...
- Difficulté d'exploiter des données hétérogènes dans des BdD NoSQL (*Le manque de schéma complexifie la couche applicative!*)



Une partie des utilisateurs de BdD & Big Data veulent retrouver SQL !

Ils veulent « juste » un SQL qui travaille à plus grande échelle !

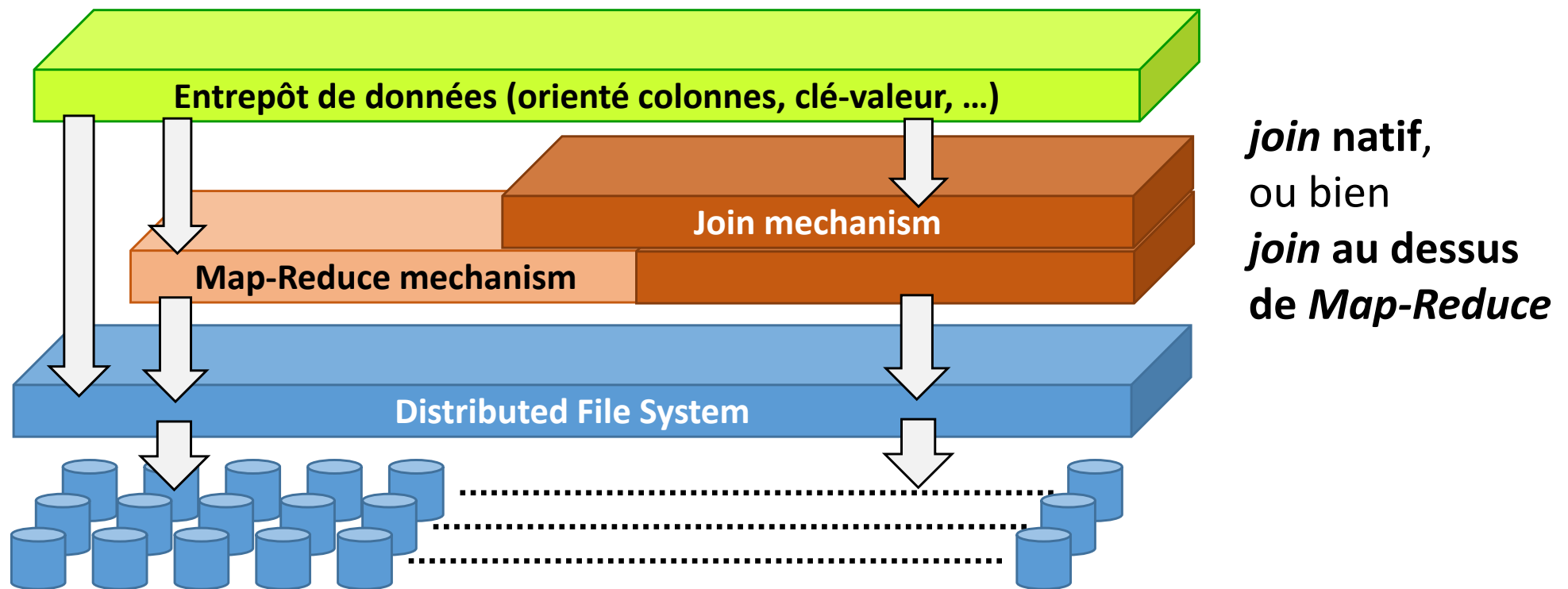
→ repousser le théorème CAP !

Principes d'architecture NoSQL

Architecture de principe d'une Bdd NoSQL :

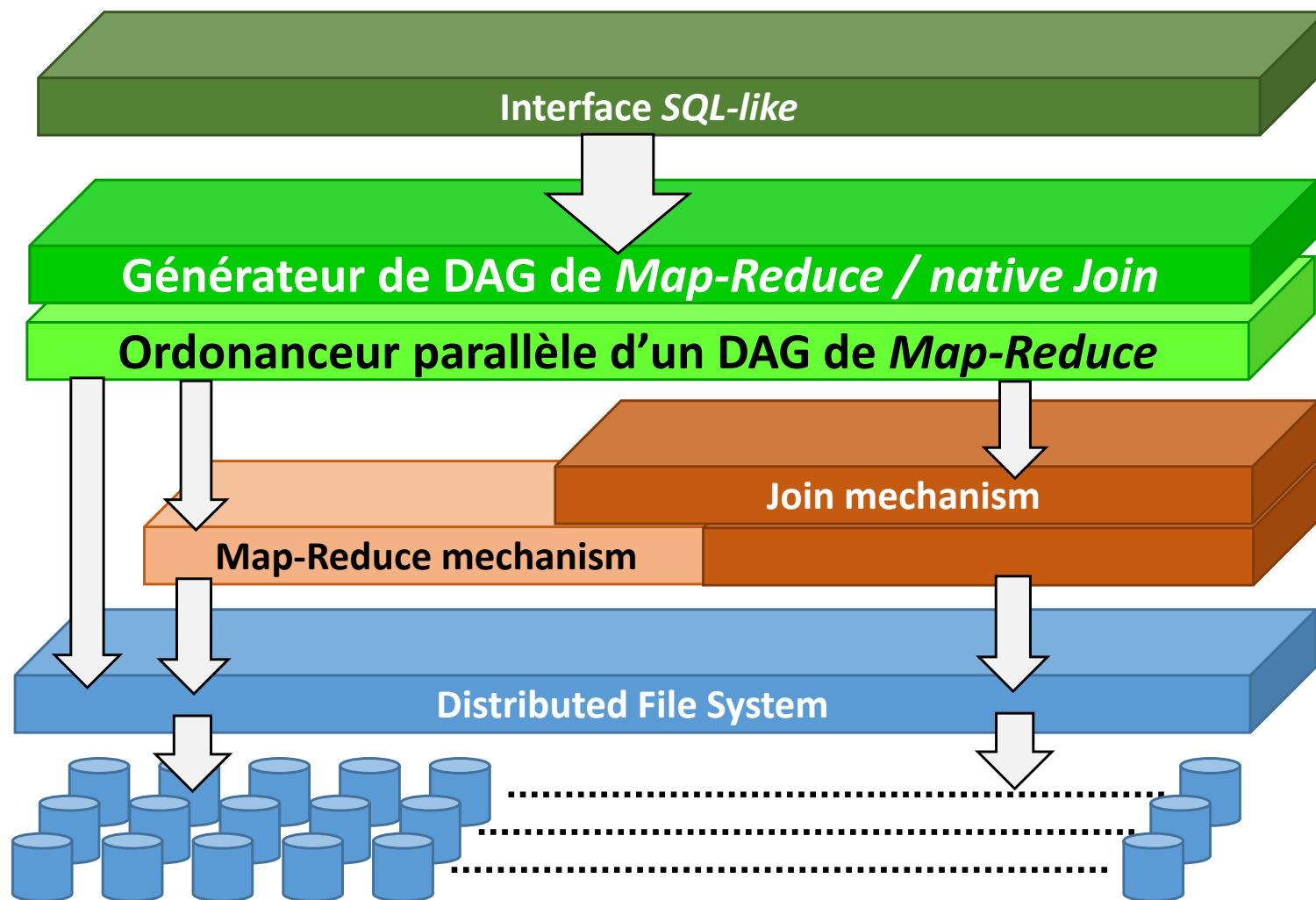
Au début les « entrepôts de données » (NoSQL) n'offraient pas les fonctionnalités de SQL... il fallait les interroger en Map-Reduce...

... et rapidement il a fallu porter SQL sur ces environnements :



NoSQL \rightarrow *SQL-like*

Principes d'une BdD **SQL** au dessus d'une architecture Big Data :



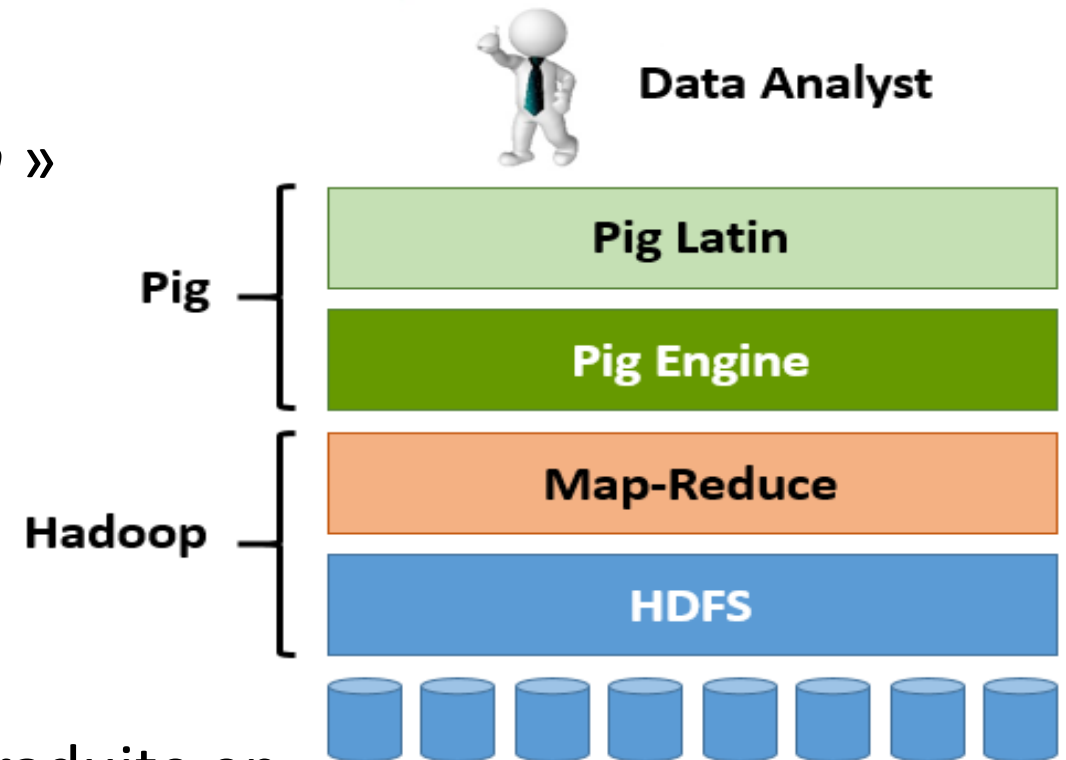
\rightarrow Traduction d'une *requête SQL* en graphe d'opérations *Map-Reduce*

\rightarrow Une BdD « ***SQL-like*** » à très large échelle sur un HDFS

NoSQL → SQL-like : PIG

En 2006 **Yahoo!** développe **PIG** pour faire « un peu comme SQL » au dessus d'Hadoop :

- pour simplifier le travail de ses équipes de *data-analysts*
- devient OpenSource en 2007
- Langage de requêtes et de programmation : « *Pig-Latin* »
- Ex de commandes :
 FILTER, GROUP BY,
 JOIN, SORT ...
 FOREACH...



... et chaque commande est traduite en un *graphe d'opérations Map-Reduce*

NoSQL → *SQL-like* : PIG

Exemple 1/3 :

```
grunt> DUMP ETUDIANT;
```

```
(Dupond, Alphonse, 10, Metz)
```

```
(Dupond, Grégoire, 9, Rennes)
```

```
(Durant, Hubert, 12, Gif)
```

```
(Talon, Achille, 15, Gif)
```

```
grunt> ETUDIANT2 = FOREACH ETUDIANT GENERATE $0, $1, $2+1 ;
```

```
grunt>DUMP ETUDIANT2;
```

```
(Dupond, Alphonse, 11)
```

```
(Dupond, Grégoire, 10)
```

```
(Durant, Hubert, 13)
```

```
(Talon, Achille, 16)
```

NoSQL → *SQL-like* : PIG

Exemple 2/3 :

```
grunt> DUMP VEHICULE;
```

```
(AA 123 AB, bleue, Twingo)
```

```
(AZ 451 GT, noire, C3)
```

```
(BC 634 FY, jaune, Twingo)
```

```
(DE 398 AA, blanche, DS4)
```

```
grunt>DUMP CONSTRUCTEUR;
```

```
(Twingo, Renault)
```

```
(C3, Citroen)
```

```
(DS4, Citroen)
```

```
grunt> VOITURE = JOIN VEHICULE BY $2, CONSTRUCTEUR BY $0;
```

```
grunt>DUMP VOITURE;
```

```
(AA 123 AB, bleue, Twingo, Twingo, Renault)
```

```
(AZ 451 GT, noire, C3, C3, Citroen)
```

```
(BC 634 FY, jaune, Twingo, Twingo, Renault)
```

```
(DE 398 AA, blanche, DS4, DS4, Citroen)
```


NoSQL → *SQL-like* : PIG

Exemple 3/3 :

```
grunt> VOITURE2 = FOREACH VOITURE GENERATE $0, $1, $2, $4 ;
```

```
grunt> VOITURE3 = ORDER VOITURE2 BY $2, $0 DESC ;
```

(DE 398 AA, blanche, DS4, Citroen)

(BC 634 FY, jaune, Twingo, Renault)

(AZ 451 GT, noire, C3, Citroen)

(AA 123 AB, bleue, Twingo, Renault)

NoSQL → *SQL-like* : PIG

Bilan de PIG

Exécution de PIG :

- Traduit les requêtes en **DAG d'opérations *Map-Reduce***
- Exprime un **maximum de parallélisme** entre les opérations du DAG (souci de performance)
- Le « PIG-engine » exécute le DAG, et **fixe le nombre de *Reducers au mieux*** (mais on peut le guider)

Interface PIG-Latin :

SQL : est un langage déclaratif → on dit ce que l'on veut obtenir
alors que :

Pig-Latin : reste un langage procédural → on dit quoi faire
(pour obtenir le résultat)

NoSQL → *SQL-like* : Hive

En 2005 Facebook crée Hive : « langage déclaratif proche de SQL »

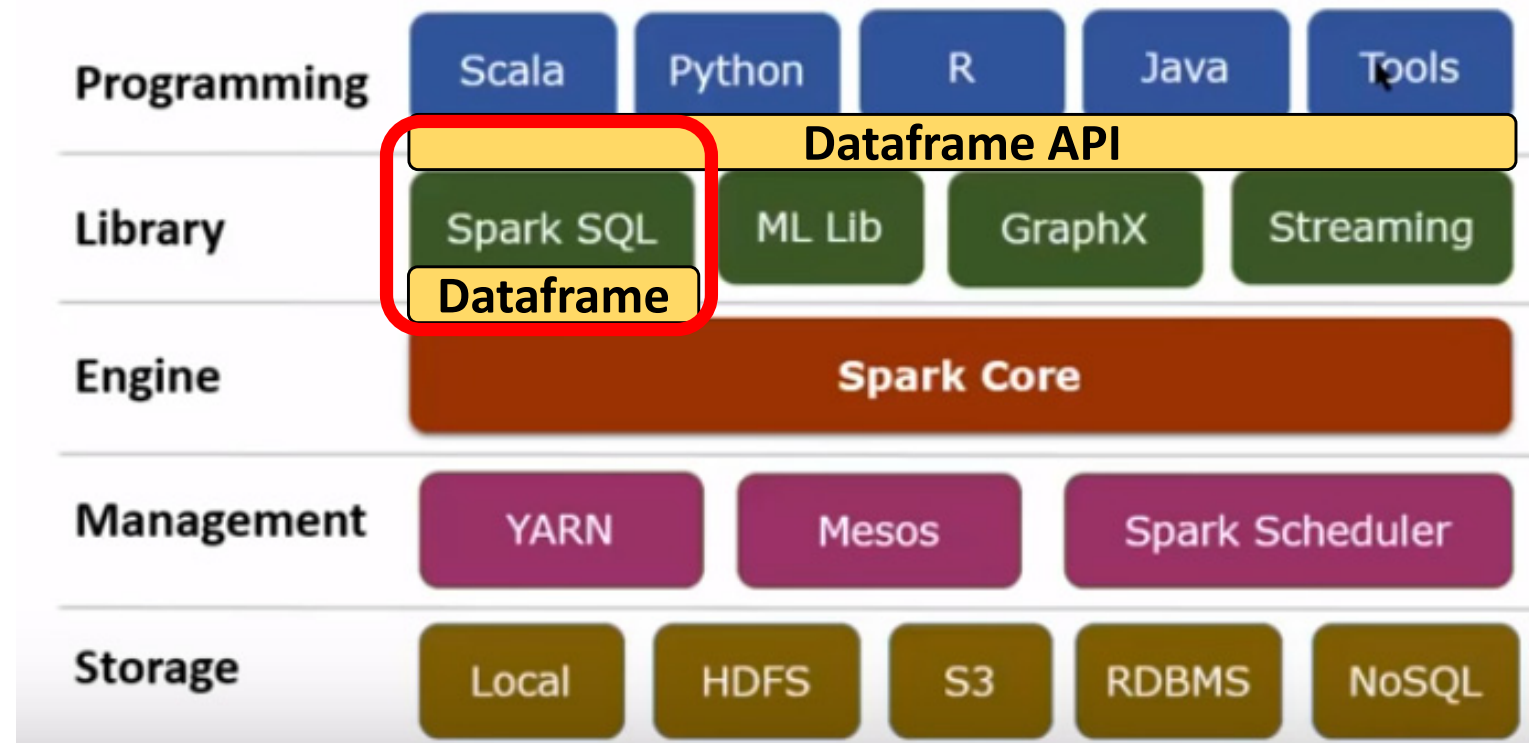
- Encore plus proche de SQL que PIG
- Pour des *data-analysts* qui ne peuvent pas programmer du *Map-Reduce* (pas programmer en langage procédural)
- ... et Hive génère des graphes de *Map-Reduce* au dessus d'Hadoop.

NoSQL → SQL-like : Spark-SQL

Une implantation très avancée de SQL au dessus de Spark

- Dans la distribution standard de Spark
- Utilise les DataFrames (au dessus des RDDs)

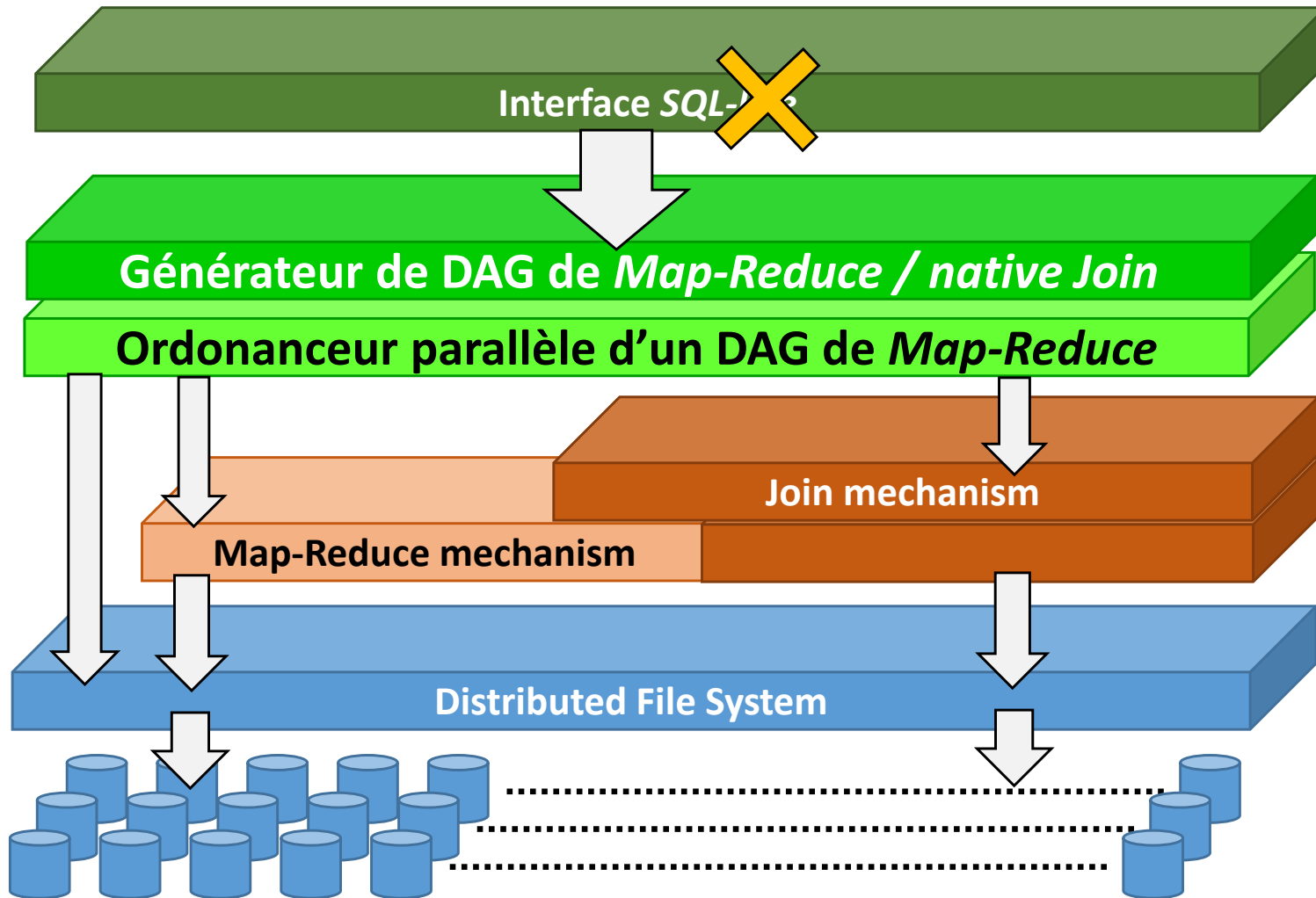
Spark Framework



Voir 2^{ème} partie du cours sur les Bdd SQL et NoSQL

NoSQL → SQL-Like → SQL

SQL entièrement disponible au dessus d'une architecture Big Data



→ Traduction d'une requête SQL en graphe d'opérations Map-Reduce

→ BdD « ~~SQL-Like~~ » à très large échelle sur un Map-Reduce + ~~HDFS~~ ^{XXX}

De SQL à SQL en passant par le BigData

Web, GAFA → « Web Scale » data

Les Bdd SQL ne suffisent plus :

- trop faible volumétrie
- manque de capacités « analytics »

Map-Reduce & Distributed File System

Programmation
Map-Reduce

Bdd large échelle et moins contraignantes

Bdd NoSQL sur
Map-Reduce

Retrouver la simplicité et fiabilité de SQL

SQL sur Map-Reduce

Emergence et principes des BdD NoSQL

1. Motivations et émergence des BdD NoSQL
2. Principes et variété des BdD NoSQL
3. Aperçu des différentes BdD NoSQL
4. Le retour de SQL...
5. **Format de données JSON**

Format JSON

Définition

JSON : **J**ava**S**cript **O**bject **N**otation

Un format de données **texte/ASCII**, **structuré** et **hiérarchique**

Collection JSON (ou objet JSON)

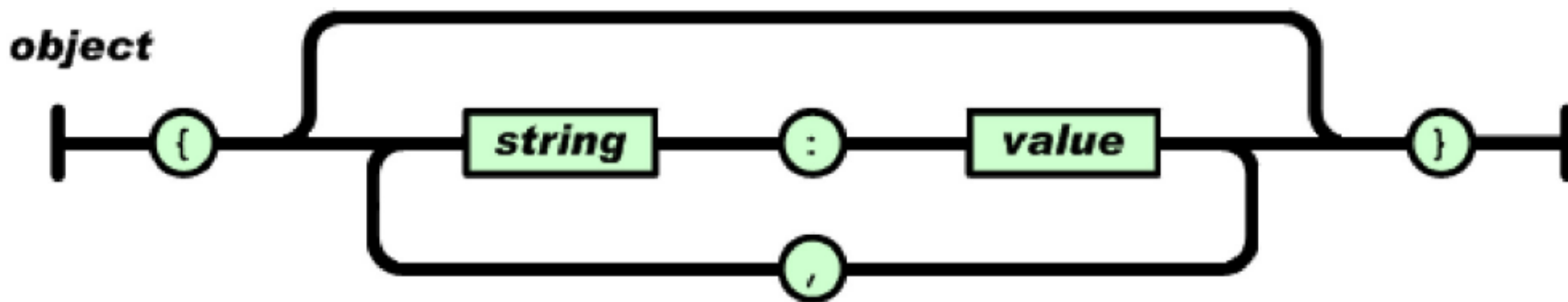
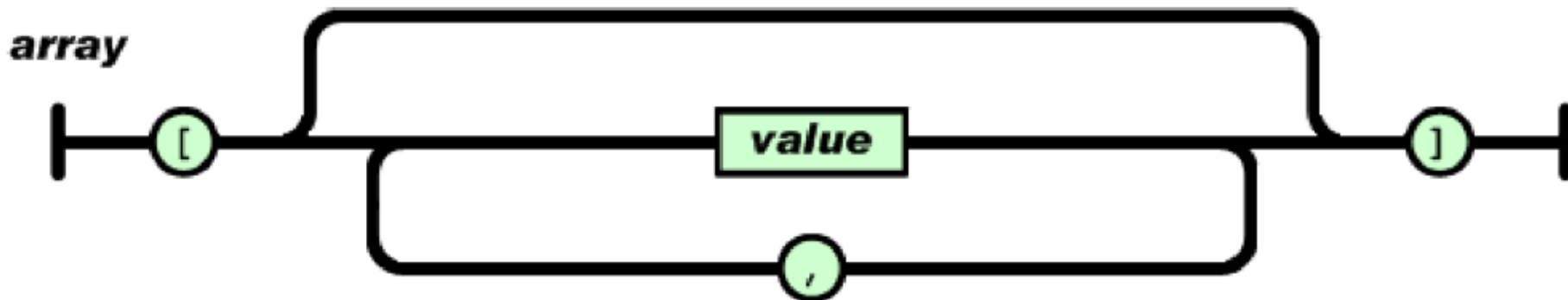


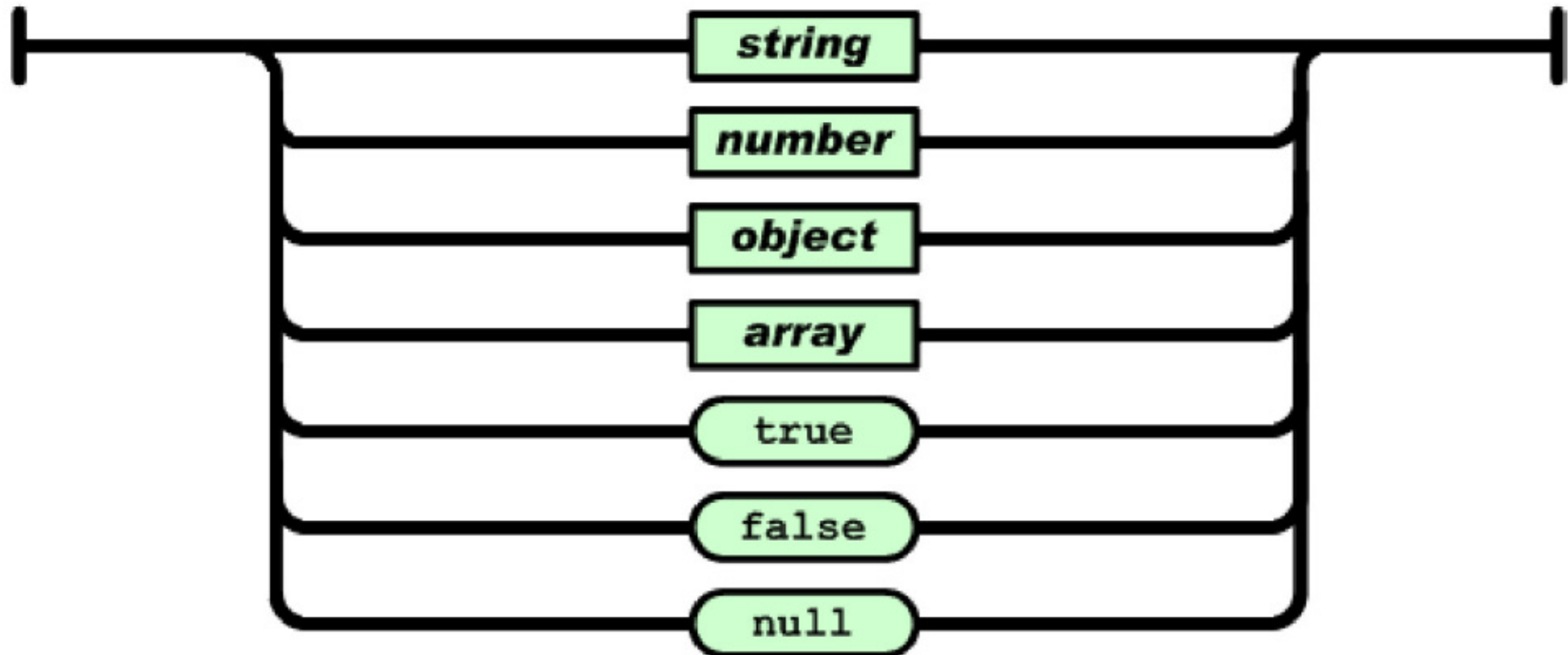
Tableau JSON



Format JSON

Valeurs JSON

value



Format JSON

Données textuelle,
structurées et
hiérarchique

```
{
  "livre" : {
    "Vue de l'esprit" : [ {"auteurs" : ["Hofstadter", "Dennet"] },
                        {"éditions" : "InterEditions"},
                        {"année" : 1987}
                      ],
    "Eagle" : [ {"auteurs" : ["Kidder"] },
                {"éditions" : "Flammarion"},
                {"année" : 1982}
              ]
  },
  "film" : {
    "2001 odyssée de l'espace" : [ {"réalisateurs" : ["Kubrick"] },
                                    {"année" : 1968}
                                  ]
  },
  "livre" : {
    "Cosmos" : [ {"auteurs" : ["Sagan"] },
                 {"éditions" : "Mazarine"},
                 {"année" : 1981}
               ]
  }
}
```

Emergence et principes des Bdd NoSQL

