Initiation aux Big Data

Cédric du Mouza et Nicolas Travers

CNAM – Mastère DéSiGéo

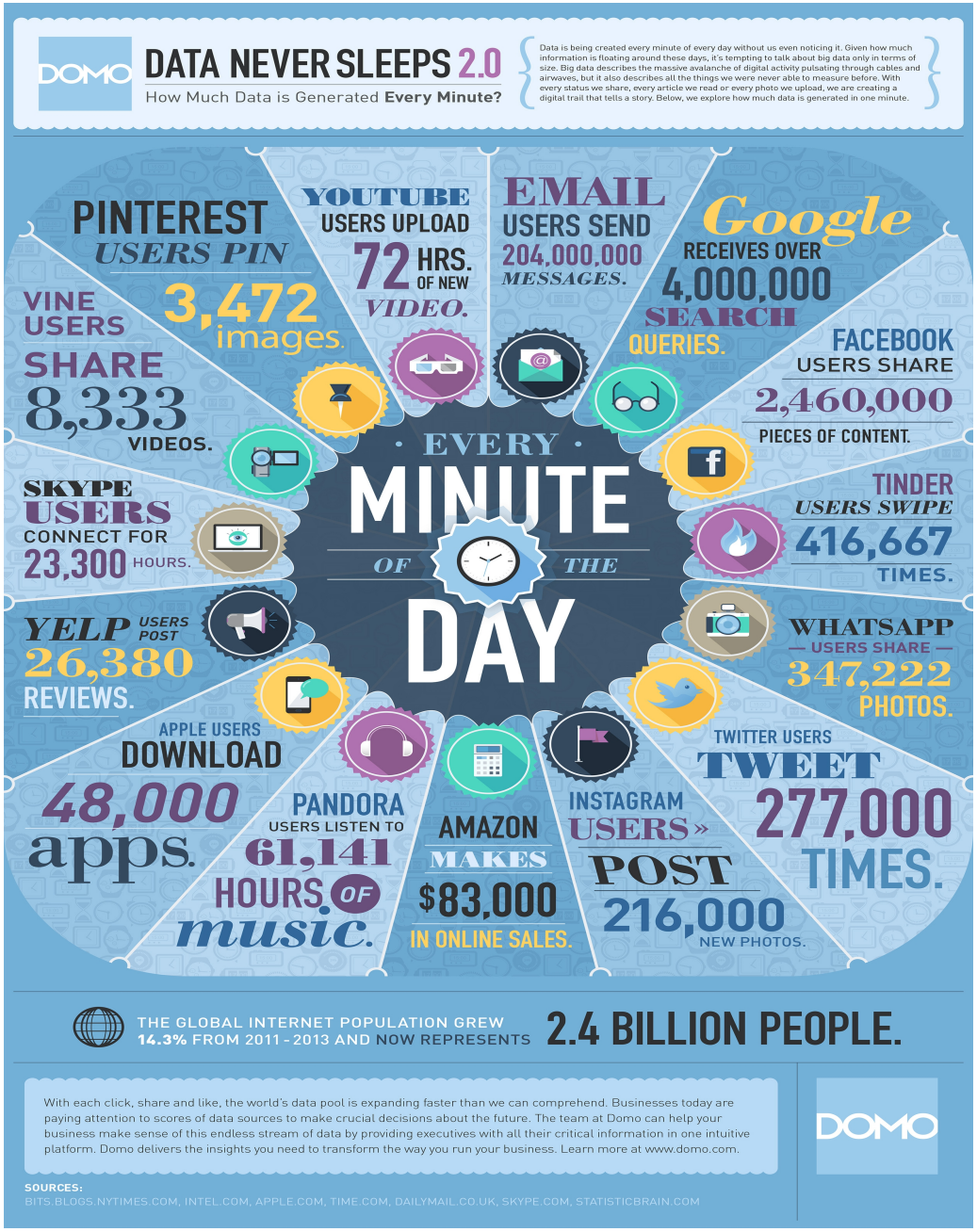
Organisation du cours

- Cours 1: Introduction aux Big Data et Map/Reduce
- Cours 2: NOSQL et MongoDB
- Cours 3: TP MongoDB
- Cours 4: Les BD graphes
- Cours 5: TP Neo4J (ou autre BD graphes)
- Cours 6: TP MapReduce (?)

Plan

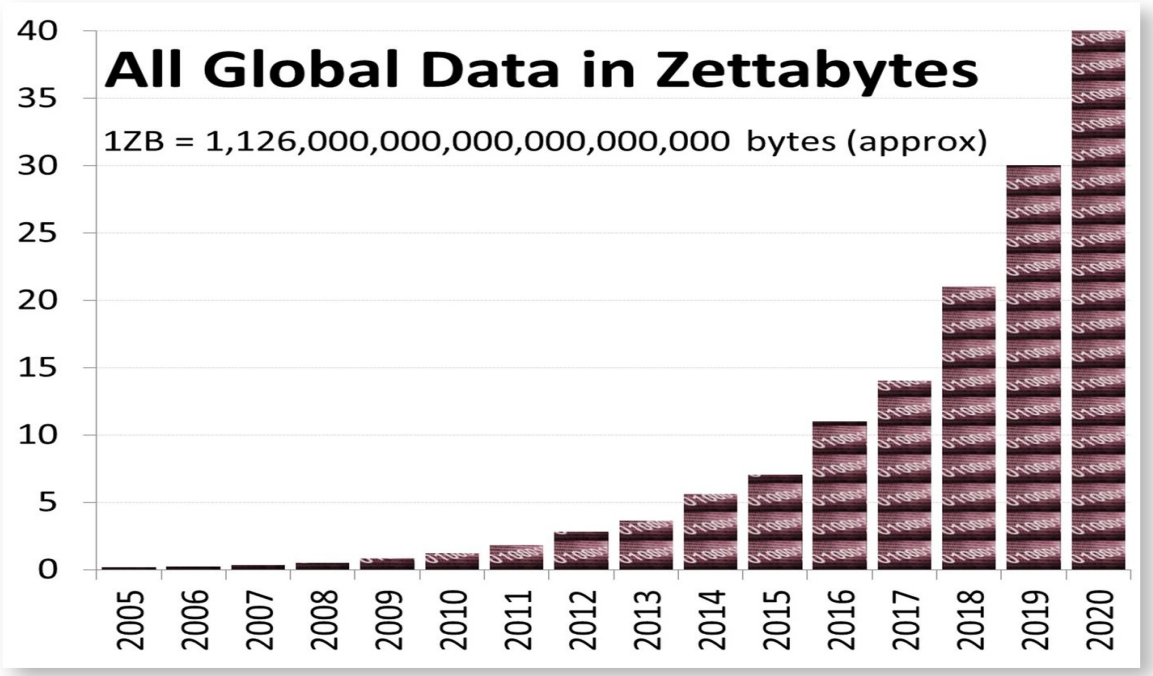
Contexte (1)

Croissance de la quantité des données exponentielle



Contexte (2)

- La quantités de données digitales produites double tous les 2 ans.
- En d'autres termes, on a produit autant de données digitales ces 2 dernières années que tout ce qui a été produit auparavant.



Contexte (3) : encore des chiffres

Compagnies	Données traitées (2014)	Données stockées (2014)
Google	100 Po	15 000 Po
Ebay	100 Po	90 Po
Facebook	600 To	300 Po
Twitter	100 To	100 To
Baidu	10-100 Po	2 000 Po
NSA	29 Po	10 000 Po

Contexte (4) : et le business?

- on estime que le volume de données professionnelles double tous les 1,2 ans
- les ¾ des décideurs estiment que les Big Data vont affecter significativement leurs systèmes de stockage
- le Big Data serait un marché à 50 milliards de $ en 2017
- en Europe, l'utilisation du Big Data pour améliorer l'efficacité des “traitements” permettrait d'économiser 100 milliards de $

Contexte (5)

Les volumes à gérer sans précédents impliquent:

- Données hétérogènes, complexes et souvent liées
  - produites par des applications parfois différentes,
  - par des utilisateurs différents,
  - avec des liens explicites (par exemple citations, ancres url, etc) ou implicites (à extraire ou à apprendre)

- Nombreux serveurs/clusters
  - un serveur unique ne peut stocker cette quantité d'information, garantir des temps d'accès pour grand nombre d'utilisateur, faire des calculs rapides, etc

- Besoin de distribuer les calculs et les données
  - comme plusieurs serveurs/clusters, besoin d'algorithmes permettant le calcul et la distribution des données à large échelle

Contexte (6) : Exemples

Data centers de quelques grands acteurs du Big Data :

- Google DataCenter :
  - 70000 servers/data center et 16 data centers, ~1M de serveurs
- Facebook :
  - 5 data centers
- Amazon :
  - 7 data centers, 450 000 severs
- Microsoft :
  - environ 1M serveurs

Big Data : Exemples d'utilisation

- Décodage du génôme humain: le génôme d'une personne (env. 100Go) décodé en 30mns
- Prédiction des résultats des élections US en 2012 à partie d'analyse de tweets
- Découverte d'un effet secondaire dû à la prise de deux médicaments par analyse des requêtes d'internautes (Yahoo)
- Étude des déplacements de population (migration, tourisme, circulation urbaine, etc)

SGBDR : Limitations

- Fonctionnalités
  - Jointures entre les tables
  - Construction de requêtes complexes
  - Contraintes d'intégrité solides

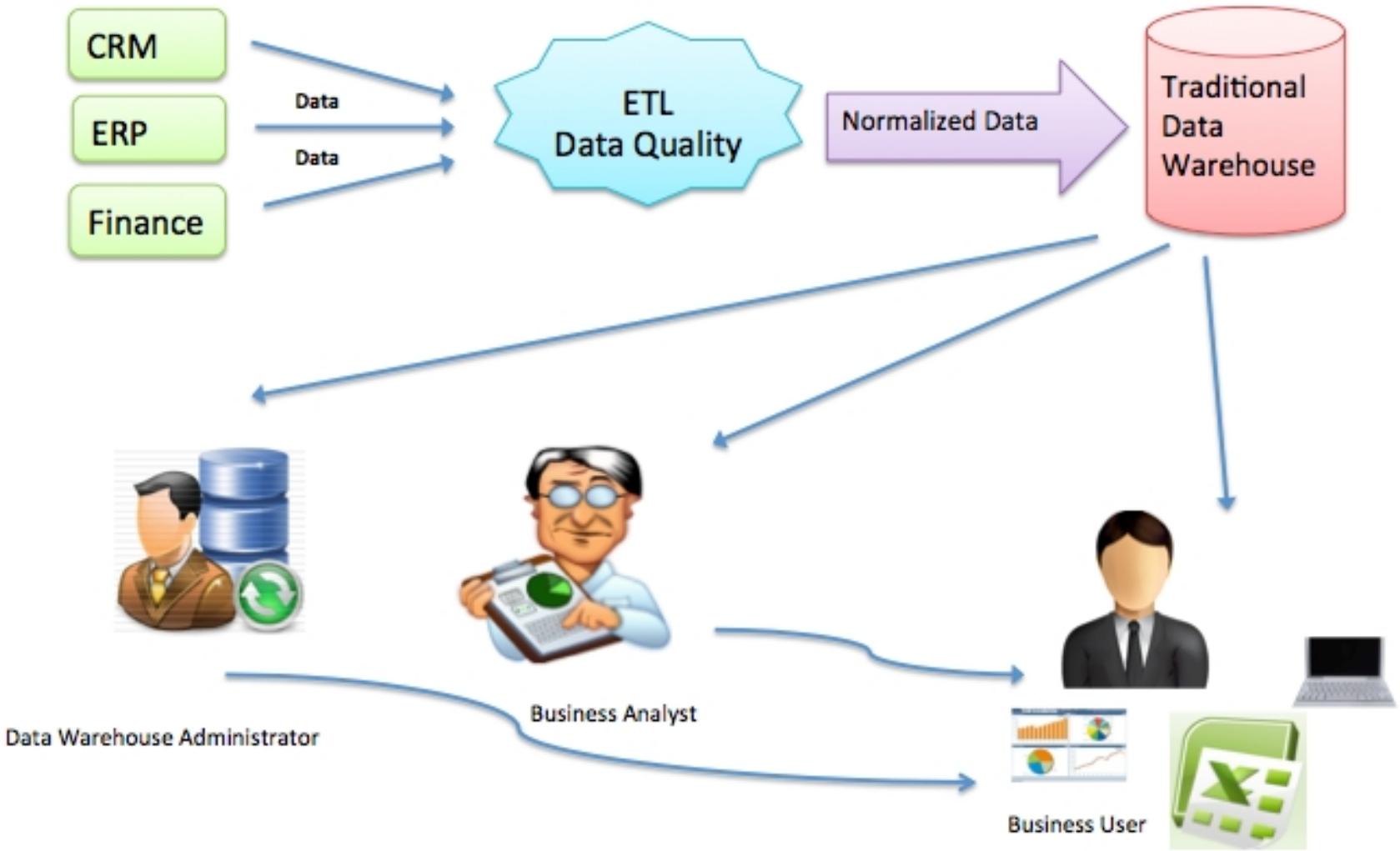
- Limites dans le contexte distribué: comment distribuer/partitionner les données
  - Liens entre entités -> Même serveur
  - Mais plus on a de liens, plus le placement des données est complexe

SGBDR : Limitations (2)

- Propriétés **ACID** pour les transactions
  - ***Atomicité***: une transaction s'effectue entièrement ou pas du tout
  - ***Cohérence***: le contenu d'une base doit être cohérent au début et à la fin d'une transaction (mais pas forcément durant son exécution)
  - ***Isolation***: les modifications d'une transaction ne sont visibles/modifiables que quand celle-ci a validé
  - ***Durabilité***: une fois la transaction validée, l'état de la base est permanent (non affecté par les pannes ou autre)

- Contexte distribué :
  - Contraintes ACID très complexes à assurer (techniques de verrouillages distribués par exemple)
  - Incompatible avec les performances

Décisionnel: ancienne méthode



Décisionnel: limites (les 3V du Big Data)

L'approche classique ne permet pas de gérer:

- Le **volume**: les entrepôts sont conçus pour gérer des Go ou To de données alors que la croissance exponentielle des données nous conduit aux Po ou Eo
- Le **type (variety)**: le nombre de types, incluant les données textuelles semi ou non structurées, augmente
- La **vitesse (velocity)**: les données sont créées de plus en plus vite et nécessitent des traitements en temps-réel

ACID vs BASE

- Systèmes distribués modernes utilisent le modèle BASE
  - *B*asically *A*vailable : garantie minimale pour taux de disponibilité face grande quantité de requêtes
  - *S*oft-state : l'état du système peut changer au cours du temps même sans nouveaux inputs (cela est du au modèle de consistance).
  - *E*ventually consistent : tous les réplicats atteignent le même état, et le système devient à un moment consistant, si on stoppe les inputs

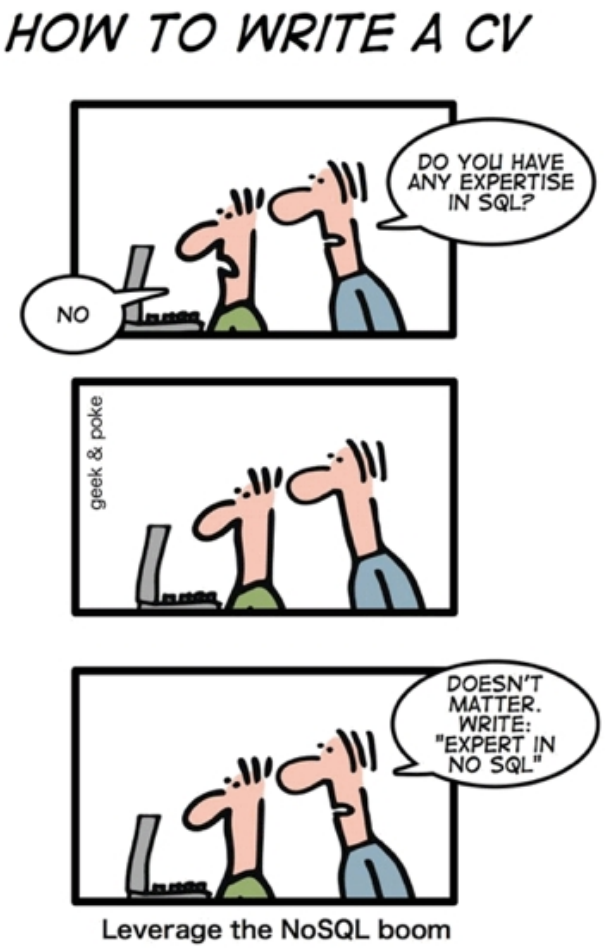

BIG DATA is GOING TO IMPACT US SOON!
CALL BRUCE WILLIS!!
DATA
www.nuxeo.com
Soulcié

La Solution : NoSQL

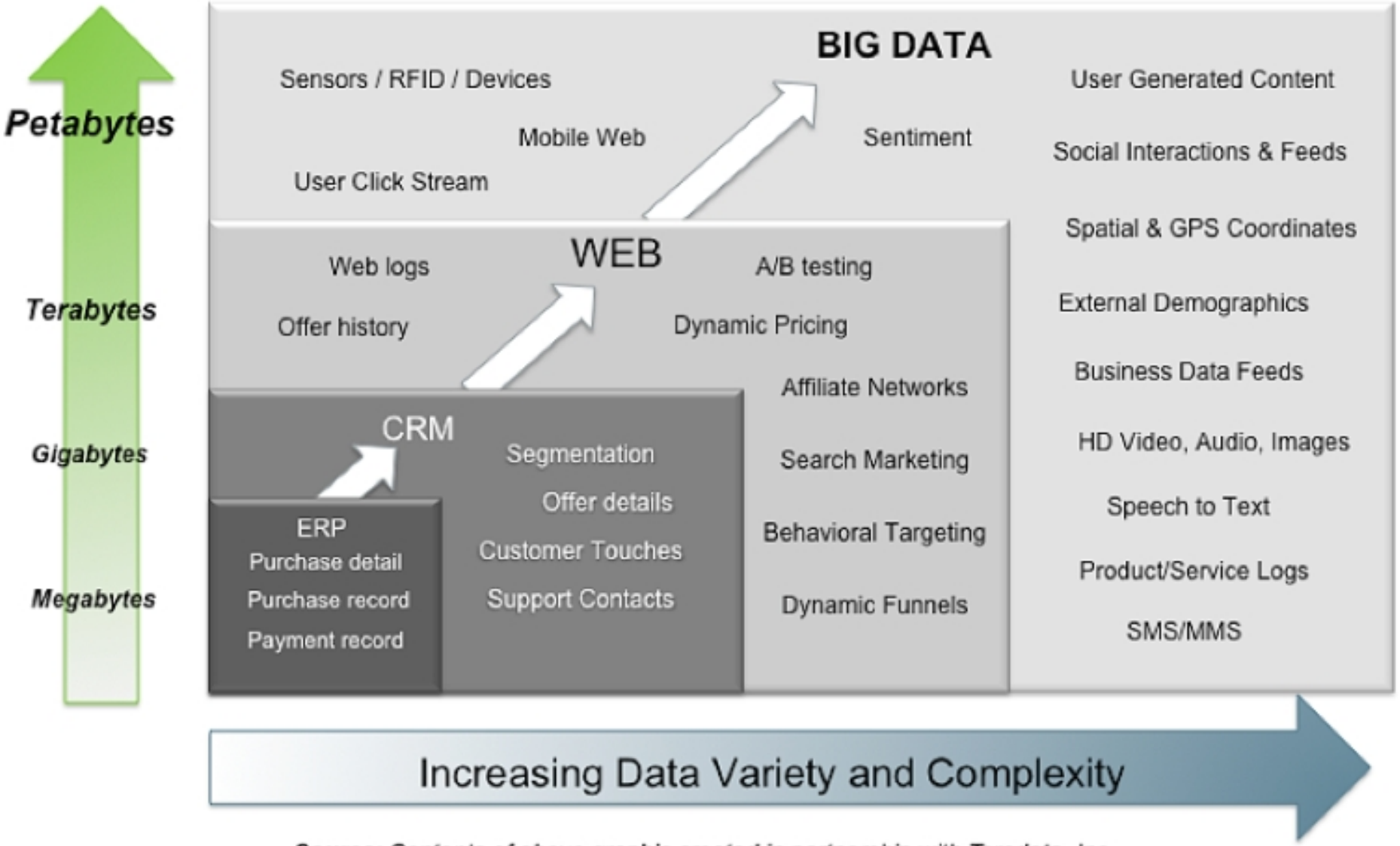
- NoSQL : **Not Only SQL**
  - Nouvelle approche de stockage et de gestion de données
  - Permet le passage à l'échelle via un contexte hautement distribué
  - Gestion de données complexes et hétérogènes
    - Pas de schéma pour les objets

- Ne remplace pas les SGBDR !!
  - Quantité de données énorme (PétaBytes)
  - Besoin de temps de réponse
  - Cohérence de données faible

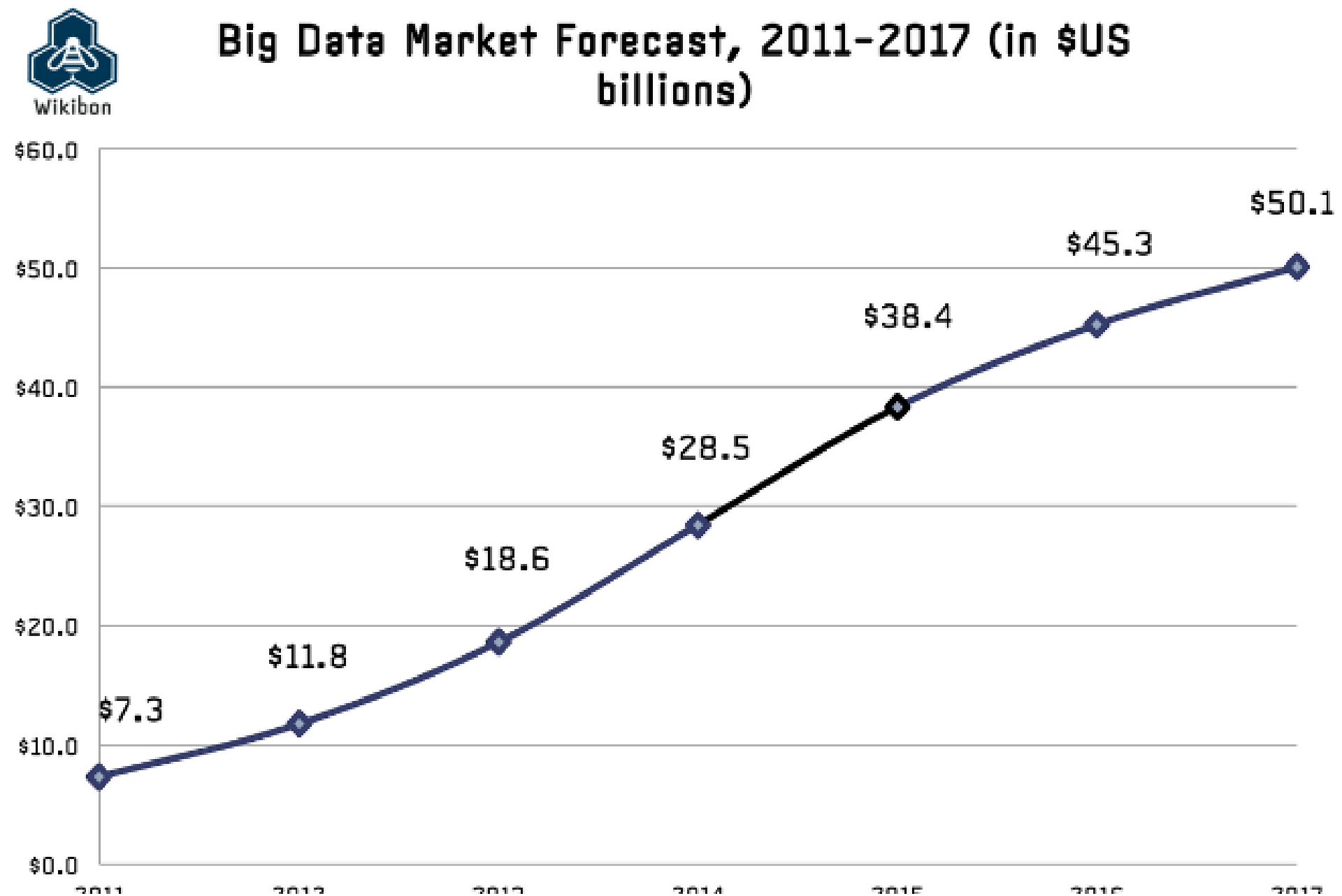
NoSQL ne remplace pas les SGBDR



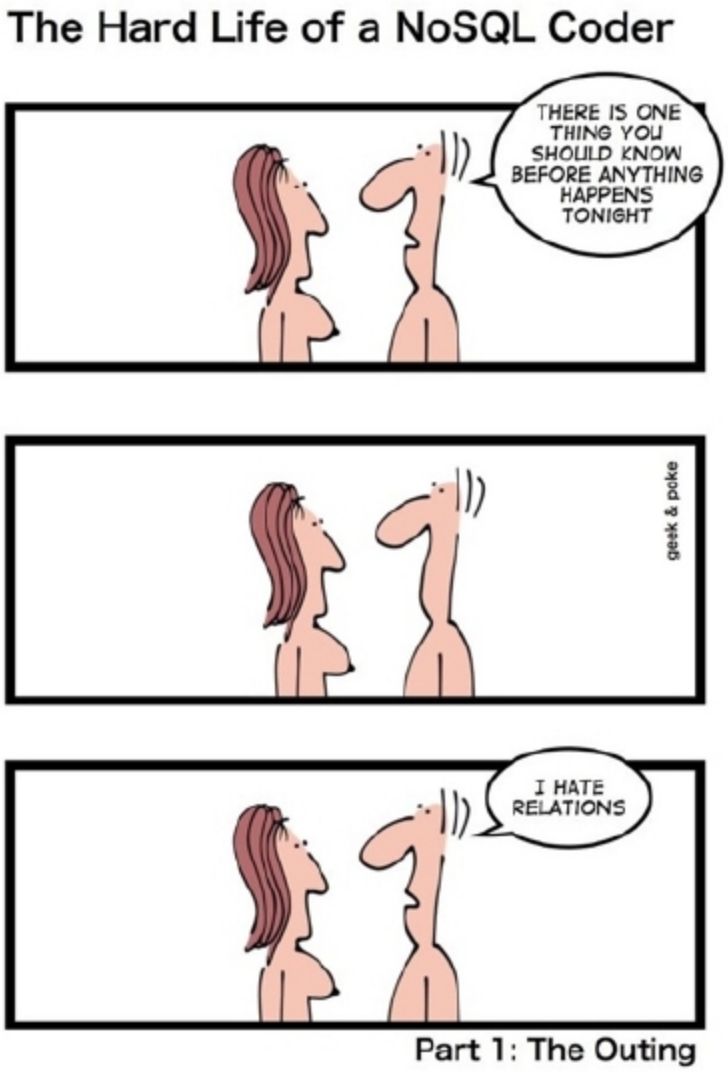
Big Data = Transactions + Interactions + Observations



***Source**: Contents of above graphic created in partnership with Teradata, Inc.*


Wikibon
Big Data Market Forecast, 2011-2017 (in $US billions)
$60.0
$50.0
$40.0
$30.0
$20.0
$10.0
$0.0
$7.3
$11.8
$18.6
$28.5
$38.4
$45.3
$50.1

Les bases de données NoSQL



NoSQL BD : Principales Caractéristiques

- Pas de relations
  - Pas de schéma physiques ou dynamiques
- Données complexes
  - Imbrication, tableaux
- Distribution de données (milliers de serveurs)
  - Parallélisation des traitements (Map/Reduce)
- Replication des données
  - Disponibilité vs Cohérence (pas de transactions)
  - Peu d'écritures, beaucoup de lectures

Partitionnement des données (1)

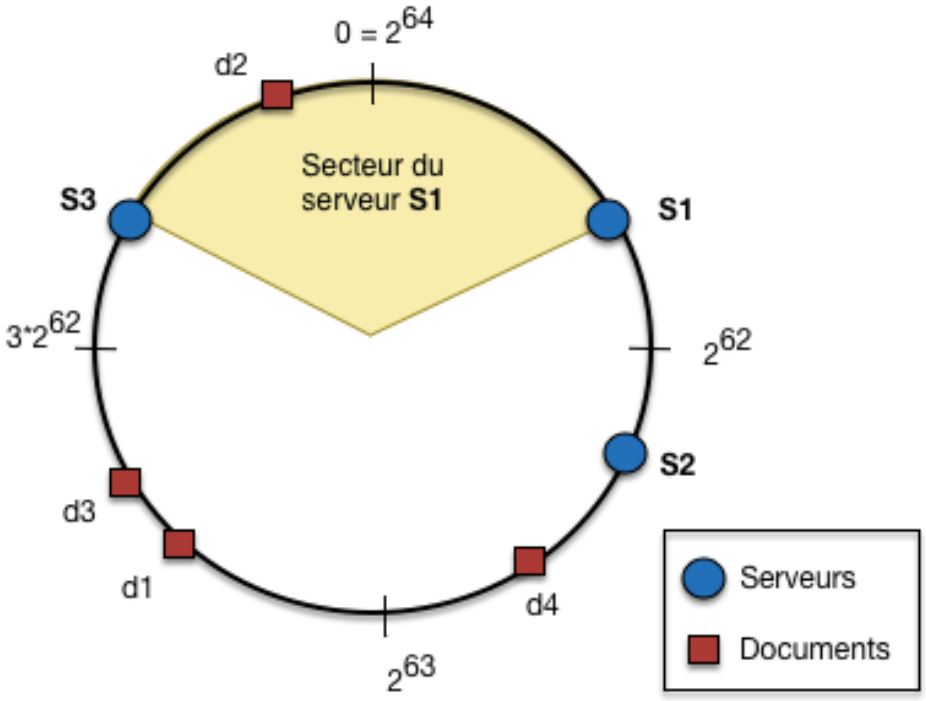
Le sharding

- Mécanisme de partitionnement horizontal (par tuple)
- Fonction de hachage pour déterminer le placement de paquets de données sur un ensemble de serveurs
  - *Mais éventuellement on peut essayer de placer sur le même serveur les données liées et/ou devant être mises à jour en même temps*
  - *Mais on peut essayer d'équilibrer la charge des serveurs*

Partitionnement des données (2)

Consistent Hashing (hachage cohérent)

- Technique de hachage unique et dynamique pour les données et les serveurs
- Distribution en anneau (virtuel)



Mises à jour

Gestion concurrence multi-version

- Idem que dans SGBD pour accès à une ressource en lecture ou écriture

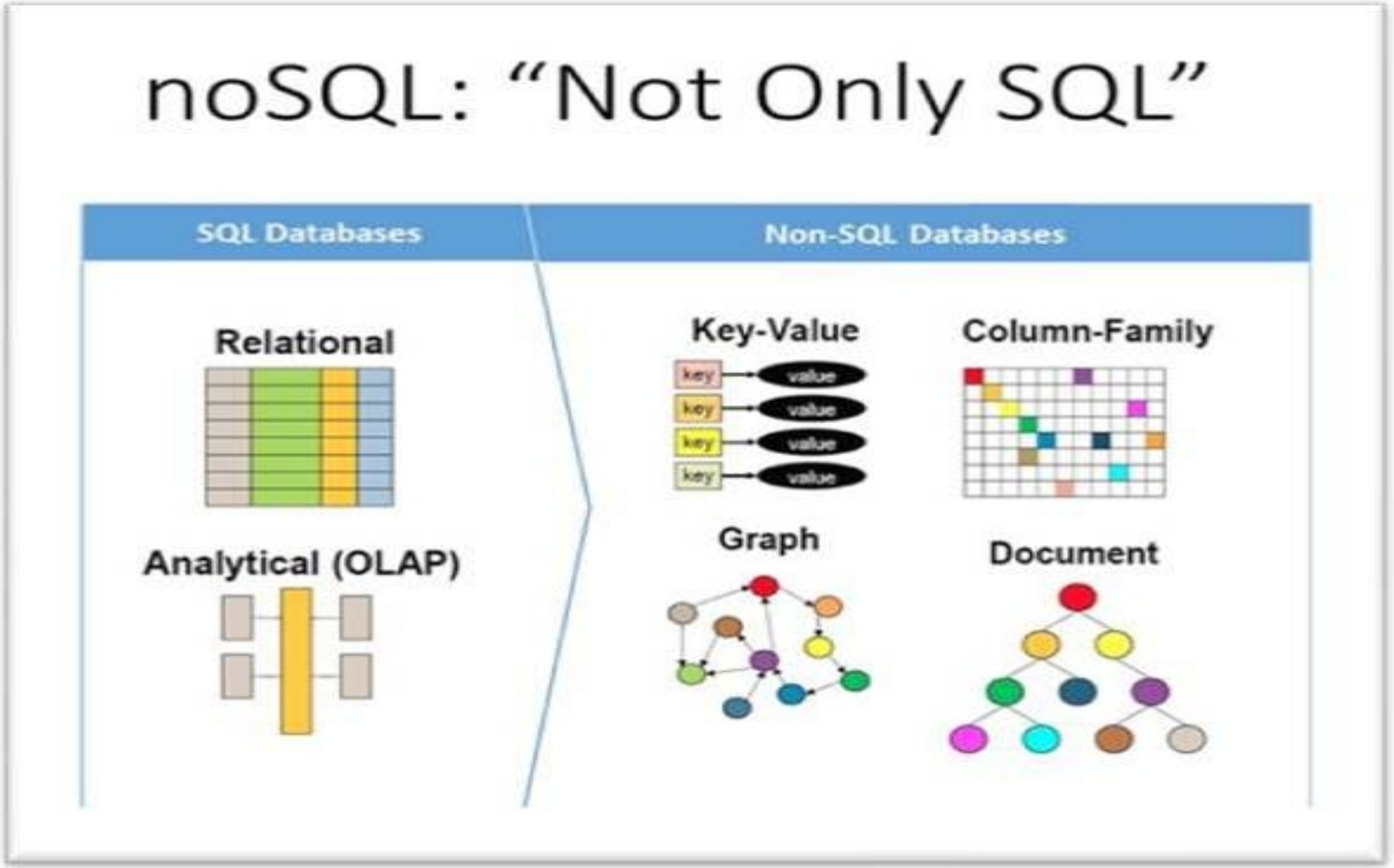
Mise-à-jour:

- différent des SGBDR où on supprime la donnée et on écrit la nouvelle;
- en NoSQL étiquette sur l'ancienne version pour dire que obsolète, ajout de la nouvelle version
- anciennes versions nettoyées périodiquement

Problème de cohérence des données dupliquées

- Horloges vectorielles (un noeud garde un vecteur de versions/timestamp pour lui et les autres noeuds répliques)

NoSQL: une grande famille



Différents systèmes NoSQL

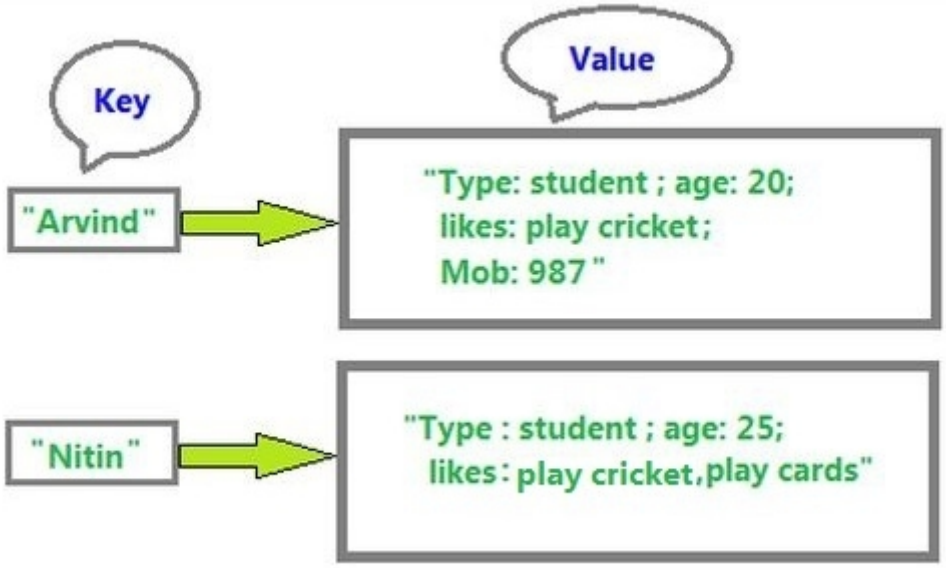
- **Clé-valeur** (Key-Value Store)
  - Données identifiées par clé unique (utilisées pour l'interrogation)
  - *DynamoDB, Voldemort, Redis, Riak, MemcacheDB*
- **Colonnes** (Column data)
  - Relation 1-n "one-to-many" (messages, posts)
  - *HBase, Cassandra, Hypertable*
- **Documents**
  - Données complexes, attributs/valeurs
  - *MongoDB, CouchDB, Terrastore*
- **Graphes**
  - Réseaux sociaux…
  - *Neo4j, OrientDB, FlockDB*

I – NoSQL & Clé-Valeurs

- Similaires à un "*HashMap*" distribué
- Couple Clé+Valeur
  - Pas de schéma pour la valeur (chaine, objet, entier, binaires…) qui peut donc être différente pour chaque
- Conséquences :
  - Pas de structure ni de types
  - Pas d'expressivité d'interrogation (pré/post traitement pour manipuler concrètement les données)

➢ DynamoDB (Amazon), Redis (VMWare), Voldemort (LinkedIn)

Key-value: un exemple



I - NoSQL & Clé-Valeurs (2)

- Opérations CRUD (HTTP)
  - Create(key,value)
  - Read(key)
  - Update(key,value)
  - Delete(key)

- Passage à l'échelle horizontal

  partionnement/distribution des tuples

- Difficile au niveau vertical

  segmentation des données

I - NoSQL & Clé-Valeurs (3)

Avantages:

- Modèle très simple
- Très bonne scalabilité en lecture et en écriture
- Modification facile du contenu associé à une clé et par extension du « schéma » (ajout d'une colonne par ex.)
- Augmente la disponibilité

Inconvénients:

- Interrogation simple car accès par clé => la complexité reportée au pré/post traitement

I - NoSQL & Clé-Valeurs (4)

Exemples d'utilisation:

- Logs de sites Web ou d'application
- Profils utilisateurs de site Web/réseaux sociaux
- Données de capteurs
- Cache Web ou BD
- Paniers sur sites de e-commerce
- …

II – NoSQL & Colonnes

- Stockage des données par colonnes
  - SGBD : tuples (lignes)
- Facile d'ajouter une colonne (pas une ligne!)
  - Schéma peut être dynamique (d'un tuple à l'autre)

➢ BigTable/Hbase (Google), Cassandra (Facebook&Apache), SimpleDB (Amazon)

II – NoSQL & Colonnes (2)

Row Oriented
(RDBMS Model)

Multi-valued

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

null

Column Oriented
(Multi-value sorted map)

id	Name
1	Ricky
2	Ankur
3	Sam

id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

II – NoSQL & Colonnes (3)

- **Avantages** :
  - Supporte le semi-structuré (XML, JSon) donc multi-valué et *sparsité*
  - Indexation de chaque colonne
  - Passage à l'échelle horizontal
  - Compression possible si les données d'une colonne proches
  - Possibilité de regrouper des colonnes en super-colonnes
- **Inconvénients** :
  - Lecture de données complexes difficile
  - Difficulté de relier les données (distance, trajectoires, temps)
  - Requêtes pré-définies (pas à la volée)

II – NoSQL & Colonnes (4)

Exemples d'utilisation:

- Comptage (vote en ligne, compteur, etc)
- Journalisation
- Recherche de produits dans une catégorie (Ebay)
- Reporting large échelle (agrégats calculés sur une colonne)

III – NoSQL & Documents

- Basé sur le modèle clé-valeur
  - Ajout de données semi-structurées (JSon ou XML)
- Requêtes : Interface HTTP
  - Plus complexe que CRUD

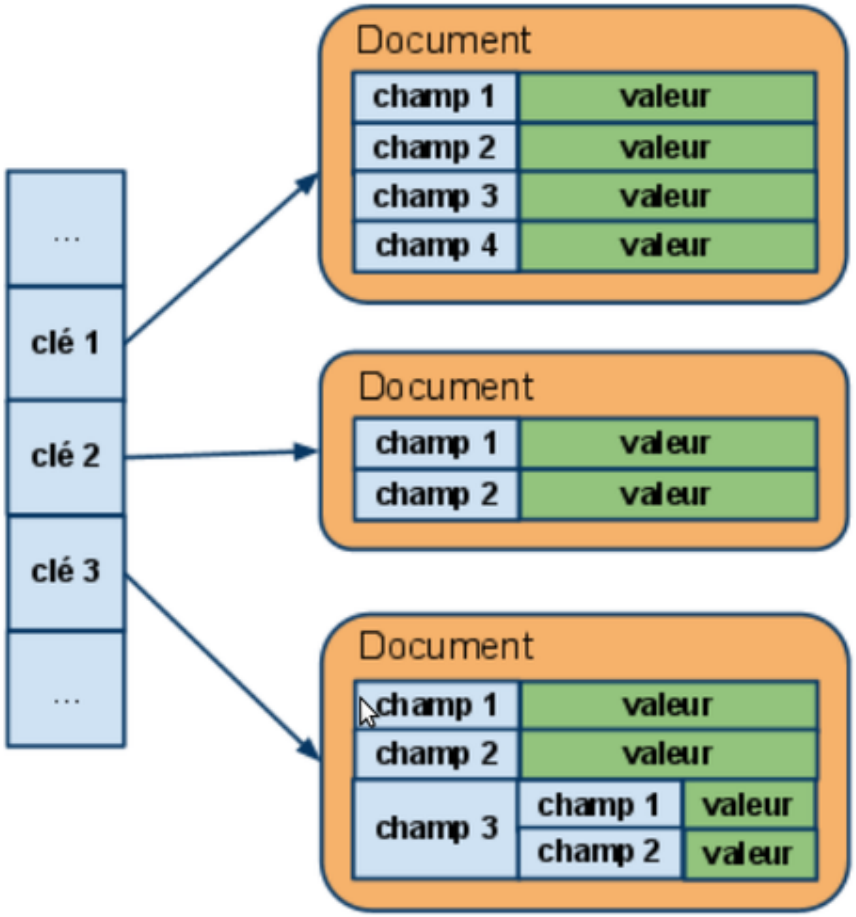
➢ MongoDB, CouchDB (Apache), RavenDB, Terrastore

III – NoSQL & Documents (2)

Comme clé-valeur mais chaque valeur est un document

- Un document est composé de champs et de valeurs associées
- Types simples existent (Int, String, Date)
- Schéma non nécessaire (peut varier d'un document à l'autre)
- Imbrication de données (schéma arborescent): chaque donnée peut être aussi composée de couples clé-valeur
- Chaque donnée du document peut être interrogée

III – NoSQL & Documents (3)



III – NoSQL & Documents (4)

- **Avantages :**
  - Modèle simple mais bonne puissance d'expression (structure imbriquée)
  - Interrogation de tout attribut (+indexation)
  - Passe facilement à l'échelle (surtout si sharding supporté)

- **Inconvénients :**
  - Inter-connexion de données complexe
  - Interrogation sur la clé (+index)

III – NoSQL & Documents (5)

Exemples d'utilisation:

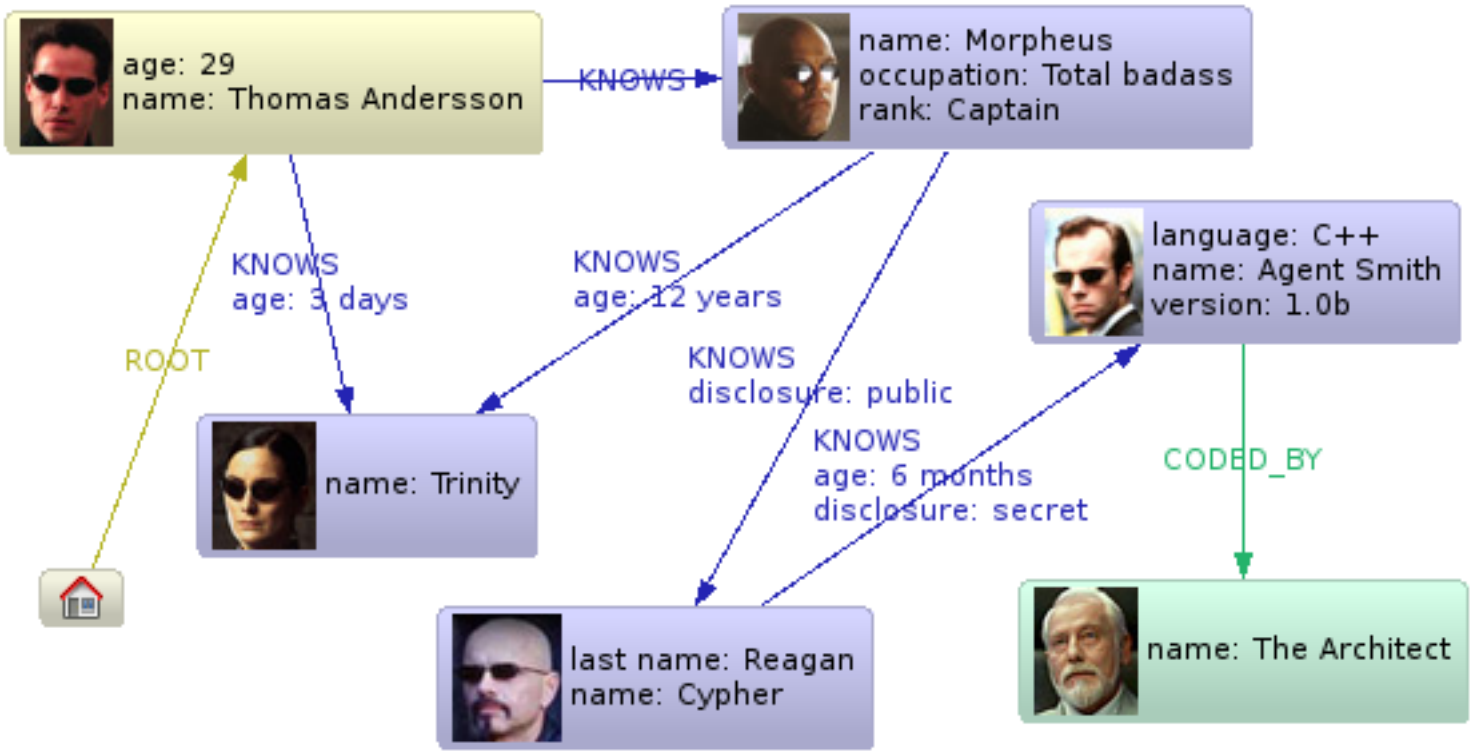
- Gestion de contenu: bibliothèques numériques, collections de produits, dépôts de logiciels « xxxStores », collections multimédia, etc
- Collection d'événements complexes
- Gestion de boîtes email
- Gestion des historiques d'utilisateurs sur réseaux sociaux

IV – NoSQL & Graph

- Stockage des noeuds, relations et propriétés
  - Théorie des graphes
  - Interrogation par traversées de graphe
  - Appel des données sur demande (parcours performants)
  - Modélisation non triviale

➢ Neo4j, OrientDB (Apache), FlockDB (Twitter)

IV – NoSQL & Graph (2)



IV – NoSQL & Graph (3)

- Avantages
  - Modèle adapté pour le stockage de grands graphes
  - Offre des fonctionnalités de calculs sur grands graphes
  - les objets (cf. documents)
  - les arcs (avec propriétés)

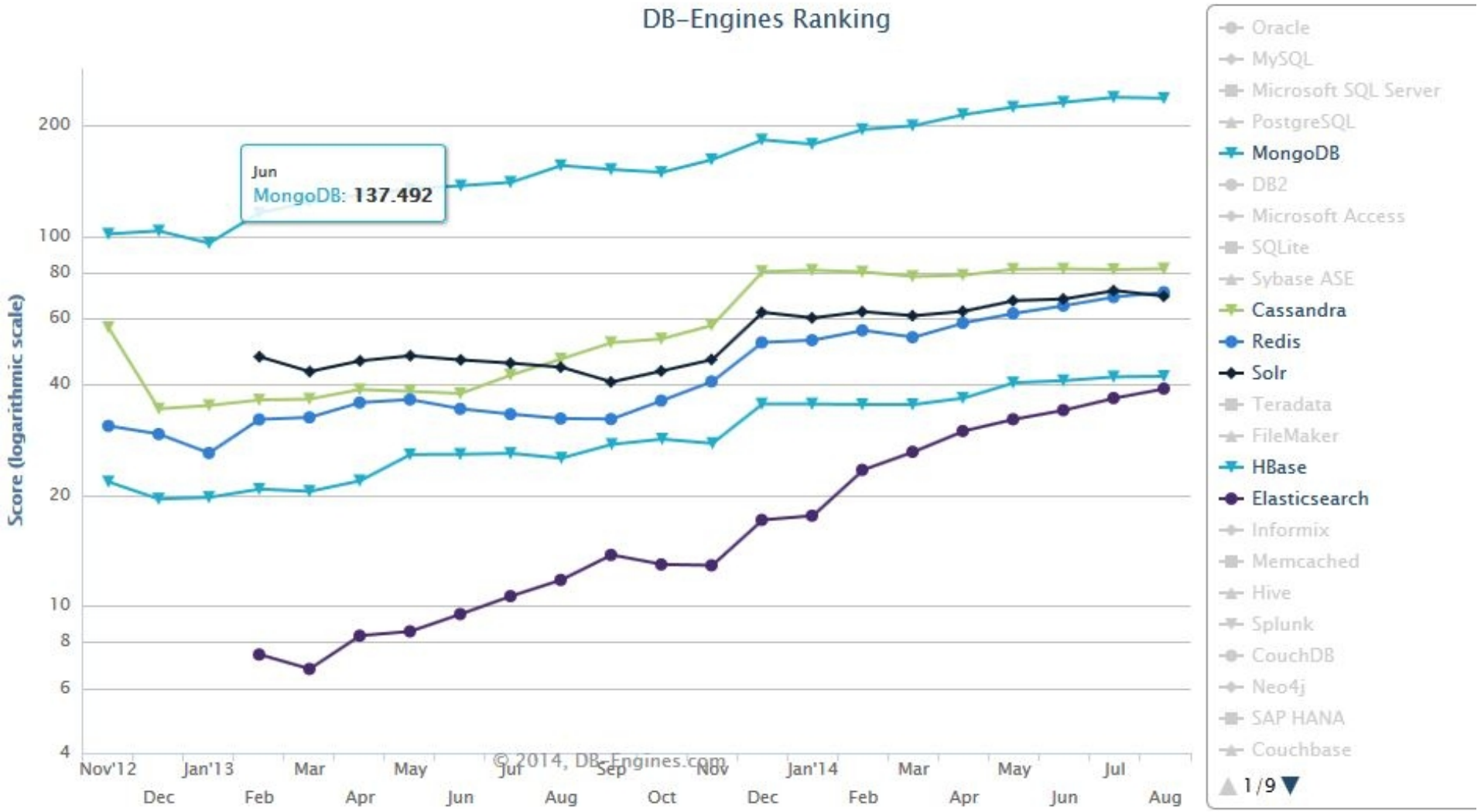
- Inconvénients:

  Sharding/partitionnement difficile

IV – NoSQL & Graph (4)

Exemples d'applications:

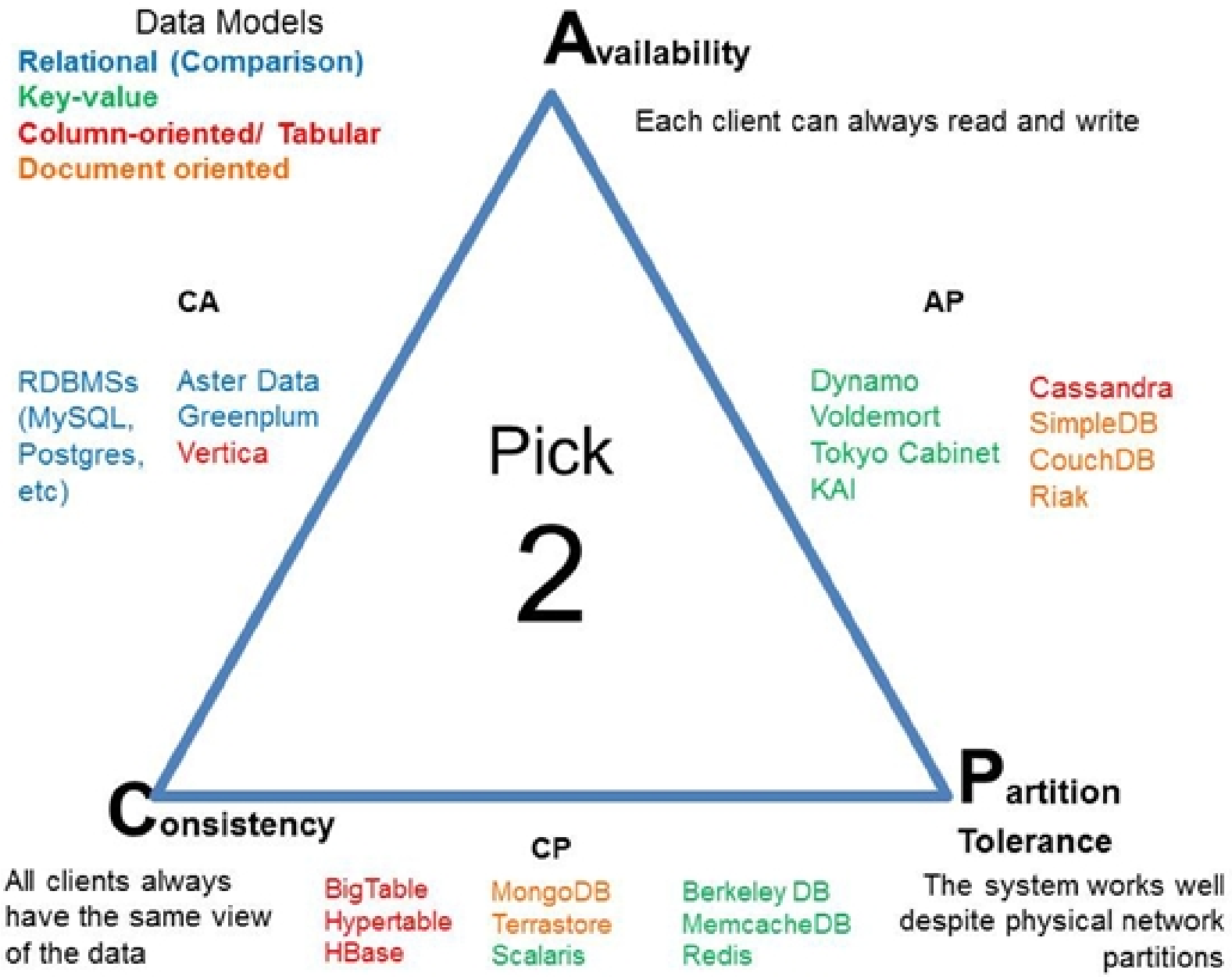
- Calcul sur les graphes sociaux (recommandations, plus courts chemins, atteignabilité,…)
- Calculs sur les réseaux des SIG: réseaux routiers, canalisations, électricité, …
- Web social (linked data)


DB-Engines Ranking
Score (logarithmic scale)
200
100
80
60
40
20
10
8
6
4
Jun
MongoDB: 137.492
Nov'12
Dec
Jan'13
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan'14
Feb
Mar
Apr
May
Jun
Jul
Aug
© 2014, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
DB2
Microsoft Access
SQLite
Sybase ASE
Cassandra
Redis
Solr
Teradata
FileMaker
HBase
Elasticsearch
Informix
Memcached
Hive
Splunk
CouchDB
Neo4j
SAP HANA
Couchbase
1/9

Théorème de Brewer (2000) Dit : "*Théorème de CAP*"

- 3 propriétés fondamentales pour les systèmes distribués
  1. ***Consistency***: Tous les serveurs voient la même données (valeur) en même temps (ou ***Cohérence***)
  2. ***Availability***: Si un serveur tombe en panne, les données restent disponibles
  3. ***Partition Tolerance***: Le système même partitionné doit répondre correctement à toute requête (sauf en cas de panne réseau)

➢ Théorème : Dans un système distribué, il est impossible que ces 3 propriétées co-existent, vous devez choisir 2 d'entre elles.


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick
2
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

DONNÉES SEMI-STRUCTURÉES AVEC JSON

JSon : En pratique

- XML utilisé initialement pour les échanges de communications Machine/Machine
  - Trop verbeux et couteux en place

- JSON (JavaScript Object Notation)
  - Léger, orienté texte, indépendant d'un langage d'interrogation
  - Utilisé par certains Web Services (Google API, Twitter API) ou web dynamique (Ajax)

JSon : Structures

- CLE + VALEUR
  - “nom” : “Travers”
  - Clés avec des guillemets, ne pas utiliser ”.-,”
- Objet : collection de paires “noms/valeurs”
  - Encapsulé dans des accolades
  - { “nom” : “Travers”,
    “prenom” : “Nicolas”,
    “genre” : 1}
- Liste ordonnée de valeurs
  - “liste” : [“SQL”, “XML”, “NoSQL”, “Optimisation BD”, “Recherche d’Information”]

Documents JSon

- Principe : Clé-Valeur
  - “nom” : “Travers”
- Types de données
  - Atomique : String, Integer, flottants, booléens…
  - Liste : tableaux
  - Documents : objets {…}

Liste de valeurs : précisions

- Pas de contraintes sur le contenu de la liste
  - “cours” : [“SQL”, 1, 4.2, null, “Recherche d’Information”]
- Peut contenir des objets
  - “doc” : [{“test” : 1},
    {“test” : {“imbrication” : 1.0}},
    {“test” : “texte”, “valeur” : null}]

JSon : Identifiant

- La clé « _id » est communément utilisée pour identifier un objet
  - Dans une base, l'objet de même identifiant écrase la version précédente
  - Peut être défini automatiquement
    - Exemple MongoDB : "_id" : ObjectId(1234567890)

JSon : Exemple complet

```
{
   “_id” : 1234
   “nom” : “Travers”, “prenom” : “Nicolas”,
   “travail” : {
                “Etablissement” : “Cnam”,
                “localisation” : {
                                “rue” : “2 rue conté”,
                                “ville” : “Paris”,
                                “CP” : 75141
                },
   },
   “themes” : [ “BD” , “Optimisation BD”, “XML”, “NoSQL”, “RI” ]
}
```

NoSQL vs Joins

- Contexte distribué = lien entre les données complexe
- Si l'on considère deux collections de documents JSon
  - Pour une jointure entre les deux collections
    - Sur quel serveur se fait la jointure ?
    - Distribution des données à joindre ?
    - Problème de coût (plus cher qu'en local)

Jointures : Solutions (1/3)

1. Effectuer la **jointure dans l'application**
   - Récupérer les éléments de coll1
   - Pour chaque élément de coll1, interroger coll2
   - Peut être très couteux
   - Equivalent relationnel d'une boucle imbriquée avec index non optimisée
2. Réunir le tout dans **une même collection**
   - Avec MapReduce créer la jointure dans le « *Reduce* » en groupant par valeur de jointure
   - Coût du « *shuffle* » augmenté, ainsi que toute requête sur une des deux collections
   - Equivalent relationnel d'un index de type *Cluster*

Jointures : Solutions (2/3)

3. **Dénormaliser** le schéma

- Réunir les informations des deux collections en un même document lorsque c'est possible
- Créer des sous-documents imbriqués contenant les valeurs jointes
- Problème de cohérence des données (répétitions)
- Equivalent au résultat de la jointure
- Demande un travail de réflexion et de modélisation
  - Préférer la dimension : distributivité

- Dans l'exemple de document JSon précédent, quel a été le travail de dénormalisation ?

Jointures : Solutions (3/3)

4. **Framework** d'exécution de jointure sur les serveurs

- Les données de jointures sont distribués sur les serveurs pour un calcul local
- La base NoSQL doit permettre cette distribution (ex: *Hadoop/Pig*)
- Dépend de la taille de la collection à distribuer
  - Algorithmes de jointure basés sur le relationnel (hachage, tri-fusion)

Conclusion

- NoSQL :
  - Dédié à un contexte extrêmement distribué
  - Calcul fortement distribué
  - 4 types de calculs complexes (clé-valeur, document, colonnes, graphes)
- Ne doit pas remplacé automatiquement un SGBD
  - Propriétés ACID
  - Requêtes complexes
  - Performance de jointure