MIF18 - Les SGBD Non-Relationnels

Fabien Duchateau

fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2014 - 2015

Transparents disponibles sur
`http://liris.cnrs.fr/~ecoquery/dokuwiki/doku.php?id=enseignement:bdav:start`

Introduction

Depuis les années 1970, dominance du modèle Relationnel

Avec l'émergence du Web et d'Internet, se pose le problème du passage à l'échelle (millions d'utilisateurices interagissant avec un système donné)

Réflexions pour passer d'un système large échelle vertical ("dopage" du serveur) à un système large échelle horizontal (ajout de machines)

Explosion du volume de données à stocker et à traiter (phénomène "Big Data")

Le Big Data

« Big Data » : modélisation, stockage et traitement (analyse) d'un ensemble de données très volumineuses, croissantes et hétérogènes, dont l'exploitation permet entre autre :

- Prise de décisions
- Découverte de nouvelles connaissances
- Possibilités de nouveaux « business models » (e.g., accès à un service contre des informations)

Causes :

- Faible coût du stockage
- Faible coût des processeurs
- Mise à disposition des données

Le Big Data (2)

Les « 3V », caractéristiques du Big Data :

- **Volume** (plusieurs zettaoctets générés par an sur le Web)
- **Vélocité** (fréquence de génération des données)
  - notion de flux (stream)
  - 7 Go par jour pour Twitter (2012)
  - 7000 To par seconde prévus pour le radiotélescope "Square Kilometre Array"
- **Variété** (hétérogénéité)
  - données brutes, structurées ou pas, etc.
  - images, texte, géo-démographiques, profils utilisateurices, etc.

---

http://fr.wikipedia.org/wiki/Big_data
http://fr.wikipedia.org/wiki/Zettaoctet

Quelques applications du Big Data

Sciences :

- Création de cartes de navigation par Fontaine Maury, au 19^{eme} siècle, à partir de vieux journaux de bord (précurseur)
- Large Hadron Collider (LHC), un accélérateur de particules qui génère un flux de 40 millions de données par seconde
- Linked Open Data (e.g., DBpedia, Freebase)

Politique :

- Prédiction du résultat des élections États-uniennes 2012
- Transparence (Open Data)

---

http://en.wikipedia.org/wiki/Big_data#Big_science
http://linkeddata.org
http://fr.wikipedia.org/wiki/Open_data

Quelques applications du Big Data (2)

Industrie, secteur public, santé, etc. :

- Amazon gère des millions d'opérations par jour
- Programmes de surveillance (e.g., Prism)
- Décodage du génome humain
- Prédiction des analystes de Google quelques semaines avant l'épidémie de grippe aviaire en 2009
- Découverte d'un effet secondaire dû à la prise de deux médicaments par analyse des requêtes des internautes (Yahoo)
- Confirmation de la censure par analyse de la fréquence de noms de personnes (Chagall en Allemagne, Trotsky en URSS)

http://labsoftnews.typepad.com/lab_soft_news/2013/03/

Contexte

Contexte du Big Data :

- Demande émergente pour des SGBD distribués
  - à forte disponibilité
  - résistants au morcellement

Contexte des applications Web :

- Schémas dynamiques (nombre d'attributs extensible)
- Nombre important de lectures/écritures
- Relations complexes

Contexte

Contexte du Big Data :

- Demande émergente pour des SGBD distribués
  - à forte disponibilité
  - résistants au morcellement

Contexte des applications Web :

- Schémas dynamiques (nombre d'attributs extensible)
- Nombre important de lectures/écritures
- Relations complexes

Les SGBD Relationnels sont moins adaptés pour le Big Data et le Web $\Rightarrow$ mouvement NoSQL, NotOnlySQL, NoRel, NewSQL, ...

Plan du cours

Théorème CAP

Le théorème de Brewer ou **théorème CAP** = un système distribué ne peut garantir en même temps les trois propriétés suivantes :

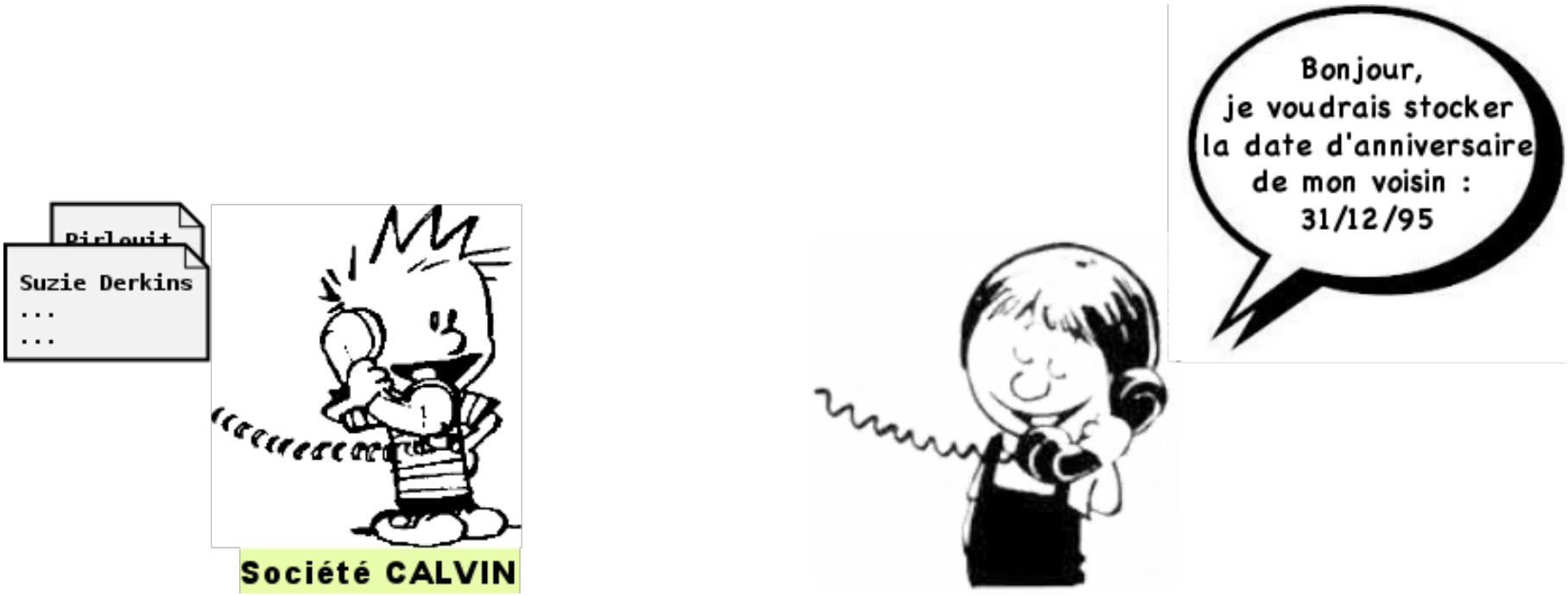
- Cohérence (consistency) : tous les noeuds du système voient la même information au même moment
- Disponibilité (*availability*) : toute requête reçoit une réponse
- Résistance au morcellement (*partition tolerance*) : fonctionnement autonome en cas de morcellement du réseau (sauf panne totale)

`http://ksat.me/a-plain-english-introduction-to-cap-theorem/`
`http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`

Illustration du théorème CAP

Calvin décide de créer une entreprise pour mémoriser et restituer des informations fournies par les clients.

Principe de mémorisation :



Calvin and Hobbes, http://fr.wikipedia.org/wiki/Calvin_et_Hobbes

Illustration du théorème CAP

Calvin décide de créer une entreprise pour mémoriser et restituer des informations fournies par les clients.

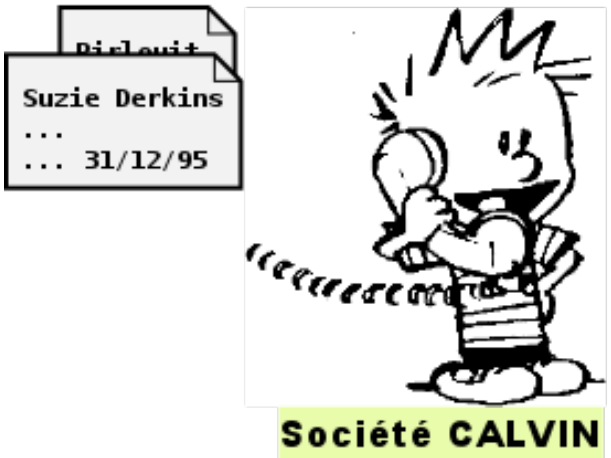
Principe de mémorisation :

Calvin and Hobbes, http://fr.wikipedia.org/wiki/Calvin_et_Hobbes

Illustration du théorème CAP

Calvin décide de créer une entreprise pour mémoriser et restituer des informations fournies par les clients.

Principe de restitution :



Calvin and Hobbes, http://fr.wikipedia.org/wiki/Calvin_et_Hobbes

Illustration du théorème CAP

Calvin décide de créer une entreprise pour mémoriser et restituer des informations fournies par les clients.

Principe de restitution :

Calvin and Hobbes, http://fr.wikipedia.org/wiki/Calvin_et_Hobbes

Illustration du théorème CAP

Calvin décide de créer une entreprise pour mémoriser et restituer des informations fournies par les clients.

Principe de restitution :

Calvin and Hobbes, http://fr.wikipedia.org/wiki/Calvin_et_Hobbes

Illustration du théorème CAP (2)

Face au succès, Calvin est rapidement débordé d'appels.

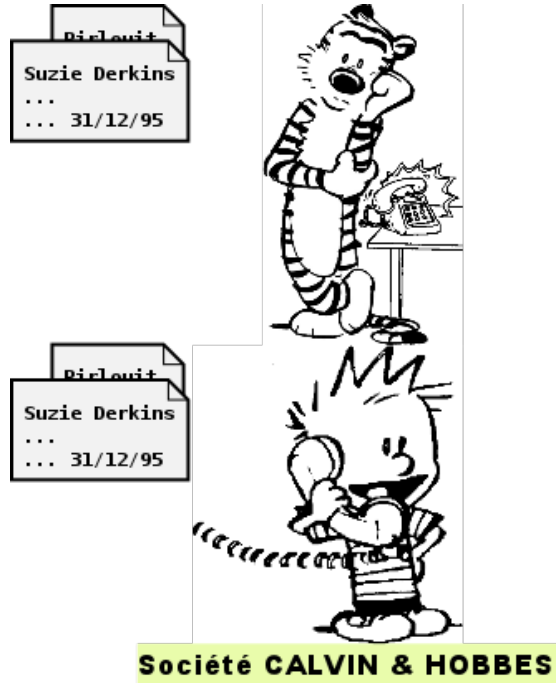
- ✓ **Cohérence**
- ✓ **Résistance au morcellement**
- ✗ **Disponibilité**

Calvin and Hobbes, Les Bidochons, Gaston Lagaffe, Tintin, Le Chat

Illustration du théorème CAP (2)

Face au succès, Calvin est rapidement débordé d'appels.

- ✓ **Cohérence**
- ✓ **Résistance au morcellement**
- ✗ **Disponibilité**



Solution à la "non disponibilité" :
faire équipe avec Hobbes

Calvin and Hobbes, Les Bidochons, Gaston Lagaffe, Tintin, Le Chat

Illustration du théorème CAP (3)

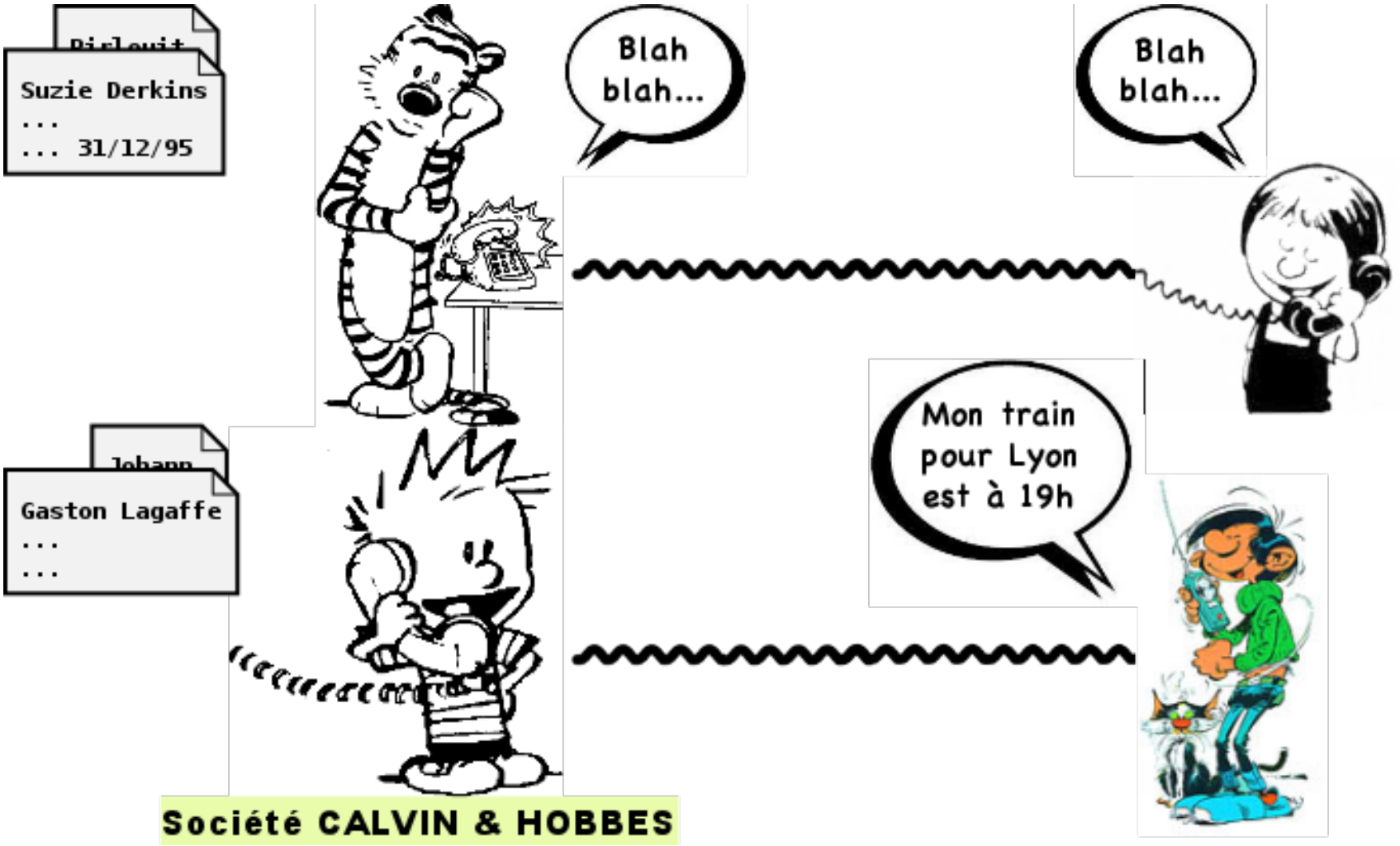


Illustration du théorème CAP (3)

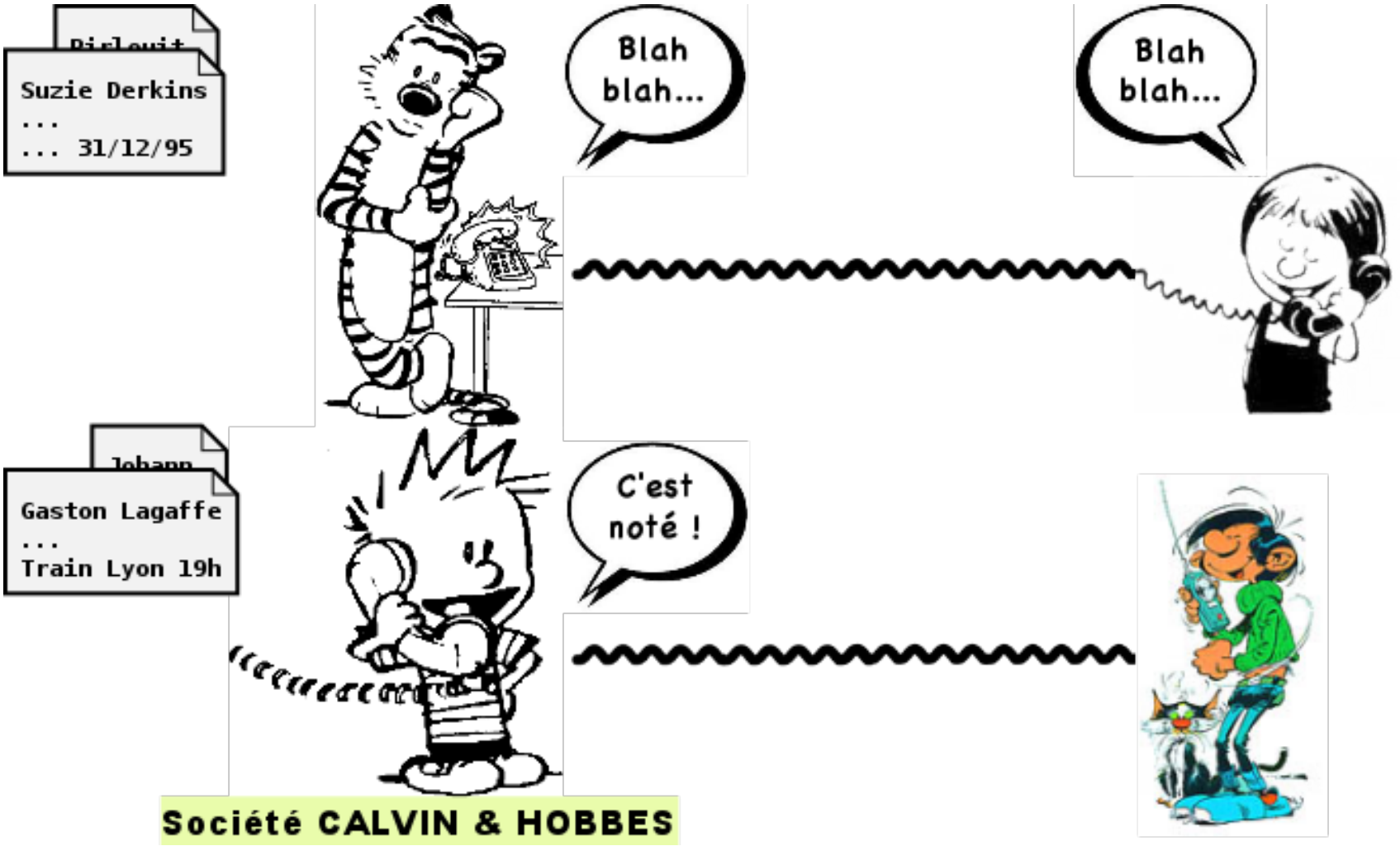


Illustration du théorème CAP (3)

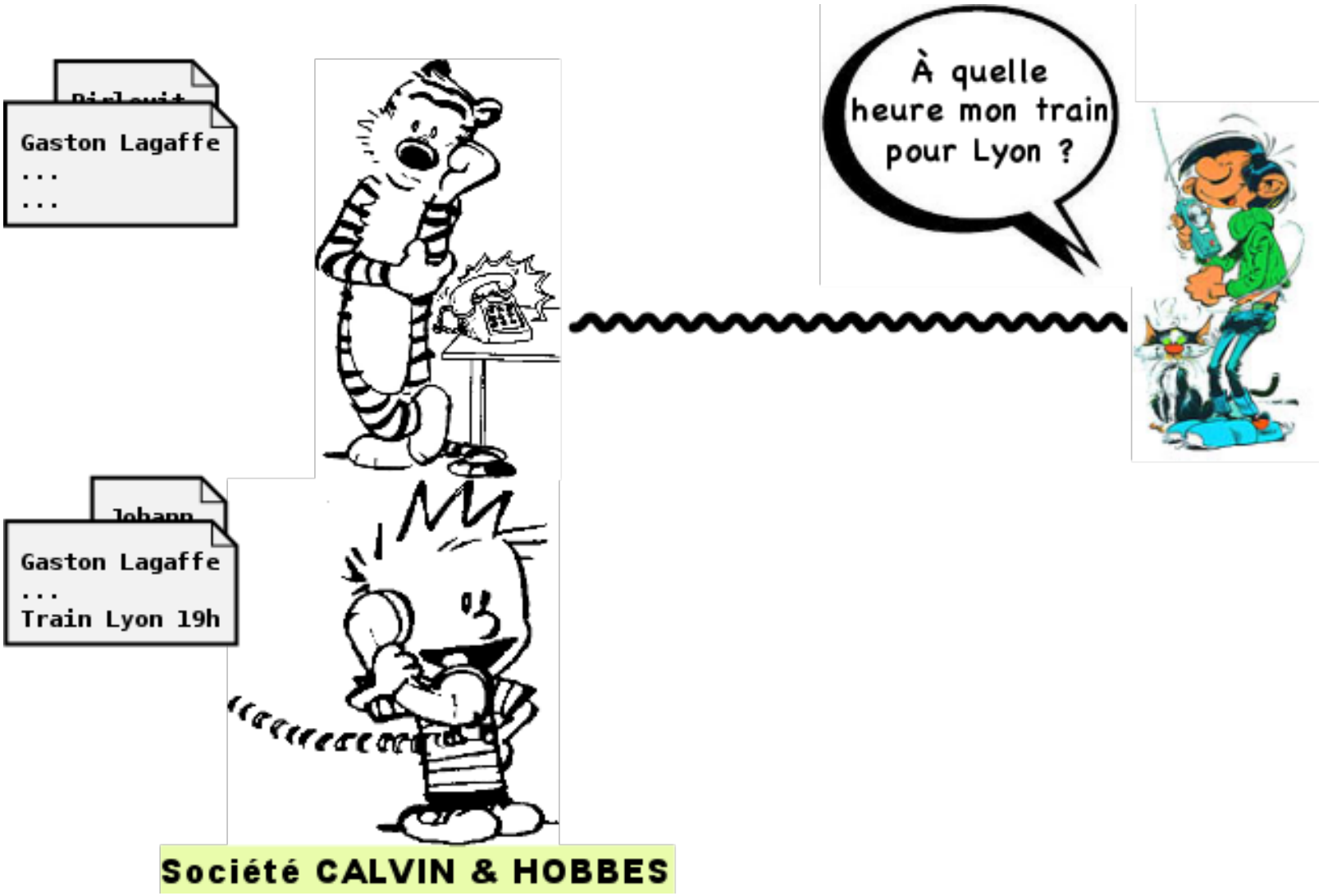
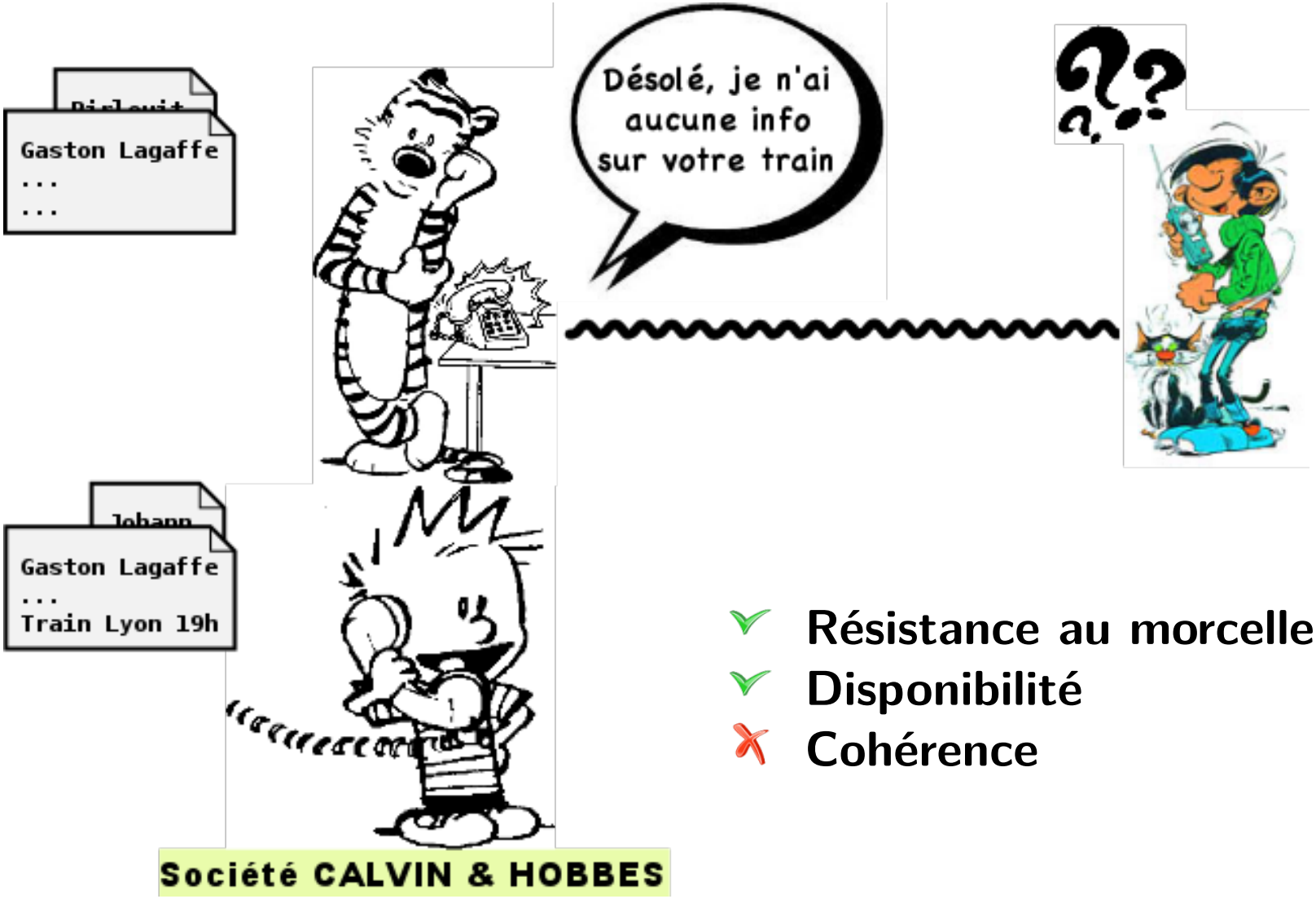


Illustration du théorème CAP (3)



- ✓ **Résistance au morcellement**
- ✓ **Disponibilité**
- ✗ **Cohérence**

Illustration du théorème CAP (3)

Solution possible pour l'incohérence : mise à jour des fiches entre Calvin et Hobbes

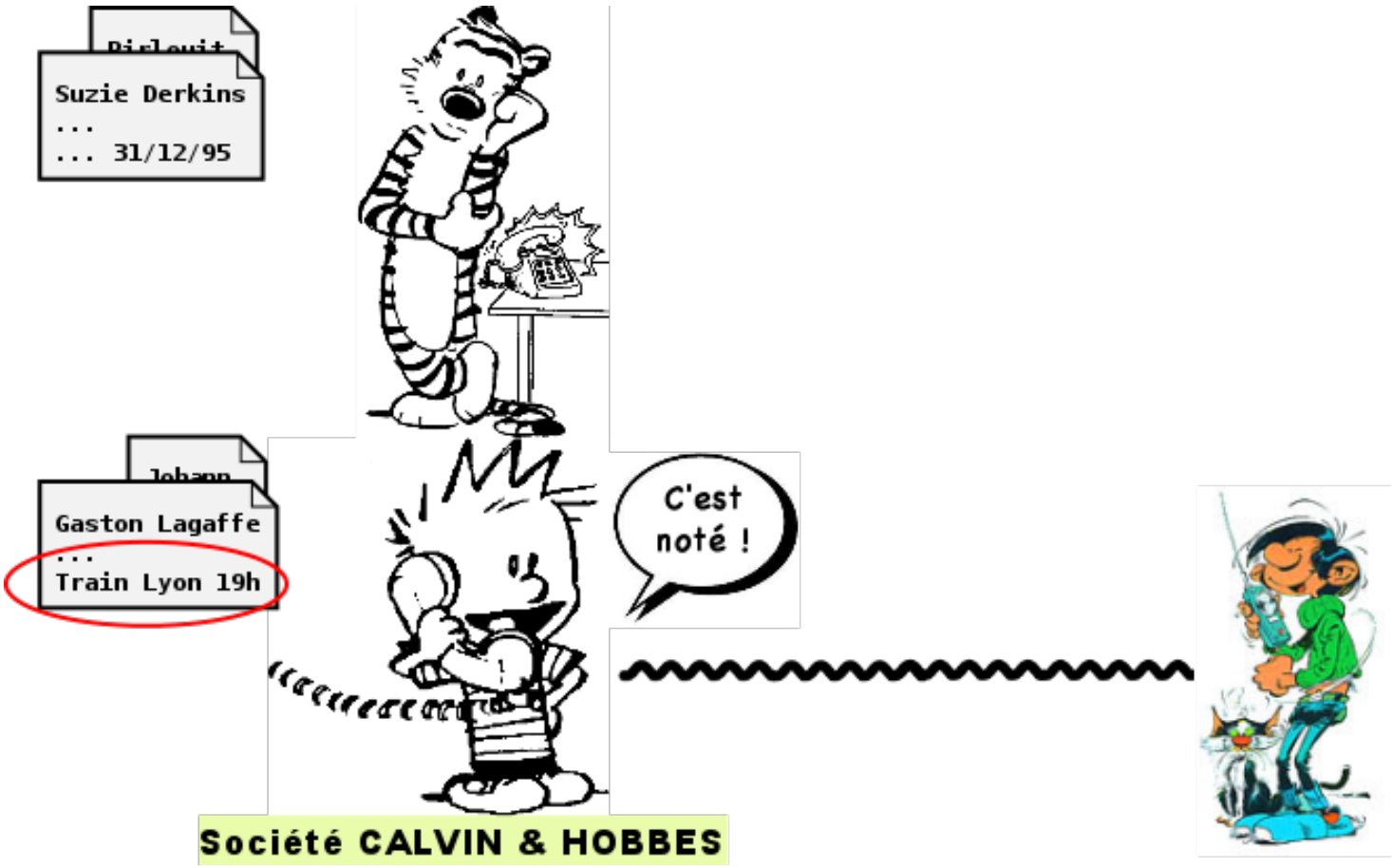


Illustration du théorème CAP (3)

Solution possible pour l'incohérence : mise à jour des fiches entre Calvin et Hobbes

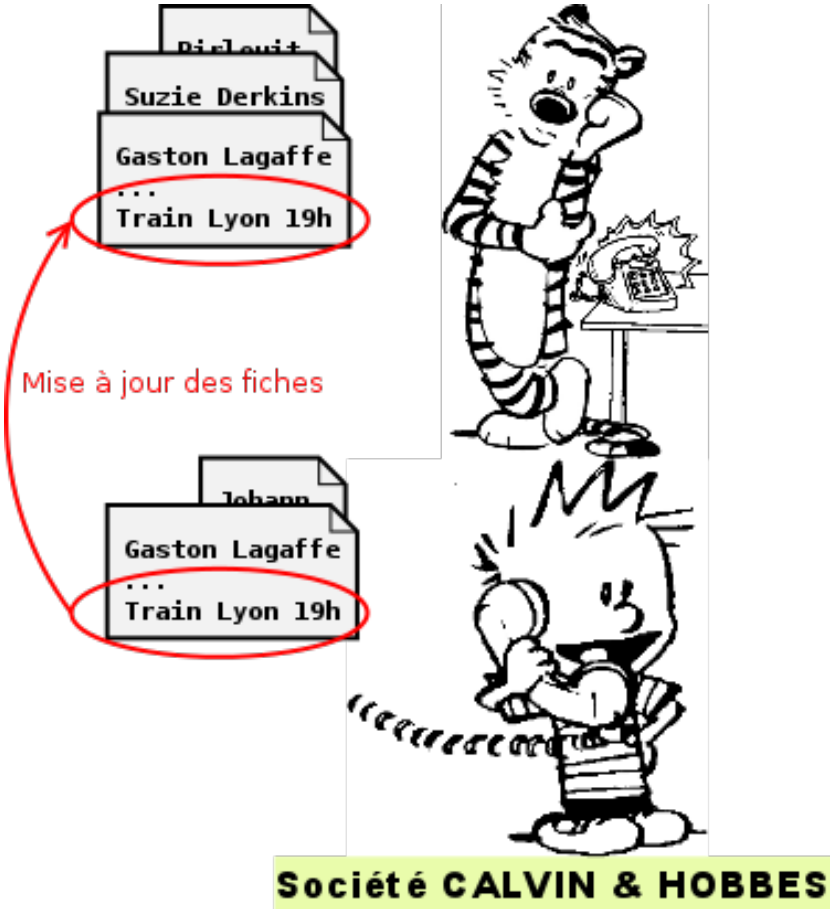
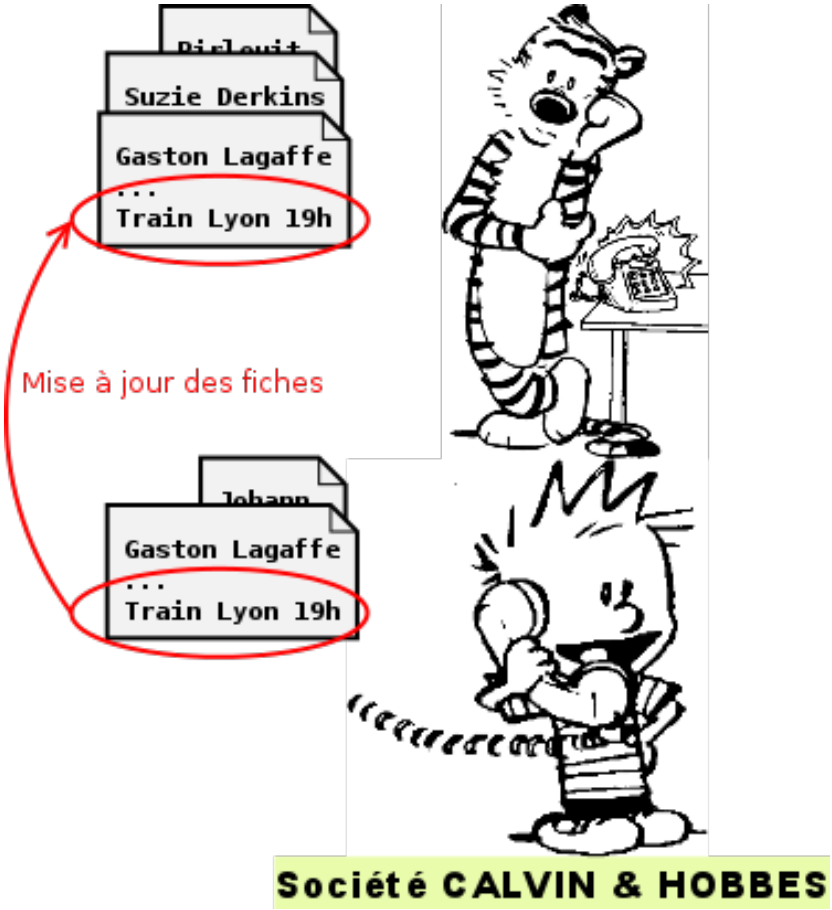


Illustration du théorème CAP (3)

Solution possible pour l'incohérence : mise à jour des fiches entre Calvin et Hobbes



De nouveau un problème de disponibilité !

Illustration du théorème CAP (4)

Et lorsque Calvin et Hobbes ne communiquent plus ?

Illustration du théorème CAP (4)

Et lorsque Calvin et Hobbes ne communiquent plus ?

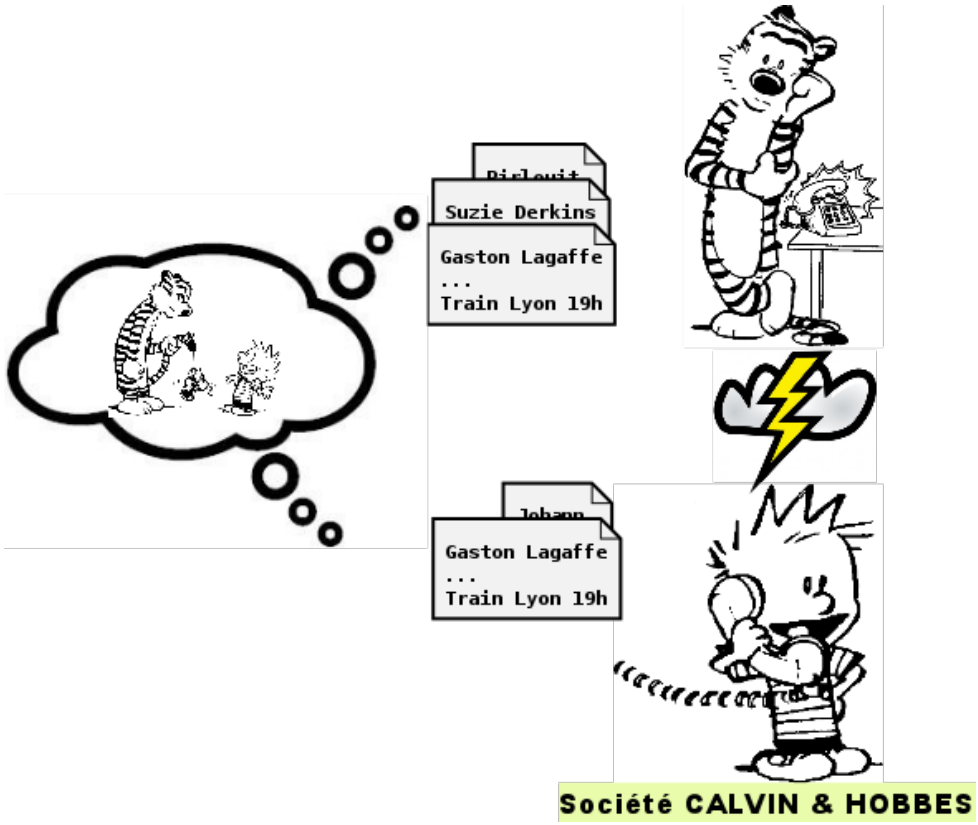


Illustration du théorème CAP (4)

Et lorsque Calvin et Hobbes ne communiquent plus ?

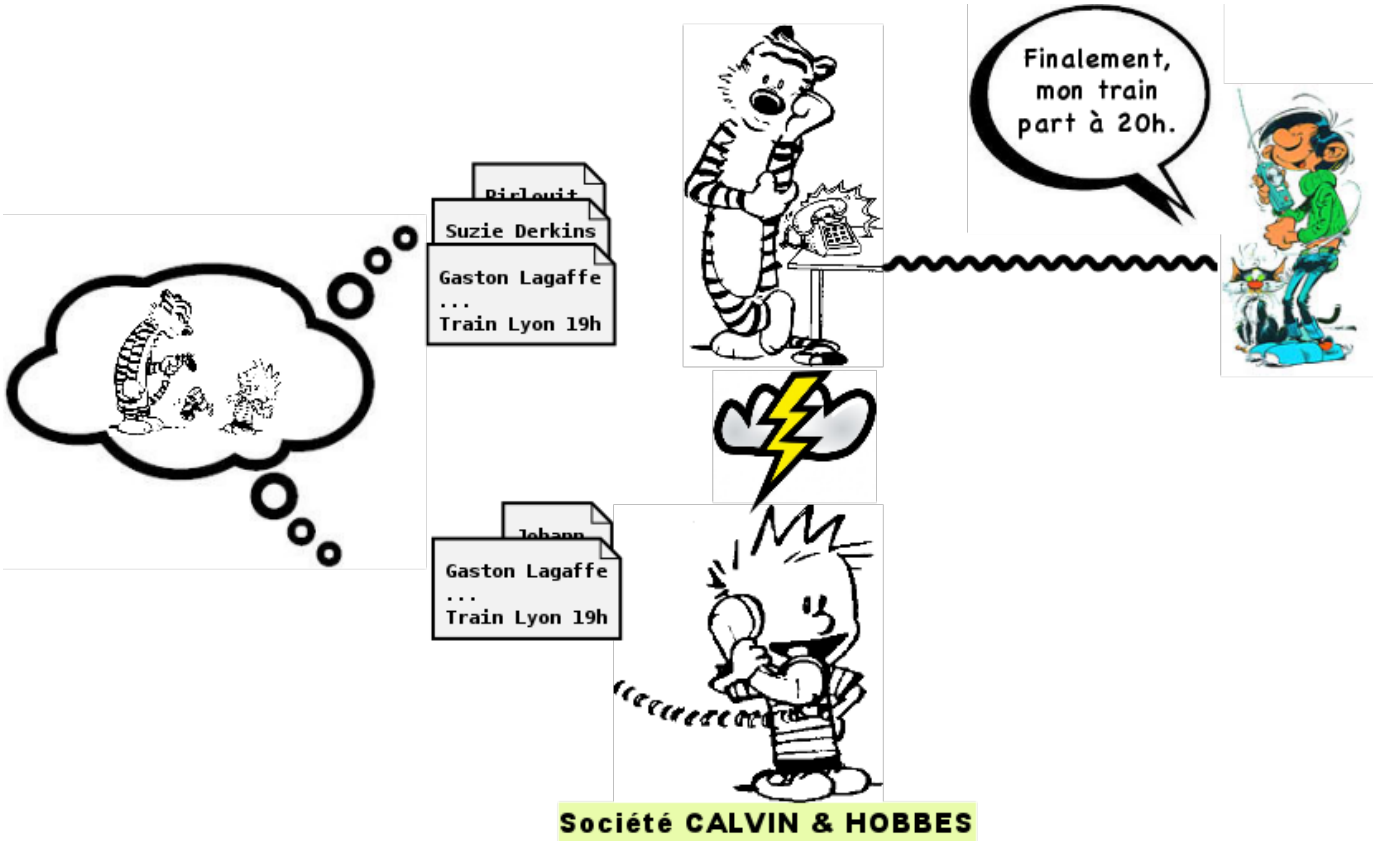
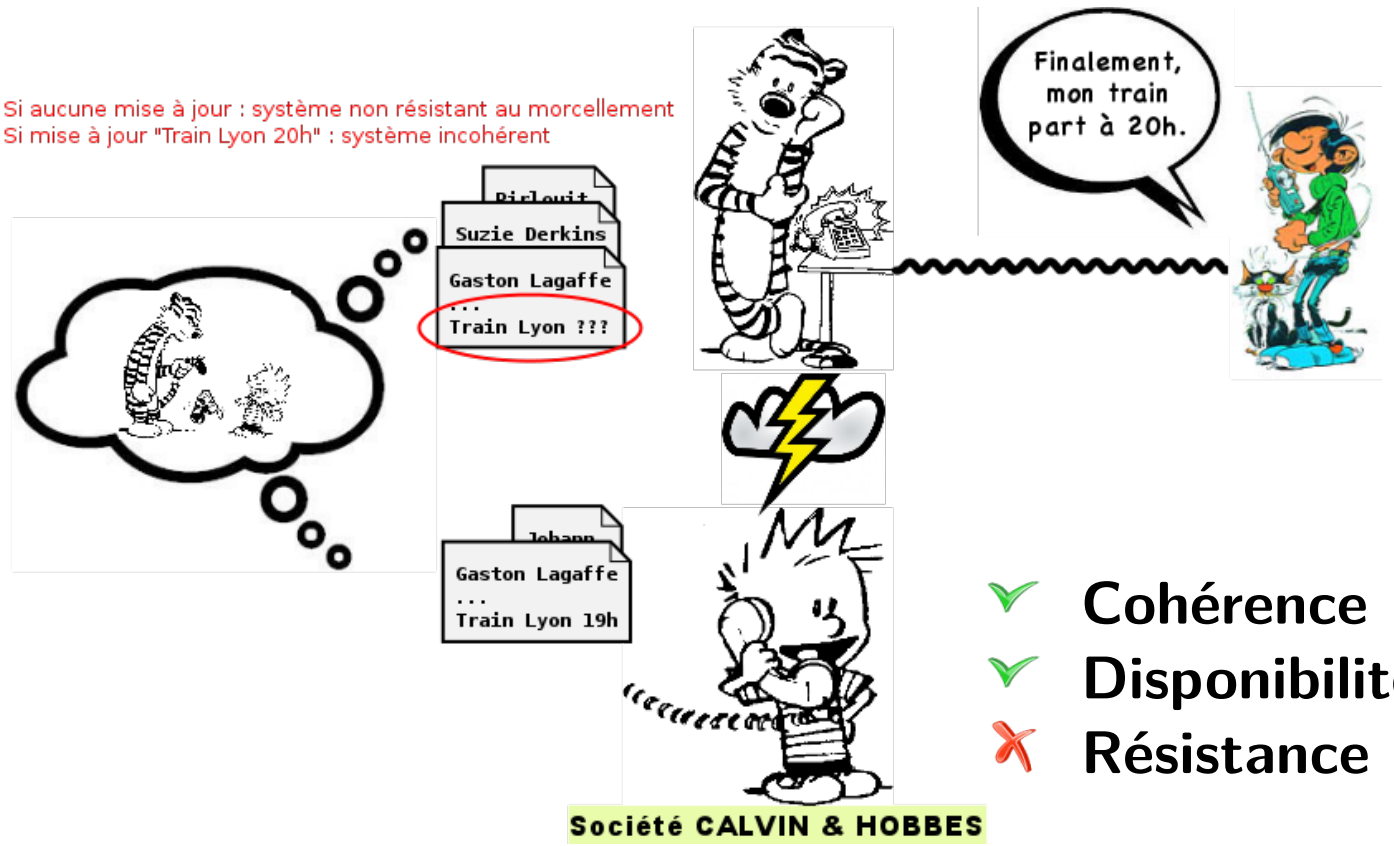


Illustration du théorème CAP (4)

Et lorsque Calvin et Hobbes ne communiquent plus ?



- ✓ **Cohérence**
- ✓ **Disponibilité**
- ✗ **Résistance au morcellement**

Illustration du théorème CAP (4)

Solution possible : "réconcilier" le système et éventuelle mise à jour des fiches

Illustration du théorème CAP (4)

Solution possible : "réconcilier" le système et éventuelle mise à jour des fiches

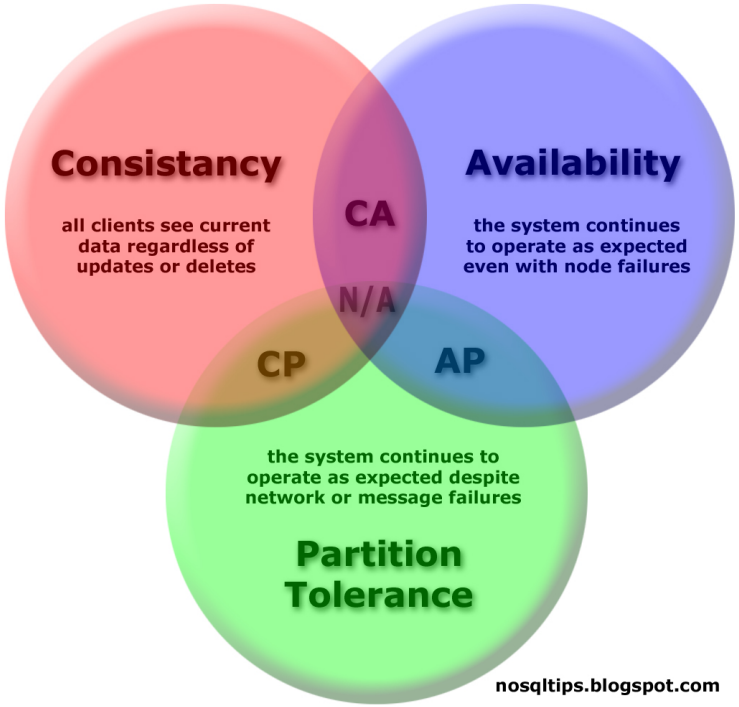
Société CALVIN & HOBBES

De nouveau un problème de disponibilité !

Caractéristiques des SGBD NoRel

Caractéristiques :

- Garantie de deux propriétés parmi cohérence, disponibilité et résistance au morcellement
- Performance (scalabilité horizontale)
- Pas de schéma de table fixé
- Éviter les jointures (données dénormalisées)
- Pas de transactions
- Pas de fonctionnalité de requêtage complexe (SQL)



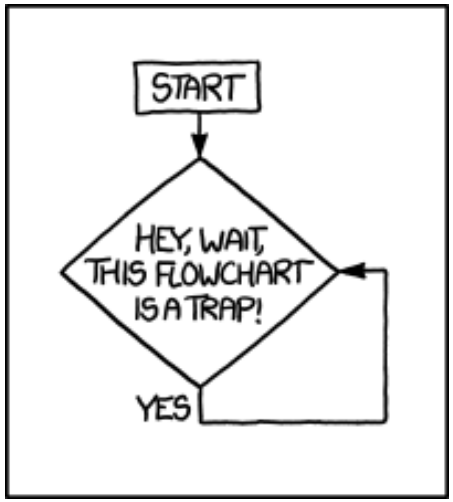
`http://en.wikipedia.org/wiki/NewSQL`

Caractéristiques des SGBD NoRel (2)

SGBD NoRel $\approx$ entrepôt clé-valeur fortement optimisé

Les SGBD NoRel sont dites BASE, pour :

- Basically Available (haute disponibilité)
- Soft-state (changement de l'état du système, même sans mise à jour)
- "Eventually consistent" (cohérence sur le long terme)



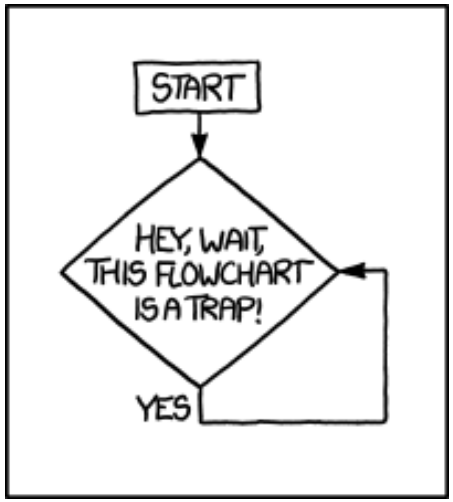
http://en.wikipedia.org/wiki/Eventual_consistency

Caractéristiques des SGBD NoRel (2)

SGBD NoRel $\approx$ entrepôt clé-valeur fortement optimisé

Les SGBD NoRel sont dites BASE, pour :

- Basically Available (haute disponibilité)
- Soft-state (changement de l'état du système, même sans mise à jour)
- "Eventually consistent" (cohérence sur le long terme)



Comment assurer la cohérence dans un SGBD distribué ?

`http://en.wikipedia.org/wiki/Eventual_consistency`

Cohérence

Dans la philosophie NoRel, on accède aux données par des clés

Cohérence d'une clé (distribuée) : toutes les lectures pour cette clé retournent la même donnée

La cohérence est un problème crucial lors d'écritures simultanées de données sur des serveurs distribués (voire sur des centres de données distribués)

- Solution en Relationnel avec les transactions distribuées
- Trop lente et dépendance à la propriété de disponibilité

⇒ Propositions de technique pour la cohérence à long terme

`http://en.wikipedia.org/wiki/Eventual_consistency`

Cohérence (2)

Différentes formes de cohérences :

- "Strong consistency" : tous les serveurs sont cohérents après une mise à jour
- "Weak consistency" :
  - "read your writes" : des lectures successives d'un même processus retournent toujours la dernière donnée mise à jour
  - "session consistency" : idem que "read your writes", mais pour une session donnée
  - "monotonic reads" : des lectures successives retournent toujours la dernière donnée lue ou une plus récente
  - "monotonic writes" : une écriture se termine avant toute autre écriture successive

---

Large Scale and Big Data : Processing and Management, 2014, `http://books.google.fr/books?id=X-rMAwAAQBA`

Cohérence (3)

- "Causal consistency" : l'ordre séquentiel des opérations n'est préservé que pour les opérations sur une même clé
- "Eventual consistency" (cohérence à long terme) : si aucune nouvelle mise à jour n'est effectuée pour une clé donnée, alors toutes les prochaines lectures finiront par lire la même valeur associée à cette clé (à plus ou moins long terme)
- "Timeline consistency" : tous les serveurs exécutent les opérations sur une même clé dans le même ordre

Cohérence (3)

- "Causal consistency" : l'ordre séquentiel des opérations n'est préservé que pour les opérations sur une même clé
- "Eventual consistency" (cohérence à long terme) : si aucune nouvelle mise à jour n'est effectuée pour une clé donnée, alors toutes les prochaines lectures finiront par lire la même valeur associée à cette clé (à plus ou moins long terme)
- "Timeline consistency" : tous les serveurs exécutent les opérations sur une même clé dans le même ordre

Un SGBD peut proposer plusieurs formes de cohérence, et l'application détermine la plus adaptée

Cohérence sur le long terme

Trois techniques pour atteindre la cohérence à long terme :

- **Two-Phase Commit** : protocole qui coordonne les processus impliqués dans une transaction atomique distribuée à commiter ou annuler (lent et non tolérant aux pannes)
- **Paxos-style consensus** : protocole qui trouve une valeur au consensus parmi les processus participants

---

http://en.wikipedia.org/wiki/Two-phase_commit_protocol
http://en.wikipedia.org/wiki/Paxos_algorithm

Cohérence sur le long terme (2)

- **Read-repair** : écriture de toutes les versions incohérentes, et lors d'une prochaine lecture, le conflit sera détecté et résolu (souvent en écrivant sur un certain nombre de répliques)

  Exemple avec Dynamo (Amazon) : dans un panier, les ajouts de produits sont cruciaux, mais les suppressions de produits le sont moins car le client corrigera l'erreur. Dans ce cas, l'union du contenu des deux paniers en conflit permet de ne pas perdre d'ajout de produits. Utilisation des horloges vectorielles pour limiter l'incohérence des paniers au niveau des suppressions de produits

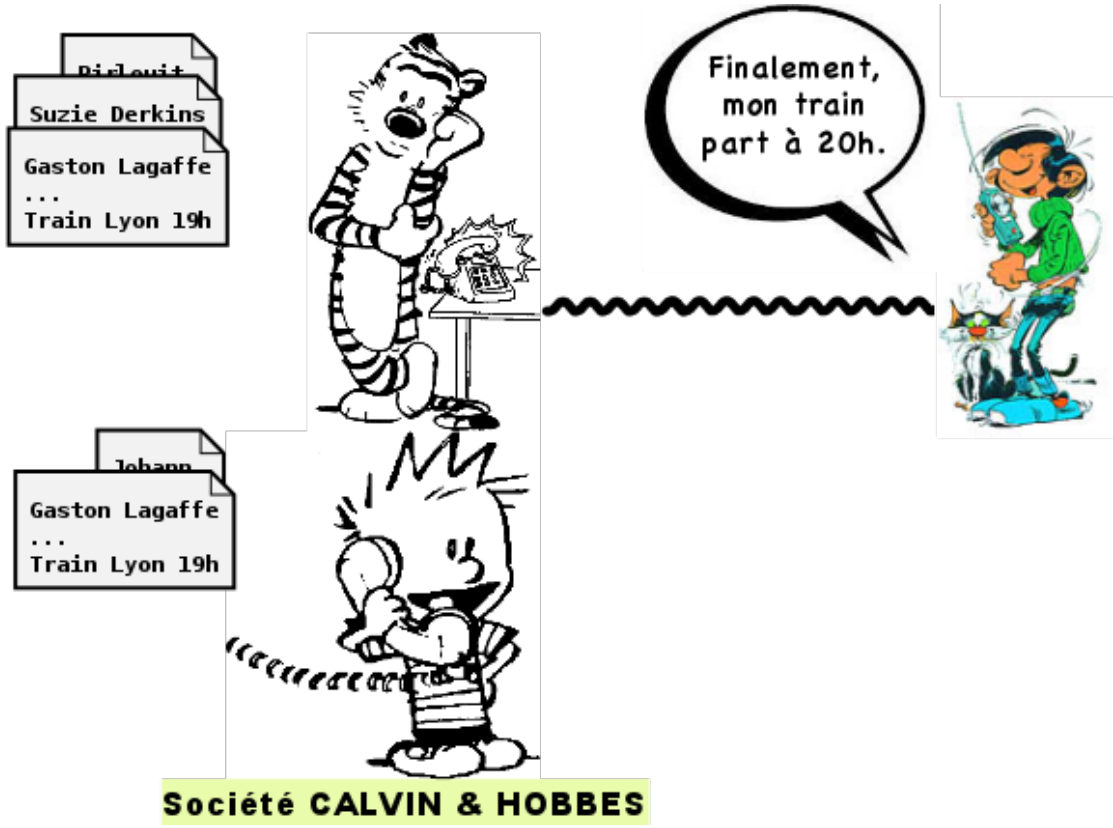
---

Philosophie *"When in doubt, take customer's order !"*
`http://the-paper-trail.org/blog/2008/08/26/`
`http://en.wikipedia.org/wiki/Vector_clock`

Cohérence sur le long terme (3)

En pratique, la cohérence sur le long terme est réalisée par un processus de fond (e.g., gestionnaire/clerc)

Gestionnaires/clercs

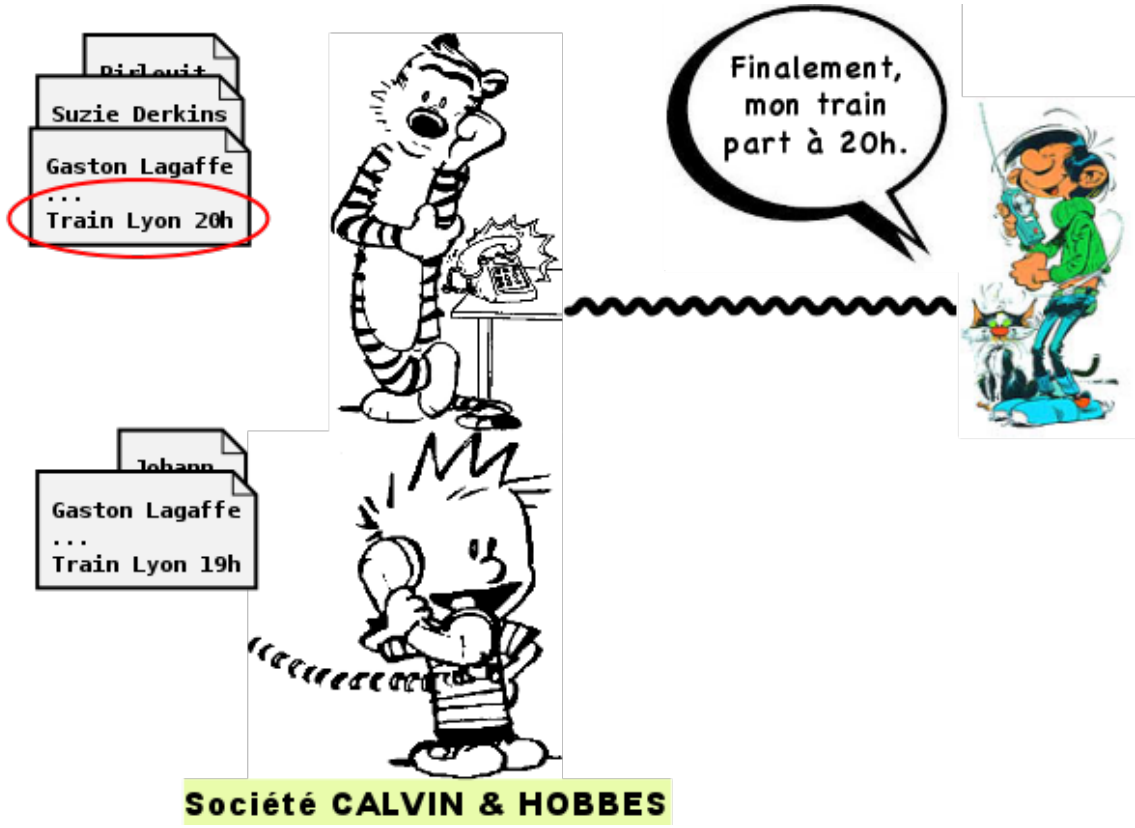


Société CALVIN & HOBBES

Cohérence sur le long terme (3)

En pratique, la cohérence sur le long terme est réalisée par un processus de fond (e.g., gestionnaire/clerc)

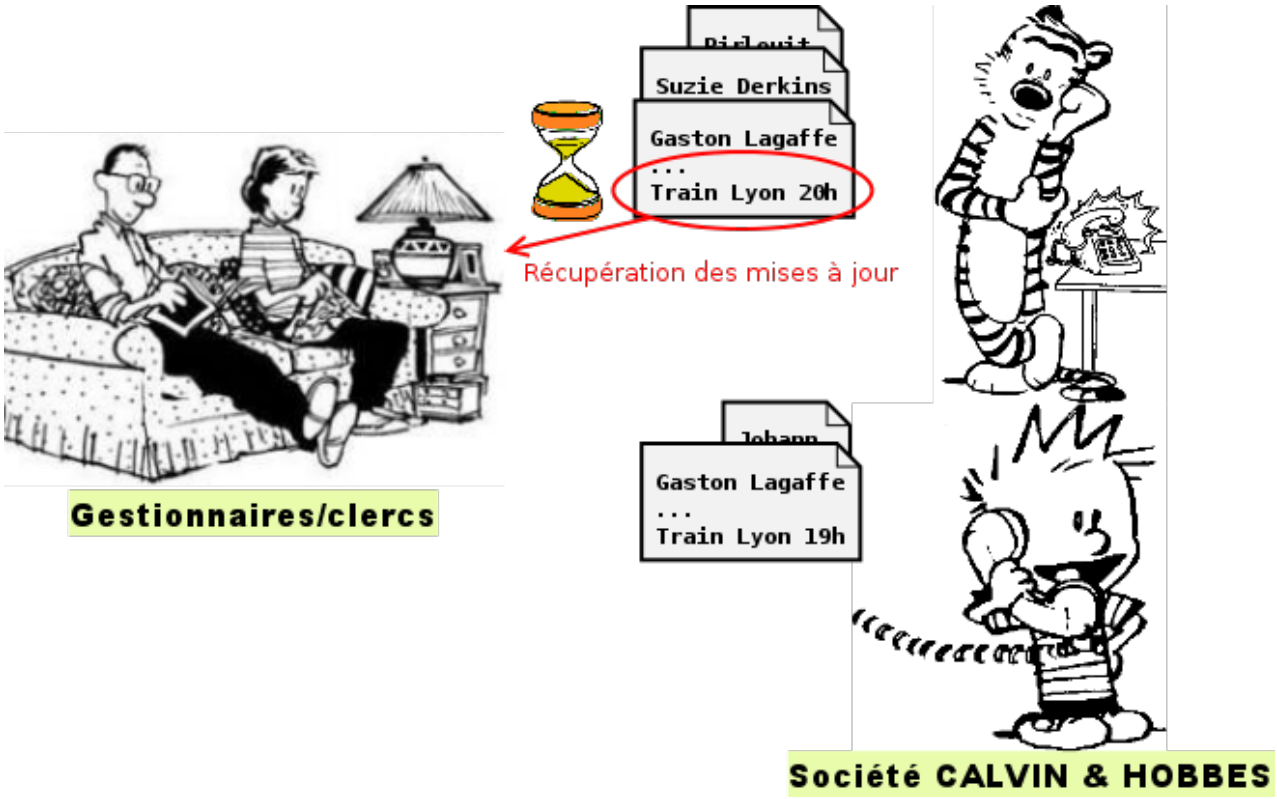
Gestionnaires/clercs



Société CALVIN & HOBBES

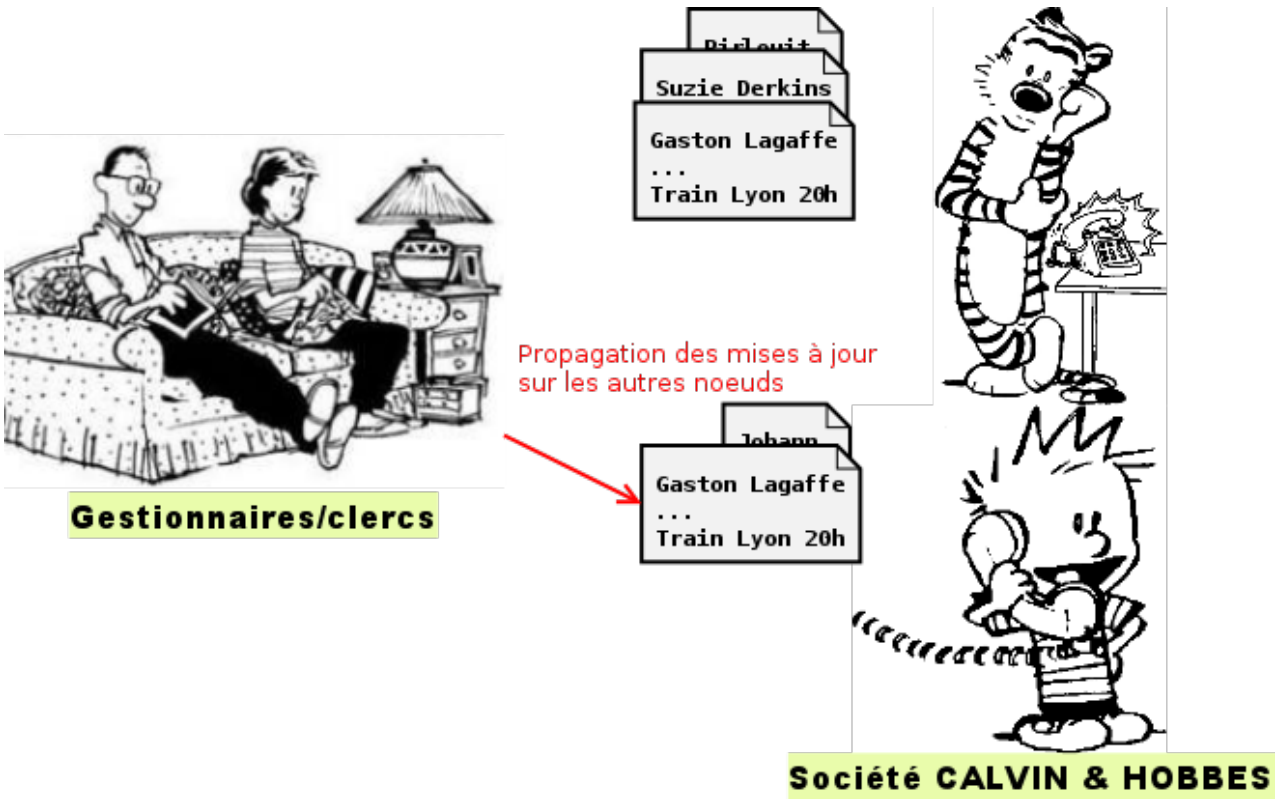
Cohérence sur le long terme (3)

En pratique, la cohérence sur le long terme est réalisée par un processus de fond (e.g., gestionnaire/clerc)



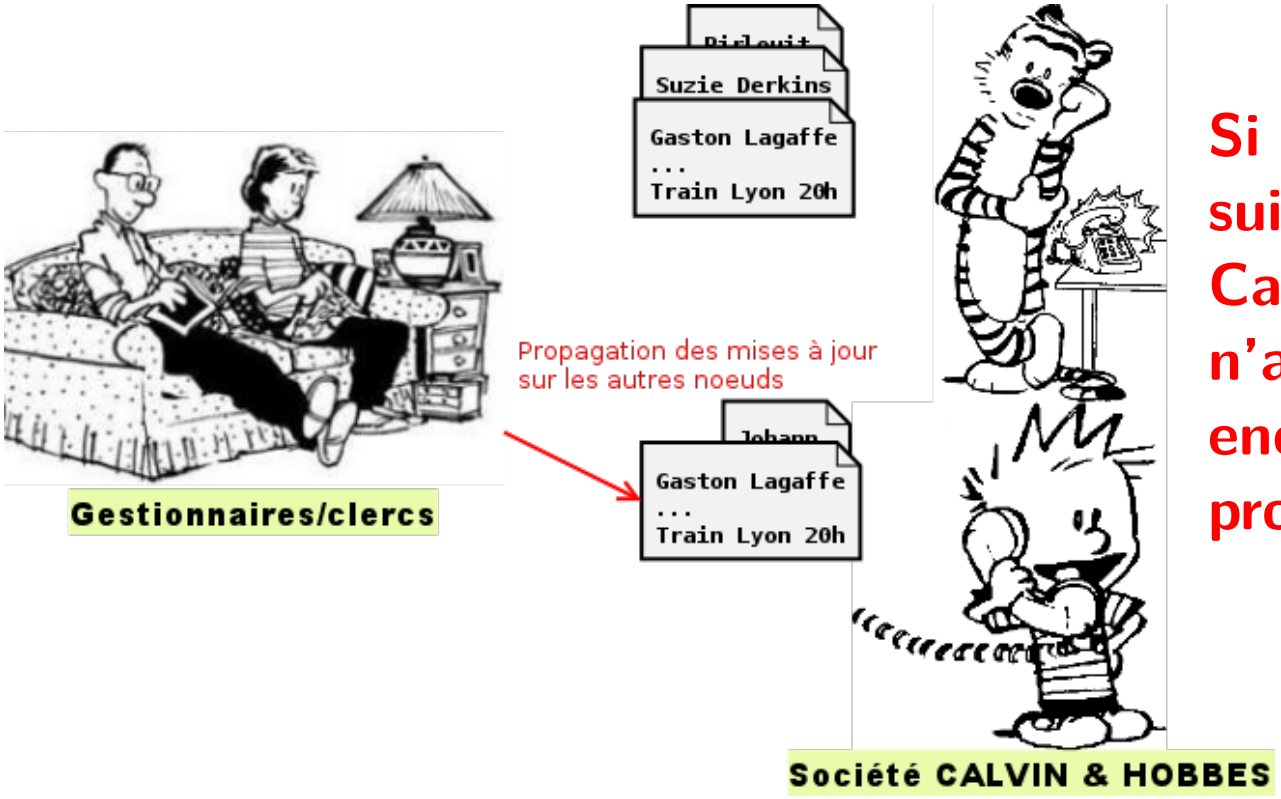
Cohérence sur le long terme (3)

En pratique, la cohérence sur le long terme est réalisée par un processus de fond (e.g., gestionnaire/clerc)



Cohérence sur le long terme (3)

En pratique, la cohérence sur le long terme est réalisée par un processus de fond (e.g., gestionnaire/clerc)



Si Gaston rappelle de suite et tombe sur Calvin, la mise à jour n'a peut-être pas encore été faite par le processus clerc

En résumé

- Théorème CAP : deux critères sur trois satisfiables
  - en général, cohérence atteinte sur le long-terme
- Schéma dynamique et non contraint
- Passage à l'échelle horizontal

"YOUR CONTENT IS CONSISTENT. IT'S ALWAYS BAD."

Plan du cours

Catégories des SGBD NoRel

Domaine en pleine évolution !

Proposition de classements selon :

- Le modèle de données
- Les caractéristiques non fonctionnelles
- Les propriétés du théorème de CAP

Dans la suite, classement selon le modèle de données

Steven Yen, *NoSQL is a horseless carriage*, 2009, `http://dl.getdropbox.com/u/2075876/nosql-steve-yen.pdf`

`http://fr.slideshare.net/bscofield/nosql-codemash-2010`

Le modèle Relationnel

Un modèle que vous connaissez bien...

- Relations, attributs, tuples, etc.
- Propriétés de cohérence et de disponibilité
- SGBD : PostgreSQL, Oracle, MySQL, etc.

Exemple de tables Relationnelles

Table ECRIVAIN

id	nom	pays	dateNaiss
2	GRR Martin	USA	20/09/48

Table LIVRE

id	titre	prix	datePubli	ecrivain
1	Le trône de fer	15	01/08/96	2

Entrepôt clé-valeur

Entrepôt clé-valeur = ce modèle est aussi appelé « key-value store » ou tableau associatif

- La clé est un identifiant unique
- La valeur peut être structurée ou pas

Implémentation minimale :

1. valeur = **get**(clé)
2. **insert**(clé, valeur)
3. **delete**(clé)

Les implémentations proposent souvent des fonctionnalités ou propriétés supplémentaires

Entrepôt clé-valeur (2)

SGBD type entrepôt clé-valeur :

- Serveur standalone (Redis)
- Distribué (Dynamo, Riak, Voldemort)
- Uniquement en mémoire (Memcached)
- ...

Performances (++), passage à l'échelle (++), flexibilité (++), prise en main (++), nombre de fonctionnalités (∼/−)

---

http://redis.io/
http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
http://basho.com/riak/
http://www.project-voldemort.com/
http://www.memcached.org/

Entrepôt clé-valeur (3)

Exemple d'un entrepôt clé-valeur

```
"nom-ecrivain2" : "GRR Martin"
"pays-ecrivain2" : "USA"
"dateNaiss-ecrivain2" : 20/09/48
"titre-livre1-ecrivain2" : "Le trône de fer"
"prix-livre1-ecrivain2" : 15
"datePubli-livre1-ecrivain2" : "01/08/96"
```

BD orientée colonnes

BD orientée colonnes = organisation des données en colonnes

- Une colonne stocke le nom de la colonne et la valeur associée (et un timestamp)
- Une supercolonne stocke des colonnes ($\approx$ ligne en Relationnel)
- Une famille de colonnes stocke des colonnes ou supercolonnes

Avantages :

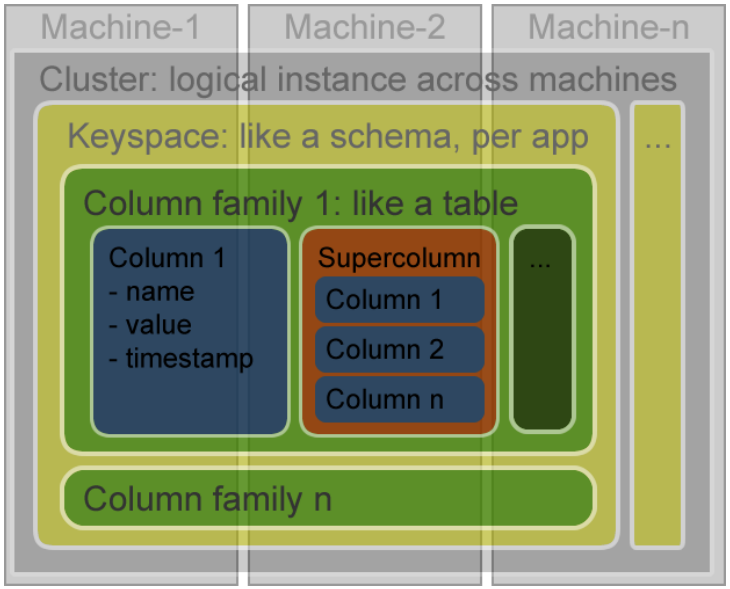
- Schéma dynamique (ajout de colonne)
- Pas de stockage de valeurs nulles
- Possibilité d'avoir des "super colonnes" pour stocker des listes de listes (e.g., Cassandra)

Ne pas confondre avec des BD Relationnelles orientées colonnes, qui sérialisent les données par colonne (e.g., MonetDB, Vertica)

BD orientée colonnes (2)

SGBD orienté colonnes :

- BigTable (propriété de Google)
- Cassandra (implémentation libre de BigTable)
- HBase, Hypertable (basé sur BigTable et Hadoop DFS)
- ...



Performances (++), passage à l'échelle (++), flexibilité (+), prise en main (+), nombre de fonctionnalités (−)

---

http://cassandra.apache.org/
http://cassandra-php.blogspot.fr/
http://hbase.apache.org/
http://hypertable.com/
http://en.wikipedia.org/wiki/Hadoop

BD orientée colonnes (3)

Exemple d'une BD orientée colonnes

La première ligne représente la famille de colonne *écrivain*
La seconde ligne représente la famille de colonne *livre*

1	nom : GRR Martin	pays : USA	dateNaiss : 20/09/48

2	titre : Le trône de fer	prix : 15	datePubli : 01/08/96	auteur : 1

BD orientée graphes

BD orientée graphes = éléments interconnectés avec un nombre indéterminé de relations entre eux

- Noeuds pour représenter des entités
- Arc étiqueté et directionnel entre deux noeuds
- Propriété sur un noeud ou un arc

Avantages :

- Émergence de "patterns" par analyse des noeuds
- Facilité d'évolution du schéma
- Bonnes performances pour des requêtes type graphe (e.g., plus court chemin)
- Permet de représenter les 3 autres modèles

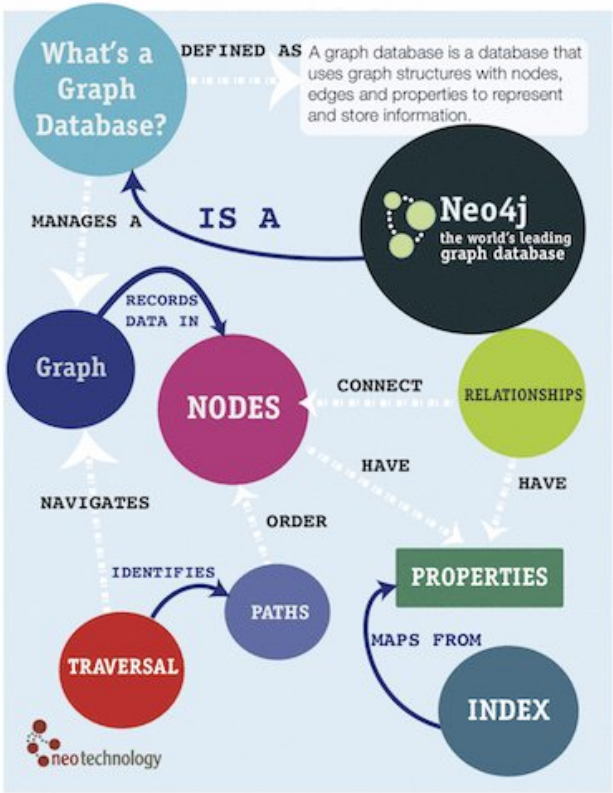
`http://en.wikipedia.org/wiki/Graph_database`

BD orientée graphes (2)

SGBD orienté graphes :

- Neo4J (ACID, avec transactions)
- Allegro Graph (triplestore, raisonnement Prolog)
- Virtuoso (triplestore et SGBDR)
- ...

Performances (∼), passage à l'échelle (∼), flexibilité (++), prise en main (-), fonctionnalités (théorie des graphes)

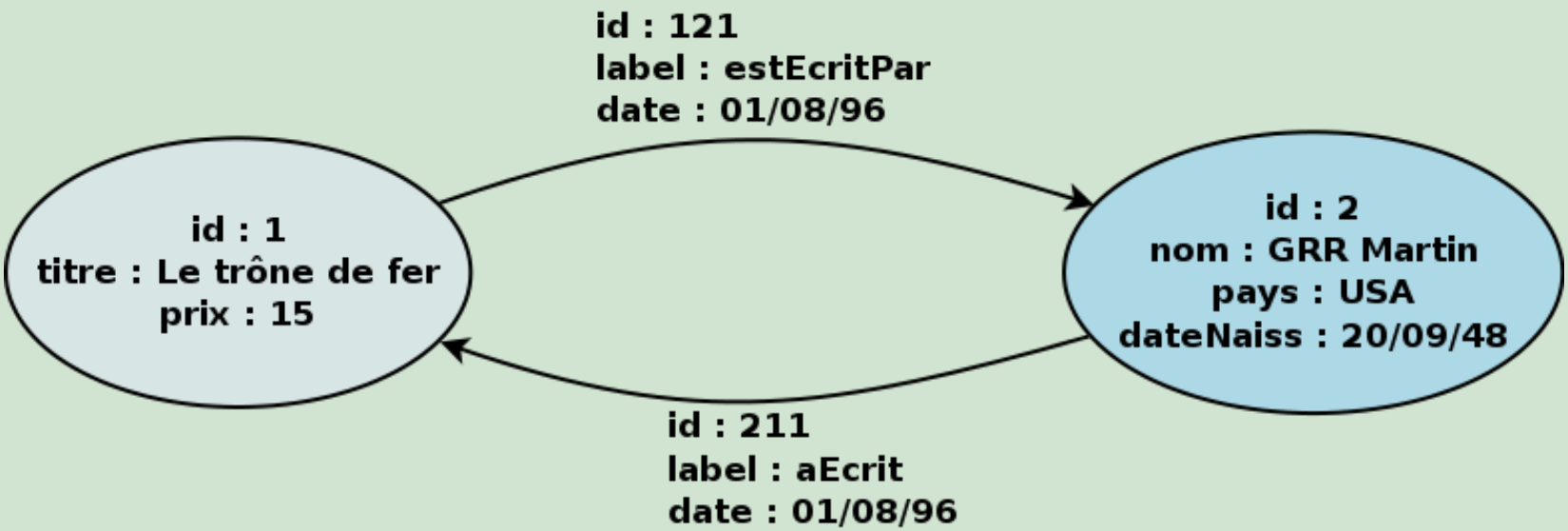


---

http://www.neo4j.org/
http://www.franz.com/agraph/allegrograph/
http://virtuoso.openlinksw.com/

BD orientée graphes (3)

Exemple d'une BD orientée graphes



BD orientée documents

BD orientée documents = collection de documents (clé-document)

- Notion abstraite de « document »
- Chaque document a des champs et une clé
- Deux instances de document peuvent avoir des champs différents
- Organisation des documents selon des tags, métadonnées, collections

Avantages :

- Recherche de documents basée sur leur contenu
- Facilité d'évolution du schéma

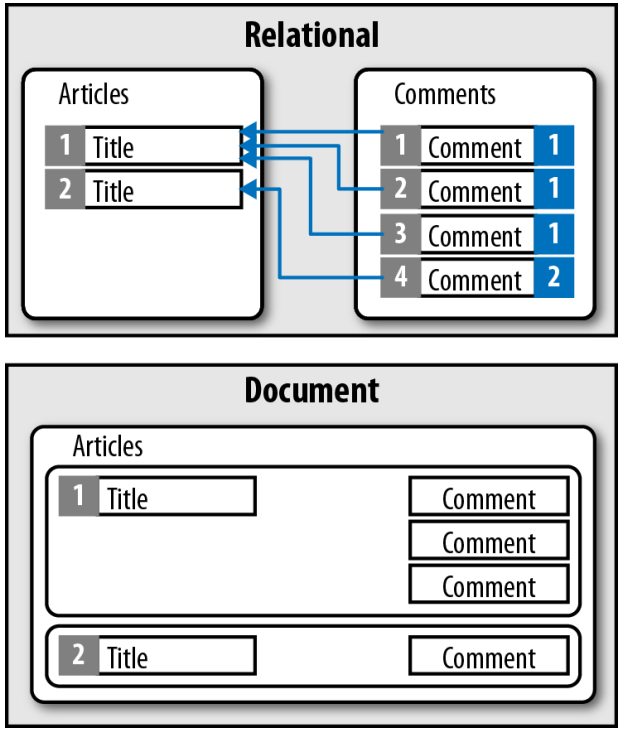
---

http://en.wikipedia.org/wiki/Document-oriented_database

BD orientée documents (2)

SGBD orienté documents :

- CouchDB (stockage JSON, requêtes en Javascript via Map Reduce)
- MongoDB (documents "à la JSON", requêtes en Javascript)
- Couchbase (documents JSON)
- ...

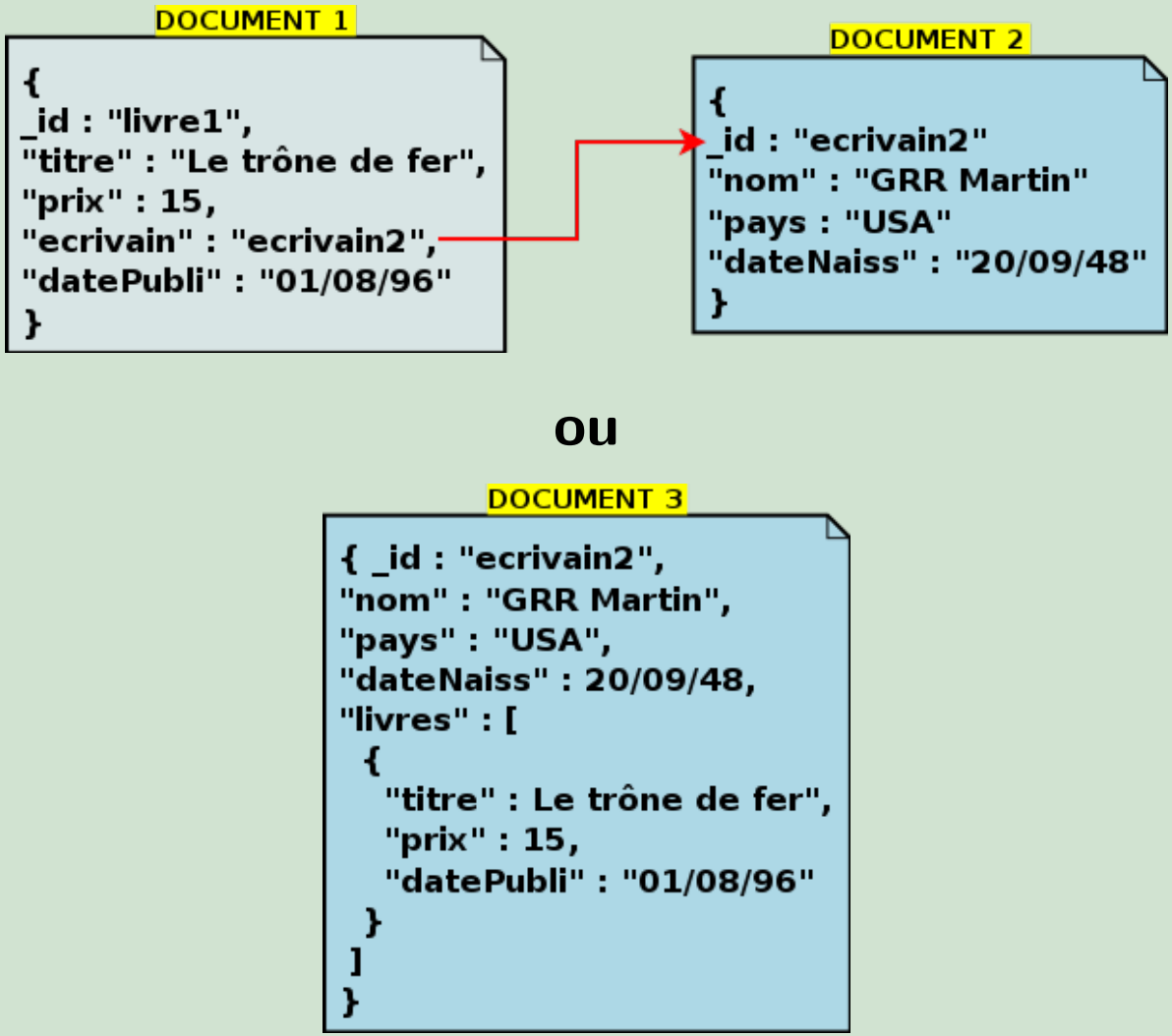


Performances (++), passage à l'échelle (∼/++), flexibilité (++), prise en main (+), nombre de fonctionnalités (∼/−)

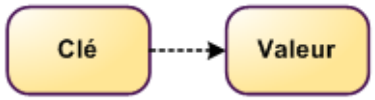
`http://couchdb.apache.org/`
`http://www.mongodb.org/`
`http://www.couchbase.com/`

BD orientée documents (3)

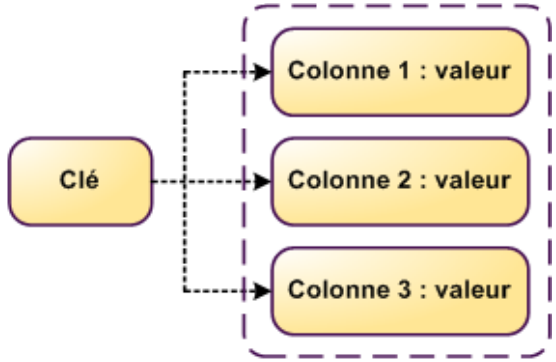
Exemple d'une BD orientée documents



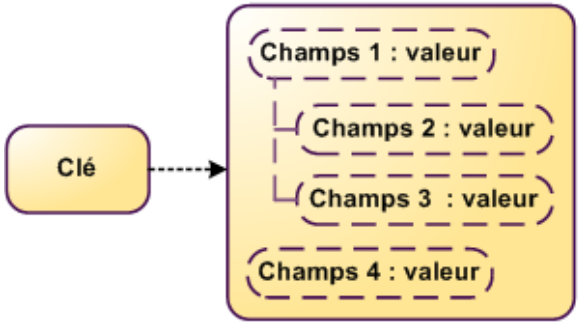
Synthèse des catégories de SGBD No-Rel



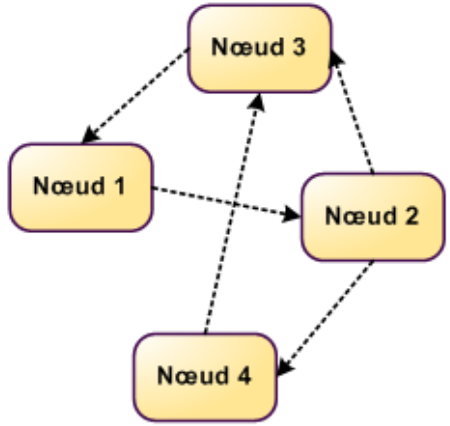
BDD Clé-Valeur



BDD Orientée colonnes



BDD Orientée document



BDD Orientée graphe

http://blog.xebia.fr/

D'autres catégories de SGBD NoRel

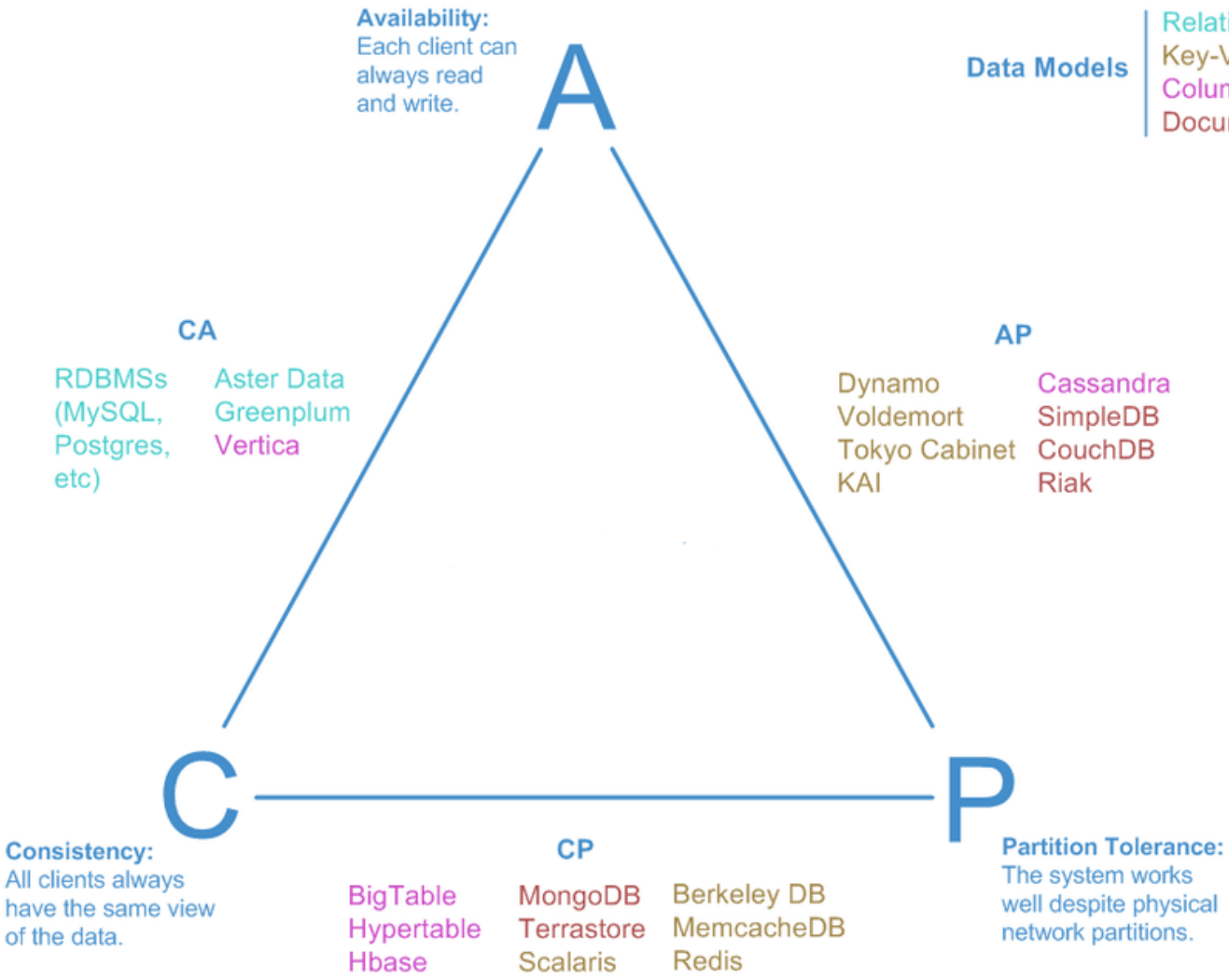
Émergence ou regain d'intérêt pour d'autres modèles :

- BD multi-modèles : OrientDB, AlchemyDB
- BD multi-dimensionnelles (matrices creuses) : Globals, GT.M
- BD multi-valeurs (non respect de la première forme normale, i.e., "pas d'attributs multi-valués") : Reality, OpenQM
- BD basée sur les événements : Event Store
- ...

Sont aussi considérés NoSQL des SGBD gérant du RDF ou des triplets (triplestores)

`http://db-engines.com/en/ranking/`
`http://nosql-database.org/`

En résumé


Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
C
P
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.

Plan du cours

Caractéristiques de MongoDB

Le SGBD MongoDB :

- Orienté documents
- Open-source
- Populaire (5^{eme} SGBD le plus utilisé)
- Passage à l'échelle horizontal (sharding) et réplication
- Système CP (cohérent et résistant au morcellement)
  - "strong consistency" (lectures sur serveur primaire)
  - "eventual consistency" (lectures sur différents serveurs)
- Journalisation
- Utilisation de Map Reduce

`http://www.mongodb.org/`
`http://cookbook.mongodb.org/`
`http://db-engines.com/`

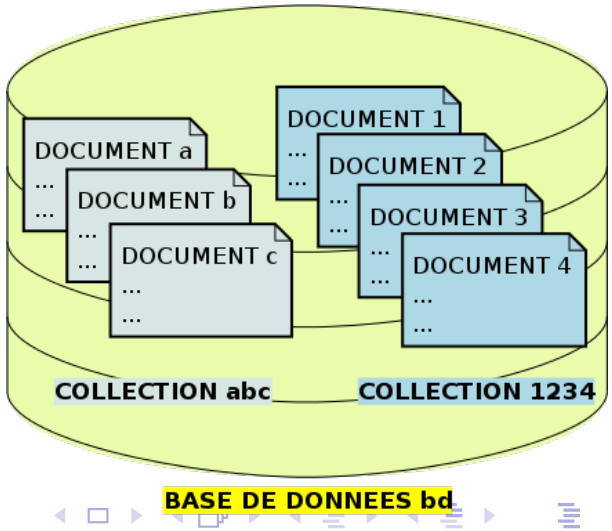
Concepts principaux

Base de données ($\sim$ *base de données* en modèle Relationnel) :

- Ensemble de collections
- Espace de stockage

Collection ($\sim$ *table* en modèle Relationnel) :

- Ensemble de documents qui partagent un objectif ou des similarités
- Pas de "schéma" prédéfini



Concepts principaux (2)

Document ($\sim$ *ligne, tuple* en modèle Relationnel) :

- Un enregistrement dans une collection
- Syntaxe et stockage au format BSON
- Identifiant d'un document (clé **"_id"**)

BSON = "Binary JSON" (JavaScript Object) avec améliorations :

- Ensemble de champs ou paires clé/valeur
- Une valeur peut être un objet complexe (liste, document, ensemble de valeurs, etc.)
- Représentation de nouveaux types (e.g., dates)
- Facilité de parsing (e.g., entiers stockés sur 32/64 bits)

`http://bsonspec.org/`

Concepts principaux (3)

Syntaxe d'un document en MongoDB (format BSON) :

- **_id** est un identifiant (généré ou manuel)
- **att-1** est une clé dont la valeur est une chaîne de caractères
- **att-2** est une clé dont la valeur est un entier
- **att-3** est une clé dont la valeur est une liste de valeurs
- **att-k** est une clé dont la valeur est un document inclus

```
{
_id : <identifiant>,
"att-1" : "val-1",
"att-2" : val-2,
"att-3" : ["val-31", "val-32", ...]
...
"att-k" : {
            "att-k1" : "val-k1",
            ...
          }
}
```

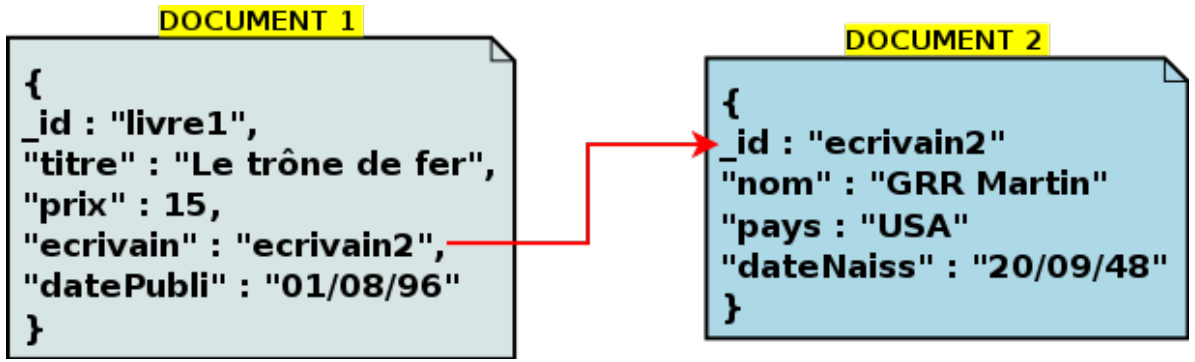
Relations inter-documents

Relation entre les documents de différentes collections :

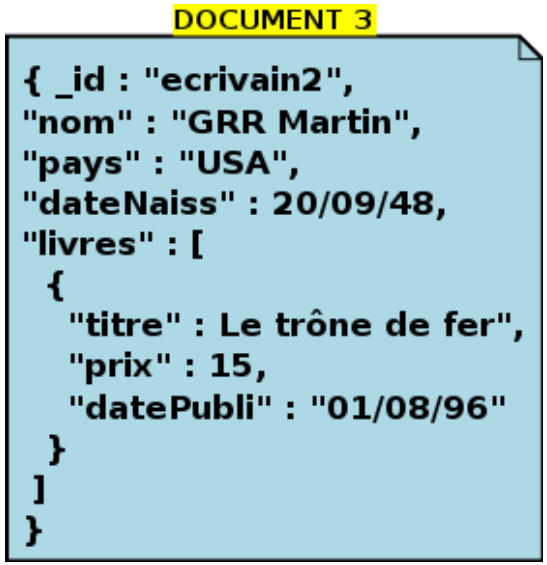
- Par référence : l'identifiant d'un document (son "_id") est utilisé comme valeur attributaire dans un autre document
  - se rapproche du modèle de données normalisées
  - nécessite des requêtes supplémentaires côté applicatif
- Par inclusion ("embedded") : un "sous-document" est utilisé comme valeur
  - philosophie "Non-Relationnel" (pas de jointures)
  - meilleures performances

Relations inter-documents (2)

Relation par référence :



Relation par inclusion (embedded) :



Conception d'une BD

Considérations pour concevoir une BD au niveau physique :

- Accès par une clé ou l'identifiant d'un document
- Pas de schéma ⇒ ajout d'une paire clé/valeur à tout moment
- Pas de jointure ⇒ redondances possibles
- Inclusion ⇒ limite les lectures
- Opérations atomiques sur un seul document, mais pas sur plusieurs ⇒ état incohérent temporaire (souvent tolérable)

Conception d'une BD

Considérations pour concevoir une BD au niveau physique :

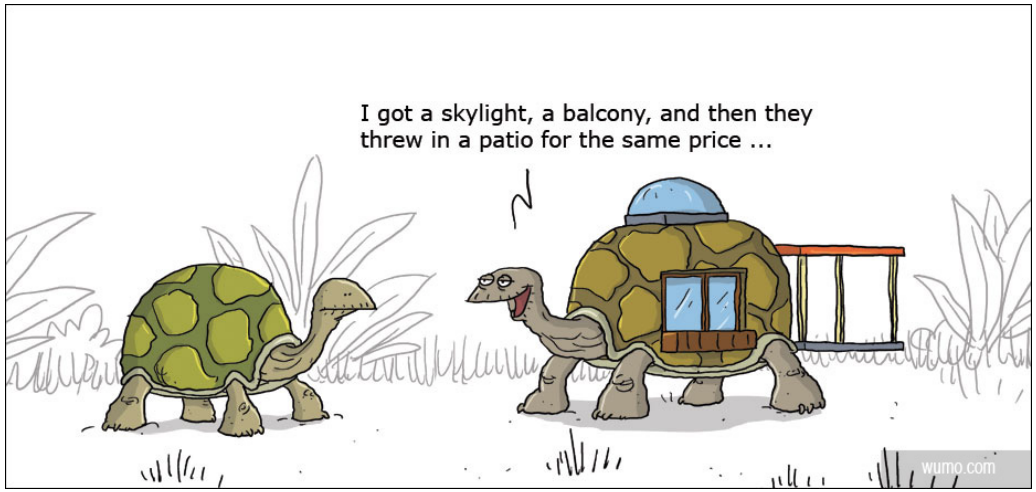
- Accès par une clé ou l'identifiant d'un document
- Pas de schéma ⇒ ajout d'une paire clé/valeur à tout moment
- Pas de jointure ⇒ redondances possibles
- Inclusion ⇒ limite les lectures
- Opérations atomiques sur un seul document, mais pas sur plusieurs ⇒ état incohérent temporaire (souvent tolérable)

"Application-driven" : les besoins applicatifs guident la conception pour identifier, stocker et accéder aux concepts

Conception d'une BD (2)

Pas de méthodologie bien définie :

- **Relation 1 :1** (facture/client) : par inclusion, éventuellement par référence (selon les besoins d'atomicité et de mise à jour)
- **Relation 1 :n** : par référence si le *n* est grand (ville/habitants), sinon par inclusion (article/commentaires)
- **Relation n :m** (livres/auteurs) : par référence



CRUD = IFUR

Opérations disponibles sur les documents :

- Insert
- Find
- Update
- Remove

Toute opération sur un seul document est atomique

Lors d'insertion et mises à jour, la base de données et la collection sont automatiquement créées si elles n'existent pas

---

http://docs.mongodb.org/manual/crud/
http://docs.mongodb.org/manual/core/write-operations-introduction/

Opération INSERT

Syntaxe pour insérer un document dans une collection *coll* :

- <*doc*>, un document ou un tableau de documents à insérer
- <*options*>, un document pouvant contenir :
  - un booléen *ordered* (insertion de plusieurs documents stoppée en cas d'erreur)
  - un document *writeConcern* (type de garantie d'écriture)

```
db.coll.insert(
        <doc>,
        <options>
)
```

http://docs.mongodb.org/manual/reference/method/db.collection.insert

Opération INSERT (2)

Concernant l'identifiant du document à insérer :

- Si la clé "_id" est présente dans le document, s'assurer de l'unicité de sa valeur
- Sinon, MongoDB ajoute un identifiant automatiquement (type *ObjectID* sur 12 octets)

Le document optionnel *writeConcern* décrit la garantie du succès d'une opération, avec :

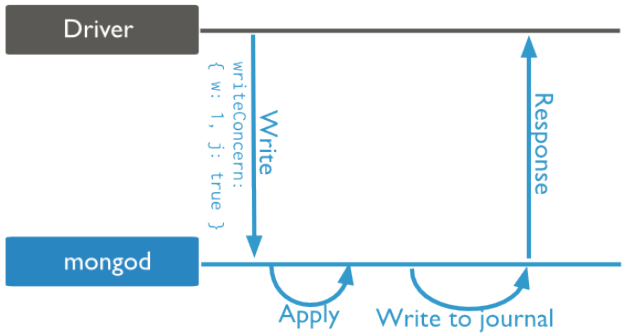
- *w*, nombre de confirmations d'écriture (e.g., *0*, *1*, *"majority"*)
- *j*, un booléen pour écrire dans le journal
- *wtimeout*, une limite de temps en ms

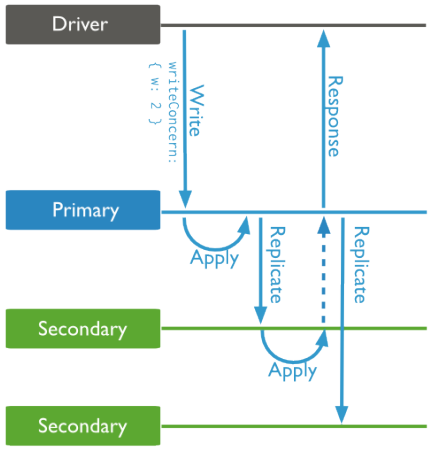
`http://docs.mongodb.org/manual/reference/object-id`
`http://docs.mongodb.org/manual/core/write-concern/`
`http://docs.mongodb.org/manual/reference/write-concern/`

Opération INSERT (3)

```
db.users.insert (          ← collection
   {
     name: "sue",          ← field: value
     age: 26,              ← field: value
     status: "A"           ← field: value
   }                                          } document
)
```

Exemple d'insertion d'un document dans la collection "users"





Exemples de writeConcern : "journal" et "2 écritures"

Opération FIND

Syntaxe pour rechercher des documents dans **une** collection *coll* :

- *<critères>*, un document contenant les critères de sélection des documents pertinents
- *<projection>*, un document contenant les champs qui seront présents dans les documents résultats

```
db.coll.find(
        <critères>,
        <projection>
)
```

Retourne un ensemble de documents (plus exactement un curseur vers ces documents)

---

http://docs.mongodb.org/manual/reference/method/db.collection.find

Opération FIND (2)

Le document *<projection>* :

- Contient soit une liste de champs projetés, soit une liste de champs exclus
- Syntaxe : { clé$_1$: <boolean>, ..., clé$_k$: <boolean>}
- La valeur du booléen détermine la projection de l'attribut :
  - 1/*true* ≡ attribut projeté (dans les documents résultats)
  - 0/*false* ≡ attribut non projeté (exclu)
- Champ *"_id"* inclus par défaut, mais possibilité de l'exclure (y compris dans un document qui liste des champs projetés)

Tous les champs sauf *prix* et *titre* : {prix : false, titre : false}
Uniquement le champ *prix* : {prix : 1, _id : 0}

`http://docs.mongodb.org/manual/reference/operator/query/`

Opération FIND (3)

Le document <*critères*> permet de :

- Retourner tous les documents d'une collection

```
db.coll.find( {} )
```

- Retourner des documents dont la valeur d'une clé est égale à une constante

```
db.coll.find( {clé1 : <const>} )
```

- Retourner des documents en utilisant des opérateurs

```
db.coll.find( {clé1 : <opérateur>, ..., clék : <opérateur>} )
```

`http://docs.mongodb.org/manual/reference/operator/query/`

Opération FIND (4)

Différents types d'opérateurs :

- Comparaison de valeur (*$ne*, *$gt*, *$gte*, *$lt*, *$lte*)

Documents avec une quantité supérieure à 10 :

db.coll.find({quantité : {$gt : 10}})

Documents avec une quantité inférieure ou égale à 25 :

db.coll.find({quantité : {$lte : 25}})

Documents avec une quantité différente de 50 :

db.coll.find({quantité : {$ne : 50}})

- Comparaison avec un tableau de valeurs (*$in*, *$nin*)

Documents avec une activité parmi *VTT*, *trail*, *canoë* :

db.coll.find({activité : {$in : ["VTT", "trail", "canoë"]}})

L'opérateur $ne inclut les documents qui ne contiennent pas le champ

Opération FIND (5)

- Comparaison logique (*$and*, *$or*, *$not*, *$nor*)

Documents avec un prix à 5 et une quantité supérieure à 1 :

db.coll.find({$and : [{prix : 5}, {quantité : {$gt : 1}}]})

Documents avec un prix qui ne soit pas supérieur à 5 :

db.coll.find({{prix : {$not : {$gt : 5}}}})

- Existence ou type d'un élément (*$exists*, *$type*)

Documents avec un champ *"prix"* :

db.coll.find({$prix : {$exists : true}})

Opération FIND (6)

- D'évaluation (*$mod*, *$regex*, *$text*, *$where*)

Documents dont l'un des champs contient *JRR* ou *Tolkien* :

```
db.coll.find({$text : {$search : "JRR Tolkien"}})
```

Documents avec un champ *nom* contenant *RR* :

```
db.coll.find({nom : {$regex : '*RR*'}})
db.coll.find({nom : {$regex : /*RR*/}})
```

Idem mais avec option insensible à la casse (*i*) :

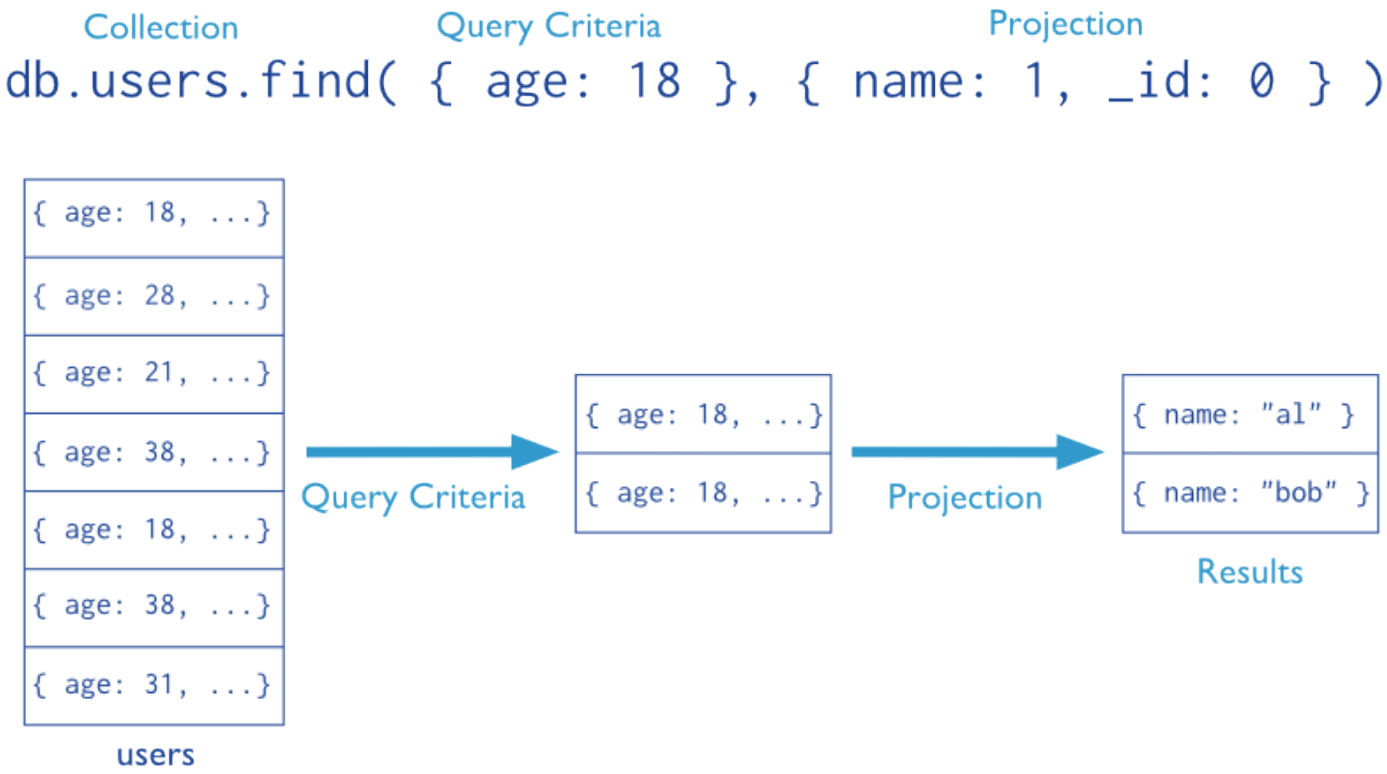
```
db.coll.find({nom : {$regex : '*RR*', $options : 'i' }})
db.coll.find({nom : {$regex : /*RR*/i }})
```

- De tableaux (*$all*, *$elemMatch*, *$size*), spatiaux (*$near*, etc.)

---

http://docs.mongodb.org/manual/reference/operator/query-array/
http://docs.mongodb.org/manual/reference/operator/query-geospatial/

Opération FIND (7)



Exemple de requête avec sélection et projection : retourne tous les documents contenant un champ âge supérieur à 18 en ne gardant que le champ nom (exclusion du champ "_id")

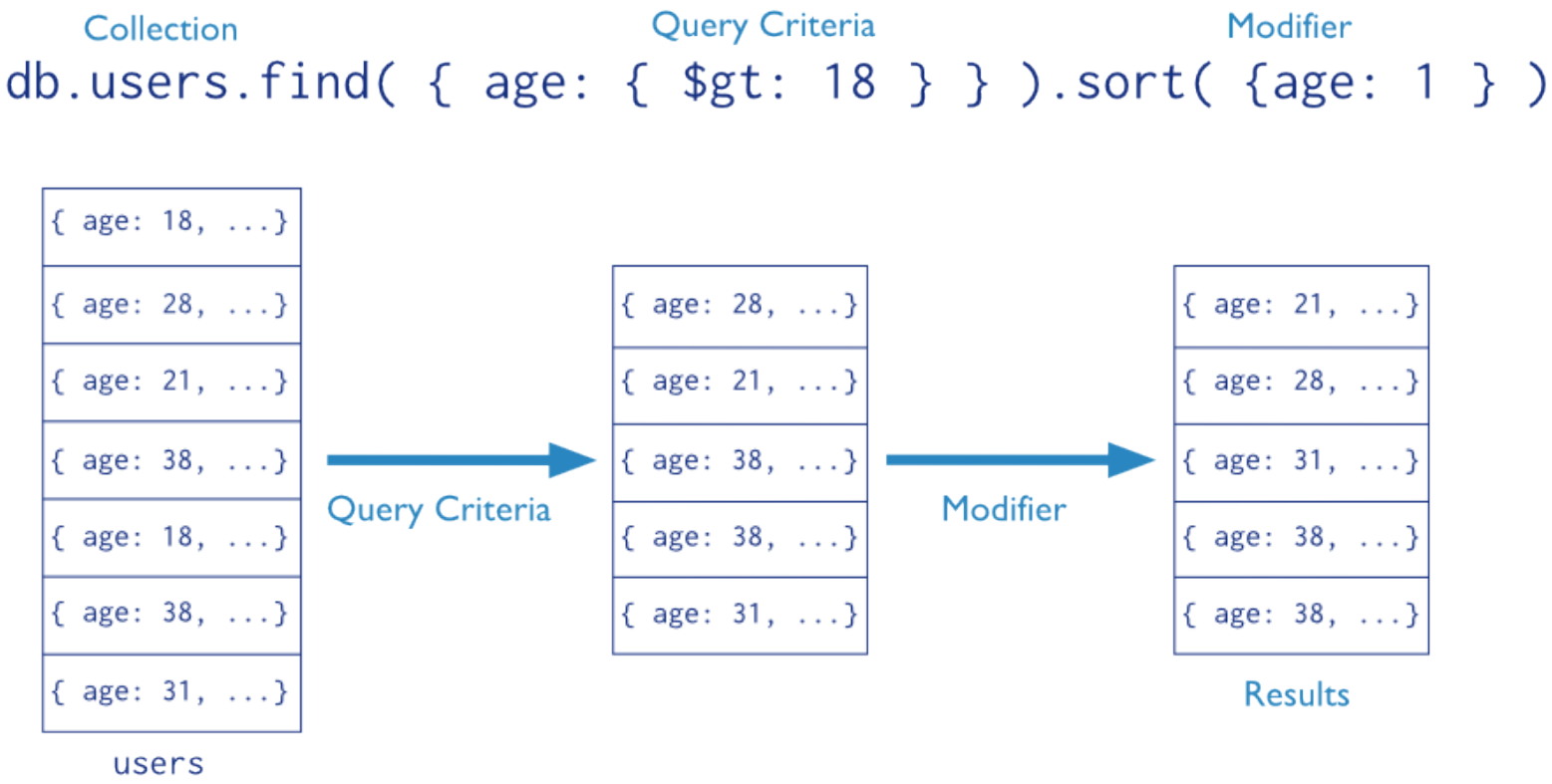
Opération FIND (8)

La méthode FIND retourne un curseur vers une liste de documents, sur lequel il est possible d'appliquer des méthodes *modifiers* :

1. SORT({ CLÉ$_1$: 1|-1, ..., CLÉ$_k$: 1|-1}) : les documents sont triés dans l'ordre naturel (1 pour ascendant, -1 pour descendant), selon le premier champ, puis le second en cas d'égalité, et ce jusqu'au champ k
2. SKIP(N) : les n premiers documents sont supprimés du résultat
3. LIMIT(N) : n documents sont retournés au maximum

Tri sur le prix puis titre : db.coll.find().sort({prix : -1, titre : 1})
Limite de 10 documents : db.coll.find().limit(10)
Combinaison : db.coll.find().sort({prix : -1}).skip(10).limit(50)

Opération FIND (9)



Exemple de requête avec modificateur : retourne tous les documents contenant un champ âge supérieur à 18 et en triant les documents résultats par âge croissant

Opération FIND (10)

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

Exemple de requête sur la collection "users" : retourne les 5 premiers documents contenant un champ âge supérieur à 18 en gardant les champs nom, adresse et "_id"

Opération UPDATE

Syntaxe pour mettre à jour un document dans une collection *coll* :

- <*query*>, un document utilisant des opérateurs de requête
- <*màj*>, un document avec les mises à jour
- <*options*>, un document pouvant contenir :
  - un booléen *upsert* (création d'un document si aucun résultat pour query)
  - un booléen *multi* (pour mettre à jour plusieurs documents)
  - un document *writeConcern* (type de garantie d'écriture)

```
db.coll.update(
        <query>,
        <màj>,
        <options>
)
```

http://docs.mongodb.org/manual/reference/method/db.collection.update

Opération UPDATE (2)

Contenu du document *<màj>* :

- Uniquement des opérateurs de mise à jour :
  - le(s) document(s) retourné(s) par *<query>* ont leurs champs mis à jour
  - opérateurs sur champs (e.g., *$inc*, *$rename*, *$set*)
  - opérateurs sur tableaux (e.g., *$pop*, *$push*, *$addToSet*)

- Uniquement des paires de clé/valeur :
  - le document *<màj>* remplace l'intégralité du premier document répondant aux critères de *<query>*
  - la valeur de *"_id"* est conservée

`http://docs.mongodb.org/manual/reference/operator/update/`

Opération UPDATE (3)

```
db.users.update(                          <-- collection
    { age: { $gt: 18 } },                 <-- update criteria
    { $set: { status: "A" } },            <-- update action
    { multi: true }                       <-- update option
)
```

Exemple de mise à jour du statut (nouvelle valeur "A") pour tous les documents (option "multi") contenant un champ âge supérieur à 18

Opération REMOVE

Syntaxe pour supprimer une base de données *db* :

```
db.dropDatabase()
```

Syntaxe pour supprimer une collection *coll* :

```
db.coll.remove()
```

---

http://docs.mongodb.org/manual/reference/method/js-database/
http://docs.mongodb.org/manual/reference/method/js-collection/

Opération REMOVE (2)

Syntaxe pour supprimer des documents de la collection *coll* :

- <*query*>, un document utilisant des opérateurs de requête
- <*options*>, un document pouvant contenir :
  - un booléen *justOne* (pour supprimer un seul document)
  - un document *writeConcern* (type de garantie d'écriture)

```
db.coll.remove(
        <query>,
        <options>
)
```

http://docs.mongodb.org/manual/reference/method/db.collection.remove/

Opération REMOVE (3)

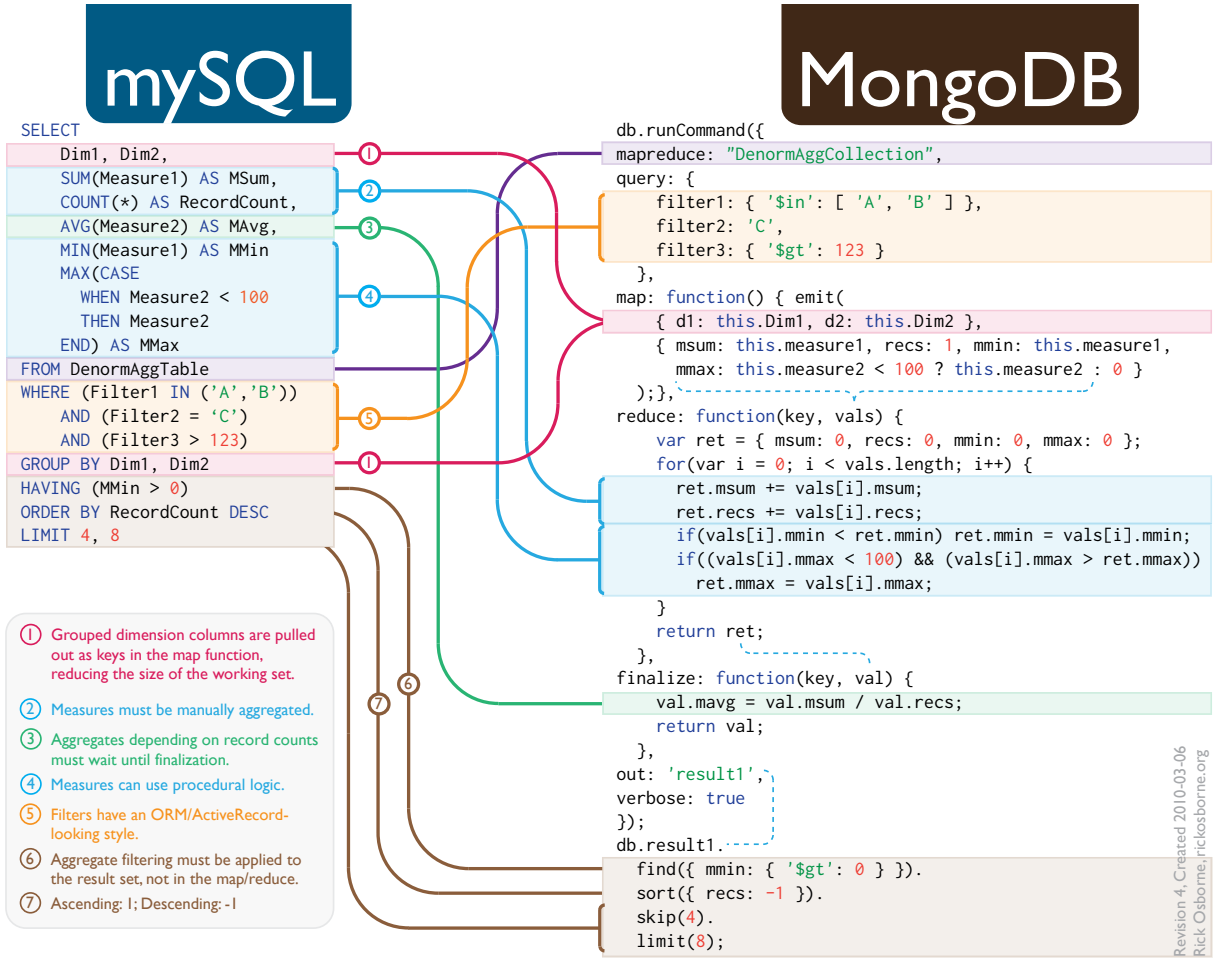
```
db.users.remove(
    { status: "D" }
)
```

collection

remove criteria

Exemple de suppression de tous les documents de la collection "users" dont le statut vaut "D"

MapReduce : SQL versus MongoDB



http://docs.mongodb.org/manual/reference/sql-comparison/

En résumé

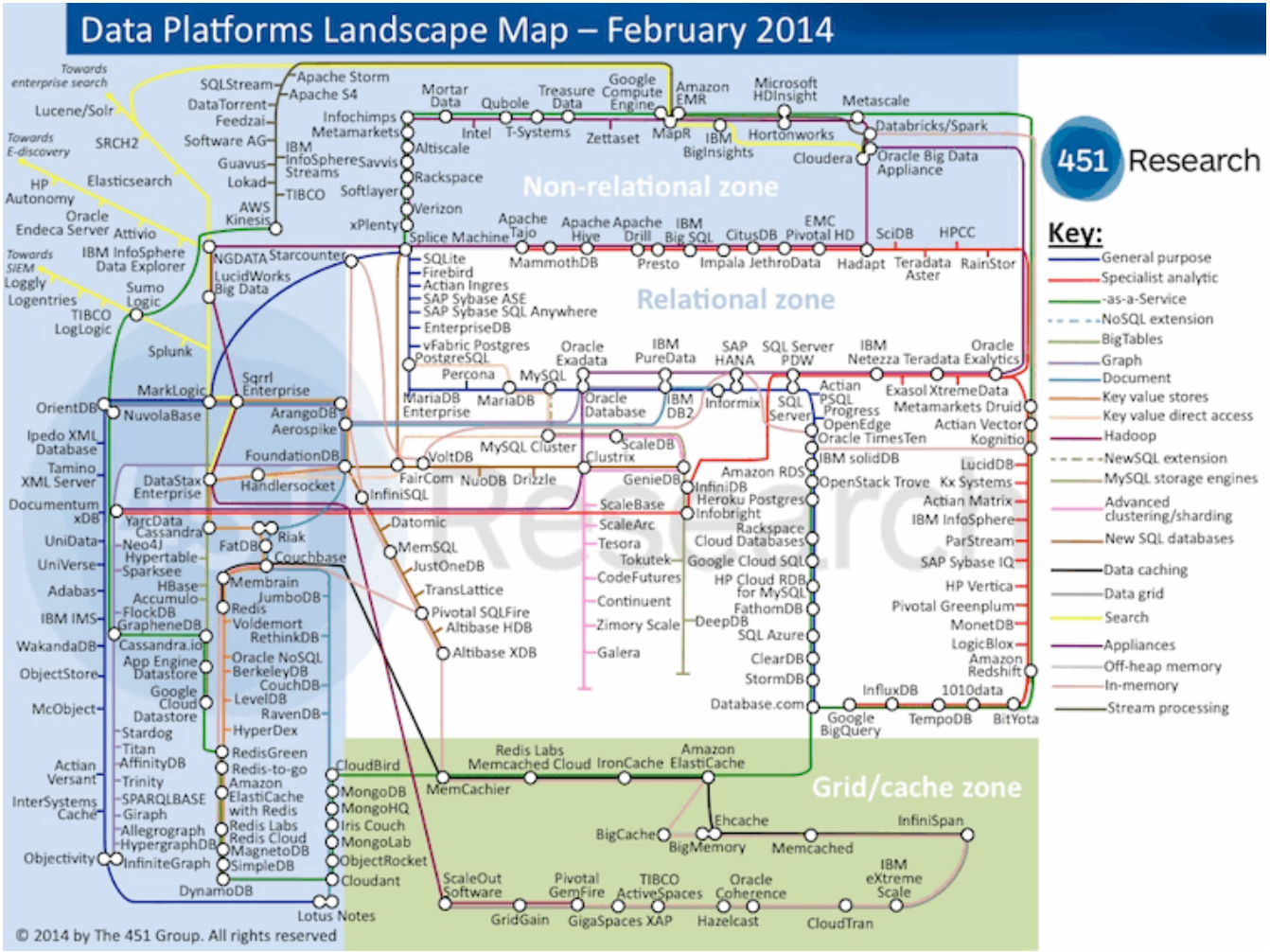
MongoDB, un SGBD orienté documents :

- Requêtes CRUD et requêtes agrégatives (regroupements)
- Indexation
- Aspects sécurité et administration
- Sharding (distribution des données) et réplication
- Map Reduce avec Javascript (distribution des traitements)
- Des "drivers" dans une quinzaine de langages

Vidéos tutorielles ($\sim$ une dizaine sur les concepts, $<$ 20 minutes) :
`http://mrbool.com/course/introduction-to-mongodb/323`

`http://docs.mongodb.org/ecosystem/drivers/`

Bilan



http://blogs.the451group.com/information_management/2014/03/18/updated-data-platforms-landscape-map-february-2014/

Bilan (2)

- **Big Data** + **Web** $\Rightarrow$ nouveaux besoins pour la modélisation, le stockage et le traitement de données **volumineuses**, véloces (**flux**) et variées (**hétérogénéité**)
- **Théorème de CAP** (consistency, availability, partition tolerance)
- **Mouvement NoSQL / NoRel / NewSQL** (paradigmes clé-valeur, colonne, graphe et document)
- **MongoDB** (concepts, modélisation, requêtes IFUR)

Perspective : traitement distribué avec **Map-Reduce**

Des questions ?

http://luc-damas.fr/