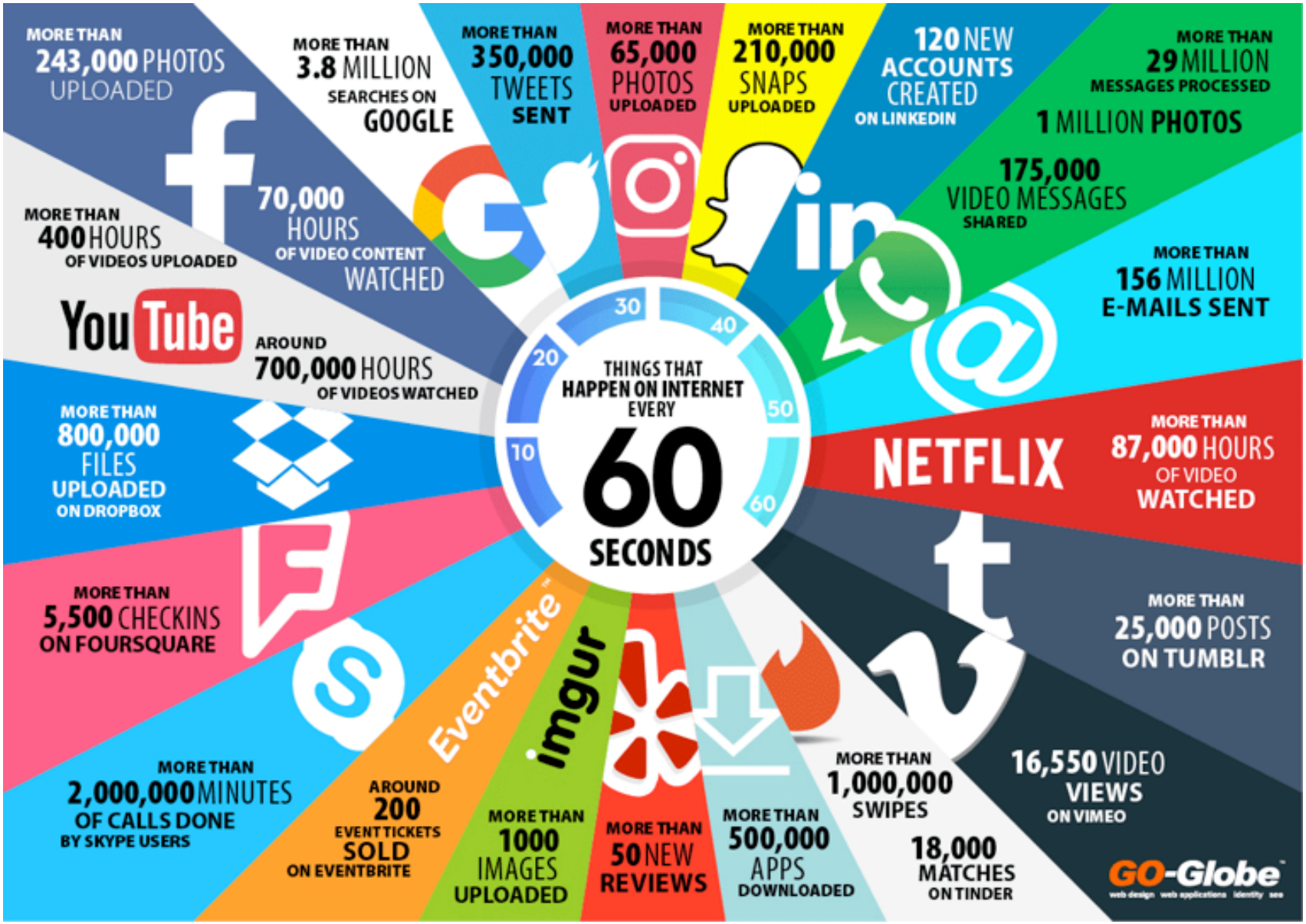
Introduction au NoSQL


MORE THAN 243,000 PHOTOS UPLOADED
MORE THAN 3.8 MILLION SEARCHES ON GOOGLE
MORE THAN 350,000 TWEETS SENT
MORE THAN 65,000 PHOTOS UPLOADED
MORE THAN 210,000 SNAPS UPLOADED
120 NEW ACCOUNTS CREATED ON LINKEDIN
MORE THAN 29 MILLION MESSAGES PROCESSED
1 MILLION PHOTOS
175,000 VIDEO MESSAGES SHARED
MORE THAN 156 MILLION E-MAILS SENT
70,000 HOURS OF VIDEO CONTENT WATCHED
MORE THAN 400 HOURS OF VIDEOS UPLOADED
YouTube
AROUND 700,000 HOURS OF VIDEOS WATCHED
THINGS THAT HAPPEN ON INTERNET EVERY 60 SECONDS
MORE THAN 800,000 FILES UPLOADED ON DROPBOX
NETFLIX
MORE THAN 87,000 HOURS OF VIDEO WATCHED
MORE THAN 5,500 CHECKINS ON FOURSQUARE
MORE THAN 25,000 POSTS ON TUMBLR
MORE THAN 2,000,000 MINUTES OF CALLS DONE BY SKYPE USERS
AROUND 200 EVENT TICKETS SOLD ON EVENTBRITE
MORE THAN 1000 IMAGES UPLOADED
MORE THAN 50 NEW REVIEWS
MORE THAN 500,000 APPS DOWNLOADED
MORE THAN 1,000,000 SWIPES
18,000 MATCHES ON TINDER
16,550 VIDEO VIEWS ON VIMEO
GO-Globe

Big Data

Motivations

- Comment passer à l'échelle ?
  - Des millions d'utilisateurs interagissant avec un système donné
  - Explosion du volume de données à stocker et à traiter
    - 7 To par seconde prévus pour le radiotélescope « Square Kilometre Array »

- Idée:
  - passer d'un système large échelle vertical (« dopage » du serveur) vers un système large échelle horizontal (ajout de machines)

Big Data

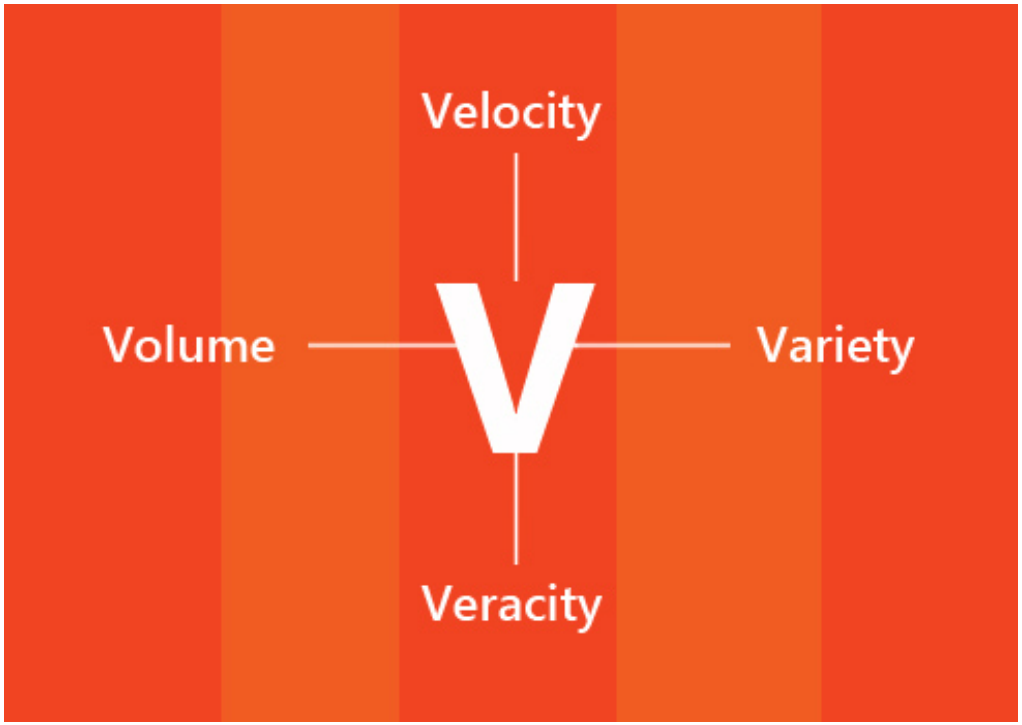
Big Data : modélisation, stockage et traitement (analyse) d'un ensemble de données très volumineuses, croissantes et hétérogènes, dont l'exploitation permet entre autres :

- la prise de décisions
- la découverte de nouvelles connaissances

- Facilité par :
  - le faible coût du stockage
  - le faible coût des processeurs
  - la mise à disposition des données

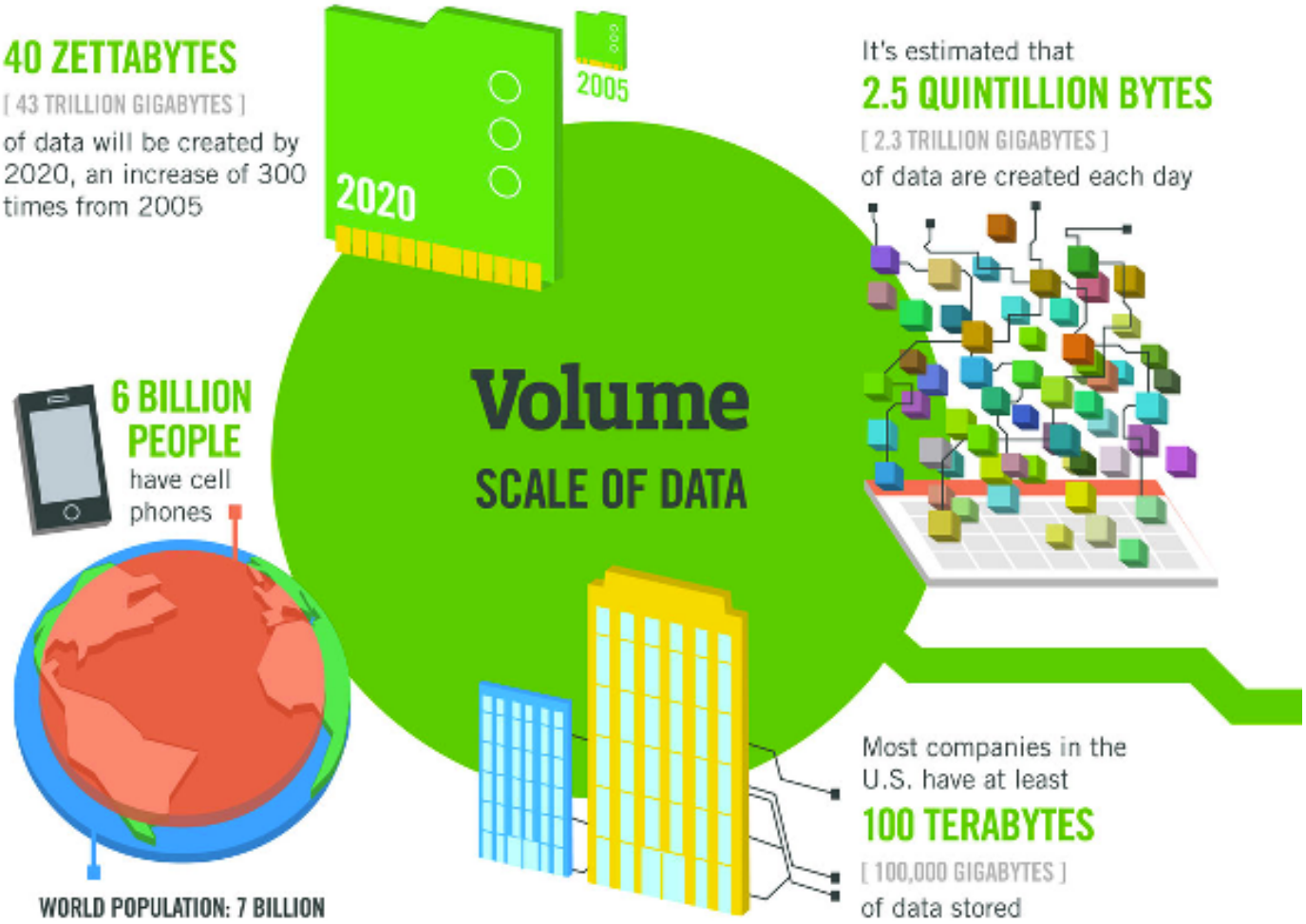
Big Data et motivations

Les 4 V caractéristiques des Big Data



Les 4V

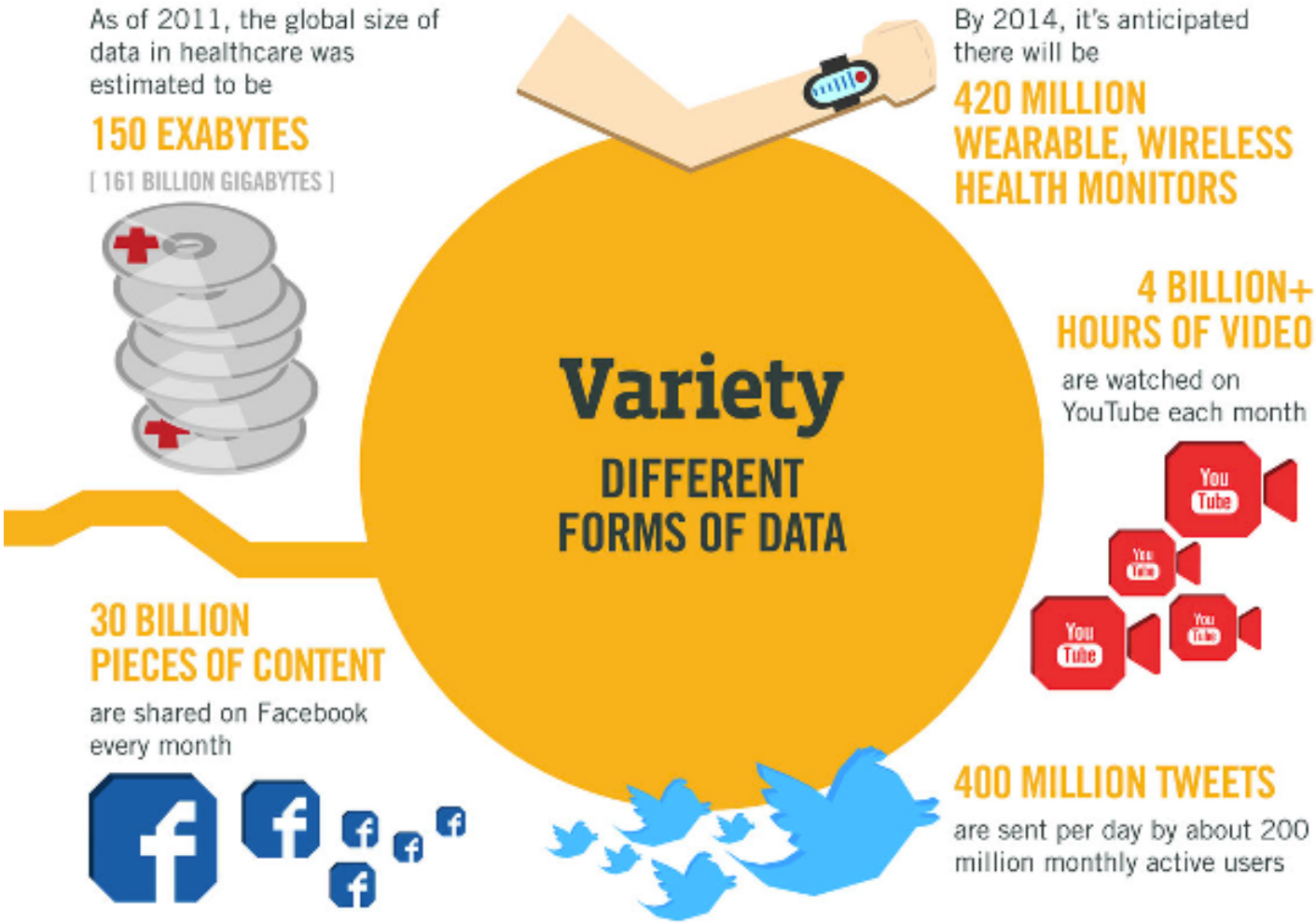
Volume



Source : IBM

Les 4V

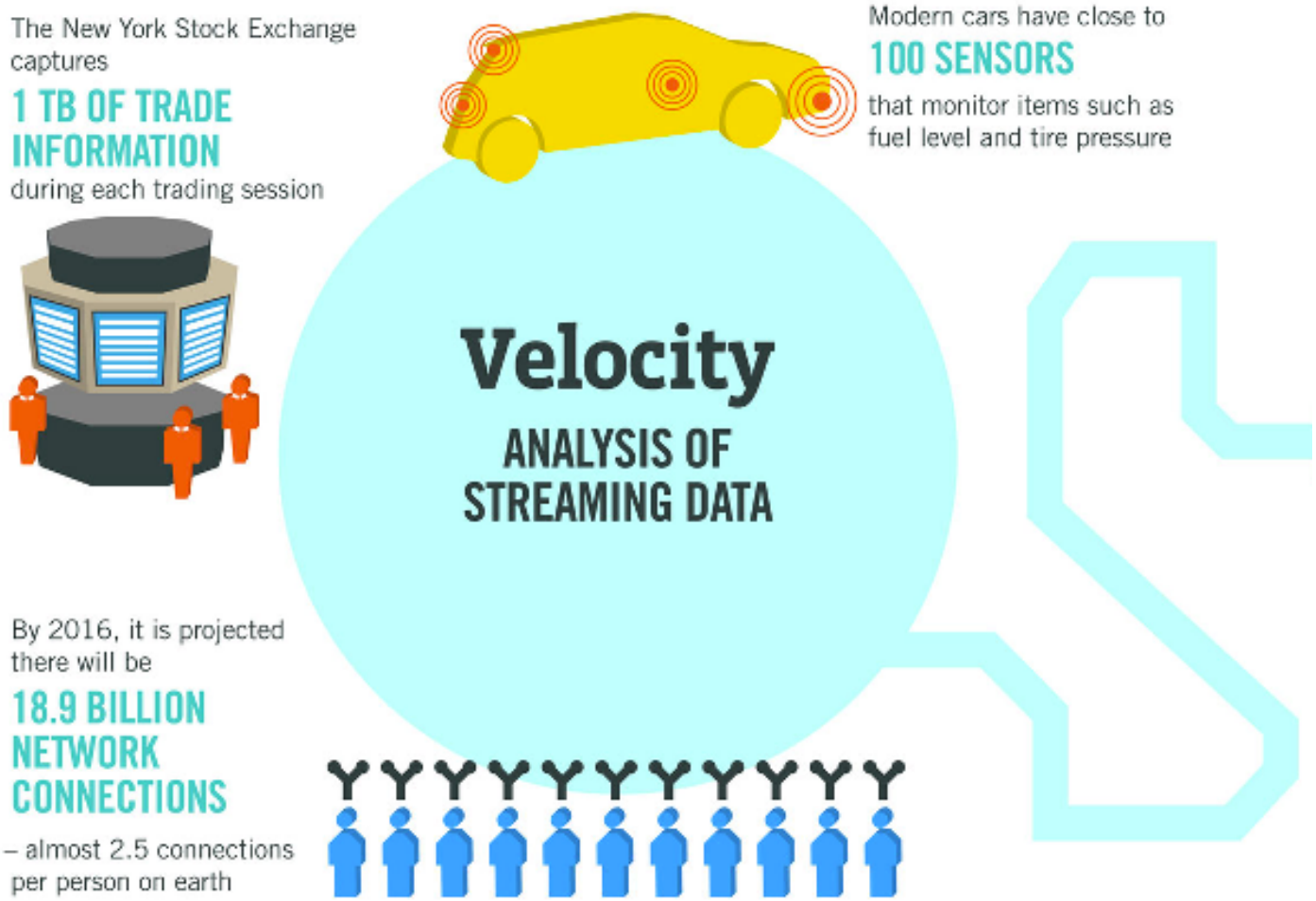
Variété



Source : IBM

Les 4V

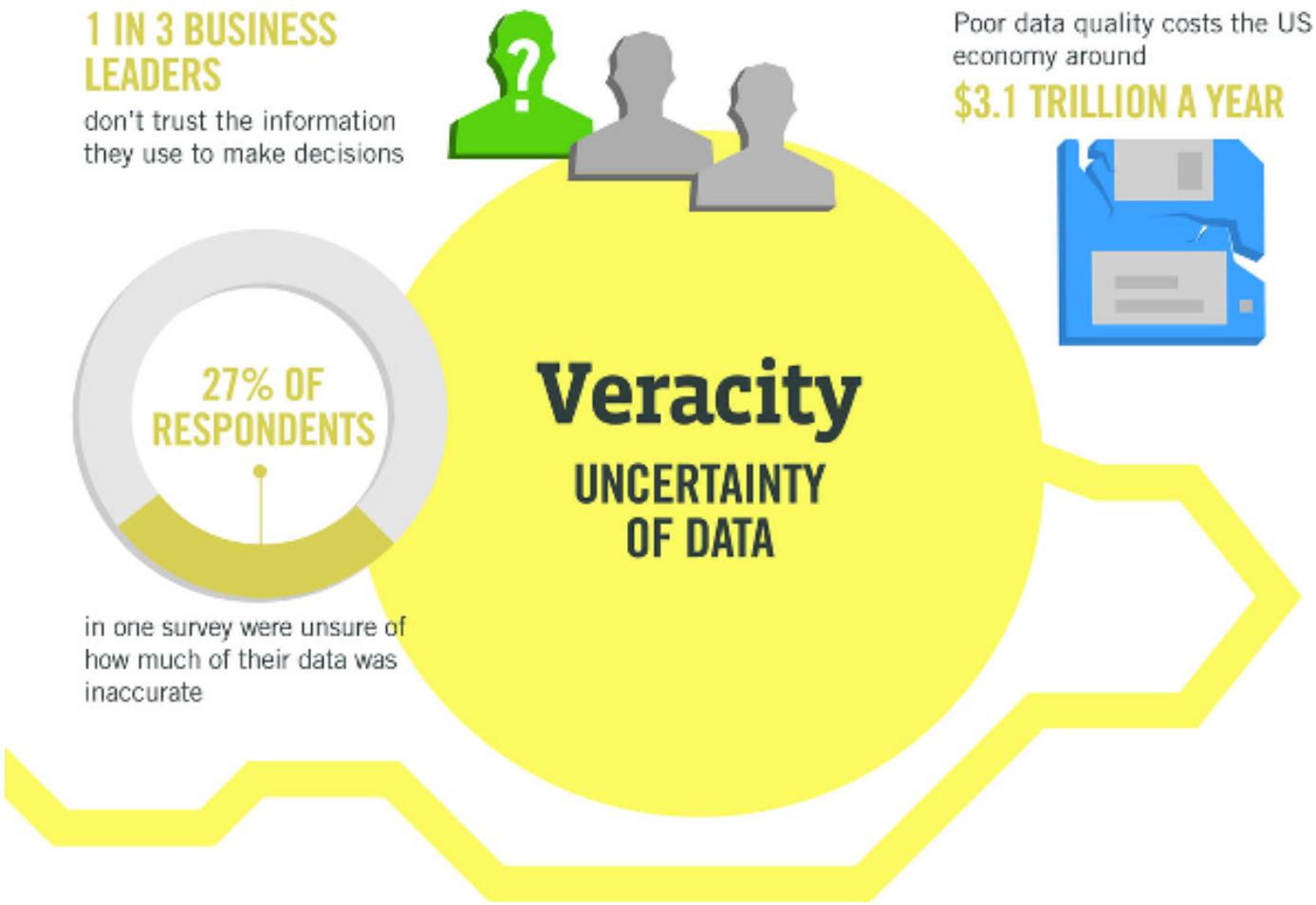
Vélocité



Source : IBM

Les 4V

(Véracité ?)

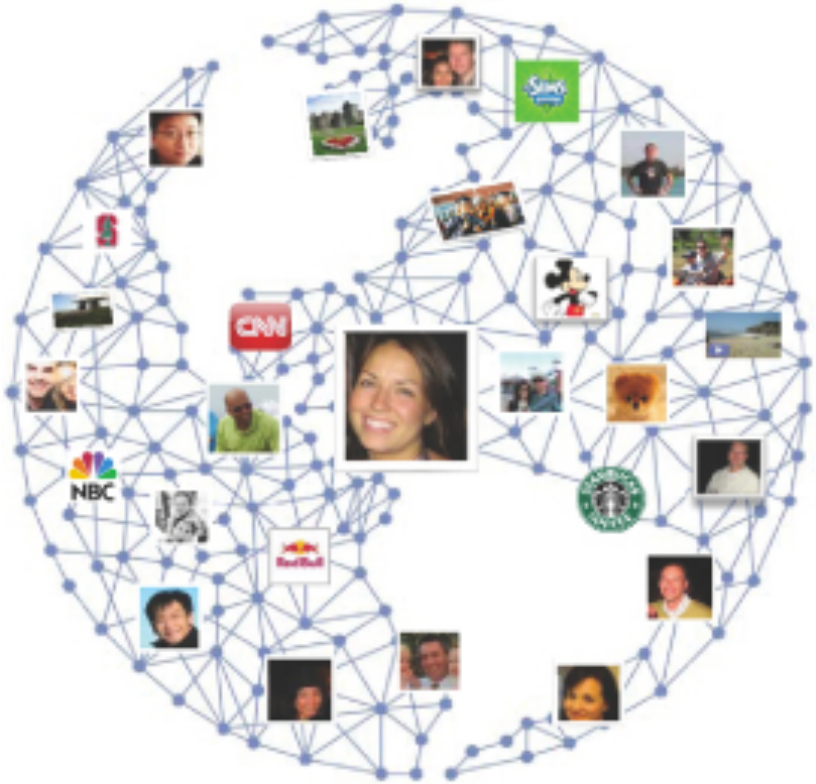


Source : IBM

Géants du Web et Big Data

Facebook

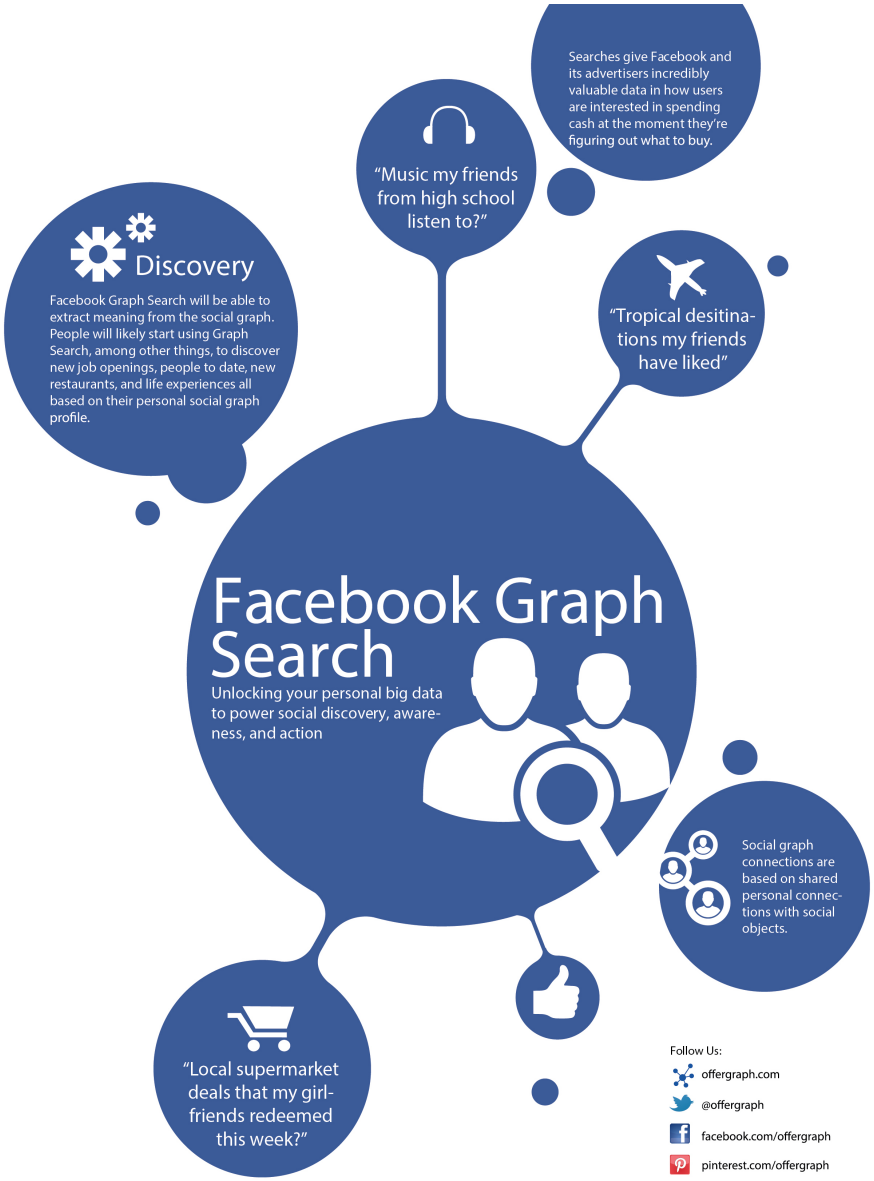
- Chaque jour, Facebook gère :
  - 2,5 milliards d'objets
  - 500 To de nouvelles données
  - 2,7 milliards de « Like »
  - 300 millions de photos intégrées

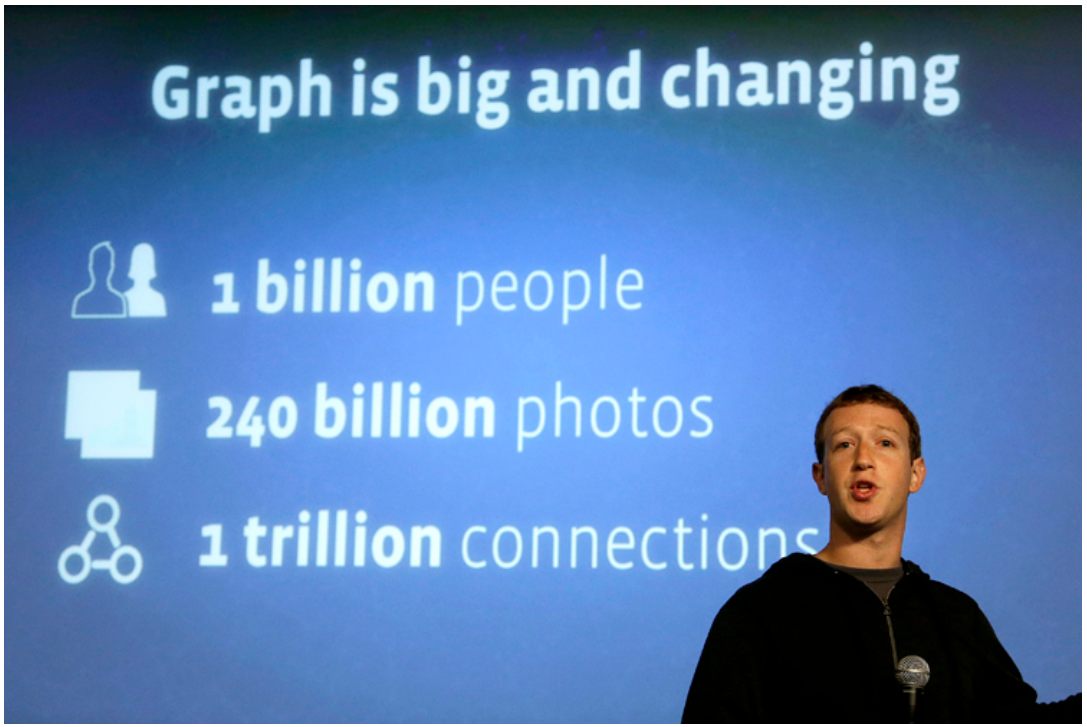



HOW FACEBOOK IS USING BIG DATA –
THE GOOD, THE BAD, AND THE UGLY

Géants du Web et Big Data

Facebook

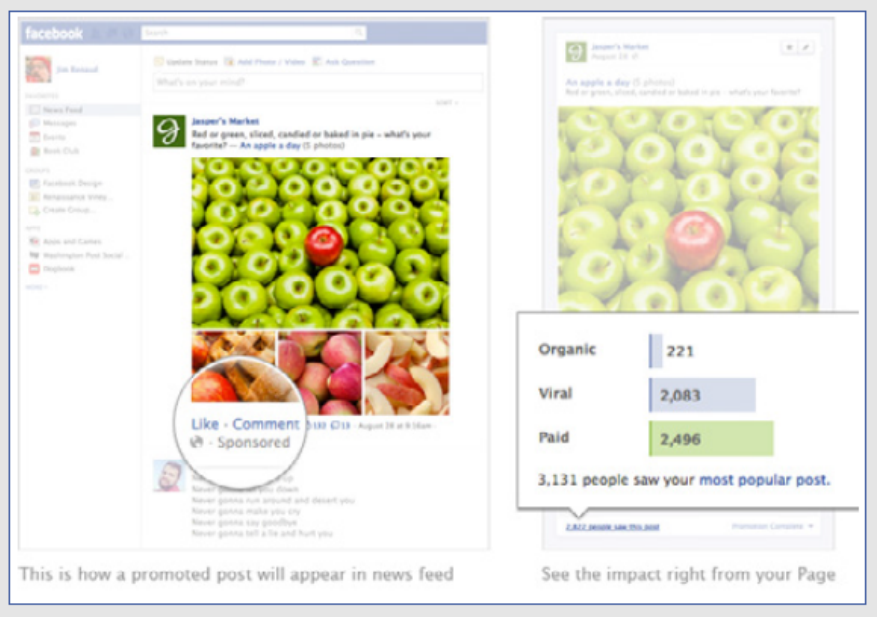




Mark Zuckerberg (Credit: AP/Jeff Chiu), 2013

Géants du Web et Big Data

Facebook



- Profil très détaillé des utilisateurs
  - relations personnelles
  - relations professionnelles
  - goûts musicaux
  - opinions politiques…

Source : ZOG Digital

Géants du Web et Big Data

Facebook

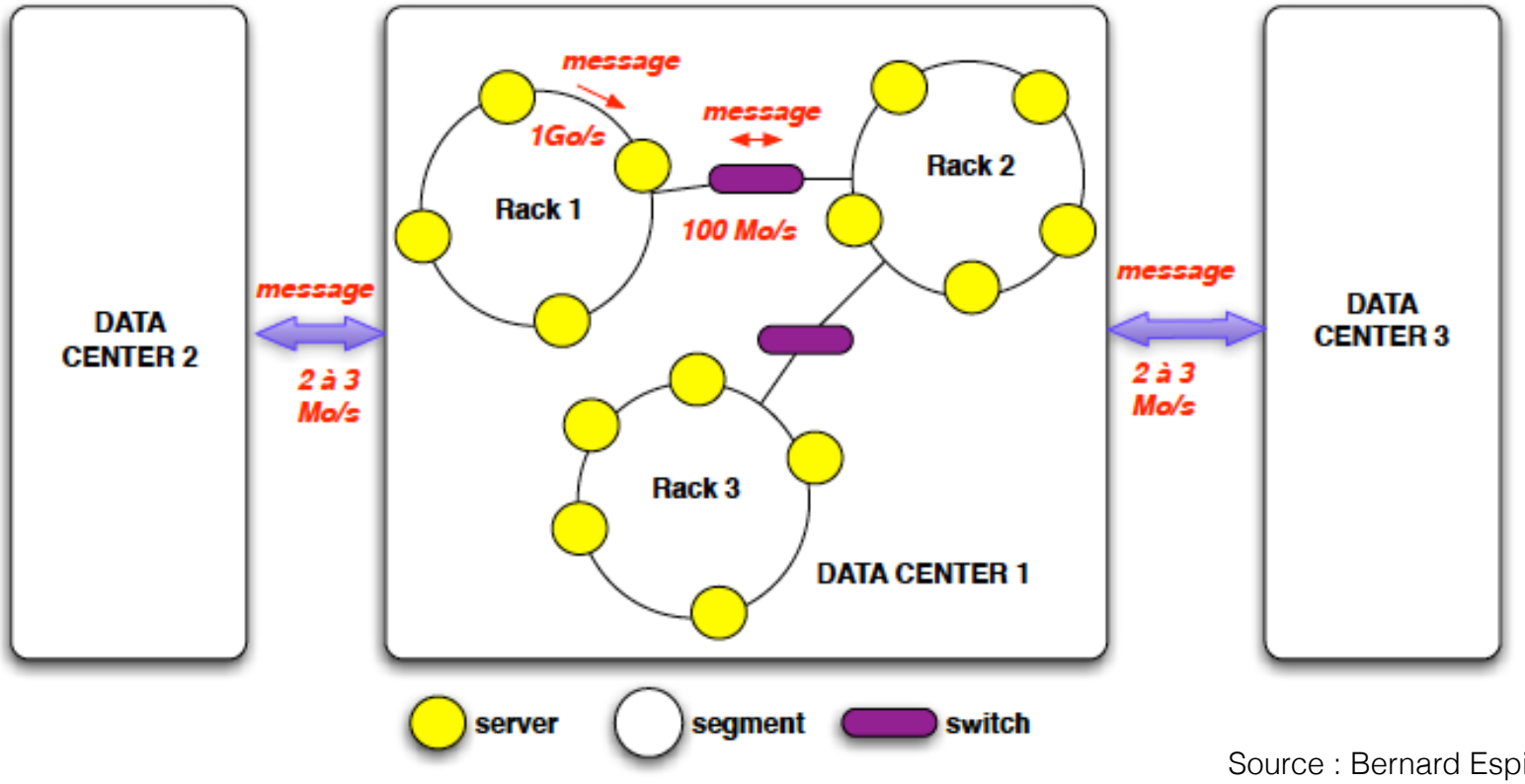
- Data Center de Facebook (2010)
  - 2500 CPu (serveurs)
  - 1 PetaOctet d'espace disque (=1000 Tera = 1 million de Go)

- A développé l'outil **Cassandra**, devenu projet Apache en 2009
  - Non utilisé par Facebook aujourd'hui, qui lui a préféré Hbase en 2010

- Données stockées sur architecture Hadoop dans une base de données HBase
  - Hive : outil d'analyse de données initialement développé par Facebook
  - Volume physique de 100 PetaOctets (2012)

Géants du Web et Big Data

[Aparté sur les Data Centers]



Source : Bernard Espinasse

- Les serveurs communiquent par envoi de messages, ils ne partagent pas de disque ni de ressource de traitement
  - **Architecture « shared nothing »**

Géants du Web et Big Data

Google

- Google a développé son propre système de fichiers distribué : GoogleFS (**GFS**)
  - Environnement de stockage redondant sur un cluster composé de machines « jetables » de puissance moyenne
  - Déploiement d'une solution basée sur les primitives Map et Reduce

- Système de gestion de données basé sur GFS : **BigTable**

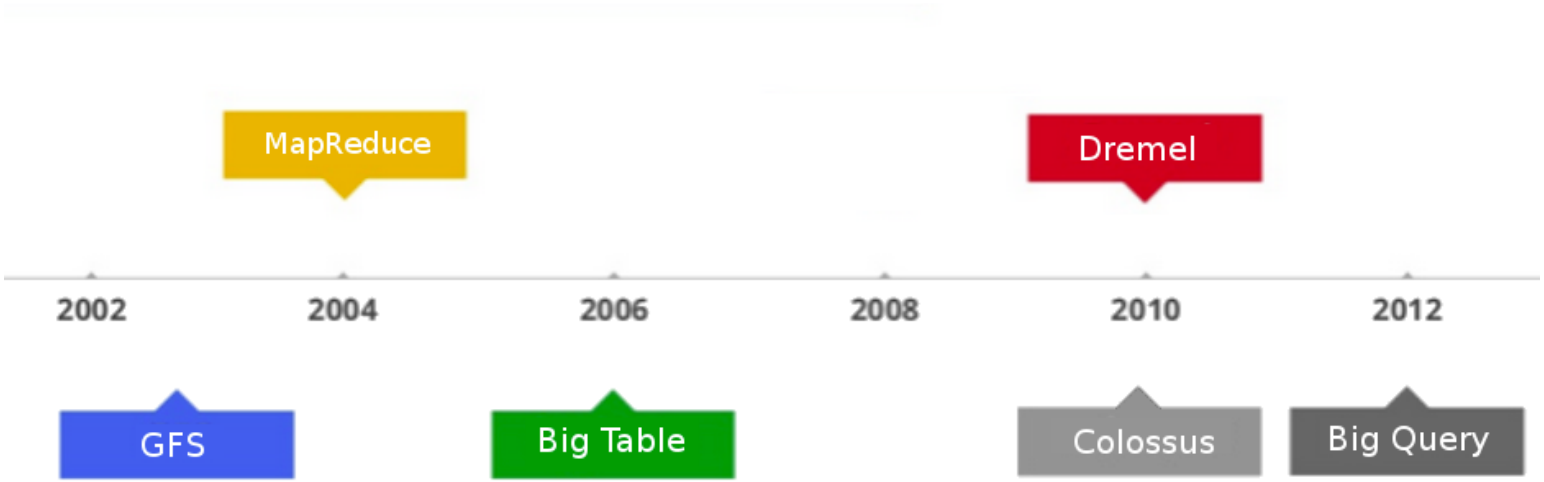
Géants du Web et Big Data

Google

- Google a développé son propre système de fichiers distribué : GoogleFS (**GFS**)
  - Environnement de stockage redondant sur un cluster composé de machines « jetables » de puissance moyenne
  - Déploiement d'une solution basée sur les primitives Map et Reduce
  - **A inspiré l'implémentation libre MapReduce sur système de fichier HDFS (Hadoop File System), Apache**
- Système de gestion de données basé sur GFS : **BigTable**
  - **A inspiré l'implémentation libre Hbase**

Géants du Web et Big Data

Google



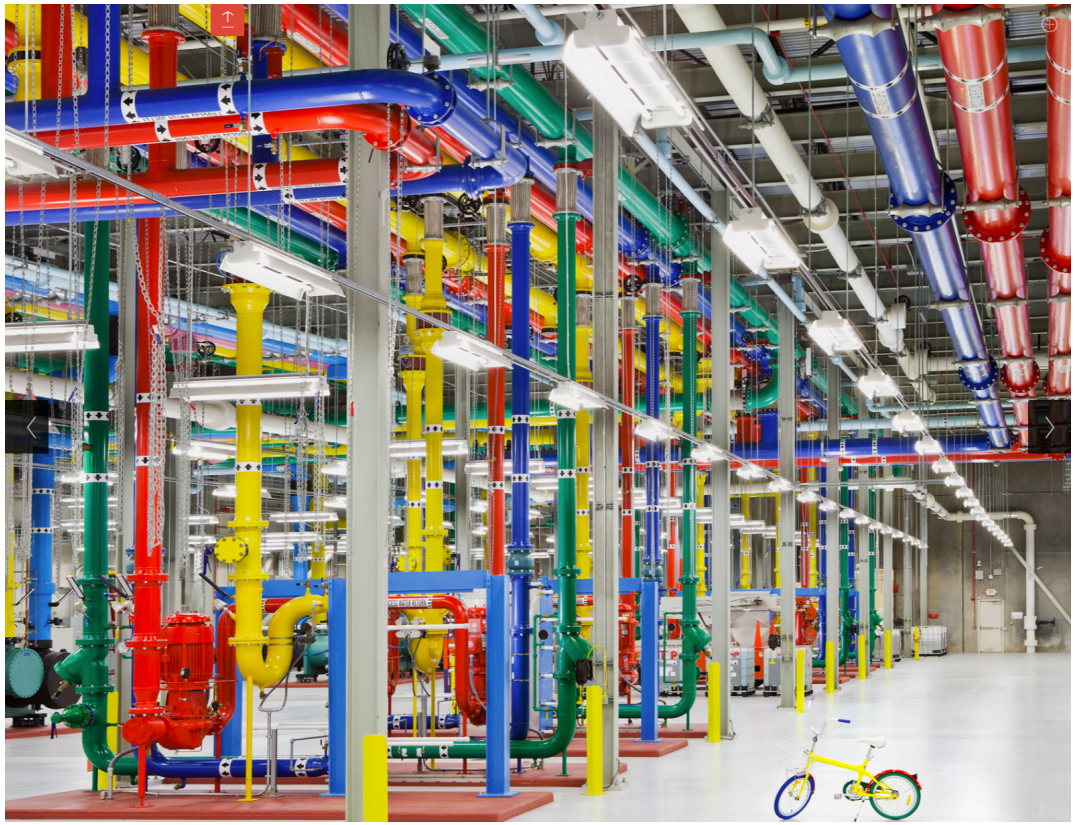
http://www.duchess-france.org/dans-les-coulisses-de-google-bigquery/

- BigQuery créé pour supporter le besoin croissant d'analyse de données
- Proposé maintenant en tant que plateforme (SaaS)
  - « Append-only tables » : on ne peut pas modifier (UPDATE) ou supprimer (DELETE) des entrées dans une table, seulement y ajouter des entrées

Géants du Web et Big Data

Google

- Data Center de Google (2010)
  - Un Data Center Google contient entre 100 et 200 racks, chacun contenant 40 serveurs
  - environ 5000 serveurs par data-center, pour un total de 1 million de serveurs (estimation d'après la consommation électrique)

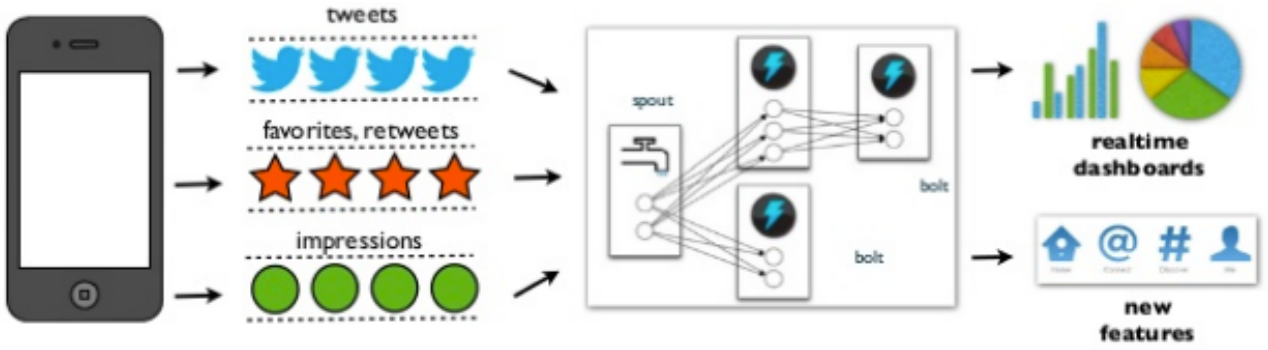


Source: Google
https://www.google.fr/about/datacenters/

Géants du Web et Big Data

Twitter

- Outil **Storm**
  - Traitement temps réel des données réparties sur un cluster de serveurs
  - Idée : proposer une alternative à Hadoop qui réalise ses traitements distribués en mode batch
  - Rendu OpenSource en 2011 (Apache)



http://fr.slideshare.net/KrishnaGade2/storm-at-twitter/

Géants du Web et Big Data

Amazon

- **Dynamo** (2007)
  - Entrepôt de paires clés-valeurs entièrement distribué dans une architecture sans maître
- Dynamo a eu une grande influence dans le monde NoSQL
  - Dynamo n'est pas distribué librement, mais de nombreux projets se sont basés sur ce principe pour proposer une solution libre
    - Cassandra
    - Riak
    - Voldemort (LinkedIn)
- DynamoDB
  - Offre cloud basée sur une technologie semblable

Géants du Web et Big Data




- Ce sont les géants du Web qui ont impulsé le mouvement BigData et NoSQL
- Souvent avec des solutions propriétaires qui ont vu ensuite leur équivalent OpenSource proposé

Bases de données NoSQL

- NoSQL : Not Only SQL
  - Terme inventé en 2009 lors d'un évènement sur les BD distribuées

- Terme vague et incorrect car certains moteurs se basent sur SQL (Cassandra par exemple) mais effet polémique et marketing certain :-)

Et les SGBD relationnels dans tout ça ?

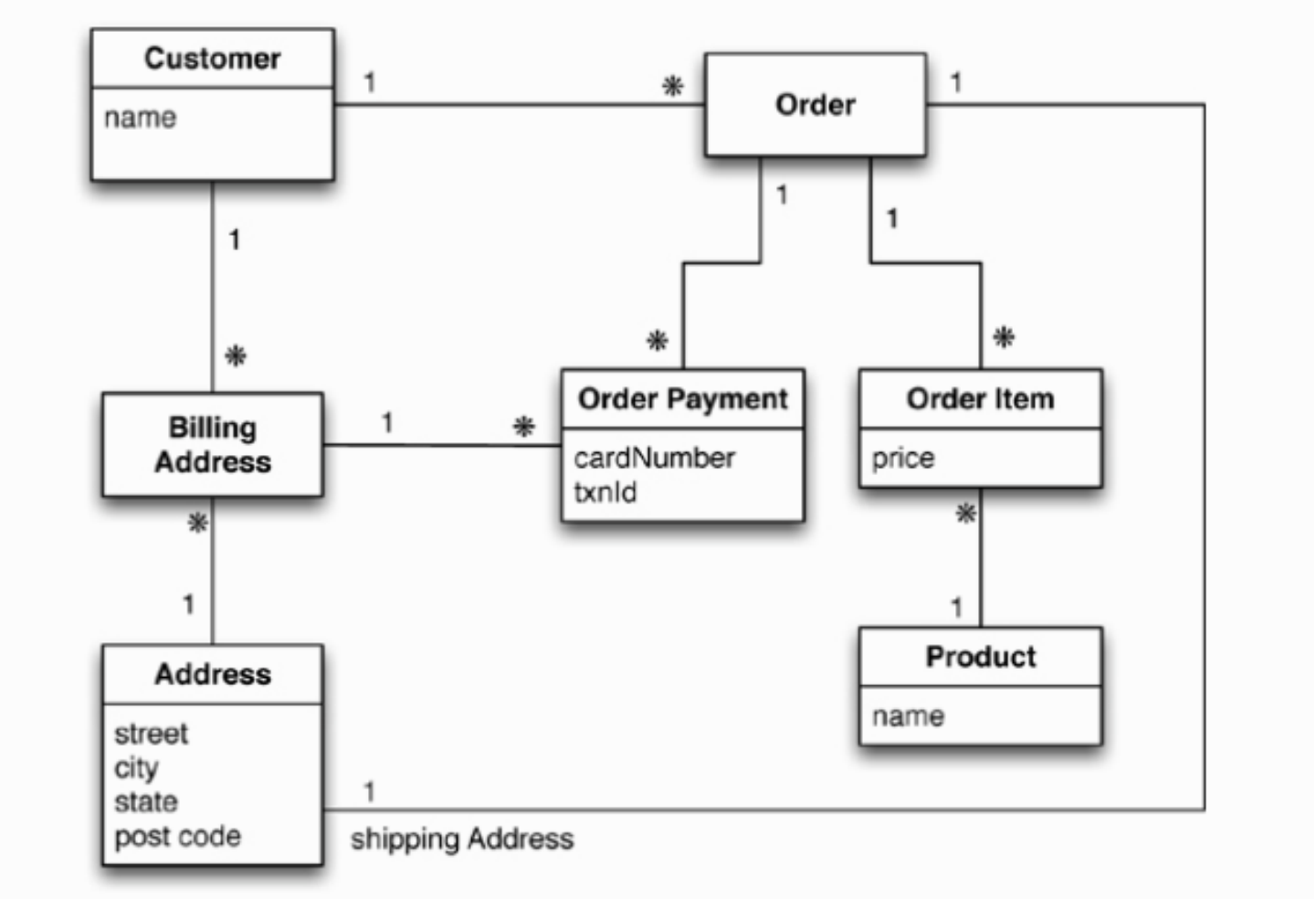
- SGBD relationnels
  - Système de jointure entre les tables permettant de construire des requêtes complexes impliquant plusieurs entités
  - Un système d'intégrité référentielle permettant de s'assurer que les liens entre les entités sont valides
  - Gestion des transactions respectant les contraintes ACID
    - Atomicité
    - Consistence
    - Isolation
    - Durabilité

Et les SGBD relationnels dans tout ça ?

- **Atomicité** : une transaction se fait au complet… ou pas du tout
- **Cohérence** : les modifications apportées sur la base doivent être valides, en accord avec l'ensemble de la base et ses contraintes d'intégrité
- **Isolation** : les transactions lancées au même moment ne doivent jamais interférer
- **Durabilité** : toutes les transactions sont lancées de manière définitive

Exemple d'application

Produits-Clients-Commande



Exemple d'application

Produits-Clients-Commande

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

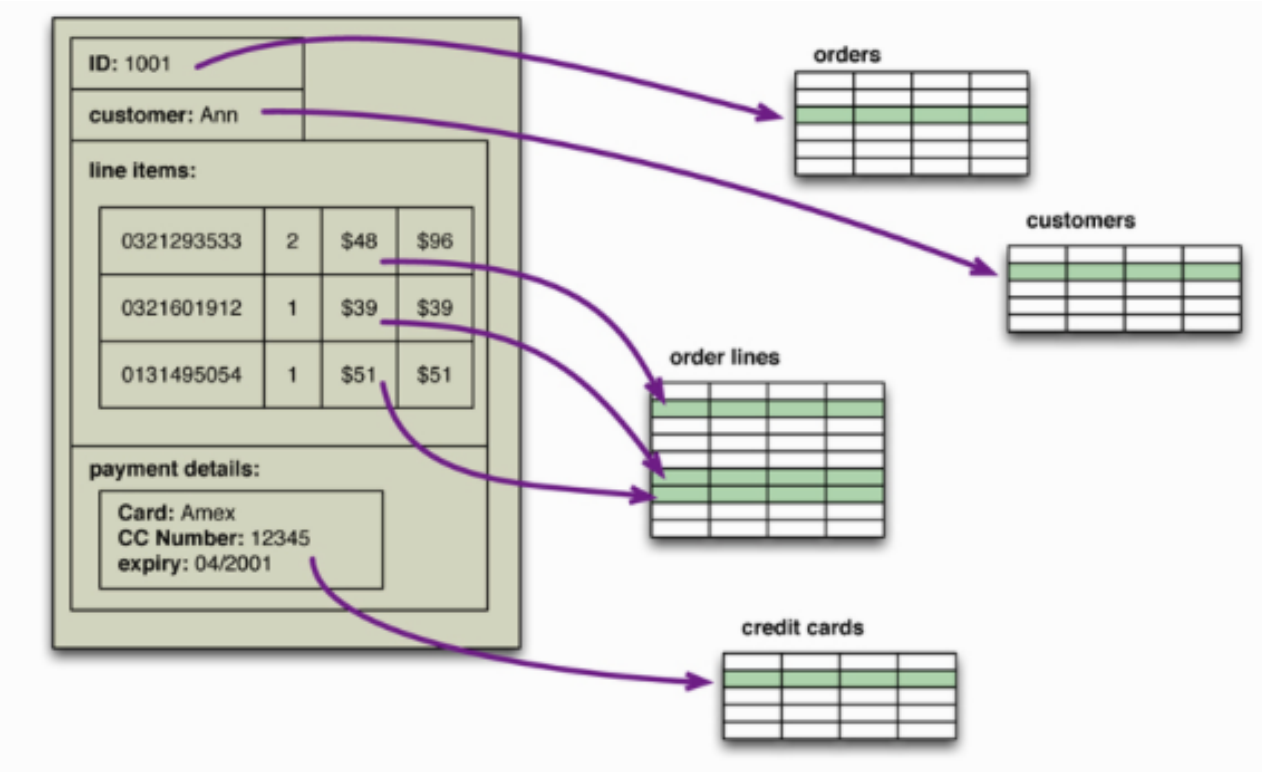
OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Limites des SGBD relationnels

Impedance mismatch (défaut d'impédance)

- Première forme normale :
  - Les valeurs dans les N-uplets doivent être atomiques
- Ce n'est pas le cas pour les structures de données en mémoire qui peuvent être plus riches …
- Il faut transformer les données



Limites des SGBD relationnels

Impedance mismatch

- Pour transformer les données:
  - SGBD orientés objets -> tombés en désuétude
  - Frameworks de mapping objet-relationnel
    - Hibernate, iBatis, …

- … mais les performances de la BD en souffrent…

Limites des SGBD relationnels

les NULL

- SGBDR : schéma rigide
  - Un moteur relationnel doit indiquer d'une façon ou d'une autre l'absence de valeurs dans une cellule
    - marqueur NULL

- Coût en stockage
- Impose la gestion d'une logique à 3 états : vrai, faux et inconnu

```
select *
from Contact
where titre='mme' or titre <> 'mme' or titre is null;
```

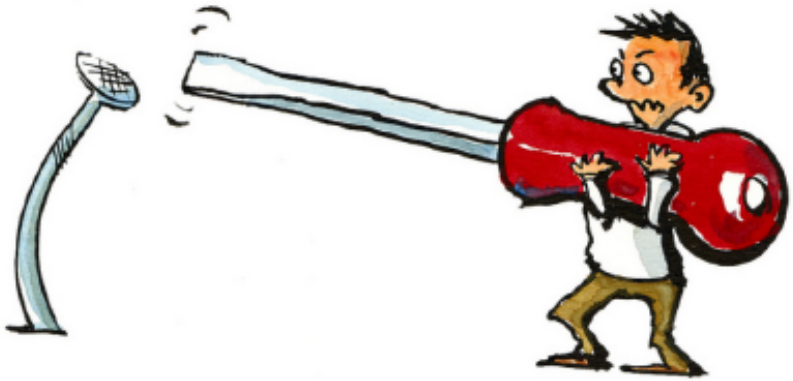
Limites des SGBD relationnels

BD d'intégration vs BD d'application

- Les BD relationnelles sont des BD d'intégration
  - un seul schéma pour toutes les applications, qui stockent leurs données dans une même base.

« One size fits all »

The Law of the Hammer



If the only tool you have is a hammer, everything looks like a nail.

Abraham Maslow - The Psychology of Science - 1966

The Law of the Relational Database

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

A Walk in Graph Databases - 2012

Limites des SGBD relationnels

BD d'intégration vs BD d'application

- Problèmes
  - le schéma est rendu plus complexe à cause de nombreuses applications qui se servent des données
  - si un changement se fait dans le schéma, toutes les applications sont touchées…

=> peut-être faudrait-il faire des BD d'application ?

… et perdre en universalité de schéma…

Limites des SGBD relationnels

Contextes fortement distribués

- Devant l'explosion de la quantité de données et des traitements associés :
  - on booste les serveurs (mémoire, puissance, espace disque)
  - ou on multiplie les petites machines dans un cluster
    - on distribue les données
    - on pousse les programmes vers ces serveurs
      - plus efficace de transférer un petit programme sur le réseau plutôt qu'un grand volume de données

Limites des SGBD relationnels

Contextes fortement distribués

- Coût considérable pour les SGBD relationnels
  - Niveau données
    - Avec la plupart des SGBD relationnels, les données d'une BD liées entre elles sont placées sur le même noeud du serveur
    - Si le nombre de liens est important, il est de plus en plus difficile de placer les données sur des noeuds identiques
  - Niveau traitements
    - Difficile de maintenir avec des performances correctes les contraintes ACID à l'échelle du système distribué entier

Théorème de CAP

- 3 propriétés fondamentales pour les systèmes distribués :
  - **Coherence** (Consistance) : tous les noeuds du système voient exactement les mêmes données au même moment
  - **Partition tolerance** (Résistance au morcellement) : le système étant partitionné, aucune panne moins importante qu'une coupure totale du réseau ne doit l'empêcher de répondre correctement
    - en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome
  - **Availability** (Disponibilité) : la perte des noeuds n'empêche pas les survivants de continuer à fonctionner correctement, les données restent accessibles

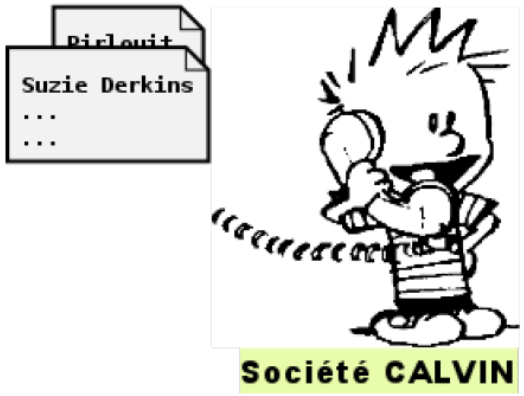
Théorème de CAP (Brewer, 2000) :
Dans un système distribué, il est impossible d'obtenir ces 3 propriétés en même temps, il faut en choisir 2 parmi 3

Théorème CAP

Illustration

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

Principe de mémorisation



Source exemple: Fabien Duchateau

Théorème CAP

Illustration

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

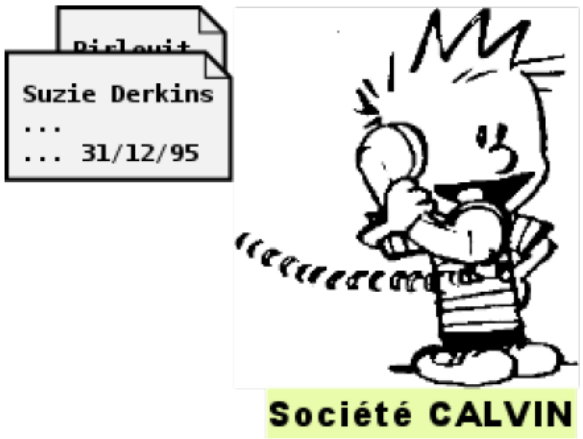
Principe de mémorisation

Théorème CAP

Illustration

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

Principe de restitution



Théorème CAP

Illustration

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

Principe de restitution

Théorème CAP

Illustration

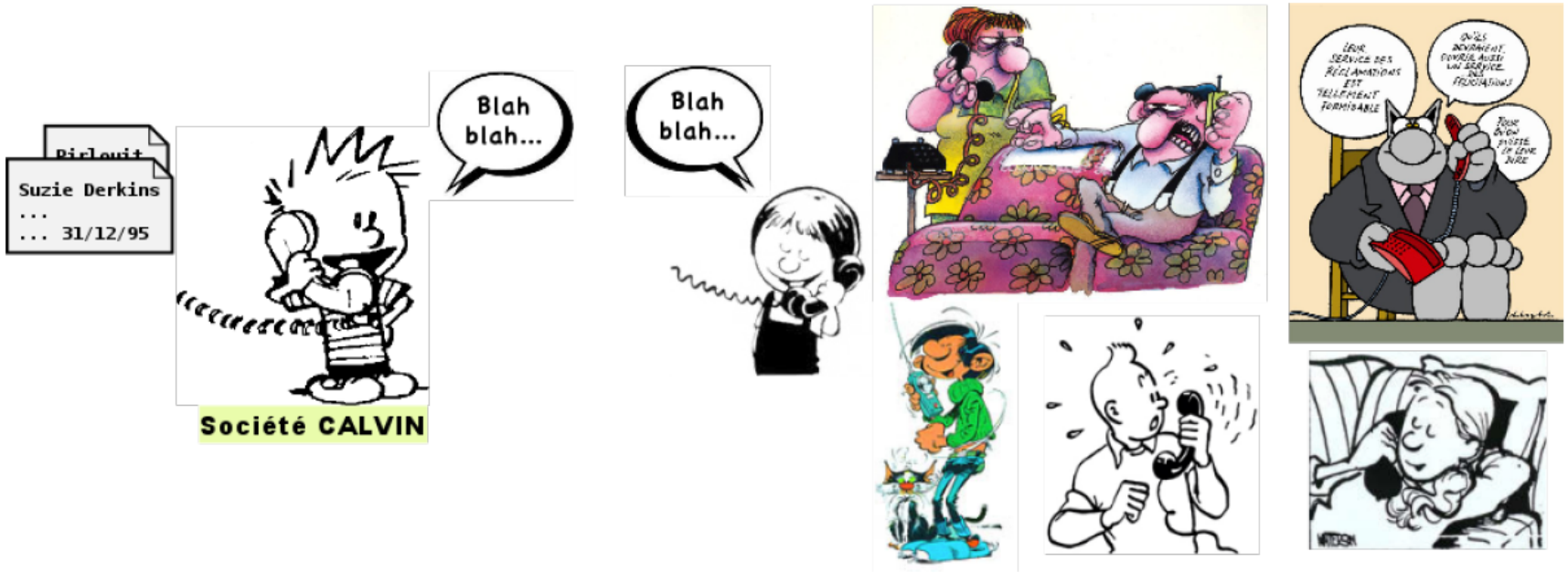
- Calvin décide de créer une entreprise pour mémoriser et restituer des informations fournies par les clients

Principe de restitution

Théorème CAP

Illustration

- Face au succès, Calvin est rapidement débordé d'appels
  - ☑ Cohérence
  - ☑ Résistance au morcellement
  - ☐ ~~Disponibilité~~

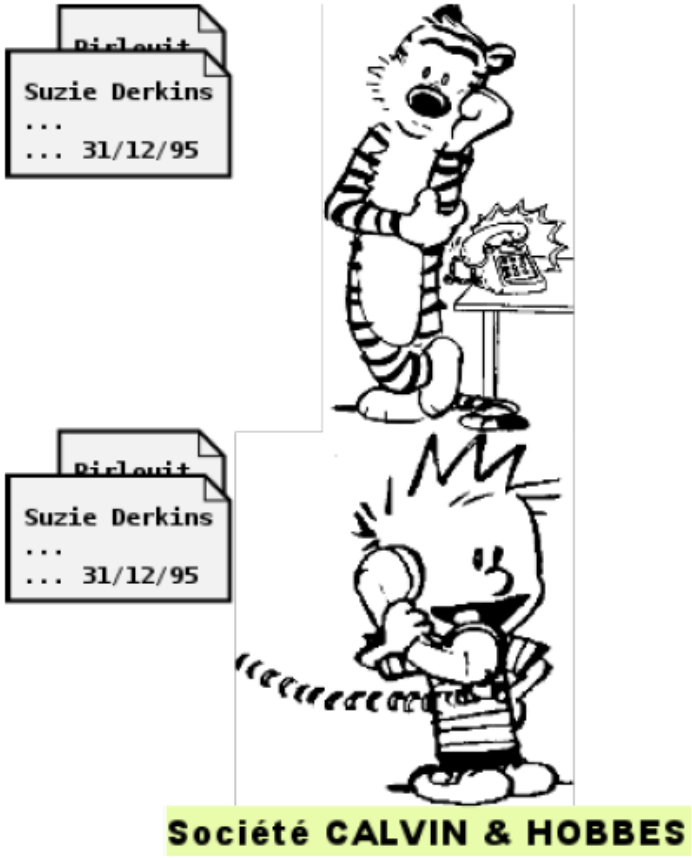


Théorème CAP

Illustration

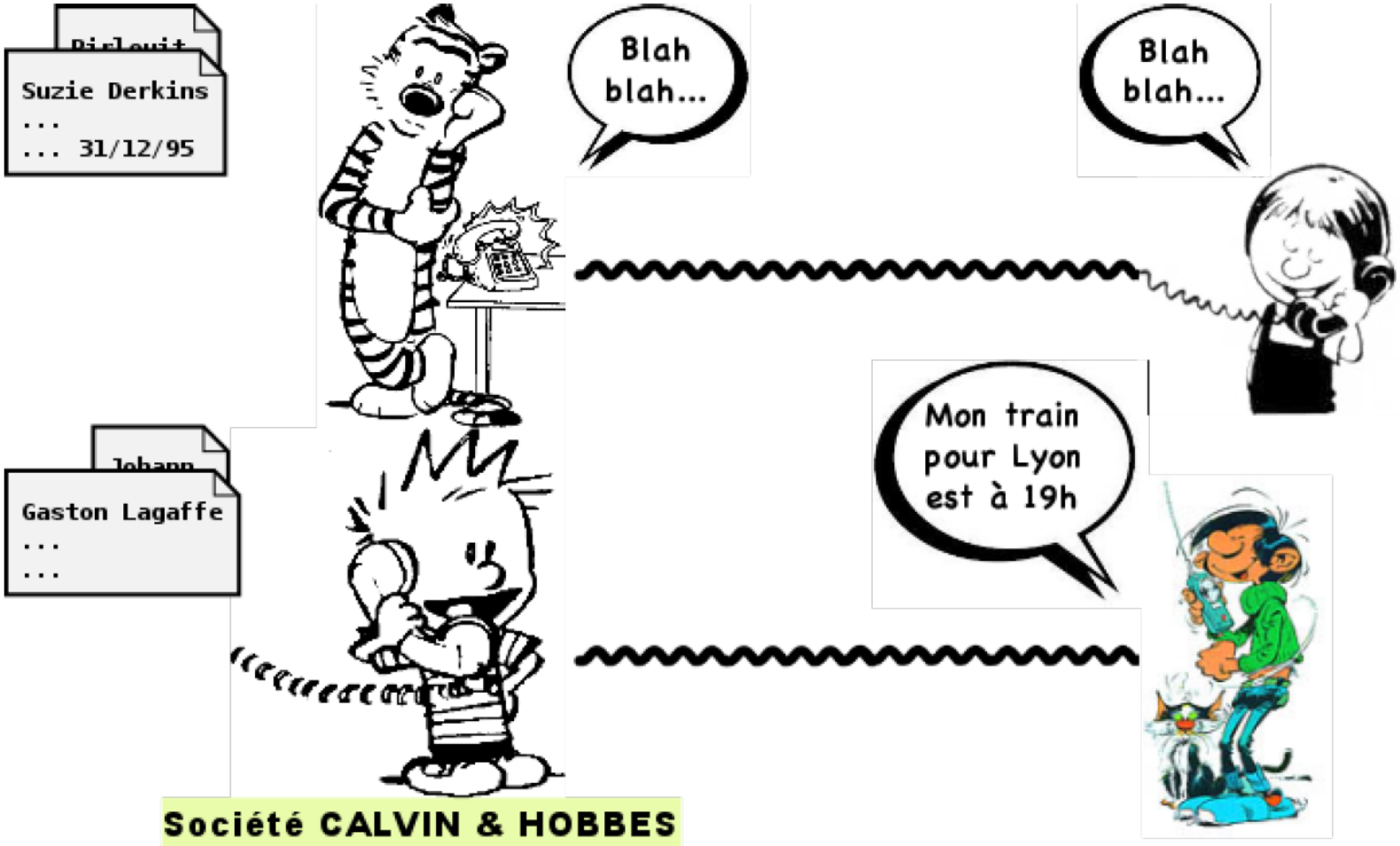
- Face au succès, Calvin est rapidement débordé d'appels
  - ☑ Cohérence
  - ☑ Résistance au morcellement
  - ☐ ~~Disponibilité~~

- Solution à la non disponibilité
  - ➡ faire équipe avec Hobbes



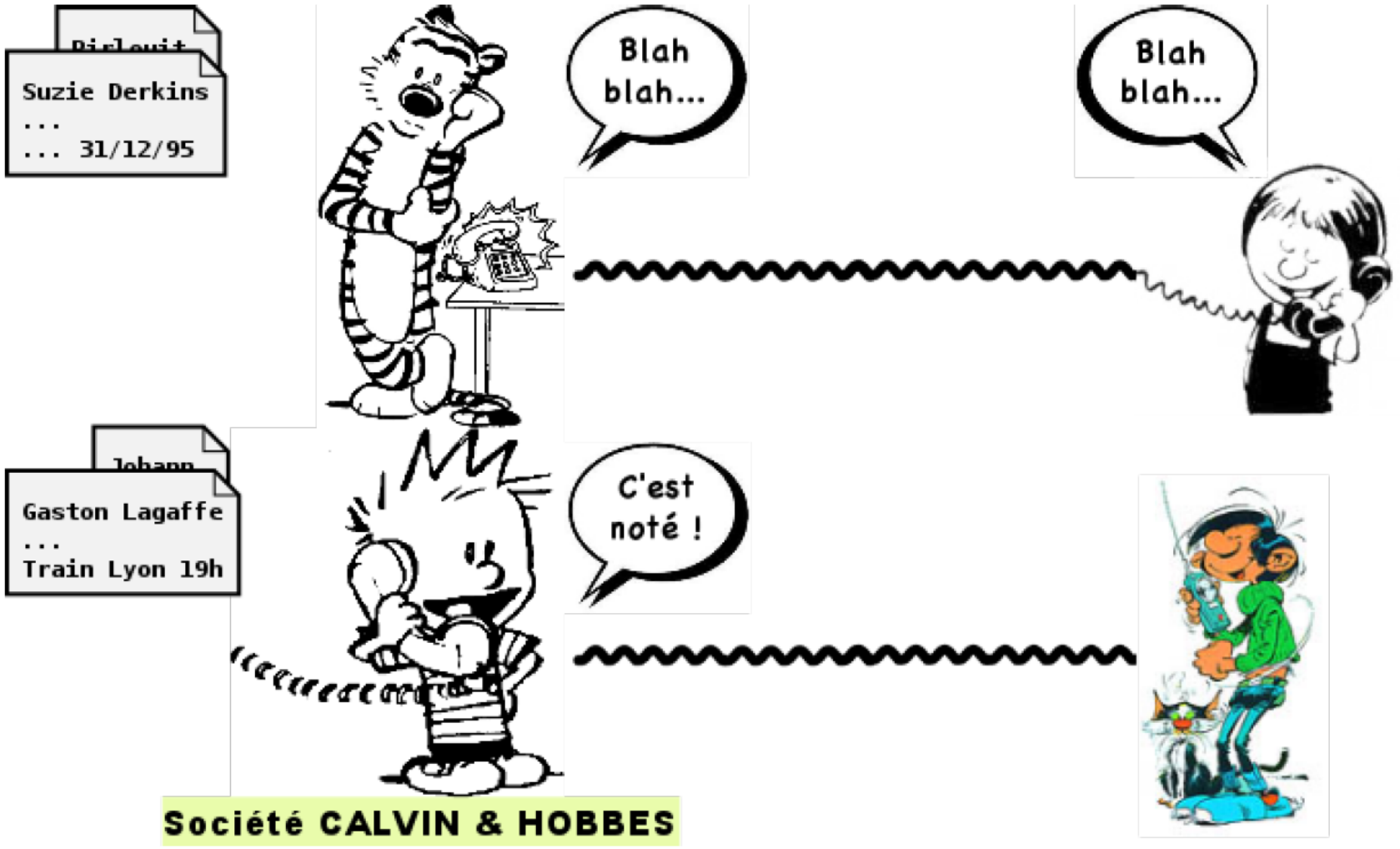
Théorème CAP

Illustration



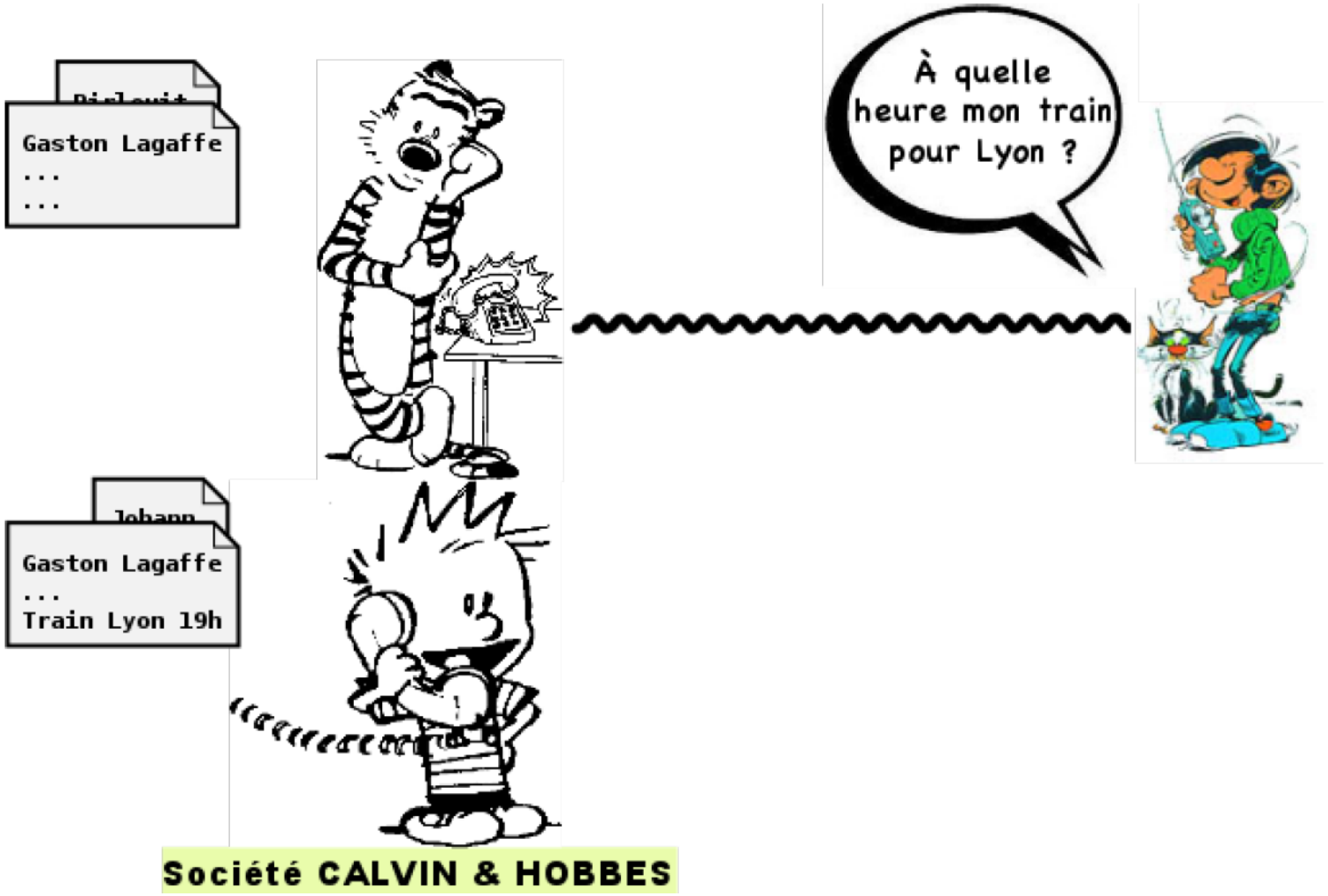
Théorème CAP

Illustration



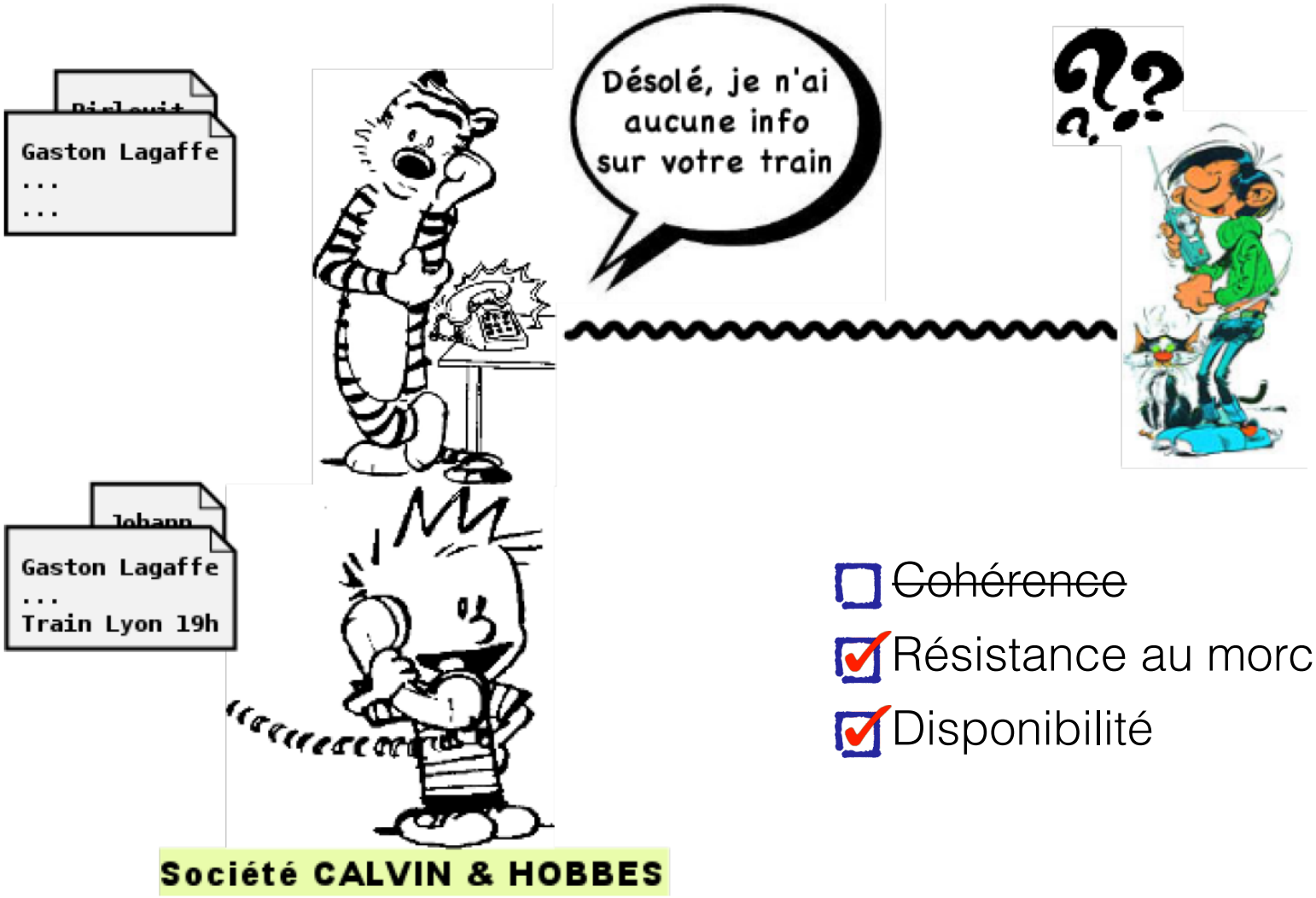
Théorème CAP

Illustration



Théorème CAP

Illustration



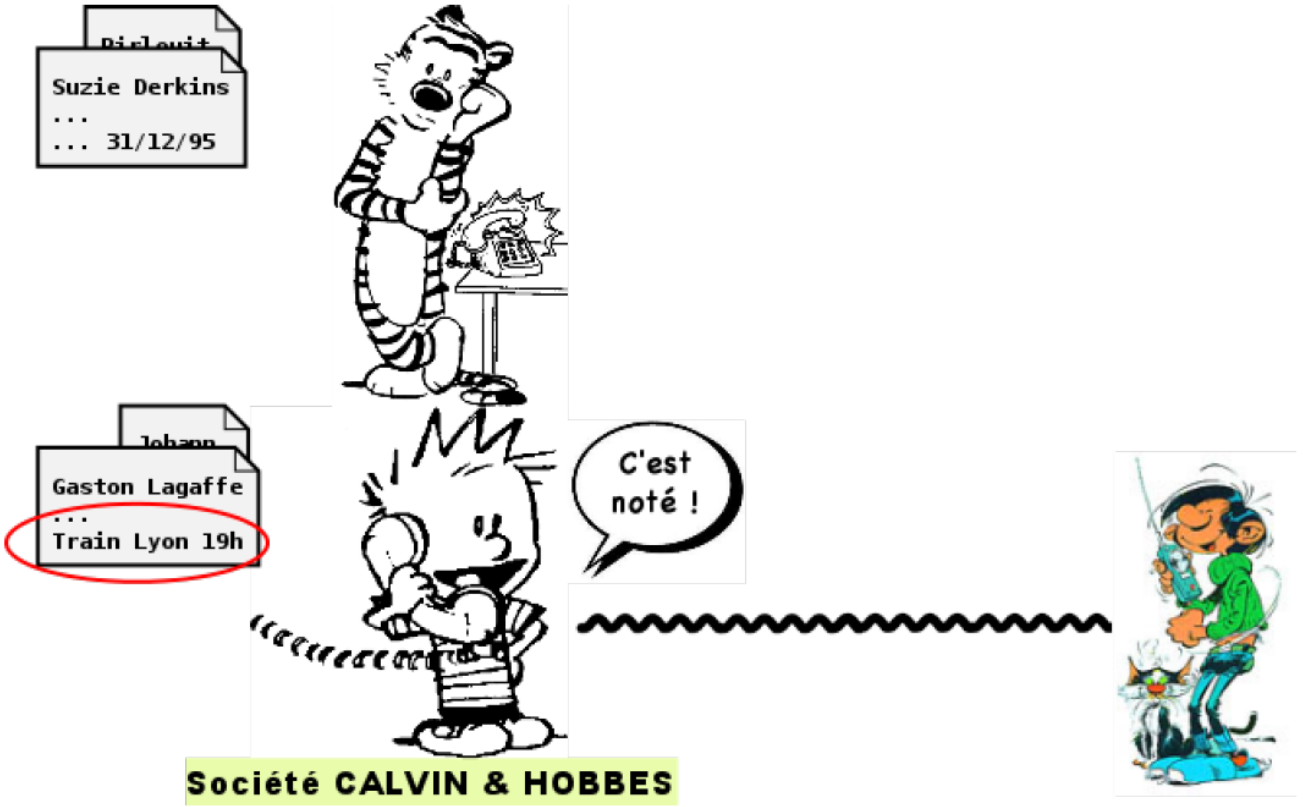
- ☐ ~~Cohérence~~
- ☑ Résistance au morcellement
- ☑ Disponibilité

Théorème CAP

Illustration

- Solution à l'incohérence :

  ➡ mise à jour des fiches entre Calvin et Hobbes



Théorème CAP

Illustration

- Solution à l'incohérence

➡ mise à jour des fiches entre Calvin et Hobbes

De nouveau un problème d'indisponibilité !

Théorème CAP

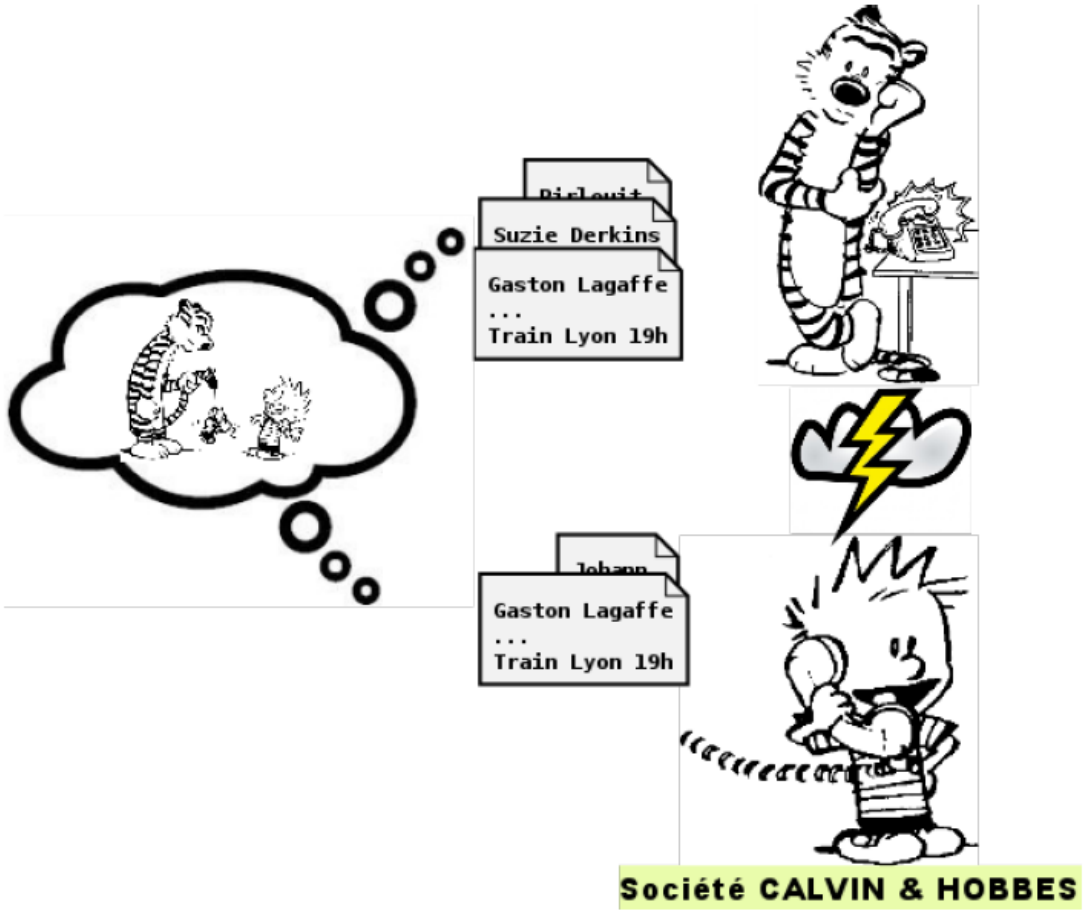
Illustration

- Et lorsque Calvin et Hobbes ne communiquent plus ?

Théorème CAP

Illustration

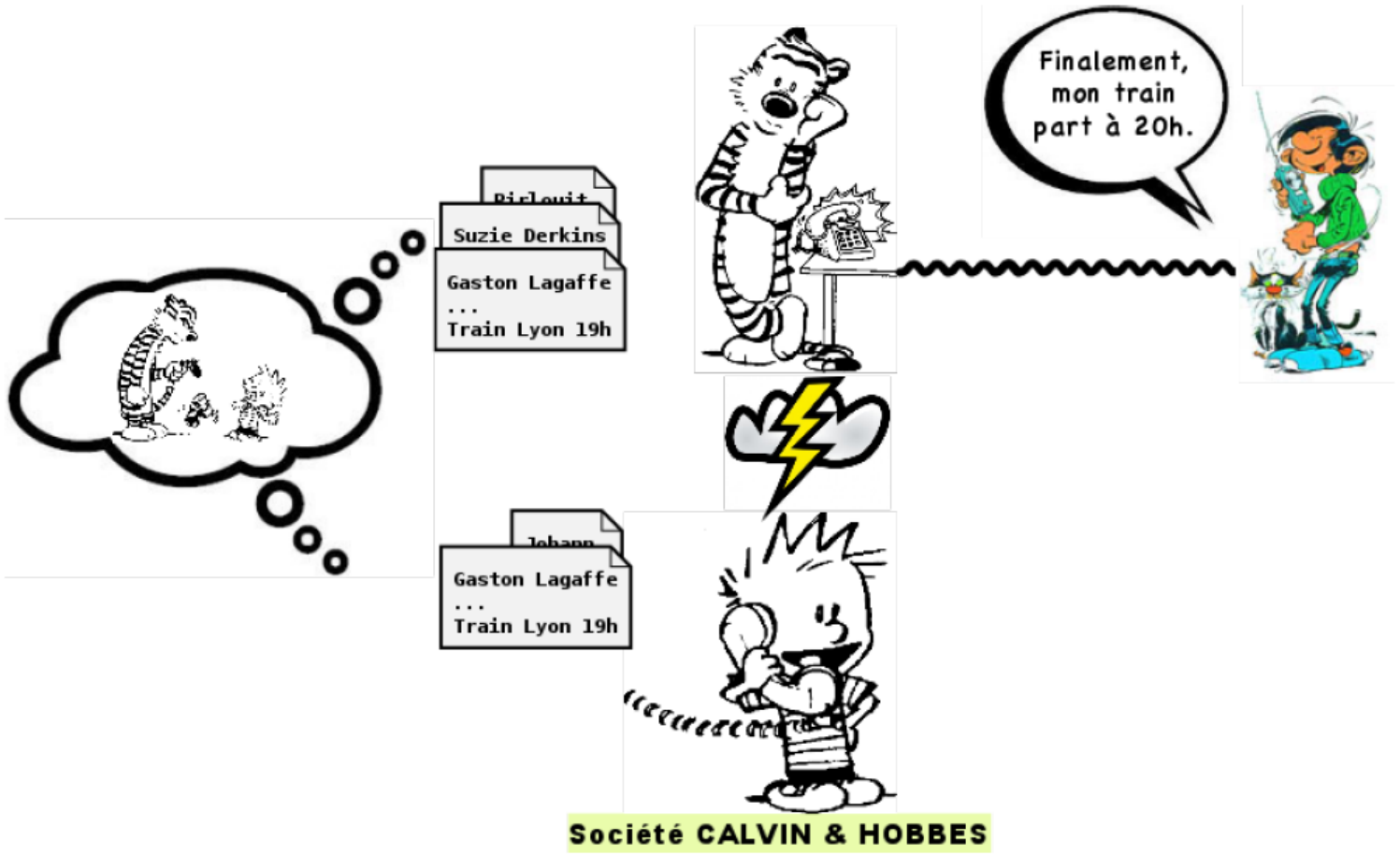
- Et lorsque Calvin et Hobbes ne communiquent plus ?



Théorème CAP

Illustration

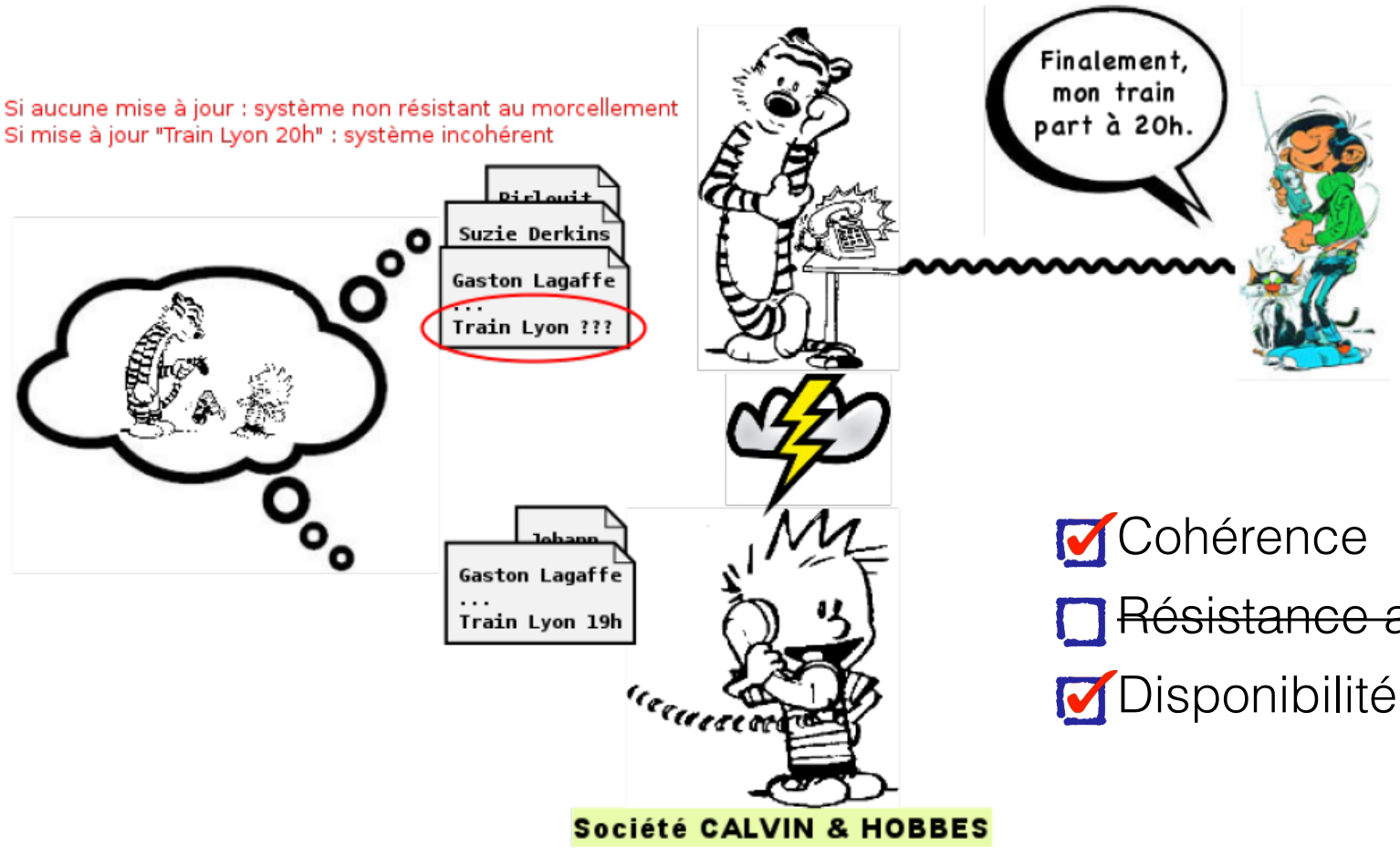
- Et lorsque Calvin et Hobbes ne communiquent plus ?



Théorème CAP

Illustration

- Et lorsque Calvin et Hobbes ne communiquent plus ?



☑ Cohérence

☐ ~~Résistance au morcellement~~

☑ Disponibilité

Théorème CAP

Illustration

- Solution possible : « réconcilier » le système et éventuelle mise à jour des fiches

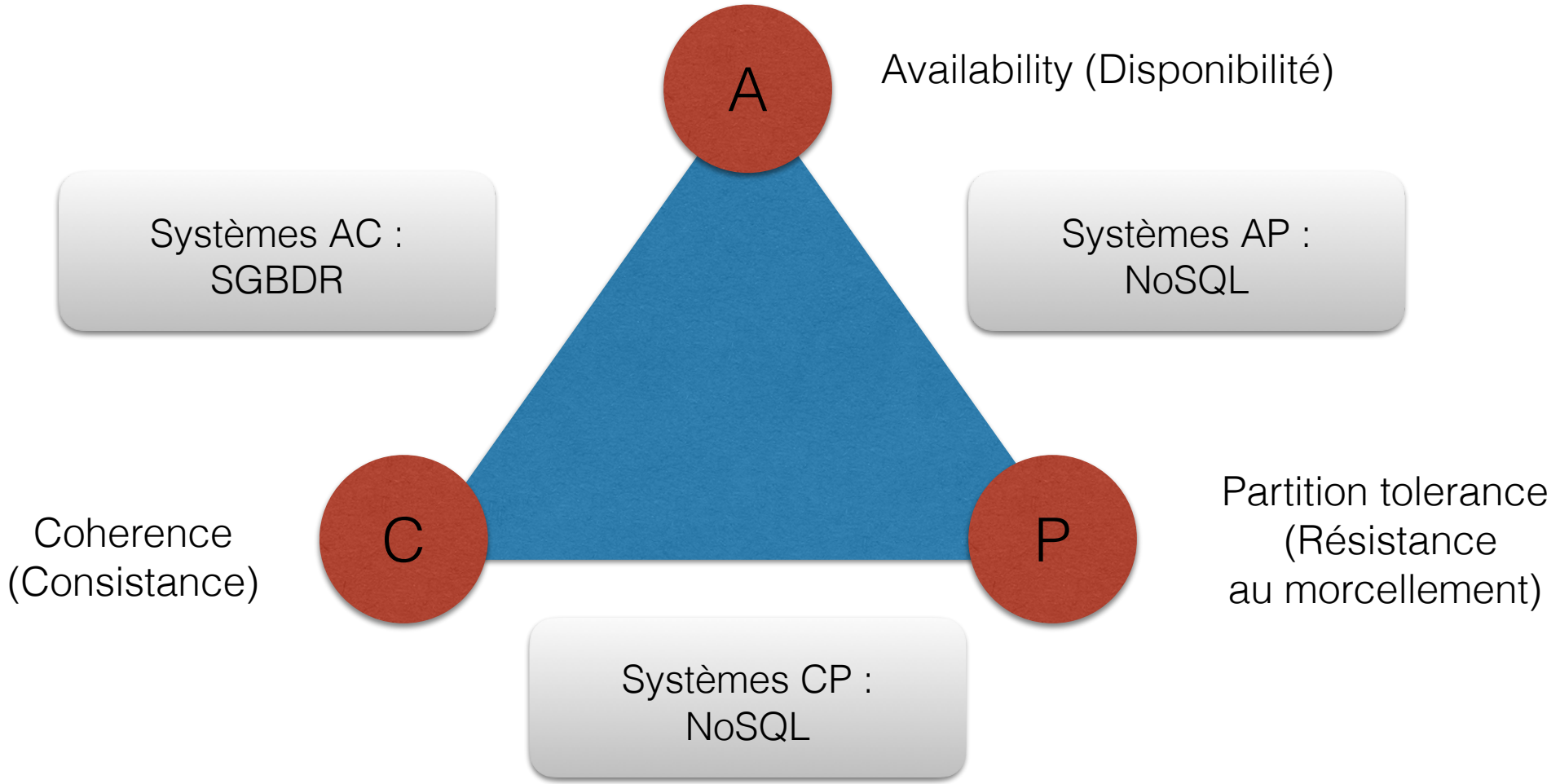
Théorème CAP

Illustration

- Solution possible : « réconcilier » le système et éventuelle mise à jour des fiches

De nouveau un problème d'indisponibilité !

Théorème de CAP



Choix techniques du NoSQL

- Quels protocoles d'accès aux données ?
- Quels modèles de distribution ?
  - Comment gérer la cohérence ?
- Le Big Data analytique

Protocoles d'accès aux données

- Les moteurs NoSQL implémentent différentes méthodes d'accès aux données
  - Des bibliothèques existent pour tous les grands langages clients afin de les rendre plus faciles d'utilisation

Protocoles d'accès aux données

Exemple d'interfaces client-serveurs

- Protocole de type RPC (Remote Procedure Call) liant les clients et un serveur via des interfaces bien définies maintenant généralement une connexion TCP
  - Interfaces natives
    - ex : MongoDB
  - Protocol Buffers (protobuf) proposé par Google
    - ex : Riak, HBase
  - Thrift
  - …
- REST (Representational State Transfer)
  - ex : CouchDB

Modèles de distribution

- En fonction du modèle de distribution :
  - possibilité de traiter un volume de données de plus en plus grand
  - possibilité de gérer plus de lectures/écritures
  - possibilité de faire face à des ralentissements réseau ou à des pannes
- Deux notions fondamentales pour la mise en place du modèle de distribution :
  - **Réplication**
    - copie des données sur plusieurs noeuds
      - maître-esclave ou peer-to-peer
  - **Sharding**
    - répartition des données sur différents noeuds

Modèles de distribution

Un modèle de distribution peut être basé sur du sharding, ou de la réplication, ou les 2 à la fois

- Complexité croissante de distribution :
  - serveur seul
  - sharding peer-to-peer
  - réplication maître-esclave
  - réplication peer-to-peer
  - combinaison du sharding et de la réplication

Sharding

(Shard : éclat, tesson)

Sharding : éclatement des données en partitions distribuées sur plusieurs machines, de façon aussi automatique que possible par le moteur NoSQL

- Caractéristiques particulières
  - l'éclatement est géré automatiquement par le système
  - l'éclatement est par nature distribué : cela n'a pas de sens de créer plusieurs shards sur la même machine
  - la répartition de charge est possible

Sharding

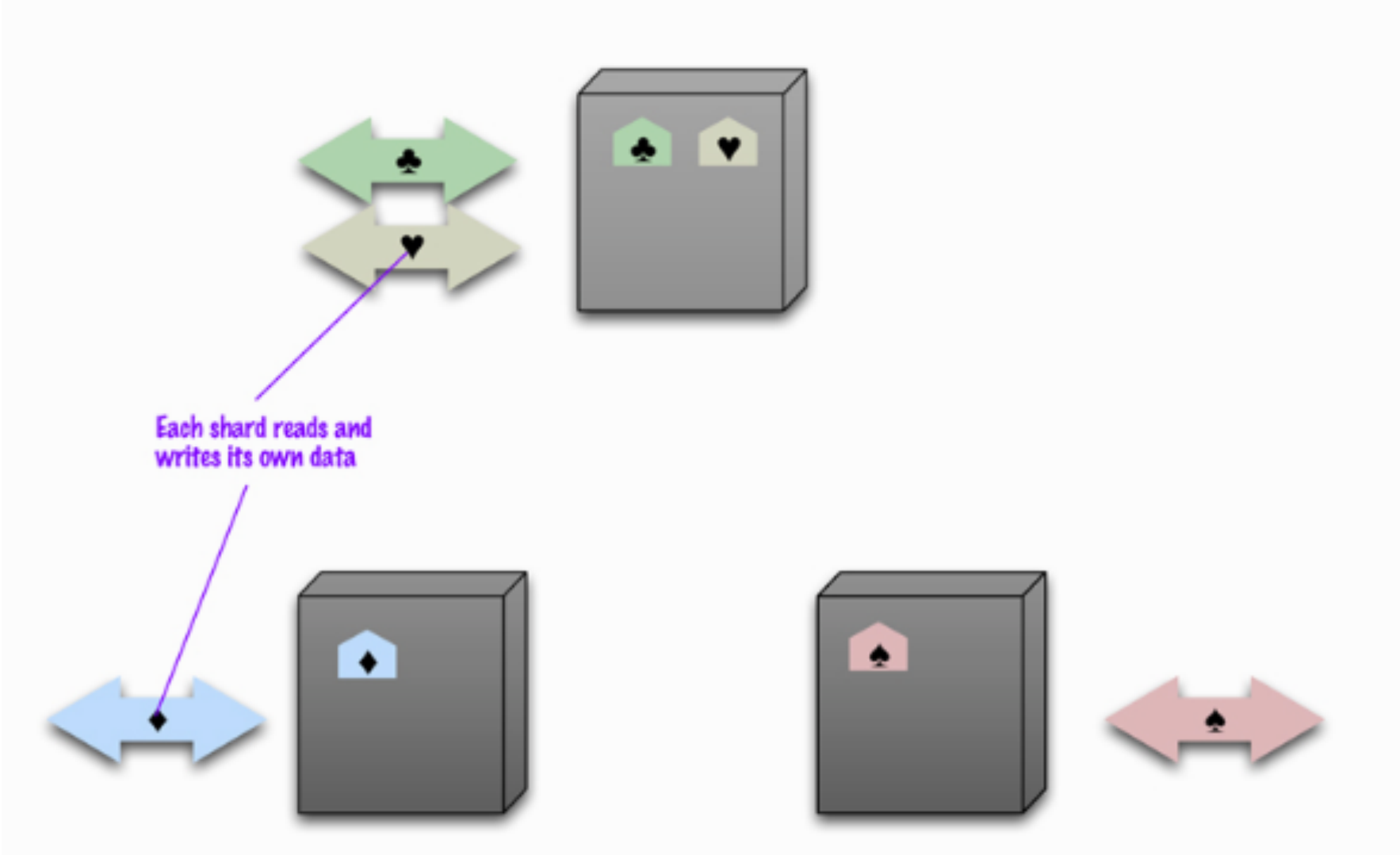
(Shard : éclat, tesson)

- Mécanisme de **partitionnement horizontal** (par N-uplets)
  - Les objets-données sont stockés sur des noeuds serveurs différents en fonction d'une clé (par exemple une fonction de hachage)

partition = hash(o) mod n
avec o= objet-données à placer
n= nombre de noeuds

  - Les objets-données peuvent aussi être partitionnés afin que :
    - les objets-données accédés ou mis à jour en même temps résident sur le même noeud
    - la charge soit uniformément répartie entre les noeuds
      - pour cela des objets-données peuvent être répliqués
- Certains systèmes utilisent aussi un **partitionnement vertical** (par colonnes) dans lequel des parties d'un enregistrement sont stockées sur différents serveurs

Sharding peer-to-peer

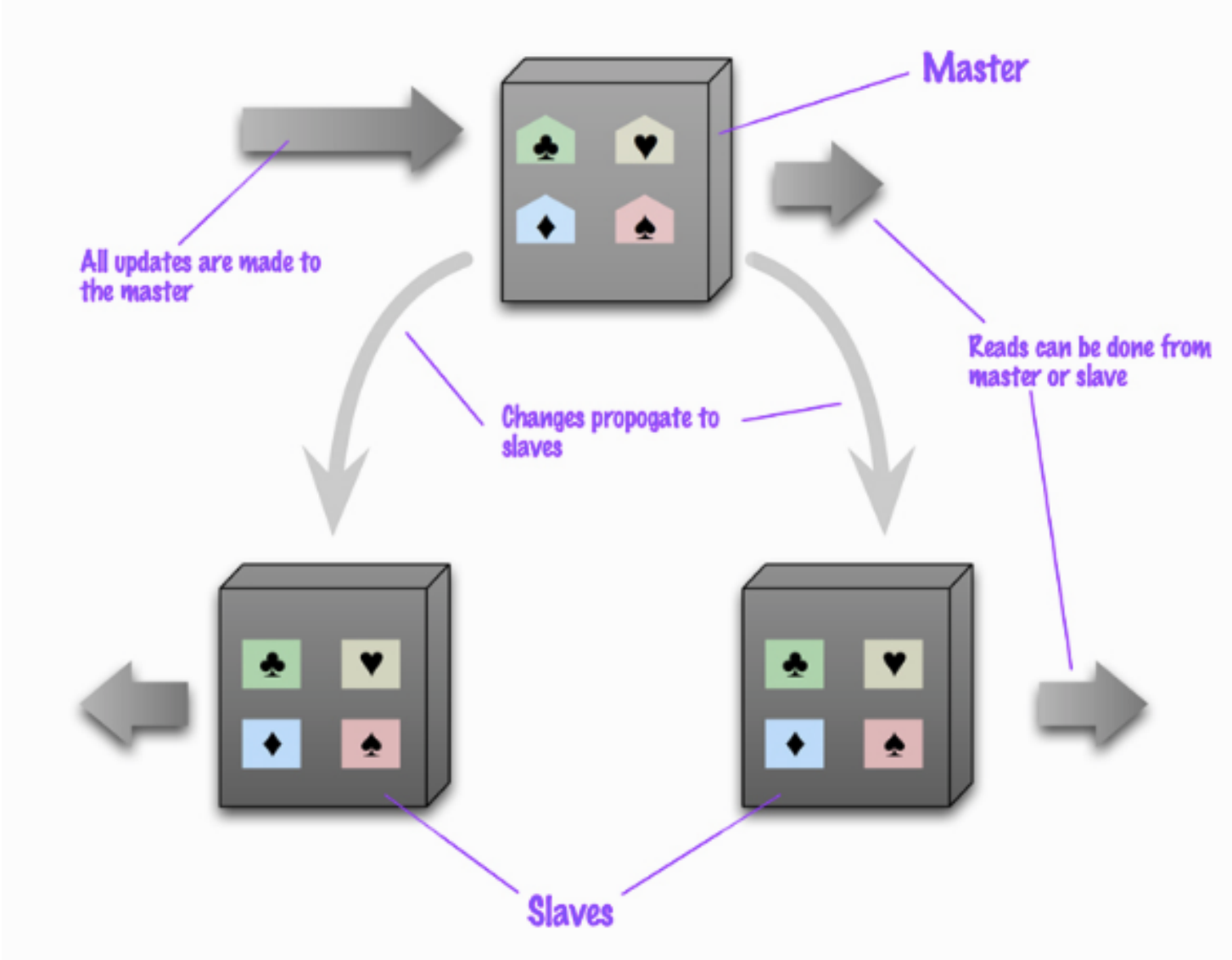


Source: Pramod J. Sadalage

Réplication maître-esclave

- La machine maître conserve la configuration du système, reçoit les demandes de mise à jour et les propage vers les machines esclaves
  - les machines esclaves sont accessibles en lecture seule
- Avantages
  - Intéressant en cas de système demandant beaucoup d'accès en lecture sur les données
  - Résilience en lecture : si le master tombe, les esclaves peuvent toujours fournir les données
- Inconvénients
  - Présence d'un SPOF (Single Point of Failure) en cas de mise à jour
    - MAIS : possibilité de transformer un esclave en maître très rapidement pour la restauration
  - Incohérences possibles si accès en lecture sur différents esclaves pour lesquels le niveau de mise à jour n'est pas le même

Réplication maître-esclave



Source: Pramod J. Sadalage

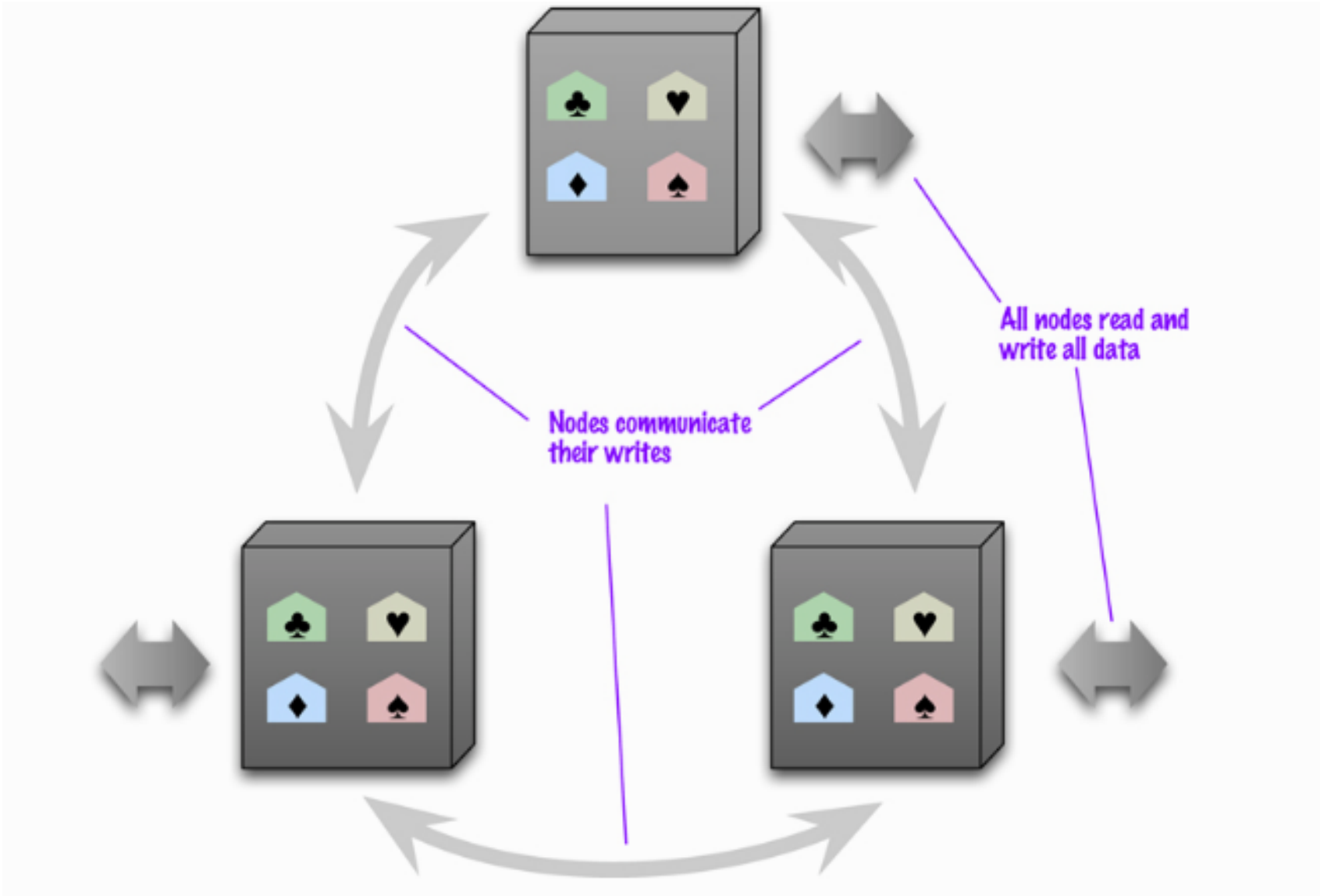
Réplication peer-to-peer

- Tous les noeuds ont le même poids, acceptent les écritures, et la perte de l'un d'entre eux ne remet pas en cause l'accès aux données

- Il faut mettre en place un protocole de bavardage (*gossip protocol*) entre les noeuds

- Inconvénient : Consistence (encore!)
    - 2 personnes peuvent mettre à jour le même N-uplet en même temps sur 2 noeuds différents…

Réplication peer-to-peer



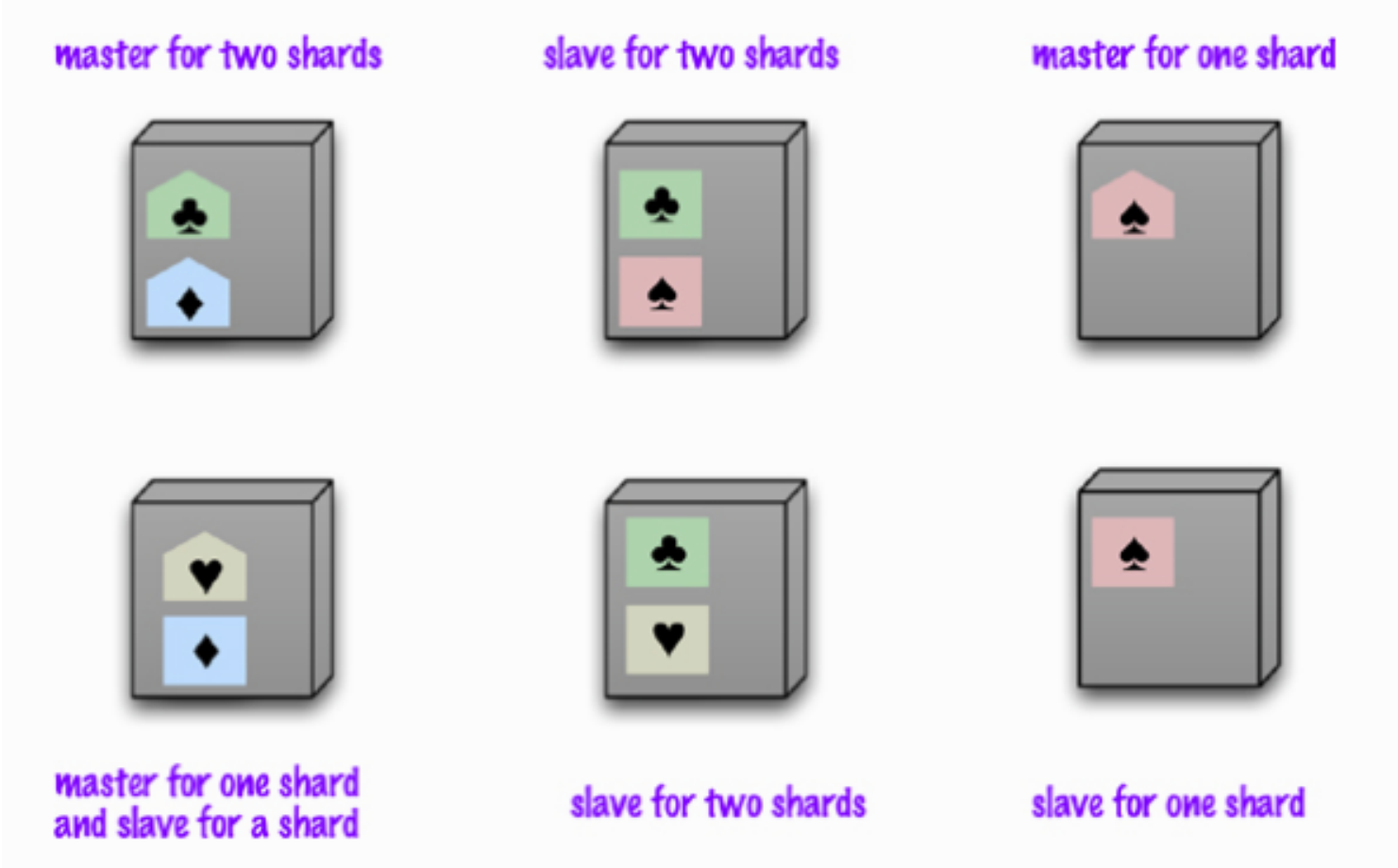
Source: Pramod J. Sadalage

Combinaison du sharding et de la réplication

- Réplication maître esclave et sharding
  - Plusieurs maîtres, mais chaque objet-donnée n'est que sur un seul maître
  - Un noeud peut être maître pour certaines données et esclave pour d'autres
  - On peut aussi choisir de faire des maîtres et esclaves « exclusifs »

Combinaison du sharding et de la réplication

- Réplication maître esclave et sharding

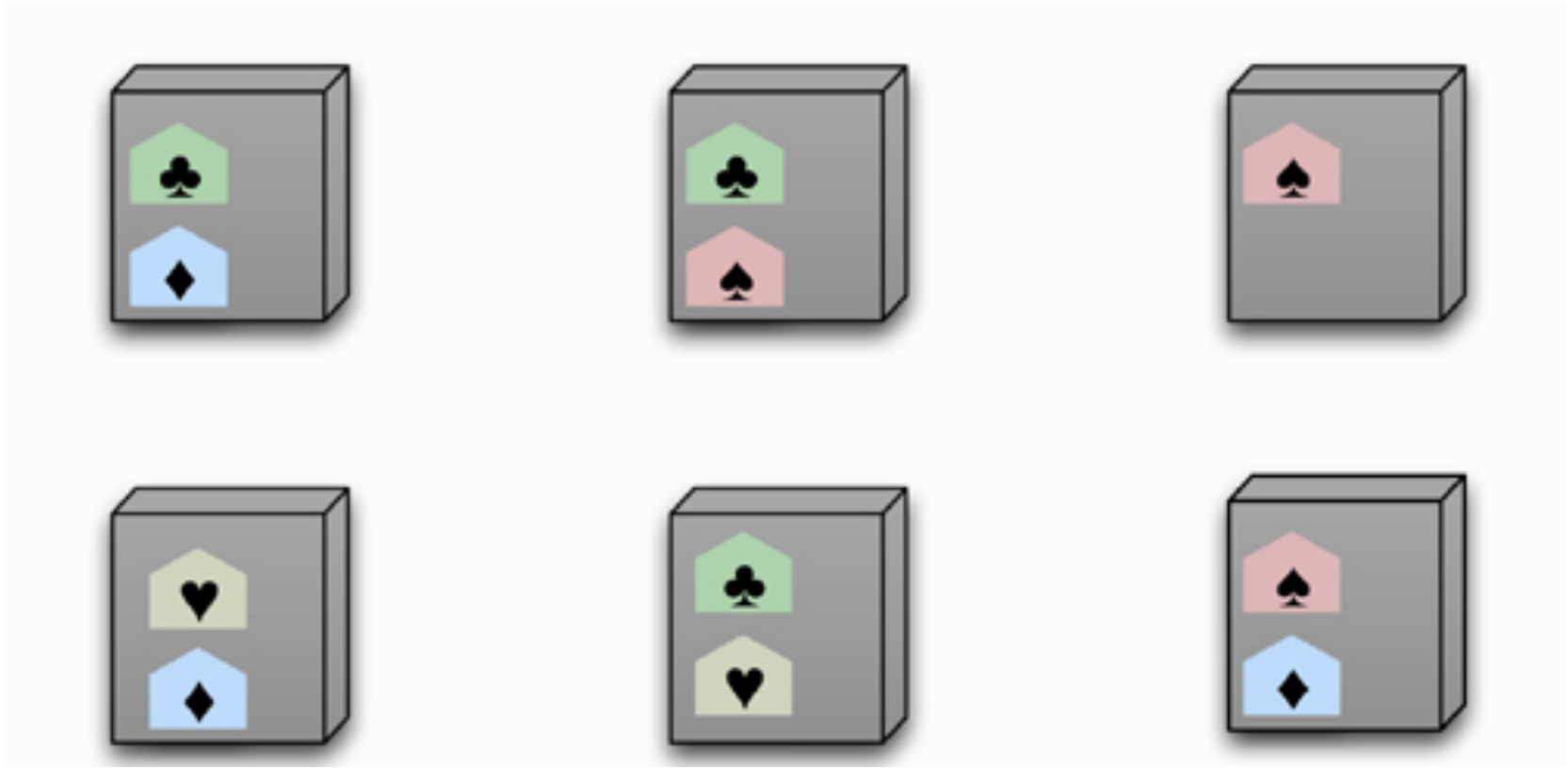


Source: Pramod J. Sadalage

Combinaison du sharding et de la réplication

- Réplication peer-to-peer et sharding
  - Si un noeud tombe, les shards sur ce noeud seront déplacés sur les autres noeuds

Combinaison du sharding et de la réplication



Source: Pramod J. Sadalage

En résumé

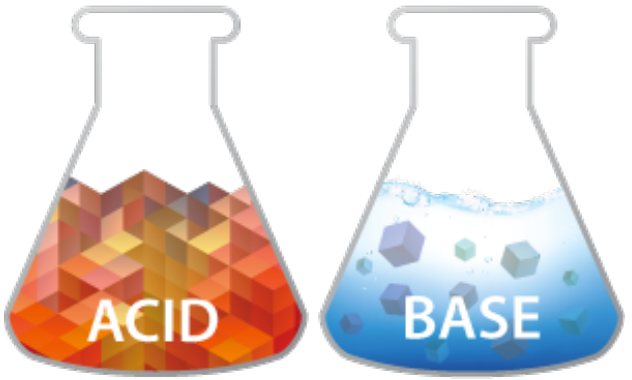
- Deux façons de distribuer les données
  - le sharding distribue les données sur des noeuds différents, et chaque noeud est la seule source pour un ensemble d'objets-données
  - la réplication copie les données sur plusieurs noeuds, chaque objet-données pouvant se trouver à plusieurs endroits

  => un système peut utiliser l'une des techniques ou les 2
- la réplication se fait de 2 façons
  - réplication maître-esclave : le maître gère les écritures et la synchronisation avec les esclaves, et les esclaves ne sont accessibles qu'en lecture
  - réplication peer-to-peer : l'écriture et la lecture se font sur tous les noeuds, qui se synchronisent entre eux

  => la réplication maître-esclave réduit les conflits de mise à jour mais la réplication peer-to-peer évite d'avoir un SPOF en écriture

ACID vs BASE

- **BASE** : (Basically Available, Soft state, Eventual consistency)
- Basically Available : toute requête a une réponse, mais pas forcément cohérente
- Soft state : l'état du système change en quasi-permanence ; même lorsqu'il n'y a pas d'écriture, les données peuvent être propagées et donc l'état du système peut changer
- Eventual consistency (cohérence finale) : les données vont être propagées là où elles le doivent à un moment ou à une autre, mais le système continue à recevoir des entrées sans vérifier la cohérence des transactions précédentes



Comment gérer la cohérence finale ?

Gestion de la cohérence des données

- Les conflits en écriture arrivent quand 2 clients veulent écrire la même donnée en même temps
- Les conflits en lecture-écriture arrivent quand un client lit des données inconsistantes pendant l'écriture par un autre client

- Les approches pessimistes verrouillent les données pour prévenir des conflits
- Les approches optimistes essaient de prévenir les conflits

Gestion de la cohérence des données

- Les systèmes distribués sont sujets à des conflits en lecture-écriture car certains noeuds peuvent avoir reçu des mises à jour et d'autres non
  - La **cohérence finale** est faite quand les écritures ont été propagées vers tous les noeuds

- Une nécessité est souvent la cohérence lecture-écriture propre (« read your writes »), c'est-à-dire qu'un client écrit et veut voir directement les nouvelles valeurs
  - Peut être difficile à mettre en place quand l'écriture et la lecture se font sur des noeuds différents

Cohérence finale

(ou cohérence sur le long terme)

- Trois techniques pour atteindre la cohérence finale
  - **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
  - **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)

Cohérence finale

Exemple

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 de deux paniers en conflit permet de ne pas perdre d'ajout de produits.

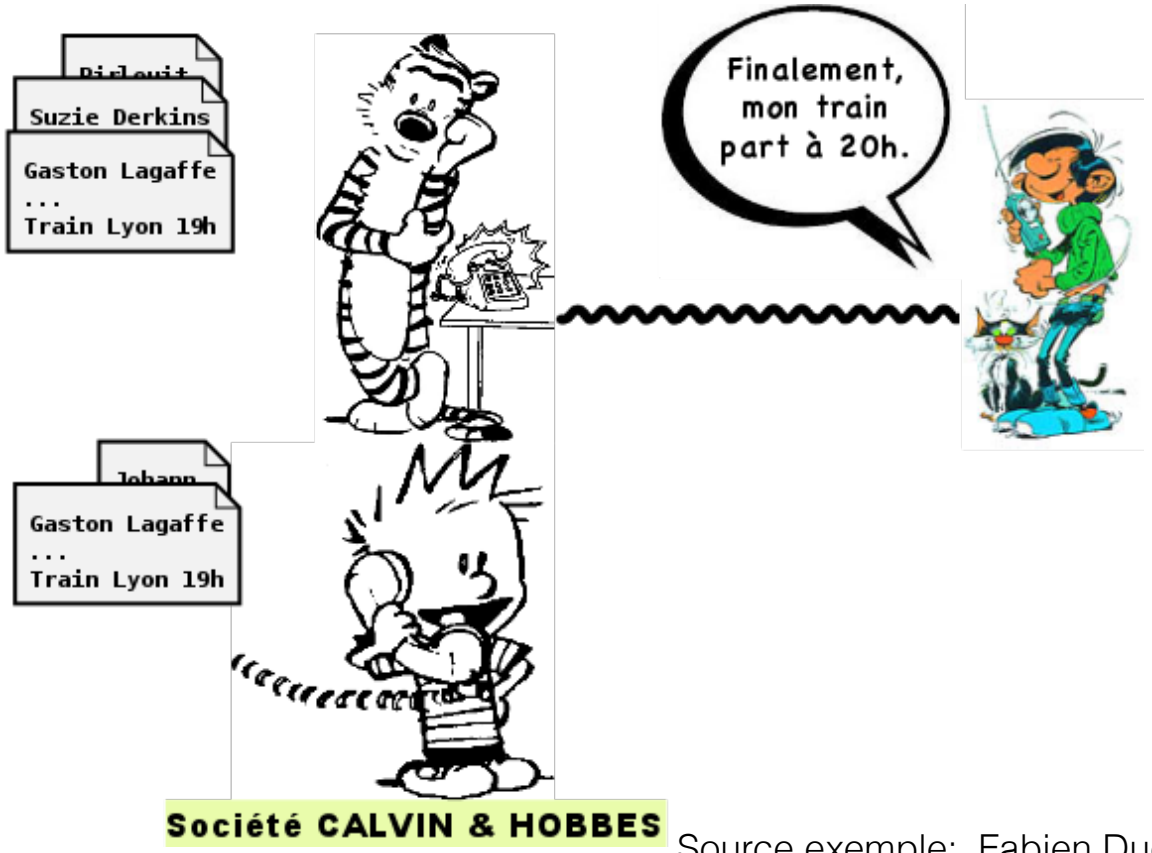
Utilisation des horloges vectorielles (vector-clocks) pour limiter l'incohérence des paniers au niveau des suppressions de produits.

http://the-paper-trail.org/blog/2008/08/26/

Cohérence finale

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

Gestionnaires/clercs



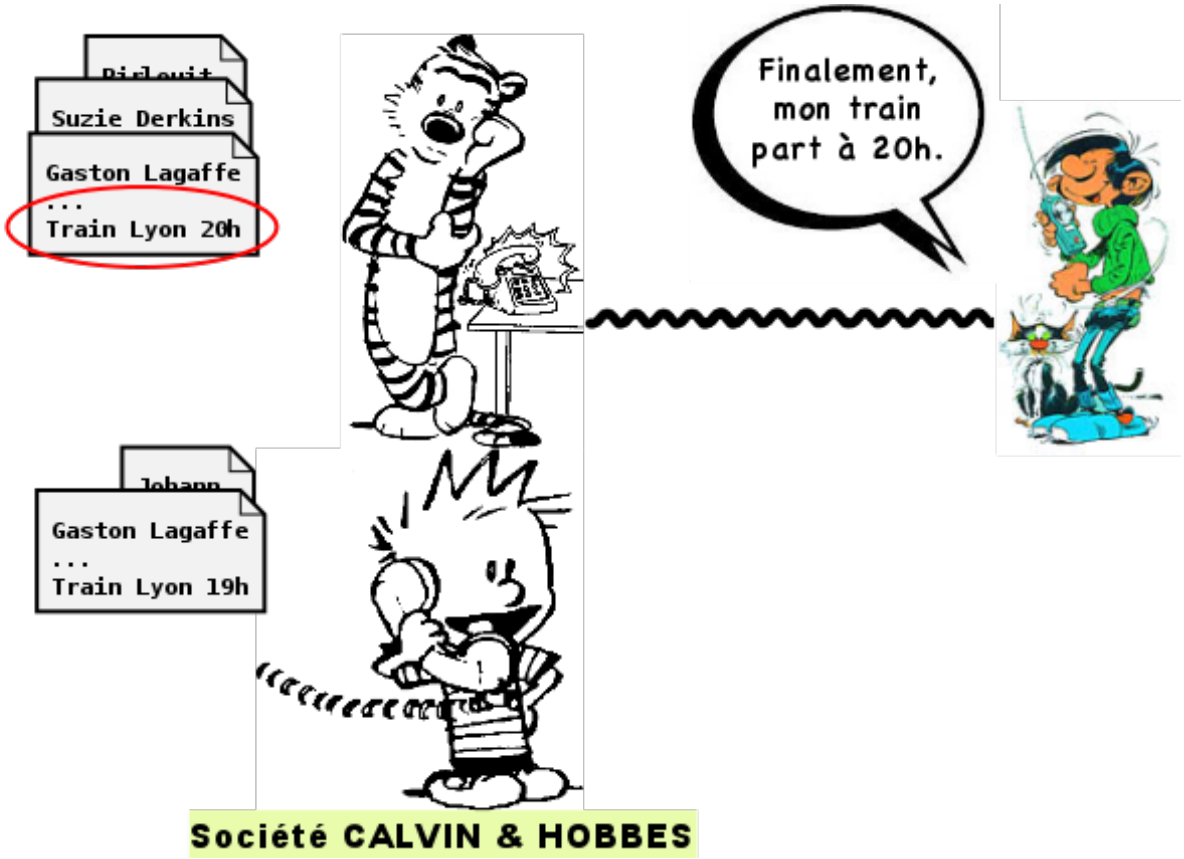
Société CALVIN & HOBBES

Source exemple: Fabien Duchateau

Cohérence finale

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

Gestionnaires/clercs



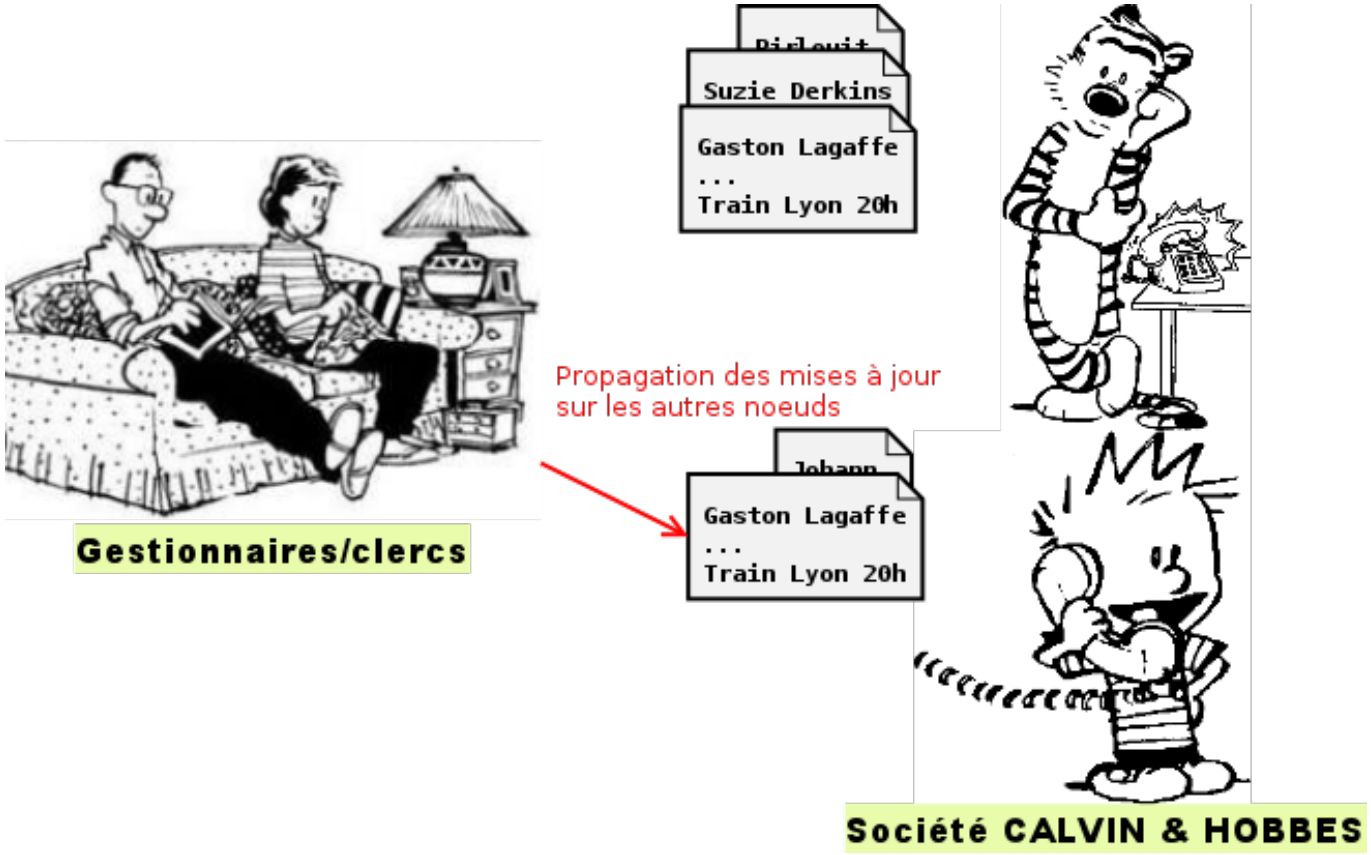
Société CALVIN & HOBBES

Cohérence finale

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

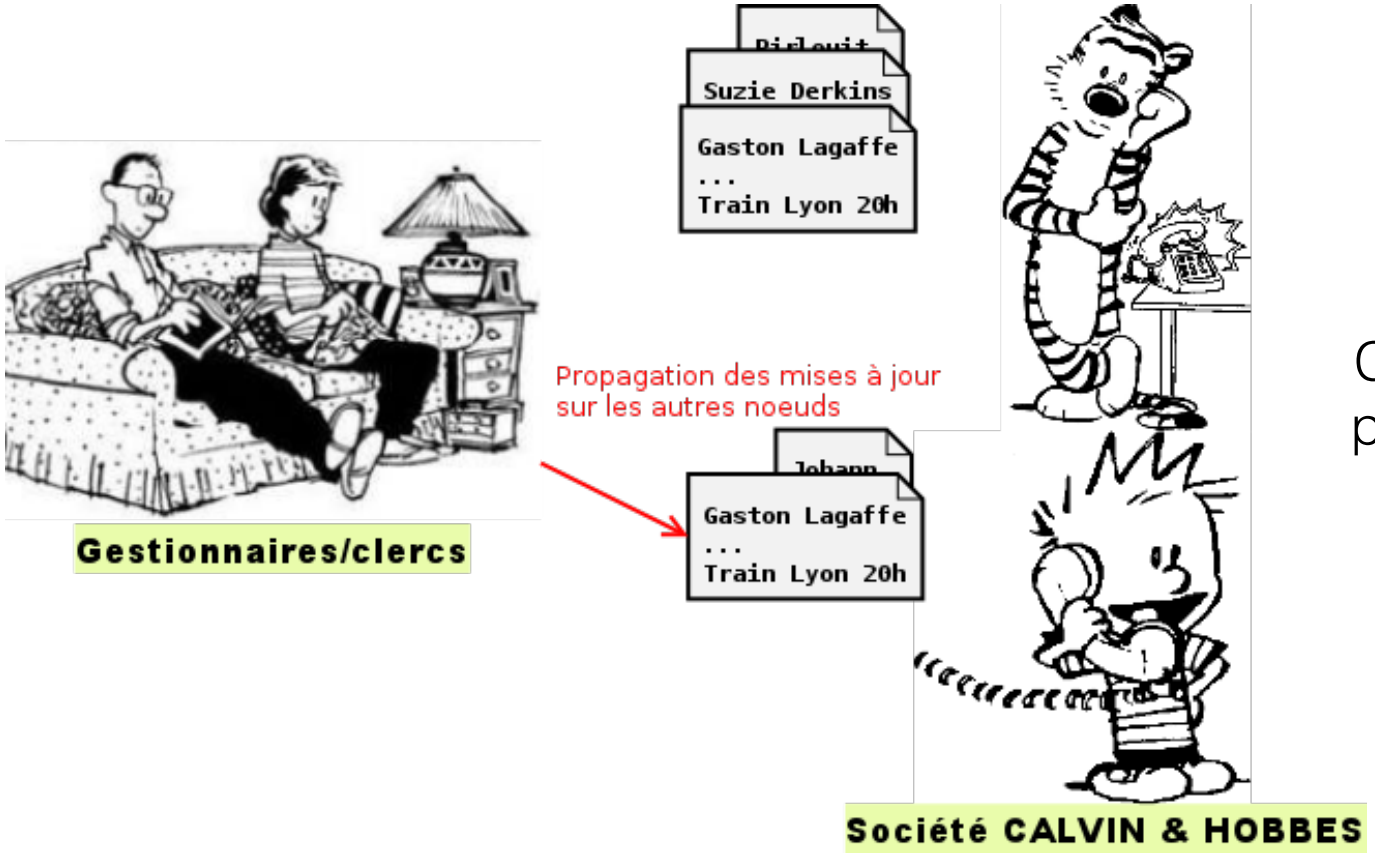
Cohérence finale

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



Cohérence finale

- 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

Gestion de la cohérence des données

- Pour avoir une bonne cohérence, il faut impliquer beaucoup de noeuds à cette tâche, ce qui augmente la latence

  => il faut équilibrer entre la cohérence et la latence

Le Big Data Analytique

- On peut classer les orientations Big Data en 2 catégories :
  - **BD opérationnelles**
    - correspond à l'OLTP en relationnel
    - BD constamment modifiée pour soutenir une activité en perpétuel changement
    - Montée en charge sur des PetaOctets
    - Exemples : Cassandra, Hbase
  - **BD analytiques**
    - correspond à l'OLAP en relationnel
    - Traitement massif des données et agrégations
    - Approche principale : Map-Reduce (cf. Google !)

Map-Reduce

Présentation

- Modèle de programmation parallèle pour le traitement de grands ensembles de données
- Développé par Google pour le traitement de gros volumes de données en environnement distribué
  - permet de répartir la charge sur un grand nombre de serveurs (cluster)
  - abstraction quasi-totale de l'infrastructure matérielle
    - gère entièrement, de façon transparente le cluster, la distribution de données, la répartition de la charge, et la tolérance aux pannes
  - ajouter des machines augmente la performance
- La librairie Map-Reduce existe dans plusieurs langages (C++, C#, Java, Python, Ruby…)

Map-Reduce

Usage

- Utilisé par les grands acteurs du Web notamment pour :
  - Construction des index (Google Search)
  - Détection de spams (Yahoo)
  - Data Mining (FaceBook)
- Mais aussi pour :
  - de l'analyse d'images astronomiques, de la bioinformatique, de la simulation météorologique, …
  - trouver le nombre d'occurrences d'un pattern dans un très gros volume de données
  - classifier de très grands volumes de données provenant par exemple de paniers d'achats de clients (Data Mining)

Map-Reduce

Gestion de données

- Principales opérations à faire sur de grands ensembles de données :
  1. itérer sur un grand nombre d'enregistrements
  2. extraire quelque chose ayant un intérêt, de chacun de ses enregistrements
  3. regrouper et trier les résultats intermédiaires
  4. agréger tous ces résultats ensemble
  5. générer le résultat final
- Dans le modèle de programmation Map-Reduce, le développeur implémente 2 fonctions : la fonction **Map** et la fonction **Reduce**
  - opérations 1 et 2 : la fonction Map traite une paire clé/valeur et génère un ensemble de paires de clés intermédiaires/valeurs
  - opérations 3, 4 et 5: la fonction Reduce fusionne toutes les valeurs intermédiaires associées avec la même clé intermédiaire

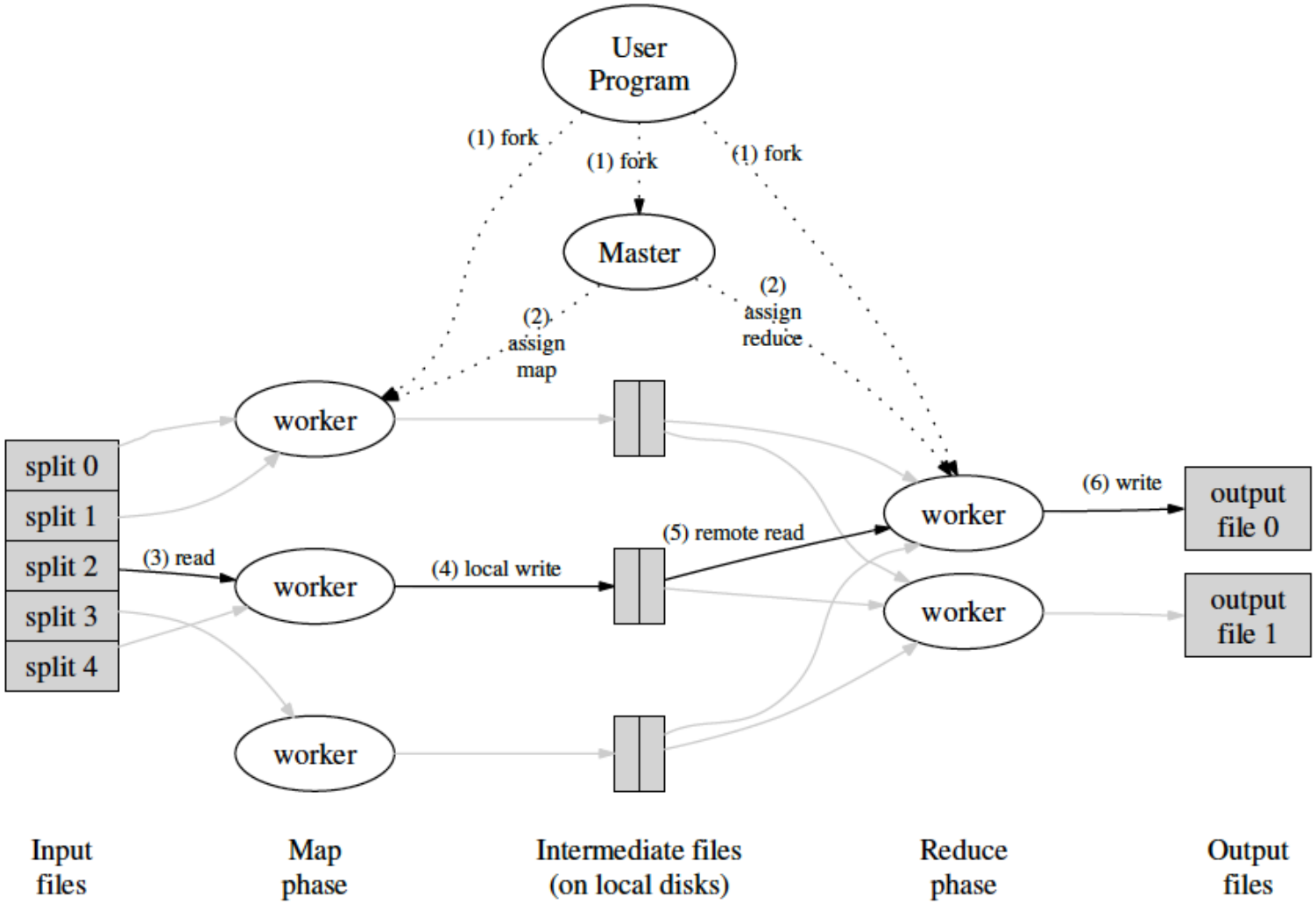
Map-Reduce

Fonctionnement

- Lorsque l'application MapRedue est lancée, elle crée un composant **Master** responsable de la distribution des données et de la coordination de différentes unités de travail ou **Workers**
- Le **Master** attribue aux **Workers** les tâches ***Map*** et ***Reduce***
- Un Worker possède 3 états :
  - *idle* : est disponible pour un nouveau traitement
  - *in-progress* : un traitement est en cours d'exécution
  - *completed* : il a fini un traitement, il informe alors le Master de la taille et de la localisation de ses fichiers intermédiaires
- Le **Master** gère la synchronisation, la réorganisation, le tri et le regroupement des données
  - Lorsqu'un **Worker** de type ***Map*** a fini son traitement, le **master** regroupe, trie et renvoie le résultat à un **Worker** de type ***Reduce***

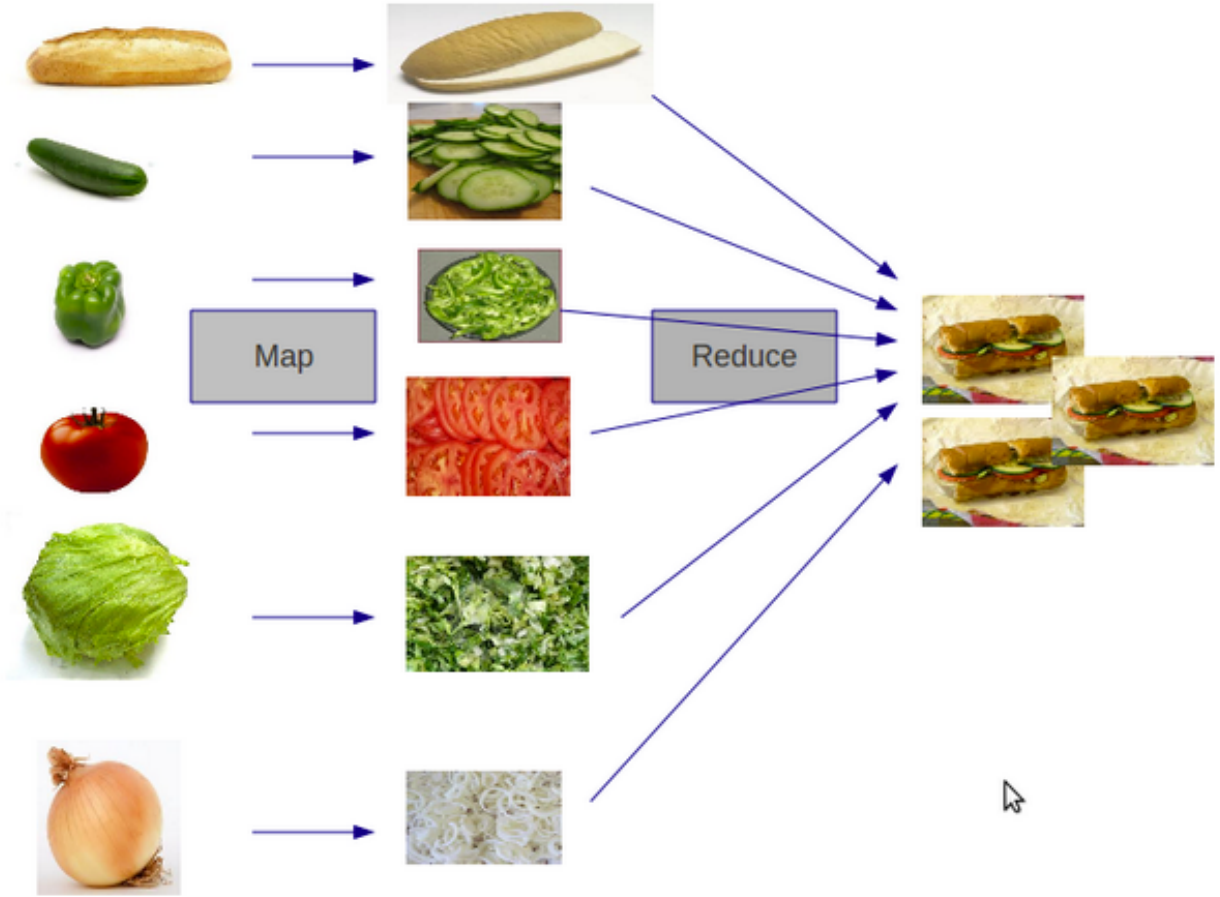
Map-Reduce

Fonctionnement



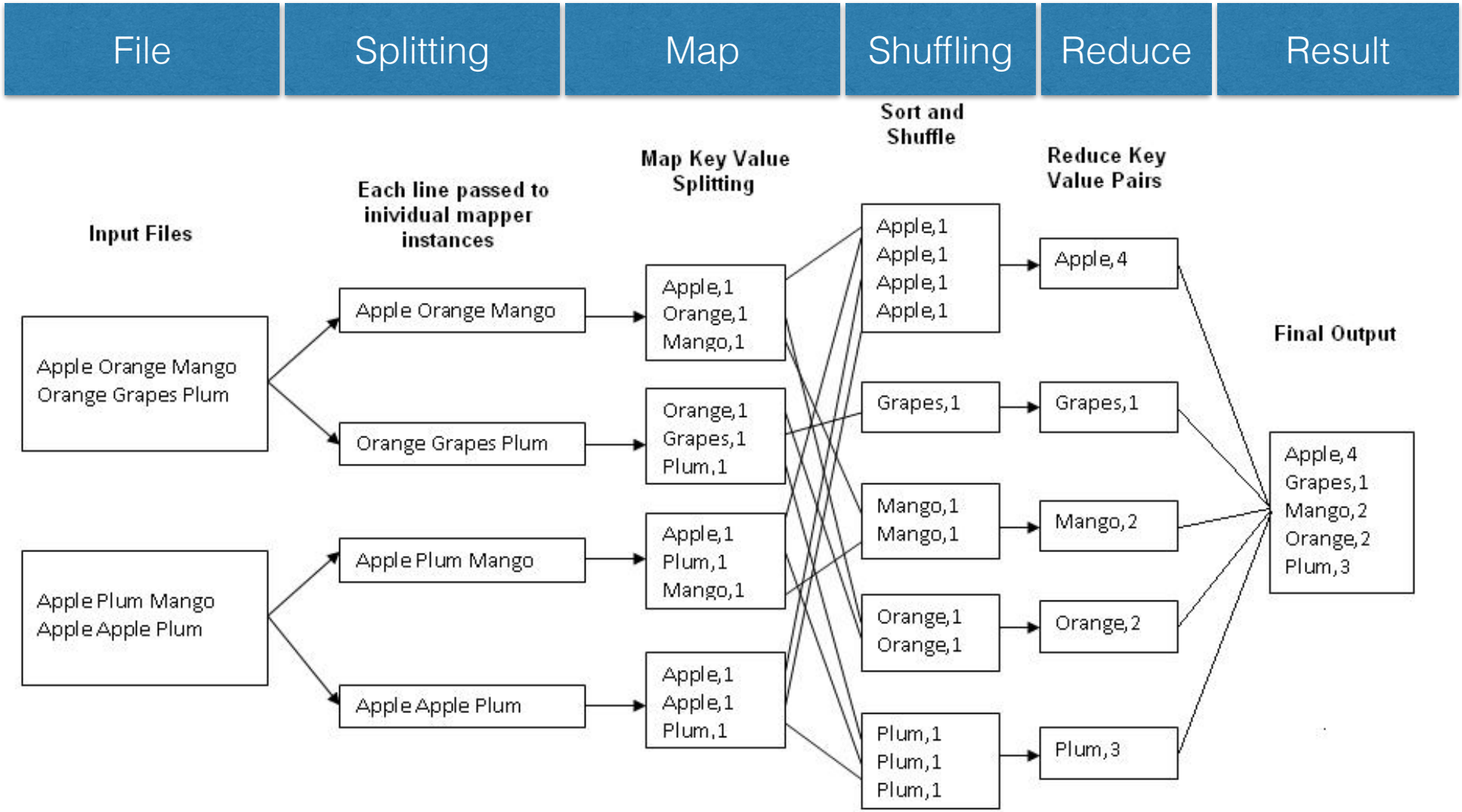
Map Reduce

Exemple



Map-Reduce

WordCount

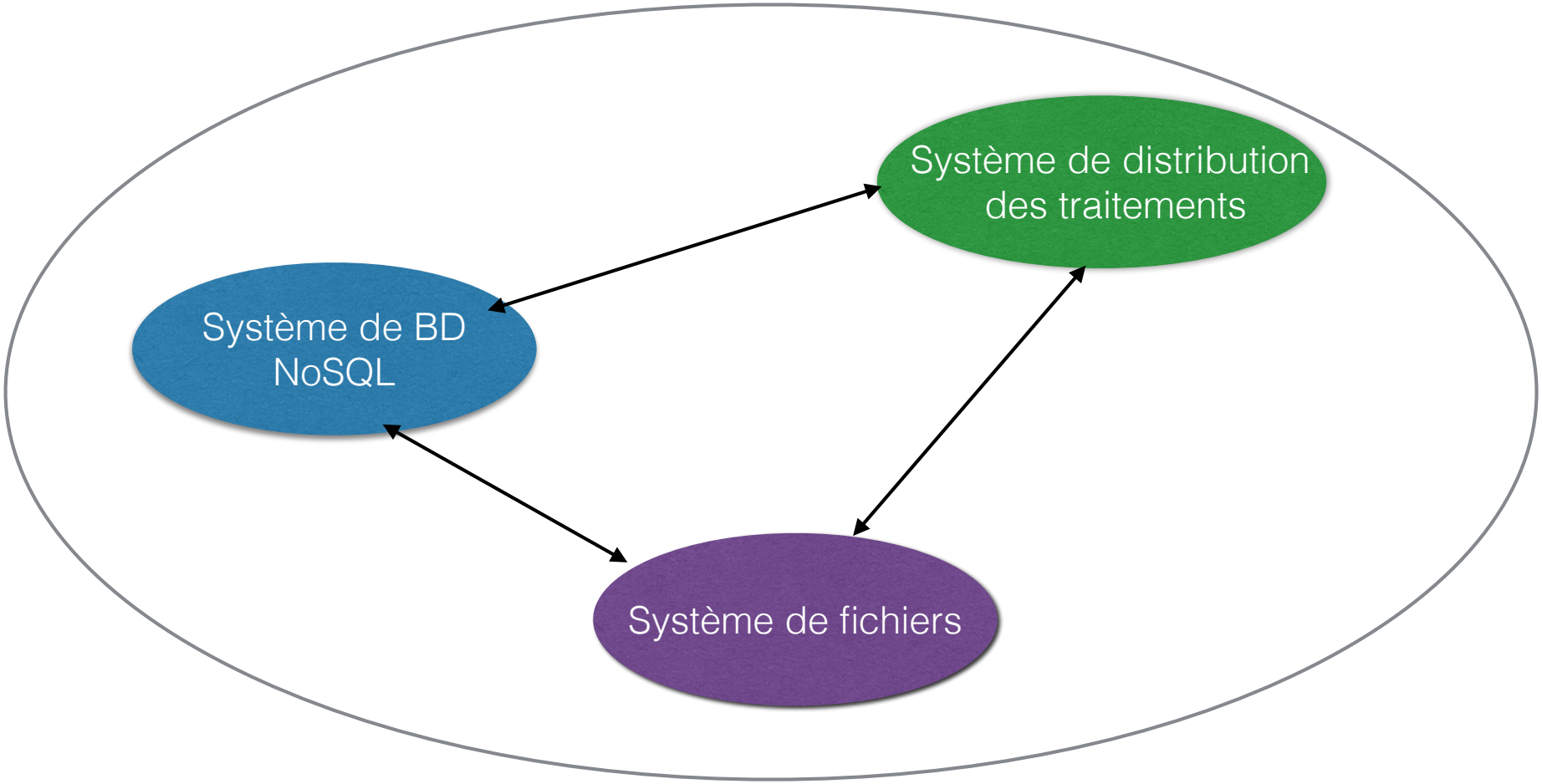


Map-Reduce

WordCount

- Étapes
  - **File** : on lit les fichiers documents en entrée et on initialise les différents « Workers MapReduce »
  - **Splitting** : on distribue les données à traiter sur les différents noeuds du cluster de traitement
  - **Map** : on effectue le compte de chacun des mots et ceci en local sur chaque noeud du cluster de traitement
  - **Shuffling** : on regroupe tous les mots ainsi que leur compte à partir de tous les noeuds de traitement
  - **Reduce** : on effectue le cumul de toutes les valeurs de chaque mot
  - **Result** : on agrège tous les résultats des différentes étapes Reduce et on retourne le résultat final

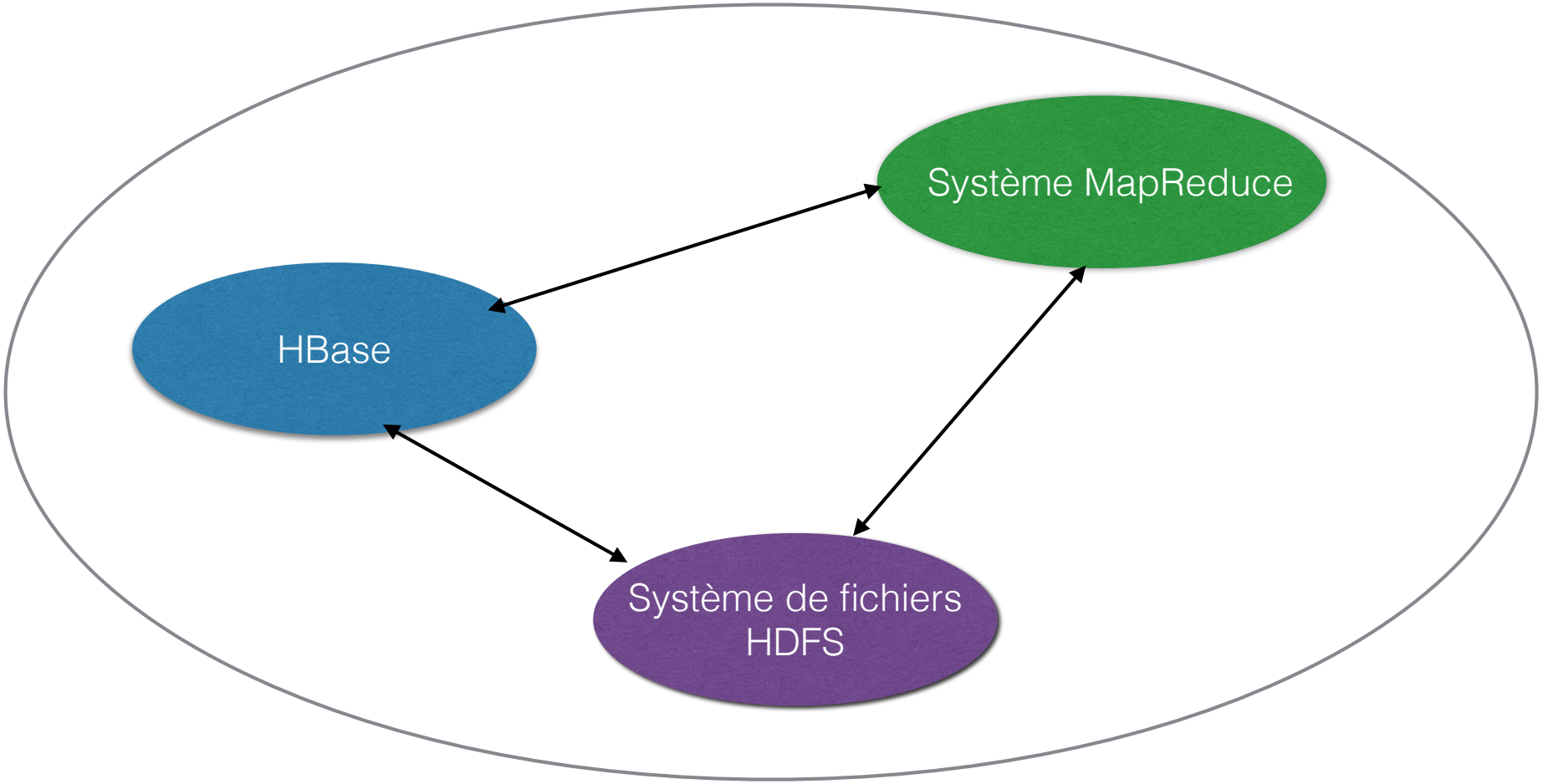
Exemple de framework Big Data



Source : Lacomme et al.

Exemple de framework Big Data

L'exemple Hadoop



Source : Lacomme et al.

Exemple de framework Big Data

L'exemple Hadoop

- Hadoop = High-Avaibility Distributed Object-Oriented Platform
- Framework libre et open source

- Hadoop a été créé en 2002 pour les besoins du projet « Apache Nutch » intégrant MapReduce à la suite de la sortie de l'article de Google en 2004
  - Yahoo est un contributeur majeur
  - Projet indépendant de la fondation Apache depuis 2008

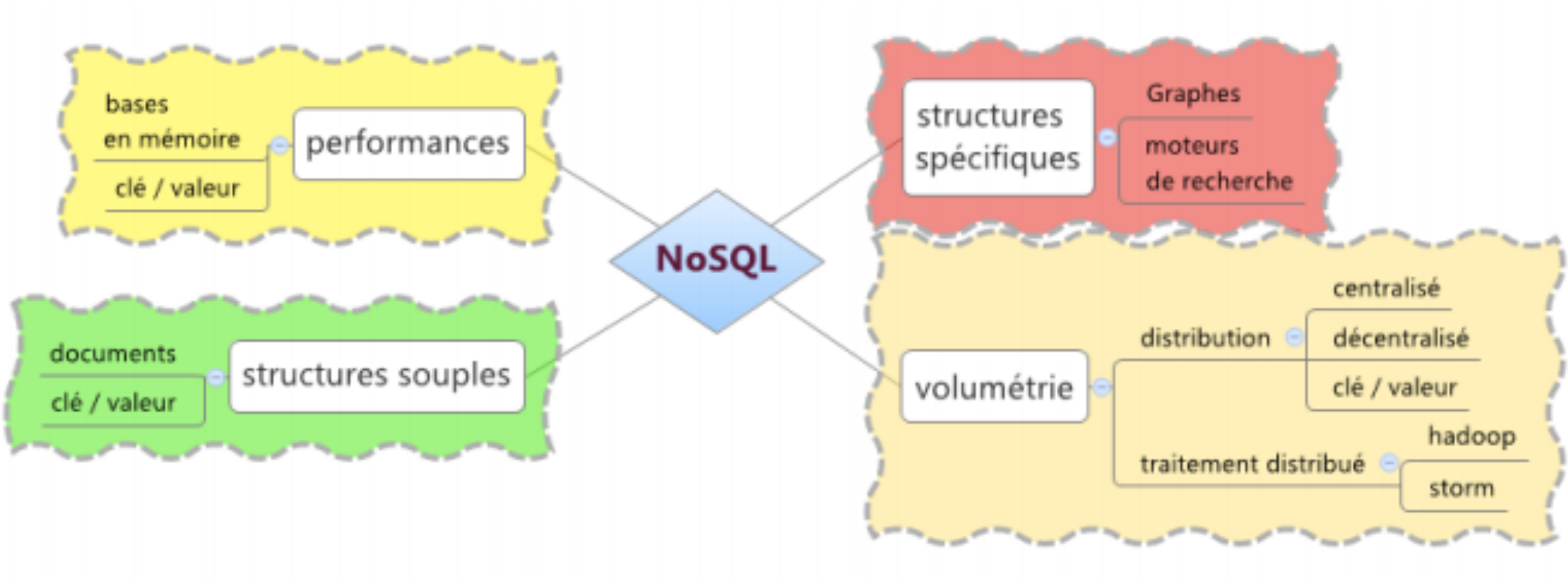
- Utilisé par les géants du Web comme Yahoo, Twitter, LinkedIn, eBay, Amazon…

Classification des modèles NoSQL

- De nombreuses classifications sont possibles !
  - En fonction des usages
  - En fonction du théorème CAP
  - En fonction du modèle de données
    - clé-valeur / document / colonne / graphe
    - NoSQL orienté agrégats / NoSQL orienté graphes

Classification des modèles noSQL

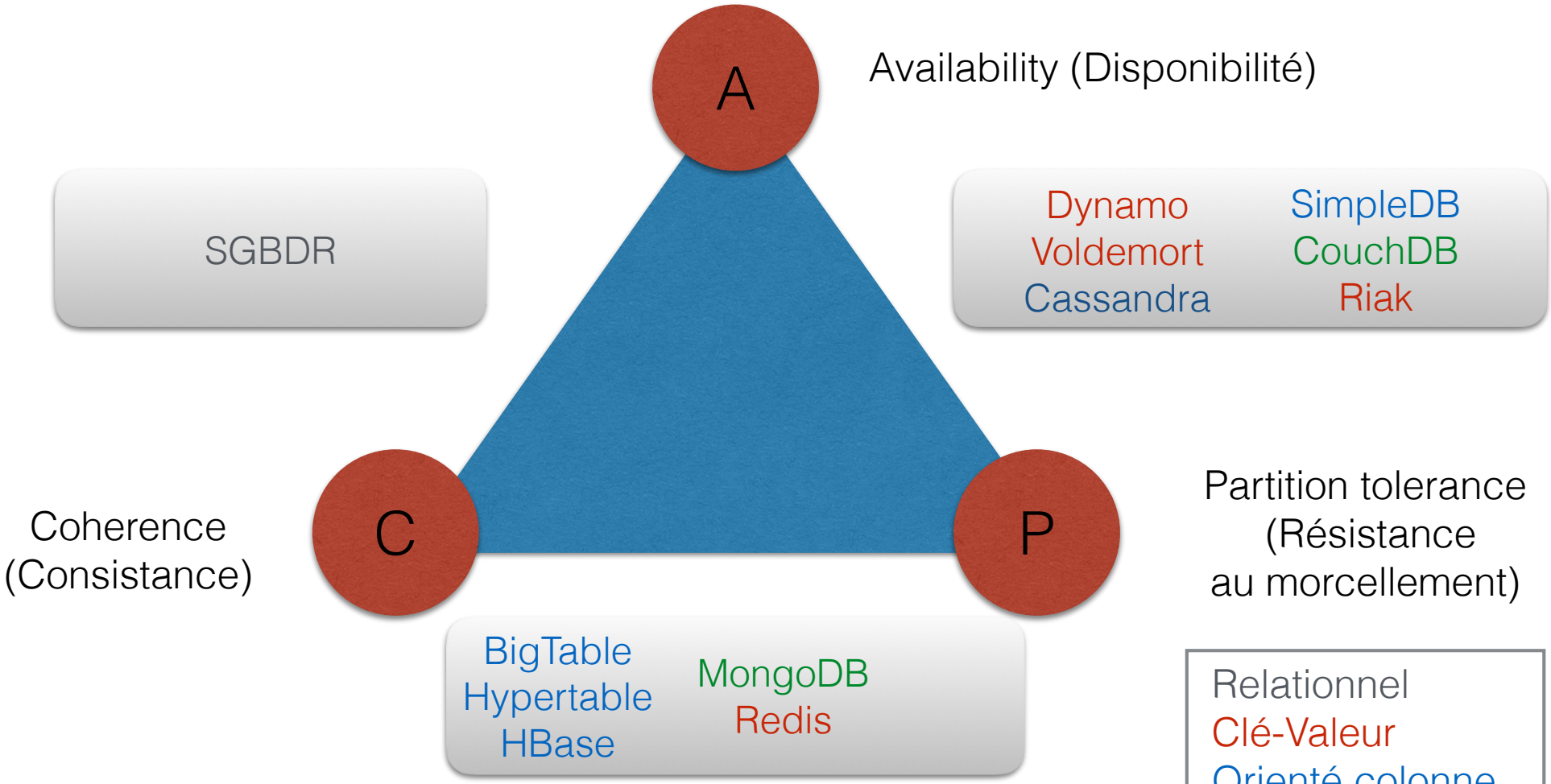
En fonction des usages



Source: R. Bruchez

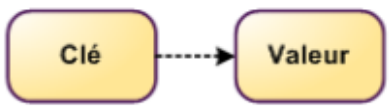
Classification des modèles noSQL

En fonction du théorème CAP

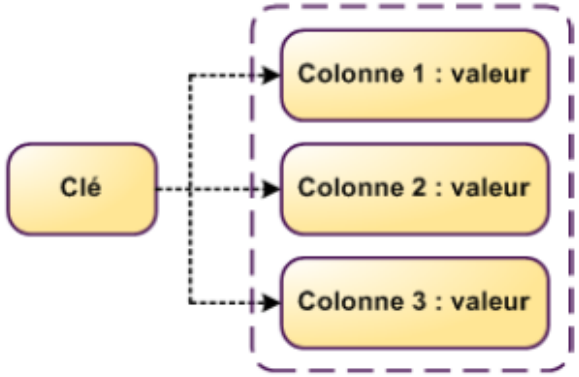


Classification des modèles noSQL

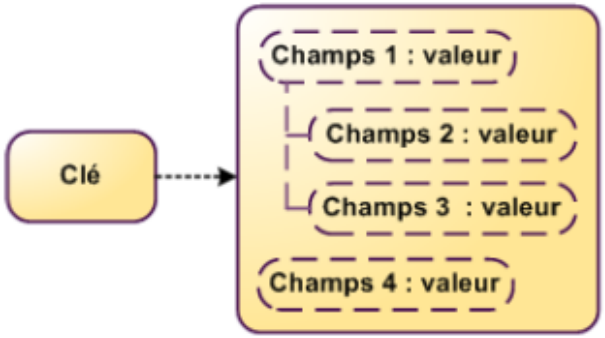
En fonction du schéma des données



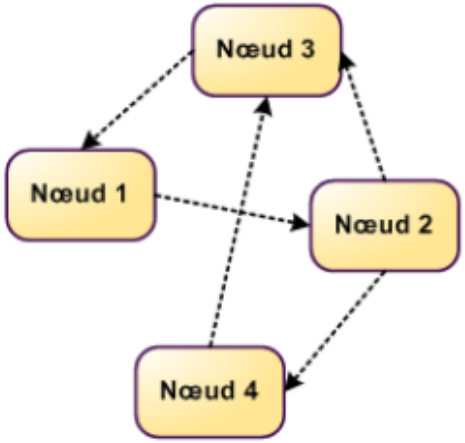
BDD Clé-Valeur



BDD Orientée colonnes



BDD Orientée document



BDD Orientée graphe

Source: Fabien Duchateau

Classification des modèles noSQL

En fonction du schéma des données

Type de modèle	Nom de système	Date de création	Moteur d'interrogation (langage d'interrogation)
Clé/valeur	Redis Oracle NoSQL Voldemort Riak	2009 2012 2008 2008	
Document	CouchDB MongDB ElasticSearch	2005 / 2010	Oui
Colonne	HBase Hypertable Cloudata Cassandra	2007 2007 2011 2008	(HSQL) (CQL)
Graphe	Neo4J FlockDB	2003 2010	(Cypher)

Classification des modèles noSQL

En fonction du schéma des données

- Bases de données orientées agrégats
  - Un agrégat est une collection d'objets reliés que l'on souhaite traiter comme une unité
  - BD NoSQL concernées : Clé-valeur, document et colonne

N'aiment pas les jointures

- Bases de données graphe

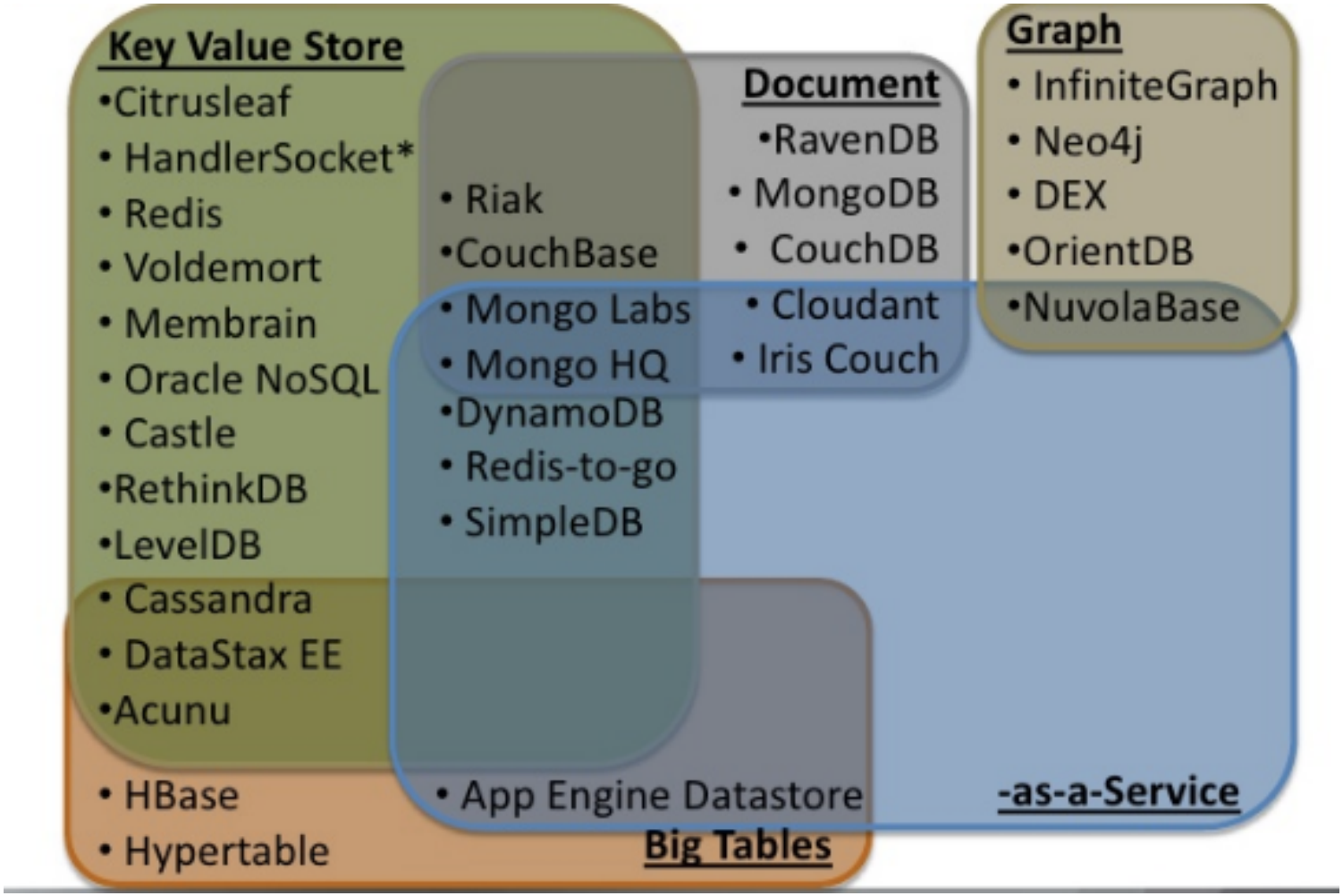
Aiment un peu les jointures

- (Bases de données relationnelles)

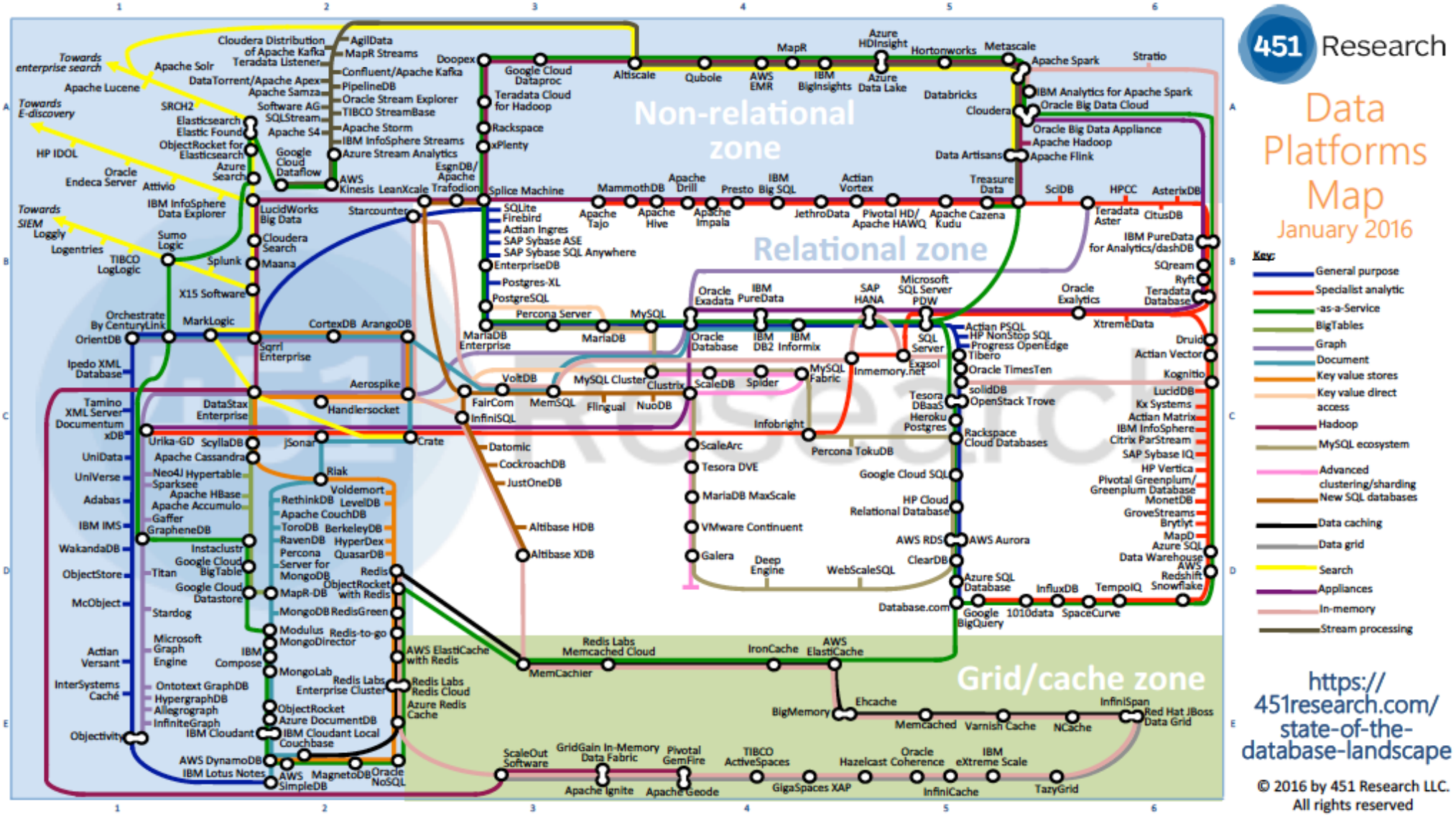
Adorent les jointures

Classification des modèles noSQL

En fonction du schéma des données



Source: 451 Research



Source: 451 Research, Janvier 2016

Ces modèles ont-ils des points communs ?

- Schéma implicite
  - Contre-exemple de taille : Cassandra, qui impose un schéma fort, prédéfini côté serveur
- Absence de relations
  - Chaque collection est indépendante des autres, ce qui permet une distribution aisée des données
  - Contre-exemple fonctionnel : certains moteurs de requêtes comme Hive permettent d'exprimer dans leurs clauses des conditions de jointure
  - Contre exemple structurel : les bases graphes qui sont basées sur des relations
- Logiciels libres

Pour en savoir plus

- http://nosql-database.org/
  Tous les systèmes avec protocole d'accès, langages, interrogation, gestion de la réplication et du sharding

Références

- Livres
  - La révolution Big Data: les données au coeur de la transformation de l'entreprise. J.C. Cointot. Junod, 2014
  - Bases de données noSQL et Big Data. P. Lacomme, S. Aridhi, R. Phan, Ellipse, 2014.
  - NOSQL Distilled: a brief guide to the emerging world of polyglot persistence. Pramod J. Sadalage, Martin Fowler. Addison-Wesley, 2013
  - Les bases de données et le Big Data: Comprendre et mettre en oeuvre. Rudi Bruchez, Eyrolles, 2015
- Supports de cours
  - NoSQL Databases, Christof Strauch, http://www.christof-strauch.de/nosqldbs.pdf
  - Introduction aux systèmes NoSQL, Bernard Espionnasse, mai 2016
  - Les SGBD non relationnels, Fabien Duchateau (accédé en juin 2016)
  - http://blogs.ischool.berkeley.edu/i290-abdt-s12/