Les Bases de Données NoSQL

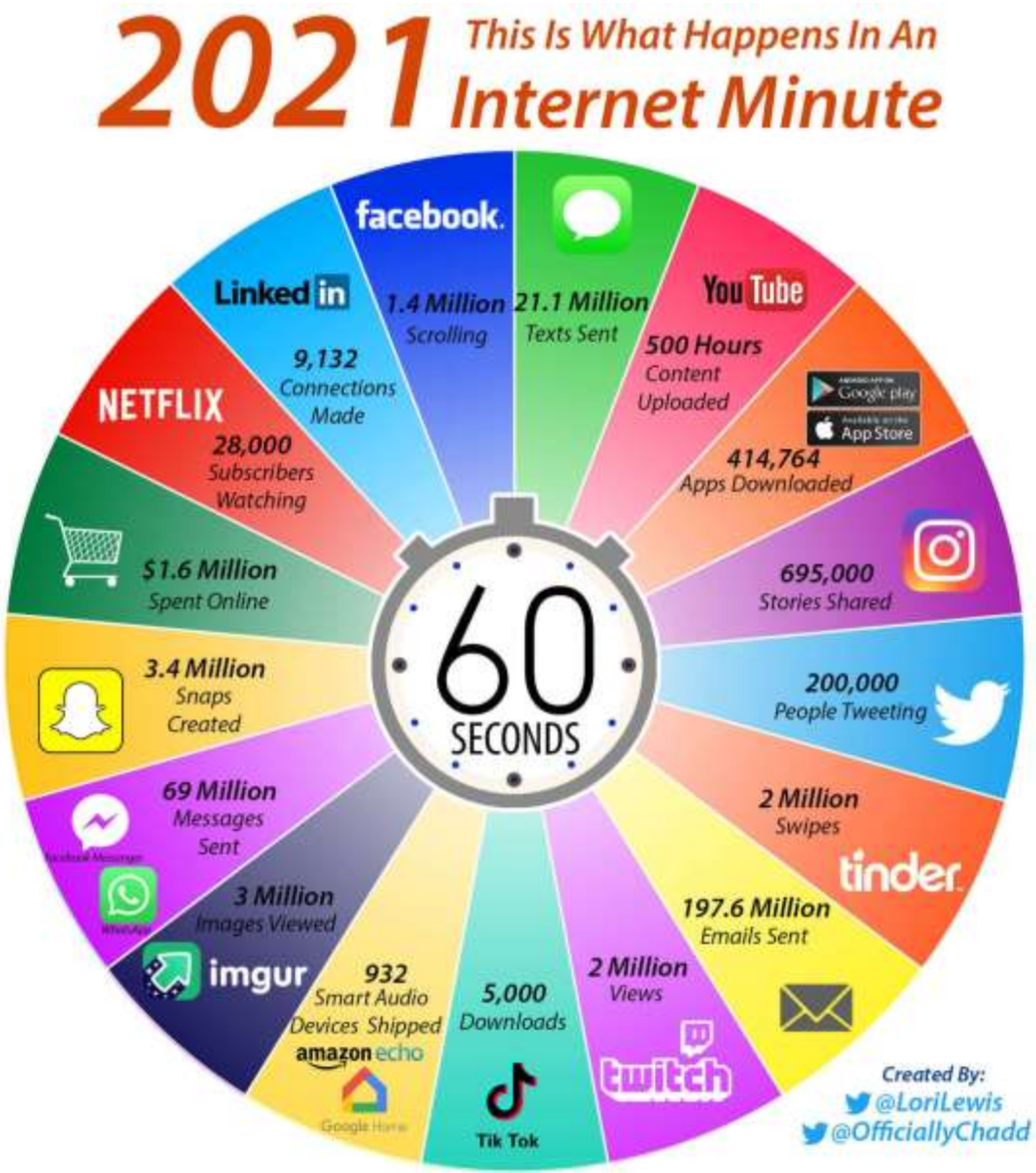
ENSA Khouribga

Pr. SOUSSI Nassima

n.soussi@usms.ma

Sommaire

Introduction au Big Data



Nous parlons en

***Brontobytes*.**

Nom	Symbole	Valeur (en octet)
Kilo-octet	ko	10^3
Méga-octet	Mo	10^6
Giga-octet	Go	10^9
Téra-octet	To	10^{12}
Péta-octet	Po	10^{15}
Exa-octet	Eo	10^{18}
Zetta-octet	Zo	10^{21}
Yotta-octet	Yo	10^{24}
Bronto-octet	Bo	10^{27}

Big Data : Définition

– Le mot **Big Data** est inventé par les Anglo-Saxon pour désigner l'explosion du volumes de données.

– Représente des collections de données très *volumineuses* caractérisées par:

- Une ***vitesse*** de croissement exponentielle,
- Une ***variété*** si grandes qu'ils dépassent l'intuition et les capacités humaines d'analyse et même celles des outils informatiques classiques de gestion de base de données ou de l'information.

Big Data : Définition

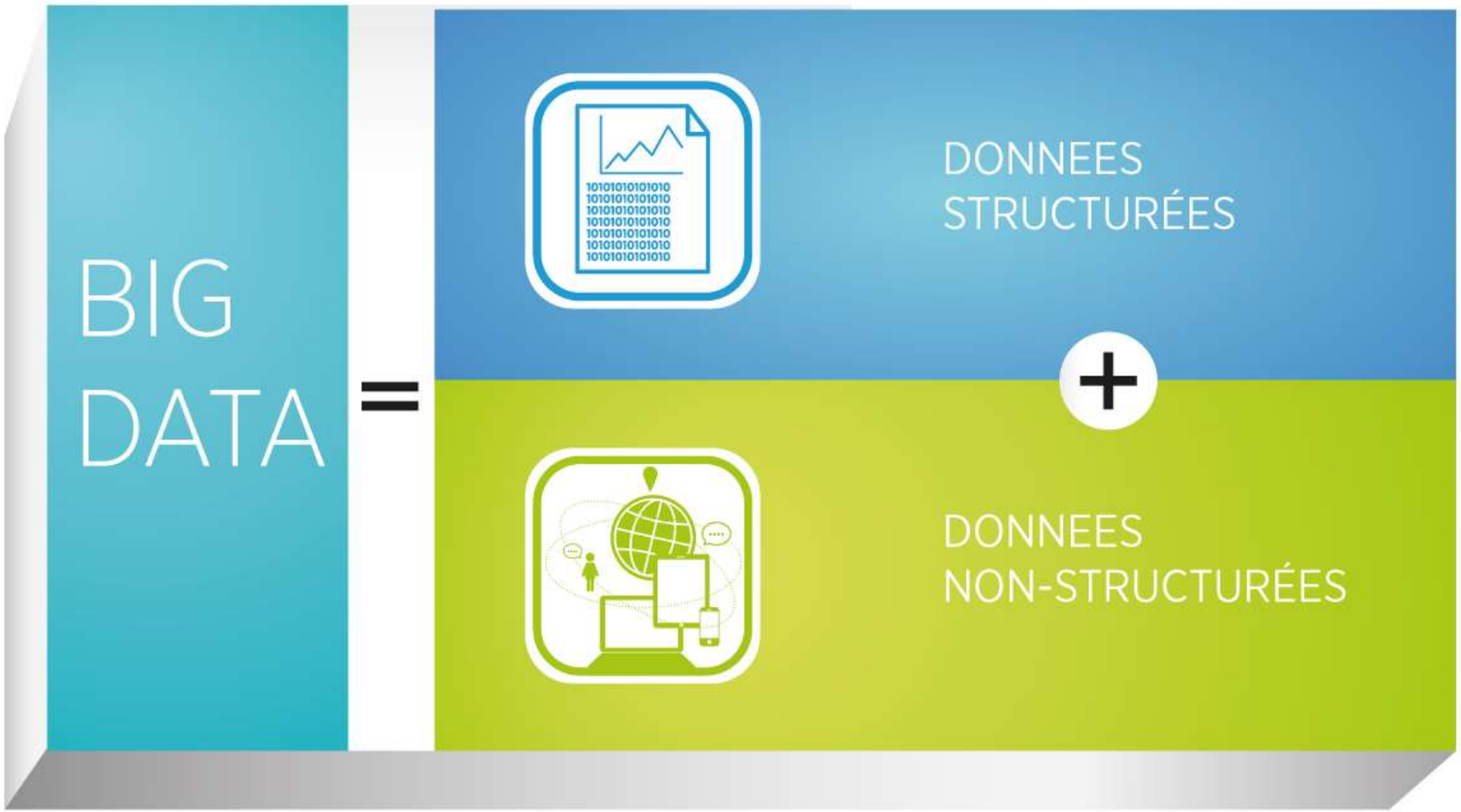
⇒ Statistiquement parlons:

– Si nous prenons toutes les données qui ont été créées dans le monde jusqu'au 2008, la **même quantité de données** est maintenant **générée chaque minute**.

– Plus de 90% des données dans le monde ont été créées au cours de ces dernières années :

- **80%** des données est non structurées,
- **20%** des données qui peuvent être traitées par des systèmes traditionnels.

Big Data : Définition



Big Data : Caractéristiques (Règles)

Pour caractériser le Big Data, on parle de la règle des **3V** qui est devenue par extension, règle des **4V**, puis règle des **5V**.

Big Data : Caractéristiques (Règles)

1. Volume

– Fait référence aux énormes quantités de données à traiter qui ne cesse de s'accroître chaque seconde.

– Sur **Facebook** seulement, nous envoyons 10 millions de messages par jour, « Likons » 4,5 millions de fois et téléchargeons 350 millions de nouvelles photos chaque jour.

=> Nous ne parlons plus en ***Téraoctets*** mais en ***Zettabytes*** et ***Brontobytes***.

Big Data : Caractéristiques (Règles)

2. Variété

Fait référence à l'hétérogénité de format/type de données *(Géolocalisation, vidéos, échanges vocaux, posts sur les réseaux sociaux…)* et des sources de données.

Big Data : Caractéristiques (Règles)

3.Vitesse (Vélocité)

Fait référence à la fréquence du traitement et de génération de données.

Exemples:

1. les transactions bancaires frauduleuses détectées en quelques secondes,
2. le temps que prennent les logiciels pour analyser les réseaux sociaux et capter les comportements qui déclenchent l'achat, dans des millisecondes !

Big Data : Caractéristiques (Règles)

4. Véracité

– Concerne l'authenticité de données (Fiabilité de données).

– Avec autant de formes de grosse donnée, la qualité et la précision sont moins vérifiables (*Faux profils sur les réseaux sociaux, fautes d'orthographe, les abréviations, le langage familier, ...*).

=> Il est indispensable de multiplier les précautions et penser à vérifier systématiquement la vté des données d'entrée et de traiter notamment le cas d'obsolescence des données.

Big Data : Caractéristiques (Règles)

5. Valeur

- Concerne **la création de valeur** (objectif)
- Le succès d'un projet Big Data n'a d'intérêt aux utilisateurs que s'il apporte de la valeur ajoutée et de nouvelles connaissances.

=> **Exploiter les données**, c'est avant tout **répondre à des objectifs commerciaux ou Marketing**. La définition des objectifs orientera l'utilisation des Big Data.

Big Data : Caractéristiques (Règles)

Le Big Data va au-delà des **5Vs** qui ont été extensiers en **8Vs**, puis en **10Vs** :

5 Vs + Visualisation, Variabilité, Validité **= 8 Vs +** Vulnérabilité, Volatilité **= 10 Vs**

Big Data : Caractéristiques (Règles)

6. Visualisation:

– La visualisation des données est une méthode cruciale pour rendre les informations complexes plus accessibles et compréhensibles, en utilisant des graphiques, des tableaux de bord et d'autres outils visuels.

Exemple d'outils de visualisation de Big Data: Tableau, Google Chart, D3.js (Data Driven Document), …

Big Data : Caractéristiques (Règles)

7. Variabilité:

- Fait référence souvent aux changements (*réguliers ou aléatoires*), fluctuations ou évolutions observés dans les données au fil du temps, ce qui nécessite une gestion attentive lors de l'analyse et de l'utilisation de ces informations.

- Variation en termes de format, de fréquence d'arrivée, de structure et d'autres caractéristiques.

Exemple : Données météorologique.

Big Data : Caractéristiques (Règles)

7. Variabilité vs. Variété

- La variété est une **sous-catégorie** de la variabilité.
- La variabilité englobe un large éventail de caractéristiques, dont la variété des types de données et des sources de données est une composante essentielle.
- Les deux concepts sont liés dans la mesure où ils se réfèrent à la diversité et à la complexité des données dans le domaine du Big Data.

Big Data : Caractéristiques (Règles)

8. Validité :

- La validité a quelques similitudes avec la véracité. Comme le sens du mot l'indique, la validité des mégadonnées signifie à quel point les données sont correctes et précises aux fins pour lesquelles elles sont utilisées.
- Il est intéressant de noter qu'une partie considérable des mégadonnées reste inutile, ce qui est considéré comme des «***données obscures***». La partie restante des données non structurées collectées est d'abord nettoyée pour analyse.

Big Data : Caractéristiques (Règles)

8. Validité : *(suite)*

60% du temps d'un scientifique est consacré au nettoyage de ses données avant de pouvoir effectuer une analyse.

=> Avant l'analyse des données massives, il faut passer par le nettoyage profont des données pour garantir une qualité des données cohérente avec des définitions communes.

Big Data : Caractéristiques (Règles)

9. Vulnérabilité :

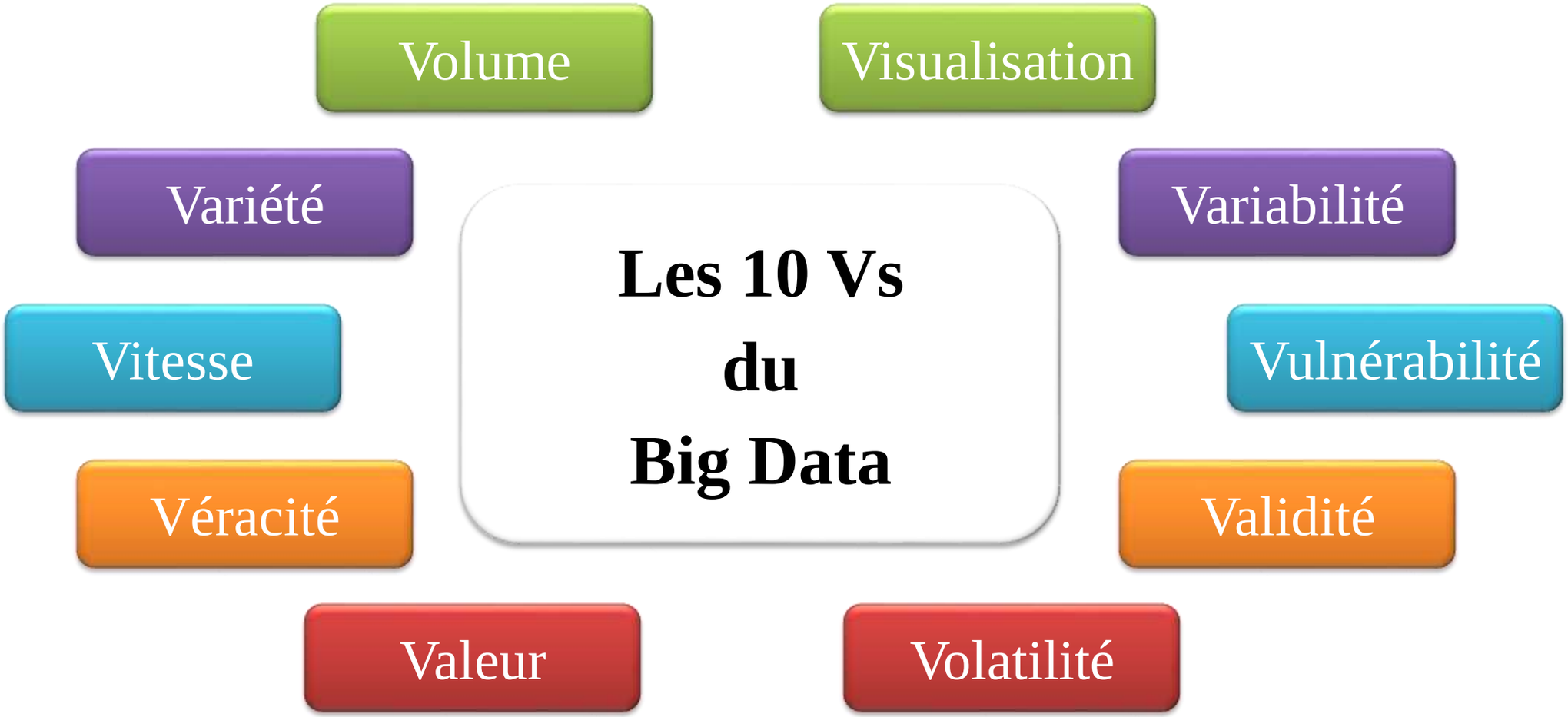
Le Big Data apporte de nouveaux problèmes de sécurité.

10. Volatilité :

– La volatilité des mégadonnées fait référence à la durée de validité des données et à la durée de leur stockage.

⇒ Il faut déterminer *à quel moment les données ne sont plus pertinentes* pour l'analyse actuelle.

Big Data : Caractéristiques (Règles)



Devant ce contexte d'explosion de données semi et non structurées les **systèmes de stockage classique** présentent plusieurs limitations …

Les Limites des Systèmes de Stockage Classique

Les Systèmes de Stockage Classique

- **Fichiers**
- **Base de Données Relationnelle**
- **Entrepôt de Données Classique**

Les Systèmes de Stockage Classique : Fichiers

- **Mode d'accès le plus direct**
  – Données accessible depuis n'importe quel outil de lecture.
- **Fichiers organisés en répertoires**
  – Arborescence permettant de structurer les fichiers.
  – Sur poste de travail ou serveur distant.

Les Systèmes de Stockage Classique : Fichiers

– Fichiers **Plats** : Première ligne (en-tête)
Nombre de colonnes identique

Fichier Plat.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
DepartmentID,Name,GroupName,ModifiedDate
1, Engineering,Research and Development,2002-06-01 00:00:00
2,Tool Design,Research and Development,2002-06-01 00:00:00
3,Sales,Sales and Marketing,2002-06-01 00:00:00
4,Marketing,Sales and Marketing,2002-06-01 00:00:00
5,Purchasing,Inventory Managment,2002-06-01 00:00:00
6,Research and Development,Research and Development,2002-06-01 00:00:00
7,Production,Manufacturing,2002-06-01 00:00:00
8,Production Control,Manufacturing,2002-06-01 00:00:00
9,Human Resources,Executive Gener and Administration,2002-06-01 00:00:00
10,Finance,Executive General and Administration,2002-06-01 00:00:00
11,Information Services,Executive General and Administration,2002-0-01 00:00:00
12,Document Control,Quality Assurance,2002-06-01 00:00:00
13,Quality Assunrance,Quality Assurance,2002-06-01 00:00:00
14,Facilities and Maintenance,Executive General and Administration,2002-06-01 00:00:00
15,Shipping and Receiving,Inventory Management,2002-06-01 00:00:00
```

Les Systèmes de Stockage Classique : Fichiers

– Fichiers **XML**: Très utiliser pour l'échange de données.
Structure arborescence: racine avec enfants.

```
<questionnaire>
    <item categorie="Entertainment">
        <question>Who is known as the original drummer of the Beatles?</question>
        <reponses>
            <reponse  valeur="true">Pete Best</reponse>
            <reponse  valeur="false">Ringo Starr</reponse>
            <reponse  valeur="false">Stu Sutcliffe</reponse>
            <reponse  valeur="false">George Harrison</reponse>
        </reponses>
        <aide>Was fired before the Beatles hit it big.</aide>
        <fait>Pete stayed until shortly after their first audition for EMI in 1962, but was fired on August
16th of that year, to be replaced by Ringo Starr.</fait>
    </item>
    <item categorie="History">
        <question>What ship rests in the bottom of Pearl Harbor?</question>
        <reponses>
            <reponse valeur="true">USS Arizona</reponse>
            <reponse  valeur="false">USS Iowa</reponse>
            <reponse  valeur="false">USS Wisconsin</reponse>
            <reponse valeur="false">USS Kentucky</reponse>
        </reponses>
        <aide>Was also the 48th state admitted into the U.S. and the last of the contiguous states admitted.
</aide>
        <fait>The USS Arizona Memorial, dedicated in 1962, spans the sunken hull of the battleship without
touching it. The 19,585 pound anchor of the Arizona is displayed at the entrance of the visitor center
        </fait>
    </item>
</questionnaire>
```

Les Systèmes de Stockage Classique : Fichiers

- Fichiers **JSON**:
  - Similaire à XML avec une syntaxe plus légère.
  - Plus facile à aborder en programmation.
  - Fréquemment utilisé pour les service web et les BD NoSQL.

```json
{
  "Time": "2017-12-26T22:39:08",
  "Uptime": 4,
  "Vcc": 3.25,
  "POWER": "ON",
  "Wifi": {
    "AP": 1,
    "SSId": "JAZZTEL_david",
    "RSSI": 72,
    "APMac": "20:89:86:1E:31:E2"
  }
}
```

Les Systèmes de Stockage Classique : Fichiers

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Arduino SD Card Web Page</title>
        <style type="text/css">
            h1 {
                font-family: courier, courier-new, serif;
                font-size: 20pt;
                color: blue;
                border-bottom: 2px solid blue;
            }
            p {
                font-family: arial, verdana, sans-serif;
                font-size: 12pt;
                color: #6B6BD7;
            }
            .red_txt {
                color: red;
            }
        </style>
    </head>
    <body>
        <h1>Arduino SD Card Page with CSS</h1>
        <p>Welcome to the Arduino web page with CSS styling.</p>
        <p class="red_txt">This text is red.</p>
        <p>This paragraph has one word that uses <span class="red_txt">red</span> text.</p>
    </body>
</html>
```

- Fichiers **HTML**:
  - Langage des pages web.
  - Interpréter par les navigateurs
  - Intégrant des annexes (jpg, css, js, …)

Les Systèmes de Stockage Classique : Fichiers

– Fichiers **Logs**:

Trace enregistrée par un serveur lors d'un événement.

- Démarrage ou erreur sur une application
- Clic sur une page web
- Démarrage d'une session
- …

Pas de format universel mais certains élément sont toujours présent:

- Timestamp (date + heure)
- Identifiant du système ayant produit le log
- Informations pour l'interprétations (adresse page web, …)

Les Systèmes de Stockage Classique : Fichiers

- Fichiers **Brute**:
  - Suite de caractère peu structuré
  - Traitement du langage naturel (NPL) pour avoir des informations.

- Fichiers **Binaire**:
  - Suite d'octets non lisible: Image, video, programme, …
  - Prétraitement à faire avant d'exploiter les données.

Les Systèmes de Stockage Classique : Fichiers

Limitations des Fichiers:

Compliquer de traiter efficacement des grands volumes de données via les fichiers de structures hétérogènes.

Les Systèmes de Stockage Classique : BDR

Une base de données relationnelle offrent:

– Un 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.

Les Systèmes de Stockage Classique : BDR

Limitations:

Le modèle relationnel est moins efficace et peu adapté au stockage et à l'interrogation des données (semi/non structurées):

– Toutes les lignes d'une table ont les mêmes colonnes.

– Elle respecte un schéma de base imposé à toute la BD → Mise à jour du schéma des données est peu performant et couteux en temps de développement.

– Interrogation par jointures → requête complexe, lourdes et couteuse en temps d'exécution.

Les Systèmes de Stockage Classique : BDR

Limitations: *(suite)*

– Incapable de stocker et gérer de très grands volumes de données (de l'ordre du brontobyte) : limité à quelques To,

– Elle ne savent pas se distribuer massivement.

– Non adaptée pour faire des calculs de statistiques complexes sur des valeurs ni sur des données volumineuses!

Les Systèmes de Stockage Classique : DW

– Le **Data warehouse** est une base données conçue pour stocker la data pour des fins décisionnelles → Il s'agit d'une vaste base de données relationnelle dédié au décisionnel.

– Les informations qui y sont stockées offrent une vue d'ensemble chronologique afin de pouvoir offrir la meilleure capacité d'analyse possible de la data.

– dispose d'un outil intégré nommé ETL pour l'extraction, la transformation et le chargement de données.

– Les données sont orientées « **sujet** », « **thème** ».

Les Systèmes de Stockage Classique : DW

➢ Avantages :

Le grand **avantage** d'un Data warehouse pour une entreprise est le fait qu'il représente une façon pratique de visualiser les données passées sans affecter les opérations actuelles.

⇒ Les entreprises exploitent la donnée stockée dans le Data warehouse dans une optique de Business intelligence.

⇒ Elles peuvent optimiser leurs actions commerciales grâce à une connaissance plus fine de leur marché, de leurs clients, concurrents et prospects.

Les Systèmes de Stockage Classique : DW

➢ Limites :

Les **entrepôts de données classiques** s'appuient sur des bases de données relationnelles pour stocker des données structurées relatives à l'historique de l'entreprise dans des tables de fait.

→ Il est nécessaire de changer ou adapter l'architecture existante des DW pour stocker, traiter et faire l'analyser en temps réel des grandes quantités de données non structurées.

Les Systèmes de Stockage Classique : DW

Parmi les **travaux de recherches** orientés vers l'adaptation des Data Warehouse au Big Data :

– Mallek, H., Ghozzi, F., & Gargouri, F. (2020). Towards Extract-Transform-Load Operations in a Big Data context. *International Journal of Sociotechnology and Knowledge Development (IJSKD)*, *12*(2), 77-95.

– Dehdouh, K., Boussaid, O., & Bentayeb, F. (2020). Big Data Warehouse: Building Columnar NoSQL OLAP Cubes. *International Journal of Decision Support System Technology (IJDSST)*, *12*(1), 1-24.

– …

Big Data : Besoins

Les principaux besoins liés à l'émergence du Big Data sont:

- Un schéma de données flexible.
- La capacité de stockage, traitement et lecture de données non structurées et semi-structurées.
- La performance des requêtes en lecture.
- La haute disponibilité des données.
- La capacité à gérer une montée en charge horizontale.
- La parallélisassions des traitements des requêtes.

Base de Données NoSQL

BD NoSQL: Définition

– Le terme NoSQL a été créé par Johan Oskarsson en 2009 au cour d'un meeting à San Fransisco.

– Elle s'écarte au paradigme classique des BD relationnelle.

– Permet de gérer une grandes quantité de données avec une haute scalabilité et flexibilité.

Scalabilité : Définition

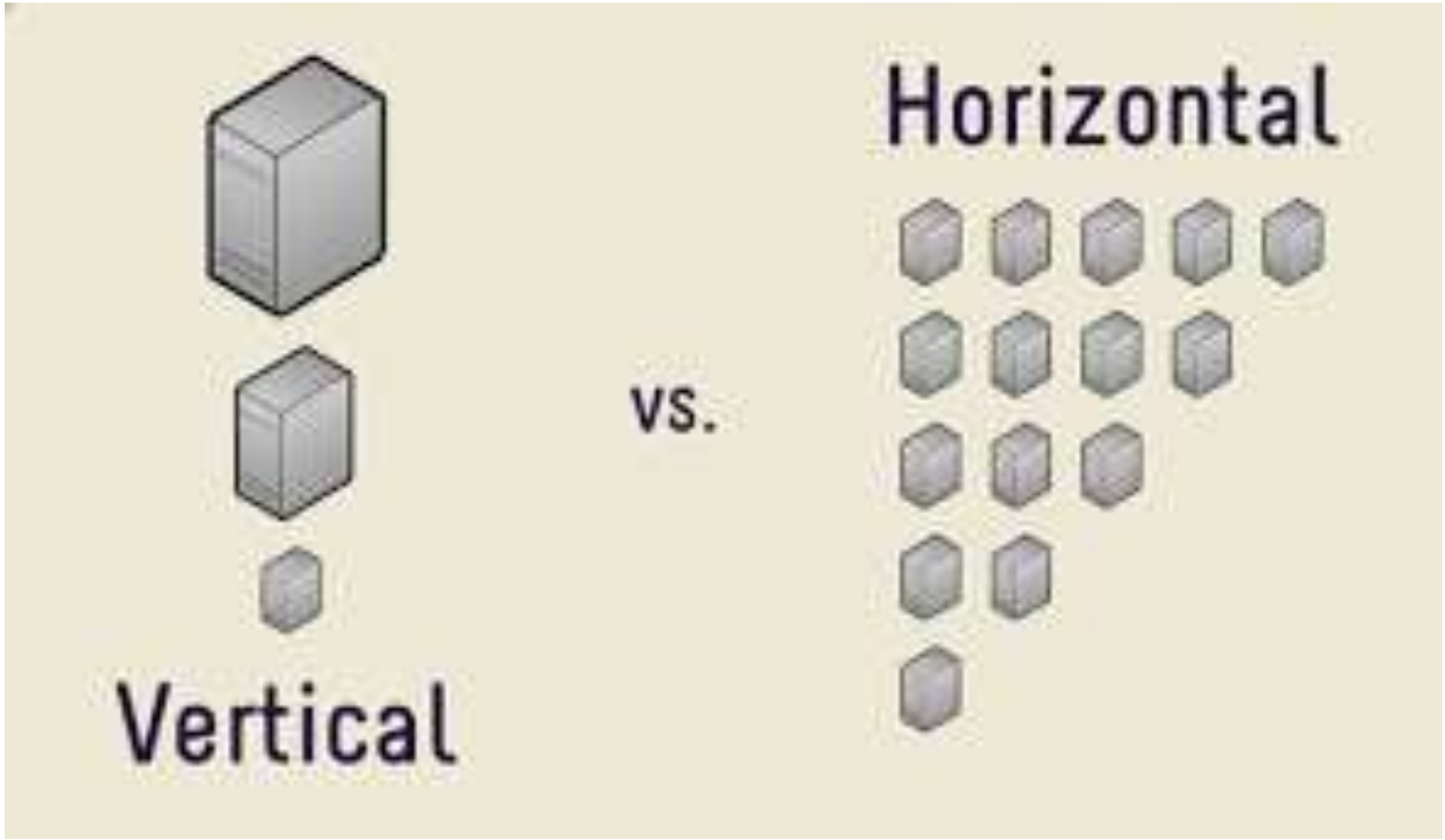
Lorsque la **charge du serveur augmente**, et qu'il est temps de rajouter des ressources matérielles à votre infrastructure, on parle de la **scalabilité**:

⇒ C'est la capacité qu'a l'architecture pour évoluer en cas de montée en charge si nécessaire.

– **Scalabilité horizontale** : possibilité d'ajouter des serveurs d'un type donné.

– **Scalabilité verticale** : possibilité d'upgrader un serveur (ajout de processeurs, RAM, disques…).

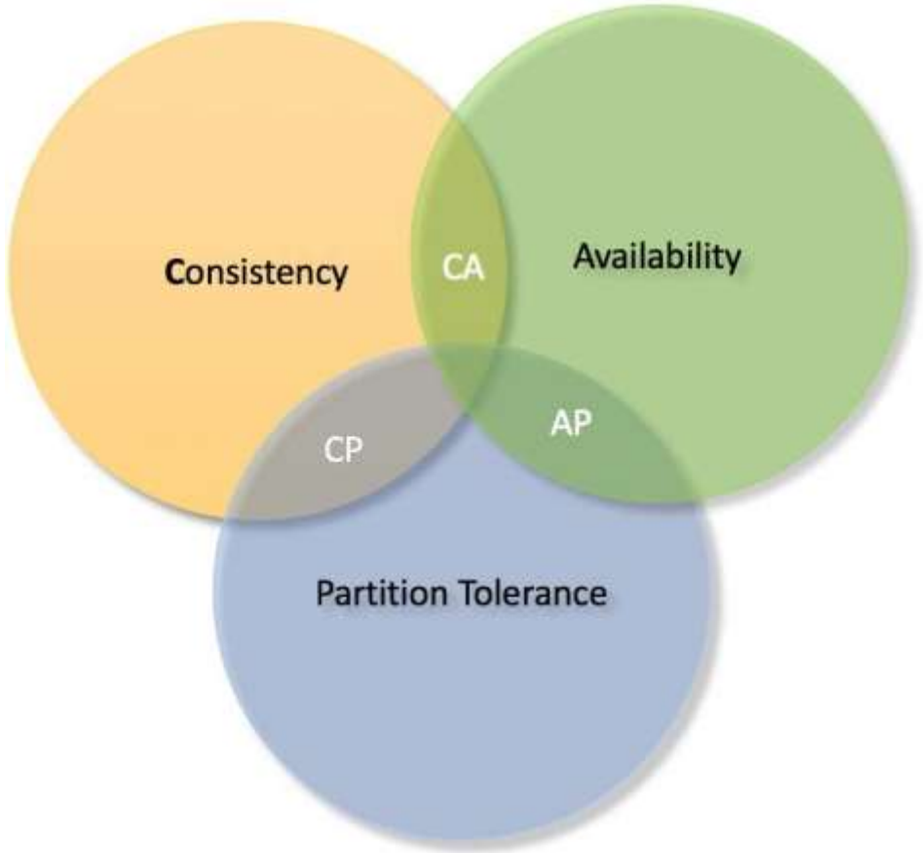
Scalabilité : Horizontale vs. Verticale



BD NoSQL: Théorème du CAP

Les bases de données NoSQL suivent le **théorème CAP** qui annonce qu'il est impossible pour un système distribué de fournir les 3 propriétés suivante **à la fois**:

- Disponibilité
- Cohérence
- Tolérance au partitionnement



BD NoSQL: Théorème du CAP

– **Cohérence** (Consistency) : tous les nœuds (clients) du système **voient exactement les mêmes données au même moment**.

– **Disponibilité** (Availability) : **tous les nœuds actifs** du système réparti **renvoient une réponse valide à toute demande**, sans exception.

– **Tolérance au partitionnement** (Distribution) : Un partitionnement est une rupture de communication au sein d'un système réparti, une perte ou un retard temporaire de connexion entre deux nœuds. La tolérance au partitionnement signifie que le cluster **doit continuer à fonctionner quel que soit le nombre d'interruptions entre les nœuds du système**.

BD NoSQL: Théorème du CAP

Les bases de données NoSQL sont tenues de **respecter** deux principes sur trois. On choisi l'un des trois cas en fonction des besoins à combler:

- Cohérence et Disponibilité (CA)
- Disponibilité et la tolérance au partitionnement (AP)
- Cohérence et la tolérance au partitionnement (CP)

BD NoSQL: Théorème du CAP

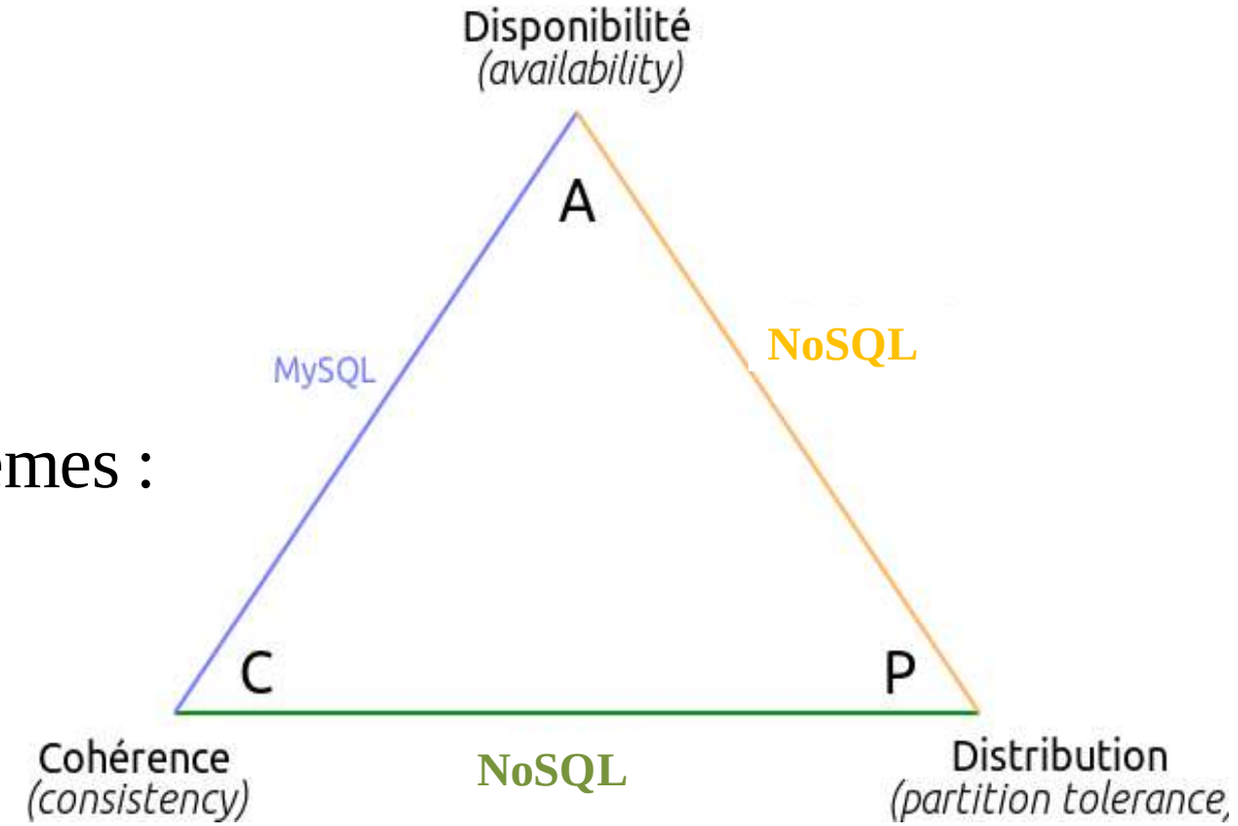
Les « **SGBDR** » sont des systèmes:

– **AC** *(Cohérent et Disponible)*

Les SGBD « **NoSQL** » sont des systèmes :

– **AP** *(Disponible et Distribué)*

– **CP** *(Cohérent et Distribué)*

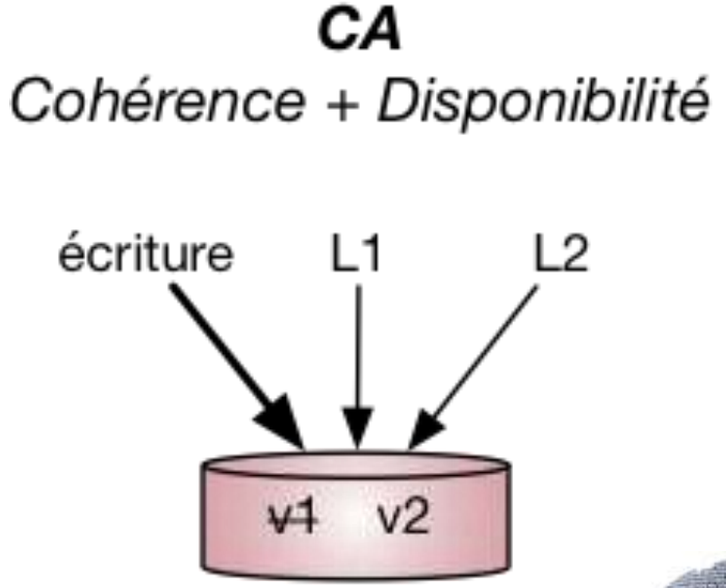


BD NoSQL: Théorème du CAP

Le Couple **CA** (*Cohérence et Disponibilité*):

– Il représente le fait que lors d'opérations concurrentes sur une même donnée, les requêtes L1 et L2 retournent la nouvelle version (v2) et sans délai d'attente.

– Cette combinaison n'est possible que dans le cadre de bases de données transactionnelles telles que les SGBDR.



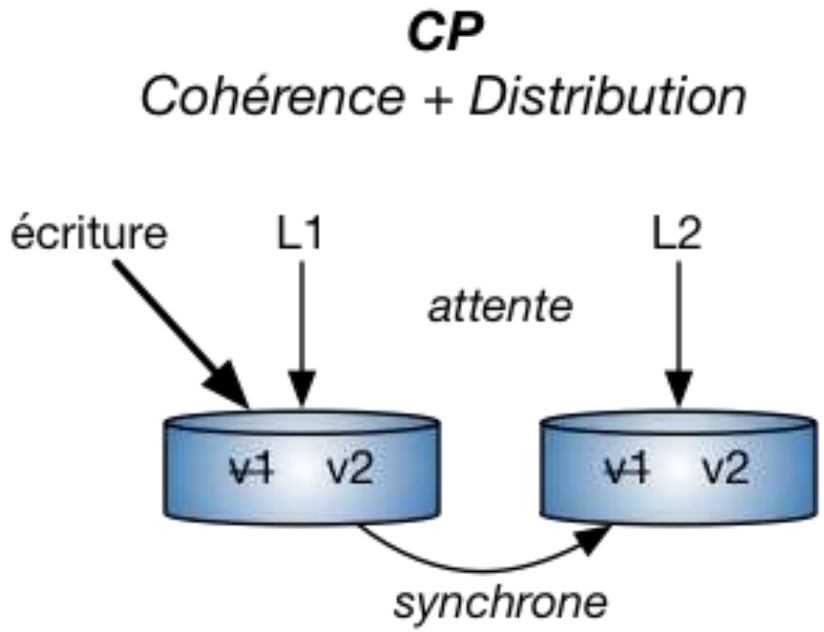
BD NoSQL: Théorème du CAP

Le Couple **CP (Cohérence – Distribution)**:

– Il propose de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réplication).

– En même temps, il est nécessaire de vérifier la cohérence des données en garantissant la valeur retournée malgré des mises à jour concurrentielles.

– L1 et L2 attendent la synchronisation pour voir v2



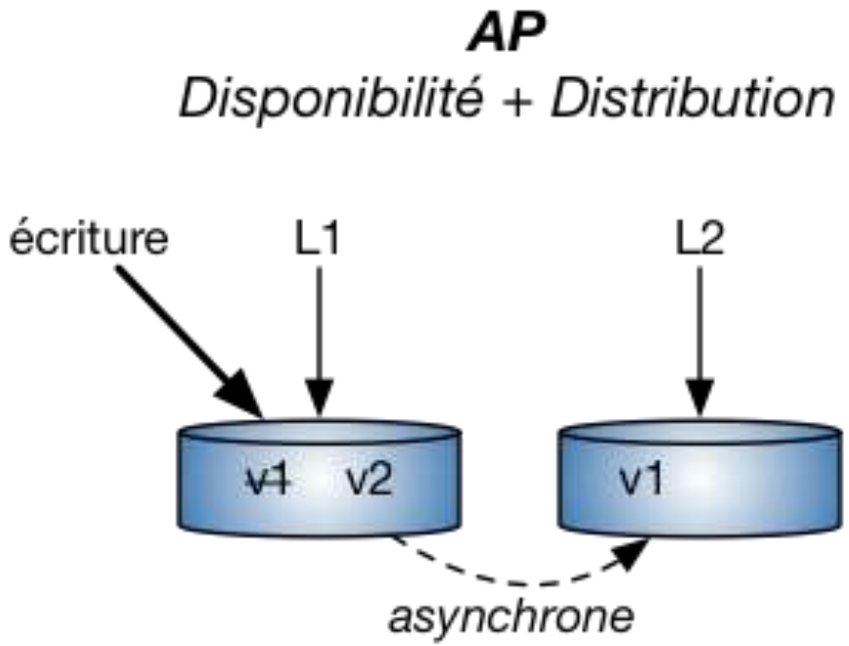
BD NoSQL: Théorème du CAP

Le Couple **AP (Disponibilité – Distribution):**

– Il s'intéresse à fournir un temps de réponse rapide tout en distribuant les données et les réplicas.

– Les mises à jour sont asynchrones sur le réseau, et la donnée est finalement cohérente.

– L1 voit la version v2, tandis que L2 voit la version v1.

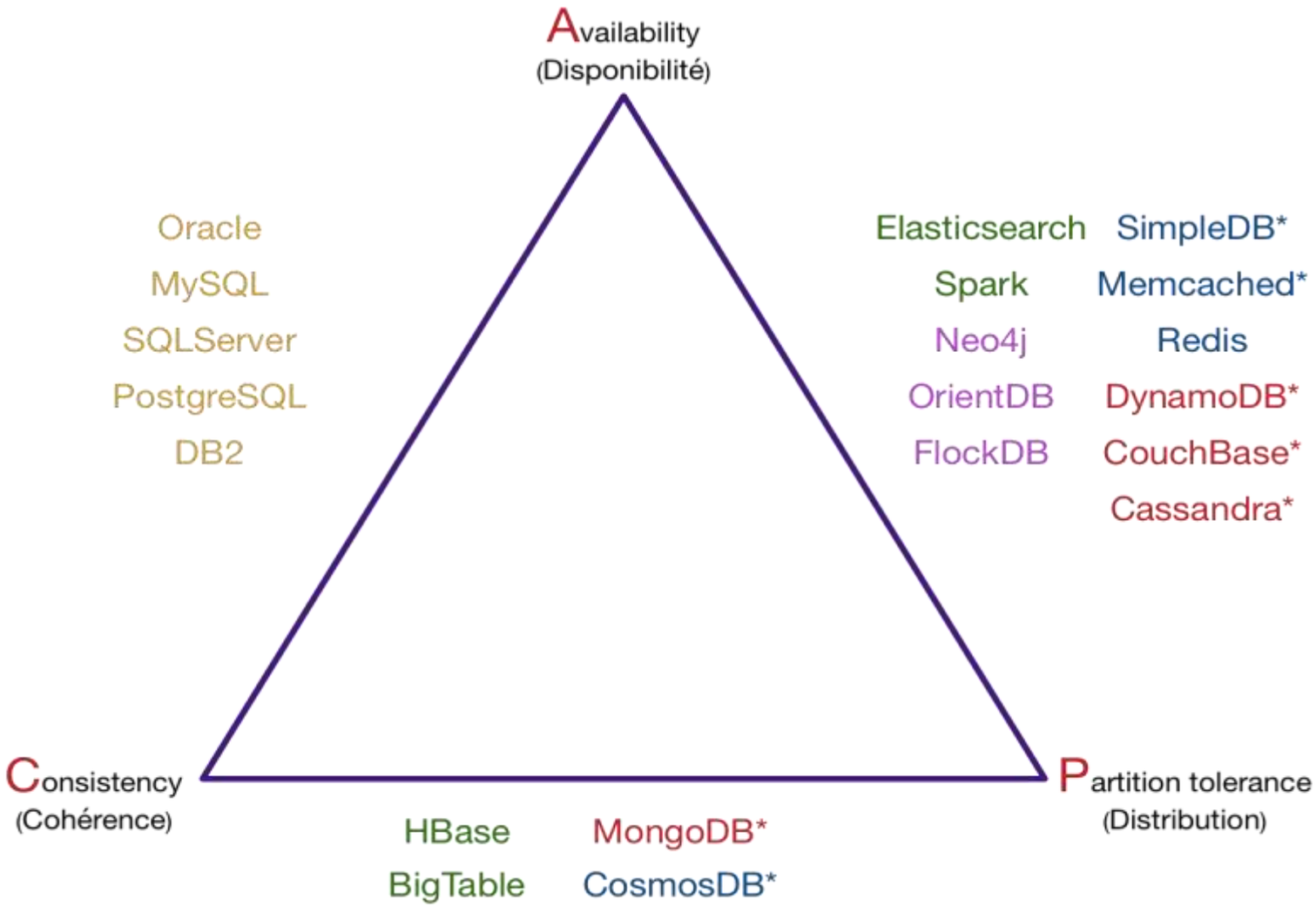


BD NoSQL: Théorème du CAP

Modèle

Relationnel
Orienté Clé-Valeur
Orienté Colonne
Orienté Document
Orienté Graphe

* Possibilité de changer
le mode de cohérence
Cohérence <-> Disponibilité



BD NoSQL: Caractéristiques

Les BD NoSQL :

– Adoptent une représentation de données non relationnelle

– Ne remplacent pas les BD relationnelles mais c'est une alternative, un complément apportant des solutions plus intéressantes dans certains contextes

– Apportent une plus grande performance dans le contexte des applications Web avec des volumétries de données exponentielle.

BD NoSQL: Caractéristiques

Les BD NoSQL: *(suite)*

– Utilisent une très forte distribution de ces données et des traitements associés sur de nombreux serveurs

– Pas de schéma pour les données ou schéma dynamique

– Données de structures complexes ou imbriquées.

– Partitionnement horizontal des données (sharding) sur plusieurs noeuds (serveurs).

– Réplication des données sur plusieurs nœuds.

Fondements des Systèmes NoSQL

Fondements des Systèmes NoSQL

1. Sharding
2. Map Reduce
3. MVCC

Fondement 1 : Sharding

« Sharding » = Partitionnement des données sur plusieurs serveurs

– **Un ensemble de techniques** qui permet de **répartir** les données sur plusieurs nœuds/serveurs (shards) pour assurer la scalabilité de l'architecture.

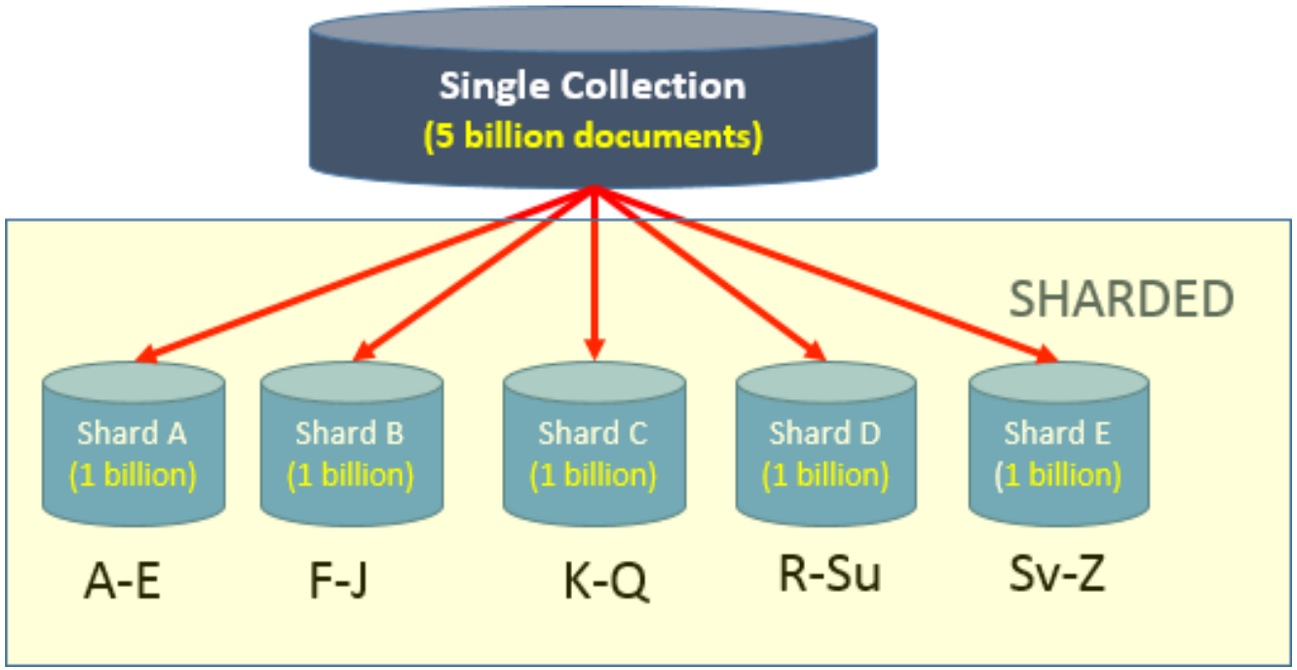
– **Mécanisme de partitionnement horizontal** des données dans lequel les objets-données sont dispachés uniformément sur des nœuds serveurs différents en fonction d'une clé de sharding.

– La répartition des données est gérée automatiquement par le système.

Fondement 1 : Sharding

Concrètement, il s'agit de créer un cluster appelé **sharded cluster** composé de plusieurs machines (appelées **shard**) sur lesquelles les données contenues dans notre base vont être réparties selon une clé de sharding.

An Example of sharding a single collection to five sharded servers on an aphanumeric key field



Fondement 1 : Sharding

– L'ensemble des serveurs forment un cluster.

– Quand les besoins s'augmentent il suffit d'ajouter des serveurs au cluster.

⇒ On a ainsi un système que l'on peut faire évoluer facilement selon les besoins (grande scalabilité).

Remarque: La capacité d'extension est théoriquement infinie car il n'y a à ce jour pas de limites au nombre de machines que l'on peut ajouter.

Fondement 1 : Sharding

Avantages :

– **Le "*Sharded Cluster*" est économique** : On ne travaille plus avec une seule machine extrêmement performante, mais avec un cluster composé de plusieurs machines de milieux de gamme, nettement moins coûteuses.

– **les requêtes se feront sur des jeux de données moins volumineuse et pourront être parallélisées**, ce qui permet d'obtenir des temps de réponse plus rapides.

– Ajouter des serveurs supplémentaires sans interruption du service.

Fondement 2 : MapReduce

– **Modèle de traitement parallèle** (framework de calcul distribué) pour le traitement d'une grandes masses de données.

– **Développé par Google** pour le traitement de gros volumes de données en environnement distribué :

- Permet de répartir la charge sur un cluster,
- Gère entièrement, de façon transparente la distribution de données dans le cluster.

– **La librairie *MapReduce* existe dans plusieurs langages** (C++, C#, Java, Python, …)

Fondement 2 : MapReduce

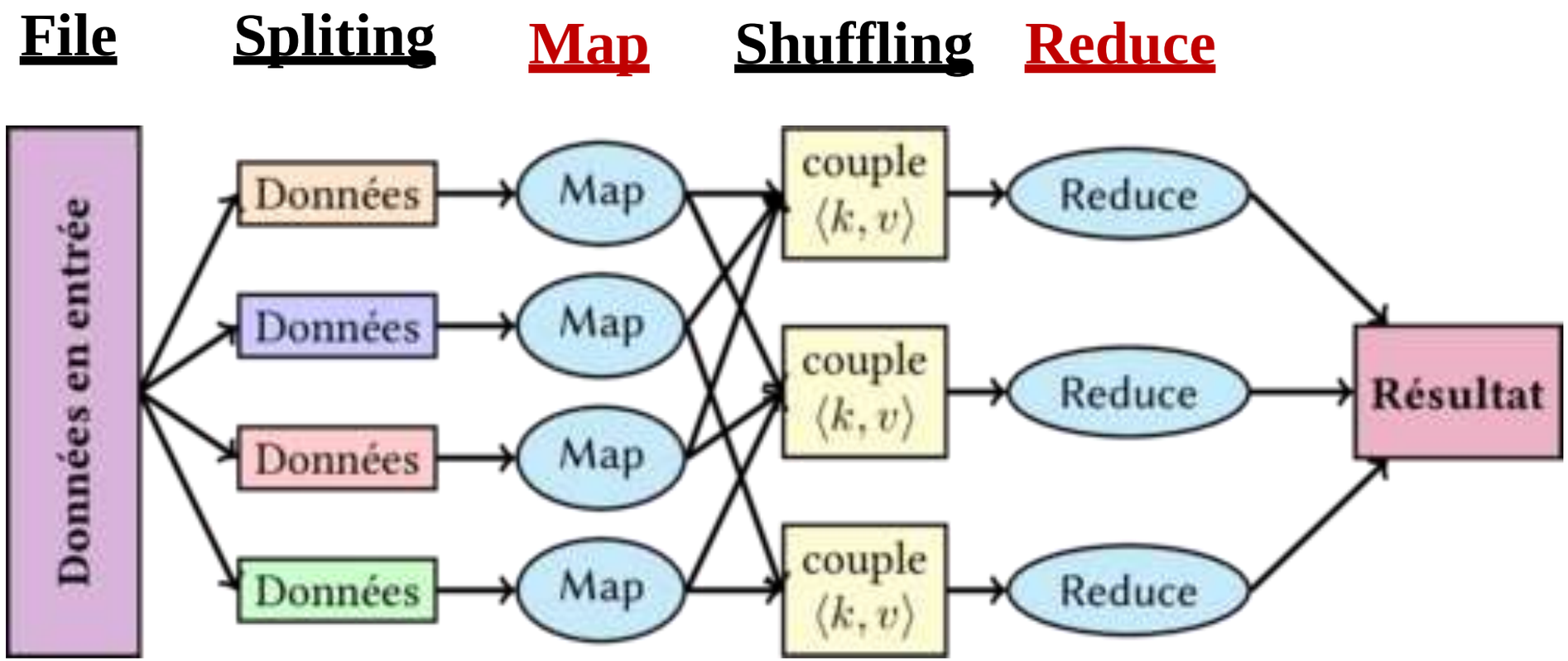
Dans le modèle de programmation MapReduce, le développeur implémente deux fonctions principales:

– **La fonction Map** : extraction/calcul d'une information sur chaque n-uplet,

– **La fonction Reduce** : regroupement de ces informations.

Fondement 2 : MapReduce

L'algorithme *MapReduce* s'exécute en 5 phases :



Fondement 2 : MapReduce

1. **L'étape File** : lire le fichier document en entrée et initialiser les différents « Workers MapReduce »
2. **L'étape Splitting** : distribuer les données à traiter sur les différents noeuds du cluster de traitement
3. **L'étape Map** : effectuer le calcul demandé en local sur chaque nœud du cluster. Cette fonction opère sur la clé du split ainsi que sa valeur pour retourner à la fin du traitement une liste des pairs {clé,valeur}.

$$Map(key_{in},val_{in}) = \{ (key_1,value_1), \ldots, (key_n,value_n)\}$$

On peut avoir des clés identiques dans cette liste

Fondement 2 : MapReduce

4. **L'étape Shuffling** : phase intermédiaire qui permet de transférer les outputs des mappeurs aux réducteurs après avoir les trier par clé.

$$Shuffle((key_1,value_1), \ldots, (key_n,value_n)) = \{(key_1,\{value_1, \ldots,value_p\}), \ldots, (key_m,\{value_1, \ldots,value_b\})\}$$

5. **L'étape Reduce** : calcule une valeur à partir de la liste des valeurs associées à chaque clé pour les regrouper et retourner le résultat finale.

$$Reduce((key_1,\{value_1, \ldots,value_p\}) = \{(key_1,value_{out\_1})$$

Fondement 2 : MapReduce

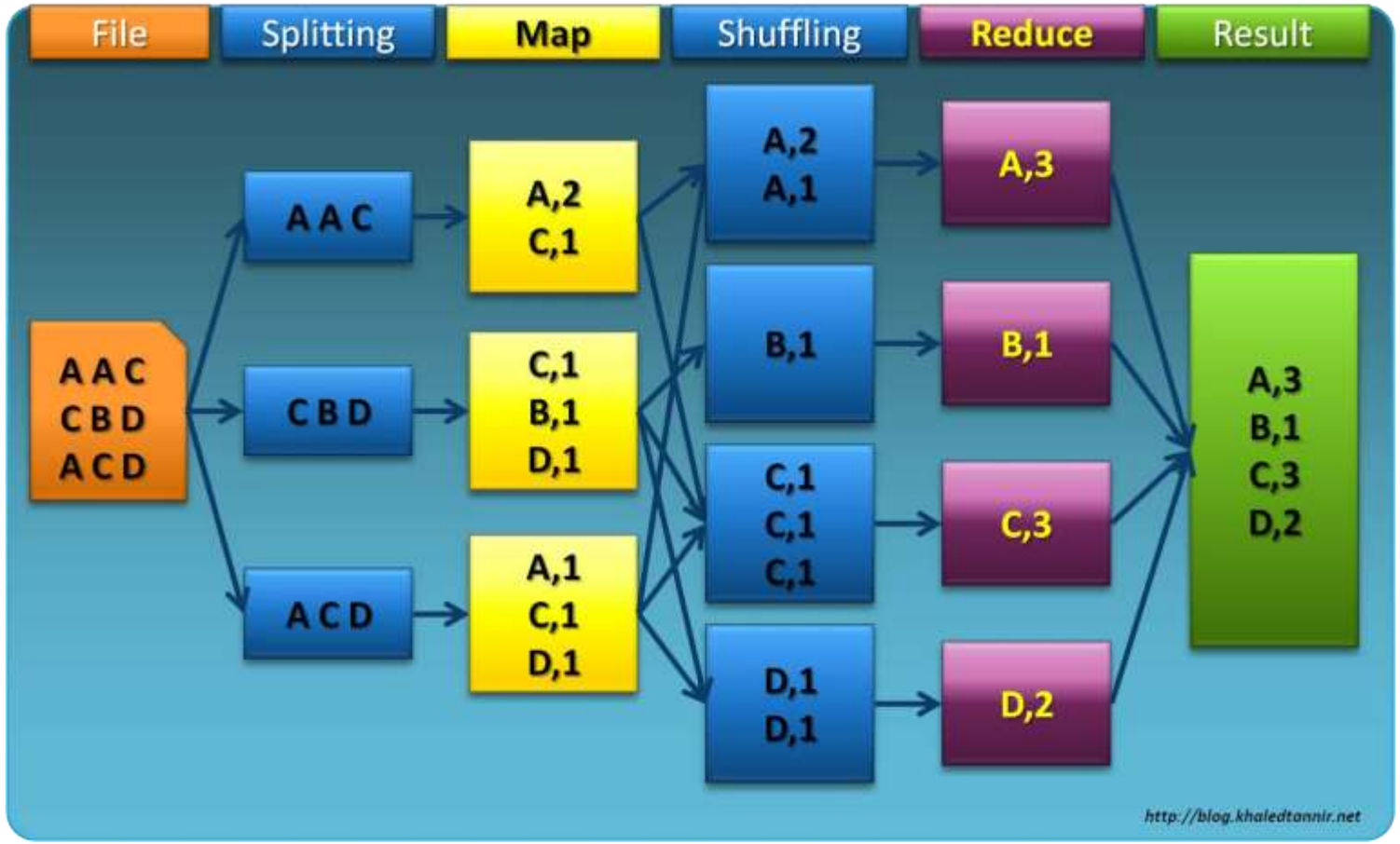
Exemple d'utilisation :

Soit un fichier document contenant 3 lignes composée chacune de 3 mots parmi les mots {A, B, C, D}. Il s'agit de compter tous les mots contenus dans ce fichier.

Le traitement MapReduce effectué pour compter les mots du document est :

Fondement 2 : MapReduce

Exemple d'utilisation :



Fondement 2 : MapReduce

Exemple d'utilisation :

Les étapes réalisées sont :

1. **L'étape File :** Lire du fichier d'entrée
2. **L'étape Splitting** : distribuer les données à traiter sur les différents nœuds du cluster.
3. **L'étape Map** : on effectue le compte de chacune des lettres et ceci en local sur chaque nœud du cluster de traitement

Fondement 2 : MapReduce

Exemple d'utilisation :

Les étapes réalisées sont :

4. **L'étape Suffling** : on regroupe toutes les lettres ainsi que leur compte à partir de tous les noeuds de traitement
5. **L'étape Reduce**: on effectue le cumule de toutes les valeurs de chaque lettre
6. **L'étape Result** : on agrège tous les résultats des différentes étapes Reduce et on retourne le résultat final

Fondement 3 : MVCC

MVCC ⇔ Contrôle de Concurrence Multi-Version

– Méthode de **contrôle de concurrence** couramment utilisée par les SGBD pour gérer des accès simultanés à la base de données avec mises à jour.

– La gestion des mises à jour des données s'effectuent non par écrasement des anciennes données par les nouvelles mais en indiquant que les anciennes données sont obsolètes et en ajoutant une nouvelle version (il existe **plusieurs version enregistrées**, seule la plus récente étant correcte)

– Nécessite généralement une **purge régulière**.

Fondement 3 : MVCC

MVCC ⇔ Contrôle de Concurrence Multi-Version

– les écritures peuvent être virtuellement isolées des opérations de lecture qui s'effectuent sur les anciennes versions dans la base et qui ont été générées par copie et maintenues tant que la transaction est vivante.

– MVCC permet à chaque utilisateur connecté de voir une capture de la base. Les modifications apportées ne seront pas visibles par les autres utilisateurs avant que la transaction ne soit validée (commit).

Fondement 3 : MVCC

MVCC ⇔ Contrôle de Concurrence Multi-Version

– En cas de **mise à jour concurrente**, la transaction qui est la première à valider les modifications, gagne; avec une annulation forcée aux autre transaction.

– **L'avantage du MVCC** est que les lectures ne sont jamais bloquées, et les écritures ne se bloquent pas entre elles.

=> Il y a donc globalement peu de temps d'attente, donc une meilleure concurrence d'accès à la base.

Fondement 3 : MVCC

MVCC ⇔ Contrôle de Concurrence Multi-Version

Les inconvénients de ce système sont :

– L'abandon de certaines transaction, en fin de traitement, si une opération concurrente, portant sur les mêmes informations, a validé ses modifications en premier.

– Le coût du maintien de multiples versions des objets en base (*surtout dans le cas de bases de données avec de nombreuses transactions fortement concurrentielles*).

– La maintenance éventuelle des versions obsolètes à purger.

Typologie des BD NoSQL

Type de BD NoSQL

Les types de base de données NoSQL sont:

- Les bases orientées **clé-valeur**
- Les bases orientées **documents**
- Les bases orientées **colonnes**
- Les bases orientées **graphes**

Type de BD NoSQL : clé-valeur

– Elle se caractérise par un **modèle** basique et le plus simple des différents types de BD NoSQL.

– Elles fonctionnent comme un **grand tableau associatif** ou une **énorme table de hachage** distribuée sur le réseau. Tout repose sur le couple Clé/Valeur.

- **La clé :** identifie la donnée de manière unique et permet de la gérer.
- **La valeur :** contient n'importe quel type de données (numérique simple, chaîne de caractères ou un objet sérialisé).

Type de BD NoSQL : clé-valeur

– Chaque objet est identifié par une clé unique qui représente la seule manière d'interroger les données.

=> On ne peut requêter le système que par la clé, et pas sur le contenu de la valeur.

– Cette absence de **structure** ou de **typage** ont un impact important sur l'interrogation de données.

Type de BD NoSQL : clé-valeur

Statistiquement, un grand nombre applications demandent à lire des données à partir de leurs identifiants.

=> engendre le besoin de BD stockant des paires clé-valeur

key	value
1	"cours: 'SQL', prof: { tout: 'Jollois' }, outils: 'cours-SQL'"
2	"cours: 'Python', prof: { tp: ['Metivier', 'Bidot'], amphi: 'Metivier' }, outils: 'Anaconda'"
3	"cours: 'PPP', outils: ['Moodle', 'Powerpoint']"
4	"cours: 'Maths', prof: { amphi: 'Ottenwaelter', td: 'Plihon' }, outils: 'calculatrice'"
5	"responsables: { chef: 'Gey', dde: 'Ottenwaelter', projet: 'Jollois' }
6	"cours: 'Dataviz', prof: { amphi: 'El Metni', tp: 'Da In' }, complement: 'Challenge Dataviz STID'"

Type de BD NoSQL : clé-valeur

Leur exploitation est basée sur 4 opérations (CRUD):

- **C**reate : créer un nouvel objet avec sa clé
  => **create(key, value)**
- **R**ead : lit un objet à partir de sa clé => **read(key)**
- **U**pdate : met à jour la valeur d'un objet à partir de sa clé
  => **update(key, value)**
- **D**elete: supprime un objet à partir de sa clé
  => **delete(key)**

Type de BD NoSQL : clé-valeur

Forces :

✓ **Interface de requête très simple** et souvent accessible facilement.

✓ **Scalabilité horizontale performante et très facile** : lors de l'ajout d'un nouveau nœud, il suffira de redéfinir les intervalles des clés dont les serveurs du cluster sont responsables.

✓ **Opérations de lecture et écriture très performantes**.

✓ **Répartition des données aisée**.

Type de BD NoSQL : clé-valeur

Faiblesses :

– Modèle de données **TROP** simple : **pauvre pour les données complexes**

– **Interrogation de données par clé seulement** (médiocre).

– **On ne peut agir que sur la valeur dans son entier:** on ne peut pas réaliser d'opérations CRUD sur une partie de la valeur.

Type de BD NoSQL : clé-valeur

– **Exemple de BD clé-Valeur**

- **Redis :** projet sponsorisé par *VMWare*
- **Voldemort :** développé par Linkedln en interne puis passage en open source.
- **Amazon Dynamo : Riak** est son implémentation Open Source.

Type de BD NoSQL : Document

– Elles stockent une collection de "**documents**" identifié par une clé unique.

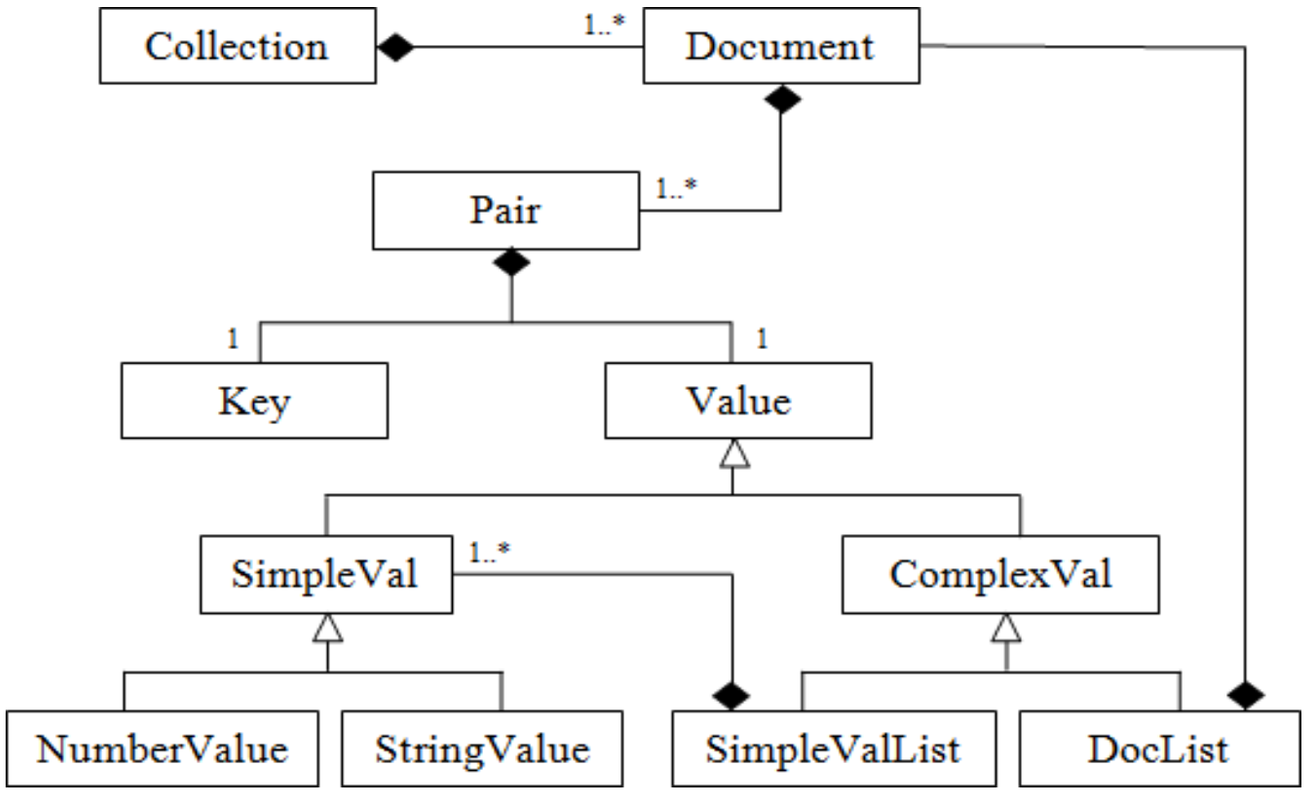
– Elles sont basées sur le modèle « clé-valeur » mais la **valeur** est un document en format semi-structuré hiérarchique de type JSON ou XML.

– Les **documents** n'ont pas de schéma, mais une structure arborescente : ils contiennent une liste de champs.

– Un champ a une valeur qui peut être de type **simple** (*entier, chaine de caractère, date*, ...) ou **complexe** (*plusieurs couples clé/valeur*).

Type de BD NoSQL : Document

⇒ **Modèle d'une BD NoSQL avec** JSON**:**




```json
{
  "Time": "2017-12-26T22:39:08",
  "Uptime": 4,
  "Vcc": 3.25,
  "POWER": "ON",
  "Wifi": {
    "AP": 1,
    "SSId": "JAZZTEL_david",
    "RSSI": 72,
    "APMac": "20:89:86:1E:31:E2"
  }
}
```

Type de BD NoSQL : Document

– Bien que les documents soient semi-structurés, ces BD sont dites "schemaless" : il n'est pas nécessaire de définir au préalable les champs utilisés dans un document.

– Les documents peuvent être très hétérogènes au sein de la BD.

– Contrairement au modèle clé/valeur, on peut effectuer des requêtes sur le contenu des documents/objets : pas possible avec les BD clés/valeurs simples

– Elles sont principalement utilisées dans le développement de CMS (***C****ontent* ***M****anagement* ***S****ystem*).

Type de BD NoSQL : Document

Forces :

- ✓ Modèle de données simple mais puissant
- ✓ Mise à jour possible d'un document.
- ✓ Forte expressivité de requêtage (requêtes assez complexes sur des structures imbriquées)
- ✓ Permet de structurer la valeur à stocker
- ✓ Possibilité d'effectuer des requêtes sur le contenu du document.
- ✓ Stocker d'importants volumes de données distribuées et répliquées.

Type de BD NoSQL : Document

Faiblesses :

– Inadaptée pour les données interconnectées

– Lenteur éventuelle sur les données complexe.

– La forte duplication des données et la difficulté à gérer la non-cohérence de ces dernières.

Type de BD NoSQL : Document

– Exemples de BD orientée document

- MongoDB
- CouchDB
- DynamoDB

Type de BD NoSQL : Colonne

– Traditionnellement, les données sont représentées en ligne avec un ensemble des attributs.

– Le stockage **orienté colonne** change ce paradigme en se focalisant sur chaque attribut et en les distribuant.

=> Possibilité de focaliser les requêtes sur une ou plusieurs colonnes, sans avoir à traiter les informations inutiles (les autres colonnes)

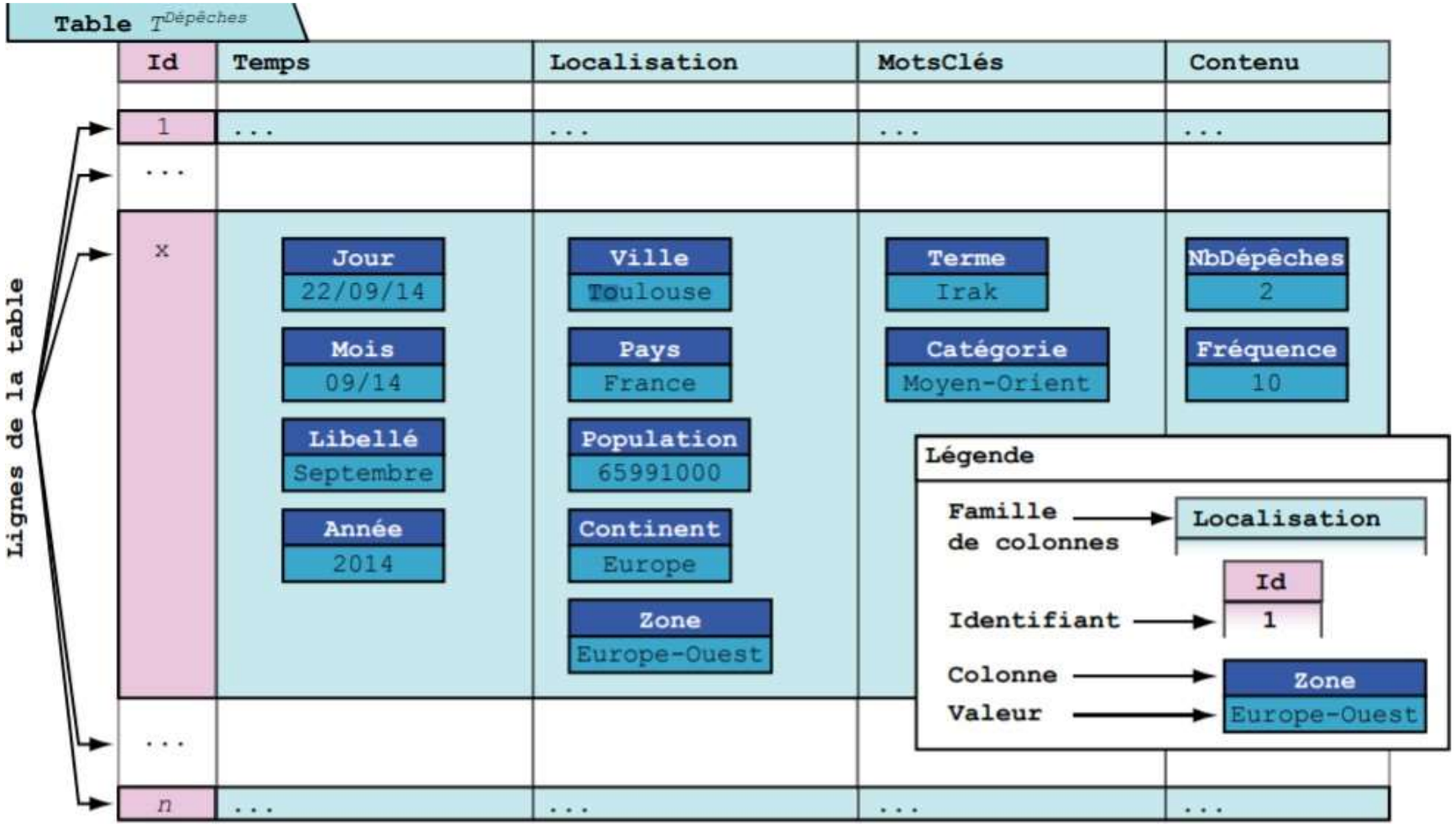
Type de BD NoSQL : Colonne

– Les bases orientées colonnes fonctionnent par familles de colonnes. Elles représentent l'équivalent de sous-tables.

– Chaque ligne d'une "table" comporte **exactement** le nombre de colonnes qu'elle a besoin pour stocker ses valeurs.

=> Pas de NULL stockés dans des enregistrements vides comme c'était le cas dans les SGBDR (à noter qu'un NULL occupe quand même de l'espace mémoire).

Type de BD NoSQL : Colonne



Remarque: On considère que la famille de colonne existe si elle est non vide

Type de BD NoSQL : Colonne

Concepts de base:

1. Colonne :

- entité de base représentant un champ de donnée
- chaque colonne est définie par un couple clé/valeur

Type de BD NoSQL : Colonne

Concepts de base:

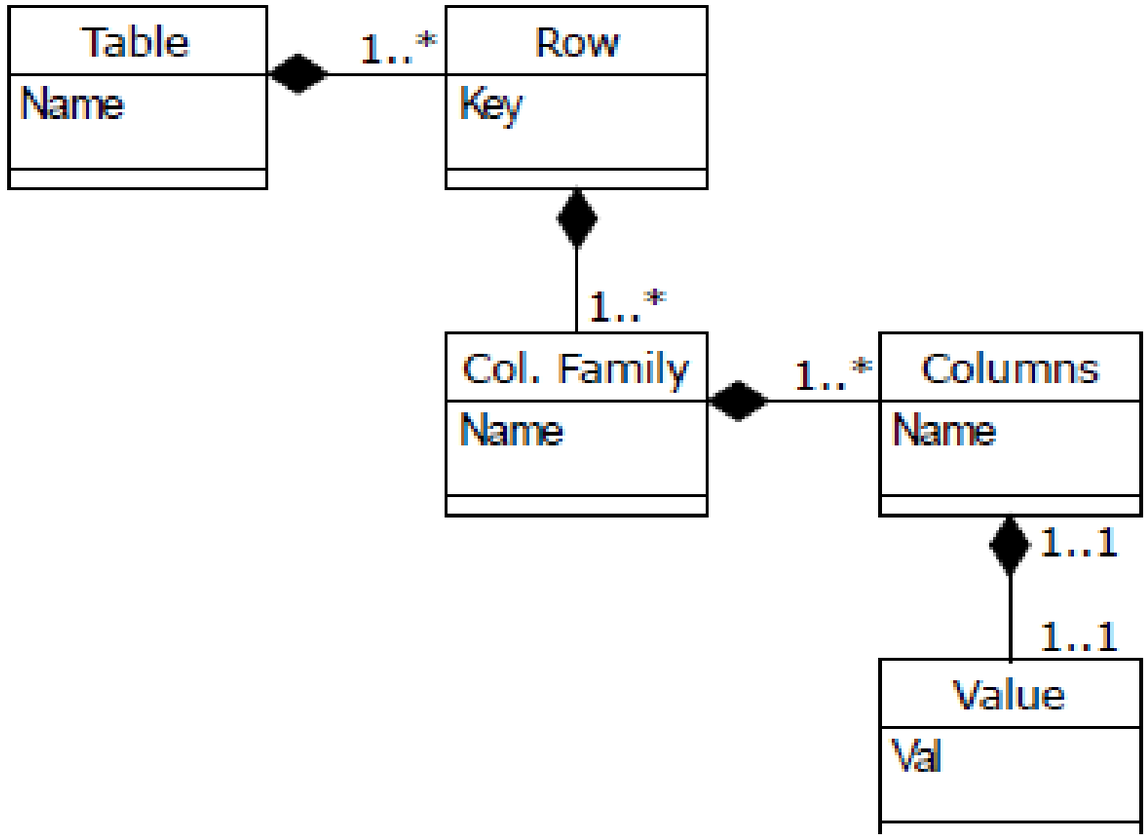
2. Famille de colonnes :

- permettent de regrouper plusieurs colonnes
- les colonnes sont regroupées par ligne
- chaque ligne est identifiée par un identifiant unique

(*assimilées aux tables dans le modèle relationnel*)

Type de BD NoSQL : Colonne

Modèle de base:



Type de BD NoSQL : Colonne

Forces:

- ✓ Modèle proche du relationnel
- ✓ Modèle de données simple mais puissant (expression de structures imbriquées)
- ✓ Très grandes flexibilité

Faiblesses :

- ❖ N'est pas adapté aux données interconnectées ou complexes (de type hiérarchique)

Type de BD NoSQL : Colonne

– **Exemple de BD orientée colonne**

- **BigTable** (Google)
- **HBase :** Open Source de BigTable de Google utilisé pour l'indexation des pages web, Google Earth, Google analytics, ...
- **SimpleDB** (Amazon)
- **Elasticsearch**

Type de BD NoSQL : Graphe

Les trois premières familles NoSQL n'adressent pas le problème de corrélations entre les éléments.

⇒ Les BD orienté graphe **offrent une réponse adéquate à un modèle de données** très volumineux et fortement connecté.

⇒ La modélisation d'un problème avec un graphe est très intuitive et correspond à la réalité métier. **Ce type de base est très flexible** car elle n'impose pas de schéma de données figées.

Type de BD NoSQL : Graphe

– Les données sont représentées sous forme de **graphe.**

– S'appuie sur les notions de **nœuds**, de **relations** et de **propriétés** qui leur sont rattachées (*chaque propriété est représentée par une paire clé/valeur*).

1. **Nœuds :** représentent les entités

Chaque **nœud** du graphe se caractérise par :

a. Identifiant,
b. Ensemble de label décrivant sa sémantique,
c. Ensemble de ses propriétés.

Type de BD NoSQL : Graphe

2. **Relations :** *(sous forme d'arcs)* définissent les relations entre les nœuds connectés.

   Chaque relation du graphe se caractérise par :

   a. Identifiant,

   b. Label (nom),

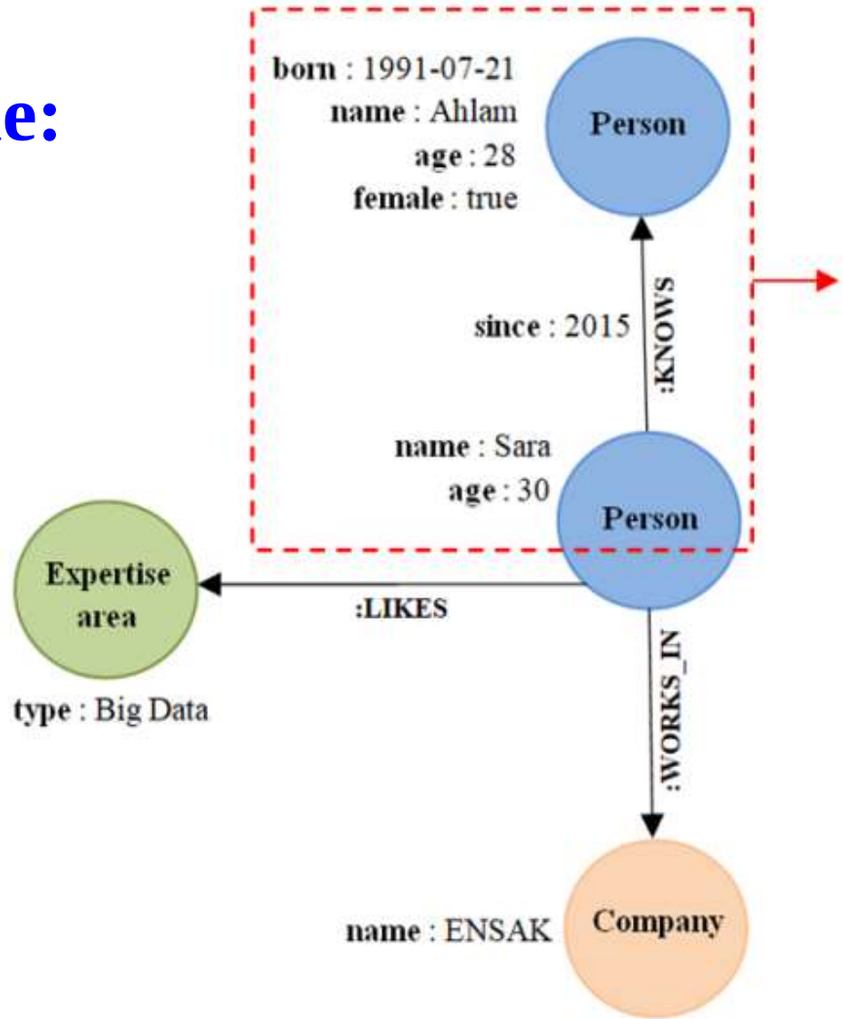
   c. Identifiant du nœud de départ,

   d. Identifiant du nœud cible,

   e. Ensemble de ses propriétés.

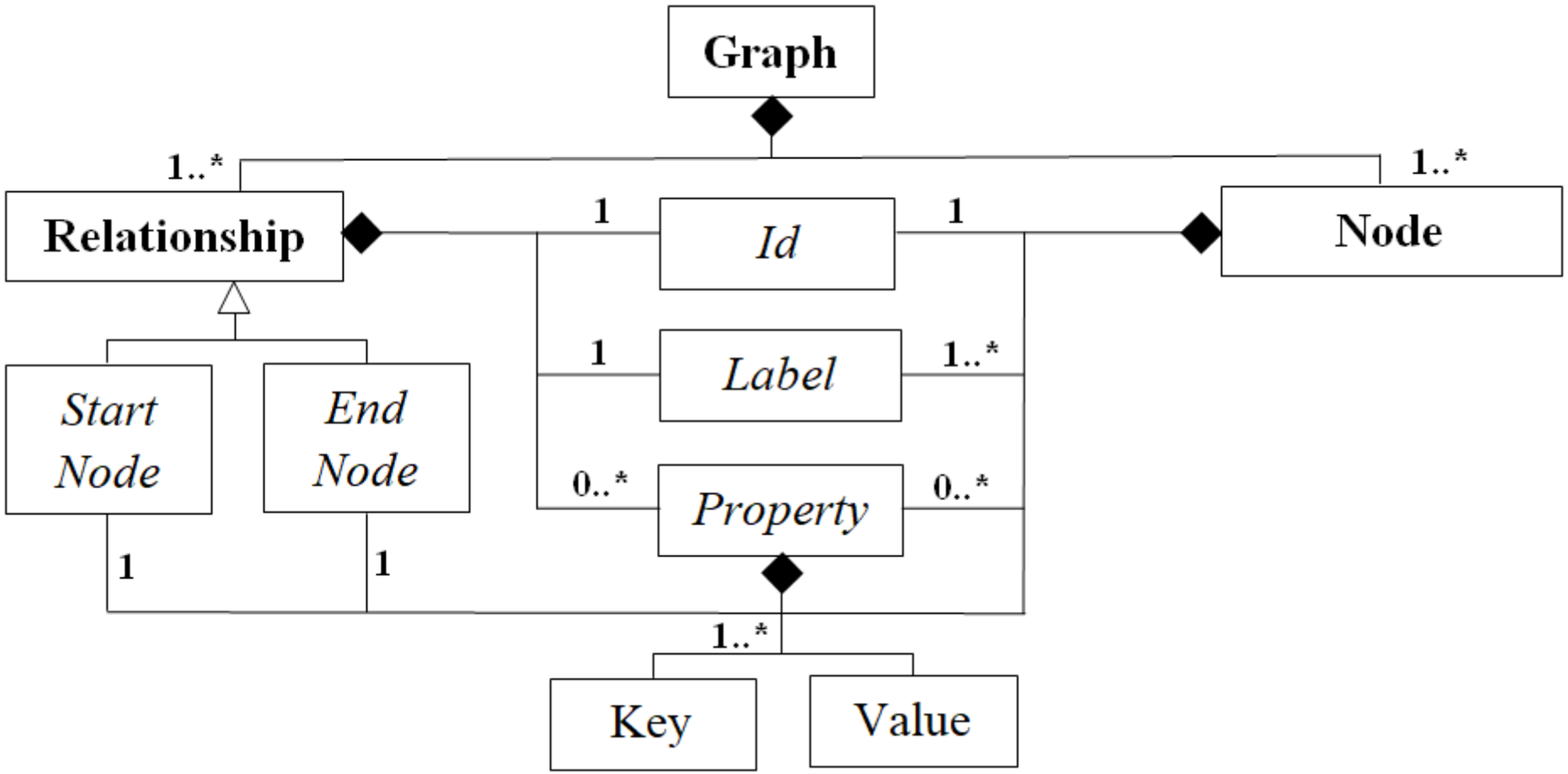
Type de BD NoSQL : Graphe

Exemple:




```
...
{"type":"node",
 "id":"0",
 "labels":["Person"],
 "properties":{
        "name":"Sara",
        "age":35}
}
{"type":"node",
 "id":"1",
 "labels":["Person"],
 "properties":{
        "born":"1991-07-21",
        "name":"Ahlam",
        "age":28,
        "female":true}
}
{"id":"0",
 "type":"relationship",
 "label":"KNOWS",
 "properties":{"since":2015},
 "start":{"id":"0","labels":["Person"]},
 "end":{"id":"1","labels":["Person"]}
}
...
```

Type de BD NoSQL : Graphe



Type de BD NoSQL : Graphe

Forces :

- ✓ Modèle de données puissant
- ✓ Très efficace pour les données liées
- ✓ Exécuter des algorithmes de graphes régulièrement utilisés dans un modèle de données fortement connecté.

Faiblesses :

- ❖ Partitionnement (sharding) compliqué
- ❖ Parcours moins efficace si le graphe est très connecté

Cas d'utilisations des SGBD NoSQL

Cas d’utilisations : clé-valeur

Elles sont parfaitement adaptées à:

- Dépôt de données avec besoins de requêtage très simples
- Les profils, préférences d’utilisateur
- Les données de panier d’achat
- Les données de capteur
- Les logs de données
- …

Cas d’utilisations : clé-valeur

– Ce type de bases de données est un choix judicieux lorsque les données sont **codées de multiples façons sans schéma rigoureux**.

– Elles ne sont pas spécialement adaptées **à la gestion des relations complexes** entre différents ensembles de données ni à **l'interrogation au moyen de tout élément autre que la clé définie**.

Cas d’utilisations : Document

– Ce type de BD est parfaitement adapté au stockage de différents types de données pour chaque document, grâce à la flexibilité qu'il offre en matière de **recherches à l'échelle de toutes les données**.

– Elles constituent un choix judicieux lorsque le schéma n'est pas rigide, mais qu'il est tout de même nécessaire d'**effectuer des requêtes en utilisant un élément autre qu'une simple clé**.

Cas d'utilisations : Document

La flexibilité de leurs schéma les rend parfaitement adaptées à:

- **Gestion de sessions**,
- **Systèmes de gestion de contenu** (bibliothèques numériques, collections de produits, dépôts de logiciels, collections multimédia, etc.),
- **Gestion des historiques d'utilisateurs** sur réseaux sociaux.
- …

Cas d'utilisations : Document

Inconvénients :

– Elles ne sont pas spécialement adaptées au traitement de **transactions complexes**.

– Leurs utilisation n'est pas idéale non plus pour les applications qui recourent à **l'agrégation de données**.

⇒ leur schéma flexible signifie que les données ne sont pas homogènes sur tous les documents et ont, par conséquent, **peu de chances d'être correctement agrégées**.

Cas d’utilisations : Colonnes

– Elles sont efficaces pour les systèmes :

- Affichant **peu d'opérations en écriture**,
- Dans lesquels un faible nombre de colonnes présentant de nombreuses lignes doivent être lues fréquemment et simultanément.

– Ce type de stockage est adapté à la **journalisation d'événements**, à la **gestion de contenu.**

Cas d'utilisations : Graphe

– Elles sont adaptées aux éléments de **données interconnectés** entre eux avec un nombre indéterminé de relations.

⇒ La mise en œuvre de réseaux sociaux (Ex: LinkedIn ou Facebook) représente le cas d'utilisation le plus courant des bases de données orientées graphes.

– Les bases de données orientées graphes sont également pratiques pour la prise en charge de *moteurs de recommandations*, comme ceux utilisés par les sites de vente de détail en ligne.

Cas d'utilisations : Graphe

– D'autres applications sont évidemment possibles, comme l'acheminement et l'expédition des livraisons, les réseaux des transports publics, les cartes routières, les topologies réseau, …

– Elles **ne sont pas adaptées** aux données qui se changent fréquemment, ni aux mises à jour en temps réel de grands volumes de données.

– **Si vous prévoyez de partitionner** la base de données sur un réseau, ce type de base de données subira probablement une dégradation de ses performances.