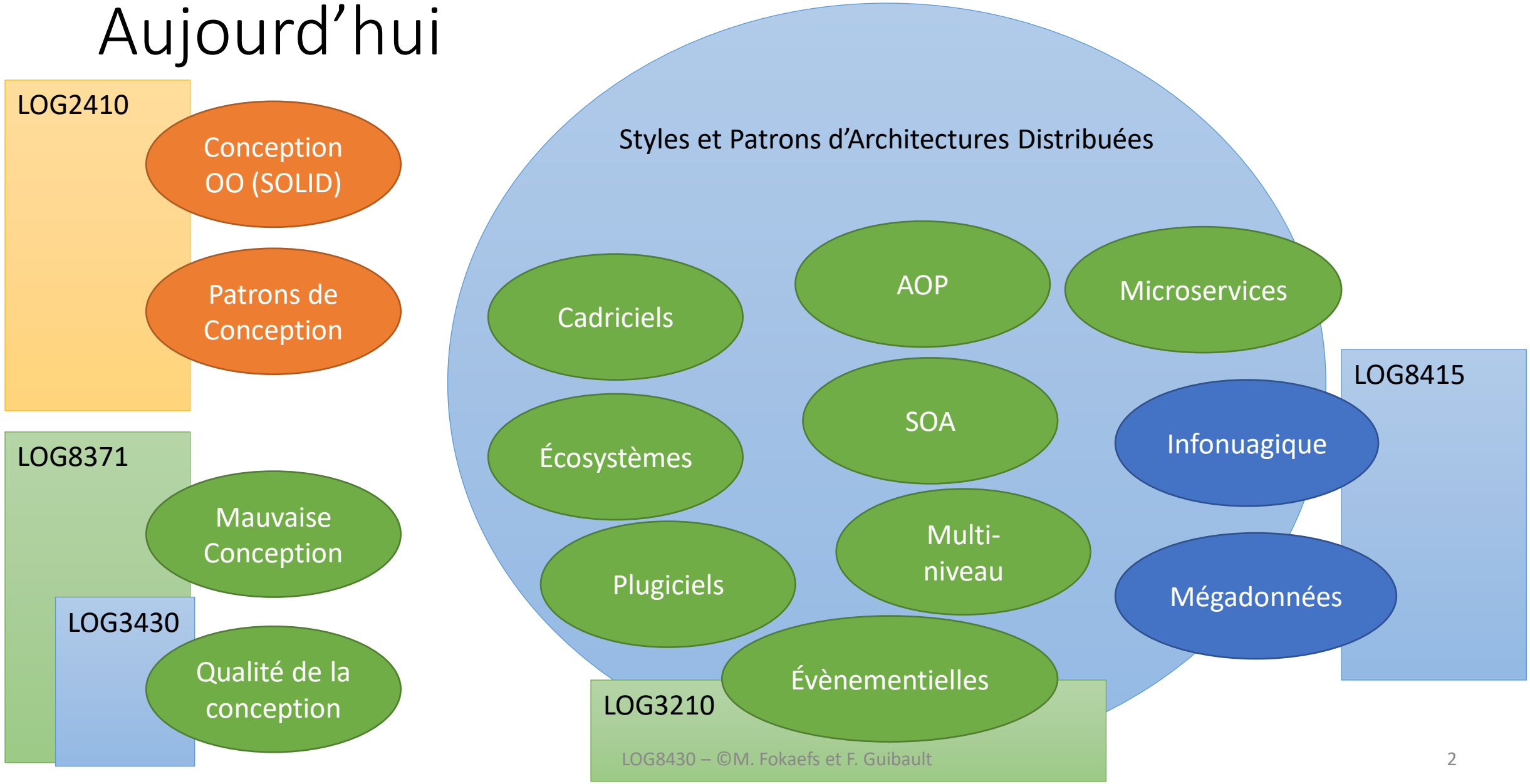
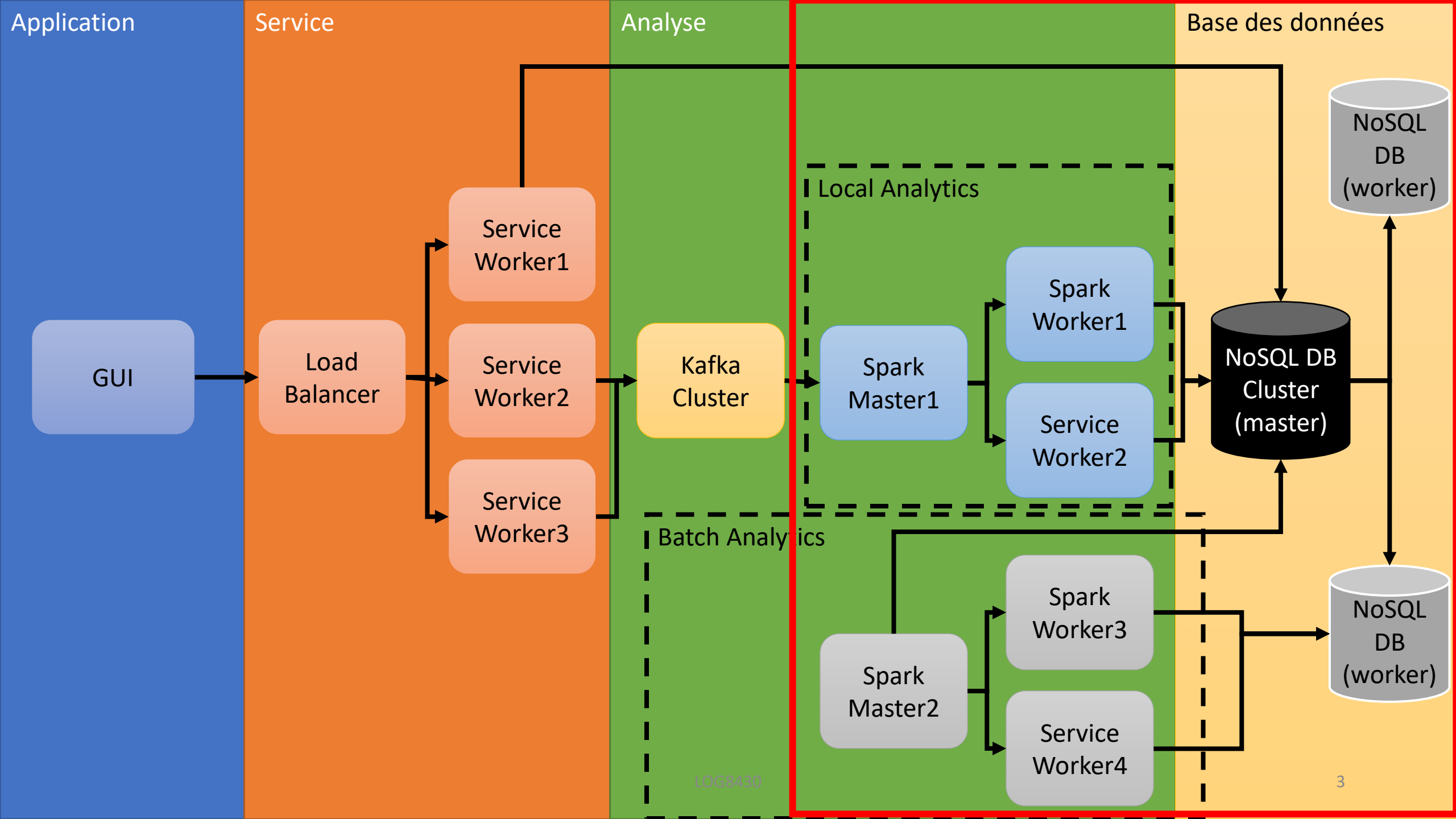
LOG8430 : Architectures des Megadonnées

Architectures et systèmes des bases de données, et architectures du traitement des données

Aujourd'hui


LOG2410
Conception OO (SOLID)
Patrons de Conception
LOG8371
Mauvaise Conception
LOG3430
Qualité de la conception
Styles et Patrons d'Architectures Distribuées
Cadriciels
AOP
Microservices
Écosystèmes
SOA
Infonuagique
Plugiciels
Multi-niveau
Mégadonnées
Évènementielles
LOG3210
LOG8415


Application
Service
Analyse
Base des données
GUI
Load Balancer
Service Worker1
Service Worker2
Service Worker3
Kafka Cluster
Local Analytics
Spark Master1
Spark Worker1
Service Worker2
NoSQL DB Cluster (master)
NoSQL DB (worker)
Batch Analytics
Spark Master2
Spark Worker3
Service Worker4
NoSQL DB (worker)

Architectures logiciels des données

- NoSQL
- Bases de données distribuées
- La théorème CAP
- Propriétés BASE et ACID

- Wide-column
- Document-based
- Key-value
- Graph

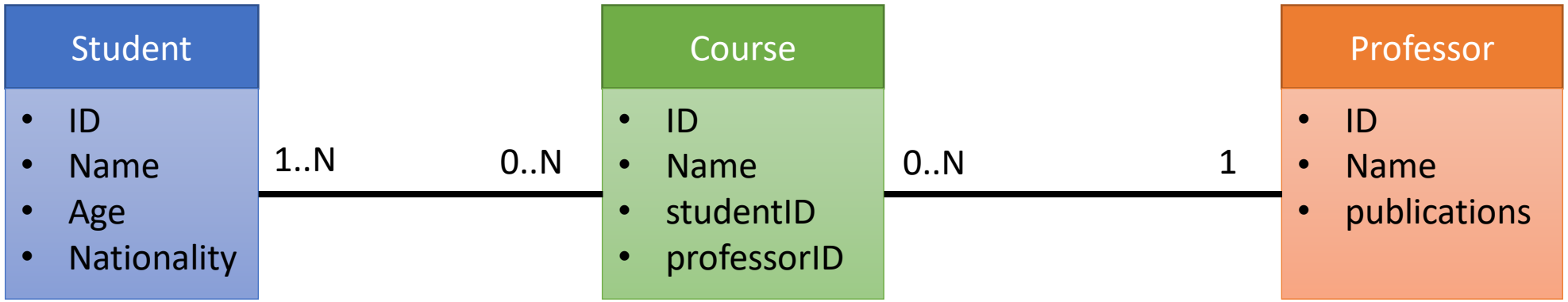
- Architecture Lambda
- Architecture Kappa

Question

- On veut organiser une université. On a des étudiants avec leurs attributs (ID, nom, âge, nationalité etc.). On a aussi des professeurs avec leurs attributs (ID, nom, publications, etc.). Finalement, on a des cours qui sont suivis par les étudiants et sont enseignés par les professeurs.
- Concevez un système ou un modèle de données (un diagramme E-R).

Question

- On veut organiser une université. On a des étudiants avec leurs attributs (ID, nom, âge, nationalité etc.). On a aussi des professeurs avec leurs attributs (ID, nom, publications, etc.). Finalement, on a des cours qui sont suivis par les étudiants et sont enseignés par les professeurs.
- Concevez un système ou un modèle de données (un diagramme E-R).



Problème

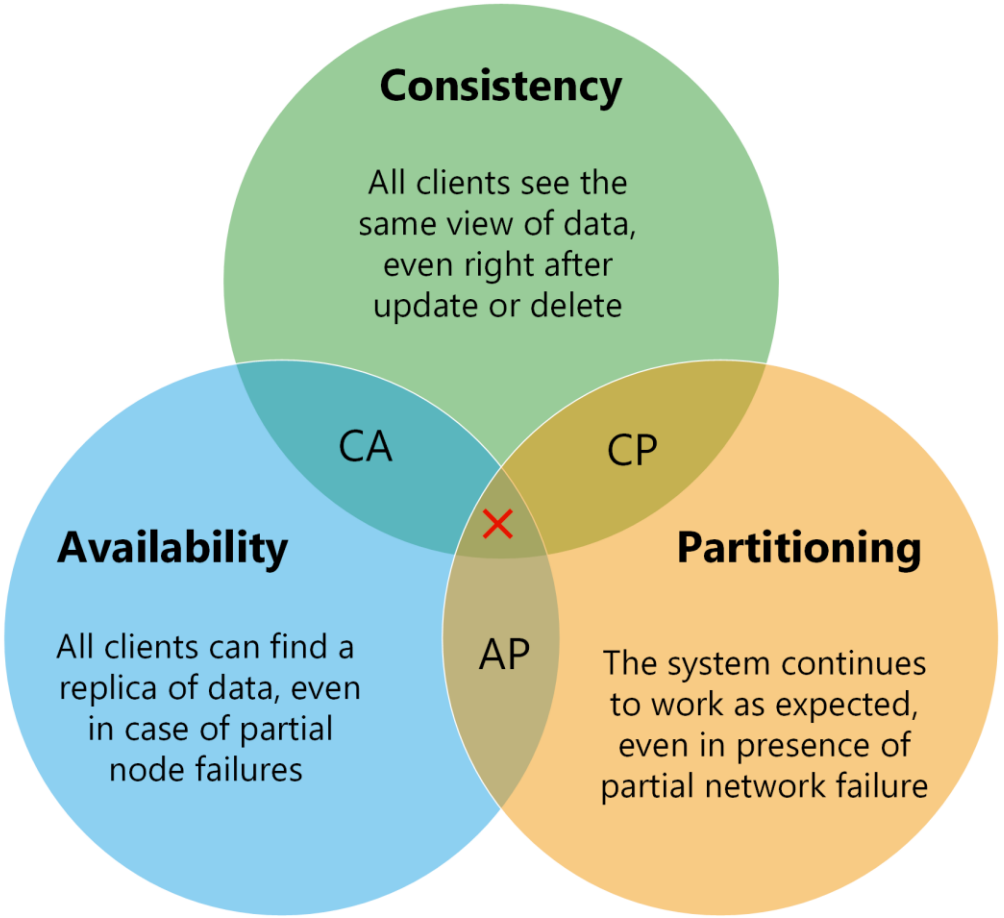
- Maintenant, ne parlons pas d'étudiants et de professeurs, mais parlons d'étoiles, de planètes et de systèmes planétaires.
- Il n'y a pas que des centaines d'objets, mais des milliards.
- Aussi, on a plus de types d'objets et des relations plus complexes.
- En explorant l'univers, nous en découvrons plus.
- Est-il suffisant d'utiliser une base de données relationnelle (RDB)?
- On a besoin d'une infrastructure plus puissante pour supporter le volume et la complexité des données et plus flexible pour accommoder les nouveaux types de données.

Bases de données distribuées

- L'objectif primaire des bases de données distribuées est la gestion de grands volumes de données en garantissant une disponibilité (presque) absolue.
- Pour cette raison, le système est installé sur plusieurs serveurs et les données sont partitionnées et partagées parmi les nœuds.
- Le système utilise la duplication, la réplication et le partitionnement des données.
  - La réplication est utilisée pour mettre à jour les données sur tous les nœuds. C'est une procédure longue et complexe.
  - Pour la duplication, il y un master qui contient la base de données. Ce master duplique la base et il s'assure que les autres nœuds ont les données. Les utilisateurs ne peuvent changer que le master.
  - Les partitions existent entre plusieurs nœuds, mais les relations entre-elles sont bien définies. Alors, il est possible de reconstruire la base entière.
- La distribution et les transactions sont transparentes aux utilisateurs.

Le théorème CAP

- En faveur de la disponibilité et de la performance, les systèmes distribués sacrifient souvent la cohérence.
- **C**onsistency : Tous les clients voient la même vue des données, même après une mise à jour.
- **A**vailability : Tous les clients peuvent obtenir une copie des données.
- **P**artition Tolerance : Le système continue à fonctionner, même après l'échec d'un nœud.
- La tolérance au partitionnement est la propriété de base des systèmes distribués, alors elle est garanties par définition.
- Alors on doit prendre une décision entre la cohérence et la disponibilité.
- …ou peut-être pas… On va voir!



ACID vs BASE

- Non, ce n’est pas un cours de chimie!
- Les systèmes traditionnels (relationnels) suivent les propriétés ACID pour leurs transactions :
  - **Atomique** : Toutes les opérations d'une transaction réussissent ou l’ensemble de la transaction est annulée.
  - **Cohérente** : À la fin d'une transaction, la base de données est structurellement correcte.
  - **Isolée** : Les transactions ne se font pas concurrence. Les accès litigieux aux données sont arbitrées par la base de données, de sorte que les transactions semblent s'exécuter de manière séquentielle.
  - **Durable** : Les résultats de l'application d'une transaction sont permanents, même en cas d'échec.
- Les systèmes distribués peuvent suivre les propriétés ACID, mais ils suivent plus souvent les propriétés BASE :
  - **Basic Availability** : La base de données semble fonctionner la plupart du temps.
  - **Soft-state** : Les bases n'ont pas besoin d'être cohérentes en écriture, pas plus que les différentes répliques doivent être cohérentes les unes avec les autres.
  - **Eventual Consistency** : Les bases présentent une cohérence à un moment ultérieur (par exemple, paresseusement au moment de la lecture).

Implémentations des bases distribuées : NoSQL

- Les bases de données NoSQL sont devenu très populaires dans l'ère des mégadonnées.
- Elles n'exposent ni interfaces SQL ni interfaces normalisées.
  - Cela augmente l'efficacité et la flexibilité dans la définition des requêtes.
- Elles utilisent aussi plusieurs types d'organisation de données.
  - Cela augmente beaucoup l'efficacité du système.
- Mais n'est-ce pas possible d'avoir des bases relationnelles et distribuées?
  - Oui il existe plusieurs tels systèmes : MySQL Cluster, IBM Db2
  - Ils offrent des avantages de performance, mais ils sont différents des systèmes NoSQL.

NoSQL : Types d’organisation des données

- Key-value
- Document
- Graph
- Wide Column

Key-value

- Les données sont organisées en pairs de clés et de valeurs.
- Les valeurs suivent les notions d'objets, comme dans le cas de la programmation orientée objet.
  - Elles peuvent être assez complexes.
- La différence principale entre KV et RDB et que le premier n'a pas un schéma qui définit la structure des valeurs.
- Alors on peut avoir des attributs différents ou des attributs manquants entre les objets du même type.
  - Cela offre une flexibilité augmentée.
- Les données peuvent être partitionnées et partagées parmi les nœuds en utilisant les clés.

KV : Propriétés, avantages, implémentations

- Ces systèmes sont très efficaces pour des applications simples.
- Il est possible de stocker toutes les données en mémoire, ce qui augmente l'efficacité.
  - Il est encore possible d'utiliser un stockage persistent, c.-à-d. le disque.
  - Il est aussi possible d'utiliser une méthode hybride.
- Ils peuvent supporter des données complexes en offrant une efficacité dans la cherche des valeurs basée sur les clés.
- Implémentations
  - BerkeleyDB
  - Redis
  - Memcached
  - Riak
  - Voldemort

Document

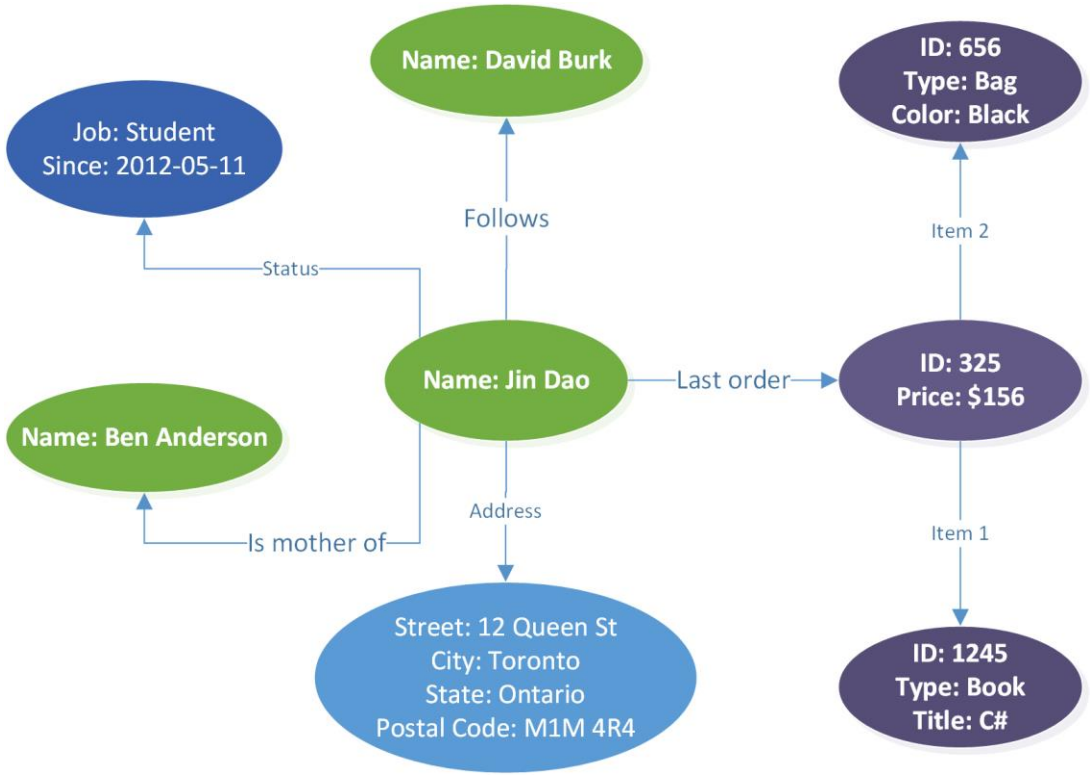
- Ces systèmes traitent les données en tant que fichiers.
- Les fichiers ont un encodage spécifique, p.ex. JSON, XML etc.
- On peut demander des documents à l'aide de clés.
- Les documents peuvent être de divers tailles et complexités.

Document : Propriétés, avantages, implémentations

- Ces systèmes sont appropriés pour les applications où les données sont des documents (bien sûr!), mais aussi lorsqu'on a des données ayant une structure variable.
- Contrairement au KV, où les valeurs sont opaques, ici on peut demander des documents en fonction de leurs attributs.
- On peut aussi définir des métadonnées pour faciliter la récupération des fichiers.
- Dans ce cas, les clés sont les indices primaires et on peut définir des indices secondaires sous forme de valeurs des attributs ou des métadonnées.
- Implémentations :
  - CouchDB
  - MongoDB
  - RavenDB

Graph

- Dans ces systèmes, les données sont représentées en tant que graphes.
- Ils répondent au besoin de combiner une modélisation graphique des données et les tableaux relationnels de RDB.
- Les nœuds représentent des entités et les arêtes des relations. Les nœuds peuvent avoir des attributs.
- Ils permettent de stocker n'importe quelle relation de façon pratique et efficace.

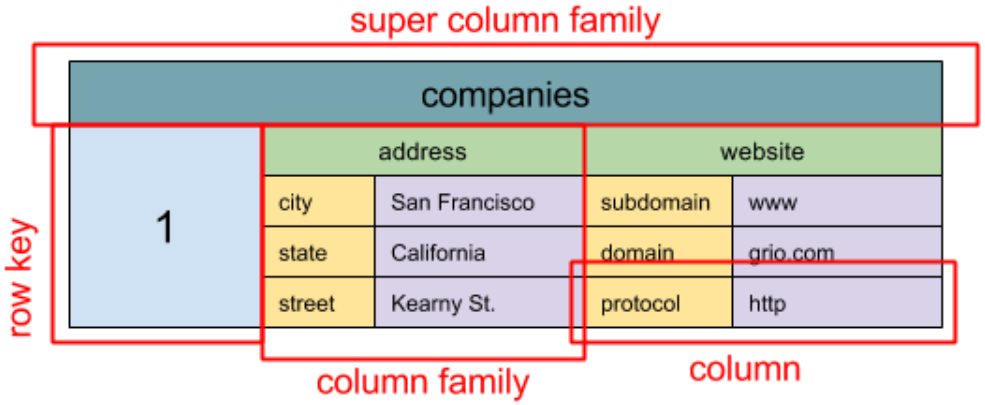


Graph : Propriétés, avantages, implémentations

- Les graphes existent partout!
- Parfois, on a besoin de stocker les données des graphes en préservant leur forme, en tant que graphe.
- Il existe plusieurs algorithmes pour traverser ou analyser les graphes qui sont déjà efficaces.
- Certaines requêtes ne sont pas possibles avec les langages traditionnels des requêtes.
- La puissance et la capacité du système dépendent de l'implémentation sous-jacente.
  - Il y a des systèmes qui sont implémentés au-dessus des RDB ou des KV.
- Implémentations
  - Neo4j
  - OrientDB
  - Titan

Wide-column

- Ce sont les systèmes les plus proches des bases de données relationnelles, au moins au niveau conceptuel.
- Ils conservent les concepts de tableaux, de rangées et de colonnes.
  - Cela donne un sens d'un schéma.
- Cependant, les concepts relationnels forment une interface pour les clients.
  - Au niveau de l'implémentation, les systèmes sont basés sur des systèmes de fichiers distribués.
- La définition des colonnes est très flexibles.
  - On groupe plusieurs attributs dans des familles de colonnes.
  - On peut omettre certaines colonnes.



Wide-column : Propriétés, avantages, implémentations

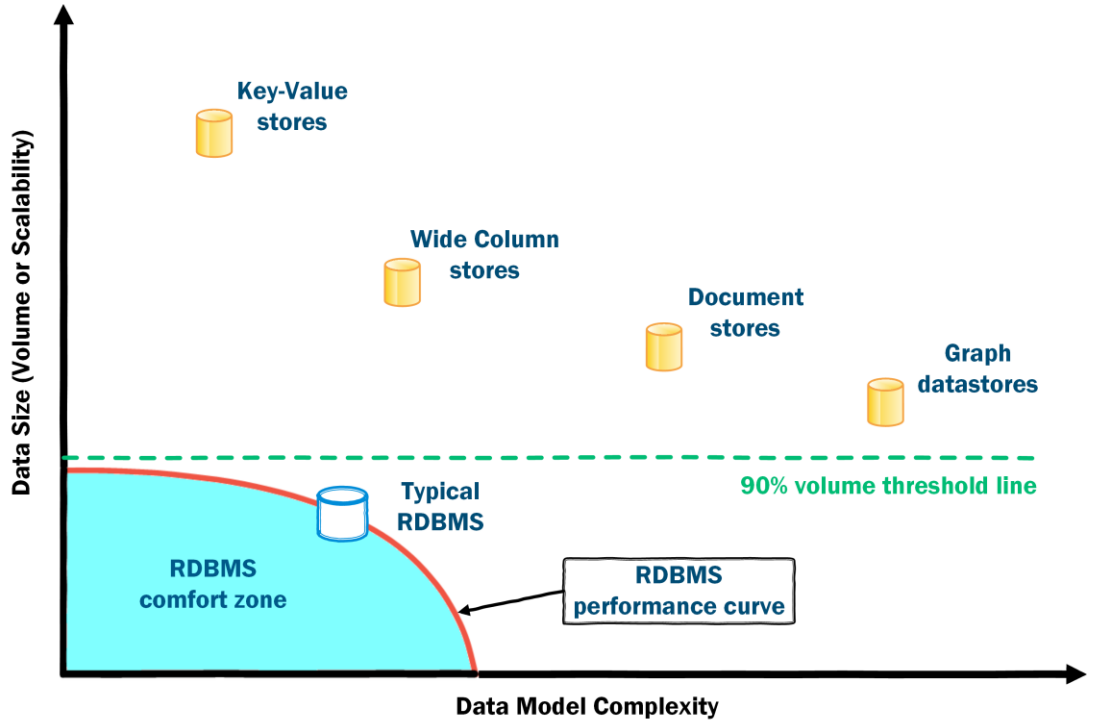
- Ce type de BD facilite la migration des bases de données relationnelles vers les systèmes distribués.
- La flexibilité supplémentaire offerte par les familles de colonnes améliore la maintenabilité des données et la gestion des données variantes.
- Ils peuvent supporter d'énormes quantités de données.
- Les systèmes wide-column sont conçus pour fonctionner avec les systèmes distribués d'analyse des mégadonnées tels que MapReduce et Hadoop. Ils sont conçus pour offrir :
  - une disponibilité et une sécurité augmentées;
  - des lectures et des écritures rapides.
- Implémentations
  - HBase
  - Accumulo
  - Cassandra

Comment va-t-on sélectionner la solution NoSQL?
Selon le type d'application

- Key-value
  - Mise en cache
  - Lorsqu'on a des données séquentielles (listes, queues etc.)
  - Pour supporter des systèmes publish/subscribe
  - Pour maintenir un état
- Document
  - Avec des données très complexes et imbriquées
  - Lorsqu'on a des clients en JavaScript
- Graph
  - Lorsque les relations entre les données sont très complexes
  - Pour travailler avec des hiérarchies et des taxonomies
- Wide-column
  - Lorsqu'on a des données sans structure, qui ne changent pas beaucoup et qui doivent être stockées pendant longtemps
  - Lorsque on a besoin d'un stockage évolutif

Comment va-t-on sélectionner la solution NoSQL?
Selon les propriétés des données

- Modèle de données et patron d'accès.
- Exigences des requêtes.
- Propriétés non-fonctionnelles (performance, sécurité, évolutivité etc.)



Architectures de traitement

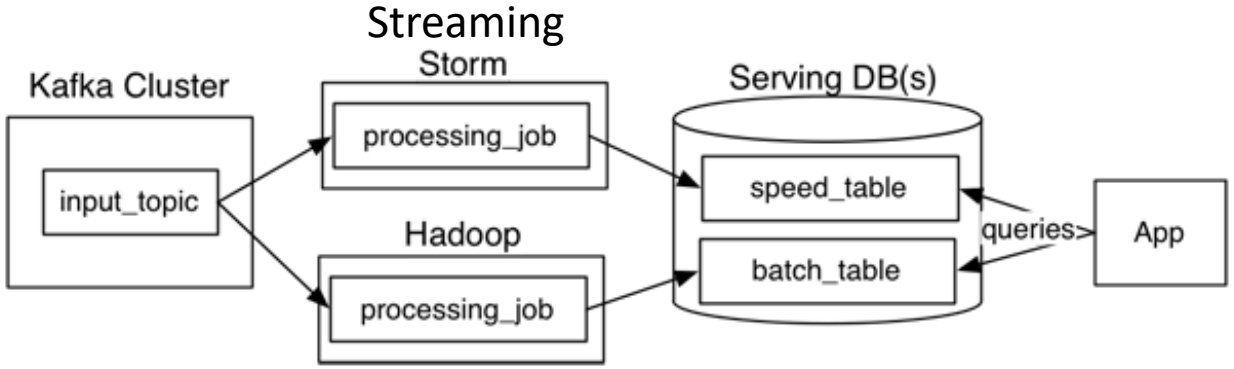
- Architecture Lambda
  - « How to beat the CAP theorem », octobre 2011
- Architecture Kappa
  - « Questionning the Lambda Architecture », juillet 2014

Architecture Lambda

- Rappelez-vous! Selon le théorème CAP, on ne peut pas garantir en même temps la disponibilité et la cohérence.
  - On considère que la tolérance au partitionnement est garantie par défaut.
- Si on garantit la disponibilité quel est le problème avec la cohérence?
  - C'est couteux et lent de mettre à jour toutes les réplications d'énormes quantités de données.
- Mais, si on pourrait réduire…
  - La quantité de données?
  - Ou le nombre de réplications?
- On pourrait séparer les données entre volatiles et immuables.

Architecture Lambda

- Pour les données volatiles, on peut garantir la disponibilité.
  - On garde peu des réplications et on applique des analyses incrémentales sur de petites quantités de données.
  - On analyse les données en temps réel (streaming).
- Pour les données immuables, on peut garantir la cohérence.
  - En utilisant les résultats des analyses en temps réel, on rend les données immuables.
  - On applique des analyses plus complexes et couteuses sur ces données (batch).
  - On favorise la précision plutôt que la performance.



Architecture Lambda

Avantages

- On peut garantir que certaines données sont cohérentes et disponibles.
- On garantit la performance des analyses en temps réel.
- On garantit la précision des analyses sur la plupart de données.
- Une architecture évolutive et tolérante aux pannes.
- Un bon équilibre entre performance et fiabilité.

Désavantages

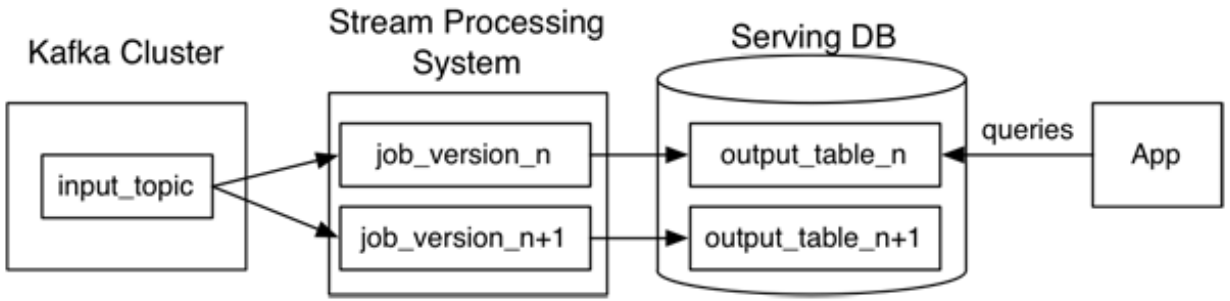
- Le codage peut être un processus assez compliqué.
- Le modèle de données peut être difficile à migrer ou à maintenir.
- La répétition du traitement de données (streaming et batch) crée une surcharge de performance.

Architecture Kappa

- Le retraitement de chaque flux de données n’est pas toujours nécessaire.
- Le retraitement peut être nécessaire juste lorsque la logique des analyses change.
- Il est possible de prévoir et de planifier les retraitements.
- Il y a des applications qui ne requièrent pas de stockage persistent des données pendant longtemps.

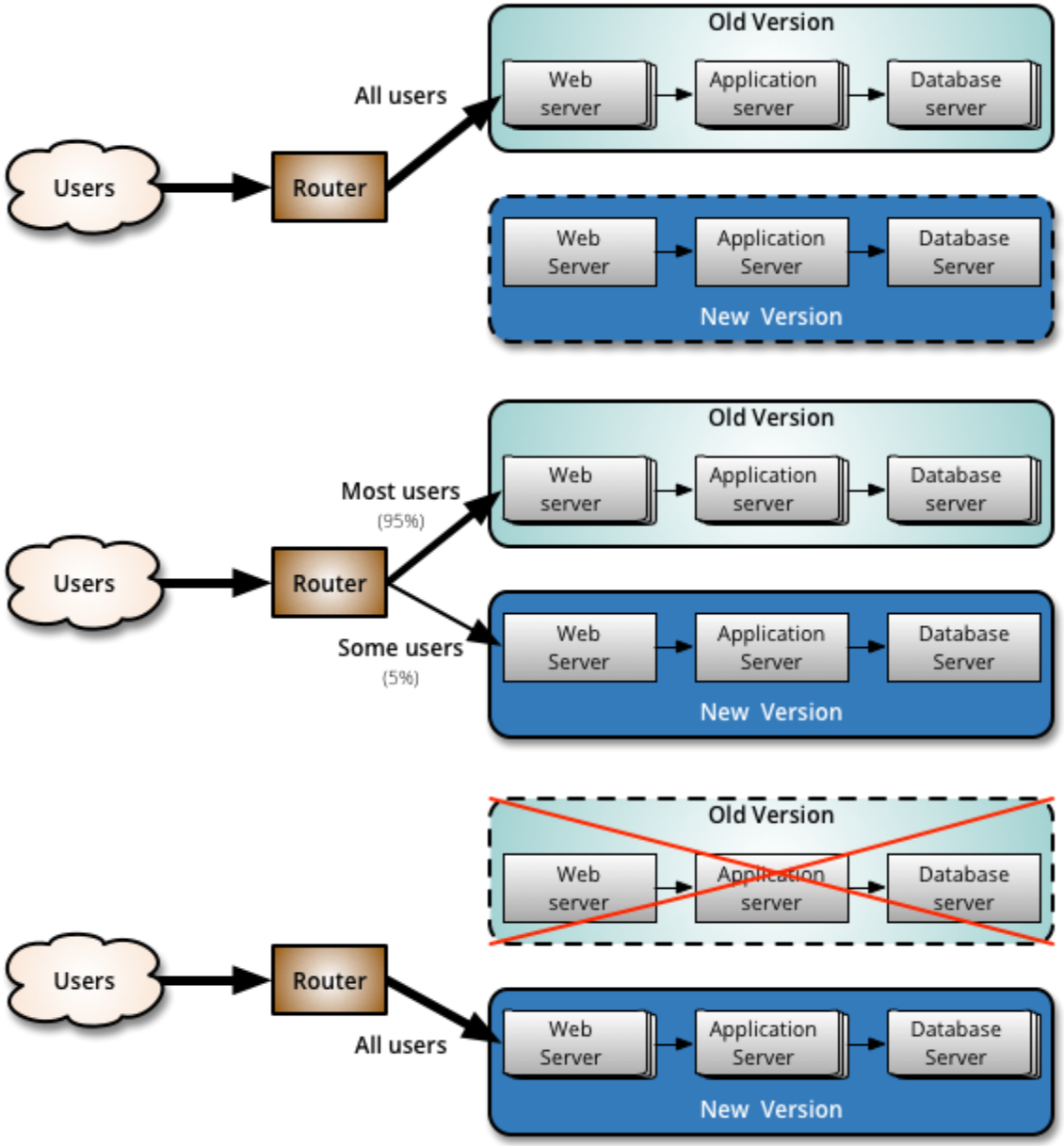
Architecture Kappa

- On garde juste le niveau de streaming.
- On utilise des systèmes orientés messages pour gérer l'entrée des données et faciliter le retraitement.
- La solution offre une flexibilité augmentée par rapport à la maintenance et l'évolution des algorithmes d'analyse.
  - On peut changer l'algorithme et retraiter les données avec la nouvelle analyse.



Intermission : Canary Deployment

- On peut remplacer un algorithme ou un service avec une nouvelle version en temps réel.
- Déployer les deux versions ensemble.
- Graduellement, rediriger les demandes vers la nouvelle version.
- Lorsque toutes les demandes sont servies par la nouvelle version, arrêter l'ancienne version.



Architecture Kappa

Avantages

- Flexibilité augmentée pour le codage.
- Simplicité augmentée pour le modèle des données.
- Facilité de développement des systèmes d’apprentissage en ligne qui n’exigent pas de stockage persistent.
- Utilisation de la mémoire pour une efficacité augmentée.

Désavantages

- Tolérance aux pannes et capacité de récupérer réduites.

Comment va-t-on sélectionner notre architecture de traitement?

Lambda

- Les demandes des utilisateurs doivent être servies sur une base ad-hoc en utilisant le stockage de données immuable.
- Des réponses rapides sont nécessaires et le système doit pouvoir gérer diverses mises à jour sous la forme de nouveaux flux de données.
- Aucunes des données stockées ne doivent être effacées et cela devrait permettre l'ajout de mises à jour et de nouvelles données à la base de données.

Kappa

- Plusieurs événements ou requêtes sur les données sont mis en file d'attente pour être traités en fonction d'un stockage dans un système de fichiers distribué.
- L'ordre des événements et des requêtes n'est pas prédéterminé. Les plates-formes de traitement de flux peuvent interagir avec la base de données à tout moment.
- Nécessité de résilience et de haute disponibilité car la gestion de téraoctets de stockage est requise pour chaque nœud du système afin de supporter la réplication.

La prochaine fois