

# Carte du cours

LOG2410

Conception  
OO (SOLID)

Patrons de  
Conception

LOG8371

Mauvaise  
Conception

LOG3430

Qualité de la  
conception

Styles et Patrons d'Architectures Distribuées

Cadriels

AOP

Microservices

Écosystèmes

SOA

Infonuagique

Plugiciels

Multi-  
niveau

Mégadonnées

LOG3210

Évènementielles

LOG8415

# LOG8430 : Architectures des Megadonnées

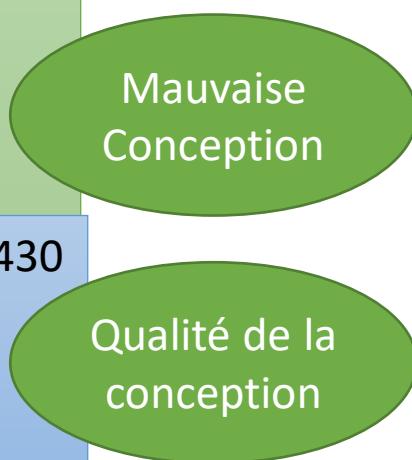
Entrée et analyse de données

# Aujourd’hui

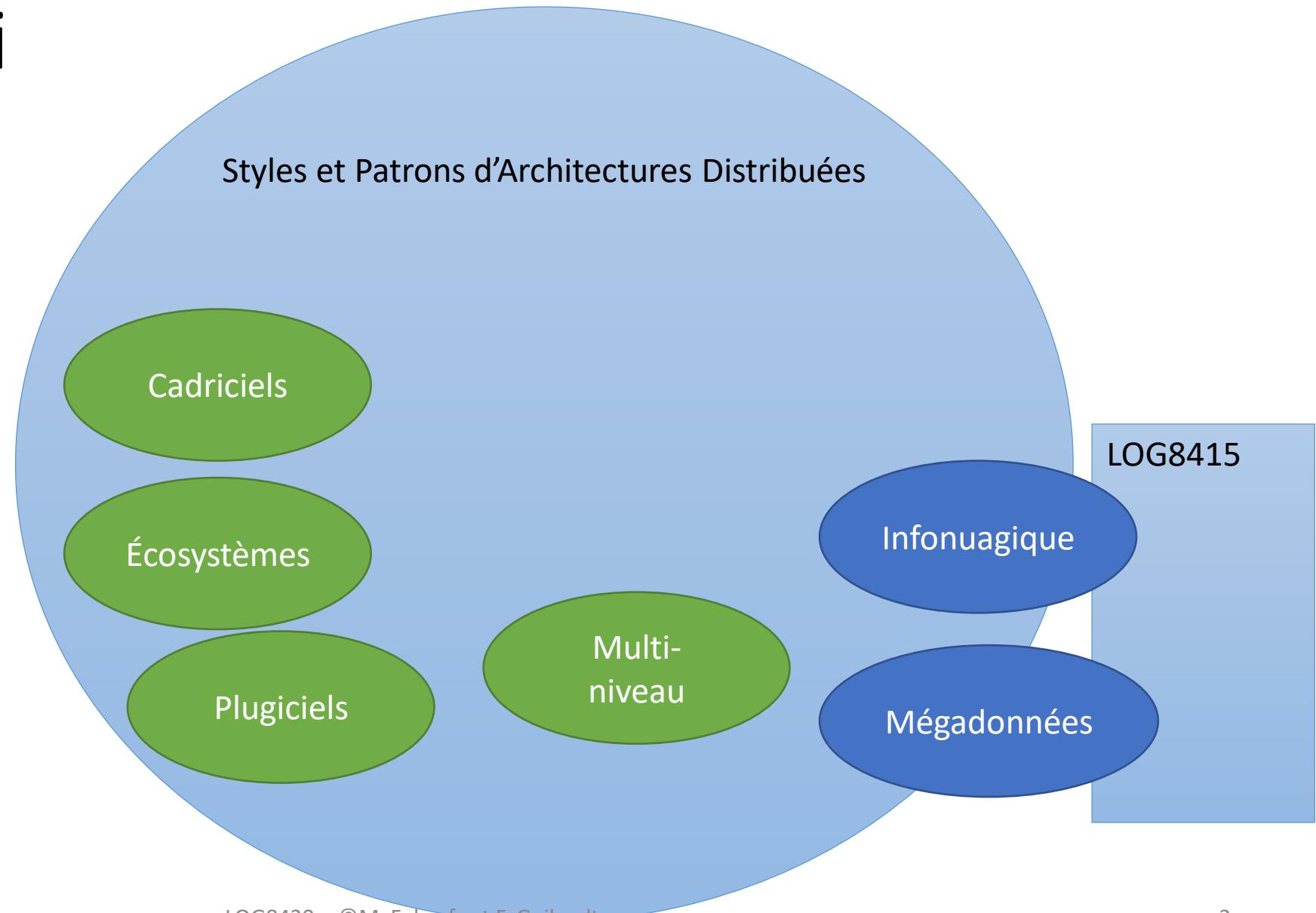
LOG2410

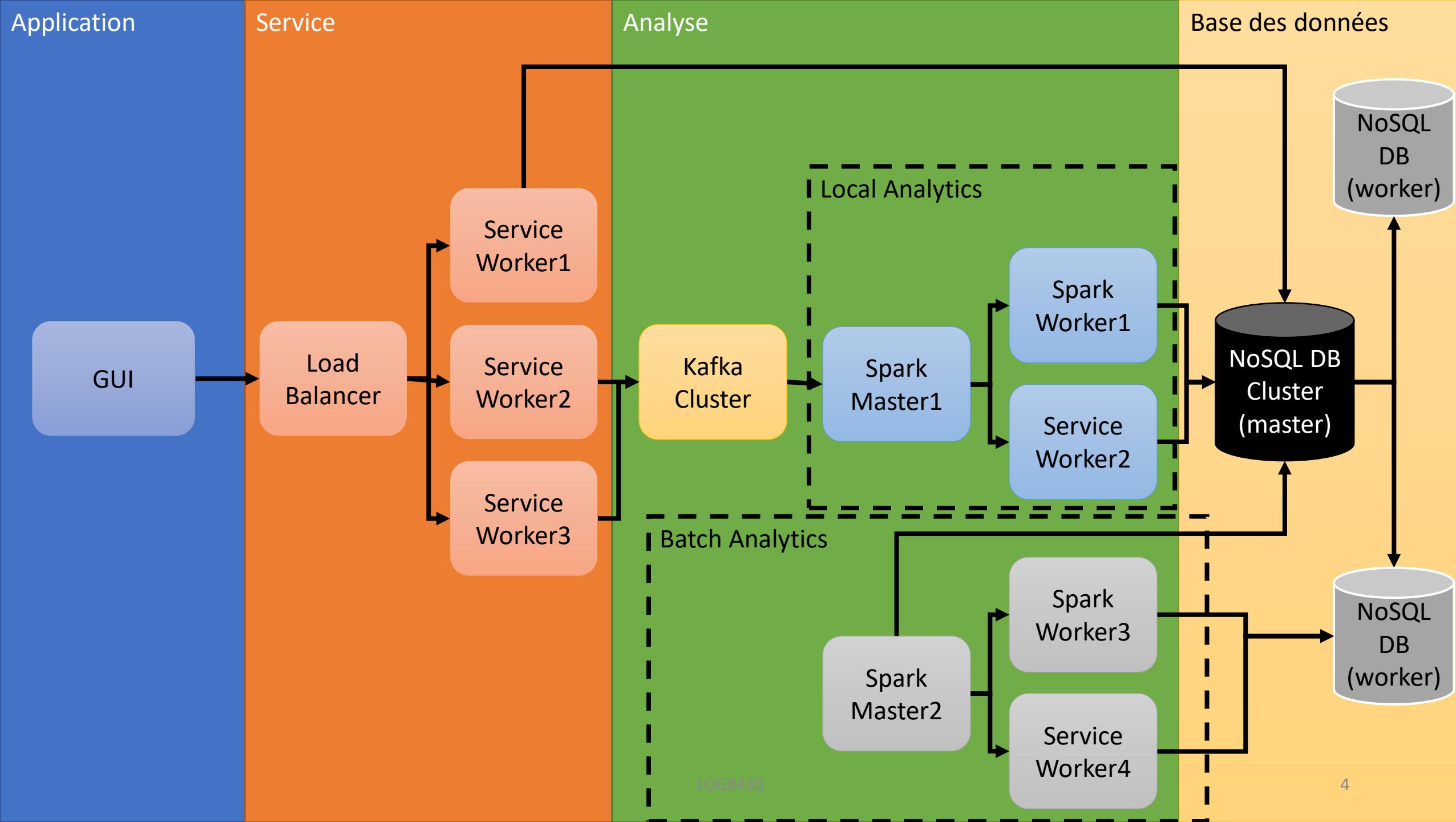


LOG8371



LOG3430





# Analyses et données dans la nouvelle ère

## Accès aux données

- Services
- Encapsulation
- Gestion de l'entrée des données
- Intergiciel orienté messages

## Analyses de données

- Algorithmes
- Architectures d'analyse des mégadonnées

## Systèmes de mégadonnées

- Bases des données NoSQL

# Un petit problème...

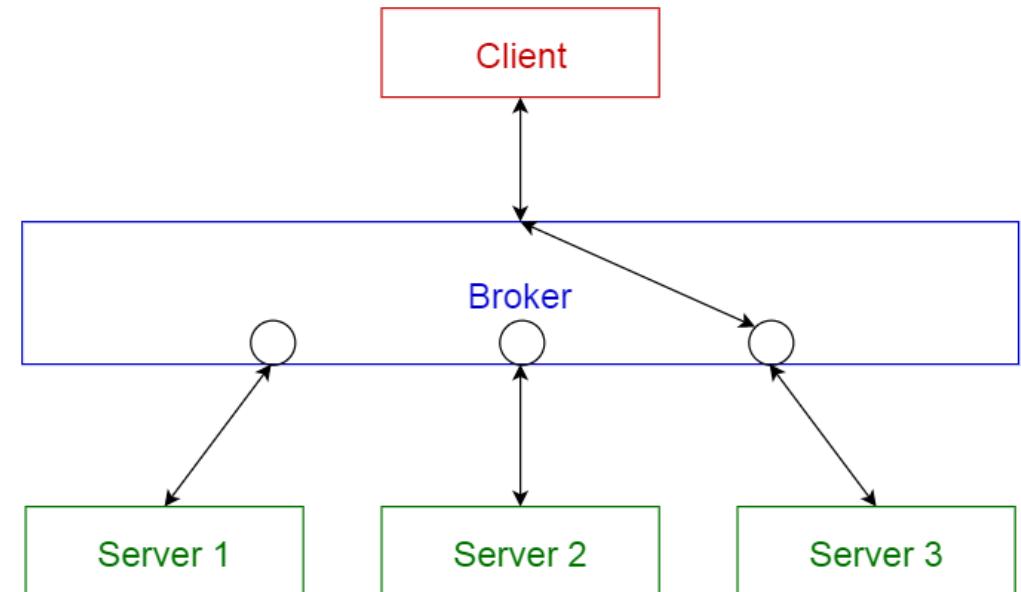
- Supposons qu'on a un bâtiment « intelligent » qui peut contrôler ses conditions environnementales (température, humidité, luminosité). Les capteurs et l'équipement viennent de différents fabricants et avec des spécifications différentes. On n'a qu'un seul système de contrôle et d'analyse, qui va accepter les données.
- Quels sont les défis d'implémentation pour ce système?

# Un petit problème...

- Plusieurs sources, une seule destination.
  - Divers formats de données.
  - Interopérabilité devient difficile.
- Pour rendre les analyses efficaces, il faut avoir assez de données.
  - On ne peut pas traiter chaque élément d'information individuellement.
  - On doit attendre d'avoir une quantité suffisante de données.
- Les capteurs n'ont pas beaucoup d'intelligence.
  - Doit-on implémenter un service par capteur ou par type de capteur pour accepter les données?
  - Qui a la responsabilité de gérer l'entrée des données?
- Les services web ne sont pas une solution suffisante (pas fausse, mais insuffisante).
  - Ils requièrent une connexion directe entre le producteur et le consommateur.
  - Ils requièrent (parfois) une interaction synchrone.

# Intergiciels orientés messages

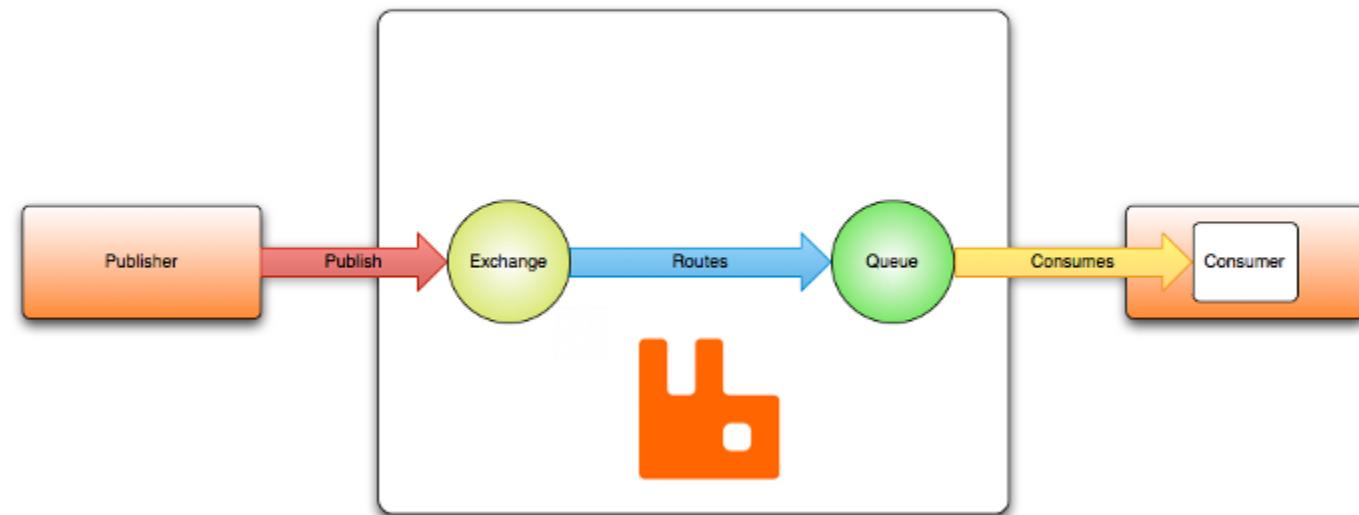
- Dans une architecture orientée message, les modules sont vraiment découplés et interopérables.
- L'entité principale de l'architecture est le message.
- Les messages correspondent aux données qui doivent être communiquées entre les modules.
- Les modules se séparent entre producteurs et consommateurs de messages.
- Le courtier est l'intergiciel.



# Advanced Message Queuing Protocol (AMQP)

- AMQP est un standard ISO pour la spécification d'intergiciels orientés messages.
- AMQP définit les entités d'un système de messages et leur mode de communication.
- Les entités selon l'AMQP:
  - Courtier
  - Producteur
  - Consommateur
  - Échange
  - Queue
  - Message

"Hello, world" example routing



# AMQP : Courtier

- Le courtier implémente l'interopérabilité et il fournit et enlève plusieurs responsabilités aux modules.
- Le courtier permet aussi l'asynchronicité : le producteur soumet un message et continue son travail; le consommateur peut recevoir le message lorsqu'il est prêt.
- Le courtier a la responsabilité d'acheminer les messages entre les producteurs et les consommateurs.
- Le courtier peut aussi transformer les messages. Cela aide le traitement des données en divers formats.
- Le courtier est responsable de garantir la sécurité, la performance, la fiabilité et l'évolutivité du système.

# AMQP : Producteurs et Échanges

- Les producteurs soumettent des messages au courtier et en particulier aux échanges.
- Les échanges acheminent les messages aux queues selon un algorithme :
  - **Direct** : L'échange achemine le message à une queue spécifiée par une clé.
  - « **Fanout** » : L'échange achemine le message à toutes les queues liées à l'échange. La clé est ignorée.
  - « **Thème** » : Le message est acheminé selon la clé et un paramètre supplémentaire pour permettre la multidiffusion.
  - « **En-tête** » : Un échange en-tête achemine le message à une queue spécifié par plusieurs paramètres qu'une clé.
- Outre l'algorithme, les échanges ont aussi des propriétés :
  - Nom
  - Durabilité : L'échange survit à une relance du courtier.
  - Suppression automatique : lorsque il n'y a pas de queues liées à l'échange.
  - Arguments supplémentaires : utilisés par les extensions ou les plugins.

# AMQP : Consommateurs et Queues

- Les consommateurs consomment les messages en s'enregistrant sur les queues.
- Les consommateurs peuvent retirer des messages de la queue ou pousser des messages sur la queue.
- La queue fonctionne comme un stockage temporaire pour les messages.
- Les queues peuvent être durables ou pas. Une queue durable est stockée sur disque et elle survit à une relance du courtier.
- Une queue peut être exclusive à une connexion.
- Les queues sont liées aux échanges par des *liaisons*.
  - Les liaisons représentent des règles selon lesquelles les échanges acheminent les messages aux queues.

# AMQP : Messages

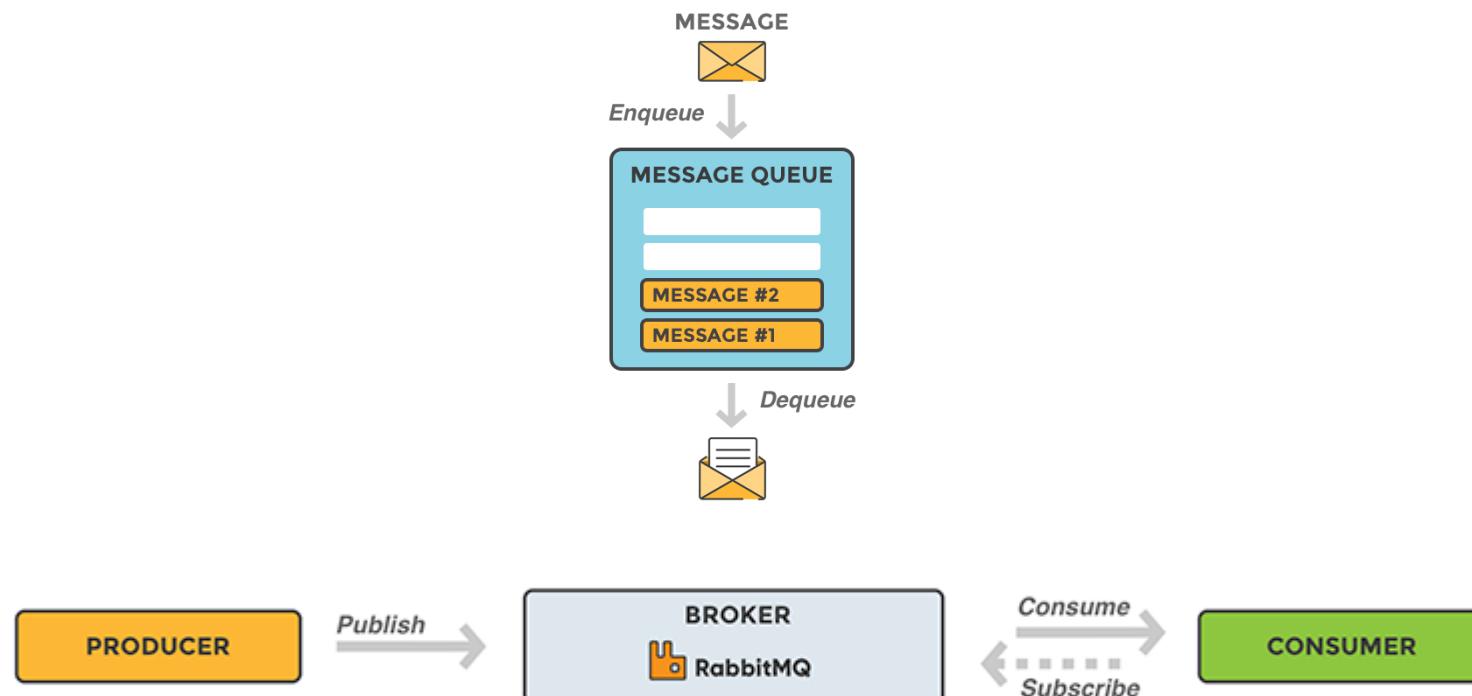
- Les messages sont définis par leurs données et leurs attributs.
- Le contenu (les données) du message est optionnel et il est ignoré par le courtier.
- Le message peut être déclaré persistent, ce qui implique que le message est stocké sur disque et il peut survivre à une relance.
- Les attributs du message incluent:
  - Type et encodage du contenu
  - Clé d'acheminement
  - Persistent ou non
  - Priorité
  - ID du producteur
  - Horodatage et temps d'expiration du message

# Implémentations d'AMQP

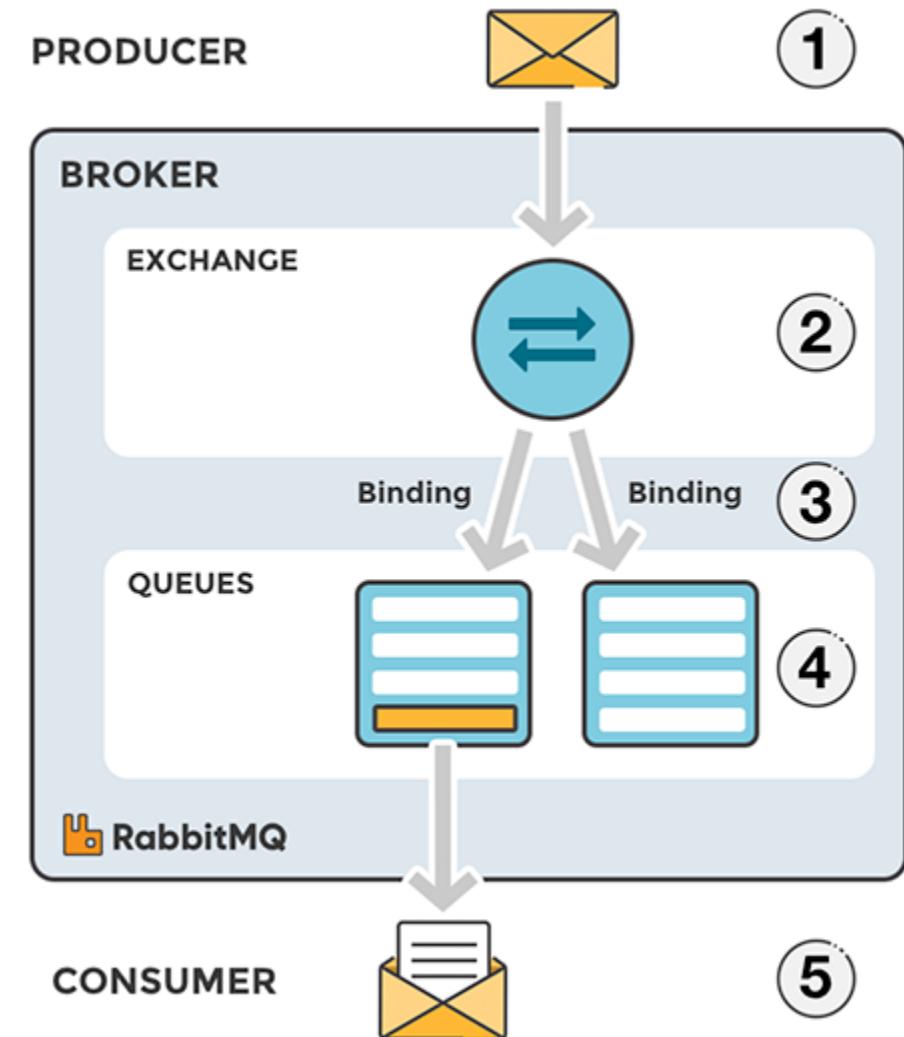
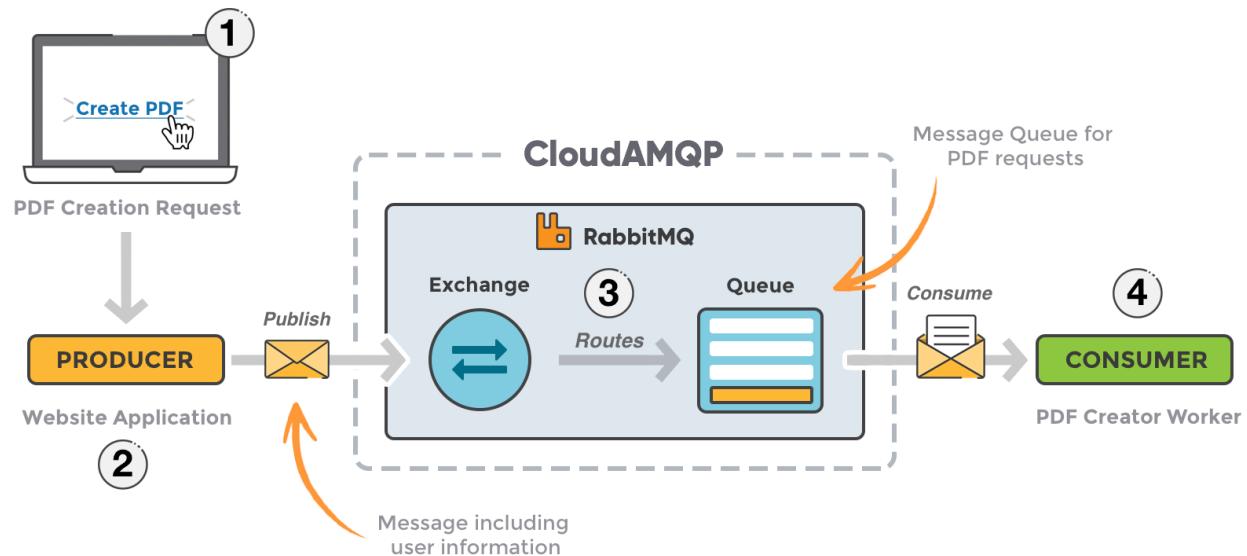


# RabbitMQ

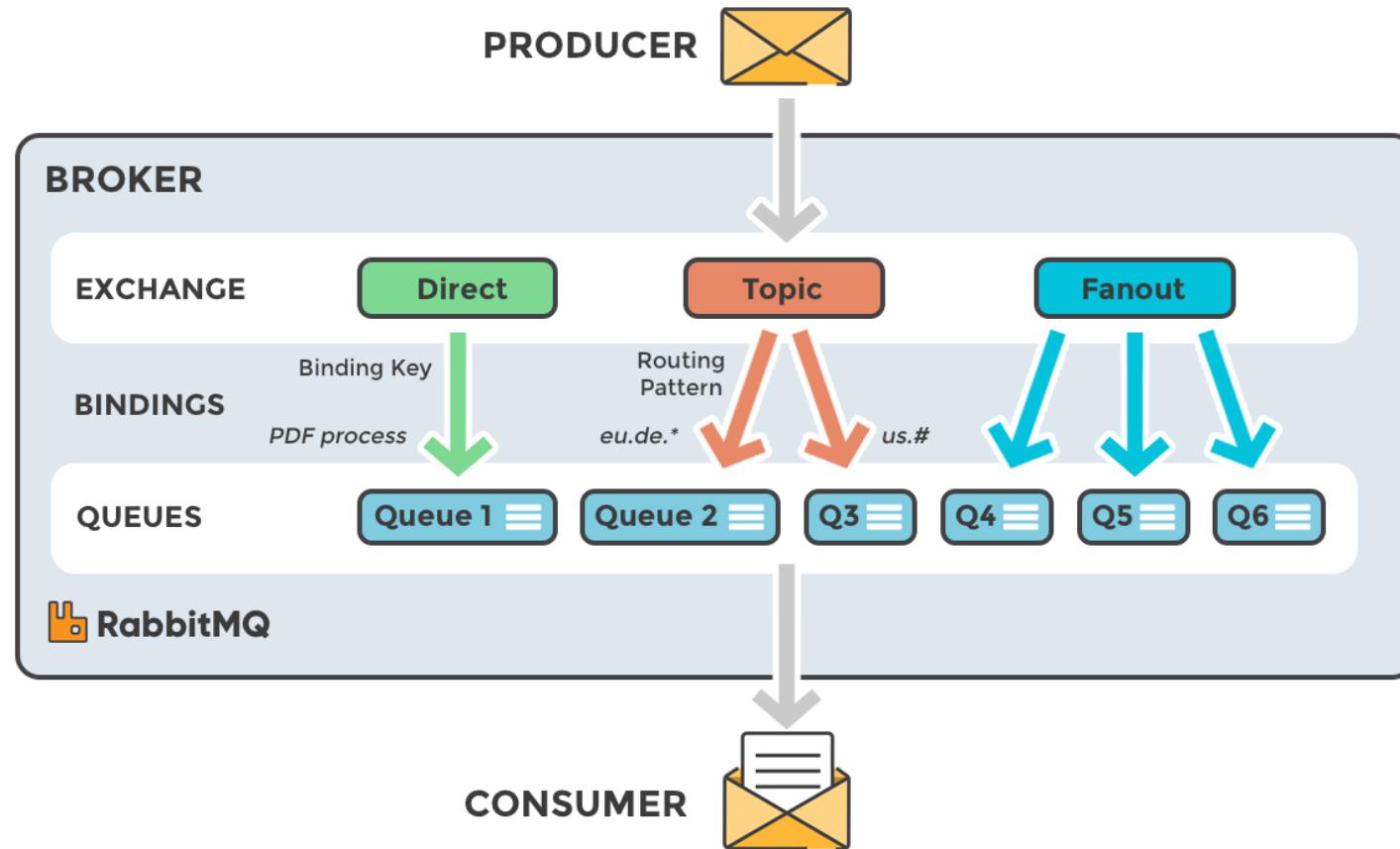
- RabbitMQ est une implémentation directe d'AMQP.
  - Tous les éléments discutés précédemment sont spécifiés dans RabbitMQ.



# RabbitMQ : Un exemple

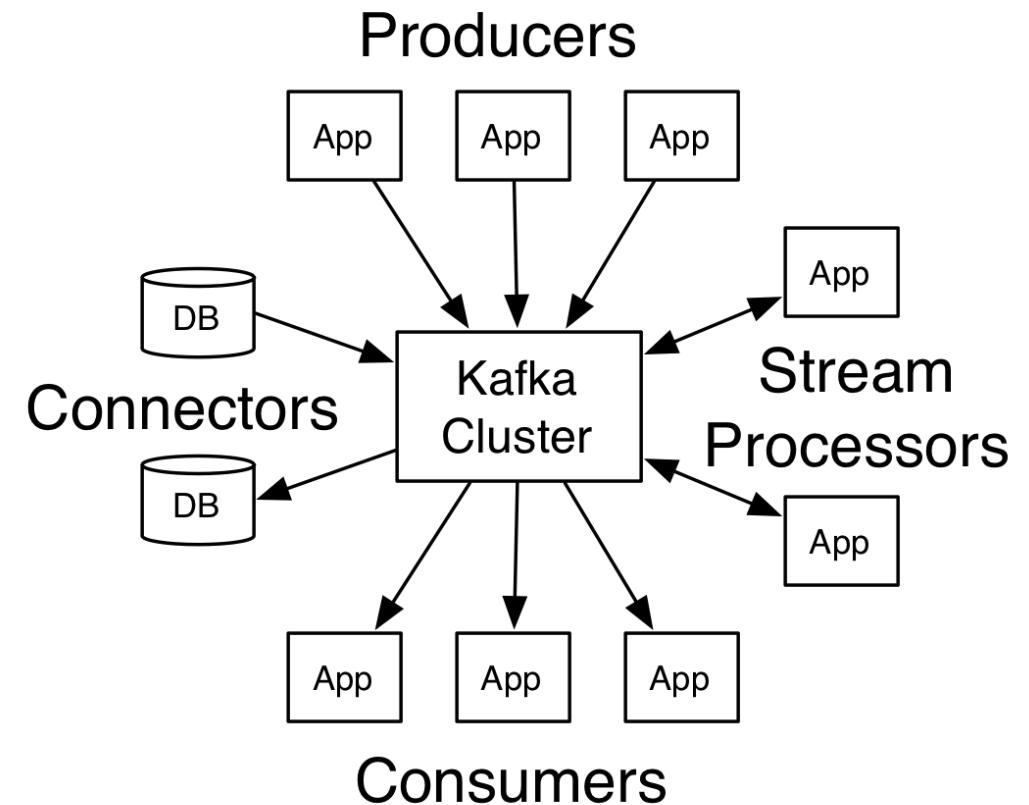


# RabbitMQ : Types d'échange



# Apache Kafka

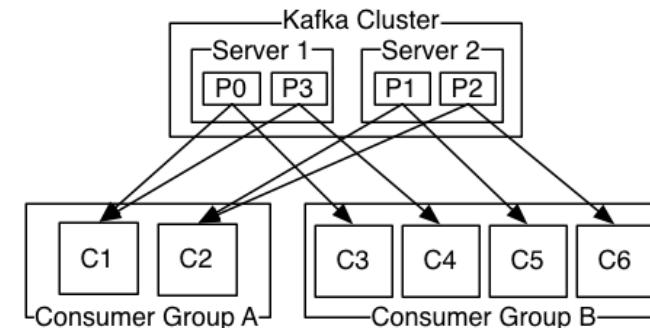
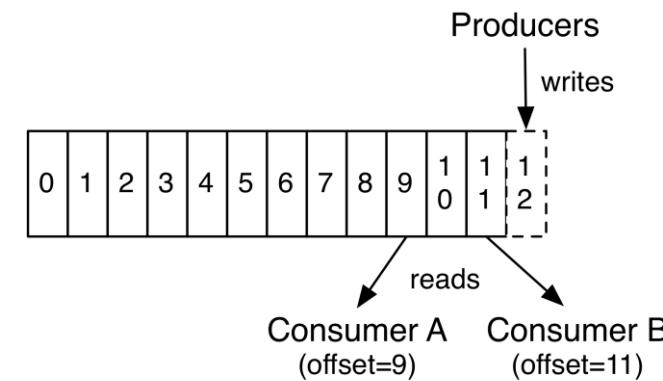
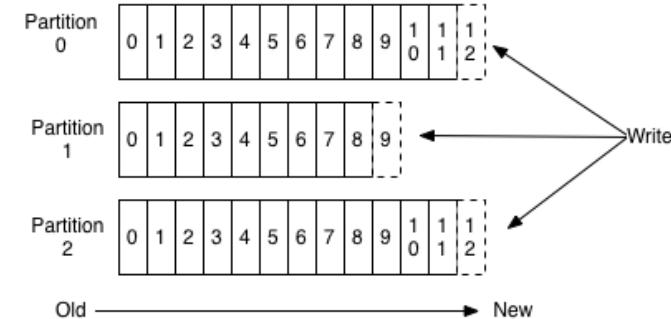
- Kafka est défini comme un service distribué de streaming.
- Les streams sont comme les queues mais avec quelques petites différences.
- Les streams permettent la diffusion des données et pas juste la multidiffusion.
- Kafka utilise la transmission selon des thèmes.
  - Les consommateurs s'enregistrent sur des thèmes et attendent les données en temps réel.
- Toutes les données sont persistantes sur disque.



# Kafka : Thèmes

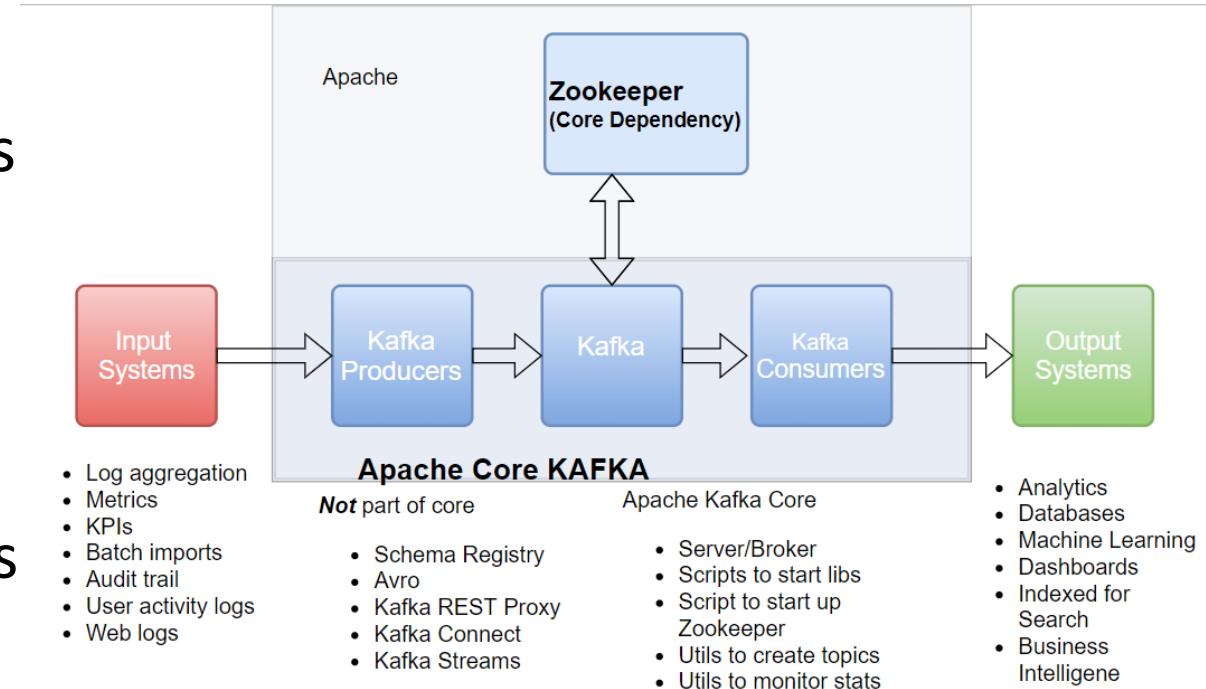
- L'ordre des messages est dicté par les producteurs, mais les consommateurs peuvent contrôler l'ordre de lecture.
- Grâce aux thèmes et au groupement des consommateurs, Kafka offre les avantages du parallélisme et de la diffusion.
- Des messages peuvent être lus par plusieurs consommateurs.

Anatomy of a Topic



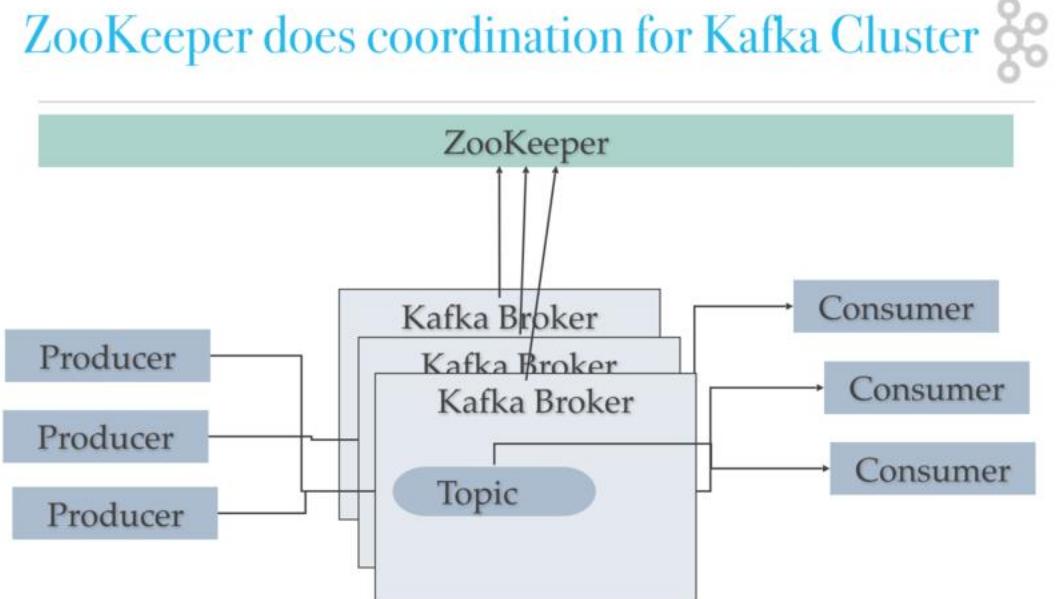
# Kafka : Architecture de base

- « Input Systems » = Producteurs
- « Output Systems » = Consommateurs
- « Kafka Producers » = Échanges
- « Kafka Consumers » = Queues
- « Kafka » = Courtier
- Pour Kafka, les échanges et les queues sont l'équivalent des stubs pour les services.
  - Ce sont des classes locales qui seront appelées par les systèmes externes.



# Kafka : ZooKeeper

- Kafka peut fonctionner en groupe de serveurs/courtiers.
- Cela aide la performance et la fiabilité.
  - Les messages peuvent être répliqués entre les courtiers pour augmenter la tolérance aux pannes.
- ZooKeeper est un service de découverte qui peut gérer le groupe.
  - Les clients communiquent avec le service ZooKeeper pour se connecter à un courtier.



# RabbitMQ vs Kafka

## RabbitMQ

- Courtier intelligent/Consommateurs simples
- Fiabilité et découplage sont les priorités.
- Plusieurs types d'échange sont supportés.
- Transmission directe ou multidiffusion.
- Sécurité est augmentée.
- La haute performance est possible mais avec des ressources significatives.

## Kafka

- Courtier simple/Consommateurs intelligents
- Streaming permet l'utilisation d'une seule connexion avec un taux d'entrée augmentée.
- Seul l'échange par thème est supporté.
- Transmission par diffusion.
- Sécurité peut être un problème.
- La performance est augmentée.

# Intergiciels orientés messages

## Avantages

- Flexibilité
- Découplage/Interopérabilité
- Fiabilité
- Sécurité
- Évolutivité

## Désavantages

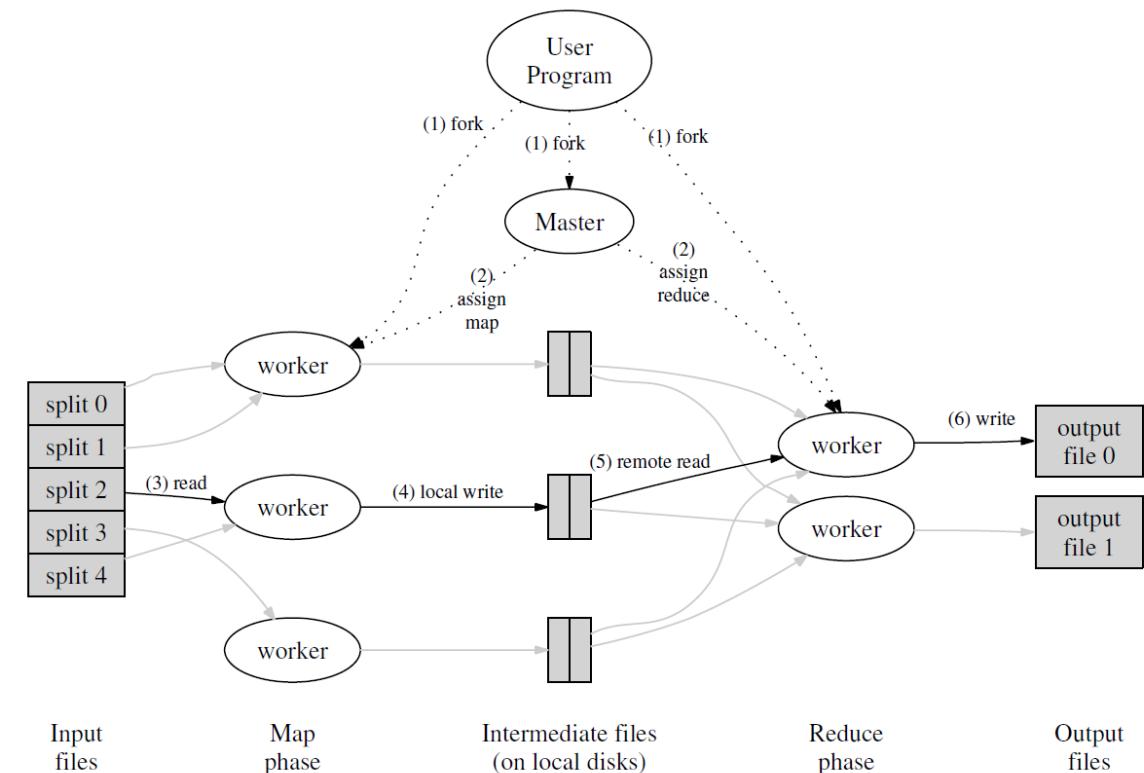
- Performance
- Complexité de l'infrastructure et de l'architecture
- Découplage n'est pas toujours désiré.

# Analyse : MapReduce

- MapReduce est un modèle de programmation pour l'analyse distribuée des mégadonnées.
- Le modèle contient un algorithme, une implémentation et une infrastructure.
- Dans une implémentation MapReduce, le développeur doit définir deux tâches :
  - Une tâche *map* qui représente un travail d'analyse (p.ex., un tri, une addition, une recherche etc.)
  - Une tâche *reduce* qui représente un travail de sommation, qui va agréger les résultats des tâches *map* individuelles.
- Les tâches peuvent être exécutées sur plusieurs nœuds de traitement et elles requièrent plusieurs nœuds de stockage pour faciliter le traitement des données et pour stocker les résultats finaux si nécessaire.
  - Systèmes de données distribués qui sont traditionnellement utilisés incluent GFS (Google File System) et HDFS (Hadoop Distributed File System)

# MapReduce : Processus

- Les données sont définies et organisées en pairs de clés-valeurs.
- Les données sont partitionnées automatiquement ou explicitement par le développeur.
  - Par taille, par distribution géographique ou temporelle.
- Les données sont partagées parmi les ouvriers selon les clés.
- Une copie des tâches (map et reduce) est transmise à chaque ouvrier.
- Les ouvriers exécutent les tâches *map* en utilisant des données d'entrée selon les clés et ils stockent les résultats intermédiaires.
- D'autres ouvriers agrègent les résultats map en exécutant les tâches *reduce*.
- Les résultats sont stockés dans un ou plusieurs fichiers.
- Les développeurs peuvent traiter les fichiers ou les donner à un autre tâche MapReduce.



# MapReduce : Architecture

- Le modèle suit une architecture « master-slave ».
- Le master est responsable pour la distribution des tâches.
- Les ouvriers peuvent accepter des tâches *map* aussi bien que des tâches *reduce*.
- Le master est responsable pour balancer la charge entre les ouvriers.
- Le master contrôle périodiquement le bon fonctionnement des ouvriers. Cela garantit la fiabilité du système et la haute disponibilité du service.
- Le master de son côté est un point de faiblesse.
  - De nouvelles versions résolvent ce problème.
- Le système distribué des fichiers suit aussi une architecture « master-slave ».

# MapReduce : Avantages et Désavantages

## Avantages

- Efficacité grâce au parallélisme.
- Disponibilité.
- Tolérance aux pannes.
- Évolutivité.
- Mobilité minimale des données.
  - Le traitement des données peut se produire sur le serveur qui stocke les données.

## Désavantages

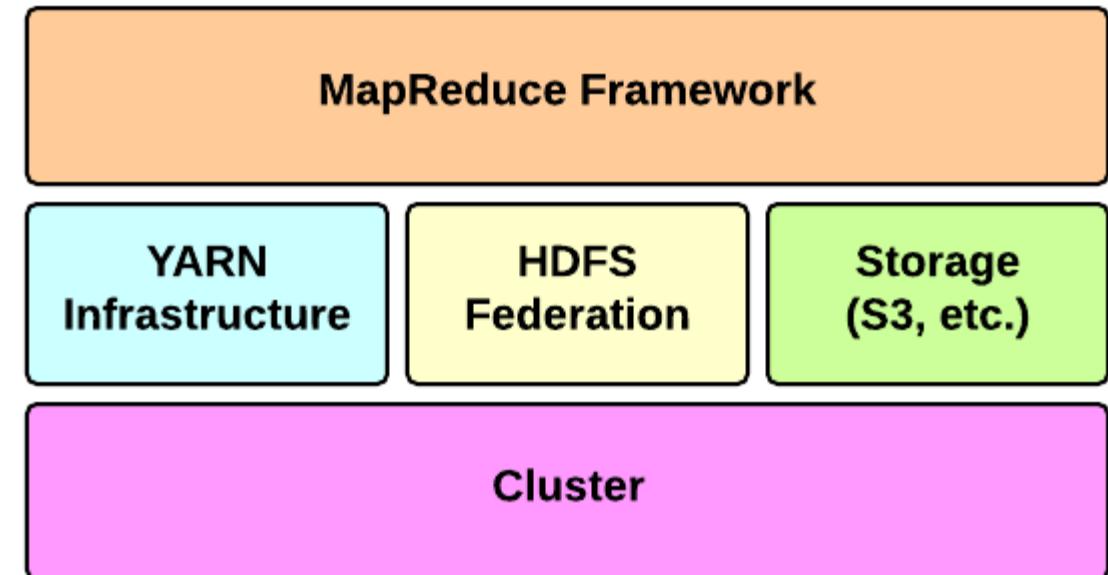
- « Region hotspotting »
  - L'utilisation des clés pour partitionner et partager les données parmi les ouvriers peut charger un ouvrier particulier.
  - On peut résoudre cette situation en utilisant le hachage sur les clés.
- Ce ne sont pas tous les types de traitement qui peuvent être implémentés par MapReduce.

# Implémentations de MapReduce



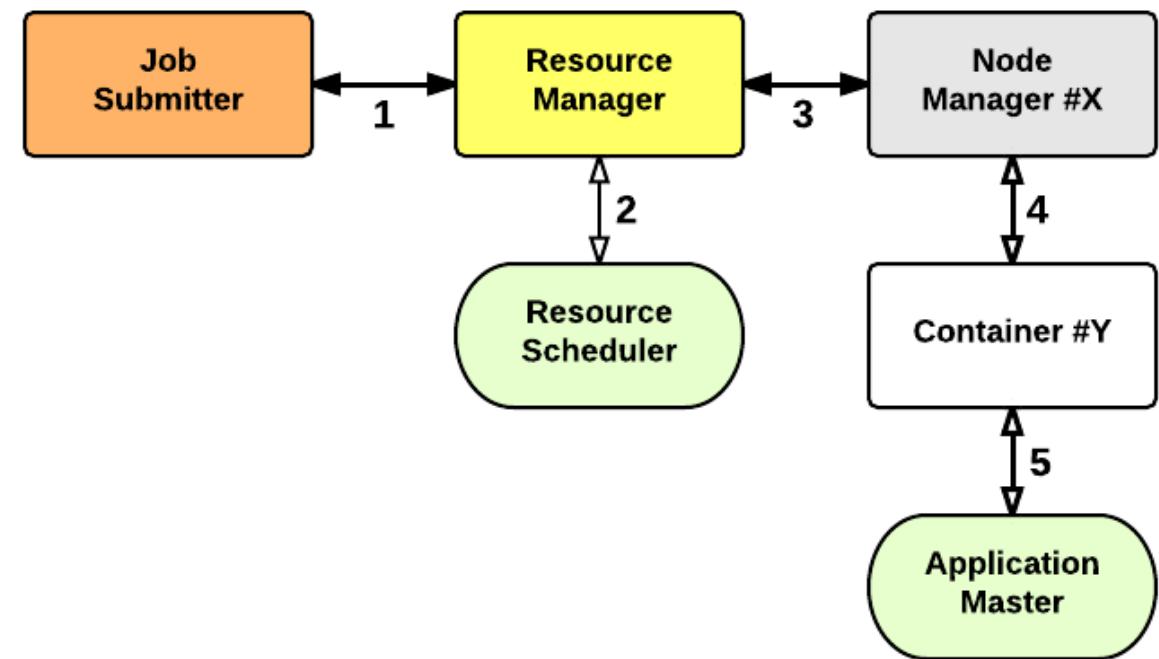
# Hadoop : Architecture de Base

- Hadoop est une implémentation directe de MapReduce.
- Elle supporte les mêmes modules principaux.
- Le « Cluster » représente l'infrastructure physique (ou virtuelle) pour le traitement de données.
- YARN est le système de gestion du cluster.
- HDFS est le système de fichiers distribué.
- S3 est le stockage physique où le HDFS existe.
- MapReduce est l'implémentation du modèle.



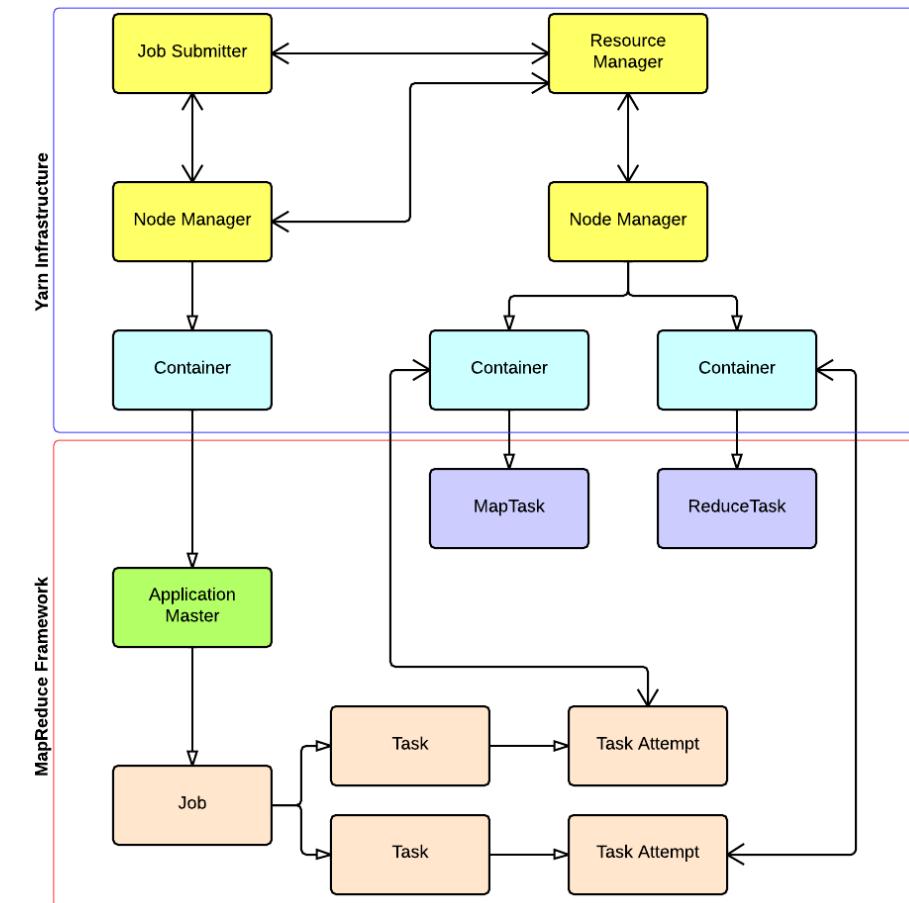
# YARN : Flux de travail

- Un client soumit un travail pour une application.
- Le gestionnaire de ressources (GR) cherche les ressources disponibles.
- Le GR contacte le gestionnaire de nœuds (GN).
- Le GN lance un conteneur avec les ressources disponibles.
- Le conteneur exécute le master de l'application



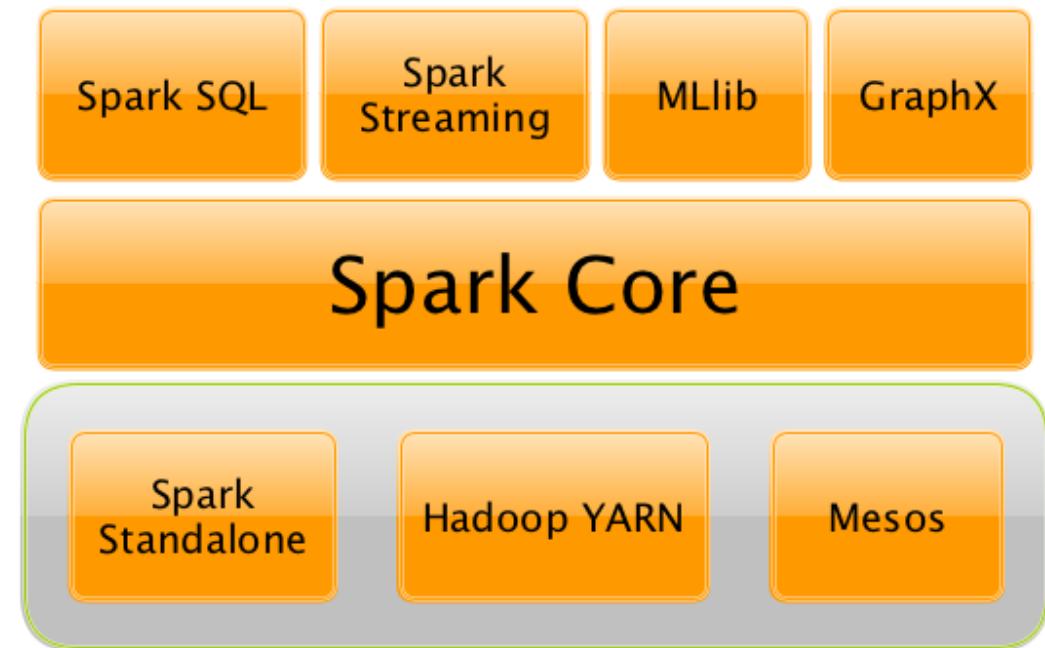
# Hadoop : MapReduce Architecture

- Les conteneurs contiennent le master et les ouvriers.
- Le master de l'application lance le travail et le travail est responsable de définir les tâches (map et reduce).
- Les tâches sont déployées dans les conteneurs pour s'exécuter.
- Les ressources nécessaires sont déterminées et fournies par YARN.



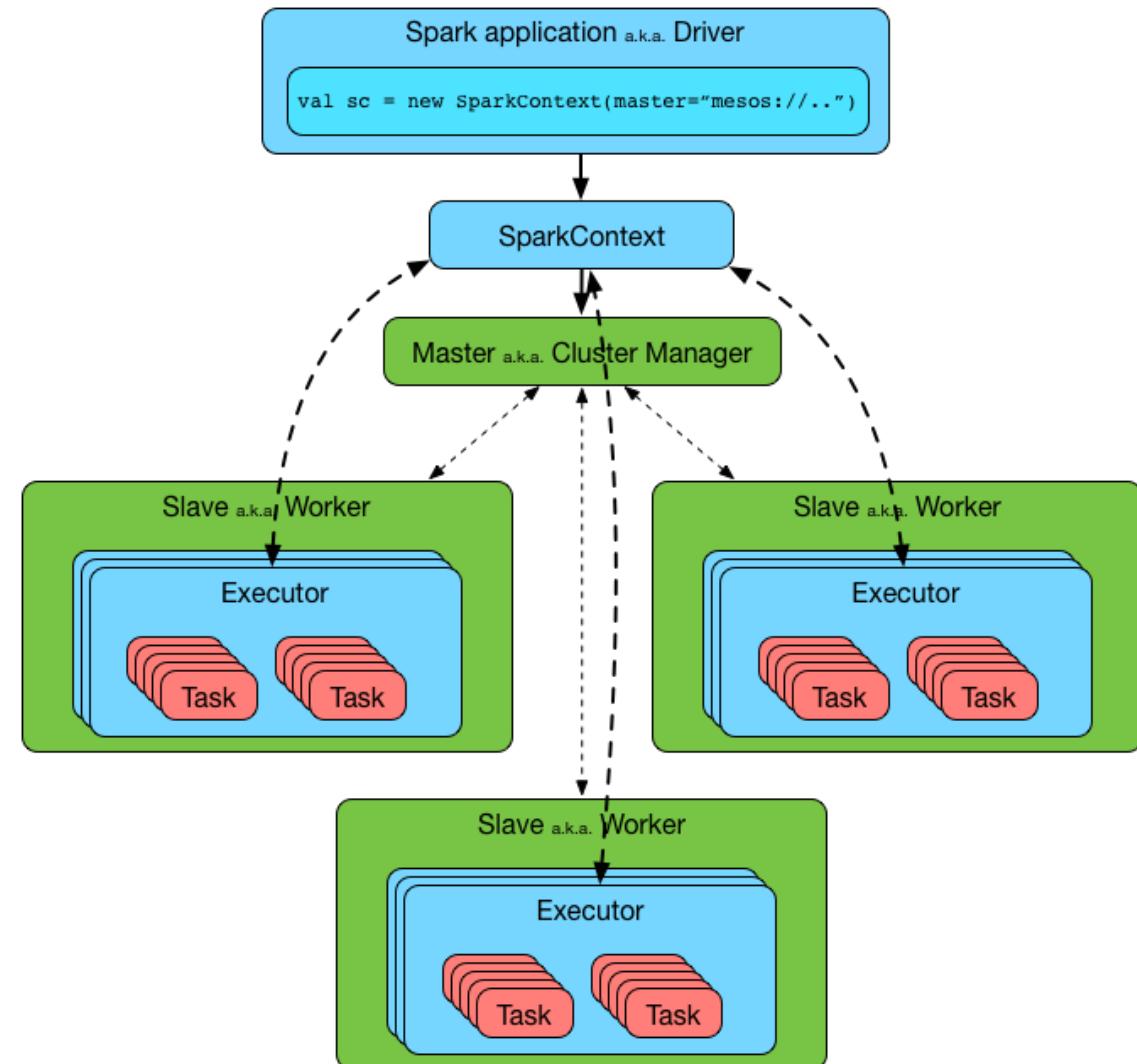
# Spark : La plateforme

- Spark est une autre implémentation de MapReduce (Spark Core).
- Elle fonctionne aussi en groupe de ressources.
- Elle peut fonctionner avec plusieurs gestionnaires de ressources, mais elle fournit son propre gestionnaire aussi.
- Spark fournit aussi plusieurs capacités, outre MapReduce.
  - Spark SQL, pour les opérations avec les bases des données relationnelles.
  - Spark Streaming
  - MLlib, pour l'apprentissage automatique.
  - GraphX, pour l'analyse des graphes.



# Spark : Architecture

- Une exécution de Spark est divisée en deux types de processus, un *driver* et des *executors*.
- Le *driver* est le travail/application et il définit un contexte.
- Le contexte spécifie la configuration du travail.
- Le travail définit les tâches et lance les *executors*, un pour chaque tâche.



# Spark : Resilient Distributed Dataset

- Les RDDs expliquent la puissance de Spark.
- Même si Spark peut fonctionner avec un système de fichiers comme HDFS, grâce aux RDDs, Spark fonctionne en mémoire.
- Cela le rend plus rapide et plus efficace que Hadoop. (100x plus rapide)
- Mais, si les données sont extrêmement nombreuses, le système est forcé d'utiliser le disque. Toutefois, Spark est encore plus rapide. (10x plus rapide)
- Les RDDs peuvent être récupérés en cas d'un échec des serveurs.
- Cela augmente la tolérance aux pannes.

# Hadoop vs Spark

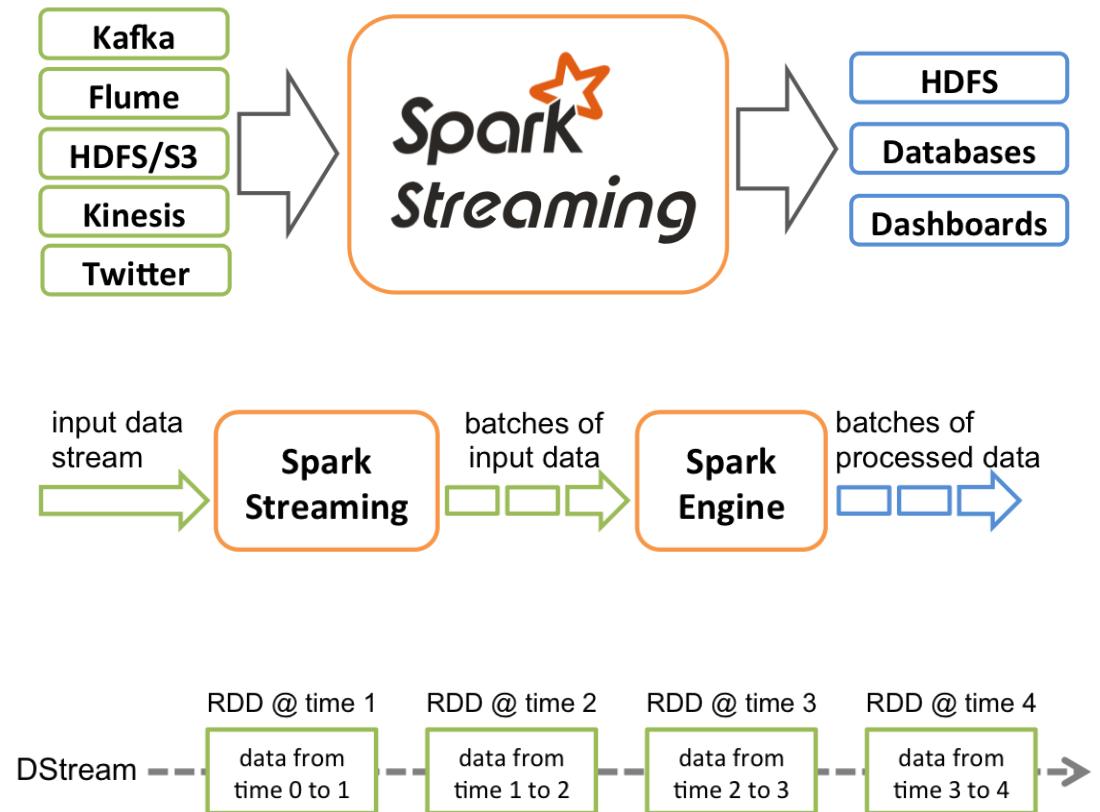
	Hadoop	Spark
Performance	o	+
Utilisabilité/Portabilité	+	++
Frais	o	-/+
Compatibilité	+	+
Traitement de données	-	+
Tolérance aux pannes	+	++
Évolutivité	++	+
Sécurité	+	o

# Streaming Analytics

- Des données peuvent être analysées en lots ou en flux.
- La version de MapReduce qu'on a vue analyse les données en lots.
  - Une grande quantité de données entrent dans le système et elles sont toutes analysées lors d'une exécution du système.
- En cas de flux, la connexion entre le producteur des données et le système d'analyse reste toujours ouverte. Les données sont analysées en morceau comme elles viennent.
- Cela permet l'analyse de données en temps réel, ce qui est important pour des systèmes tels que les systèmes d'alarmes ou financiers.

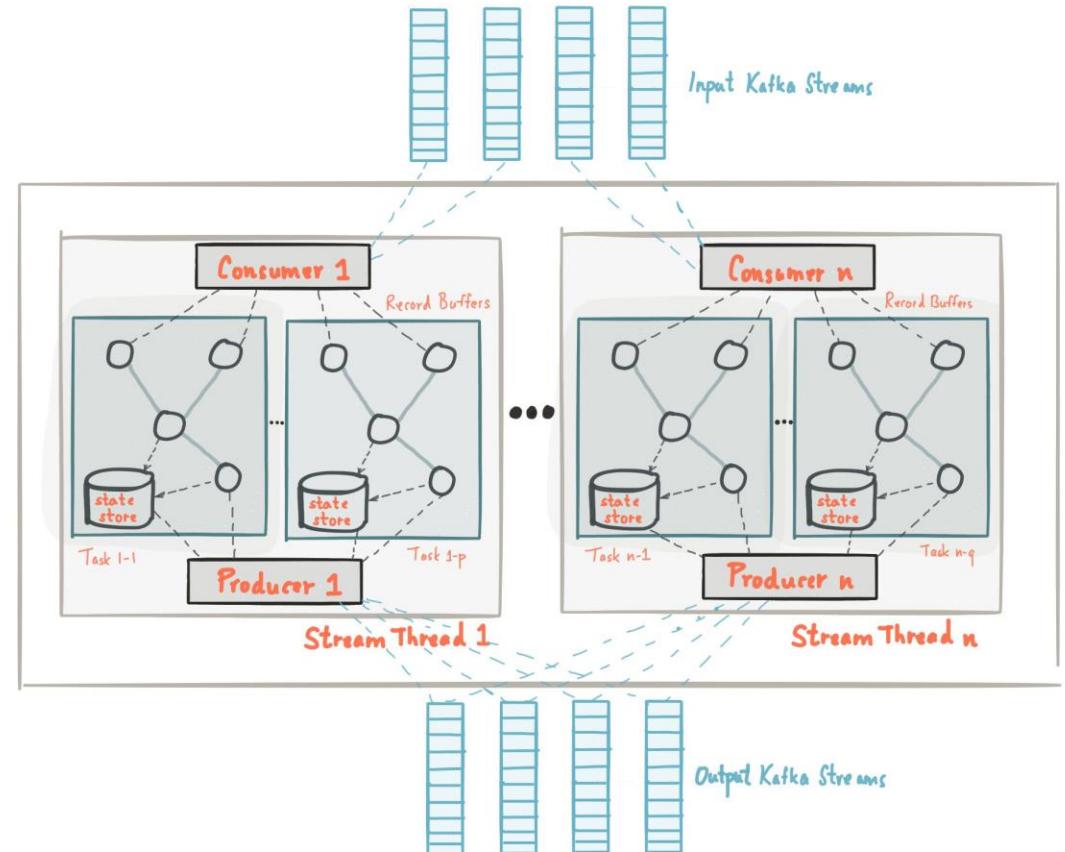
# Spark Streaming

- Spark streaming accepte des flux de données de plusieurs sources.
- À intervalles fréquents, Spark extrait des données et les divise en petits lots, qui sont insérés dans le système principal.
- Les flux de données sont représentés comme DStreams (streams discrets), qui sont des fils de RDDs.



# Kafka Streaming

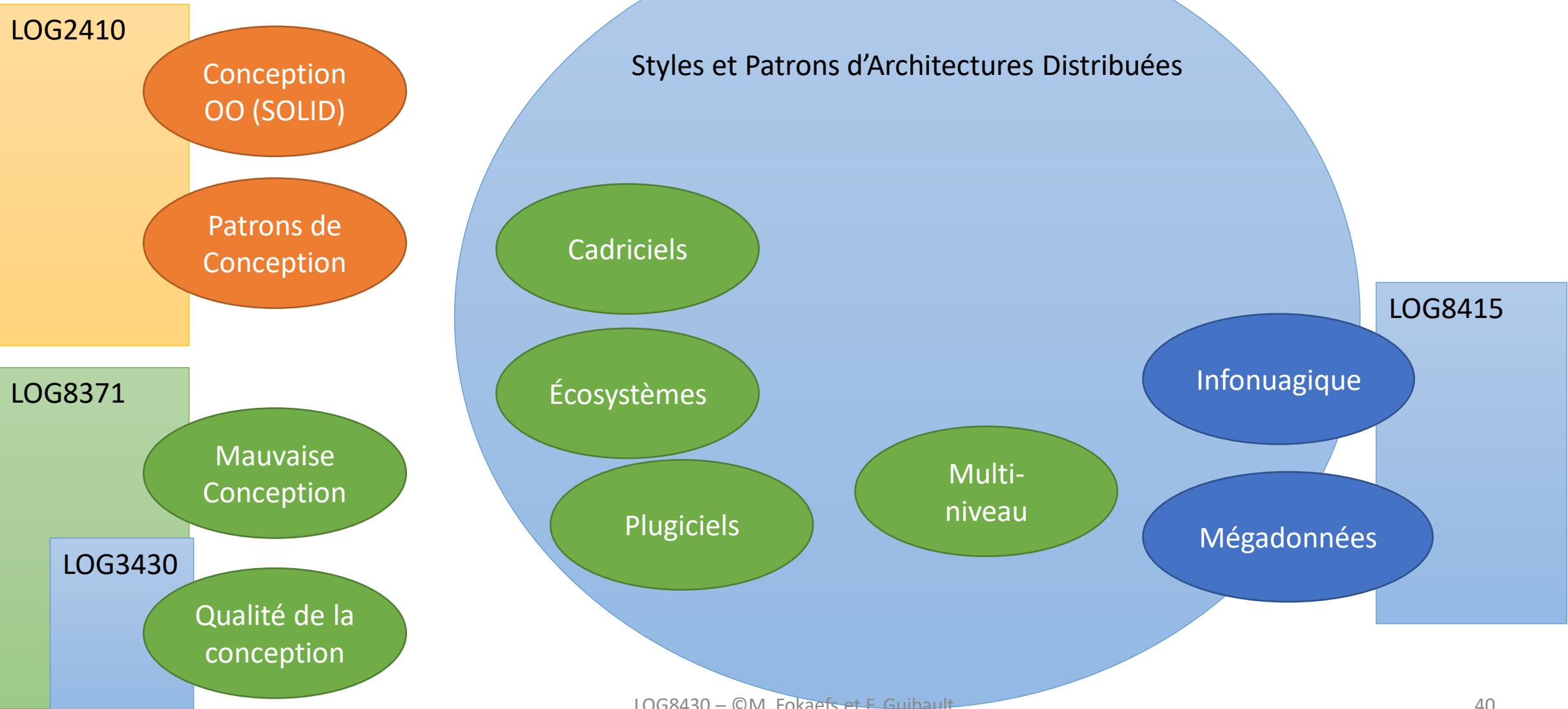
- Pour Kafka, chaque thème correspond à un flux.
- Kafka utilise les filières pour paralléliser le traitement des flux.
- Kafka stocke l'état local pour chaque partition (stateful operation).



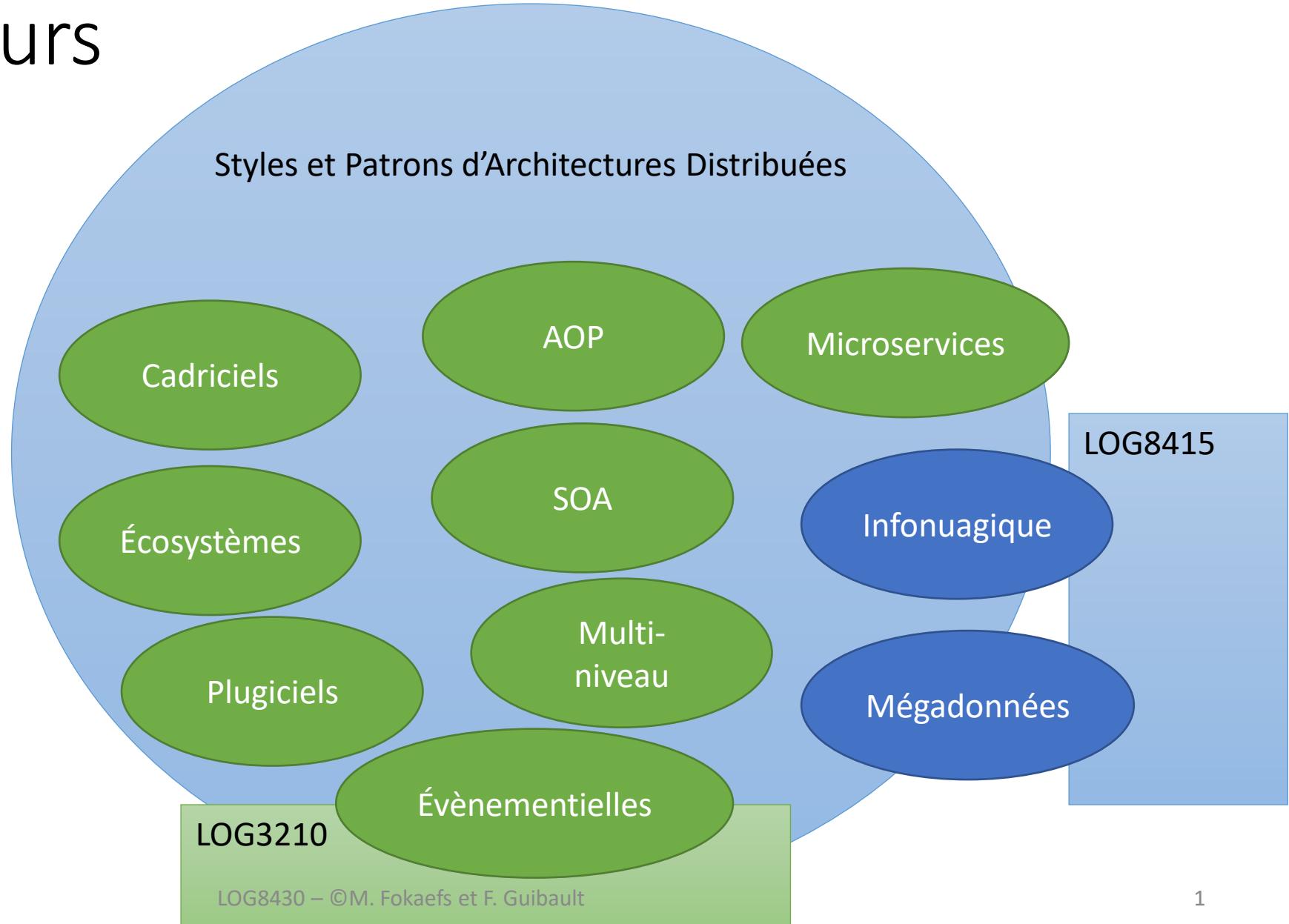
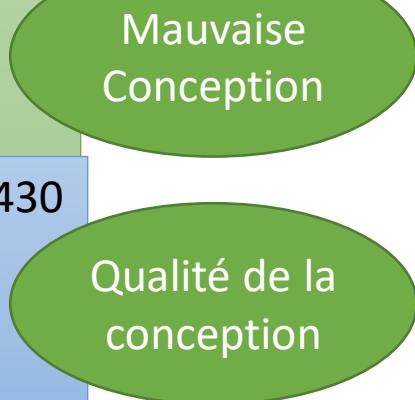
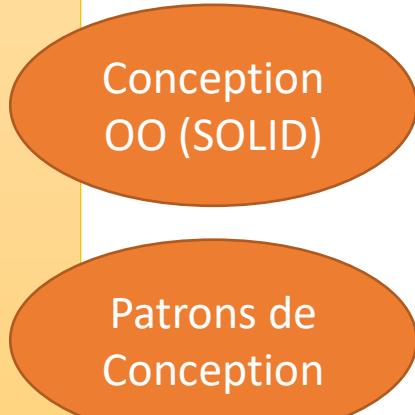
# Spark Streaming vs Kafka Streaming

- La définition des flux et l'entrée des données est basée sur le temps en Spark et sur les évènements en Kafka.
- Spark est une fausse analyse stream, car en réalité il analyse les données en petits lots.
- Kafka est plus efficace et meilleur pour réagir aux évènements.
- Spark est plus stable et portable, et il est compatible avec toutes les technologies de Hadoop.

# Carte du cours



# Carte du cours

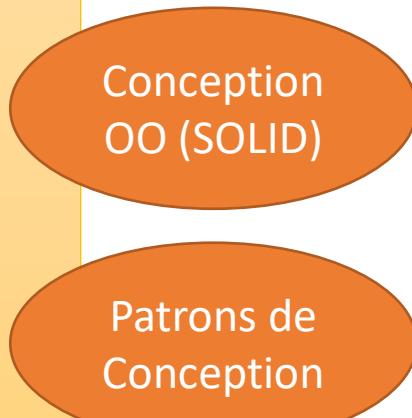


# LOG8430 : Architectures des Megadonnées

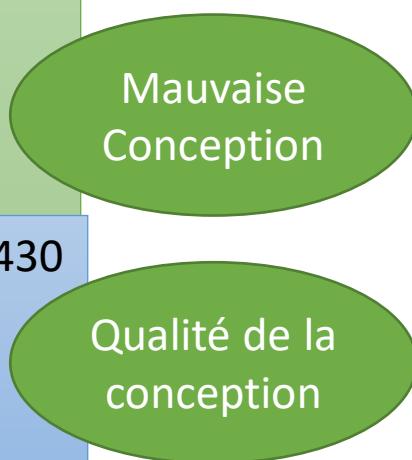
Architectures et systèmes des bases de données, et architectures du  
traitement des données

# Aujourd’hui

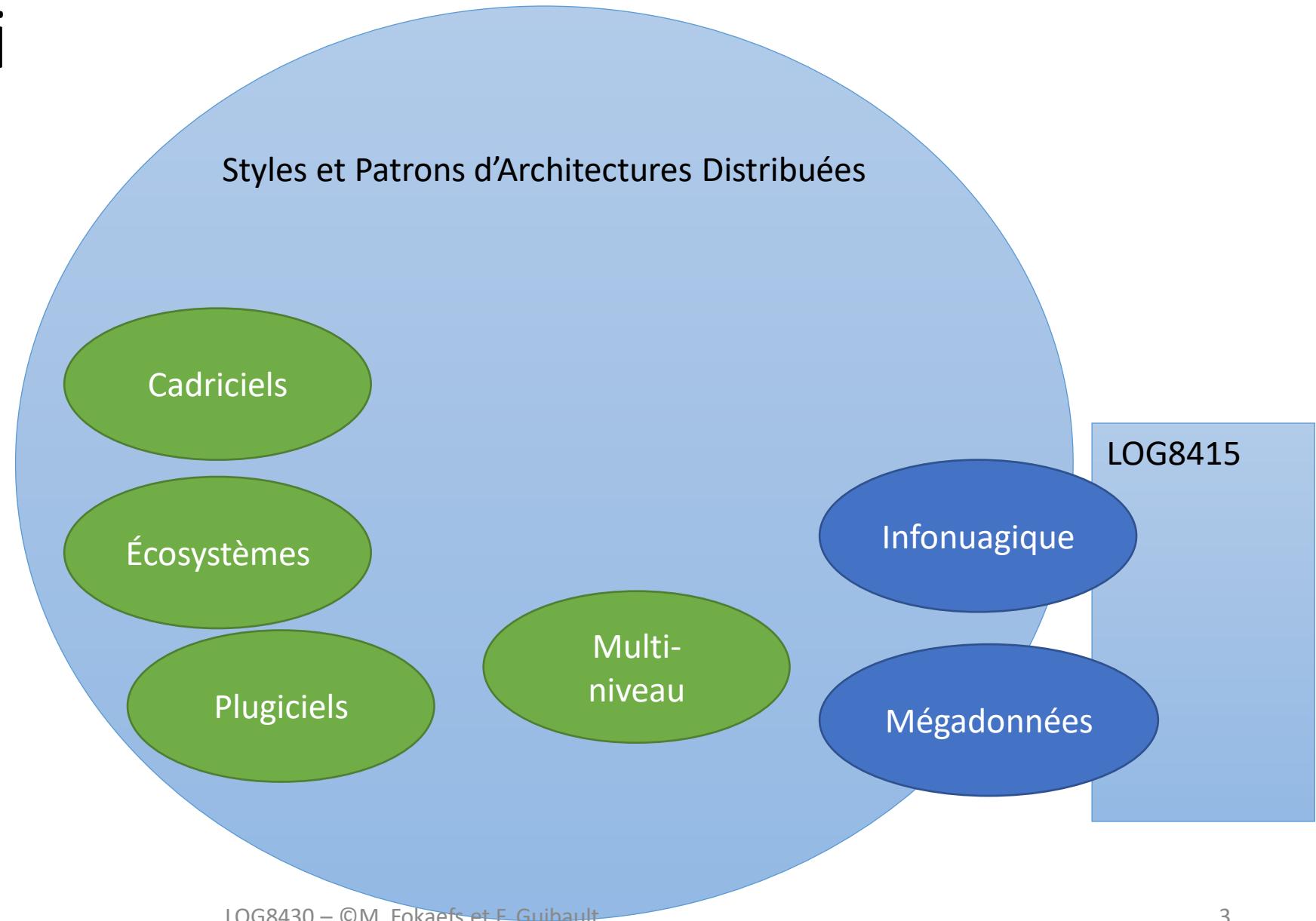
LOG2410

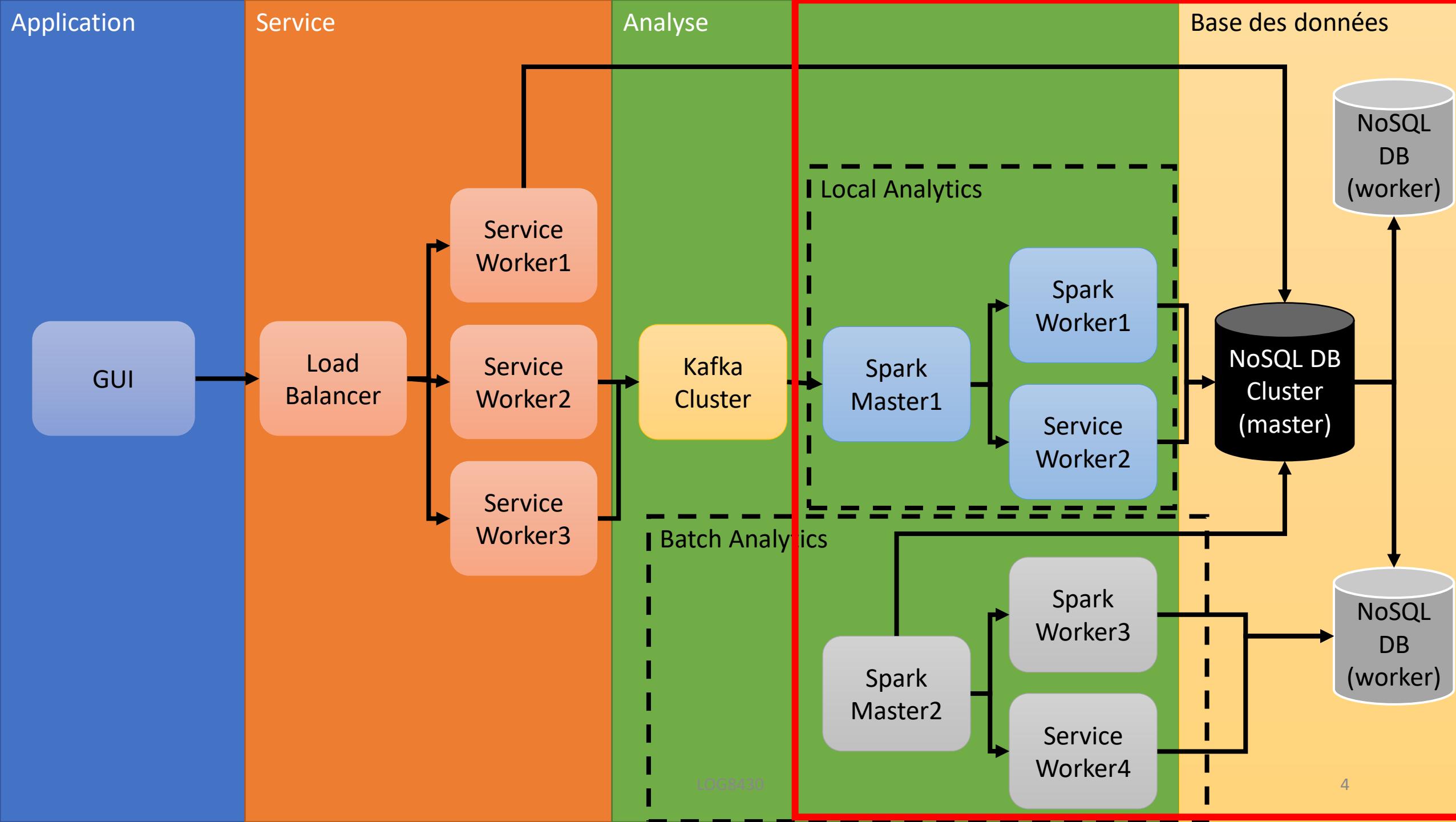


LOG8371

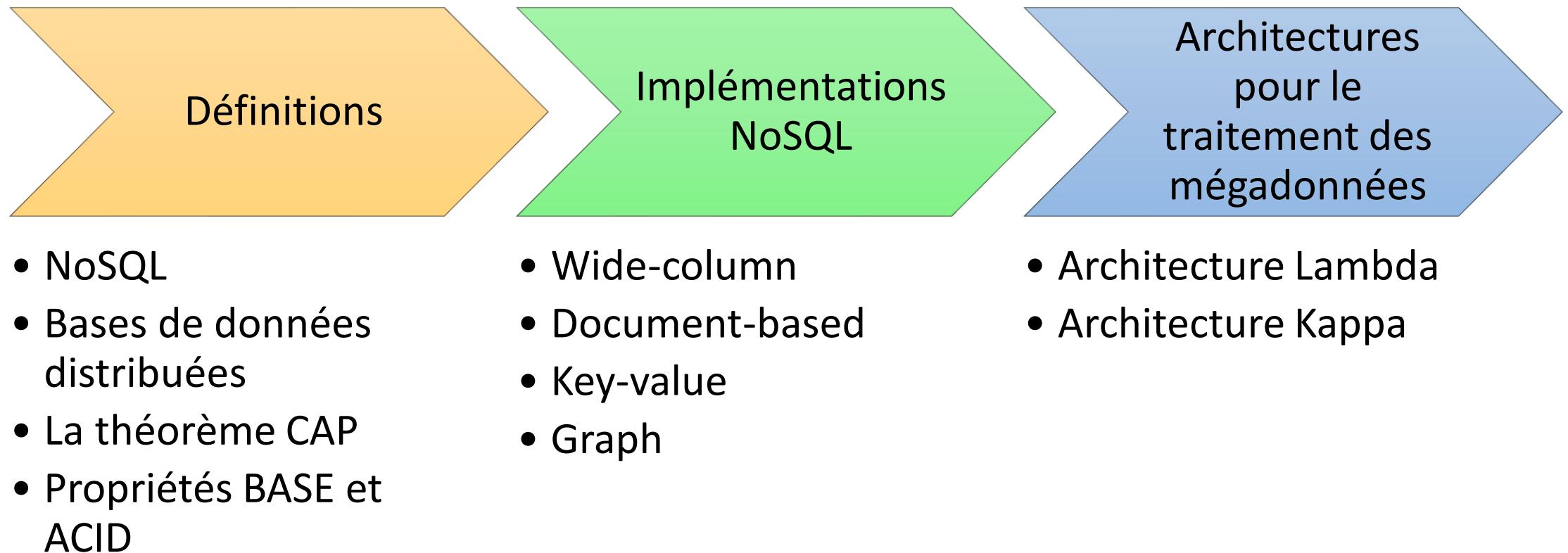


LOG3430





# Architectures logiciels des données

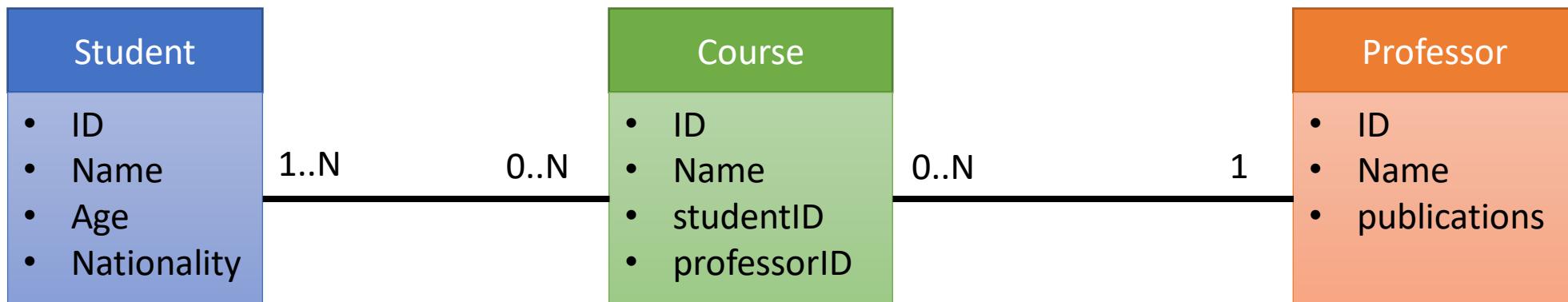


# Question

- 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

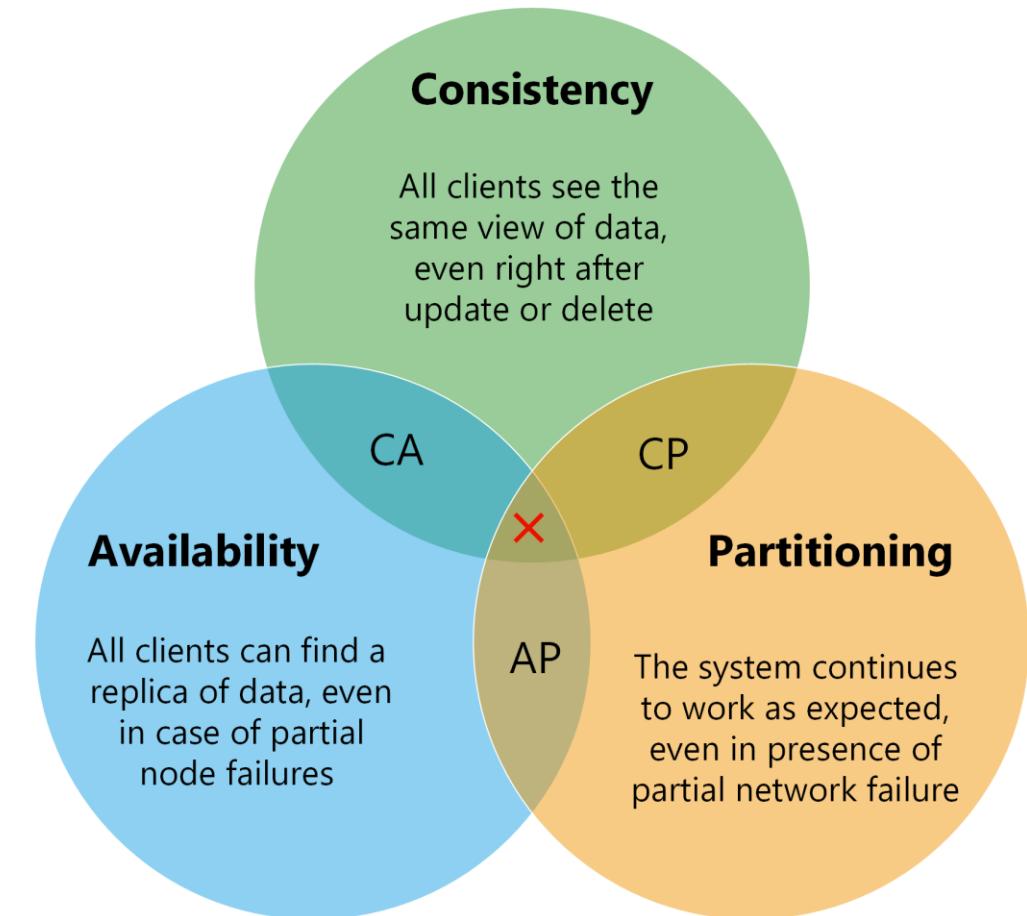
- Maintenant, ne parlons pas d'étudiants et de professeurs, mais parlons d'étoiles, de planètes et de systèmes planétaires.
- Il n'y a pas que des centaines d'objets, mais des milliards.
- Aussi, on a plus de types d'objets et des relations plus complexes.
- En explorant l'univers, nous en découvrons 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.
- **Consistency** : Tous les clients voient la même vue des données, même après une mise à jour.
- **Availability** : Tous les clients peuvent obtenir une copie des données.
- **Partition 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 garantie 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 est 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 persistant, 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 recherche 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.

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```



Embedded sub-document

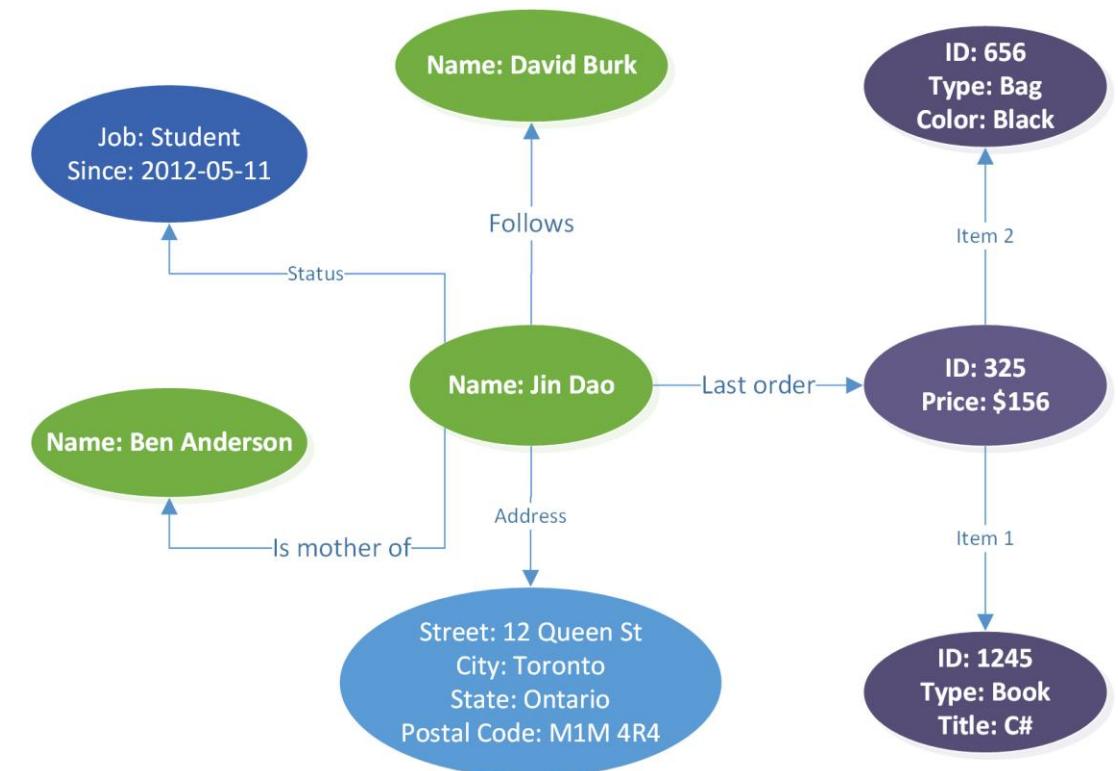
Embedded sub-document

# 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.

The diagram illustrates a row in a wide-column database. It is organized into three levels of hierarchy:

- super column family:** A large red-outlined box labeled "companies".
- column family:** A red-outlined box within the "companies" box, divided into two columns: "address" (containing "city" and "state") and "website" (containing "subdomain" and "domain").
- column:** Individual cells within the "address" and "website" columns, such as "San Francisco" under "city" or "grio.com" under "domain".

A vertical red box on the left is labeled "row key" and contains the value "1".

companies			
1	address		website
	city	San Francisco	subdomain
	state	California	domain
	street	Kearny St.	protocol

# 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?

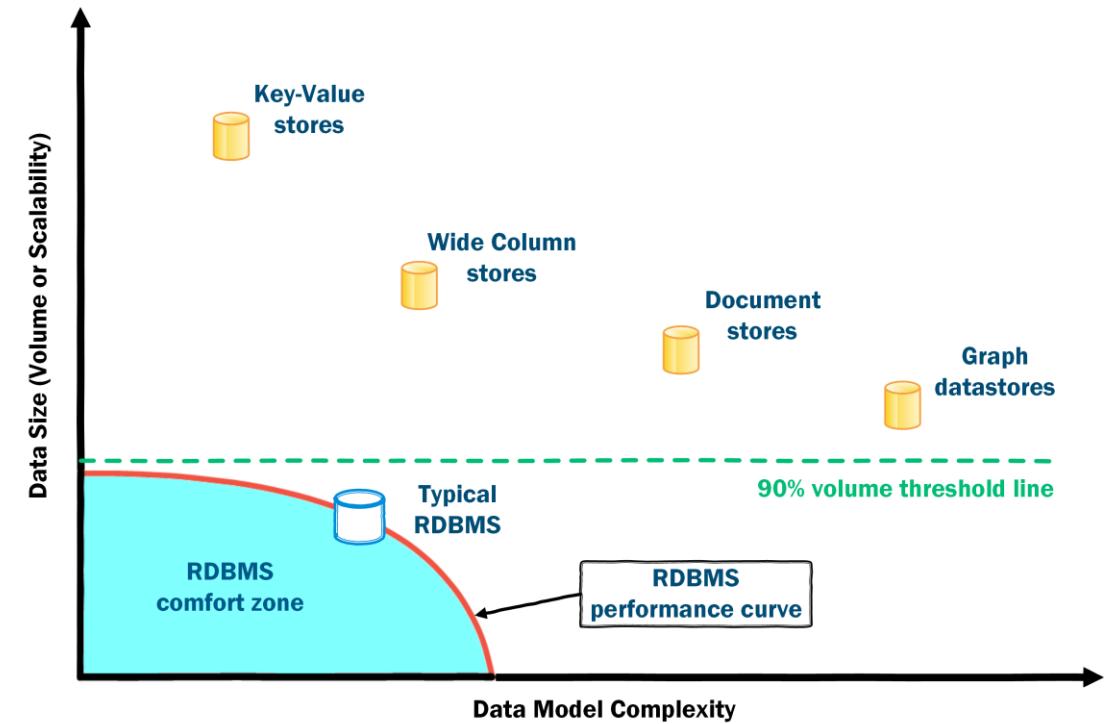
## Selon le type d'application

- 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

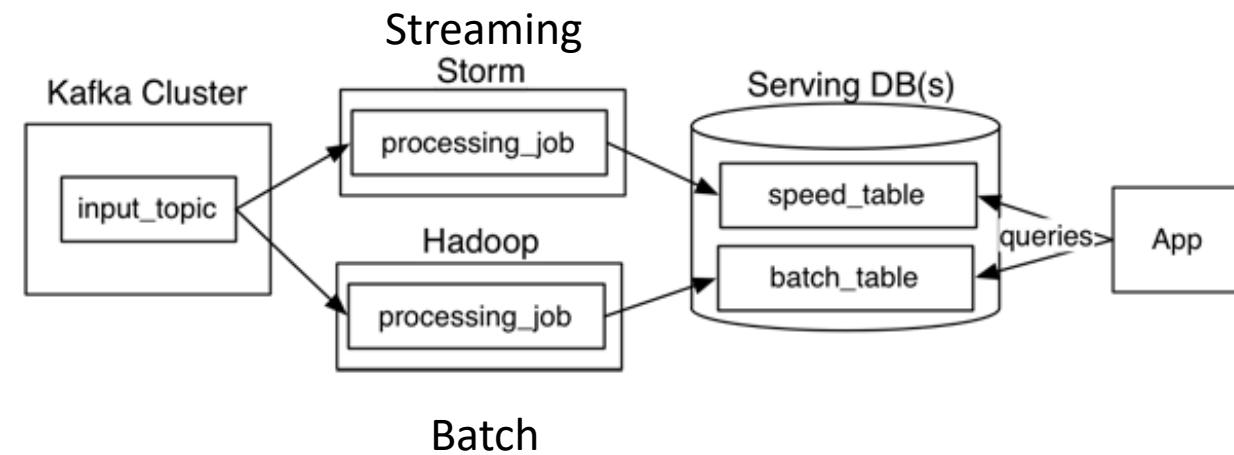
- Architecture Lambda
  - « How to beat the CAP theorem », octobre 2011
- Architecture Kappa
  - « Questionning the Lambda Architecture », juillet 2014

# 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 <sup>qui ne changent pas</sup>, 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

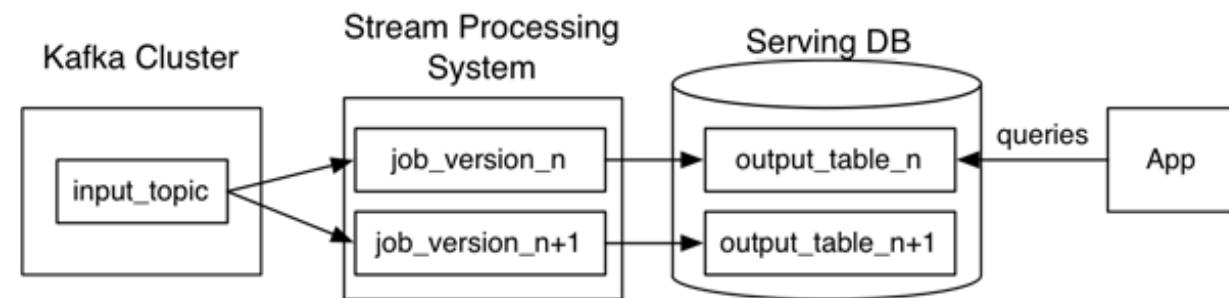
- Le codage peut être un processus assez compliqué.
- Le modèle de données peut être difficile à migrer ou à maintenir.
- La répétition du traitement de données (streaming et batch) crée une surcharge de performance.

# 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 persistant des données pendant longtemps.

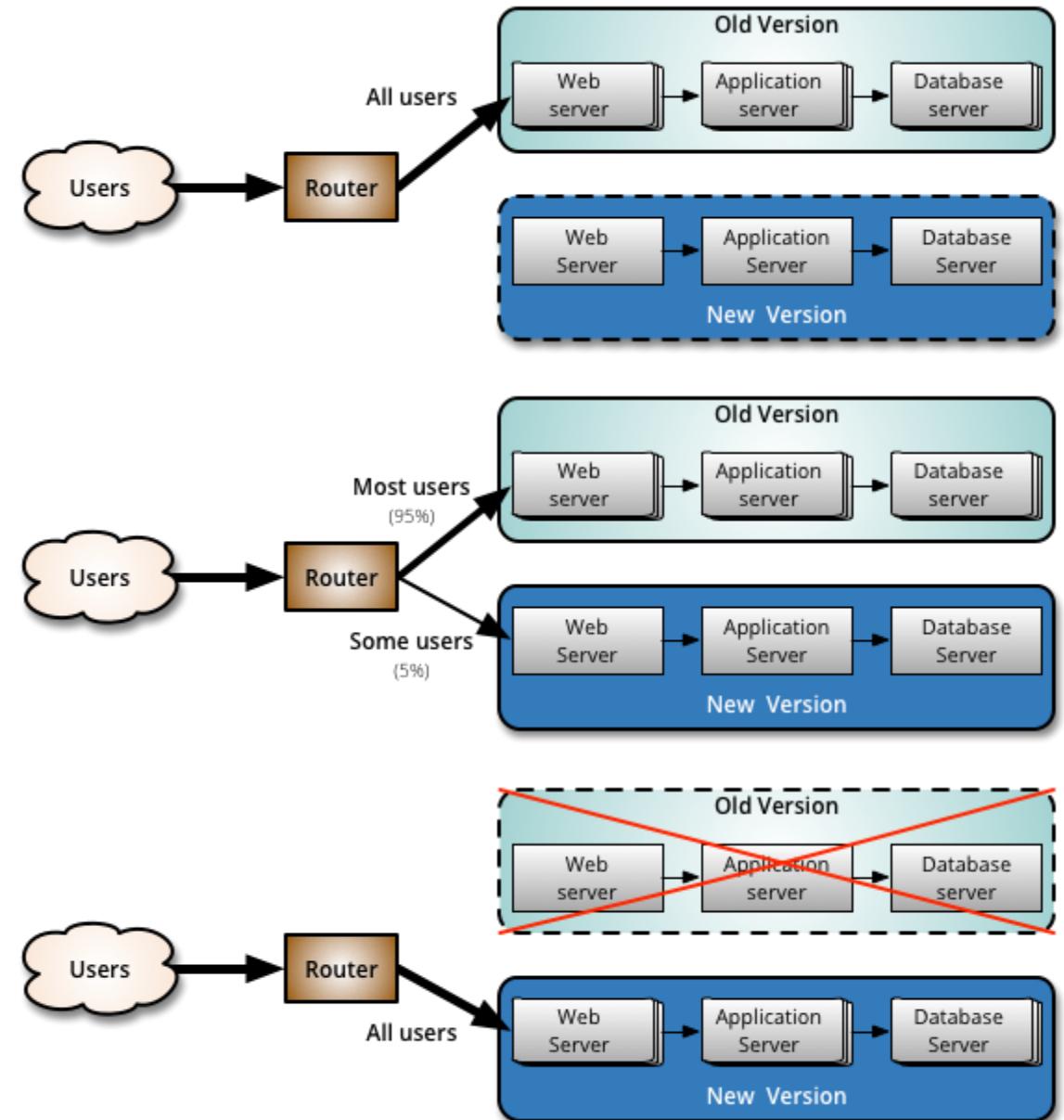
# Architecture Kappa

- 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 persistant.
- 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

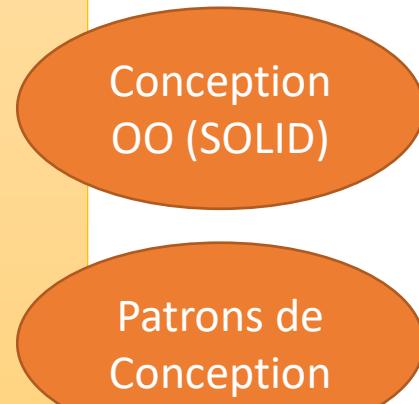
- Les demandes des utilisateurs doivent être servies sur une base ad-hoc en utilisant le stockage de données immuable.
- Des réponses rapides sont nécessaires et le système doit pouvoir gérer diverses mises à jour sous la forme de nouveaux flux de données.
- Aucunes des données stockées ne doivent être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

## Kappa

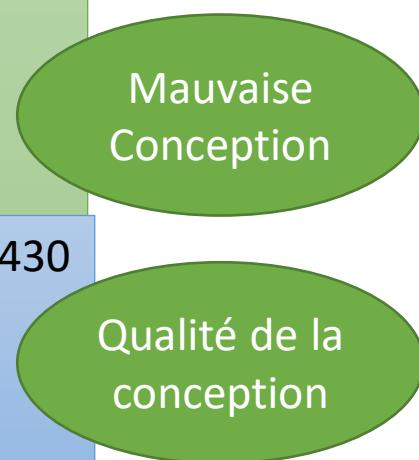
- Plusieurs événements ou requêtes sur les données sont mis en file d'attente pour être traités en fonction d'un stockage dans un système de fichiers distribué.
- L'ordre des événements et des requêtes n'est pas prédéterminé. Les plates-formes de traitement de flux peuvent interagir avec la base de données 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

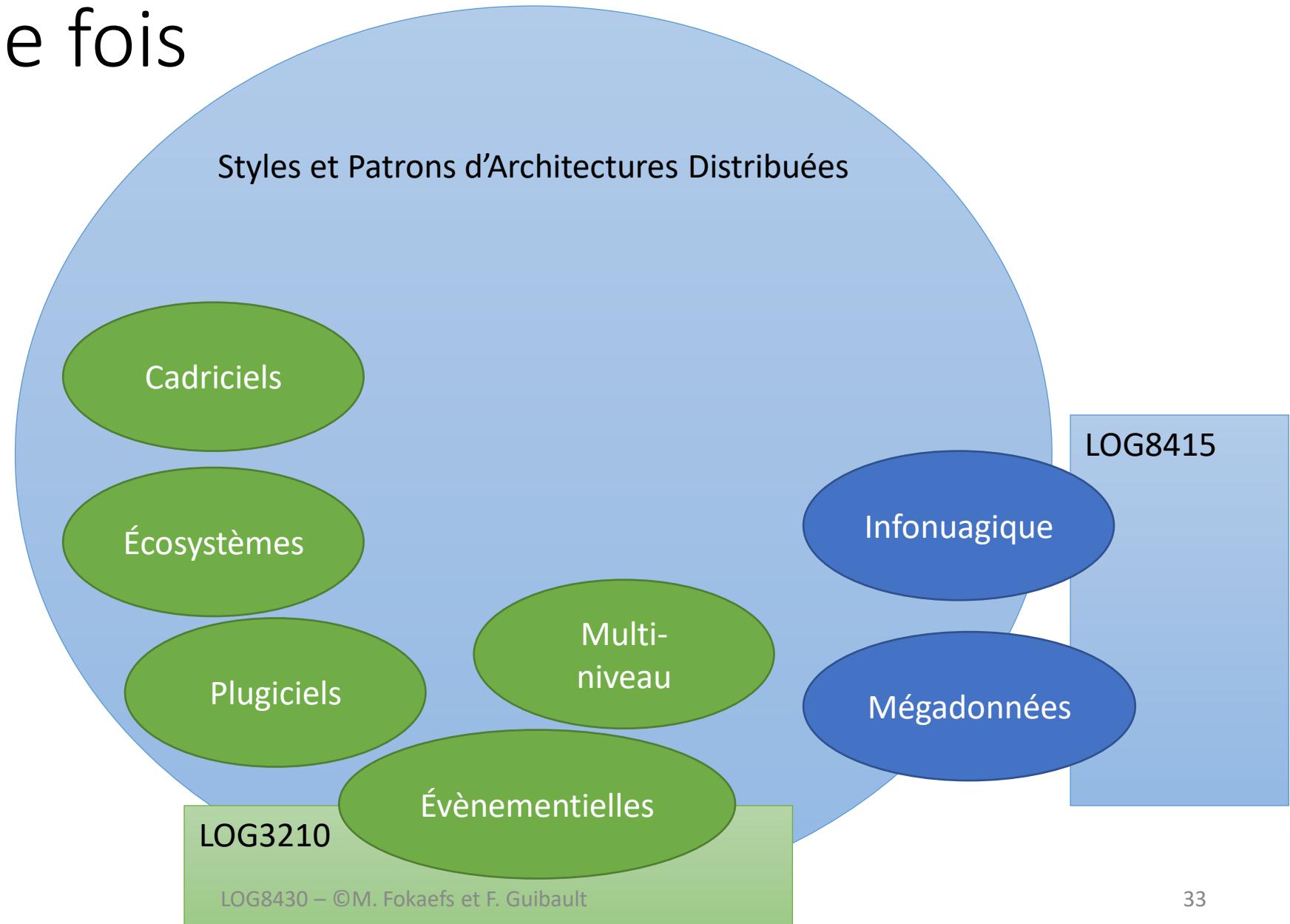
LOG2410



LOG8371



LOG3430



LOG3210

# LOG8430E: Blockchain

©M. Fokaefs et M. Rasolroveyic

# What is Blockchain?

- Blockchain acts as an immutable (permanent and unalterable), consensus-driven (trust verification), decentralized (networked copies) and transparent public record of data secured using a P2P (peer-to-peer) network on multi-clouds.
- Common use cases of Blockchain:
  - Healthcare and patients data storage
  - Tracking the transactions that will include an exchange of value between parties of a network.
  - Supply Chain and Food-traceability
  - Finance and Banking
  - Internet of Things
- First use case of Blockchain:
  - Bitcoin

# Definition:

- In a decentralized network, participants can agree on the state of the system without the need to either know or trust each other.
- In Blockchain, data can not be altered and we can only update the state of the system and it will keep track of all the transaction that have taken place in an immutable distributed digital ledger.



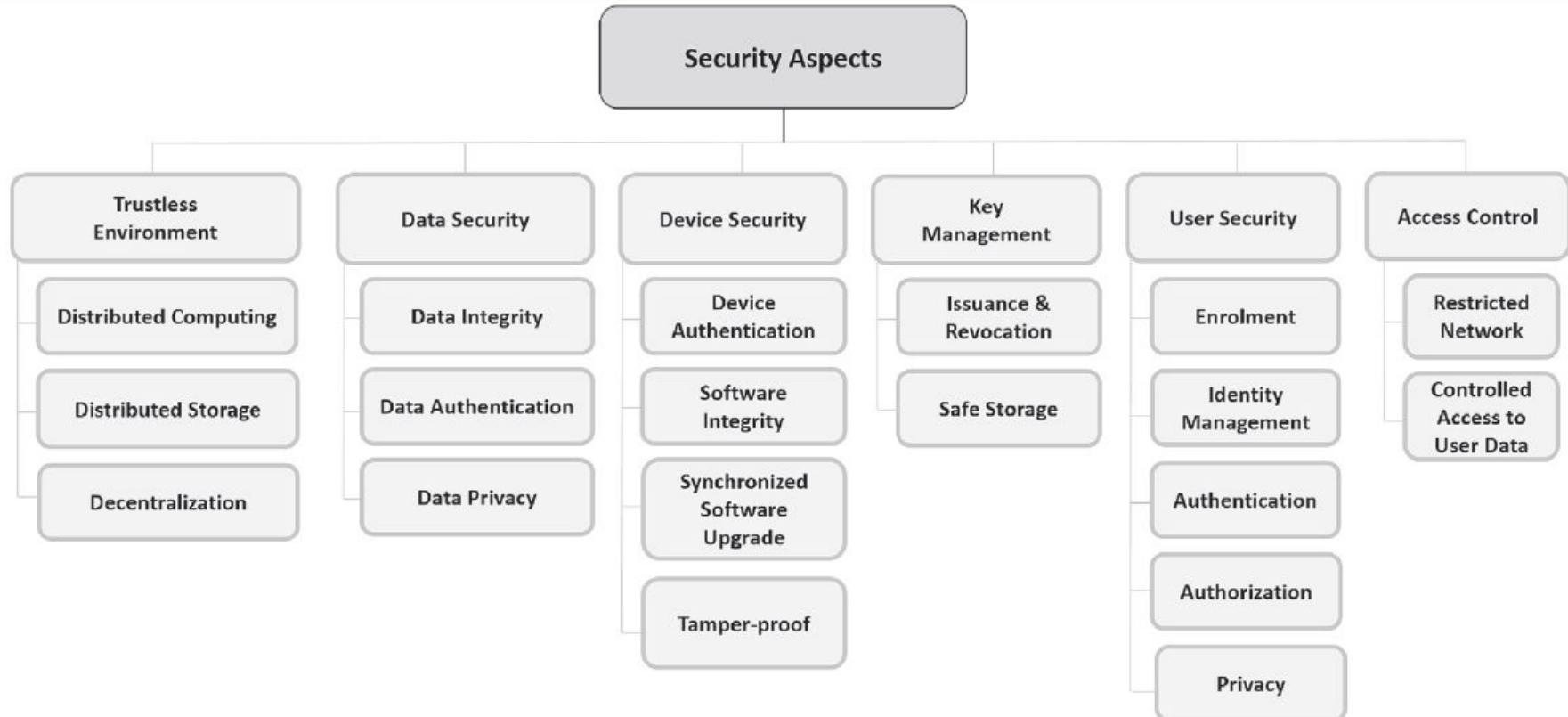
# The Benefits of Using Blockchain:

- Save time for inter-bank transfer
- Guarantee Confidentiality, Integrity and Availability (CIA) of the system
- Improving traceability and transparency. (All the data are visible to network participants)
- Reducing cost by removing central intermediaries



src: <https://www.f5.com/labs/articles/education/what-is-the-cia-triad>

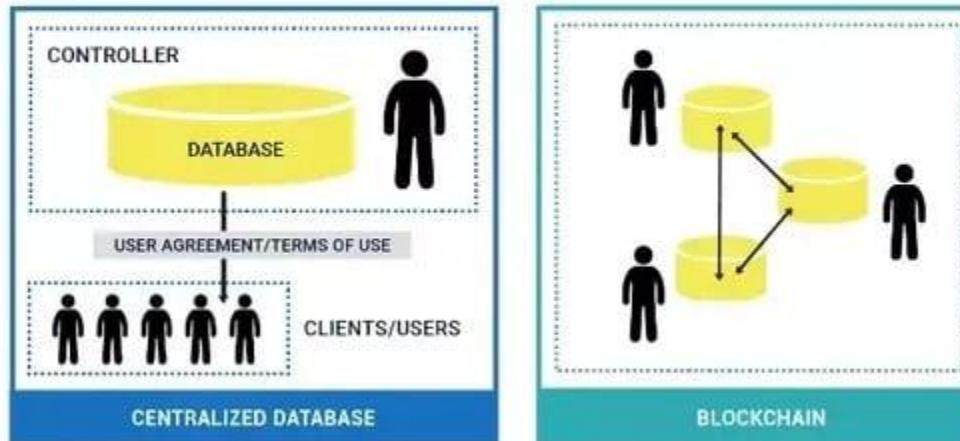
# Security aspects of Blockchain



What is the main difference between  
Blockchain and traditional databases?

# Blockchain VS. Database

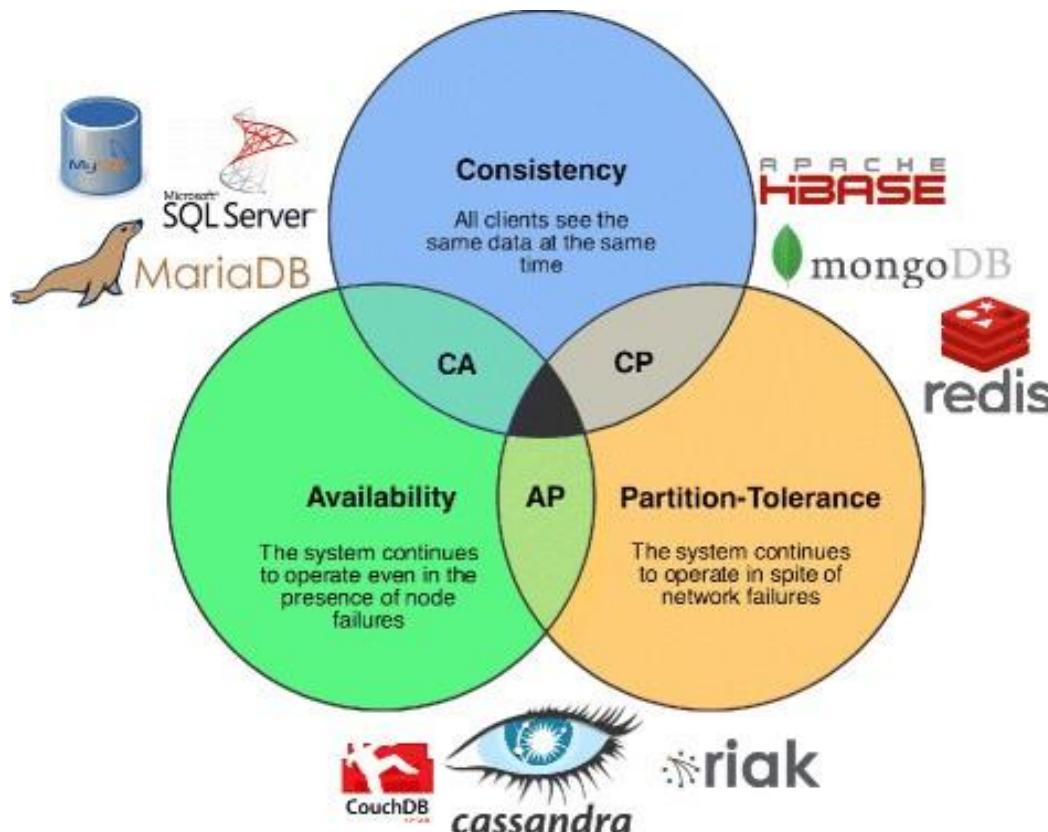
## CENTRALIZED DATABASES VS. BLOCKCHAIN



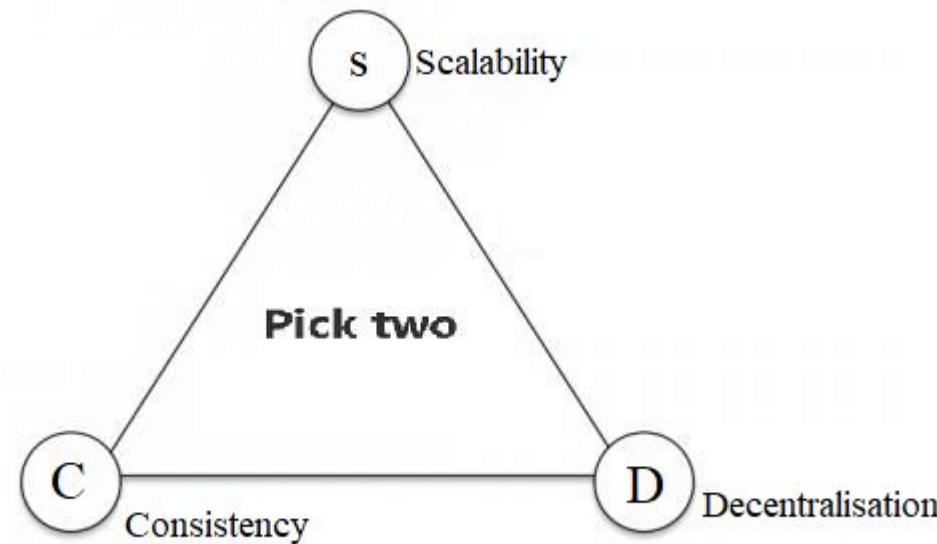
Database: Centralized, Permissioned and Requires Admin

Blockchain: Decentralized, Permissionless and No Admin

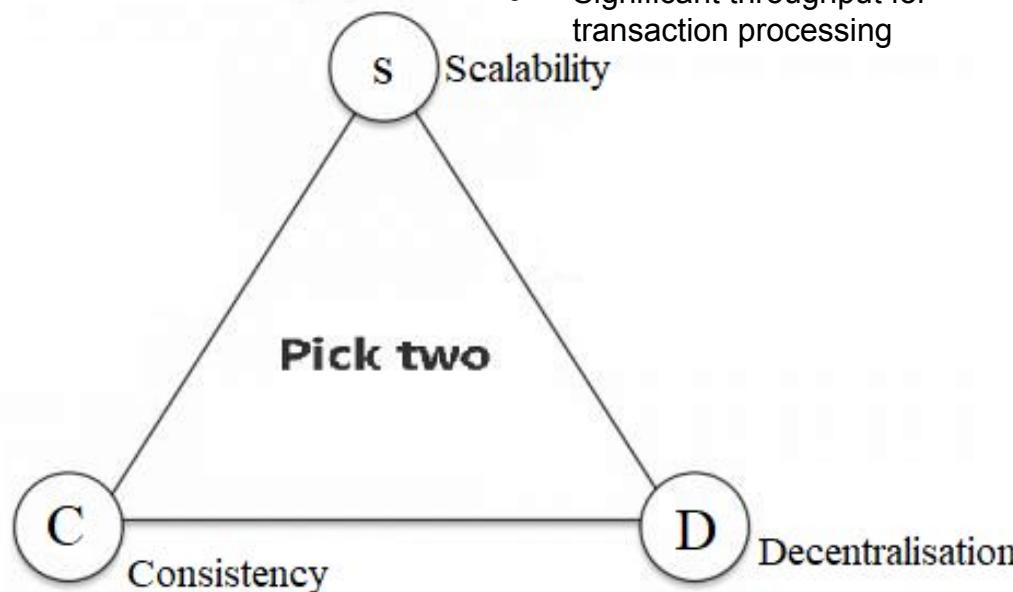
# CAP Theorem



# The Blockchain Trilemma:



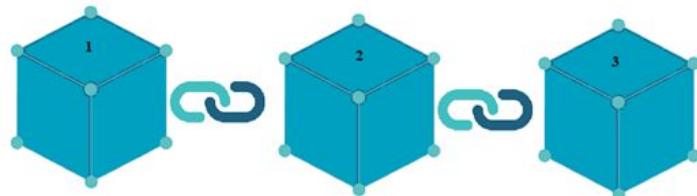
# DLT Theorem:



- Timestamping
- Data immutability and integrity
- Security and Fault Tolerance

# Why Blockchain is tamper-proof?

- If an adversary alters the Block K, this will produce a new hash which is different the one in Block K+1.
  - As a result, it will invalidate the link between Block K and Block K+1



Hash: **1B7G**

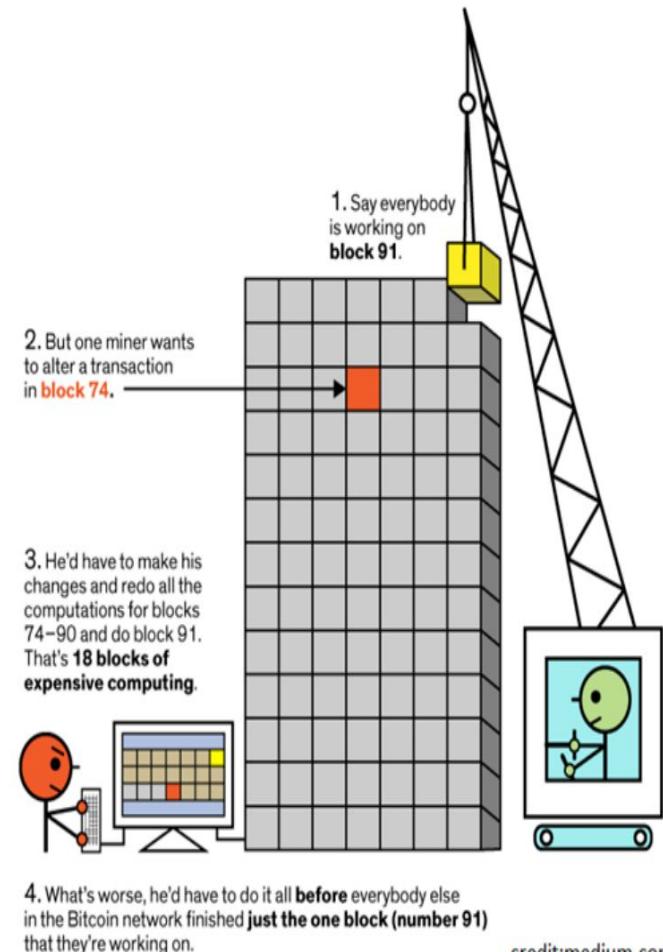
Previous hash: **0000**

Hash: **P54U**

Previous hash: **1B7G**

Hash: **CF45**

Previous hash: **P54U**

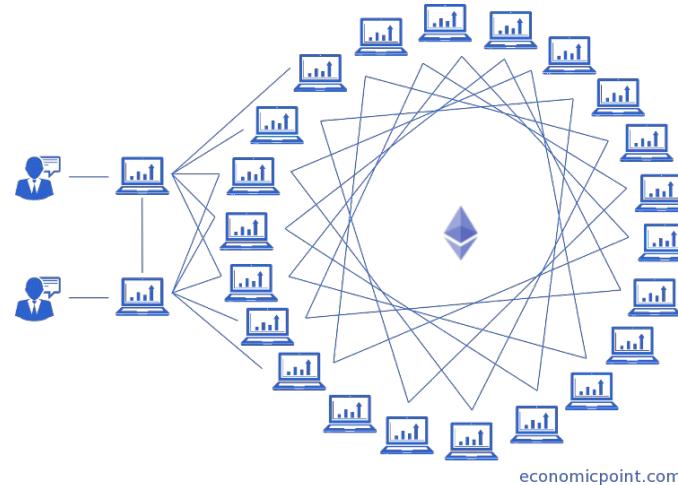


credit:medium.com

# What makes Blockchain tamper-proof?

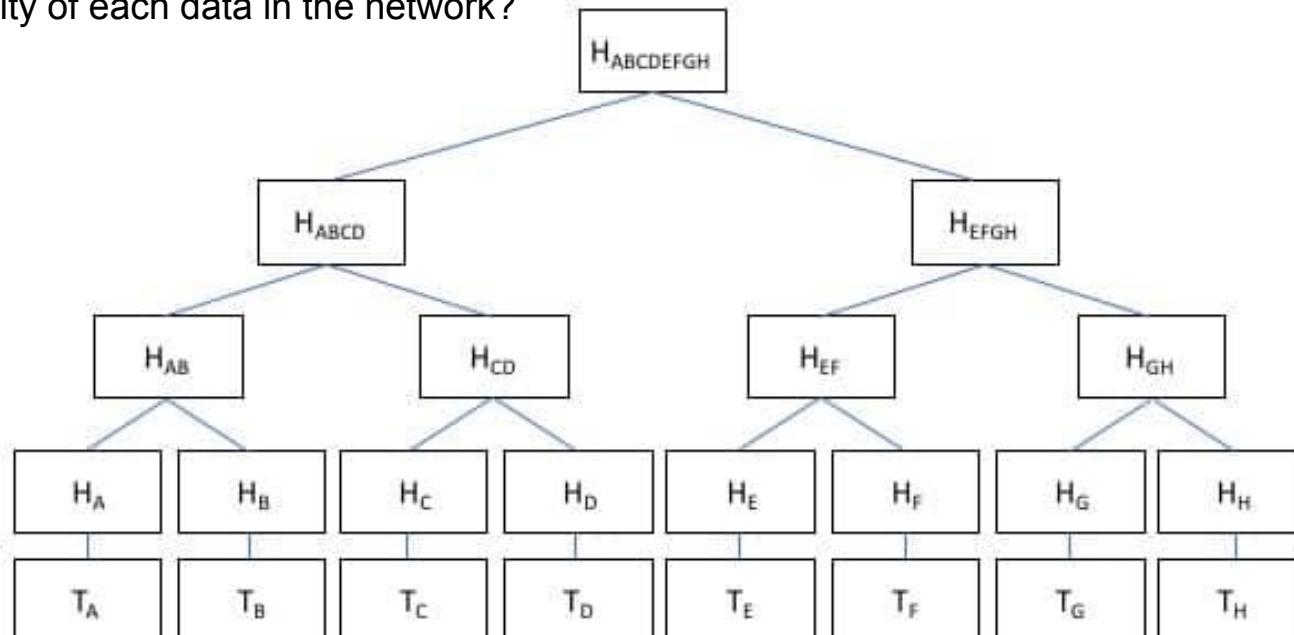
**Consensus protocol:** There is a **unanimous acceptance** between participants of the network for every change or new submission.

**Cryptographic fingerprint:** Called hash, it takes a lot of computing power and energy to generate initially.



# Merkle Tree:

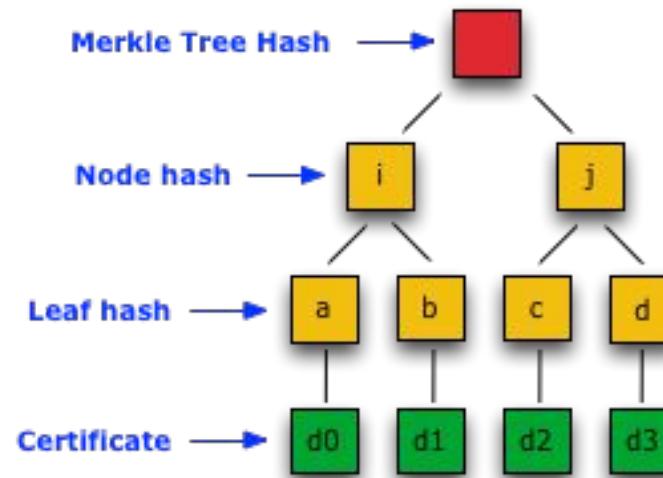
- Is there a way to recognize any tampering in the Blockchain?
- What approach is needed to check the validity and authenticity of each data in the network?



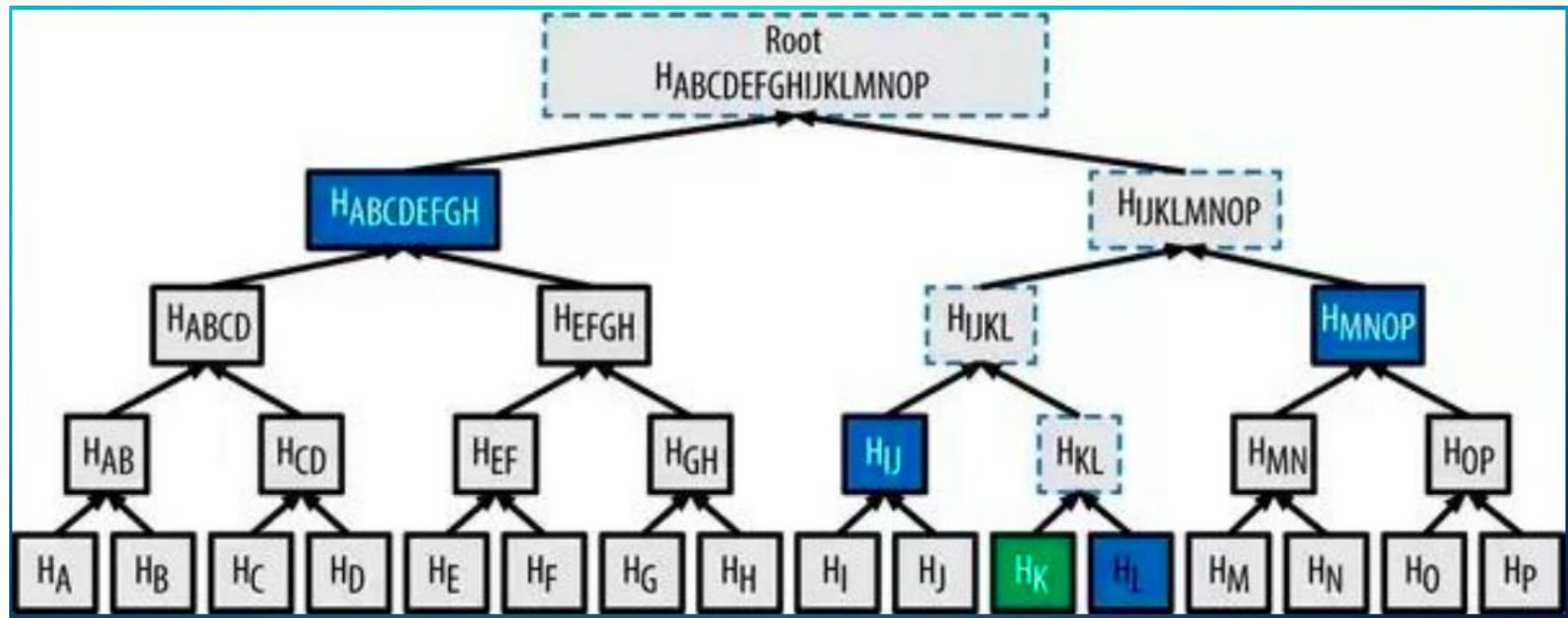
Source: [bitcoininsider.org](http://bitcoininsider.org)

# Merkle Tree: Proof of Membership

- How can we audit the block D3?



# Merkle Tree: Proof of Membership

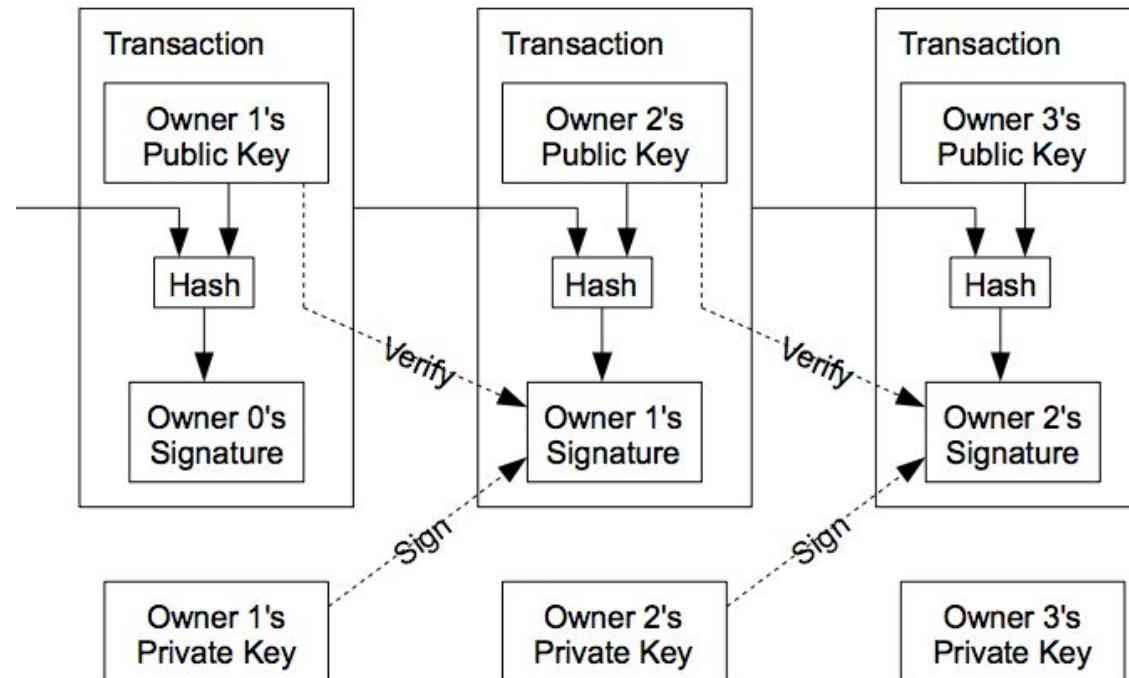


source: medium.com

## Merkle Tree: The procedure of transaction verification in Blockchain

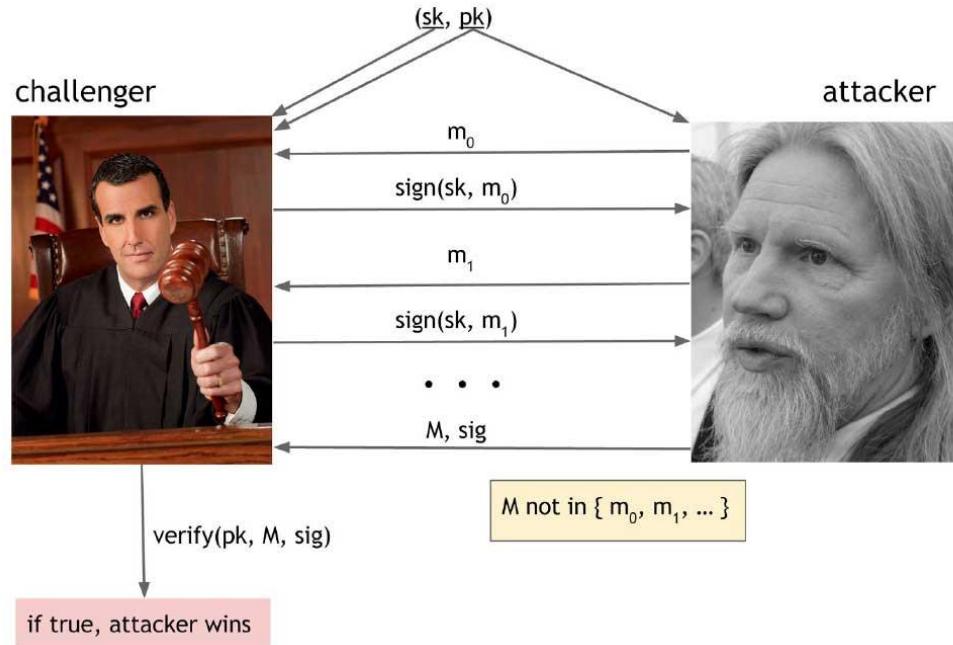
- Merkle Tree enables users to authenticate for themselves when a transaction has been inserted to the network blocks.
  - By tracing the Merkle root which is retrieved from block header.
- In order to authenticate and verify the Transaction A is added to the Block, the user only need few hashes including merkle root.
  - There is no need for client to know about any other transactions.
  - This will save significantly in terms of transmission bandwidth and delay.

# Digital Signature in Blockchain



# Digital Signature: Unforgeability

“An adversary who knows your public key and your signatures on some messages cannot can neither forge nor generate your signature on some other message.”



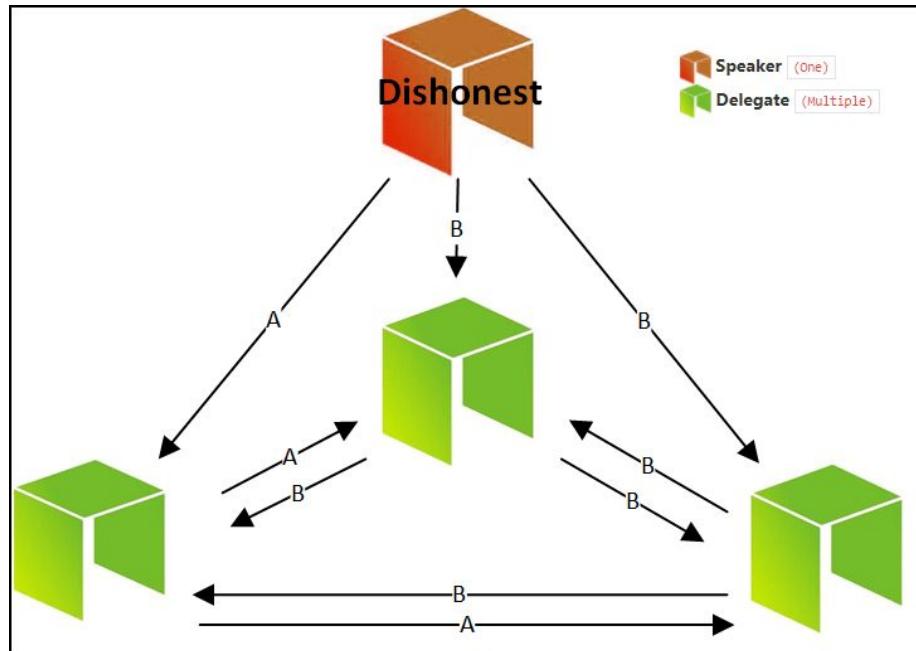
# Consensus Protocol:

There  $n$  nodes in the network which have an input value. Some of these nodes could be either faulty or malicious. The consensus protocol will have the following two steps:

- 1) Terminate all honest nodes in agreement on the value
- 2) The value must be only generated by an honest node

# Consensus

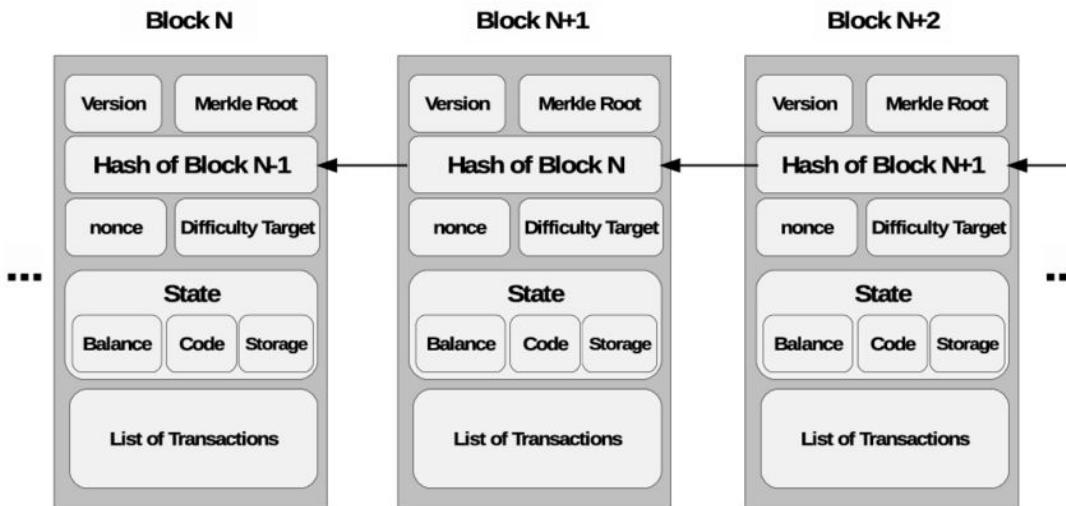
In Blockchain network, nodes need to agree on exactly which transaction were broadcast and the order in which these transactions happened.



source: [blockgeeks.com](http://blockgeeks.com)

# Consensus

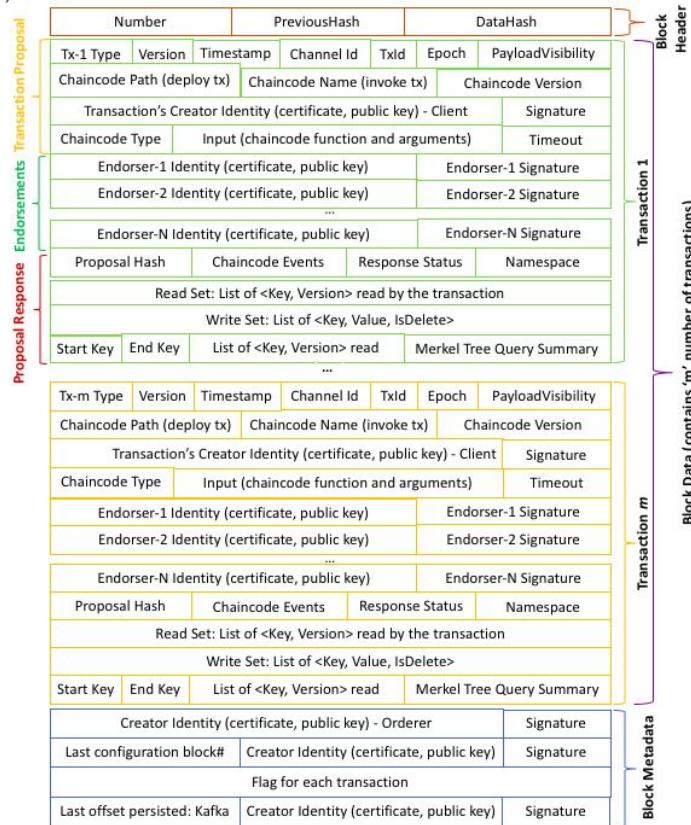
- In Blockchain, all the peers must agree on exactly which transaction were broadcast and the order in which these transactions happened.
- All the times, the participants of the P2P network own a ledger which contains of a sequence of blocks, each consists a list of transactions that they have approved.



source: Khaled Salah et al.

# Consensus

- Network participants in Blockchain network might have different copies of the ledger and the transactions which also includes the list of pending transactions. (Why?)



# Consensus

- Does each node have the same list of transactions that are broadcasted?
- Does each node have the same ledger?

# Byzantine Generals Problem:



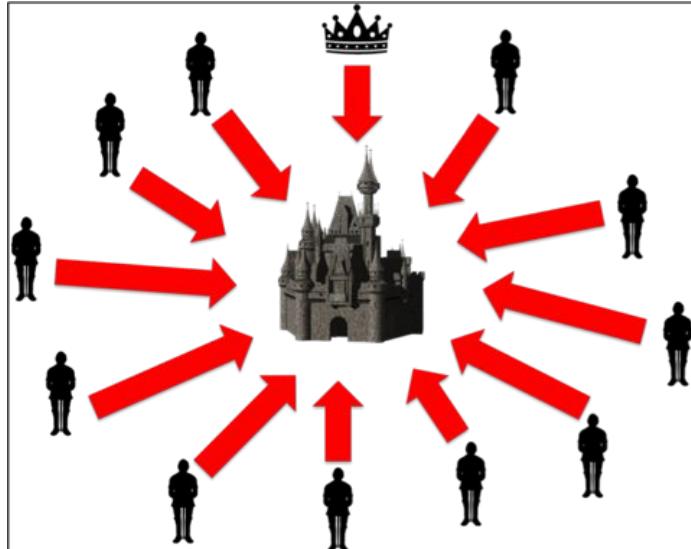
# Byzantine Generals Problem:

- There is an army which consists several generals.
- Generals communicate via messengers.
- There are generals who are traitors.
- **Our goal:** The honest generals arrive at the same decision (either attack or retreat). The traitorous general will not be able to impose false decision to honest generals.

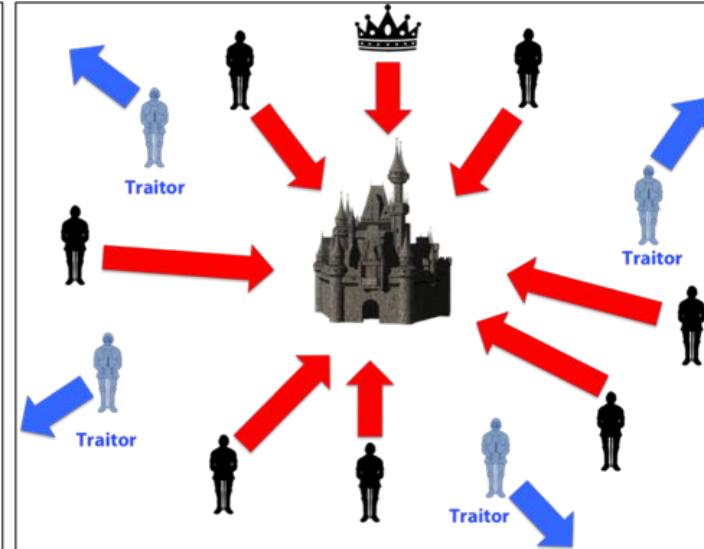
How?

# Byzantine Generals Problem:

- If 1/3 or more are traitors it will be impossible.
- $(3f+1)$  where  $f$  is the number of traitorous generals.



**Coordinated Attack Leading to Victory**



**Uncoordinated Attack Leading to Defeat**

source: ivanotech.com

- How many nodes we need for BFT?

# Blockchain Consensus Protocol: Proof of Work

- Incentives: Nodes will have incentives to reach to the consensus (getting rewarded)
- Randomness : The nodes will be selected in a random order, without any central authority.
- The nodes are following pseudo-anonymity which means that they don't have to be persistent.
- There is neither starting point nor eding for to reach to the consensus. This could have for a long period. (eg., 1 hour or more in Bitcoin network)
- Do all the nodes need to participate in the consensus?

# Blockchain Consensus Protocol: Proof of Work

How the coins can not be stolen in Blockchain in example of Bitcoin?

“• If Alice wants to steal bitcoin (transfer Bitcoin from the bitcoin owner to her address), it needs to sign the transfer transactions using the private key of the owner (which she does not have!)”

# How Blockchain can avoid Denial of Service Attacks?

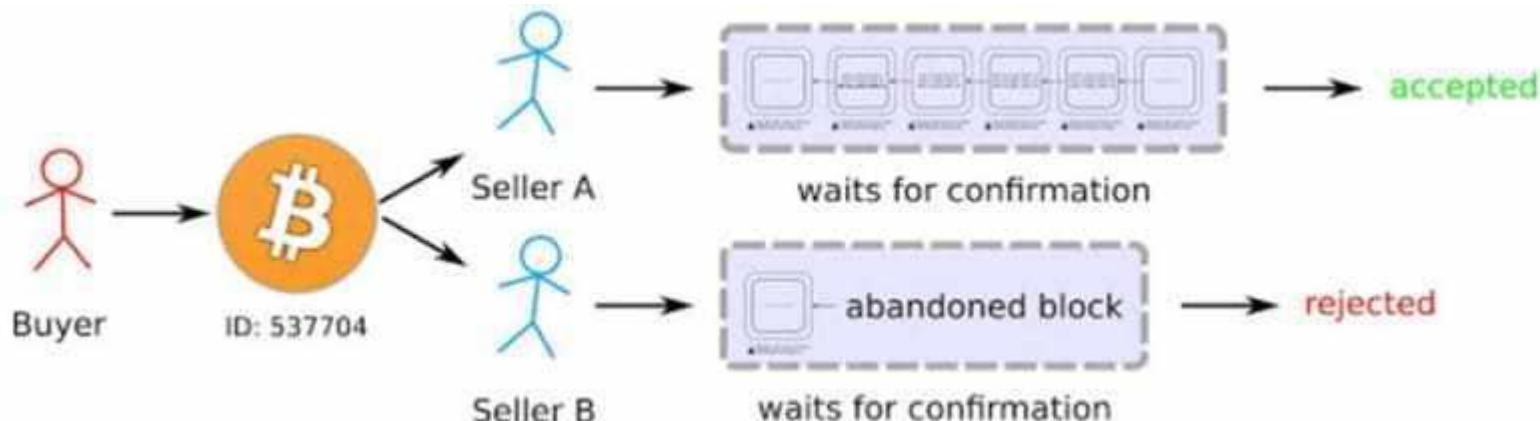
- Alice can block the inclusion of a transaction (of Bob) in any block she creates the transaction will remain in the the network.
  - This means that it will be included in the next block which is created by another node.
- The most critical point is the random selection of the participating nodes in the protocol.

# How Blockchain can avoid double-spending attacks?

- Alice buys a video game from a Bob.
- The transaction will be broadcasted (payment instruction, hash, Alice's signature).
- Hash is the pointer to some previous transaction. (for example where Alice has received Bitcoin).
- Transaction will be included in the Block.
- Bob lets Alice to download the video game. However.
- Alice will use the same Bitcoin to buy a product from Josh.
- The transaction will be broadcasted (Hash, ALice's signature and payment instruction).
- Transaction will be included in the next Block.
- It happens that Alice is the node selected to create this Block.
- She ignores the block that has been created with Bob's transaction and starts from previous one.
- Now we have two chains. (Alice to Bob and Alice to Josh)

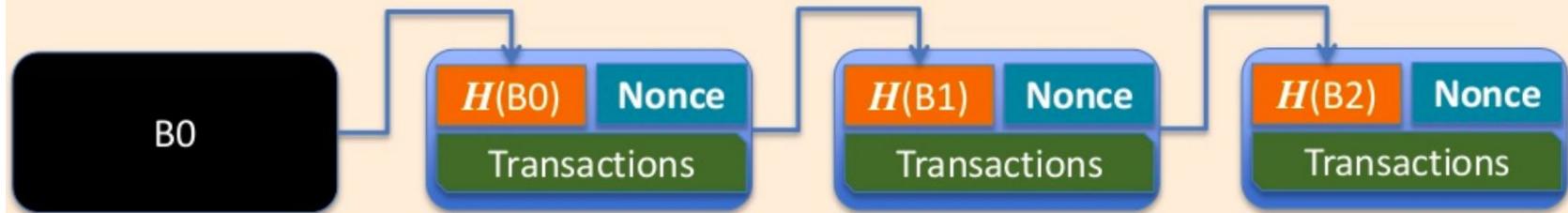
# How Blockchain can avoid double-spending attacks?

- Which transaction will be succeed?
- 



source: coinsutra.com

# Bitcoin's Proof-of-Work



Find a nonce  $x$  such that:

$$\text{SHA-256}(\text{SHA-256}(r \parallel x)) < T/d$$

$r$  = header      includes  $H(\text{previous block})$   
                          root of Merkle tree of transactions

Source: David Evans, University of Virginia

# How Blockchain can avoid double-spending attacks?

- The node that will create the next block will “determine” which transaction will be included.
  - The nodes will only build on one of the chains.
- The nodes will only agree to longest and most difficult chain to reach consensus.
- Can we have 2 or more blocks that are different but VALID at the same time in the network?

# What are the options for Bob?

- Allowing the Alice to get the video game as soon as Alice has broadcasted the transaction?
- Allowing Alice to get the video game as soon as the transaction is confirmed in a new Block?
- Allowing Alice to get the video game after certain amount of confirmation (which means that several block are appended to the block that includes the transaction to Alice to Bob)?
- It is recommended that we wait for 6 confirmation in Bitcoin.

# How Blockchain can avoid double-spending attacks?

- Double spending has nothing to do with cryptography since both transactions are considered to be valid authentic and cryptographically valid.
- It the consensus which will determine to approve which transaction should be included in the Blockchain.
- But, it does not guarantee which one will be included.
- It needs several confirmation to decrease the probability of the double spending transaction.

# The Vulnerability of proof-of-work in Blockchain:

- If the adversary take the control 51% resources, they can be able to make decision on behalf of the all participants.



# Why it doesn't make 51% in Blockchain?

- It is expensive to control 51% of hash power around the world participants.
- Not only the hardware, but also the electricity consumption.
- Bitcoin network is currently consuming more electricity than Ireland.
- More importantly, it makes zero sense from financial point of view.
- The attacker will be self-destroyed: Bitcoin value will go to zero.
- Blockchain miners have spent hundreds of millions of dollars on the infrastructure required to compete in the mining sector.



# Example a Bitcoin miner:

[Retour aux résultats](#)



Cliquez pour afficher une image agrandie

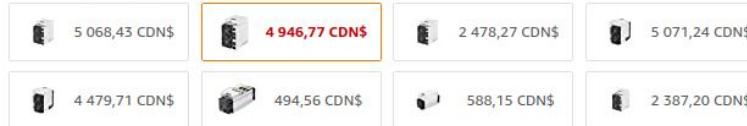
## DragonX AntMiner

[Visitez le Store DragonX](#)

5 évaluations

Prix : **4 946,77 CDN\$** + 3,95 CDN\$ Livraison

Couleur: **S17 59T**



- Bitcoin Mining Hash Rate: 59TH/s ±5%, with psu
- Power Consumption: 2385W ±10%, Extremely Efficient Bitcoin / Bitcoin Cash Miner
- Built-in web management portal, build-in power supply
- Please be aware that AntMiner S17 59TH is not accept return and refund, the manufacturer will provide warranty for 180 days, if you return within 20 days, we will charge 40% restock fee.
- More USED Miners and hosting service available

Avez-vous besoin de plus d'informations sur les produits en français? Contactez-nous

**4 946,77 CDN\$**

+ 3,95 CDN\$ Livraison

Livré : **20 - 28 oct.**

**En stock.**

Quantité: **1**



[Ajouter au panier](#)



[Acheter maintenant](#)



[Transaction sécurisée](#)

Ships from Canada and sold by [lighting-geek](#).

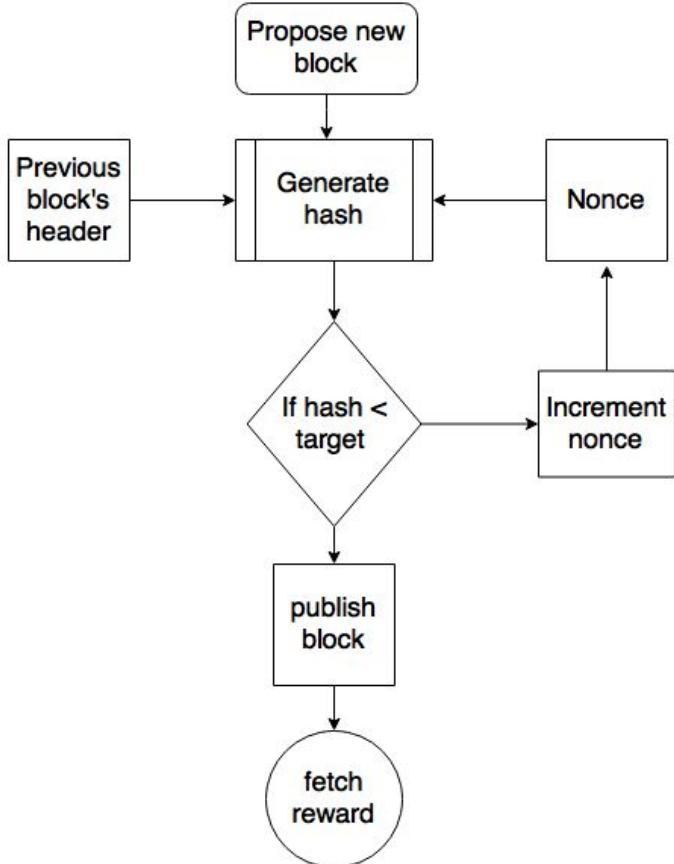
[Entrez votre adresse](#)

[Ajouter à votre liste d'envies](#)

# The mining algorithm:

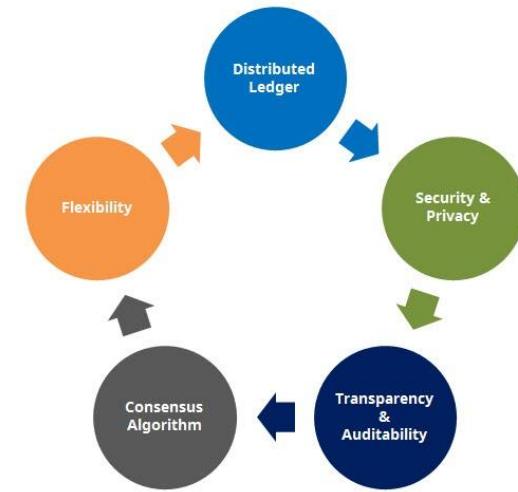
The mining algorithm consists of the following steps:

1. The previous block's header is retrieved from the bitcoin network.
2. Assemble a set of transactions broadcasted on the network into a block to be proposed.
3. Compute the double hash of the previous block's header combined with a nonce and the newly proposed block using the SHA-256 algorithm.
4. Check if the resultant hash is lower than the current difficulty level (target) then PoW is solved. As a result of successful PoW the discovered block is broadcasted to the network and miners fetch the reward.
5. If the resultant hash is not less than the current difficulty level (target), then repeat the process after incrementing the nonce.



# Key characteristics of Blockchain

- Permissionless
- Trustless
- Transparent
  - It is visible to anyone but difficult to realize Immutable (When transaction confirmed, it can neither be manipulated nor removed)
- Immutable
  - It is computationally infeasible to either change or remove.
- Secure (Unhackable)



## Some characteristics of Bitcoin (summary)

- Block added each 10 minutes
- Uses SHA-256 as a hash function
- Uses ECDSA for signature
- Miners are mainly paid by minted (created) coins (bitcoins created by new block)

# Ethereum



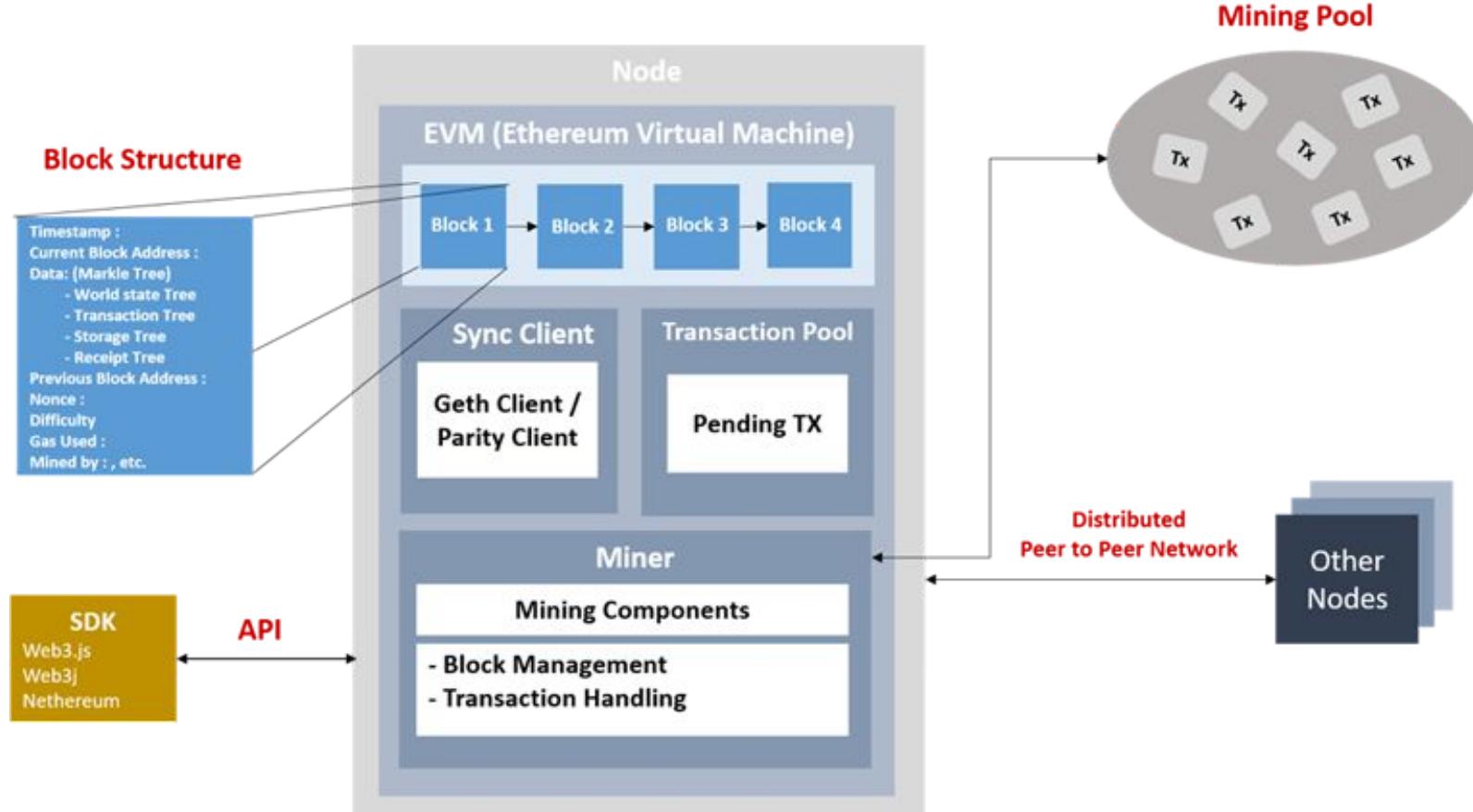
# Ethereum

- In general, Ethereum Blockchain is designed a public network. (Based on PoW, P2P and cryptographic primitives)
- In compare to Bitcoin, Ethereum is not a digital currency payment system.
  - Ether is utility which is used to pay for the use of Ethereum platform.
- Ethereum is a general-purpose programmable Blockchain that runs a virtual machine (EVM).
  - Ethereum's language is Turing complete (Solidity) compared to the basic scripting used in Bitcoin.
  - Any program of any complexity can be computed by Ethereum.

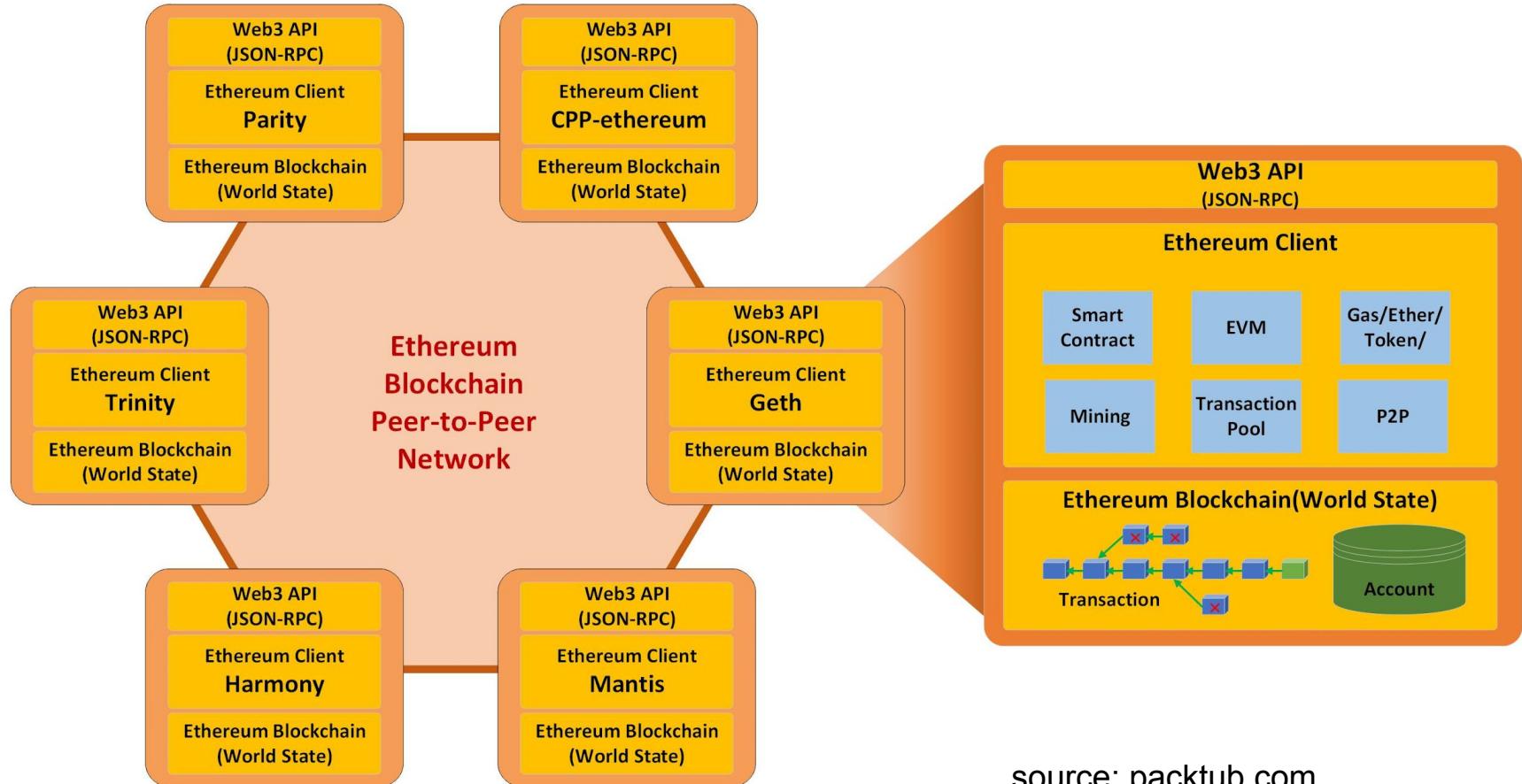
# Programming in Ethereum

- In Ethereum, each instruction (e.g., computation, data access, data storage, etc.) has a predetermined cost in units of gas.
- In order to submit a transaction in Ethereum, the programmer must include an amount of "gas" that can be consumed.
  - The code execution will be aborted if the amount of the gas consumed exceeds the gas available in the transaction.
  - When the transaction completes any unused gas is sent back to the sender of the transaction.
  - If you don't send enough gas, what you sent is still lost, because the miner still tried doing the work, so will take it and your transaction will never been confirmed. However, all the operations will be reverted.\*\*

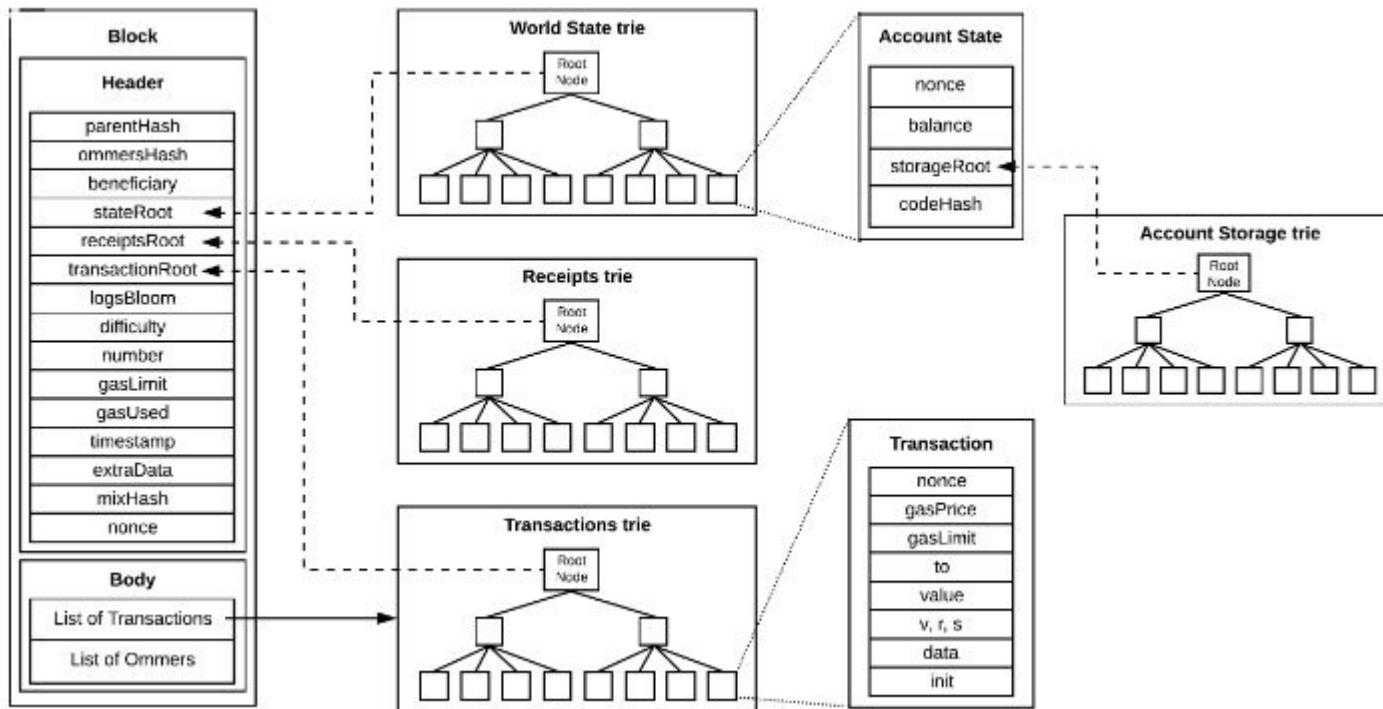
# Ethereum Architecture



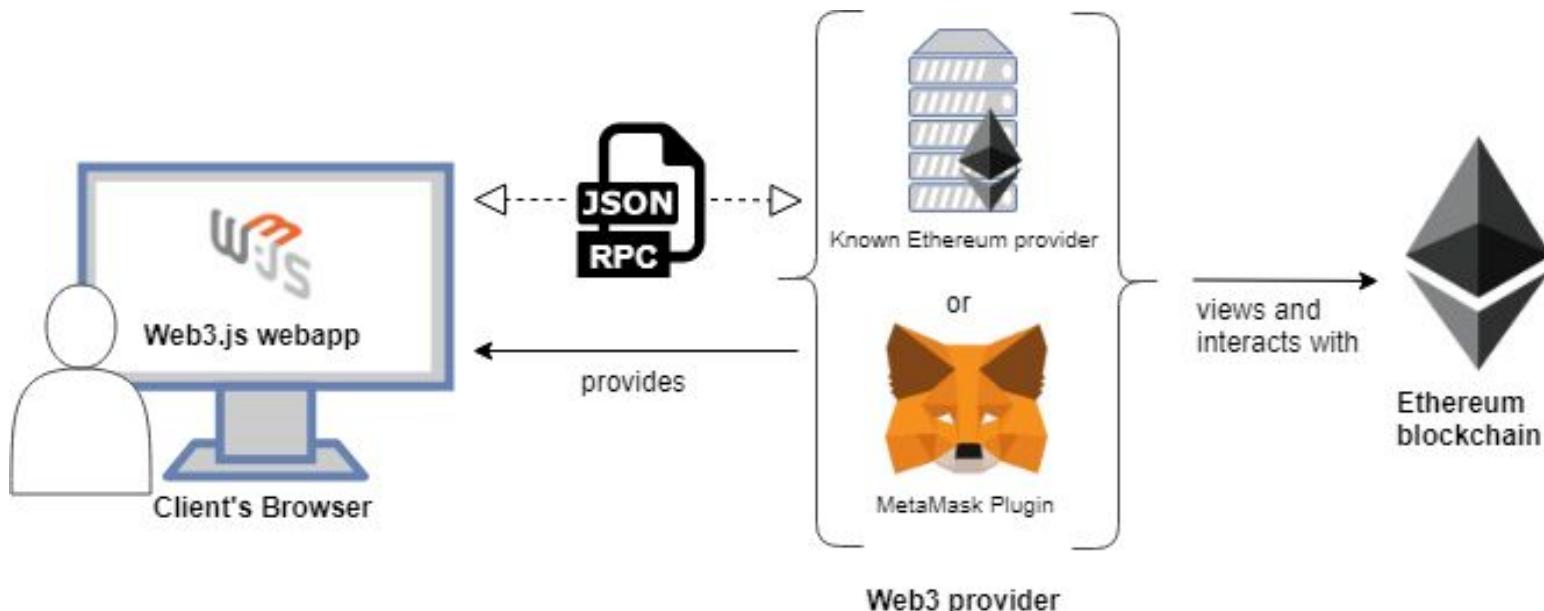
# Ethereum Architecture



# Block Architecture in Ethereum

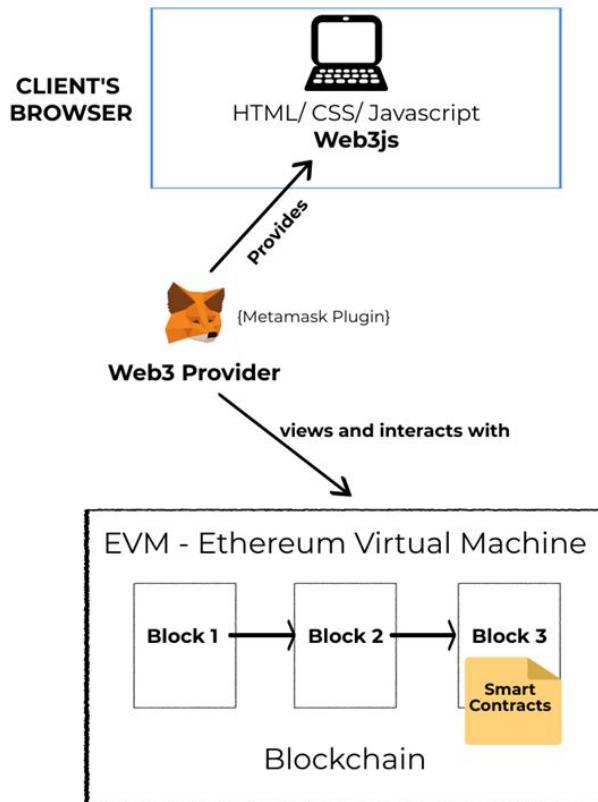


# Ethereum Web3 Architecture

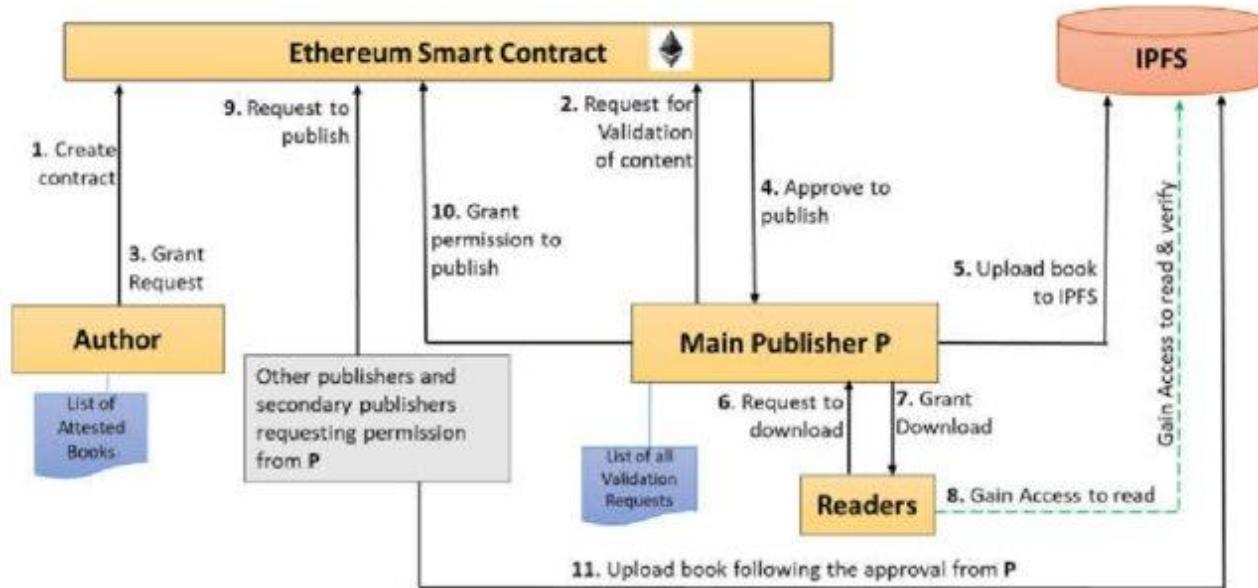


Source: Github

# Client interaction with Ethereum



# Ethereum example for book authenticity validation:



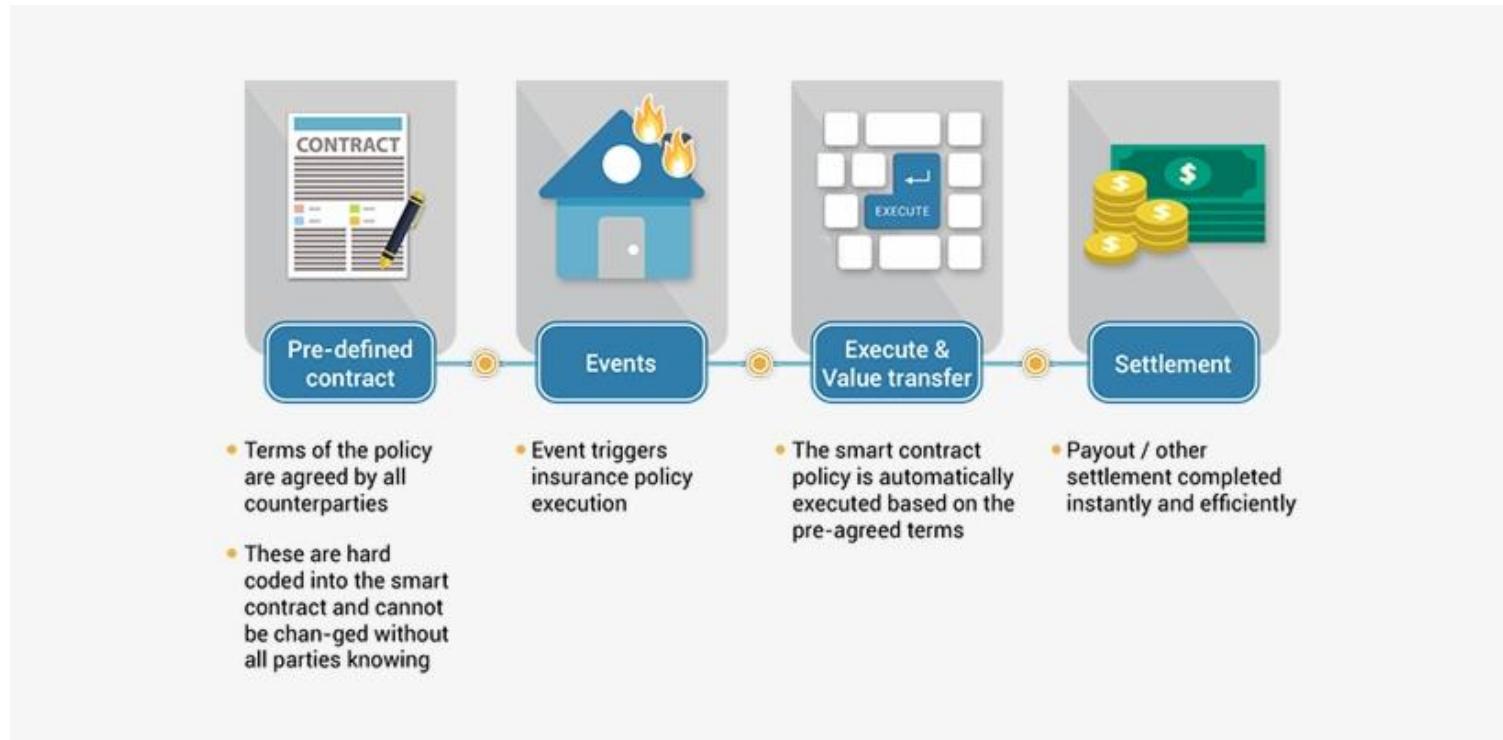
Source: Khaled Salah et al.



# Smart Contract

- “Smart contracts define the rules and penalties around an agreement in the same way that a traditional contract does, but also automatically enforce those obligations.”
  - For example, if the car payment is not made on-time, the car gets digitally locked until the payment is received.
- “A smart contract is a piece code and data that is deployed to a blockchain (e.g., Ethereum)”
- “A smart contract can perform calculations, store information, and automatically send funds to other accounts”.
- A smart contract is immutable and unalterable which can be trusted by participants to execute operations ranging from simple cryptocurrency transfer to more complicated operations.

# Smart Contract



# The characteristics of smart-contracts:

- **Deterministic:** All participants of the network should produce the same output by giving them the same input.
- **Immutable:** Once it is deployed in the Blockchain network (Ethereum), it can neither be removed nor altered.
- **Verifiable:** A smart-contracts have a unique address.
  - Smart-contracts operate in a highly constrained and minimalistic execution environment (the EVM).
  - The most popular programming language for smart-contracts is Solidity.

**MOST IMPORTANT:** smart-contract needs to be triggered, it can't decide: oh it's the first day of the month I need to do this.

- Challenges?
  - Bugs!

# Smart-contract example

```
985 contract ERC20 is Context, IERC20 {
986     mapping (address => uint256) private _balances;
987
988     mapping (address => mapping (address => uint256)) private _allowances;
989
990     uint256 private _totalSupply;
991
992     string private _name;
993     string private _symbol;
994     constructor (string memory name_, string memory symbol_) {
995         _name = name_;
996         _symbol = symbol_;
997     }
998
999     function name() public view virtual returns (string memory) {
1000         return _name;
1001     }
1002     function symbol() public view virtual returns (string memory) {
1003         return _symbol;
1004     }
1005
1006     function decimals() public view virtual returns (uint8) {
1007         return 18;
1008     }
1009
1010     function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
1011         _transfer(_msgSender(), recipient, amount);
1012         return true;
1013     }
1014
1015     function allowance(address owner, address spender) public view virtual override returns (uint256) {
1016         return _allowances[owner][spender];
1017     }
1018 }
```

# Example of Blockchain Marketplace for NFTs

← → ⌂ opensea.io/assets?search[sortAscending]=false&search[sortBy]=FAVORITE\_COUNT

Dashboard Docker usage in B... Start your own Hy... https://www.awse... devops Docker Container... Important Notes -... Tutorial Hyperledg... Other Bookmarks Reading List

**OpenSea** Search items, collections, and accounts Marketplace Stats Resources Create

Filter Status Price Collections Chains Categories

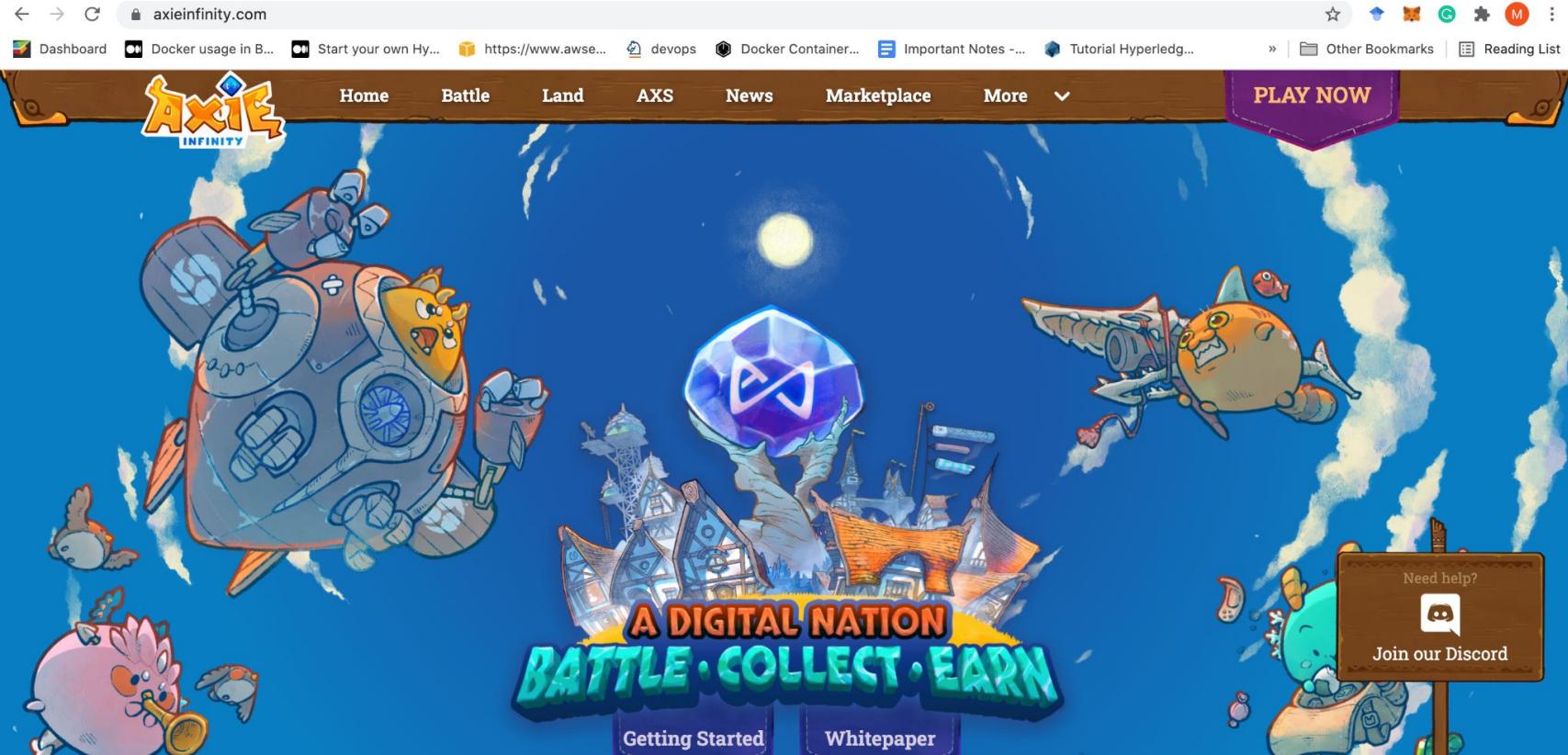
58,542,456 results

All items Most Favored

Image	Name	Price	Last	Offers	Views
	iceth Ethereum gold chain	0.19	0.049	4 days left	39.1K
	Crypto Memes Off... Paper Hands	0.0006	0.049	0.25	36.2K
	BIGGI NFT (TIER-5) BIGGI #45	0.049	0.049	0	32.2K
	CryptoHAM NFT (T... HAM #17	0.06	0.06	0	29.2K
	BIGGI NFT (TIER-5) BIGGI #47	0.059	0.059	0	25.1K

<https://opensea.io/assets/0x495f947776749ce646f68ac8c248420045cb7h5e/139171055655226709170195435774612541103864380699514397445196724883541251102>

# Example of videogames on Blockchain



# Example of NFT marketplace for NBA teams

nbatopshot.com/transactions/top-sales

Dashboard Docker usage in B... Start your own Hy... https://www.awse... devops Docker Container... Important Notes -... Tutorial Hyperledg... Other Bookmarks Reading List

**TOP SHOT** BETA PACKS MARKETPLACE COMMUNITY COLLECTION CHALLENGES HELP

## MARKETPLACE

FOR SALE LATEST SALES TOP SALES

MOMENT	PRICE	SERIAL	SET	SERIES	BUYER	SELLER	DATE / TIME	TX
 LEBRON JAMES	<b>\$230,023.00</b>	Legendary #23/79 (LE)	2020 NBA Finals	1	@easyaces	@GrindBuySell	Aug 25, 21 11:17 PM	
 LEBRON JAMES	<b>\$210,000.00</b>	Legendary #12/59 (LE)	From the Top	1	@bigdog_broth...	@easyaces	Mar 20, 21 11:39 AM	
 LEBRON JAMES	<b>\$208,000.00</b>	Legendary #29/49 (LE)	Cosmic	1	@jesse	@Sparky_24	Feb 22, 21 3:36 PM	
 LEBRON JAMES	<b>\$179,000.00</b>	Legendary #17/59 (LE)	From the Top	1	@Spicy_Spicy...	@BUCKNASTY	Mar 16, 21 5:20 PM	
 FRED VANVLEET	<b>\$140,190.00</b>	Common #11511/15000 (LE)	Base Set	2	@popo	@GPK_JunkY	Apr 14, 21 6:25 PM	
 LEBRON JAMES	<b>\$125,000.00</b>	Legendary #12/59 (LE)	From the Top	1	@easyaces	@www	Feb 24, 21 10:55 PM	

# Salary for Blockchain Engineers:

## COMPARE BLOCKCHAIN ENGINEER SALARIES BY REGION

Blockchain Engineers are highest in demand in [SF Bay Area](#), [New York](#), and [London](#). Browse and compare average salaries in locations where this role is also popular:

1. <a href="#">SF Bay Area</a>	\$162,288	8. <a href="#">Austin</a>	\$135,028
2. <a href="#">Seattle</a>	\$153,181	9. <a href="#">Denver</a>	\$133,465
3. <a href="#">New York</a>	\$153,113	10. <a href="#">Washington D.C.</a>	\$131,073
4. <a href="#">Dallas/Ft Worth</a>	\$145,750	11. <a href="#">San Diego</a>	\$123,333
5. <a href="#">Boston</a>	\$144,985	12. <a href="#">Toronto</a>	C\$118,281
6. <a href="#">Los Angeles</a>	\$142,427	13. <a href="#">London</a>	£72,946
7. <a href="#">Chicago</a>	\$141,880	14. <a href="#">France</a>	€55,951

# Examples:

- There are some potential applications that can be on-chain only (no outside data at all): like chess game, gambling (you just put the rules in the smart-contract and then it's based off of ETH coins, so never had outside data needed)

# Electric Vehicle Charging

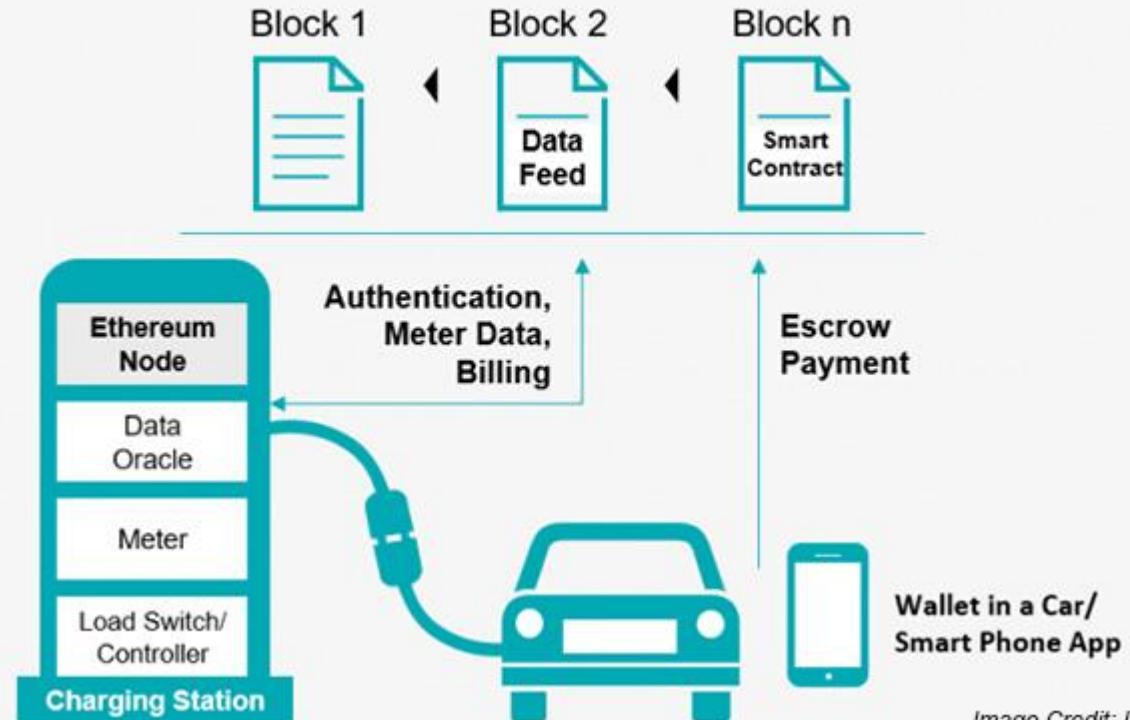


Image Credit: IBTimes Ltd

# Smart Contracts Use Cases



Record Storing



Trading Activities



Supply Chains



Mortgage



Real Estate  
Market



Employment  
Arrangements



Copyright  
Protection



Healthcare  
Services



Government  
Voting



Insurance  
Claims



Internet-of-  
Things (IoT)



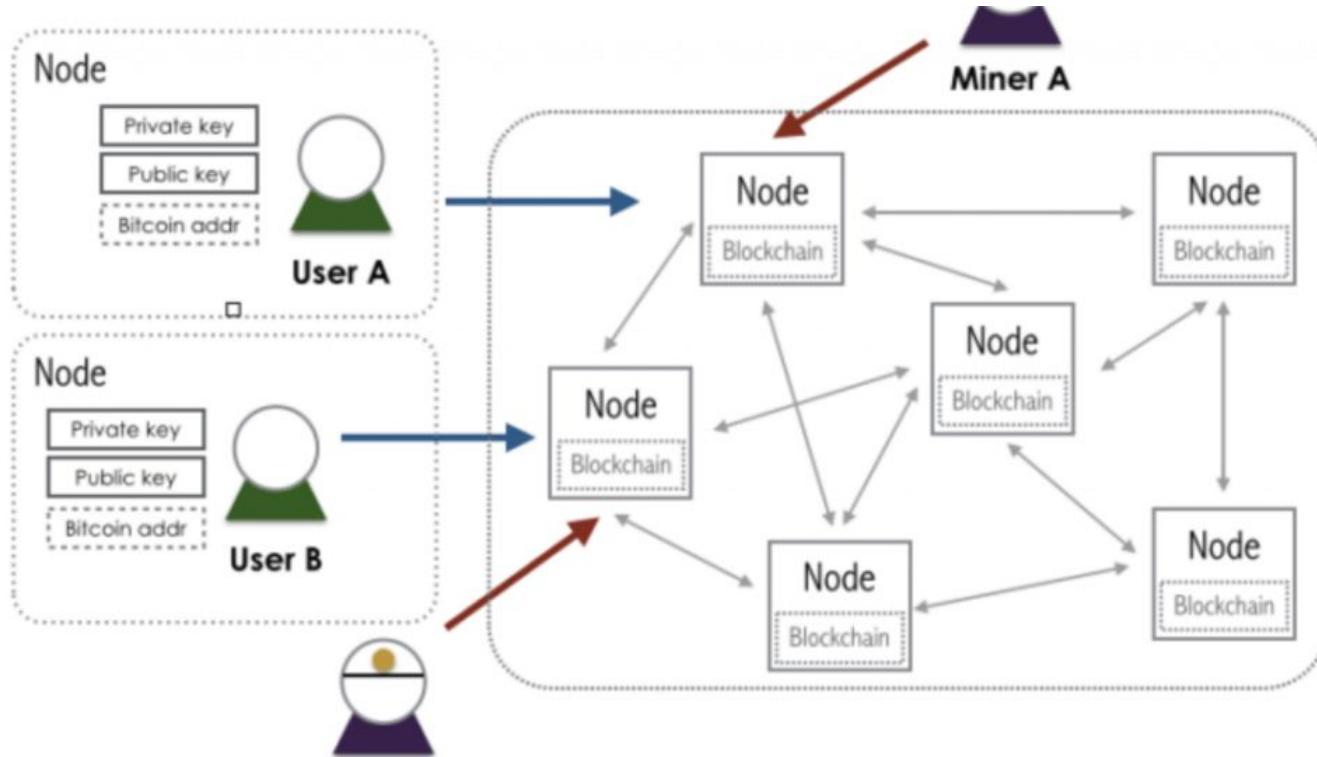
# Private Blockchain

**A private blockchain is a permissioned blockchain.** Private blockchains work based on access controls which restrict the people who can participate in the network. There are one or more entities which control the network and this leads to reliance on third-parties to transact. In a private blockchain, only the entities participating in a transaction will have knowledge about it, whereas the others will not be able to access it. **Hyperledger Fabric of Linux Foundation is a perfect example of a private blockchain.**

## Similarities Of Public And Private Blockchains

- **Both function as an append-only ledger** where the records can be added but cannot be altered or deleted. Hence, these are called immutable records.
- **Each network node in both these blockchains has a complete replica of the ledger.** Both are decentralized and distributed over a peer-to-peer network of computers.
- **In both, the validity of a record is verified**, thus providing a considerable level of immutability, until the majority of the participants agree that it is a valid record and reach consensus. This helps prevent tampering with the records.
- **Both blockchains rely on numerous users to authenticate edits** to the distributed ledger thus helping in the creation of a new master copy which can be accessed by everyone at all times.

# Private Blockchain

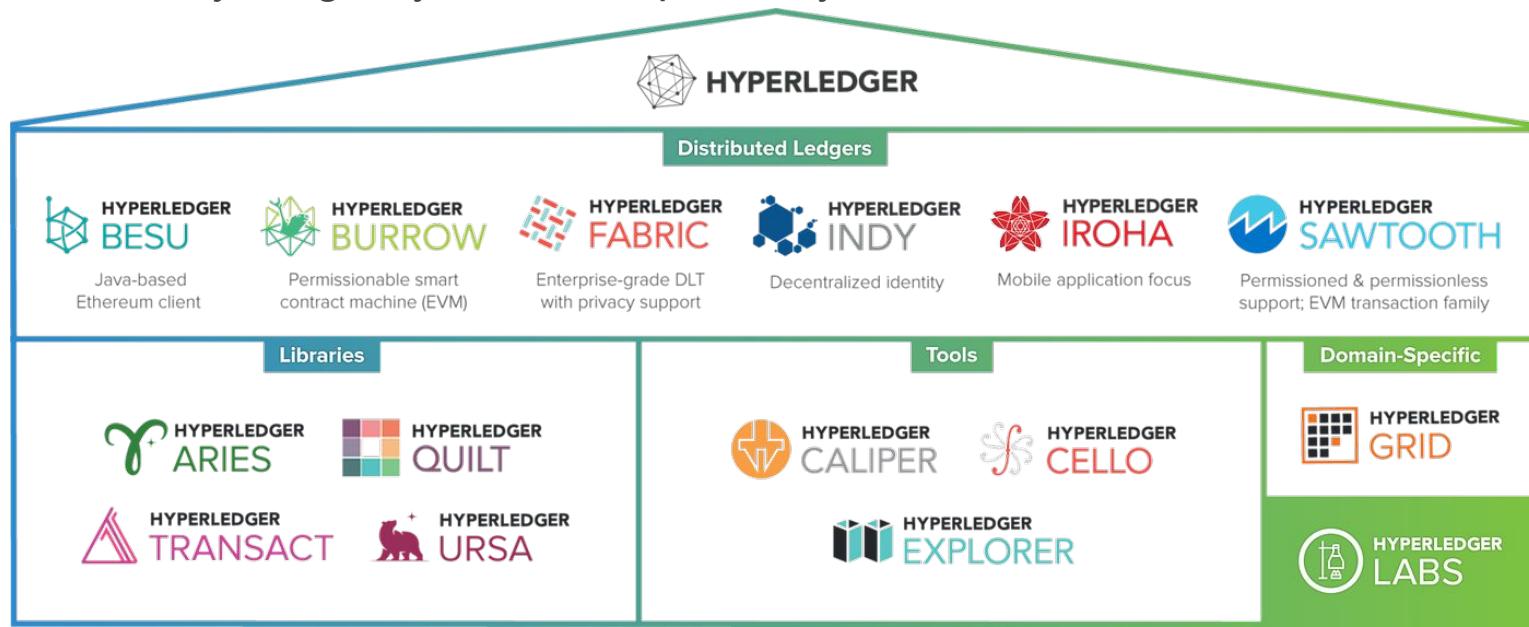


# Public Vs. Private

	<b>Public Blockchain</b>	<b>Private Blockchain</b>
Access	Open read/write. Allows anyone to participate, execute contracts, run node, become a miner	Permissioned read/writes. All participants and nodes needs to be approved.
Consensus	Proof of Work (PoW), Proof of Stake (PoS), etc	Custom: PBFT (Practical Byzantine Fault Tolerant), multi-signature, etc
Currency	Mostly required. Used for transactions, rewarding miners and other utility (PoS)	Not required
Apps	Commonly known as dApps (decentralized Apps), they are decentralized, guarantee privacy and anonymity. Execution costs currency (to reward miners) and take time for data to verified across all nodes. Equivalent to open Internet.	Apps are built to custom business needs. Private Blockchain tech helps cut down processing times, less data redundancy and introduce efficiency between (or within) systems or organizations, and the same time restrict public access to sensitive data.
Examples	You can build something which serves everyone: from fun games like Crypto Kitties to important services like universal KYC (Civic), decentralized file storage (Storj, Filecoin), Anonymous SSO	Popular examples could be banks sharing a common ledger for fraud detection, international forex transactions, medical institutions storing and sharing patients health data with each other, etc.
Popular platforms	Ethereum, Bitcoin, NEO, Ripple, Stellar	Hyperledger, Corda, Multichain
Languages (Smart Contracts)	Ethereum: Solidity NEO: VB.net, C#, Java, Python	Hyperledger: Chaincode Corda: Kotlin
Frameworks and Development Tools	Ethereum: Truffle, Metamask, Mist Wallet	Hyperledger: Hyperledger-Compose, Visual Studio Corda: Flow

# Hyperledger

- Hyperledger is an ecosystem and an umbrella project consisting of different open source Blockchain platforms that are hosted by the Linux Foundation .
- It is a collaborative project to host different applications such as financing, banking Internet of Things, supply chains, and others, and ensure transparency, immutability, longevity, and interoperability .

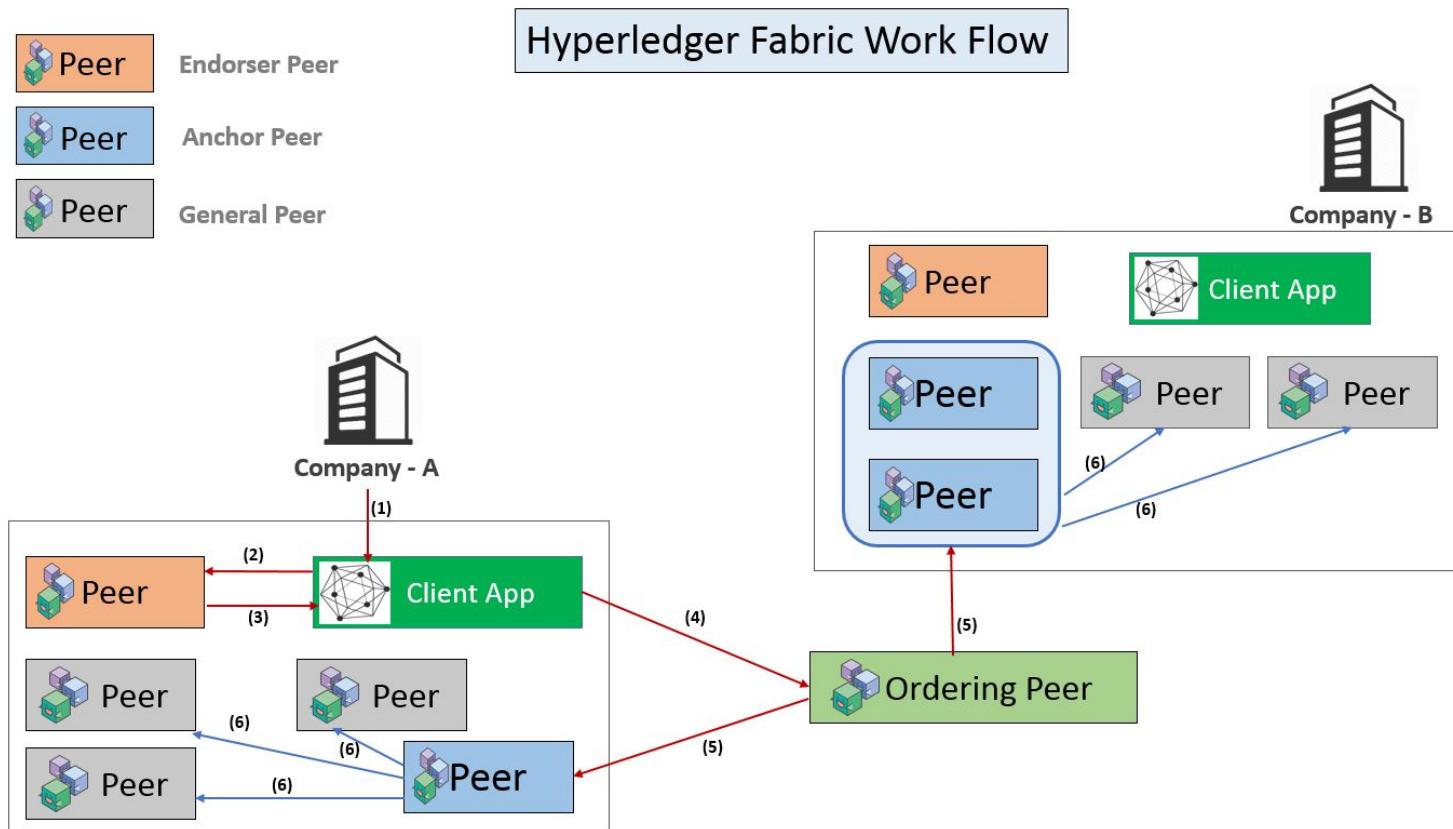


# Hyperledger Fabric

Hyperledger Fabric is an open-source Blockchain platform which was proposed by IBM. The purpose of this platform is to overcome some limitations of other Blockchain platforms such as Ethereum and Tendermint. In particular, order-executive or hard-coded consensus are two shortcomings that affect the throughput and latency in other Blockchain platforms. The consensus algorithm in Hyperledger Fabric relies on Crash Fault Tolerant (CFT) which is based on Raft protocol.

- The Objective of Hyperledger is to create a platform which can be adapted to use cases in business
- Intended clients: Companies, Governments and their customers.

# Hyperledger Fabric Architecture



# Complementary technologies in Fabric:

## CouchDB

- records the state of chains
- uses Javascript for request processing



## Docker Technology

- Chaincode Execution Environment
- Provides language-agnostic environment without bytecode
- The chaincodes have interaction through the Fabric SDK



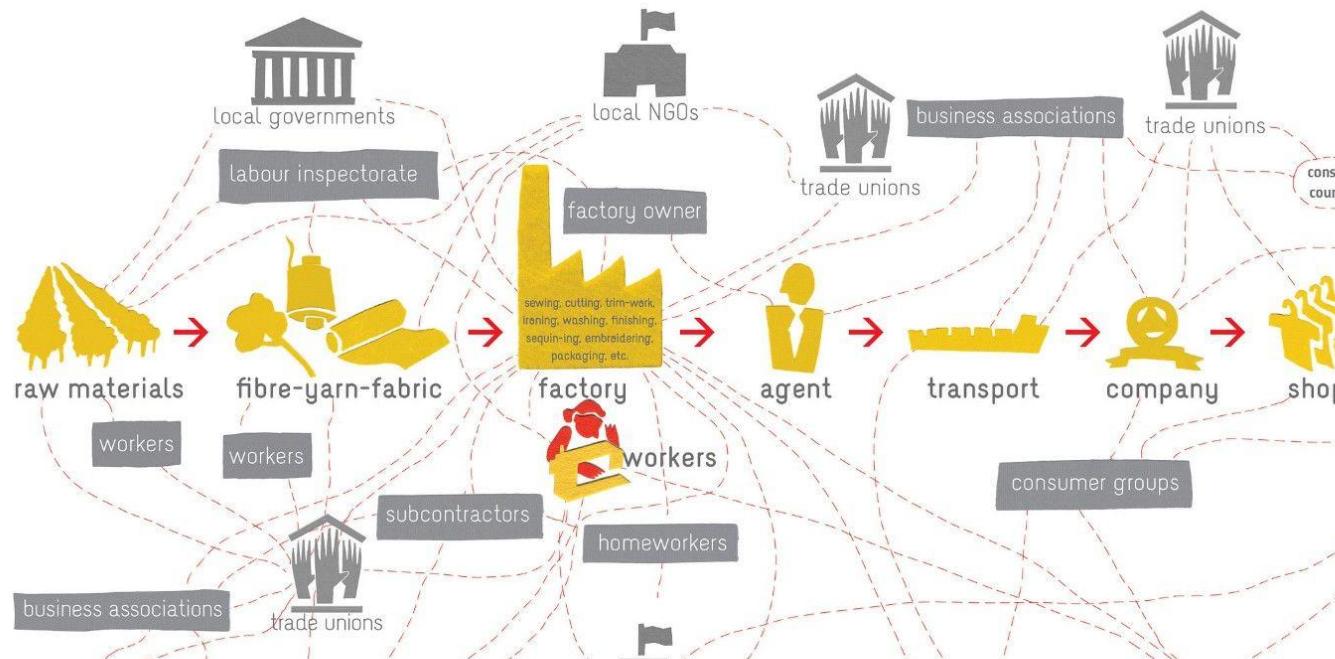
# Comparison

Ser	Features	Bitcoin	Ethereum	Hyperledger-Fabric V 0.1
1.	Fully developed	✓	✓	✓
2.	Miner participation	Public	Public, Private, Hybrid	Private
3.	Trustless operation	✓	✓	Trusted validator nodes
4.	Multiple applications	Financial only	✓	✓
5.	Consensus	PoW	PoW, PoS ("Casper")	PBFT (Now changed to RAFT in v1.4)
6.	Consensus finality	X	X	✓
7.	Blockchain forks	✓	✓	X
8.	Fee less	X	X	Optional
9.	Run smart contracts	X	✓	✓
10.	TX integrity and authentication	✓	✓	✓
11.	Data Confidentiality	X	X	✓
12.	ID management	X	X	✓
13.	Key management	X	X	✓ (through CA)
14.	User authentication	Digital Signatures	Digital Signatures	Based on enrolment certificates
15.	Device authentication	X	X	X
16.	Vulnerability to attacks	51%, linking attacks	51%	>1/3 faulty nodes
17.	TX throughput	7 TPS	8-9 TPS	>3500 TPS (depending upon number of endorsers, orderers and committers)
18.	Latency in single confirmation of a TX	10 min (60 min for a confirmed TX)	15–20 s	Less than Bitcoin, Ethereum & IOTA
19.	Is it Scalable?	X	X	X

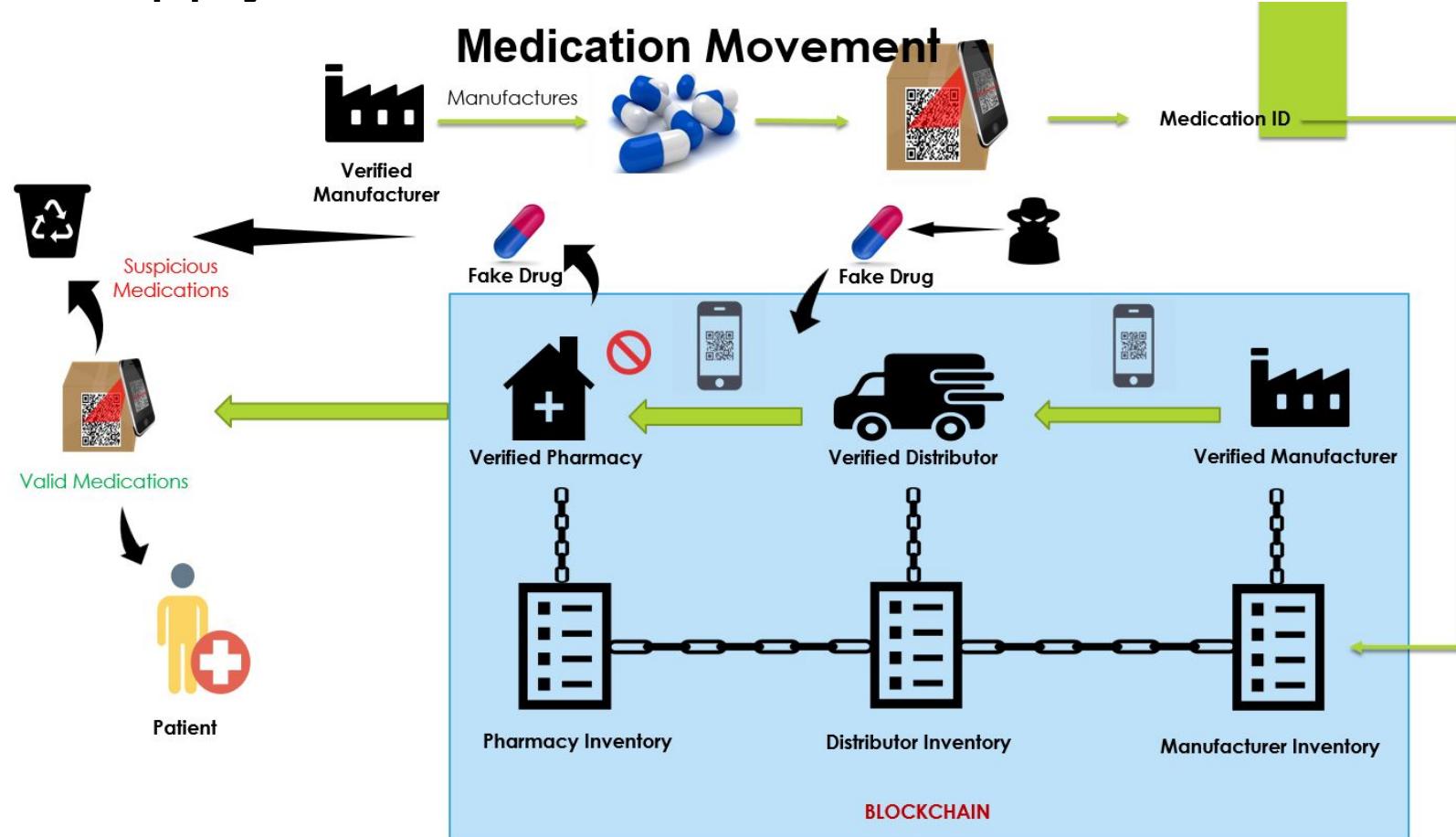
# Fabric Use Case: Collaboration of Walmart and IBM



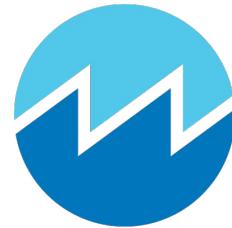
# Fabric Supply-chain



# Fabric Supply-chain



# Hyperledger Sawtooth

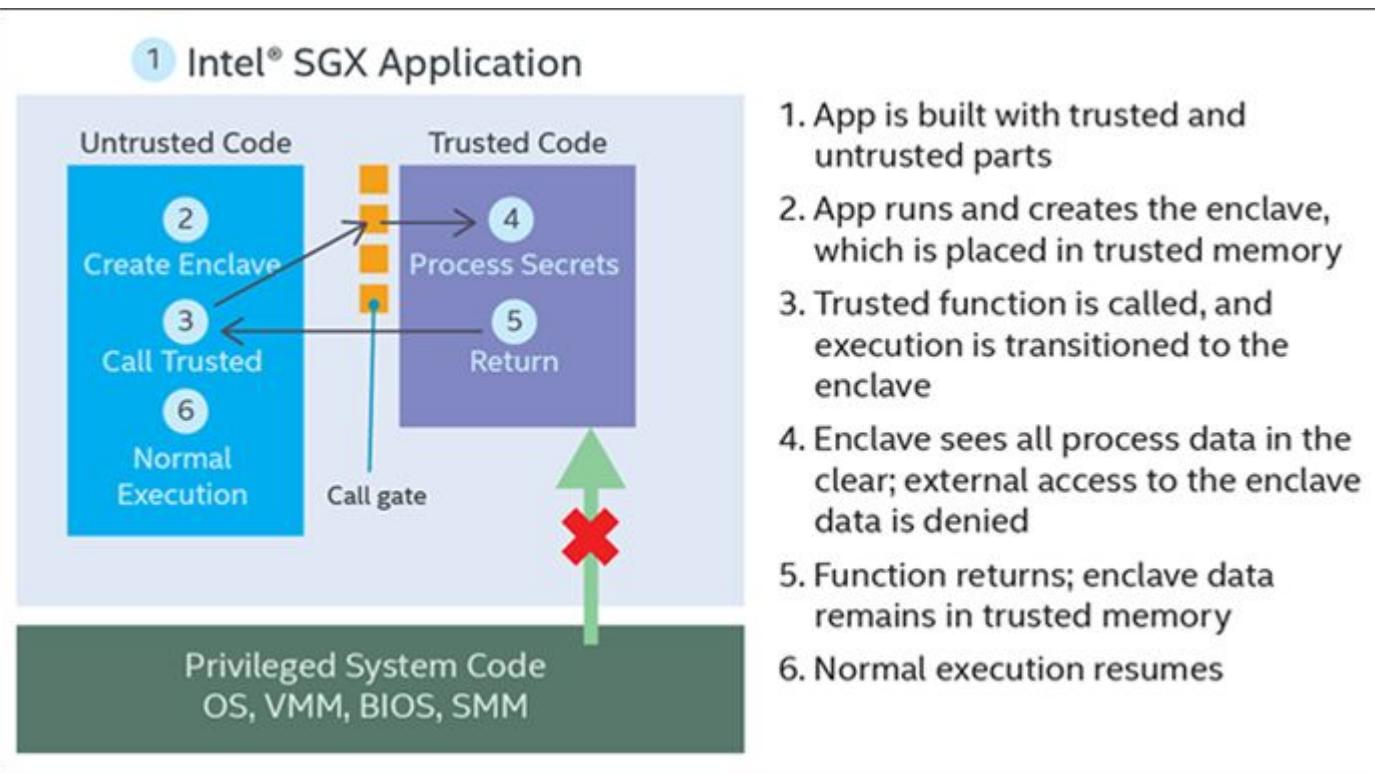


**HYPERLEDGER  
SAWTOOTH**

- Hyperledger Sawtooth is an open-source Blockchain platform which is designed especially for supplying chain management.
- It is part of the Linux Foundation umbrella project developed by Intel.
- The main difference of Sawtooth with other Blockchain platforms is that the transactions and data can be executed in parallel instead of in series, which will result in better performance of the system.
- It supports dynamic consensus protocols including RAFT, PoET and PBFT
- Each validator in Sawtooth chooses a random time to respond, the first to wake up "wins" and sends the block

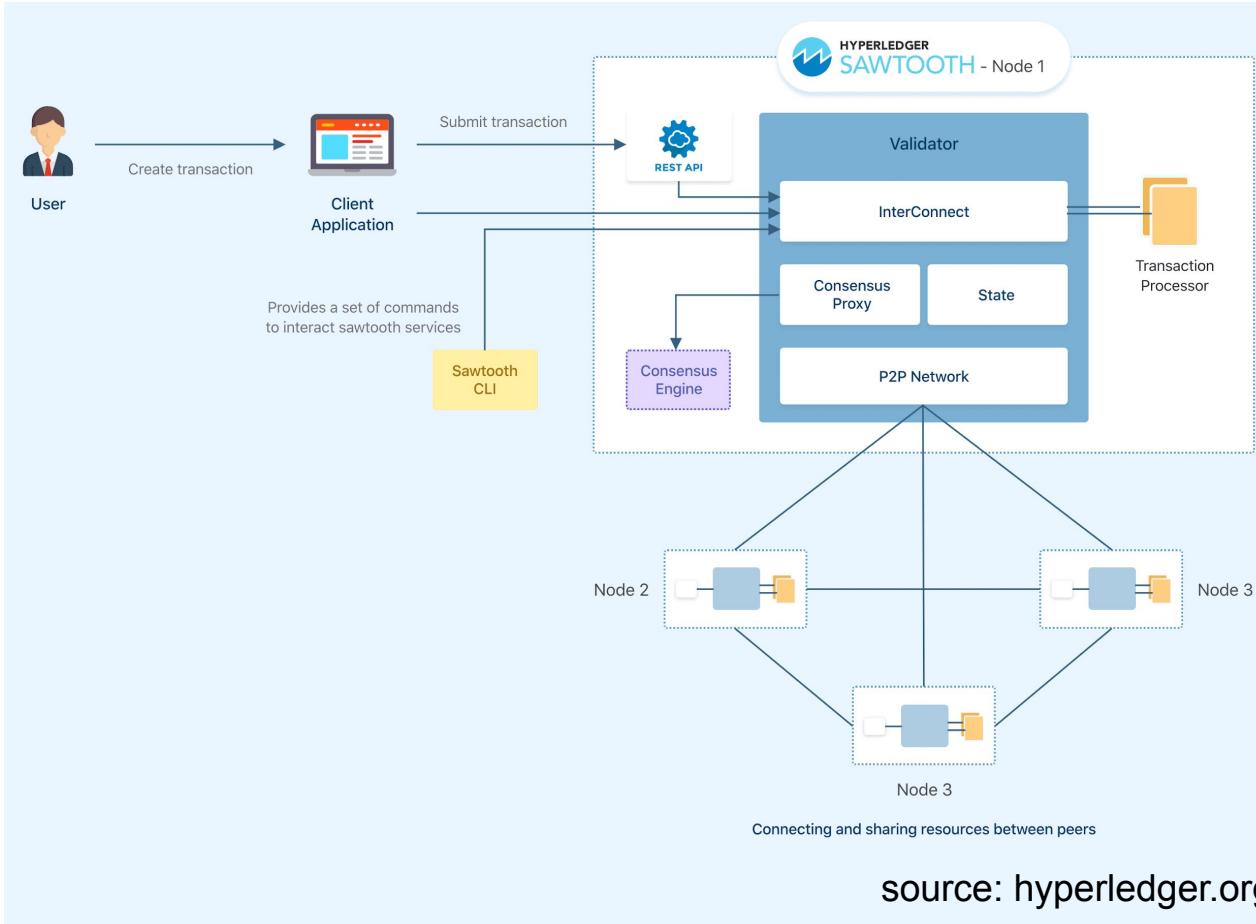


# Intel SGX Architecture



source: Intel

# Sawtooth Architecture



# References:

- 1) Antonopoulos, A. M. (2014). *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.".
- 2) Chuen, D. L. K. (Ed.). (2015). *Handbook of digital currency: Bitcoin, innovation, financial instruments, and big data*. Academic Press.
- 3) Antonopoulos, A. M., & Wood, G. (2018). *Mastering ethereum: building smart contracts and dapps*. O'reilly Media.
- 4) Gaur, N., Desrosiers, L., Ramakrishna, V., Novotny, P., Baset, S. A., & O'Dowd, A. (2018). *Hands-on blockchain with hyperledger: building decentralized applications with hyperledger fabric and composer*. Packt Publishing Ltd.
- 5) Badr, B., Horrocks, R., & Wu, X. B. (2018). *Blockchain By Example: A developer's guide to creating decentralized applications using Bitcoin, Ethereum, and Hyperledger*. Packt Publishing Ltd.
- 6) Bashir, I. (2017). *Mastering blockchain*. Packt Publishing Ltd.
- 7) Baset, S. A., Desrosiers, L., Gaur, N., Novotny, P., O'Dowd, A., Ramakrishna, V., ... & Wu, X. B. (2019). *Blockchain Development with hyperledger: build decentralized applications with hyperledger fabric and composer*. Packt Publishing Ltd.\
- 8) Hafid, A. (2020). *Blockchain and its applications* [Lecture notes]. Retrieved from <https://studium.umontreal.ca/course/view.php?id=169234>
- 9) Makhdoom, I., Abolhasan, M., Abbas, H., & Ni, W. (2019). Blockchain's adoption in IoT: The challenges, and a way forward. *Journal of Network and Computer Applications*, 125, 251-279.