

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are positioned diagonally, with the blue one partially covering the green one.

# Introduction au Data Engineering

Un cours de Yann Fornier

# Présentation

Yann Fornier

Ingénieur en Aérospatial - Data Manager - Support à la digitalisation dans la stratégie d'entreprise.



# Présentation

Yann Fournier

Enseignant en Informatique - Ecole d'Ingénieur, Ecole de Commerce, Université

Domaines privilégiés : Informatique Quantique, Cloud Computing, Blockchain





# Objectifs du cours

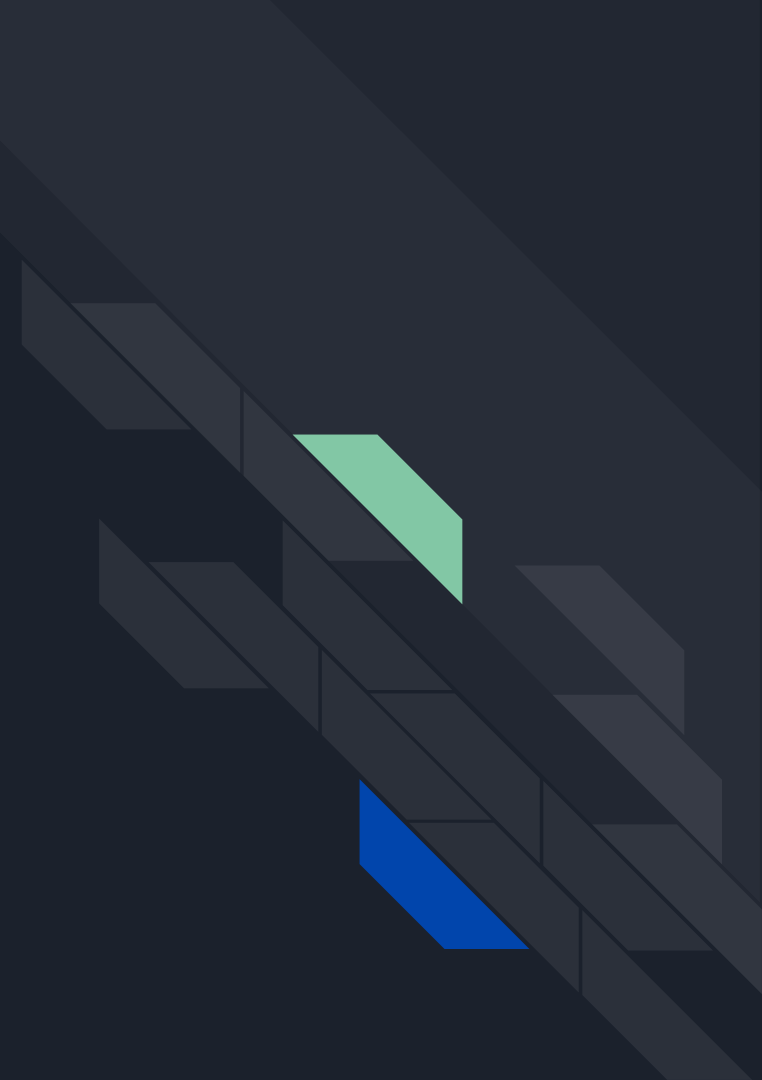
Comprendre les concepts fondamentaux du Data Engineering.

Acquérir une vue d'ensemble des outils, technologies et bonnes pratiques utilisés en Data Engineering.

Apprendre à concevoir et implémenter un pipeline de données simple.

Préparer les étudiants à des études plus avancées dans le domaine du Data Engineering.

# Module 1: Introduction au Data Engineering





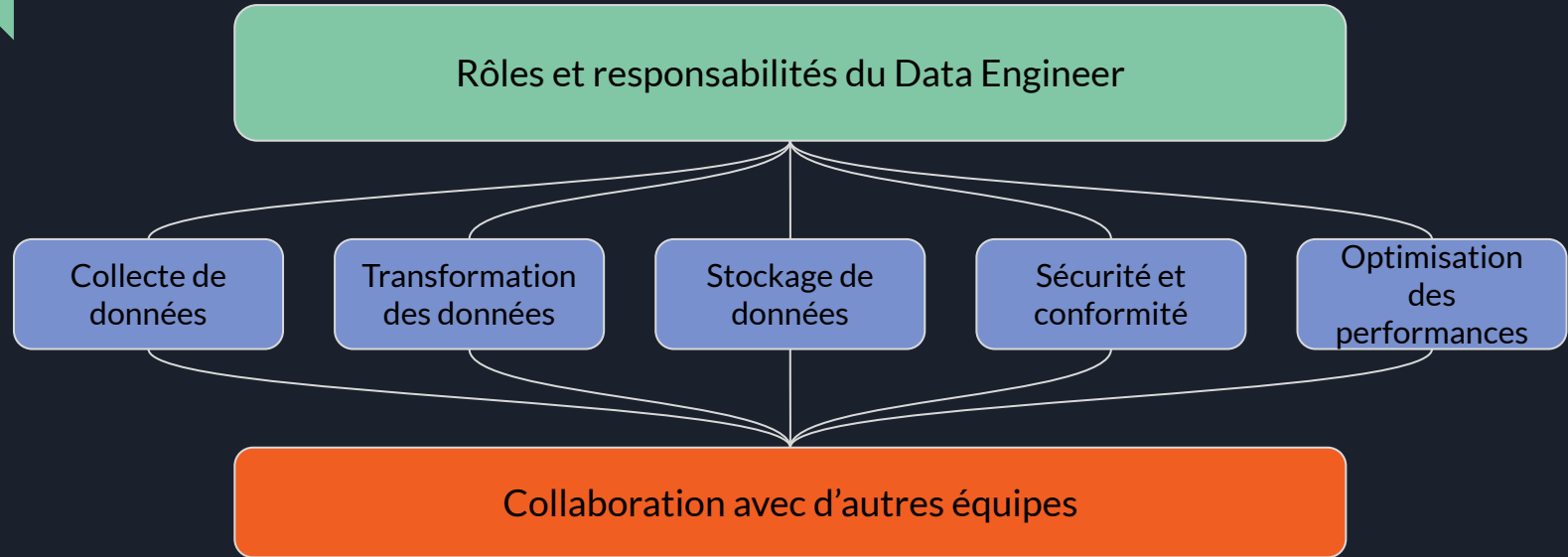
# Définition et importance du Data Engineering

Qu'est ce que le "Data Engineering" ?

Le **Data Engineering**, ou **ingénierie des données** en français, est une discipline de l'informatique et de la science des données qui se concentre sur la conception, la construction et la gestion de systèmes qui collectent, stockent, traitent et organisent de grandes quantités de données.

L'objectif principal du data engineering est de créer une infrastructure de données robuste, fiable et évolutive qui permet aux entreprises et aux organisations d'exploiter efficacement leurs données pour la prise de décision, l'analyse et d'autres applications.

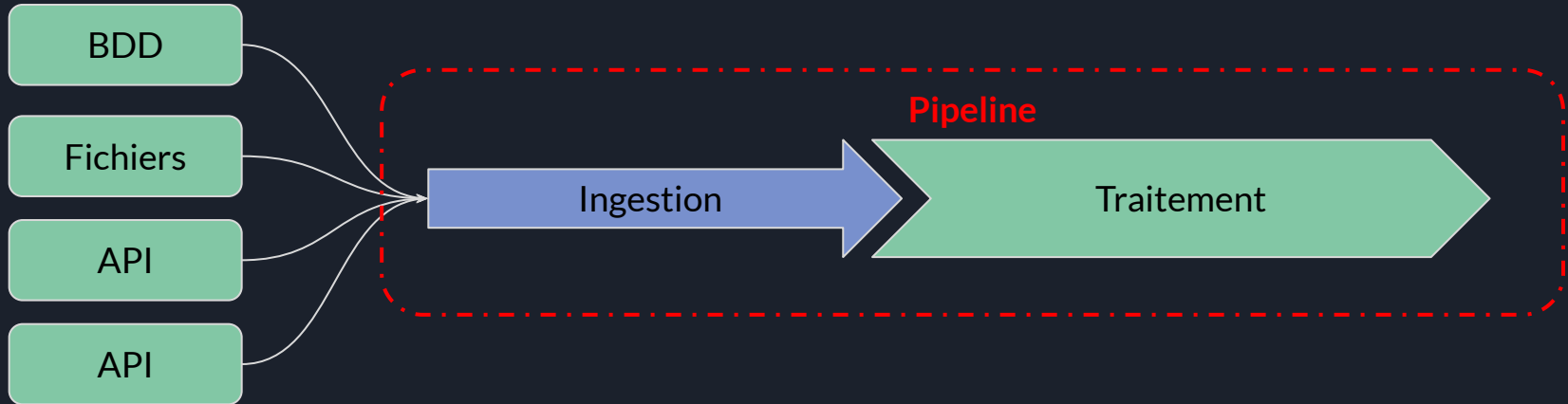
# Définition et importance du Data Engineering



# Rôles et responsabilités du Data Engineer

Collecte de  
données

Les data engineers développent des **pipelines de données** pour collecter des données provenant de différentes sources, telles que des **bases de données**, des **fichiers**, des **API**, des **flux en temps réel**, etc. Cela peut impliquer l'intégration de données provenant de systèmes disparates.

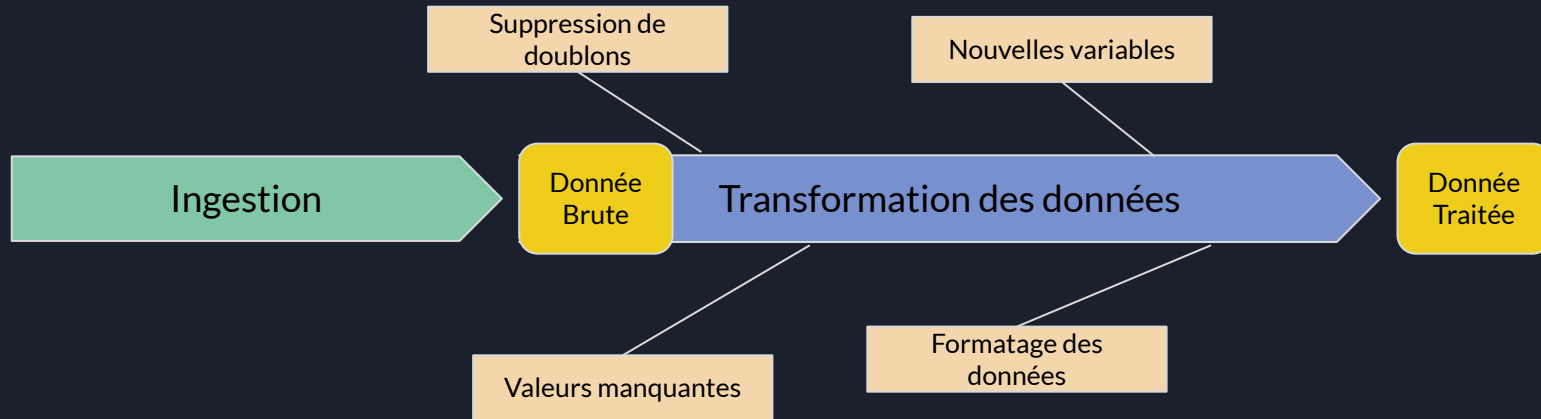




# Rôles et responsabilités du Data Engineer

Transformation  
des données

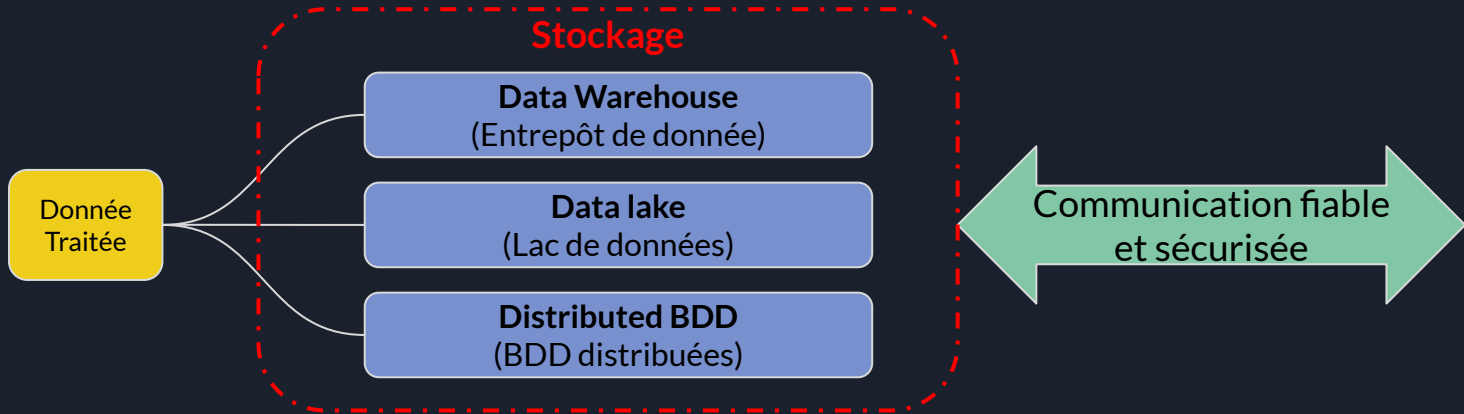
Les données collectées doivent souvent être nettoyées, transformées et agrégées pour être utilisables. Cela peut inclure des tâches comme la suppression des doublons, la gestion des valeurs manquantes, le formatage des données, et la création de nouvelles variables à partir des données brutes.



# Rôles et responsabilités du Data Engineer

Stockage de  
données

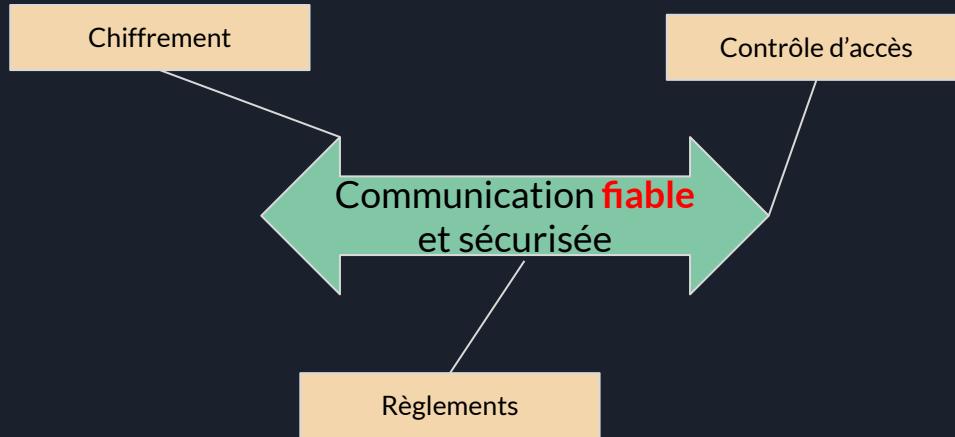
Les data engineers conçoivent et mettent en œuvre des systèmes de stockage de données, tels que des entrepôts de données (data warehouses), des lacs de données (data lakes) et des bases de données distribuées, en fonction des besoins spécifiques de l'organisation. Ils veillent à ce que ces systèmes soient optimisés pour le stockage et la récupération efficaces des données.



# Rôles et responsabilités du Data Engineer

Sécurité et  
conformité

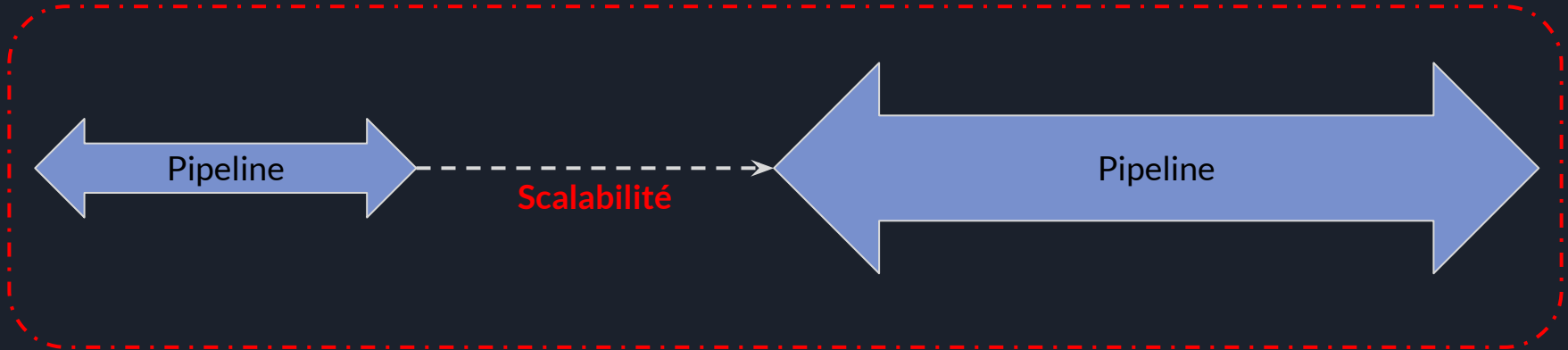
Les data engineers s'assurent que les systèmes de données sont sécurisés, en mettant en place des contrôles d'accès, des mécanismes de chiffrement, et en respectant les réglementations sur la protection des données (comme le RGPD en Europe).



# Rôles et responsabilités du Data Engineer

Optimisation des  
performances

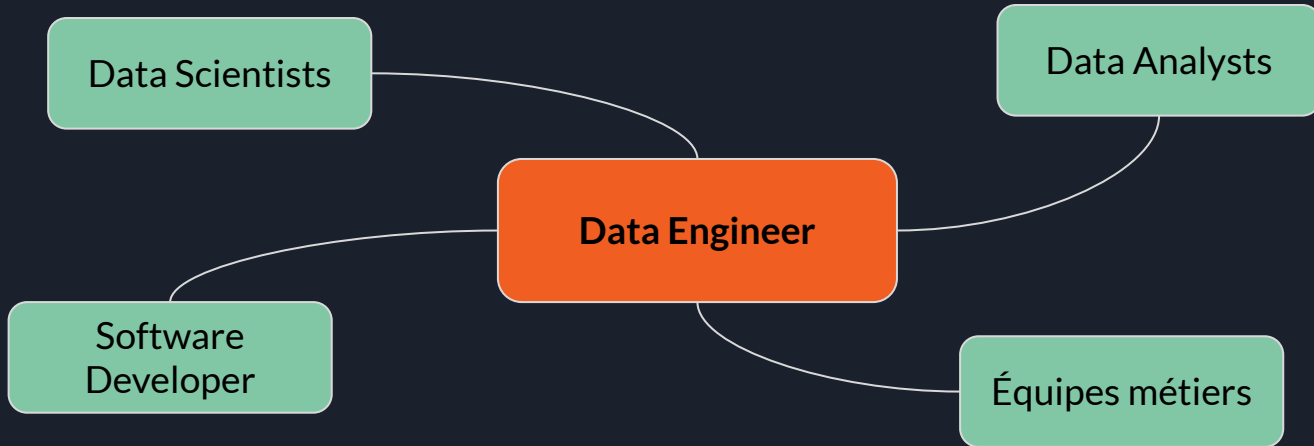
Ils optimisent les systèmes pour gérer des volumes de données croissants et pour s'assurer que les requêtes et les analyses sont effectuées rapidement et efficacement.



# Rôles et responsabilités du Data Engineer

## Collaboration avec d'autres équipes


Les data engineers travaillent souvent en étroite collaboration avec des data scientists, des analystes de données, des développeurs de logiciels et des équipes métiers pour s'assurer que les systèmes de données répondent aux besoins de l'organisation.



# Les métiers du Big Data



# Les métiers du Big Data

DATA/IA	RÉMUNÉRATION ANNUELLE BRUTE EN K€				
	0-2 ans	2-5 ans	5-10 ans	10 ans et +	
Développeur BI	35 - 45	45 - 60	60 - 70	70 - 80	★ ★ ★
Data engineer/Data scientist	38 - 50	50 - 60	60 - 70	70 - 85	★ ★ ★
DataOps engineer	38 - 50	50 - 65	65 - 75	75 - 85+	★ ★ ★
Analyst BI/Data analyst/Data steward	38 - 48	45 - 60	60 - 70	70 - 80	★ ★ ★
Chef de projet BI/Big data	40 - 45	45 - 60	60 - 70	70 - 80	★ ★ ★
Architecte data	80 - 90	90 - 110	110 - 130	130 - 150+	★ ★ ★
DBA	40 - 45	45 - 55	50 - 60	60 - 70	★ ★ ★
Ingénieur IA/Développeur IA	40 - 45	45 - 60	60 - 70	70 - 80+	★ ★ ★
Data protection officer	38 - 45	40 - 60	55 - 75	75 - 85+	★ ★ ★
Machine learning engineer	40 - 50	50 - 60	60 - 75	75 - 90+	★ ★ ★
Data visualisation consultant	38 - 48	50 - 60	60 - 70	70 - 90+	★ ★ ★

"Etude des salaires en 2023 des métiers du Big Data" - Michael Page

<https://www.lepont-learning.com/fr/cartographie-metiers-data-demandes/>



# Rôles et responsabilités du Data Engineer

En résumé, le data engineering est une discipline cruciale pour toute organisation qui souhaite exploiter pleinement le potentiel de ses données. C'est une fondation essentielle pour des activités telles que l'analyse de données, l'apprentissage automatique, et la business intelligence.



A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. They are set against a dark blue background with faint, lighter blue diagonal stripes.

# BIG DATA

## C'est quoi ?

Qu'est ce qu'une "data" ?





“Donnée”



Big Data

“Donnée  
d’entreprise”

Digitalisation

“Donnée”

Traitement

Numérique

# Les processus liés aux données

Catégorisation des données

"Data"

**Catégorisation**

Données  
Industrielles

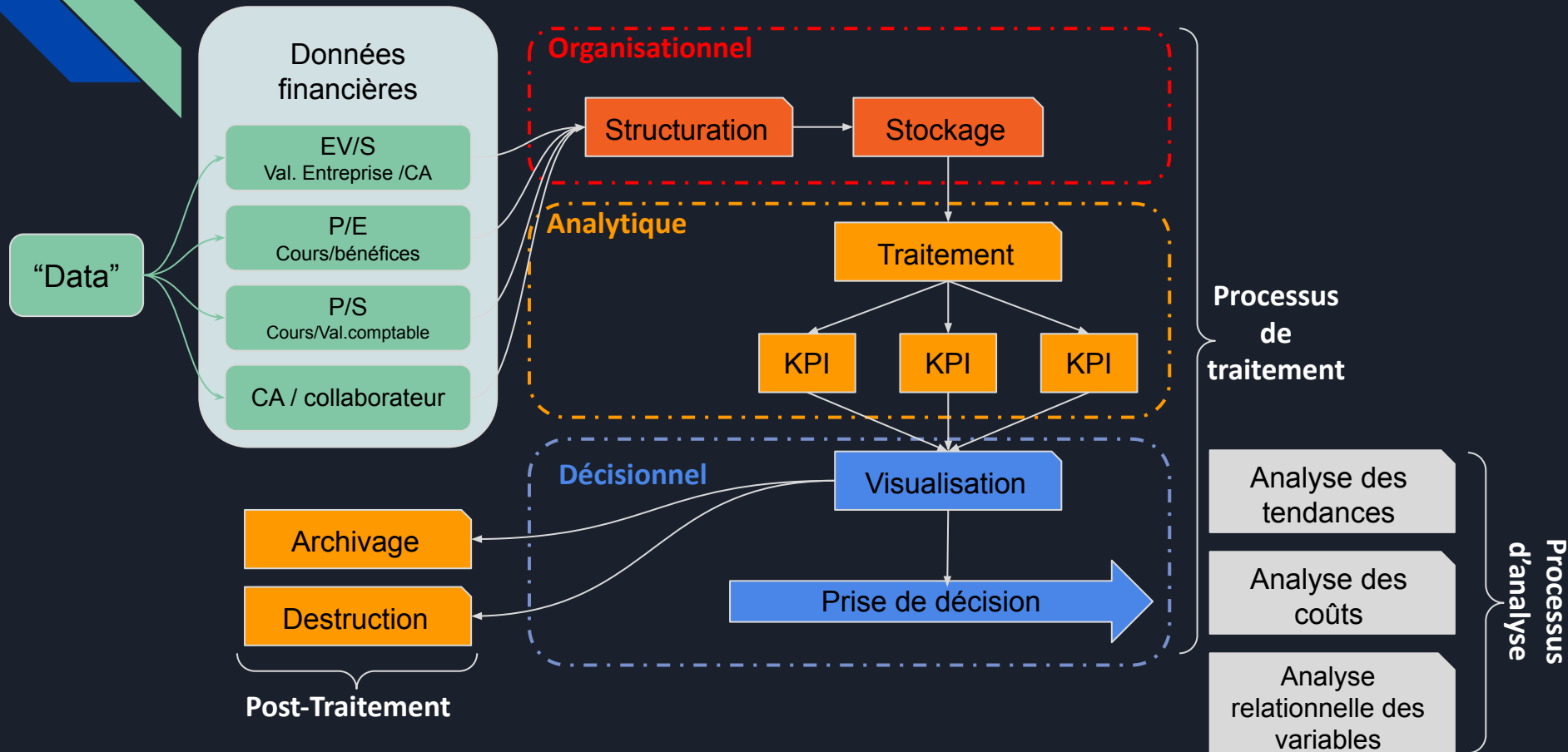
Données RH

Données  
Commerciales

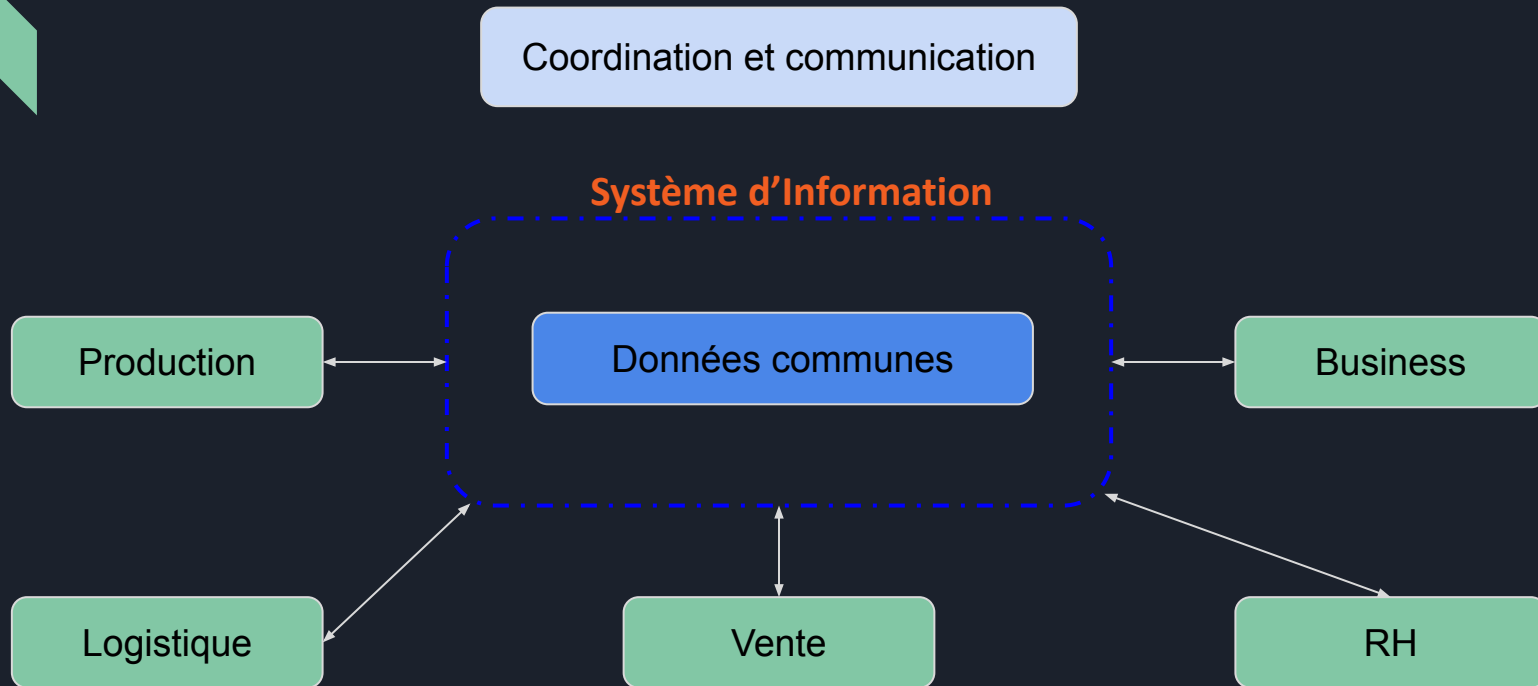
Données  
Financières

Et plus encore...

# Les processus liés aux données



# Les processus liés aux données





# Les types de données

## Données Quantitatives

Ces données peuvent être mesurées et exprimées en chiffre.

*Revenus, chiffres, âges...*

## Données Qualitatives

Ces données décrivent des caractéristiques qui ne peuvent pas être mesurées.

*Genre, statut matrimonial, préférences...*

## Données Catégorielles

Ces données sont organisées en catégories, souvent par couleur ou symbole

*Classement par couleur, symboles...*

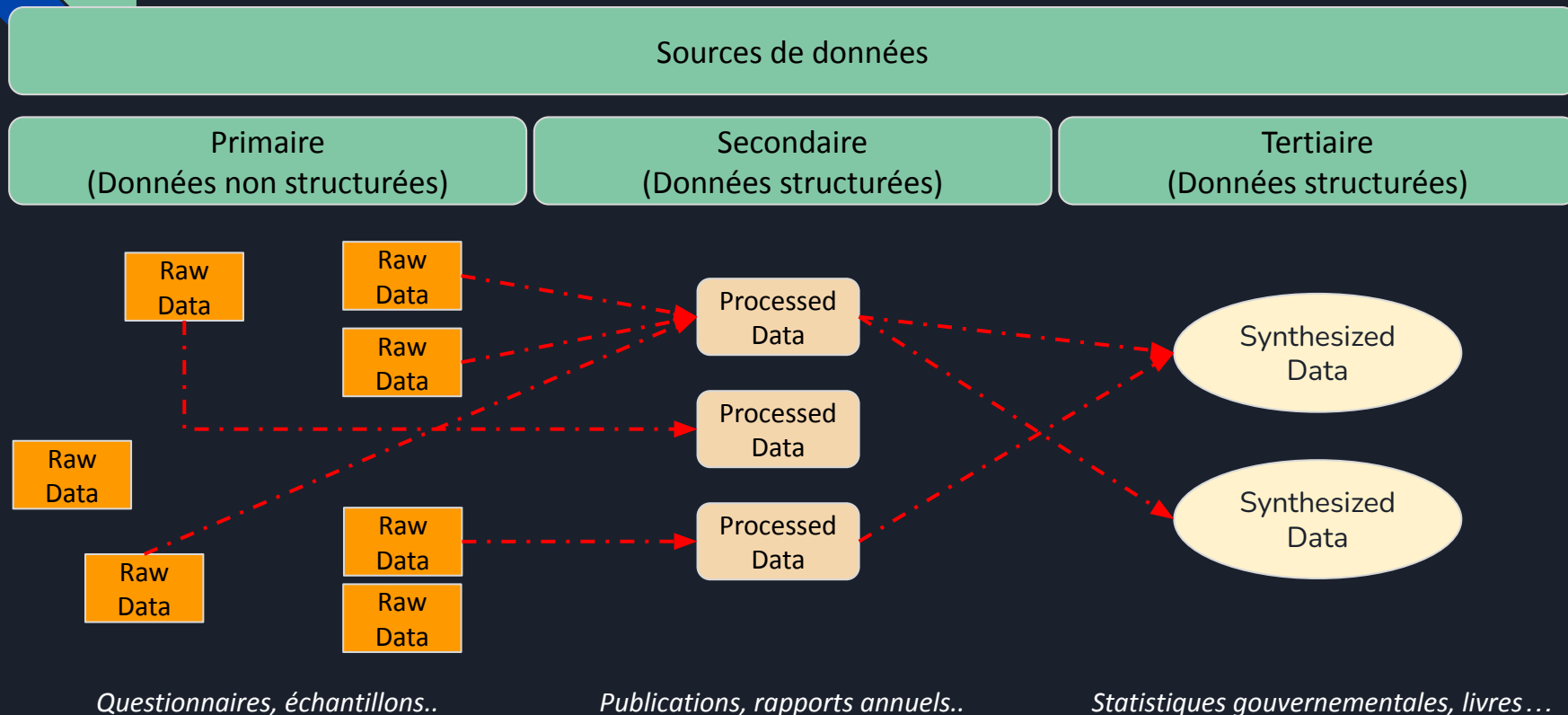
## Données Textuelles

Ces données sont composées de phrases, paragraphes ou textes entiers.

*Commentaires, critiques, articles de journaux...*

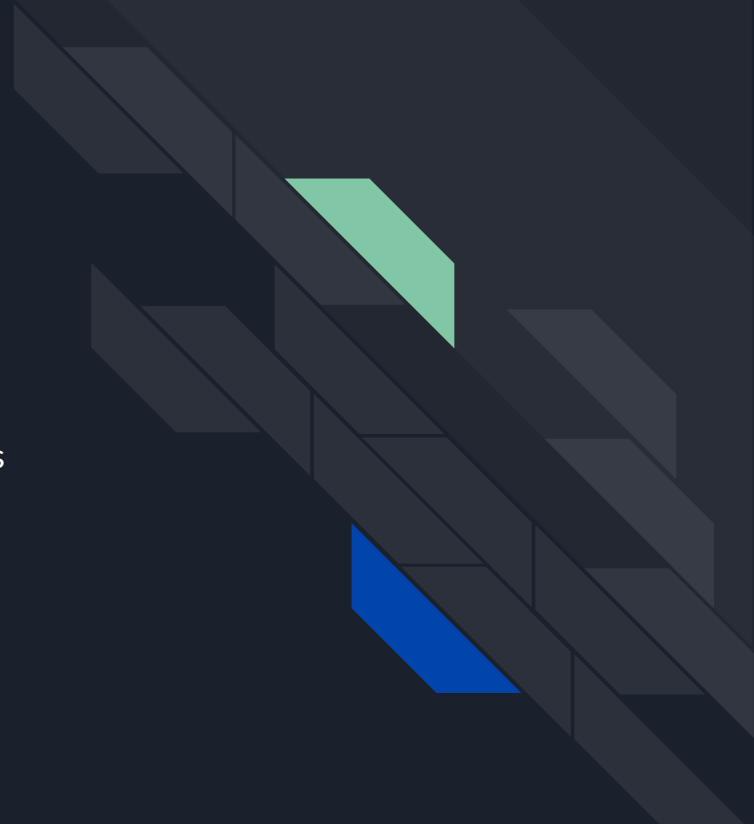


# Les sources de données et leur transformation

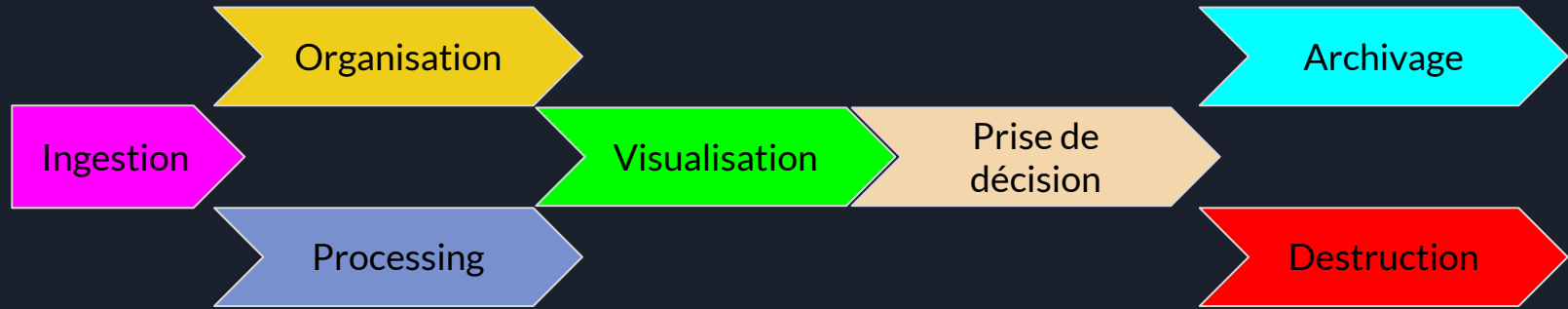


# Cycle de vie des Données

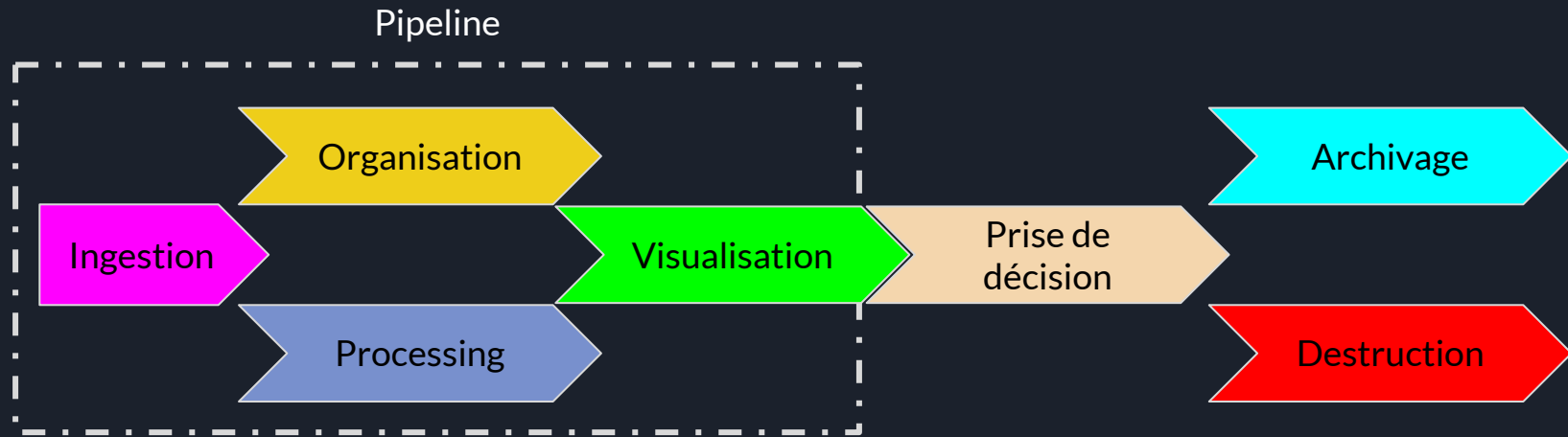
Capture, stockage, transformation et consommation des données



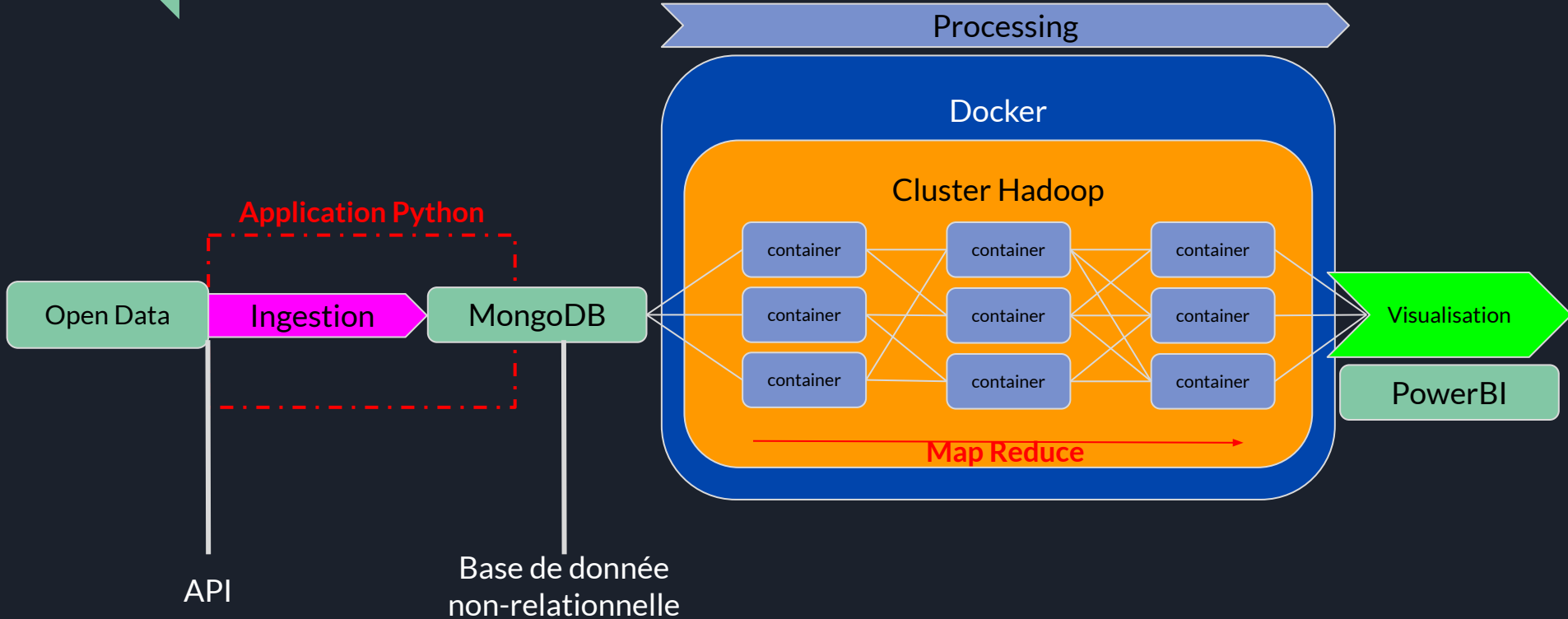
# Le cycle de vie de la donnée



# Le Pipeline du Big Data



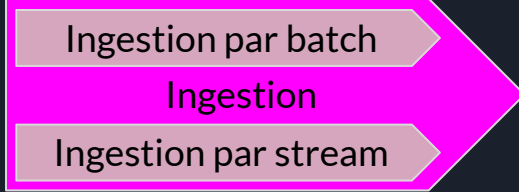
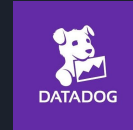
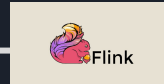
# Architecture d'un projet Big Data (Ingénieur)



# Pipeline outil final

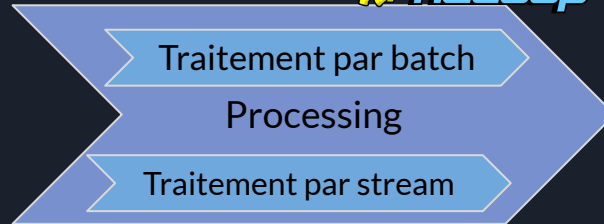


Pipeline



Fault Tolerance and Recovery


Google File System (GFS)





# Activité en groupe

Présentation sur l'importance des données dans une entreprise, revue de quelques exemples concrets. Au travers de vos expériences, montrez que les données que vous utilisez au quotidien sont un atout clé pour les entreprises et leur réussite.

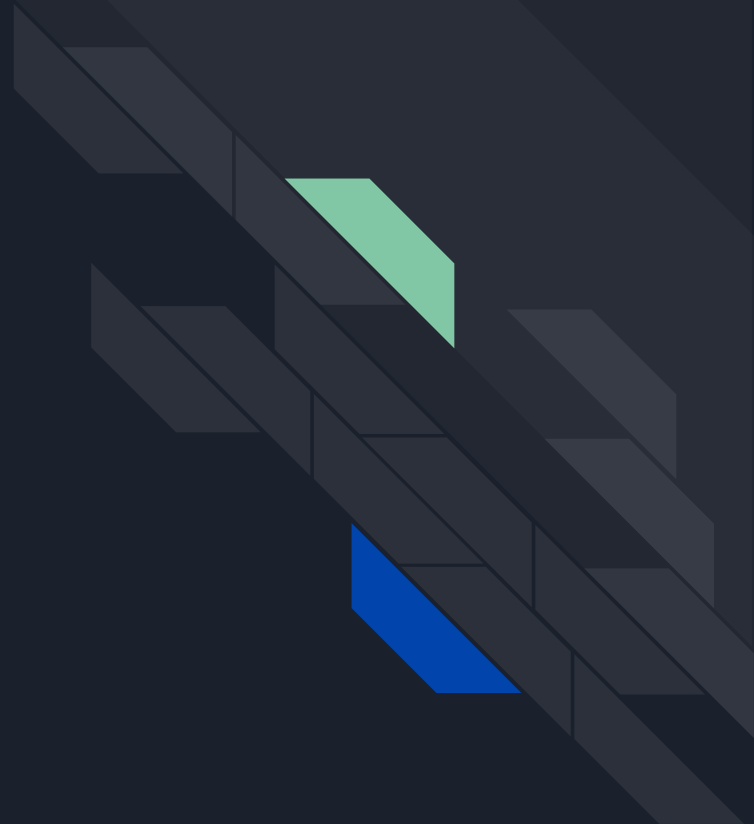


## Module 2 : Outils et technologies de base en Data Engineering (1/2)



# Introduction aux bases de données

BDD Relationnelles et Non-Relationnelles



# Bases de données Relationnelles (BDDR)





# Introduction

Les “BDD” ou “database” dans leur appellation commune, ont pour but de stocker, organiser et analyser les données.

Elles désignent une collection d’informations organisées afin de faciliter la consultation de données, leur gestion et leur mise à jour.

# Présentation des bases de données

Les bases de données sont aujourd'hui omniprésentes dans le monde des entreprises mais également dans le quotidien de tous.





# Historique des bases de données

Le terme de **base de données** est né en 1964 pour désigner une collection d'informations partagées par différents utilisateurs d'un système d'informations militaires.

Années 70 : Création des Bases de Données Réseaux

Ensemble de fichiers reliés par pointeurs

Langage d'interrogation par navigation



# Historique des bases de données

Années 80 : Avènement des Bases de données Relationnelles

Relations entre ensemble de données

Langage d'interrogation par assertion logique

Années 90 : Orientation décisionnelle (Data mining, OLAP)

Années 2000 : Avènement du Web

Années 2010 - 2020 : Avènement du Cloud pour les bases de données



# Présentation des bases de données

Il existe aujourd'hui 2 grands types de bases de données : **Relationnelles** et **Non Relationnelles**.



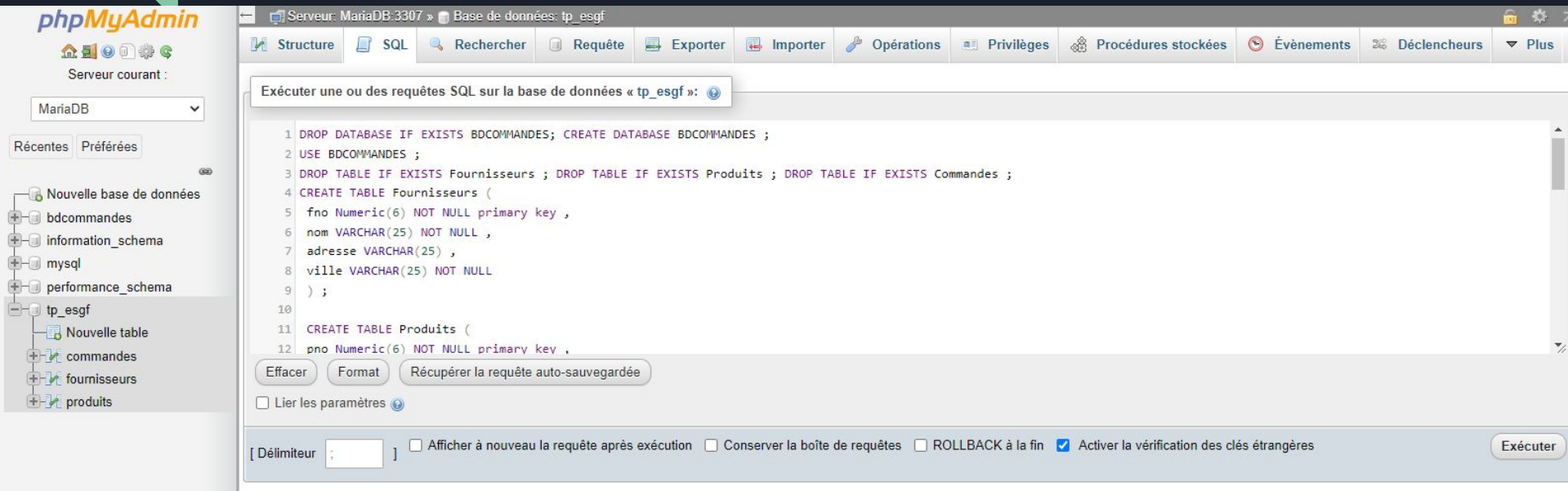
The diagram consists of a large light green rounded rectangle containing two yellow rounded rectangles. The light green rectangle is labeled 'Bases de données' at the top center. Inside it, on the left, is a yellow rectangle labeled 'Bases de données Relationnelles', and on the right is a yellow rectangle labeled 'Bases de données Non Relationnelles'.

## Bases de données

Bases de données  
Relationnelles

Bases de données  
Non Relationnelles

# Exemple de BDDR sur un serveur Apache via phpMyAdmin



The screenshot displays the phpMyAdmin web interface. On the left, the database structure tree shows 'tp\_esgf' selected, with sub-items 'commandes', 'fournisseurs', and 'produits'. The main panel is titled 'Serveur: MariaDB:3307 » Base de données: tp\_esgf'. It features a toolbar with icons for Structure, SQL, Rechercher, Requête, Exporter, Importer, Opérations, Privileges, Procédures stockées, Événements, Déclencheurs, and a Plus menu. Below the toolbar, a text box prompts to 'Exécuter une ou des requêtes SQL sur la base de données « tp\_esgf »:'. The SQL editor contains the following code:

```
1 DROP DATABASE IF EXISTS BDCOMMANDES; CREATE DATABASE BDCOMMANDES ;
2 USE BDCOMMANDES ;
3 DROP TABLE IF EXISTS Fournisseurs ; DROP TABLE IF EXISTS Produits ; DROP TABLE IF EXISTS Commandes ;
4 CREATE TABLE Fournisseurs (
5   fno Numeric(6) NOT NULL primary key ,
6   nom VARCHAR(25) NOT NULL ,
7   adresse VARCHAR(25) ,
8   ville VARCHAR(25) NOT NULL
9 ) ;
10
11 CREATE TABLE Produits (
12   pno Numeric(6) NOT NULL primary key ,
```

Below the editor are buttons for 'Effacer', 'Format', and 'Récupérer la requête auto-sauvegardée'. A checkbox 'Lier les paramètres' is present. At the bottom, there is a 'Délimiteur' field with a semicolon, a checkbox 'Afficher à nouveau la requête après exécution', a checkbox 'Conserver la boîte de requêtes', a checkbox 'ROLLBACK à la fin', a checked checkbox 'Activer la vérification des clés étrangères', and an 'Exécuter' button.



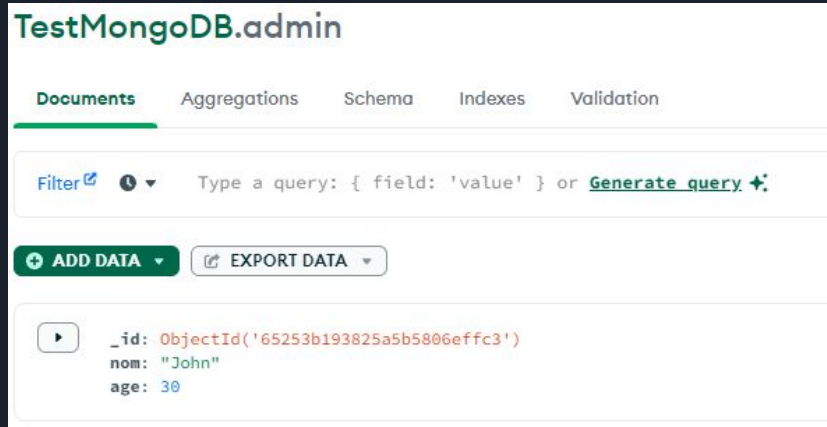
# Bases de données Non-Relationnelles



# Introduction

Les bases de données NoSQL sont conçues pour gérer des volumes massifs de données non **structurées** ou **semi-structurées**, ainsi que pour offrir une évolutivité horizontale.

La représentation et l'optimisation des données dans un contexte NoSQL nécessitent une compréhension approfondie des modèles de données, des requêtes et des besoins spécifiques de l'application.





# Modèles de Données en NoSQL

Les bases de données NoSQL prennent en charge divers modèles de données, notamment :

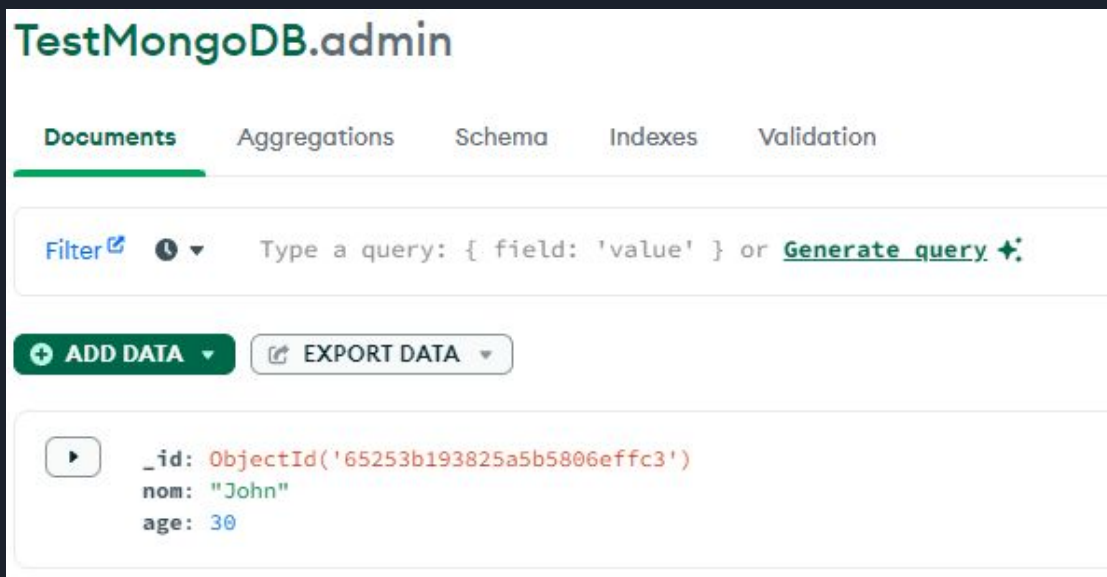
Document : Stocke les données dans des documents (par exemple, JSON ou BSON) pouvant être hiérarchiques.

JSON : JavaScript Object Notation

BSON : Binary JSON (développé par MongoDB)

# Modèle de données en NoSQL

Clé-Valeur : Stocke les données sous forme de paires clé-valeur.

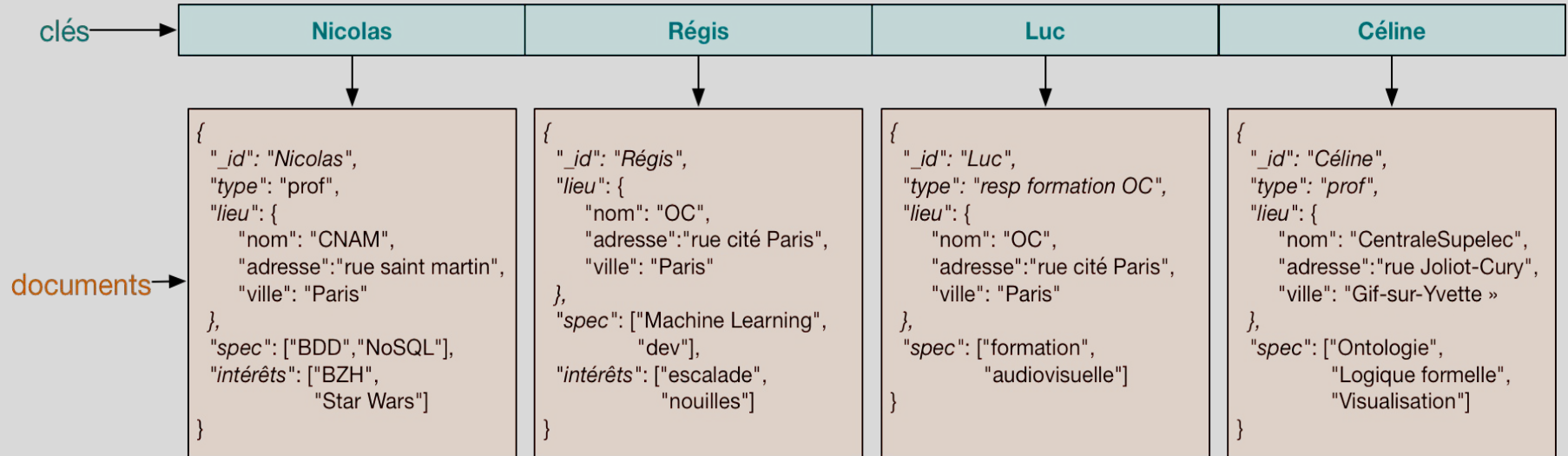


The screenshot displays the TestMongoDB.admin web interface. At the top, the title "TestMongoDB.admin" is shown in green. Below it, a navigation bar contains five tabs: "Documents" (highlighted with a green underline), "Aggregations", "Schema", "Indexes", and "Validation". A search bar with a "Filter" button and a clock icon is present, followed by the text "Type a query: { field: 'value' } or [Generate query](#) with a plus icon". Below the search bar are two buttons: a green "ADD DATA" button with a plus icon and a white "EXPORT DATA" button with a document icon. The main content area shows a single document with a play button icon on the left. The document's fields are: "\_id" with a red value "ObjectId('65253b193825a5b5806effc3')", "nom" with a red value "John", and "age" with a blue value "30".

```
{
  "_id": "ObjectId('65253b193825a5b5806effc3')",
  "nom": "John",
  "age": 30
}
```

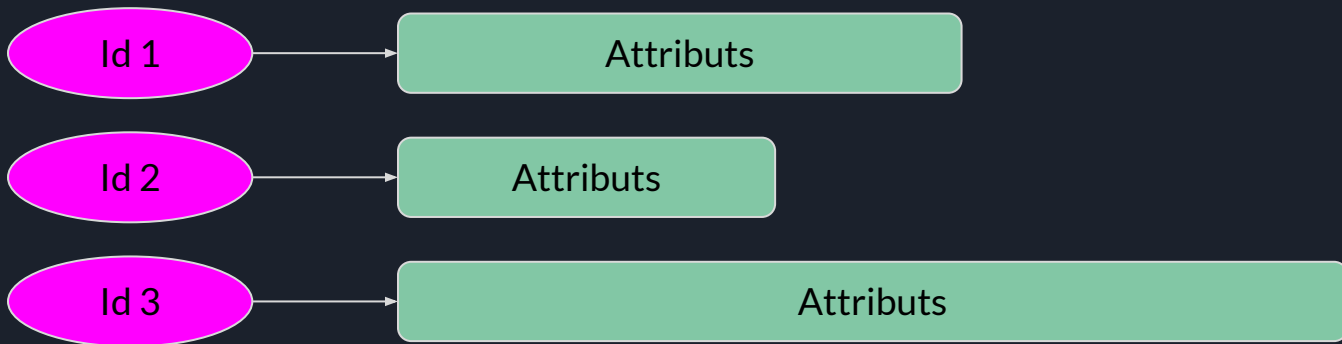
# Modèles de données en NoSQL

**Colonnes Familles :** Stocke les données sous forme de colonnes plutôt que de lignes, adapté aux cas d'utilisation de type base de données en colonnes.



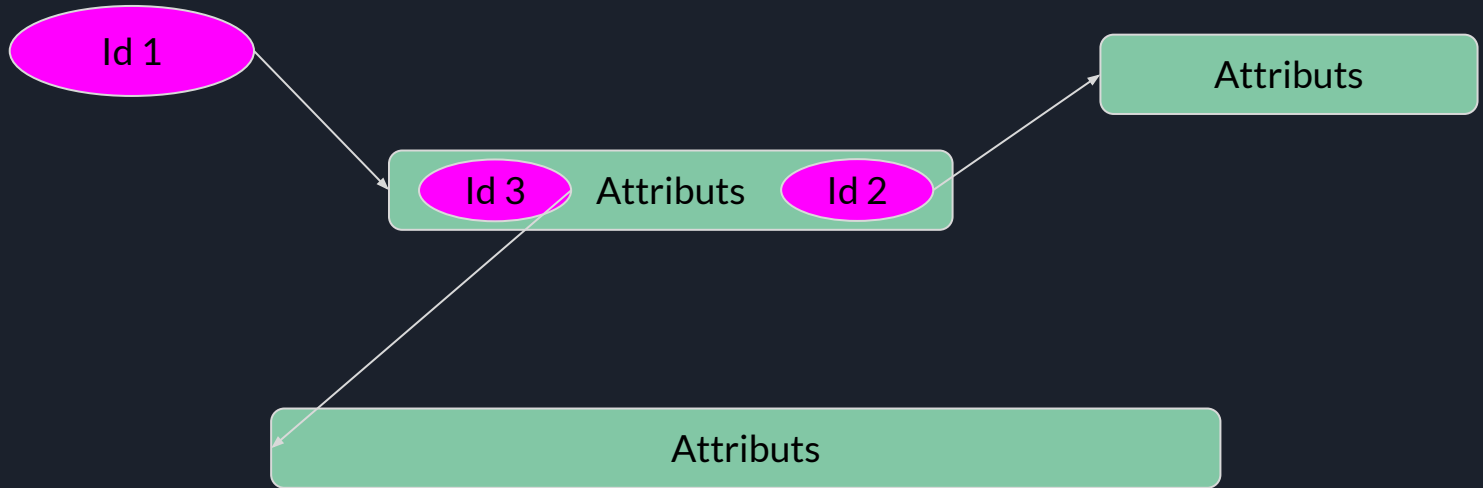
# Conception de Schéma Flexible

L'un des avantages clés des bases de données NoSQL est la flexibilité du schéma. Vous pouvez ajouter de nouveaux champs ou modifier la structure des données sans avoir à migrer une base de données complexe. Cette flexibilité permet de s'adapter rapidement aux besoins changeants de l'application.



# Modèle de données en NoSQL

Graphe : Stocke les données sous forme de graphes, adapté aux données interconnectées.





# MongoDB

MongoDB est une base de données NoSQL documentaire qui stocke les données dans un format de document JSON appelé BSON (Binary JSON). Voici quelques caractéristiques clés de MongoDB :

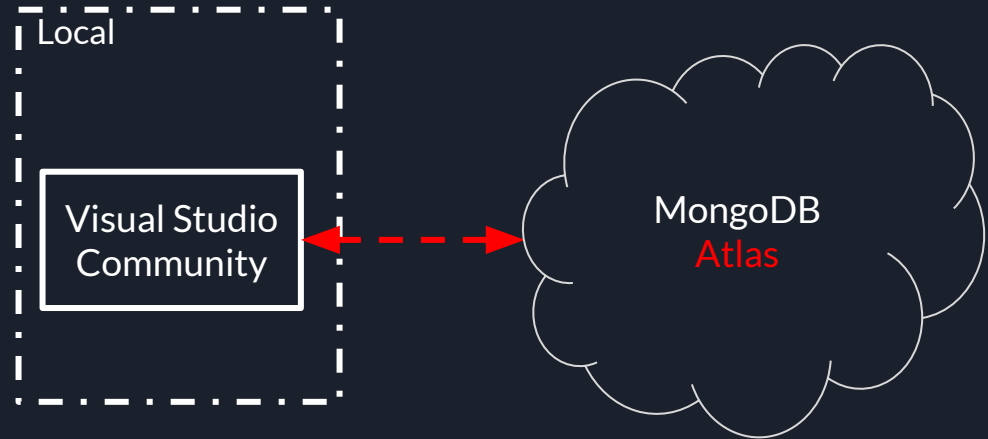
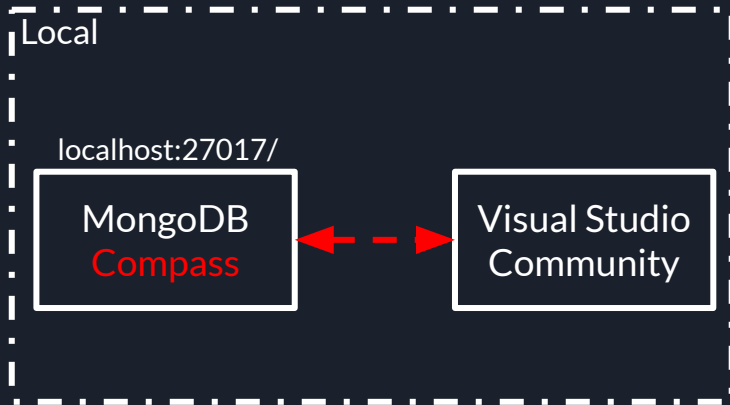


# MongoDB Atlas et MongoDB Compass



# MongoDB Atlas vs MongoDB Compass

Utilisation de mongoDB en local vs Cloud





# AWS DynamoDB



**Amazon**  
DynamoDB

# Qu'est-ce qu'Amazon DynamoDB ?

---

Service de base de données NoSQL rapide et souple, adapté à toutes les échelles



**Amazon DynamoDB**

- Tables de base de données NoSQL
- Stockage virtuellement illimité
- Les éléments peuvent avoir différents attributs
- Requêtes à faible latence
- Débit de lecture/écriture pouvant être mis à l'échelle

## Composants de base d'Amazon DynamoDB

---

- Les tables, les éléments et les attributs sont les composants de base de DynamoDB
- DynamoDB prend en charge deux sortes de clés primaires :  
clé de partition et clé de partition et de tri

# Partitionnement

---



Partitionnement de table par clé à mesure que les données augmentent

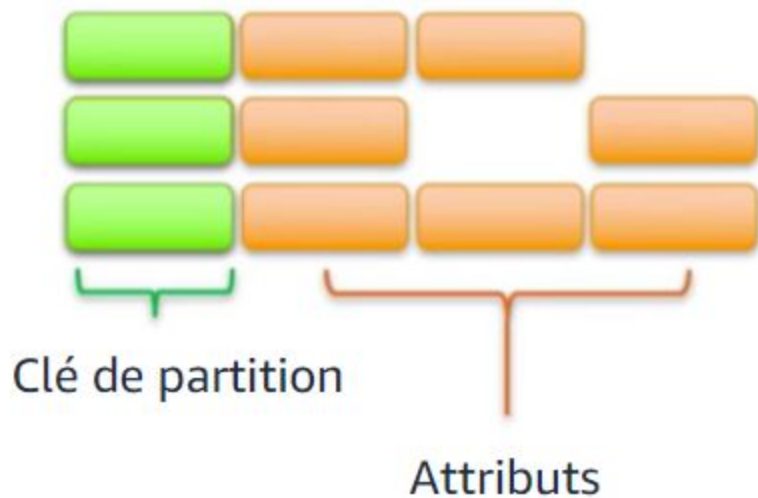


QUERY par clé pour trouver des éléments efficacement  
SCAN pour trouver des éléments selon n'importe quel attribut

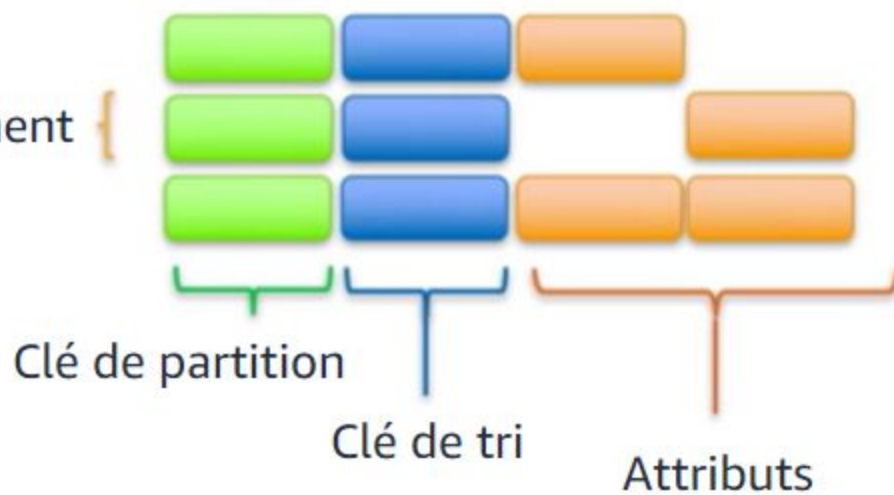


# Les éléments dans une table doivent avoir une clé

Clé unique



Clé composite





## Points clés à retenir de la section 2



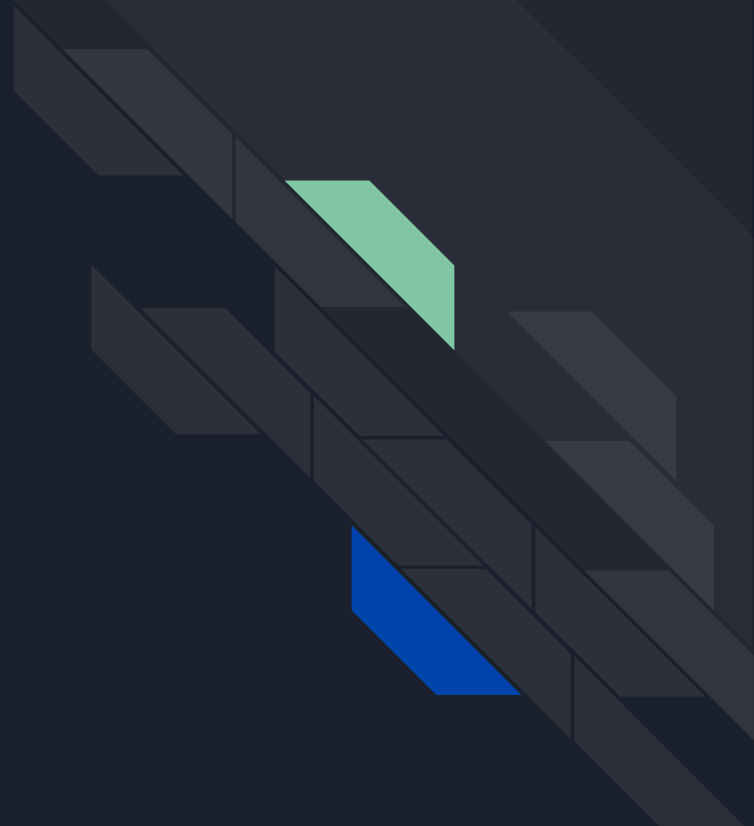
### Amazon DynamoDB :

- Fonctionne exclusivement sur SSD.
- Prend en charge les modèles de stockage de documents et de magasin clé-valeur.
- Réplique automatiquement vos tables dans les régions AWS de votre choix.
- Convient parfaitement aux applications mobiles, web, de jeux, publicitaires et Internet des objets (IoT).
- Est accessible depuis la console, AWS CLI et les appels d'API.
- Assure une latence constante inférieure à 10 millisecondes à toute échelle.
- N'impose pas de limite de taille de table ni de débit.



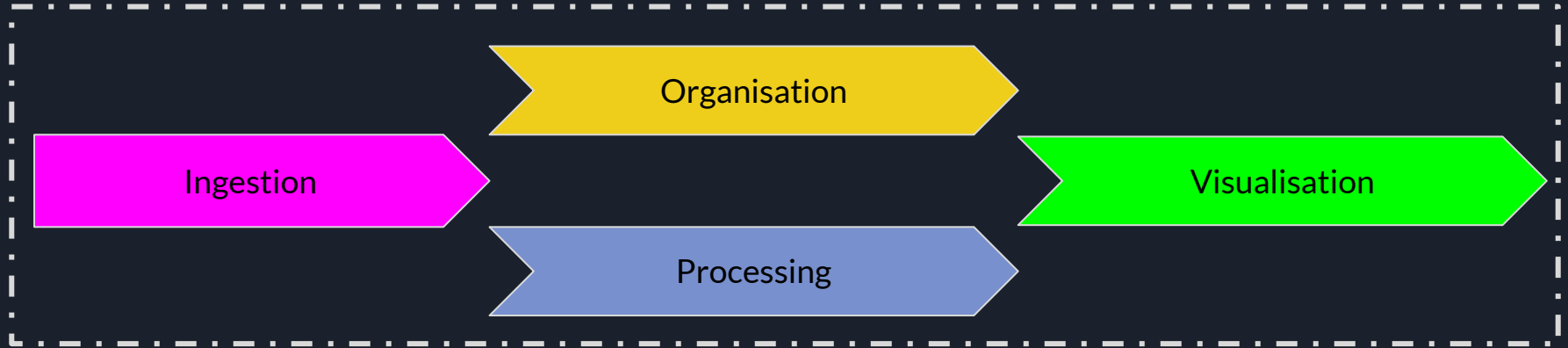
# Outils d'ingestion des données

Ingestion par batch vs ingestion par stream



# Introduction

Pipeline





# Ingestion par batch vs ingestion par stream

L'ingestion de données, c'est le processus de collecte et de transport de données depuis différentes sources vers un système de stockage centralisé ou une base de données pour les analyser. Il existe deux principales approches pour l'ingestion de données : par batch et en temps réel.

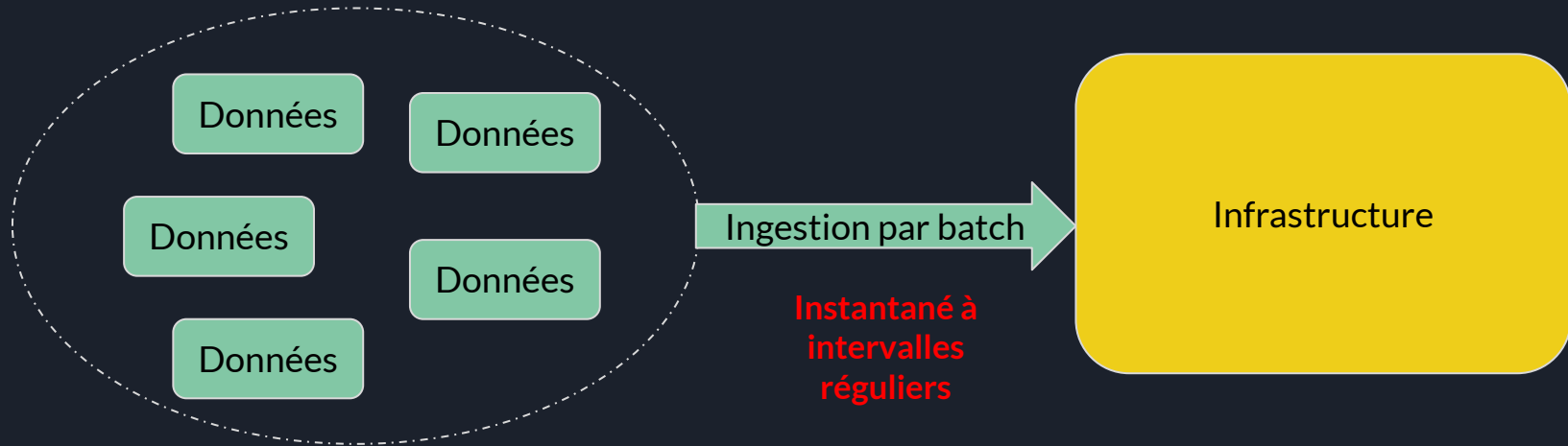
Ingestion par **batch**

Ingestion par **stream**  
(temps réel)

# Ingestion par batch vs ingestion par stream

L'ingestion **par batch** consiste à regrouper et à traiter un **ensemble de données à des intervalles réguliers**. Par exemple, les données peuvent être collectées toutes les heures, toutes les nuits, ou une fois par jour, puis traitées et ingérées en une seule opération.

Cette méthode est souvent utilisée pour des processus où le temps n'est pas un facteur critique, comme les rapports quotidiens, les analyses rétrospectives, ou le traitement de grandes quantités de données accumulées sur une période donnée.





# Ingestion par batch vs ingestion par stream

## Avantages

Traiter de grandes quantités de données à la fois de manière optimisée

Simple à mettre en place et à gérer

## Inconvénients

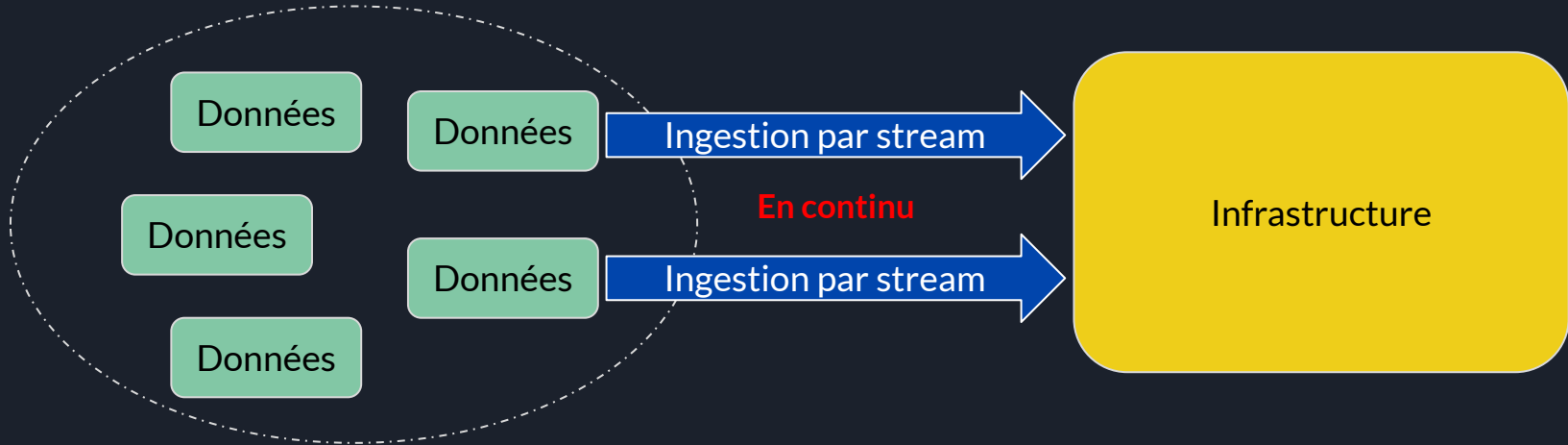
Les données ne sont pas disponibles immédiatement après leur génération.

Si une réponse immédiate est nécessaire, l'ingestion par batch n'est pas appropriée.

# Ingestion par batch vs ingestion par stream

L'ingestion **en temps réel** (ou **stream processing**) consiste à traiter et à ingérer les données au fur et à mesure qu'elles sont générées. Les données sont collectées, traitées, et stockées sans délai significatif.

Cette méthode est utilisée dans les scénarios où la rapidité est essentielle, comme la surveillance de systèmes en temps réel, les transactions financières, les analyses de flux de clics, ou les applications IoT (Internet of Things).





# Ingestion par batch vs ingestion par stream

## Avantages

Les données sont disponibles quasi instantanément. La réactivité du SI est augmentée

Utile pour les applications où les conditions changent rapidement et nécessitent une prise de décision immédiate

## Inconvénients

Ce type d'ingestion est long à mettre en place car elle nécessite une infrastructure et des systèmes capable de traiter des données en continu

Gourmande en ressource, demande un traitement constant et rapide des données en entrée.



# Ingestion par batch vs ingestion par stream

## Ingestion par **batch**

Traite des ensemble de données à intervalles réguliers. Convient pour des applications où la vitesse n'est pas cruciale.

## Ingestion par **stream** (temps réel)

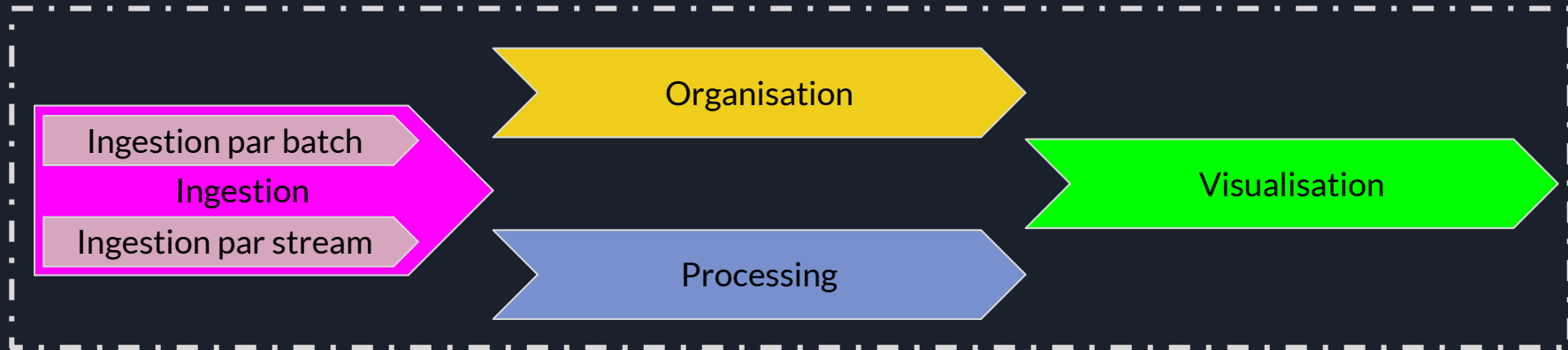
Traite les données immédiatement après leur création. Essentiel pour des applications nécessitant des réponses instantanées ou une analyse continue des données.

Le choix entre ces deux méthodes dépend des besoins spécifiques du système ou de l'application en question, en termes de latence, de volume de données, de complexité, et de ressources disponibles.



# Introduction

Pipeline



# Outils d'ingestion des données

Introduction à Apache Kafka



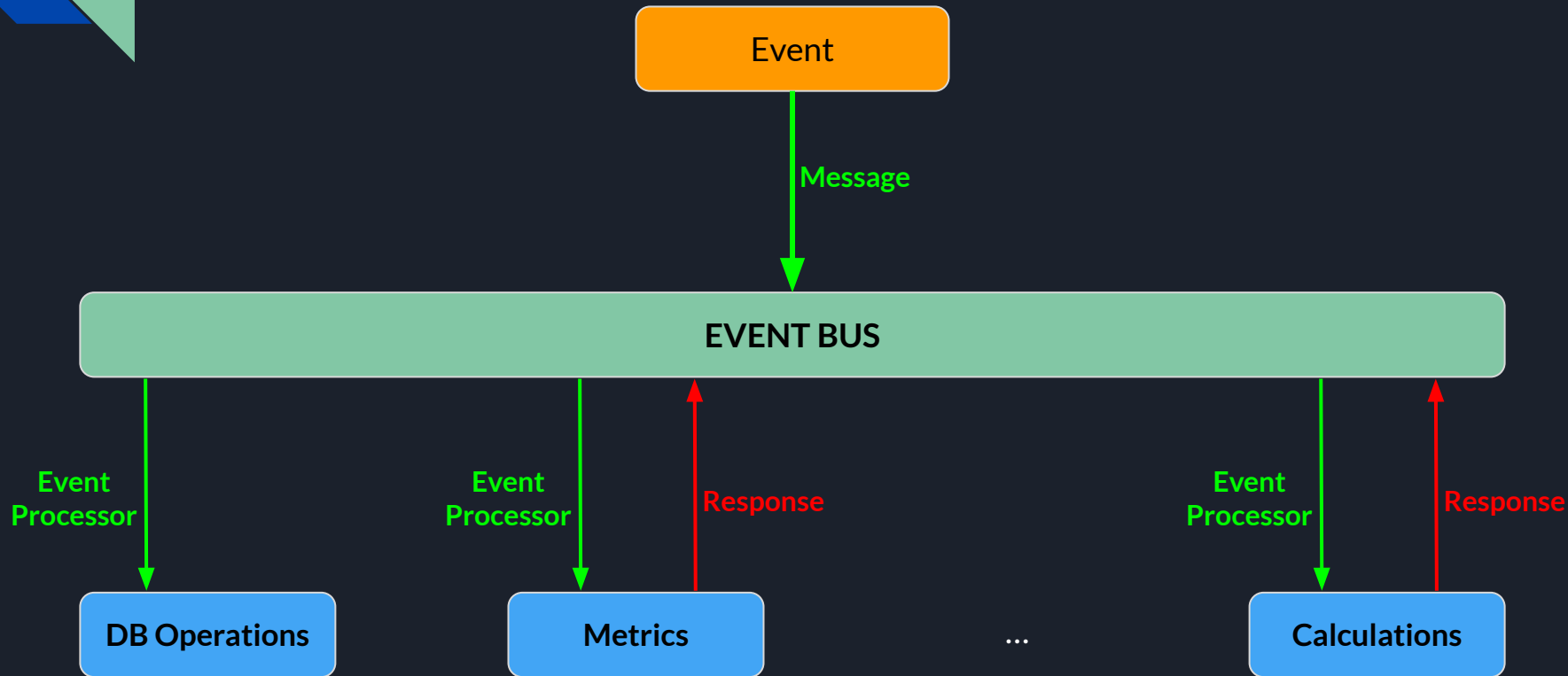


# L'Event Bus

Ce pattern consiste à mettre en place un mécanisme de **communication asynchrone** entre les différents éléments d'une application, en utilisant **un bus de messages ou un système de notification**. L'Event Bus permet de découpler les échanges de messages entre les services, de manière à améliorer la flexibilité et la robustesse de l'application.

Il s'agit d'une couche de communication indépendante qui permet de **découpler** les différents services et composants de l'application, de manière à ce qu'ils n'aient pas à se connaître ou à se dépendre les uns des autres.

# L'Event Bus



# Mise en place d'un Event Bus

## Un bus de messages

Permet de souscrire et publier des messages asynchrones en utilisant un **broker**\*



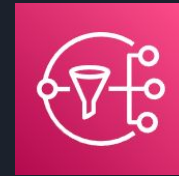
## Un bus d'événements

Permet de souscrire et publier des événements asynchrones en utilisant un **système de gestion d'événements**



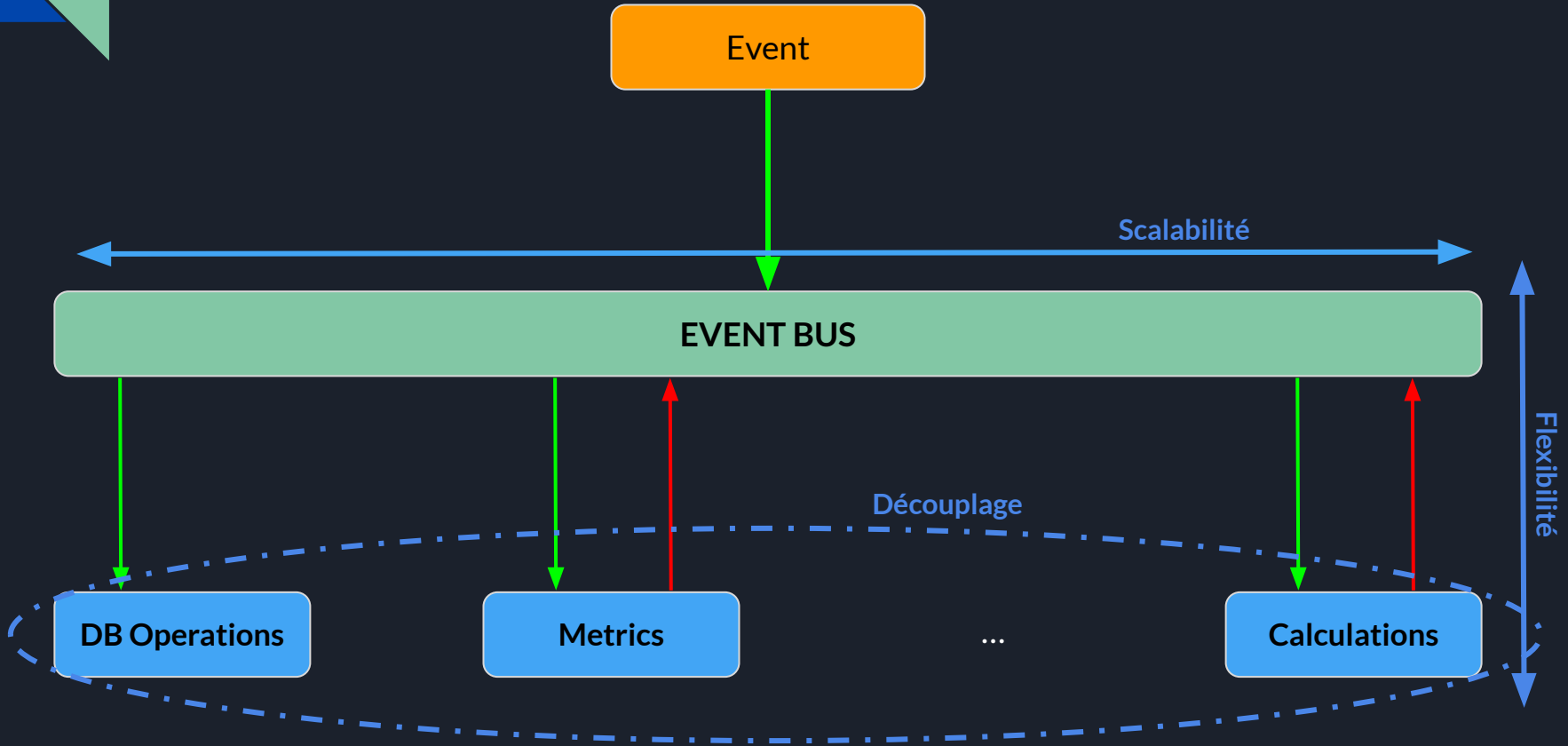
## Un bus de notification

Permet de souscrire et publier des notifications asynchrones en utilisant un **système de gestion de notification**



\*broker : Serveur avec lequel les clients communiquent

# Avantages d'un event Bus



# Apache Kafka

Apache Kafka est une plateforme de streaming distribuée open-source, conçue pour traiter en temps réel des flux de données massifs de manière fiable, rapide et évolutive. Initialement développée par LinkedIn, elle a été open-source en 2011 et est maintenant gérée par la fondation Apache.

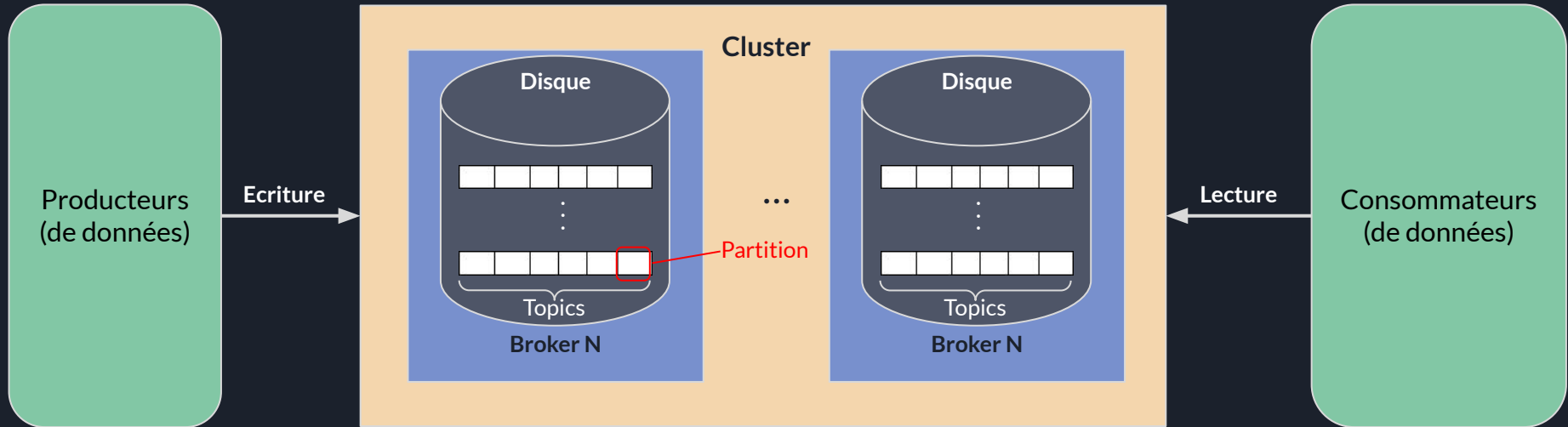


*Apache Kafka*

<https://kafka.apache.org/>

# Apache Kafka

## Les principaux composants d'un Cluster Kafka

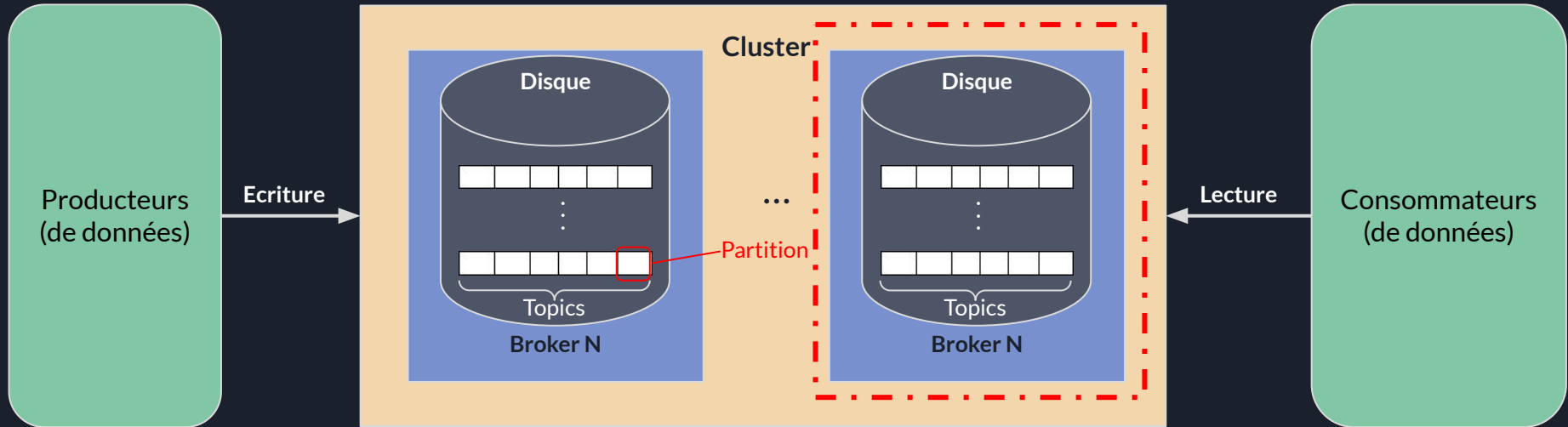




# Apache Kafka

## Les Brokers :

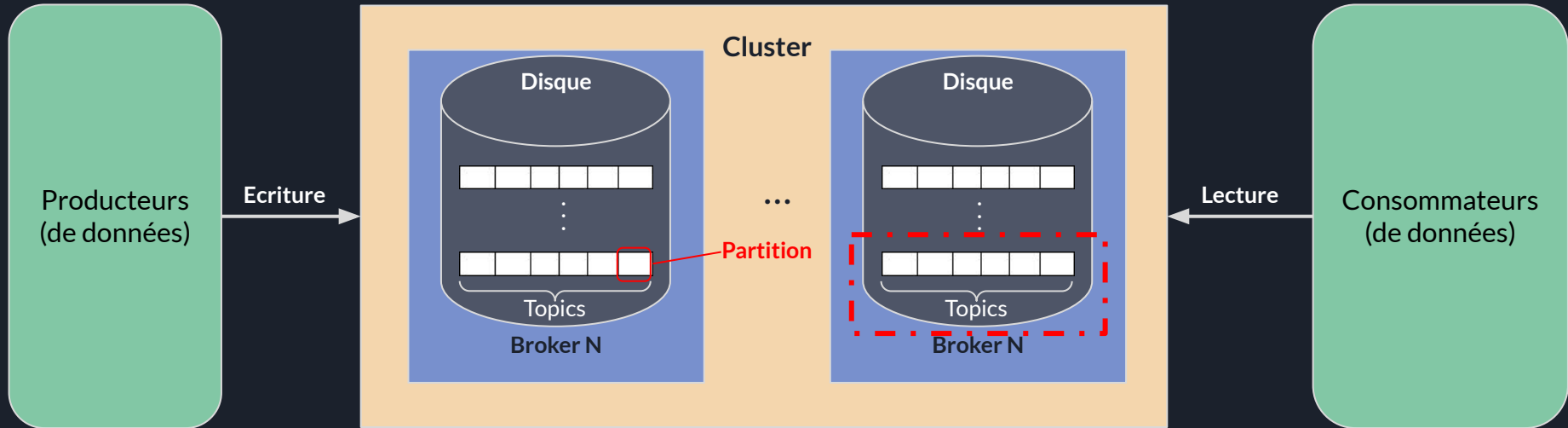
Serveurs Kafka qui stockent les données  
et servent les consommateurs



# Apache Kafka

## Les Topics :

Canaux logiques où les données sont envoyées. Chaque topic est subdivisé en Partition



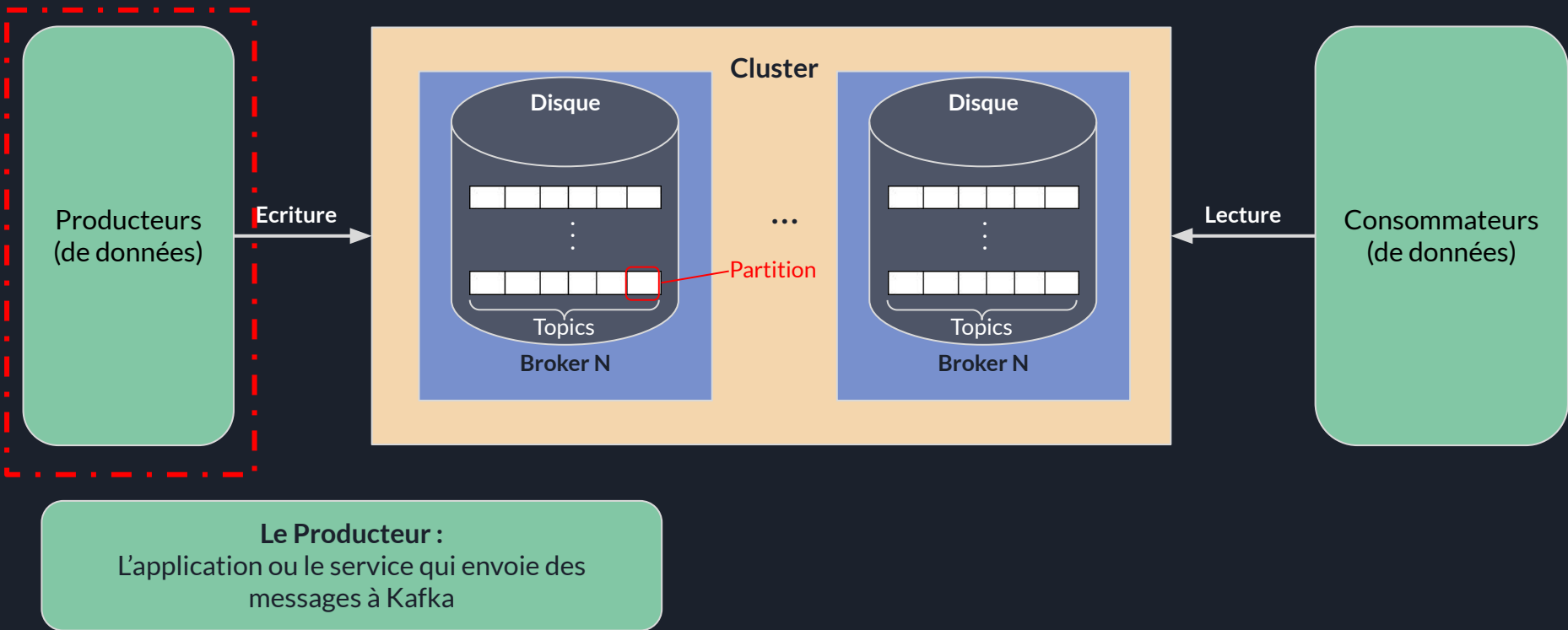
## Les Partitions:

Une subdivision d'un topic qui permet de paralléliser le traitement des données.

Kafka est fondamentalement un **système de log distribué**.

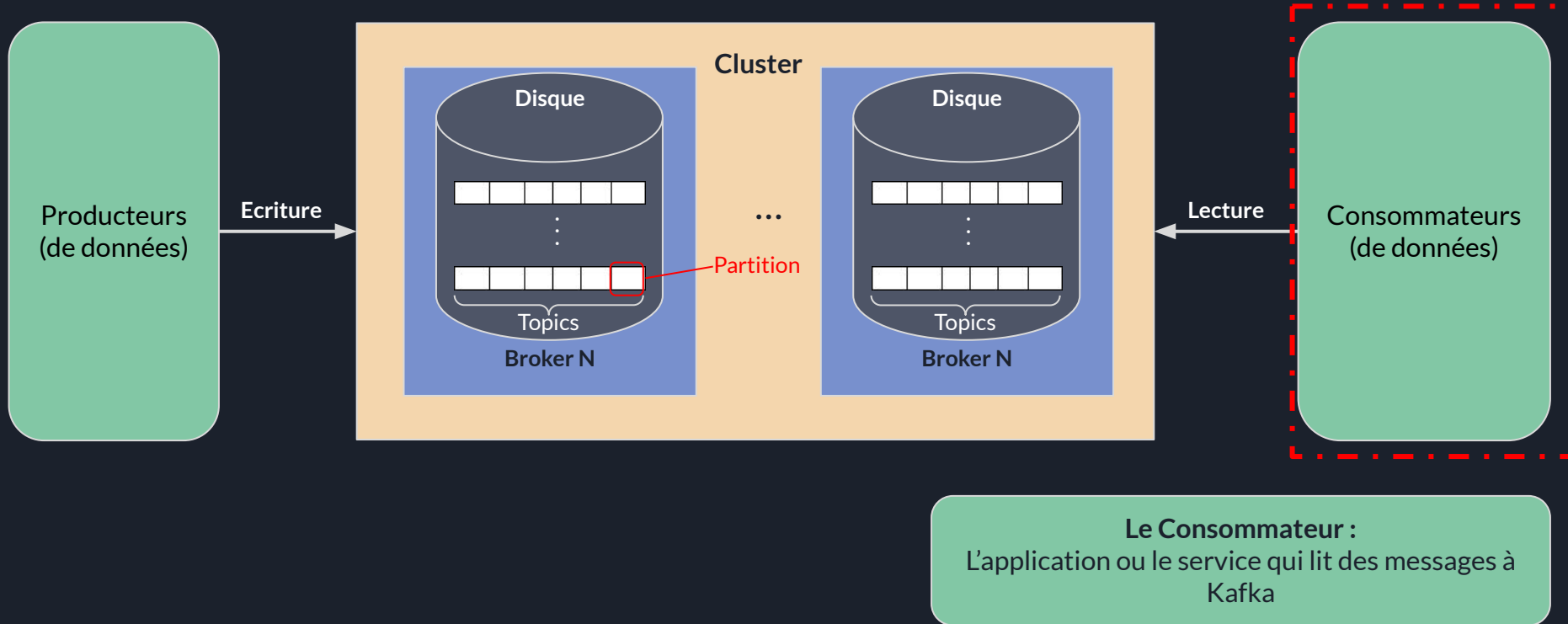
# Apache Kafka

Un producteur envoie des messages à un topic, et un consommateur les lit de manière asynchrone.

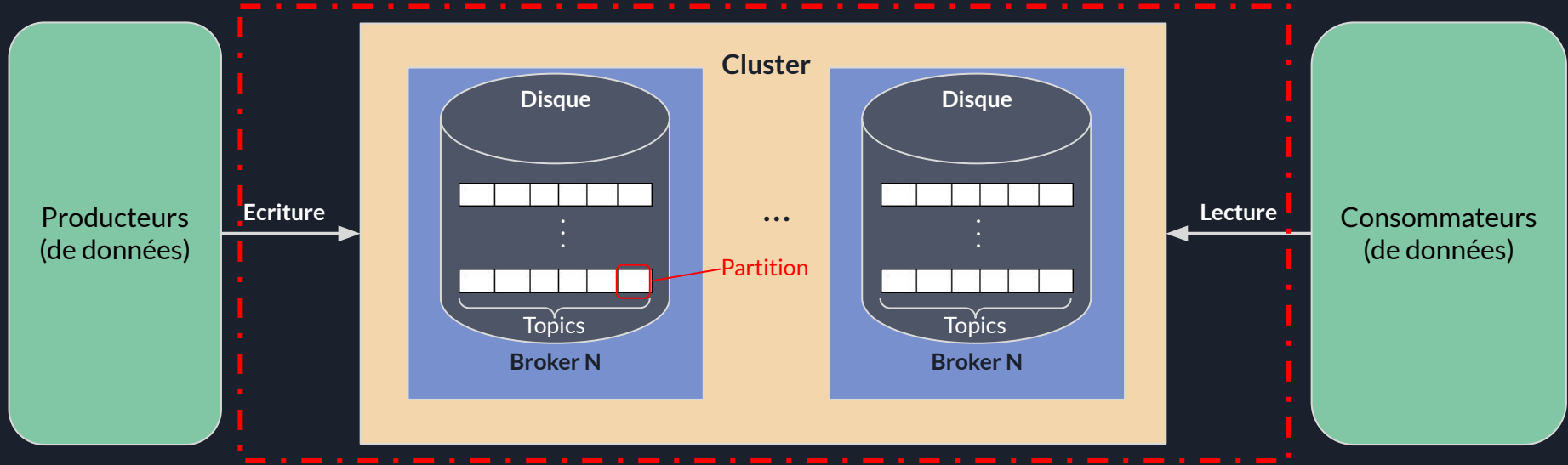


# Apache Kafka

Un producteur envoie des messages à un topic, et un consommateur les lit de manière asynchrone.



# Apache Kafka



## Apache Zookeeper :

Utilisé par Kafka pour la coordination du cluster, comme le suivi de l'état des brokers

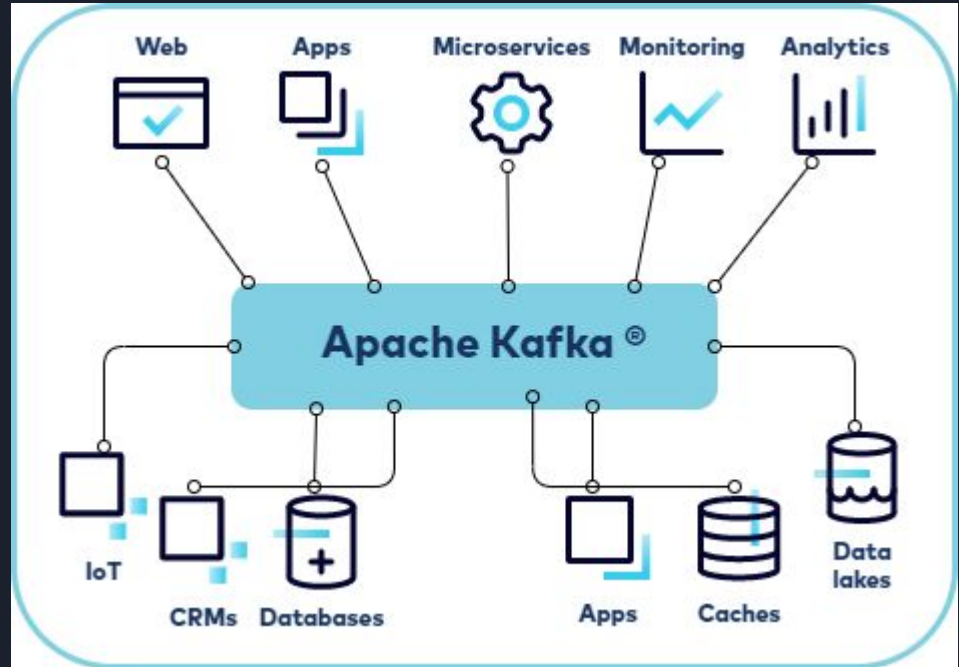
# Apache Kafka

Les principales caractéristiques de Kafka

Système Distribué

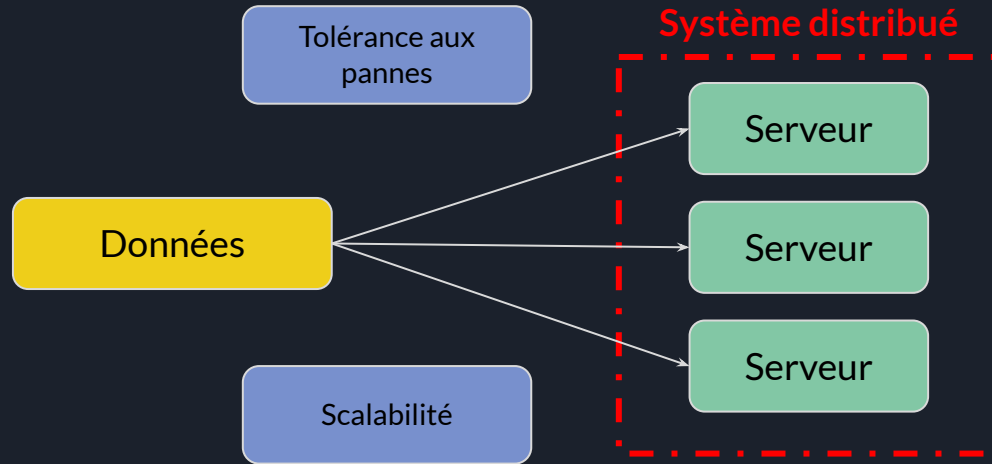
Gestion des logs

Producteur Consommateur



# Apache Kafka

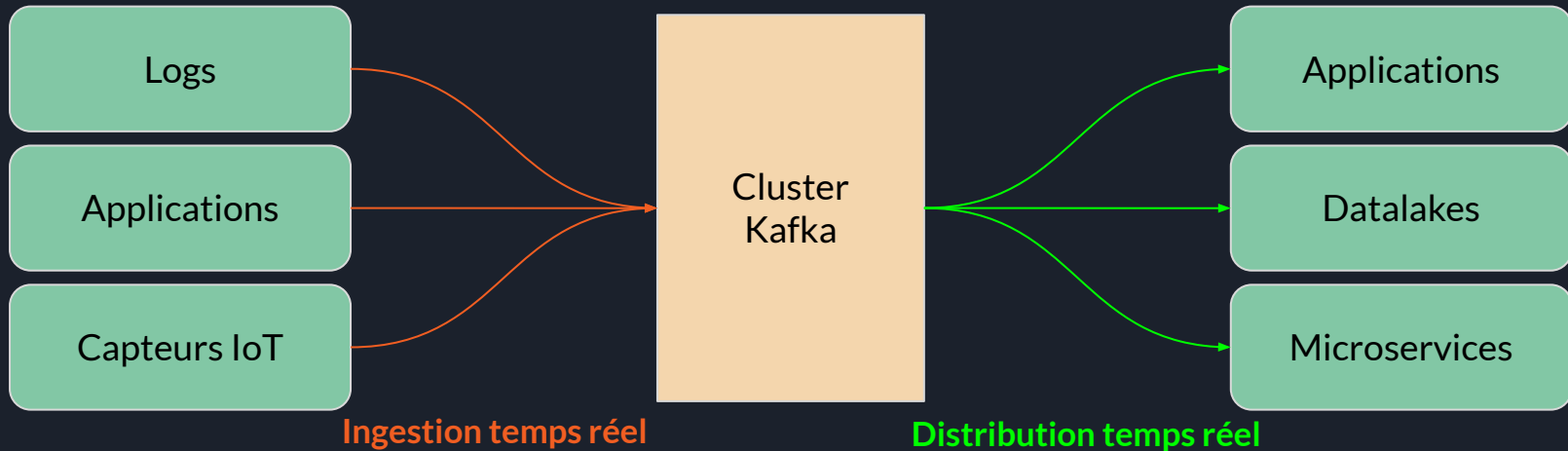
Système distribué : Kafka est conçu pour fonctionner sur un cluster de serveurs, où les données sont réparties sur plusieurs nœuds pour assurer la tolérance aux pannes et la scalabilité.



Kafka réplique les données sur plusieurs serveurs (appelés "brokers") pour garantir que les données ne sont pas perdues en cas de panne d'un ou plusieurs serveurs.

# Apache Kafka

Traitement en temps réel : Kafka permet le traitement en temps réel des flux de données. Il est couramment utilisé pour ingérer des données provenant de diverses sources (comme des logs, des capteurs IoT, etc.), les traiter en temps réel, puis les distribuer à divers systèmes ou applications.

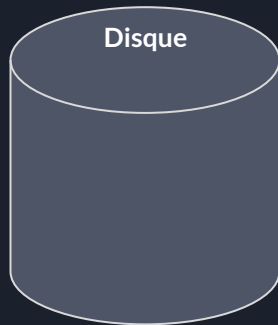






# Apache Kafka

Kafka stocke les messages sur disque, ce qui permet de relire les messages plus tard, même après qu'ils ont été consommés.



Persistence des données



# Apache Kafka

En résumé, Apache Kafka est une plateforme robuste et flexible pour gérer et traiter des flux de données à grande échelle, avec des applications dans de nombreux domaines allant du traitement en temps réel à l'intégration de systèmes complexes.

# TP : Apache Kafka

Rendez vous sur la page suivante :

<https://kafka.apache.org/quickstart>

Et réalisez le TP d'installation de Kafka

/!\ Il est indispensable d'avoir installé et paramétré Docker pour réaliser ce TP /!\

Rendu : Compte rendu de TP avec captures d'écran et explications de vos fenêtres.

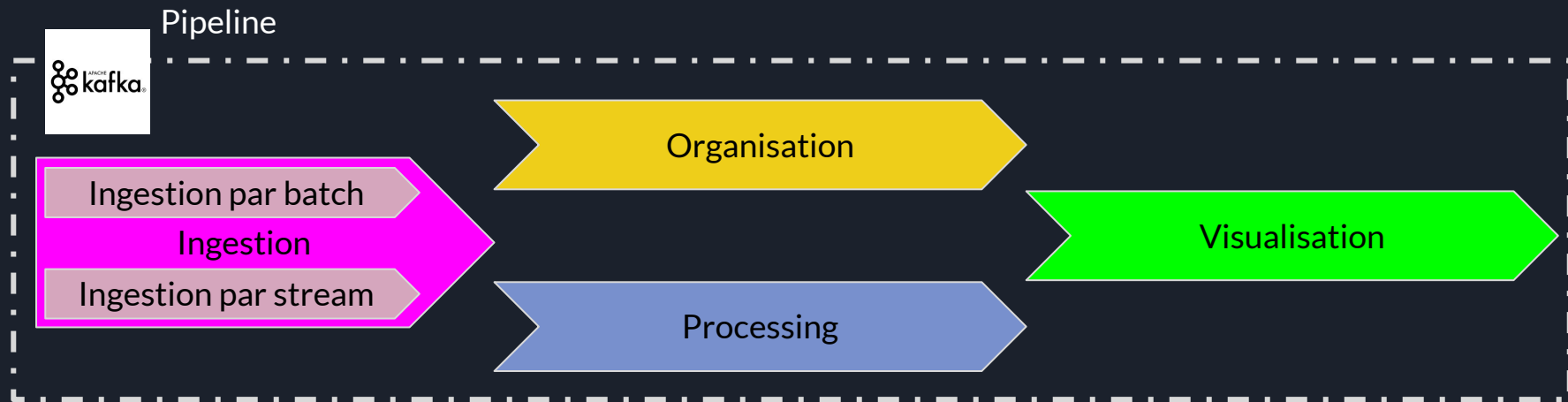
Format PDF obligatoire


Date de rendu :

Adresse de rendu : [yann.fornier@gmail.com](mailto:yann.fornier@gmail.com)



# Pipeline outil



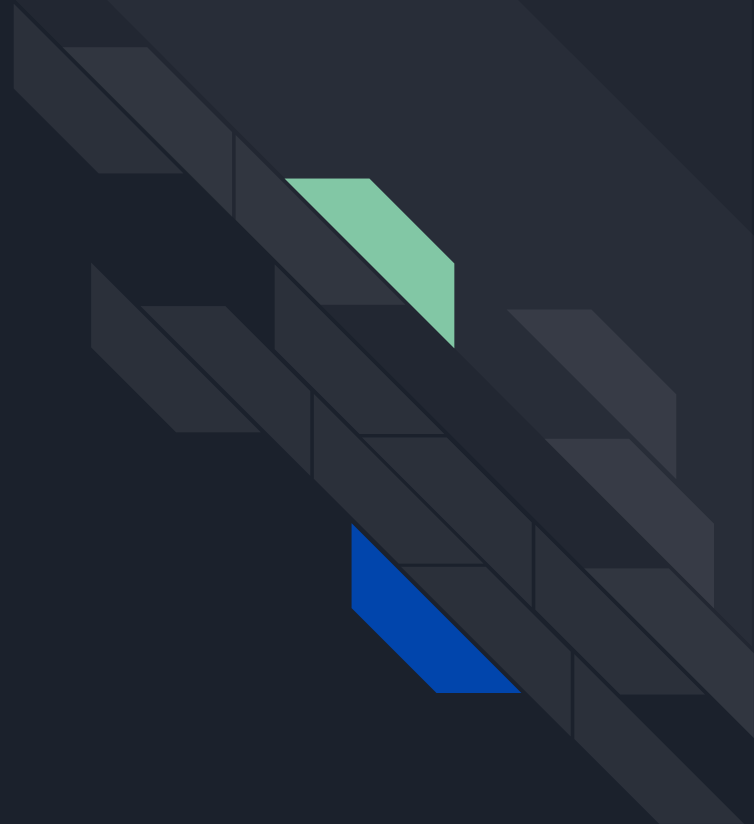


# Module 3 : Outils et technologies de base en Data Engineering (2/2)

# Stockage et transformation des Données

Traitement par Batch vs Traitement par Stream

Introduction à Apache Hadoop et Spark





# Traitement par batch vs traitement par stream

Traitement par **batch**

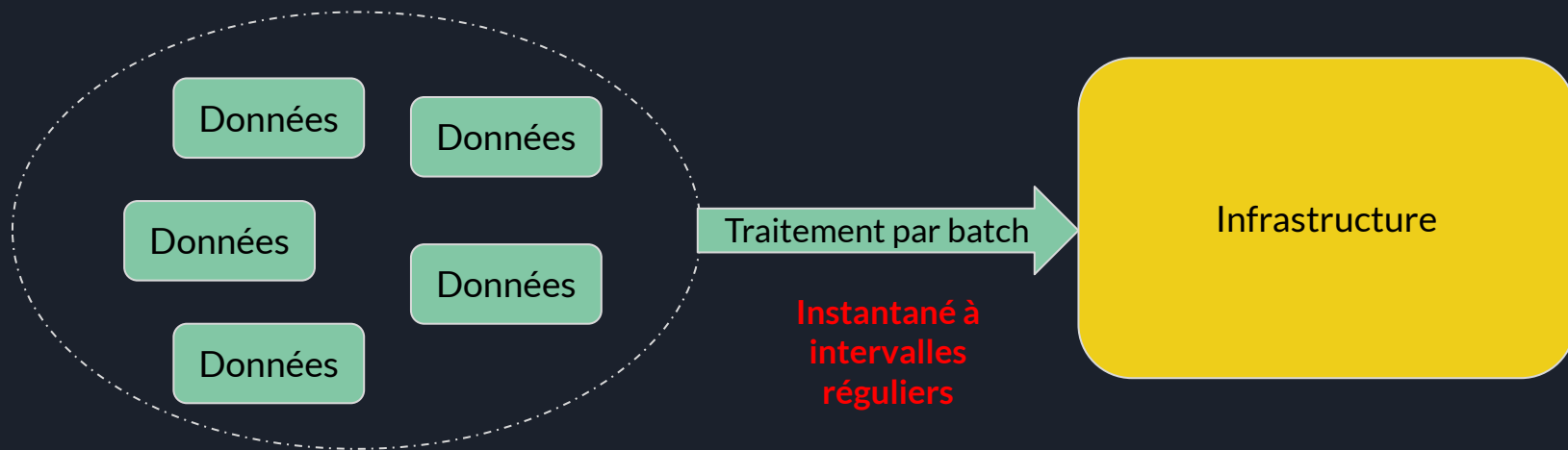
Traitement par **stream**  
(temps réel)

# Traitement par batch vs Traitement par stream

Le traitement par batch consiste à accumuler des données sur une période donnée et à les traiter toutes ensemble, en une seule opération. Les données sont collectées, stockées, puis traitées à intervalles réguliers, selon un plan prédéfini (par exemple, toutes les heures, tous les jours, ou toutes les semaines).

Les données sont groupées en "**lots**" (ou "**batches**") et chaque lot est traité en une seule opération.

Exemples d'utilisation : Calcul des fiches de paie mensuelles, génération de rapports financiers, sauvegardes de bases de données, agrégation de logs sur une journée, etc.







# Traitement par batch vs Traitement par stream

## Avantages

Plus facile à gérer et à mettre en œuvre pour des processus bien définis.

Efficacité en termes de ressources :  
Permet de traiter de grandes quantités de données en utilisant les ressources de manière optimisée.

Moins d'impact sur les systèmes en production : Le traitement peut être planifié pendant les périodes de faible utilisation.

## Inconvénients

Le traitement des données n'est pas immédiat, ce qui introduit un délai (ou latence) entre la collecte des données et leur disponibilité après traitement.

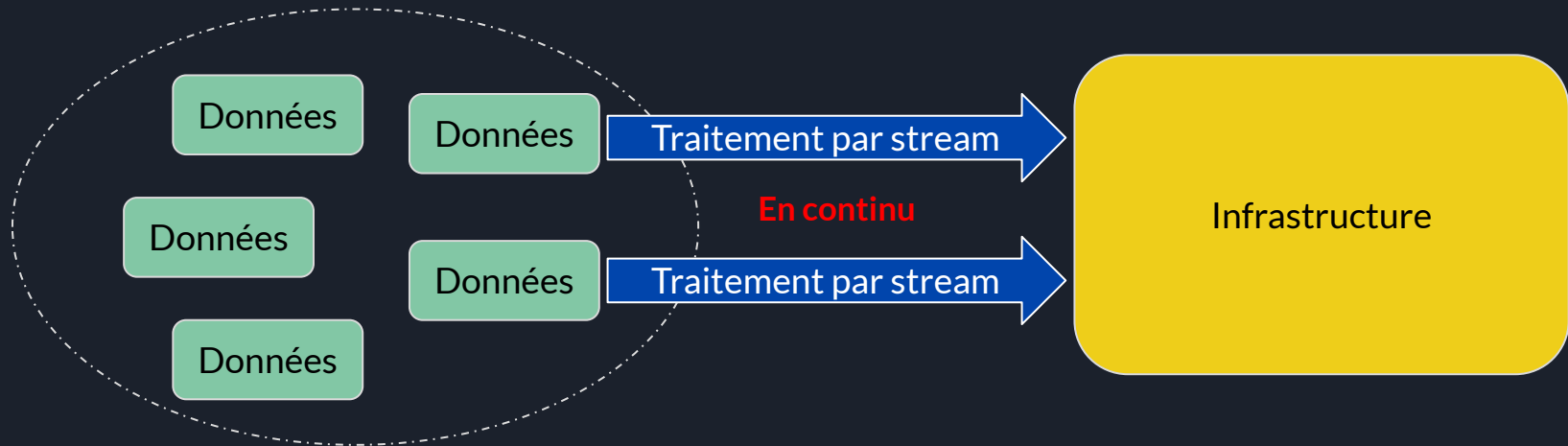
Risque d'échec global : Si un batch échoue, l'ensemble du lot doit souvent être retraité, ce qui peut entraîner des retards.

# Traitement par batch vs Traitement par stream

Le traitement en streaming consiste à traiter les données de manière continue et en temps réel, au fur et à mesure qu'elles arrivent. Les données sont traitées immédiatement, sans attendre qu'un lot complet soit accumulé. Les données sont ingérées et traitées en continu, souvent en petites portions ou événements. Les résultats sont produits quasi instantanément après que les données sont générées ou reçues.

Les systèmes de traitement en streaming doivent être capables de gérer de gros volumes de données en continu et de s'adapter à des variations de charge.

Exemples d'utilisation : Surveillance des transactions financières pour détecter les fraudes, suivi des véhicules en temps réel, traitement des flux de clics sur un site web, surveillance d'objets connectés (IoT), etc.





# Traitement par batch vs Traitement par stream

## Avantages

Permet de réagir instantanément à des événements ou des changements dans les données.

Utile pour les systèmes nécessitant une analyse continue et des actions immédiates.

Peut s'adapter aux changements de volume de données sans interruption.

## Inconvénients

Plus difficile à concevoir, à mettre en place, et à maintenir en raison des exigences en matière de traitement en temps réel.

Nécessite souvent plus de ressources pour garantir une faible latence et une haute disponibilité.

Défis liés à la précision : Comme les données sont traitées immédiatement, il peut être plus difficile de corriger les erreurs ou de retravailler les données après coup.



# Traitement par batch vs Traitement par stream

## Traitement par **batch**

Adapté aux tâches où la latence n'est pas un problème et où de grandes quantités de données peuvent être traitées de manière ponctuelle.

## Traitement par **stream** (temps réel)

Nécessaire lorsque la réactivité est cruciale, comme dans les systèmes qui nécessitent des décisions en temps réel ou une analyse continue des données.

Le choix entre ces deux approches dépend des exigences spécifiques du système ou de l'application, notamment en termes de latence, de volume de données, de complexité de traitement, et de ressources disponibles. Dans certains cas, une combinaison des deux (par exemple, un système hybride qui traite certaines données en streaming et d'autres par batch) peut être la meilleure solution.



# Etude de cas : Etude des solutions de traitement par Batch et par Stream

Par groupes, vous allez effectuer une recherche et une présentation sur les solutions du marché suivantes :

Groupe 1

Google File System

Groupe 2

Hadoop

Groupe 3

Apache Sparks

Groupe 4

Apache Cassandra



# Google File System (GFS)

*Remplacé par Colossus*

GFS a été conçu pour répondre aux besoins de stockage de données dans les applications Google. il est optimisé pour la gestion de fichiers de taille importante.

Fault Tolerance and Recovery

# Google File System (GFS)



# Google File System (GFS)

GFS est un système de fichiers distribué développé par Google pour gérer efficacement de grandes quantités de données.

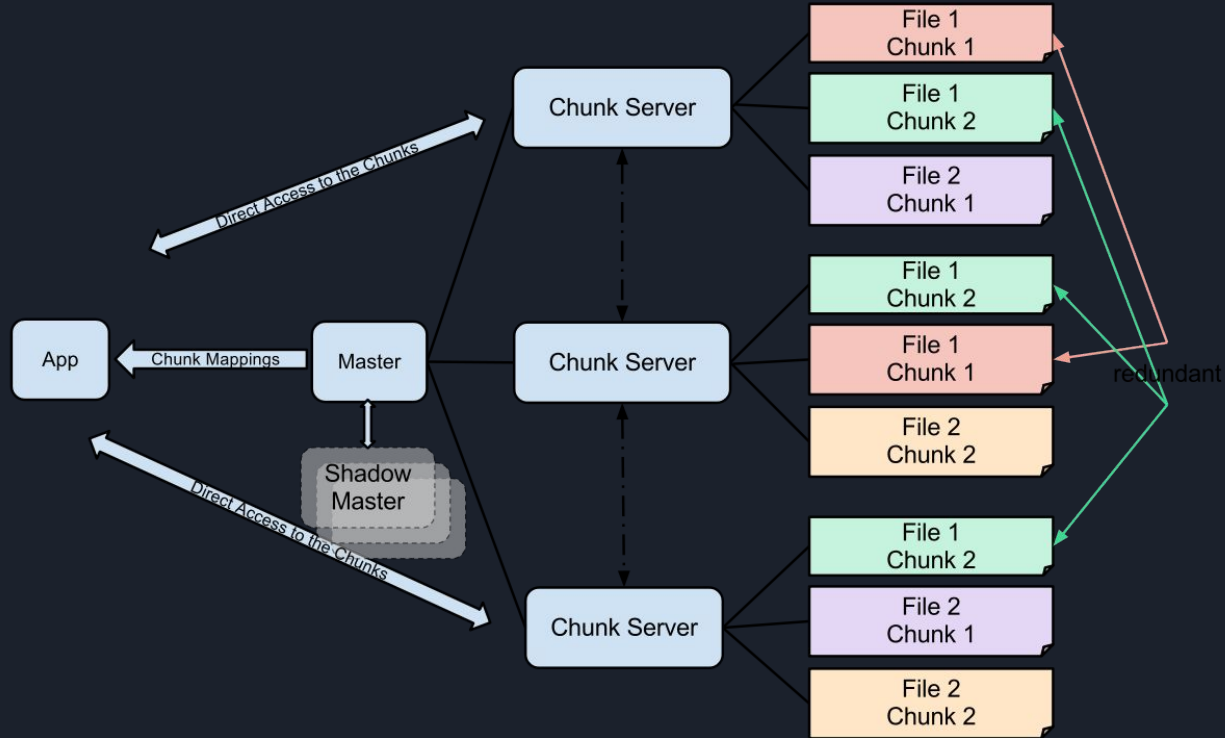
Redondance des données

GFS réplique  
automatiquement les  
données pour garantir leur  
disponibilité en cas de  
défaillance matérielle

Haute performance

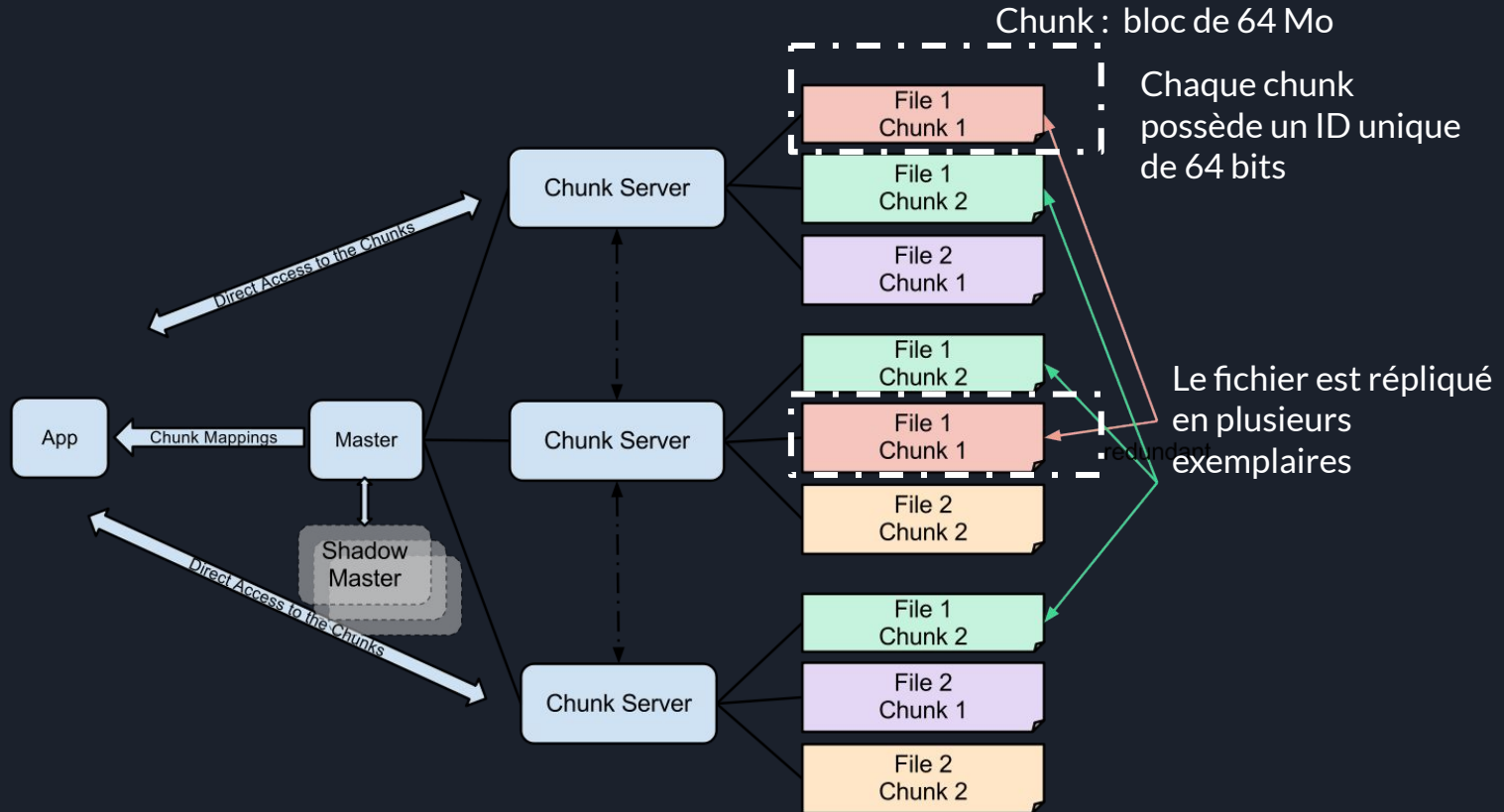
Efficacité dans la gestion  
des fichiers volumineux et  
les multiples lectures en  
parallèle

# Google File System (GFS)

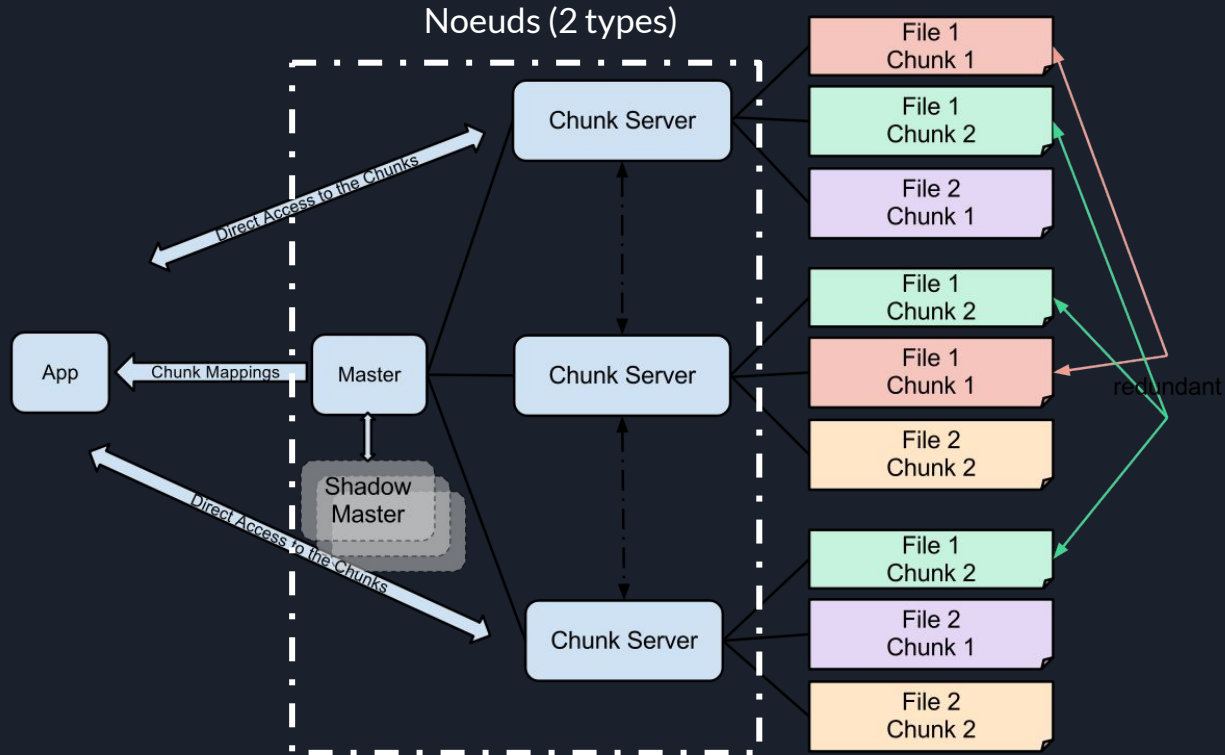




# Google File System (GFS)

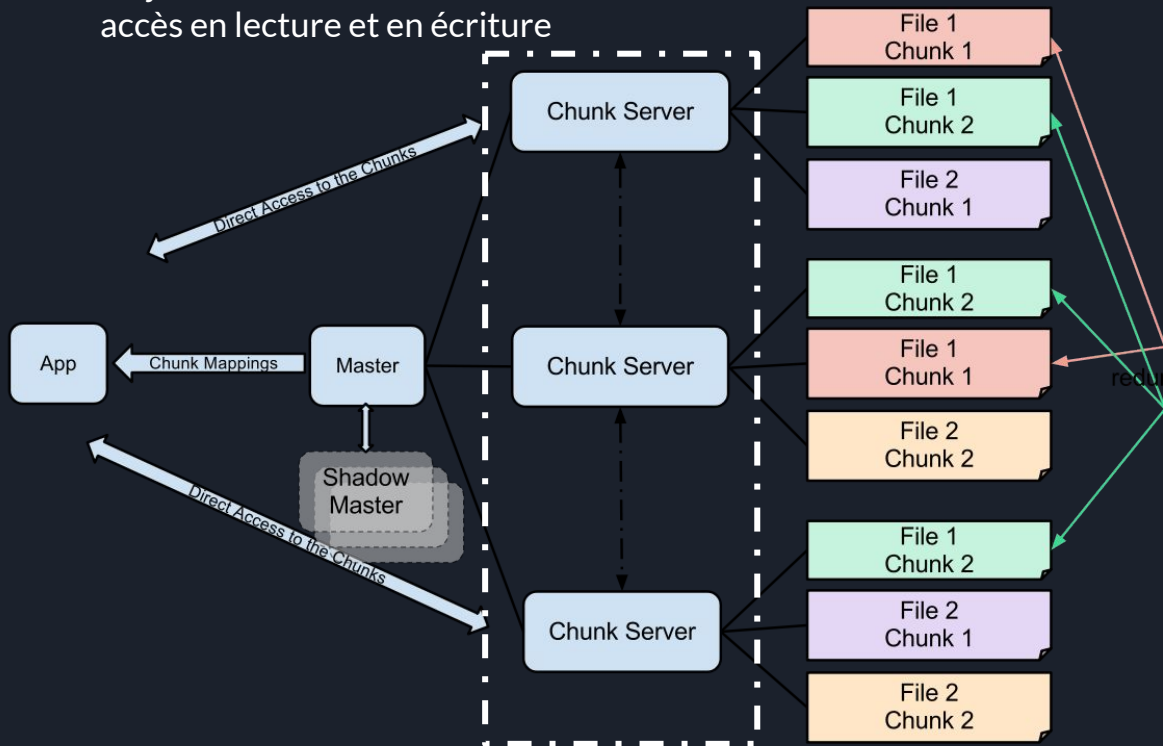


# Google File System (GFS)



# Google File System (GFS)

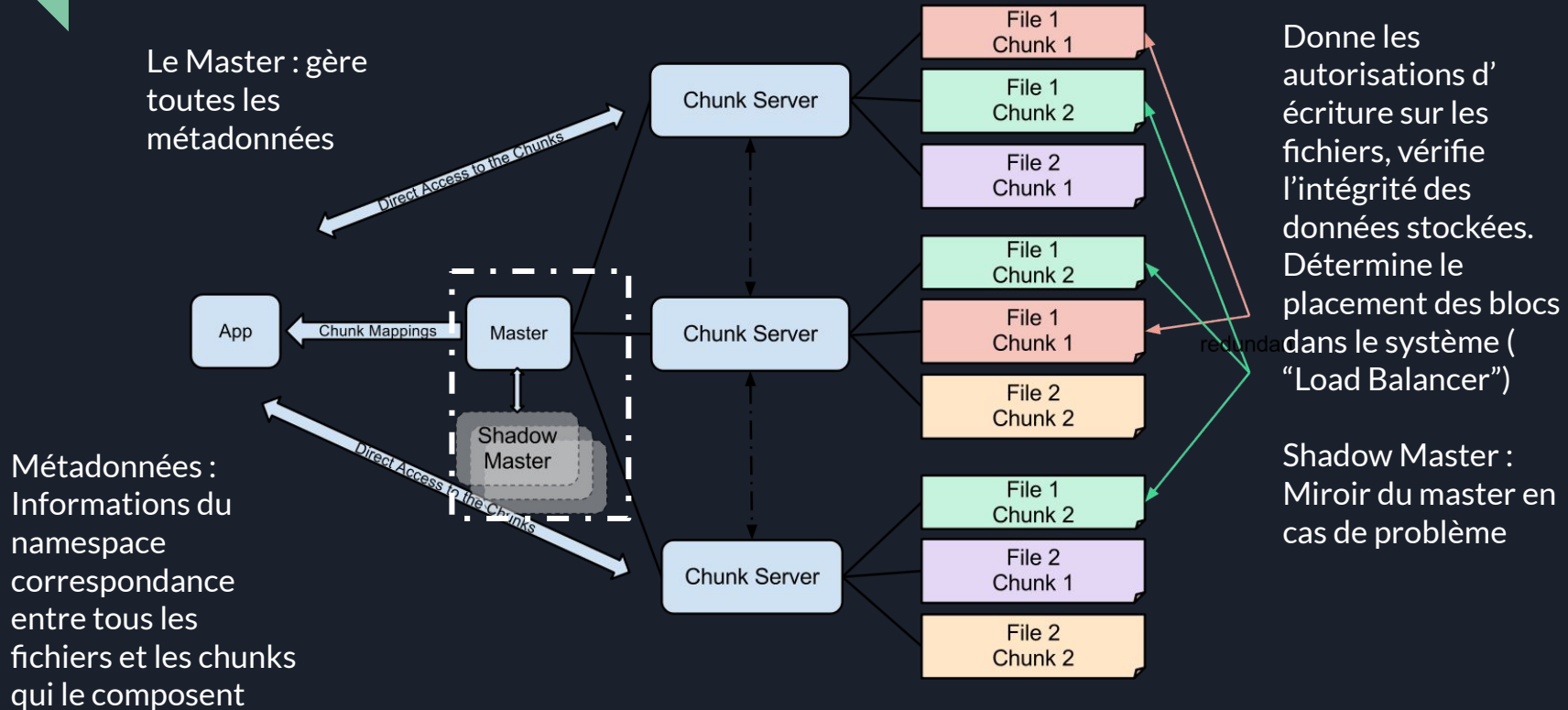
Objectif : stocker les chunks et effectuer les accès en lecture et en écriture



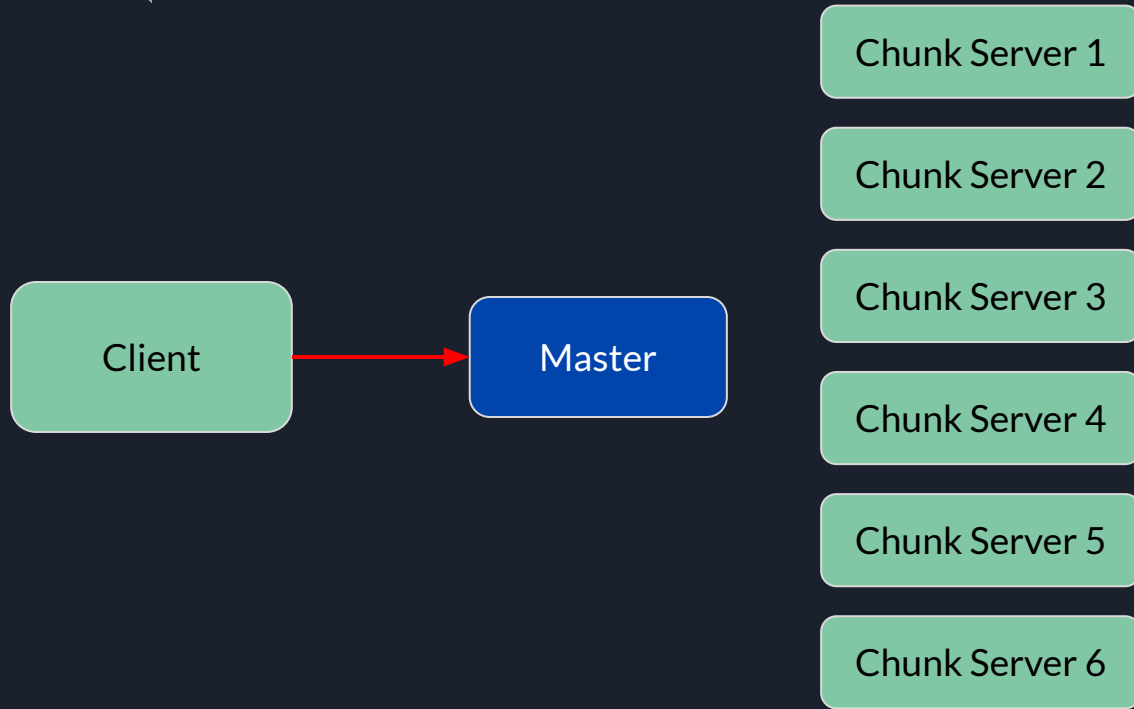
Effectue l'opération d'écriture sur le bloc considéré comme "copie primaire" puis donne l'ordre aux copies secondaires d'effectuer la même opération.

Opérations d'écriture effectuées dans le même ordre sur toutes les répliques

# Google File System (GFS)



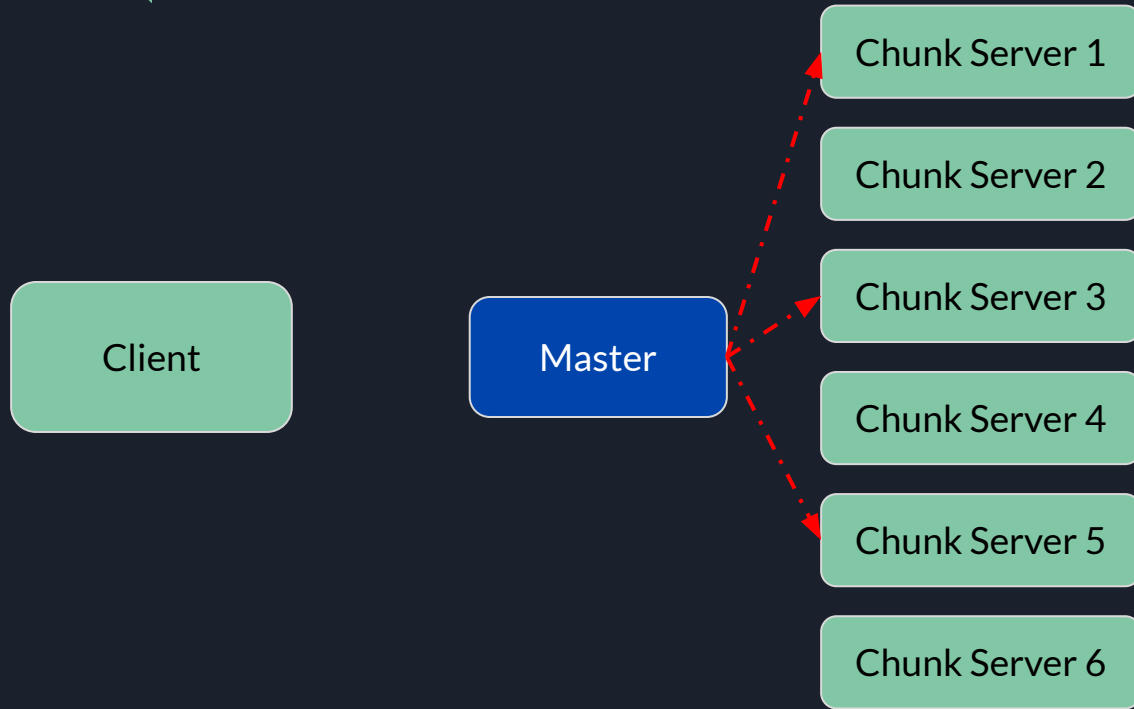
# Les opérations dans GFS



## Lecture

Pour effectuer une lecture, un client commence par demander au master l'adresse des machines possédant une copie du chunk qui l'intéresse.

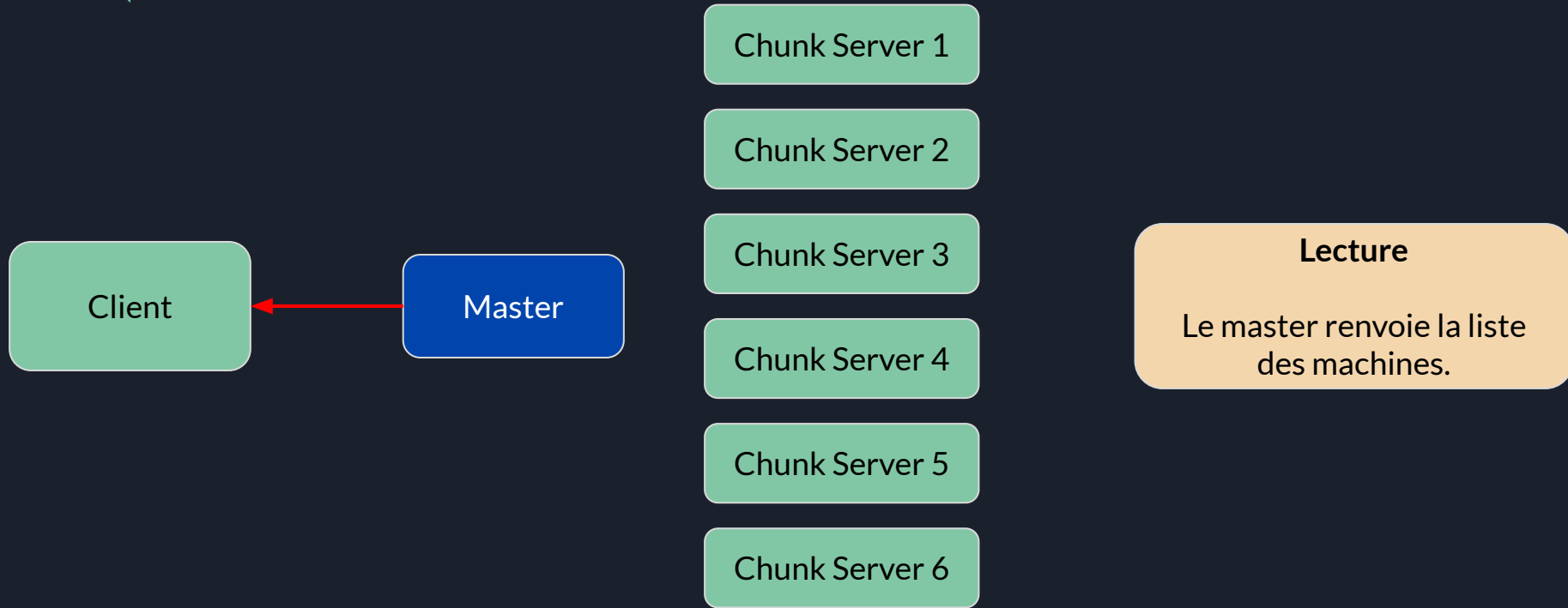
# Les opérations dans GFS



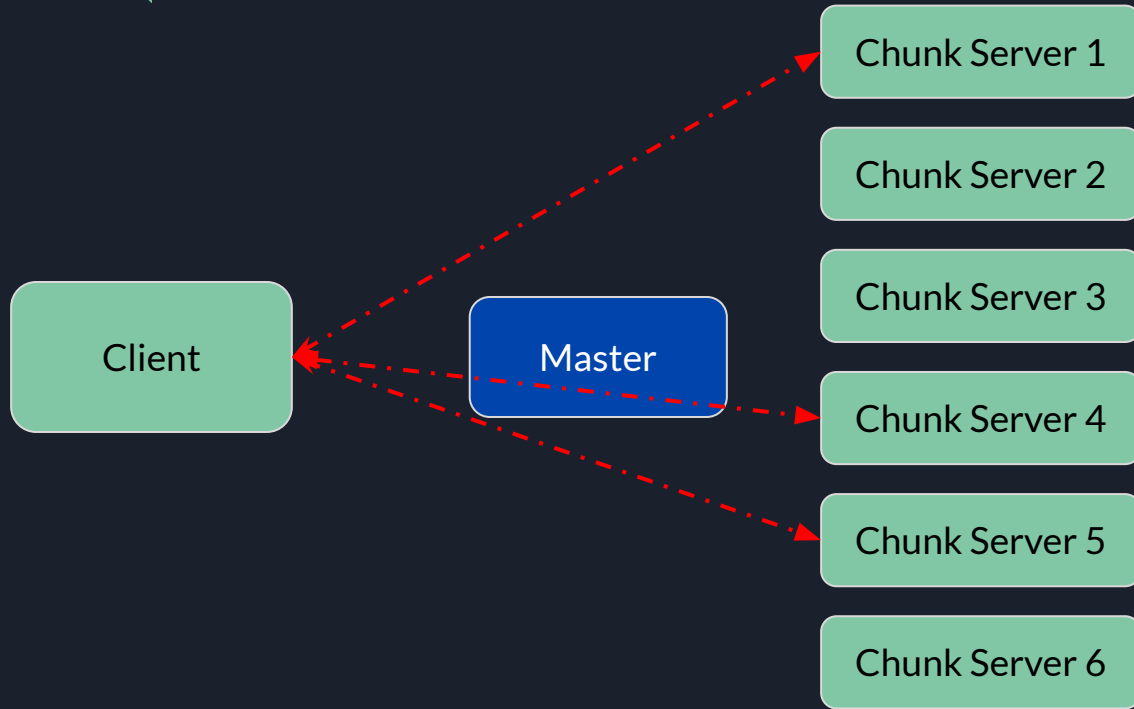
## Lecture

Le master interroge les chunkservers qui possèdent le fichier.

# Les opérations dans GFS



# Les opérations dans GFS

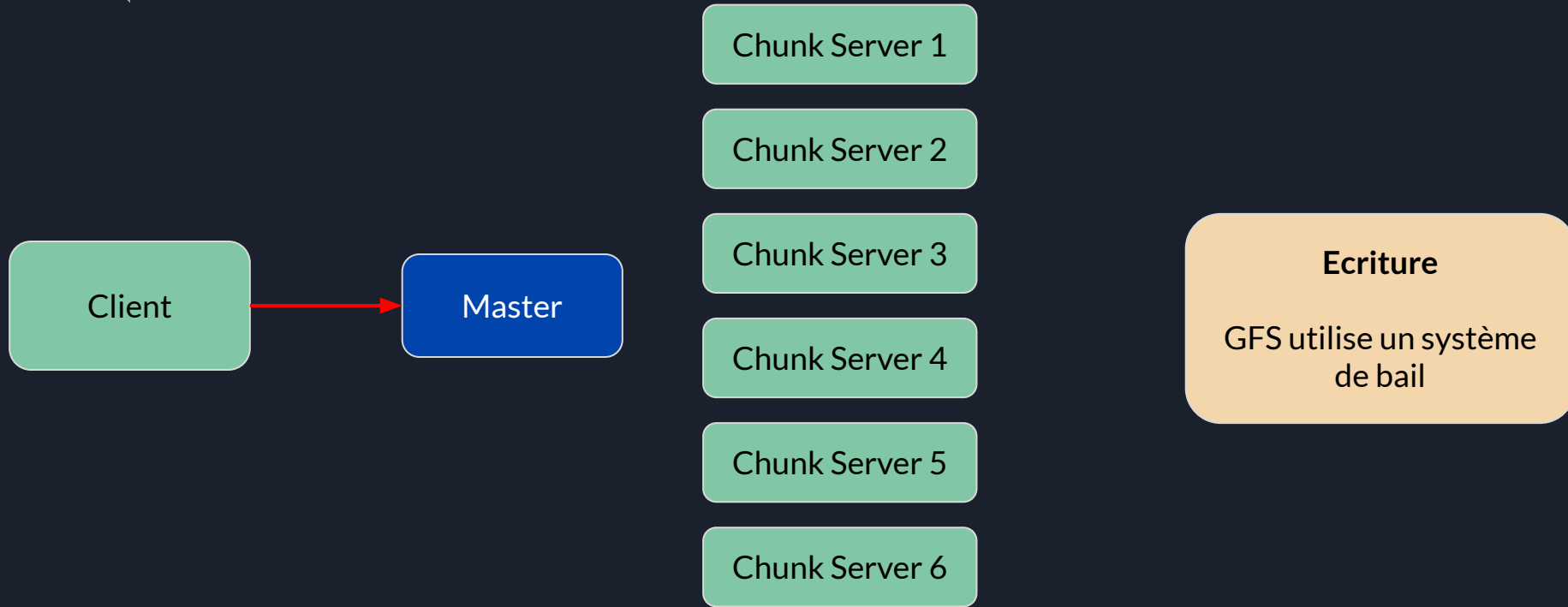


## Lecture

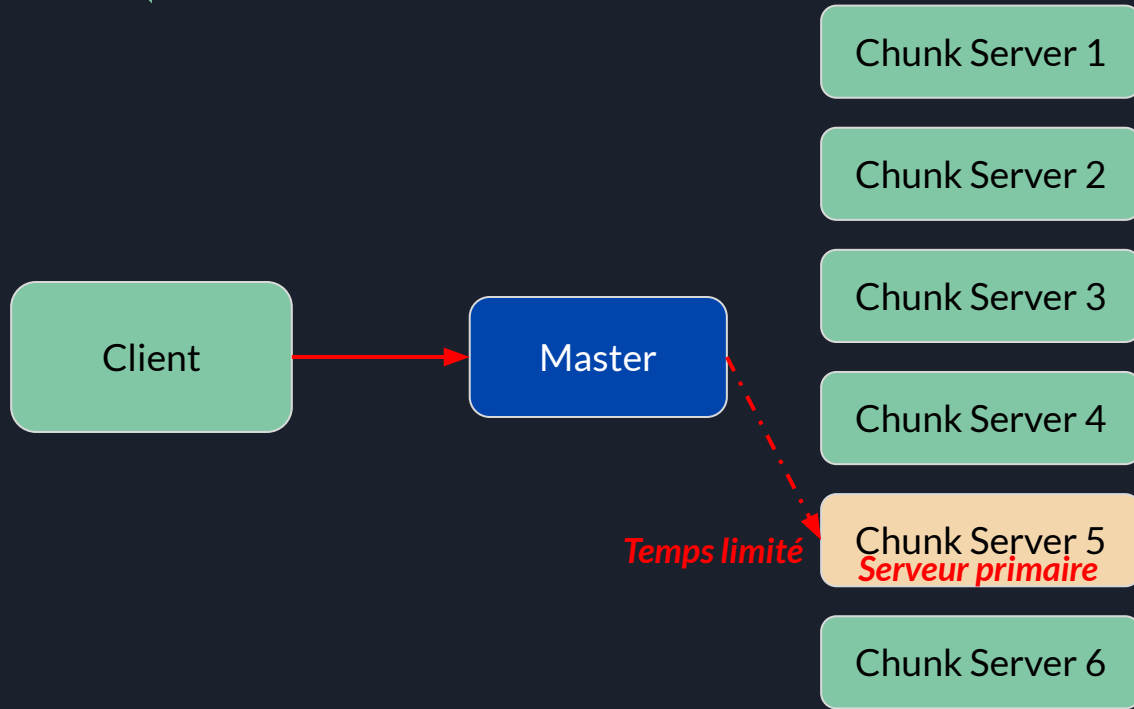
Le client s'adresse ensuite directement à l'un des chunkservers qui lui envoie les données désirées.



# Les opérations dans GFS



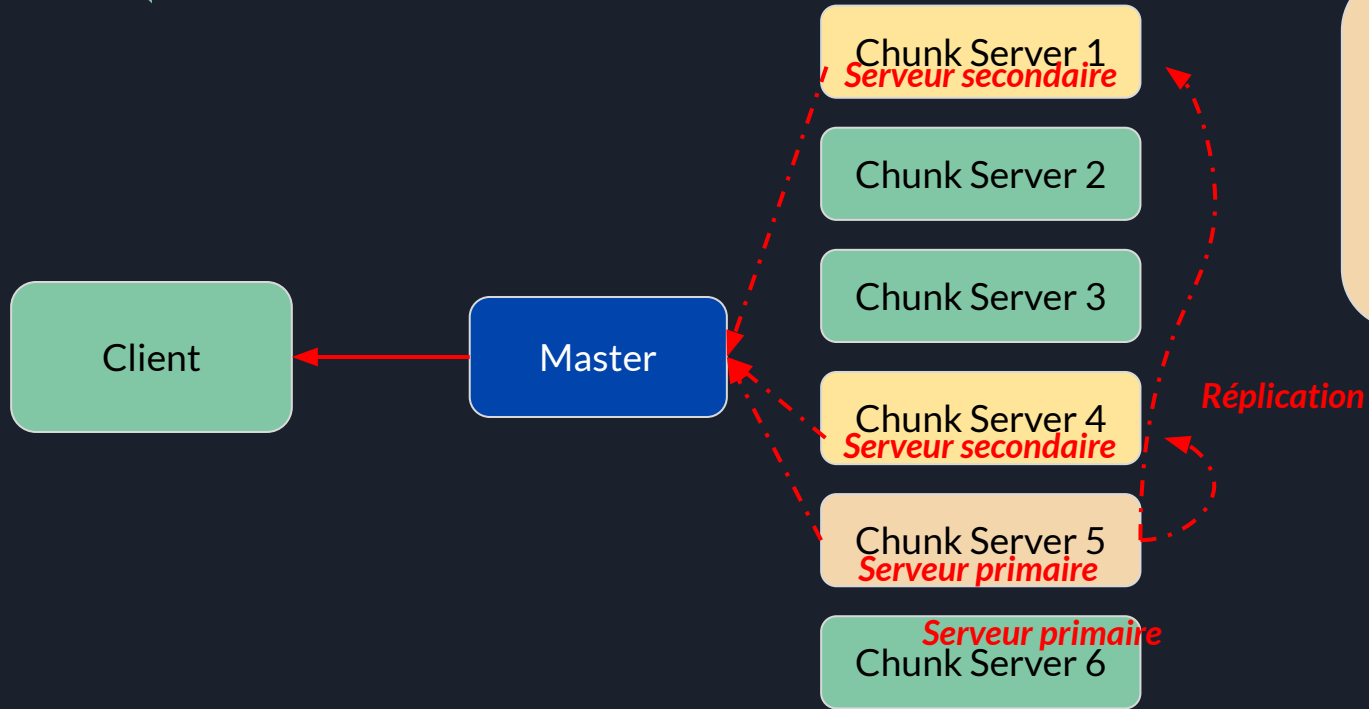
# Les opérations dans GFS



## Ecriture

Il choisit l'un des chunkservers possédant une copie du bloc et lui accorde un temps limité pour effectuer des écritures, ce serveur est alors appelé "primaire".

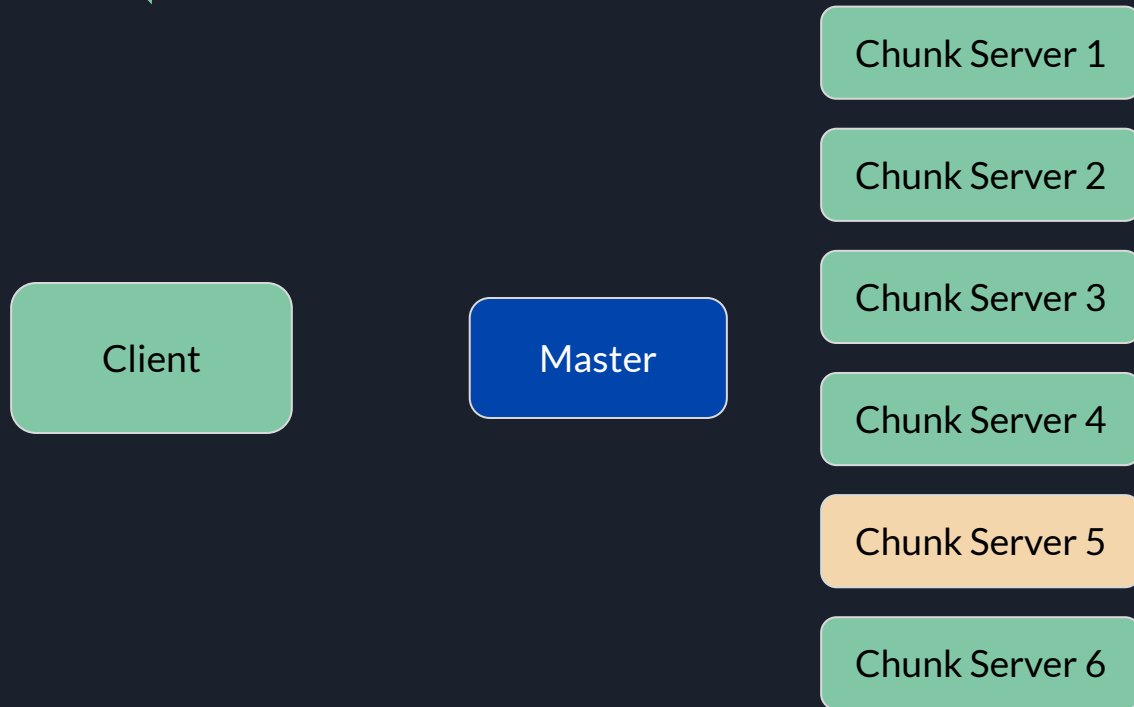
# Les opérations dans GFS



## Ecriture

Le master envoie ensuite au client l'adresse du primaire et des copies secondaires

# Les opérations dans GFS

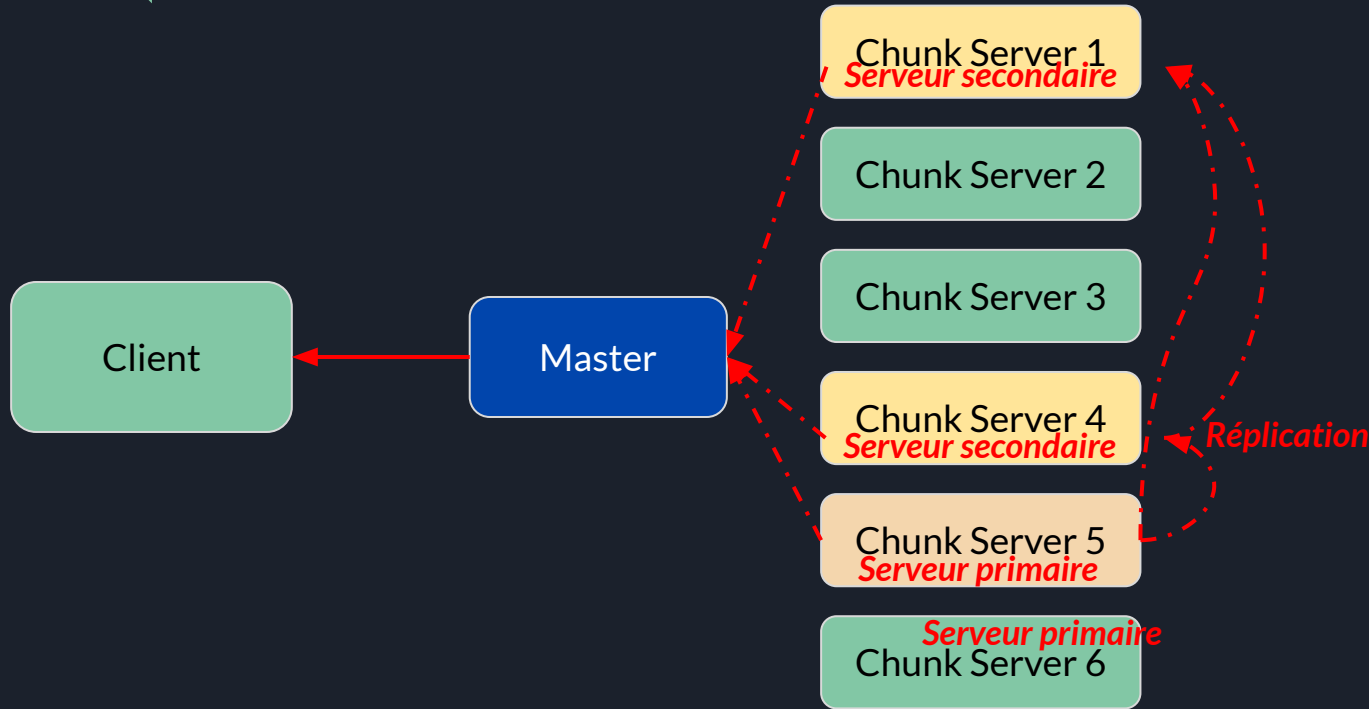


## Mise à jour des checksum

Lors d'une écriture, les clients doivent mettre à jour la checksum des blocs (64ko) concernés.

*Checksum*

# Les opérations dans GFS



## Flux de données :

Lors d'une écriture, les données sont transmises à un pipeline : le client envoie ses données au chunkserver primaire. Ce dernier tente de trouver la machine la plus proche de lui (par l'adresse IP) puis envoie l'information le réseau étant en mode "full-duplex" un serveur peut transmettre les données dès qu'il commence à les recevoir.



# HADOOP

Hadoop est un framework open source écrit en Java pour le traitement distribué de données volumineuses.





# HADOOP

Le framework d'Hadoop se décompose en 4 modules : le système de stockage HDFS (Hadoop Distributed File System) et le framework de traitement MapReduce.

HDFS  
*Hadoop Distributed File  
System*

Hadoop Map Reduce

Hadoop YARN  
*Yet Another Resource  
Negotiator*

Hadoop Common



# HDFS

HDFS  
*Hadoop Distributed File  
System*

Le noyau d'Hadoop est constitué d'une partie de stockage que l'on nomme HDFS et d'une partie de traitement appelée "Map-Reduce". Hadoop fractionne les fichiers en gros blocs et les distribue à travers les noeuds du cluster.

Pour traiter les données, il transfère le code à chaque noeud qui va traiter les données dont il dispose.



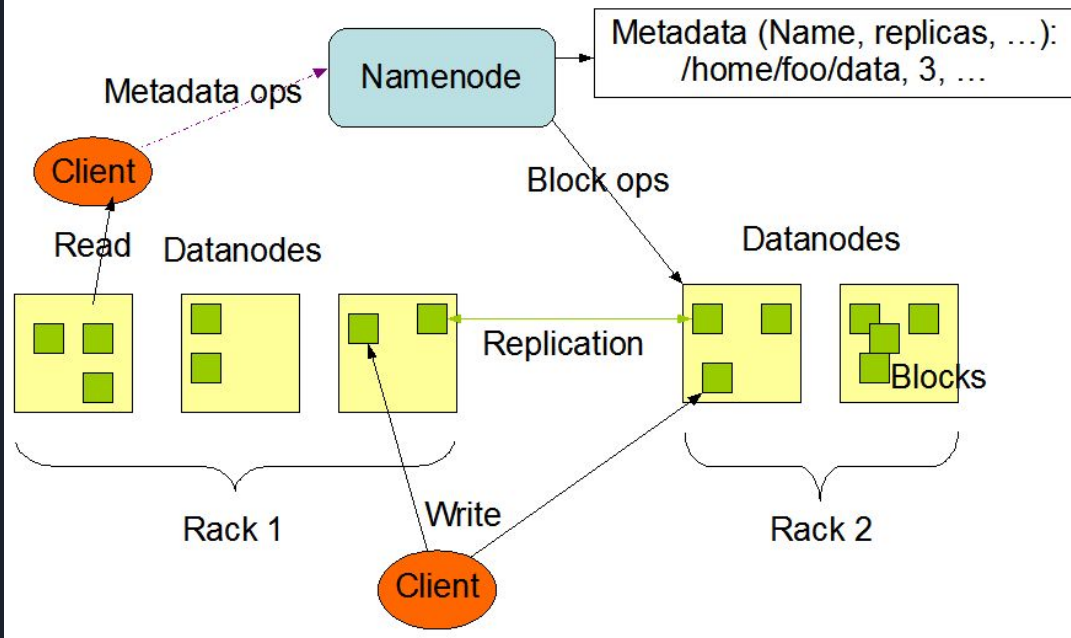
# HDFS

HDFS développé  
par Hadoop à partir  
de Google FS

NameNode

DataNode

## HDFS Architecture





# HDFS

NameNode

Le NameNode gère l'espace de noms, l'arborescence du système de fichiers et les métadonnées des fichiers et des répertoires.

Il centralise la localisation des blocs répartis dans le cluster.

Il est unique et dispose d'une instance back-up secondaire qui gère l'historique des modifications dans le système.

Ce NameNode "back-up" permet la continuité du fonctionnement du cluster en cas de panne du NameNode d'origine



# HDFS

DataNode

Le DataNode est un composant qui stocke et restitue les blocs de données. Lors du processus de lecture d'un fichier, le NameNode est interrogé pour localiser l'ensemble des blocs de données. Pour chacun d'entre eux, le NameNode renvoie l'adresse du DataNode le plus accessible, c'est à dire que le Data Node qui dispose de la plus grande bande passante.

Les DataNodes communiquent de manière périodique au NameNode la liste des blocs de données qu'ils hébergent. Si certains de ces blocs ne sont pas assez répliqués dans le cluster, l'écriture de ces blocs s'effectue en cascade par copie sur d'autres



# Apache Spark

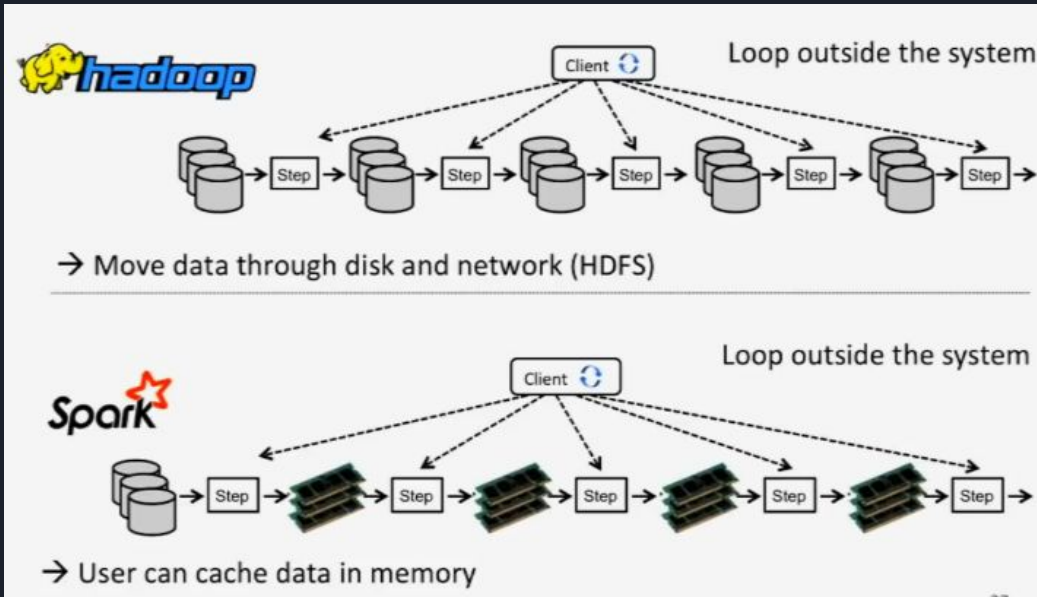
Spark est un moteur de traitement de données open source pour le traitement de données en temps réel et en batch. On l'utilise beaucoup dans le Big Data pour l'apprentissage automatique et les applications d'IA

Le moteur d'analyse de Spark traite les données 10 à 100 fois plus vite que les autres solutions. Il s'adapte en répartissant le travail de manière distribuée avec un parallélisme et une tolérance aux pannes intégrés.



# Apache Spark vs Hadoop

Il y a souvent confusion et comparaison entre Hadoop et Spark (en particulier Map-Reduce). La principale différence réside dans le fait que Spark traite et conserve les données en mémoire pour les étapes suivantes, sans écrire ni lire sur le disque. Ce qui permet d'accélérer considérablement les vitesses de traitement.





# Apache Cassandra

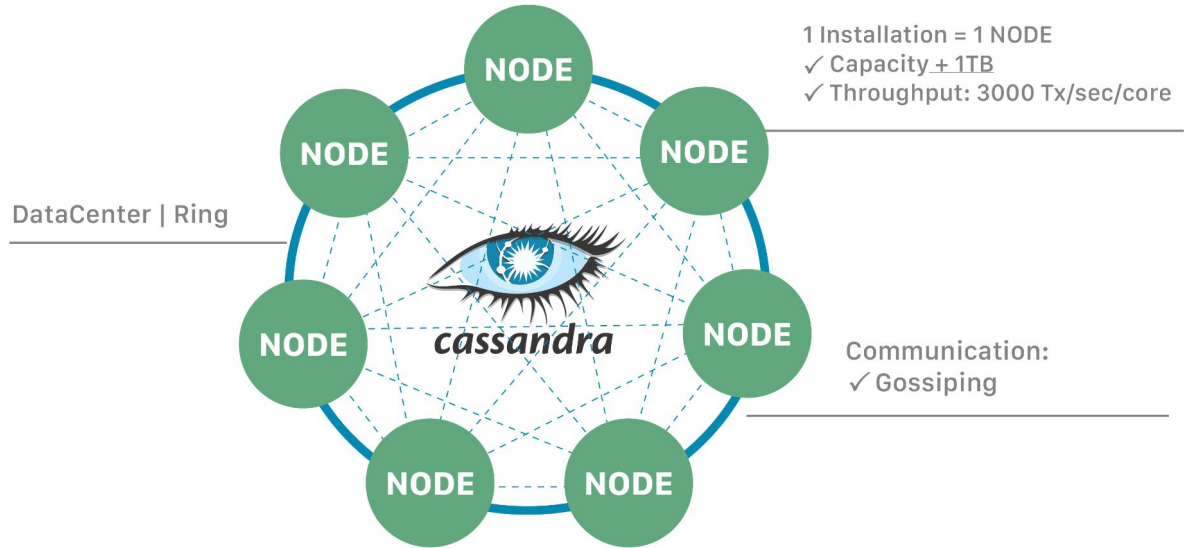
Cassandra est une base de données NoSQL distribuée conçue pour gérer de grandes quantités de données sur plusieurs serveurs.



# Apache Cassandra

Apache Cassandra est un SGBD de type NoSQL conçu pour gérer des quantités massives de données sur un grand nombre de serveurs, assurant une haute disponibilité en éliminant les SPoF (Single Point of Failure).

## ApacheCassandra™ = NoSQL Distributed Database





# Le langage de BDD pour Cassandra

Le langage de requête pour Cassandra s'appelle le CQL (Cassandra Query Language). Des implémentations (SDK) existent pour Java (JDBC), Python (DBAPI2), Node.js (Helenus) etc..

Example query 1:

```
CREATE TABLE playlists(  
  Id uuid,  
  Song_order int,  
  Song_id uuid,  
  title text,  
  album text,  
  artist text,  
  PRIMARY KEY (id, song_order));
```

Example query 2:

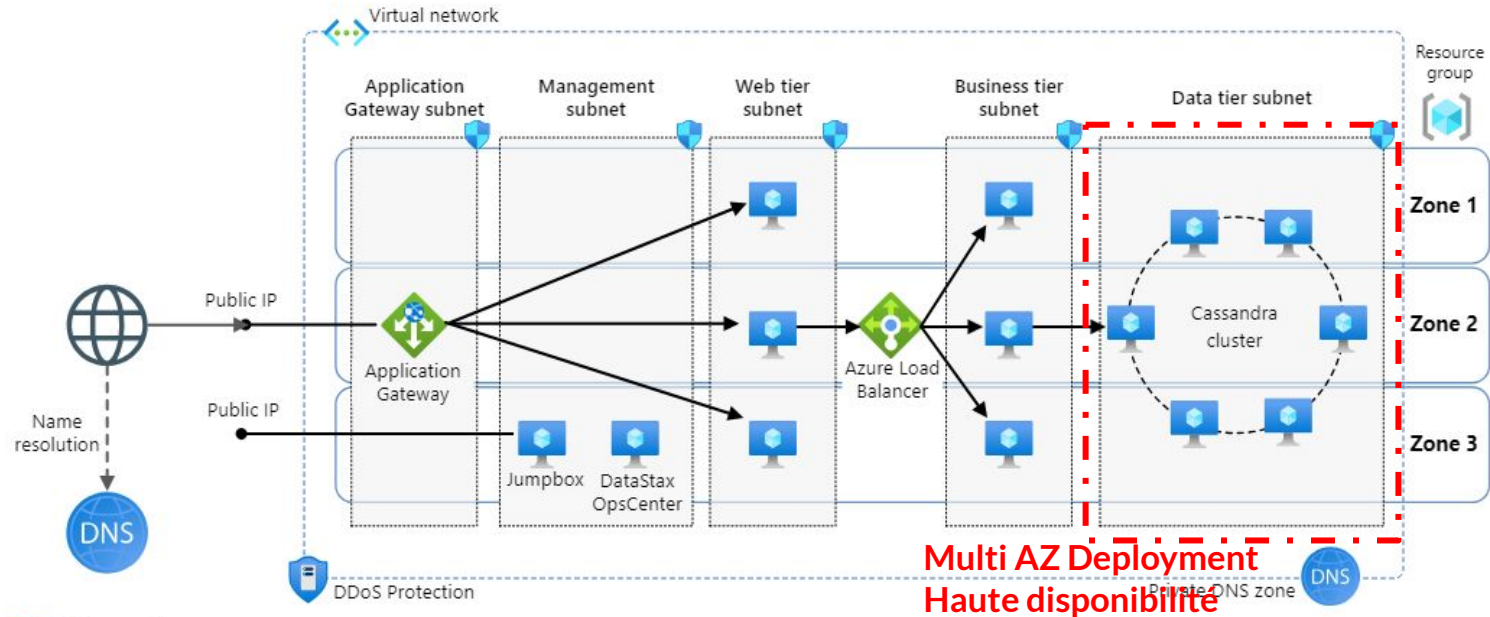
```
INSERT INTO playlist (id, song_order, song_id, title, artist, album)  
VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 4,  
       7db1a490-5878-11e2-bcfd-o800200c9a66,  
       'ojo Rojo', 'Fu Manchu', 'No One Rides for Free');
```

Example query 3:

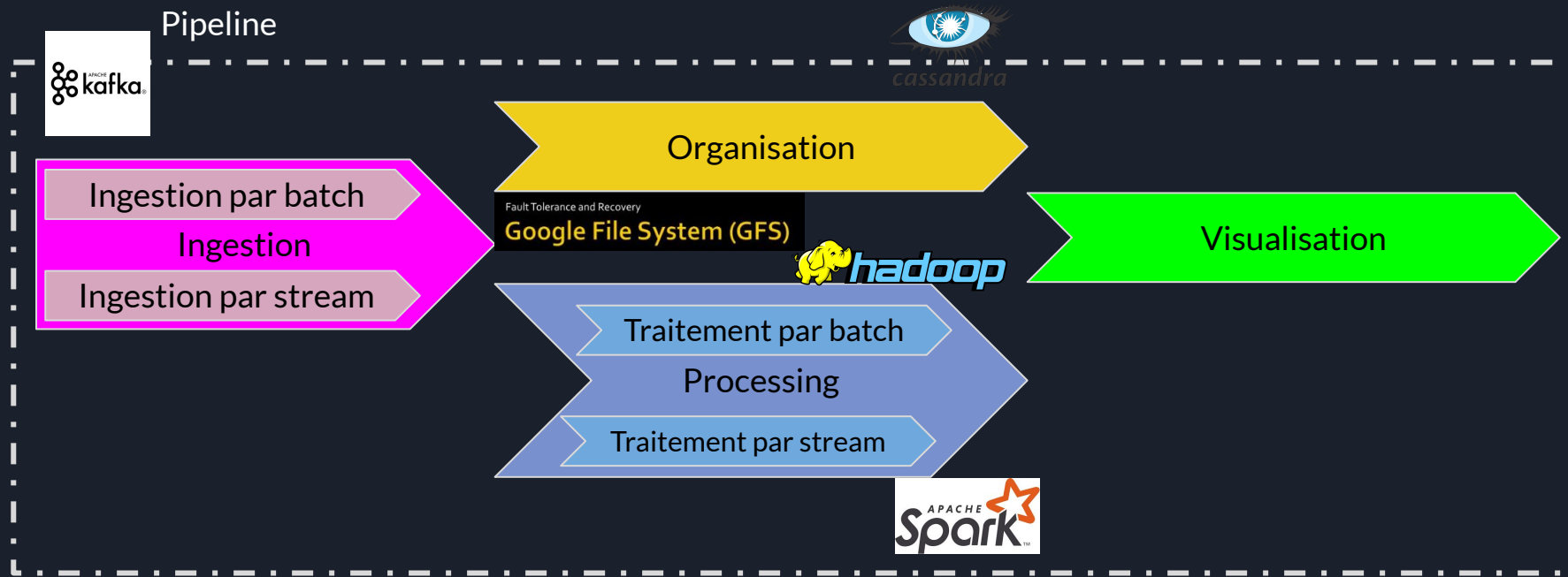
```
SELECT * FROM playlists;
```



# Apache Cassandra dans la pratique



# Pipeline outil





# Conclusion

Le Cloud Programming est un domaine en constante évolution avec de nombreuses solutions adaptées à différents besoins. Les solutions telles que MongoDB, GFS, Hadoop, DynamoDB, Spark et Cassandra sont largement utilisées dans le monde entier pour gérer et traiter efficacement les données à grande échelle dans des environnements cloud. Le choix de la solution dépendra des exigences spécifiques de votre projet et de la capacité à répondre à ces exigences.

Fin du cours

