UE BigData
FISE A3 - TAF LOGIN

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

2024-2025
Hélène Coullon

partiellement inspiré des cours de
Jacques Noyé et Guillaume Rosinosky

Introduction

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Définition ?

Accueil › Grand dictionnaire terminologique

mégadonnées

— anglais : big data

Domaines :
informatique
intelligence artificielle

Auteur :
Office québécois de la langue française

Dernière mise à jour : 2020

Définition

Ensemble d'une très grande quantité de données, structurées ou non, se présentant sous différents formats et en provenance de sources multiples, qui sont collectées, stockées, traitées et analysées dans de courts délais, et qui sont impossibles à gérer avec des outils classiques de gestion de bases de données ou de gestion de l'information.

Explosion combinatoire - légende de Sissa

Cette légende illustre **notre problème à appréhender les ordres de grandeur** et en particulier les effets exponentiels ! (c'est aussi notre problème pour la croissance et l'émission de CO2 d'ailleurs)

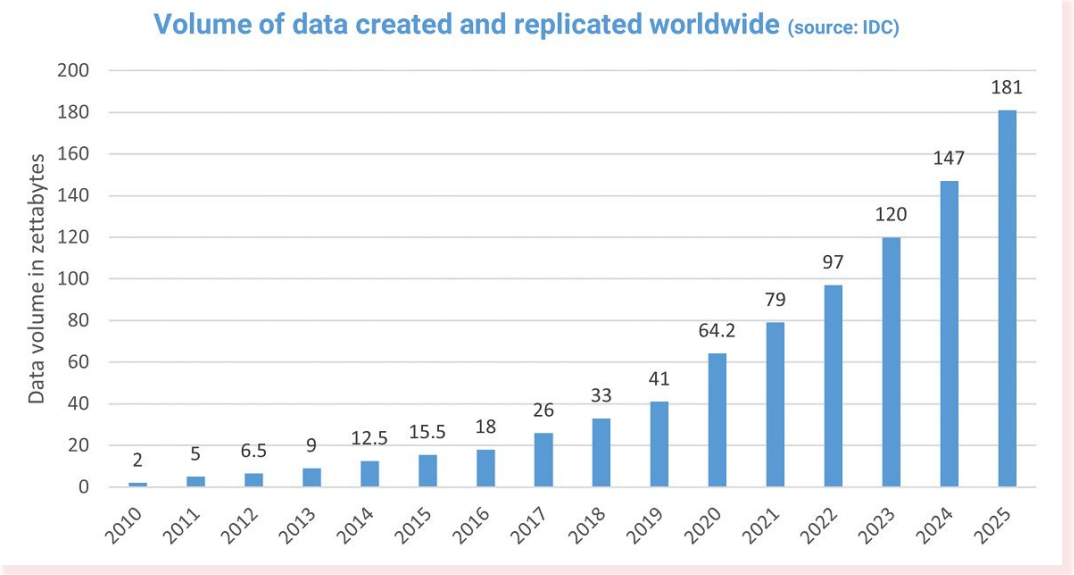
Le sage Sissa invente un jeu d'échecs pour divertir le roi. Pour remercier Sissa, le roi lui demande de choisir sa récompense. Sissa choisit de demander au roi de prendre le plateau du jeu et, sur la première case, poser un grain de riz, ensuite deux sur la deuxième, puis quatre sur la troisième, et ainsi de suite, en **doublant à chaque fois le nombre de grains de riz** que l'on met.

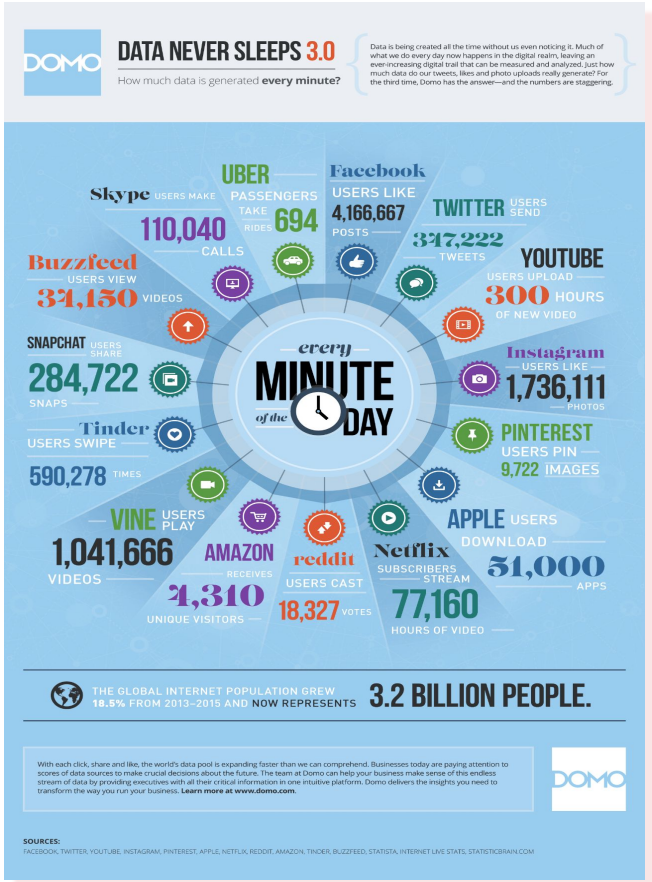
- $1 - 2^{64}$ = 18 milliards de milliards de grains
- 720 000 millions de tonnes
- 1 000 ans de production mondiale de riz actuelle

Les ordres de grandeur pour le stockage

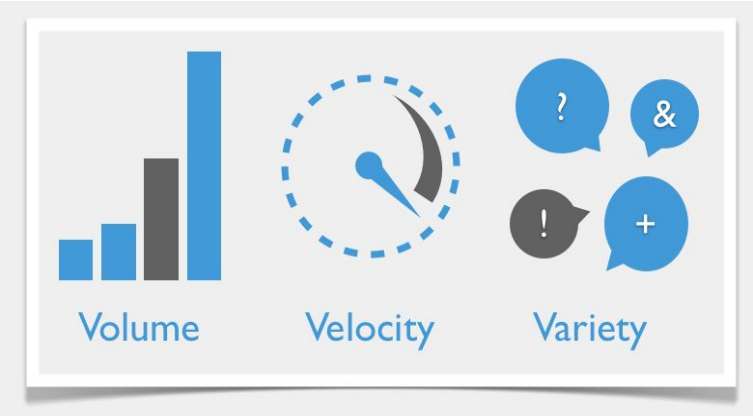
bit (b)	0 ou 1		1 mm
octet (o) / byte (B)	8 bits	1 caractère ASCII	1 m
kilooctet (ko / KB)	1000 octets, 10^3 octets	Une page de texte (2 Ko)	1 km
mégaoctet (Mo/MB)	1000 ko, 10^6 octets	Tout Shakespeare (5 Mo). Un morceau de pop (4 Mo).	10^6 m - Lille-Marseille
gigaoctet (Go/GB)	1000 Mo, 10^9 octets	Un film (1-2 Go)	10^9 m - Terre-Lune (0,38)
téraoctet (To/TB)	1000 Go, 10^{12} octets	Tous les livres de la America's Library of Congress (15 To)	10^{12} m - Terre-Saturne (1,3)
pétaoctet (Po/PB)	1000 To, 10^{15} octets	Google traite 1 Po en 1 heure	10^{15} m - année-lumière (9,5)
exaoctet (Eo/PB)	1000 Po, 10^{18} octets	10 milliards d'exemplaires de The Economist	10^{18} m - le plus proche trou noir
zettaoctet (Zo/ZB)	1000 Eo, 10^{21} octets	Information présente sur Internet en 2024: 174 Zo	10^{21} m
yottaoctet (Yo/YB)	1000 Zo, 10^{24} octets	Currently too big to imagine	10^{26} m - la taille de l'univers

Taille des données



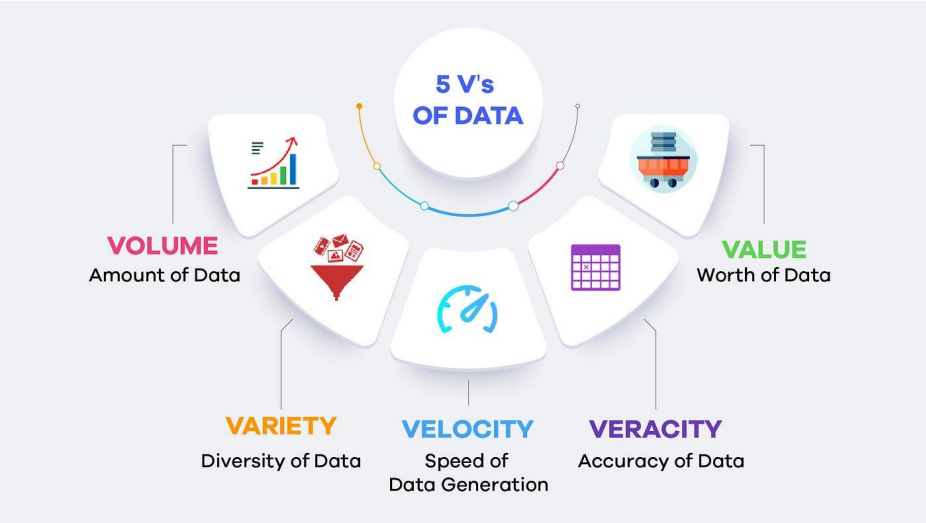


Les 3 V du BigData



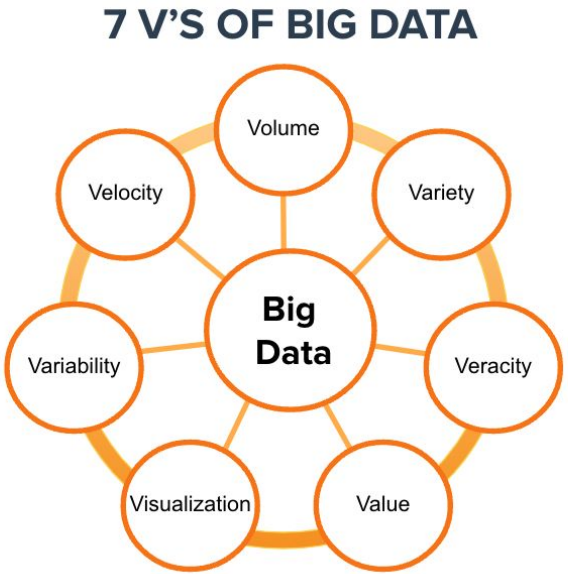
- **Volume**
  - La masse de données générées est de plus en plus grande
- **Vélocité**
  - La fréquence à laquelle sont produites les données est de plus en plus grande
- **Variété**
  - une grande variété de données collectées

Les 5 V du BigData



- Volume
- Vélocité
- Variété
- **Véracité**
  - Lié à la qualité de l'information, à son intégrité, à la fiabilité de la source
- **Valeur**
  - Les données brutes ont souvent peu d'intérêt, il faut réussir à créer de la valeur à partir des données

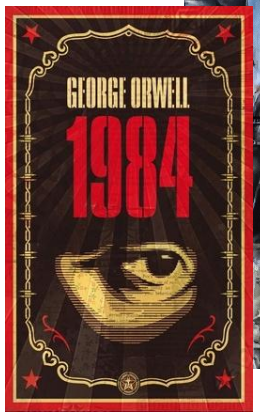
Les 7 V du BigData

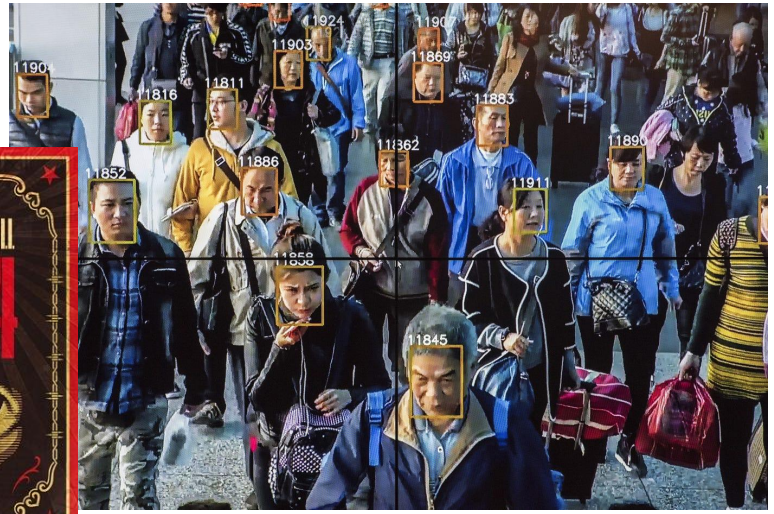


- Volume
- Vélocité
- Variété
- Véracité
- Valeur
- **Visualisation**
  - Rendre l'information exploitable par le plus grand nombre
- **Variabilité**
  - Nature changeante de la donnée dont le format ou la valeur peut varier avec le temps

Les côtés obscurs

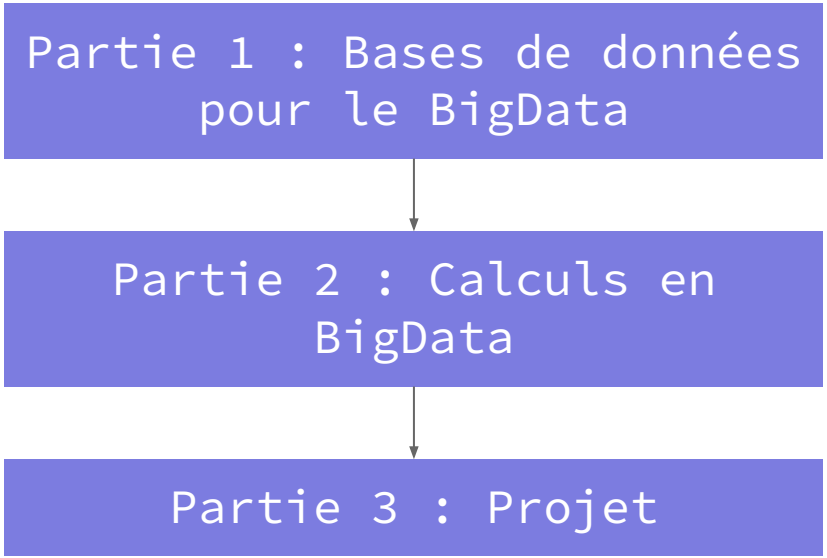
- Consommation de ressources et production de gaz à effet de serre
- Appropriation de données personnelles
- Banditisme
- Espionnage
- Manipulation de l’opinion
- Contrôle social
- etc.





Déroulement de l'UE

- Par ½ journée
  - cours
  - TP
  - Tutoriel noté
- Projet
- Evaluation
  - quizz et tuto - CG2
  - projet - CG4
  - projet et tuto - CG14



Contenu de l'UE

- **[2h30] 11/02**
  - Introduction, installation outils, et rappels SQL (avec MySQL)
- **[3h45] 18/02 matin**
  - Cours NoSQL
  - TP JSON et MongoDB
  - Travail en groupe NoSQL
- **[2h30] 18/02 aprem**
  - Travail en groupe NoSQL
- **[3h45] 25/02**
  - Parallélisme et Map-Reduce
  - Spark et Spark SQL
- **[3h45] 04/03 matin**
  - Spark Streaming et Kafka
- **[2h30] 04/03 aprem**
  - NoSQL Neo4J (orienté graphes) - **Thomas Hassan / Orange Labs**
- **[1h45] 11/03 matin Quizz et Oraux tutos NoSQL**
- **Projets par 4**
  - **[2h+3h45] 11/03**
  - **[3h45+2h30] 18/03**
  - **[3h45] 25/03**
- **Oraux projets le 25/03 après-midi**

Rappels SQL

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Les BD relationnelles et SQL

Les BD relationnelles :

- Le modèle relationnel apparaît en 1970. Il est dû à Edgar F. Codd.
- Il s'appuie sur le concept mathématique d'algèbre relationnelle

SQL : IBM développe un prototype de SGBD relationnel et son langage SQUARE (Specifying Queries in A Relational Environment) qui devient SEQUEL (Structured English Query Language) puis SQL suite à conflit juridique sur le nom, par la suite interprété Structured Query Language.

Exemples de requêtes SQL

Cours et exemples ici

SGBD relationnel

Système de gestion de bases de données

- Définition de la structure ou du schéma de données
- Définition et mise à jour des données
- Accès aux données (ou requêtes)
- Administration (utilisateurs, sécurité, intégrité, concurrence, reprise sur faute, performances etc.)

Quelques exemples de SGBD relationnels :

- SQL server (microsoft - propriétaire)
- Oracle database (Oracle corporation - propriétaire)
- MySQL (Oracle corporation - ouvert)
- PostgreSQL (projet collaboratif)

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP1 et TP2

- Installation et premiers tests MySQL
- Revoir les bases de SQL avec MySQL

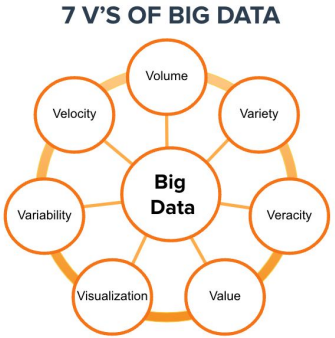
Contenu de l'UE

- **[2h30] 11/02**
  - Introduction, installation outils, et rappels SQL (avec MySQL)
- **[3h45] 18/02 matin**
  - Cours NoSQL
  - TP JSON et MongoDB
  - **Travail en groupe NoSQL**
- **[2h30] 18/02 aprem**
  - **Travail en groupe NoSQL**
- **[3h45] 25/02**
  - Parallélisme et Map-Reduce
  - Spark et Spark SQL
- **[3h45] 04/03 matin**
  - Spark Streaming et Kafka
- **[2h30] 04/03 aprem**
  - NoSQL Neo4J (orienté graphes) - **Thomas Hassan / Orange Labs**
- **[1h45] 11/03 matin Quizz et Oraux tutos NoSQL**
- **Projets par 4**
  - **[2h+3h45] 11/03**
  - **[3h45+2h30] 18/03**
  - **[3h45] 25/03**
- **Oraux projets le 25/03 après-midi**

Bases de données NoSQL

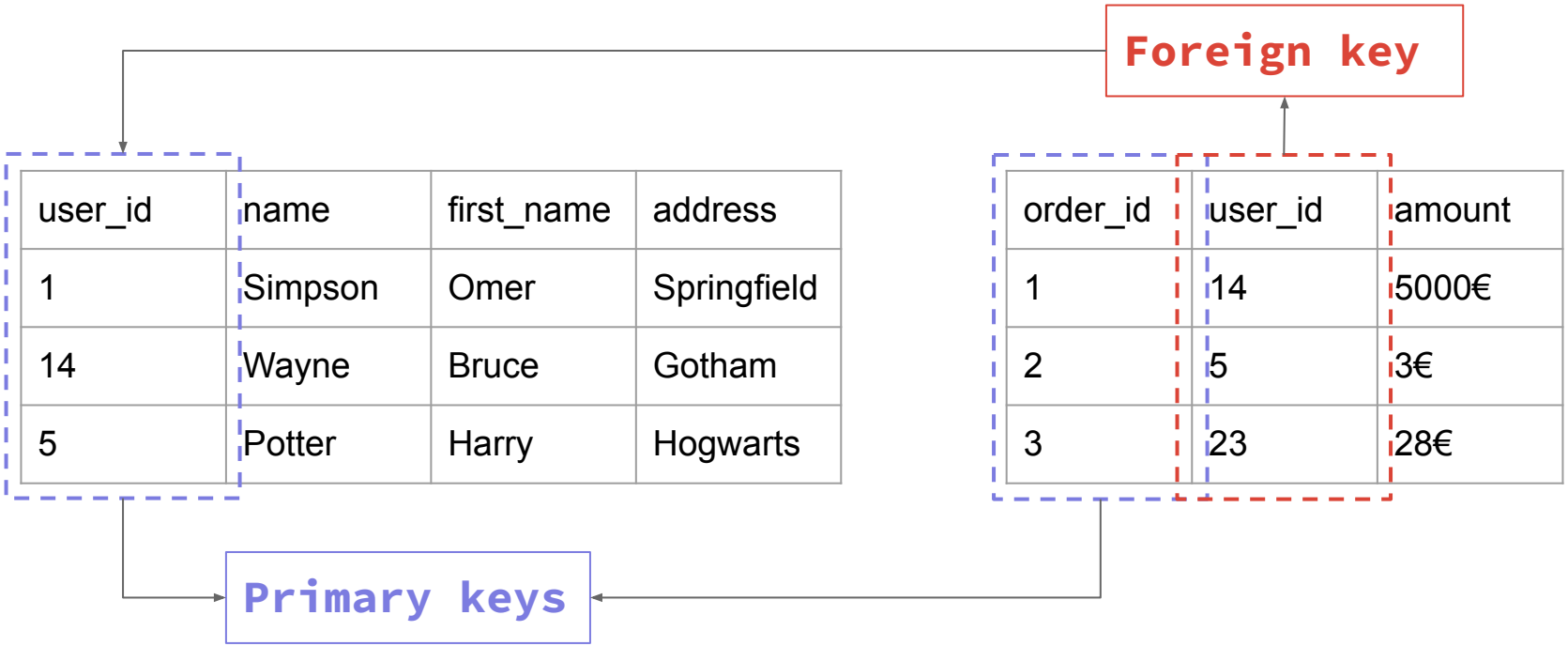
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Caractéristiques des données du BigData



- volume
  - il faut pouvoir **partitionner** les données
  - il faut **éviter les mécanismes qui ralentissent les requêtes**
  - il faut pouvoir faire des calculs parallèles
- variété
  - les données peuvent être **structurées semie- ou non-structurées**
  - la **nature** des données est très **variée**
- variabilité
  - les **schémas sont un frein** si les données évoluent dans le temps
- vélocité
  - on ne peut pas tout stocker, il faut traiter à la volée

Résumé BD relationnelles



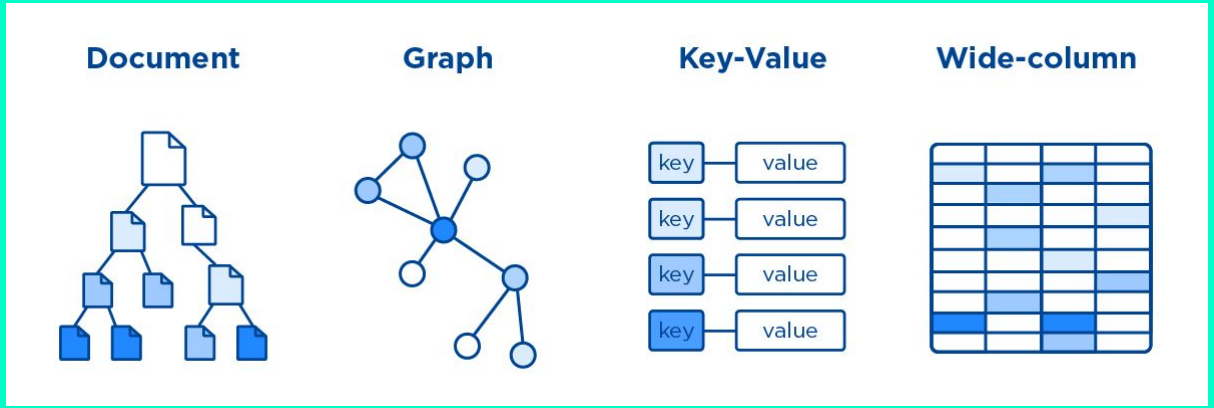
Données structurées / Données qui doivent valider le schéma

Not Only SQL (NoSQL)

Utilisent des modèles de données dont la structure est différente de celle du modèle relationnel en table, lignes, et colonnes

- Première apparition du terme en 1998
  - Carl Strozz - bases légères et open source
- Popularisé par les GAFAM
  - 2000 Neo4J
  - 2004 couchDB
  - 2008 Cassandra
- Adaptées au BigData pour différentes raisons…

4 TYPES PRINCIPAUX



Orientées Colonnes

- Chaque colonne est traitée séparément, et les valeurs sont stockées de façon contigüe
- Hautes performances pour les requêtes d'agrégation comme SUM, COUNT, AVG et MIN

order_id	user_id	amount
1	14	5000€
2	5	3€
3	23	28€
…		
2Billions		6859€

SUM

Orientées documents

- Ensemble de collections contenant des documents
- Un document est typiquement un objet JSON associé à une clé unique
- CRUD sur des documents JSON

{“id”: “iuhd768”, “type”: “mobile”, “name”: “iPhone”, “version”: 15, …}

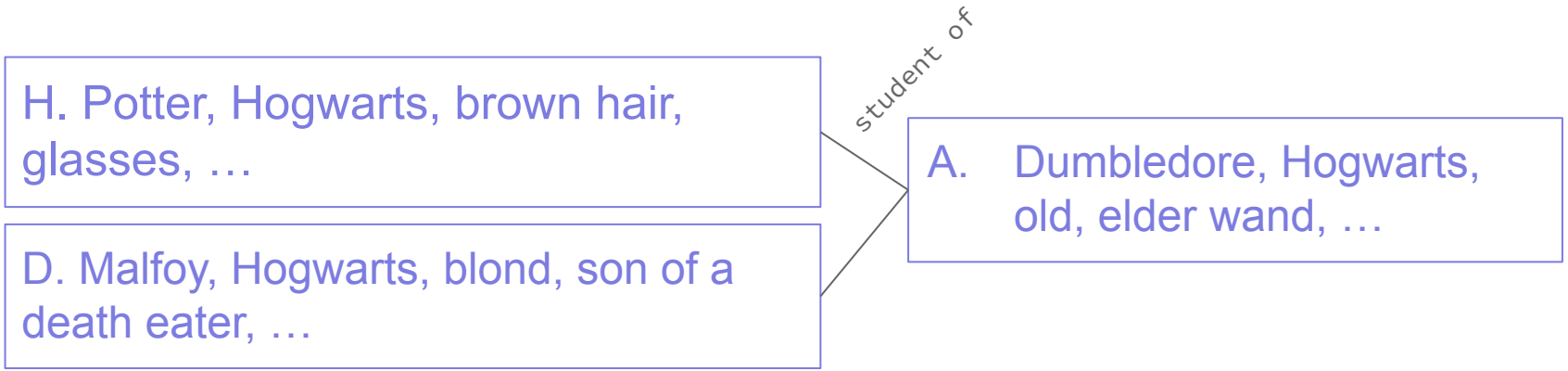
{“id”: “leo8”, “type”: “camera”, “name”: “Sony aR7”, “optics”: “mirrorless”, …}

{“id”: “po65h”, “type”: “DVD”, “name”: “Harry Potter and the goblet of fire”, “year”: “2005”, …}

Object storage est similaire mais
les objets peuvent être non-structurés

Orientées Graphes

- L’entité est stockée sous forme de noeud, et les relations comme arêtes
- Facilite la visualisation des relations entre les noeuds
- On l’utilise principalement pour les réseaux sociaux, la logistique, etc.



Orientées clés-valeurs

- Les données sont stockées sous forme de paires clé / valeur (~table de hachage)
- Permet la prise en charge de larges volumes de données
- Les données sont entreposées dans un tableau de “hash” au sein duquel chaque clé est unique
- Stocker facilement des données sans schéma
- On récupère la valeur entière, pas de requêtes complexes

“dummy”	{“test”: “ok”, “nothing”: True}
786	100111001011
“oih78zz”	42

Autres types

Séries temporelles

- Bases faites pour les événements ou mesures enregistrés dans le temps et associés à un timestamp
- Exemples : monitoring, données de capteurs IoT, enregistrement de clics, transactions financières etc.

timestamp	city (tag key)	country (tag key)	field	field value
2022-01-01T12:00:00Z	London	UK	temperature	12.1
2022-02-01T12:00:00Z	London	UK	temperature	12.5
2022-04-01T12:00:00Z	London	UK	temperature	5.4

Orientées recherche

- Requêtes pour faire de la recherche d'information dans des objets/documents JSON (orienté Read, pas Update)
- Données semi-structurées
- Optimisé pour la recherche d'information

{"id": "iuhd768", "type": "mobile", "name": "iPhone", "version": 15, …}

{"id": "dkhb789", "type": "mobile", "name": "Sony aR7", "optics": "mirrorless", …}

{"id": "po65h", "type": "DVD", "name": …}

type	ids
"mobile"	["iuhd768","dkhb789"]

inverted index

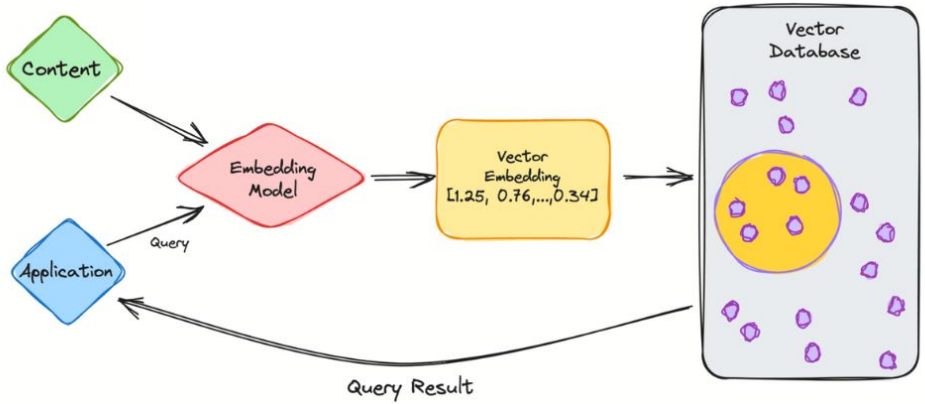
Vectorielles

- Recherche de similarités dans des données non-structurées
- Calcul d'un vecteur représentant la donnée pour comparer des données de nature différentes par des distances
- Indexation des données pour accélérer les requêtes
- Retrieval Augmented Generation (RAG) : LLM utilise des bases vectorielles pour dynamiquement enrichir sa génération

Milvus

chroma

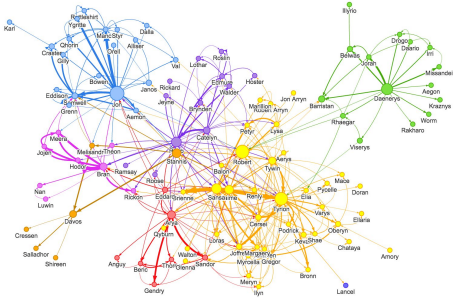
Pinecone



Visualisation de la donnée

- Prometheus, Grafana pour les time-series
- Elastic Kibana
- Neo4j Browser, Neo4J Bloom, Neovis.js
- …

Mais il faudra découvrir par vous même, peut être dans votre tutoriel ou votre projet ?



TP Noté
Conception d'un tutoriel NoSQL

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

6 sujets :

- REDIS (key-value store)
- ClickHouse (columnar)
- InfluxDB (time series)
- Elasticsearch (search)
- Milvus (Vector)
- Sharding with MongoDB

Inscriptions ici

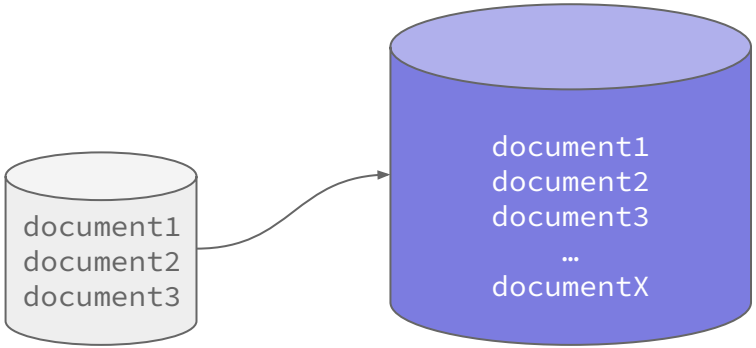
Rendus :

- tutoriel conteneurisé
- repo GitHub et README du tuto
- utilisation d'un dataset dispo en ligne de votre choix
- visualisation des données (si vous avez le temps)

Scalabilité distribution

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Scalabilité verticale - gérer plus de données sur 1 serveur

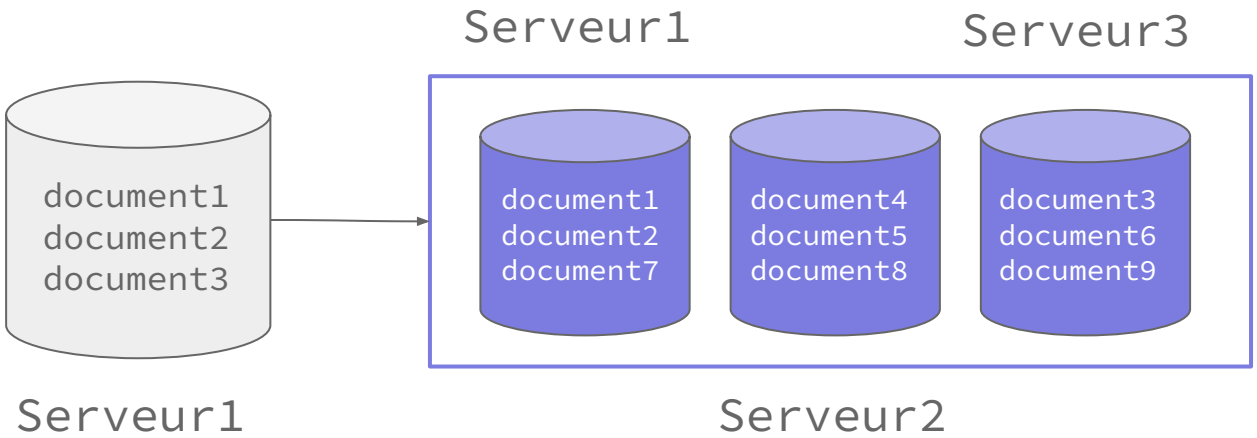


\+ de CPU
\+ de RAM
\+ de disque

Scalabilité horizontale - Plus de serveurs

Plus de serveurs pour

- stocker les données -> il faut **partitionner** !
- répondre aux requêtes



Partitionnement vertical

- Répartir les colonnes d'une base de données sur les différents serveurs
- S'applique plutôt aux BD relationnelles normalisées
- Problème : les jointures demandent des échanges de messages entre les serveurs

Serveur1

user_id	name	first_name	address
1	Simpson	Omer	Springfield
14	Wayne	Bruce	Gotham
5	Potter	Harry	Hogwarts

Serveur2

order_id	user_id	amount
1	14	5000€
2	5	3€
3	23	28€

Partitionnement horizontal - Sharding

- Bien sélectionner la shard key (pour bien répartir la charge)
  - range-based sharding : [A-I]->S1,[J-R]->S2,[S-Z]->S3
  - hask-based sharding : hash1->S1,hash2->S2,hash3->S3
  - directory-based sharding : red->S1,blue->S2,green->S3
  - geographical-based sharding …
- Chaque type de base fait du sharding d'une certaine façon pour optimiser ses requêtes
- MySQL propose aussi du sharding maintenant !

Shard1

user_id	name	first_name	address
1	Simpson	Omer	Springfield
14	Wayne	Bruce	Gotham

Shard2

user_id	name	first_name	address
5	Potter	Harry	Hogwarts

DÉNORMALISATION

Un modèle de données **dénormalisé** embarque **toutes les données associées dans un seul document** au lieu de répartir l'information dans différentes collections et documents.

(Concerne principalement les BD NoSQL qui gère des documents)

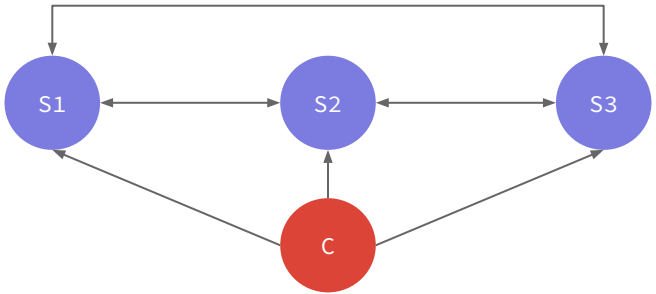
Objectifs :

- éviter au maximum les jointures qui ralentissent les requêtes surtout quand on a de très gros volumes de données
- faciliter le partitionnement horizontal (sharding)

Une **réplication** de l'information est parfois nécessaire entre différents documents et alors il faut assurer la **cohérence** !

(A noter qu'il y a quand même un "join" dans MongoDB)

BD distribuées



- plusieurs noeuds/serveurs stockent et gèrent les réponses aux requêtes
- plusieurs instances de bases de données qui collaborent
- gestion de la réplication (ne pas perdre de données)
- gestion de la consistance des données
- tolérance aux pannes ou au partitionnement

Oui… mais pas tout à à la fois !

Problème CAP

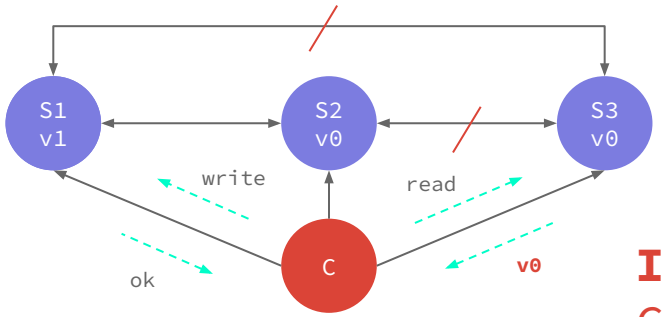
3 propriétés souhaitables sur les bases de données distribuées :

- La **cohérence** (consistency) signifie que la donnée est la même sur tous les noeuds (version la plus récente)
- La **disponibilité** (availability) signifie que toute requête obtient une réponse, même si un ou plusieurs nœuds sont en panne
- La **tolérance au partitionnement réseau** signifie que le cluster doit continuer à fonctionner malgré un nombre quelconque de pannes de communication entre les nœuds du système

Seulement 2 de ces 3 propriétés peuvent être vérifiées à la fois

Intuition de la preuve

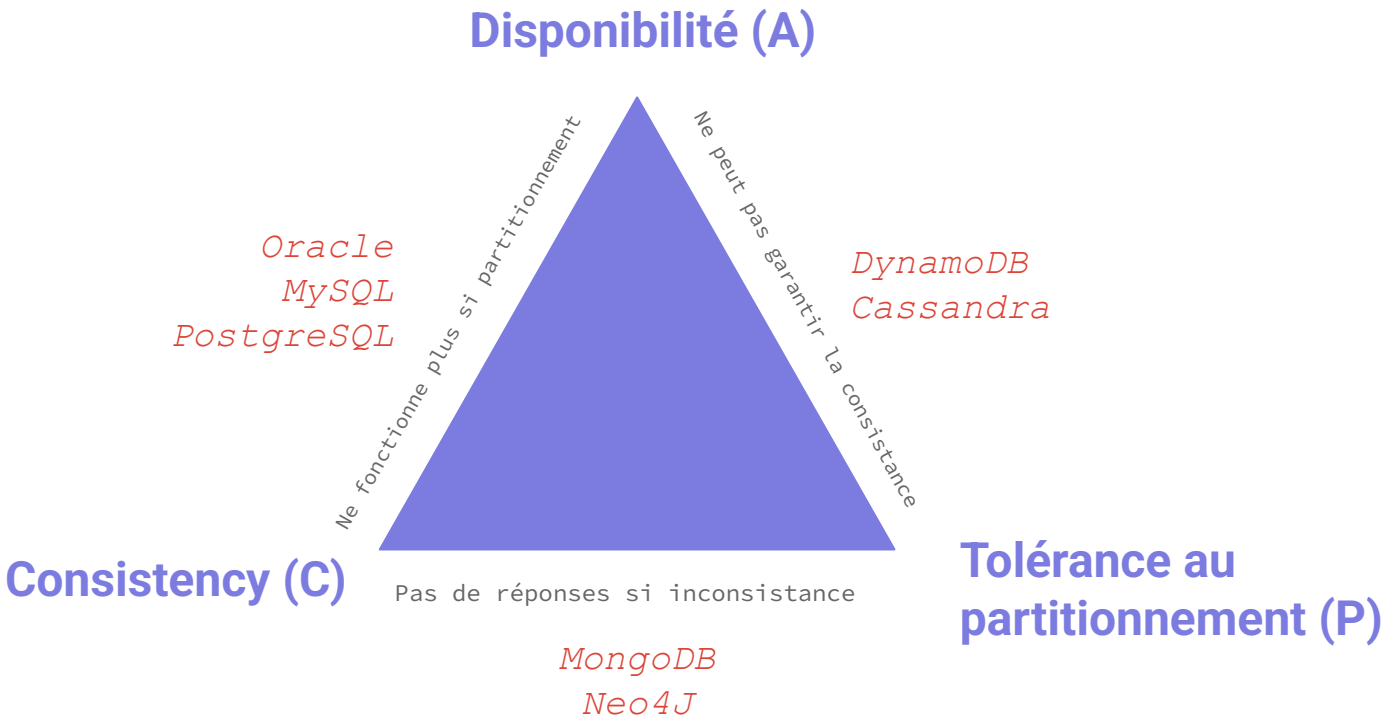
Par l'absurde : imaginons que les 3 propriétés sont validées



Inconsistance
Contradiction

Tolérance **Disponibilité**

Problème CAP



MongoDB
Rappels JSON

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

JSON - JavaScript Object Notation

C’est un format où les données sont semi-structurées

- Format textuel d’échange de données
- Dérive de la représentation littérale des valeurs de JavaScript (version ECMAScript 3, 1999)
- Une grammaire qui décrit des données simples à lire et écrire à la fois pour un humain et une machine
- Sa simplicité permet de l’utiliser indépendamment d’un quelconque langage hôte (modulo quelques conversions)

Des arbres !

- Valeurs atomiques
  - Des chaînes de caractères
  - Des nombres
  - 3 constantes : true, false et null.
- 2 types de valeurs composites
  - Une collection de paires nom/valeur ~= objet, dictionnaire, enregistrement, liste de paire
    - Une collection est délimitée par des accolades
    - Ses paires sont séparées par des virgules
    - Le nom et la valeur d'une paire sont séparés par un deux-points
  - Une séquence de valeurs ~= un tableau, un vecteur, une liste…
    - Un tableau est délimité par des crochets
    - Ses éléments sont séparées par des virgules

Exemples

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP3

- Passer de SQL à JSON et vice-versa
- Utiliser Mongo

TP Noté (Reste de la séance)
Conception d'un tutoriel NoSQL

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

6 sujets :

- REDIS (key-value store)
- ClickHouse (columnar)
- InfluxDB (time series)
- Elasticsearch (search)
- Milvus (Vector)
- Sharding with MongoDB

Inscriptions ici

Rendus :

- tutoriel conteneurisé
- repo GitHub et README du tuto
- utilisation d'un dataset dispo en ligne de votre choix
- visualisation des données (si vous avez le temps)

Contenu de l'UE

- [2h30] 11/02
  - Introduction, installation outils, et rappels SQL (avec MySQL)
- [3h45] 18/02 matin
  - Cours NoSQL
  - TP JSON et MongoDB
  - **Travail en groupe NoSQL**
- [2h30] 18/02 aprem
  - **Travail en groupe NoSQL**
- [3h45] 25/02
  - Parallélisme et Map-Reduce
  - Spark et Spark SQL
- [3h45] 04/03 matin
  - Spark Streaming et Kafka
- [2h30] 04/03 aprem
  - NoSQL Neo4J (orienté graphes) - **Thomas Hassan / Orange Labs**
- [1h45] 11/03 matin **Quizz et Oraux tutos NoSQL**
- **Projets par 4**
  - **[2h+3h45] 11/03**
  - **[3h45+2h30] 18/03**
  - **[3h45] 25/03**
- **Oraux projets le 25/03 après-midi**

Calculs parallèles
Map-Reduce

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Motivation

Début des années 2000, Google automatise l'exploration et l'indexation du web

- Plus de 1 milliard de pages de 20 Ko = 20 To
- Performance d'un disque dur : ~ 20 Mo/s, 10 Go
- Il faut :
  - pour enregistrer le web : 2000 disques durs
  - pour traiter le web :
    - 1 mois pour "lire" le web avec une machine
    - 8 minutes avec 2000 machines

Il faut **paralléliser** les calculs ! -> **Map-Reduce**

Calculs parallèles

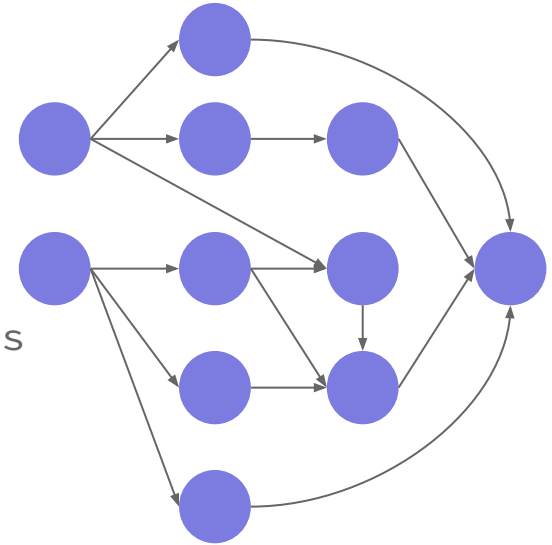
Le calcul parallèle ça date de bien longtemps ! ~Années 1960

On parallélise pour

- aller plus vite
- traiter de plus gros problèmes
- traiter plus de données

Différents types de parallélisme :

- Parallélisme de **tâches**
  - exécution de plusieurs tâches en même temps
  - ordre partiel de tâches
  - partage des données
- Parallélisme de **données**
  - division des données
  - application du même calcul sur les différentes données



Map-Reduce ?

- Parallélisme de **données**
- Calculs “simples” sur beaucoup de données
  - indexes inversés
  - liens entre les pages web
  - page rank etc.
- Aller plus vite et traiter plus de données
- Ne cherche pas à optimiser au maximum (pas HPC)
- Mais cherche à **abstraire la complexité**
  - un **modèle de programmation simple**
  - parallélisme **automatique** / **distribution** des données
  - les **communications** et la **coordination** des calculs
  - la tolérance aux pannes (fault tolerance)
  - l’équilibrage de charges (load balancing)

Google MapReduce (2000)

Map et Reduce sont deux fonctions classiques d'ordre supérieur de la programmation fonctionnelle

- Une bibliothèque (C++)
  - fournit les primitives de programmation sous forme d'appels à la bibliothèque
  - gère les "détails" précédents une fois pour toute

Les étapes :

1. **Lecture** des données
2. **Map** : extrait de chacune des données un résultat intermédiaire
3. **Shuffle** (GroupByKey, Sort) : réarrange et trie les résultats intermédiaires
4. **Reduce** : filtre, agrège, résume, transforme… les résultats intermédiaires
5. **Ecriture** du résultat

Seules les étapes Map et Reduce dépendent du problème.

Map

Soit une liste `l = list(e1, …, en)`

- `map(f, l)` applique f à chacun des éléments de la liste
- `map(f, l) = list(f(e1), …, f(en))`
- Le principe s'étend aux collections.

Reduce

Soit une liste `l = list(e1, …, en)`

- `reduce(op, l)` où `op` est un opérateur binaire **associatif**
- effectue le calcul `e1 op e2… op en-1 op en`

Dans les cas limite (n <= 1)

- la réduction retourne un résultat optionnel (e1 pour n = 1)
- un 3ème paramètre définit une valeur à associer à la liste vide

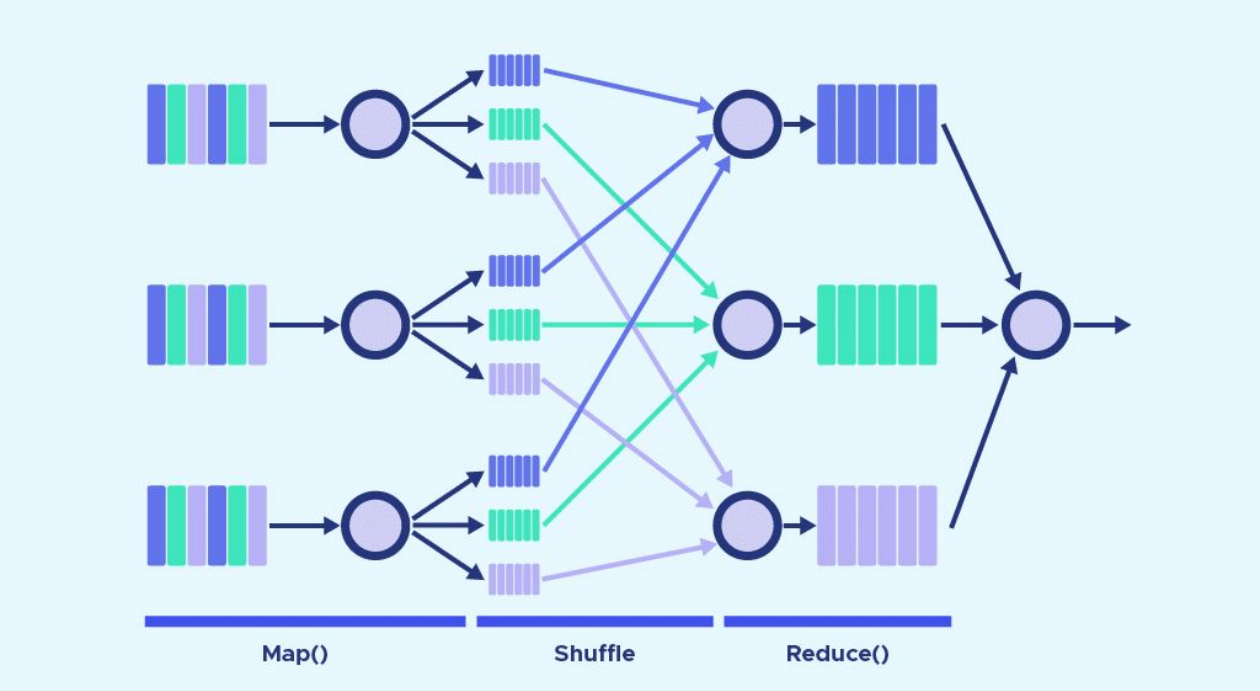
Hadoop

Projet Apache depuis 2006 (basé sur des développements antérieurs notamment de Doug Cutting, et inspirés par GFS et Google MapReduce)

- Écrit en **Java**
- Fournit le modèle de programmation **MapReduce** au dessus de
  - HDFS (Hadoop Distributed File System)
  - YARN (Yet Another Resource Negotiator) qui gère l'allocation des ressources et l'ordonnancement des tâches (depuis 2012)

En voie de désuétude… mais le modèle de programmation reste d'actualité !

Vision de haut niveau



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP5-1

- Map-Reduce en Python
  - comprendre sans parallélisme ces fonctions d’ordre supérieur

Machines parallèles

HPC, Cloud, clusters

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Cloud vs HPC

- La **Data** c'est plutôt le monde du **Cloud**
  - orienté "services"
  - disponibilité
  - élasticité
  - **la donnée c'est du pouvoir**
- Le **calcul parallèle** c'est plutôt le monde du **HPC**
  - calculer **le plus vite possible** (Calcul Haute Performance)
  - résoudre des problèmes nouveaux grâce à la puissance de calcul (décodage du génome humain, IA, forages, astronomie, météo et climat, etc.)
  - plutôt orienté "science"
  - **le calcul c'est du pouvoir**

Le Cloud/Data/IA comme le HPC/IA sont désormais des éléments géo-politiques. Et les deux mondes sont de plus en plus proches

Le BigData est un peu au milieu mais un peu plus proche du Cloud

Parlons machines HPC

- Top500 … La course à la puissance de calcul
  - Et la Chine ?
- Green500 … “Green” mmoui

FLOPS (FLoating point Operations Per Second) : la performance s’exprime en opérations (additions ou multiplications) à virgule flottante par seconde

Cluster de machines

Définition

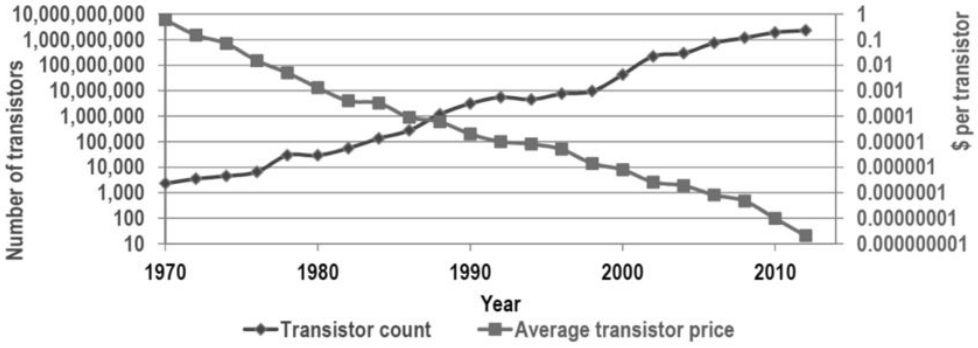
- A **grand nombre de serveurs** très **proches** les uns des autres
- Interconnectés par un **réseau haute performance**

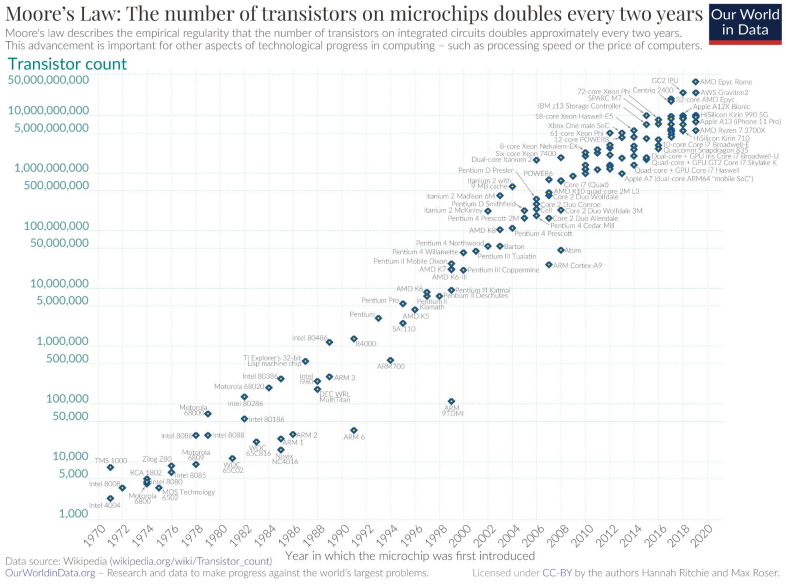
Remarques

- Une machine HPC est un très gros cluster de machines
- Les centres de calculs HPC sont constitués de plusieurs clusters
- Les clouds sont aussi un ensemble de clusters divisés en régions
- Hadoop ou Spark sont typiquement déployés sur des clusters
- Il est bon de noter qu’on peut aussi faire des clusters de machines virtuelles

Loi de Moore et architectures

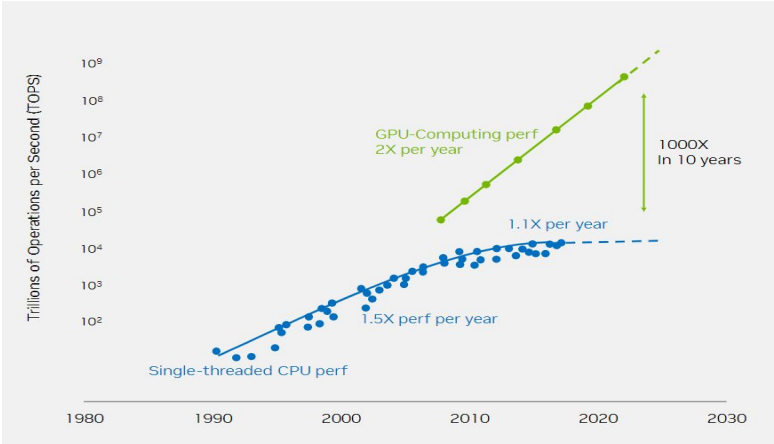
- **Loi empirique de 1965** (revue en 1975)
- Doublement du nombre de transistors sur une puce CPU tous les deux ans à coût constant



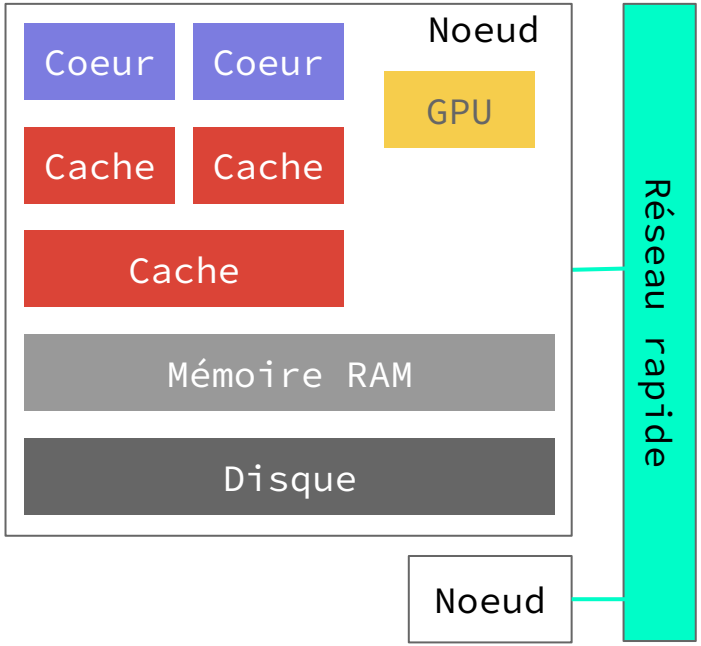


Loi de Moore et architectures

- Mauvaise interprétation - fréquence des processeurs
  - stagne depuis 2004 à 5GHz
  - record de 500 GHz par IBM
    - transistor équipé d'une puce à base de silicium-germanium
    - refroidi à -269 °C à l'hélium liquide
- Les architectures se sont complexifiées
  - la loi reste vrai
  - **multi-core**
  - **accélérateurs graphiques**

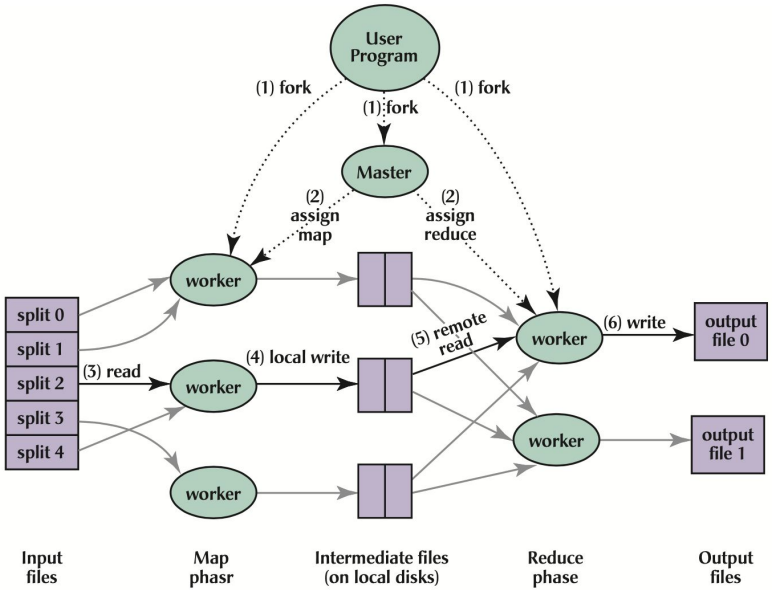


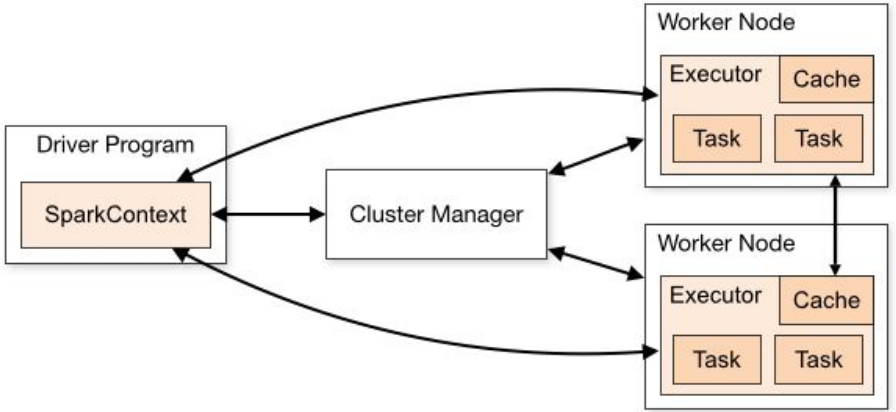
ARCHITECTURE D'UNE MACHINE ET D'UN CLUSTER



- **Coeur** : ressource de calcul
- **Processeur** / CPU : composé de 1 ou plusieurs coeurs
- **Mémoire** : organisée de façon hiérarchique
- **Noeud** : plusieurs processeurs qui partagent une même mémoire (NUMA)
- **Réseau rapide** : relie les noeuds entre eux
- **Système de fichier** : entrées/sorties sur disques

MapReduce, Hadoop et Spark



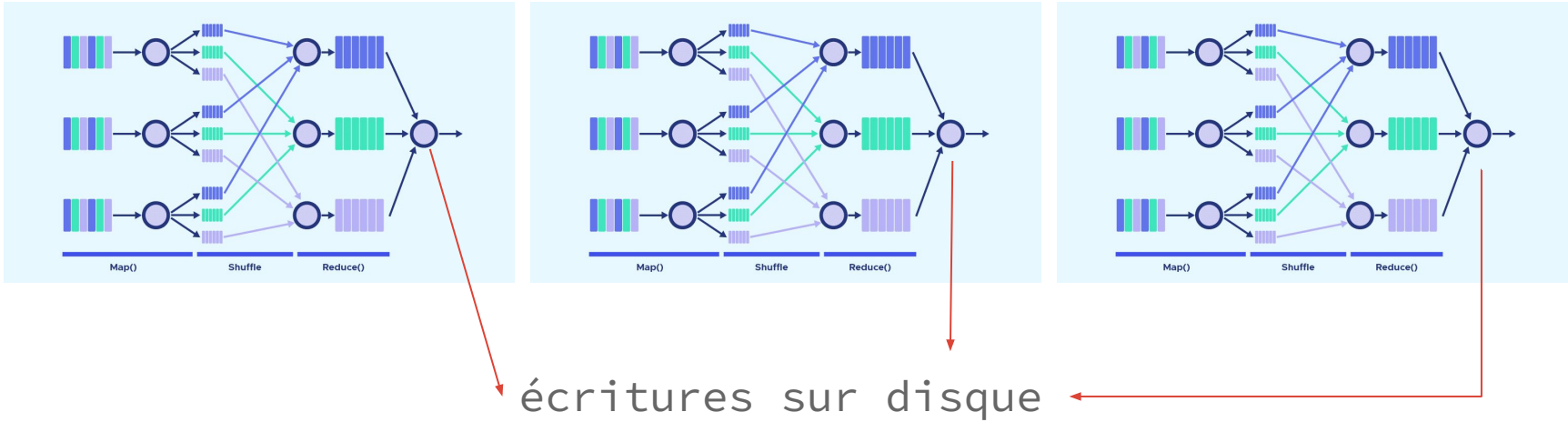


SPARK

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Problème d'écritures et de lectures sur disque

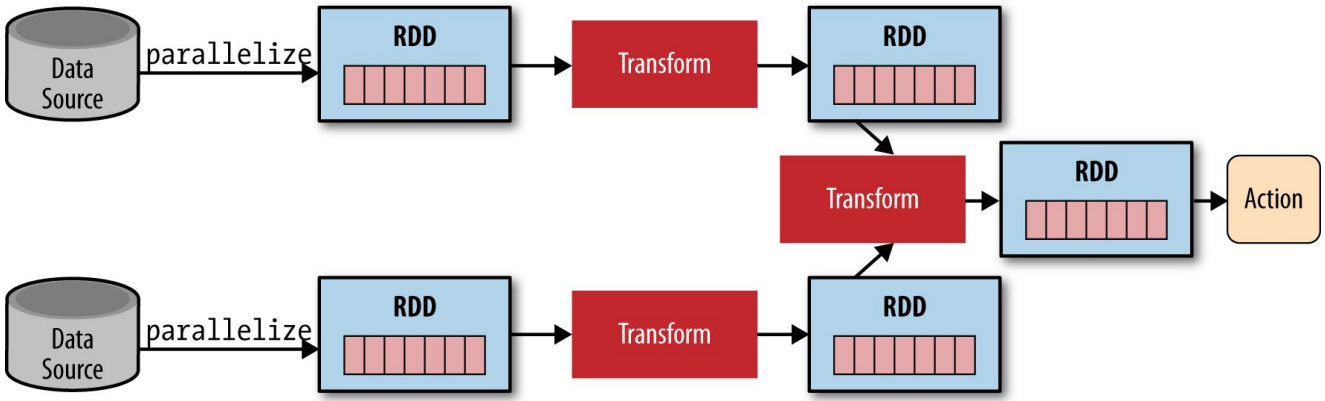
Avec Google MapReduce / Hadoop



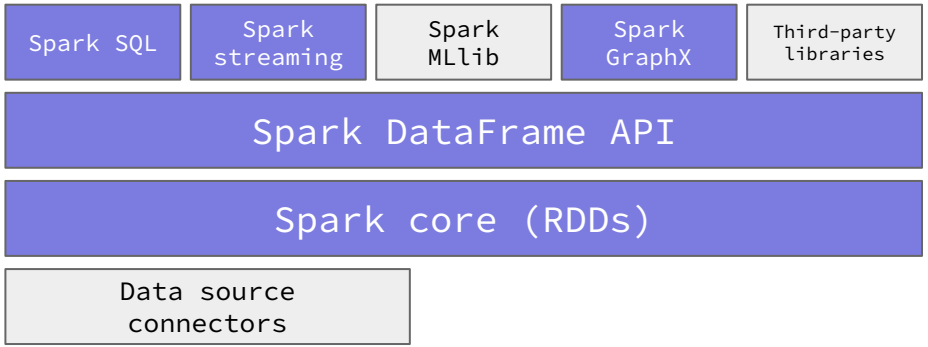
Aussi ne pas faire nécessairement un reduce après le map ? plusieurs map, des filtres, et autres opérations

In-memory approach

- Ne plus faire d'écritures intermédiaires
- Pouvoir enchaîner des **transformations** sur des **données en mémoire** et déjà **partitionnées**
- Ecrire seulement lorsqu'une **action** est demandée (write, reduce etc.)

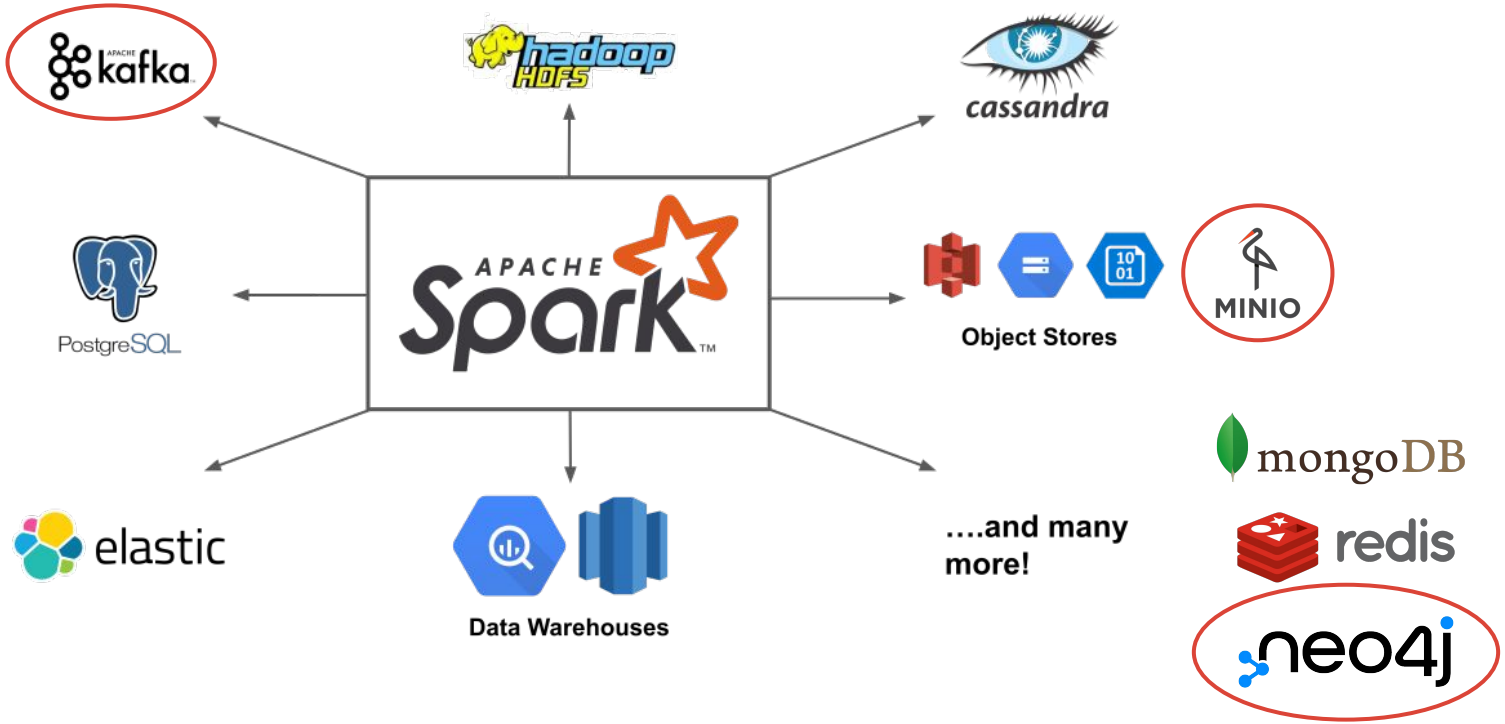


Spark



- **Unified analytics engine** for large-scale data processing
- High-level APIs in Java, Scala, Python and R
- Rich set of **higher-level tools** including
  - **Spark SQL** for SQL and structured data processing
  - MLlib for machine learning
  - GraphX for graph processing
  - **Streaming** for incremental computation and stream processing

Spark data sources



Spark core - RDD

- The main abstraction Spark provides is a **Resilient Distributed Dataset** (RDD)
- An RDD is a **Collection of elements partitioned** across the nodes of the Spark cluster
- RDDs can be operated on in parallel
- **2 types** of computations on RDDs
  - **Transformations** create a new dataset from an existing one
  - **Actions** return a value to the *driver* program after running a computation on the dataset

Spark core - Transformations et actions

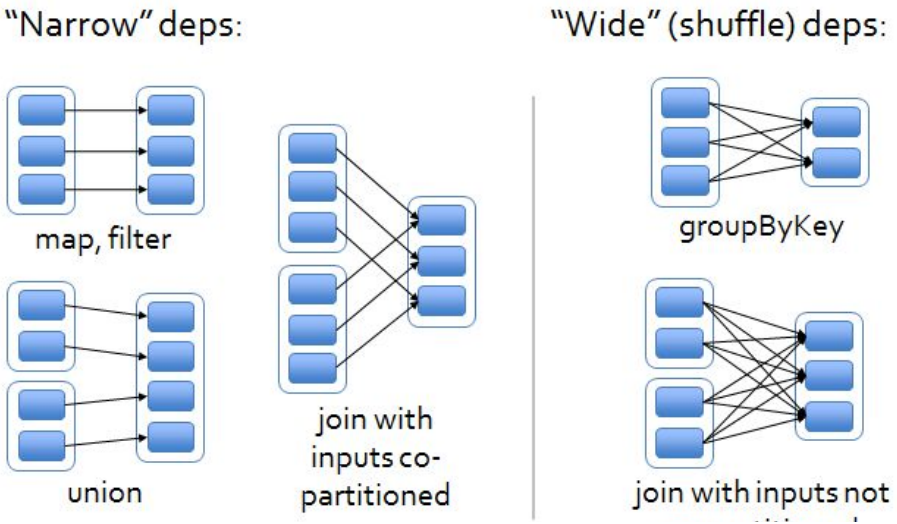
Création initiale d'un RDD : Fonction parallelize

Transformations

- Map
- Filter
- GroupBy
- etc.

Actions

- Collect
- Reduce
- Save
- etc.

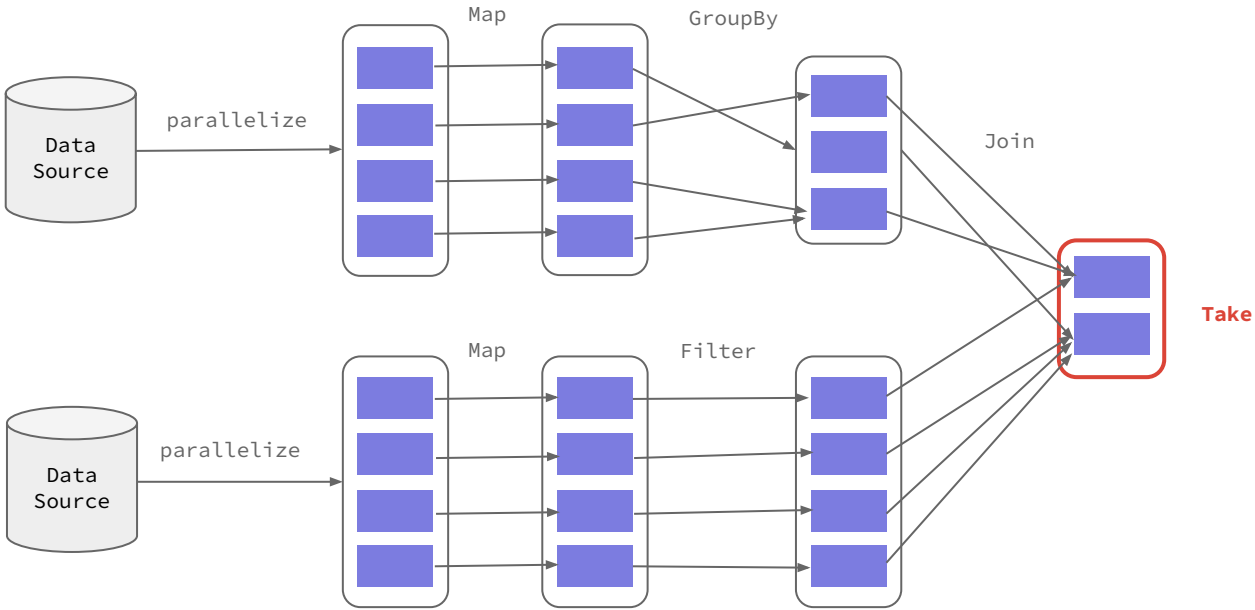


Spark core - Transformations et actions

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	RDD[T] ⇒ RDD[U]
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	RDD[T] ⇒ RDD[T]
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	RDD[T] ⇒ RDD[U]
	$sample(fraction : \mathrm{Float})$	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	$groupByKey()$	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	$union()$	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	$join()$	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	$cogroup()$	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	$crossProduct()$	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	$count()$	:	RDD[T] ⇒ Long
	$collect()$	:	RDD[T] ⇒ Seq[T]
	$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	RDD[T] ⇒ T
	$lookup(k : \mathrm{K})$	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.,* HDFS

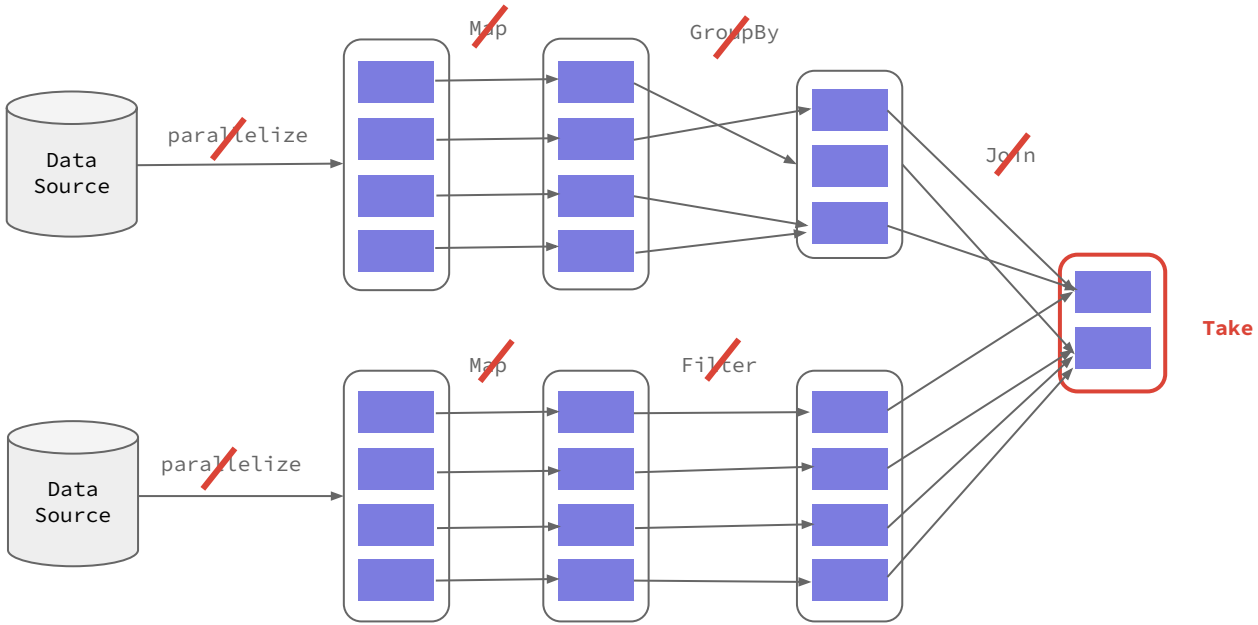
Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

Spark core - Example

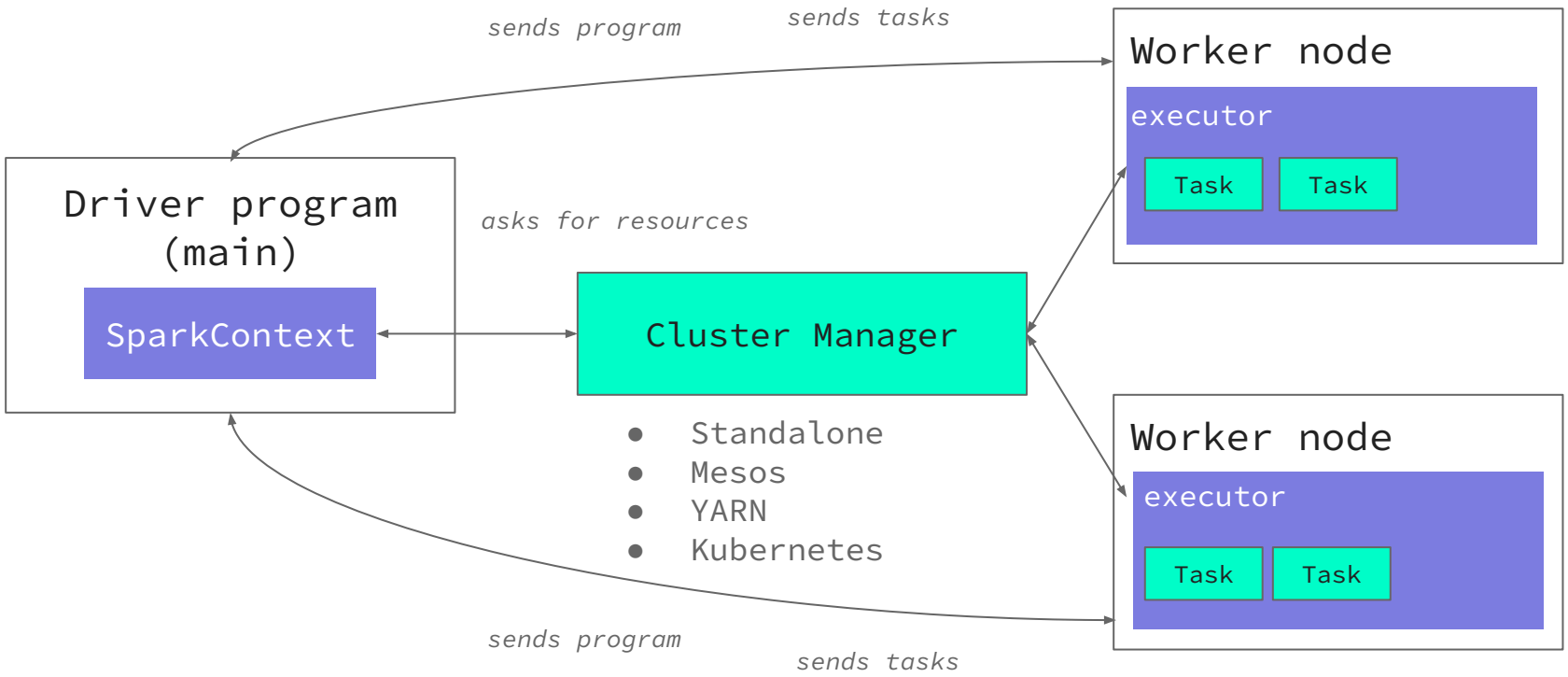


Lazy evaluation

- Nothing is computed on RDDs until an action is called!
- Each transformed RDD may be recomputed each time you run an action on it
- Can also persist an RDD in memory using the `persist` (or cache) method



Architecture de Spark



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP5-2

- TP Spark
- Utilisation d’un connecteur pour vos bases NoSQL respectives