databases, data engineering & big data

Introduction to Big Data & Data Engineering

ESME SUDRIA

Luc Marchand - luc.marchand.pro@proton.me
Maxence Talon - maxencetallon@gmail.com

About me

Data Engineer at TotalEnergies Digital Factory

maxencetallon@gmail.com

- 2 years at Octo Technology
- 1 years until now a Total Energies
- 2th year as a teacher at ESME
- Main topics of interests
  - Spark & Databricks
  - data engineering & Ml ops
  - Best practices / Software craftsmanship

If you have any question, feel free to drop me a mail at any time

Global Syllabus

01 Introduction and main concepts

02 SQL, set up env and practical work

03 NoSQL world

04 **Introduction to Big Data & Data Engineering**

05 Kafka & event driven architectures

06 Spark & Delta

07 Warehouse, DBT & BI

08 IA - MLOps & RAG

Course syllabus

01 What is Big Data ?

02 Big data architectures

03 What is a data engineer ?

05 Columns -Oriented Storage

04 Transformations & orchestration

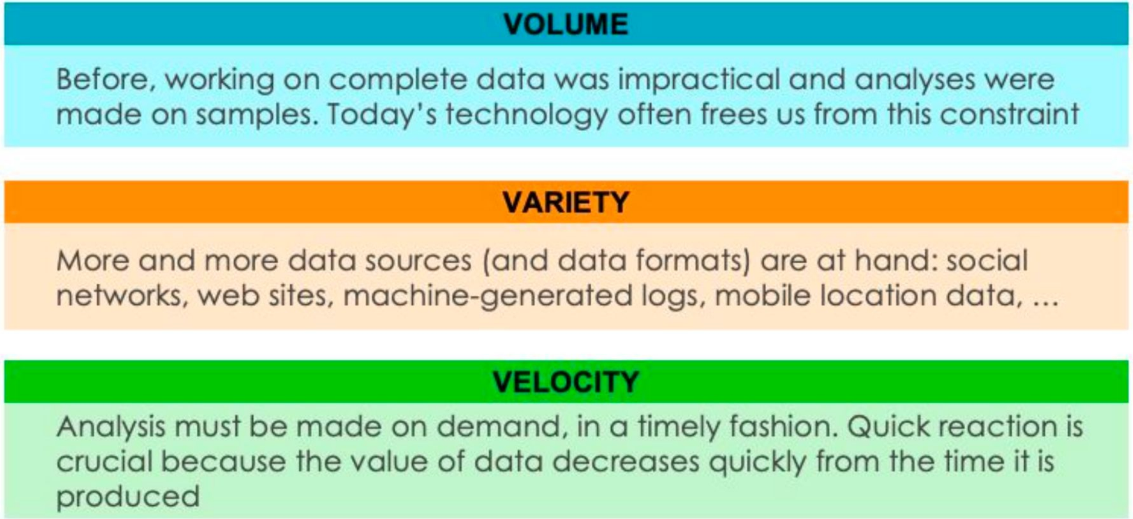
06 Hadoop ,Map Reduce & Spark

01

What is Big Data ?

A data-centric definition (Gartner’s definition)

Big Data is characterized by the 3 V’s **V**olume, **V**ariety, **V**elocity



VOLUME

Before, working on complete data was impractical and analyses were made on samples. Today's technology often frees us from this constraint

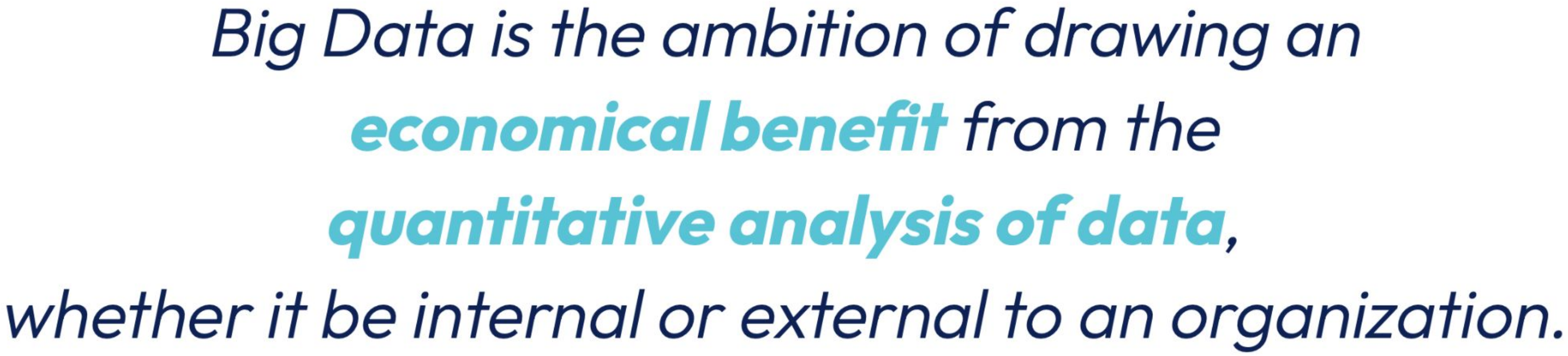
VARIETY

More and more data sources (and data formats) are at hand: social networks, web sites, machine-generated logs, mobile location data, ...

VELOCITY

Analysis must be made on demand, in a timely fashion. Quick reaction is crucial because the value of data decreases quickly from the time it is produced

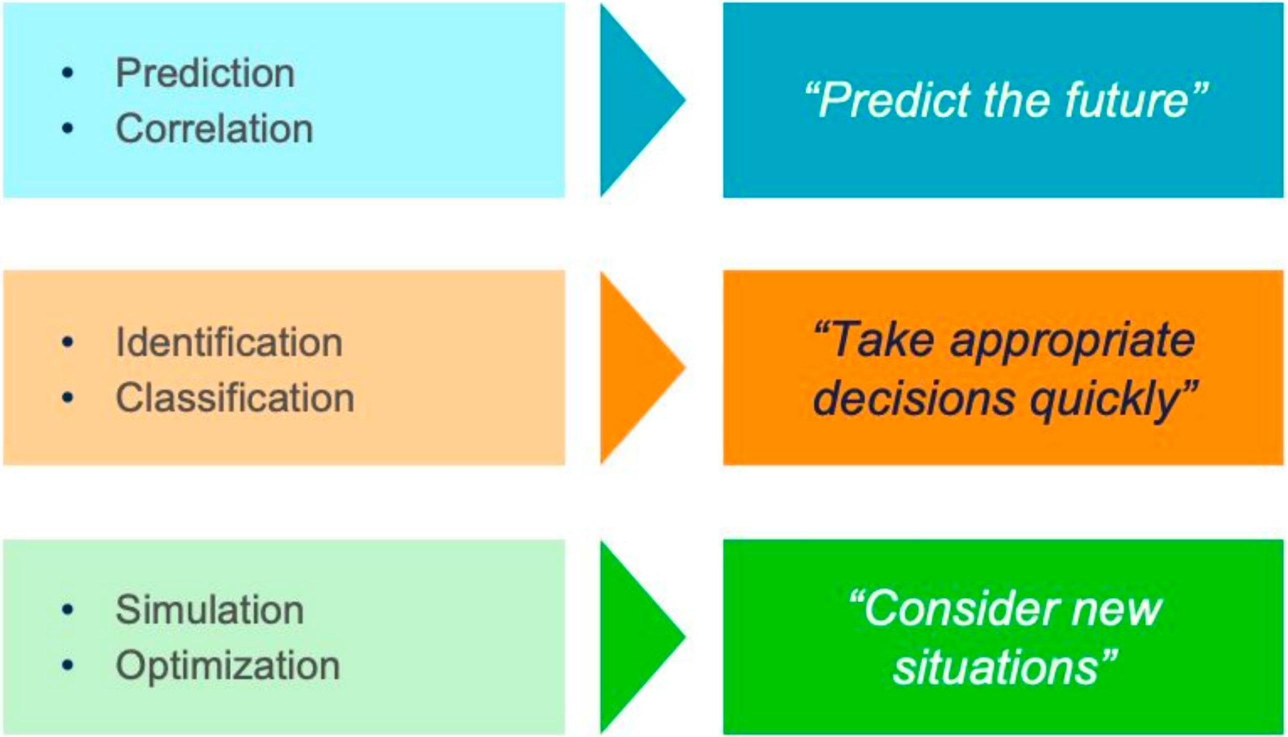
A general definition



Big Data is the ambition of drawing an ***economical benefit*** *from the* ***quantitative analysis of data****,*
whether it be internal or external to an organization.

Other definitions can be found on this website : https://www.opentracker.net/article/definitions-big-data

What do you mean by quantitative analysis?



- Prediction
- Correlation

"Predict the future"

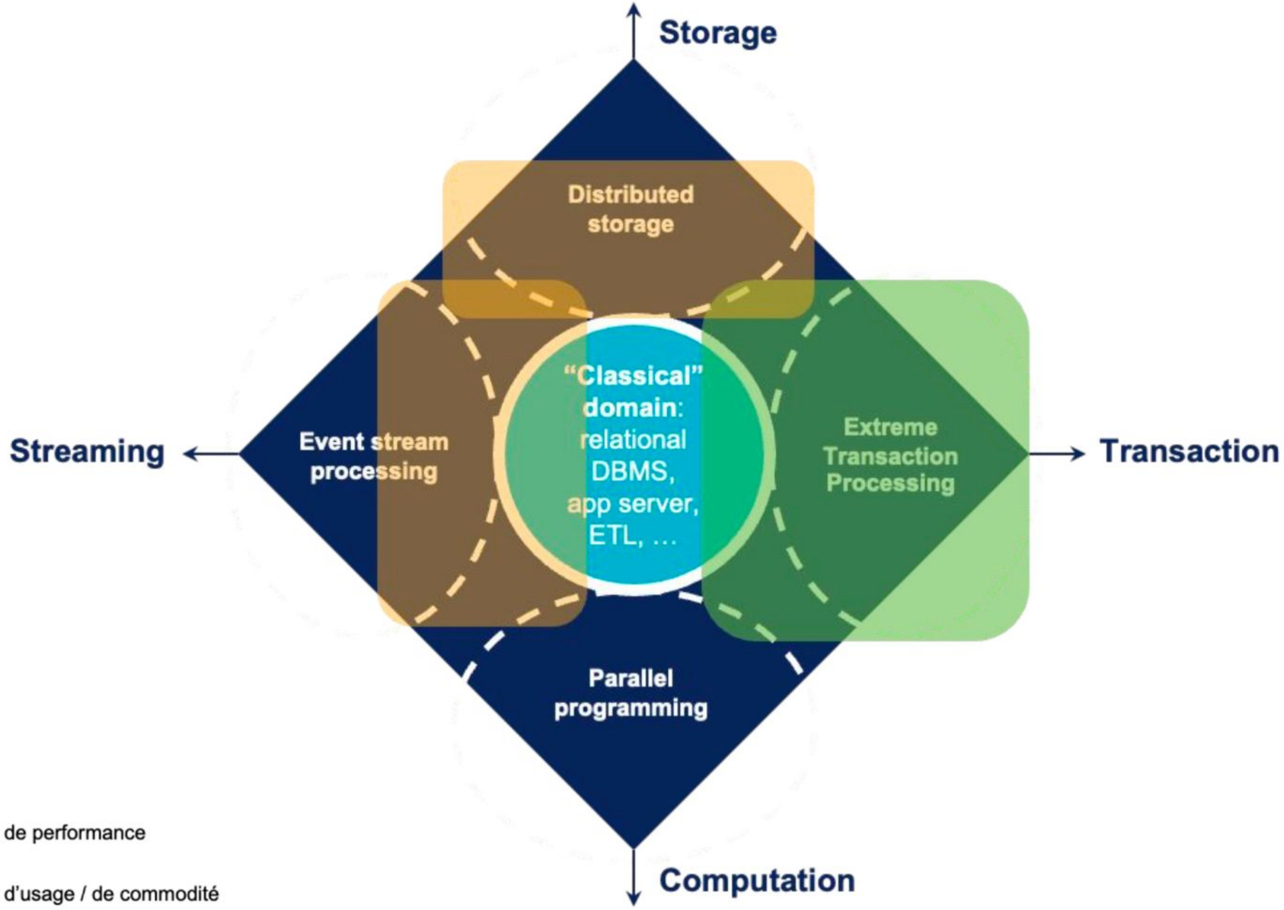
- Identification
- Classification

"Take appropriate decisions quickly"

- Simulation
- Optimization

"Consider new situations"

Choisir une famille technologique à partir du modèle en diamant de limitations des DBMS classiques

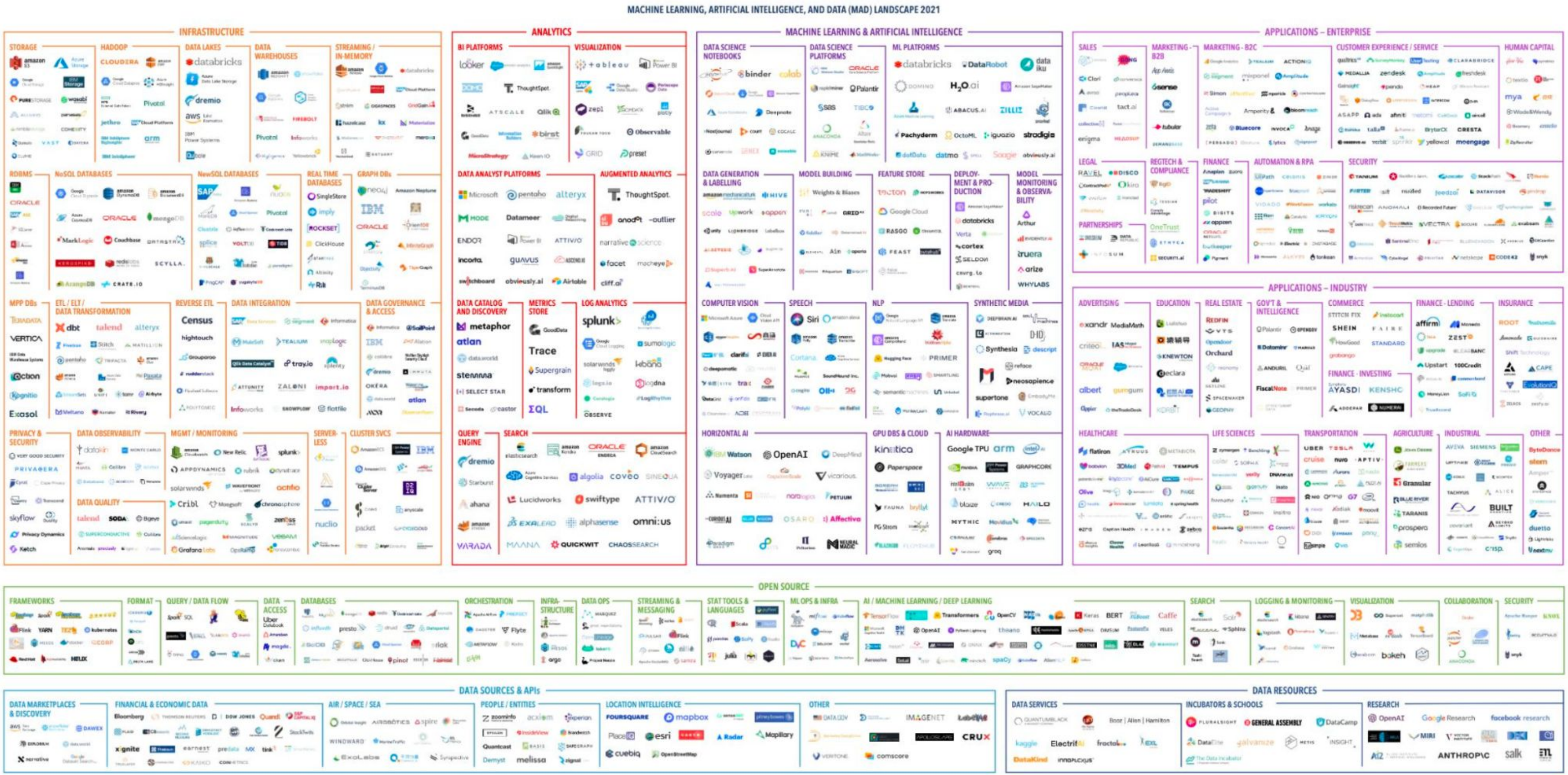


🟩 zone de performance

🟨 zone d'usage / de commodité

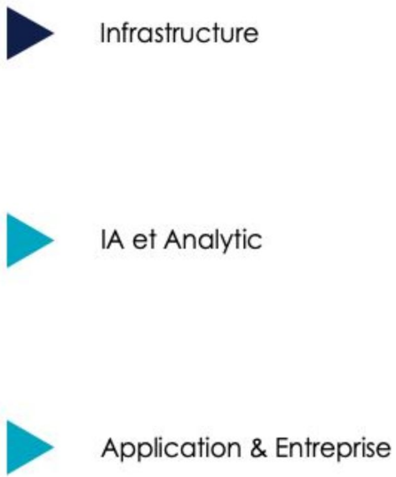
Le Big data par le prisme de l’écosystème

Le Big Data rassemble sous une même bannière un large écosystème technologique



https://mattturck.com/data2021/

Transporter et stocker des données à un niveau jamais atteint



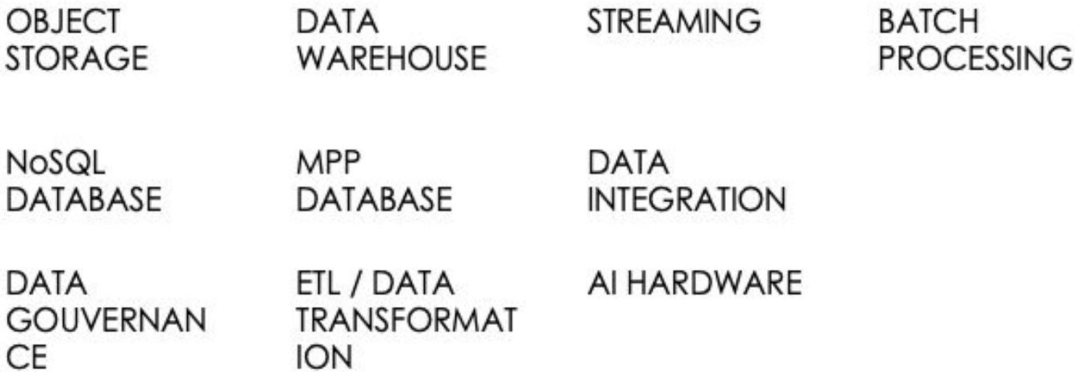
Infrastructure

IA et Analytic

Application & Entreprise

L'infrastructure est la couche primaire d'un environnement big data. Elle fournit des solutions technologiques pour transporter et stocker des volumes de données qui dépassent les capacités d'une machine unique.

Chaque minute, Netflix diffuse environ 220 000 heures de vidéo à ses abonnés.



OBJECT STORAGE

DATA WAREHOUSE

STREAMING

BATCH PROCESSING

NoSQL DATABASE

MPP DATABASE

DATA INTEGRATION

DATA GOUVERNANCE

ETL / DATA TRANSFORMATION

AI HARDWARE

Des technologies que vous avez / aller aborder dans ce cours

- Infrastructure
- IA et Analytic
- Application & Entreprise

Data Warehouse

dbt

Streaming/Event

kafka

Object storage

data lake

Batch processing

APACHE Spark™

No Sql

mongoDB

Transformer des données jusque là inexploitables en connaissances activables

L'IA et l'analytique est la seconde couche d'un environnement big data.Le volume de donnée produit dépasse notre capacité à le stocker pour différer notre traitement.

L'usage de la vidéo ou le speech to text ouvre la voie à des usages en dehors du canvas classique de l'IT (au travers d'un écran) grace aux approches de computer vision, NLP et depuis 2022 de l'essor des LLM.

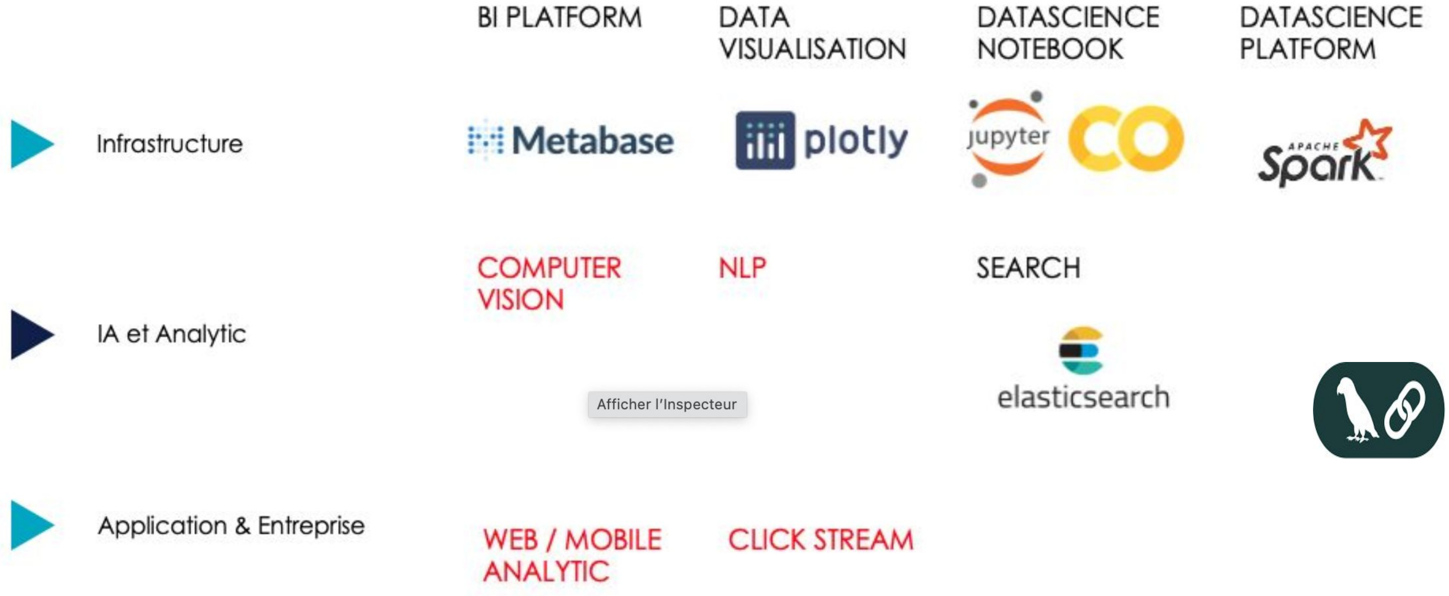
Infrastructure

IA et Analytic

Application & Entreprise

BI PLATFORM	DATA VISUALISATION	DATASCIENCE NOTEBOOK	DATASCIENCE PLATFORM
COMPUTER VISION	NLP	SEARCH	RAG
WEB / MOBILE ANALYTIC	CLICK STREAM		

Transformer des données jusque là inexploitables en connaissances activables



Infrastructure

BI PLATFORM

Metabase

DATA VISUALISATION

plotly

DATASCIENCE NOTEBOOK

jupyter

CO

DATASCIENCE PLATFORM

APACHE Spark

IA et Analytic

COMPUTER VISION

NLP

SEARCH

elasticsearch

Afficher l'Inspecteur

Application & Entreprise

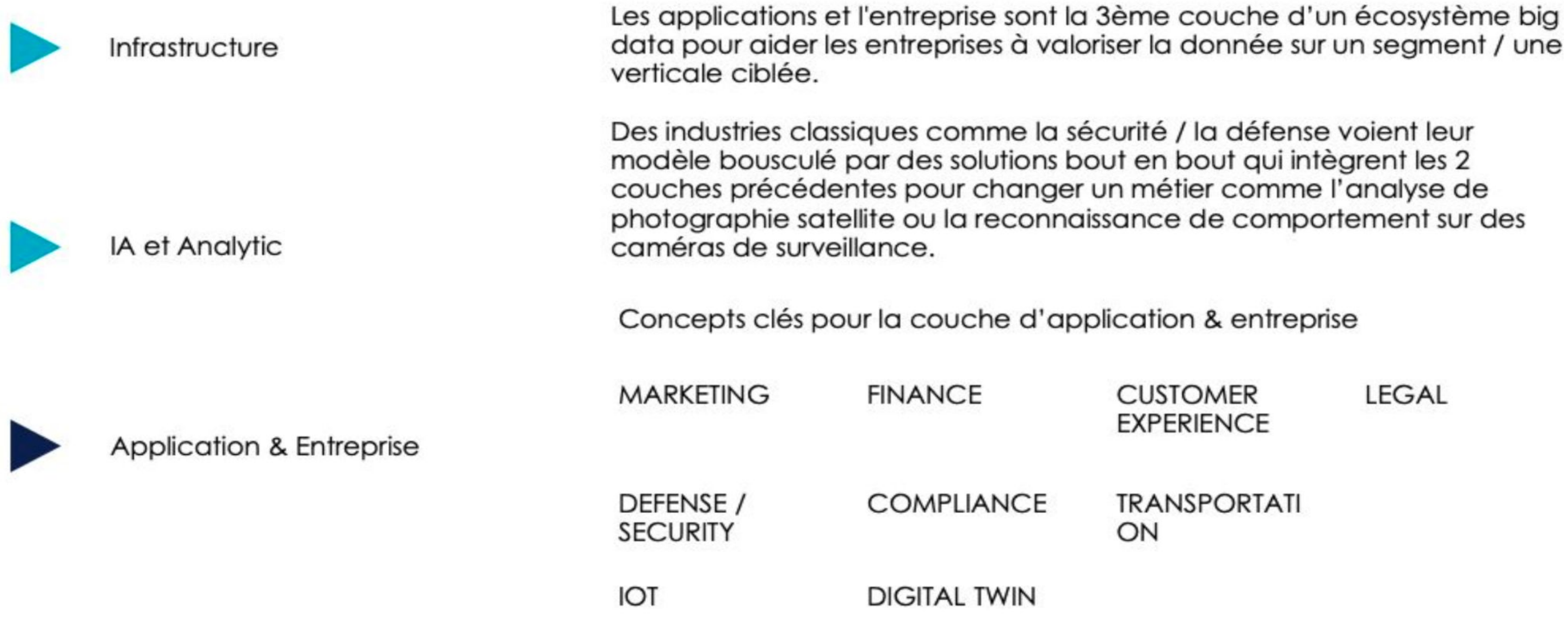
WEB / MOBILE ANALYTIC

CLICK STREAM

RAG

LangChain

Outiller des métiers historiques et faire émerger de nouveaux usages



- Infrastructure
- IA et Analytic
- Application & Entreprise

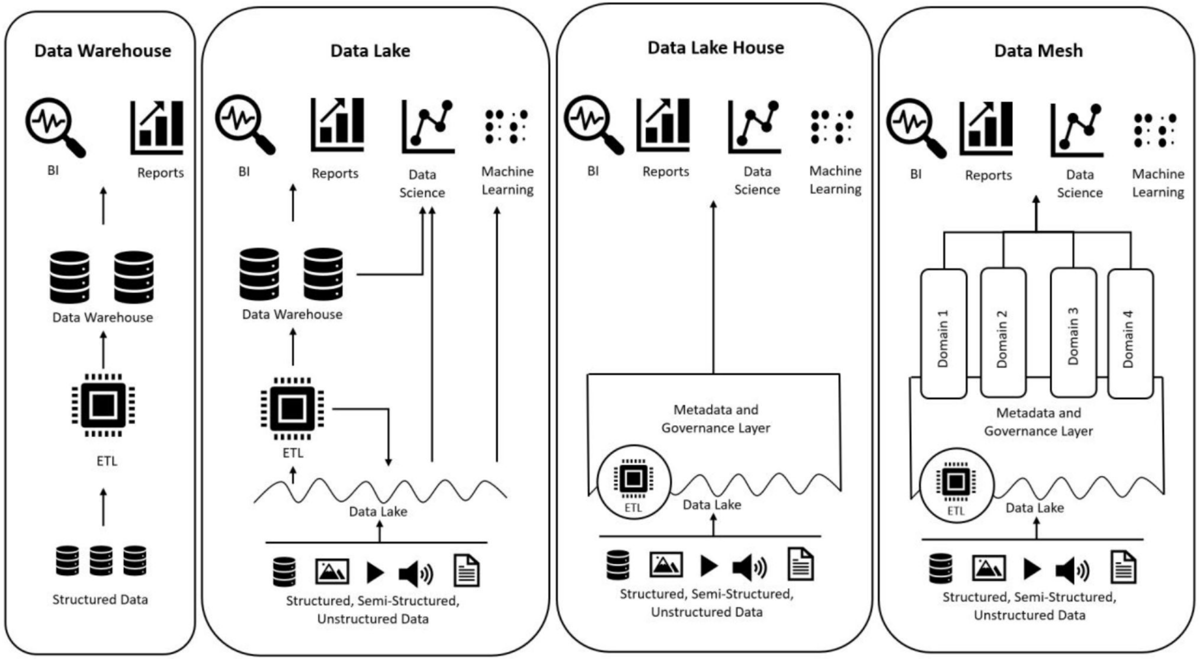
Les applications et l'entreprise sont la 3ème couche d'un écosystème big data pour aider les entreprises à valoriser la donnée sur un segment / une verticale ciblée.

Des industries classiques comme la sécurité / la défense voient leur modèle bousculé par des solutions bout en bout qui intègrent les 2 couches précédentes pour changer un métier comme l'analyse de photographie satellite ou la reconnaissance de comportement sur des caméras de surveillance.

Concepts clés pour la couche d'application & entreprise

MARKETING
FINANCE
CUSTOMER EXPERIENCE
LEGAL
DEFENSE / SECURITY
COMPLIANCE
TRANSPORTATI ON
IOT
DIGITAL TWIN

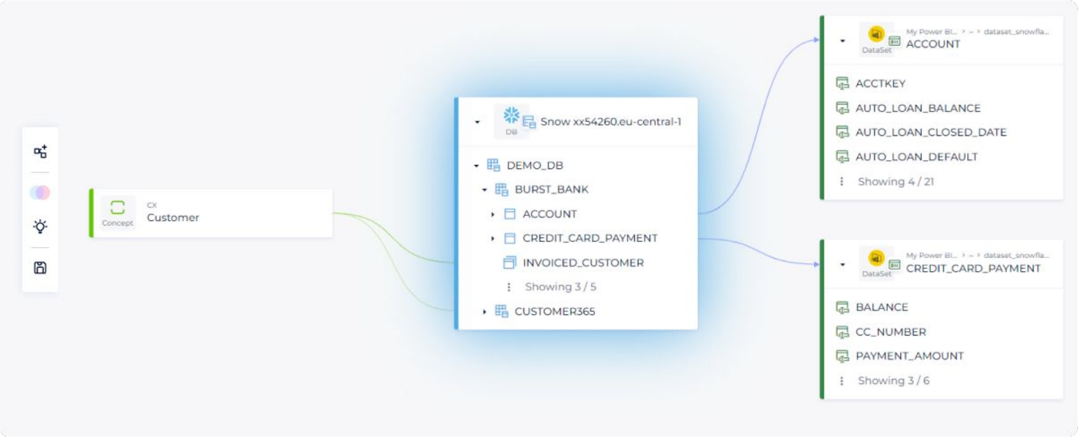
Récapitulatif des différentes architectures



Data Governance & Data Catalogue

Data governance in a data lake ensures the quality, security, and compliance of data by establishing policies and processes that unify and manage diverse data sources, enabling effective integration and analysis.

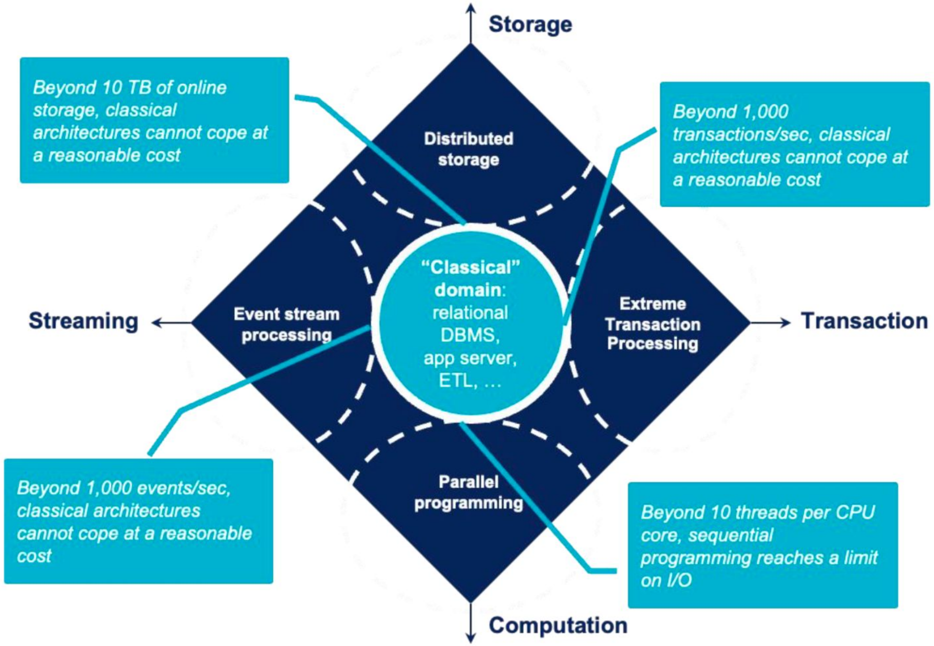
A data catalog in a data lake serves as a comprehensive inventory that organizes and provides metadata for all data assets, facilitating easy discovery, governance, and efficient data utilization.



02

Big data architectures

An overview of the Big Data technological landscape



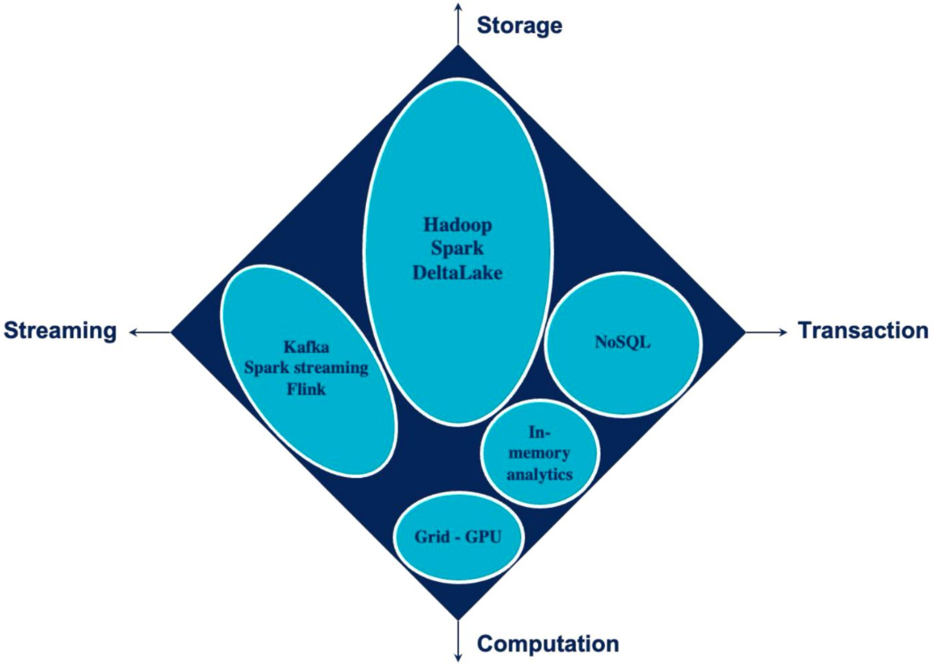
Beyond 10 TB of online storage, classical architectures cannot cope at a reasonable cost

Beyond 1,000 transactions/sec, classical architectures cannot cope at a reasonable cost

Beyond 1,000 events/sec, classical architectures cannot cope at a reasonable cost

Beyond 10 threads per CPU core, sequential programming reaches a limit on I/O

An overview of the Big Data technological landscape



NoSQL and Extreme Transaction Processing

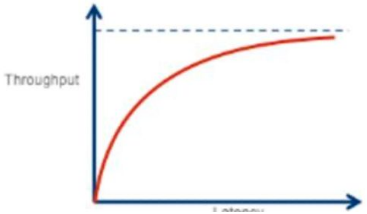
Extreme Transaction Processing

- (see the course on NoSQL for details)

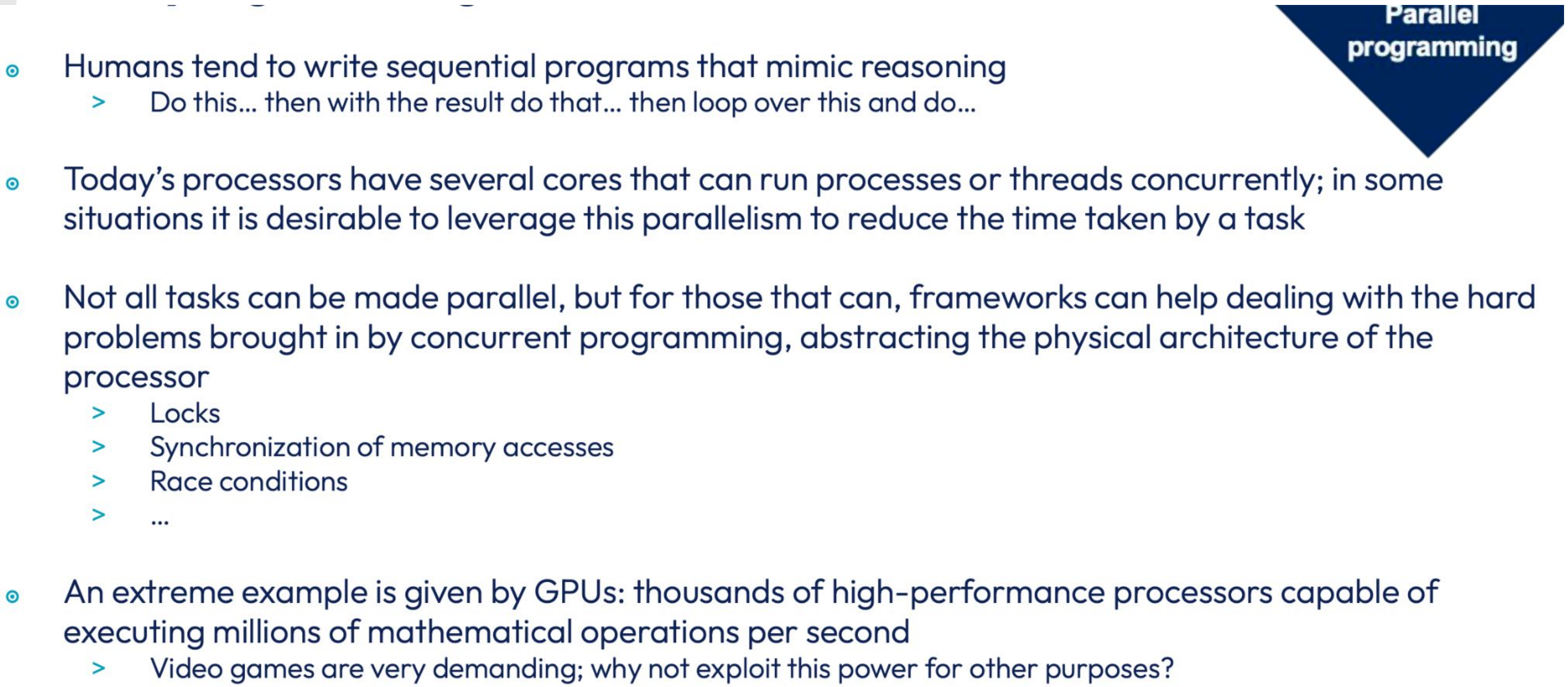
- As a quick reminder, NoSQL is a new database architecture that permits
  - Higher storage capabilities
  - Quicker transactions
  - More flexible data models

- … but at the cost of the ACID properties

- The performance is measured by 2 figures: **throughput** and **latency**
  - Throughput is the number of transactions (fine-grained read or write operations) that can be requested in a given time. It is measured in tps (transactions per second)
  - Latency is the delay between a request and its achievement. It is measured in milliseconds
  - The two are not independent: usually latency increases with throughput, when requests come too fast



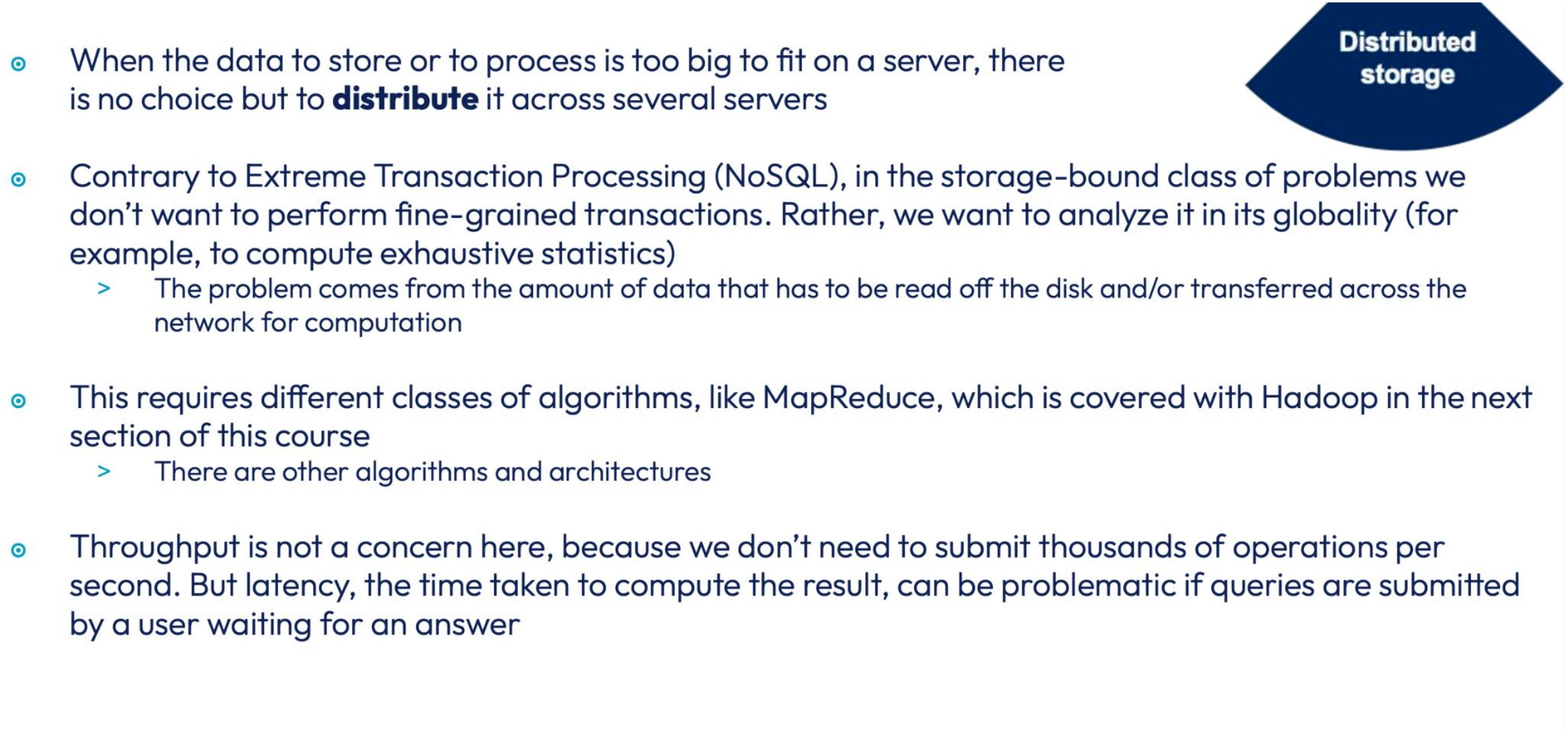
Parallel programming



Parallel programming

- Humans tend to write sequential programs that mimic reasoning
  - > Do this… then with the result do that… then loop over this and do…
- Today's processors have several cores that can run processes or threads concurrently; in some situations it is desirable to leverage this parallelism to reduce the time taken by a task
- Not all tasks can be made parallel, but for those that can, frameworks can help dealing with the hard problems brought in by concurrent programming, abstracting the physical architecture of the processor
  - > Locks
  - > Synchronization of memory accesses
  - > Race conditions
  - > …
- An extreme example is given by GPUs: thousands of high-performance processors capable of executing millions of mathematical operations per second
  - > Video games are very demanding; why not exploit this power for other purposes?

Distributed storage



Distributed storage

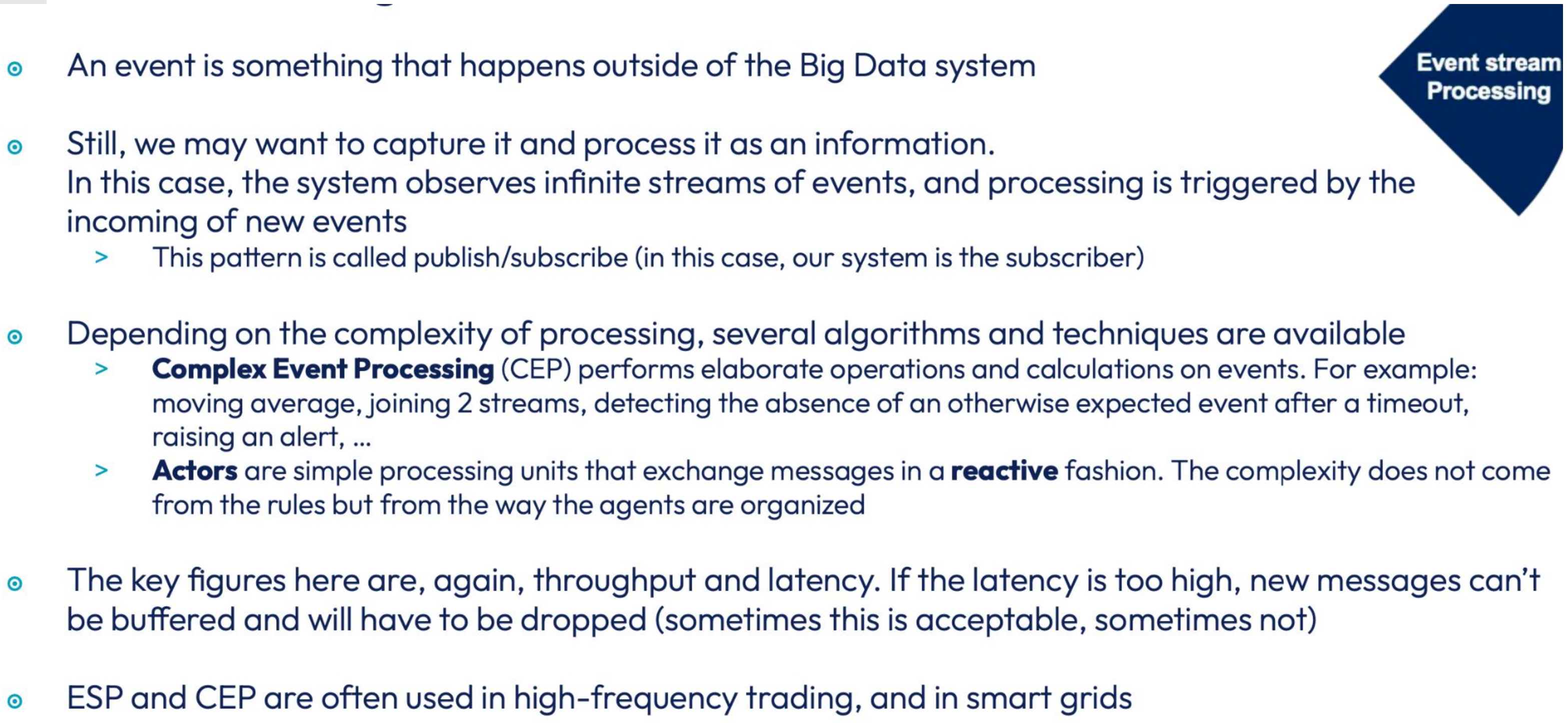
- When the data to store or to process is too big to fit on a server, there is no choice but to **distribute** it across several servers

- Contrary to Extreme Transaction Processing (NoSQL), in the storage-bound class of problems we don’t want to perform fine-grained transactions. Rather, we want to analyze it in its globality (for example, to compute exhaustive statistics)
  - The problem comes from the amount of data that has to be read off the disk and/or transferred across the network for computation

- This requires different classes of algorithms, like MapReduce, which is covered with Hadoop in the next section of this course
  - There are other algorithms and architectures

- Throughput is not a concern here, because we don’t need to submit thousands of operations per second. But latency, the time taken to compute the result, can be problematic if queries are submitted by a user waiting for an answer

Event Stream Processing



Event stream Processing

- An event is something that happens outside of the Big Data system
- Still, we may want to capture it and process it as an information.
  In this case, the system observes infinite streams of events, and processing is triggered by the incoming of new events
  - This pattern is called publish/subscribe (in this case, our system is the subscriber)
- Depending on the complexity of processing, several algorithms and techniques are available
  - **Complex Event Processing** (CEP) performs elaborate operations and calculations on events. For example: moving average, joining 2 streams, detecting the absence of an otherwise expected event after a timeout, raising an alert, …
  - **Actors** are simple processing units that exchange messages in a **reactive** fashion. The complexity does not come from the rules but from the way the agents are organized
- The key figures here are, again, throughput and latency. If the latency is too high, new messages can't be buffered and will have to be dropped (sometimes this is acceptable, sometimes not)
- ESP and CEP are often used in high-frequency trading, and in smart grids

03

What is a data engineer ?

Multiple definitions

Technical definition

A data engineer is an IT professional responsible for :

- designing,
- building,
- managing the infrastructure and systems that support data storage, processing, and analysis.

They create architectures for data generation, work on ETL (Extract, Transform, Load) processes, and ensure that data pipelines are efficient, reliable, and secure.

Data engineers handle large volumes of data, often preparing it for data scientists and analysts.

Business-Focused Definition

In a business context, a data engineer ensures that the organization has access to clean, consistent, and usable data for decision-making.

They manage the backend data operations and build tools to enable data-driven insights, helping the company transform raw data into valuable information that drives strategic planning and operational efficiency.

Developer’s Perspective

For developers, a data engineer is a specialist who bridges the gap between raw data and data-driven applications.

They optimize databases, maintain data warehouses, and develop APIs and data models, allowing developers to create scalable, data-centric applications.

They focus on data structure, storage optimization, and system reliability.

From an Analytics/Scientific View

Data engineers are essential to the data science workflow, as they prepare and preprocess data for analysis.

They create and maintain data pipelines that allow data scientists to focus on building models and deriving insights without worrying about data quality or accessibility.

They work closely with data scientists to ensure that data sources are reliable, up-to-date, and accurate.

Globally

A data engineer

- designs,
- builds,
- manages the infrastructure and systems that enable efficient data storage, processing, and access, ensuring that data is reliable, organized, and available for analysis and decision-making.

04

Transformations & Orchestration

What’s a ETL ? or ELT

Extract **T**ransform **L**oad or **E**xtract **L**oad **T**ransform :

Extract: Data is gathered from various sources.

Transform: The data is cleaned and converted into a suitable format for analysis.

Load: The transformed data is then loaded into a target system

Basic ETL

```python
import pandas as pd
from sqlalchemy import create_engine

# Step 1: Extract
data = pd.read_csv('input_data.csv')

# Step 2: Transform
# Example transformation: Remove rows with missing values and filter for a specific condition
cleaned_data = data.dropna()
filtered_data = cleaned_data[cleaned_data['column_name'] > 10]  # Adjust condition as needed

# Step 3: Load
# Create a database connection (replace with your database URL)
engine = create_engine('sqlite:///my_database.db')  # Example using SQLite

# Write the DataFrame to a SQL table
filtered_data.to_sql('my_table', con=engine, if_exists='replace', index=False)

print("ETL process completed successfully and data loaded into the database!")
```

How do you Run our ELT in the Cloud ?

The world of the function

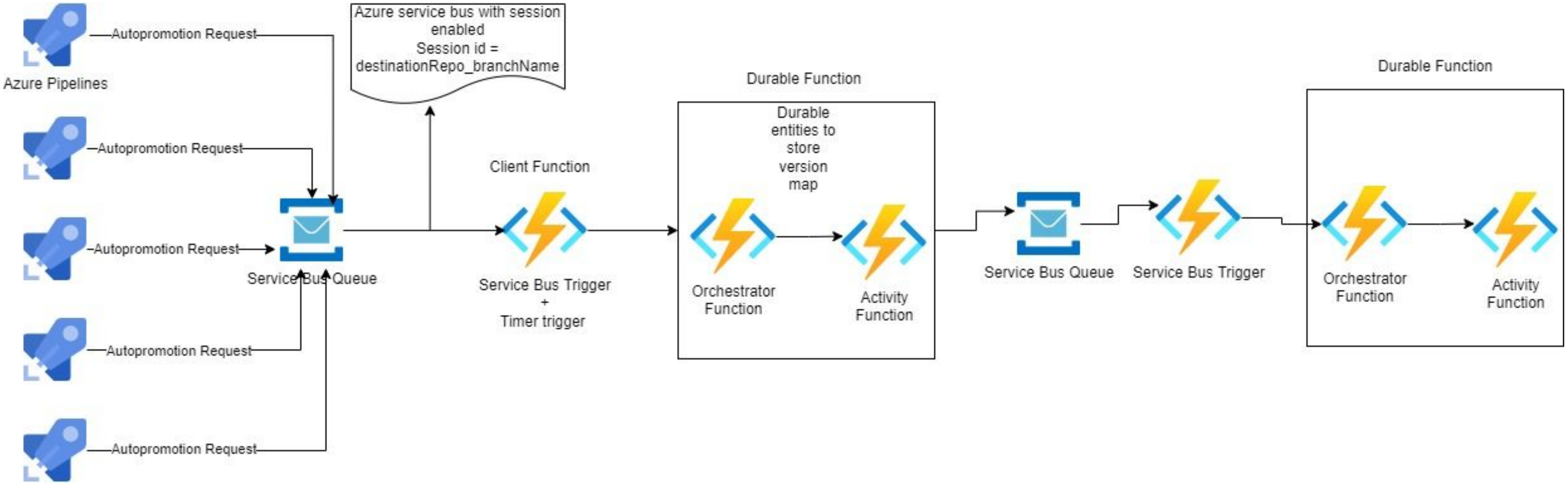
Function as a Service (FaaS): is a cloud computing model that allows developers to run code in response to events without managing servers, enabling scalable and cost-effective application development.

Event-Driven Execution , Cost Efficiency ,Scalability

Azure Functions

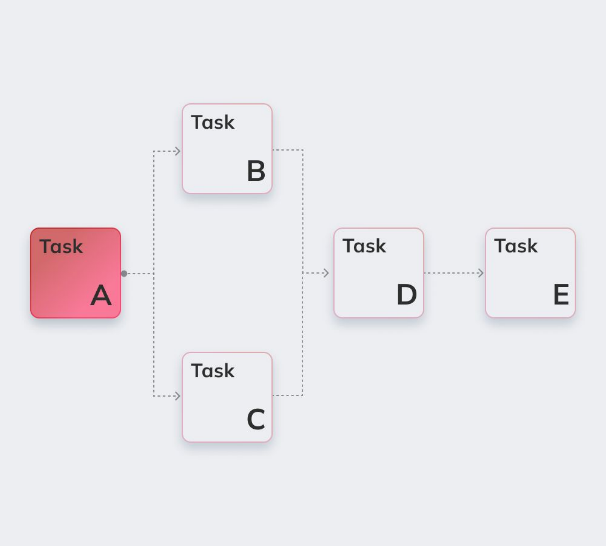
Google
Cloud
Functions

FaaS nightmares



Pipeline orchestration

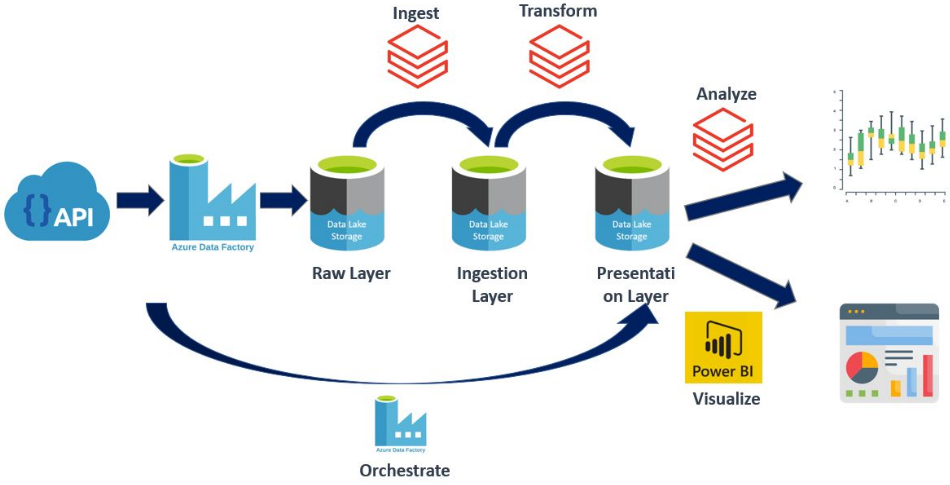
Apache Airflow's orchestration enables the automated scheduling, execution, and monitoring of complex workflows through a user-defined Directed Acyclic Graph (DAG) structure.



Data Layers

A data layer is a structured framework that collects and organizes data from various sources within a system, such as a website or application. It acts as an intermediary, ensuring that data is consistently captured and made accessible for analytics and other tools, facilitating better data management and integration across different platforms.

Exemple :



05

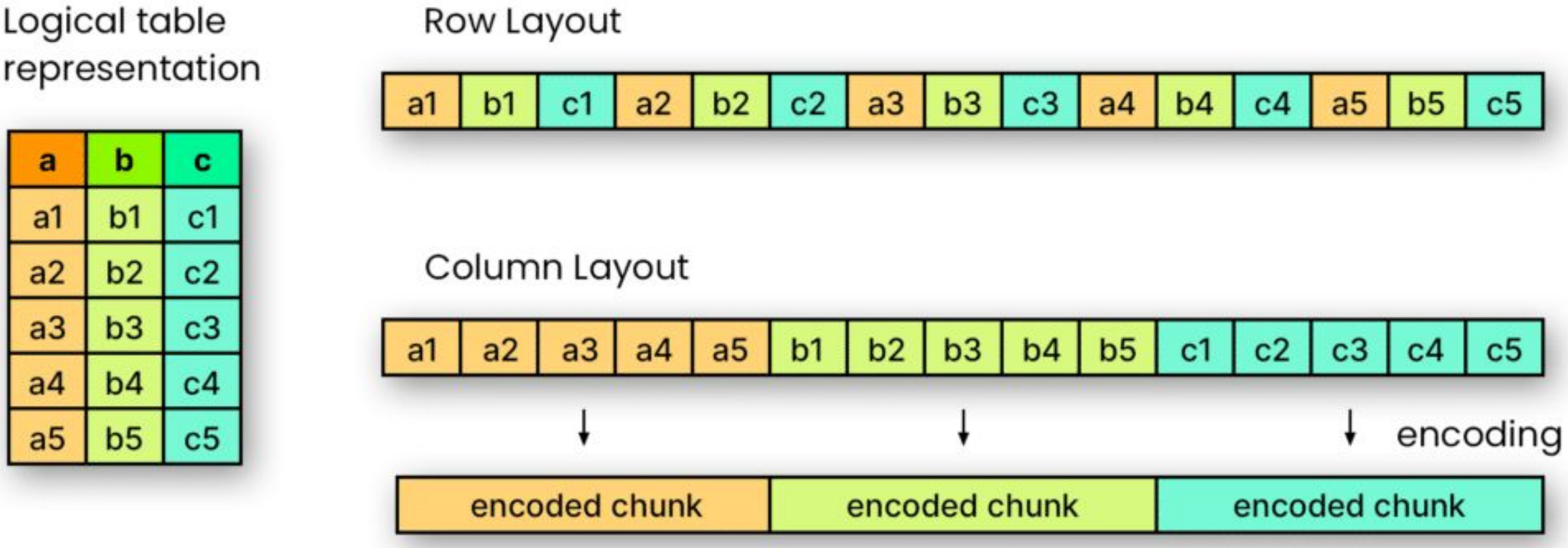
Stockage colonnes

Column-oriented storage

Column-oriented storage is a data management technique that organizes and stores data by columns rather than rows. This approach enhances query performance and data compression, making it particularly effective for analytical workloads and big data applications, where accessing specific columns quickly is crucial.

Index columnstore

Column-oriented storage



Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row Layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5

Column Layout

a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5

encoded chunk	encoded chunk	encoded chunk

Column-oriented storage

	Column-oriented storage	Row-oriented storage
Pros	- Optimized for Analytics - Efficient Compression - Scalability	- Efficient for Transactions - Simplicity
Cons	- Slower for Transactions - Complex Schema Design	- Less Efficient for Analytics - Limited Compression

06

Hadoop , MapReduce & Spark

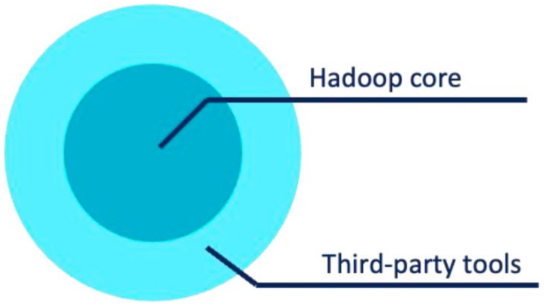
What is Hadoop ?

- Hadoop is a collection of open source projects providing a **distributed** and **scalable** framework for Big Data **storage** and **processing**
  - Hadoop is mostly written in Java

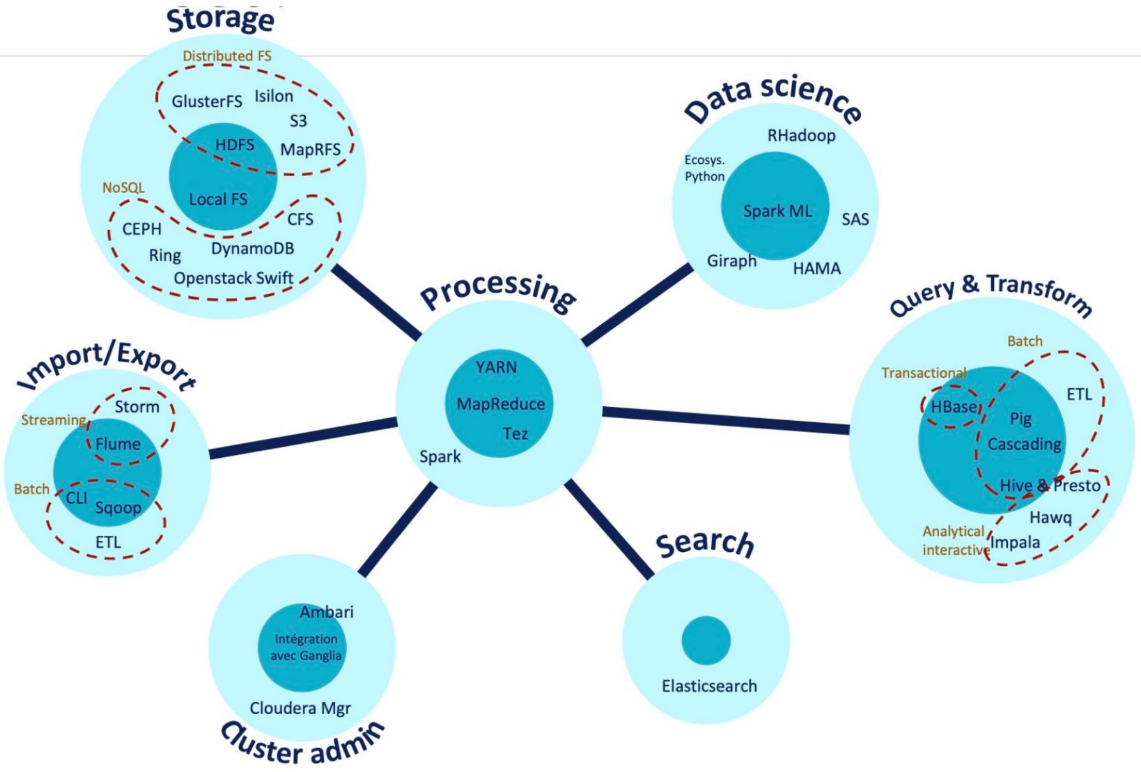
- The project began in 2006 and has been managed by the Apache Foundation since 2009

- It was the most popular Big Data solution on the market, with hundreds of users around the world
  - The most prominent users were also big contributors: Yahoo!, Facebook, Ebay

- It is the basis of a thriving software ecosystem

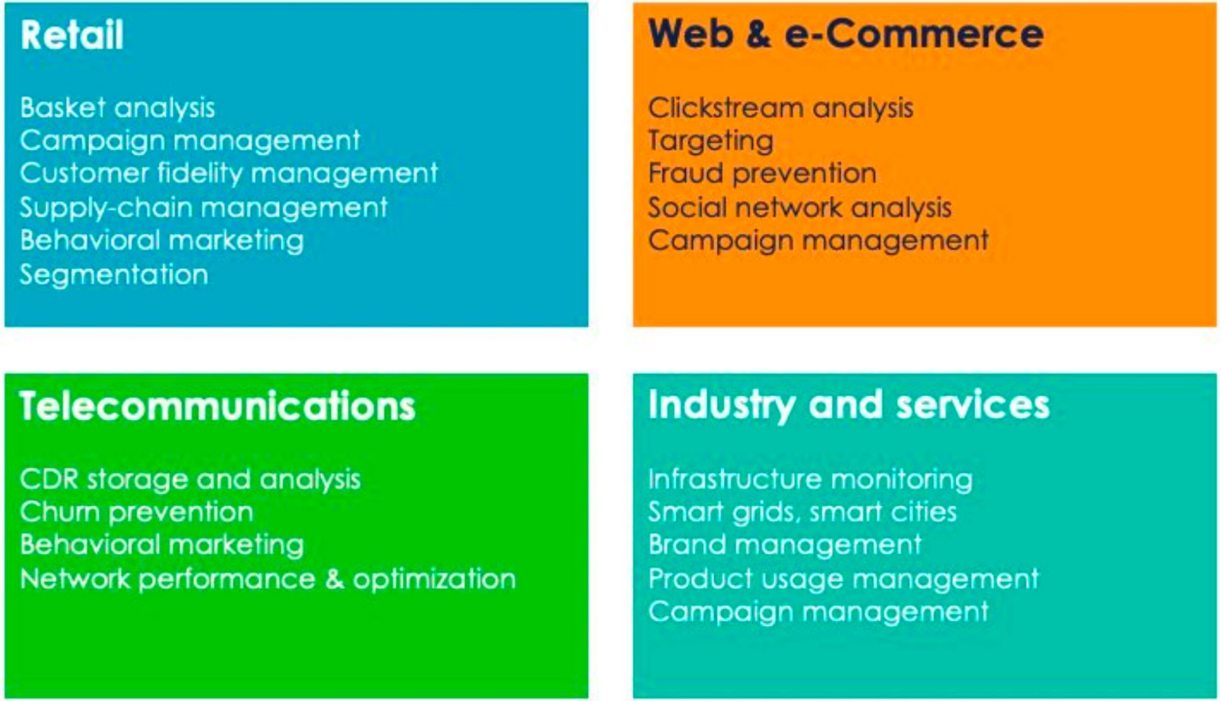


The Hadoop ecosystem



What can one do with Hadoop

Pretty much anything ;-) Here are some typical use cases



Retail

Basket analysis
Campaign management
Customer fidelity management
Supply-chain management
Behavioral marketing
Segmentation

Web & e-Commerce

Clickstream analysis
Targeting
Fraud prevention
Social network analysis
Campaign management

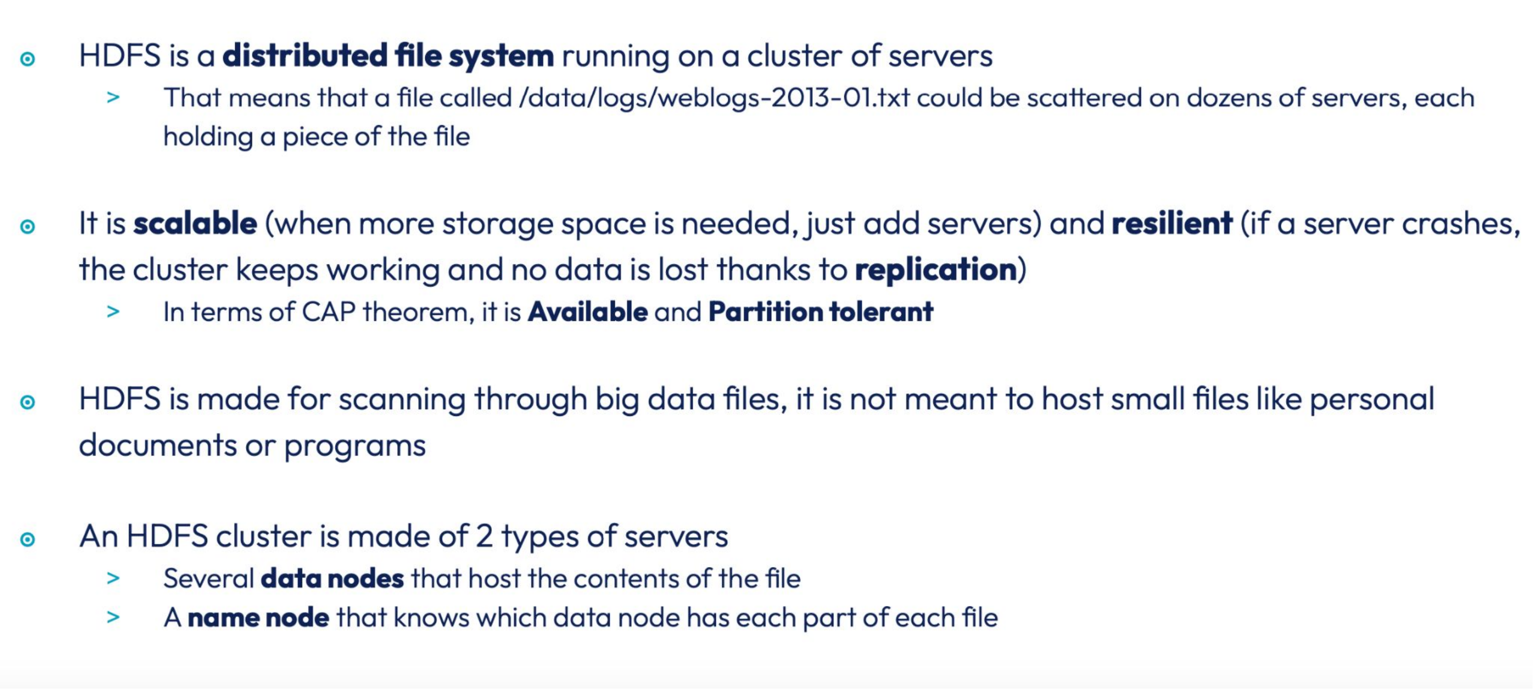
Telecommunications

CDR storage and analysis
Churn prevention
Behavioral marketing
Network performance & optimization

Industry and services

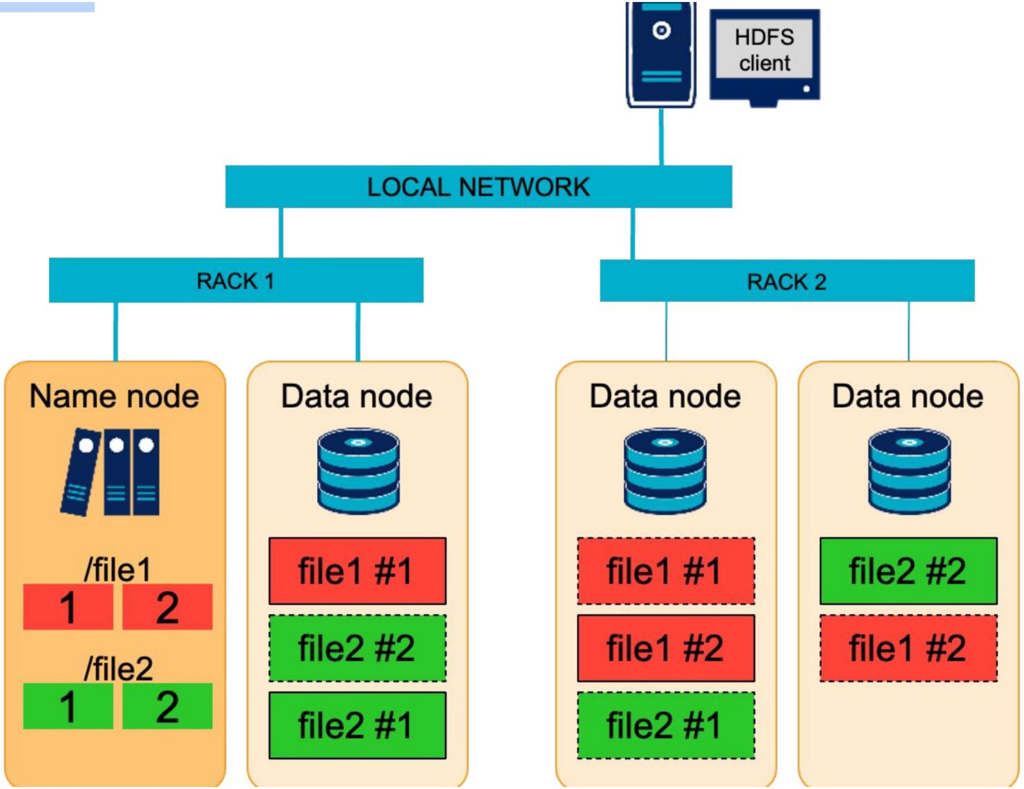
Infrastructure monitoring
Smart grids, smart cities
Brand management
Product usage management
Campaign management

HDFS - the Hadoop Distributed File System



- HDFS is a **distributed file system** running on a cluster of servers
  - That means that a file called /data/logs/weblogs-2013-01.txt could be scattered on dozens of servers, each holding a piece of the file
- It is **scalable** (when more storage space is needed, just add servers) and **resilient** (if a server crashes, the cluster keeps working and no data is lost thanks to **replication**)
  - In terms of CAP theorem, it is **Available** and **Partition tolerant**
- HDFS is made for scanning through big data files, it is not meant to host small files like personal documents or programs
- An HDFS cluster is made of 2 types of servers
  - Several **data nodes** that host the contents of the file
  - A **name node** that knows which data node has each part of each file

An HDFS Cluster

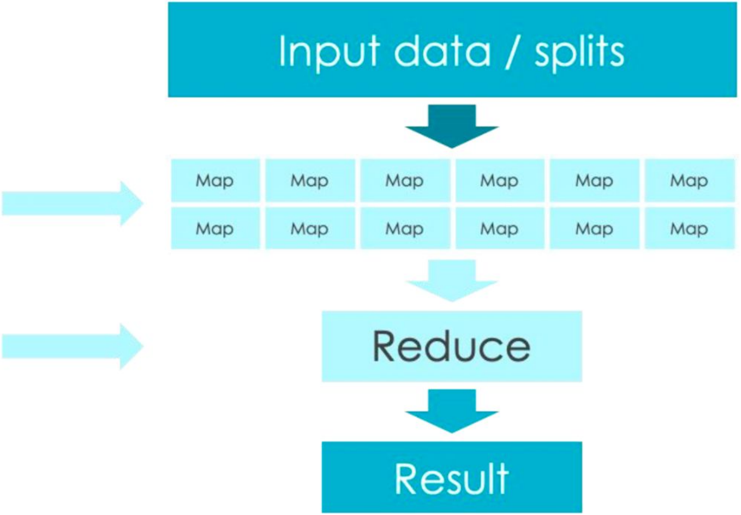


MapReduce for distributed data processing

- MapReduce is
  - An **algorithm** for running in parallel programs that analyze data
  - A **framework** for developing such programs

- Its works by subdividing the data to process in smaller chunks called **splits**. Splits are processed in parallel during the **Map phase**. At the end the **Reduce phase** combines the results of the independants maps

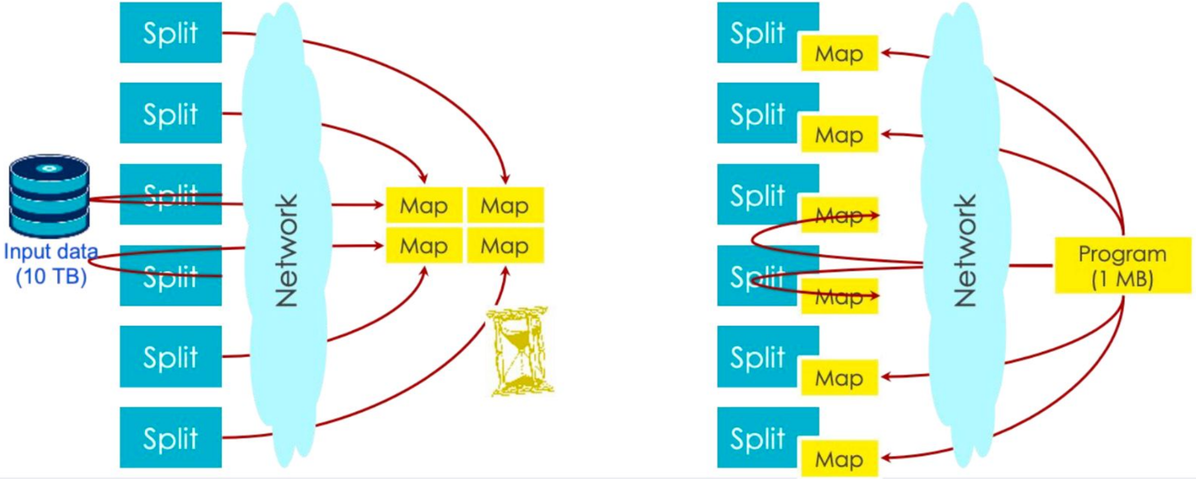
The programmer writes a pair of programs: a **mapper** and a **reducer**. Hadoop takes care of the rest



MapReduce - The Map phase

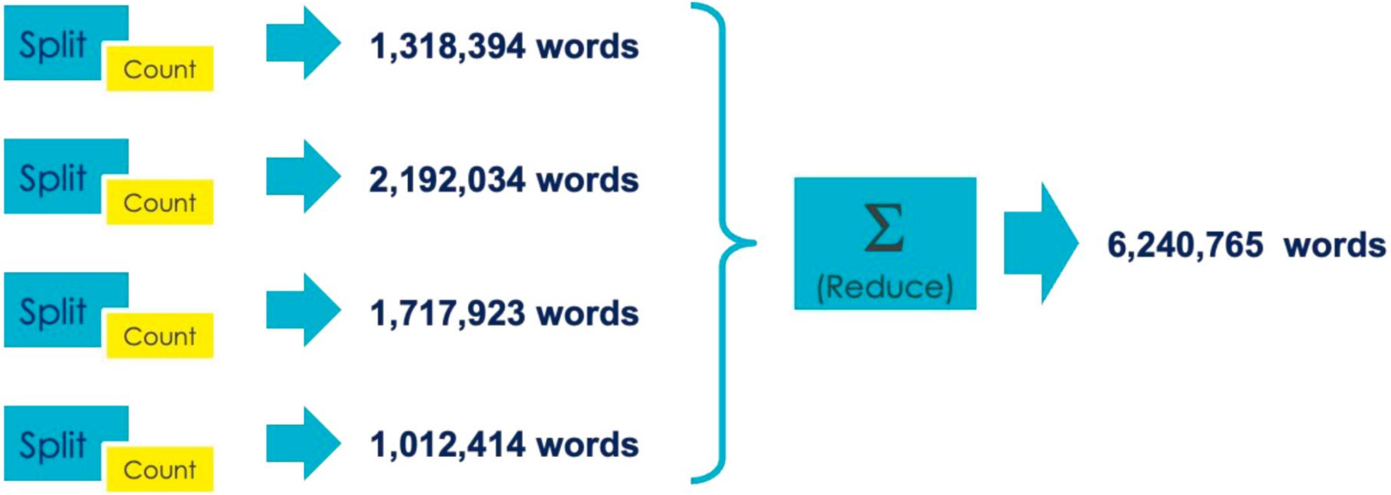
- In Big Data scenarios, reading data from disk and transfering splits back and forth between servers is costly

- It is much faster to bring small programs (binaries, scripts, ...) close to the data; this principle is called **data locality**
  - **Q**: When the data resides on HDFS, how does MapReduce know where each split belongs?



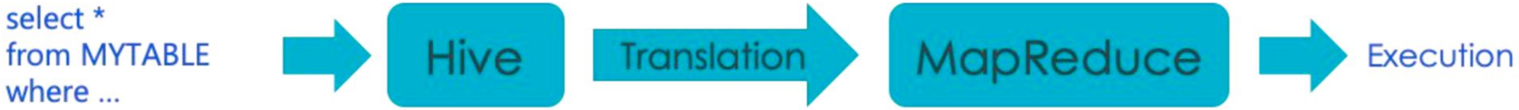
MapReduce (cont’d) - The Reduce phase

- All mappers have worked independently of each other, producing many fragments of result data
- The **Reduce phase** consists in gathering all those fragments together, and producing the final results
- **Example**: counting words in a set of documents



MapReduce (cont'd) - Jobs

- A Map and Reduce sequence that executes on a cluster is called a job
- There is a special service on the cluster, called the **job tracker**, that waits for job submissions
- On each data node, there is a **task tracker** that awaits orders from the job tracker. Such orders include launching a mapper, launching a reducer, or reporting on job execution progress
- Jobs are submitted directly by a user, or by other tools that hide the complexity of MapReduce
    - Example: Hive is a SQL frontend to MapReduce. Hive translates SQL to MapReduce code

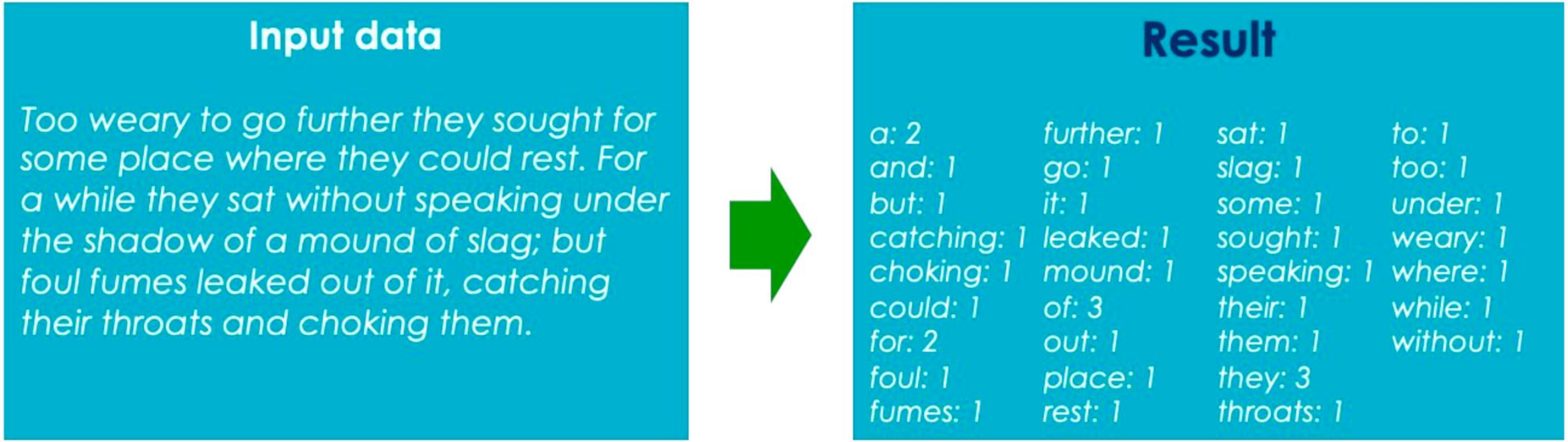


A full example

Counting the number of occurrences of words in documents

This is MapReduce’s “Hello World” example

What we want to achieve (only with many big files as input):



Input data

Too weary to go further they sought for some place where they could rest. For a while they sat without speaking under the shadow of a mound of slag; but foul fumes leaked out of it, catching their throats and choking them.

Result

a: 2
and: 1
but: 1
catching: 1
choking: 1
could: 1
for: 2
foul: 1
fumes: 1
further: 1
go: 1
it: 1
leaked: 1
mound: 1
of: 3
out: 1
place: 1
rest: 1
sat: 1
slag: 1
some: 1
sought: 1
speaking: 1
their: 1
them: 1
they: 3
throats: 1
to: 1
too: 1
under: 1
weary: 1
where: 1
while: 1
without: 1

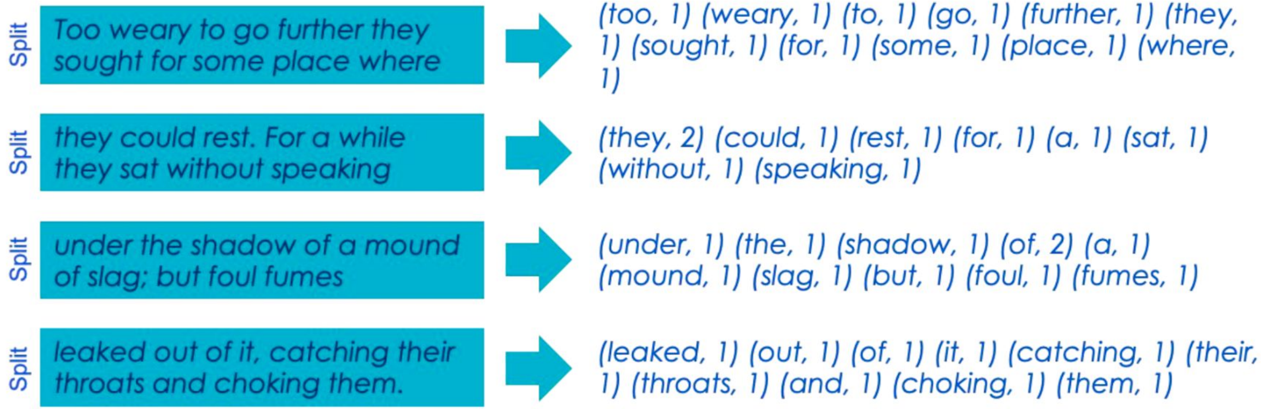
We need to write a mapper and a reducer programs, and submit them as a job

A full example - the Mapper

- Remember, each mapper will receive a portion of the input data (a split), and will work **independently** of the others

MAPPER ALGORITHM

```
Split input_fragment into words, ignoring case and punctuation
For each word in input_fragment
        Set N to the number of occurrences of word in the fragment
        Emit a pair (word, N)
```



Split: *Too weary to go further they sought for some place where* → *(too, 1) (weary, 1) (to, 1) (go, 1) (further, 1) (they, 1) (sought, 1) (for, 1) (some, 1) (place, 1) (where, 1)*

Split: *they could rest. For a while they sat without speaking* → *(they, 2) (could, 1) (rest, 1) (for, 1) (a, 1) (sat, 1) (without, 1) (speaking, 1)*

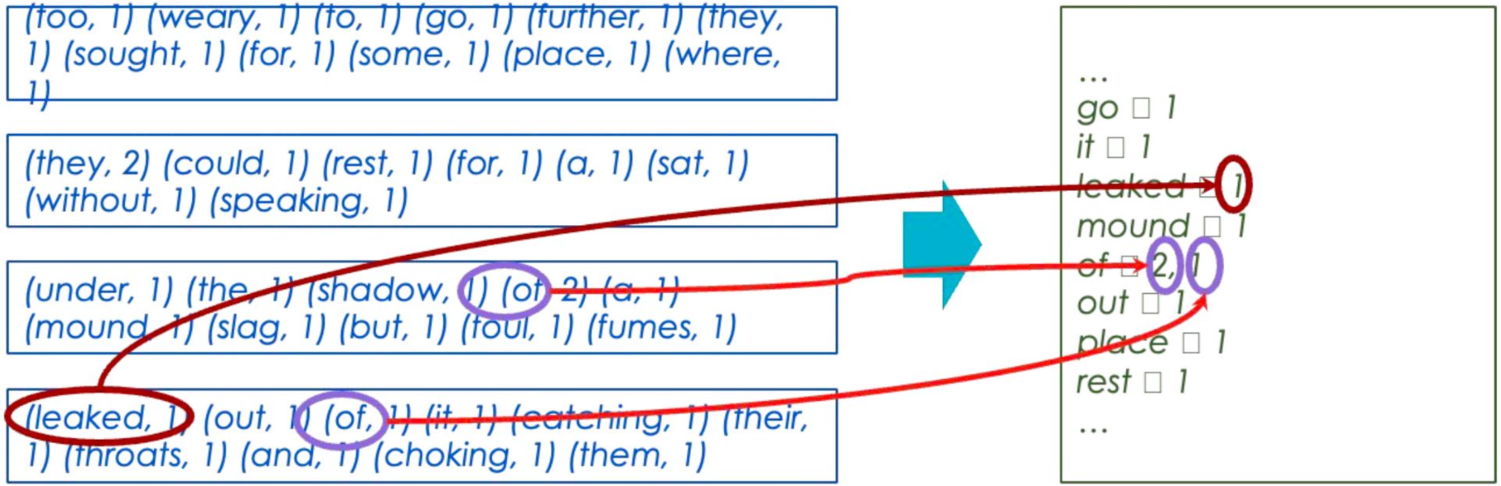
Split: *under the shadow of a mound of slag; but foul fumes* → *(under, 1) (the, 1) (shadow, 1) (of, 2) (a, 1) (mound, 1) (slag, 1) (but, 1) (foul, 1) (fumes, 1)*

Split: *leaked out of it, catching their throats and choking them.* → *(leaked, 1) (out, 1) (of, 1) (it, 1) (catching, 1) (their, 1) (throats, 1) (and, 1) (choking, 1) (them, 1)*

A full example - the Shuffle & Sort

The **shuffle & sort** is part of the MapReduce algorithm, and is performed behind the scenes by Hadoop. The programmer doesn't have to write a program for it

The purpose of this step is to group together the (word, N) pairs emitted by the mappers, so the reducer have all information belonging to a given word in one place. There is no computation involved in this step
As a nice by-product, the resulting data is sorted on the keys (here, on words)



(too, 1) (weary, 1) (to, 1) (go, 1) (further, 1) (they, 1) (sought, 1) (for, 1) (some, 1) (place, 1) (where, 1)

(they, 2) (could, 1) (rest, 1) (for, 1) (a, 1) (sat, 1) (without, 1) (speaking, 1)

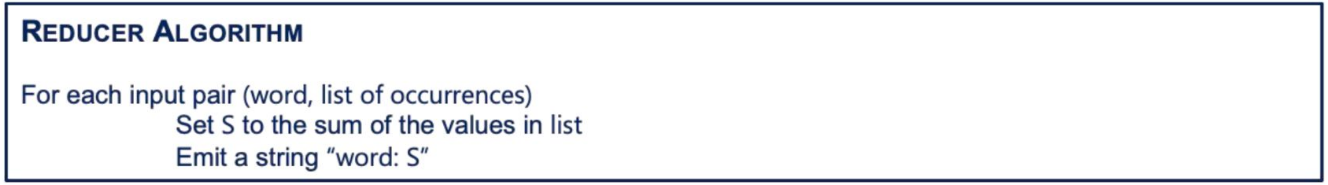
(under, 1) (the, 1) (shadow, 1) (of, 2) (a, 1) (mound, 1) (slag, 1) (but, 1) (foul, 1) (fumes, 1)

(leaked, 1) (out, 1) (of, 1) (it, 1) (catching, 1) (their, 1) (throats, 1) (and, 1) (choking, 1) (them, 1)

…
go □ 1
it □ 1
leaked □ 1
mound □ 1
of □ 2, 1
out □ 1
place □ 1
rest □ 1
…

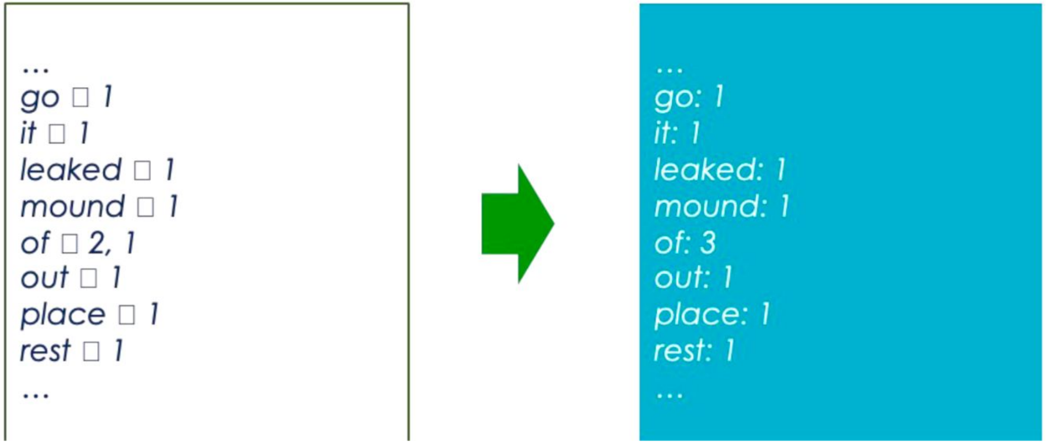
A full example - the Reducer

Now the reducer, with the product of shuffle & sort, has all the information needed to count the occurrences of each word



REDUCER ALGORITHM

```
For each input pair (word, list of occurrences)
        Set S to the sum of the values in list
        Emit a string "word: S"
```



...
go □ 1
it □ 1
leaked □ 1
mound □ 1
of □ 2, 1
out □ 1
place □ 1
rest □ 1
...

...
go: 1
it: 1
leaked: 1
mound: 1
of: 3
out: 1
place: 1
rest: 1
...

There we are!

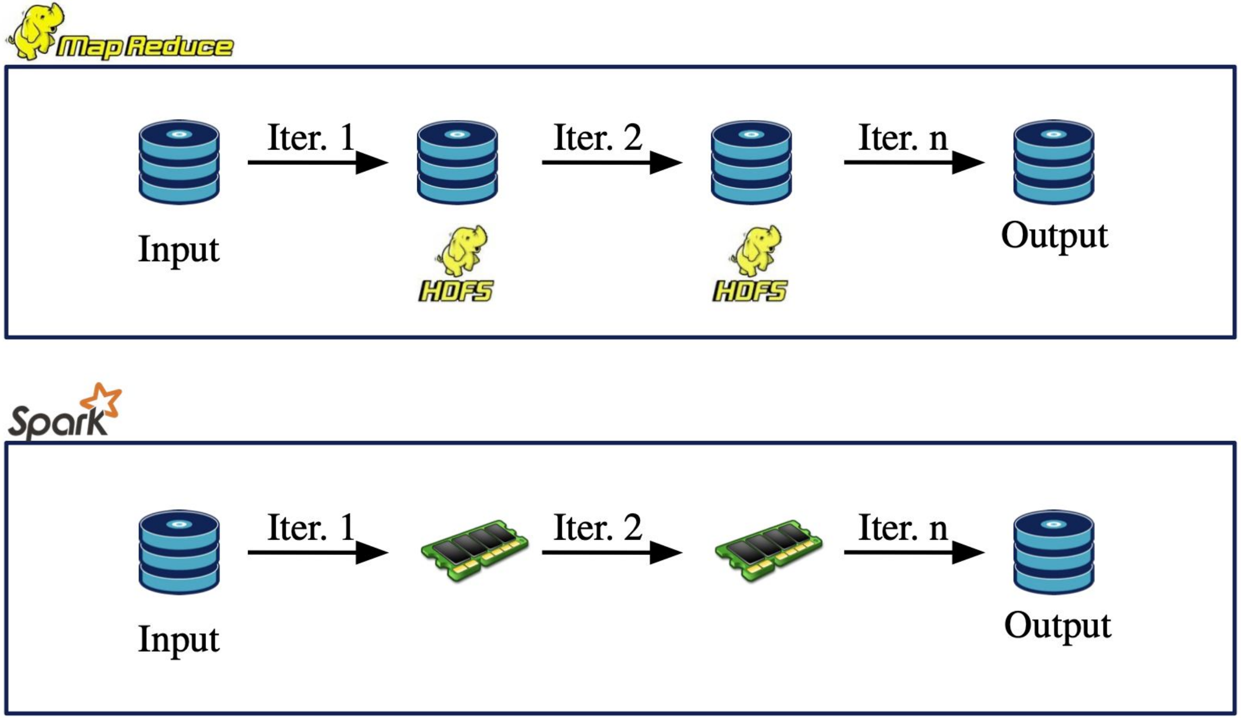
What is Spark ?

Apache Spark is an open-source, **distributed processing system** used for big data workloads.
It utilizes **in-memory caching** and optimized query execution for fast queries against data of any size. Simply put, Spark is a fast and general engine for large-scale data processing.

The fast part means that it’s faster than previous approaches to work with Big Data like classical MapReduce. The secret for being faster is that Spark runs on memory (RAM), and that makes the processing much faster than on disk drives.

The general part means that it can be used for multiple things like running distributed SQL, creating data pipelines, ingesting data into a database, running Machine Learning algorithms, working with graphs or data streams, and much more.

Spark


Map Reduce
Input
Iter. 1
HDFS
Iter. 2
HDFS
Iter. n
Output
Spark
Input
Iter. 1
Iter. 2
Iter. n
Output

Most commons components (details)

⊙ Apache Spark Core – Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built upon. It provides in-memory computing and referencing datasets in external storage systems.

⊙ Spark SQL – Spark SQL is Apache Spark's module for working with structured data. The interfaces offered by Spark SQL provides Spark with more information about the structure of both the data and the computation being performed.

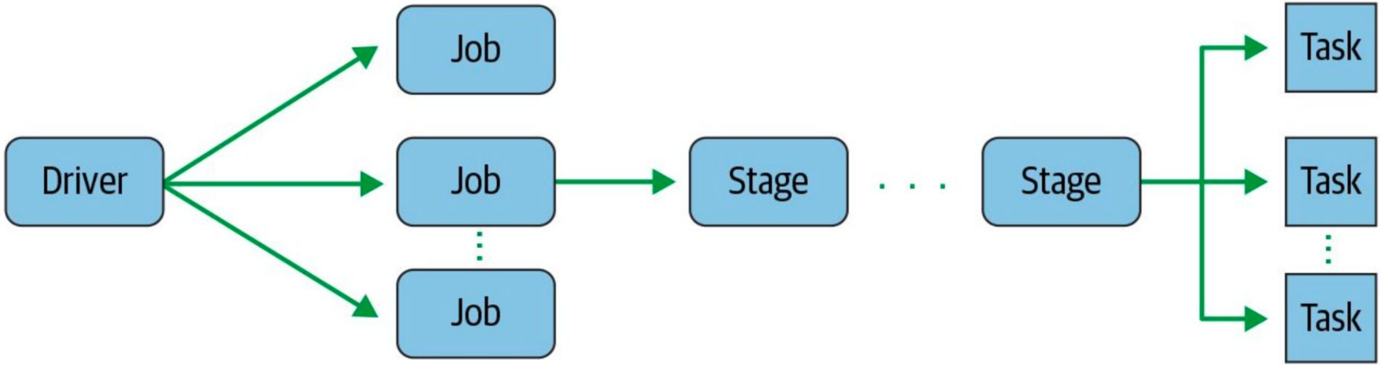
⊙ Spark Streaming – This component allows Spark to process real-time streaming data. Data can be ingested from many sources like Kafka, Flume, and HDFS (Hadoop Distributed File System). Then the data can be processed using complex algorithms and pushed out to file systems, databases, and live dashboards.

Others components (details)

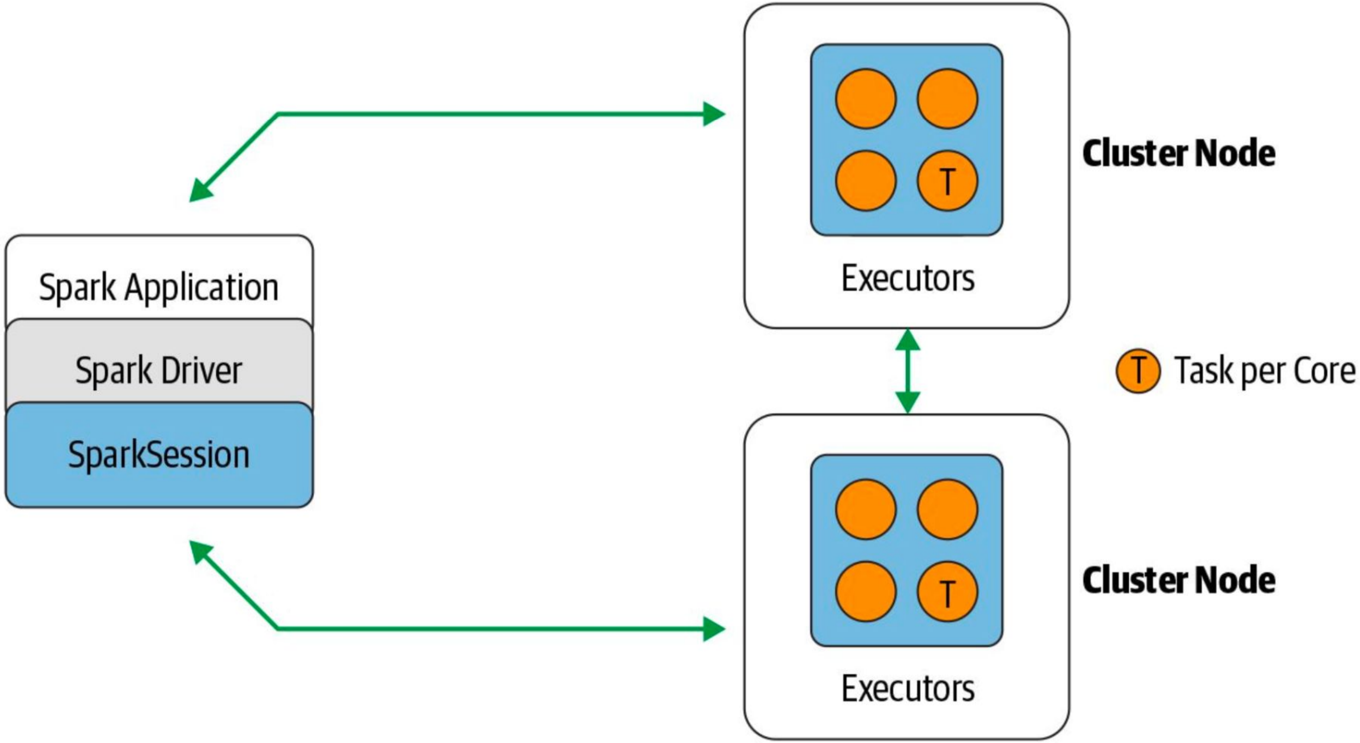
⊙ MLlib (Machine Learning Library) – Apache Spark is equipped with a rich library known as MLlib. This library contains a wide array of machine learning algorithms- classification, regression, clustering, and collaborative filtering. It also includes other tools for constructing, evaluating, and tuning ML Pipelines. All these functionalities help Spark scale out across a cluster.

⊙ GraphX – Spark also comes with a library to manipulate graph databases and perform computations called GraphX. GraphX unifies ETL (Extract, Transform, and Load) process, exploratory analysis, and iterative graph computation within a single system.

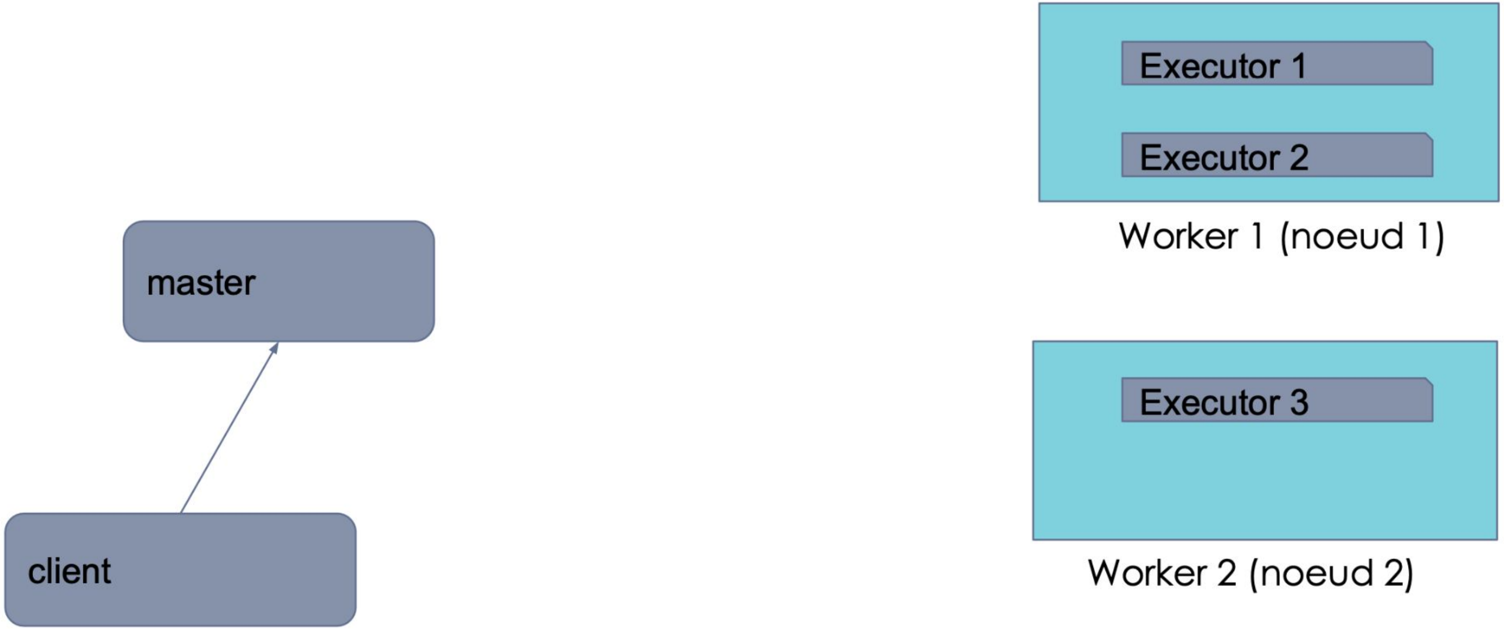
Spark Execution



Spark Cluster



Spark Cluster



master

client

Executor 1

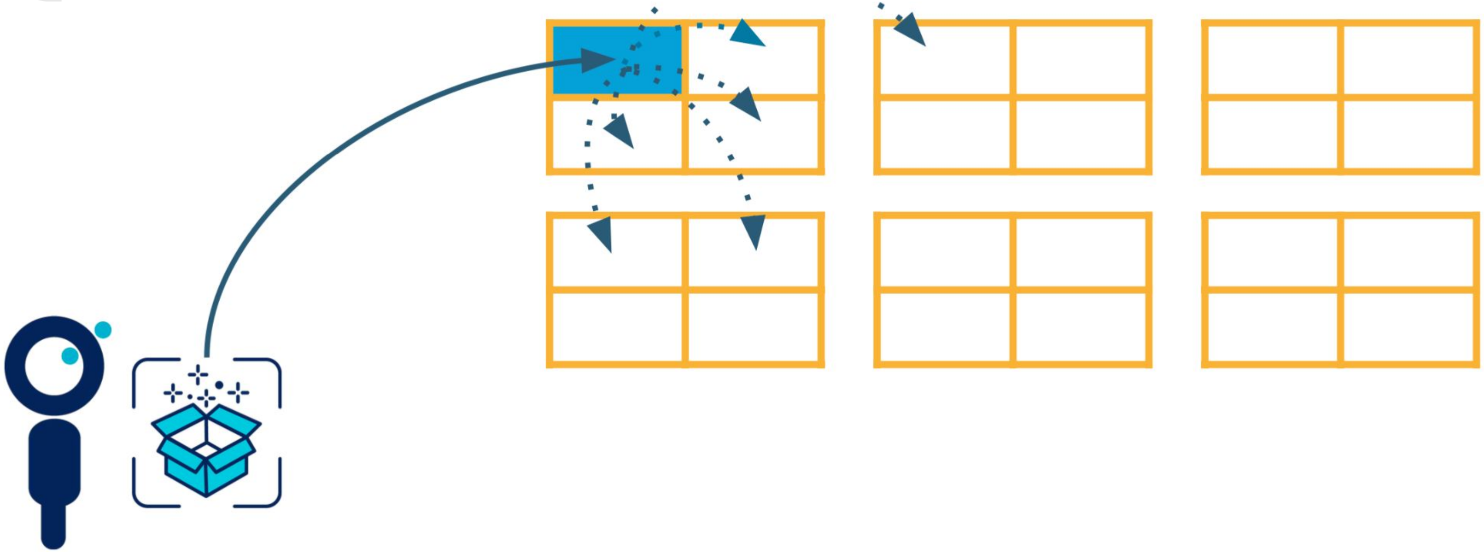
Executor 2

Worker 1 (noeud 1)

Executor 3

Worker 2 (noeud 2)

Architecture



- Spark permet de paralléliser les calculs et offre une couche d’abstraction
  - Spark Driver : coordonne le traitement et le répartit entre les Executors
  - Spark Executor : unité de calcul (~1 coeur & un processus java)
  - Spark worker : noeud du cluster (peut contenir plusieurs executors)

Concepts de base

⊙ Spark application : un programme utilisateur construit en utilisant les apis de spark. Ça consiste en un programme driver et des executors sur un cluster.

⊙ SparkSession: un objet qui fournit un point d'entrée à l'interaction avec les fonctionnalités de spark et permet la programmation de spark avec ses Apis. Rq: dans un shell spark interactif, le driver spark instancie automatiquement cet objet. Dans une application spark, il faudra le créer manuellement.

⊙ job : un traitement parallèle consistant en plusieurs "tasks" qui sont déclenchées suite à une spark action.

⊙ stage : chaque job est réparti en plusieurs stages dépendants les uns des autres.

⊙ task : une unité de traitement exécutée sur un spark executor.

Transformations, actions, and lazy evaluation

- Les traitements distribués de spark sont de 2 types :
  > transformations : transforme un dataframe en un autre dataframe sans transformer l’original (immutability)
  > actions: déclenche l’évaluation d’un ensemble de transformation