
BIG
DATA

Big Data: Pourquoi ?

Données

Frameworks

Plateformes

Analyse

Intelligence artificielle

Big Data: Contexte

Nouvelles solutions « open source »

Nouveaux challenges autours de la « data »

Nouveaux besoins IT :

- Data engineer
- Data analyste
- Data scientist
- DevOps engineer

Big Data: Problématiques et challenges !

- Des données hétérogènes : structurées, non structures
- Différentes sources de données : base de données, fichiers, streaming
- Données non fiables et manquantes
- Sécurisation des données : droit sur les données, droit d'accès, intégrité des données, données anonymisés...

- Prétraitement des données :
  - Extraction, chargement, transformation, visualisation → Mieux comprendre ses données

- Analyse prédictive : prédiction, classification, clustering

  → Trouver des patterns dans des données métier

  → Automatiser des process : Détection d'anomalie..

  → Créer des nouveaux modèles pour répondre aux nouveaux besoins

Big Data: Définition

- « Big data » comme le ‘small data’, mais en plus grand et plus large
- Avoir des données plus grandes nécessite des approches différentes de résolution : technologies et architectures
- De nouveaux paradigmes pour une résolution des problèmes d’une manière plus efficace
- « Big data » génère de la valeur à partir du stockage et du calcul de grandes quantités de données numériques
- « Big data » c’est l’analyse et le calcul d’une sur des données massives

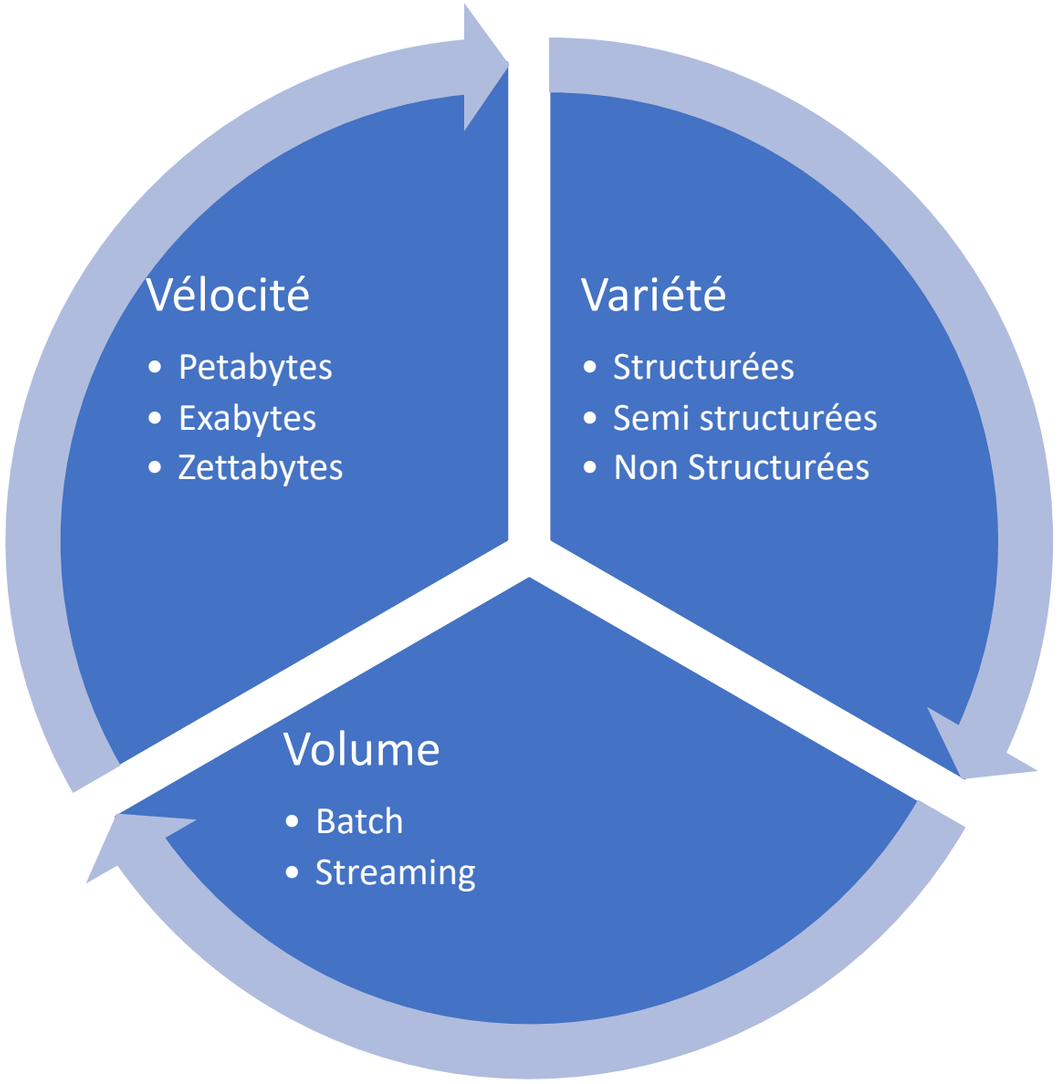
Big Data: Objectif

Comme d’autres paradigme et technologies d’informations, le « big data » a pour but:

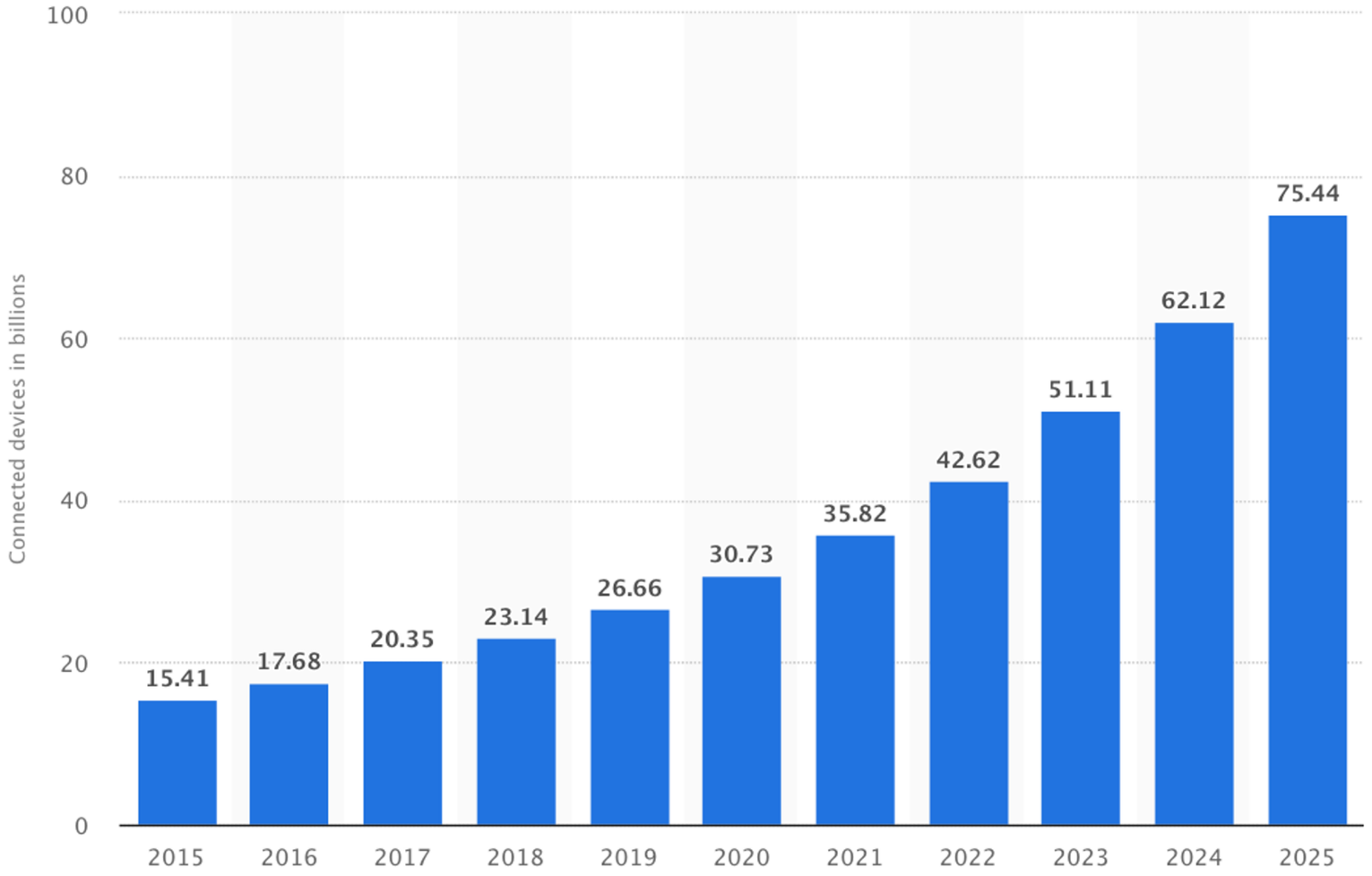
- Réduire les coûts de la gestion des données: sauvegarde et traitement
- Améliorer le temps d’exécution nécessaire au traitement
- Trouver et identifier de nouveaux services


data

Big Data: 3V

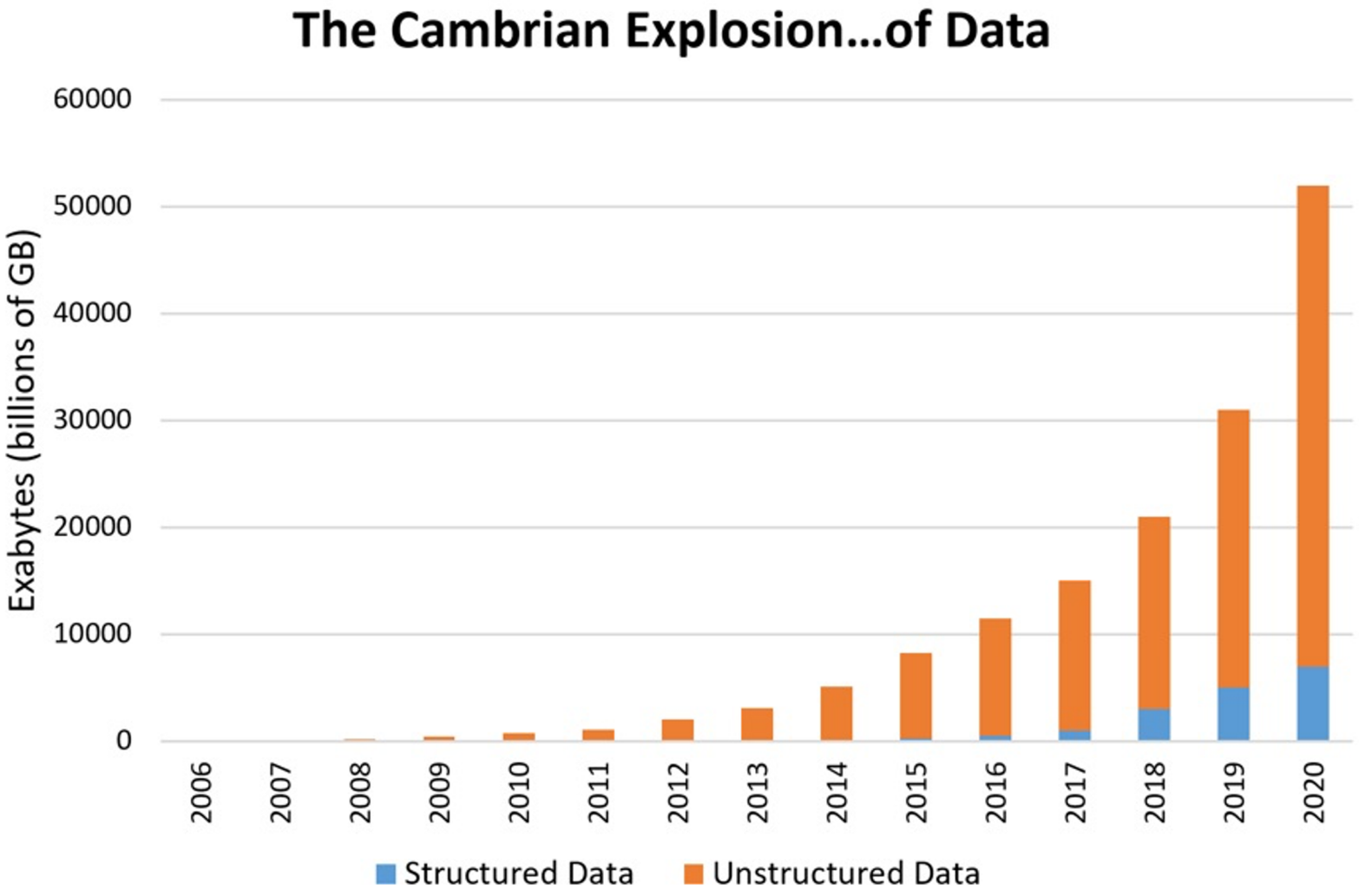

Vélocité
• Petabytes
• Exabytes
• Zettabytes
Variété
• Structurées
• Semi structurées
• Non Structurées
Volume
• Batch
• Streaming

Big Data: Evolution des objets connectés



Source: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

Big Data: Evolution du volume de données



Source : https://www.eetimes.com/author.asp?section_id=36&doc_id=1330462

Big Data: Volume

- Coût moins élevé du stockage: disques moins chères, solution de stockage open source : stockage objet, NFS..
- Coût de la maintenance de l'infrastructure moins élevé: automatisation de déploiement et du maintien en condition opérationnelle : sauvegarde
- Accessibilité des appareils numériques : plus de photos et vidéos, meilleure qualité de photos et des vidéos

  → plus de stockage
- Émergence des objets connectés : montres connectés, domotiques, voitures connectées, circuit RFID, smatphones

Big Data: Vélocité

- Evolution exponentielle de la quantité des données

  (utilisateurs facebook uplod plus de 900 millions de photos par jour)
- Flux de données issue de navigations web: user tracking et moteur de recommondation
- Communication et échanges machine à machines, entres des milliards d’objets
- L’infrastructure IT et les capteurs génèrent des données massives en temps réel
- Les jeux en ligne et les systèmes supportant des millions de connexions parallèles

Big Data: Variété

- Big data, il ne s’agit pas que de données structurées. Big data est aussi des données géospatiales, données 3D, vidéos, logs, streaming...

- Plusieurs types de données
  - données structurées : base de données, csv, parquet
  - données semi-structurées : xml, json, log
  - données non-structurées : documents, vidéos..


Plateforme

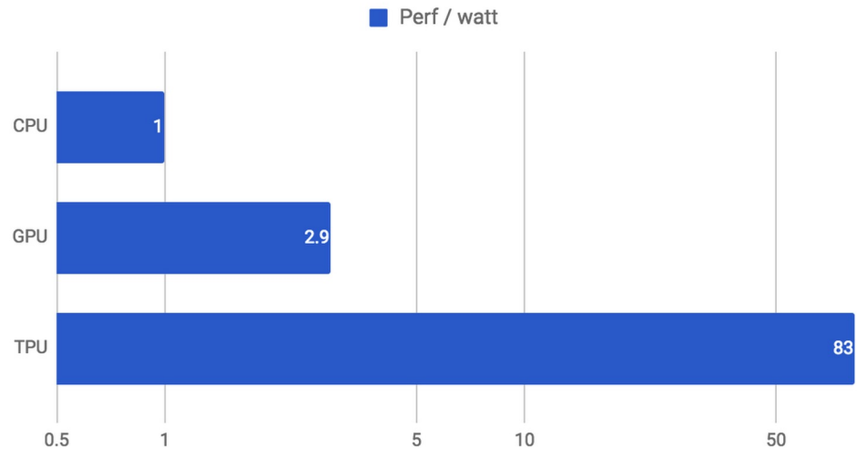
Big Data: Computing

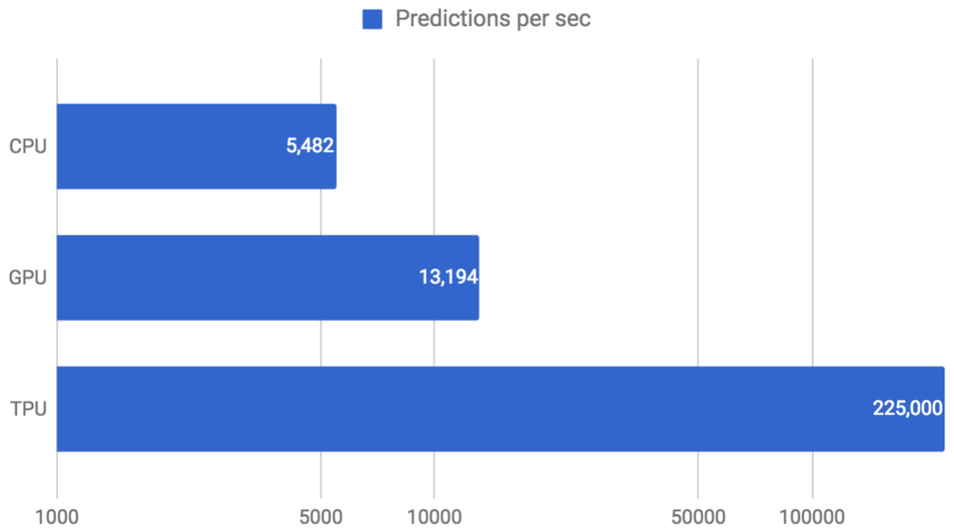
CPU

GPU

TPU

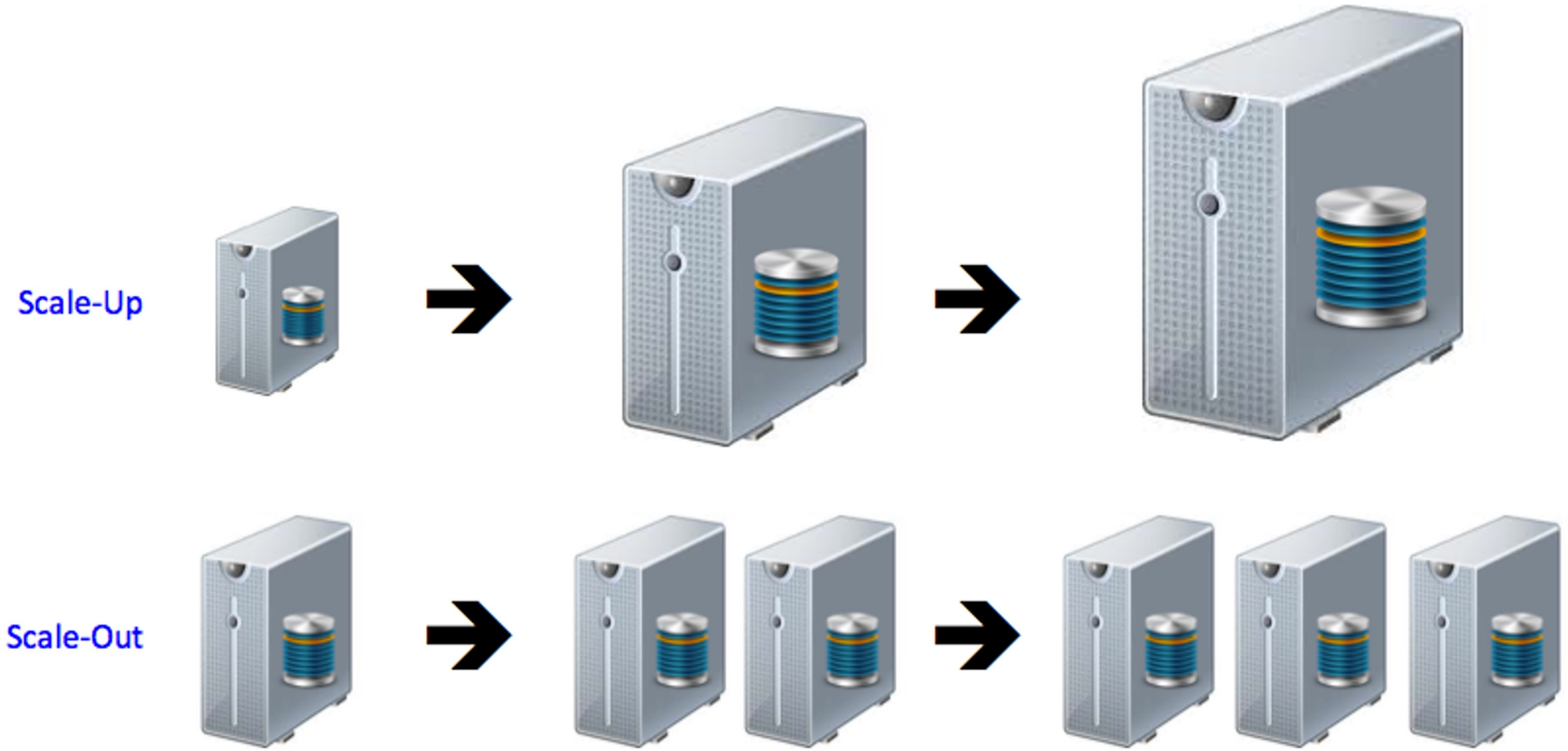
Big Data: Computing



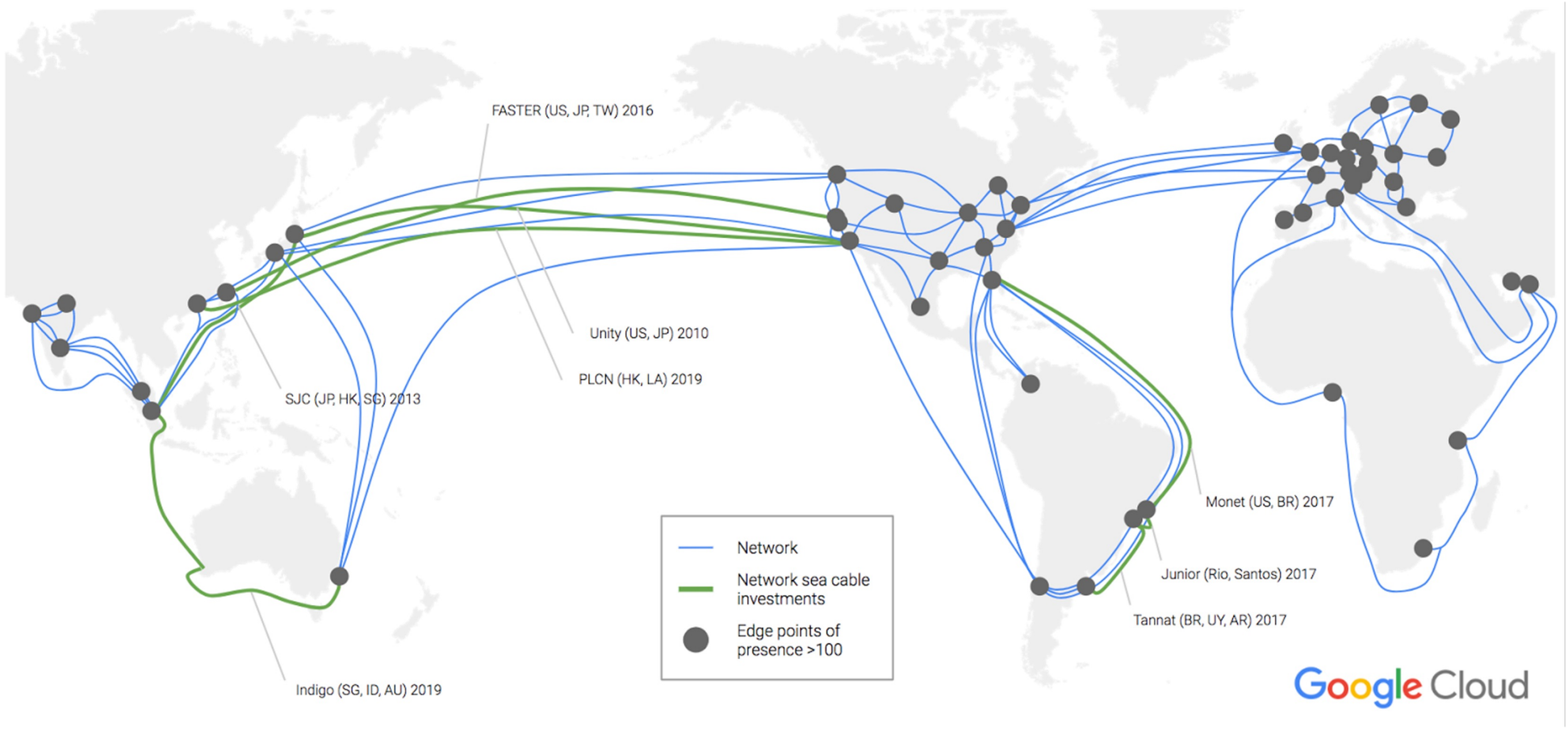


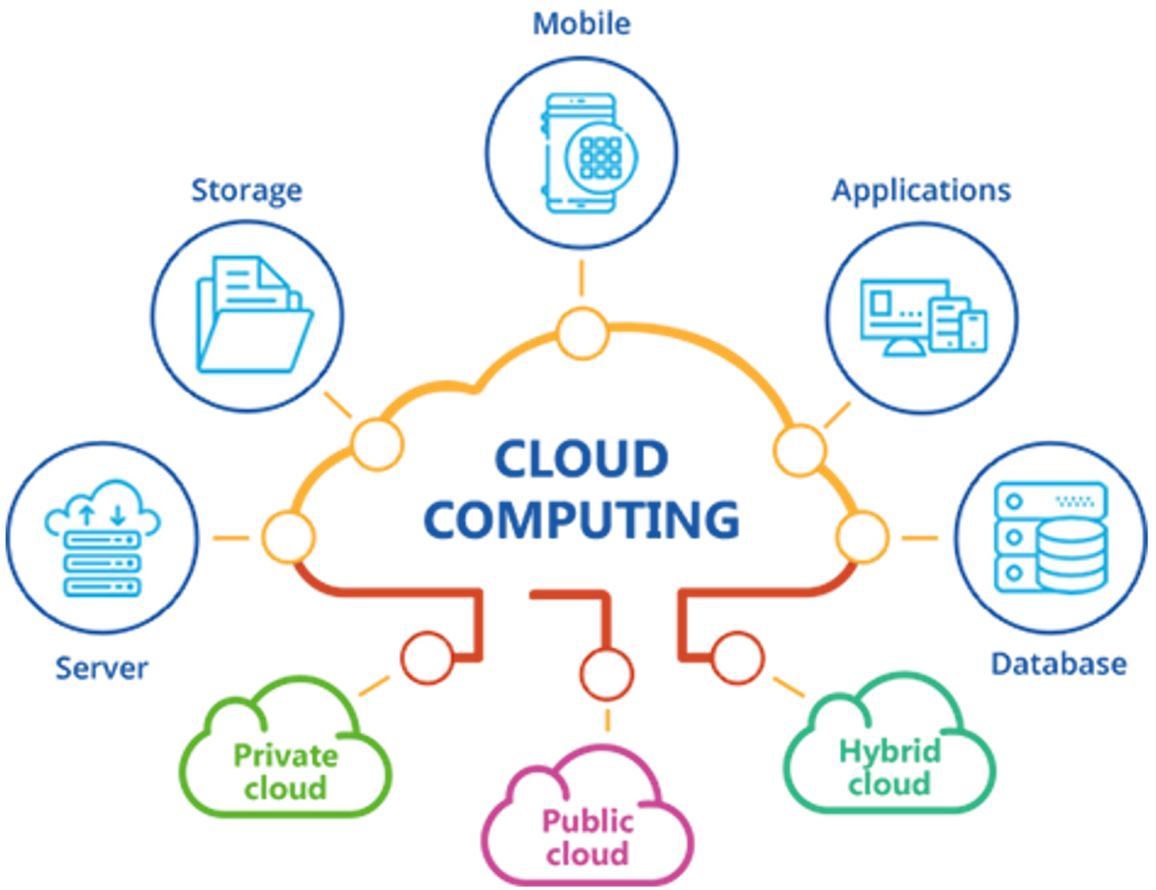
Source: https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

Big Data: Scale



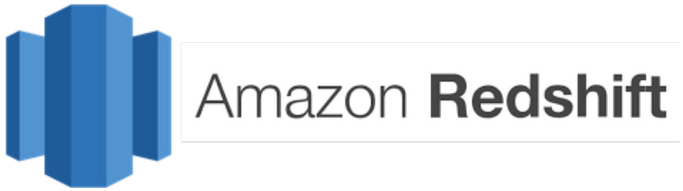
Big Data: Réseau



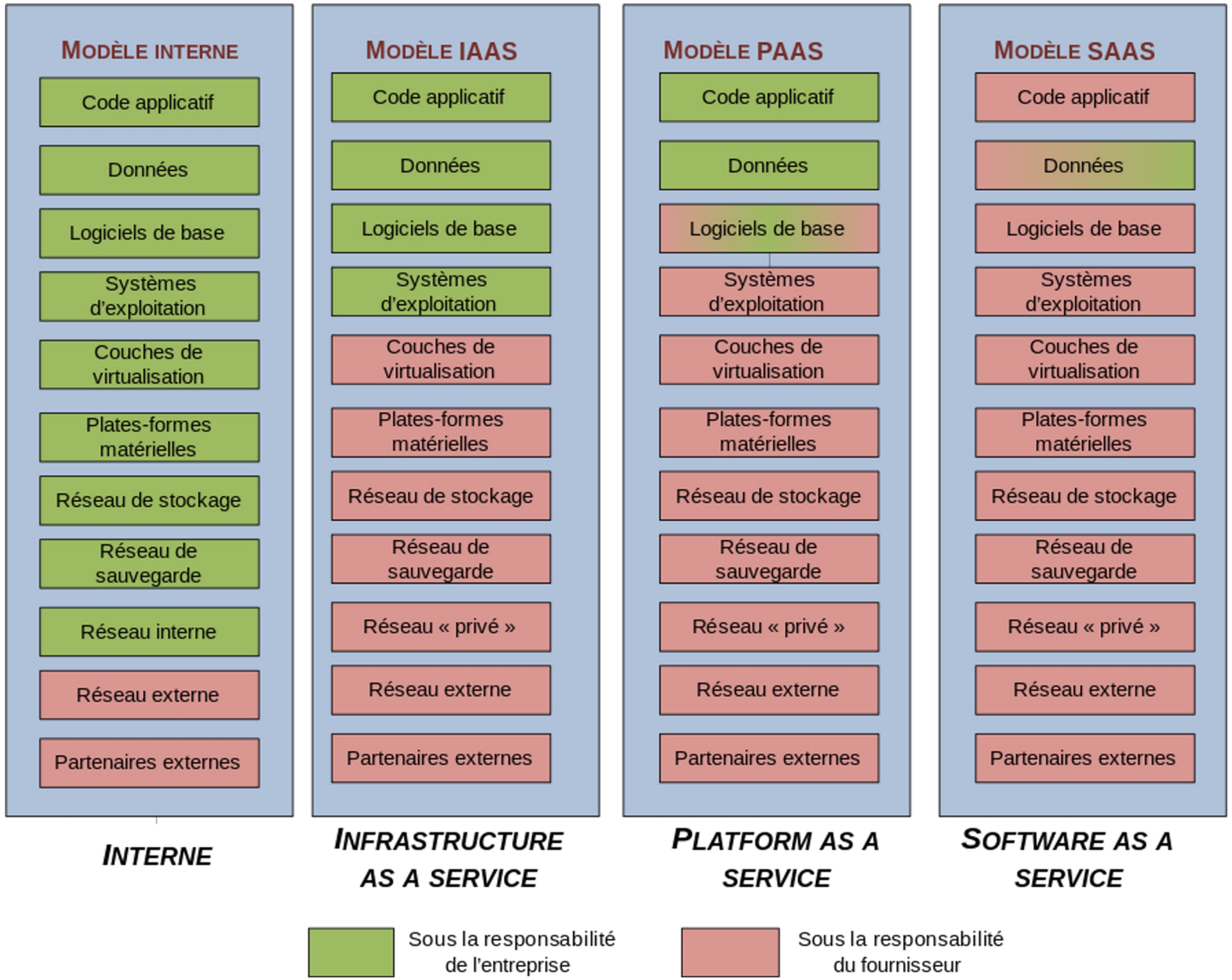

Mobile
Storage
Applications
CLOUD
COMPUTING
Server
Database
Private
cloud
Public
cloud
Hybrid
cloud

Big Data: Cloud Computing

- Infrastructure disponible immédiatement
- Infrastructure as a service
- Des services:
  - Managés
  - Sécurisés
  - Résilients
  - Avec haute disponibilités
- « Pay as you go »



Big Data: Cloud Computing



Big Data: Stockage

- Stockage type objet <clé, objet>
- Serverless : aucune infrastructure à gérer
- Objet sauvegardé sous des « buckets »
  - Structure plate de données
  - Données non modifiables
  - Données non lisible
  - Opérations acceptées : Get, Put

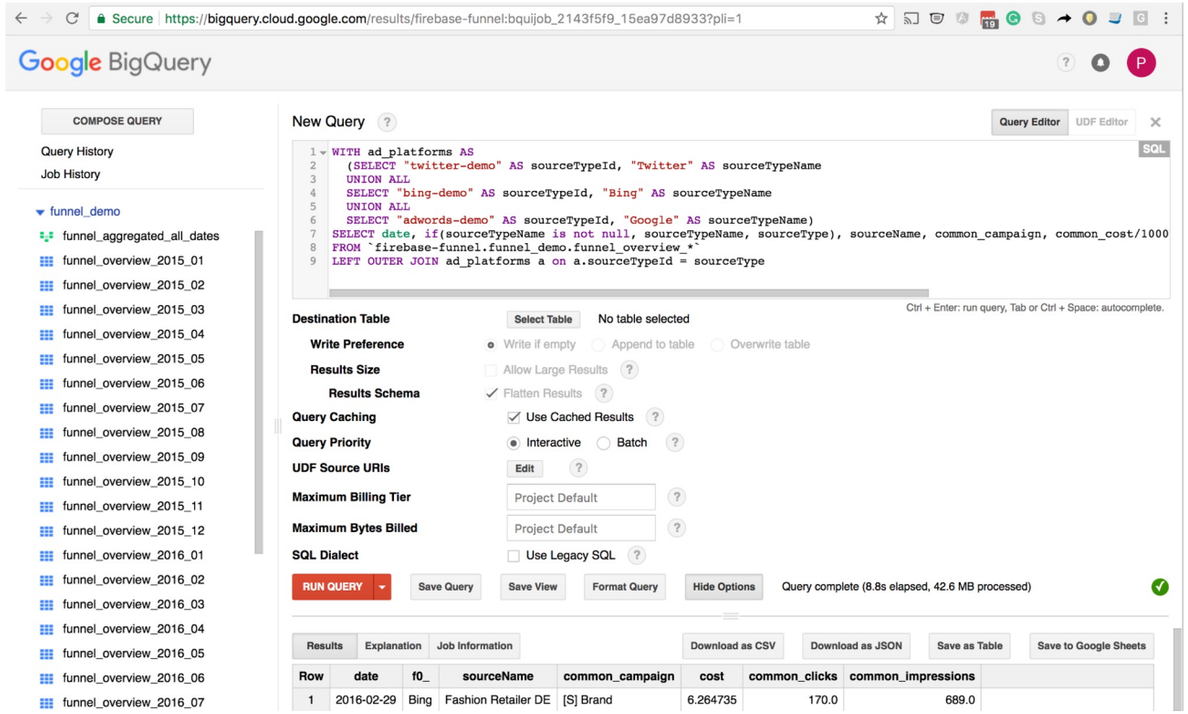
- Exemples de services:
  - Google Cloud Storage : GCS
  - Amazon Simple Storage Services : S3

Big Data: Analytique

Google BigQuery :

BigQuery est l'entrepôt de données d'entreprise sans serveur et hautement adaptable de Google, conçu pour augmenter la productivité de des analystes de données

Il permet d’analysez toutes les données en créant un entrepôt logique sur un stockage en colonnes géré, ainsi que des données issues d'espaces de stockage d'objets et de feuilles de calcul

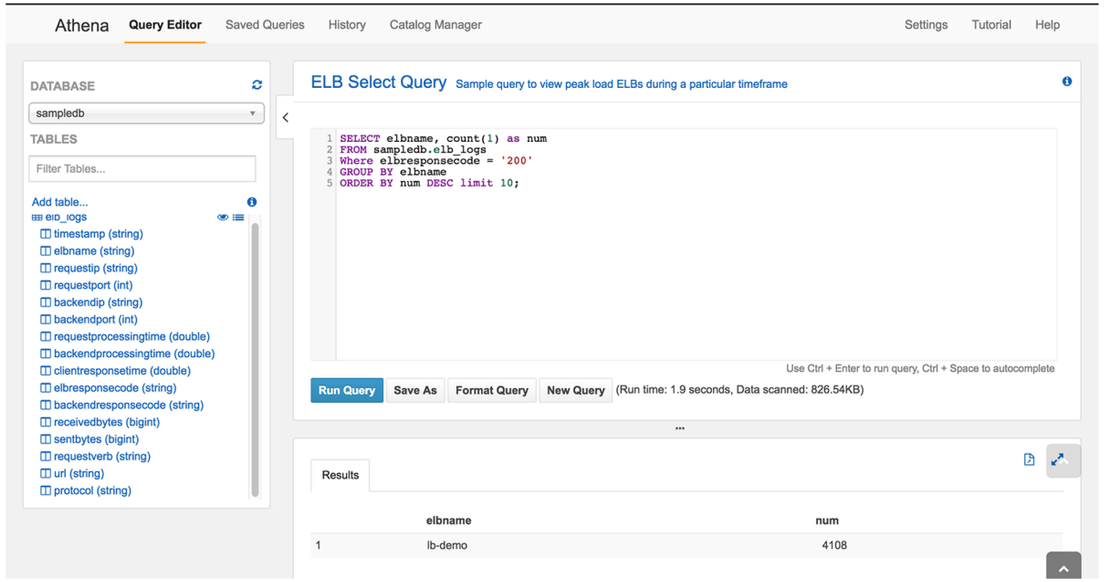


Big Data: Analytique

Amazon Athena

Amazon Athena est un service de requête interactif qui facilite l'analyse des données dans Amazon S3 à l'aide de la syntaxe SQL standard

Athena fonctionne sans serveur. Il n'existe aucune infrastructure à gérer et vous ne payez que pour les requêtes que vous exécutez.



Big Data: Machines virtuelles

Une machine virtuelle, ou VM (Virtual Machine), est un environnement d'application ou de système d'exploitation (OS, Operating System) installé sur un logiciel qui imite un matériel dédié

- Haute disponibilité & Accessibilité
- Infrastructure as a code
- Caractérisé par : Taille vCpu, memoire
- Sécurité: Attaque, réseau

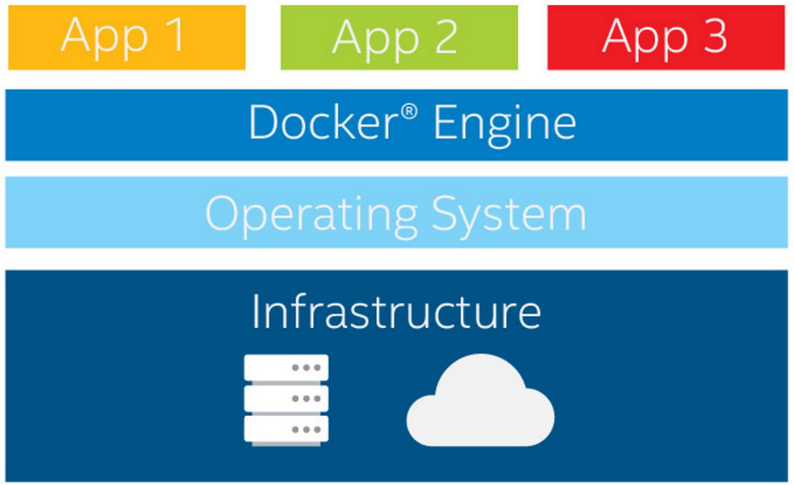
Exemples de services:

- Amazon EC2
- Google GCE

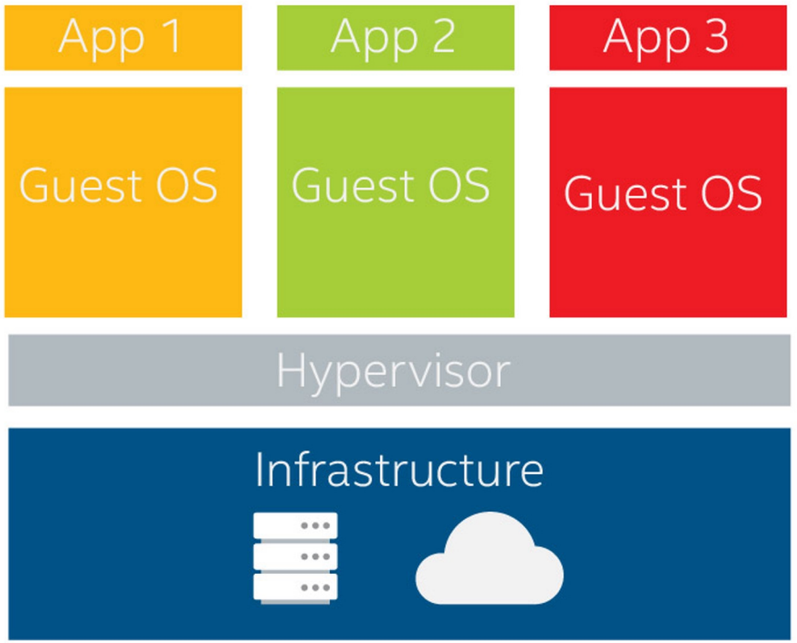
Amazon EC2

Google Compute Engine

Big Data: Docker



Docker Container Approach



Virtualization Approach

Big Data: Lambda & Cloud functions

- « AWS Lambda » est un service de calcul sans serveur qui exécute votre code en réponse à des événements et gère automatiquement les ressources de calcul sous-jacentes.

- « Google Cloud Function » est une solution de calcul qui permet aux développeurs de créer des fonctions autonomes à but unique qui répondent aux événements, sans qu'il soit nécessaire de gérer un serveur ou un environnement d'exécution.


rapidminer
Apache Solr
Dato
elastic
IBM Watson
clarifai
lucene
Spark
ORACLE NOSQL DATABASE
cassandra
TEZ
CouchDB
APACHE HBASE
NumPy
MADlib
SciPy
APACHE STORM™
HIVE
Map Reduce
mongoDB
hadoop
DATASTAX
influxdata
SciDB
MATLAB
MESOS
TACHYON
sas
OOZIE
APACHE DRILL
Apache Ambari
nifi
Couchbase
SPSS
DataTorrent
redislabs
amazon DynamoDB
mllib
plotly
Scala
accumulo
kognitio
mahout
data iku
kibana
HPCC SYSTEMS
MariaDB
TensorFlow
splunk>
sumologic
NUODB
TERADATA
VERTICA
WEKA The University of Waikato
sense
OrientDB
INFINITEGRAPH
Clustrix
Pivotal
dremio
memsql
NLTK
CONTINUUM
DataRobot
neo4j
APACHE GIRAPH
GraphLab
Greenplum
NETEZZA
SAP HANA
Data Flair

Big Data: Hadoop

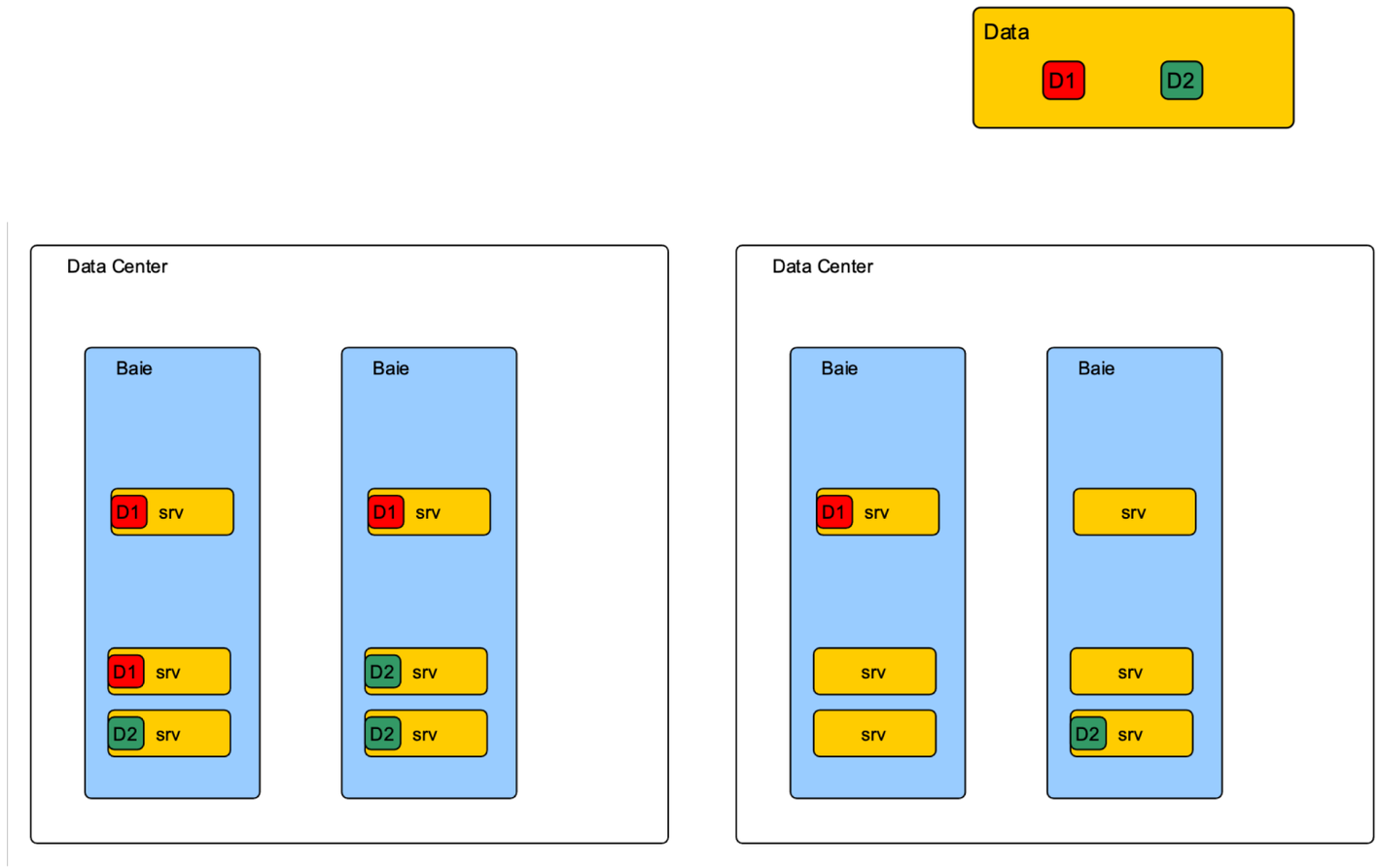
- Hadoop est un framework de stockage et de calcul de données distribuées
- Hadoop utilise des serveurs à usage générique

- Le core de hadoop est composé de :
  - HDFS hadoop distributed file system : est système de stockage de données distribués et réparties
  - MapReduce : Nouveau paradigme de développement, qui consiste à lancer des mapper, reducer
    - Mapper : charger les données et construire un dictionnaire à clé valeur.
    - Reducer : utiliser la sortie du mapper pour agréger les données

Big Data: Hadoop Histoire

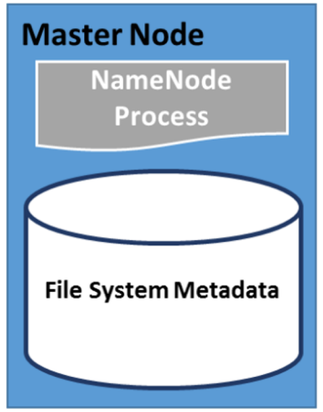
- Google file system introduit par google en 2003
- Paradigme MapReduce introduit par Google en 2004
- Première version communautaire du projet Apache Hadoop en 2006 utilisant :
  - MapReduce
  - HDFS
- Première implémentation dans un data center par yahoo en 2006

Big Data: Hadoop HDFS

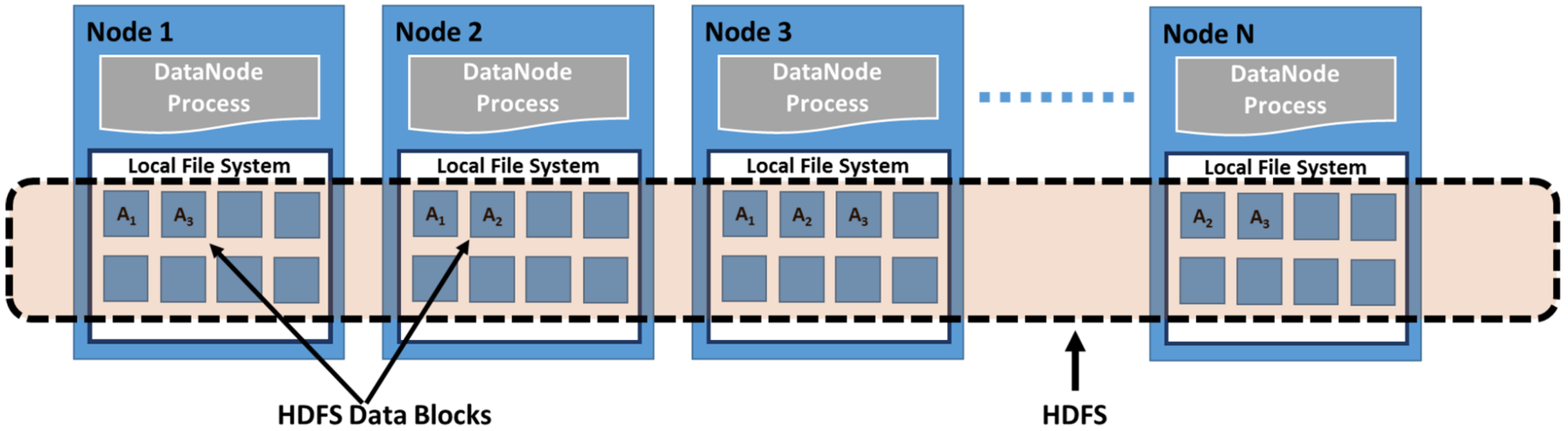


Big Data: Hadoop Architecture

MapReduce est un framework de calcul distribué
Le calcule se fait au plus prêt de la données



MapReduce → YARN
Mapper, Shuffle, Reduce

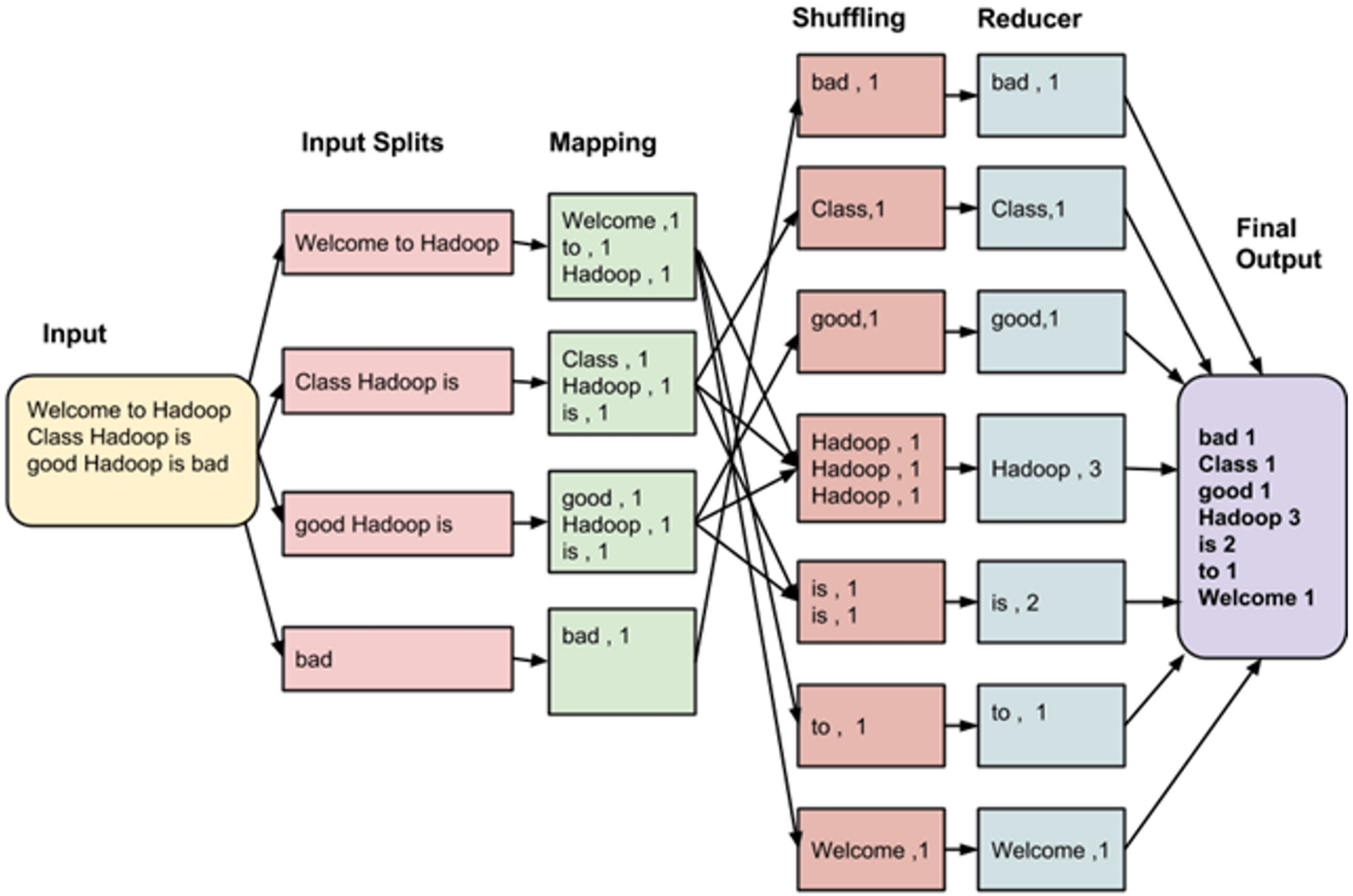


Big Data: Hadoop

- MapReduce est un paradigme de programmation
- Fonctionne exclusivement sur le couple <clé, valeur>
- L'entrée et la sortie d'un job sont toujours sous la forme <clé, valeur>
- La clé et la valeur doivent être sérialisables
- La séquence d'un job MapReduce :

(input) <k1,v1> → map → <k2,v2> → shuffle→ <k2,v2> → reduce → <k3,v3> (output)

Big Data: Hadoop MapReduce

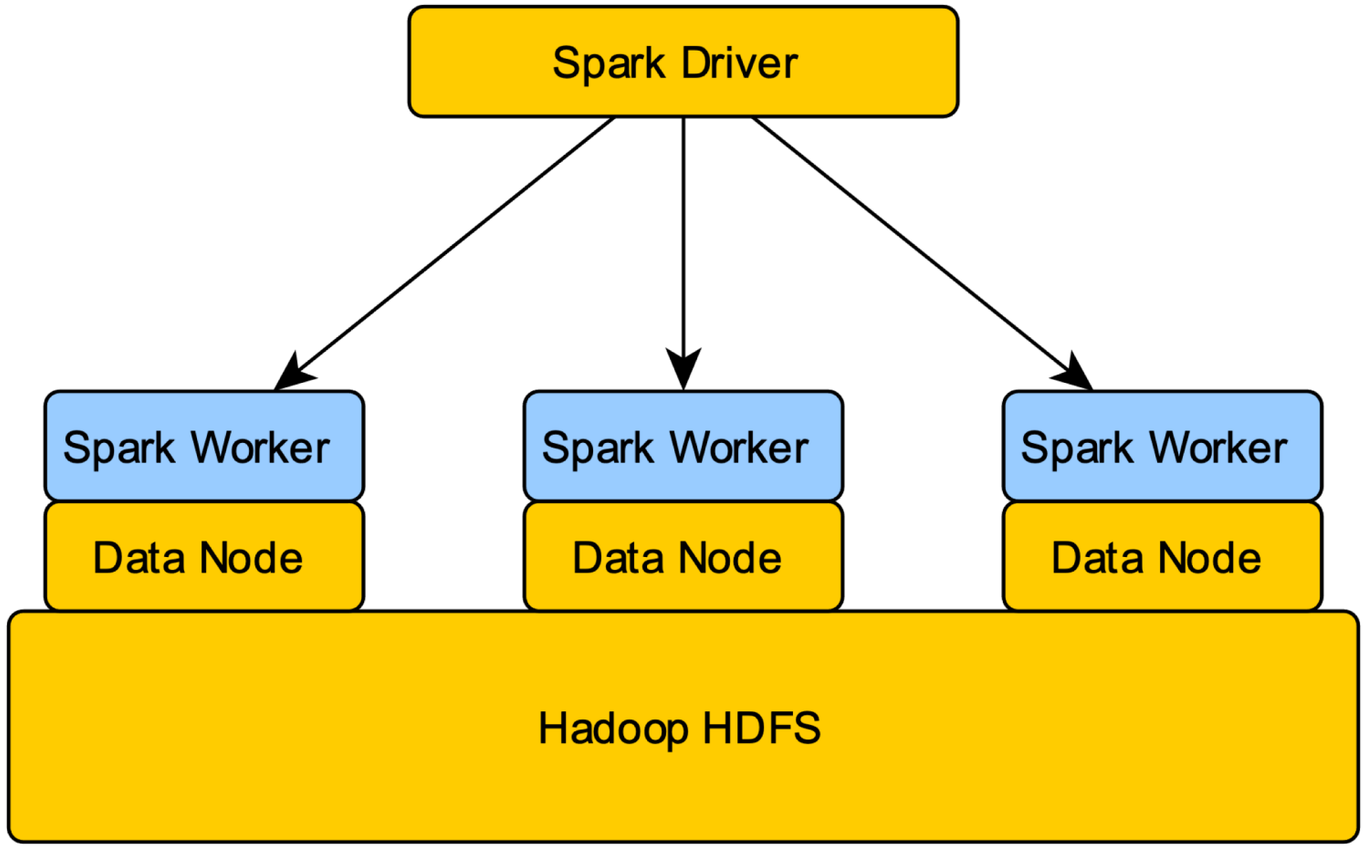


Big Data: Spark

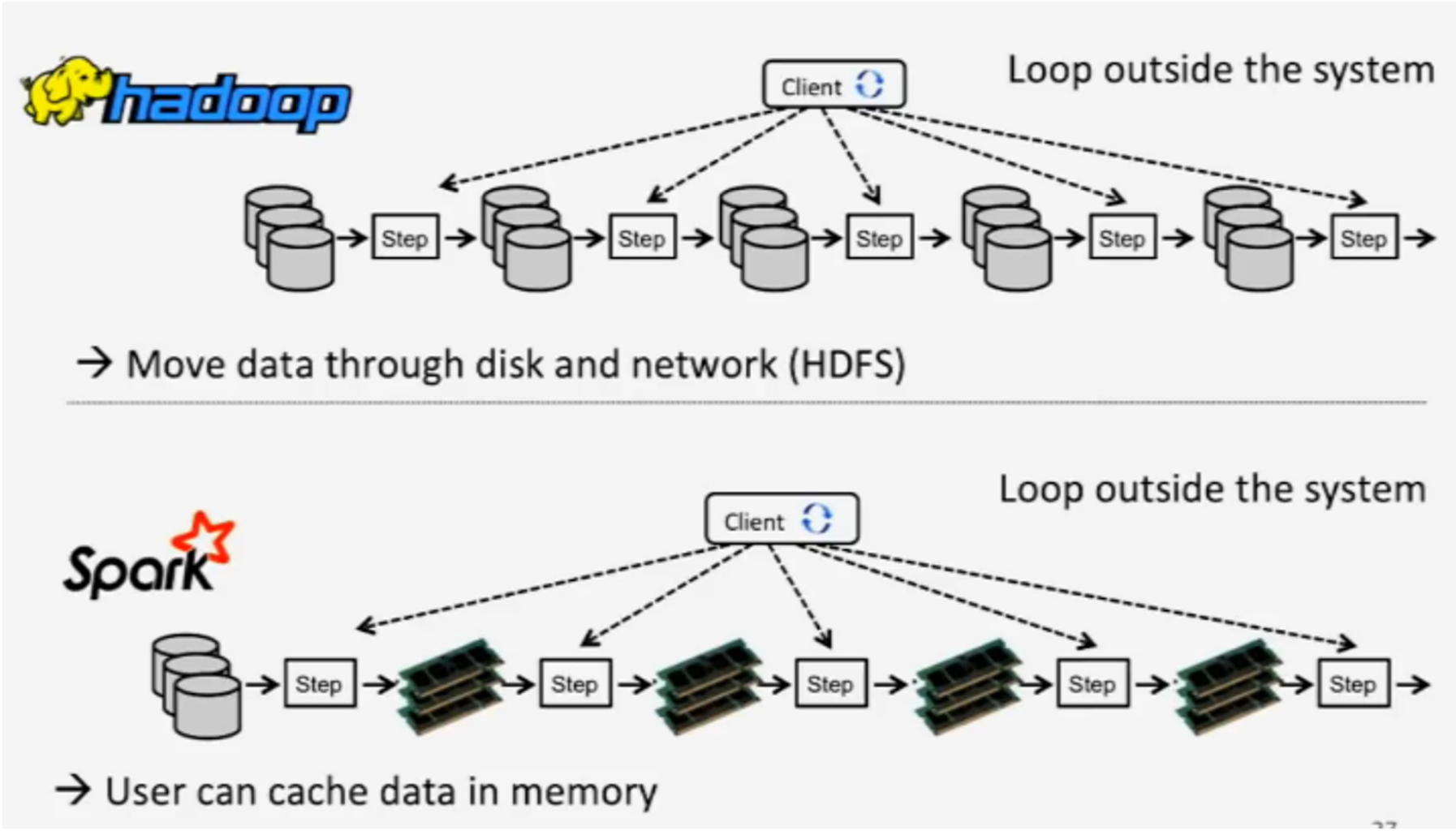
- Est un framework de développement :
  - De calcule : MapReduce
  - Distribué
  - Exécution en mémoire
  - Fonctionnel
  - Lazy Execution « paresseux »

- Spark peut être jusqu'à 100 fois plus rapide que Hadoop
- Permet de lancer des requêtes en mode interactif
- Spark offre plusieurs API : Java, Python, Scala, R et SQL

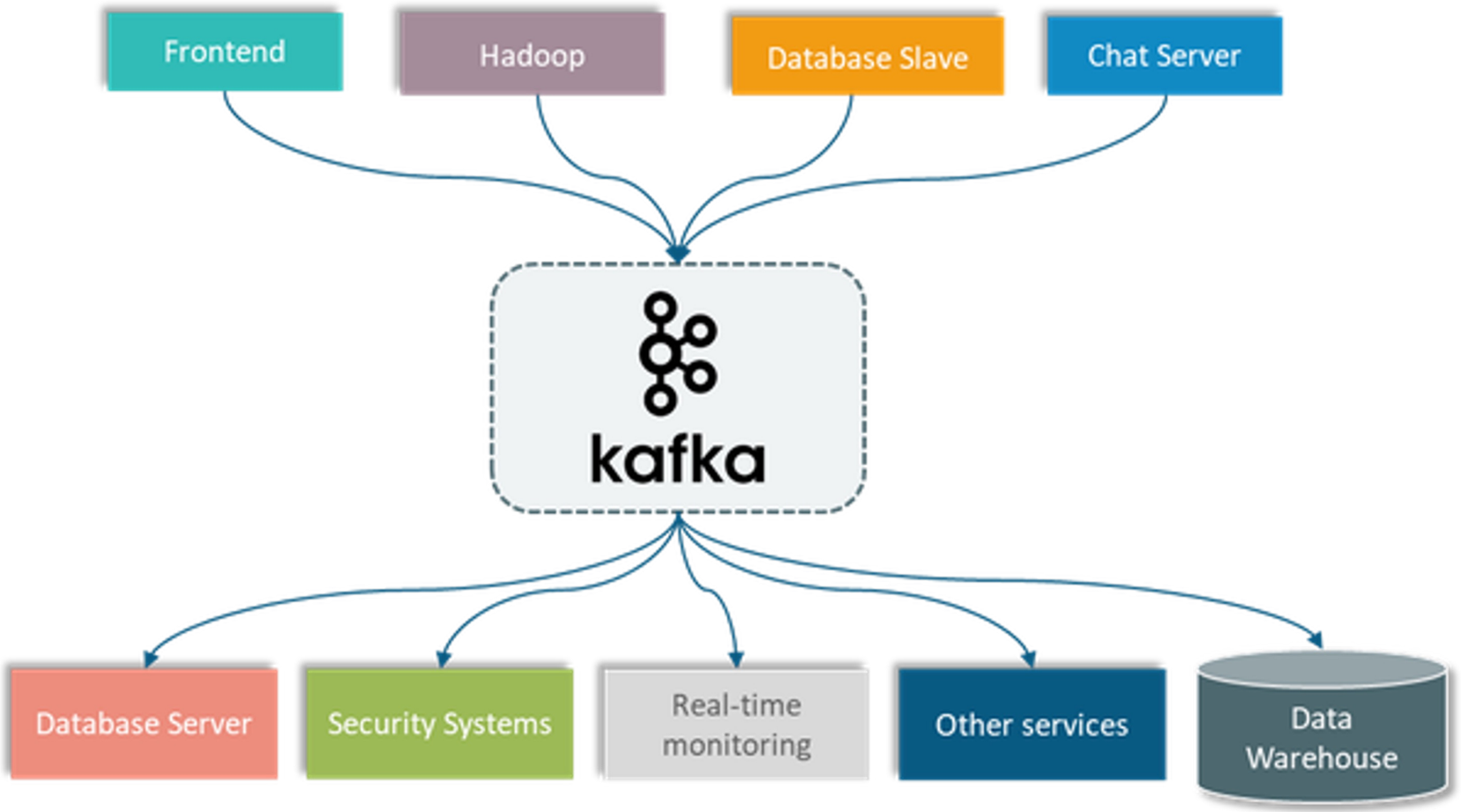
Big Data: Spark Architecture



Big Data: Spark vs Hadoop

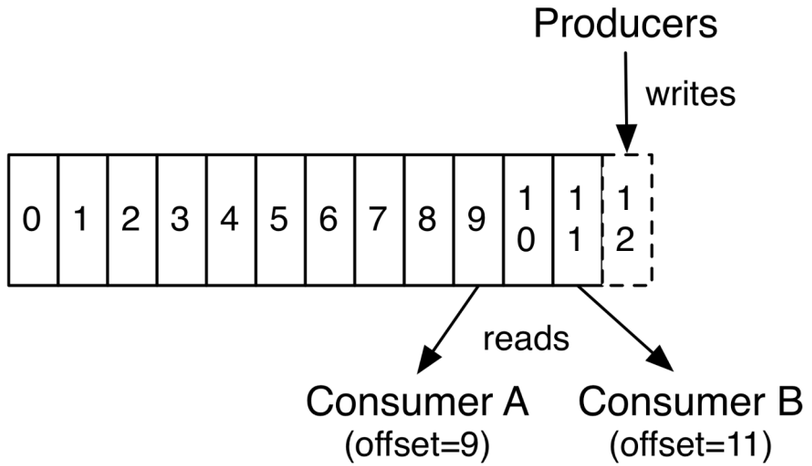


Big Data: Kafka

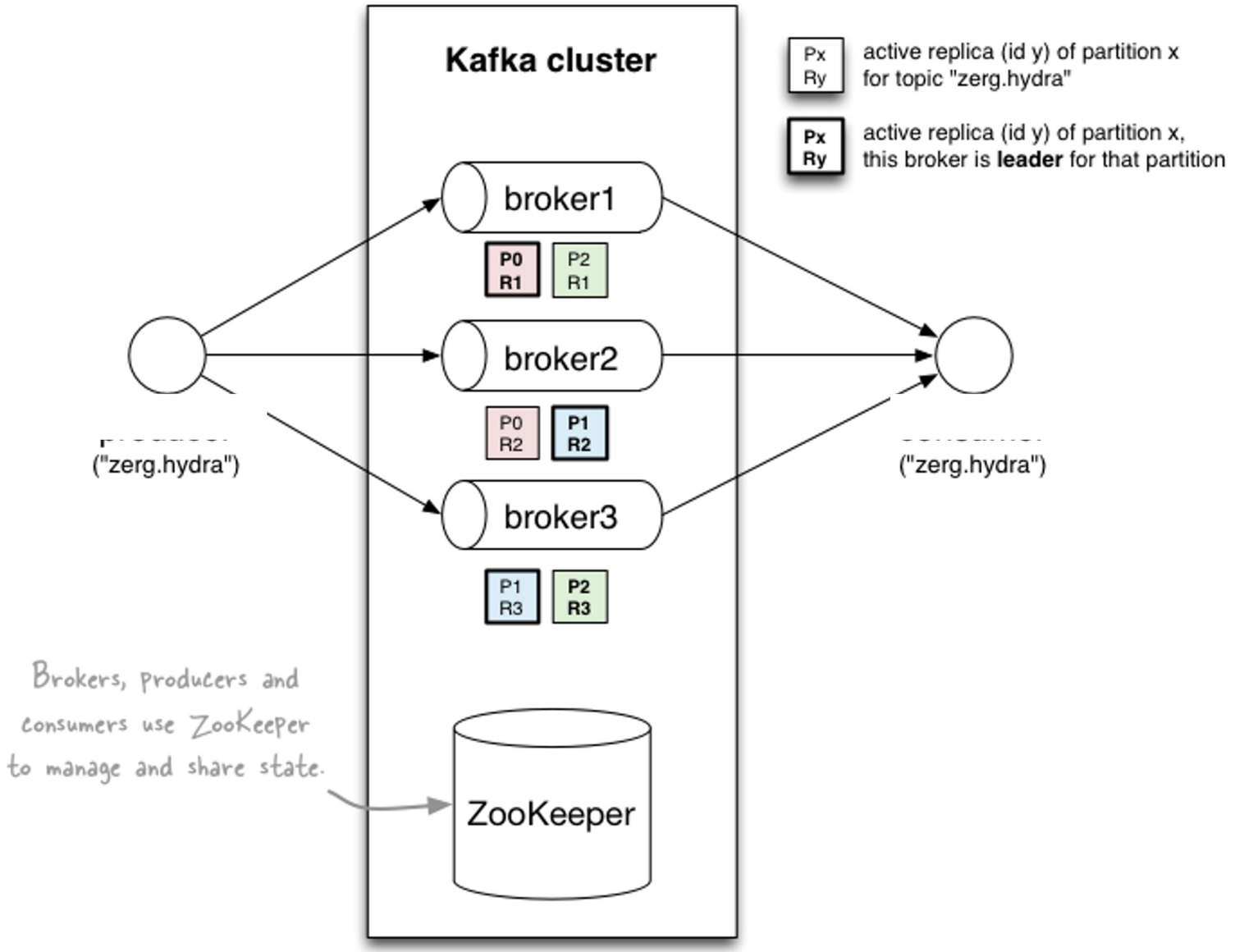


Big Data: Kafka

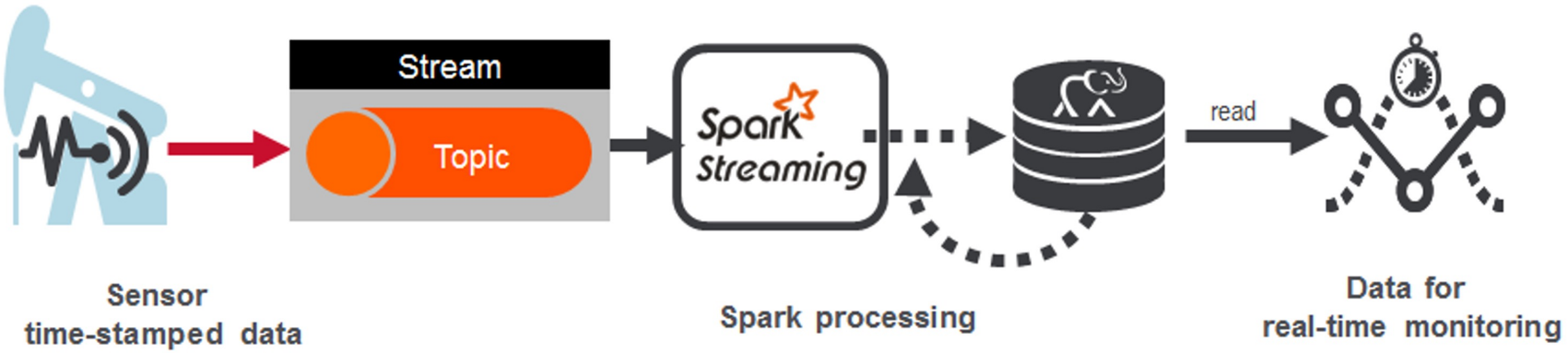
- Un bus de message
- Performant : millions de messages par seconde
- Résiliant : distribué et réplication
- Une base de données bas niveau
- Un message est composé d'une valeur, d'une clé et d'un timestamp
- Kafka organise les messages en catégories appelées topics



Big Data: Kafka



Big Data: Architecture




Data Lake
Security
Data Governace
Data Quality
Data Discovery
Data Auditing
Data Storage
Data Lineage
DATA Exploration
Data Ingestion
Teamwork

Big Data: DataLake

- Sécurité : Privacy by design
  - Réseau
  - Accès
  - Autorisation
  - Chiffrement et anonymisation
- Cycle de vie
- Etat de la donnée
  - En mouvement
  - Statique : Chaude, Tiède et Froide
- Retracer le cycle de vie de la données

- Exploration de la données
  - Mettre en place les outils nécessaire pour exploiter les données et les analyser. Notebook jupyter ou Tableau software
- Data Ingestion et discovery :
- Mettre en place des mécanisme d'ingestion et sauvegarde
- Découverte de la structure de la données.
- Data Auditing
- Garder trace de toutes le transformations et de qui a fait quoi

RGPD

Règlement Général sur
la Protection des Données

Big Data: RGPD

Du RGPD se dégage quatre principes clés :

le consentement, la transparence, le droit des personnes et la responsabilité.

- **Le consentement** "le consentement devrait être donné par un acte positif clair par lequel la personne concernée manifeste de façon libre, spécifique, éclairée et univoque son accord au traitement des données à caractère personnel la concernant, par exemple au moyen d'une déclaration écrite, y compris par voie électronique, ou d'une déclaration orale." Le consentement peut être retiré à tout moment par les personnes le demandant.

- **La transparence** les organisations doivent fournir aux individus des informations claires et sans ambiguïté sur la façon dont sont traitées leurs données. Celles-ci doivent être accessibles par tous, via des documents contractuels, des formulaires de collecte ou les pages "privacy" des sites web.

Big Data: RGPD

Le droit des personnes

- le droit à l'oubli pour tous les utilisateurs : les organisations n'ont plus qu'un mois pour supprimer les données suite à une demande.
- Le droit à la portabilité des données: il permet à un individu de récupérer les informations qu'il a fournies sous une forme réutilisable pour, le cas échéant, les transférer à un tiers.

Le principe de responsabilité

- Il regroupe toutes les mesures qui visent à responsabiliser d’avantage les entreprises dans le traitement des données à caractère personnel. Les organismes doivent par exemple mettre en place des mesures adéquates pour garantir la sécurité des données. Elles doivent également appliquer le "privacy by design", un concept qui impose de réfléchir à la protection des données personnelles en amont de la conception d'un produit ou d'un service. Elles doivent aussi choisir des sous-traitants qui soient conformes au RGPD ou encore désigner un data protection officer (DPO), chargé de contrôler la conformité de l'organisme avec le RGPD.


25%
1101 0010 1100
1101 0010 1100
1101 0010 1100
%

Big Data: Analyse de données

Analyse qualitative

- Nettoyage des données :
- Suppression des champs vides ou mal formaté
- Suppression des valeurs aberrantes
- Reformatage des données
- Analyse des distributions des données
- Définition de la variables cible
- Analyse des corrélation entre variables
- Construction de Dashborad de visualisation
- Etude de l’impact des variables sur la cible
- Normalisation des données
- Analyse de la composante principale PCA

Analyse quantitative

- Analyse des distribution des données par rapport à la cible
- Vérification du nombre d’entrée pour chaque catégorie de la cible

Big Data: Analyse de données

Développeur Python:

- Pandas: est une bibliothèque Open Source fournissant des structures de données adaptées à l'analyse de données.
- Numpy: est une bibliothèque de calcul scientifique
- Matplotlib : est une bibliothèque de visualisation 2D
- Jupyter: est un notebook de développement dans une interface web

Recommandation :

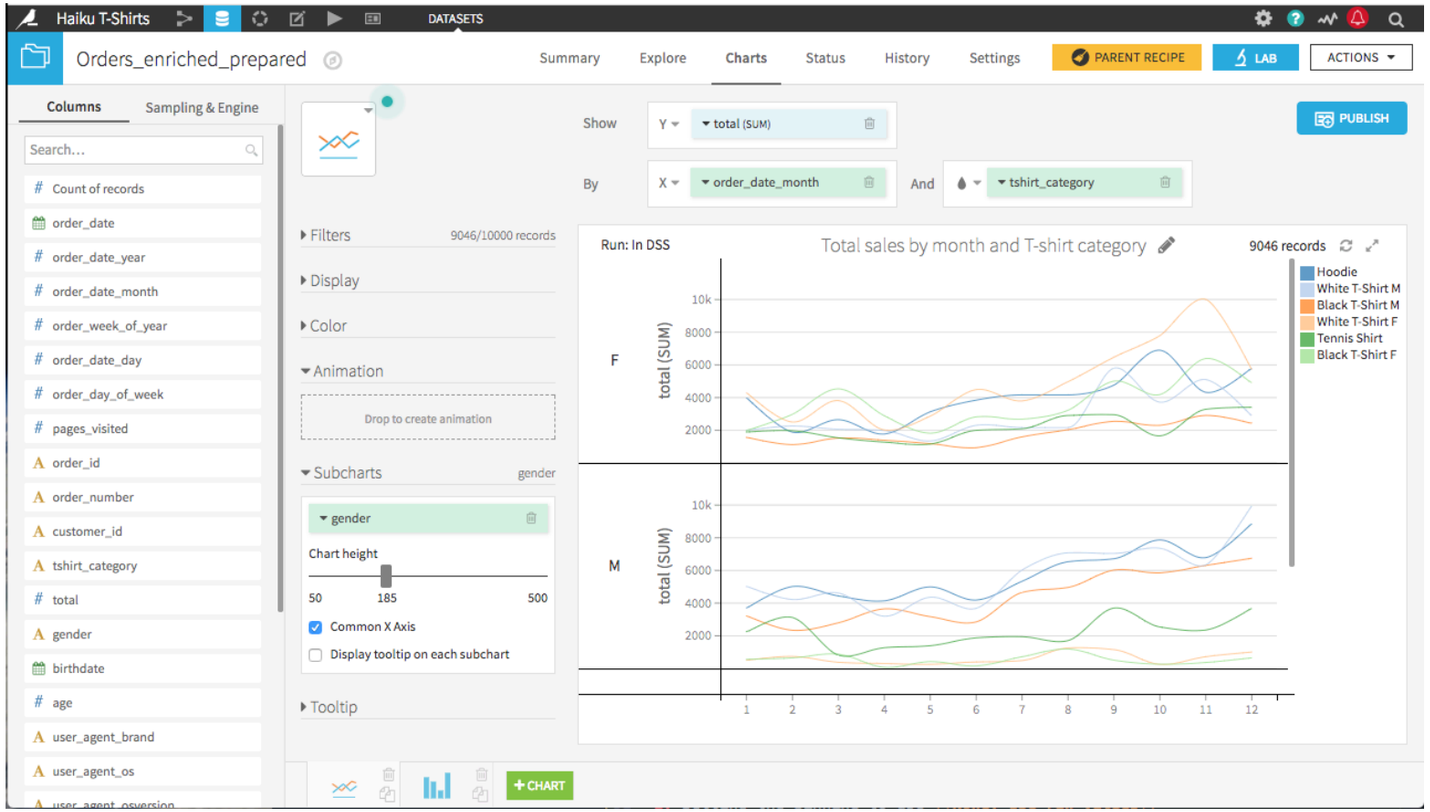
- Utiliser toujours un environnement virtuel python
- Utiliser un système de versionning : git, svn

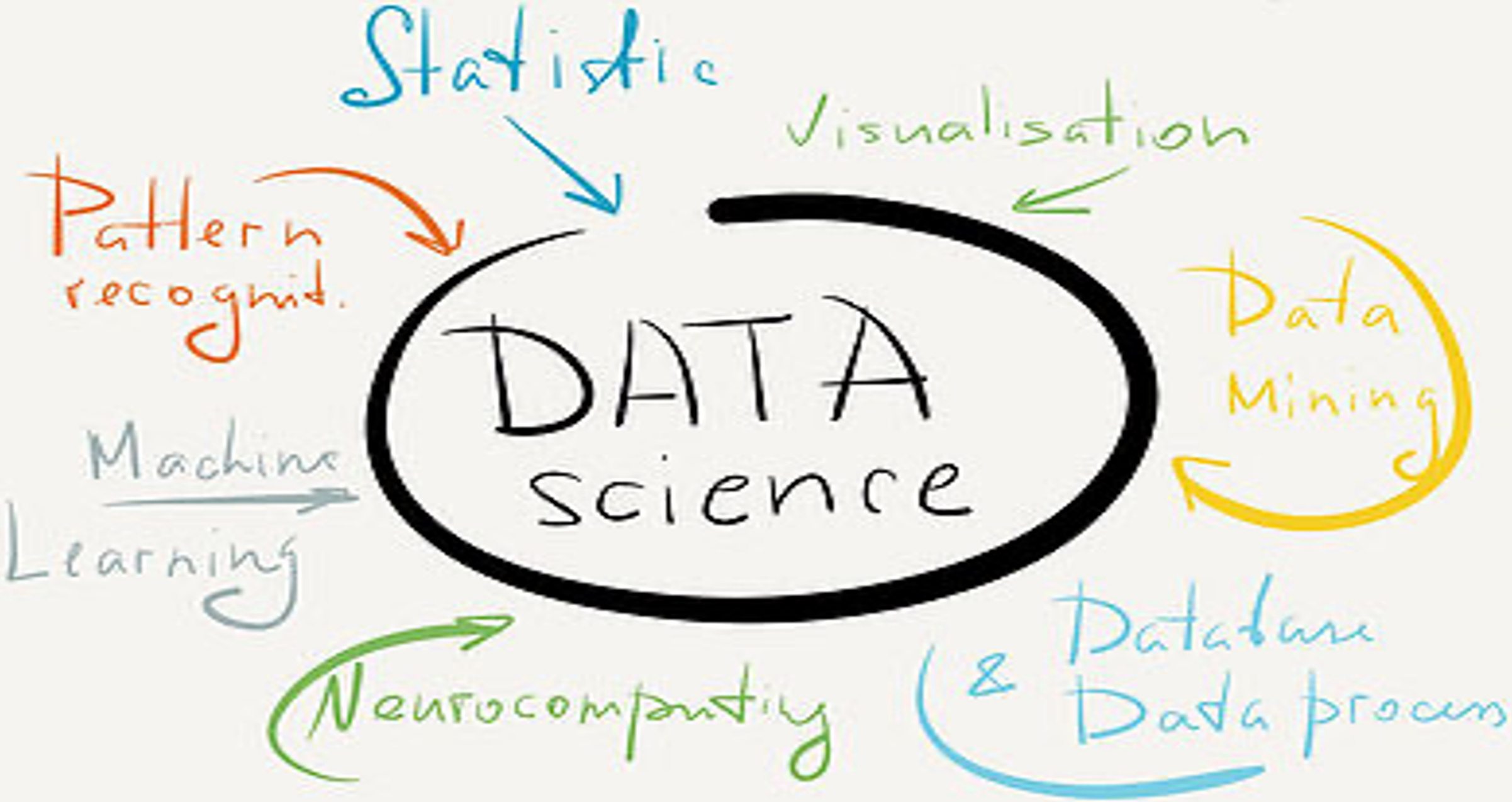
Big Data: Analyse de données

Utilisation d’outil orienté analyse de données

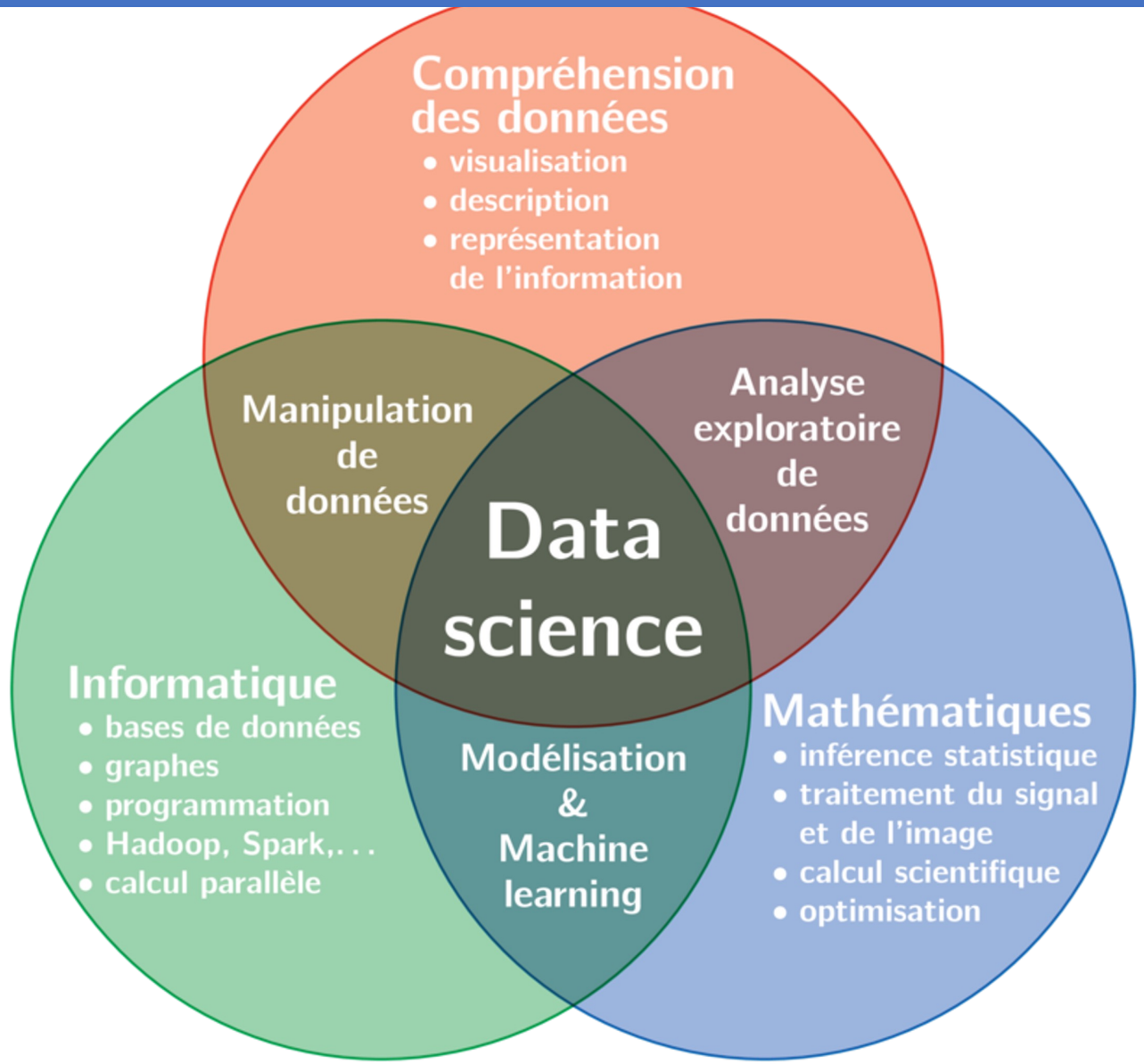
Tableau server, Dataiku..

Des outils plus adaptés pour le métier de data-analyste

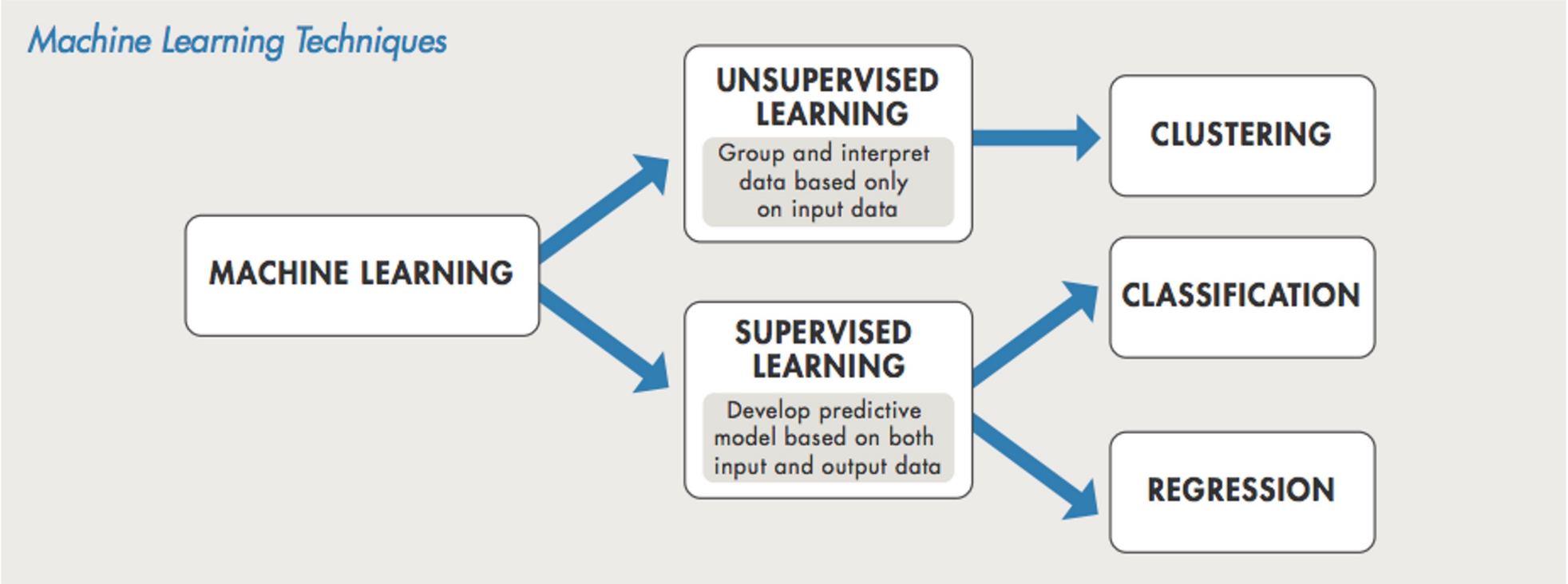



Statistic
Visualisation
Pattern recognit.
DATA science
Data Mining
Machine Learning
Neurocomputing
Database & Data proces

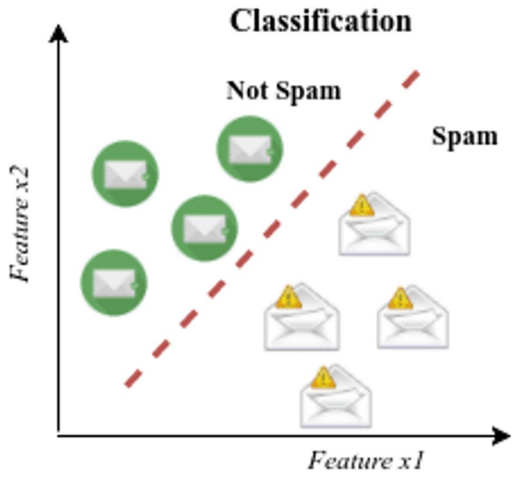
Big Data: Data Science

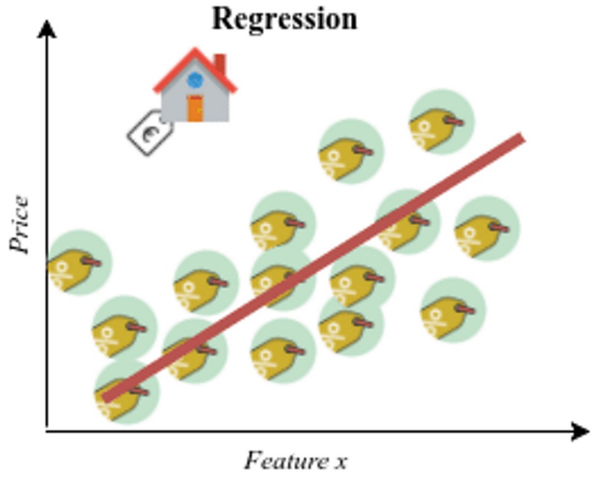


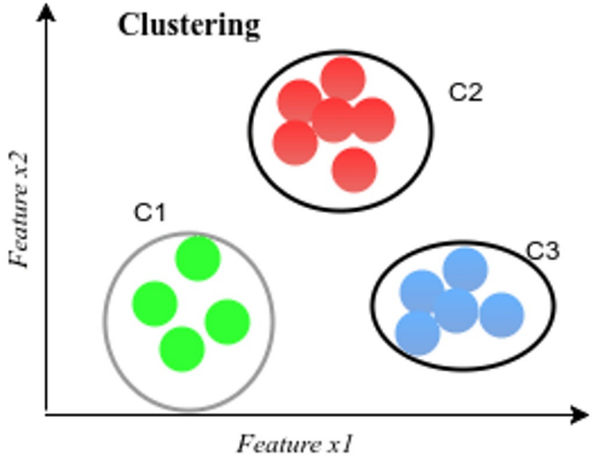
Big Data: Data Science



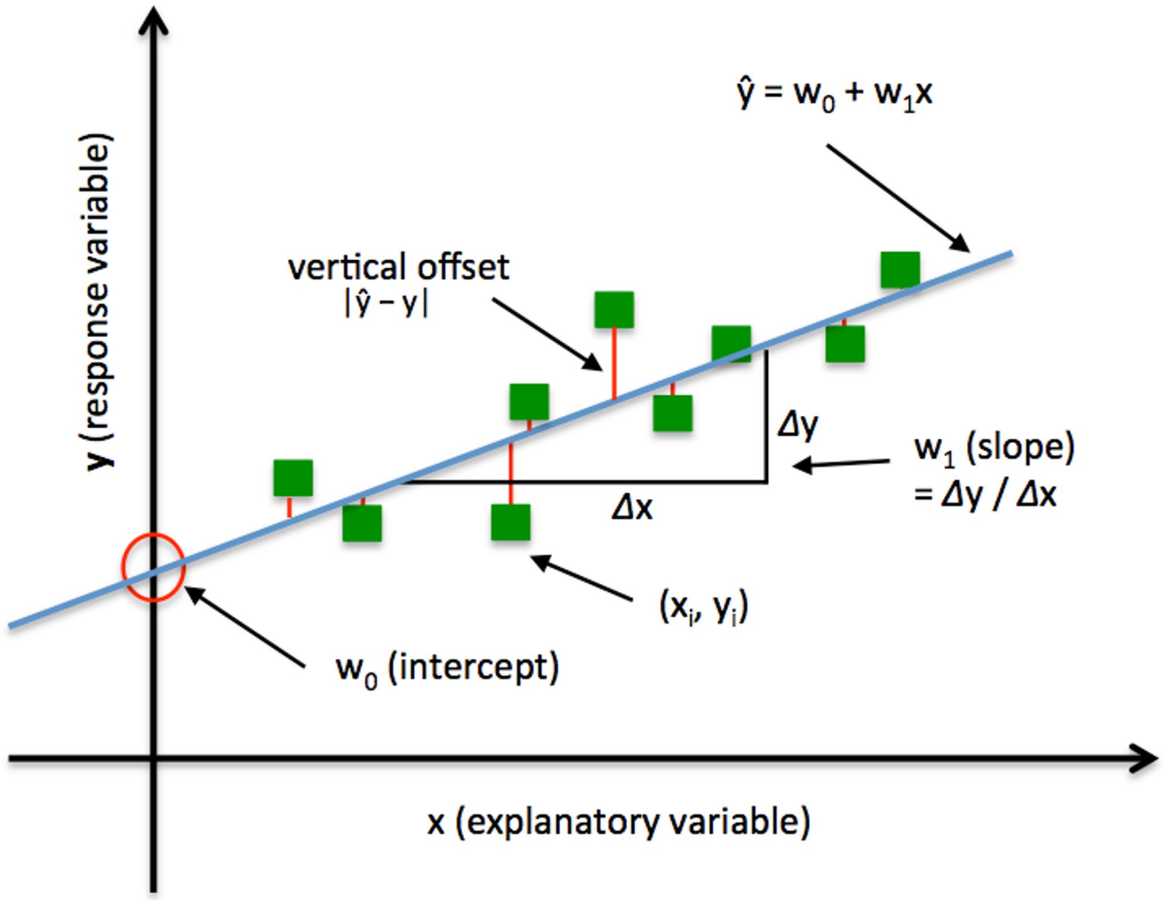
Big Data: Data Science





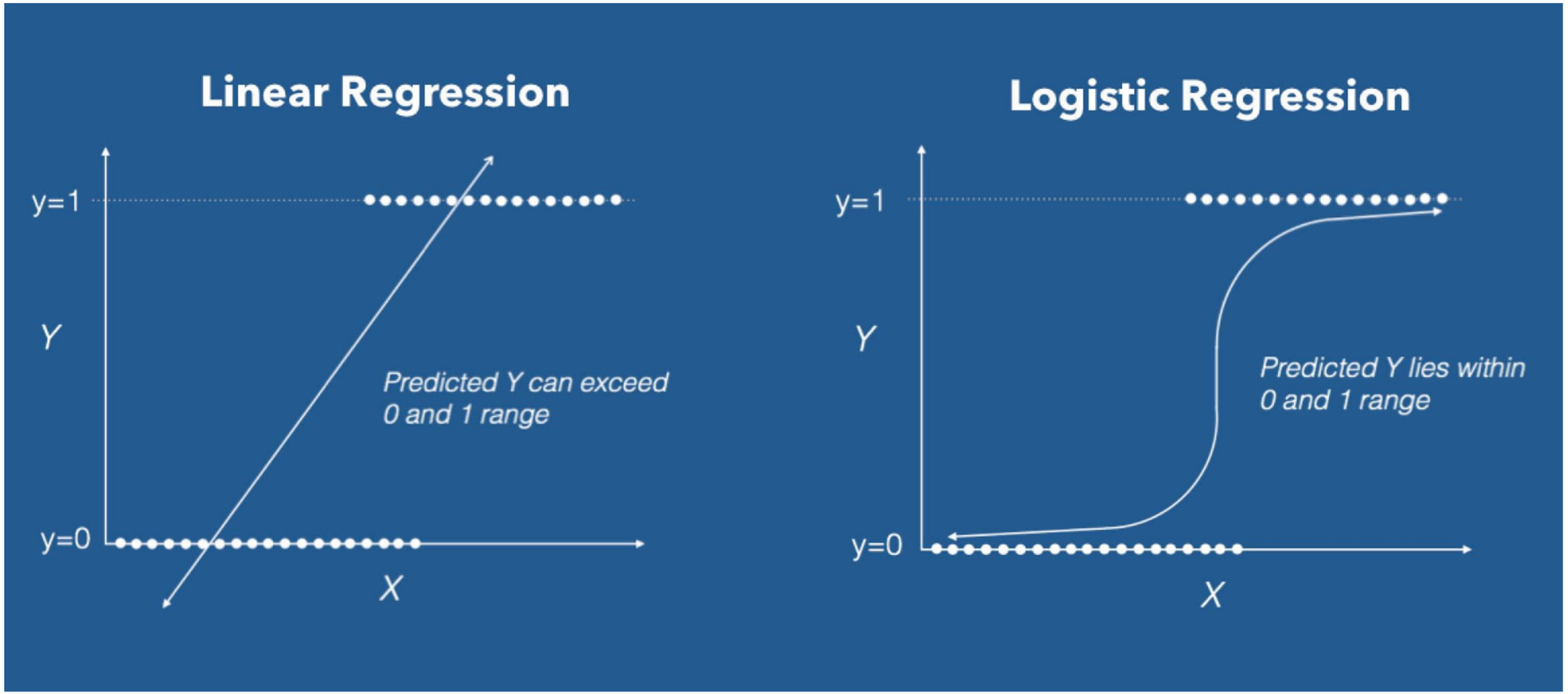


Big Data: Régression linéaire



Big Data: Régression linéaire

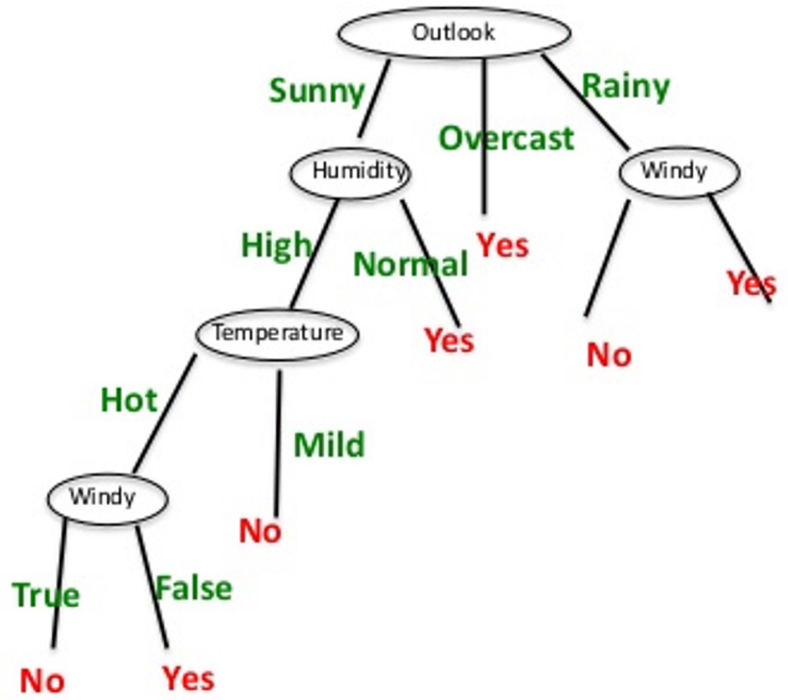
Big Data: Régression logistique



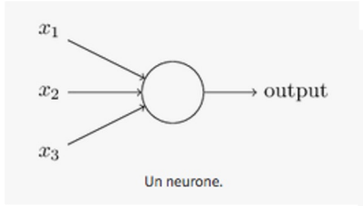
Big Data: Arbre de décision

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	Yes
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

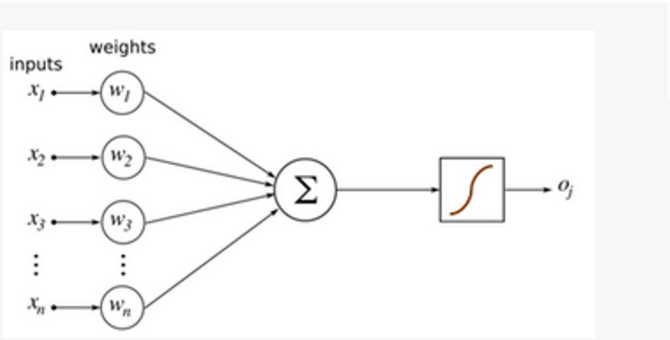
```
outlook = sunny
|   humidity = high
|   |   temperature = hot
|   |   |   windy = TRUE: no
|   |   |   windy = FALSE: yes
|   |   temperature = mild: no
|   |   temperature = cool: null
|   humidity = normal: yes
outlook = overcast: yes
outlook = rainy
|   windy = TRUE: no
|   windy = FALSE: yes
```



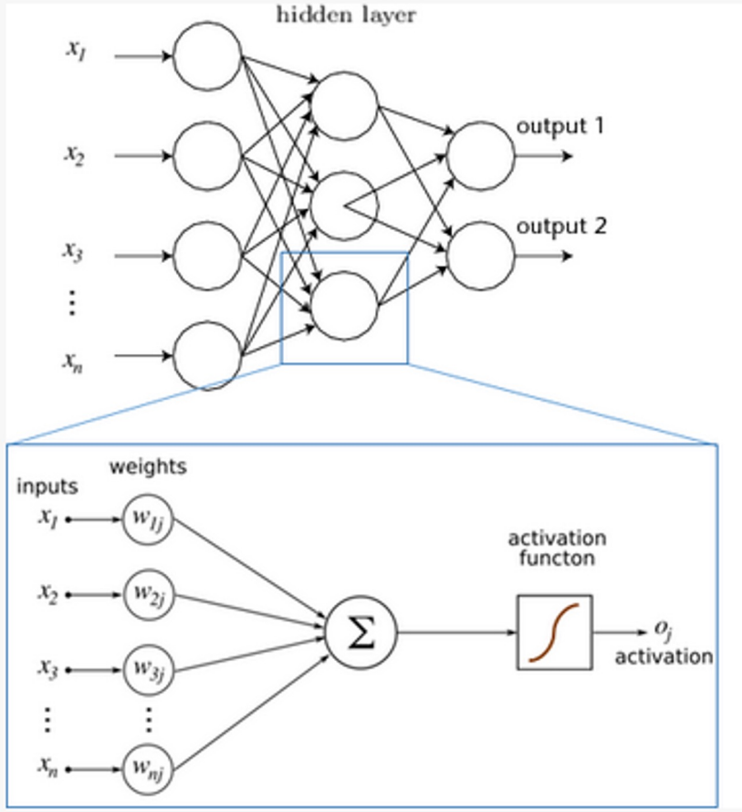
Big Data: Réseau de neurones



Un neurone.



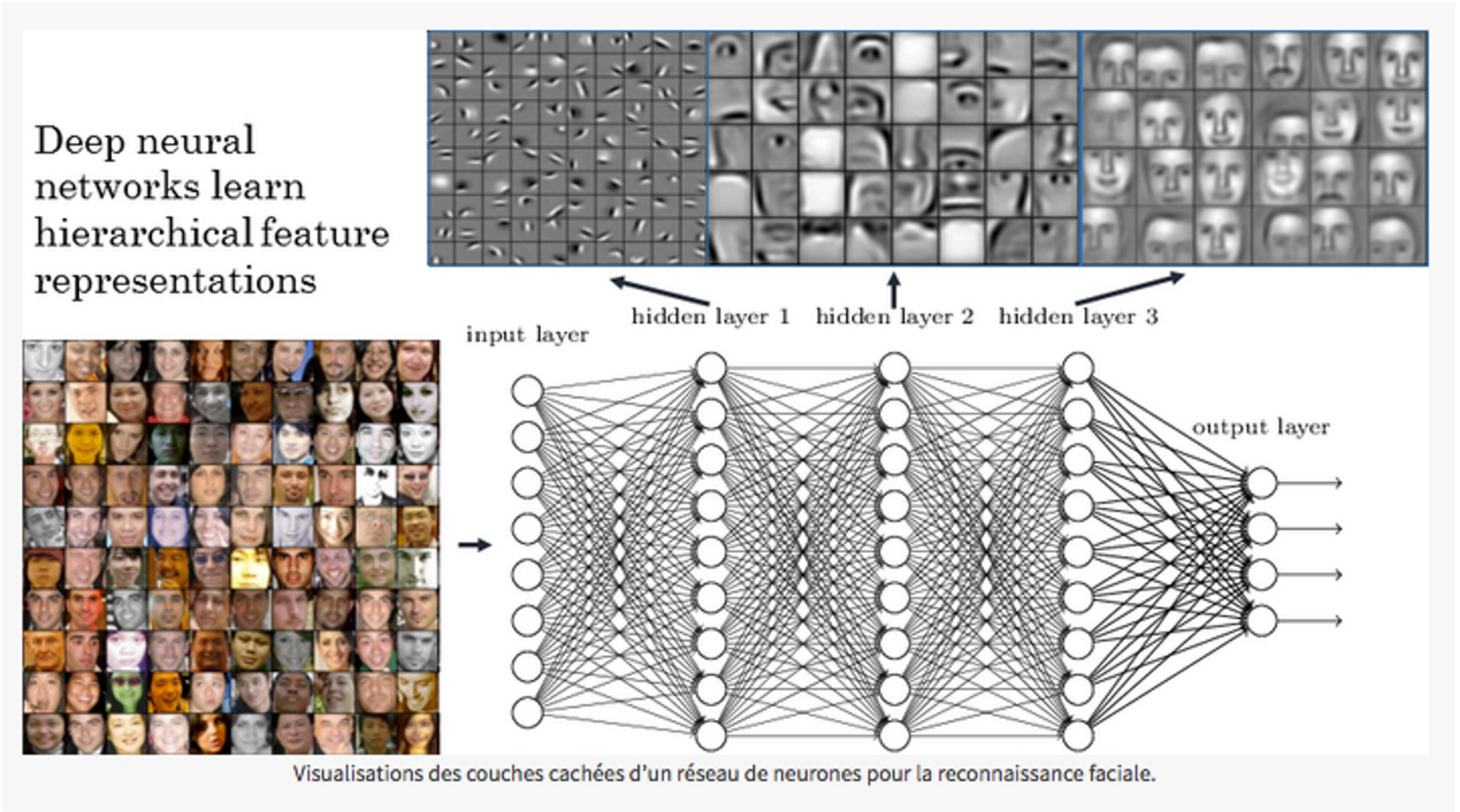
Un réseau de un neurone.



Réseau de neurones multi-couches ou *deep neural network.*

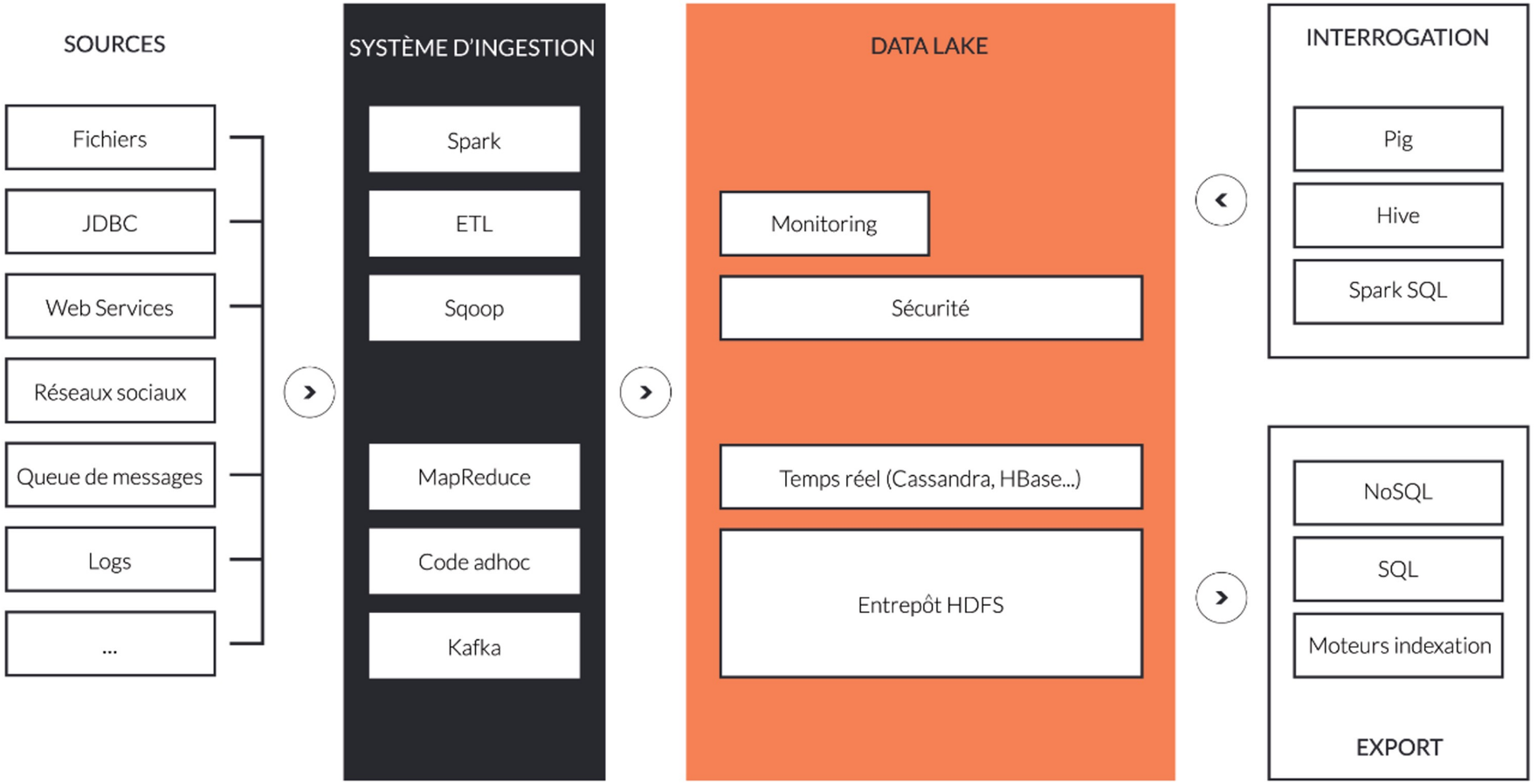
- https://playground.tensorflow.org

Big Data: Deep learning



Visualisations des couches cachées d'un réseau de neurones pour la reconnaissance faciale.

Big Data: Data pipeline



Big Data: Tools

Data Warehouse

- Est une base de données orientée analyse de données
  - Besoin de reporting, analytique & dashboard
  - Données sauvegardé sous format colonnes
  - Gère les complexes structures des données: STRUCT, ARRAY, MAP
  - Données dénormalisées
  - Interaction avec du SQL
  - Découplage entre le stockage et le calcule

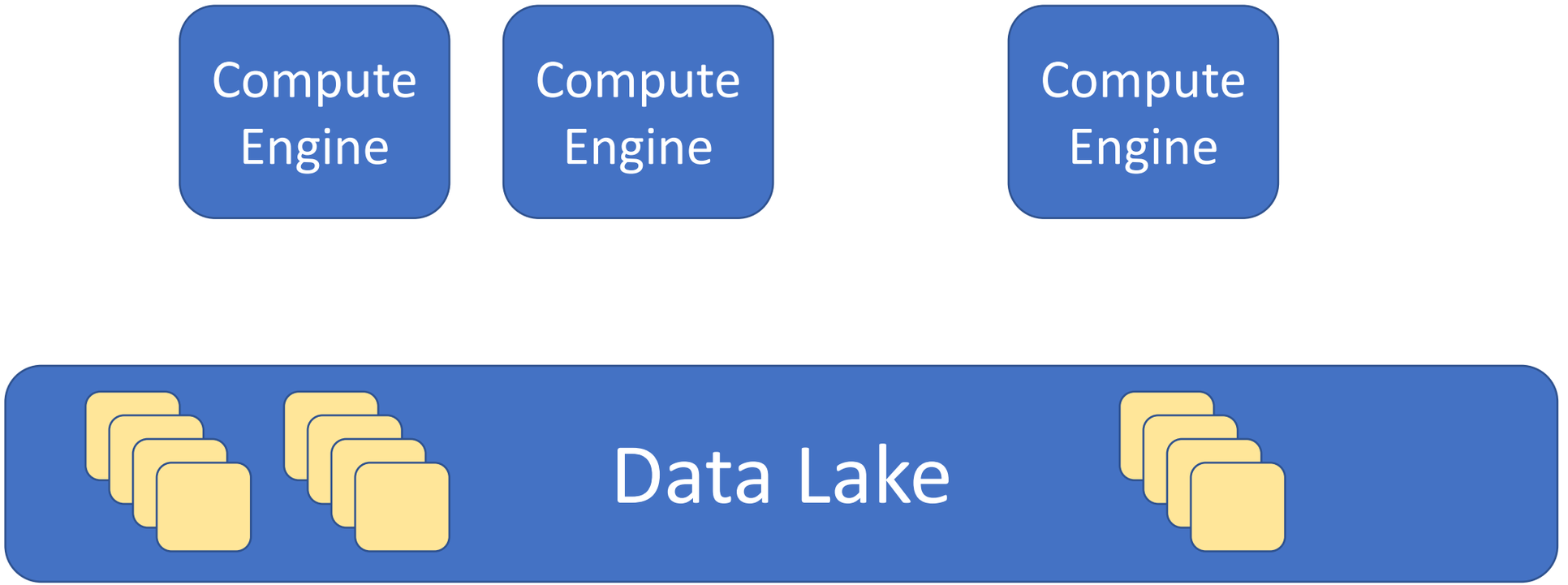
Data Warehouse: Solutions

Google BigQuery

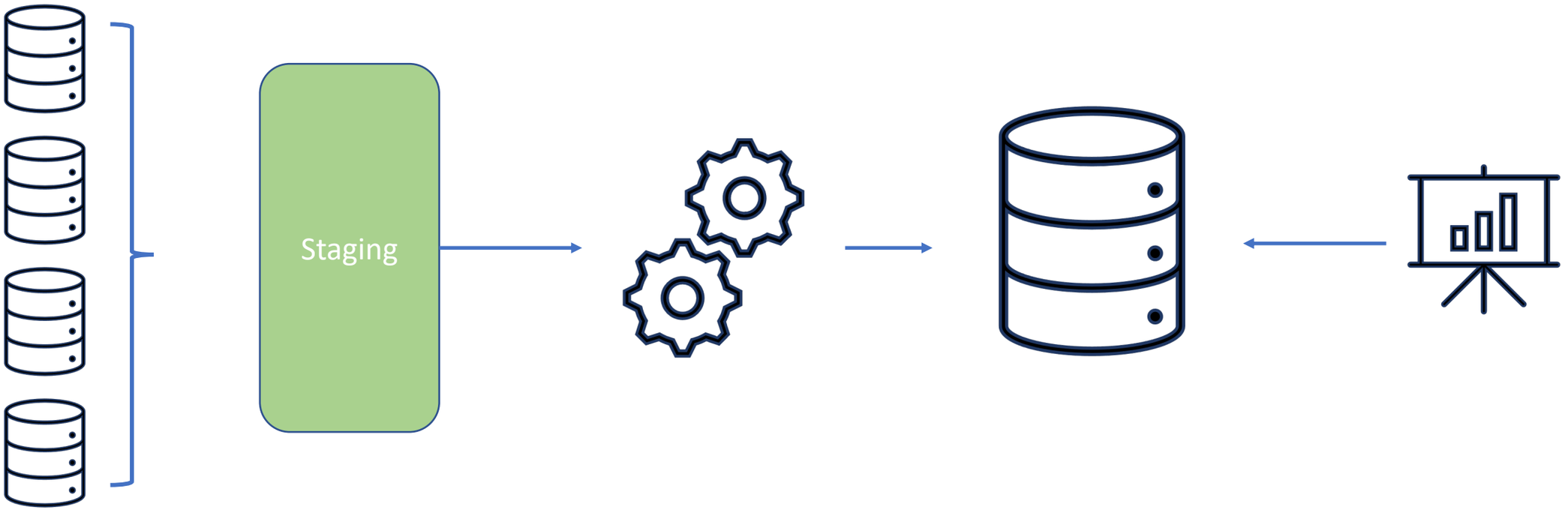
snowflake

Amazon Athena

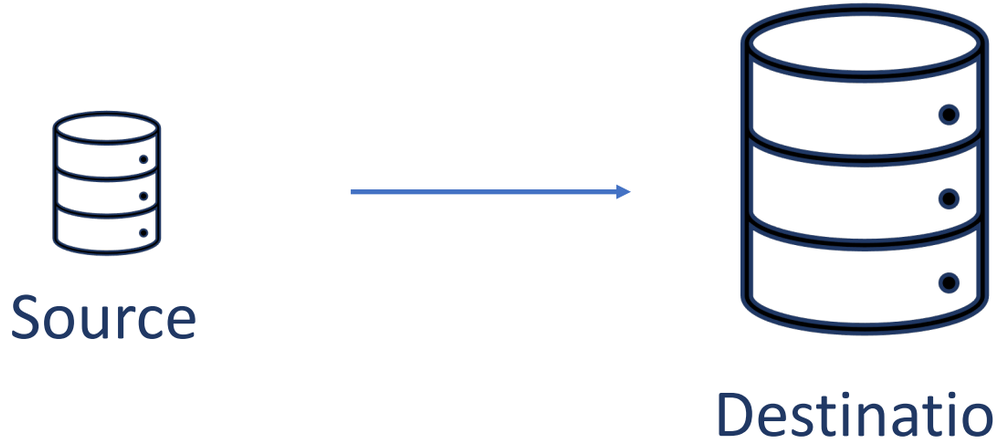
Data Warehouse: Infrastructure



Data Warehouse: ETL



Data Warehouse: Slowly Changing Dimensions SDC

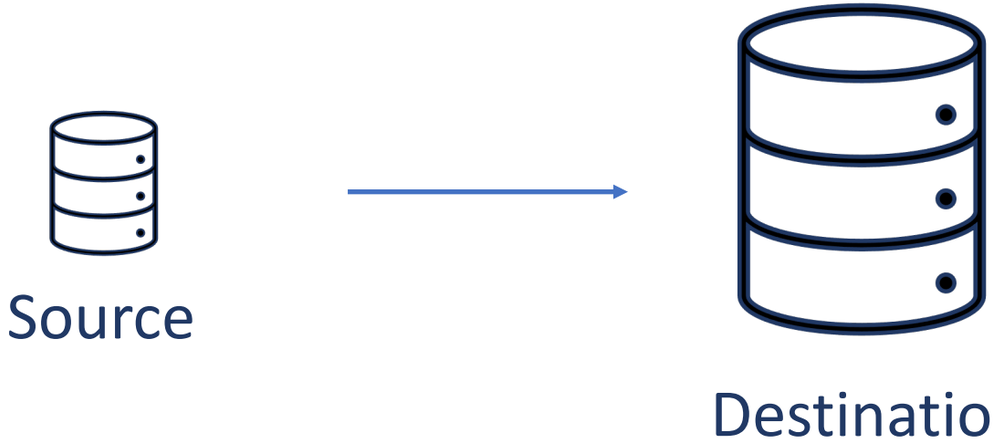


City	timestamp
Paris	2023-04-12 12:11:34

City	timestamp
Paris	2023-04-12 12:11:34

- Overwrite: updating the dwh table when a dimension changes

Data Warehouse: Slowly Changing Dimensions SDC



City	timestamp
Paris	2023-04-12 12:11:34

City	timestamp
Paris	2023-03-12 12:11:34
Paris	2023-04-12 12:11:34

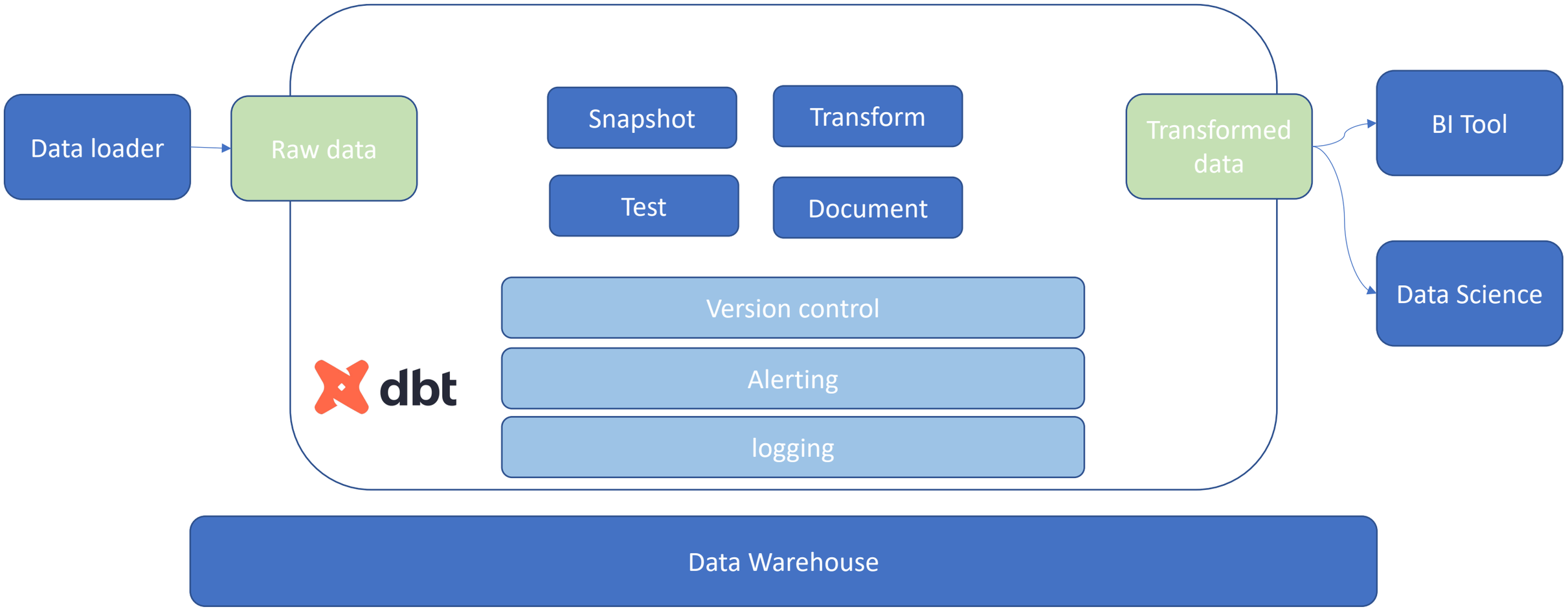
- Add new row: keeping full history, adding additional historic data (rows) for each dimension change

Data Warehouse: DBT

DBT is a transformation workflow

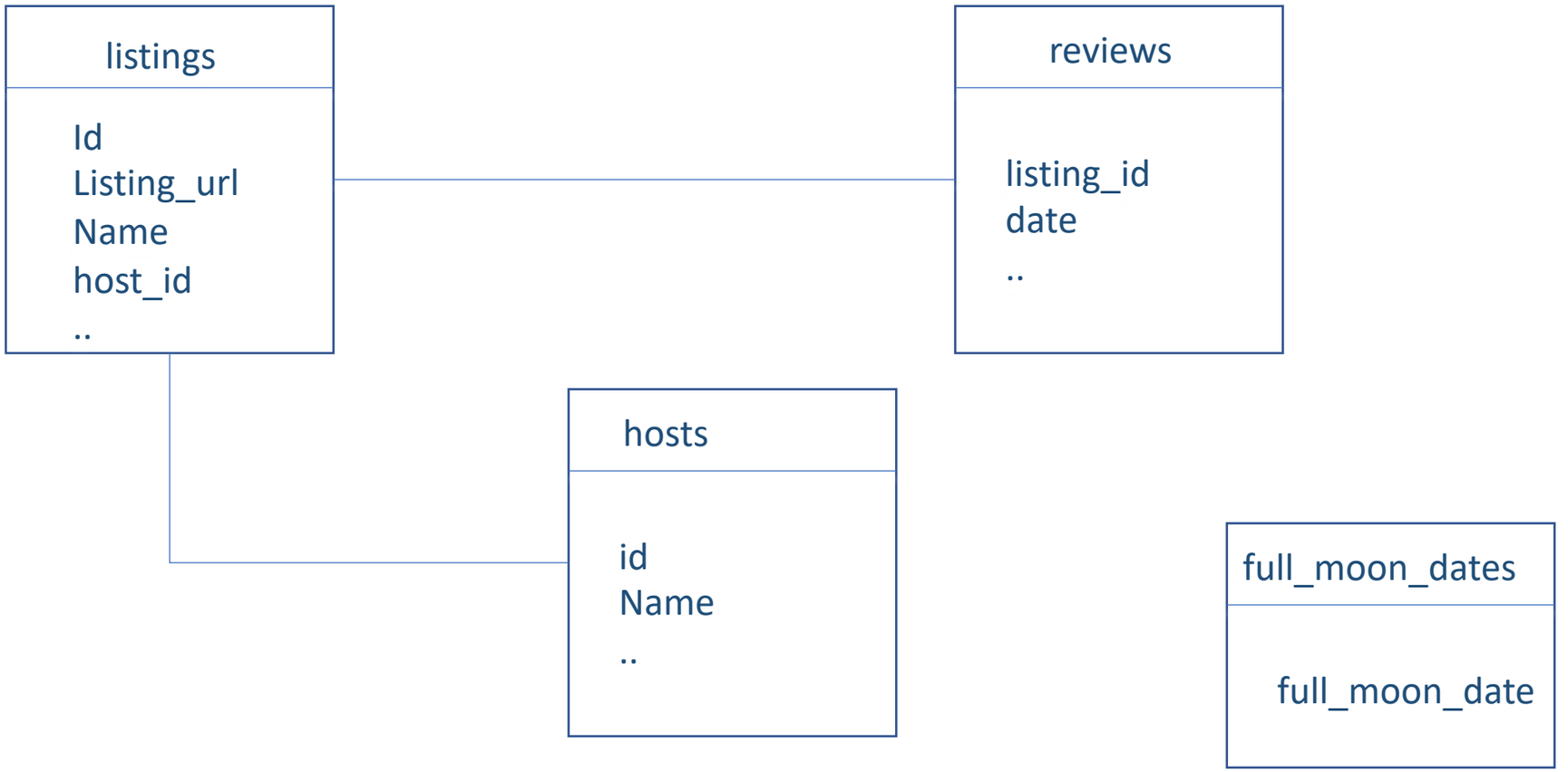
- Improve productivity
- High quality results
- Centralize the code
- Define data model
- Define tests
- Add monitoring and visibility

Big Data: DBT

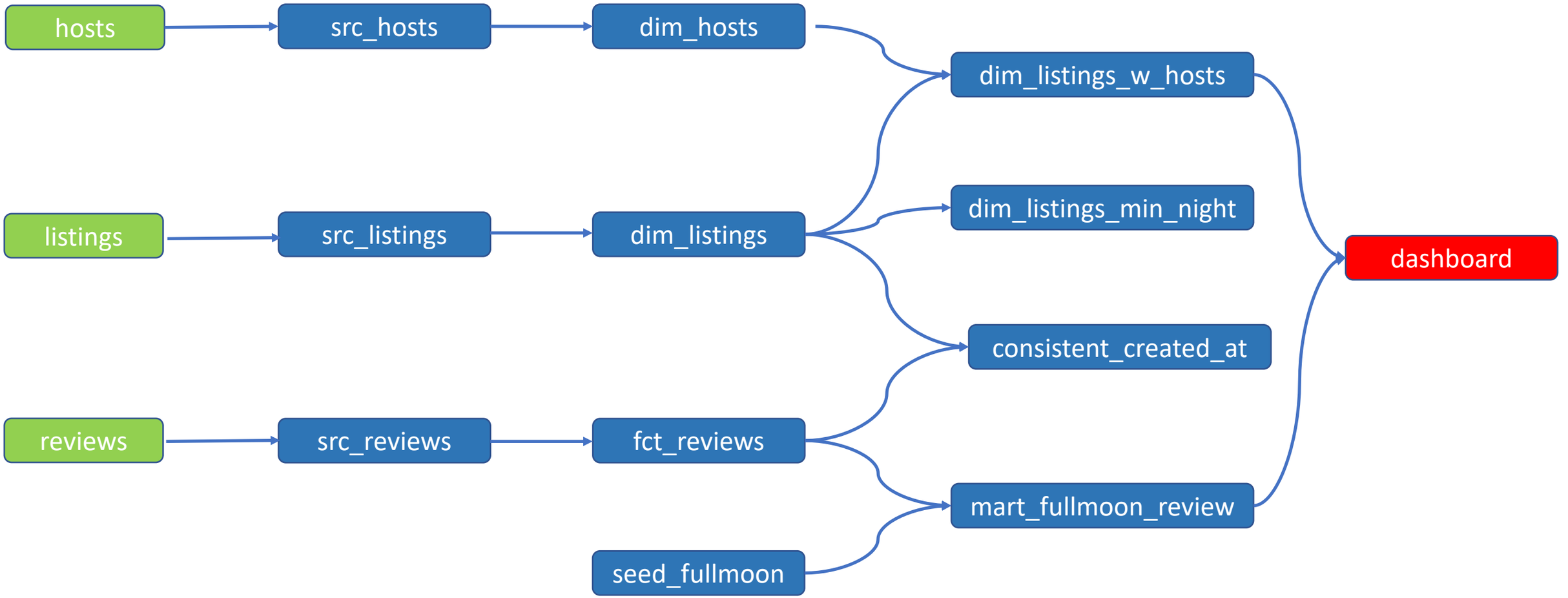

Data loader
Raw data
Snapshot
Transform
Test
Document
Transformed data
BI Tool
Data Science
Version control
Alerting
logging
dbt
Data Warehouse

Data Warehouse: Entities

We ‘ll study the impact of full moons on reviews



Big Data: Data flow



DBT: Setup

- Install python 3
- Create virtualenv
- Activate virtualenv
- Install dbt and dbt-snowflake-connector
    - pip install dbt-snowflake
        - [It will install the connector and the dbt core]

Data Warehouse: DBT

Models

- Basic building blocs of business logic
- Materialized as views, tables
- They live in SQL files in the `models` folder
- Models can reference each other and use templates

Common Table Expression

- Temporary result set that can be used in a SQL query
- CTEs break up complex queries into simpler blocks of code that can connect and build on each other

DBT: Staging Layer

```sql
WITH raw_listings AS (
    SELECT
       *
    FROM
       AIRBNB.RAW.RAW_LISTINGS
)
SELECT
       id AS listing_id,
       name AS listing_name,
       listing_url,
       room_type,
       minimum_nights,
       host_id,
       price AS price_str,
       created_at,
       updated_at
FROM
       raw_listings
```

DBT: Exercise

EXERCISE 1

Create a model which builds on top of our raw_hosts table.

1) Call the model models/src/src_hosts.sql
2) Use a CTE (common table expression) to define an alias called raw_hosts . This CTE select every column from the raw hosts table AIRBNB.RAW.RAW_HOSTS
3) In your final SELECT , select every column and record from raw_hosts and rename the following columns:
    - id to host_id
    - name to host_name

Execute `dbt run`

DBT: Exercise

EXERCISE 2

Create a model which builds on top of our raw_reviews table.

1) Call the model models/src/src_reviews.sql
2) Use a CTE (common table expression) to define an alias called raw_reviews . This CTE select every column from the raw hosts table AIRBNB.RAW.RAW_REVIEWS
3) In your final SELECT , select every column and record from raw_reviews and rename the following columns:
    - date to review_date
    - comments to review_text
    - sentiment to review_sentiment

Execute `dbt run`

DBT: Materialisation

View

Use it:
Lightweight representation
Data is not often used

Dont use it:
Data is read from the same model several times

Table

Use it:
You read from this data repeatedly

Dont use it:
Building single use model
Model is popultated incrementally

Incremental

Use it:
Fact tables
Appends to tables

Dont use it:
You want to update historical data

Ephemeral (CTE)

Use it:
You merly want an alias to you data
Appends to tables

Dont use it:
You read from the model several times

DBT: Core Layer

```
WITH src_listings AS (
       SELECT
       *
       FROM {{ ref('src_listings') }}
)
SELECT
       listing_id,
       listing_name,
       room_type,
       CASE
              WHEN minimum_nights = 0 THEN 1
              ELSE minimum_nights
       END AS minimum_nights,
       host_id,
       REPLACE(price_str,'$')::NUMBER(10,2) AS price,
       created_at,
       updated_at
FROM
       src_listings
```

DBT: Exercise

EXERCISE

Create a model in the `models/dim` folder called `dim_hosts.sql`.

1) Use a CTE to reference the `src_hosts` model
2) Select every column and every record and add a cleansing step to host_name

    if host_name is not null keep the original value

    if host_name is null, replace it withc Ànonymous`

    Use the NVL(column_name, default_value_name) function

Execute: `dbt run`

DBT: Core Layer

```
{{
        config(
                materialized = 'incremental’,
                on_schema_change='fail’
        )
}}
WITH src_reviews AS (
        SELECT * FROM {{ ref('src_reviews') }}
)
SELECT * FROM src_reviews
WHERE review_text is not null


{% if is_incremental() %}
AND review_date > (select max(review_date) from {{ this }})
{% endif %}
```

DBT: Exercise

```
SELECT * FROM "AIRBNB"."DEV"."FCT_REVIEWS" WHERE listing_id=3176;

INSERT INTO "AIRBNB"."RAW"."RAW_REVIEWS"
VALUES (3176, CURRENT_TIMESTAMP(), 'Zlatan', 'excellent stay!', 'positive');
```

DBT: Seeds

- Seeds are local files that you upload to data warehouse from dbt
- Source is an abstraction layer on top of your input table
- Source freshness can be checked automatically

- Download the following file:

https://dbtlearn.s3.us-east-2.amazonaws.com/seed_full_moon_dates.csv

Execute `dbt seed`

DBT: Mart Layer

```
{{ config(
        materialized = 'table',
) }}
WITH fct_reviews AS (
        SELECT * FROM {{ ref('fct_reviews') }}
),
full_moon_dates AS (
        SELECT * FROM {{ ref('seed_full_moon_dates') }}
)
SELECT
        r.*,
        CASE
                WHEN fm.full_moon_date IS NULL THEN 'not full moon’
                ELSE 'full moon’
        END AS is_full_moon
FROM
        fct_reviews r
LEFT JOIN full_moon_dates fm
ON (TO_DATE(r.review_date) = DATEADD(DAY, 1, fm.full_moon_date))
```

DBT: Sources

```yaml
version: 2
sources:
  - name: airbnb
    schema: raw
    tables:
      Generate model
      - name: listings
        identifier: raw_listings
      Generate model
      - name: hosts
        identifier: raw_hosts
      Generate model
      - name: reviews
        identifier: raw_reviews
        loaded_at_field: date
        freshness:
          warn_after: {count: 1, period: hour}
          error_after: {count: 24, period: hour}
```

DBT: Tests

- There are two types of tests: singular and generic
- Singular are SQL queries stored in tests which are expected to return an empty resultset
- There are four built-in generic tests:
  - unique
  - not_null
  - accepted_values
  - relationships

DBT: Tests

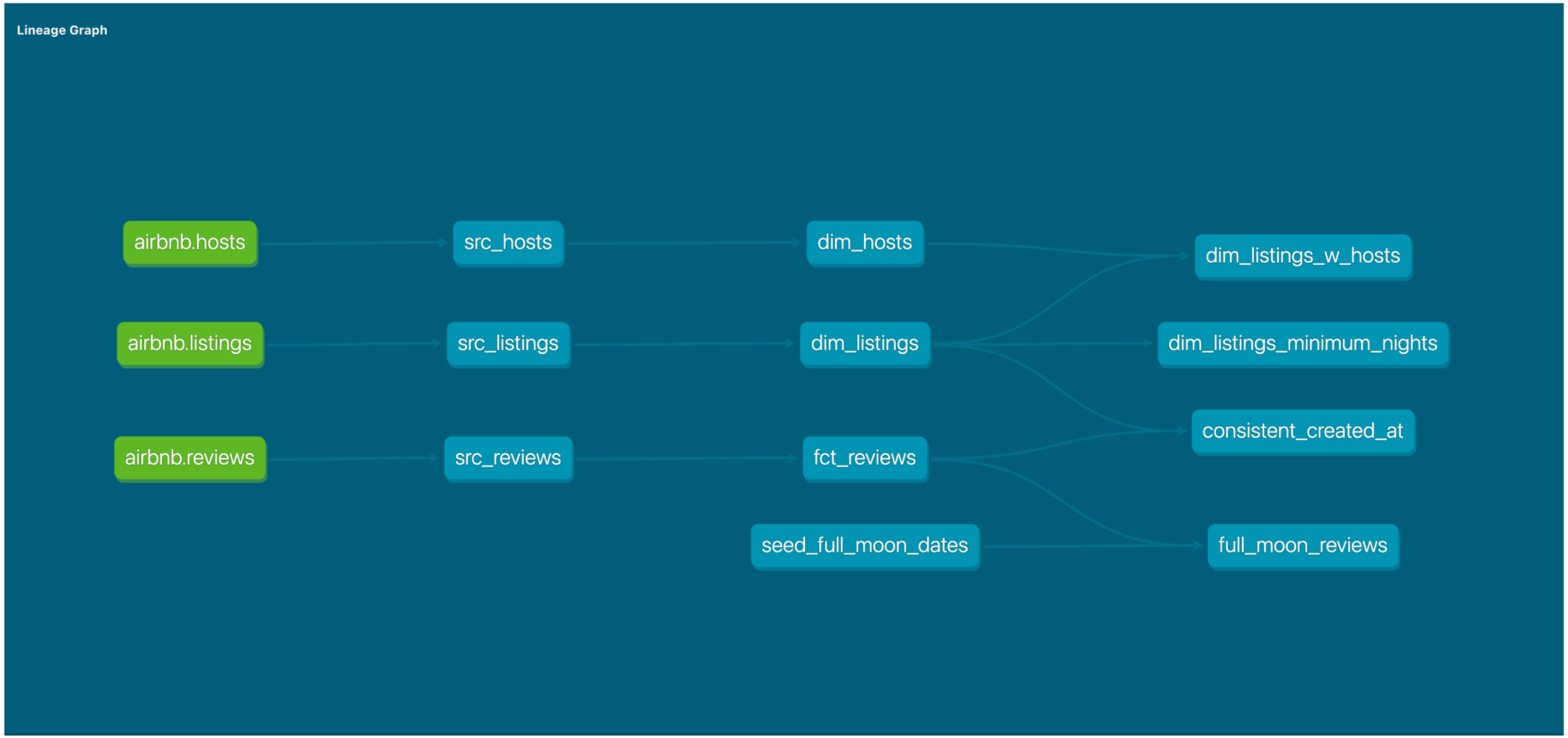
```
SELECT
      *
FROM
      {{ ref('dim_listings') }}
WHERE minimum_nights < 1
LIMIT 10
```

DBT: Tests

EXERCISE

Create a singular test in *tests/consistent_created_at.sql* that checks that there is no review date that is submitted before its listing was created: Make sure that every review_date in fct_reviews is more recent than the associated created_at in dim_listings .

DBT: Docs


Lineage Graph
airbnb.hosts
airbnb.listings
airbnb.reviews
src_hosts
src_listings
src_reviews
dim_hosts
dim_listings
fct_reviews
seed_full_moon_dates
dim_listings_w_hosts
dim_listings_minimum_nights
consistent_created_at
full_moon_reviews

DBT: commands

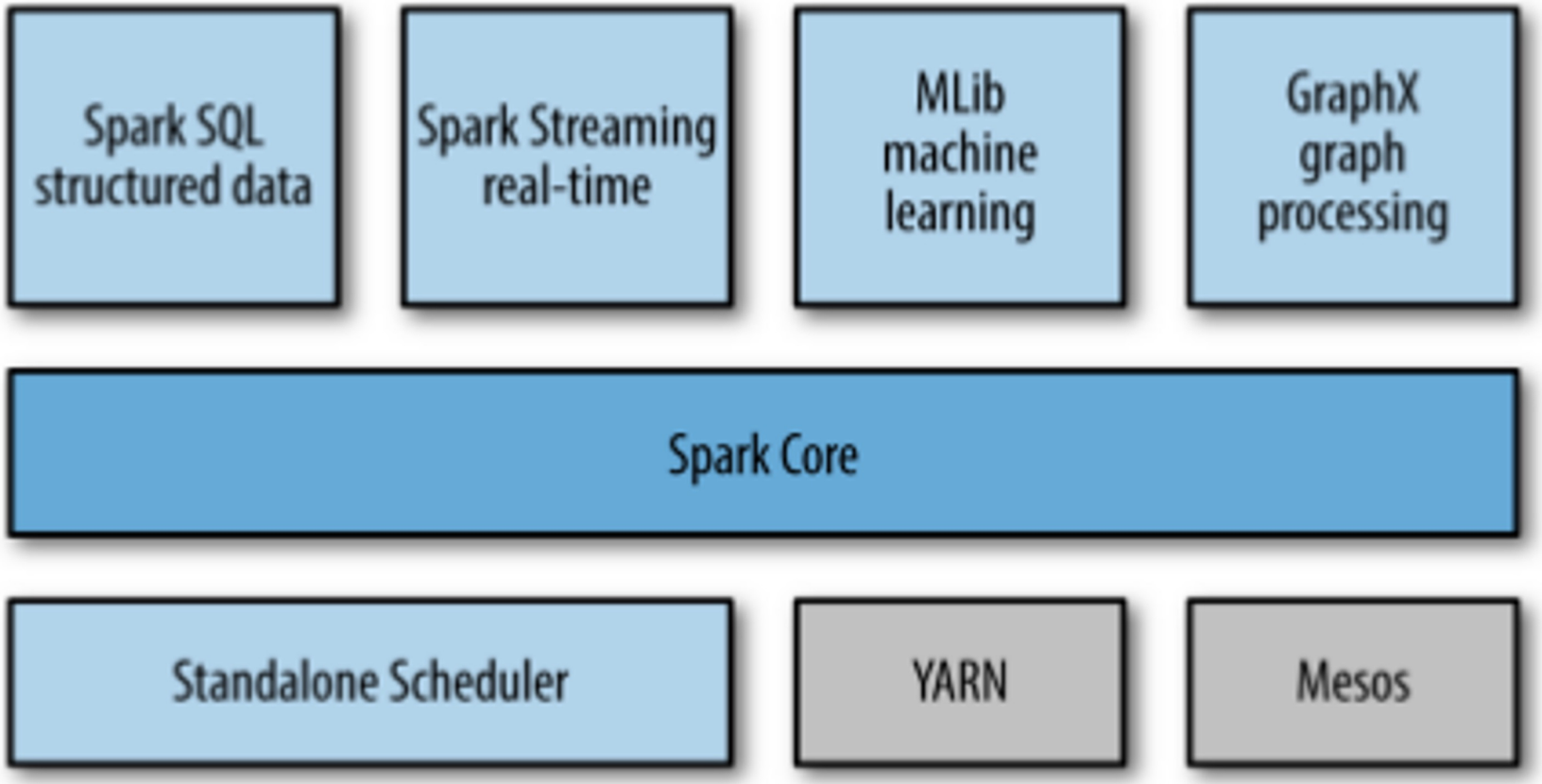
```
dbt debug

dbt compile

dbt run

dbt seed

dbt source freshness

dbt test

dbt docs generate

dbt docs serve
```

Apache Spark

Apache Spark is a fast cluster computing framework which is used for processing, querying and analyzing Big data. Being based on In-memory computation, it has an advantage over several other Big Data Frameworks.

- **Speed:** It is 100x faster than traditional large-scale data processing frameworks
- **Powerful Caching:** Simple programming layer provides powerful caching and disk persistence capabilities
- **Deployment:** Can be deployed through Mesos, Hadoop via Yarn, or Spark's own cluster manager
- **Real Time:** Real-time computation & low latency because of in-memory computation
- **Polyglot:** Supports programming in Scala, Java, Python and R

Spark Architecture



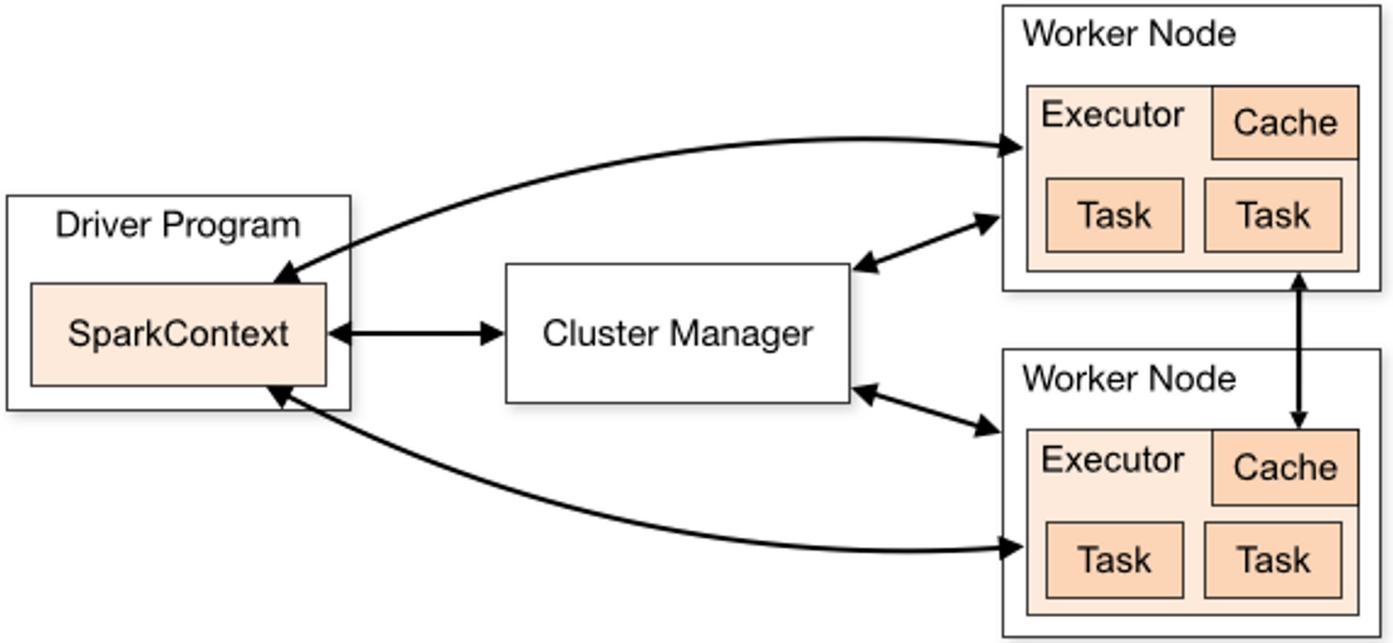
Spark Ecosystem

Spark Streaming is a scalable, fault-tolerant system that follows the RDD batch paradigm. It is basically operated in mini-batches or batch intervals which can range from 500ms to larger interval windows.

Spark SQL is a higher-level abstraction module over the Spark Core. It is majorly used for processing structured and semi-structured datasets. It also provides an optimized API that can read the data from the various data source containing different files formats.

Spark MLlib is a machine-learning library. It is a wrapper over Spark Core to do data analysis using machine-learning algorithms. It works on distributed systems and is scalable. We can find implementations of classification, clustering, linear regression, and other machine-learning algorithms in Spark MLlib.

Spark Workflow



Resilient Distributed Datasets (RDDs)

RDDs are the building blocks of any Spark application. RDDs Stands for:

- ***Resilient:*** It is fault tolerant and is capable of rebuilding data on failure.
- ***Distributed:*** Data is distributed among the multiple nodes in a cluster.
- ***Dataset:*** Collection of partitioned data with values.

It is a layer of abstracted data over the distributed collection. It is immutable in nature and follows *lazy transformations*.

With RDDs, you can perform two types of operations:

- **Transformations:** These operations are applied to create a new RDD.
- **Actions:** These operations are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

Spark RDD

- ***In-Memory Computations:*** It improves the performance by an order of magnitudes.
- ***Lazy Evaluation:*** All transformations in RDDs are lazy, i.e, doesn't compute their results right away.
- ***Fault Tolerant:*** RDDs track data lineage information to rebuild lost data automatically.
- ***Immutability:*** Data can be created or retrieved anytime and once defined, its value can't be changed.
- ***Partitioning:*** It is the fundamental unit of parallelism in PySpark RDD.
- ***Persistence:*** Users can reuse PySpark RDDs and choose a storage strategy for them.
- ***Coarse-Grained Operations:*** These operations are applied to all elements in data sets through maps or filter or group by operation.

https://spark.apache.org/docs/latest/rdd-programming-guide.html

Spark RDD Workflow

Spark RDD Input:Outpput

Function name	Purpose	Example
sc.parallelize	Transform a python list to an RDD	list = [1,2,3,4] rdd = sc.parallelize(list)
sc.textFile	Create an RDD from a text file	file_name = /documents/book.txt rdd = sc.textFile(file_name)
saveAsTextFile	Save the RDD as a text file	file_name = /documents/book.txt rdd.saveAsTextFile(file_name)

Spark RDD Transformations

Function name	Purpose	Example	Result
map	Apply a function to each element in the RDD and return an RDD of the result	rdd = [1,2,3,4] rdd.map(lambda x: x +1)	[2,3,4,5]
flatMap	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned	rdd = [[1,2],[3,4]] rdd.flatMap(lambda x: x +1)	[2,3,4,5]
filter	Retrun an RDD consitent of only elements that pass the condition	rdd = [1,2,3,4] rdd.filter(lambda x: x != 1)	[2,3,4]
distinct	Remove duplicates	rdd = [1,2,2,3] rdd.distinct()	[1,2,3]
sortBy	Sort the element of an RDD	rdd = [3,2,4,1] rdd.sort()	[1,2,3,4]
reduceBykey	Apply a transformation and return a new RDD based on the reduce function	rdd = [(Paris, 13), (Paris, 24), (Lyon,12), (Lille, 40)] rdd.reduceByKey(lambda a, b: a + b)	[(Paris, 37), (Lyon, 12), (Lille, 40)]

Spark RDD Actions

Function name	Purpose	Example	Result
collect	Return all the elements of the dataset as an array at the driver program	rdd = [1,2,3,4] rdd.collect()	[1,2,3,4]
take	Return an array with the first *n* elements of the dataset.	rdd = [1,2,3,4] rdd.take(2)	[1,2]
first	Return the first element of the dataset (similar to take(1)).	rdd = [1,2,3,4] rdd.first()	1
count	Return the number of elements in the dataset.	rdd = [1,2,3,4] rdd.count()	4
reduce	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one)	rdd = [1,2,3,4] rdd.reduce(lambda a, b: a + b)	10

Spark Dataframe

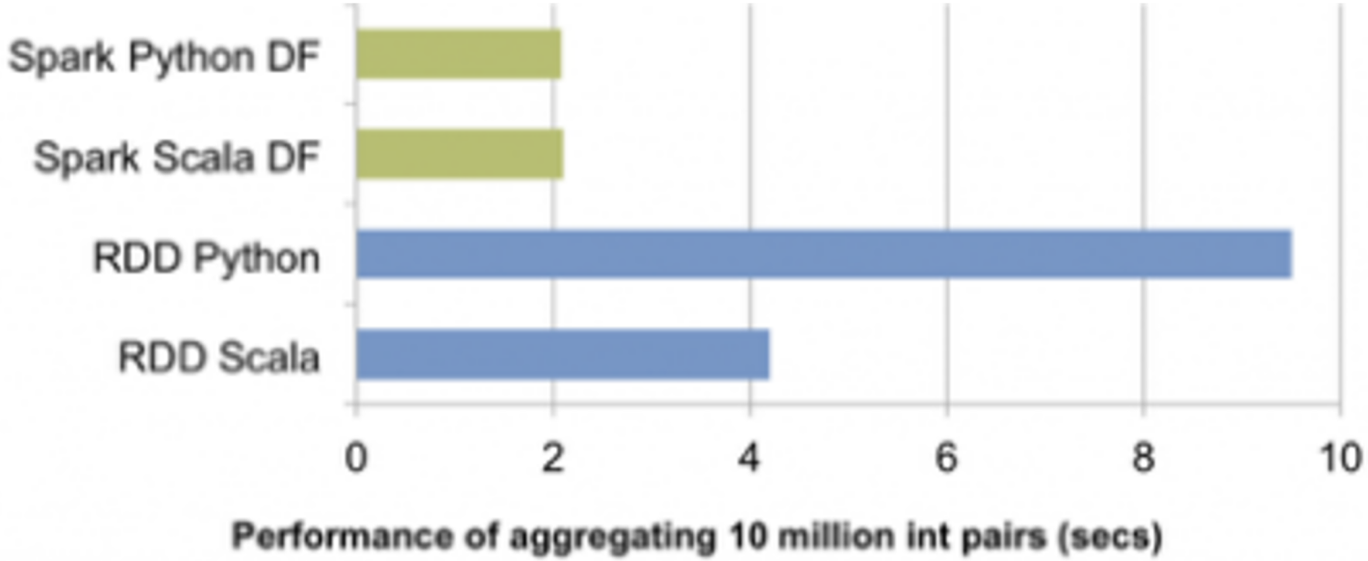
Dataframe in Spark is the distributed collection of structured or semi-structured data.

This data in Dataframe is stored in rows under named columns which is similar to the relational database tables or excel sheets.

It also shares some common attributes with RDD like Immutable in nature, follows lazy evaluations and is distributed in nature. It supports a wide range of formats like JSON, CSV, TXT and many more. Also, you can load it from the existing RDDs or by programmatically specifying the schema.

Spark Dataframe vs RDD

Faster than RDDs



Benefits from Catalyst optimizer