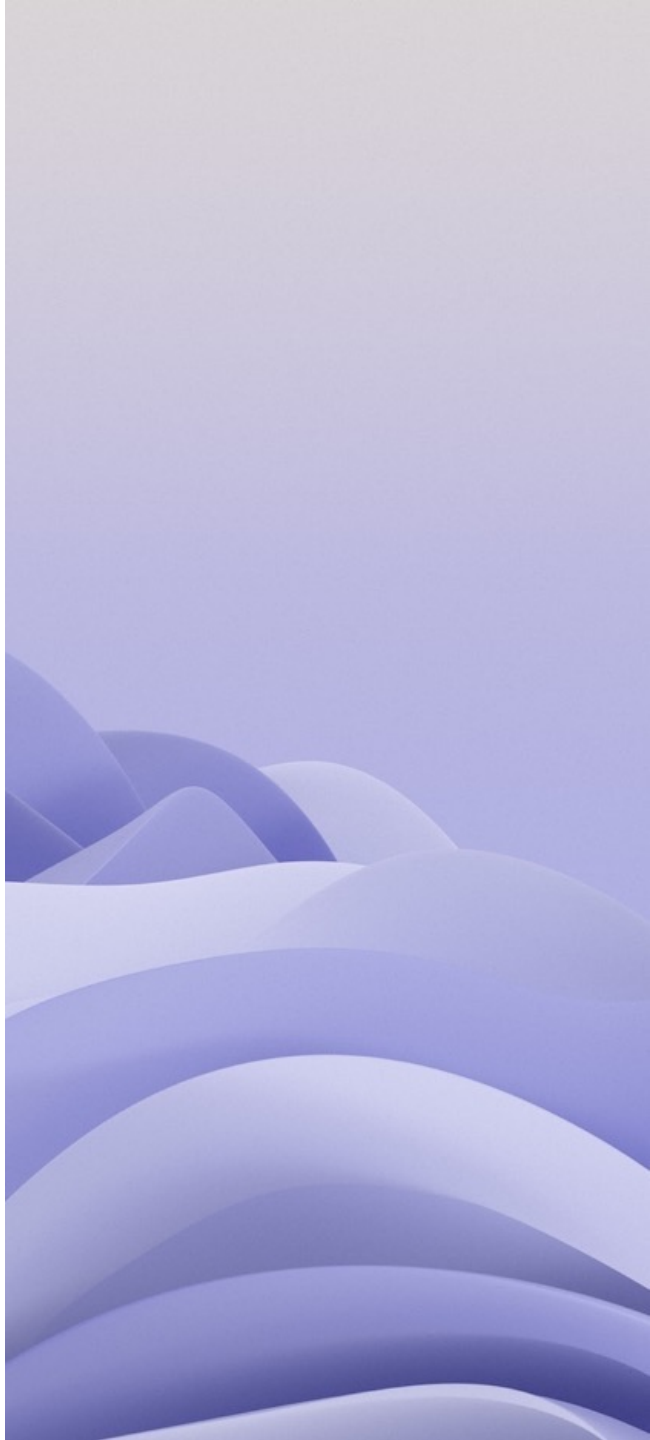
Tuesday 31st October 2023

Big Data

Data Mining, warehouse, viz, nosql applied to health data



20231003 – #V1

Speaker introduction

Simon Coulet

Senior Data Engineer

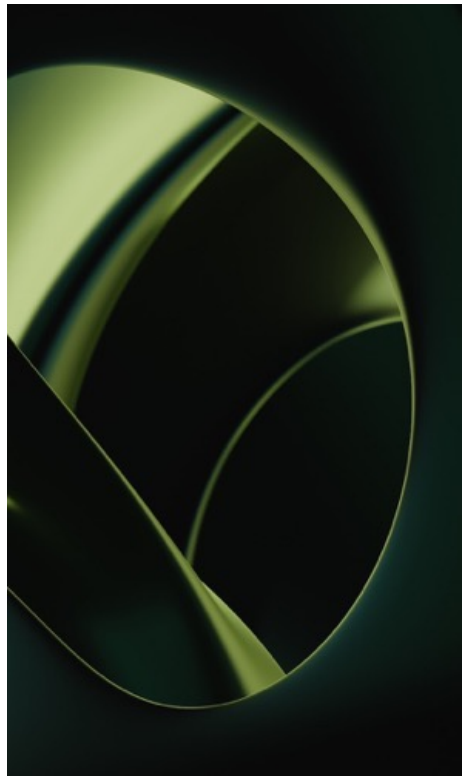
simon.coulet@saegus.com
LinkedIn : Simon Coulet

Agenda

Big Data fundamentals

Data storage

Batch processing



Introduction to distributed computation

Big data fundamentals

sægus
SMART SHIFT · FOR REAL

Why ?

Decision, Analytics, Scalability

Decision

Growing importance of data in decisions and innovation

Analytics

Frameworks to support advanced analytics and AI

Scalability

Adapt to growing demand, elasticity of resources

Big Data vs Data science

Data Science

Looks to create models that capture the underlying patterns of complex systems, codifies models into working applications

Big Data

Collects and manages large amount of varied data to serve large-scala web application and vast sensor networks

Collecting does not mean discovering !

Context

Decision, Analytics, Scalability

Technology evolution

Distributed computing revolution

ITjobs

- Data Engineers
- Data Scientist
- DevOps

ITjobs

- Data Engineers
- Data Scientist
- DevOps

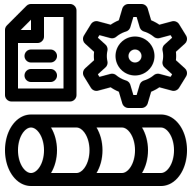
Gouvernance

Security, confidentiality, legal, ethic

Data jobs

A comparison

Data Engineer



Designs, builds, maintains the infrastructure that enables Data Analysts and Data Scientists to perform their work

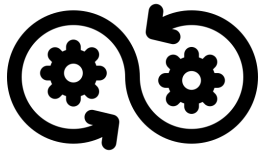
Data scientist

Uses statistical and ML techniques to build predictive models and uncover insights from data

Data analyst (BI)

Analyzes data to identify patterns, trends, and insights that can inform business decisions via dashboards

DevOps



Develops and maintains the software development lifecycle, including continuous integration, continuous delivery, and continuous deployment

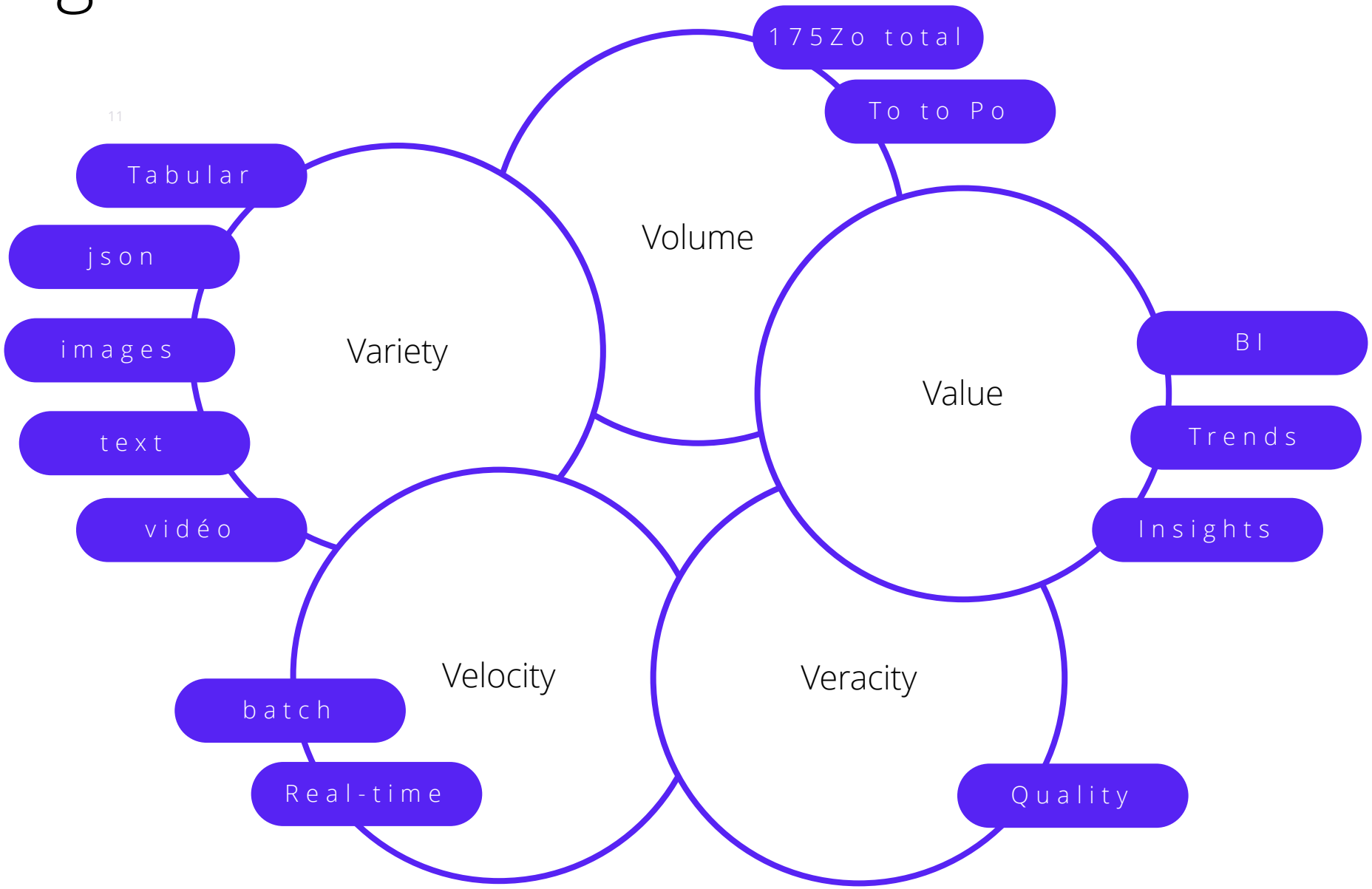
ML Engineer

Builds and deploy ML at scale, industrializes data scientist's work

DataOps

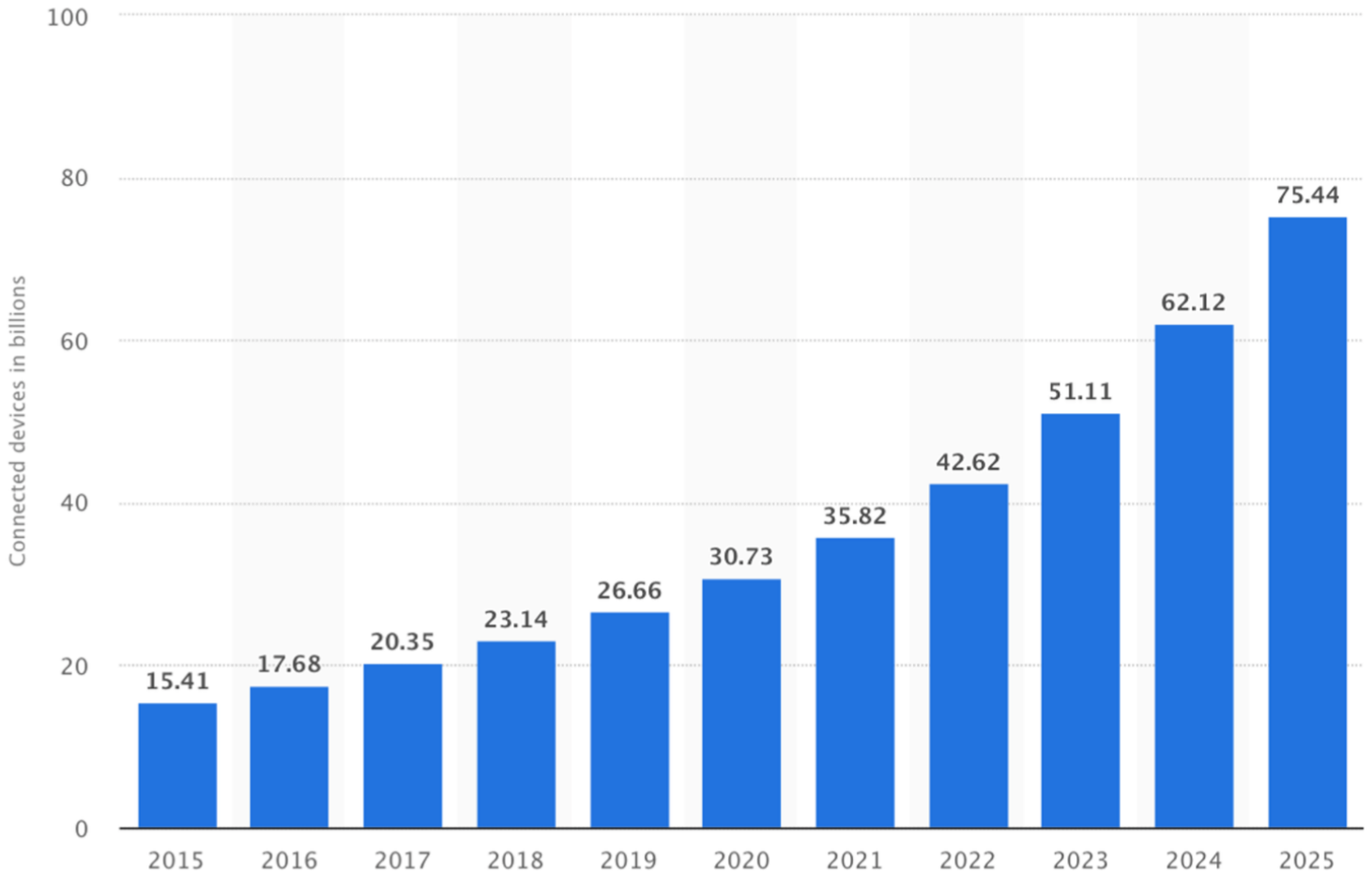
Manages end-to-end data pipelines, from ingestion to delivery, using devops principles

5 V of Big Data



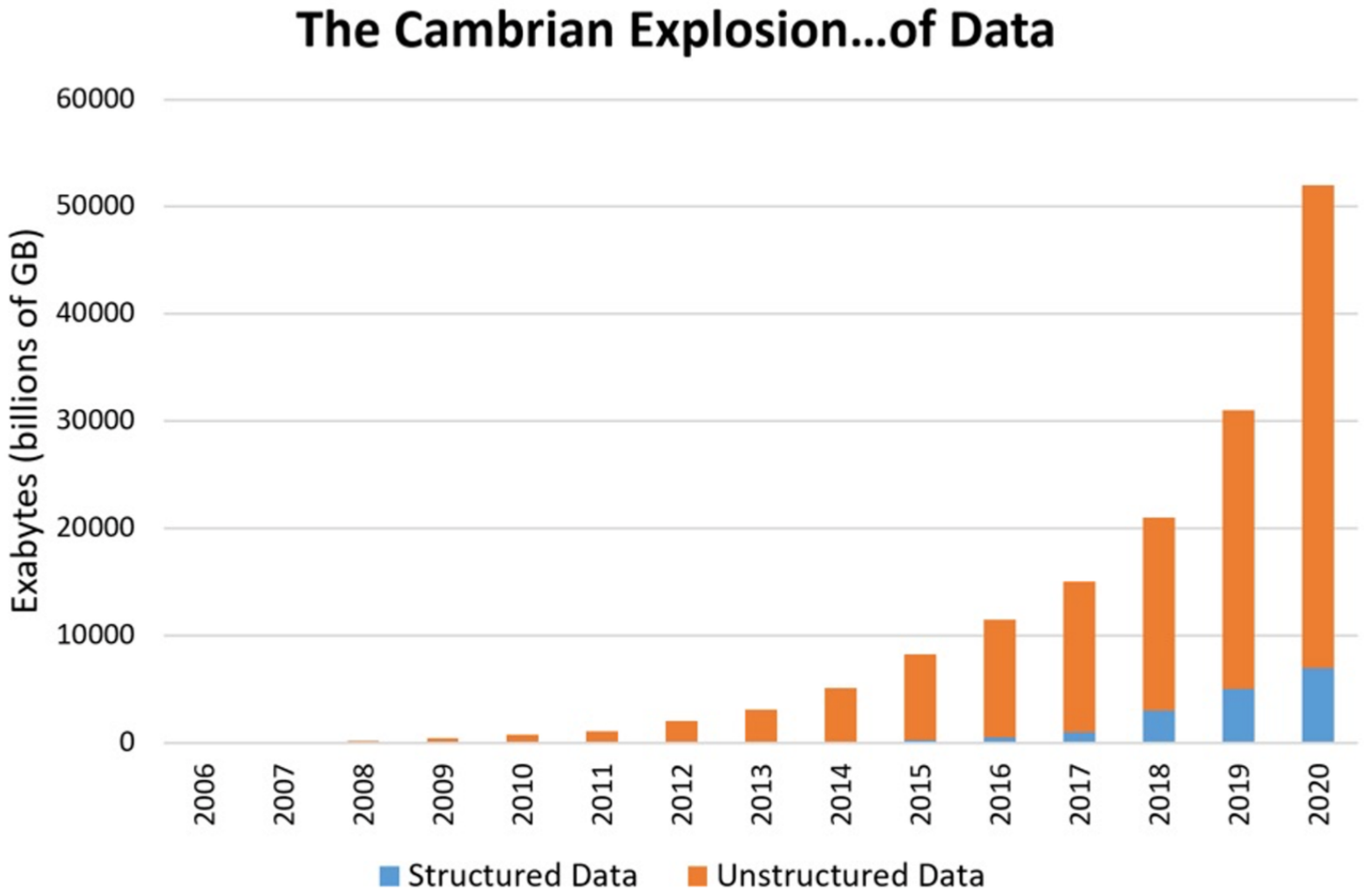
IoT evolution

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



Big data volume evolution

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



Challenges : velocity

Decision, Analytics, Scalability

Web navigaton

User tracking, recommendation systems

Streaming

Real time procesisng for data in motion

Example

400Go per hour in my mission

Challenges : variety

Decision, Analytics, Scalability

Structured

Postgre, csv, parquet

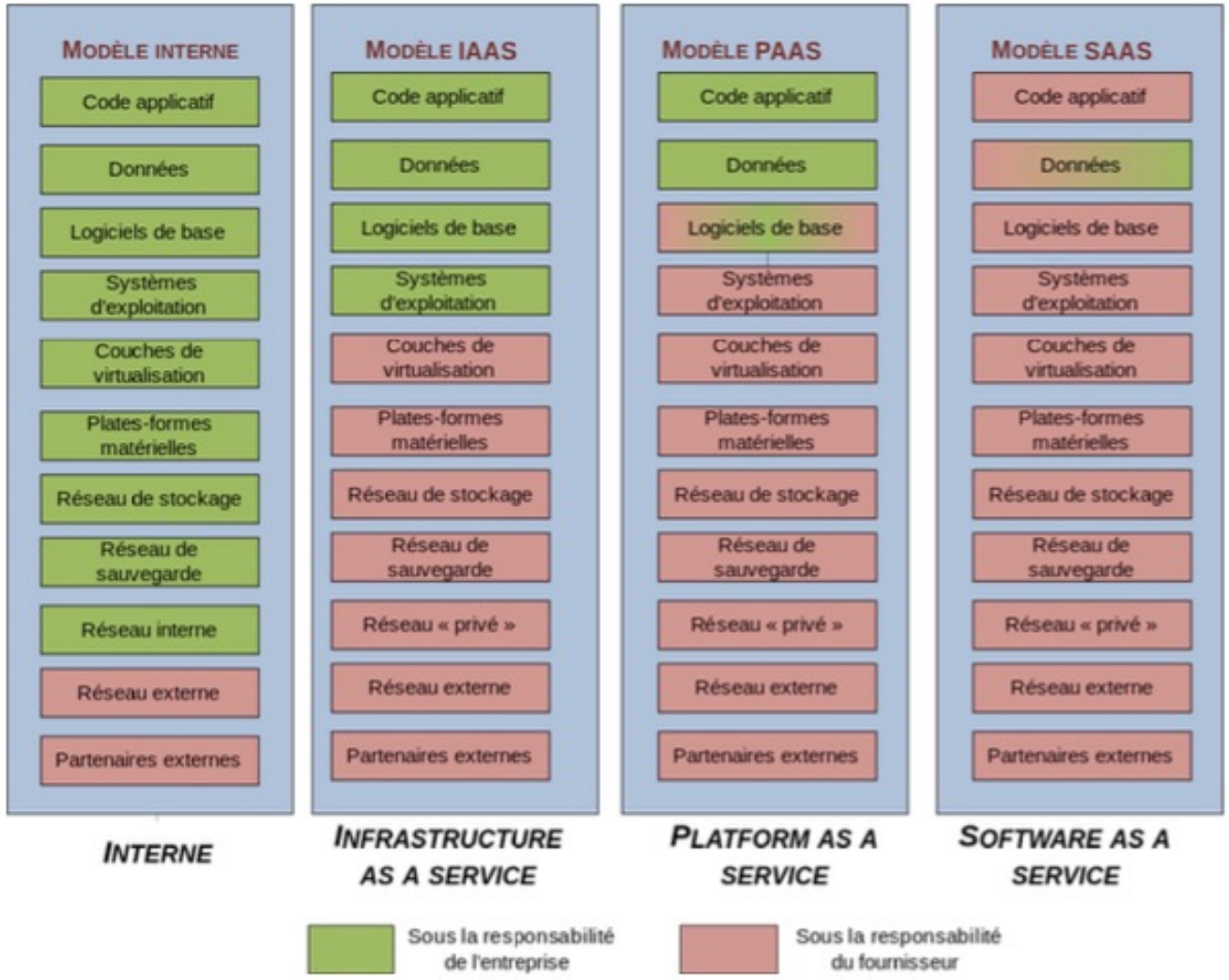
Semi-structured

Xml, json, logs

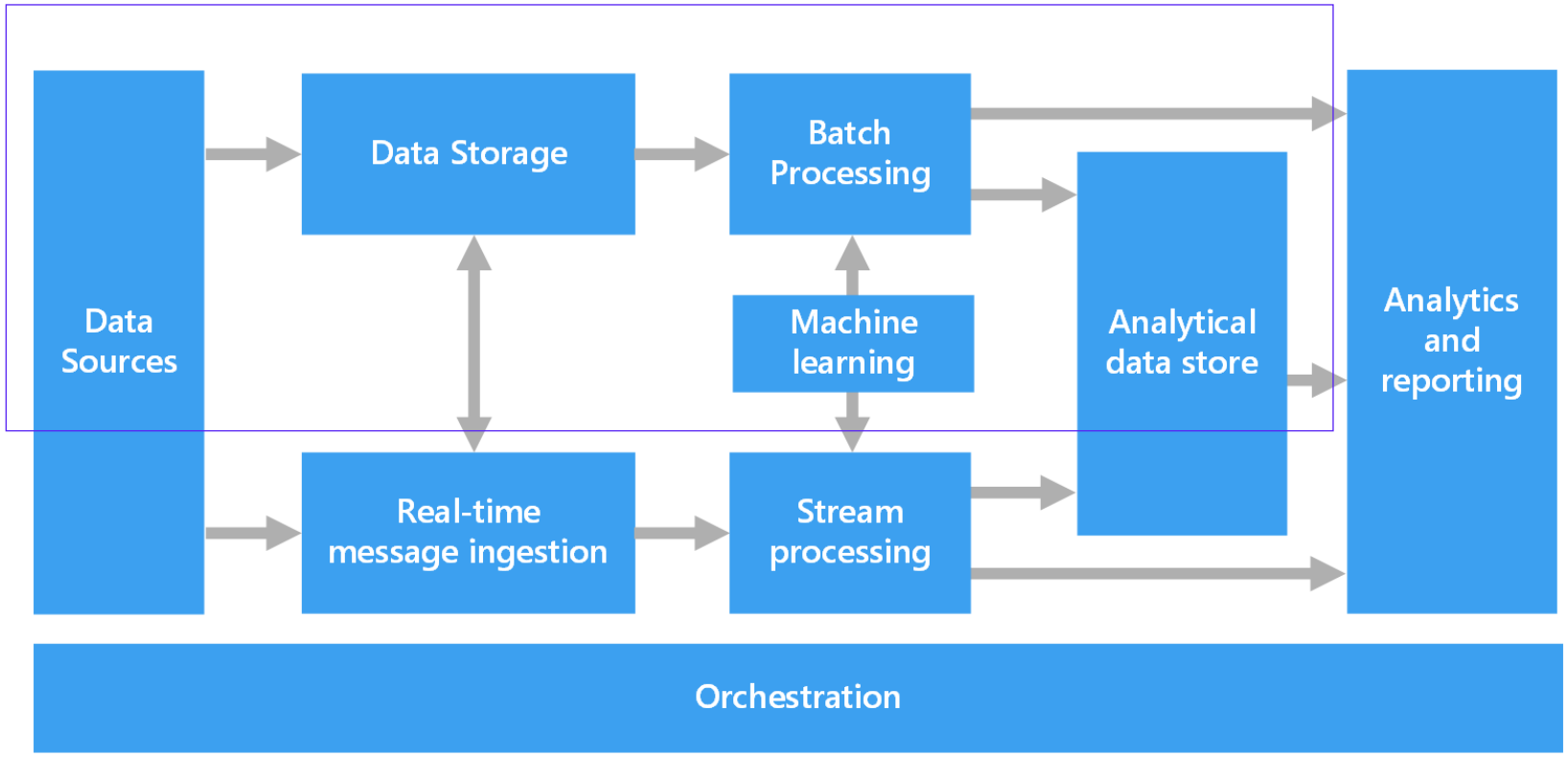
Non-structured

pdf, images, videos

Cloud Computing



Big Data Architecture



Data Storage

saegus
SMART SHIFT · FOR REAL

Data Storage

Referred as « Data Lake »

Google Cloud Storage

amazon S3

Object Naming and Key/Value Pair

Utilize object naming of easy identification and key/value pairs for detailed metadata storage. Employ bucket prefixes for efficient data organization and retrieval.

Scalable

Easily increase or decrease the amount of storage you need as your data grows or shrinks

Support massive amounts of data storage, and you can upgrade the service plan without moving any data from one location to another.

Cost-efficiency:

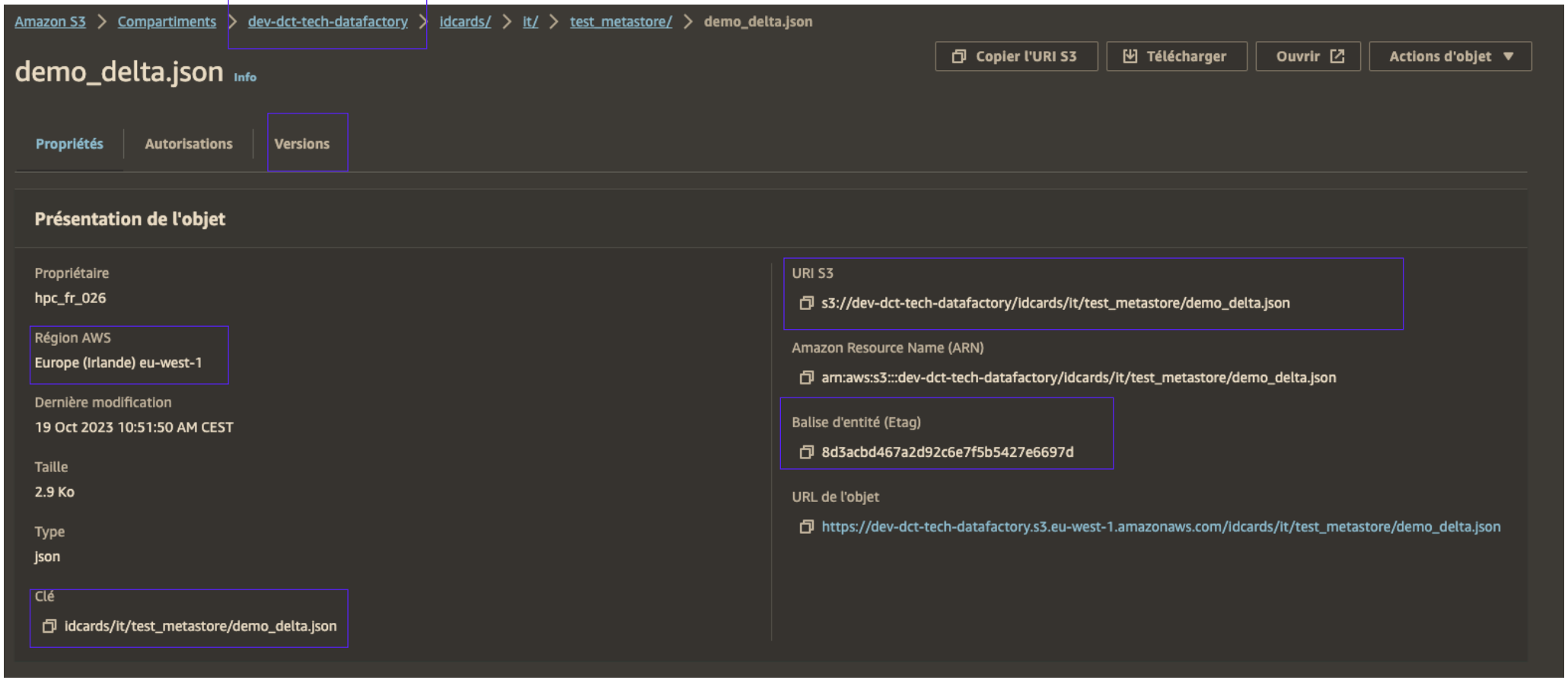
Cost-effective storage options, and you only pay for the storage and data transfer you use.

Pay-as-you-go pricing models, which means that you can scale up and down with demand and pay only for what you use.

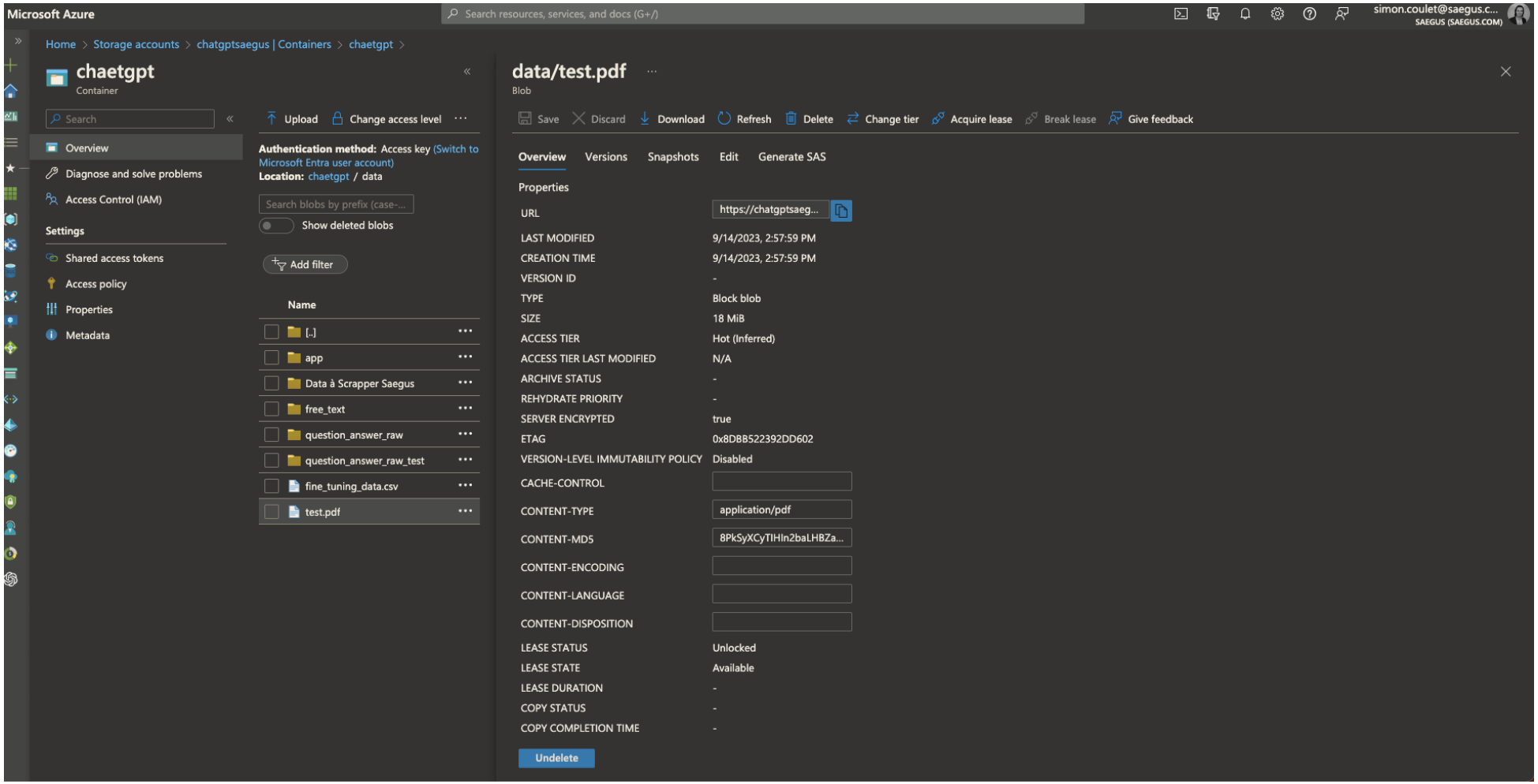
Security

Secure storage options, and the cloud computing provider handles the management and security of the storage servers, infrastructure, and networks to guarantee access to data at any time with elastic capacity.

AWS S3 Example :



Azure Blob storage Example :



Data Storage

Data Lake vs Data Warehouse

Datalake

- Raw
- Massive storage
- Usage not yet defined
- Data in native format
- Flexible, no-schema yet
- Cheap to store, cold/warm tier

amazon S3

Google Cloud Storage

Microsoft Azure Blob Storage

Data Warehouse

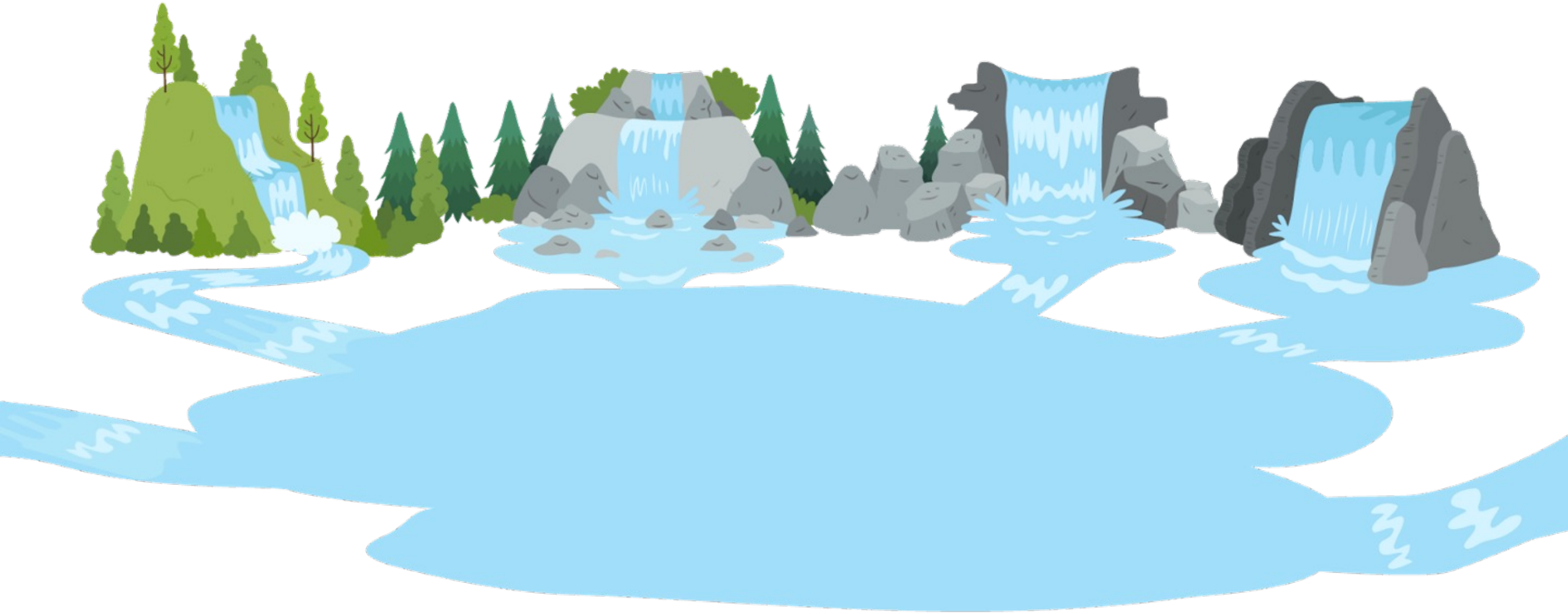
- Calculated, refined
- Storage + computation power
- Optimized for SQL-based querying
- Data is standardized
- Higher costs

amazon REDSHIFT

Google Big Query

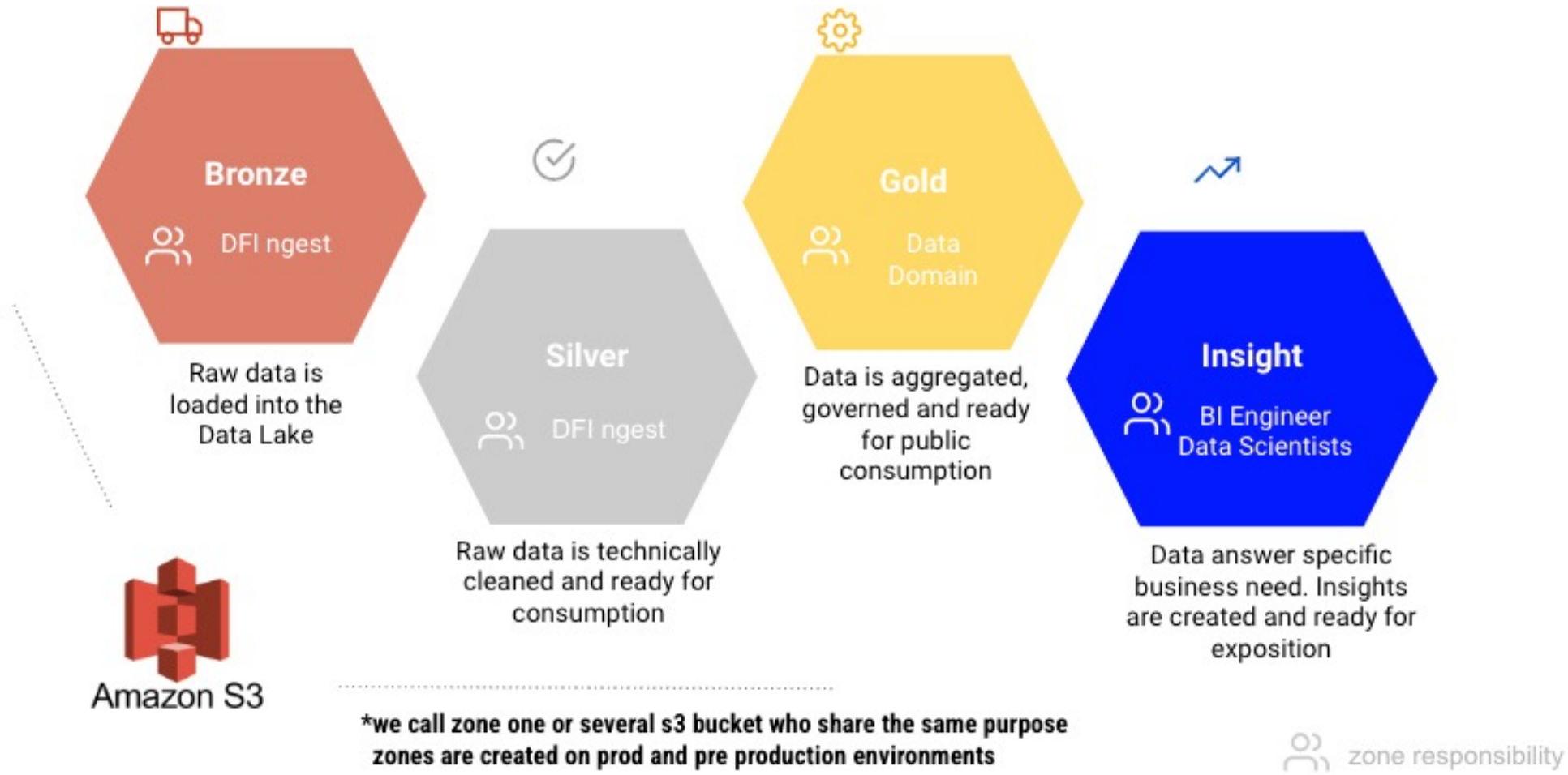
snowflake

Data Lake

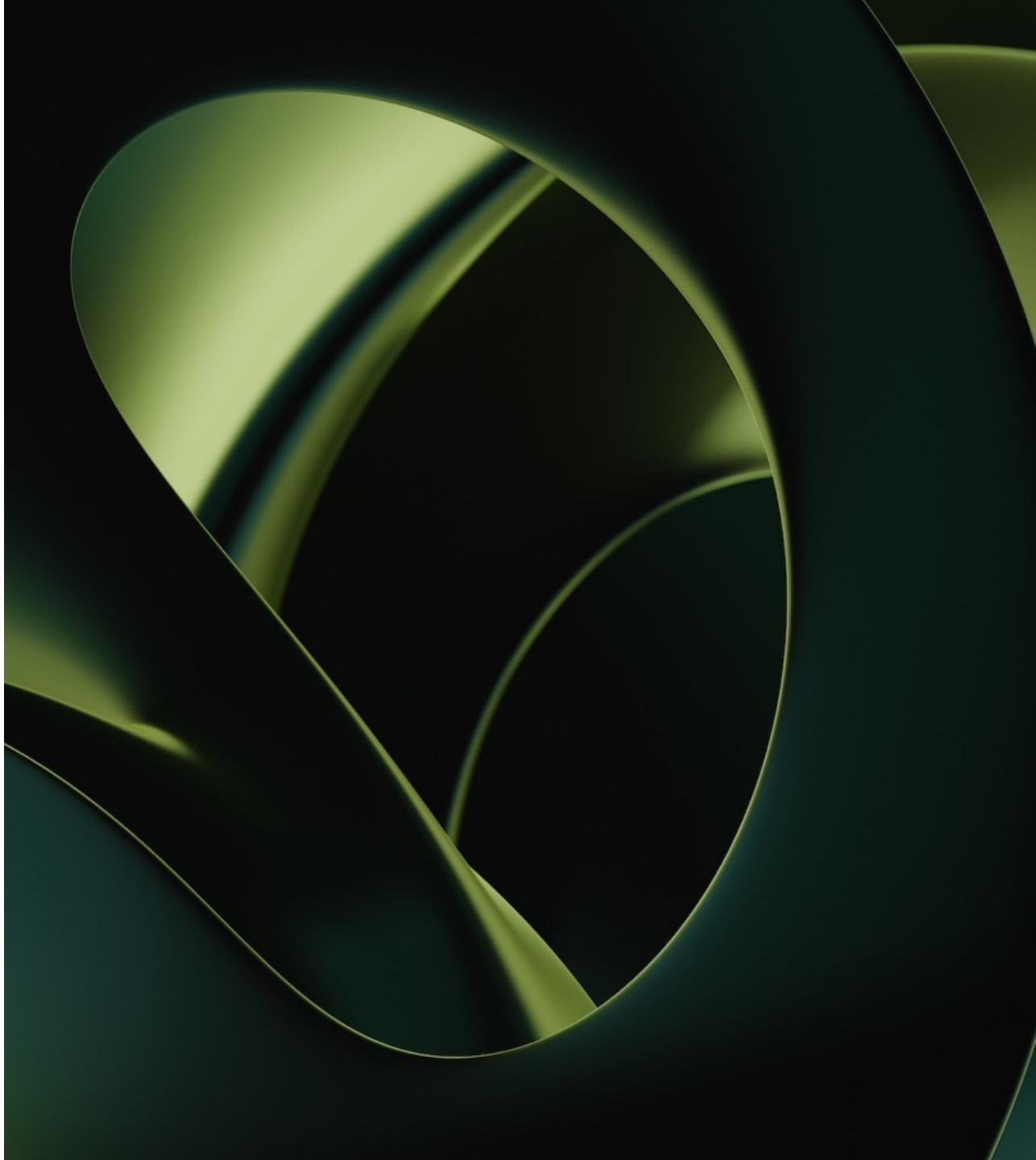


Data Storage : An example

Different zones



Introduction to batch processing



sægus
SMART SHIFT · FOR REAL

Batch Processing

Definition

Batch processing involves executing data processing jobs in groups, or "batches," ideal for large, complex datasets that do not require immediate analysis or response.

Why ?

Cost-effective for processing high volumes of data, where real-time analysis is not critical, allowing for efficient resource utilization during off-peak hours.

Common use-cases

Frequently used for end-of-day reports, monthly billing runs, and scenarios where data is collected over time before processing.

Benefits

Offers robustness and reliability for repetitive tasks, with errors in one batch not necessarily halting the entire process, enabling better error handling and recovery.

Batch Processing

ETL vs ELT : def of E, T, L

Extract

Data is collected from various DBs, CRM, social medias

-> NiFi, AWS Glue, Talend

Transform

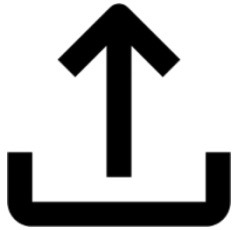
Clean, standardized, enriched, checked crossed, based on business needs

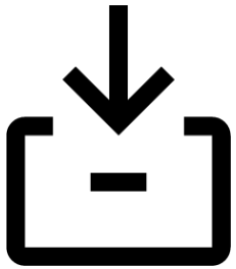
-> Spark, dbt, Talend

Load

Load in target system

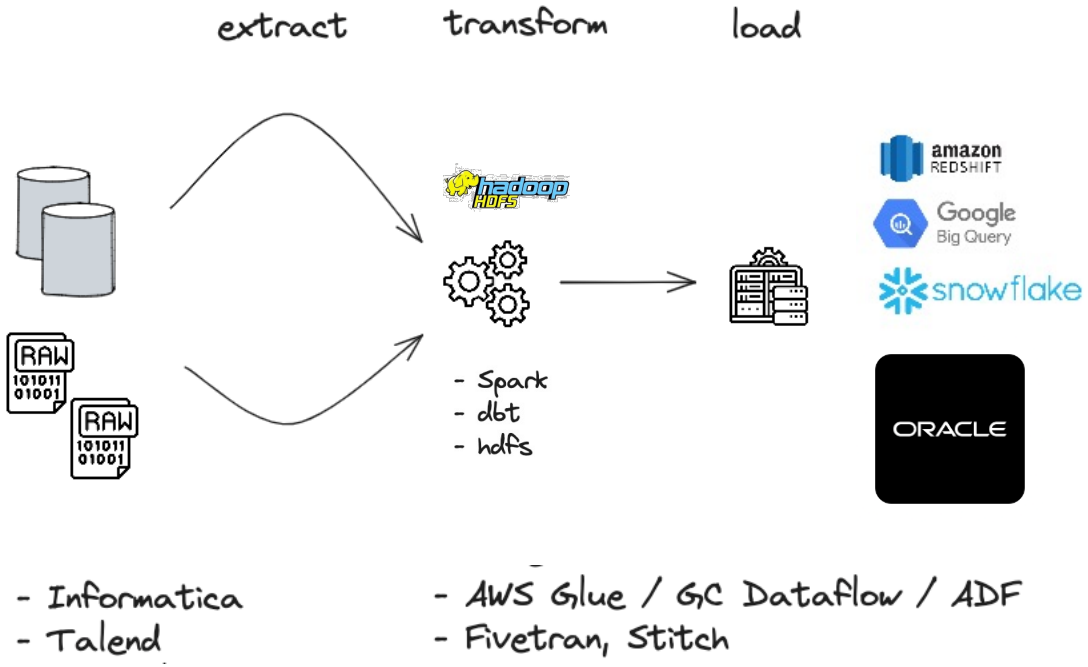
➔ Datalake or Data Warehouse





Batch Processing

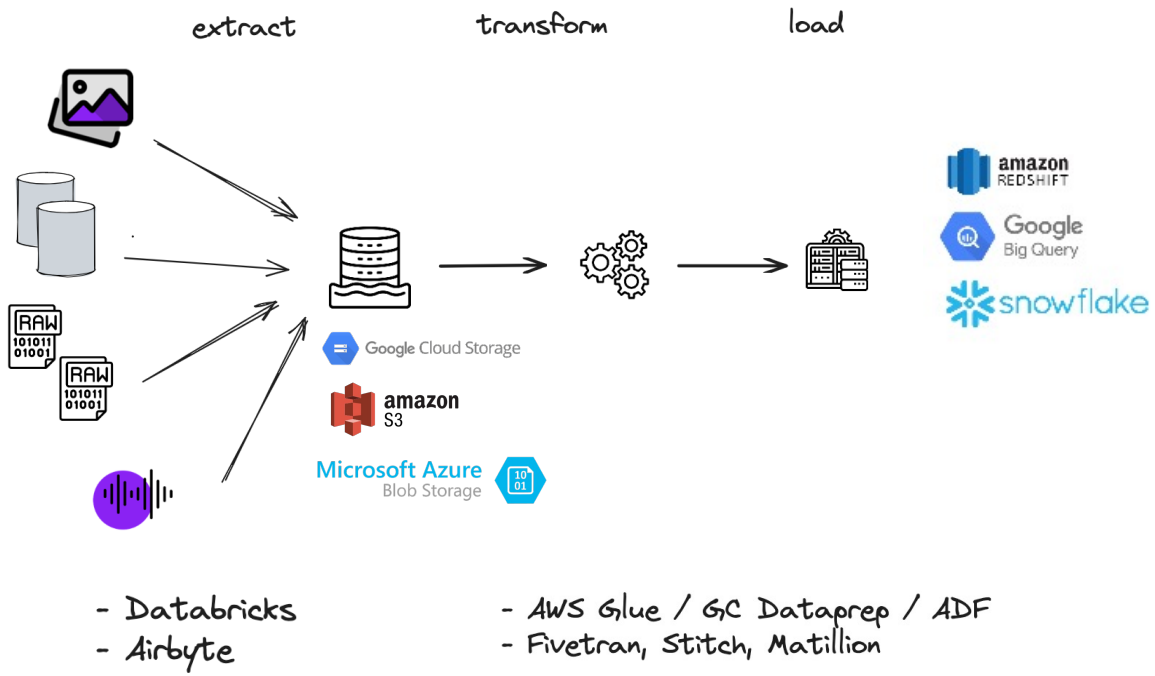
ETL vs ELT : ETL



- Most used approach since the 70s
- Reliable and control
- Solutions at high cost Informatica, Talend, Oracle's etl...
- On-promise focus
- Rigidity in data processing
- Structured data emphasis

Batch Processing

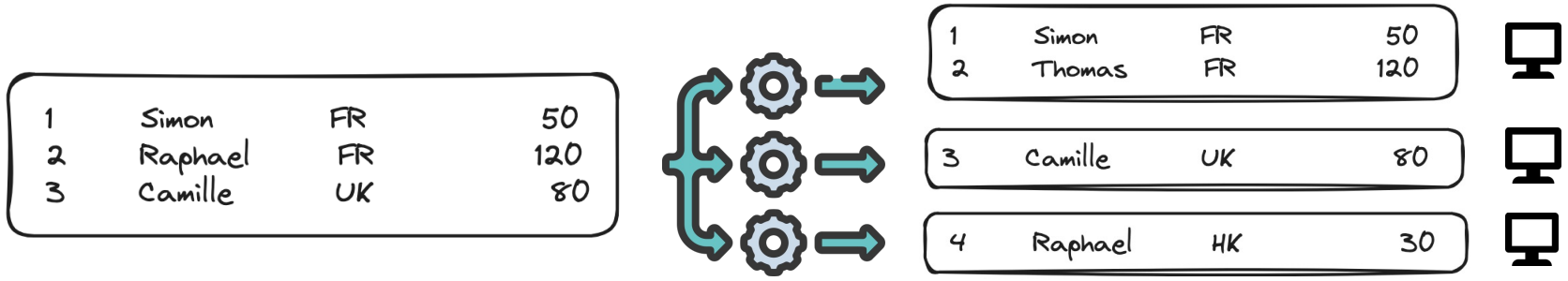
ETL vs ELT : ELT



- Emerging popular approach
- More scalable, data stored in datalake for cheap, transformation applied on demand
- Solutions at high cost Informatica, Talend, Oracle's etl...
- Cloud-Centric
- Business flexible
- Untructured data support
- Real-time is easier since available in datalake
- Data governance may be hard

Principles of Distributed Computing

Parallel processing



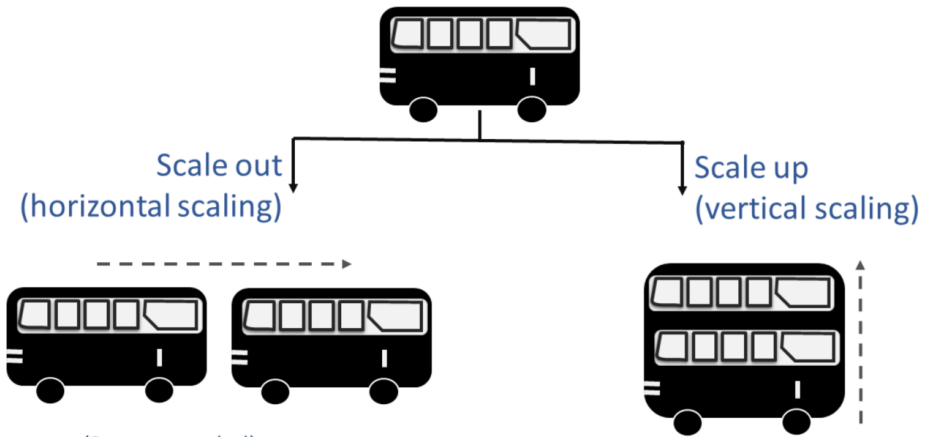
Breaking down complex operations into smaller, independent tasks that can be executed simultaneously across different processor or machines

TODO !

+ fault tolerance
+ scale in vs scale out
+ MapReduce
+ resource manager

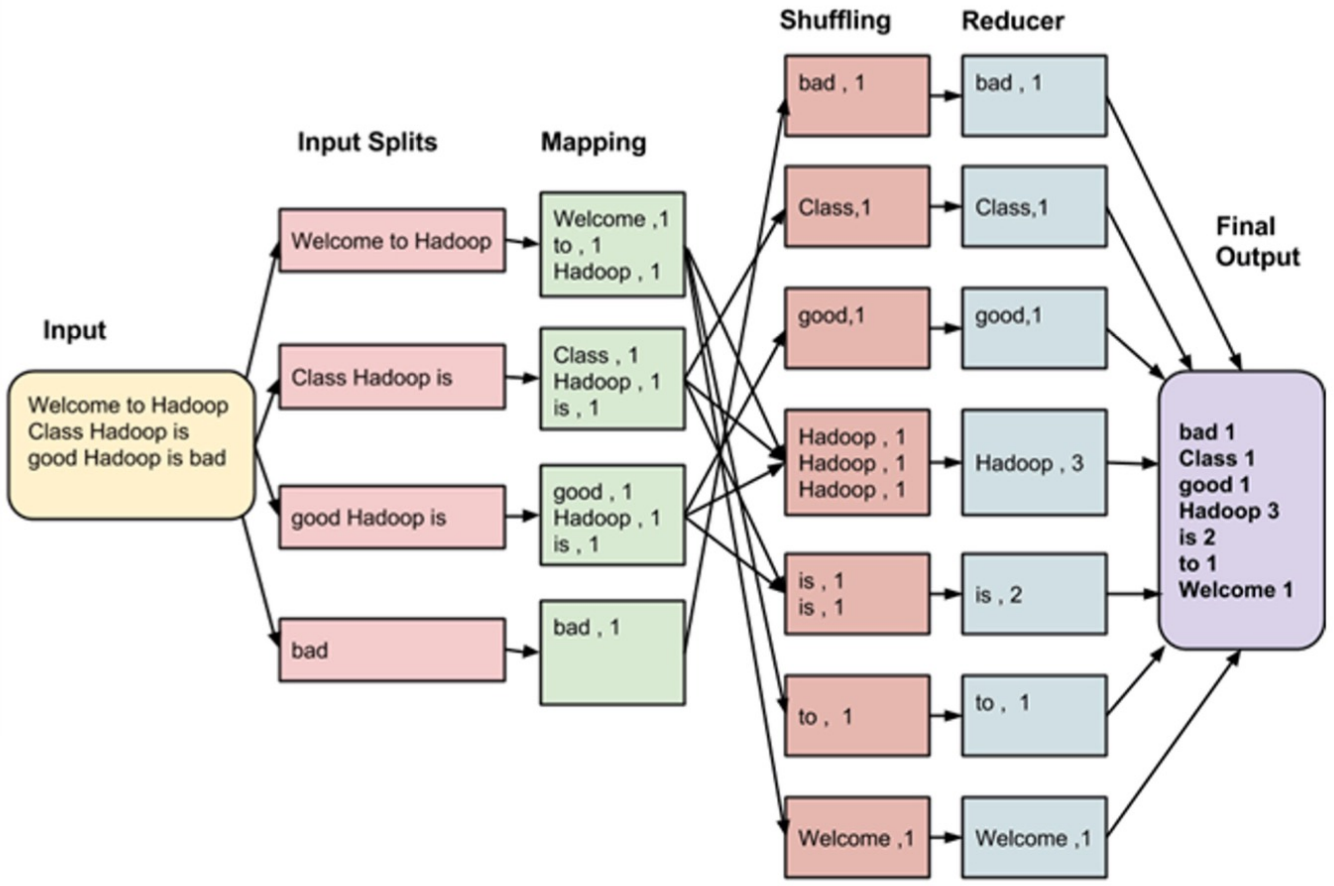
Principles of Distributed Computing

Scale out vs Scale up

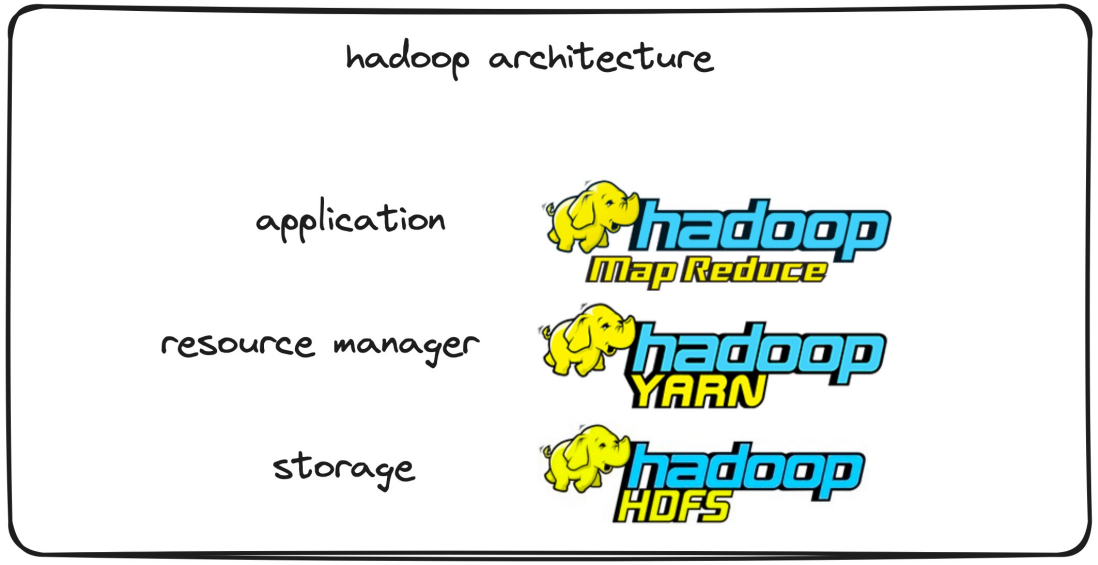


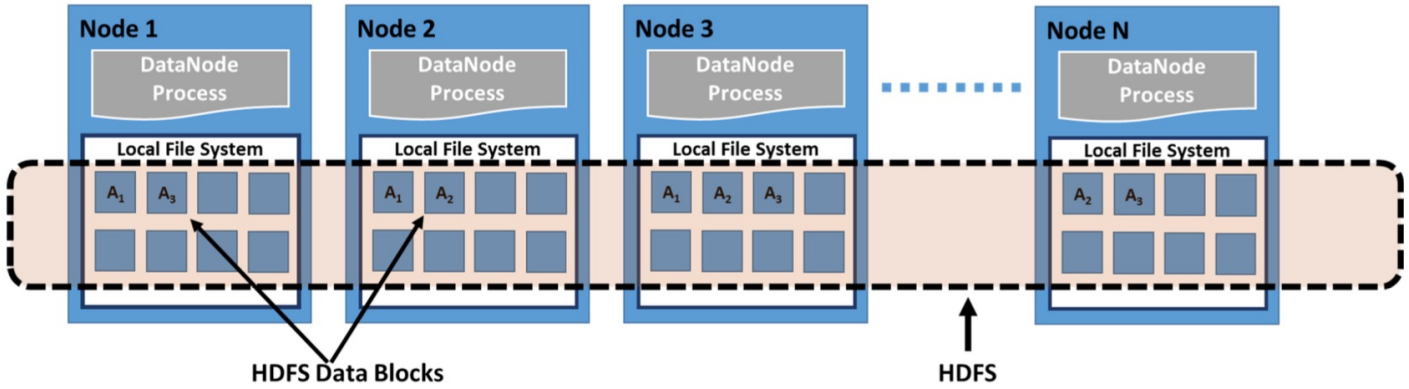
Principles of Distributed Computing

MapReduce



Hadoop / HDFS





Spark (1/6)

Why ?

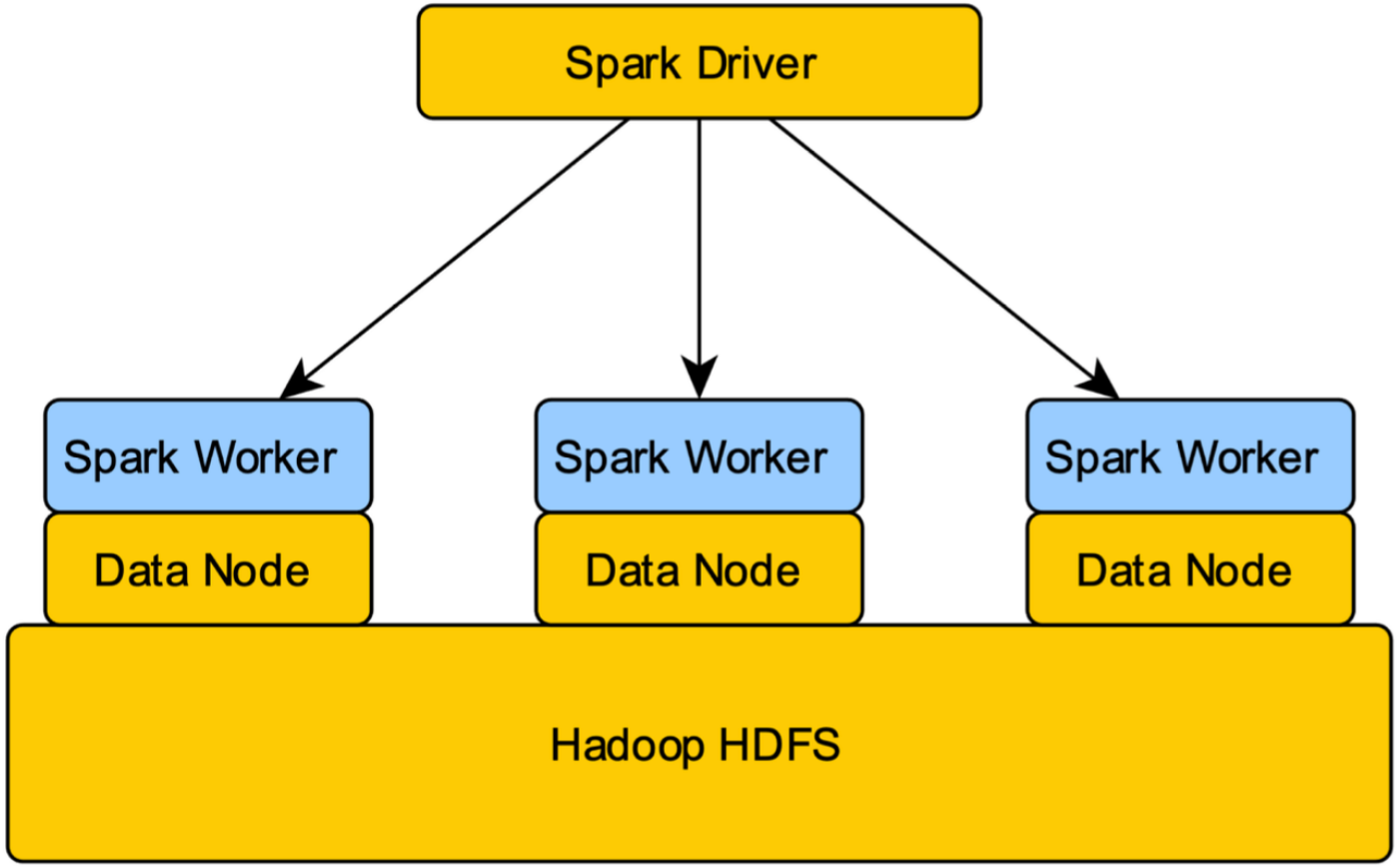
- Need something to deal with streaming (real-time) AND batch 👌
- Need for a powerful engine, <1s, in-memory analytics 👌
- Batch, Streaming, Graph, ML in 1 place ? 👌
- Compatible with existing open source 👌

Advantages over Hadoop/HDFS :

- Speed : 10-100x times faster thanks to in-memory
- More « userfriendly » (java)
- Better fault tolerance
- Better integration + optimization of queries
- Lazy evaluation

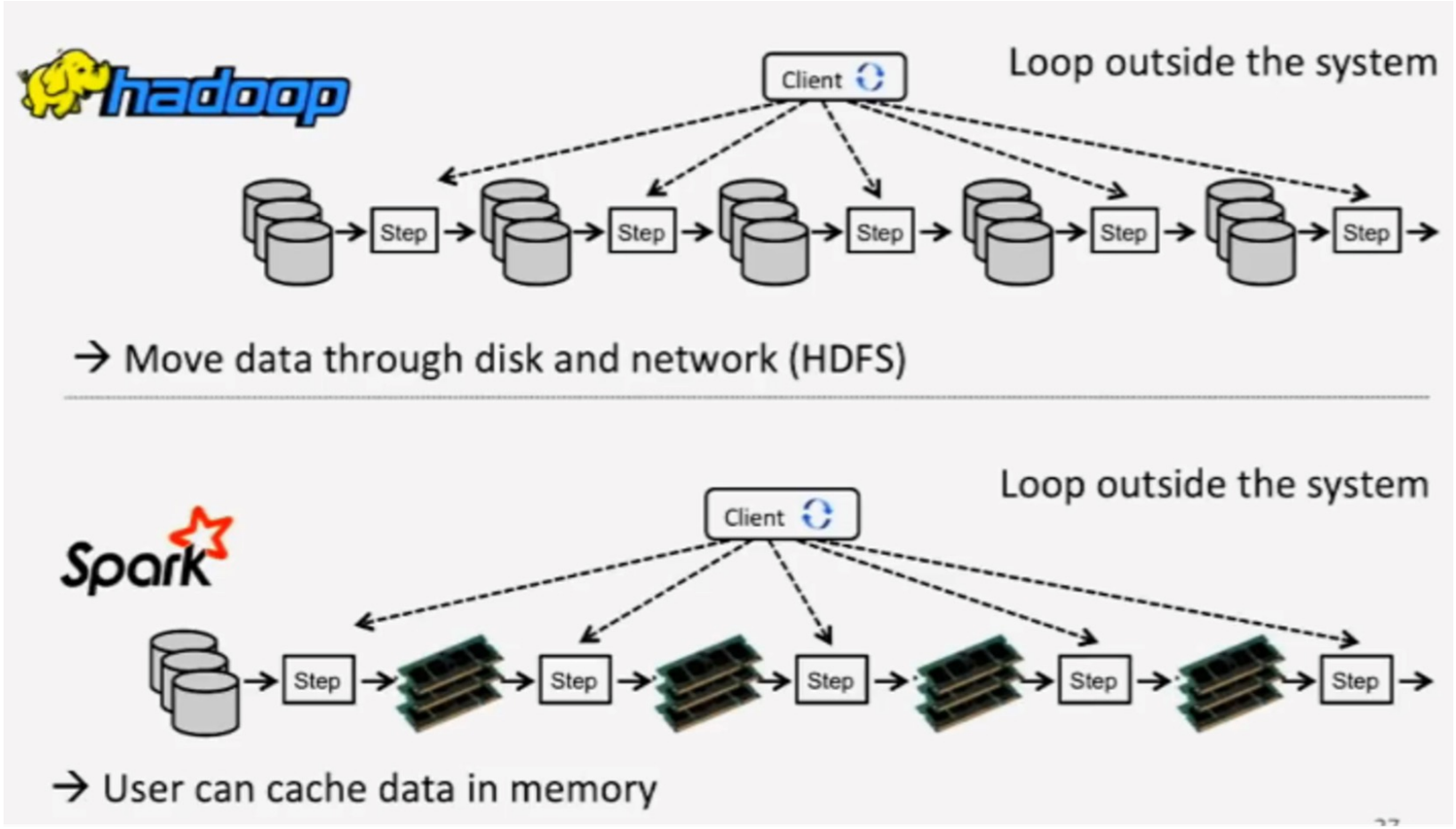
Spark (2/6)

Spark Architcture simplified



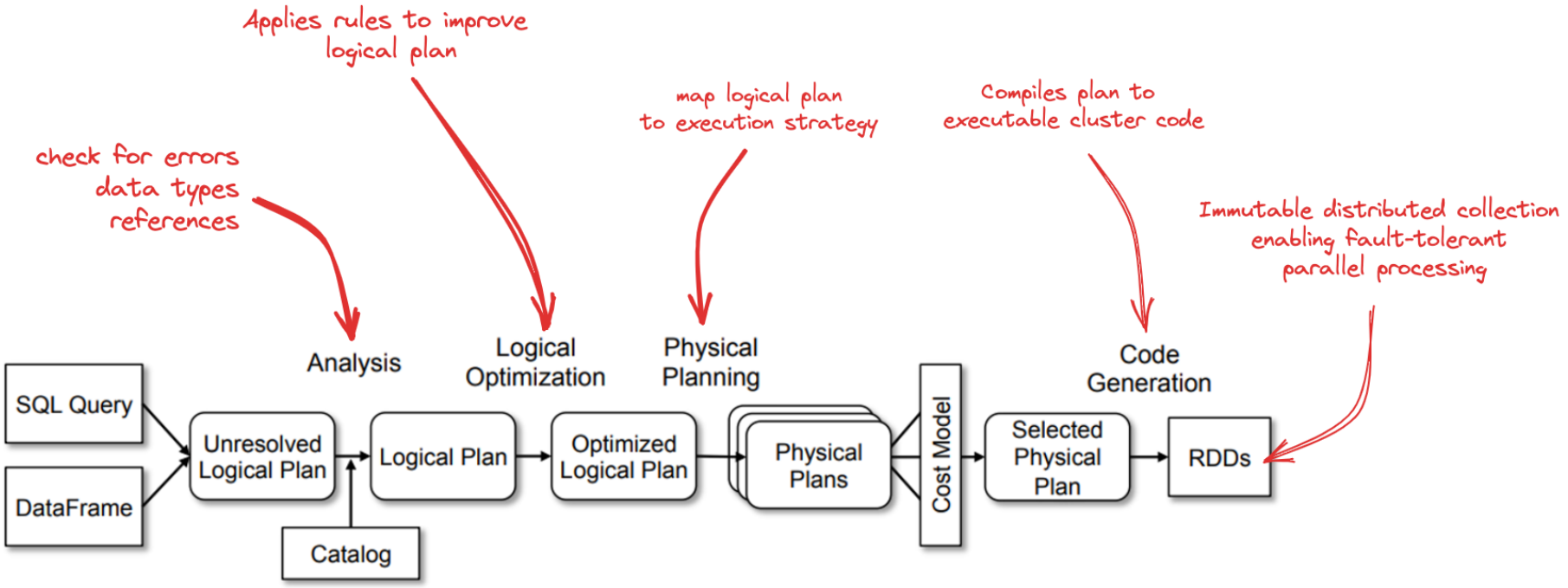
Spark (3/6)

Spark vs Hadoop



Spark (4/6)

Catalyst optimizer plan



Spark (5/6)

Lazy evaluation

```
columns = ["name","age"]
data = [("Alain",34),
        ("Ahmed",45),
        ("Ines",30),
        ("Fatima",28),
        ("Marie",40)]

df = spark.createDataFrame(data).toDF(*columns)
```

```

from pyspark.sql.functions import col, create_map, lit
from itertools import chain

mapping = {'Alain': 'm', 'Ahmed': 'm', 'Ines': 'f', 'Fatima': 'f', 'Marie': 'f'}


gender_mapping = create_map([lit(x) for x in chain(*mapping.items())])

df = df.withColumn('gender', gender_mapping.getItem(col("name")))  # First definition
df = df.withColumn('gender', lit('Unknown'))  # Second definition: overwrite the first one
```

▸ df: pyspark.sql.dataframe.DataFrame = [name: string, age: long ... 1 more field]

```
df.explain(True)
```

```
== Parsed Logical Plan ==
Project [name#310, age#311L, Unknown AS gender#318]
+- Project [name#310, age#311L, map(Alain, m, Ahmed, m, Ines, f, Fatima, f, Marie, f)[name#310] AS gender#314]
   +- Project [_1#306 AS name#310, _2#307L AS age#311L]
      +- LogicalRDD [_1#306, _2#307L], false

== Analyzed Logical Plan ==
name: string, age: bigint, gender: string
Project [name#310, age#311L, Unknown AS gender#318]
+- Project [name#310, age#311L, map(Alain, m, Ahmed, m, Ines, f, Fatima, f, Marie, f)[name#310] AS gender#314]
   +- Project [_1#306 AS name#310, _2#307L AS age#311L]
      +- LogicalRDD [_1#306, _2#307L], false

== Optimized Logical Plan ==
Project [_1#306 AS name#310, _2#307L AS age#311L, Unknown AS gender#318]
+- LogicalRDD [_1#306, _2#307L], false

== Physical Plan ==
*(1) Project [_1#306 AS name#310, _2#307L AS age#311L, Unknown AS gender#318]
+- *(1) Scan ExistingRDD[_1#306,_2#307L]
```

```
df.show()
```

▸ (2) Spark Jobs

```
+------+---+-------+
|  name|age| gender|
+------+---+-------+
| Alain| 34|Unknown|
| Ahmed| 45|Unknown|
|  Ines| 30|Unknown|
|Fatima| 28|Unknown|
| Marie| 40|Unknown|
+------+---+-------+
```

Spark (6/6)

Dataframe

We traditionally manipulate Spark's Dataframe

- RDD
- + organised as columns
- + has header
- + datatypes
- + (not)nullable

Some feature

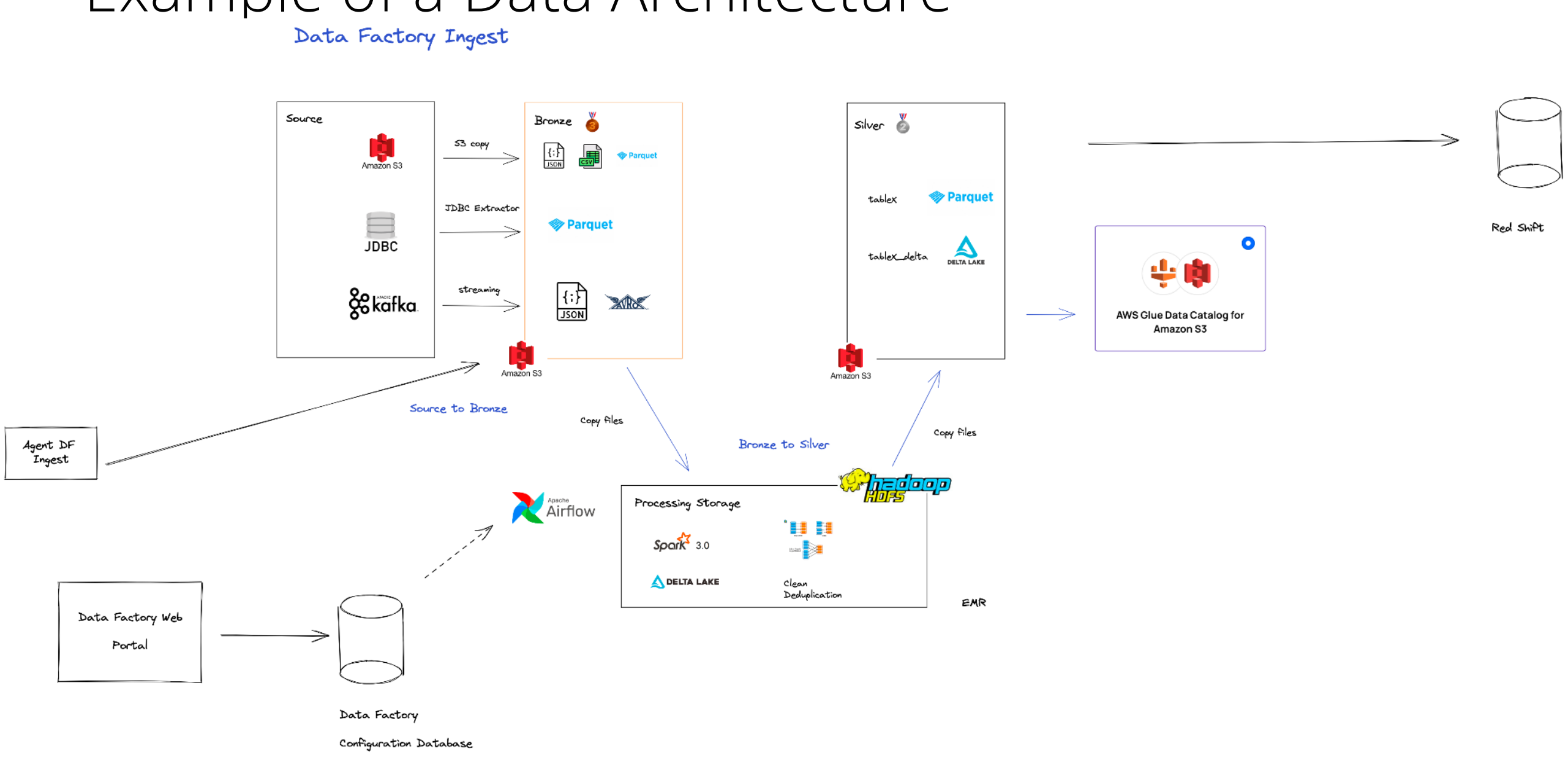
- Structured/unstructured (csv, txt, json, xml, avro, parquet)

```
1 #Print the schema and view the DataFrame
2 df.printSchema()
3 df.show()
```

```
root
 |-- Category: string (nullable = true)
 |-- ID: long (nullable = true)
 |-- Truth: boolean (nullable = true)
 |-- Value: double (nullable = true)

+--------+---+-----+------+
|Category| ID|Truth| Value|
+--------+---+-----+------+
|       A|  1| true|121.44|
|       B|  2|false|300.01|
|       C|  3| null| 10.99|
|       E|  4| true| 33.87|
+--------+---+-----+------+
```

Example of a Data Architecture



Data formats

- Columnar storage
- Efficient compression
- Optimized Query Performance
  - Include statistics in files
  - Only reads pertinent columns
  - Benefits from partitionning
- Schema evolution
- Snappy, gzip compressiion
- Integration with big data tools

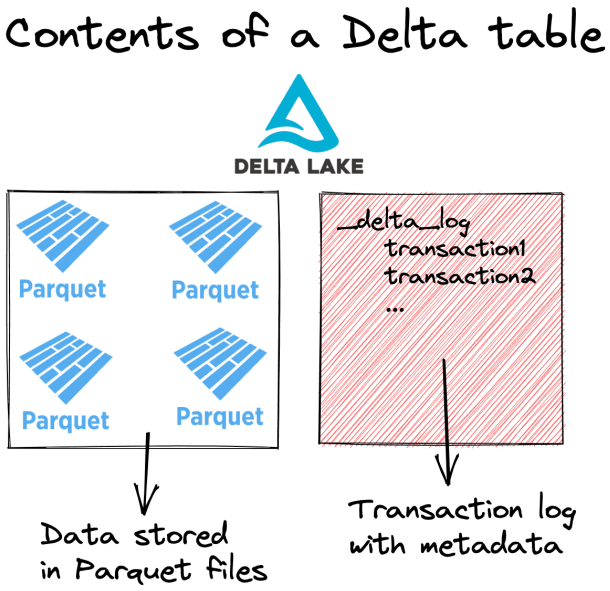
Date	Magasin	Produit	Montant
2020-01-15	Paris	Smartphone	500
2020-01-20	Paris	Tablette	300
2020-02-10	Lyon	Smartphone	450
2020-02-18	Lyon	Tablette	350

what was the total
sales in Paris in january 2020 ?

```
/sales_data/
    /magasin=Paris/
        /date=2020-01/
        /date=2020-02/
    /magasin=Lyon/
        /date=2020-01/
        /date=2020-02/
```

Jeu de données	Taille sur Amazon S3	Durée d'exécution de la requête	Données scannées	Coût
Données stockées dans des fichiers CSV	1 To	236 secondes	1,15 To	5,75 $
Données stockées au format Apache Parquet	130 Go	6,78 secondes	2,51 Go	0,01 $
Économies	-87 % en utilisant Parquet	34 x plus rapide	-99 % de données scannées	99,7 % d'économies

Data formats



Parquet + historisation -> ACID transactions
Transaction = Sequence of operations treated like only one unit of work

- Exp:

```
BEGIN TRANSACTION;
INSERT INTO Commandes (ID_Client, DateCommande, MontantTotal) VALUES (123, '2023-11-09', 150.00);
UPDATE Clients SET Solde = Solde - 150.00 WHERE ID_Client = 123;
COMMIT;
```

ACID

- **Atomicity**:
  Transactions are all-or-nothing, ensuring complete success or no effect at all.
- **Coherence:**
  Transactions transition the system from one valid state to another, maintaining business rules.
- **Isolation:**
  Transactions operate independently, unaffected by other concurrent transactions.
- **Durability:**
  Once a transaction is committed, its effects are permanent, surviving system failures.

Time travel (data versioning)

Upserts and deletes

Databricks

Why ?

- Facilitates a unified analytics platform
- Easy to set up
- Easy to industrialize
- Integrated workspace for Python, Scala, SQL
- Delta Lake (storage layer) have ACID properties
- Built on top of spark
- Implemented in each cloud platform's marketplace

Databricks

Why ?

Basics

- Load Dataframe
- Perform some transformations
- Save as parquet

Glossary

Date	Magasin	Produit	Montant
2020-01-15	Paris	Smartphone	500
2020-01-20	Paris	Tablette	300
2020-02-10	Lyon	Smartphone	450
2020-02-18	Lyon	Tablette	350

- **BI** : Business intelligence, dashboarding to be able to make a decision ; PowerBI, Tableau
- **Data** : From a single row to a batch of multiple nodes
- **Raw** : Data that comes directly from the source, no modification occured
- **Calculated** : Data is enriched through some transformations, inner joins etc…
- **Metadata** : Data around data : name of files, size, last modified date
- **Datalake** : Technology to massively store raw data
- **Data Warehouse :** Also stores data but can query files etc…
- **Batch processing :** you process a bulk of data, exp : monthly billing
- **Real-time processing :** you process data as soon as you receive it
- **Distributed computing** : you can’t load Tb of data, so you have to distribute over several machines
- **Node/Worker** : A machine that computes a simple task
- **MapReduce** : First framework to compute distributely

Things to remember

- ➔ The 5V's (Volume, variety, velocity, veracity, value)
- ➔ What is a metadata
- ➔ The difference between structured, semi-structured and unstructured data and their principal formats
    - ➔ Exp of question ; is mySQL/Postgre structured or unstructured ? Json ? Csv ? An image ?
- ➔ Raw vs Calculated Data
- ➔ Storage : What is a datalake vs datawarehouse
- ➔ Processing : Batch vs Real-time
    - ➔ Why do we need distributed computing
    - ➔ Scale-in/scale out principle
    - ➔ Parallelization principle
    - ➔ Make computation faster
        - ➔ Partitionment (know what it is)
- ➔ Csv vs parquet
- ➔ Spark main features + basic functions (what we saw in TP)