














2023/2024

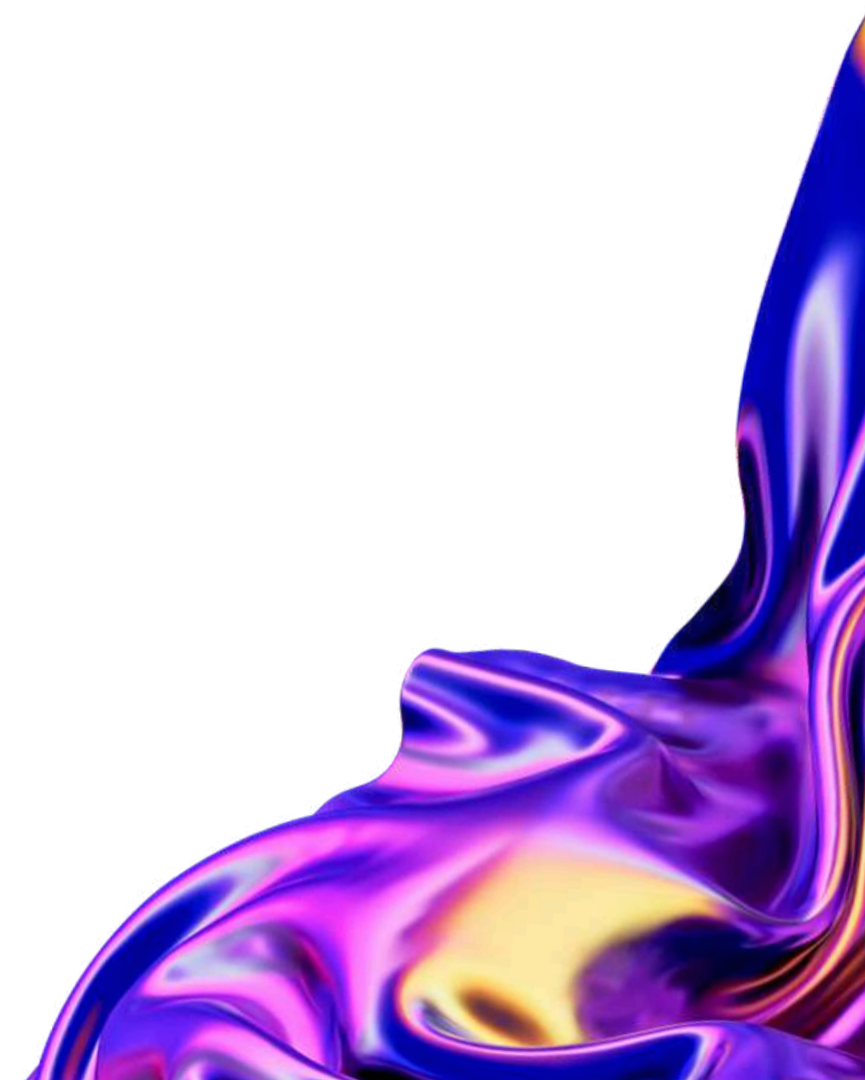
Deep Dive into Data Pipelines

Cervini Stella
Ferati Delor
Sabino Giuseppe

Technological Infrastructure for Data Science

Table of contents

-  Introduction: Big Data and Value Chain
-  Data Pipelines
-  ETL
-  ELT
-  Data Pipelines vs ETL (ELT).
-  Phases of a Traditional Data Pipeline
-  Orchestration (and Choreography)
-  Modern Data Pipelines
-  Medallion Architecture
-  Challenges in Data Pipeline Management
-  Opportunities in Data Pipeline Management





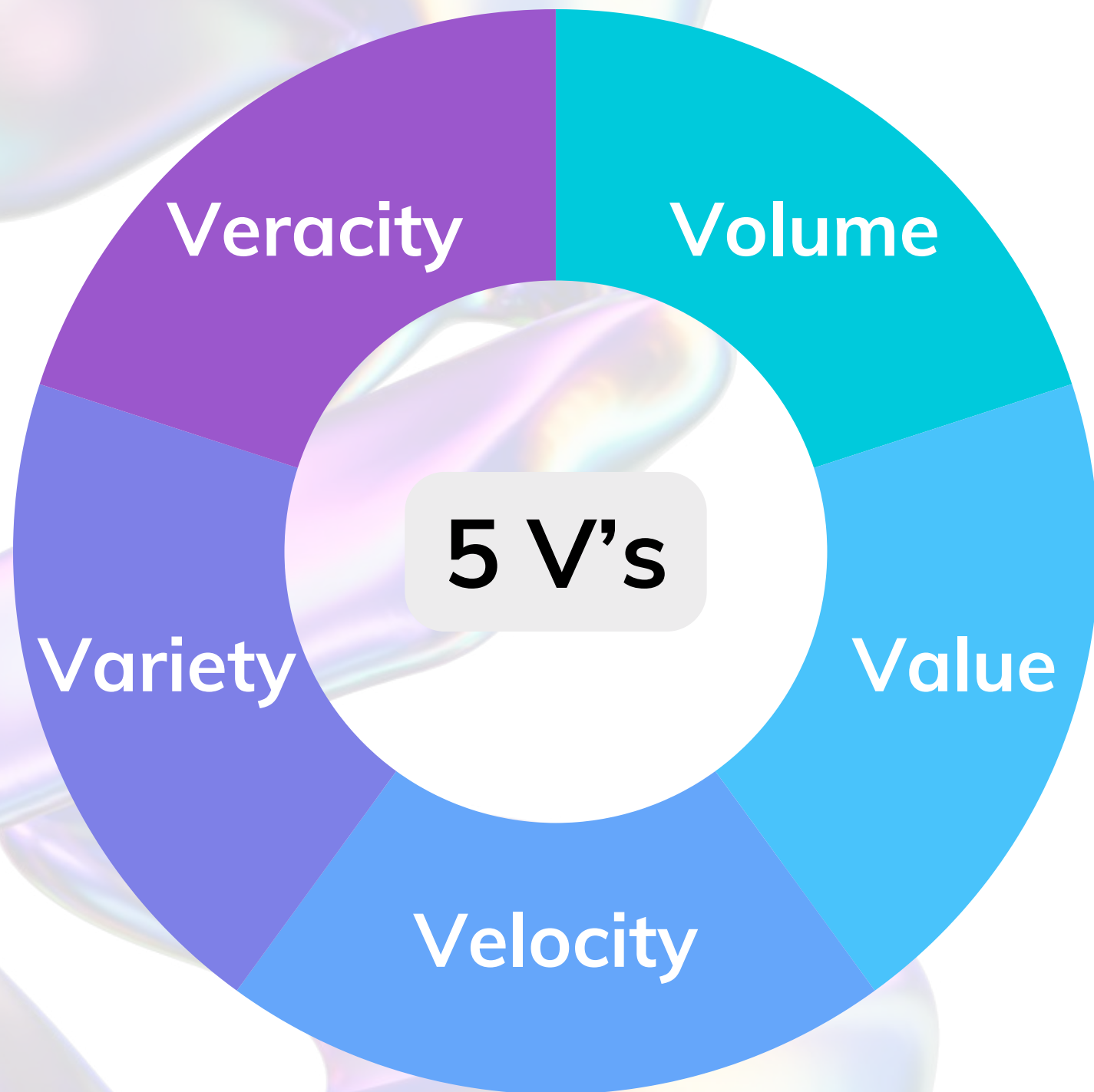
Introduction: Big Data

Big data is a combination of structured, semi-structured and unstructured data collected by organizations that can be mined for information and used in advanced analytics applications.

Systems that process and store big data have become a common component of data management architectures in organizations, combined with tools that support big data analytics uses.



Introduction: Big Data



Big data is often characterized by the **three V's**:

- **volume (amount of data)**: dealing with large scales of data within data processing (e.g. Global Supply Chains, ...).
- **variety (range of data types/sources)**: dealing with data using differing syntactic formats (e.g. CSV, XML, DBMS), schemas, and meanings (e.g. Enterprise Data Integration).
- **velocity (speed of data)**: dealing with streams of high frequency of incoming real-time data (e.g. Sensors, Pervasive Environments, ...)

Other relevant characteristics of Big Data are:

- **value**: the richness that data brings
- **veracity**: different quality and accuracy in data

💡 Big Data is a resource!

The ability to effectively manage information and extract knowledge is now seen as a **key competitive advantage**, and many organizations are building their core business on their ability to collect and analyze information to extract business knowledge and insight.

The Vs of big data challenge the fundamentals of existing technical approaches and require **new forms of data processing** to enable enhanced decision-making, insight discovery, and process optimization.

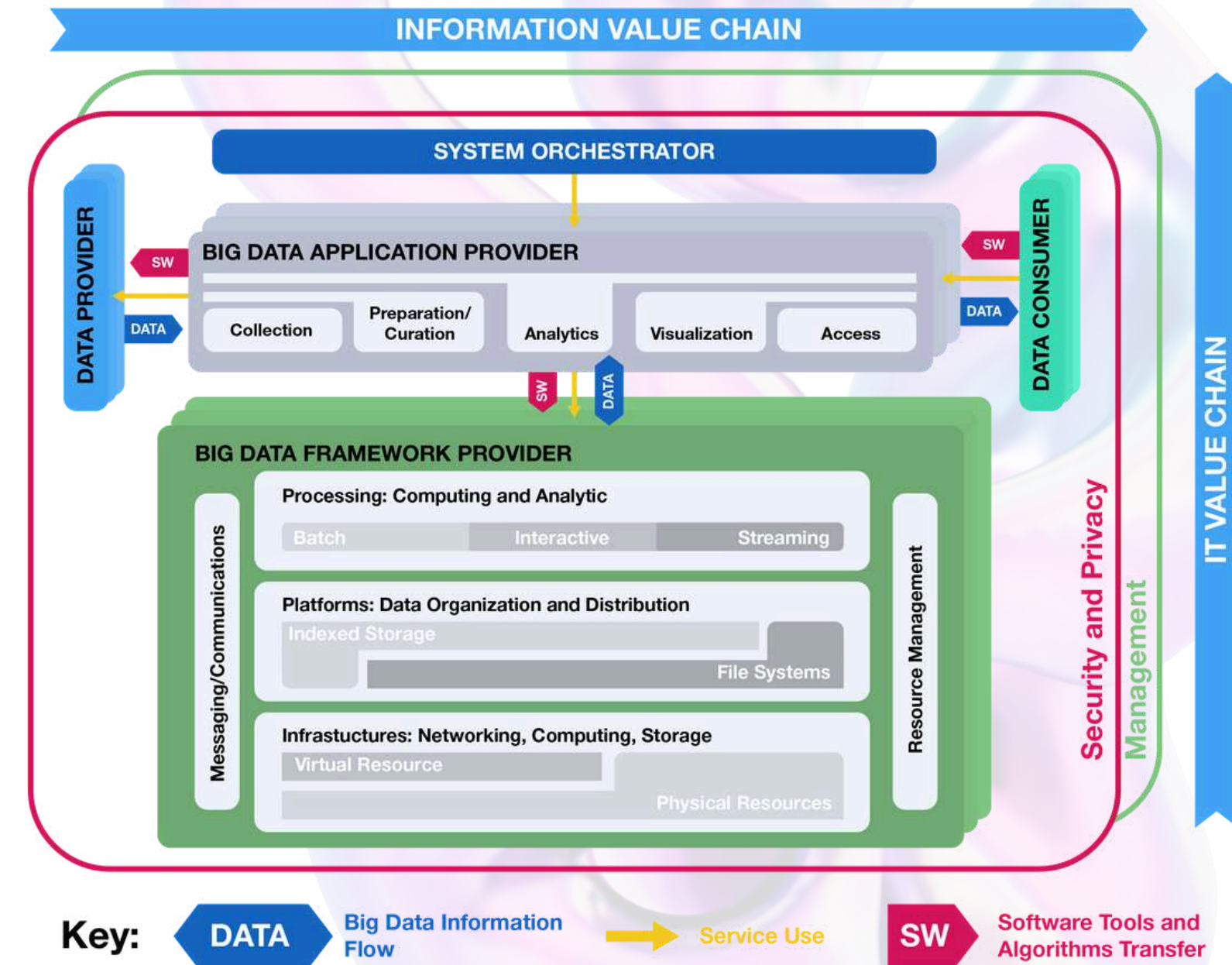


➡ The Value Chain

Value Chains have been used as a decision support tool to model the chain of activities that an organization performs in order to deliver a valuable product or service to the market (Porter 1985). The value chain categorizes the generic value-adding activities of an organization, allowing them to be understood and optimized.

A value chain is made up of a **series of subsystems** each with inputs, transformation processes, and outputs.

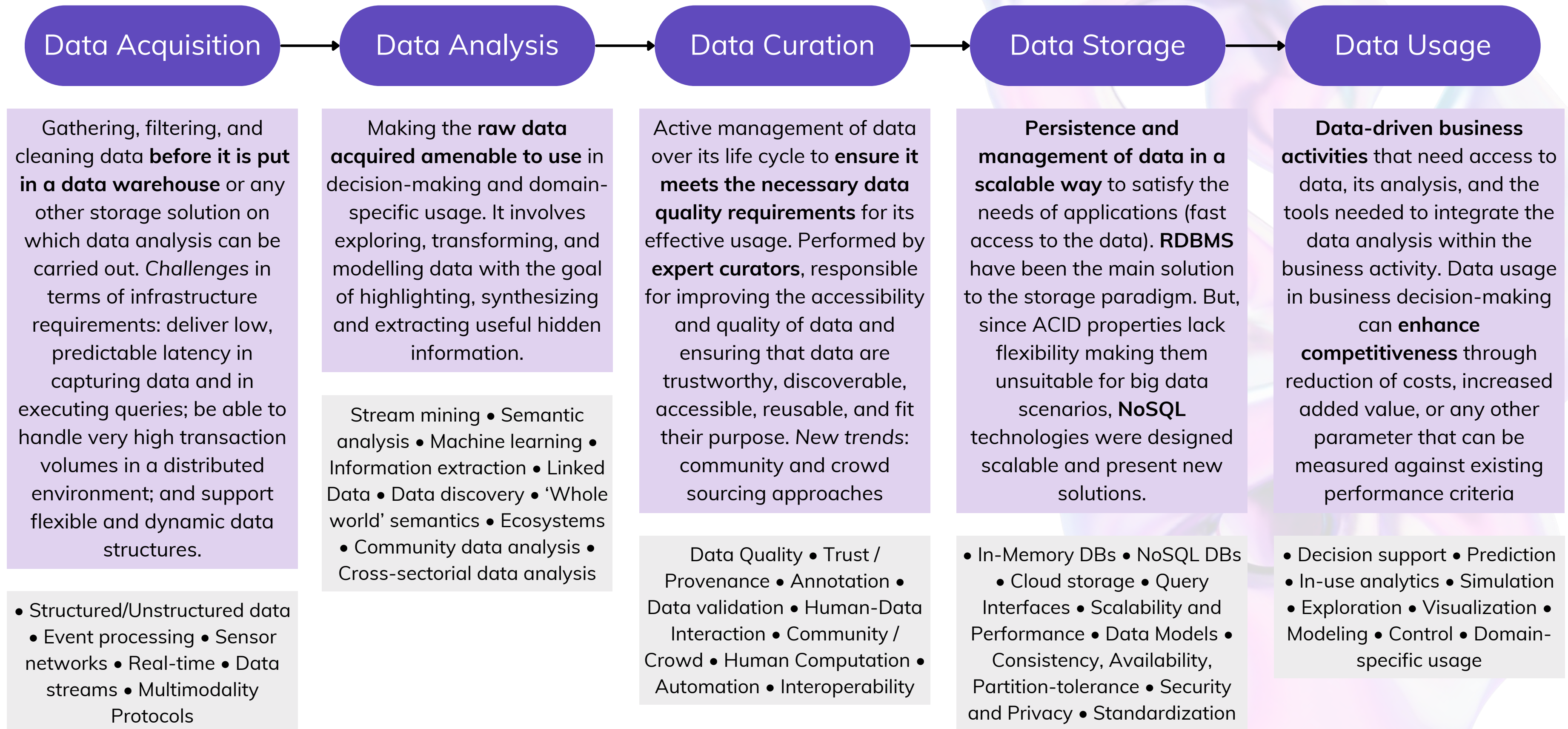
As an analytical tool, the value chain can be applied to information flows to **understand the value creation of data technology.**



The NIST Architecture

Source: <https://www.bigdataframework.org/big-data-architecture/>

⇒ The Value Chain



Data Pipelines

Definition

*Theoretically, a **data pipeline** refers to a **directed acyclic graph (DAG)** composed of a sequence of nodes. These nodes process (e.g., merge, filter) data while the output of one node will be the input of the next node. At least one **source node** produces the data at the beginning of the DAG, and at least one **sink node** finally receives the processed data.*

*Practically, **data pipelines** typically constitute a piece of software that automates the manipulation of data and moves them from diverse source systems to defined **destinations**. Thus, data pipelines represent **digitized data processing workflows** based on a set of programmed scripts or simple software tools.*

Data Pipelines

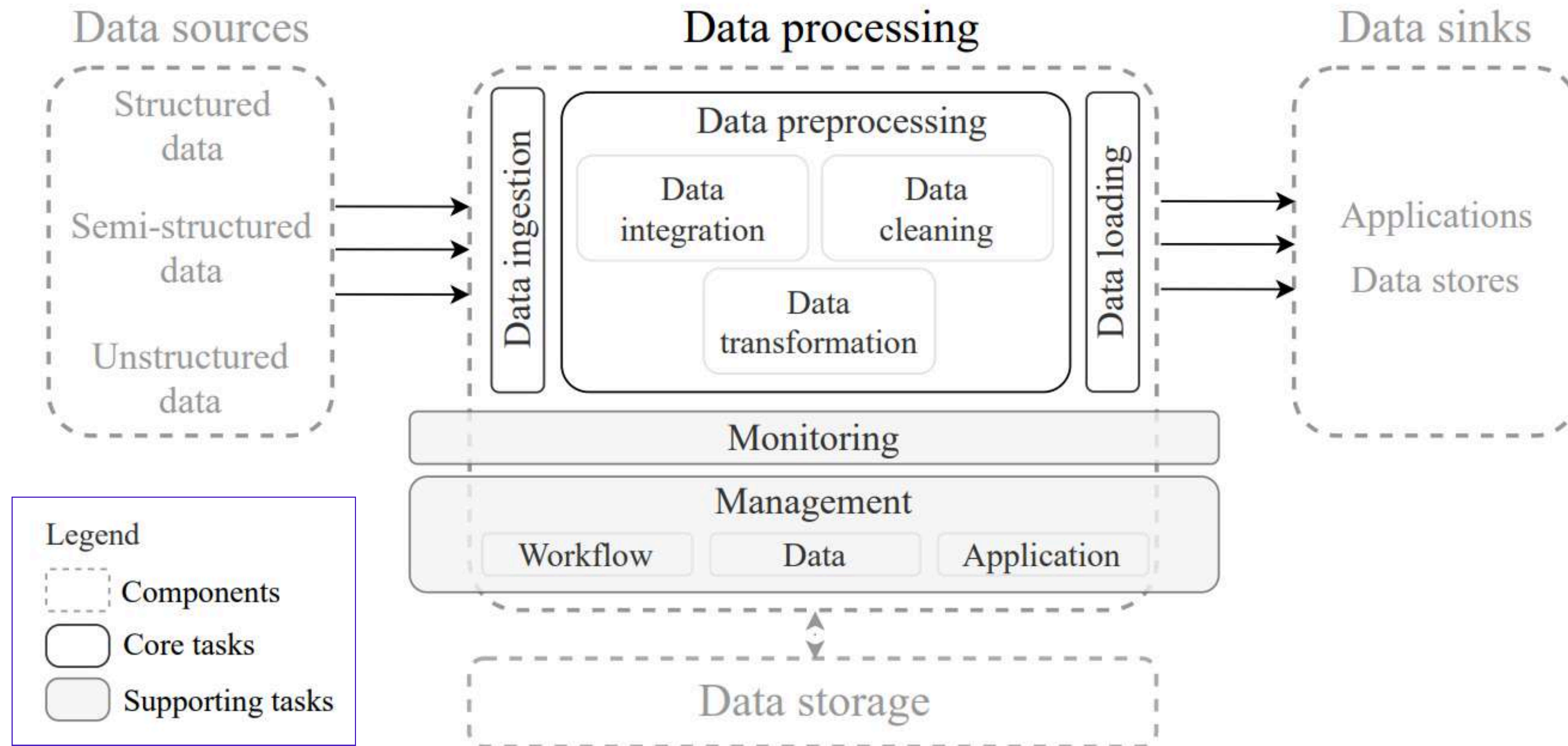
Definition

Given the increasing importance and complexity of data processing, data pipelines are nowadays even **treated as complete software systems** with their own ecosystem comprising several technologies and software tools

The **primary purpose** of this set of complex and interrelated software bundles is to enable efficient data processing, transfer, and storage, control all data operations, and orchestrate the entire data flow from source to destination.

Data Pipelines

Generic architecture of a Data Pipeline



Data Pipelines

Data Pipeline Components

while the internal structure of a data pipeline can vary significantly, its main components are typically the same

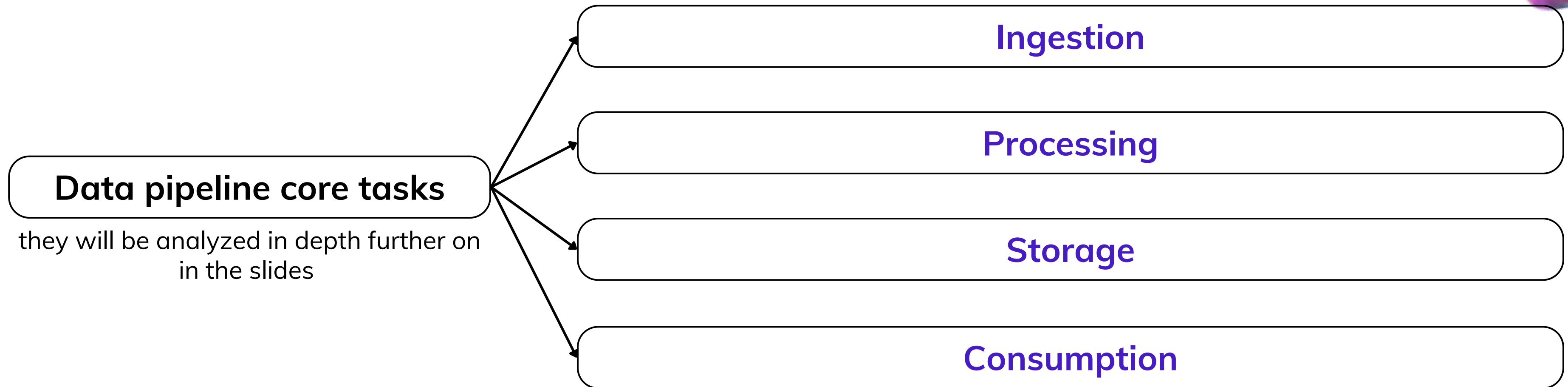
Data sources: the starting point of every data pipeline is its data sources. They can be of **different types**, e.g., databases, sensors, or text files. The data produced by data sources are typically either in a **structured** (e.g., relational database), **semi-structured** (e.g., e-mails, web pages), or **unstructured** (e.g., videos, images) form.

Data processing: within this component, the core tasks for manipulating and handling data (i.e., **data ingestion, data preprocessing, and data loading**) are provided.

Data storage: the internal storage of the pipeline. It stores raw, ingested, and processed data, depending on the configuration of the pipeline. A common distinction is made between **temporary** or **long-term storage** of the data.

Data sinks: it describes the **destinations** where the data are finally provided. These can be other pipelines, applications of any type, or external data storage systems (e.g., data warehouses, databases).

Data Pipelines



Data Pipelines

Data pipeline supporting tasks

the data processing tasks of a pipeline are usually supported by several monitoring and management task

Monitoring: the set of tasks refers to **overseeing the data** quality, all data operations, and controlling the performance of the pipeline. By using logging mechanisms and creating alerts, monitoring prevents data quality issues and eases debugging and recovering from failures.

Management: related to the data, the workflow, and the overall application. **Workflow management** describes all activities regarding the orchestration and dependencies between the different data processing tasks (i.e., the entire data flow). **Data management** includes tasks such as creating and maintaining metadata, data catalogs, and data versioning. **Application management** encompasses the configuration and maintenance (e.g., upgrades, version control) of all software tools a pipeline is built upon.

Data Pipelines

Types of Data Pipelines

Pipelines can be classified based on several characteristics. The **three most common types** of classification are described in the following.

Data ingestion strategy

Data ingestion
frequency

batch mode

streaming mode

Data processing method*

Application and order
of data processing
tasks

ETL

Raw Data Load

Streaming ETL

**Data Stream
Processing**

ELT

**Change,
Data,
Capture
(CDC)**

Use case

different purposes of
data pipelines

**data collection
pipelines**

**data preprocessing
pipelines**

*Also known as **design patterns**

Data Pipelines

Design Patterns

- **Raw Data Load**: it involves the **movement and loading of raw data from one location to another** (e.g. between databases or from an on-premise data center to the cloud). Since this design pattern involves moving raw data, it involves only the extraction and loading process. Raw data loading can be slow and time-consuming, especially as data volume increases. This design pattern works best for one-time operations and is not ideal for recurring situations.
- **Extract, Transform, Load (ETL)**: the most popular design pattern for loading data into data warehouses, lakes, and operational data stores. It involves the **extraction, transformation and loading of data from one location to another**. Most ETL processes use batch processing which introduces latency to operations. ETL works for most data integration processes where data sources undergo integration at different times.

Data Pipelines

Design Patterns

- **Streaming ETL**: identical to the standard ETL pattern, but where data stores act as the origin for the traditional ETL pattern, **data streams become the origin in this case**. The process involves parsing and filtering, followed by transformation and loading into a data lake. Streaming ETL utilizes a streaming processing tool like Apache Kafka or StreamSets Data Collector Engine to perform the complex ETL process.
- **Extract, Load, Transform (ELT)**: this architecture **collects data and then loads it into a target destination before transforming**. By placing the transformation process as the last process in the workflow, there is a reduced latency than the ETL design. A major concern for this design is its effect on data quality and violation of data privacy rules. When data gets loaded without certain transformation processes that hide sensitive data, it may expose sensitive information.

Data Pipelines

Design Patterns

- **Data Stream Processing**: ideal for feeding real-time data to high volume and high-performance applications like the Internet of Things(IoT) and financial applications. Data arrives from a string of devices in a **continuous stream**, gets parsed and filtered into records of interest and processed before being sent to various destinations like dashboards feeding real-time applications.
- **Change, Data, Capture (CDC)**: it introduces freshness to data processed using the ETL batch processing pattern. By **detecting changes occurring during the ETL process** and sending to message queues using CDC, they're made available for downstream processing, which processes them in mini-batches, making data available immediately.

These are templates **used to create data pipeline foundations**. The data pipeline design choice depends on multiple factors like how data is consumed/received, business use cases, and data volume.

Data Pipelines

Data Pipelines Use Cases (Purposes)

Data pipelines are typically used for **data movement and preparation** for, or as part of, other applications (e.g., visualization, analysis tools, ML and deep learning applications, or data mining).

Data collection pipelines: usage of data pipelines to gather data from a variety of sources and to move them to a central place for further usage. They are applied in nearly every business domain (e.g., manufacturing, medicine, finance).

Data preprocessing pipelines: used for preparing the data so that they are in a suitable form when fed to algorithmic models (AI-based systems, ML pipelines, or in data science projects).

Data Pipelines

Architectures

A **data pipeline architecture** refers to the **design of tools and processes** that help transport data between locations for easy access and application to various use cases.

Data pipeline architecture aims to **make the data pipeline process seamless and efficient** by designing an architecture that helps improve the functionality and flow of data between various sources and along the pipeline.

There are **different types** of Data Pipelines:

- Batch Architecture
- Streaming Architecture
- Lambda Architecture (Streaming/Batch)

Data Pipelines

Architectures

A **batch pipeline** is designed typically for **high volume data workloads** where **data is batched together** in a specific time frame. This time frame can be hourly, daily, or monthly depending on the use case.

The data formats consumed might consist of files like CSV, JSON, Parquet, Avro and files might be stored in different cloud object stores or data stores of some kind.

Example: a company generates sales transactions that are recorded in an operational database. The business analysts want this data for analysis so you set up a pipeline that extracts and processes all sales transactions from the database at the end of each day for storage in your data warehouse where analysts can work with the data through dashboards.

Data Pipelines

Architectures

A **streaming pipeline** is designed for **data that gets generated in real time or near real time**. This data is crucial in making instantaneous decisions and can be used for different IoT devices, fraud detection, and log analysis.

Example: credit card companies rely on a streaming data pipeline architecture to continuously stream transaction data to detect irregularities and stop fraud before it happens. Similarly, applications present recommendations to a customer based on their real-time choices leading to better customer experience (a la Netflix, Amazon, or YouTube).

Data Pipelines

Architectures

The **Lambda Architecture** is a deployment model for data processing that organizations use to **combine a traditional batch pipeline with a fast real-time stream pipeline** for data access. It is a common architecture model in IT and development organizations' toolkits as businesses strive to become more data-driven and event-driven in the face of massive volumes of rapidly generated data, often referred to as "big data."

Example: stock trading. Traders need real-time processing to make decisions about which securities to buy and sell throughout the day. This requires processing a massive amount of data very quickly, which stream processing excels at.

But analysts and stock traders also need highly accurate data that they can later audit, like financial reports. In this particular example of lambda architecture, stock traders can get instant processed data from the speed layer (data streaming layer) and get more complete data from the batch layer.

What is ETL?

ETL stands for **Extract**, **Transform** and **Load**. It's a process originally used in data integration and traditional data warehousing techniques. The classical approach involves on-premise ETL specific tools, whose purpose is to integrate data from various sources (commonly large enterprise systems) into the target system.

In the context of *Big Data* pipelines, ETL refers to the **set of activities** involved in collecting, processing and loading large volumes of data into new cloud-based analytical solutions.

The ETL process is often extended or adapted to suit the unique characteristics of **distributed and parallel processing frameworks**. Technologies like Apache Flink, Apache Beam and in particular Apache Spark are used for large-scale data processing and ETL in Big Data environments.



Extract

This step involves “**pulling**” or “**pushing**” data from different source systems. In Big Data scenarios, these sources can include various storage systems, databases, log files, social media, IoT (sensors) and more.

In order to **enhance query efficiency**, the common approach is to *push* requests towards the source system.

The extraction phase can be performed in 2 *main* ways:

- **Full Extraction**: all the data are extracted from the source structure without filters, with the aim of completely overwriting the target each time. It is particularly useful in structures with not many data and fairly “static”
- **Delta Extraction**: refers to the process of extracting only the data that has changed since the last extraction, focusing on obtaining incremental changes or additions to the dataset. This can be achieved through various methods, such as using timestamps, CDC techniques, or maintaining a log of changes. It’s particularly useful to use this extraction with IoT data or large transaction tables to have an efficient and cost-effective flow.

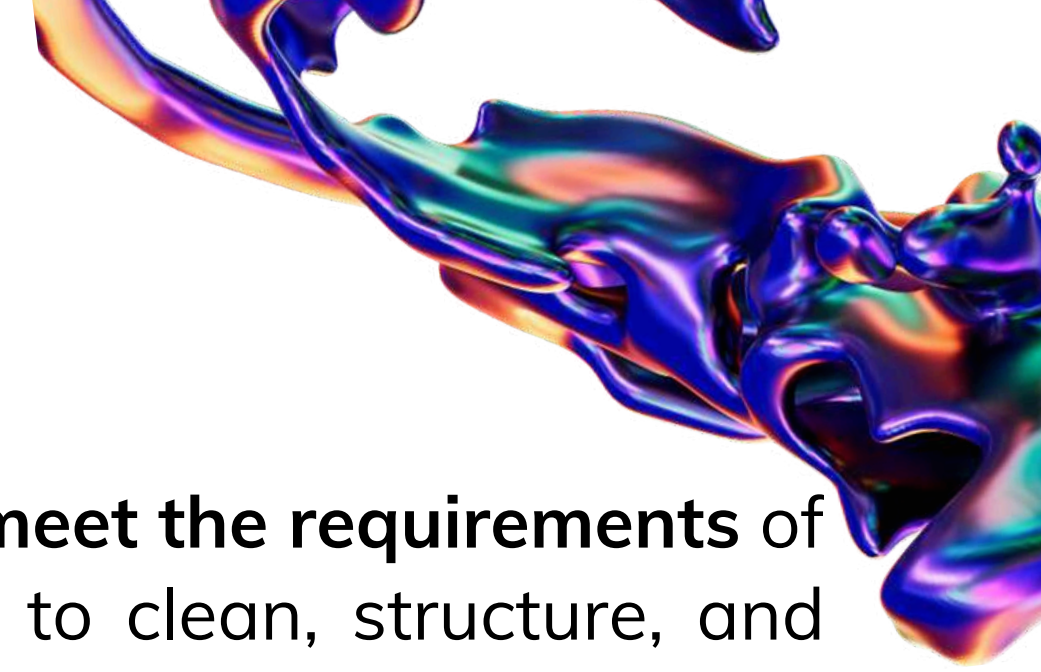


Transform

This is a critical step where the extracted data is **processed and modified to meet the requirements** of the target system. The primary objectives of the transformation phase are to clean, structure, and enhance the data so that it is suitable for analysis and reporting.

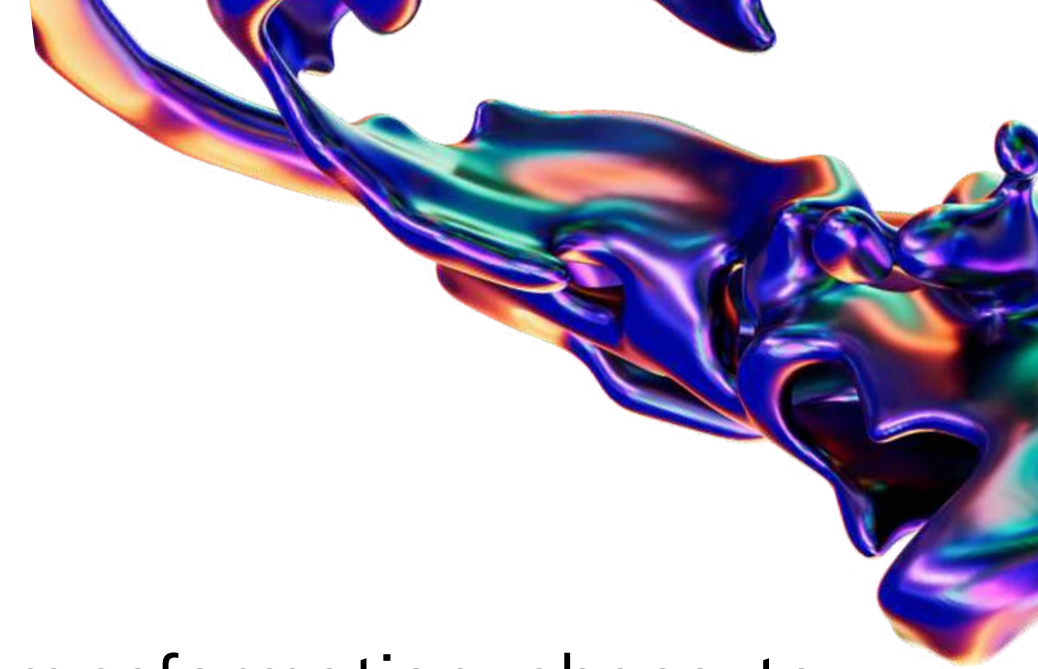
Here are the key aspects of the transformation phase:

- **Data Cleaning**: data from different source systems may contain inconsistencies, errors, or missing values. Data cleaning involves activities such as correcting errors, handling missing values, and ensuring data quality. This step is crucial for maintaining the accuracy and reliability of the data;
- **Data Integration**: this step of transformation includes the consolidation of data from different sources into a unified format. This ensures that data from various systems can be effectively analyzed together. An example could be the **reconciliation of Master Data**;
- **Data Enrichment**: involves enhancing the dataset with additional information that may not be present in the source data. This can include appending geospatial data, adding derived metrics, or integrating external data sources;



Transform

- **Applying Business Rule:** business rules are often applied during the transformation phase to ensure that the data conforms to specific business requirements. These rules may involve calculations, derivations, or other logic to generate new data elements or modify existing ones. In a classical ETL approach the Business Rules **are predefined**; although this isn't always the case in the modern ELT cloud - process
- **Aggregation and Summarization:** depending on the requirements of the target system, the transformation phase may involve aggregating or summarizing data. This is common in data warehousing scenarios where summarized data is often used for reporting purposes.



Loading

In this phase, data is **loaded into the target system**, which could be a data warehouse, data mart, or another storage system like a cloud - based solution. The loading phase **completes the cycle** of moving data from source systems to a destination where it can be queried and analyzed effectively.

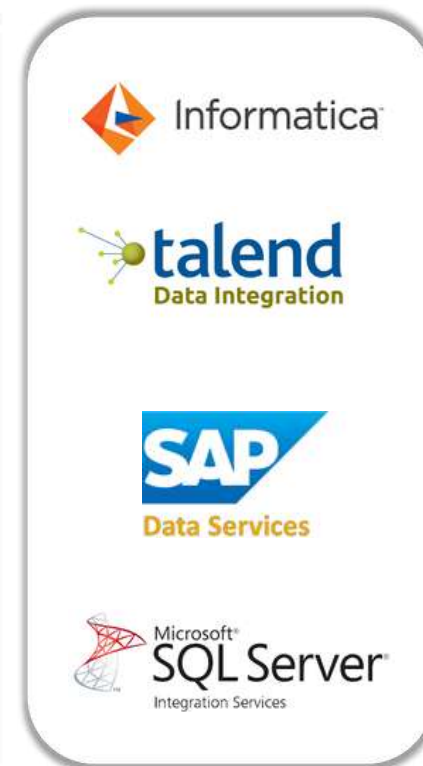
There are generally two types of *loading*, according to the extract step:

- **Full Load (Initial Load)**: In the initial load, all the data from the source system is loaded into the target system. This is often done when the ETL process is set up for the first time or when a major system update or data migration is required.
- **Incremental Load**: After the initial load, subsequent loads are usually incremental. In this approach, only the new or changed data since the last load is transferred from the source to the target system. This helps in reducing the amount of data that needs to be processed, making the ETL process more efficient and cost-effective.



Some ETL Tools in 2024

Figure 1: Magic Quadrant for Data Integration Tools



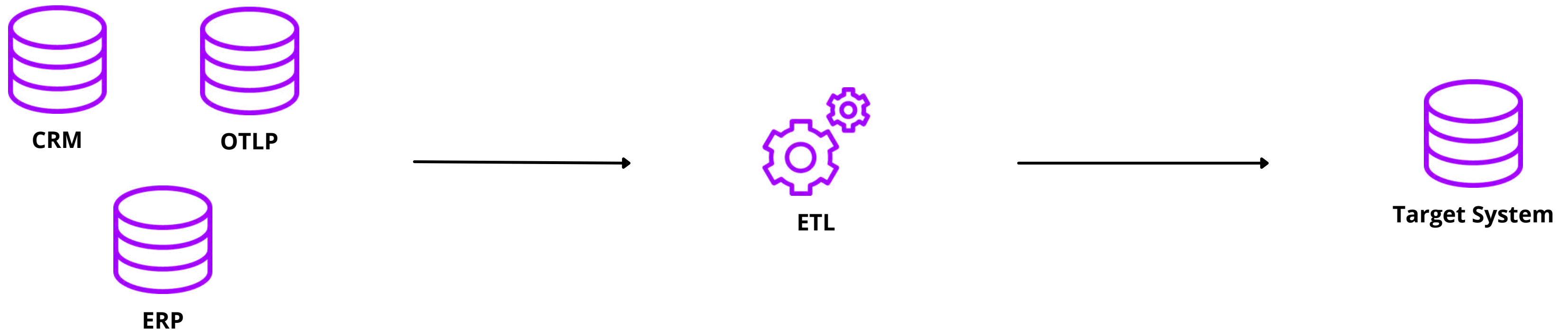
*These ETL tools, in particular, manage to endure even today despite the rise of cloud solutions and distributed processing frameworks such as Spark. The reason for this is because they are tools with many years of **history and optimization**, offering comprehensive **connector coverage for any type of data source**.

ETL

Traditional Job

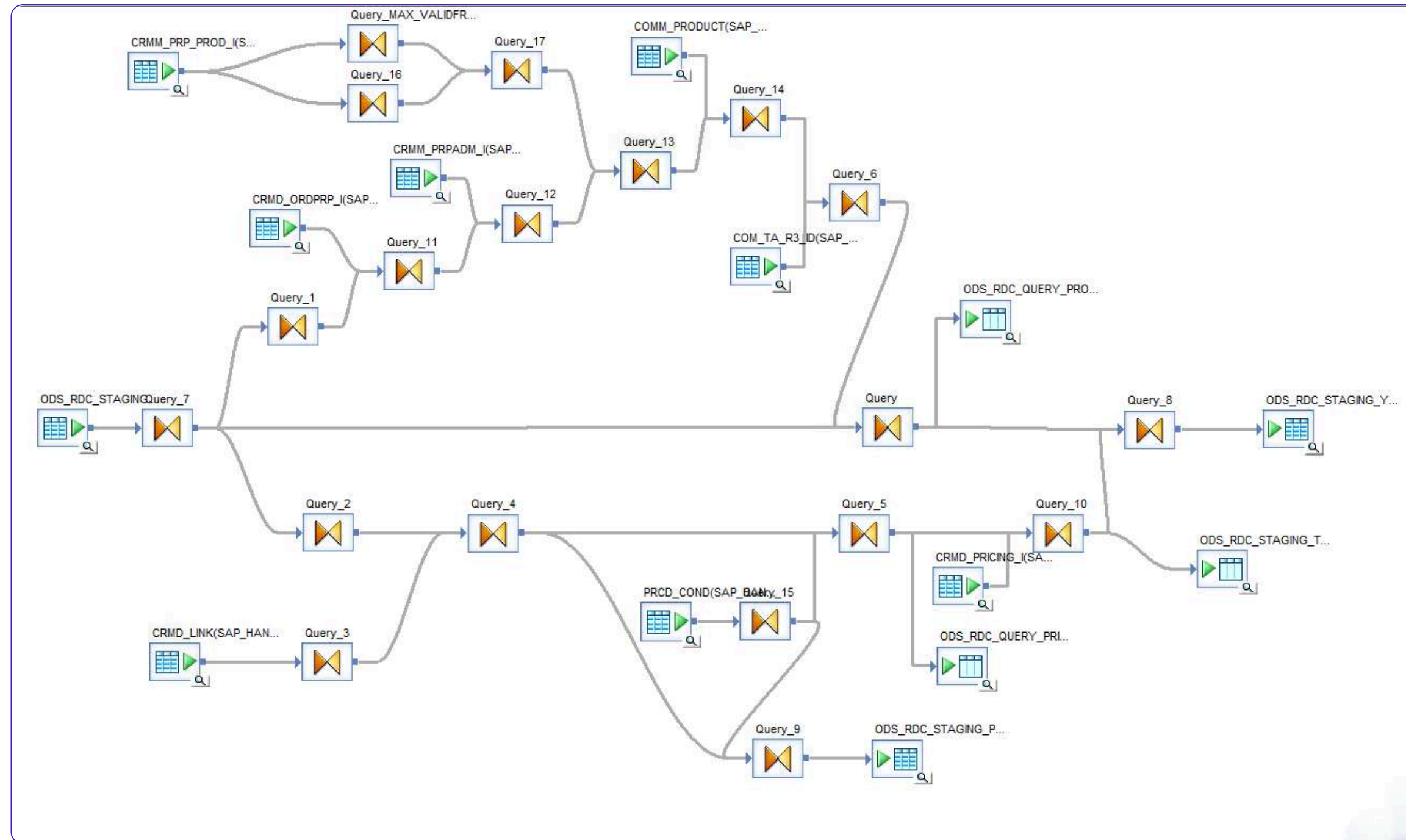
What primarily guides the development of an ETL procedure are the **business rules**, meaning what is effectively needed by end users.

As mentioned earlier, in traditional jobs, these rules are predefined. This results in very precise and rigid procedures regarding the transformation rules to be applied:



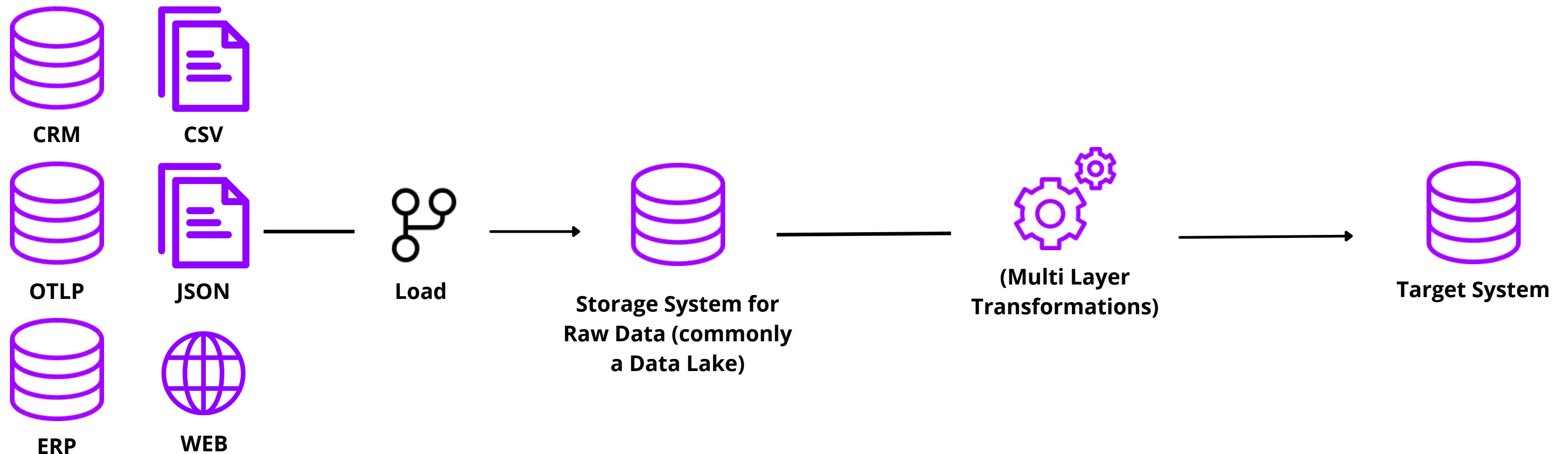
ETL

Traditional Job - SAP Data Services Example 1



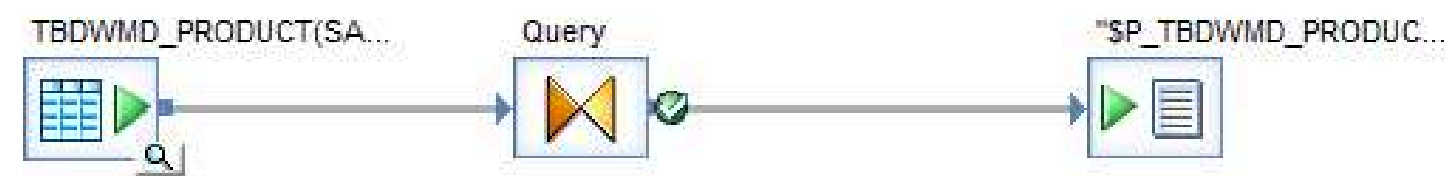
ELT

Is an alternative approach to data integration that differs from the traditional method. In ELT, the transformation of data occurs directly within the target data store, such as a data lakehouse, rather than in an intermediary staging area as in ETL. The *main difference* in this scenario is the absence of predefined rules; the goal is to **discover insights and patterns** during the **gradual transformation of the data**.



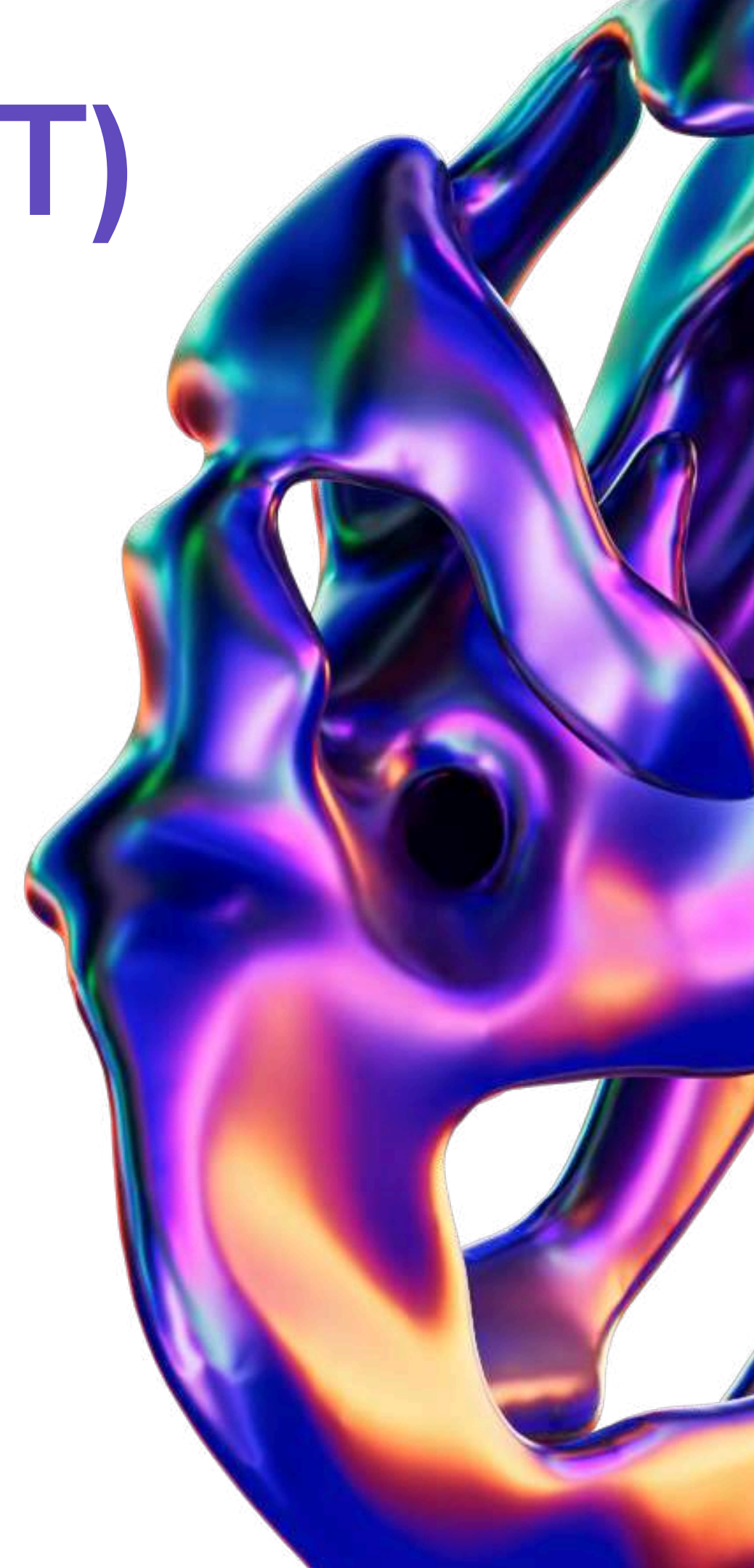
ELT

Modern Approach - SAP Data Services Example 2



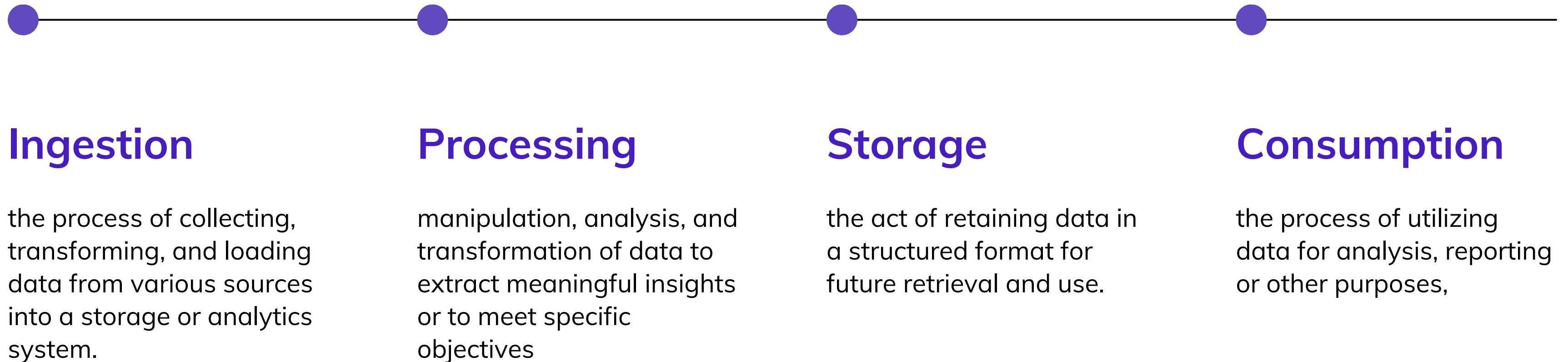
Data Pipelines vs ETL (ELT)

- A data pipeline is a broader concept that encompasses the end-to-end process of moving and processing data from source to destination. It includes not only ETL/ELT processes but also other stages and components involved in data movement (as example, **Orchestration**);
- A data pipeline may involve various operations like data ingestion, data storage, data analysis and Machine Learning. It's a more comprehensive term that covers the entire flow of data from source to destination;
- Data pipelines can be used for various purposes, including **real-time** data streaming, **batch** processing, and general data integration across different systems.



Phases of a *Traditional* Data Pipeline

After clarifying the data structure to process, the following operations are performed



Generation

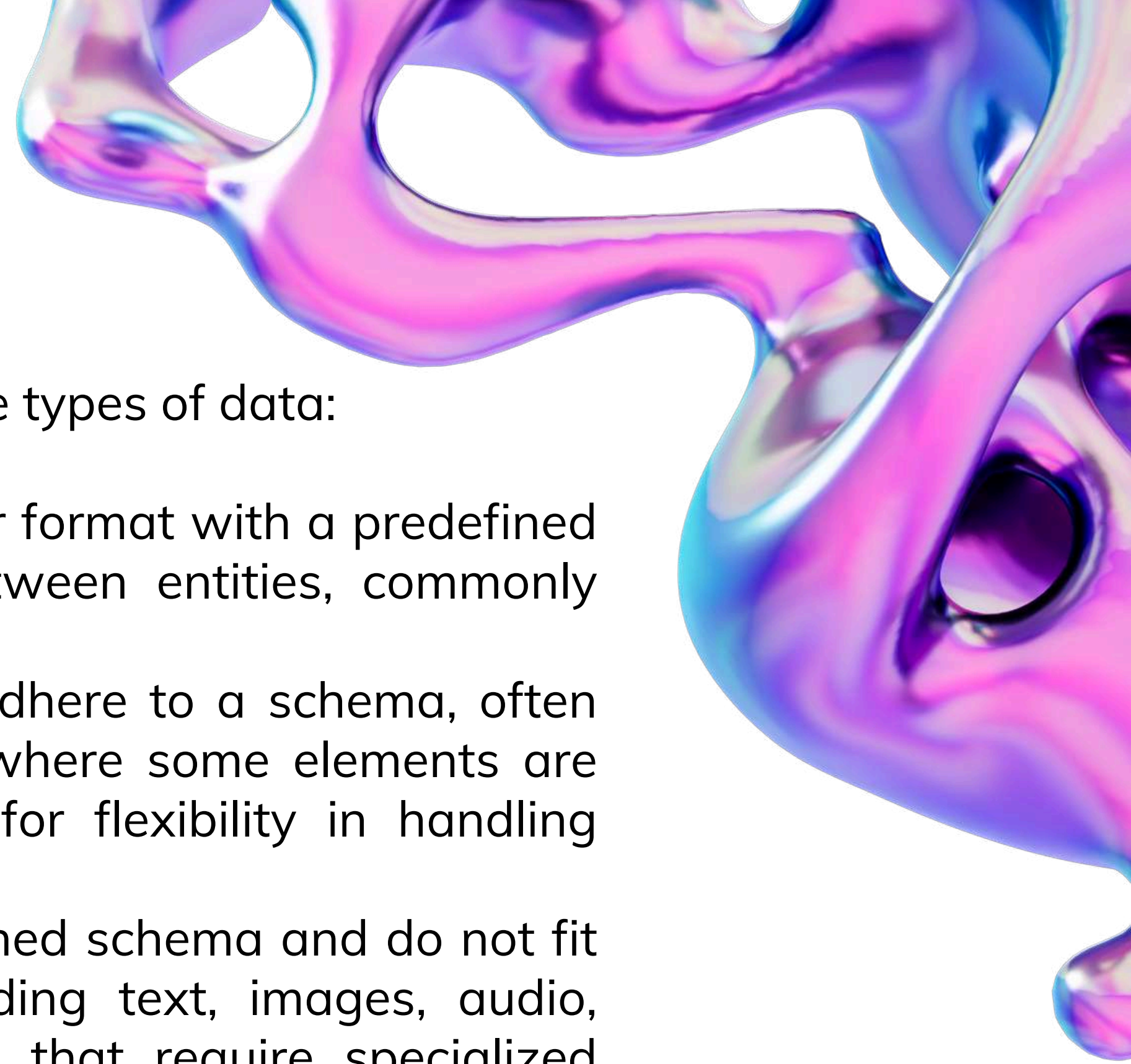
Data generation marks the inception of big data. Various **sources** contribute to the vast pool of big data including trading data, mobile data, user behavior, sensing data, Internet data, and other. For instance, the **Internet** has emerged as a significant contributor to big data, with substantial volumes of data generated daily through activities such as search queries, chat conversations, and microblog posts; they hold valuable insights into collective behaviors and trends. Leveraging accumulated big data enables the extraction of useful information.



Generation

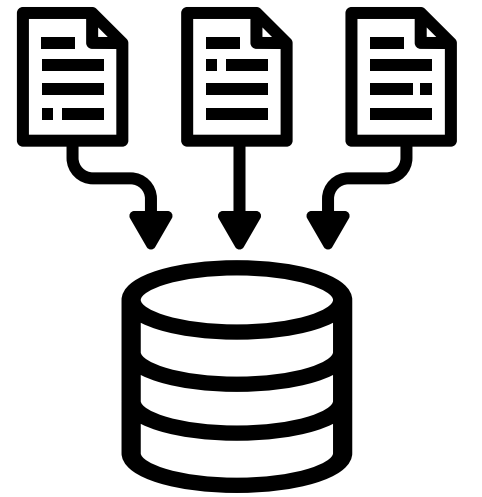
Following what has been said, there can be three types of data:

- **Structured data:** data organized in a tabular format with a predefined schema and well-defined relationships between entities, commonly found in relational databases.
- **Semi-structured data:** data that partially adhere to a schema, often represented in formats like XML or JSON, where some elements are organized while others are not, allowing for flexibility in handling diverse and dynamic data.
- **Unstructured data:** data that lack a predefined schema and do not fit into traditional relational databases, including text, images, audio, video, emails, and other forms of content that require specialized processing techniques for analysis.



Ingestion

The process of obtaining and importing for storage or use in a database



Data ingestion usually comes in three flavour:

- **Batch processing:** involves loading "batches" of data into a storage system at scheduled intervals, usually during low-traffic hours to minimize impact on other operations. Batch processing typically organizes data into groupings spanning hours, days, weeks, or months. Common endpoints for batch data include Amazon Redshift Amazon Athena, or Google BigQuery.
- **Streaming data:** streaming data is employed when continuous data updates are required. It continually transfer data in real-time from source to destination. For extensive, unstructured datasets where timeliness is paramount, the ELT method is preferable over ETL. Streaming processing platforms provide reduced latency compared to batch systems since data events are promptly processed following their incidence. Some useful tools and platform are (Apache Kafka, Apache Pulsar, Apache Flink)
- **Lambda Architecture:** deployment model for data processing that organizations use to combine a traditional batch pipeline with a fast real-time stream pipeline for data access. It is a common architecture model in IT and development organizations' toolkits as businesses strive to become more data-driven and event-driven in the face of massive volumes of rapidly generated data, often referred to as "big data."

Ingestion

Apache Kafka

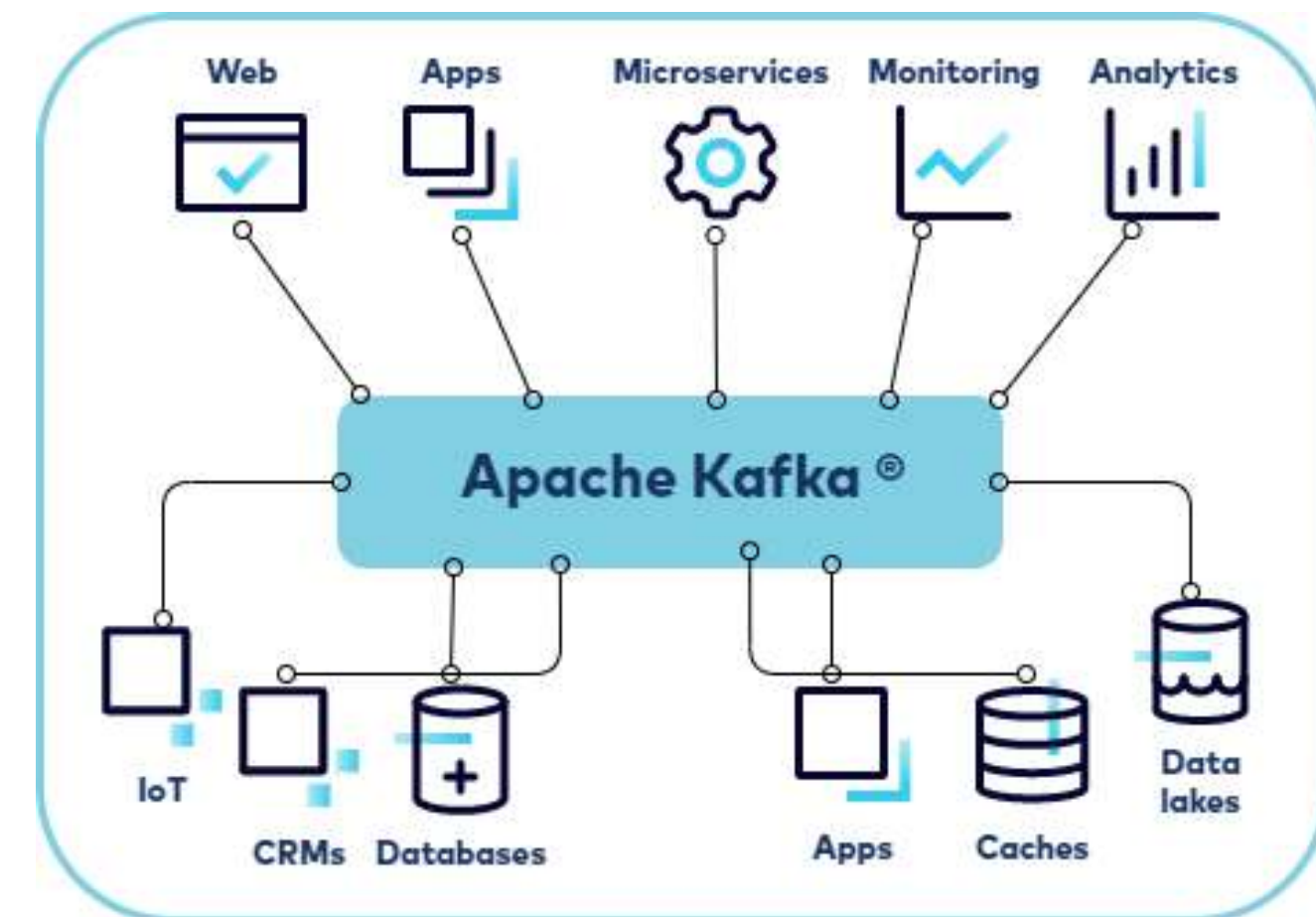


Apache Kafka is a **distributed data streaming** platform that enables storing and processing streams of records in real-time. It is designed to handle data streams from multiple sources by distributing them to multiple consumers simultaneously. It facilitates the movement of large amounts of data from one point to another in real-time.

Developers can leverage Kafka's functionalities through four APIs:

- **Producer API:** allows applications to publish data streams to Kafka.
- **Consumer API:** enables applications to subscribe to data streams from Kafka for processing.
- **Streams API:** adds advanced functionalities for continuous processing of data streams within Kafka, allowing analysis, aggregation, and transformation of records.
- **Connector API:** enables developers to integrate external data sources with Kafka, facilitating import and export of data.

Kafka operates as a fault-tolerant and highly available distributed cluster that can span across multiple servers and even multiple data centers. It ensures consistent performance and reliability, regardless of whether you have 50 KB or 50 TB of persistent storage on the server.



Ingestion

Apache Kafka



Advantages:

- **Scalability:** Kafka's partitioned log model allows for distributing data across multiple servers, making it scalable beyond the limits allowed by a single server.
- **Speed:** Kafka decouples data streams to ensure very low latency, making it extremely fast.
- **Durability:** partitions are distributed and replicated across many servers, and data is all written to disk. This helps protect against server failures, making the data highly fault-tolerant and durable.

Disadvantages:

- **Lack of monitoring tools:** there is no system in place to manage performance and operation usage effectively.
- **Performance degradation:** the system performs routine activities within brokers such as message deletion, which helps save memory but strains the system due to the high number of disk accesses required.
- **Difficulty in data migration:** admin migration tools are inefficient, and moving topics and changing the number of partitions over time is very costly and risky, especially when dealing with data that must not be lost.



Ingestion

Apache NiFi



Apache NiFi is an integrated data logistics platform for **automating** the movement of data between different systems. It provides **real-time control** that simplifies data movement management between any source and any destination. It is independent of data source, supporting disparate and distributed sources of various formats, schemas, protocols, speeds, and sizes. Apache NiFi enables monitoring of data in real-time. It is built on technology previously called "**Niagara Files**," developed and widely used within the NSA. It is flexible, extensible, and suitable for a wide range of devices, from small network edge devices like a Raspberry Pi to enterprise data clusters and the cloud.

Ingestion

RTA Data Pipeline Guiding Principles

Step-by-step approach to help you decide how to process different data sources:

1. Discover your **data sources** (databases, APIs, ...)
2. Start **collecting** them with NiFi to see what the data looks like. You can do basic cleanup, transformations, enrichment, and field definitions. Then you can route to Kafka for additional processing.
3. Determine which **RTA(Real-Time Analytics)** you need. Joining data streams like weather and transit on latitude and longitude; transforming data further; or making computations, aggregations.
4. You can store data for future usage thanks **Flink SQL** that can store in an underlying streaming table format in Apache Paimon or in an open lakehouse format like Apache Iceberg.
5. Finally you need to run on an enterprise data platform that provides things like monitoring, fault tolerance, auto-scaling, data security, data governance, and control of the underlying hybrid cloud environment.

Ingestion

OLAP vs OLTP

OLAP (On-Line Analytical Processing) refers to software designed for **interactive and rapid analysis of information**. It is a database system oriented towards analysis, allowing quick extraction of insights from large volumes of aggregated data. It serves as the technological component situated between the Data Warehouse and reporting/visualization tools. It is a Business Intelligence tool aimed at leveraging information extracted from large databases to support strategic business decisions.

OLTP (On-Line Transaction Processing) can be translated as software used to **manage applications dealing with transactions**. OLTP's role is to record the insertion, updating, and eventual deletion of data (INSERT, UPDATE, and DELETE) during a transaction through simple and concise queries, characterized by streamlined storage and processing processes.



Ingestion

OLAP vs OLTP

Criteri	Origine dati	Struttura	Modello	Volume	Tempo di risposta
OLAP	Historical and aggregated data from multiple sources.	Multidimensional or relational databases.	Star schema, snowflake schema, or other analytical models.	terabyte (TB) and petabyte (PB).	Long, typically in seconds or minutes.
OLTP	Real-time transactional data from a single source.	Only relational databases.	Normalized or denormalized models.	gigabyte (GB).	Short, typically in milliseconds.

Ingestion

Change Data Capture

Change data capture (CDC) refers to the process of identifying and recording alterations made to data in a database and delivering those modifications in real-time to a downstream process or system. This operation maintains system synchronization and enables dependable data replication and seamless cloud migrations without downtime.

There are several methods to implement a change data capture system:

- **Log-based CDC.** Represents the most efficient approach to implementing CDC. When a new transaction enters a database, it is logged into a log file without affecting the source system. These alterations can be extracted and moved from the log.
- **Query-based CDC.** Data in the source is queried to detect alterations. However, this approach is more intrusive to the source systems as it necessitates something like a timestamp within the data itself.
- **Trigger-based CDC.** Adjustments are made to the source application to trigger the recording of changes to a change table, which are then moved. Nonetheless, this approach diminishes database performance due to the need for multiple writes whenever a row is updated, inserted, or deleted.

Ingestion

CDC and **OLAP** can solve data latency issues; it allows fresh data to be available for processing in ETL pipelines.

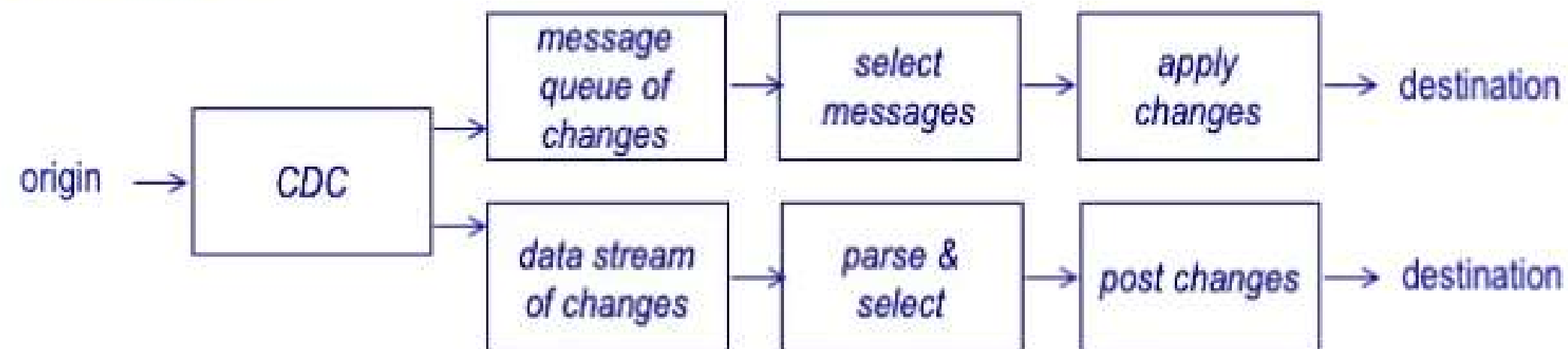
Advantages:

- 1.Reduced latency for Datawarehouse workloads.
- 2.Increased data freshness, near real-time or micro-batch ETL.

Disadvantages:

- 1.Increased complexity.
- 2.CDC applications can add extra cost.

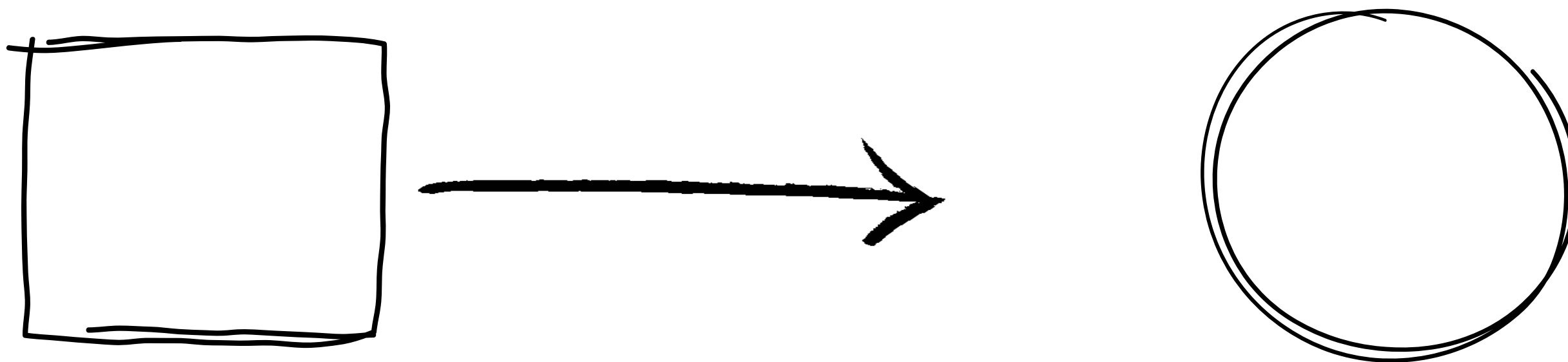
Changed Data Capture



Processing

In this phase, a series of processes are executed to **convert** the data into the required format for the target data repository. These processes incorporate automation and governance mechanisms to handle repetitive tasks, such as business reporting, and to ensure consistent **cleansing** and **transformation** of the data.

For example, if the data stream arrives in a nested JSON format, the transformation stage will focus on extracting the JSON structure to retrieve the key fields necessary for analysis.



Storage

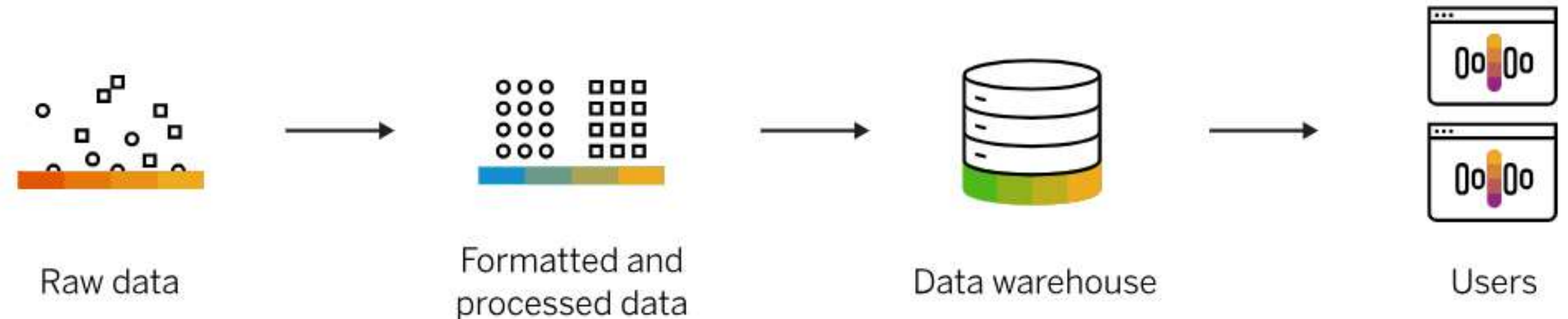
Data storage is the **preservation** of information using technology developed specifically to store and make it accessible as needed. Data storage refers to the use of recording media to store data using computers or other devices.

Different types of storage systems:

- **Distributed File Systems:** File systems like the Hadoop File System (HDFS) provide the ability to store large volumes of unstructured data reliably on commodity hardware. Designed for handling large data files, they are well-suited for rapid data ingestion and bulk processing.
- **NoSQL Databases:** Perhaps the most prominent category of big data storage technologies. They employ data models distinct from the relational world, often lacking the transactional properties of ACID.
- **NewSQL Databases:** A modern iteration of relational databases aiming for scalability similar to NoSQL databases while retaining the transactional guarantees of traditional database systems.
- **Big Data Querying Platforms:** Technologies offering query interfaces for big data repositories like distributed file systems or NoSQL databases. The primary focus is on providing a high-level interface.

Storage

Data Warehouse



A **data warehouse** is a type of data management system designed to enable and support business intelligence activities, particularly analytics. They serve to execute queries and analyses and contain large amounts of historical data. It centralizes and consolidates large amounts of data from multiple sources. A typical data warehouse often includes:

- A **relational database** for storing and managing data;
- An **ELT solution** for preparing data for analysis;
- **Statistical analysis**, reporting, and data mining features;
- **Customer analytics** tools for visualizing and presenting data to business users;
- Other more sophisticated analytical applications that generate insights actionable by applying data science algorithms and artificial intelligence (AI).

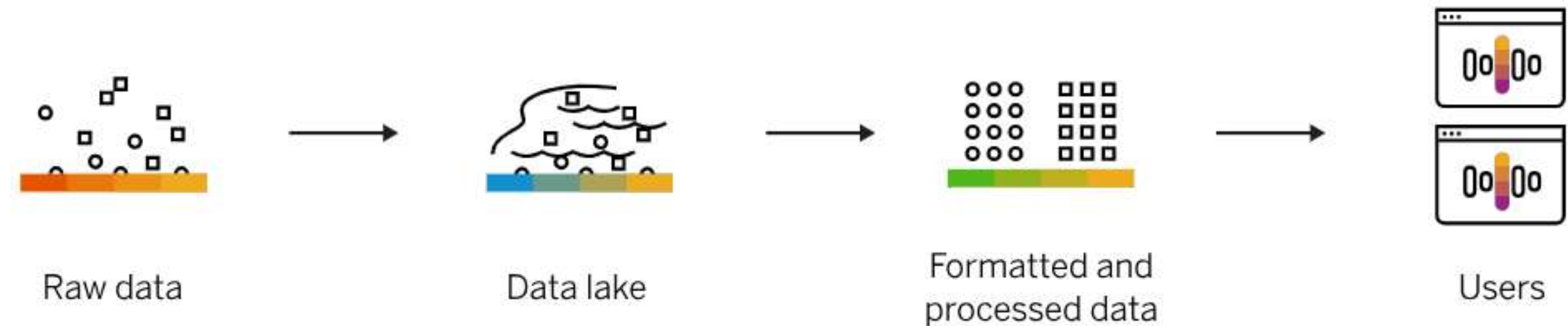
Data warehouses are:

- **Object-oriented.** They can analyze data on a specific subject or functional area.
- **Integrated.** They create consistency among different types of data from disparate sources.
- **Non-volatile.** The data is stable and does not change.

Data warehouse analysis examines change over time.

Storage

DataLake



A data lake is a **repository** dedicated to storing, analyzing, and correlating data in its **native format**, both structured and unstructured. Its distinctive feature is the ability to retrieve and organize data based on the desired analysis type.

This innovation, represents a significant evolution and improvement of the tool. While data warehouses require data modeling before storage, limiting flexibility in data usage, data lakes do **not require preliminary data structuring**. Instead, their ability to accommodate structured, semi-structured, and unstructured data makes them extremely flexible.

The **advantages** of data lakes include:

- Significant expansion of **accessible** information
- **Flexibility** in querying data and applying a wide range of tools
- Reduction of **storage costs** and virtually infinite scalability
- Reduction of **time to market**
- Sharing and democratization of access to information

This new data architecture offers a more dynamic and scalable approach to data analysis, enabling broader and more flexible use of business information.

Storage

Data Lakehouse



A Data Lakehouse is a data architecture that **combines the architectures and capabilities of both a data warehouse and a data lake**.

This type of configuration **reduces redundancies** and simplifies various aspects of data management, from architecture design to maintenance, optimization, and data governance. The major advantage is primarily that data is available in less time for the warehouse and its uses as it does not need to be moved from one platform to another for analysis.

While the architecture of a Data Lakehouse can be very **beneficial** for some areas, in other cases, it may not significantly reduce the overall workload required, and consequently, the benefits it generates may not meet expectations. While the adoption of a Data Lakehouse can generally bring benefits, there are also some exceptions. Some vendors argue that it is advantageous to have what they define as a single source of truth or a single copy of the data, but separating the data lake from the data warehouse provides **greater clarity** in governance and alignment of consumption models that would be blurred in the Lakehouse, potentially complicating data governance and quality.

Consumption

Data pipelines serve a multitude of functions, here are three applications of their usage within enterprises:

- **Exploratory Data Analysis (EDA)**: EDA refers to the crucial process of conducting investigations on data. Its aim is to uncover patterns, identify anomalies, test hypotheses, and validate assumptions, often using summary statistics and graphical representations. EDA plays a fundamental role in determining the optimal approach to manipulate data sources to extract necessary insights, thereby facilitating data scientists in their quest for meaningful answers;
- **Data Visualizations**: data visualizations present data using charts or animations. These visual representations effectively convey intricate data relationships and insights driven by data, making complex information easily understandable;
- **Machine Learning**: a machine learning data pipeline is a comprehensive structure that coordinates the entire process of directing data into input and extracting output from a machine learning model (or a set of multiple models). It includes raw data input, features, outputs, the machine learning model itself, model parameters, and prediction results. By using statistical methods, algorithms are trained to make classifications or predictions, thereby uncovering essential insights in data mining projects.

Orchestration

Data pipeline orchestration involves the coordination, management, and regulation of data flow and processing within pipelines. It ensures **precise execution timing, sequencing, and operational parameters for each task** within the pipeline. The interconnected elements of a data pipeline must operate **cohesively**, guided by predefined operational principles dictating when tasks activate, respond, and contribute to the overarching objective of data processing.

It goes beyond simple task execution, focusing on orchestrating a dynamic, interconnected process where outputs seamlessly transition to subsequent stages, with each step depending on the successful completion of its predecessors.

Orchestration

The Need for Orchestration in Data Pipelines

- **Network Complexity:** data pipelines are complex networks of data movement. They can cover various computing environments, involve different areas of control and ownership, and switch between batch and real-time operations. Orchestrating these pipelines needs a system that manages these complications while ensuring flawless execution in the right order.
- **Resource Optimization:** inefficiently managed data pipelines can use up a lot of computer power. Paying attention to efficiency when managing them can save a lot of money. This means adjusting or changing resources for pipeline tasks, and not redoing pipelines when it's not necessary.

Orchestration

The Need for Orchestration in Data Pipelines

- **Operational Resilience:** data pipelines can encounter different problems such as breakdowns, errors in data quality, interruptions in processing, and sudden changes in data formats and amounts. It is possible to make data pipelines more resilient to these and other challenges by including features like spotting errors early, automatically trying again when things go wrong, and having backup plans
- **Dynamic Scalability:** lots of data pipelines deal with changing amounts of data, not just from day to day, but also depending on the time of year. Managing data pipelines can include making sure they can handle these changes by automatically adjusting how much computer power they need, so they work well no matter how much data is coming through.

Orchestration

Source: <https://www.ascend.io/blog/what-is-data-pipeline-orchestration-and-why-you-need-it/>

Key Components of Data Pipeline Orchestration

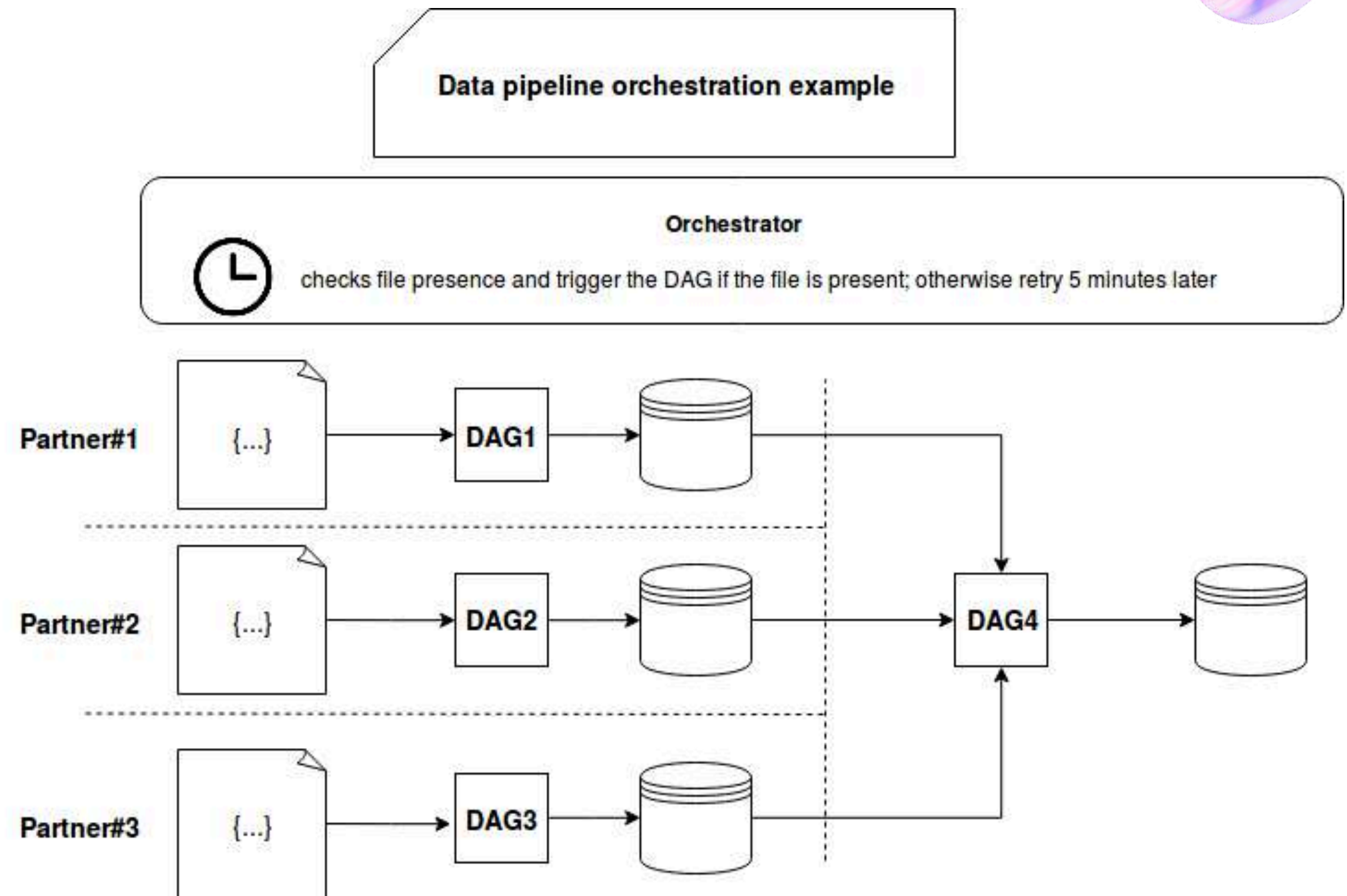
- **Workflow Definition:** it details the execution sequence of tasks, their dependencies, and the conditions under which they run. This is often expressed in a directed acyclic graph (DAG), where nodes represent tasks and edges define dependencies.
- **Task Scheduling:** task scheduling is the temporal aspect of orchestration, determining when each task should actually be executed, with which datasets, and on which compute resources.
- **Dependency Management:** dependency management ensures tasks are executed in an order that respects the interdependencies. This means the sequences in the task network commence for each task only once all its upstream dependencies have successfully completed.
- **Resource Management:** orchestration also involves the allocation and optimization of computational resources. Each task that is generated by the orchestration needs CPU, memory, and I/O capacity assigned and a compute engine designated to execute, while also leveraging “warm” resources and “packing” tasks into workloads that minimize contention and waste.
- **Error Handling and Recovery:** robust orchestration can handle task failures that arise from intermittent resources, unanticipated data structures, logic failures, and many other sources of error. Techniques for this include setting policies for retries, fallbacks, or alerts, and ensuring the system can recover from unexpected states.
- **Monitoring and Logging:** throughout the orchestrated workloads, continuous monitoring and detailed logging of each task’s performance and outcome are essential for maintaining operational visibility. This aids in debugging, performance tuning, and compliance with audit requirements.

Orchestration

Example

The orchestration approach can be presented with **DAG abstraction** used by Apache Airflow to define data processing workflows.

For instance, you can have the a data pipeline composed of the steps integrating the data coming from our different partners and one final DAG to make some final computation on them:



Choreography

On the other hand, the **choreographic approach** is based on the idea that a central control element, the orchestrator (i.e., middleware such as Kubernetes) is redundant. The individual components, i.e., microservices, must be capable of **self-managing** without external intervention and coupled in a loose manner (loose coupling) and not in a tight manner (tight coupling), in other words, in a general and not too tight manner.

This means that the **loosely coupled microservices** must be able to provide the greatest possible value to the business on their own, without directly impacting other components. In this way, its complexity is reduced because there is no controller to program and manage. Additionally, as one might guess, there is no single node, i.e., a potentially critical element for the entire infrastructure in the event of problems.

Choreography



In a Choreography approach, a component called an "**event broker**" (or message broker) is used. Choreography occurs according to a sequence of steps. Each microservice that has performed an action sends a message via certain channels. Other microservices, subscribed to that specific channel, receive the message. At that point, they themselves know what to do because they're designed to respond automatically to certain events (event driven).

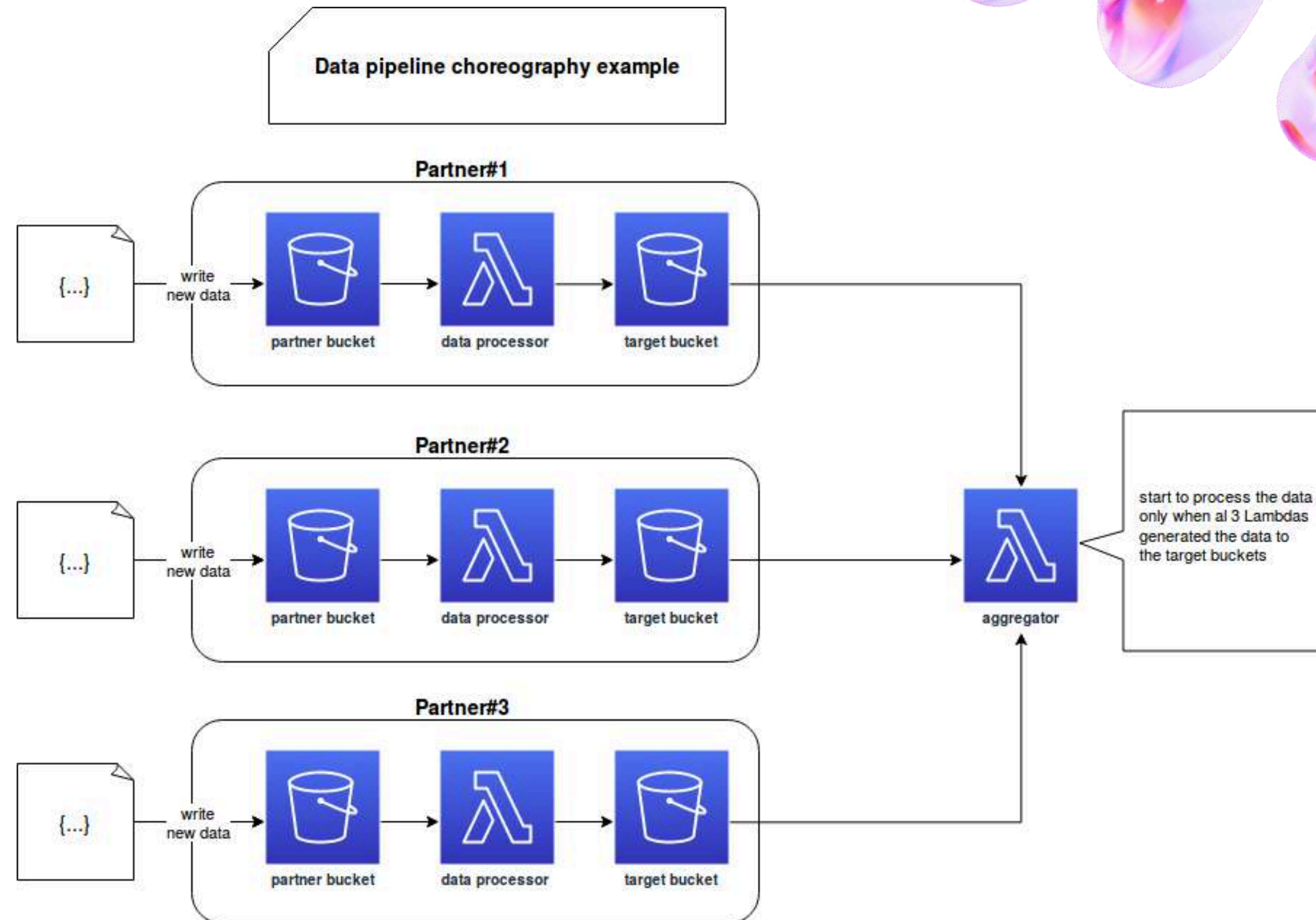
In this sense, **communication occurs asynchronously**, and, returning to the analogy of the choreographed dance, coordination takes place between dancers (the microservices) who listen to the music (the Event Broker) in order to dance.

Using loose service coupling logic allows microservices to be changed (added or removed as necessary) without rendering the underlying logic unserviceable and in need of being rewritten.

Choreography

Choreography is a set of decoupled micro services knowing what they expect and what they spit out.

Each micro service is self-contained and communicate **via message/event driven architecture**. Each service knows what event to listen to, how to react to event and what event to produce as the outcome which other services in the flow might be listening to.



Use Case

- 1) **Data Generator**: Every day, on a virtual machine, two datasets are transferred. One containing about 100 bank transfers in CBI format and the other with approximately 8 million invoices in QAS format.
- 2) **Data Ingestion and Processing**: Every night, at 1:00 AM, a batch process is initiated to process the invoice file, which includes cleaning/pre-processing the data and loading them into MongoDB. This process runs at night as it takes about 2 hours to complete. At 7:00 AM, a similar process is run for the bank transfer dataset, converting the CBI files in a CSV .
- 3) **Data Storage**: At this point, the data is stored in MongoDB in two different collections, each with a specific format.
- 4) **Data Consumption**: Subsequently, the bank transfer dataset is processed by a Named Entity Recognition (NER) network to recognize important codes and payers of the transfers, in order to proceed with reconciliation.

Use Case

1) **Data Generator**: Every day, on a virtual machine, two datasets are transferred. One containing about 100 bank transfers in CSV format and the other with approximately 8 million invoices in QAS format.

Fatture

N. documento	Rip	Pos	Pp	BPartner	Contratto	Conto contr.	Data reg.	Scad.netto	Imp. divisa int.	TD	Riferimento	Cognome	Nome
1234567889	000	0003	000	1234567890	1234567890	123456789012	22.02.2024	23.03.2024	110,01	BO	123456789012	SABINO	GIUSEPPE

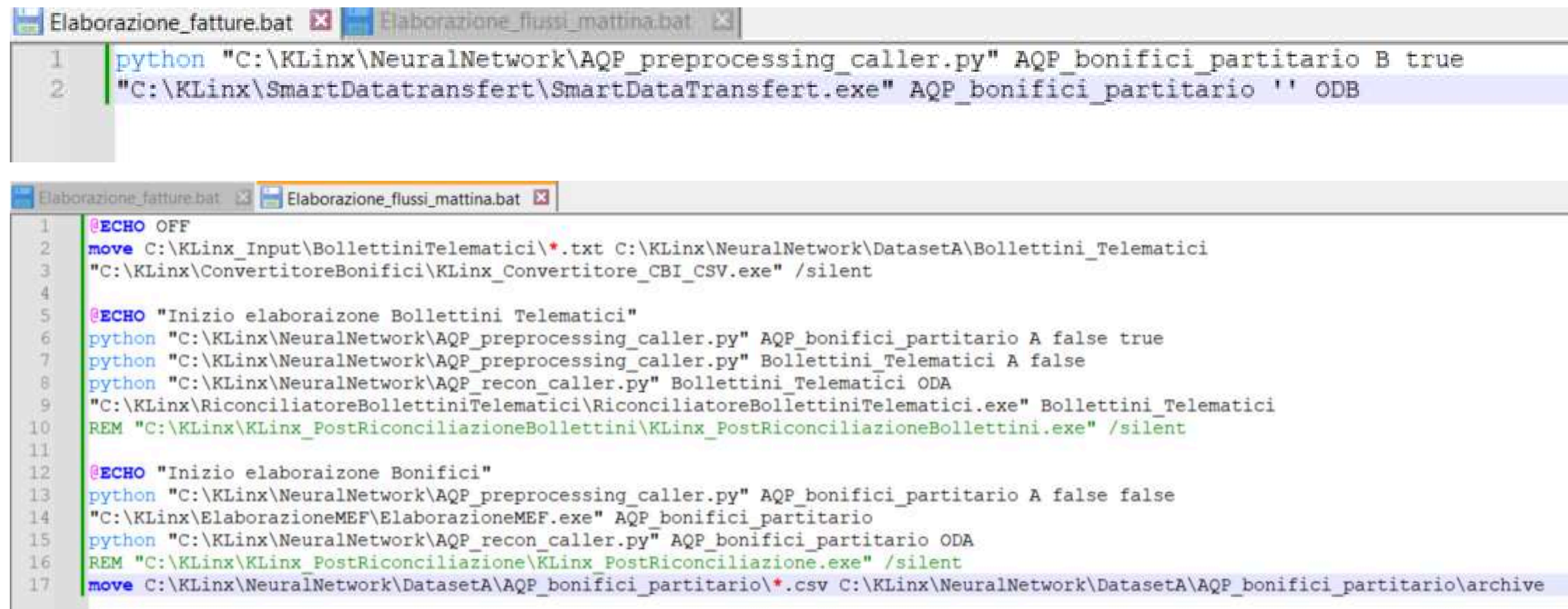
Bonifici

DATA_CONTABILE	DATA_VALUTA	CRI	RIF_CLIENTE	CAUSALE_ABI	ORDINANTE	DIVISA	IMPORTO	CAUSALE	ABI	CAB	CONTO
20240228	20240227	1234567890		07	GIUSEPPE SABINO	EUR	110.01	Pagamento fattura numero 123456789012 da parte di Sabino Giuseppe	12345	12345	000012345678

In formato Csv per una semplicità di rappresentazione visiva

Use Case

2)**Data Ingestion and Processing:** Every night, at 1:00 AM, a batch process is initiated to process the invoice file, which includes cleaning/pre-processing the data and loading them into MongoDB. This process runs at night as it takes about 2 hours to complete. At 7:00 AM, a similar process is run for the bank transfer dataset, converting the CBI files in a CSV .



```
Elaborazione_fatture.bat Elaborazione_flussi_mattina.bat
1 python "C:\KLinx\NeuralNetwork\AQP_preprocessing_caller.py" AQP_bonifici_partitario B true
2 "C:\KLinx\SmartDataTransfert\SmartDataTransfert.exe" AQP_bonifici_partitario ' ' ODB

Elaborazione_fatture.bat Elaborazione_flussi_mattina.bat
1 @ECHO OFF
2 move C:\KLinx_Input\BollettiniTelematici\*.txt C:\KLinx\NeuralNetwork\DatasetA\Bollettini_Telematici
3 "C:\KLinx\ConvertitoreBonifici\KLinx_Convertitore_CBI_CSV.exe" /silent
4
5 @ECHO "Inizio elaborazione Bollettini Telematici"
6 python "C:\KLinx\NeuralNetwork\AQP_preprocessing_caller.py" AQP_bonifici_partitario A false true
7 python "C:\KLinx\NeuralNetwork\AQP_preprocessing_caller.py" Bollettini_Telematici A false
8 python "C:\KLinx\NeuralNetwork\AQP_recon_caller.py" Bollettini_Telematici ODA
9 "C:\KLinx\RiconciliatoreBollettiniTelematici\RiconciliatoreBollettiniTelematici.exe" Bollettini_Telematici
10 REM "C:\KLinx\KLinx_PostRiconciliazioneBollettini\KLinx_PostRiconciliazioneBollettini.exe" /silent
11
12 @ECHO "Inizio elaborazione Bonifici"
13 python "C:\KLinx\NeuralNetwork\AQP_preprocessing_caller.py" AQP_bonifici_partitario A false false
14 "C:\KLinx\ElaborazioneMEF\ElaborazioneMEF.exe" AQP_bonifici_partitario
15 python "C:\KLinx\NeuralNetwork\AQP_recon_caller.py" AQP_bonifici_partitario ODA
16 REM "C:\KLinx\KLinx_PostRiconciliazione\KLinx_PostRiconciliazione.exe" /silent
17 move C:\KLinx\NeuralNetwork\DatasetA\AQP_bonifici_partitario\*.csv C:\KLinx\NeuralNetwork\DatasetA\AQP_bonifici_partitario\archive
```

Use Case

3)**Data Storage**: At this point, the data is stored in MongoDB in two different collections, each with a specific format.

```
_id: 133459
DATA_CONTABILE: 2024-02-28T00:00:00.000+00:00
DATA_VALUTA: 2024-02-27T00:00:00.000+00:00
CRI: "1234567890"
RIF_CLIENTE: ""
CAUSALE_ABI: "07"
ORDINANTE: "GIUSEPPE SABINO"
DIVISA: "EUR"
IMPORTO: 110.01
CAUSALE: "Pagamento fattura 123456789012 da parte di Sabino Giuseppe"
ABI: "12345"
CAB: "12345"
CONTO: "000012345678"
_loading_time: "20240228_0712"
_assigned: 0
_processed: 0
_processed_by_SQL: 0
_id_elaborazione: 148
```

```
_id: 195043
N documento: "1234567889"
Rip: "000"
Pos: "0003"
Pp: "000"
BPartner: "1234567890"
Contratto: "1234567890"
Conto contr: "123456789012"
Data reg: 2022-02-24T00:00:00.000+00:00
Scadnetto: 2022-03-24T00:00:00.000+00:00
Imp divisa int: 110.01
TD: "B0"
Riferimento: "123456789012"
Nome_Cognome: "GIUSEPPE SABINO"
Nome_Cognome_copy: "SABINO GIUSEPPE"
Rip_Pos_Pp: "0000003000"
_loading_time: "20240228_0325"
_assigned: 0
_processed: 0
_processed_by_SQL: 0
_id_elaborazione: 148
```


Use Case

4)**Data Consumption**: Subsequently, the bank transfer dataset is processed by a Named Entity Recognition (NER) network to recognize important codes and payers of the transfers, in order to proceed with reconciliation.

```
_id: ObjectId('65d773be79d9674ca08a27e3')
NN_match_id: 14628
NN_scores: '{"CAUSALE_Riferimento": 2, "CAUSALE_BPartner": 0, "CAUSALE_Contra..."'
NN_recon_level: 1
NN_tipo_abbinamento: "REGOLARI"
Executed_by: "default"
Execution_time: "20240222_1717"
_id_elaborazione: 149
ORDINANTE_Nome_Cognome_SCORE: ""
ORDINANTE_Nome_Cognome_THRESHOLD_1: ""
ORDINANTE_Nome_Cognome_THRESHOLD_2: ""
Causale_Nome_Cognome_SCORE: ""
Causale_Nome_Cognome_THRESHOLD_1: ""
Causale_Nome_Cognome_THRESHOLD_2: ""
▶ A: Array (1)
▶ B: Array (1)
```

Modern Data Pipelines

The rise of affordable and elastic cloud services has enabled new data management options and necessitated new requirements for building data pipelines to information.

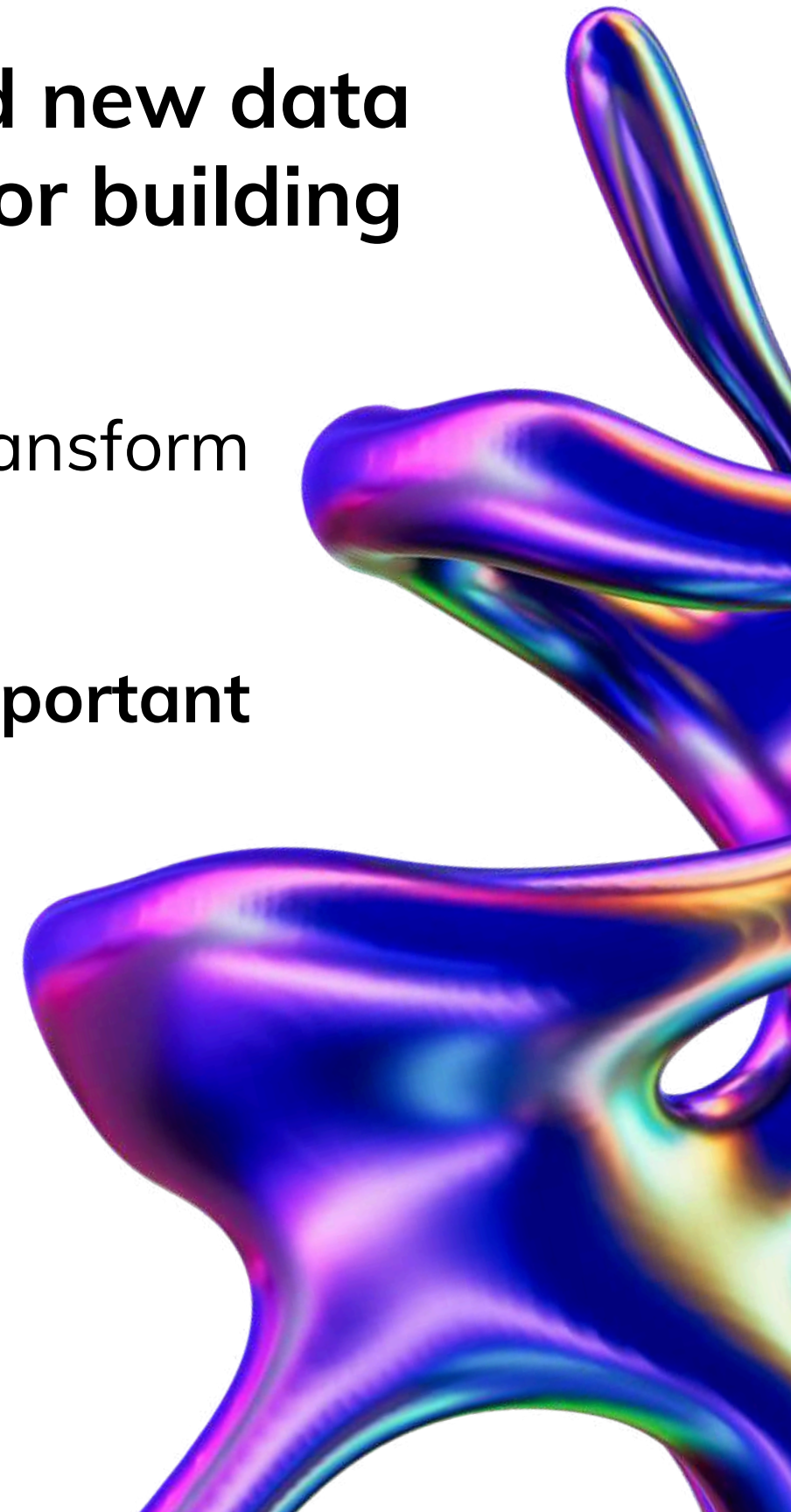
Data pipelines automate the transfer of data from place to place and transform that data into specific formats for certain types of analysis.

Today's modern data pipelines have arisen in response to **three important paradigm** shifts in the software industry.

Paradigm shift #1: On-premises ETL to cloud-driven ELT

Paradigm shift #2: Batch processing to continuous processing

Paradigm shift #3: Consolidation of systems for structured, semi-structured, and unstructured data



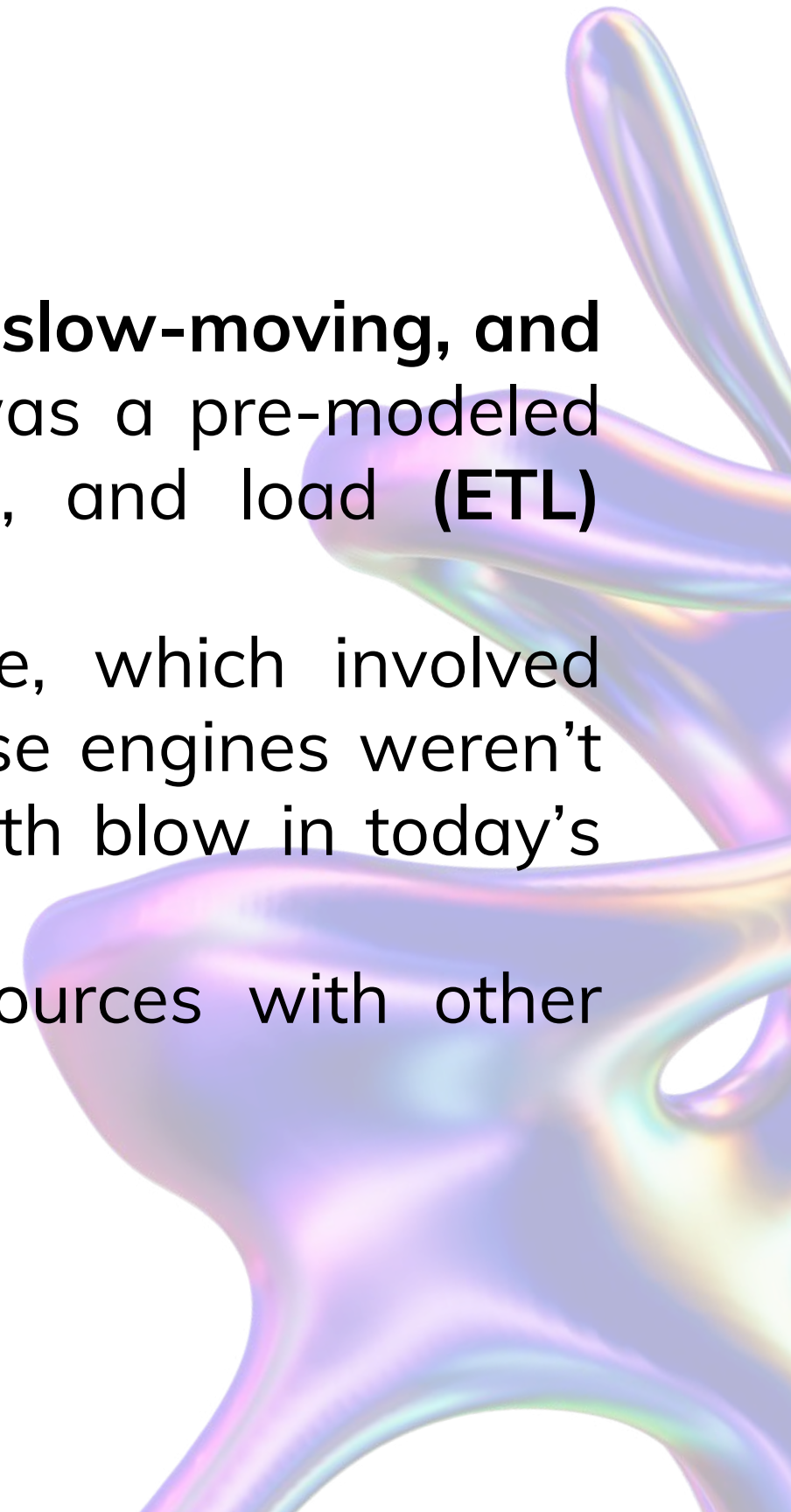
Modern Data Pipelines

Paradigm shift #1: On-premises ETL to cloud-driven ELT

Yesterday's data pipelines were designed to accommodate **predictable, slow-moving, and easily categorized data** from business applications. The destination was a pre-modeled data warehouse and the pipelines depended on extract, transform, and load (**ETL workflows**).

These traditional ETL operations used a separate processing engine, which involved **unnecessary data movement** and tended to be **slow**. Furthermore, these engines weren't designed to accommodate schemaless, semi-structured formats—a death blow in today's world of continuous and streaming data sources.

Traditional ETL architectures on premises, ETL jobs contend for resources with other workloads running on the same infrastructure



Modern Data Pipelines

Paradigm shift #1: On-premises ETL to cloud-driven ELT

Modern data integration workloads leverage the processing power of target data platforms in the cloud. Here, data pipelines are designed to extract and load the data first, and then transform it once the data reaches its destination (**ELT rather than ETL**).

Modern ELT architectures move these transformation workloads to the cloud, enabling superior **scalability and elasticity**.

It is worth mentioning that not every cloud solution is created equally though. The scalable and elastic ones have the capability to **separate each workload and scale** among readily available compute and storage resources, providing **limitless processing resources** of the cloud.

There's **no need to prepare data before loading it**. Raw data can be loaded and transformed later, once the requirements are understood, providing more options. A fully managed cloud service relieves the user of many of the management, maintenance, and provisioning challenges associated with on-premises infrastructure.

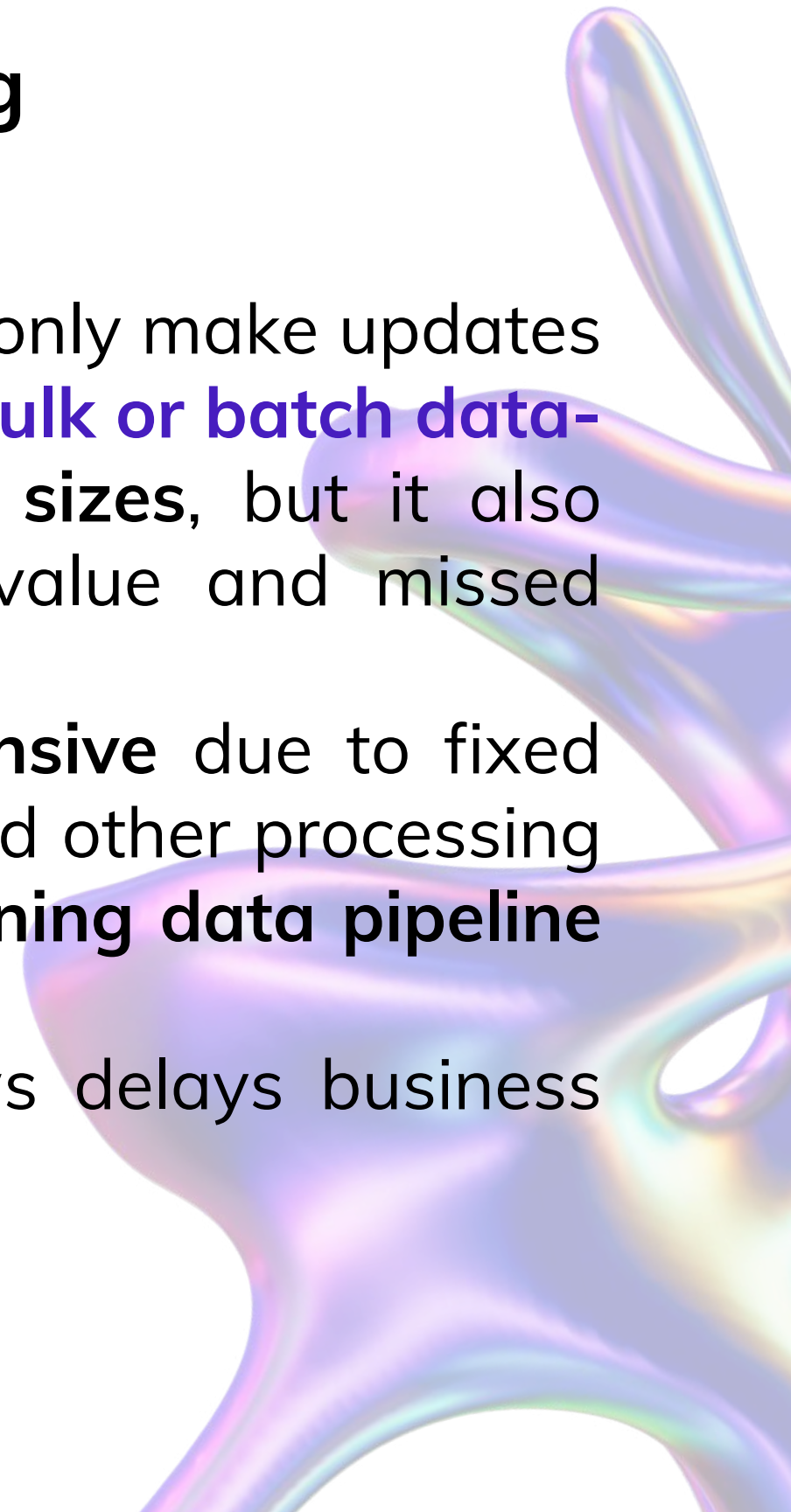
Modern Data Pipelines

Paradigm shift #2: Batch processing to continuous processing

Even though many businesses produce data continuously, but they may only make updates available for analytics at periodic intervals (weekly, daily, or hourly), via **bulk or batch data-update processes**. This ensures **good compression and optimal file sizes**, but it also **introduces latency**. Latency delays time to insight, leading to lost value and missed opportunities.

Until recently, enabling these continuous analysis scenarios was **expensive** due to fixed constraints on on-premises resources. Businesses also had to work around other processing jobs that used those same computing resources, which often meant **running data pipeline jobs during off hours**, commonly called “*maintenance windows*”.

Waiting to upload batch data during scheduled maintenance windows delays business results. **Data becomes stale**, and in some cases it is no longer relevant.



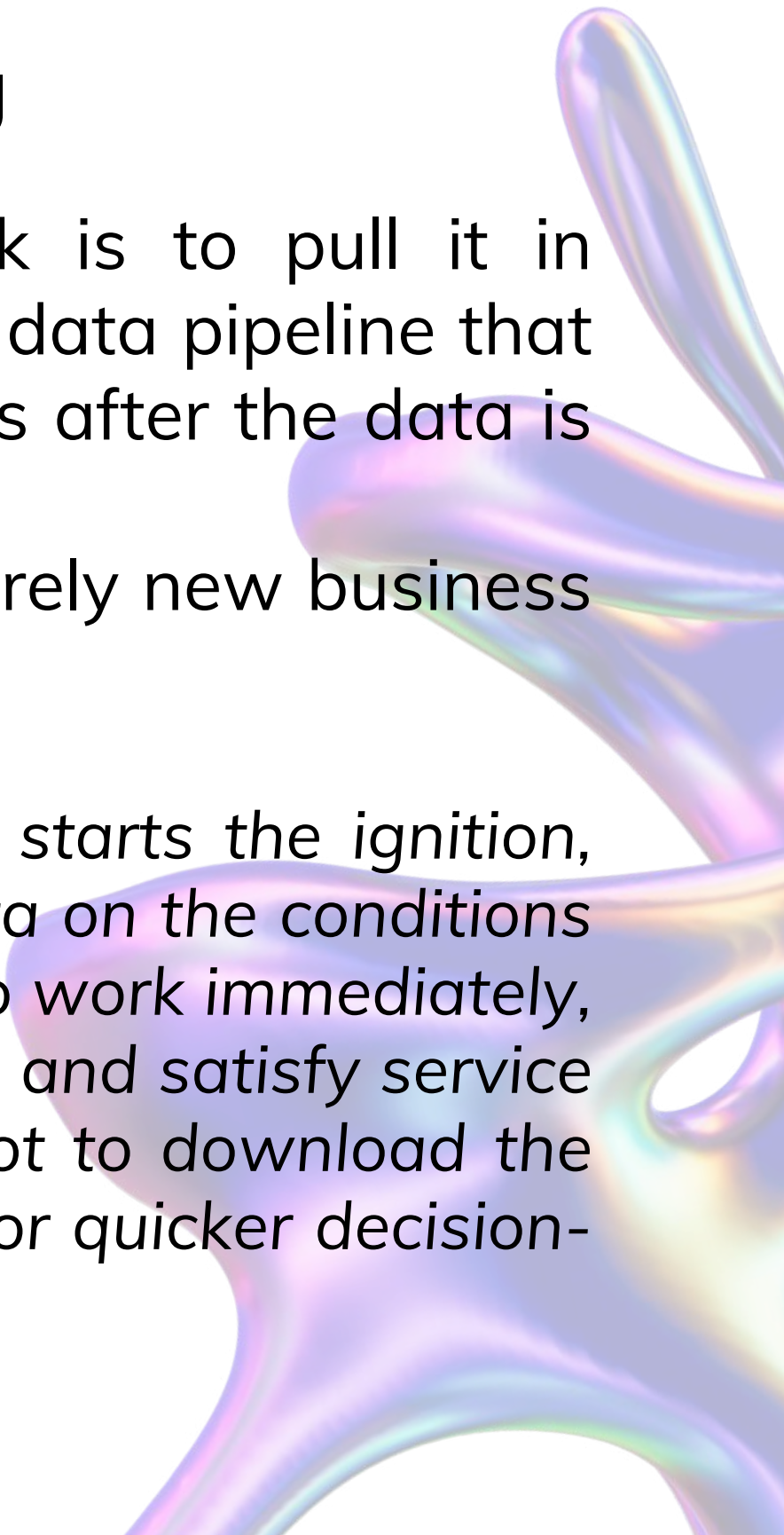
Modern Data Pipelines

Paradigm shift #2: Batch processing to continuous processing

A more natural way to ingest new data into your technology stack is to pull it in continuously, as data is born. **Continuous ingestion** requires a streaming data pipeline that makes new data available almost immediately (a few seconds or minutes after the data is generated).

Loading data continuously not only enables **more timely insights** but entirely new business models.

Example: vehicles emit data from hundreds of sensors every time the driver starts the ignition, turns the wheel, brakes, and accelerates— along with passively generated data on the conditions and performance of onboard equipment. If this data can be captured and put to work immediately, it can be used to monitor fleet performance, predict maintenance requirements, and satisfy service level agreements. Rather than waiting for these vehicles to come into a depot to download the information, a streaming ingestion service can collect it all the time, allowing for quicker decision-making.

An abstract, colorful, liquid-like graphic on the right side of the slide. It features flowing, organic shapes in shades of purple, blue, and pink, resembling a stylized splash or a modern logo element.


Modern Data Pipelines

Paradigm shift #3: Consolidation of systems for structured, semi-structured, and unstructured data

Database schemas are **constantly evolving**.

Modifications at the application level necessitate changes to the underlying database schema. Application developers may need to constantly interface with database administrators to accommodate these changes, which **delays the release** of new features.

However, legacy systems might stand in the way of this goal. **Data warehouses** typically ingest and store structured data, defined by a relational database schema. **Data lakes** store many types of data in raw and native forms, including semi-structured data and some types of unstructured data. It may be **challenging to integrate end-to-end data processing** across these different systems and services.



Modern Data Pipelines

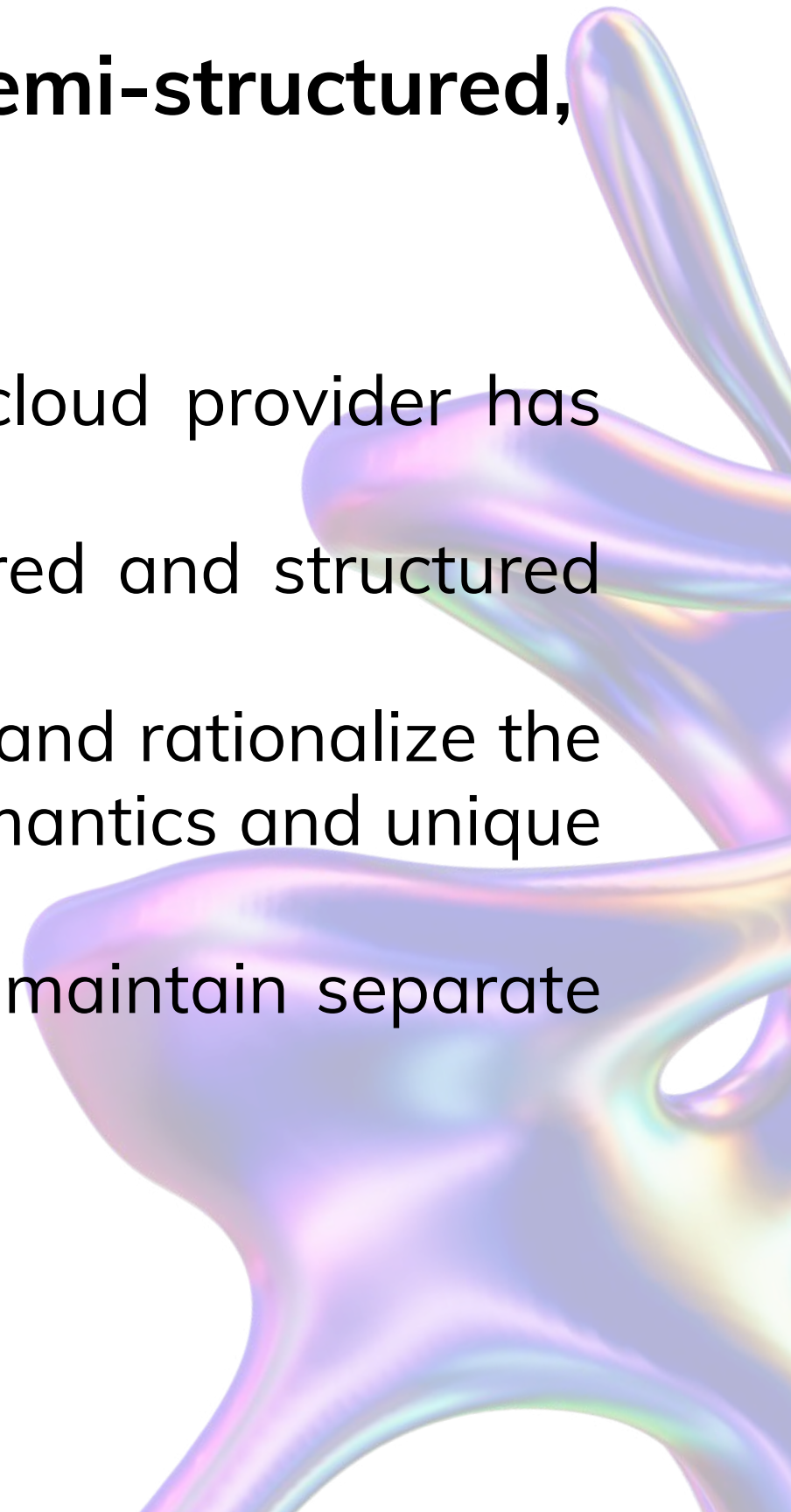
Paradigm shift #3: Consolidation of systems for structured, semi-structured, and unstructured data

Even **cloud-based systems** can impose limitations, since each public cloud provider has unique services for data storage, data transformations, and analytics.

Cloud providers may also **require separate services** to store unstructured and structured data.

Implementing end-to-end data pipelines requires the user to understand and rationalize the differences among all of these cloud environments, including different semantics and unique management procedures.

If it is not possible to rationalize these differences you will be forced to maintain separate data silos, which **complicates data governance**.



Medallion Architecture

Data pipeline use case

A **medallion architecture** is a data design pattern used to logically organize data in a lakehouse, with the goal of incrementally and progressively improving the structure and quality of data as it flows through each layer of the architecture (from *Bronze* \Rightarrow *Silver* \Rightarrow *Gold layer tables*). Medallion architectures are sometimes also referred to as "multi-hop" architectures.

Tools like **Azure Databricks** provides **Delta Live Tables (DLT)** that allow users to instantly build data pipelines with Bronze, Silver and Gold tables from just a few lines of code. And, with streaming tables and materialized views, users can create streaming DLT pipelines built on Apache Spark Structured Streaming that are incrementally refreshed and updated.

The **main benefits** on adopting a pipeline like this is one are:

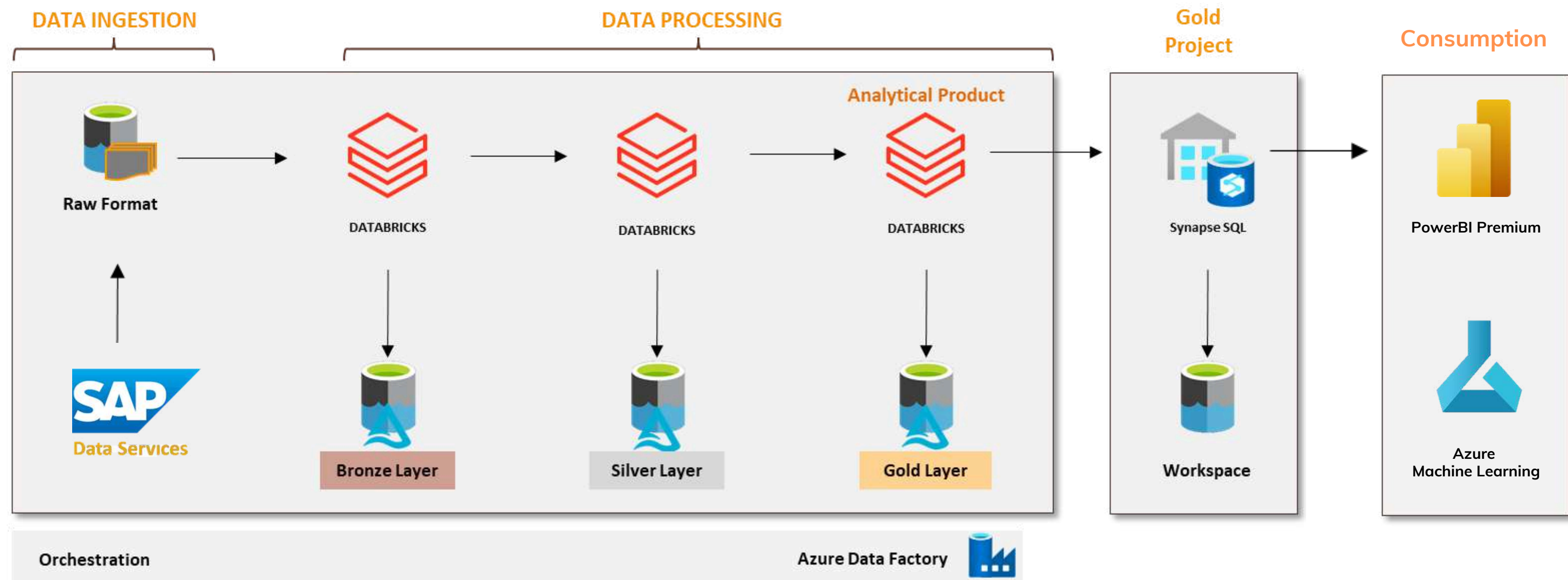
- Easy to understand and implement;
- Enables incremental ETL
- Can recreate your tables from raw data at any time
- ACID transactions, time travel



Medallion Architecture

Data pipeline use case

Here is the diagram of the complete data pipeline (after the cloud - migration) of a large Italian consumer goods company:





Challenges in Data Pipeline Management

As organizations attempt to take advantage of these paradigm shifts, fulfilling their expectations has become progressively more difficult.

Data of every shape and size is being generated, and **it often ends to be kept in databases, data lakes, and data marts** (in the cloud and on premises). However, different applications have different expectations for the delivery of data.

Organizations with legacy data pipeline architectures operate with a **fixed set of hardware and software resources**, which leads to reliability and performance issues. As data pipeline workloads increase, system administrators may need to **switch priorities** of other workloads contending for those same resources.

Complex data integration and orchestration tools may be necessary to create new data pipelines and hand-coded interfaces.

Challenges

Data pipelines involve a **complex chain of interconnected activities** that starts with a data source and ends in a data sink.

Data pipelines are **important** for data-driven organizations since a data pipeline can process data in multiple formats from distributed data sources with minimal human intervention, accelerate data life cycle activities, and enhance productivity in data-driven enterprises.

However, there are challenges in implementing data pipelines. The **challenges** include **data quality issues, infrastructure maintenance problems, and organizational barriers**.

Challenges

Infrastructure Challenges

Data pipelines are developed to **solve complex data infrastructure challenges**. However, data pipeline management has to **deal with some infrastructural challenges**:

- **Integrating new data sources**: data pipelines collect data from multiple distributed devices and make it available in a single access point. However, the data sources increase rapidly in most of the scenarios. Therefore, data pipelines should be able to **integrate the new data source** and also accommodate the data from that new source which is often **difficult due to many reasons**, such as:
 - The data source can be entirely different from the existing sources.
 - Format of the data produced by the source might not be compatible with the data pipeline standards.
 - Addition of the new source may introduce overhead on the data handling capability of the pipeline.

Challenges

Infrastructure Challenges

- **Data pipeline scalability:** the ability of a data pipeline to **scale** with the increased amount of ingested data, while **keeping the cost low** is a real challenge experienced by the data pipeline developers. When the data produced by the source increases, the data pipeline **loses the ability to transfer the data** from one node to another leading to the data pipeline breakage and loss of data.
- **Increased number of nodes and connectors in upstream:** data pipelines are a chain of nodes performing activities connected through connectors that enable data transportation between the two nodes. Nodes can have more than one capability. However, for the easy detection of faults, each of the nodes should be preferably assigned a single capability. Thus, the **number of nodes and connectors increases** in the upstream in relation to the data product yielded from the pipeline. This in turn **increases the complexity** of the data pipeline and decreases ease of implementation. This lead to **inevitable delays** in adding new types of activity data, which resulted in sticking new activities into inappropriate existing types to avoid human effort, or worse, not capturing activities at all.

Challenges

Infrastructure Challenges

- **Trade-off between data pipeline complexity and robustness:** to build a robust data pipeline, we should have two essential components called *fault detection* and *mitigation strategies*. **Fault detection** identifies faults at each of the data pipeline stages and **mitigation strategies** help to reduce the impact of the fault. Including these two components **increases the complexity** of data pipelines. Moreover, it requires the data pipeline developers to anticipate the faults that can occur at each stage and define mitigation actions such that the data flow through the pipeline is not hampered. Some of the common faults can be anticipated and mitigated. However, it is **not possible to identify all possible faults and define mitigation actions for those**.
- **Repeated alarms:** sending alarms is the most common and simple mitigation action automatically taken by the data pipelines. Some faults **take time to get fixed** and during this time, the person or the team responsible for fixing the issues will get repeated alarms for the same issue. In the worst scenario, this can even lead to new alarms left unnoticed. An alternative could be sending the notification only once and then waiting for a fix for some time.

Challenges

Organizational Challenges

Organization level challenges to data pipeline management:

- **Dependency on other organizations:** data pipelines can be spread between more than one company. Therefore, **co-operation and collaboration** are required from all the participating companies to maintain a healthy data pipeline. In most cases, external companies will have very minimal knowledge of what is happening in the other part of the pipeline.
- **Lack of communication between teams:** data pipelines are meant to share data between various teams in the organization. However, **each team builds pipelines for their use case** and thus at least some initial activities are repeated in several data pipelines leading to redundant storage of data. Moreover, if any of the steps fails, the responsible person gets a notification from different teams at the same time for the same issue.

Challenges

Organizational Challenge

- **Increased responsibilities of Data Pipeline owner:** all faults in the data pipeline cannot be fixed automatically. Some faults demand either partial or complete human intervention. Therefore, a flow guardian or data pipeline owner is assigned for each of the pipelines who pays attention to data pipeline activities and takes care of the faults requiring a manual fix. Further, it is **hard to assess what code might break** due to any given change in data. With the increased use of data pipelines, the responsibilities of the flow guardian or data pipeline owner also increase.
- **DataOps-DevOps Collaboration:** DataOps is concerned with a set of practices for the development of software and management of data respectively. DataOps combines DevOps with data scientists and data engineers to support development. The challenge of **managing and delivering massive volumes of discordant data** to those who can use it to generate value is proving extremely hard. Moreover, people working with data are less interested in learning new technologies and tools while it is not a hassle for DevOps users.

Challenges

Data Quality Challenges

Improper data pipeline management may lead to **data quality challenges**:

- **Missing data files:** data files can be lost completely or partially during the transmission from one node to another. Fault detection mechanism can identify the exact point of disappearance. However, obtaining the missing files once again is a **complicated task**. Missing data files are only detected at the end of the data pipeline and in some cases, this results in poor quality data products.
- **Operational errors:** data pipelines encounter operational errors which hampers the overall functioning. Operational errors are very **common in non-automated data pipelines**. Some parts of the data pipelines cannot be completely automated. **Human errors** at these steps are the reasons for operational errors.
- **Logical changes:** data drifts and change in data distribution results in the data pipeline failures due to the incompatible logic defined in the data pipeline. Therefore, the data pipeline needs to be **monitored continuously for change in data** distributions and data shifts. Besides, data pipelines should be updated frequently by changing the business logic according to the changes in data sources.

Opportunities in Data Pipeline Management

The previous section illustrated the challenges of data pipelines when implemented in real-world. However, there are **many opportunities** the data pipeline offers through automating fault detection and mitigation.

Implementing such solutions may provide advantages.

Advantages

Solve data accessibility challenges: data generated by assorted multiple devices is collected, aggregated, and stored in central storage by data pipelines without human intervention. As a result, accessing data from devices located on the customer premises is a difficult and tedious task (assuming proper data access permissions). Most often, the devices will be distributed around the globe and teams has to prepare legal agreements complying with the rules of that specific country where the device is located for accessing data. When the data is stored after aggregation, data loses its granularity, and as a result, teams working with fine-grained data has to collect data separately. With data pipelines, teams can **access data from any point of the data pipeline** if they have necessary permissions. This **eliminates repeated collection and storage of the same data by multiple teams.**

Advantages

Save time and effort of human resources: automation of data-related activities is maximized through the implementation of data pipelines thereby **reducing the human intervention**. When a data pipeline has inbuilt monitoring capability, faults will be automatically detected and alarms will be raised. This reduces the effort of data pipeline managers and flow guardians. As the data pipeline is completely automated, **requests by teams will be answered quickly**.

On the other hand, if the workflow is not automated, the data analyst has to investigate and find out where the error has occurred and then inform the responsible person to send the data again which eventually delays the entire data analysis process. Moreover, the effort of the data analyst is also wasted while investigating the source of data error.

Advantages

Improves traceability of data workflow: data workflow consists of several interconnected processes that make it complex. Consequently, it is difficult to detect the exact point that induced error. For instance, if the end-user realizes that part of the data is missing, it might be lost during data transmission, while storing the data in a particular schema or due to unavailability of an intermediate process. Guessing all the different possibilities of data loss and has to investigating all the possibilities to recover the lost data is a time-consuming task especially when the data workflow is long and complex. After implementing data pipelines, the process of **detecting faults is automated thereby increasing traceability.**

Advantages

Supports heterogeneous data sources: data pipelines can handle multiple assorted data sources. Data ingestion is a process through which data from multiple sources are made available to the data pipeline in a uniform format. Data ingestion is the process of **streaming in massive amounts of data** in our system, from several external sources, for running analytics and other operations required by the business.

Accelerates Data life cycle activities: the data pipeline encompasses the data life cycle activities from collection to refining; from storage to analysis. Activities involved in the data pipeline are automatically executed in a predefined order and consequently, human involvement is minimized. As the **activities are triggered by themselves**, the data pipeline **accelerates the data life cycle process**. Moreover, most of the data pipeline activities are automated thereby increasing the speed of data life cycle process and productivity.

Advantages

Standardize the Data Workflow: the activities in a data workflow and their execution order are defined by a data pipeline providing organizations with an **overall view of the entire data management process**. Thus, it enables better **communication and collaboration** between various teams in the organization. Further, data pipelines reduce the burden on IT teams thereby reducing support and maintenance costs as well. Standardization through data pipelines also enables **monitoring** for known issues and quick troubleshooting of common problems.

Advantages

Improved Data Analytics and Machine Learning Models: organizations can make use of carefully designed data pipelines for the **preparation of high quality, well-structured, and reliable datasets** for analytics and also for developing machine learning as well as deep learning models. Besides, data pipelines automate the movement, aggregation, transformation, and storage of data from multiple assorted sources. Machine learning models are highly sensitive to the input training data. Therefore, **quality of training data is very important**. Data pipelines are traceable since the stages are predefined yielding better quality data for the models. Moreover, data pipelines ensure a smooth flow of data unless it fails in one of the steps.

Advantages

Data Sharing between teams: Data pipelines enable easy data sharing between teams. For instance, data cleaning is an activity performed by all the teams before feeding the data to ML/DL models. Therefore, there is a possibility of the same data going through the same sequence of steps within different teams of the same organization. Further, data storage also is wasted in such cases due to redundant storage. With the implementation of data pipelines, the **teams can request data from a particular step** in some other data pipeline and can **process the subsequent steps in their data pipeline**. However, the data pipeline should be able to serve the requests in such cases.

Advantages

Critical Element for DataOps: DataOps is a process-oriented approach on data that spans from the origin of ideas to the creation of graphs and charts which creates value. It merges two data pipelines namely **value pipeline and innovation pipeline**. Value pipeline is a series of stages that produce value or insights and innovation pipeline is the process through which new analytic ideas are introduced into the value pipeline. Therefore, **data pipelines are critical elements for DataOps together with Agile data science, continuous integration, and continuous delivery practices.**



Thank you for your attention!

Cervini Stella
Ferati Delor
Sabino Giuseppe