# Data analytics

## (Modern solutions)

Jana Kostičová

# Back to Business Intelligence

**BI** = providing insights to support business decision-making

- It helps to optimize operations - the end product is separate from the analytics
- Used internally, customers / users don't see or interact with them
- Mostly a secondary function over data collected primarily for operational purposes

- BI system downtime:
  - Some issues can appear (delayed decision-making, missed opportunities for optimization, and less efficient operations)
  - The company usually continues its core operations because BI is not used for day-to-day business

# Emergence of modern data analytics (~2000)

Most prominent triggers for changes

1. Proliferation of smartphones, sensors, and Web 2.0/social platforms
    - New data sources: sensor measurements, IoT data, location data, social media data, user interaction data, location data, …
    - New data characteristics: very large data volumes (“big data”), continuous data streams, high data heterogeneity, and new data formats
2. New kinds of use cases for data analytics
    - Data-driven products / services created new uses for that data → real-time delivery, predictive / prescriptive analytics
3. Advances in technology
    - Machine learning. Natural language processing, Artificial intelligence

# Data-driven products and services

Big tech companies: data are valuable not just for decision-making but also for building new revenue streams.

Data analytics used for BI **+ as a part of products / services**

2000

**Amazon:** recommendation system, **Google:** targeted advertising

**Facebook, LinkedIn**: selling insights about their users (through ads)

**Google Maps:** real-time traffic / route suggestions based on mobile phone locations

2010

**Netflix, Uber, Spotify:** machine learning used to offer personalized services, “data monetization”

**Tesla:** Autopilot feature relies on data analytics gathered from Tesla’s fleet of vehicles

**Airbnb:** predictive algorithms used to help hosts optimize pricing and availability



# Examples

## Traditional DWH - illustrative telecom operator

- assuming 2 million active subscribers
- ~60 million CDR + SMS records/day
- **~50–110 TB over 5 years history** (500 B–1 KB per record)
- Highly structured data
- Historical / batch analytical workloads

versus:

## Modern data analytics - Uber

- **~350 PB** (2026)
- Heterogeneous sources and formats
- Evolving schemas
- Diverse workloads, including real-time analytics

# Extended requirements of data analytics

Original requirements still hold:

- Data analytics over integrated data
- Data analytics over potentially large data volumes
- Data analytics over historical data
- Data designed for analytical queries
- Efficient execution of data analytics

(New) extended requirements:

- Data analytics over **even larger (and increasing)** data volumes
- Data analytics over (a combination of) stored data and **data streams**
- Data analytics over (a combination of) structured, semi-structured and **unstructured** data
- Data analytics delivered in **“real-time”**
- More  complex **predictive** & **prescriptive** data analytics

- Cost-effective data analytics

# Data processing models vs latency requirements

**Data processing models (how data are processed)**

- **Batch processing** -> process accumulated data periodically
  - Traditional batch: large / infrequent batches
  - Micro-batch: small / frequent batches
- **Stream processing** -> process continuously arriving data

End-to-end vs processing latency!

**End-to-end latency requirements (how quickly results are expected)**

- **Real-time (RT)** - very low latency, often microseconds to milliseconds
- **Near-real-time (NRT)** - some delay is acceptable, often seconds to minutes
- **Higher latency** - hours, days, etc.

| | **Real-time** | **Near-real-time** | **Higher latency** |
|---|---|---|---|
| **Stream processing** | Possible, common | Possible, common | Possible, but uncommon |
| **Batch processing** | Hardly possible | Possible, common (mostly mico-batch) | Possible, common (mostly traditional batch) |

# Limitations of traditional methods (1)

**Horizontal scaling (scaling out):** Adding more nodes to a cluster to increase capacity.

**Vertical scaling (scaling up):** Increasing the resources of an individual node, such as adding more CPU cores, memory, or storage.

## Scalability of traditional RDBMSs

- Difficult to scale horizontally (ACID, data distribution complexity, centralized management, ..)
- Vertical scaling is possible, but more expensive and less flexible
- Coupled storage & computation

Relational MPP DWHs (such as Teradata) scaled horizontally, but

- Nodes had to use supported HW
- Expansion required additional HW integration and configuration

## Relational model

- Designed primarily for structured, tabular data
- Schema-on-write: schemas are defined before data is loaded into the DWH

-> Less natural for heterogeneous and semi-structured data, or data with evolving / unclear structure

# Limitations of traditional methods (2)

## Batch processing

- DWH loads were triggered daily (weekly, monthly, ..)
- Time-consuming ETLs and analytical queries
- Latency in the analysis ( > 24 hrs)

⇨ Could be difficult to implement streaming / real time pipelines

## Limited data analytics capabilities

- Mostly descriptive and diagnostic analytics; data mining tools also enabled some predictive analytics
- Traditional relational analytics was primarily designed for structured, tabular data
  - Less suited to analyzing semi-structured and unstructured data, or their combinations
  - No access to raw data (useful patterns can be lost during ETL)

# Modern data analytics solutions

## Technologies

- New distributed storage systems & computation frameworks
- New DB types, new file formats

} Can handle large data volumes & heterogenous data more efficiently

- Stream & RT / NRT processing technologies
- Advanced analytics techniques

## Architecture

- Core principles of traditional architectures still relevant
- New concepts emerged:
  - Lambda / Kappa architecture
  - Data lake, Data lakehouse, ELT
  - Data mesh

## Commercial cloud environments

- On-demand access to scalable storage and computing
- Analytics affordable also for smaller businesses
- Many of new technologies are intended for specific cloud environment



*Difficult to choose the proper technology stack*

Many of new technologies have applications beyond data warehousing and data analytics.

# New technologies

# New storage systems & computation frameworks

- Can store heterogenous data, allow for flexible horizontal scalability (add storage capacity)
- **Distributed file systems (DFS)**
  - *HDFS (inspired by Google File System, 2003)*
  - Hierarchical structure similar to file system
- **Distributed cloud object storage systems**
  - *GCS, Amazon S3, ADLS, MinIO, …*
  - Flat namespace of objects identified by keys

+ **Computation frameworks**
*MapReduce (inspired by Google MapReduce, 2004), Apache Spark*

- Allow for parallel processing over distributed data
- Flexible horizontal scalability (add processing capacity)
- Decoupled storage & compute

**Hardware:** Mostly standard / commodity hardware is sufficient

- HDFS – explicit architectural principle
- Proprietary object storage systems – conceptually similar, but underlying HW is hidden

# Data distribution

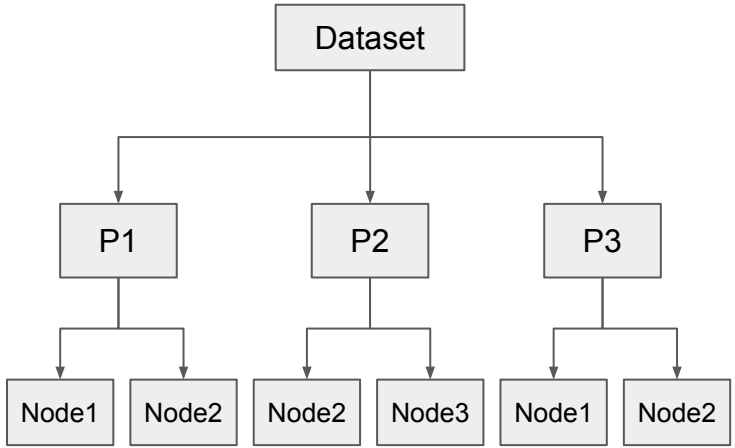
The process of distributing data across multiple nodes typically involves two key concepts:

1. **Replication** - storing separate copies of data at two or more nodes

   → fault tolerance (avoiding single point of failure), availability

2. **Partitioning (sharding)** - data are divided into smaller parts and then store on separate nodes, i.e., particular partitions (shards) do not share data

   → horizontal scalability, parallel processing




Replication and partitioning are typically combined.
Data distribution is largely automatic (only the rules / parameters need to be configured).

# Main technological challenges

- **Replication:** synchronizing replicas / maintaining consistency
- **Partitioning:** distributing data evenly and avoiding hotspots
- **Distributed processing:** executing queries/computations over partitioned data efficiently
- **Fault tolerance:** continuing operation despite node failures
- **Rebalancing:** redistributing data when the cluster changes

# New database types

- **NoSQL databases:** *Apache Cassandra, MongoDB, Amazon DynamoDB, Google Bigtable*
  - Non-relational data models: key-value pairs, key-document pairs, column families, graphs
  - Many of them are distributed and support parallel processing
- **Columnar databases:** *Amazon Redshift, Google BigQuery*
  - Use the relational model, but store data by columns rather than rows
  - Efficient execution of analytical queries

\+ NewSQL databases
  - Primarily intended for OLTP (transactional processing), sometimes HTAP (Hybrid Transactional / Analytical Processing)

# New file formats

**Column-oriented formats:** *Apache Parquet, ORC (Optimized row columnar)*

- Binary formats, store data by columns
- Efficient processing of analytical queries that access only a subset of columns
- Better compression because values in the same column tend to be similar
- Often used for storing processed analytical data

**Serialization formats:** *Apache Avro*

- Binary format for structured data
- Efficient serialization / deserialization
- Used for data exchange and streaming

---

- All three are schema-based / self-describing format (schema is stored in the file metadata or schema registry)
- All three are used in the data lakehouse architecture, together with open table formats (see later)
- All three originated in industry as open-source formats for the Hadoop & big data ecosystem
    - Parquet - Twitter & Cloudera, ORC - Hortonworks, Avro - Yahoo!

# Technologies for processing data streams

Goal: process a continuously arriving stream of events

*Apache Kafka* - message broker / persistent event log

(originally developed at LinkedIn in 2011 for processing log data with low latency)

- Messages are not deleted after delivery (retention policy is defined)
- More consumers can read the same event stream, each at different offset
- Support for parallel processing

*Apache Flink, Apache Storm, Spark Structured Streaming* - stream processing

- Stream processing frameworks
- Usually consume stream from a message broker
- May feed another message broker
- Support for parallel processing

# Techniques for RT / NRT processing

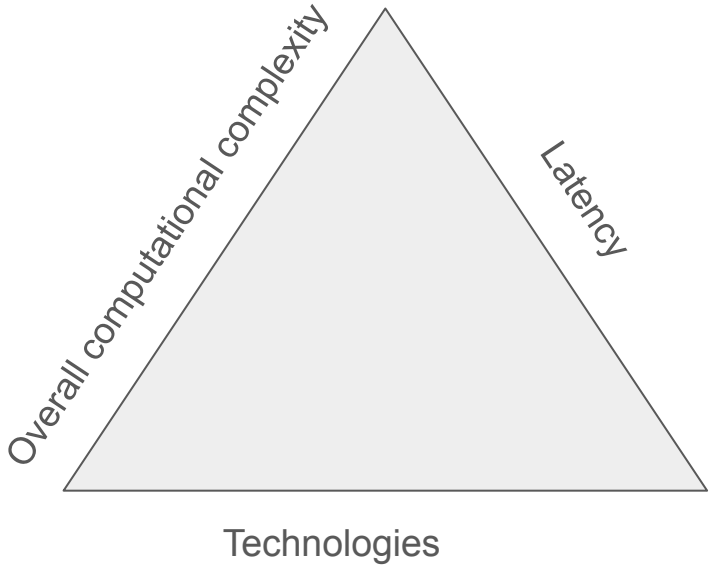
Goal: reduce end-to-end latency

Low-latency data ingestion:

- CDC (Change Data Capture) - low-latency extraction of changes from operational databases
- Event streaming (e.g., using *Kafka*)
- Efficient serialization / deserialization (e.g., using *Apache Avro*)

Low-latency processing and analyses:

- Parallel / distributed processing
- In-memory processing & precomputations - *Redis, Memcached*
- Columnar storage (e.g., using *Apache Parquet*)
- Fast serialization / deserialization




Higher overall computational complexity generally results in higher latency. This can be (partially) compensated by using more sophisticated techniques

# Advanced analytics techniques

Machine learning (ML), Deep learning (DL)

- ML / DL models integrated with DWHs and data lakes / lakehouses
- Enhance the capabilities of all types of data analytics
- Applications:
  - Image / video recognition and analysis
  - Audio / speech recognition and analysis
  - Pattern recognition and anomaly detection
  - Recommendation and personalization
- Data lakes / lakehouses provide access to large volumes of raw and heterogeneous data

Natural Language Processing (NLP)

- Analysis and processing of text data
- Customer reviews, social media posts, documents, etc.
- Text classification, sentiment analysis, information extraction, etc.

# New architectures

# Data lake

- Introduced in 2010 by J. Dixon [4]
- Centralized repository of data in its raw format
  - All data (including historical)
  - Structured / semi-structured / unstructured data
    *(CSV, JSON, XML, Parquet, images, videos, texts, ..)*
  - Support for both batch and streaming data ingestion
  - Schema on-read: schema is defined when the data is queried, data discovery driven by metadata
- Motivation
  - Support data science and machine learning
  - Flexible access to diverse raw data & low storage costs
- Typical implementation
  - Distributed storage system + metadata catalog
  - Data accessed via a query engine or a computation framework
- Criticism
  - Poorly maintained data lakes turn to data swamps - strict metadata management is important

**Example solutions**

storage + metadata catalog

*AWS: S3 + Glue Data Catalog*

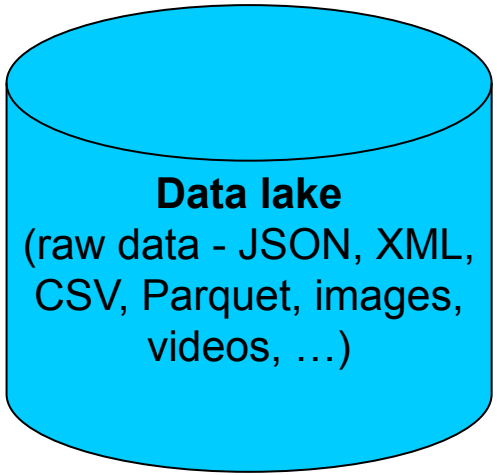
*Microsoft Azure: ADLS + Purview*

*Open-source: HDFS + Hive Metastore*

# Metadata catalog

= a searchable registry of registered datasets in a data lake

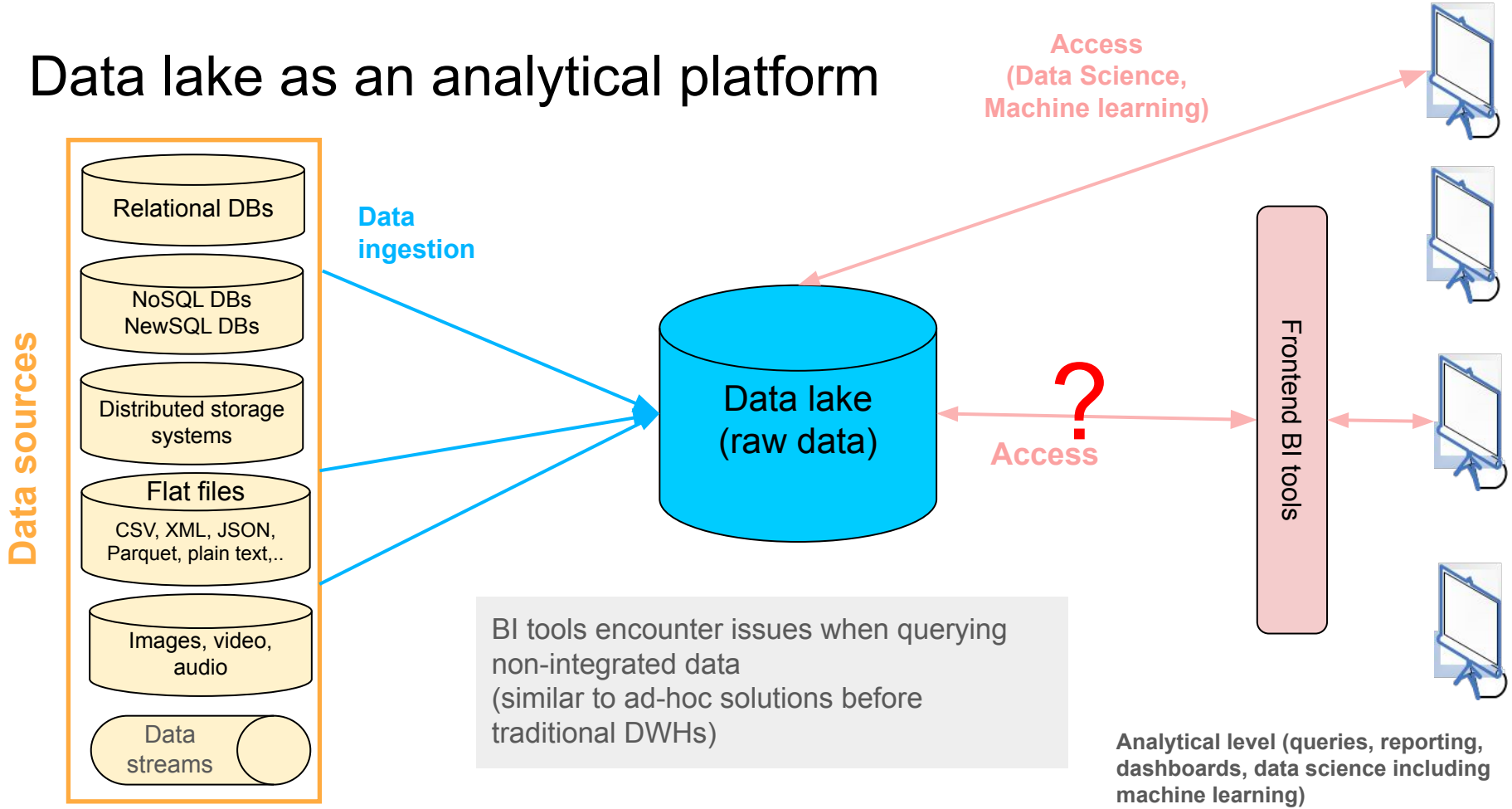
- Created manually or automatically (automatic discovery, processing pipelines)
- Consulted by query engines (schema on-read)
- Not used to validate data writes - data may get inconsistent → “data swamp” scenario



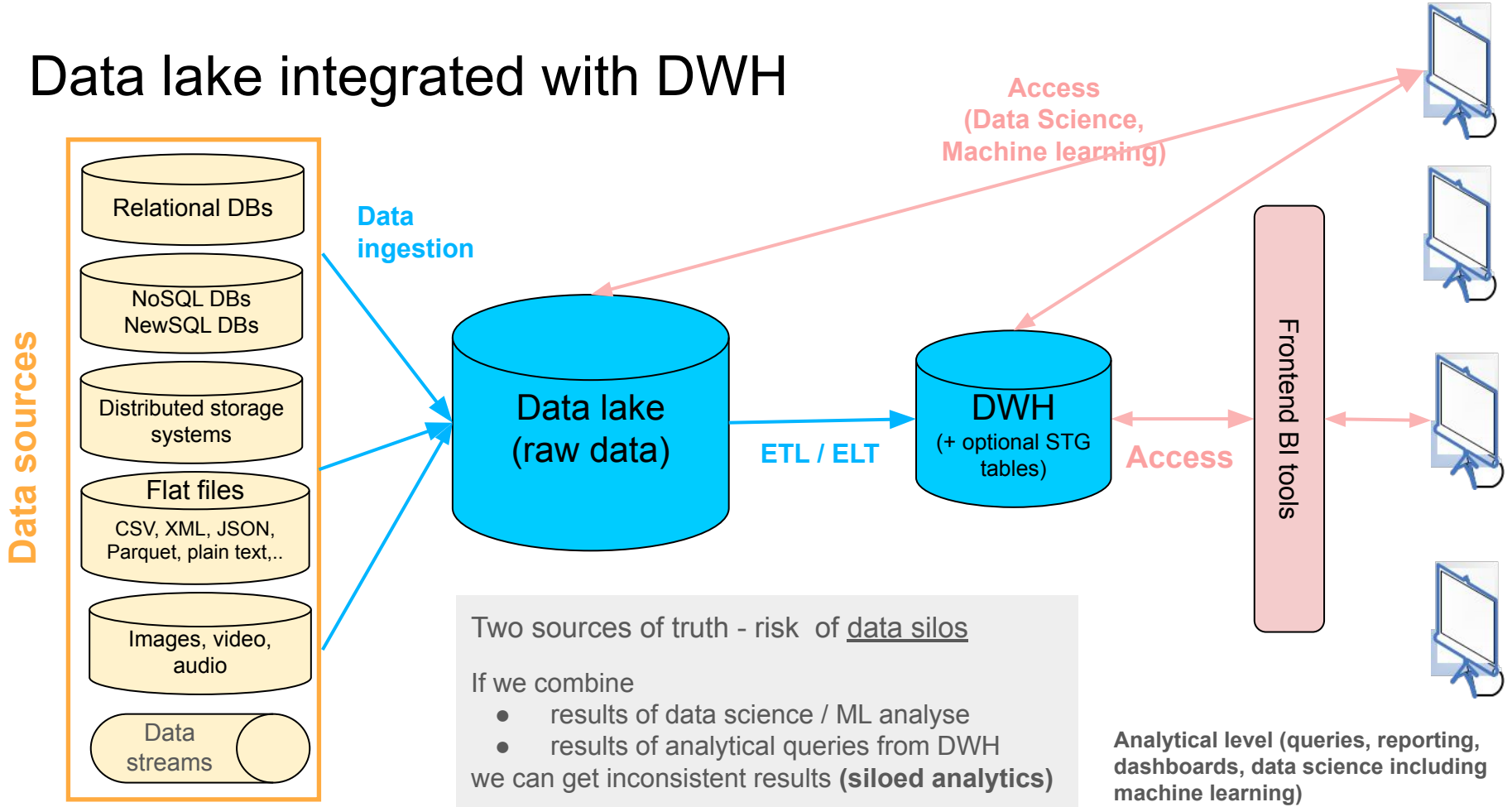

**Metadata catalog**

Table name:
- Schema
- Location of corresponding files

# Data lake as an analytical platform



# Data lake integrated with DWH




Two sources of truth - risk of data silos

If we combine
- results of data science / ML analyse
- results of analytical queries from DWH

we can get inconsistent results **(siloed analytics)**

**Analytical level (queries, reporting, dashboards, data science including machine learning)**

# ETL vs ELT

## ETL - Extract, Transform, Load

- Used both in traditional and modern analytical solutions
- Data are transformed before being loaded into the target environment

## ELT - Extract, Load, Transform

- Used in modern analytical solutions (together with ETL)
- Data are transformed after being loaded into the target environment
  - Target contains both the source data and the transformed data (although source data can be deleted after transformation, depending on the use case)
- Advantages
  - Transformations use target platform's computation resources, no need for a separate ETL server
  - Easy to build multiple transformation pipelines from the same source data, without re-extracting it from the source system
  - Easy to change and re-run existing transformations

# Data lake with zone architecture

The concept of data lake has evolved in time

1. Narrow definition: A data lake stores raw (unprocessed) data [5]
2. Extended definition: A data lake can also contain multiple **zones**, with data transformed and refined between zones [6]

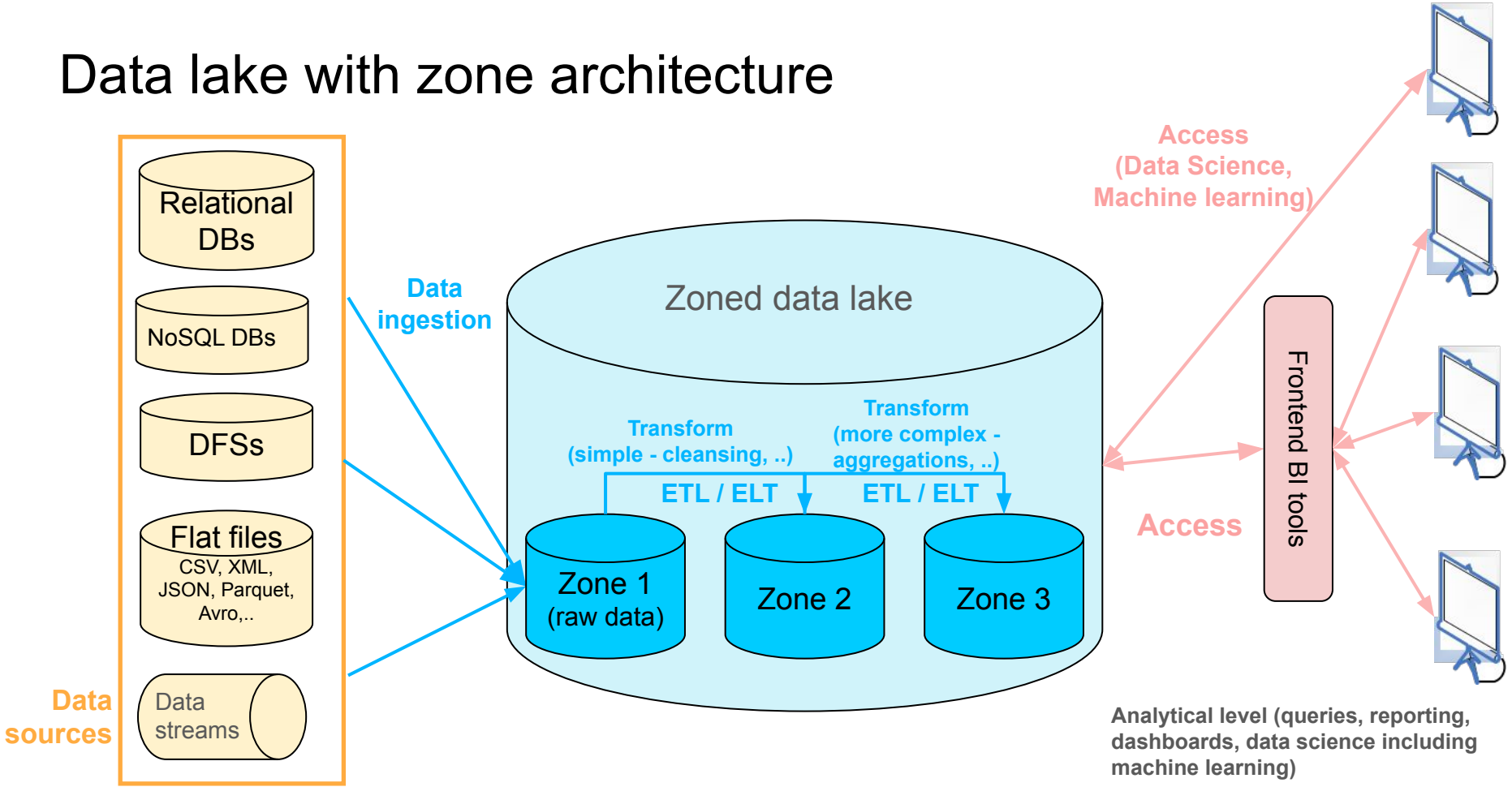
**Zone architecture** (documented in research by 2015–2017)

- Data are organized into zones according to their processing level and intended purpose
- Different zones can be accessed directly, but each zone typically serves different use cases
- No DWH-like table management capabilities
- **Risk of data swamps:** There is still a risk of data swamps
- **Risk of data silos:** Architectural separation is often reduced, but logical silos can still arise between or within zones

Zone architecture can be seen as incorporating DWH-like data processing into a data lake platform.

Example: Databricks / Medallion architecture (bronze, silver, and gold zone)

# Data lake with zone architecture



# Data lakehouse

→ data lake + data warehouse, combines data lake with DWH-like table management

The concept was introduced in 2020 by Databricks.

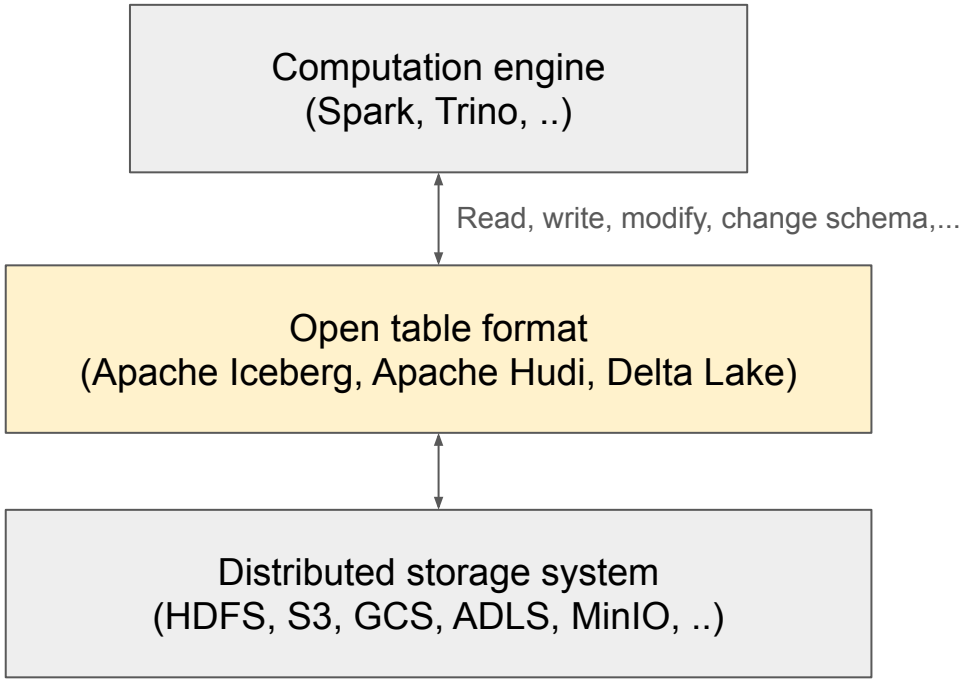
- Table management layer (essential) - a defining characteristic of a data lakehouse
- Zone architecture (optional, but widely used)

**Table management layer**

- ACID transactions
  - To maintain a consistent table state
  - To prevent e.g. writing incomplete results, concurrent jobs overwriting each other's changes, …
- Schema evolution
  - To allow changes to schemas of existing tables
- Snapshot-based time travel
  - To query historical table state and reproducibility / recovery
  - Alternative to data historization in traditional DWHs
- Motivation
  - To prevent "data swamp" scenarios
  - To get familiar relational semantics of tables with the scalability of distributed storage

Open table formats: *Delta Lake, Apache Iceberg, Apache Hudi*

# Data lakehouse architecture




**Open table format** simplified:
It contains table definitions:

- schema +
- what data files comprise (a specific snapshot of) the table

**Open table format ≠ storage system !**

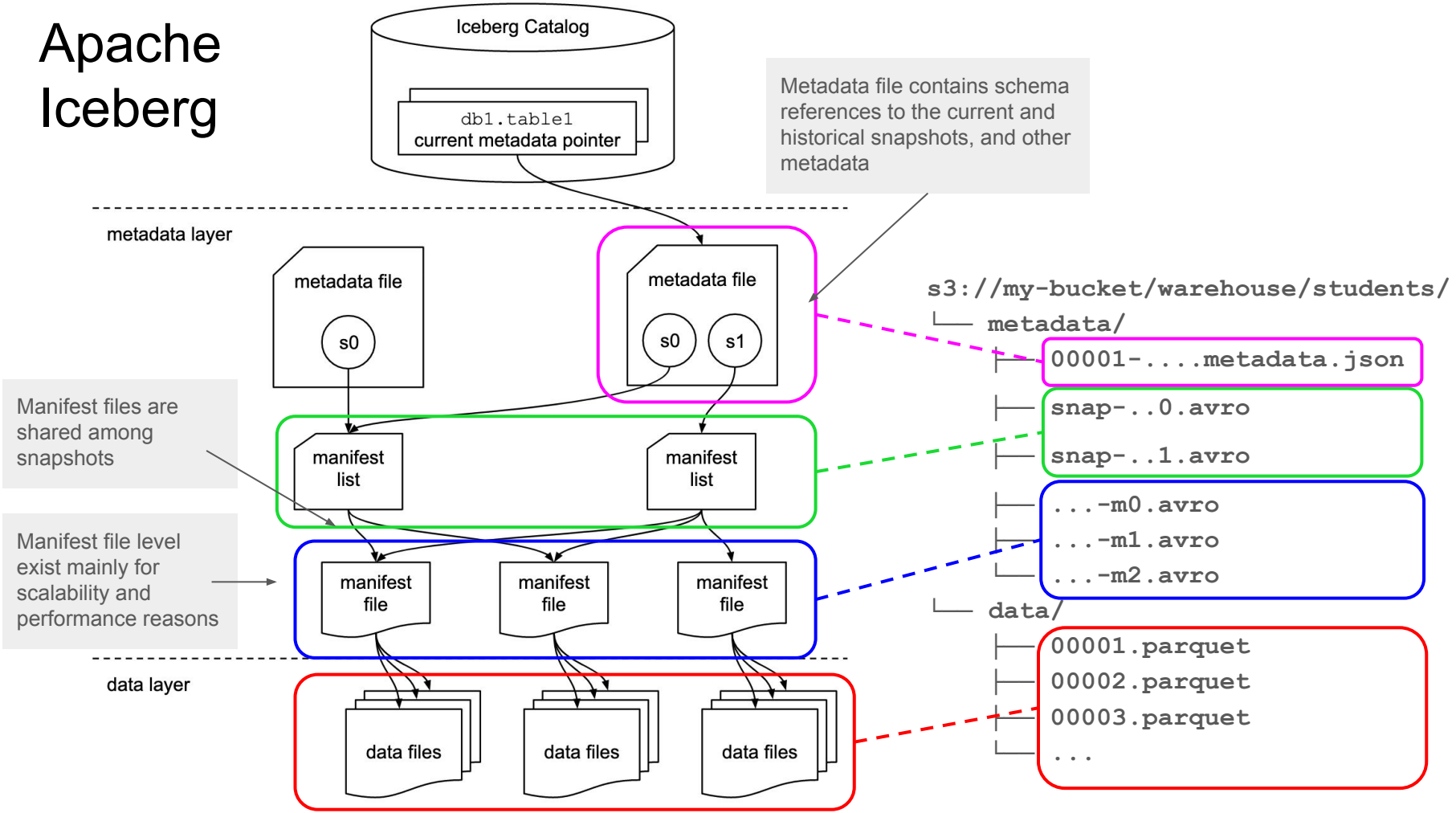
**Open table format ≠ computation engine !**

On write:

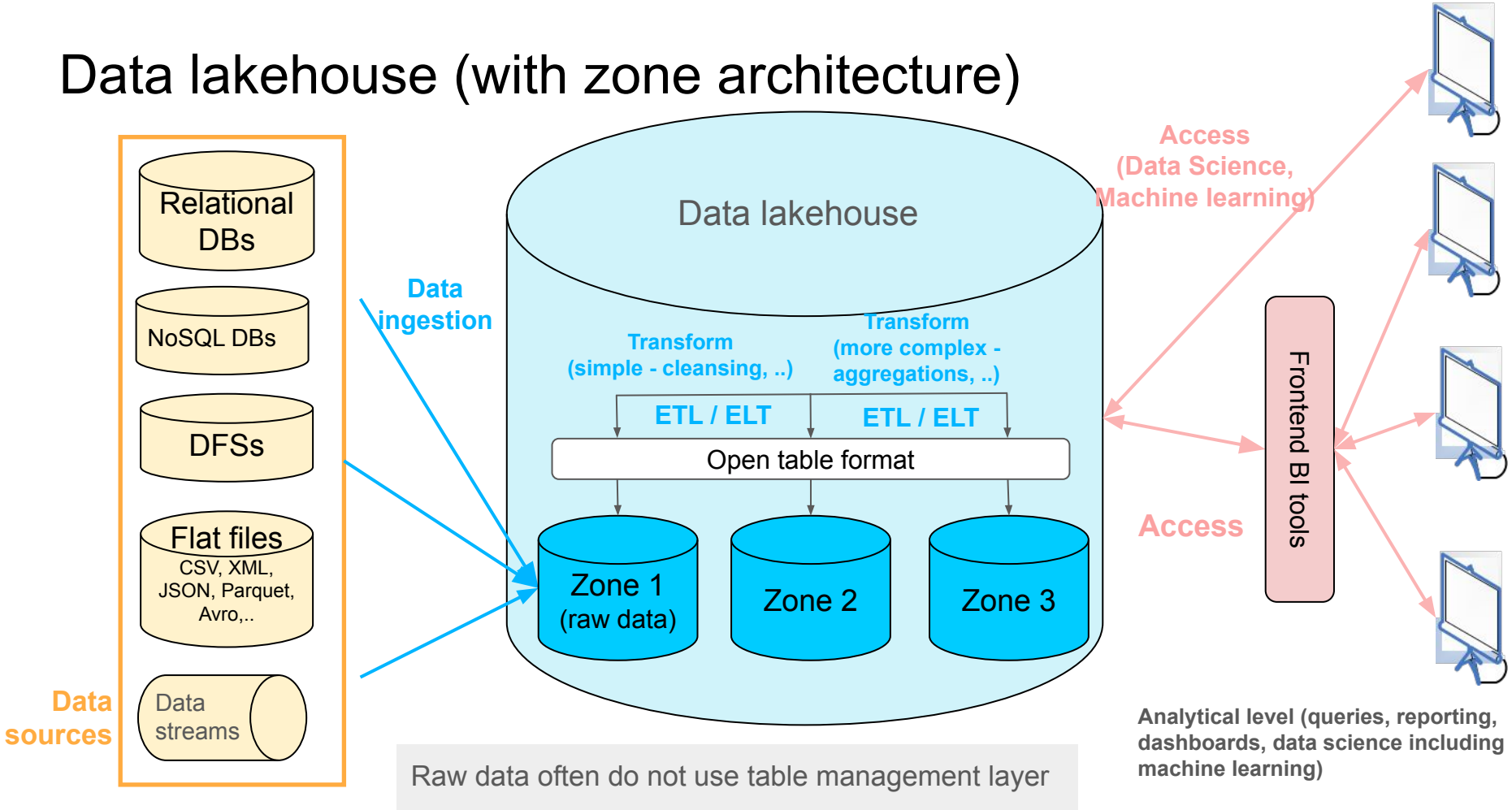
- Schema compatibility check, including NOT NULL check
- No native support for referential integrity

→ Writes can reject data that violate enforced schema constraints

# Apache Iceberg



# Data lakehouse (with zone architecture)

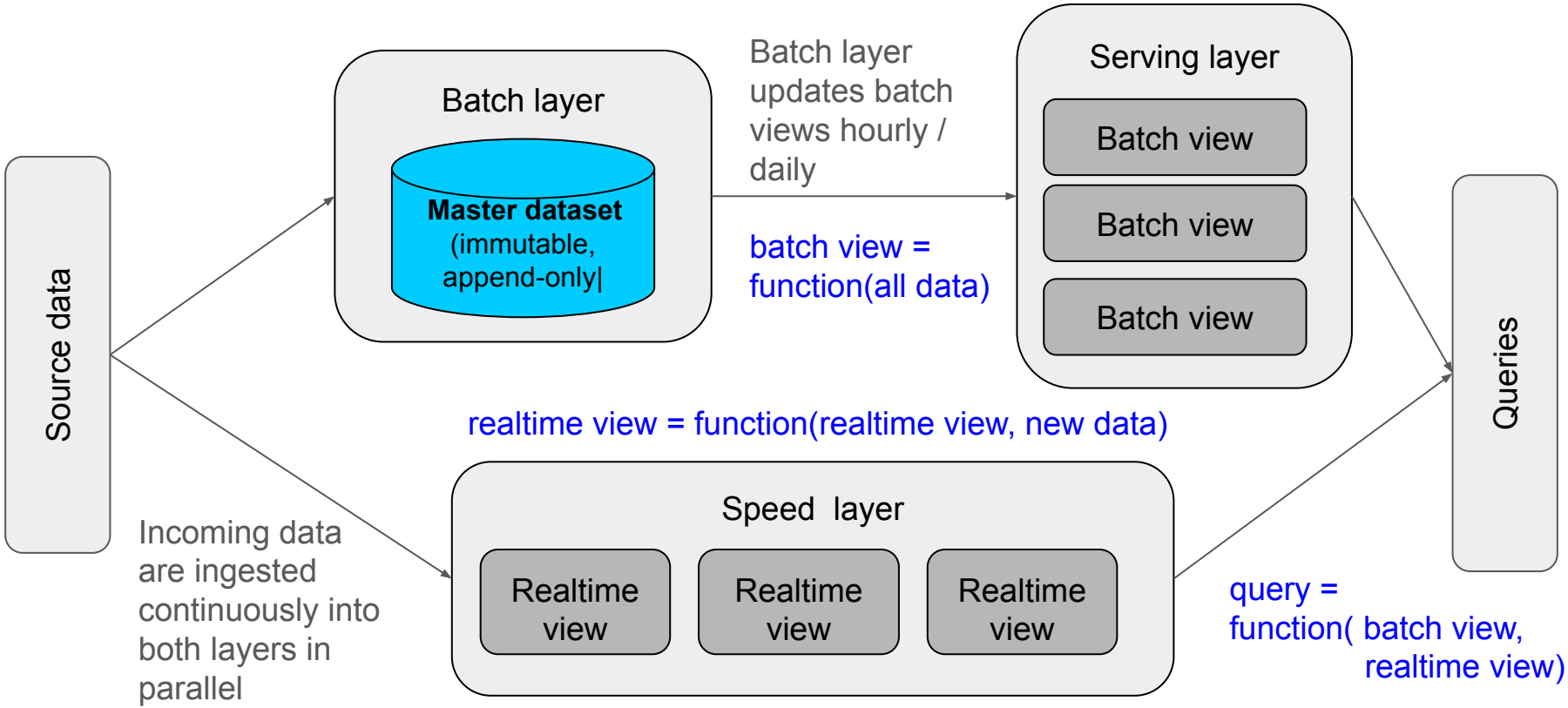


# Lambda architecture

- Introduced by Nathan Marz in 2011
- Combines batch and stream processing
- 3 layers:
  - Batch processing: batch layer + serving layer
  - Low-latency processing: speed layer

# Lambda architecture

- Master dataset is the authoritative collection of historical data
- Batch views are authoritative results computed from the master dataset
- Speed layer provides fresher (unauthoritative) results through incremental processing until the batch views catch up



# Batch vs speed views

Batch and speed layers may compute the same logical view:

- Batch view contains results computed from the master dataset up to the latest processed batch processing cutoff
- Real-time view contains results for newer data not yet reflected in the batch view
- Query results combine the two views

*Example:*

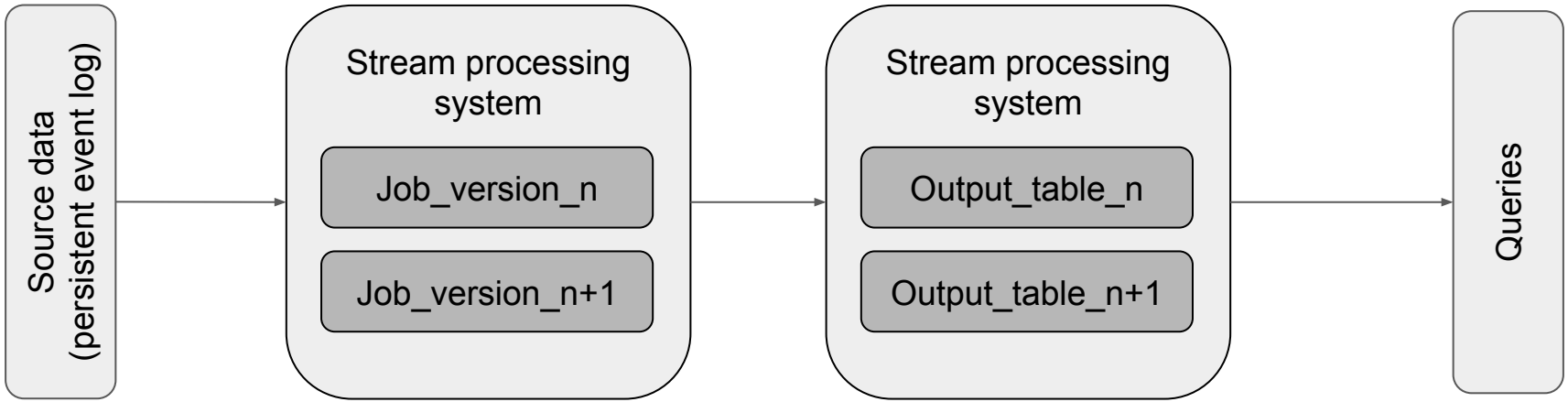
1. *The batch view contains transactions up to 10:00*
2. *The speed layer computes results for transactions from 10:00 to 10:05*
3. *A query combines both results*
4. *At the next batch update, the batch view incorporates those newer transactions, and the query must avoid counting them twice*

# Lambda architecture - criticism

- Inherent complexity - a need to maintain two separate processing paths for the same data
  - Duplicate implementation of business logic
  - High operational complexity
  - Consistency problems
  - Full recomputation of batch views can be expensive

# Kappa architecture

- Introduced by Jay Kreps (LinkedIn, Confluent)  in 2014
- More simple - only one stream processing pipeline
  - Similar to Lambda's speed layer, extended to handle historical reprocessing by replaying events
- If processing logic changes, historical events can be replayed and re-processed
- Criticism:
  - Stream processing can be inefficient for some types of computation (a batch engine could do a better job)



# Summary

# Traditional vs modern data warehouses (1)

- Core traditional concepts still in use
  - Dimensional model, 2-layer / 3-layer architectures
- New architectural concepts
  - Data lake, zone architecture, data lakehouse, lambda / kappa architecture
  - Still evolving
  - Many companies adopt hybrid or custom architectures tailored to their specific needs
- A shift in the use of data analytics in business
  - BI + part of products / services / automated decision processes
- A significant shift in technologies (also when using traditional architectures)
  - Data storage
    - RDBMSs replaced or complemented by new DB types, new storage systems and computation frameworks (many of them distributed)
  - Data processing
    - ETL tools able to read from and write to new storage systems
    - ETL replaced or complemented by parallel computation mechanisms (MapReduce, Spark, …)
    - ELT
  - Integration of new data analytics methods

# Traditional vs modern data warehouses (2)

- Batch vs real-time data analytics
  - Often separate real-time data pipelines (Lambda-like or hybrid approaches)
  - Some data pipelines cannot be accelerated inherently
- More affordable data analytics
  - Cloud solutions
  - More lightweight data analytics tools, embedded data analytics

- Adaptation of traditional DWH ?
  - Migrate to new storage systems & computation frameworks, migrate to new architectures, migrate to cloud, design RT / NRT pipelines
    → Might get expensive
    → Might get complex for poorly documented DWHs
  - In some cases building new solution from scratch is more efficient

Optimal solution strongly depends on actual requirements and the properties of the current data analytics solution

# References


[1] How Uber Scaled Data Replication to Move Petabytes Every Day, uber.com, 29.1.2026
[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung (Google): The Google file system, 2003.
[3] Jeffrey Dean, Sanjay Ghemawat (Google): MapReduce: Simplified Data Processing on Large Clusters, 2004.
[4] James Dixon: Pentaho, Hadoop, and data lakes., 2010
[5] Anne Laurent, Dominique Laurent, Cédrine Madera: Introduction to Data Lakes: Definitions and Discussions, 2020.
[6] Pegdwendé Sawadogo and Jérôme Darmont: On data lake architectures and metadata management, 2020.
[7] What is a data lake?, aws.amazon.com
[8] Michael Armbrust, Ali Ghodsi, Reynold Xin, Matei Zaharia (Databricks): Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics, 2020
[9] Warehousing and Advanced Analytics, 2021, CC BY 3.0.


Credit:

- Vectorportal.com, CC BY
- Michael Armbrust Ali Ghodsi, Reynold Xin, Matei Zaharia: Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics, 2021, CC BY 3.0.