The big picture

Goals

- Recall foundational concepts related to the contemporary big data landscape.
- Define the core challenges with building a big data architecture.
- Explain how the core components of a Lakehouse relate to the Inmon Architecture.
- Explain how Delta Lake can be used to build a Lakehouse.
- Describe the core components of Delta Lake.

The big data landscape

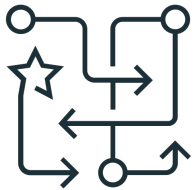
databricks

The big data problem

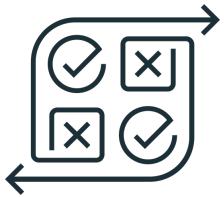
Volume

Velocity

Variety



Veracity



Value

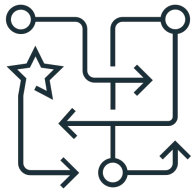
Brainstorm

- How do the 5 V's of big data tie back into your Moovio dataset?

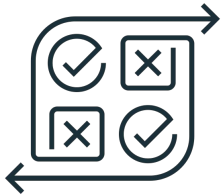
Volume

Velocity

Variety



Veracity



Value

Service level agreements (SLAs)

Data freshness

Query speed

Data reliability

Ease of use

At Moovio, your SLA includes:

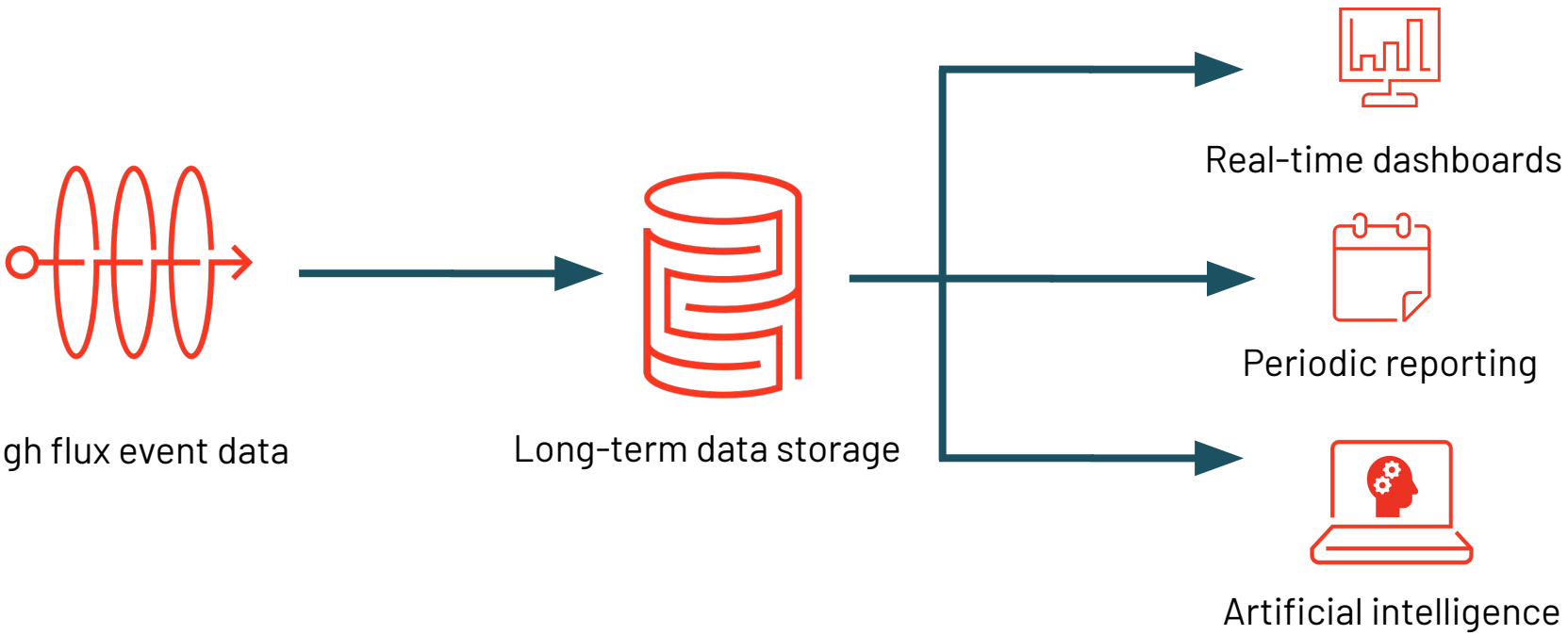
Data freshness

Query speed

Data reliability

Ease of use

Big data needs



A single source of truth

Enterprise Decision Support System (EDSS)

- Information system used by an organization to help decision-making
- Can be built:
  - On-premises or in the cloud
  - Using data lake, data warehouse or data lakehouse technology

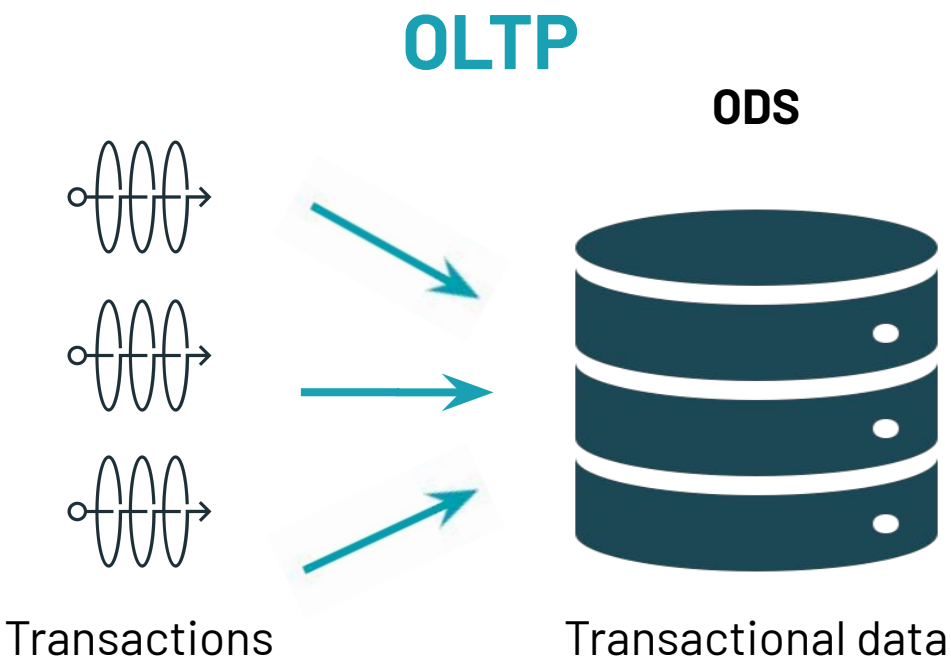
Analytical and operational layers

An **EDSS** performs **Online Analytical Processing (OLAP)**

An **ODS** performs **Online Transaction Processing (OLTP)**

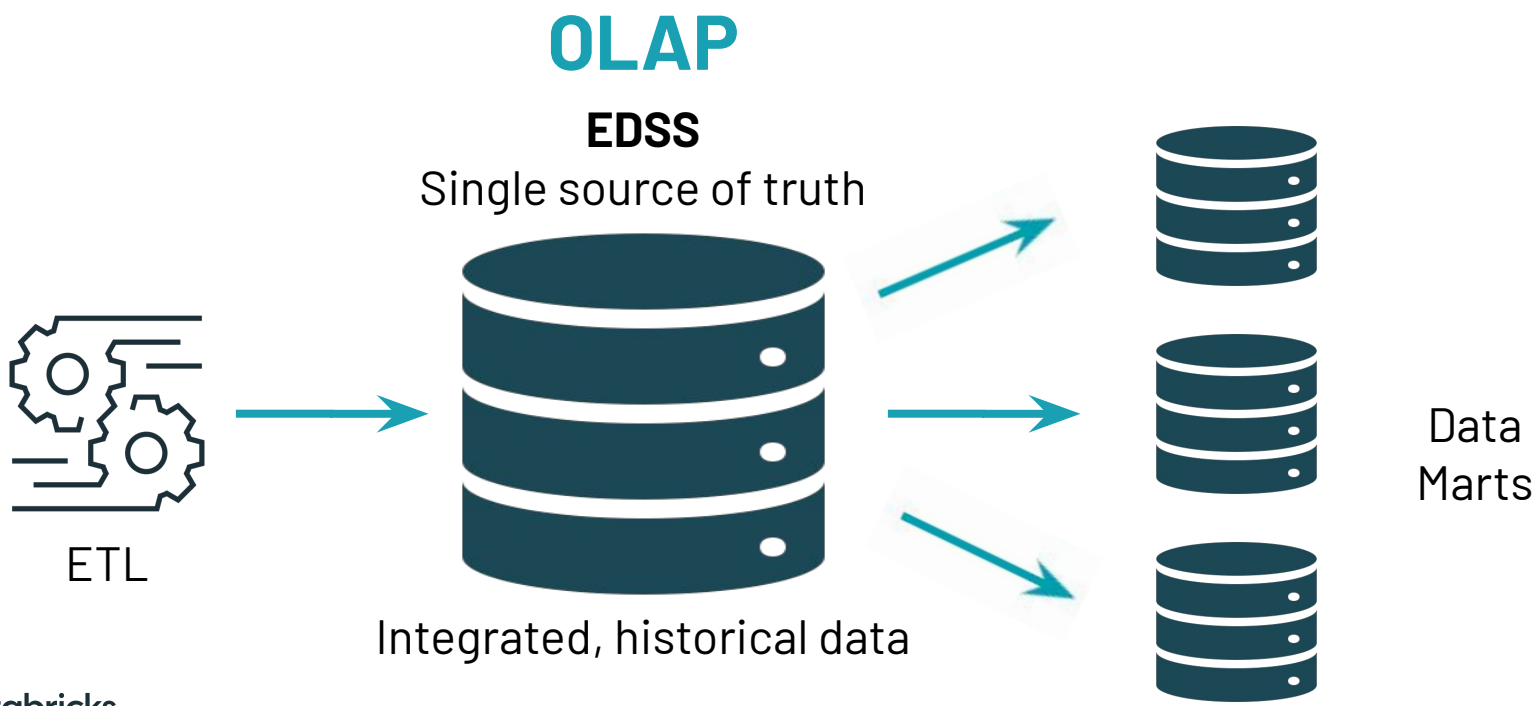
ODS and OLTP

Operational Data Store and Online Transaction Processing



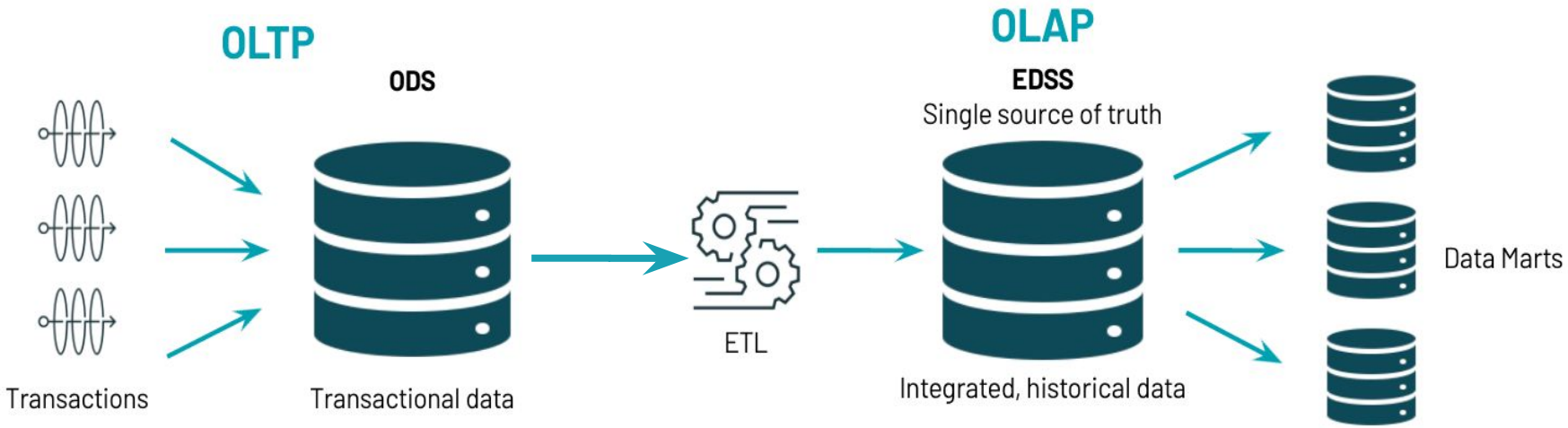
EDSS and OLAP

The Enterprise Decision Support System and Online Analytical Processing



Complete data system

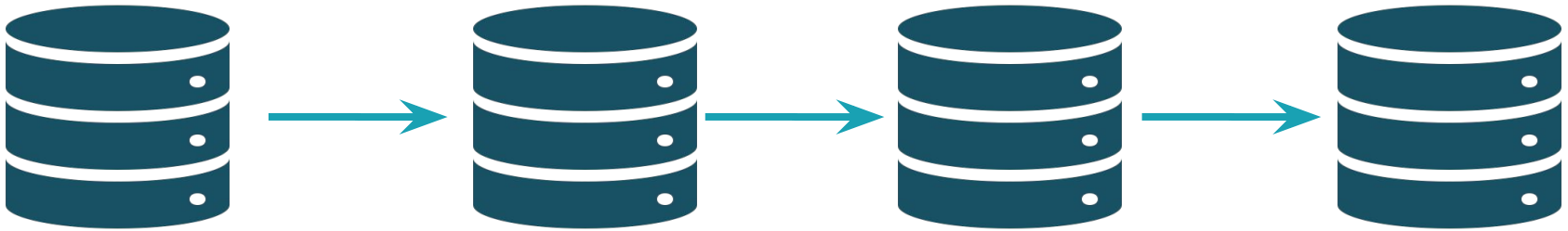
- An ETL process pulls data from the ODS to be loaded into the EDSS



Brainstorm

- Relate this process and its components (EDSS and ODS) to Moovio's data infrastructure.

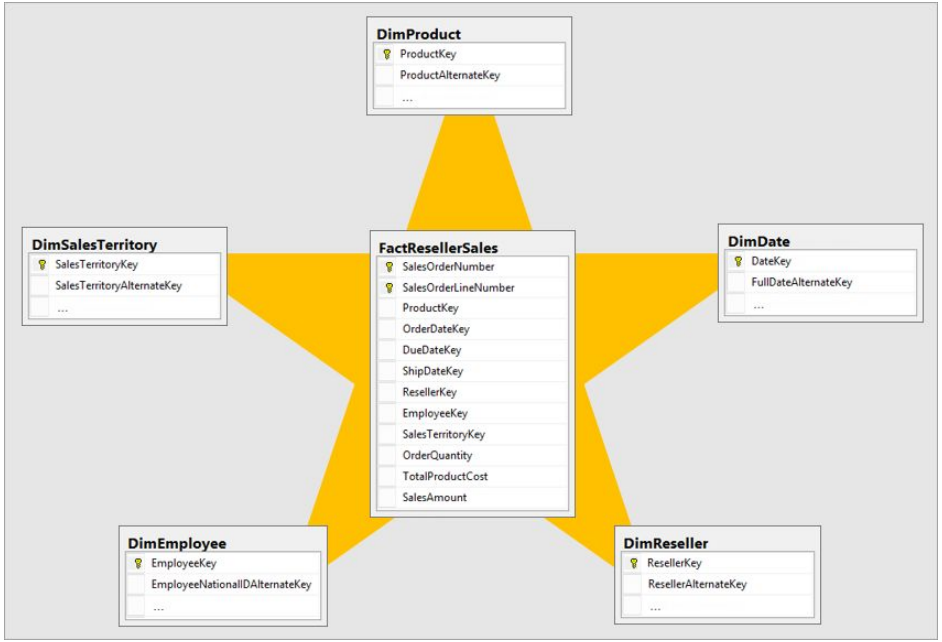
Levels of data



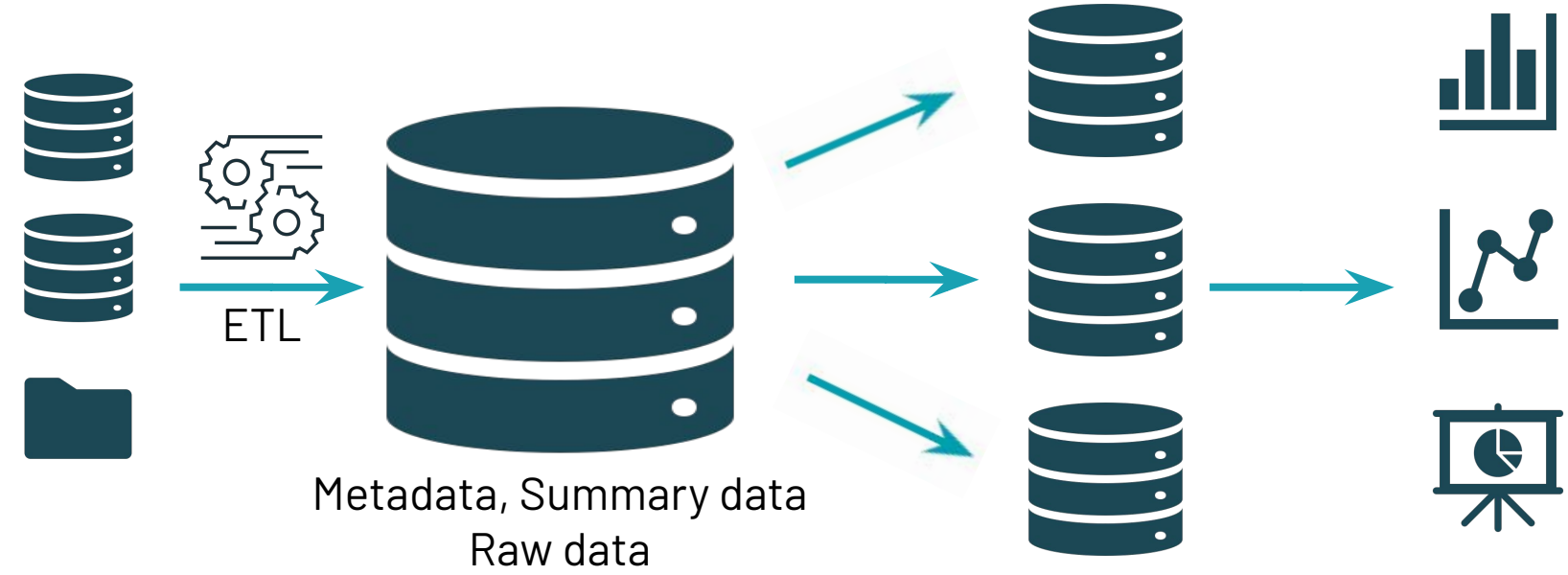
Fact and dimension tables

Dimensional modeling

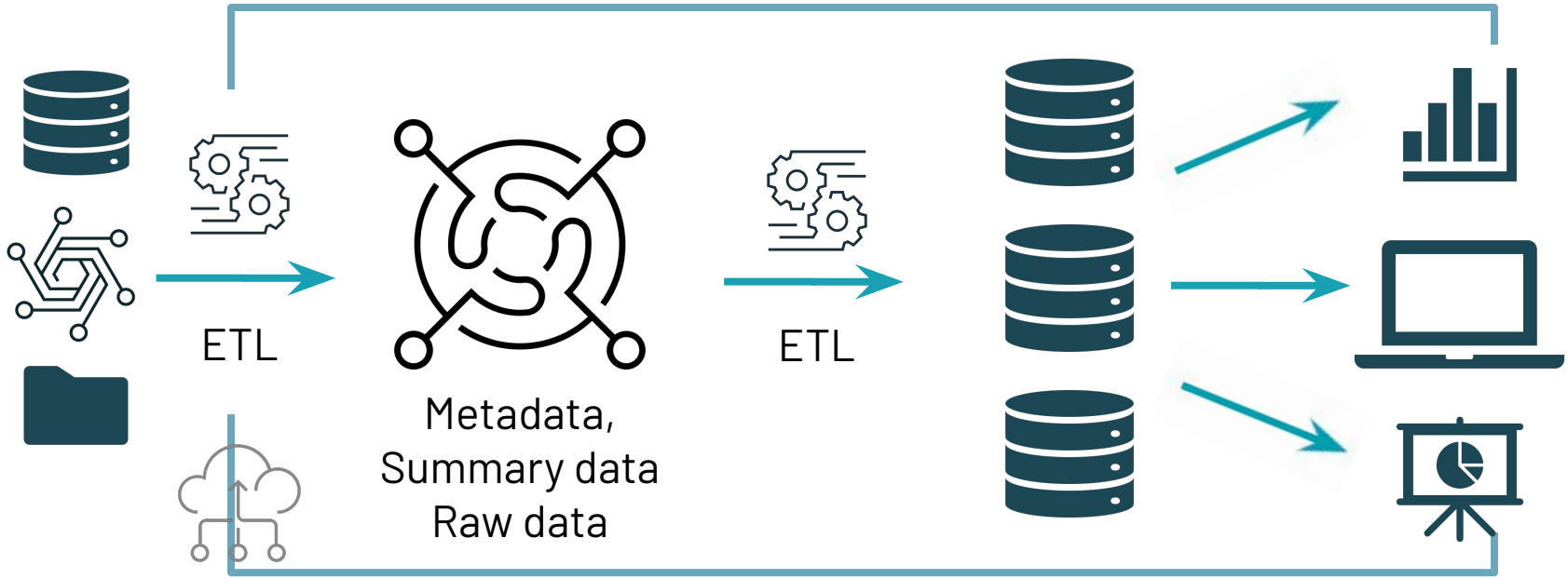
- Fact tables
- Dimension tables
- Aggregate fact tables



Bill Inmon's Theoretical Data Warehouse Architecture



Lakehouse



Operational
ODS

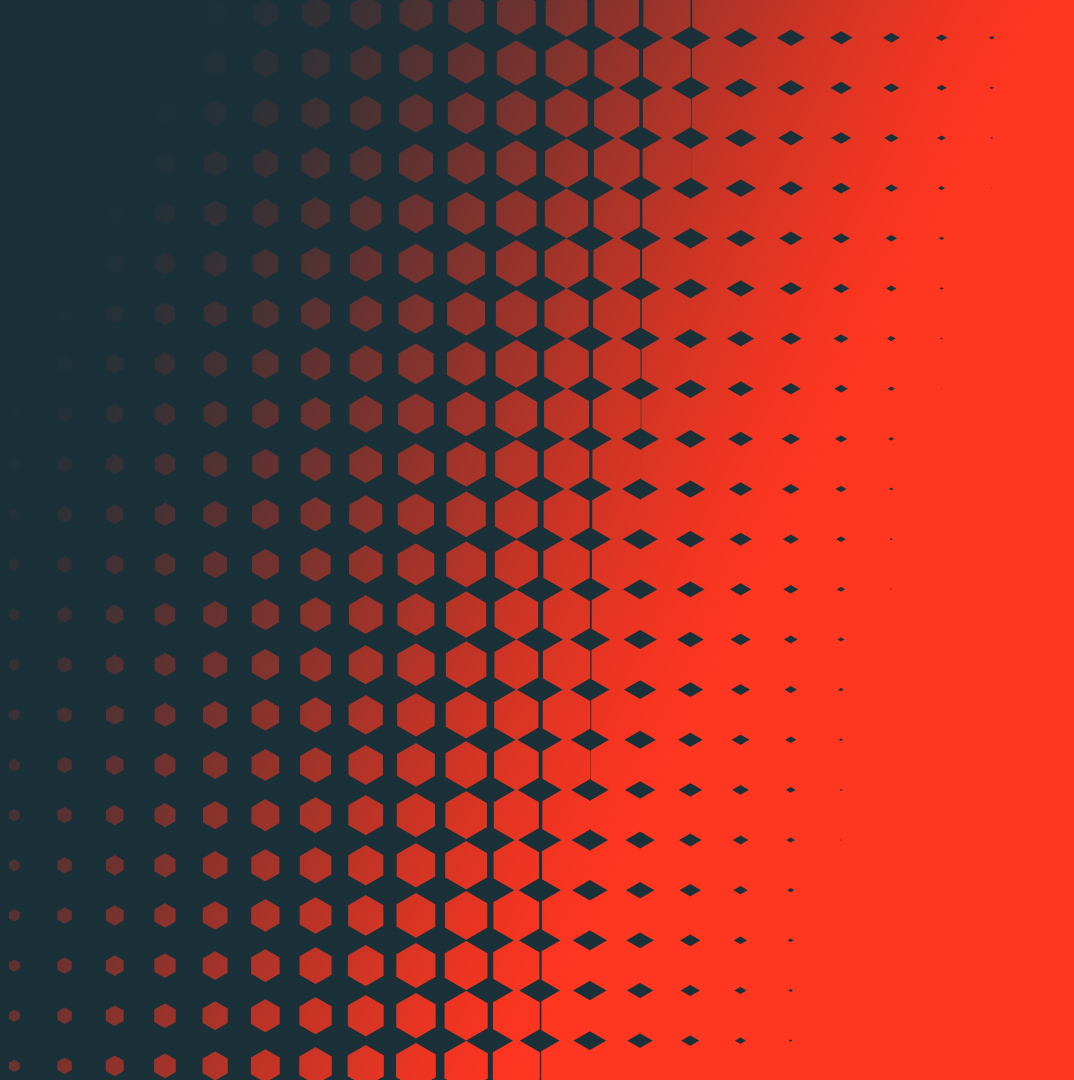
Atomic
EDSS
Data Lake

Departmental
Data Marts
Data warehouse

Individual
End users

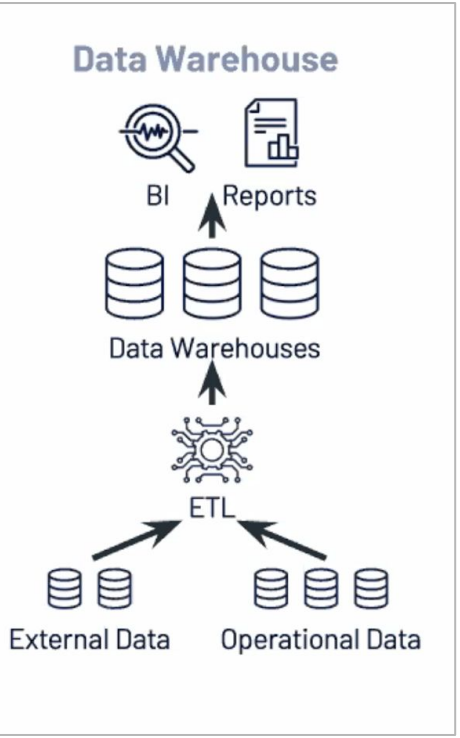
databricks

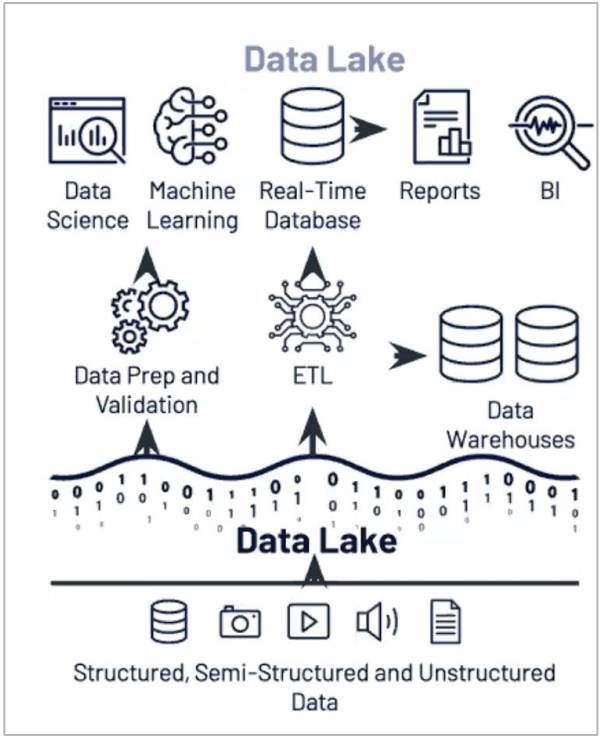
The Lakehouse

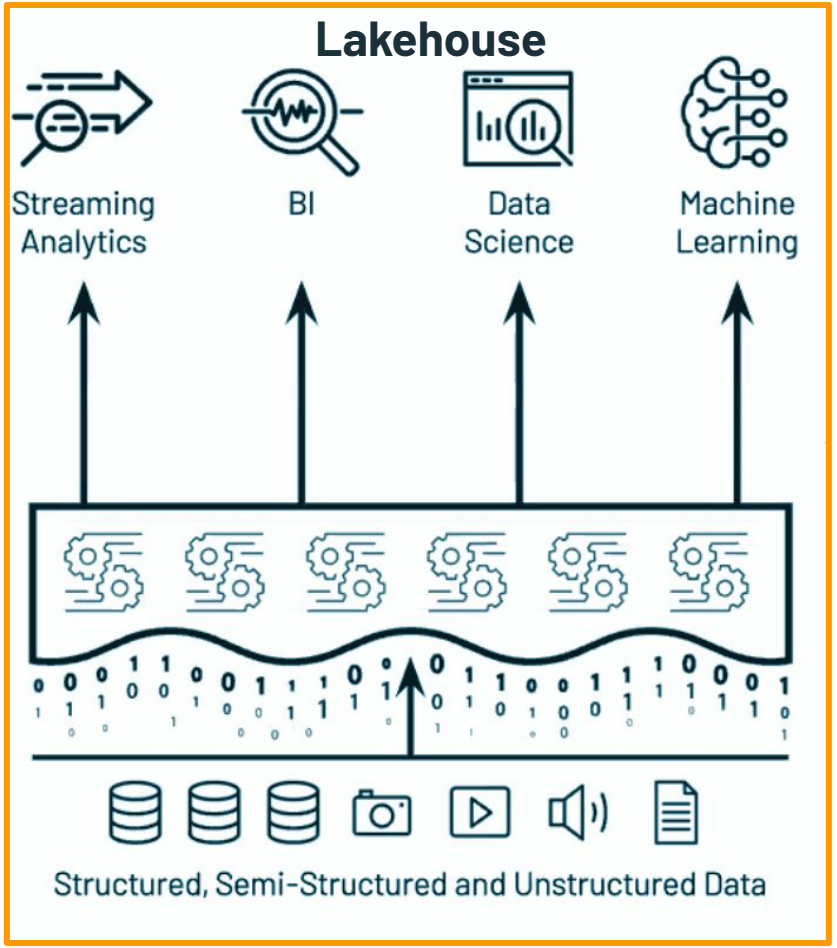


databricks

What is a Lakehouse?

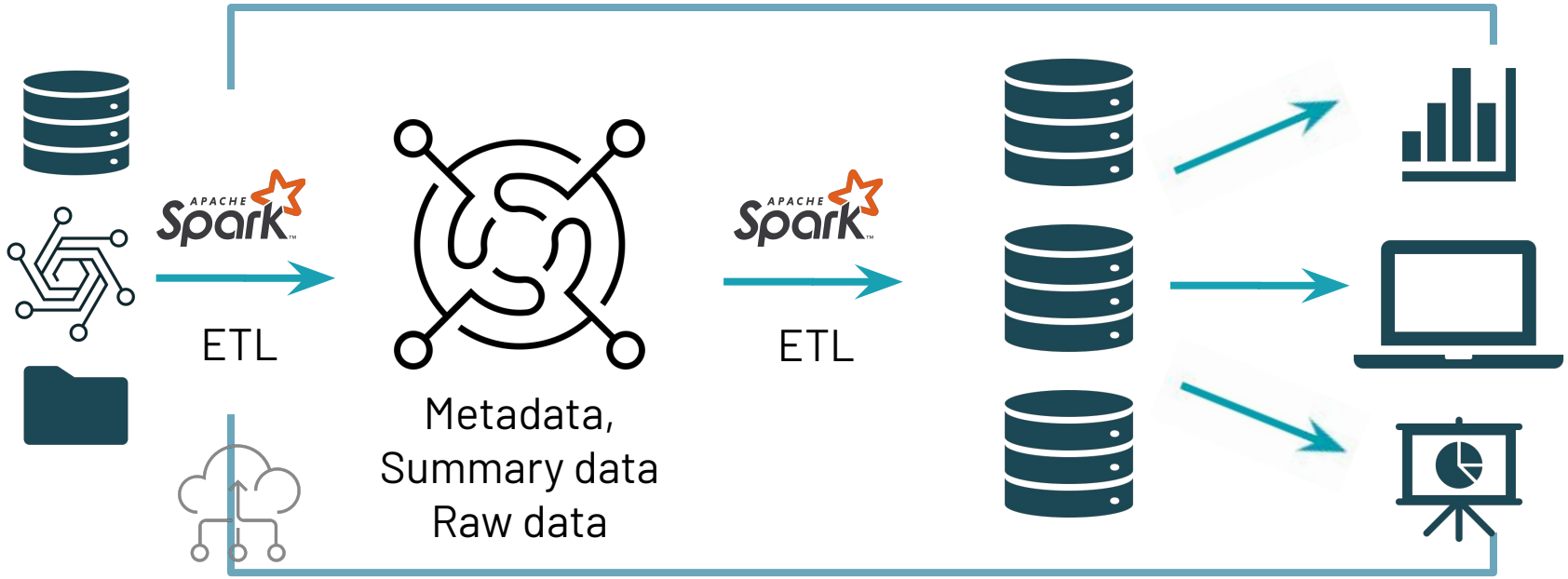






databricks

Levels of data in a Lakehouse



Operational
ODS

Atomic
EDSS
Data Lake

Departmental
Data Marts
Data warehouse

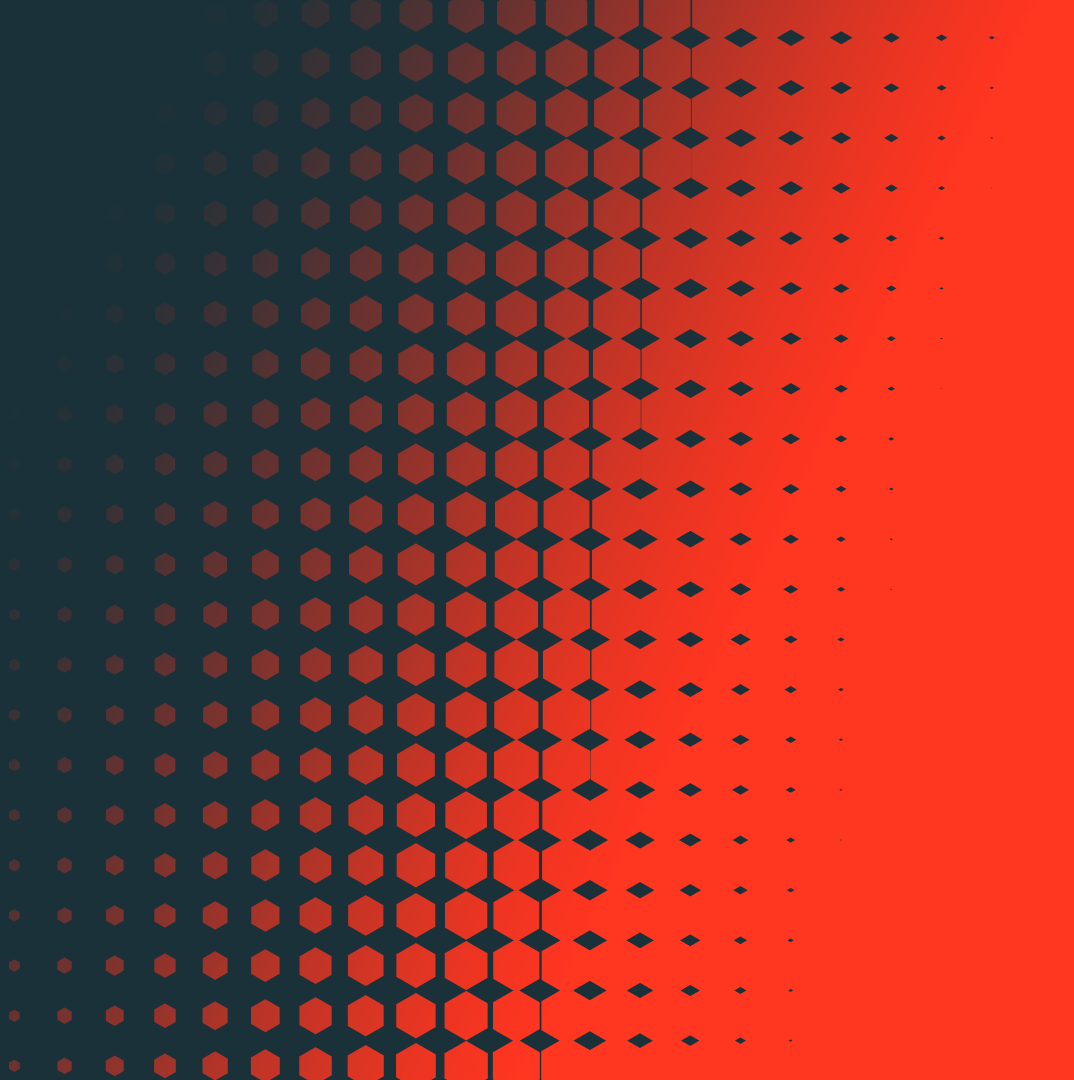
Individual
End users

databricks

Benefits of a Lakehouse

- Separation of compute and storage
- Infinite storage capacity
- Leverage best aspects of a data warehouse
- Low data gravity
- High data throughput
- No limits on data structure
- Mix batch and streaming workloads

The Delta architecture



databricks

What is **Delta Lake**?

- Technology designed to be used with Apache Spark to build robust data lakes

Delta Lake features

- ACID transactions on Spark
- Scalable metadata handling
- Streaming and batch unification
- Schema enforcement
- Time travel
- Upserts and deletes
- Fully configurable/optimizable
- Structured streaming support

Delta Lake components

Delta Lake components

Delta Lake storage layer

- Highly performant and persistent
- Low-cost, easily scalable object storage
- Ensures consistency
- Allows for flexibility

Delta Lake components

Delta Lake components

- Data in Parquet/Delta files
- Transaction log
- Registered in metastore (optional)

Data - Parquet files

- File format for tabular data stored as columns
- Fast and powerful
- Delta files = Parquet + versioning + metadata

Apache Parquet

- Initial effort by Twitter & Cloudera
- Open source storage format
  - Hybrid storage model (PAX)
- Widely used in Spark/Hadoop ecosystem (default format with Catalyst optimizer)
- One of the primary formats used by Databricks customers

Primitive Data Types

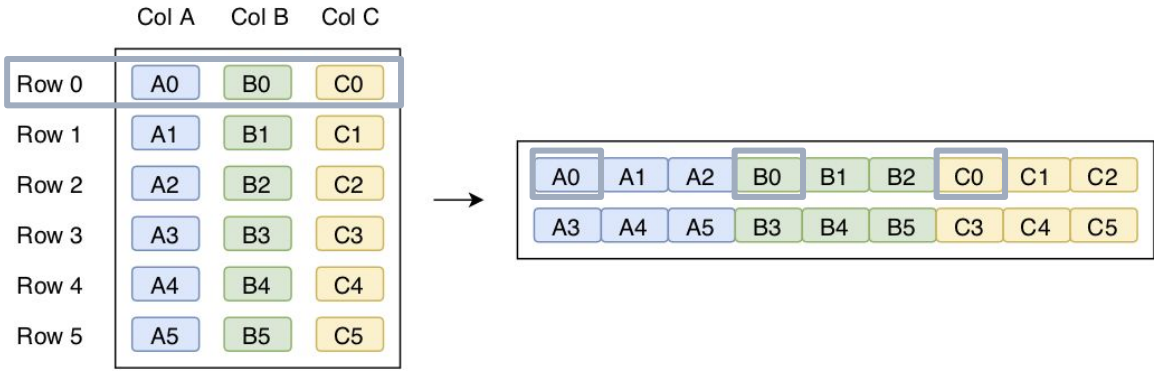
All data are encoded using a minimal set of types:

- BOOLEAN: 1 bit boolean
- INT32: 32 bit signed ints
- INT64: 64 bit signed ints
- INT96: 96 bit signed ints
- FLOAT: IEEE 32-bit floating point values
- DOUBLE: IEEE 64-bit floating point values
- BYTE_ARRAY: arbitrarily long byte arrays

Complex Types

- Parquet is self-describing for schema
- Logical types extend how primitive types should be interpreted
  - 16 bit integers are stored as INT32
  - Strings are stored as BYTE_ARRAY with UTF-8 encoding
- Arbitrarily nested data can be efficiently encoded
- Null values are not encoded in the data

Hybrid storage model



- Columnar storage
- Horizontal & vertical partitioning
- Used by Parquet & ORC
- Best of both worlds

Parquet “Files”

- On disk usually not a single file
- Logical file is defined by a root directory
  - Root dir contains one or multiple files

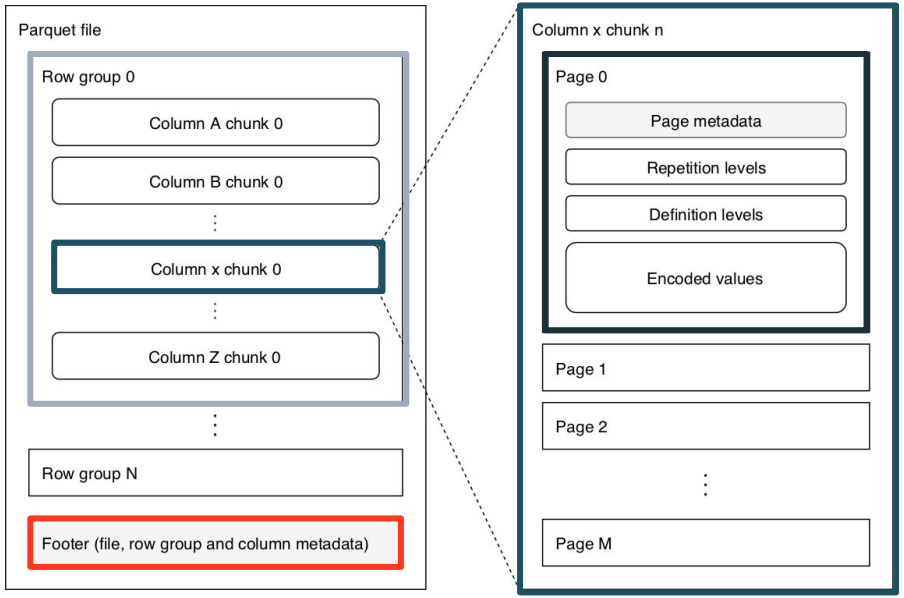
```
./example_parquet_file/
./example_parquet_file/part-00000-87439b68-7536-44a2-9eaa-1b40a236163d-c000.snappy.parquet
./example_parquet_file/part-00001-ae3c183b-d89d-4005-a3c0-c7df9a8e1f94-c000.snappy.parquet
```

  - **or** contains sub-directory structure with files in leaf directories

```
./example_parquet_file/
./example_parquet_file/country=Netherlands/
./example_parquet_file/country=Netherlands/part-00000-...-475b15e2874d.c000.snappy.parquet
./example_parquet_file/country=Netherlands/part-00001-...-c7df9a8e1f94.c000.snappy.parquet
```

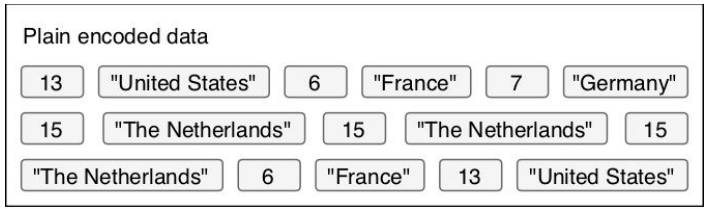
Data Organization

- Row-groups (*default 128MB*)
- Column chunks
- Pages (*default 1MB*)
  - Metadata
    - Min
    - Max
    - Count
  - Rep/def levels
  - Encoded values



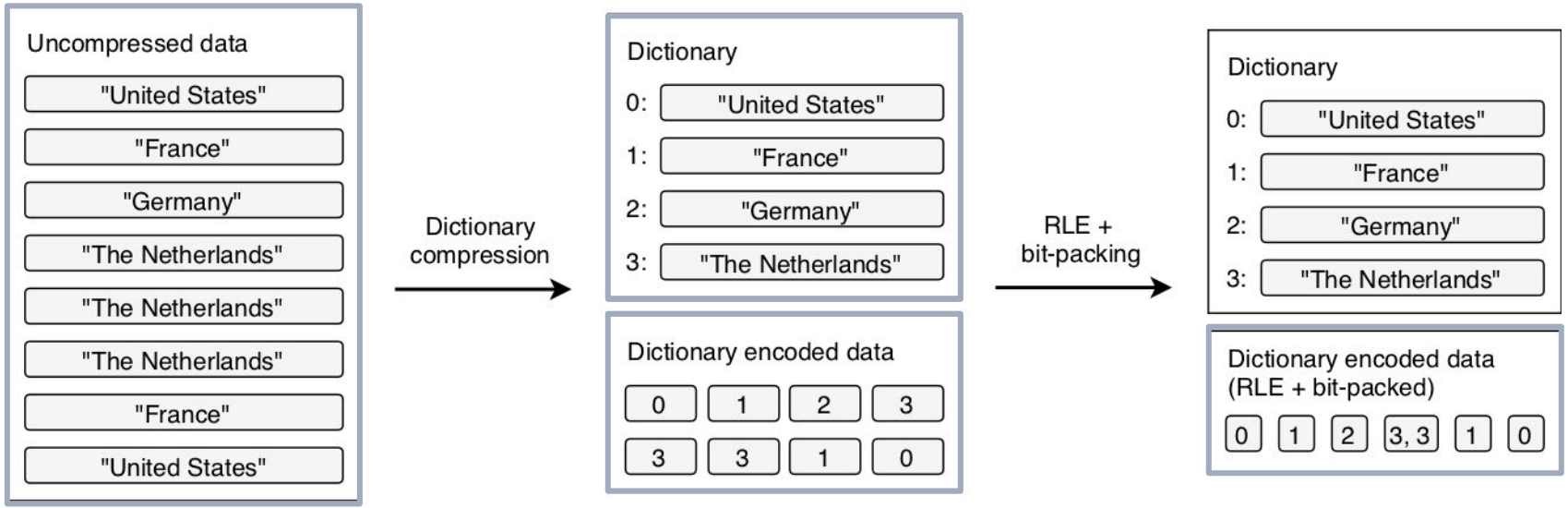
Encoding Schemes

- **PLAIN**
  - Fixed-width: back-to-back
  - Non fixed-width: length prefixed



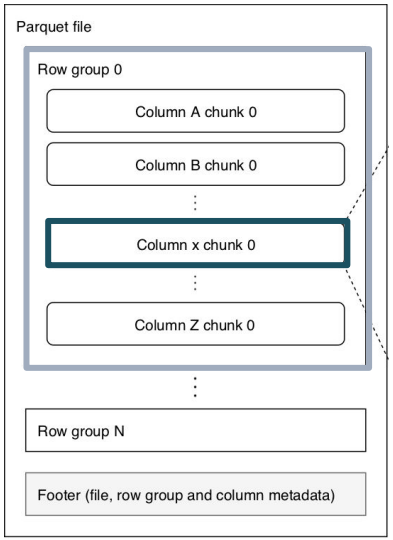
- **RLE_DICTIONARY**
  - Run-length encoding + bit-packing + dictionary compression
  - Assumes duplicate and repeated values

RLE_DICTIONARY



Dictionary Encoding

- Smaller files means less I/O
- Note: single dictionary per column chunk, size limit

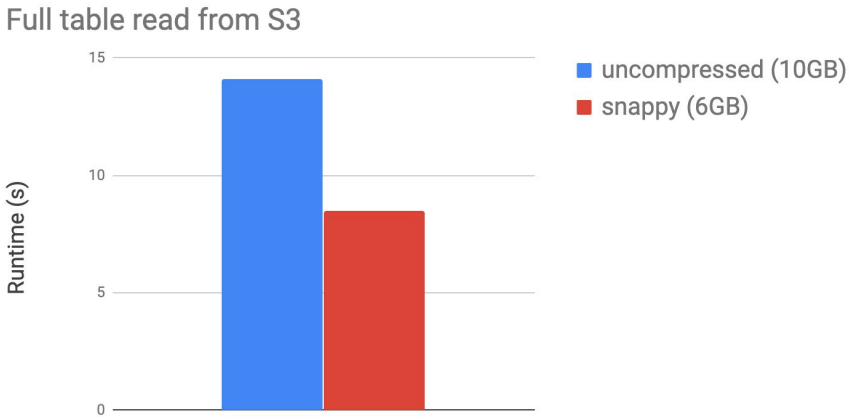


Dictionary too big?

Automatic fallback to `PLAIN`...

Page Compression

- Compression schemes (snappy, gzip, lzo...)
- Default: snappy
- Balances decompression speed vs I/O savings trade-off



Predicate Pushdown

```
SELECT * FROM table WHERE x > 5

Row-group 0: x: [min: 0, max: 9]
Row-group 1: x: [min: 3, max: 7]
Row-group 2: x: [min: 1, max: 4]
```

- Leverages min/max statistics
- Can also use dictionary filtering for fine-grained predicates
- Requires typed predicates

Partition Skipping

- Embed common predicates in directory structure
- Spark will skip directories

```
df.write.partitionBy("date").parquet(...)
```

```
./example_parquet_file/date=2019-10-15/...
./example_parquet_file/date=2019-10-16/...
./example_parquet_file/date=2019-10-17/part-00000-...-475b15e2874d.c000.snappy.parquet
```

…

Transaction log

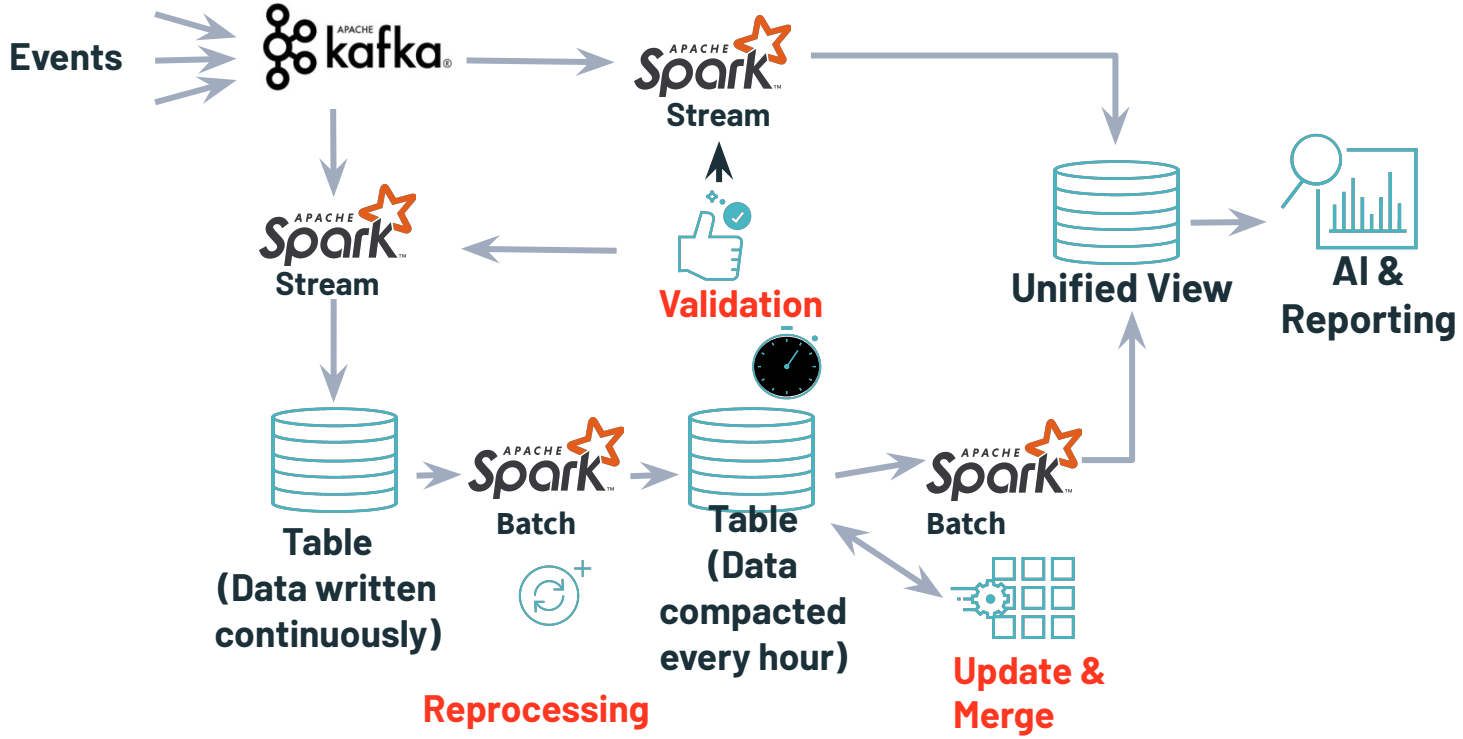
- Record of all transactions on a Delta table
- Prevents read conflicts
- Commits - ordered, atomic, json files
- Created automatically in the *_delta_log* subdirectory

Delta Lake components

Delta Engine

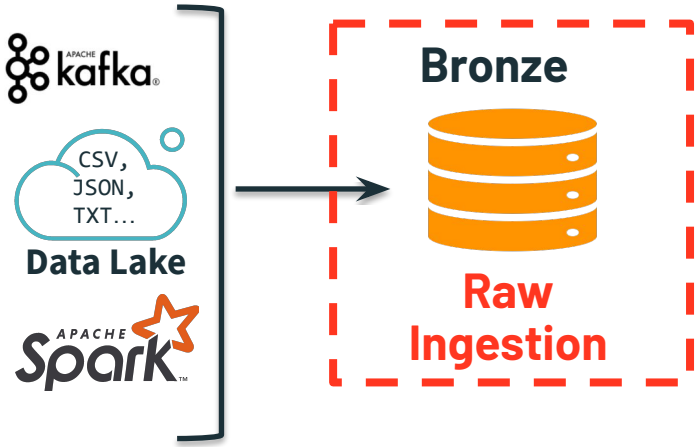
- File management optimizations
- Auto-optimized writes
- Performance optimization via Delta caching

The goal of a data engineer

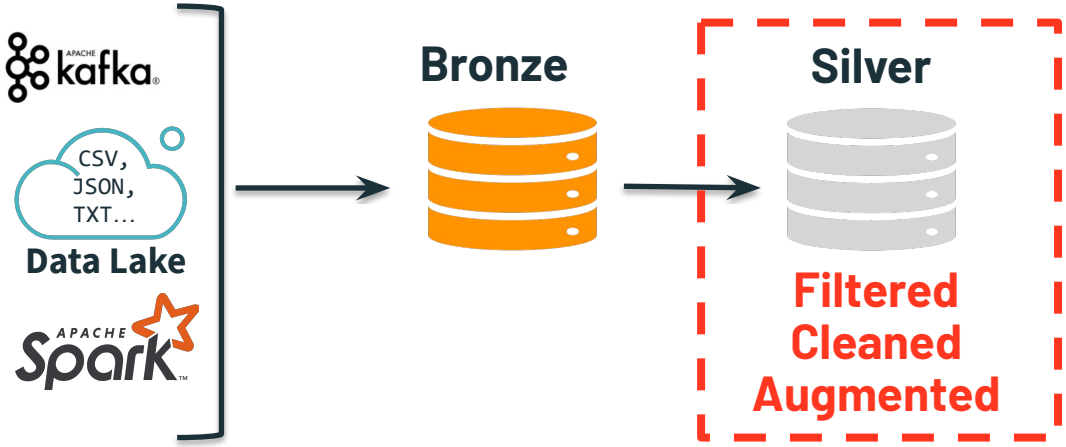


The Delta architecture design pattern

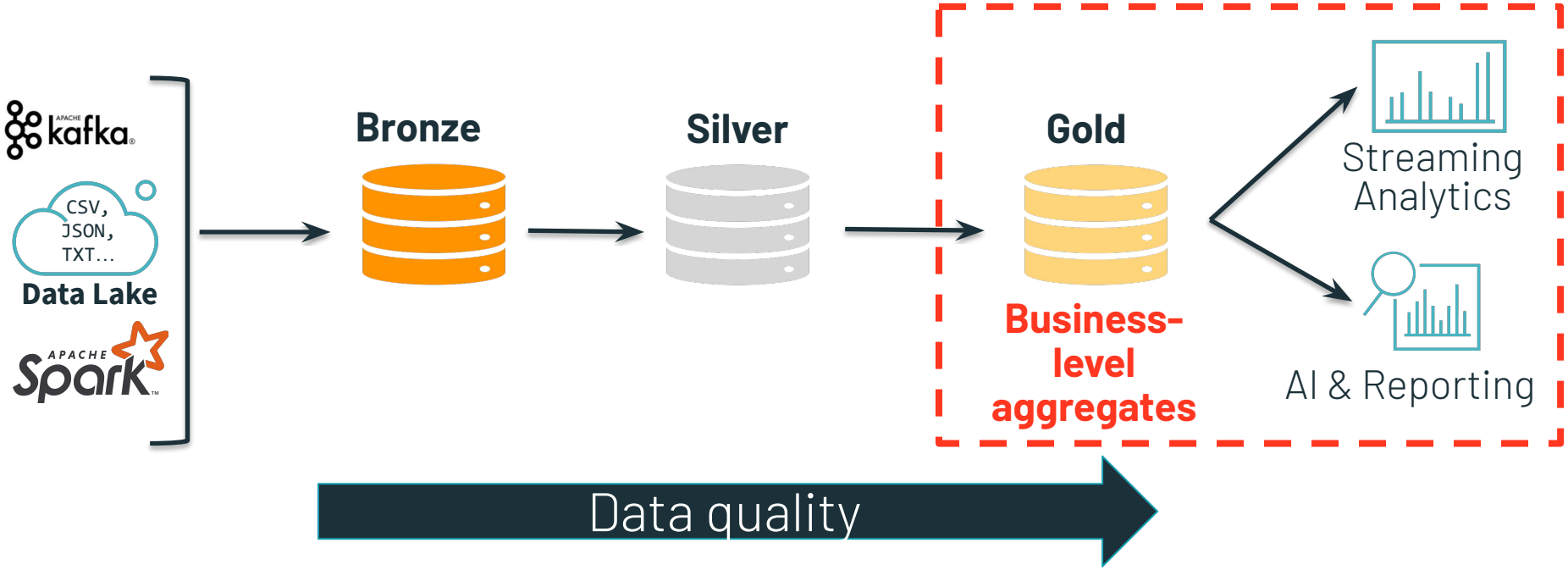
Delta architecture - Bronze



Delta architecture - Silver



Delta architecture - Gold




databricks