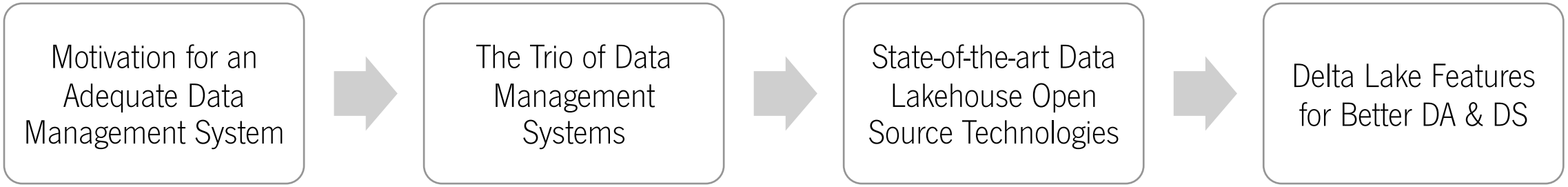
The **Data Lakehouse** for Simpler, Faster and More Integrated Data Analysis and Data Science

Carlos Costa

DSPT Webinar

April, 2022

Agenda

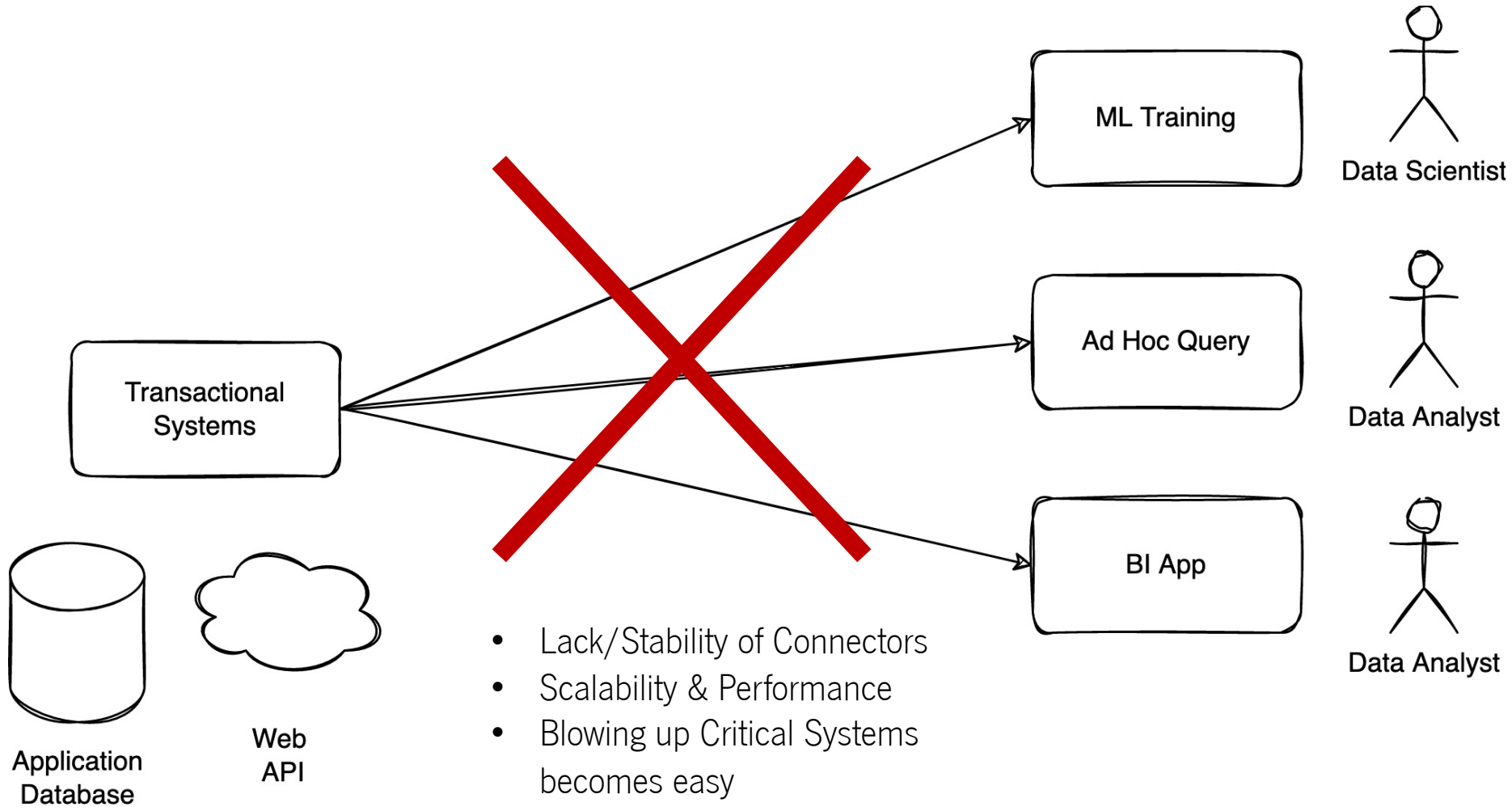

Motivation for an Adequate Data Management System
The Trio of Data Management Systems
State-of-the-art Data Lakehouse Open Source Technologies
Delta Lake Features for Better DA & DS

Disclaimer

These are my opinions and interpretations of the state of the art… does not mean I'm wrong or right! Use it as food for though… or ammo for discussion :P

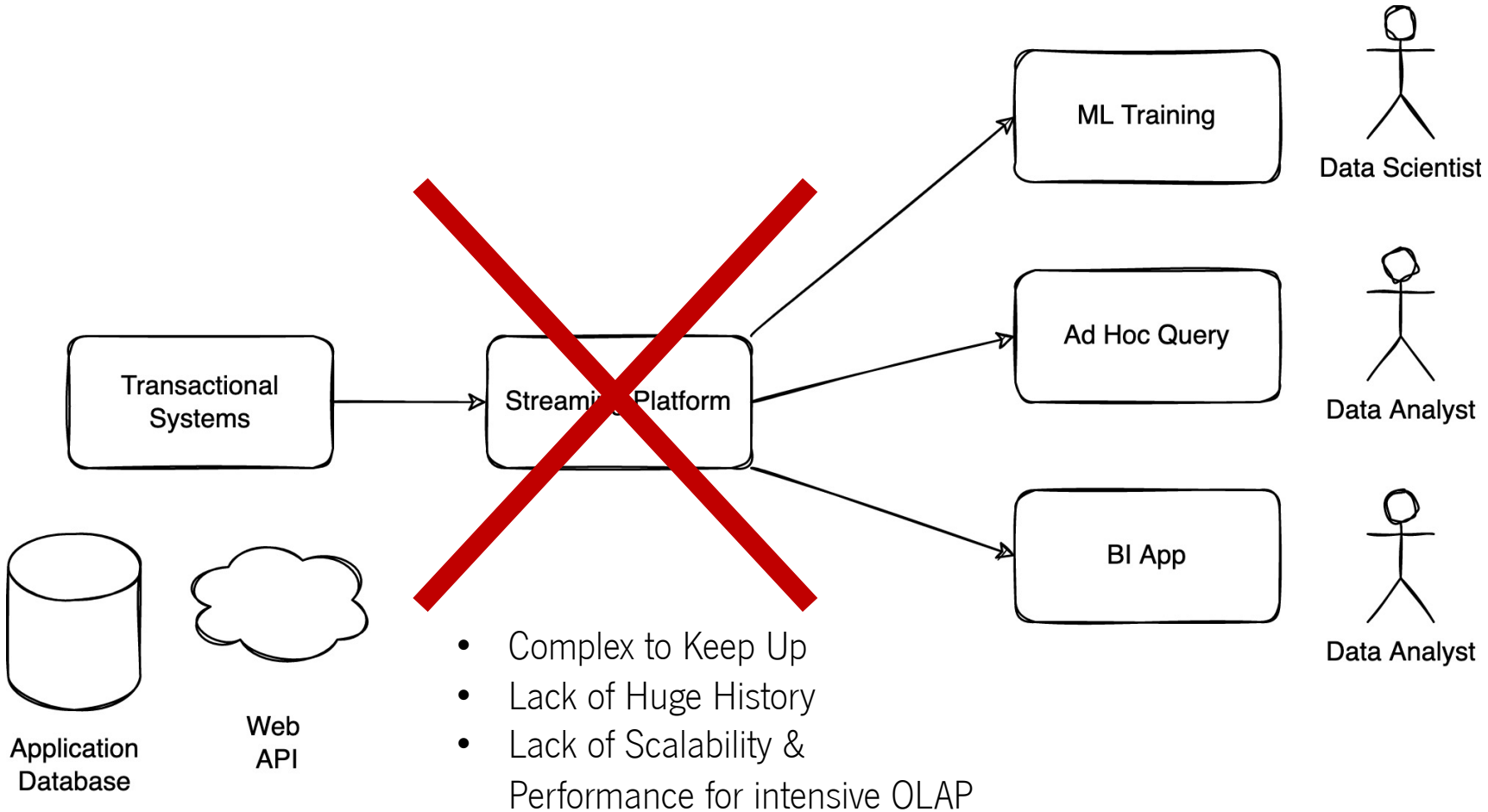
Motivation for an Adequate Data Management System

”Give me access to the Source” Pattern



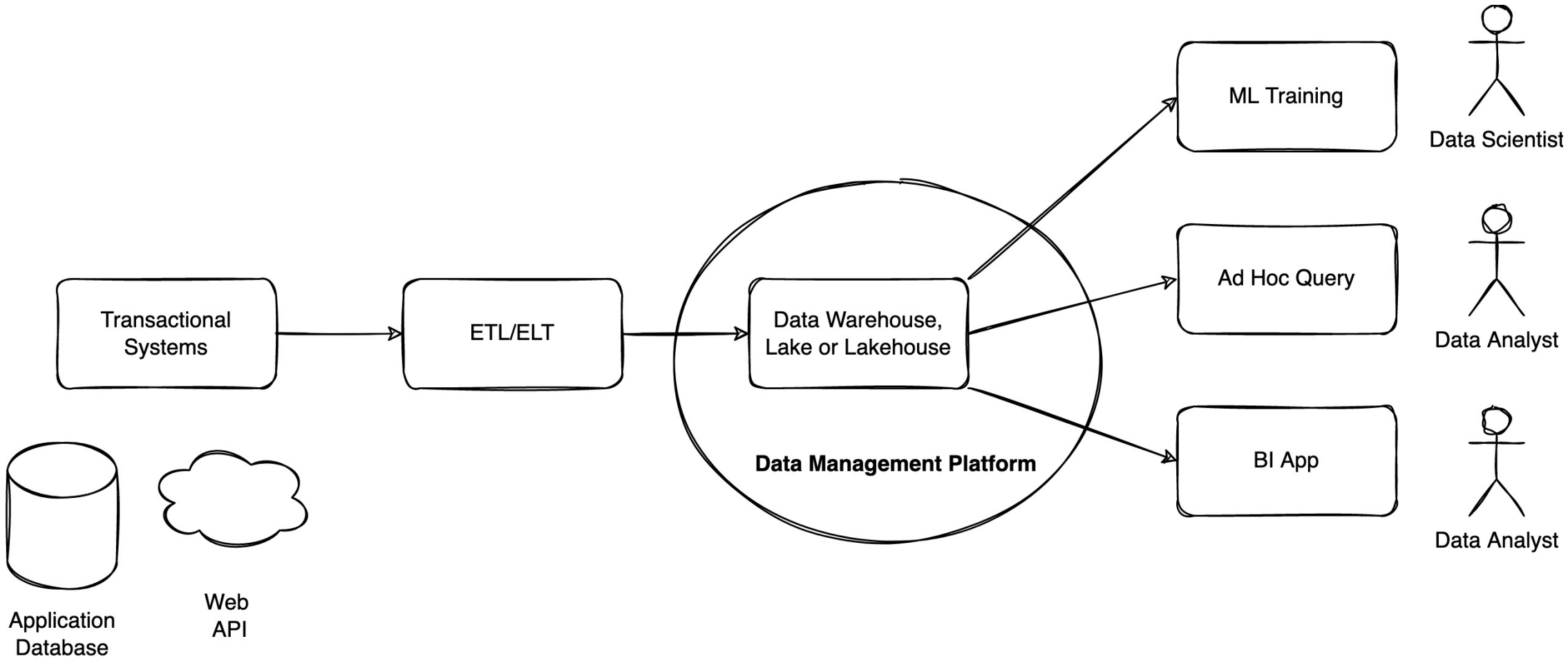
- Lack/Stability of Connectors
- Scalability & Performance
- Blowing up Critical Systems becomes easy

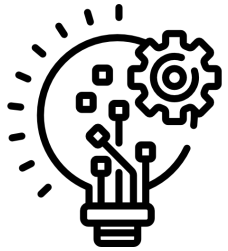
The “Streaming, Streaming, Streaming!!!” Pattern



- Complex to Keep Up
- Lack of Huge History
- Lack of Scalability & Performance for intensive OLAP

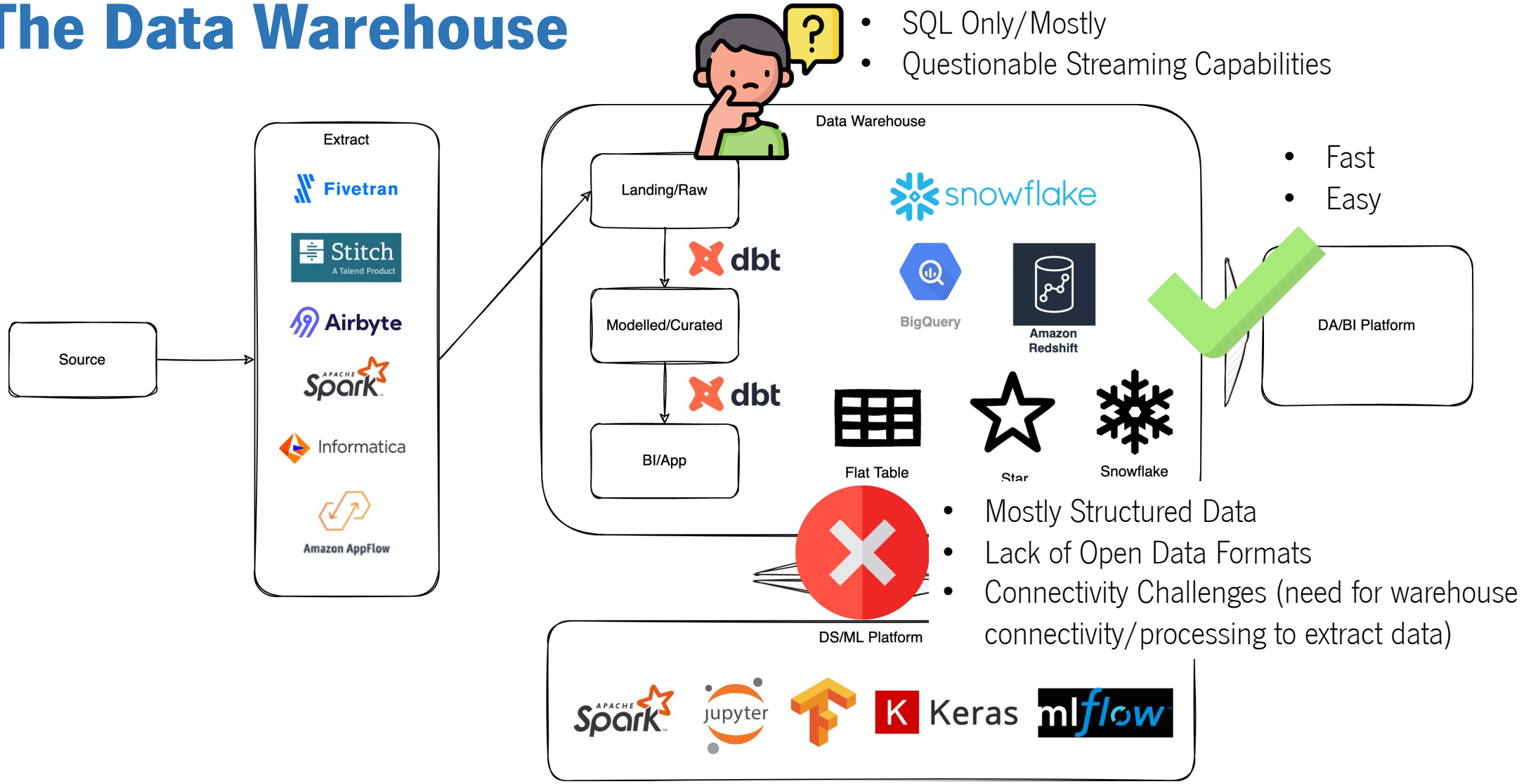
The Data Management Platform





The Trio of Data Management Platforms

The Data Warehouse



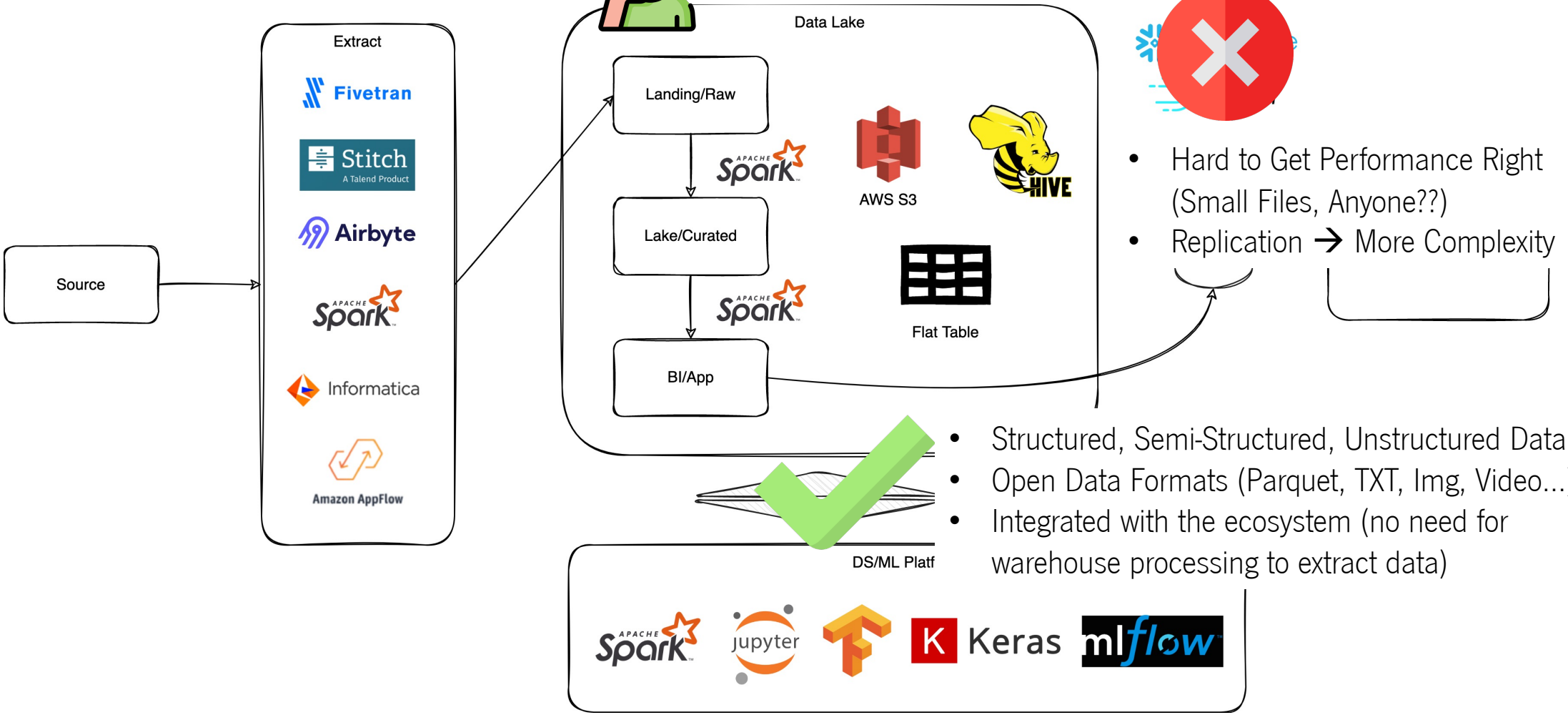
- SQL Only/Mostly
- Questionable Streaming Capabilities

- Fast
- Easy

- Mostly Structured Data
- Lack of Open Data Formats
- Connectivity Challenges (need for warehouse connectivity/processing to extract data)

The Data Lake

- Good to have Python, Scala, Java, SQL, but...
- Lack of ACID Transactions
- Lack of proper Schema Mngmt & DML

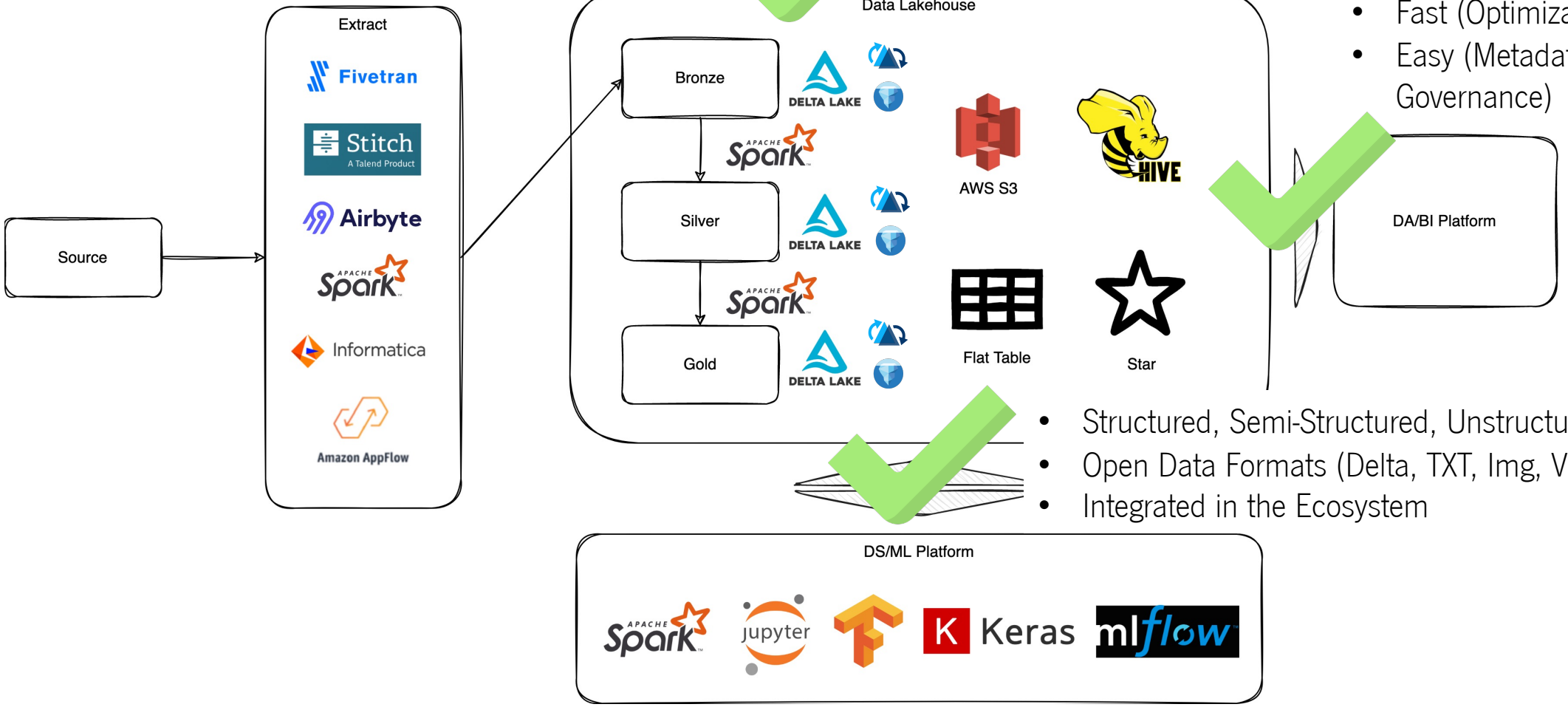


- Hard to Get Performance Right (Small Files, Anyone??)
- Replication → More Complexity

- Structured, Semi-Structured, Unstructured Data
- Open Data Formats (Parquet, TXT, Img, Video...)
- Integrated with the ecosystem (no need for warehouse processing to extract data)

The Data Lakehouse

- Python, Scala, Java, SQL...
- ACID Transactions
- Proper Schema Mngmt & DML

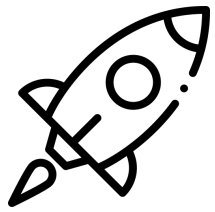


- Fast (Optimizations)
- Easy (Metadata & Governance)

- Structured, Semi-Structured, Unstructured Data
- Open Data Formats (Delta, TXT, Img, Video...)
- Integrated in the Ecosystem

Wrapping it Up

	Data Warehouse	*Data Lake*	*Data Lakehouse*
The Good	• BI/OLAP Speed • Easiness of Management and Access	• Handling All Volumes, Varieties and Velocities of Data • Several Programming Languages • Good Data Science Integration	• All the Benefits of the Data Warehouse and Data Lakehouse
The Bad	• Unstructured Data Handling • Streaming Capabilities are Questionable • SQL Only/Mostly	• Lack of ACID • Lack of proper DML • Lack of proper Schema Mngmt.	• *(Not so bad)* Performance is influenced by the querying processing technology (e.g., Spark, Presto, etc....)
The Ugly	• An "Alien" to the Data Science Ecosystem	• Replication into BI/OLAP oriented Stores (e.g., DW) • Complexity of the Replication	• ???



State-of-the-art Data Lakehouse Open-Source Technologies

- Open-Sourced by **Uber**
- *"Hadoop Upserts Deletes and Incrementals"*
- "Hudi brings transactions, record-level updates/deletes and change streams to data lakes"
- *Can use Parquet underneath*
- https://hudi.apache.org

- Open-Sourced by **Netflix**
- *"The open table format for analytic datasets"*
- "Iceberg is a high-performance format for huge analytic tables. Iceberg brings the reliability and simplicity of SQL tables to big data..."
- *Can use Parquet underneath*
- https://iceberg.apache.org

- Open-Sourced by **Databricks**
- *"Build Lakehouses with Delta Lake"*
- *"Delta Lake is an open-source storage framework that enables building a Lakehouse architecture with compute engines..."*
- *Parquet only underneath*
- https://delta.io

Delta Lake Features For Better DA & DS

ACID Transactions

- Readers see a consistent snapshot view of the data since the start of the query…
  - Even if the table was modified since then
- Multiple Writers can write to the same table partition concurrently and:
  - Always see a consistent snapshot view of the data
  - A serial order for these writes is ensured (isolation)
- General Steps:
  - Read latest state of the table to identify the files that need to be modified
  - Stage the new files to write
  - Validate if there are no conflicts and commit.

my_table/

_delta_log/

000000.json — *Add 2.parquet*

000001.json — *Remove 1.parquet*

…

000010.checkpoint.parquet

sales_date=2022-04-18/

sales_date=2022-04-19/

sales_date=2022-04-20/

Sources:
https://docs.delta.io/latest/concurrency-control.html
https://databricks.com/blog/2019/08/21/diving-into-delta-lake-unpacking-the-transaction-log.html

Comprehensive Data Manipulation Language (DML)

```
DELETE FROM dummy_table WHERE date < ‘2022-04-25’
DELETE FROM delta.`/foo/bar` WHERE date < ‘2022-04-25’
```

```
UPDATE dummy_table SET month = ‘January' WHERE month = ‘Jan’;
```

```
MERGE INTO dummy_table
USING new_data
ON dummy_table.id = new_date.id
WHEN MATCHED THEN
   UPDATE SET dummy_table.id = new_date.id, dummy_table.firstName = new_date.firstName, ...
WHEN NOT MATCHED THEN
   INSERT (...) VALUES (...)
```

Source: https://docs.delta.io/latest/

Schema Enforcement & Evolution

Fail if schema does not match ... **OR**

```
> df.write...option("mergeSchema", "true")
```

OR (to set it across the Spark Session)

```
> spark.conf.set("spark.databricks.delta.schema.autoMerge.enabled", "true")
```

OR (for overwrites)

```
> df.write.option("overwriteSchema", "true")
```

Source: https://docs.delta.io/latest/

Constraints

NOT NULL

```
> CREATE TABLE dummy_table (
        dummy_id INT NOT NULL,
        ...
)
> ALTER TABLE table CHANGE COLUMN dummy_name DROP NOT NULL;
```

CHECK CONSTRAINTS

```
> ALTER TABLE dummy_table ADD CONSTRAINT c CHECK (date > '2022-04-25')
```

Source: https://docs.delta.io/latest/

Unified Batch & Streaming

Read the Stream

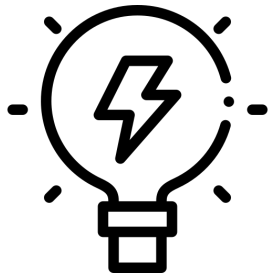
```
events = spark.readStream.format("delta").load("/foo/bar")
```

Transform

…

Write the Stream

```
events
  .writeStream
  .format("delta")
  .outputMode("append")
  .option("checkpointLocation", "/.../_checkpoints/...")
  .toTable(”table")
```



Tip of the day!

Batch = Streaming

```
.trigger(Trigger.Once())
```

Source: https://docs.delta.io/latest/

Audit History

```
DESCRIBE HISTORY delta.'/foo/bar/' -- full history
DESCRIBE HISTORY '/data/events/' LIMIT 1 -- last version only
DESCRIBE HISTORY eventsTable -- use table name
```

OR

```
from delta.tables import *
delta_table = DeltaTable.forPath(spark, '/foo/bar')
all_history_df = deltaTable.history()
last_version_df = deltaTable.history(1)
```

***Git Log* for your Data!**

```
+-------+-------------------+------+--------+---------+--------------------+----+--------+--------------------+
|version|          timestamp|userId|userName|operation| operationParameters| job|clusterId|operationMetrics    |
+-------+-------------------+------+--------+---------+--------------------+----+--------+--------------------+
|      5|2019-07-29 14:07:47|  null|    null|   DELETE|[predicate -> ["(...|null|    null| [numTotalRows -> ...|
|      4|2019-07-29 14:07:41|  null|    null|   UPDATE|[predicate -> (id...|null|    null| [numTotalRows -> ...|
|      3|2019-07-29 14:07:29|  null|    null|   DELETE|[predicate -> ["(...|null|    null| [numTotalRows -> ...|
|      2|2019-07-29 14:06:56|  null|    null|   UPDATE|[predicate -> (id...|null|    null| [numTotalRows -> ...|
|      1|2019-07-29 14:04:31|  null|    null|   DELETE|[predicate -> ["(...|null|    null| [numTotalRows -> ...|
```

Source: https://docs.delta.io/latest/

Time Travel

```
df1 = spark.read.format("delta").option("timestampAsOf", timestamp_string)...
df2 = spark.read.format("delta").option("versionAsOf", version)...
```

Use it for Fixing Problems (re-insert deleted user events)

```
df = spark.read.format("delta") \
  .option("timestampAsOf", yesterday) \
  .load("/foo/bar") \

df.where("user = 'dummy_user'") \
  .write.format("delta") \
  .mode("append") \
  .save("/foo/bar")
```

***Git Reset* for your Data!**

Source: https://docs.delta.io/latest/

Time Travel for Analytics

```
df = spark.read.format("delta”) \
  .option("timestampAsOf", last_week) \
  .load("/foo/bar”)

last_week_count = df.select(”user") \
  .distinct() \
  .count()

count = spark.read.format("delta") \
  .load("/foo/bar”) \
  .select(”user") \
  .distinct() \
  .count()

new_customers_count = count - last_week_count
```

Feature Store

(Customer, Product, Product Images Metadata, Stores...)

Reproducible Experiments

Read Changes

between 2 versions with ***Except***

Source: https://docs.delta.io/latest/

Compactation & Other Optimizations

Optimize the layout of data (compact small files into larger ones)

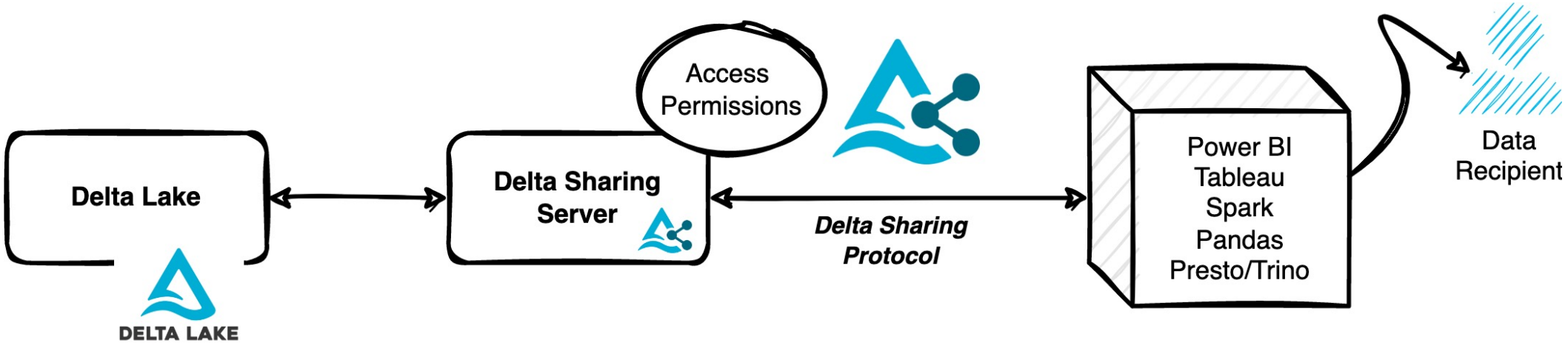
```
OPTIMIZE table
OPTIMIZE delta.`/foo/bar`

-- only optimize a subset of the table
OPTIMIZE table WHERE date >= '2022-04-25'
```

Data Skipping

- Used at query time to **skip unnecessary data**
- **Min** and **Max** values for each column **collected automatically** when writing
  - for *x* configurable number of columns
- Best Performance when used in combination with **"ordering before writing" pattern**

Source: https://docs.delta.io/latest/

Delta Sharing



> spark.read.format("deltaSharing").load(table_path)

> delta_sharing.load_as_pandas(...)

Source: https://delta.io/sharing/

Thank You!

Ask me anything ☺

Carlos Costa

DSPT Webinar

April, 2022