Lakehouse Technology as the Future of Data Warehousing

Wenchen Fan

databricks DataFun.

About databricks

Cloud-based data and AI platform for over 7000 customers

- Over 10 million VMs processing exabytes of data per day
- Exabytes of data under management

800+ engineers

Used for ETL, data science, ML and data warehousing

This Talk

Lakehouse systems: what are they and why now?

Building lakehouse systems

Ongoing projects

What Matters to Data Platform Users?

One might think performance, functions, etc, but these are secondary!

The top problems enterprise data users have are often with the data itself:

- Access: can I even get this data in the platform I use?
- Reliability: is the data correct?
- Timeliness: is the data fresh?

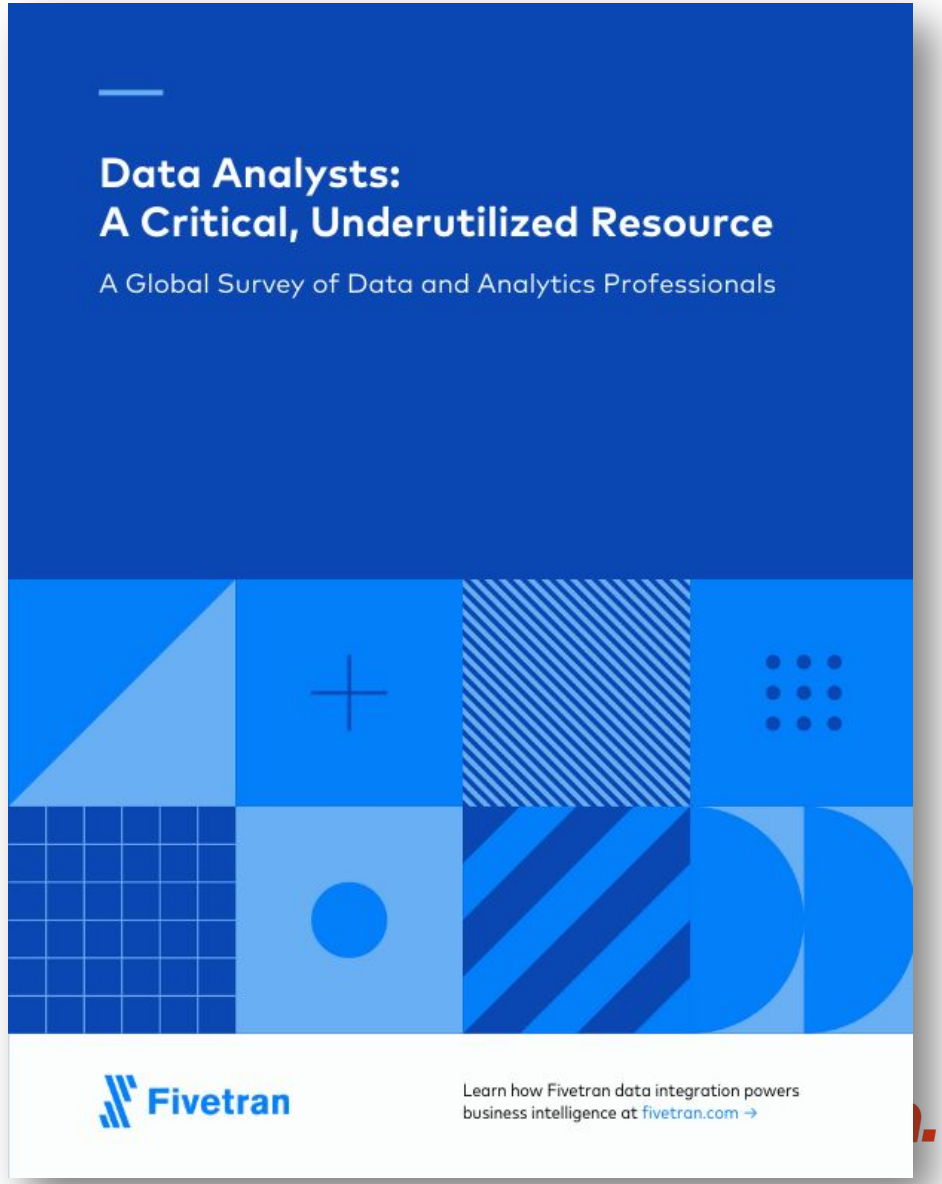
Without great data, you can't do any analysis!

Fivetran Data Analyst Survey

60% reported data quality as top challenge

86% of analysts had to use stale data, with **41%** using data that is >2 months old

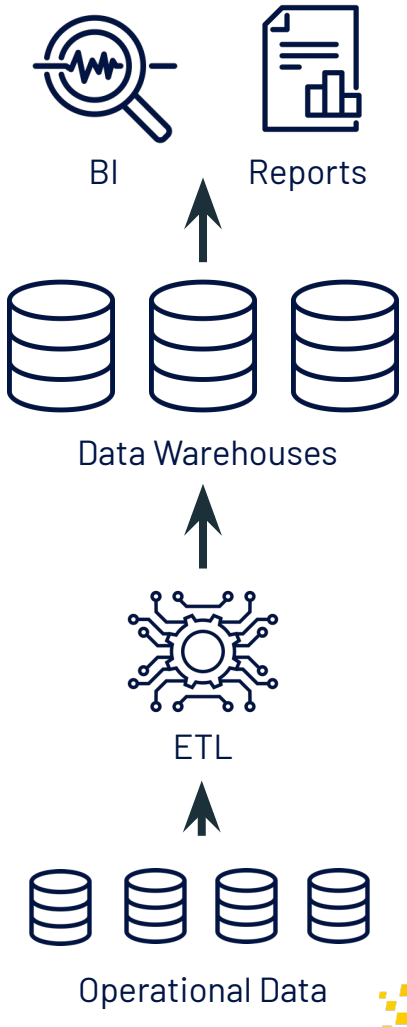
90% regularly had unreliable data sources



Getting high-quality, timely data is hard… but it's also a problem with system architectures!

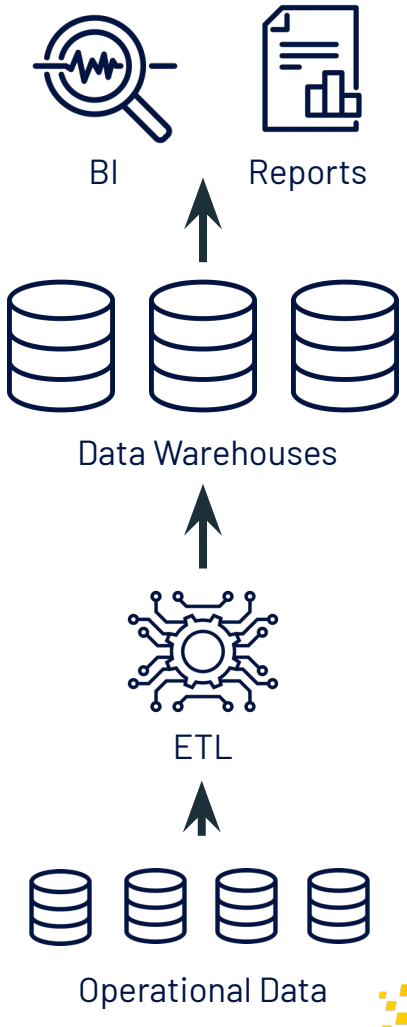
1980s: Data Warehouses

- ETL data directly from operational database systems
- Rich management and performance features for SQL analytics: schemas, indexes, transactions, etc



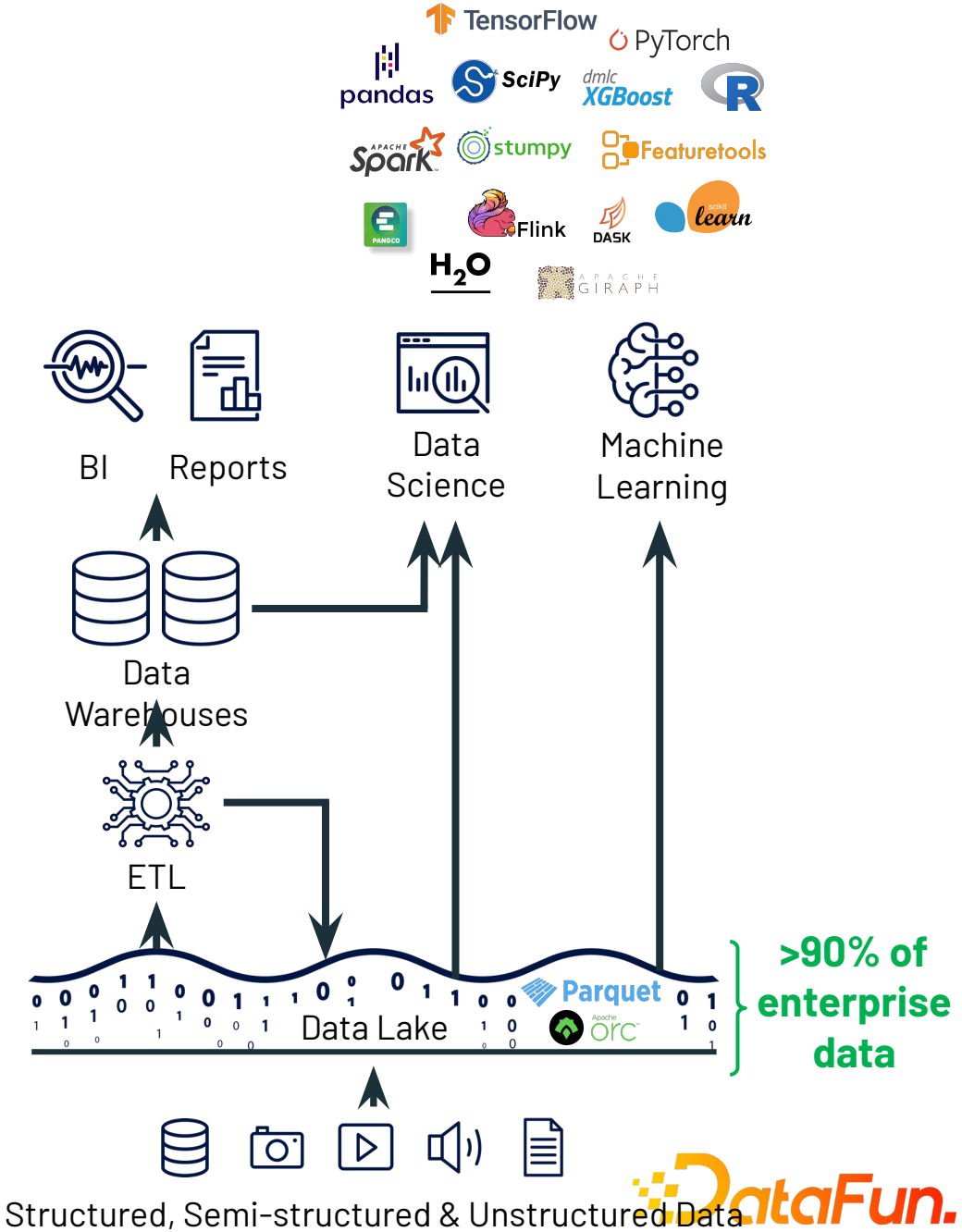
2010s: New Problems for Data Warehouses

- Could not support rapidly growing unstructured and semi-structured data: time series, logs, images, documents, etc
- High cost to store large datasets
- No support for data science & ML



2010s: Data Lakes

- **Low-cost storage** to hold all raw data with a file API (e.g. S3, HDFS)
- **Open file formats** (e.g. Parquet) accessible directly by ML / DS engines
- ETL jobs load specific data into warehouses, possibly for further ELT



Two incompatible architectures get in the way

Highly reliable and efficient

Business Intelligence

SQL Analytics

Governance and Security
Table ACLs

Data Warehouse

Structured tables

Copy subsets of data

All of the data and very adaptable

Data Science & ML

Data Streaming

Governance and Security
Files and Blobs

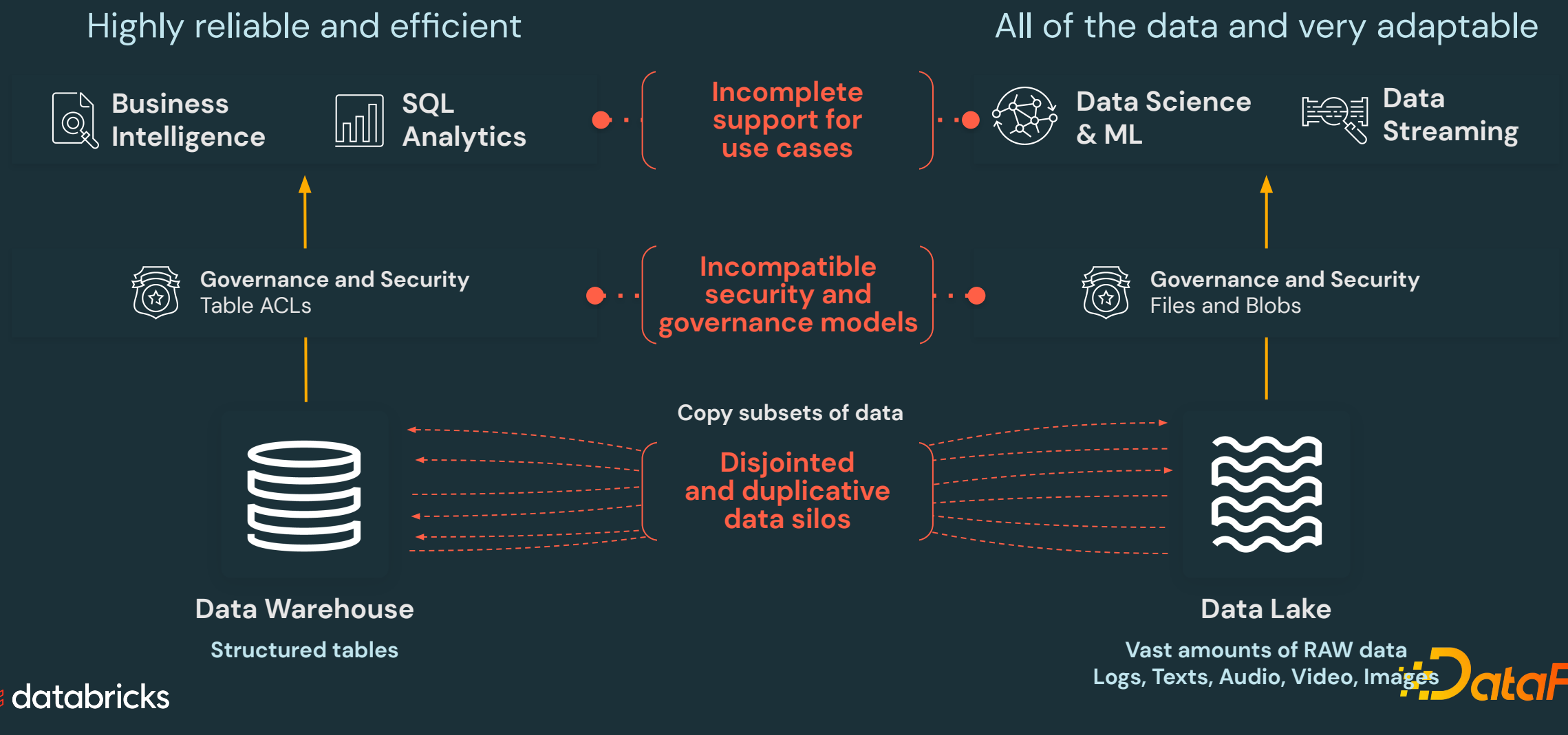
Data Lake

Vast amounts of RAW data
Logs, Texts, Audio, Video, Images

databricks

DataFun.

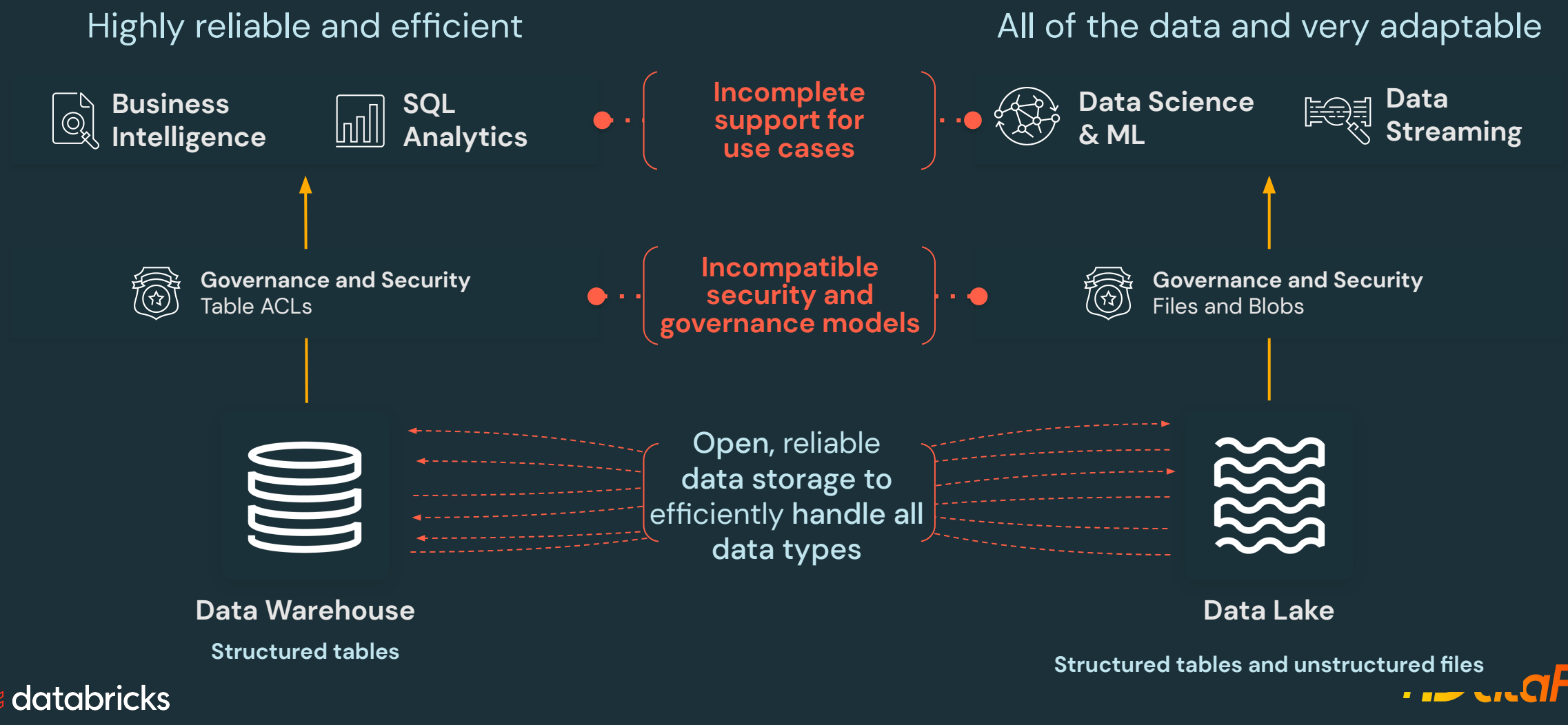
Two incompatible architectures get in the way



databricks

DataFun.

There is no need to have two disparate platforms



databricks

There is no need to have two disparate platforms

Highly reliable and efficient

All of the data and very adaptable

Business Intelligence

SQL Analytics

Incomplete support for use cases

Data Science & ML

Data Streaming

Governance and Security
Table ACLs

One security and governance approach for all data assets on all clouds

Governance and Security
Files, Blobs, and Table ACLs

Open, reliable data storage to efficiently handle all data types

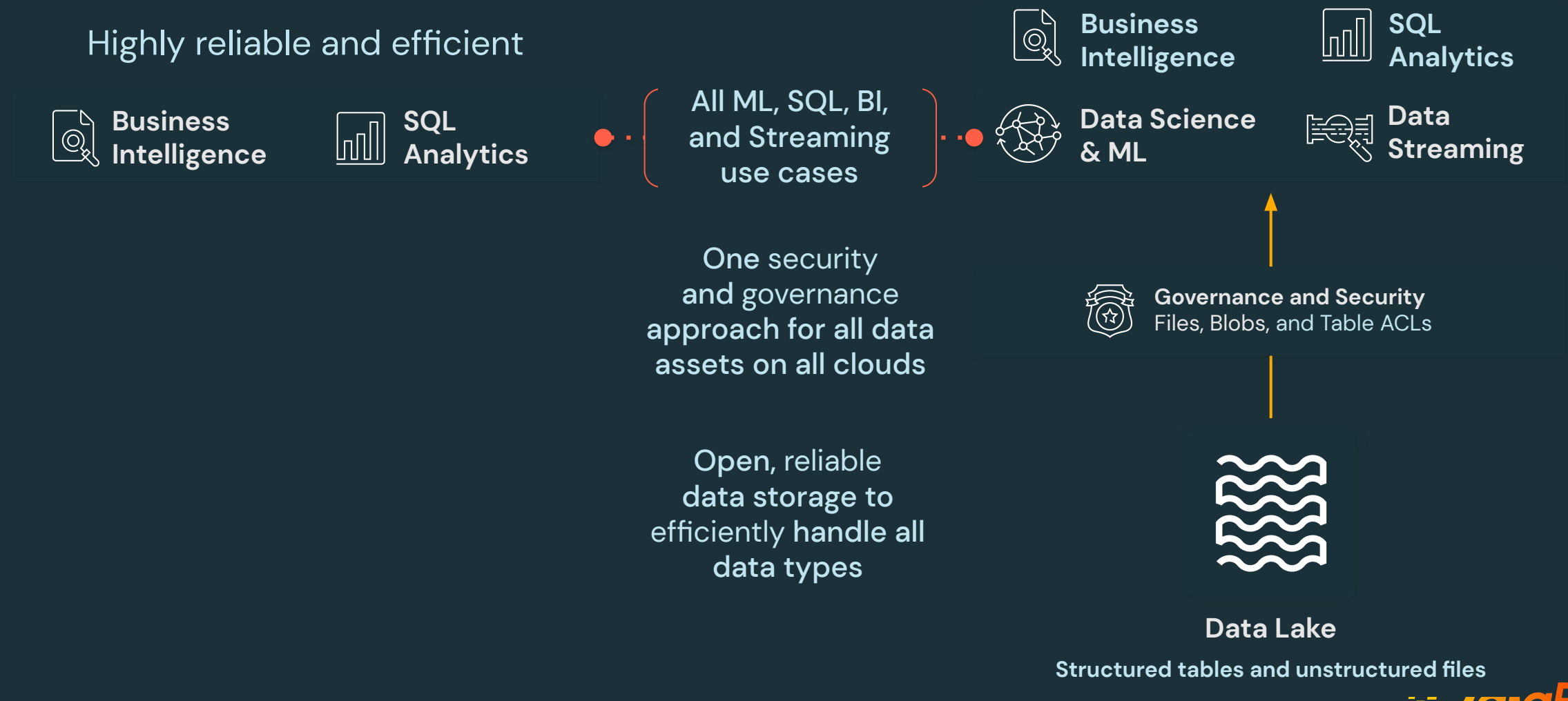
Data Lake

Structured tables and unstructured files

databricks

DataFun.

There is no need to have two disparate platforms



databricks

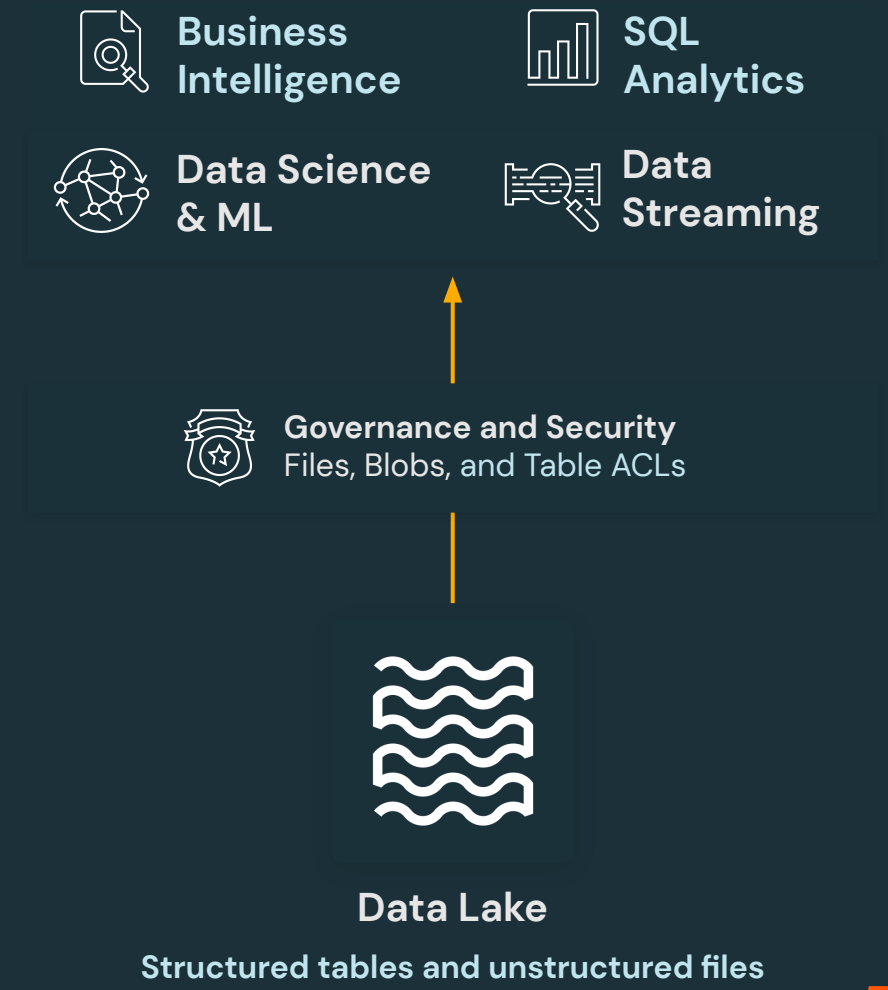
DataFun.

There is no need to have two disparate platforms

All ML, SQL, BI, and Streaming use cases

One security and governance approach for all data assets on all clouds

Open, reliable data storage to efficiently handle all data types



This Talk

Lakehouse systems: what are they and why now?

Building lakehouse systems

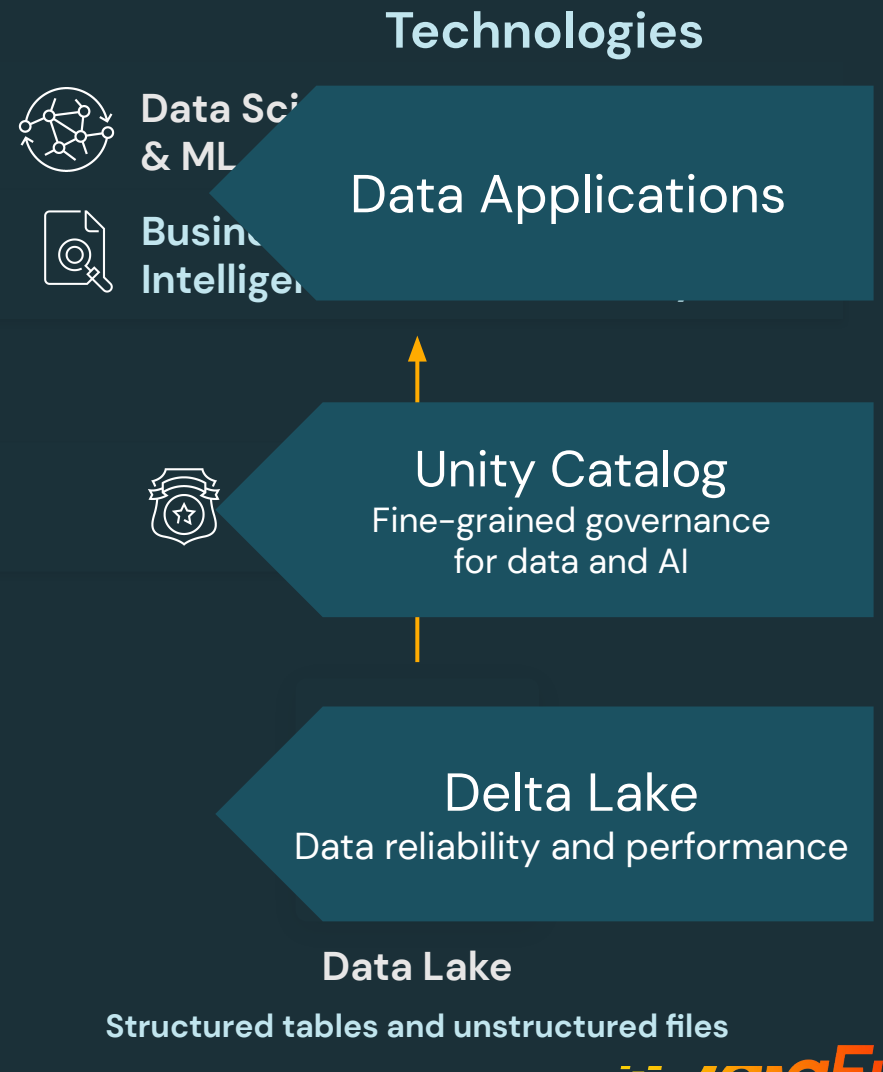
Ongoing projects

This is the lakehouse paradigm

All ML, SQL, BI, and Streaming use cases

One security and governance approach for all data assets on all clouds

Open, reliable data storage to efficiently handle all data types

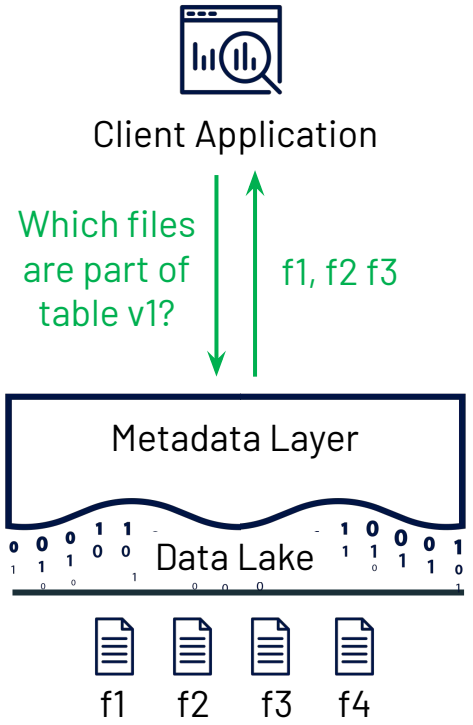


Key Technologies Enabling Lakehouse

1. Metadata layers on data lakes: add transactions, versioning & more
2. Lakehouse engine designs: performant SQL on data lake storage
3. Declarative I/O interfaces for data science & ML

Metadata Layers on Data Lakes

- Track **which files** are part of a table version to offer rich management features like transactions
  - Clients can then access the underlying files at high speed



- Examples:

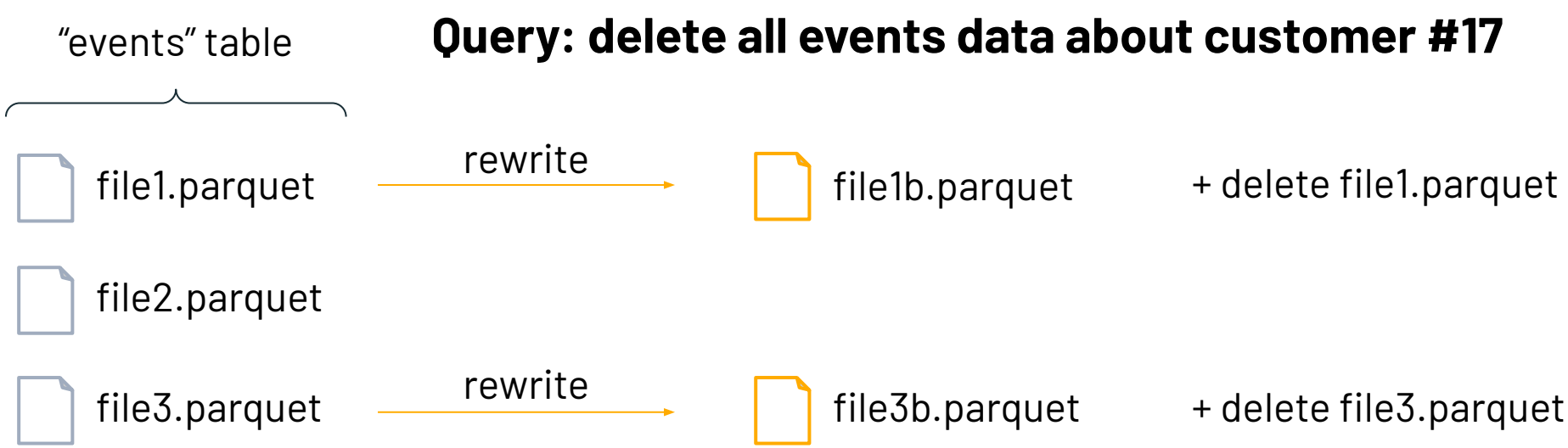
DELTA LAKE

ICEBERG

HIVE

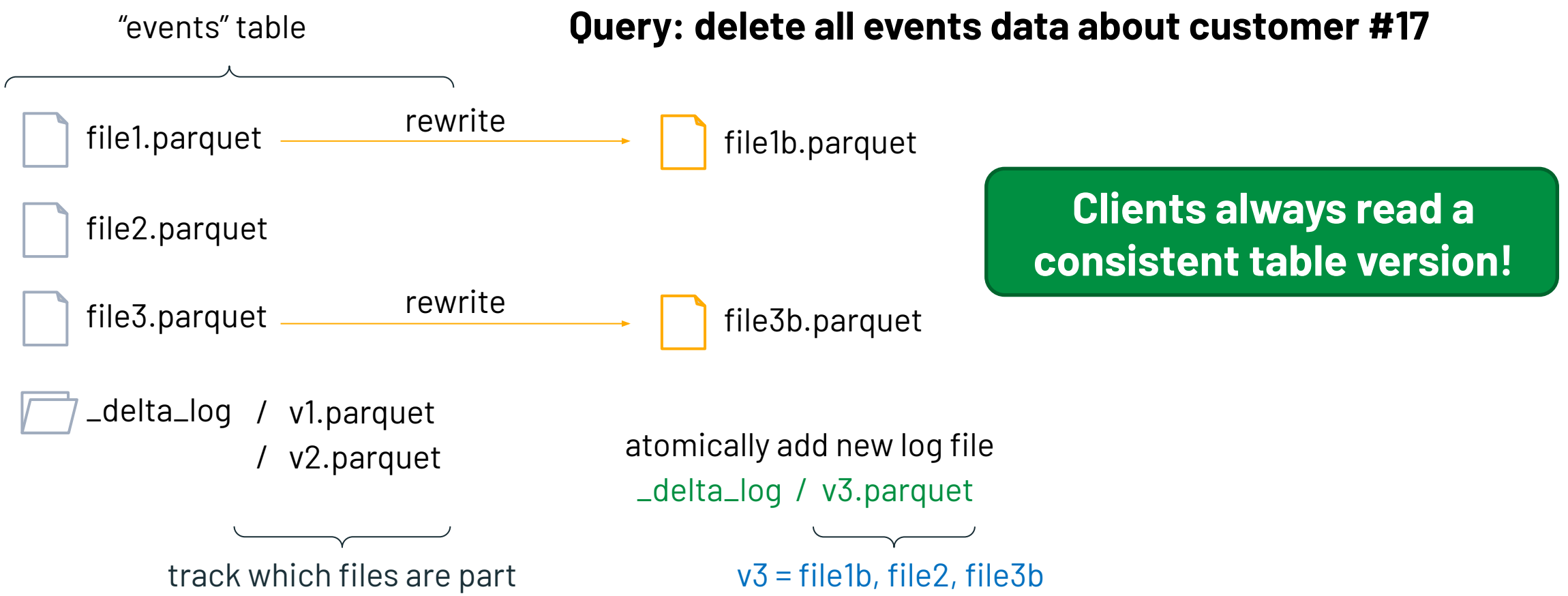
ACID

Example: Traditional Data Lake



Problem: What if a query reads the table while the delete is running?

Example: DELTA LAKE



databricks

Armbrust et al, VLDB 2020

DataFun.

Other Management Features with DELTA LAKE

- Time travel to old table versions
- Zero-copy CLONE by forking the log
- DESCRIBE HISTORY
- Schema enforcement & constraints

```
SELECT * FROM my_table
TIMESTAMP AS OF “2020-05-01”
```

```
CREATE TABLE my_table_dev
SHALLOW CLONE my_table
```

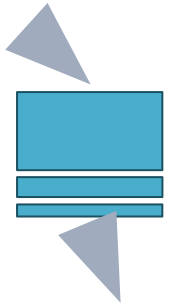
```
1 DESCRIBE HISTORY flightdelays
```

▸ (1) Spark Jobs

version	timestamp	userId	userName	operation	notebook
7	2019-10-08T16:47:22	101543	...@databricks.com	MERGE	▸{"notebookId":"25"}
6	2019-10-08T16:44:16	101543	...@databricks.com	MERGE	▸{"notebookId":"25"}
5	2019-10-06T19:26:53	101543	...@databricks.com	UPDATE	▸{"notebookId":"25"}

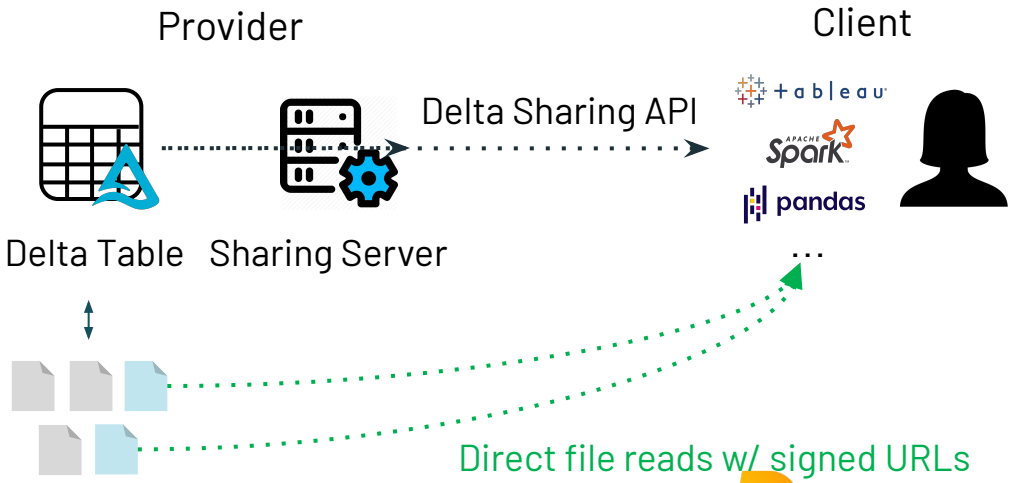
Other Management Features with DELTA LAKE

- Streaming I/O: treat a table as a stream of changes to remove need for message buses like Kafka



```
spark.readStream
     .format("delta")
     .table("events")
```

- Secure cross-organization sharing with Delta Sharing
  - Using cloud storage signed URLs to give clients fast access to data



Key Technologies Enabling Lakehouse

1. Metadata layers on data lakes: add transactions, versioning & more
2. Lakehouse engine designs: performant SQL on data lake storage
3. Declarative I/O interfaces for data science & ML

The Challenge

- Most data warehouses have full control over the data storage system and query engine, so they design them together

- The key idea in a Lakehouse is to store data in **open** storage formats (e.g. Parquet) for direct access from many systems

- How can we get great performance with these standard, open formats?

Enabling Lakehouse Performance

Even with a fixed, directly-accessible storage format, 4 optimizations help:

- Auxiliary data structures like statistics and indexes
- Data layout optimizations within files
- Caching hot data in a fast format
- Execution optimizations like vectorization

Minimize I/Os for cold data

Match DW performance on hot data

New query engines such as Databricks Photon Engine use these ideas

Optimization 1: Auxiliary Data Structures

- Even if the base data is in Parquet, we can build other data structures to speed up queries, and maintain them transactionally

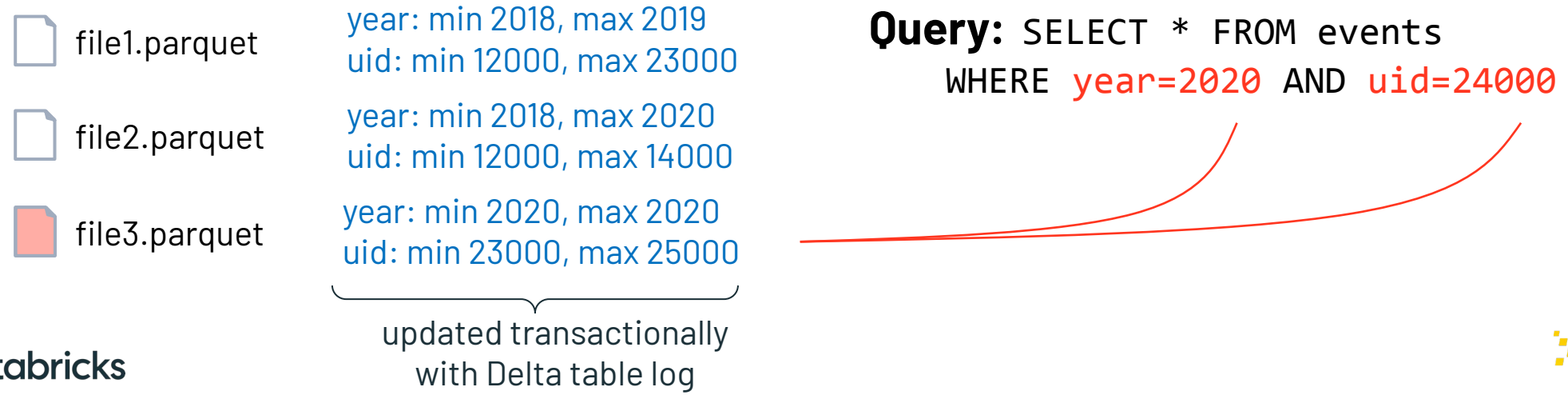
- **Example:** min/max zone maps for data skipping

Query:
```
SELECT * FROM events
  WHERE year=2020 AND uid=24000
```

Optimization 1: Auxiliary Data Structures

- Even if the base data is in Parquet, we can build other data structures to speed up queries, and maintain them transactionally

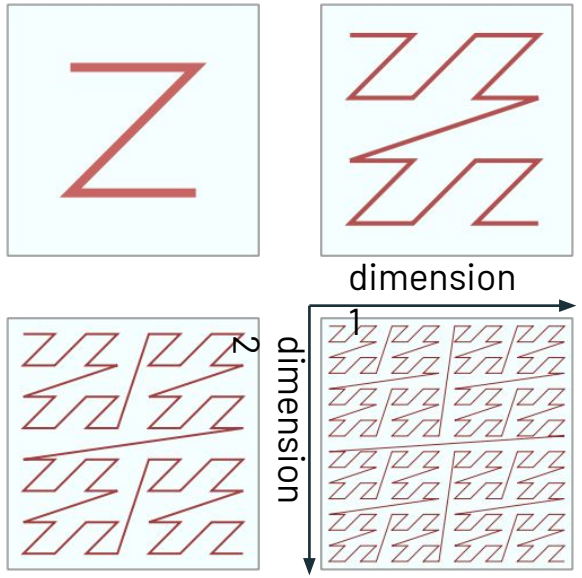
- **Example:** min/max zone maps for data skipping



Optimization 2: Data Layout

- Even with a fixed storage format such as Parquet, we can optimize the data layout *within* tables to minimize I/O

- **Example:** Z-order sorting for multi-dimensional clustering



Delta Lake 2.0: We are open-sourcing all of Delta

Coming Soon!


ACID Transactions


Scalable Metadata


Time Travel


Schema Evolution


OPTIMIZE

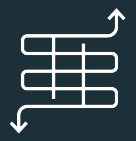

OPTIMIZE ZORDER


Change data feed


Generated column support w/ partitioning


Clones


Unified Batch/Streaming


Schema Enforcement


Audit History


DML Operations


Table Restore


S3 Multi-cluster writes

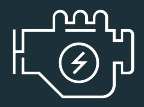

Data Skipping via Column Stats


Identity Columns


Iceberg to Delta converter


Compaction


MERGE Enhancements


Stream Enhancements

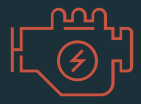

Simplified Logstore


Generated Columns


Multi-part checkpoint writes


Column Mapping

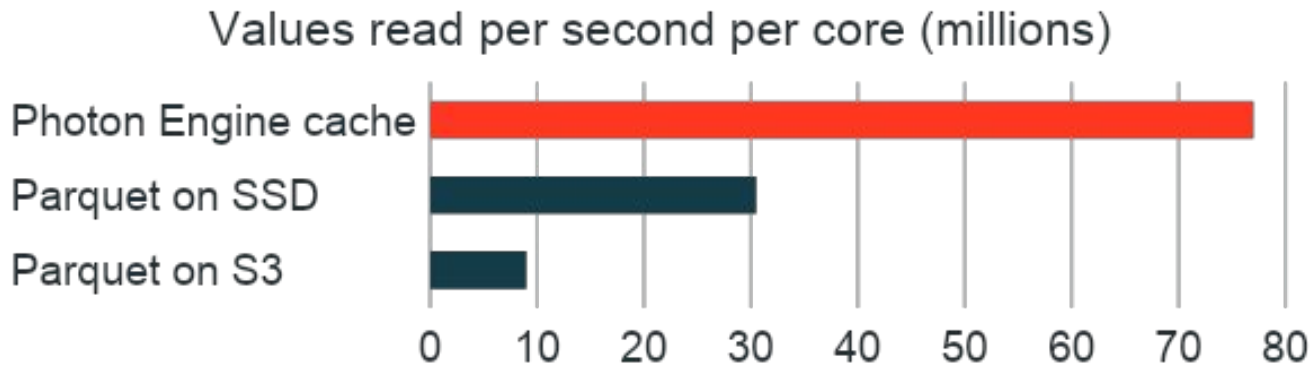

Subqueries in deletes and updates


Fast metadata only deletes

Optimization 3: Caching

- Most data warehouses cache hot data in SSD or RAM
- Can do the same in Lakehouse, using the metadata layer for consistency

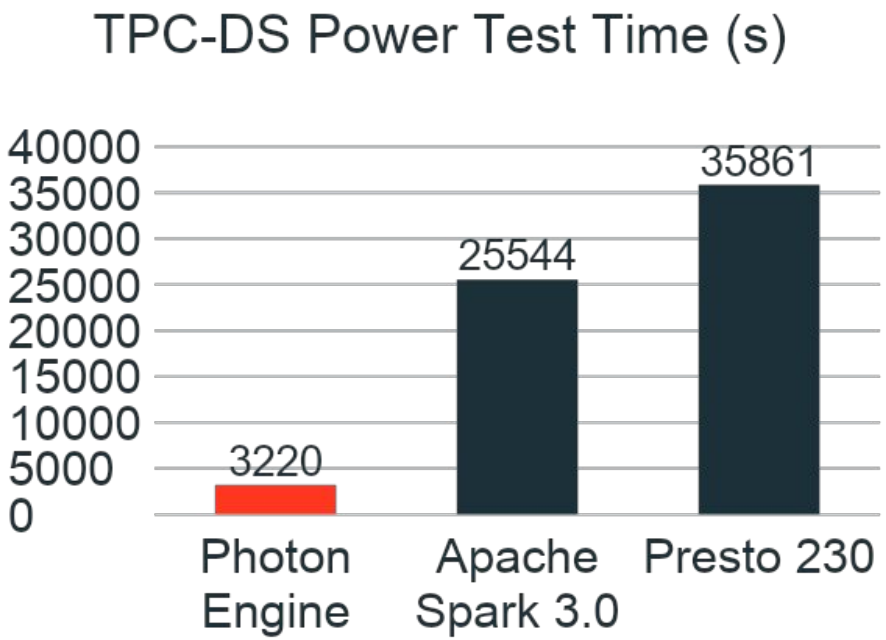
- **Example:** SSD cache in Photon Engine



Optimization 4: Vectorized Execution

- Many existing ideas can also be applied over open formats like Parquet

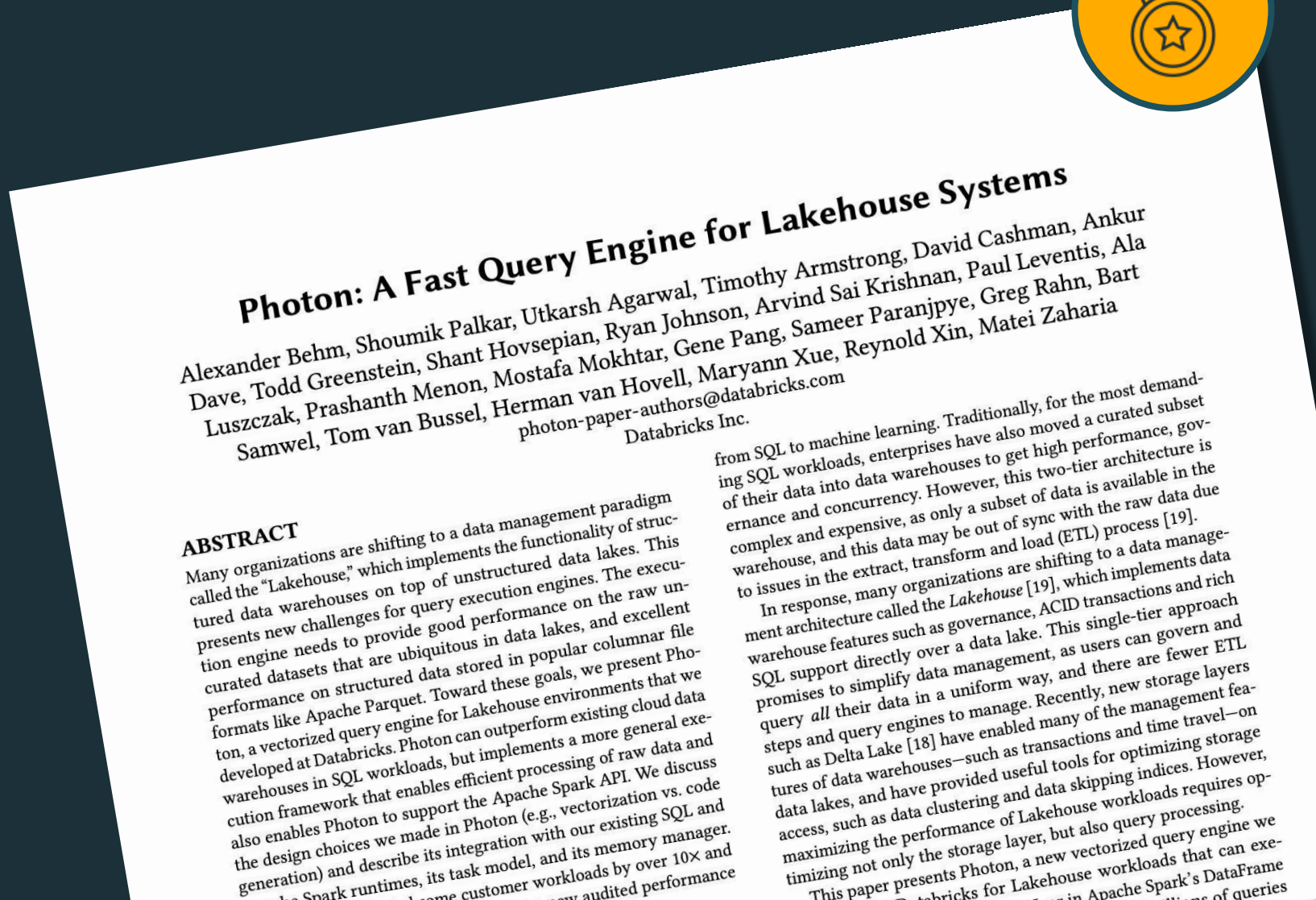
- **Example:** Databricks Photon vectorized engine



Photon

The Query Engine for Lakehouse Systems

The SIGMOD 2022 Best Industry Paper Award is awarded annually to one paper based on ***"the combination of real-world impact, innovation, and quality of the presentation."***



Photon: A Fast Query Engine for Lakehouse Systems

Alexander Behm, Shoumik Palkar, Utkarsh Agarwal, Timothy Armstrong, David Cashman, Ankur Dave, Todd Greenstein, Shant Hovsepian, Ryan Johnson, Arvind Sai Krishnan, Paul Leventis, Ala Luszczak, Prashanth Menon, Mostafa Mokhtar, Gene Pang, Sameer Paranjpye, Greg Rahn, Bart Samwel, Tom van Bussel, Herman van Hovell, Maryann Xue, Reynold Xin, Matei Zaharia
photon-paper-authors@databricks.com
Databricks Inc.

ABSTRACT

Many organizations are shifting to a data management paradigm called the "Lakehouse," which implements the functionality of structured data warehouses on top of unstructured data lakes. This presents new challenges for query execution engines. The execution engine needs to provide good performance on the raw uncurated datasets that are ubiquitous in data lakes, and excellent performance on structured data stored in popular columnar file formats like Apache Parquet. Toward these goals, we present Photon, a vectorized query engine for Lakehouse environments that we developed at Databricks. Photon can outperform existing cloud data warehouses in SQL workloads, but implements a more general execution framework that enables efficient processing of raw data and also enables Photon to support the Apache Spark API. We discuss the design choices we made in Photon (e.g., vectorization vs. code generation) and describe its integration with our existing SQL and Spark runtimes, its task model, and its memory manager. [illegible] customer workloads by over 10× and [illegible] audited performance

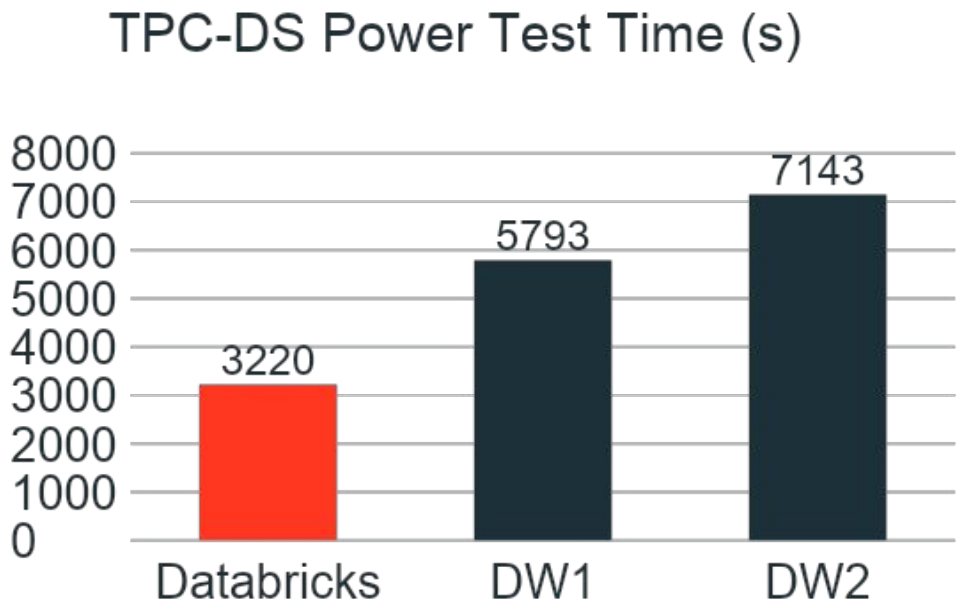
from SQL to machine learning. Traditionally, for the most demanding SQL workloads, enterprises have also moved a curated subset of their data into data warehouses to get high performance, governance and concurrency. However, this two-tier architecture is complex and expensive, as only a subset of data is available in the warehouse, and this data may be out of sync with the raw data due to issues in the extract, transform and load (ETL) process [19].

In response, many organizations are shifting to a data management architecture called the *Lakehouse* [19], which implements data warehouse features such as governance, ACID transactions and rich SQL support directly over a data lake. This single-tier approach promises to simplify data management, as users can govern and query *all* their data in a uniform way, and there are fewer ETL steps and query engines to manage. Recently, new storage layers such as Delta Lake [18] have enabled many of the management features of data warehouses—such as transactions and time travel—on data lakes, and have provided useful tools for optimizing storage access, such as data clustering and data skipping indices. However, maximizing the performance of Lakehouse workloads requires optimizing not only the storage layer, but also query processing.

This paper presents Photon, a new vectorized query engine we [illegible] tabricks for Lakehouse workloads that can exe- [illegible] in Apache Spark's DataFrame [illegible] of queries

Putting These Ideas Together

Lakehouse engines can match DW performance on either hot or cold data!



Databricks Sets Official Data Warehousing Performance Record

by **Reynold Xin** and **Mostafa Mokhtar**
Posted in **COMPANY BLOG** | **November 2, 2021**

Today, we are proud to announce that **Databricks SQL** has set a **new world record in 100TB TPC-DS**, the gold standard performance benchmark for data warehousing. **Databricks SQL outperformed the previous record by 2.2x**. Unlike most other benchmark news, this result has been formally audited and reviewed by the TPC council.

Key Technologies Enabling Lakehouse

1. Metadata layers on data lakes: add transactions, versioning & more

2. Lakehouse engine designs: performant SQL on data lake storage

3. Declarative I/O interfaces for data science & ML

ML over a Data Warehouse is Painful

Unlike SQL workloads, ML workloads need to process large amounts of data with non-SQL code (e.g. TensorFlow, XGBoost)

- SQL over JDBC/ODBC is too slow for this at scale

Export data to a data lake? → adds a third ETL step and more staleness!

Maintain production datasets in both DW & lake? → even more complex

ML over a Lakehouse

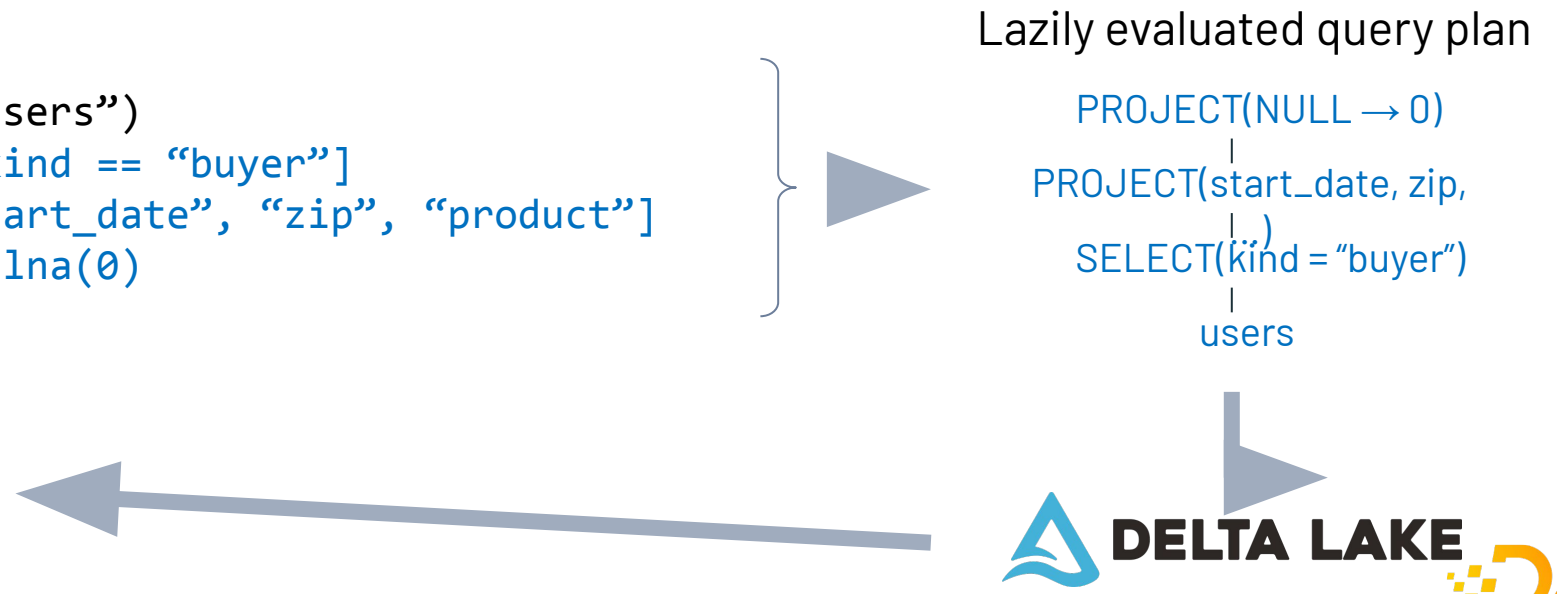
Direct access to data files without overloading the SQL frontend

- ML frameworks already support reading Parquet!
- Declarative APIs such as Spark DataFrames can help optimize queries

```
users = spark.table("users")
buyers = users[users.kind == "buyer"]
train_set = buyers["start_date", "zip", "product"]
                  .fillna(0)

...

model.fit(train_set)
```

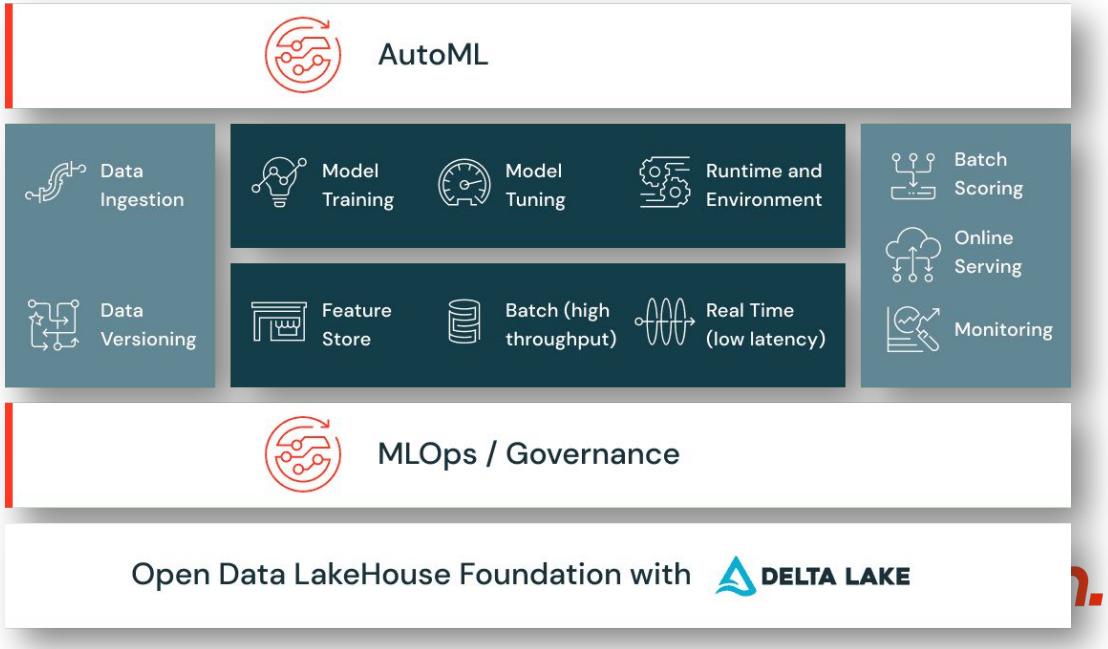


Data-Integrated ML Goes Much Further

Databricks Machine Learning lets data and ML users collaborate:

- ML model metrics become tables thanks to MLflow Tracking
- Feature Store runs Delta for storage and Spark [Streaming] for pipelines
- Models can be used in SQL or ETL jobs

Much simpler than using separate data and ML platforms

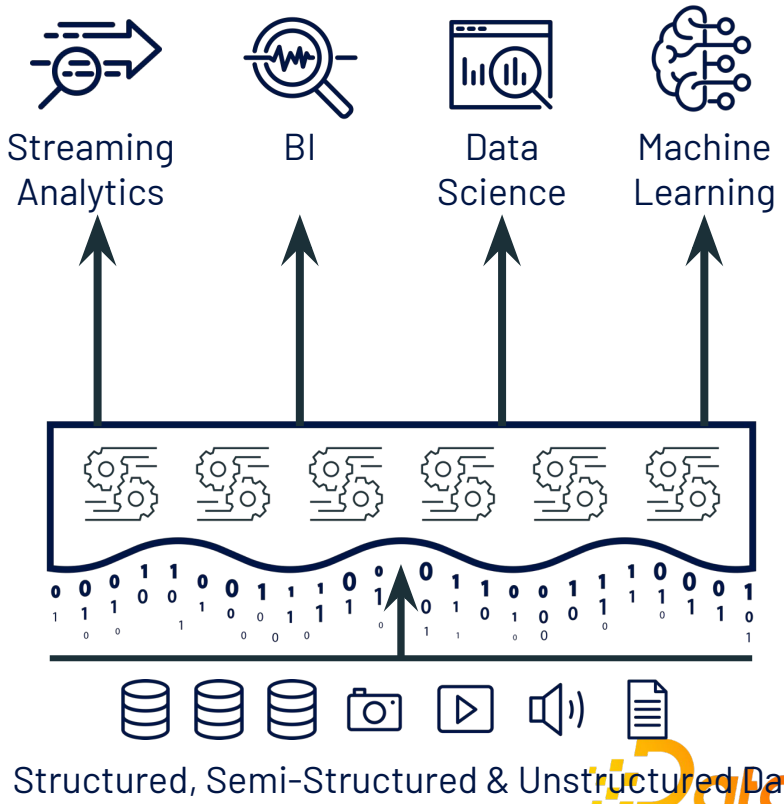


databricks

Summary

Lakehouse systems combine the benefits of data warehouses & lakes

- Open interfaces for direct access from a wide range of tools
- Management features via metadata layers (transactions, versioning, etc)
- Performance via new query engines
- Low cost equal to cloud storage



Result: simplify data architectures to improve **access**, **reliability** & **timeliness**

This Talk

Lakehouse systems: what are they and why now?

Building lakehouse systems

Ongoing projects

We Think There's a Lot More to Do in Data!

Enterprises are just starting to use large-scale data and ML

In five years, there'll be 10-100x more users working with these tools and 10-100x more data and ML applications

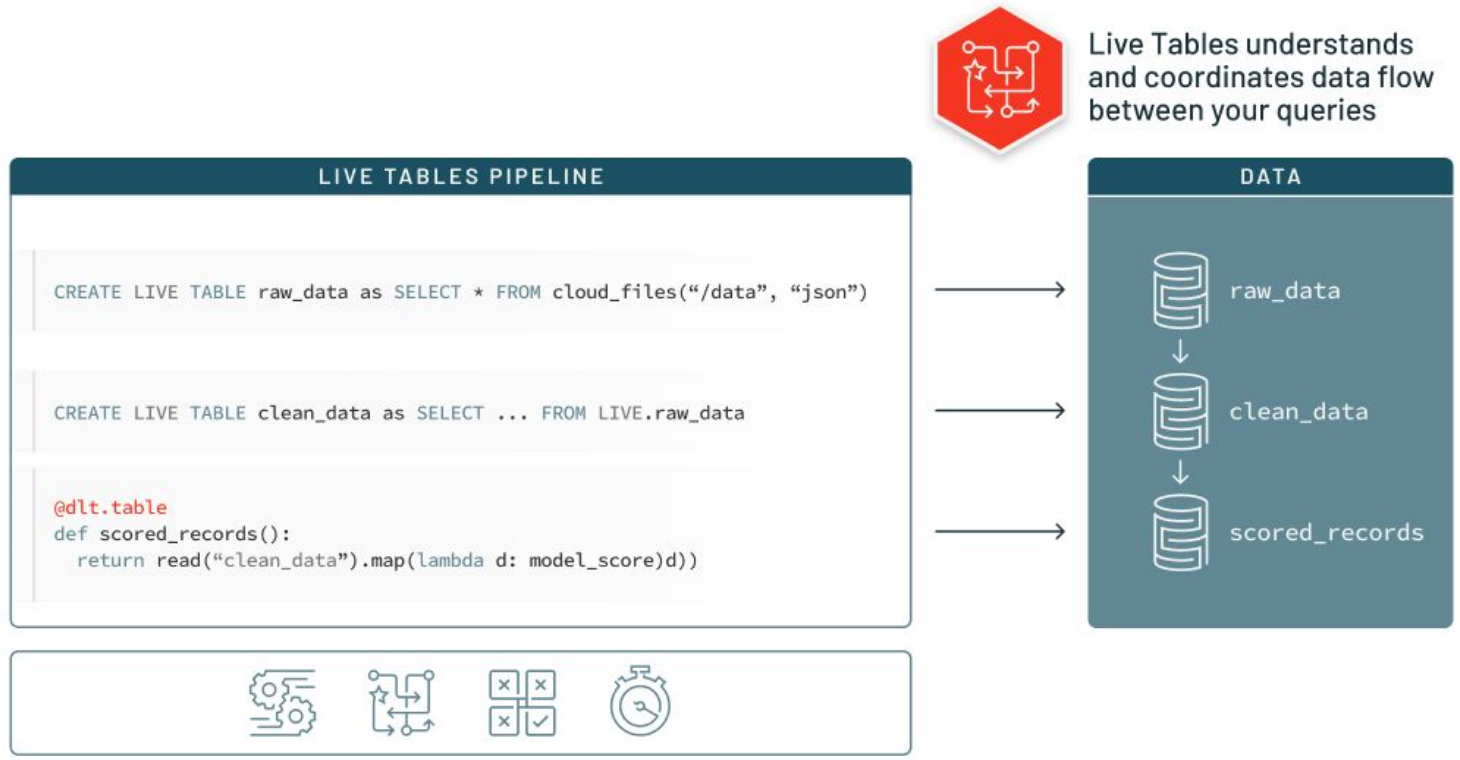
Some ongoing projects: declarative data pipelines (Delta Live Tables), centralized governance (Unity Catalog), and next-gen engine designs

Delta Live Tables: Declarative Data Pipelines

Declarativity was great in SQL, but SQL lives within a larger pipeline (e.g., Airflow tasks)

What if we had a data model of the pipeline's ops and tables?

Analyze cross-task, fork to test, roll back, inject checks, etc



databricks

See Michael Armbrust's blog post and demo

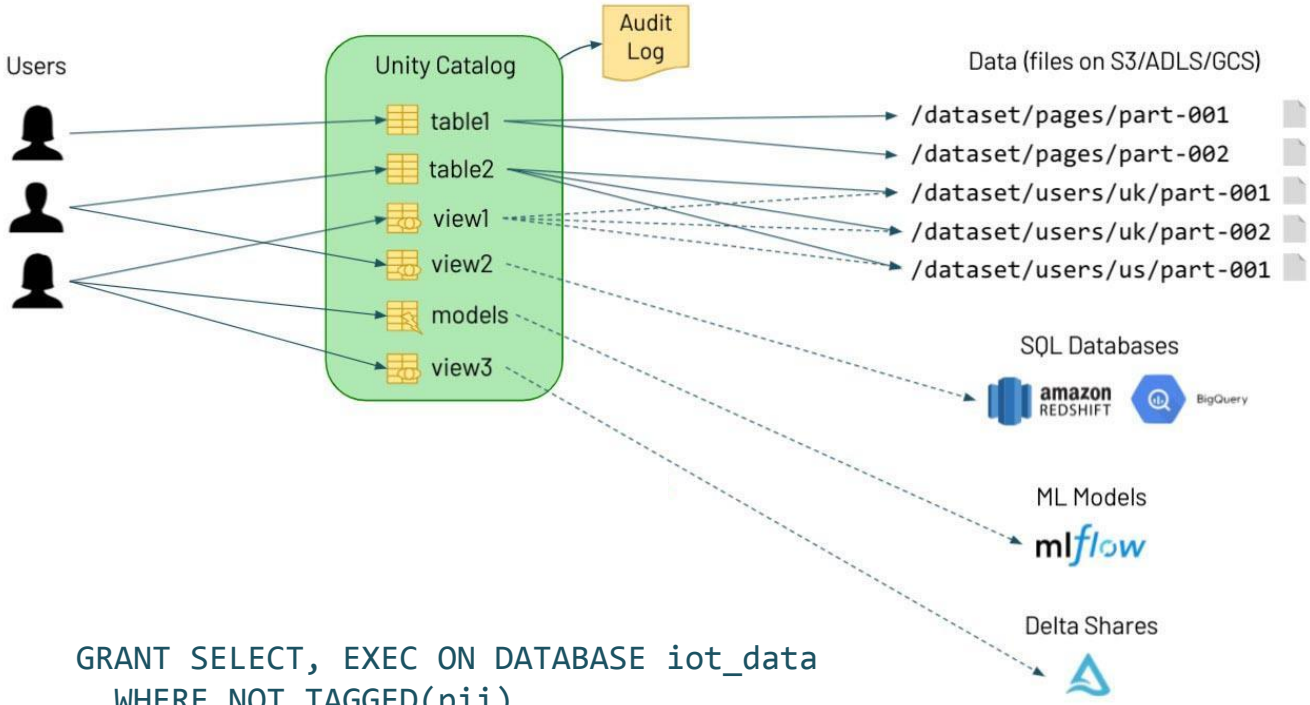
DataFun.

Unity Catalog: Central Governance for Data & ML

Governance requirements for data are rapidly evolving

Unity Catalog provides rich yet efficient access control for millions of data & ML assets

Also gives unified lineage




```
GRANT SELECT, EXEC ON DATABASE iot_data
  WHERE NOT TAGGED(pii)
  TO product_managers
```

databricks

See Matei's blog post

DataFun.

New Engine Projects

Photon: native, vectorized engine for compute operators

Aether: ongoing effort to revamp entire scheduling & exec framework

Streaming: just started a new team to revamp our engine

Conclusion

Databricks tackled one of the key problems orgs have: a simple platform to let diverse users work with *all* their data, in use cases from SQL to ML

There's a lot left to do in this space!


The best data warehouse
is a lakehouse.
databricks
DataFun.