Lakehouse: Supporting Modern Data and AI Workloads

Reynold Xin @rxin
Jan 31, 2022, Berkeley cs291-162

whoami

- Reynold Xin
- 2010 – 2013: PhD in databases @ UC Berkeley
- 2013 – 2016: Spark (query engine) development @ Databricks
  - API revamp, e.g. DataFrames
  - Engine rewrites
  - Performance efforts, e.g. Sort Benchmark 2014 and current (2016) world record
- 2016 – present: Lakehouse @ Databricks

The Times, They Are a-Changing...

All enterprises are starting to use large-scale data (petabytes+)

All enterprises are using machine learning

Computing is moving to the cloud

Great opportunity for new data systems!

About databricks

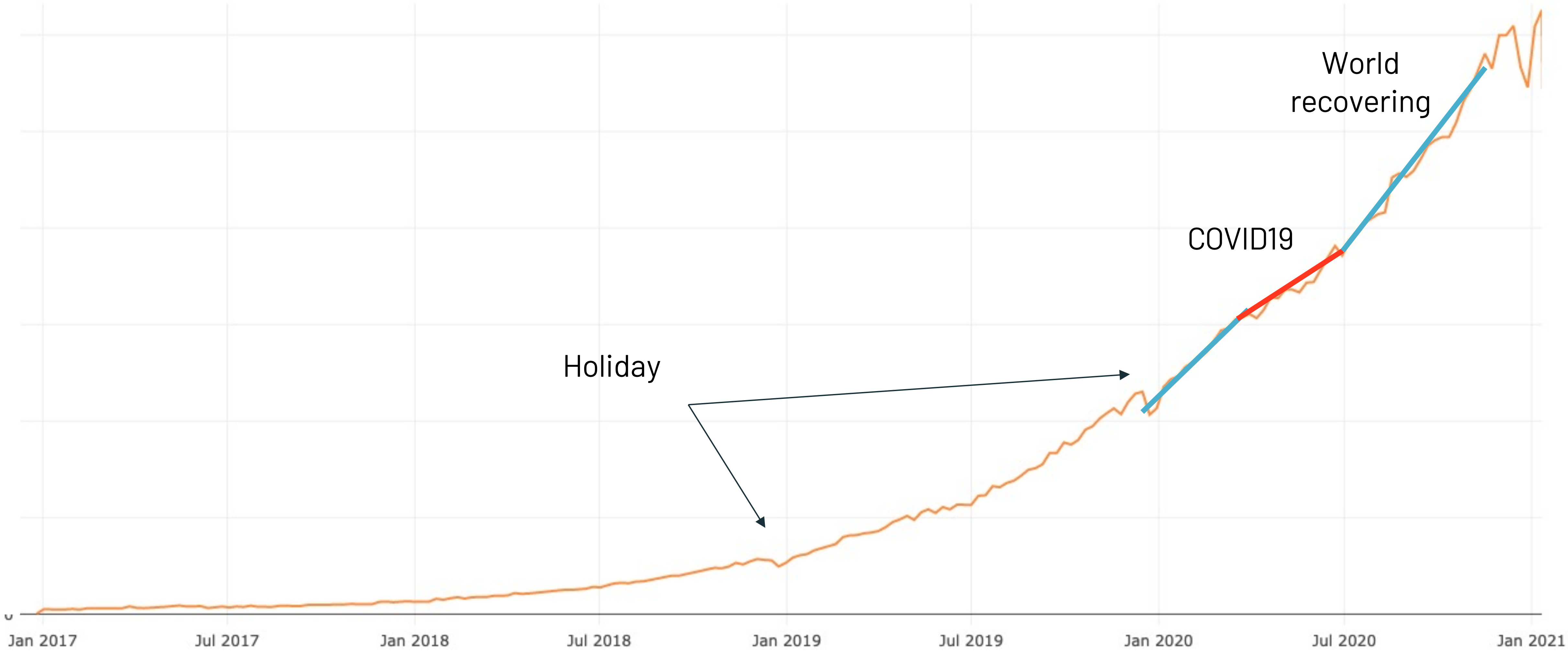
Cloud-based data and ML platform for over 5000 customers

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

Approximately 800 engineers

Used for ETL, data science, ML and data warehousing

Compute resources growth



databricks

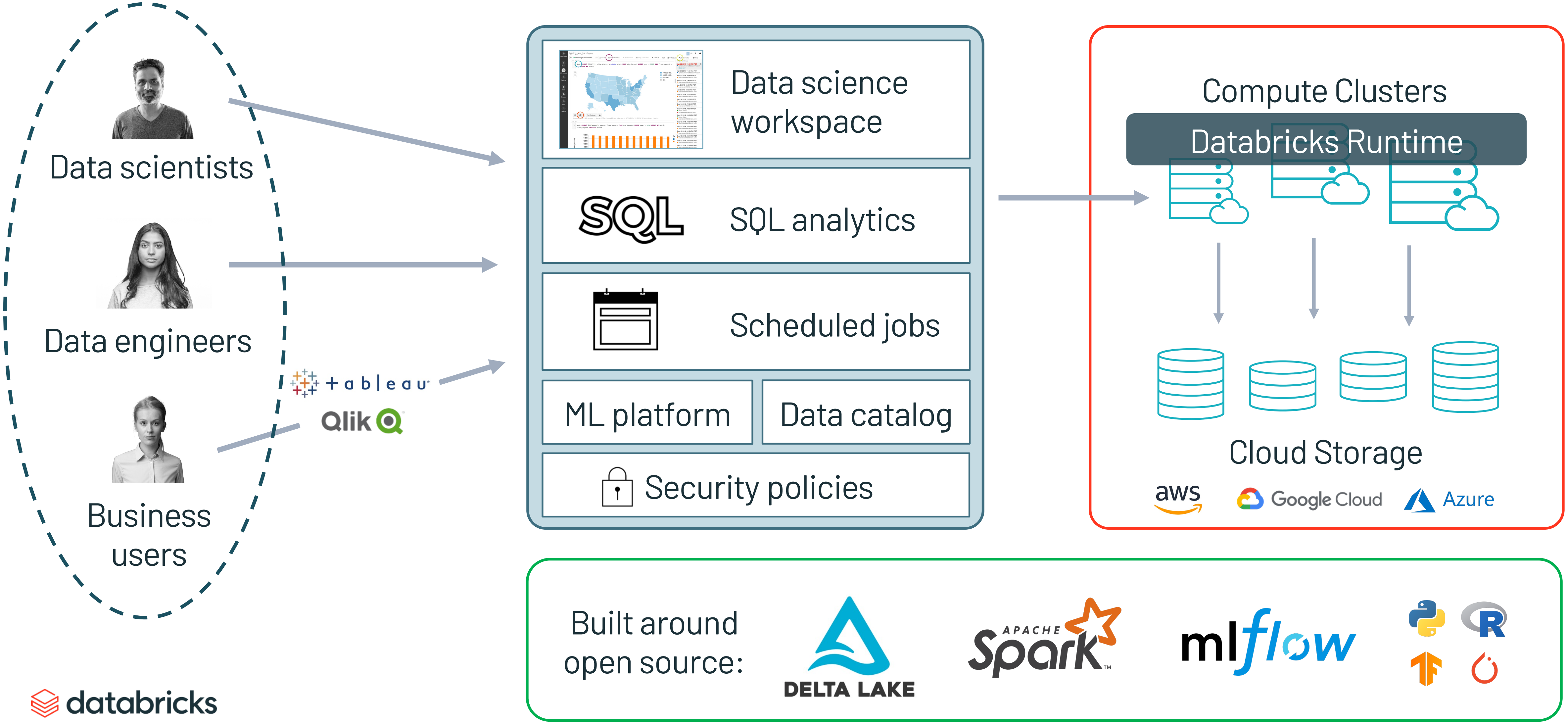
What's Unique About Databricks?

New system architecture, the **lakehouse**, that combines the best features of data lakes and warehouses

Cloud-first: embrace elasticity and scale

Diverse users: integrated platform from BI analysts to ML engineers

Our Platform



databricks

Example Use Cases

Shell — Optimize production using ML and BI on petabyte-scale data

HSBC — Manage and query 170 PB of data that used to be in 14 databases

REGENERON — Correlate 500,000 patient records with DNA to design therapies

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?

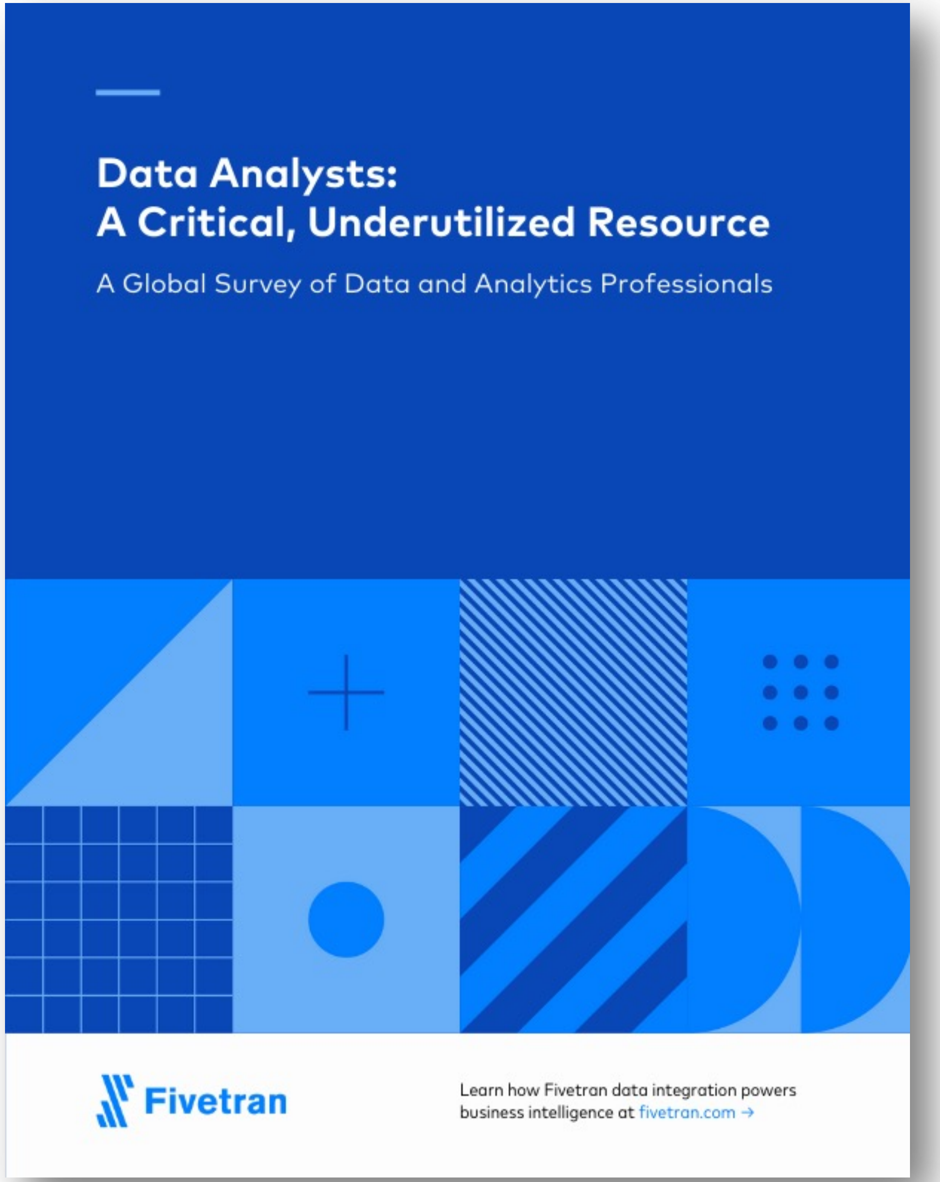
Without great data, you can't do any analysis!

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

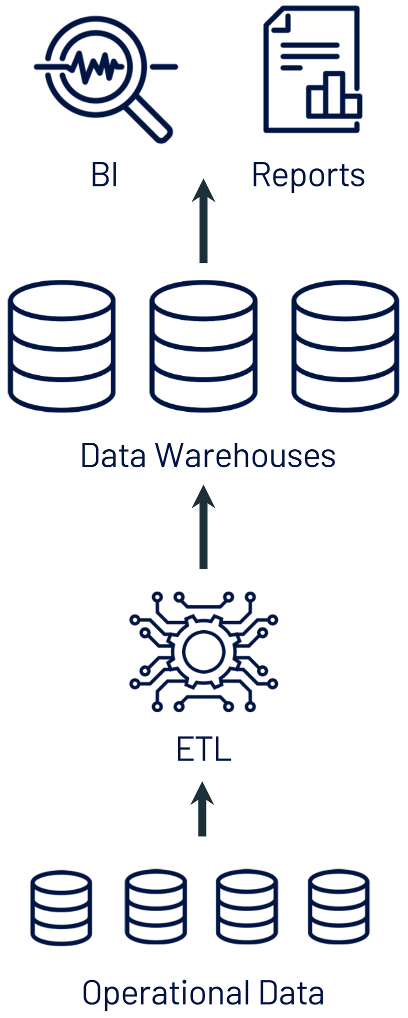
90% regularly had unreliable data sources



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

1980s: Data Warehouses

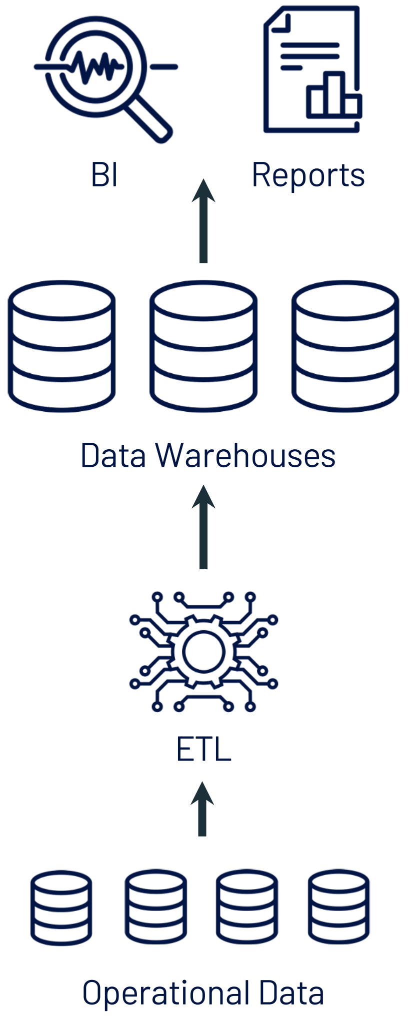
- ETL data directly from operational database systems
- Rich management and performance features for SQL analytics: schemas, indexes, transactions, etc



databricks

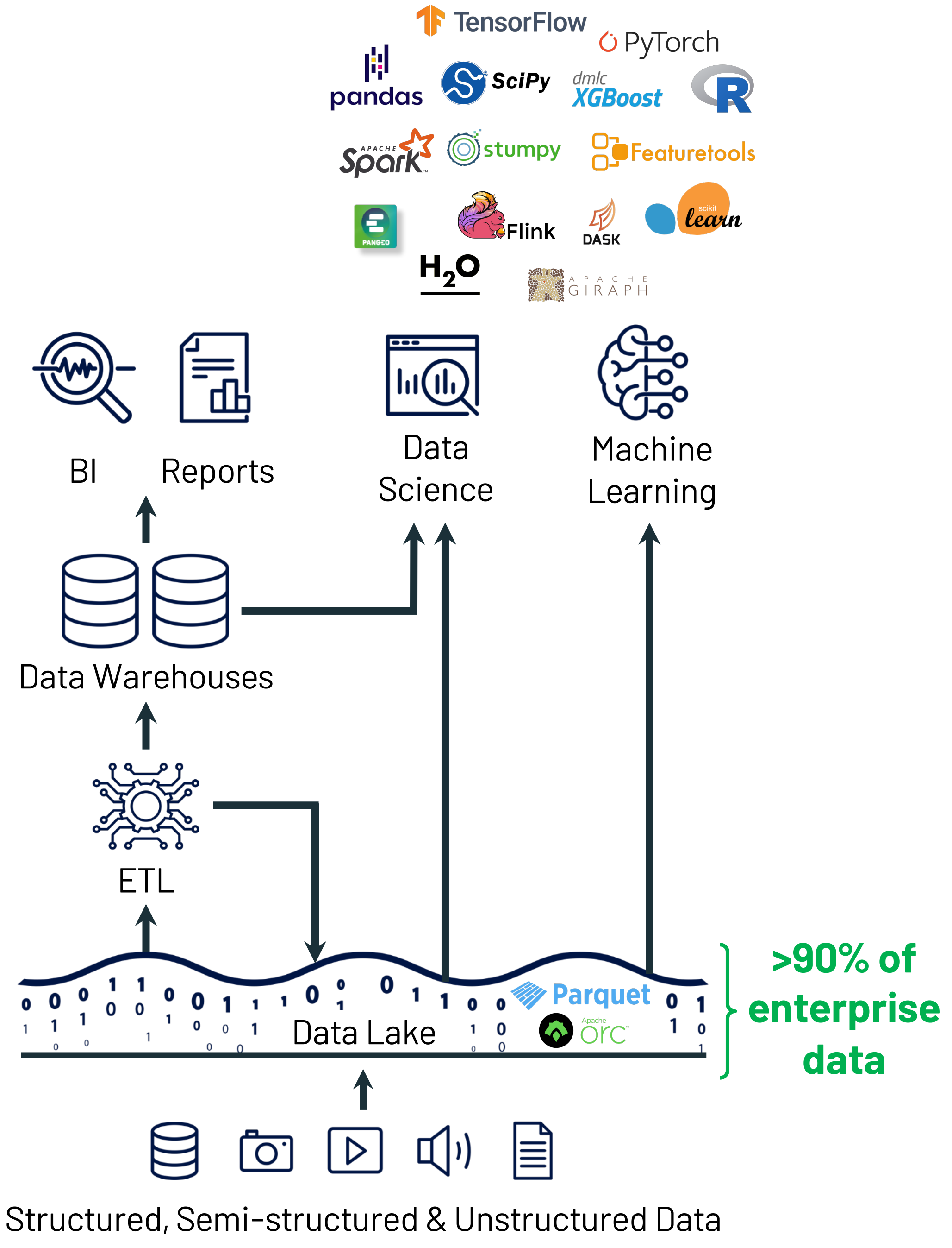
2010s: New Problems for Data Warehouses

- 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



Problems with Today's Architectures

Cheap to store all the data, but the 2-tier architecture is much more complex!

Data reliability suffers:

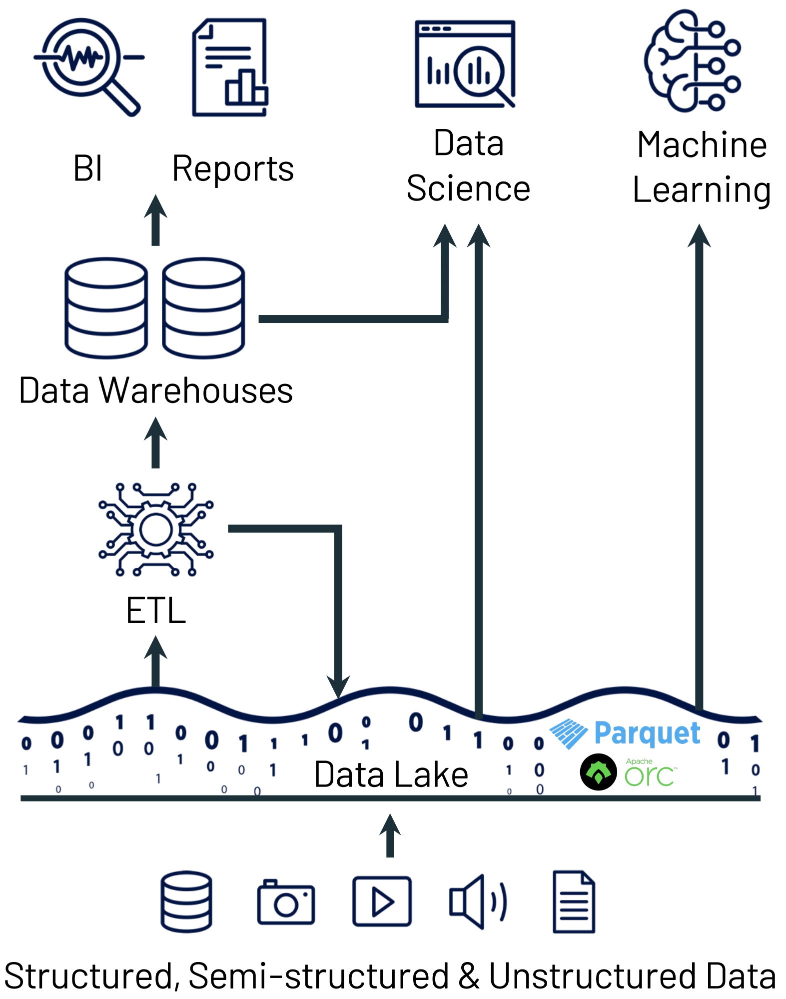
- Multiple storage systems with different semantics, SQL dialects, etc
- Extra ETL steps that can go wrong

Timeliness suffers:

- Extra ETL steps before data available in DW

High cost:

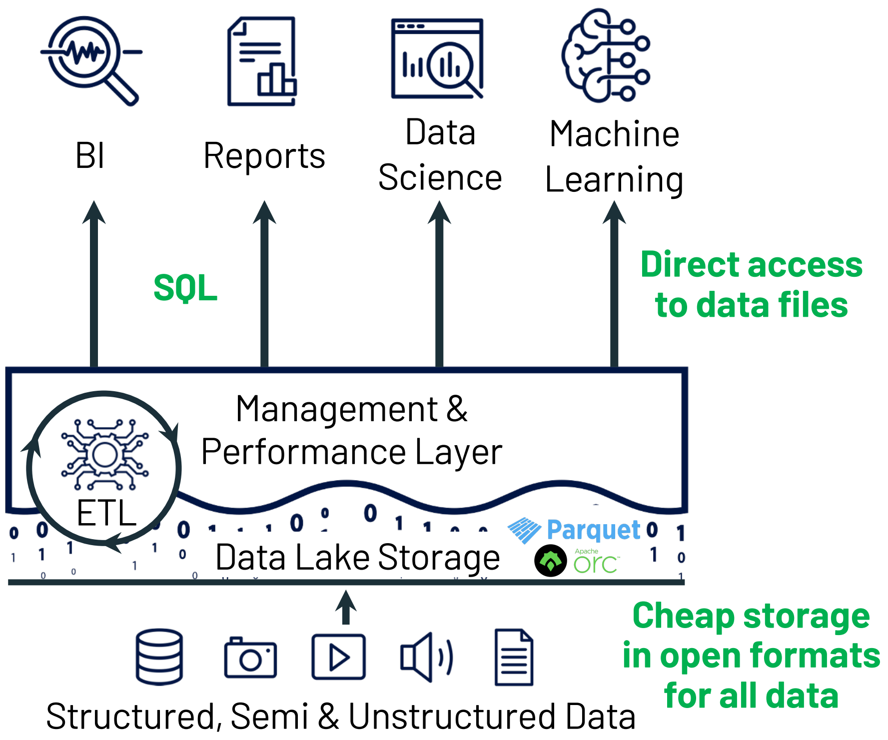
- Continuous ETL, duplicated storage



databricks

Lakehouse Systems

Implement data warehouse management and performance features on top of **directly-accessible data in open formats**



Can we get state-of-the-art performance & governance features with this design?

This Talk

Lakehouse systems: what are they and why now?

Building lakehouse systems

Ongoing projects

Lakehouse Technology

New techniques to provide data warehousing features directly on data lake storage

- Retain existing open file formats (e.g. Apache Parquet, ORC)
- Add management and performance features on top (transactions, data versioning, indexes, etc)
- Can also help eliminate other data systems, e.g. message queues

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

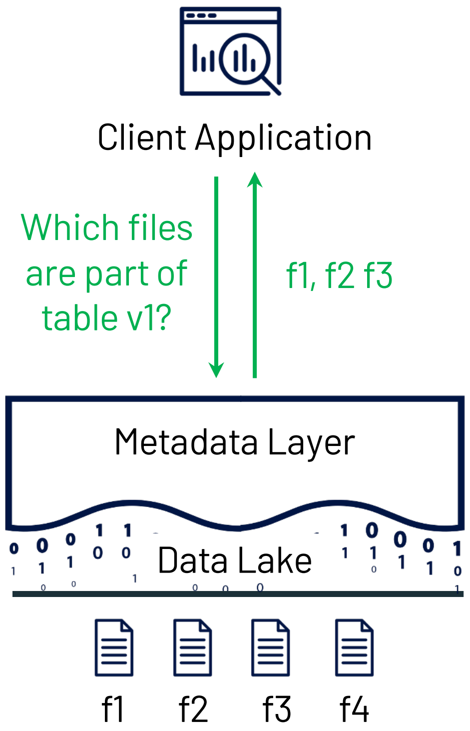
- Track **which files** are part of a table version to offer rich management features like transactions
  - Clients can then access the underlying files at high speed

- Examples:

DELTA LAKE

ICEBERG

HIVE ACID



Example: Traditional Data Lake

Query: delete all events data about customer #17

"events" table

"events" table			
file1.parquet	rewrite →	file1b.parquet	+ delete file1.parquet
file2.parquet			
file3.parquet	rewrite →	file3b.parquet	+ delete file3.parquet

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

Example: DELTA LAKE

"events" table

Query: delete all events data about customer #17

file1.parquet —rewrite→ file1b.parquet

file2.parquet

file3.parquet —rewrite→ file3b.parquet

_delta_log / v1.parquet
/ v2.parquet

$\underbrace{\text{/ v1.parquet, / v2.parquet}}$ track which files are part of each version of the table (e.g. v2 = file1, file2, file3)

atomically add new log file
_delta_log / v3.parquet

$\underbrace{\text{v3.parquet}}$ v3 = file1b, file2, file3b

Clients always read a consistent table version!

Armbrust et al, VLDB 2020

Other Management Features with DELTA LAKE

- Time travel to old table versions

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

- Zero-copy CLONE by forking the log

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

- DESCRIBE HISTORY

```
DESCRIBE HISTORY flightdelays
```

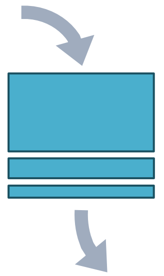
(1) Spark Jobs

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"}

- Schema enforcement & constraints

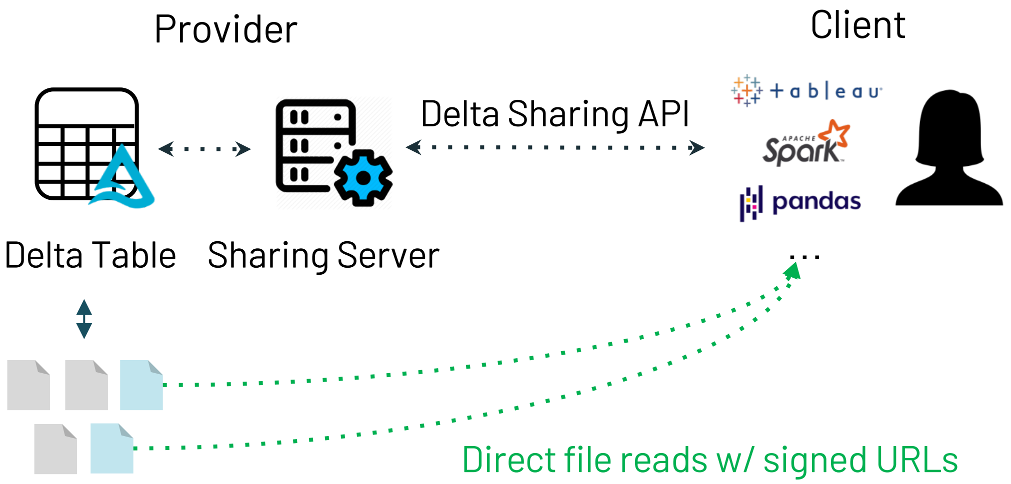
Other Management Features with DELTA LAKE

- 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



DELTA LAKE Adoption

Already >50% of Databricks workload

Broad industry support

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

file1.parquet	year: min 2018, max 2019 uid: min 12000, max 23000
file2.parquet	year: min 2018, max 2020 uid: min 12000, max 14000
file3.parquet	year: min 2020, max 2020 uid: min 23000, max 25000

$\underbrace{\hphantom{xxxxxxxxxxxxxxxxxxxx}}_{\text{updated transactionally with Delta table log}}$

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

file1.parquet	year: min 2018, max 2019 uid: min 12000, max 23000
file2.parquet	year: min 2018, max 2020 uid: min 12000, max 14000
file3.parquet	year: min 2020, max 2020 uid: min 23000, max 25000

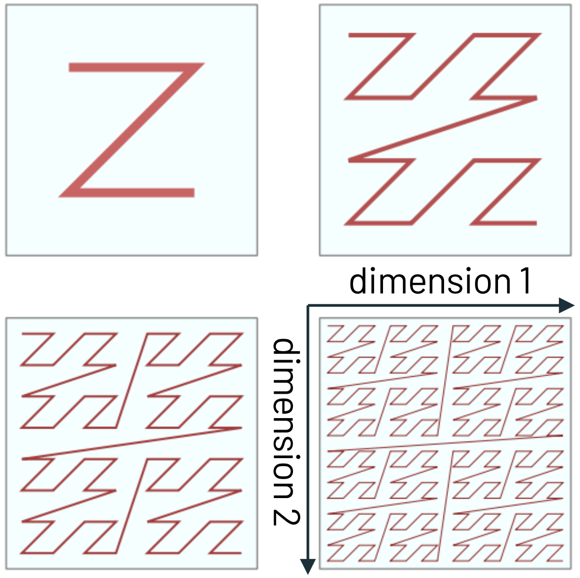
updated transactionally
with Delta table log

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

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

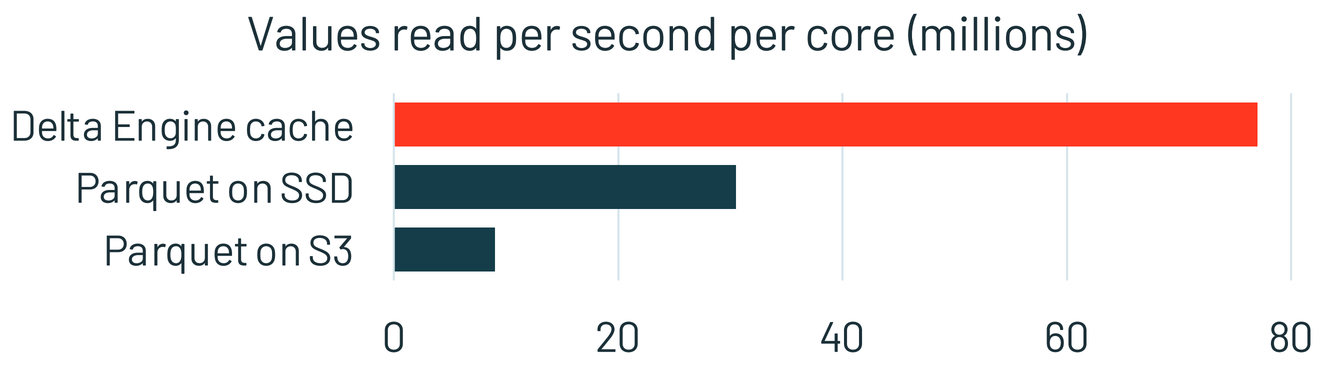
- **Example:** Z-order sorting for multi-dimensional clustering



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

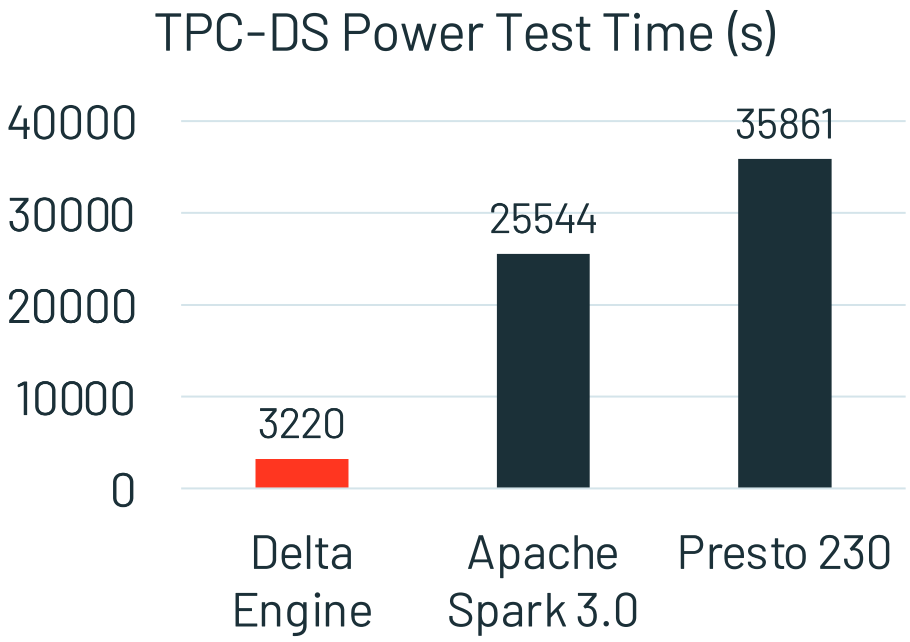
- **Example:** SSD cache in Photon Engine



Optimization 4: Vectorized Execution

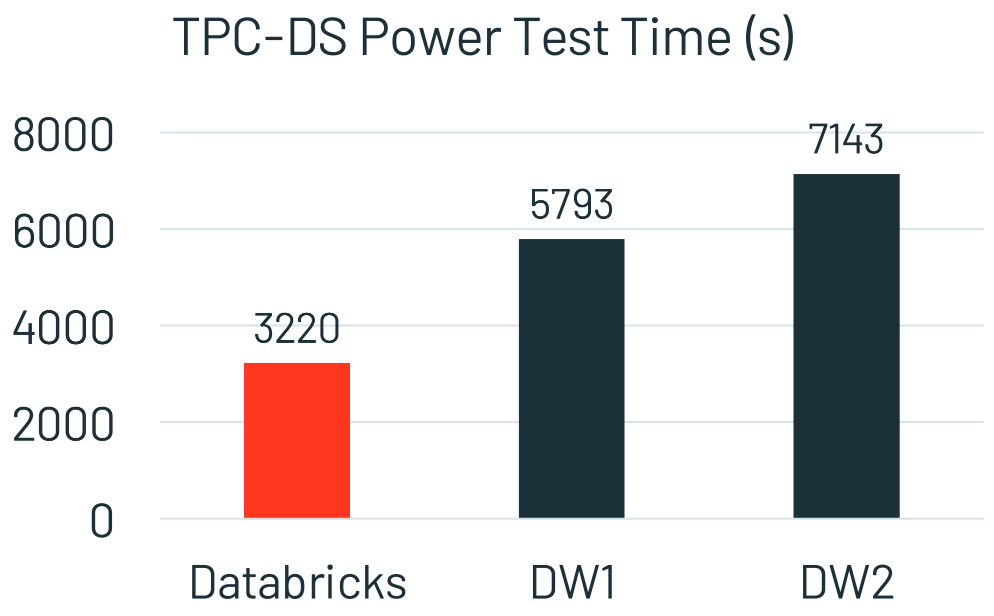
- Many existing ideas can also be applied over open formats like Parquet

- **Example:** Databricks Photon vectorized engine



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.

Unique Challenges in Lakehouse

- Statistics are not always known or up-to-date: use adaptive query execution to replan at runtime (added in Apache Spark 3.0)
- Data is less processed, e.g., using strings instead of IDs: optimize the pathways for strings and semi-structured data
- Unstructured data are large and unpredictable: design the engine to tolerate large (>1 GB) fields and to carefully manage memory

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

databricks

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)
```

Lazily evaluated query plan

PROJECT(NULL → 0)

PROJECT(start_date, zip, ...)

SELECT(kind = "buyer")

users

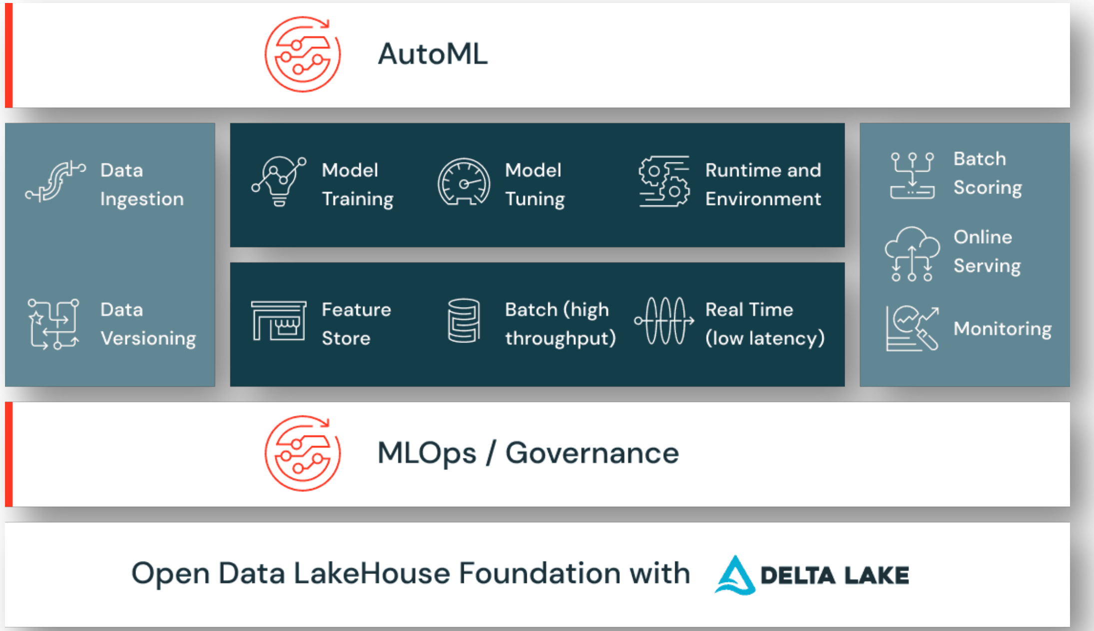
DELTA LAKE

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

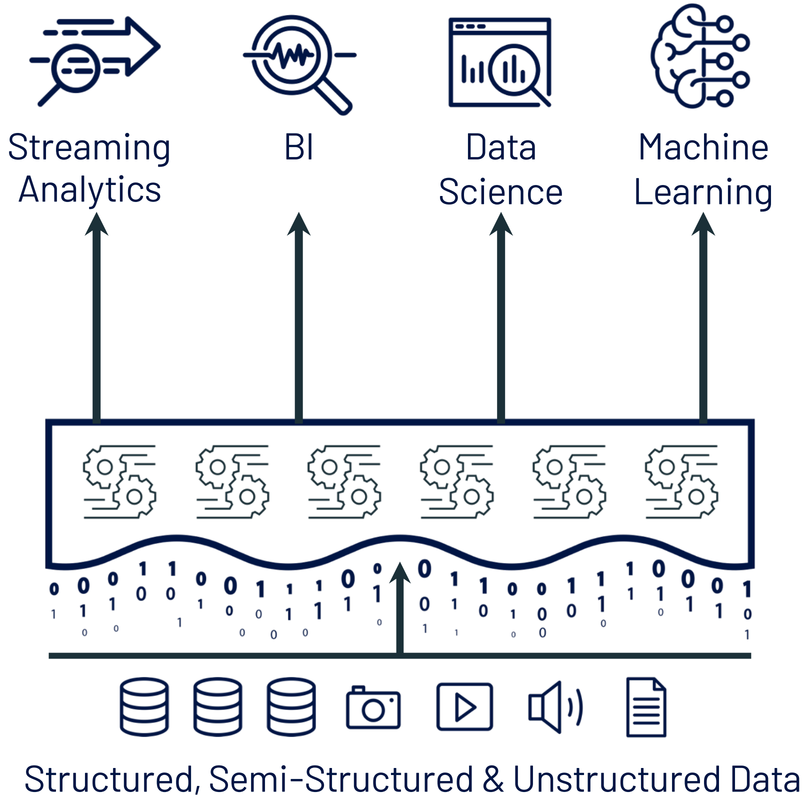
Much simpler than using separate data and ML platforms



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

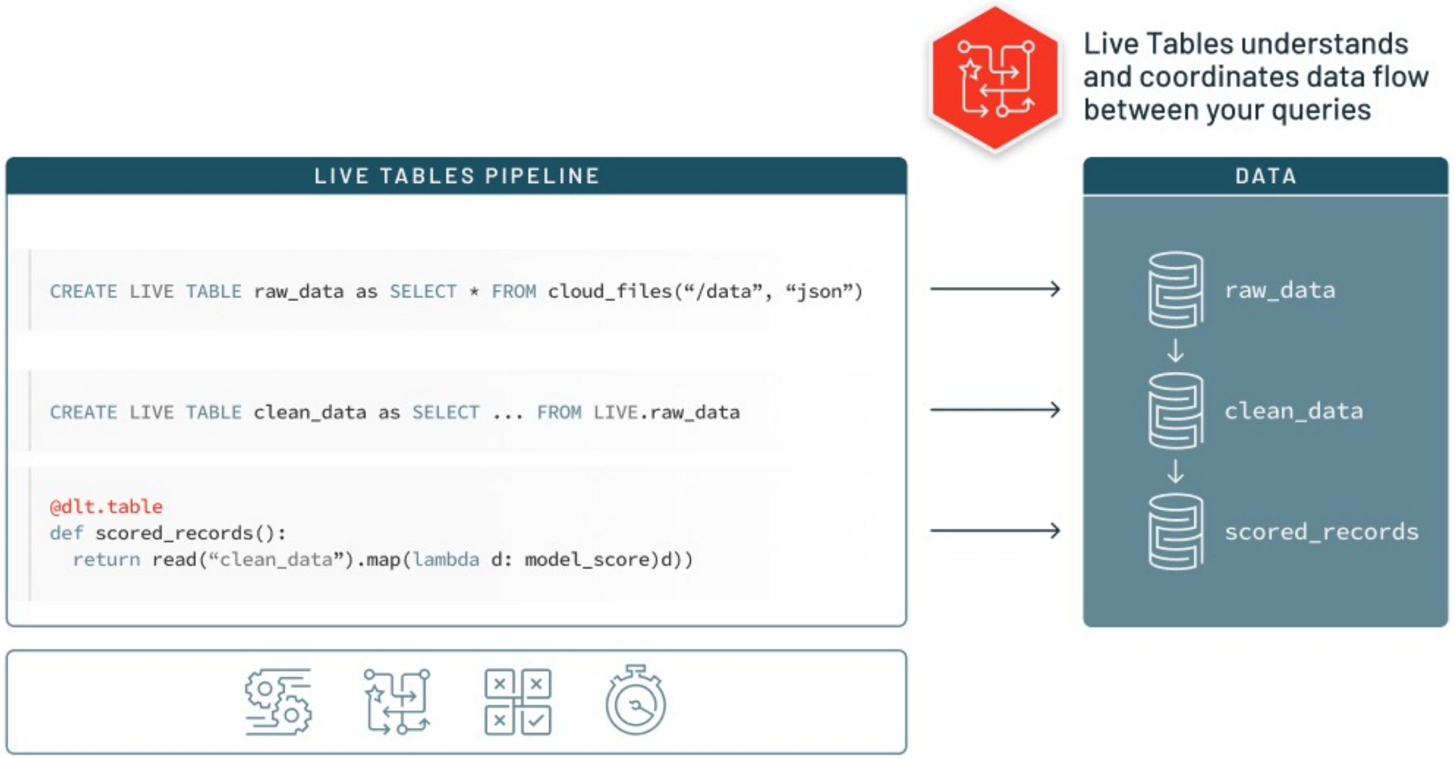
Some ongoing projects: declarative data pipelines (Delta Live Tables), centralized governance (Unity Catalog), and next-gen engine designs

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



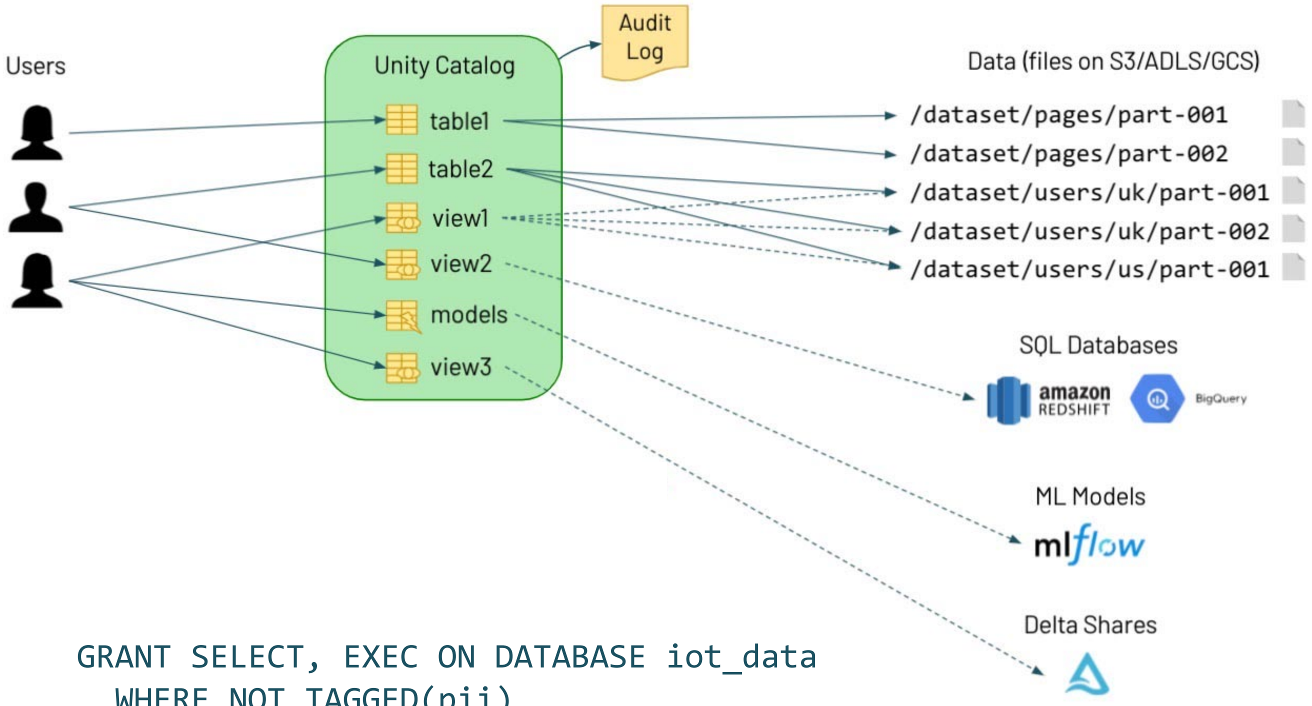
See Michael Armbrust's blog post and demo

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
```


See [Matei's blog post]

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!

Questions?