Lakehouse Technology as the Future of Data Warehousing

Reynold Xin @rxin
Mar 16, 2021, Stanford CS245

databricks

whoami

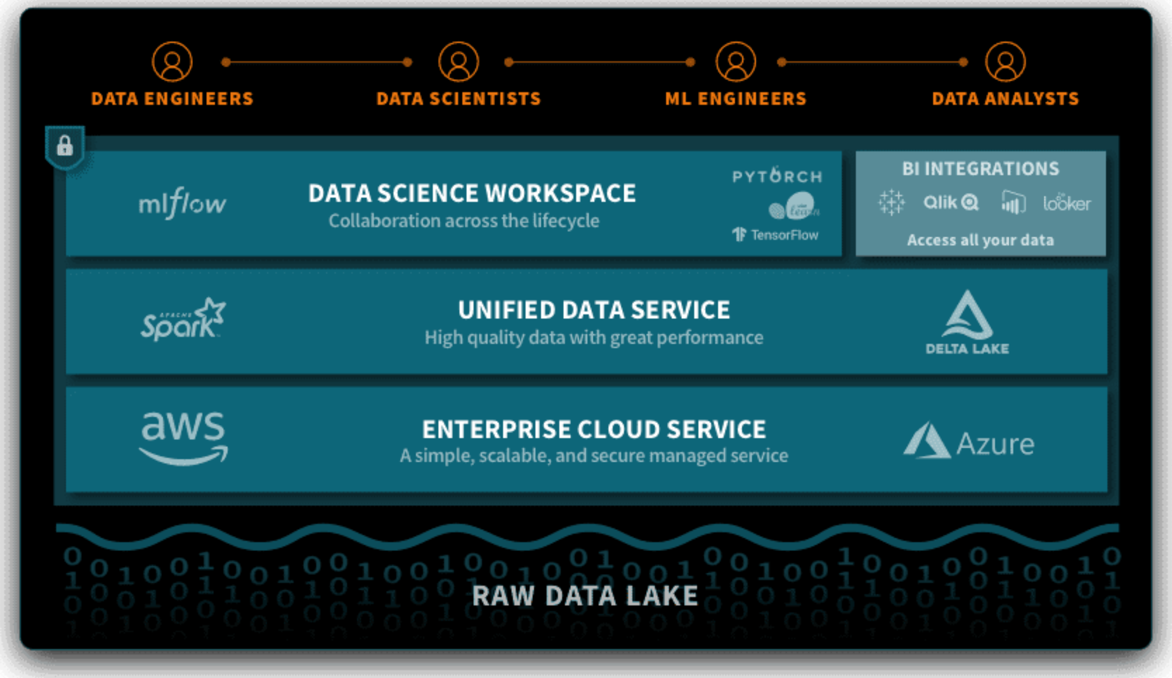
- Reynold Xin
- 2010 – 2013: PhD in databases @ UC Berkeley
- 2013 – 2016: Spark (query engine) development @ Databricks
  - API revamp, e.g. DataFrames
  - Engine rewrites
  - Performance efforts, e.g. Sort Benchmark 2014 and current (2016) world record
- 2016 – present: Lakehouse @ Databricks

About databricks

Cloud data platform for analytics, engineering and data science

Runs a fleet of millions of VMs to process exabytes of data/day

>5000 enterprise customers



DATA ENGINEERS
DATA SCIENTISTS
ML ENGINEERS
DATA ANALYSTS

DATA SCIENCE WORKSPACE
Collaboration across the lifecycle

BI INTEGRATIONS
Access all your data

UNIFIED DATA SERVICE
High quality data with great performance

ENTERPRISE CLOUD SERVICE
A simple, scalable, and secure managed service

RAW DATA LAKE

Talk Outline

- Many problems with data analytics today stem from the complex data architectures we use

- New “Lakehouse” technologies can remove this complexity by enabling fast data warehousing, streaming & ML directly on data lake storage

databricks

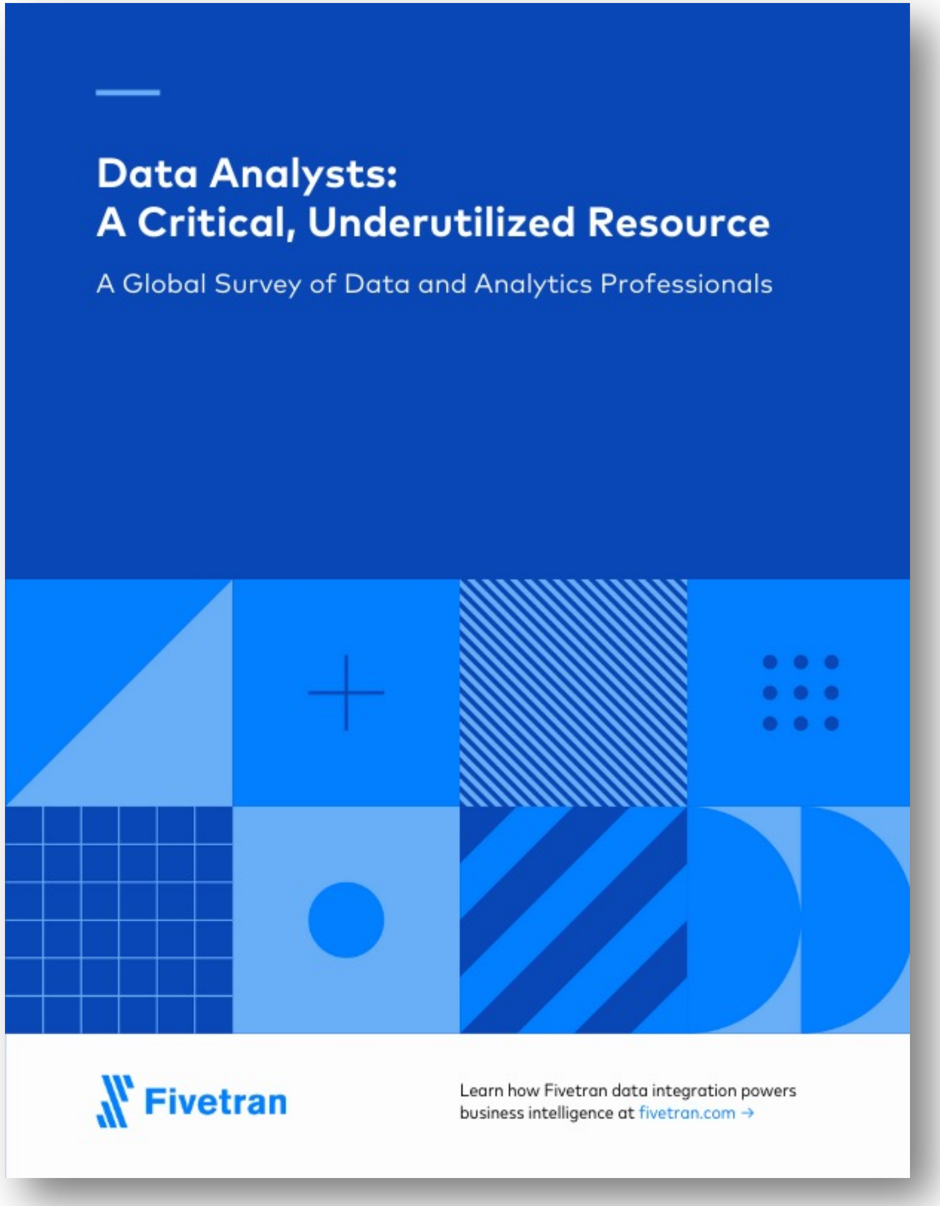
The biggest challenges with data today: **data quality** and **staleness**

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



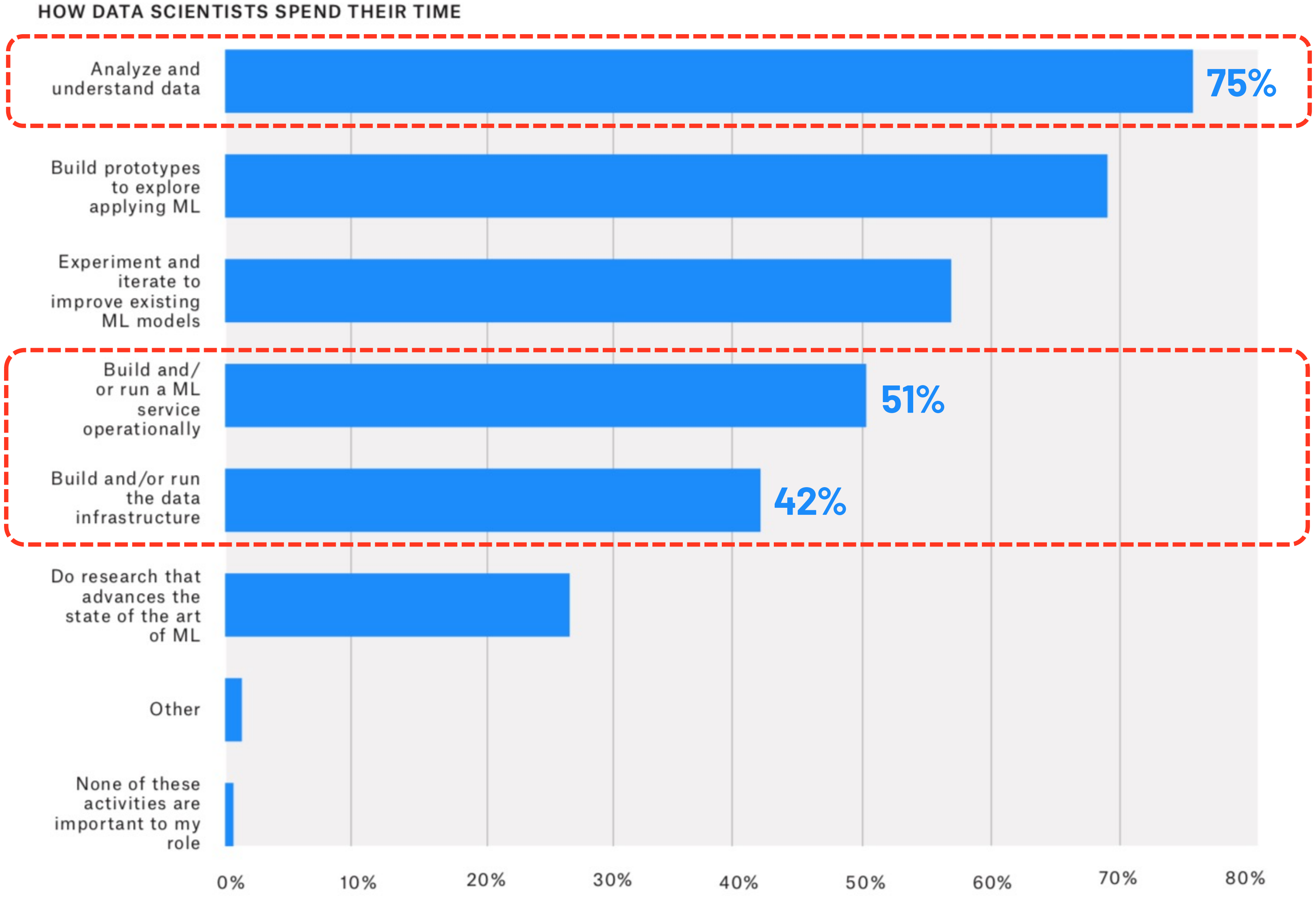
Data Analysts:
A Critical, Underutilized Resource

A Global Survey of Data and Analytics Professionals

Fivetran

Learn how Fivetran data integration powers business intelligence at fivetran.com →

kaggle Data Scientist Survey



databricks

Getting high-quality, timely data is hard...

but it’s partly a problem of our own making!

The Evolution of Data Management

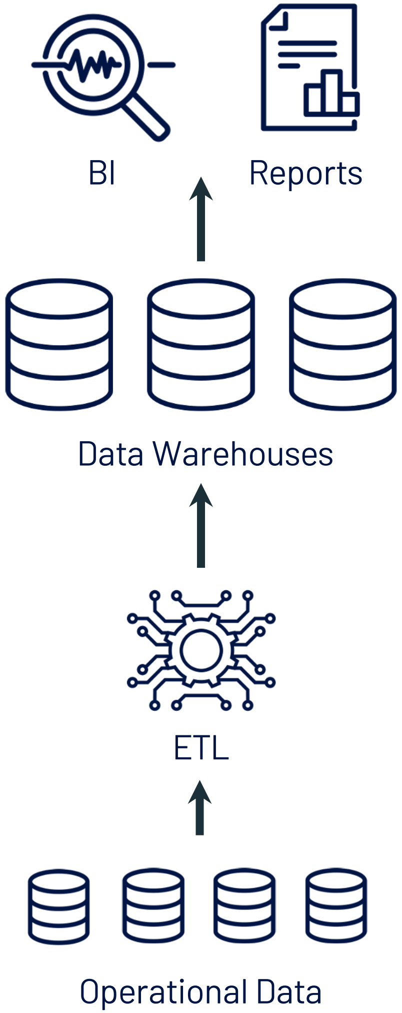
databricks

1980s: Data Warehouses

- ETL data directly from operational database systems

- Purpose-built for SQL analytics & BI: schemas, indexes, caching, etc

- Powerful management features such as ACID transactions and time travel

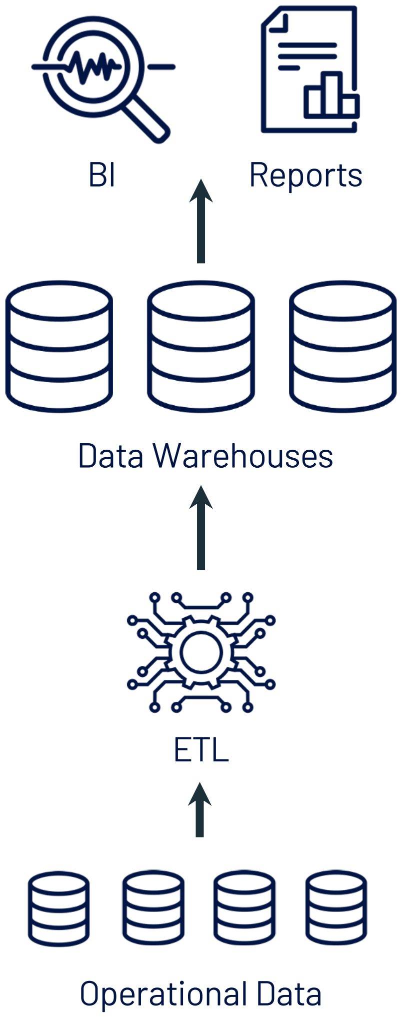


2010s: New Problems for Data Warehouses

- Could not support rapidly growing unstructured and semi-structured data: time series, logs, images, documents, etc

- High cost to store large datasets

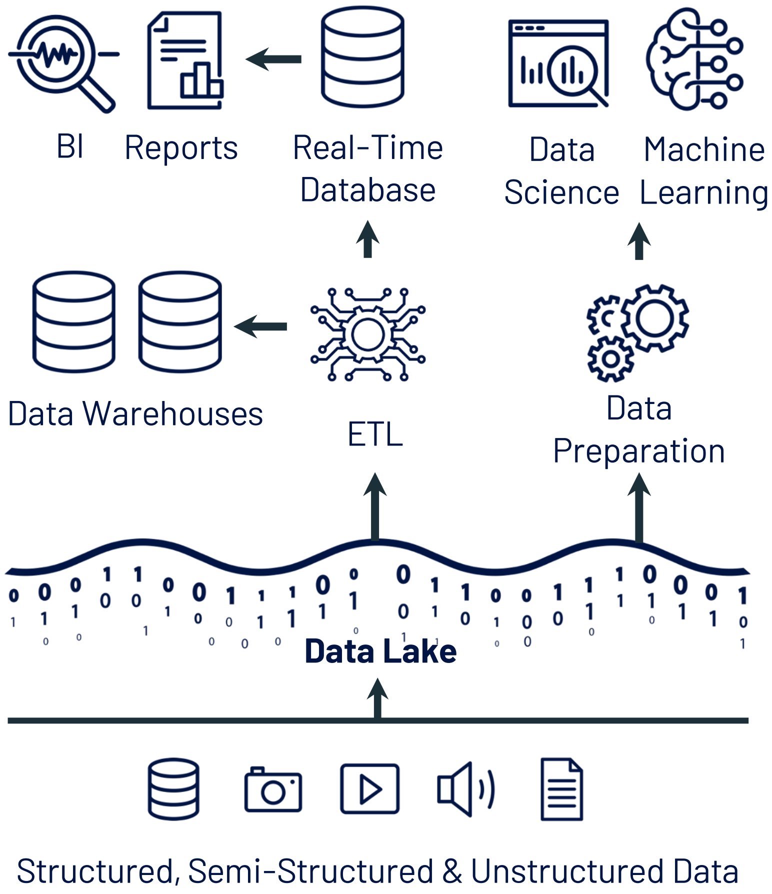
- No support for data science & ML



databricks

2010s: Data Lakes

- Low-cost storage to hold *all* raw data (e.g. Amazon S3, HDFS)
  - $12/TB/month for S3 infrequent tier!
- ETL jobs then load specific data into warehouses, possibly for further ELT
- Directly readable in ML libraries (e.g. TensorFlow) due to open file format



Problems with Today's Data Lakes

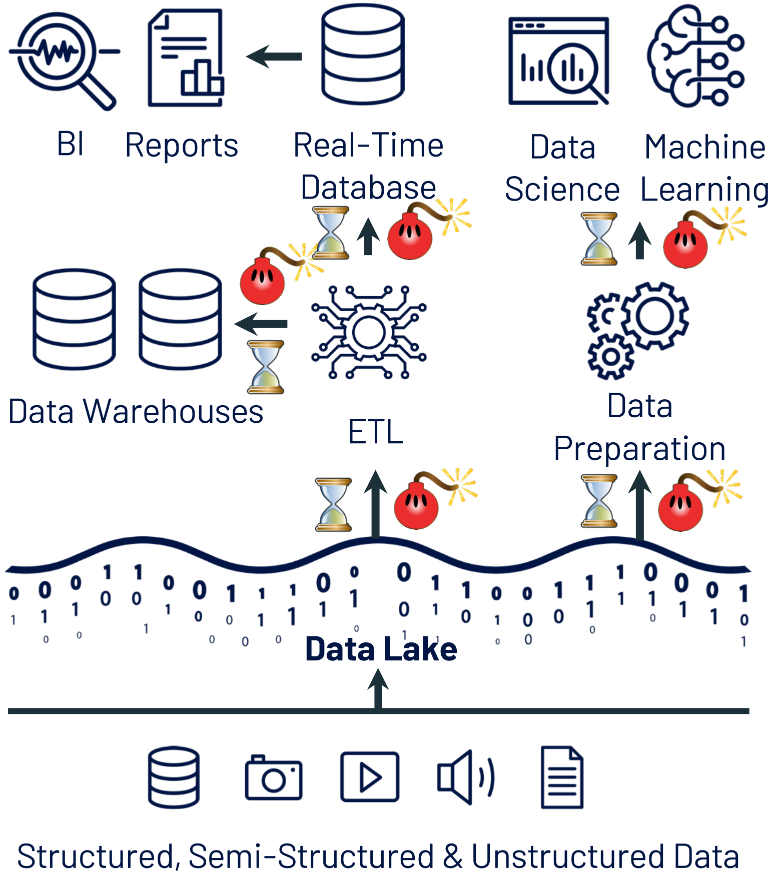
Cheap to store all the data, but system architecture is much more complex!

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 is available in data warehouses



Problems with Today's Data Lakes

Cheap to store all the data, but system 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 is available in data warehouses

Summary

At least some of the problems in modern data architectures are due to unnecessary system complexity

- We wanted low-cost storage for large historical data, but we designed separate storage systems (data lakes) for that
- Now we need to sync data across systems all the time!

What if we didn’t need to have all these different data systems?

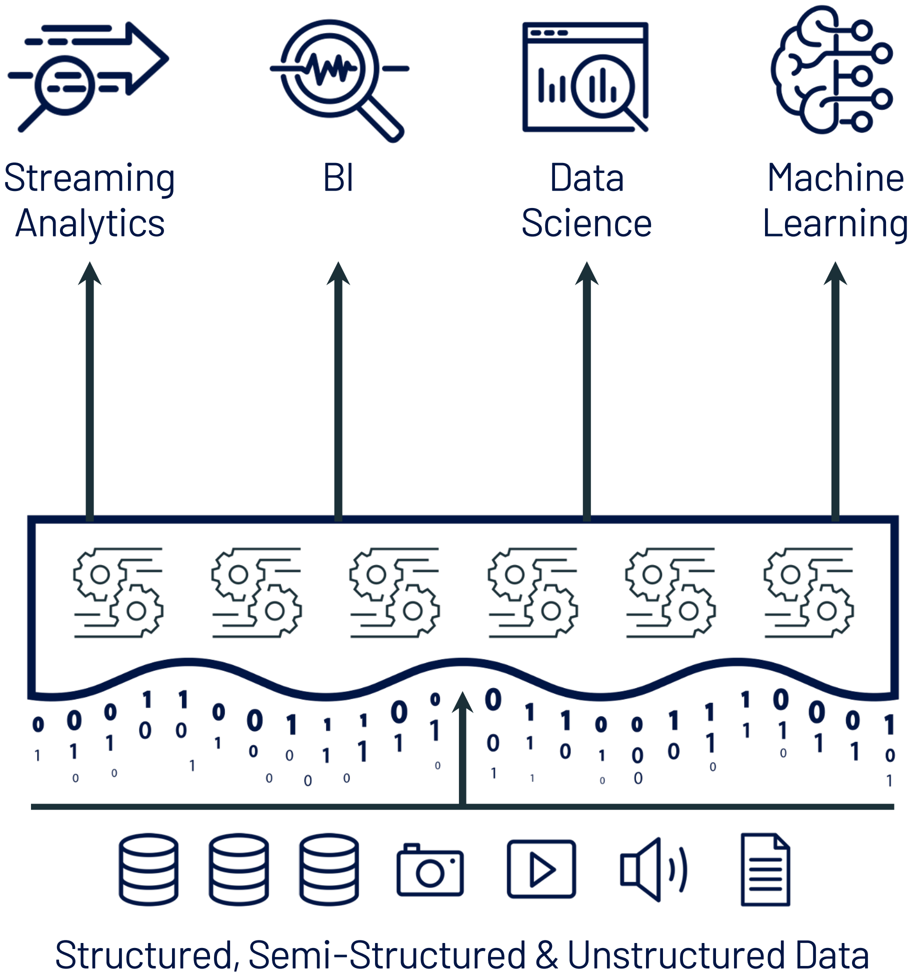
Lakehouse Technology

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 parts: metadata layers such as Delta Lake (from Databricks) and Apache Iceberg (from Netflix) + new engine designs

Lakehouse Vision



Single platform for every use case

Management features (transactions, versioning, etc)

Data lake storage for all data

Key Technologies Enabling Lakehouse

1. Storage layer: add transactions, versioning & more

2. New query engine designs: great SQL performance on data lake storage systems and file formats

3. Optimized access for data science & ML

Key Technologies Enabling Lakehouse

1. Metadata layers for data lakes: add transactions, versioning & more
2. New query engine designs: great SQL performance on data lake storage systems and file formats
3. Optimized access for data science & ML

Metadata Layers for Data Lakes

- A data lake is normally just a collection of files
- Metadata layers keep track of *which files* are part of a table to enable richer management features such as transactions
  - Clients can then still access the underlying files at high speed
- Implemented in multiple systems:

DELTA LAKE

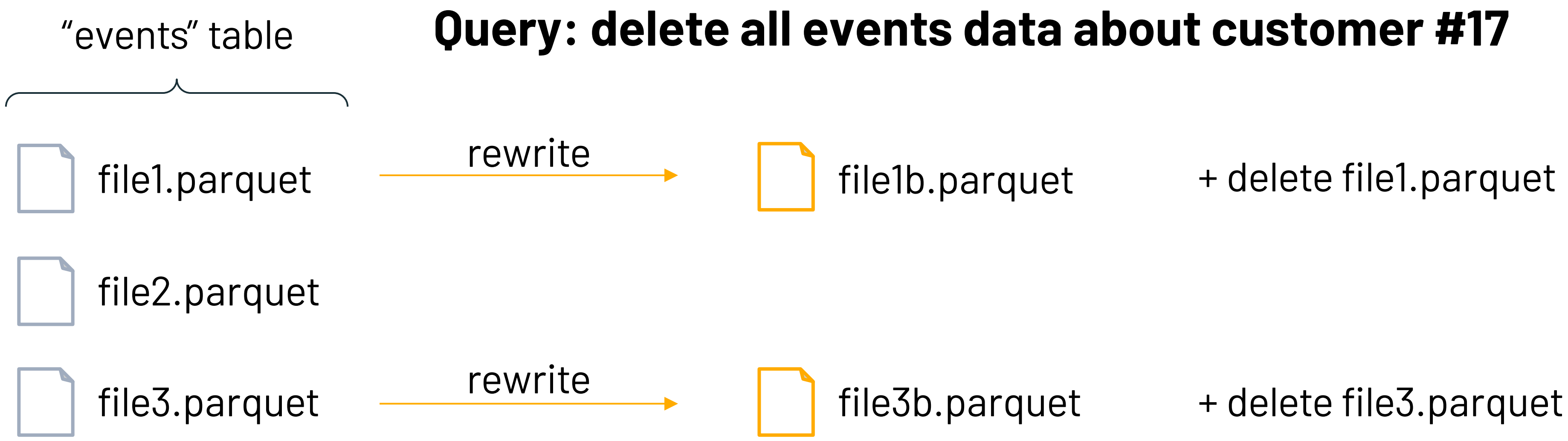
ICEBERG

HIVE ACID

Keep metadata in the object store itself

Keep metadata in a database

Example: Basic Data Lake



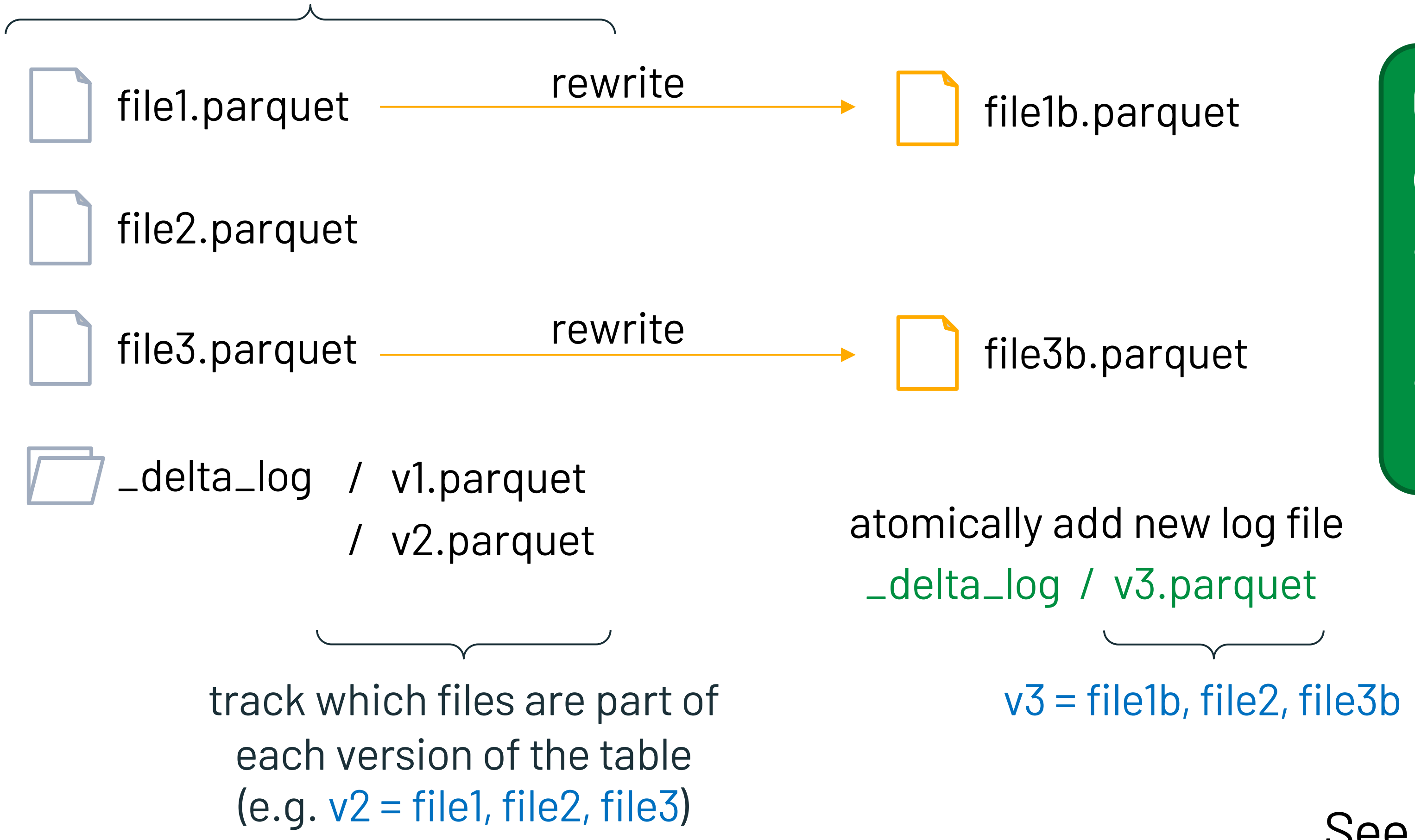
Problem: What if a query reads the table while the delete is running?

Problem: What if the query doing the delete fails partway through?

Example with DELTA LAKE

Query: delete all events data about customer #17

“events” table



file1.parquet —rewrite→ file1b.parquet

file2.parquet

file3.parquet —rewrite→ file3b.parquet

_delta_log / 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

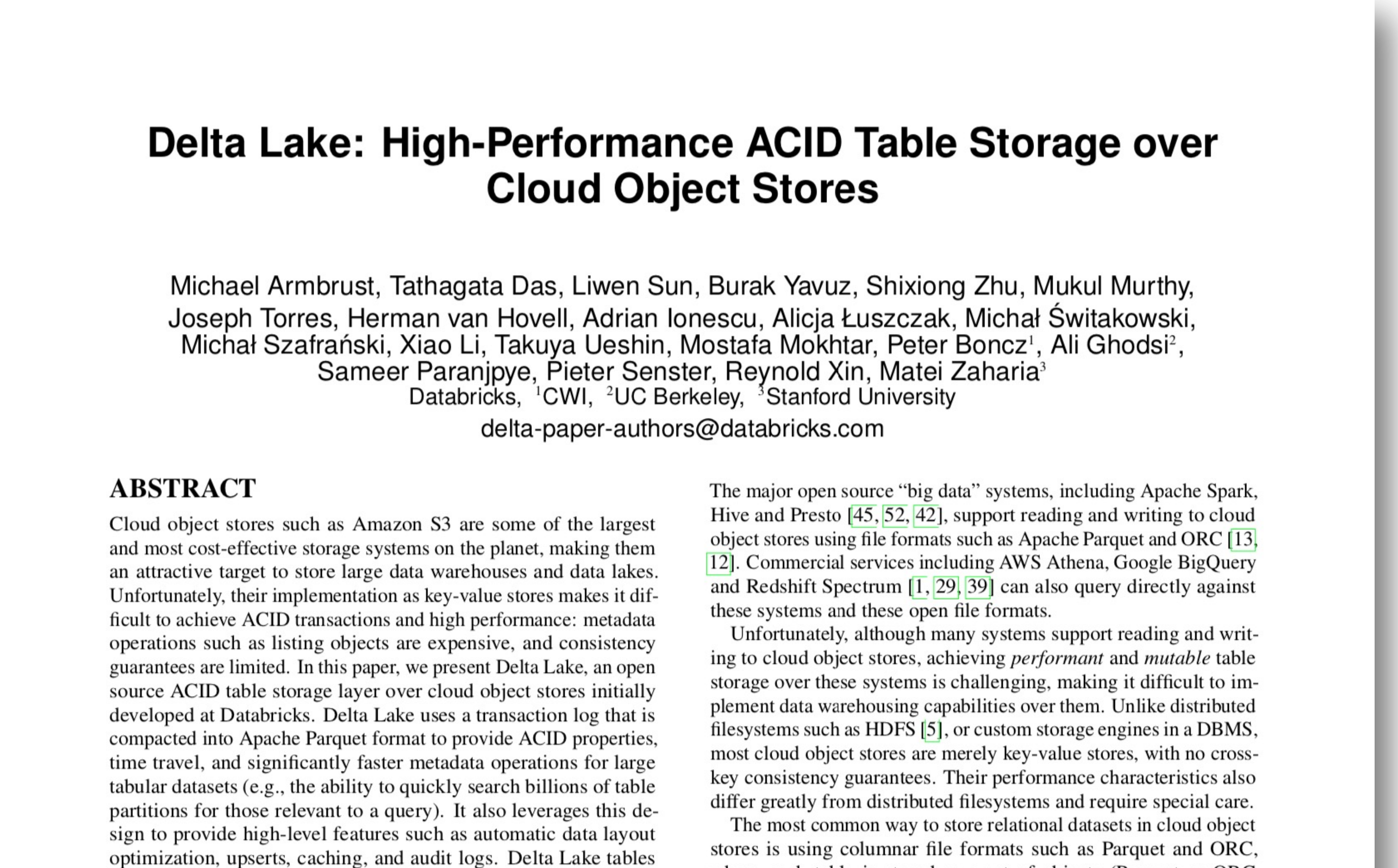
v3 = file1b, file2, file3b

Clients now always read a consistent table version!

- If a client reads v2 of log, it sees file1, file2, file3 (no delete)
- If a client reads v3 of log, it sees file1b, file2, file3b (all deleted)

See our VLDB 2020 paper for details

See our VLDB Paper for Details



Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores

Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, Michał Świtakowski, Michał Szafrański, Xiao Li, Takuya Ueshin, Mostafa Mokhtar, Peter Boncz[1], Ali Ghodsi[2], Sameer Paranjpye, Pieter Senster, Reynold Xin, Matei Zaharia[3]
Databricks, [1]CWI, [2]UC Berkeley, [3]Stanford University
delta-paper-authors@databricks.com

ABSTRACT

Cloud object stores such as Amazon S3 are some of the largest and most cost-effective storage systems on the planet, making them an attractive target to store large data warehouses and data lakes. Unfortunately, their implementation as key-value stores makes it difficult to achieve ACID transactions and high performance: metadata operations such as listing objects are expensive, and consistency guarantees are limited. In this paper, we present Delta Lake, an open source ACID table storage layer over cloud object stores initially developed at Databricks. Delta Lake uses a transaction log that is compacted into Apache Parquet format to provide ACID properties, time travel, and significantly faster metadata operations for large tabular datasets (e.g., the ability to quickly search billions of table partitions for those relevant to a query). It also leverages this design to provide high-level features such as automatic data layout optimization, upserts, caching, and audit logs. Delta Lake tables

The major open source "big data" systems, including Apache Spark, Hive and Presto [45, 52, 42], support reading and writing to cloud object stores using file formats such as Apache Parquet and ORC [13, 12]. Commercial services including AWS Athena, Google BigQuery and Redshift Spectrum [1, 29, 39] can also query directly against these systems and these open file formats.

Unfortunately, although many systems support reading and writing to cloud object stores, achieving *performant* and *mutable* table storage over these systems is challenging, making it difficult to implement data warehousing capabilities over them. Unlike distributed filesystems such as HDFS [5], or custom storage engines in a DBMS, most cloud object stores are merely key-value stores, with no cross-key consistency guarantees. Their performance characteristics also differ greatly from distributed filesystems and require special care.

The most common way to store relational datasets in cloud object stores is using columnar file formats such as Parquet and ORC,

Other Management Features with DELTA LAKE

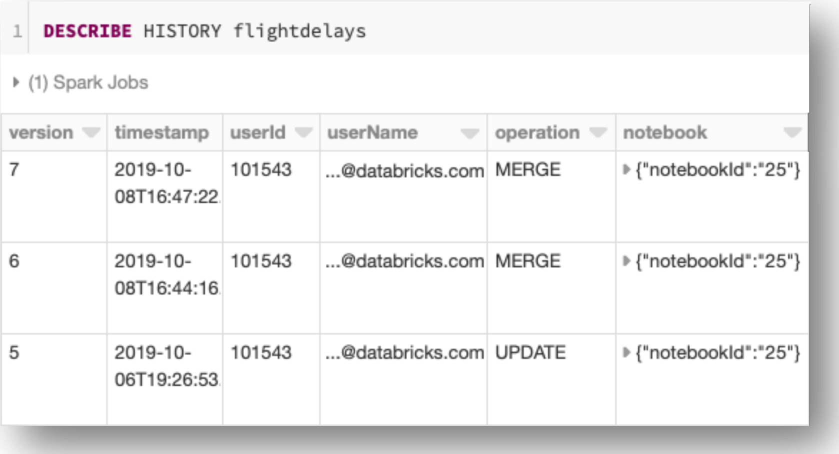
- Time travel to an old table version

```
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
```

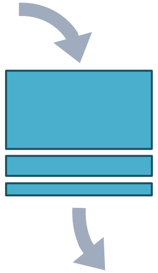
(1) Spark Jobs

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"}

- INSERT, UPSERT, DELETE & MERGE

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")
```

- Schema enforcement & evolution

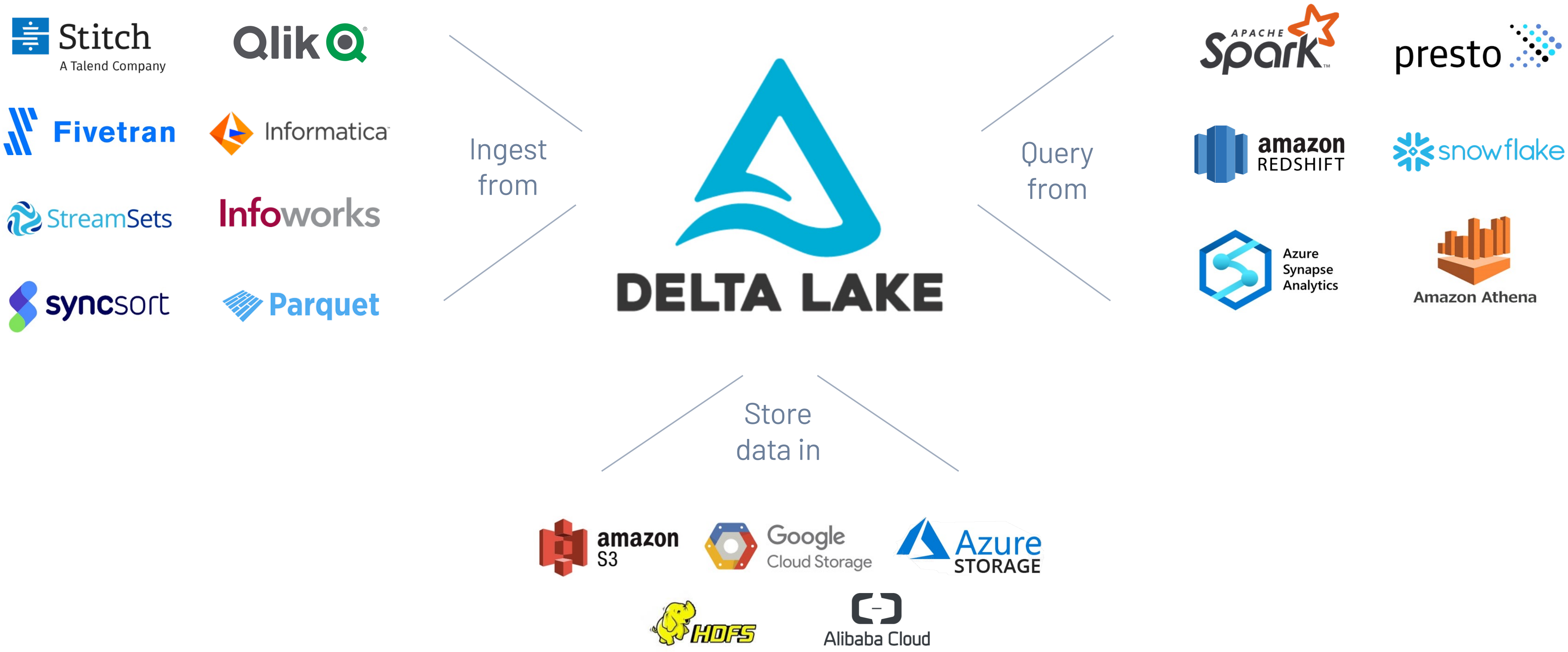
- Expectations for data quality

```
CREATE TABLE orders (
  product_id INTEGER NOT NULL,
  quantity INTEGER CHECK(quantity > 0),
  list_price DECIMAL CHECK(list_price > 0),
  discount_price DECIMAL
    CHECK(discount_price > 0 AND
          discount_price <= list_price)
);
```

DELTA LAKE Adoption

- Used by thousands of companies to process exabytes of data/day
- Grew from zero to ~50% of the Databricks workload in 3 years
- Largest deployments: exabyte tables and 1000s of users

Available Connectors



Ingest from

Stitch A Talend Company

Qlik

Fivetran

Informatica

StreamSets

Infoworks

syncsort

Parquet

DELTA LAKE

Query from

APACHE Spark

presto

amazon REDSHIFT

snowflake

Azure Synapse Analytics

Amazon Athena

Store data in

amazon S3

Google Cloud Storage

Azure STORAGE

HDFS

Alibaba Cloud

Key Technologies Enabling Lakehouse

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

2. New query engine designs: great SQL performance on data lake storage systems and file formats

3. Optimized access 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, four optimizations can enable great SQL performance:

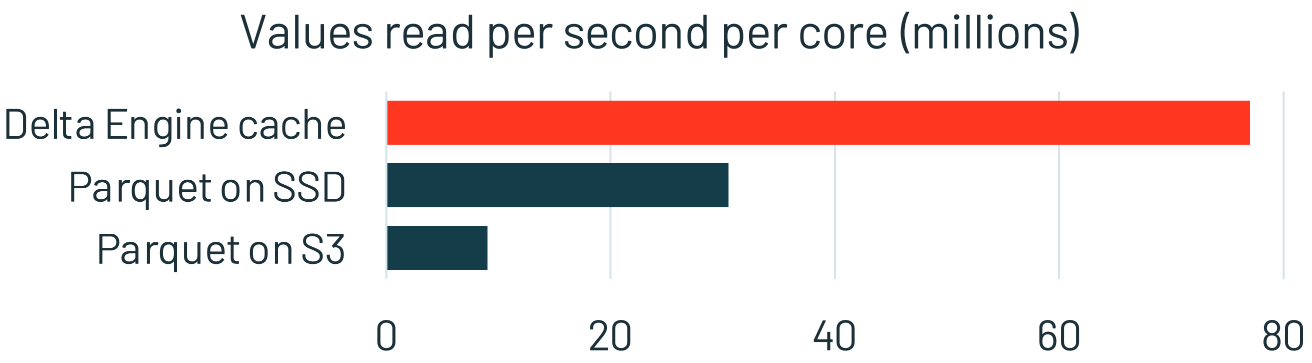
- Caching hot data, possibly in a different format
- Auxiliary data structures like statistics and indexes
- Data layout optimizations to minimize I/O
- Vectorized execution engines for modern CPUs

New query engines such as Databricks Delta Engine use these ideas

Optimization 1: Caching

- Most query workloads have concentrated accesses on “hot” data
  - Data warehouses use SSD and memory caches to improve performance
- The same techniques work in a Lakehouse if we have a metadata layer such as Delta Lake to correctly maintain the cache
  - Caches can even hold data in a faster format (e.g. decompressed)

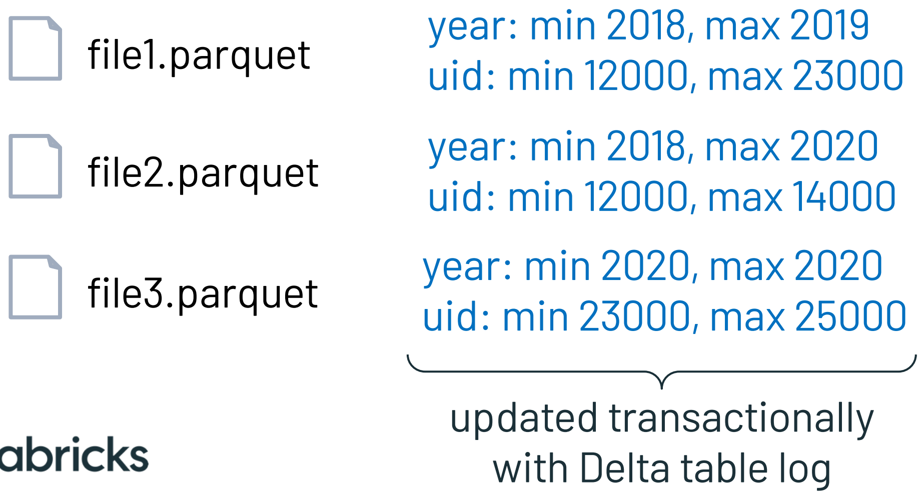
- **Example:** SSD cache in Databricks Delta Engine



Optimization 2: Auxiliary Data Structures

- Even if the base data is in Parquet, we can build many other data structures to speed up queries and maintain them transactionally
  - Inspired by the literature on databases for "raw" data formats

- **Example:** min/max statistics on Parquet files 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

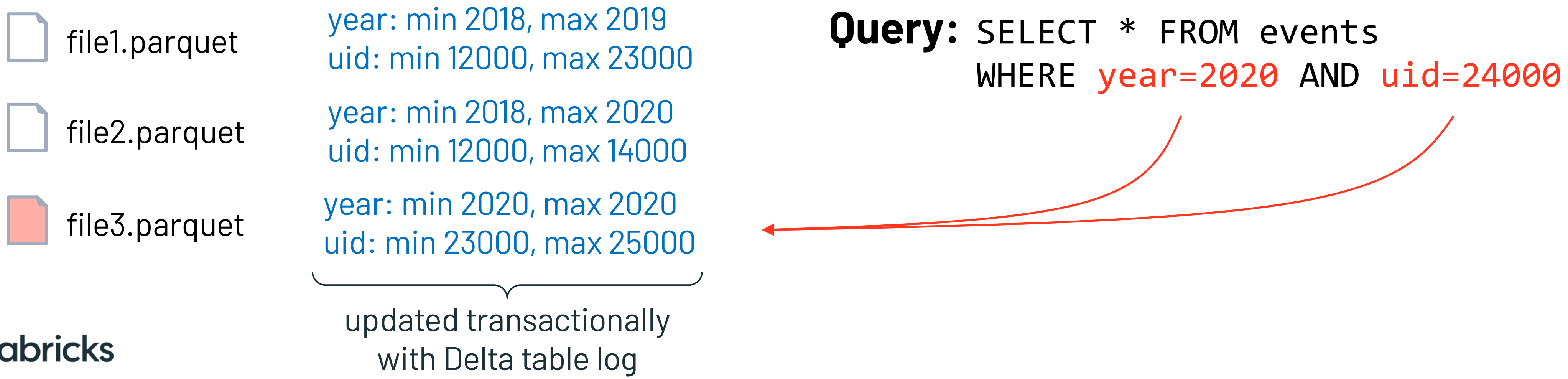
Query:

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

Optimization 2: Auxiliary Data Structures

- Even if the base data is in Parquet, we can build many other data structures to speed up queries and maintain them transactionally
  - Inspired by the literature on databases for “raw” data formats

- **Example:** min/max statistics on Parquet files 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

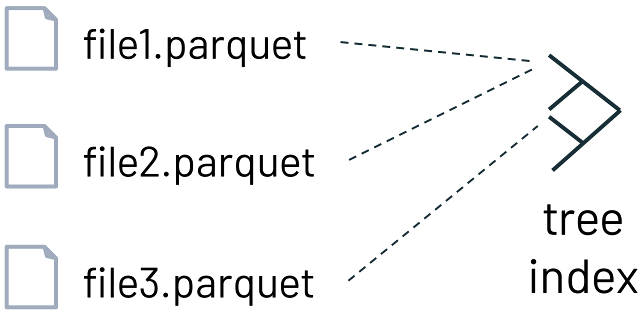
updated transactionally
with Delta table log

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

Optimization 2: Auxiliary Data Structures

- Even if the base data is in Parquet, we can build many other data structures to speed up queries and maintain them transactionally
  - Inspired by the literature on databases for “raw” data formats

- **Example:** indexes over Parquet files



Query:

```
SELECT * FROM events
WHERE type = “DELETE_ACCOUNT”
```

Optimization 3: Data Layout

- Query execution time primarily depends on amount of data accessed
- Even with a fixed storage format such as Parquet, we can optimize the data layout *within* tables to reduce execution time

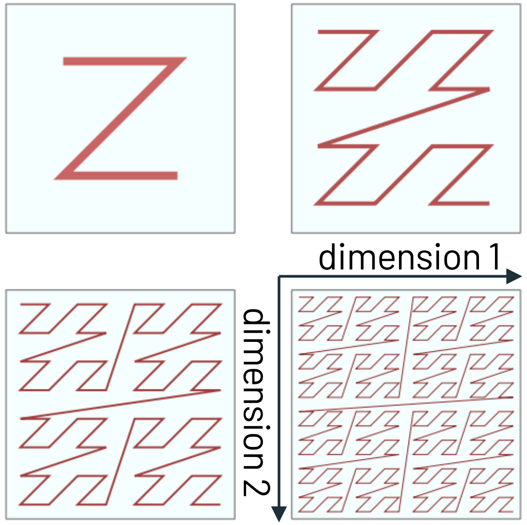
- **Example:** sorting a table for fast access

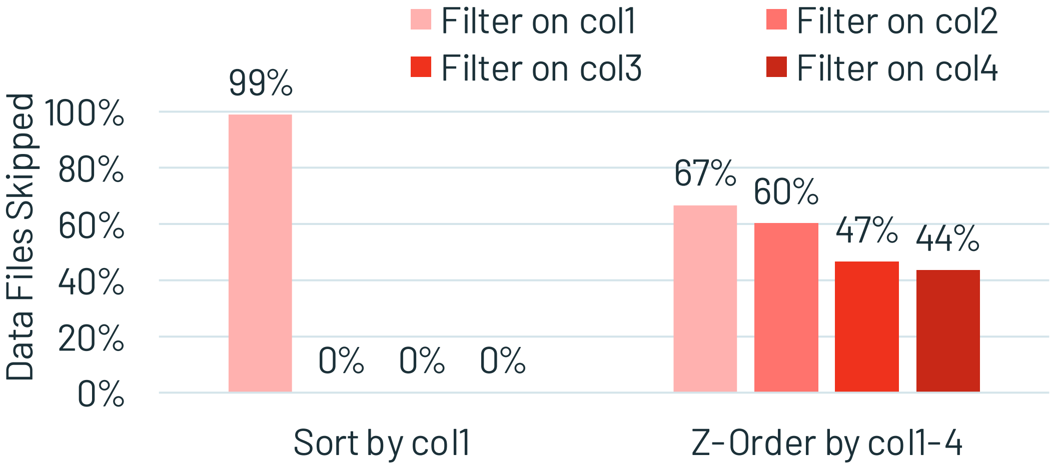
file1.parquet	uid = 0…1000
file2.parquet	uid = 1001…2000
file3.parquet	uid = 2001…3000
file4.parquet	uid = 3001…4000

Optimization 3: Data Layout

- Query execution time primarily depends on amount of data accessed
- Even with a fixed storage format such as Parquet, we can optimize the data layout within tables to reduce execution time

- **Example:** Z-ordering for multi-dimensional access

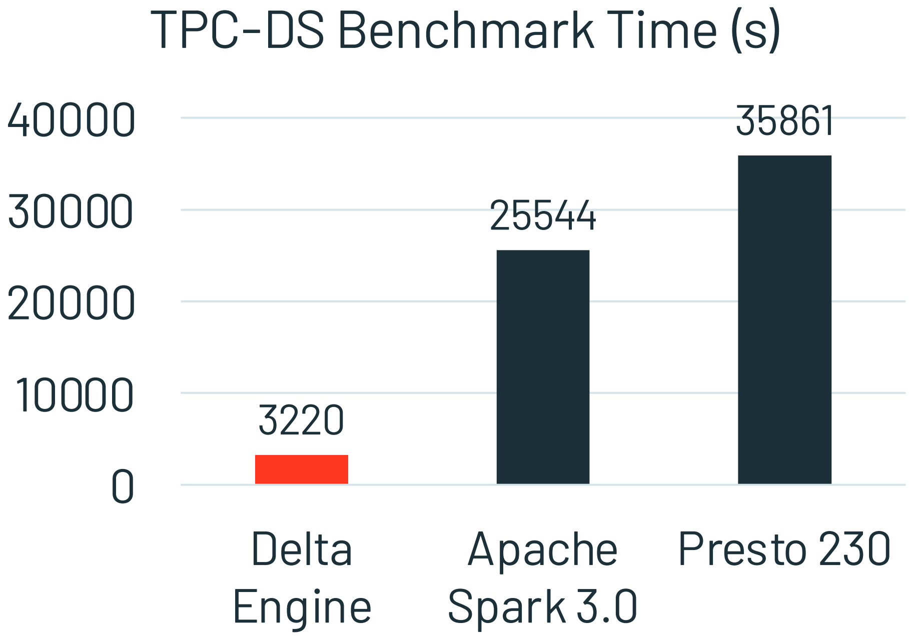




Optimization 4: Vectorized Execution

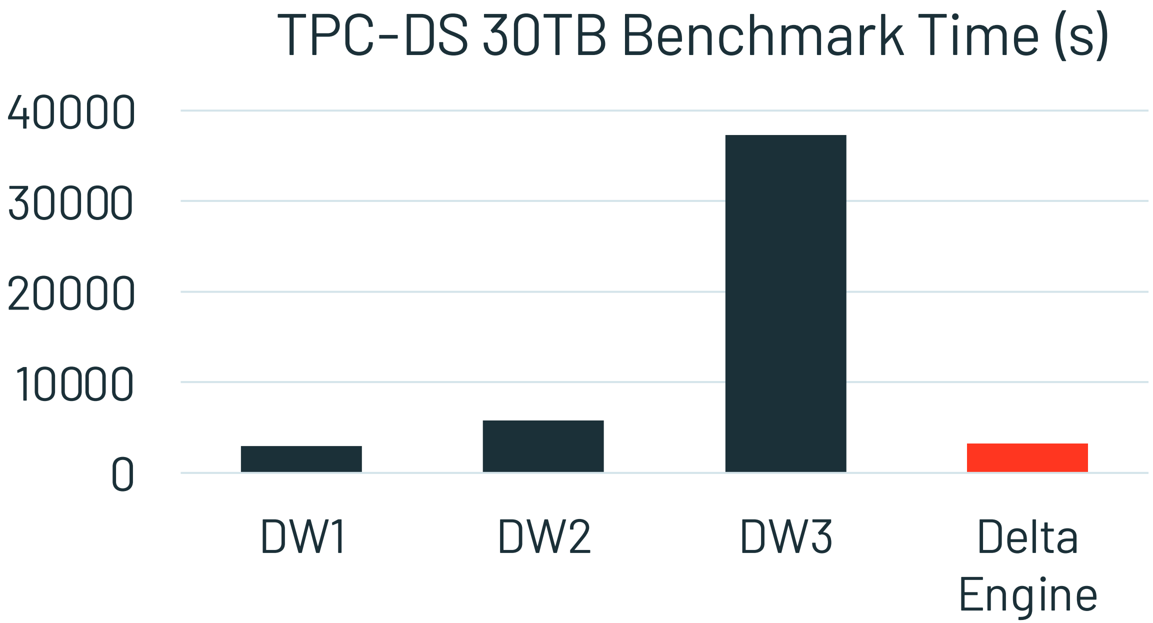
- Modern data warehouses optimize CPU time by using vector (SIMD) instructions on modern CPUs, e.g., AVX512
- Many of these optimizations can also be applied over Parquet

- Databricks Delta Engine: ~10x faster than Java-based engines



Putting These Optimizations Together

- Given that (1) most reads are from a cache, (2) I/O cost is the key factor for non-cached data, and (3) CPU time can be optimized via SIMD...
- Lakehouse engines can offer similar performance to DWs!



Key Technologies Enabling Lakehouse

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

2. New query engine designs: great SQL performance on data lake storage systems and file formats

3. Optimized access 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, etc)
- SQL over JDBC/ODBC interface 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 (e.g., just run a GPU cluster to do deep learning on S3 data)
  - ML frameworks already support reading Parquet!
- New declarative APIs for ML data prep enable further optimization

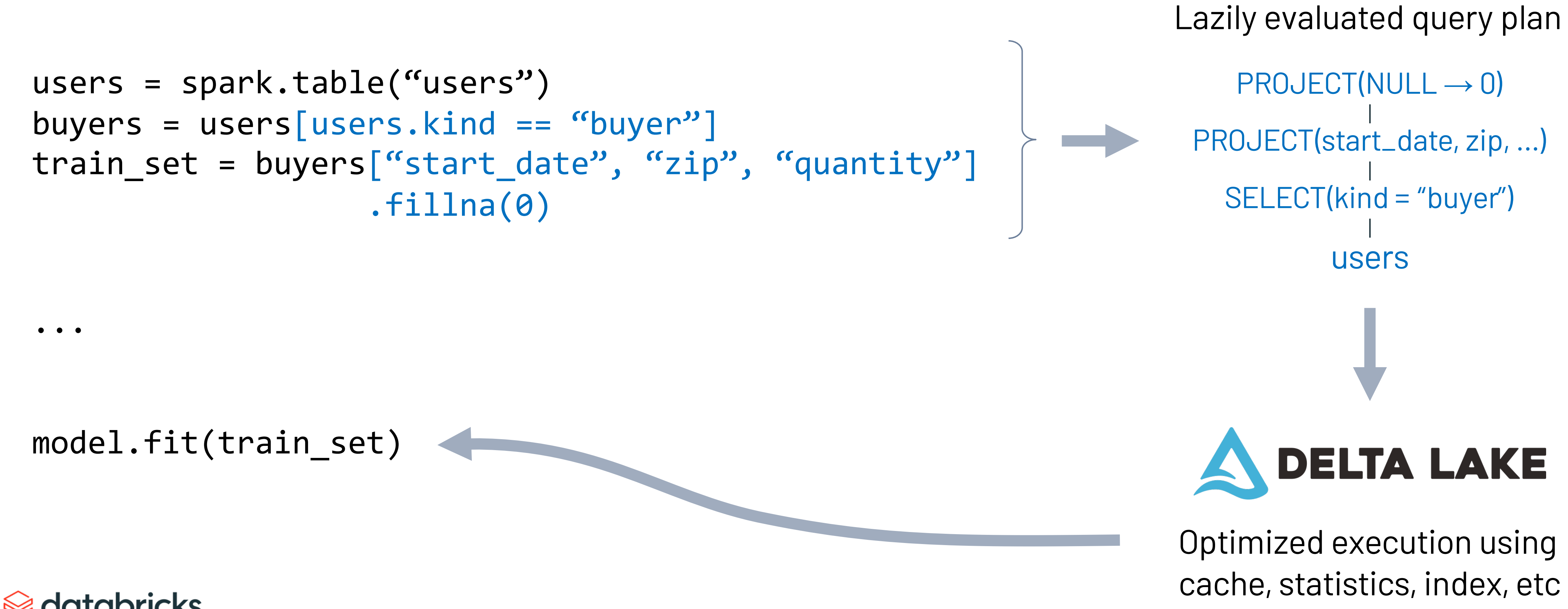
Example: Spark's Declarative DataFrame API

Users write DataFrame code in Python, R or Java

```
users = spark.table(“users”)
buyers = users[users.kind == “buyer”]
train_set = buyers[“start_date”, “zip”, “quantity”]
                  .fillna(0)
```

Example: Spark’s Declarative DataFrame API

Users write DataFrame code in Python, R or Java



```
users = spark.table(“users”)
buyers = users[users.kind == “buyer”]
train_set = buyers[“start_date”, “zip”, “quantity”]
                  .fillna(0)

...

model.fit(train_set)
```

Lazily evaluated query plan

PROJECT(NULL → 0)
PROJECT(start_date, zip, ...)
SELECT(kind = “buyer”)
users

DELTA LAKE

Optimized execution using cache, statistics, index, etc

ML over Lakehouse: Management Features

Lakehouse systems' management features also make ML easier!

- Use time travel for data versioning and reproducible experiments
- Use transactions to reliably update tables
- Always access the latest data from streaming I/O

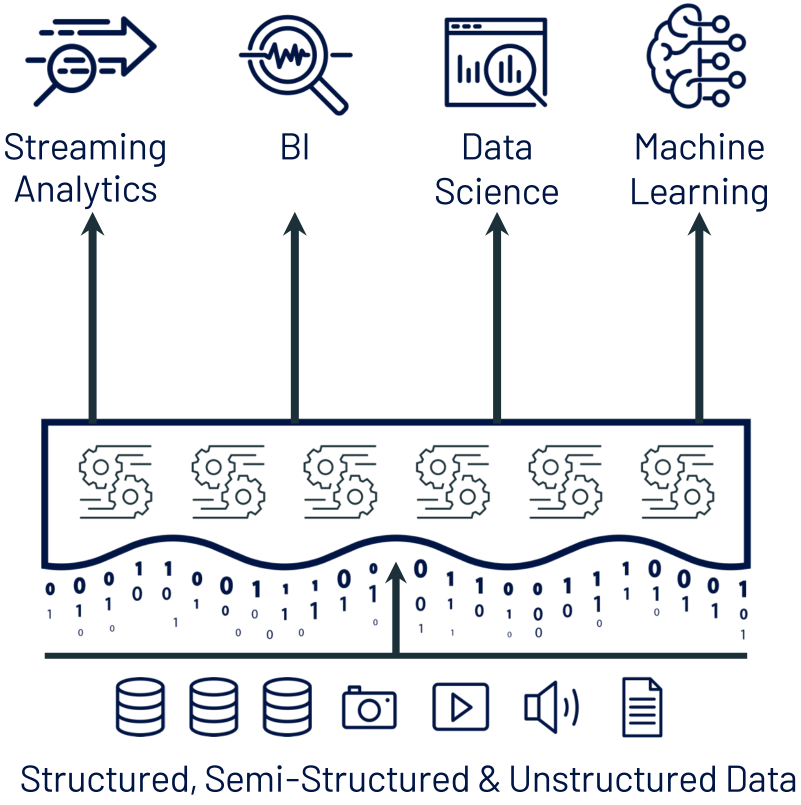
Example: organizations using Delta Lake as an ML "feature store"

Summary

Lakehouse systems **combine** the benefits of data warehouses & lakes

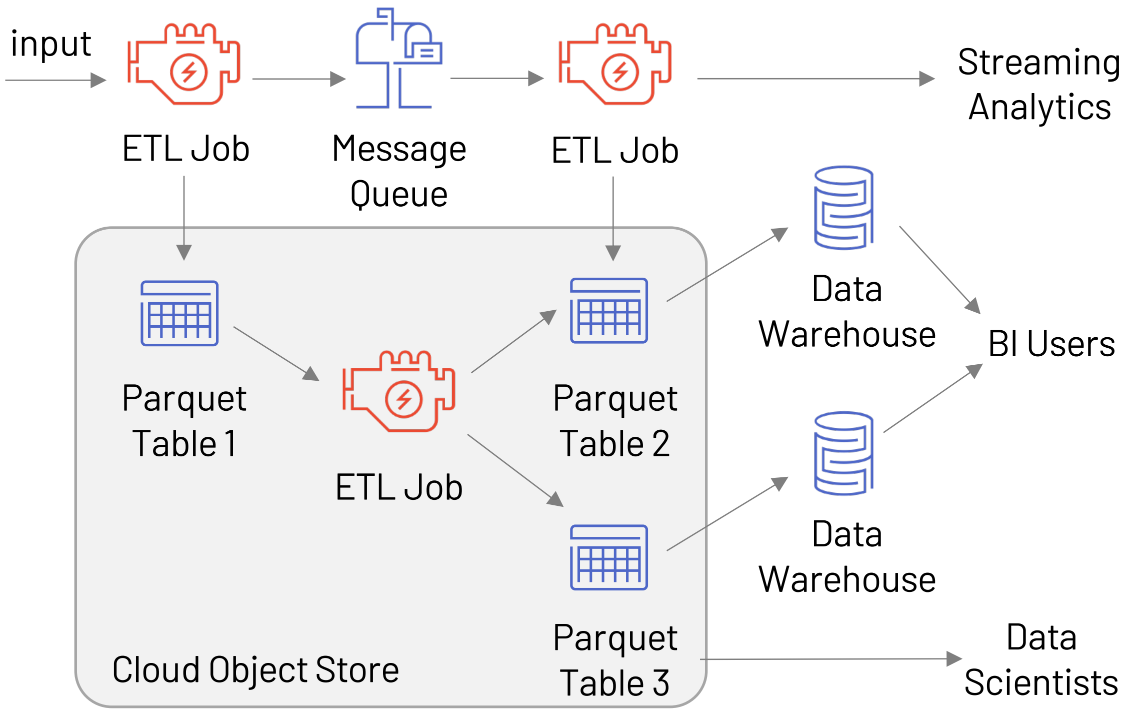
- Management features via metadata layers (transactions, CLONE, etc)
- Performance via new query engines
- Direct access via open file formats
- Low cost equal to cloud storage

Result: simplify data architectures to improve both **reliability** & **freshness**

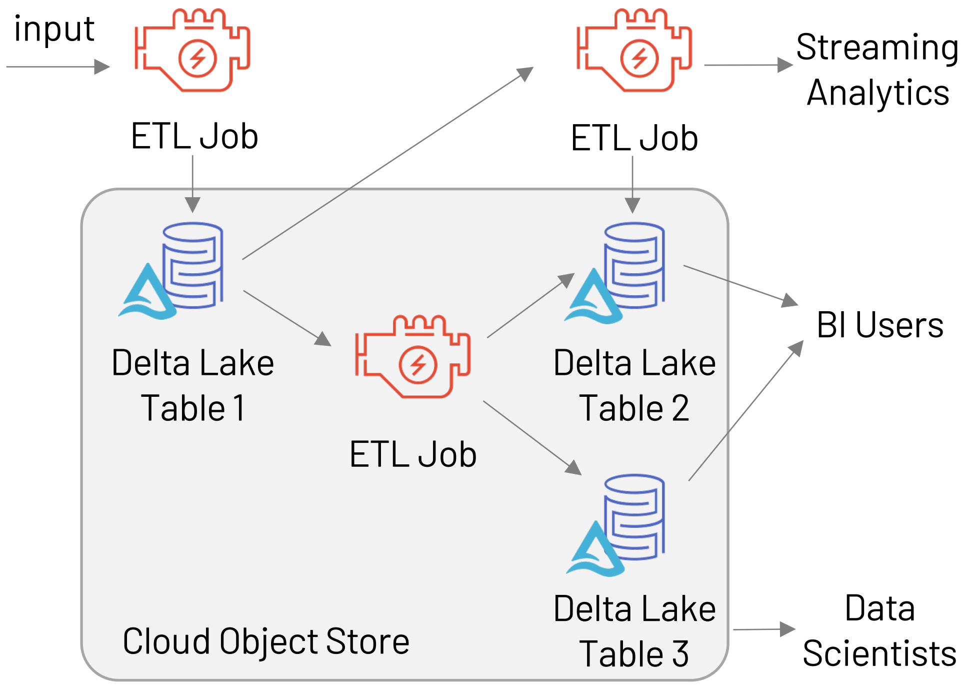


Before and After Lakehouse

Typical Architecture with Many Data Systems



Lakehouse Architecture: All Data in Object Store



Fewer copies of the data, fewer ETL steps, no divergence & faster results!

Learn More

Download and learn Delta Lake at delta.io

DELTA LAKE

View free content from our conferences at spark-summit.org:

SPARK+AI SUMMIT 2020

DATA+AI SUMMIT EUROPE

17-19 NOVEMBER | VIRTUAL

FORMERLY KNOWN AS SPARK+AI SUMMIT