DATA+AI SUMMIT 2022

Evolution of Data Architectures

And building a Lakehouse

Vini Jaiswal
Developer Advocate, Databricks

ORGANIZED BY databricks

/vinijaiswal @vini_jaiswal

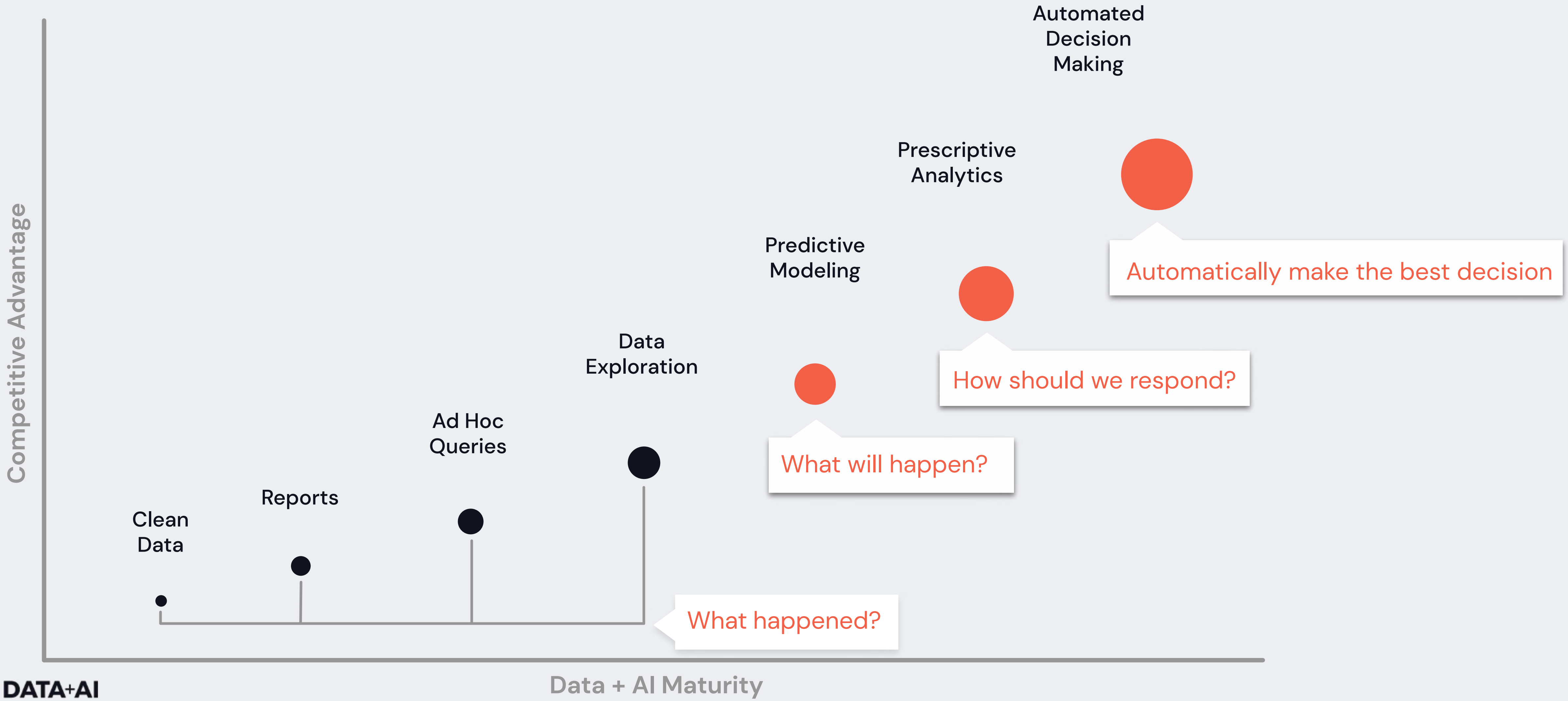
twitter Google Uber

Data, analytics, and AI enabled tech's leaders to disrupt industries

facebook NETFLIX TESLA

Tech leaders are to the right of the Data Maturity Curve

From hindsight to foresight



Every company wants to leverage data and AI

Goldman Sachs

Use AI to approve and underwrite a new Apple Card in less than 5 minutes on iPhone

COMCAST

Uses ML and voice recognition to create a highly innovative, and an Emmy winning viewer experience.

AMGEN

Speed up the development of a groundbreaking cancer treatment

Lakehouse adoption across industries

5CA · 80 ACRES FARMS · co op · aPlaceforMom · ancestry · RUNTASTIC · ABN·AMRO · Alchimie

almirall · Arca. Part of Accenture · asurion · AURA · AusNet services · ARTC · BEGHOU · BELIEVE · Bigtincan · bind · Bright Health Group · CAA · bond. · catenamedia · checkout.com

City of Seattle · CONDÉ NAST · Confused.com · CA Crédito Agrícola · current health · Cuscal · DODO BRANDS · edmentum · edp · DTE · EMSA · EQUILEND

TD Bank · EY · Flinders University · flipp · FollowAnalytics · Ford · FORDDIRECT · FRAMEBRIDGE · Fraudio · GEMINI · gousto · Grab

healthgrades · grammarly · HIVERY · Hitz Hitachi Zosen · hometrack · HybridTheory · impact · MEMORY · HALLIBURTON · Inchcape

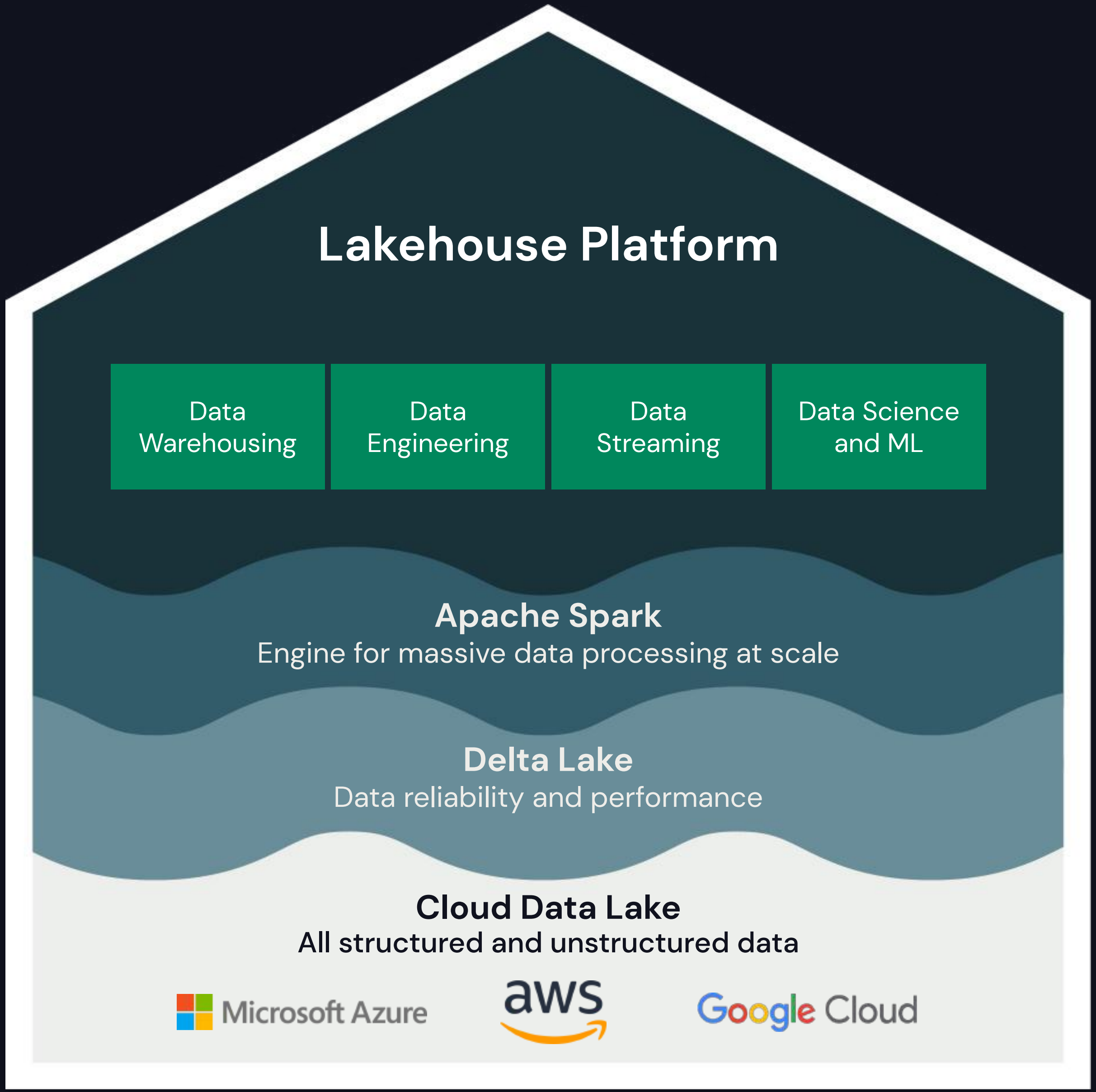
INTEGRA · JAMCITY · JB HI-FI · kaizen gaming · KERRY · Kollective · ITERABLE · LaLiga · LUMENTUM · LUNO · MARKERSTUDY INSURANCE SERVICES LIMITED · MiQ

METIS · Mitsubishi Tanabe Pharma · ThermoFisher SCIENTIFIC · Moneris · Money Dashboard · Namely · Schneider Electric · october · octopusenergy

oh! unmissable · PLAINS MIDSTREAM CANADA · poly · Progressive Leasing · Putnam INVESTMENTS · samsara · punchh · Railsbank · Relay42 · rbi · santalucía

SENSE CORP · StackAdapt · STARSHIP · SCRIBD · take blip · TEACHERS HEALTH · headspace · TEALIUM · Dolby · udaan

ULTRADENT · UNSW SYDNEY · VIVCOURT TRADING · every.tv · FIRST ORION · Vungle · wehkamp · wejo · WGU · winc. · xe χρυσή ευκαιρία · WILD LIFE · xp investments · zip



Lakehouse Platform

Simple

Unify your data warehousing and AI use cases on a single platform

Multicloud

One consistent data platform across clouds

Open

Built on open source and open standards

Most enterprises still struggle with data, analytics, and AI

Fivetran Data Analyst Survey

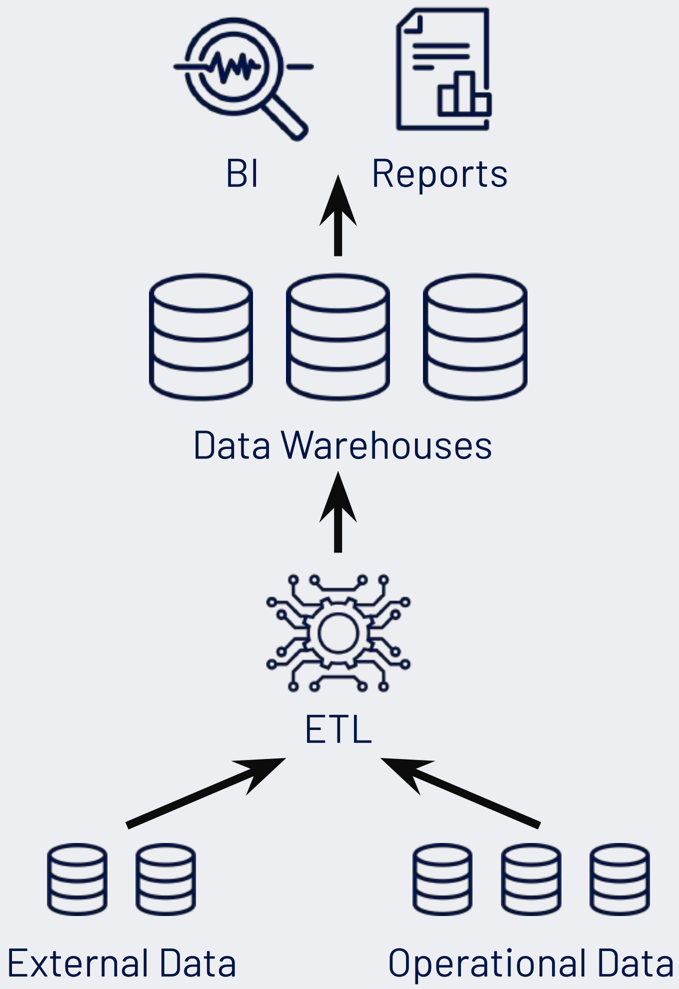
60% reported data quality as top challenge

86% of analysts using stale data.

90% regularly had unreliable data sources

Getting high-quality, timely data is hard

But it's also a problem with system architectures!



Data Warehouses

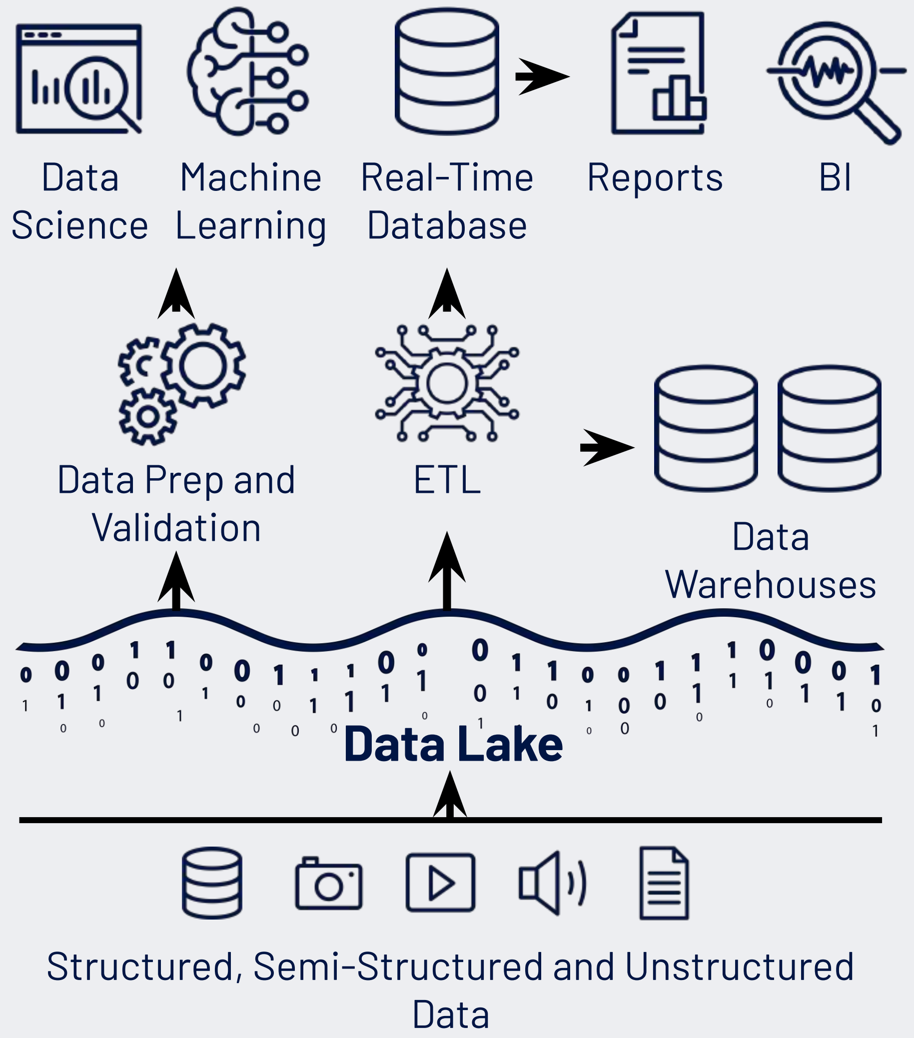
- ✓ High quality, reliable data
- ✓ Great for Business Intelligence

- ✖ Closed, proprietary format
- ✖ Only structured data
- ✖ No support for data science, ML, streaming
- ✖ Expensive to scale out

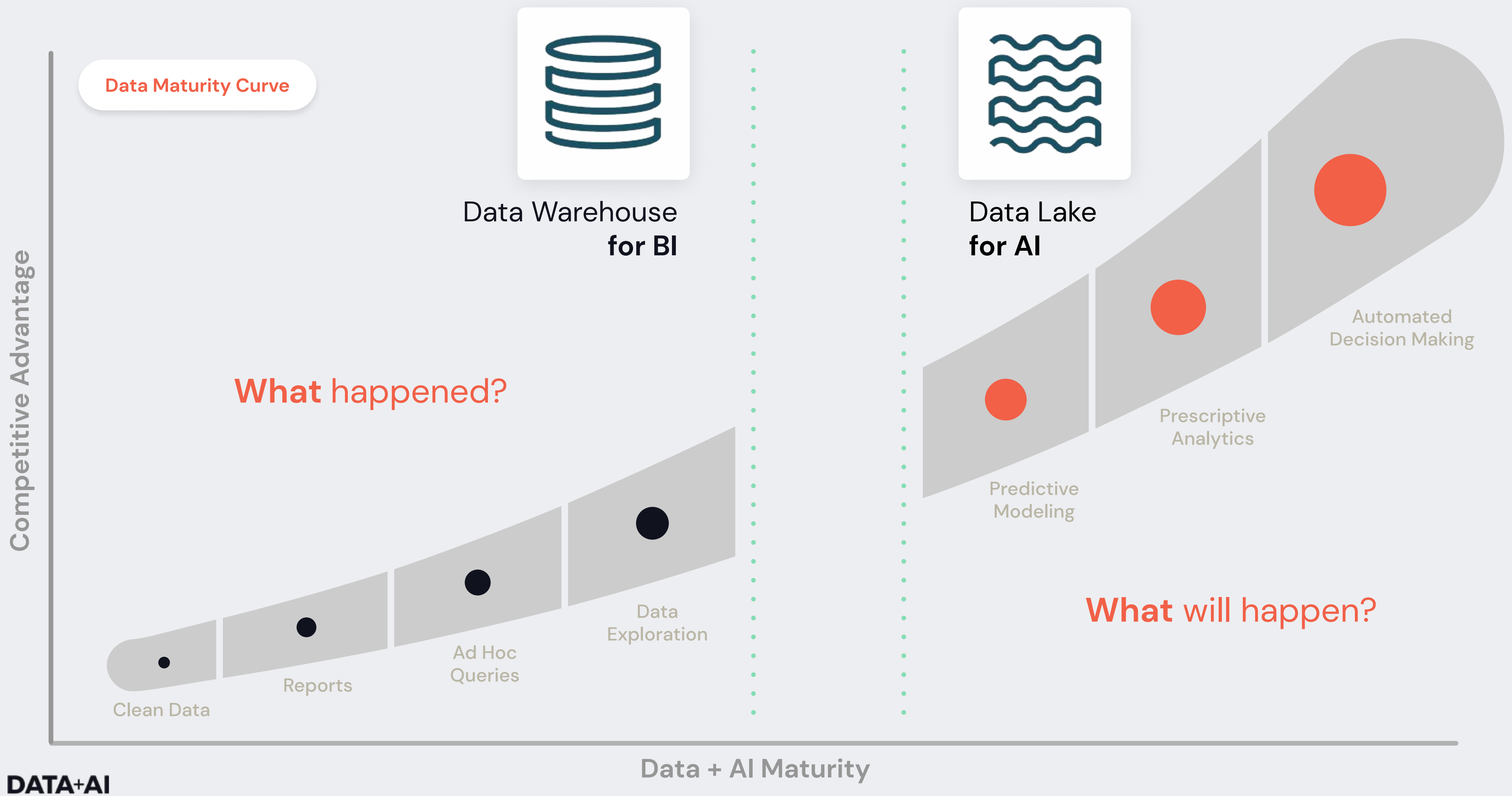
Data Lakes

- ✓ Open format
- ✓ Scalability and Flexibility
- ✓ All data types and use cases

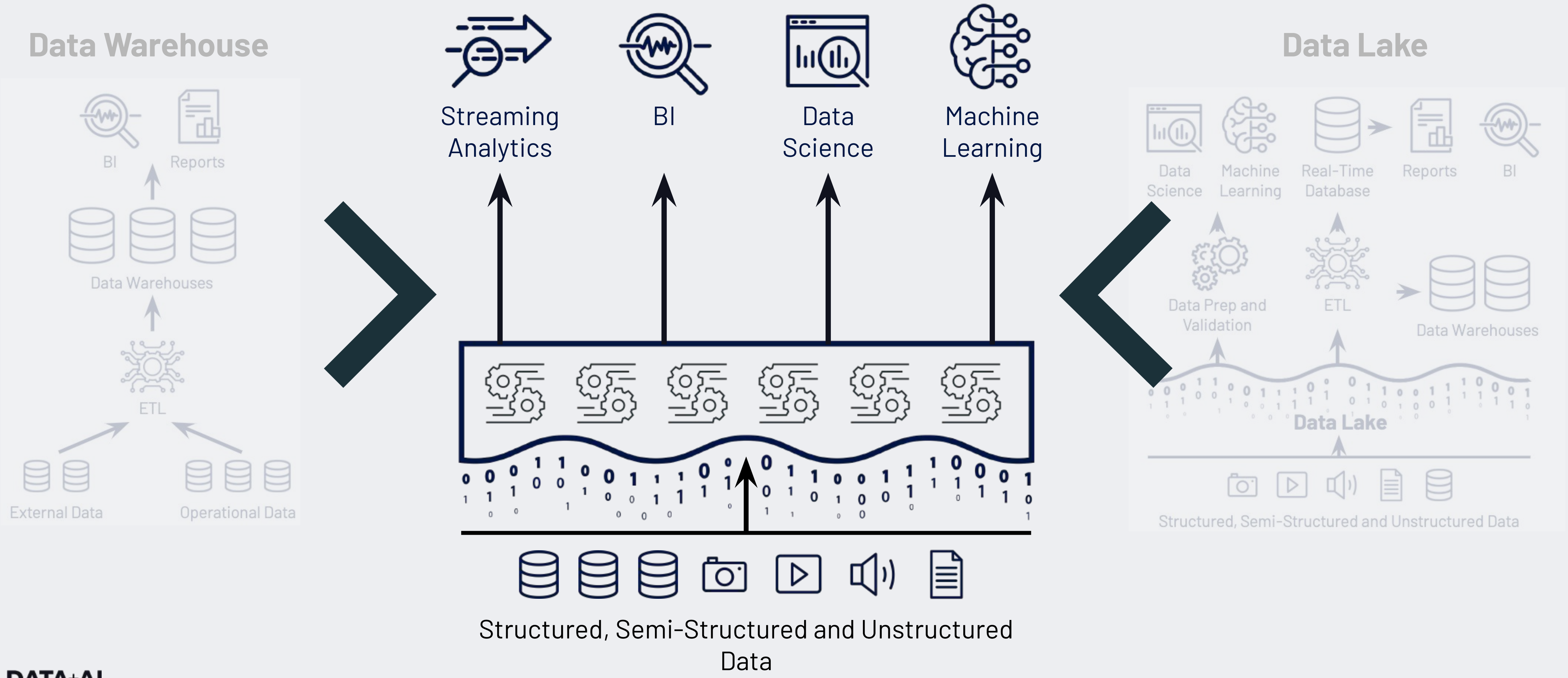
- ✖ Low data quality
- ✖ Complex to manage and govern
- ✖ Unreliable data swamps

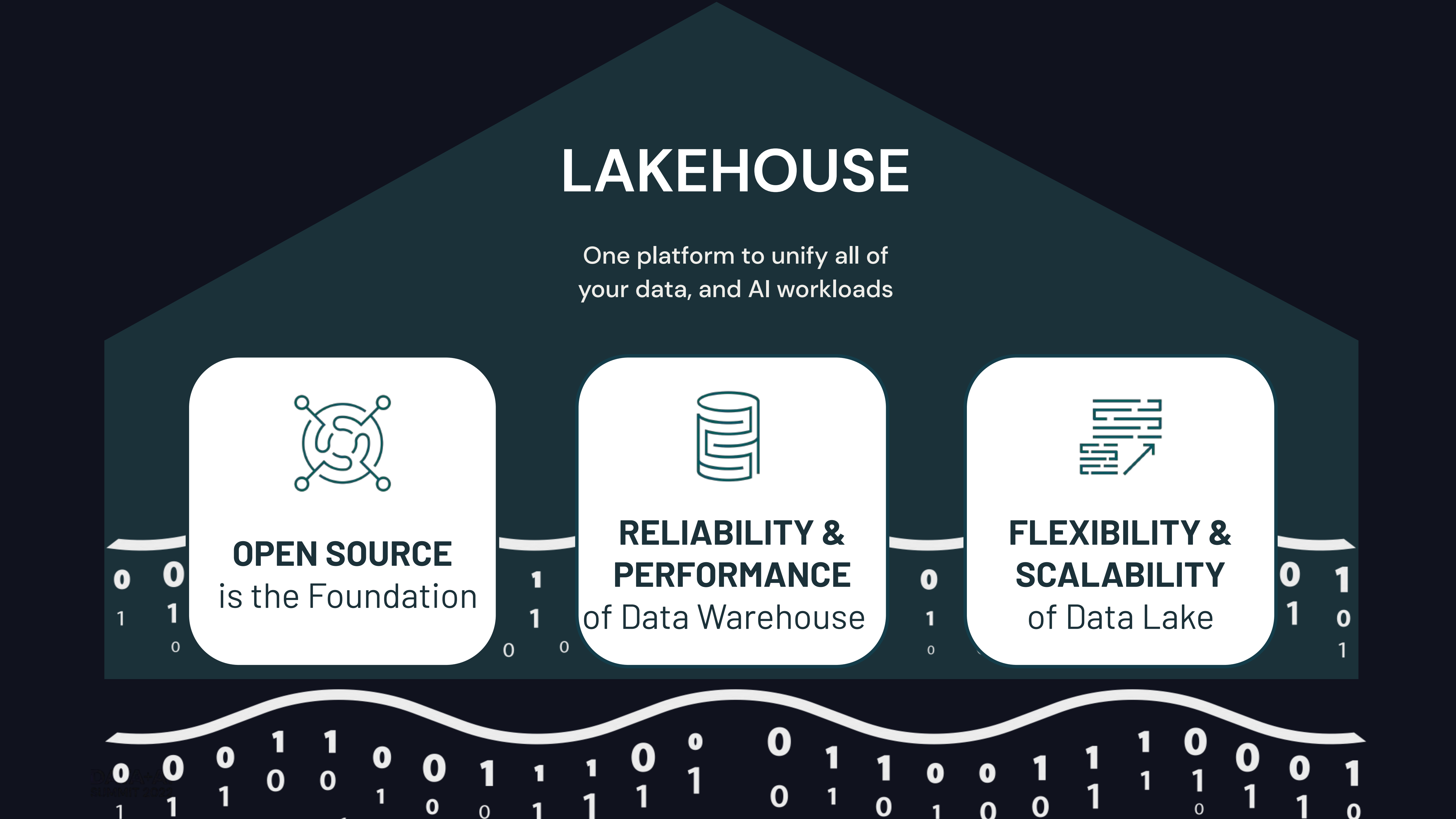


Realizing this requires two disparate, incompatible data platforms



Lakehouses – Best of Data warehouses + Data Lakes




LAKEHOUSE
One platform to unify all of
your data, and AI workloads
OPEN SOURCE
is the Foundation
RELIABILITY &
PERFORMANCE
of Data Warehouse
FLEXIBILITY &
SCALABILITY
of Data Lake

Evolution of Data Architectures

Data Warehouses

- High quality, reliable data
- Great for Business Intelligence

Data Lakes

- Open format
- Scalability and Flexibility
- All data types and use cases

Lakehouses

- Directly-accessible data in open formats
- Reliability and Performance
- Flexibility and Scalability
- AI and BI workloads

1980s: Datawarehouses	2010s: DW challenges	2010s: Data Lakes	Today's challenges	Today: Data Lakehouses

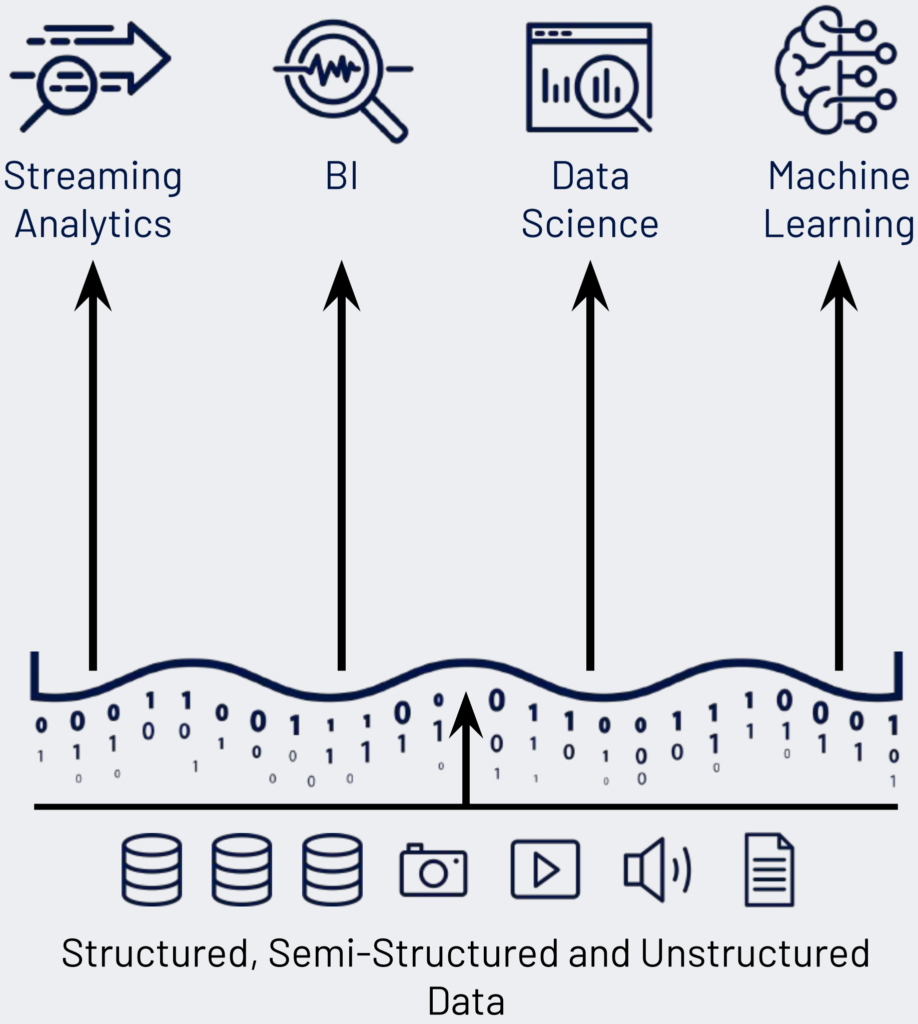
Problems for Data Warehouses

- Closed, proprietary format
- Only structured data
- No support for data science, ML, streaming
- Expensive to scale out

Problems with today's architectures

- Low data quality
- Complex to manage and govern
- Unreliable data swamps

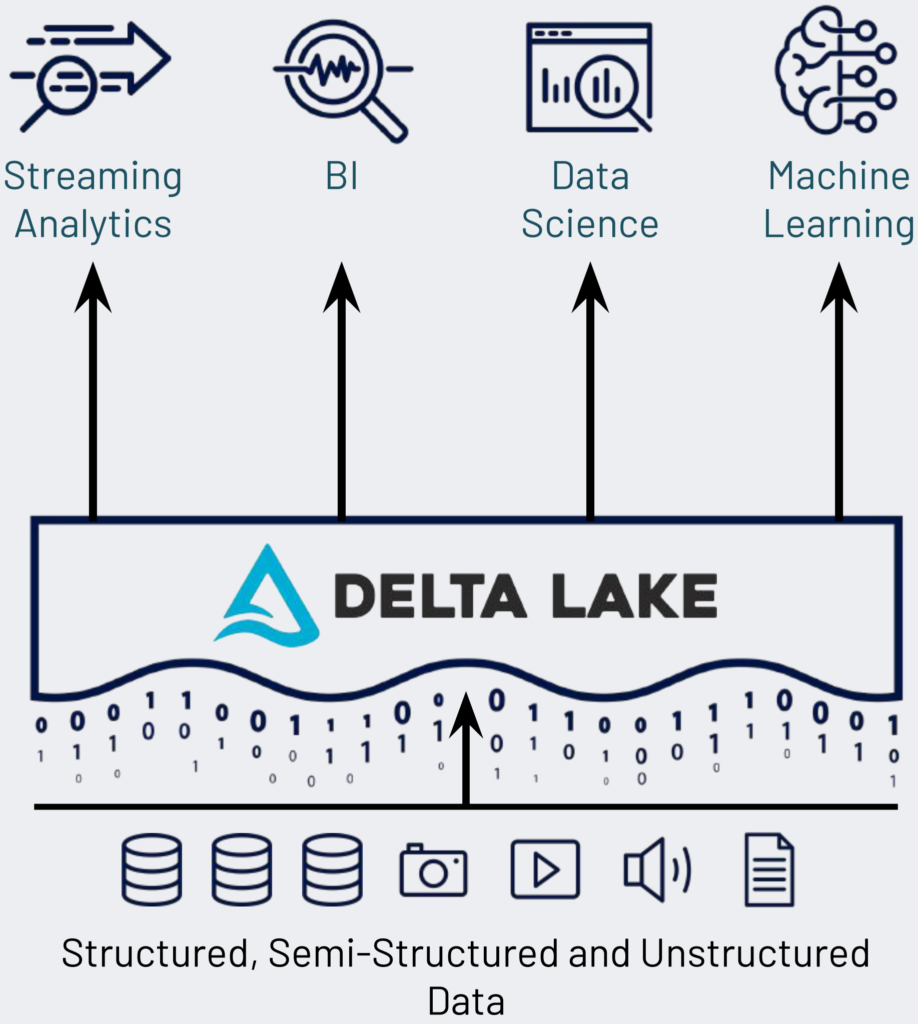
Lakehouse Architecture



One platform for all Data use cases

Scalable, low-cost directly accessible Cloud Data Lakes

Lakehouse Architecture

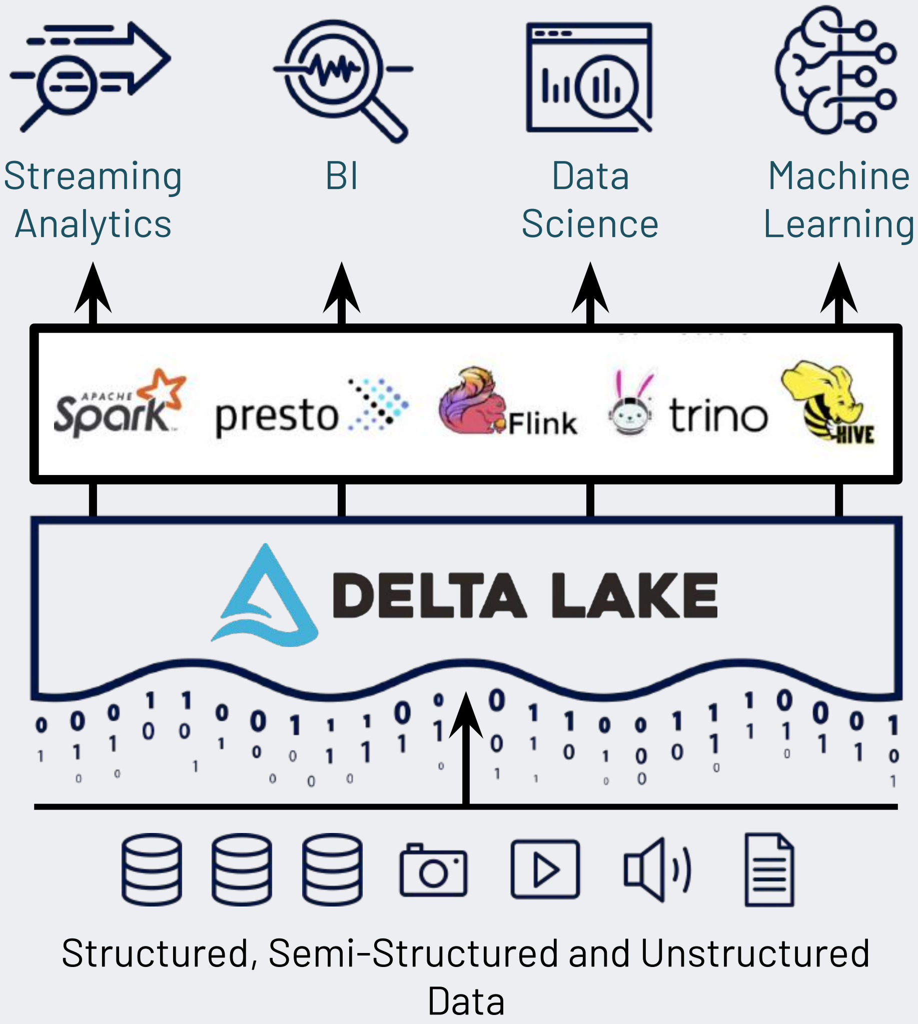


One platform for all Data Use Cases

Open, Transactional Layer for Curated Data

Scalable, low-cost directly accessible Cloud Data Lakes

Lakehouse Architecture



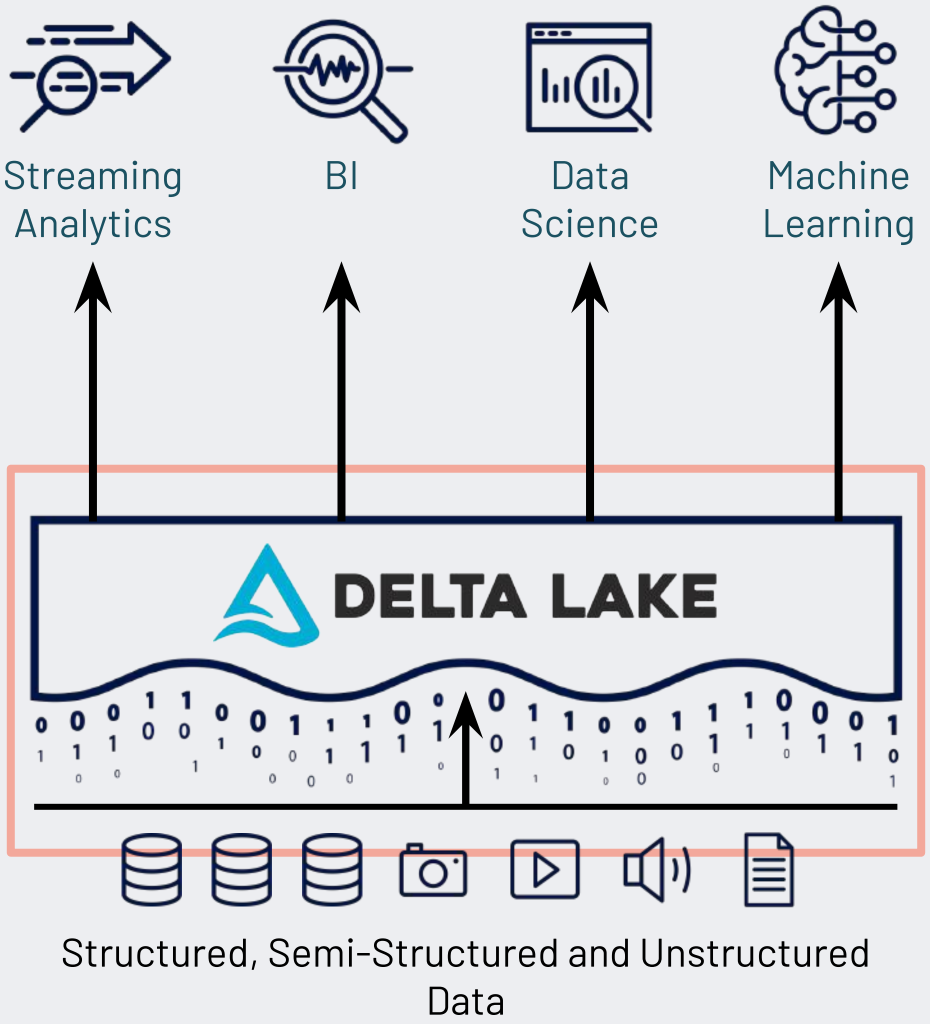
One platform for all Data Use Cases

High perf query engine(s)

Open, Transactional Layer for Curated Data

Scalable, low-cost directly accessible Cloud Data Lakes

Delta Lake: The Foundation of Lakehouses



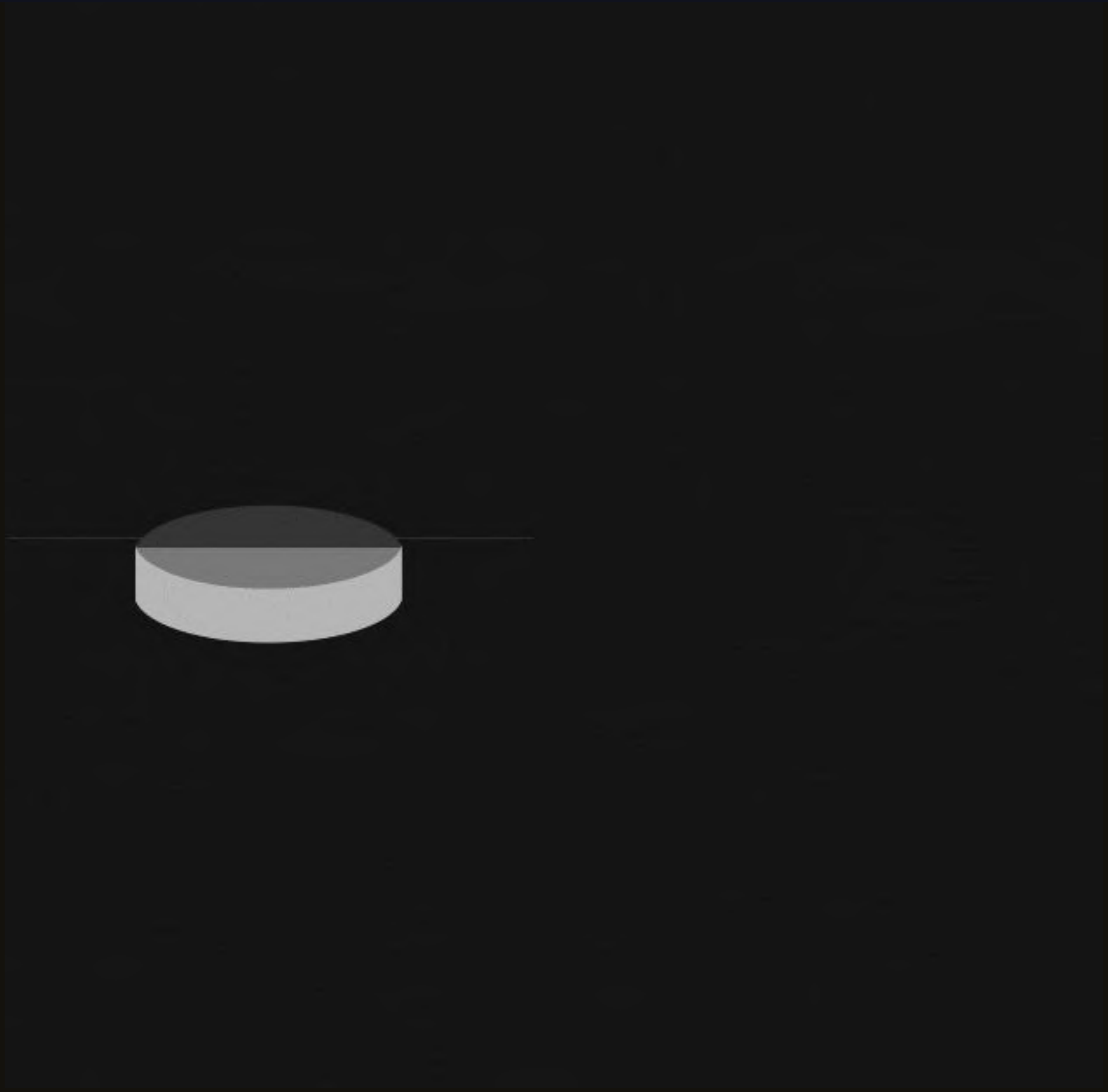
One Data Foundation for BI, Data Science & ML

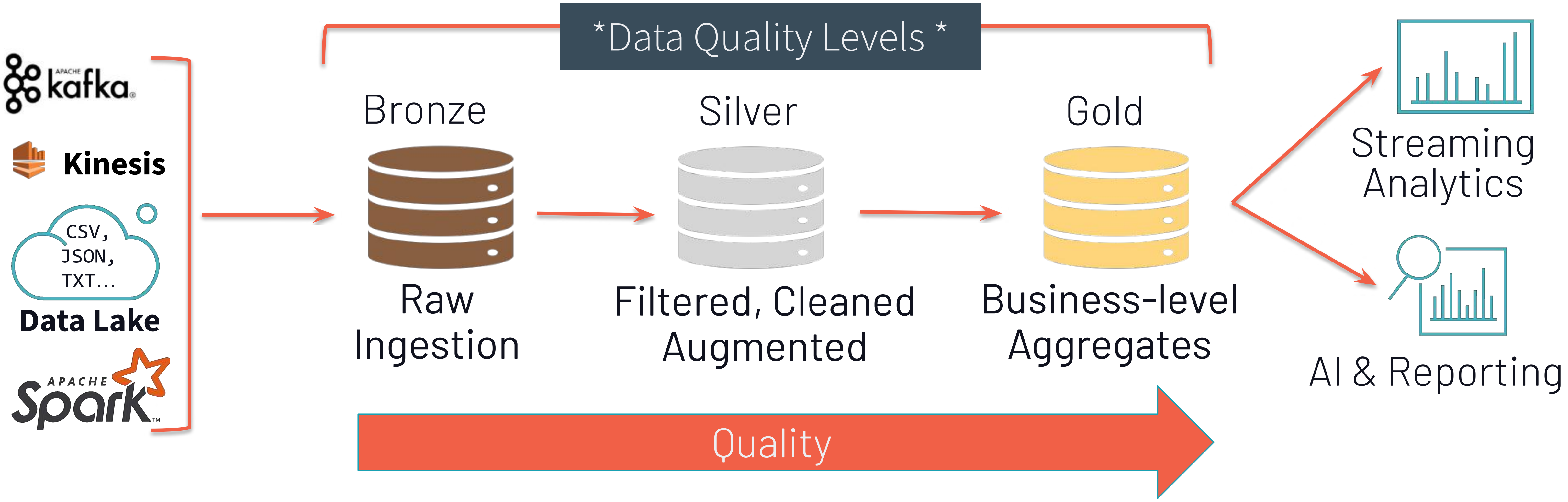
- Adds reliability, performance, governance, and quality to existing data lakes
- Based on open data format (Parquet)
- Simplifies data engineering with a curated data lake approach

DATA+AI SUMMIT 2022

THE LINUX FOUNDATION

How to build a Lakehouse?





Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.

Kafka

Kinesis

CSV,
JSON,
TXT...

Data Lake

Apache Spark

*Data Quality Levels *

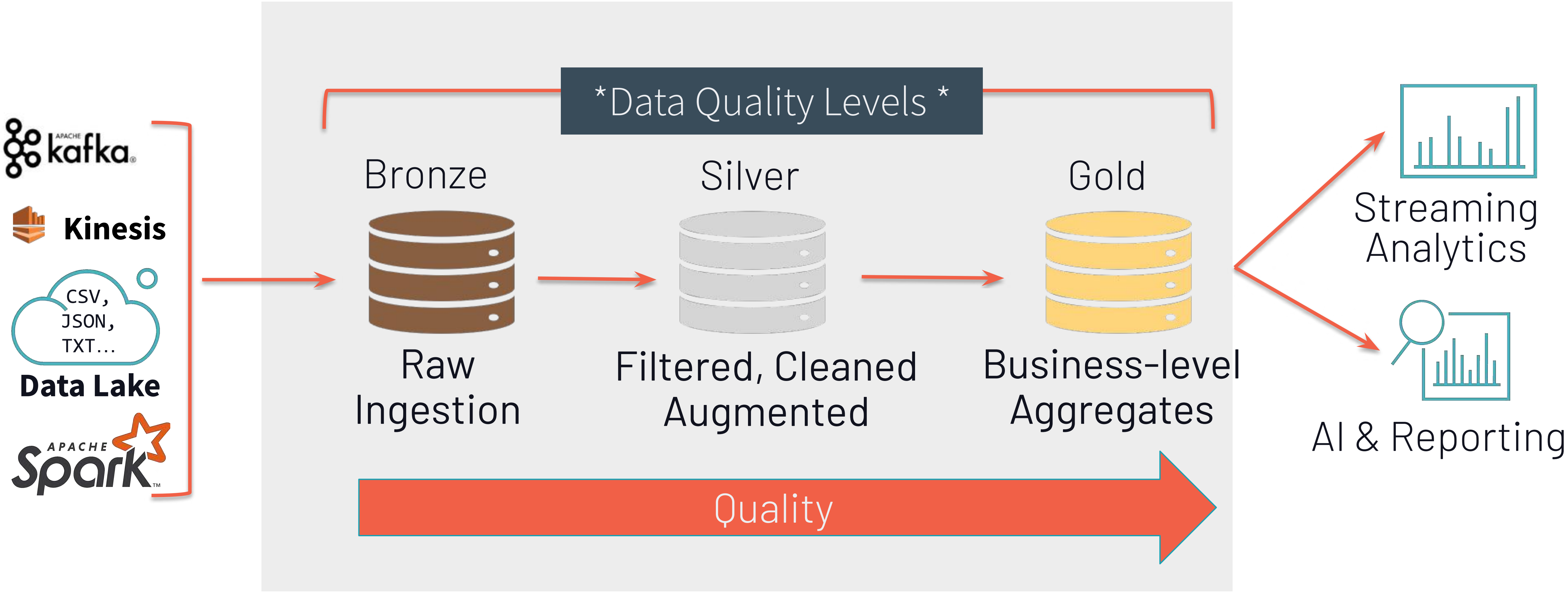
```
parquet_path = "file:/dbfs/tmp/delta_demo/loans_parquet/"

df = (spark.read.format("parquet").load(parquet_path)
      .withColumn("type", lit("batch"))
      .withColumn("timestamp", current_timestamp()))

df.write.format("delta").mode("overwrite").saveAsTable("loans_delta")
```

Quality

Streaming Analytics

AI & Reporting

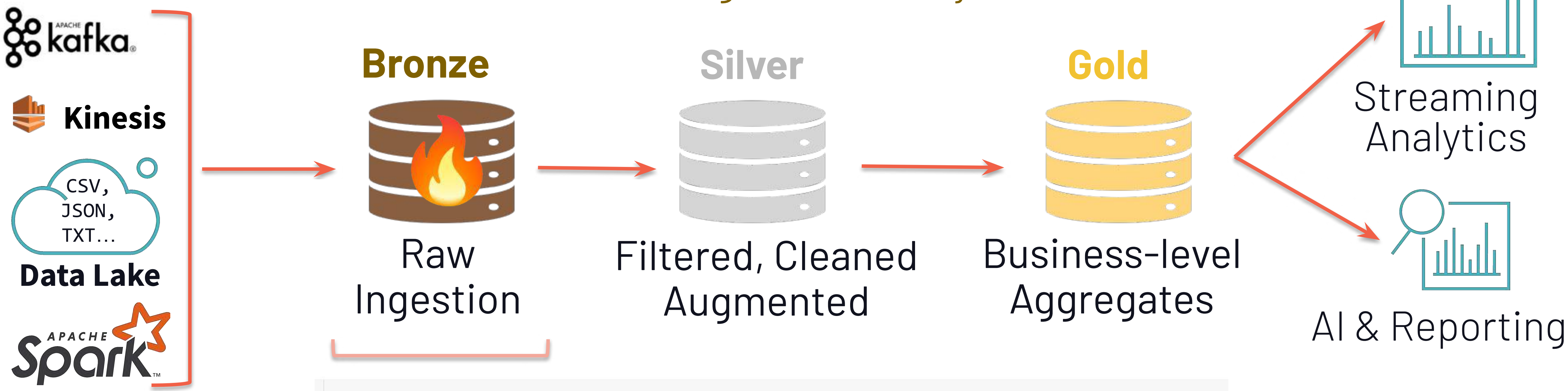
Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.



Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.

BRONZE TABLE

- Dumping ground for raw data
- Often with long retention (years)




```
bronzeDF = spark.sql("SELECT * FROM delta.`{}` limit 3".format(bronzePath))
display(bronzeDF)
```

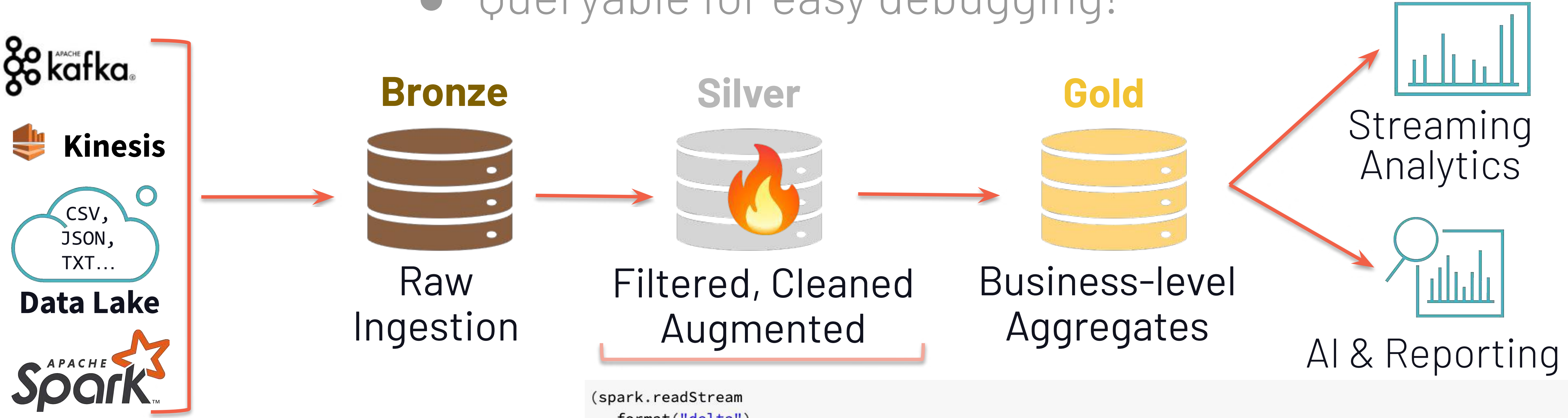
bronzeDF: pyspark.sql.dataframe.DataFrame = [kafka_timestamp: timestamp, channel: string ... 17 more fields]

ble Data Profile

	kafka_timestamp	channel	comment
1	1969-12-31T23:59:59.999+0000	#en.wikipedia	/* Update */
2	1969-12-31T23:59:59.999+0000	#en.wikipedia	Rescuing 1 sources and tagging 0 as dead.) #IABot (v2.0.8
3	1969-12-31T23:59:59.999+0000	#en.wikipedia	/* Continued discussion */ purpose of guidelines?

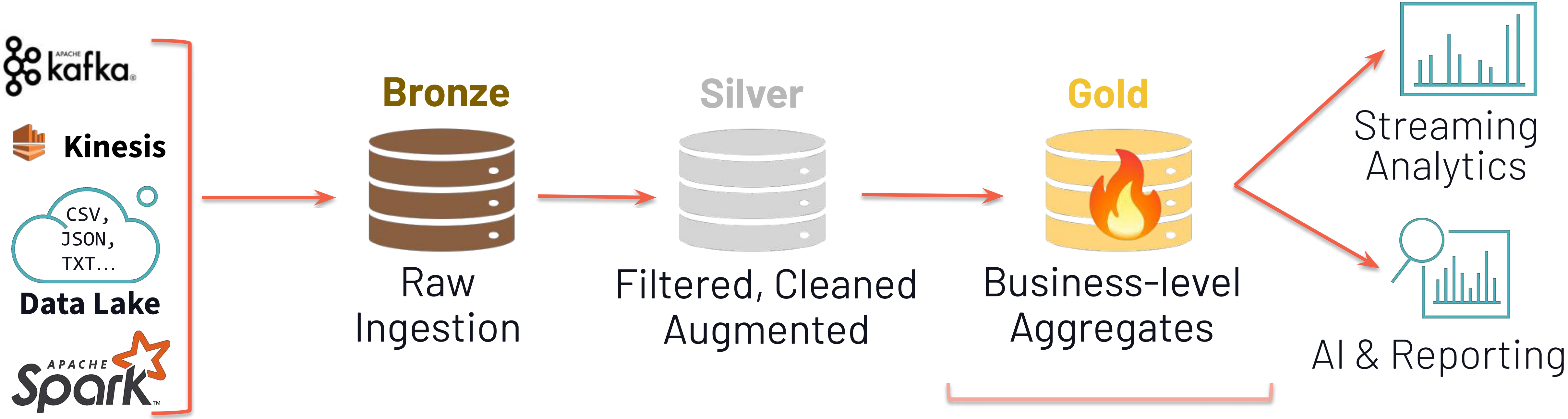
SILVER TABLE

- Intermediate data with some cleanup applied.
- Queryable for easy debugging!




```
(spark.readStream
  .format("delta")
  .load(bronzePath)
  .select(col("wikipedia"),
          col("isAnonymous"),
          col("namespace"),
          col("page"),
          col("pageURL"),
          col("geocoding"),
          unix_timestamp(col("timestamp"), "yyyy-MM-dd'T'HH:mm:ss.SSSX").cast("timestamp").alias("timestamp"),
          col("user"))
  .writeStream
  .format("delta")
  .option("checkpointLocation", checkpointPath + "/silver")
  .outputMode("append")
  .queryName("stream_2p")
  .start(silverPath)
```

GOLD TABLE Clean data, ready for consumption.




```
goldDF = (spark.readStream
  .format("delta")
  .load(silverPath)
  .withColumn("countryCode", col("geocoding.countryCode3"))
  .filter(col("namespace") == "article")
  .filter(col("countryCode") != "null")
  .filter(col("isAnonymous") == True)
  .groupBy(col("countryCode"))
  .count()
  .withColumnRenamed("count", "total")
  .orderBy(col("total").desc())
)
```

Internals of DELTA LAKE

DELTA LAKE

Scalable storage

- table data stored as Parquet files on cloud storage

```
pathToTable/
     +---- 000.parquet
    +---- 001.parquet            +----
002.parquet
    +    ...

    |
    +---- _delta_log/
        +---- 000.json
        +---- 001.json
        ...
```

Scalable transaction log

- sequence of metadata files to track operations made on files in the table
- stored in cloud storage along with table
- read and process metadata in parallel

Contents of Delta Directory

DELTA LAKE

Python

```
1  %fs ls dbfs:/Users/vini.jaiswal@databricks.com/demo/customer_t2
```

		name	size
1	demo/customer_t2/_delta_log/	_delta_log/	0
2	demo/customer_t2/part-00000-418067bd-bf5e-4d28-ac68-ea8cea49b956-	part-00000-418067bd-bf5e-4d28-ac68-ea8cea49b956-c001.snappy.parquet	542929260
3	demo/customer_t2/part-00000-45b799c4-6305-4c75-a3fd-fa1e840a8b99-	part-00000-45b799c4-6305-4c75-a3fd-fa1e840a8b99-c002.snappy.parquet	461126371
4	demo/customer_t2/part-00000-626b910a-882d-44ab-badf-756afde9f6e1-	part-00000-626b910a-882d-44ab-badf-756afde9f6e1-c000.snappy.parquet	540468009
5	demo/customer_t2/part-00001-405b13dc-0f72-419a-ab75-ddf993ba51af-	part-00001-405b13dc-0f72-419a-ab75-ddf993ba51af-c002.snappy.parquet	541858629
6	demo/customer_t2/part-00001-a794b5c5-8876-4aa9-89c0-92bb15cf5055-	part-00001-a794b5c5-8876-4aa9-89c0-92bb15cf5055-c001.snappy.parquet	542859315
7	demo/customer_t2/part-00001-c3f739cd-e309-43de-ab84-d005efb6bb7a-	part-00001-c3f739cd-e309-43de-ab84-d005efb6bb7a-c000.snappy.parquet	541186721

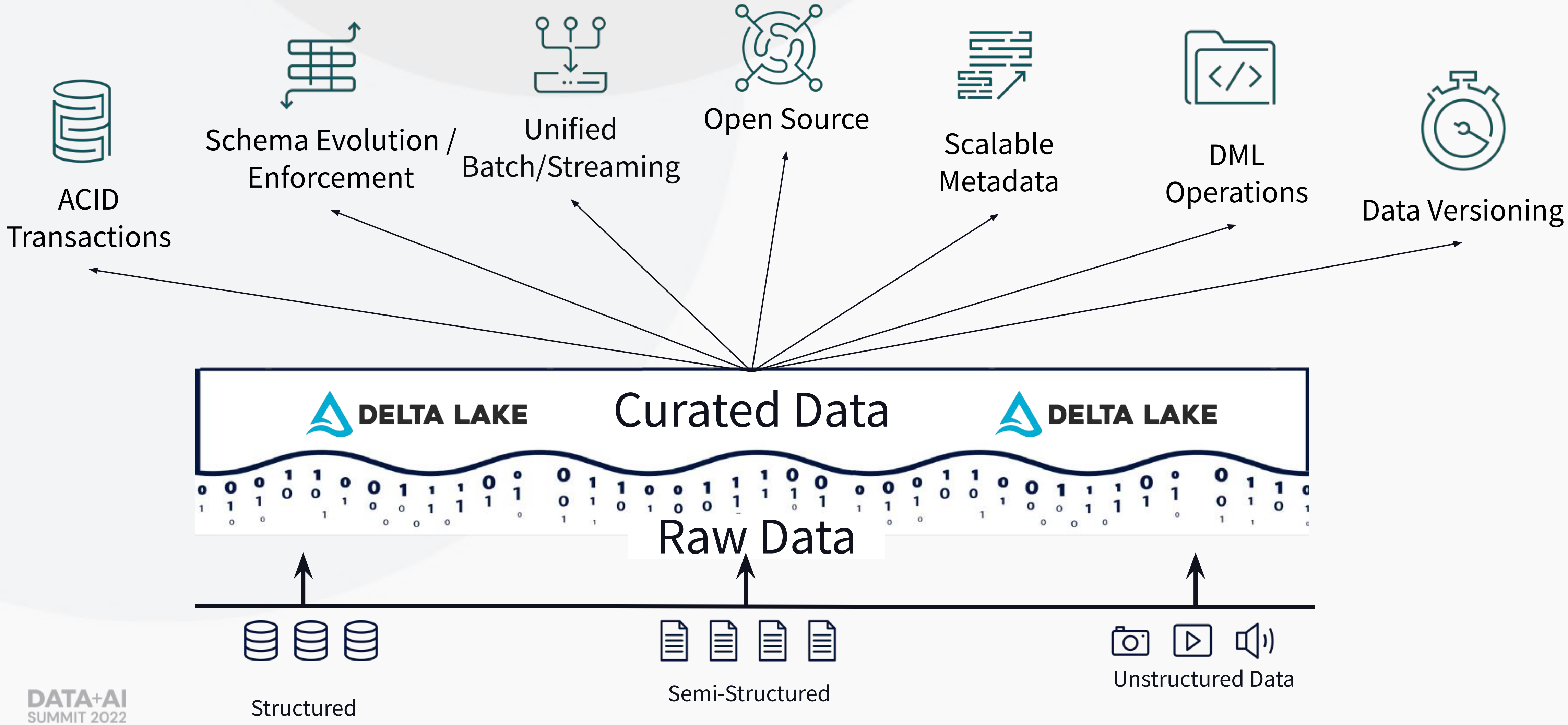
Delta Transaction logs

DELTA LAKE

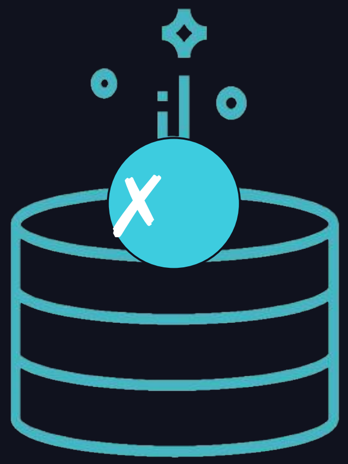
```
%fs ls dbfs:/Users/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log
```

		name	size
16	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000016.json	00000000000000000016.json	58080
17	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000017.crc	00000000000000000017.crc	95
18	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000017.json	00000000000000000017.json	21718
19	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000018.crc	00000000000000000018.crc	96
20	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000018.json	00000000000000000018.json	46743
21	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000019.crc	00000000000000000019.crc	96
22	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000019.json	00000000000000000019.json	58080
23	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000020.checkpoint.parquet	00000000000000000020.checkpoint.parquet	94341
24	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000020.crc	00000000000000000020.crc	97
25	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000020.json	00000000000000000020.json	347291
26	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000021.crc	00000000000000000021.crc	95
27	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/00000000000000000021.json	00000000000000000021.json	41461
28	s/vini.jaiswal@databricks.com/demo/customer_t2/_delta_log/_last_checkpoint	_last_checkpoint	26

Delta Lake Features



Data reliability challenge # 1



Failed production jobs leave data in corrupt state requiring tedious recovery

Example: Data Corruption

Run result unavailable: job failed with error message Unexpected failure while waiting for the cluster (0422-091004-4zud3ebj) to be ready.Cause Unexpected state for cluster (hhjj-02348-4zud3ebj): BOOTSTRAP_TIMEOUT(SUCCESS):[id: InstanceId(i-0f8a1c082d3aa434b), status: INSTANCE_INITIALIZING, workerEnvId:WorkerEnvId(workerenv-28425385-ldsajlf-34832-sdf33-fdg-ffvfbf), lastStatusChangeTime: 3794703740372, with threshold 700 seconds timed out after 704477 milliseconds. Please check network connectivity from the data plane to the control plane.,instance_id:i-7320kj2b3484

Data reliability challenge # 2

Lack of consistency makes it almost impossible to mix appends and reads, batch and streaming

Resolution of Consistency issues in Legacy Data Pipelines

- New rows to be inserted
- Rows that will be replaced
- Rows that are not impacted
- Create a new temp
- Delete the original table
- "Rename" the temp table
- Drop the temp table

How Delta Lake solves consistency and data corruption problems?

Transaction Log Commits

DELTA LAKE

Changes to the table are stored as *ordered, atomic commits*

Each commit is JSON file in _delta_log with a *set of actions*

```
|
+---- _delta_log/
          +---- 000.json
          +---- 001.json
          ...
```

INSERT actions

```
Add 001.parquet
Add 002.parquet
```

UPDATE actions

```
Remove 001.parquet
Remove 002.parquet
Add 003.parquet
```

Consistent Snapshots

DELTA LAKE

Readers read the log in atomic units thus reading consistent snapshots

```
|
+---- _delta_log/
    +---- 000.json
    +---- 001.json
    ...
```

```
Add 001.parquet
Add 002.parquet
```

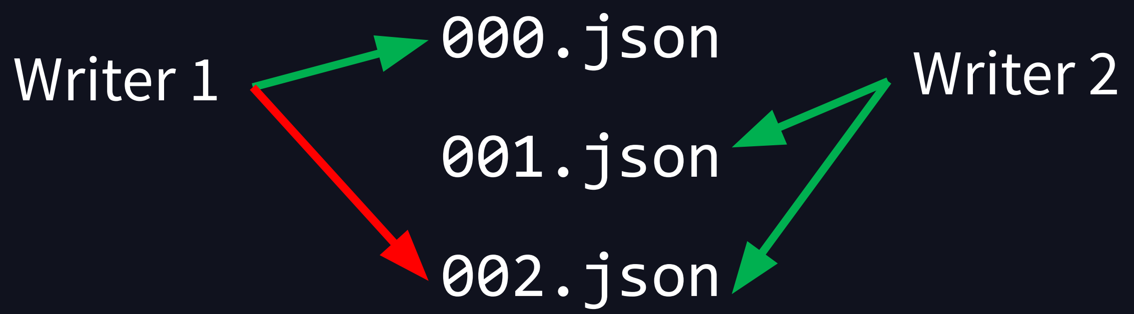
```
Remove 001.parquet
Remove 002.parquet
Add 003.parquet
```

readers will read
either [001+002].parquet
or 003.parquet
and nothing in-between

ACID via Mutual Exclusion on Log Commits

Concurrent writers need to agree on the order of changes (optimistic concurrency control)

New commit files must be created mutually exclusively using storage-specific API guarantees



only one of the writers trying to concurrently write 002.json must succeed

Data reliability challenge # 3

Lack of schema enforcement creates inconsistent and low quality data

Example: Lack of schema Enforcement in Parquet

```
stream_query = generate_and_append_data_stream(
    table_format = "parquet",
    table_path = parquet_path)
```

```
1  spark.sql("select * from loans_parquet").sho
```

▸ (1) Spark Jobs

```
+-------+-----------+---------+----------+
|loan_id|funded_amnt|paid_amnt|addr_state|
+-------+-----------+---------+----------+
|      0|       1000|   182.22|        CA|
|      1|       1000|   361.19|        WA|
|      2|       1000|   176.26|        TX|
|      3|       1000|   1000.0|        OK|
|      4|       1000|   249.98|        PA|
|      5|       1000|    408.6|        CA|
|      6|       1000|   1000.0|        MD|
|      7|       1000|   168.81|        OH|
|      8|       1000|   193.64|        TX|
|      9|       1000|   218.83|        CT|
|     10|       1000|   322.37|        NJ|
|     11|       1000|   400.61|        NY|
|     12|       1000|   1000.0|        FL|
|     13|       1000|   165.88|        NJ|
```

Total records = 14705

```
1  spark.read.format("parquet").load(parquet_path).show() # wrong schema!
```

▸ (2) Spark Jobs

```
+--------------------+-----+-------+-----------+-----------------+----------+
|           timestamp|value|loan_id|funded_amnt|        paid_amnt|addr_state|
+--------------------+-----+-------+-----------+-----------------+----------+
|2022-03-22 22:40:...|   36|  10036|       8894| 8544.701793101429|        CA|
|2022-03-22 22:40:...|   44|  10044|       7203| 7143.422329828758|        CA|
|2022-03-22 22:40:...|   52|  10052|       7113| 6134.901645020872|        TX|
|2022-03-22 22:40:...|   60|  10060|       5427| 4402.746113849142|        WA|
|2022-03-22 22:40:...|   68|  10068|       5850| 4549.965501260052|        NY|
|2022-03-22 22:40:...|   76|  10076|       7977| 6774.398332299471|        NY|
|2022-03-22 22:40:...|   84|  10084|       6576| 5119.430008447199|        WA|
|2022-03-22 22:40:...|   35|  10035|       5763|3947.6444236157877|        CA|
|2022-03-22 22:40:...|   43|  10043|       8847| 7344.280401545113|        TX|
|2022-03-22 22:40:...|   51|  10051|       5117| 4770.738672461943|        NY|
|2022-03-22 22:40:...|   59|  10059|       6742| 5882.987591904314|        TX|
|2022-03-22 22:40:...|   67|  10067|       7144| 6760.007491001625|        CA|
|2022-03-22 22:40:...|   75|  10075|       6755| 4809.795858246685|        TX|
|2022-03-22 22:40:...|   83|  10083|       5817| 4424.808390048687|        WA|
|2022-03-22 22:40:...|  136|  10136|       9237| 8319.033551236855|        NY|
|2022-03-22 22:40:...|  144|  10144|       6953|6538.8935002252965|        WA|
|2022-03-22 22:40:...|  152|  10152|       8212| 8174.558041588866|        NY|
|2022-03-22 22:40:...|  160|  10160|       6456| 6059.975786688777|        TX|
```

TOTAL RECORDS = 51

Where did the two new columns `timestamp` and `value` come from? And where did my existing rows go?

How does Delta Lake enforce schema?

Example of schema handling in DELTA LAKE

Intentional failure

```
rawDF = miniDataDF.filter("CustomerID=20993").withColumn("Customer__ID", lit("8"))

# Error expected as enforcement of schema
(rawDF.write.format("delta").mode("append").saveAsTable('customer_data_delta_mini'))
```

```
org.apache.spark.sql.AnalysisException: A schema mismatch detected when writing to the Delta table.
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
/databricks/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:

/databricks/spark/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in get_return_value(answer, gateway_client, target_id, name)
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:

Py4JJavaError: An error occurred while calling o404.saveAsTable.
: org.apache.spark.sql.AnalysisException: A schema mismatch detected when writing to the Delta table.
To enable schema migration, please set:
'.option("mergeSchema", "true")'.
```

Schema Evolution with DELTA LAKE

- Schema evolution allows users to easily change a table’s current schema to accommodate data that is changing over time.
- Most commonly used operations for
  - append
  - overwrite
- Use `.option('mergeSchema', 'true')` to your `.write` or `.writeStream` Spark command.

Performance features

DELTA LAKE

Data Skipping

DELTA LAKE

- Column min/max values automatically collected when writing files and committing to log
- Read queries can skip files using min/max values

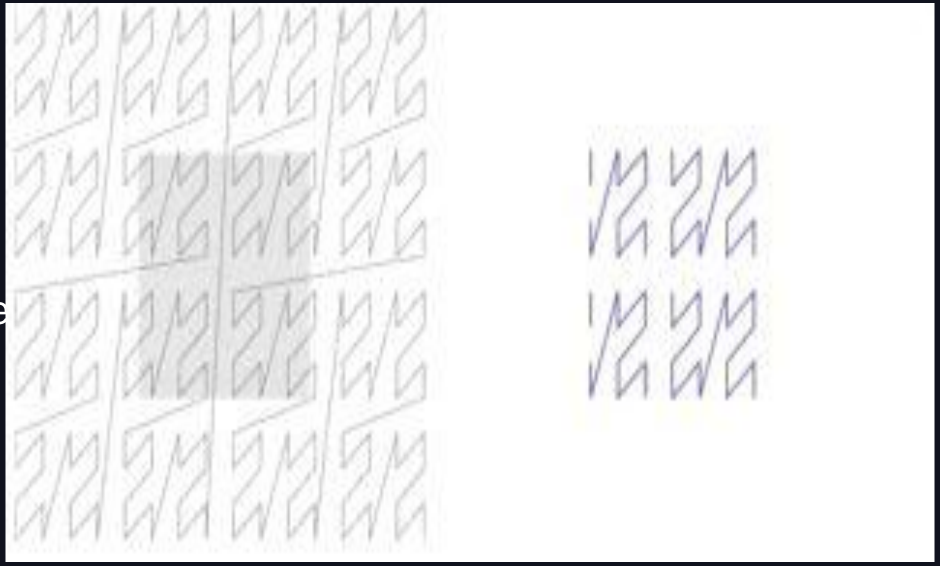
```
SELECT * FROM events
WHERE year=2020 AND uid=24000
```

~~file1.parquet~~
year: min 2018, max 2019
uid: min 12000, max 23000

~~file2.parquet~~
year: min 2018, max 2020
uid: min 12000, max 14000

skipped as data range outside selected value

file3.parquet
year: min 2020, max 2020
uid: min 23000, max 25000



Generated Columns

- Automatically generate data for new columns using any expression on other columns
- Compliant with SQL standards
- Can be used for partitioning or bucketing
- Automatic filter generation and data skipping

id	idBucket	eventTime	*eventDate*
1234	34	2021-05-24 09:00:00.000	2021-05-24

```sql
CREATE TABLE events(
    id bigint,
    idBucket bigint GENERATED ALWAYS AS (
      id % 100
    ),
    eventTime timestamp,
    eventDate date GENERATED ALWAYS AS (
      CAST(eventTime AS DATE)
    )
)
USING delta
PARTITIONED BY (eventDate, idBucket)
```

```sql
... WHERE eventTime < '2021-05-24
09:00:00.000'
```

generate
extra filter

```sql
... WHERE eventTime < '2021-05-24 09:00:00.000’
  AND eventDate < '2021-05-24'
```

Z-Ordering

DELTA LAKE

Optimize data layout with Z-order

Multi-column data clustering that is better than simple multi-column sorting

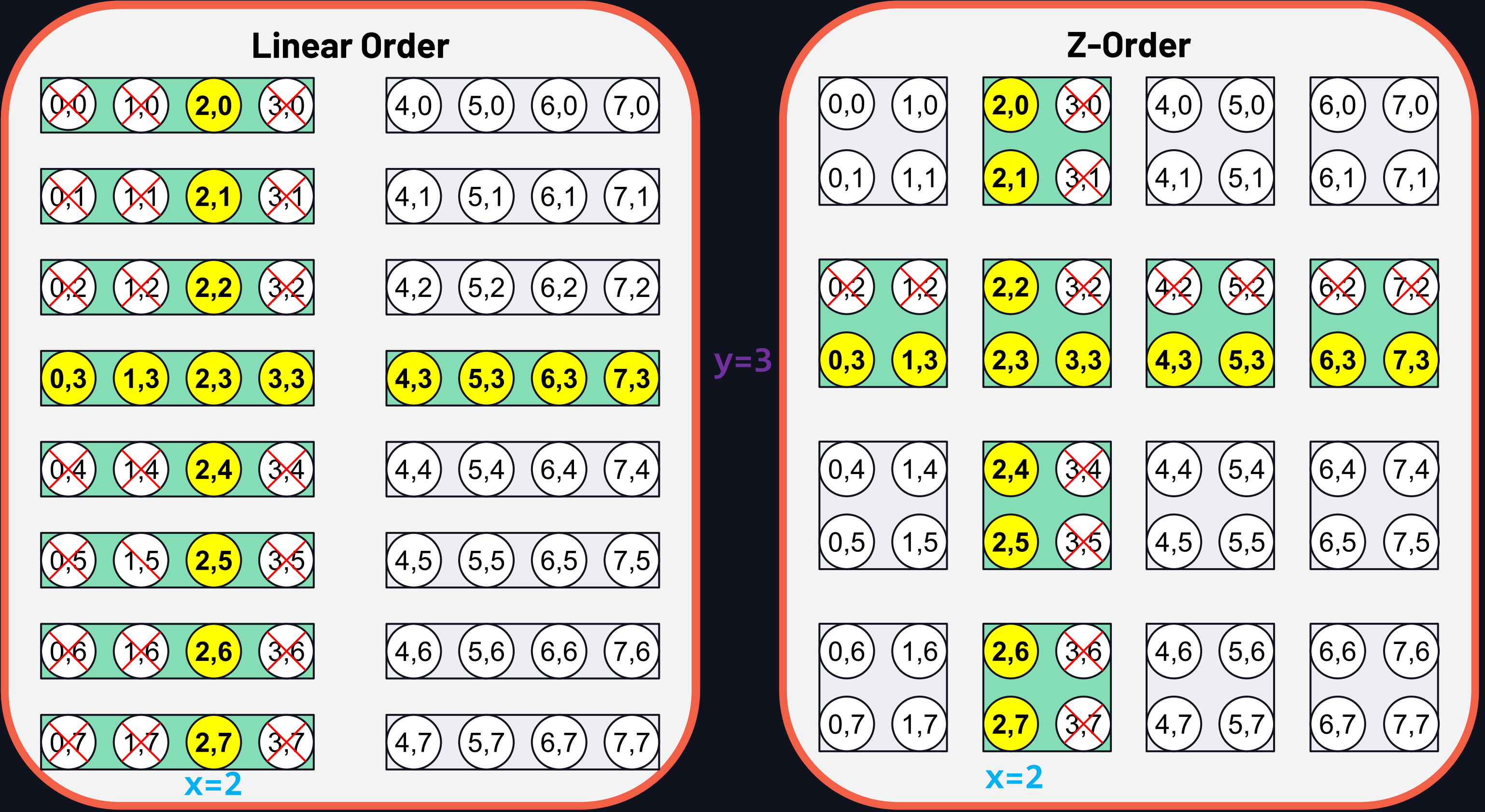
```
OPTIMIZE deltaTable
ZORDER BY (x, y)
```

With column stats, this enables better data skipping leading to faster queries

```
SELECT * FROM points
WHERE x = 2 OR y = 3
```

9 files scanned in total 👎
21 false positives 👎

7 files scanned in total 👍
13 false positives 👍



Data Versioning with DELTA LAKE

- Audit
- Reproduce Experiments
- Rollbacks

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

```
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"}

Pace of innovation

Delta Lake Pace of Innovation Highlights

Apr 2019 (0.1)

- Open Source Delta Lake
- ACID Transactions
- Schema Management
- Scalable Metadata Handling
- Time Travel
- Unified Batch and Streaming

Sep 2019 (0.4)

- Scala/Java APIs for DML commands
- Scala/Java APIs for query commit history
- Scala/Java APIs for vacuuming old files
- Python DML APIs
- Convert-to-Delta
- Python and SQL utility operations

Dec 2019 (0.5)

- Support for other processing engines (via manifest)
- Improved concurrency
- Improved file compaction
- Improved insert-only merge performance
- Convert-to-Delta using SQL
- Experimental Snowflake and Redshift Spectrum support

Apr 2020 (0.6)

- Support for schema evolution in merge operations
- Improved merge performance with automatic repartitioning
- Improved merge performance with no insert clause
- Operation metrics in DESCRIBE HISTORY
- Support for reading Delta tables from any file system

Jun 2020 (0.7)

- Support for defining tables in the Hive metastore
- Support for SQL DML
- Support for automatic and incremental Presto/Athena manifest generation
- Support for controlling the retention of the table history
- Support for adding user-defined metadata in Delta table commits
- Support Azure Data Lake Storage Gen2
- Improved support for streaming one-time triggers

Feb 2021 (0.8)

- Unlimited MATCHED and NOT MATCHED clauses for merge operations (Scala, Java, Python)
- MERGE operation now supports schema evolution of nested columns
- MERGE INTO and UPDATE operations now resolve nested struct columns by name
- Check constraints on Delta tables
- Start streaming a table from a specific version
- Ability to perform parallel deletes with VACUUM
- Use Scala implicits to simplify read and write APIs

May 2021 (1.0)

- Generated Columns
- Multi-cluster writes
- Cloud Independence
- Spark 3.1 Support
- PyPI Install
- Delta Everywhere

Dec 2021 (1.1)

- Delta Standalone Writer
- Delta Sink for Apache Flink
- Delta Source for PrestoDB
- Delta Source for Apache Pulsar
- Performance improvements in MERGE operation
- Support for Generated Columns in MERGE operation

April 2022 (1.2)

- Compacting small files (optimize) into larger files
- Data skipping using column statistics.
- S3 multi-cluster writes
- Table restore to an earlier version.
- Column renaming without rewriting files.
- Arbitrary characters support in column names
- Automatic data skipping using generated columns.
- Google Cloud storage support - GA

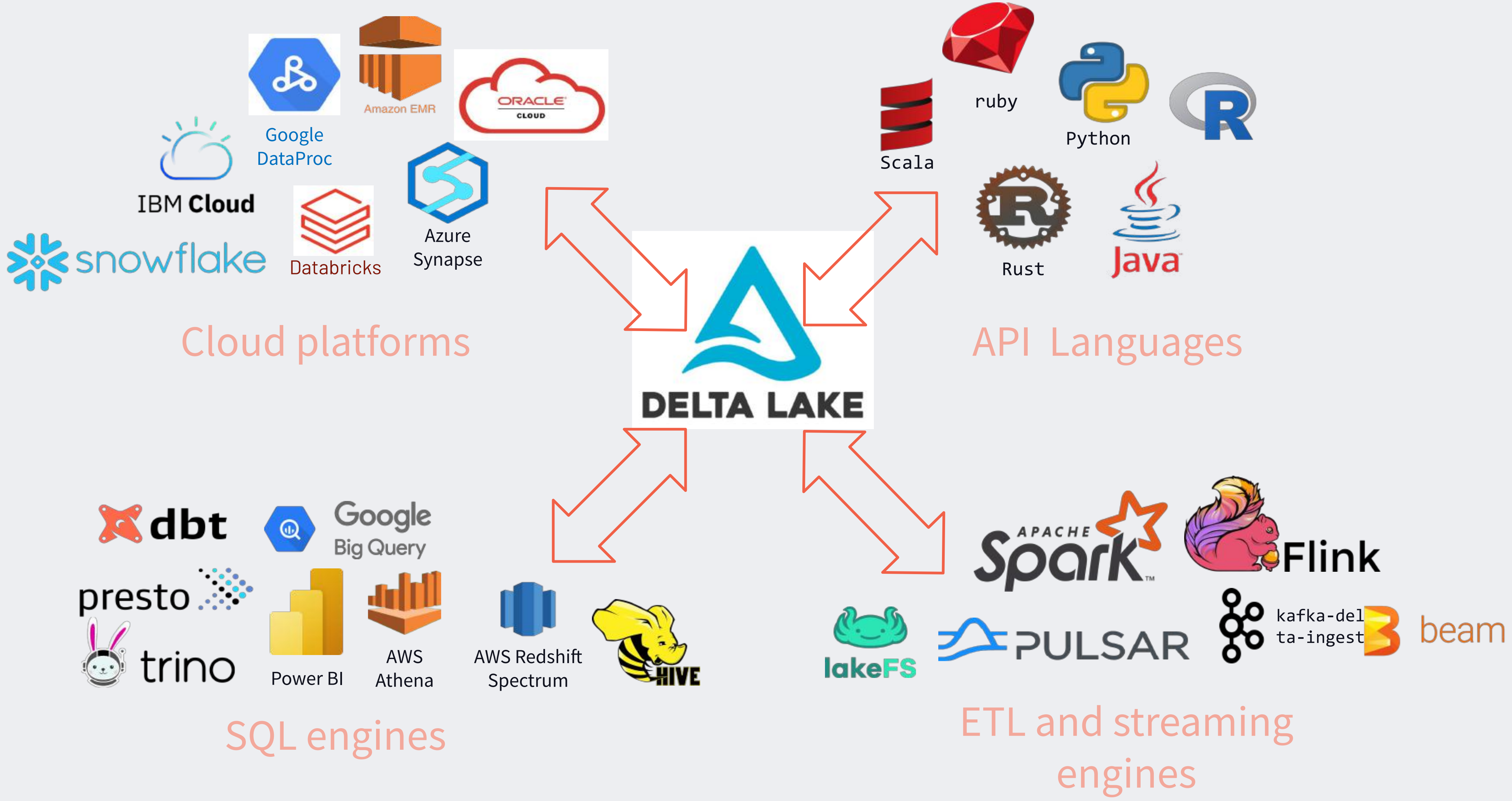
Delta Lake 2.0

Unlock the power of Delta Lake

Coming Soon!

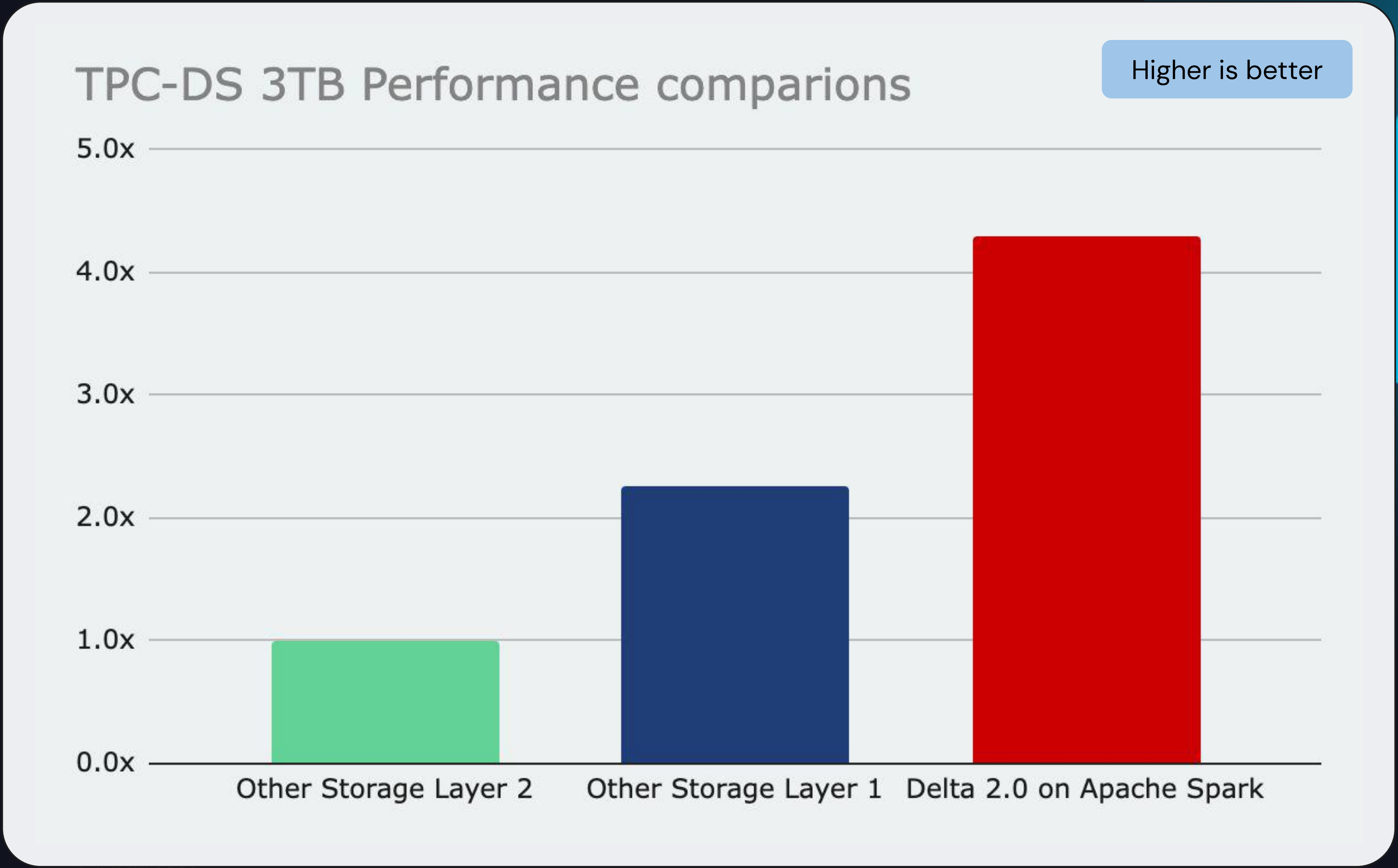
ACID Transactions	Scalable Metadata	Time Travel	Schema Evolution	OPTIMIZE	OPTIMIZE ZORDER	Change data feed	Generated column support w/ partitioning	Clones
Unified Batch/Streaming	Schema Enforcement	Audit History	DML Operations	Table Restore	S3 Multi-cluster writes	Data Skipping via Column Stats	Identity 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	Deletion Vectors

Delta Lake Ecosystem



Delta Lake performance

TPC-DS Benchmark Comparison (Higher is better)



Delta Lake is

1.9x faster than Storage Format 1

4.3x faster than Storage Format 2

Performance Optimizations roadmap

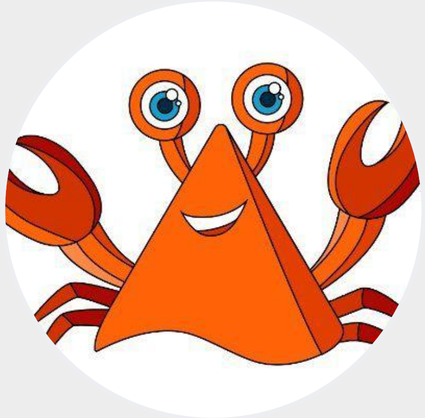
https://github.com/delta-io/delta/issues/920

Issue	Description	Target CY2022
927	**OPTIMIZE (file compaction)**: Table optimize is an operation to rearrange the data and/or metadata to speed up queries and/or reduce the metadata size	Released in 1.2
923	**File skipping using columns stats**: This is a performance optimization that aims at speeding up queries that contain filters (WHERE clauses) on non-partitionBy columns.	Released in 1.2
931	**Automatic data skipping using generated columns**: Enhance generated columns to include automatic data skipping	Released in 1.2
1134	**OPTIMIZE ZORDER**: Data clustering via multi-column locality-preserving space-filling curves with offline sorting.	Q3/Q4
	MERGE Performance Improvements: We will be providing a project improvement plan (PIP) document shortly on the proposed design for discussion.	Q2/Q3

DELTA LAKE

Community

Adoption and Social Channels



Adoption of Delta Lake

databricks
Tencent 腾讯
COMCAST
Alibaba Group
VIACOM
JAM CITY
Informatica
Booz | Allen | Hamilton
CONDÉ NAST
talend
ebay
DOLLAR SHAVE CLUB
edmunds
wehkamp
EVERQUOTE
tableau
acorns
McAfee
Together is power.
upwork
SCRIBD
ATTUNITY
Mc Graw Hill Education

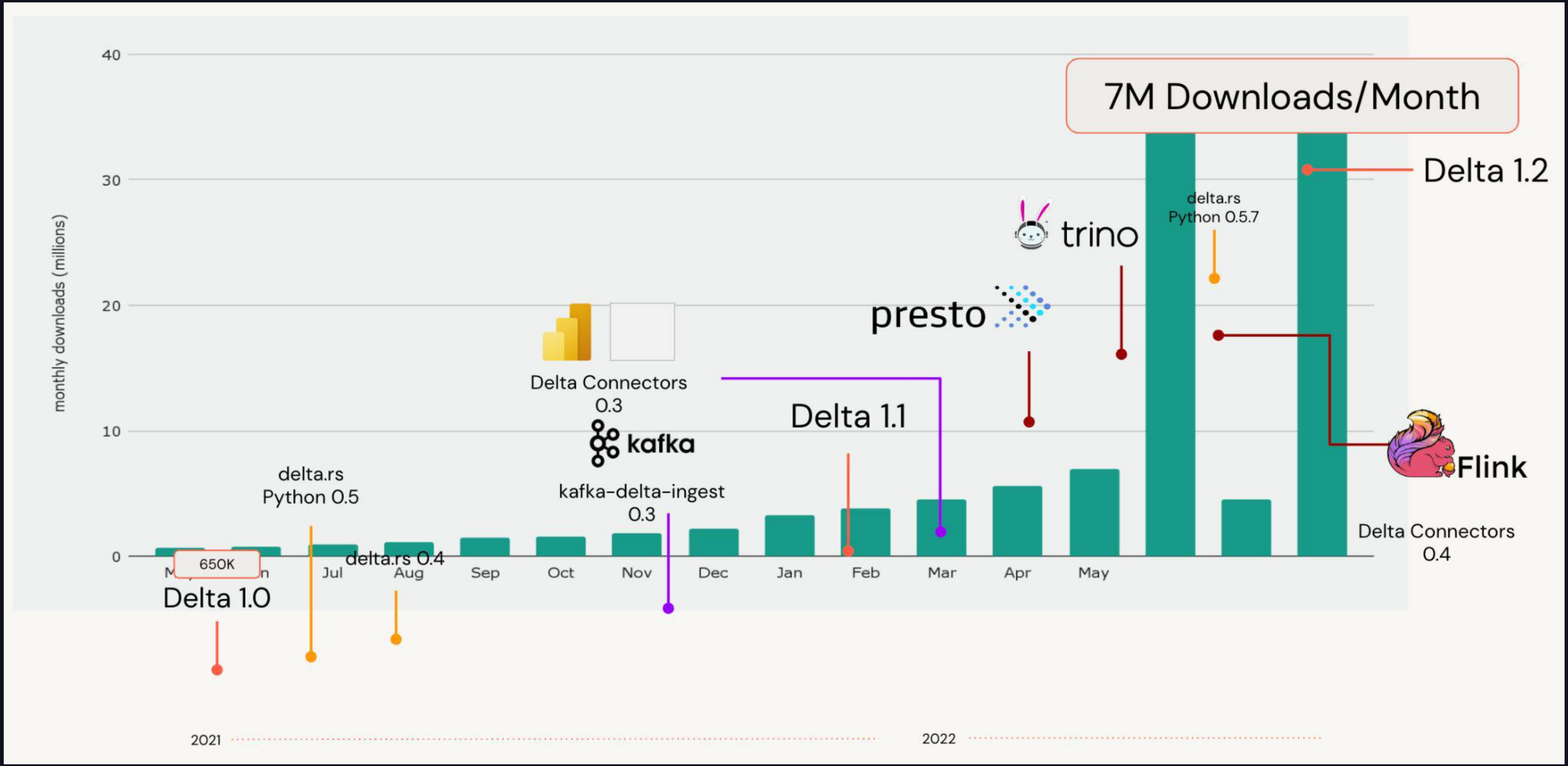
The Foundation of Lakehouse

1+ Exabytes
Processed / day

6000+
Slack Members

50+
Companies Contributing

The most widely used lakehouse format in the world



Engage with Delta Lake community

DELTA LAKE
delta.io

delta-users
Slack

Delta Lake
YouTube channel

delta-users
Google Group

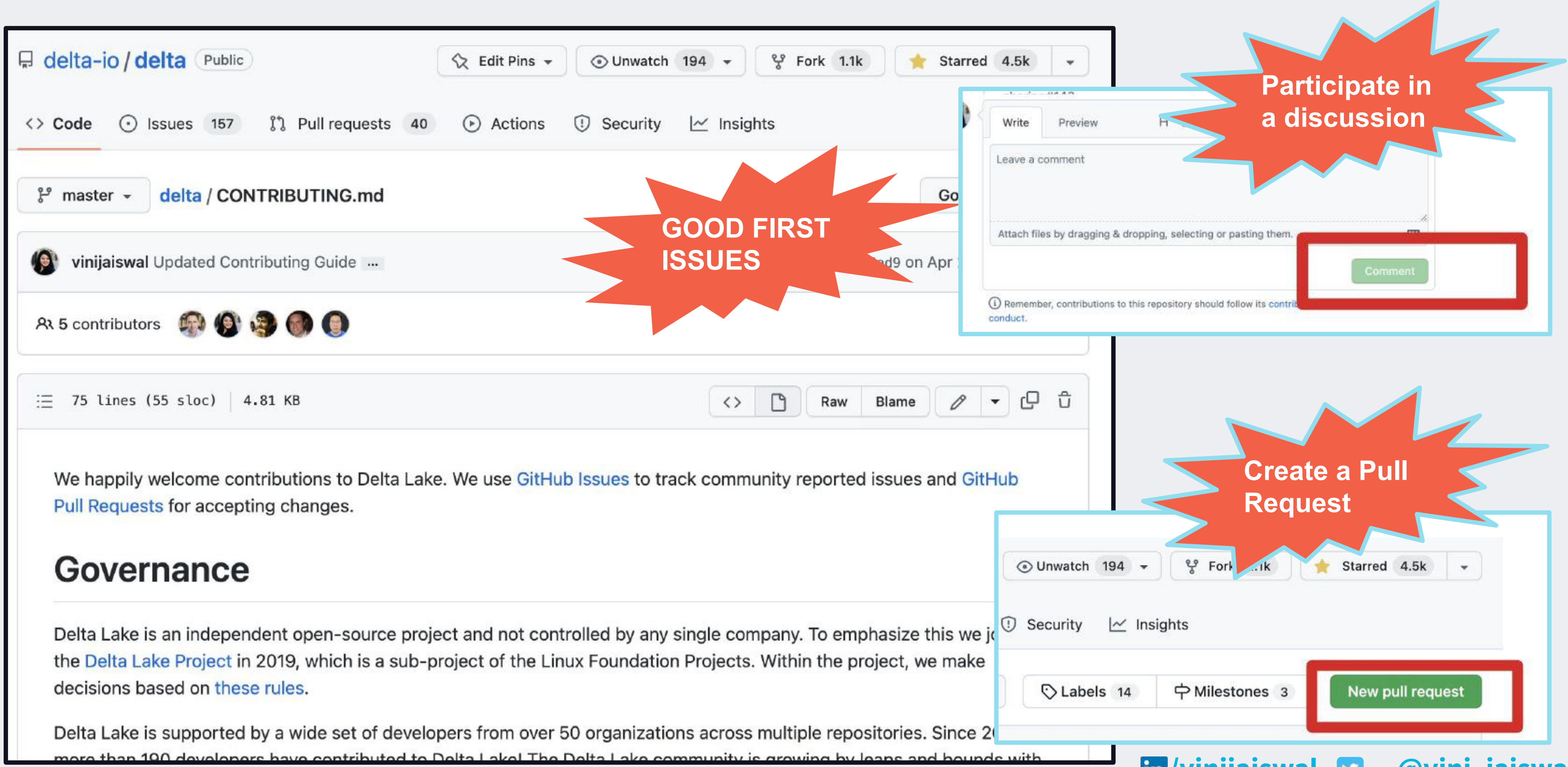
Delta Lake
GitHub Issues

Delta Lake
Linkedin

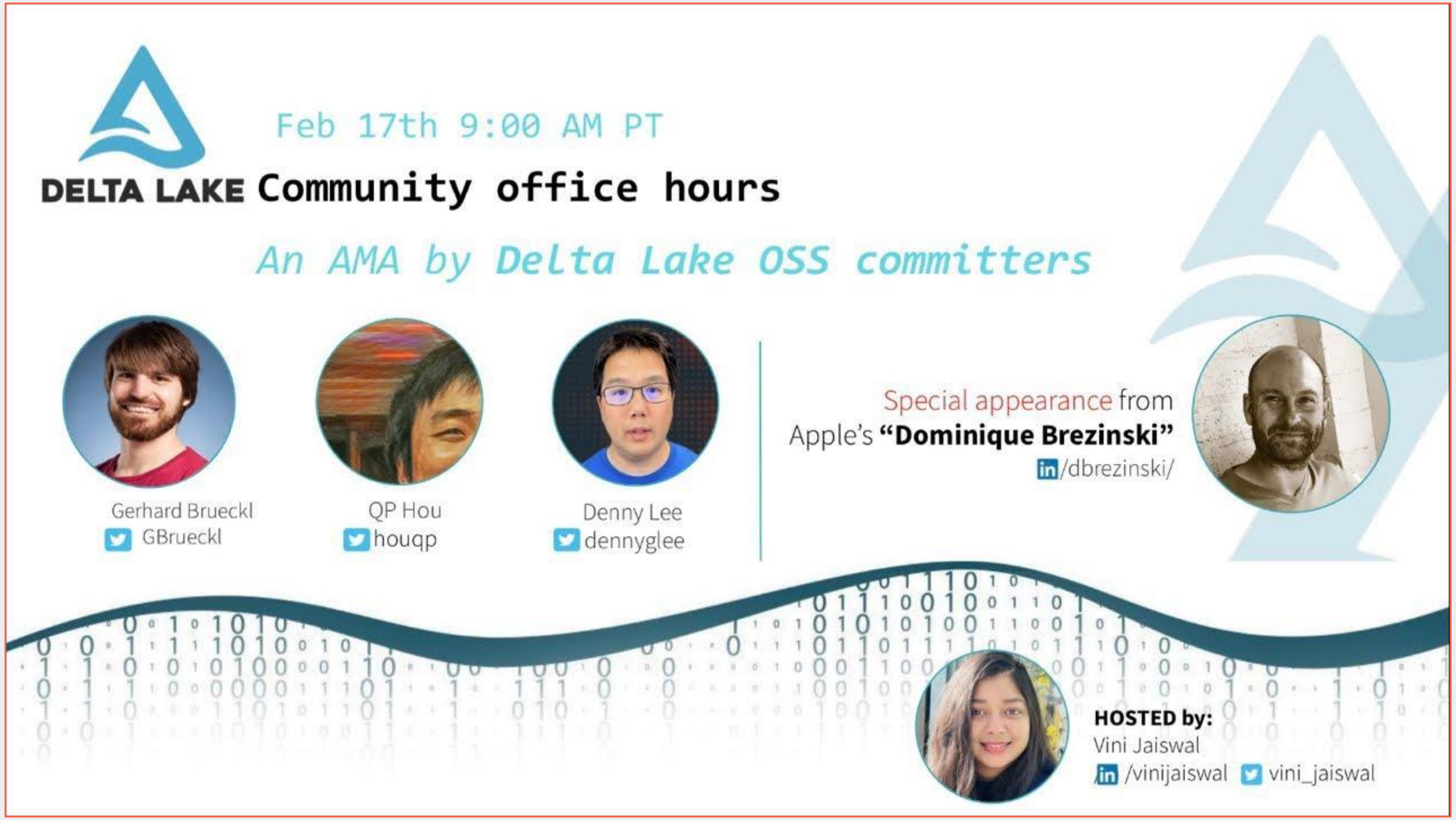
Meetup
data-ai-online

Contributing to the Project

https://github.com/delta-io/delta/blob/master/CONTRIBUTING.md



Have questions, join our community AMAs every two weeks



DATA+AI SUMMIT 2022

Mascone Center, San Francisco

OR

Virtually for Free!

Come join the Delta Community at the Data and AI Summit

- We have exciting line up of technical sessions and events.
- You might also get a chance to meet some of the creators!!!

17+
Technical Sessions
Delta Lake - Use Cases and Deep Dives

Ask Me Anything
Delta Lake Panels

Panel 1: June 28
10:30 AM PST
Panel 2: *June 29*
11:40 AM PST

Keynote
Delta Lake 2.0

June 28
10:30 AM PST

DELTA LAKE

Thank you!

in /vinijaiswal @vini_jaiswal