

Gonna Go Back in Time with Apache Iceberg

Daniel Hinojosa

{APICONF}

December 1-3 · Clearwater, FL

Register by August 15th - Save \$300!

[EVENT DETAILS](#)

DEVOPS<>VISION

December 1-2 · Clearwater, FL

Register by September 1st - Save \$400!

[EVENT DETAILS](#)

TECHLEADER SUMMIT

December 3-5 · Clearwater, FL

Register by August 1st - Save \$250!

[EVENT DETAILS](#)

ARCHCONF2025

December 8-11 · Clearwater, FL

Register by August 1st - Save \$250!

[EVENT DETAILS](#)

Conferences

Great Lakes Software Symposium

August 1 - 2, 2025

Central Iowa Software Symposium

September 5 - 6, 2025

Twin Cities Software Symposium

September 12 - 13, 2025

Northern Virginia Software Symposium

September 19 - 20, 2025

Pacific Northwest Software Symposium

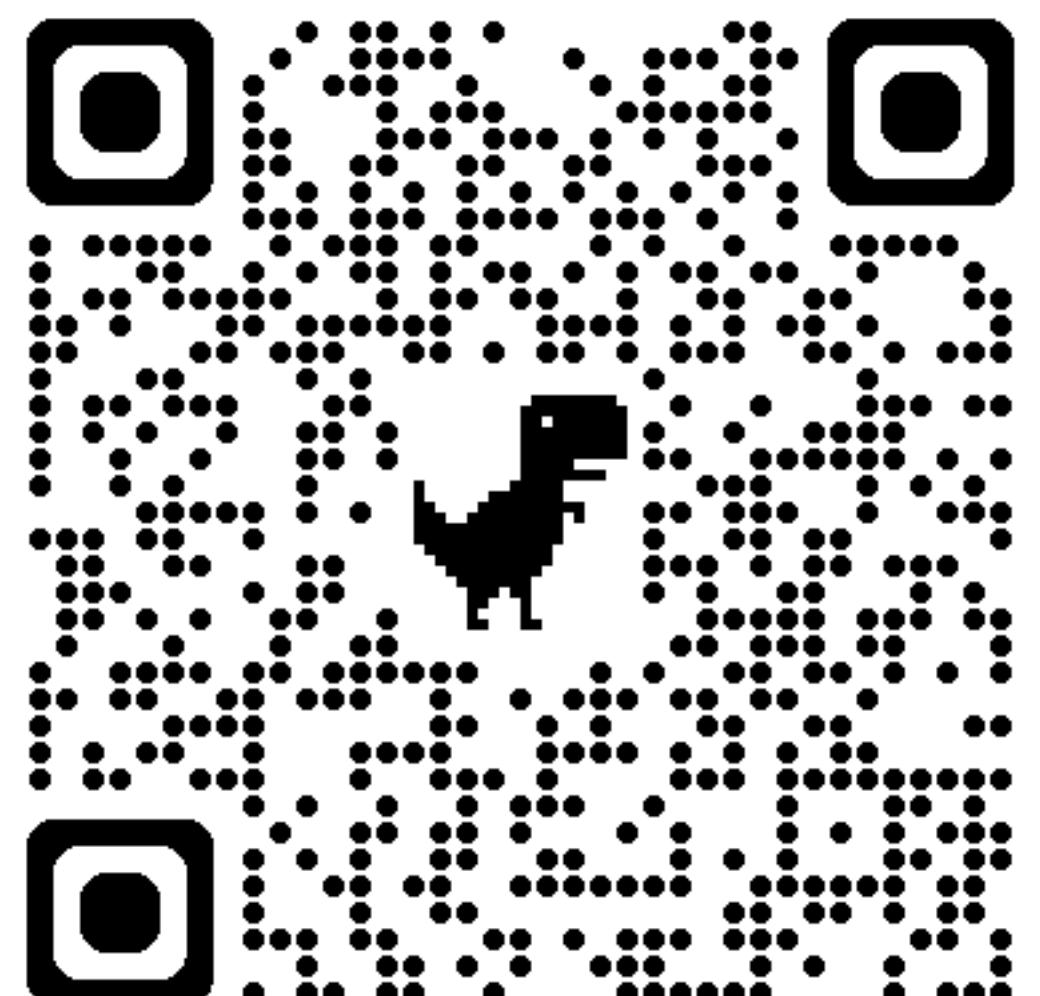
October 10 - 11, 2025

In this Presentation

- History of Data Storage
- Object Storage
- Introduction to Iceberg
- Big Data Infrastructure
- Demo of Iceberg
- Compaction
- Deletions

Slides and Material:

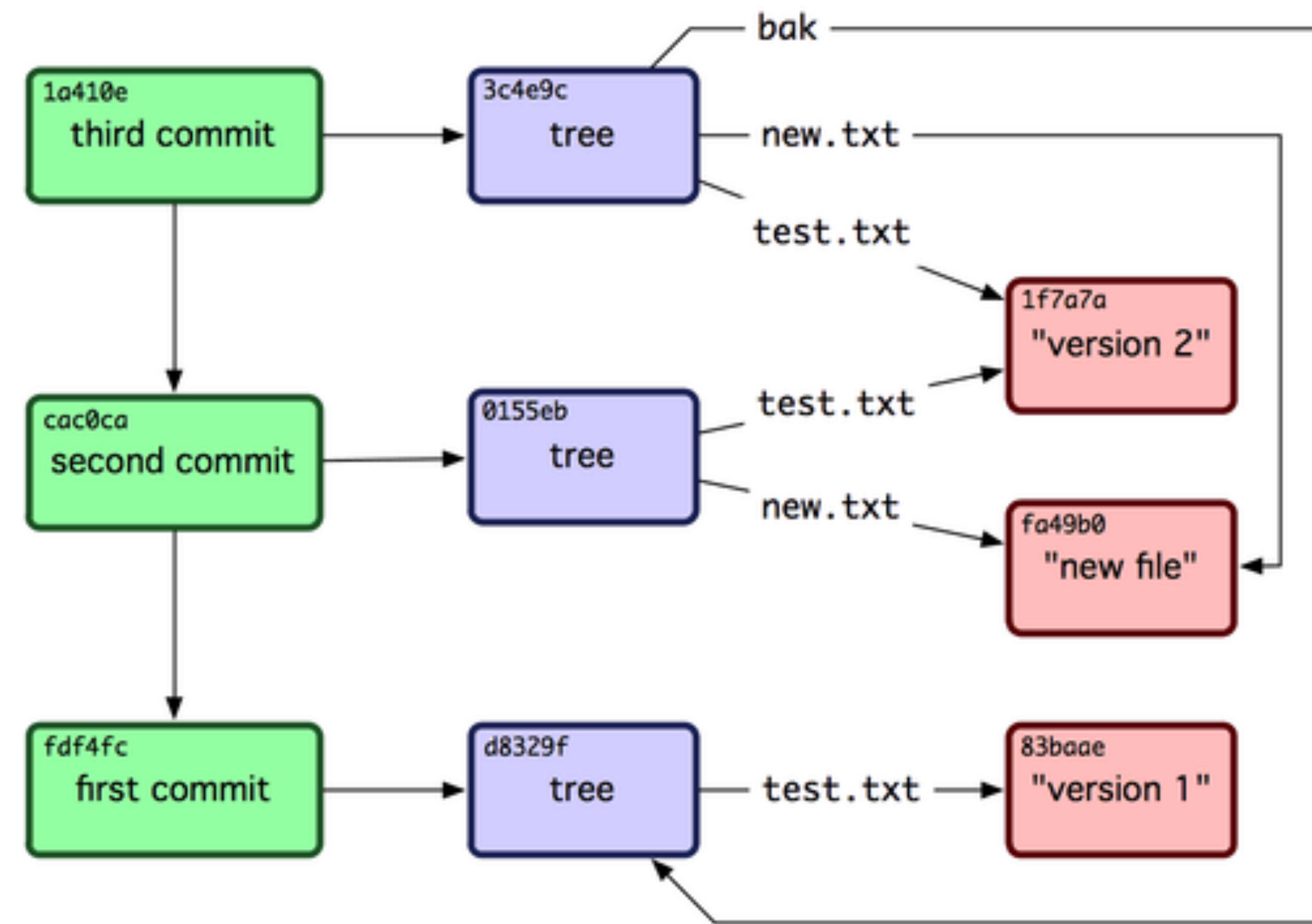
<https://github.com/dhinojosa/nfjs-iceberg-presentation>



A Familiar Pattern



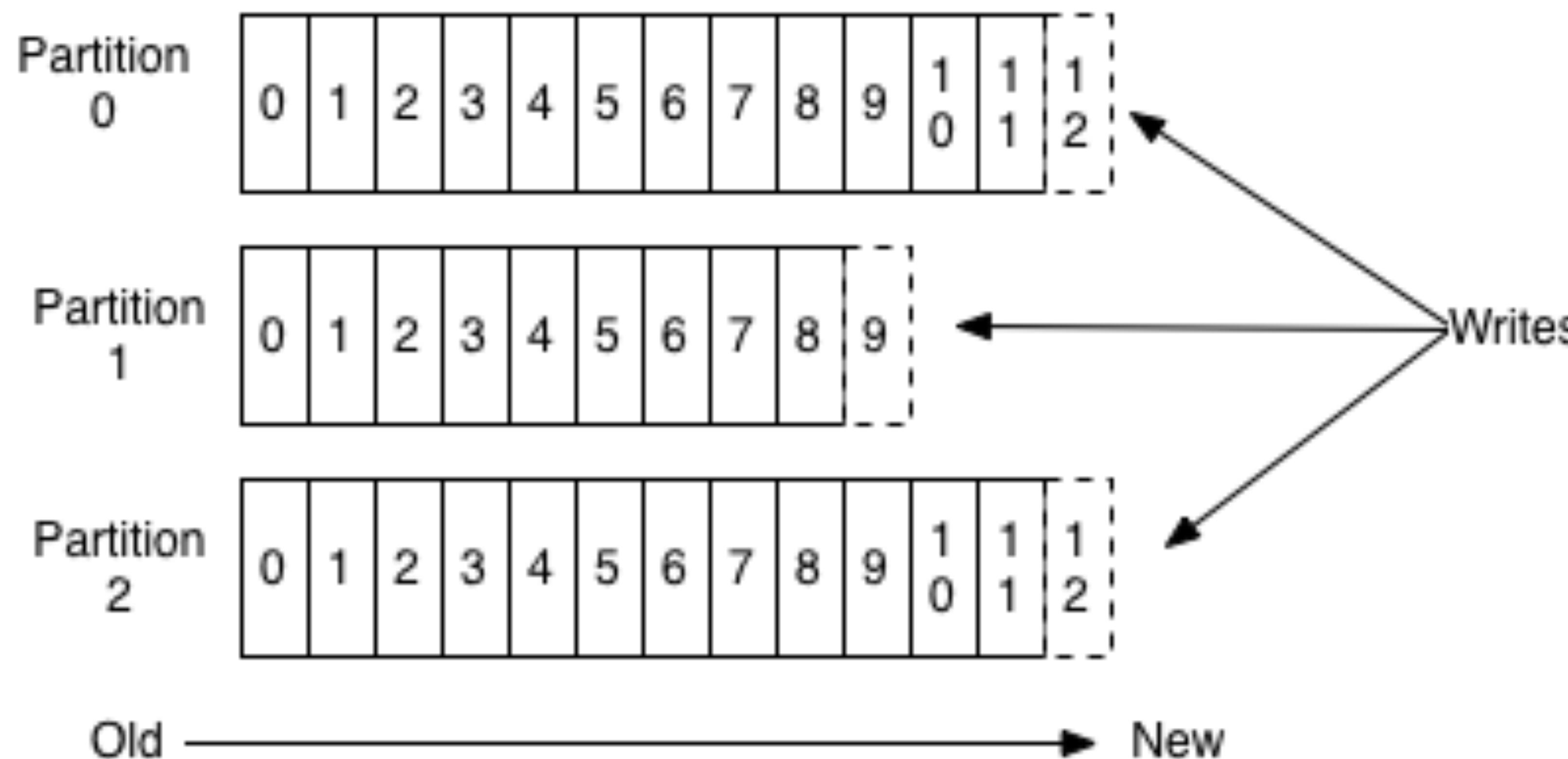




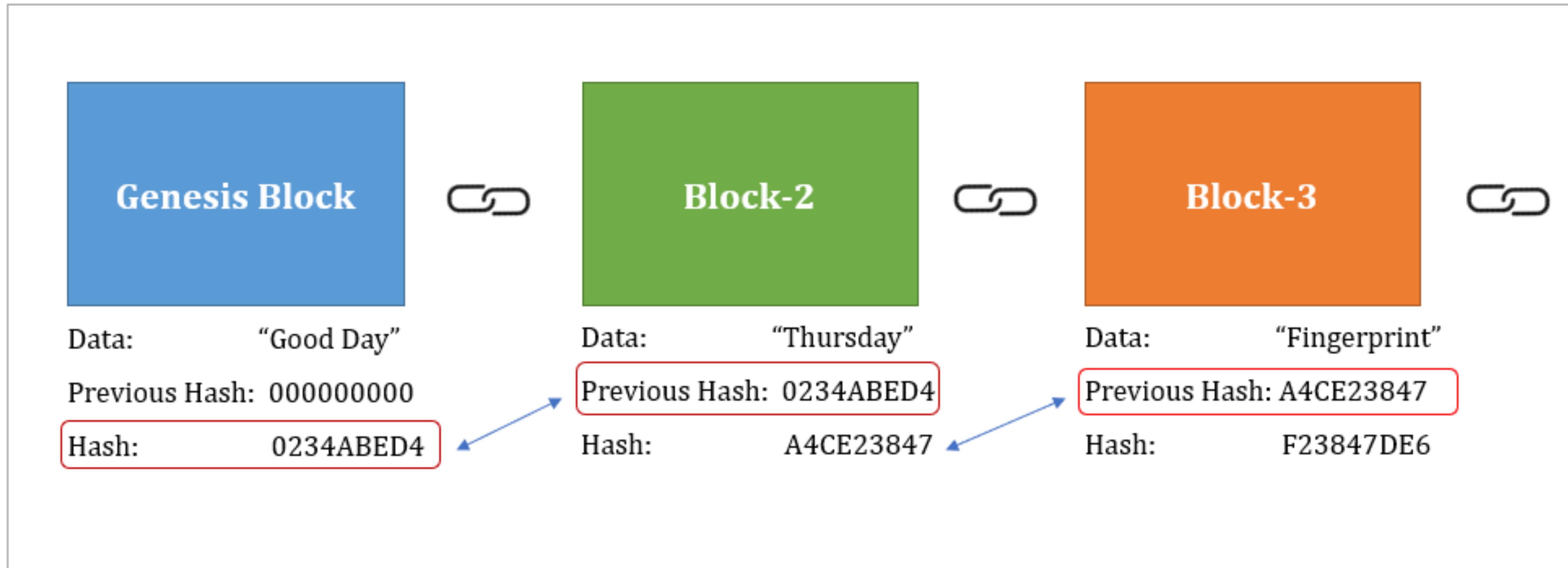


kafka

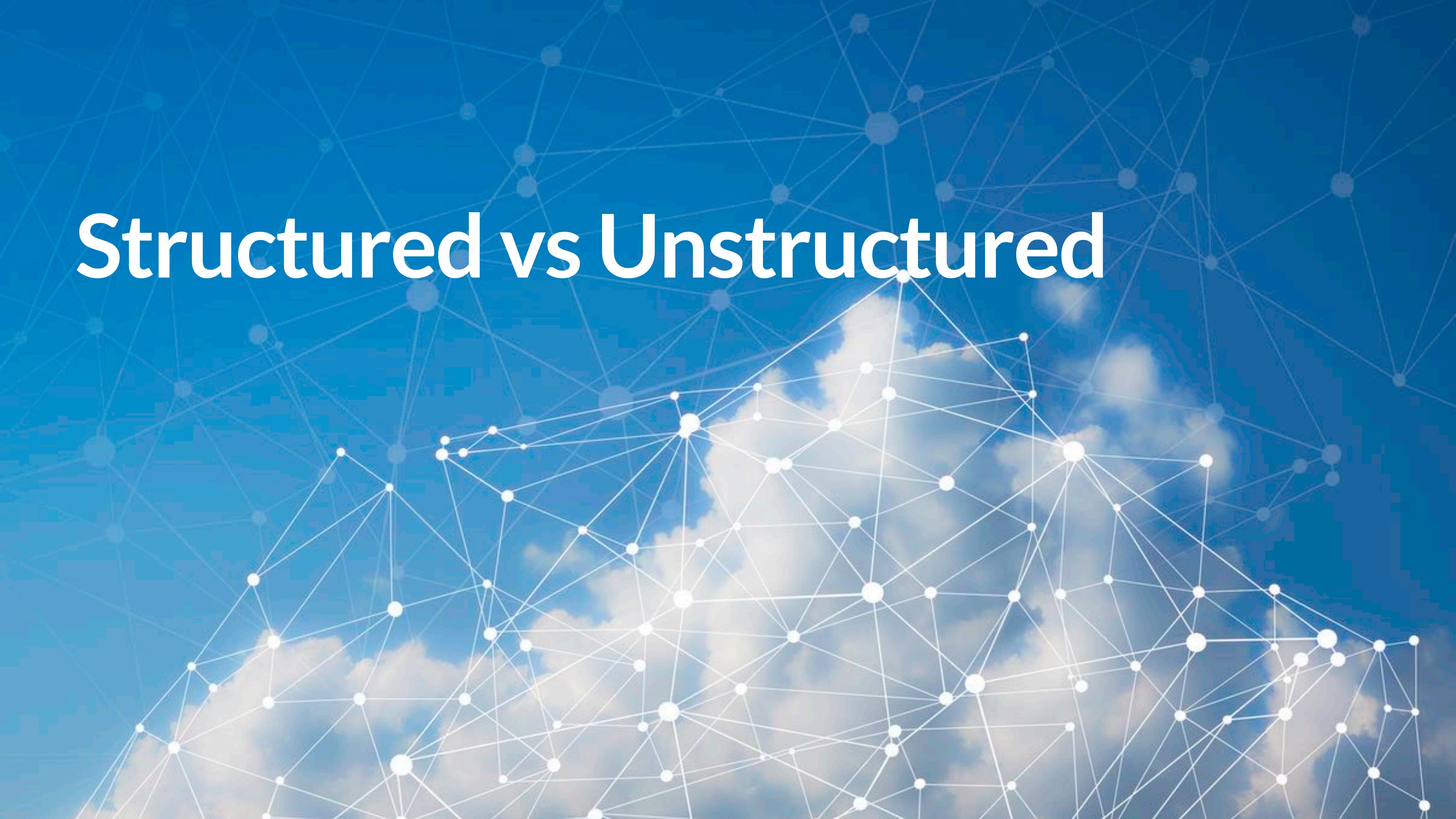
Anatomy of a Topic



Blockchain



Structured vs Unstructured



Structured vs Unstructured Data

- Structured:

- Data that conforms to a predefined schema or structure, making it easily searchable and queryable.
- The schema defines the types, relationships, and constraints of the data.
- Can exist in non-tabular formats like JSON, XML, or Avro if a schema is defined.

- Unstructured Data:

- Data that lacks a predefined schema or consistent organization.
- Examples: Text files, images, videos, audio files, and logs without consistent structure.

Structured vs Unstructured Data

Unstructured Logging

```
6:01:00 accepted connection on port 80 from 10.0.0.3:63349
6:01:03 basic authentication accepted for user foo
6:01:15 processing request for /super/slow/server
6:01:18 request succeeded, sent response code 200
6:01:19 closed connection to 10.0.0.3:63349
```

Structured Logging

```
time="6:01:00" msg="accepted connection" port="80" authority="10.0.0.3:63349"
time="6:01:03" msg="basic authentication accepted" user="foo"
time="6:01:15" msg="processing request" path="/super/slow/server"
time="6:01:18" msg="sent response code" status="200"
time="6:01:19" msg="closed connection" authority="10.0.0.3:63349"
```

Structured vs Unstructured Data

Structured vs. Unstructured

Aspect	Structured Data	Unstructured Data
Storage	SQL/NoSQL databases (e.g., MySQL, JSON, Parquet)	Object stores or file systems (e.g., S3, HDFS)
Querying	SQL or tools like Presto/Spark with schema	Advanced NLP, computer vision, or ML
Format	Tabular, JSON, XML with schema	Free-form (e.g., raw text, videos, images)
Schema Dependency	Schema-defined before or during use	No schema or schema inferred dynamically
Cost to Store	Higher per unit due to indexing, schema enforcement, and redundancy requirements	Lower per unit because raw data requires minimal processing or indexing

History of Data Storage

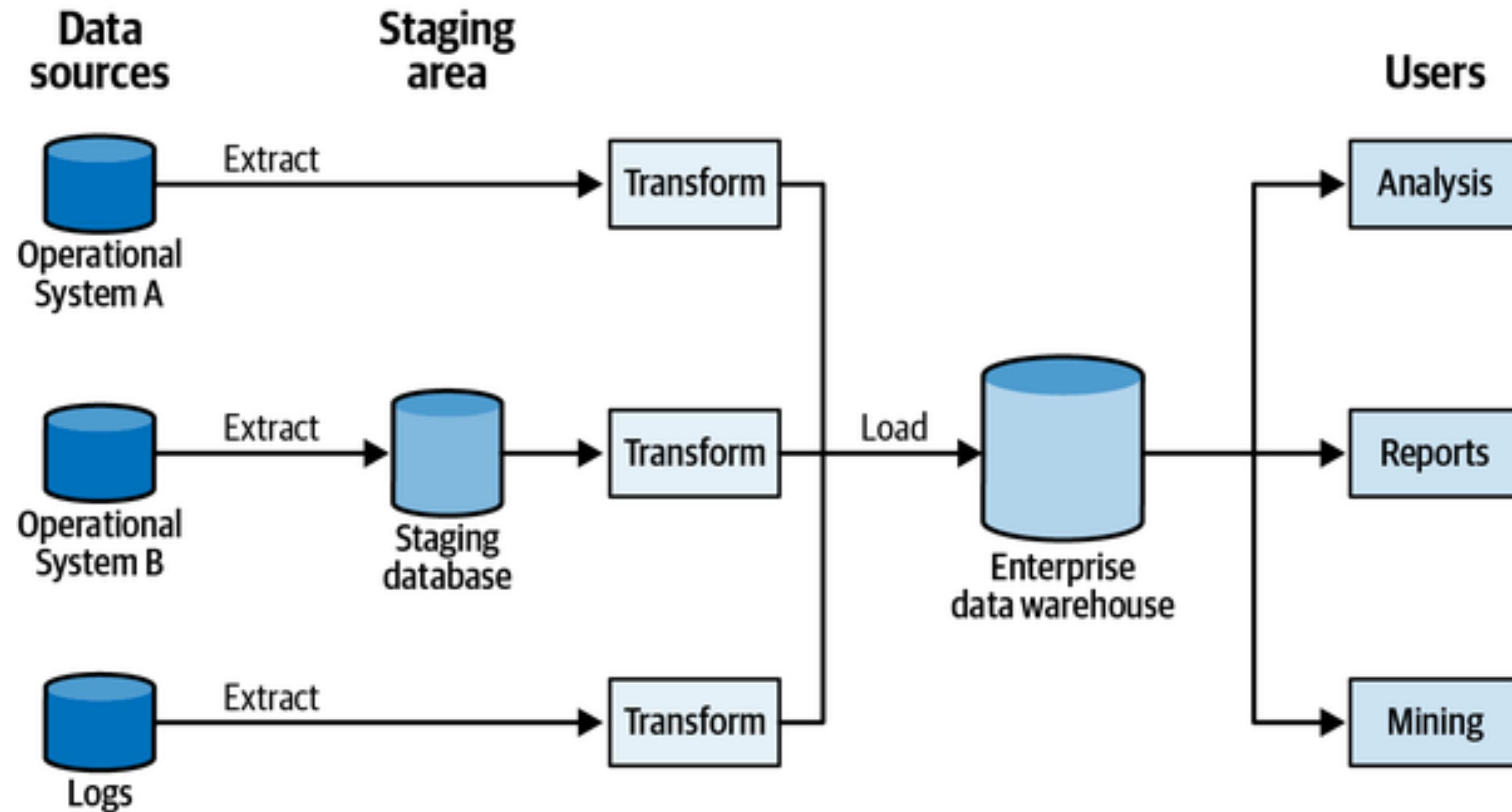


Data Warehousing

Data Warehousing

- A Data Warehouse, a centralized repository for *structured data*, designed for querying and analysis
- Consolidates data from multiple sources.
- **Schema-on-Write: Data schema is applied when stored.**
- Optimized for read-heavy analytical workloads.
- Supports historical analysis through snapshot storage.
- Single source of truth for analytics.
- Enables complex reporting and BI tools.
- Provides pre-aggregated, summarized data for fast querying.

The Diagram



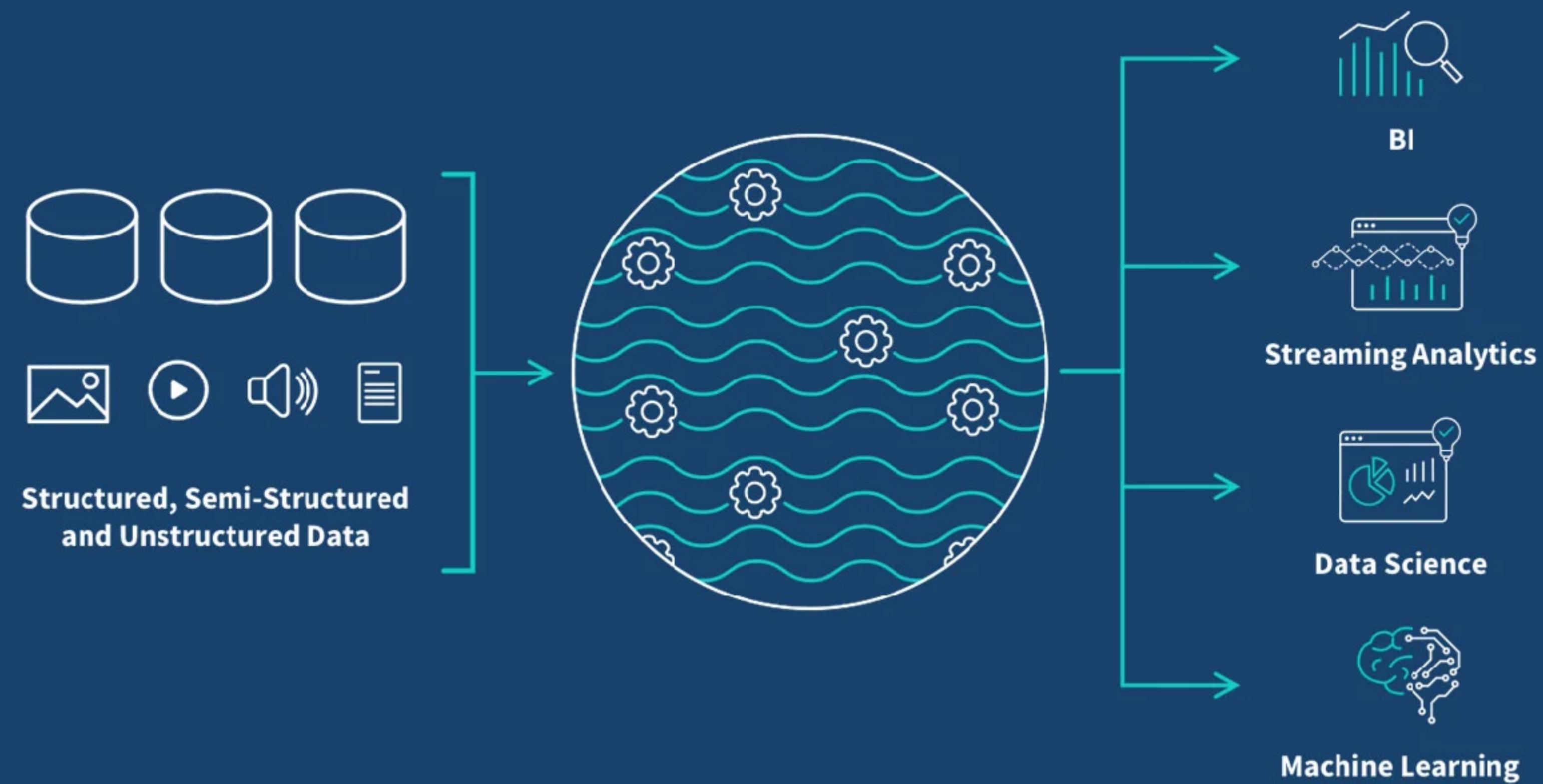
What was the problem?

- Large Storage, expensive operational costs
- Difficult Schema Evolution
- Batch Oriented Pipelines made real-time analysis difficult
- High Cost & Vendor Lock In
- Scaling Issues
- You used your vendor's compute engine

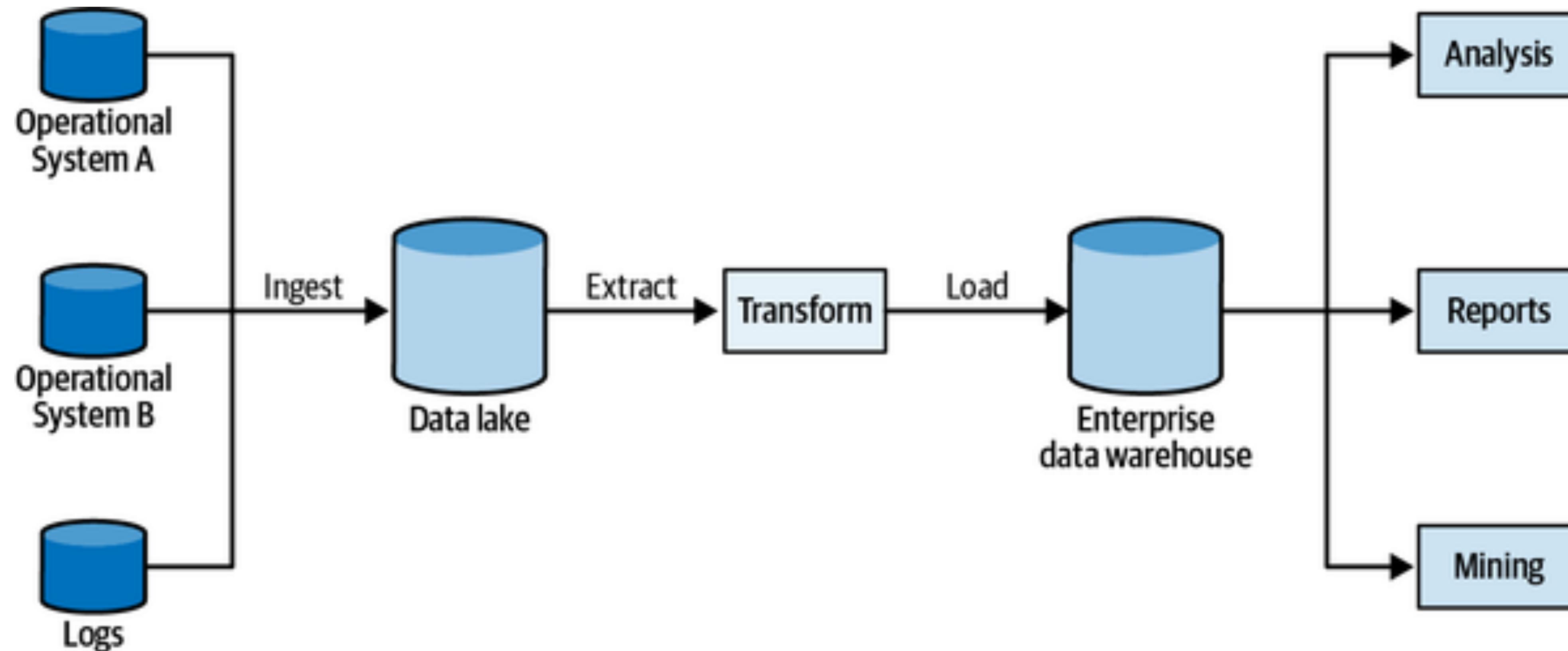
Data Lakes

Data Lakes

- A Data lake is a unified repository for all data
- A data lake is a centralized repository that stores raw, unprocessed data in its native format.
- **Store everything:** Handles structured, semi-structured, and unstructured data.
- **Schema-on-Read:** Data schema is applied when queried, not when stored.
- **Scalability:** Designed to scale with increasing data volumes.
- **Cost-Effective:** Often built on cheap, scalable storage like object stores (e.g., Amazon S3).
- You use whatever engine you would like: Spark, Flink, Trino, Presto, etc.

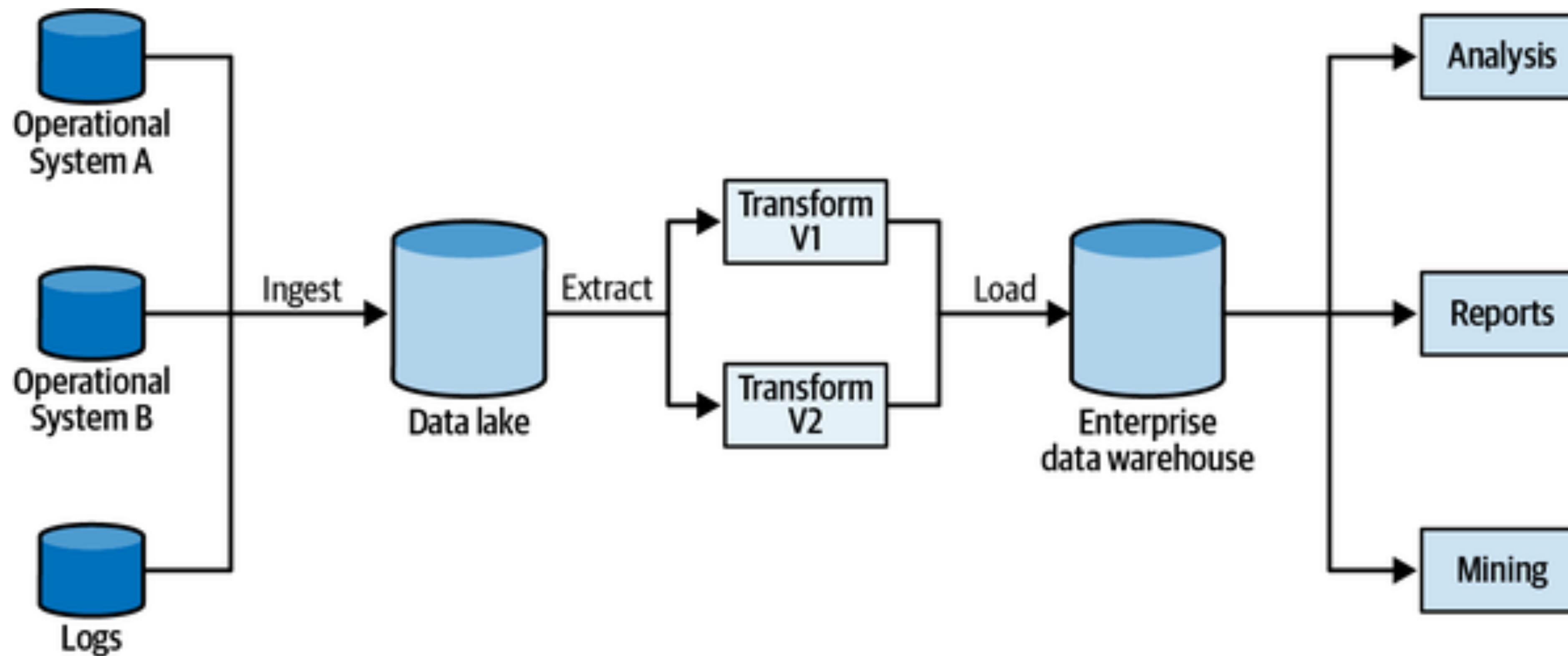


Data Lake



Data Lake

It's not uncommon for data engineers to implement and support multiple versions of the same ETL script to accommodate different versions of the operational model



Small Details

- Supports large volumes of data in diverse formats: CSV, JSON, Parquet, Avro, video, images, logs, etc.
- Typically implemented using object storage (e.g., Amazon S3, Azure Data Lake Storage).
- Integrates with tools like Kafka, Flume, or AWS Glue for ingesting data.
- Supports AI/ML workflows, ad-hoc analysis, and business intelligence tools.
- Implements data governance and security policies (e.g., AWS Lake Formation).

Tradeoffs

- **Advantages:**

- Cost-effective for storing raw, unstructured data.
- Flexible schema-on-read approach.
- Scalable for petabyte-scale storage and analytics.
- Facilitates AI/ML and real-time processing.

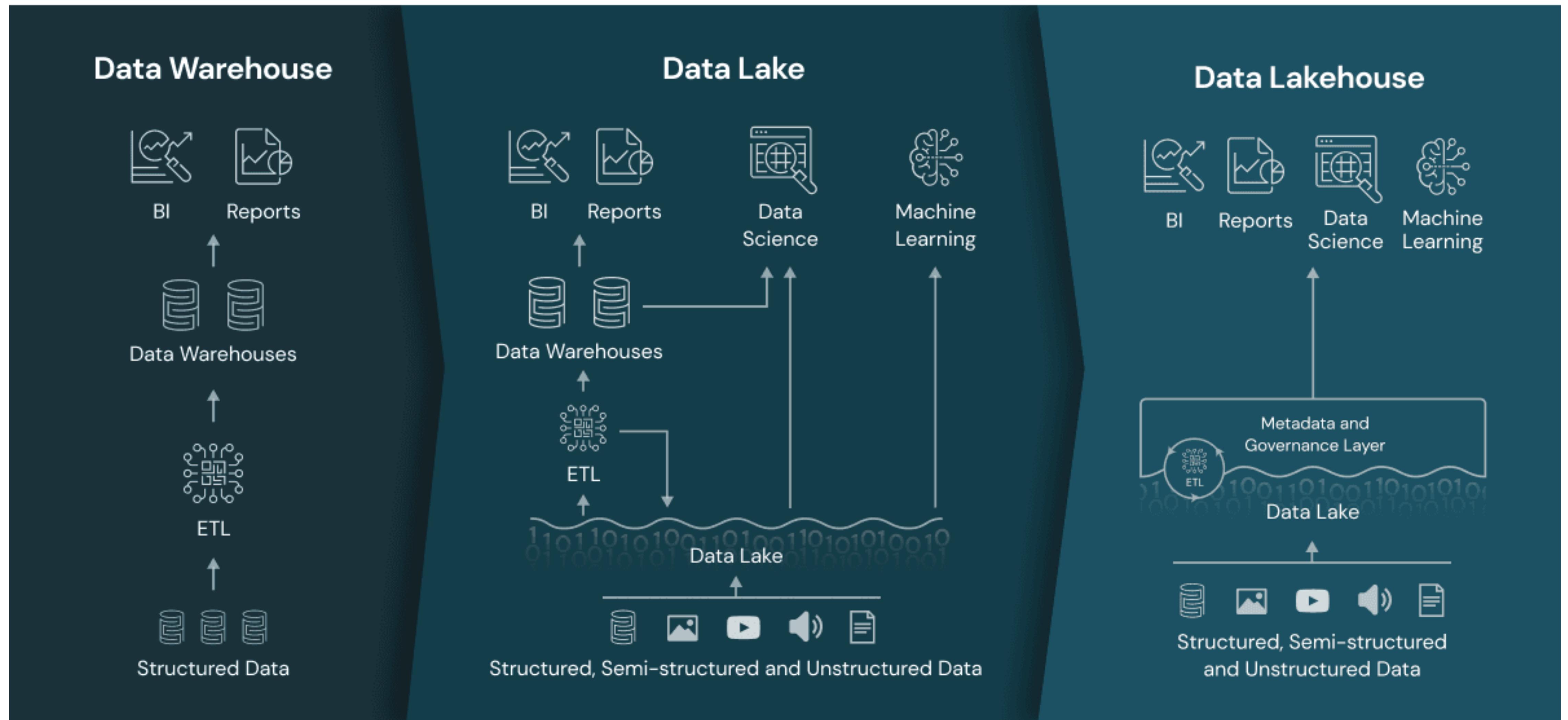
- **Disadvantages:**

- Risk of becoming a “Data Swamp”: Poorly managed data lakes can lead to unorganized, low-quality data.
- Higher complexity for users to derive value (requires robust querying and data governance).
- Latency in querying raw data compared to structured warehouses.
- Security challenges with managing diverse data formats and access policies.

Data Lakehouse

Data Lakehouse

- A Data Lakehouse bridges this gap
- A data lakehouse combines the best of data lakes and data warehouses into a single system.
 - Handles structured, semi-structured, and unstructured data.
 - Provides high-performance analytics and supports AI/ML workloads.
 - Enforces data governance while retaining the scalability and flexibility of data lakes.



Small Details

- **Unified Architecture:** Single system for storage and analytics.
- **Open Storage Format:** Supports Parquet, ORC, or Delta for easy access and compatibility.
- **Real-Time Data Processing:** Enables faster insights with streaming capabilities.
- **Integrated Governance:** Ensures quality, lineage, and compliance across all data.

Open Table Format



Open Table Format

- Category of technologies for how big data tables are stored and managed
- Decouples compute from storage
- Enables ACID guarantees on Data Lakes
- Supports Schema Evolution and Partition Evolution
- Provides Time Travel and Rollback with metadata snapshots

<https://www.teradata.com/insights/data-platform/what-are-open-table-formats>

Other Competitors

- **Apache Iceberg:** Excels in schema and partition evolution, efficient metadata management, and broad compatibility with various data processing engines.
- **Apache Hudi:** Best suited for real-time data ingestion and upserts, with strong change data capture capabilities and data versioning.
- **Apache Paimon:** A lake format that enables building a real-time lakehouse architecture with Flink and Spark for both streaming and batch operations.
- **Delta Lake:** Provides robust ACID transactions, schema enforcement, and time travel features, making it ideal for maintaining data quality and integrity.

Apache Iceberg – The Open Table Format for Lakehouse AND Data Streaming

Who runs the Open Table Format?

- Since it is technology category, Apache Iceberg, Delta Lake, and Apache Hudi, are all maintained by their respective open-source communities.
 - Iceberg is managed by the Apache Software Foundation.
 - Delta Lake is developed by Databricks
 - Hudi is maintained by the Apache Software Foundation

<https://aws.amazon.com/blogs/big-data/choosing-an-open-table-format-for-your-transactional-data-lake-on-aws/>

<https://delta.io/blog/open-table-formats/>

Schema Formats



Avro

Avro

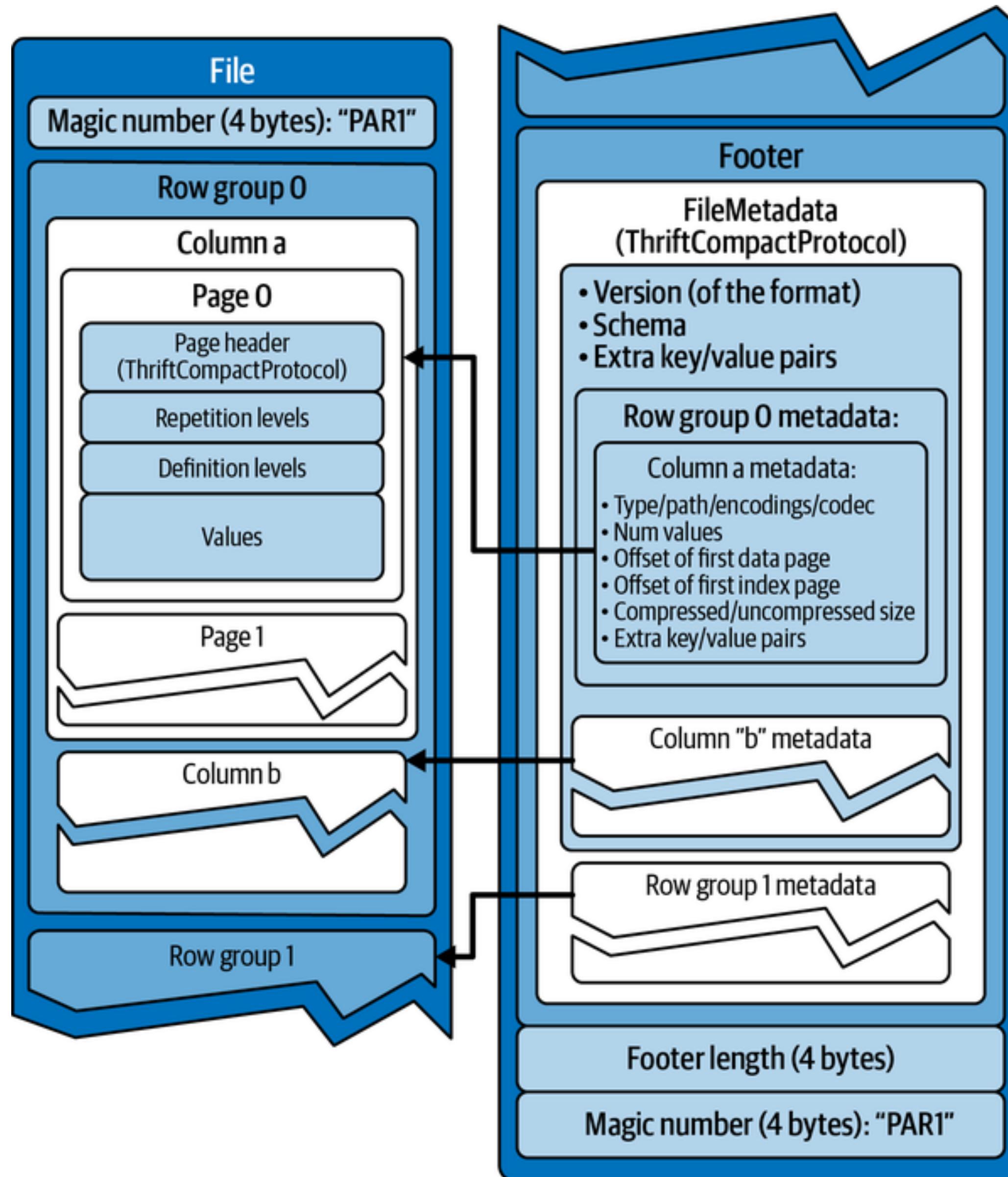
- Created by Doug Cutting; Creator of Hadoop
- Serialization is defined by schema
- Schemas are JSON Based
- Codegen available at command line
- Codegen available by Maven, Gradle, SBT Plugin
- Supports the following languages:
- C, C++, Java, Perl, Python, Ruby, PHP
- Named after WWI, WWII Aircraft Company, A.V. Roe



Parquet

Parquet

- Introduced by Twitter and Cloudera in 2013
- Designed for efficient columnar storage on Hadoop ecosystem
- Row-based formats (like CSV, JSON) had poor performance for analytics
- Column-based formats enables easy selective read of columns during queries
- Built for analytics and OLAP workloads



Internal Structure

- **File → Row Groups → Column Chunks → Pages**
- **Row Groups:** Horizontal partitioning of data (like batches of rows)
- **Column Chunks:** Data for one column within a row group
- **Pages:**
 - Data Page – actual values
 - Dictionary Page – optional
 - Index Page – optional (used in advanced filters)

OLTP vs OLAP



OLTP vs OLAP

- Online analytical processing (OLAP) and online transaction processing (OLTP) are data processing systems that help you store and analyze business data.
- OLAP combines and groups the data so you can analyze it from different points of view
- OLTP stores and updates transactional data reliably and efficiently in high volumes
- OLTP databases can be one among several data sources for an OLAP system.

[What is the difference between OLTP and OLAP](#)

OLTP vs OLAP

OLAP vs OLTP

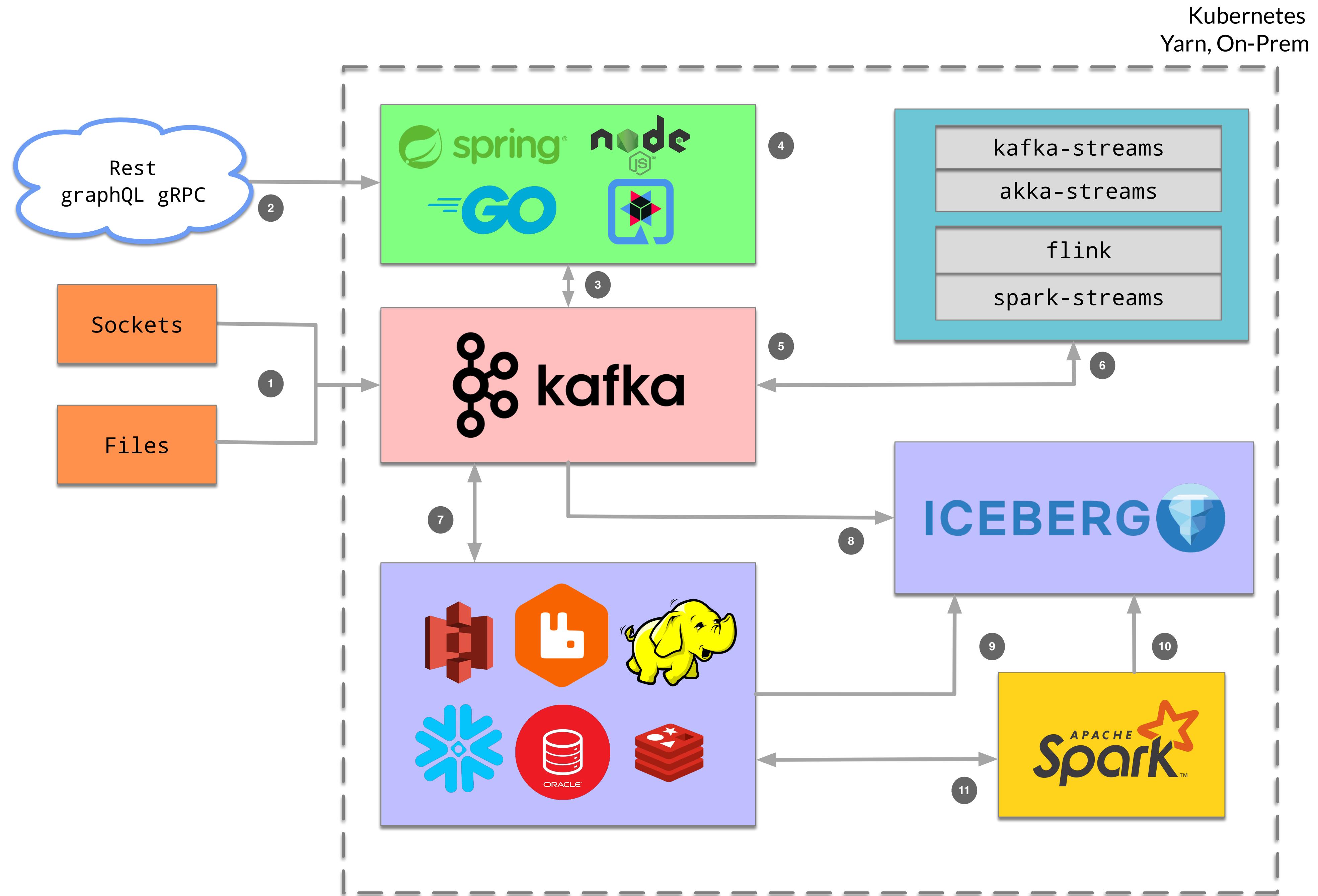
Aspect	OLAP (Analytical)	OLTP (Transactional)
Purpose	Analytics, reporting, decision-making	Real-time transaction processing
Query Complexity	Complex, multi-dimensional queries	Simple, fast queries
Data Type	Historical, aggregated, and derived data	Current, detailed, and transactional data
Operations	Read-intensive	Read and write-heavy
Schema Design	Star or snowflake schema	Normalized schema (3NF or higher)
Performance Goal	Optimized for query speed	Optimized for transaction speed
Concurrency	Few concurrent users	Many concurrent users
Example Use Cases	BI dashboards, trend analysis, forecasting	Order processing, inventory tracking
Examples	Snowflake, Google BigQuery, Tableau	MySQL, PostgreSQL, Oracle DB

Apache Iceberg

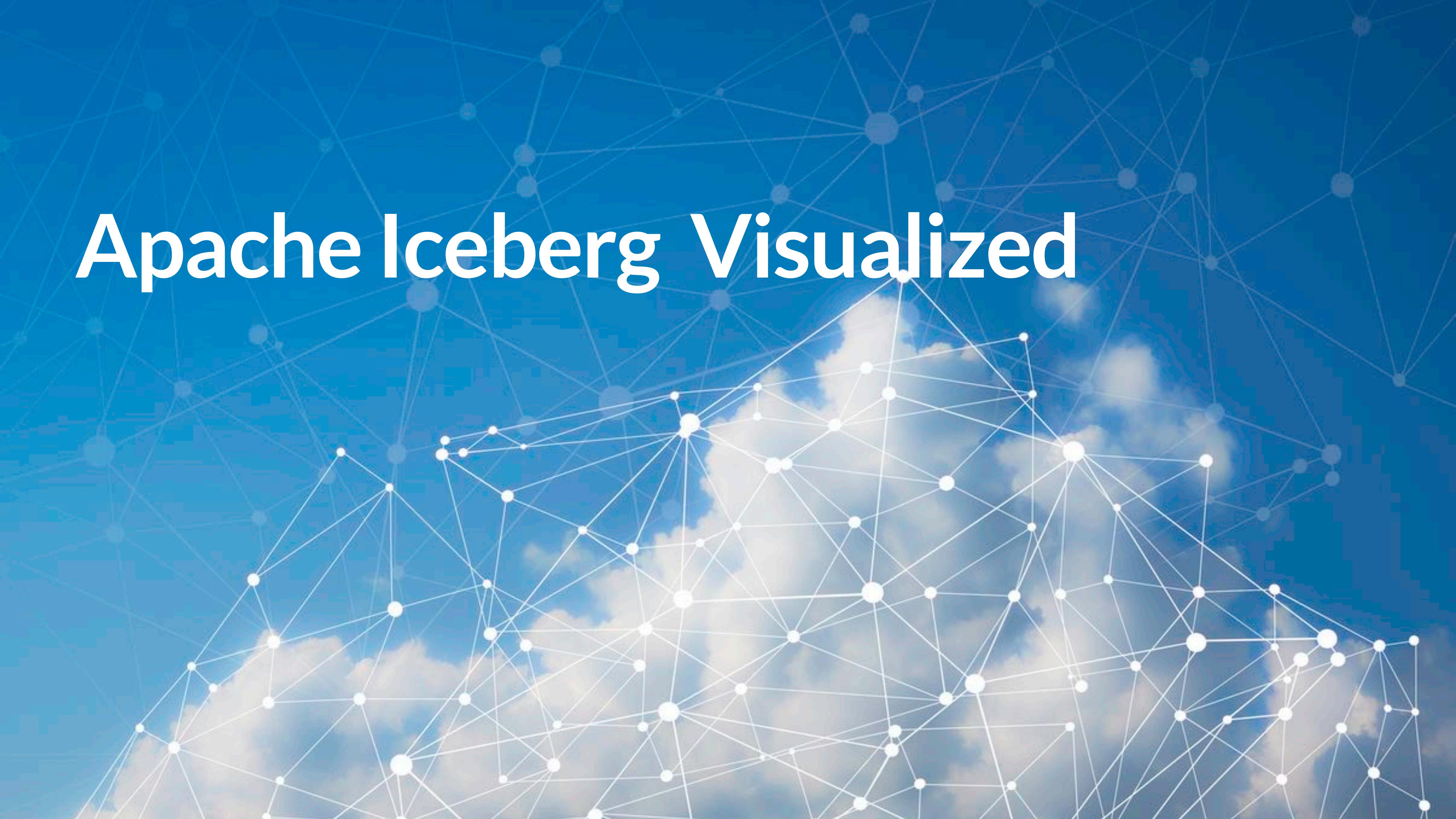




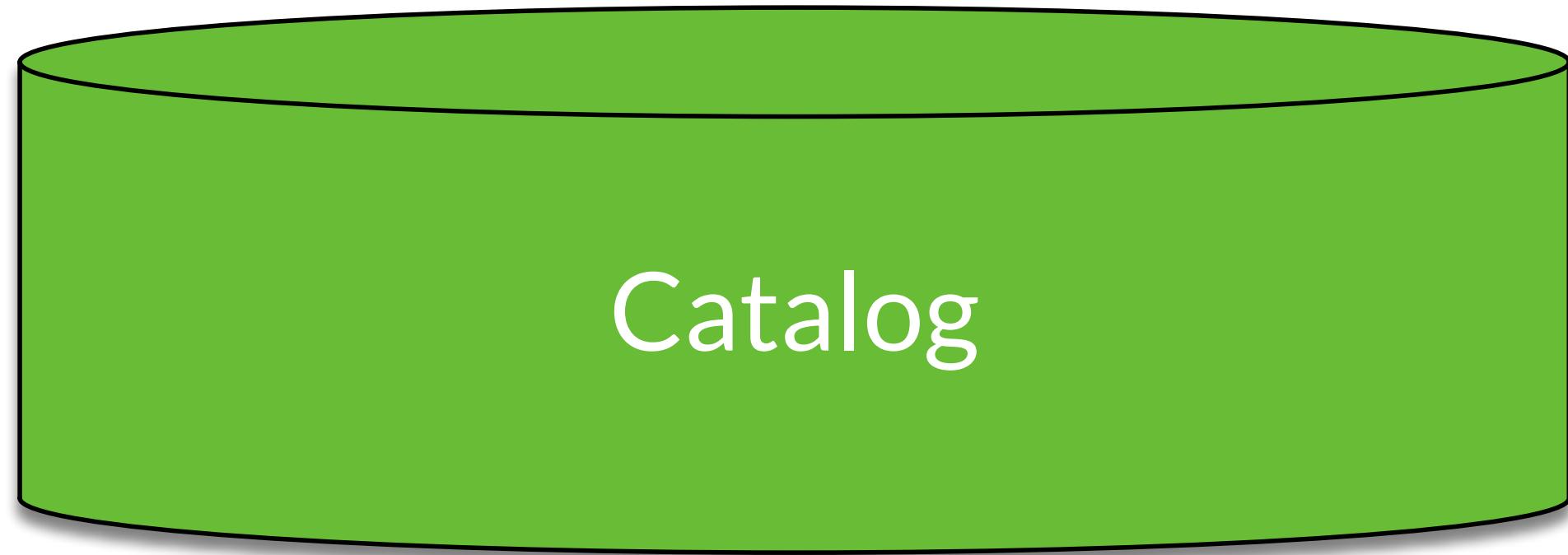
- Iceberg is an open table format.
- Defines how data is stored
 - How partitions are managed
 - How a schema evolves
 - How time travel works
- Iceberg organizes files like Parquet, Avro. Typically in Parquet due to it's columnar structure
- It is engine agnostic - Iceberg lets multiple engines read from and write from the *same table* safely
 - Spark, Flink, Trino, Dremio, etc
Data is stored in object storage (S3, GCS, etc)
 - ACID Compliant
 - OLAP - Meant for Large Scale Analytical Queries with Fast Scans and evolving Data Sets



Apache Iceberg Visualized

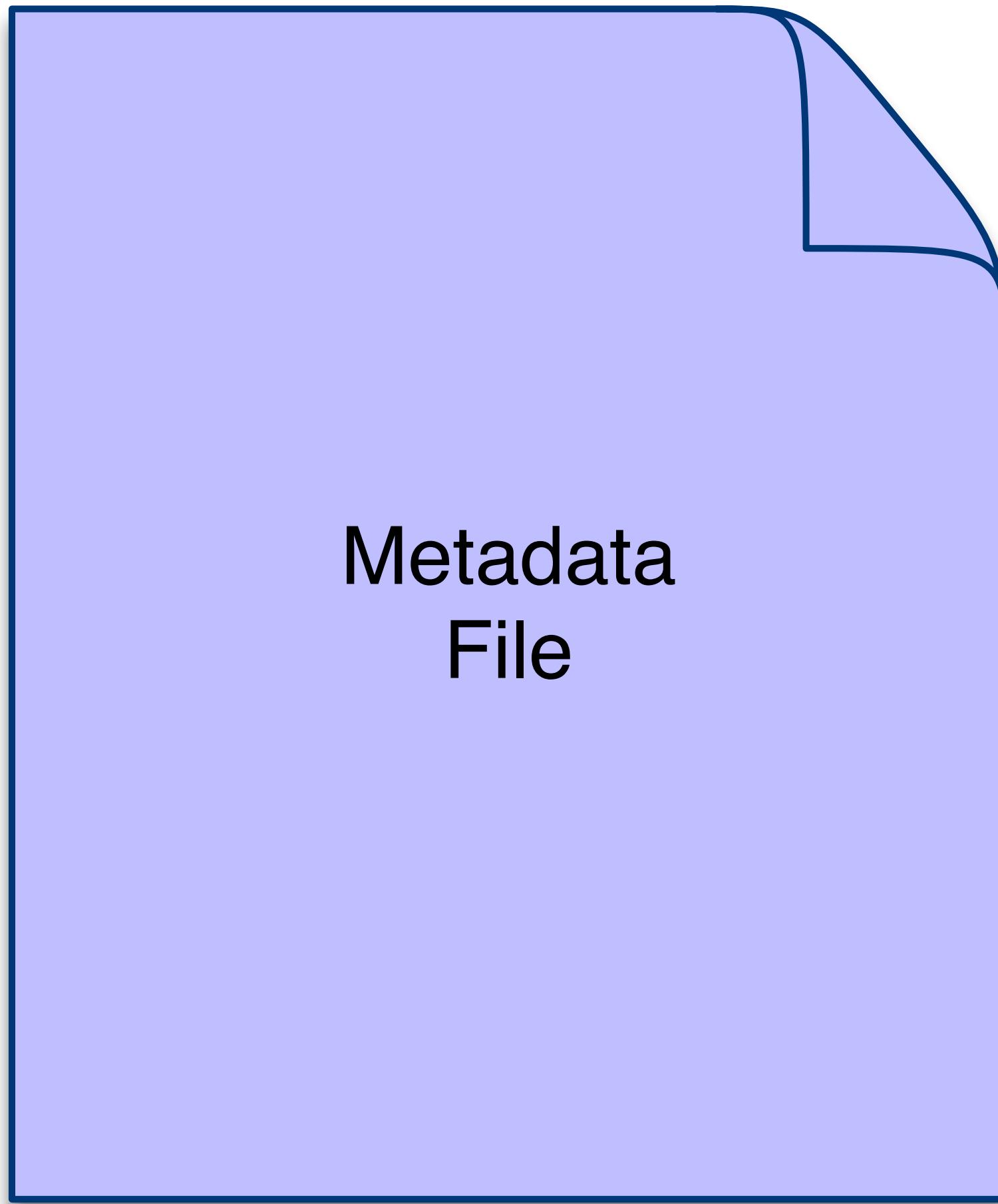


Catalog



- Manages the metadata for Iceberg tables and namespaces
- It tracks all iceberg tables and their metadata locations like `analytics.orders`
- **Multiple Implementations:** Hive, Hadoop, Rest, Nessie
- Coordinates atomic table operations using snapshot
- Entry point for data engines like Flink, Spark, Trino, etc.

Metadata File



- Current State and Structure of the Iceberg Table
- Stored as JSON
- Contains References to All Snapshot which of consistent views of the table
- Store current schemas and table schemas
- **Points to the Manifest Lists**

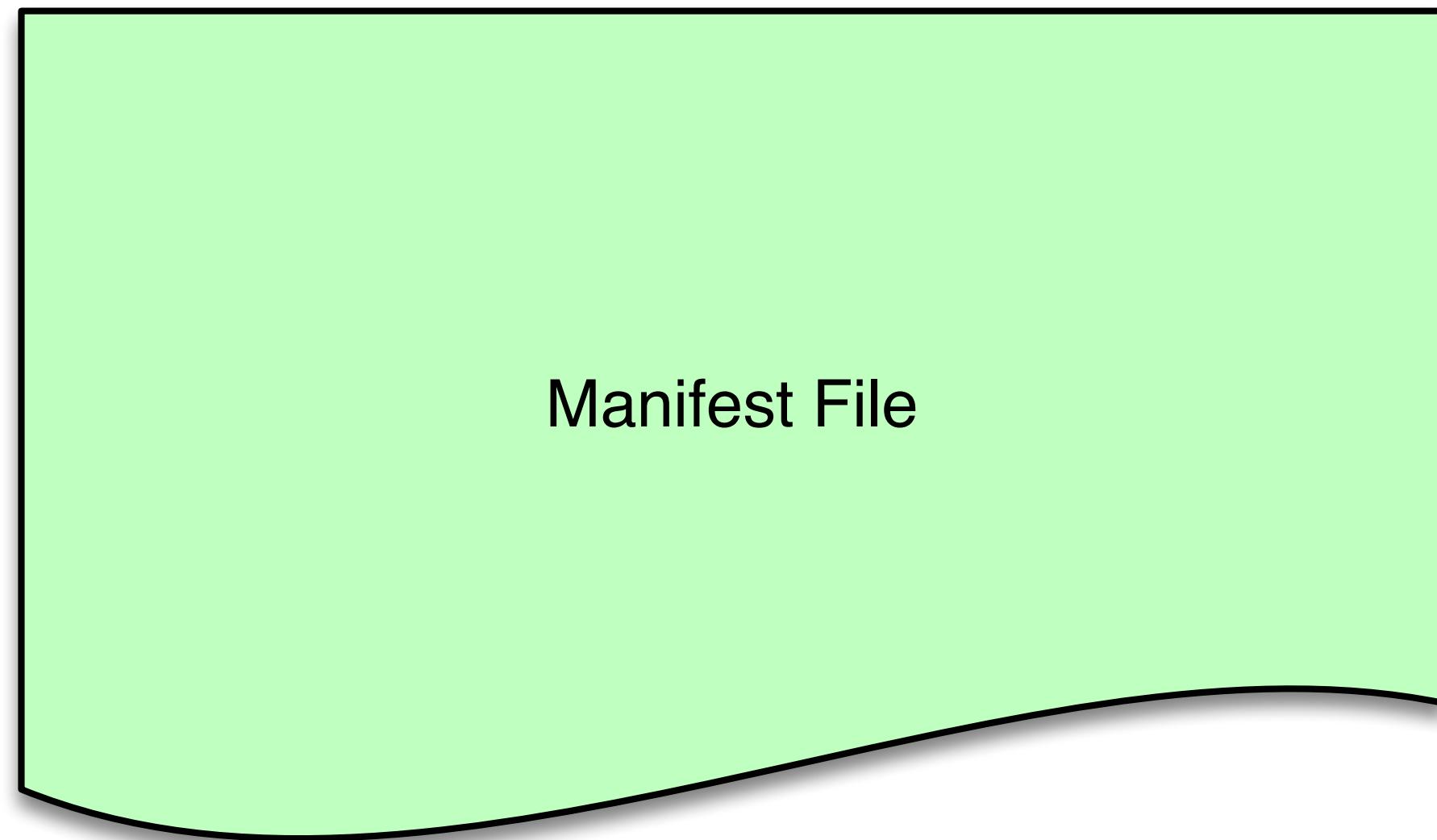
Manifest List



Manifest List

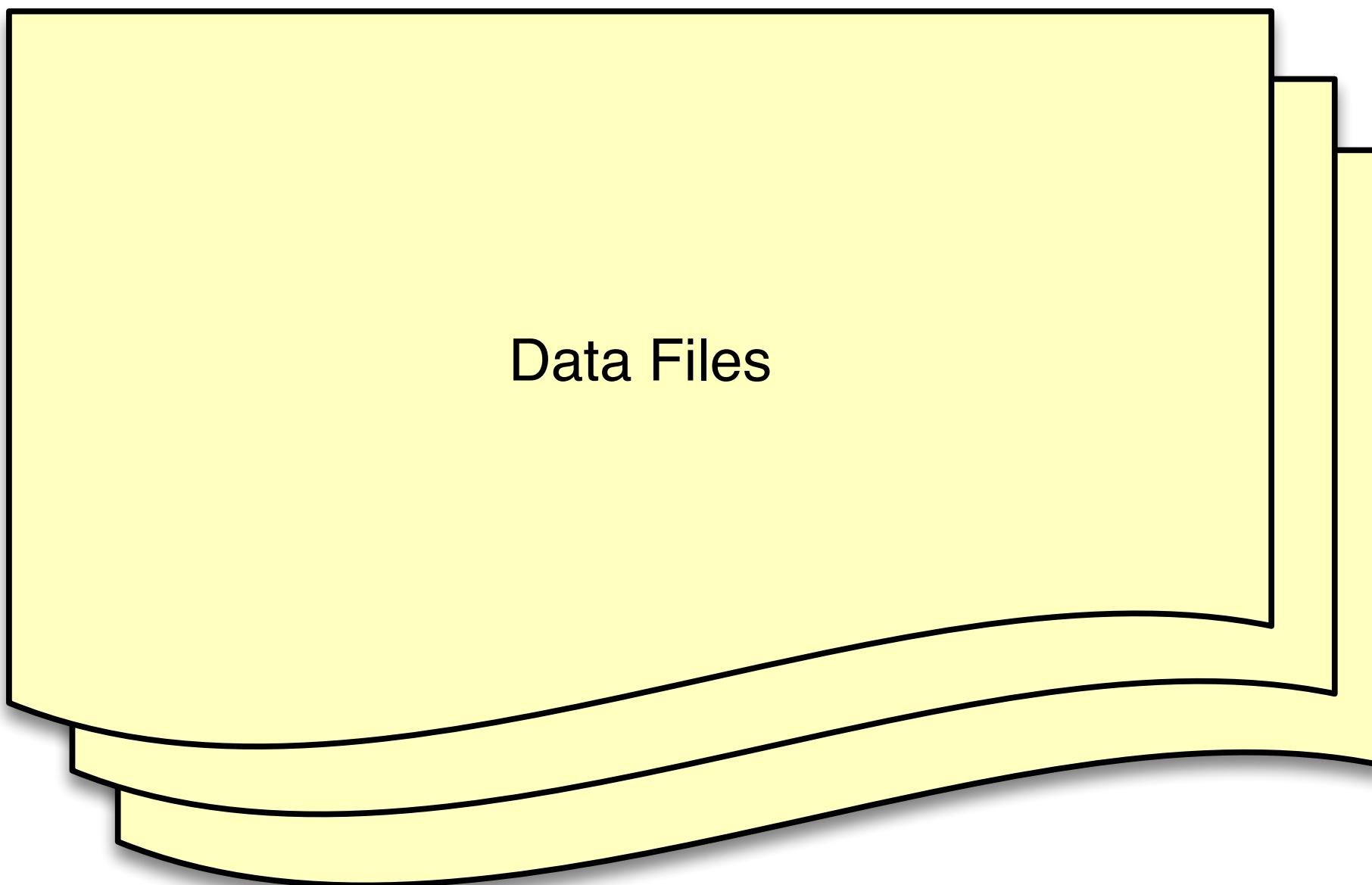
- A manifest list is a metadata file that **lists all the manifest files for a single snapshot.**
- Stored in Avro
- Contains metadata like file counts, partition bounds, and stats used for query planning and pruning.
- Immutable

Manifest File

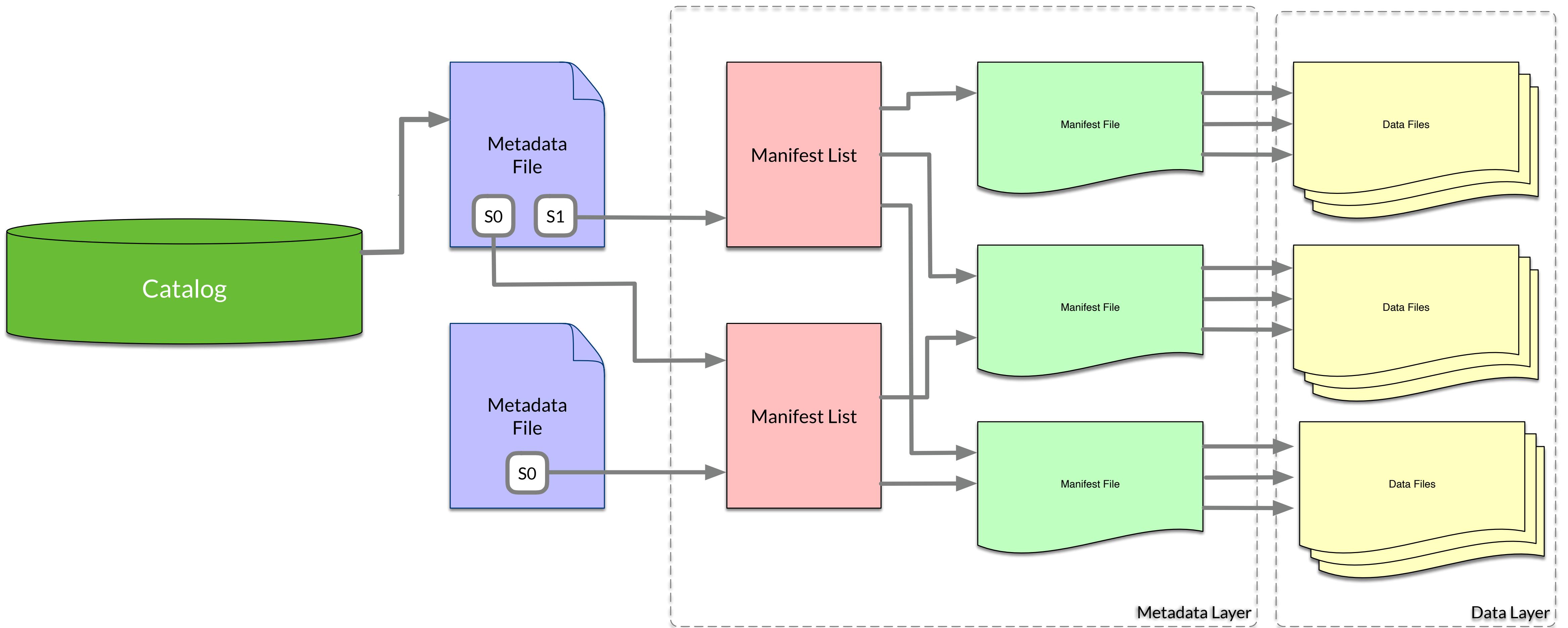


- Each manifest file contains a list of data files (and delete files) that belong to a partition or snapshot
- Grouped by Partition: Manifest files are typically organized by partition
- Stores stats like row counts, column bounds (min/max), null counts, and file size
- Stored as Avro
- Immutable and Append Only

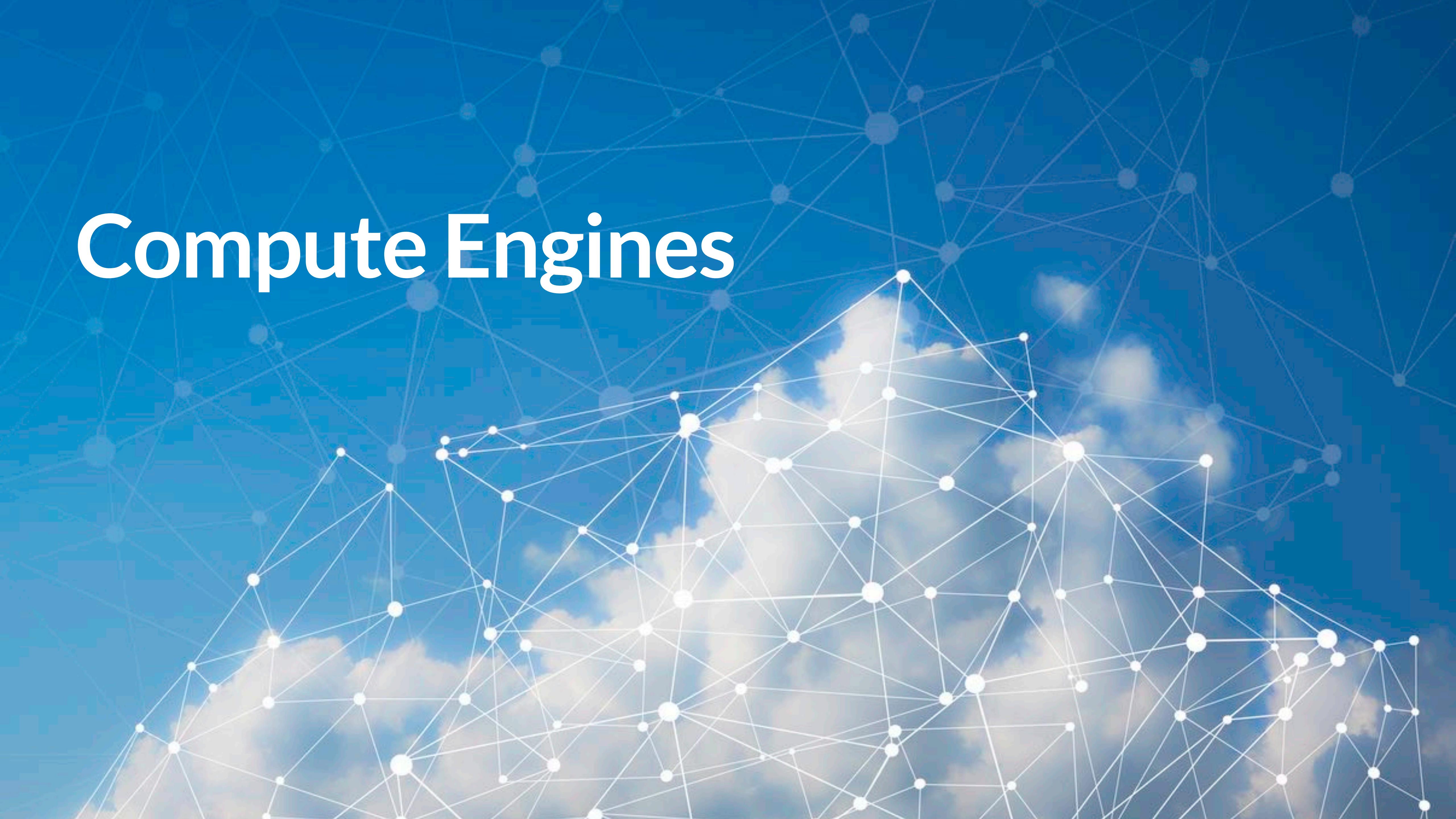
Data Files



- Storage of actual table data
- Columnar File Format like Parquet
- Immutable

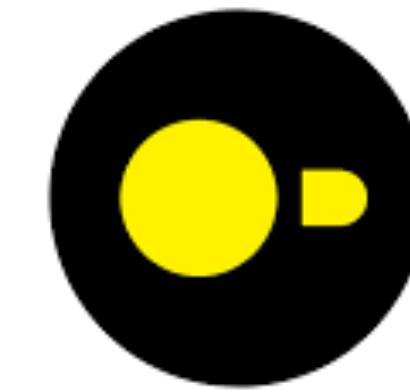


Compute Engines





APACHE
Spark™



DuckDB



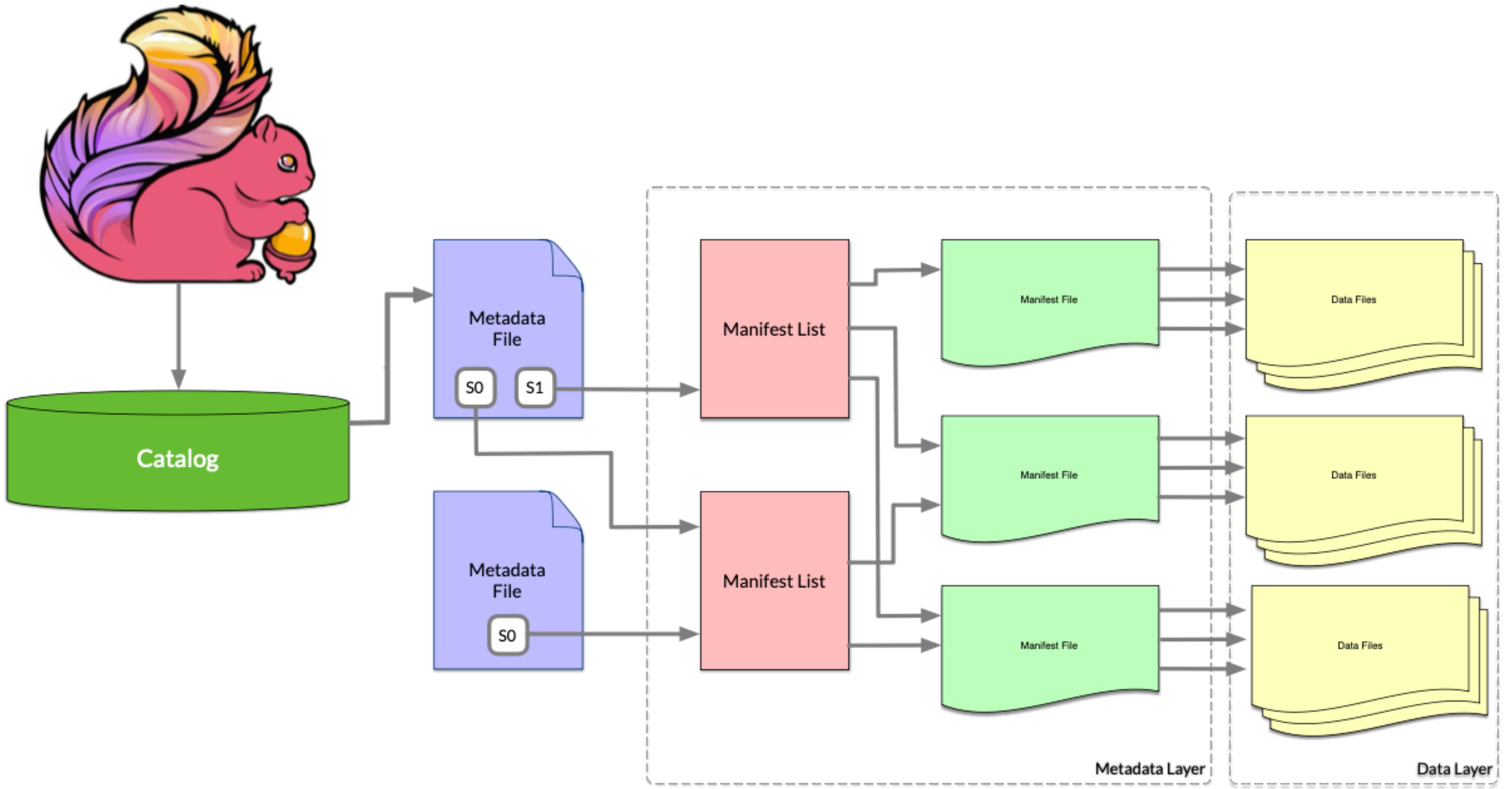
trino



presto



dremio



Demo: Apache Iceberg



- Let's see how we can run Apache Iceberg and all the components involved
- We will then insert some data, and view the snapshot
- We can then view the snapshots

Time Travel



Time Travel in Iceberg

- We can query data as it existed in a previous point in time
- Because we have *immutable* snapshots we can roll back in time
- Use AS OF VERSION (snapshot ID) or AS OF TIMESTAMP in SQL

Normal Query



Snapshot 0
2025-06-12

Snapshot 1
2025-06-13

Snapshot 2
2025-06-17

Manifest List

Manifest List

Manifest List

Time Travel Query



Snapshot 0
2025-06-12

Snapshot 1
2025-06-13

Snapshot 2
2025-06-17

Manifest List

Manifest List

Manifest List

Can it be made permanent?

- You can make the call: `CALL system.rollback_to_snapshot('my_table', 1234567890);` to move to the latest snapshot (for Spark only)
- This will make the snapshot the new current version
- It does not delete newer data or snapshots

What if I really want to “Hard Reset”?

- Rollbacks are metadata-level operations.
- Previous and newer snapshots still exist—unless explicitly expired.
- You can still time travel to newer (post-rollback) snapshots unless:
- You call `expire_snapshots` to remove them permanently.
- Retention policies prune them automatically
- For Permanent Rollback use: `CALL system.expire_snapshots('my_table', retain_last = 1);`

Demo: Time Travel



- Let's rewind to either the snapshot or the date and see if we see a different perspective

Hidden Partitioning



Hidden Partitioning

- Traditional table formats require explicit partitioning columns
- Iceberg introduces hidden partitioning to simplify queries and avoid user errors
- No need to include partition columns in queries (e.g., no WHERE year = 2024)
- Partition values are derived using transforms on columns

Hidden Partitioning Reasoning

- Prevents data skipping mistakes when users forget to filter on partition columns
- Enables automatic filtering during query planning
- Makes schema evolution and partition evolution safer and easier
- Encourages query simplicity: filters are applied without knowing internal partitioning

Hidden Partitioning Examples

- `identity(col)` - default (no transformation)
- `bucket(col, 16)` - evenly distributes data into 16 buckets
- `truncate(col, 4)` - truncates string/numeric value
- `year(ts) / month(ts) / day(ts)` - extract date parts
- Enables fine-grained partitioning without over-exposing schema details

Schema Evolution



Schema Evolution

- Iceberg allows safe, in-place schema evolution
- Supports adds, drops, renames, reorders, and type promotions
- **No need to rewrite existing data files**
- Schema changes are versioned in *metadata snapshots*
- Old queries on historical snapshots will use the schema at the time

What can I do with a schema change?

- Add, Drop, Rename, Reorder Columns
- Promote Types; e.g from int → long, float → double

What can I not do with a schema change?

- Demote Types; e.g from long → int
- Change Column Id

Backward and Forward Compatibility

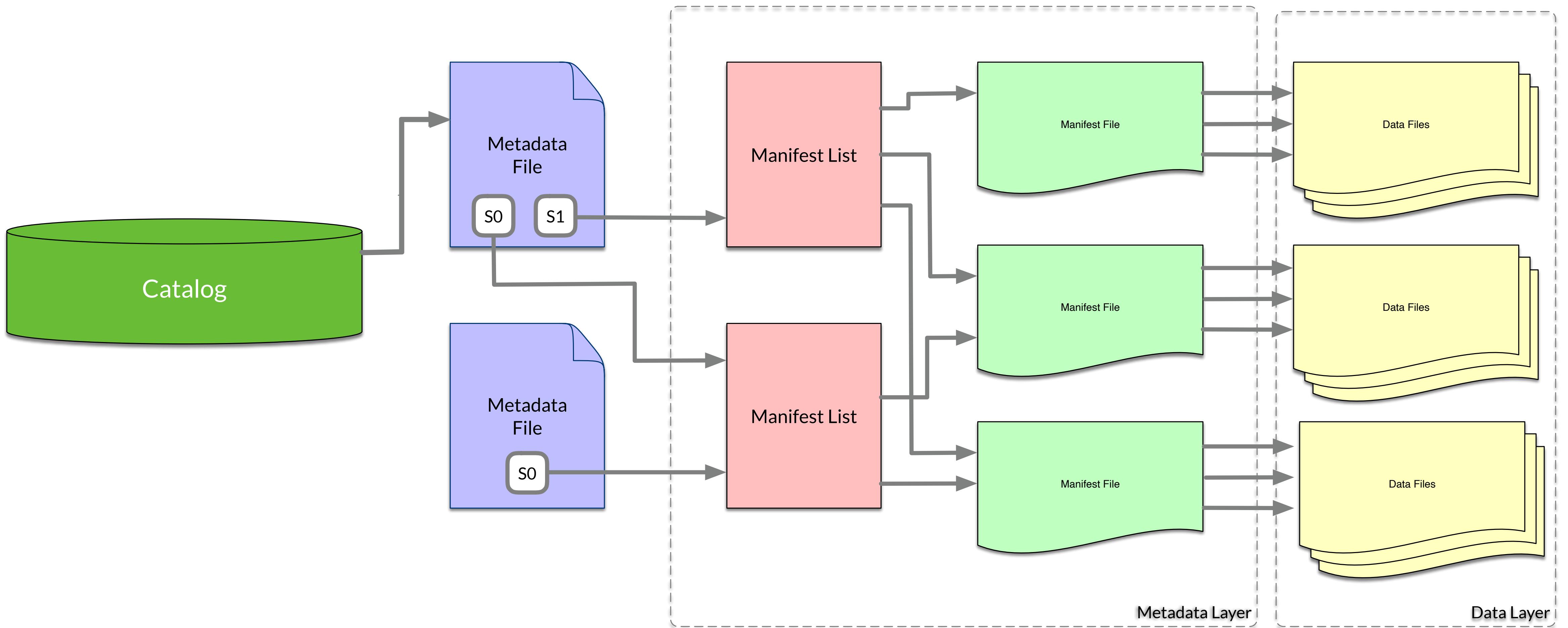
- Iceberg tracks column IDs, not just names – enables safe renames
- New reads must handle old data (backward compatible)
- Old readers can skip new columns (forward compatible)
- Schema must be evolution-safe: don't drop critical fields required by old readers

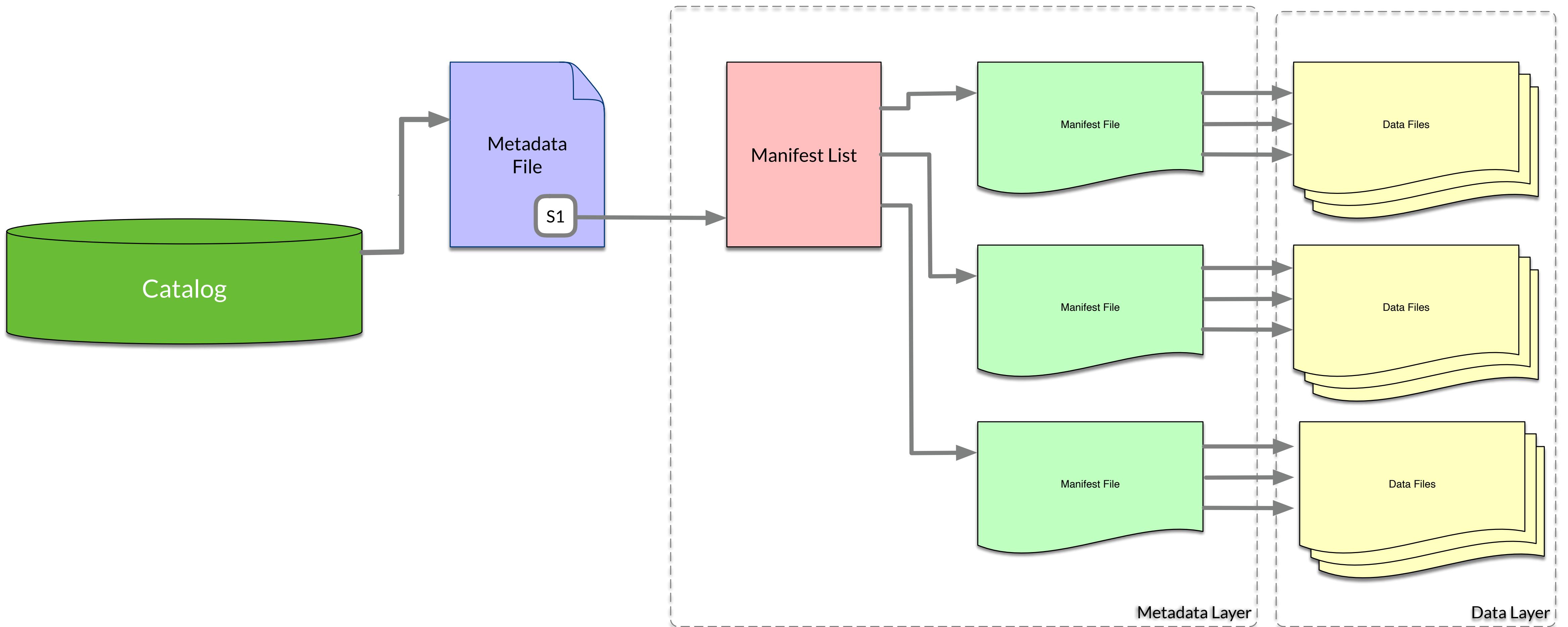
Pruning



Pruning and Expiration Policies

- Snapshots track the state of a table over time (version history).
- Use `CALL system.expire_snapshots(. . .)` to manually expire old snapshots.
- Expiration removes **snapshot metadata** and **orphaned manifest and metadata files**
- You can express the criteria by which to prune using `retain_last` or `older_than`
- Helps storage footprint and performance
- Can be scheduled and supports incremental cleanup





Query Pruning



Query Pruning

- Queries scan only relevant files using partition & metadata information – no full table scans.
- Filters like WHERE vendor_id = 4 eliminate all files not matching that partition.
- Manifests track min/max values and partition info – entire manifests can be skipped.
- Iceberg stores column-level stats (min, max, null count) per file – files outside filter bounds are excluded.
- Reduces I/O and increases performance

Delete Files



Delete Files

- Important to remember that data is immutable
- Deletes are tracked using separate files: delete files, which maintain row-level or file-level deletion information.
- This allows efficient updates and deletions with minimal I/O cost.

Types of Delete Files

- Equality Deletes:
 - Contain rows that match certain column values (e.g., `id = 42`).
 - Typically used in row-level deletes.
- Position Deletes:
 - Specify the exact row position in a specific data file to delete.
 - Useful when deleting known rows (like from compaction or merge-on-read engines).

How Deletion Works

1. User Issues a DELETE: `DELETE FROM customers WHERE id = 42;`

2. Nothing is actually deleted; a delete file is created in its place

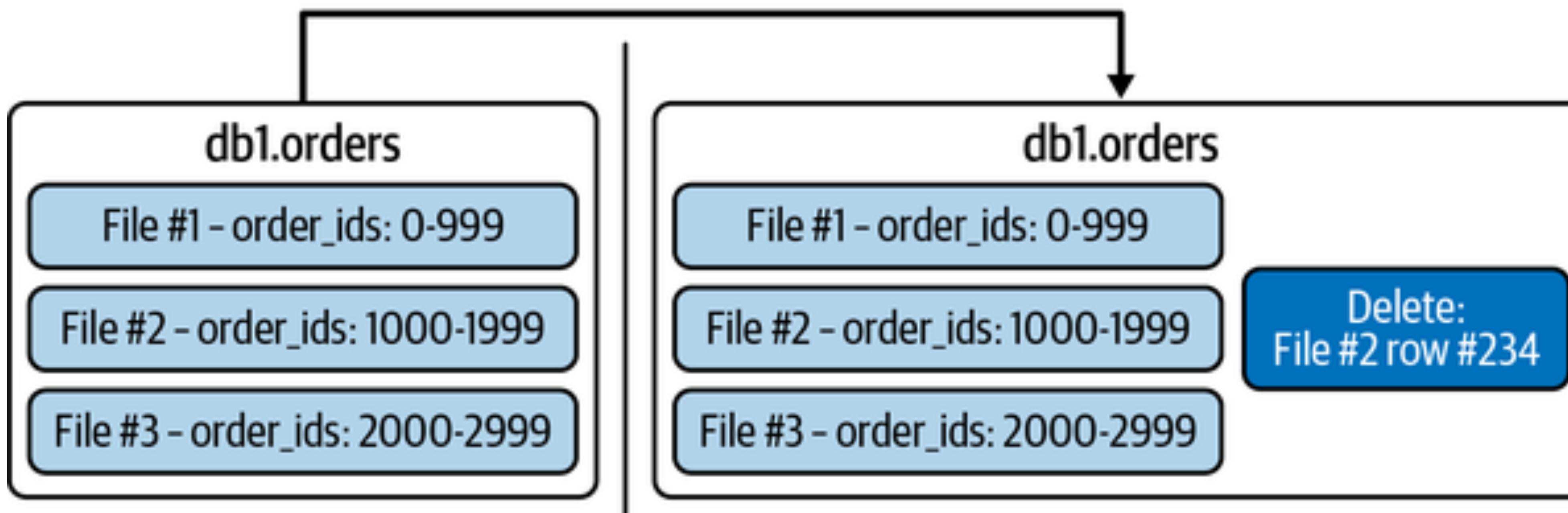
3. New Snapshot Is Created:

- The snapshot includes:
- The original data files
- The new delete file

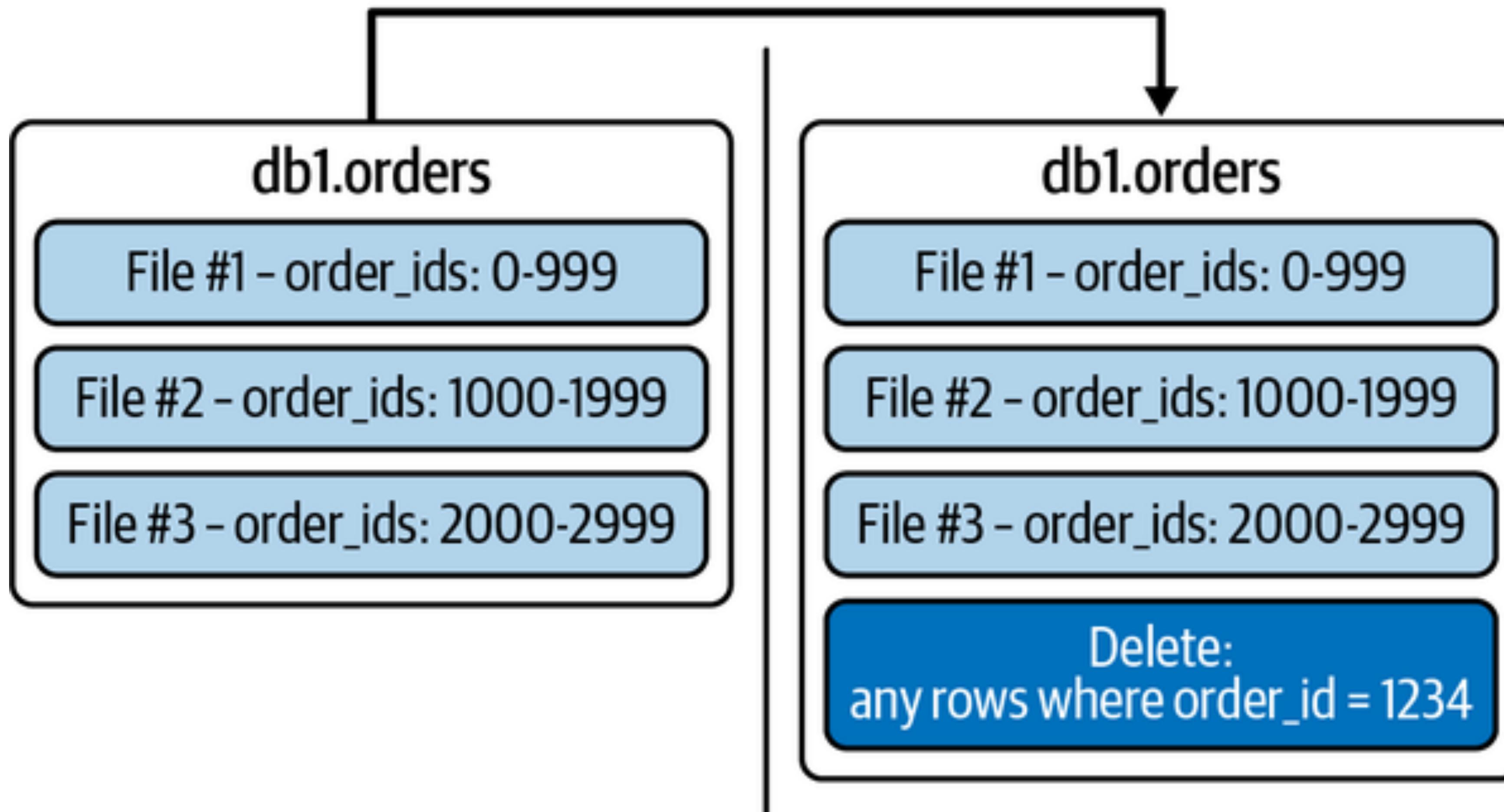
4. At Read Time:

- Delete files are applied on-the-fly.
- Engines skip or filter out rows marked for deletion.

```
DELETE FROM orders  
WHERE order_id = 1234
```



```
DELETE FROM orders  
WHERE order_id = 1234
```



Copy On Write vs Merge On Read

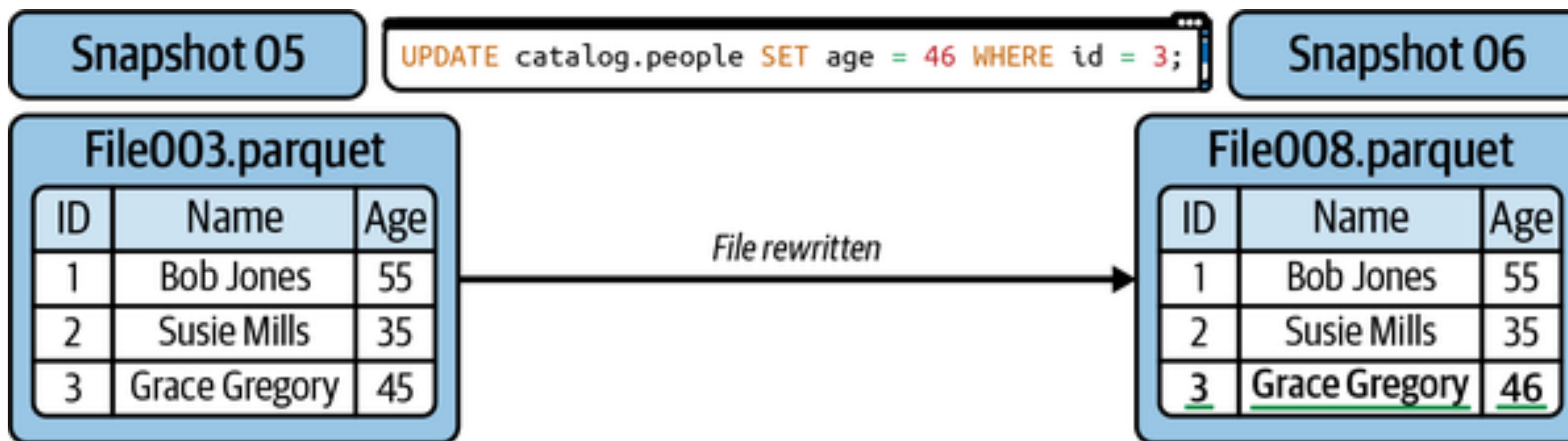


Copy-on-Write Versus Merge-on-Read

- When you are adding new data, it just gets added to a new datafile, but when you want to update preexisting rows to either update or delete them, there are some considerations.
- Recall everything is immutable, so if you're updating 10 rows, there is no guarantee they are in the same file, so you may have to rewrite 10 files and every row of data in them to update 10 rows for the new snapshot.

Copy on Write

- The default approach is referred to as copy-on-write (COW). In this approach, if even a single row in a datafile is updated or deleted, that datafile is rewritten, and the new file takes its place in the new snapshot.



Merge On Read

- The alternative to copy-on-write is merge-on-read (MOR), where instead of rewriting an entire datafile, you capture in a delete file the records to be updated in the existing file, with the delete file tracking which records should be ignored.
- **If you are deleting a record:**
 - The record is listed in a delete file.
 - When a reader reads the table, it will reconcile the datafile with the delete file.
- **If you are updating a record:**
 - The record to be updated is tracked in a delete file.
 - A new datafile is created with only the updated record.
 - When a reader reads the table, it will ignore the old version of the record because of the delete file and use the new version in the new datafile.

Configuring COW or MOR

- The following table properties determine whether a particular transaction is handled via COW or MOR:
 - `write.delete.mode` - Approach to use for delete transactions
 - `write.update.mode` - Approach to use for update transactions
 - `write.merge.mode` - Approach to use for merge transactions

Compaction



Compaction

- Process of rewriting small or fragmented files into larger, optimized files.
- Helps reduce metadata overhead and read amplification.
- Not required immediately—lazy and configurable.
- Especially useful after many small writes or deletes.

Many smaller files

File 1
10 records

Open, read, close 1x

File 2
10 records

Open, read, close 2x

File 3
10 records

Open, read, close 3x

File 4
10 records

Open, read, close 4x

File 5
10 records

Open, read, close 5x

Fewer bigger files

File 1
50 records

Open, read, close 1x

Why is Compaction Required?

- Improves query performance by reducing the number of files to scan.
- Enhances read efficiency (fewer I/O operations).
- Reduces planning and metadata load (fewer manifest entries).
- Cleans up old delete files and snapshot metadata.

Types of Compaction

- File Compaction (Data File Compaction):
 - Merges small Parquet/Avro files into larger ones.
- Rewrite Deletes:
 - Rewrites data files by applying delete files.
 - Produces clean, delete-free files.
- Manifest Rewrite:
 - Rewrites the manifest files to compact metadata structure.

When Does it Happen?

- Manually triggered using Flink, Spark, or Iceberg APIs.
- Can be scheduled as part of maintenance jobs.
- Often done after ingest jobs, particularly for streaming writes.

ACID Transactions



ACID Transactions

- Iceberg uses snapshot isolation, not locking, to handle ACID guarantees
- Atomic writes: pipelined file additions (data, manifests, metadata updates) occur all-or-nothing, avoiding partial commits
- Batch ingest safety: if an ingestion job fails mid-stream, no partial state is visible
- Schema evolution is transactional: applying schema changes creates a new snapshot; failure means no schema change is recorded
- Row-level updates/deletes are supported atomically, with conflict detection if simultaneous writers update/delete is atomic for target rows

Integrating for ML



Integration for ML

- Use Trino, Spark, or Flink SQL clients (via JDBC or Python APIs) to query Iceberg tables from Python notebooks.
- Fetch the most recent snapshot or partitioned slices of data for training ML models using Pandas, Scikit-learn, or TensorFlow.
- Use Iceberg's snapshot-based time travel to re-train or validate models using consistent historical data.
- Add or modify input features in Iceberg tables as model requirements change—without breaking existing pipelines.

Integrating for DataMesh

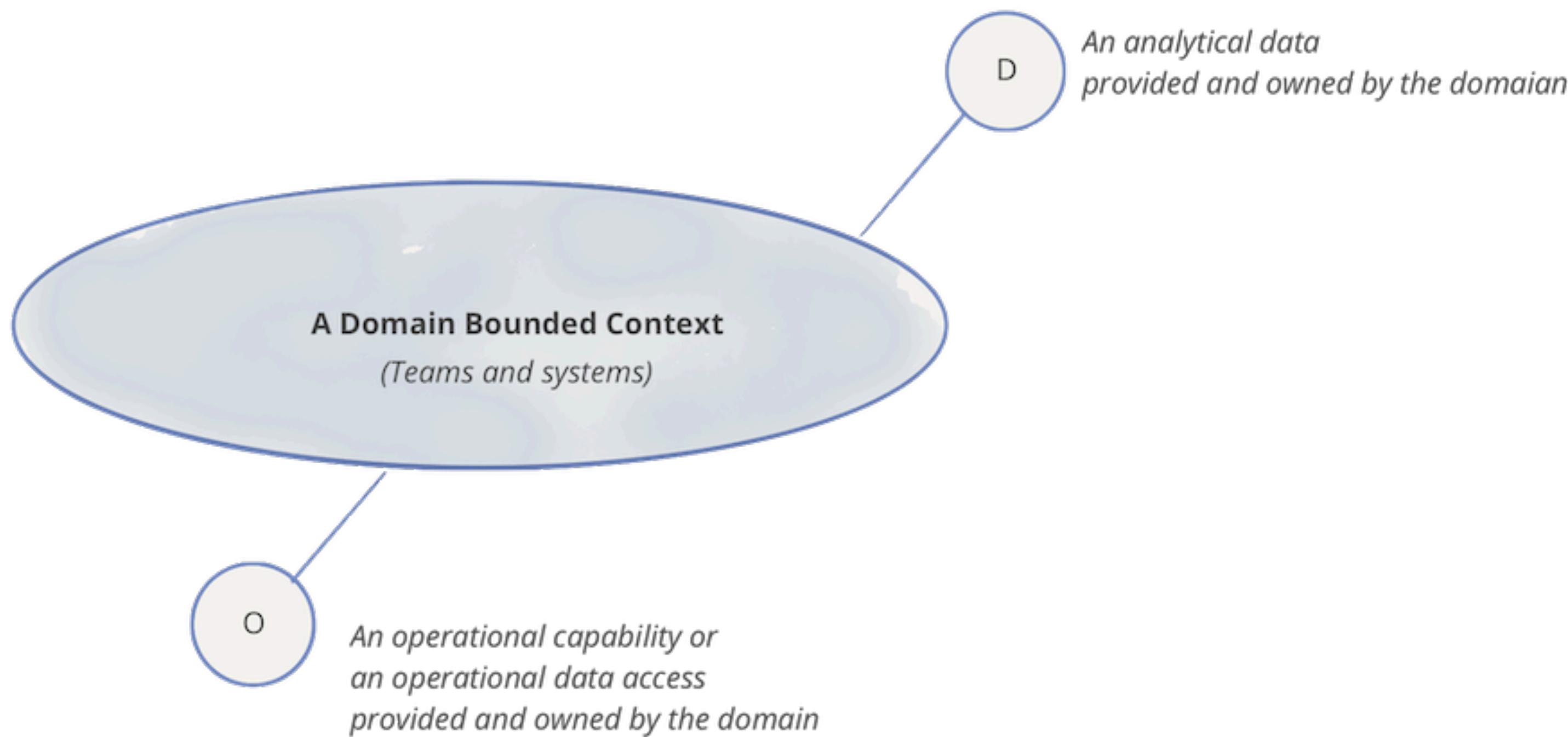


Data Mesh

- There are two patterns: Database per Service, and Shared Database
- Data Mesh is an attempt at bringing not only the database but owning all data
- Data Mesh is about providing data ownership per context, and follows these principles:
- Domain-oriented decentralized data ownership and architecture
- Data as a product
- Self-serve data infrastructure as a platform
- Federated computational governance
- The accountability of data quality shifts upstream as close to the source of the data as possible.

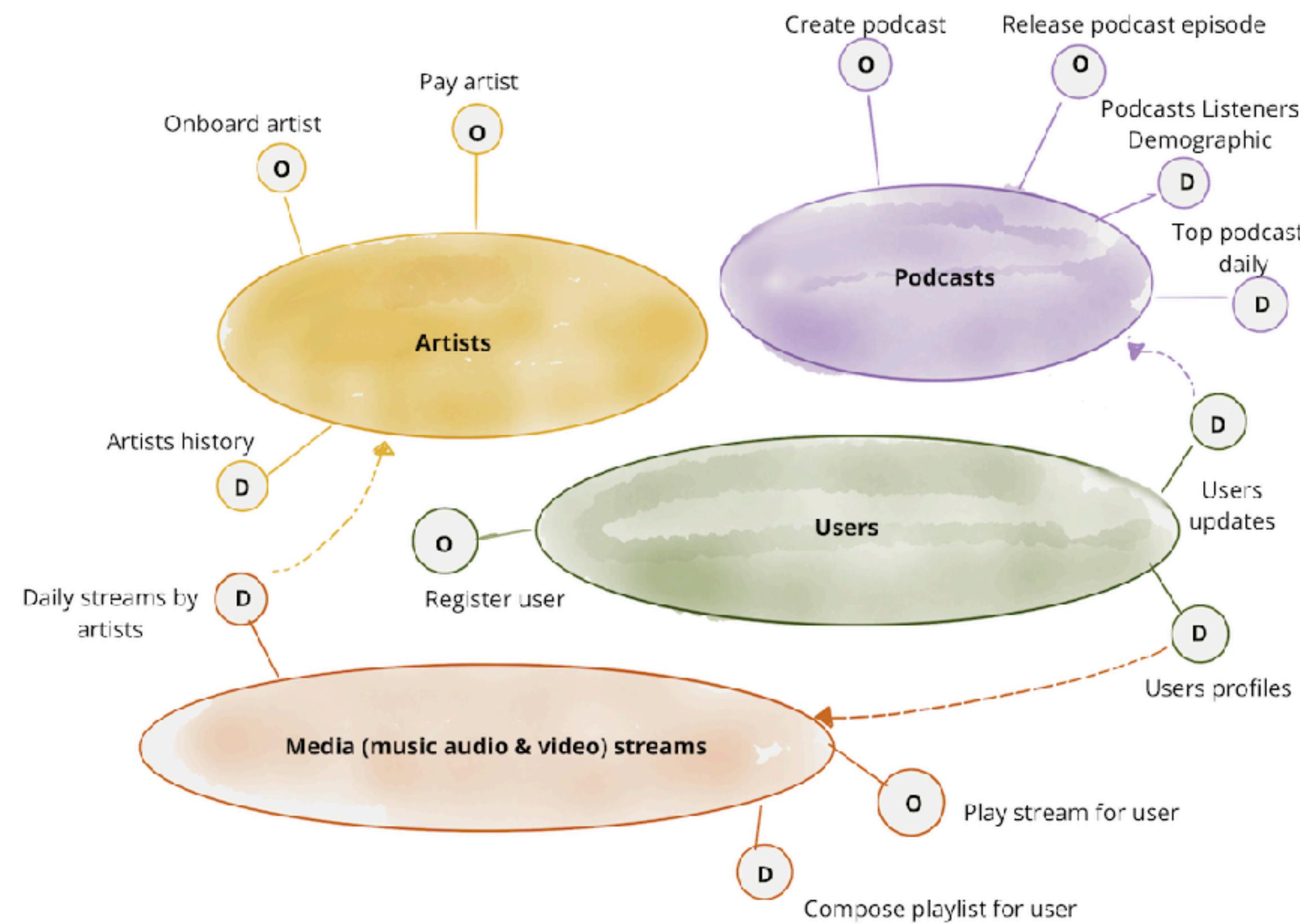
The corps was composed of all arms of the service, was **self-sustaining**, and could fight on its own until other corps could join in the battle. The corps itself was a headquarters to which units could be attached. **It might have attached two to four divisions of infantry with their organic artillery, it had its own cavalry division and corps artillery, plus support units.** With this organization a corps was expected to be able to hold its ground against, or fight off an enemy army for at least a day, when neighboring corps could come to its aid. "Well handled, it can fight or alternatively avoid action, and maneuver according to circumstances without any harm coming to it, because an opponent cannot force it to accept an engagement but if it chooses to do so it can fight alone for a long time."

The Diagram



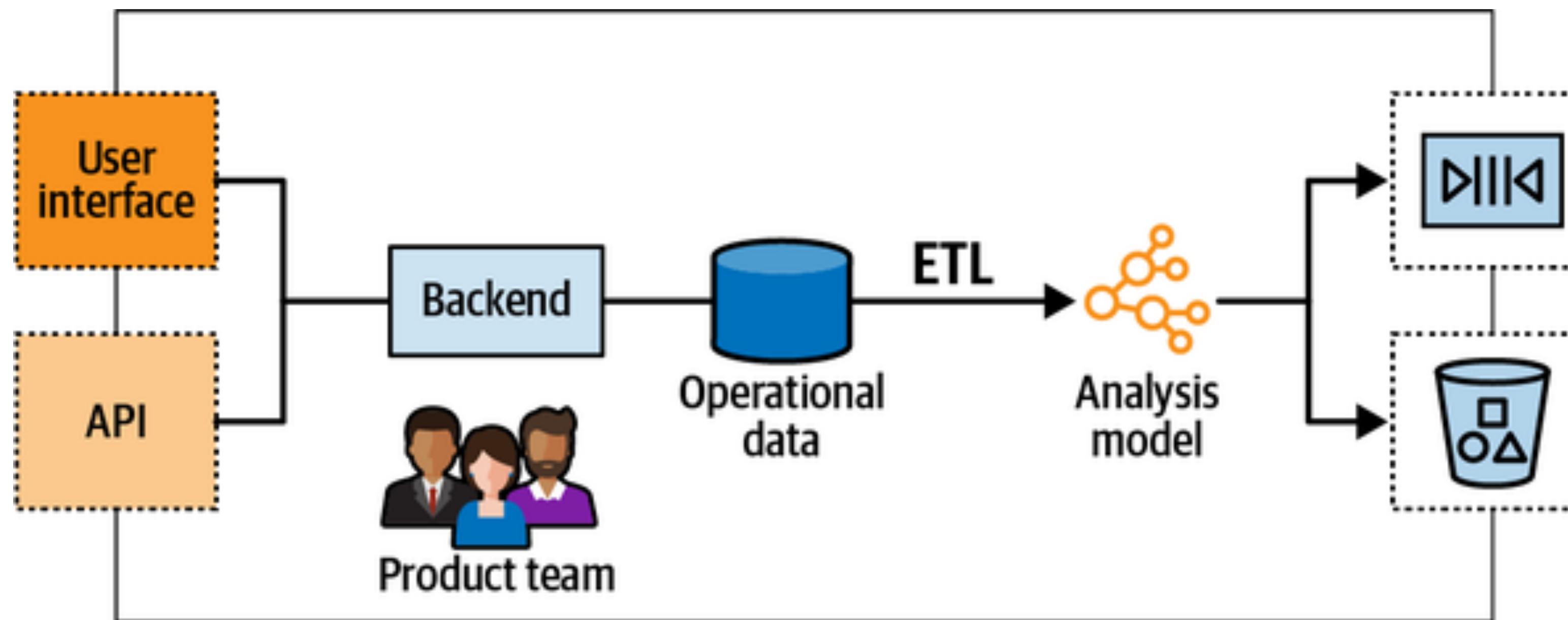
<https://martinfowler.com/articles/data-mesh-principles.html>

The Diagram

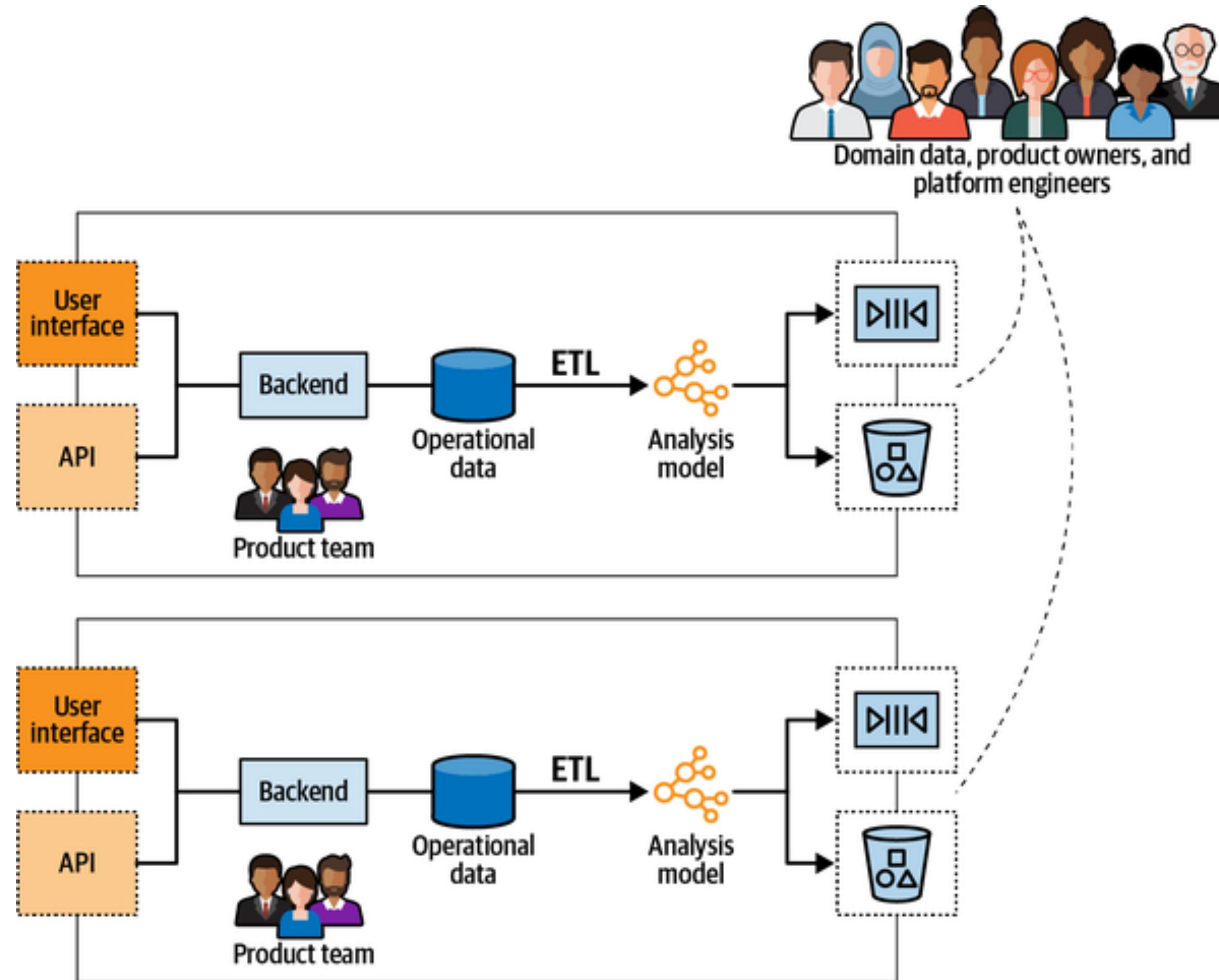


<https://martinfowler.com/articles/data-mesh-principles.html>

The Diagram



The Diagram



Open Metadata

The screenshot shows the homepage of the Open Metadata website. At the top, there is a navigation bar with links to 'Get Started', 'Documentation', 'Community', 'MCP', 'Blog', 'Case Studies', 'Product Updates', and a 'GitHub' button. To the right of the GitHub button is a purple button labeled 'Try OpenMetadata'. Below the navigation bar, there is a dark banner with a 'Try' icon and the text 'Try OpenMetadata as a managed service for free, from Collate.' followed by a 'Get Started' button. The main title 'Open and unified metadata platform for data discovery, observability, and governance' is displayed prominently in large, bold, black and purple text. Below the title, a subtitle reads: 'A single place for all your data and all your data practitioners to build and manage high quality data assets at scale. Built by Collate and the founders of Apache Hadoop, Apache Atlas, and Uber Databook.' At the bottom of the page, there are two buttons: 'JOIN SLACK' and 'TRY OPENMETADATA'.

Open and unified metadata platform for data discovery, observability, and governance

A single place for all your data and all your data practitioners to build and manage high quality data assets at scale. Built by Collate and the founders of Apache Hadoop, Apache Atlas, and Uber Databook.

JOIN SLACK

TRY OPENMETADATA

The screenshot also includes a preview of the Open Metadata platform's interface, showing a detailed view of a data asset's lineage, schema, and activity feed.

What is Open Metadata?

- Open-source metadata and data catalog platform
- Built for data discovery, governance, lineage, and observability
- Supports modern data stacks: Flink, Iceberg, Spark, Trino, Kafka, etc.
- Built for cataloging data, search and discovery, lineage tracking, governance, quality, schema versioning and more.



Iceberg, DataMesh, and Open Metadata

- Iceberg tables can be discovered, described, and governed through OpenMetadata's unified catalog.
- Visualize data lineage: track how Iceberg tables are created, modified, or queried by different engines (e.g., Flink, Spark, Trino).
- Monitor schema evolution and detect breaking changes across environments.
- Govern access, usage, and retention policies for Iceberg-managed data assets across teams.

Thank You



- Email: dhinojosa@evolutionnext.com
- Github: <https://www.github.com/dhinojosa>
- Linked In: <http://www.linkedin.com/in/dhevolutionnext>