Exabyte-scale Streaming Iceberg IO with Beam, Ray, and DeltaCAT

BEAM SUMMIT
NYC 2025

Patrick Ames
Principal Engineer, Amazon

Overview

- A Brief History of Amazon BI
- Apache Iceberg and DeltaCAT Overview
- The Iceberg Streaming Problem
- An Open Exabyte-Scale Solution
- Current State & Future Work

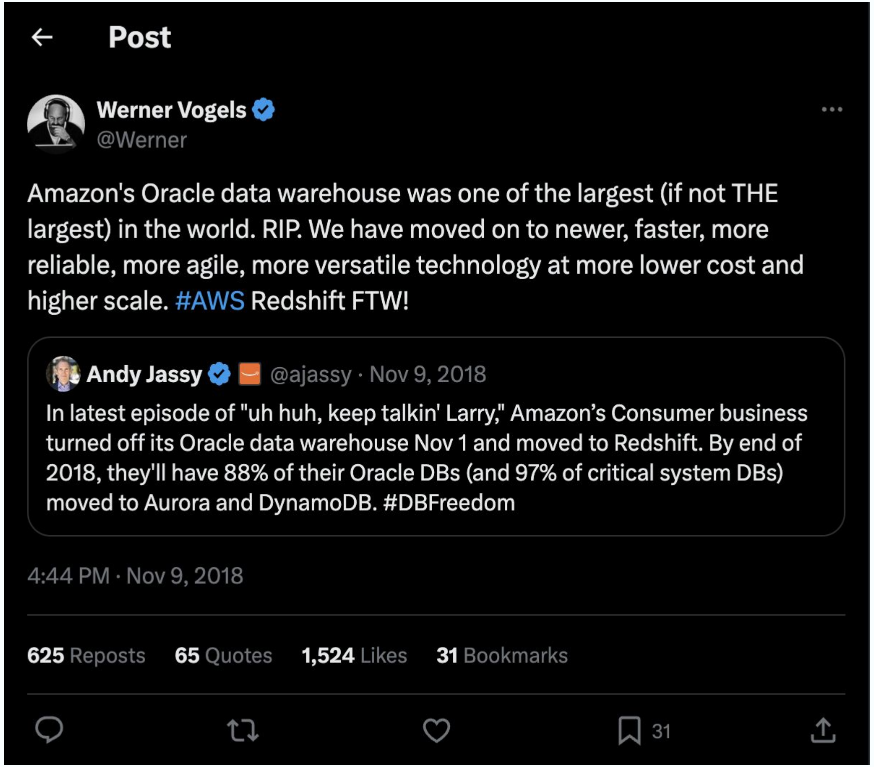
A Brief History of Amazon BI

2016–2018

PB-Scale Oracle Data Warehouse Deprecation

- Migrated 50PB from Oracle Data Warehouse to S3-Based Data Catalog
- Decoupled storage with Amazon Redshift & Apache Hive on Amazon EMR Compute

RIP Oracle Data Warehouse 🪦



← Post

Werner Vogels @Werner

Amazon's Oracle data warehouse was one of the largest (if not THE largest) in the world. RIP. We have moved on to newer, faster, more reliable, more agile, more versatile technology at more lower cost and higher scale. #AWS Redshift FTW!

> **Andy Jassy** @ajassy · Nov 9, 2018
>
> In latest episode of "uh huh, keep talkin' Larry," Amazon's Consumer business turned off its Oracle data warehouse Nov 1 and moved to Redshift. By end of 2018, they'll have 88% of their Oracle DBs (and 97% of critical system DBs) moved to Aurora and DynamoDB. #DBFreedom

4:44 PM · Nov 9, 2018

625 Reposts **65** Quotes **1,524** Likes **31** Bookmarks

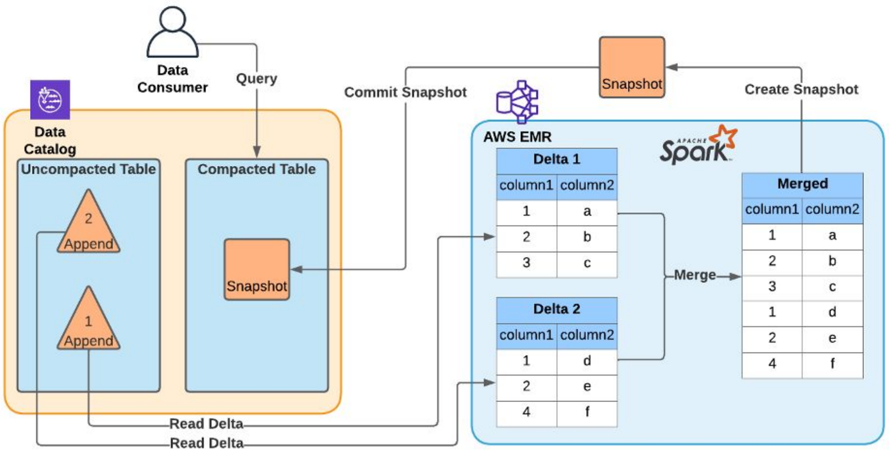
A Brief History of Amazon BI

2018–2019

EB-Scale Data Catalog & Lakehouse Formation

- Bring your own compute (EMR Spark, AWS Glue, Amazon Redshift Spectrum, etc.)
- LSM-based CDC "Compaction" using Apache Spark on Amazon EMR

Append-Only Compaction



Delta 1

column1	column2
1	a
2	b
3	c

Delta 2

column1	column2
1	d
2	e
4	f

Merged

column1	column2
1	a
2	b
3	c
1	d
2	e
4	f

Append deltas arrive in a table's CDC log stream, where each delta contains pointers to one or more S3 files containing records to insert into the table. During a compaction job, no records are updated or deleted so the delta merge is a simple concatenation, but the compactor is still responsible for writing out files sized appropriately to optimize reads (i.e. merge tiny files into larger files and split massive files into smaller files).

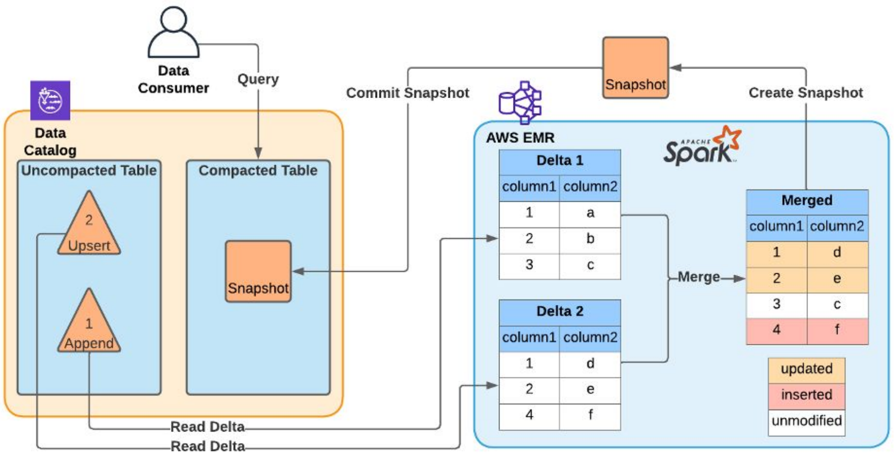
A Brief History of Amazon BI

2018–2019

EB-Scale Data Catalog & Lakehouse Formation

- Bring your own compute (EMR Spark, AWS Glue, Amazon Redshift Spectrum, etc.)
- LSM-based CDC “Compaction” using Apache Spark on Amazon EMR

Upsert Compaction



Delta 1

column1	column2
1	a
2	b
3	c

Delta 2

column1	column2
1	d
2	e
4	f

Merged

column1	column2
1	d
2	e
3	c
4	f

updated
inserted
unmodified

Append and Upsert deltas arrive in a table’s CDC log stream, where each Upsert delta contains records to update or insert according to one or more merge keys. In this case, column1 is used as the merge key, so only the latest column2 updates are kept per distinct column1 value.

A Quick Intro to Ray

RAY

- **Pythonic**
  - Provides distributed Python APIs for ML, data science, and general workloads.
- **Intuitive**
  - Relatively simple to convert single-process Python to distributed.
- **Scalable**
  - Can integrate PB-scale datasets with data processing and ML pipelines.
- **Performant**
  - Reduces end-to-end latency of data processing and ML workflows.
- **Efficient**
  - Reduces end-to-end cost of data processing and ML.
- **Unified**
  - Can run all steps of mixed data processing, data science, and ML pipelines.

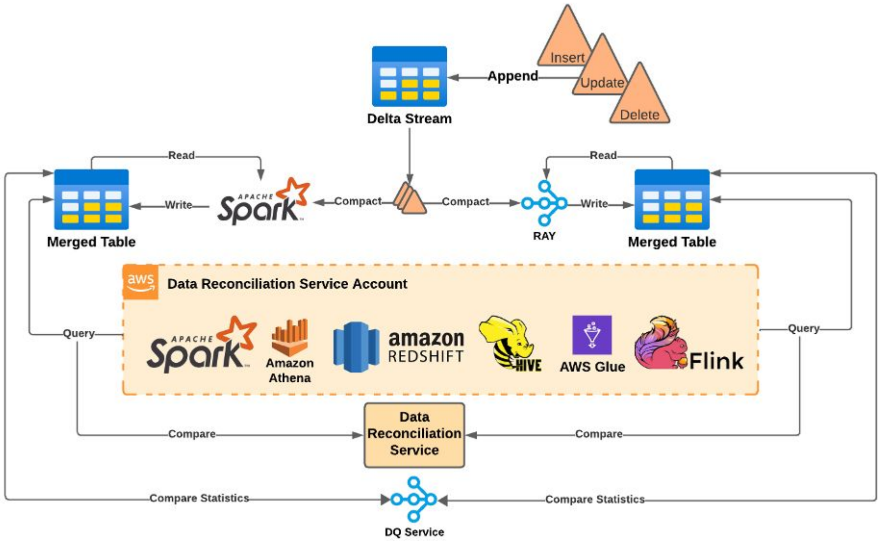
A Brief History of Amazon BI

2019–2023

Ray Integration

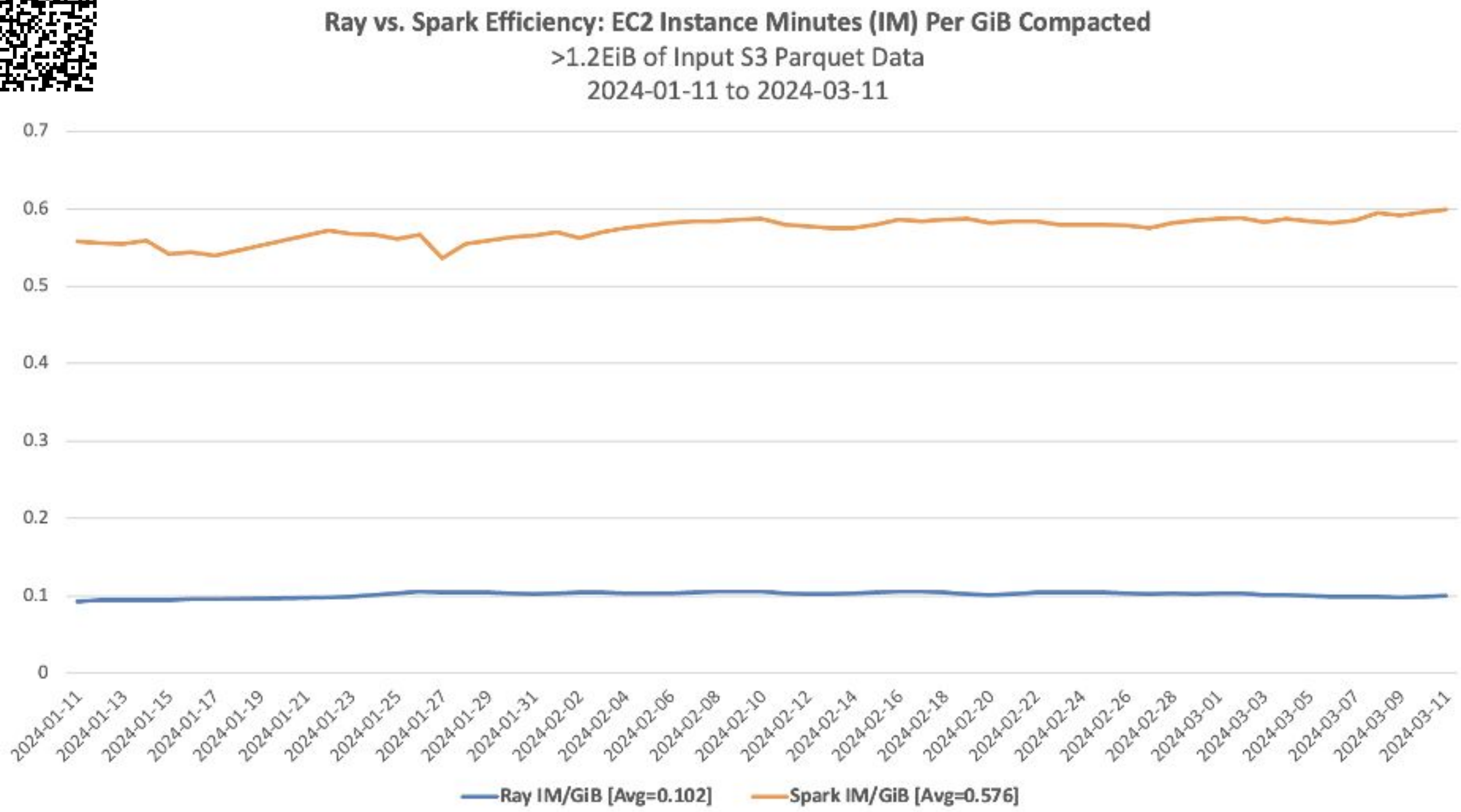
- EB-Scale Data Quality Analysis
- Spark-to-Ray Compaction Migration
- Reduced Cost by 82% (equivalent to $120MM/year of Amazon EC2 on-demand charges)

Ray Shadow Compaction



New deltas arriving in a data catalog table's CDC log stream are merged into two separate compacted tables maintained separately by Apache Spark and Ray. The Data Reconciliation Service verifies that different data processing frameworks produce equivalent results when querying datasets produced by Apache Spark and Ray, while the Ray-based DQ Service compares key dataset statistics.

Ray vs. Spark Compactor Efficiency



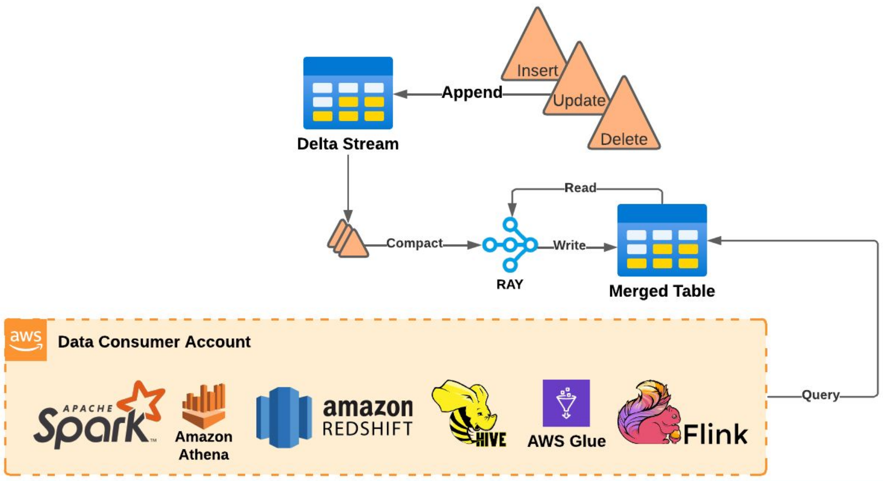
A Brief History of Amazon BI

2024–2025

Ray Exclusivity

- Migrate all Table Queries to Ray Compactor Output
- Turn off Spark Compactor
- OSS implementation of Ray Compactor in DeltaCAT

Ray Exclusive Compaction



New deltas arriving in a data catalog table's CDC log stream are merged into only one compacted table maintained by Ray. Amazon BI tables are gradually being migrated from Spark compaction to Ray-Exclusive compaction, starting with our largest tables.

A Quick Intro to DeltaCAT

A Portable Pythonic Data Lakehouse Powered by Ray

- **Catalog**: High-level APIs to create, discover, organize, share, and manage datasets.
- **Compute**: Distributed data management procedures to read, write, and optimize datasets.
- **Storage**: In-memory and on-disk multimodal dataset formats.
- **Sync**: Synchronize DeltaCAT datasets to data warehouses and other table formats.

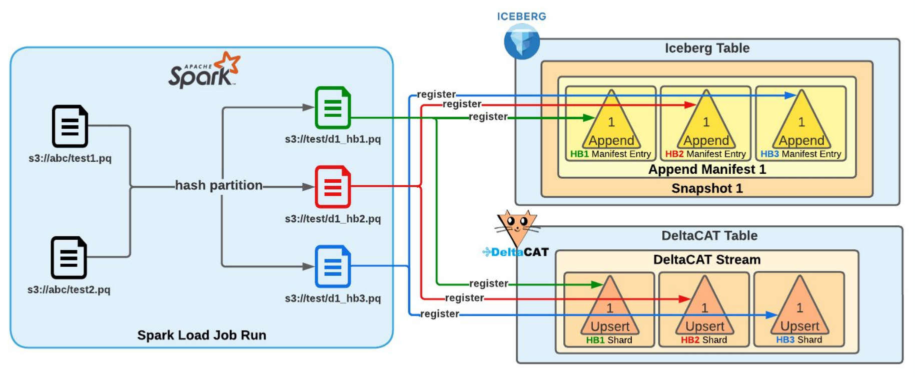
v1.X Used in Production @ Amazon (But Also Overfit to Amazon’s Use-Case)
v2.0 in Development for General Purpose Use (e.g. Iceberg Table Management)

A Brief History of Amazon BI

2025+
Lakehouse Unification

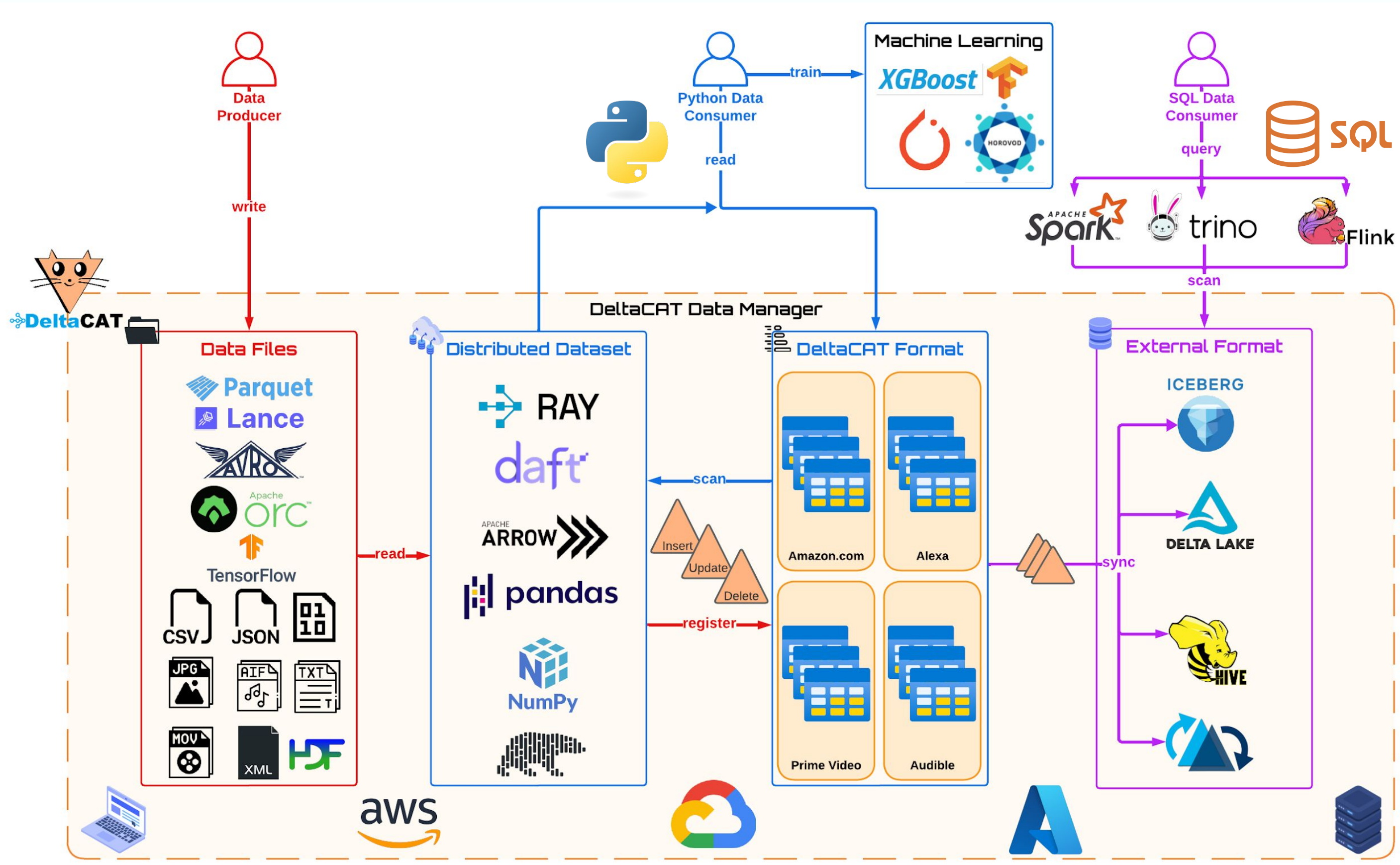
- Reuse data files across multiple table formats
- Batch & Streaming Compute Integration with Iceberg
- OSS Implementations of Iceberg Table Maintenance Jobs using Ray in DeltaCAT

Shared Data Files

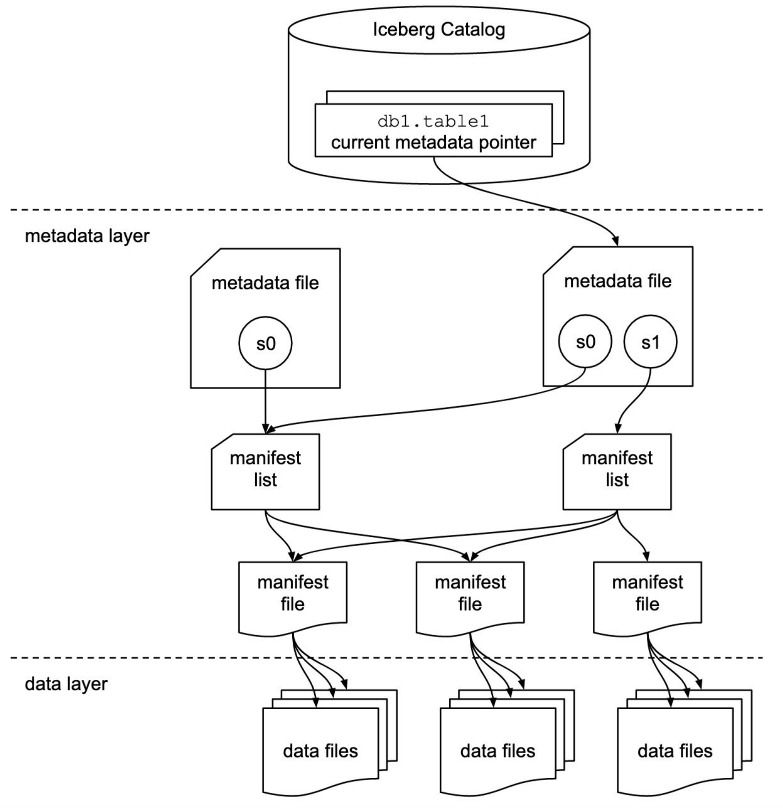


New deltas arriving in a data catalog table's CDC log stream reuse the same data files across Iceberg, DeltaCAT, and any other compatible table formats (e.g., Hudi, Hive, etc.).

DeltaCAT Overview



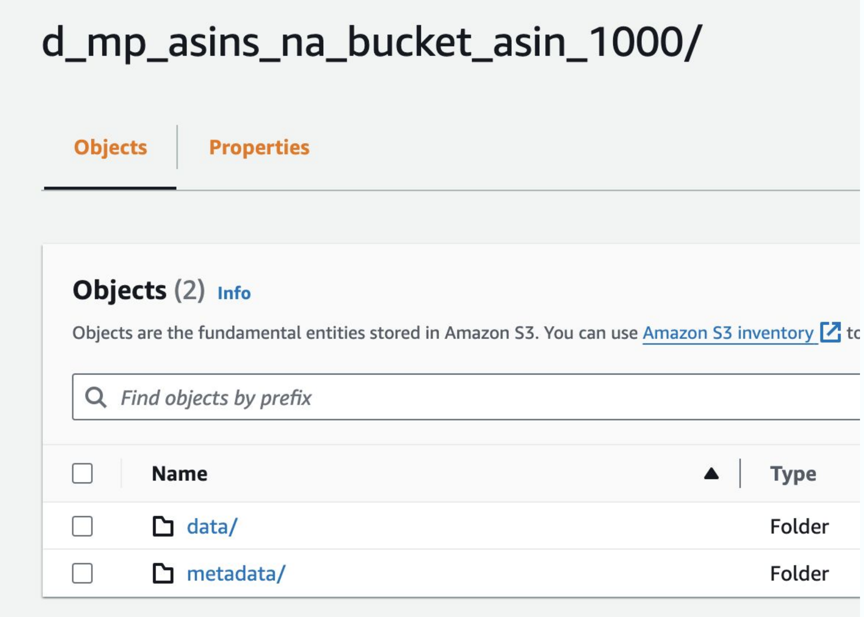
Iceberg Overview



- **Table Metadata Format**
    - Table Metadata File
    - Snapshot File
    - Manifest File
    - Data File

- **Catalog**
    - Table Metadata Pointer
    - Java & Python **Interfaces**
    - REST/Hive/JDBC/Glue/etc. **Implementations**

Iceberg Table Directory



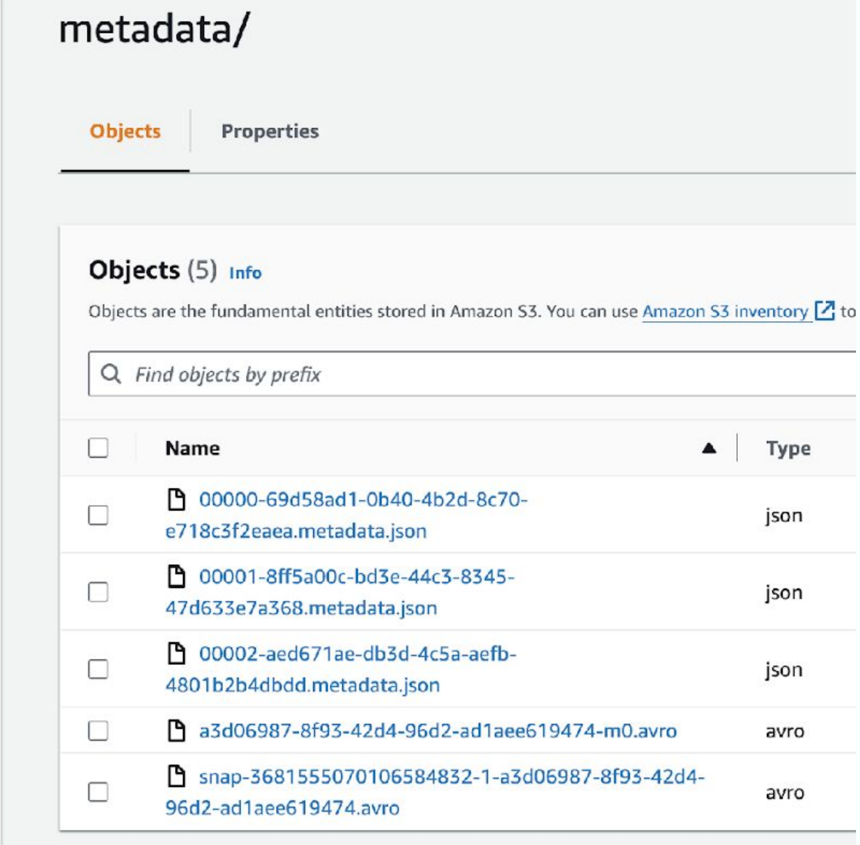
d_mp_asins_na_bucket_asin_1000/

Objects (2) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to

□	Name ▲	Type
□	data/	Folder
□	metadata/	Folder

Iceberg Metadata Directory



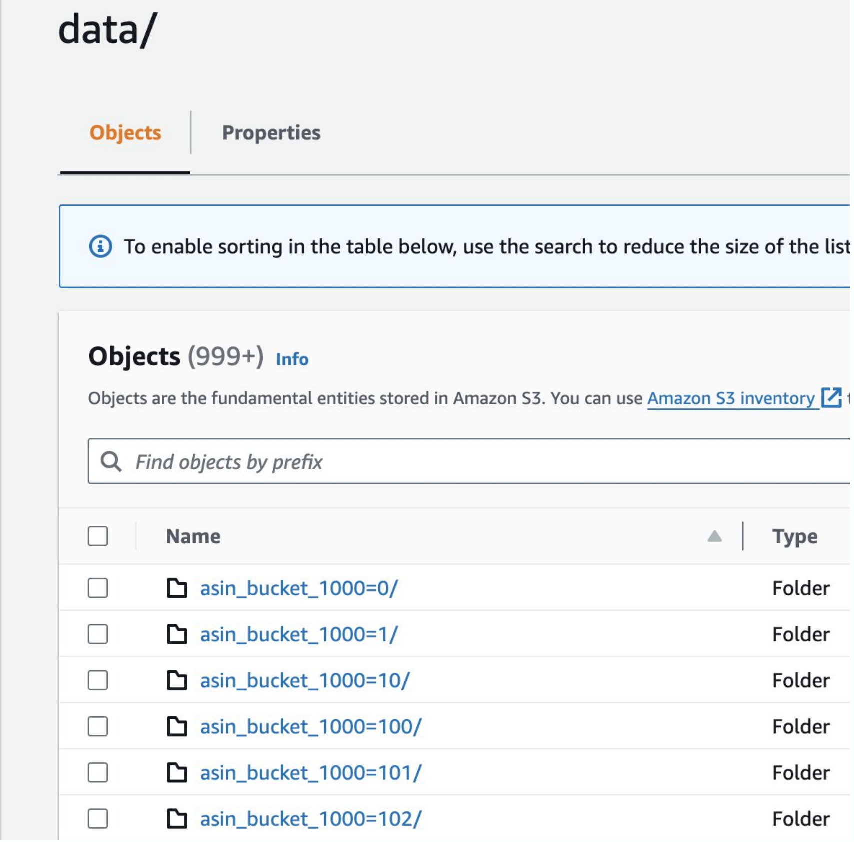
metadata/

Objects (5) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to

□	Name ▲	Type
□	00000-69d58ad1-0b40-4b2d-8c70-e718c3f2eaea.metadata.json	json
□	00001-8ff5a00c-bd3e-44c3-8345-47d633e7a368.metadata.json	json
□	00002-aed671ae-db3d-4c5a-aefb-4801b2b4dbdd.metadata.json	json
□	a3d06987-8f93-42d4-96d2-ad1aee619474-m0.avro	avro
□	snap-3681555070106584832-1-a3d06987-8f93-42d4-96d2-ad1aee619474.avro	avro

Iceberg Data Directory



data/

Objects | Properties

ⓘ To enable sorting in the table below, use the search to reduce the size of the list

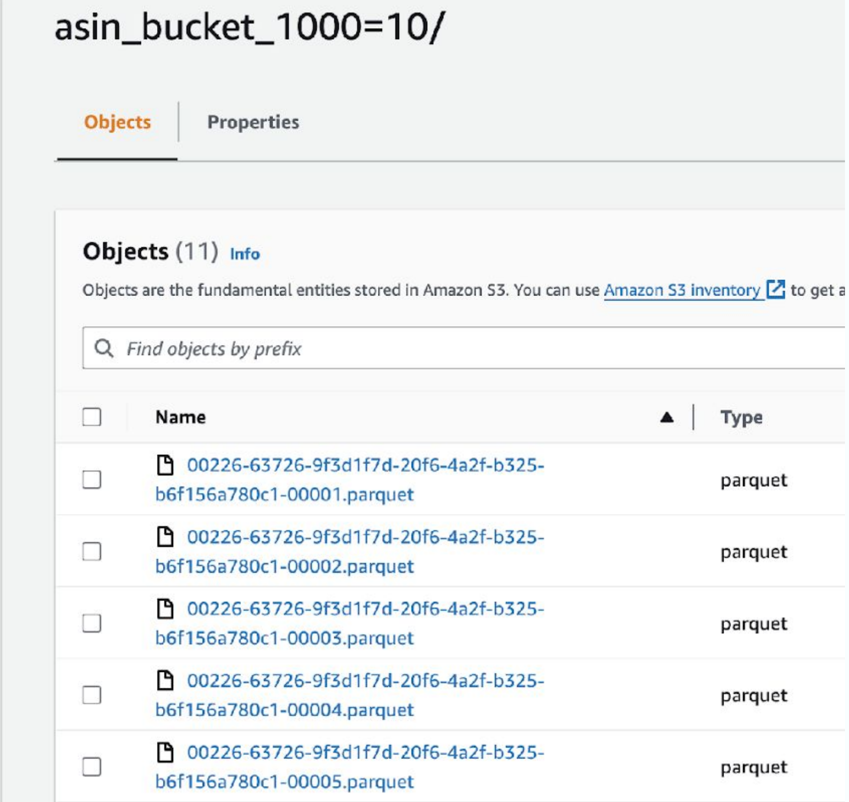
Objects (999+) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory t

🔍 *Find objects by prefix*

□	Name ▲	Type
□	asin_bucket_1000=0/	Folder
□	asin_bucket_1000=1/	Folder
□	asin_bucket_1000=10/	Folder
□	asin_bucket_1000=100/	Folder
□	asin_bucket_1000=101/	Folder
□	asin_bucket_1000=102/	Folder

Iceberg Partition Directory



asin_bucket_1000=10/

Objects | Properties

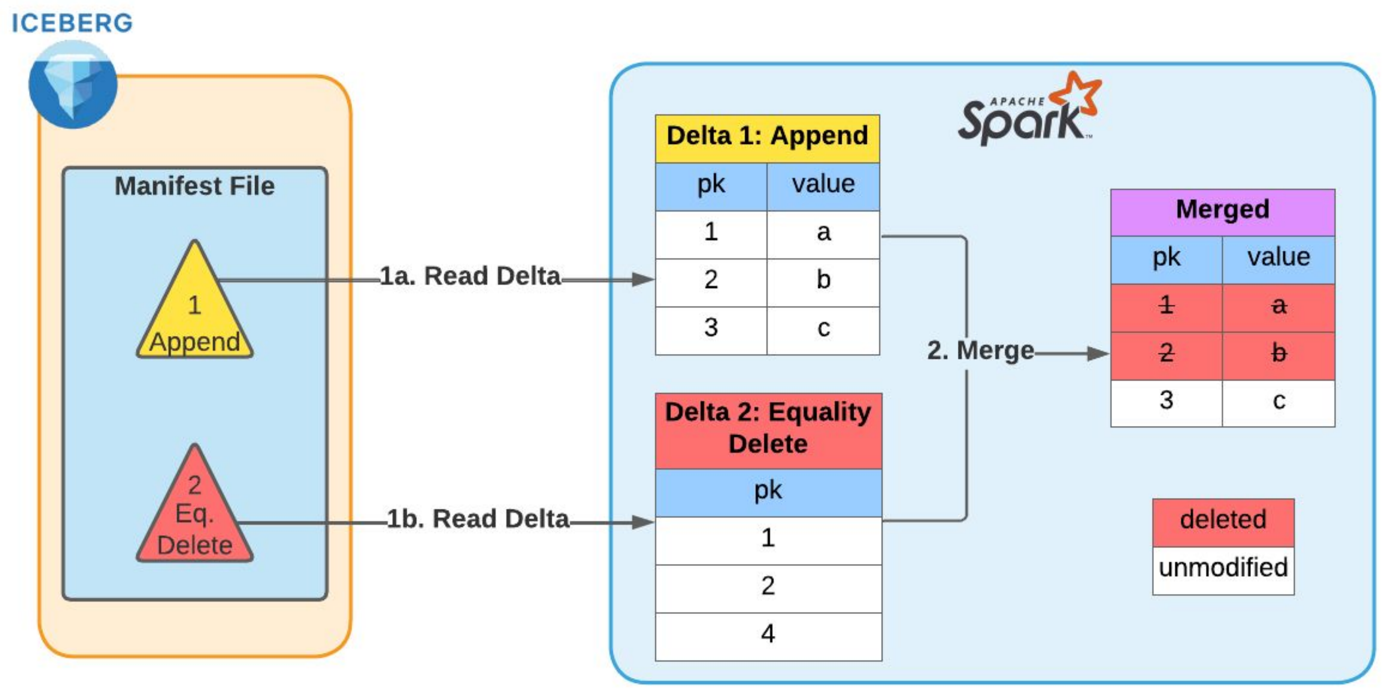
Objects (11) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a

🔍 *Find objects by prefix*

□	Name ▲	Type
□	00226-63726-9f3d1f7d-20f6-4a2f-b325-b6f156a780c1-00001.parquet	parquet
□	00226-63726-9f3d1f7d-20f6-4a2f-b325-b6f156a780c1-00002.parquet	parquet
□	00226-63726-9f3d1f7d-20f6-4a2f-b325-b6f156a780c1-00003.parquet	parquet
□	00226-63726-9f3d1f7d-20f6-4a2f-b325-b6f156a780c1-00004.parquet	parquet
□	00226-63726-9f3d1f7d-20f6-4a2f-b325-b6f156a780c1-00005.parquet	parquet

Iceberg Equality Deletes



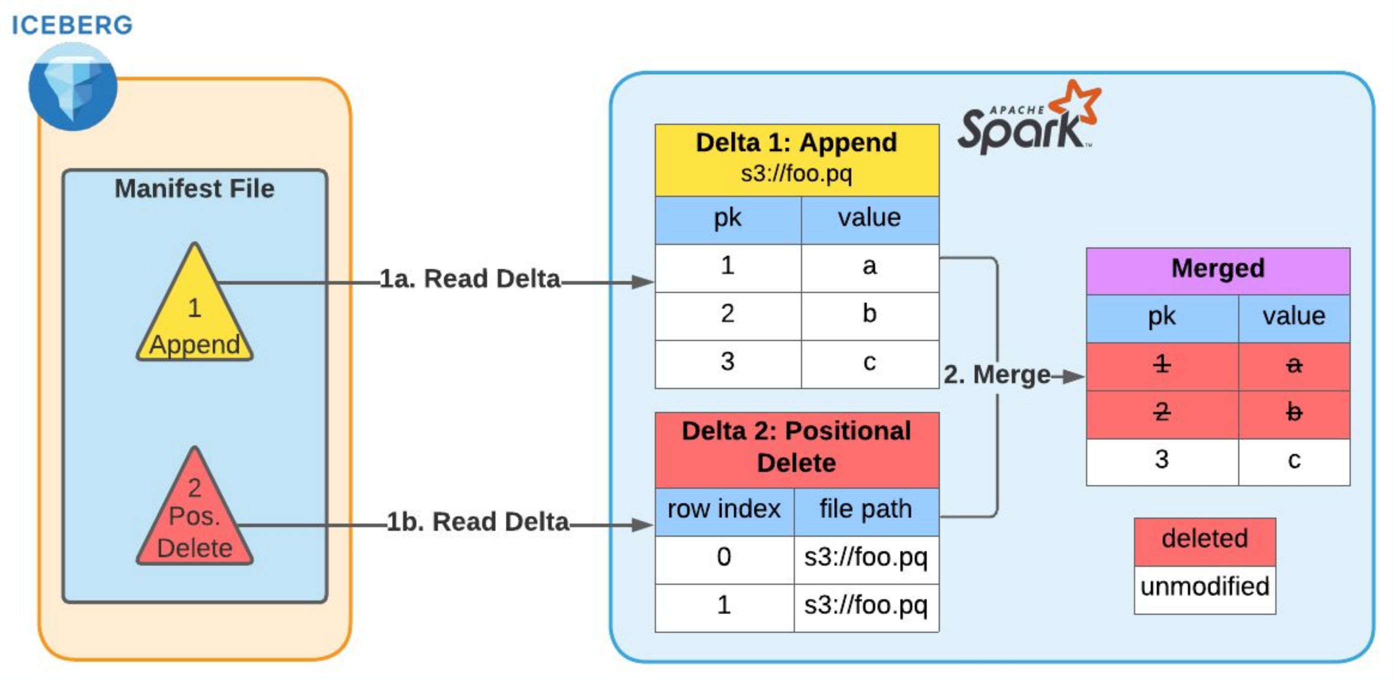
Delta 1: Append	
pk	value
1	a
2	b
3	c

Delta 2: Equality Delete
pk
1
2
4

Merged	
pk	value
~~1~~	~~a~~
~~2~~	~~b~~
3	c

deleted
unmodified

Iceberg Positional Deletes



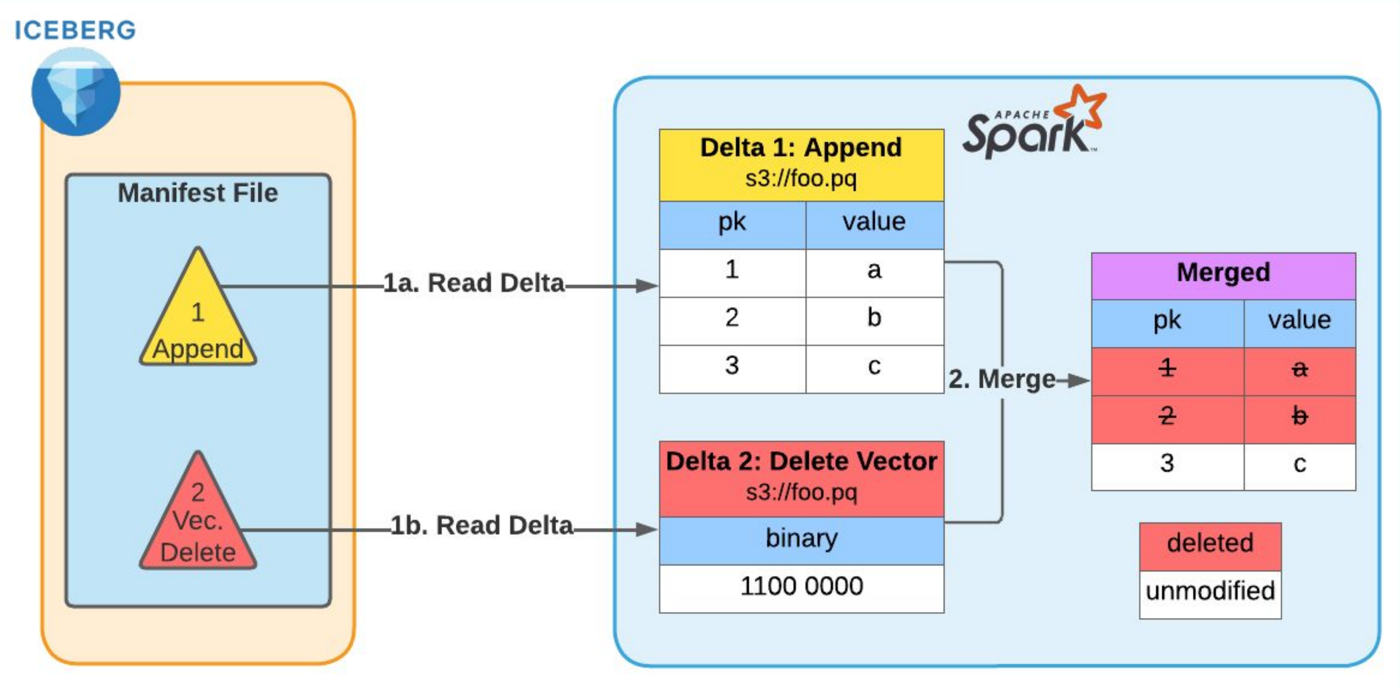
Delta 1: Append s3://foo.pq	
pk	value
1	a
2	b
3	c

Delta 2: Positional Delete	
row index	file path
0	s3://foo.pq
1	s3://foo.pq

Merged	
pk	value
~~1~~	~~a~~
~~2~~	~~b~~
3	c

deleted
unmodified

Iceberg Delete Vectors



Delta 1: Append
s3://foo.pq

pk	value
1	a
2	b
3	c

Delta 2: Delete Vector
s3://foo.pq

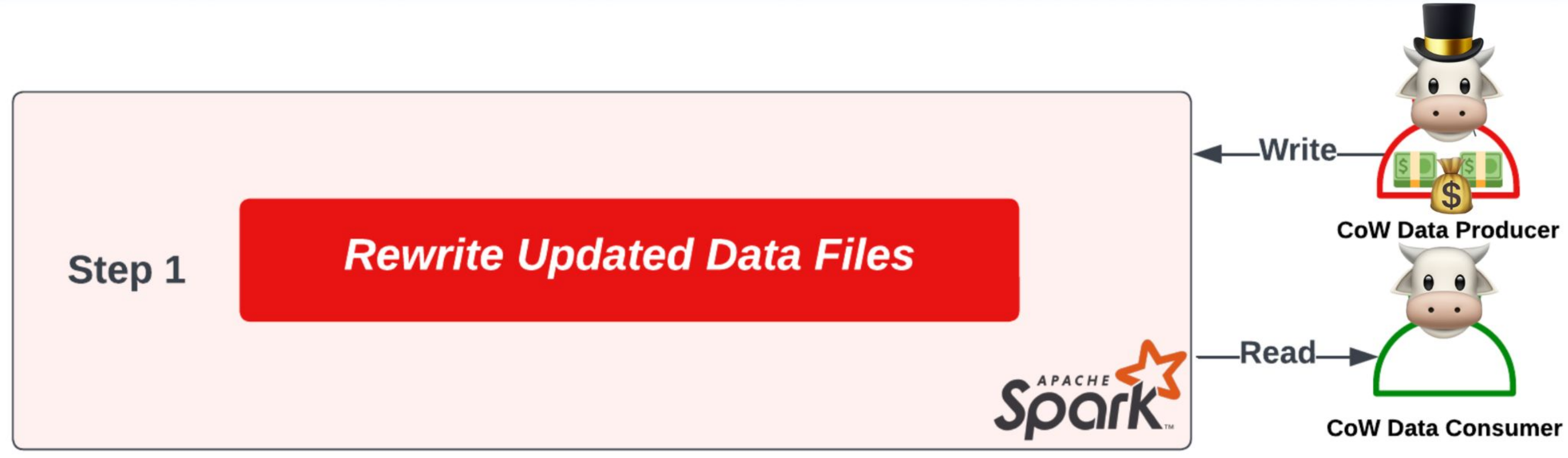
binary
1100 0000

Merged

pk	value
~~1~~	~~a~~
~~2~~	~~b~~
3	c

deleted
unmodified

Iceberg Copy on Write



Step 1

Rewrite Updated Data Files

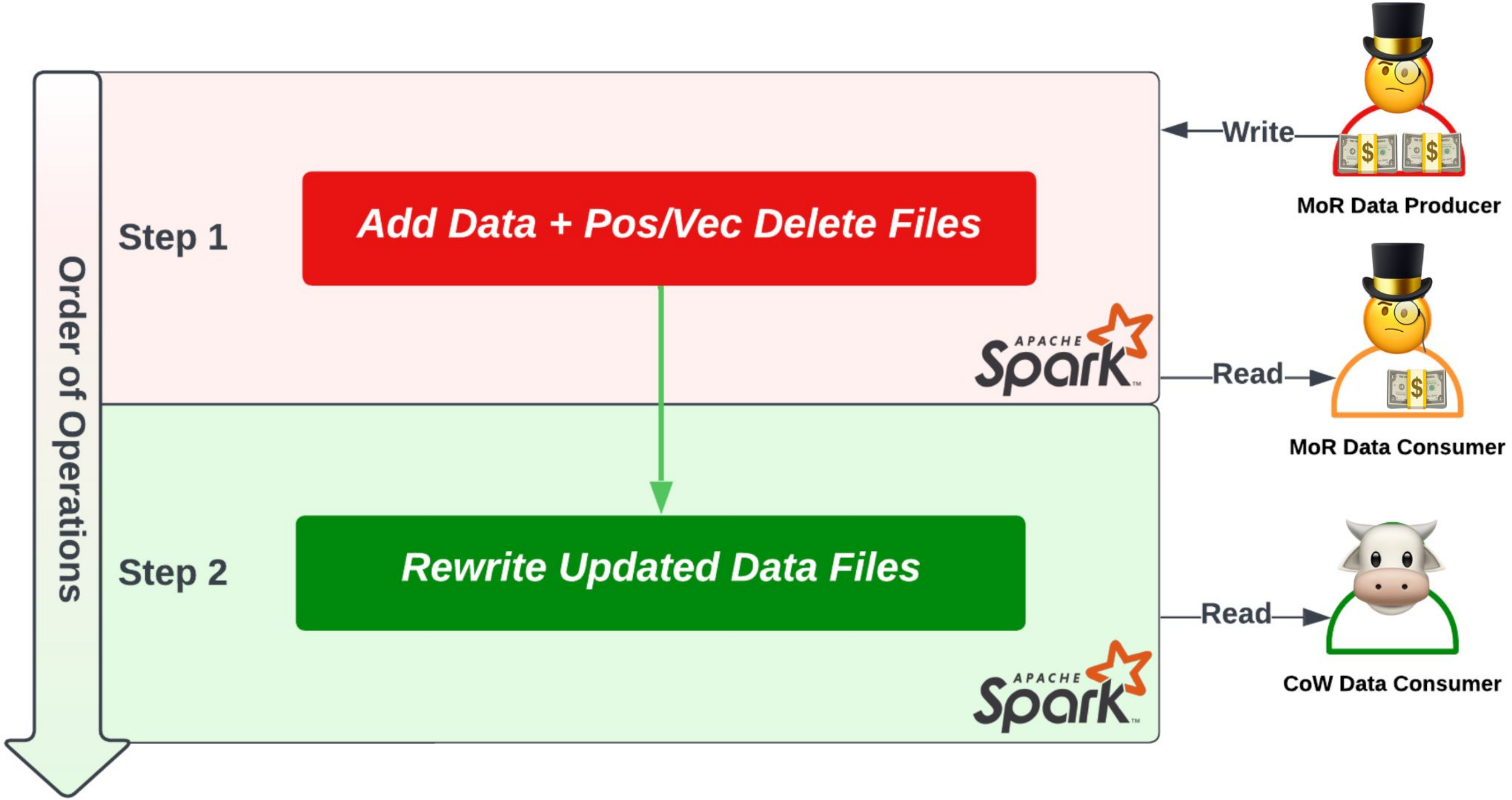
Write

Read

CoW Data Producer

CoW Data Consumer

Iceberg Merge on Read



The Iceberg Streaming Problem

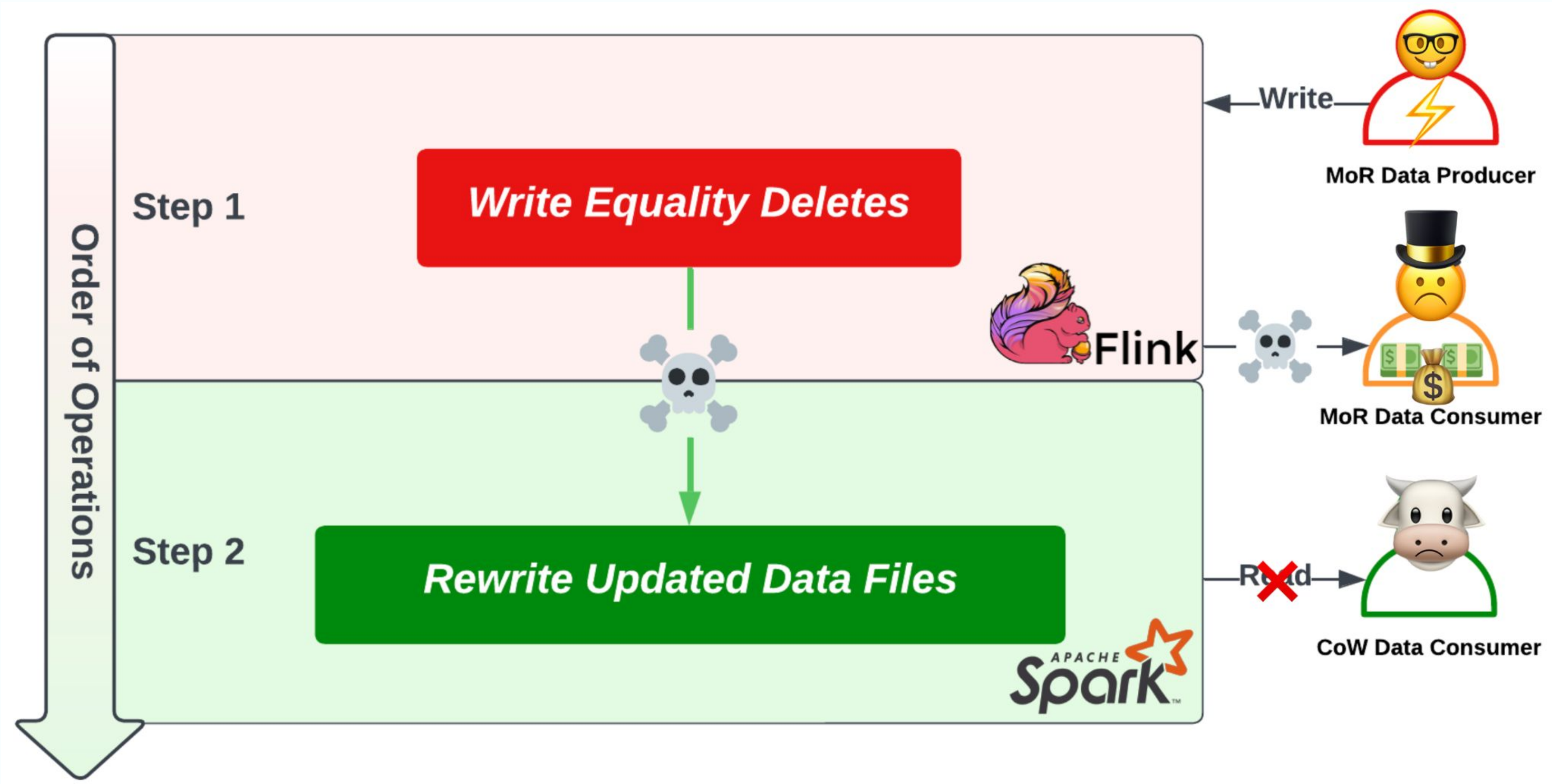
- Merge-on-Read Tables
    - **Streaming** Frameworks like Flink Prefer to Write
        - **Equality Deletes** (Cheap Writes, Expensive Reads)
        - **Fast & Conflict-Free** but may cause **OOM Errors** on Read ☹️
    - **Batch** Frameworks like Spark Write
        - **Positional Deletes** (IcebergV2)
        - **Binary Delete Vectors** (IcebergV3)
        - Moderately Expensive Writes, Cheaper Reads
        - **Less practical for High-Frequency Writes** at TB-PB Scale ☹️
        - Why? **High-Latency** and Susceptible to **Irresolvable Write Conflicts**

- Copy-on-Write Tables
    - **Always Rewrite Data Files** w/ Deletes Applied (Cheapest Reads, Expensive Writes)
    - **Impractical for High-Frequency Writes** at TB-PB Scale ☹️

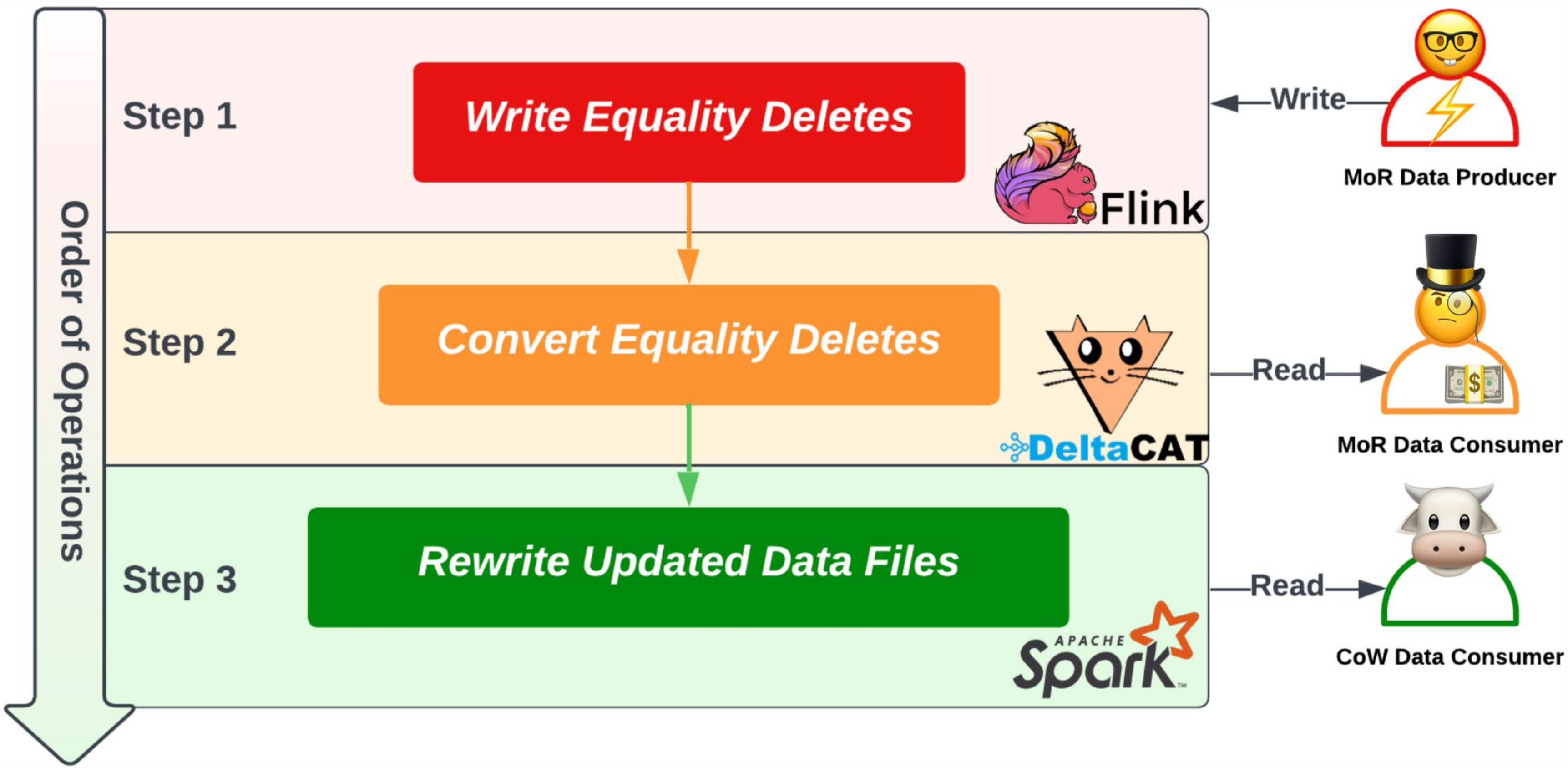
The Iceberg Streaming Problem

- **Issues & PRs Raised and Abandoned Since 2020**
  - 2020-05-08: Add an action to rewrite equality deletes as position deletes
  - 2021-02-04: Spark: support replace equality deletes to position deletes
  - 2021-03-23: Add an action to rewrite equality deletes
  - 2023-02-27: Data file rewriting spark job fails with oom
  - 2023-12-04: RewritePositionDeleteFiles cannot work with equality delete file?
  - 2024-03-27: Spark rewrite Files Action OOM
  - 2024-07-09: Add RocksDBStructLikeSet for storing equality deletes

The Iceberg Streaming Problem



An Open Exabyte-Scale Solution



Order of Operations

Step 1

Write Equality Deletes

Step 2

Convert Equality Deletes

Step 3

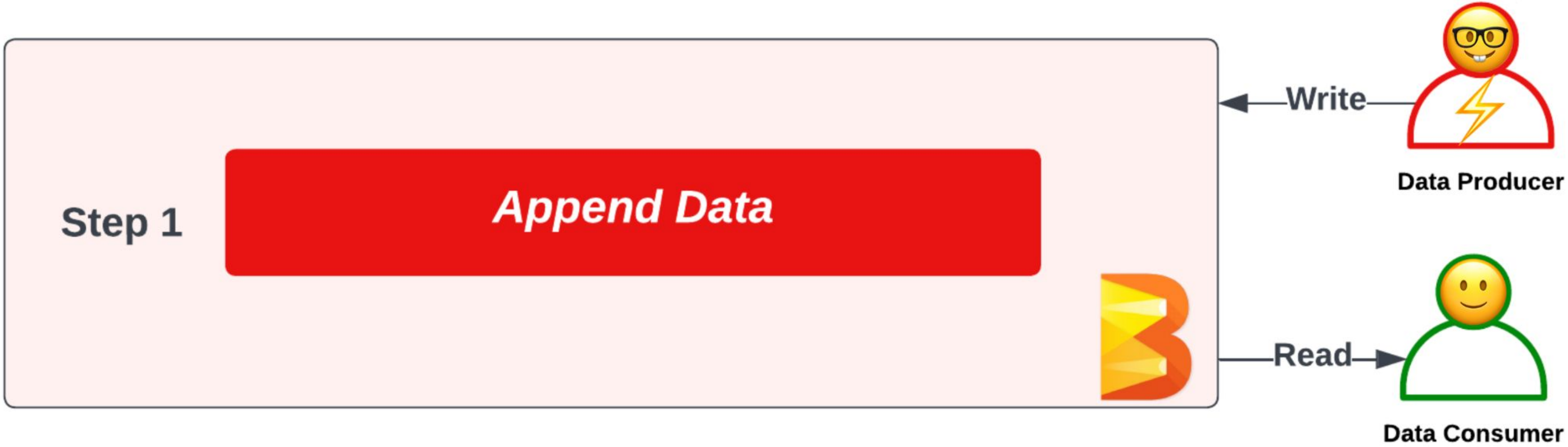
Rewrite Updated Data Files

MoR Data Producer

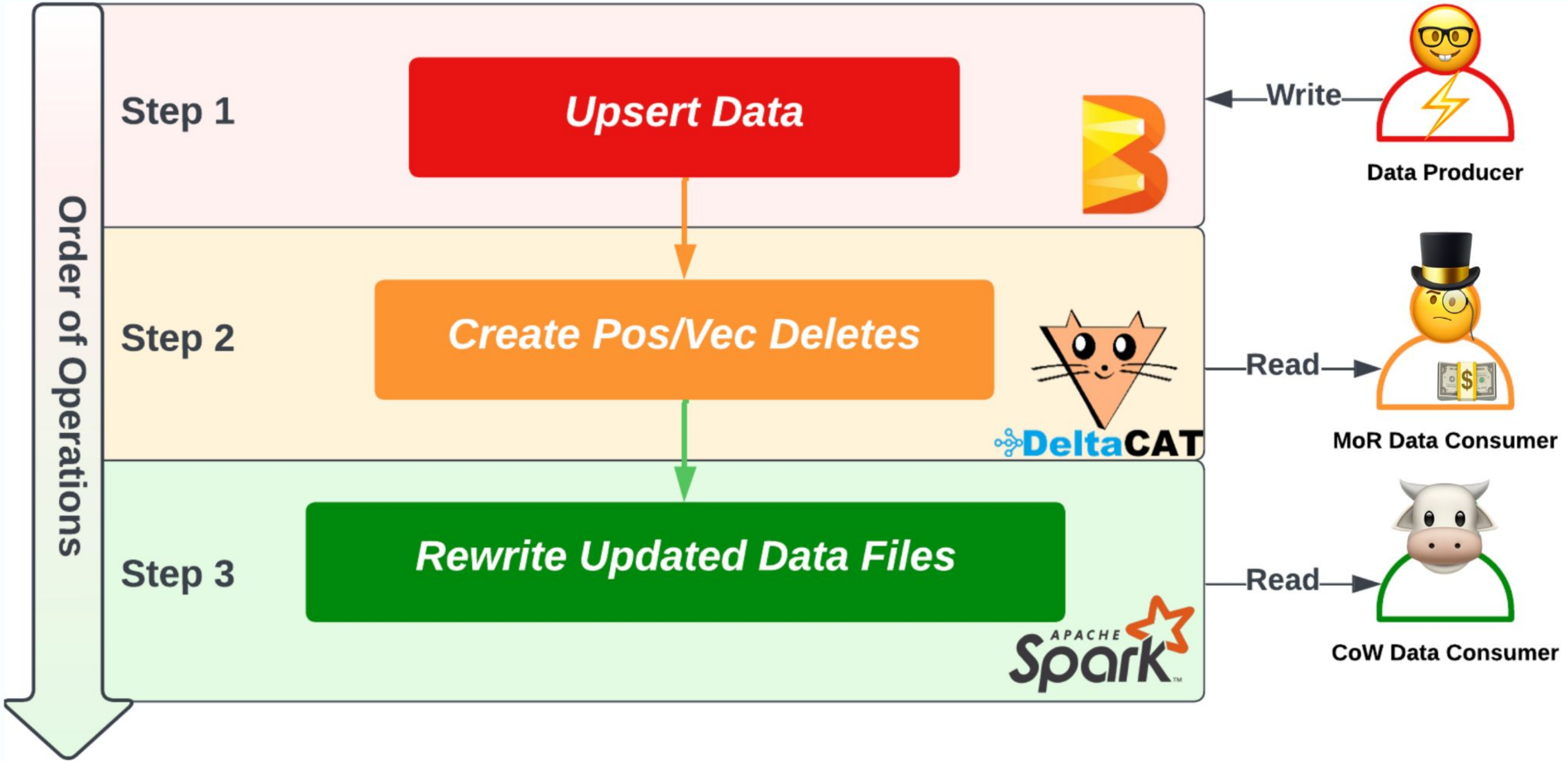
MoR Data Consumer

CoW Data Consumer

An Open Exabyte-Scale Solution



An Open Exabyte-Scale Solution



Current State

amazon

Testing with PB-scale production Iceberg tables

Q4 2025: Production onboarding to **EB-scale** production data catalog

DeltaCAT

Python converter job ready for experimental use

Beam Iceberg **upserts** via `beam.managed.Read()/Write()` wrapper

Flink Iceberg **delete conversion** (equality to positional/vector)

Run **manually** or **automatically** via Iceberg table monitor agent

Run **locally** or on a **distributed** Ray cluster (in GCP, AWS, etc.)

https://github.com/ray-project/deltacat/tree/2.0/deltacat/examples/experimental/iceberg/converter/

Future Work

- **Python Native Beam IO Connector?**
    - +1 Github Issue #561 @ https://github.com/ray-project/deltacat/issues/561

- **Flink IO Connector?**
    - +1 Github issue #562 @ https://github.com/ray-project/deltacat/issues/562

- **Ray Positional/Vector Delete Materialization for Iceberg?**
    - +1 Github issue #121 @ https://github.com/ray-project/deltacat/issues/121

Patrick Ames

QUESTIONS?

@Patrick Ames on Ray Slack
https://github.com/ray-project/deltacat
https://github.com/pdames