Toward a highly-reliable and high-performance MetaStore for Datafuse

Datafuse Meetup

Outline

- **Tables Format**
  - Hive Tables
- Iceberg Tables
- Improvement Over Iceberg
  - Buffered Writes
  - Iceberg v2
  - Nessie
- Possible Direction

Table Format

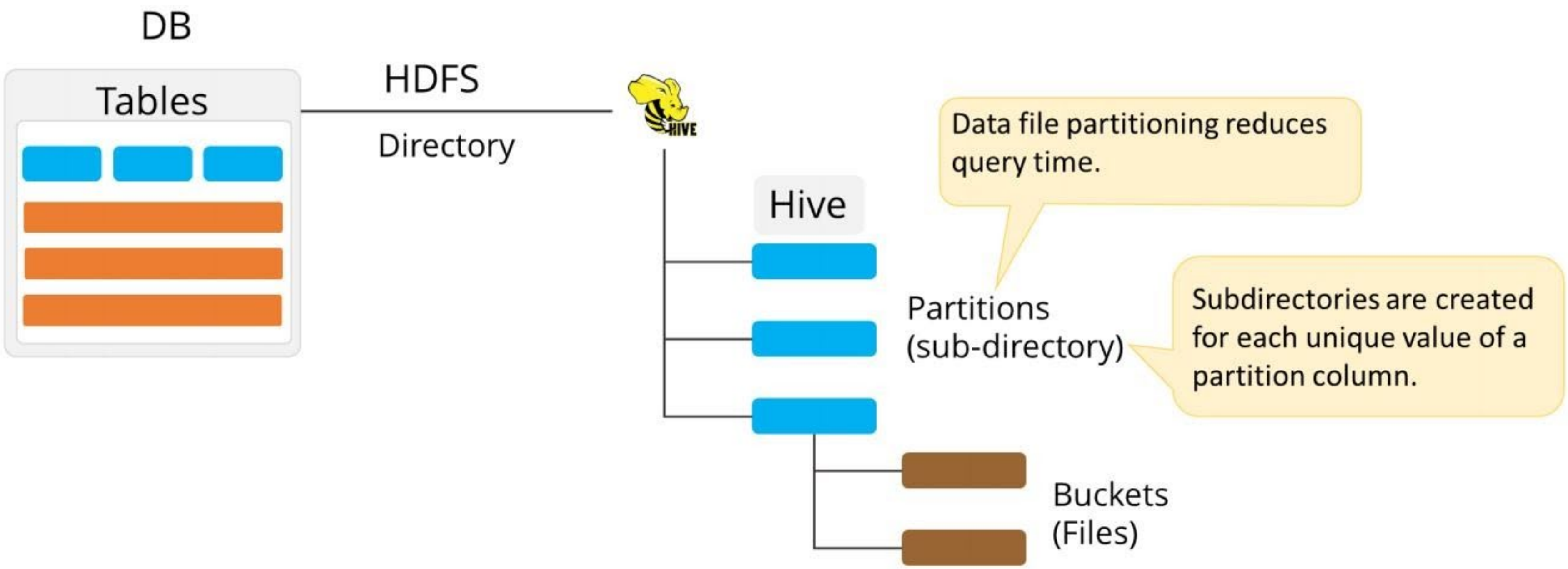
- The function of a table format is to determine:
  - How to manage or organize a table
  - How to track all the files that make up a table
  - Eg: Hive or Iceberg table format
- Table format is not file format
  - Files in the table is Avro, Parquet, ORC, etc.
- Determines multiple aspect of the data warehouse
  - What correctness guarantees are possible
  - How hard is it to write fast queries
  - How the table can change over time
  - Job performance

What is a good table format

- Should support expected database table behaviors
  - Atomic changes to commit all rows or nothing
  - Schema evolution without unintended consequences
  - Efficient access like predicate or projection pushdown
- Bonus features
  - Hidden layout: no need to know the table structure
  - Layout evolution: change the table structure over time

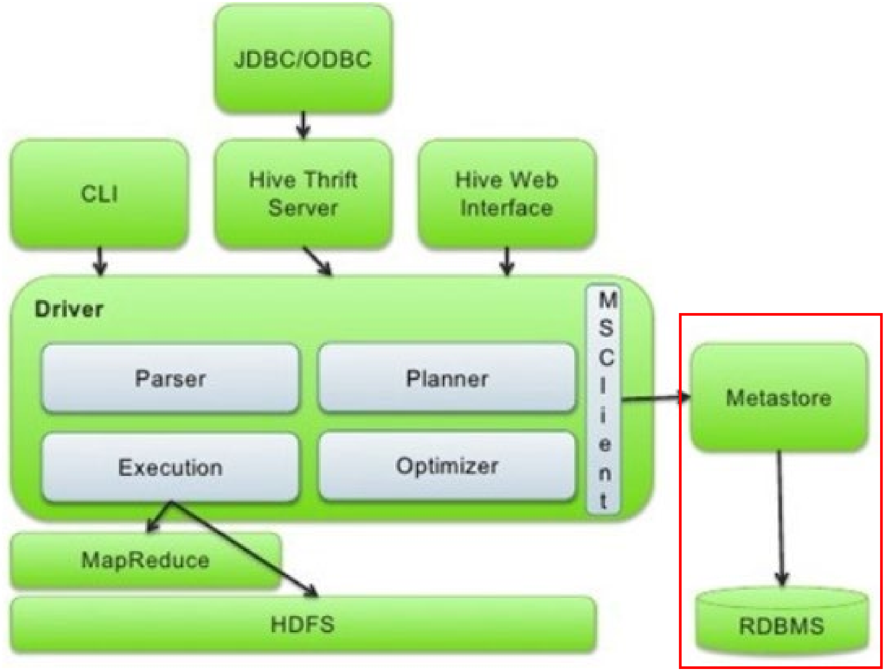
Hive Tables

- Key idea: organize data in a directory tree
- Filter directories as columns
  - SELECT ... WHERE date = '20180513' AND hour = 19



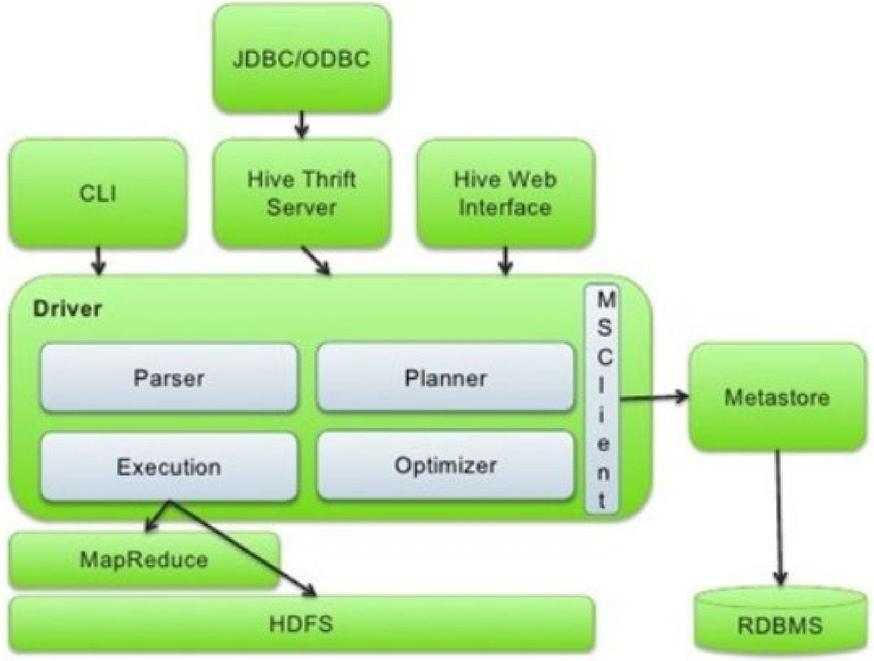
Hive MetaStore

- HMS keeps metadata in SQL database
  - Tracks information about partitions
  - Tracks schema information
  - Tracks table statistics
- Allows filtering by partition values
- Uses external SQL database
  - Such as MySQL or PostgreSQL
- Only FS tracks the files in each partition
  - No per-file statistics



Design Problems

- Table state is stored in two places
  - Partitions in the Hive Metastore
  - Files in a file system
- Requires atomic move of objects in FS
  - Move a staging directory to actual location
  - S3’s move is not atomic
- Keeps track of data at the “folder” level
  - Needs to perform file listing to plan jobs
  - O(n) listing calls, n = # matching partitions
  - Data to appear missing for list operations
    - with eventually consistency like S3
- Metastore is a bottleneck for query planning



Example – Netflix Atlas

- Historical Atlas data:
  - Time-series metrics from Netflix runtime systems
- Sample Query:
  - select distinct tags['type'] as type from atlas
    where name = 'metric-name' and
    date > 20180222 and date <= 20180228
    order by type;
- Hive table – with Parquet filters:
  - 400k+ partitions
  - EXPLAIN query: **9.6 min** (planning wall time)

```
date=20180513/
 |- hour=18/
 |    |- ...
 |- hour=19/
 |    |- 000000_0
 |    |- ...
 |    |- 000031_0
 |- hour=20/
 |    |- ...
 |- ...
```

Hive Table Limitations

- Data Reliability
  - Scheme changes lead to type inconsistency and corruptions
  - Failed jobs results in partial results being read by consumer
  - Concurrent readers and writers results in conflicts and data loss
  - Data restatement result in inconsistent reads
- Performance
  - Slow directory and file listing
  - Coarse-grained split planning results in slow scan

Outline

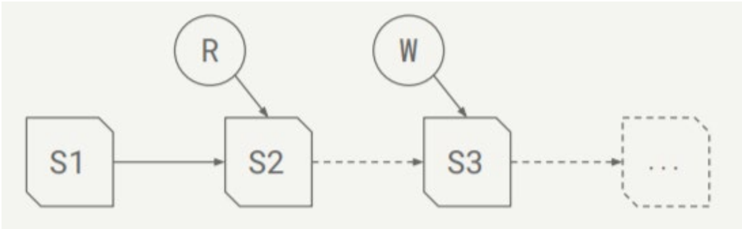
- Tables Format
  - Hive Tables
- **Iceberg Tables**
- Improvement Over Iceberg
  - Buffered Writes
  - Iceberg v2
  - Nessie
- Possible Direction

Apache Iceberg

- Apache Iceberg is an **open table format**
  - designed for huge, petabyte-scale tables
- Designed to address issues of Hive when used with S3
  - Consistency
  - Performance
- Integrated with Spark, Trino, Flink, etc.

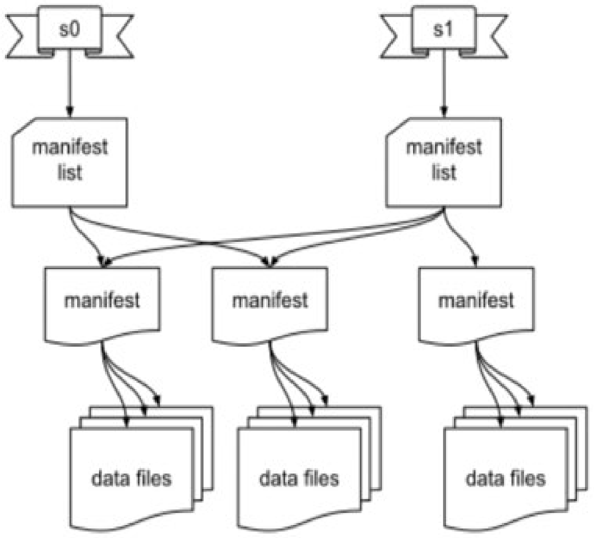
Iceberg Design

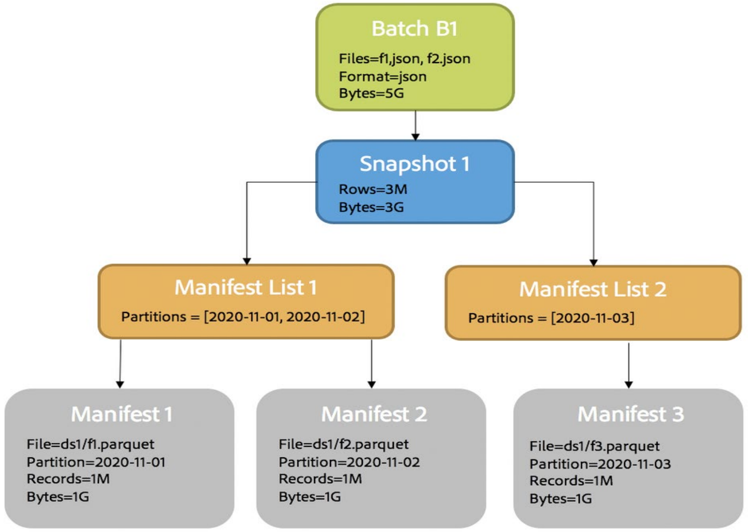
- Keep track of a complete list of **all** files within a table
  - With a tree structure
  - No expensive list operations for folders
- Changes to a table use an atomic object/file level commit
  - Update the path to a new metadata file
- Snapshot isolation without locking (MVCC)
- The table state stored in multiple metadata files



Metadata Files

- Snapshot metadata file
  - the table schema, the partition specification
  - a path to the manifest list
- Manifest list
  - paths to the manifest files
  - partition stats, data file count
- Manifest file
  - an immutable Avro file
  - a list of paths to related data files
  - per-column upper and lower bounds for the data file
- Data files
  - the file to store physical data with format Parquet, ORC, etc.





Benefits

- Readers always see a consistent view of the data
  - No dirty reads
- Writers work in isolation
  - Not affecting the live table
  - All changes are atomic
    - Perform a metadata swap when completed with a CAS operation
  - Optimistic concurrency control
- O(1) RPC to read the snapshot
  - No directory/prefix listing
- The statistics stored for each data file speeds up query planning
- Time-travel and incremental read

Reliability

Performance

Atlas Historical Queries

- Iceberg table – partition data filtering:
  - 15,218 splits, combined
  - 13 min (wall time) / 61.5 hr (task time) / **10 sec (planning)**
- Iceberg table – partition and min/max filtering:
  - 412 splits
  - 42 sec (wall time) / 22 min (task time) **/ 25 sec (planning)**

Hive needs to process 400k splits, planning time is **9.6 min**

Outline

- Tables Format
  - Hive Tables
- Iceberg Tables
- **Improvement Over Iceberg**
  - Buffered Writes
  - Iceberg v2
  - Nessie
- Possible Research

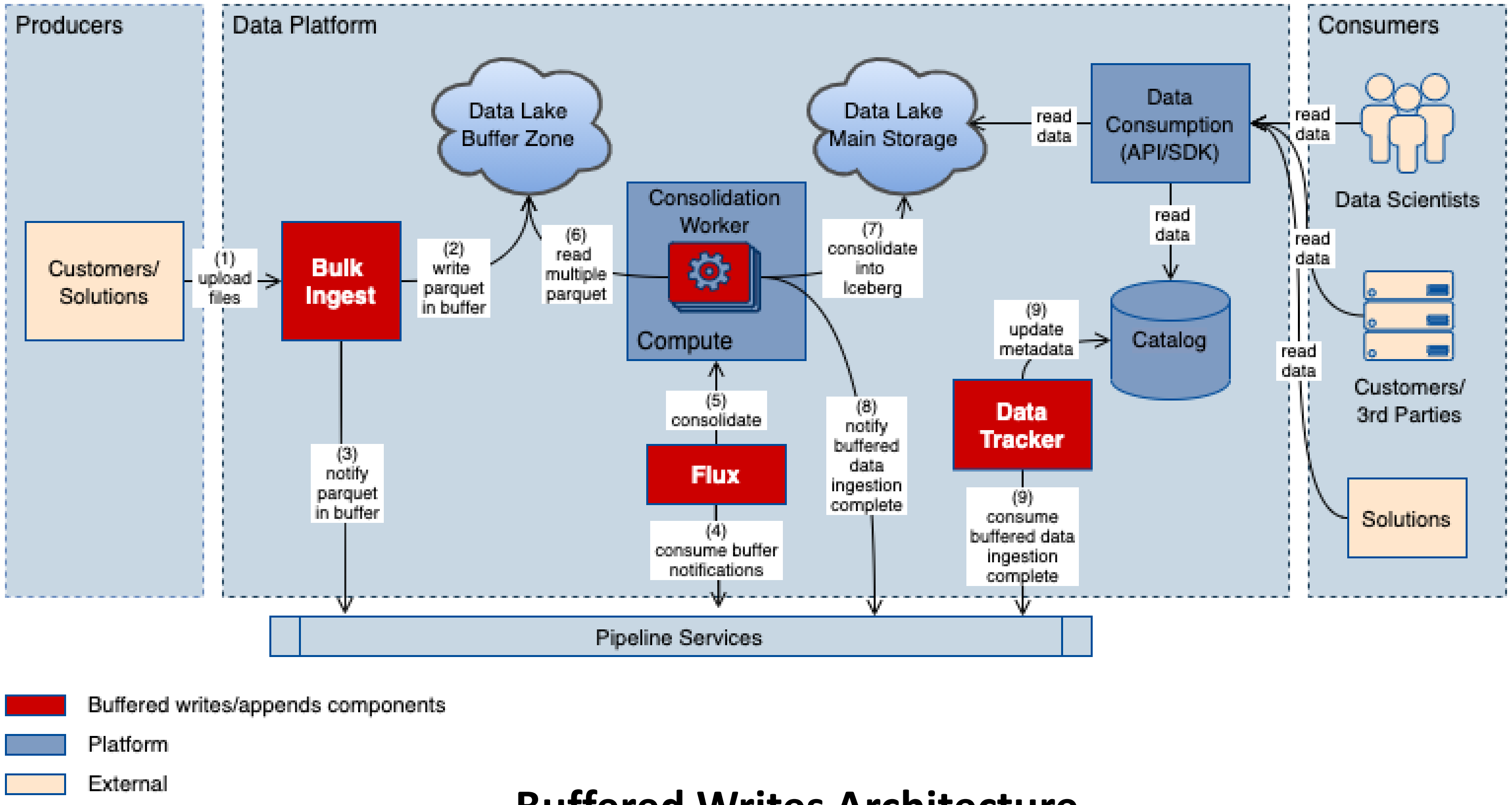
High Concurrent Writes

- Iceberg uses optimistic concurrency control to allow concurrent writes
- Iceberg automatically retries to ensure updates succeed
- Limitations:
  - accrue a queue of multiple retries
  - delay data being ingested in a timely manner
  - wasted compute
  - large write-amplification (?)
  - scan metadata files of the previous snapshot before composing a new snapshot
    - Limit throughput
    - 15 commits/minute per dataset on AEP

Adobe AEP's Buffered Write Solution

- Optimizing writes by package more writes with less files and commits
- Less files means faster scanning
- Auto-scaling – separate jobs to process and move the data
- Requires component to orchestrate the buffered writes

Buffered Writes in AEP



Buffered Writes Architecture

Ingesting ~200k small files per hour into a single Iceberg Table

Iceberg v2

- Iceberg v1: **Analytic Data Tables**
  - defines how to manage large analytic tables
  - uses immutable file formats, like Parquet, Avro, and ORC
  - data overwrite/rewrite use-cases were available
    - but they solely relies on data file-level operations

- Iceberg v2: add **Row-level Deletes**
  - aims to add support for encoding of row-level deletes
  - enables delete or replacement of rows in immutable data files
    - not rewrite the actual files

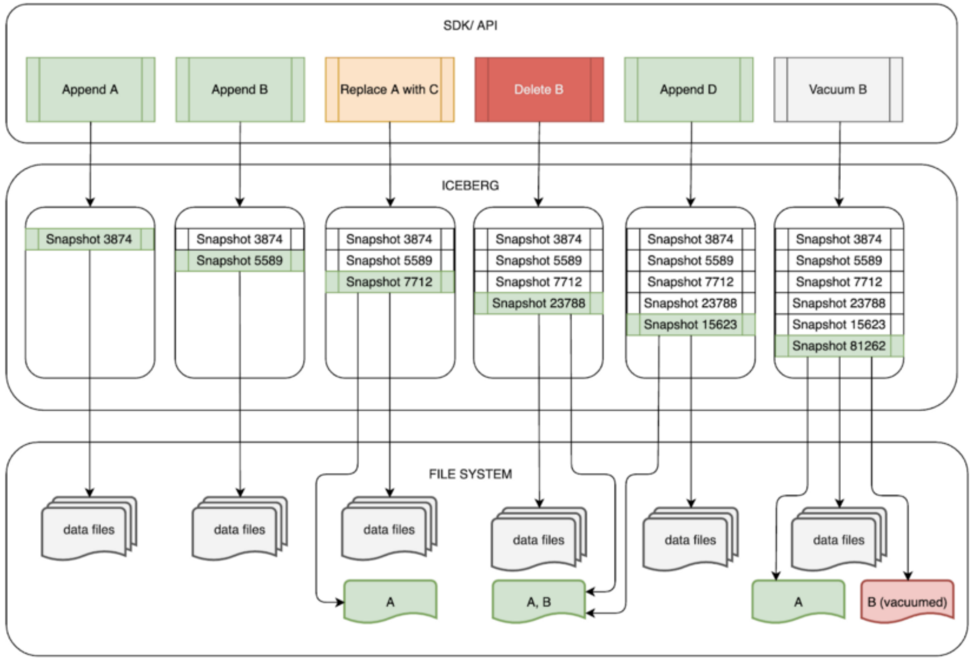
> "Support for row-level deletes will allow Iceberg to be used for several new use cases, like UPSERT, MERGE INTO, etc.
>
> The approach is to add two ways to **encode deletes in the Iceberg** spec so that engines can delete rows in existing files and append new data to replace those rows."

AEP – Data Restatement

- Data restatement is about mutating data through DML operations
    - update, delete
- Customers patching their data (i.e., fixes)
- Accommodate compliance regulations (i.e., GDPR)
- Accommodate system data operations (i.e., compaction)
    - storage optimization
    - faster access patterns

AEP – Tombstone Extension

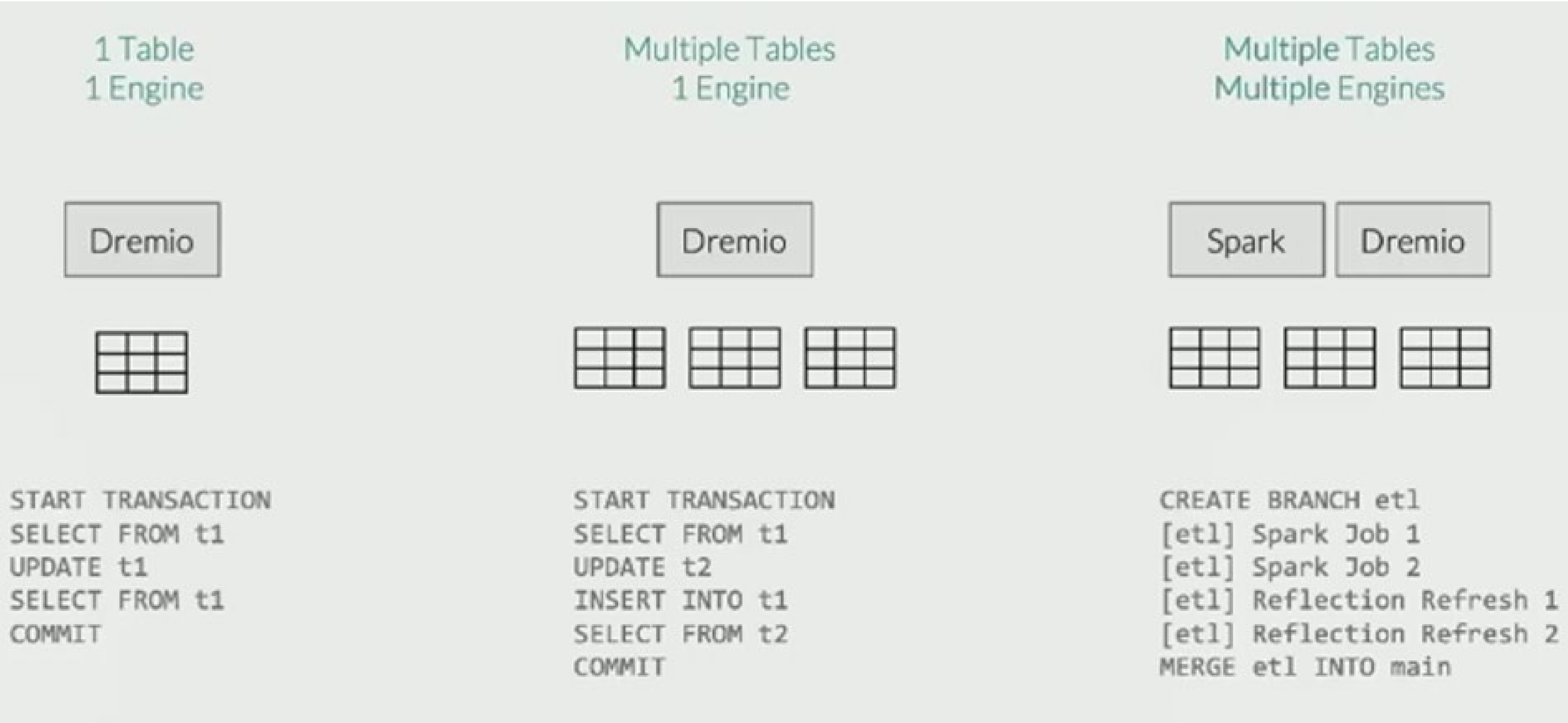
- Tombstone data is maintained in immutable files (avro)
- Snapshots continue to pass on their tombstone reference to newer snapshots along as the tombstone set isn't altered
- A new tombstones file is created when a change of tombstones happens

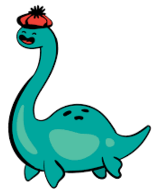


AEP – Data Restatement

- Tombstone data is maintained in immutable files (avro)
- Snapshots continue to pass on their tombstone reference to newer snapshots along as the tombstone set isn’t altered
- A new tombstones file is created when a change of tombstones happens

Nessie – A Git-like Experience for data lake



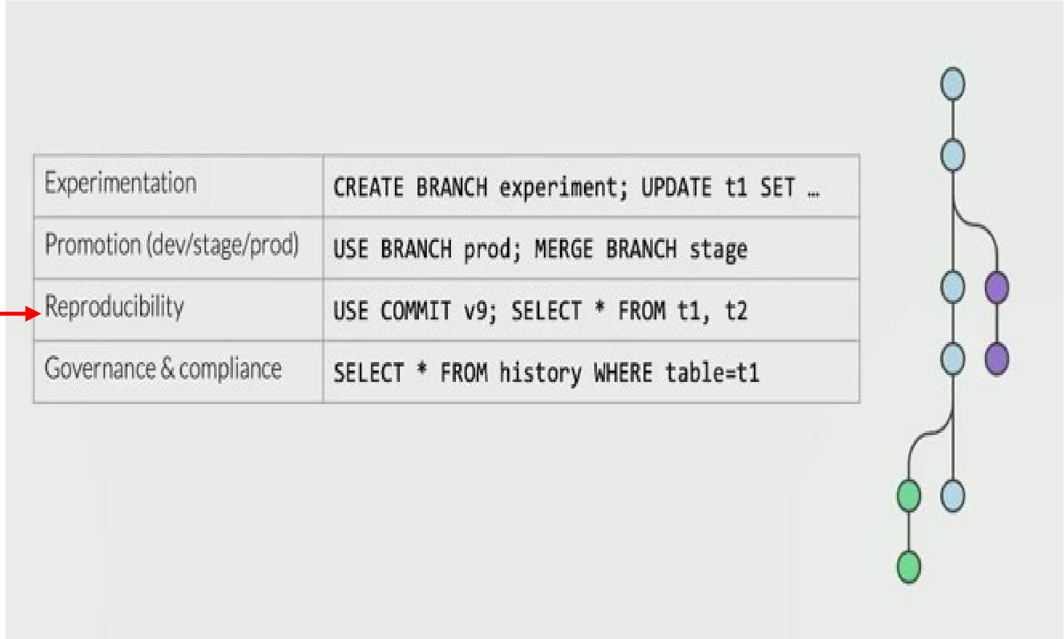


Enable multi-table and multi-engine transactions

Git-like Version Control w/ Commit, Branch, Tag

- Commits
  - Atomic changes to a set of files
- Branch
  - Enable cross-table transactions
  - Group a set of changes
- Tag
  - Named reference to a commit
- Merge
  - applies the changes of one branch onto another

MLOps



Experimentation	CREATE BRANCH experiment; UPDATE t1 SET …
Promotion (dev/stage/prod)	USE BRANCH prod; MERGE BRANCH stage
Reproducibility	USE COMMIT v9; SELECT * FROM t1, t2
Governance & compliance	SELECT * FROM history WHERE table=t1

Four scenarios where Nessie is wanted

Scale & Performance

- Built for very large data warehouses
  - several magnitudes larger than the largest in the world today
  - supports millions of tables
- Built as a cloud native technology
  - designed to be highly scalable, performant and resilient
- Aim for thousands of commits/second
  - 400-500 commits per second currently
- Highly scalable
  - separation of txn mgmt from table metadata mgnt

Outline

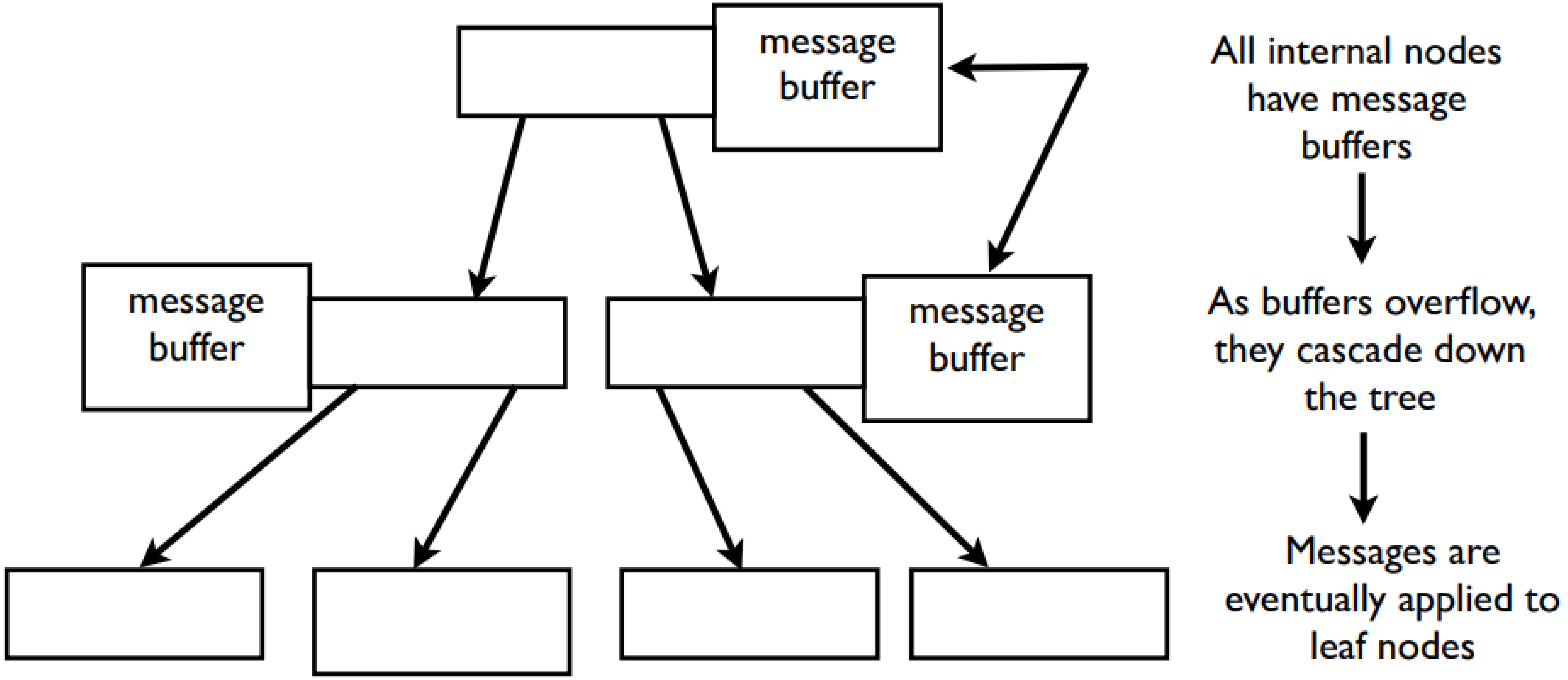
- Tables Format
  - Hive Tables
- Iceberg Tables
- Improvement Over Iceberg
  - Buffered Writes
  - Iceberg v2
  - Nessie
- **Possible Direction**

Three Key Techniques

- Coalescing small writes into larger ones
- Using messages for updates and deletions
- Versioning
    - Space Efficient
    - Fast
- All three techniques can be realized by one data structure
    - B^{ε} -DAG, which has been proven very efficient for creating file system snapshots

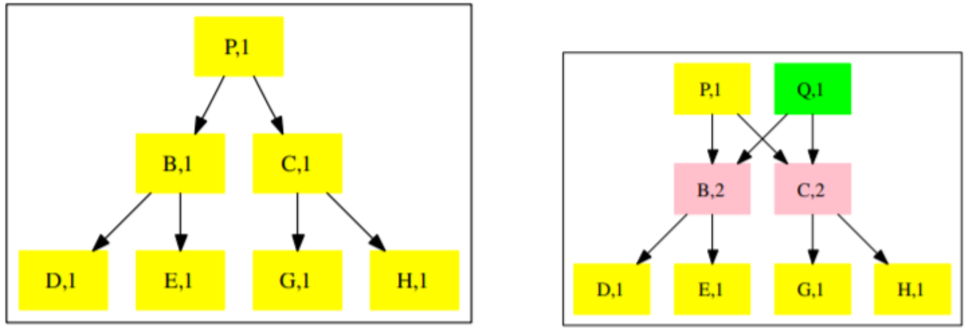
B^{ε} –DAG (1)

- **B^{ε} –tree** + clone operation

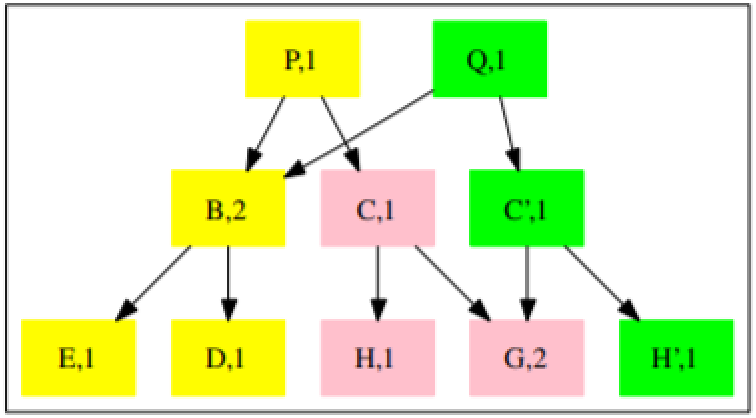


B^{ε} –DAG (2)

- B^{ε} –tree + **clone operation**
- Avoid immediate copy-on-write to reduce update latency
  - Buffer mutations in the inner node
  - Reduce write amplification
- Benefits
  - Fast snapshot creation/update
  - Maximize space sharing



(a) Create a snapshot



(b) Copy-on-write for update

Possible Direction

- The metastore can implement a B^{ε} –DAG
    - Buffered writes
    - Snapshots (Nessie)
    - Fast update and deletion
        - Important for real-time processing
- A lot of unsettled questions
    - How to implement this for a Bw-tree
    - How to work with S3 storage
    - What is the table format
    - How to scale out
        - Nessie is using MongoDB (NOSQL)
    - How to cache

References

[1] https://www.qubole.com/blog/optimizing-s3-bulk-listings-for-performant-hive-queries/

[2] Iceberg--A modern table format for big data.

[3] https://lakefs.io/hive-metastore-why-its-still-here-and-what-can-replace-it/

[4] https://www.simplilearn.com/tutorials/hadoop-tutorial/data-file-partitioning

[5] https://medium.com/expedia-group-tech/a-short-introduction-to-apache-iceberg-d34f628b6799

[6] https://www.qubole.com/blog/optimizing-hadoop-for-s3-part-1/

[7] https://medium.com/plumbersofdatascience/hive-architecture-in-depth-ba44e8946cbc

[8] https://issues.apache.org/jira/browse/HIVE-20517

[9] any plan for Iceberg Table on S3? https://github.com/apache/iceberg/issues/1468

[10] Iceberg at Adobe. https://experienceleaguecommunities.adobe.com/t5/adobe-experience-platform-blogs/iceberg-at-adobe/ba-p/393615

[11] High Throughput Ingestion with Iceberg. https://medium.com/adobetech/high-throughput-ingestion-with-iceberg-ccf7877a413f

[12] Row level deletion. https://github.com/apache/iceberg/milestone/4

[13] Iceberg Series: ACID Transactions at Scale on the Data Lake in Adobe Experience Platform. https://medium.com/adobetech/iceberg-series-acid-transactions-at-scale-on-the-data-lake-in-adobe-experience-platform-f3e8fe0cef01

[14] B-trees, Shadowing, and Clones. https://www.usenix.org/legacy/events/lsf07/tech/rodeh.pdf

[15] On Supporting Efficient Snapshot Isolation for Hybrid Workloads with Multi-Versioned Indexes. https://www.vldb.org/pvldb/vol13/p211-sun.pdf