ICEBERG REPLICATION

Rahul Buddhisagar, Shailesh Shiwalkar, Teddy Choi

CLOUDERA

What is APACHE ICEBERG ?

- Bootstrapped in Netflix and then contributed to the open source community
- The open table format for analytics datasets
- Works with several compute engines; Spark, Trino, PrestoDB, Flink, Hive, and Impala

Why Replication?

- Ensure business continuity
- Identifying critical data
- Evaluate & Tune

Why not just copy?

Copying storage isn't sufficient.
Needs assembly.

- **Schema evolution**
- **Hidden partitioning**
- **Time travel**

What is SCHEMA EVOLUTION?

Challenge: Update a table schema

Format

Folder-level operations

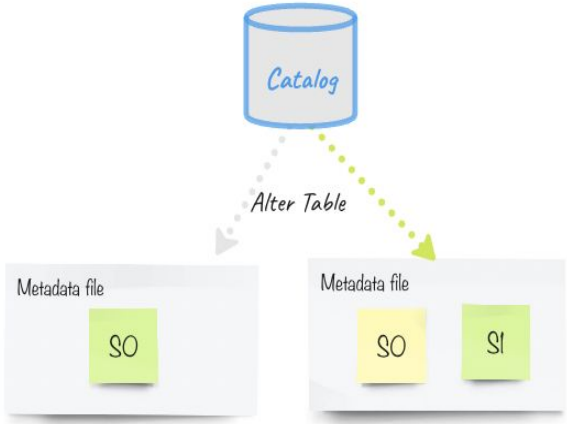
- Resource intensive
- Slow and inefficient

File-level

- Track all files, metadata and data
- Atomic update to new metadata file
  - lockless readers
  - isolated writers

- Metastore/Catalog no longer a bottleneck

```
sql> ALTER TABLE BayAreaProperties
ADD COLUMNS (ZipCode varchar(10));
```



What is PARTITIONING?

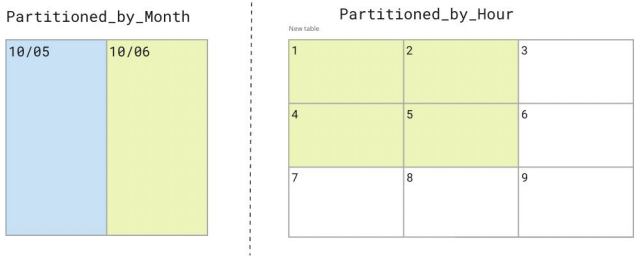
Challenge: Maintaining partitions is a lot of work

Hidden Partitioning

- No need for specific partition filter
- Several partitioning options
- Agnostic to a physical layout

```
sql>INSERT INTO SaleTransactions PARTITION
(event_date) SELECT merchant, memo,
event_time, format_time(event_time,
'YYYY-MM-dd') FROM unstructured_log_source;
```

```
sql>Select * from SaleTransactions where
date > 10-01-2020 AND date < 10-04-2021
```



Evolution

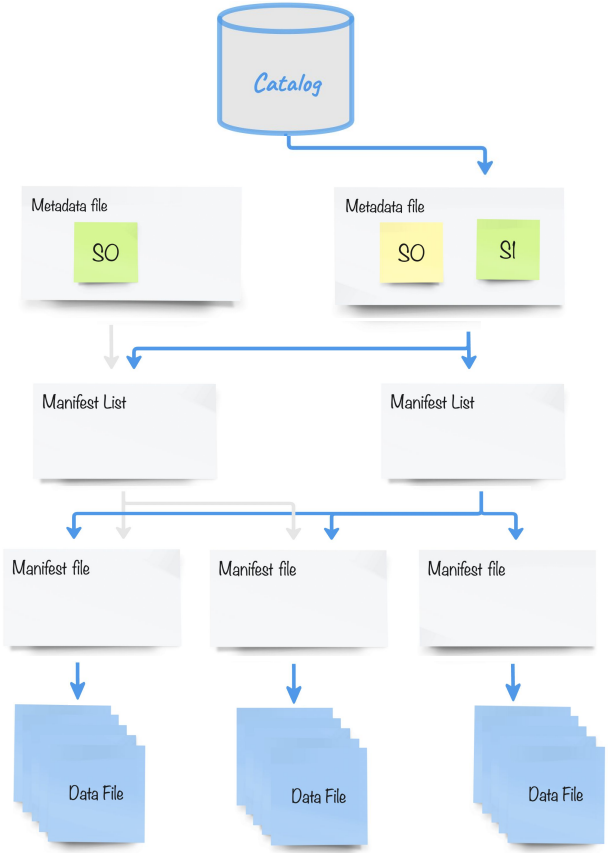
- Operation at the metadata level
- Split Planning

What is TIME TRAVEL?

Challenge: Wish we could go back in time to fix a mistake

Structure

- Metadata Files
  - Schema & partition information, snapshot details
- Manifest List File
  - Information about all Manifest files
  - Groups all the manifest files part of the snapshot
- Manifest file : List of data files
- Data files : map the data on the actual disk



TIME TRAVEL

Applications

- Time based OR snapshot ID based
- Rollback
- Projections
- Experiment

```
sql>SELECT * FROM db.emea_region.snapshots;

+-------------------------+---------------------+---------------------+---------------------+
| made_current_at         | snapshot_id         | parent_id           | is_current_ancestor |
+-------------------------+---------------------+---------------------+---------------------+
| 2019-02-08 03:29:51.215 | 5781947118336215154 | NULL                | true                |
| 2019-02-08 03:47:55.948 | 5179299526185056830 | 5781947118336215154 | true                |
| 2019-02-09 16:24:30.13  | 296410040247533544  | 5179299526185056830 | false               |
| 2019-02-09 16:32:47.336 | 2999875608062437330 | 5179299526185056830 | true                |
| 2019-02-09 19:42:03.919 | 8924558786060583479 | 2999875608062437330 | true                |
| 2019-02-09 19:49:16.343 | 6536733823181975045 | 8924558786060583479 | true                |
+-------------------------+---------------------+---------------------+---------------------+
```

```
sql> SELECT revenue FROM emea_region  FOR SYSTEM_VERSION AS OF <SNAPSHOT_ID>;
sql> SELECT revenue FROM apac_region  FOR SYSTEM_TIME AS OF <TIMESTAMP>;
```


TIMETRAVEL!

REPLICATION MANAGEMENT SYSTEM

Management Plane, Control Plane and Data Plane 101

Management Plane

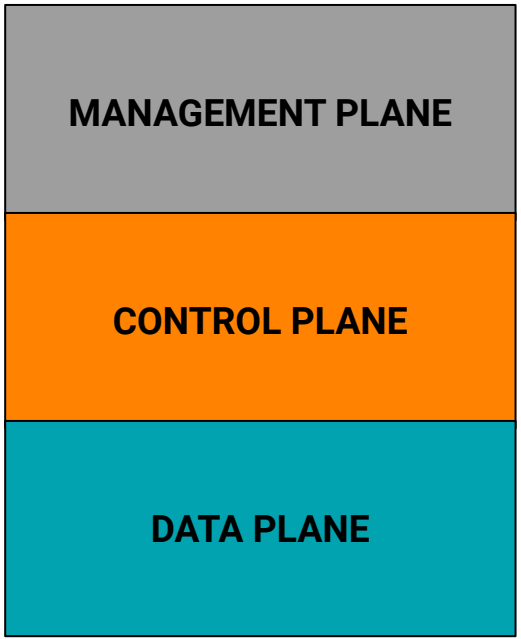
- The management plane decides how replication policies are managed, monitored and how to provide useful metrics.

Control Plane

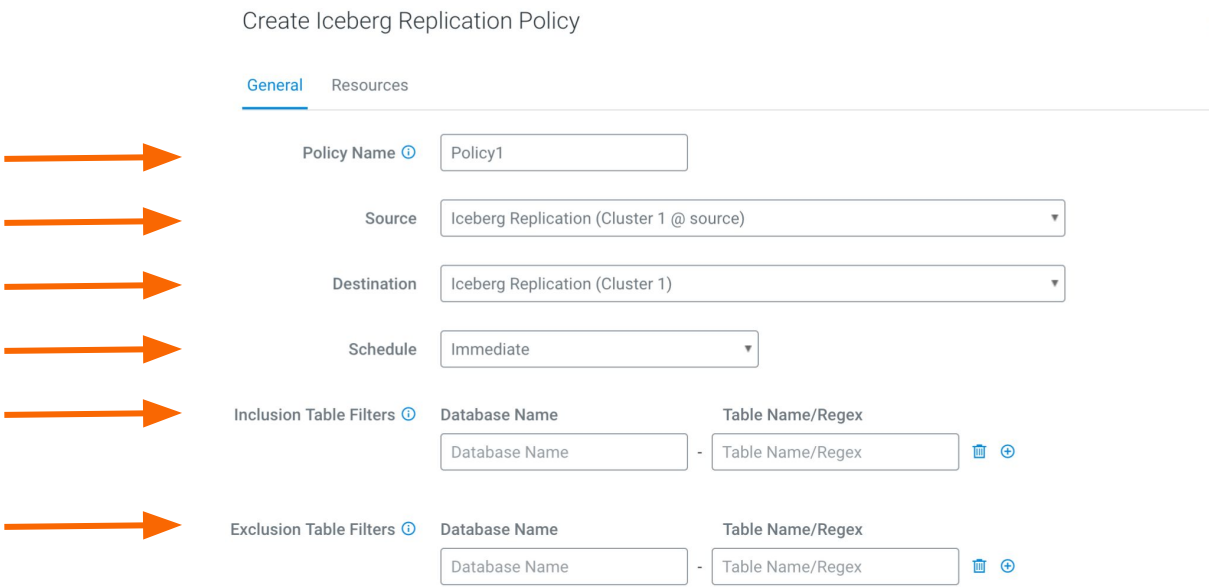
- The control plane is responsible for replication policy execution between the source and the destination.

Data Plane

- The data plane is responsible for the actual movement of data and how it scales as the data set scales.



Iceberg Replication Policy Definition



REPLICATION FLOW IN A NUTSHELL

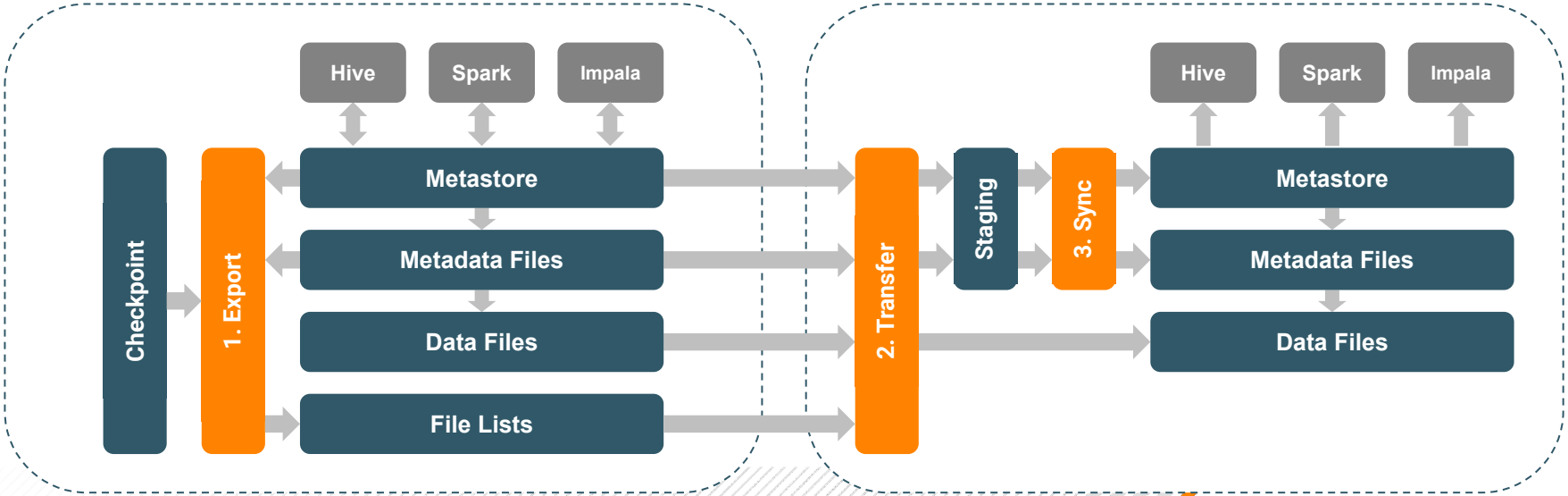
Export, transfer, then synchronize

SOURCE

Plan the replication by comparing versions

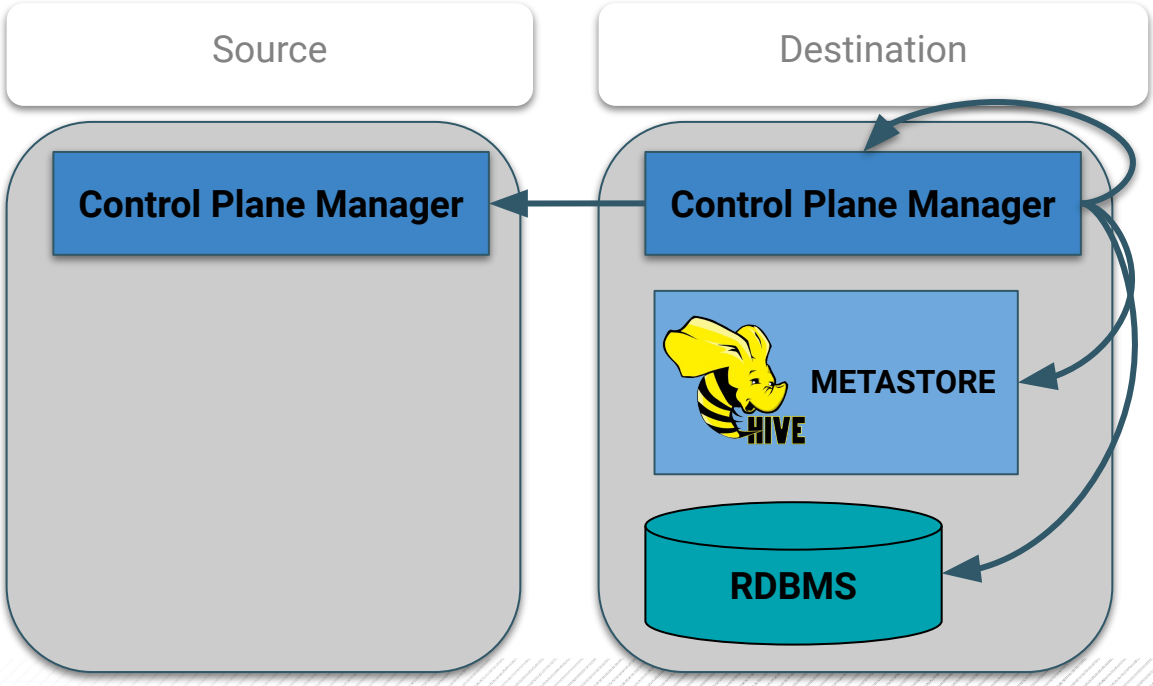
DESTINATION

Copies, transforms files, then update the metastore



Executing a Replication Policy

Steps involved in executing an Iceberg Replication Policy

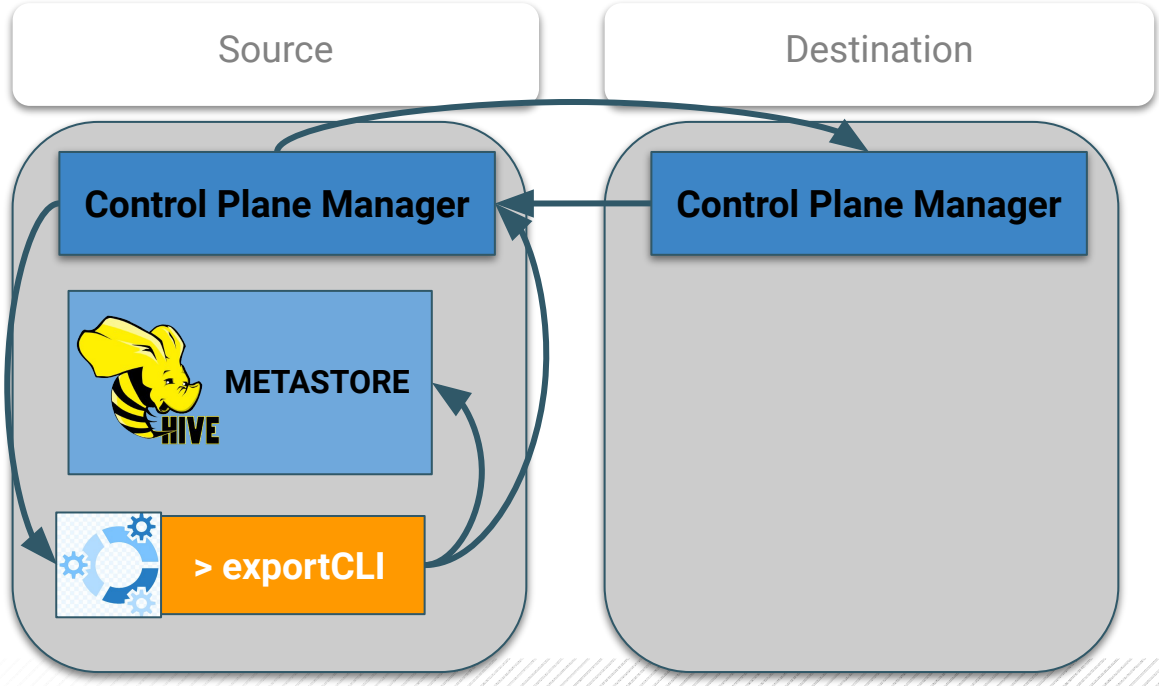


Export Steps

1. Reads the Policy Definition from the RDBMS
2. Evaluates the table filters
3. Get the latest status for each table (checkpoint file)
4. Provides the Table list and checkpoint file to the Source

Executing a Replication Policy

Steps involved in executing an Iceberg Replication Policy



Export Steps

1. Launch exportCLI, args Table list, checkpoint file

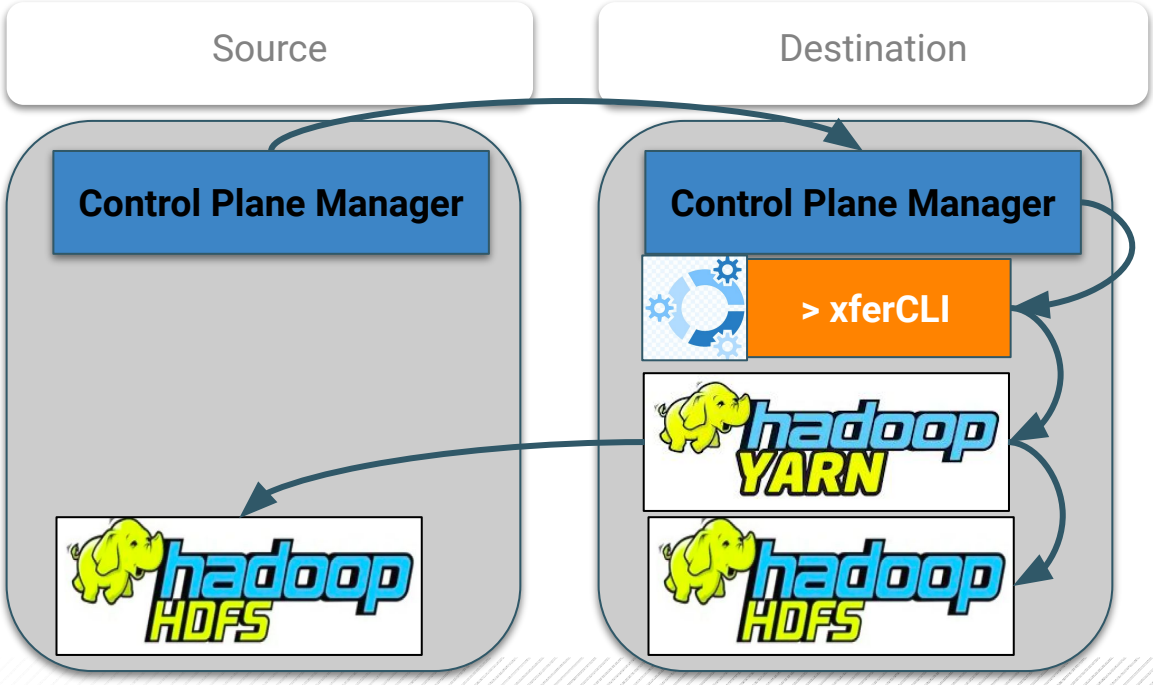
2. Reads table definition from the Catalog (Hive metastore)

3. Generates result, export.json

4. Provides result file export.json to the Destination

Executing a Replication Policy

Steps involved in executing an Iceberg Replication Policy



Xfer Steps

1. Launch xferCLI, args export.json

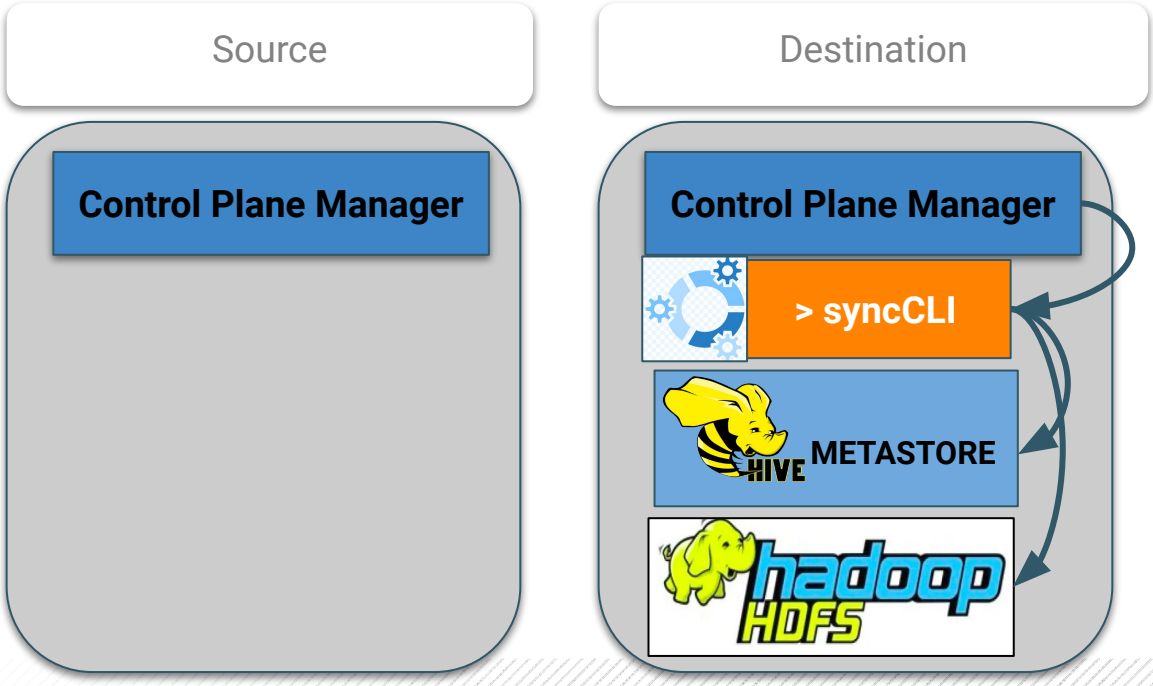
2. Launches multiple Distributed Copy (distcp) jobs

3. Reads data and metadata files from source

4. Writes data files to final path, writes metadata files to staging path

Executing a Replication Policy

Steps involved in executing an Iceberg Replication Policy



Sync Steps

1. Launch syncCLI

2. Updates metadata files and copies to the final path

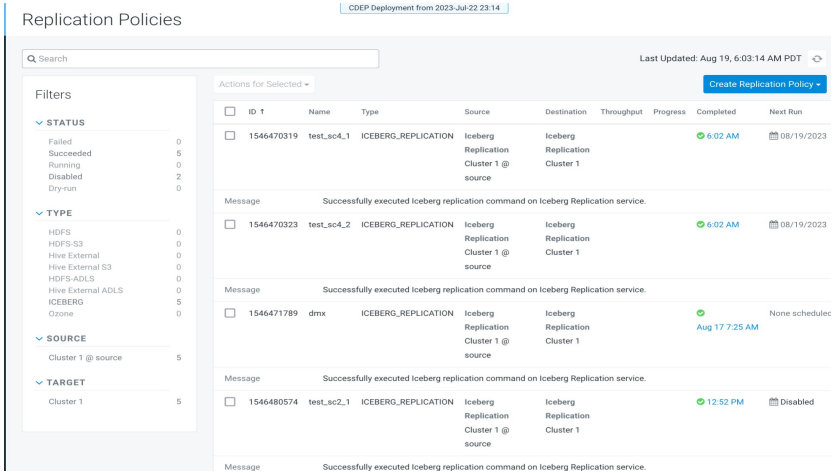
3. For each table, updates Catalog (HMS) as a last step

MANAGEMENT PLANE

A unified plane for the policy management, monitoring, replication history and metrics

EXECUTION MONITORING

Check replication status, historical runs



Monitoring

- Current Execution Status

-- export xfer sync done

- Result: Success or Failure
- Runtime of each step

Centralized monitoring for all policies

- History of previous runs
- Spot the outlier

MANAGEMENT PLANE

A unified plane for the policy management, monitoring, replication history and metrics

Replication Policies

Replication History

Name Victory Type ICEBERG Source iceberg_replication (Cluster 1 @ Source) Destination Iceberg Replication (Cluster 1) Next Run June 19, 2023 10:17 AM

Start Time	Duration	Outcome	Number of tables Processed	Number of files copied	Number of files deleted	Number of manifests transformed
June 18, 2023 10:22 AM	2 min	Successful	0	5	0	0
June 18, 2023 10:19 AM	2 min	Successful	0	5	0	0
June 18, 2023 10:17 AM	1 min	Successful	0	5	0	0

1 - 3 of 3

Policy Stats

- Data, Metadata file replicated
- Manifest files transformed

Troubleshooting & Diagnostics

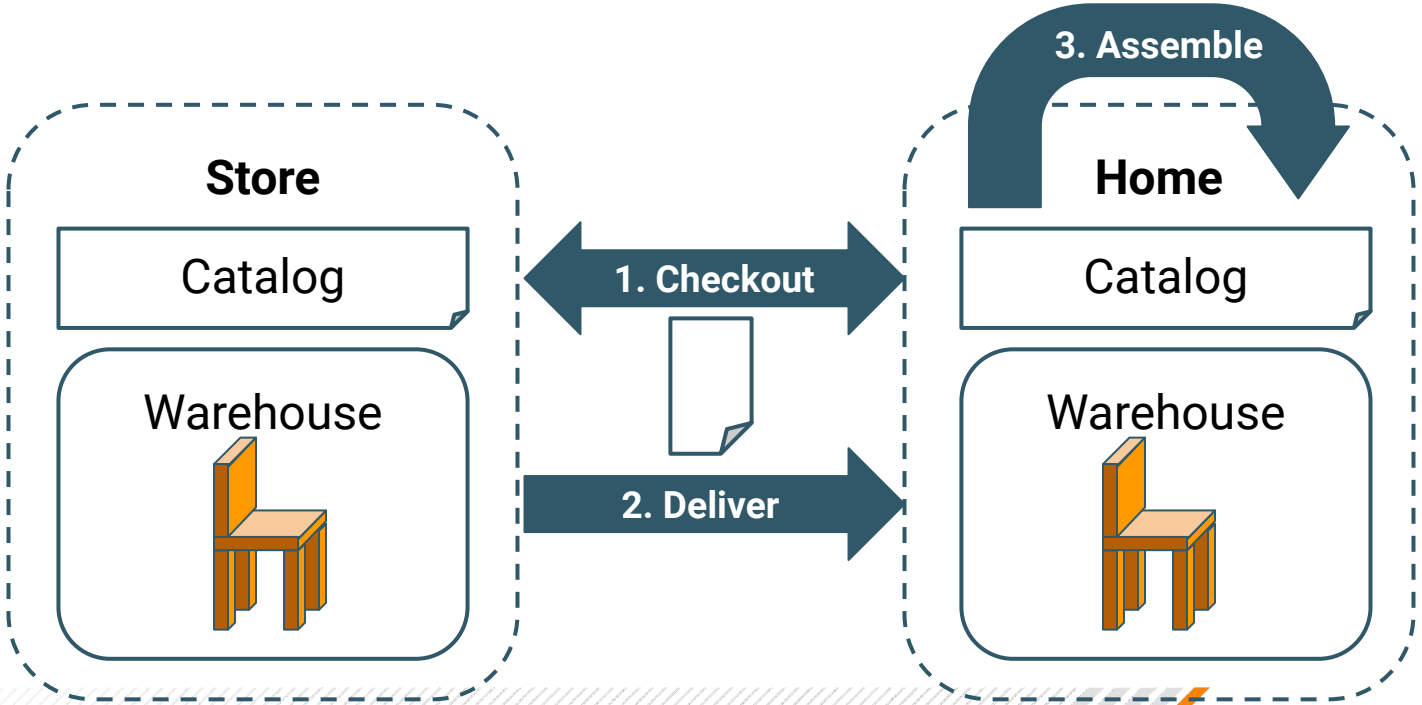
- Collect Diagnostics from Source
- Collect Diagnostics from Destination

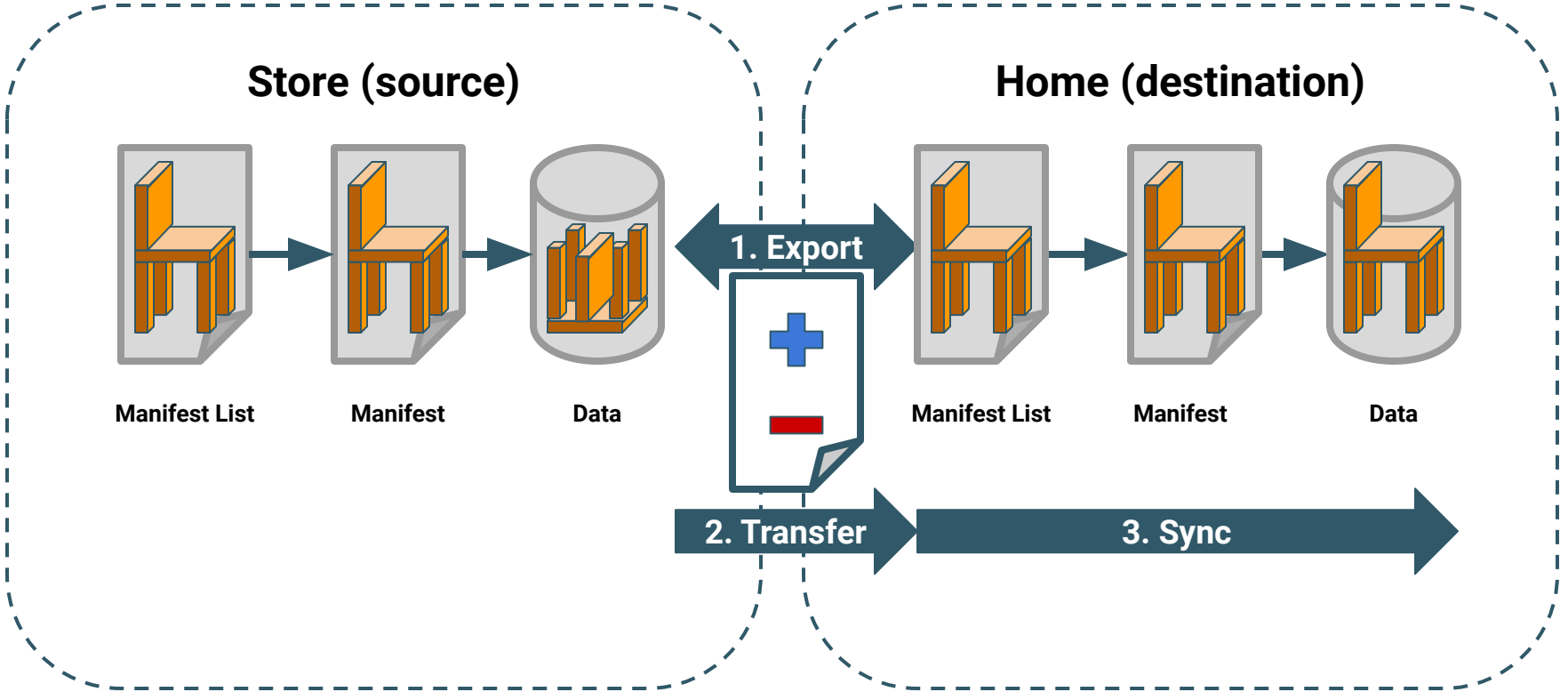
DATA PLANE

Teddy Choi

ICEBERG REPLICATION AS FURNITURE SHOPPING

Checkout, deliver, and assemble the tables




Store (source)
Home (destination)
1. Export
Manifest List
Manifest
Data
Manifest List
Manifest
Data
2. Transfer
3. Sync

REPLICATION PROCESS

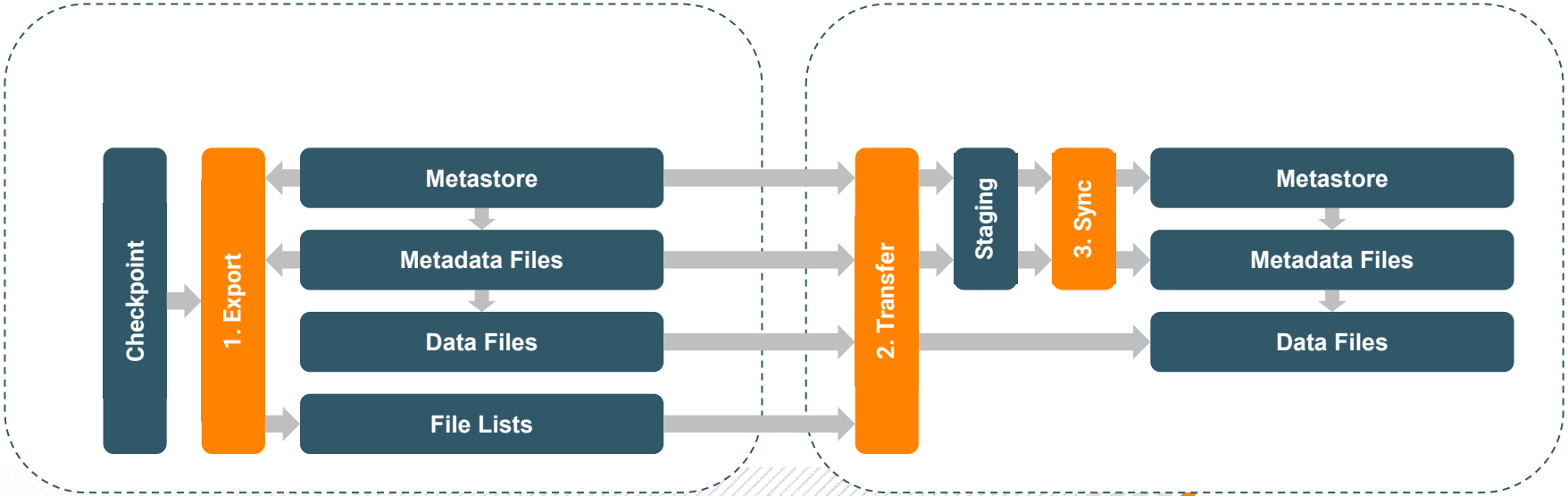
Export, transfer, then synchronize

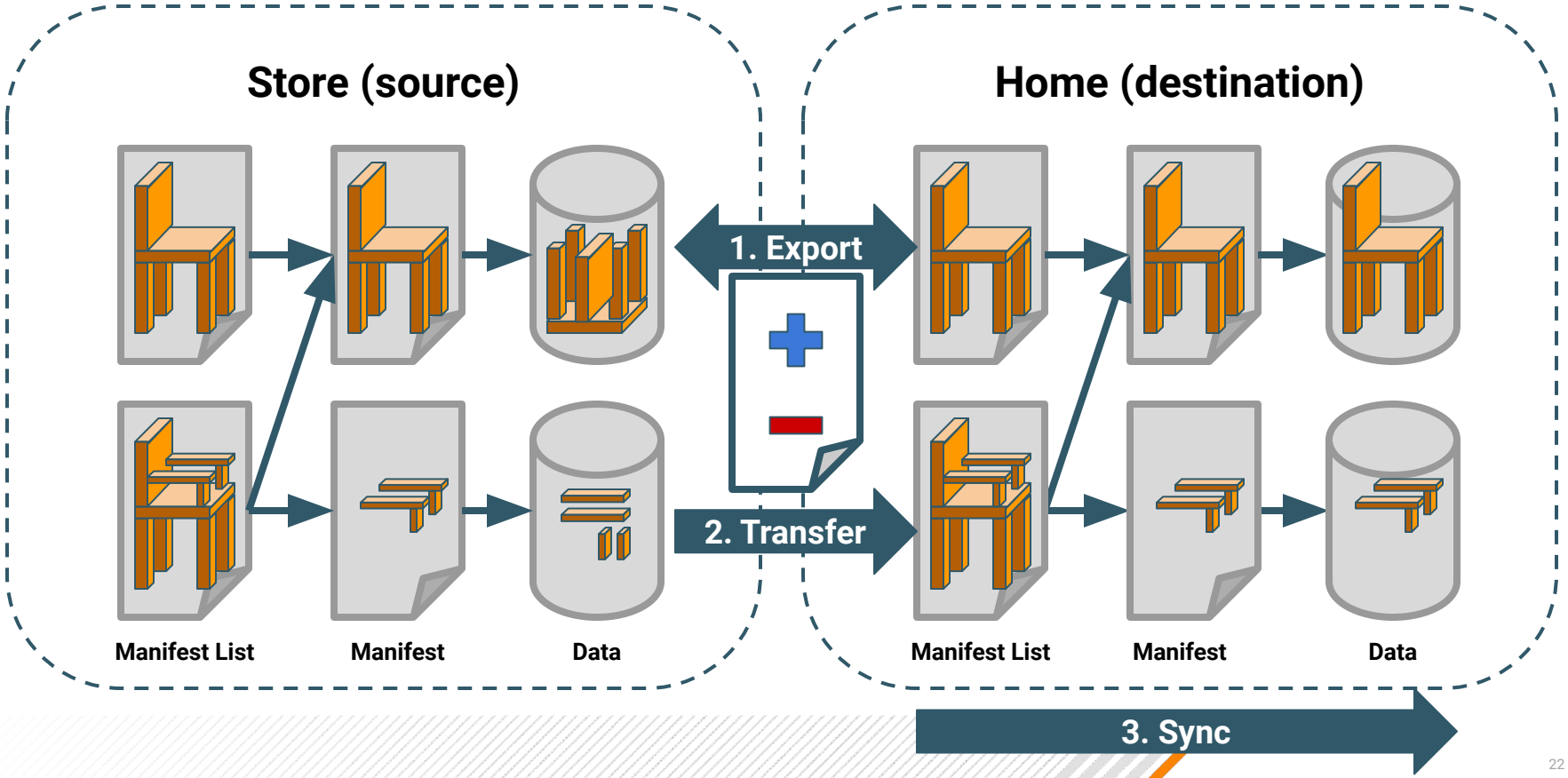
SOURCE

Plan the replication by comparing versions

DESTINATION

Copies, transforms files, then update the metastore




Store (source)
Home (destination)
1. Export
2. Transfer
3. Sync
Manifest List
Manifest
Data
Manifest List
Manifest
Data

REPLICATION PROCESS

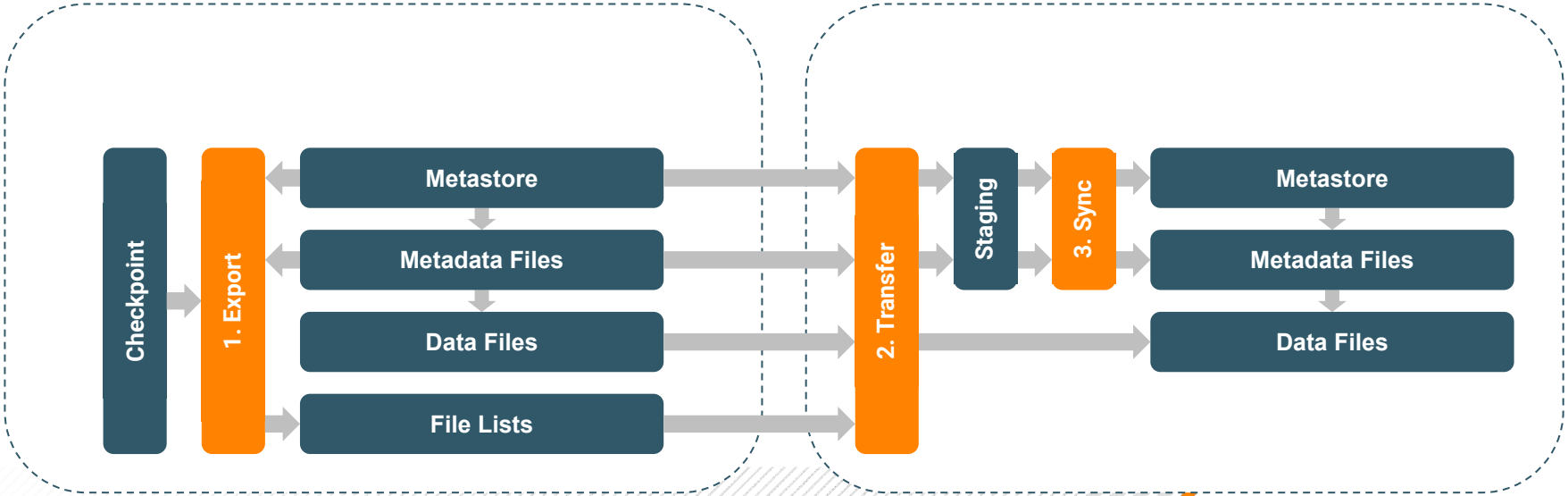
Export, transfer, then synchronize

SOURCE

Plan the replication by comparing versions

DESTINATION

Copies, transforms files, then update the metastore



Q: WHY DON'T YOU SIMPLY...

A: Because of limitations of file systems that we should support

List files recursively

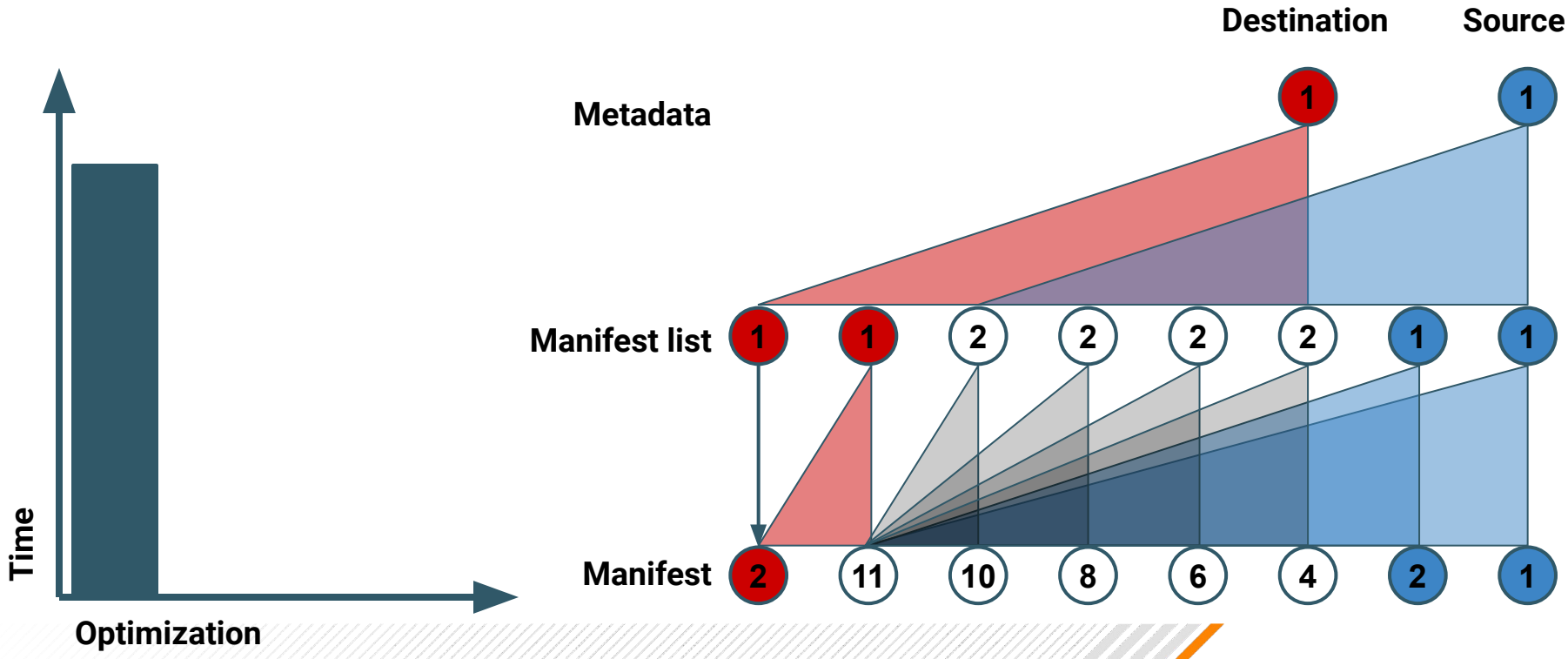
- **Simple** and fast on HDFS, Ozone
- **Slow and expensive on S3**
- Files may **not follow dir layout** (migrated external tables)

Read all metadata files

- **Simple** and fast for small tables
- Can grow to millions
- **HDFS namenode bottleneck**

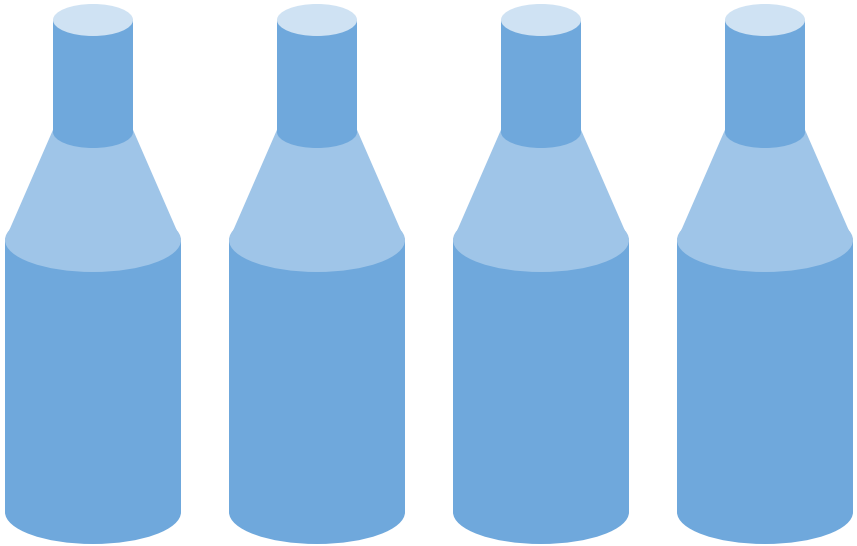
DAYS: DEPTH-FIRST-SEARCH

O(snapshots^2) = O(6^2) = 58 file reads, time: 58



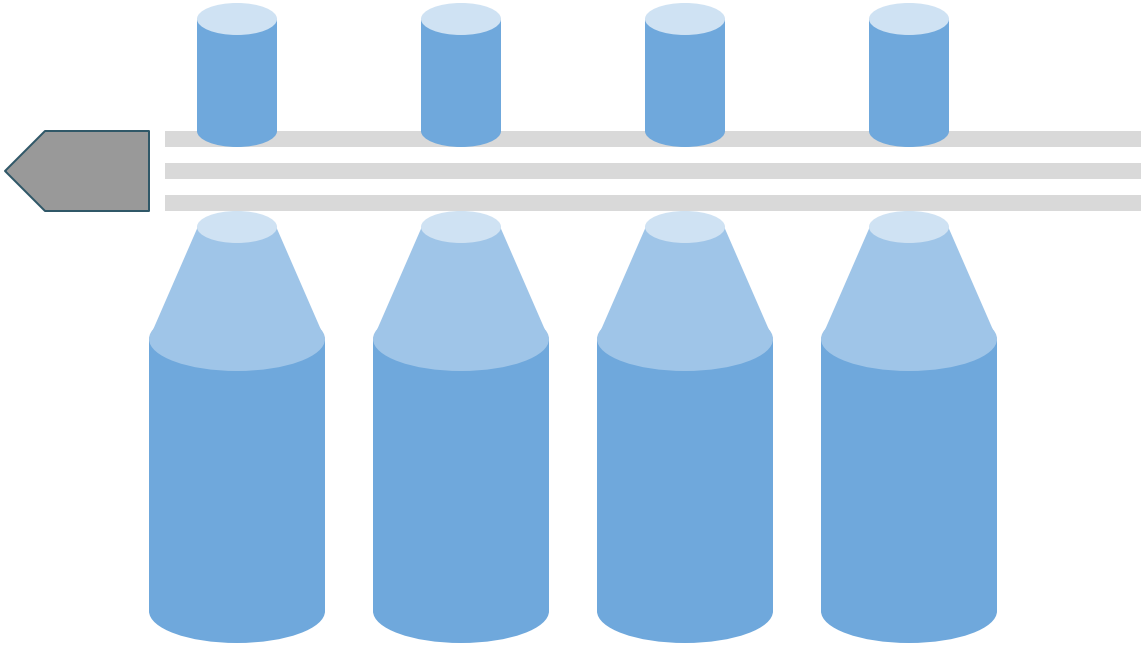
BOTTLENECKS

There can be a lot of hidden bottlenecks



LET’S BREAK BOTTLENECKS!

And I’m good at breaking them!

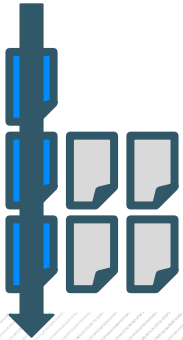


BOTTLENECK 1: REPEATED READS

It's slow because of repeated reads on same files, which can be deduplicated

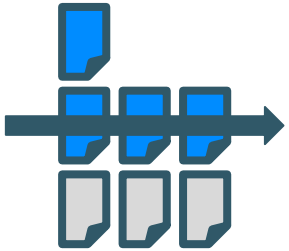
Depth-first-search

- Simple and **fast for tree traversal**
- It **causes repeated reads in graphs** that allows overlapping subgraphs



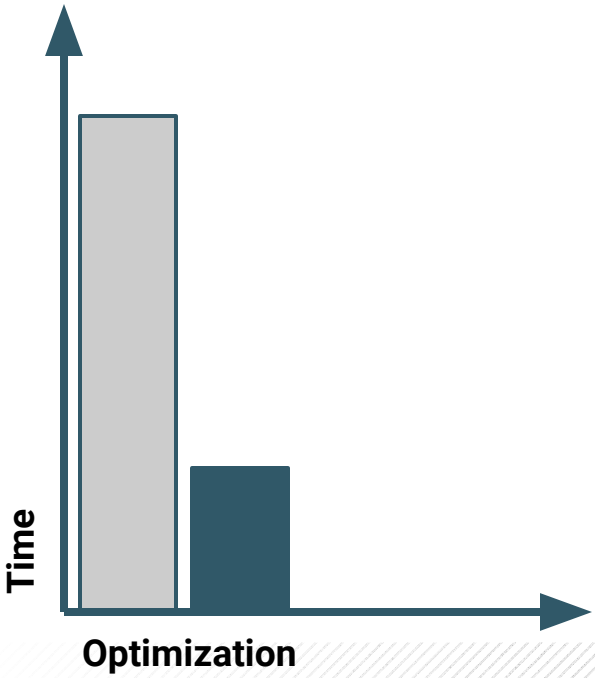
BFS with deduplication

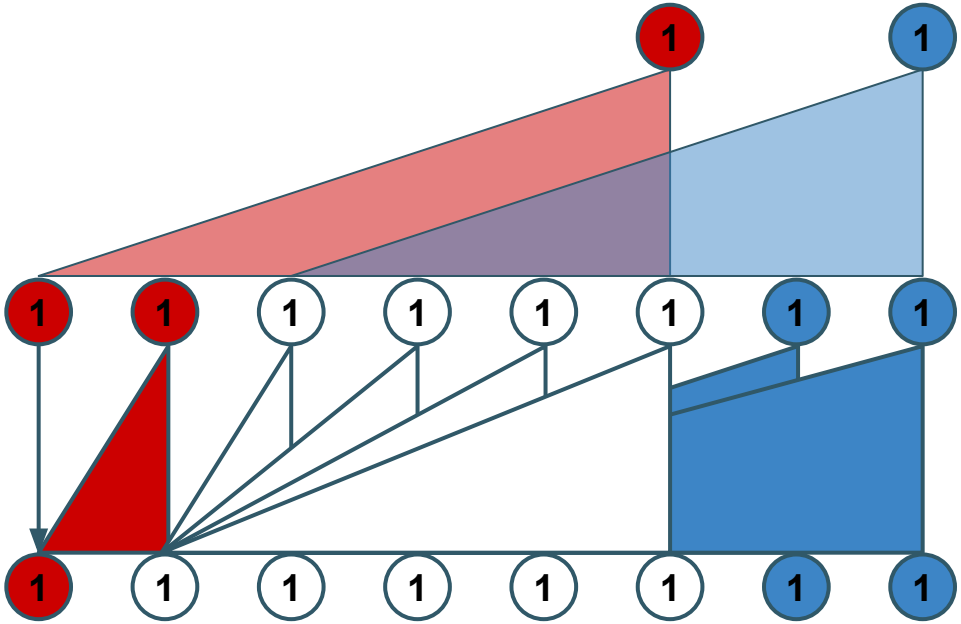
- BFS enables deduplication
- A dedup **takes O(1)** with a hash table
- For millions of files, it takes **GBs of memory**



HOURS: BREADTH-FIRST-SEARCH WITH DEDUP

O(snapshots) = O(6) = 18 file reads



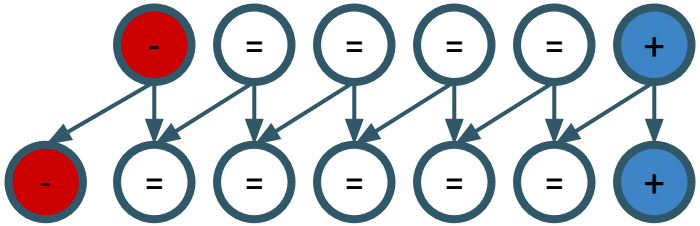


BOTTLENECK 2: UNNECESSARY READS

It's slow because it reads all files even when a typical replication has a fraction of data

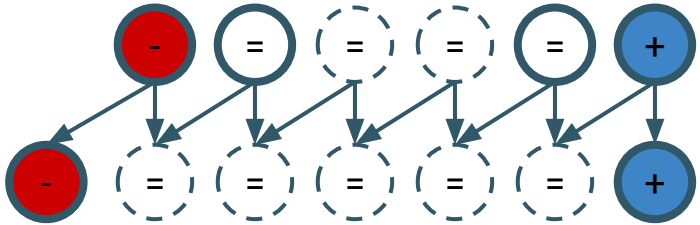
Common snapshots

- Should be **skipped for efficiency**
- Iceberg **subgraphs can overlap**
- Some changes are **referenced by common subgraphs**



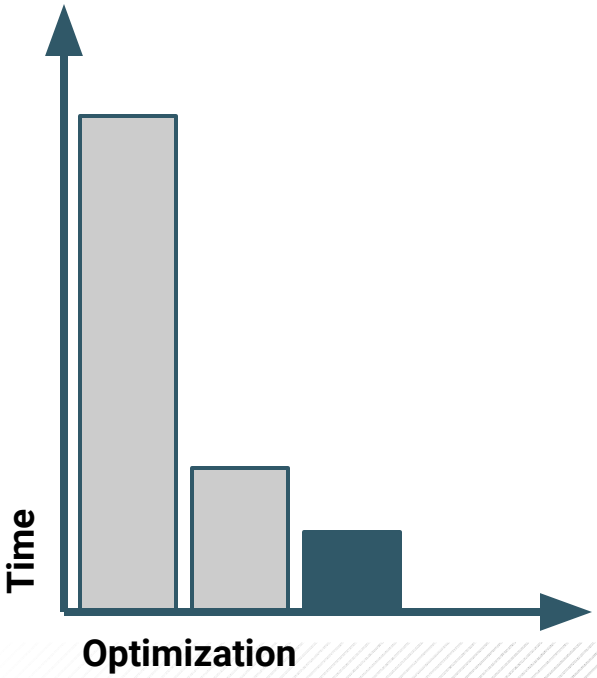
Optimistic concurrency + UUID

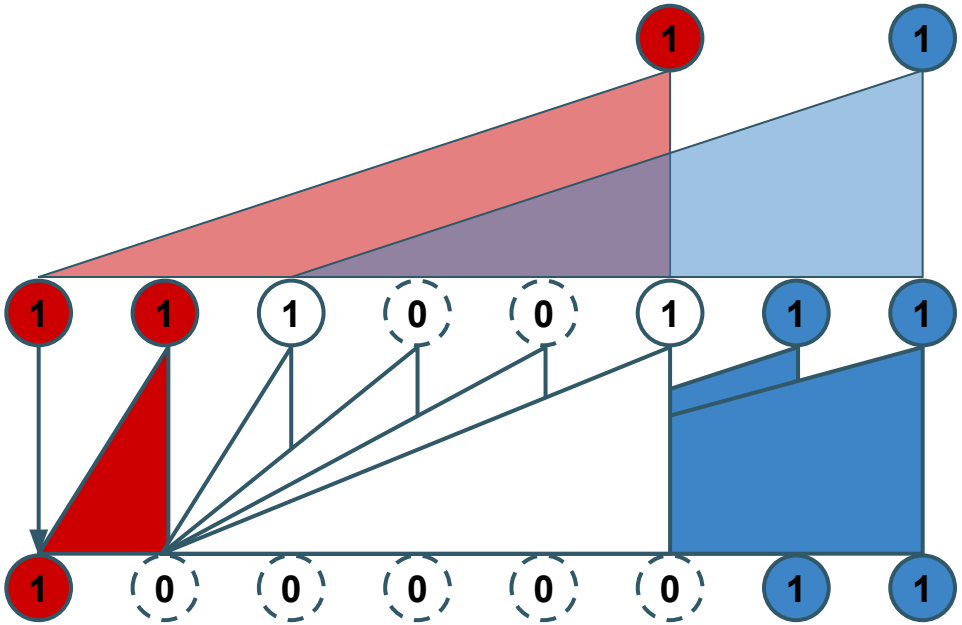
- *"the failed writer retries by writing a new metadata tree based on the new current table state"*
- We **just need to additionally read the start and end** of common snapshots



MINUTES: SKIPPING BREADTH-FIRST-SEARCH

11 file reads



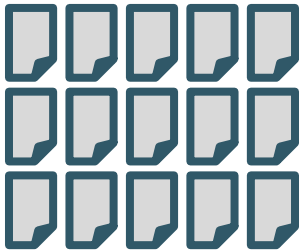


BOTTLENECK 3: TOO MANY SMALL FILES

It's slow because of too many small files syndrome on HDFS namenode

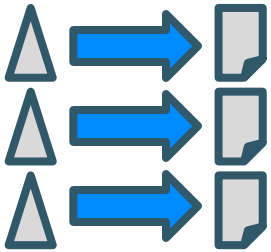
Too many small files syndrome

- Small files still takes **a couple of seconds** for namenode, datanode connection establishment



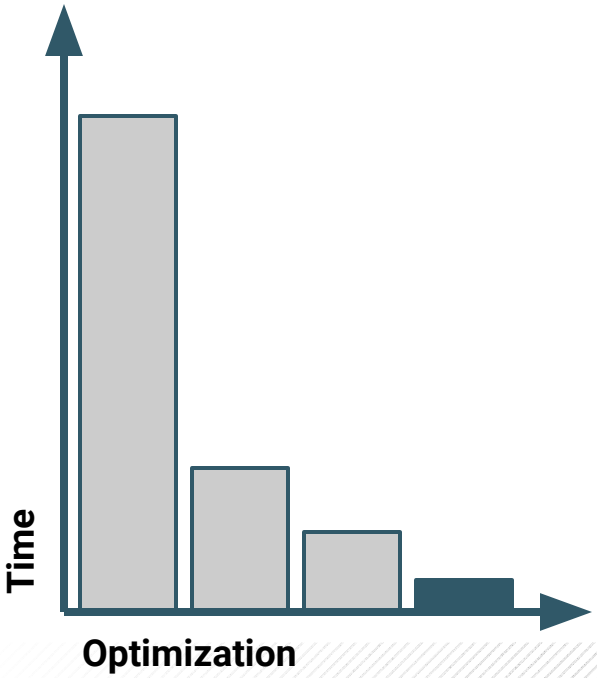
Multi-threaded

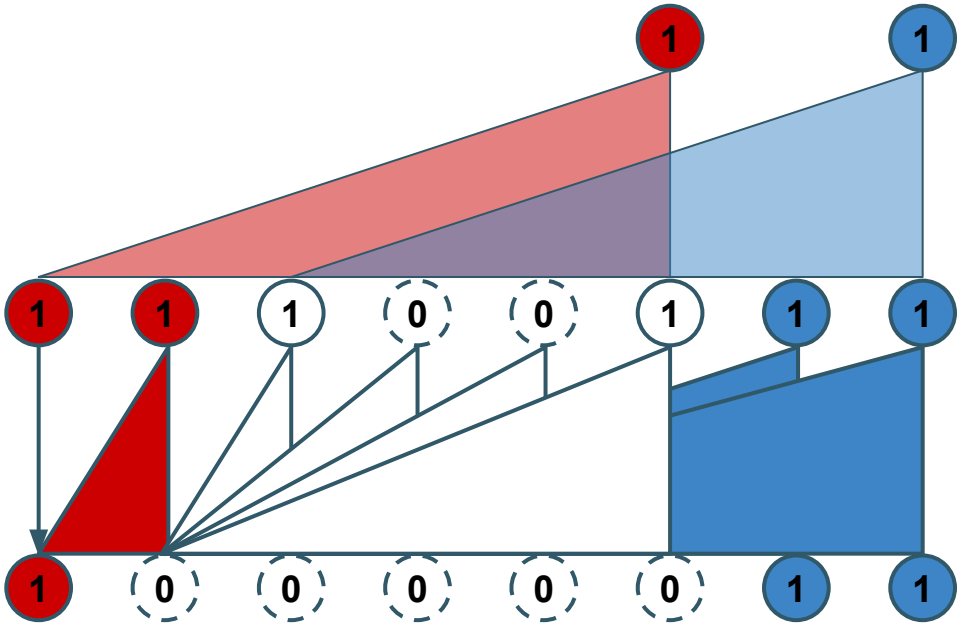
- Tables are **independent from each other**
- Easy to make it **multi-threaded per table**



SECONDS: MULTI-THREADED SKIPPING BFS

11 file reads, 4 threads, time: 3





BOTTLENECK 4: FILE READS

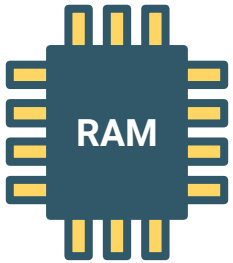
It's slower than file listing because it reads metadata files, which can be cached

Still need to read many files

- **HDFS or Ozone file listing is faster** when all files follow the directory layout

Iceberg file cache

- All Iceberg files are **immutable**
- Metadata files are **small**
- **Cached to reduce file reads**



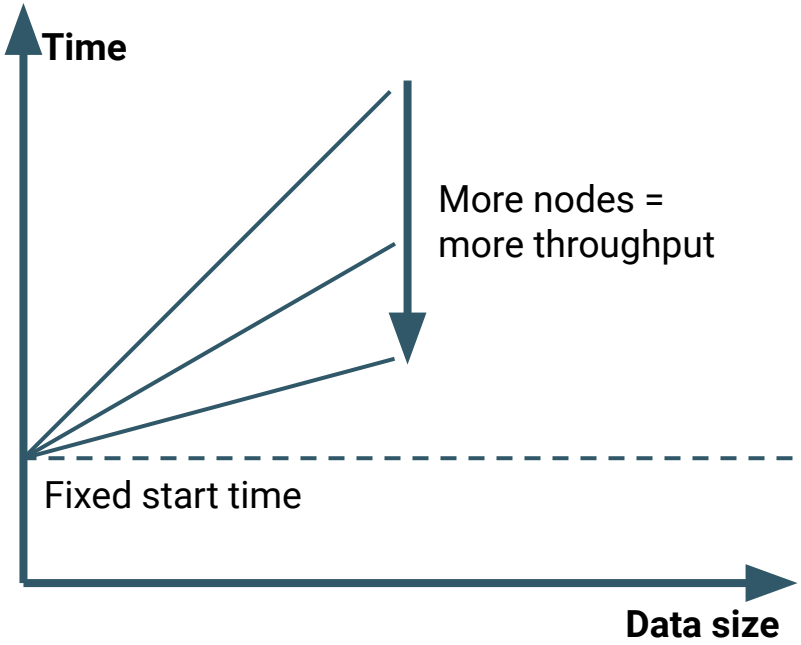
SUMMARY OF OPTIMIZATIONS

Optimized from O(snapshots^2) to O(changed ÷ threads)

	Naive export/sync	**Optimized export/sync**
Thread model	Single-threaded	Multi-threaded
Graph traversal	Depth-first-search	Breadth-first-search
Overlapping subgraphs	Duplicated	Deduplicated
Common snapshots	Processed	Skipped
Previously read files	Not cached	Cached
Execution time	O(snapshots^2)	O(changed ÷ threads)

LINEAR SCALABILITY

Iceberg replication is linearly scalable



SCALABILITY OF EACH PROCESS

The transfer process is the slowest. It distributed its workload.

Export— multi-threaded to reduce IO wait.

Transfer— distributed, scale-out.

Sync— multi-threaded to reduce IO wait.

ENTERPRISE READY

Optimized processes with all formats, versions, and Hadoop environments

ALL DATA FILE FORMATS

Apache Parquet— popular columnar format for Apache Spark and others

Apache ORC— popular columnar format for Apache Hive and others

Apache Avro— data serialization format for Apache Hadoop ecosystem

ALL TABLE SPEC VERSIONS

Iceberg table spec V1— basic operations

Iceberg table spec V2— row-level updates

ALL HADOOP CONFIGS

NameNode High Availability— a hot standby for HDFS namenode

Kerberos— strong authentication

TLS— data in transit encryption

ROADMAP

More storages, computes, use cases, and scale

STORAGE
More vendors

Apache HDFS— supported

Apache Ozone— an open source object store.

Amazon S3— a popular object store.

More to come

COMPUTE
More vendors

Private to private— supported

Public to public

Public to private

Private to public

USE CASE
Richer use cases

Failover— a standby cluster will take over a failed active cluster automatically.

Migrated external table— doesn't follow the default Iceberg directory structure.

Statistics— for query optimizers, with HMS and Puffin.

SCALE
Larger scale

Micro batch— more responsive replication

Fully distributed— for more linear scalability

THANK YOU

Francis Pang, Teddy Choi, Amit Saonerkar, Vinit Patni, Bo Soon Park, Harshal Kaushikbhai Ka Patel, Rakshith Chandraiah, Shivam Kumar, Subhasis Gorai, Shreenidhi Saigaonkar, Robert Kucsora, Anoop Sharma, Lavanya Subhash, Hemanth Kumar MS, Imran Khan, Shailesh Shiwalkar, Rahul Buddhisagar

SPECIAL THANKS TO

Bill Zhang, Siyi Wu, Shiv Moorthy, Imran Rashid, Vincent Kulandaisamy, Karthik Krishnamoorthy, Vinay Santurkar, Mani Ramasubramani, Sudhir Menon