DELTA LAKE

Under the ~~Hood~~ Sediments V2

Burak Yavuz, Denny Lee

Who are we

- Software Engineer – Databricks

  “We make your streams come true”
- Apache Spark™ Committer
- MS in Management Science & Engineering - Stanford University
- BS in Mechanical Engineering - Bogazici University, Istanbul

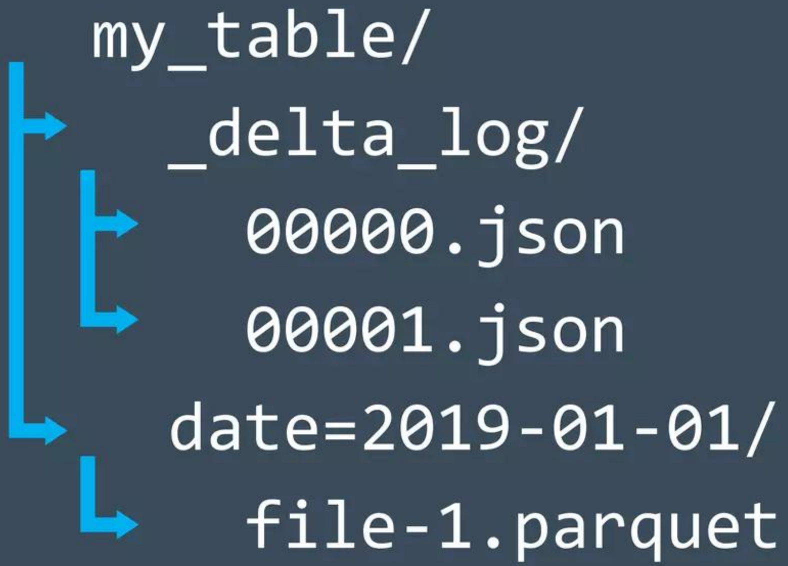
Who are we?

- Developer Advocate – Databricks
- Working with Apache Spark™ since v0.6
- Former Senior Director Data Science Engineering at Concur
- Former Microsoftie: Cosmos DB, HDInsight (Isotope)
- Masters Biomedical Informatics - OHSU
- BS in Physiology - McGill

Outline

- The Delta Log (Transaction Log)
  - Contents of a commit
  - Optimistic Concurrency Control
  - Computing / updating the state of a Delta Table
- Time Travel
- Batch / Streaming Queries on Delta Tables
- Demo

Delta On Disk



```
my_table/
  _delta_log/
    00000.json
    00001.json
  date=2019-01-01/
    file-1.parquet
```

Table = result of a set of actions

Update Metadata – name, schema, partitioning, etc

Add File – adds a file (with optional statistics)

Remove File – removes a file

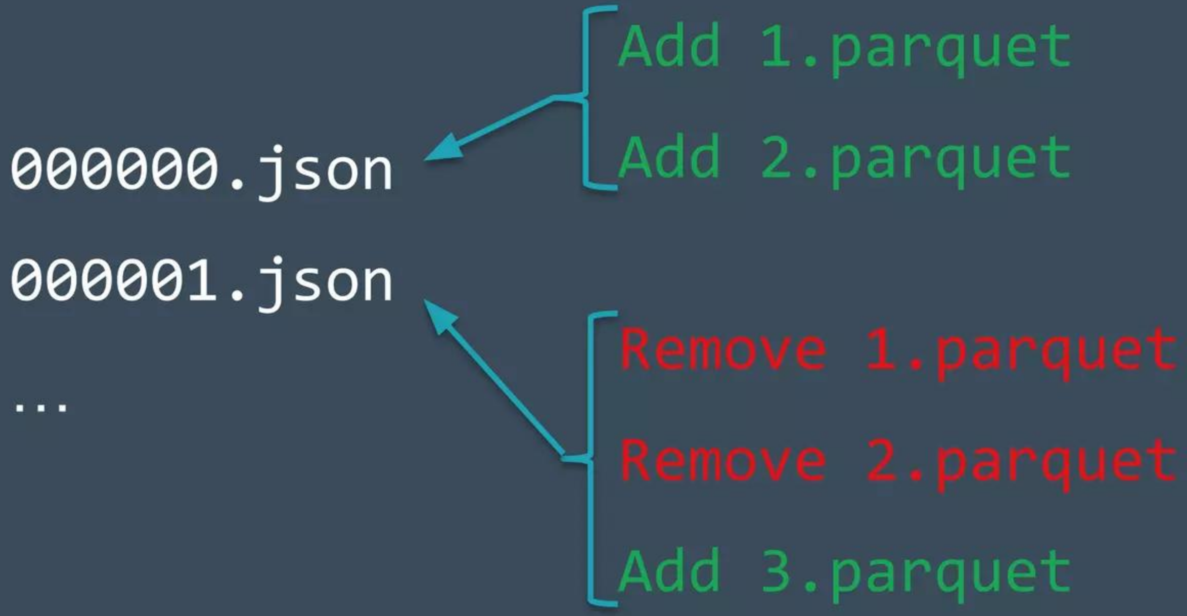
Set Transaction – records an idempotent txn id

Change Protocol – upgrades the version of the txn protocol

Result: Current Metadata, List of Files, List of Txns, Version

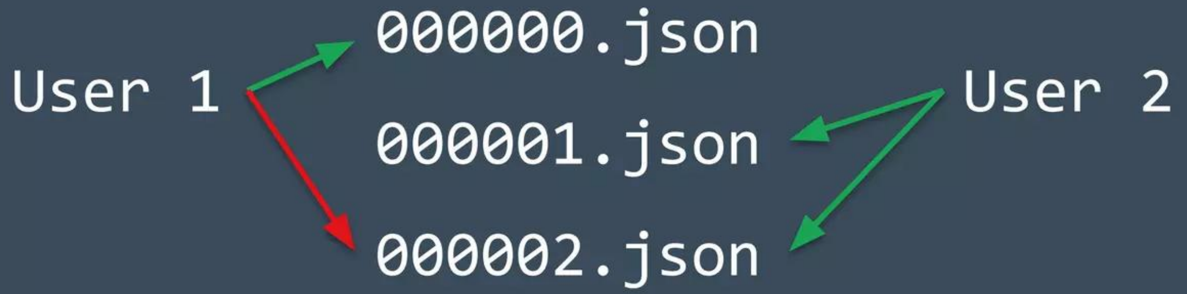
Implementing Atomicity

Changes to the table are stored as ordered, atomic units called commits



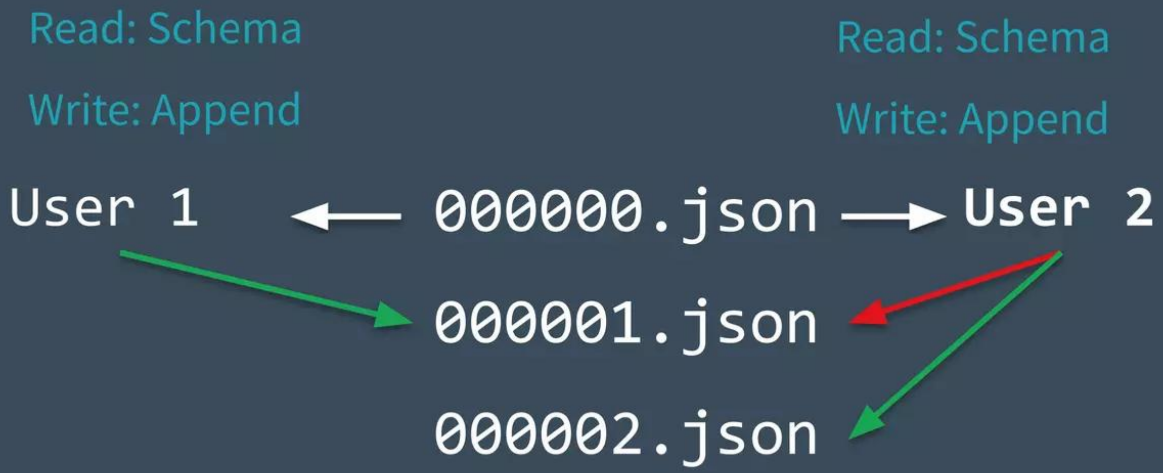
Ensuring Serializability

Need to agree on the order of changes, even when there are multiple writers.



Solving Conflicts Optimistically

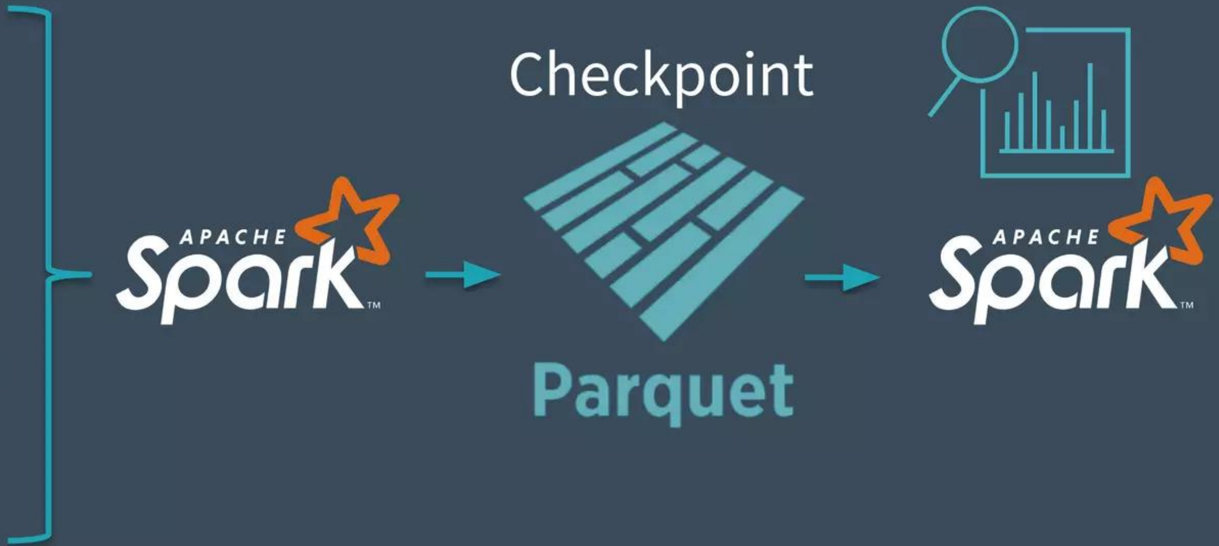
1. Record start version
2. Record reads/writes
3. Attempt commit
4. If someone else wins, check if anything you read has changed.
5. Try again.



Handling Massive Metadata

Large tables can have millions of files in them! How do we scale the metadata? Use Spark for scaling!

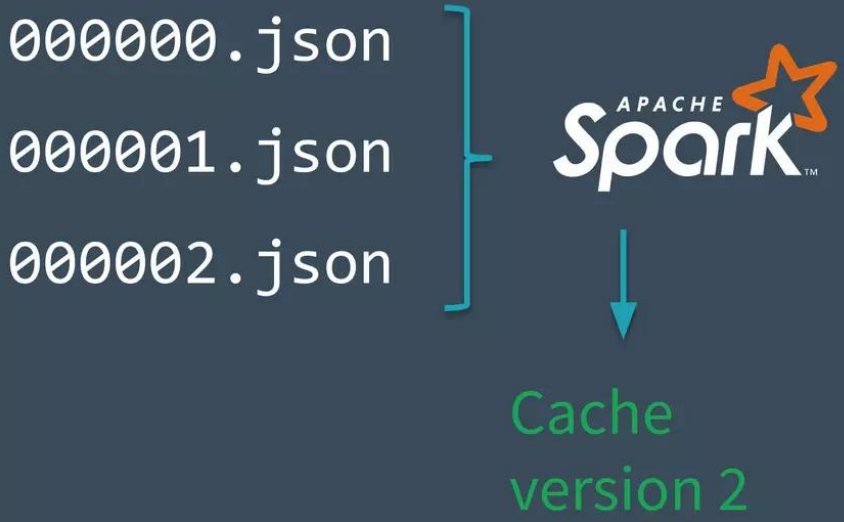
```
Add 1.parquet
Add 2.parquet
Remove 1.parquet
Remove 2.parquet
Add 3.parquet
```



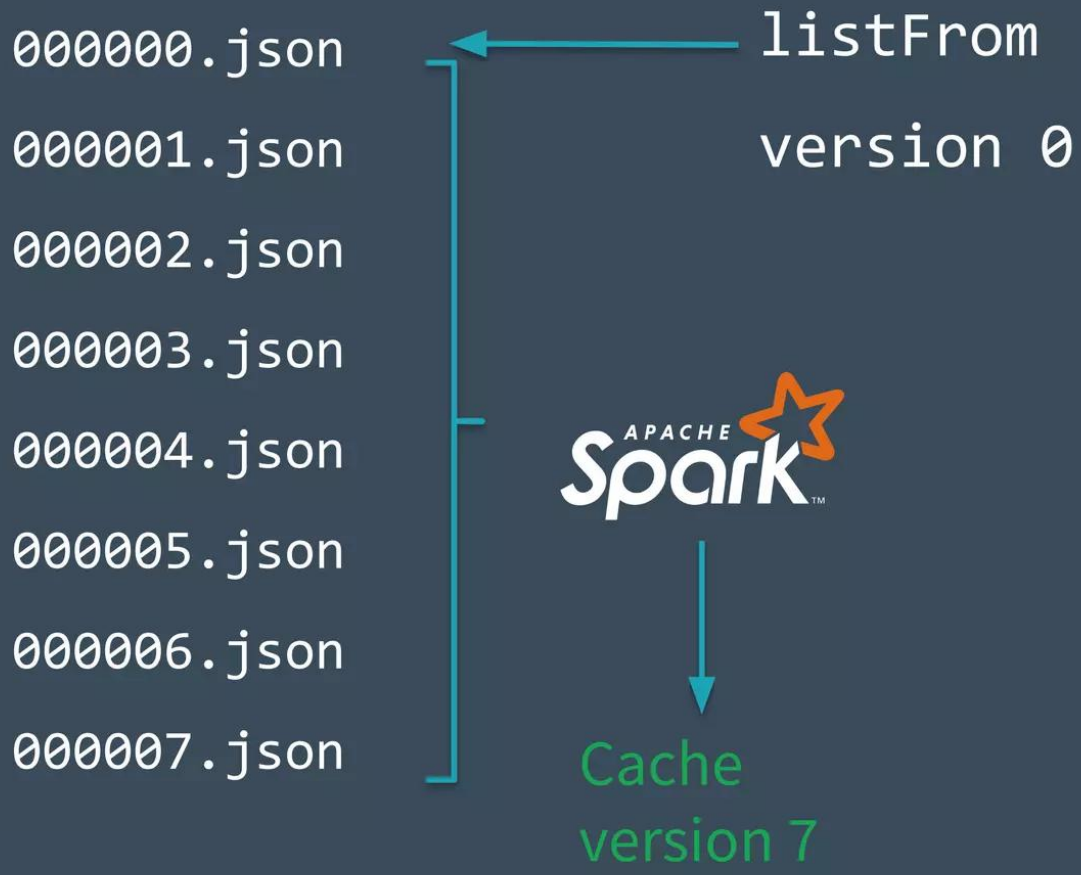
Checkpoints

- Contains the latest state of all actions at a given version
- No need to read tons of small JSON files to compute the state
- Why Parquet?
  - No parsing overhead
  - Column pruning capabilities

Computing Delta’s State



Updating Delta's State

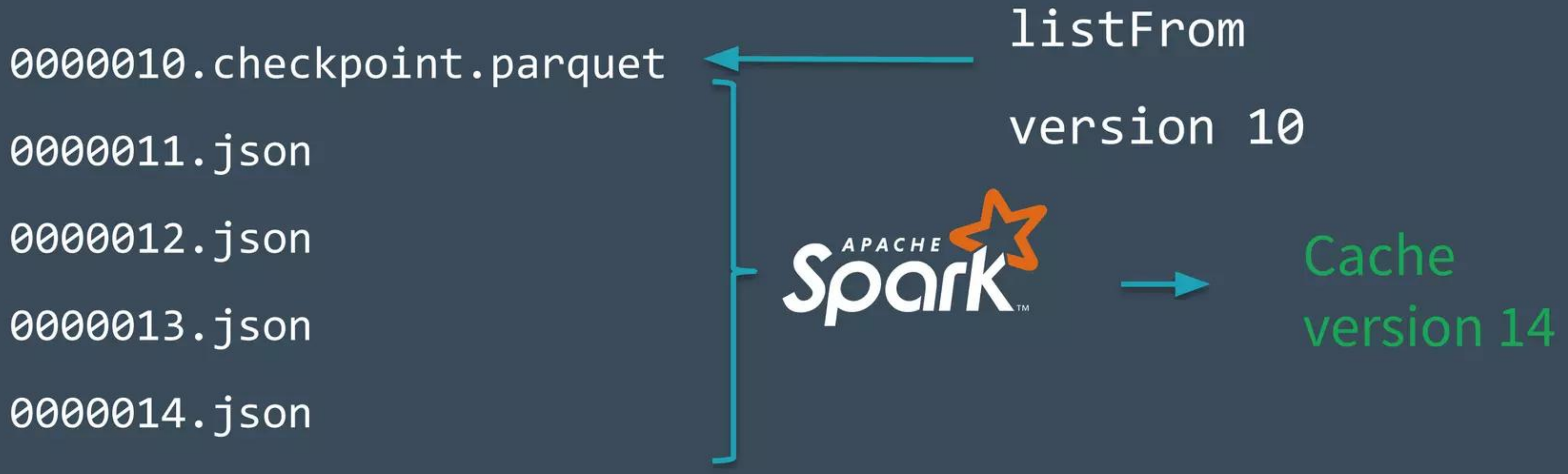


Updating Delta's State

```
000000.json          ←  listFrom
                        version 0
...

000007.json

000008.json

000009.json

0000010.json

0000010.checkpoint.parquet
0000011.json
0000012.json
```

Spark → Cache version 12

Updating Delta's State



Outline

- The Delta Log (Transaction Log)
- Time Travel
  - How it works
  - Limitations
- Batch / Streaming Queries on Delta Tables
- Demo

Time Travelling by version

```
SELECT * FROM my_table VERSION AS OF 1071;
SELECT * FROM my_table@v1071  -- no backticks to specify @
spark.read.option("versionAsOf", 1071).load("/some/path")
spark.read.load("/some/path@v1071")
```

$\downarrow$

```
deltaLog.getSnapshotAt(1071)
```

Time Travelling by timestamp

```
SELECT * FROM my_table TIMESTAMP AS OF '1492-10-28';

SELECT * FROM my_table@14921028000000000 -- yyyyMMddHHmmssSSS

spark.read.option("timestampAsOf", "1492-10-28").load("/some/path")

spark.read.load("/some/path@14921028000000000")
```

```
deltaLog.getSnapshotAt(1071)
```

Time Travelling by timestamp

Commit timestamps come from storage system modification timestamps

```
001070.json     375-01-01
001071.json    1453-05-29
001072.json    1923-10-29
001073.json    1920-04-23
```

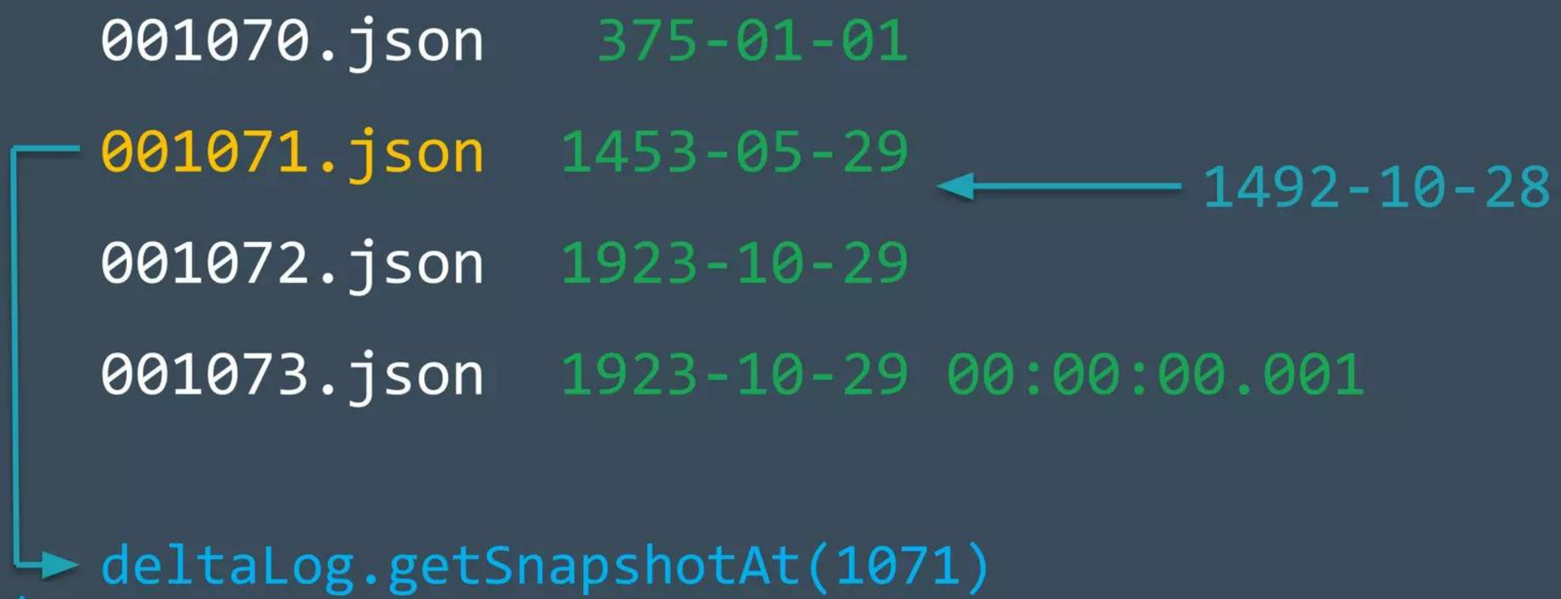
Time Travelling by timestamp

Timestamps can be out of order. We adjust by adding 1 millisecond to the previous commit’s timestamp.

```
001070.json     375-01-01   375-01-01
001071.json    1453-05-29  1453-05-29
001072.json    1923-10-29  1923-10-29
001073.json    1920-04-23  1923-10-29 00:00:00.001
```

Time Travelling by timestamp

Price is right rules: Pick closest commit with timestamp that doesn’t exceed the user’s timestamp.



```
001070.json   375-01-01
001071.json  1453-05-29   <——————  1492-10-28
001072.json  1923-10-29
001073.json  1923-10-29 00:00:00.001

deltaLog.getSnapshotAt(1071)
```

If interested in more, check out the Time Travel deep dive session:

Data Time Travel by Delta Time Machine

THURSDAY, 15:00 (GMT)

Time Travel Limitations

- Requires transaction log files to exist
  - delta.logRetentionDuration = "interval <interval>”
- Requires data files to exist
  - delta.deletedFileRetentionDuration = "interval <interval>"
  - If you Vacuum, you lose data
- Therefore time travel in order of months/years infeasible
  - Expensive storage
  - Computing Delta’s state won’t scale

Outline

- The Delta Log (Transaction Log)
- Time Travel
- Batch / Streaming Queries on Delta Tables
- Demo

Batch Queries on a Delta Table

1. Update the state of the table
2. Perform data skipping using provided filters
   - Filter AddFile events according to metadata, e.g. partitions and stats
3. Execute query

Streaming Queries on a Delta Table

1. Update the state of the table
2. Skip data by using partition filters – cache snapshot
3. Start processing files 1000 (maxFilesPerOffset) files at a time
   - No guaranteed order
   - Also maxBytesPerTrigger can be used
4. Once snapshot is over, start tailing json files
   - Ignore files that have dataChange=false => Optimized files are ignored
   - Use ignoreChanges if you have data removed or updated
   - GOTCHA: Vacuum may delete the files referenced in the json files

Streaming Queries on a Delta Table

When using startingVersion or startingTimestamp

1. Start tailing json files at corresponding version
   - Ignore files that have dataChange=false => Optimized files are ignored
   - GOTCHA: Vacuum may delete the files referenced in the json files
   - startingVersion and startingTimestamp is inclusive
   - startingTimestamp will start processing from the next version if a commit hasn't been made at the given timestamp (unlike Time Travel)

startingTimestamp in Streaming

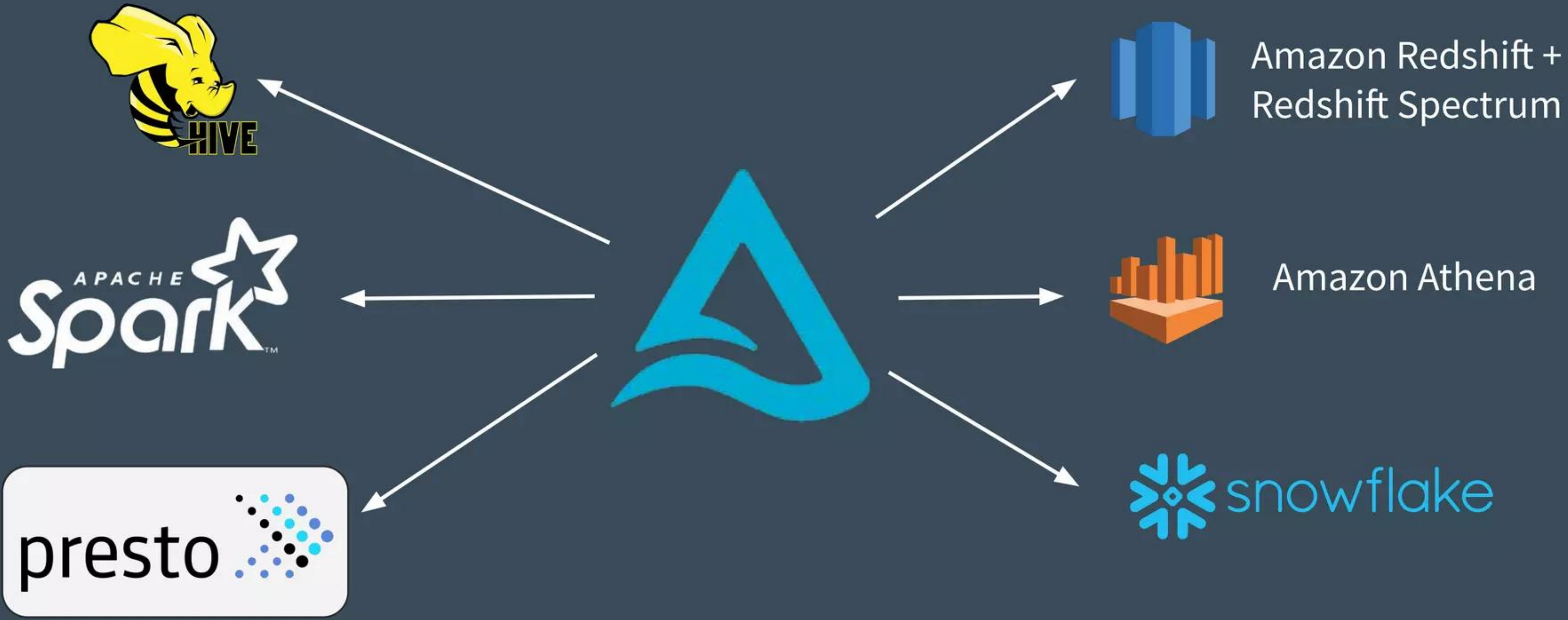
Process all changes beginning at that timestamp

```
001070.json    375-01-01
001071.json   1453-05-29
                          ←——————  1492-10-28
001072.json   1923-10-29
001073.json   1923-10-29 00:00:00.001
```

Demo

Delta Lake Connectors

Standardize your big data storage with an open format accessible from various tools



Delta Lake Partners and Providers

More and more partners and providers are working with Delta Lake

Google Dataproc

Privacera

ATTUNITY

tableau

talend

Azure Synapse Analytics

Informatica

Streamsets

Qlik

WANDisco

Users of Delta Lake

Tencent 腾讯 | Alibaba Group | COMCAST | VIACOM | ciena | JAM CITY

edmunds | databricks | healthdirect Australia | Booz | Allen | Hamilton | McAfee Together is power. | CONDÉ NAST

TILTING POINT | Mc Graw Hill Education | acorns | EVERQUOTE | Namely | usermind

ZEISS | split | upwork | DOLLAR SHAVE CLUB | wehkamp

Thank You

"Do you have any questions for my prepared answers?"

– Henry Kissinger