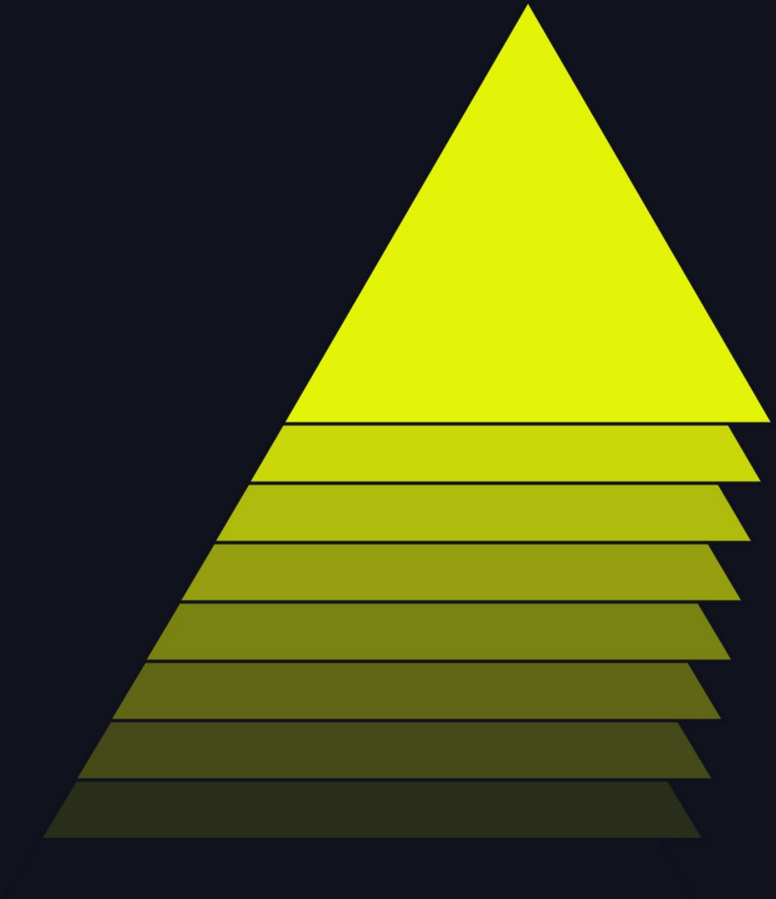
DATA+AI SUMMIT BY databricks

Towards Multi-Statement Transactions in Delta

Prakhar Jain, Databricks



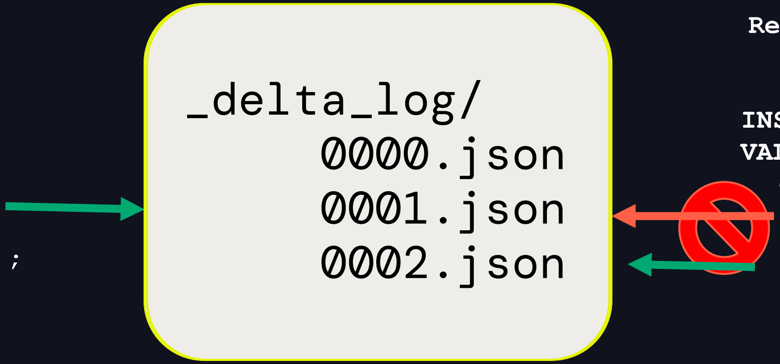
Delta Lake commits have limitations

- No multi-statement transactions
- No multi-table transactions
- No catalog integration

Modern Day Delta Commits

- Filesystem based commits
  - Leverages atomic filesystem primitives i.e. put-if-absent
- Commit => Write *<version>.json* in _delta_log directory

```
INSERT INTO mydb.employees
VALUES
   (1, 'John Doe', '1980-01-01'),
   (2, 'Jane Smith', '1990-05-15'),
   (3, 'Bob Johnson', '1985-12-31');
```



Retry Commit

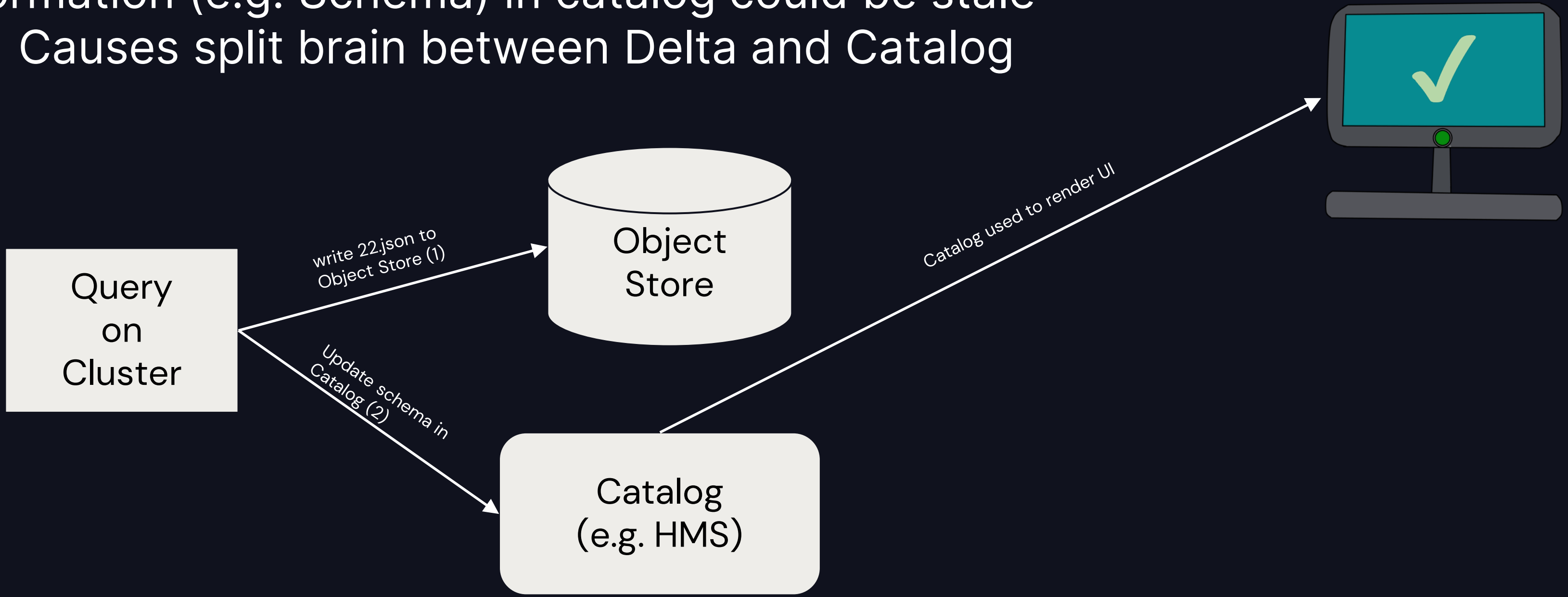
```
INSERT INTO mydb.employees
VALUES
   (4, 'Mike Johnson', '1980-01-01');
```

NO MULTI-TABLE TRANSACTIONS

- Cloud object stores do not provide APIs to write multiple files atomically
- E.g. Can’t write following two commit files together
  - 145.json on table1
  - 124.json on table2

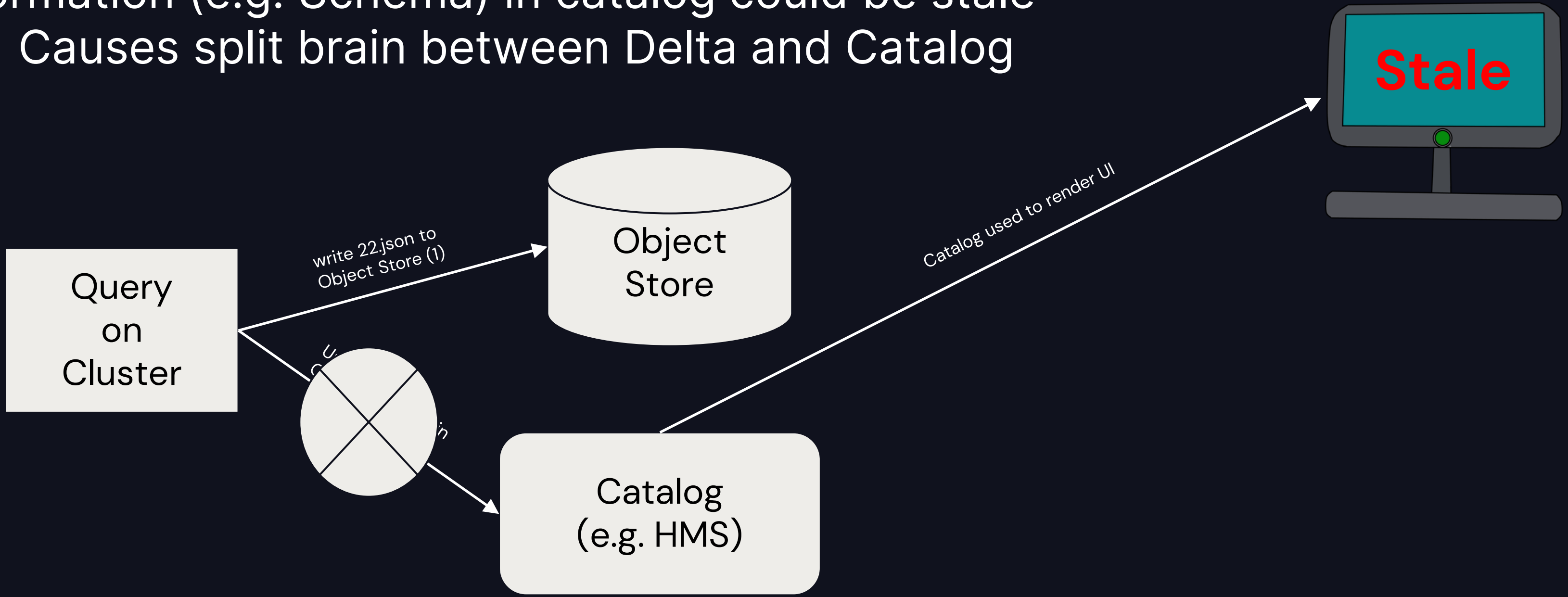
NO CATALOG INTEGRATION

- Catalog is updated in a best effort manner (#2409)
- Information (e.g. Schema) in catalog could be stale
  - Causes split brain between Delta and Catalog


Query on Cluster
write 22.json to Object Store (1)
Object Store
Update schema in Catalog (2)
Catalog (e.g. HMS)
Catalog used to render UI
✓

NO CATALOG INTEGRATION

- Catalog is updated in a best effort manner (#2409)
- Information (e.g. Schema) in catalog could be stale
  - Causes split brain between Delta and Catalog



Delta Managed Commits

Managed Commits

Introduction

- An open and flexible way of doing commits on a Delta table
- Every table has a single *commit owner*. The commit owner:
    - Defines how commits happen
    - Handles coordination between multiple writers
    - Source of truth for the latest commits (for readers)
- **Open:** Anyone can define their Commit Owner
- Commit-owner info stored in DeltaLog

Available as preview in Delta 4.0-preview!

Managed Commits

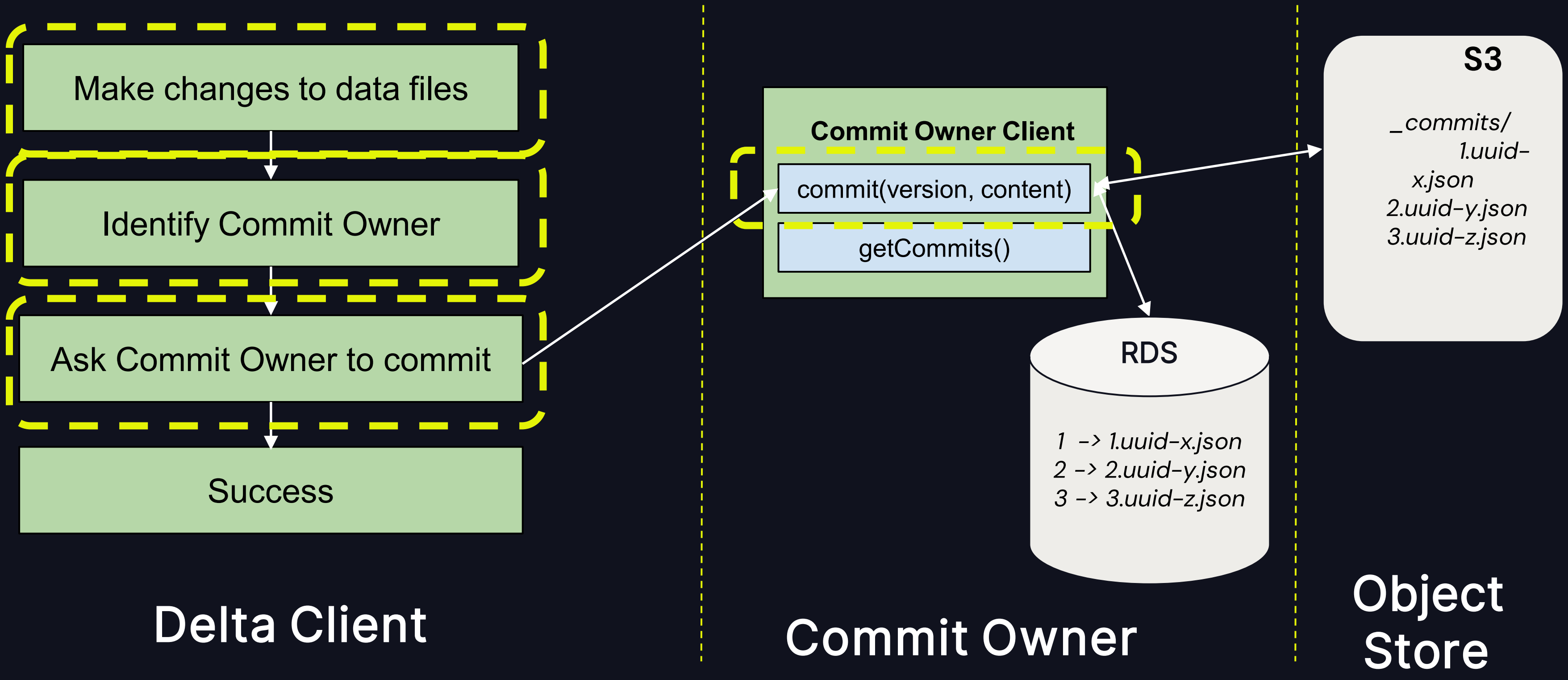
Commit Owner Interface

Any vendor could implement this simple interface to become a commit owner!

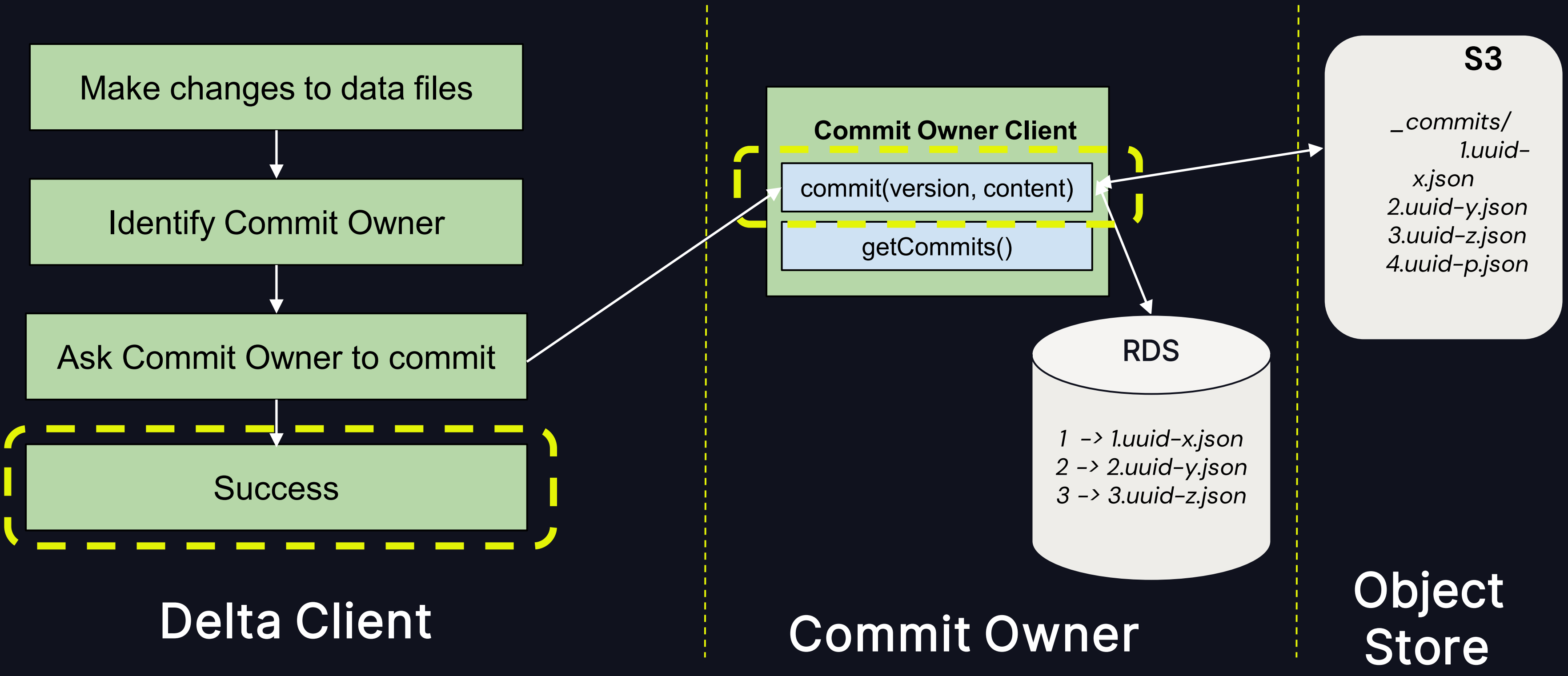
Pseudocode

```
Interface CommitOwnerClient {
  // Commits the given iterator of changes to a given version
  def commit(version, content): Commit

  // Returns the uuid commits in the given range (if any)
  def getCommits(from, optional_to): Array[Commit]
}


Class Commit {
  String path;
  Int length;
  Long commitTime;
}
```

Commit flow with Managed Commits



Commit flow with Managed Commits



Managed Commits

Implementation Details

- New Writer table feature: *managedCommit*
- New commit format:

  <version>.<unique-id>.json

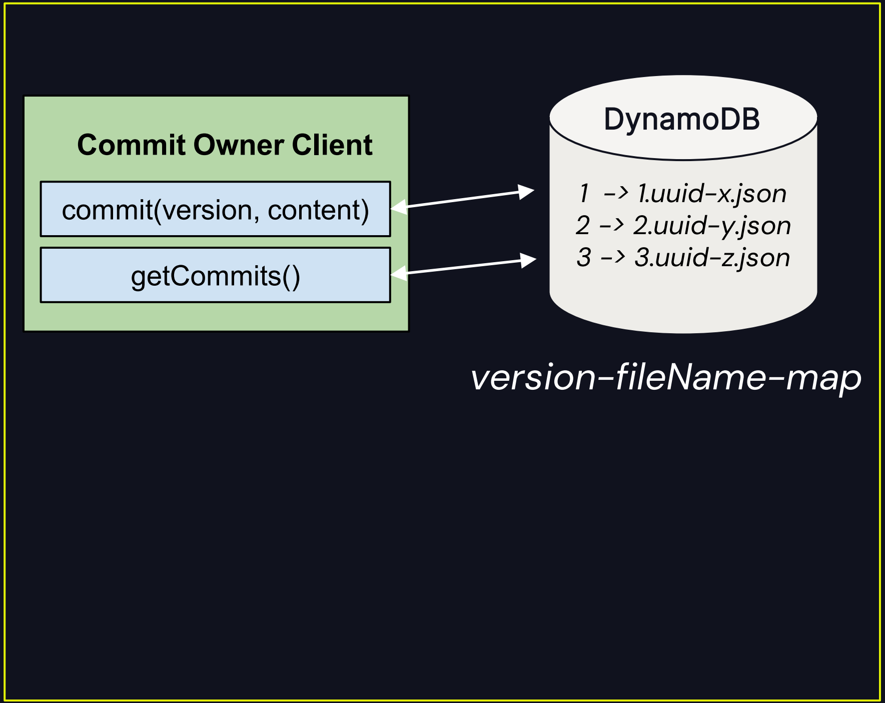
  E.g. 23.46d70172.json

- Delta Client makes data file changes
- Delta Client contacts commit-owner to do the actual commit
  - Commit Owner writes the commit file and persists the commit as per its own spec

Commit Owner

Holds Information about recent commits

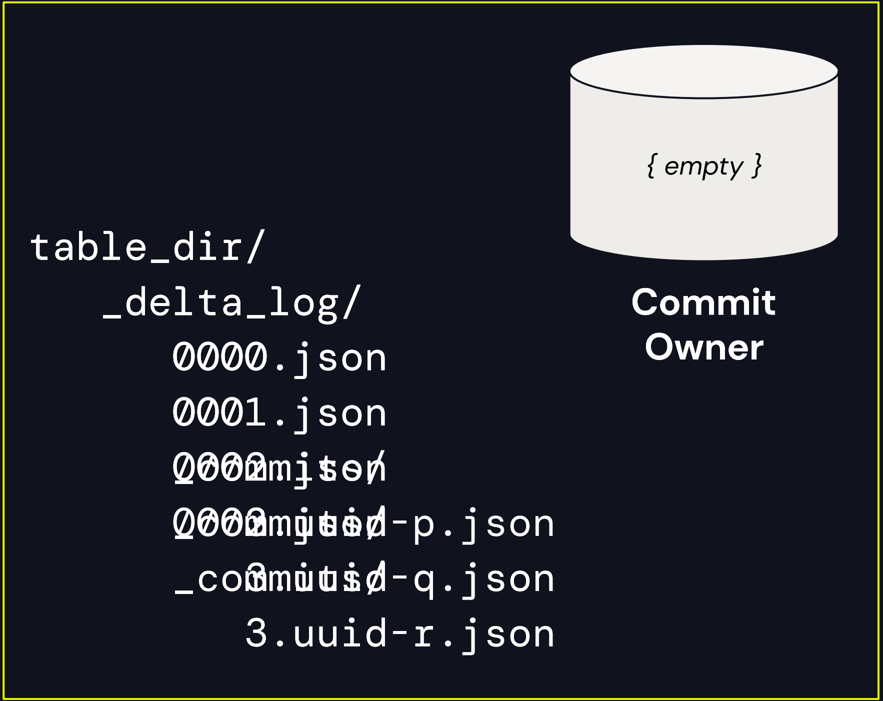
- Commit owner tracks commit-to-fileName map
- Where is it tracked?
  - Depends on the implementation
- Could be
  - Catalog e.g. Unity Catalog, HMS
  - DynamoDB
  - any persistent storage



Reads are Backward Compatible

Backfilling Commits

- Backfilling: Copying uuid commit into self discoverable format: **<version>.json**
- Enables older Delta clients to READ managed-commit tables
- Commit owner could stop tracking the file after backfilling



```
table_dir/
   _delta_log/
      0000.json
      0001.json
      0002.json
      _commits/
      0003.json
      _commits/
         2.uuid-p.json
      _commits/
         3.uuid-q.json
         3.uuid-r.json
```

Looking forward

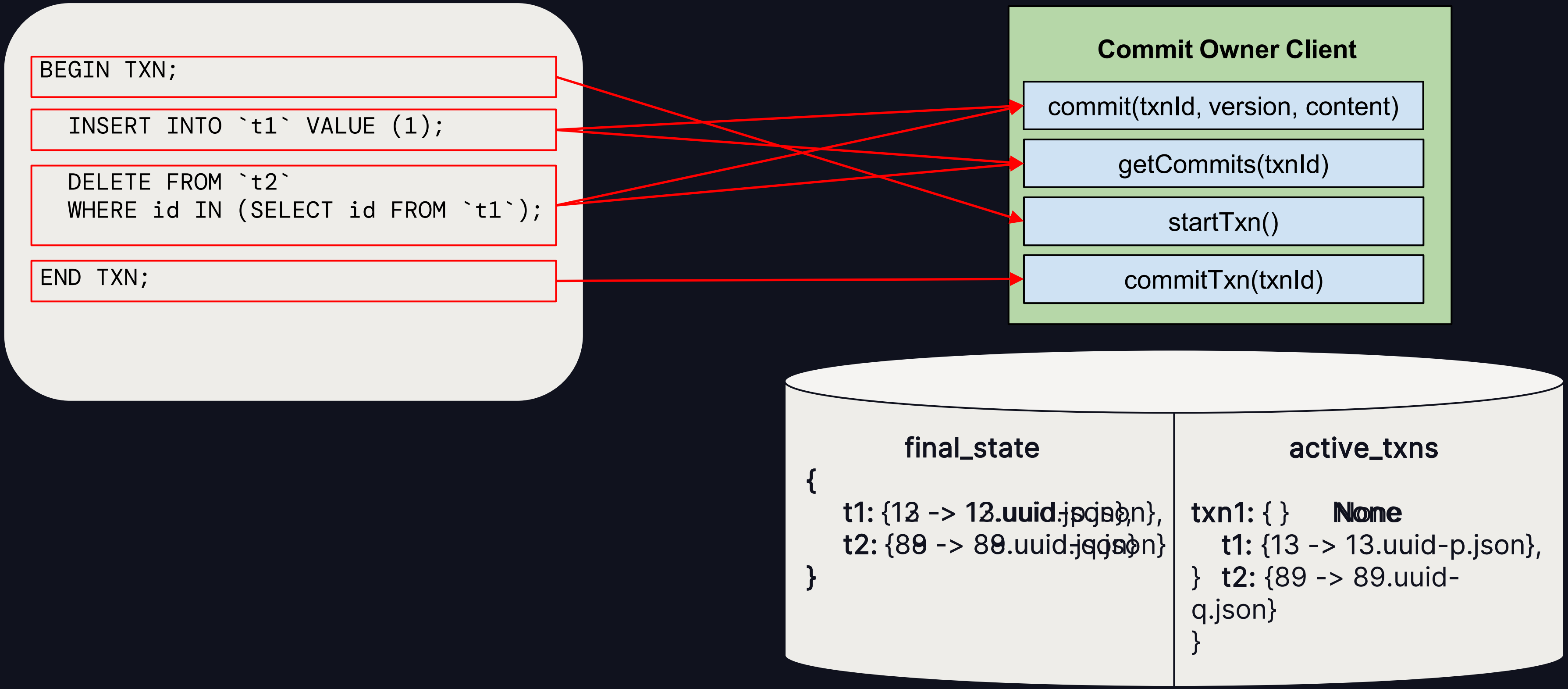
Commit Owner = Transaction Manager

- **Managed Commits** => all commits go through a commit owner
- Allows Catalog based commits
  - If Commit Owner = Catalog, then it knows changes to the table atomically as it is the one who brokers the commit.
- Commit Owner becomes the central coordinator
  - Key requirement for Multi-table-Multi-Statement Txns

Next Steps

- Extend the Managed Commit API to support multiple-statements and multiple-tables

Multi-Statement Transactions



```
BEGIN TXN;
  INSERT INTO `t1` VALUE (1);
  DELETE FROM `t2`
  WHERE id IN (SELECT id FROM `t1`);
END TXN;
```

Commit Owner Client
commit(txnId, version, content)
getCommits(txnId)
startTxn()
commitTxn(txnId)
final_state
{
t1: {12 -> 12.uuid.json},
t2: {88 -> 88.uuid.json}
t1: {13 -> 13.uuid-p.json},
t2: {89 -> 89.uuid-q.json}
}
active_txns
txn1: { }
None
txn1: {
t1: {13 -> 13.uuid-p.json},
t2: {89 -> 89.uuid-q.json}
}

Thank you!

Relevant Github Issues

- Delta Managed Commits #2598
- Delta Multi-table transactions: #832

DATA+AI SUMMIT