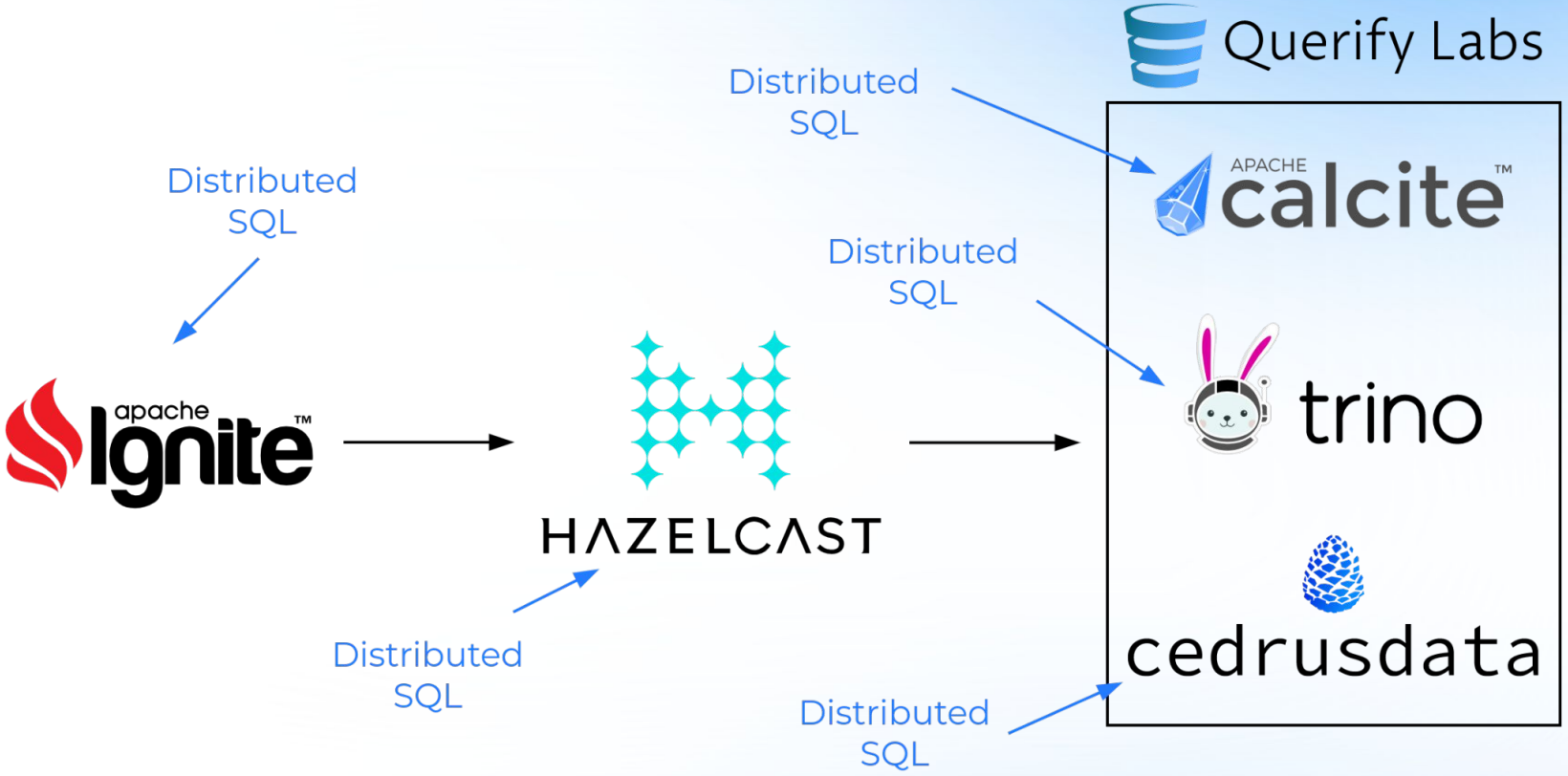
Querify Labs

Как работает Apache Iceberg на примере Trino

Владимир Озеров
Кверифай Лабс / CedrusData

О себе


Querify Labs
Distributed SQL
Distributed SQL
Distributed SQL
Distributed SQL
Distributed SQL
apache Ignite
HAZELCAST
APACHE calcite
trino
cedrusdata

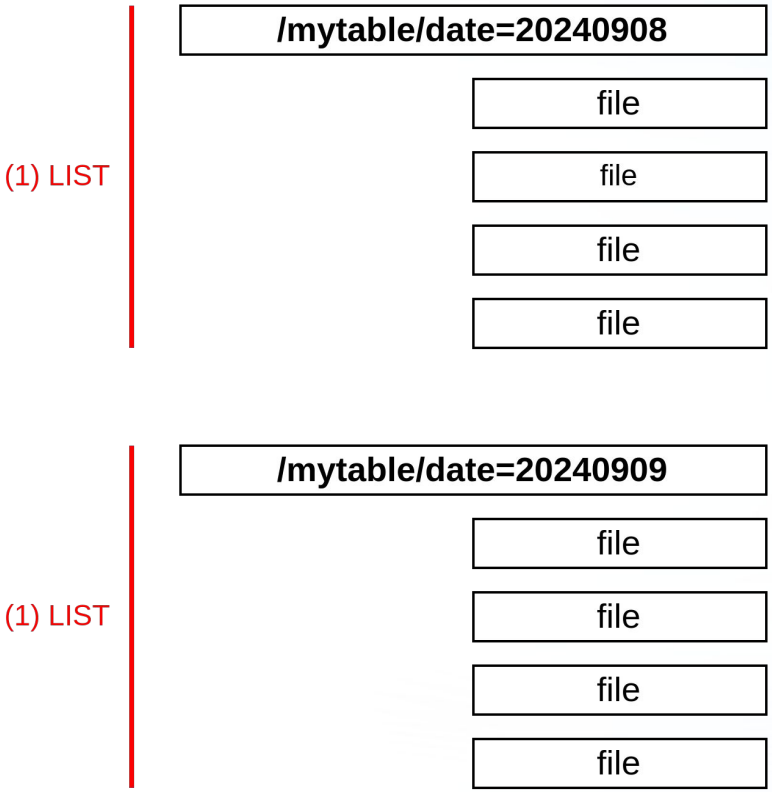
Мотивация

План

- Мотивация
- Как работает Iceberg
- Как Trino работает с Iceberg

История появления

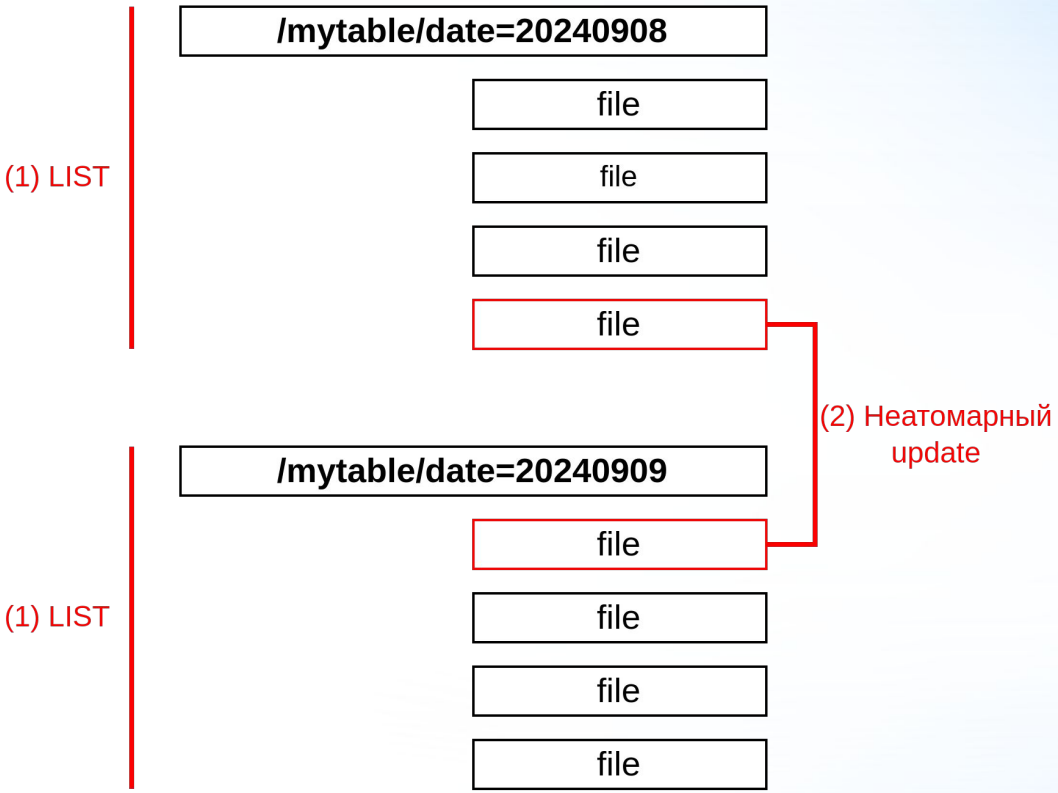
Мотивация: очень большие таблицы в data lake



SQL к очень большим Hive таблицам из Spark и Presto:

1. Долгое планирование из-за обилия операций LIST

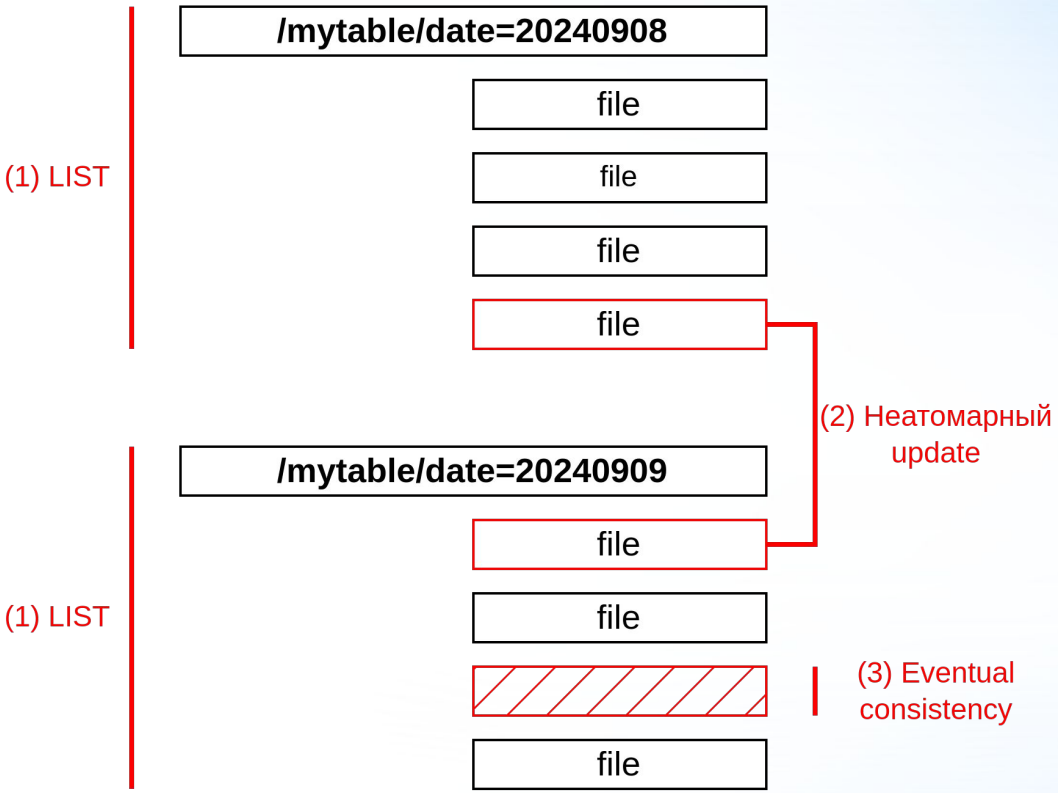
Мотивация: очень большие таблицы в data lake



SQL к очень большим Hive таблицам из Spark и Presto:

1. Долгое планирование из-за обилия операций LIST
2. Невозможность атомарного добавления или изменения нескольких файлов

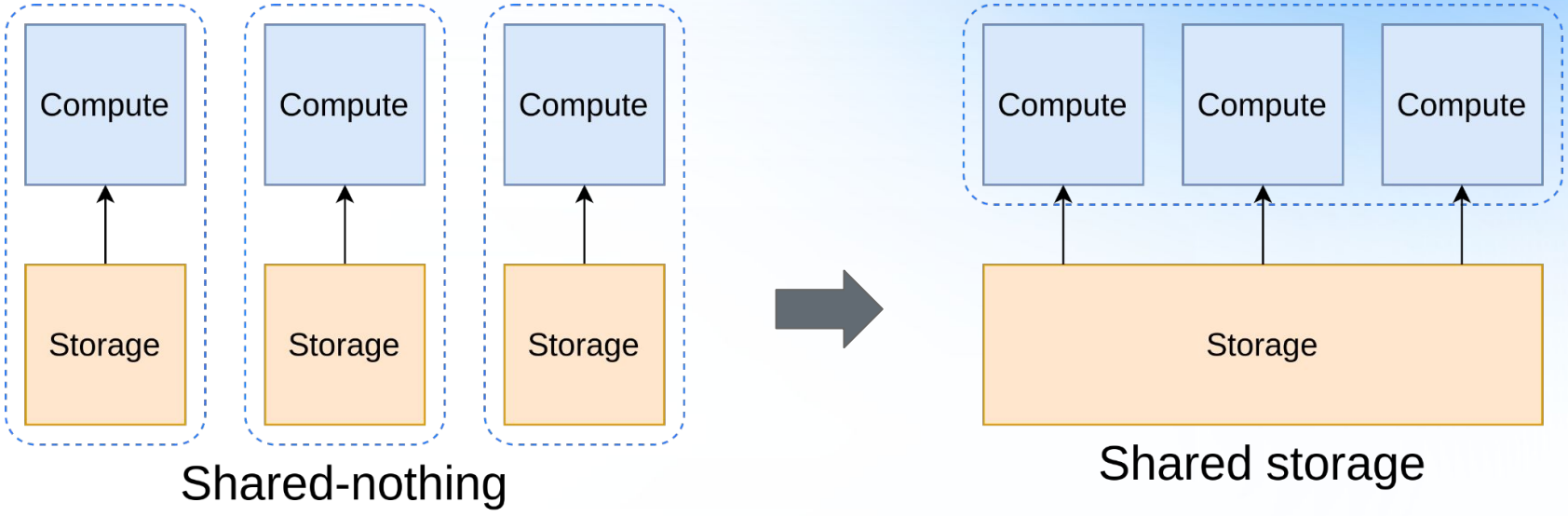
Мотивация: очень большие таблицы в data lake



SQL к очень большим Hive таблицам из Spark и Presto:

1. Долгое планирование из-за обилия операций LIST
2. Невозможность атомарного добавления или изменения нескольких файлов
3. Слабые гарантии консистентности со стороны storage, которые могут приводить к неправильным результатам

Мотивация: переход к shared storage



Уменьшение стоимости и time-to-market:

- Меньше дублирования данных
- Эластичное масштабирование вычислительных мощностей
- Множественные движки под разные нагрузки

Табличные форматы

- Поддерживает транзакции
- Поддерживает эволюцию схемы
- Высокая производительность при работе с большими таблицами

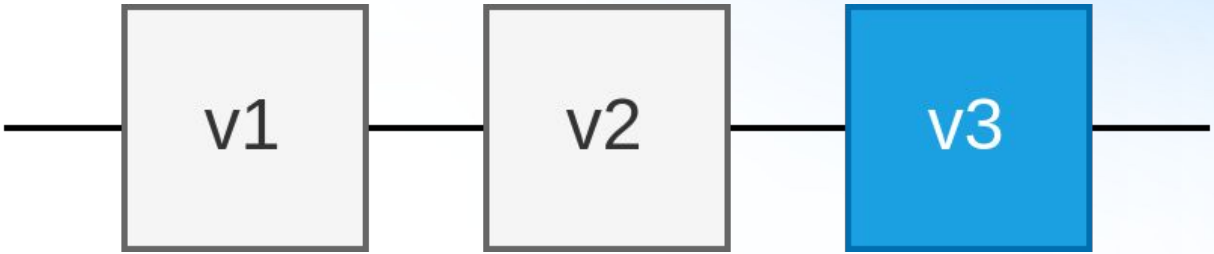
Open-source,
первопроходцы,
сложный

Open-source,
активное развитие

Размытые границы
коммерческих
интересов Databricks

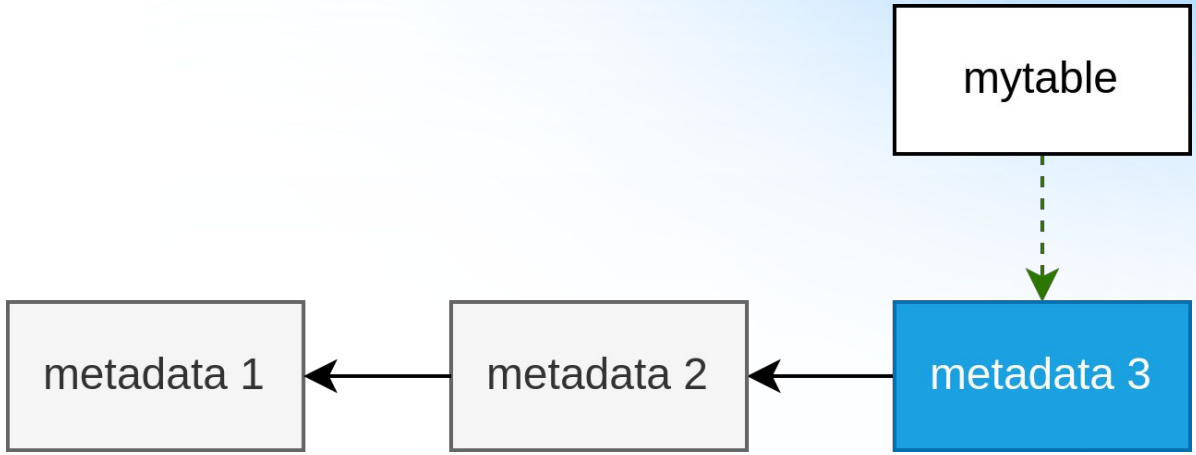
Табличный формат Apache Iceberg

Журналирование в СУБД



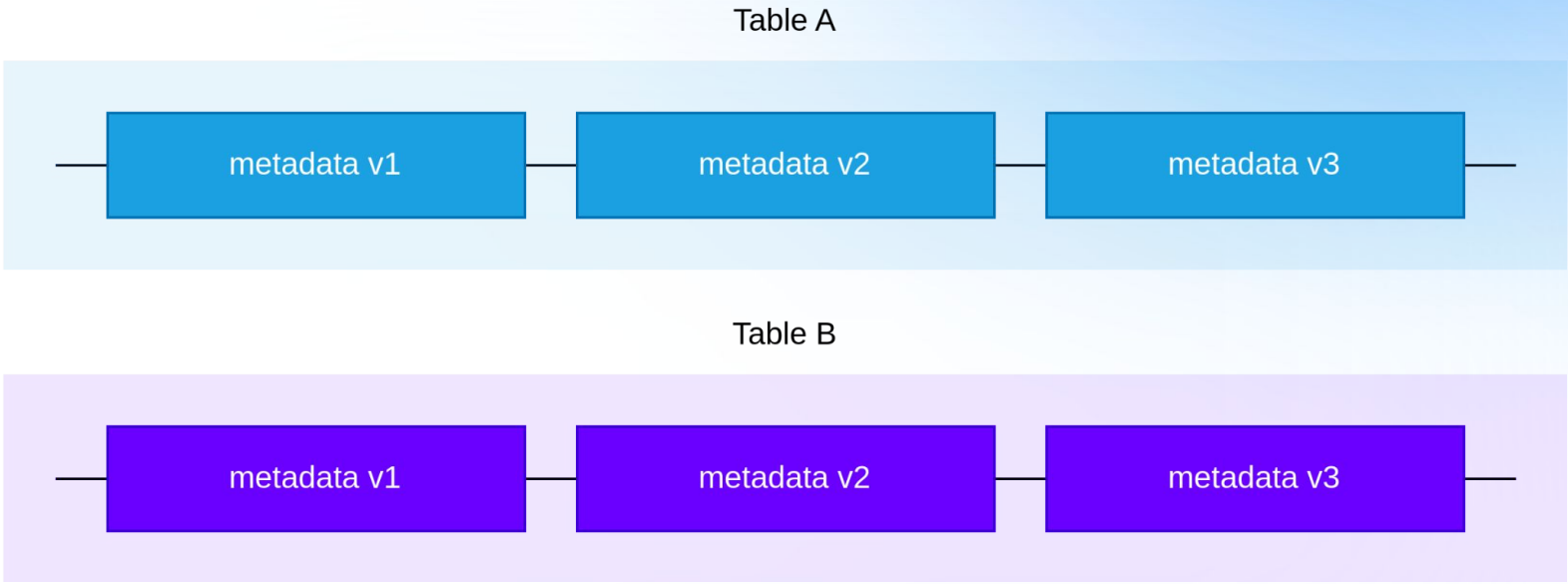
Журнал – линейная история изменений

Архитектура Iceberg: metadata



На каждое изменение таблицы Apache Iceberg создает новый metadata file, который полностью описывает таблицу.

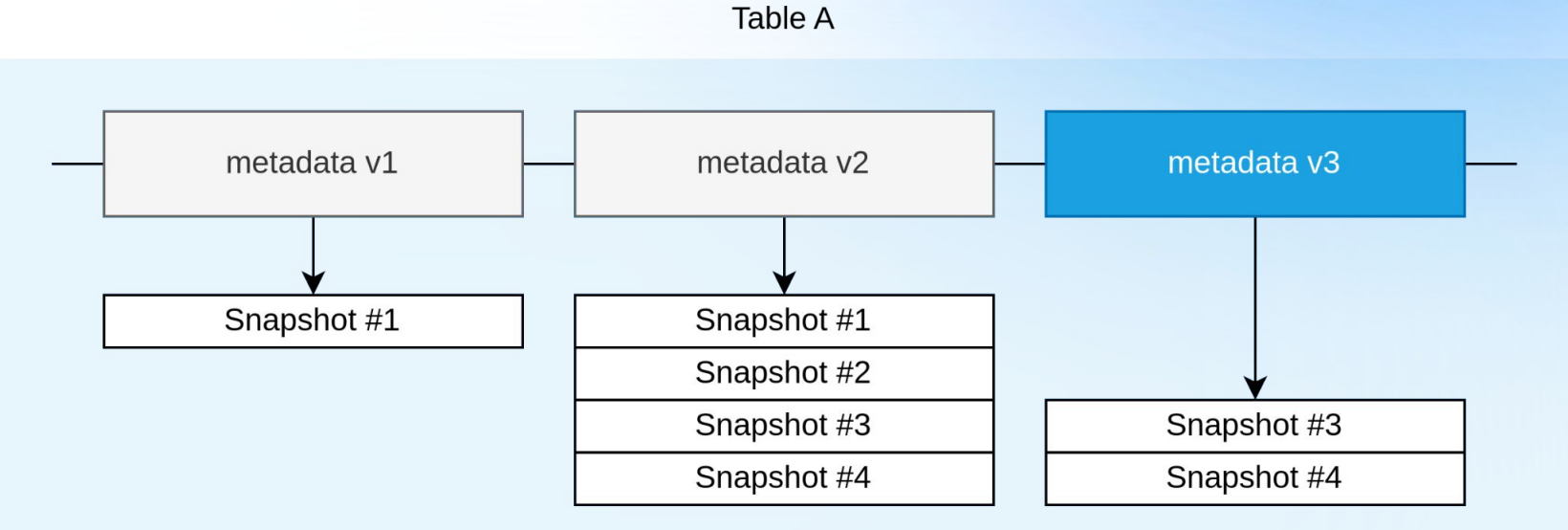
Архитектура Iceberg: границы транзакций



Поддерживаются транзакции в рамках одной таблицы

- Баланс практической пользы и сложности реализации
- В настоящий момент идет работа на multi-table транзакциями

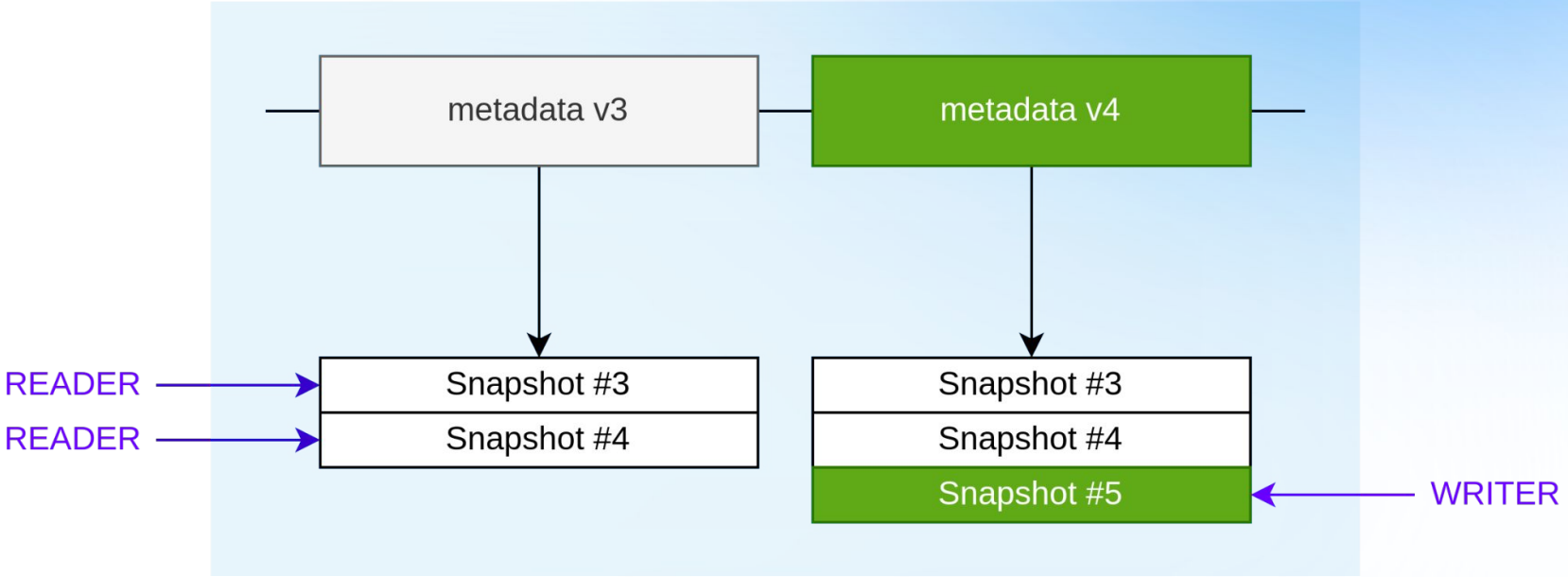
Архитектура Iceberg: отслеживание изменений



Metadata – запись в журнале, которая описывает все известные состояния таблицы:

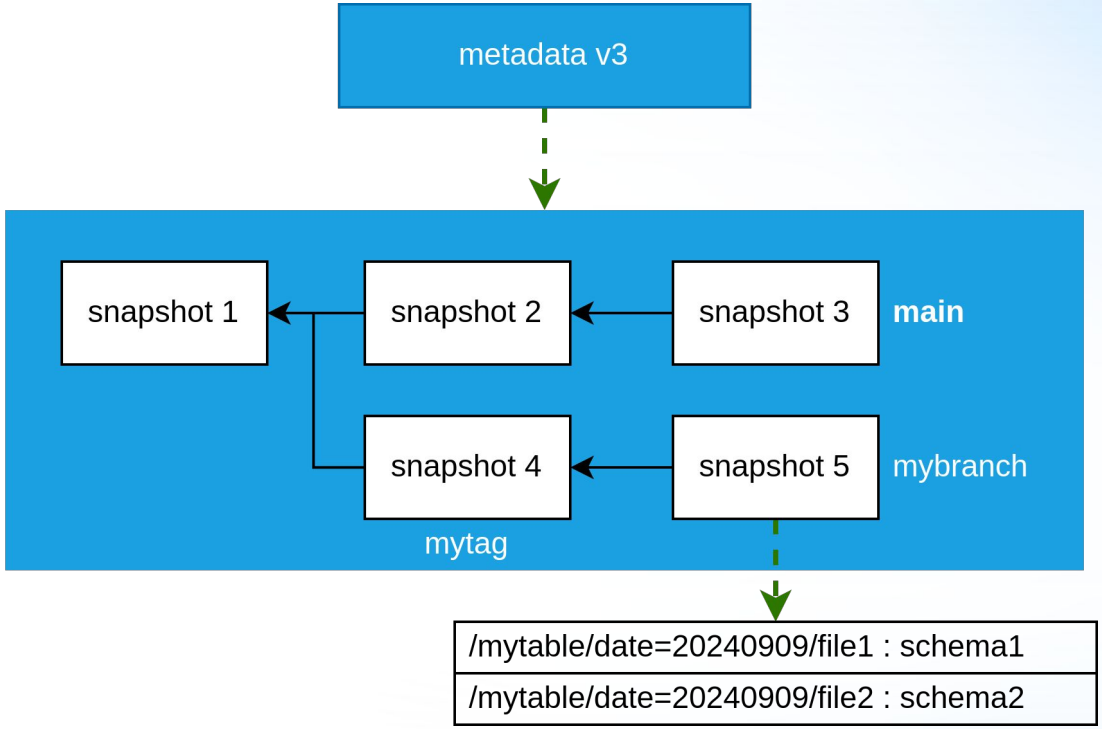
- Изменения данных
- Изменения метаданных (схема, статистики, и т.д.)

Архитектура Iceberg: изоляция транзакций с помощью MVCC



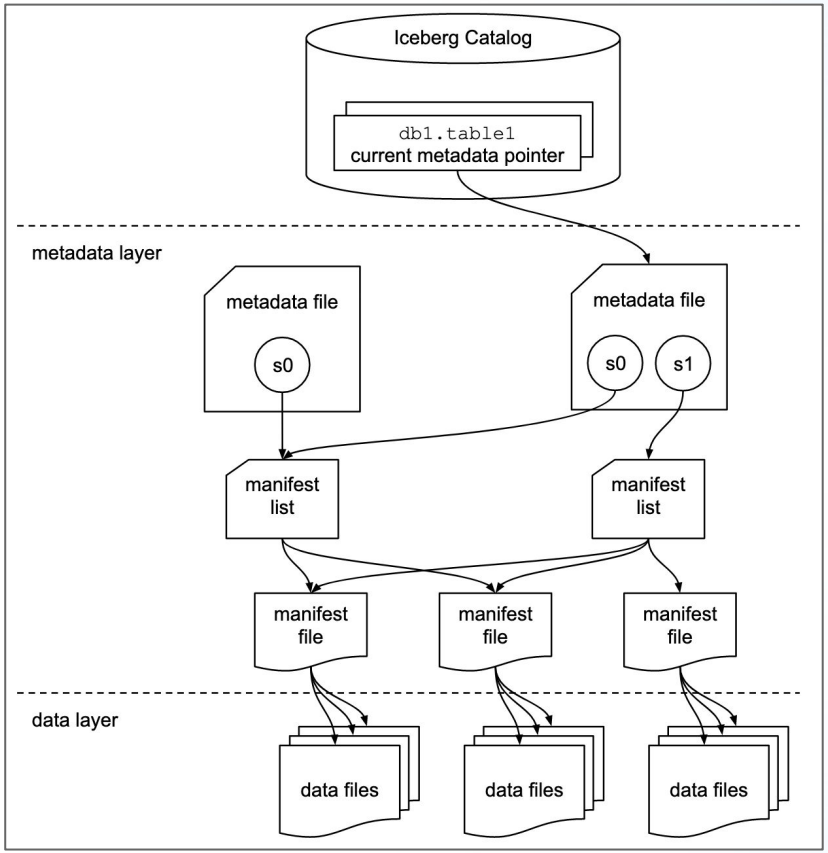
- Множественные версии данных позволяют читать и писать данные без блокировок
- Старые снэпшоты необходимо периодически удалять
  - Условная аналогия – PG vacuum

Архитектура Iceberg: snapshot



- Metadata содержит список snapshot
- Snapshot-ы организованы в **бранчи** – линейные истории изменений. По умолчанию таблица имеет один main бранч
- С отдельным snapshot может быть сопоставлен **тэг**
- Snapshot содержит список **файлов данных,** из которых состояла таблица на момент его создания

Все метаданные Iceberg



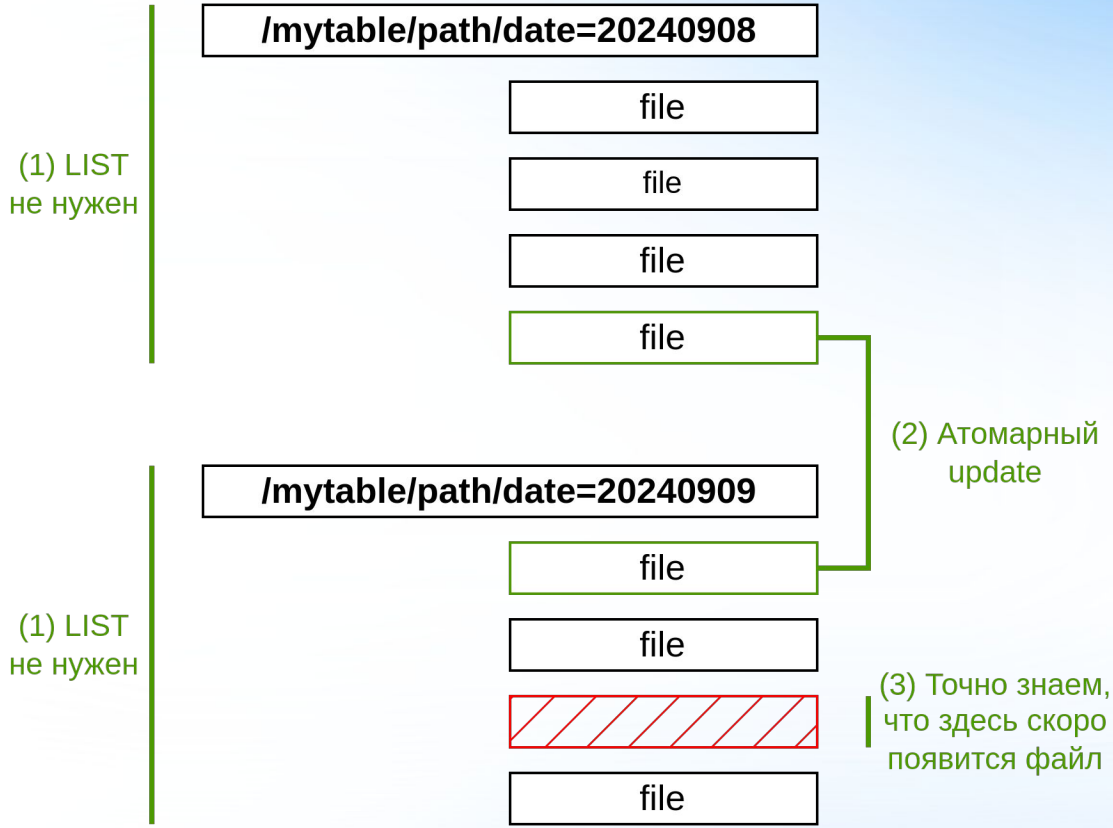
Metadata – ключевой файл метаданных в Apache Iceberg.

Все остальное – это оптимизации для уменьшения копирования:

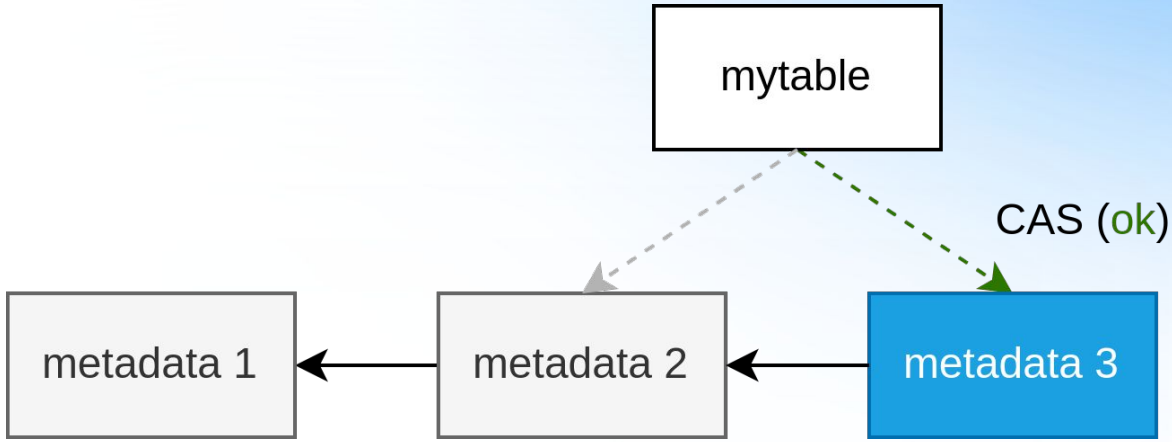
- Manifest list (Avro)
- Manifest (JSON)
- Статистики (Puffin)*

*не показаны на рисунке

Решение проблем Hive



Атомарное изменение метаданных



Для обеспечения линейной истории изменений таблицы необходима возможность атомарного переключения таблицы на новые метаданные. За это в Apache Iceberg отвечает **каталог**

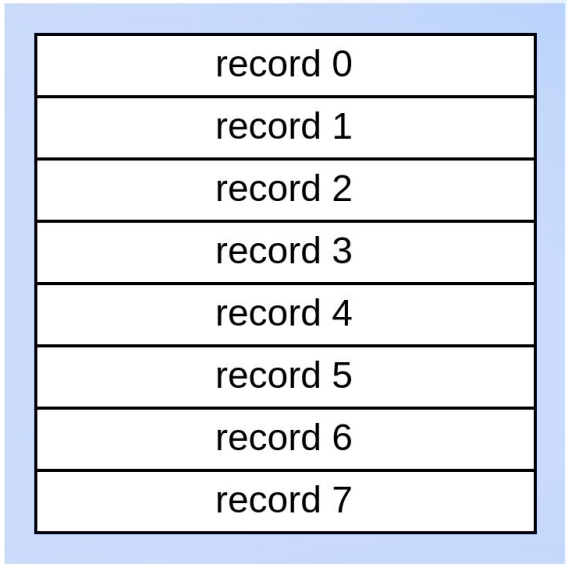
- Семантически: compare_and_set(old_metadata_file_path, new_metadata_file_path)
- Как это сделать?
  - Через storage. Например, атомарный rename в HDFS
  - Через сторонний сервис, который позволяет смоделировать CAS. Например, Postgres :-)

Типы каталогов

- **Hive Catalog** – делегируем атомарность в Hive Metastore
- **JDBC Catalog** – атомарно изменяем метаданные через СУБД
- **REST Catalog** – абстрактный сервер, который принимает запросы на изменение метаданных по фиксированному REST протоколу. Реализацию сервера вы должны выбрать сами
  - Примеры: Tabular, Polaris, CedrusData
- Другие:
  - Hadoop Catalog (атомарный rename)
  - Glue Catalog
  - Snowflake Catalog
  - ...

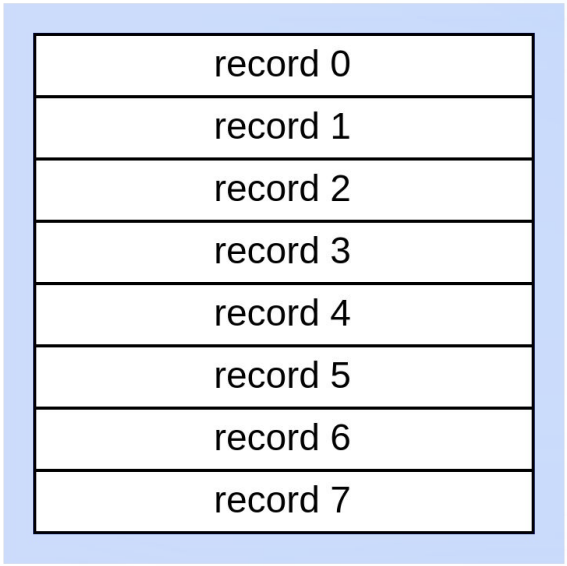
Данные: INSERT/CTAS

```
INSERT INTO mytable
VALUES ...
```



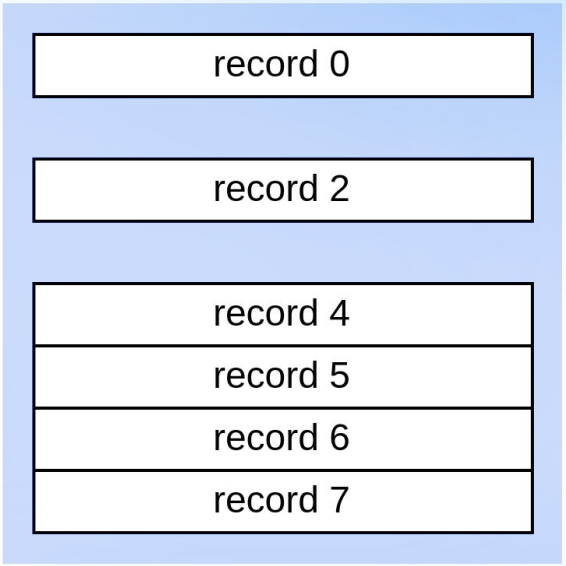
Данные: COPY-ON-WRITE, DELETE

```
INSERT INTO mytable
VALUES ...
```



Data File 1

```
DELETE FROM mytable
WHERE x=5 AND y=10
```

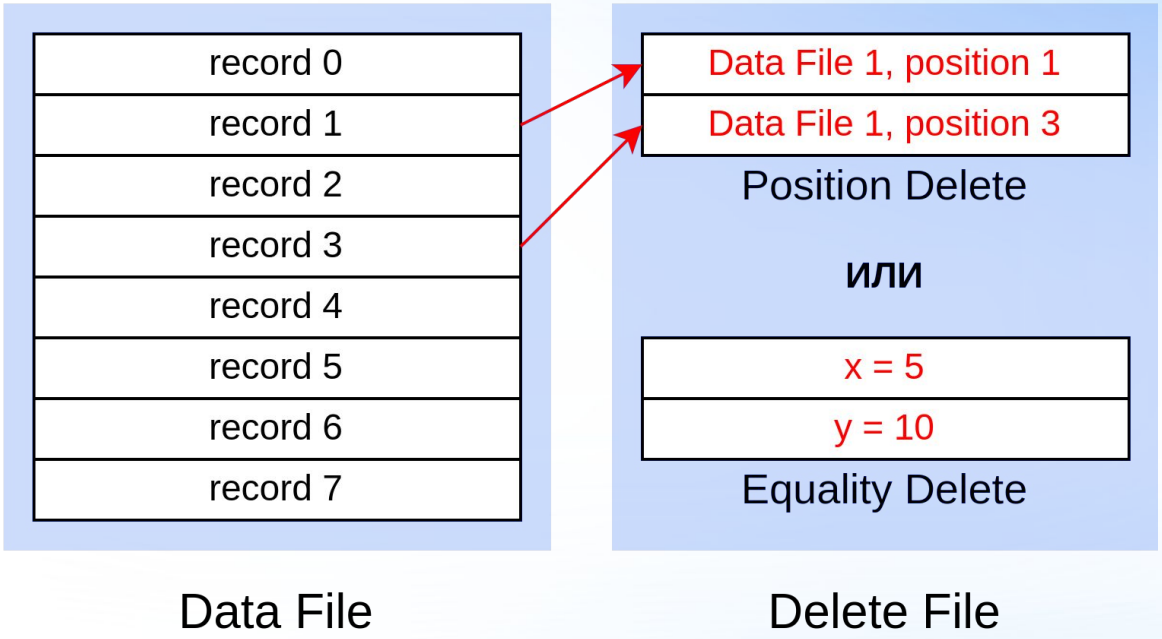


Data File 2

Данные: MERGE-ON-READ, DELETE

```
INSERT INTO mytable
VALUES ...
```

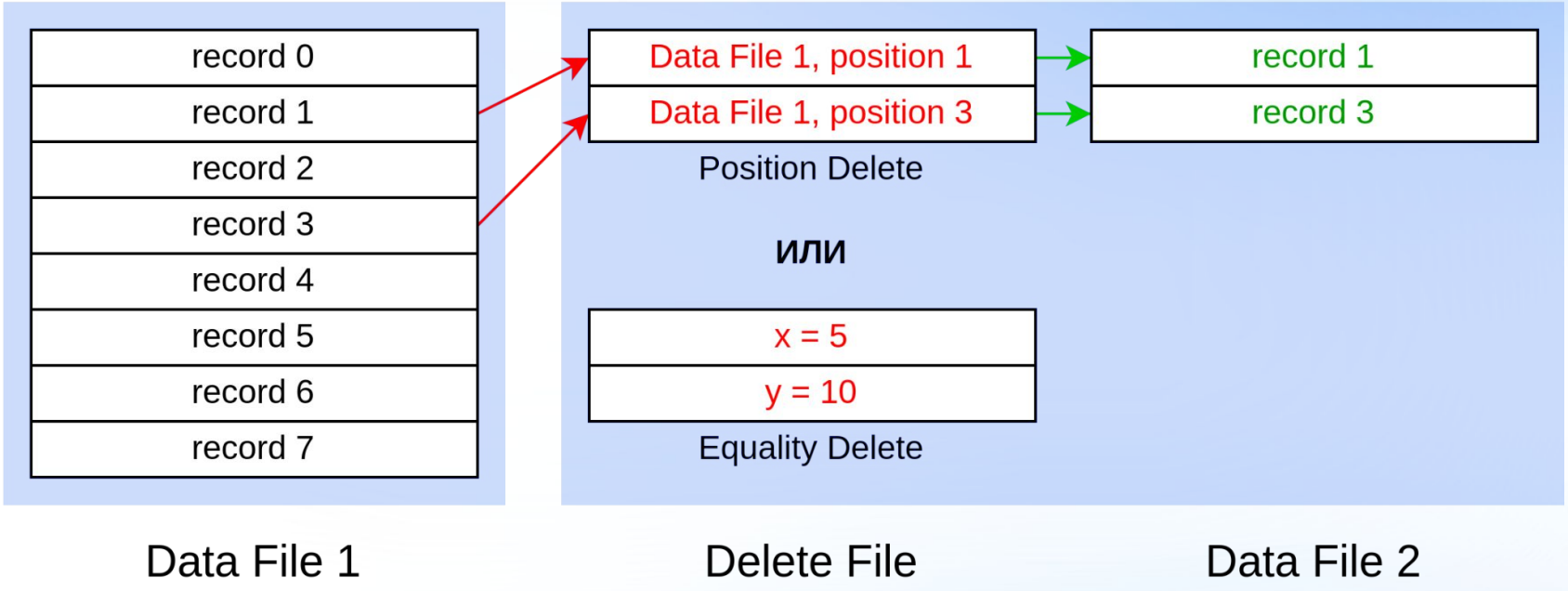
```
DELETE FROM mytable
WHERE x=5 AND y=10
```



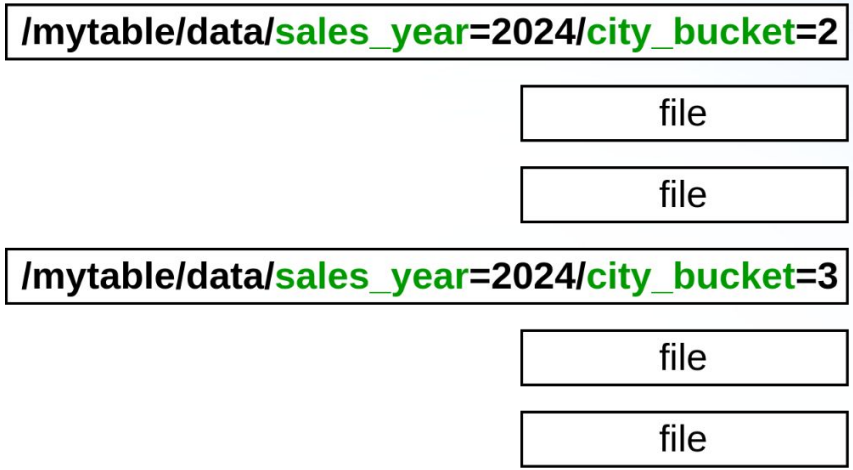
Данные: MERGE-ON-READ, UPDATE

```
INSERT INTO mytable
VALUES ...
```

```
UPDATE mytable
SET z=15
WHERE x=5 AND y=10
```



Данные: партиционирование




```
sales_year  = year(sales_date)

city_bucket = bucket(city, 16)
```

- Partitioning transform – функция, которую нужно применить к одной или нескольким колонкам. Создает новую **виртуальную** колонку.
- Схема партиционирования – список partitioning transform
- Генерализация Hive partitioning/bucketing
  - Hive partition == Iceberg **identity(col)** transform
  - Hive bucketing == Iceberg **bucket(col, N)** transform

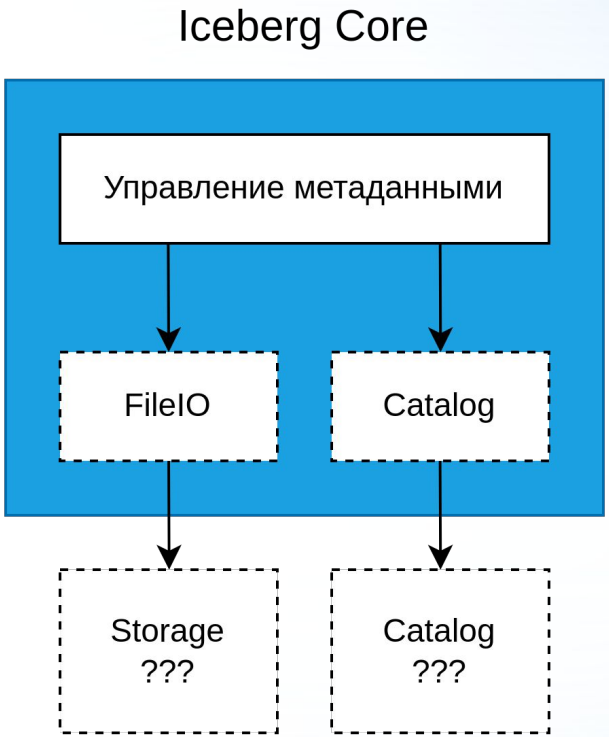
Данные: статистики

- **Точные** статистики для data skipping
  - Min, max, null count
  - Хранятся в manifest files
- **Приблизительные** статистики для cost-based optimization
  - NDV (theta sketches)
  - Хранятся в отдельный бинарных файлах в формате Puffin

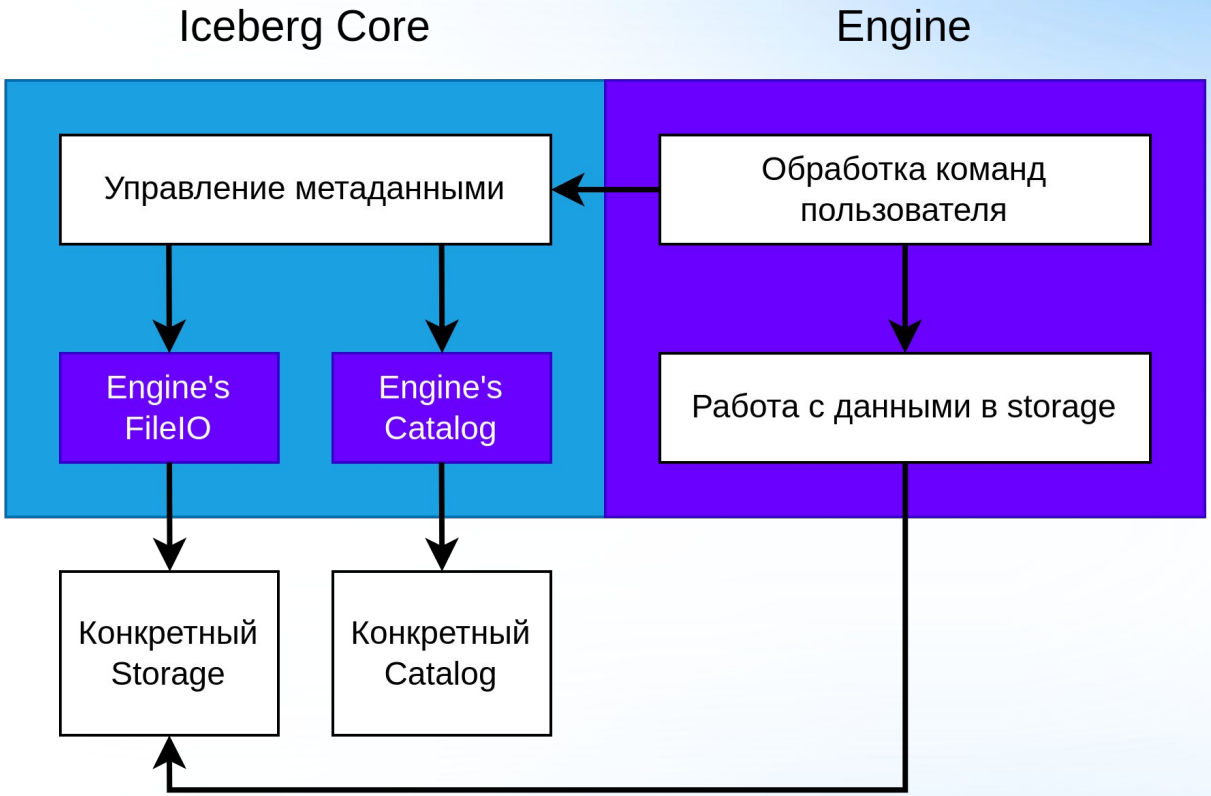
Данные: identifier и sort

- **Identifier fields** – уникальный ключ в рамках таблицы
  - Запрещены NULL значения
  - Моделирует PRIMARY KEY CONSTRAINT
  - Цель: помочь оптимизатору (оценка кардинальностей, aggregate elimination, и т. д.)
- **Sort fields** – каким образом данные отсортированы в рамках файла
  - Поддерживает partition transforms. Например, `ORDER BY day(sales_date)`
  - Цель: помочь оптимизатору и движку (ускорение предикатов, merge join, и т.д.)
- Iceberg не проверяет, отвечают ли данные заданным требованиям identifier/sort!

Структура библиотеки



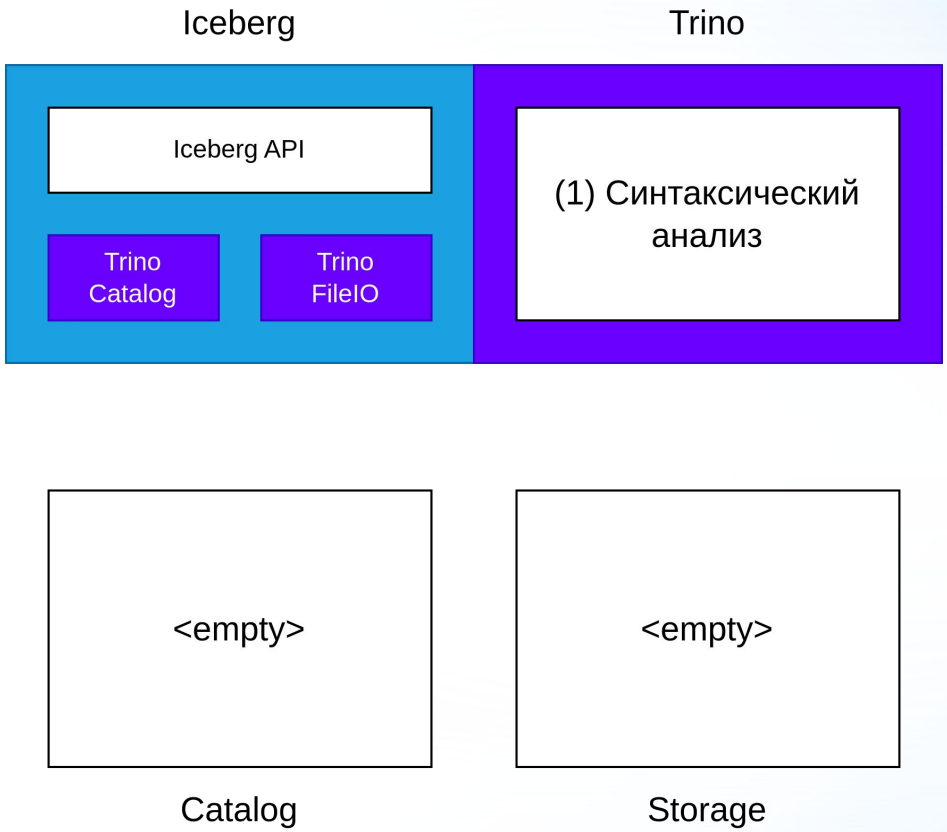
Структура библиотеки



Движки

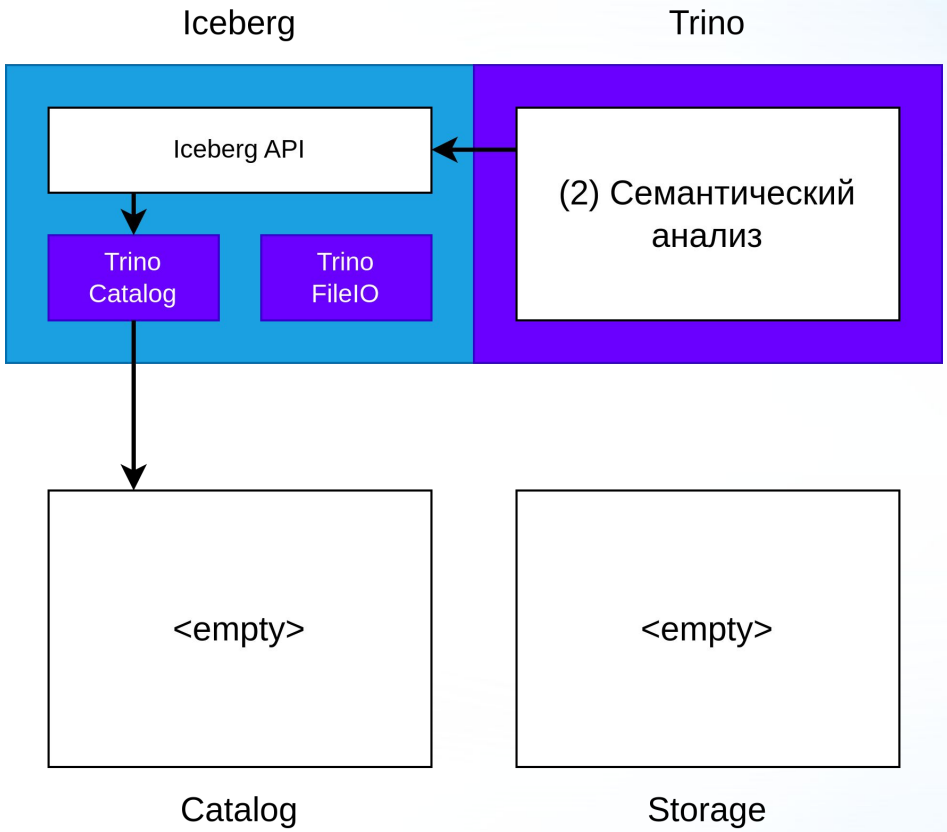
- Apache Spark: https://iceberg.apache.org/docs/1.5.1/spark-getting-started/
- Apache Flink: https://iceberg.apache.org/docs/1.5.1/flink/
- Trino: https://trino.io/docs/current/connector/iceberg.html

Trino: CREATE TABLE AS SELECT



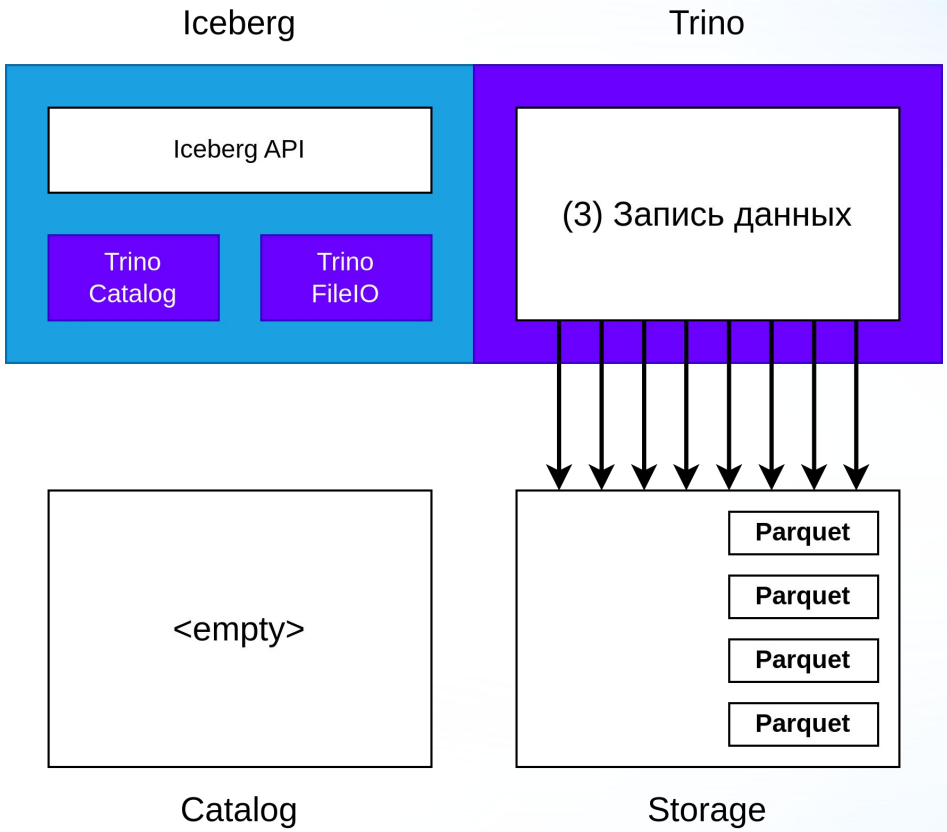
1. Движок парсит запрос

Trino: CREATE TABLE AS SELECT



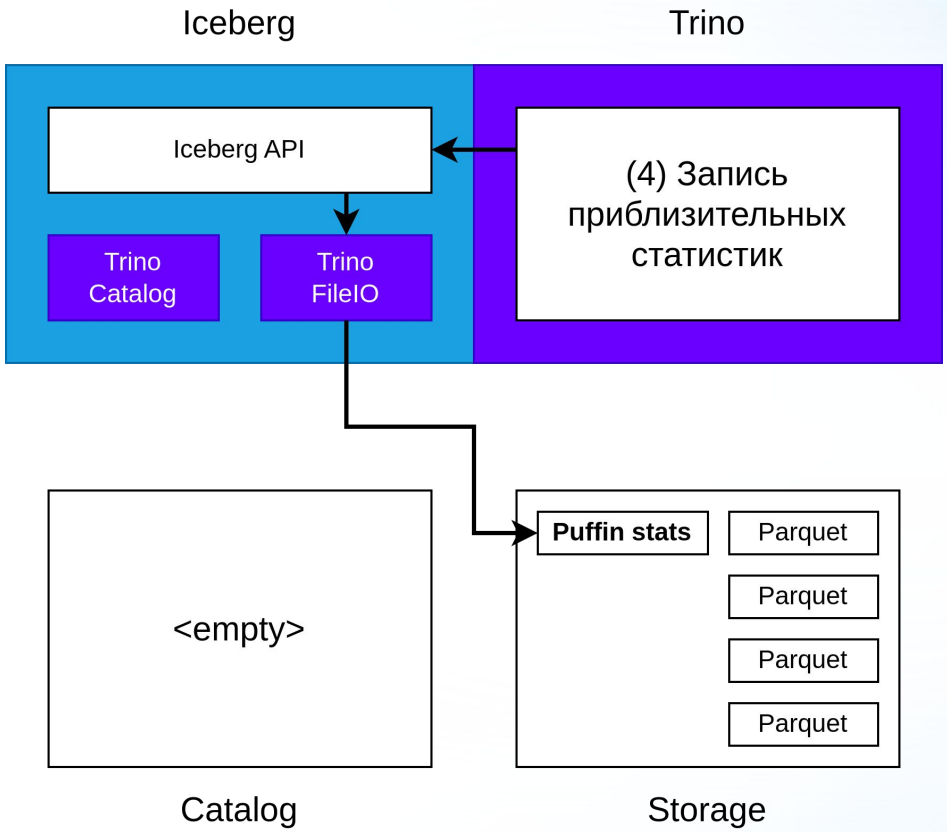
1. Движок парсит запрос
2. Проверка наличия объектов в каталоге

Trino: CREATE TABLE AS SELECT



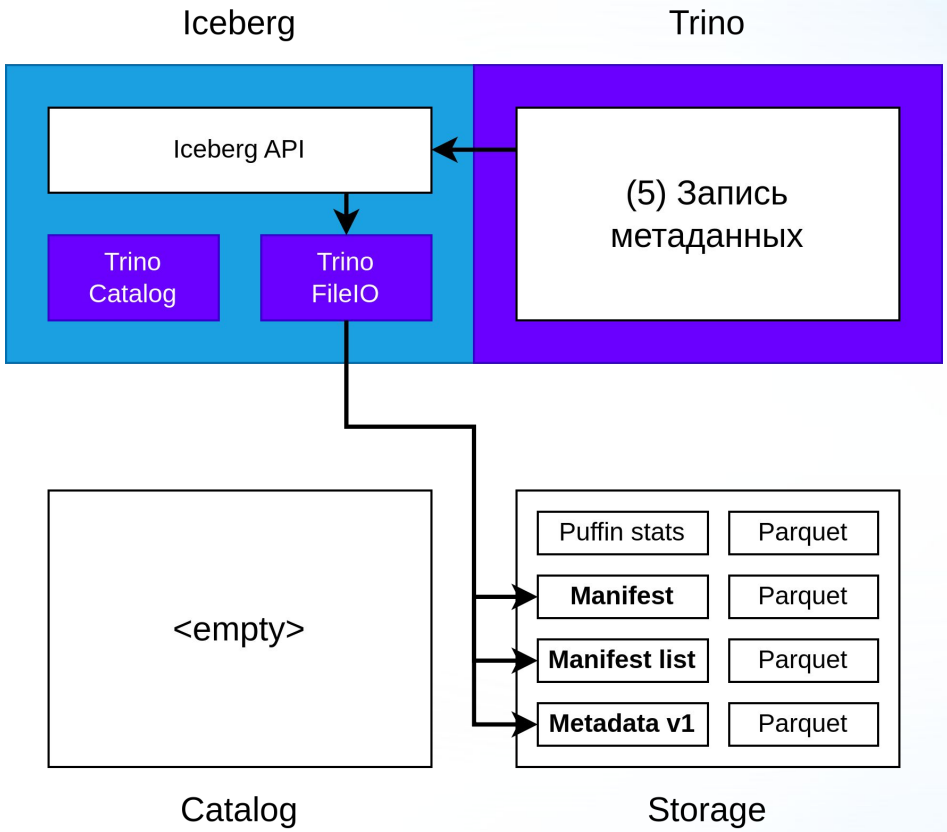
1. Движок парсит запрос
2. Проверка наличия объектов в каталоге
3. Запись данных в целевом формате

Trino: CREATE TABLE AS SELECT



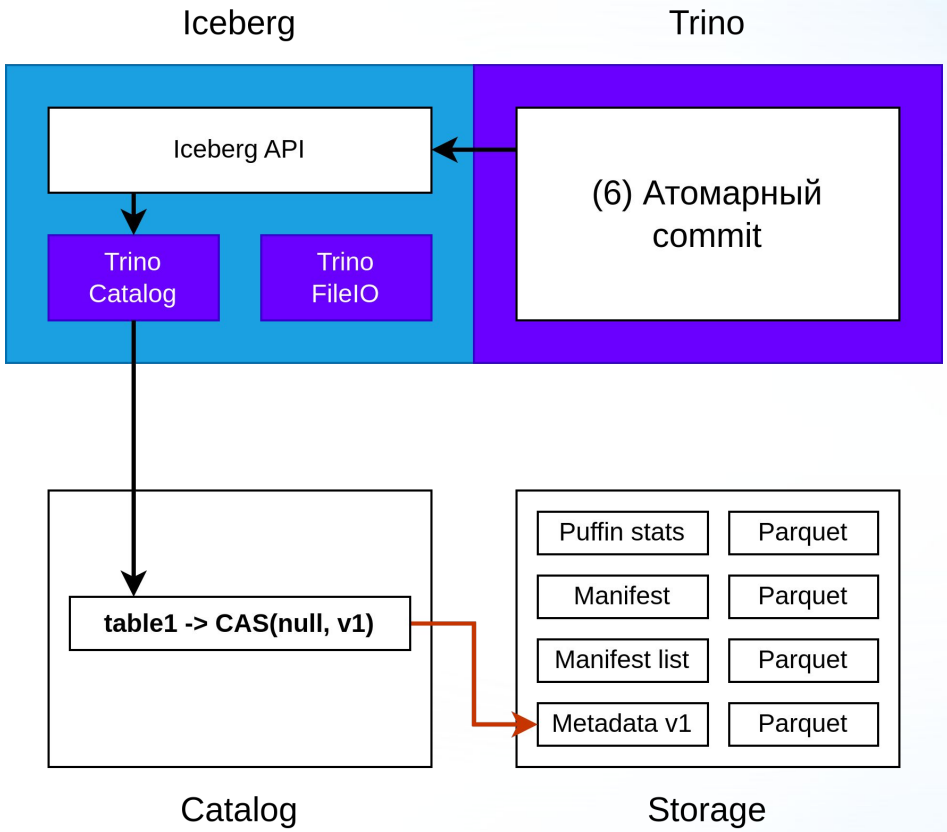
1. Движок парсит запрос
2. Проверка наличия объектов в каталоге
3. Запись данных в целевом формате
4. Запись Puffin статистик

Trino: CREATE TABLE AS SELECT



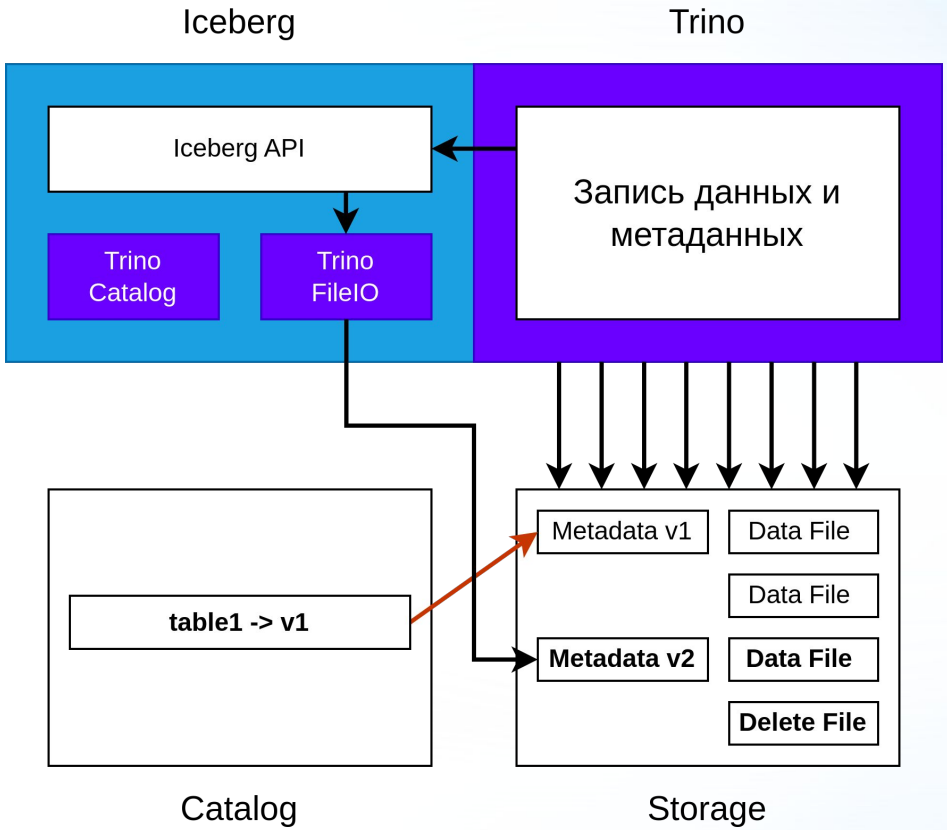
1. Движок парсит запрос
2. Проверка наличия объектов в каталоге
3. Запись данных в целевом формате
4. Запись Puffin статистик
5. Запись метаданных Iceberg

Trino: CREATE TABLE AS SELECT



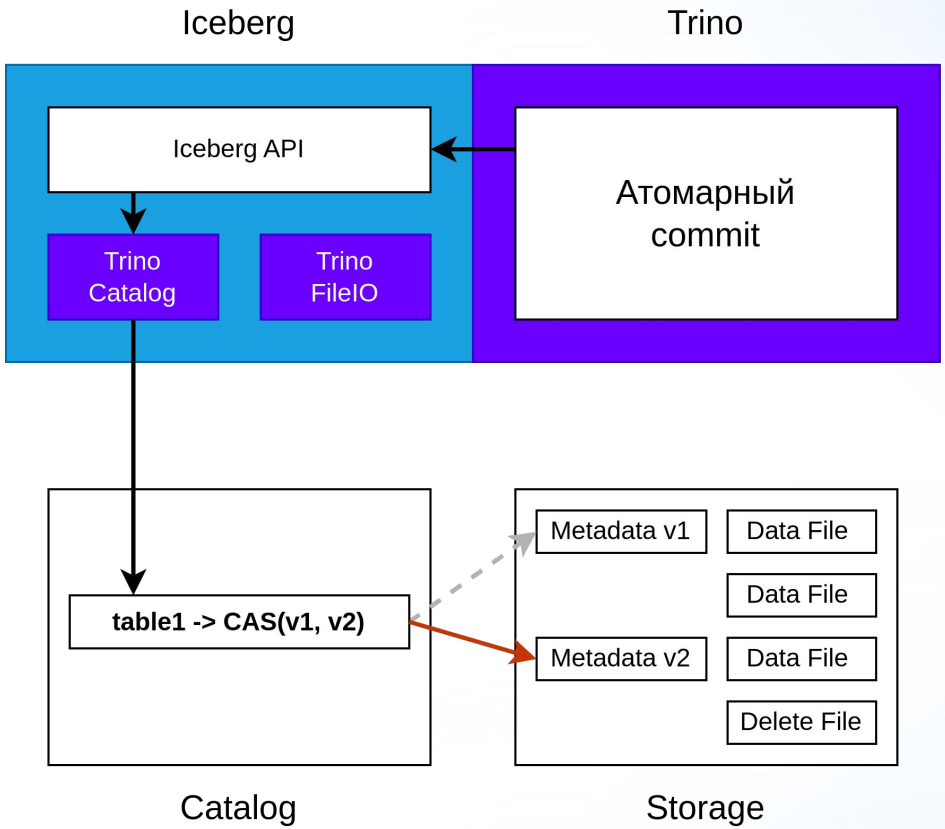
1. Движок парсит запрос
2. Проверка наличия объектов в каталоге
3. Запись данных в целевом формате
4. Запись Puffin статистик
5. Запись метаданных Iceberg
6. Атомарная публикация изменений

Trino: UPDATE/DELETE



1. Синтаксический и семантический анализ
2. Движок и Iceberg создают необходимые файлы

Trino: UPDATE/DELETE



1. Синтаксический и семантический анализ
2. Движок и Iceberg создают необходимые файлы
3. Атомарная публикация изменений

Trino: ключевые возможности

- Поддержка каталогов: REST, JDBC, Hive Metastore, Nessie, Glue, Snowflake
- Поддержка чтения и записи
- Data skipping на основе статистик Iceberg и Parquet/ORC
- Predicate pushdown на основе partitioning transform
- Быстрая фильтрация на основе sort fields
- Time travel
- Maintenance:
  - Удаление устаревших и ненужных файлов
  - Объединение мелких файлов
  - Регистрация имеющихся файлов в каталоге

Trino: чего не хватает

- Оптимизация shuffle на основе partitioning transform
- Data skipping на основе Parquet column index / bloom filter
- Использование sort fields для merge join / streaming aggregation
- Использование identifier fields для нахождения более оптимального плана

Когда хромает сам формат: Trino Materialized Views

Иногда движок умеет делать **больше**, чем позволяет спецификация формата

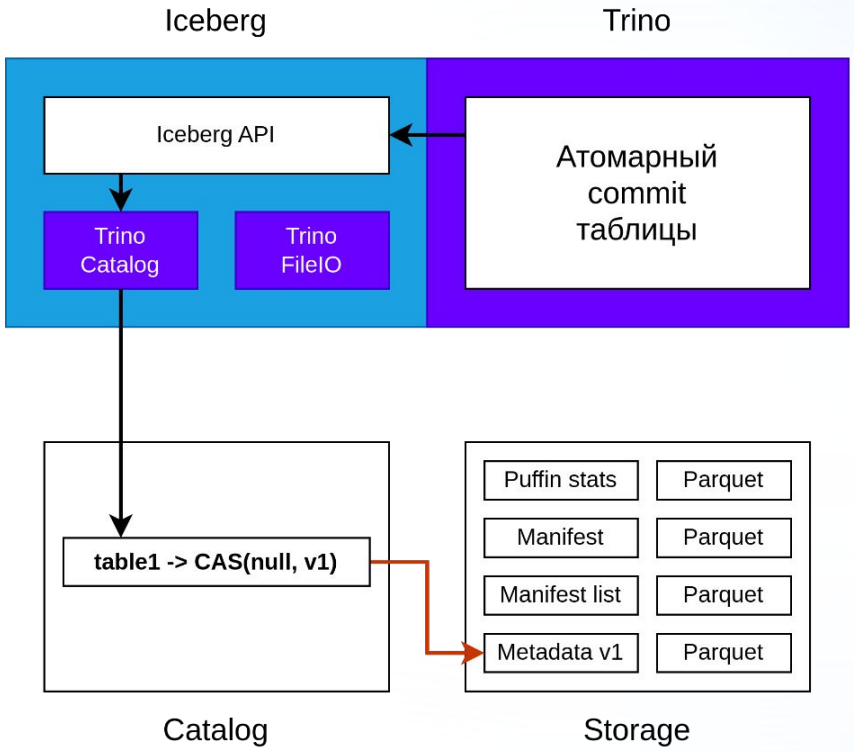
- Trino Materialized views:
  - Позволяют избавиться от повторяющихся вычислений
  - Iceberg – отличный кандидат для транзакционного refresh!
  - Materialized views отсутствуют в спецификации Iceberg!

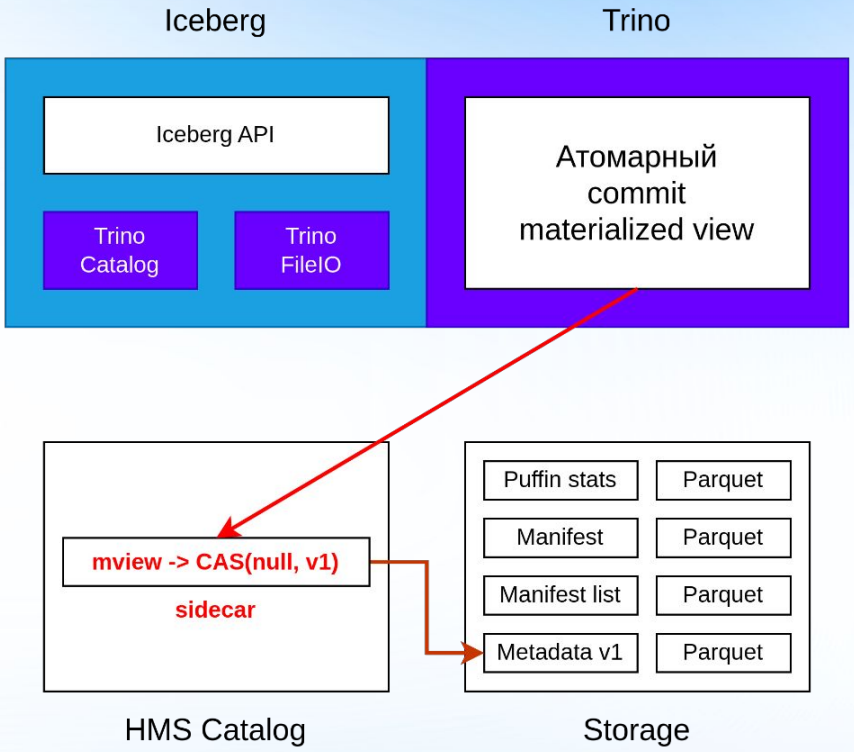
Когда хромает сам формат: Trino Materialized Views

Иногда движок умеет делать **больше**, чем позволяет спецификация формата

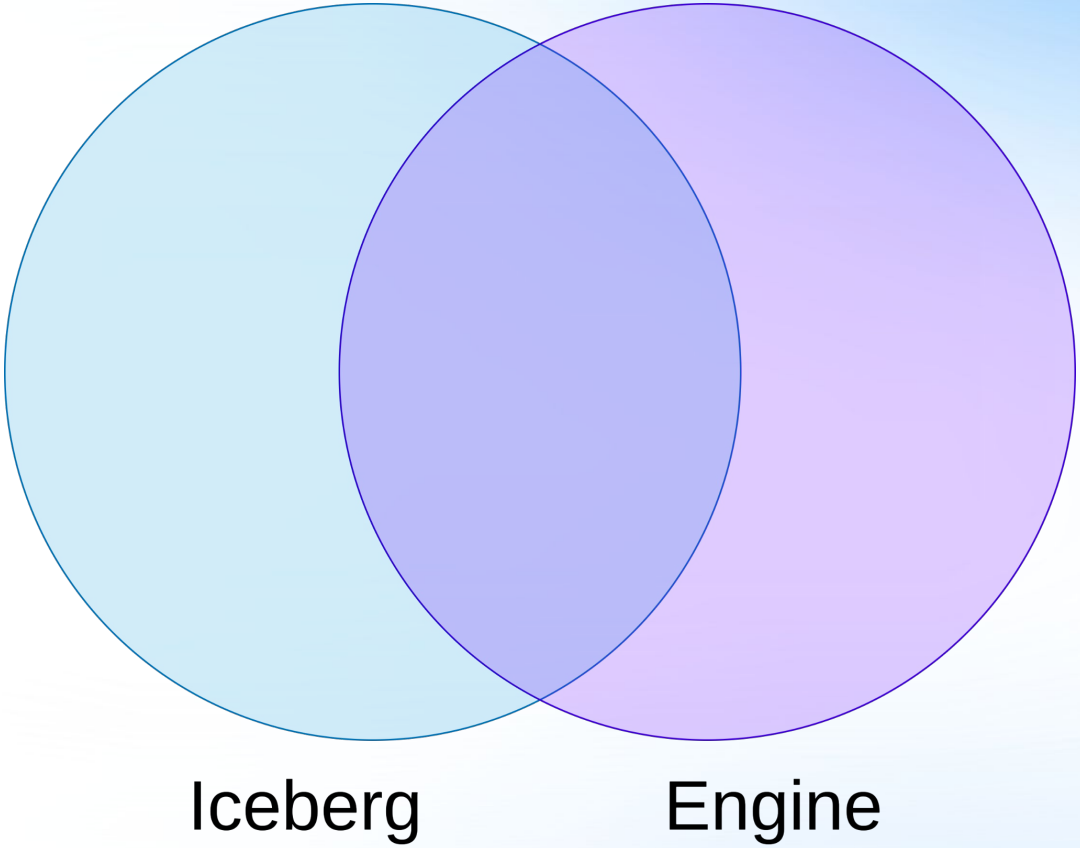
- Trino Materialized views:
  - Позволяют избавиться от повторяющихся вычислений
  - Iceberg – отличный кандидат для транзакционного refresh!
  - Materialized views отсутствуют в спецификации Iceberg!
- Решение: **закостылим**!
  - Каталогом должен являться Hive Metastore
  - Пишем файлы, как если бы это была обычная Iceberg таблица
  - Сохраняем метаданные materialized view в скрытом месте в HMS вместе с дополнительной метаинформацией (например, оригинальный SQL, время последнего refresh), чтобы другие движки не распознавали файлы materialized view как таблицу

Trino Materialized Views





Возможности Iceberg и движка



Заключение

- **Apache Iceberg** – табличный формат, который описывает протокол транзакционных изменений данных в data lake
  - Транзакции в рамках одной таблицы
  - Придерживается парадигмы MVCC: читатели не блокируют писателей
  - Гарантирует атомарность и изоляцию изменений
  - Нужен каталог для атомарного переключения журнала
- **Trino** – движок, который может использовать Iceberg для реальной работы с данными
  - Быстрые параллельные чтения и записи
  - Эффективный data skipping
  - Эффективное использование статистик и метаданных Iceberg для выбора оптимального плана запроса
- **Apache Iceberg + Trino + каталог**
  - Современный стек для аналитической платформы
  - Активное развитие интеграции продуктов

Спасибо

- Телеграм: https://t.me/cedrusdatachat
- YouTube: https://www.youtube.com/@cedrusdata
- Сайт: https://cedrusdata.ru