

# **Modern Storages and Data Warehousing**

## **Week 3 - File formats**

Попов Илья, [i.popov@hse.ru](mailto:i.popov@hse.ru)

**1 - В предыдущих сериях**

# Recap прошлых занятий

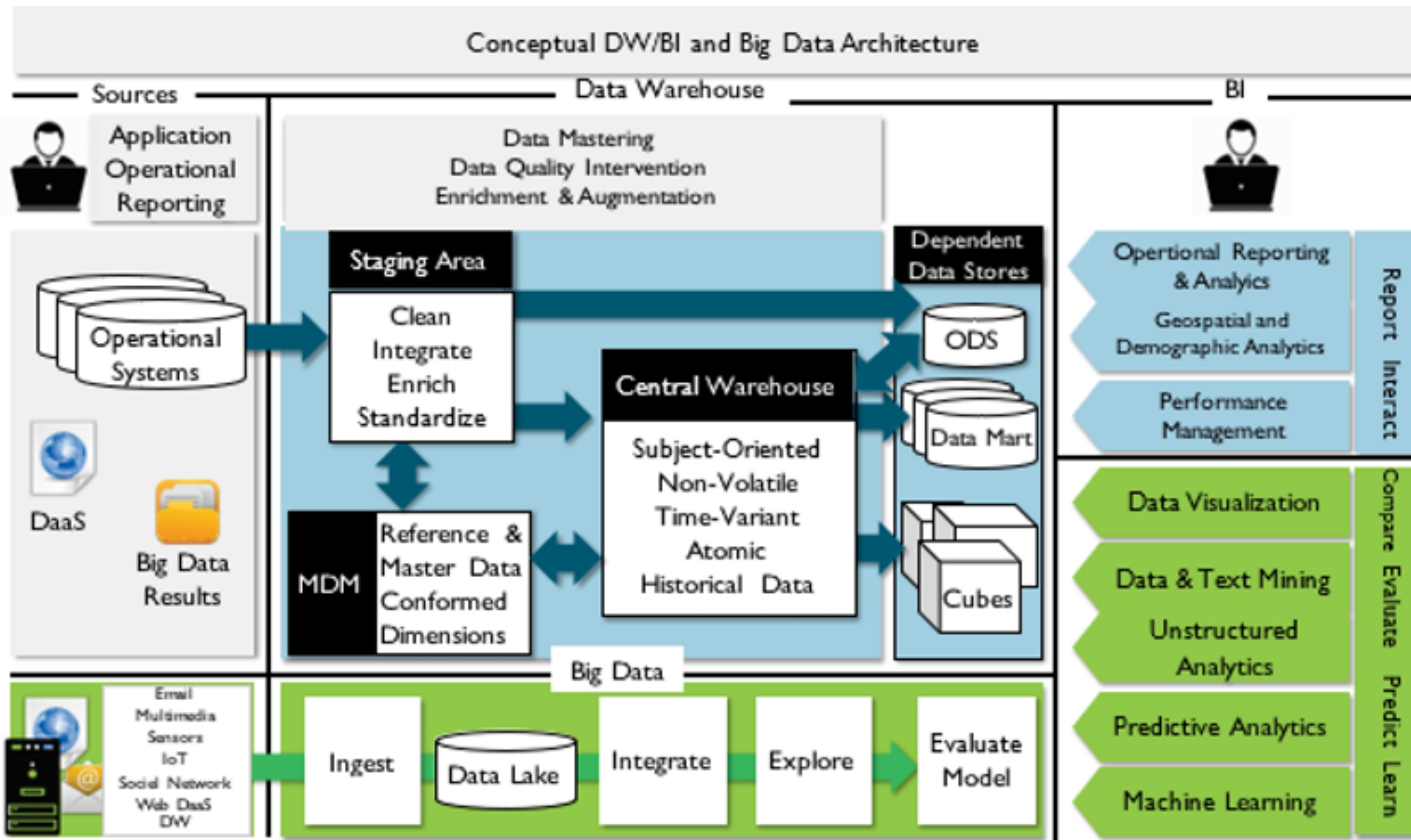
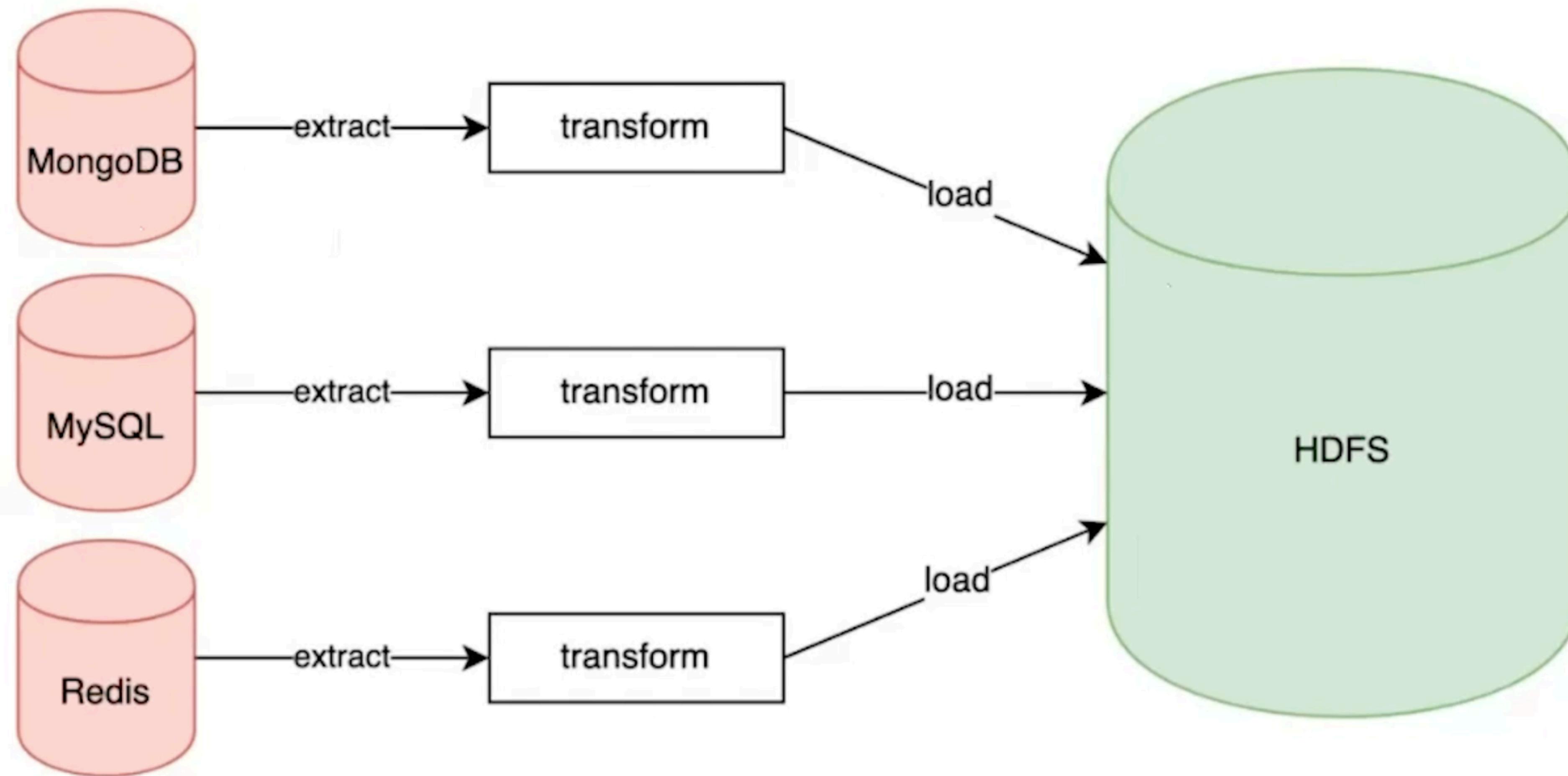


Figure 5: Date Warehouse Concept

# Recap прошлых занятий

- › Мы знаем, где рождаются данные (**OLTP**), почему проблемы **нельзя решить только масштабированием** и как его не убить при работе (**репликация**)
- › Мы знаем, куда сложить данные надолго и дешево (**S3**)
- › Мы знаем, куда сложить большой объем данных для последующей обработки (**HDFS**), как работает Hadoop-экосистема и как нам потом все-таки эти данные обработать (**MapReduce, Spark**)

# То, что мы знаем сейчас, выглядит примерно так



# Ресар прошлых занятий

- › Мы знаем, где рождаются данные (**OLTP**), почему проблемы **нельзя решить только масштабированием** и как его не убить при работе (**репликация**)
- › Мы знаем, куда сложить данные надолго и дешево (**S3**)
- › Мы знаем, куда сложить большой объем данных для последующей обработки (**HDFS**), как работает Hadoop-экосистема и как нам потом все-таки эти данные обработать (**MapReduce, Spark**)

Проблема:

- › Эту штуку мы хотим ускорить
- › Данные с прода можем возить не только мы в удобном виде, но и внешние подрядчики в неудобном виде

# **2 - Форматы хранения данных**

# Самый простой кейс

```
<?xml version="1.0"?>
<catalog>
    <book id="bk101">
        <author>Gambardella, Matthew</author>
        <title>XML Developer's Guide</title>
        <genre>Computer</genre>
        <price>44.95</price>
        <publish_date>2000-10-01</publish_date>
        <description>An in-depth look at creating applications
            with XML.</description>
    </book>
    <book id="bk102">
        <author>Ralls, Kim</author>
        <title>Midnight Rain</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
        <publish_date>2000-12-16</publish_date>
        <description>A former architect battles corporate zombies,
            an evil sorceress, and her own childhood to become queen
            of the world.</description>
    </book>
    ...

```

```
spark.read
    .option("rowTag", "book")
    .option("rootTag", "books")
    .format("com.databricks.spark.xml")
    .load(path)
    .withColumn("price", cast("price"))
    .agg(f.max("price"))
    .show()
```

# Самый простой кейс

```
<?xml version="1.0"?>
<catalog>
    <book id="bk101">
        <author>Gambardella, Matthew</author>
        <title>XML Developer's Guide</title>
        <genre>Computer</genre>
        <price>44.95</price>
        <publish_date>2000-10-01</publish_date>
        <description>An in-depth look at creating applications
            with XML.</description>
    </book>
    <book id="bk102">
        <author>Ralls, Kim</author>
        <title>Midnight Rain</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
        <publish_date>2000-12-16</publish_date>
        <description>A former architect battles corporate zombies,
            an evil sorceress, and her own childhood to become queen
            of the world.</description>
    </book>
    ...

```

```
spark.read
    .option("rowTag", "book")
    .option("rootTag", "books")
    .format("com.databricks.spark.xml")
    .load(path)
    .withColumn("price", cast("price"))
    .agg(f.max("price"))
    .show()
```

**Мысли?**  
**Бро/не бро?**  
**Будем так делать?**

# Какие форматы бывают вообще?

- › Специфичные для языка:
  - python — Pickle**
  - Java — Serializable, Kryo**
  - Ruby — Mashral**

# Какие форматы бывают вообще?

› Специфичные для языка:

**python — Pickle**

**Java — Serializable, Kryo**

**Ruby — Mashral**

**Мысли?  
Бро/не бро?  
Будем так делать?**

# Какие форматы бывают вообще?

- › Специфичные для языка:
  - python — Pickle**
  - Java — Serializable, Kryo**
  - Ruby — Mashral**
- › Нельзя десериализовать в другом языке
- › Плохо с версионированием
- › Плохо с производительностью

# Какие форматы бывают вообще?

〉 Текстовые форматы:

**JSON**

**XML**

**CSV**

**Yaml**

# Какие форматы бывают вообще?

〉 Текстовые форматы:

**JSON**

**XML**

**CSV**

**Yaml**

**Мысли?  
Бро/не бро?  
Будем так делать?**

# Какие форматы бывают вообще?

- › Текстовые форматы:
  - JSON**
  - XML**
  - CSV**
  - Yaml**
- › Проблемы с типами данных (различия в записи int/long/float/string)
- › Проблемы с экранированием
- › Избыточность
- › Неэффективное представление

# Какие форматы бывают вообще?

› Текстовые форматы:

**JSON**

**XML**

**CSV**

**Yaml**

› Решение проблемы - давайте просто рядом положим схему данных



# Что такое схема данных?

```
{  
    "type": "record",  
    "namespace": "com.example",  
    "name": "Person",  
    "fields": [  
        { "name": "firstName", "type": "string" },  
        { "name": "lastName", "type": "string" },  
        { "name": "age", "type": "int" }  
    ]  
}
```

# Что такое схема данных?

```
{ "type": "record", "namespace": "com.example", "name": "Person", "fields": [ { "name": "firstName", "type": "string"}, { "name": "lastName", "type": "string"}, { "name": "age", "type": "int"} ] } { "type": "record", "namespace": "com.example", "name": "Person", "fields": [ { "name": "firstName", "type": "string"}, { "name": "lastName", "type": "string"}, { "name": "age", "type": "int"}, { "name": "interests", "type": {"type": "array", "items": "string"}}, { "name": "address", "type": { "type": "record", "name": "mailing_address", "fields": [ {"name": "street", "type": "string"}, {"name": "city", "type": "string"}, {"name": "country", "type": "string", "default": "NONE"}, {"name": "zip", "type": "string", "default": "NONE"} ]}, "default": {} ] }
```

# Эволюция схемы

- › Со временем мы захотим менять/изменять формат, как наши данные хранятся
- › Эволюция схемы - это процесс её изменения во времени
- › О чём нужно думать:
  - Обратная совместимость - новый код может прочитать старую схему**
  - Прямая совместимость - старый код может прочитать новую схему**

# Эволюция схемы

- › Со временем мы захотим менять/изменять формат, как наши данные хранятся
- › Эволюция схемы - это процесс её изменения во времени
- › О чём нужно думать:
  - Обратная совместимость - новый код может прочитать старую схему**
  - Прямая совместимость - старый код может прочитать новую схему**
- › В разных форматах совместимость реализуется по-разному

# Apache Avro

- › Релиз в 2009 как часть Hadoop-экосистемы
- › Поддержка Spark из коробки
- › Библиотеки для сериализации/десериализации на любом языке
- › Использует схему данных и прямую/обратную совместимость

# Пример

- › Пример для MessagePack - один из видов JSON с бинарным представлением
- › Стерилизованное сообщение:

```
{  "userName": "Martin",  "favoriteNumber": 1337,  "interests": [    "daydreaming", "hacking"  ]}
```
- › Занимает 66 байт

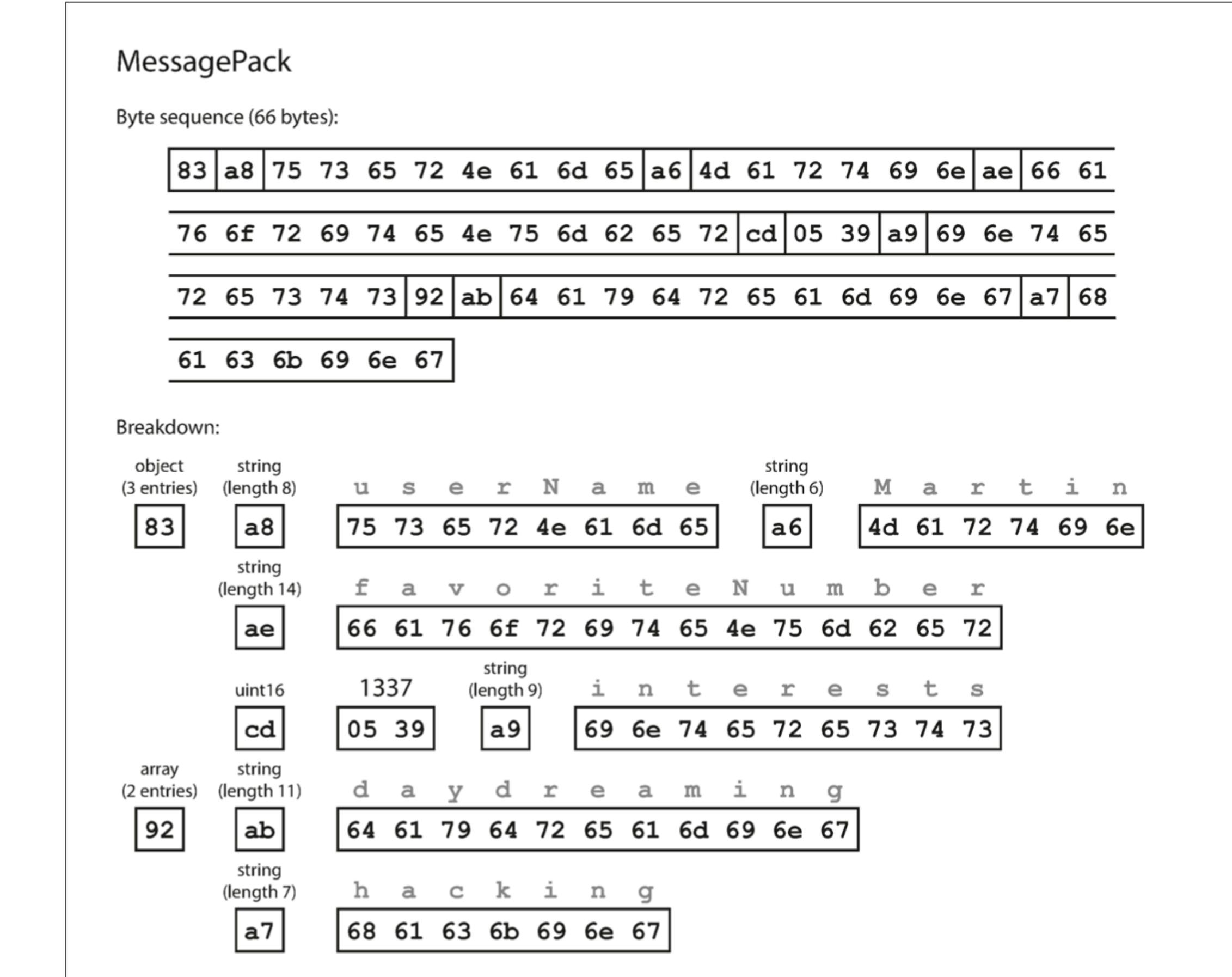


Figure 4-1. Example record ([Example 4-1](#)) encoded using MessagePack.

# Пример

- › Пример для Apache Avro
- › Стерилизованное сообщение:

```
{  
    "userName": "Martin",  
    "favoriteNumber": 1337,  
    "interests": [  
        "daydreaming", "hacking"  
    ]  
}
```
- › Занимает 32 байта

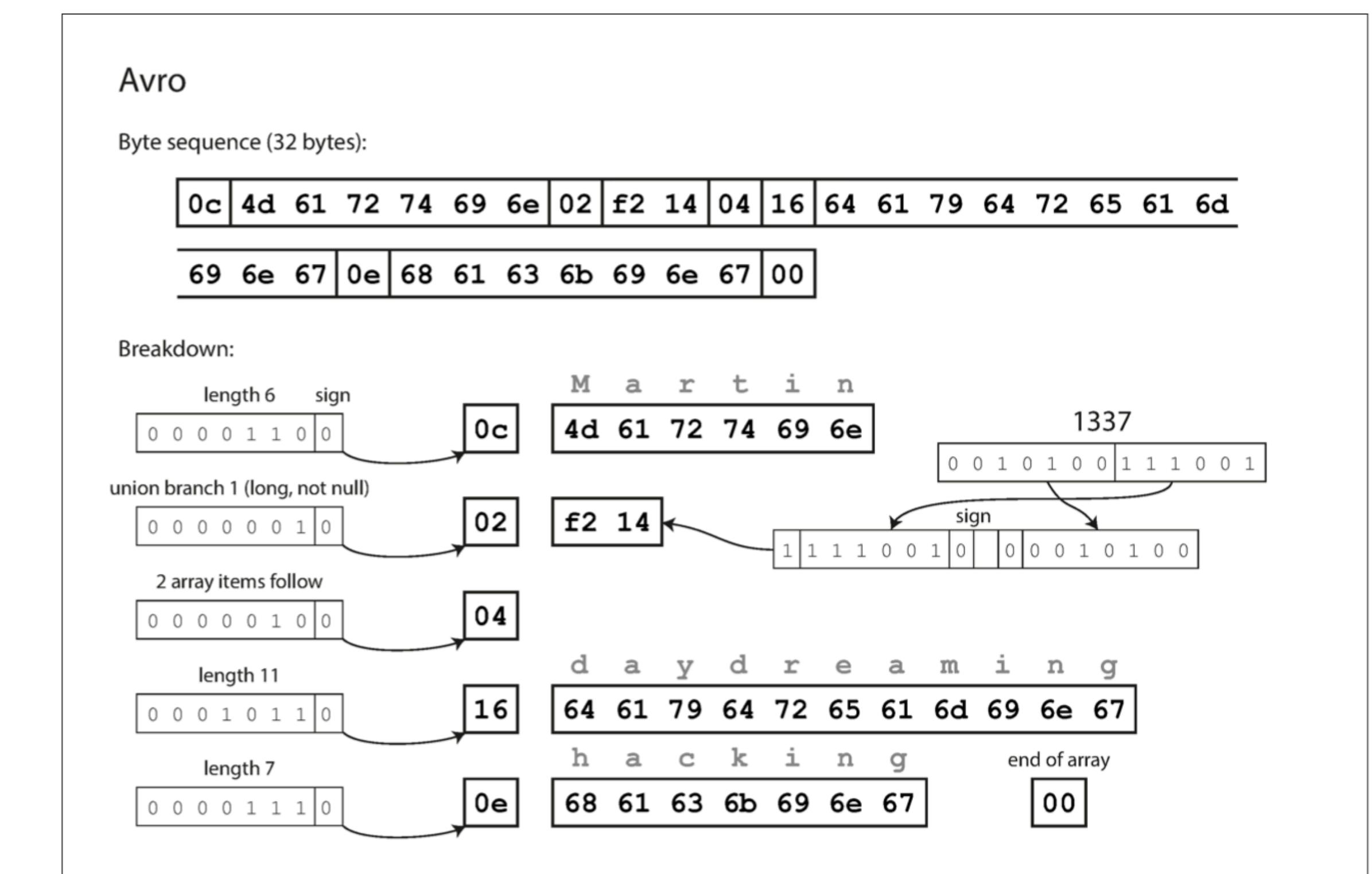


Figure 4-5. Example record encoded using Avro.

# Apache Avro

- › Схема хранится рядом с кодом в бинарном или json-формате
- › Схема хранится в данных в начале arvo-файла
- › Парсер смотрит на эти 2 схемы и понимает, как читать данные
  
- › Можно удалять колонки (если задано default-значение)
- › Можно добавлять колонки (если задано default-значение)
- › Можно менять колонки местами
- › Можно менять тип данных, если типы конвертируются друг в друга



**Фиксировать схему данных -  
это хорошо**

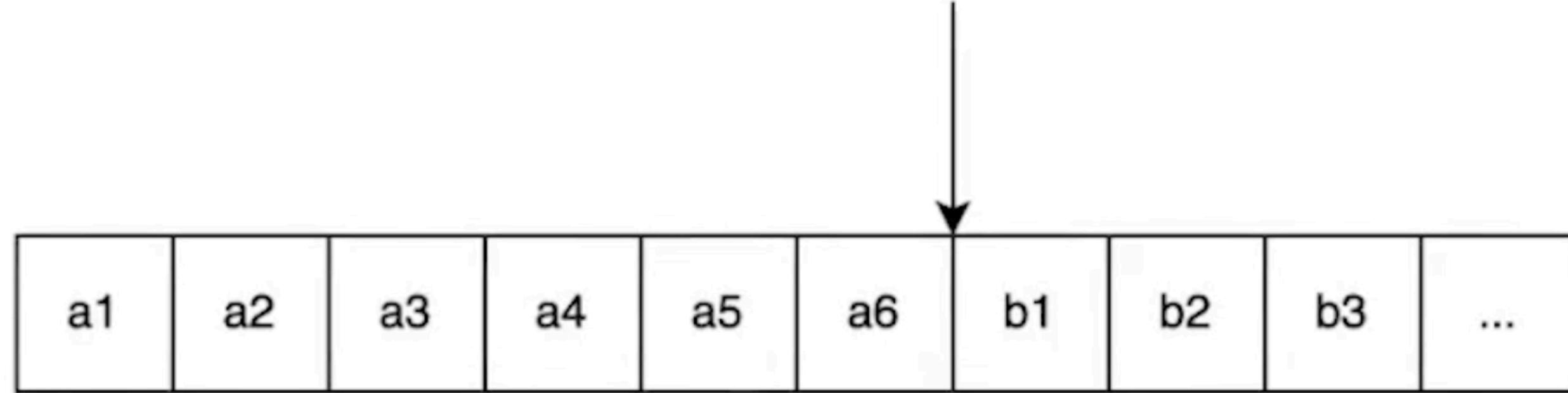
# Построчные форматы

	a	b	c	d
1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5
6	a6	b6	c6	d6

a1	b1	c1	d1	a2	b2	c2	d2	...
----	----	----	----	----	----	----	----	-----

# Поколоночные форматы

	a	b	c	d
1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5
6	a6	b6	c6	d6



# Поколоночные форматы

Плюсы:

- › Можно прочитать только одну или несколько колонок
- › Можно оптимизировать хранение - применить кодирование (dictionary encoding) или сжатие (RLE)
- › Можно использовать векторизированные вычисления

# Поколоночные форматы

Плюсы:

- › Можно прочитать только одну или несколько колонок
- › Можно оптимизировать хранение - применить кодирование (dictionary encoding) или сжатие (RLE)
- › Можно использовать векторизированные вычисления

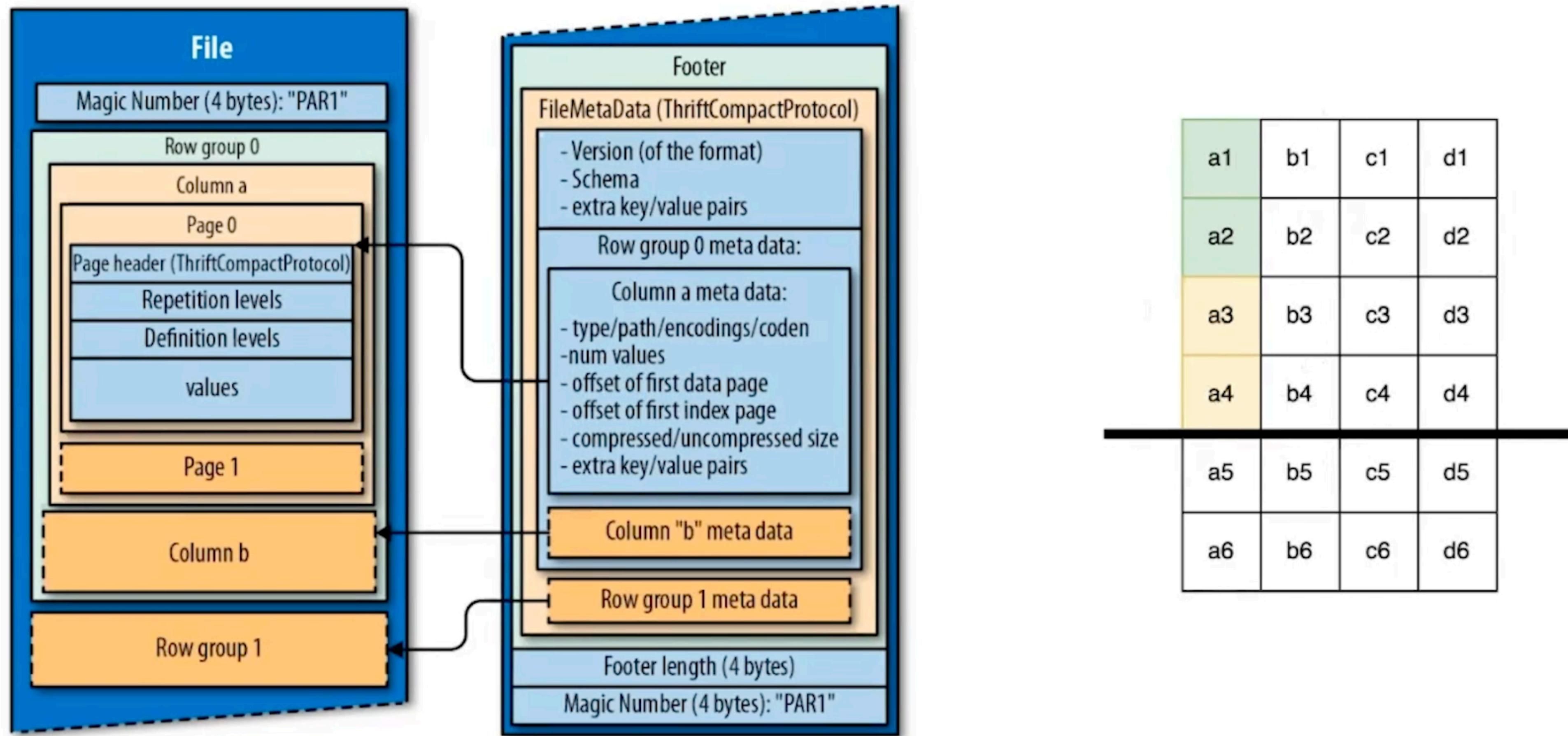
Важно:

- › Эффективность сжатия зависит от сортировки

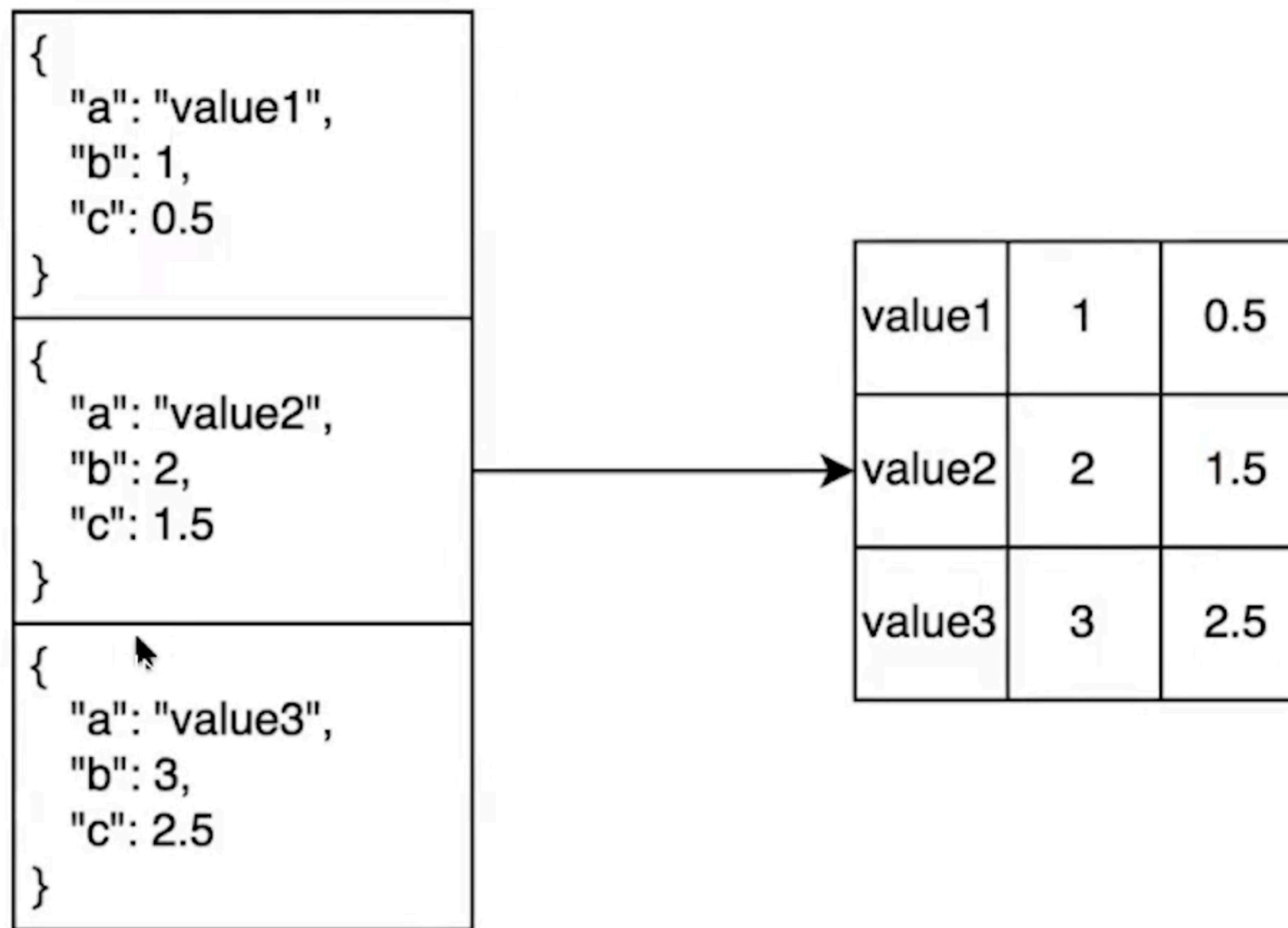
# Apache Parquet

- › Релиз в 2009 как часть Hadoop-экосистемы
- › Поддержка Spark из коробки
- › Библиотеки для сериализации/десериализации на любом языке
- › Использует схему данных и прямую/обратную совместимость
  
- › Использует метаданные и pushdown predicate

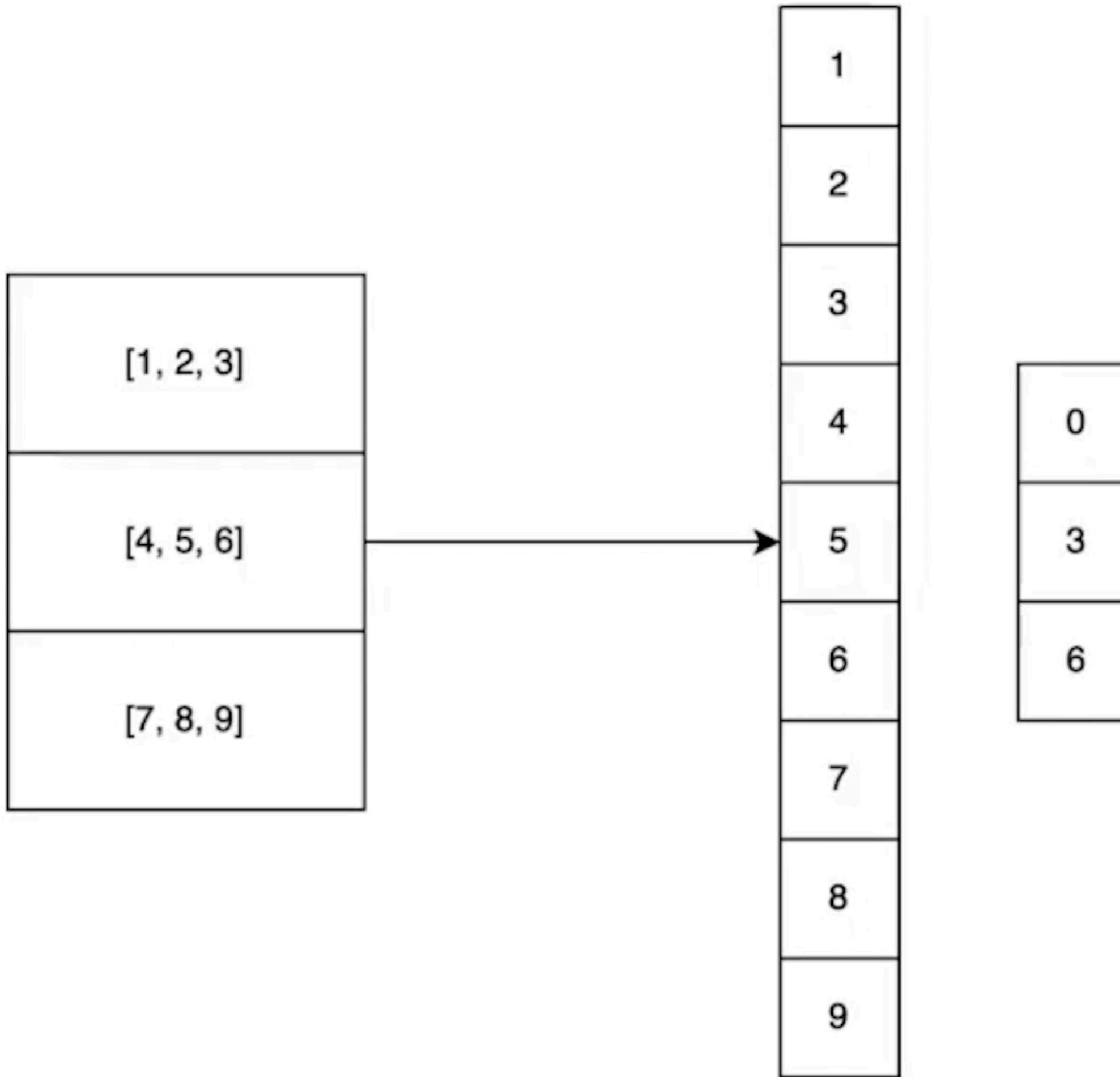
# Apache Parquet



# Сложные типы



# Сложные типы





Колоночные форматы - это  
хорошо для Big Data

# One more thing - Partition Discovery

```
path
└ to
  └ table
    └ gender=male
      └ ...
      └ country=US
        └ data.parquet
      └ country=CN
        └ data.parquet
      └ ...
    └ gender=female
      └ ...
      └ country=US
        └ data.parquet
      └ country=CN
        └ data.parquet
      └ ...
```

```
root
|--- name: string (nullable = true)
|--- age: long (nullable = true)
|--- gender: string (nullable = true)
|--- country: string (nullable = true)
```

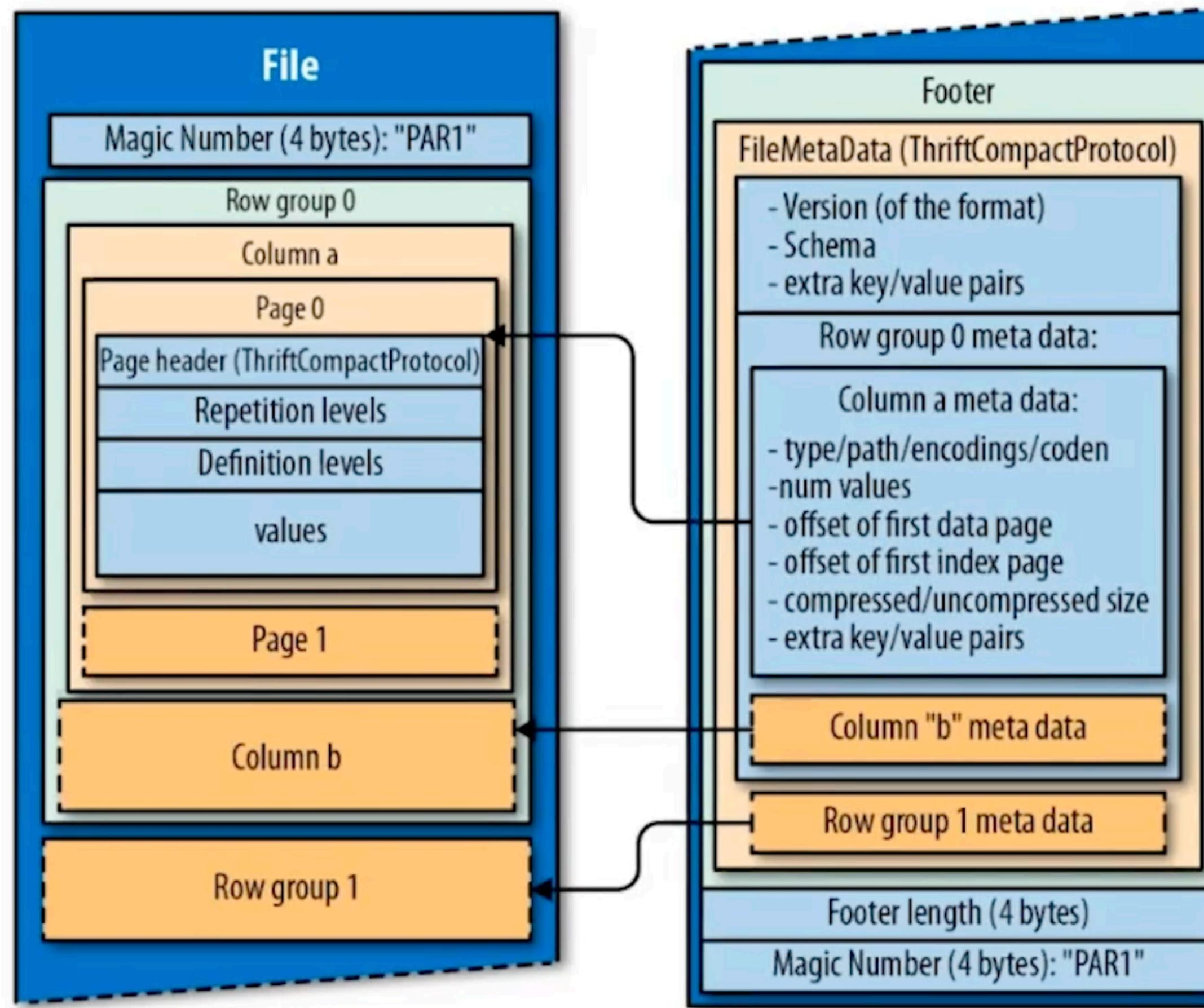
**3 - Iceberg**

# Partition Discovery

```
path
└ to
  └ table
    ├── gender=male
    │   ├── ...
    │   ├── country=US
    │   │   └── data.parquet
    │   ├── country=CN
    │   │   └── data.parquet
    │   └── ...
    └── gender=female
        ├── ...
        ├── country=US
        │   └── data.parquet
        ├── country=CN
        │   └── data.parquet
        └── ...
```

```
root
|--- name: string (nullable = true)
|--- age: long (nullable = true)
|--- gender: string (nullable = true)
|--- country: string (nullable = true)
```

# Apache Parquet



a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3
a4	b4	c4	d4
a5	b5	c5	d5
a6	b6	c6	d6

# Apache Parquet

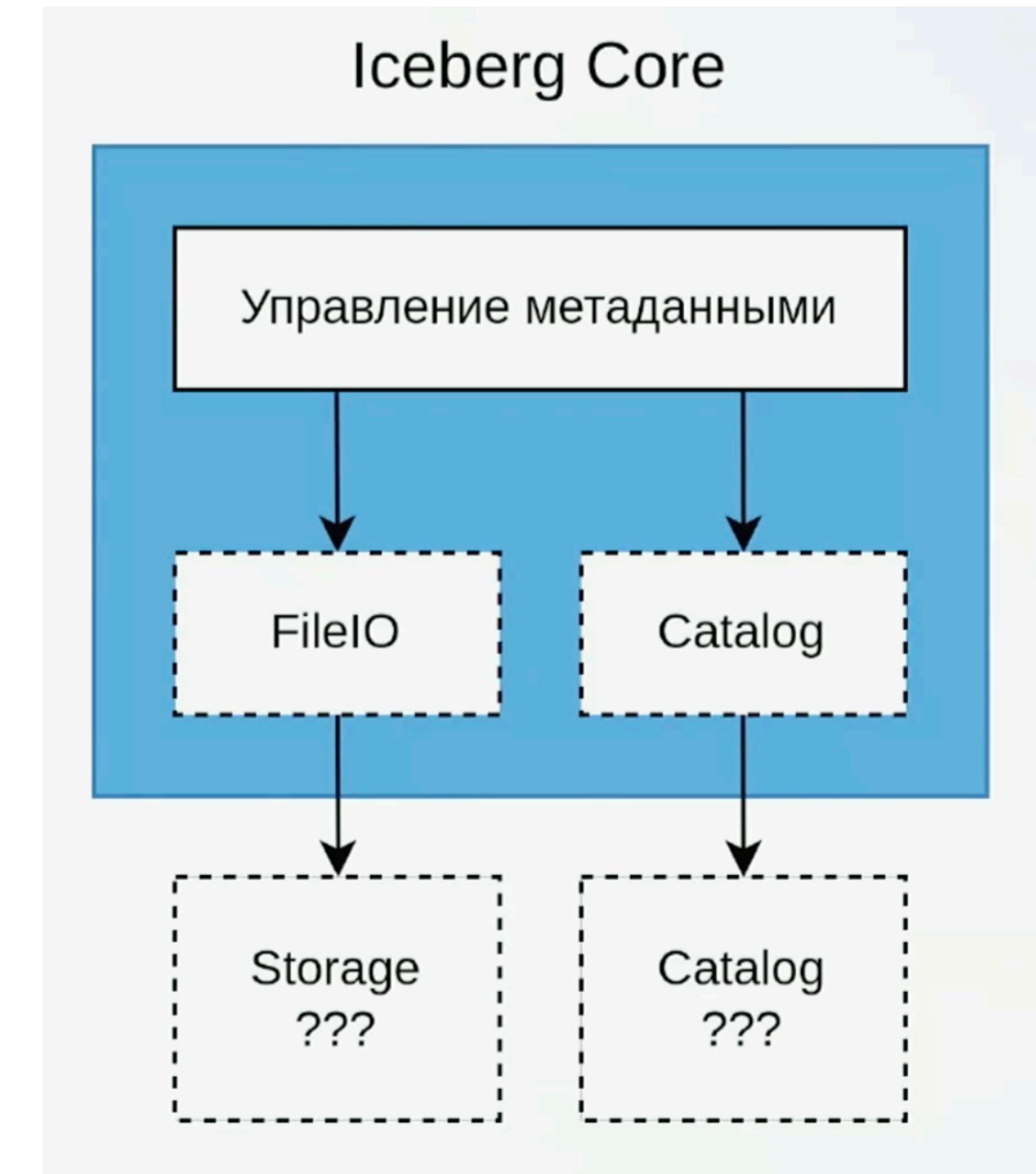
- › Для того, чтобы прочитать большой файл, нужно:
- › Получить список всех его партиций
- › Прочитать метаинформацию/статистики из этих партиций
- › Если в момент чтения произойдет запись/изменение, то будет конфликт
- › В некоторых видах хранилищ операция `list` может быть очень затратной (например, в S3)

# Apache Parquet

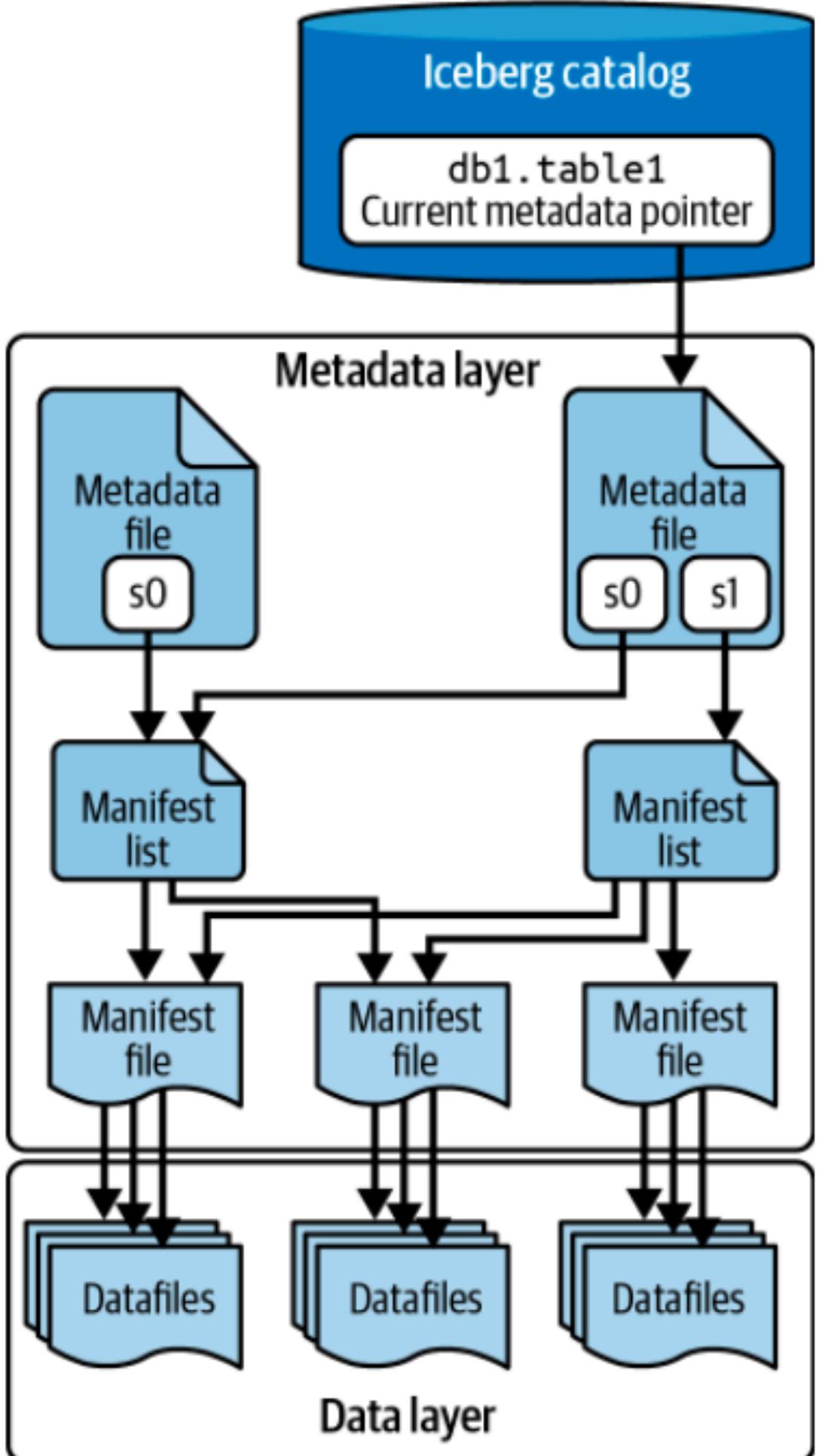
- › Для того, чтобы прочитать большой файл, нужно:
- › Получить список всех его партиций
- › Прочитать метаинформацию/статистики из этих партиций
- › Если в момент чтения произойдет запись/изменение, то будет конфликт
- › В некоторых видах хранилищ операция `list` может быть очень затратной (например, в S3)
  
- › Хотим:
- › Не делать кучу запросов/не нагружать систему для чтения метаданных
- › Иметь какое-нибудь подобие ACID-гарантий

# Apache Iceberg

- › Был разработан в 2017 году внутри Netflix
- › В 2018 году стал Open Source и уехал в Apache Foundation
- › На тренде “облачного DWH” обрел взрывную популярность
- › Iceberg предоставляет:
  - › API для управления метаданными (Iceberg Core)
  - › FileIO для записи файлов и общения со Storage
  - › Контракт для общения с Catalog



# Как устроен Iceberg

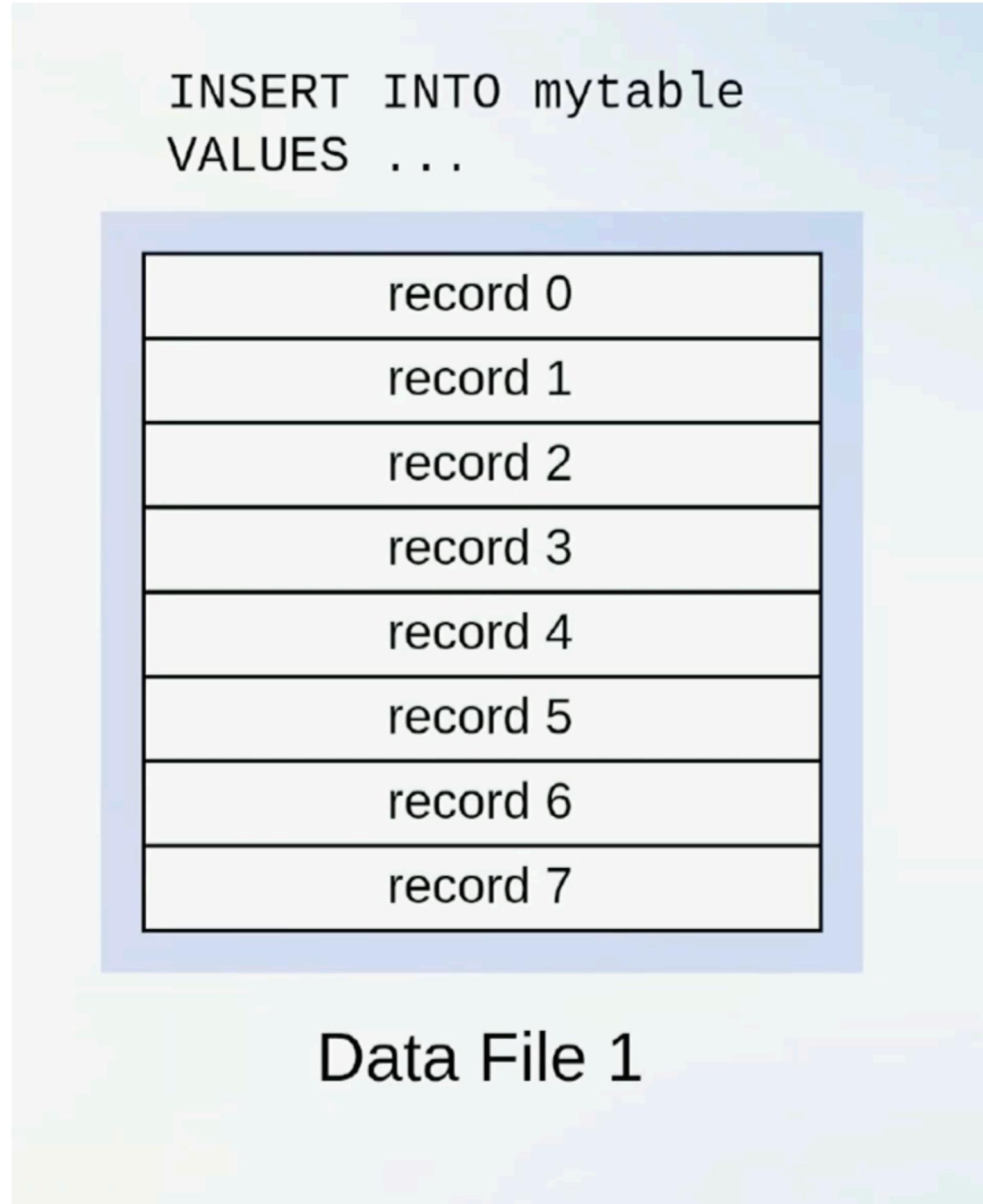


Metadata – ключевой файл метаданных в Apache Iceberg.

Все остальное – это оптимизации для уменьшения копирования:

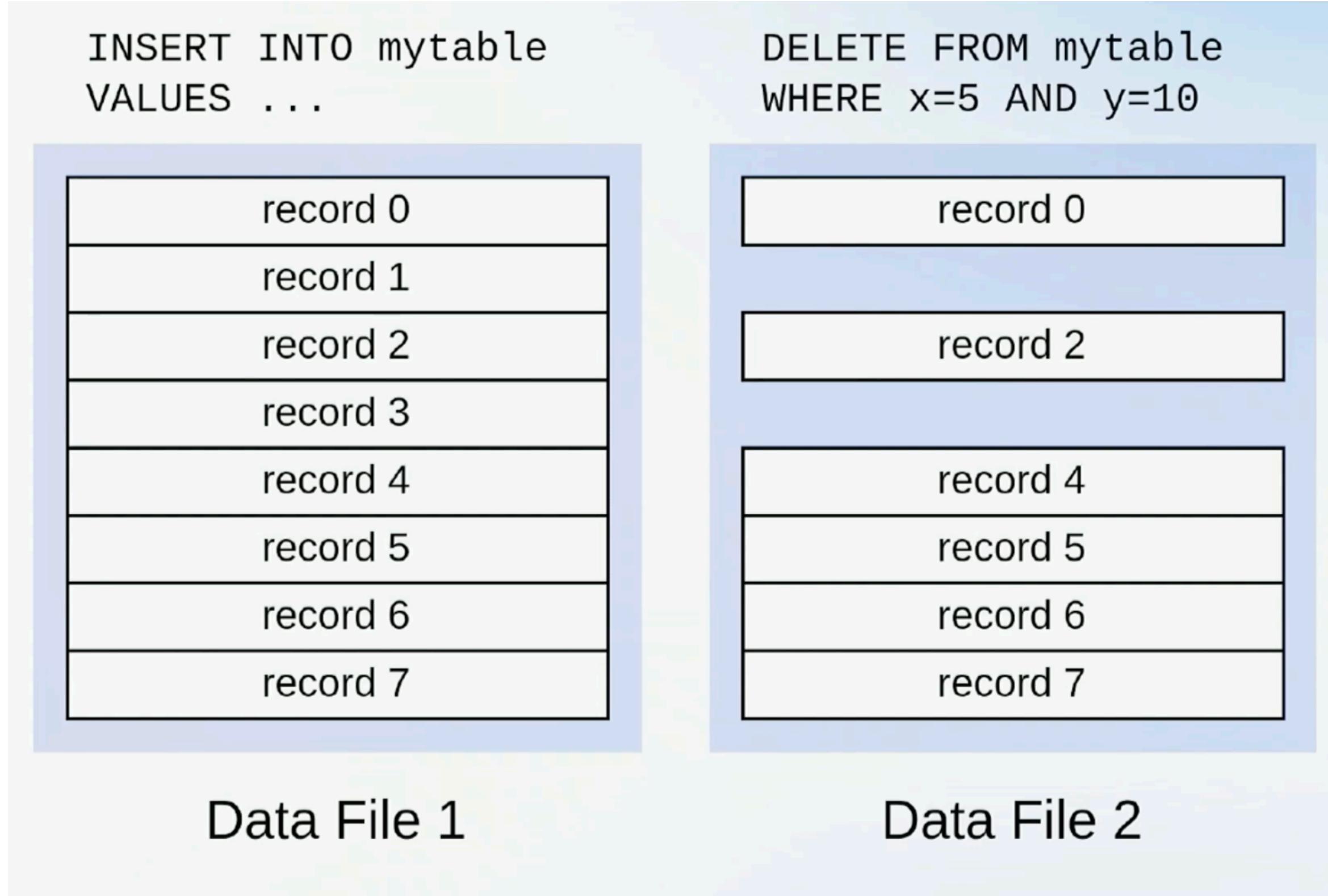
- Manifest list (Avro)
- Manifest (JSON)
- Статистики (Puffin)

# Что умеет Iceberg



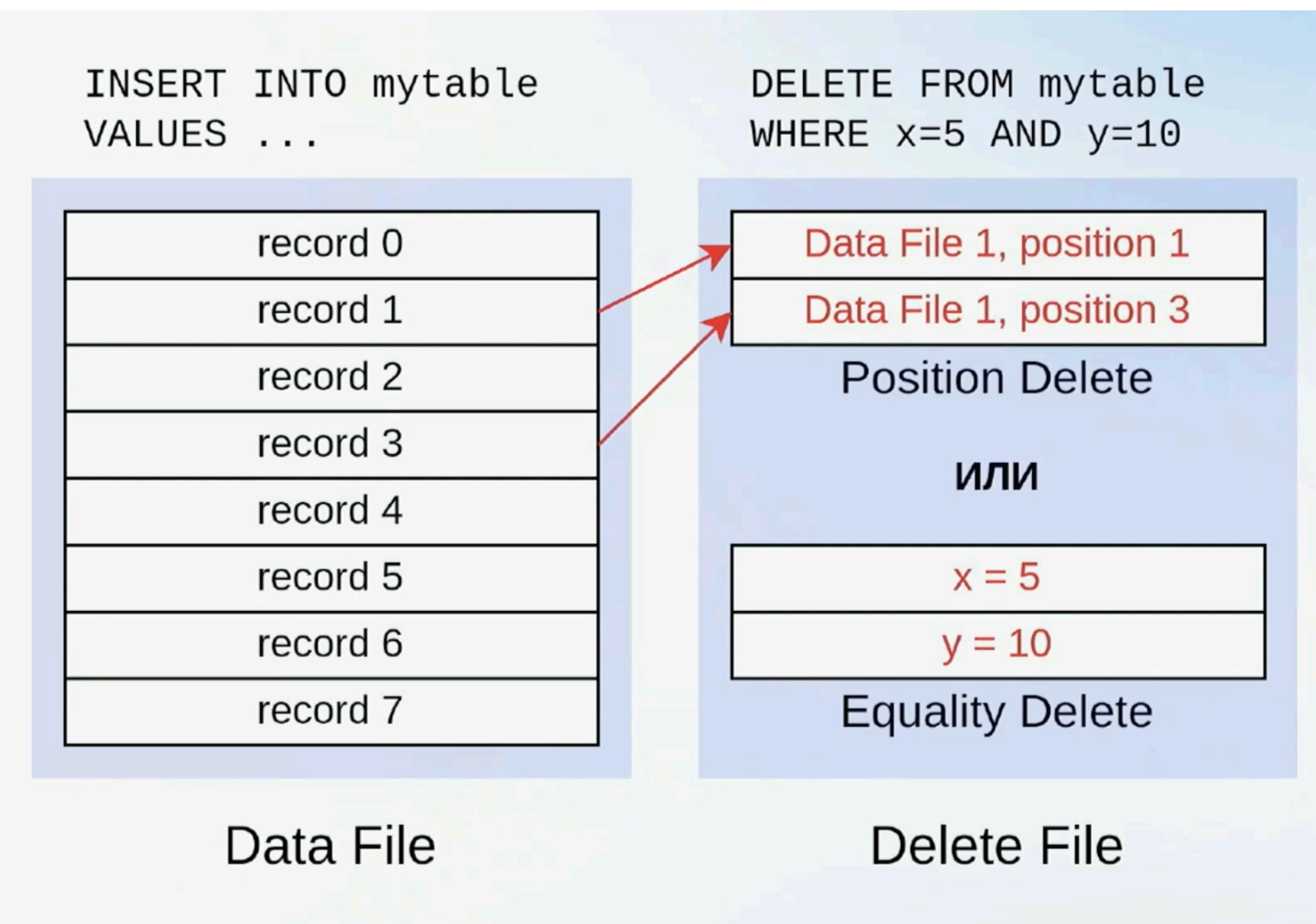
- › Хранит любой бинарь внутри - Avro, ORC, Parquet, Arrow, ProtoBuf ...

# Что умеет Iceberg



- › Хранит любой бинарь внутри - Avro, ORC, Parquet, Arrow, ProtoBuf ...
- › Умеет в Copy-on-write

# Что умеет Iceberg

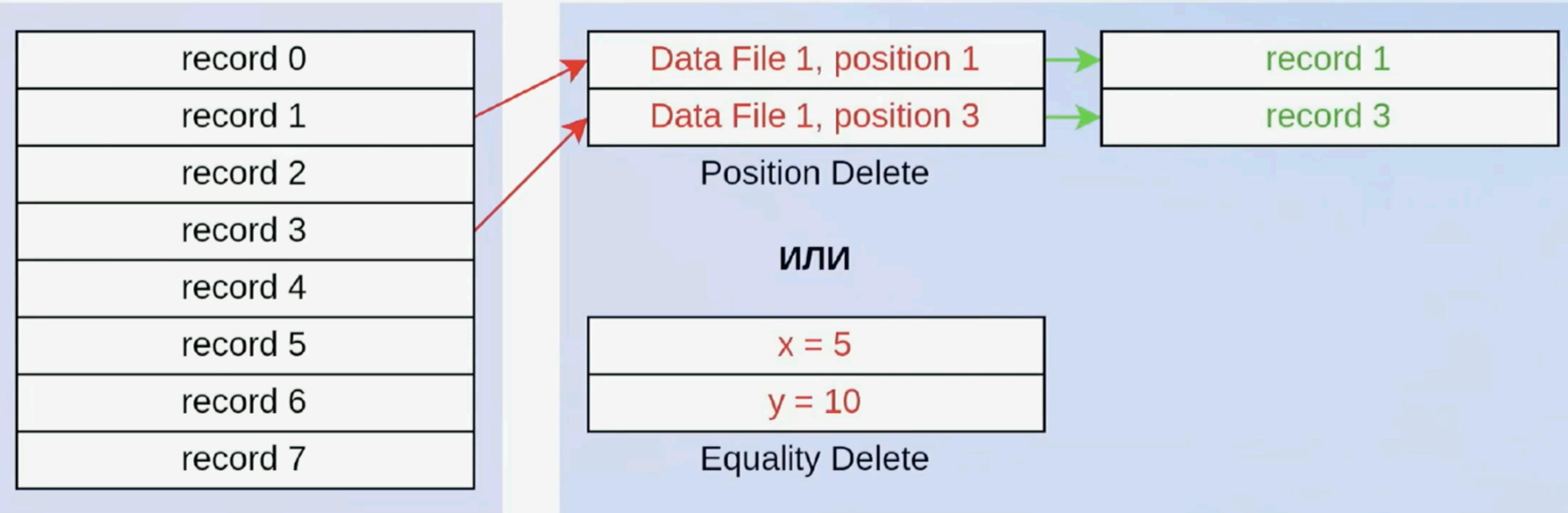


- › Хранит любой бинарь внутри - Avro, ORC, Parquet, Arrow, ProtoBuf ...
- › Умеет в Copy-on-write
- › Умеет в Merge-on-read

# Что умеет Iceberg

INSERT INTO mytable  
VALUES ...

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



Data File 1

Delete File

Data File 2

# Что умеет Iceberg

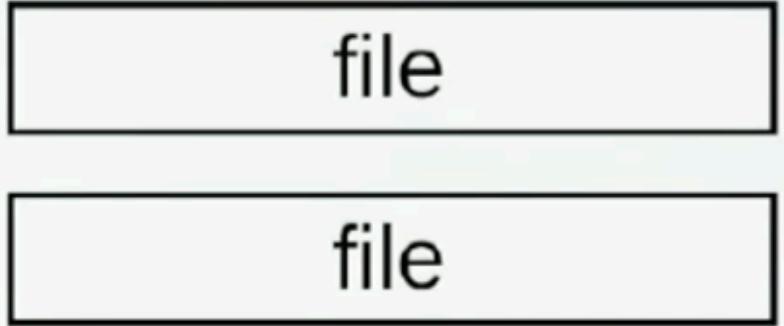


- › Хранит любой бинарь внутри - Avro, ORC, Parquet, Arrow, ProtoBuf ...
- › Умеет в Copy-on-write
- › Умеет в Merge-on-read
- › Умеет в виртуальные колонки дляパーティционирования

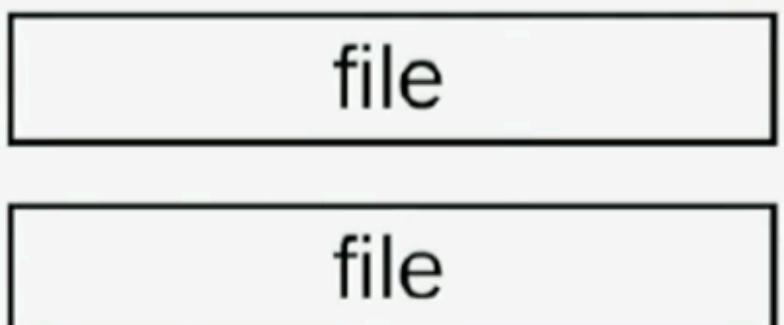
```
sales_year = year(sales_date)
city_bucket = bucket(city, 16)
```

# Что умеет Iceberg

```
/mytable/data/sales_year=2024/city_bucket=2
```



```
/mytable/data/sales_year=2024/city_bucket=3
```

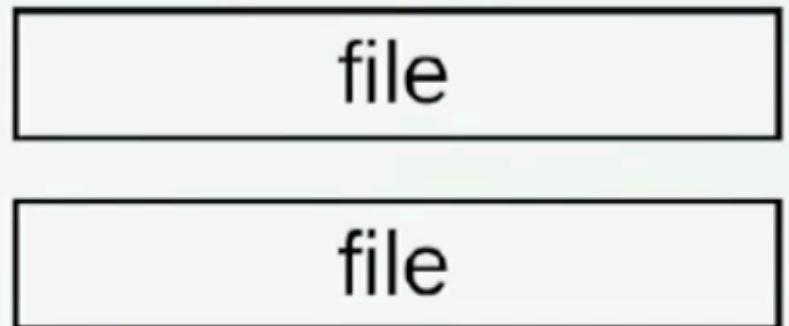


```
sales_year = year(sales_date)  
city_bucket = bucket(city, 16)
```

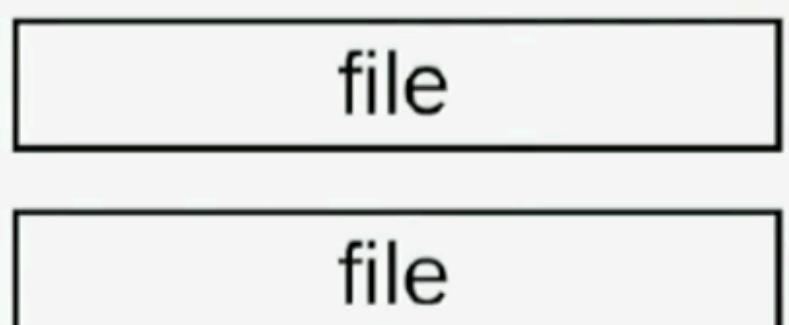
- › Хранит любой бинарь внутри - Avro, ORC, Parquet, Arrow, ProtoBuf ...
- › Умеет в Copy-on-write
- › Умеет в Merge-on-read
- › Умеет в виртуальные колонки дляパーティционирования
- › Собирает точные статистики для data skipping и приблизительные для оптимизаций (theta sketches)

# Что умеет Iceberg

```
/mytable/data/sales_year=2024/city_bucket=2
```



```
/mytable/data/sales_year=2024/city_bucket=3
```



```
sales_year = year(sales_date)  
city_bucket = bucket(city, 16)
```

- › Хранит любой бинарь внутри - Avro, ORC, Parquet, Arrow, ProtoBuf ...
- › Умеет в Copy-on-write
- › Умеет в Merge-on-read
- › Умеет в виртуальные колонки дляパーティционирования
- › Собирает точные статистики для data skipping и приблизительные для оптимизаций (theta sketches)
- › Constraints - Identifier Fields и Sort Fields

# Что еще умеет Iceberg

- › Time Travel и Snapshot Rollback
- › Schema evolution

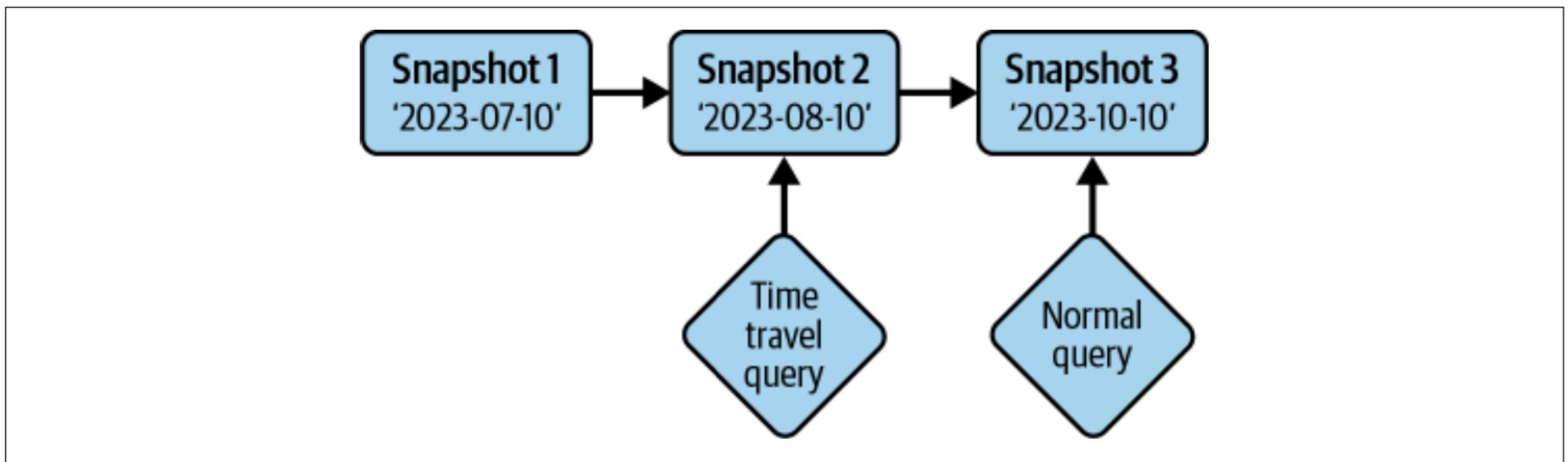


Figure 1-10. Querying the table as it was using time travel

The diagram shows two tables. The left table has rows 1 through 5. Row 4 is highlighted in red with values 'Alex', '37', 'Feb. 2nd, 2023'. An arrow labeled 'Rollback' points to the right table, which contains only the first three rows (1, 2, 3) with identical data to the original table.

ID	Name	Age	Date
1	Bob	46	Jan. 1st, 2023
2	Josie	65	Jan. 10th, 2023
3	Gene	30	Jan. 20th, 2023
4	Alex	37	Feb. 2nd, 2023
5	Tony	34	Feb. 15th, 2023

ID	Name	Age	Date
1	Bob	46	Jan. 1st, 2023
2	Josie	65	Jan. 10th, 2023
3	Gene	30	Jan. 20th, 2023

Figure 1-11. Moving the table's state to a previous point in time by rolling back

# Как записать файл

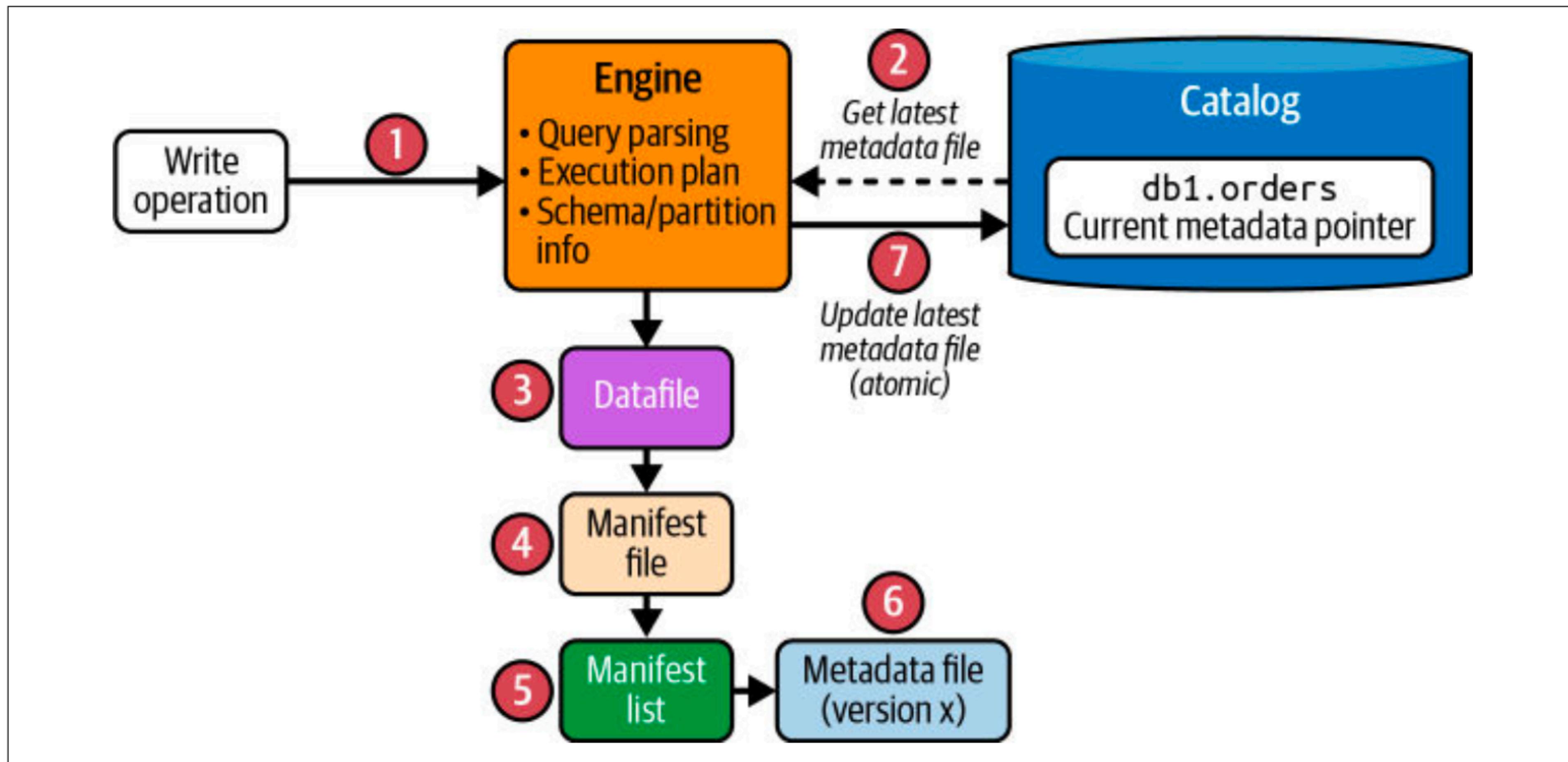
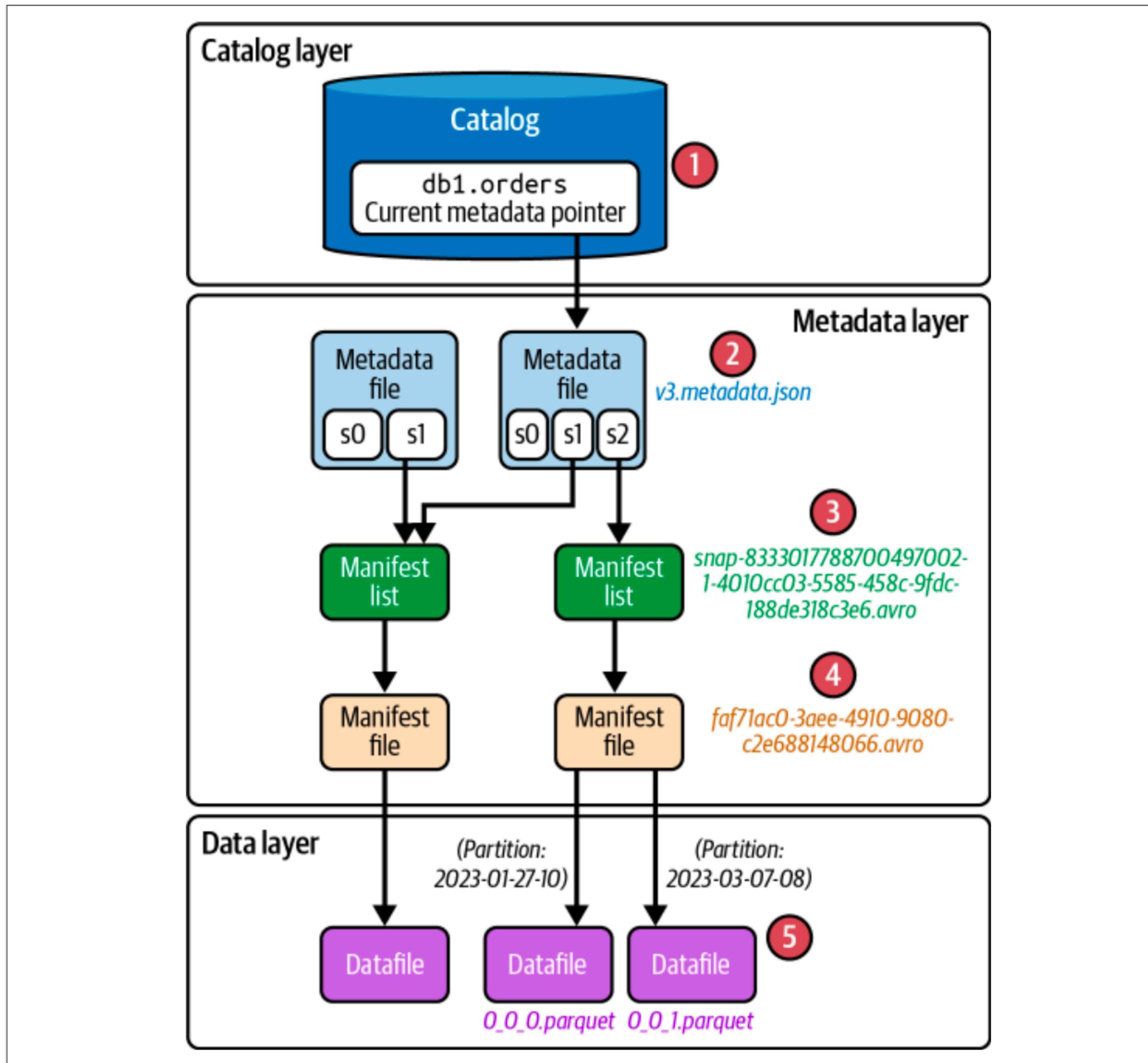


Figure 3-2. Overview of the Apache Iceberg write process

# Как прочитать файл



- › Забрать из каталога текущий metadata файл
- › Найти нужный snapshot
- › Забрать его manifest list
- › Забрать нужные manifest files
- › Прочитать data files

Figure 3-6. How a READ query works in Apache Iceberg