



LakeSoul 开源湖仓一体框架系列分享第一期：

# NativeIO 层实现原理和 在大数据AI一体化场景的实践

陈绪

北京数元灵科技有限公司

# 大纲

01

LakeSoul 背景介绍

02

NativeIO 层设计与实现

03

NativeIO 优化细节

04

NativeIO 应用与展望

# LakeSoul 开源湖仓框架介绍



LF AI & DATA

- 官网: <https://lakesoul-io.github.io/>
- GitHub: <https://github.com/lakesoul-io/LakeSoul>

起源于大型推荐和广告  
业务实时数据流场景

**2021.12**

LakeSoul 国产自  
研流批一体湖仓框  
架开源

**2022.07**

重构元数据, 提升  
并发更新和事务能  
力

**2022.10**

发布 Flink CDC  
多表自动入湖, 支  
持整库同步, 自动  
DDL变更

**2023.05**

发布 Native IO,  
扩大性能领先优势。  
LakeSoul 项目捐  
赠给 Linux 基金  
会孵化

**2023.06 ~ Now**

发布全链路流式增  
量计算, 自动合并  
等功能。  
支持 PyTorch、  
Ray、Pandas 读  
取

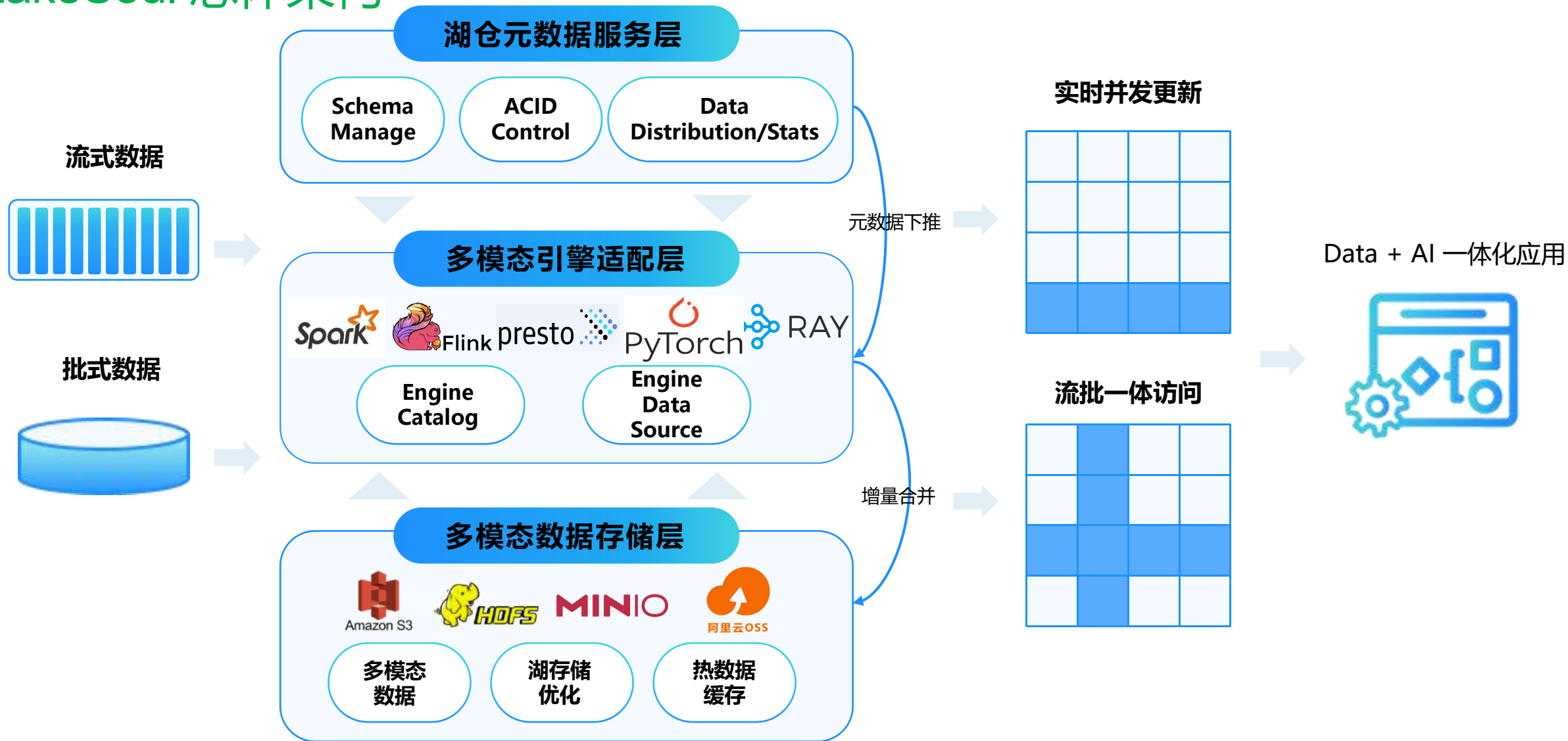
# LakeSoul 开源湖仓框架介绍

开源项目定位：一站式 Data + AI 湖仓框架



# LakeSoul 开源湖仓框架介绍

## LakeSoul 总体架构



# LakeSoul 开源湖仓框架介绍



## 当前数据湖架构面临的问题

### 数据集成

- 需要支持多种数据源采集、CDC 流式更新，提升实时性

### 数据存储

- 云原生架构，计算存储分离，云存储性能优化

### 数据处理

- 需要支持高性能、低延迟的流、批计算

### 数据应用

- 需要支持大数据、AI多种场景计算生态

# LakeSoul 开源湖仓框架介绍



## LakeSoul 框架的解决方式：Native IO 层

### 数据集成

- 支持多源异构的数据源和数据采集工具，支持实时 CDC 写入、Upsert 更新

实时性

### 数据存储

- 存算分离，弹性扩容，针对云对象存储高度优化

云原生

### 数据处理

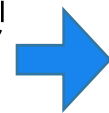
- 支持高性能的批量计算、低延迟的实时增量计算

高性能

### 数据应用

- 数据应用要求数据消费支持多样的应用场景(AI、BI)

开放生态



NativeIO

# LakeSoul NativeIO 实现原理

## 设计目标

01

### 统一封装

- 统一IO实现，封装Upsert、MOR逻辑
- 独立于计算引擎实现
- 封装 Java/Python 接口

02

### 多引擎生态

- 向量化内存格式，跨语言零拷贝
- 向量化计算框架、AI 框架对接

03

### 高性能

- 面向高吞吐批量读写设计
- 分离式弹性 Compaction，减少写放大
- 充分使用异步并行手段提升存储系统访问性能



# LakeSoul NativeIO 实现原理

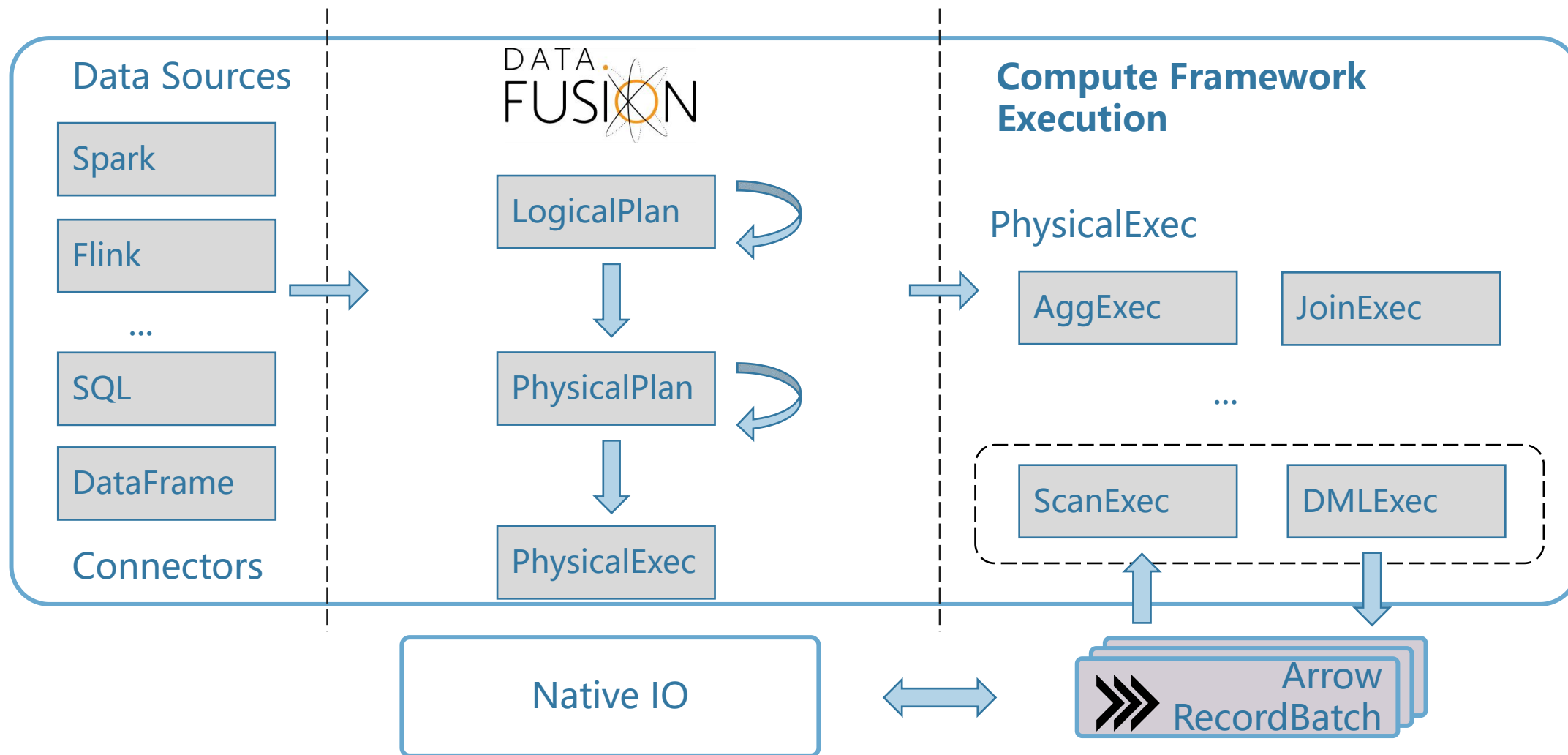


## 技术选型

- 实现语言: Rust
- 数据格式:
  - Apache Parquet(磁盘) + Apache Arrow(内存)
  - 有主键表、无主键表
- 物理实现
  - Apache DataFusion
  - Apache Arrow-RS
  - Tokio
- NativeIO SDK:
  - Arrow C data interface
  - Java: com.github.jnr:jnr-ffi
  - Rust: std::ffi, arrow::ffi
  - Python: cython, ctypes, pyo3
- Engine Connectors
  - Spark/Flink/Presto
  - PyTorch/Pandas/Ray

# LakeSoul NativeIO 实现原理

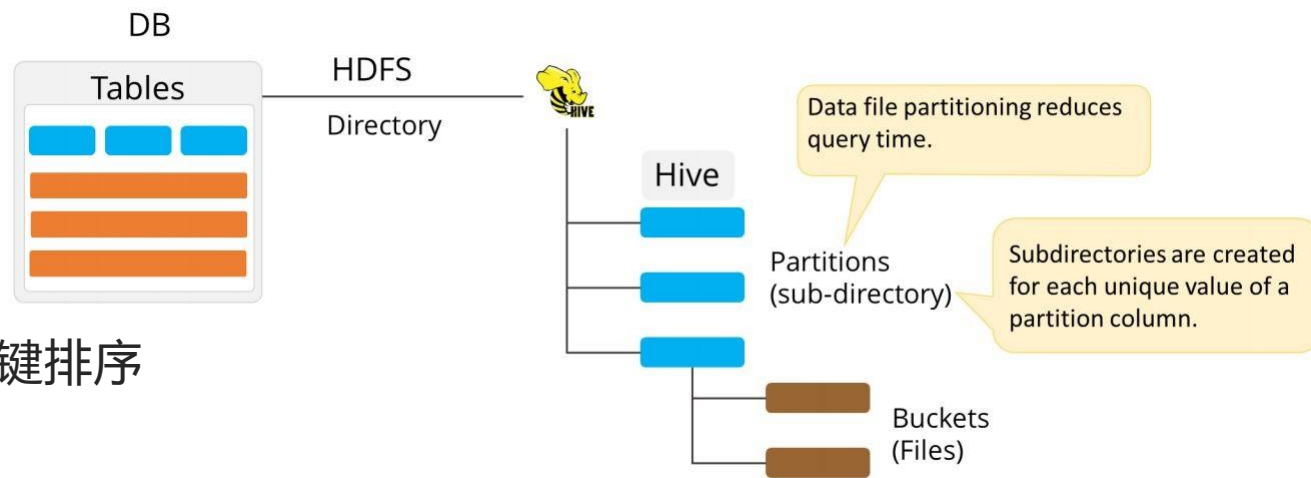
## NativeIO 总体架构



# LakeSoul NativeIO 实现原理

## 表文件组织格式

- 分区表
  - 支持多级 range 分区，每级每个分区一个子目录
  - `table_path/'date=20240110' /`
- 主键表
  - 单层 LSM-Tree
  - Upsert 时文件按照主键哈希分片，分片内对主键排序
  - 支持 CDC 格式读写
    - 带有一个 rowkind 隐藏列：I/U/D
  - 支持并发部分字段更新(Partial Update)
    - 合并读取时每一列自动忽略不含该列的文件

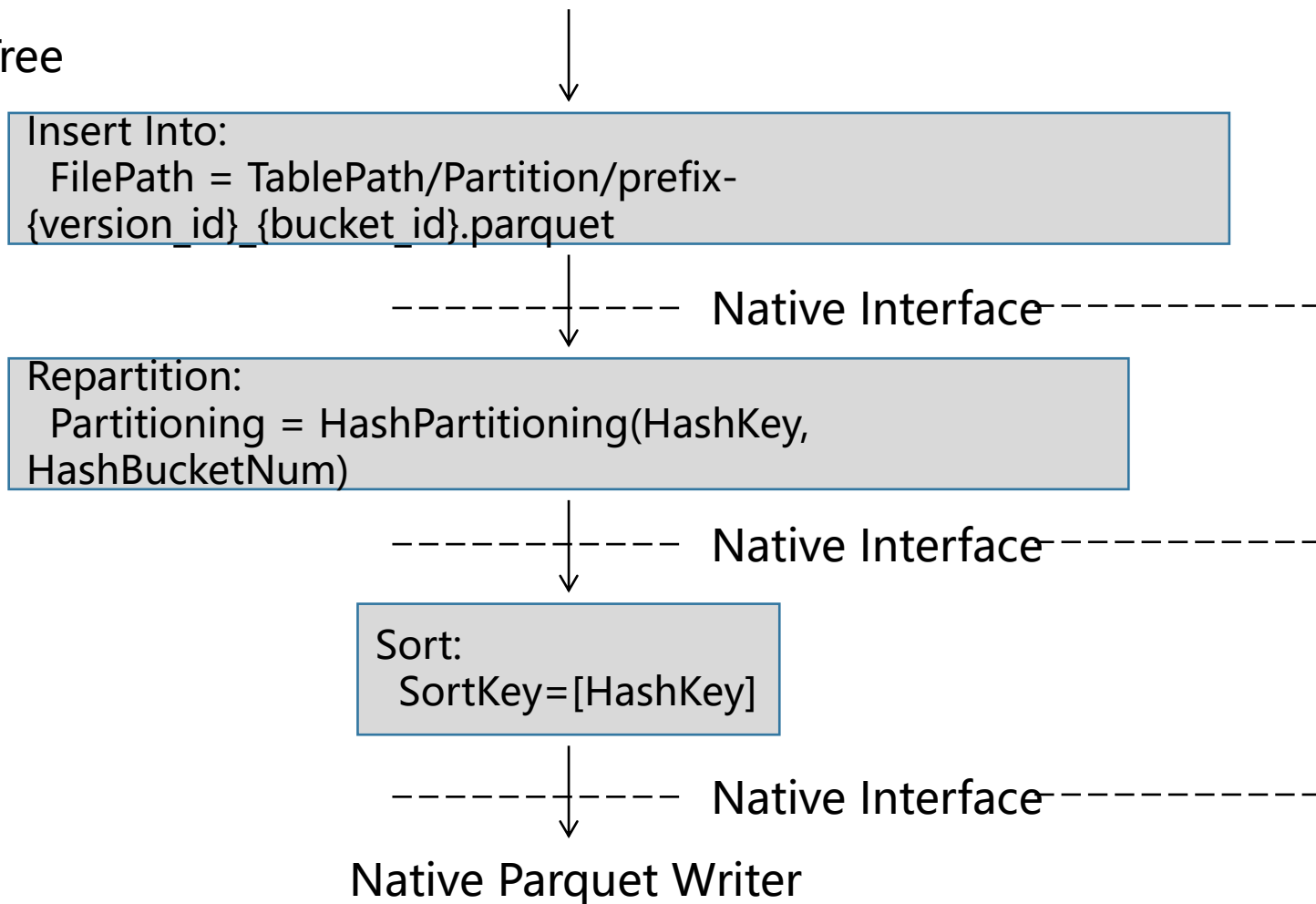


图片来源: <https://www.simplilearn.com/tutorials/hadoop-tutorial/data-file-partitioning>

# LakeSoul NativeIO 实现原理

## 主键表写入流程

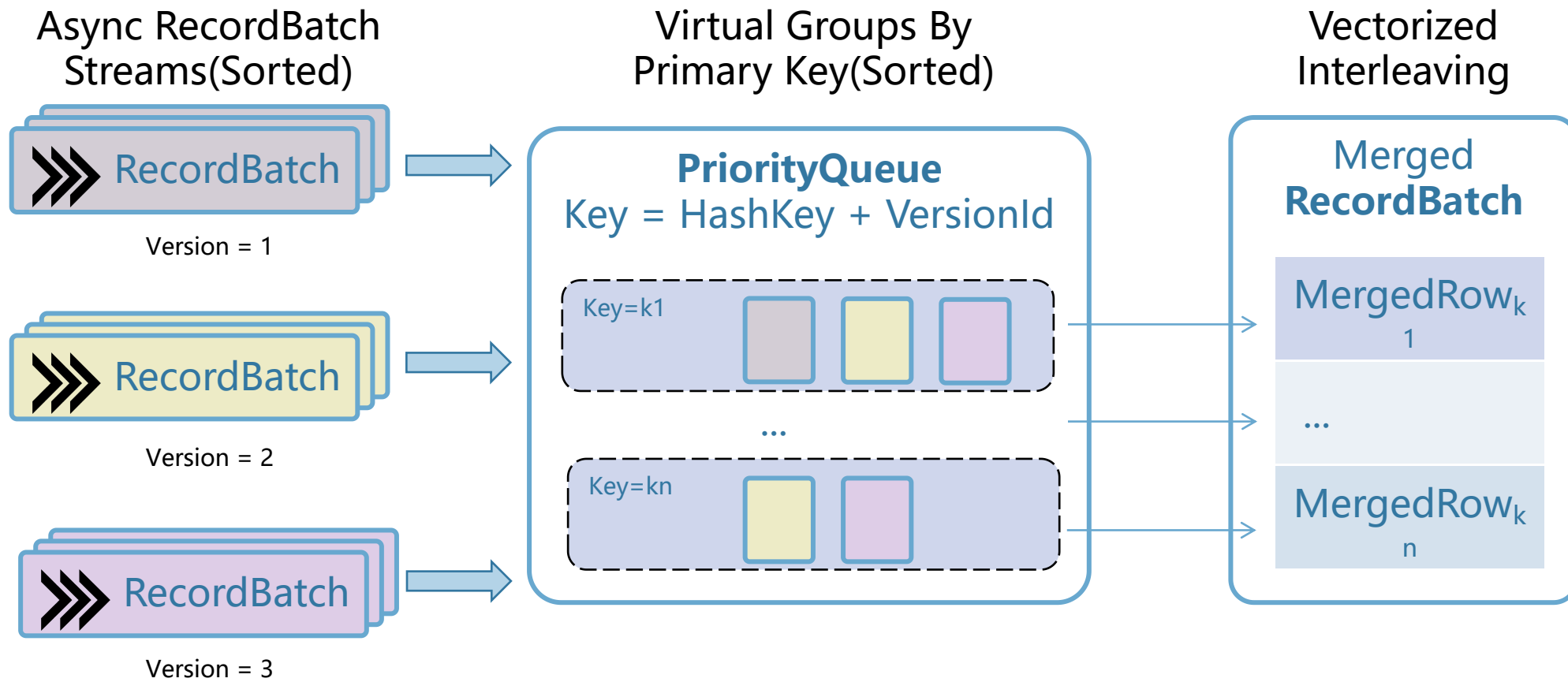
Plan Tree



- 哈希分片 (Repartition)、排序的执行位置可以灵活调整
  - Spark: 分片、排序均在 Spark 中执行
  - Flink: 分片在 Flink 中执行, 排序在 NativeIO 层中执行
- 引擎全局分片减少小文件个数
- Native 层排序采用 Spill Sort 节省内存
- 写路径不做 Compaction(另外提供自动Compaction服务)

# LakeSoul NativeIO 实现原理

## 主键表读取流程



# LakeSoul NativeIO 实现原理

## 主键表读取流程

[RecordBatch],  
batch\_size=2

Hash	Value
01	a
01	b
11	a
21	a
21	b
21	c
21	d
31	a
31	b

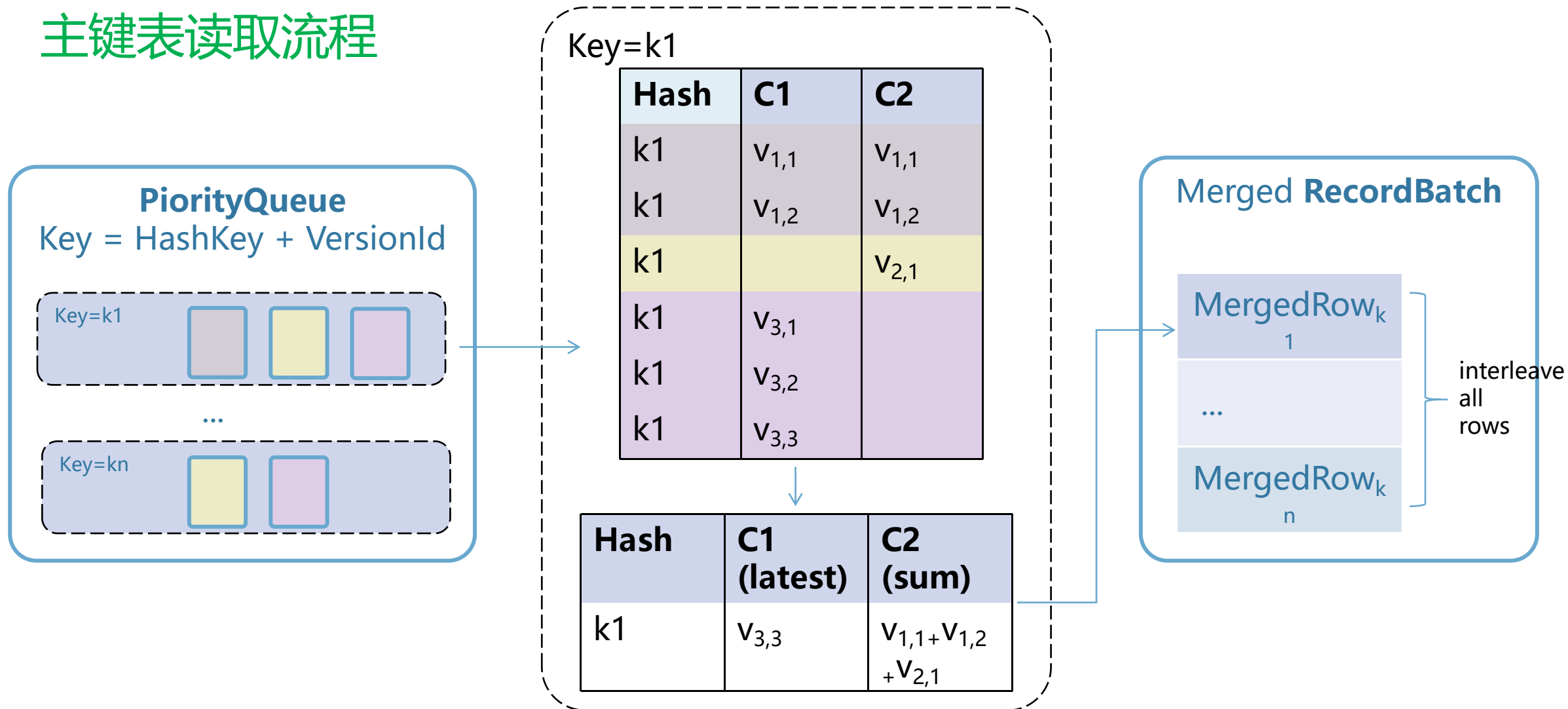
BatchRange

Hash	Value
01	a
01	b
11	a
21	a
21	b
21	c
21	d
31	a
31	b

- 相同主键可能来自于：同一个 batch，同一个文件多个 batch，不同文件的 batch
- 使用优先级队列将相同主键组织在一起（记录stream id, batch id、row id，无数据拷贝）

# LakeSoul NativeIO 实现原理

## 主键表读取流程



- 每一组选择最新的一个主键所对应的行号，形成{stream\_id, batch\_id, begin/end\_row\_id} 四元组
- 使用 Arrow Interleave 算子向量化拼接最终输出的 batch

# LakeSoul NativeIO 实现原理

## CDC(Change Data Capture) 原生支持

自动添加RowKind隐藏

PK	Value	RowKind
01	a	I
01	b	U
11	a	I
21	a	I
21	b	U
21	null	D

批读时自动过滤删除行  
(Spark/Flink/Presto/PyTorch)

PK	Value
01	b
11	a

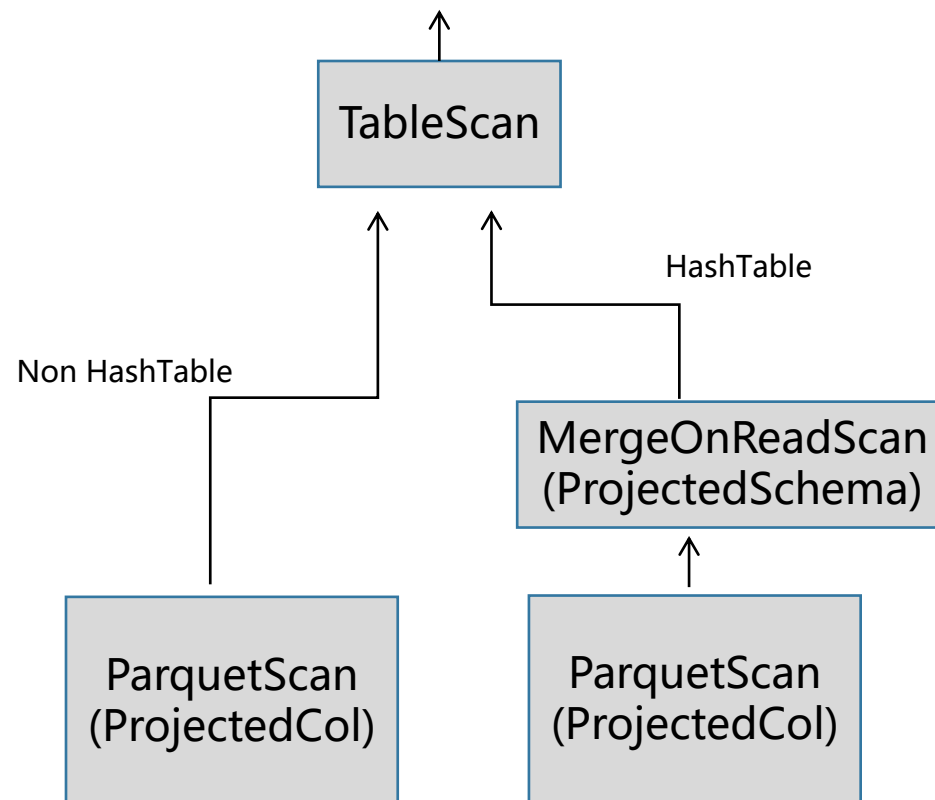
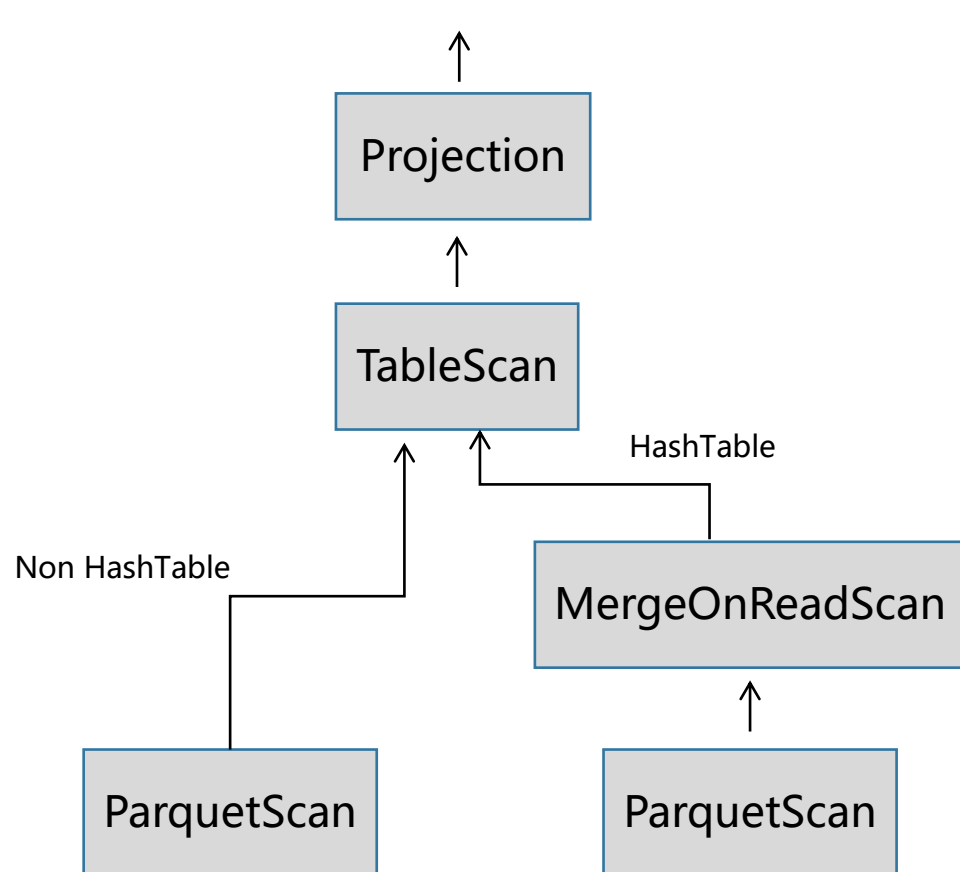
流读时自动填充 Flink  
RowData.rowKind 字段

PK	Value	RowKind
01	a	I
01	b	U
11	a	I
21	a	I
21	b	U
21	null	D



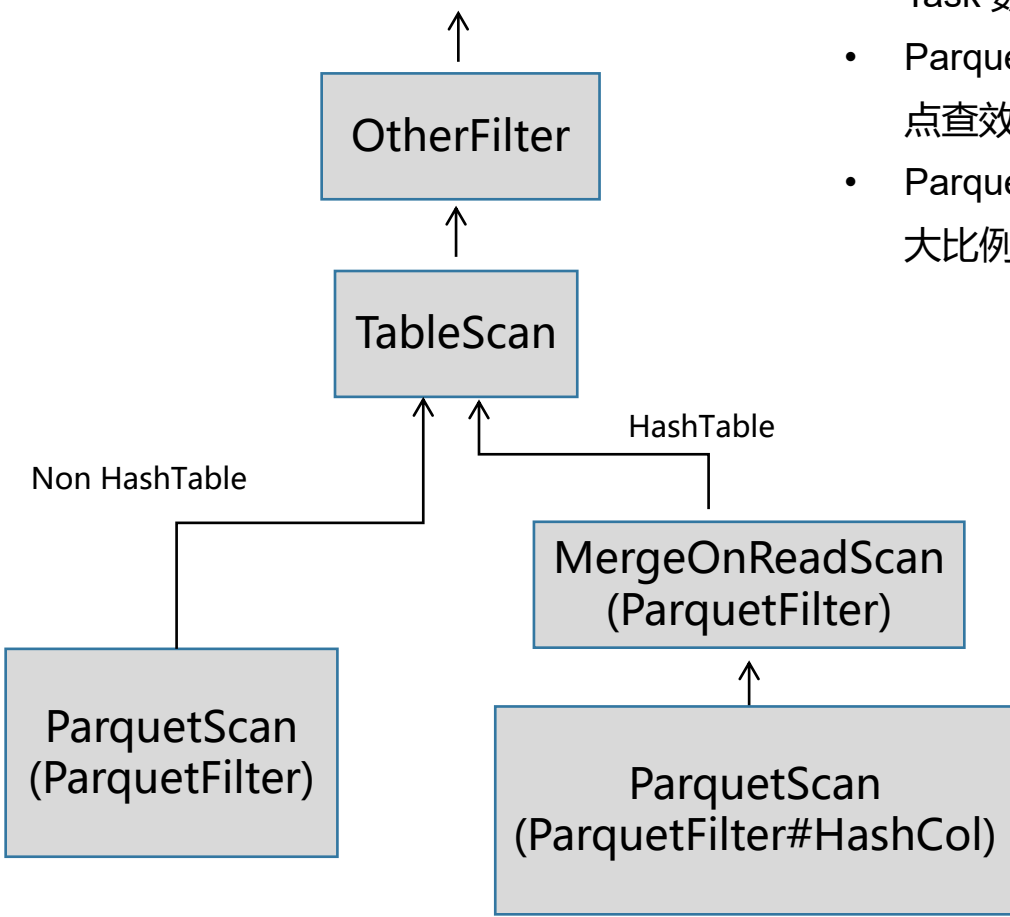
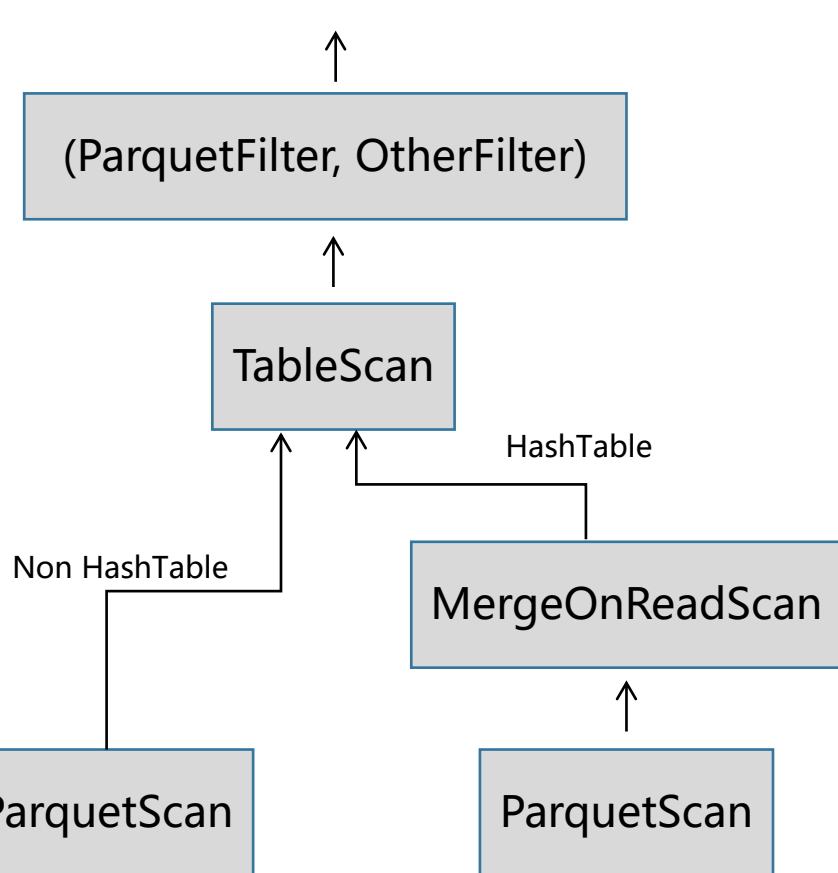
# LakeSoul NativeIO 性能优化

## 列裁剪下推



# LakeSoul NativeIO 性能优化

## Filter 下推



- 分区裁剪在引擎层提前做，减少 Plan Task 数
- Parquet RowGroup Stats 裁剪（主键点查效果明显）
- Parquet Reader Filter：默认关闭（在大比例过滤情况下有效果）

# LakeSoul NativeIO 性能优化



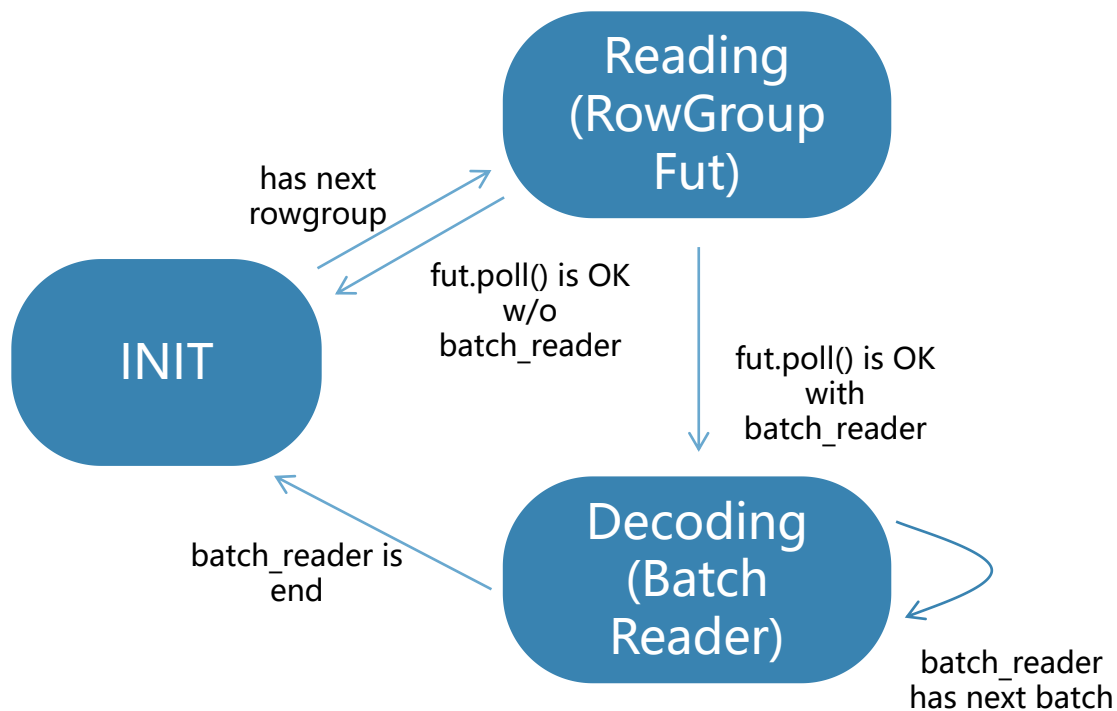
## 对象存储性能优化

- 云对象存储特性：
  - 高带宽，高并发，高延迟
  - 单线程同步：读 30MB/s，写 20 MB/s
- 对象存储性能优化：
  - 读请求拆分：~ 8MB/req
  - 写请求拆分 (Multipart Upload) : >5MB
  - 读时不跨 Part 边界：RowGroup--Part 对应
- Parquet 文件在对象存储上读写优化
  - RowGroup 大小：~ 30MB
  - 读时异步预取 RowGroup
  - 写时异步并发上传 RowGroup

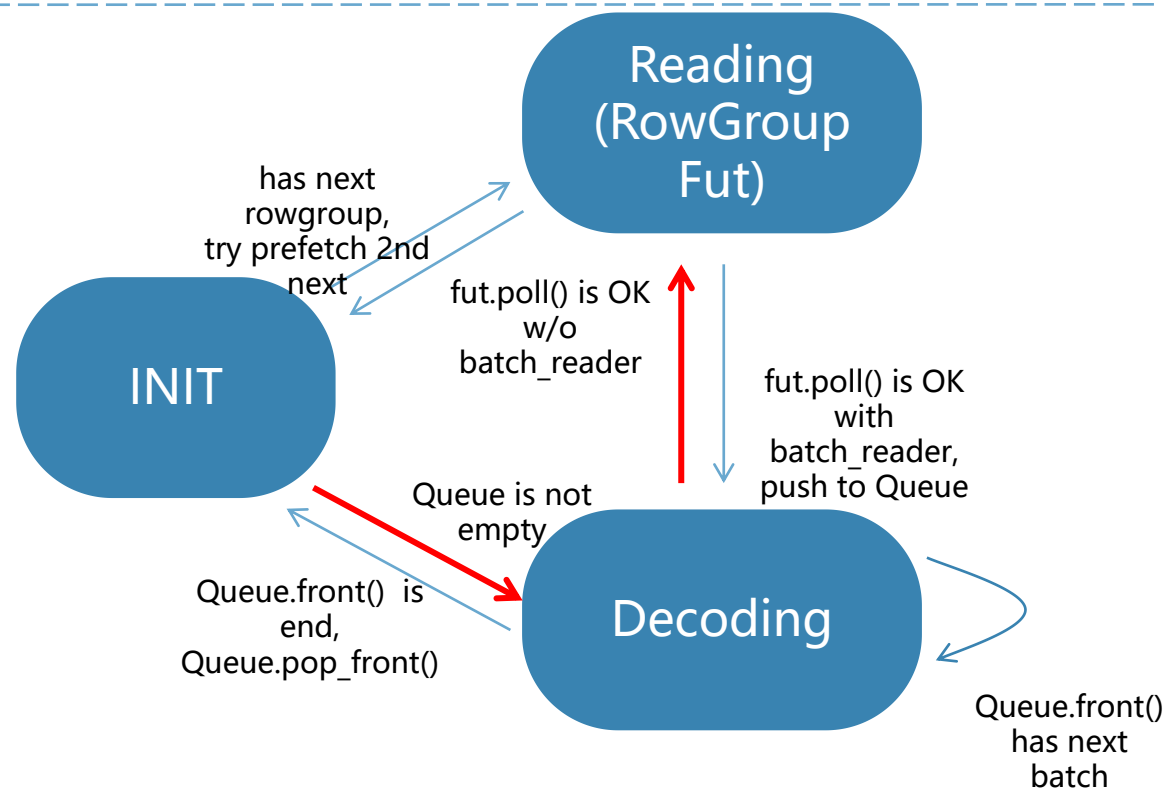
# LakeSoul NativeIO 性能优化

## 对象存储读优化: Parquet RowGroup Prefetch

原始版本, IO 和解码交替执行



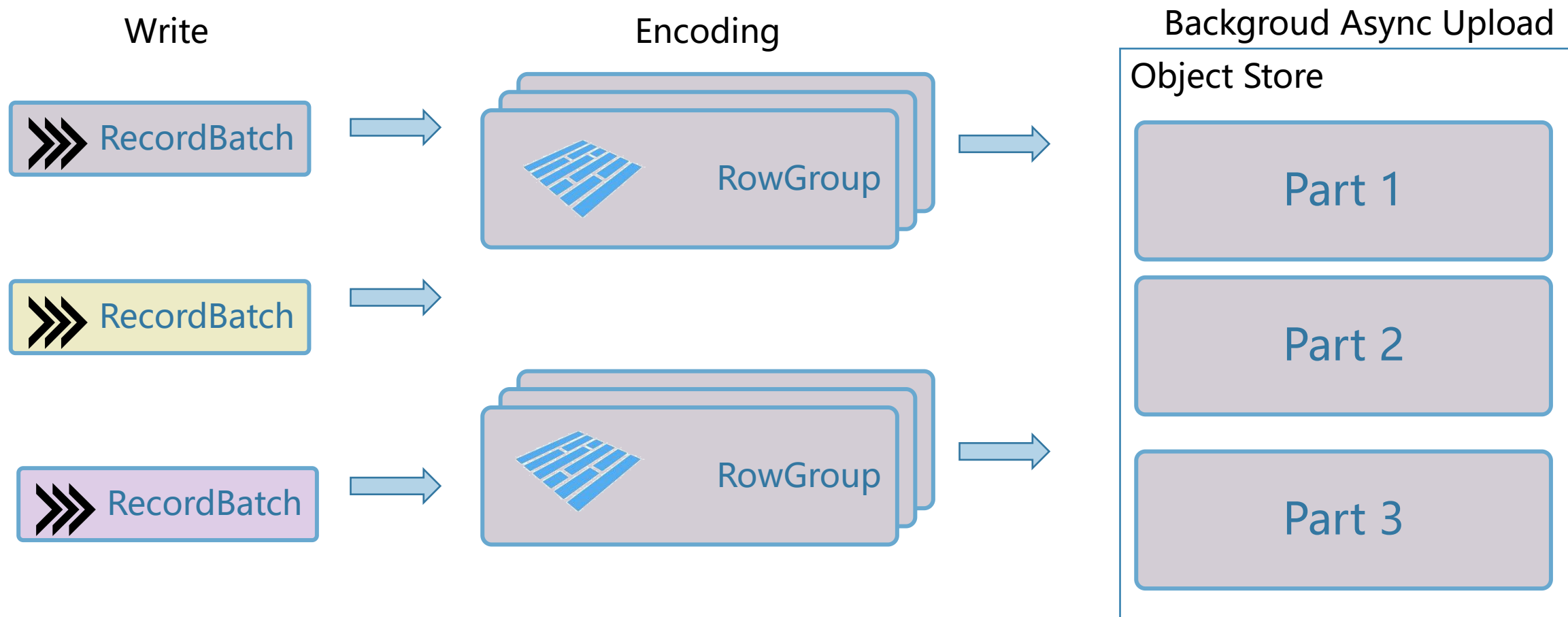
优化版本, IO 和解码并行执行



- 单核心读取 S3 时带宽占用 **200Mbps → 800Mbps**

# LakeSoul NativeIO 性能优化

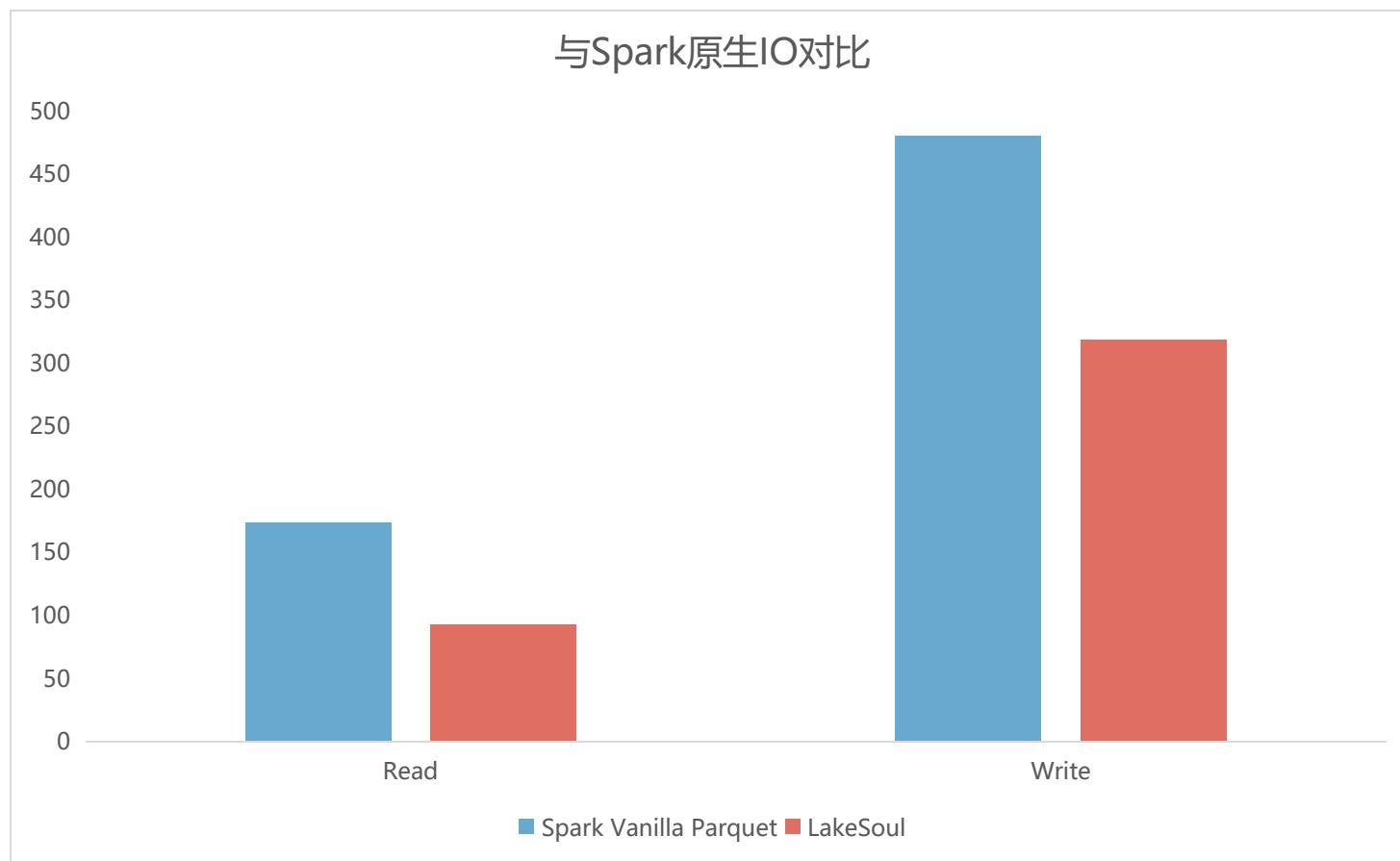
## 对象存储写优化: Parquet RowGroup Parallel Multipart Upload



- 单核心写 S3 时带宽占用 **100Mbps → 300Mbps**

# LakeSoul NativeIO 应用实践

## IO Benchmark



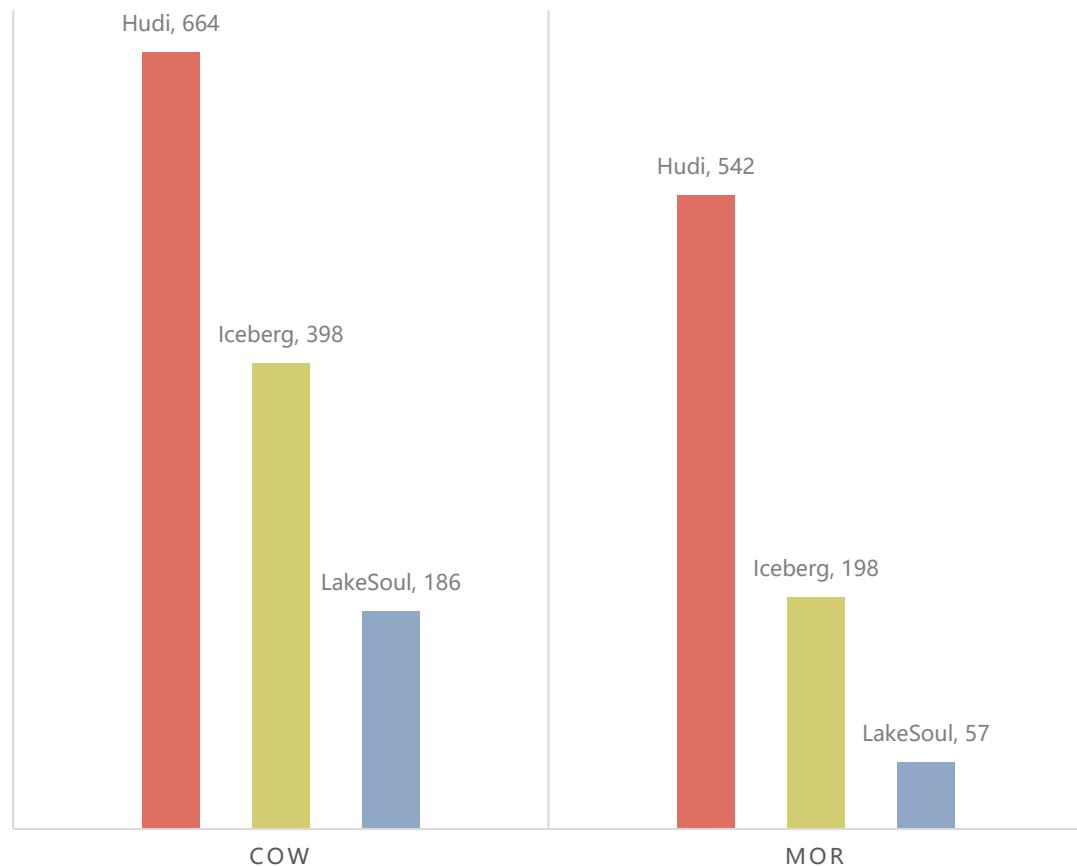
测试方式:

- TPC-H-SF100 Orders 表, 1.5亿行, 写入S3后读取
- Spark 1c8g

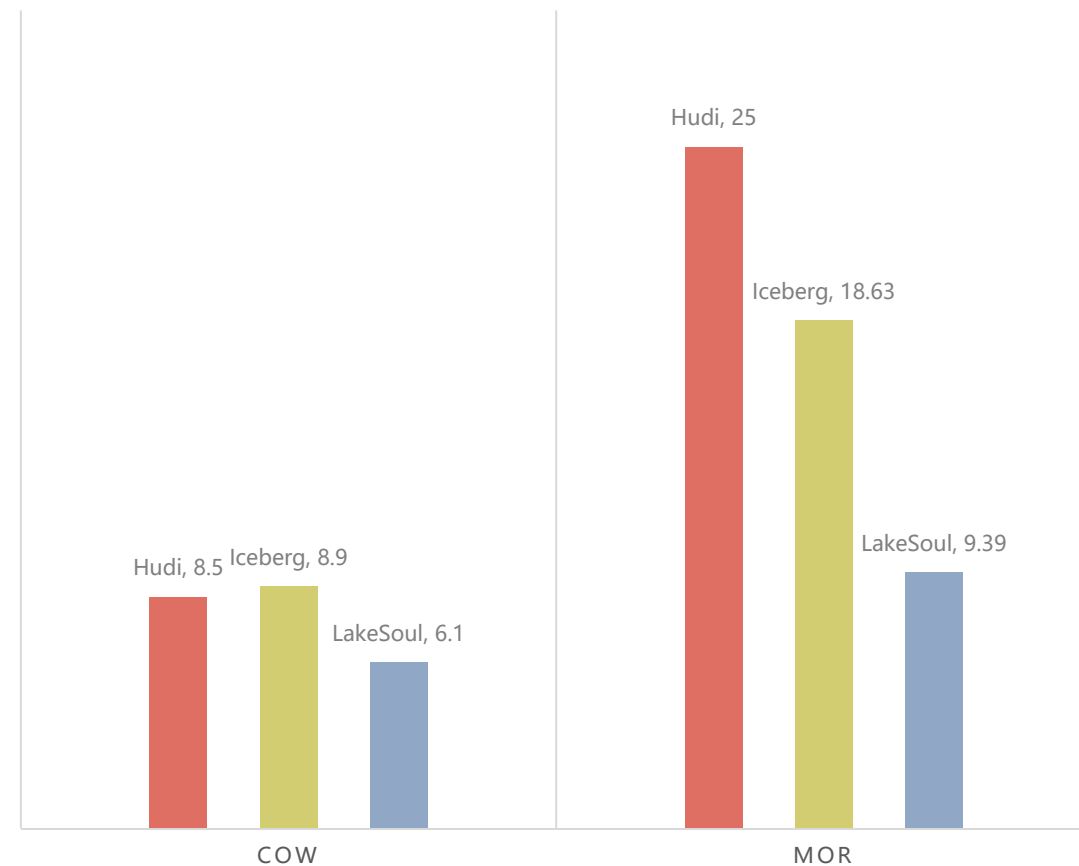
# LakeSoul NativeIO 应用实践

## COW/MOR 读写 Benchmark

WRITE TIME(SECONDS)



READ TIME(SECONDS)



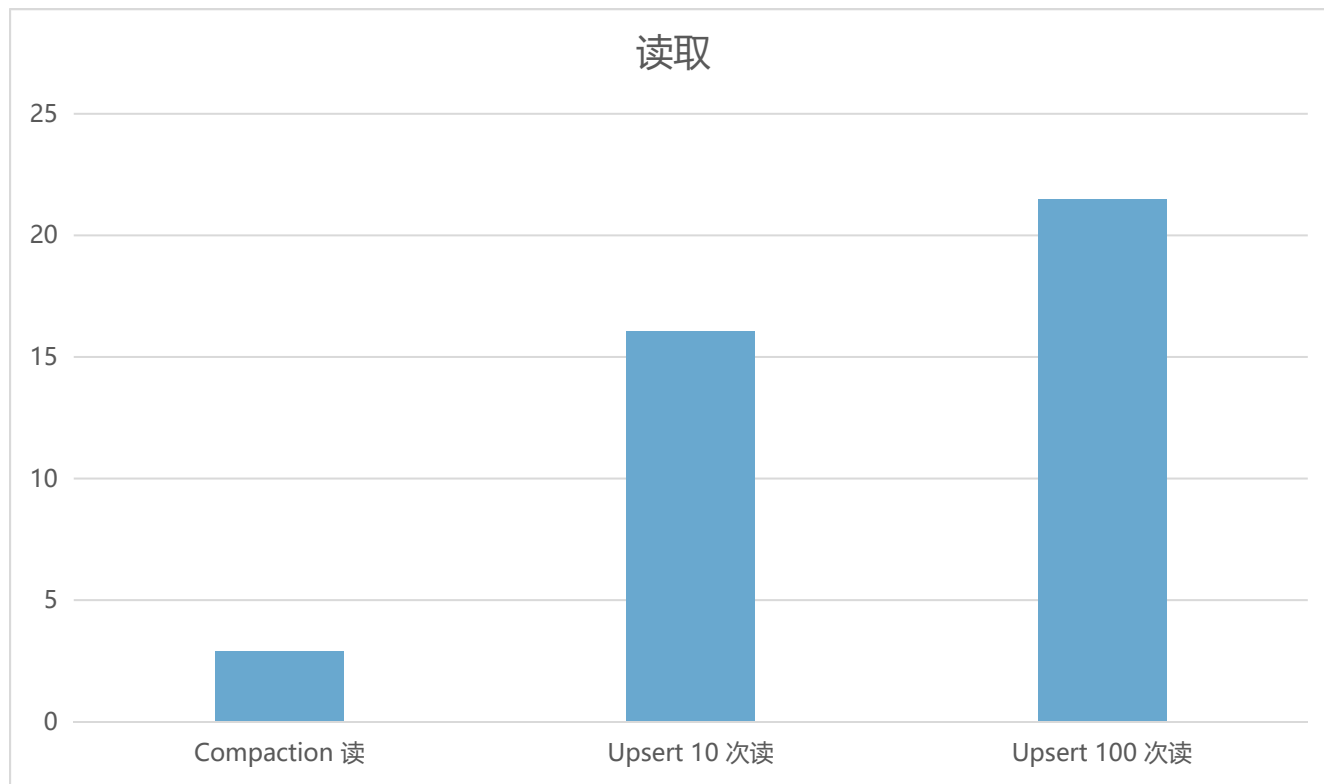
测试方式:

- 第一批插入 1000 万行数据
- 分10次 Upsert 100 万行数据
- MOR 读取时没有执行过 Compaction

<https://github.com/meta-soul/ccf-bdci2022-datalake-contest-examples/tree/mor>  
<https://github.com/meta-soul/ccf-bdci2022-datalake-contest-examples/tree/cow>

# LakeSoul NativeIO 应用实践

## MOR 小文件读取 Benchmark



测试方式:

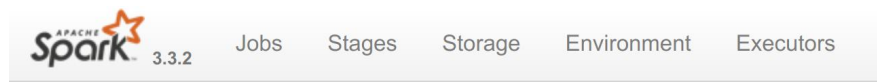
- 第一批插入 1000 万行数据
- 分10次、分100次 Upsert 100 万行数据
- MOR 读取时没有执行过 Compaction

- **文件大小: 43MB(10次)、4.8MB(100次)**
- 读时合并 10 个文件: 16s
- 读时合并 100 个文件: 21.5s
- 合并100个文件耗时增加 30%



# LakeSoul NativeIO 应用实践

## 向量化引擎: Spark Gluten



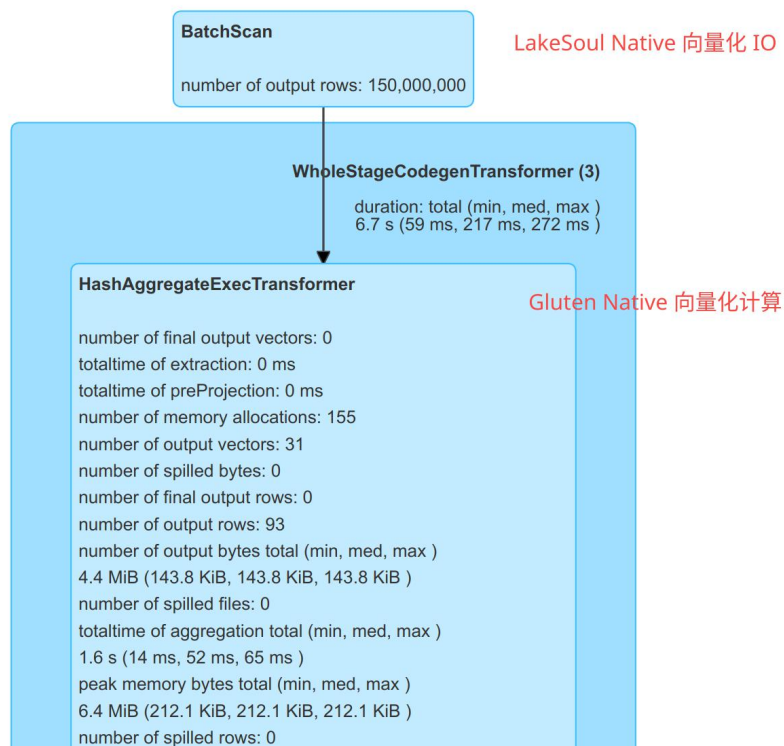
### Details for Query 2

Submitted Time: 2023/12/20 17:03:21

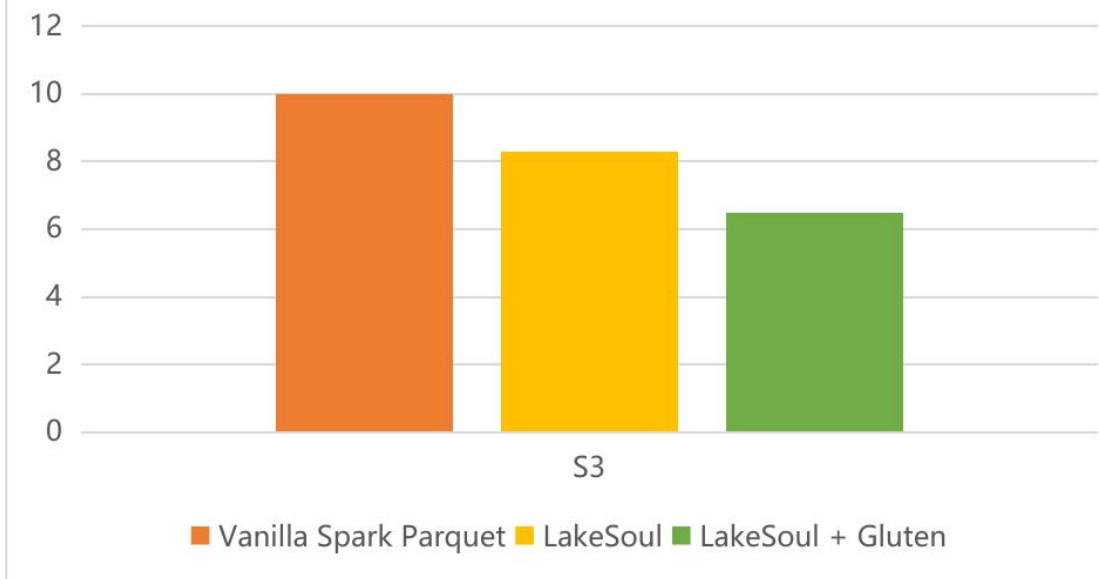
Duration: 1.0 s

Succeeded Jobs: 2 3

☐ Show the Stage ID and Task ID that corresponds to the max metric

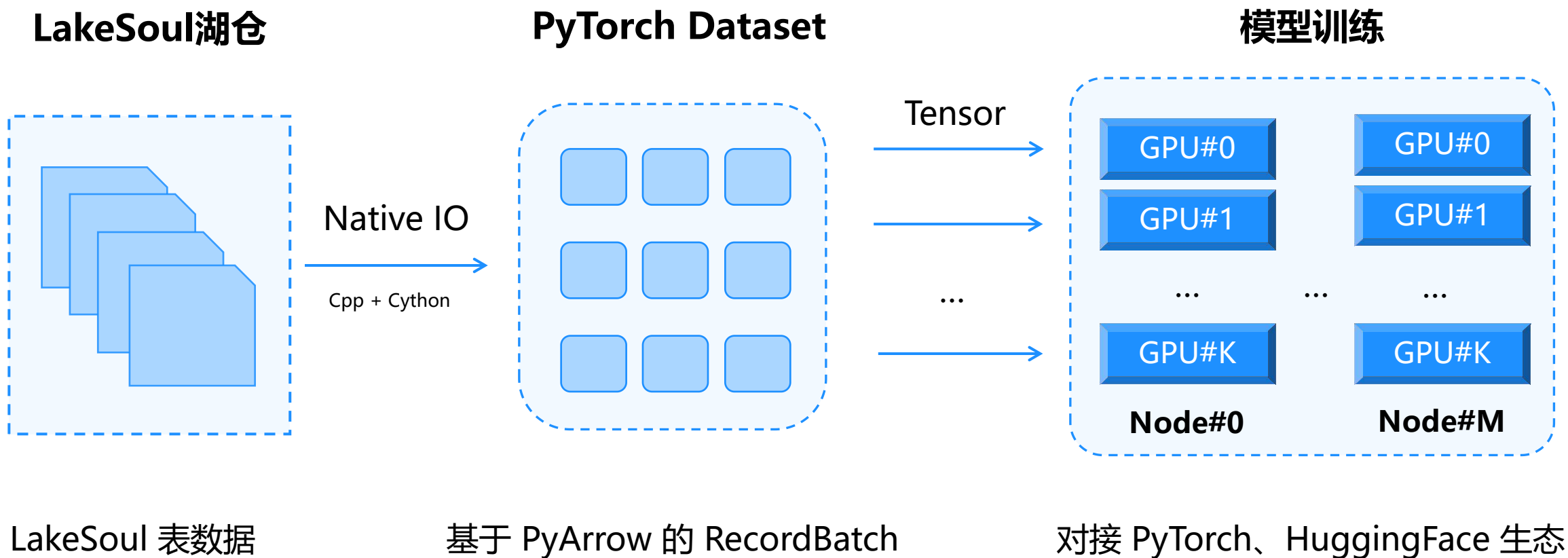


查询性能对比 (单位: 秒)



# LakeSoul NativeIO 应用实践

## AI 训练读取 LakeSoul 表



# LakeSoul NativeIO 应用实践



AI、Data Science 计算生态: PyTorch、HuggingFace、Ray、Pandas、PyArrow

```
from lakesoul.arrow import lakesoul_dataset

ds = lakesoul_dataset("table_name", partitions={'split': 'train'})

# iterate batches in dataset
# this will not load entire table to memory
for batch in ds.to_batches():
    ...

# convert to pandas table
# this will load entire table into memory
df = ds.to_table().to_pandas()
```

```
import datasets
import lakesoul.huggingface

dataset =
datasets.IterableDataset.from_lakesoul("lakesoul_table", partitions={'split': 'train'})
```

```
import ray.data
import lakesoul.ray
ds = ray.data.read_lakesoul("table_name",
partitions={'split': 'train'})
```

支持分布式训练环境

# LakeSoul NativeIO 应用实践



AI 大模型训练 <https://github.com/lakesoul-io/LakeSoul/tree/main/python/examples>

```
dataset_table = "imdb"
```

```
def read_text_table(datasource, split):  
    dataset = datasets.IterableDataset.from_lakesoul(datasource, partitions={"split": split})  
    for i, sample in enumerate(dataset):  
        yield {"text": sample["text"], "label": sample["label"]}
```

```
# Tokenize the IMDb dataset
```

```
train_tokenized_imdb = IterableDataset\  
    .from_generator(read_text_table, gen_kwargs={"datasource": dataset_table, "split": "train"})\  
    .map(preprocess_function, batched=True)\  
    .shuffle(seed=1337, buffer_size=25000)  
test_tokenized_imdb = IterableDataset\  
    .from_generator(read_text_table, gen_kwargs={"datasource": dataset_table, "split": "test"})\  
    .map(preprocess_function, batched=True)
```

# LakeSoul 近期开发计划

## • 功能

- 可插拔 WAL 支持  
亚秒级实时可见性
- 实时数据质量校验
- Hadoop/K8s 自动化部署
- 数据开发平台前端

## • 生态

- 支持更多数据库入湖
- Kafka Connect Sink
- LogStash Sink

## • 性能

- Minor compaction
- Spark Columnar Writer
- Presto Velox Connector
- Apache Doris Connector
- Clickhouse Connector

GitHub: <https://github.com/lakesoul-io/LakeSoul>



# 谢谢



欢迎扫码进群交流

快速体验：

<https://lakesoul-io.github.io/zh-Hans/docs/Getting%20Started/setup-local-env>