



LakeSoul's NativeIO Layer Implementation Principle and Practice

Chen Xu
Beijing Shuyuan Ling
Technology Co., LTD

Outline

01

LakeSoul
Background
introduction

02

NativeIO Layer design
and implementation

03

NativeIO
Optimization details

04

NativeIO Application
and prospects

LakeSoul Open source lake warehouse framework introduction

LF AI & DATA

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

Originated in the real-time data flow scenario of large recommendation and advertising businesses

2021.12
LakeSoul
Domestic self-research flow batch integrated lake warehouse framework open

2022.07

Refactoring metadata to improve concurrent update and transaction capabilities

2022.10

Release Flink CDC multi-table automatically into the lake, support the whole library synchronization, automatic DDL change

2023.05

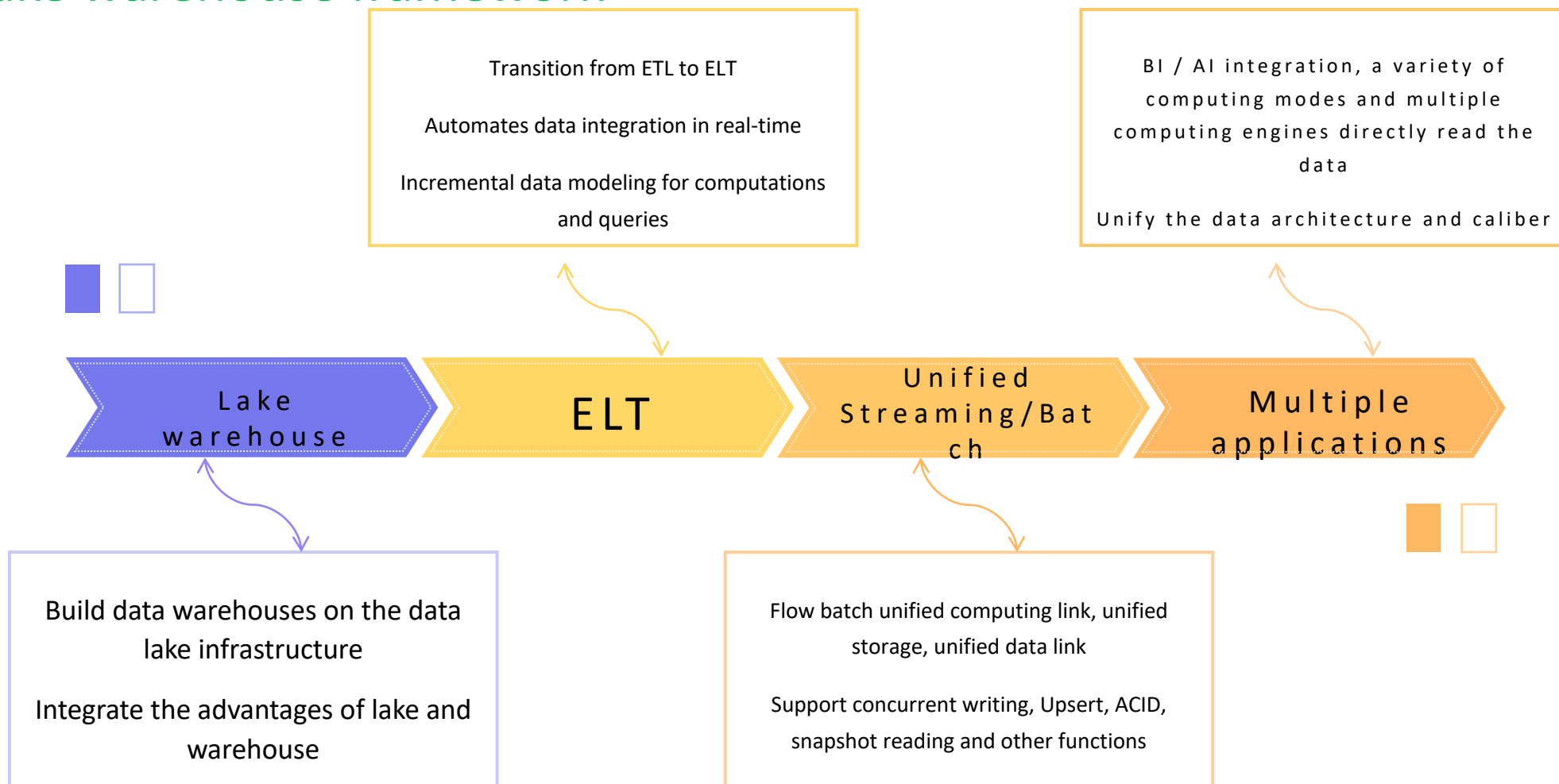
Release of Native IO to expand performance leadership. LakeSoul Project was donated to Linux, Foundation Incubation

2023.06~Now

Release the full link flow incremental calculation, automatic merge and other functions. Support for PyTorch, Ray, and Pandas reads

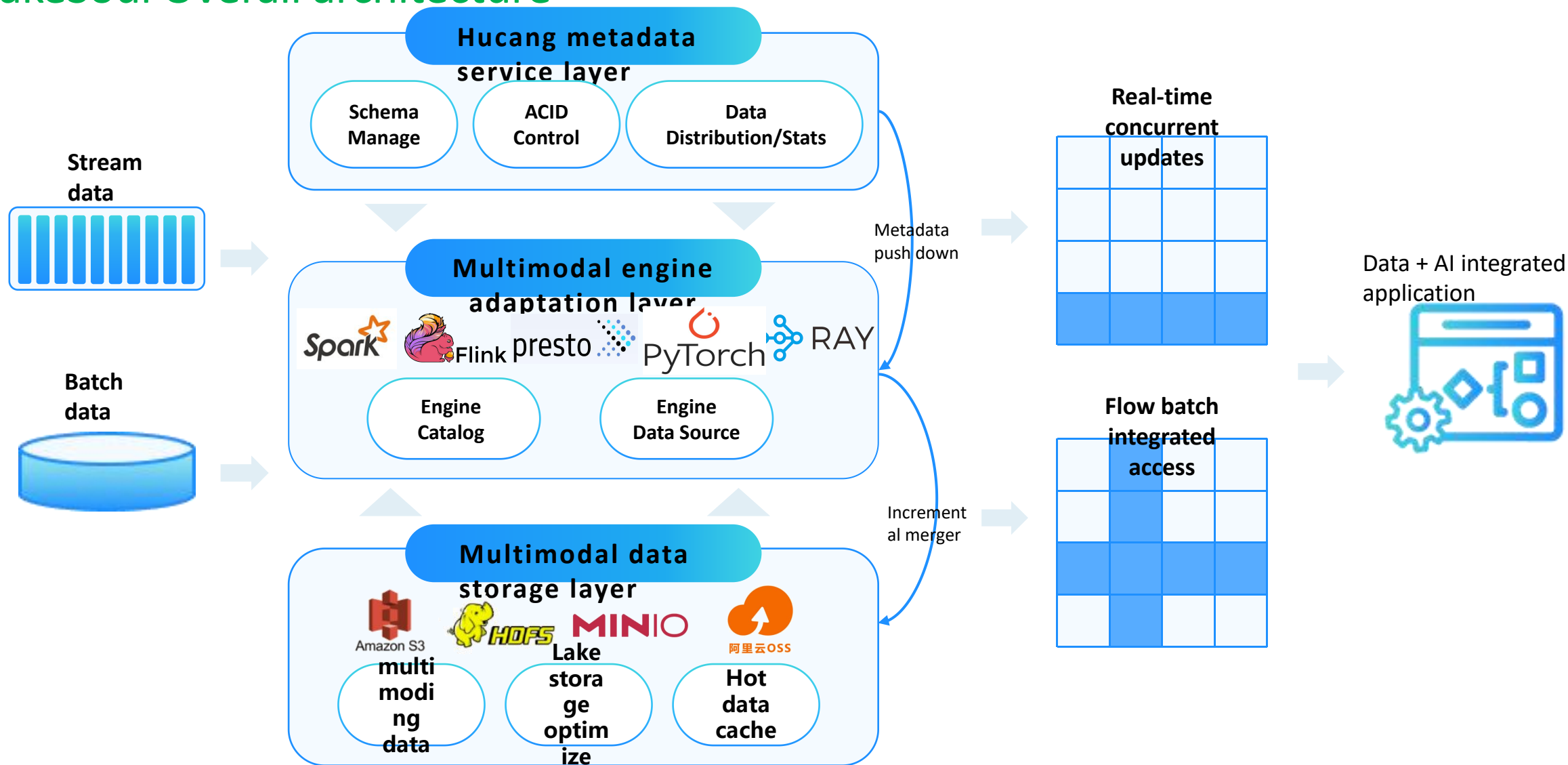
LakeSoul Open source lake warehouse framework introduction

Open source project positioning: one-stop Data + AI Lake warehouse framework



LakeSoul, Open source lake warehouse framework introduction

LakeSoul Overall architecture



LakeSoul, Open source lake warehouse framework introduction

Problems facing the current data

lake architecture

data Integration

- Need to support a variety of data source collection, CDC flow update, improve the real-time performance

data storage

- Cloud native architecture, computing and storage separation, and cloud storage performance optimization

data handling

- Need to support high performance, low latency flow, batch computing

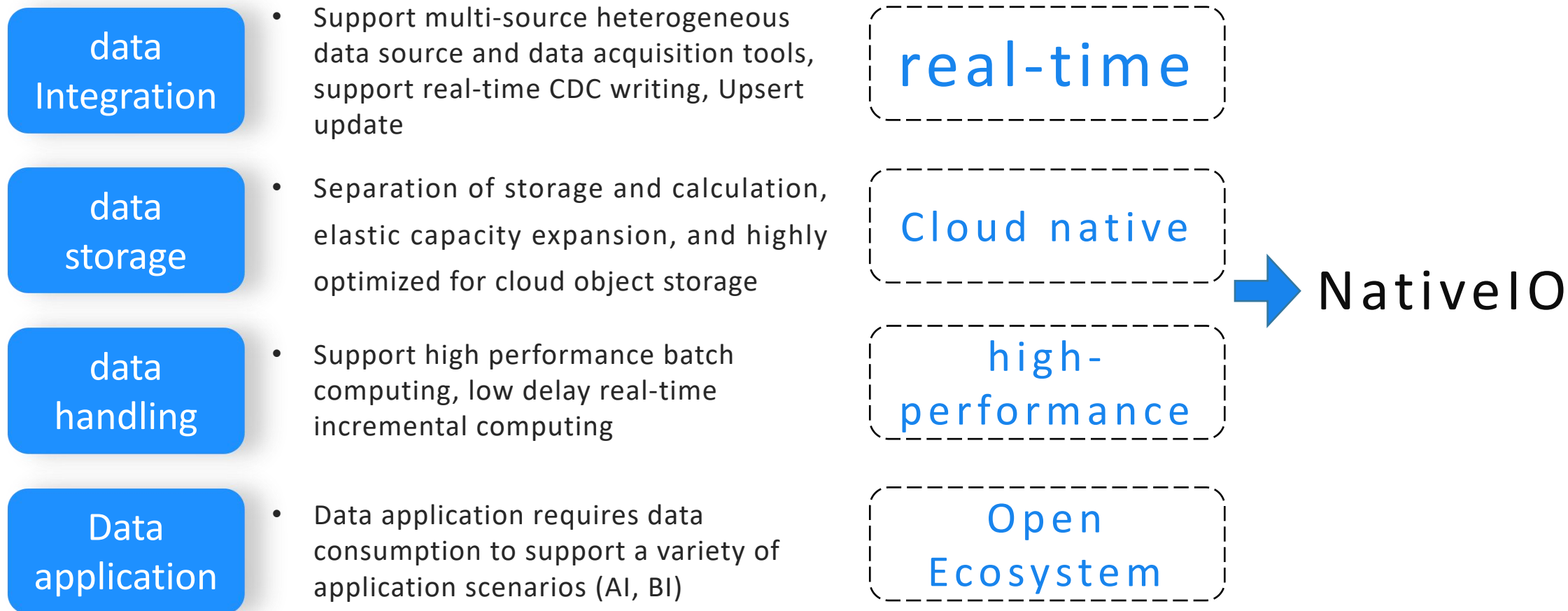
Data application

- Need to support big data, AI multiple scenarios, computing ecology

LakeSoul, Open source lake warehouse framework introduction

LakeSoul Framework solution: Native IO

layer



LakeSoul NativeIO Principles of implementation



design goal

01

Unified packaging

- Unified IO implementation, encapsulate Upsert and MOR logic
- Independent of the computational engine implementation
- Package the Java / Python interface

02

Multi-engine ecology

- Vectorized memory format, zero-copy across languages
- Vectorization computing framework, AI framework docking

03

high-performance

- For high-throughput batch reading and writing design
- Separated elastic Compaction, reduced write magnification
- Use fully the asynchronous parallel means to improve the storage system access performance

LakeSoul NativeIO Principles of implementation

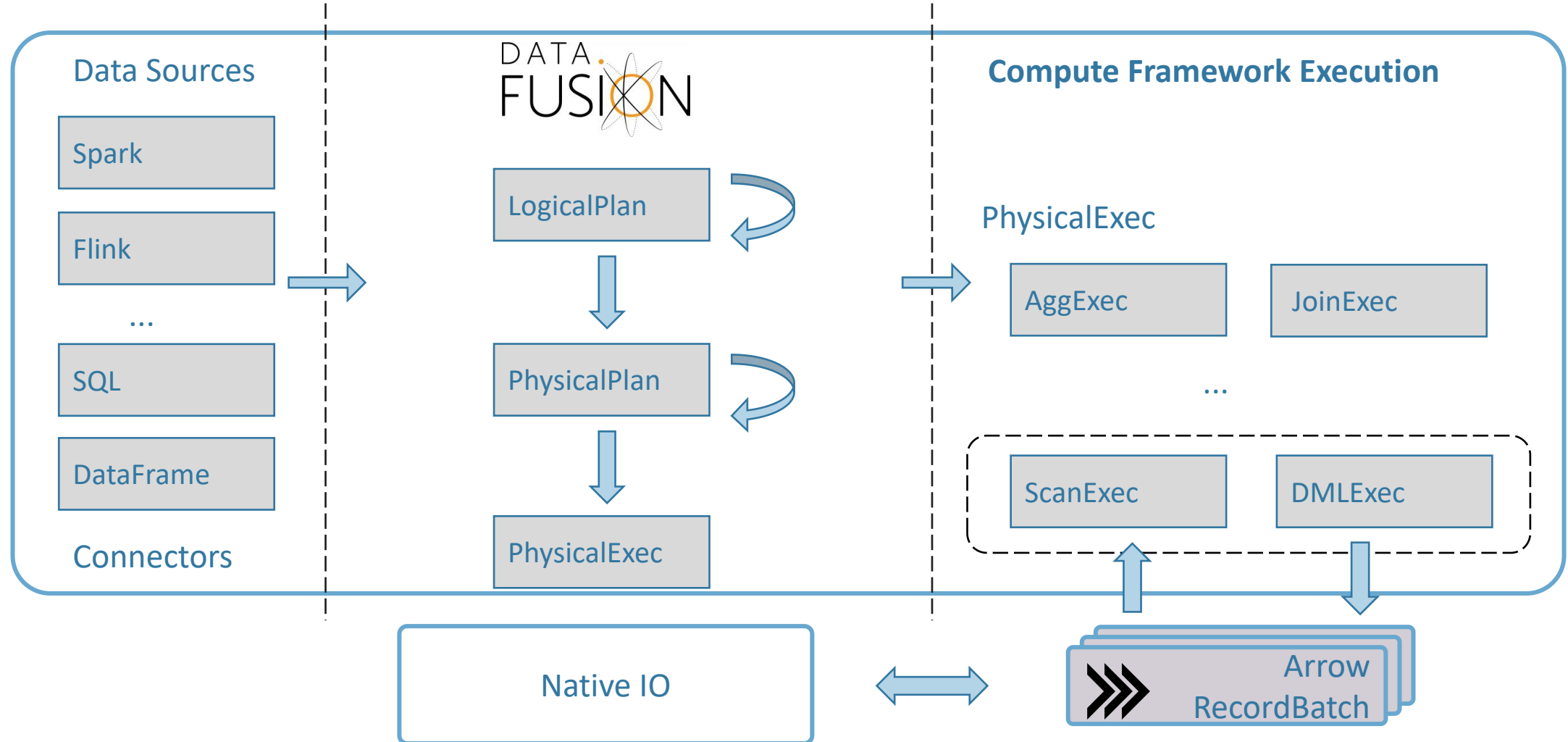


Technical selection

- Implementation language: Rust
- data format:
 - Apache Parquet (disk) + Apache Arrow (memory)
 - There are main key table, no primary key table
- Physical implementation
 - 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 Principles of implementation

NativeIO Overall architecture



LakeSoul NativeIO Principles of implementation

Table file organization format

- partition table
 - Supports multilevel range partitions, with one subdirectory per level
 - `table_path/'date=20240110'/`
- Main key table
 - Monolayer of LSM-Tree
 - Upsert When the file is divided according to the main key hash, and the main key is sorted within the shard
 - Support for reading and writing in CDC format
 - With a rowkind hidden column: I / U / D
 - Support for concurrent partial field updates (Partial Update)

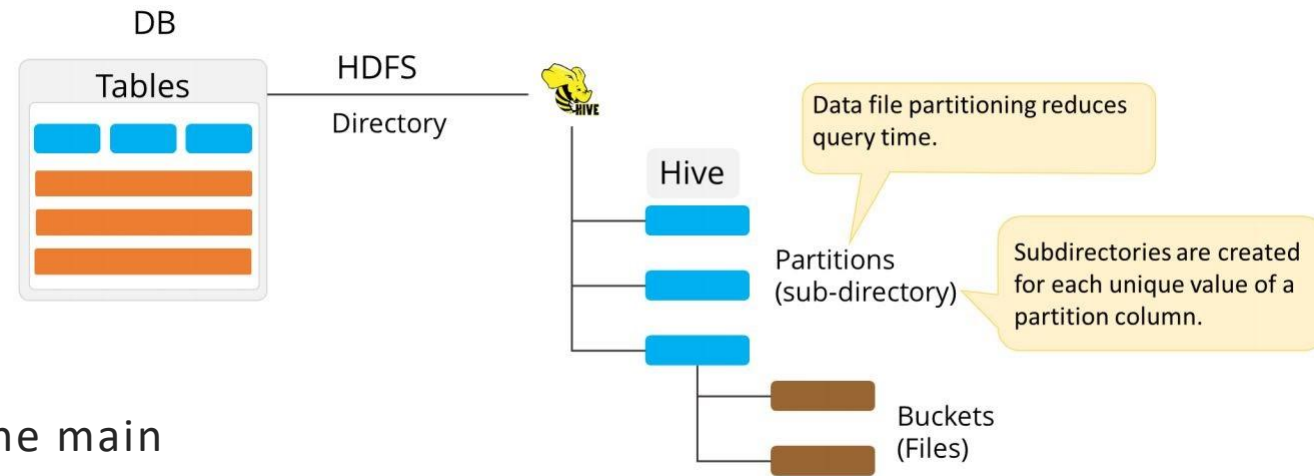
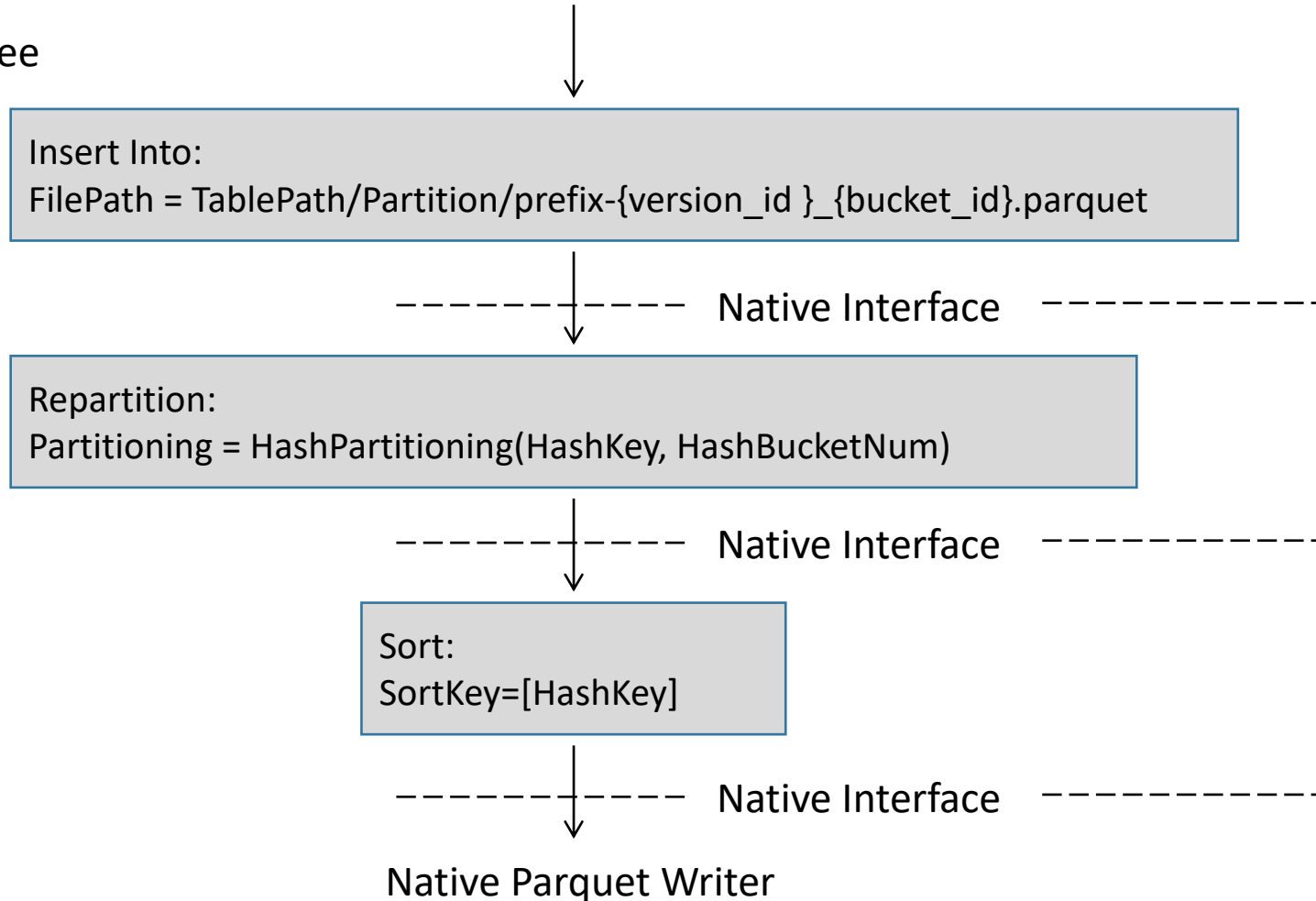


Photo credit: <https://www.simplilearn.com/tutorials/hadoop-tutorial/data-file-partitioning>

LakeSoul NativeIO Principles of implementation

Primary Key table write process

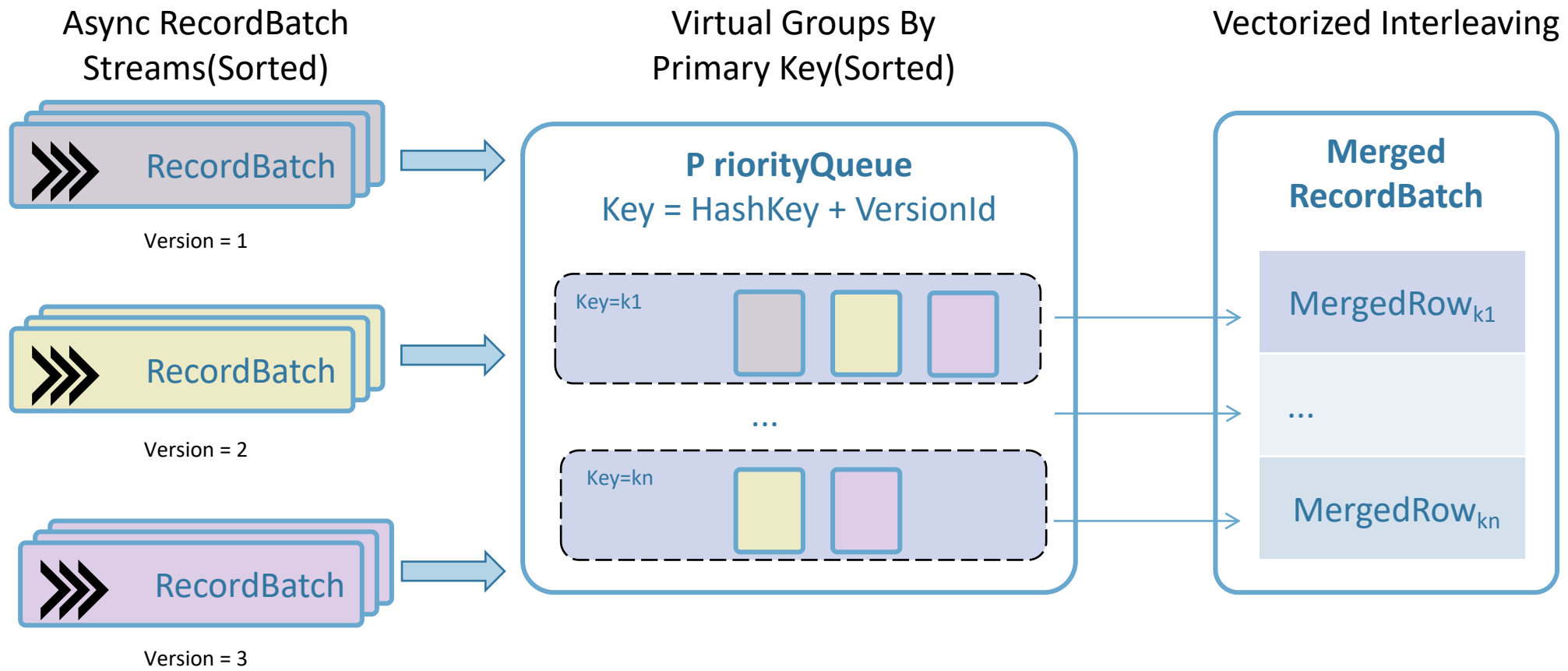
Plan Tree



- The execution position of hash slice (Repartition) and sorting can be adjusted flexibly
 - Spark: Sding and sorting are performed in Spark
 - Flink: Sding is performed in Flink and sorting is performed in NativeIO layer
- Engine global fragmentation to reduce the number of small files
- Native Level sorting uses Spill Sort to save memory
- Write path without Compaction (additional automatic Compaction service)

LakeSoul NativeIO Principles of implementation

Primary key table read process



LakeSoul NativeIO Principles of implementation

Primary key table read process

[RecordBatch],
batch_size=2

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

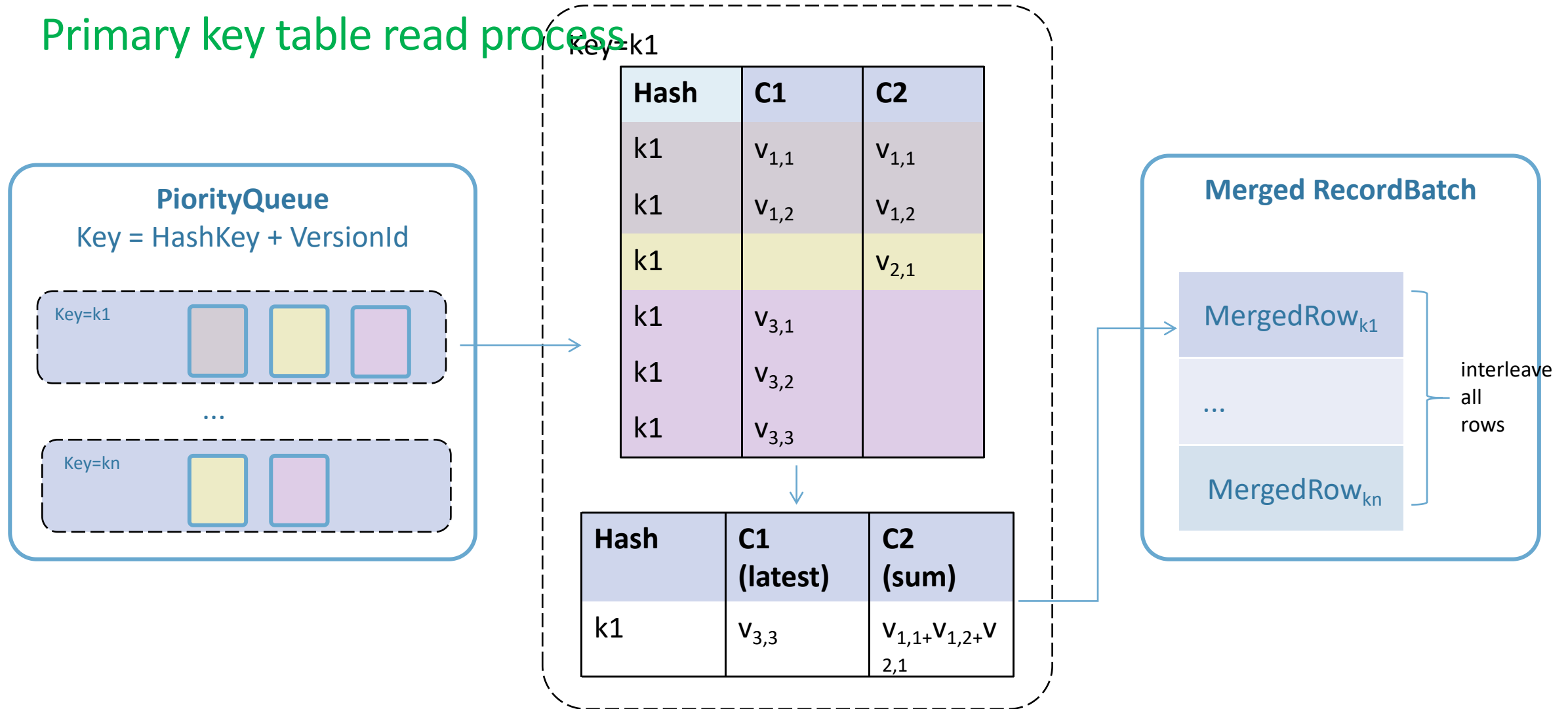
BatchRange

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

- The same primary key may come from the same batch, multiple batch per file, and a batch of different files
- Organize the same primary key together using the priority queue (record stream id, batch id, row id, no data copy)

LakeSoul NativeIO Principles of implementation

Primary key table read process



- Each group selects the line number corresponding to the latest primary key to form the {stream_id, batch_id, begin / end_row_id} tuple
- Batch final output using the Arrow Interleave operator

LakeSoul NativeIO Principles of implementation



CDC (Change Data Capture) native support

Automatically add the RowKind hidden columns

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

The deleted row is automatically filtered during the batch read
(Spark/Flink/Presto/PyTorch)

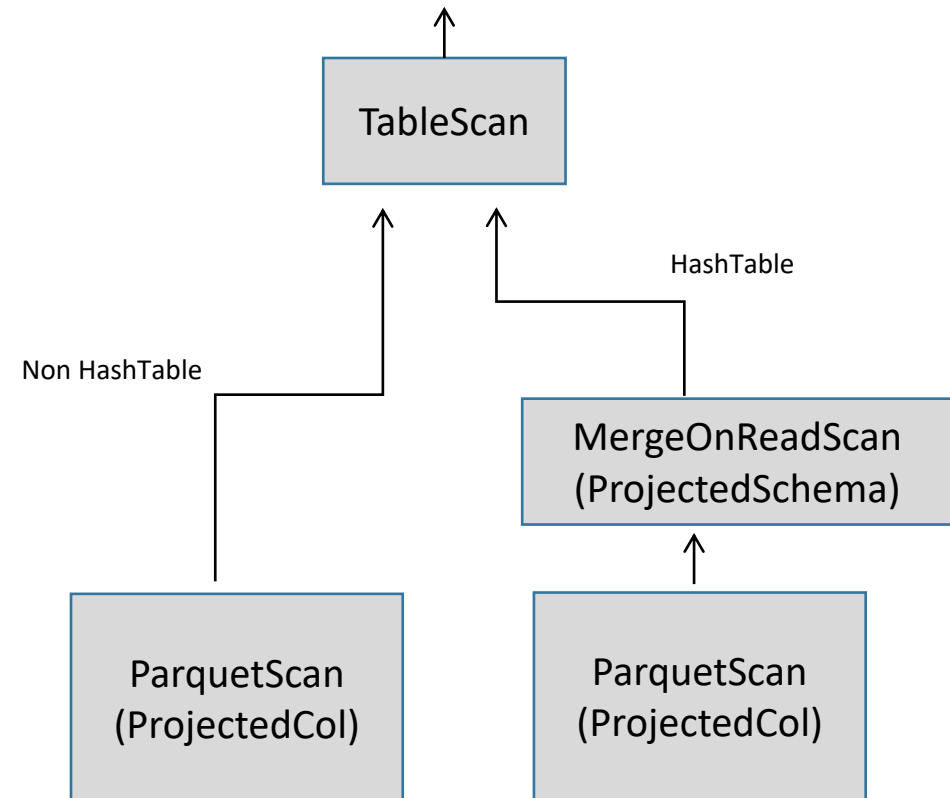
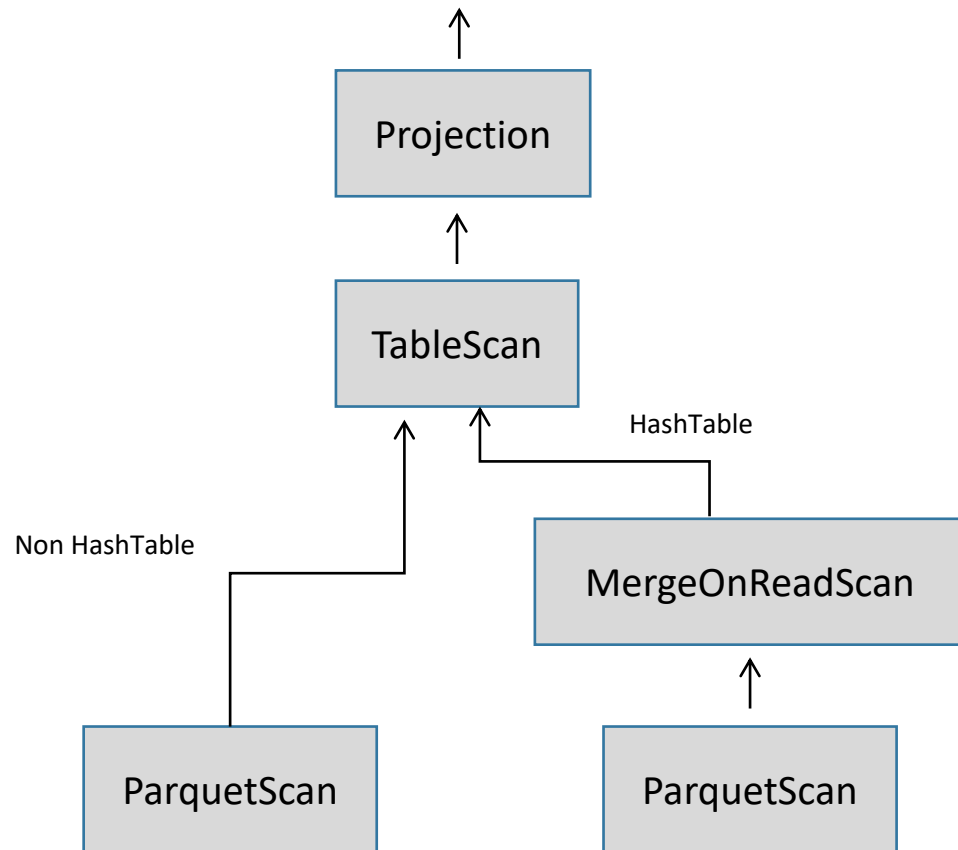
PK	Value
01	b
11	a

Flow read automatically fill Flink RowData.rowKind Field

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

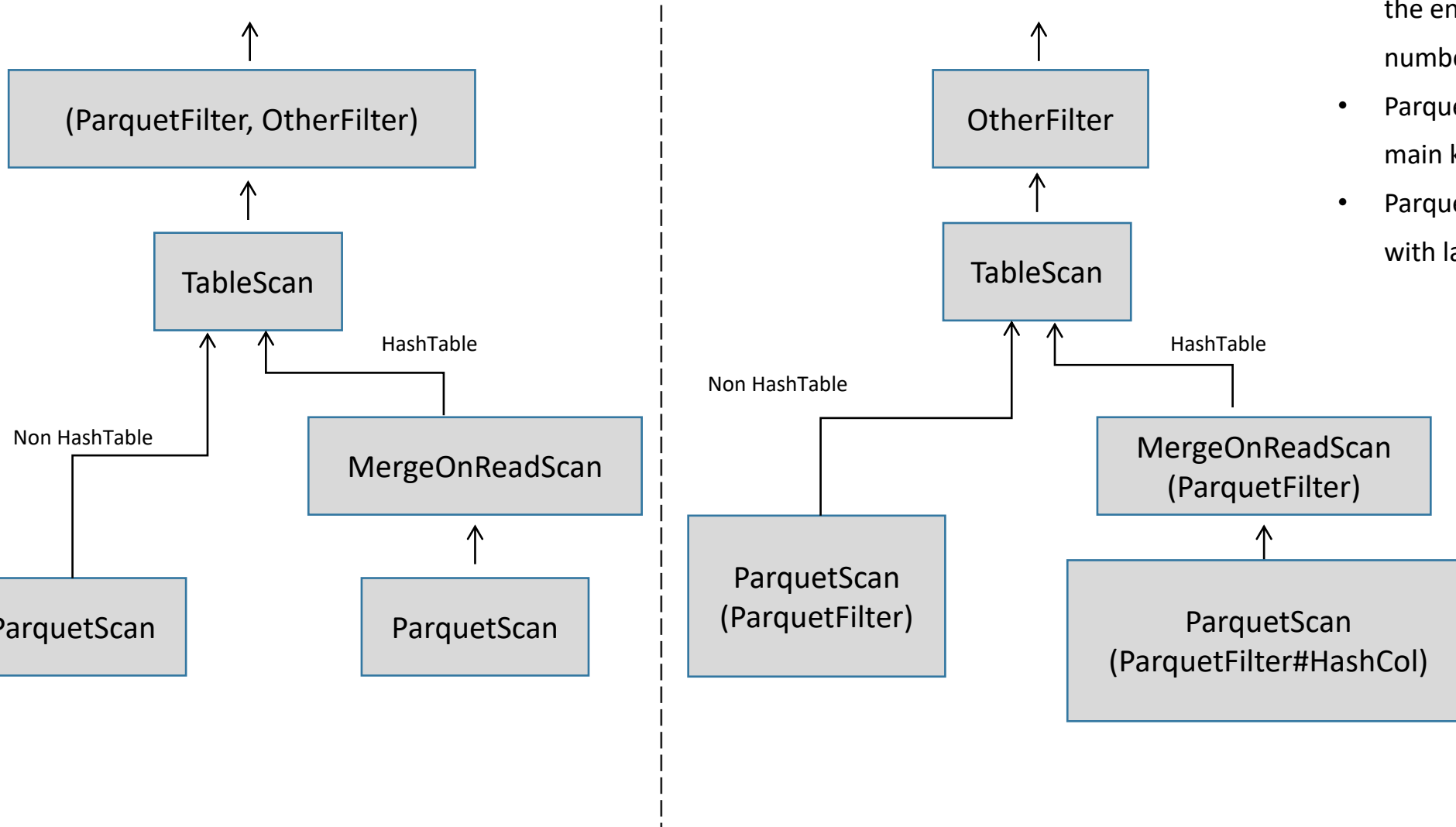
LakeSoul NativeIO Performance Optimization

Column cutting push down



LakeSoul NativeIO Performance Optimization

Filter Push it down



- Partition cropping is done in advance at the engine layer to reduce the Plan Task number
- Parquet RowGroup Stats Cropping (the main key point check effect is obvious)
- Parquet Reader Filter: Off off (effective with large proportion filtering)

LakeSoul NativeIO Performance optimization



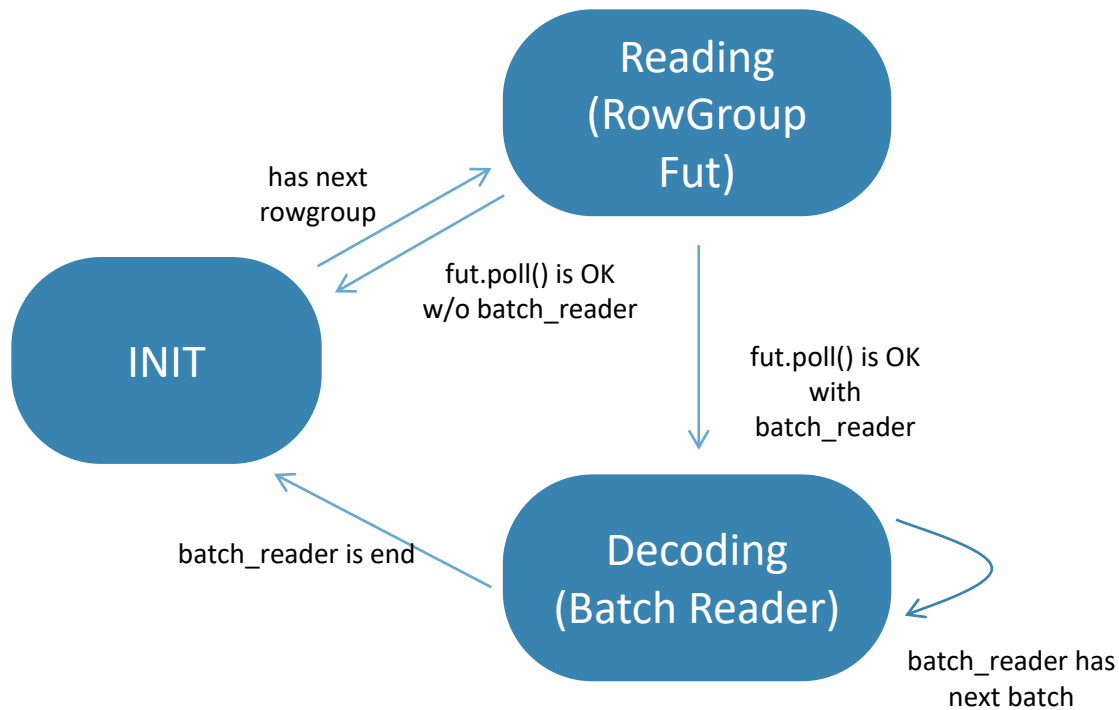
Object storage performance optimization

- Cloud Object Storage Features:
 - High bandwidth, high concurrency, and high latency
 - Single-threaded synchronization: read 30 MB/s, write 20 MB / s
- Object storage performance optimization:
 - Read the request split: ~8MB / req
 - Write the request split (Multipart Upload):> 5MB
 - Read does not cross the Part boundary: RowGroup- -Part corresponding
- Parquet File read and write optimization on the object storage
 - RowGroup Size: ~30MB
 - Asynchronously prefetch RowGroup when reading
 - Write when asynchronous concurrent upload RowGroup

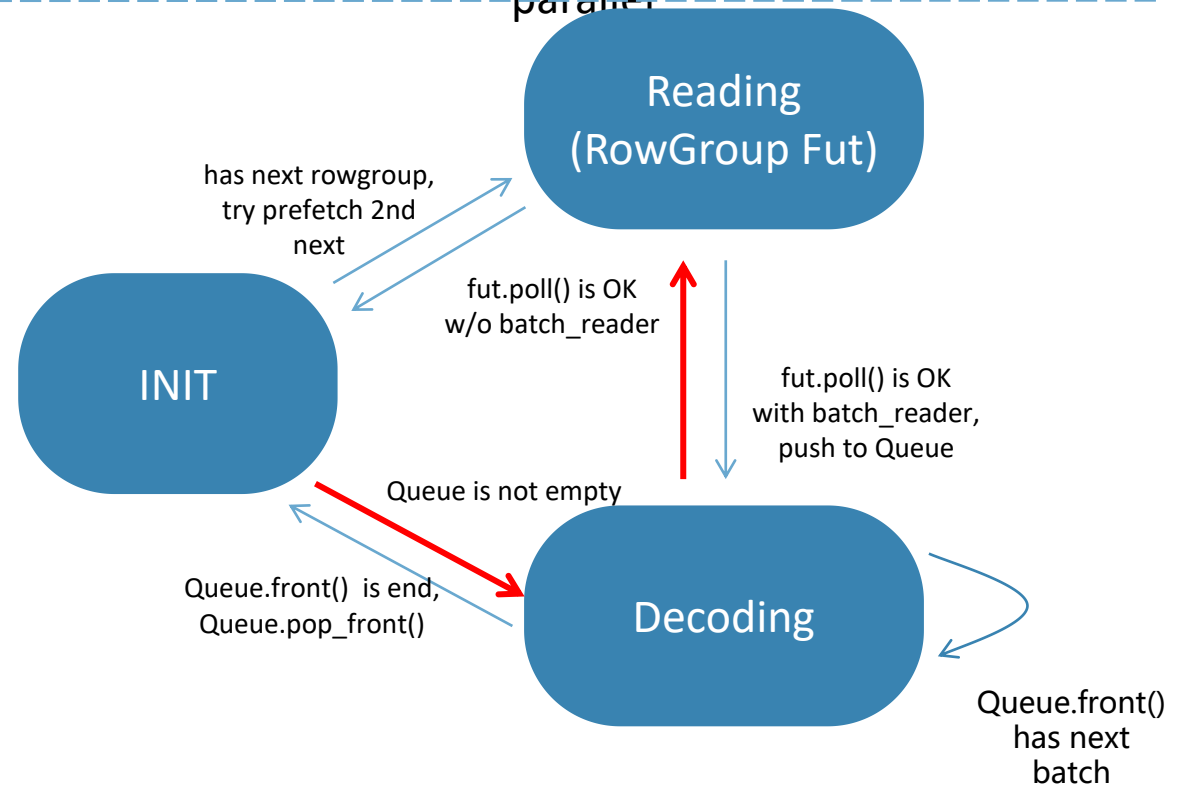
LakeSoul NativeIO Performance Optimization

Object storage and read optimization: Parquet RowGroup Prefetch

The original version, IO, and decoding were performed
alternately



The optimized version, IO, and decoding were executed in
parallel



- S3 takes 200Mbps 800Mbps

LakeSoul NativeIO Performance Optimization

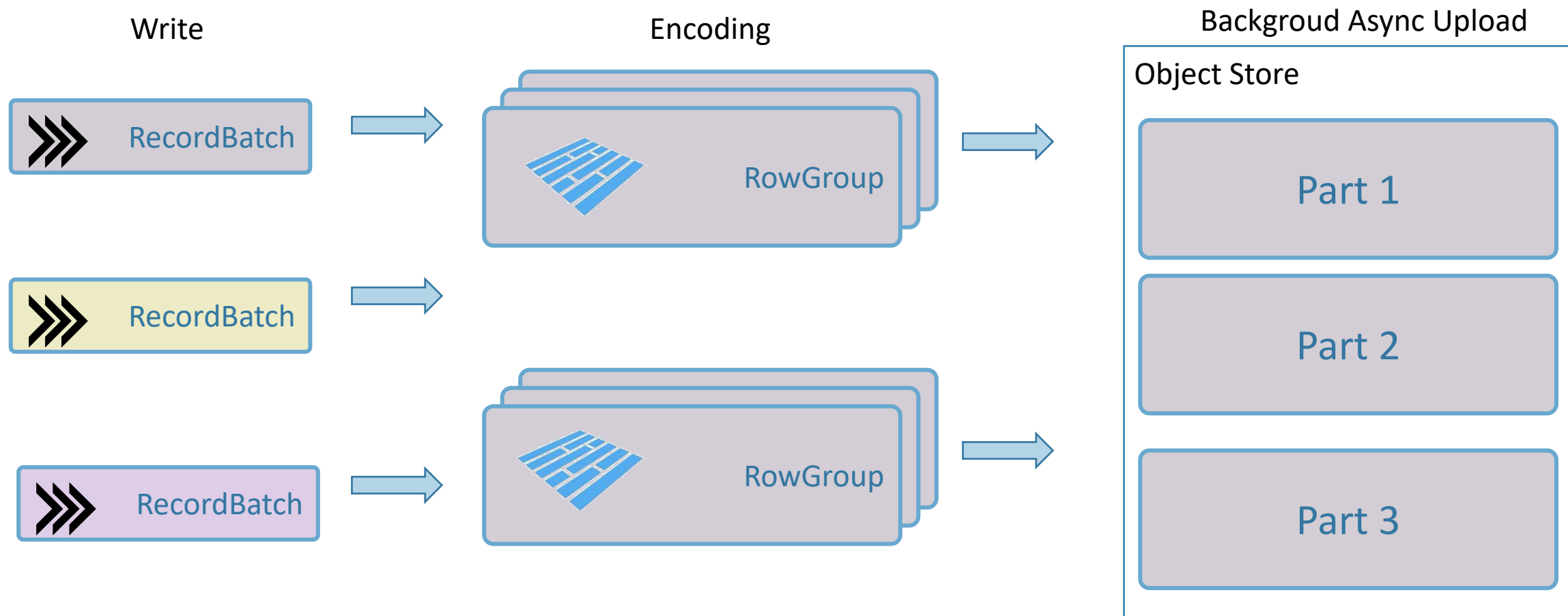
DataFun.

LakeSoul

数元灵
DMetaSoul

Object storage and write optimization: Parquet RowGroup Parallel Multipart

Upload

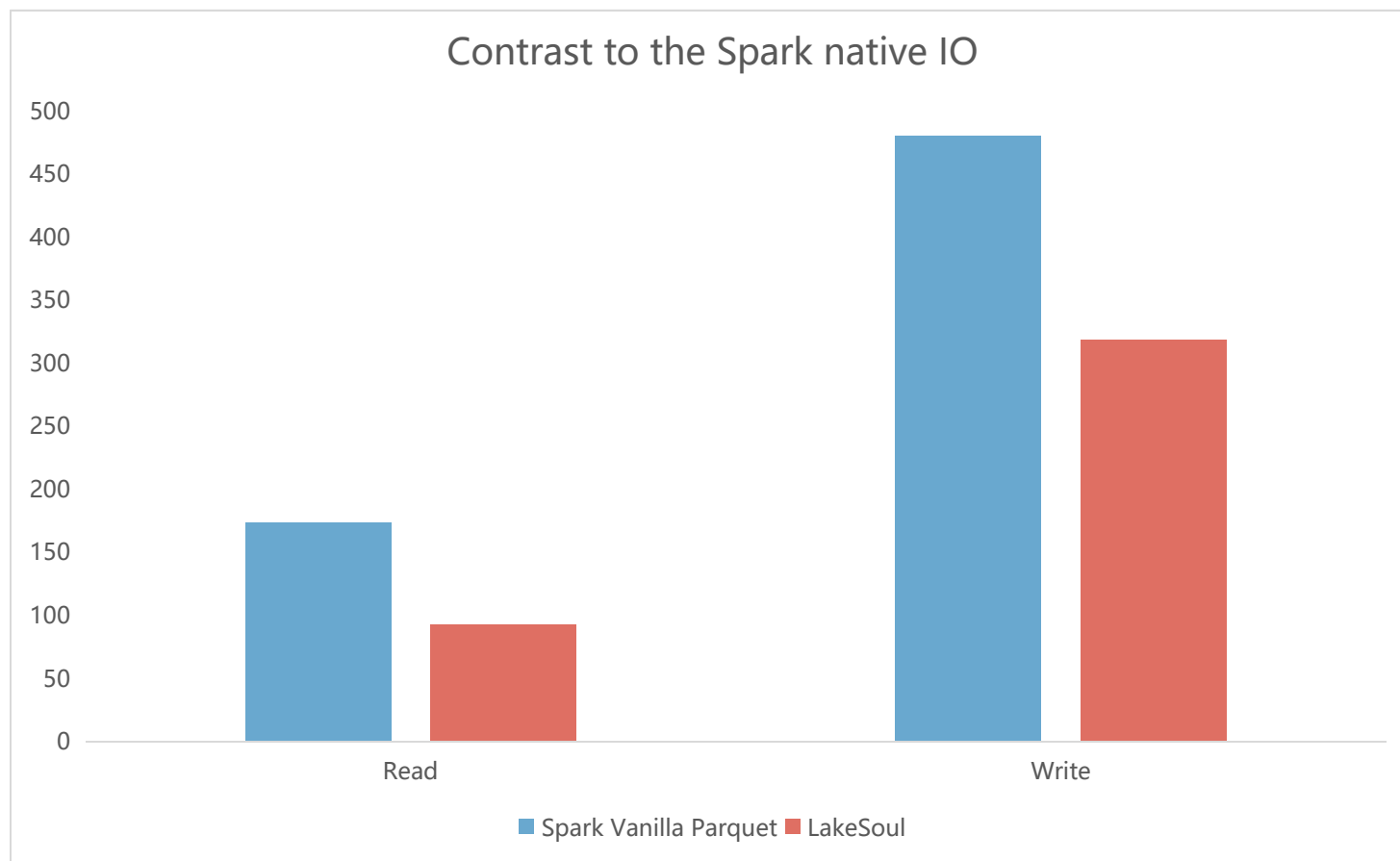


- 100Mbps 300Mbps bandwidth for single core write S3

LakeSoul NativeIO Application practice



IO Benchmark



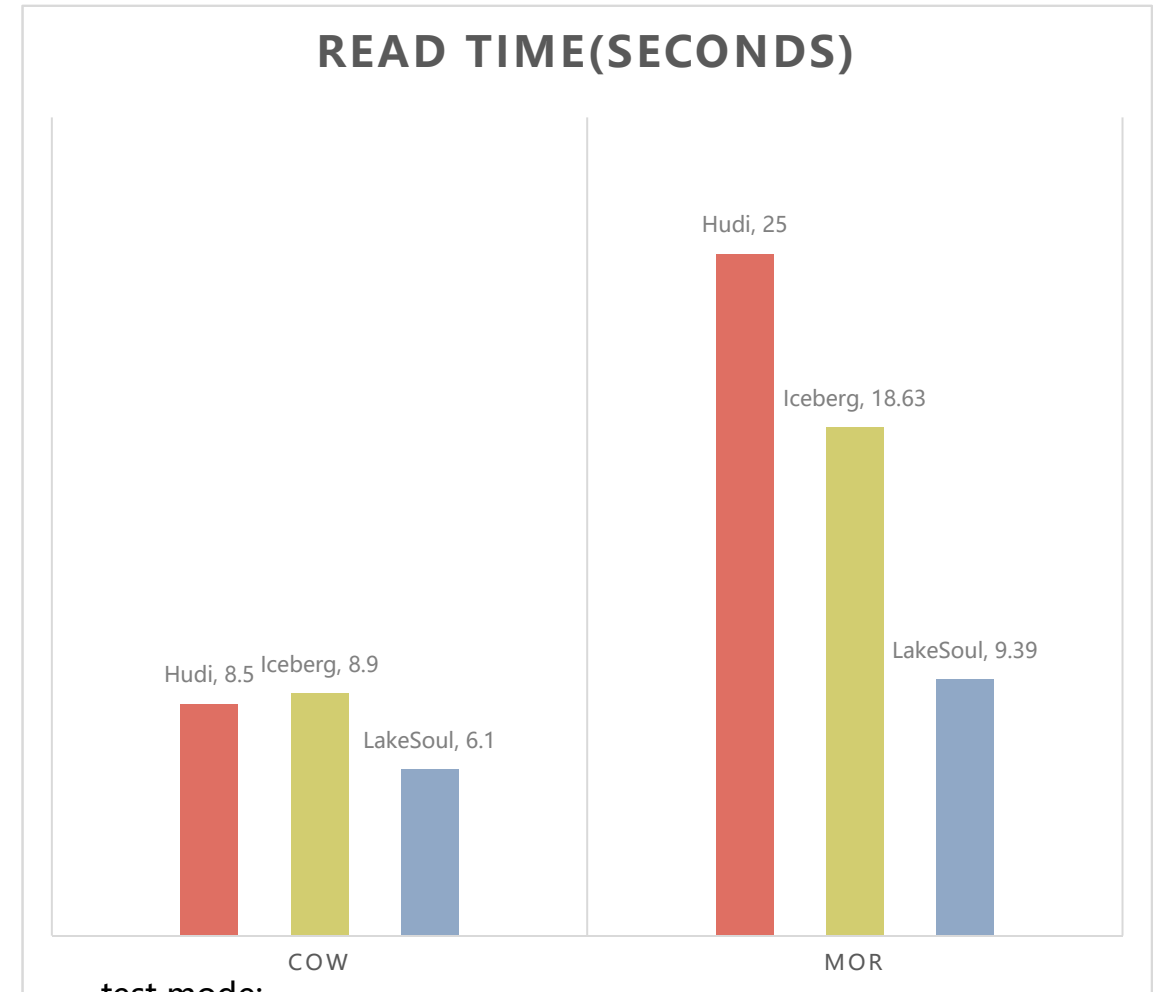
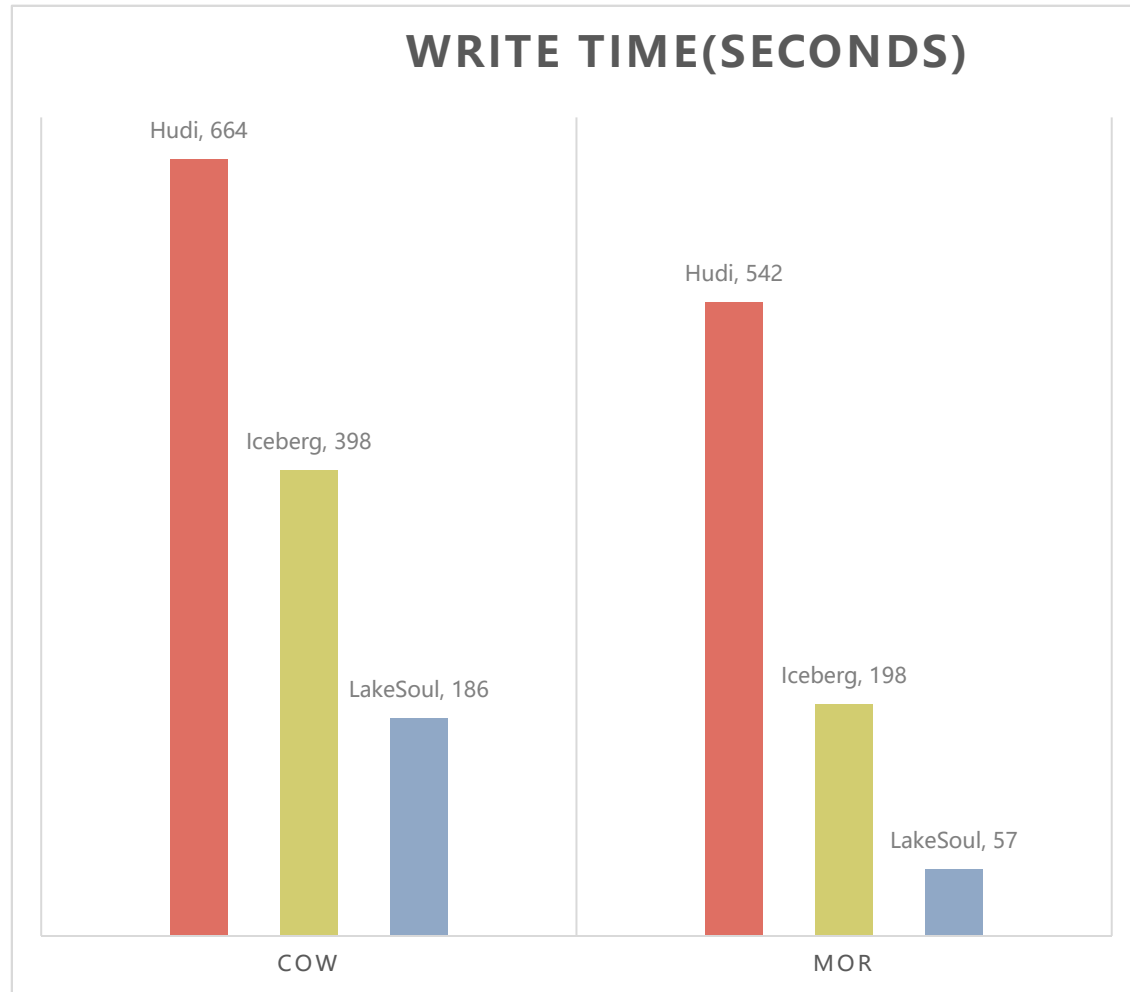
test mode:

- TPC-H-SF100 Orders table, 150 million rows, read after writing to S3
- Spark 1c8g

LakeSoul NativeIO Application practice



COW / MOR read and write Benchmark



test mode:

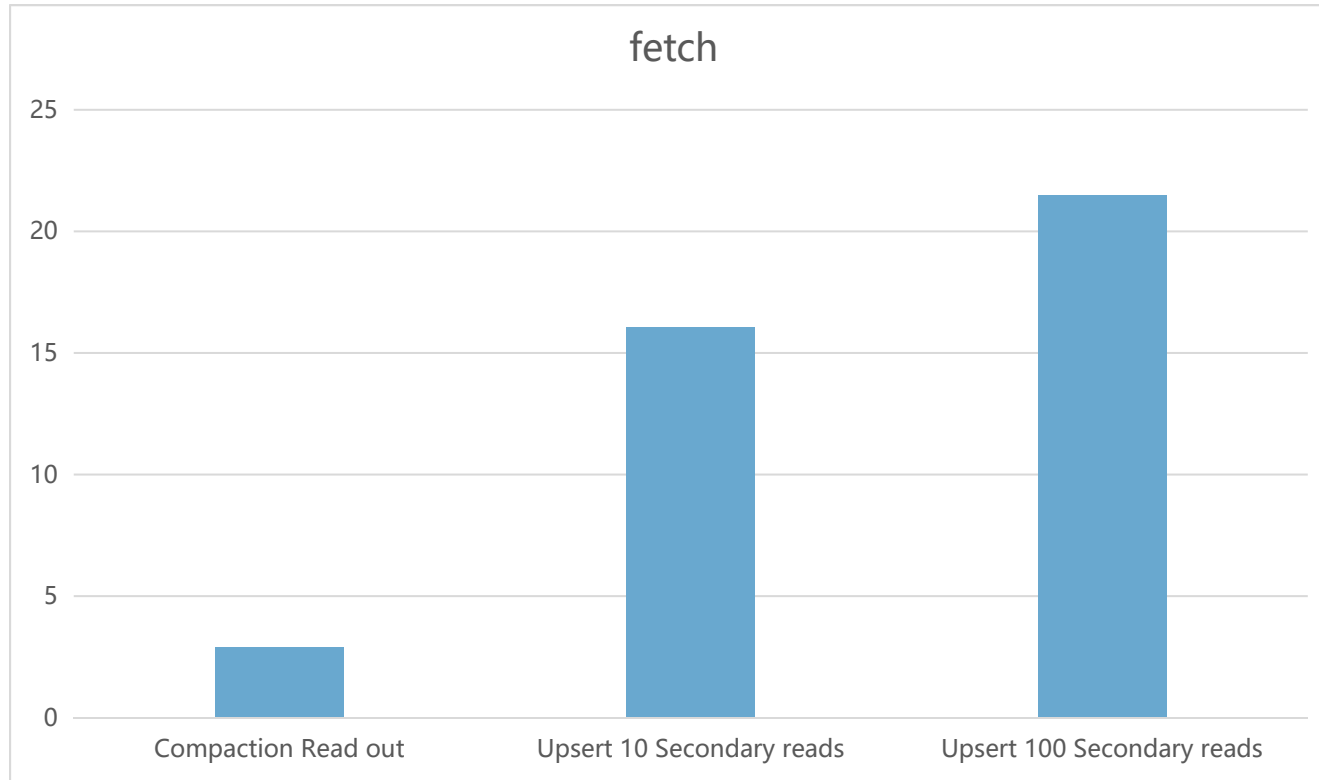
- The first batch of inserted 1000, ten thousand lines of data
- 10 Upsert 100 million lines of data
- No Compaction was performed with the MOR read

<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 Application practice



The MOR small file reads the Benchmark



- **Document size: 43MB (10 times), 4.8MB (100 times)**
- Read to merge 10 files: 16s
- Read to merge 100 files: 21.5s
- Combining 100 files takes time to increase by 30%

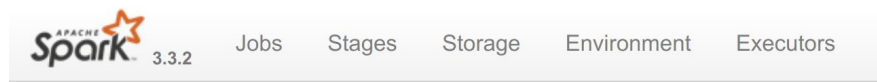
test mode:

- The first batch of inserted 1000, ten thousand lines of data
- 10 times and 100 times Upsert 100 million lines of data
- No Compaction was performed with the MOR read

LakeSoul NativeIO Application practice



Vectorization engine: Spark Gluten



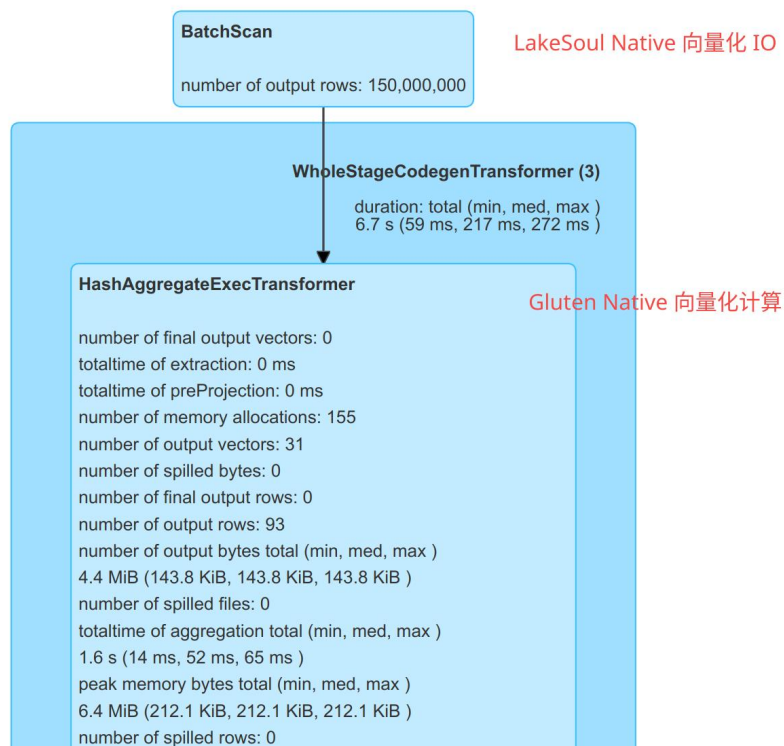
Details for Query 2

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

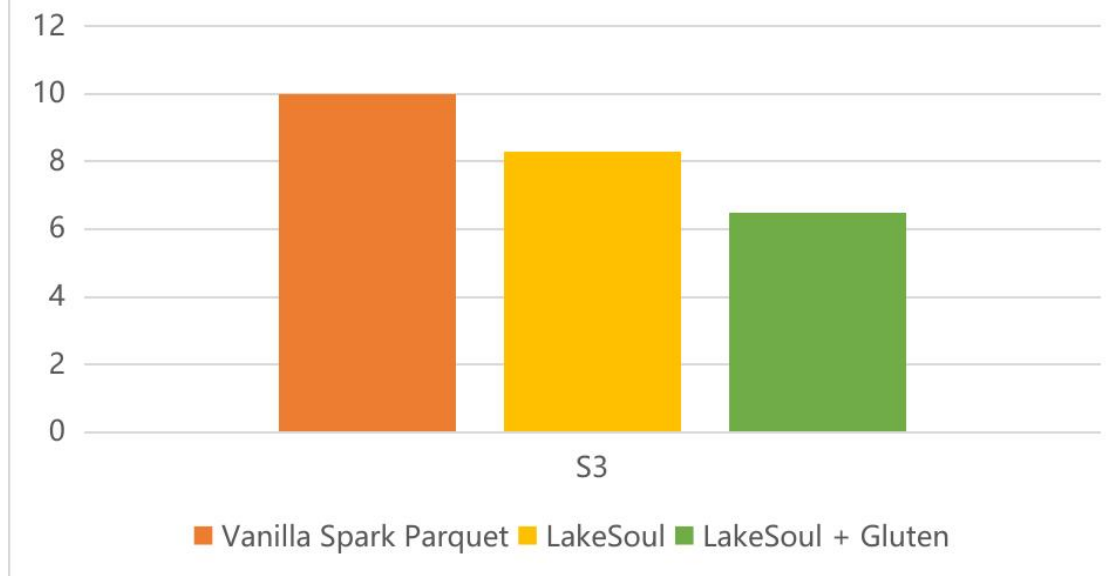
Duration: 1.0 s

Succeeded Jobs: 2 3

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

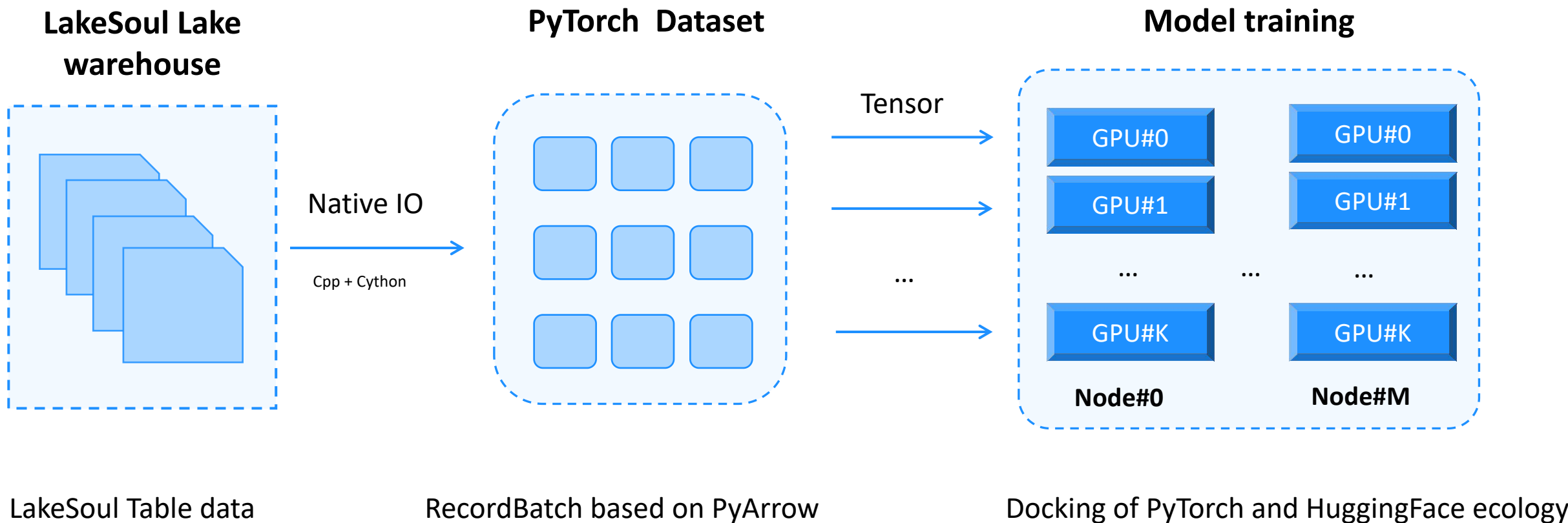


查询性能对比（单位：秒）



LakeSoul NativeIO Application practice

The AI training reads from the LakeSoul table



LakeSoul NativeIO Application practice



AI and Data Science computing ecology: 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'})
```

Support for a distributed training environment

LakeSoul NativeIO Application practice



AI large model training

<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 Recent development plan



- **function**

- Pluggable WAL support
Sub-second-level real-time visibility
- Real-time data quality verification
- Hadoop / K8s automated deployment
- Front-end of the data development platform

- **organism's habits**

- Support for more database entry into the lake
- Kafka Connect Sink
- LogStash Sink

- **function**

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

[GitHub: https://github.com/lakesoul-io/LakeSoul](https://github.com/lakesoul-io/LakeSoul)



thanks

Quick experience:

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