DATA+AI SUMMIT
BY databricks

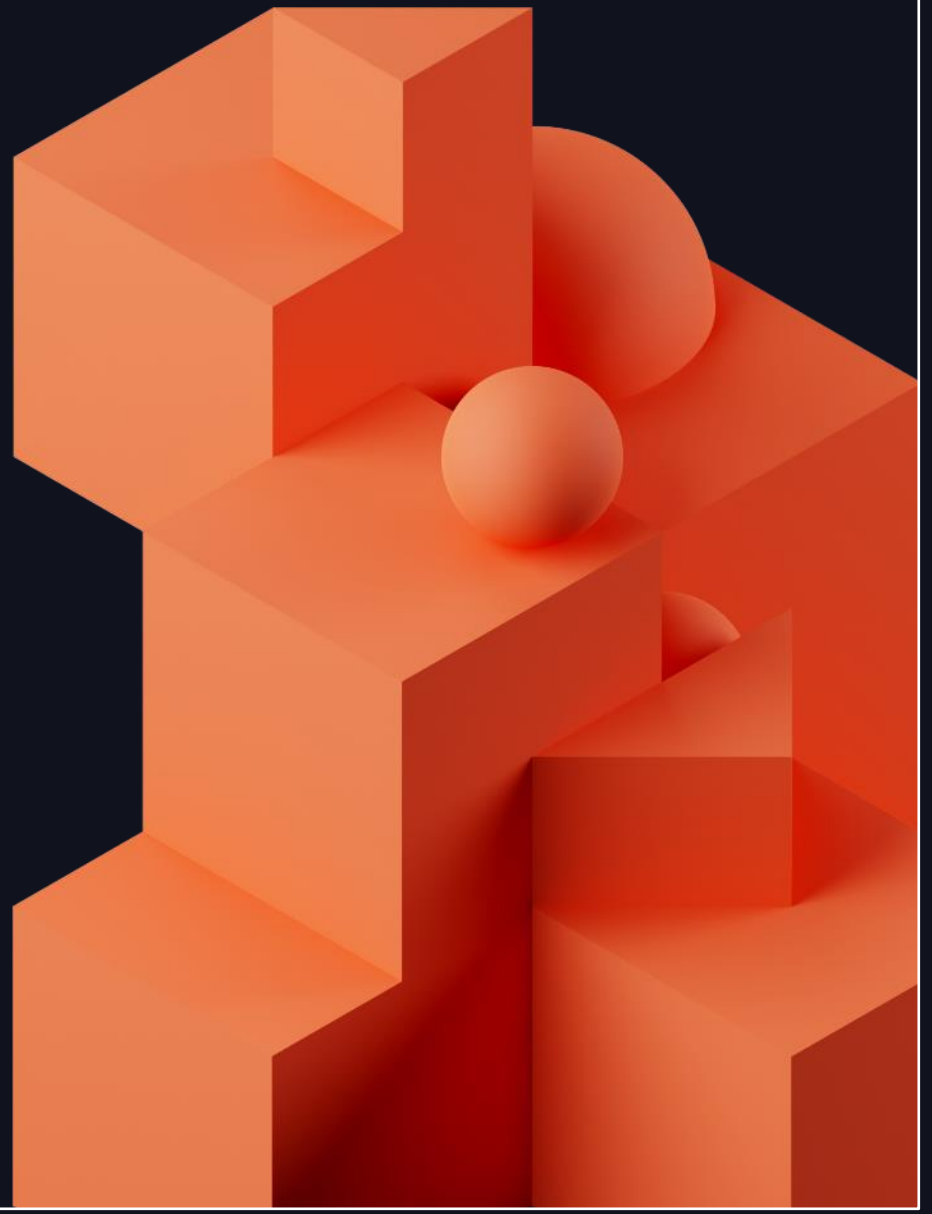
Best Exploration of Columnar Shuffle Design

Binwei Yang

Rong Ma

Chengcheng Jin

Intel Spark Team
2023



About Us

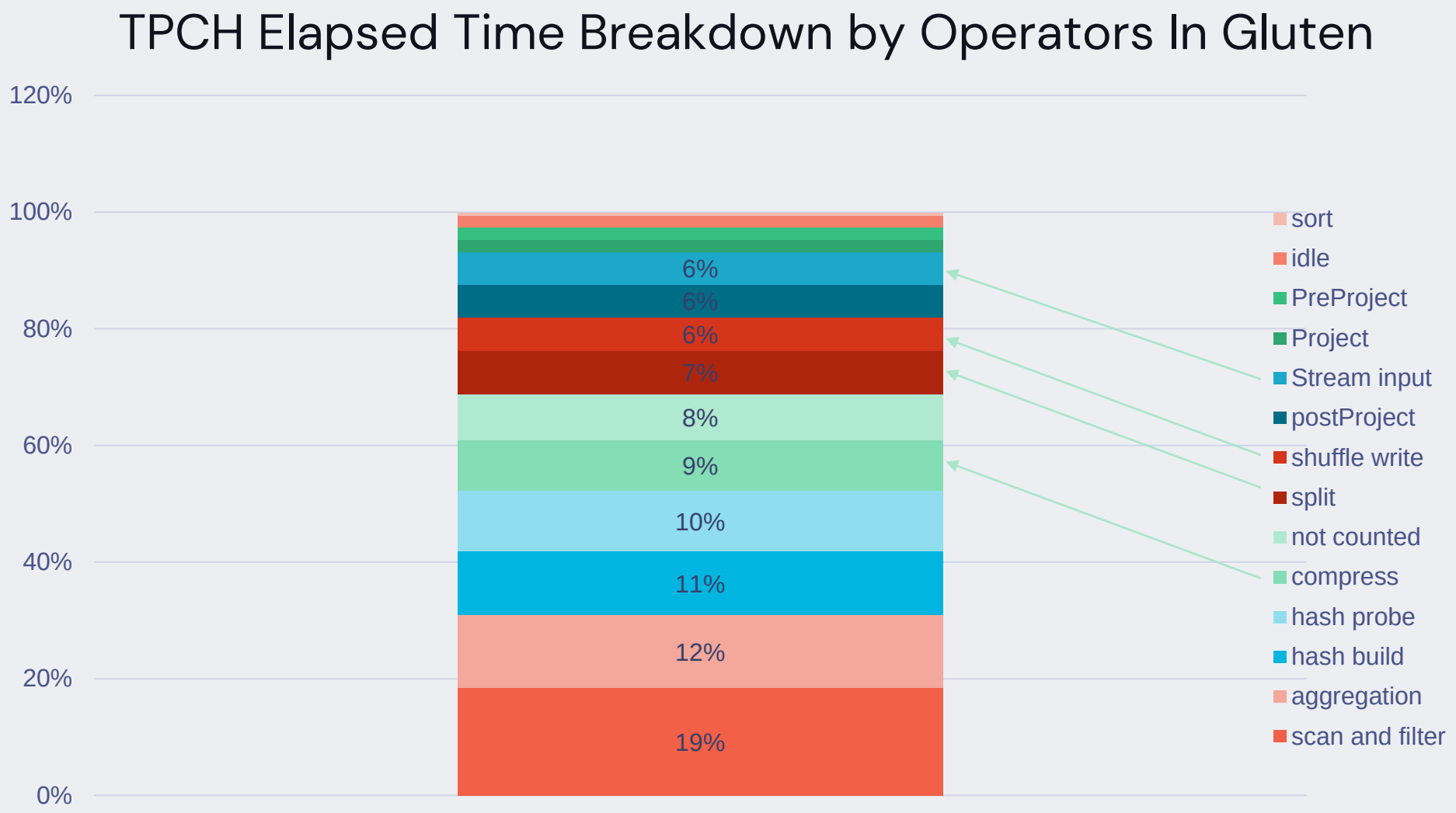
- Intel Big Data Analytics Team
- Start Gazelle Project in May. 2020. TPCH boosts by 1.8x, TPCDS 1.5x
- Transform to Gluten Project in Jan. 2022.
- Velox backend, latest perf: TPCH boosts by 3x, TPCDS 2.5x
- Gazelle uses Apache Arrow based in house designed SQL engine
- Gluten is a thin layer to offload Spark SQL engine to 3rd library like Velox and Clickhouse using Substrait to pass query plan

Gluten: A Middle Layer to Offload Spark SQL to Native Engines for Execution Acceleration

https://www.youtube.com/watch?v=0Q6gHT_N-1U

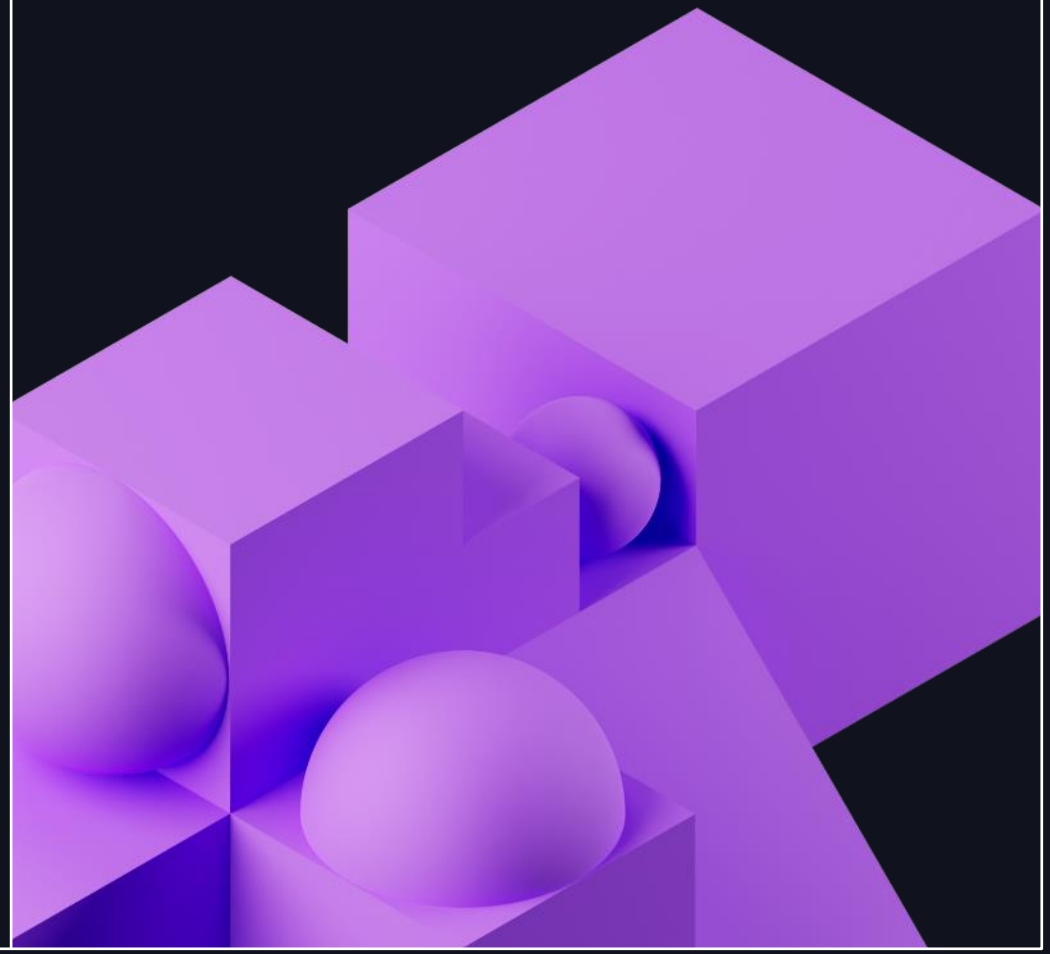
https://github.com/oap-project/gluten

Why We Care about Shuffle?

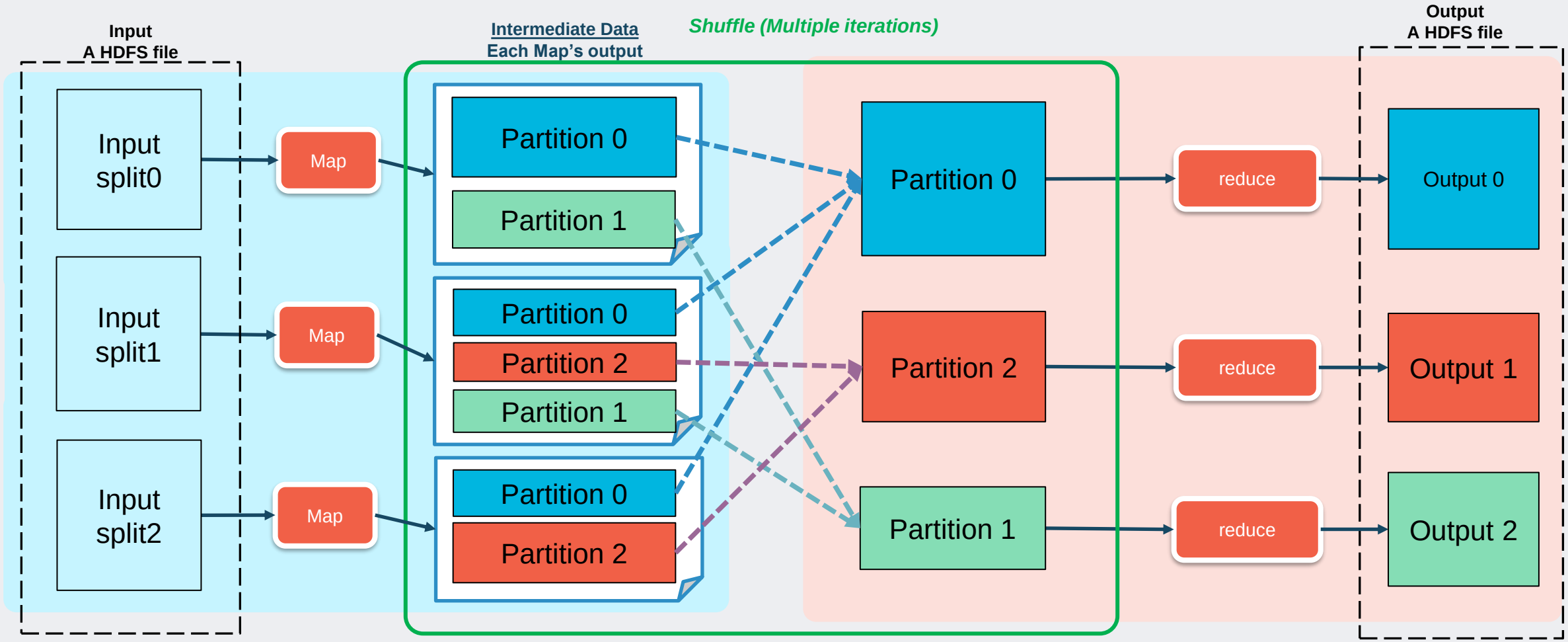


Shuffle related operations takes up to 28% of total elapsed time

Spark Shuffle



Spark Shuffle



*Best Practice of Compression Codecs in Spark - Sophia Sun, etc.

Spark Shuffle Operations

Mapper

Load

Persistent

Spill

Compress

Serialize

Split

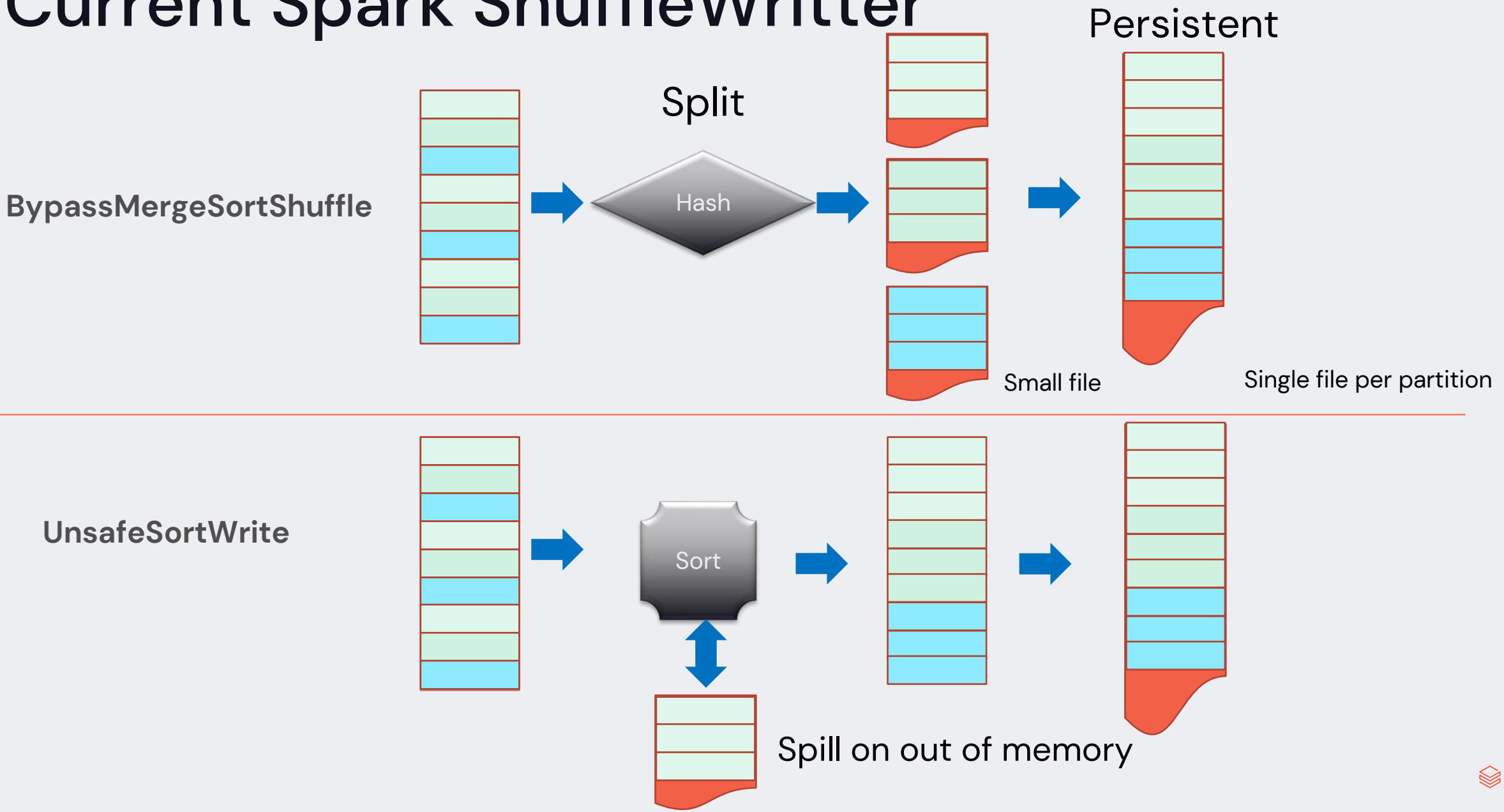
Reducer

Fetch

Decompress

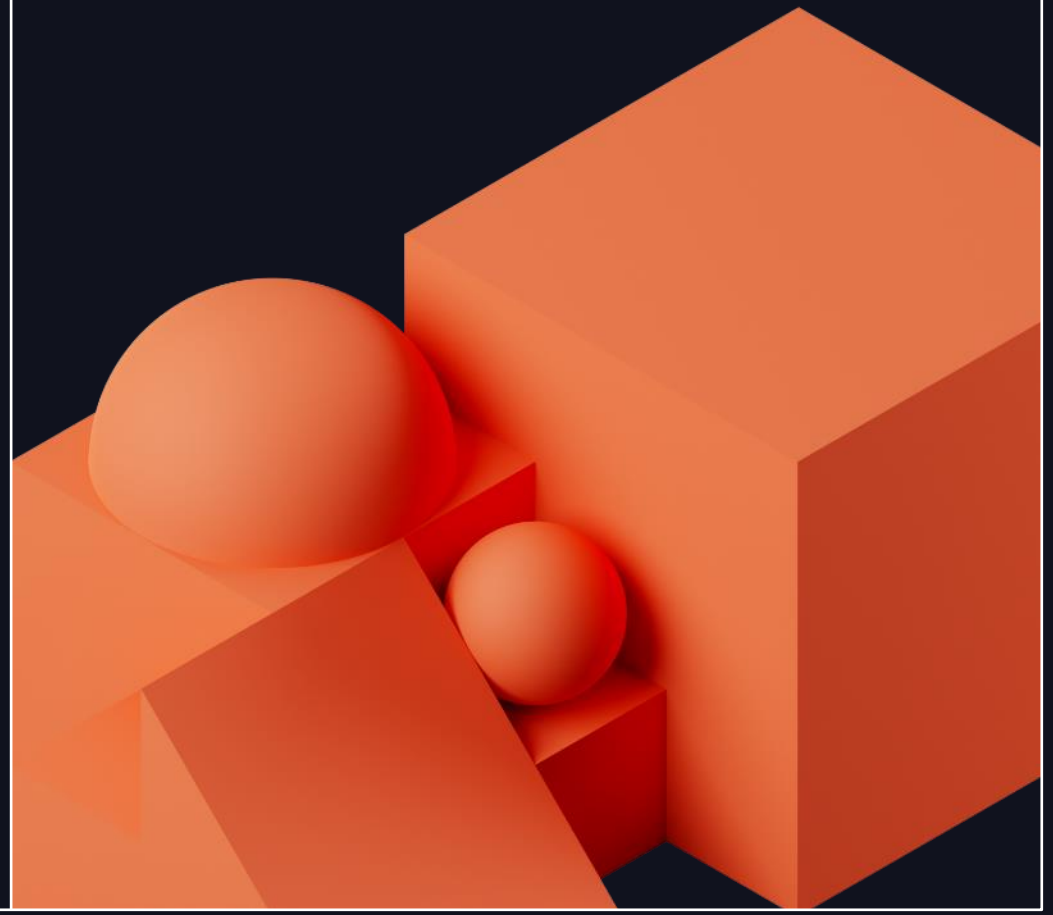
Deserialize

Current Spark ShuffleWritter



Columnar Shuffle

1_DAIS_Title_Slide



Columnar Shuffle Changes

Mapper

Load

Persistent

Spill

Compress

Serialize

Split

Reducer

Fetch

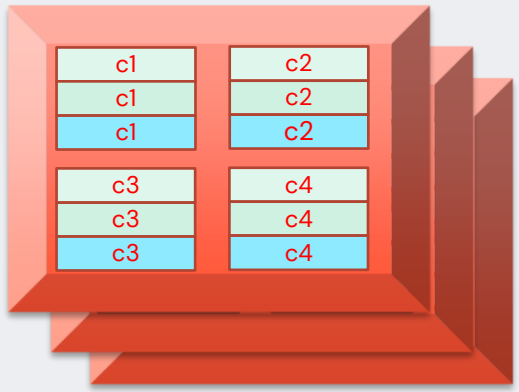
Decompress

Deserialize

Hash Based Columnar Shuffle (1)

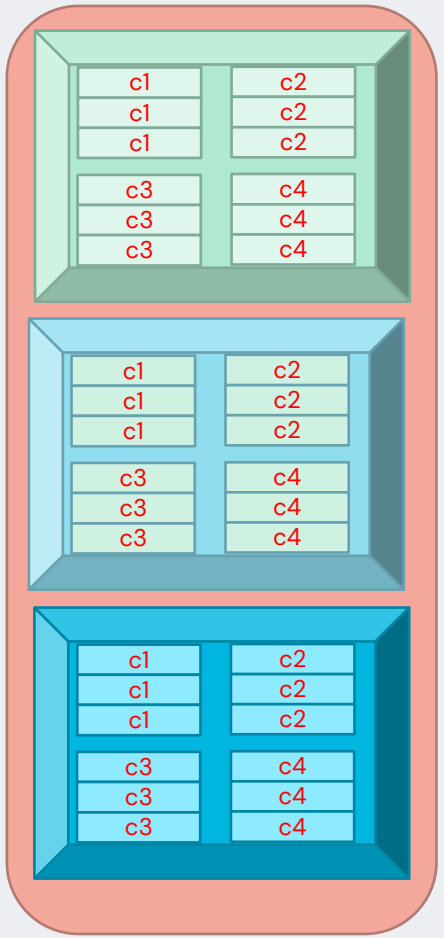
Split

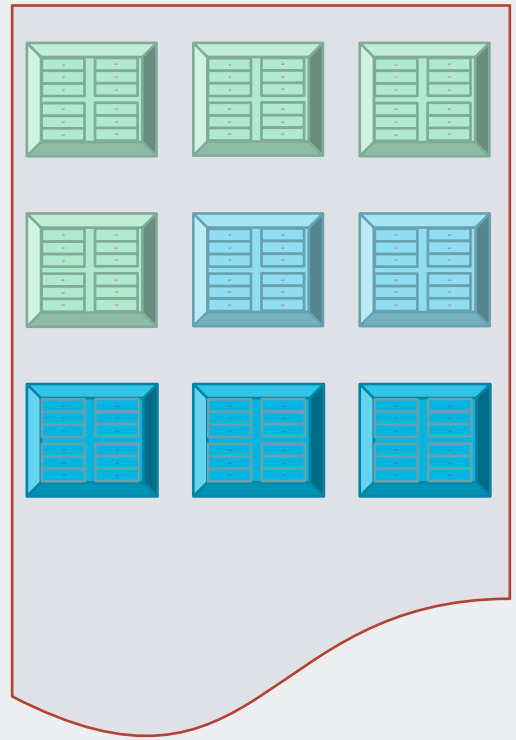
RecordBatch



Split

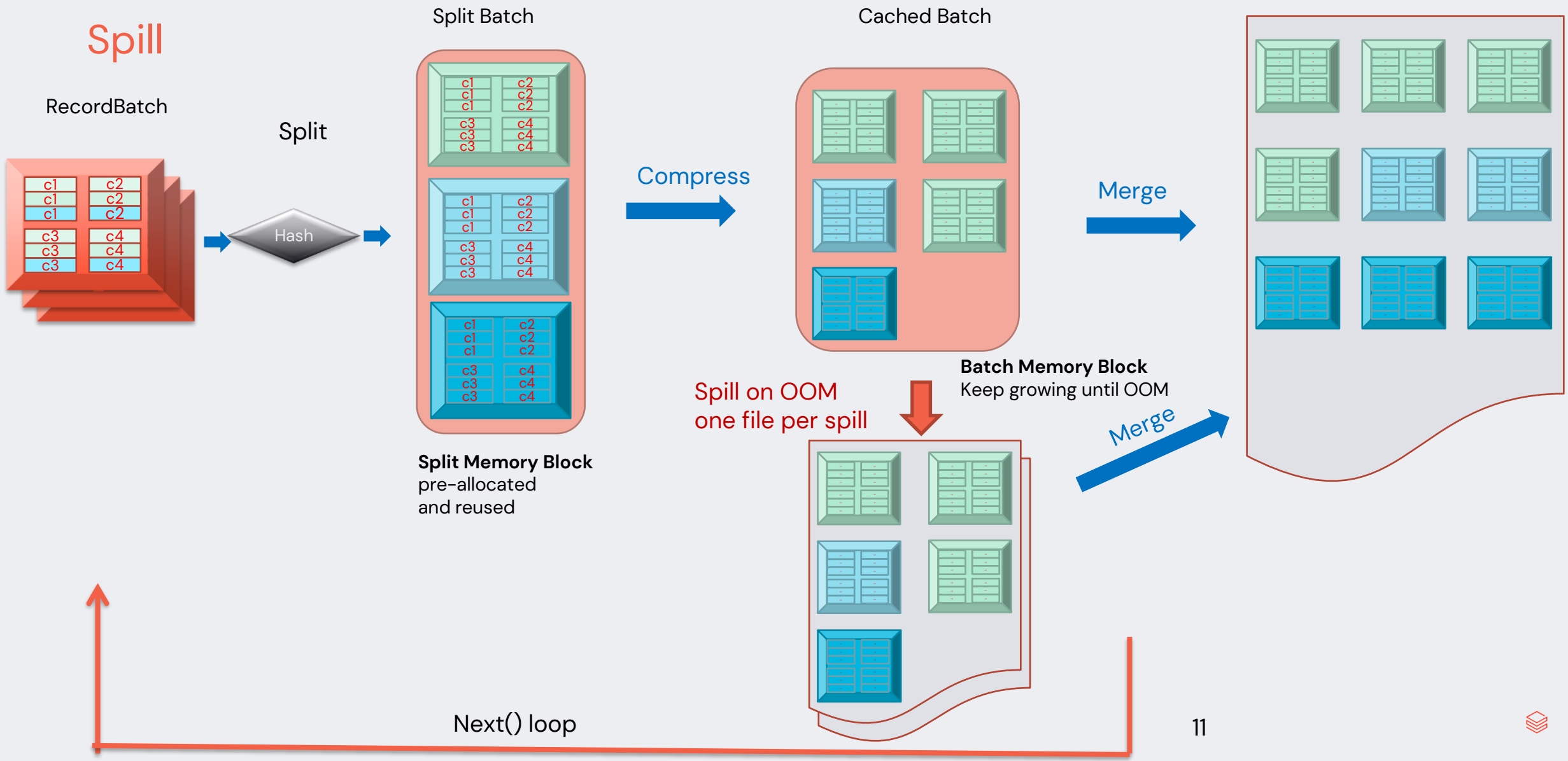
RecordBatch



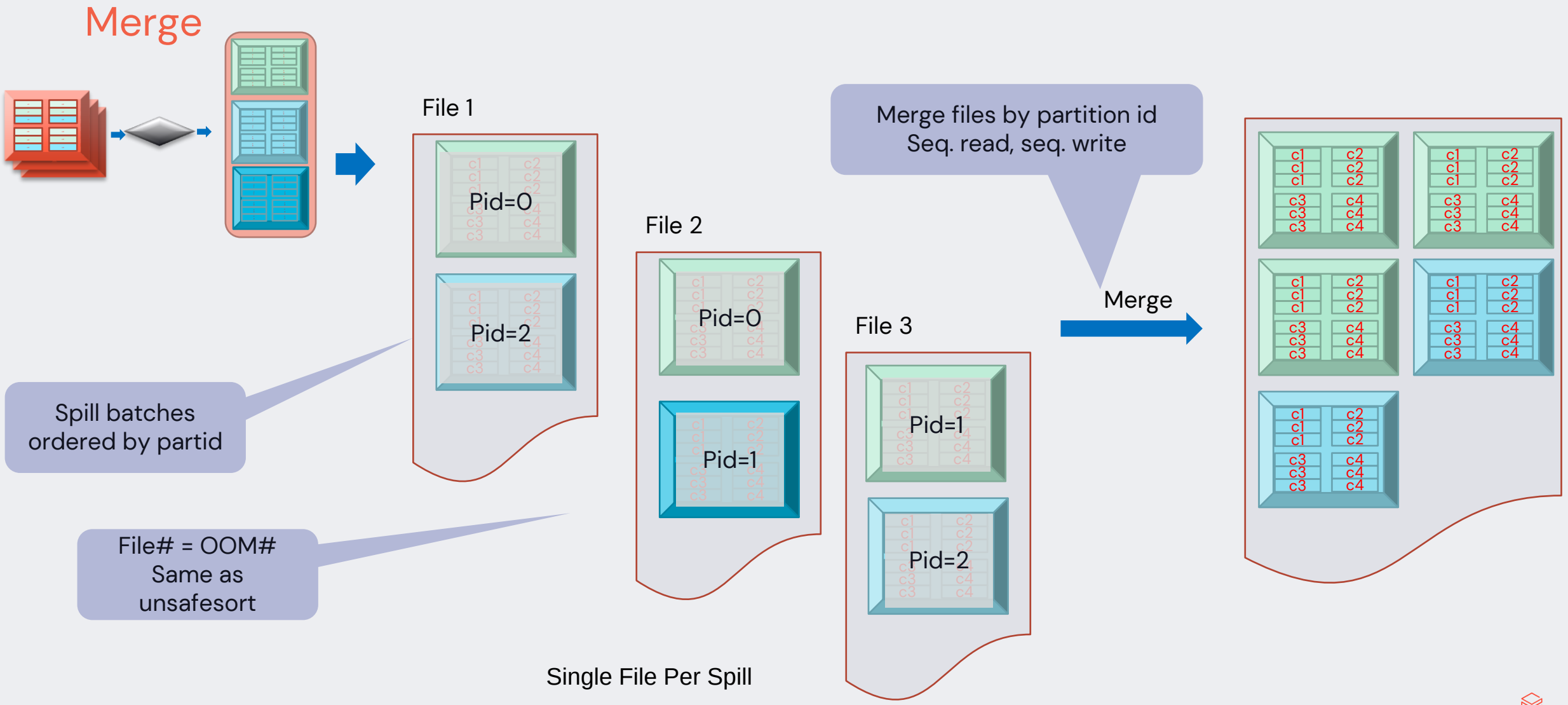


Memory block is pre-allocated
And reused

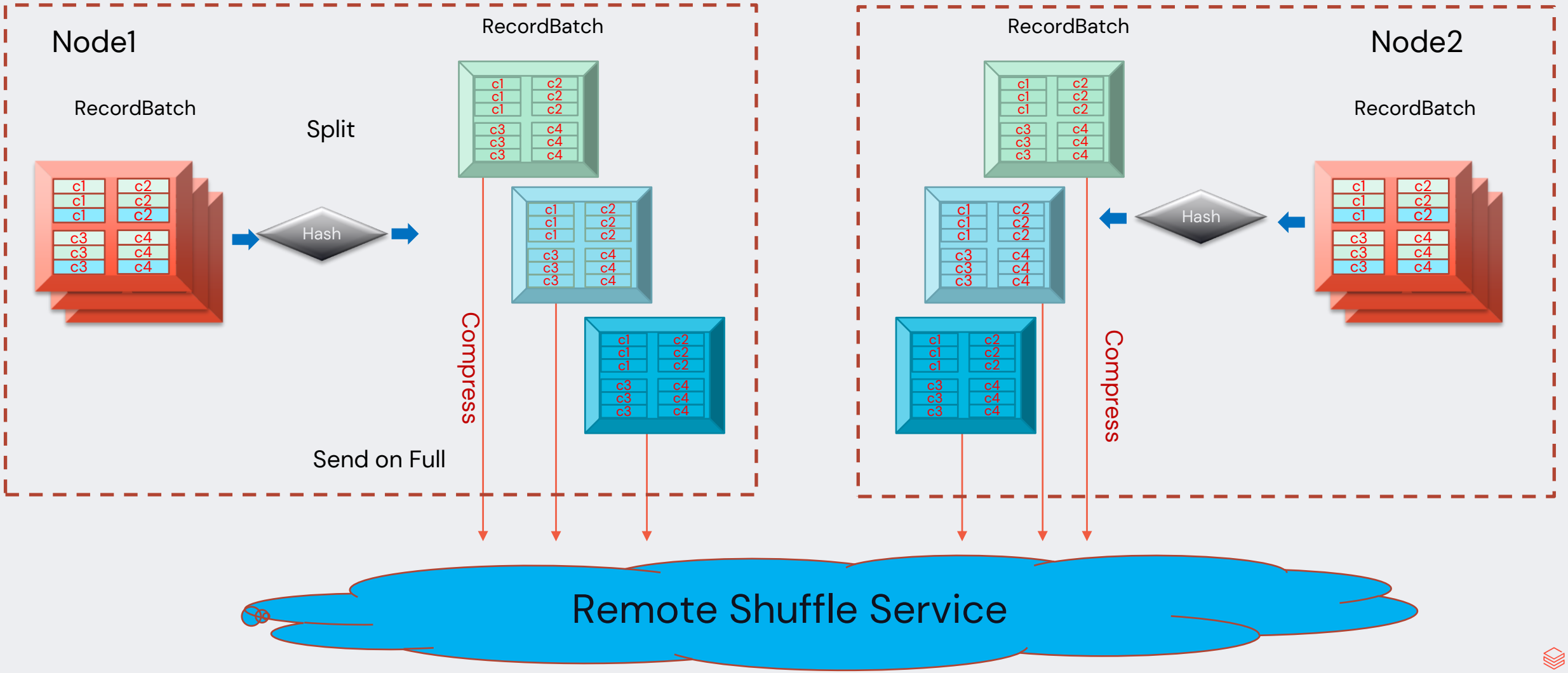
Hash Based Columnar Shuffle (2)



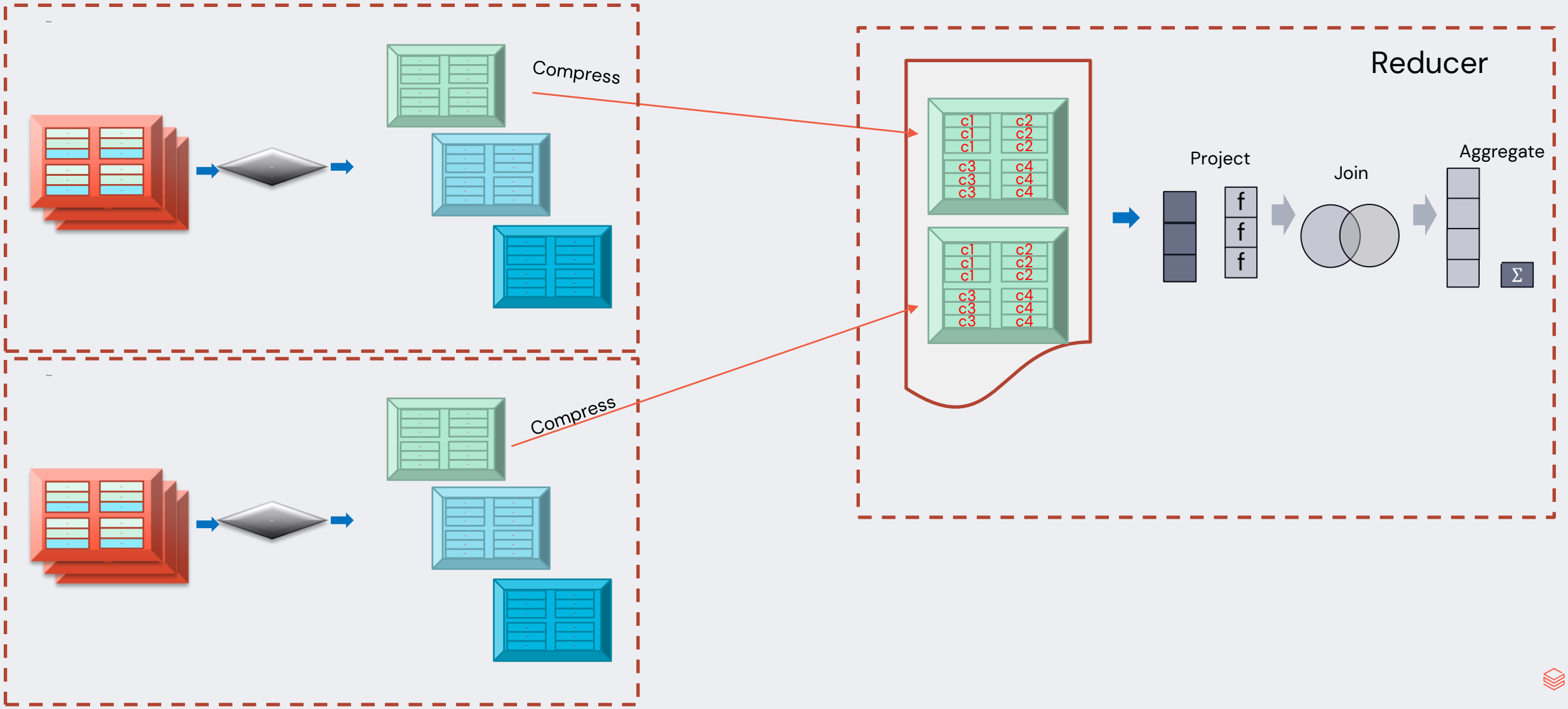
Hash Based Columnar Shuffle (3)



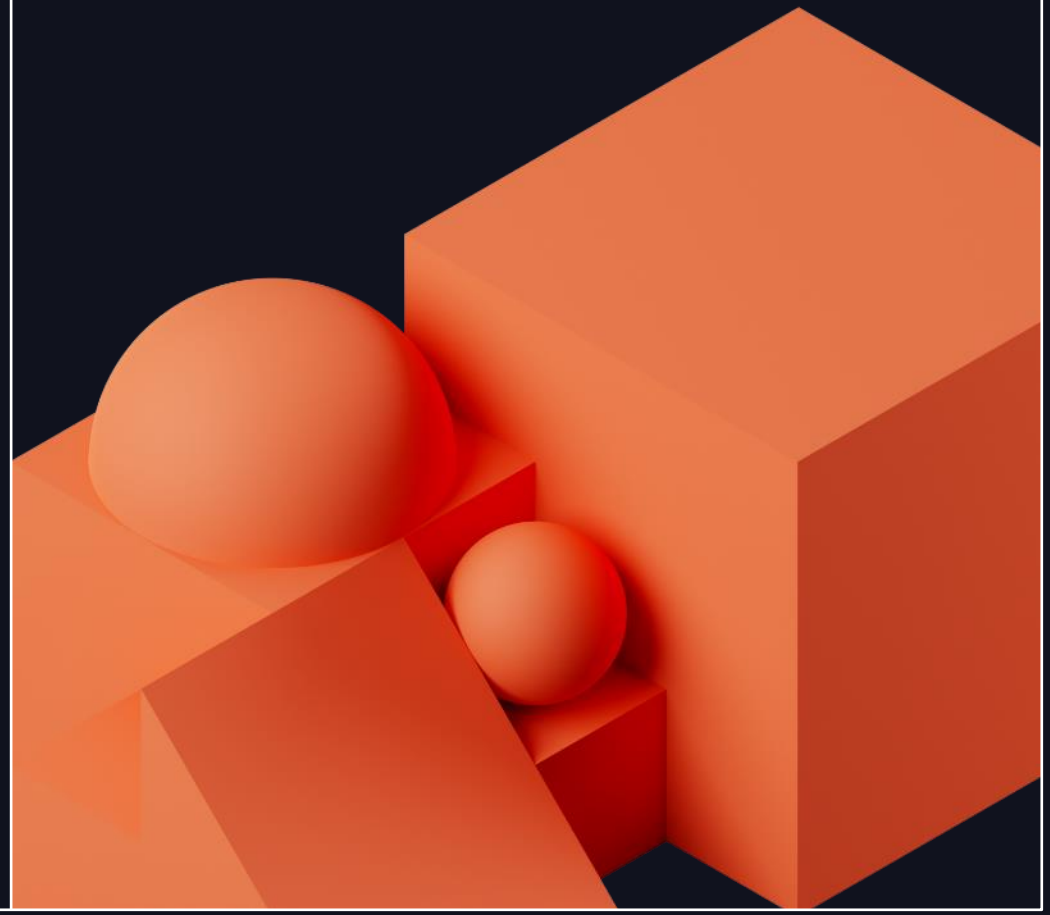
Remote Shuffle Service Support



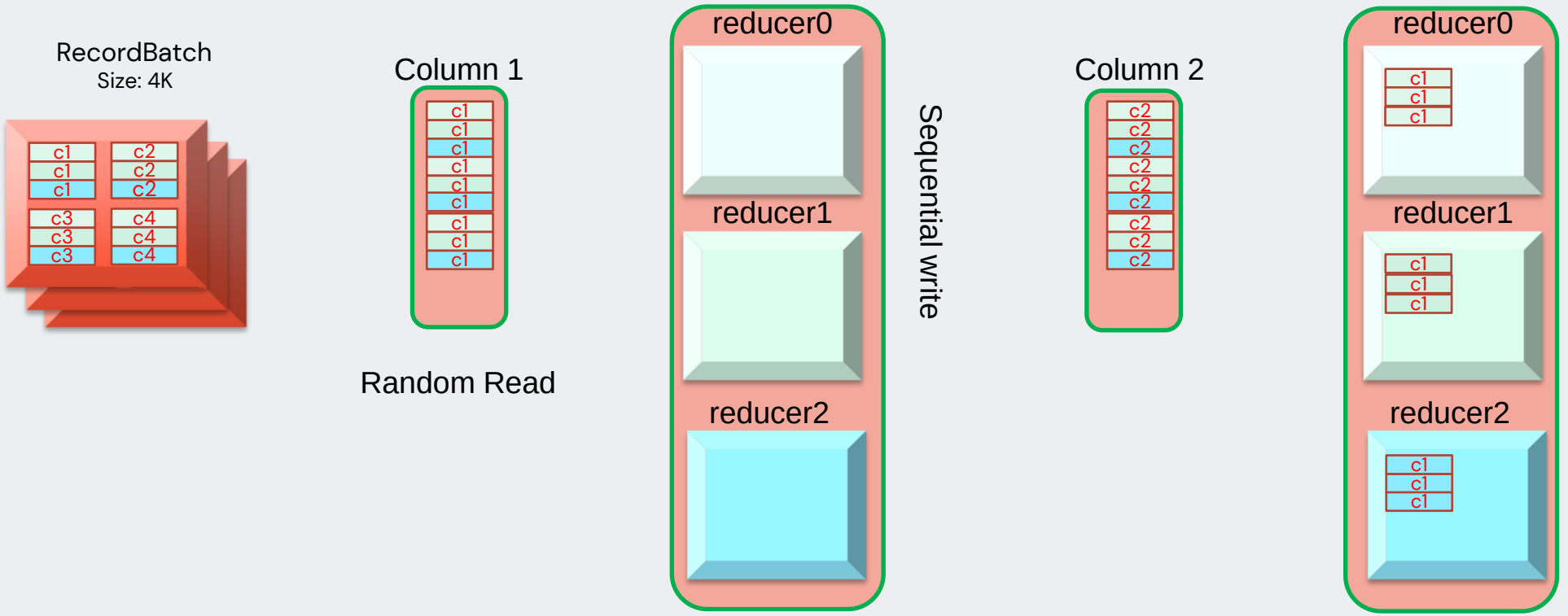
Pushed Shuffle



Split Function



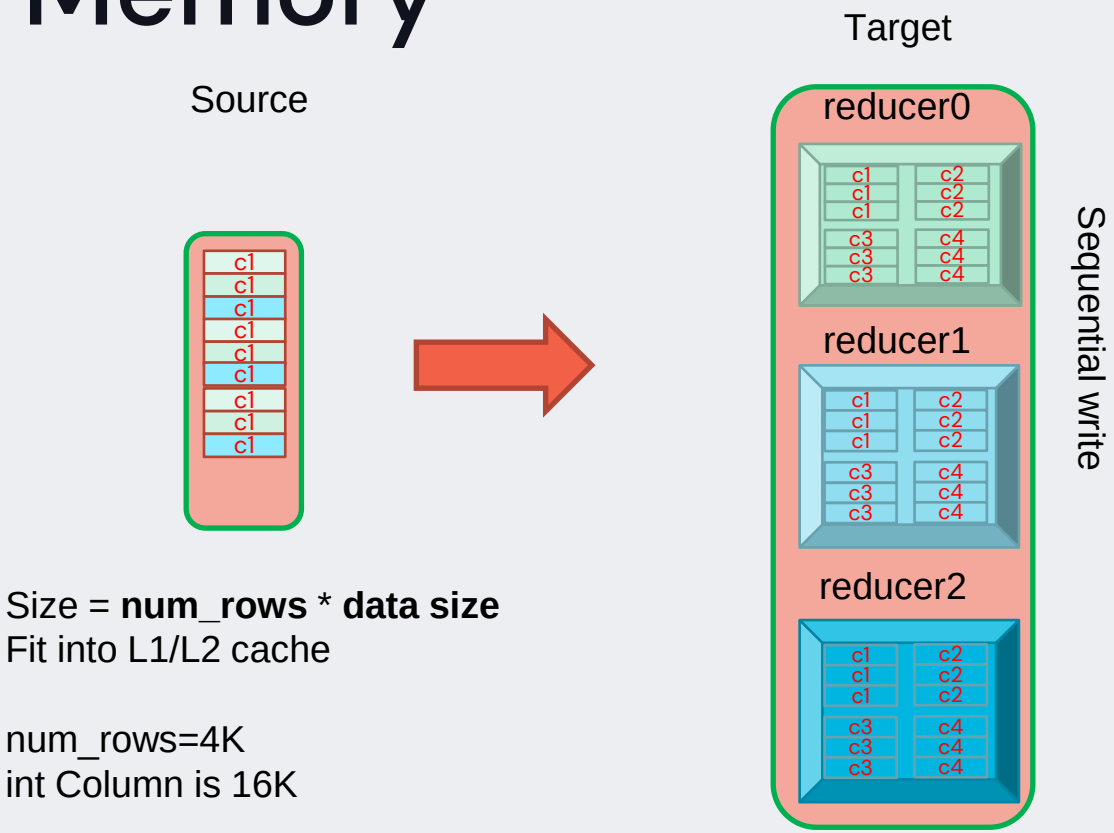
Split Function




```
for c in columns:
  for d in reducers:
    for r in row_offsets in d:
      d[n++]=c[c_offset[r]]
```

- Each column is scaned reducers# times
- **Key is to keep column data in L1/L2 cache**
- It’s a random read/sequential write patten

Memory



Size = **num_rows** * **data size**
Fit into L1/L2 cache

num_rows=4K
int Column is 16K

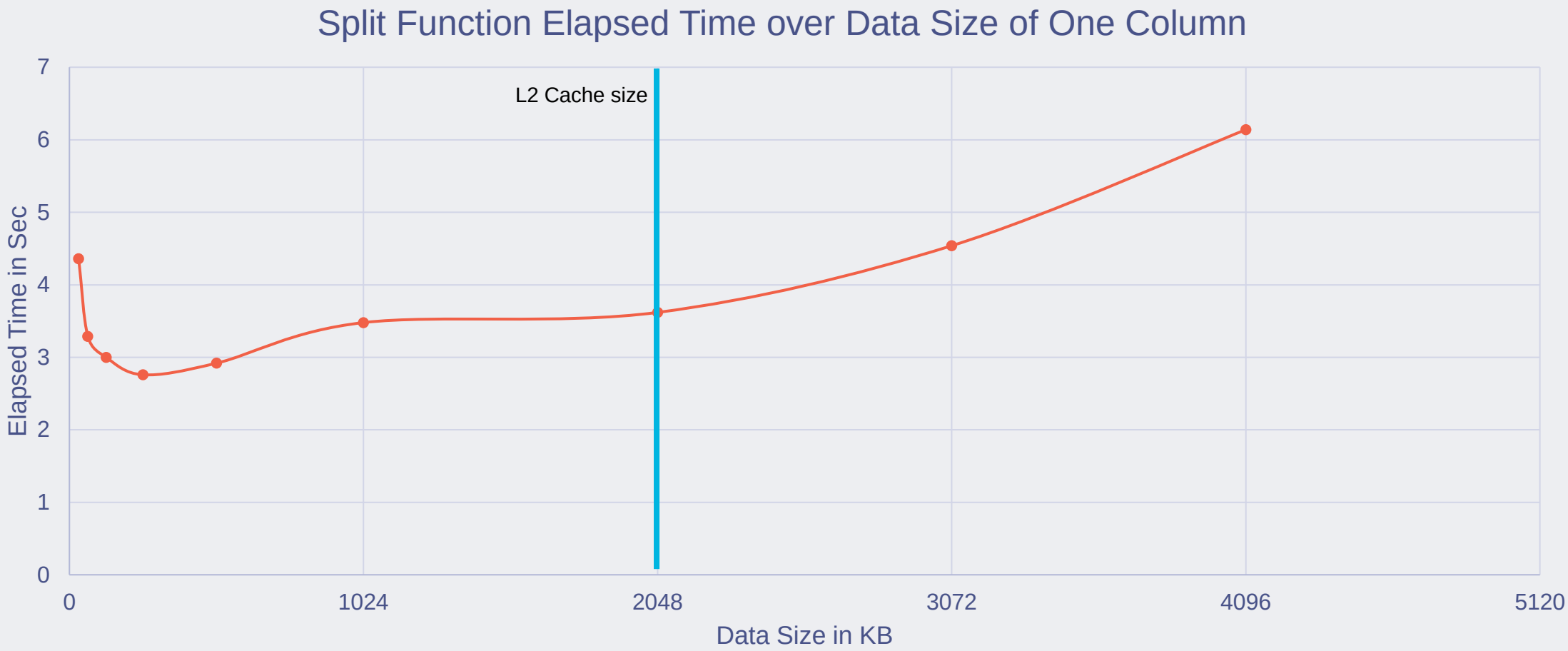
Pre-allocated size =
data size * num_rows * num_columns * num_reducers * num_executor_cores * num_executors

4k row, 8 int column, 1k reducer, 256 threads
system needs 32G pre-allocated memory

To solve the target memory allocation issue:

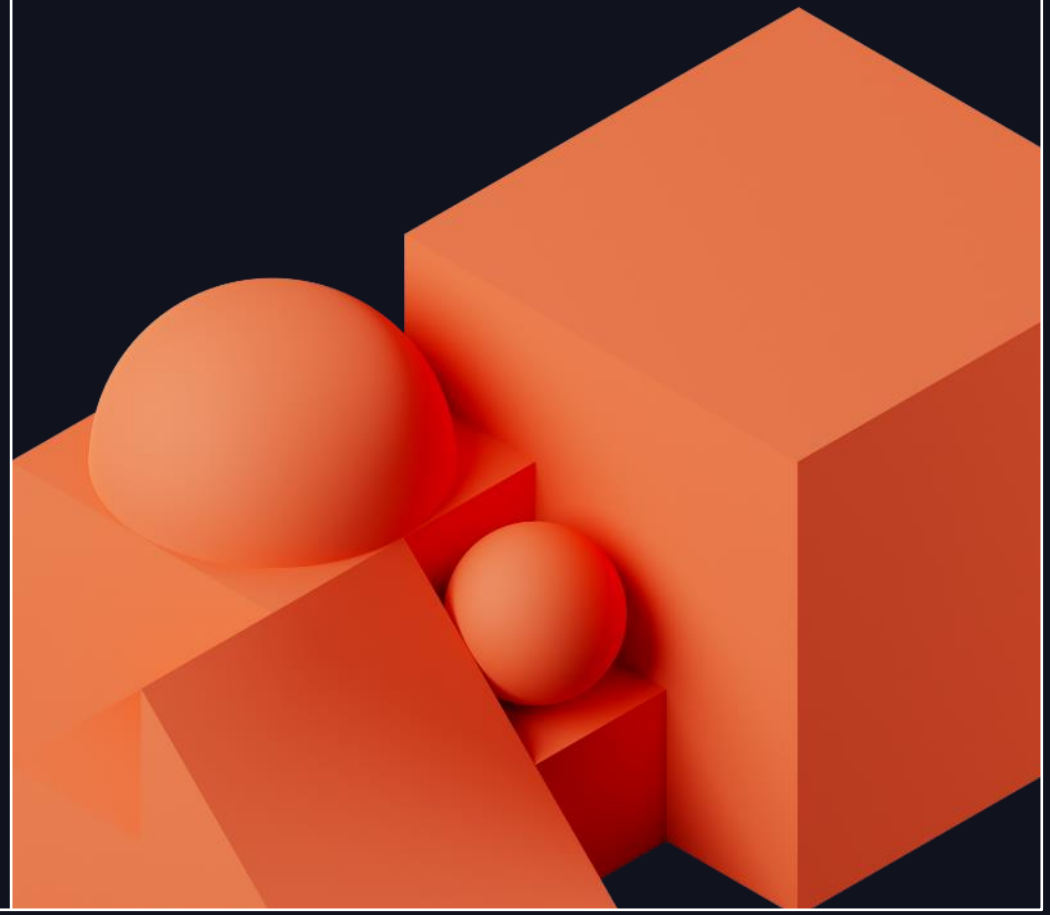
- Memory allocation for each reducer is delayed until the reducer is touched
- Target row size is calculated based on available memory when first record batch arrives
  - The less row size, the higher overhead of batch switch
- Experimental sort based shuffle is implementing by community
  - Sort is not a Columnar format friendly operation!

Split Performance

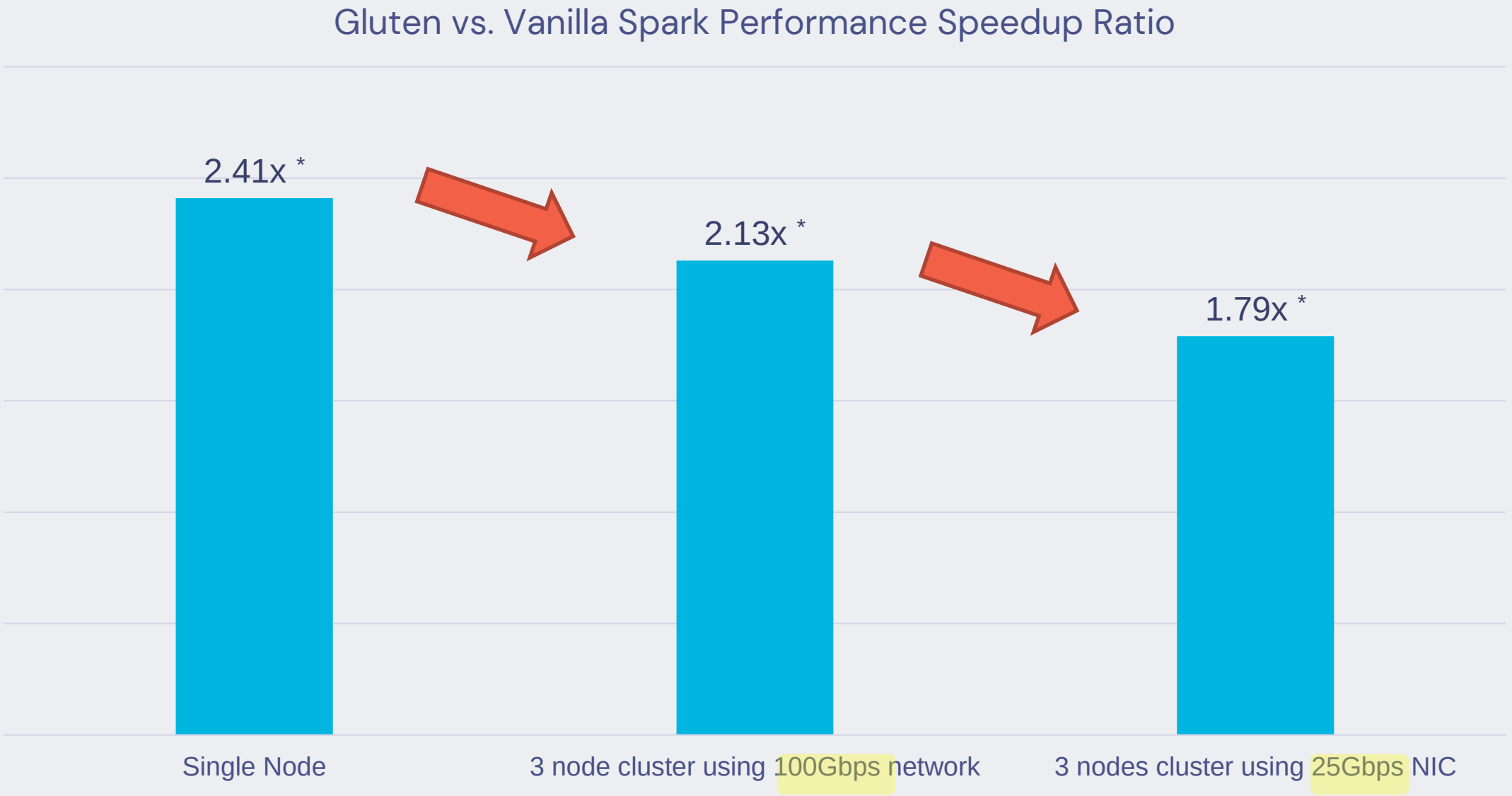


- Scalable to column number, reducer number but not column data size

Compress



Speedup on Cluster



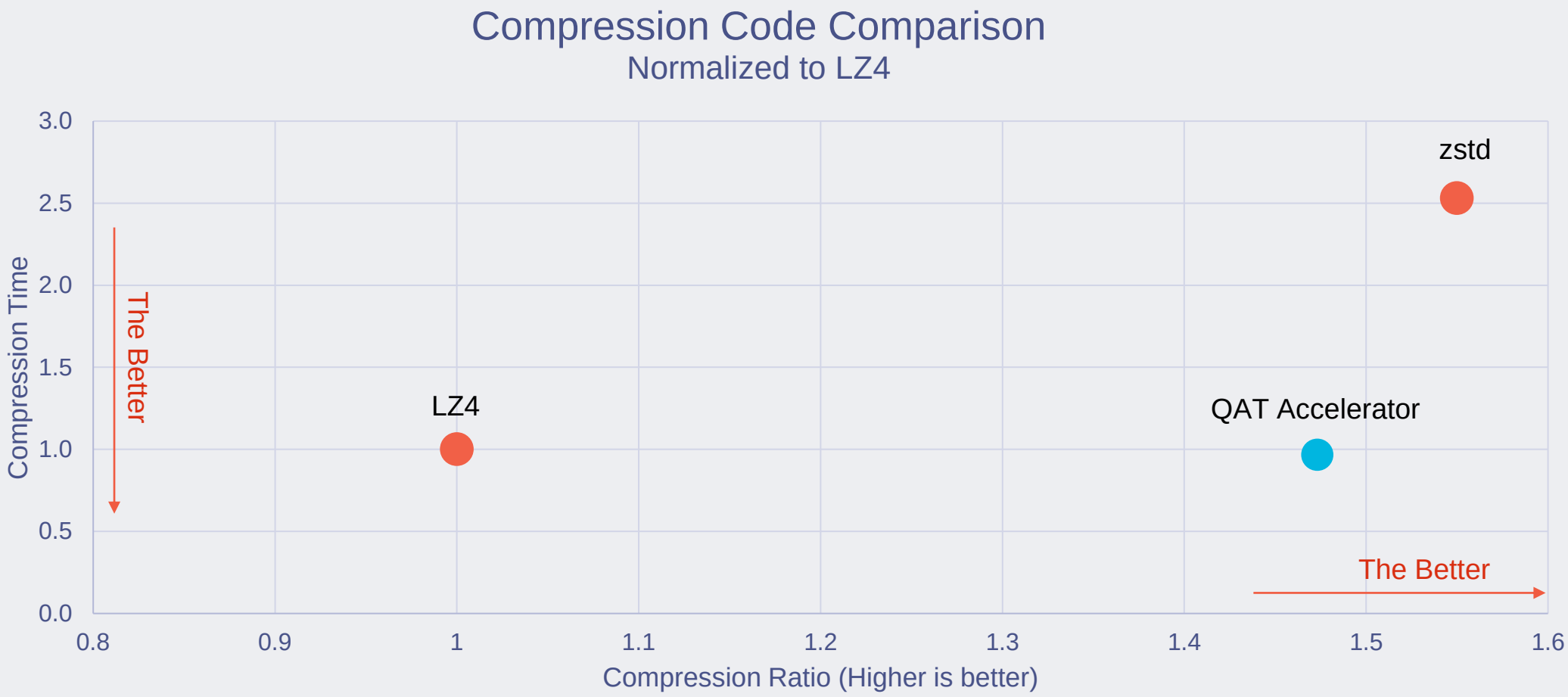
***Not the latest Performance**

The Network Bottleneck

SF6T Power Test 100Gbps Network	Spark	Gluten+Velox
Shuffle Data Size (GB)	9,176	8,085
CPU%	81%	71%
Network Throughput (MB/s)	4,540	7,871

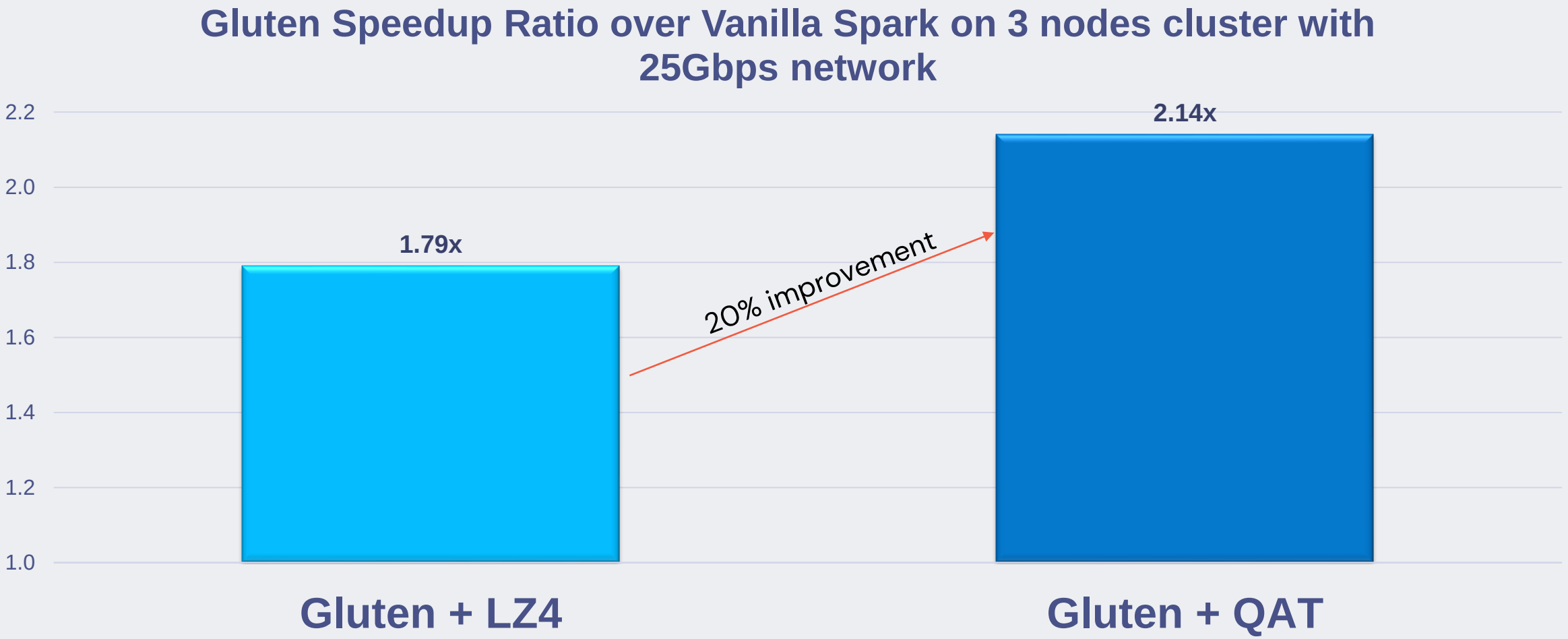
- Faster processing leads to higher pressure to I/O
- Solutions: faster network or less data

Accelerator in 4th Gen Intel® Xeon® Scalable processors



QAT vs. LZ4: Similar Compression Time, higher compression Ratio

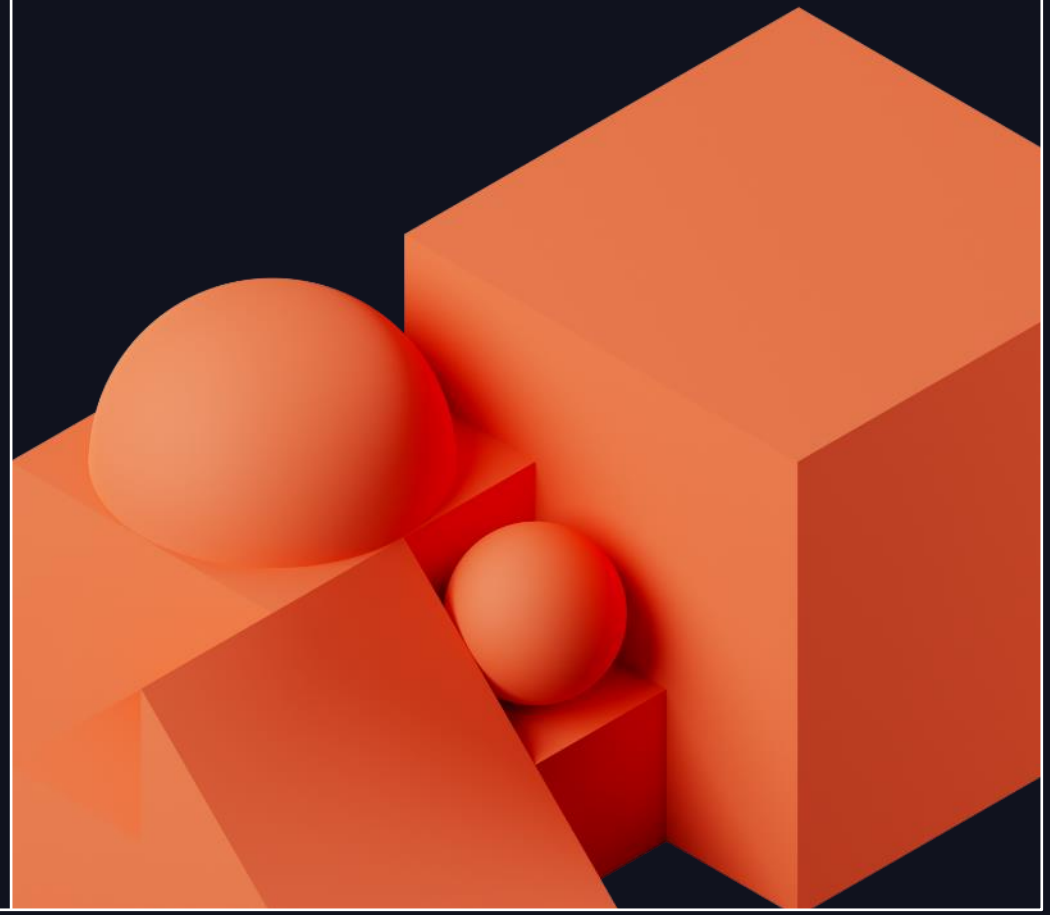
Performance Boost From QAT



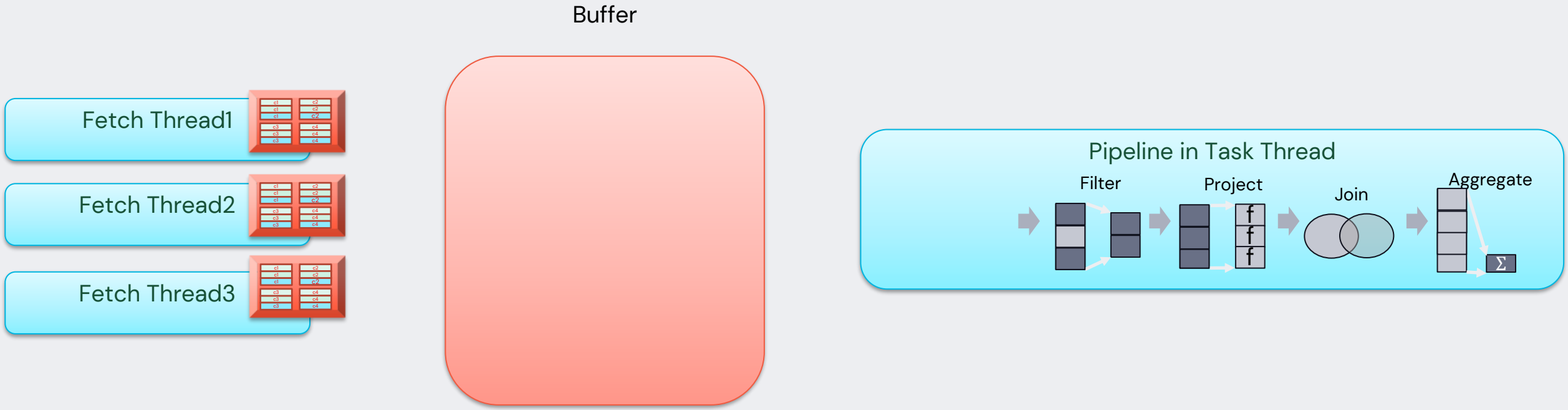
More optimizations on QAT is going on, like async mode

Reducer

AWS_Title_Slide



Fetch in Async



- Start the pipeline once one record batch is fetched instead of whole block (Vanilla Spark)
  - One block is the size a reducer fetches from a mapper

Recap

- With SQL engine implementation, Columnar Shuffle is a key operator to Performance
- Columnar Shuffle brings unique challenges
- Accelerator can be effectively used in shuffle

Q&A