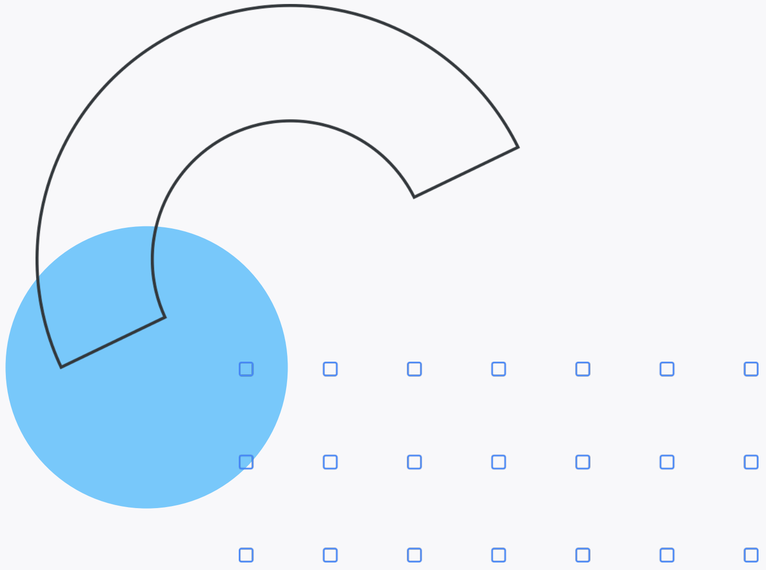
Databend

Databend Sort 优化总结

主讲人：RinChaNOW

2022.11

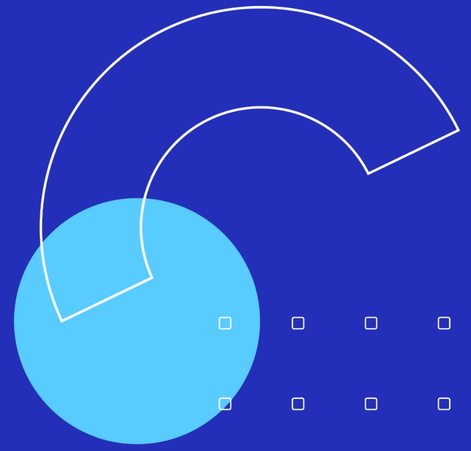


Data infra 研究社

自我介绍

Databend

- RinChanNOW
  - Github ID: RinChanNOWWW



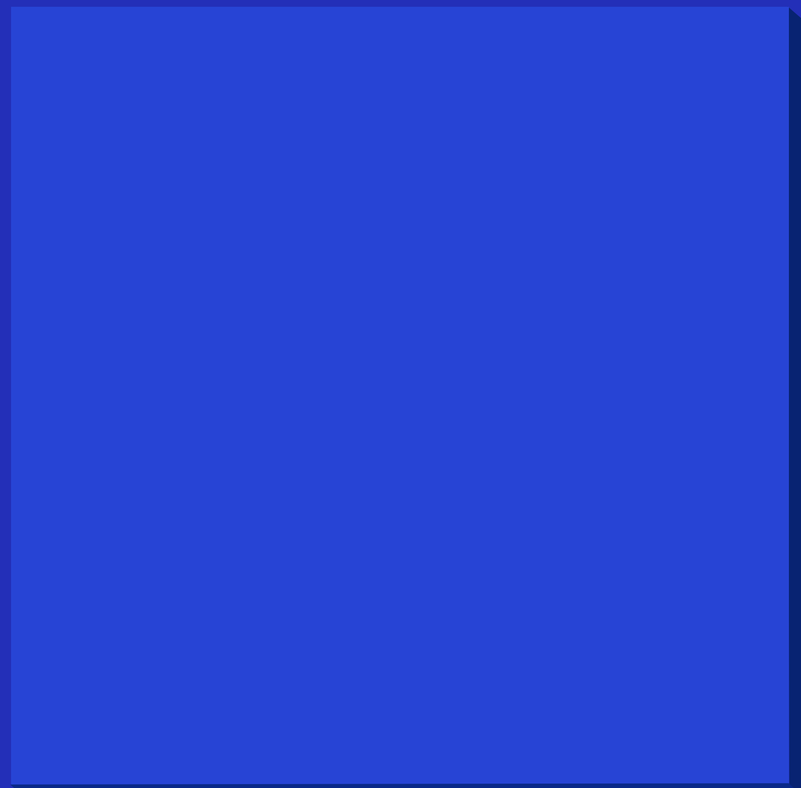
Databend

目录

CONTENTS

Databend

引言



Data infra 研究社

排序的用途

Databend

排序操作在数据库中十分常见与重要。最常见的用法是将数据表按照某几个字段进行排序，或者找出数据表中最大或最小的几行（TopN）。

```
SELECT * FROM table ORDER BY co11, col2; -- normal sort
SELECT col FROM table ORDER BY col LIMIT n; -- top n
```

Databend 的 RECLUSTER TABLE 命令也与排序息息相关，此命令会将数据表按照 CLUSTER KEY 重新组织，并最终使得底层数据分布遵循 CLUSTER KEY 有序排列。

```
ALTER TABLE [IF EXISTS] <name> RECLUSTER [FINAL] [WHERE condition]
```

Ref: https://databend.rs/doc/reference/sql/ddl/clusterkey/dml-recluster-table

Databend

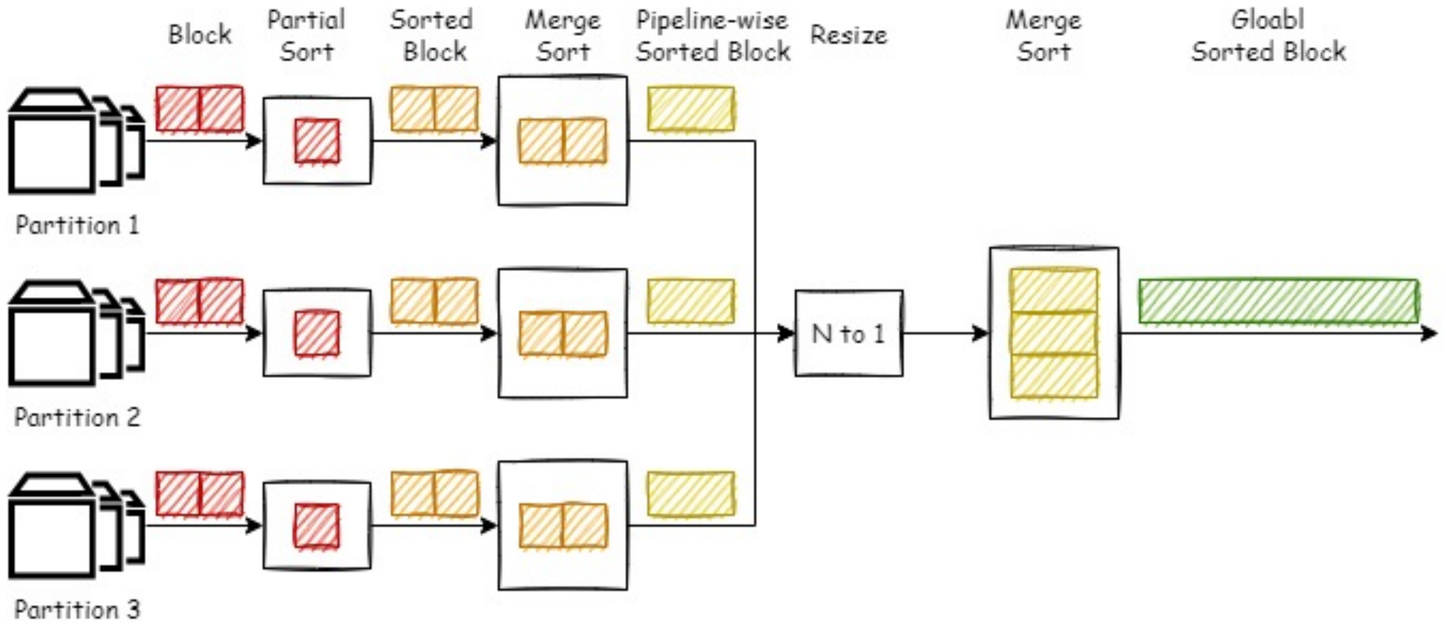
Databend Sort 历史实现

Data infra 研究社

历史排序流水线

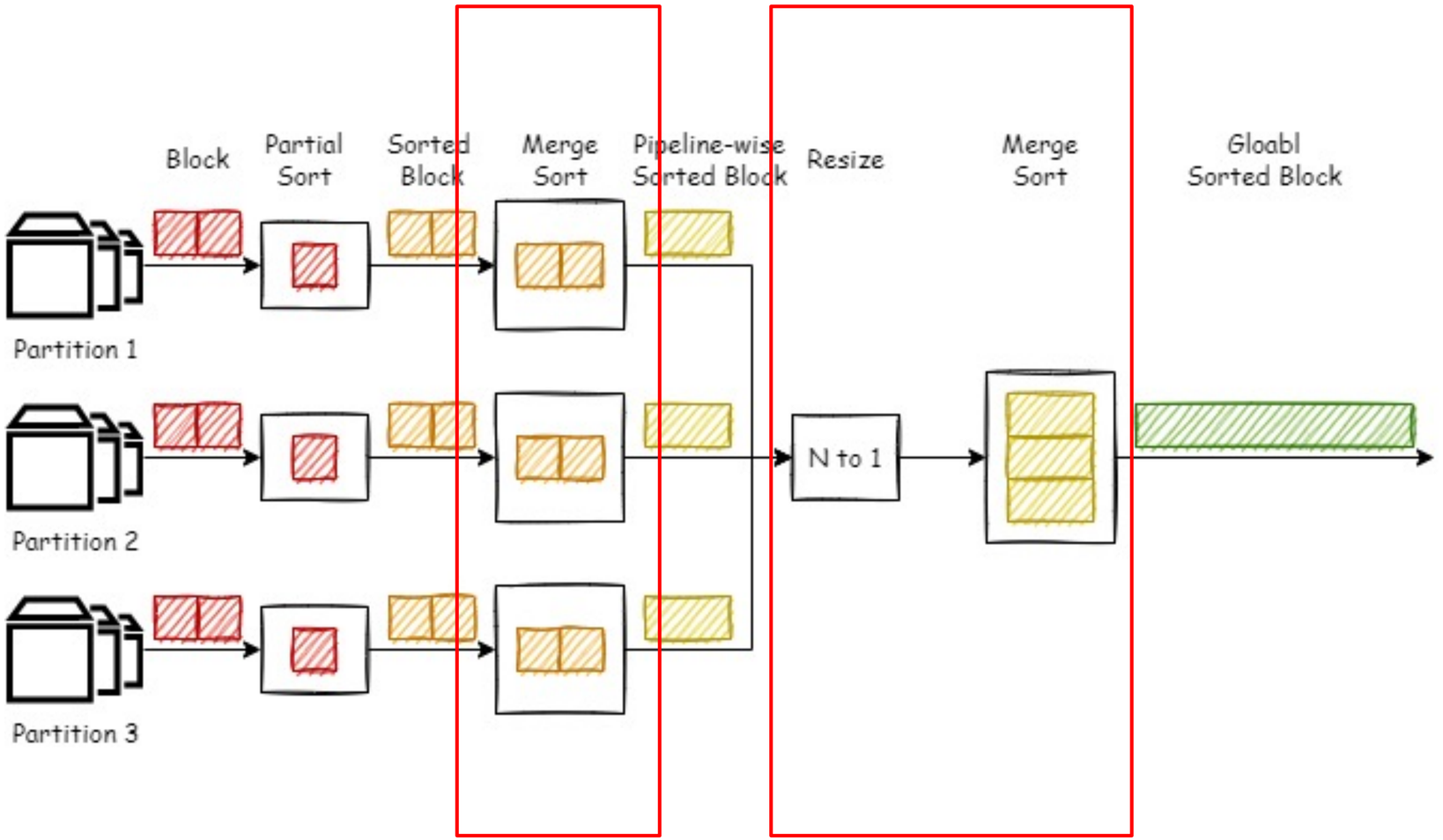
Databend

- Partial Sort: 对 Block 排序
- Merge Sort 1: 对一条流水线上的有序 Block 进行归并排序
- Merge Sort 2: 合并所有流水线，进行归并排序



历史排序流水线

Databend

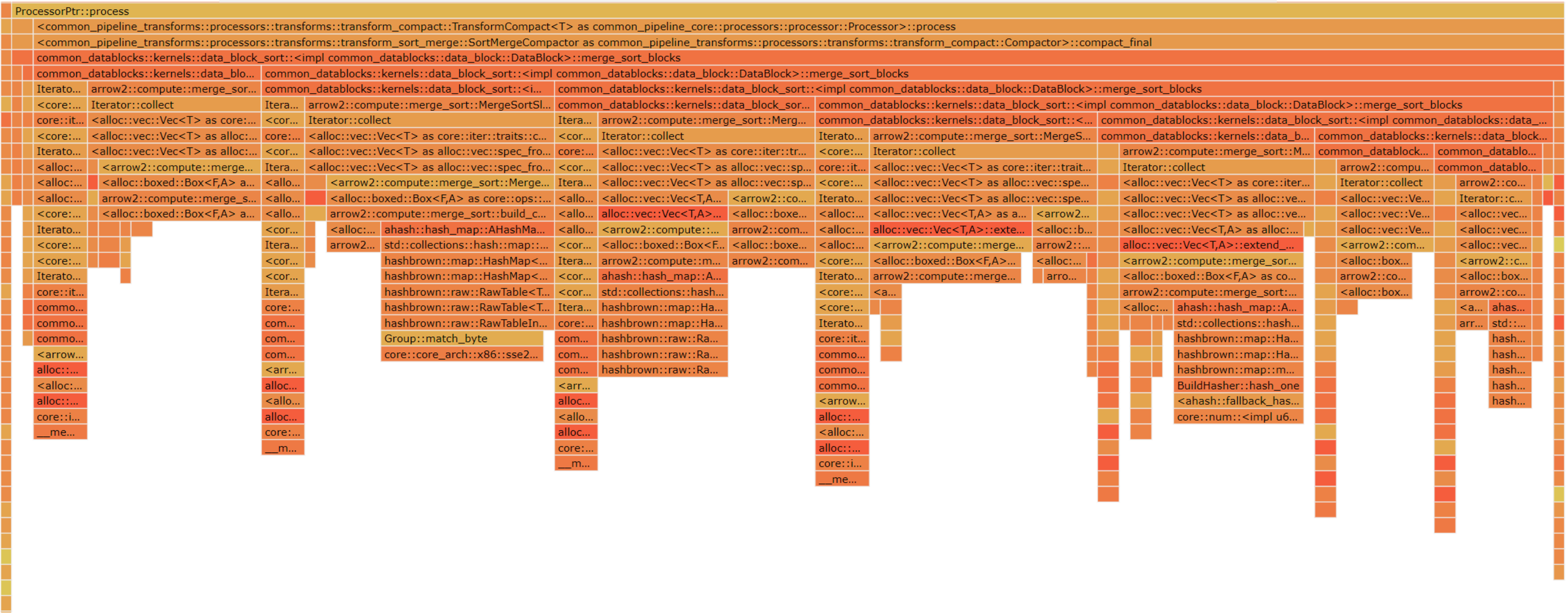


Blocking！！！

历史排序流水线

Databend

SELECT * FROM numbers(10000000) ORDER BY number



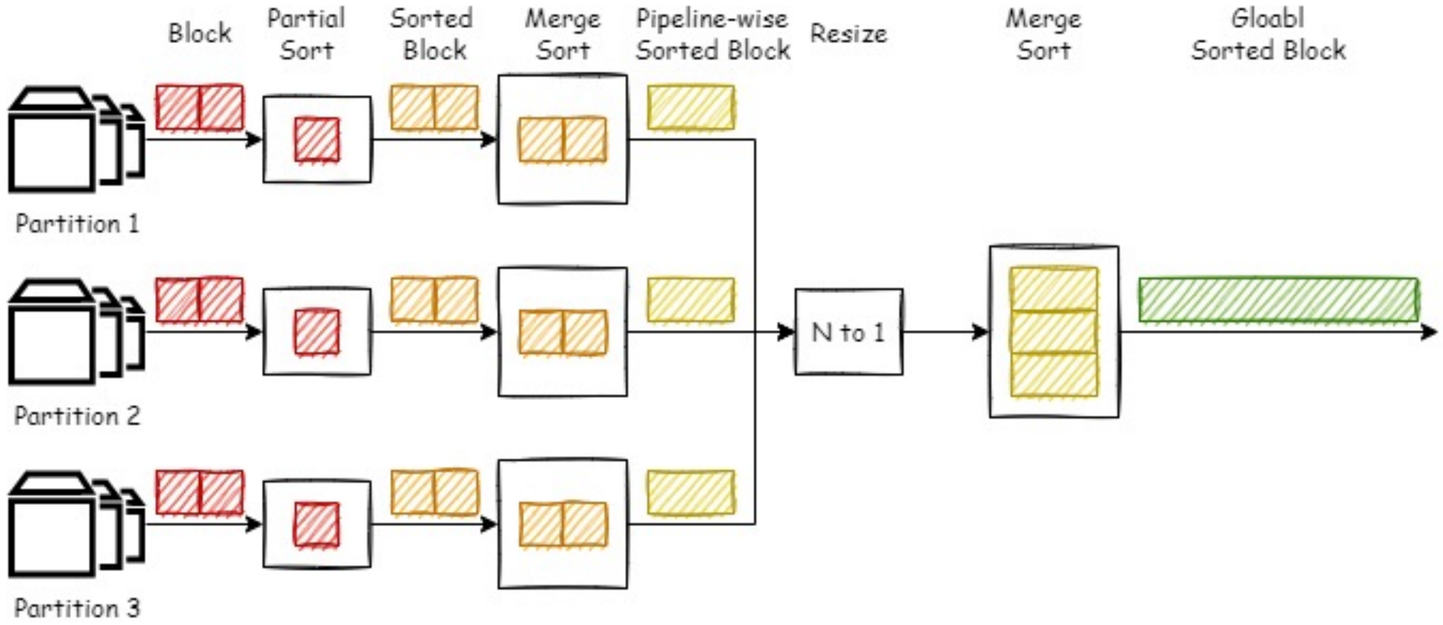
Data infra 研究社

历史排序流水线

Databend

```
SELECT col
FROM table
ORDER BY col LIMIT n;
```

历史优化策略：将 LIMIT 下推到每一个排序步骤。



Databend

Merge Sort 流式改造

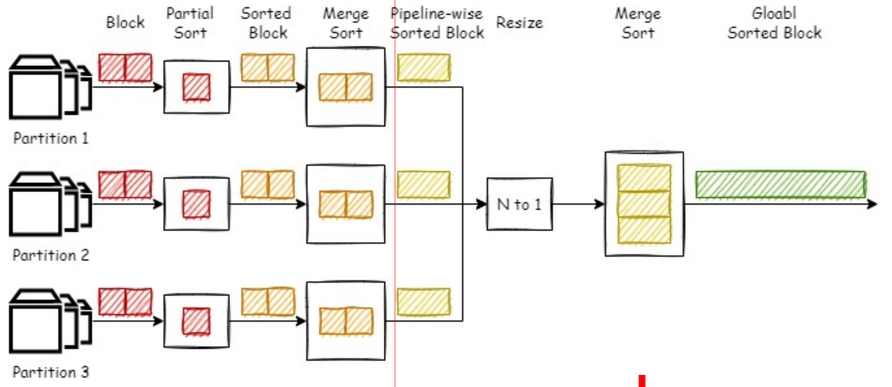
Data infra 研究社

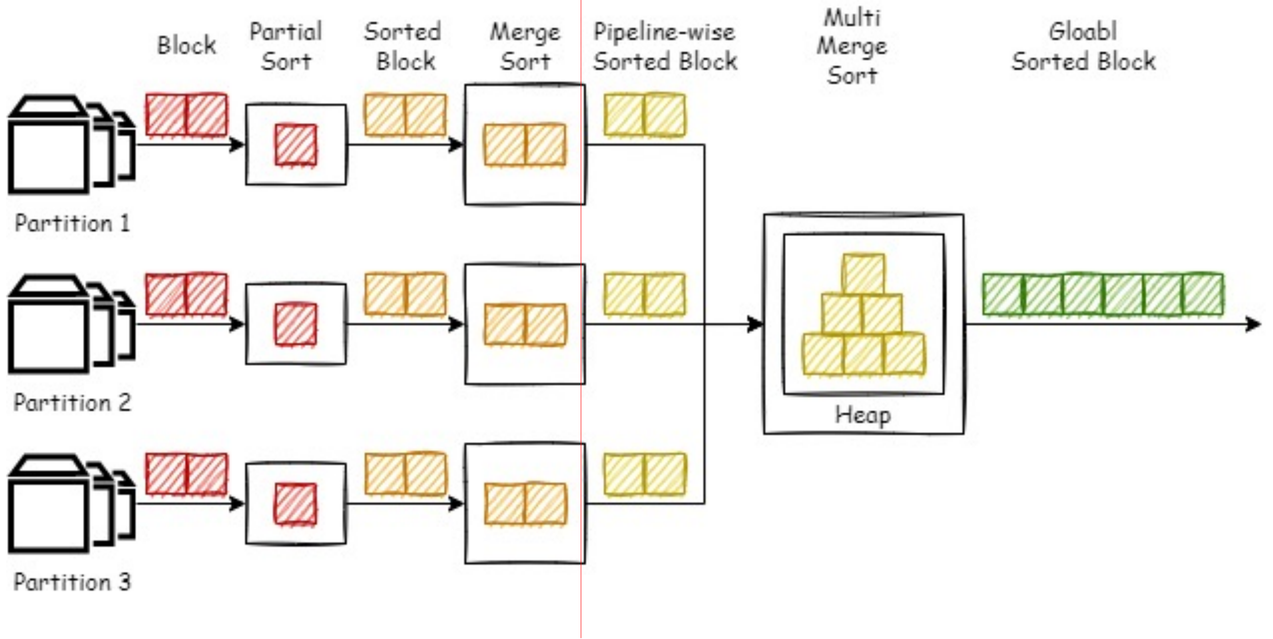
新版流水线

Databend

与旧版的主要区别：

- Block 不会合并，依旧按照分块传递。
- 将 Resize 管道和第二个 Merge Sort 合并为一个 Multi Merge Sort。
- 使用堆来进行多路归并





实现细节：Block Cursor

Databend

```
struct Cursor {
        pub input_index: usize,
        // pub block_index: usize,
        pub row_index: usize,
        // other fields
}
```

- input_index: Cursor 来源于哪个上游 input port。
- block_index: Cursor 来源于哪个 block。可以省略，因为 heap 中每一个 block 只可能来自于不同的 input（归并排序特性）。
- row_index: Cursor 指向 block 中的哪一行

实现细节：Heap

Databend

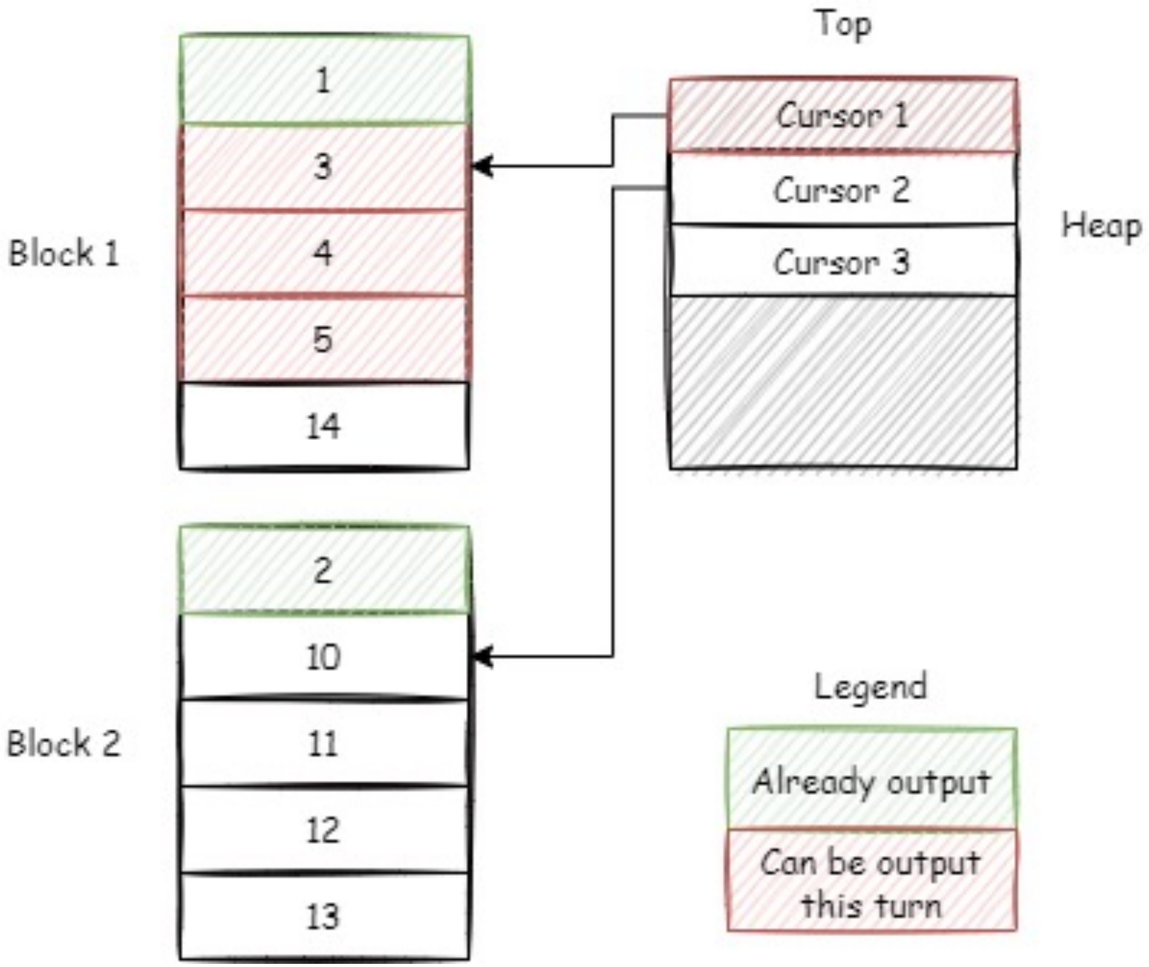
1. 若堆中元素不少于输入数，执行下一步；否则结束此流程。
2. 弹出堆顶 Cursor，将 Cursor 所指向的行推入待输出队列。
3. 将 Cursor 指向下一行，若已遍历 block 中所有行，则标记此 input 可以拉取下一个 block 数据；否则，将递增后的 Cursor 放回堆中。
4. 若待输出队列累计已达要求（limit 或 block_size），则退出此流程准备输出；否则，返回第 1 步。

https://github.com/datafuselabs/databend/blob/main/src/query/pipeline/transforms/src/processors/transforms/transform_multi_sort_merge.rs
drain_heap 方法

实现细节：Heap

Databend

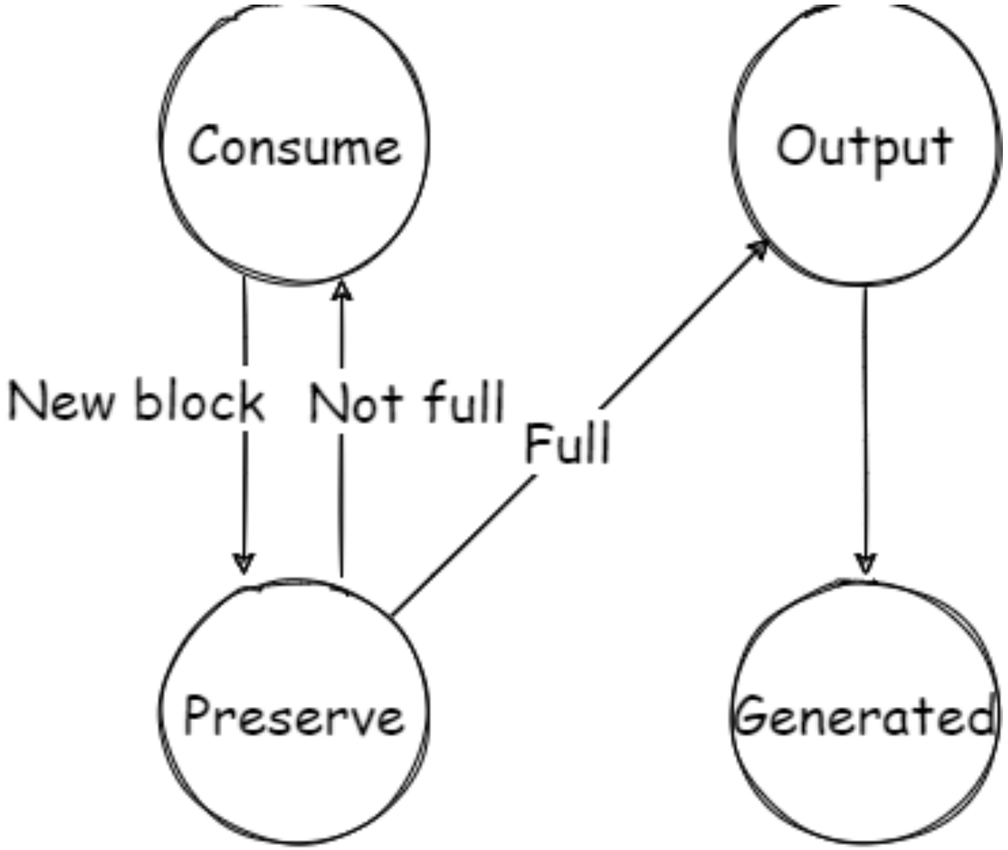
1. 若堆中元素不少于输入数，执行下一步；否则结束此流程。
2. 弹出堆顶 Cursor，将 Cursor 所指向的行推入待输出队列。
3. 将 Cursor 指向下一行，若已遍历 block 中所有行，则标记此 input 可以拉取下一个 block 数据；否则，将递增后的 Cursor 放回堆中。
4. 若待输出队列累计已达要求（limit 或 block_size），则退出此流程准备输出；否则，返回第 1 步。



实现细节：Processor 状态机

Databend

- Consume: 拉取数据 block，转为 Preserve 状态。
- Preserve: 核心逻辑状态。将 block 推入堆中并执行上一节所示的堆操作流程。若可以进行输出，则转为 Output 状态；否则回到 Consume 状态进行下一波数据拉取。
- Output: 构造输出 block，转为 Generated 状态。
- Generated: 将需要输出的 block 推入 output port。

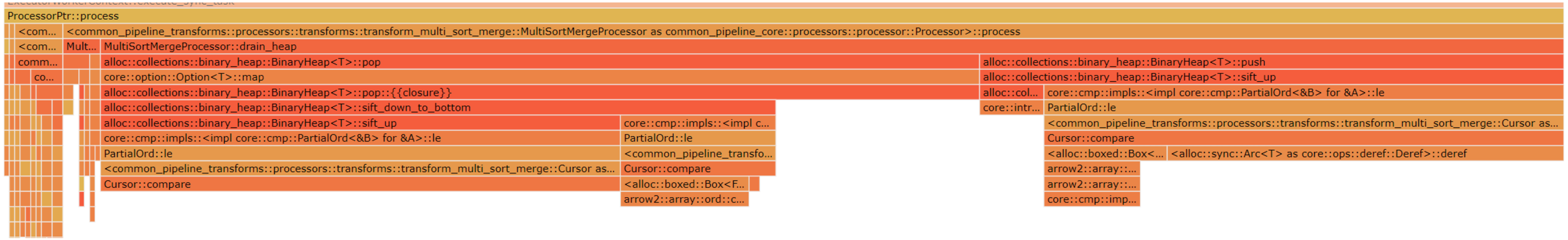


Morsel-Driven Parallelism

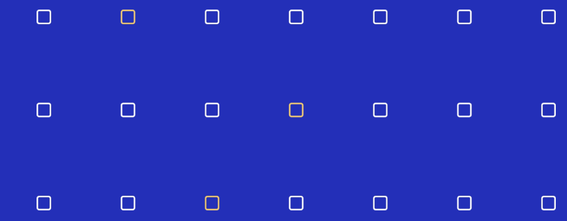
性能分析

Databend

```
SELECT * FROM numbers(10000000) ORDER BY number
```



上图是流化 MergeSort 之后（没有进行堆操作优化）的火焰图。可见，性能瓶颈从归并排序转换了堆操作，其中推操作的瓶颈又在于 Cursor 之间的比较运算。所以接下来便引出了本次的第二个优化点：Row Format。



Databend

Comparable Row Format

Data infra 研究社

Comparable Row Format

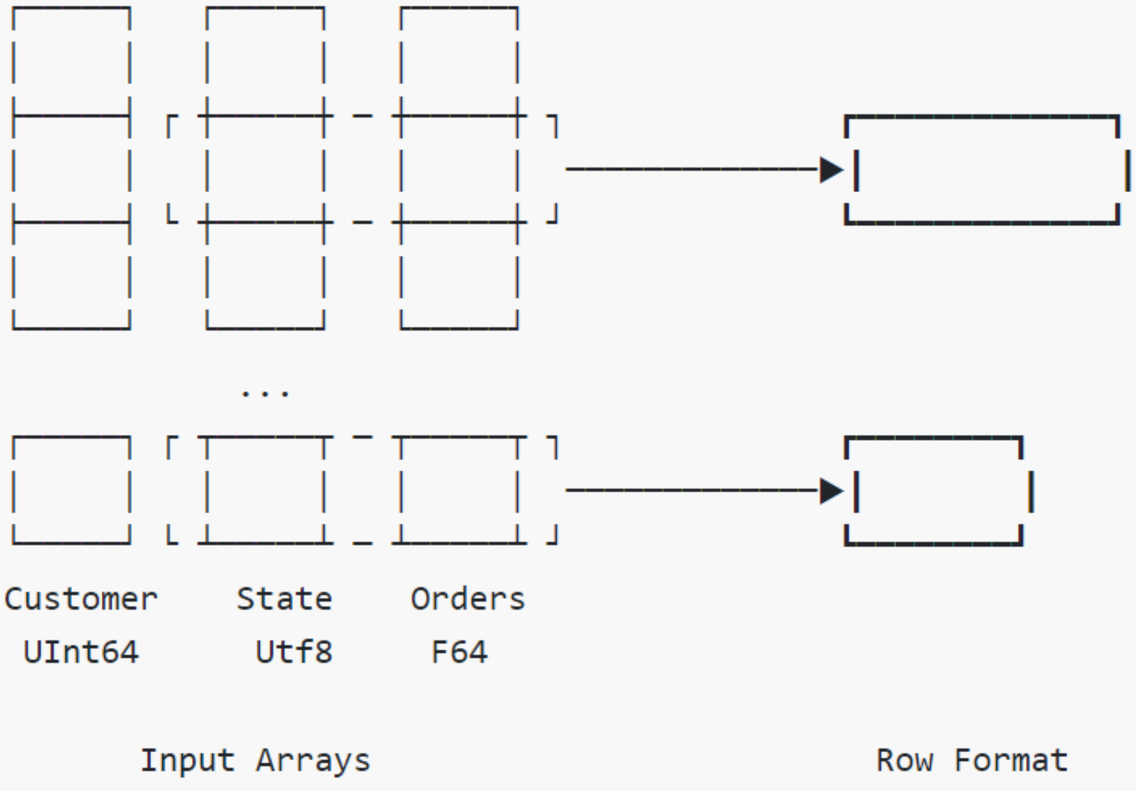
Databend

- arrow-rs

Docs: https://docs.rs/arrow/latest/arrow/row/index.html
Blog: https://arrow.apache.org/blog/2022/11/07/multi-column-sorts-in-arrow-rust-part-2/

- arrow2

Port: https://github.com/jorgecarleitao/arrow2/pull/1287



Comparable Row Format

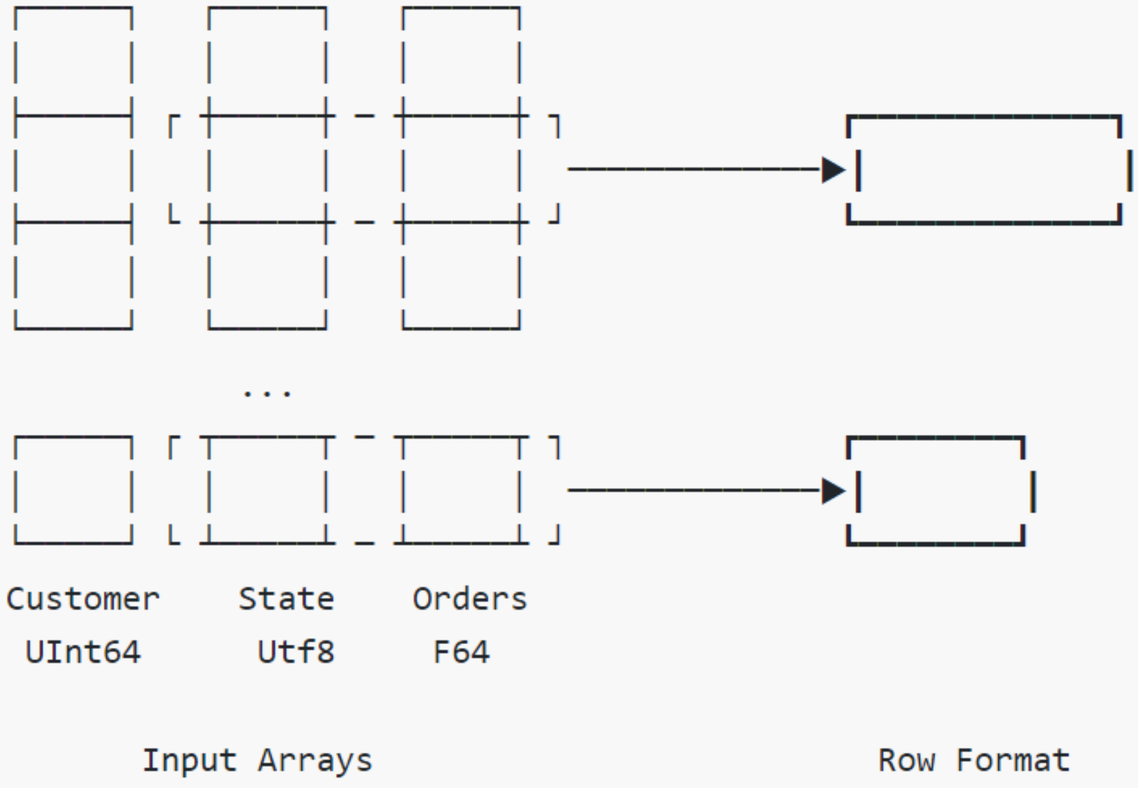
Databend

- 整个行数据编码由各个排序键编码依次连接而成。
- 所有类型的数据的第一个字节都是用来表示是否为 NULL。若为 NULL，只占一个字节。

  若不为 NULL，此字节为 0x01（一般情况，变长类型可能为 0x02）。

  若排序选项 null first 为 true，NULL 字节为 0x00。

  若排序选项 null first 为 false，NULL 字节为 0xFF。
- 如果排序选项为降序排序，则将整个数据按位反转。
- 行的比较即为底层二进制数据的 memcmp 比较，即：依次比较二进制位，碰到第一个不同二进制数是便可得出结果。



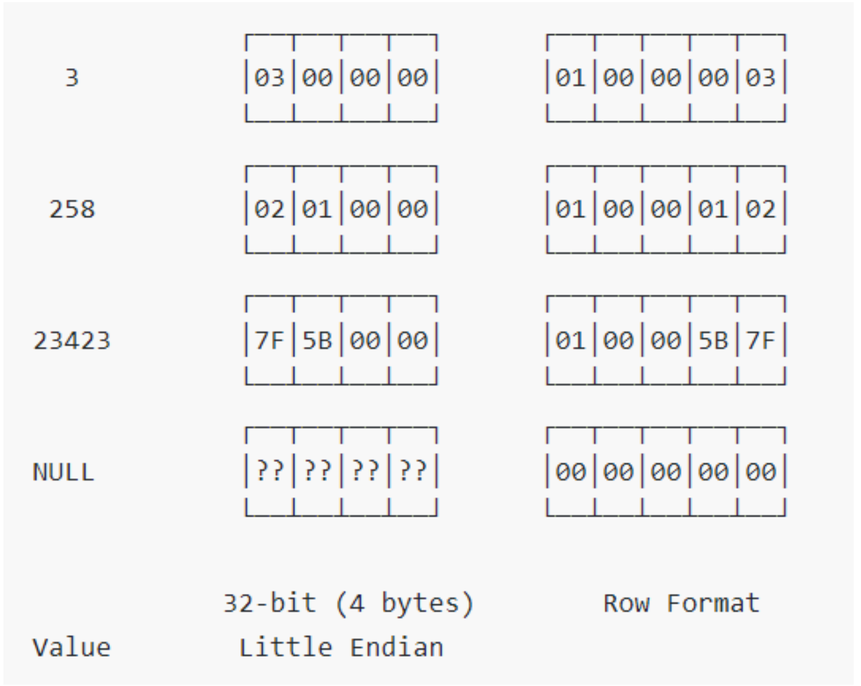
定长类型

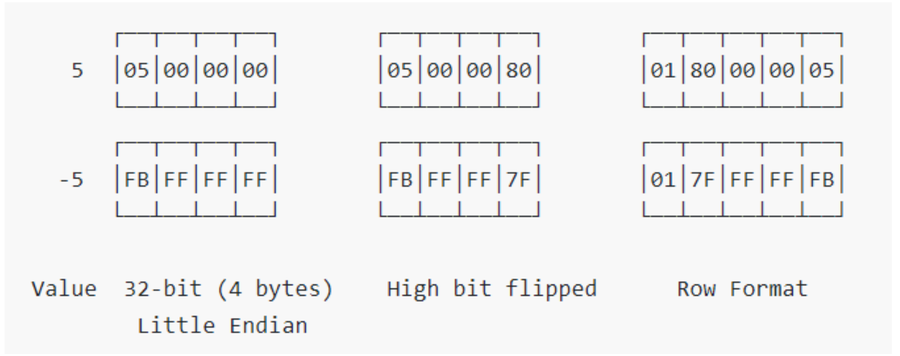
无符号整数：大端方式

有符号整数：补码+大端方式

浮点数：IEEE 754

布尔类型：单字节



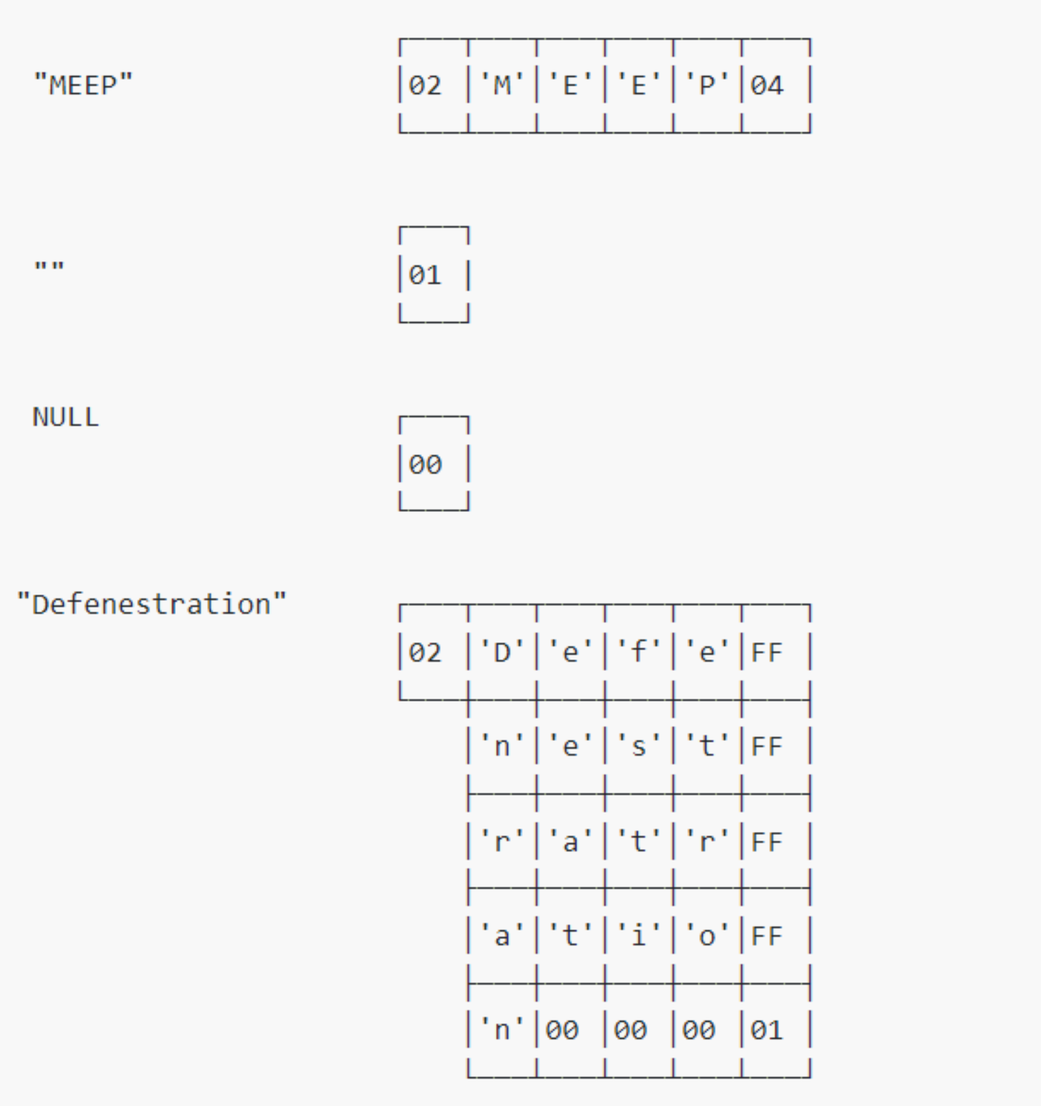


变长类型

基本思想：COBS 编码。
Ref:https://en.wikipedia.org/wiki/Consistent_Overhead_Byte_Stuffing

- NULL 用 0x00 表示
- 空串用 0x01 表示
- 非空串用 0x02 打头（使得非空串一定比空串大）
- 变长类型的数据长度被设计为 32 的整数倍

解决多排序键时无法界定各 field 的问题



字典类型

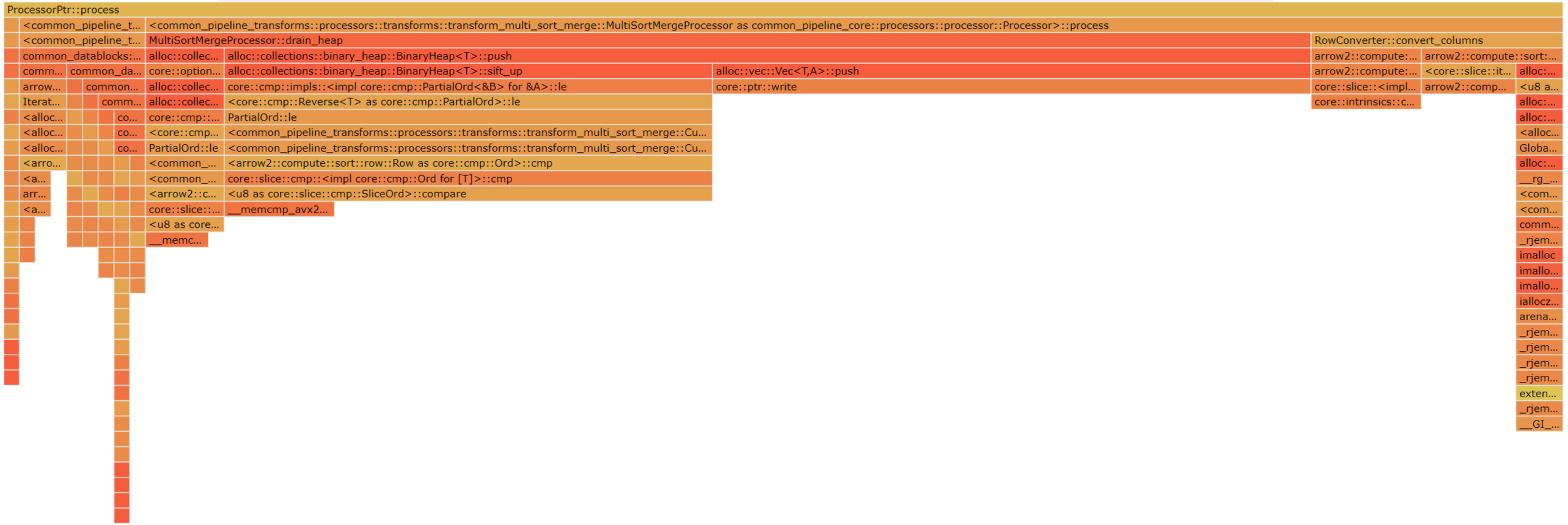
Databend

字典类型是 Arrow 中一种用来保存低基数数据的类型，在 Databend 中没有使用，这里不做过多介绍。

性能分析

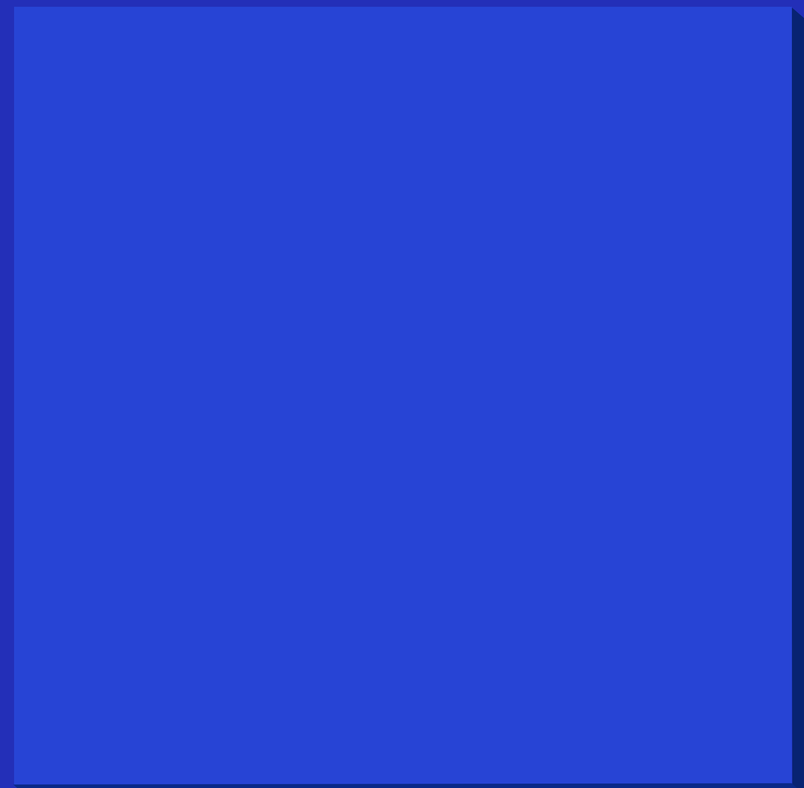
Databend

SELECT * FROM numbers(10000000) ORDER BY number



Databend

总结



Data infra 研究社

Benchmark

Databend

测试环境

query engine:
•OS: Ubuntu 18.04 LTS (Bionic Beaver)
•CPU: Intel Xeon Gold 5218R @ 80x 2.123GHz
•Memory: 90G
•Disk: 1.1T / 1,9T

minio storge:
•OS: Ubuntu 22.04 jammy
•CPU: Intel Xeon E-2124 @ 4x 4.3GHz
•Memory: 8G
•Disk: 147G / 1.9T

通过 LAN 连接

数据集

Mini hits 数据集（1w行数据）

Benchmark

Databend

new	old	old / new	SQL
0.49185555	1.58862891	3.22986883	select * from numbers(10000000) order by number;
0.49170787	1.64183498	3.33904556	select * from numbers(10000000) order by number desc;
0.09856892	0.26789701	2.717864921	select userid, flashmajor from hits order by flashmajor, userid desc;
0.06503526	0.1700541	2.614798495	select resolutiondepth from hits order by resolutiondepth;
0.27688674	0.35350362	1.276708375	select title from hits order by title;
0.29328114	0.34813435	1.187032859	select title from hits order by title desc;
0.30203853	0.46025009	1.523812508	select userid, title from hits order by userid, title;
0.29694118	0.47894412	1.6129259	select userid, title from hits order by userid desc, title;
0.29833156	0.46893471	1.571857533	select userid, title from hits order by userid, title desc;
0.29689456	0.39771965	1.339598981	select userid, title from hits order by userid desc, title desc;

Benchmark

Databend

new	old	old / new	SQL
0.02551741	0.02539468	0.995190343	select * from numbers(10000000) order by number limit 100;
0.02533295	0.02419826	0.955208928	select * from numbers(10000000) order by number desc limit 100;
0.06810979	0.27476401	4.034133859	select userid, flashmajor from hits order by flashmajor, userid desc limit 100;
0.06037709	0.17488738	2.896585112	select resolutiondepth from hits order by resolutiondepth limit 100;
0.12315882	0.1646738	1.337084912	select title from hits order by title limit 100;
0.12697252	0.17209053	1.355336808	select title from hits order by title desc limit 100;
0.12341952	0.24537162	1.988110309	select userid, title from hits order by userid, title limit 100;
0.12241615	0.28931403	2.363364883	select userid, title from hits order by userid desc, title limit 100;
0.12456732	0.2316427	1.859578419	select userid, title from hits order by userid, title desc limit 100;
0.12659582	0.25043391	1.978216263	select userid, title from hits order by userid desc, title desc limit 100;

Benchmark

Databend

new	old	old / new	SQL
0.05681156	0.16286981	2.866842769	select avg(fetchtiming) from (select * from hits order by fetchtiming desc limit 100);
0.05609226	0.17959318	3.201746195	select avg(fetchtiming) from (select * from hits order by fetchtiming desc limit 1000);
0.05605486	0.17966659	3.205192021	select avg(fetchtiming) from (select * from hits order by fetchtiming desc limit 10000);
0.05836755	0.17961846	3.077368504	select avg(fetchtiming) from (select * from hits order by fetchtiming desc limit 50000);
0.06108108	0.18346159	3.003574757	select avg(fetchtiming) from (select * from hits order by fetchtiming desc limit 90000);
0.06054325	0.16922857	2.795168247	select avg(sendtiming) from (select * from hits order by sendtiming desc limit 50000);
0.06047222	0.18473938	3.054946222	select avg(dnstiming) from (select * from hits order by dnstiming desc limit 50000);
0.06025029	0.19324225	3.207324811	select avg(connecttiming) from (select * from hits order by connecttiming desc limit 50000);
0.06222921	0.17401594	2.796370708	select avg(responsestarttiming) from (select * from hits order by responsestarttiming desc limit 50000);
0.06191917	0.16554495	2.673565392	select avg(responseendtiming) from (select * from hits order by responseendtiming desc limit 50000);

其他优化点

Databend

1. 并行二路归并排序

基于论文 Merge Path - A Visually Intuitive Approach to Parallel Merging，其基本思想是将找到两个有序序列中的排序交叉点，将两个序列按照交叉点分组，使得每个组内的两个数据部分独自归并排序，最后将所有块的结果合并。序列的分组使得每个组的归并排序可以并行执行，不会互相影响，最后直接这便可得到全序结果。

Ref: https://duckdb.org/2021/08/27/external-sorting.html。

2. 基数排序

DuckDB 与 ClickHouse 等数据库都引入了基数排序。基数排序是一种非比较的基于分布的排序方式，时间复杂度为 O(nk)，其中 k 是排序键的宽度（比如 Int32 的宽度是 4 字节），n 是排序键个数。当 n 很大的时候，基数排序的性能优势就会显现出来。

3. 特化 Cursor

如果排序键只有一个，且是 non-nullable 的简单类型，便可以将 Rows 直接 downcast 为简单类型的一维数组，对于简单类型有着非常好的优化效果。类似的，如果存在多个排序键，但是他们可以合并成一个简单的排序键（比如两个 i32 可以合并为 i64）。这是一类大的优化方向。

后续计划

Databend

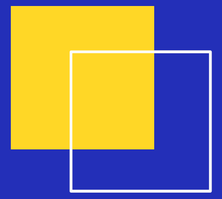
1. 聚合函数作为 order key
如果排序键为聚合函数的结果，以目前的实现来说，用于排序的始终就只有一条流水线，无法进行流式多路归并。需要针对这种情况优化流水线。ClickBench 中存在大量这种排序。

2. 特化 Cursor
等待 new expression 迁移完毕，进行 Cursor 的特化相关优化。

3. 统一 Row Format
Databend 在 group by 语句中，也维护了类似的行编码，作为 group by key。等 new expression 稳定后，可以在 Databend 侧维护一份独有的 Row Format，供各种场景使用（group by, join, sort……）。统一后，可以将排序键作为虚拟列插入 block 在流水线中传递，伴随排序的整个周期（目前只在最后的多条排序流水线归并时构建与使用）。

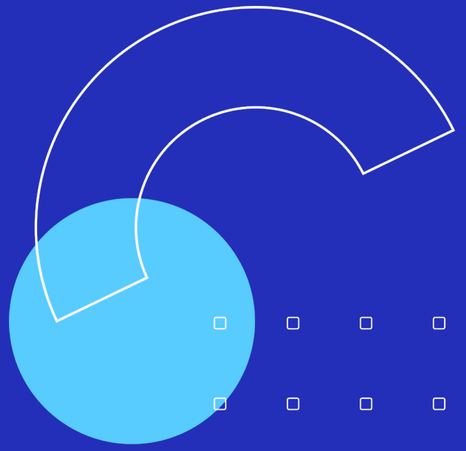
4. 引入外部排序
目前 Databend 的排序是纯内存的，需要引入外部排序来降低 OOM 风险。

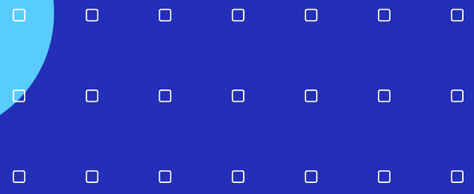
5. ……



Databend

Q & A





Data infra 研究社