НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

ИСПОЛЬЗОВАНИЕ ПЕРВИЧНОГО КЛЮЧА ПРИ ORDER BY MONOTONIC(PK), GROUP BY

Родигина Анастасия Олеговна

Москва, 2017

Почему это важно?

- Количество данных удваивается каждые два года
- Растет количество пользователей
- Компании вынуждены хранить все большее количество данных, к которым предоставляется доступ online. То есть все большее значение имеет скорость и удобство доступа к данным

Различия колоночных и строковых систем управления баз данных

- Способ хранения данных
- Работает эффективно для разных задач
- Возможность различной компрессии данных одного типа
- Возможность быстро менять структуру таблицы

Различные оптимизации, специфичные только для колоночного типа

- Late materialization, поздняя материализация, техника, когда столбцы считываются с диска и соединяются в строки как можно позже в запросном плане

- Итерирование по блокам, где множество значений из столбца передаются от одного оператора к другому, классическим и характерным для row-oriented СУБД же считается использование итераторов вулканического стиля, что аналогично pipeline в Unix. Для каждого оператора, который выполняется параллельно остальным, достаточно получить одну строку входа, чтобы произвести какую-то строку на выходе

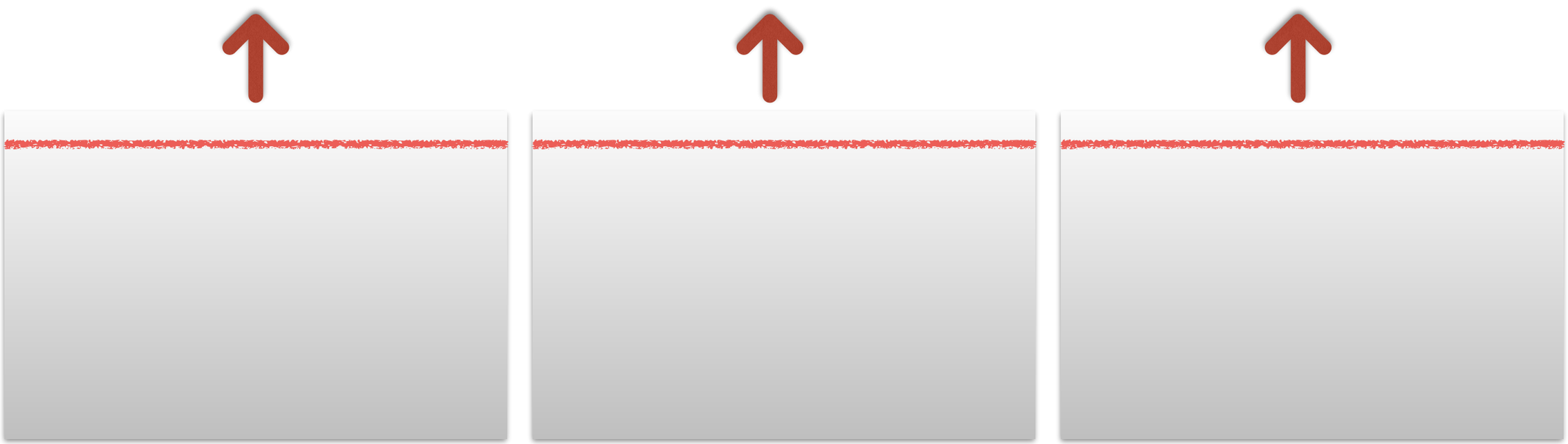
Различные оптимизации, специфичные только для колоночного типа

- Методы сжатия для конкретных столбцов, такие как run-length encoding (например, храним отсортированный набор в виде кортежей {значение, количество раз}), с непосредственной работой на сжатых данных при использовании планов поздней материализации

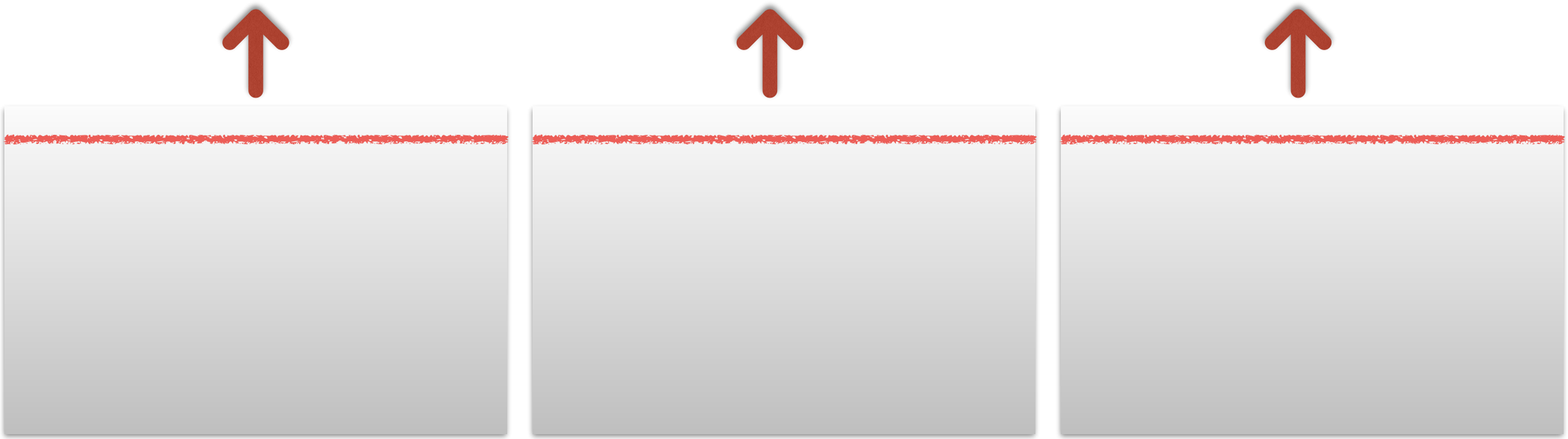
Постановка задачи

- ClickHouse обрабатывает запрос параллельно, с использованием многих процессорных ядер. Распараллеливание, в большинстве случаев, устроено так: существует конвейер выполнения запроса, который состоит из операций преобразований потоков данных. Конвейер начинается с операции чтения данных из таблицы. Уже на этом этапе всё распараллеливается: разные потоки читают разные диапазоны нужных для запроса данных, и каждый поток берёт для чтения следующий кусок данных по мере готовности

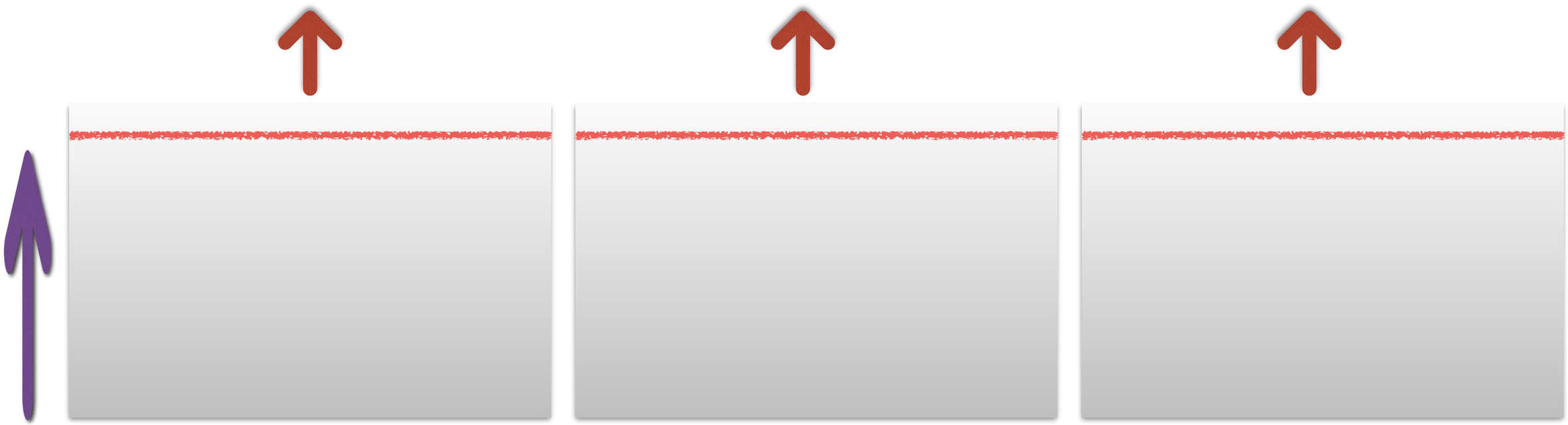
Что происходило до оптимизации



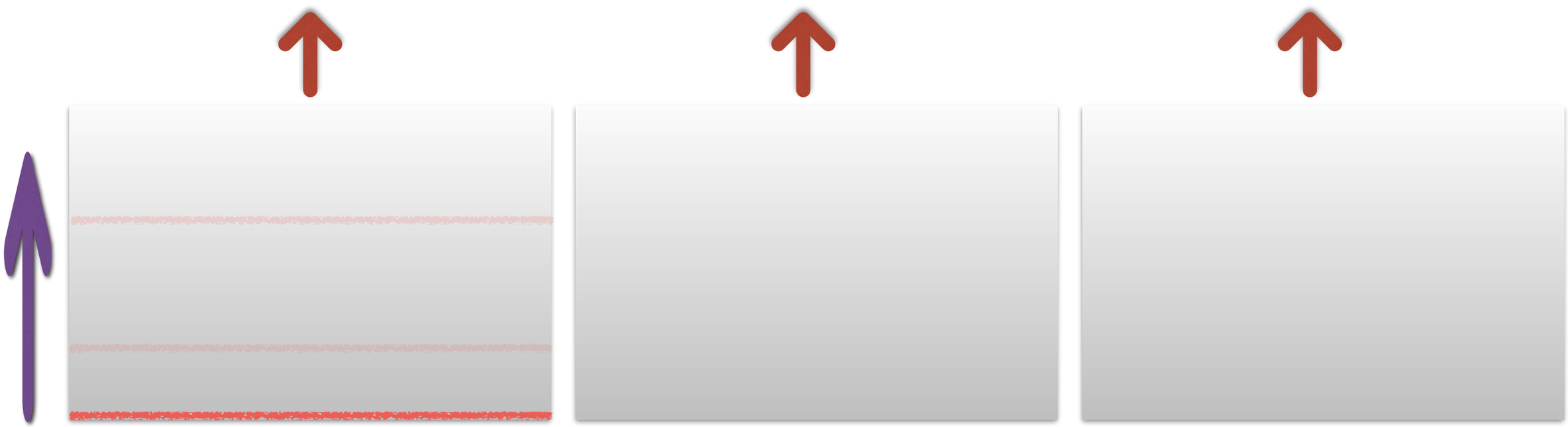
В некоторых случаях данные уже отсортированы в том порядке, который нам нужен



То есть мы можем сделать так



То есть мы можем сделать так



Постановка задачи

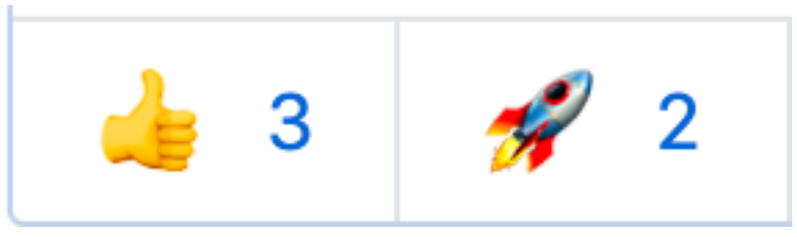
- Данные в таблицах типа MergeTree в ClickHouse хранятся в виде некоторого небольшого количества кусков, каждый из которых отсортирован по указанному при создании таблицы первичному ключу. Рассмотрим выполнение запроса с условием ORDER BY по первичному ключу, либо по любой монотонной функции от первичного ключа (например, по префиксу первичного ключа).

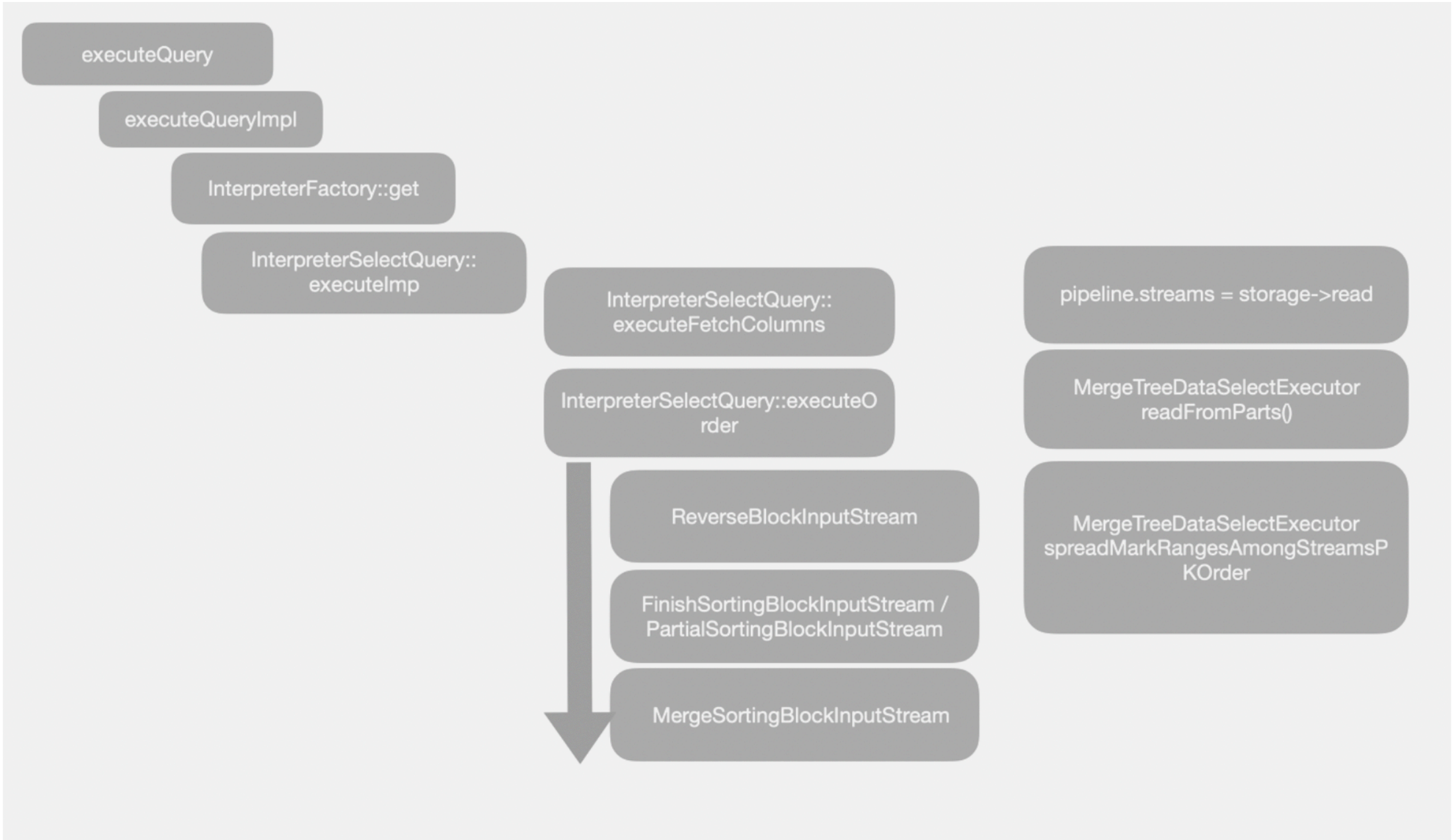
Постановка задачи

- При выполнении такого запроса, можно было бы учесть сортированность данных - читать данные из каждого куска в нужном порядке и мерджить их. Это не очень хорошо, потому что слияние сортированных потоков - довольно ресурсоёмкая операция, которая плохо распараллеливается. Впрочем, иногда можно делать эту операцию после фильтрации данных.

Почему эта задача важна и востребована? Реакция

- Многие запросы сейчас работают достаточно медленно, поступали feature requests от пользователей




executeQuery
executeQueryImpl
InterpreterFactory::get
InterpreterSelectQuery::
executeImp
InterpreterSelectQuery::
executeFetchColumns
InterpreterSelectQuery::executeO
rder
ReverseBlockInputStream
FinishSortingBlockInputStream /
PartialSortingBlockInputStream
MergeSortingBlockInputStream
pipeline.streams = storage->read
MergeTreeDataSelectExecutor
readFromParts()
MergeTreeDataSelectExecutor
spreadMarkRangesAmongStreamsP
KOrder

Итоги

- Был реализован полностью рабочий прототип оптимизации чтения в порядке первичного ключа
- Было добавлено преобразование досортировка и исправлены имеющиеся в ней баги
- Была добавлена возможность чтения в обратном порядке
- Были добавлены stateless и stateful тесты, покрывающие все возможные случаи
- Были добавлены performance-тесты, на которых был виден результат оптимизации

ПРОИЗВОДИТЕЛЬНОСТЬ

SELECT CounterID FROM test.hits ORDER BY CounterID, EventDate LIMIT 50
(500 итераций)

Метрика	Было	Стало	А именно
bytes_per_second	1310812312.518559	3165684875.291606	**в 2.5 больше**
min_time	0.039000	0.011000	**в 3.5 меньше**
queries_per_second	24.619251	59.456864	**в 2.4 раза больше**
rows_per_second	218468718.753093	527614145.881934	**в 2.4 раза больше**
total_time	0.203093	0.084095	**в 2.4 раза меньше**

ПРОИЗВОДИТЕЛЬНОСТЬ

SELECT CounterID FROM test.hits ORDER BY CounterID, EventDate DESC LIMIT 50 (500 итераций)

Метрика	Было	Стало	А именно
bytes_per_second	1031493950.64	137445871.311682	**в 7.5 меньше**
min_time	0.040000	0.010000	**в 4 раза меньше**
queries_per_second	19.373184	87.385732	**в 4.5 раза больше**
rows_per_second	171915658.440646	22907645.218614	**в 7.5 меньше**
total_time	0.258089	0.057218	**в 4.5 меньше**

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Телефон.: +7 (999) 450 7208