RAID в Userspace для ClickHouse

Выпускная квалификационная работа

Глеб Новиков, БПМИ165, специализация «Распределённые системы»
Алексей Миловидов, руководитель группы разработки ClickHouse

2020, Higher School of Economics, Applied Mathematics and Informatics

1 Пара слов про RAID и ClickHouse

2 Про то, зачем RAID в ClickHouse и какие есть альтернативы

3 Какие задачи стояли?

4 Что сделано в рамках каждой задачи и что осталось

5 Результаты и будущие разработки

Уровни памяти

1	Кэш CPU **10Kb-100Mb**	**не персистентно**
2	ОЗУ **x10-100 slower, 1-30Gb**	**не персистентно**
3	SSD **x10k-1000k slower, 100-1000Gb**	**персистентно**
4	HDD **x10kk-100kk slower, 500Gb-10Tb**	**персистентно**

HDD самые **медленные**, содержат **подвижные части**, **дешёвые**

и имеют **самый большой объём**

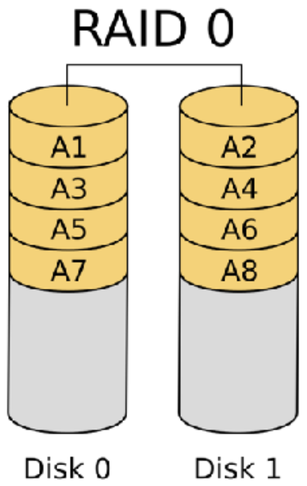
RAID
Redundant **A**rray of **I**ndependent **D**isks

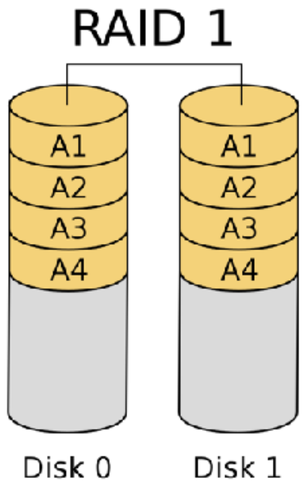
A Case for RAID, David A. Patterson, Garth Gibson and Randy H. Katz, 1988

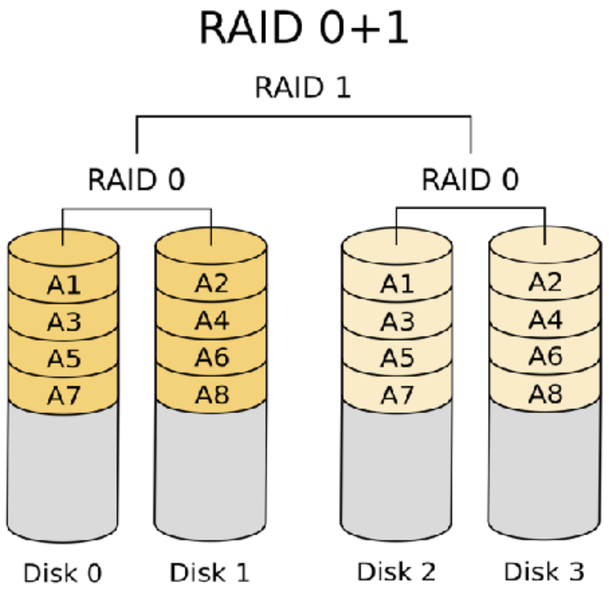
Основные цели внедрения RAID

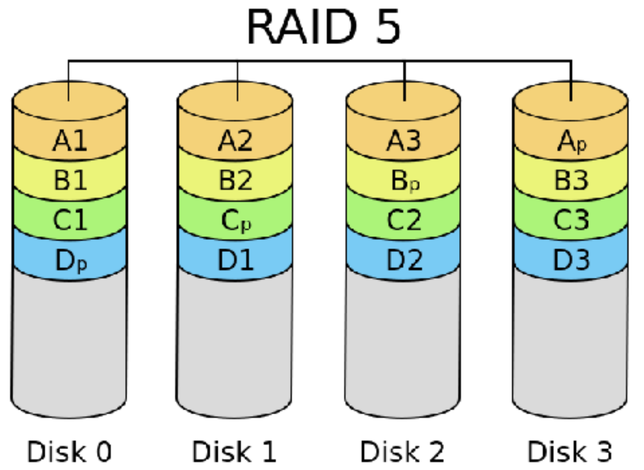
- Ускорение записи / чтения
- Отказоустойчивость
- Дёшево расширять объём без дополнительных усилий

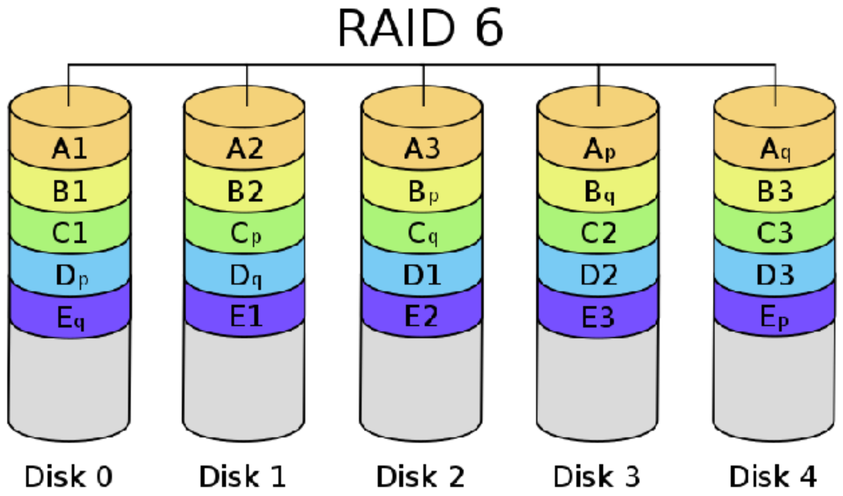
Конфигурации RAID







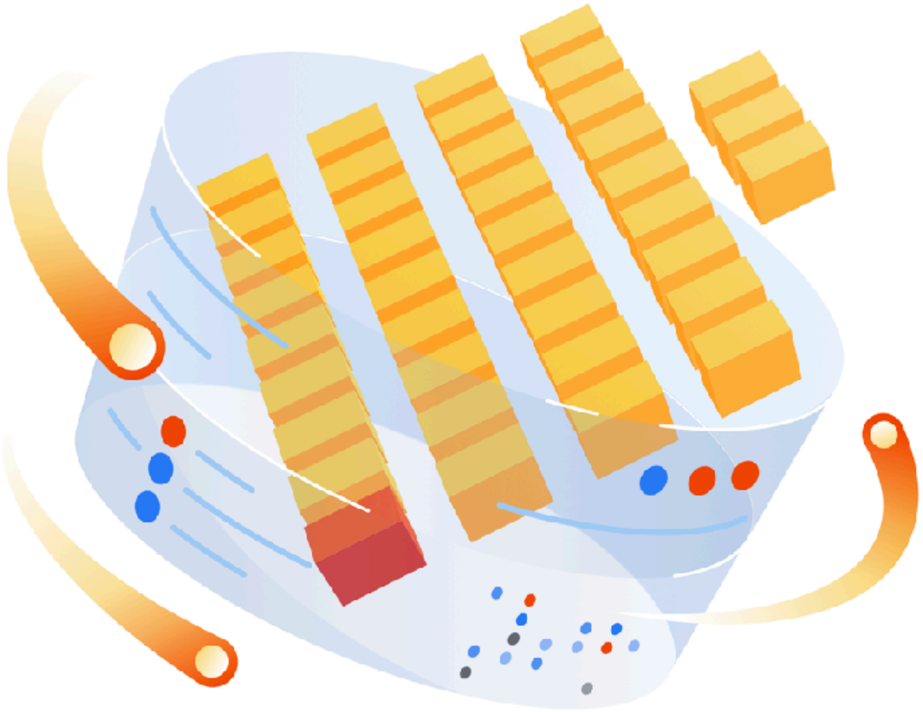




ClickHouse

Аналитическая колоночная СУБД

с открытым исходным кодом



SQL-like язык

Хранение больших объёмов сырых данных

Высокая скорость выполнения запросов

Хотим научить ClickHouse работать над несколькими дисками

Доступные опции RAID

1	Специальные RAID контроллеры	**очень дорого**
2	Программный RAID на уровне драйвера BIOS	**ОС зависит от RAID**
3	Программный RAID на уровне ОС (mdraid)	**нет гибкости для приложения**
4	Реализовать RAID в коде ClickHouse	**займёт какое-то время…OK**

Почему RAID в коде ClickHouse

- Гибкость настройки везде, где есть ClickHouse
- Раскладываем файлы как хотим, не обязательно делать классический RAID
- Возможно, станет быстрее mdraid
- Сможем экономить трафик при восстановлениях реплик из-за повреждений дисков

Задачи проекта

1. Придумать, как делать RAID для ClickHouse
2. Изучить различные механизмы работы с данными в коде
3. Подготовить механизмы работы с данными к работе с несколькими дисками
4. Реализовать прототип записи и чтения с миррорингом (RAID 1) и страйпингом (RAID 0)
5. Протестировать производительность записи и чтения

Задача 1: понять, какие RAID бывают и как мы хотим его реализовать для ClickHouse

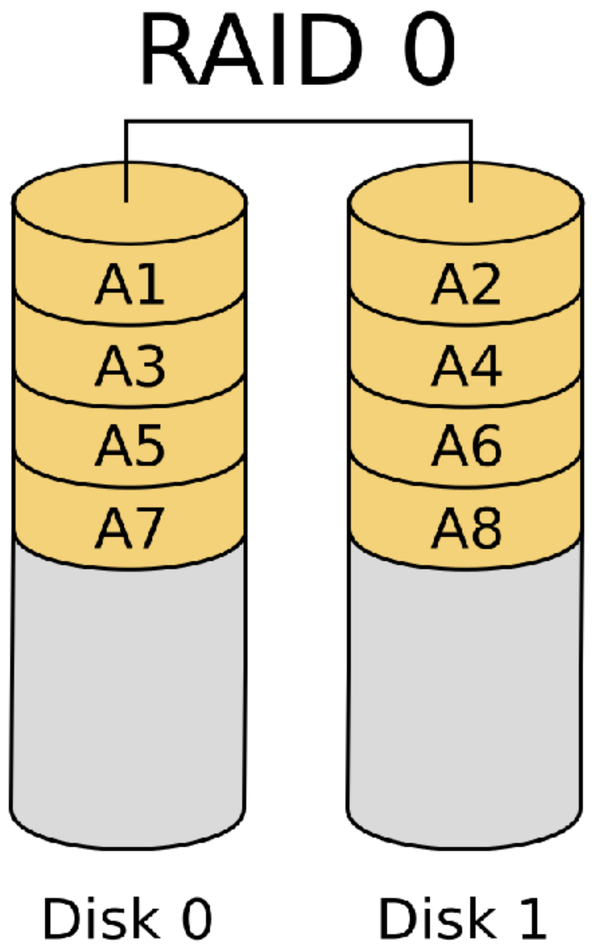
RAID 0

Запись данных

1. Данные бьются на блоки
2. Блоки записываются на несколько дисков

Чтение данных

1. Блоки читаются параллельно
2. Прочитанные блоки собираются в памяти



RAID 0

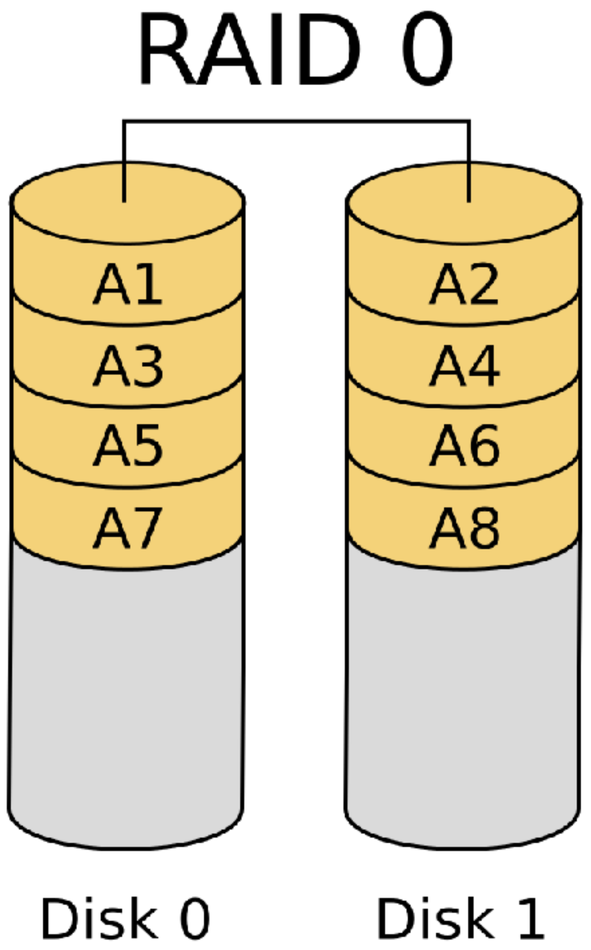
N дисков

Скорость записи **xN**

Скорость чтения **xN**

Никакой устойчивости к поломкам дисков

Объём дисков **складывается**



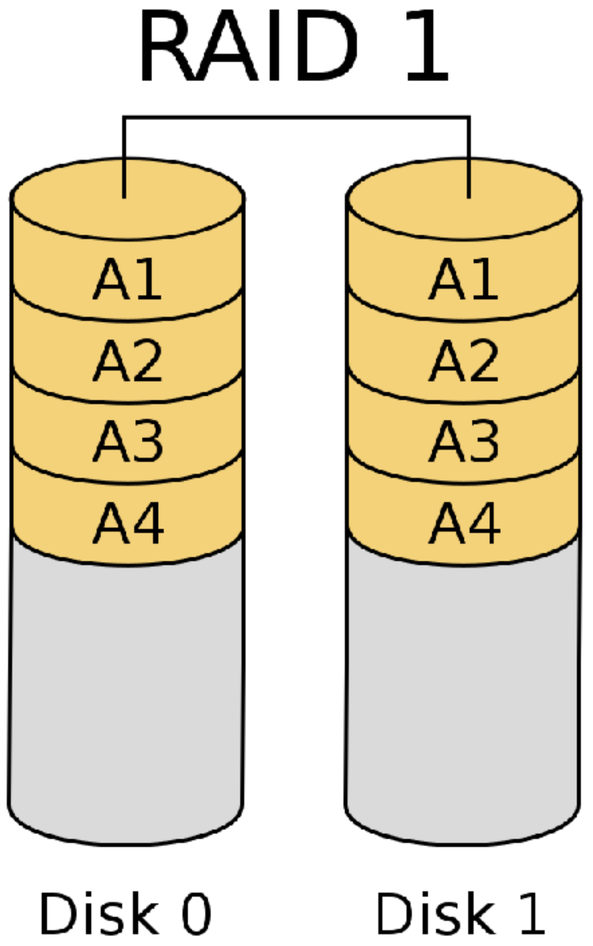
RAID 1

Запись данных

1. Данные пишутся на все диски
2. Завершается после окончания последней записи

Чтение данных

1. Блоки читаются параллельно
2. Прочитанные блоки собираются в памяти



RAID 1

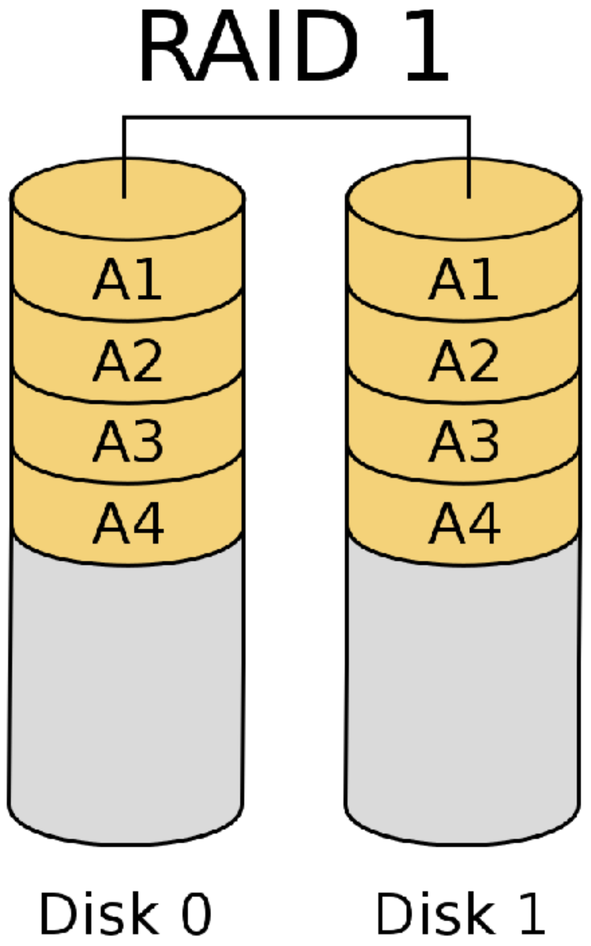
N дисков

Скорость записи **x1**

Скорость чтения **xN**

Переживает смерть **N-1** диска

N * 100% оверхед на объём дисков



RAID 5

Запись данных

1 Данные разбираются на N-1 частей

2 Вычисляется *parity блок:*

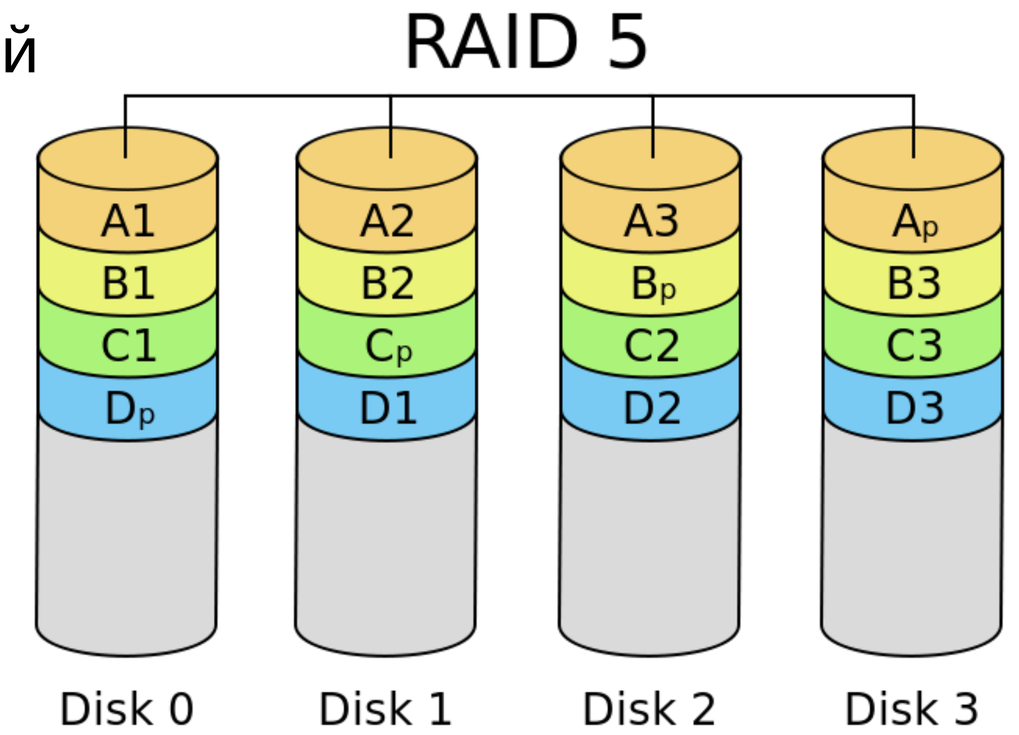
parity_byte = byte1^...^byteN

3 N-1 блоков + parity блок

Чтение данных

1 Блоки читаются параллельно

2 Блоки собираются в памяти, восстанавливаются при необходимости



RAID 5

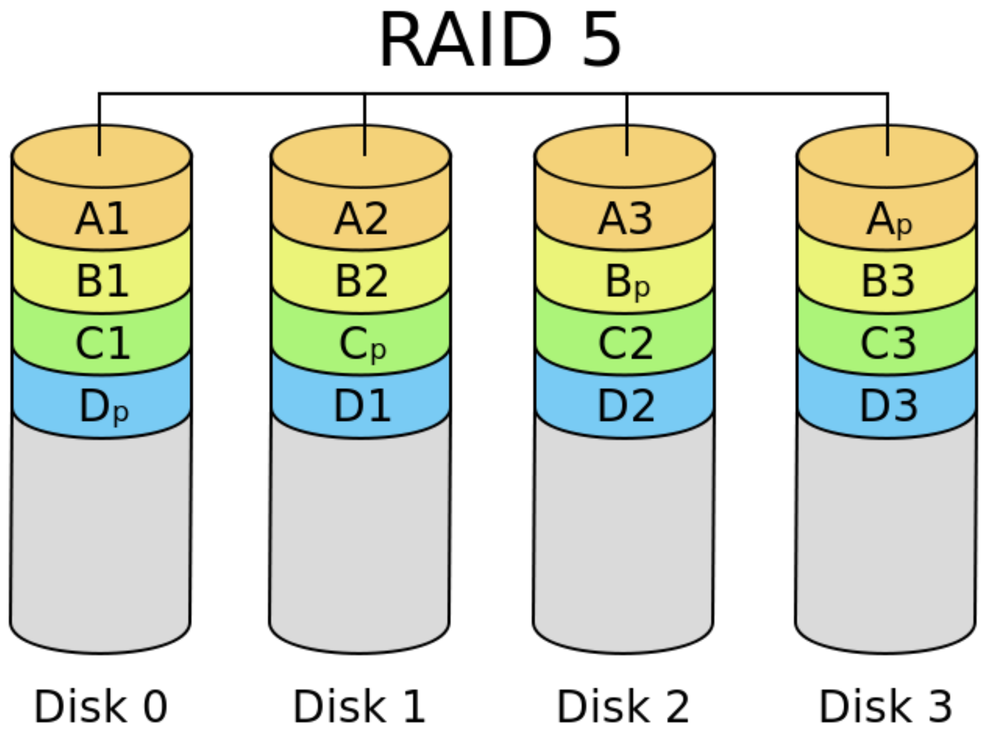
N дисков

Скорость записи **xN-1**

Скорость чтения **xN-1**

Переживает смерть **1** диска

Оверхед в размере **объёма 1 диска**



Таблицы MergeTree

1 Табличные данные хранятся **по колонкам** — отдельный файл на каждую

2 Таблица разделена на **куски (parts)** — блок строк отсортированный по первичному ключу

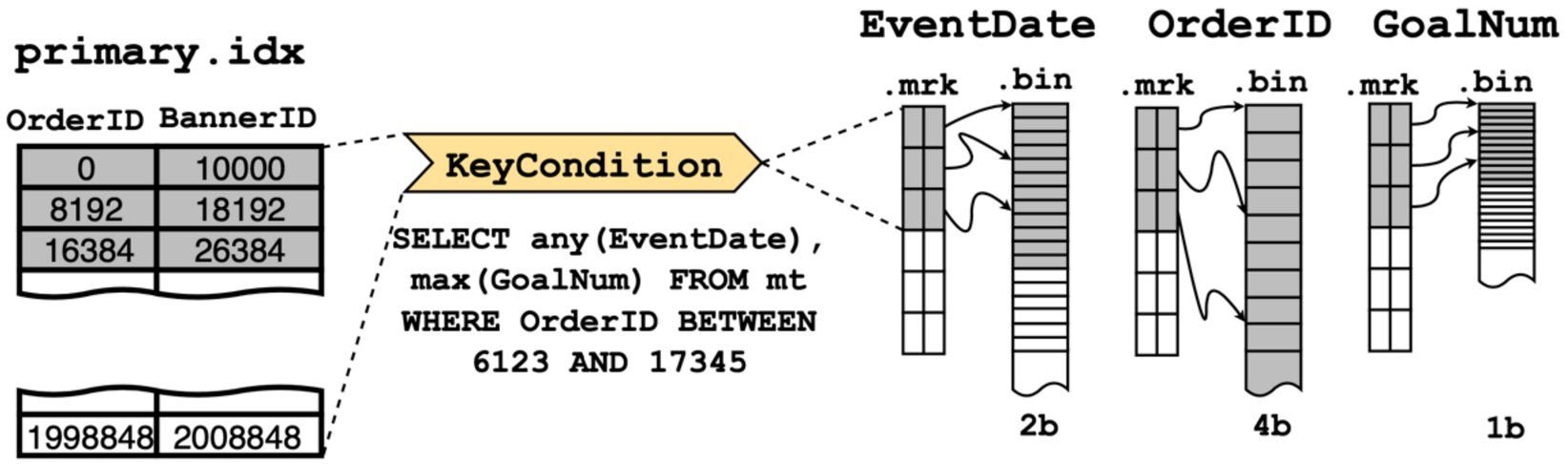
3 Куски сливаются в фоновом процессе

*Table = N parts * M columns*

Таблицы MergeTree

1 Кусок разделён на **гранулы** — какое-то количество строк

2 Индекс разреженный, хранит значение первичного ключа для первой строки каждой гранулы



Как же делать RAID в MergeTree

Как делать striping?

Шардировать на диски будем индекс, засечки и колонки одного куска (part), разбивая по **по гранулам**

Как считать parity_data для неравных блоков?

Чтобы размеры гранул были равны, будем **добивать нулями**

Как сделать быстрое чтение дёшево?

Для ускоренного чтения представим, что **данные на одном диске это целый кусок**, а не его часть

Задача 2: изучить механизмы работы с данными на дисках

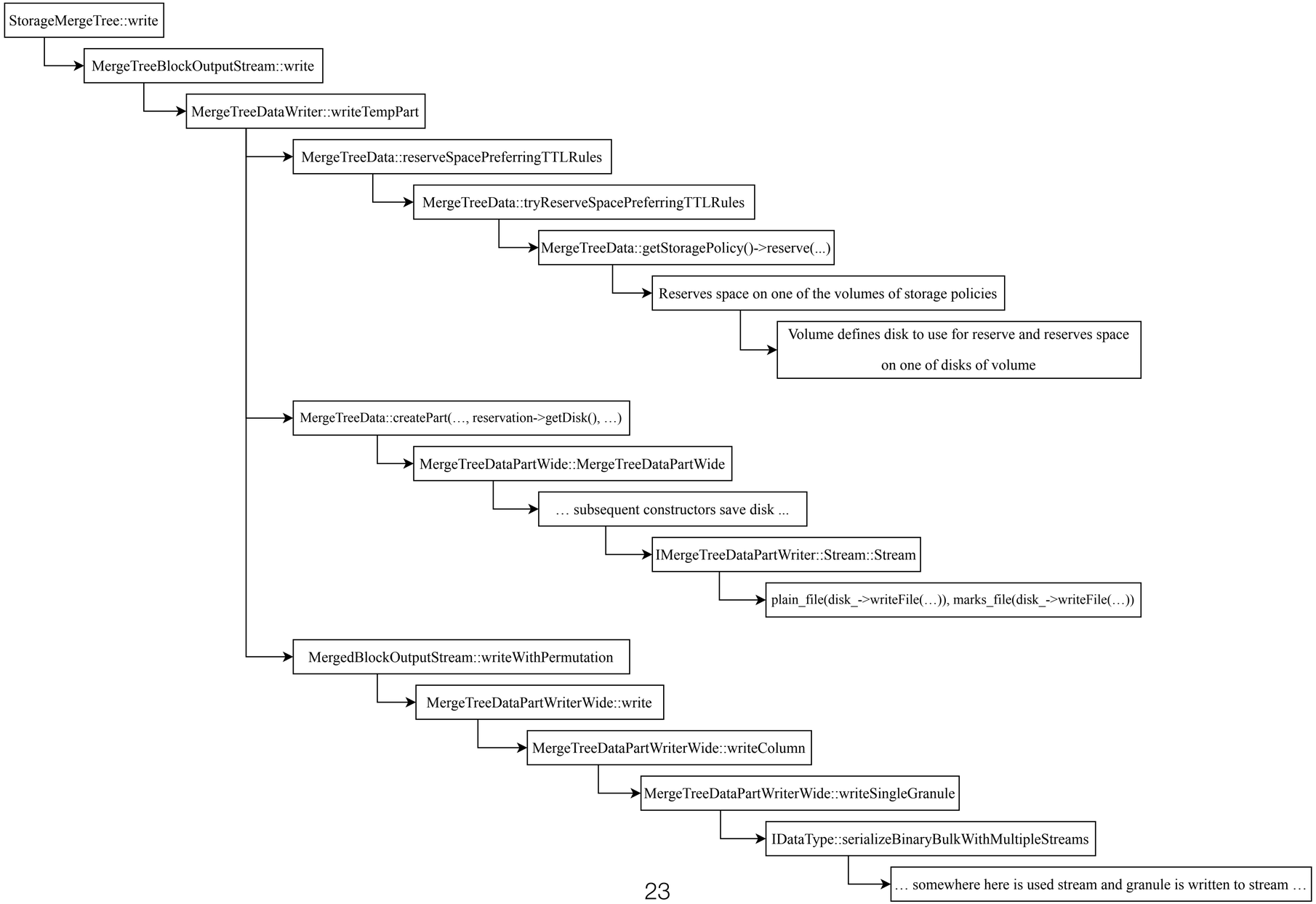
Данные MergeTree на дисках

1. Простая мета-информация и повсеместная работа с ней
2. Запись индекса, колонок и засечек в реализациях интерфейса *IMergeTreeDataPartWriter*
3. Чтение данных через *QueryPipeline* внутри *MergeTreeSelectProcessor*
4. Слияние кусков в *MergeTreeDataMergerMutator*
5. Перемещение кусков на другие вольюмы в *MergeTreePartsMover*

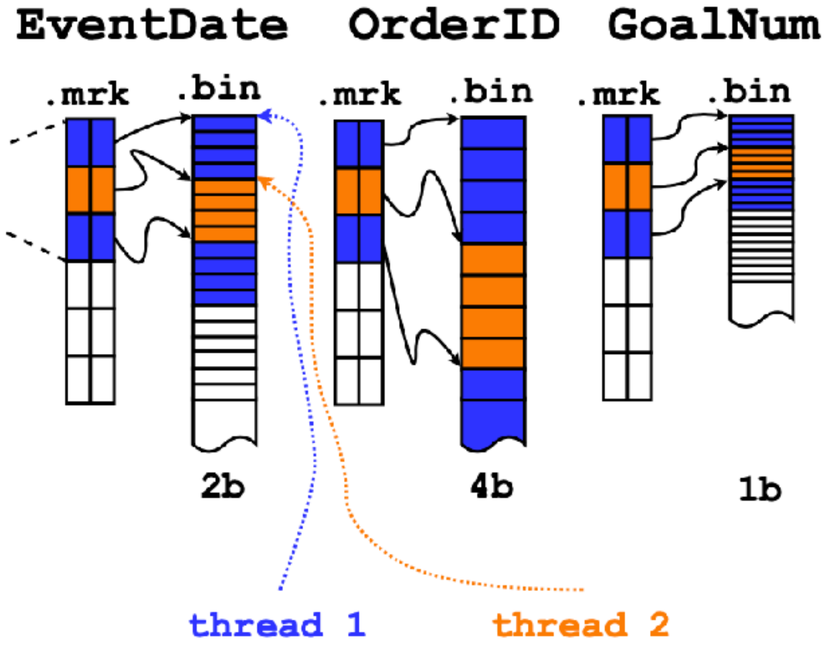
Данные MergeTree на дисках

1. Простая мета-информация и повсеместная работа с ней
2. **Запись индекса, колонок и засечек в реализациях интерфейса *IMergeTreeDataPartWriter***
3. **Чтение данных через *QueryPipeline* внутри *MergeTreeSelectProcessor***
4. Слияние кусков в *MergeTreeDataMergerMutator*
5. Перемещение кусков на другие вольюмы в *MergeTreePartsMover*

Запись данных куска



Чтение куска



Пул потоков читает гранулы

Проблемы

1. Весь MergeTreeDataPart работает над одним диском и не предполагает ничего иного
2. Кода много
3. Не всё нужно переписывать для нескольких дисков

Задача 3: поддержать работу с несколькими дисками и добавить абстракции

Переписывание кода MergeTreeData*

1 Новый интерфейс *IVolume*

2 Абстрактный *VolumePtr* во всём коде MergeTree

3 Разделение использований *VolumePtr* и переезд на резонные реализации (где-то *SingleDiskVolume*, где-то *VolumeJBOD*)

3 Отселение записи индекса в отдельный интерфейс *IMergeTreeDataPartIndexWriter*

4 Выделение политик записи в *MergeTreeDataPartWriterWide*

Задача 4: реализовать прототип RAID 0 и RAID 1

Что успели по части RAID?

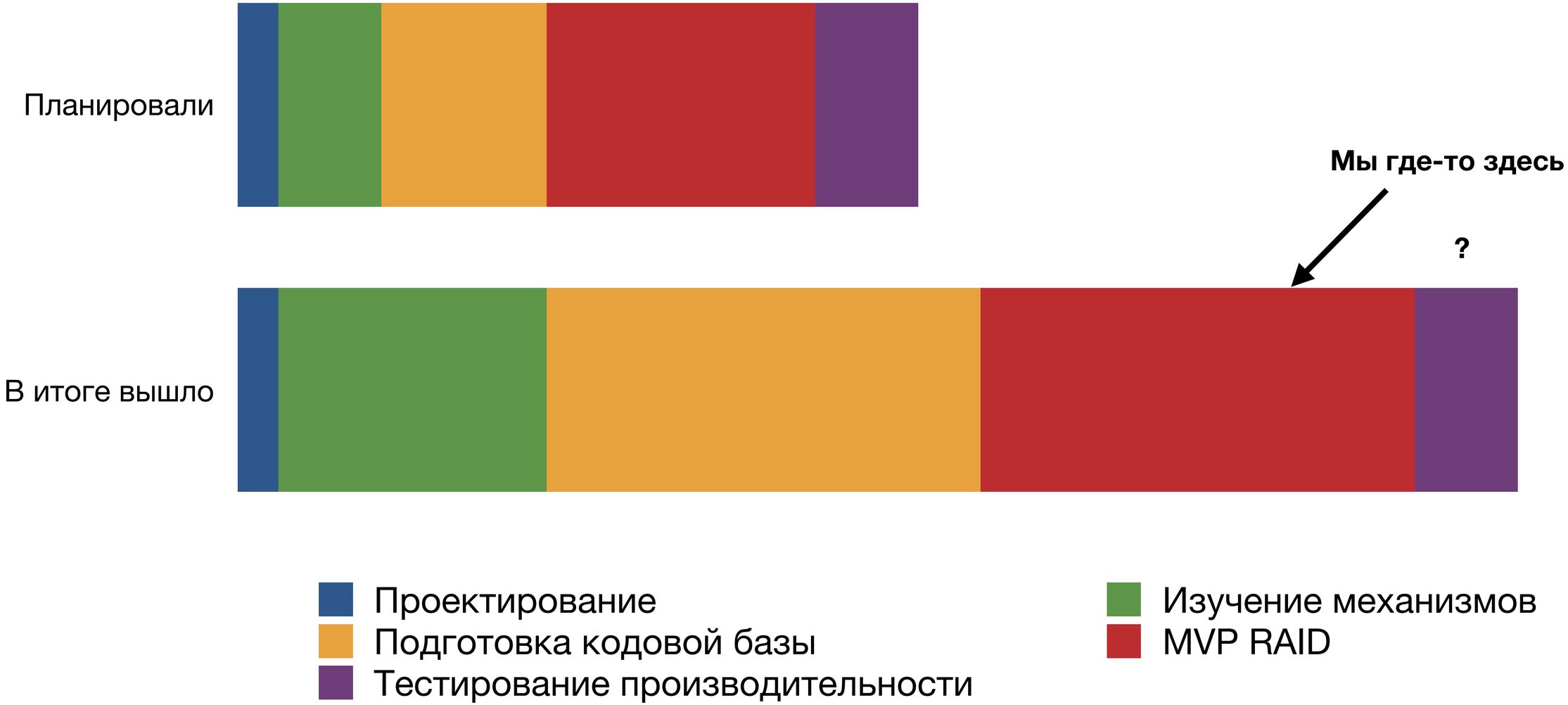
1. *VolumeRAID*
2. Конфигурирование типов RAID через *config.xml*
3. Прототип простого *MergeTreeDataPartIndexWriterRAID1*
4. Прототип простого *MergeTreeDataPartWriterWideRAID1*

Что ещё надо сделать?

1. Объединяющая абстракция для записи индексов и колонок
2. Поддержать в *MergeTreeSelectProcessor::getNewTask* чтение с нескольких дисков
3. Проверить слияние и перемещение кусков
4. Восстановление или копирование данных после смерти одного из дисков

Ретроспективно про прогресс

Ретроспективно про прогресс



Какие результаты?

1. Изучили различные конфигурации RAID и детали реализации
2. Подробно изучили архитектуру MergeTree
3. Переработали части MergeTree и адаптировали для работы над несколькими дисками
4. Продумали детали реализации различных RAID для MergeTree
5. Подготовили архитектуру MergeTree для реализации различных конфигураций RAID
6. Реализовали простой прототип RAID 1

Возможное продолжение разработки

1. После успешной реализации RAID 1 надо сделать RAID 0, RAID 5
2. Тесты производительности
3. Рассмотреть более экзотические реализации RAID
4. Продумать оптимизации для восстановления реплик
5. Можно реализовать RAID на уровне *IDisk*, но непонятно зачем, так как получится *mdraid*

Спасибо

Глеб Новиков, *ganovikov@edu.hse.ru*