Яндекс

Внутреннее устройство MergeTree

Алексей Зателепин

Преимущества MergeTree

- Вставки и чтения не блокируют друг друга
- Атомарная вставка
- Индекс для range-запросов, сэмплирование
- Партиционирование и запросы ALTER PARTITION
- Фоновые операции над записями с одинаковым PK
- Репликация
- Запросы ALTER COLUMN
- Мутации (в разработке)

Основная проблема и решение

- Для range-запросов данные должны быть упорядочены по ключу

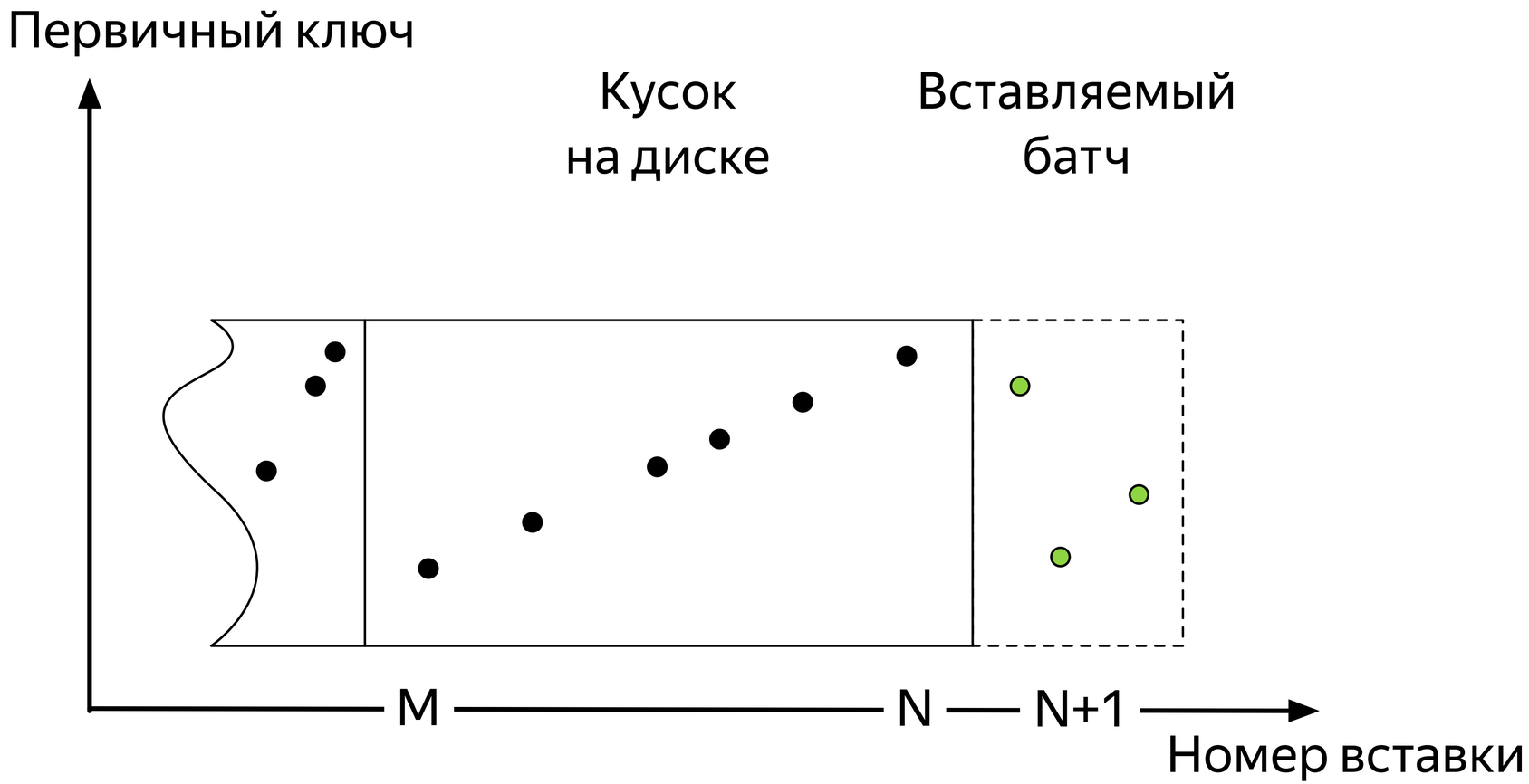
  Например, по `(CounterID, Date)`

- Но поступают они упорядоченными по времени

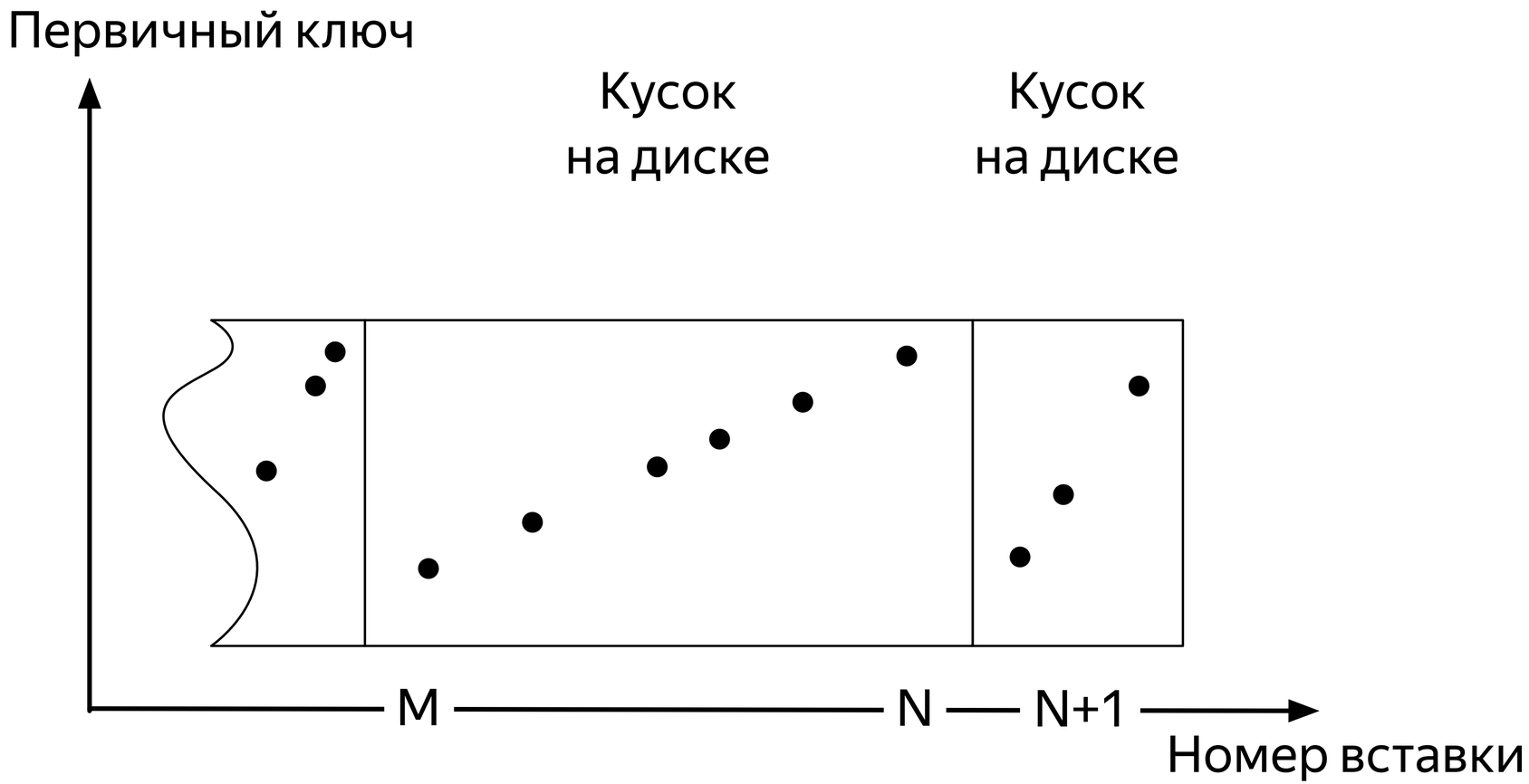
  (почти)

- Поддерживаем небольшой набор упорядоченных “кусков”

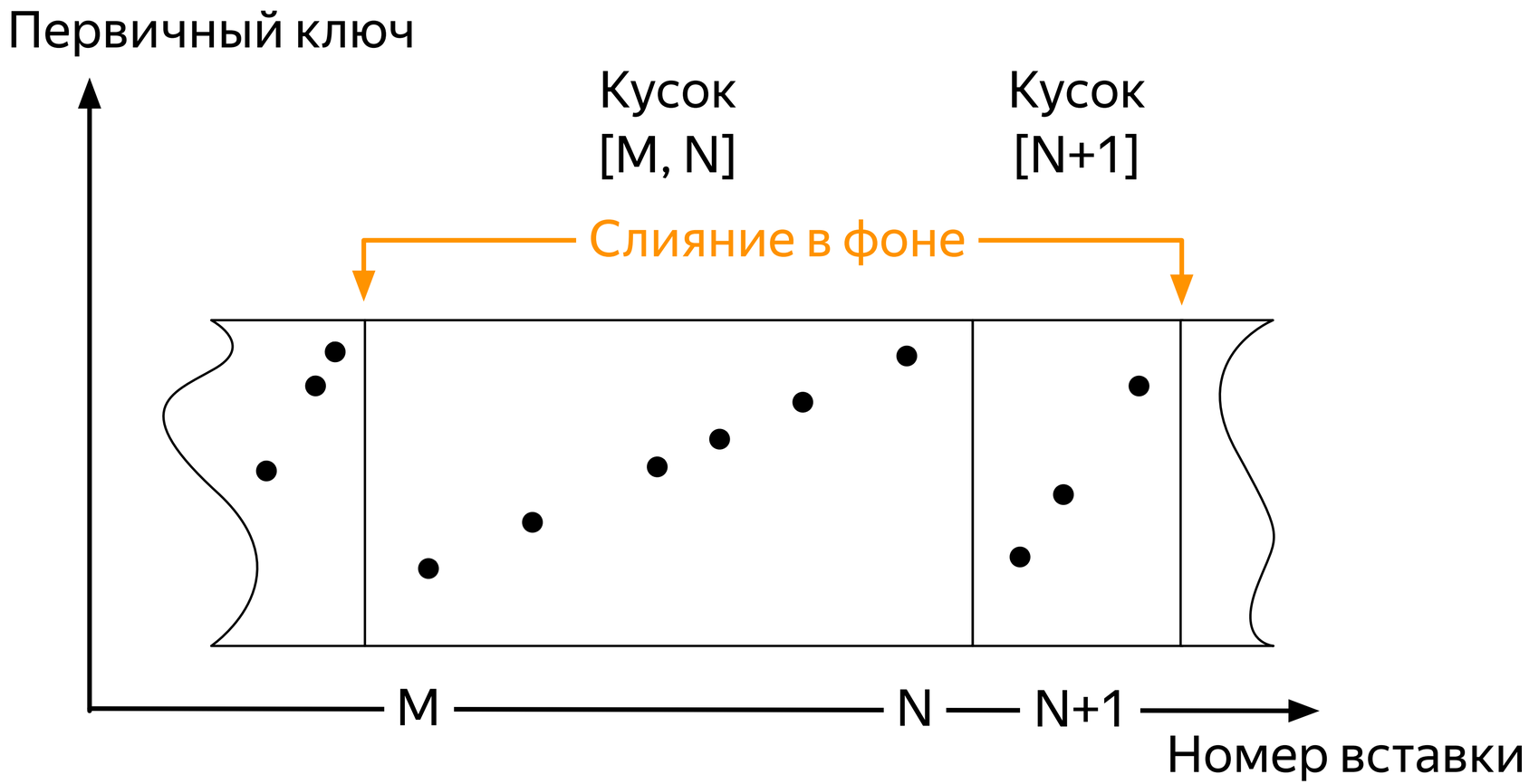
Основная идея



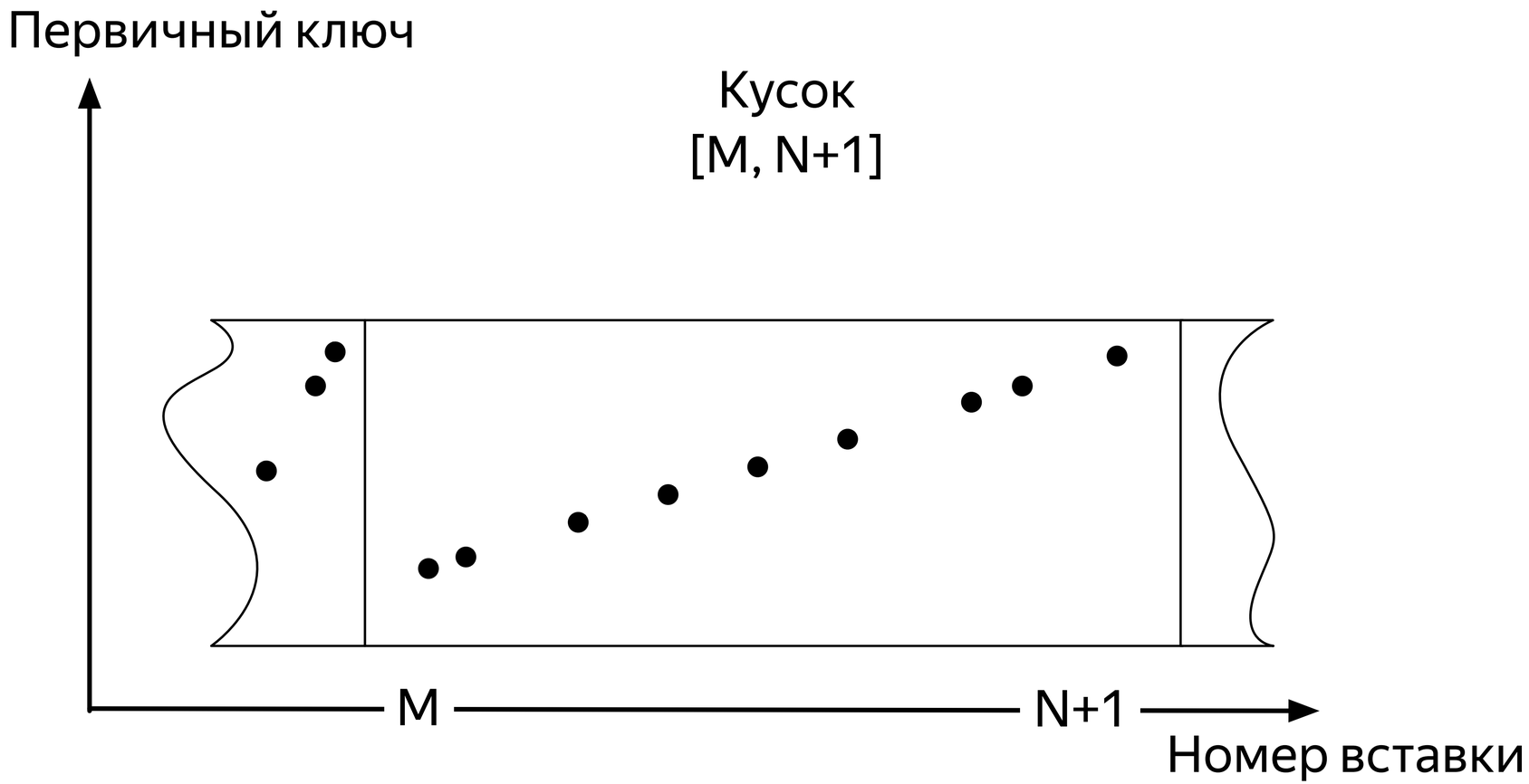
Основная идея



Основная идея



Основная идея



Данные куска

class MergeTreeDataPart

- Сжатые данные столбцов (на диске / в page cache)
- Разреженный индекс
- Чексуммы
- Значение ключа партиционирования
  Min/max значения столбцов ключа партиционирования

Метаданные куска

ID куска: `201805_12_34_6` (новый стиль)

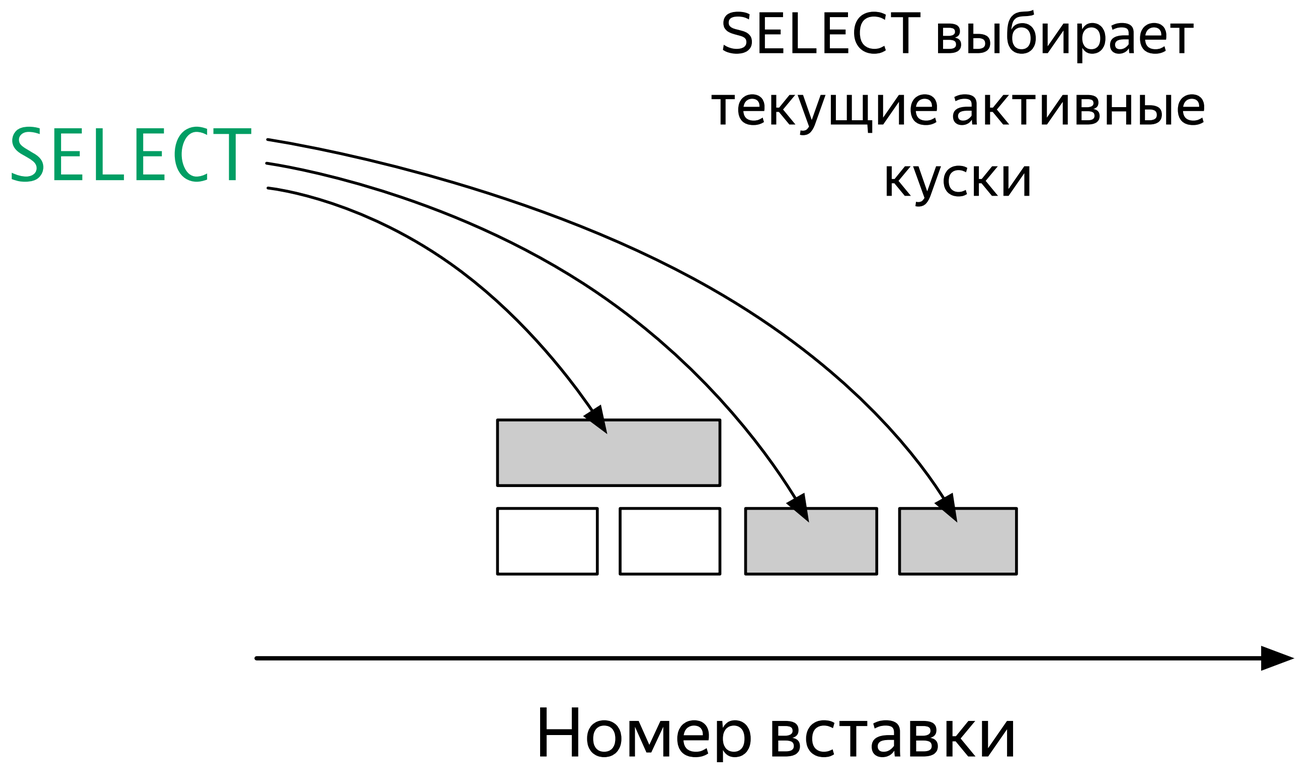
```
struct MergeTreePartInfo
{
    String partition_id; // = “201805”
    Int64 min_block;     // = 12
    Int64 max_block;     // = 34
    UInt32 level;        // = 6

    bool contains(const MergeTreePartInfo & other) const;
};
```

Множество активных кусков

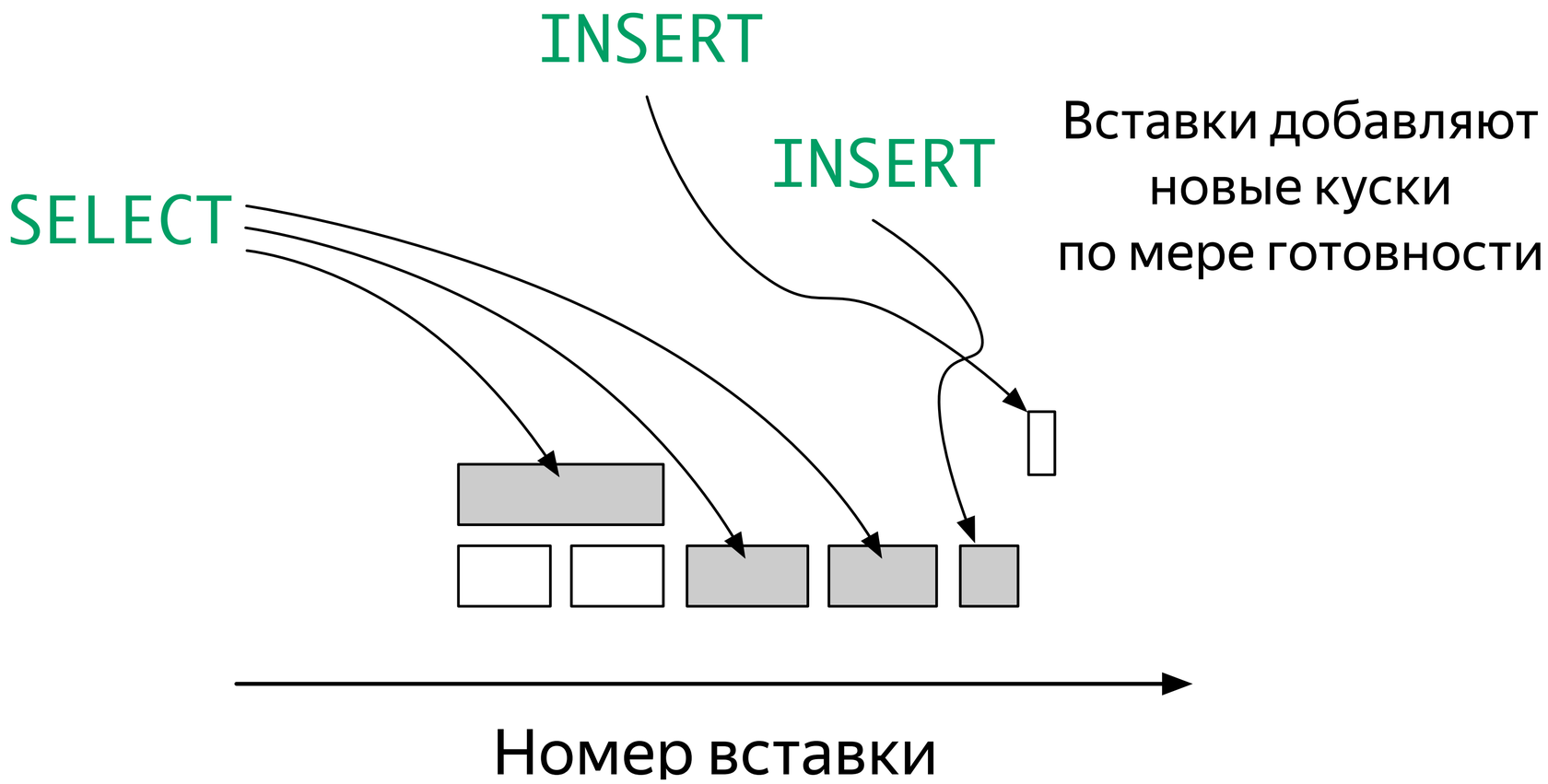
`class MergeTreeData` – персистентная структура данных

- Сами куски неизменяемы (почти)
- В любой момент есть набор активных кусков
  Активные куски не пересекаются по номерам блоков
- Куски можно добавлять или делать активными/неактивными
- Куски, на которые никто не ссылается, можно удалять

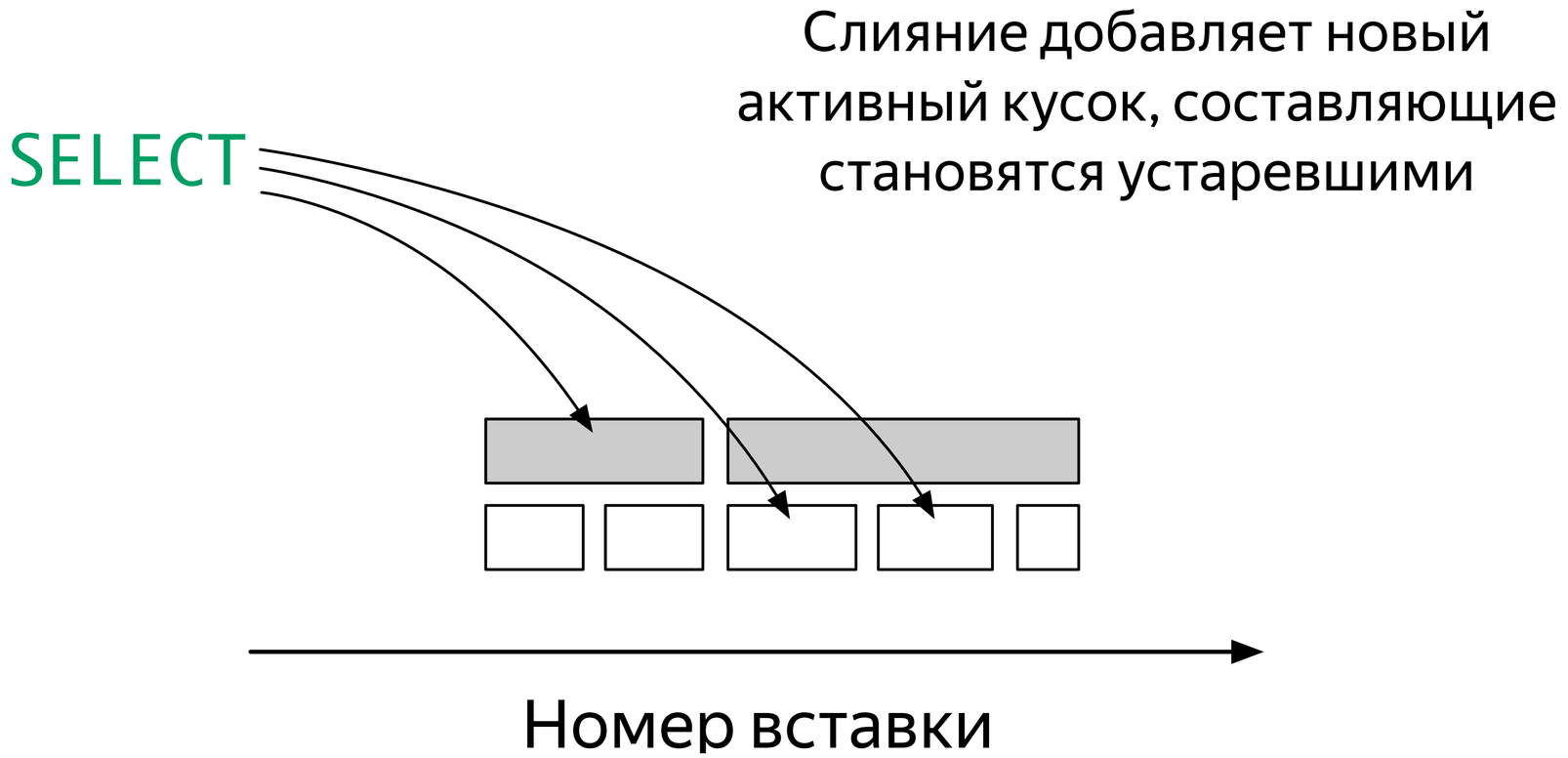
Множество активных кусков



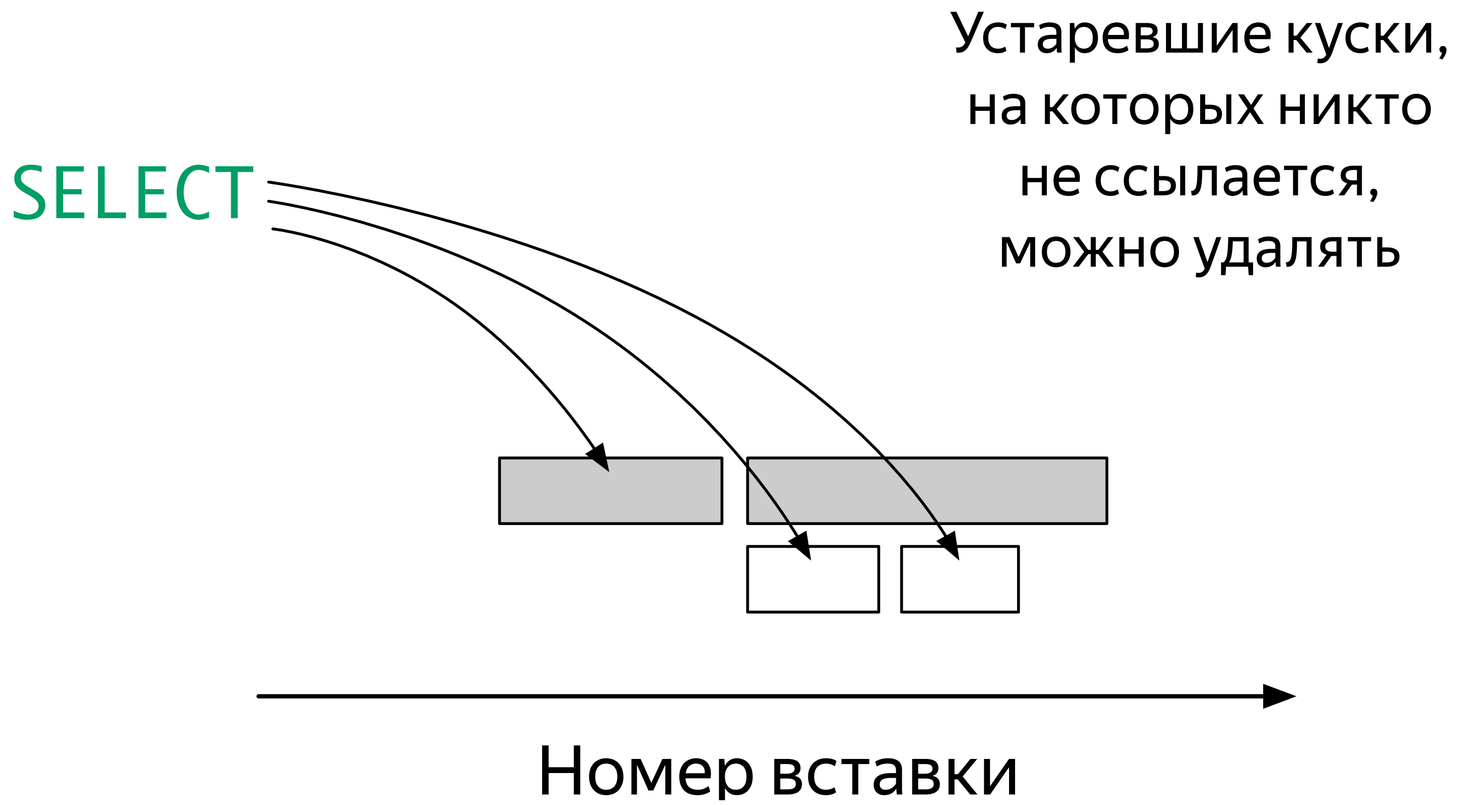
Множество активных кусков



Множество активных кусков



Множество активных кусков



Репликация

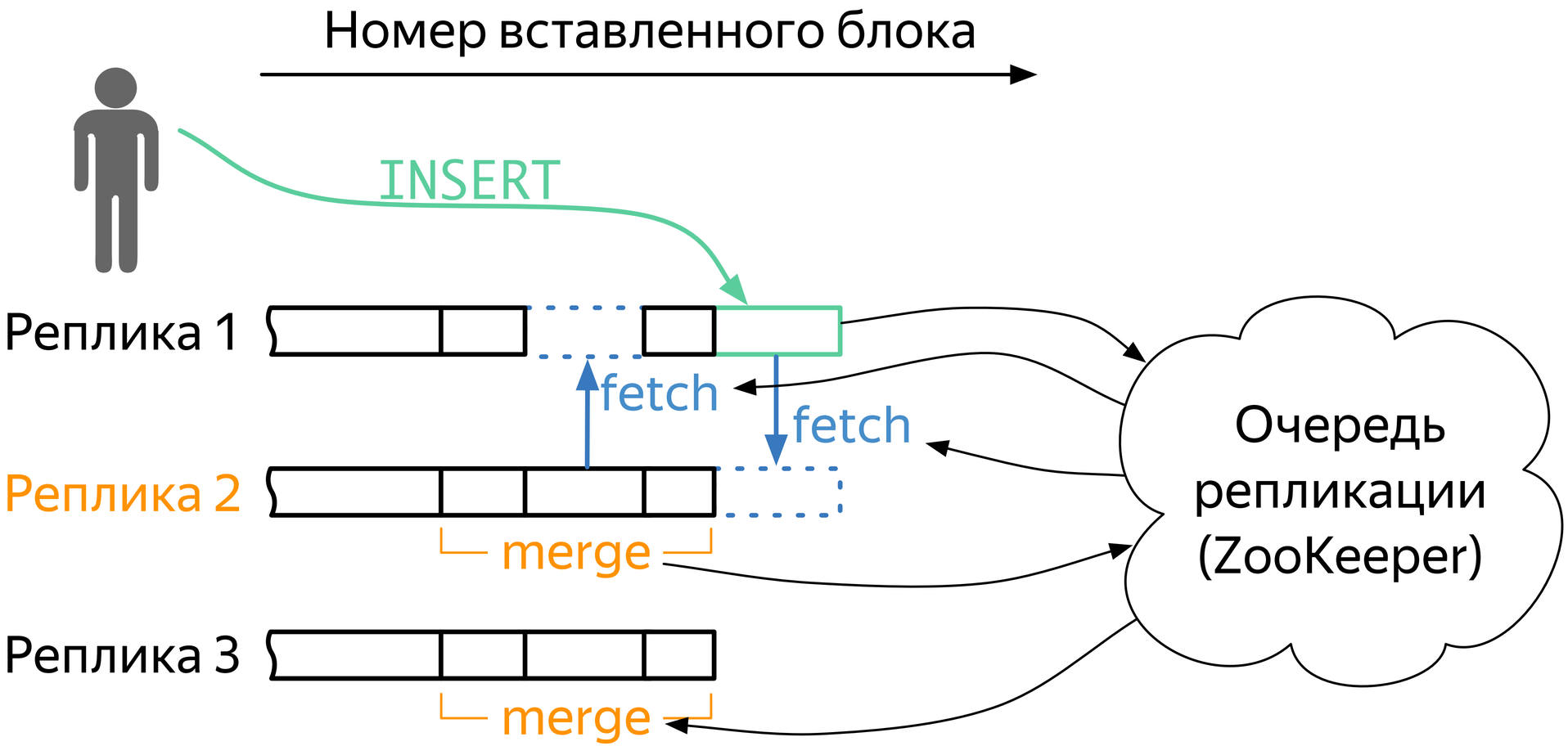
Асинхронная мастер-мастер репликация

- Работает на уровне таблицы

Реплики стараются поддерживать набор кусков одинаковым

- Скачивают новые куски друг у друга
- Выполняют одинаковую последовательность слияний
- Сверяют чексуммы кусков

Репликация


Номер вставленного блока
INSERT
Реплика 1
fetch
fetch
Реплика 2
merge
Реплика 3
merge
Очередь
репликации
(ZooKeeper)

Репликация: вставка

`class ReplicatedMergeTreeBlockOutputStream`

- Записываем данные во временный кусок
- Получаем номер блока (`class AbandonableLock`)
- Переименовываем, добавляем в `MergeTreeData` в состоянии `PreCommitted`
- Добавляем в ZK (`/log/, /blocks/, /replicas/<r>/parts/`)
- Меняем состояние на `Committed`
  (или выкидываем или отправляем кусок на проверку)

Репликация: выполнение заданий из лога

- Подписываемся на изменения в папке `/log/` в ZK
- Копируем свежие записи из `/log/` в `/replicas/<r>/queue` и в память (`class ReplicatedMergeTreeQueue`)
- Выполняем записи в фоне (`executeLogEntry()`)
  Например, `GET_PART`:

  Ищем кусок (такой же или больший) на других репликах

  Скачиваем (`class DataPartsExchange::Fetcher`)

  Сверяем чексуммы и добавляем в ZK и `MergeTreeData`

Репликация: назначение слияний

Инвариант: кусок может поучаствовать не более, чем в 1 слиянии

- Выигрываем выборы лидера (`/leader_election` в ZK)
- Далее, в цикле (`mergeSelectingThread()`):

  Обновляем очередь (считываем `/log`)

  Пытаемся назначить слияния кусков, если между ними нет активных вставок (проверяем `AbandonableLock`-и)

  Записываем назначенное слияние в ZK в `/log`

Мутации

Возможность изменить данные таблицы
Пример: `ALTER DELETE`

- Готовим измененные куски и добавляем их вместо старых
  Пример: `201805_12_34_6 → 201805_12_34_6_89`
- Тяжелая операция
  (требует перезаписи целых кусков)
- Считаем, что не коммутирует со слияниями
- Куски образуют дерево “слияний-мутаций”
  (могут участвовать либо в 1 мутации, либо в 1 слиянии)

Мутации

Demo Time!

Вопросы?