Yandex

Внутреннее устройство ClickHouse

Алексей Зателепин

Задачи, для которых подходит ClickHouse

Есть поток событий

- Действия пользователя на сайте Yandex Metrica
- Показы рекламы
- DNS запросы
- Транзакции e-commerce
- ...

Хотим сохранять эти события и делать из них какие-то выводы

Философия ClickHouse

- Интерактивные запросы по свежим данным
- Нужны структурированные очищенные данные
- Стараемся заранее ничего не агрегировать
- Язык запросов: диалект SQL + расширения

Пример запроса в системе веб-аналитики

Top-10 рефереров за неделю.

```
SELECT Referer, count(*) AS count
FROM hits
WHERE CounterID = 111 AND Date >= today() - 7
GROUP BY Referer
ORDER BY count DESC
LIMIT 10
```

Как выполнить запрос *быстро?*

Быстро прочитать данные

- Только нужные столбцы: `CounterID, Date, Referer`
- Локальность данных (нужен индекс!)
- Сжатие данных

Как выполнить запрос *быстро?*

Быстро прочитать данные

- Только нужные столбцы: `CounterID, Date, Referer`
- Локальность данных (нужен индекс!)
- Сжатие данных

Быстро обработать данные

- Векторный движок (обработка по блокам)
- Распараллеливание по всем ядрам и машинам
- Специализация и низкоуровневые оптимизации

Нужен индекс!

Выбираем по тому же принципу, что и в классических СУБД

Большинство запросов будут содержать условия на CounterID и (возможно) на Date

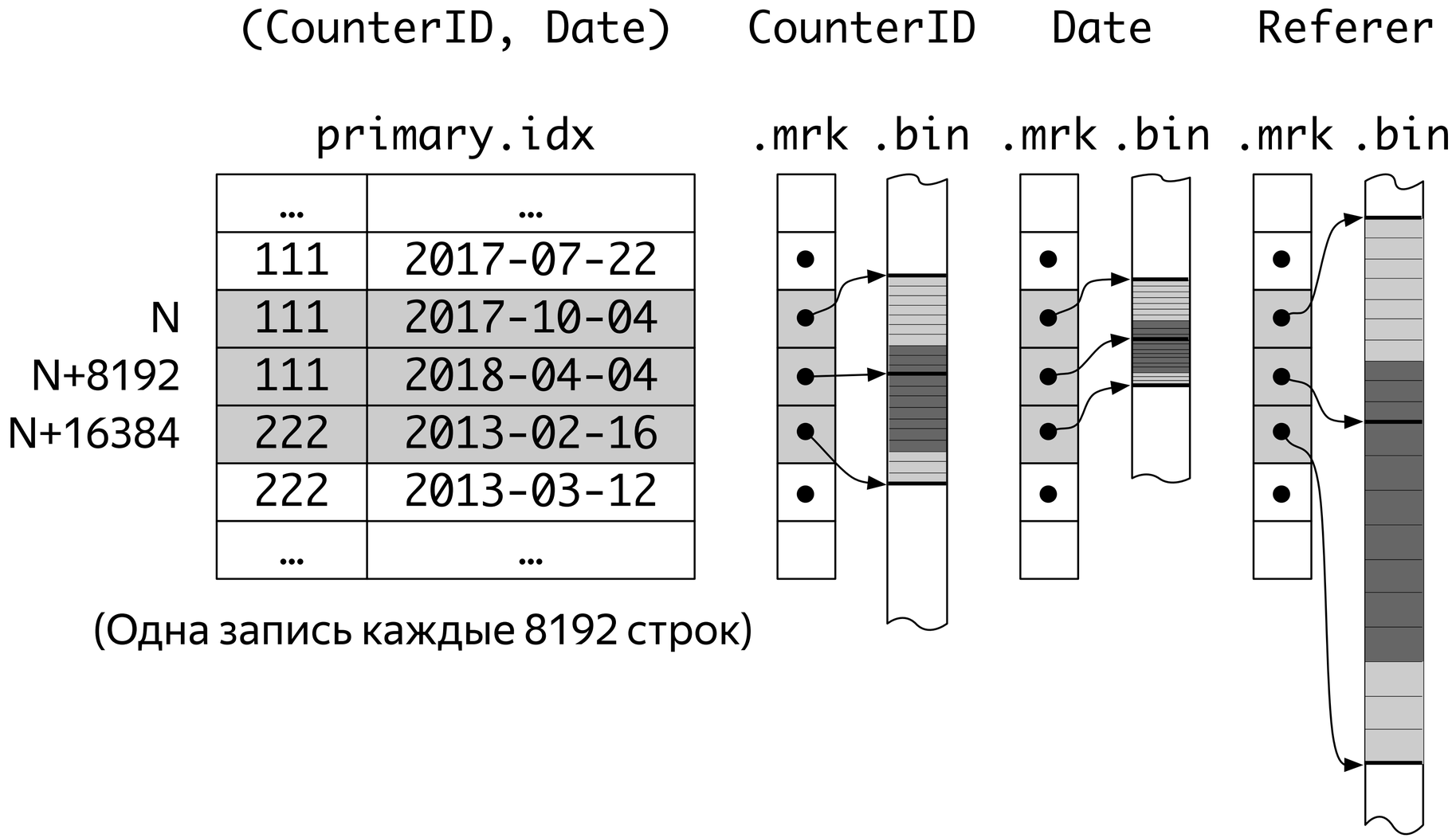
`(CounterID, Date)` подходит

Проверяем, мысленно сортируя таблицу по ключу индекса

Отличия

- Таблица будет физически упорядочена на диске
- Не обеспечивает уникальность

Внутреннее устройство индекса



(Одна запись каждые 8192 строк)

Что нужно помнить про индекс

Индекс разреженный

- Должен помещаться в память
- Гранулярность по умолчанию (8192) скорее всего ОК
- Не обеспечивает уникальности
- Производительность точечных запросов не идеальна

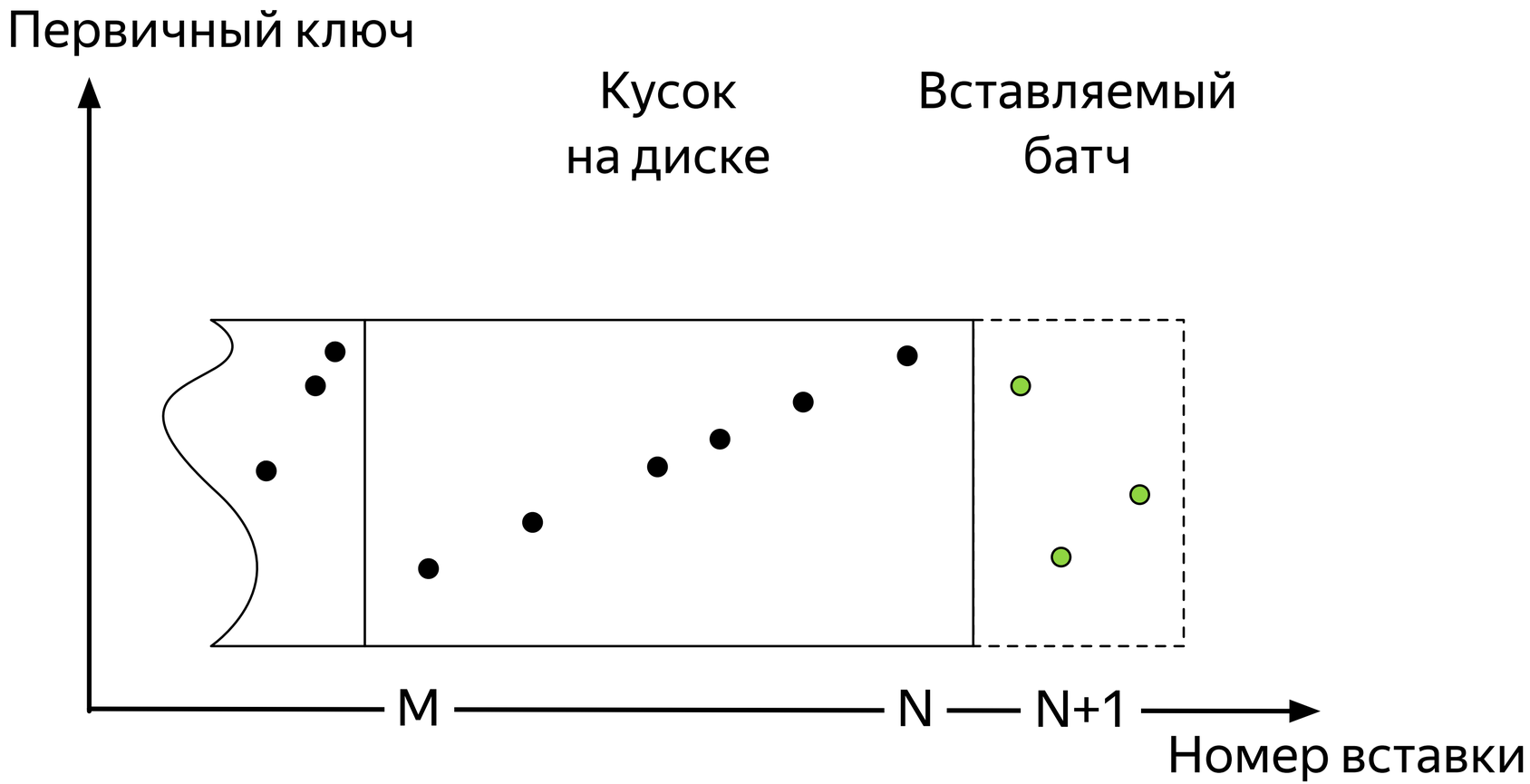
Таблица упорядочена по выражению индекса

- У таблицы может быть только один индекс
- Использование индекса всегда выгодно

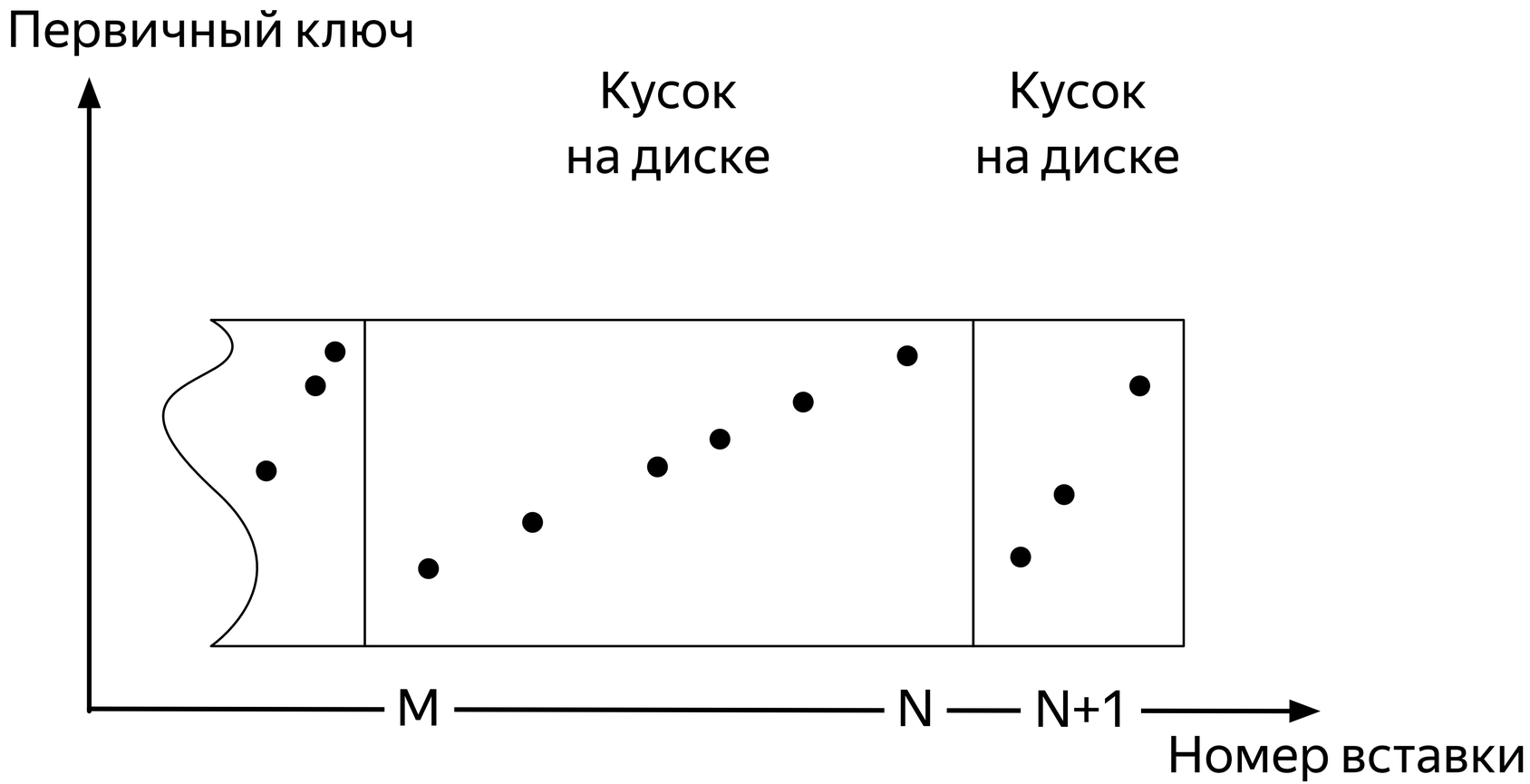
Как поддерживать таблицу упорядоченной

- Поступающие события (почти) упорядочены по времени
  - А нам нужно упорядочить по первичному ключу!
- MergeTree: поддерживаем небольшой набор отсортированных кусков
  - Идейно напоминает LSM-дерево

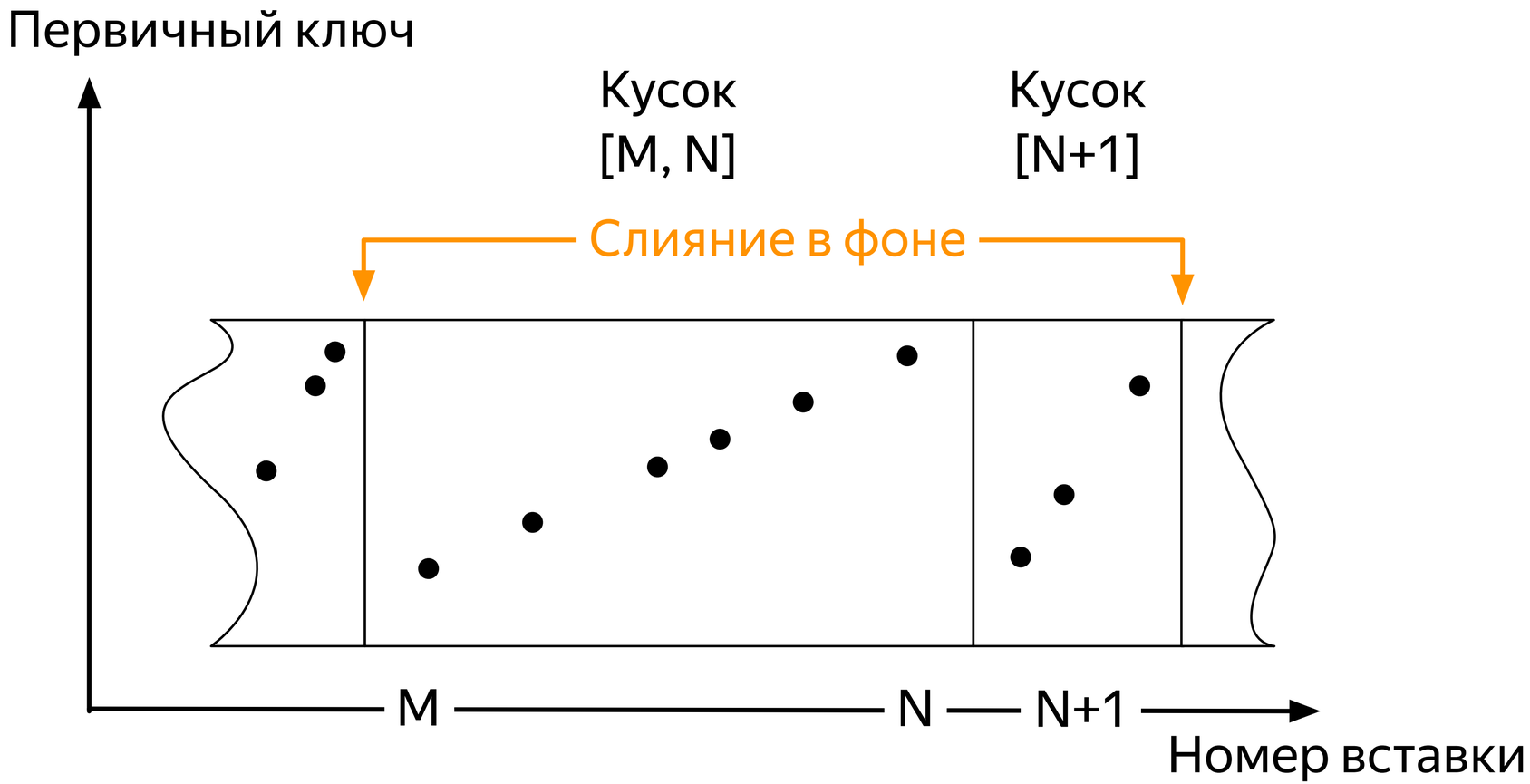
Как поддерживать таблицу упорядоченной



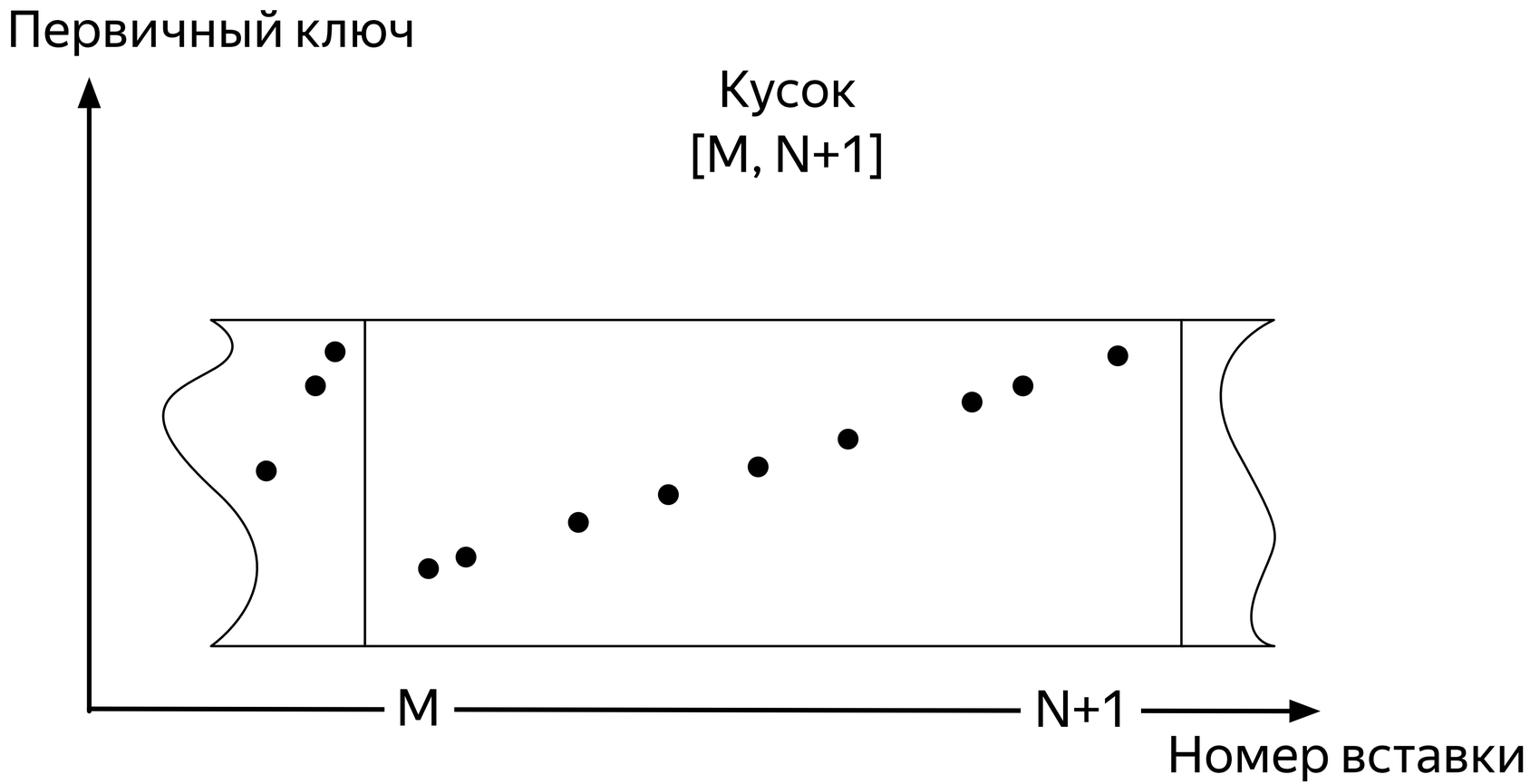
Как поддерживать таблицу упорядоченной



Как поддерживать таблицу упорядоченной



Как поддерживать таблицу упорядоченной



Что можно делать во время слияния

- Удалять старые записи
  - ReplacingMergeTree
  - CollapsingMergeTree
- Заранее агрегировать
  - AggregatingMergeTree
- Прореживание метрик
  - GraphiteMergeTree

Партиционирование MergeTree

```
ENGINE = MergeTree … PARTITION BY toYYYYMM(Date)
```

- Можно партиционировать по любому выражению
  (по умолчанию: по месяцу)
- Куски, принадлежащие различным партициям, не сливаются
- Данными в одной партиции легко манипулировать

```
ALTER TABLE DROP PARTITION
ALTER TABLE DETACH/ATTACH PARTITION
```

- MinMax индекс по столбцам ключа партиционирования

Что нужно помнить про MergeTree

Слияния идут в фоне

- Даже, когда нет запросов!

Количество кусков не должно быть слишком большим

- Частота вставок
- Метрики `MaxPartsCountForPartition` и `DelayedInserts`

Когда одного сервера недостаточно

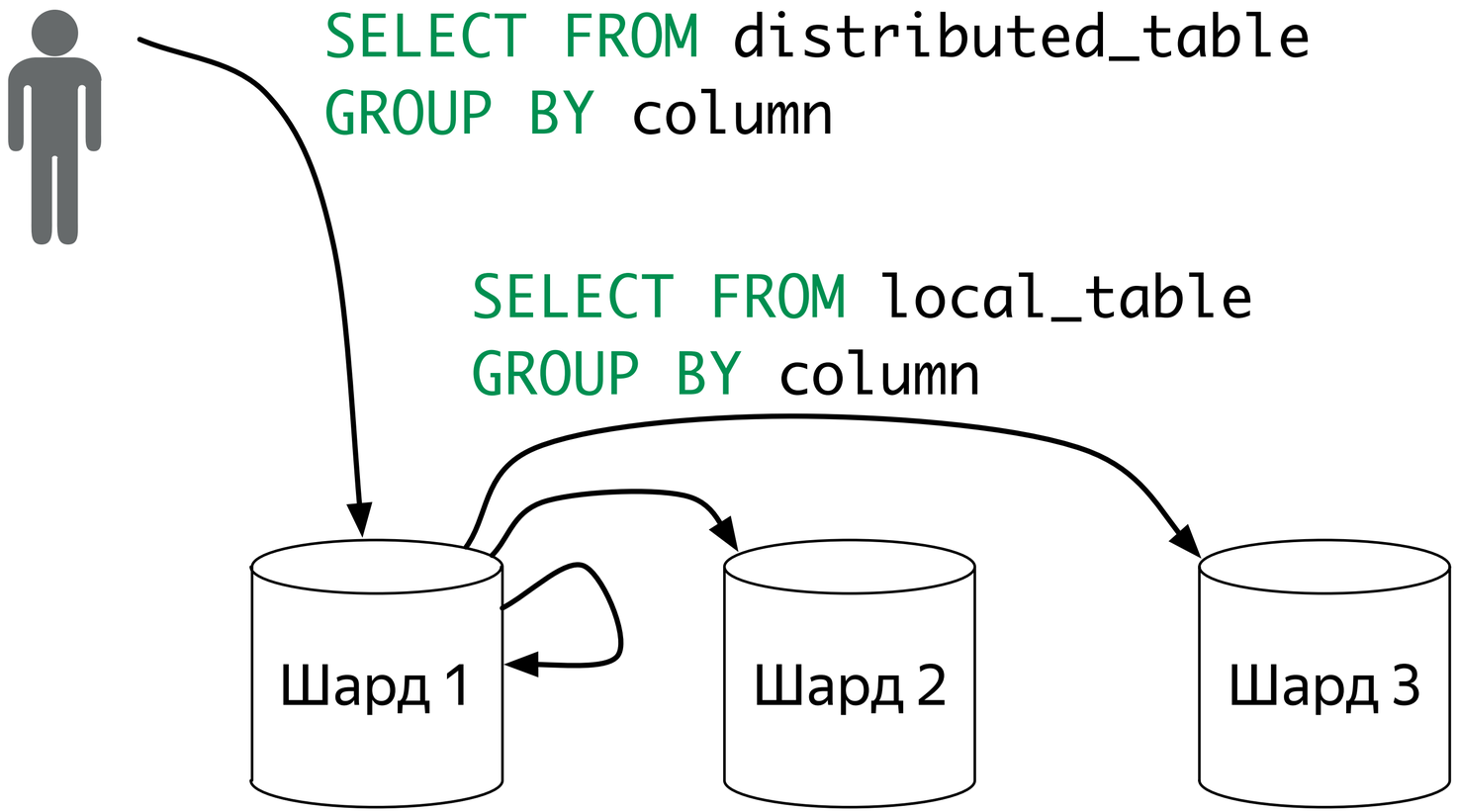
- Данные не помещаются на одну ноду...
- Хочется ещё ускорится, добавив железа...
- На сервер приходит слишком много одновременных запросов...

Когда одного сервера недостаточно

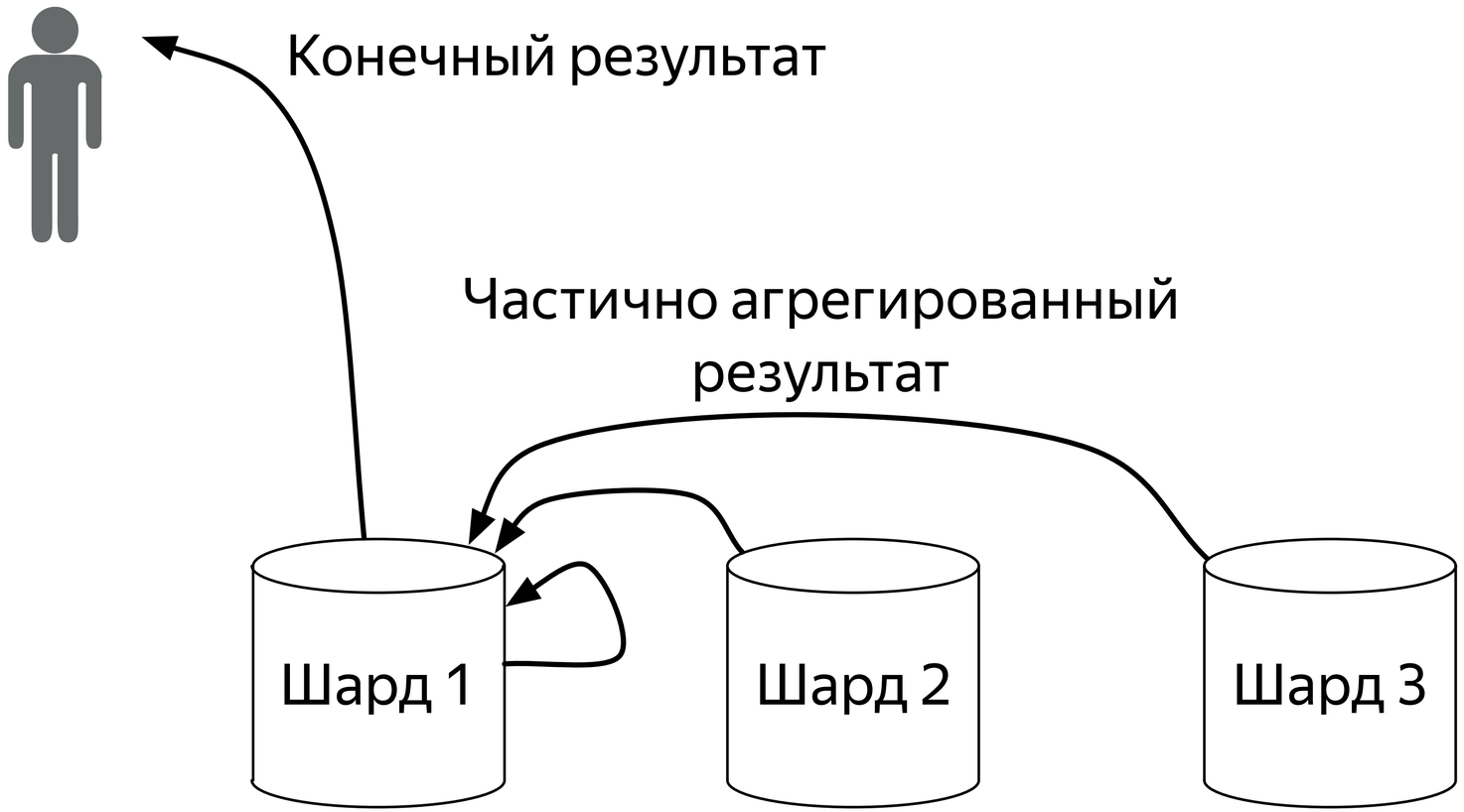
- Данные не помещаются на одну ноду...
- Хочется ещё ускорится, добавив железа...
- На сервер приходит слишком много одновременных запросов...

ClickHouse: Шардирование и Distributed таблицы!

Чтение из Distributed таблицы



Чтение из Distributed таблицы



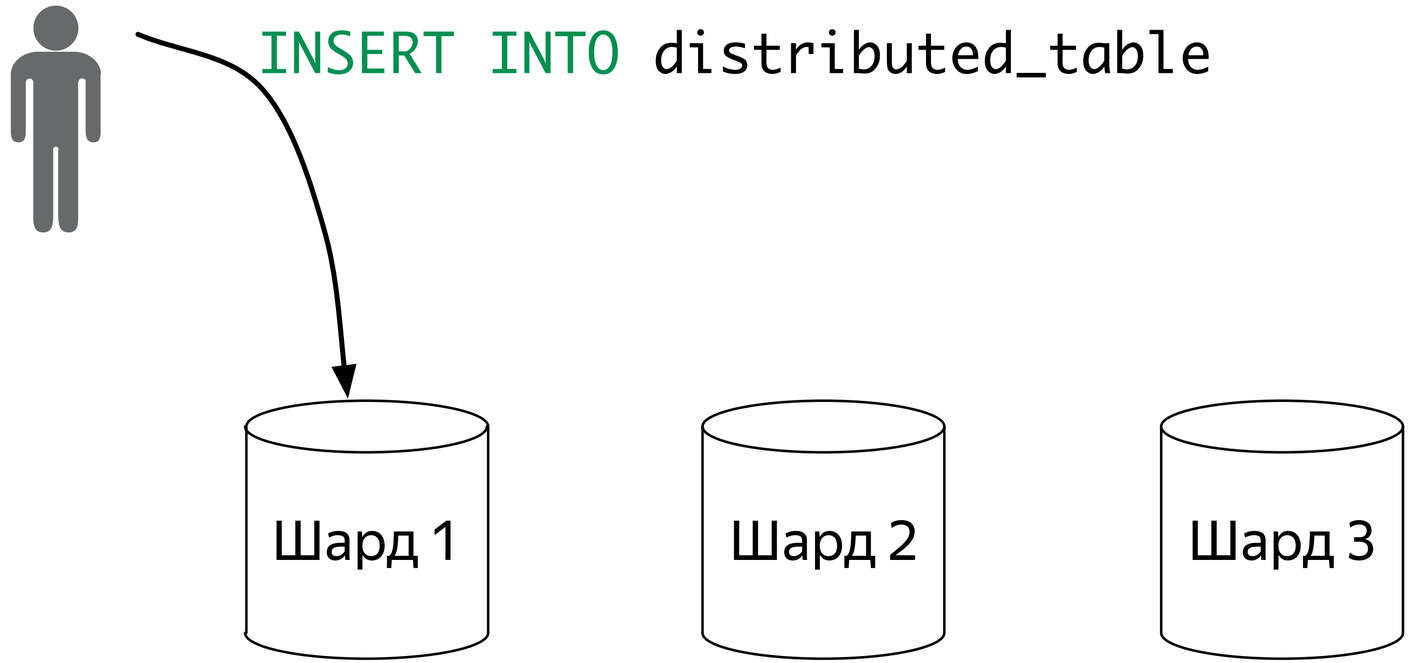
NYC taxi benchmark

CSV 227 Gb, ~1.3 млрд строк

```
SELECT passenger_count, avg(total_amount)
FROM trips GROUP BY passenger_count
```

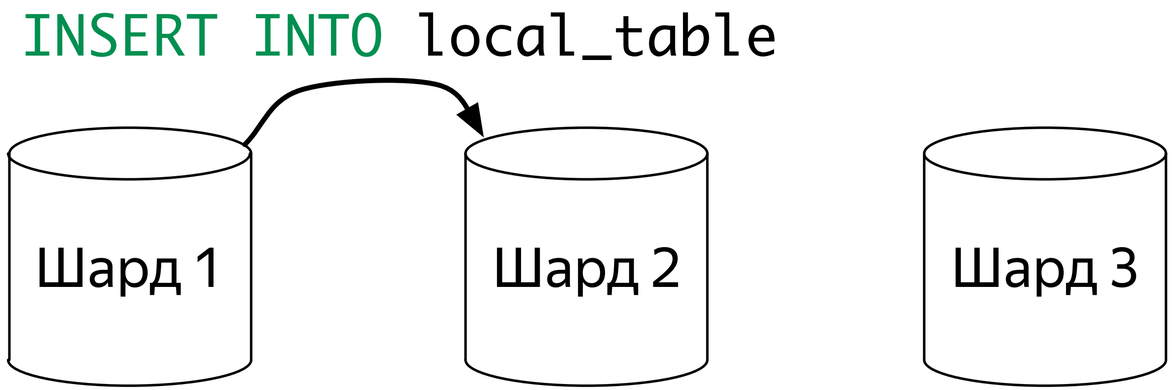
Шардов	1	3	140
Время, с.	1,224	0,438	0,043
Ускорение		x2.8	x28.5

Вставка в Distributed таблицу



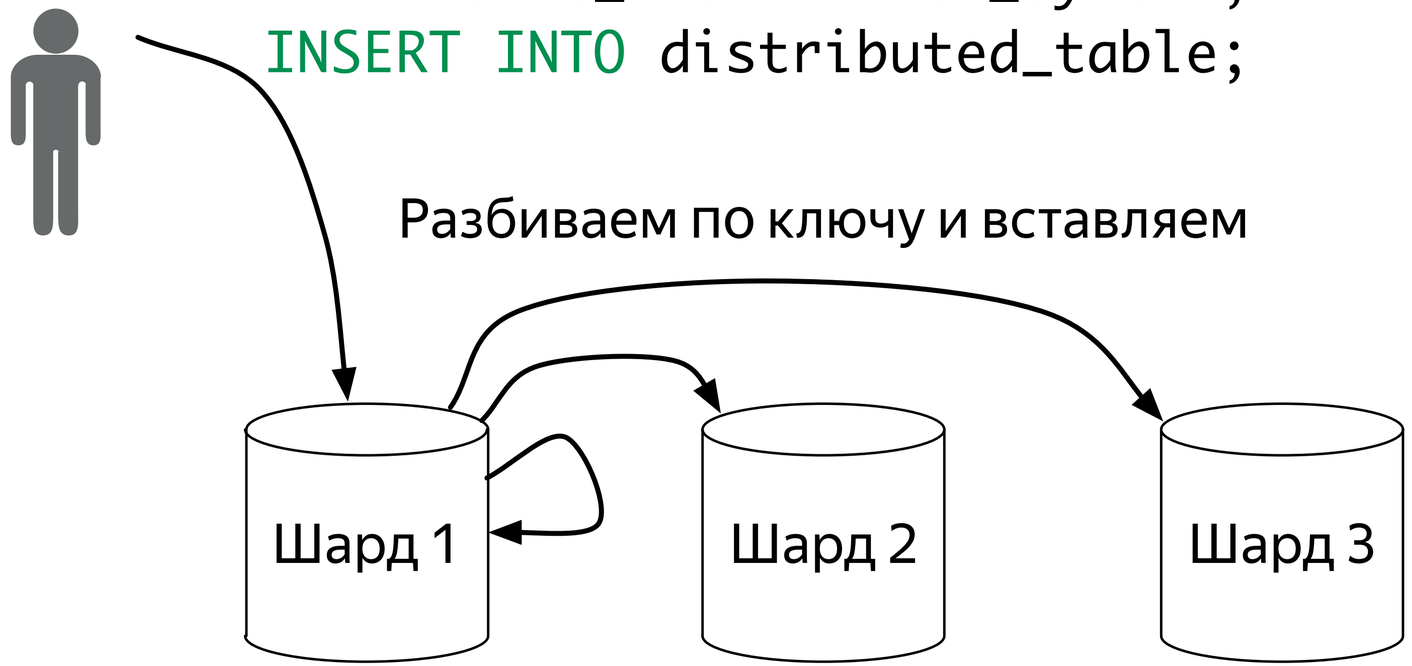
Вставка в Distributed таблицу

Асинхронно в шард №
`sharding_key % 3`



Вставка в Distributed таблицу

```
SET insert_distributed_sync=1;
INSERT INTO distributed_table;
```



Что нужно помнить про Distributed таблицы

Это просто view

- Сама по себе не хранит данных

Всегда запросит данные со всех шардов

Важно распределить данные по шардам равномерно

- вставляя напрямую в локальные таблицы
- или через вставку в Distributed таблицу
  (но помните про асинхронную вставку)

Если нельзя ломаться

- Защита от отказа оборудования
- Данные должны быть всегда доступны на чтение *и на* запись

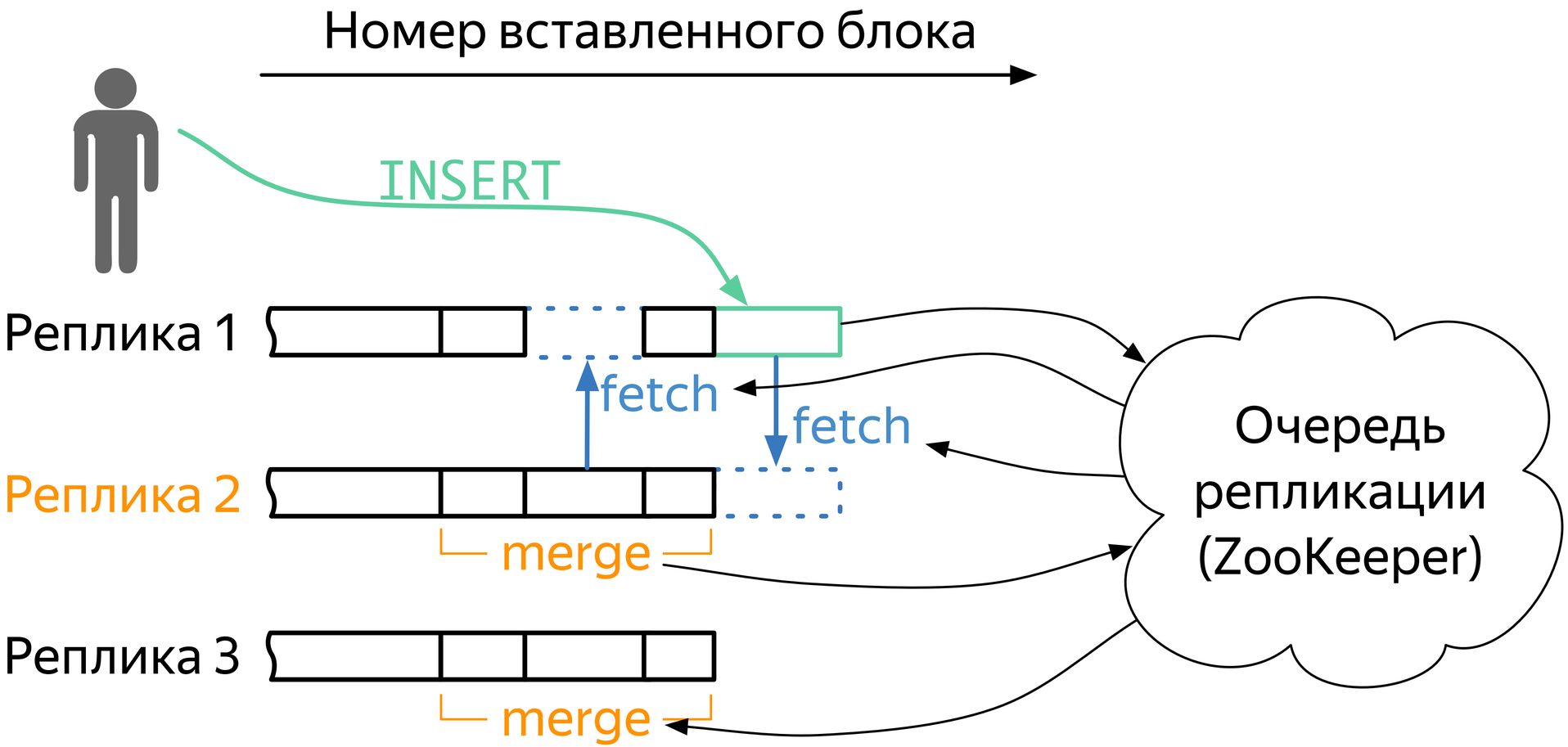
Если нельзя ломаться

- Защита от отказа оборудования
- Данные должны быть всегда доступны на чтение *и на* запись

ClickHouse: движок ReplicatedMergeTree!

- Асинхронная master-master репликация
- Работает на уровне таблицы

Внутреннее устройство репликации



Репликация и CAP–теорема

Что происходит в случае сетевого сбоя?

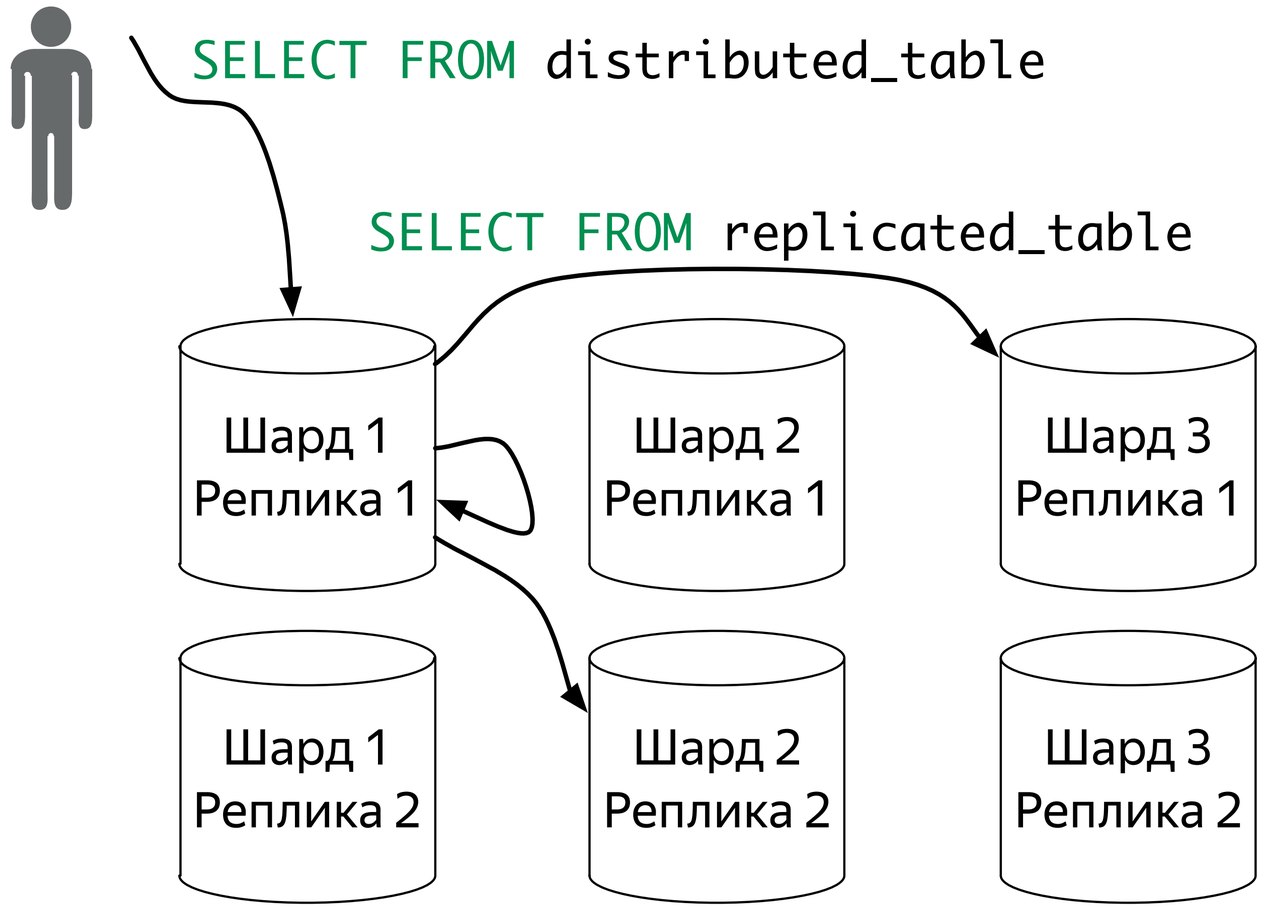
- Консистентности нет *
  Как и в любой системе с асинхронной репликацией

  *Но консистентность можно включить

- Высокая доступность (почти)*
  Выдерживает недоступность одного ДЦ, если реплики ClickHouse как минимум в 2 ДЦ и реплики ZK в 3 ДЦ.

  *На ноду, отрезанную от кворума ZK, невозможно записать данные

Всё вместе


SELECT FROM distributed_table
SELECT FROM replicated_table
Шард 1
Реплика 1
Шард 2
Реплика 1
Шард 3
Реплика 1
Шард 1
Реплика 2
Шард 2
Реплика 2
Шард 3
Реплика 2

Что надо помнить о репликации

Используйте!

- Реплики проверяют друг друга
- Не уверены, прошёл ли INSERT?
  Просто повторите - вставляемые блоки дедуплицируются
- Нужен ZooKeeper, но только для вставок
  (Для SELECT-ов дополнительной задержки нет)

Мониторьте отставание репликации

- Системные таблицы `system.replicas` и `system.replication_queue`

Ещё раз коротко

- Столбцовая СУБД
- Быстрые интерактивные запросы на данных, обновляемых в реальном времени
- Диалект SQL + расширения
- Плохо подходит для OLTP, Key–Value, хранения больших блобов
- Линейная масштабируемость
- Отказоустойчив
- Open source!

Спасибо

Вопросы? Можно сюда:

- clickhouse-feedback@yandex-team.ru
- **Telegram:** https://t.me/clickhouse_ru
- **GitHub:** https://github.com/yandex/ClickHouse/
- **Google group:** https://groups.google.com/group/clickhouse