

Дедупликация 5 миллионов событий в секунду на YDB в AppMetrica

Артем Исмагилов, Яндекс



**Saint
HighLoad⁺⁺**



О чем доклад

- Как мы построили сервис дедупликации на 5М ключей в секунду

О чем доклад

- Как мы построили сервис дедупликации на 5М ключей в секунду
- Как уменьшили потребление CPU в 10 раз с помощью нестандартного фильтра Блума



О чем доклад

- Как мы построили сервис дедубликации на 5М ключей в секунду
- Как уменьшили потребление CPU в 10 раз с помощью нестандартного фильтра Блума
- Как организовали ротацию данных и перешардирование сервиса




Контекст: AppMetrica

сервис аналитики мобильных приложений

Аудитория

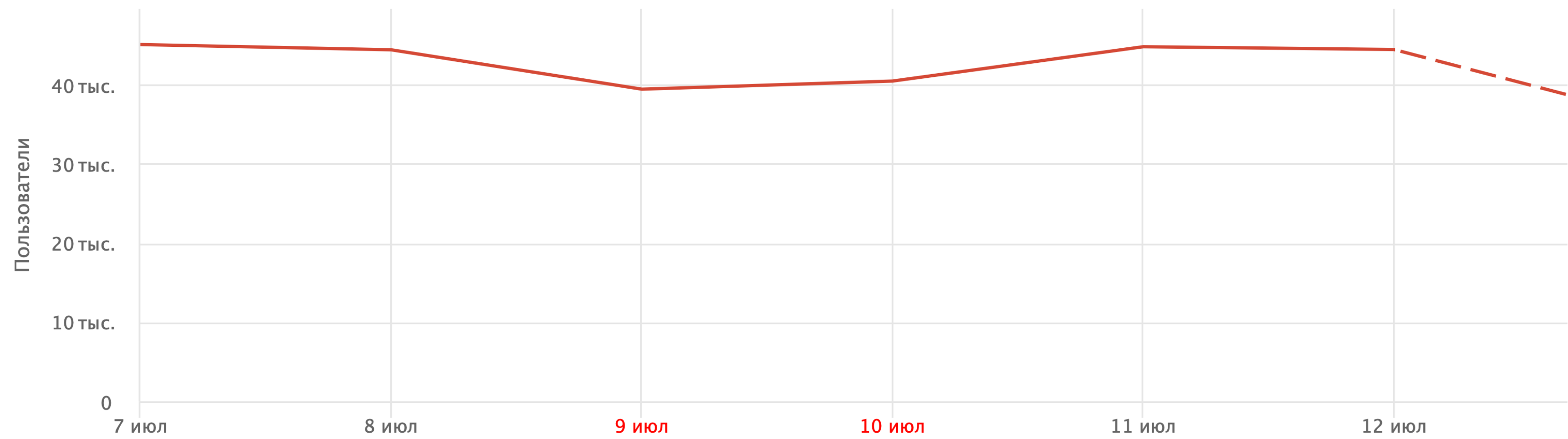
 100,00% данных 

Сегодня Вчера **Неделя** Две недели Месяц  7 — 13 июля 2022 Группировать по дням 

Сегментировать по пользователям, у которых  по сессиям, в которых  Сохранённые сегменты 

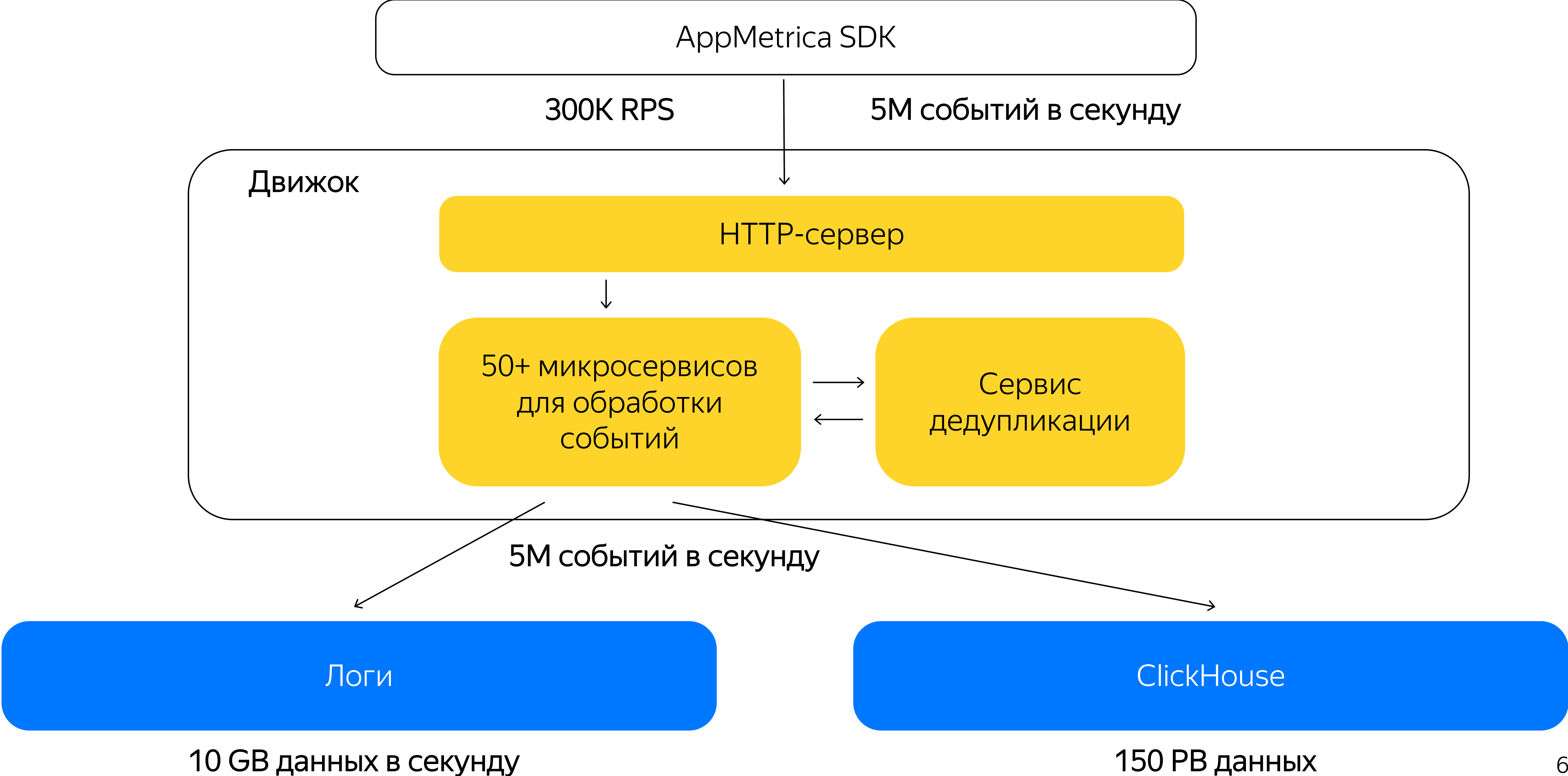


 Обновить отчёт  Экспорт 



Дата Страна Регион Город Пол Возраст

AppMetrica: общая схема

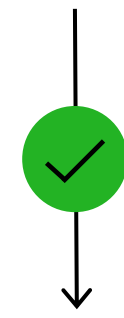


Откуда появляются дубли

События успешно
получены сервером

AppMetrica SDK

Events:
1, 2, 3...



HTTP-сервер для приема
событий

Откуда появляются дубли

События успешно
получены сервером



Events:
1, 2, 3... ✓



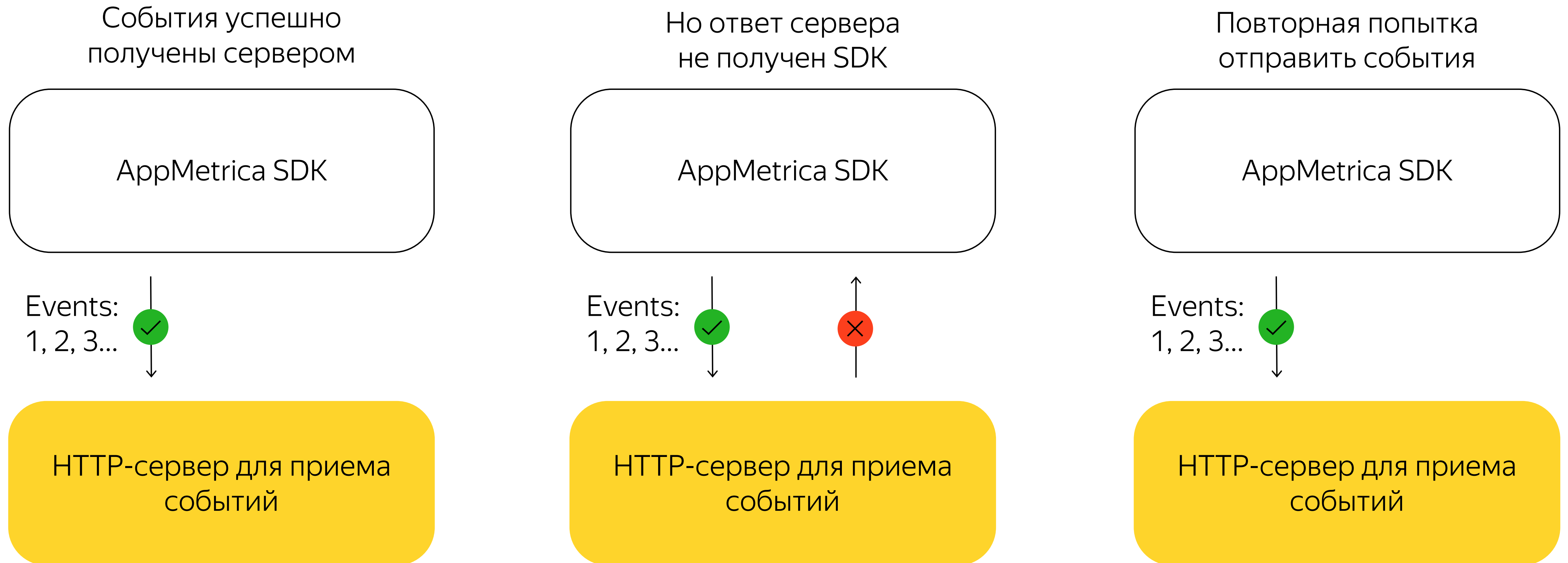
Но ответ сервера
не получен SDK



Events:
1, 2, 3... ✓ ✗



Откуда появляются дубли

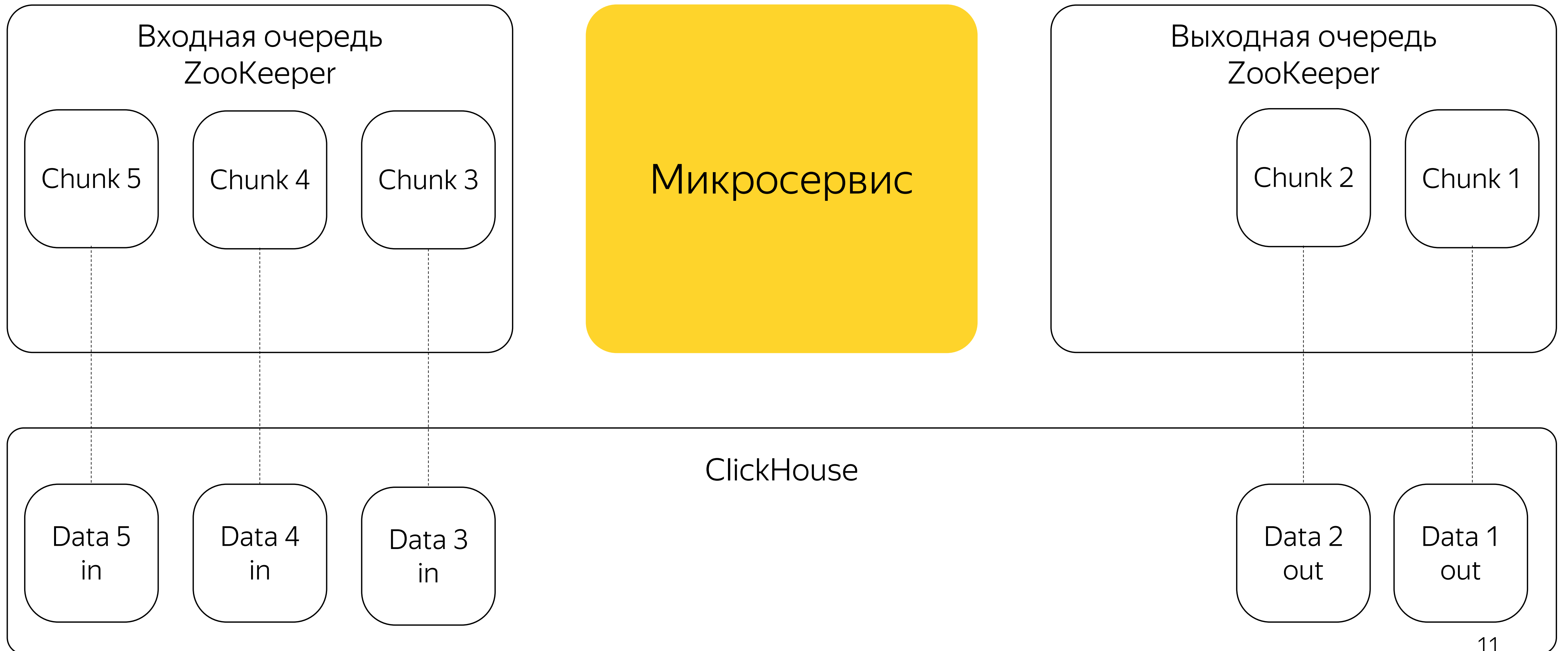


Такие события будут обработаны два раза

Зачем удалять дубли

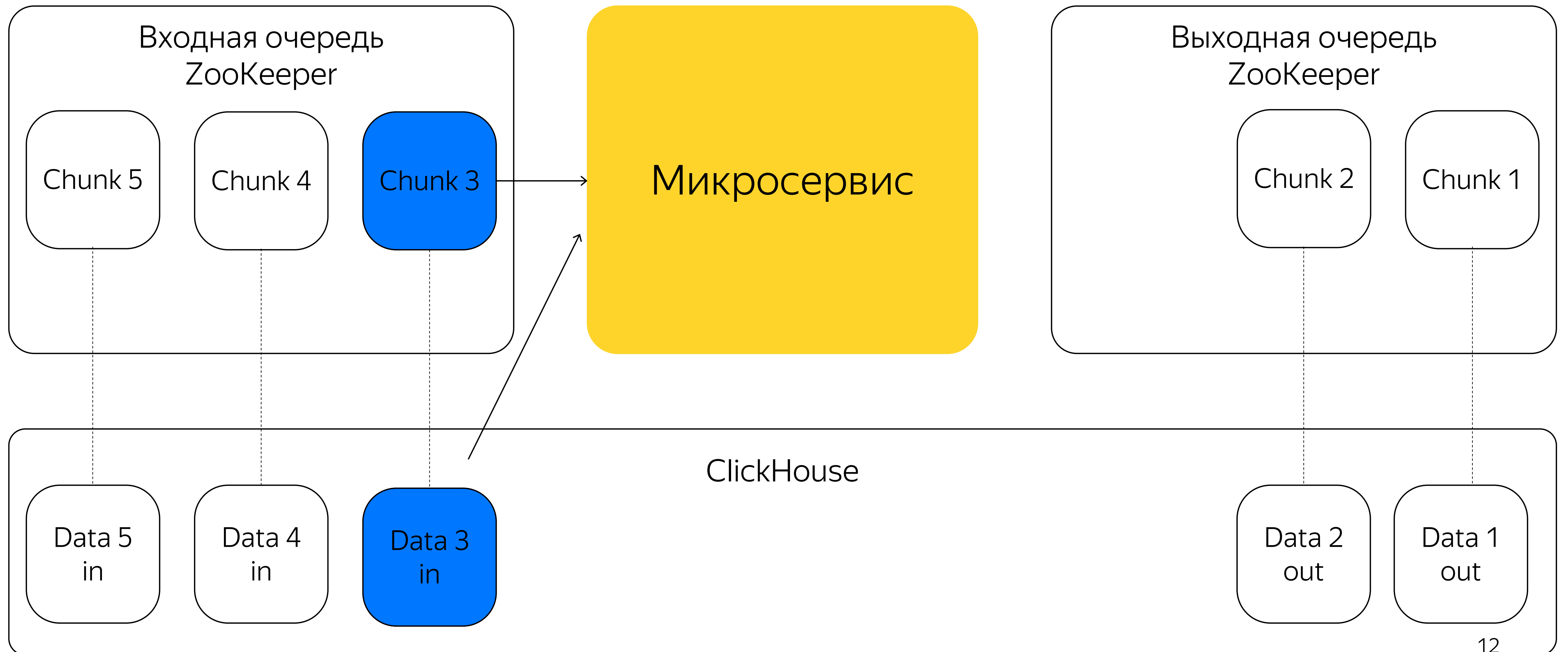
- Отчеты и данные в логах должны быть корректными
- Не хранить лишние 5 РВ данных, которые были бы дублями

Взаимодействие микросервисов движка



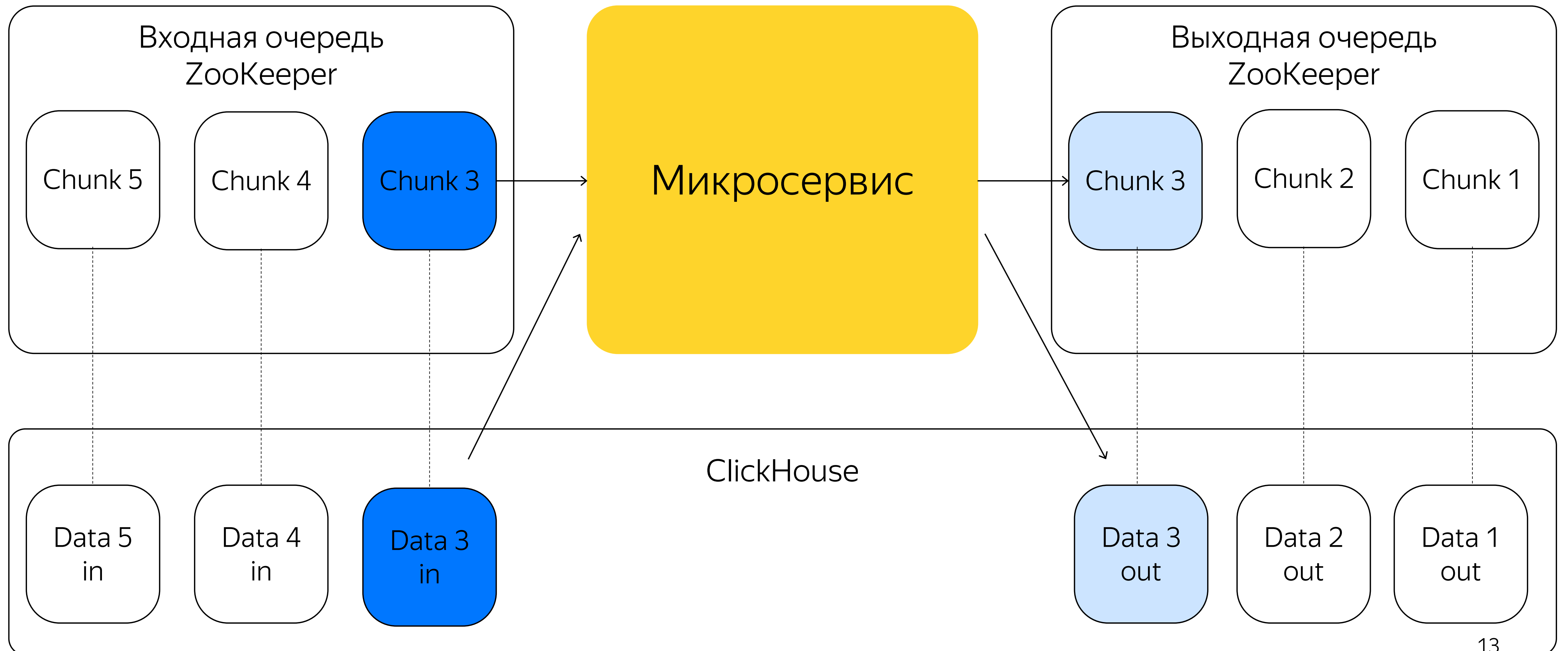
Взаимодействие микросервисов движка

Первый чанк из очереди читается и обрабатывается



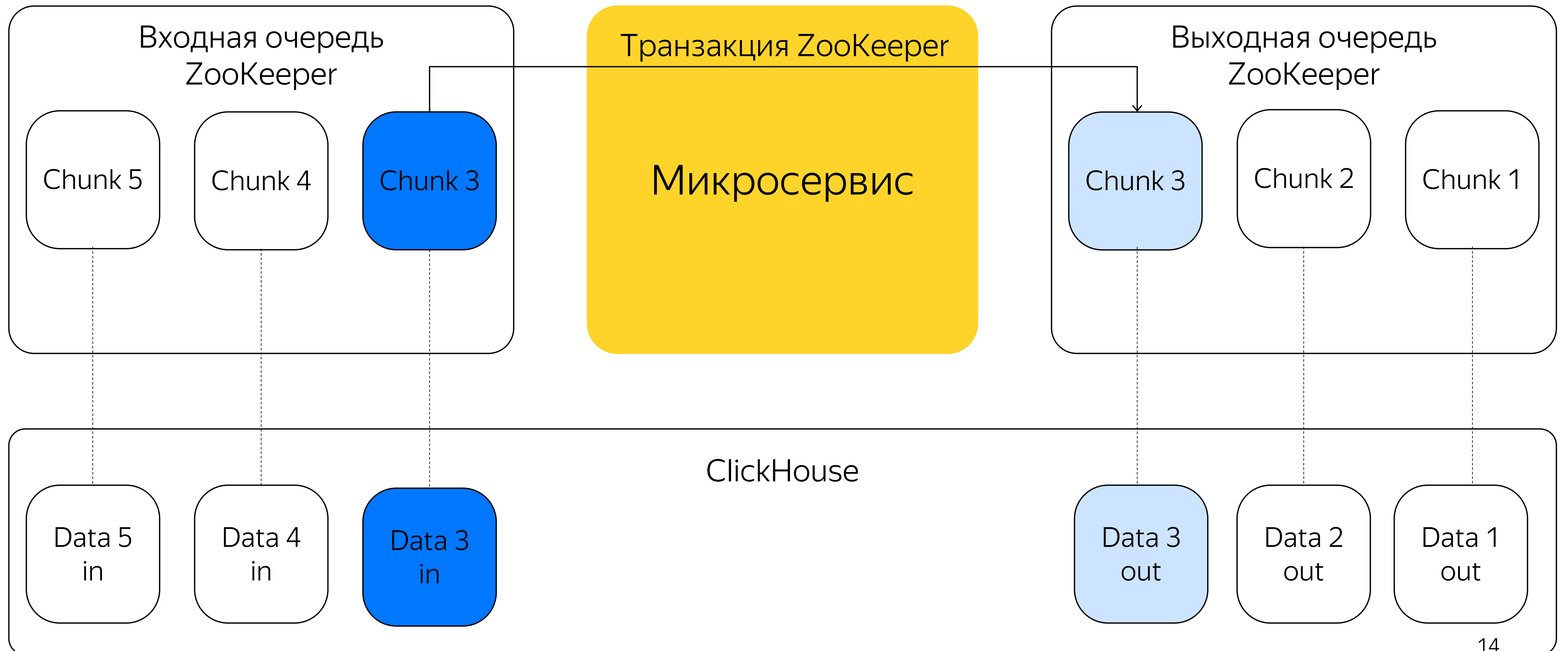
Взаимодействие микросервисов движка

Обработанные данные записываются в выходную таблицу



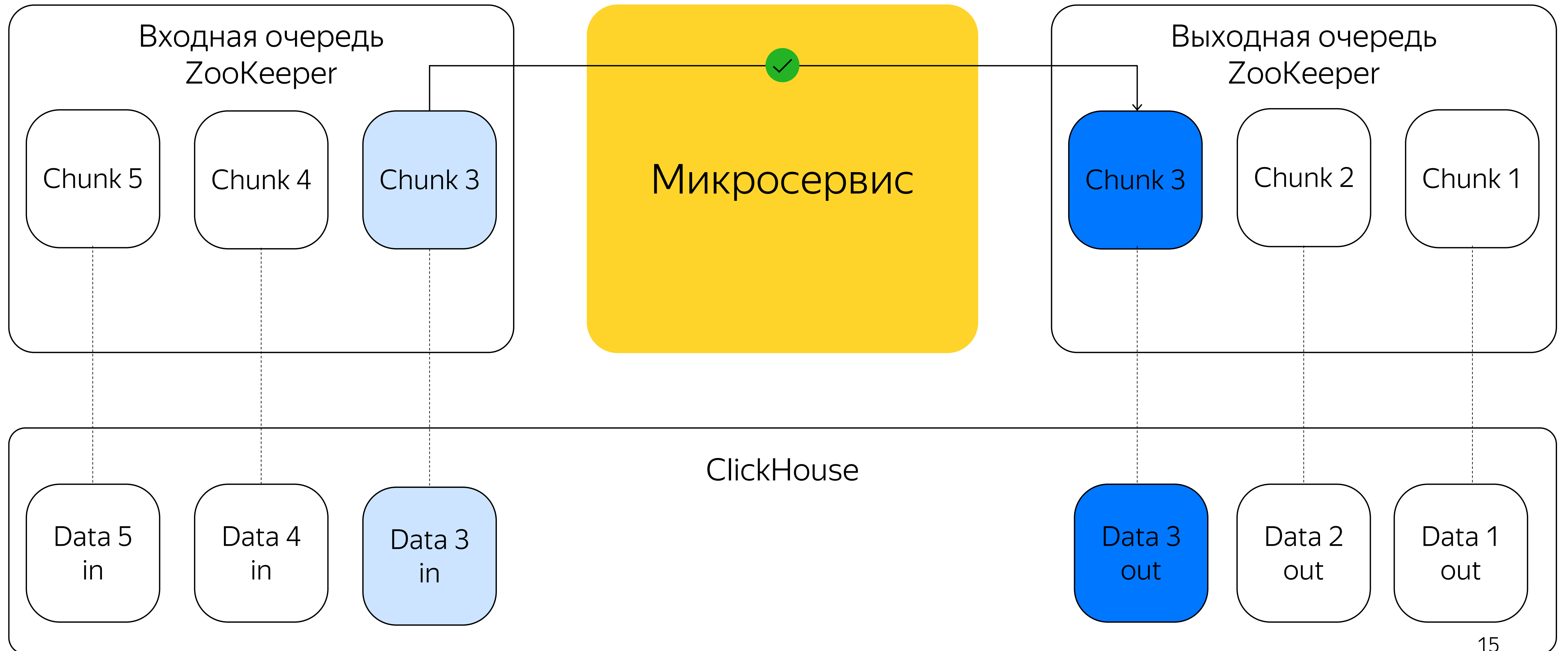
Взаимодействие микросервисов движка

Создается транзакция на удаление входного чанка и создание выходного



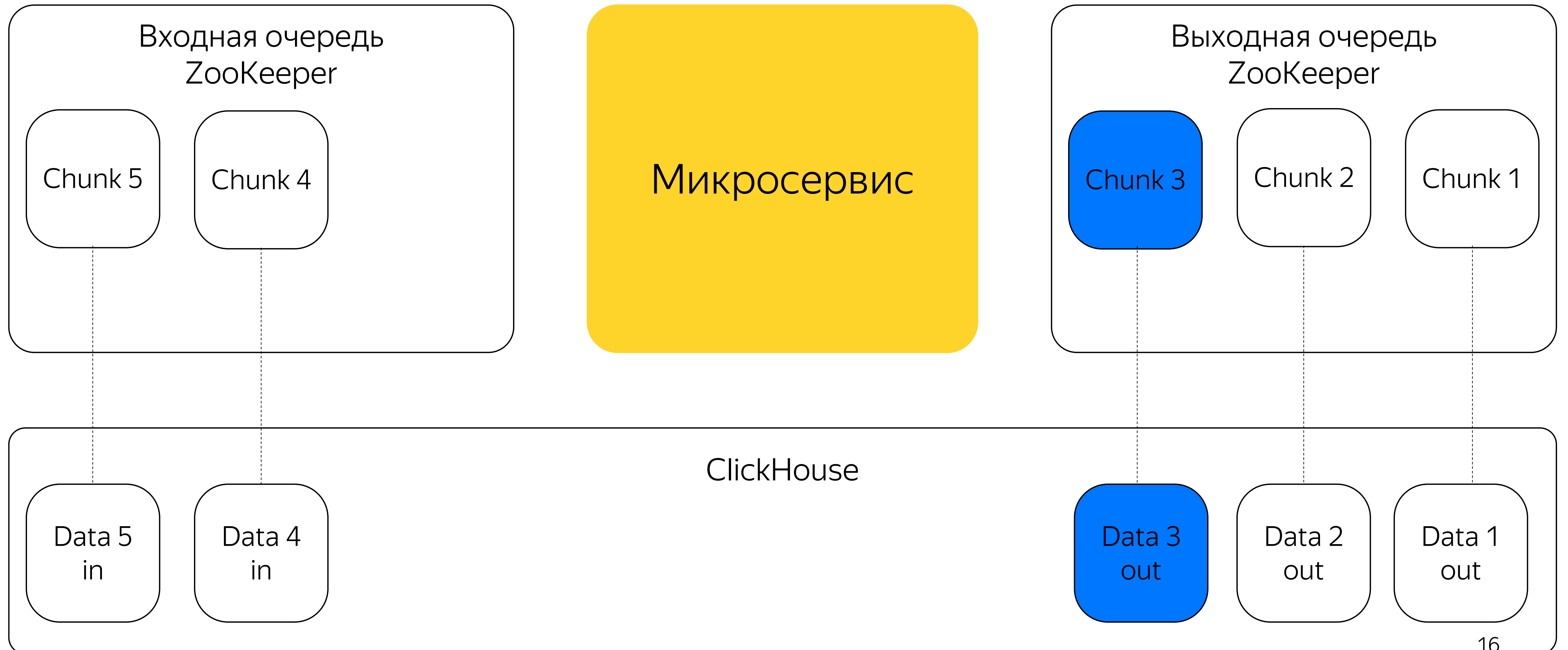
Взаимодействие микросервисов движка

Транзакция закоммичена



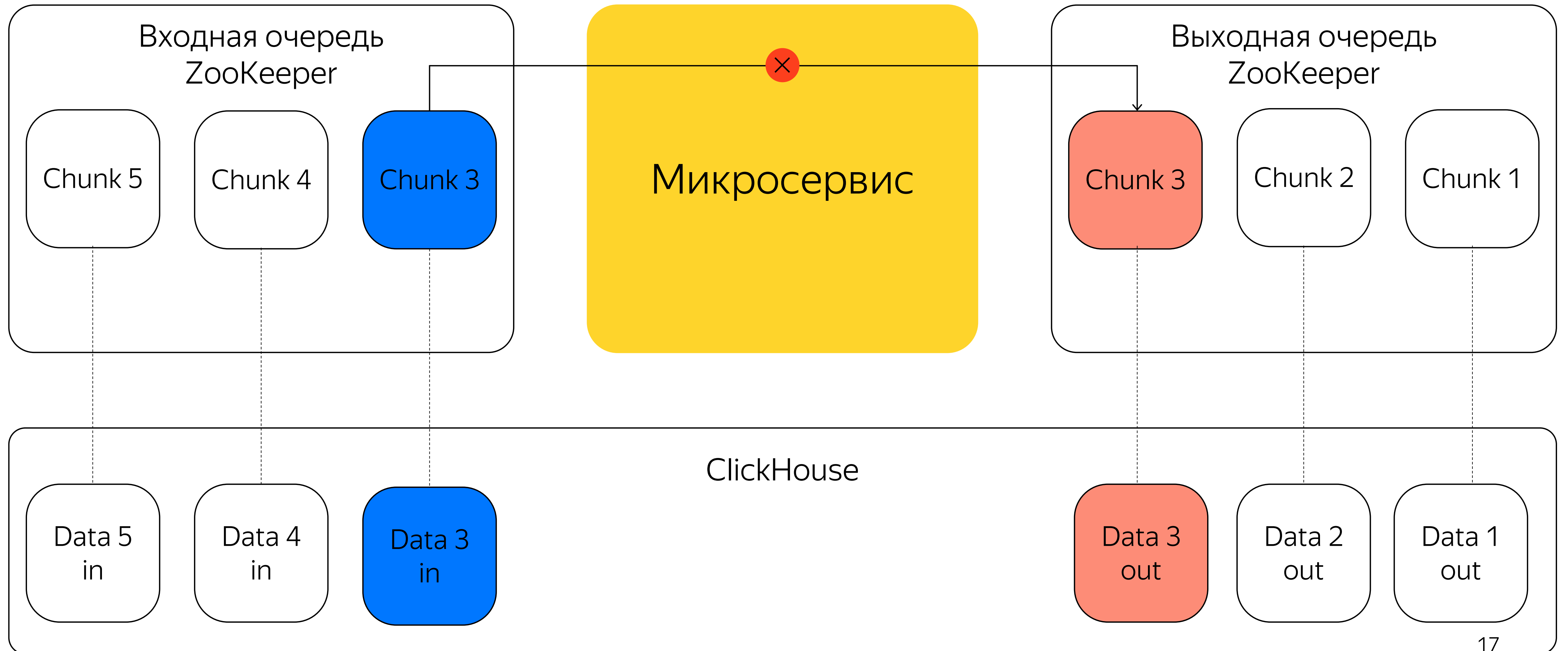
Взаимодействие микросервисов движка

Транзакция закоммичена



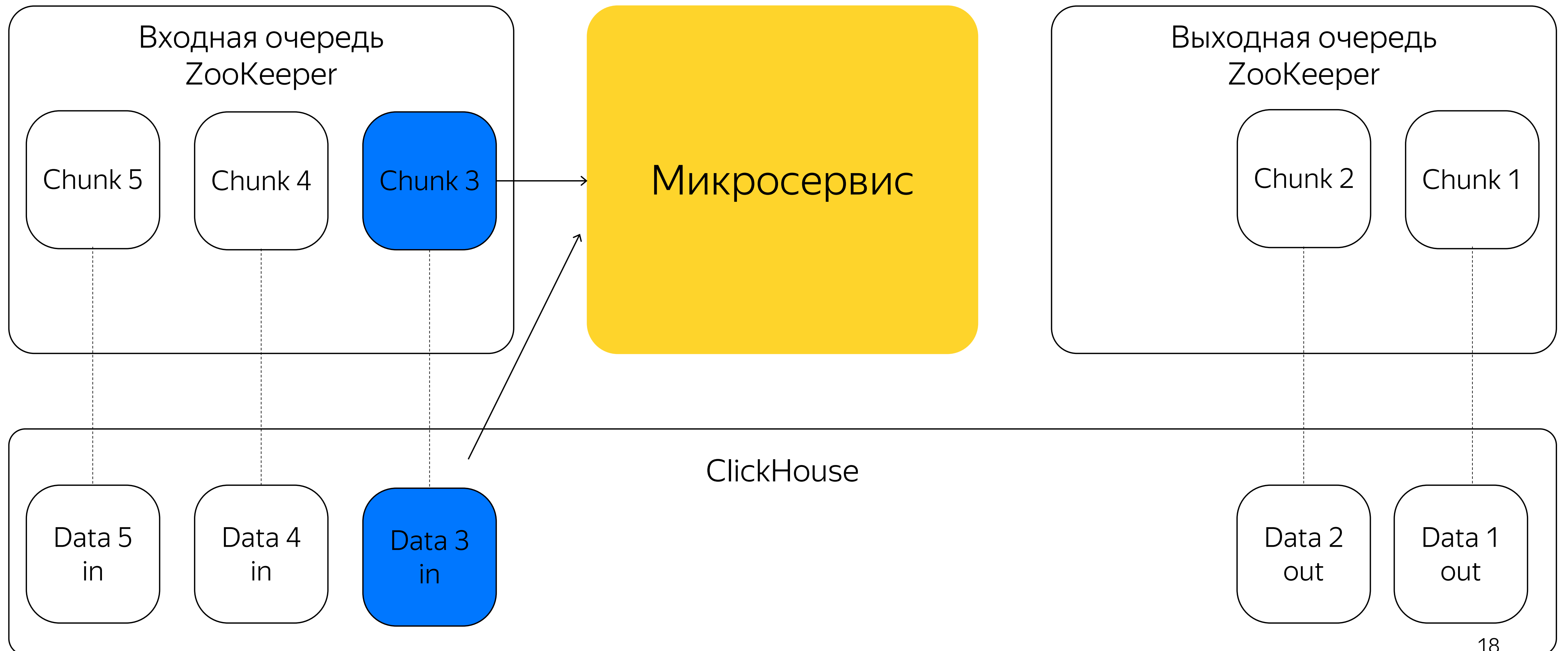
Взаимодействие микросервисов движка

... или не закоммичена

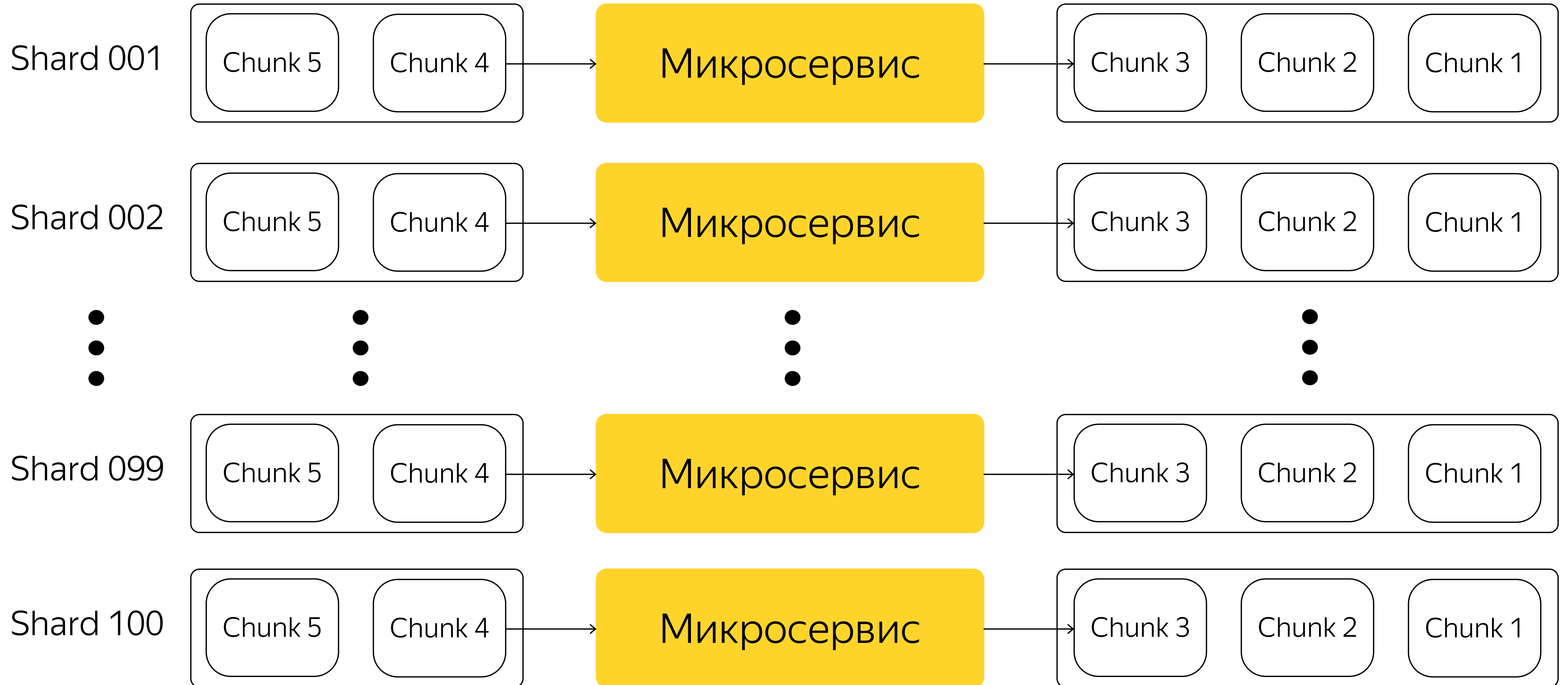


Взаимодействие микросервисов движка

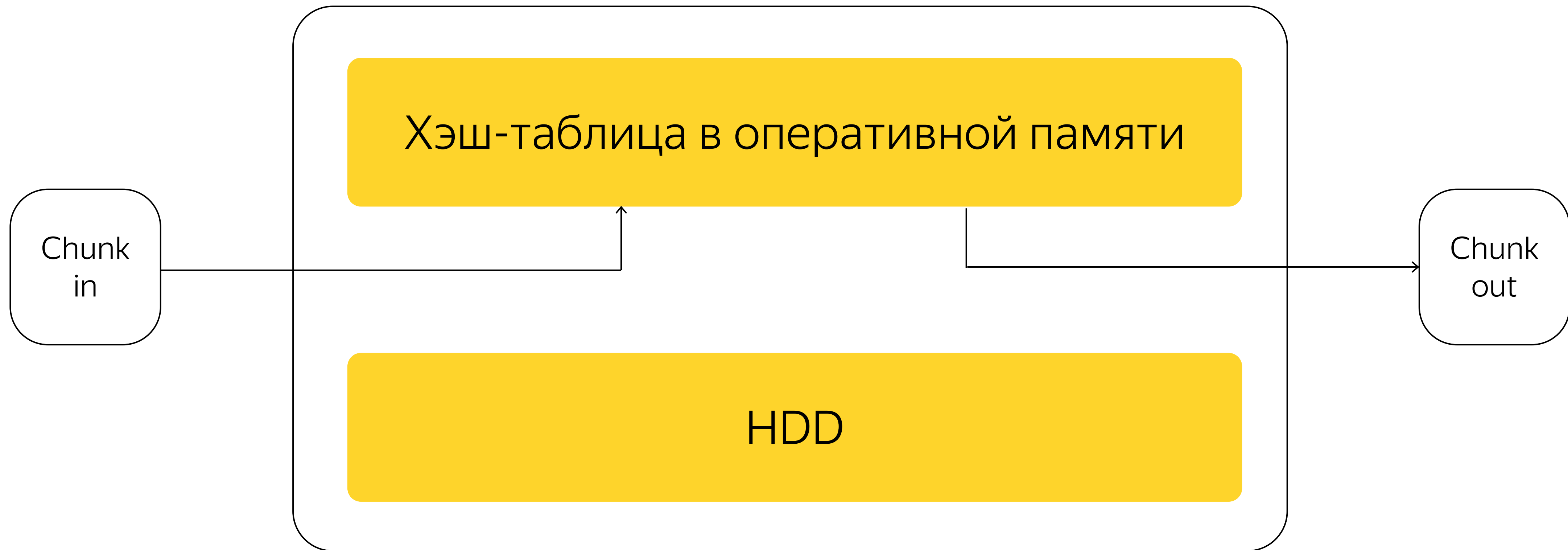
В случае ошибки данные будут обработаны повторно



Шардирование



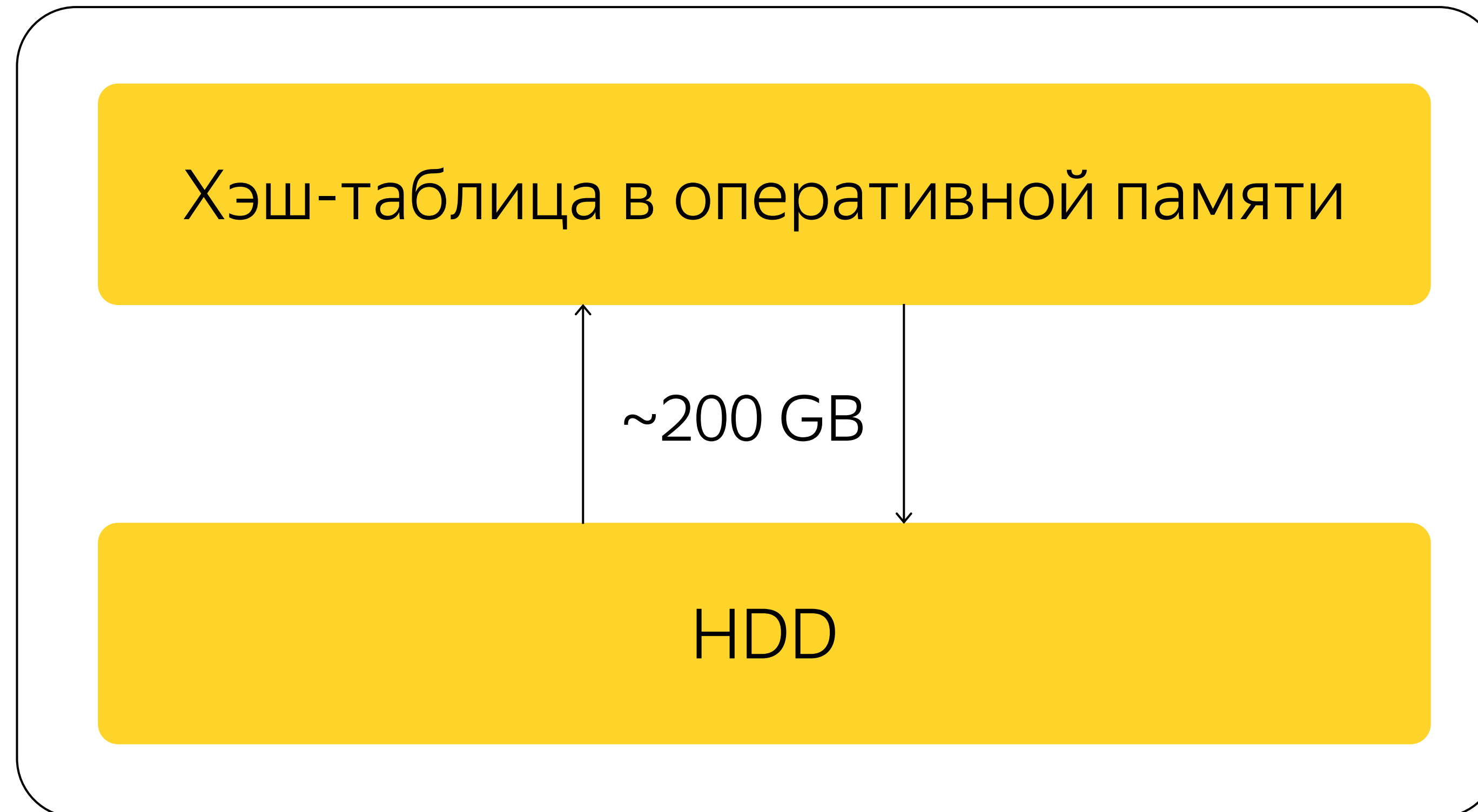
Предыдущая версия сервиса дедупликации



В случае ошибки обработки нельзя завершиться – потеряется состояние
Нужно откатить изменения и попробовать снова

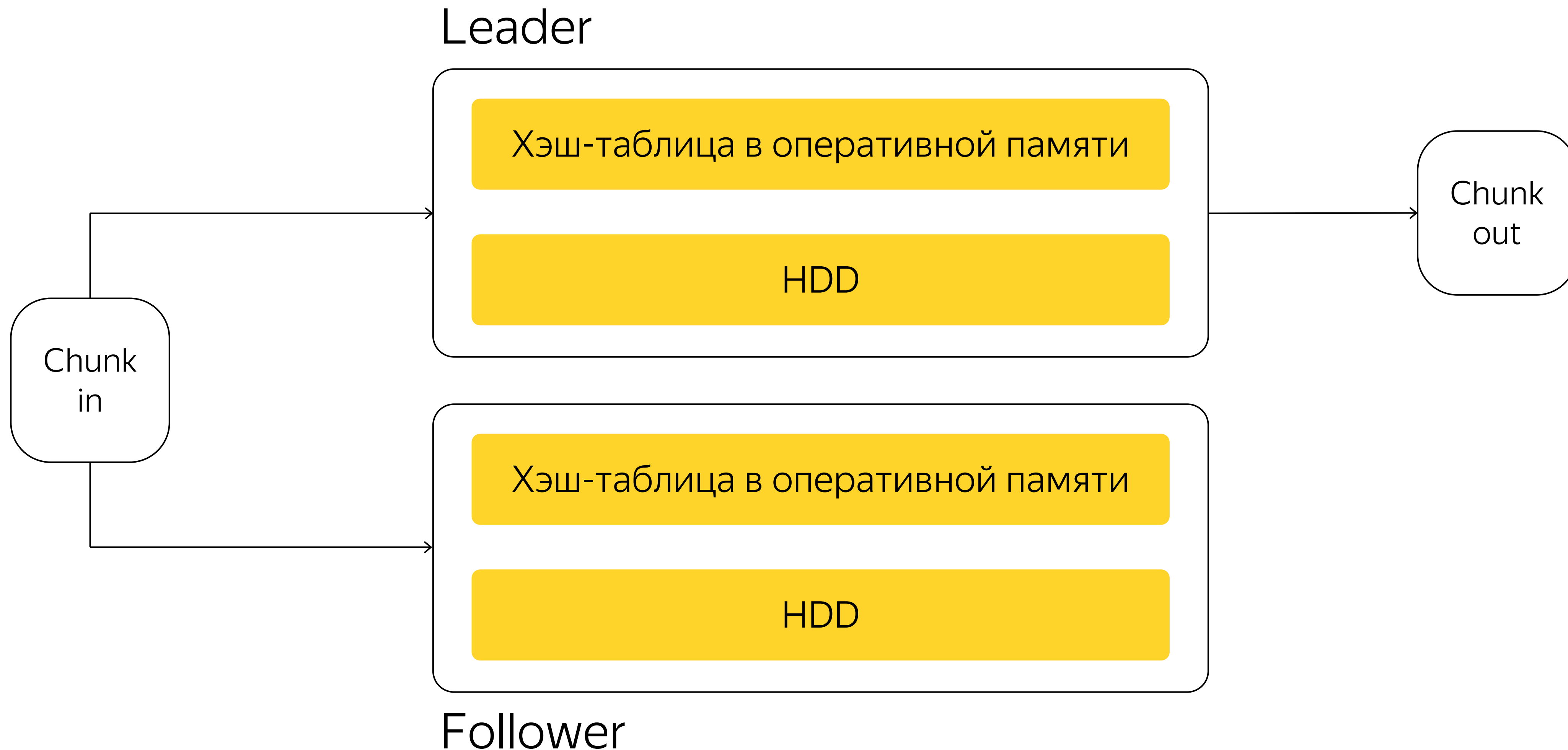
Предыдущая версия сервиса дедупликации

- Перед стартом состояние загружается с HDD
- Перед завершением работы состояние необходимо сохранить



Предыдущая версия сервиса дедупликации

Так как сервис может завершиться по внешним причинам, нужна репликация



Плюсы и минусы предыдущей версии

- ✔ Требуется мало ресурсов, так как вся обработка в оперативной памяти

Плюсы и минусы предыдущей версии

- ✔ Требуется мало ресурсов, так как вся обработка в оперативной памяти
- ✘ Легко может потерять состояние, были такие случаи

Плюсы и минусы предыдущей версии

- ✔ Требуется мало ресурсов, так как вся обработка в оперативной памяти
- ✘ Легко может потерять состояние, были такие случаи
- ✘ Невозможно увеличить размер состояния из-за ограничений по RAM

Плюсы и минусы предыдущей версии

- ✔ Требуется мало ресурсов, так как вся обработка в оперативной памяти
- ✘ Легко может потерять состояние, были такие случаи
- ✘ Невозможно увеличить размер состояния из-за ограничений по RAM
- ✘ Невозможно перешардировать без значительного даунтайма

Плюсы и минусы предыдущей версии

- ✔ Требуется мало ресурсов, так как вся обработка в оперативной памяти
- ✘ Легко может потерять состояние, были такие случаи
- ✘ Невозможно увеличить размер состояния из-за ограничений по RAM
- ✘ Невозможно перешардировать без значительного даунтайма
- ✘ Привязан к конкретным физическим машинам кластера

Плюсы и минусы предыдущей версии

- ✔ Требует мало ресурсов, так как вся обработка в оперативной памяти
- ✘ Легко может потерять состояние, были такие случаи
- ✘ Невозможно увеличить размер состояния из-за ограничений по RAM
- ✘ Невозможно перешардировать без значительного даунтайма
- ✘ Привязан к конкретным физическим машинам кластера
- ✘ Перезапуск занимает 50 минут из-за операций с HDD

Требования к новой версии

- ✔ Потребление не сильно большего количества ресурсов, чем старая версия

Требования к новой версии

- ✓ Потребление не сильно большего количества ресурсов, чем старая версия
- ✓ Гарантия сохранения состояния

Требования к новой версии

- ✔ Потребление не сильно большего количества ресурсов, чем старая версия
- ✔ Гарантия сохранения состояния
- ✔ Возможность легко менять окно дедупликации и размер состояния

Требования к новой версии

- ✓ Потребление не сильно большего количества ресурсов, чем старая версия
- ✓ Гарантия сохранения состояния
- ✓ Возможность легко менять окно дедупликации и размер состояния
- ✓ Возможность легко увеличить количество шардов

Требования к новой версии

- ✔ Потребление не сильно большего количества ресурсов, чем старая версия
- ✔ Гарантия сохранения состояния
- ✔ Возможность легко менять окно дедупликации и размер состояния
- ✔ Возможность легко увеличить количество шардов
- ✔ Легко развернуть в облаке, не нужен кластер физических машин

Прямой подход к задаче дедупликации

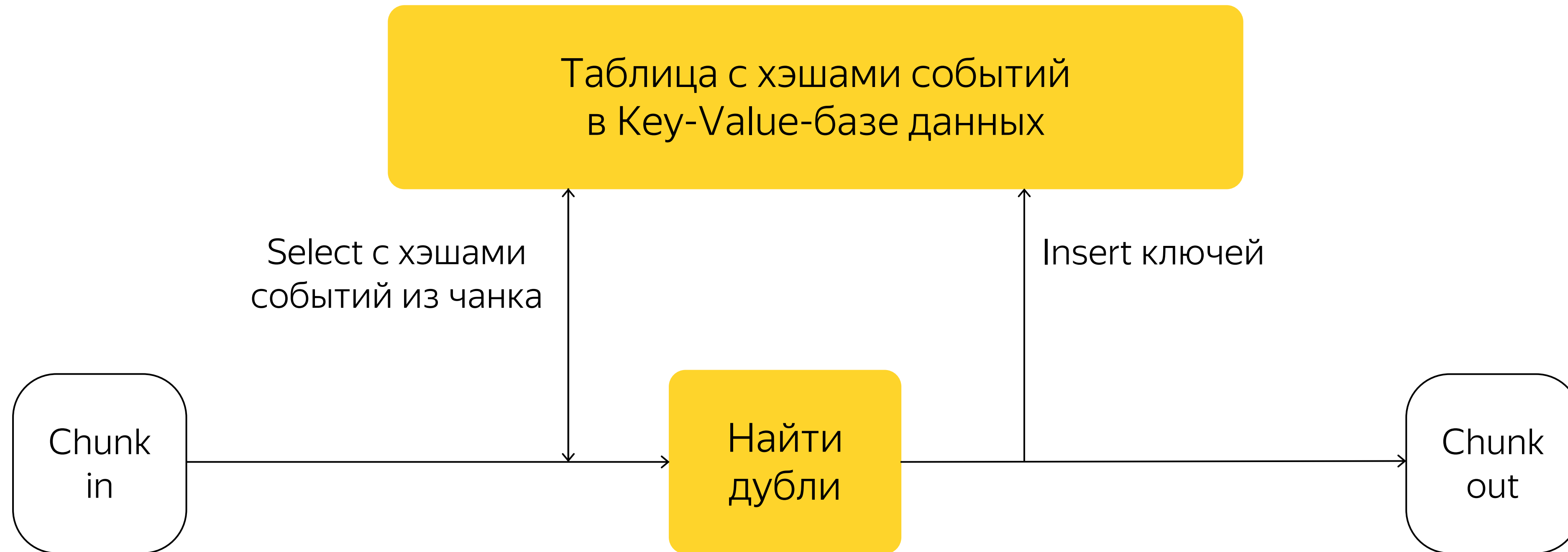


Схема таблицы – (DeviceIDHash, EventHash, InsertTime)

TTL по InsertTime

YDB

- ✓ Линейное горизонтальное масштабирование

YDB

- ✓ Линейное горизонтальное масштабирование
- ✓ Геораспределенная, zero downtime

YDB

- ✓ Линейное горизонтальное масштабирование
- ✓ Геораспределенная, zero downtime
- ✓ Key-Value-хранилище

YDB

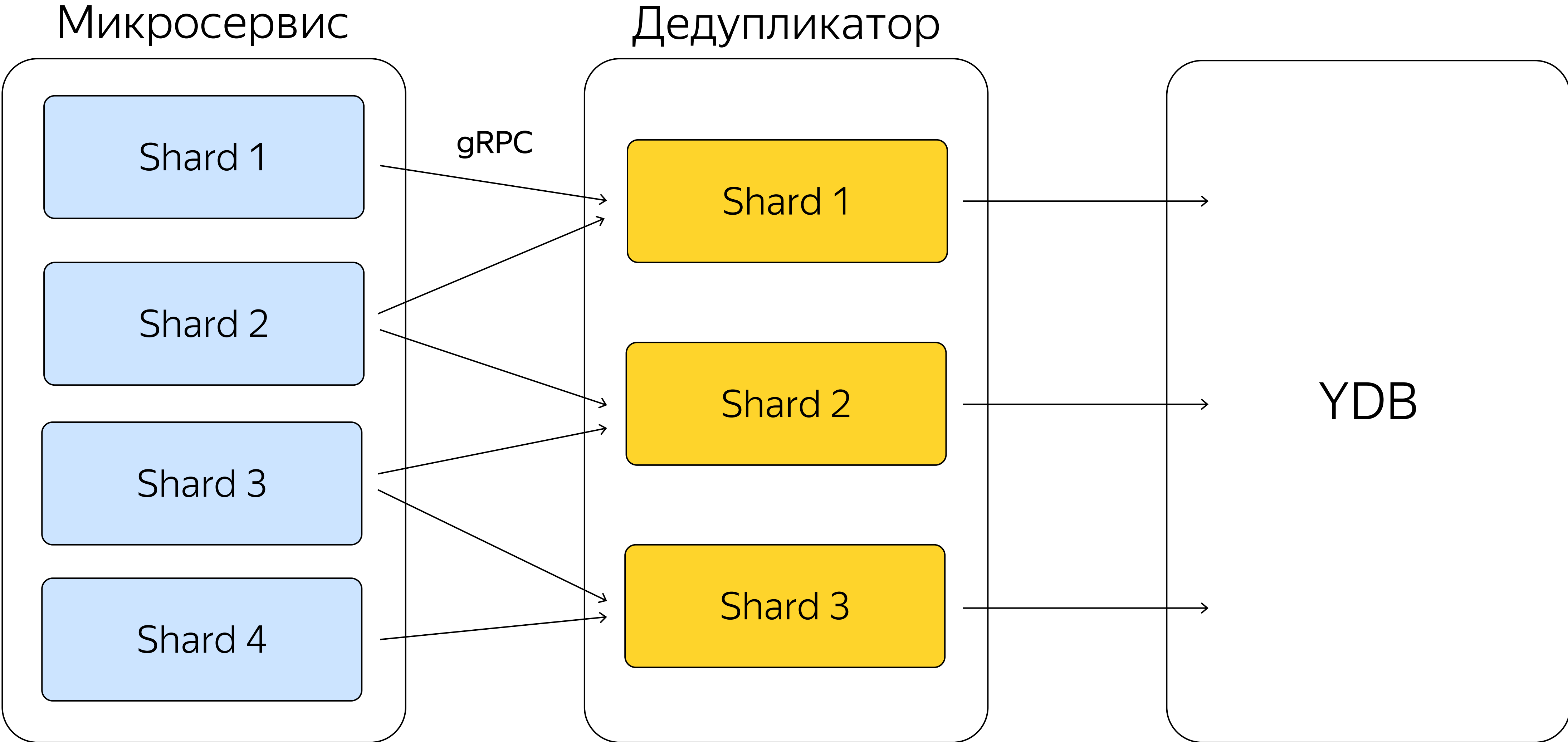
- ✓ Линейное горизонтальное масштабирование
- ✓ Геораспределенная, zero downtime
- ✓ Key-Value-хранилище
- ✓ Записи уникальны по первичному ключу, операция Upsert

YDB

- ✓ Линейное горизонтальное масштабирование
- ✓ Геораспределенная, zero downtime
- ✓ Key-Value-хранилище
- ✓ Записи уникальны по первичному ключу, операция Upsert
- ✓ Эффективная вставка операцией BulkUpsert, без транзакций

Схема работы сервиса дедупликации

Дедупликатор as a service

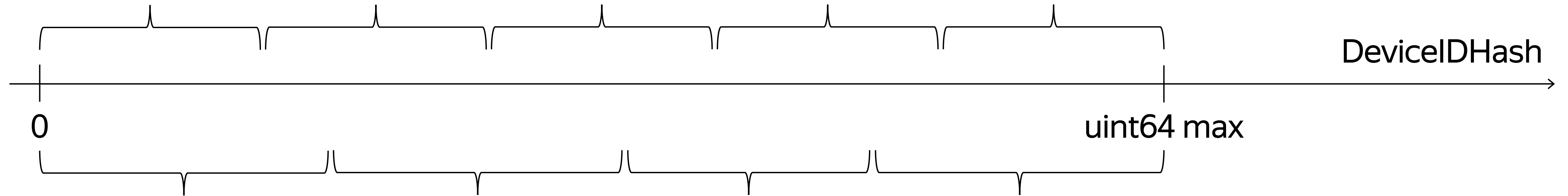


Шардирование

Схема таблицы – (DeviceIDHash, EventHash, InsertTime)

└──┘
Первичный ключ

Партиции YDB

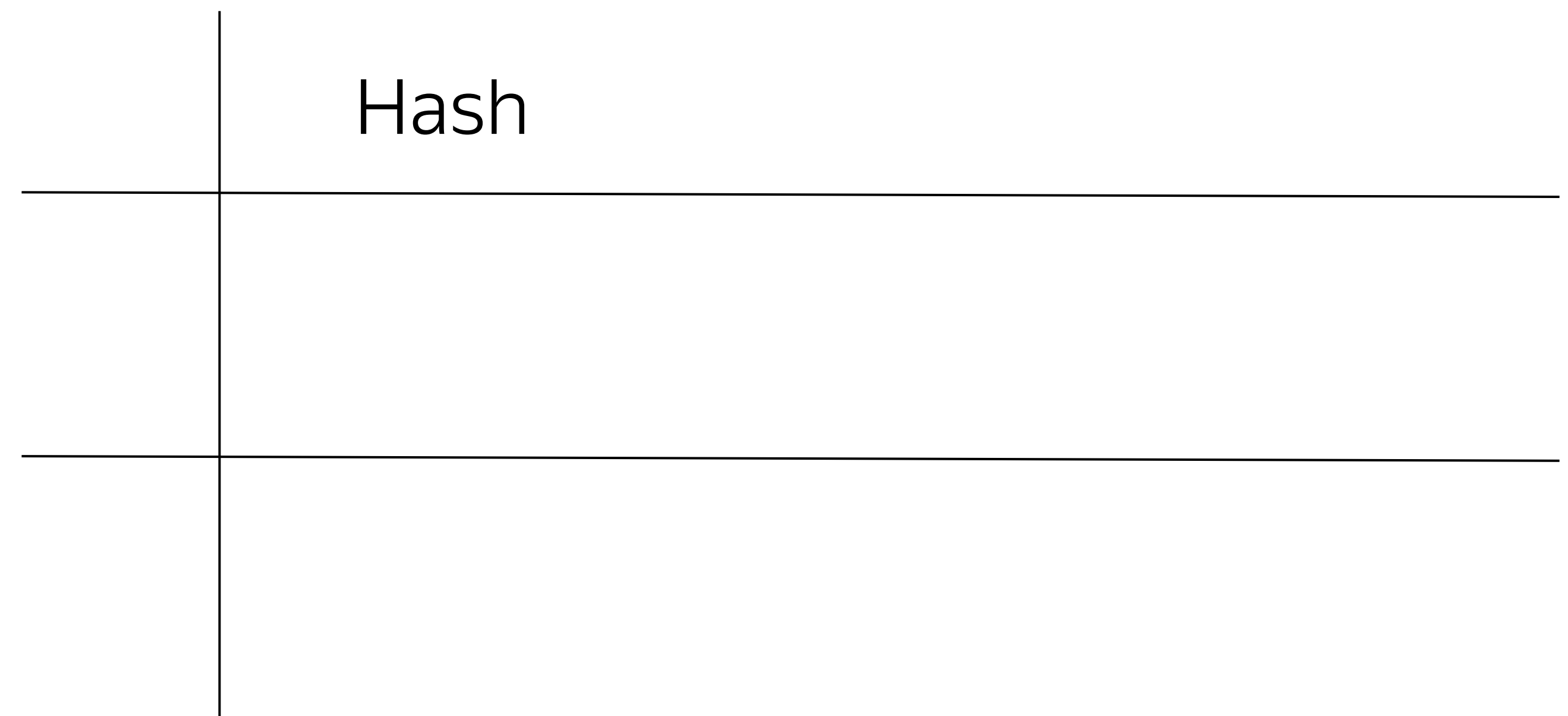
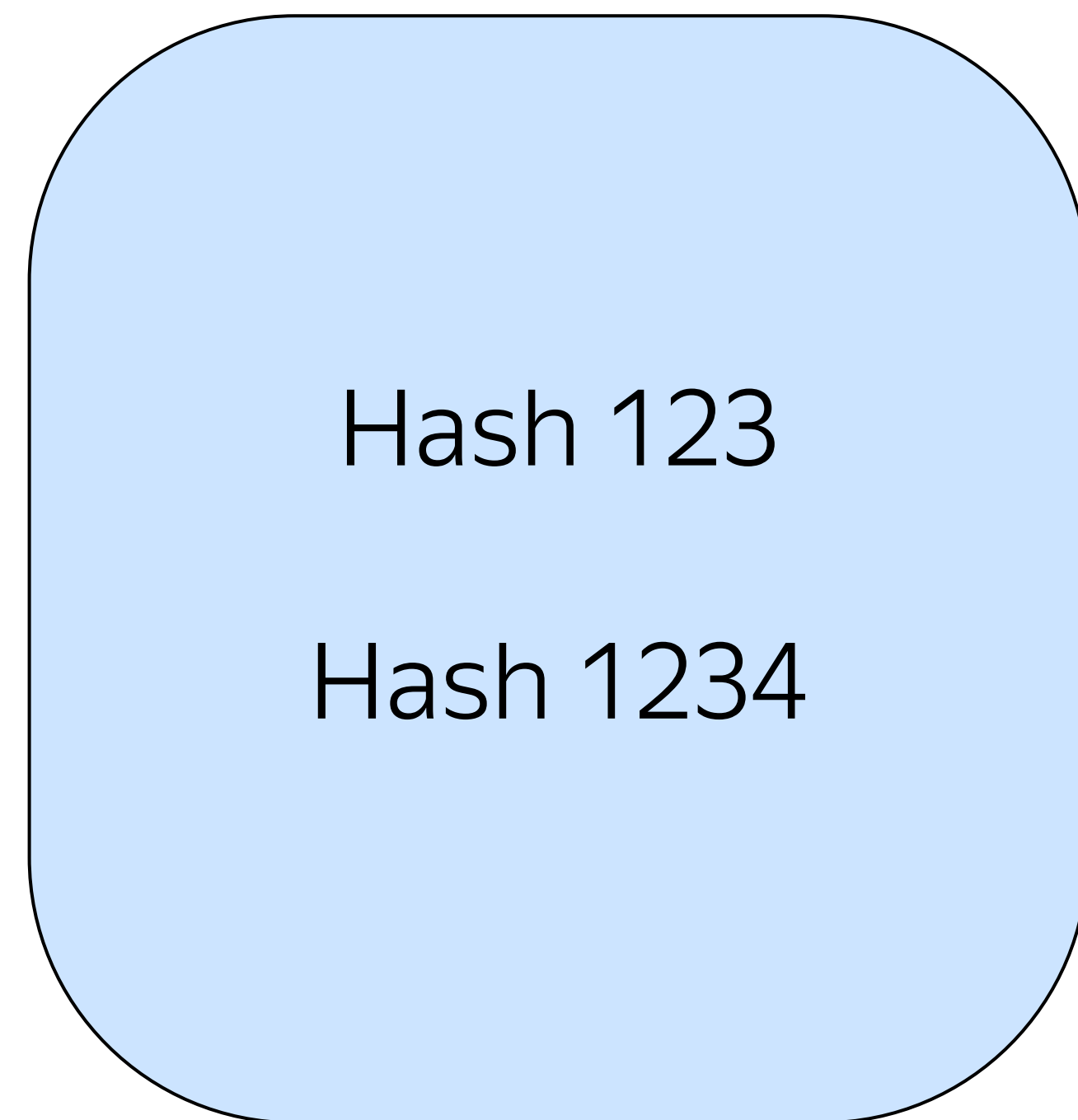


Шарды сервиса дедупликации

Транзакционная обработка данных

Chunk

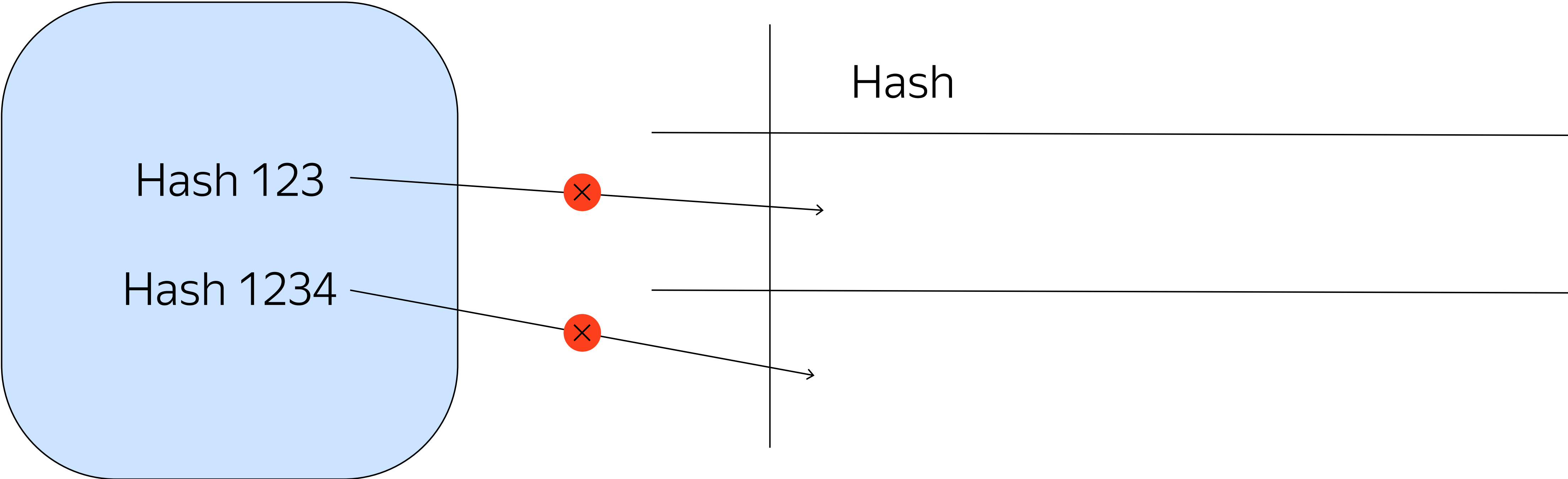
YDB



Транзакционная обработка данных

Chunk

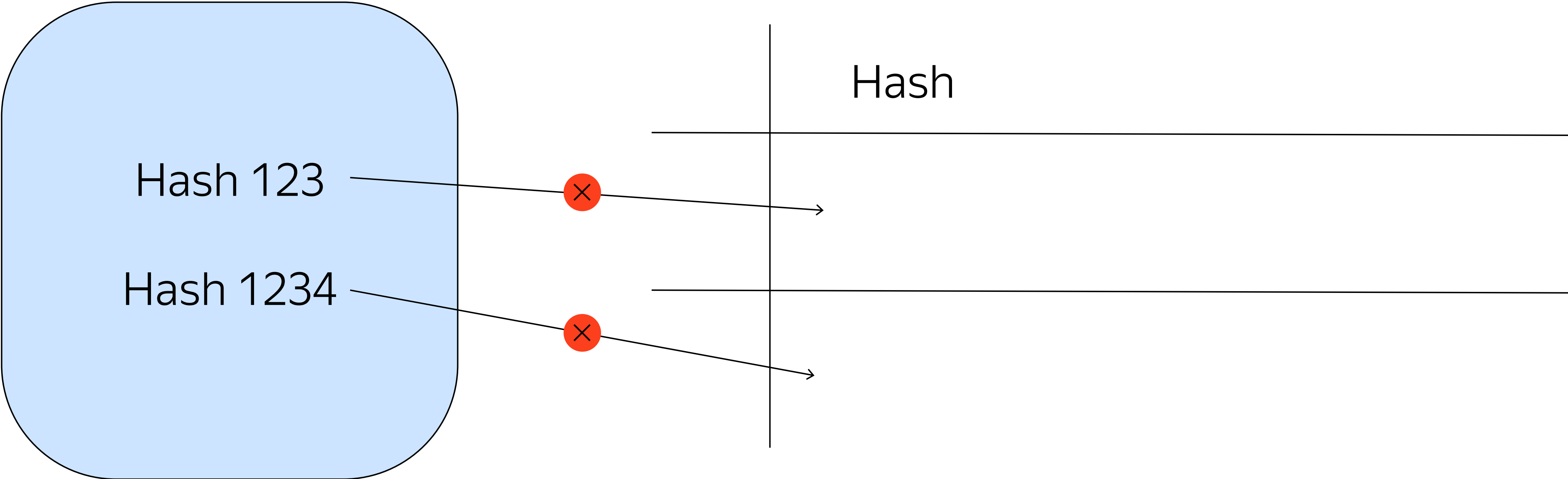
YDB



Транзакционная обработка данных

Chunk

YDB

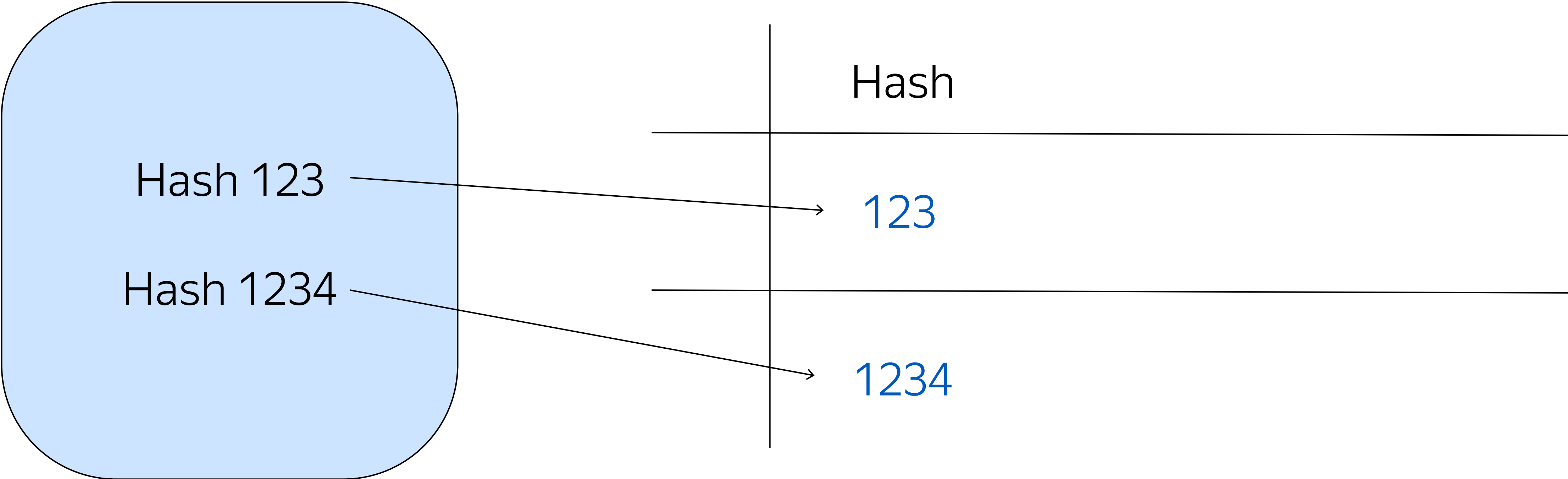


Не дубли

Транзакционная обработка данных

Chunk

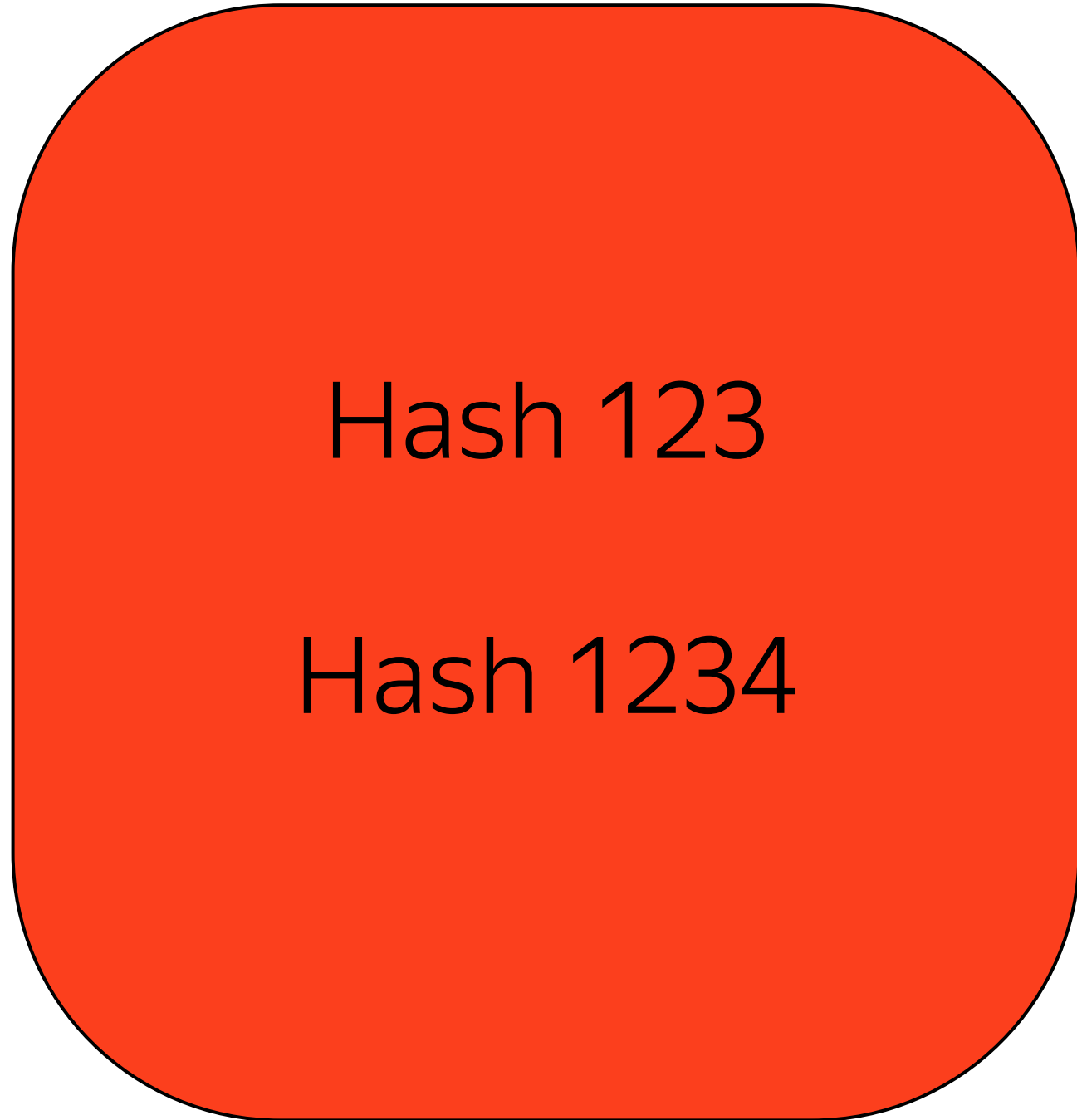
YDB



Транзакционная обработка данных

Chunk

YDB



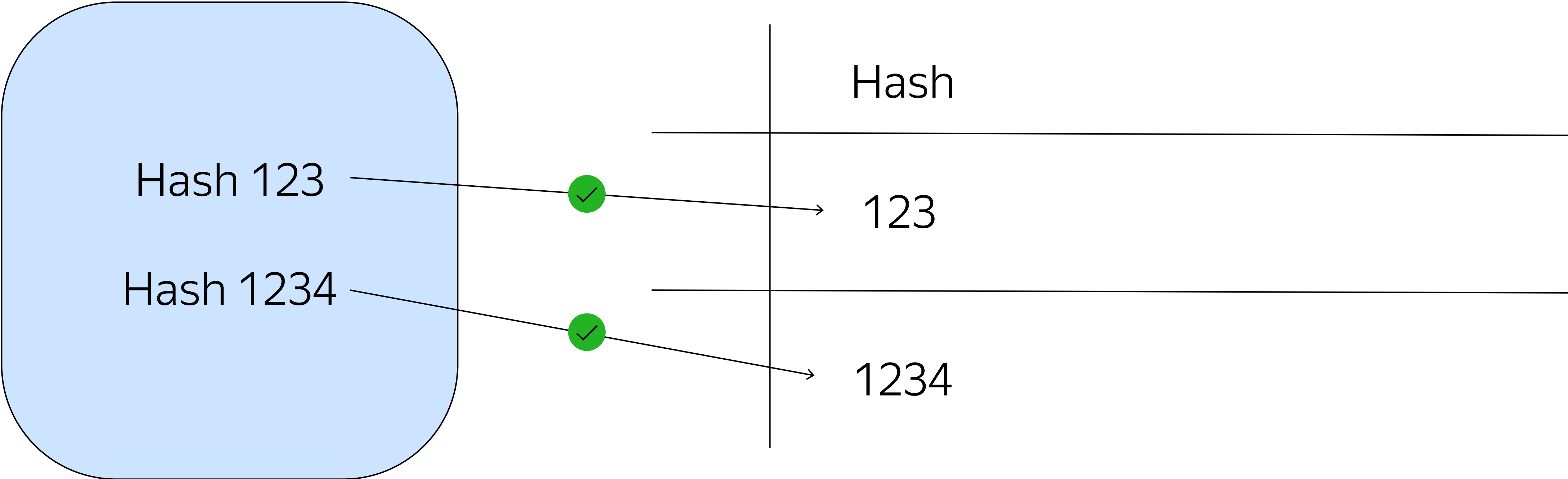
Hash
123
1234

Произошла ошибка при обработке

Транзакционная обработка данных

Chunk

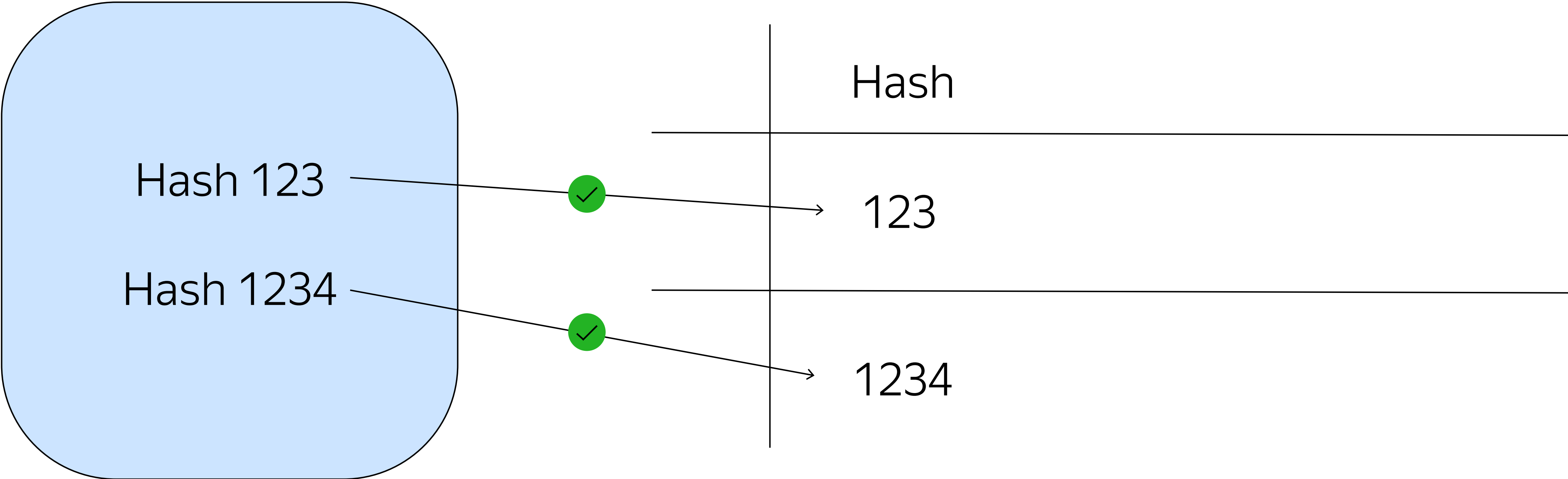
YDB



Транзакционная обработка данных

Chunk

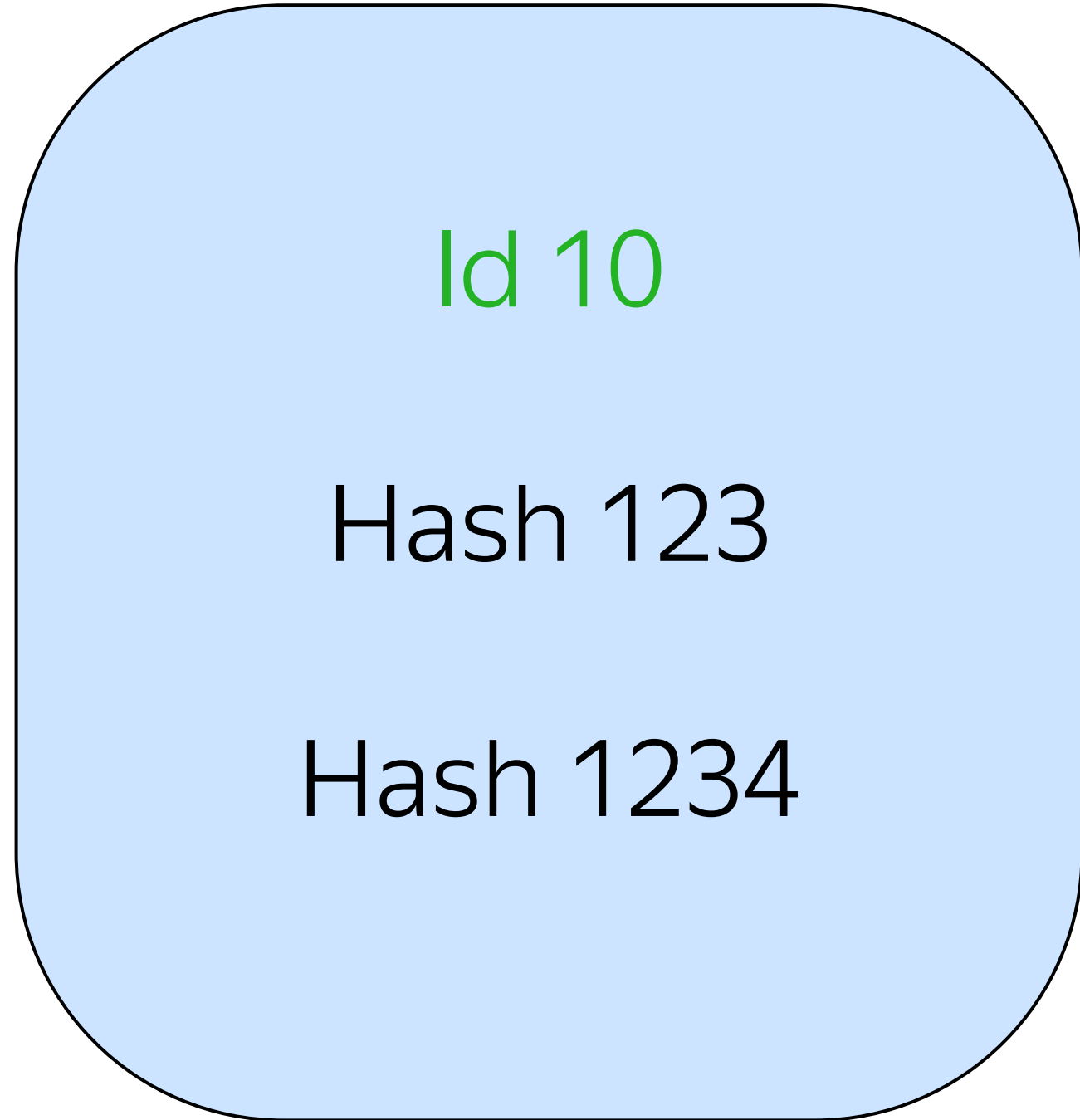
YDB



Дубли – неверный ответ

Транзакционная обработка данных

Chunk



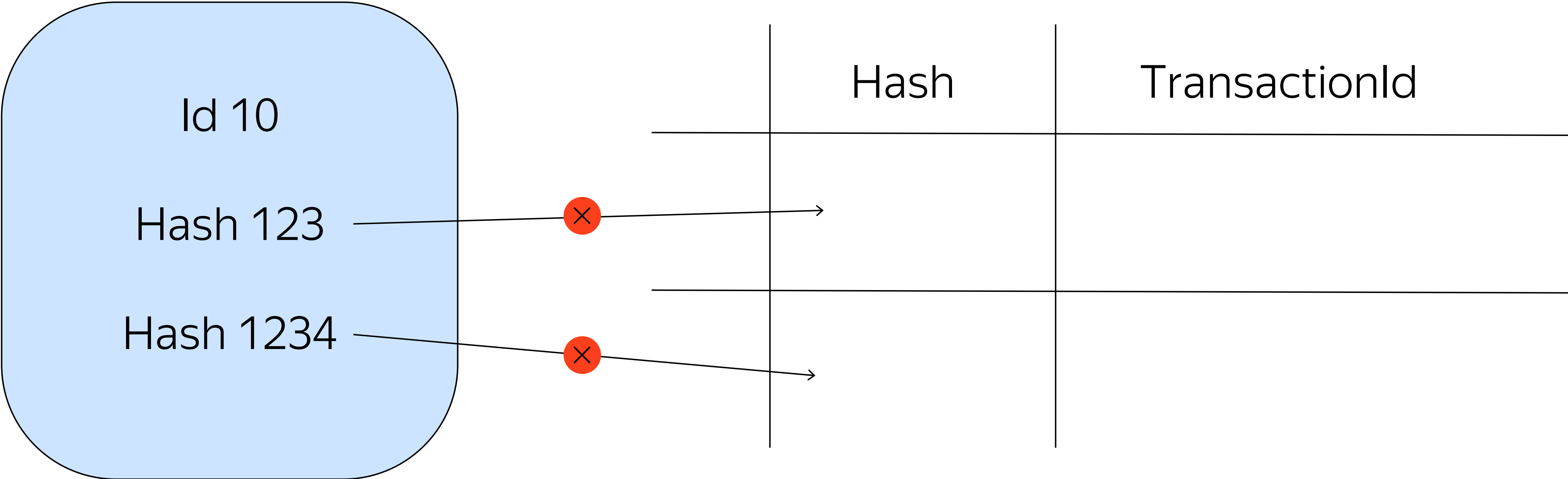
YDB

	Hash	TransactionId

Транзакционная обработка данных

Chunk

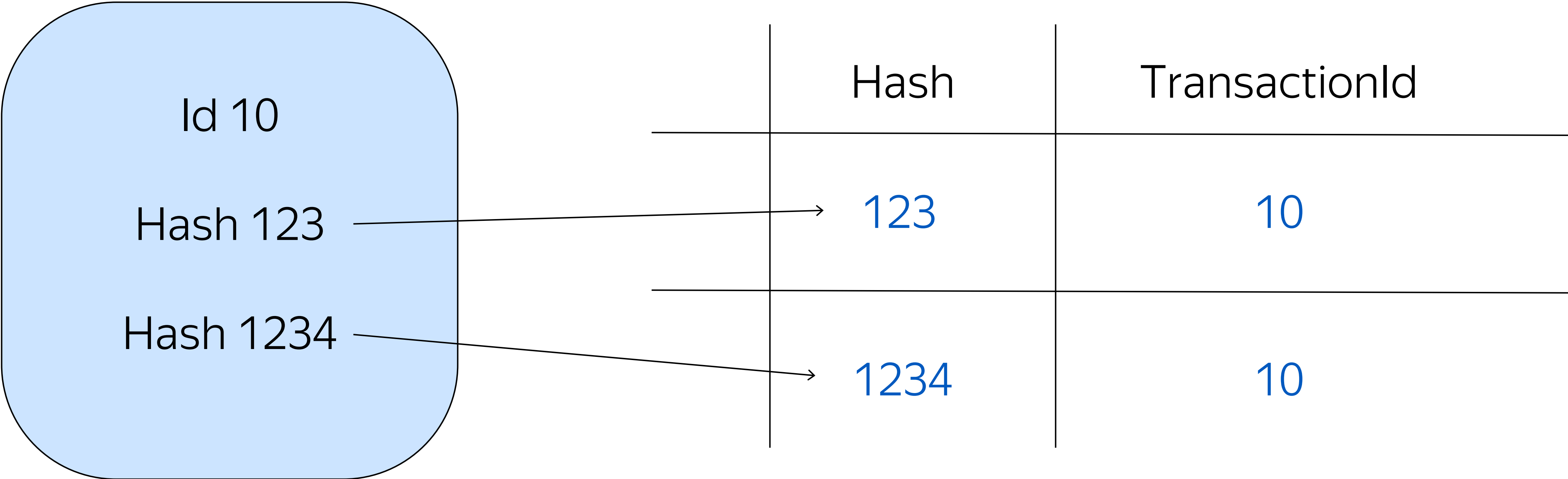
YDB



Транзакционная обработка данных

Chunk

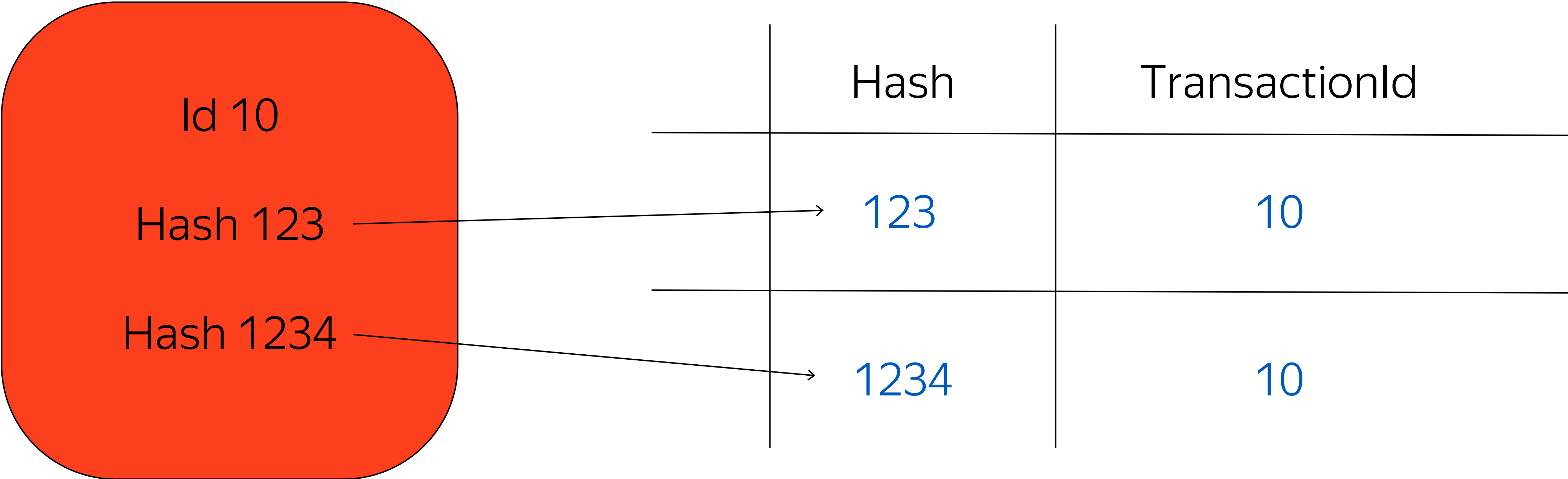
YDB



Транзакционная обработка данных

Chunk

YDB

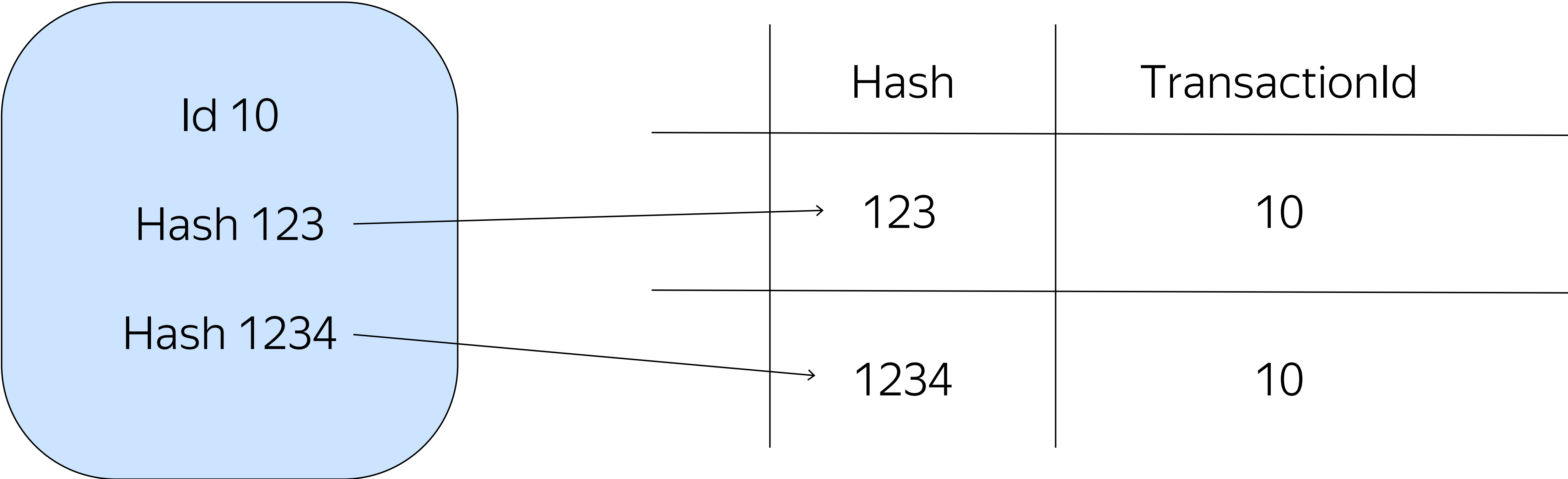


Произошла ошибка при обработке

Транзакционная обработка данных

Chunk

YDB

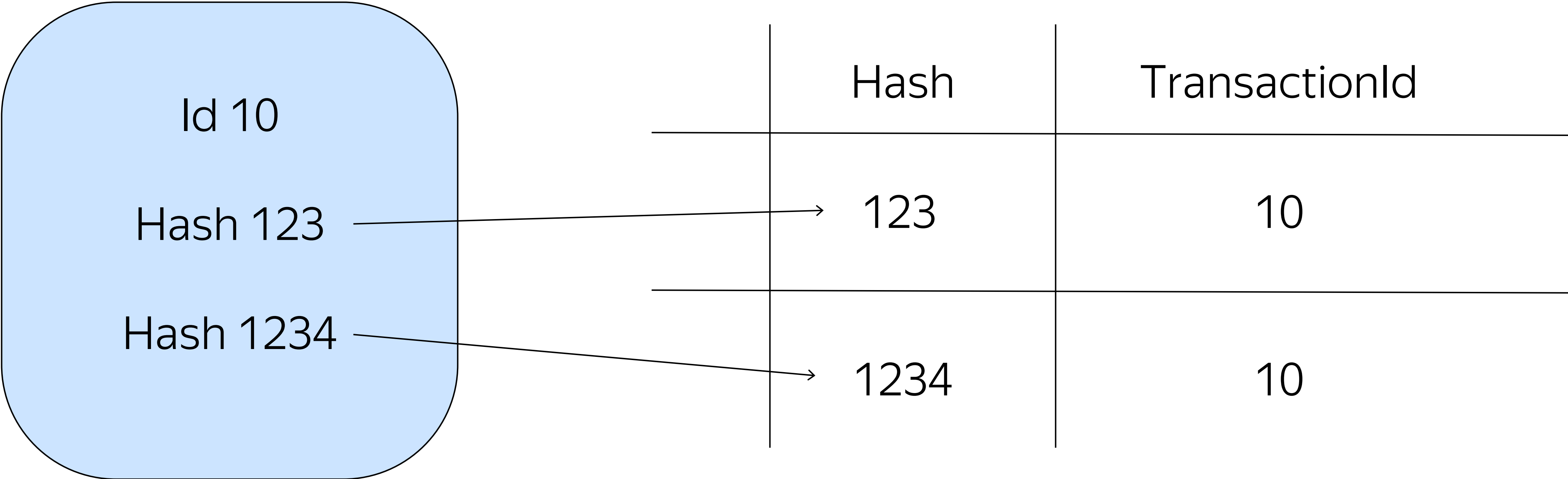


TransactionId = ChunkId – не дубли

Транзакционная обработка данных

Chunk

YDB

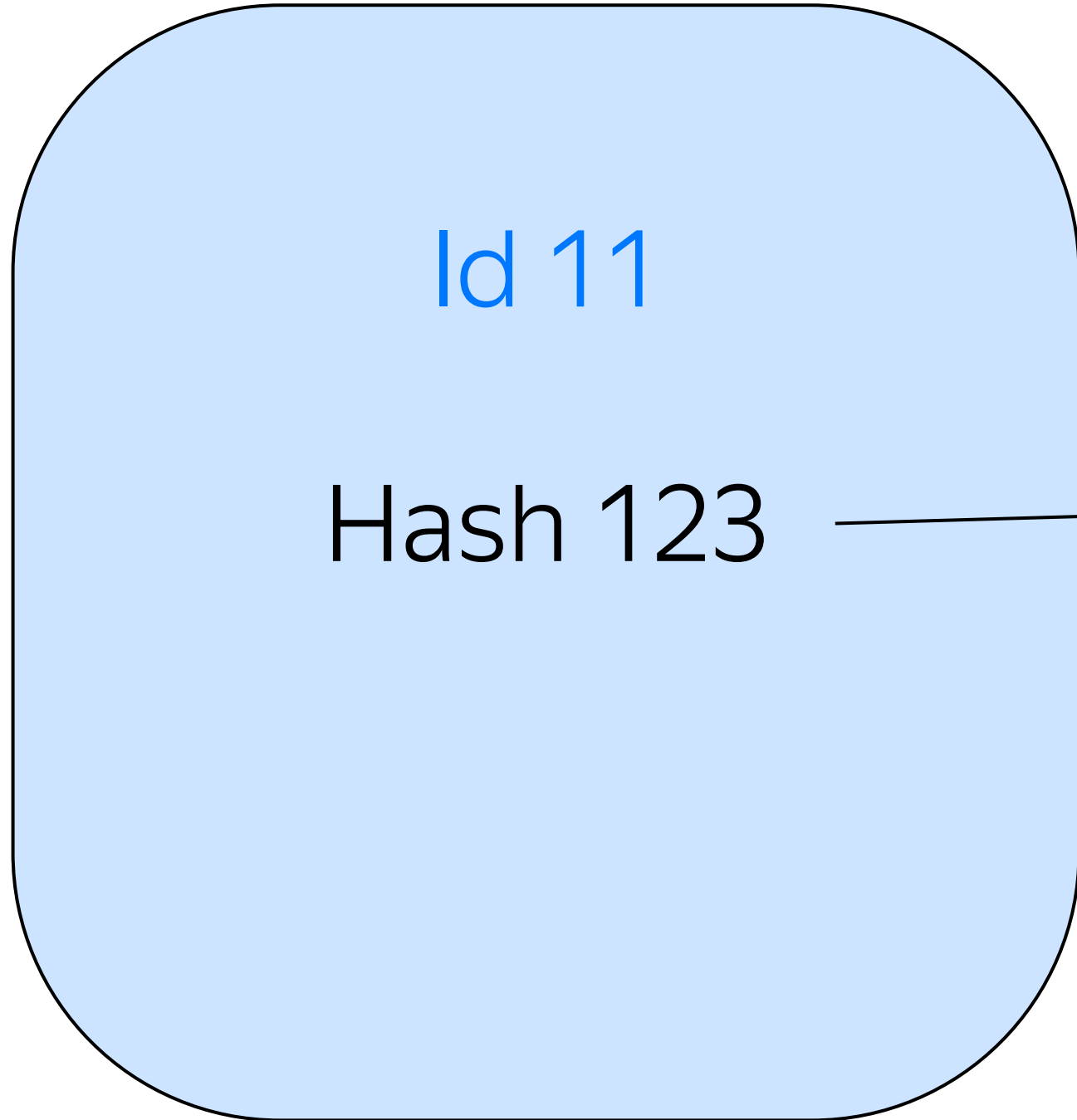


Не дубли – **верный ответ**

Транзакционная обработка данных

Chunk

YDB



Hash	TransactionId
123	10
1234	10

TransactionId != ChunkId – дубль

Промежуточные результаты

- ✔ Полноценный облачный сервис дедупликации

Промежуточные результаты

- ✓ Полноценный облачный сервис дедупликации
- ✓ Простая конструкция, хорошо работает для потока событий до 300K RPS

Промежуточные результаты

- ✓ Полноценный облачный сервис дедупликации
- ✓ Простая конструкция, хорошо работает для потока событий до 300K RPS
- ✗ TTL на большой таблице тратит много ресурсов

Промежуточные результаты

- ✓ Полноценный облачный сервис дедупликации
- ✓ Простая конструкция, хорошо работает для потока событий до 300K RPS
- ✗ TTL на большой таблице тратит много ресурсов
- ✗ Работающий TTL ухудшает производительность Select и Insert

Промежуточные результаты

- ✓ Полноценный облачный сервис дедупликации
- ✓ Простая конструкция, хорошо работает для потока событий до 300K RPS
- ✗ TTL на большой таблице тратит много ресурсов
- ✗ Работающий TTL ухудшает производительность Select и Insert
- ✗ При потоке в 300K событий в секунду потребляет 250 ядер YDB

Промежуточные результаты

- ✓ Полноценный облачный сервис дедупликации
- ✓ Простая конструкция, хорошо работает для потока событий до 300K RPS
- ✗ TTL на большой таблице тратит много ресурсов
- ✗ Работающий TTL ухудшает производительность Select и Insert
- ✗ При потоке в 300K событий в секунду потребляет 250 ядер YDB
- ✗ Для потока в 5M событий в секунду потребовалось бы 3500 ядер YDB

Промежуточные результаты

- ✓ Полноценный облачный сервис дедупликации
- ✓ Простая конструкция, хорошо работает для потока событий до 300K RPS
- ✗ TTL на большой таблице тратит много ресурсов
- ✗ Работающий TTL ухудшает производительность Select и Insert
- ✗ При потоке в 300K событий в секунду потребляет 250 ядер YDB
- ✗ Для потока в 5M событий в секунду потребовалось бы 3500 ядер YDB

Необходимо работать над производительностью

Отказ от TTL

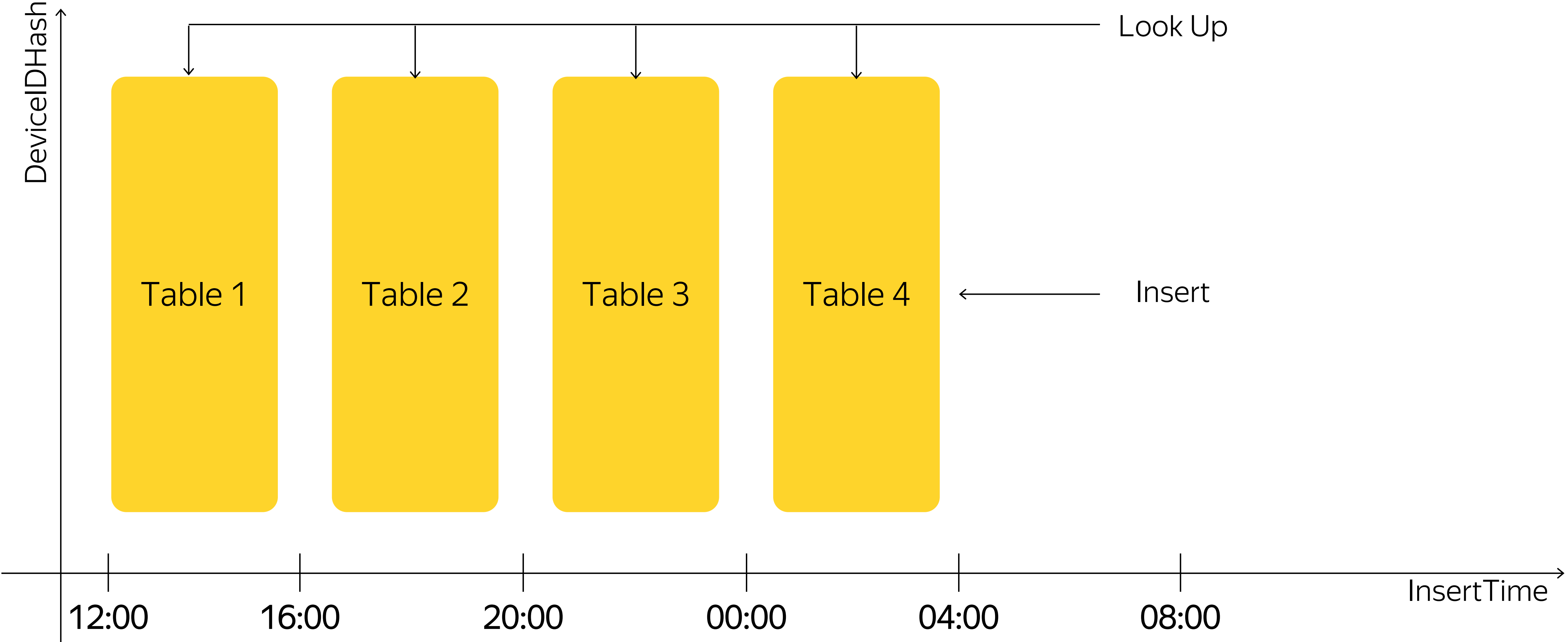
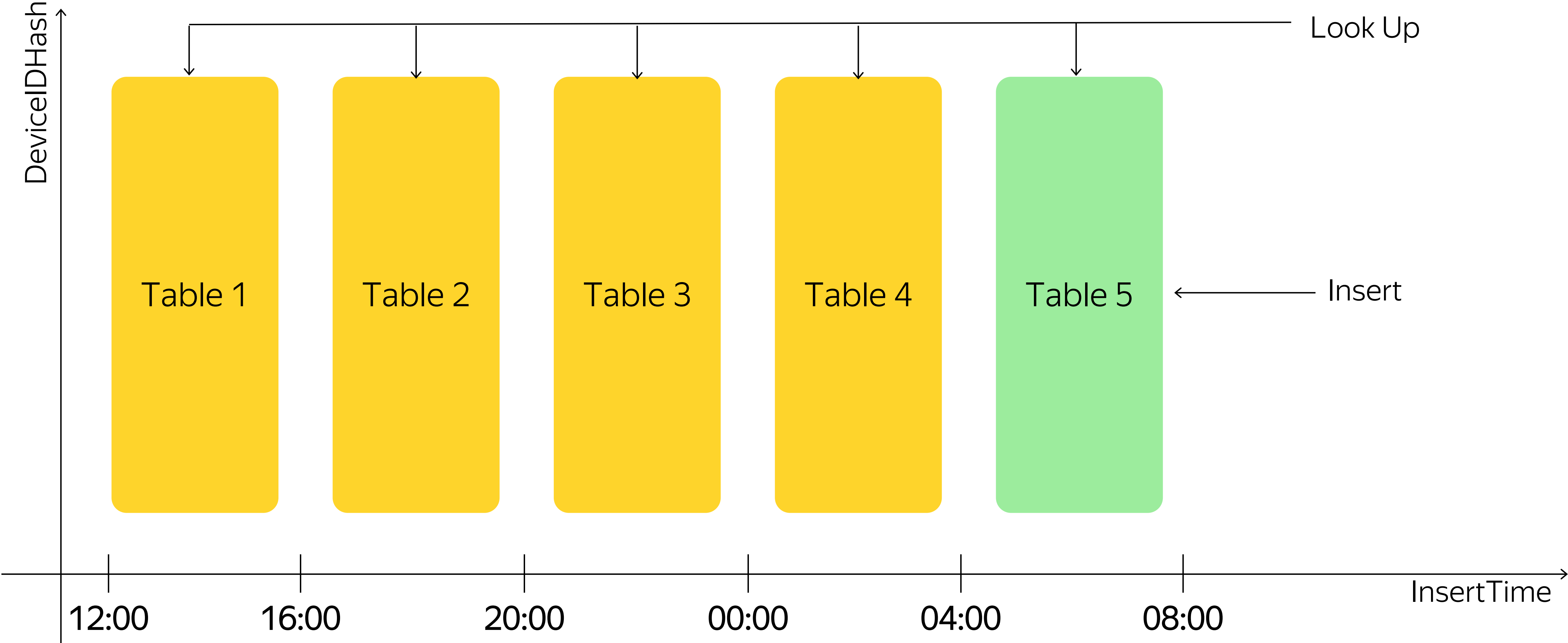


Схема таблицы – (DeviceIDHash, EventHash, TransactionID)

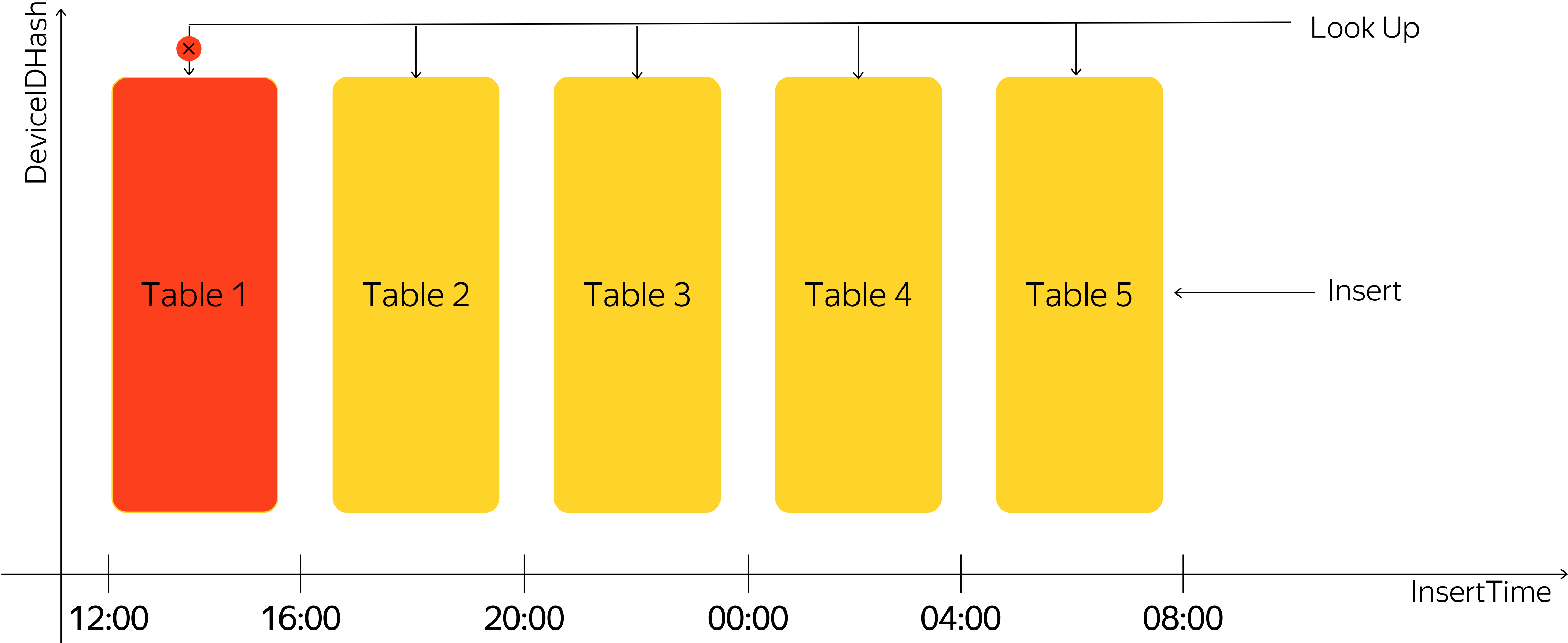
Отказ от TTL

Когда время очередной таблицы закончилось, добавляем новую



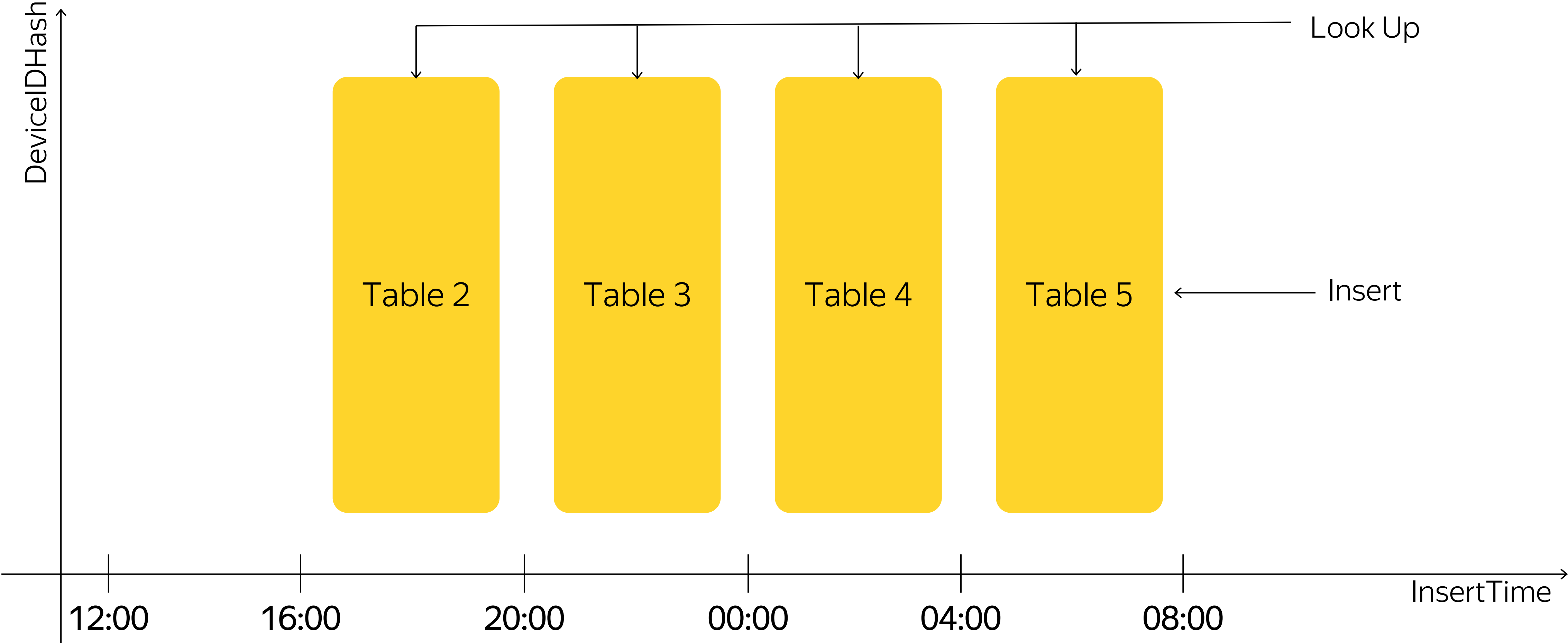
Отказ от TTL

Когда таблица стала слишком старой, удаляем ее полностью



Отказ от TTL

Когда таблица стала слишком старой, удаляем ее полностью



Отказ от TTL

- ✔ TTL не тратит ресурсы базы

Отказ от TTL

- ✓ TTL не тратит ресурсы базы
- ✓ Колонка InsertTime не занимает место на дисках

Отказ от TTL

- ✓ TTL не тратит ресурсы базы
- ✓ Колонка InsertTime не занимает место на дисках
- ✓ Размер таблицы не ограничен производительностью TTL

Отказ от TTL

- ✓ TTL не тратит ресурсы базы
- ✓ Колонка InsertTime не занимает место на дисках
- ✓ Размер таблицы не ограничен производительностью TTL
- ✗ Число look up умножается на число таблиц

Отказ от TTL

- ✓ TTL не тратит ресурсы базы
- ✓ Колонка InsertTime не занимает место на дисках
- ✓ Размер таблицы не ограничен производительностью TTL
- ✗ Число look up умножается на число таблиц
- ✗ Очень дорого делать 4 x 5M look up

Отказ от TTL

- ✓ TTL не тратит ресурсы базы
- ✓ Колонка InsertTime не занимает место на дисках
- ✓ Размер таблицы не ограничен производительностью TTL
- ✗ Число look up умножается на число таблиц
- ✗ Очень дорого делать 4 x 5M look up

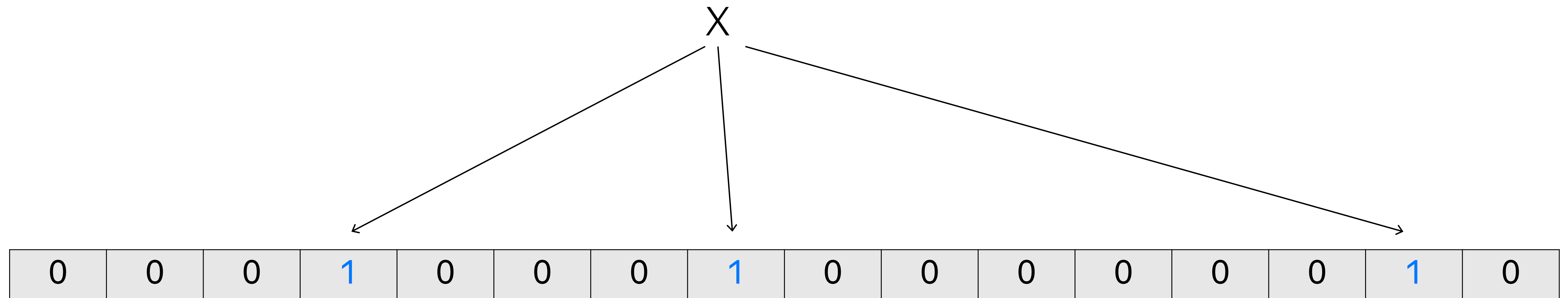
Нужно уменьшать количество look up

Фильтр Блума

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

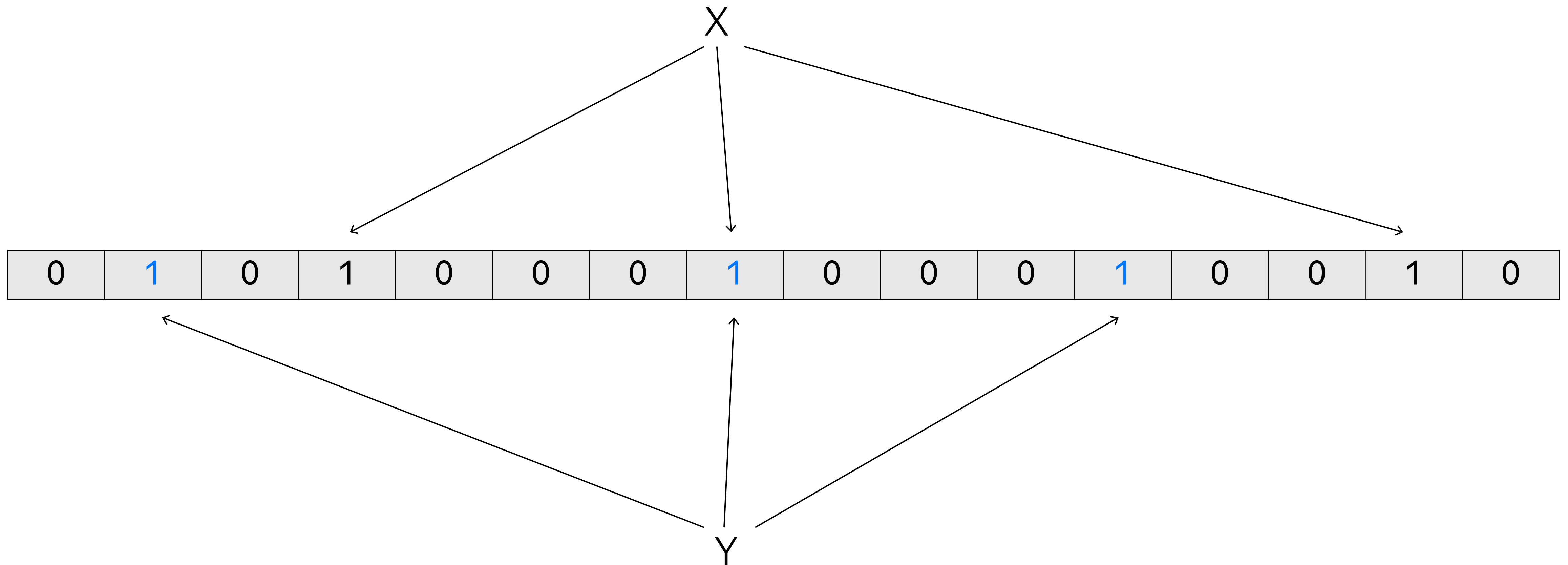
Фильтр Блума

Добавление элемента



Фильтр Блума

Добавление элемента



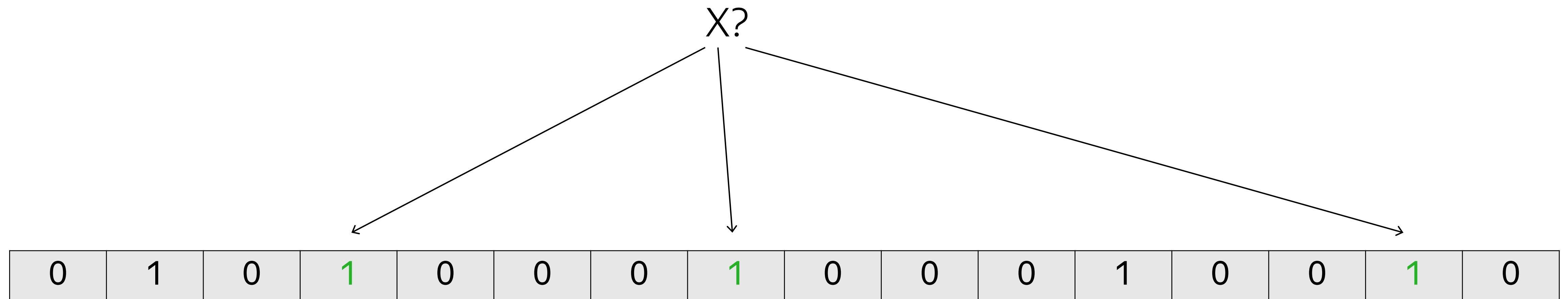
Фильтр Блума

Проверка наличия

0	1	0	1	0	0	0	1	0	0	0	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

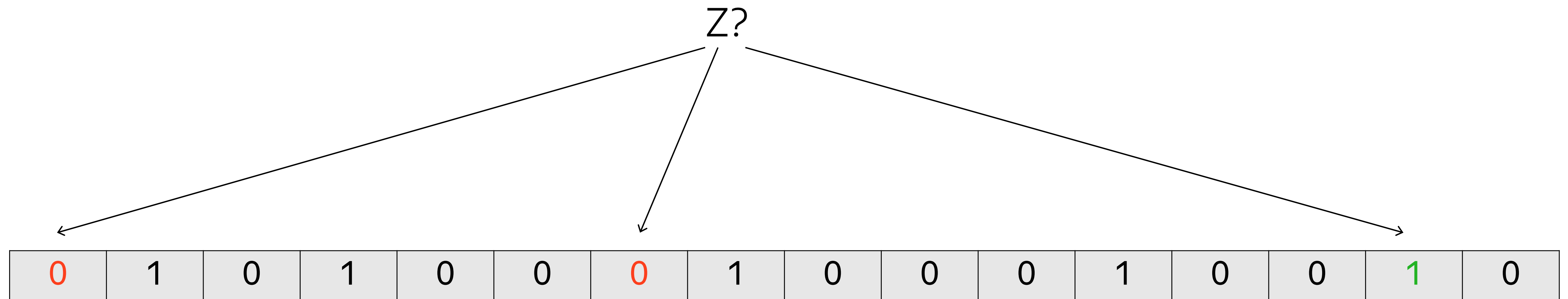
Фильтр Блума

Проверка наличия



Фильтр Блума

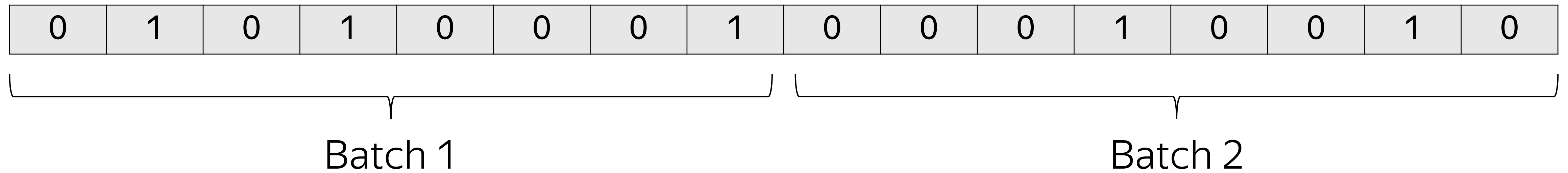
Проверка наличия



Фильтр Блума

Хранение в таблице YDB

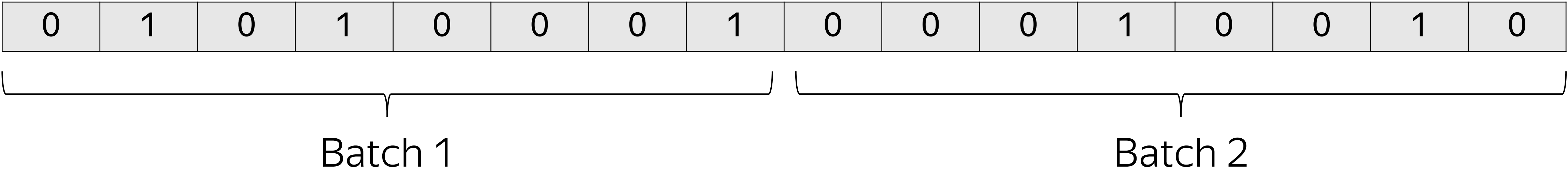
RAM



Фильтр Блума

Хранение в таблице YDB

RAM



YDB

BatchIndex	BatchBits
1	0 1 0 1 0 0 0 1
2	0 0 0 1 0 0 1 0

Фильтр Блума

Хранение в таблице YDB

BatchIndex	BatchBits
1	0 1 0 1 0 0 0 1
2	0 0 0 1 0 0 1 0

Upsert

BatchIndex: 1, BatchBits: 0 1 0 1 1 0 0 1

Фильтр Блума

Хранение в таблице YDB

BatchIndex	BatchBits
1	0 1 0 1 1 0 0 1
2	0 0 0 1 0 0 1 0

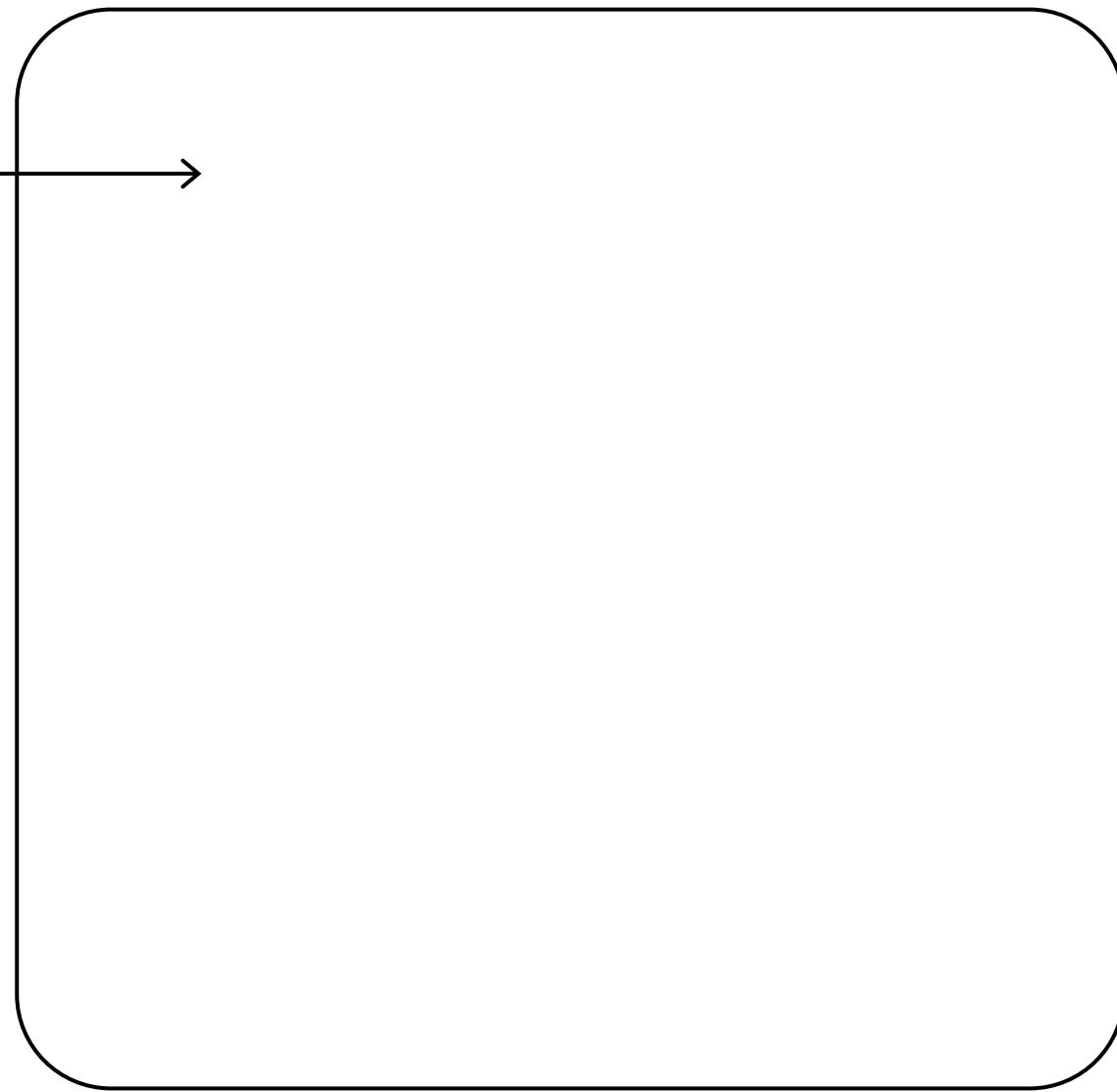
Upsert

BatchIndex: 1, BatchBits: 0 1 0 1 1 0 0 1

Фильтр Блума

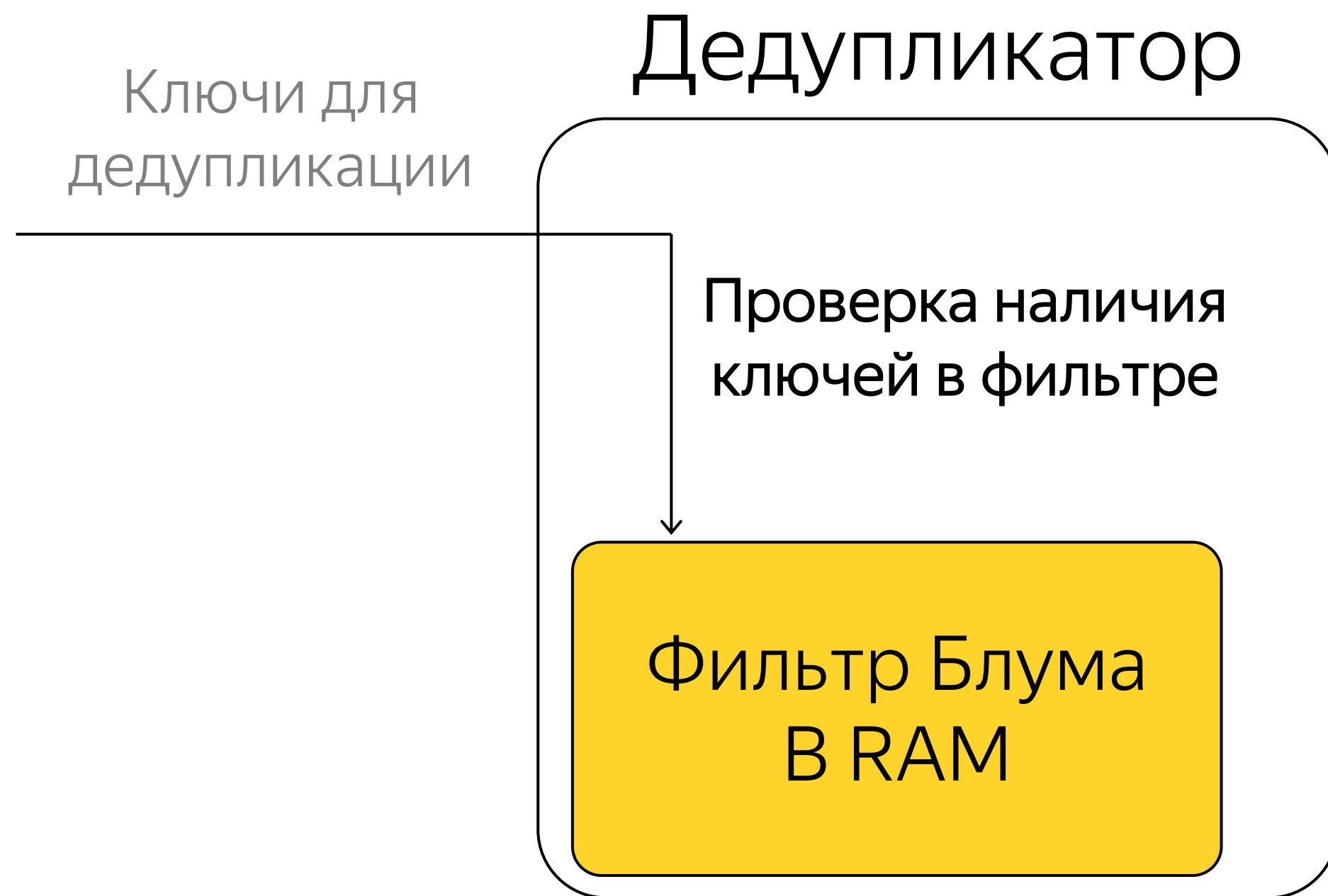
Дедупликатор

Ключи для
дедупликации



Получили запрос с новыми ключами

Фильтр Блума



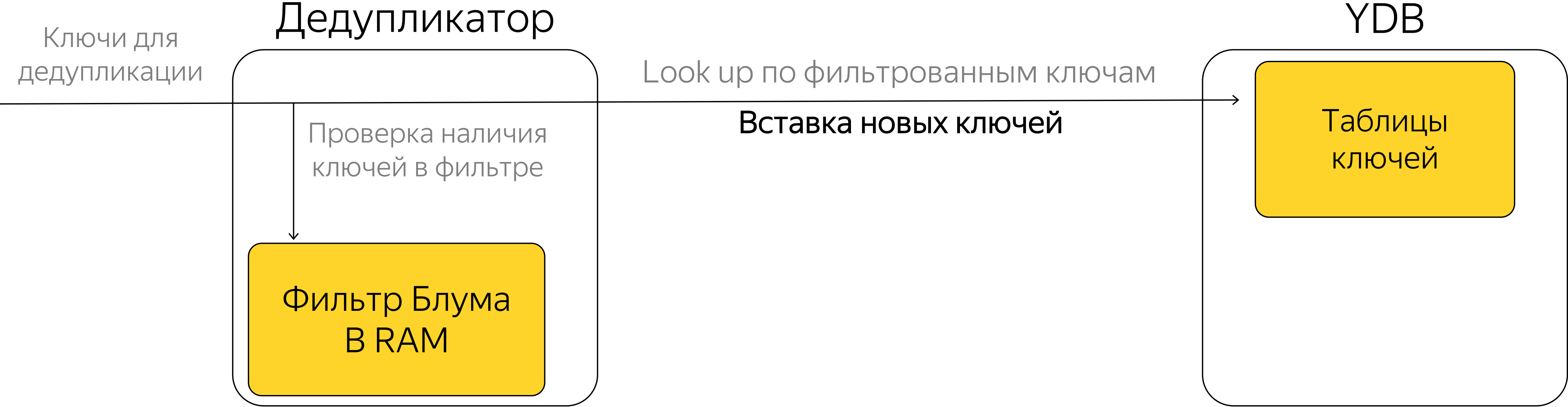
Проверили наличие ключей в фильтре

Фильтр Блума



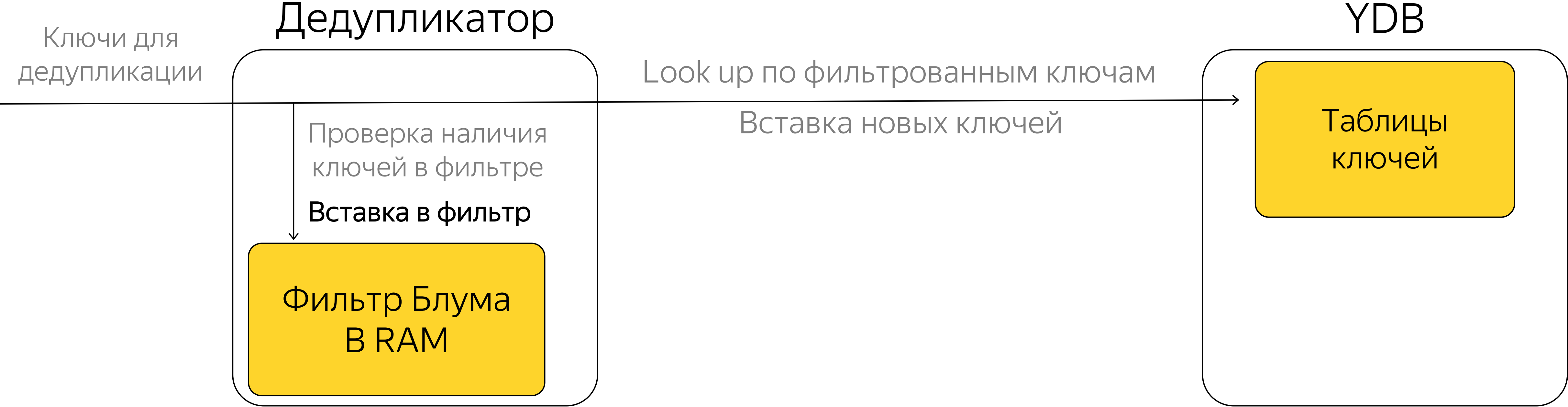
Определили дубли и добавили новые ключи в таблицу

Фильтр Блума



Определили дубли и добавили новые ключи в таблицу

Фильтр Блума



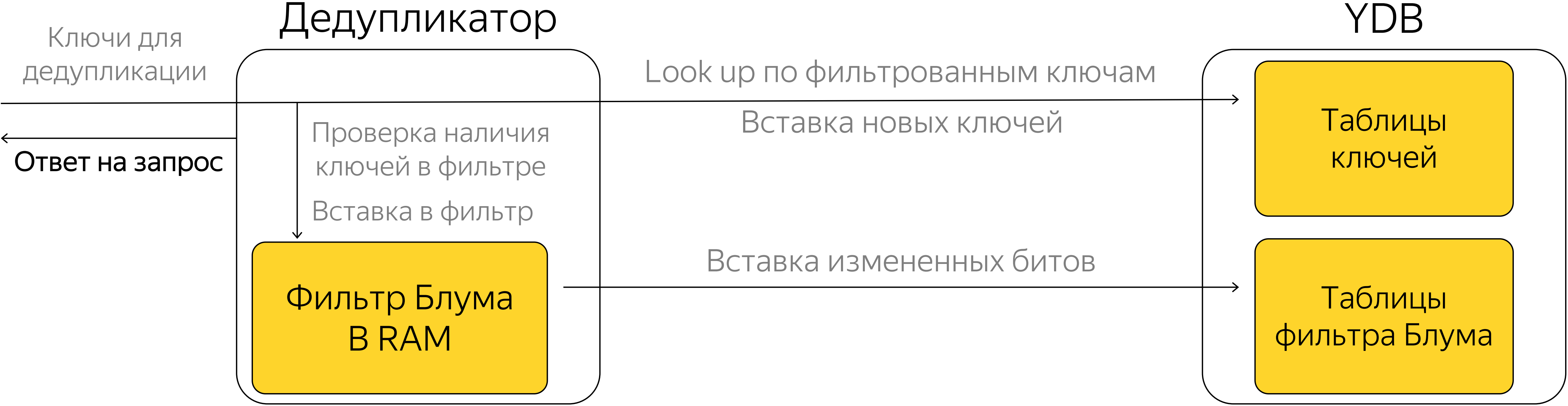
Добавили новые ключи в фильтр Блума

Фильтр Блума



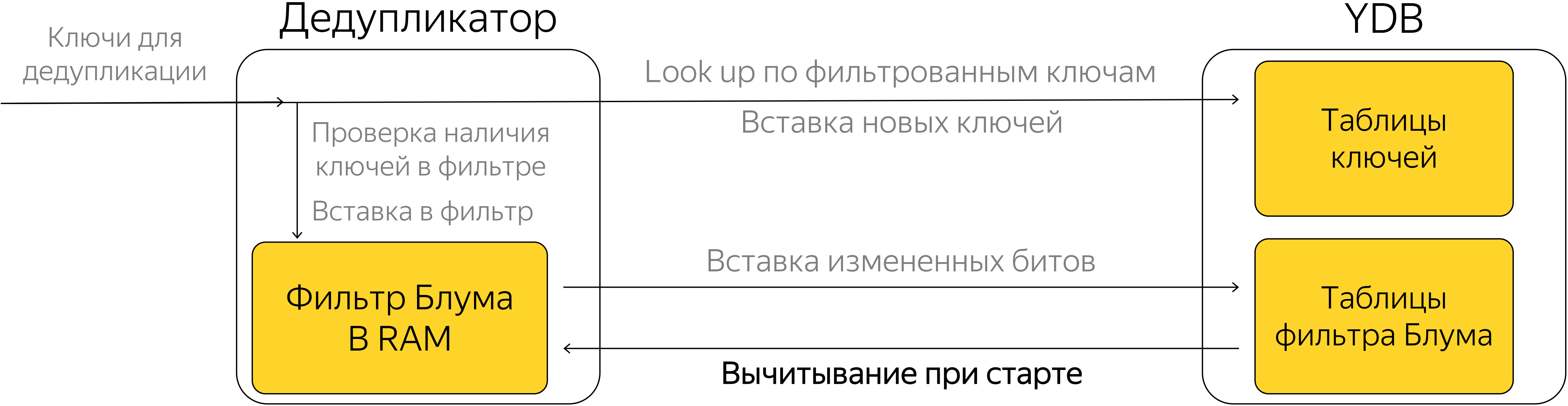
Сохранили состояние фильтра в таблице YDB

Фильтр Блума



Сохранили состояние фильтра в таблице YDB

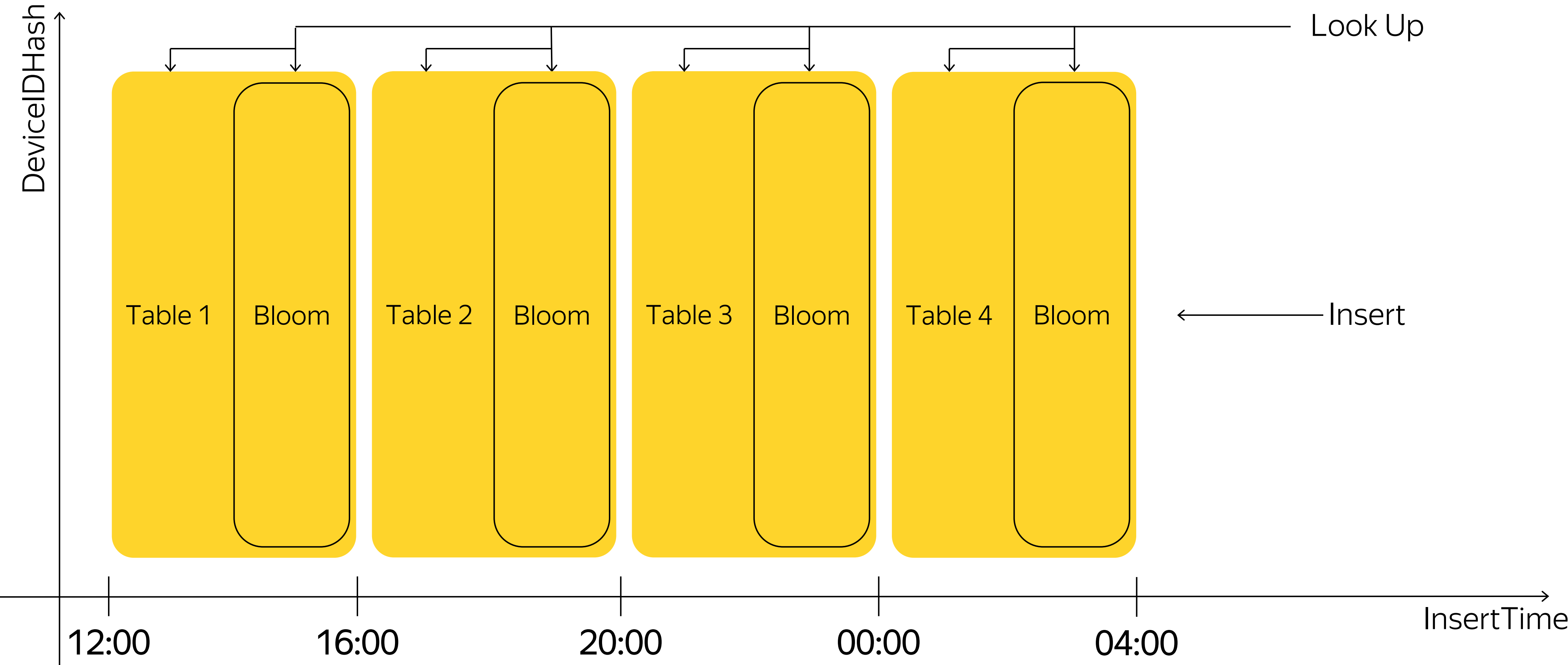
Фильтр Блума



При старте сервиса восстанавливаем фильтр Блума из YDB

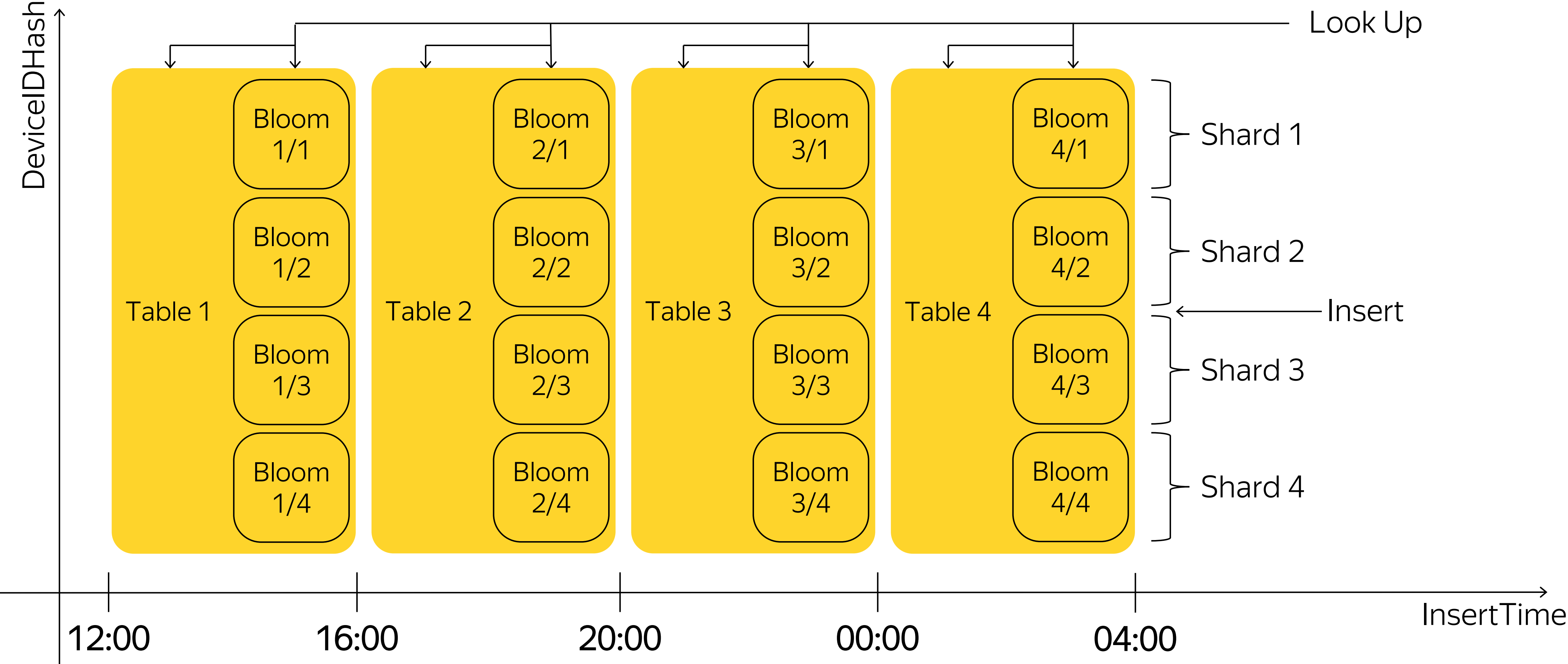
Фильтр Блума

Храним свой фильтр Блума для каждой таблицы

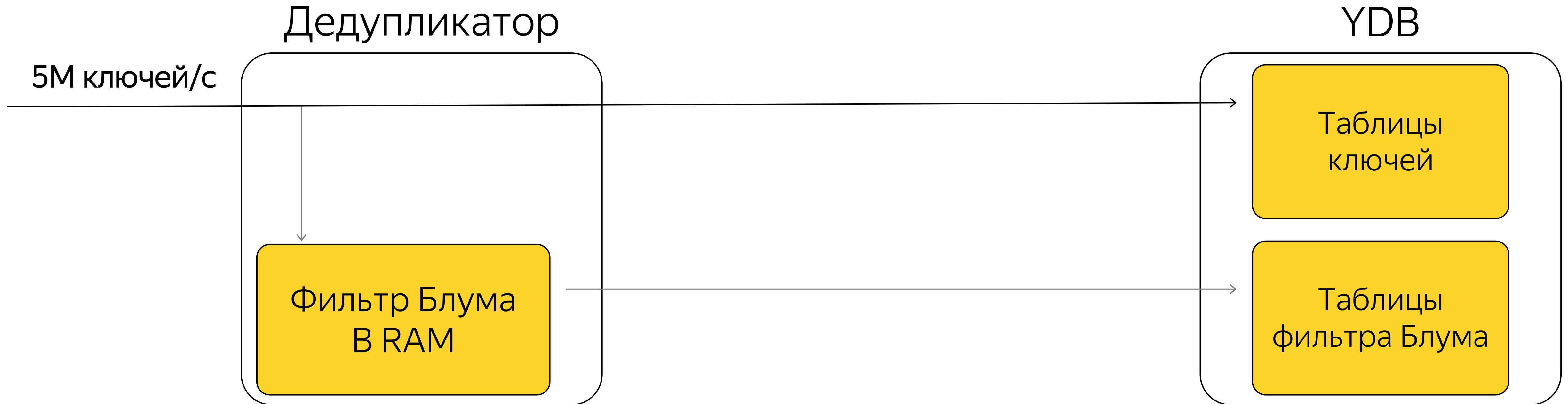


Фильтр Блума

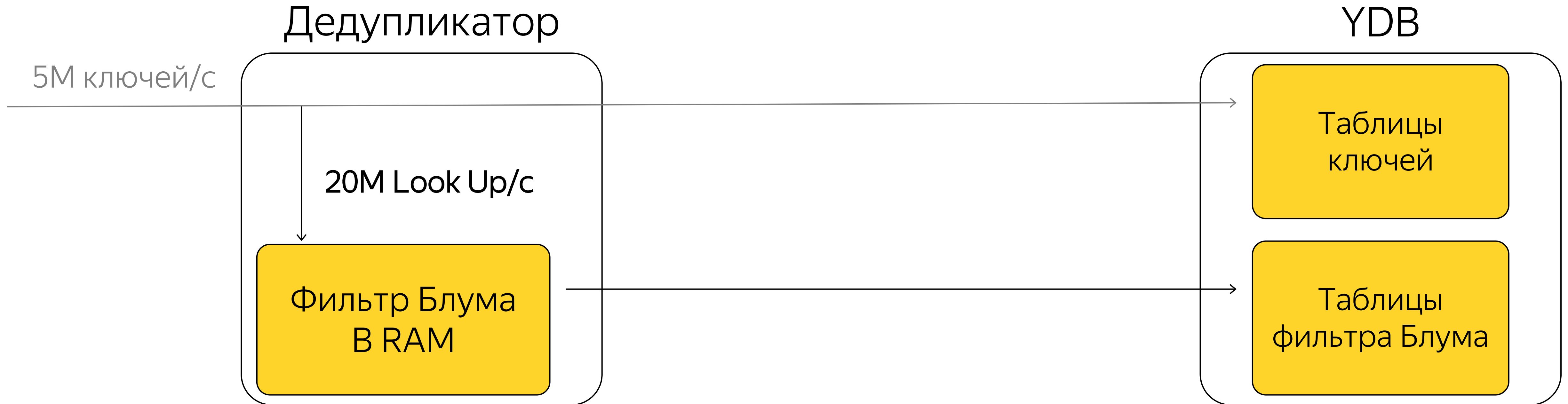
Храним фильтр Блума для каждой пары (таблица, шард)



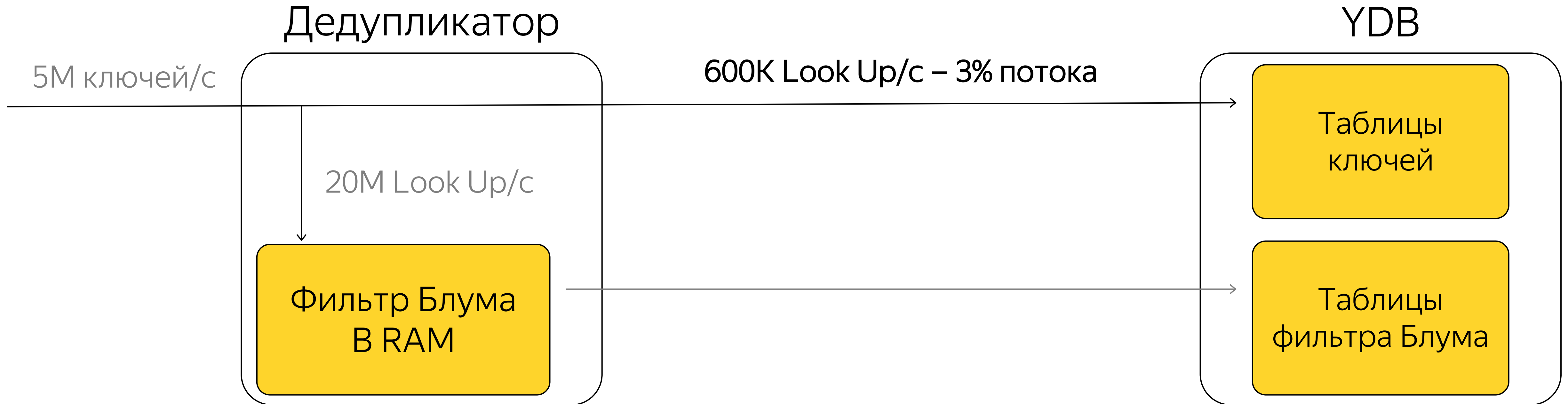
Что получилось



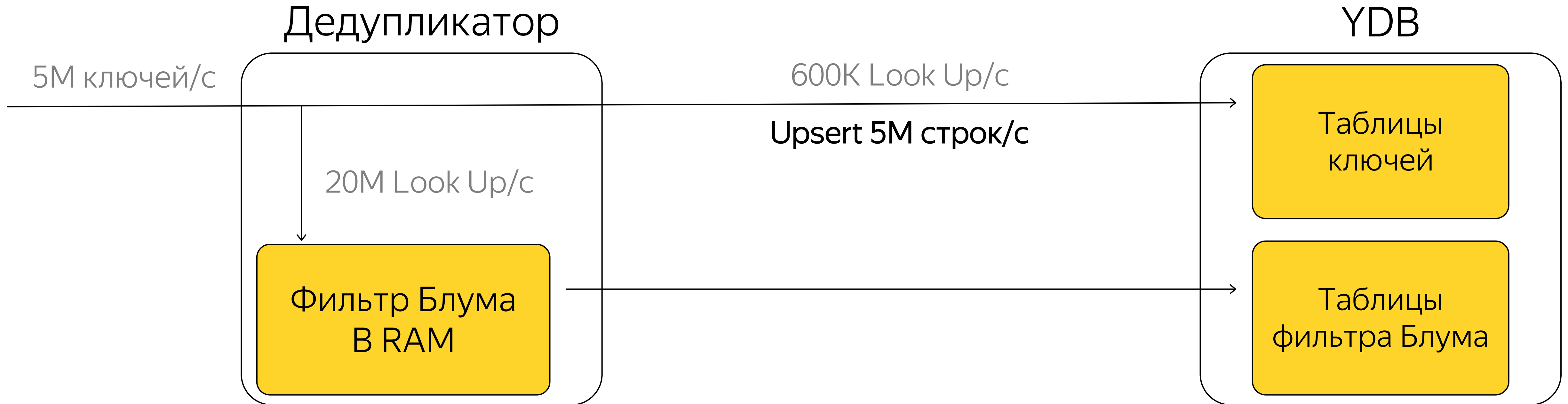
Что получилось



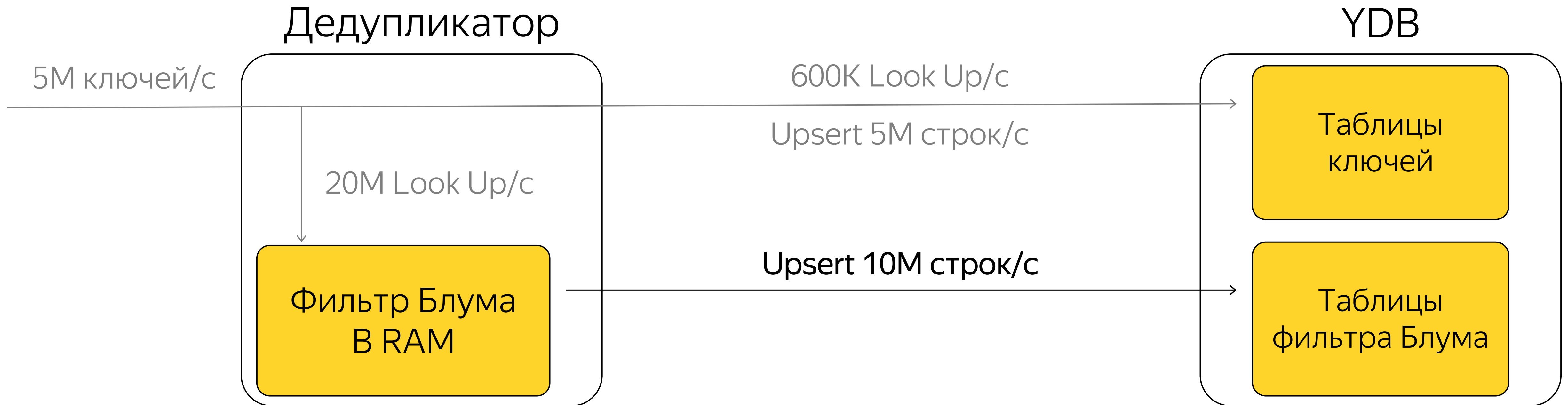
Что получилось



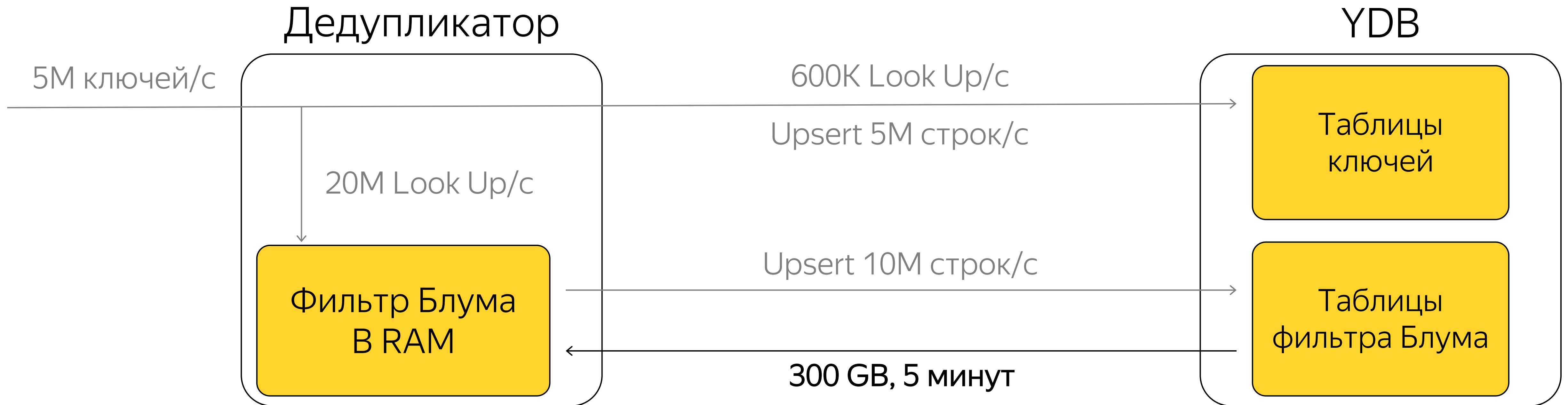
Что получилось



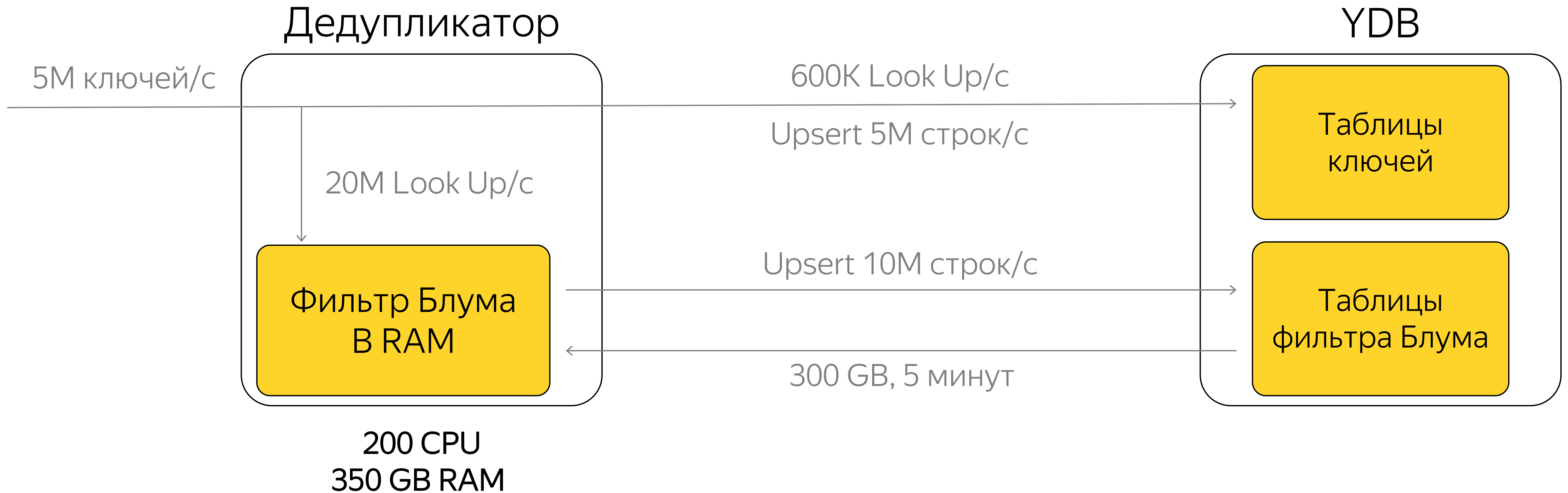
Что получилось



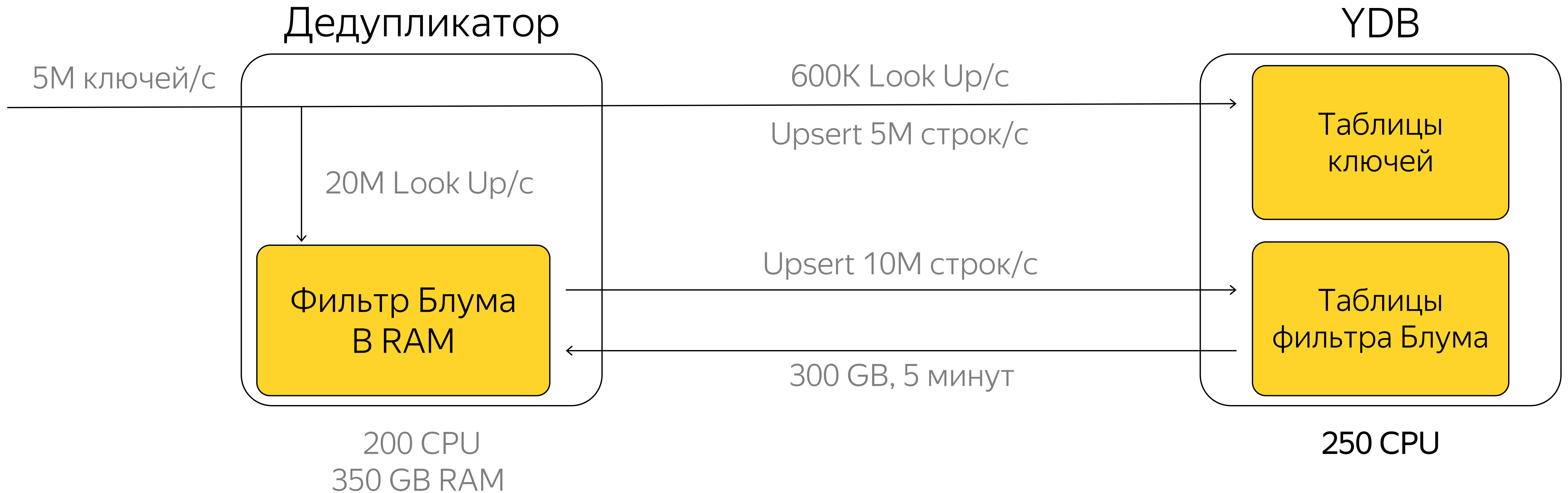
Что получилось



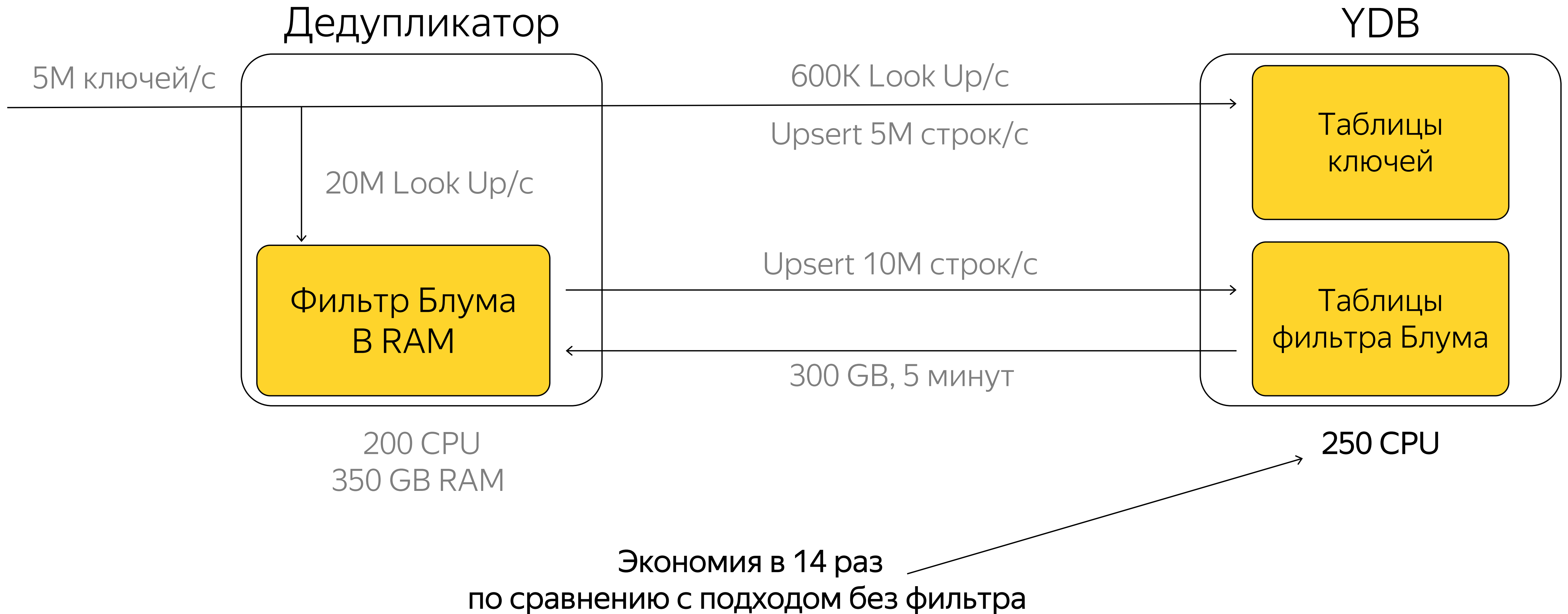
Что получилось



Что получилось

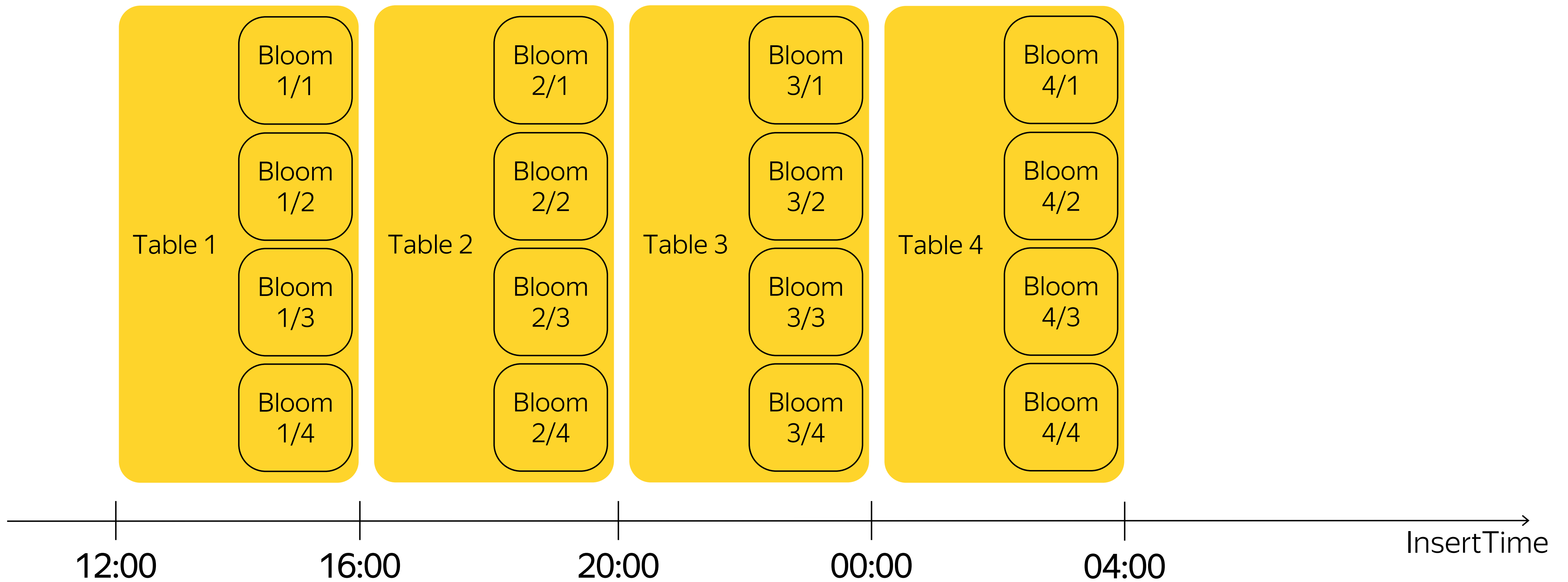


Что получилось

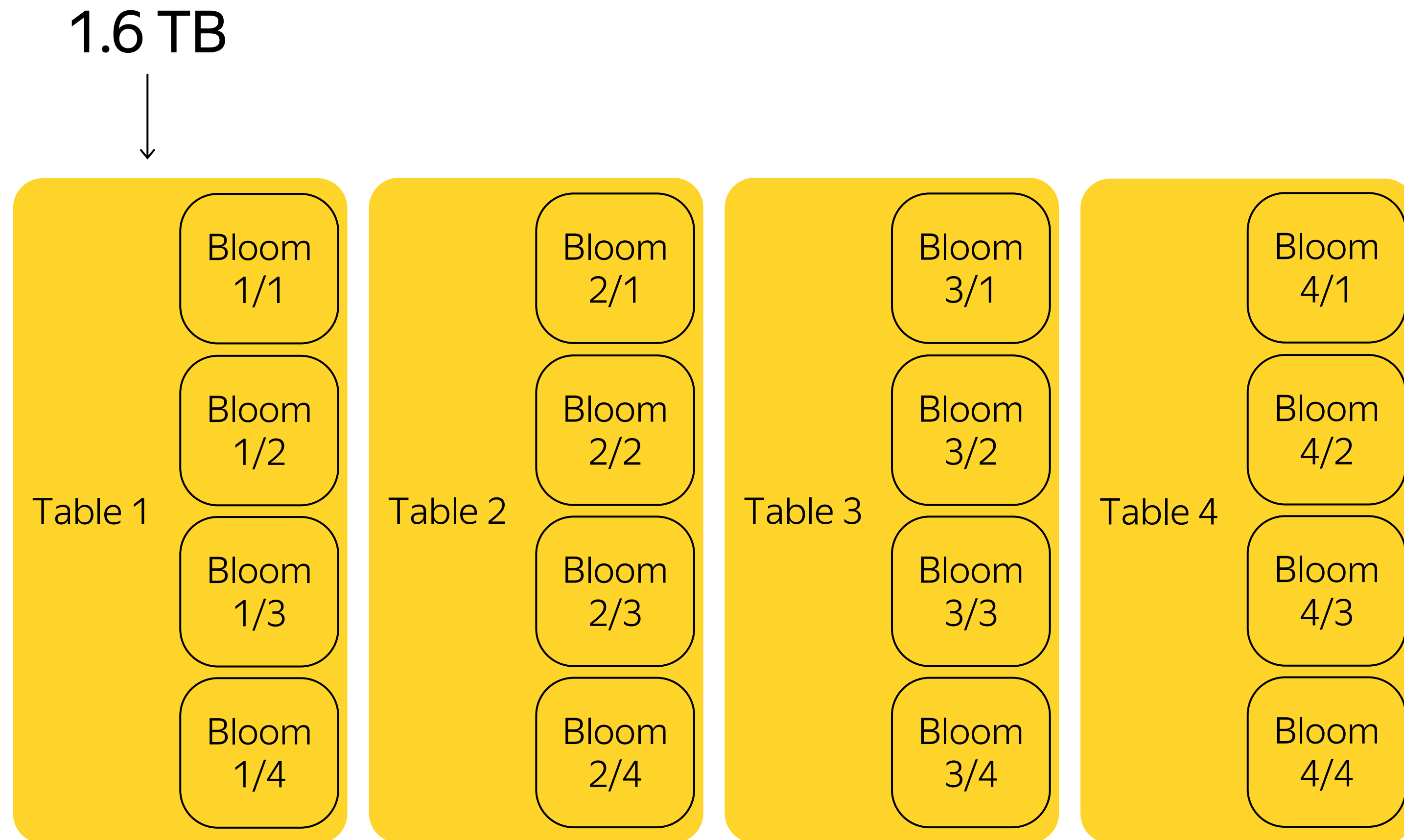


Что получилось

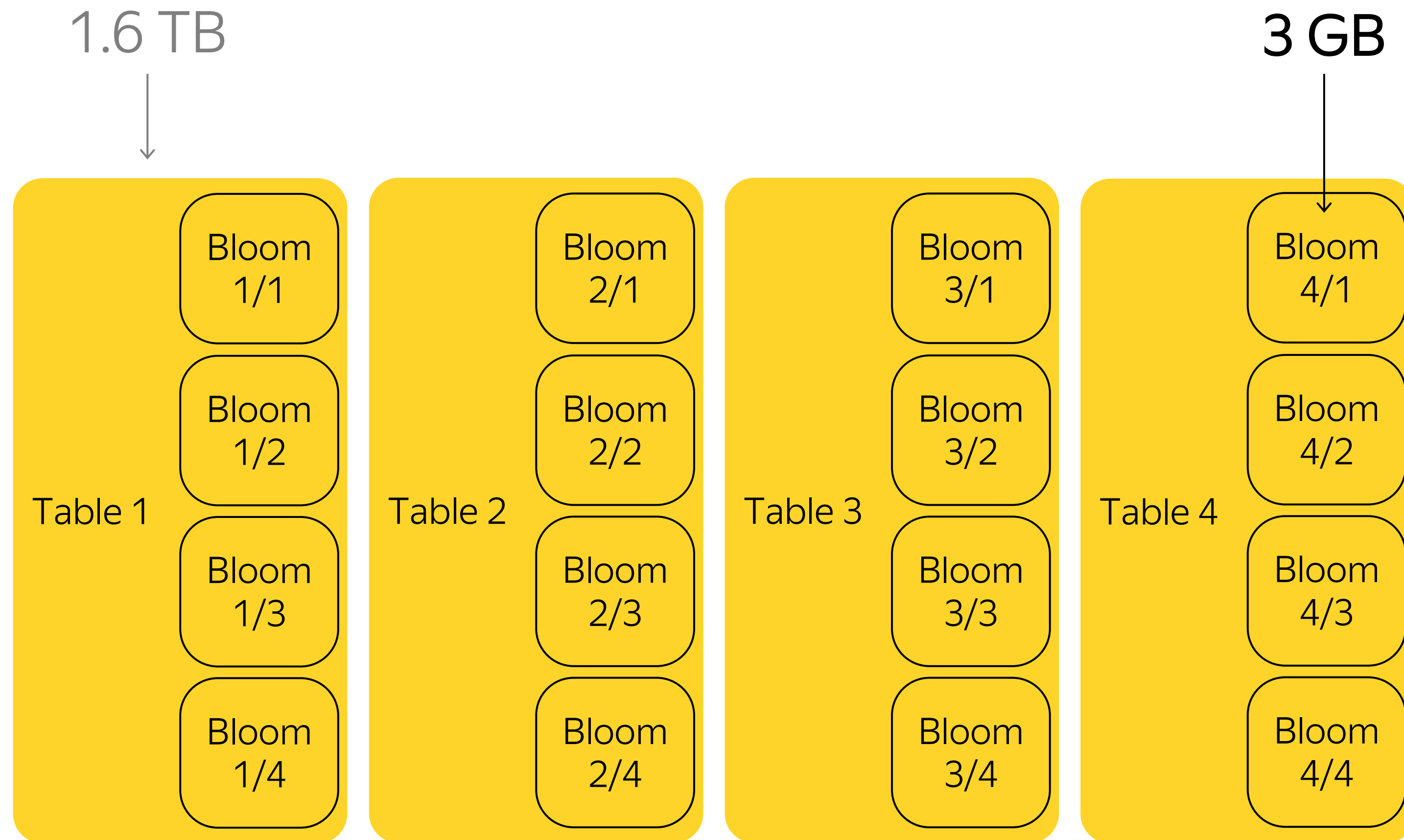
4 таблицы по 4 часа



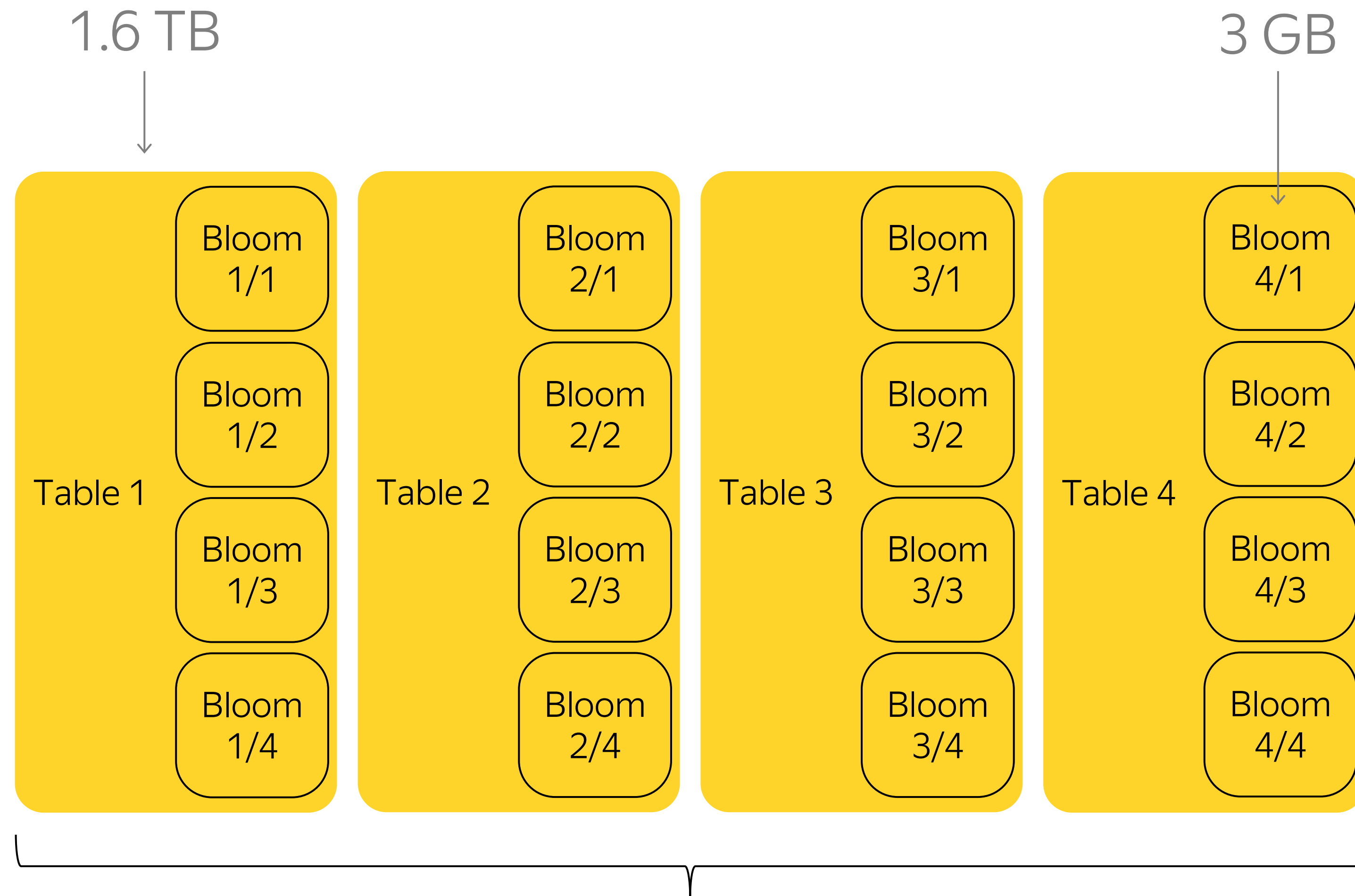
Что получилось



Что получилось

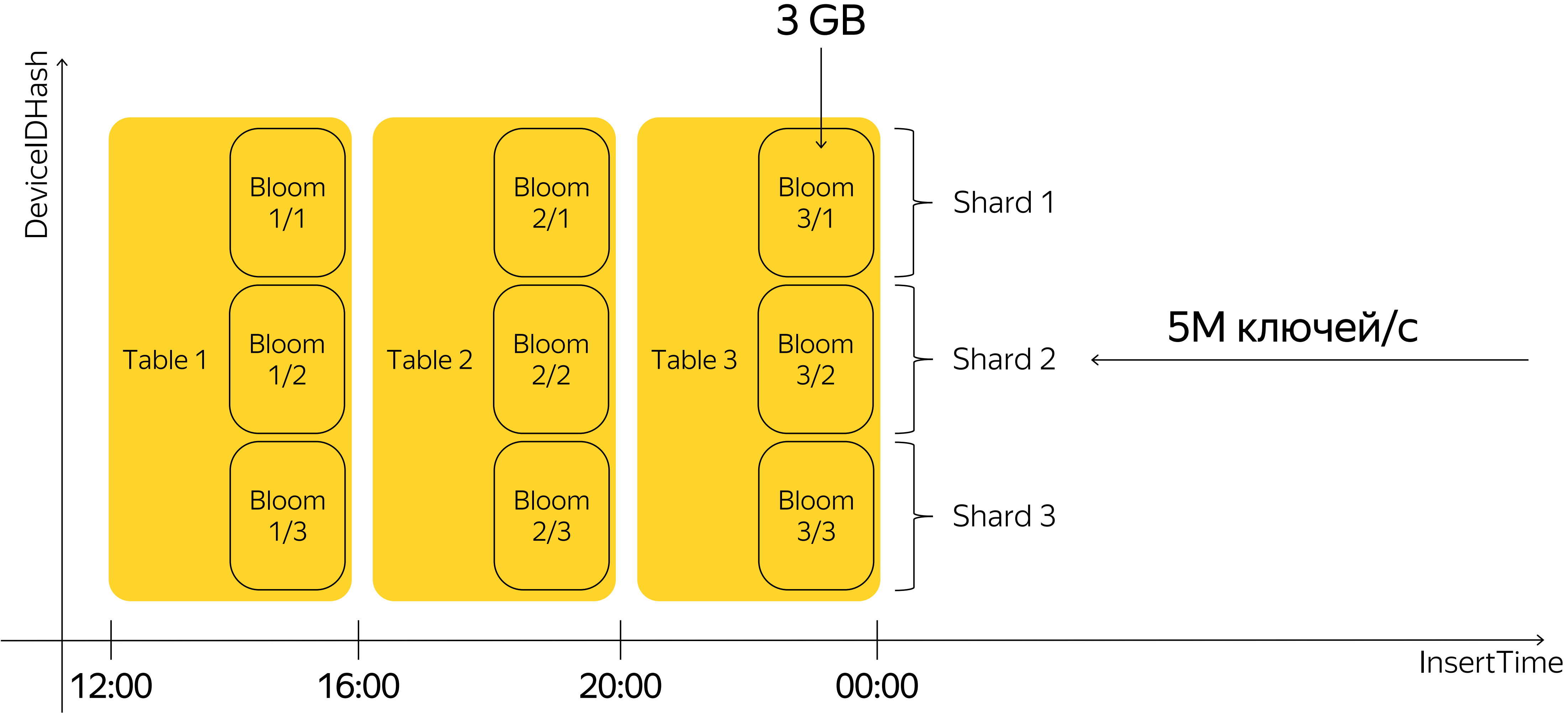


Что получилось

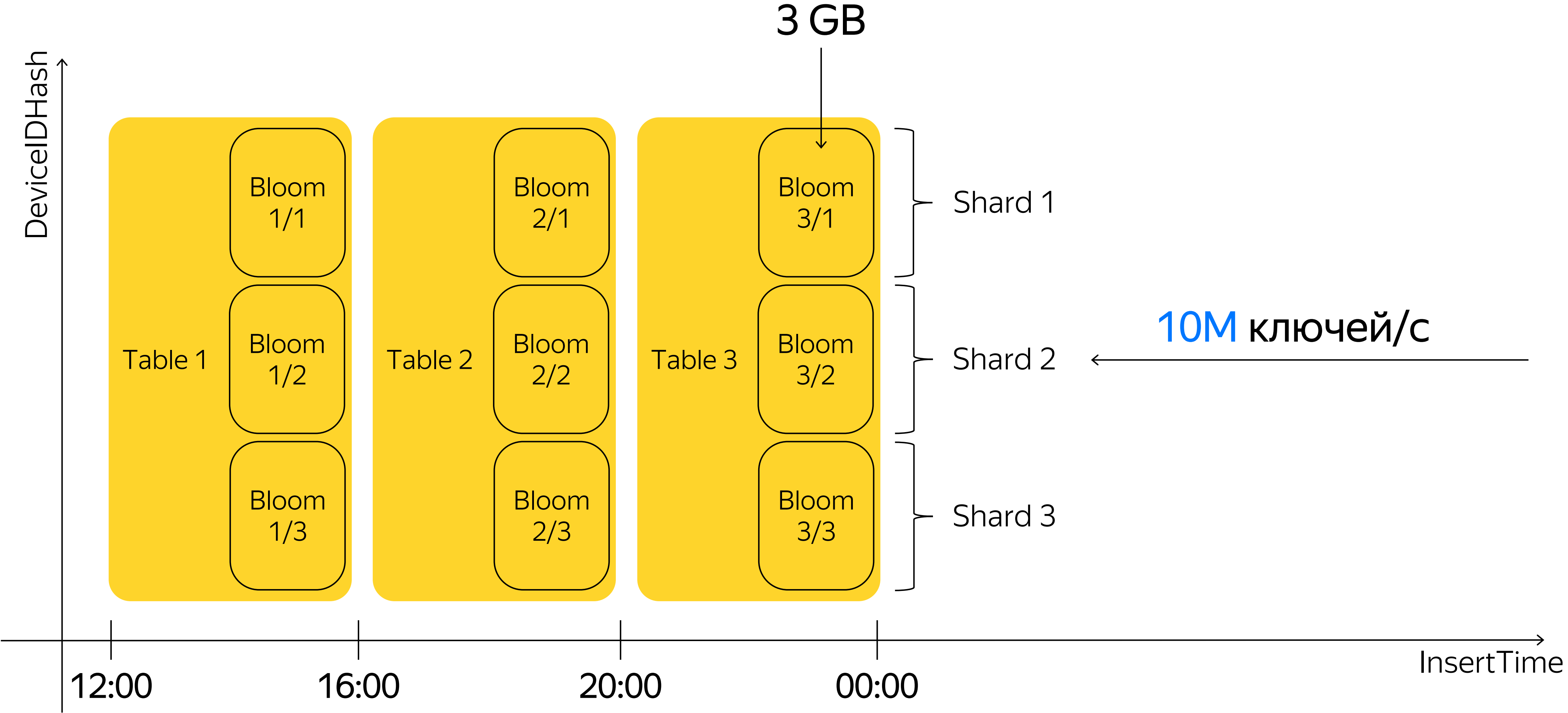


Таблицы хэшей – 5 TB
Таблицы фильтров – 600 GB

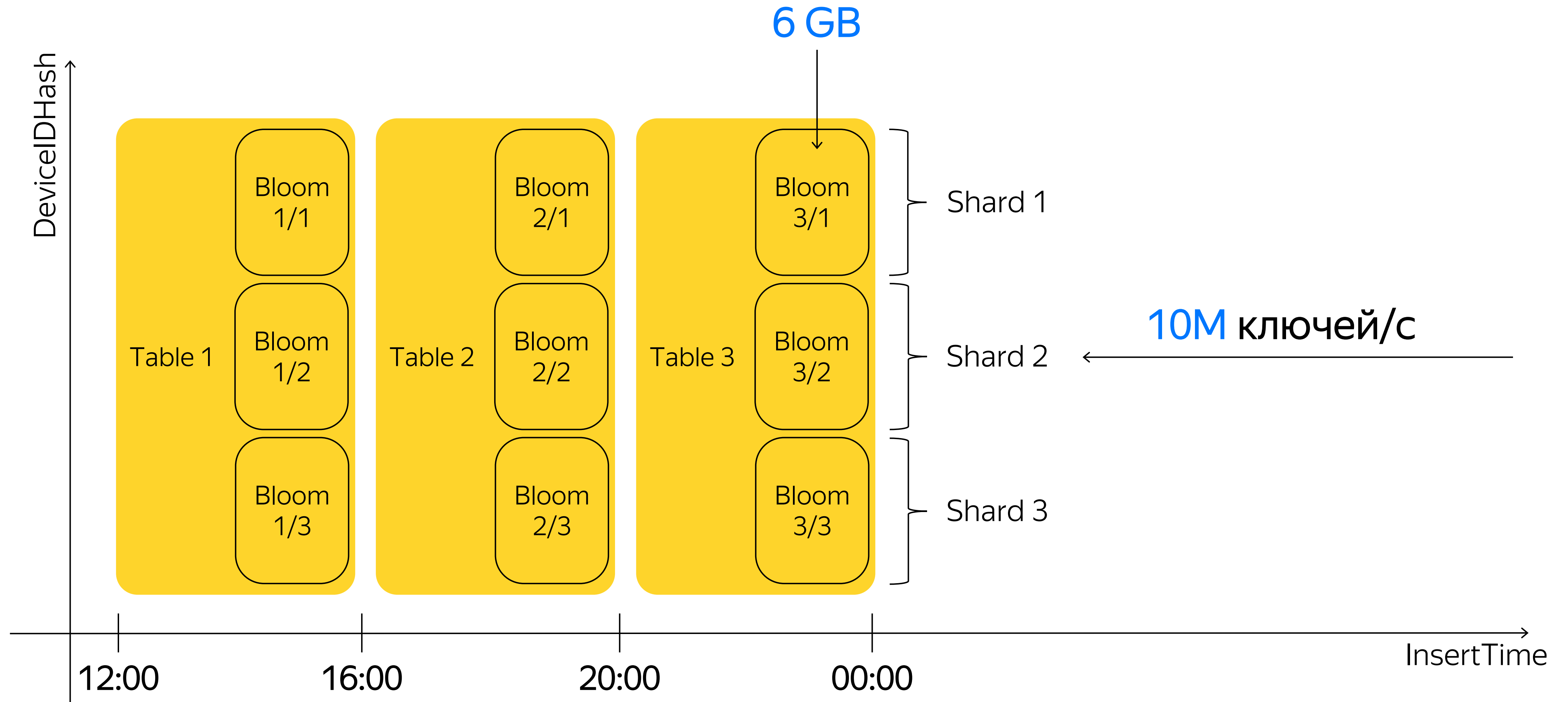
Масштабирование



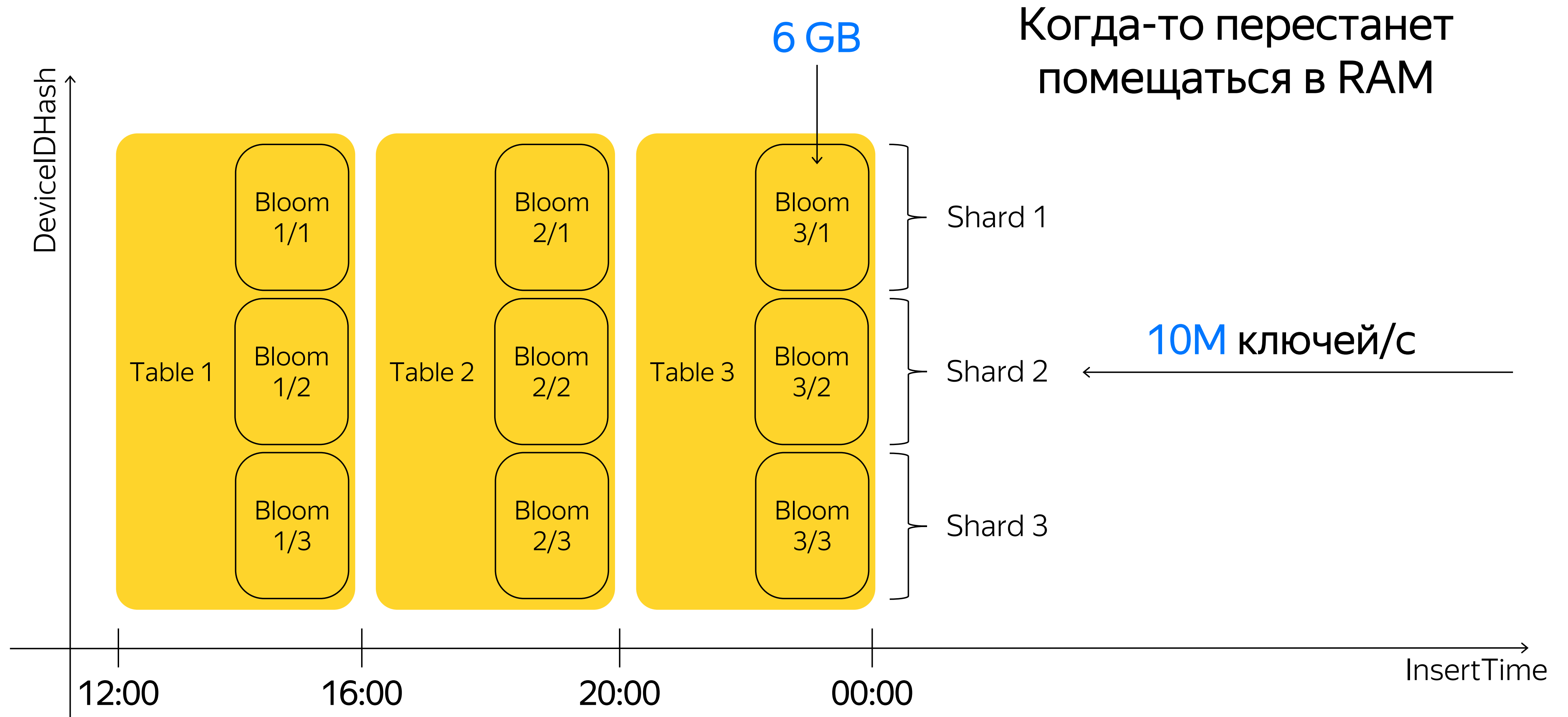
Масштабирование



Масштабирование

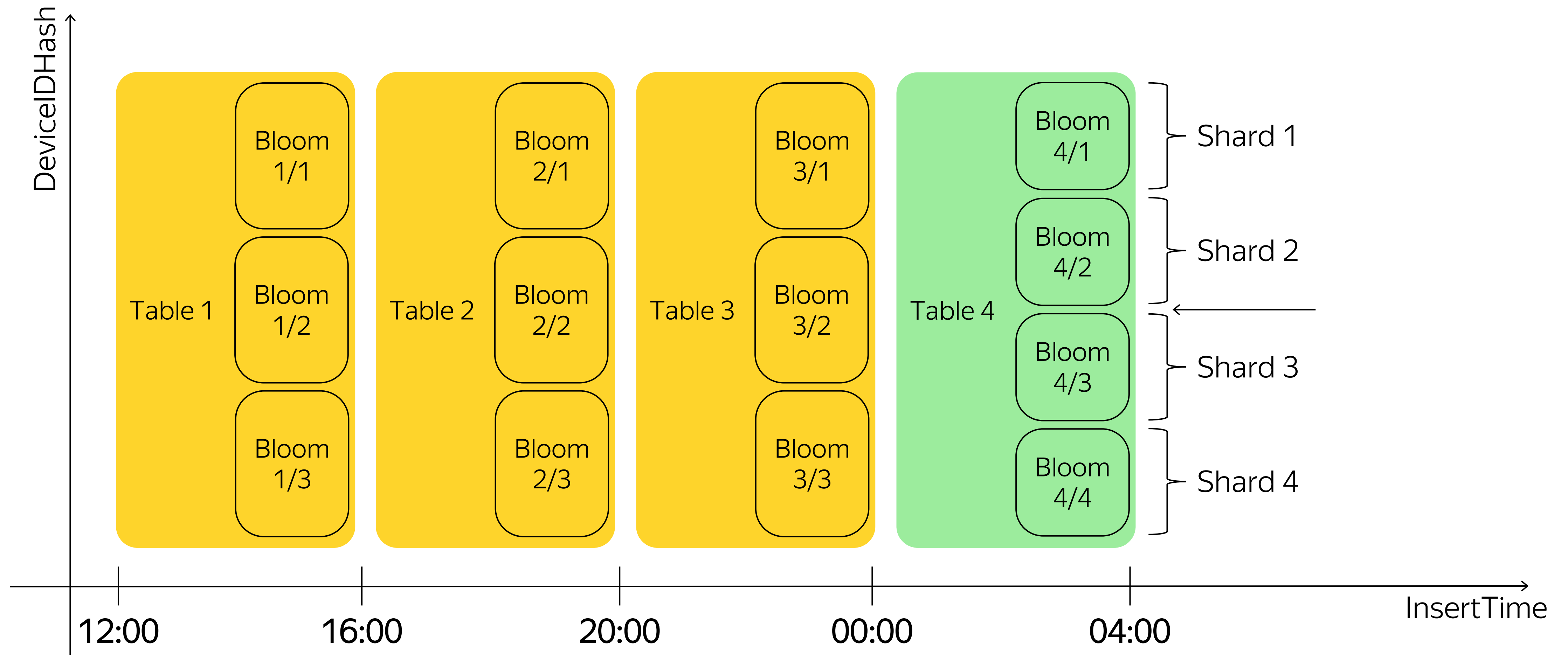


Масштабирование



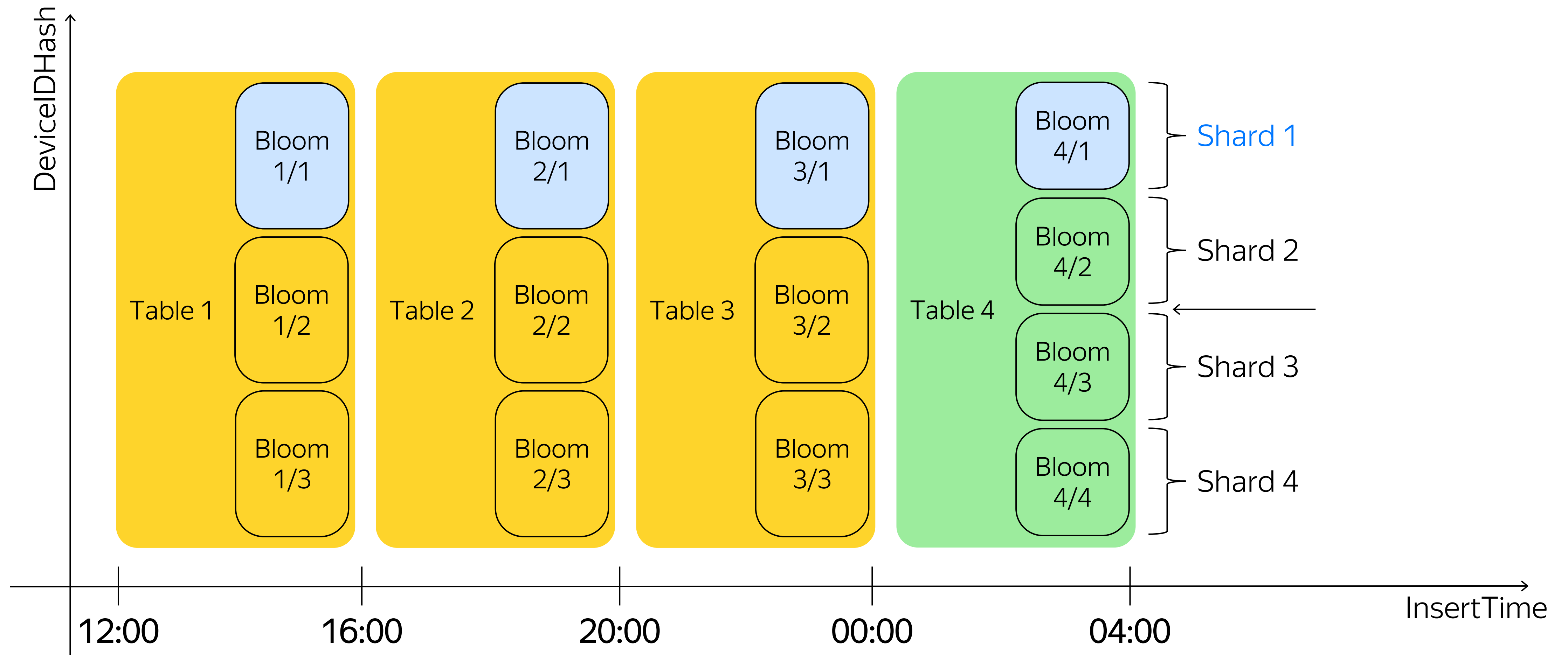
Перешардирование

Создаем новую таблицу с новым шардированием



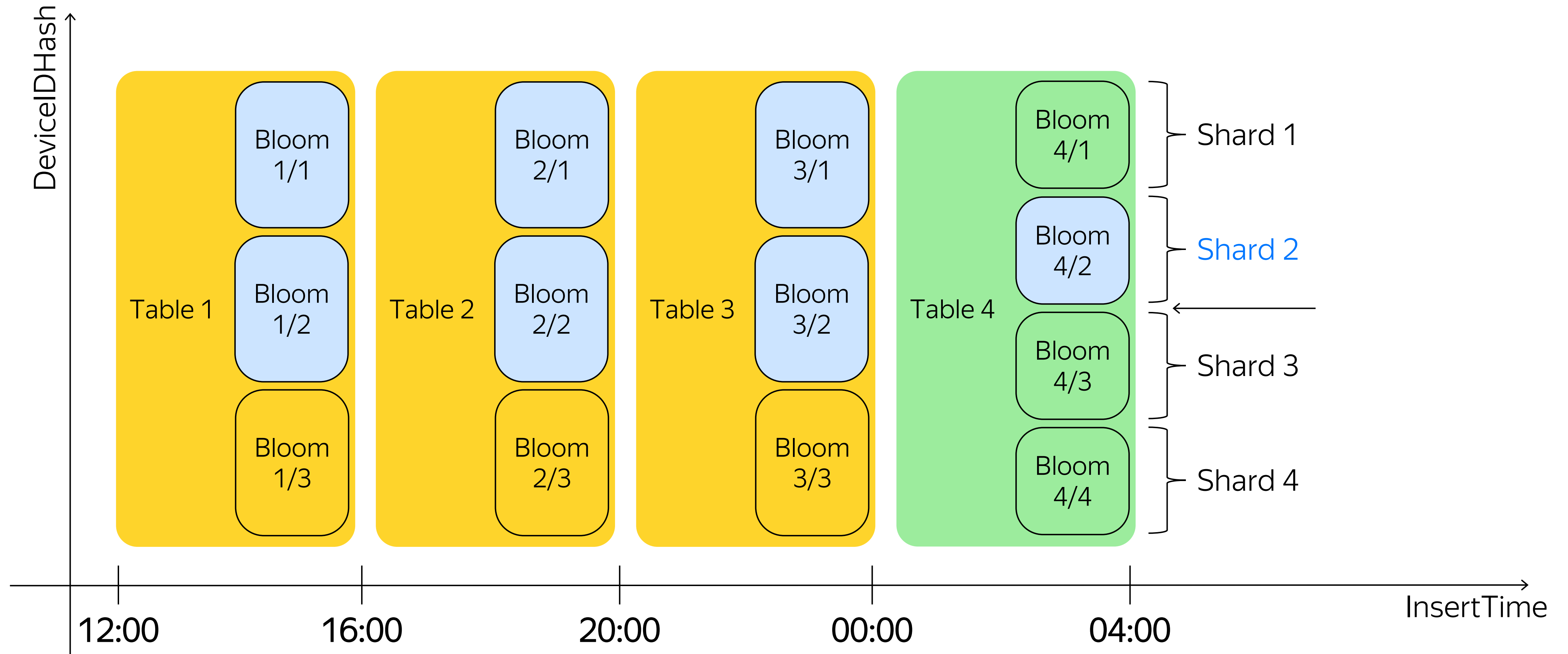
Перешардирование

Загружаем фильтры всех затронутых шардов



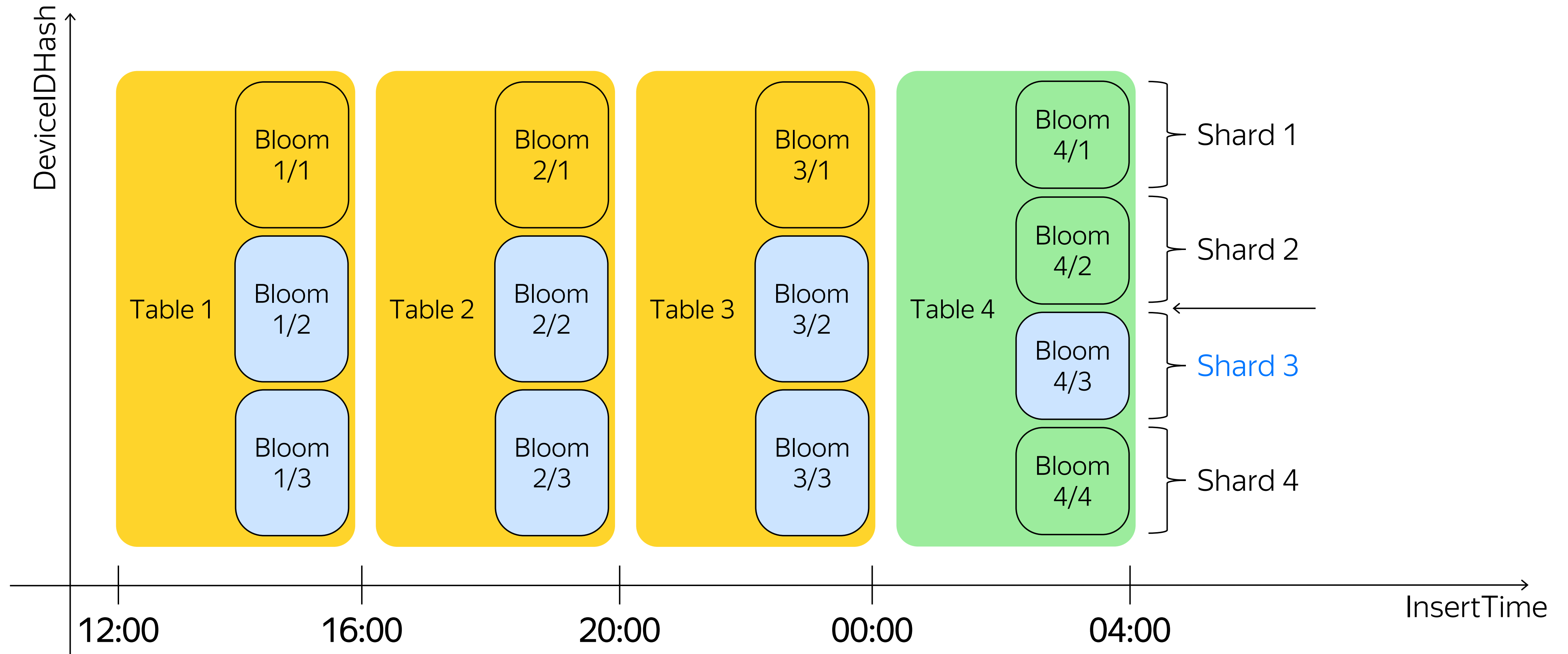
Перешардирование

Загружаем фильтры всех затронутых шардов



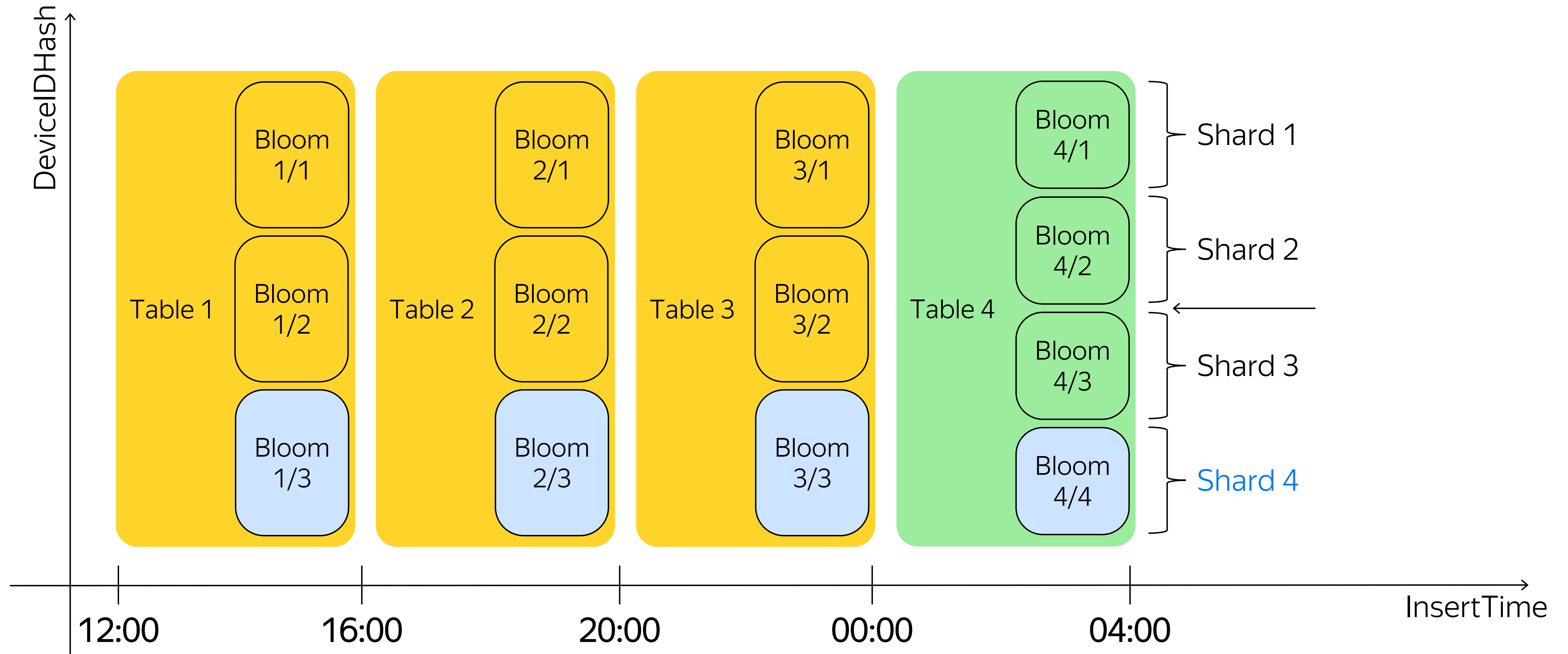
Перешардирование

Загружаем фильтры всех затронутых шардов



Перешардирование

Загружаем фильтры всех затронутых шардов



Еще одна схема работы

	Hash	
Event	123	

Еще одна схема работы

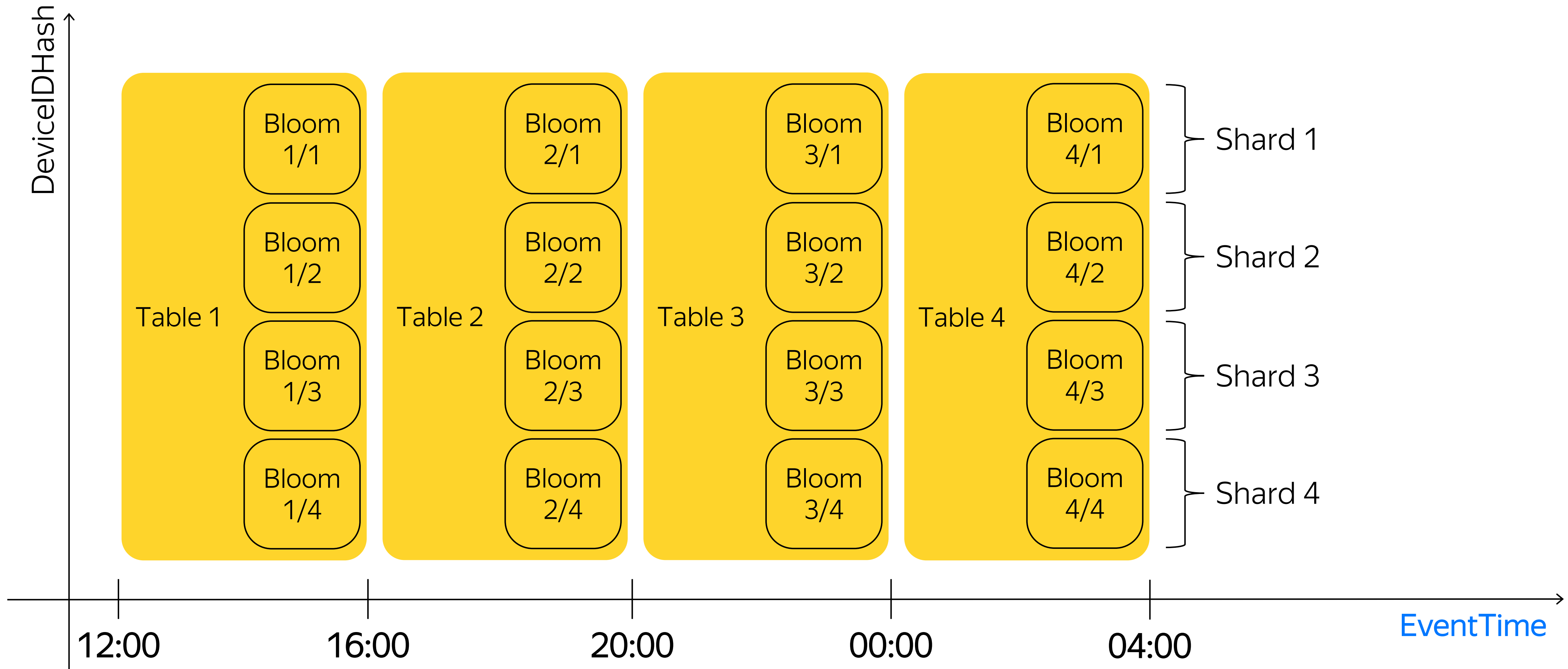
	Hash	EventTime
Event	123	2022-09-22 12:00:00

Еще одна схема работы

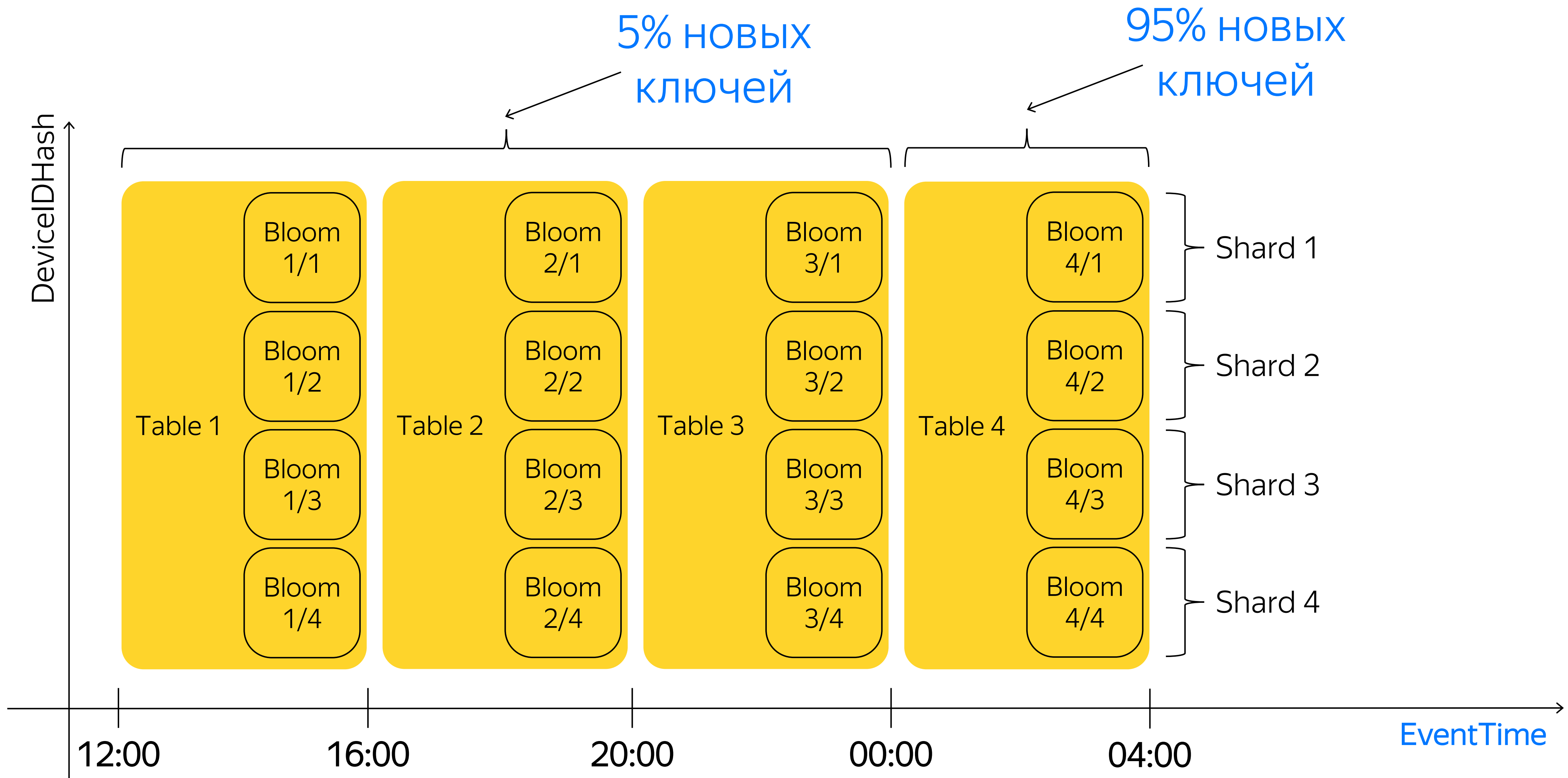
У дубля такое же,
как у оригинала

	Hash	EventTime
Event	123	2022-09-22 12:00:00

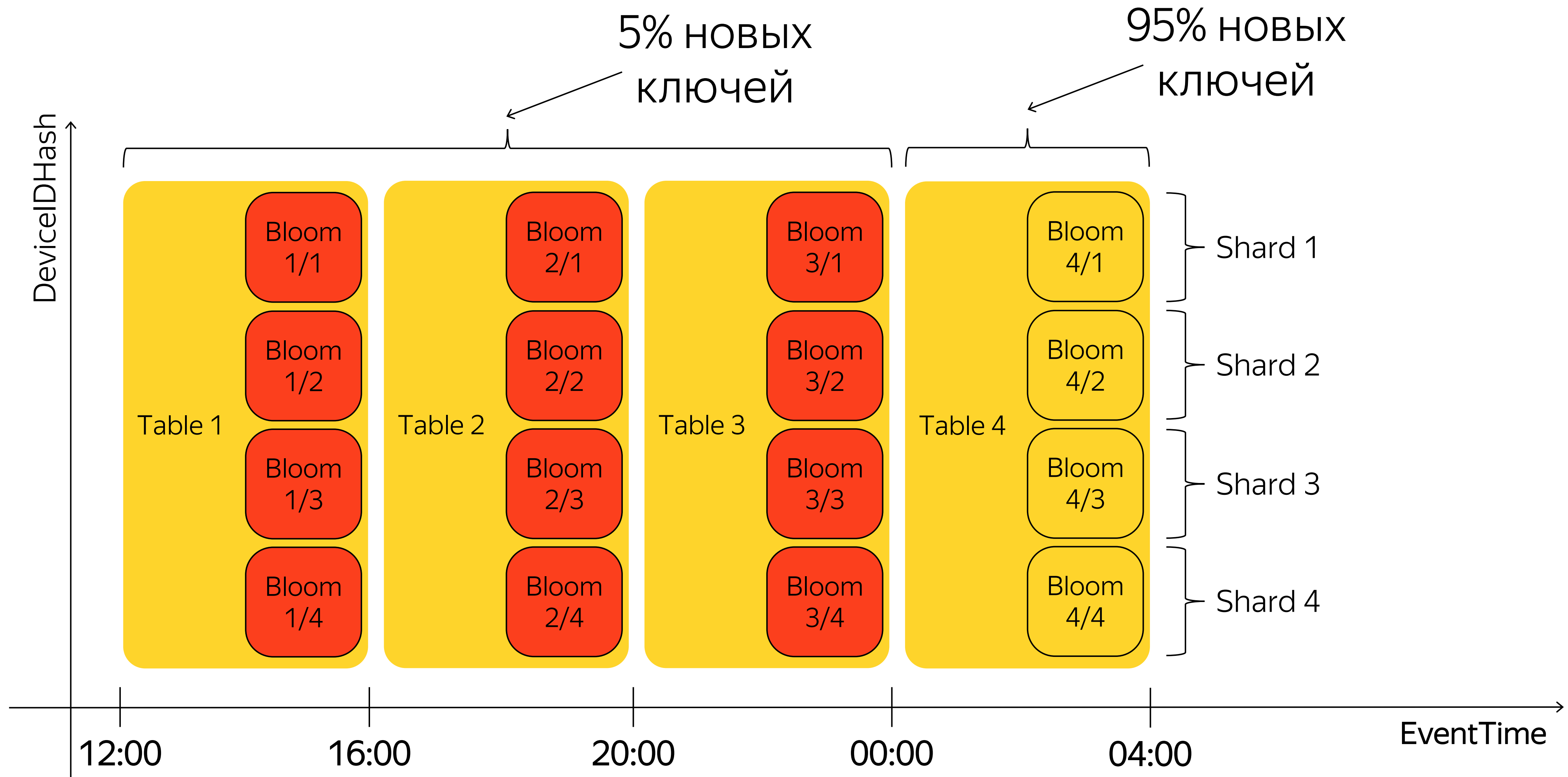
Еще одна схема работы



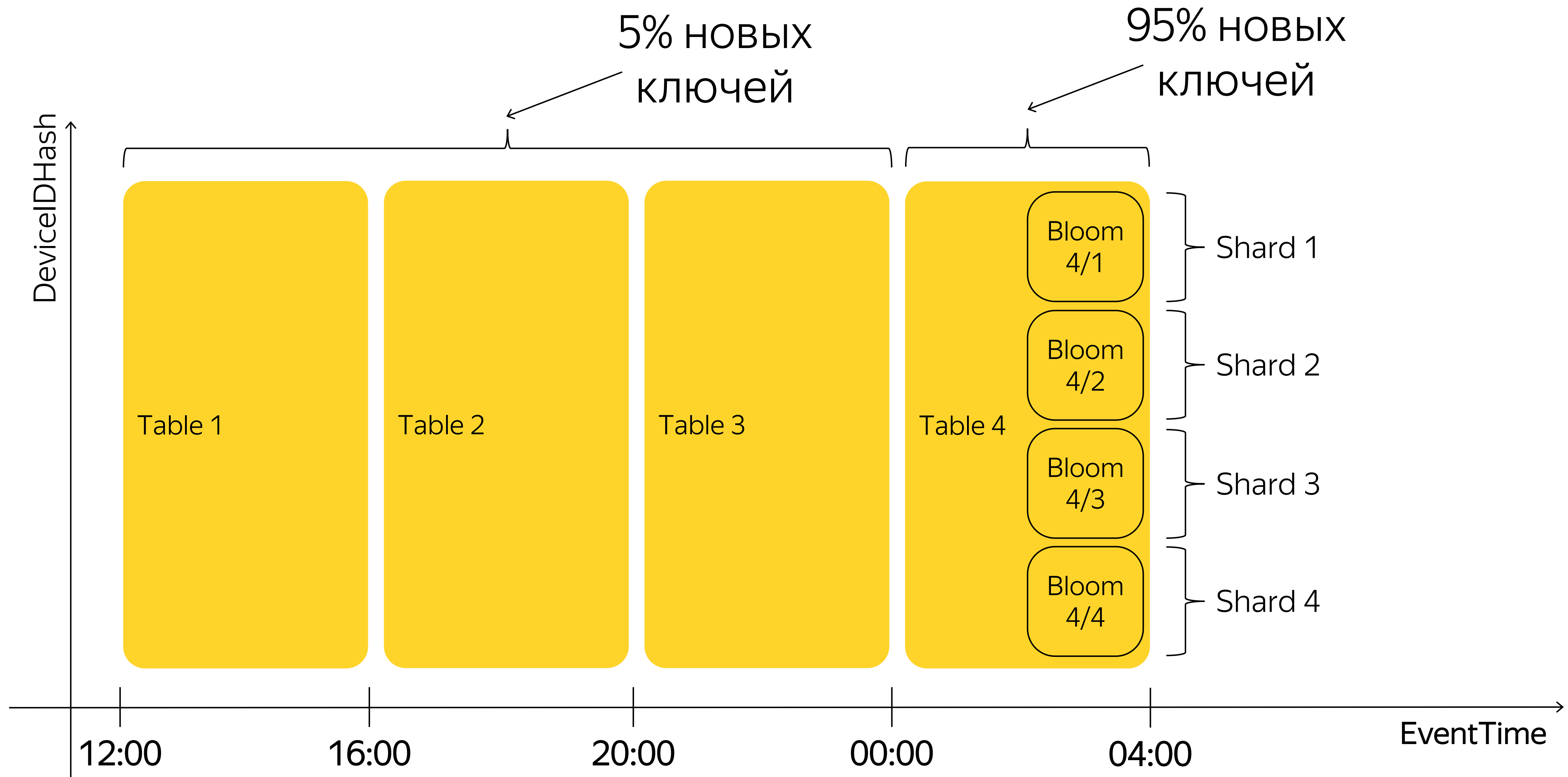
Еще одна схема работы



Еще одна схема работы



Еще одна схема работы





Оценить доклад



Спасибо

Артем Исмагилов

Разработчик

 anismagilov@yandex-team.ru

 [@artem_ismagilov](#)

Давайте обсудим:

- Как вы решаете проблему дедупликации?
- Применяете ли фильтры Блума?
Как добиваетесь их сохранности?
- Где еще можно применить такой подход с фильтрами Блума?