



Миграция на максималках: как поменять вообще всё

Василий Осадчий



Обо **мне**

Более 15 лет строю высоконагруженные системы:

- **iiko** - ресторанная ERP
- Одноклассники - **ok.ru**
- Запускал первую **Яндекс.Станцию**
- **EnelX** North America - зарядная инфраструктура для EV
- **PulsePoint** - AdTech в области здравоохранения

Сейчас создаю отказоустойчивые и масштабируемые финансовые сервисы для 01.tech

01.tech



- Делаем решения для Entertainment Industry с **2016 года**
- Растём на **5%** каждый месяц
- MAU **2M+**
- DAU **200K+**
- **10K+** RPS, **2K+** finance TPS
- AWS  on-prem



План

- [1] Проблемы, которые решаем
- [2] Схема переезда
- [3] Как не делать лишнюю работу
- [4] Миграция и репликация данных
- [5] Параллельный режим
- [6] Выводы



План

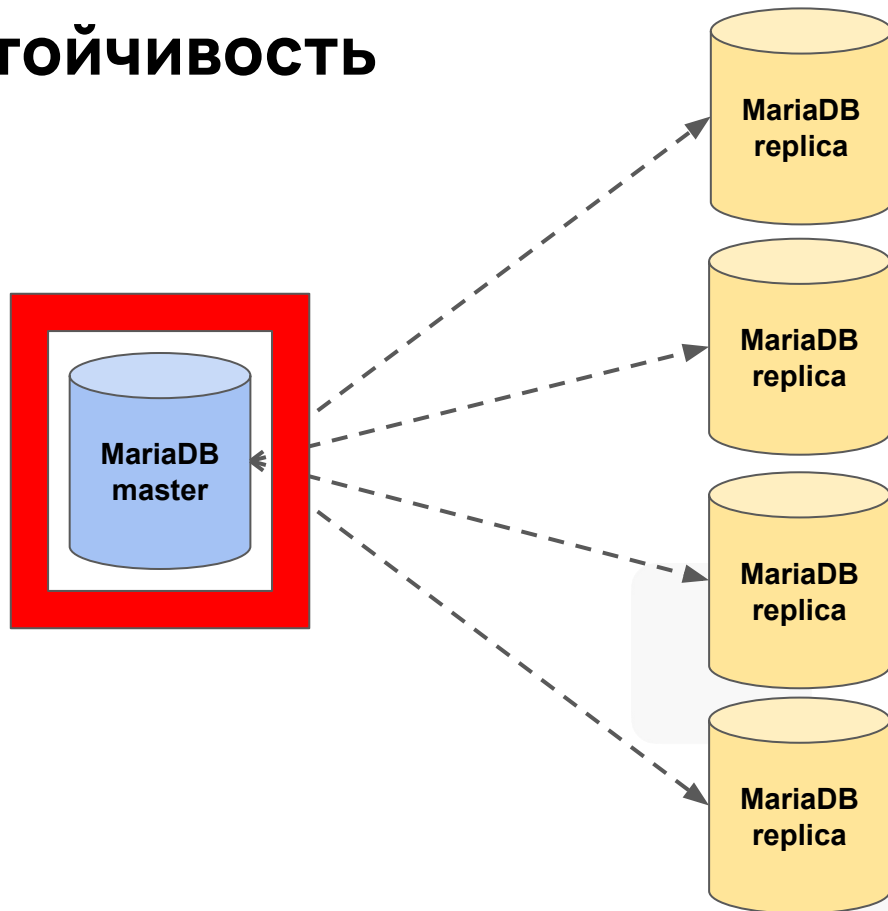
- [1] Проблемы, которые решаем
- [2] Схема переезда
- [3] Как не делать лишнюю работу
- [4] Миграция и репликация данных
- [5] Параллельный режим
- [6] Выводы



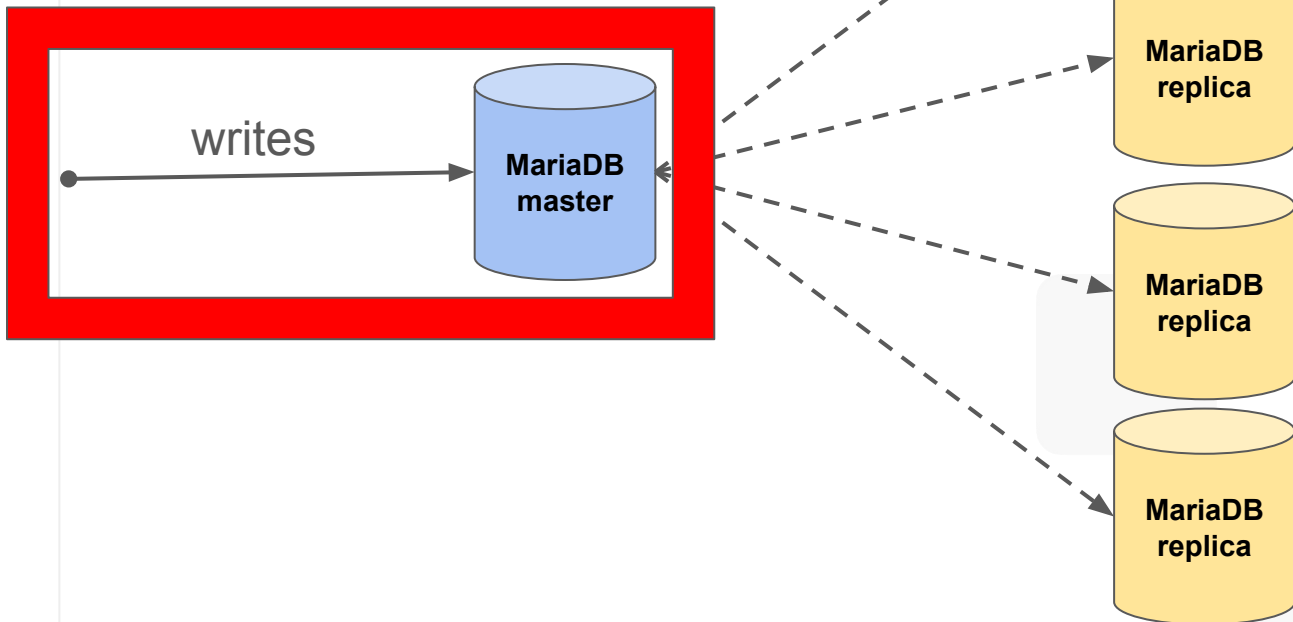
Сервис **finance-core**

- Транзакции и балансы пользователей
 - пополнения/выводы
 - операции в играх
- **~110 миллиардов** транзакций всего
- **~150 миллионов** новых транзакций в день (**1.5-3.5K** в секунду)
- Простой сервиса  простой всего бизнеса

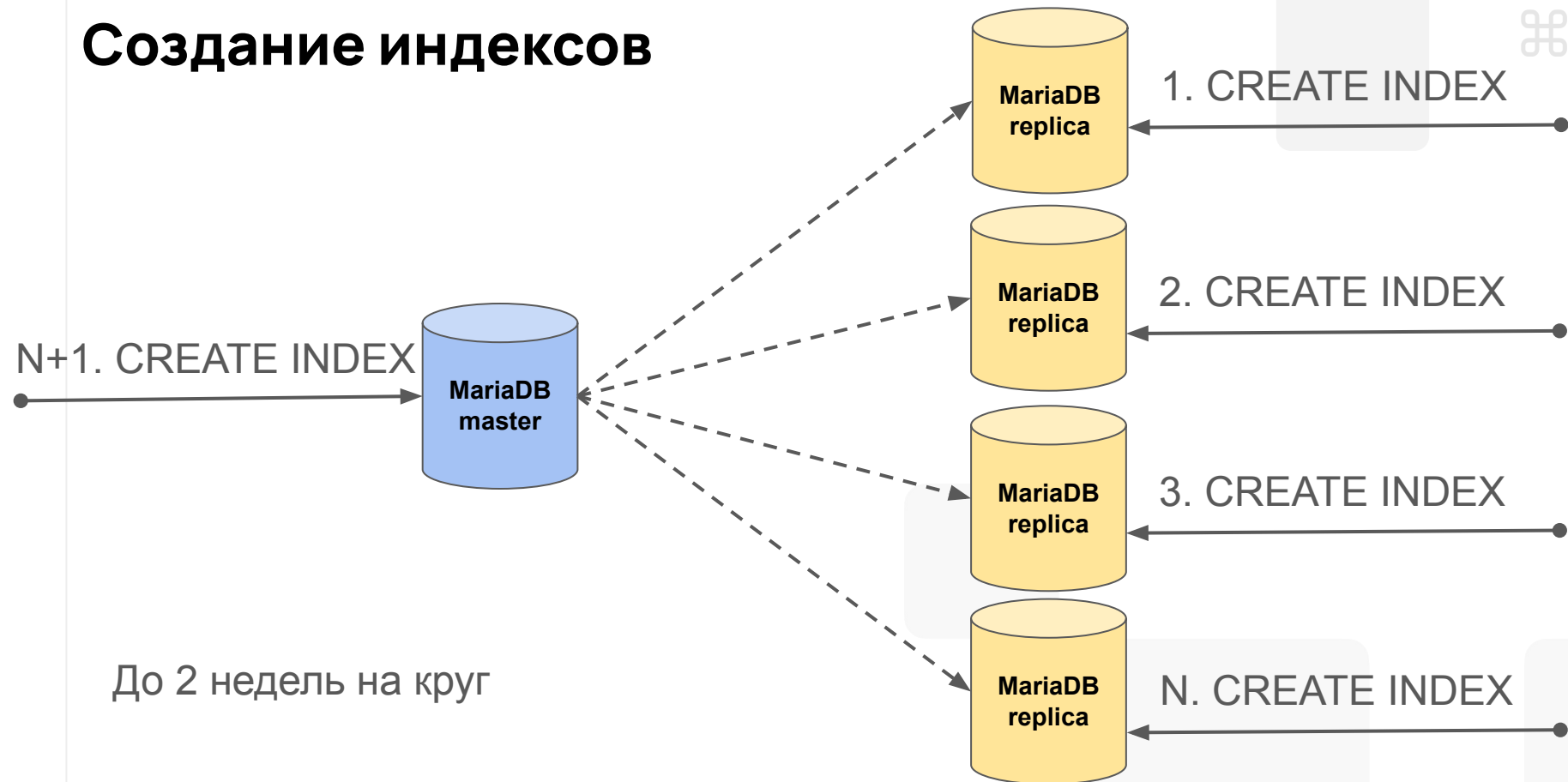
Отказоустойчивость



Масштабируемость



Создание индексов



Очень гибкий API



- SQL поверх JSON
- ~6 млн вариантов фильтров
- Индексов под все комбинации нет



Очень гибкий API

- SQL поверх JSON
- ~6 млн вариантов фильтров
- Индексов под все комбинации нет
- Пример: **deposits/getSumGroups**
 - groupByField
 - sumField
 - filter
 - 14 полей
 - 9 возможных условий (==, !=, IN, NOT IN, >, ...)



Проблемы бизнесовые

- Ограничение структуры данных для бизнеса
 - только счета пользователей
- Транзакции живут в 4 разных таблицах
 - функционал не всегда добавлялся во все 4 таблицы



Проблемы, **которые мы решаем**

[1] Отказоустойчивость

БД у текущего finance-core - MariaDB с единым master и read replicas

[2] Масштабируемость

Тоже упираемся в БД

[3] Сложность эксплуатации

5 таблиц в 2 разных базах со сложной схемой перелива данных
Создание индекса в БД микротранзакций занимает недели

[4] Ограничения структуры данных для бизнеса

Сейчас есть только счета пользователей - этого мало



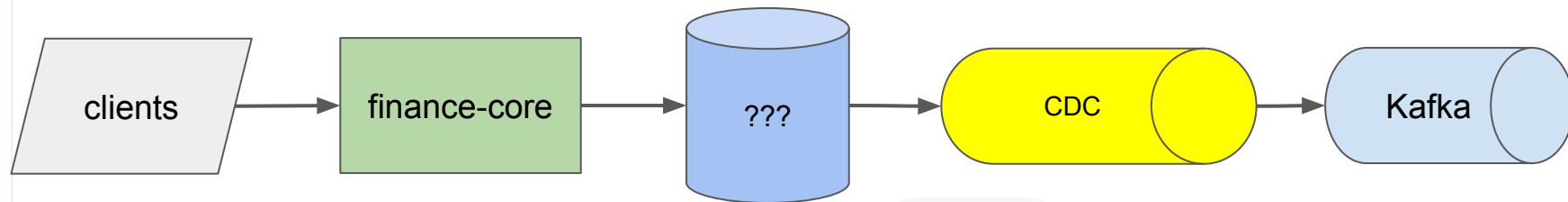
ЧЕ ДЕЛАТЬ



План

- [1] Проблемы, которые решаем
- [2] **Схема переезда**
- [3] Как не делать лишнюю работу
- [4] Миграция и репликация данных
- [5] Параллельный режим
- [6] Выводы

Целевая архитектура



YugabyteDB



- “Распределённый PostgreSQL”
- Диалект SQL обратнoсовместим с PostgreSQL
- Легко горизонтально масштабируется
- Хорошо переживает отказы нoд
- Open-source

<https://smartdataconf.ru/archive/2025/talks/6befb0ab6e614607805fb0e188ae2c11/>

А ЧТО ЕСЛИ Я СКАЖУ ТЕБЕ, ЧТО

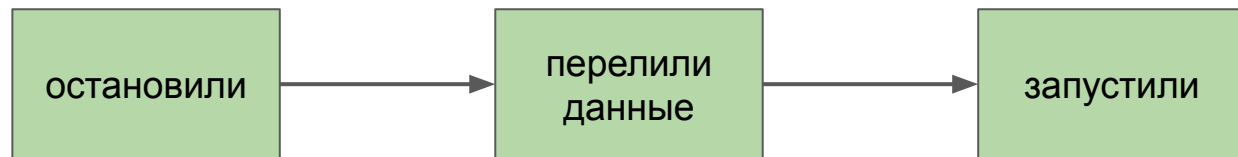


**ЭТО ЕЩЕ НЕ КОНЕЦ, ЭТО ТОЛЬКО
НАЧАЛО!**

Альтернативные **схемы**



С downtime:

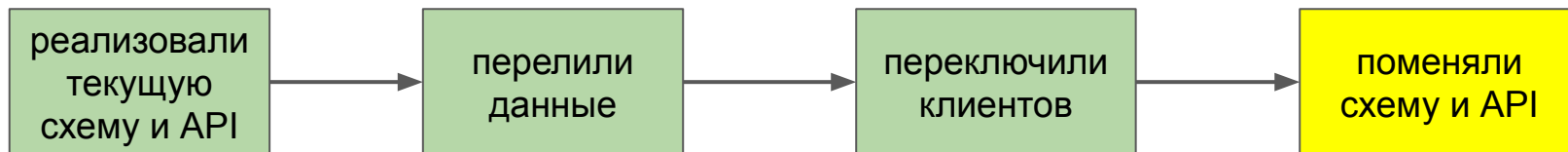


- Очень дорогой простой!
- Если накосячим, то эффект на пользователей
- Что делать с клиентами?

Альтернативные **схемы**



Поэтапно:

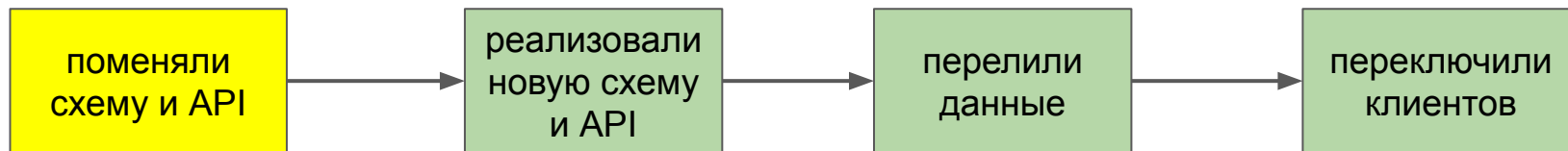


- Смена схемы и API будет как отдельная миграция
- Много работы на выброс

Альтернативные **схемы**



Поэтапно:



- Не вся команда знакома с NodeJS
- Можем не успеть - кончатся диски в старой БД



Требования к миграции

[1] Отсутствие downtime при миграции

Очень дорогой простой сервиса

[2] Гарантии, что новая система работает так же как и старая

Тех. долг + большое кол-во клиентов = сложно уверенно сказать, как оно должно работать для каждого случая

[3] Возможность отката на старую систему в случае проблем

Возможность экспериментировать не боясь положить прод
ОЧЕНЬ много чего меняем при переезде



Требования к миграции

[4] Надо поменять API не меняя код на стороне клиентов

Старый API чрезмерно гибок (SQL переложенный на JSON) + смена схемы БД обязывает

Клиентов много - долго ждать пока все на своей стороне сделают изменения

[5] Нулевая терпимость к потере данных

Это же деньги!

Схема переезда

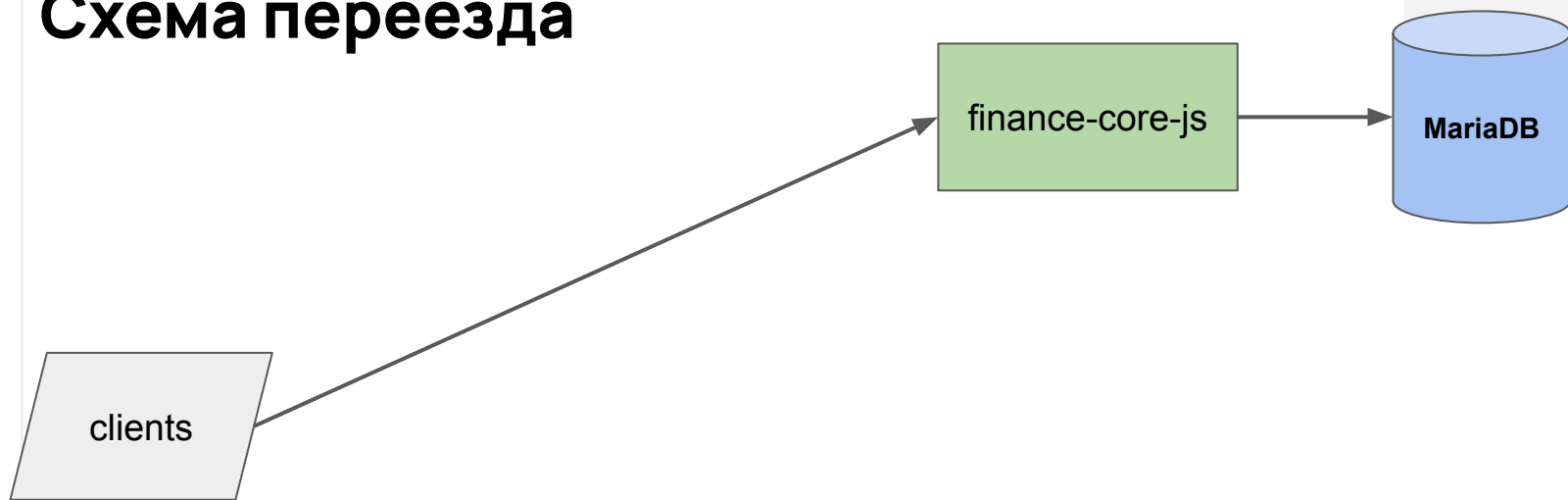


Схема переезда



Смена URL

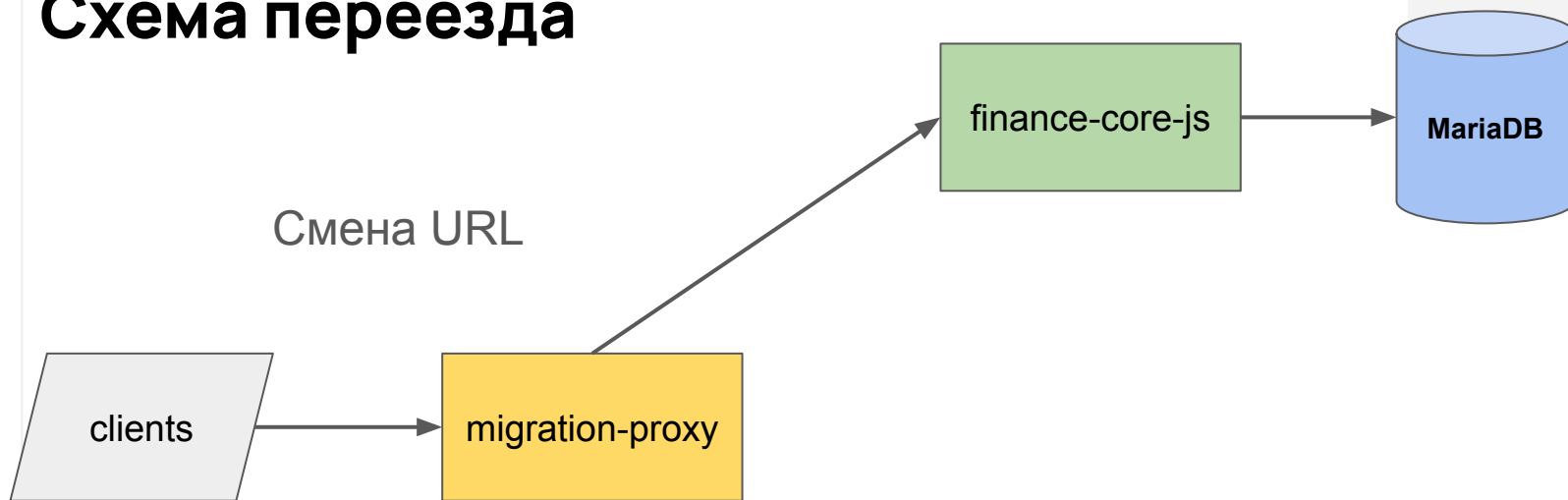


Схема переезда

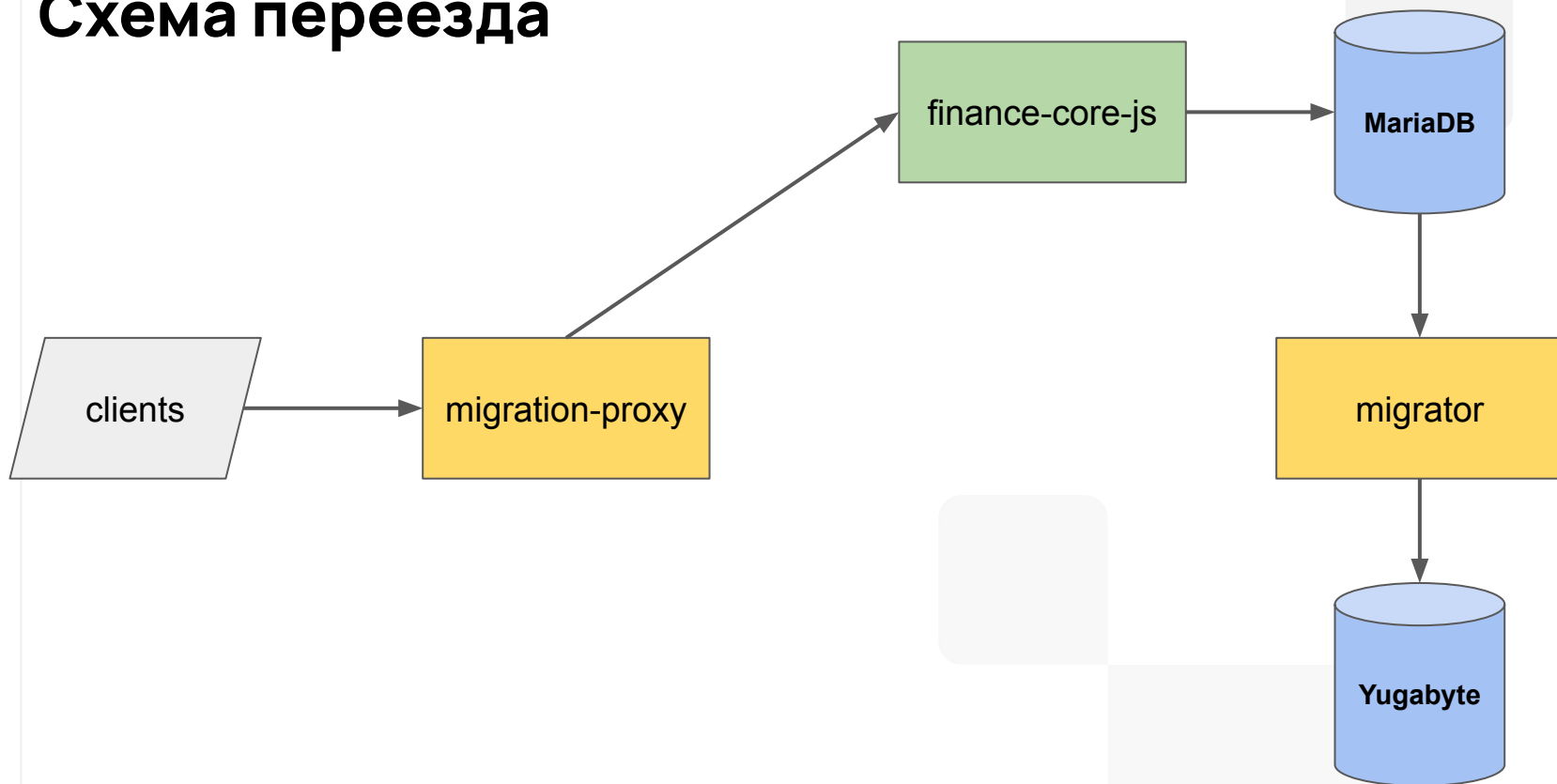


Схема переезда

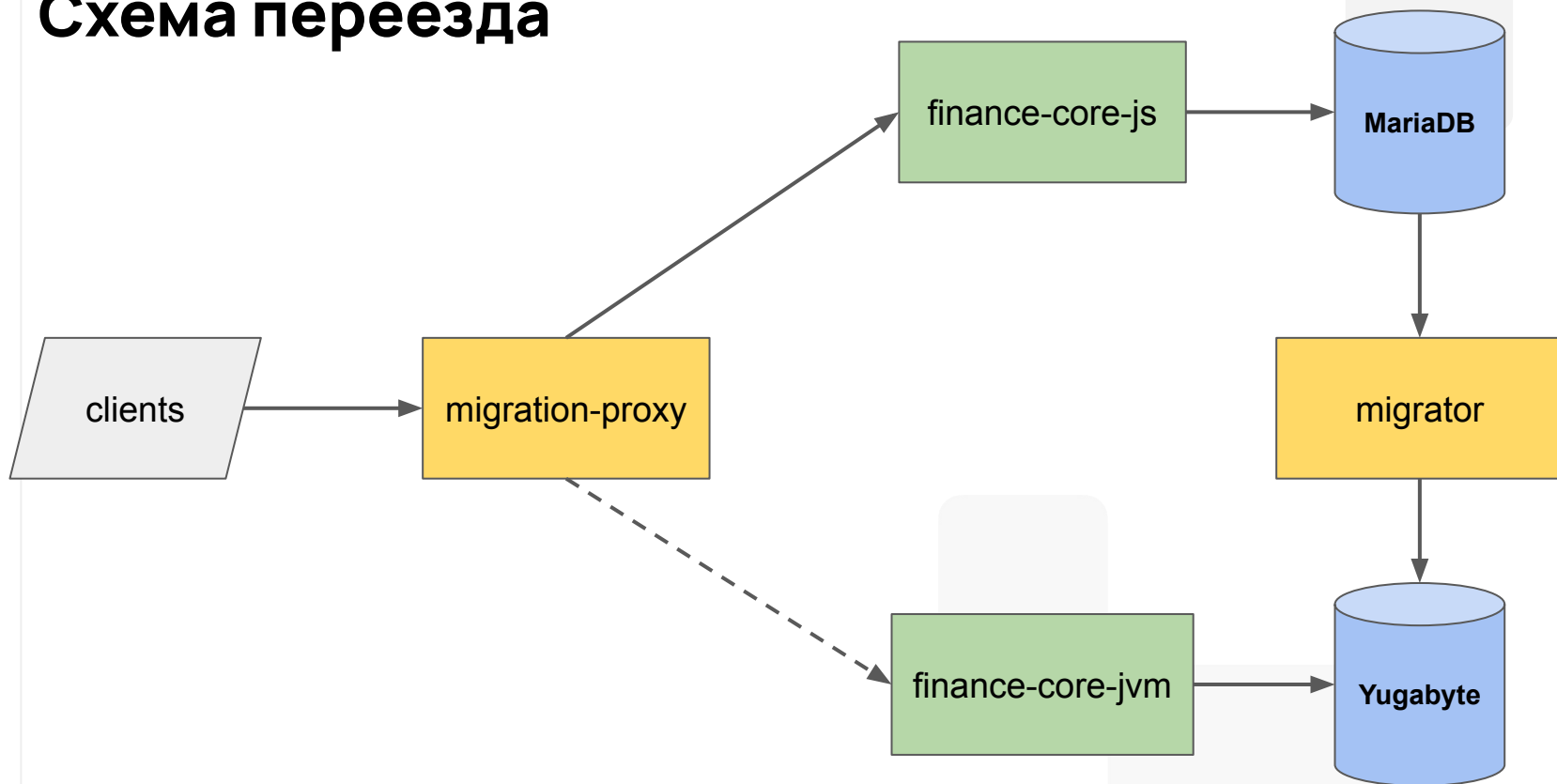


Схема переезда

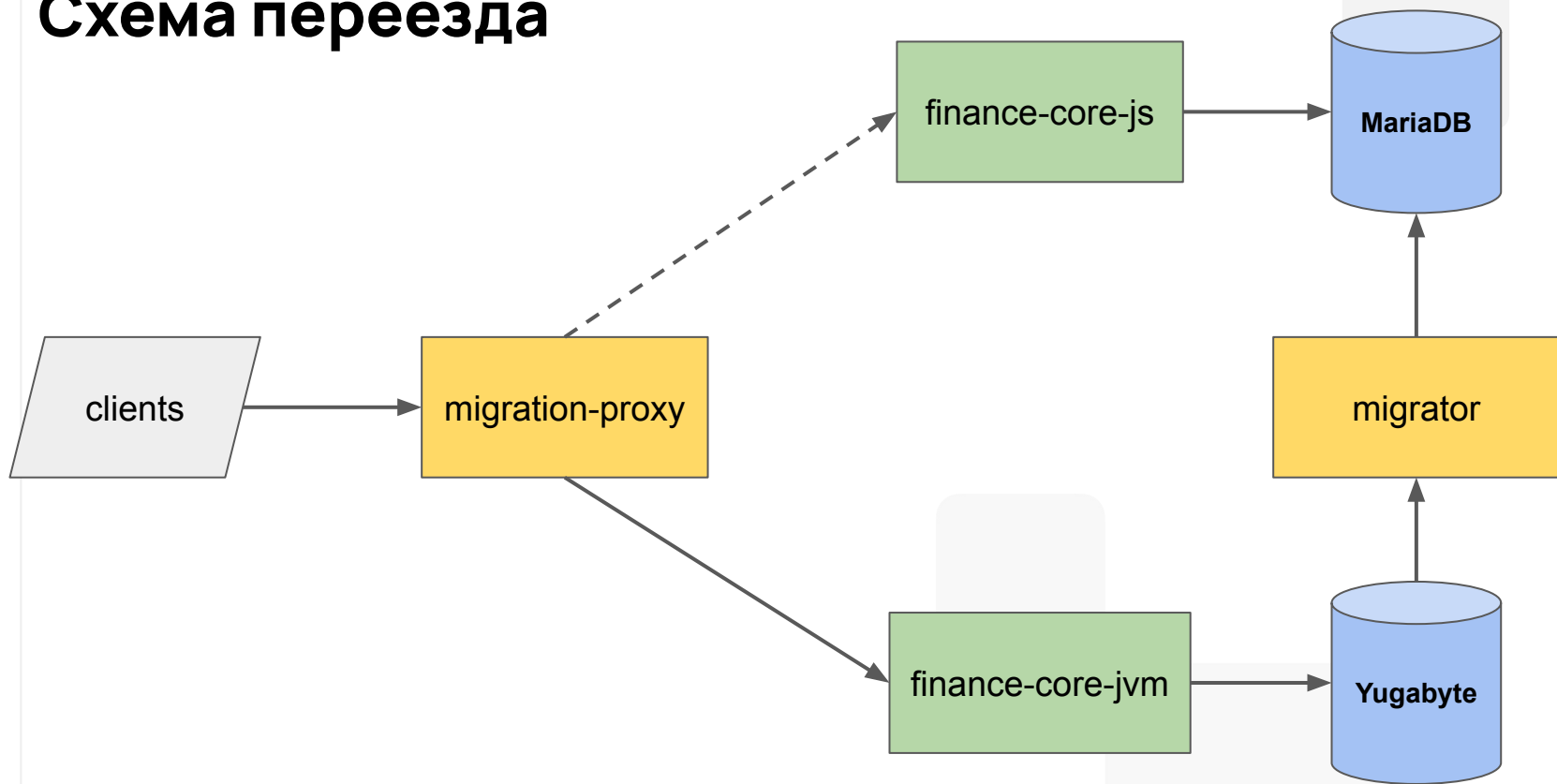


Схема переезда

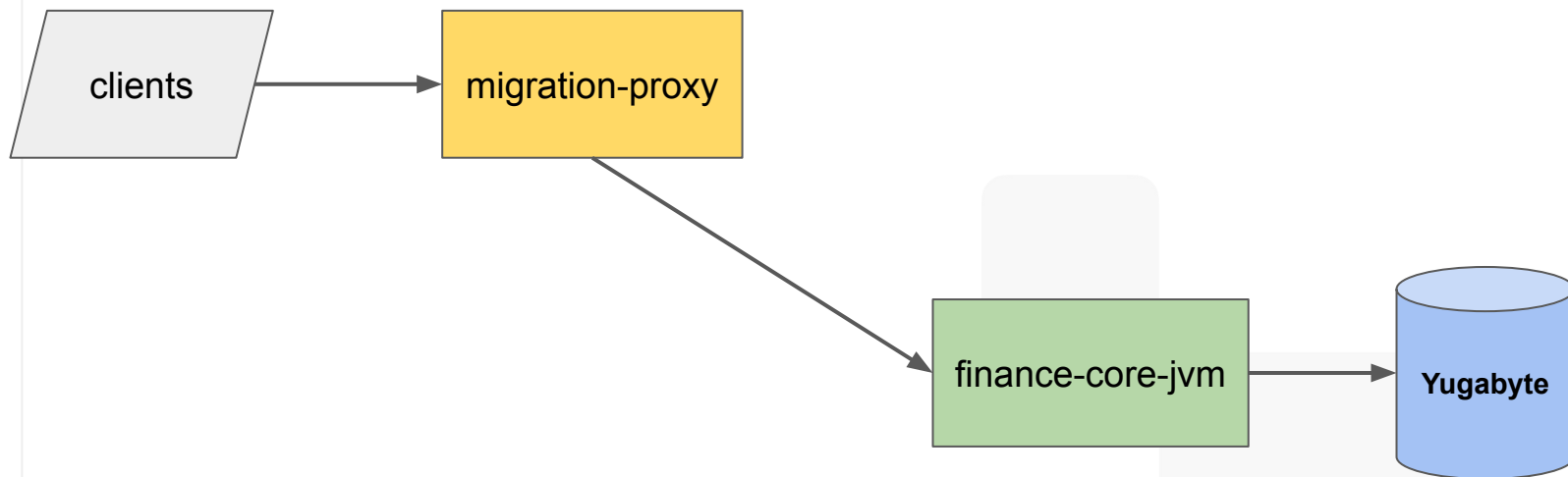
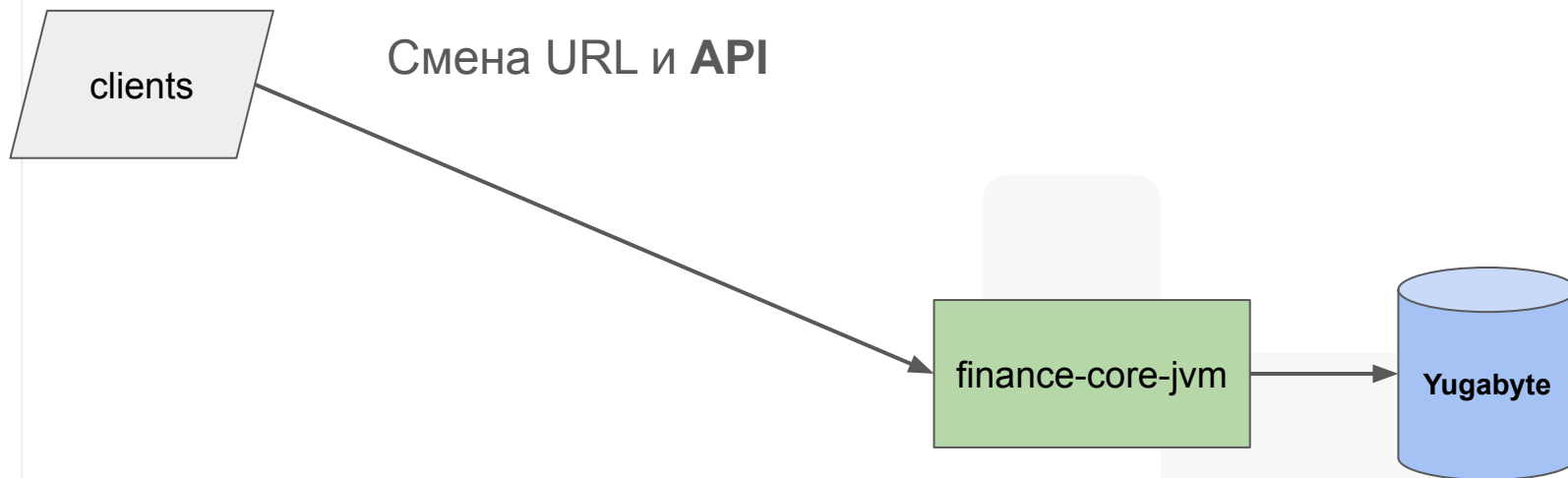


Схема переезда



Переезд -



Это всегда весело



План

- [1] Проблемы, которые решаем
- [2] Схема переезда
- [3] **Как не делать лишнюю работу**
- [4] Миграция и репликация данных
- [5] Параллельный режим
- [6] Выводы

Отсекаем лишнее



- Проблема: ~6 млн вариантов фильтров в API



Отсекаем лишнее

- Проблема: ~6 млн вариантов фильтров в API
- **А что если не всё это богатство API используется?**



Отсекаем лишнее

- Проблема: ~6 млн вариантов фильтров в API
- **А что если не всё это богатство API используется?**
- Сделали запись запросов-ответов старой системы в Kafka



Отсекаем лишнее

- Проблема: ~6 млн вариантов фильтров в API
- **А что если не всё это богатство API используется?**
- Сделали запись запросов-ответов старой системы в Kafka
- Из Kafka затыгиваем данные в ClickHouse с помощью KafkaEngine
 - без опыта получилось настроить за ~4 часа



Отсекаем лишнее

- Проблема: ~6 млн вариантов фильтров в API
- **А что если не всё это богатство API используется?**
- Сделали запись запросов-ответов старой системы в Kafka
- Из Kafka затыгиваем данные в ClickHouse с помощью KafkaEngine
 - без опыта получилось настроить за ~4 часа
- Можно удобно анализировать логи!



Анализируем логи

- masked версия запросов для группировки

```
{  
  "additionalOptions":"###",  
  "caller":"###",  
  "data":{"userId":"###"},  
  "tenantId":"###"  
}
```

Анализируем логи



- Теоретическое кол-во комбинаций - **~6млн!**
 - индексов подо все построить нереально!



Анализируем логи

- Теоретическое кол-во комбинаций - **~6млн!**
 - индексов подо все построить нереально!
- Реальное кол-во комбинаций - **136!**
 - с этим уже можно работать
 - смогли построить вменяемые индексы

Анализируем логи



- Местами смогли упростить и сделать строже API

Анализируем логи



- Местами смогли упростить и сделать строже API
 - nullable → non-null

Анализируем логи



- Местами смогли упростить и сделать строже API
 - nullable → non-null
 - поля свободного формата → строгий формат



Анализируем логи

- Местами смогли упростить и сделать строже API
 - nullable → non-null
 - поля свободного формата → строгий формат
 - нашли проблемные запросы клиентов и попросили поправить - некоторые из них давали ответ в **20MB!**



Simplify

$$(a - x)(b - x)(c - x) \dots (z - x)$$

Before you realize it



After you realize it





План

- [1] Проблемы, которые решаем
- [2] Схема переезда
- [3] Как не делать лишнюю работу
- [4] **Миграция и репликация данных**
- [5] Параллельный режим
- [6] Выводы

Вспомним **требования** к миграции

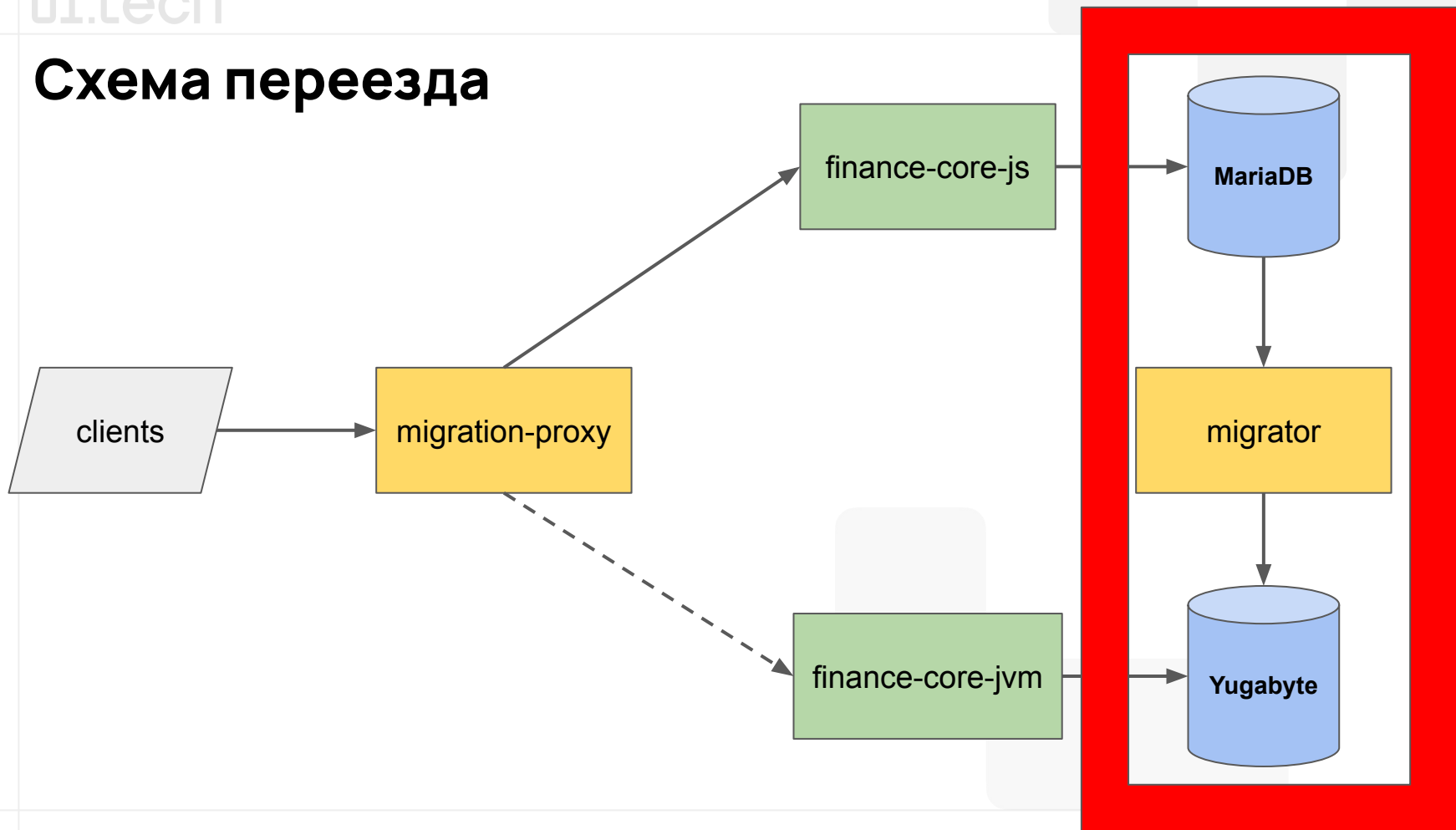


[5] Нулевая терпимость к потере данных

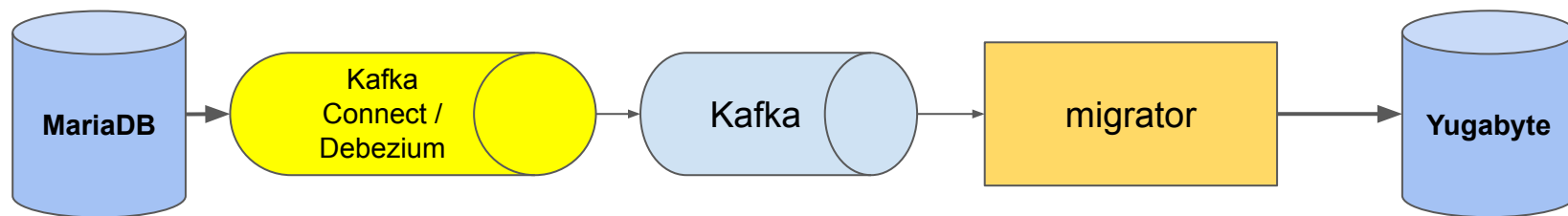
Это же деньги!

⇒ Строим *pipeline* с гарантиями, что данные не теряются

Схема переезда



Streaming **НОВЫХ** данных



Streaming **НОВЫХ** данных



- Debezium - читает binlog-и/WAL-ы БД
 - даёт гарантии, что не будет пропусков данных

Streaming **новых данных**



- Debezium - читает binlog-и/WAL-ы БД
 - даёт гарантии, что не будет пропусков данных
- Kafka
 - надёжное хранилище событий
 - развязывает producer-ов и consumer-ов

Запускаем streaming **НОВЫХ ДАННЫХ**



- Всё просто и надёжно

Запускаем streaming **НОВЫХ ДАННЫХ**



- Всё просто и надёжно
- Быстро реализовали, настроили, запустили

Запускаем streaming **НОВЫХ** данных



- Всё просто и надёжно
- Быстро реализовали, настроили, запустили
- **НО данные по факту теряются!**

Запускаем streaming **НОВЫХ ДАННЫХ**



- Всё просто и надёжно
- Быстро реализовали, настроили, запустили
- **НО данные по факту теряются!**
 - копаем - оказалось дело в обработке ошибок

Настройки @KafkaListener



- В kafka мире default-ы часто настроены на то чтобы пропускать сообщения при проблемах

Настройки @KafkaListener



- В kafka мире default-ы часто настроены на то чтобы пропускать сообщения при проблемах
- Нам нужно поведение “умри, но доставь”



Настройки @KafkaListener

- В kafka мире default-ы часто настроены на то чтобы пропускать сообщения при проблемах
- Нам нужно поведение “умри, но доставь”
 - **errorHandler** для *ConcurrentKafkaListenerContainerFactory* с бесконечными попытками

Настройки **KafkaListener**



```
open fun createKafkaErrorHandler(kafkaErrorHandlerProperties: KafkaErrorHandlerProperties): DefaultErrorHandler {  
    val bo = ExponentialBackOff().apply {  
        initialInterval = kafkaErrorHandlerProperties.initialInterval.toMillis()  
        multiplier = kafkaErrorHandlerProperties.multiplier  
        maxInterval = kafkaErrorHandlerProperties.maxInterval.toMillis()  
        maxAttempts = kafkaErrorHandlerProperties.maxAttempts  
    }  
  
    return DefaultErrorHandler(retryingKafkaConsumerRecordRecoverer, bo).apply {  
        setRetryListeners(retryListener)  
    }  
}
```



Настройки @KafkaListener

- В kafka мире default-ы часто настроены на то чтобы пропускать сообщения при проблемах
- Нам нужно поведение “умри, но доставь”
 - errorHandler для *ConcurrentKafkaListenerContainerFactory* с бесконечными попытками
 - задаём **ConsumerRecordRecoverer**, который защищает от пропусков при внутренних ошибках KafkaListener-а

Настройки **KafkaListener**



```
open fun createKafkaErrorHandler(kafkaErrorHandlerProperties: KafkaErrorHandlerProperties): DefaultErrorHandler {  
    val bo = ExponentialBackOff().apply {  
        initialInterval = kafkaErrorHandlerProperties.initialInterval.toMillis()  
        multiplier = kafkaErrorHandlerProperties.multiplier  
        maxInterval = kafkaErrorHandlerProperties.maxInterval.toMillis()  
        maxAttempts = kafkaErrorHandlerProperties.maxAttempts  
    }  
  
    return DefaultErrorHandler(retryingKafkaConsumerRecordRecoverer bo).apply {  
        setRetryListeners(retryListener)  
    }  
}
```

Настройки **KafkaListener**



```
open class RetryingKafkaConsumerRecordRecoverer: ConsumerRecordRecoverer {  
  
    override fun accept(t: ConsumerRecord<*, *>?, u: Exception?) {  Vasiliy Osadchiy *  
        throw (u ?: IllegalStateException(  
            "[topic ${t?.topic() ?: '?'}, partition ${t?.partition() ?: '?'}, offset ${t?.offset() ?: '?'}] " +  
            "Failed to consume record with unknown error"  
        ))  
    }  
}
```

Запускаем streaming **после фиксов**



- Работает как задумано - ошибки обрабатываются без пропуска данных

Запускаем streaming **после фиксов**



- Работает как задумано - ошибки обрабатываются без пропуска данных
- **НО через некоторое время опять пропуск!**



16:26

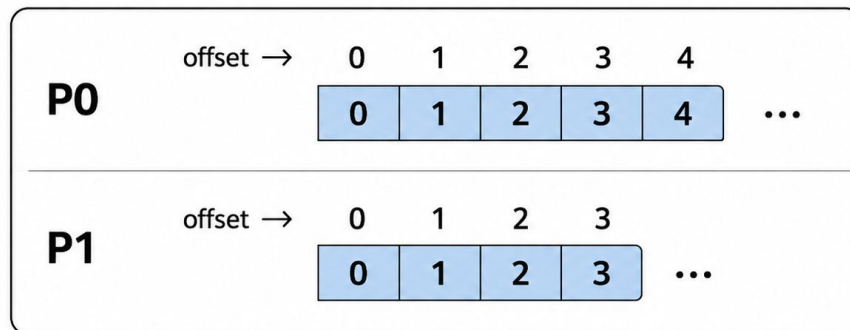
Кажется, у нас некоторое кол-во записей потерялось (или начало теряться) во время увеличения кол-ва партиций 😞 По крайней мере по времени совпадает с разницей в пару минут. Но какая именно взаимосвязь - пока не понял

Проблема с offset-ами

**1**

Изначальное состояние

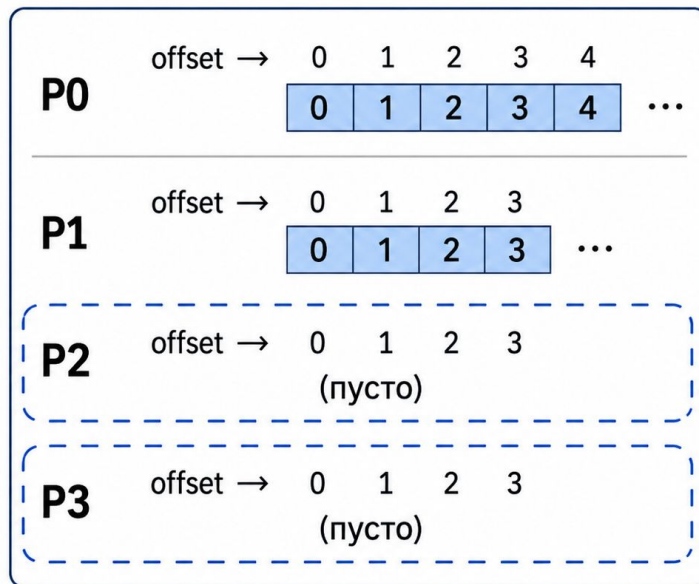
2 партии, consumer читает нормально



Проблема с offset-ами

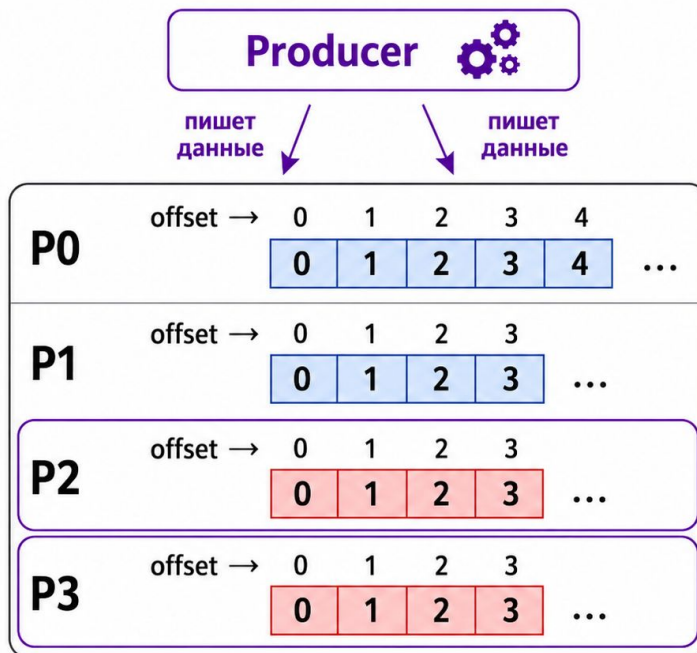


2 Добавили новые партиции: P2 и P3



Проблема с offset-ами

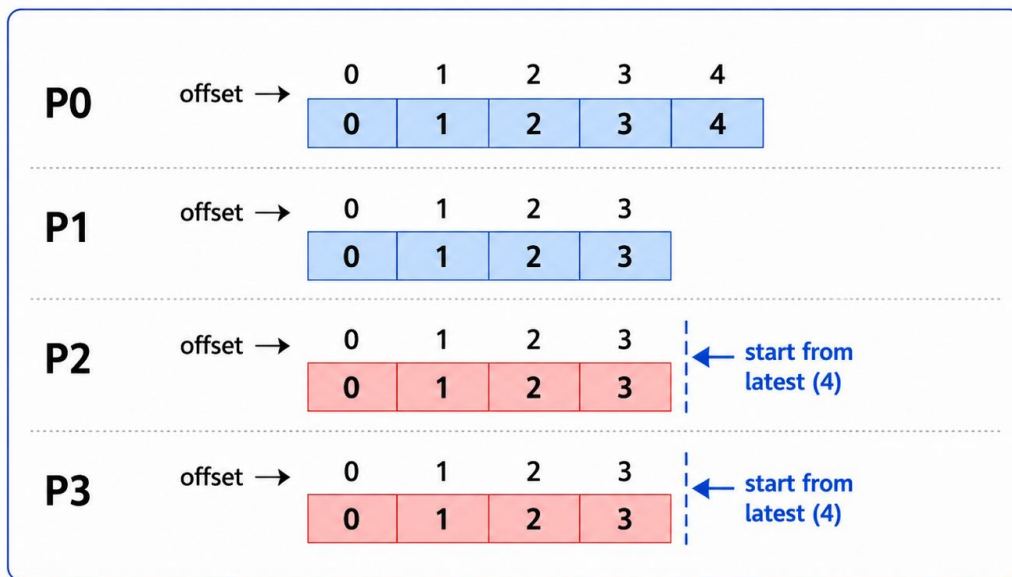
3 Producer пишет новые сообщения в P2 и P3



Проблема с offset-ами

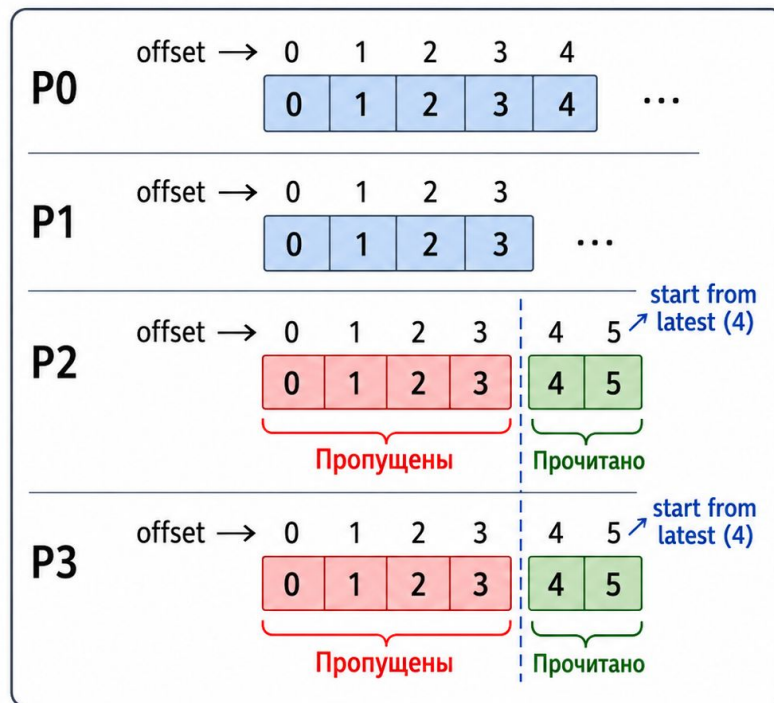
**4****Consumer получает P2 и P3**

Her committed offset. auto.offset.reset = latest



Проблема с offset-ами

5 Потребитель читает только новые сообщения в P2 и P3



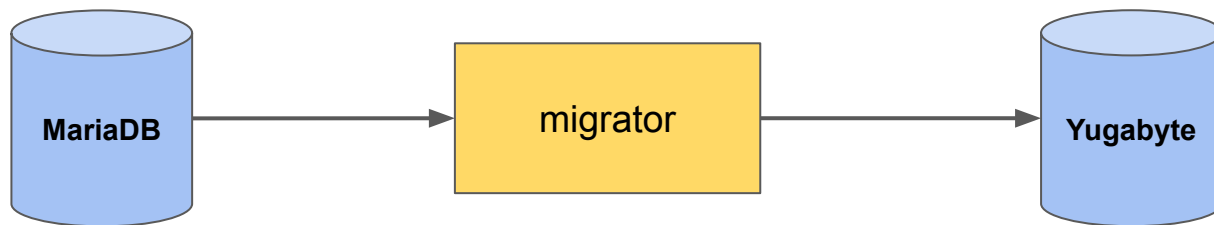


Настройки @KafkaListener

- В kafka мире default-ы часто настроены на то чтобы пропускать сообщения при проблемах
- Нам нужно поведение “умри, но доставь”
 - errorHandler для *ConcurrentKafkaListenerContainerFactory* с бесконечными попытками
 - задаём *ConsumerRecordRecoverer*, который защищает от пропусков при внутренних ошибках *KafkaListener*-а
 - **auto-offset-reset: earliest**



Backfill исторических данных



Backfill исторических данных



- Читаем все транзакции и балансы пользователя
- Записываем пачкой через команду COPY
 - самый быстрый вариант писать в Yugabyte - до 900000 строк в секунду на нашем кластере



Backfill исторических данных

- Читаем все транзакции и балансы пользователя
- Записываем пачкой через команду COPY
 - самый быстрый вариант писать в Yugabyte - до 900000 строк в секунду на нашем кластере
- Время одного backfill-а с учётом индексов - **~7 дней!**

Запускаем backfill



- Реализовали, запустили

Запускаем backfill



- Реализовали, запустили
- Работает, скорость перелива хорошая

Запускаем backfill



- Реализовали, запустили
- Работает, скорость перелива хорошая
- **НО через несколько часов застревает**

Запускаем backfill



- Реализовали, запустили
- Работает, скорость перелива хорошая
- **НО через несколько часов застревает**
 - ошибки при retry записи



Запускаем backfill

- Реализовали, запустили
- Работает, скорость перелива хорошая
- **НО через несколько часов застревает**
 - ошибки при retry записи
 - COPY может работать только с одним UNIQUE CONSTRAINT

Запускаем backfill 2



- Убрали все UNIQUE INDEX-ы, кроме PRIMARY KEY

Запускаем **backfill 2**



- Убрали все UNIQUE INDEX-ы, кроме PRIMARY KEY
- Запускаем, теперь успешно налилось

Запускаем backfill 2



- Убрали все UNIQUE INDEX-ы, кроме PRIMARY KEY
- Запускаем, теперь успешно налилось
- **НО UNIQUE INDEX-ы после этого не могут построиться**



Запускаем backfill 2

- Убрали все UNIQUE INDEX-ы, кроме PRIMARY KEY
- Запускаем, теперь успешно налилось
- **НО UNIQUE INDEX-ы после этого не могут построиться**
 - из ошибки не понятно с каким id проблема



Запускаем backfill 2

- Убрали все UNIQUE INDEX-ы, кроме PRIMARY KEY
- Запускаем, теперь успешно налилось
- **НО UNIQUE INDEX-ы после этого не могут построиться**
 - из ошибки не понятно с каким id проблема
 - SELECT-ом так просто дубли не найдёшь - очень много id-шек

Лечим backfill 2



- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли

Лечим backfill 2



- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли
- **Несколько дней** его работы и дубли нашлись

Лечим backfill 2



- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли
- **Несколько дней** его работы и дубли нашлись
 - на части таблиц PRIMARY KEY был (id_user, id), а не (id)



Лечим backfill 2

- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли
- **Несколько дней** его работы и дубли нашлись
 - на части таблиц PRIMARY KEY был (id_user, id), а не (id)
 - нашёлся один пользователь, которому сгенерили id не из SEQUENCE в БД - id перестал быть уникальным



Лечим backfill 2

- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли
- **Несколько дней** его работы и дубли нашлись
 - на части таблиц PRIMARY KEY был (id_user, id), а не (id)
 - нашёлся один пользователь, которому сгенерили id не из SEQUENCE в БД - id перестал быть уникальным
- Удалили транзакции проблемного пользователя



Лечим backfill 2

- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли
- **Несколько дней** его работы и дубли нашлись
 - на части таблиц PRIMARY KEY был (id_user, id), а не (id)
 - нашёлся один пользователь, которому сгенерили id не из SEQUENCE в БД - id перестал быть уникальным
- Удалили транзакции проблемного пользователя
- Ещё раз переналиваем данные



Лечим backfill 2

- Пишем обходчик, который постранично идёт по MariaDB и ищет дубли
- **Несколько дней** его работы и дубли нашлись
 - на части таблиц PRIMARY KEY был (id_user, id), а не (id)
 - нашёлся один пользователь, которому сгенерили id не из SEQUENCE в БД - id перестал быть уникальным
- Удалили транзакции проблемного пользователя
- Ещё раз переналиваем данные
- Успешно налилось, **НО данные легли криво**
 - не смогли прочитать то, что записали...

Где тут баг?



```
fun keyHashCode(  
    clientId: Long,  
    accountType: AccountType,  
    accountId: String,  
): Int {  
    var result = accountId.hashCode()  
  
    result = 31 * result + clientId.hashCode()  
    result = 31 * result + accountType.hashCode()  
  
    return result  
}
```

```
enum class AccountType {  
    USER,  
    SYSTEM,  
}
```

Где тут баг?



```
fun keyHashCode(  
    clientId: Long,  
    accountType: AccountType,  
    accountId: String,  
): Int {  
    var result = accountId.hashCode()  
  
    result = 31 * result + clientId.hashCode()  
    result = 31 * result + accountType.hashCode()  
  
    return result  
}
```

```
enum class AccountType {  
    USER,  
    SYSTEM,  
}
```

ФИКС

```
fun keyHashCode(  
    clientId: Long,  
    accountType: AccountType,  
    accountId: String,  
): Int {  
    var result = accountId.hashCode()  
  
    result = 31 * result + clientId.hashCode()  
    result = 31 * result + accountType.ordinal  
  
    return result  
}
```

```
enum class AccountType {  
    USER,  
    SYSTEM,  
}
```

Как состыковать **backfill** и **streaming**



- Как не потерять более новые данные более старыми?

Как состыковать **backfill** и **streaming**



- Как не потерять более новые данные более старыми?
- Время `updated_at` не подходит - нет гарантий монотонного возрастания

Как состыковать **backfill** и **streaming**



- Как не потерять более новые данные более старыми?
- Время `updated_at` не подходит - нет гарантий монотонного возрастания
- Добавляем поле `version`
 - монотонно возрастает в рамках одной строчки

Как состыковать **backfill** и **streaming**



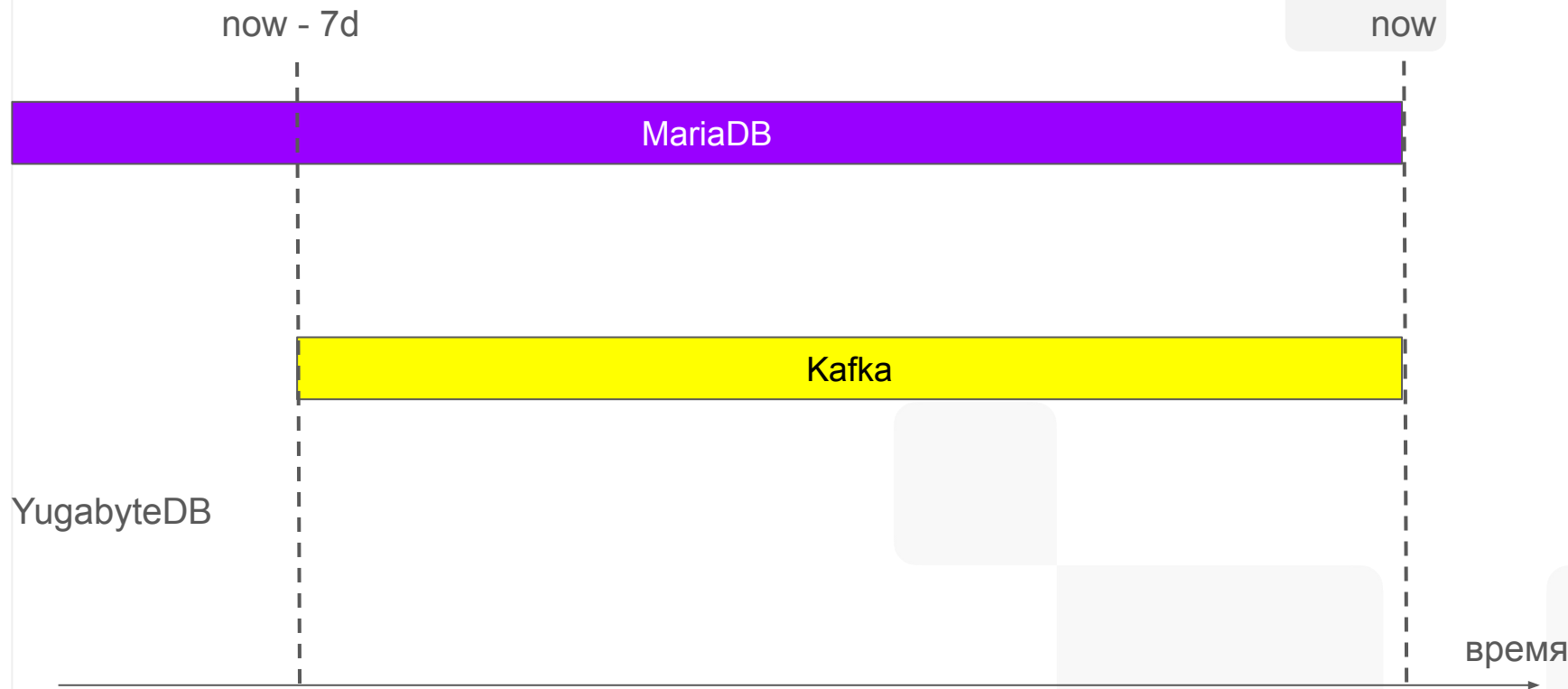
```
CREATE TRIGGER ma_deposits_increment_version
  BEFORE UPDATE ON ma_deposits
  FOR EACH ROW
  SET NEW.version = GREATEST(
    COALESCE(OLD.version, 1) + 1,
    COALESCE(NEW.version, 1)
  );
```



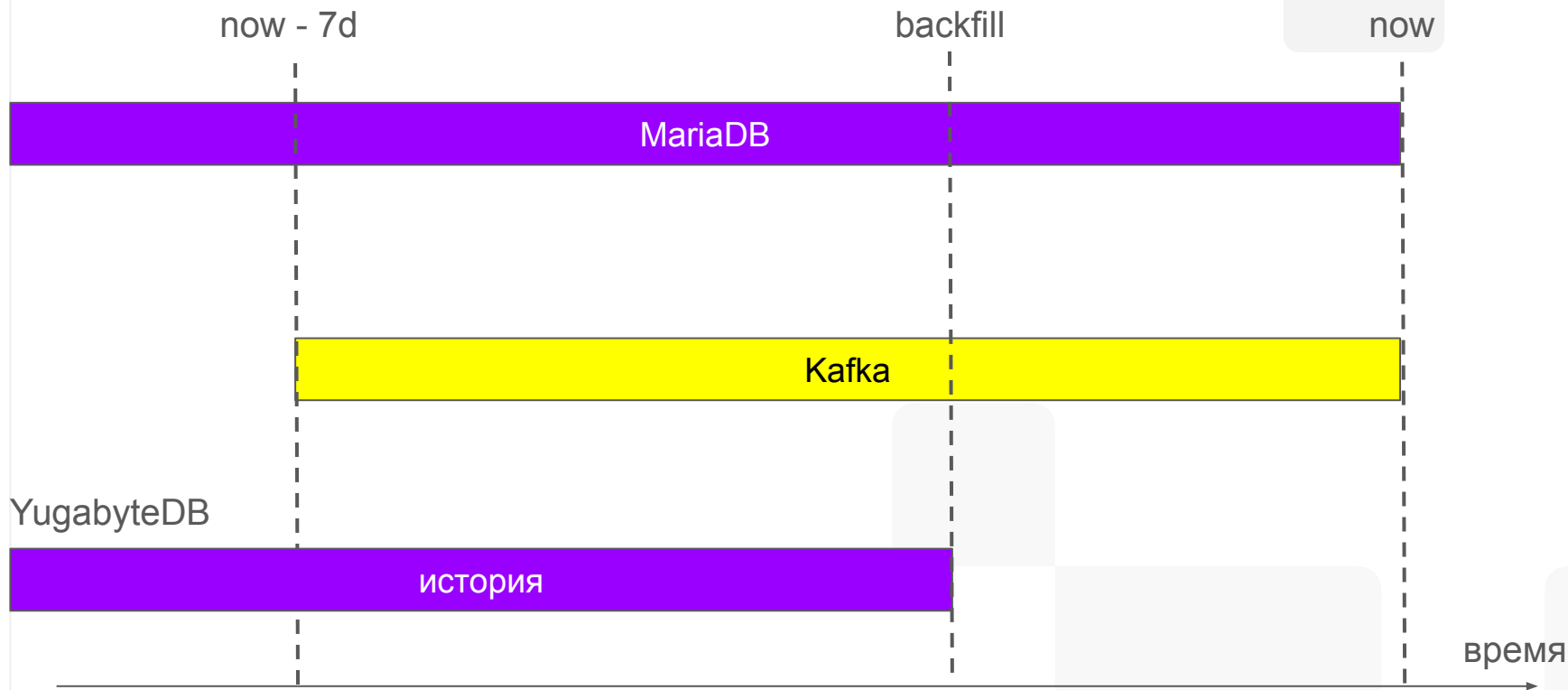
Как состыковать **backfill** и **streaming**

- Как не потерять более новые данные более старыми?
- Время `updated_at` не подходит - нет гарантий монотонного возрастания
- Добавляем поле `version`
 - монотонно возрастает в рамках одной строчки
- Вставку в БД делаем через UPSERT-ы с проверкой `version`

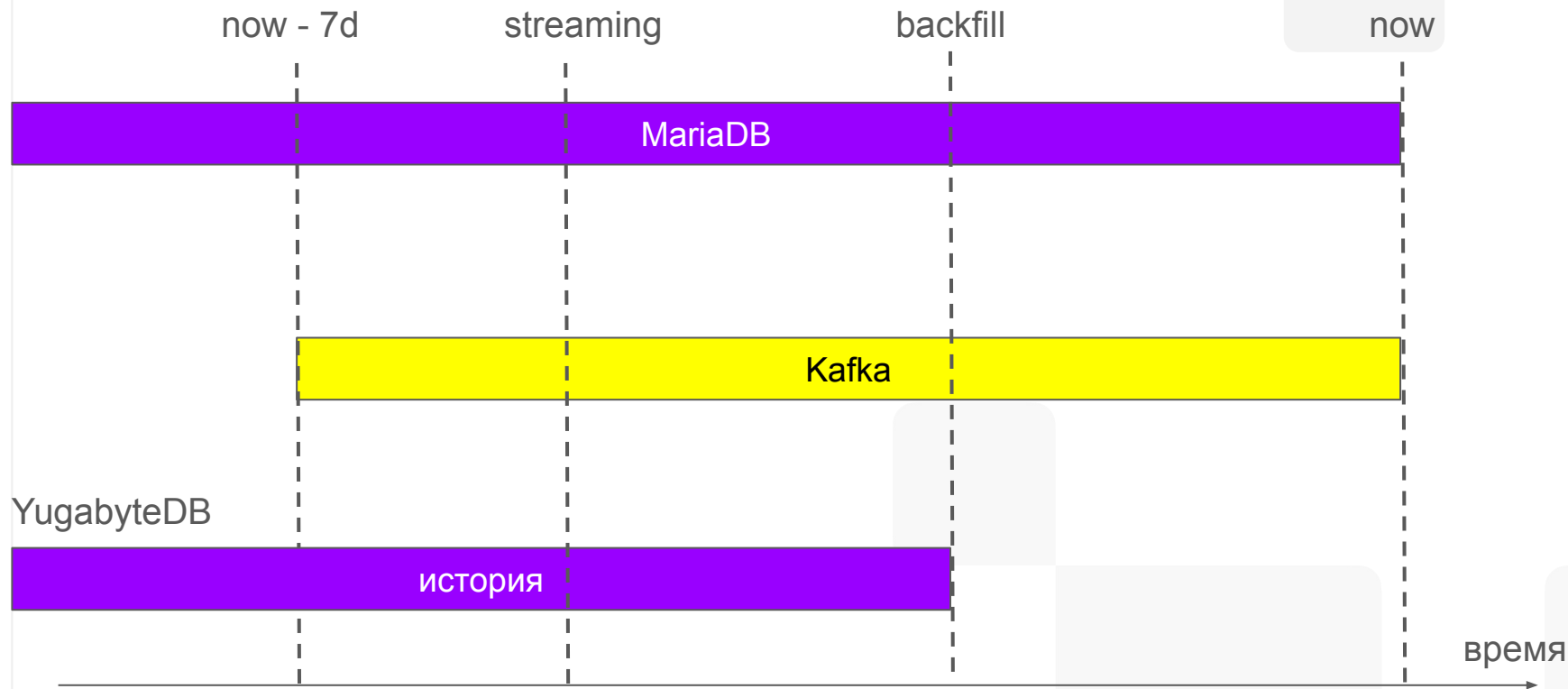
План состыковки **backfill** и **streaming**



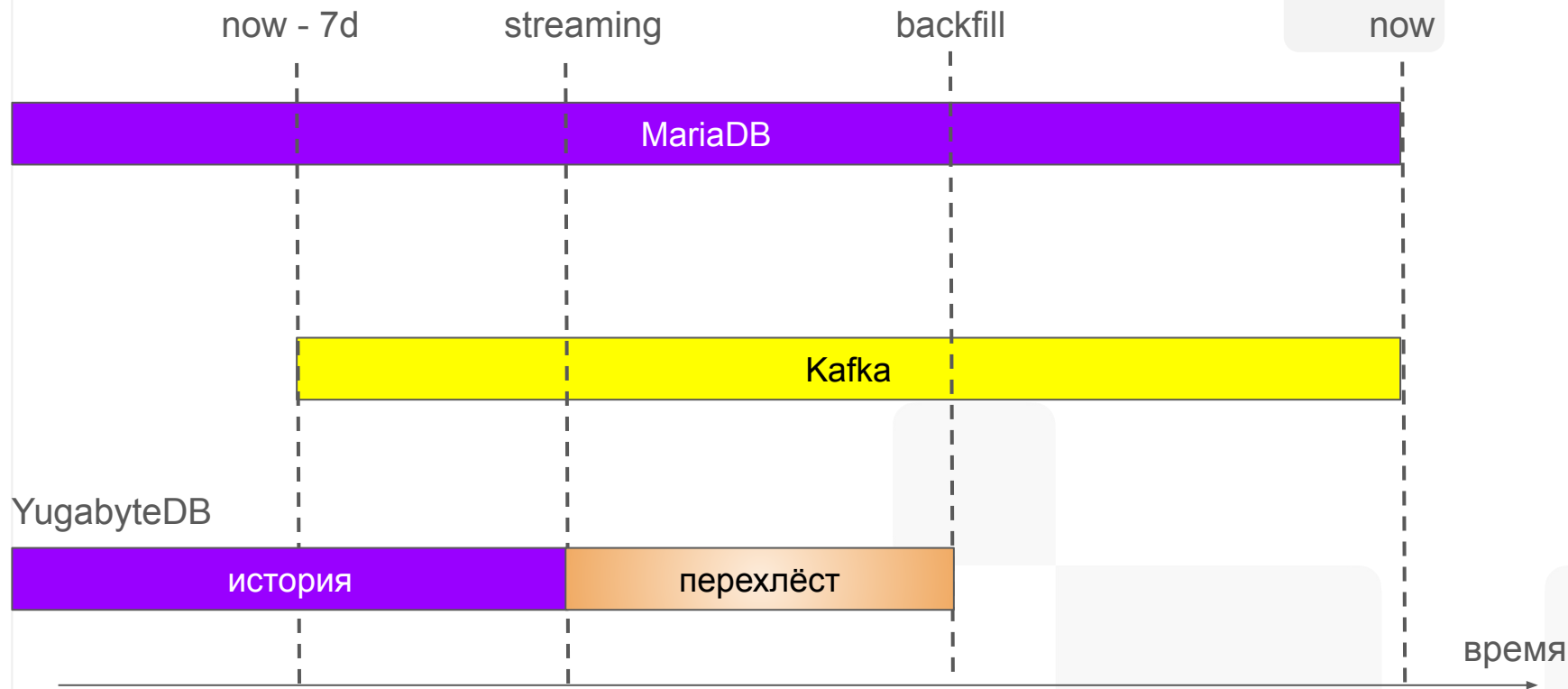
План состыковки **backfill** и **streaming**



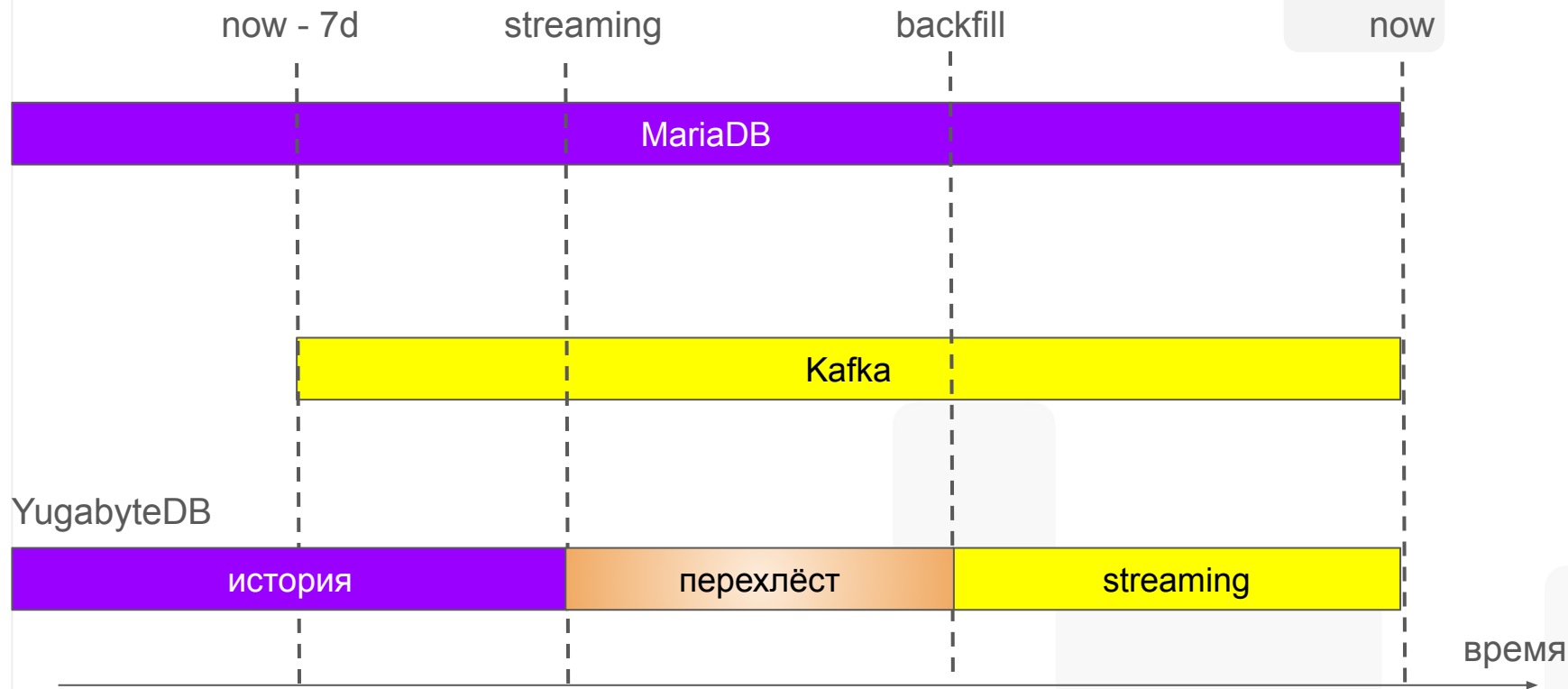
План состыковки **backfill** и **streaming**



План состыковки **backfill** и **streaming**



План состыковки **backfill** и **streaming**



Как откатывали **переключение записей**



Из требований к миграции:

[3] Возможность отката на старую систему в случае проблем

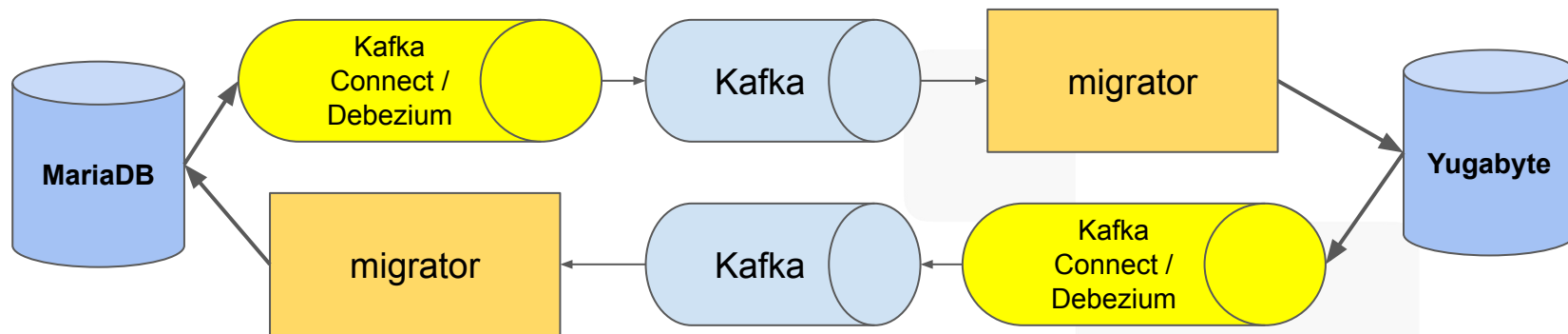
Как откатывали переключение записей



Из требований к миграции:

[3] Возможность отката на старую систему в случае проблем

- Обратный streaming:







План

- [1] Проблемы, которые решаем
- [2] Схема переезда
- [3] Как не делать лишнюю работу
- [4] Миграция и репликация данных
- [5] **Параллельный режим**
- [6] Выводы



Вспомним **требования** к миграции

[1] Отсутствие downtime при миграции

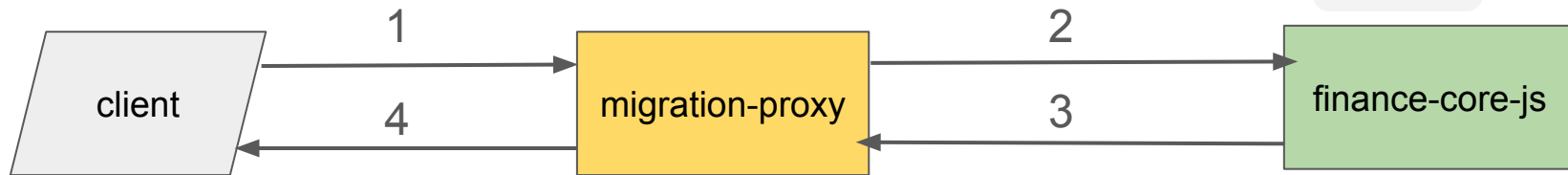
Очень дорогой простой сервиса

[3] Возможность отката на старую систему в случае проблем

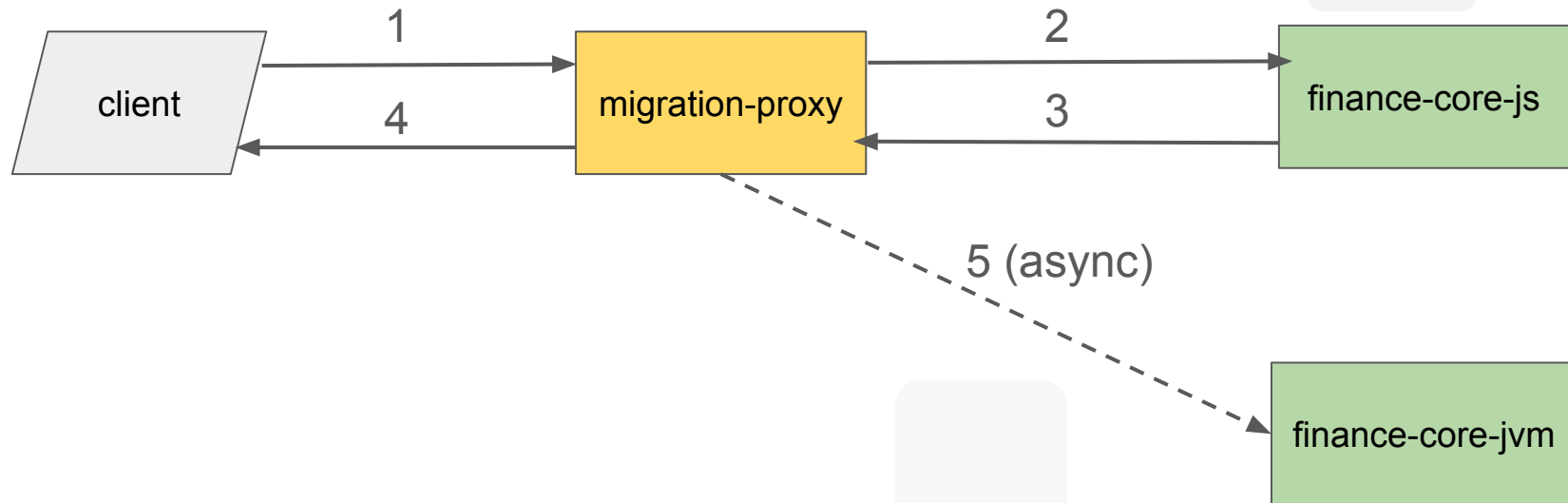
Возможность экспериментировать не боясь положить прод
ОЧЕНЬ много чего меняем при переезде

⇒ *Переключение между старой и новой системой на лету*

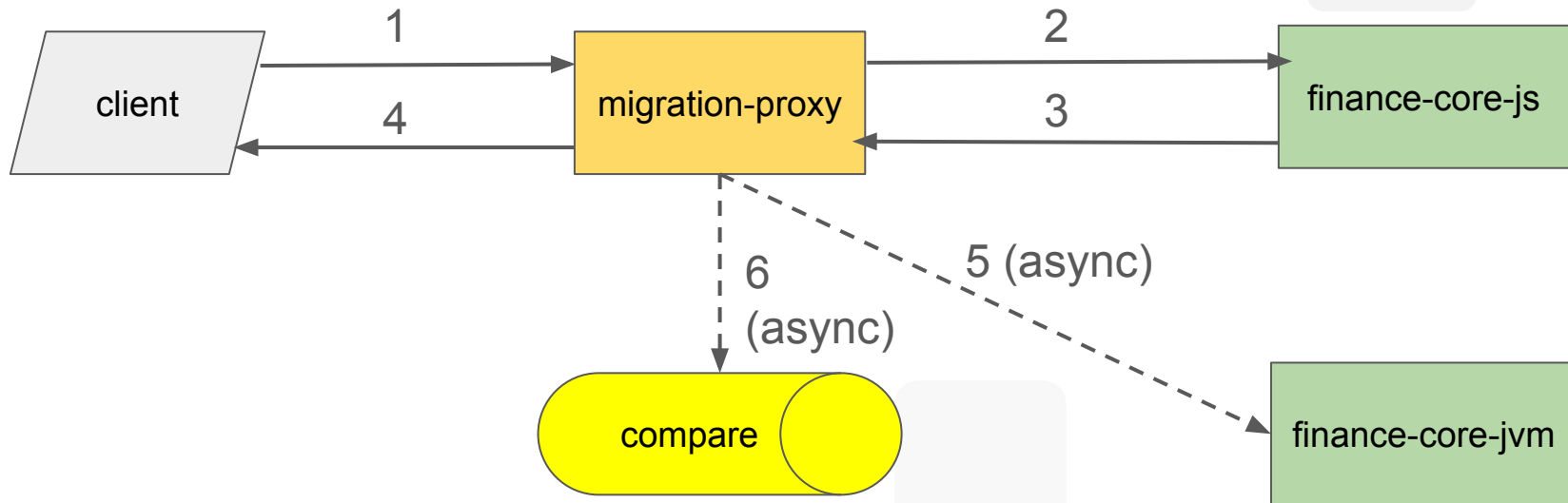
Параллельный режим “на пальцах”



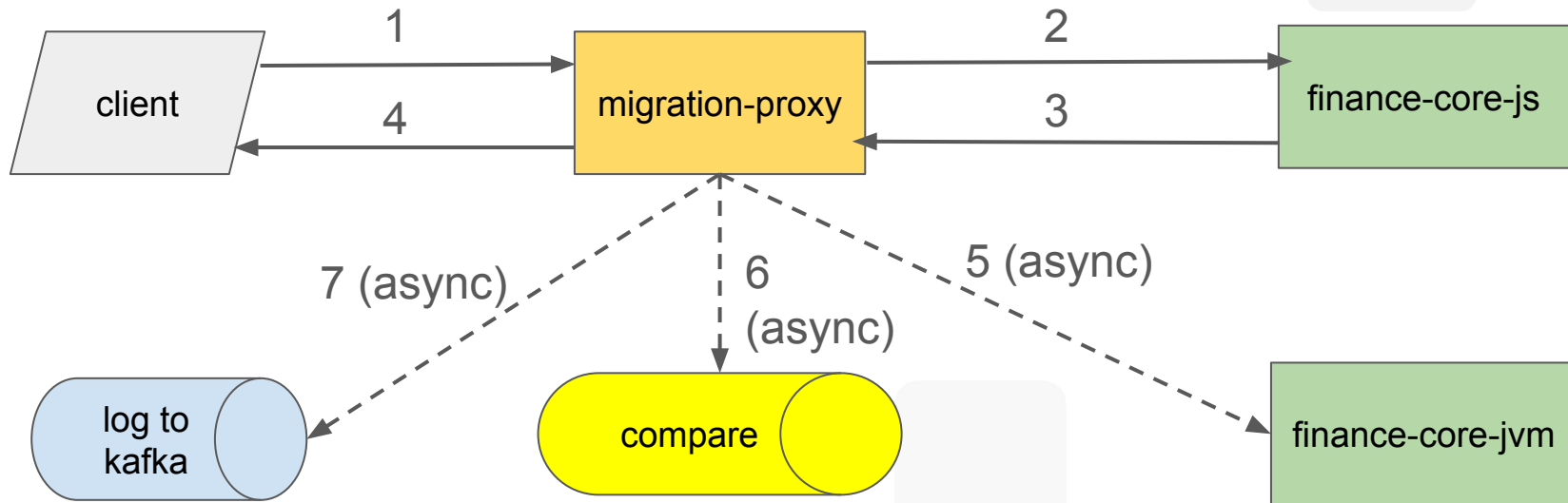
Параллельный режим “на пальцах”



Параллельный режим “на пальцах”



Параллельный режим “на пальцах”



Нюансы **параллельных вызовов**



- Максимальное изолирование влияния secondary системы на запросы клиентов
 - отдельные executor-ы



Нюансы **параллельных вызовов**

- Максимальное изолирование влияния secondary системы на запросы клиентов
 - отдельные executor-ы
- При забивании pool-а executor-а весёлые аномалии
 - напр. получение более свежих данных из secondary системы т.к. в неё запрос долетел на 30 секунд позже



Нюансы **параллельных вызовов**

- Максимальное изолирование влияния secondary системы на запросы клиентов
 - отдельные executor-ы
- При забивании pool-а executor-а весёлые аномалии
 - напр. получение более свежих данных из secondary системы т.к. в неё запрос долетел на 30 секунд позже
- Возможность настраивать на лету
 - куда пойдёт запрос (с возможностью переключить лишь часть из них)
 - какая система для запроса основная



Вспомним **требования** к миграции

[2] Гарантии, что новая система работает так же как и старая

Тех. долг + большое кол-во клиентов = сложно уверенно сказать, как оно должно работать для каждого случая

Одними тестами не отделаешься

⇒ *Специальными компараторами сравниваем ответы - все текущие запросы на prod-е становятся тестами!*

Подводные камни **сравнений**



- Естественные расхождения
 - отставание данных в **secondary** системе

Подводные камни **сравнений**



- Естественные расхождения
 - отставание данных в **secondary** системе
- Расхождения не влияющие на семантику:

Подводные камни **сравнений**



- Естественные расхождения
 - отставание данных в **secondary** системе
- Расхождения не влияющие на семантику:

12.45 ↔ 12.450000000



Подводные камни сравнений

- Естественные расхождения
 - отставание данных в **secondary** системе
- Расхождения не влияющие на семантику:

12.45 ↔ 12.450000000

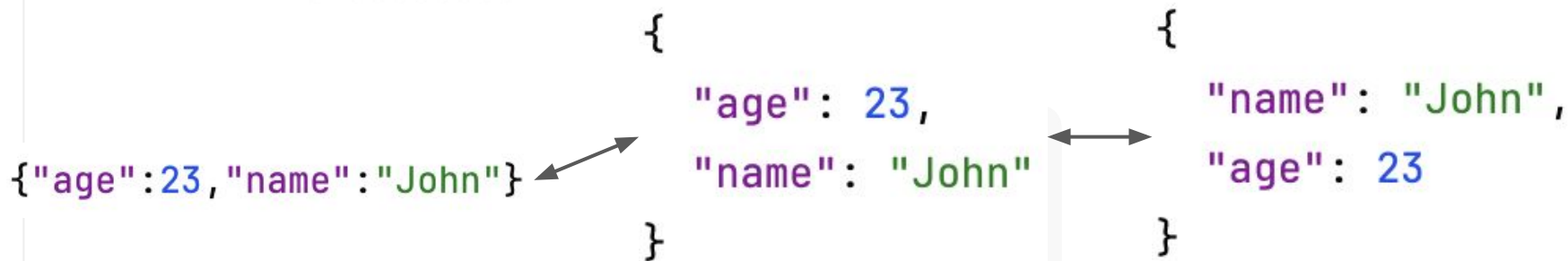
```
{  
  "age": 23,  
  "name": "John"  
}  
↔  
{  
  "name": "John",  
  "age": 23  
}
```



Подводные камни сравнений

- Естественные расхождения
 - отставание данных в **secondary** системе
- Расхождения не влияющие на семантику:

12.45 ↔ 12.450000000



Компараторы **в коде**



```
enum class MatchingResult(val priority: Int) {  
    STRICT_MATCH(1),  
    WEAK_MATCH(2),  
    MISMATCH_IN_FRESH_DATA(3),  
    MISMATCH(4);  
}
```

Компараторы **в коде**



```
abstract class ResultComparator<T> {  
    abstract fun compare(  
        legacyResult: T,  
        newResult: T,  
        context: ComparatorsContext  
    ): MatchingResult  
}
```



Вспомним **требования** к миграции

[4] Надо поменять API не меняя код на стороне клиентов
Старый API чрезмерно гибок (SQL переложенный на JSON) +
смена схемы БД обязывает

⇒ *Mapping старого API на новый в migration-proxy - от клиентов
ничего не требуется*

Нюансы по **маппингу API**



- Для клиента максимально эмулируем старую систему
 - напр. формат ошибок



Нюансы по **маппингу API**

- Для клиента максимально эмулируем старую систему
 - напр. формат ошибок
- В старом API работаем с сырым JsonObject
 - нет гарантий, что OpenAPI спека полная
 - TypeScript в runtime становится JavaScript - могут появляться поля, которых в классе/методе не объявляли



Нюансы по **маппингу API**

- Для клиента максимально эмулируем старую систему
 - напр. формат ошибок
- В старом API работаем с сырым JsonObject
 - нет гарантий, что OpenAPI спека полная
 - TypeScript в runtime становится JavaScript - могут появляться поля, которых в классе/методе не объявляли
- Насколько возможно переиспользуем логику маппинга сущностей с мигратором
 - legacy API → legacy DB DTO → new DB DTO → new API

Всё, поехали?



Всё, поехали?



Не прокатило:

- Данные могут разъезжаться при параллельной записи
 - гонки - в старой и в новой системах запросы могут обработаться в разном порядке

Всё, поехали?



Не прокатило:

- Данные могут разъезжаться при параллельной записи
 - гонки - в старой и в новой системах запросы могут обработаться в разном порядке
 - → для записи параллельный режим нам не катит

Пример **гонки**



Что **делать?**



- Параллельные вызовы только для чтений

Что **делать?**

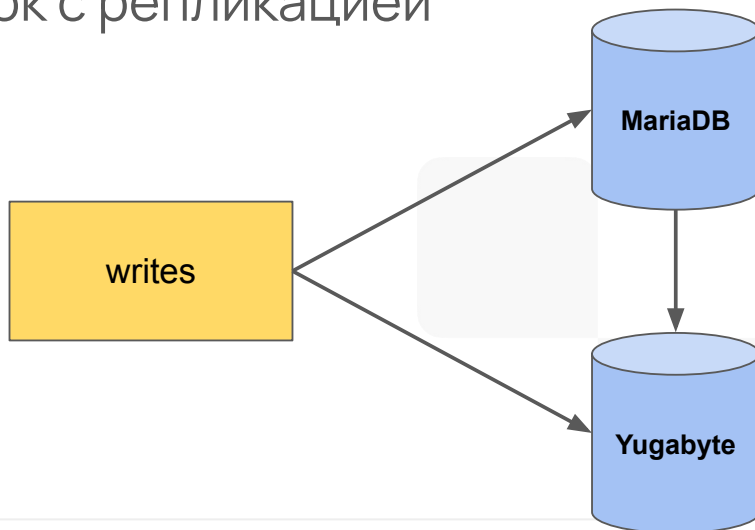


- Параллельные вызовы только для чтений
- Записи максимально покрываем тестами



Что **делать?**

- Параллельные вызовы только для чтений
- Записи максимально покрываем тестами
- При переключении записей временно их блокируем
 - избегаем гонок с репликацией





WHAT DO WE WANT



NOW



WHEN DO WE WANT IT



RACE CONDITIONS





План

- [1] Проблемы, которые решаем
- [2] Схема переезда
- [3] Как не делать лишнюю работу
- [4] Миграция и репликация данных
- [5] Параллельный режим
- [6] **Выводы**

Выводы

- Важно понимать как пользуются вашей системой!





Выводы

- Важно понимать как пользуются вашей системой!
- Debezium + Kafka норм вариант для репликации
 - **НО** надо аккуратно с настройками, чтобы не терять данные!



Выводы

- Важно понимать как пользуются вашей системой!
- Debezium + Kafka норм вариант для репликации
 - **НО** надо аккуратно с настройками, чтобы не терять данные!
- Параллельный режим - классный способ
 - протестить систему нагрузочно
 - протестить систему функционально - все запросы с prod-а становятся тестами!
 - иметь возможность откатиться

Итоги

- Перелили **100млрд+** записей



Итоги

- Перелили **100млрд+** записей
- Потребовалось **3 переналива** данных



Итоги



- Перелили **100млрд+** записей
- Потребовалось **3 переналива данных**
- **90 расхождений** пофиксили в логике чтений благодаря параллельному режиму



Итоги

- Перелили **100млрд+ записей**
- Потребовалось **3 переналива данных**
- **90 расхождений** пофиксили в логике чтений благодаря **параллельному режиму**
- **88 расхождений** пофиксили в логике записей благодаря тестам с **TestContainer-ами**



Итоги

- Перелили **100млрд+** записей
- Потребовалось **3 переналива данных**
- **90 расхождений** пофиксили в логике чтений благодаря **параллельному режиму**
- **88 расхождений** пофиксили в логике записей благодаря тестам с **TestContainer-ами**
- **5 багов** в редких кейсах поймали и пофиксили на проде



Итоги

- Перелили **100млрд+** записей
- Потребовалось **3 переналива данных**
- **90 расхождений** пофиксили в логике чтений благодаря **параллельному режиму**
- **88 расхождений** пофиксили в логике записей благодаря тестам с **TestContainer-ами**
- **5 багов** в редких кейсах поймали и пофиксили на проде
- Переключили **73 клиента** на новую систему без доработок с их стороны



Итоги

- Перелили **100млрд+** записей
- Потребовалось **3 переналива данных**
- **90 расхождений** пофиксили в логике чтений благодаря **параллельному режиму**
- **88 расхождений** пофиксили в логике записей благодаря тестам с **TestContainer-ами**
- **5 багов** в редких кейсах поймали и пофиксили на проде
- Переключили **73 клиента** на новую систему без доработок с их стороны
- **Потушили старую систему!**

