Т БАНК

Надежно отправляем события в Apache Kafka

От CDC до паттерна Transactional Outbox

Алексей Кашин

Архитектор | Т-Банк

@AlexeyKashin

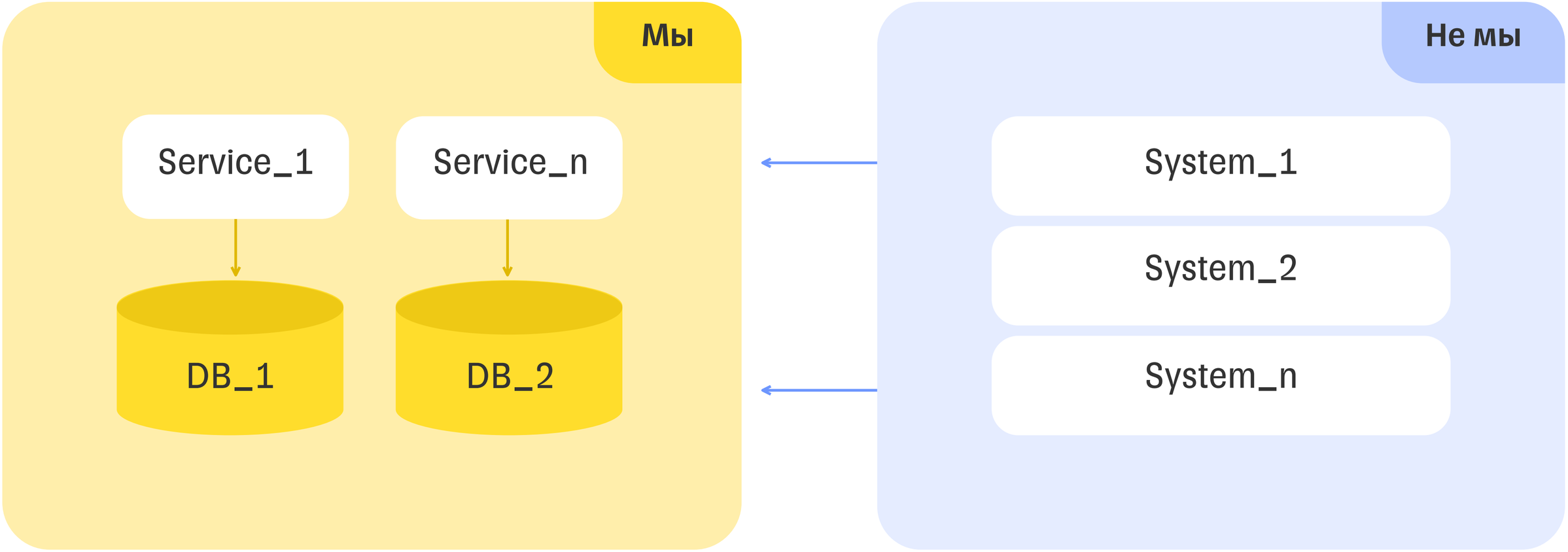
Контекст

Нужно отправлять данные

Описание контекста

Мы храним данные

Другие системы ходят за данными

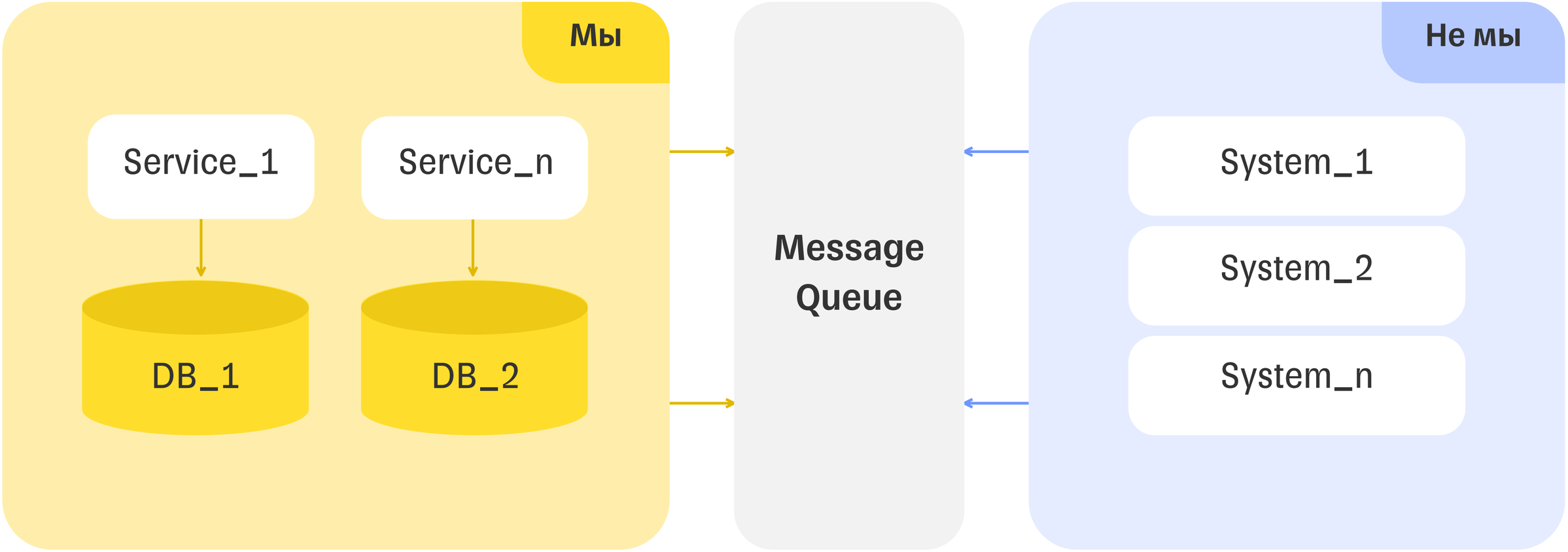


Необходимый функционал

Потребители хотят узнавать об изменении данных

Необходимый функционал

- **Отправляем** данные в message-broker
- Потребители **получают** сообщения и **реализуют** свою логику



Особенности технологий

PostgreSQL и Apache Kafka

Гарантии доставки до Apache Kafka

At-most-once

- Важна производительность и допустимы потери отправленных событий
- Продюсер отправляет сообщение и не дожидается подтверждения (ack) от брокера

At-least-once

- Важно чтобы событие было доставлено, даже если это приведет к дублированию
- Продюсер дожидается подтверждения (ack) от брокера
- Чтобы обеспечить гарантию - нужны ретраи

Exactly-once

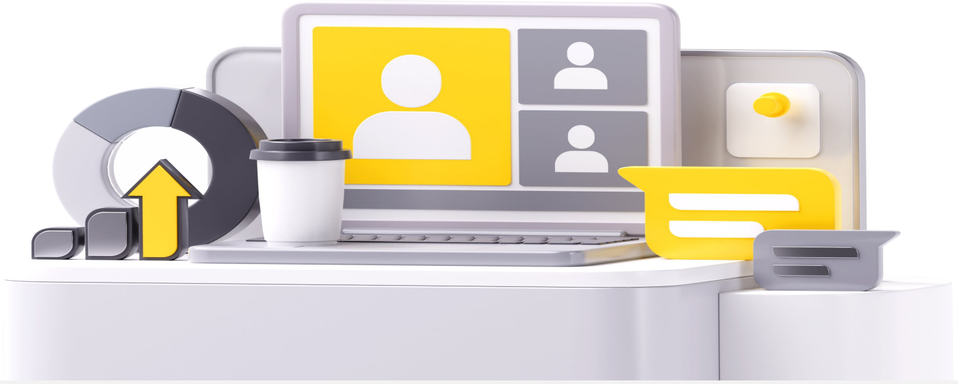
- Критична доставка и недопущение дублирования
- Идемпотентный продюсер (экземпляр, не кластер), предотвращает дублирование сообщения
- Транзакции kafka гарантируют доставку
- В техническом плане самый сложный уровень, может не гарантироваться на самом брокере

Гарантии доставки до Apache Kafka

Наш выбор at-least-once

В событии отправляем идентификатор, версию и время вставки события

Договариваемся с потребителями что могут быть дубли, нужно уметь обрабатывать их



Гарантии доставки до Apache Kafka

Доклад: Виктор Гамов — Один раз в год сады цветут: разбор семантики «exactly once» Apache Kafka

https://youtu.be/PgkRhlUwYyE

Vacuum

PostgreSQL использует многоверсионную архитектуру (MVCC — Multi-Version Concurrency Control), которая позволяет одновременно поддерживать несколько версий одной и той же строки в таблице.

Вакуум (VACUUM) в PostgreSQL — это команда и процесс, предназначенный для очистки и реорганизации базы данных. Основной задачей вакуума является управление пространством на диске, удаление "мертвых" (устаревших) версий строк и предотвращение роста размера базы данных без необходимости.

Vacuum

Vacuum

- Удаляет "мертвые" строки
- Обновляет статистику системы, «подчищает» индексы
- Не возвращает освобожденное пространство на диск операционной системе

Vacuum Full

Используется в случаях, когда необходимо значительно уменьшить размер таблицы на диске. Блокирует таблицу, опасно использовать в продакшне в рабочее время.

Автовакуум

- Автовакуум работает в фоновом режиме и автоматически запускается при накоплении определенного количества "мертвых" строк в таблице
- Автовакуум помогает поддерживать базу данных в хорошем состоянии без необходимости вручную запускать VACUUM

Vacuum

Статьи от PostgresPro

MVCC-6. Очистка — https://habr.com/ru/companies/postgrespro/articles/452320/

01

MVCC-7. Автоочистка — https://habr.com/ru/companies/postgrespro/articles/452762/

02

Самое простое решение

Прямая отправка

Сохранить в БД и отправить в kafka последовательно

```
public void save(Entity entity) {
    repository.save(entity);
    kafkaTemplate.send(TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));
}
```

Прямая отправка

Нужно ответить на ряд вопросов и решить ряд проблем:

Нужно ли
сохранять в БД
если kafka
не доступна?

Нужно ли отправлять
если не доступна
база?

Как реагировать
на задержки
kafka?

```
public void save(Entity entity) {
    repository.save(entity);
    kafkaTemplate.send(TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));
}
```

Консистентноть в распределенной системе

Сохранение в несколько источников

Решения

- Распределенная транзакция
- Saga
- CQRS/event-sourcing

```
public void save(Entity entity) {
    repository.save(entity);
    kafkaTemplate.send(TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));
}
```

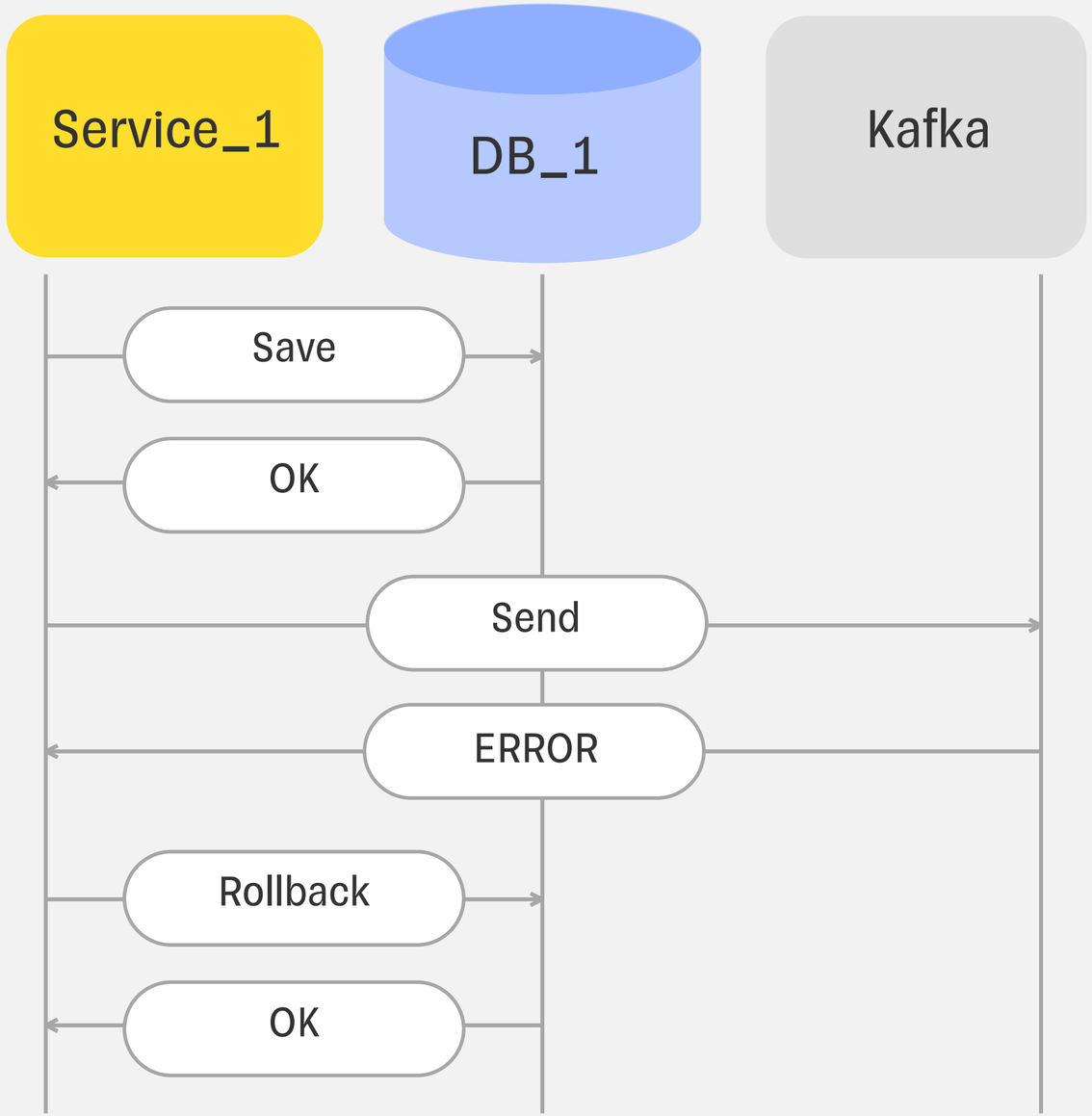
Распределенная транзакция

Saga-pattern

Полная недоступность kafka

Задержки kafka

Двухфазный коммит



Распределенная транзакция

Saga-pattern

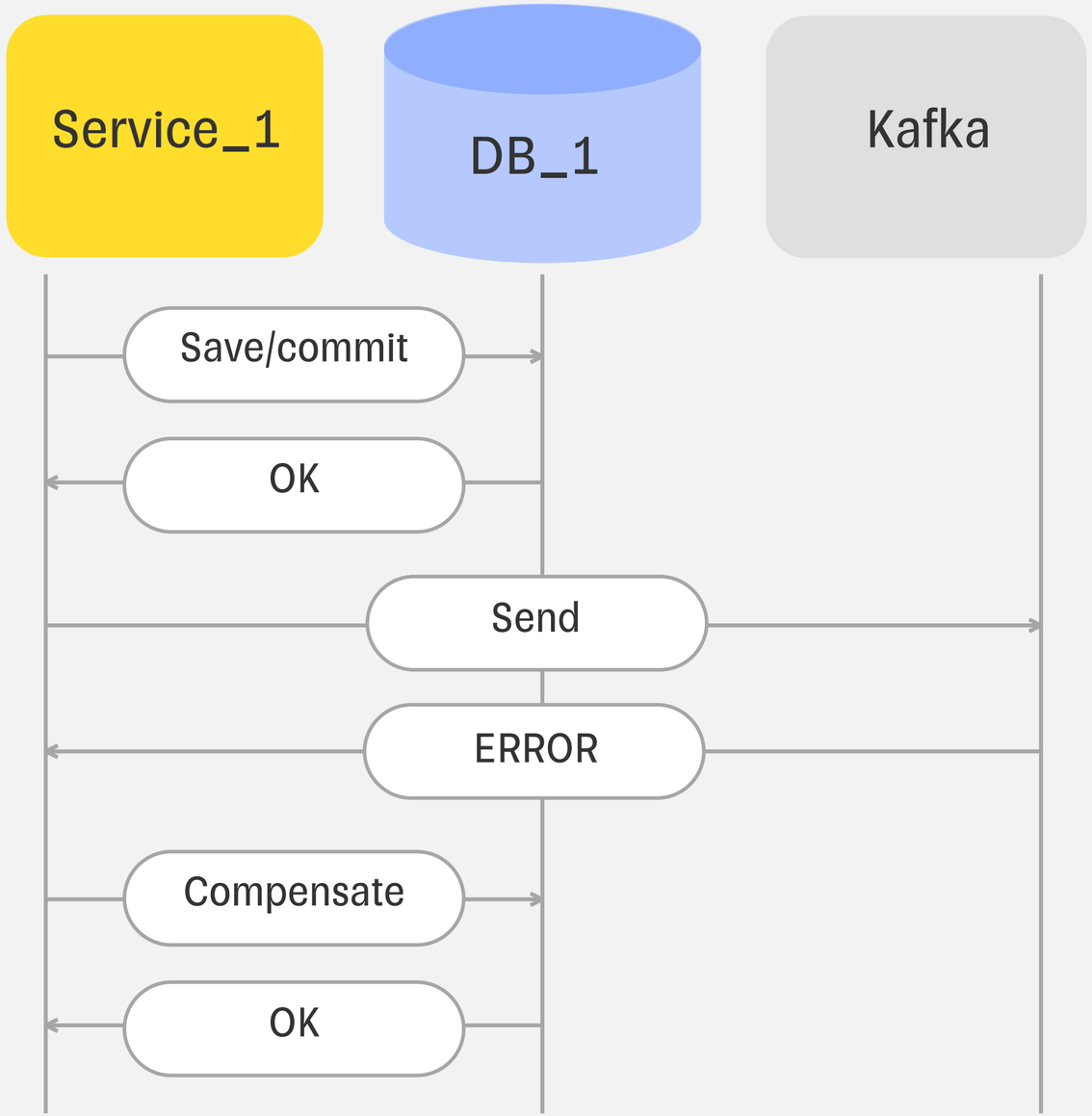
Полная недоступность kafka

Задержки kafka

Saga —
разделение
на мелкие операции
и компенсации

Нет гарантий согласованности в конкретный момент времени

Eventual Consistency



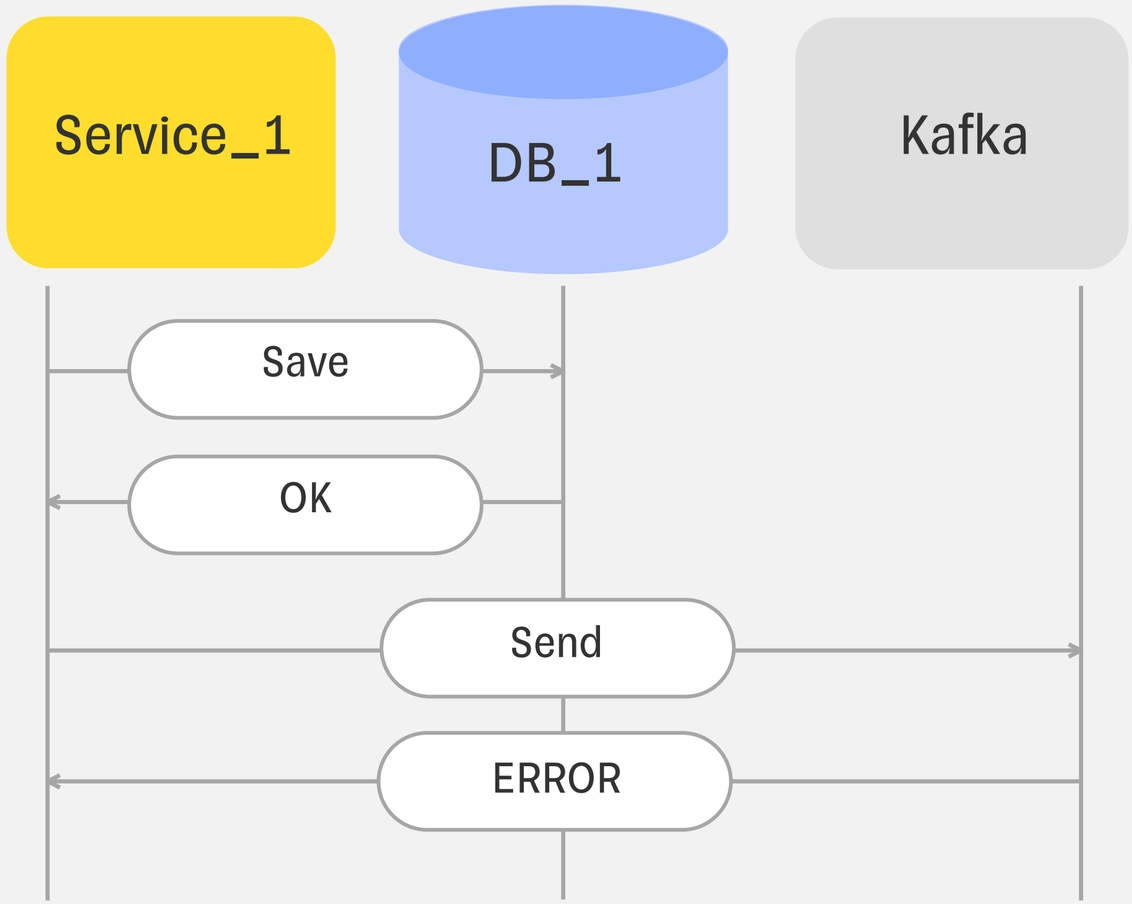
Распределенная транзакция

Saga-pattern

Полная недоступность kafka

Задержки kafka

Как пережить недоступность брокера?



Распределенная транзакция

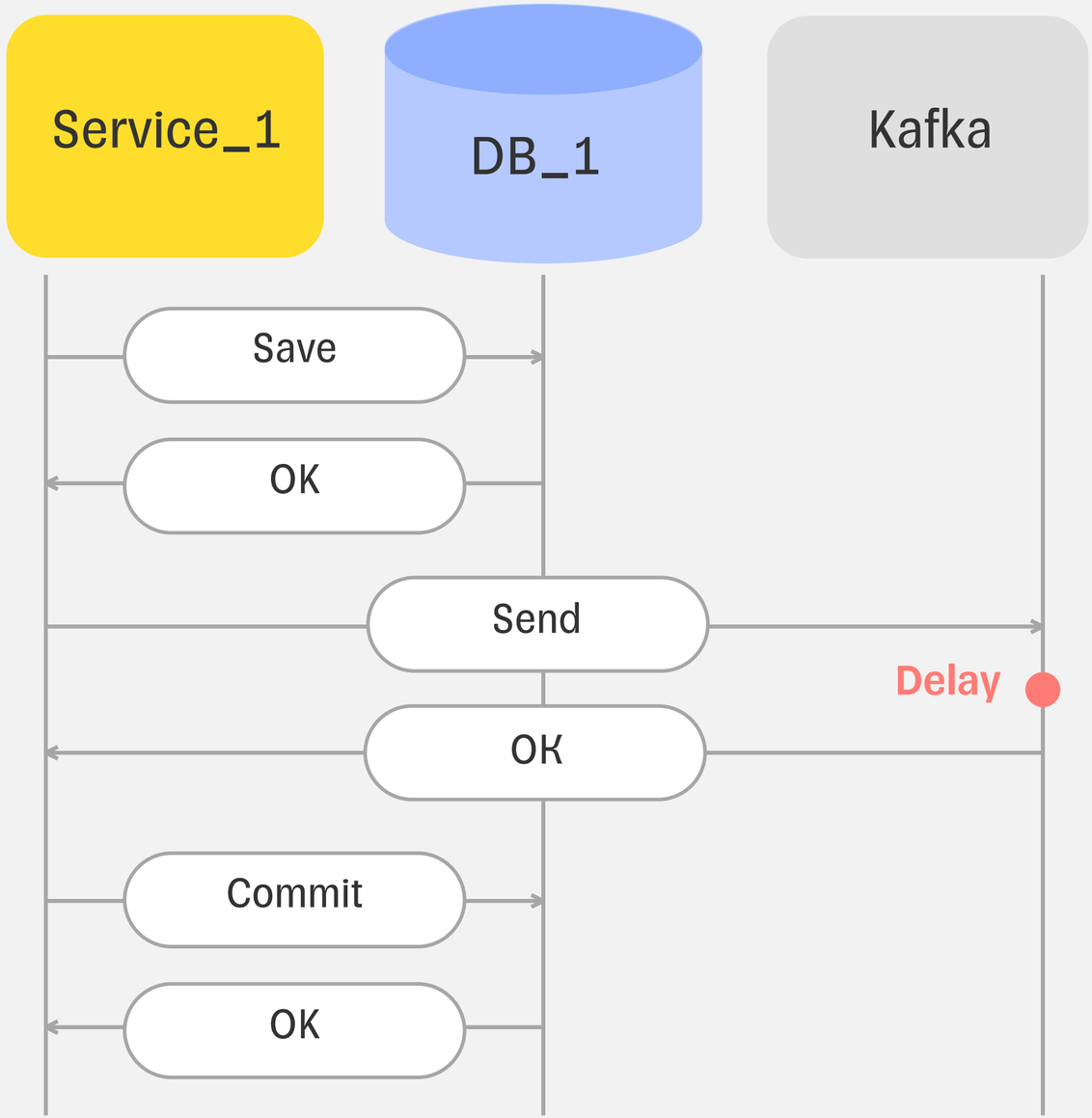
Saga-pattern

Полная недоступность kafka

Задержки kafka

Что делать если выросло ожидание ответа от kafka?

- Общее замедление сервиса
- Исчерпание коннектов



Change Data Capture

Потребности

Вывод медленных
читающих процессов
из операционной БД

Идентичная
копия данных
для восстановления

Поставка
данных как часть
процесса ETL

Подходы

Периодическая выборка

Выбор измененных данных за определенный период

Постоянные изменения

Трансляция изменений в реальном времени

Способы извлечения

Query-based

Выбор данных
для переноса запросами

Triggers

Действия выполняются
в триггерах в БД

Log-based

Данные для переноса
выбираются из лога
транзакций

Примеры

Debezium

Открытый фреймворк для CDC, который
поддерживает различные базы данных,
такие как MySQL, PostgreSQL, MongoDB и другие

AWS Database Migration Service (DMS)

Сервис от Amazon для миграции
и репликации баз данных с поддержкой CDC

Oracle GoldenGate

Инструмент от Oracle для захвата
и репликации изменений данных

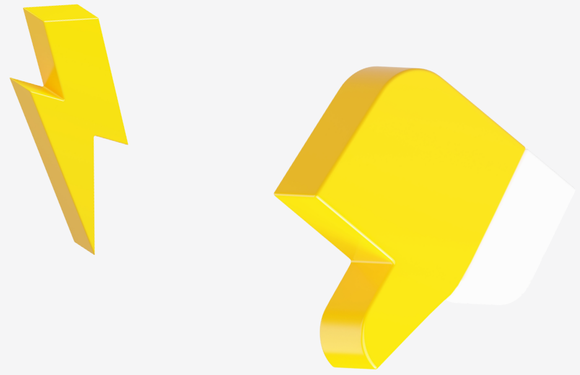
Недостатки

Сложность настройки и управления

- Общий инструмент может не подходить под конкретные требования
- Сложность работы с кастомными avro-схемами

Неуправляемые задержки

Нет гарантии своевременной доставки



Консистентность

Сложность управления транзакциями в случае изменения в разных таблицах

Change Data Capture

Доклад: Андрей Серебрянский —
Грузим в Kafka из базы: с CDC и без

https://youtu.be/llBCFID167E

Transactional outbox

Описание

Transactional Outbox — паттерн для обеспечения гарантированной доставки сообщений в распределенных системах при интеграции с использованием асинхронных коммуникаций, таких как очереди сообщений или лог изменений

Основная идея

Вместо того чтобы пытаться отправить сообщение напрямую во время выполнения бизнес-операции, сообщение записывается **в специальную таблицу базы данных (outbox).**

Эта запись в "outbox" совершается в той же транзакции, что и основные изменения данных, таким образом обеспечивая их **атомарность.**

Отдельный процесс или компонент (например, отдельный поток приложения или специальный сервис) читает записи из "outbox" и **публикует их в необходимый внешний сервис,** такой как очередь сообщений.

После успешной публикации сообщение удаляется из "outbox" или помечается как обработанное.

Outbox-таблица

Отдельная таблица

```sql
create table outbox_table
(
    id           bigserial   not null,
    created_dttm timestamp   not null,
    entity_name  varchar(50) not null,
    payload      text        not null,
    primary key (id)
);
```

Outbox-таблица

Нужны партиции для удаления отправленных записей

```sql
create table outbox_table
(
    id           bigserial   not null,
    created_dttm timestamp   not null default now(),
    entity_name  varchar(50) not null,
    payload      text        not null,
    primary key (id, created_dttm)
)
    partition by RANGE (created_dttm);
```

Партиции

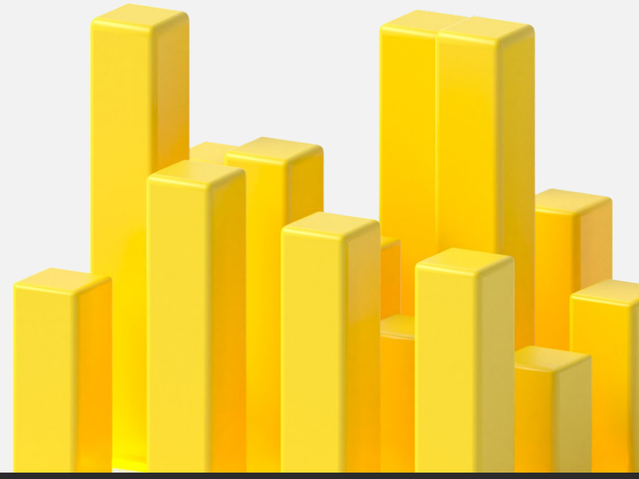
Postgresql не умеет создавать и удалять автоматически

Timescale-db гипертаблицы

https://www.timescale.com/

Cron-job в БД

Расширение pg-cron



Задача внутри приложения

```
create table if not exists outbox_table_y2024 partition of outbox_table
for values from ('2024-01-01 00:00:00') to ('2025-01-01 00:00:00');

drop table if exists outbox_table_y2024;
```

Запись в таблицу

Вручную

AOP/Proxy

События ORM/обработчик транзакций

Абстрактная реализация отправки

Простой пример отправки событий

```
public void processEvents() {
    repository.findEvents().forEach(
        entity -> kafkaTemplate.send(TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity))
    );
}
```

Проблемы

- Мы часто работаем в конкурентной среде
- В общем случае нужно не допускать повторных отправок событий
- Нельзя допускать пропуски событий для отправки

Отправка данных

Очередь в БД

- Пессимистическая блокировка
- Select for update ... Skip locked

```
select * from outbox_table
where created_dttm between current_date - interval '1 minute' and current_date
for update skip locked
```

Транзакции

Удержание транзакции

```
@Transactional
public void processEvents() {
    repository.findEvents().forEach(
        entity -> kafkaTemplate.send(
            TOPIC,
            entityConverter.getKey(entity),
            entityConverter.getMessage(entity))
    );
}
```

Транзакции

Долгая транзакция

Блокировка строк

Почему так лучше не делать?

Повышенная нагрузка на базу

Несколько транзакций

Нужен статус строки – new, locked, sent

1

```
create table outbox_table
(
    id           uuid        not null,
    created_dttm timestamp   not null,
    entity_name  varchar(50) not null,
    payload      text        not null,
    status       varchar(25) not null,
    primary key (id, created_dttm)
)
    partition by RANGE (created_dttm);
create index outbox_table_created_dttm_no_sended_idx
    on outbox_table (created_dttm) where (status = 'new');
```

2

```
update outbox_table set status = 'locked'
where id in (

    select * from outbox_table
    where status = 'new'
    for update skip locked
)
returning id;
```

Несколько транзакций

01 Блокируем записи

02 Отправляем

03 Обновляем статус

```
public void processEvents() {
    List<Entity> entities = repository.lockEvents();
    entities.forEach(
        entity -> kafkaTemplate.send(
            TOPIC,
            entityConverter.getKey(entity),
            entityConverter.getMessage(entity)));
    repository.markSent(entities);
}
```

Очередь в БД

Обновление статуса

NOTE: Нужен отдельный компонент для отправки зависших строк

1

```
update outbox_table set status = 'locked'
where id in (

    select * from outbox_table
    where status = 'new'
    for update skip locked
)
returning id;
```

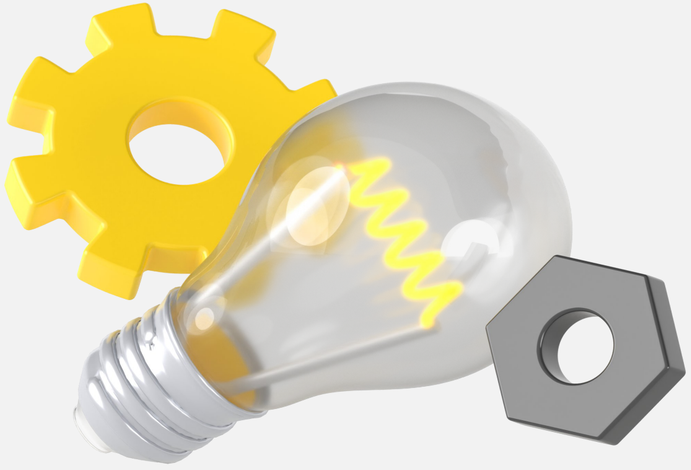
2

```
update outbox_table set status = 'sent' where id = :id
```

Проблемы очереди в БД

Долгие транзакции –
блокировки и снижение производительности

Частые обновления –
мутирующая таблица и вакуум/автовакуум



Реализация: отправка по сдвигу

Храним в отдельной таблице информацию о последней отправке

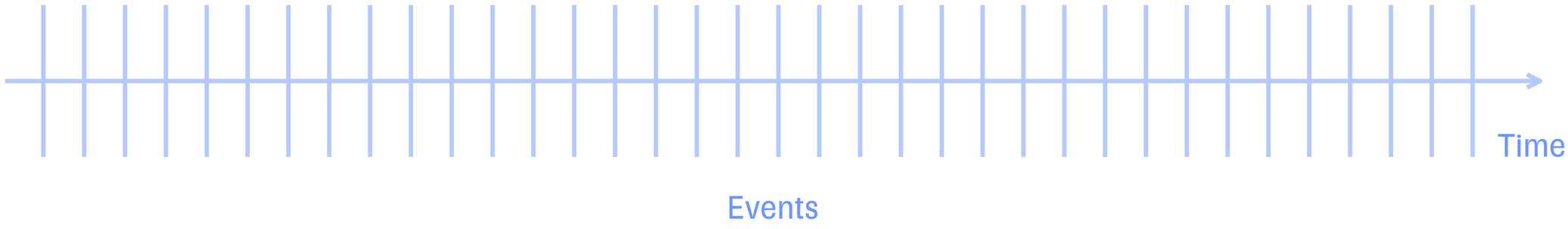
Таблица счетчиков

Отдельная таблица

```sql
create table outbox_counter
(
    counter_key     varchar(50) not null,
    counter_value   bigint      not null,
    updated_dttm    timestamp   not null,
    primary key (counter_key)
);
```

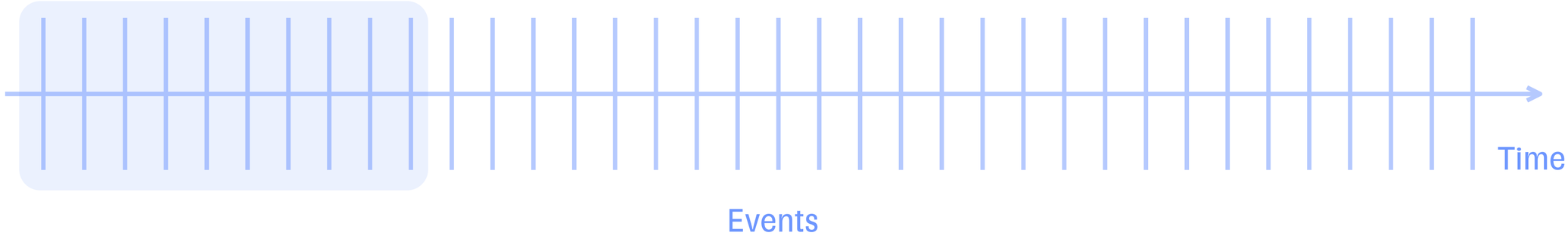
Реализация: отправка по сдвигу

События сохраняются в outbox-таблицу



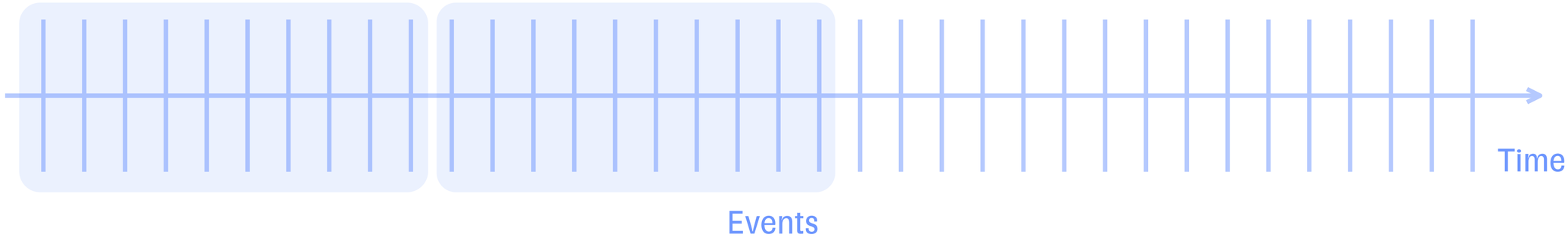
Реализация: отправка по сдвигу

Фоновое задание постепенно отправляет порции событий



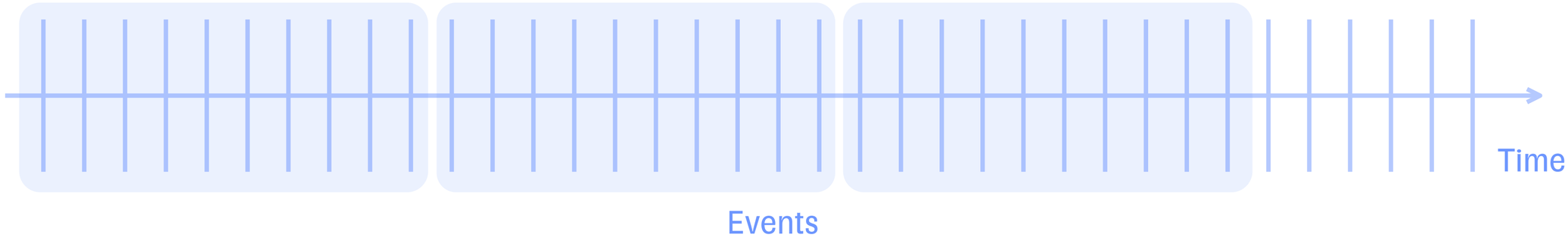
Реализация: отправка по сдвигу

Фоновое задание постепенно отправляет порции событий



Реализация: отправка по сдвигу

Фоновое задание постепенно отправляет порции событий



Реализация: отправка по сдвигу

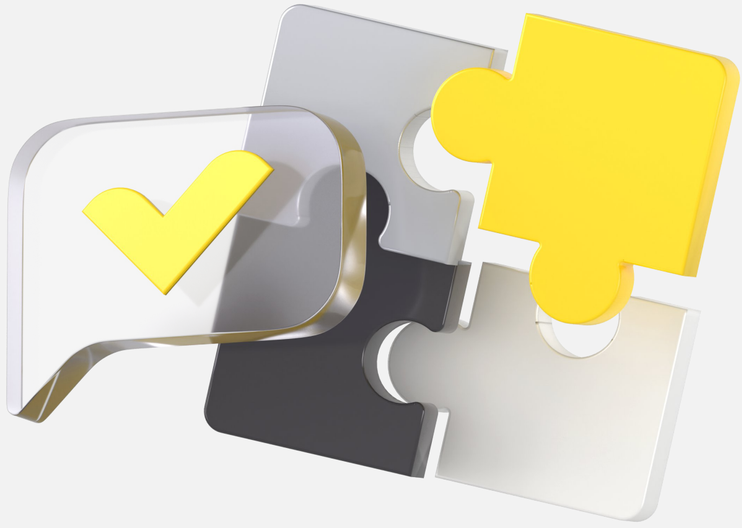
Реализация сдвига

```
public void processEvents() throws Exception {
    EntityCounter counter = entityCounterRepository.getOrCreateCounter();
    counter.increment();
    List<Entity> events = repository.findEvents(counter);
    for (Entity entity : events) {
        kafkaTemplate.send(
            TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));
    }
    entityCounterRepository.updateCounter(counter);
}
```

Сдвиг по идентификатору

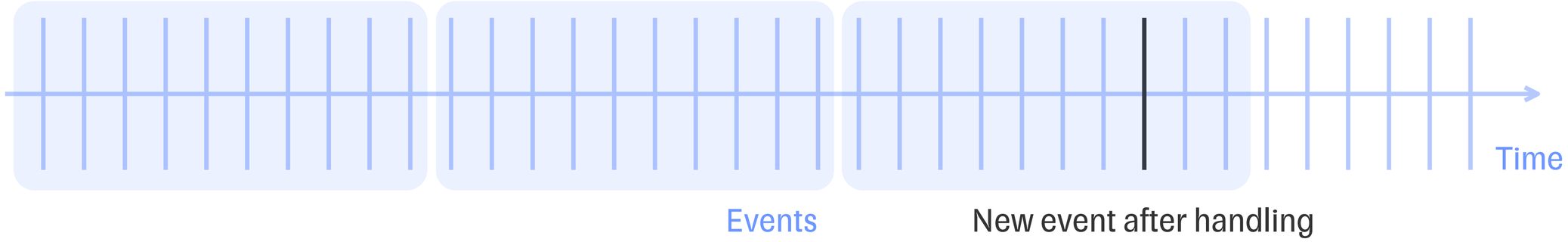
Autoincrement

Автогенерация — сиквенс, big-serial



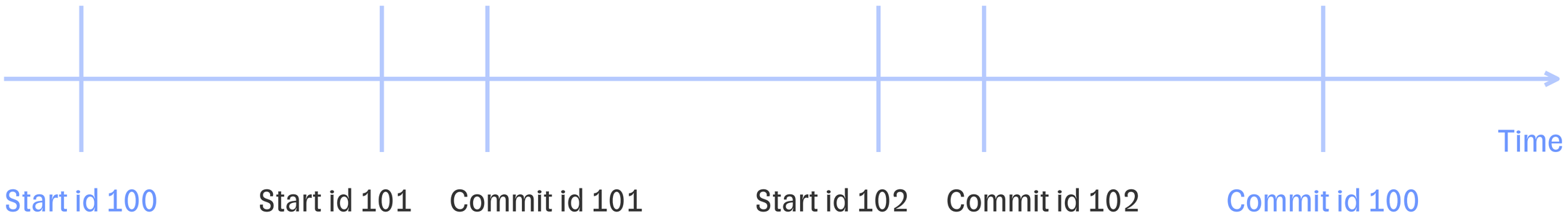
Проблема ряда идентификаторов

- Вставка с меньшим ид
- Кэширование в приложении



Задержки транзакций

- Транзакции могут выполняться в разное время



Проблема ряда идентификаторов

Уровни изоляции транзакций

Можно повысить уровень изоляции транзакций,
но это замедлит работу БД при вставке

**Распределенная блокировка
или синхронизация при вставке
в outbox**

Сдвиг по времени

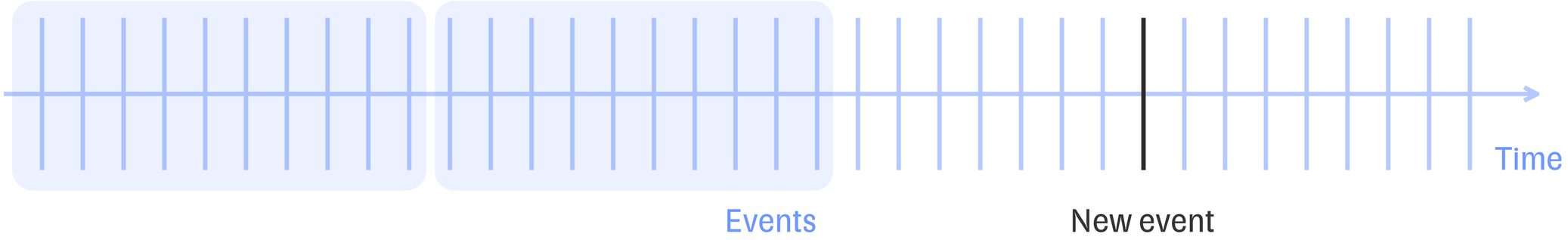
Время вставки строки

```sql
create table outbox_table
(
    id           bigserial   not null,
    created_dttm timestamp   not null default now(),
    entity_name  varchar(50) not null,
    payload      text        not null,
    primary key (id, created_dttm)
)
    partition by RANGE (created_dttm);
```

Сдвиг по времени

Отправка с задержкой

```sql
select * from outbox_table
where created_dttm between :start - interval '5' second and :finish - interval '5' second;
```



Проблемы работы со смещениями по времени

Задержки

01

Разница во времени на БД и на инстансах приложения

02

Возможные подстройки времени для синхронизации

03

Особенности отправки

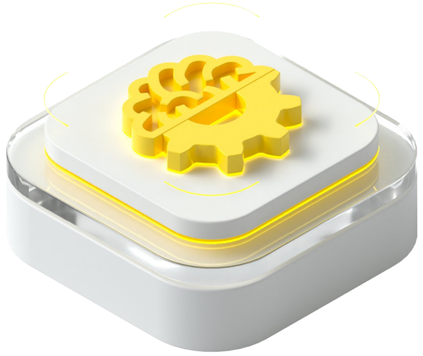
Особенности отправки в kafka

метод send()

```
public void processEvents() throws Exception {
    EntityCounter counter = entityCounterRepository.getOrCreateCounter();
    counter.increment();
    List<Entity> events = repository.findEvents(counter);
    for (Entity entity : events) {
        kafkaTemplate.send(
            TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));
    }
    entityCounterRepository.updateCounter(counter);
}
```

Прямая отправка

Метод send() отправляет сообщение не в брокер напрямую, а в буфер **RecordAccumulator** (библиотека kafka-producer)



Буфер управляется на уровне продюсера и находится **в памяти приложения**

Отправляется **пакетно,** чтобы уменьшить сетевые накладные расходы

Буфер отправки

Параметры

Linger.Ms

Как долго продюсер ждет отправки пакета

Batch.Size

Размер пакета сообщений для отправки. Если достигнут – отправка раньше чем linger.ms

Buffer.Memory

Максимальный размер буфера. Если буфер переполнится – send будет блокироваться до освобождения

Анализ отправки

Ресолвим фьючи

На одной — медленно

```
public void processEvents() throws Exception {
    EntityCounter counter = entityCounterRepository.getOrCreateCounter();
    counter.increment();
    List<Entity> events = repository.findEvents(counter);
    for (Entity entity : events) {
        ListenableFuture<SendResult<String, String>> future = kafkaTemplate.send(
            TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));

        future.addCallback(result -> log.debug("success"), ex -> {
            throw new RuntimeException(ex);
        });
        future.get();
    }
    entityCounterRepository.updateCounter(counter);
}
```

Анализ отправки

Ресолвим фьючи

Пример как делать пачкой

```
public void processEvents() {
    EntityCounter counter = entityCounterRepository.getOrCreateCounter();
    counter.increment();
    List<Entity> events = repository.findEvents(counter);
    List<Throwable> throwables = new ArrayList<>();
    List<CompletableFuture> futures = new ArrayList<>();
    for (Entity entity : events) {
        ListenableFuture<SendResult<String, String>> future = kafkaTemplate.send(
            TOPIC, entityConverter.getKey(entity), entityConverter.getMessage(entity));

        future.addCallback(result -> log.debug("success"), throwables::add);
        futures.add(future.completable());
    }
    CompletableFuture.allOf(futures.toArray(new CompletableFuture[futures.size()])).join();
    if (!throwables.isEmpty()) {
        throw new RuntimeException(throwables.get(0));
    }
    entityCounterRepository.updateCounter(counter);
}
```

Kafka транзакции

Kafka-транзакции – гарантии acid для kafka

```
public void processEvents() {
    EntityCounter counter = entityCounterRepository.getOrCreateCounter();
    counter.increment();
    List<Entity> events = repository.findEvents(counter);

    producer.initTransactions();
    try {
        producer.beginTransaction();
        for (Entity entity : events) {
            kafkaProducer.send(
                new ProducerRecord(TOPIC,
                entityConverter.getKey(entity), entityConverter.getMessage(entity)));

        }
        kafkaProducer.commitTransaction();
    } catch (ProducerFencedException | OutOfOrderSequenceException | AuthorizationException e) {
        throw e;
    } catch (KafkaException | TimeoutException | InterruptException e) {
        producer.abortTransaction();
    } catch (Exception e) {
        log.error("Unexpected exception: {}", e.getMessage());
        producer.abortTransaction();
    }
}
```

Оптимизации продюсера Apache Kafka

Доклад: Григорий Кошелев — Когда всё пошло по Кафке 2: Разгоняем продьюсеров

https://www.youtube.com/watch?v=zMLfxztAVlo

Масштабирование

Выполнение на одном узле

Нужна распределенная блокировка

Shedlock

Если используем java – хороший выбор shedlock

ShedLock — это библиотека, которая используется в распределённых системах для обеспечения, что запланированные задачи (cron jobs) выполняются только одним узлом в кластере в любой момент времени. Это решает проблему дублирования выполнения задач, когда несколько экземпляров приложения запускают одну и ту же задачу одновременно.

Альтернатива для spring – spring integration: leader election

Параллельная отправка

Может не хватить **пропускной способности** отправки если выполнение всегда будет на единственной ноде

01

Возникнет потребность в **параллельности** отправки

02

Но важно **избегать дублирования** и гарантировать последовательность отправляемых событий

03

Партиции

В Kafka есть партиции

Нужно обеспечить на продюсере выбор правильный выбор партиции для отправки

Kafka-jdbc-connector

Kafka-jdbc-connector

kafka-jdbc-source-connector – выбирает данные из БД и отправляет в kafka

Можно развернуть в инфраструктуре kafka-connect или поднять образ отдельно

Настройки вынесены в конфигурацию

Kafka-jdbc-connector

Несколько режимов работы

Increment

Ограничения
по столбцу с sequence-последовательностью

Timestamp

Ограничения по столбцу
с меткой даты и времени

Timestamp + Increment

Комбинированный режим для отслеживания добавления новых строк и изменения существующих

Custom

На основе кастомного sql-запроса

Bulk-mode

Полная загрузка
таблицы

Kafka-jdbc-connector

Ограничения

- Хранение смещений в топиках Kafka
- Возможно не достаточная наблюдаемость

Итоги

Контекст

Отправка событий вместе
с изменением в БД –
частая задача

Существующие решения

- На рынке есть **технологии,** решающие задачу (CDC/kafka-connect)
- Но если они не подходят — приходится **решать задачу самим**

Простое решение

Синхронная отправка

Удовлетворяет потребностям при низких требованиях к скорости и надежности

Transactional outbox

- Хороший способ решить задачу
- Реализовывать нужно с учетом требований и текущих ограничений систем

Т БАНК

Спасибо!