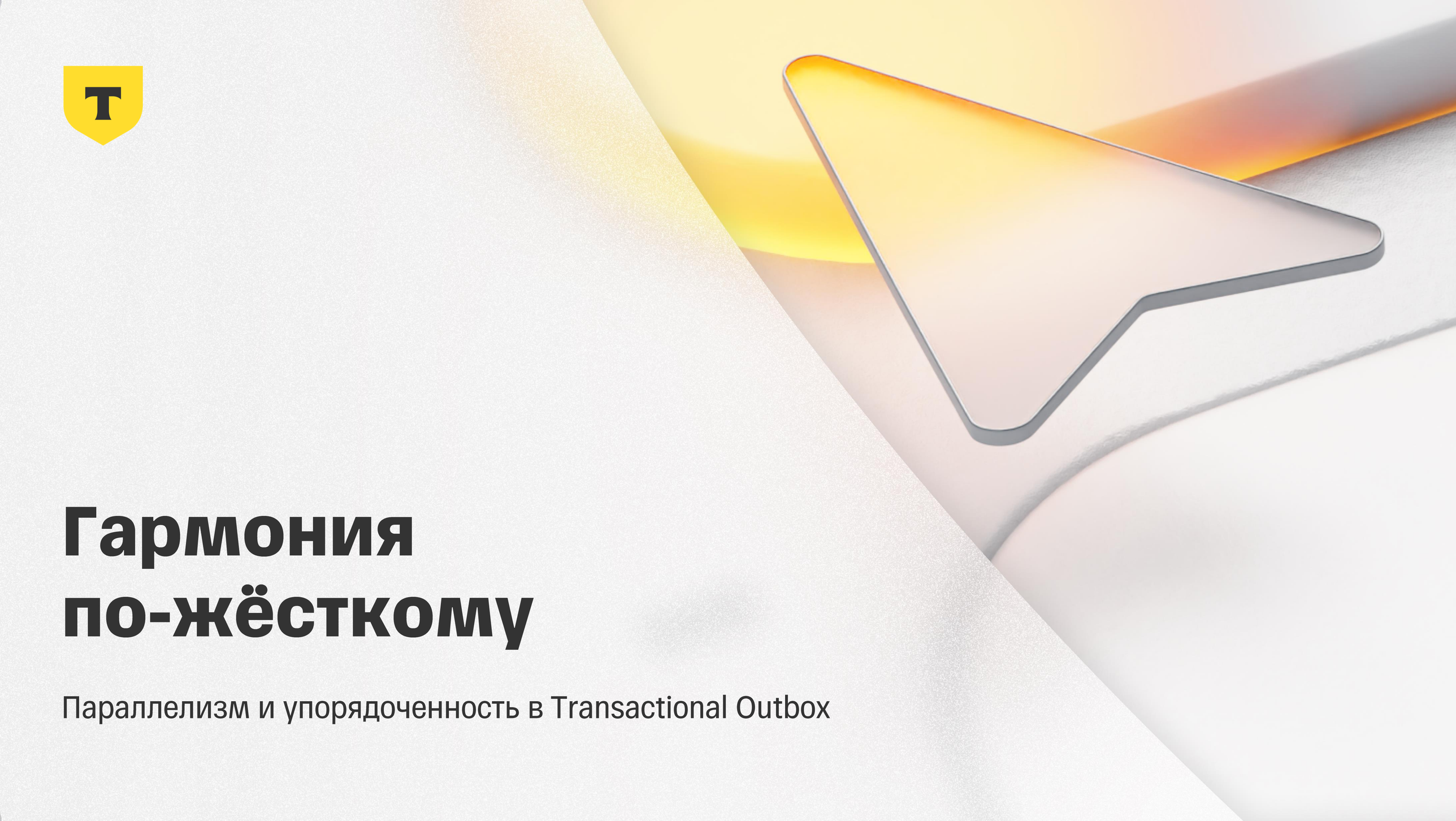


Гармония по-жёсткому

Параллелизм и упорядоченность в Transactional Outbox

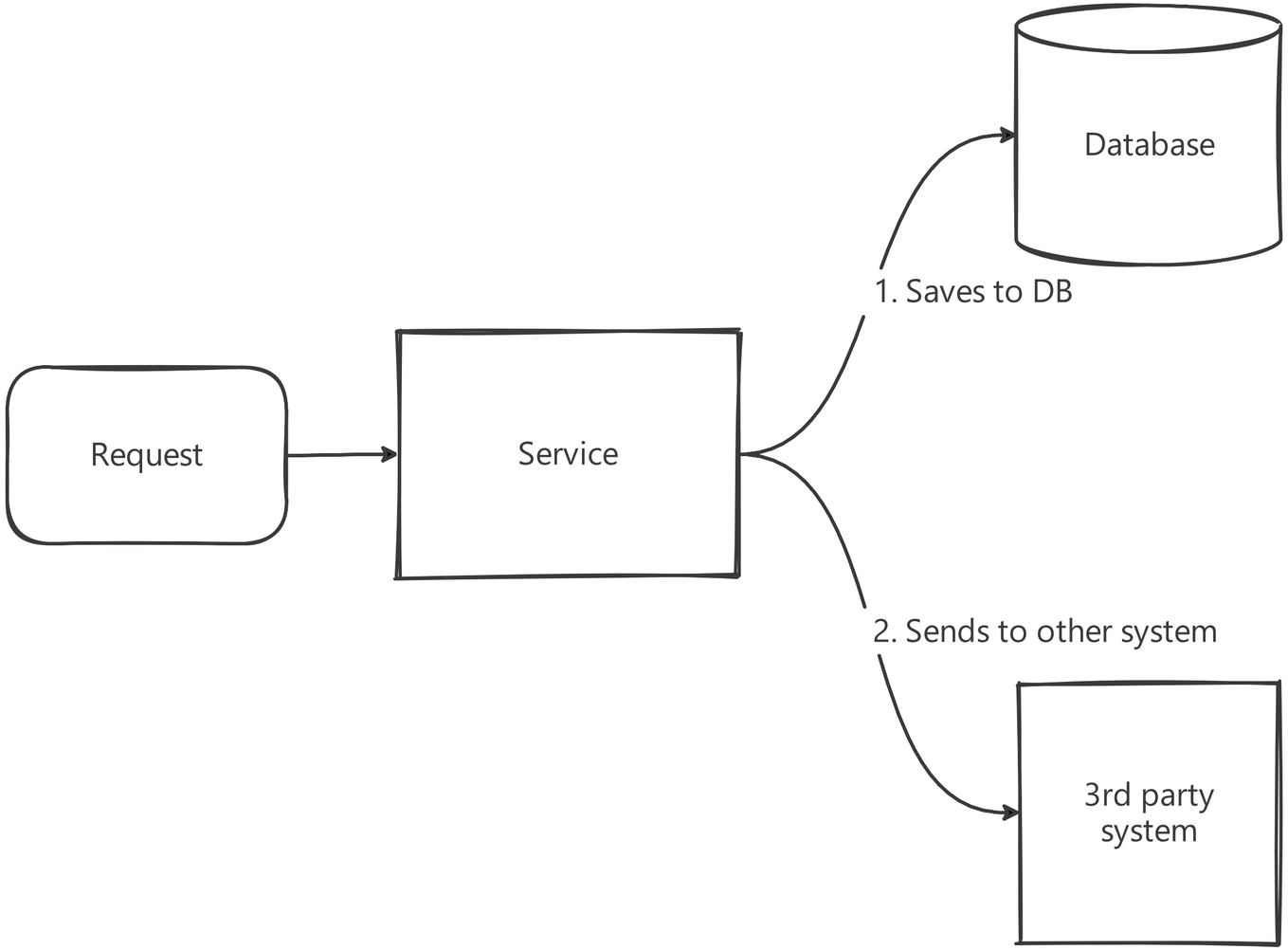
Виталий Сушков

.NET developer | T-Bank

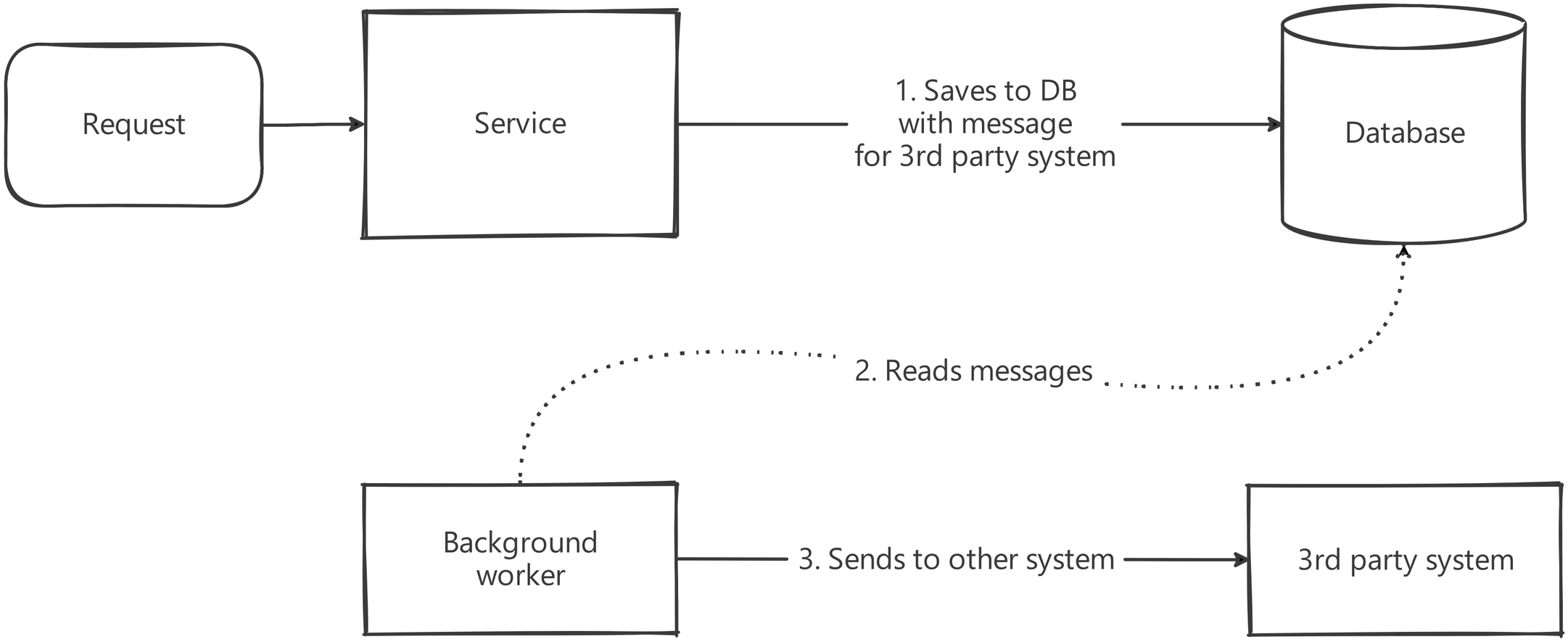
- Разрабатываю ПО с 2010 года
- С 2018 года использую .NET
- С 2022 года работаю в Т-Банке
- Автор библиотеки фоновых задач Jobby

Рассмотрим сценарий

- Выполнить транзакцию в БД
- Вызвать другую систему
- Гарантированно должны быть выполнены либо обе операции, либо ни одной

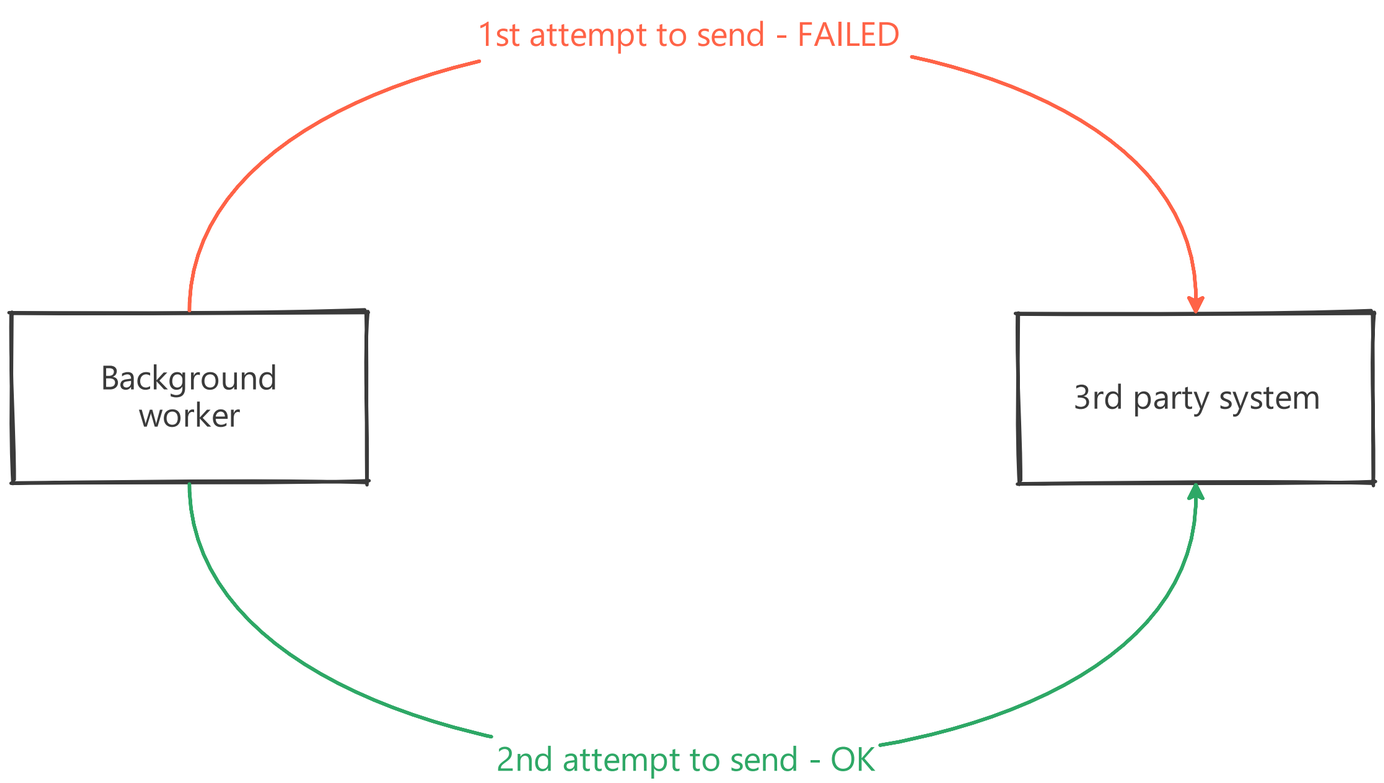


Transactional Outbox



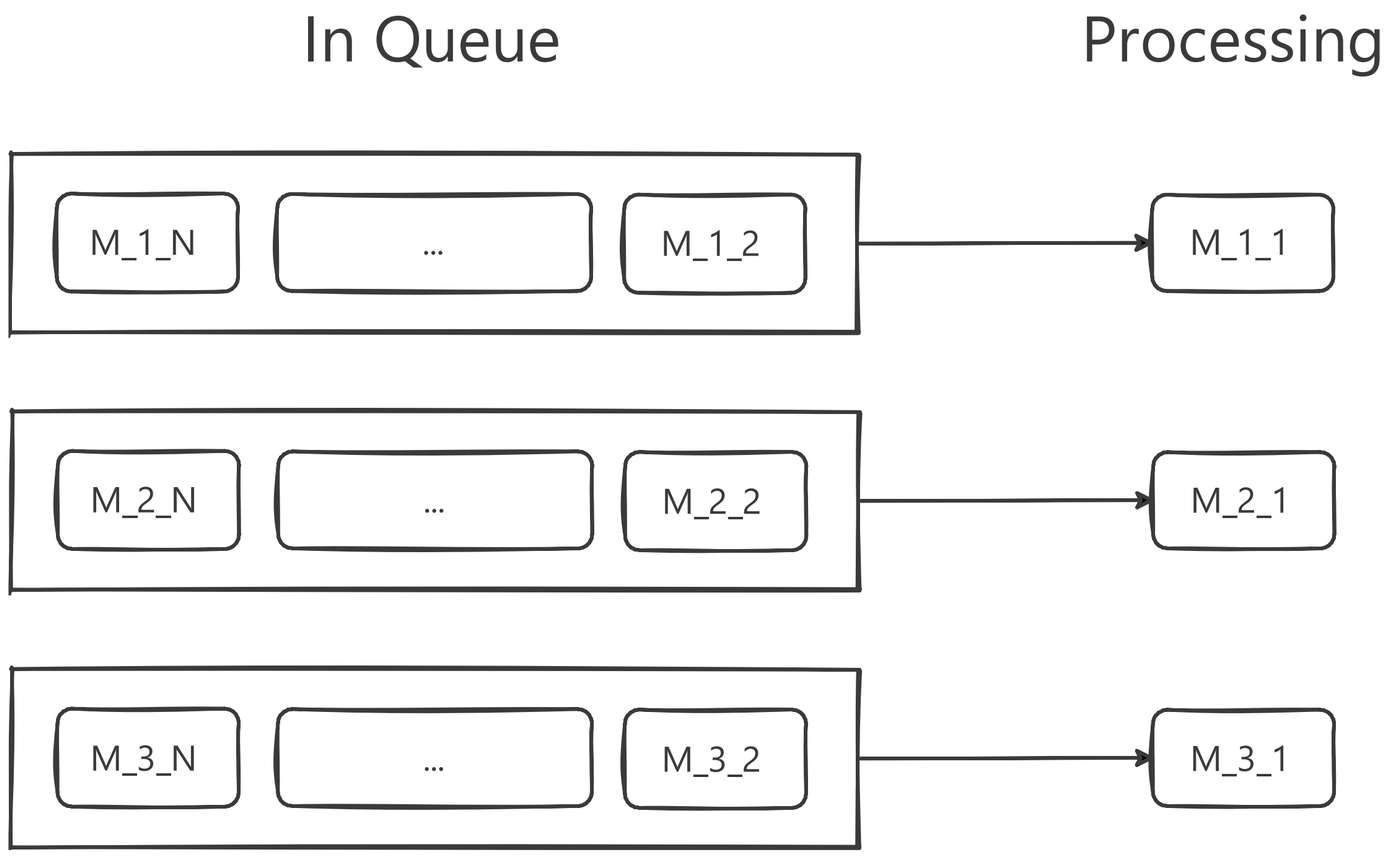
Ожидаемые гарантии

- **At least once**
- Обработка по порядку для связанных сообщений
- Параллельная обработка для не связанных сообщений



Ожидаемые гарантии

- At least once
- **Обработка по порядку для связанных сообщений**
- **Параллельная обработка для не связанных сообщений**



В докладе

- **Два принципиально разных подхода к реализации Transactional Outbox**

В докладе

- Два принципиально разных подхода к реализации Transactional Outbox
- **Эффективная реализация параллелизма и упорядоченности в обоих подходах**

В докладе

- Два принципиально разных подхода к реализации Transactional Outbox
- Эффективная реализация параллелизма и упорядоченности в обоих подходах
- **Погружение в неочевидные особенности PostgreSQL**

В докладе

- Два принципиально разных подхода к реализации Transactional Outbox
- Эффективная реализация параллелизма и упорядоченности в обоих подходах
- Погружение в неочевидные особенности PostgreSQL
- **Асинхронная архитектура фоновых сервисов на .NET**

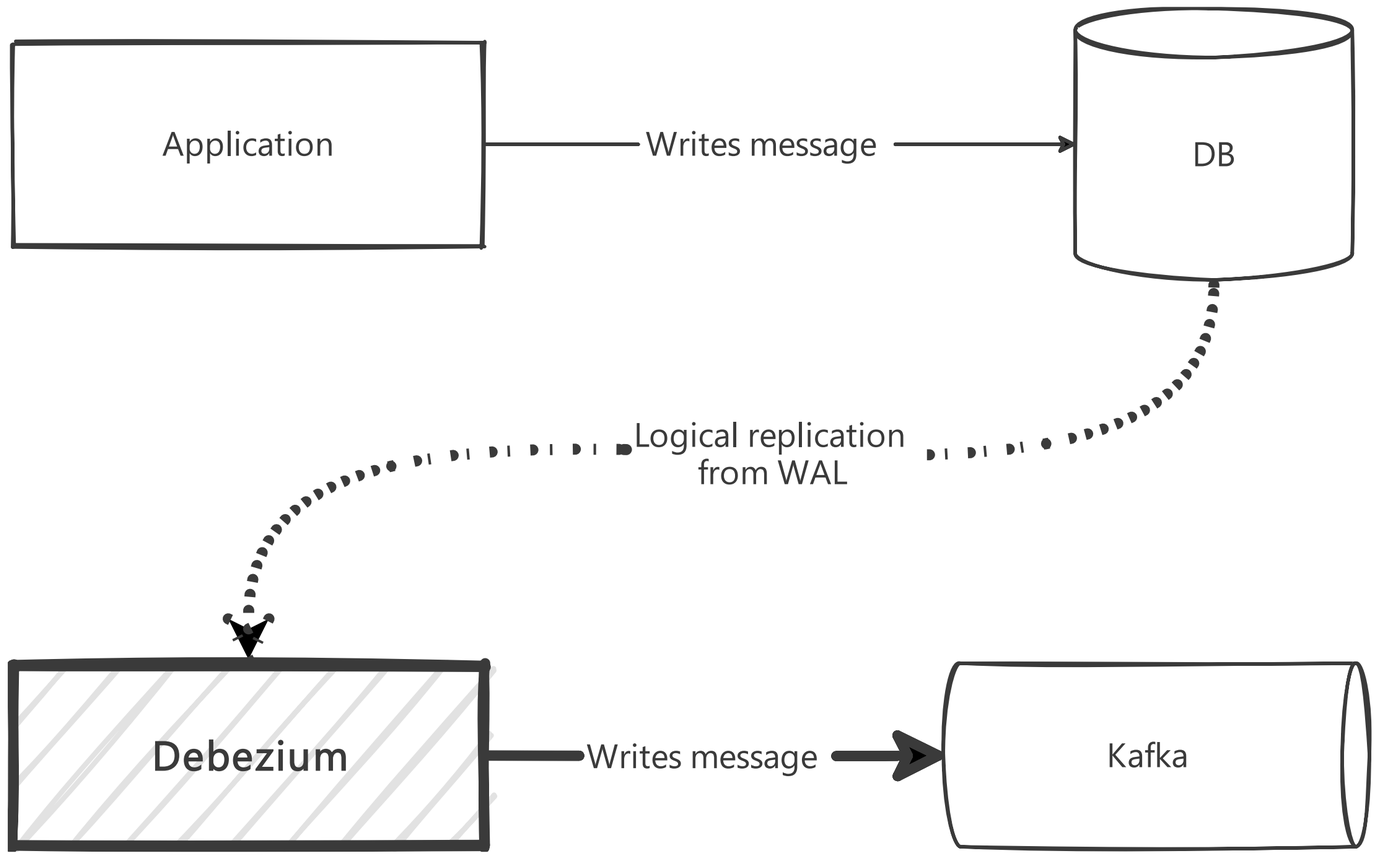
Подходы к реализации Outbox

Создание записей для Outbox через ORM

```
// Бизнес-логика
var order = await _dbContext.Orders
                            .FirstOrDefaultAsync(x => x.Id == orderId);
order.Cancel(reason);

// Сообщение для outbox
var orderCancelledEvent = new OrderCancelledEvent(orderId, reason);
var outboxMessage = _outboxMessageFactory.Create(orderCancelledEvent);
_dbContext.Add(outboxMessage);

// Сохранить в одной транзакции
await _dbContext.SaveChangesAsync();
```

CDC / Debezium

- Готовое решение
- Читает записи из WAL
- Отправляет в Kafka

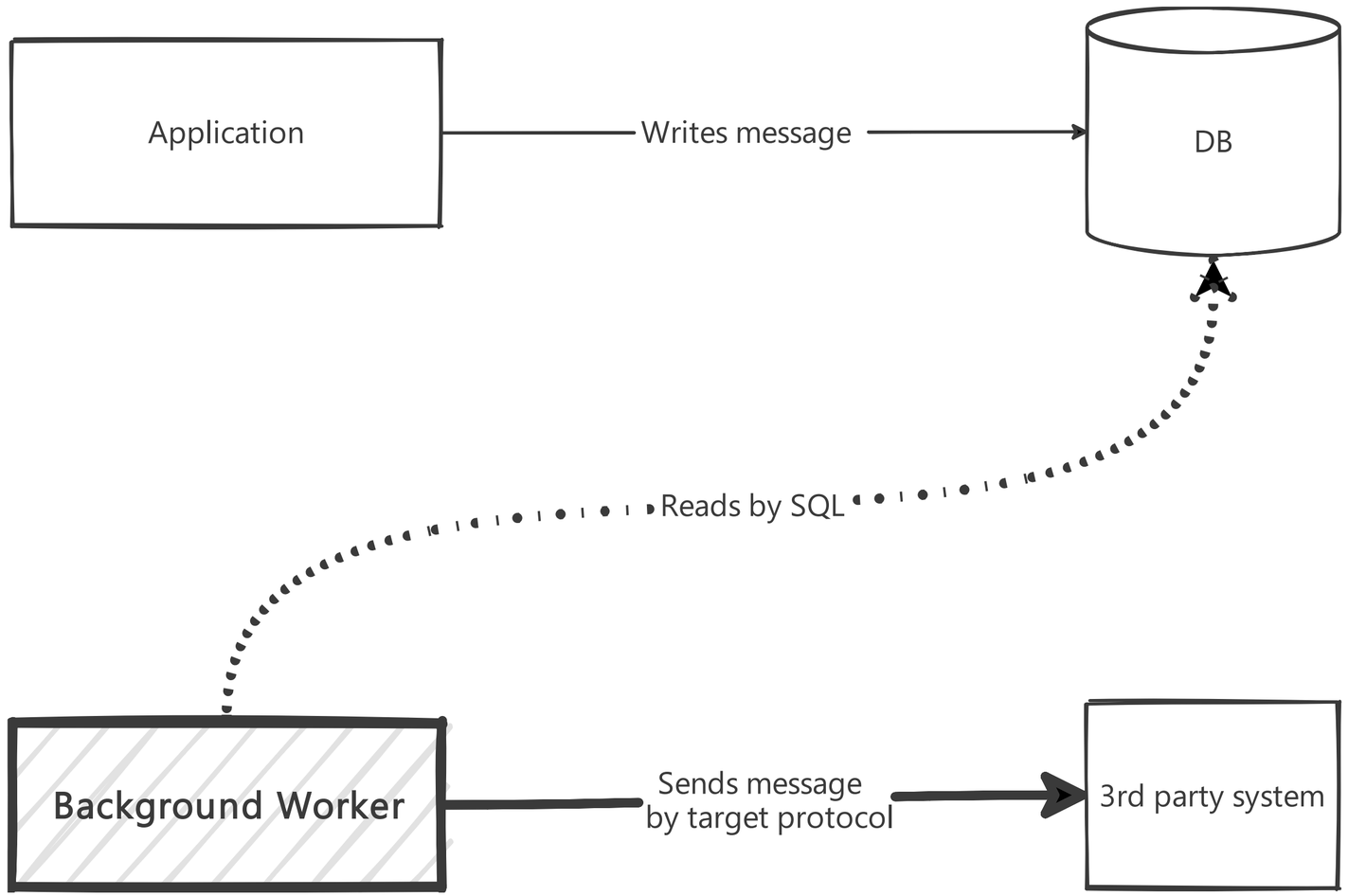


Возможные сложности с Debezium

- **Инфраструктурные и операционные сложности и накладные расходы** (Debezium, Kafka Connect, Avro, ...)
- **Низкий уровень гибкости при обработке** (фиксированный формат сообщений, сложнее сделать приоритезацию, агрегацию батчей, сложную retry-логику)
- **Требует доступа к слоту репликации**
- **Риск роста WAL при нештатных ситуациях**

Чтение через SQL-запросы

- Своя реализация на привычном стеке
- Получение записей SQL-запросами по PULL-модели
- Обработка записей целевым образом



Подход на основе статусов

Поле **status** (New / Processing / ...)

VS

Подход на основе смещений (Kafka-Inspired)

Поле **num (монотонно возрастающее)**

Подход на основе статусов

Поле **status** (New / Processing / ...)

```
-- Чтение записей для обработки
SELECT * FROM outbox
  WHERE status = New
  ORDER BY created_at FOR UPDATE;

UPDATE outbox SET status = Processing;
WHERE id = :selected

-- После обработки
DELETE FROM outbox WHERE id = ...
```

VS

Подход на основе смещений

Поле **num** (монотонно возрастающее)

Подход на основе статусов

Поле **status** (New / Processing / ...)

```
-- Чтение записей для обработки
SELECT * FROM outbox
  WHERE status = New
  ORDER BY created_at FOR UPDATE;

UPDATE outbox SET status = Processing;
WHERE id = :selected

-- После обработки
DELETE FROM outbox WHERE id = ...
```

VS

Подход на основе смещений

Поле **num** (монотонно возрастающее)

Подход на основе статусов

Поле status (New / Processing / ...)

```
-- Чтение записей для обработки
SELECT * FROM outbox
  WHERE status = New
  ORDER BY created_at FOR UPDATE;

UPDATE outbox SET status = Processing;
WHERE id = :selected

-- После обработки
DELETE FROM outbox WHERE id = ...
```

VS

Подход на основе смещений

Поле num (монотонно возрастающее)

```
-- Чтение записей для обработки
SELECT * FROM outbox
  WHERE num > :lastProcessedNum
  ORDER BY num;

-- После обработки
UPDATE consumer_offset
  SET offset = :processedMessage.Num;
```

Подход на основе статусов

Поле status (New / Processing / ...)

```
-- Чтение записей для обработки
SELECT * FROM outbox
  WHERE status = New
  ORDER BY created_at FOR UPDATE;

UPDATE outbox SET status = Processing;
WHERE id = :selected

-- После обработки
DELETE FROM outbox WHERE id = ...
```

VS

Подход на основе смещений

Поле num (монотонно возрастающее)

```
-- Чтение записей для обработки
SELECT * FROM outbox
  WHERE num > :lastProcessedNum
  ORDER BY num;

-- После обработки
UPDATE consumer_offset
  SET offset = :processedMessage.Num;
```

Подход на основе статусов

Нагрузка на БД
(SELECT + UPDATE + DELETE)

VS

Подход на основе смещений

Эффективная работа с БД
(Таблица сообщений INSERT ONLY)

Подход на основе статусов

- Нагрузка на БД (SELECT + UPDATE + DELETE)
- Сообщения не связаны между собой
- Просто распараллеливается и масштабируется

VS

Подход на основе смещений

- Эффективная работа с БД (Таблица сообщений INSERT ONLY)
- Сообщения жёстко связаны
- Сложно распараллеливать и масштабировать

Подход на основе статусов

- Нагрузка на БД (SELECT + UPDATE + DELETE)
- Сообщения не связаны между собой
- Просто распараллеливается и масштабируется
- К параллелизму сложно добавить упорядоченность

VS

Подход на основе смещений

- Эффективная работа с БД (Таблица сообщений INSERT ONLY)
- Сообщения жёстко связаны
- Сложно распараллеливать и масштабировать
- Упорядоченность из коробки

Подход на основе статусов

- Нагрузка на БД (SELECT + UPDATE + DELETE)
- Сообщения не связаны между собой
- Просто распараллеливается и масштабируется
- К параллелизму сложно добавить упорядоченность

VS

Подход на основе смещений

- Эффективная работа с БД (Таблица сообщений INSERT ONLY)
- Сообщения жёстко связаны
- Сложно распараллеливать и масштабировать
- Упорядоченность из коробки

Подход на основе статусов

Примеры применения

Доклад о Jobby с DotNext 2025 «Ускоряя невидимое: разработка эффективной библиотеки фоновых задач»

Hangfire

Очень ограниченные возможности упорядоченной обработки при сохранении параллелизма

01

Quartz

Очень ограниченные возможности упорядоченной обработки при сохранении параллелизма

02

CAP

Параллелизм и упорядоченность не поддерживаются одновременно

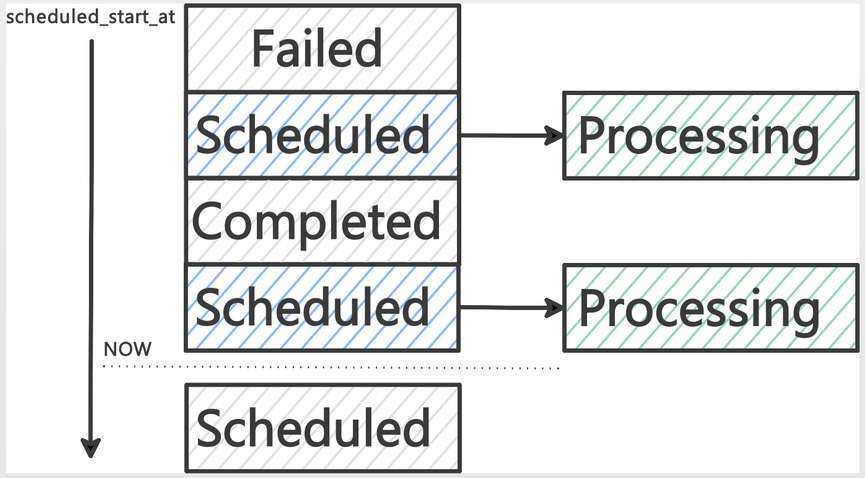
03

Jobby

Возможно совмещать параллельную и упорядоченную обработку

04

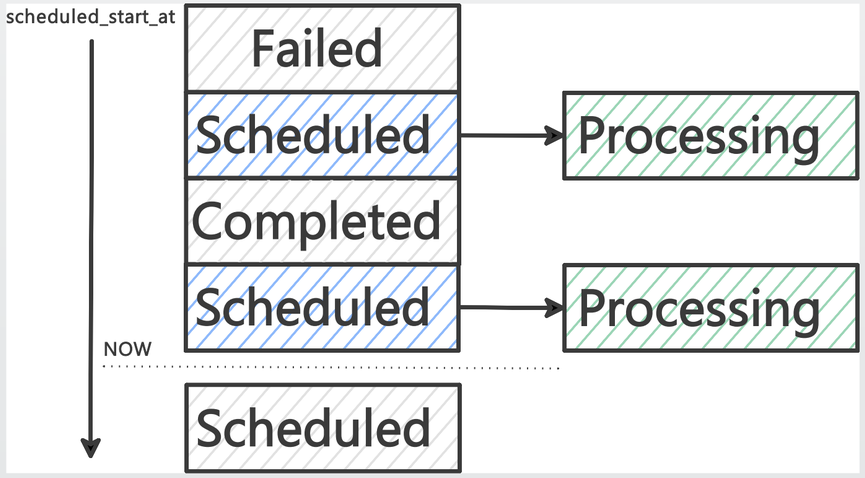
Запрос для получения готовых к запуску задач



TakeJobs...

```sql
WITH ready_jobs AS (
    SELECT id FROM jobby_jobs
    WHERE
        "status" = Scheduled
        AND scheduled_start_at <= now()
    ORDER BY scheduled_start_at
    LIMIT :batchSize
    FOR UPDATE SKIP LOCKED
)
UPDATE jobby_jobs
SET
    "status" = Processing,
    started_count = started_count + 1
WHERE id IN (SELECT id FROM ready_jobs)
RETURNING *;

-- Составной индекс (status, scheduled_start_at)
```

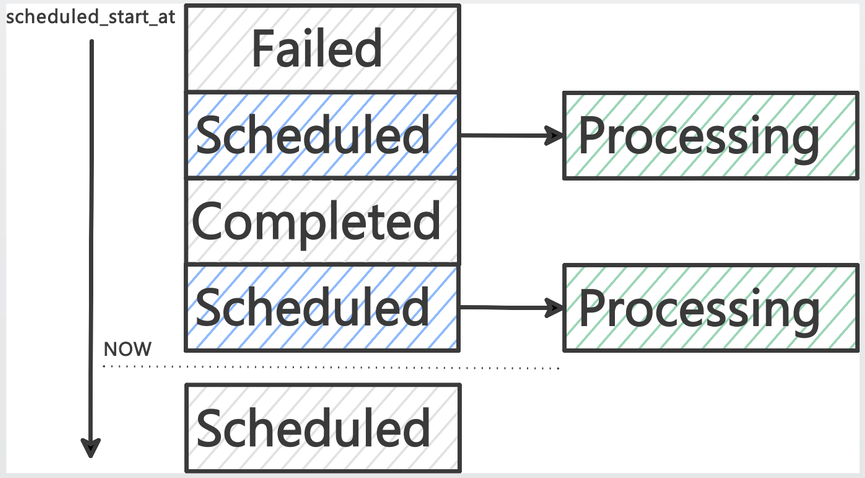
Запрос для получения готовых к запуску задач



TakeJobs...

```sql
WITH ready_jobs AS (
    SELECT id FROM jobby_jobs
    WHERE
        "status" = Scheduled
        AND scheduled_start_at <= now()
    ORDER BY scheduled_start_at
    LIMIT :batchSize
    FOR UPDATE SKIP LOCKED
)
UPDATE jobby_jobs
SET
    "status" = Processing,
    started_count = started_count + 1
WHERE id IN (SELECT id FROM ready_jobs)
RETURNING *;

-- Составной индекс (status, scheduled_start_at)
```

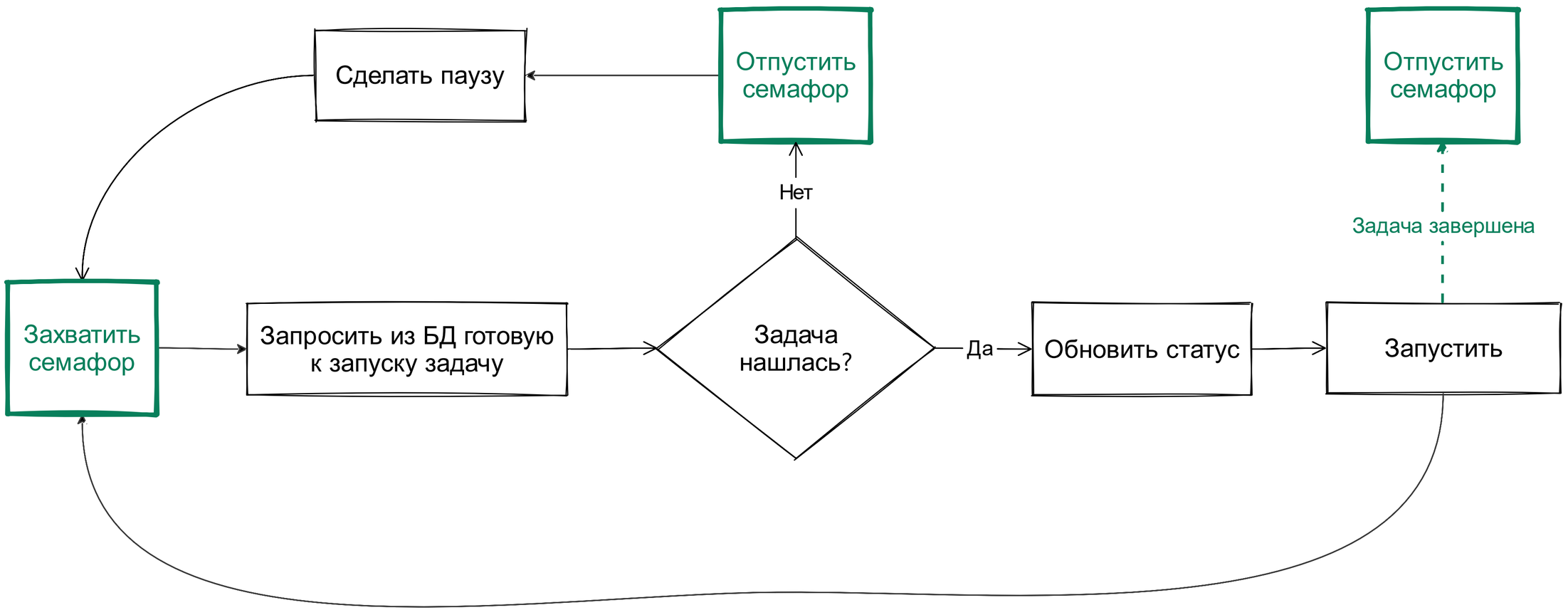
Запрос для получения готовых к запуску задач



TakeJobs...

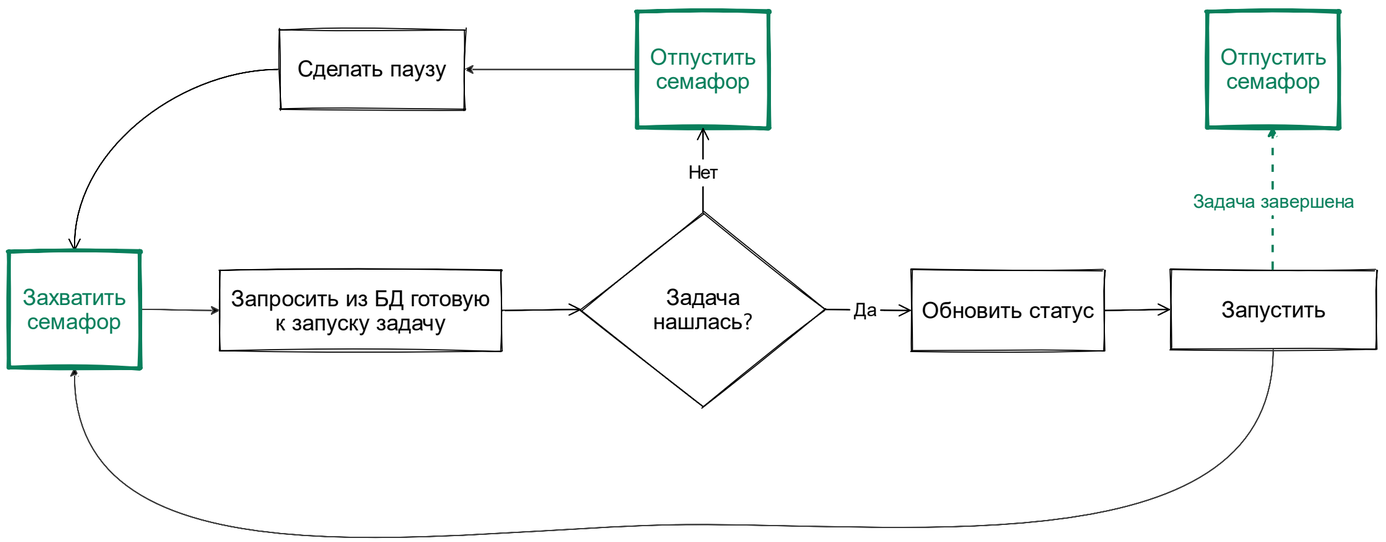
```sql
WITH ready_jobs AS (
    SELECT id FROM jobby_jobs
    WHERE
        "status" = Scheduled
        AND scheduled_start_at <= now()
    ORDER BY scheduled_start_at
    LIMIT :batchSize
    FOR UPDATE SKIP LOCKED
)
UPDATE jobby_jobs
SET
    "status" = Processing,
    started_count = started_count + 1
WHERE id IN (SELECT id FROM ready_jobs)
RETURNING *;

-- Составной индекс (status, scheduled_start_at)
```

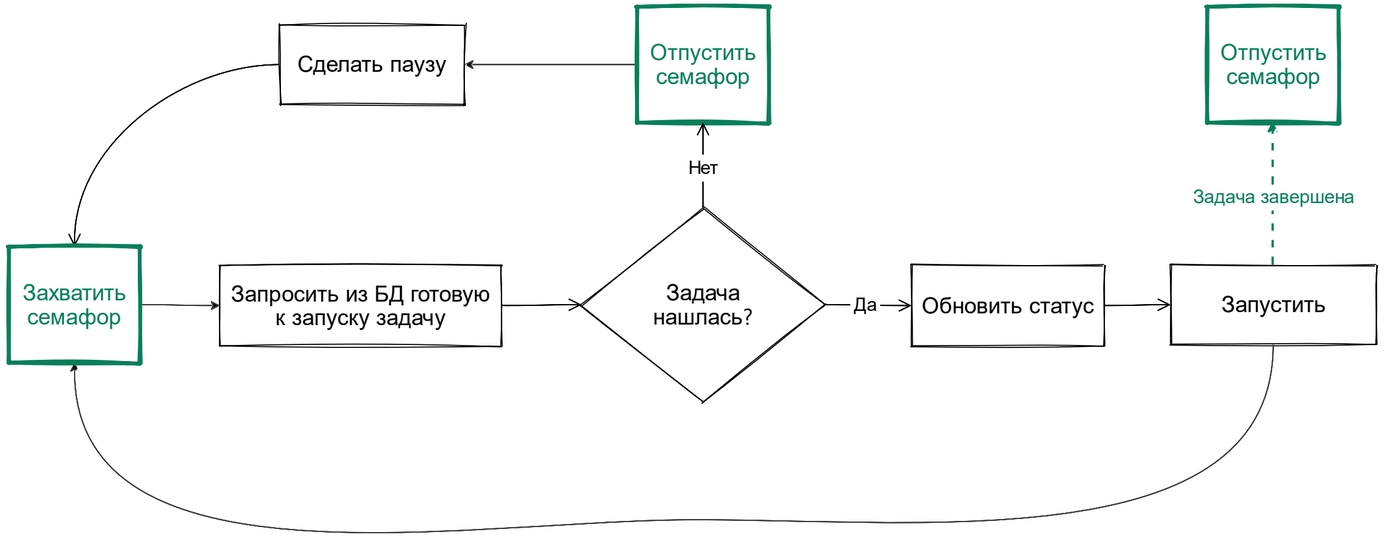
Алгоритм получения и запуска задач



Параллелизм

- Параллельное выполнение внутри одного узла
- Естественным образом масштабируется на N узлов





Группы последовательного выполнения

GroupId

```
// У задачи можно указать ID группы
await jobbyClient.EnqueueCommandAsync(command, new JobOpts
{
    SerializableGroupId = command.OrderId
});

// Задачи с одинаковой группы выполняются строго по одной

// Задачи с разными группами или без группы выполняются параллельно
```

Блокировка по факту статуса

Группа считается заблокированной если:

- В группе существует задача в статусе **Processing**
- В группе **упавшая задача** которая должна блокировать группу

Вычисляемое поле is_group_locker

```
ALTER TABLE jobby_jobs
    ADD COLUMN IF NOT EXISTS is_group_locker BOOLEAN GENERATED ALWAYS AS (
        -- Если есть группа
        group_id IS NOT NULL

        -- И задача в работе ИЛИ (упала И должна блокировать группу при падении)
        AND (
            "status" = PROCESSING
            OR (
                lock_group_if_failed = TRUE
                AND (
                    "status" = FAILED
                    OR "status" = SCHEDULED AND started_count > 0
                )
            )
        )
    ) STORED;
```

Вычисляемое поле is_group_locker

```
ALTER TABLE jobby_jobs
    ADD COLUMN IF NOT EXISTS is_group_locker BOOLEAN GENERATED ALWAYS AS (
        -- Если есть группа
        group_id IS NOT NULL

        -- И задача в работе ИЛИ (упала И должна блокировать группу при падении)
        AND (
            "status" = PROCESSING
            OR (
                lock_group_if_failed = TRUE
                AND (
                    "status" = FAILED
                    OR "status" = SCHEDULED AND started_count > 0
                )
            )
        )
    ) STORED;
```

Вычисляемое поле is_group_locker

```
ALTER TABLE jobby_jobs
    ADD COLUMN IF NOT EXISTS is_group_locker BOOLEAN GENERATED ALWAYS AS (
        -- Если есть группа
        group_id IS NOT NULL

        -- И задача в работе ИЛИ (упала И должна блокировать группу при падении)
        AND (
            "status" = PROCESSING
            OR (
                lock_group_if_failed = TRUE
                AND (
                    "status" = FAILED
                    OR "status" = SCHEDULED AND started_count > 0
                )
            )
        )
    ) STORED;
```

Ограничение уникальности блокировки группы

Partial-индекс на group_id который не позволит иметь более одного локера на одну группу

Unique Lock ×

```
-- Индекс, который не позволит иметь
-- более одного локера в группе

CREATE UNIQUE INDEX
    ON jobby_jobs(group_id)
    WHERE is_group_locker = TRUE;
```

Получение задач с учётом блокировки

```
WITH candidates AS (
  SELECT * FROM jobby_jobs c
  WHERE
      c.status = Scheduled AND c.scheduled_start_at <= now()
      AND (
          c.group_id IS NULL OR (
              NOT EXISTS (
                  SELECT 1 FROM jobby_jobs jl
                  WHERE
                      jl.group_id = c.group_id
                      AND jl.is_group_locker = TRUE
                      AND jl.id != c.id
              )
              AND pg_try_advisory_xact_lock(hashtext(c.group_id)) = TRUE
          )
      )
  ORDER BY scheduled_start_at LIMIT batch_size FOR UPDATE SKIP LOCKED
), ...
```

Получение задач с учётом блокировки

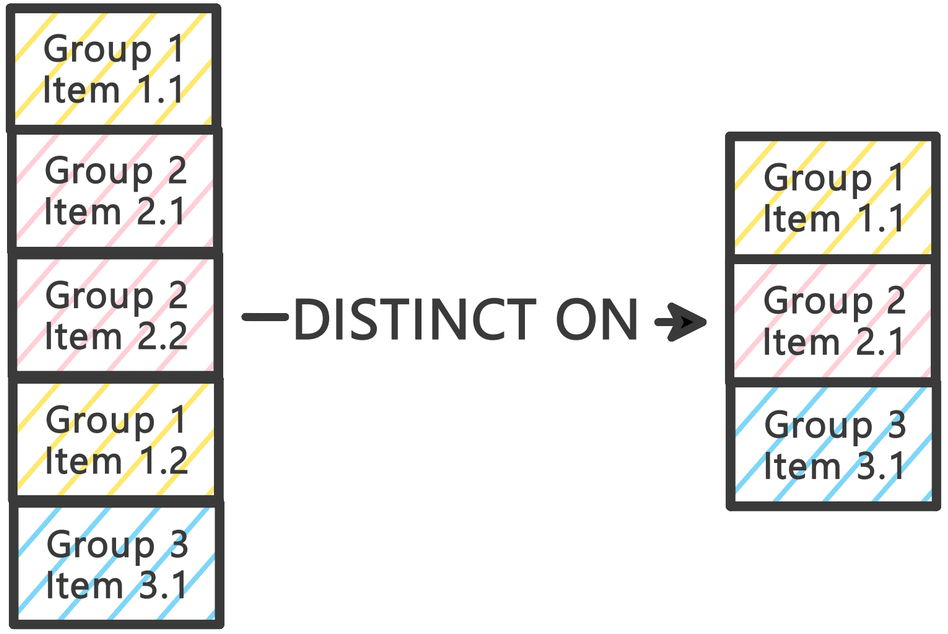
```
WITH candidates AS (
  SELECT * FROM jobby_jobs c
  WHERE
      c.status = Scheduled AND c.scheduled_start_at <= now()
      AND (
          c.group_id IS NULL OR (
              NOT EXISTS (
                  SELECT 1 FROM jobby_jobs jl
                  WHERE
                      jl.group_id = c.group_id
                      AND jl.is_group_locker = TRUE
                      AND jl.id != c.id
              )
              AND pg_try_advisory_xact_lock(hashtext(c.group_id)) = TRUE
          )
      )
  ORDER BY scheduled_start_at LIMIT batch_size FOR UPDATE SKIP LOCKED
), ...
```

Получение задач с учётом блокировки

```
WITH candidates AS (
  SELECT * FROM jobby_jobs c
  WHERE
      c.status = Scheduled AND c.scheduled_start_at <= now()
      AND (
          c.group_id IS NULL OR (
              NOT EXISTS (
                  SELECT 1 FROM jobby_jobs jl
                  WHERE
                      jl.group_id = c.group_id
                      AND jl.is_group_locker = TRUE
                      AND jl.id != c.id
              )
              AND pg_try_advisory_xact_lock(hashtext(c.group_id)) = TRUE
          )
      )
  ORDER BY scheduled_start_at LIMIT batch_size FOR UPDATE SKIP LOCKED
), ...
```

Получение задач с учётом блокировки

```
WITH
  candidates AS (...),

  -- Из каждой группы берём по одной задаче
  candidates_with_group_unique (
    SELECT DISTINCT ON (group_id), ...
    FROM candidates
    ORDER BY group_id, scheduled_start_at
    WHERE group_id IS NOT NULL
  )

UPDATE jobby_jobs
SET "status" = Processing, started_count = started_count + 1
WHERE id IN (…)
RETURNING *;
```



Получение задач с учётом блокировки

```
WITH
  candidates AS (...),
  candidates_with_group_unique AS (...)

-- Отобранным задачам обновляем статус и берём в работу

UPDATE jobby_jobs
SET "status" = Processing, started_count = started_count + 1
WHERE id IN (
  SELECT id FROM candidates WHERE group_id IS NULL
  UNION
  SELECT id FROM candidates_with_group_unique
)
RETURNING *;
```

Рекомендательные блокировки

https://postgrespro.ru/docs/postgresql/12/explicit-locking#ADVISORY-LOCKS

https://habr.com/ru/companies/tensor/articles/488024/

pg_try_advisory_xact_lock

- Блокировкой владеет транзакция
- При завершении транзакции снимается автоматически
- Безопасно использовать совместно с pgbouncer

pg_try_advisory_lock

- Блокировкой владеет сеанс
- ⚠ Нужно делать unlock явно, иначе будет висеть до закрытия соединения
- ⚠ Опасно использовать с pgbouncer в transactional mode

Аналоги в других СУБД

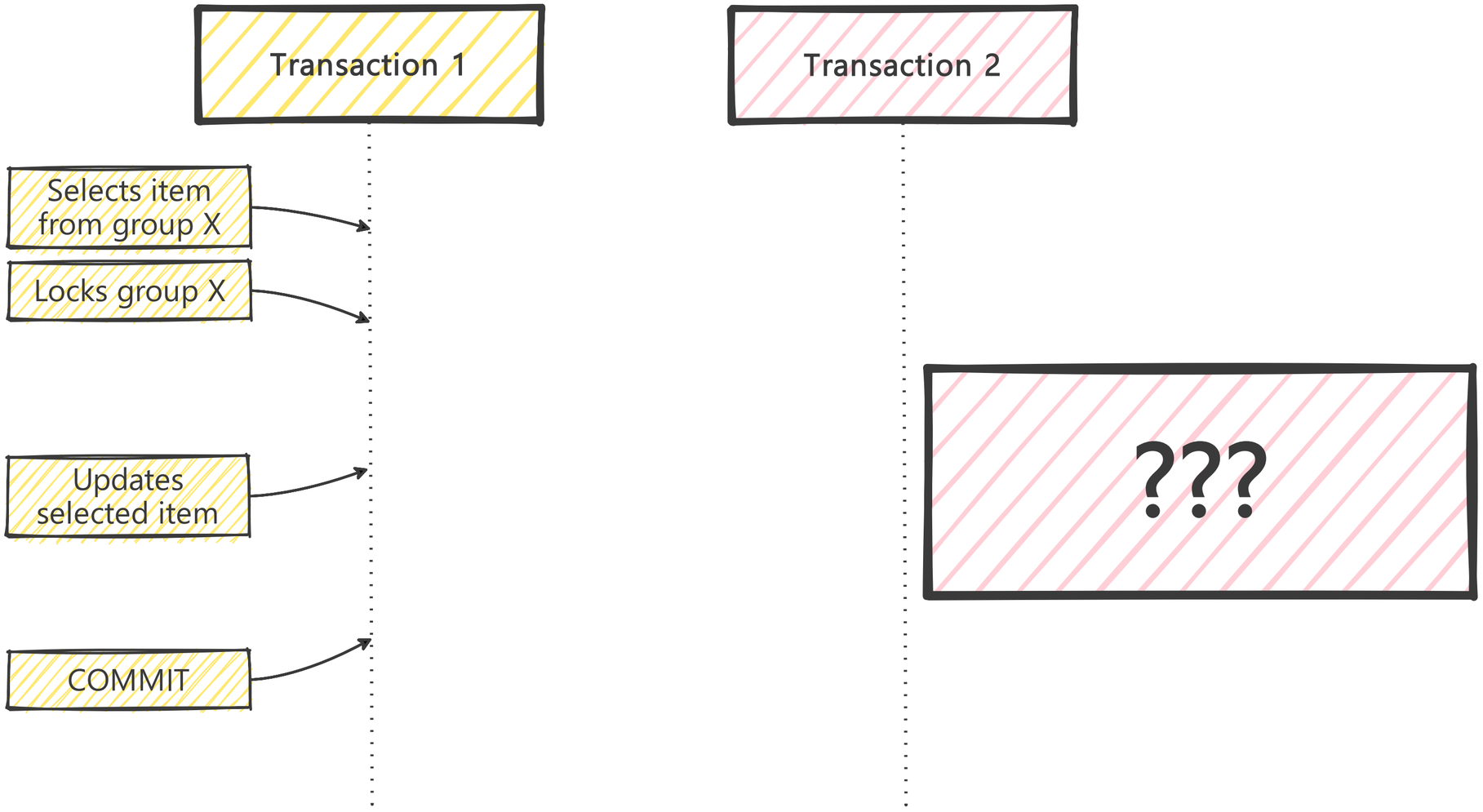
FOR UPDATE SKIP LOCKED

- Есть в **Oracle**, **MySQL 8+**
- Для MSSQL: см. READPAST / UPDLOCK

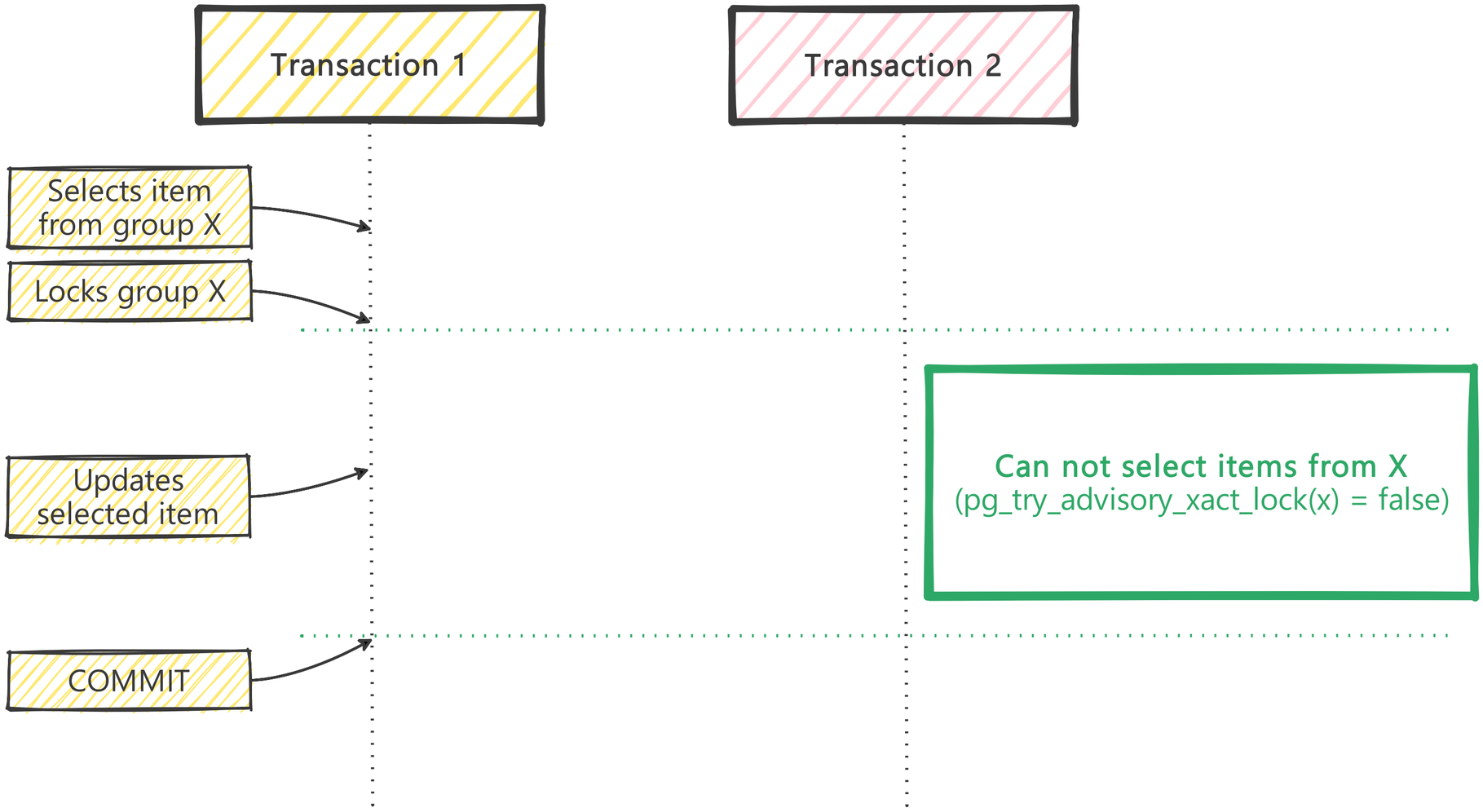
Рекомендательные блокировки

- **MySQL**: GET_LOCK / IS_FREE_LOCK / RELEASE_LOCK
- **Oracle**: DBMS_LOCK.REQUEST()
- **MSSQL**: sp_getapplock / sp_releaseapplock

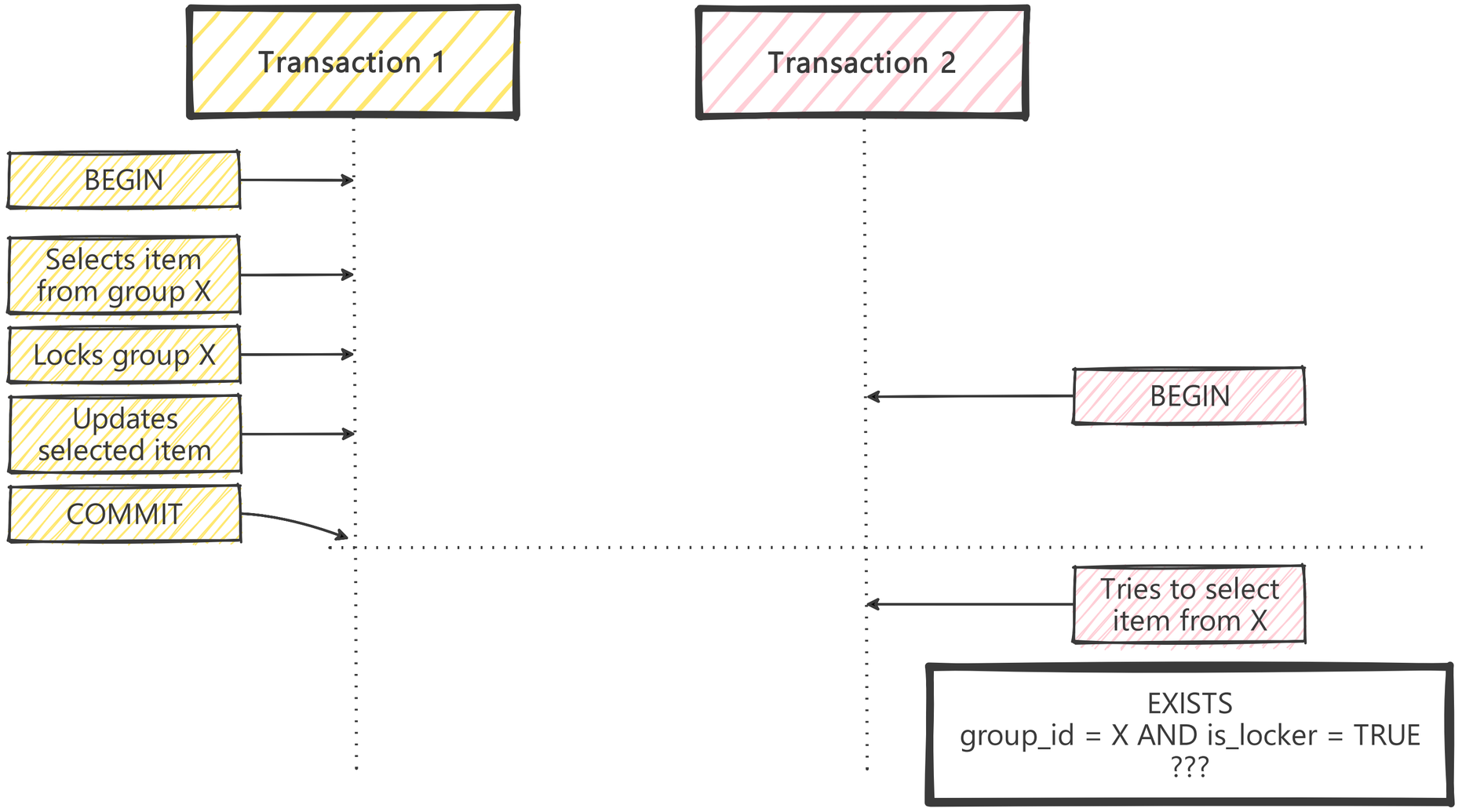
Одновременное выполнение



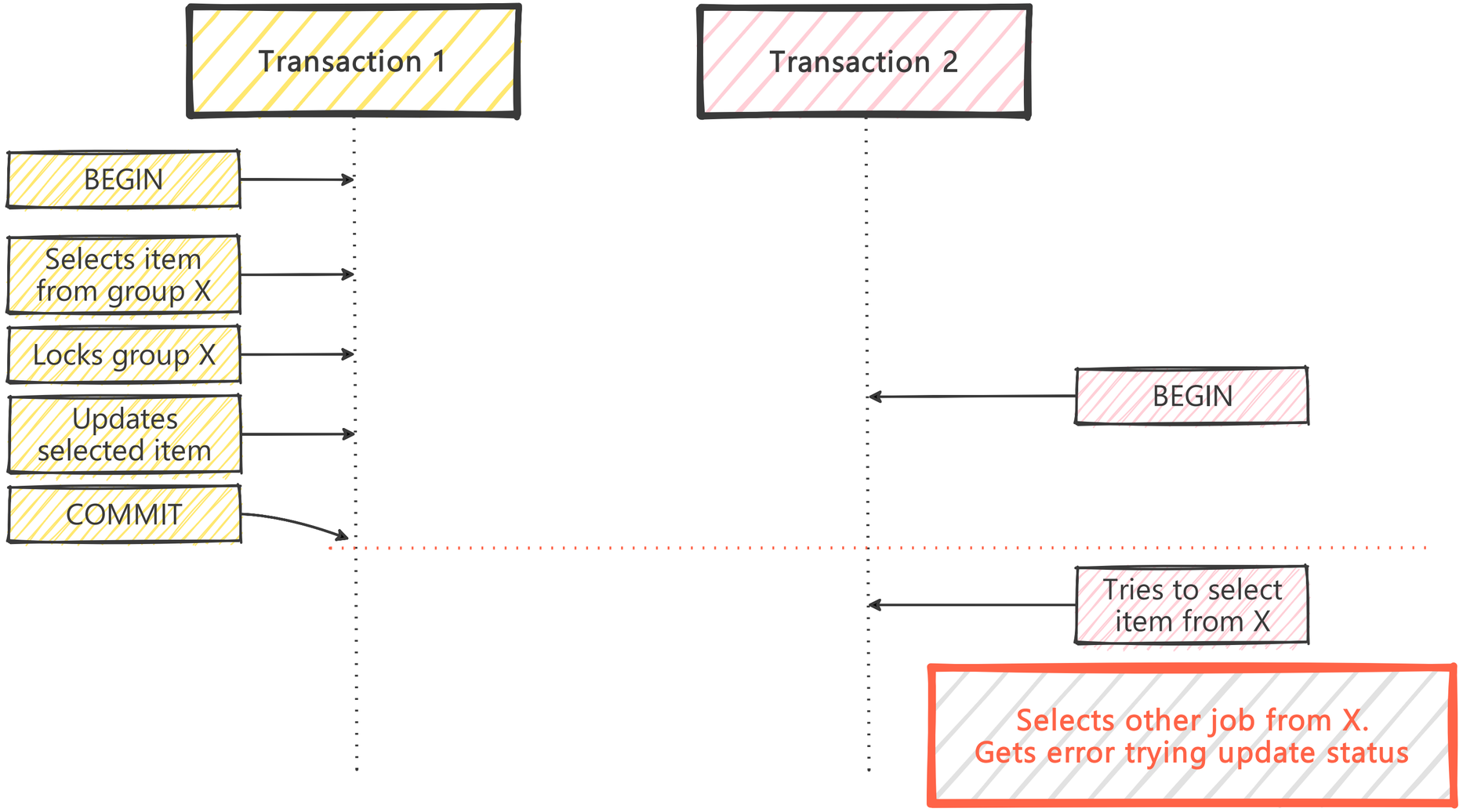
Одновременное выполнение



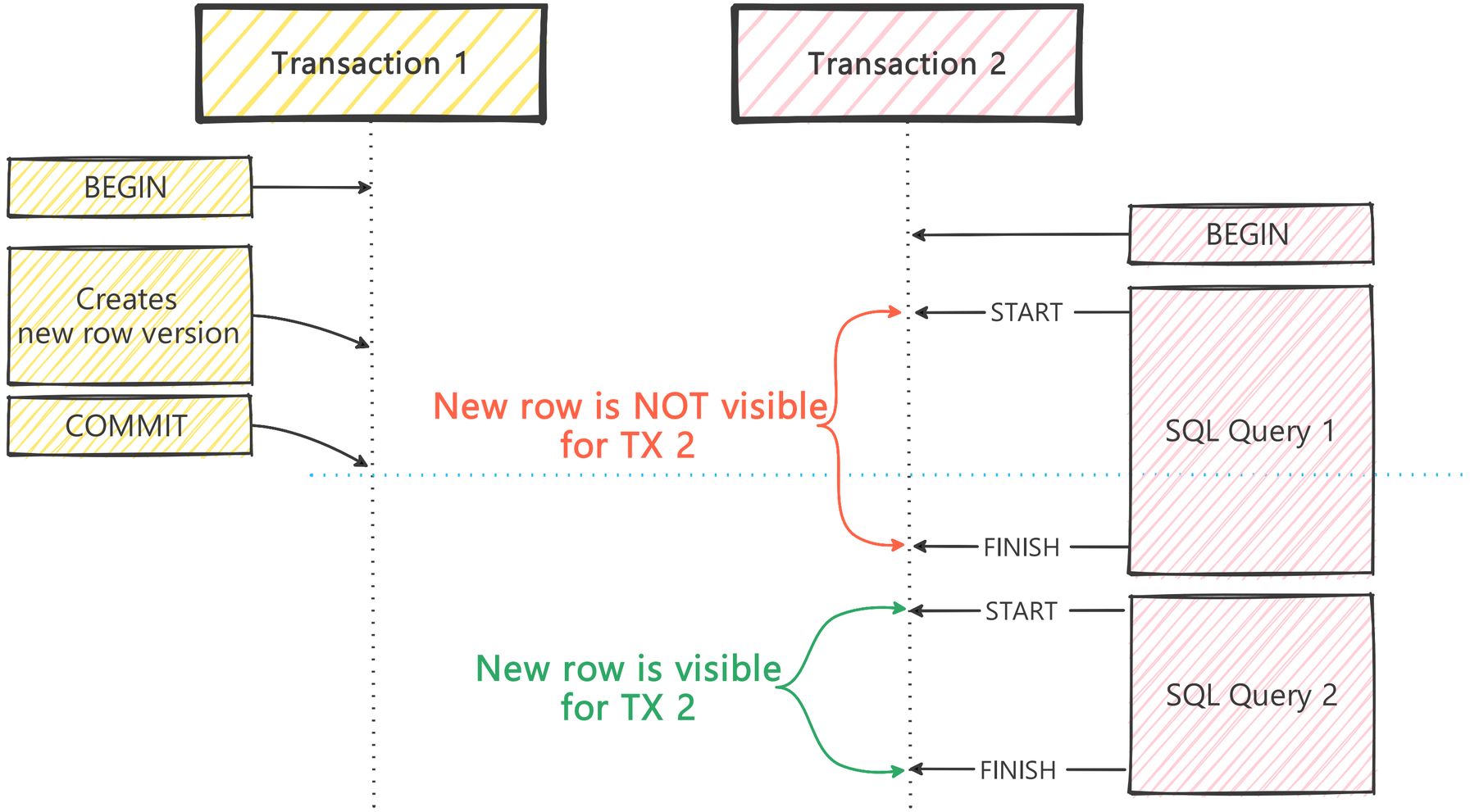
Одновременное выполнение



Одновременное выполнение - ERROR



Видимость новых версий строк



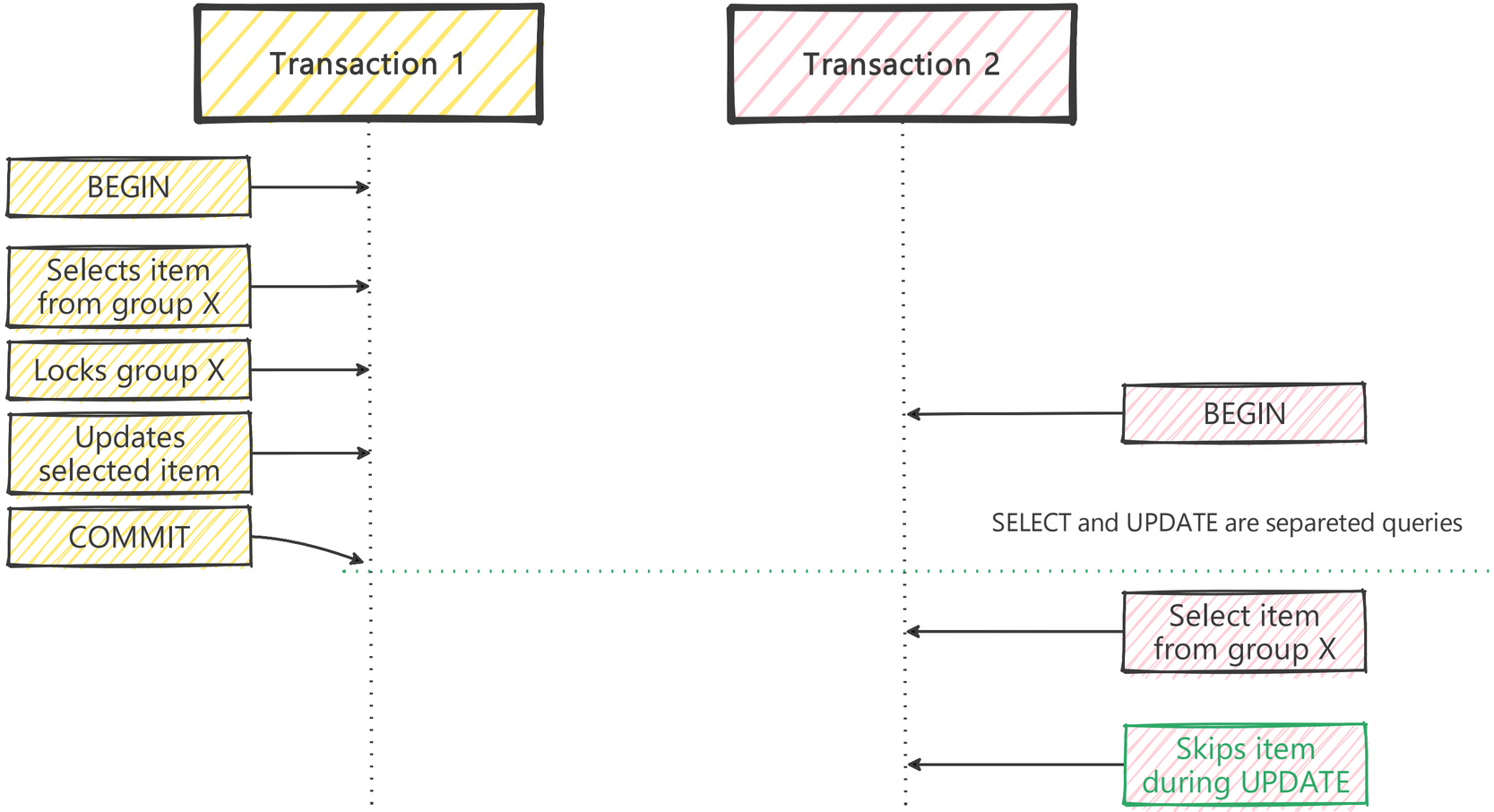
Отдельный запрос для UPDATE с проверкой существования локера

```
BEGIN;

  -- Первым запросом в транзакции только получаем кандидатов для запуска
WITH
  candidates AS (...),
  candidates_with_group_unique AS (...)
SELECT id FROM candidates WHERE group_id IS NULL
UNION
SELECT id FROM candidates_with_group_unique;

...
```

Отдельный запрос для UPDATE с проверкой существования локера

```
...
UPDATE jobby_jobs
SET "status" = Processing, started_count = started_count + 1
WHERE
  id IN (:listOfIdsFromSelect)
  AND (group_id IS NULL OR NOT EXISTS (
      SELECT 1 FROM jobby_jobs jl
      WHERE
          jl.group_id = group_id
          AND jl.is_group_locker = TRUE
          AND jl.id != id
  ))
RETURNING *;

COMMIT;
```

Отдельный запрос для UPDATE с проверкой существования локера



Проблема
очень больших заблокированных групп

```
WITH candidates AS (
  SELECT * FROM jobby_jobs c
  WHERE
      -- Быстрый поиск в индексе
      c.status = Scheduled AND c.scheduled_start_at <= now()

      -- Всё что ниже вызовет дополнительный фильтр при index scan
      -- Может быть неоптимальным если скопилось много задач в заблокированных группах
      AND (
          c.group_id IS NULL OR (
              NOT EXISTS (
                  SELECT 1 FROM jobby_jobs jl
                  WHERE
                      jl.group_id = c.group_id
                      AND jl.is_group_locker = TRUE
                      AND jl.id != c.id
              )
              AND pg_try_advisory_xact_lock(hashtext(c.group_id)) = TRUE
          )
      )
```

Альтернативный способ

Альтернатива

- Оставить исходный запрос
- **Исключить возможность существования более одной задачи в статусе Scheduled на группу**

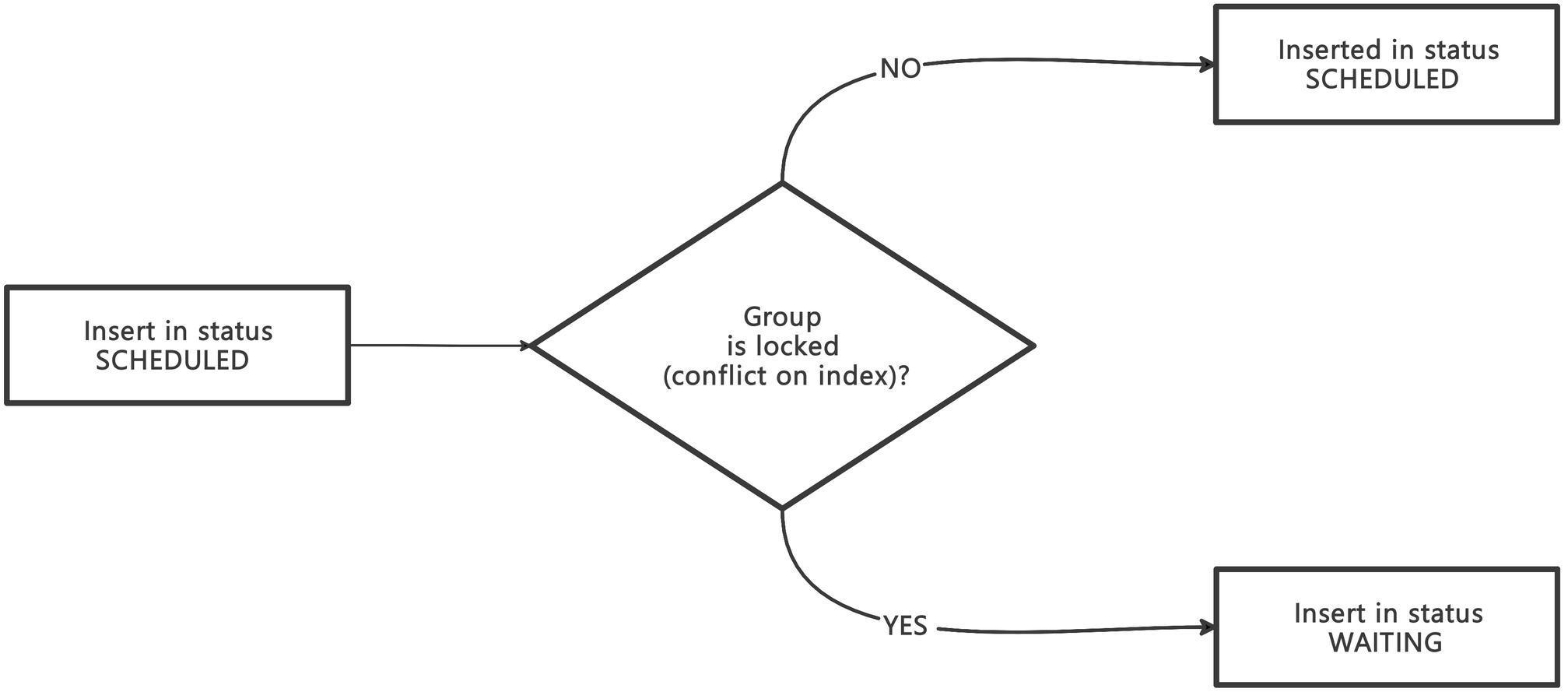
TakeJobs...

```
WITH ready_jobs AS (
    SELECT id FROM jobby_jobs
    WHERE
        "status" = Scheduled
        AND scheduled_start_at <= now()
    ORDER BY scheduled_start_at
    LIMIT :batchSize
    FOR UPDATE SKIP LOCKED
)
UPDATE jobby_jobs
SET
    "status" = <Processing>,
    started_count = started_count + 1
WHERE id IN (SELECT id FROM ready_jobs)
RETURNING *;
```

Доработка ограничения уникальности

```
ALTER TABLE jobby_jobs
    ADD COLUMN IF NOT EXISTS is_group_locker BOOLEAN GENERATED ALWAYS AS (
        -- Если есть группа
        group_id IS NOT NULL
        -- И задача в работе ИЛИ запланирована к запуску
        -- ИЛИ упала
        AND (
            "status" = PROCESSING OR "status" = SCHEDULED
            OR (
                lock_group_if_failed = TRUE
                AND "status" = FAILED
            )
.........

-- Такой индекс не позволит имеет более одной Scheduled-записи на группу
CREATE UNIQUE INDEX ON jobby_jobs(group_id)
WHERE is_group_locker = TRUE;
```

Подмена статуса при вставке



Как сделать INSERT ON CONFLICT DO OTHER INSERT

Создание записи с подменой статуса

```
WITH
try_insert_scheduled AS (
  -- Пытаемся вставить запись в статусе SCHEDULED
  INSERT INTO jobby_jobs (..., "status")
  VALUES (..., SCHEDULED)

  -- Если группа уже заблокирована - запись не добавится и ничего не вернётся
  -- DO UPDATE вместо DO NOTHING – чтобы заблокировать конфликтующую строку
  ON CONFLICT (group_id) WHERE is_group_locker = TRUE
  DO UPDATE SET ... WHERE FALSE

  RETURNING 1
)
...
```

INSERT FROM SELECT

Вариант INSERT, выполняющегося по условию

Insert

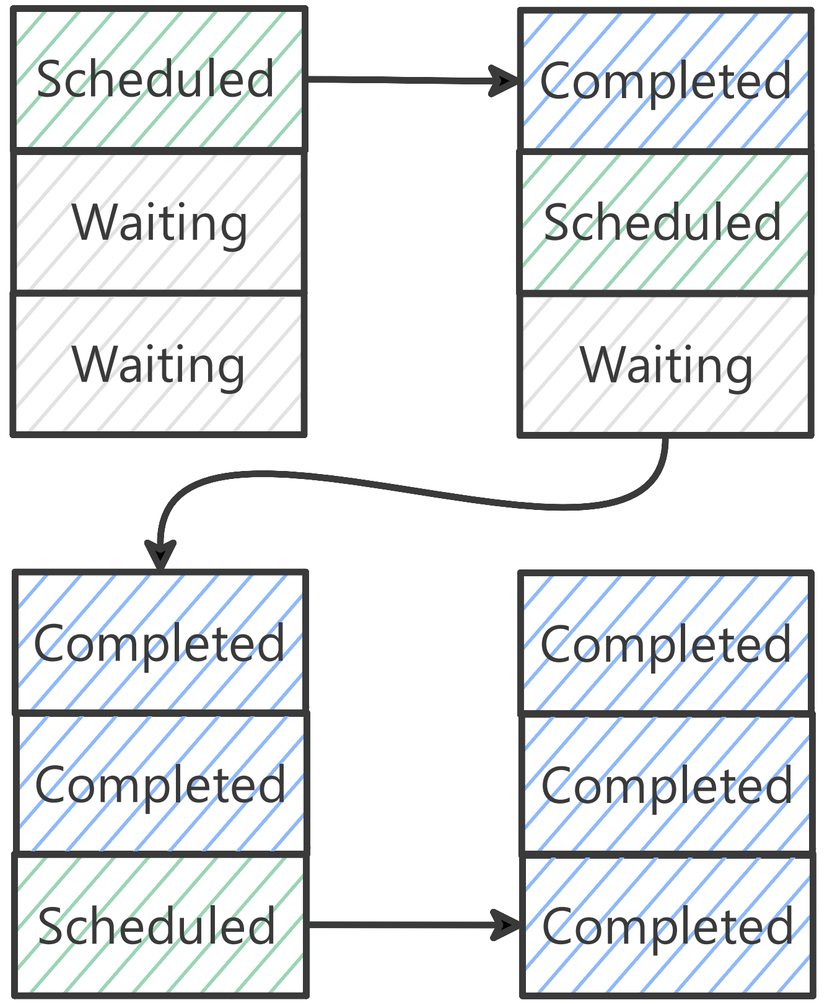
```
INSERT INTO some_table (c_1, c_2, ..., c_N)
-- Вместо VALUES – SELECT с условием
  SELECT value_1, value_2, ... value_N
  WHERE some_predicate;

-- Если указанный предикат false
-- то INSERT не выполнится
```

Создание записи с подменой статуса

```
WITH try_insert_scheduled AS (...)

-- Вставляем запись в статусе WAITING_PREV
INSERT INTO jobby_jobs(..., "status")
  SELECT ..., WAITING_PREV
  -- Но только если не отработала вставка в статусе SCHEDULED
  WHERE NOT EXISTS (
    SELECT 1 FROM try_insert_scheduled
  )


-- Получили фактически INSERT ... ON CONFLICT DO OTHER INSERT
```

Завершение с запуском следующей

```sql
-- В транзакции
-- Удаляем обработанную запись
DELETE FROM jobby_jobs
WHERE id = :completedJobId;

-- И разблокируем следующую
UPDATE jobby_jobs
SET
  "status" = SCHEDULED
WHERE id IN (
  SELECT id FROM jobby_jobs
  WHERE
    group_id = :completedJobGroupId
    AND "status" = WAITING_PREV
  ORDER BY scheduled_start_at
  LIMIT 1
);
```

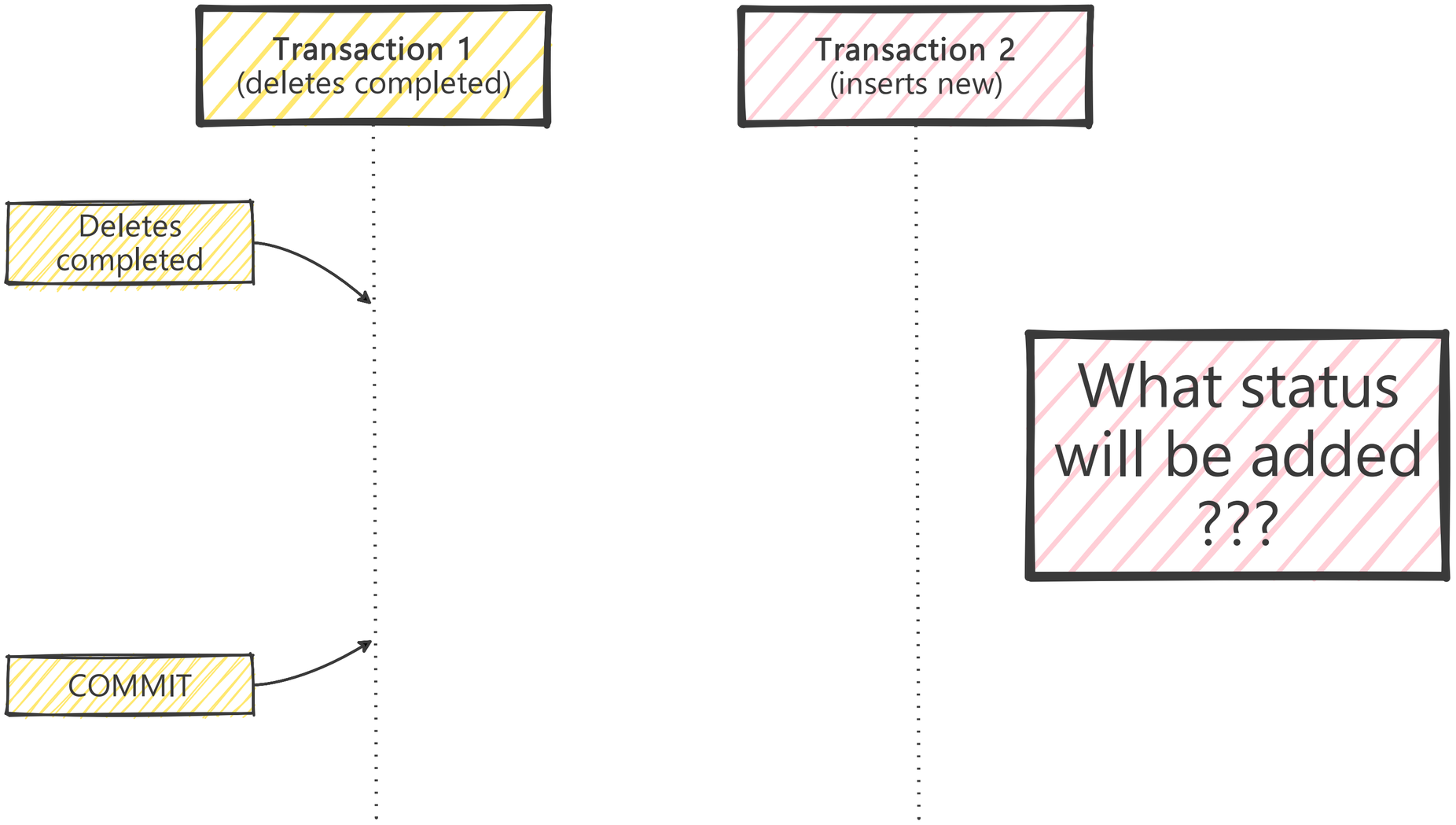


Завершение с запуском следующей задачи группы

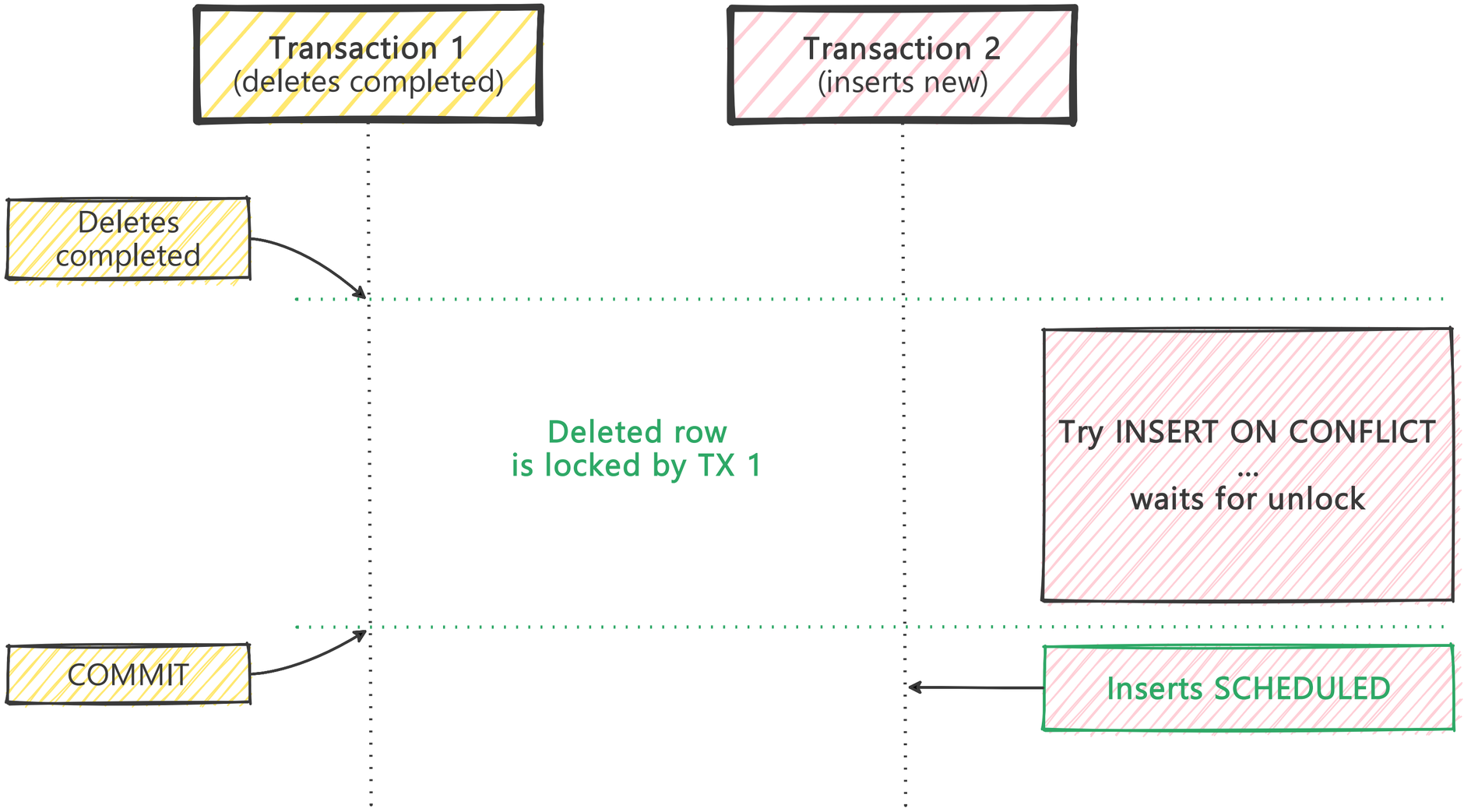
Complete

```
DELETE FROM jobby_jobs
WHERE id = :completedJobId;

UPDATE jobby_jobs
SET
  "status" = SCHEDULED
WHERE id IN (
  SELECT id FROM jobby_jobs
  WHERE
    group_id = :completedJobGroupId
    AND "status" = WAITING_PREV
  ORDER BY scheduled_start_at
  LIMIT 1
);

-- Partial индекс (group_id, scheduled_start_at)
WHERE group_id IS NOT NULL
      AND status = WAITING_PREV
```

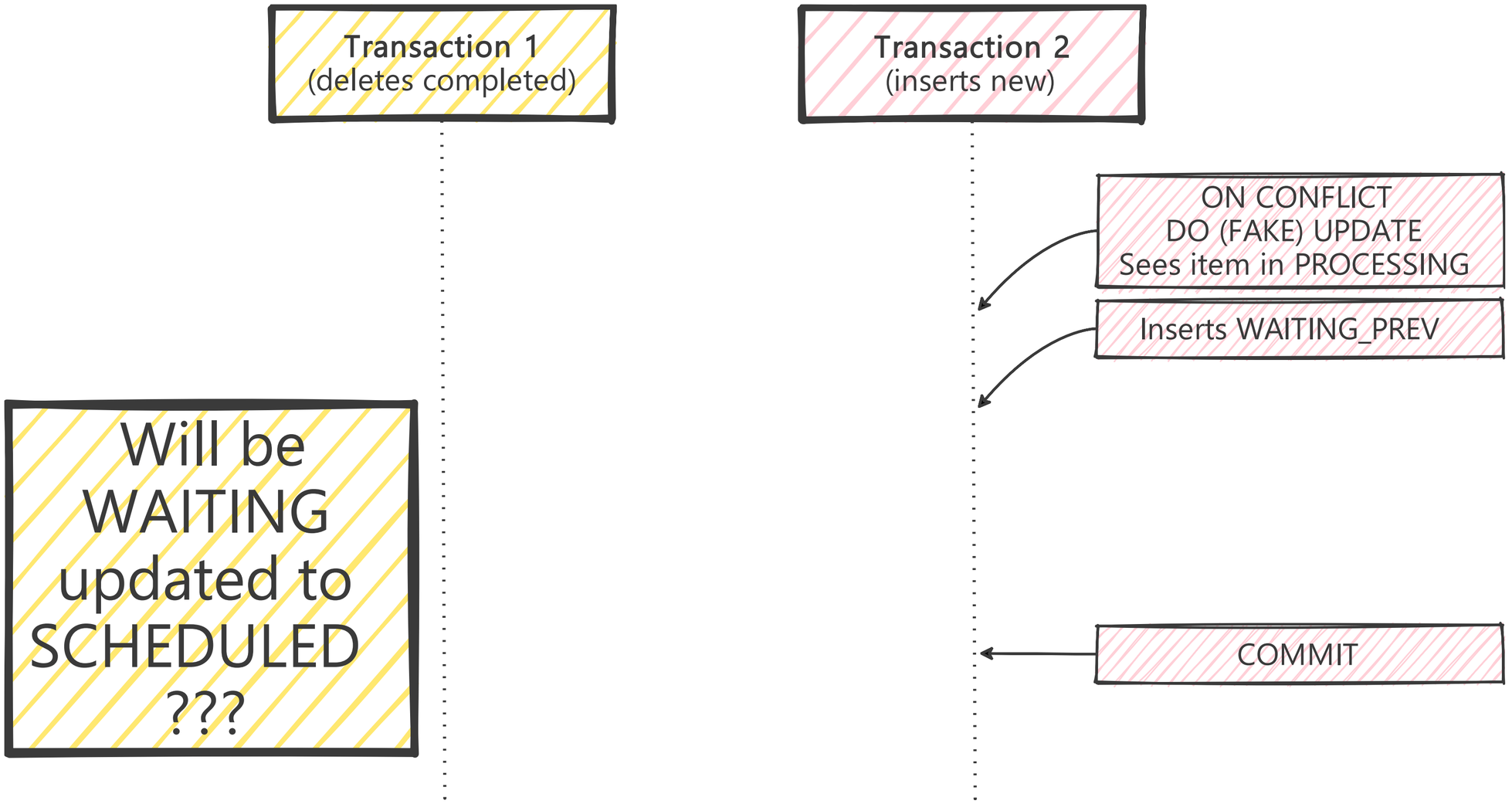
Одновременное создание и завершение



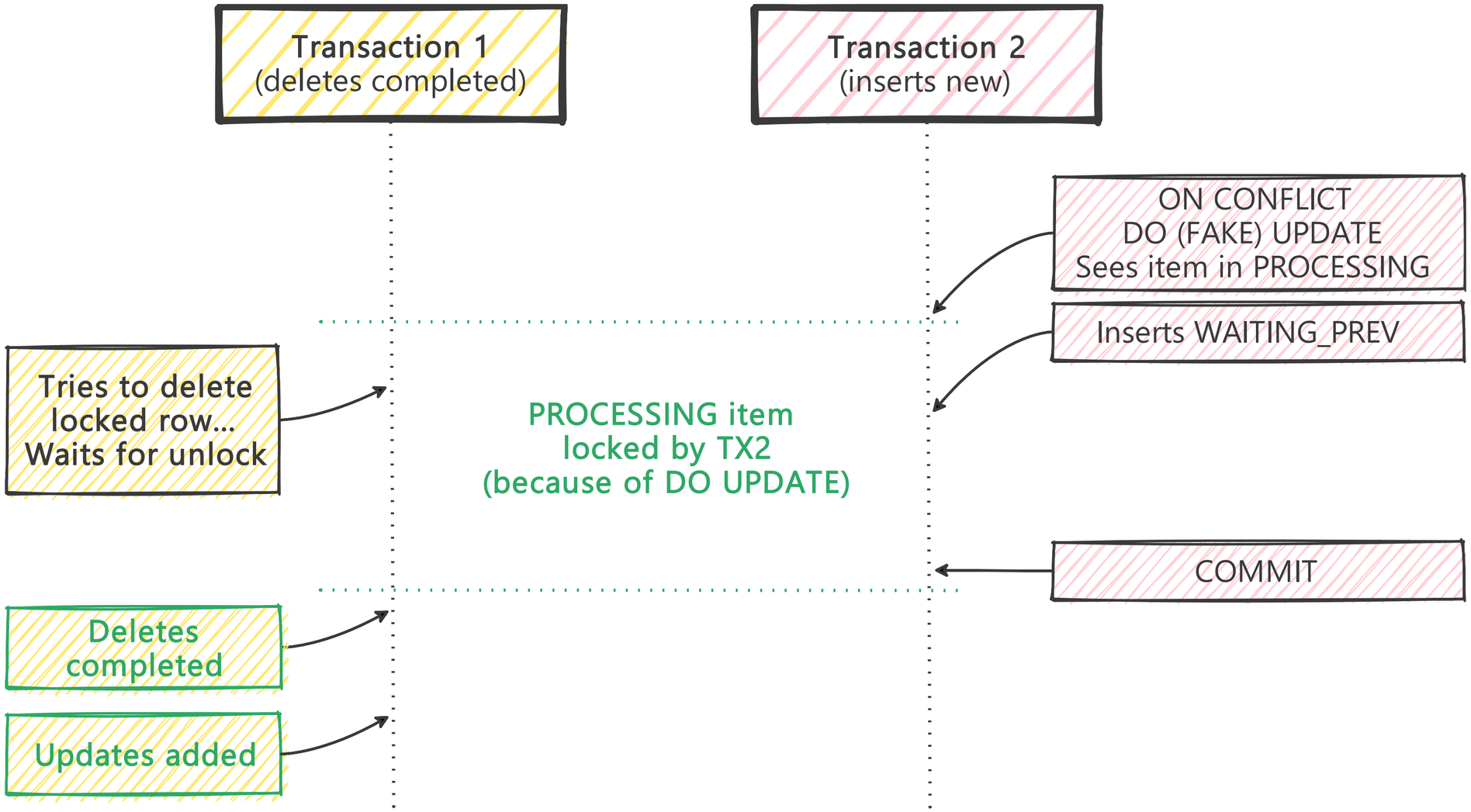
Одновременное создание и завершение



Одновременное создание и завершение



Одновременное создание и завершение



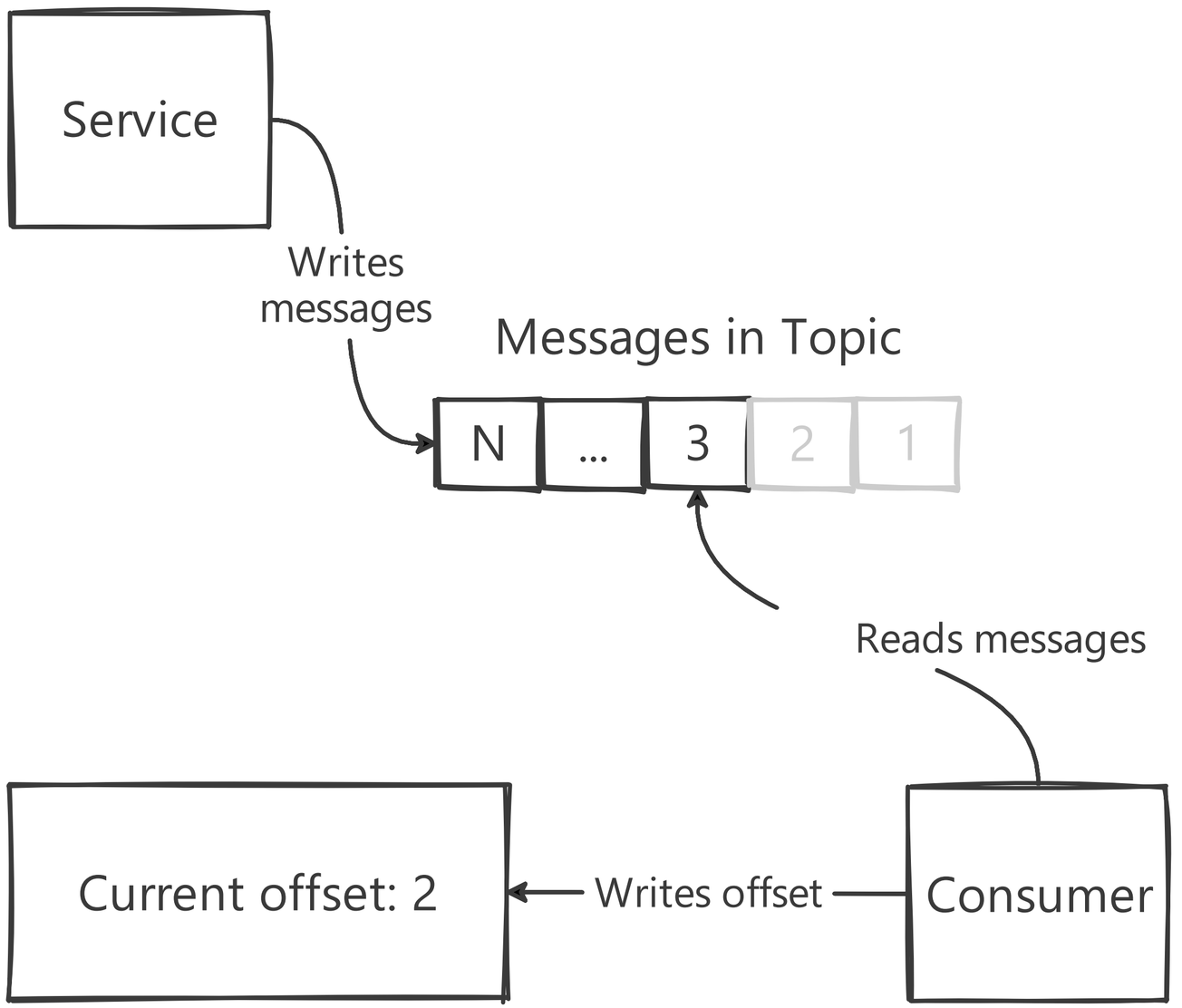
Плюсы и минусы подхода с подменой статуса

- **Стабильно высокая производительность запуска задач**
- Утяжеляется INSERT, а значит и бизнес-транзакция приложения
- +1 UPDATE для завершения задач (но не всегда) и +1 индекс
- Необходимо использовать триггер для создания записей через ORM

Подход на основе смещений

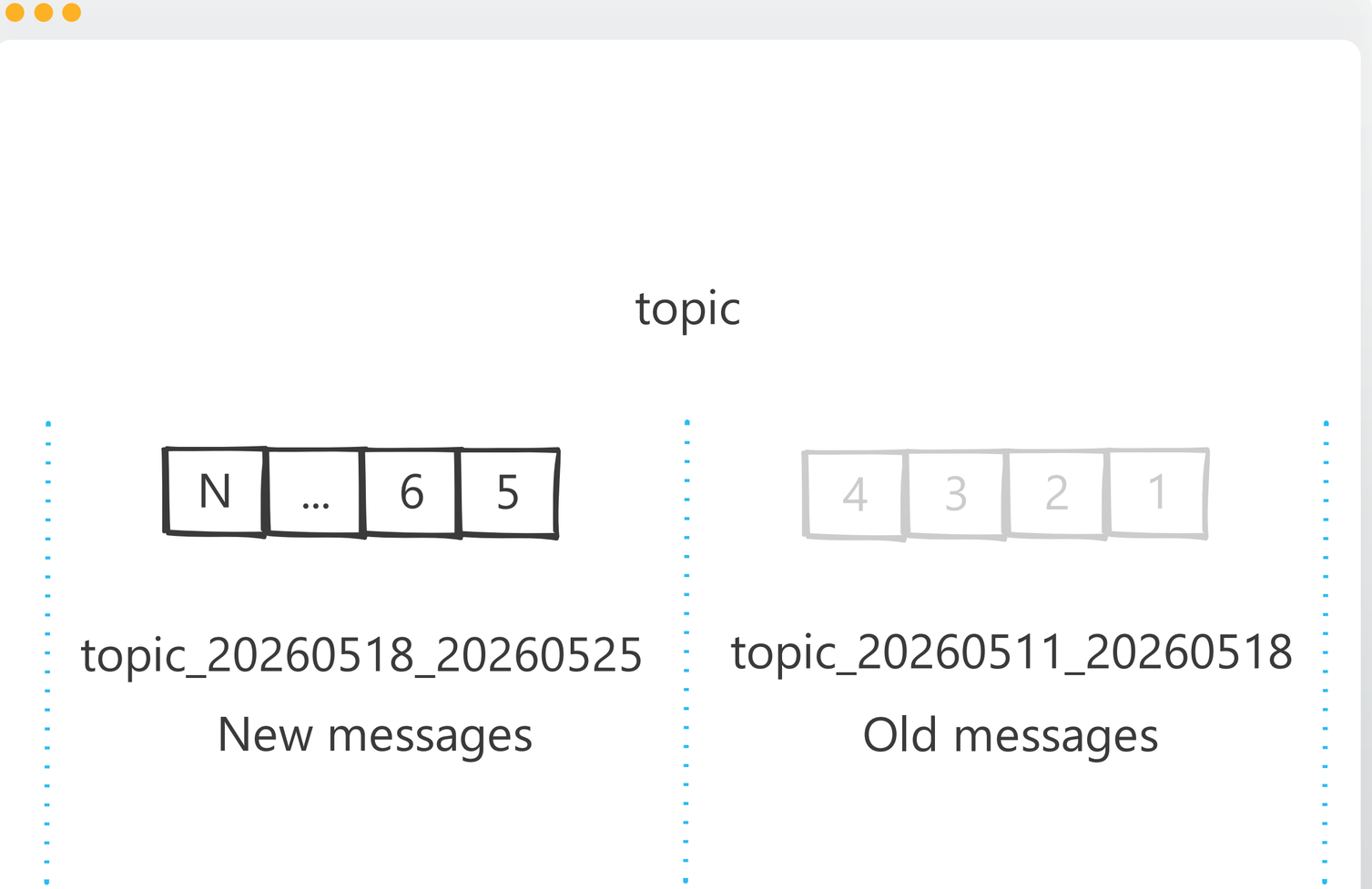
Подход на основе смещений

- Монотонно возрастающий ключ
- Чтение в порядке увеличения значения ключа
- Чтение с позиции последней обработанной записи
- Таблица с сообщениями INSERT-only



Удаление через партиции по дате

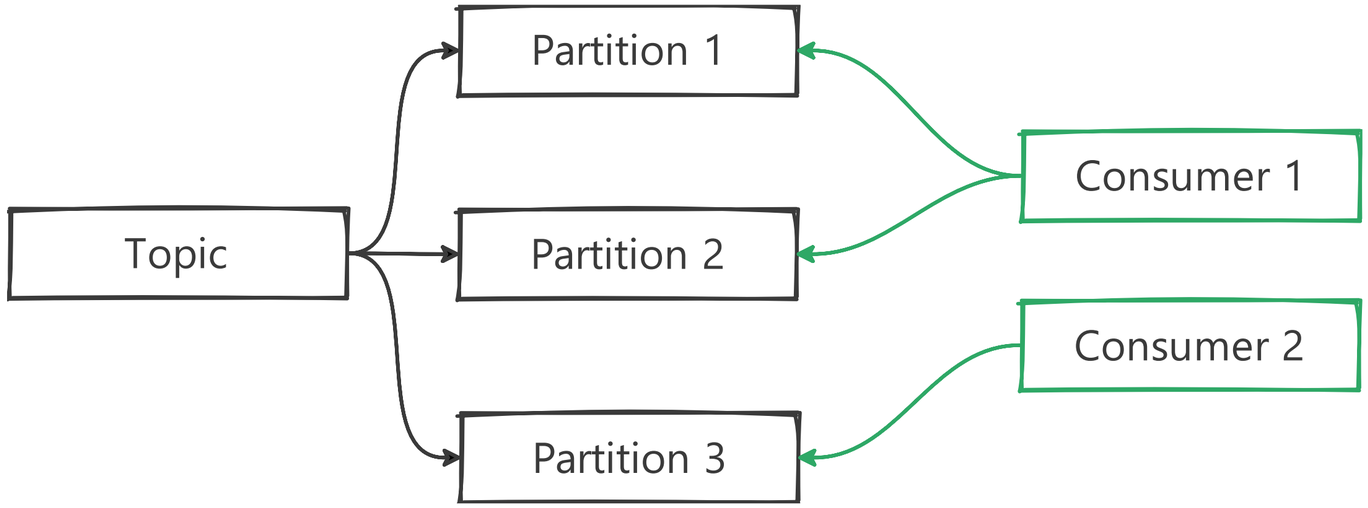
- Таблица топика партицирована по дате создания сообщений
- Старые сообщения удаляются через DROP TABLE для партиции



Как масштабировать и распараллеливать обработку?

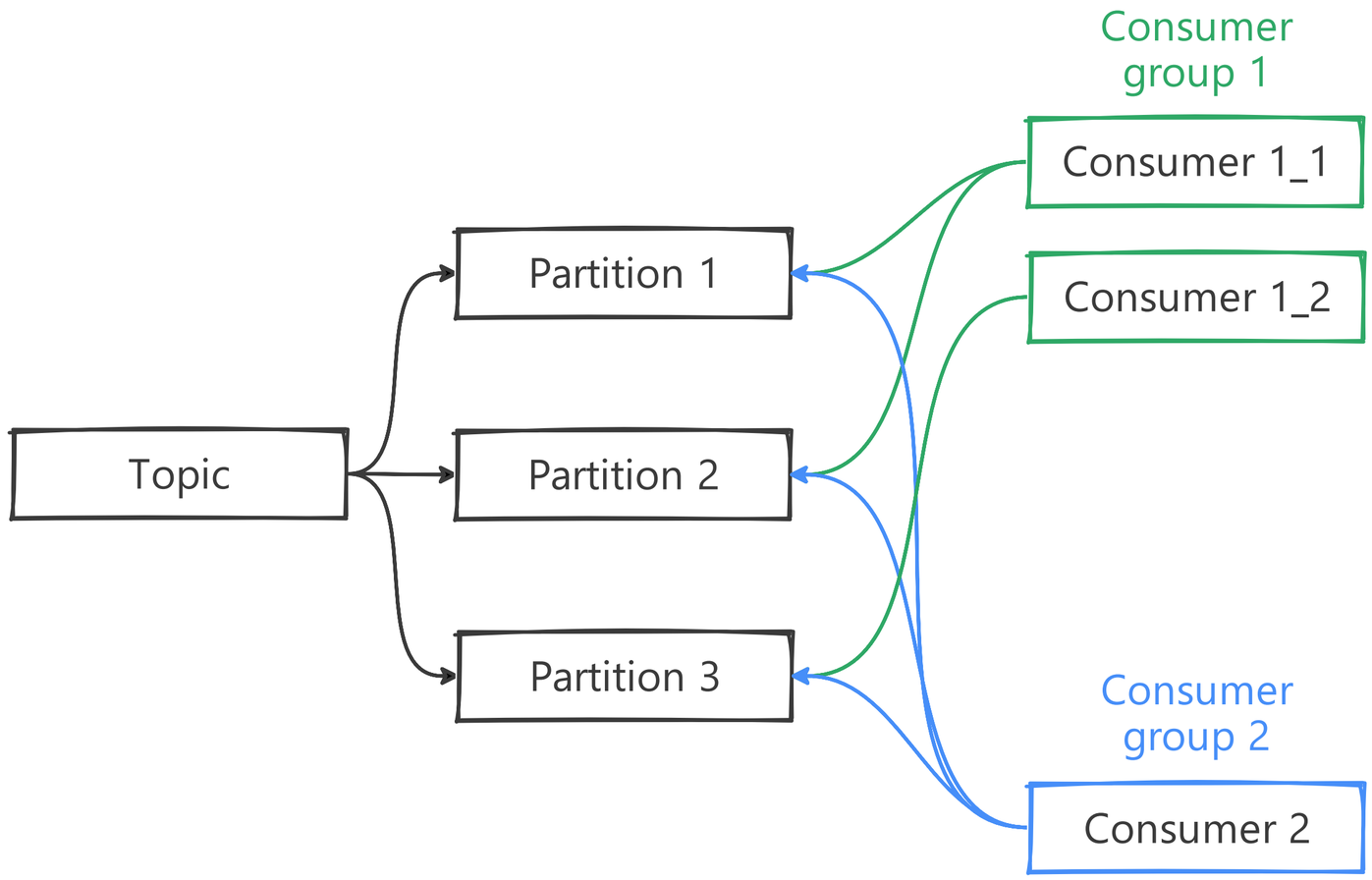
Параллелизм через партиции по ключу

- Поле partition_num в таблице топика
- Вычисляется как хэш от бизнесового group_id
- Связанные сообщения всегда в одной партиции
- Разные партиции обрабатываются параллельно



Параллелизм через партиции по ключу

- Поле partition_num в таблице топика
- Вычисляется как хэш от бизнесового group_id
- Связанные сообщения всегда в одной партиции
- Разные партиции обрабатываются параллельно



Владение консьюмеров партициями

```
CREATE TABLE topic_consumer (
  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

  -- Связь между партициями и группами консьюмеров
  topic_name TEXT NOT NULL,
  partition_num INT NOT NULL,
  consumer_group_id TEXT NOT NULL,

  -- Идентификатор текущего владельца партиции в группе
  owner_server_id TEXT DEFAULT NULL,

  -- Номер последней обработанной данной консьюмер-группой записи
  -- из данной партиции данного топика
  current_offset ??????
)
```

Владение консьюмеров партициями

```
CREATE TABLE topic_consumer (
  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

  -- Связь между партициями и группами консьюмеров
  topic_name TEXT NOT NULL,
  partition_num INT NOT NULL,
  consumer_group_id TEXT NOT NULL,

  -- Идентификатор текущего владельца партиции в группе
  owner_server_id TEXT DEFAULT NULL,

  -- Номер последней обработанной данной консьюмер-группой записи
  -- из данной партиции данного топика
  current_offset ??????
)
```

Владение консьюмеров партициями

```
CREATE TABLE topic_consumer (
  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

  -- Связь между партициями и группами консьюмеров
  topic_name TEXT NOT NULL,
  partition_num INT NOT NULL,
  consumer_group_id TEXT NOT NULL,

  -- Идентификатор текущего владельца партиции в группе
  owner_server_id TEXT DEFAULT NULL,

  -- Номер последней обработанной данной консьюмер-группой записи
  -- из данной партиции данного топика
  current_offset ??????
)
```

Захват партиций консьюмером группы

```sql
-- Захватить N партиций в указанных топиках
UPDATE topic_consumer
SET owner_server_id = :myServerId
WHERE id IN (
  SELECT id FROM topic_consumer
  WHERE
    topic_name = :topicName
    AND consumer_group_id = :myConsumerGroup
    AND owner_server_id IS NULL
  LIMIT N
  FOR UPDATE SKIP LOCKED
)
RETURNING *;

-- Освободить партицию – аналогично установить owner_server_id = NULL
```

Перераспределение партиций в группе

```
// Фоновый процесс перераспределения партиций по необходимости
while (serverWorking)
{
    int totalServers = await GetAliveServersCount(consumerGroupId);
    int totalPartitions = await GetPartitionsCount(topicName);
    int fairCount = (int)Math.Ceiling((decimal)totalPartitions / totalServers);

    if (myPartitionsCount < fairCount)
    {
        await TryCapture(fairCount - myPartitionsCount);
    }
    else if (myPartitionsCount > fairCount)
    {
        await ReleasePartitions(myPartitionsCount - fairCount);
    }

    await Task.Delay(pause);
}
```

Перераспределение партиций в группе

```
// Фоновый процесс перераспределения партиций по необходимости
while (serverWorking)
{
    int totalServers = await GetAliveServersCount(consumerGroupId);
    int totalPartitions = await GetPartitionsCount(topicName);
    int fairCount = (int)Math.Ceiling((decimal)totalPartitions / totalServers);

    if (myPartitionsCount < fairCount)
    {
        await TryCapture(fairCount - myPartitionsCount);
    }
    else if (myPartitionsCount > fairCount)
    {
        await ReleasePartitions(myPartitionsCount - fairCount);
    }

    await Task.Delay(pause);
}
```

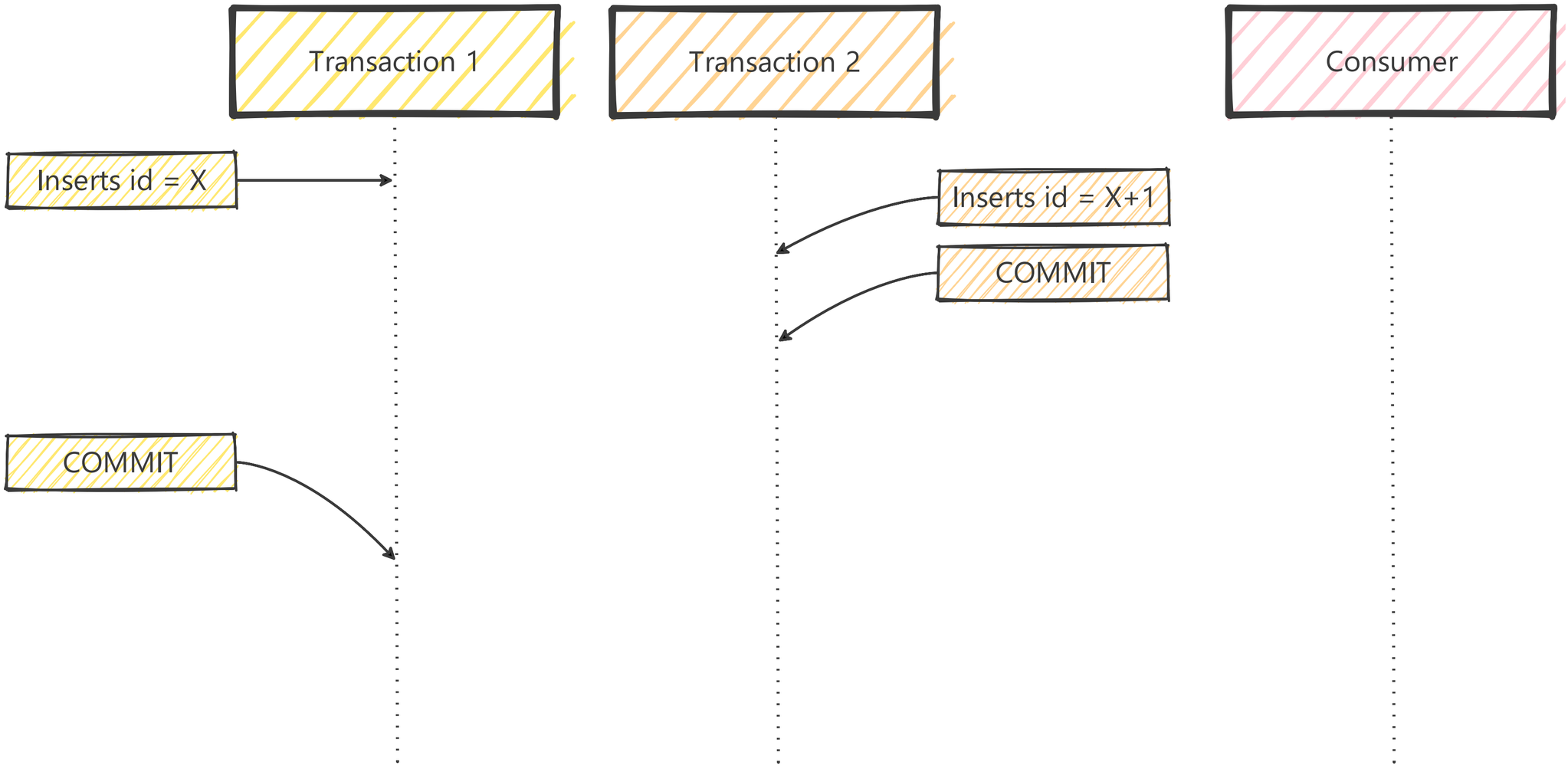
Что использовать в качестве монотонно возрастающего ключа сообщений?

Монотонно возрастающий ключ

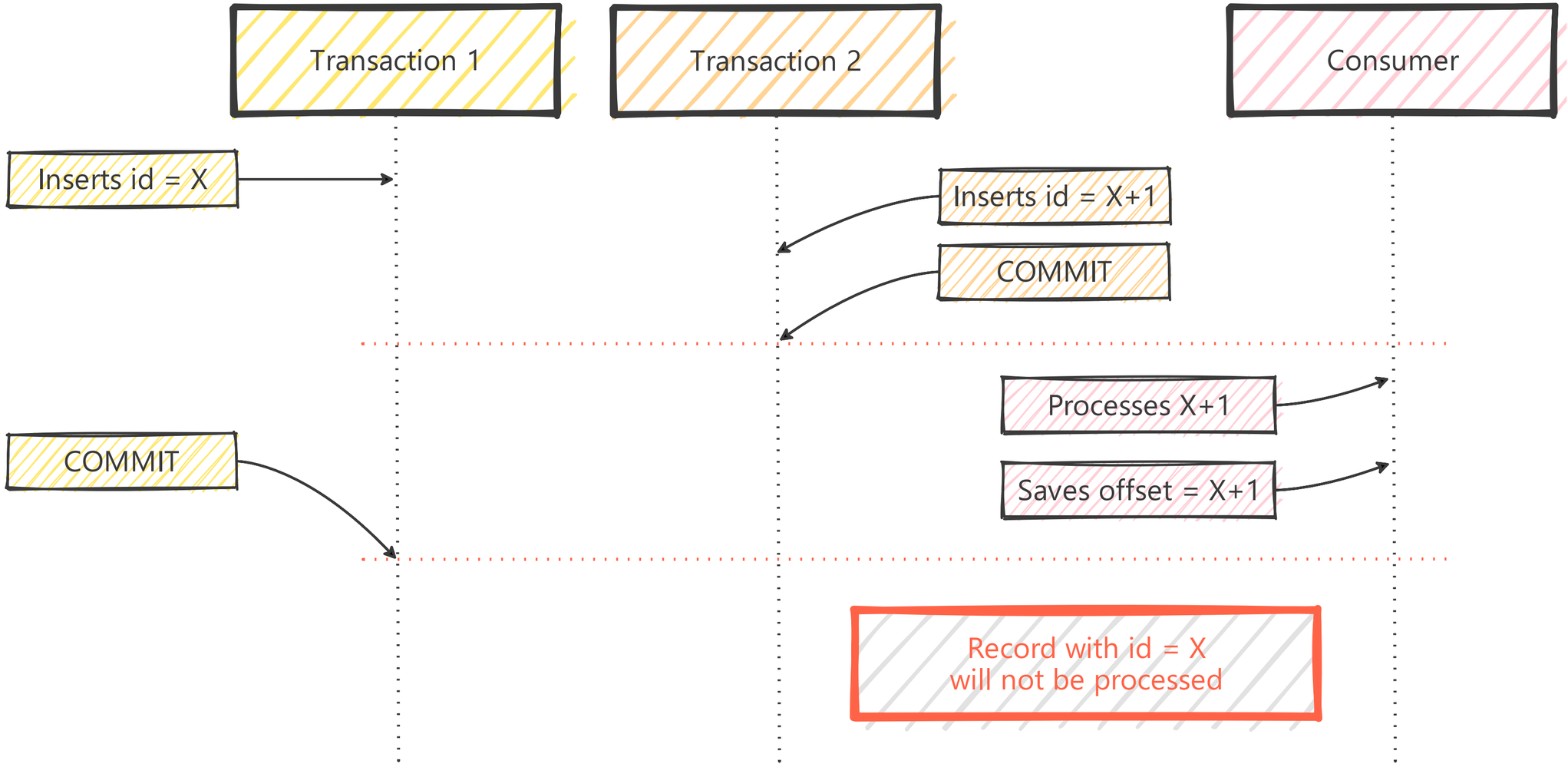
- Timestamp - **ненадёжно**
- Sequence - **???**

Статья от Oscar Dudycz о проблемах
с sequence и способе решения
«How Postgres sequences issues can impact your messaging guarantees»

Параллельная вставка двух записей



Параллельная вставка двух записей



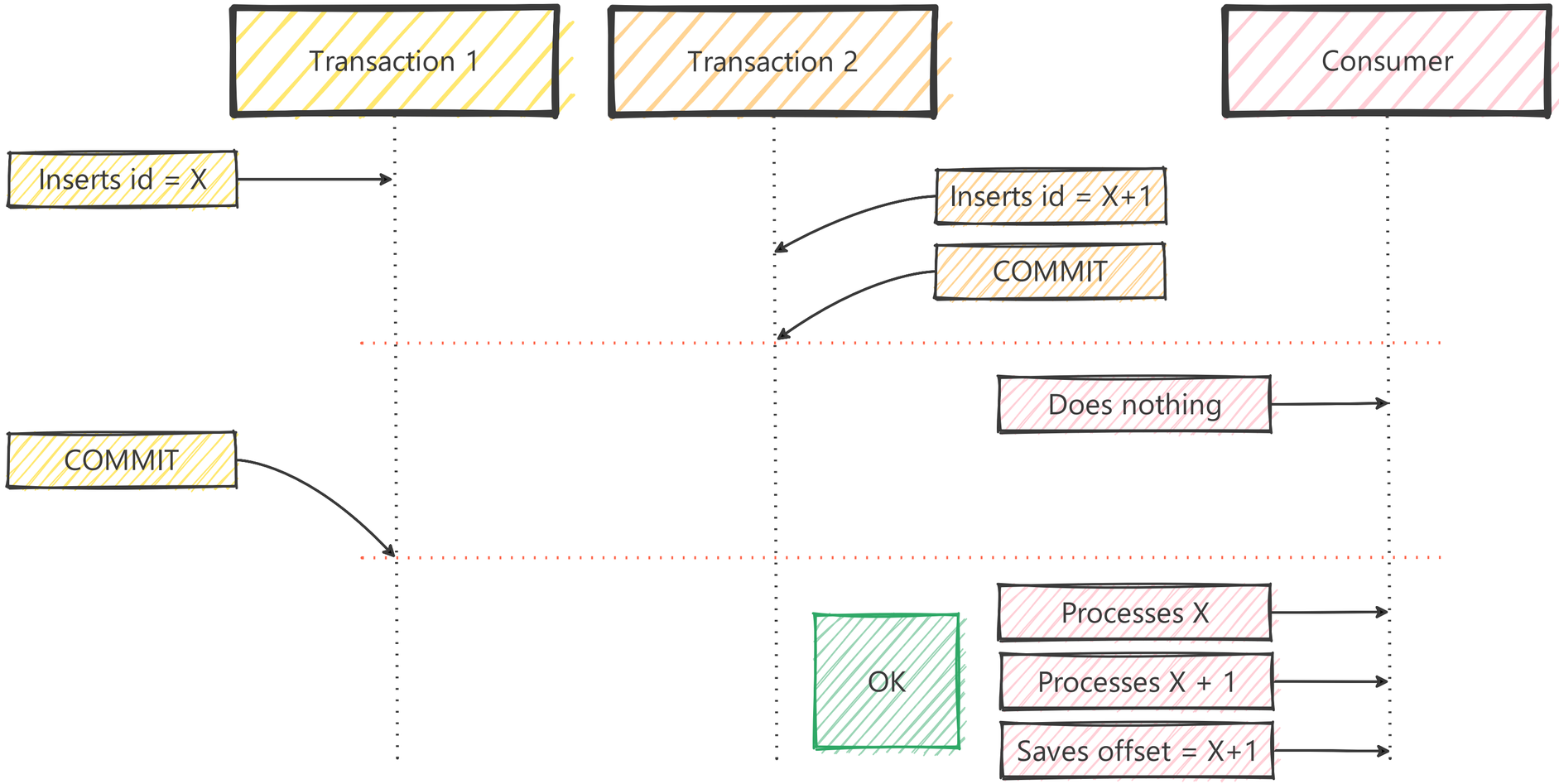
Комбинация sequence и xid8

```
-- pg_current_xact_id() возвращает текущий идентификатор транзакции

-- Добавим к записям outbox 64-битный id транзакции,
-- в которых они были созданы
ALTER TABLE messages
ADD COLUMN
    tx_id xid8 NOT NULL DEFAULT pg_current_xact_id()
```

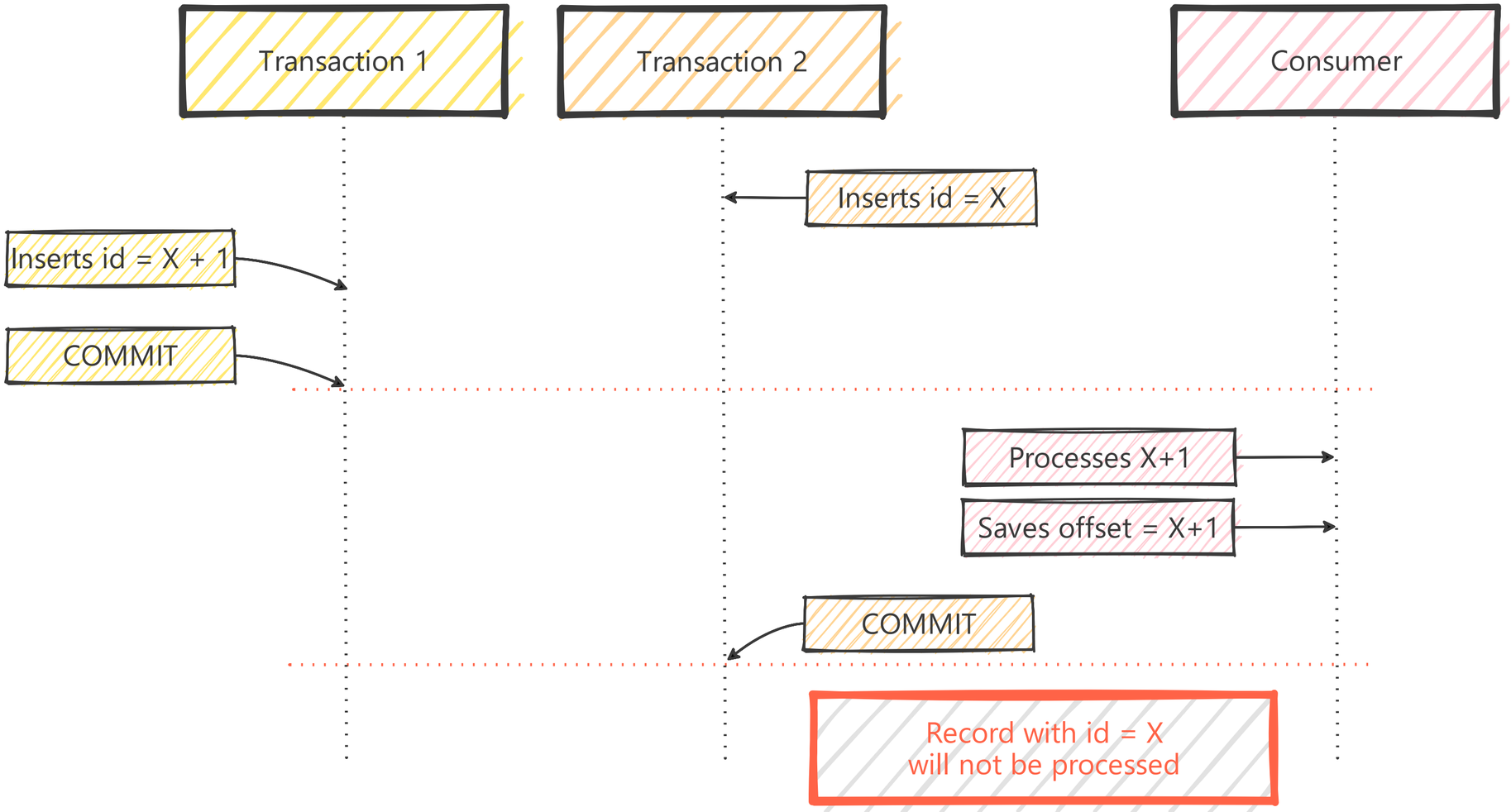
Комбинация sequence и xid8

```
-- Минимальный xid8 ещё не завершённых транзакций:
-- pg_snapshot_xmin(pg_current_snapshot())

SELECT * FROM messages
WHERE
    topic_name = :topicName
    AND partition_num = :partitionNum
    AND id > :lastProcessedId
    AND tx_id < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY id
LIMIT :processingBatchSize
```

Параллельная вставка двух записей



Параллельная вставка двух записей



Чтение по составному смещению

```
-- (tx_id, seq_id) – составной номер записи
-- (lastProcessedTxId, lastProcessedId) – последняя обработанная

SELECT * FROM messages
WHERE
  topic_name = :topicName
  AND partition_num = :partitionNum
  AND (tx_id, seq_id) > (:lastProcessedTxId, :lastProcessedId)
  AND tx_id < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY (tx_id, seq_id) ASC
LIMIT :processingBatchSize;
```

Чтение по составному смещению

```
-- (tx_id, seq_id) – составной номер записи
-- (lastProcessedTxId, lastProcessedId) – последняя обработанная

SELECT * FROM messages
WHERE
  topic_name = :topicName
  AND partition_num = :partitionNum
  -- Берём записи после последней обработанной
  AND (tx_id, seq_id) > (:lastProcessedTxId, :lastProcessedId)
  AND tx_id < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY (tx_id, seq_id) ASC
LIMIT :processingBatchSize;
```

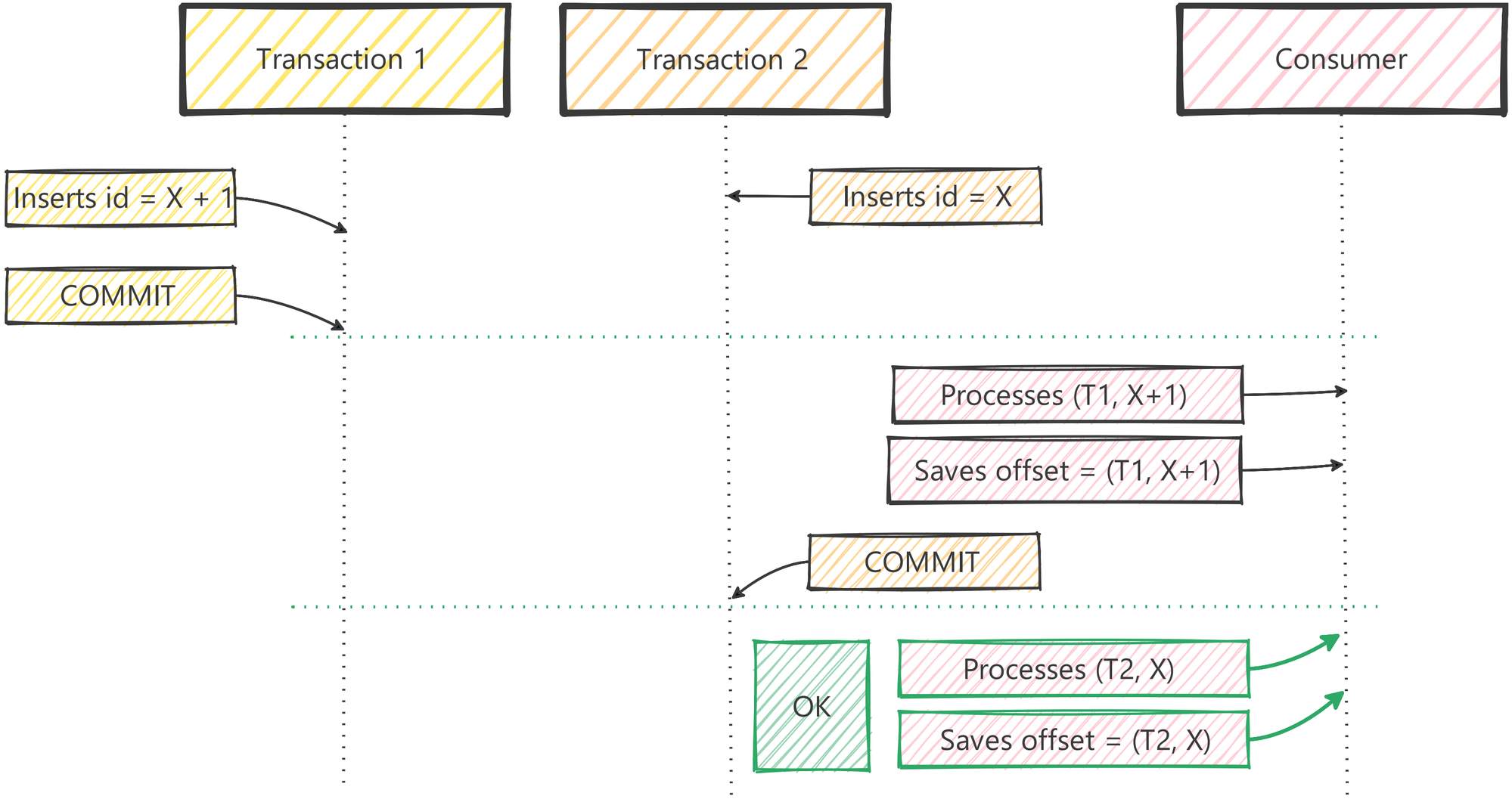
Чтение по составному смещению

```
-- (tx_id, seq_id) – составной номер записи
-- (lastProcessedTxId, lastProcessedId) – последняя обработанная

SELECT * FROM messages
WHERE
  topic_name = :topicName
  AND partition_num = :partitionNum
  AND (tx_id, seq_id) > (:lastProcessedTxId, :lastProcessedId)
  -- Учитываем только транзакции заведомо старше тех что ещё не коммитились
  AND tx_id < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY (tx_id, seq_id) ASC
LIMIT :processingBatchSize;
```

Чтение по составному смещению

```
-- (tx_id, seq_id) – составной номер записи
-- (lastProcessedTxId, lastProcessedId) – последняя обработанная

SELECT * FROM messages
WHERE
  topic_name = :topicName
  AND partition_num = :partitionNum
  AND (tx_id, seq_id) > (:lastProcessedTxId, :lastProcessedId)
  AND tx_id < pg_snapshot_xmin(pg_current_snapshot())
-- Сортировка по составному ключу
ORDER BY (tx_id, seq_id) ASC
LIMIT :processingBatchSize;
```

Чтение по составному смещению



Чтение по составному смещению

```
SELECT * FROM messages
WHERE
  topic_name = :topicName
  AND partition_num = :partitionNum
  AND (tx_id, seq_id) > (:lastProcessedTxId, :lastProcessedId)
  AND tx_id < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY (tx_id, seq_id) ASC LIMIT :processingBatchSize;

-- Составной индекс (topic_name, partition_num, tx_id, seq_id)

-- Если таблица партицирована по topic_name и/или partition_num,
-- то эти поля из индекса можно убрать

-- Может быть полезно в WHERE ограничение на created_at,
-- если есть партиции по этому ключу
```

Возможные проблемы

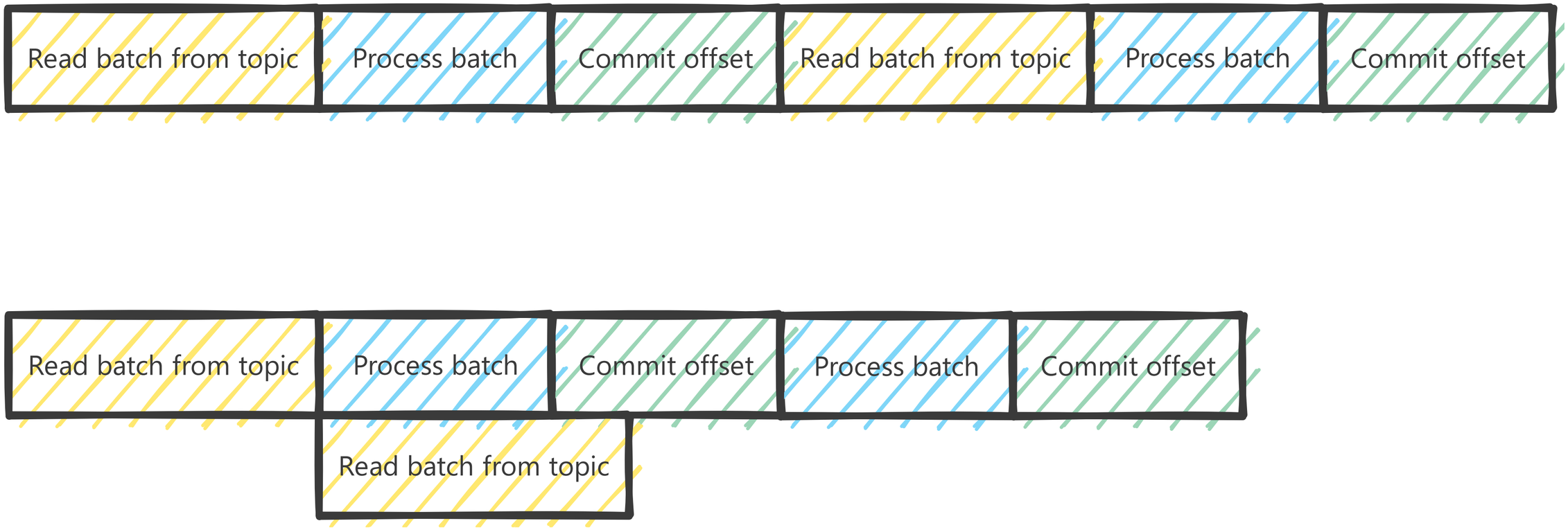
Доклад на JVM Day 2025 об альтернативном подходе от Алексея Кашина
«Надежно отправляем события в Apache Kafka. От CDC до паттерна Transactional Outbox»

- Нет аналогичных механизмов для отличных от PostgreSQL СУБД
- Чужая длинная транзакция **может остановить обработку очереди**
- Решение 1 – не допускать длинных транзакций, аналитические запросы – только на репликах
- Решение 2 – альтернативный подход с исключением недавно добавленных записей по created_at (имеет фиксированный лаг, не даёт 100% гарантии)

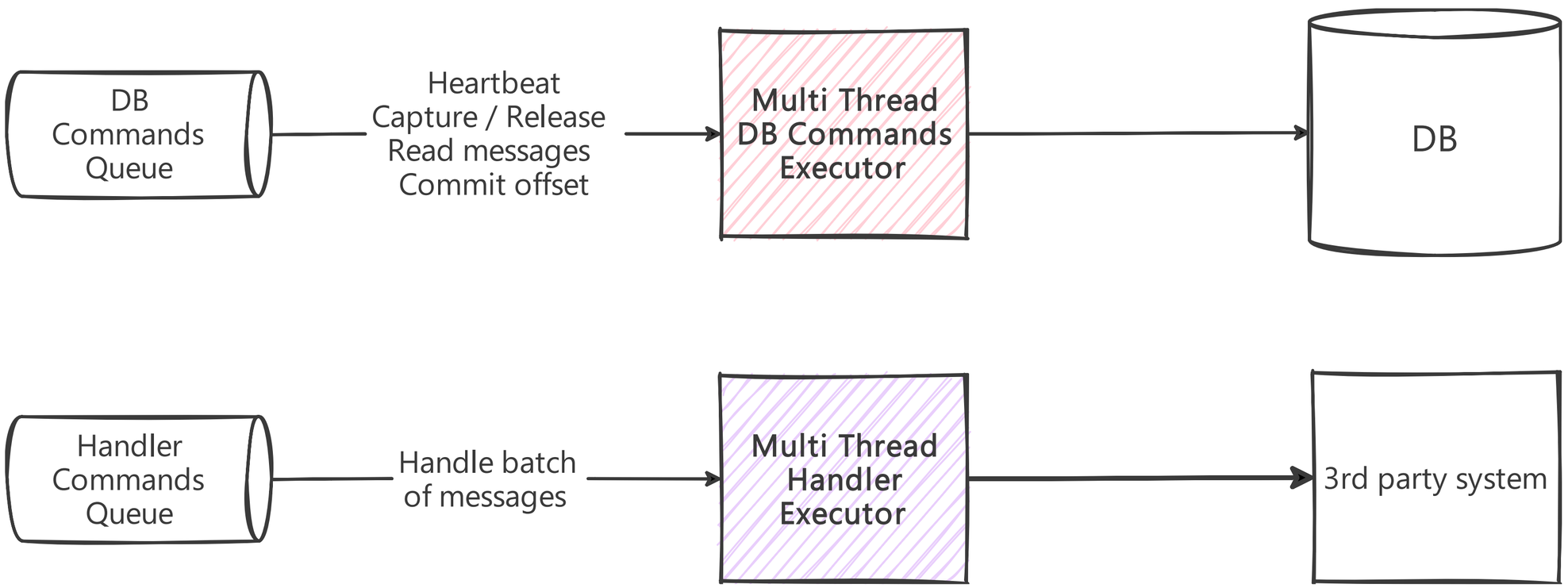
Батчинг и асинхронная загрузка сообщений

Пакетное чтение и обработка

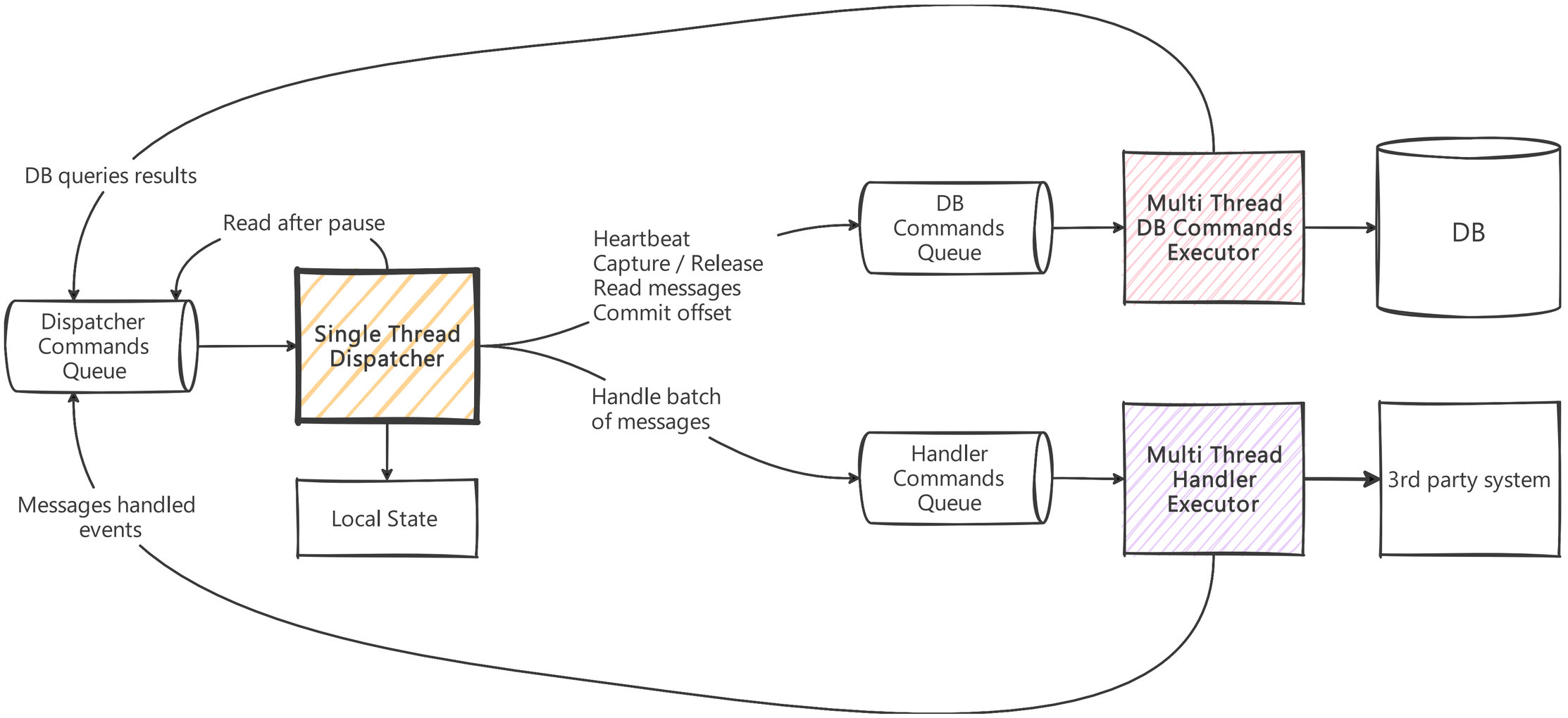
Асинхронная загрузка следующего батча



Независимость работы с БД и обработки



Диспетчер команд и монитор



Channel

- Очередь в оперативной памяти
- Потокобезопасная
- Оптимизирована
  (lock-free, минимизация аллокаций)
- Предоставляет неблокирующий метод ожидания новых записей

https://habr.com/ru/articles/508726/

Обработка команд в компонентах

```
var channel = Channel.CreateUnbounded<Command>();
// channel.Writer делаем доступным другим компонентам

// Запускаем поток-обработчик (или несколько потоков)
_ = Task.Run(async () =>
{
    while (!stoppingToken.IsCancellationRequested)
    {
        var channelIsAvailable = await channel.Reader.WaitToReadAsync(stoppingToken);
        if (!channelIsAvailable)
            break;

        while (!stoppingToken.IsCancellationRequested
                && _channel.Reader.TryRead(out var command))
        {
            await HandleCommand(command);
        }
    }
});
```

Обработка команд в компонентах

```
var channel = Channel.CreateUnbounded<Command>();
// channel.Writer делаем доступным другим компонентам

// Запускаем поток-обработчик (или несколько потоков)
_ = Task.Run(async () =>
{
    while (!stoppingToken.IsCancellationRequested)
    {
        var channelIsAvailable = await channel.Reader.WaitToReadAsync(stoppingToken);
        if (!channelIsAvailable)
            break;

        while (!stoppingToken.IsCancellationRequested
                && _channel.Reader.TryRead(out var command))
        {
            await HandleCommand(command);
        }
    }
});
```

Обработка команд в компонентах

```
var channel = Channel.CreateUnbounded<Command>();
// channel.Writer делаем доступным другим компонентам

// Запускаем поток-обработчик (или несколько потоков)
_ = Task.Run(async () =>
{
    while (!stoppingToken.IsCancellationRequested)
    {
        var channelIsAvailable = await channel.Reader.WaitToReadAsync(stoppingToken);
        if (!channelIsAvailable)
            break;

        while (!stoppingToken.IsCancellationRequested
                && _channel.Reader.TryRead(out var command))
        {
            await HandleCommand(command);
        }
    }
});
```

Обработка команды чтения сообщений

```
ValueTask HandleReadMessagesCommand(ReadMessagesCommand command)
{
    // Если партиция уже не наша - игнорируем комманду
    if (!IsPartitionCaptured(command.Partition))
        return;

    // Если другой поток уже загружает - игнорируем команду
    if (IsLoadingInProgress(command.Partition))
        return;

    // Записываем в LocalState факт того что начали загрузку записей
    SetReadingInProgress(command.Partition))

    // Отправляем команду на чтение из БД
    _dbChannel.Write(new ReadMessagesCommand(...))
```

Обработка команды чтения сообщений

```
ValueTask HandleReadMessagesCommand(ReadMessagesCommand command)
{
    // Если партиция уже не наша - игнорируем комманду
    if (!IsPartitionCaptured(command.Partition))
        return;

    // Если другой поток уже загружает - игнорируем команду
    if (IsLoadingInProgress(command.Partition))
        return;

    // Записываем в LocalState факт того что начали загрузку записей
    SetReadingInProgress(command.Partition))

    // Отправляем команду на чтение из БД
    _dbChannel.Write(new ReadMessagesCommand(...))
```

Обработка загруженных сообщений

```
ValueTask HandleLoadedMessagesCommand(LoadedMessagesCommand command)
{
    if (command.Messages.Count == 0)
    {
        // Партиция пуста – повторяем чтение после паузы
        Task.Run(() => {
            await Task.Delay(pause);
            _channel.Writer(new ReadMessagesCommand(command.Partition, command.Offset));
        });
    }
    else
    {
        …
    }
}
```

Обработка загруженных сообщений

```
ValueTask HandleLoadedMessagesCommand(LoadedMessagesCommand command)
{
    …
    else
    {
        AddToLocalCache(command.Partition, command.Messages);

        // Если обработчик не обрабатывает сейчас прошлые сообщения
        if (!IsHandlerInProgress(command.Partition))
        {
            // Отправляем в обработку из кэша
            await _handlerChannel.Write(new HandleMessagesCommand(…));
            SetHandlerInProgress(command.Partition))

            // Читаем в кэш следующие при необходимости
            await _channel.Write(new ReadMessagesCommand(...));
        }
    }
}
```

Topiqueue

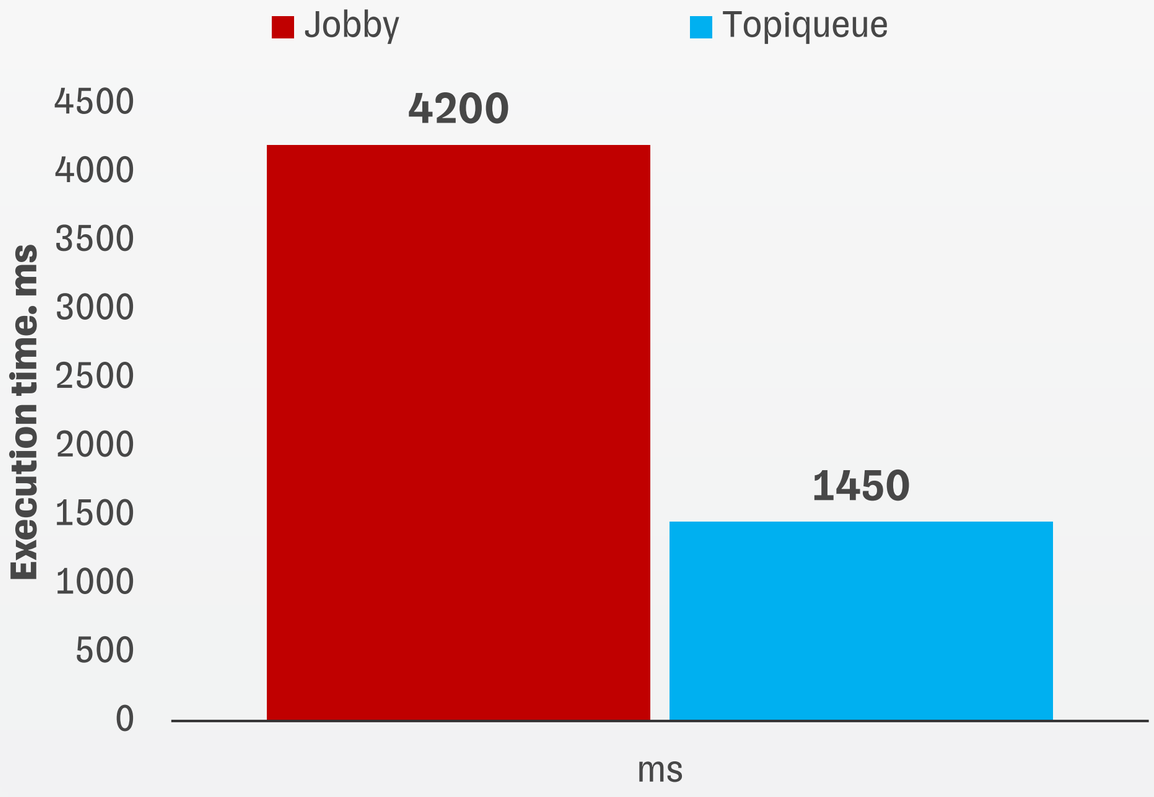
Прототип библиотеки реализующей
описанный Kafka inspired подход
с Event Loop моделью

GitHub https://github.com/fornit1917/topiqueue

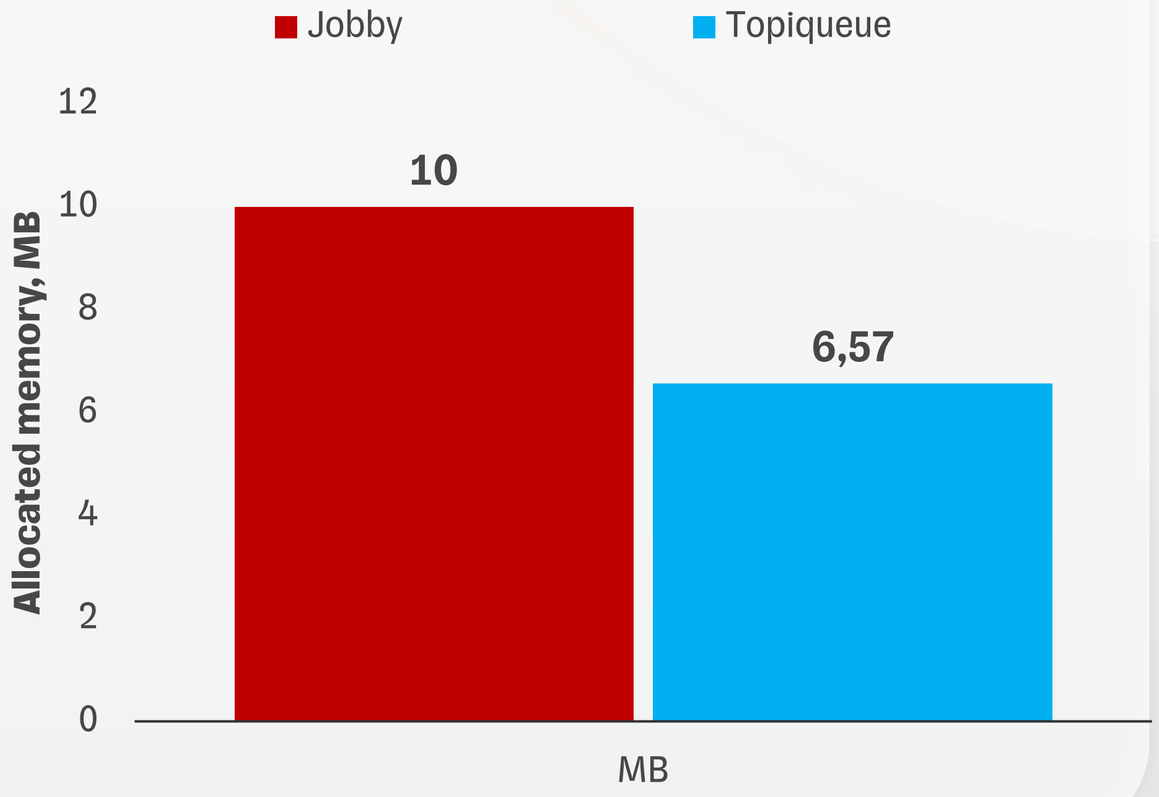
Бенчмарки

Вставка и обработка 1000 сообщений 1 поток / партиция, HandlerBatch = 1

Затраченное время

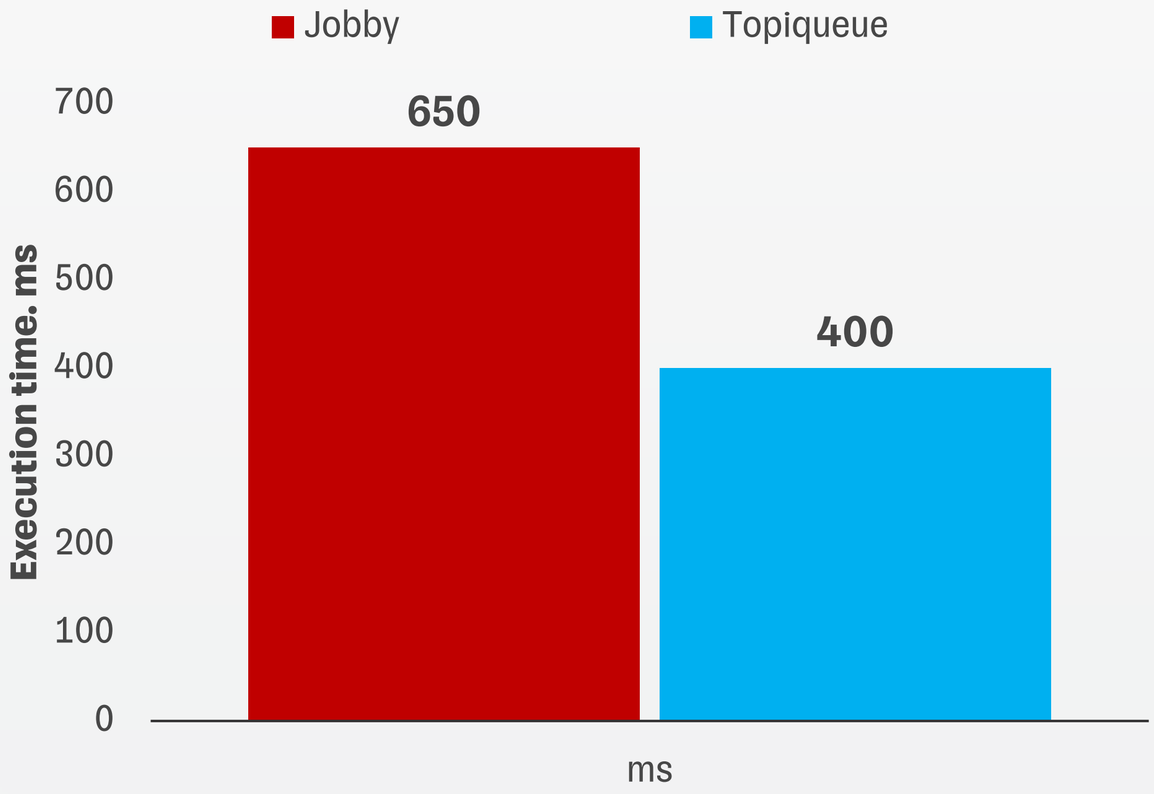


Расход памяти

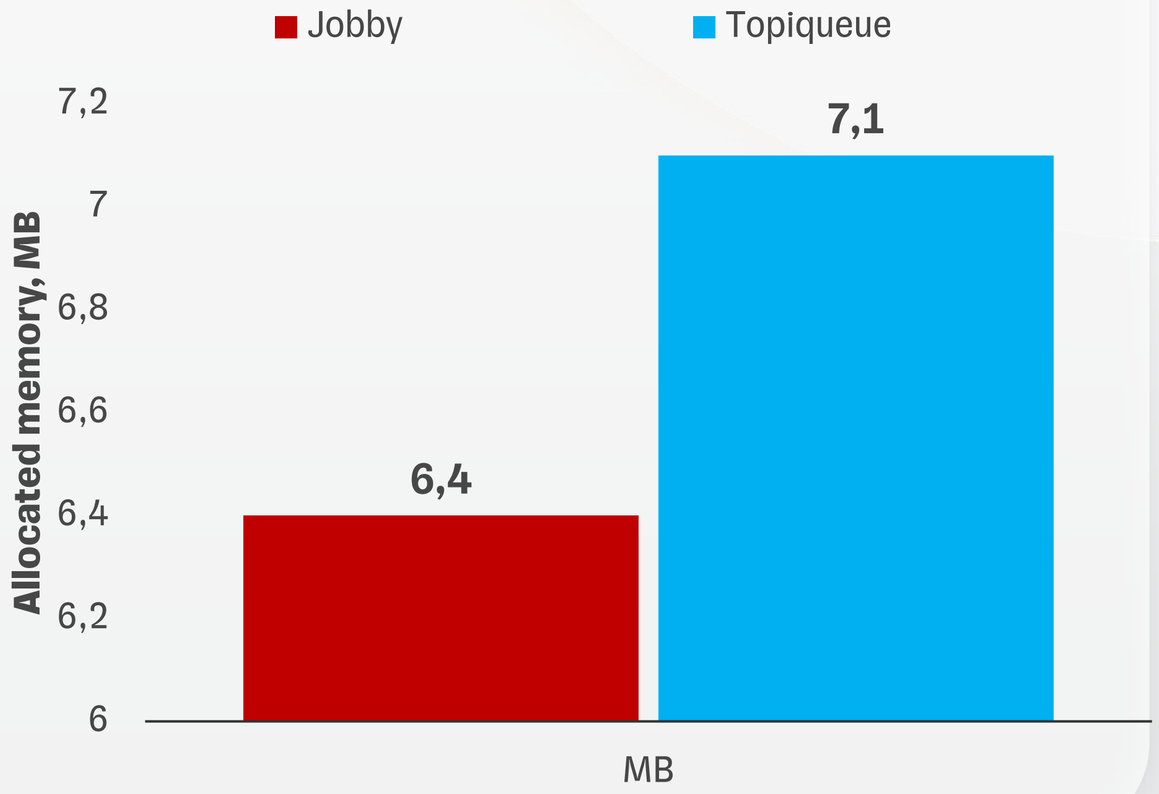


Вставка и обработка 1000 сообщений 10 потоков / партиций, HandlerBatch = 1

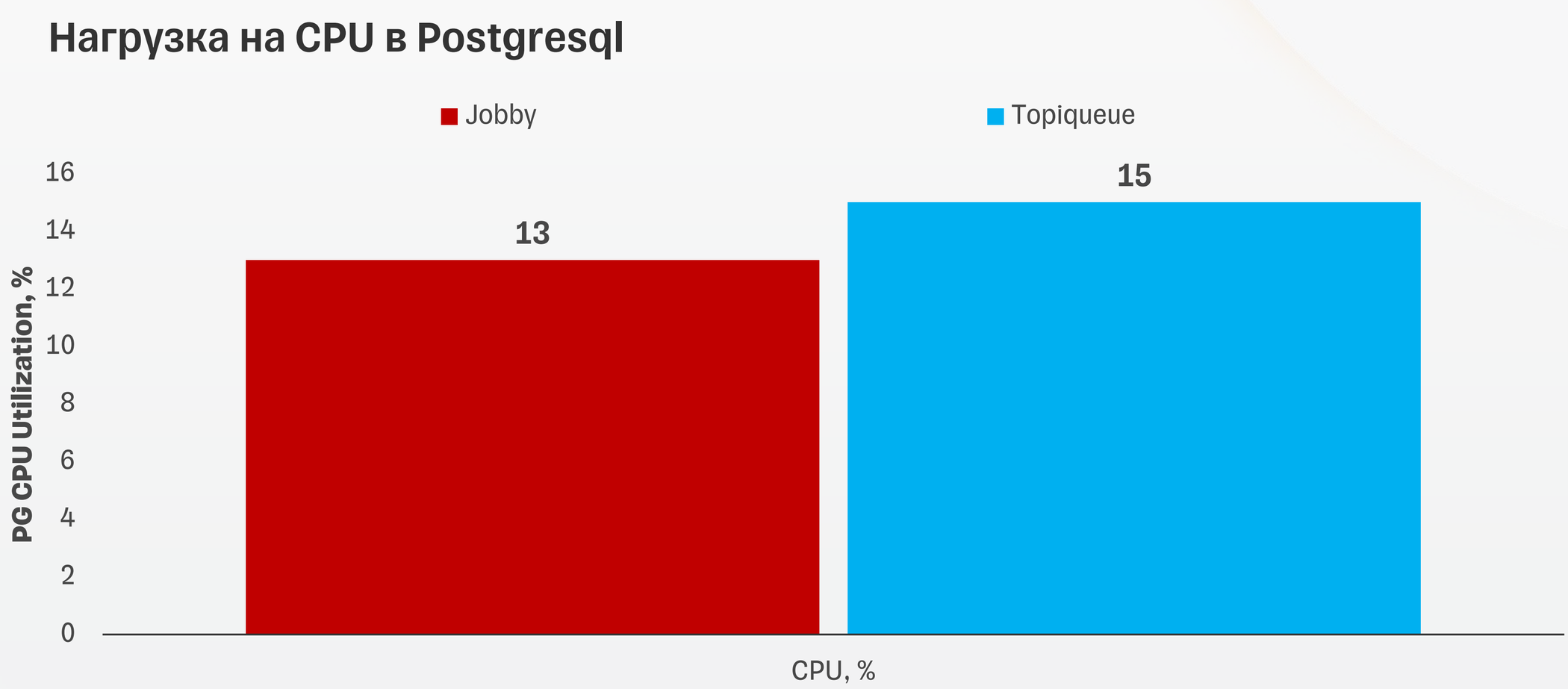
Затраченное время



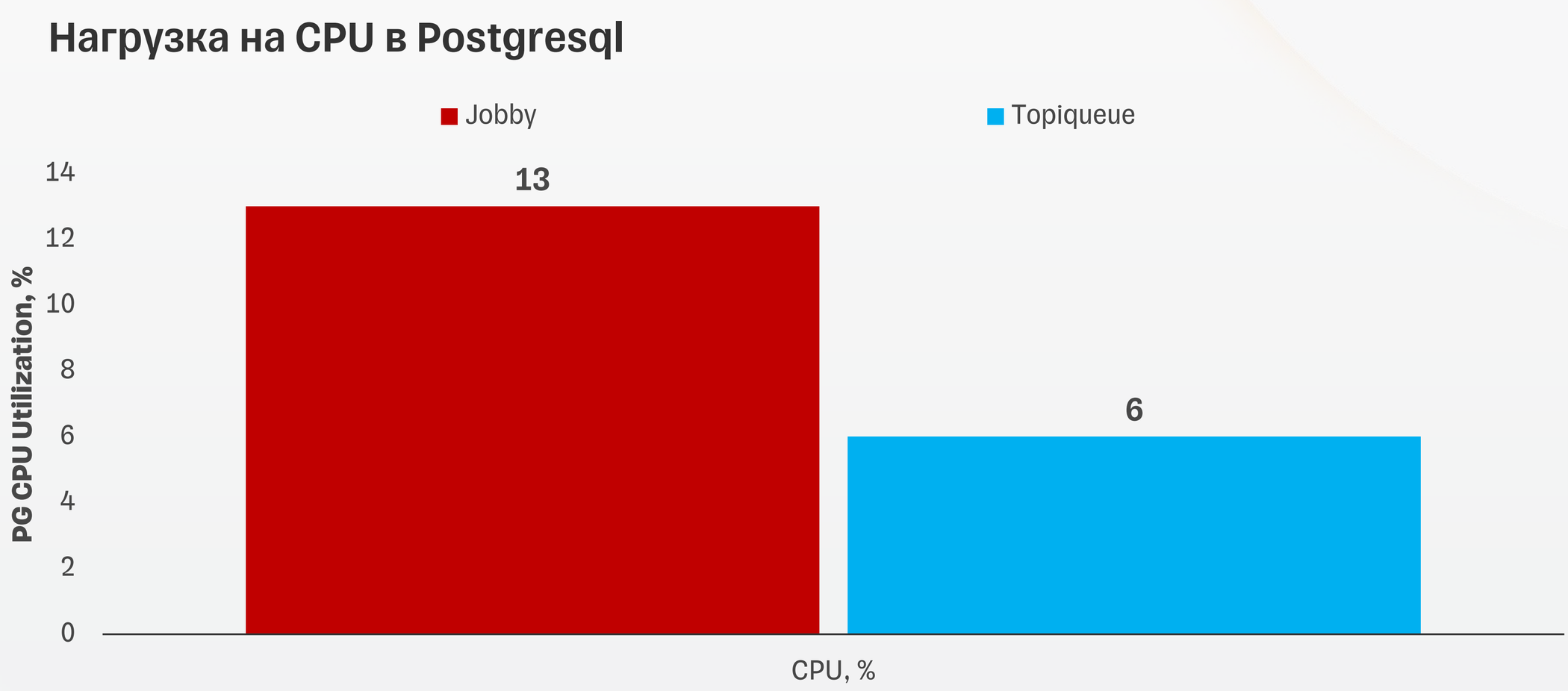
Расход памяти



Вставка и обработка, HandlerBatch = 1

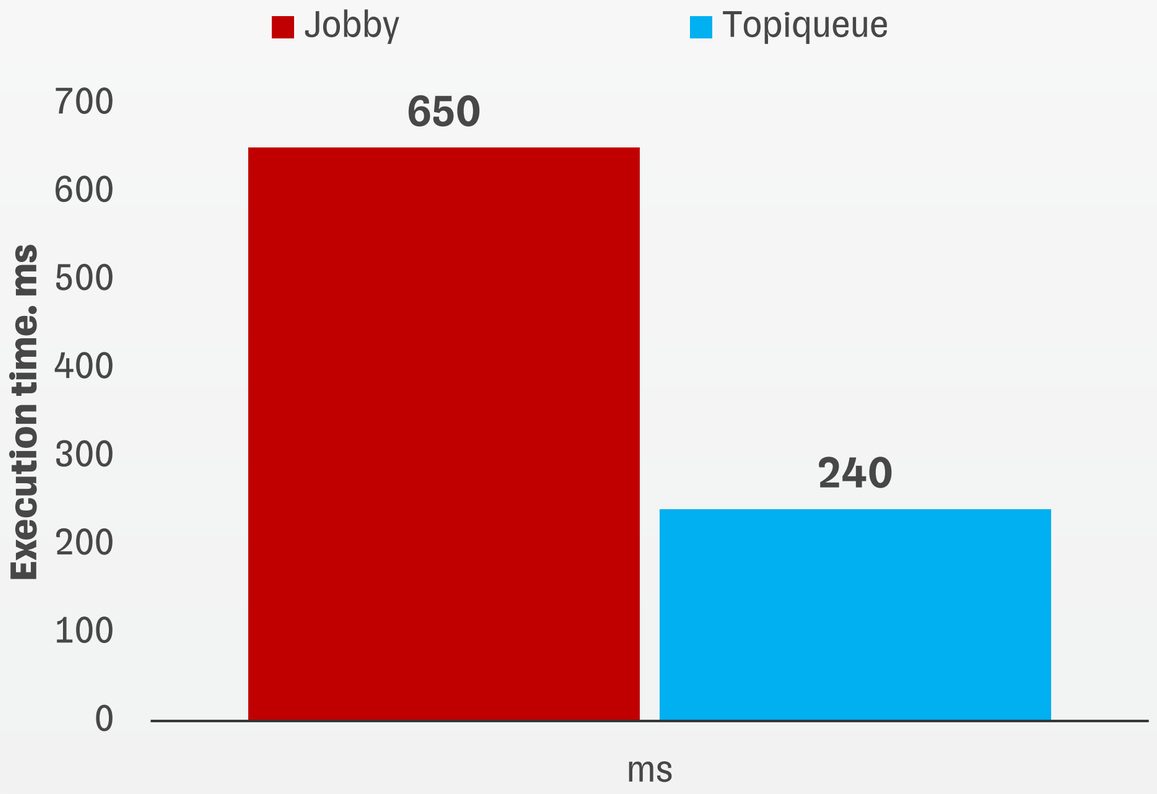


Вставка и обработка, HandlerBatch=10

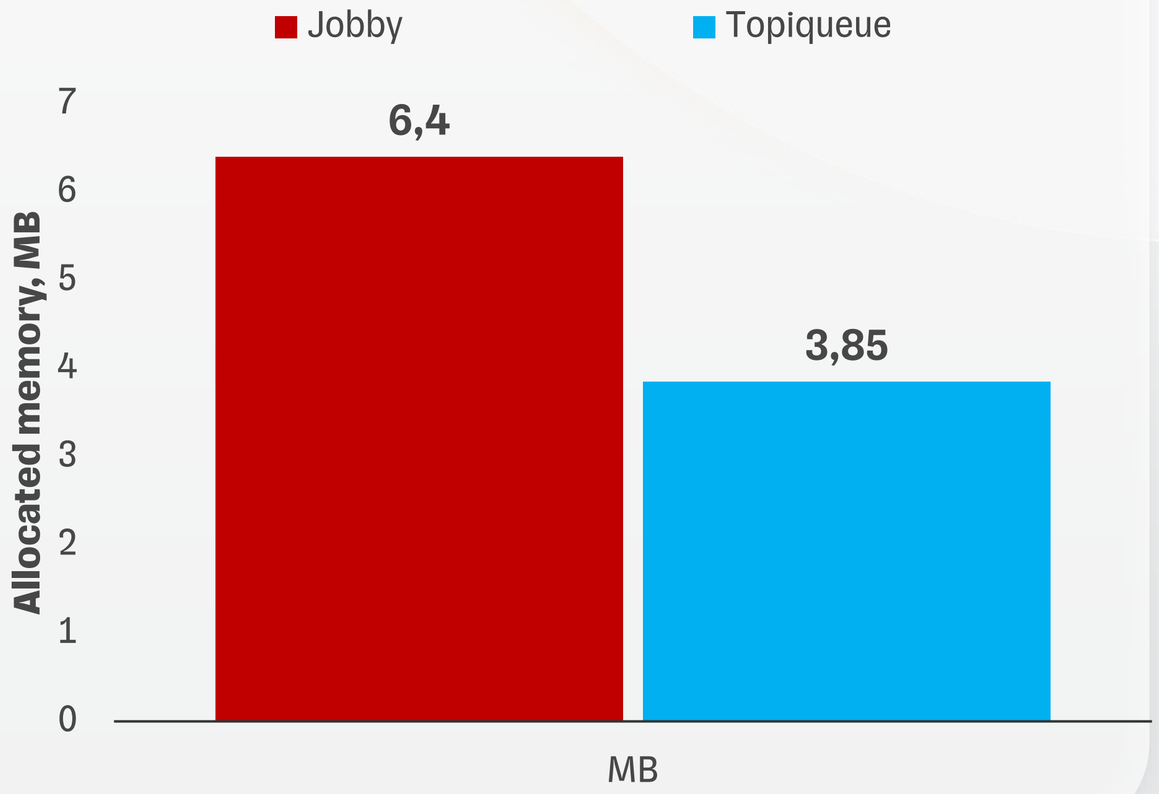


Вставка и обработка 1000 сообщений 10 потоков / партиций, HandlerBatch = 10

Затраченное время



Расход памяти



Итоги и выводы

Подход на основе статусов

- Нагрузка на БД (SELECT + UPDATE + DELETE)
- Сообщения не связаны между собой
- Просто распараллеливается и масштабируется
- К параллелизму сложно добавить упорядоченность

VS

Подход на основе смещений

- Эффективная работа с БД (INSERT ONLY подход)
- Сообщения жёстко связаны
- Сложно распараллеливать и масштабировать
- Упорядоченность из коробки

Подход на основе статусов

- Нагрузка на БД (SELECT + UPDATE + DELETE)
- Сообщения не связаны между собой
- Просто распараллеливается и масштабируется
- К параллелизму возможно добавить упорядоченность*

VS

Подход на основе смещений

- Эффективная работа с БД (INSERT ONLY подход)
- Сообщения жёстко связаны
- Возможно добавить параллелизм и масшт-ть*
- Упорядоченность из коробки

Сильные и слабые стороны

Подход на основе статусов

- Честный параллелизм
- Гибкость в отложенной обработке отдельных записей
- Создаёт мёртвые кортежи, провоцирует VACUUM

Подход на основе смещений

- Легче для БД (не провоцирует тяжелый VACUUM)
- Сверхэффективная пакетная обработка
- Сложность повторной отложенной обработки
- Высокий latency при перебалансировке партиций

Что выбрать

Личные рекомендации

Подход на основе статусов

Использовать с большинстве случаев

если не упираетесь в ресурсы БД

Подход на основе смещений

Использовать в экстремально нагруженных сценариях

когда критически важна высокая пропускная способность при низких расходах ресурсов БД

Примеры реализаций

Jobby

Topiqueue

https://t.me/dev_and_more

Спасибо!