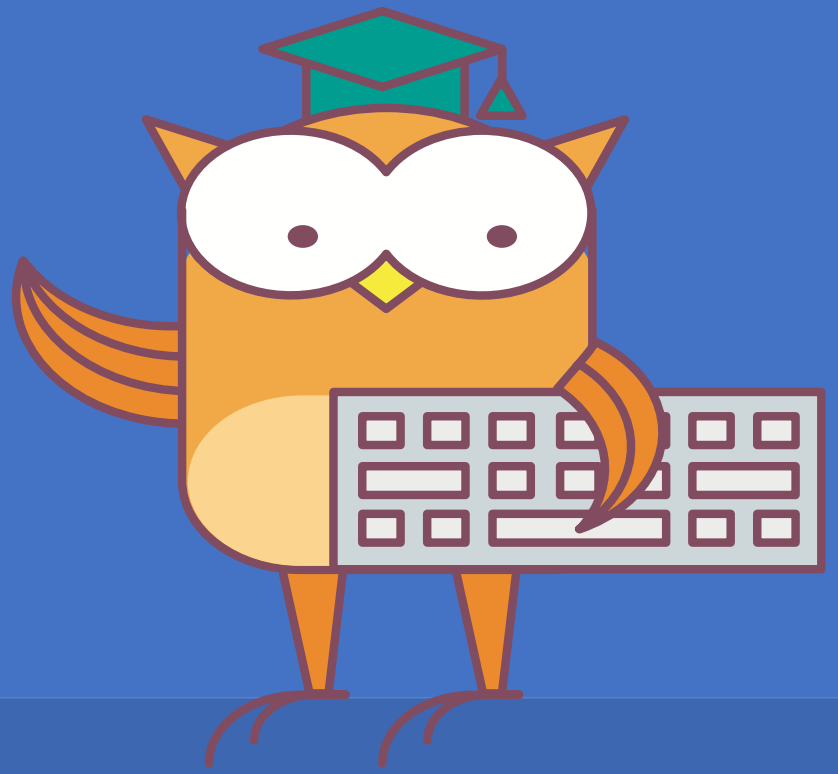
Stream processing

Архитектор ПО



Щетинников Стас

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

Преподаватель

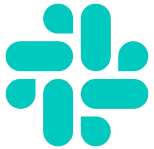
Непомнящий Евгений

- 15 лет программировал контроллеры на C++ и руководил отделом разработки
- 3 года пишу на Java
- Последнее время пишу микросервисы на Java в Мвидео

Правила вебинара

Активно участвуем

Задаем вопросы в чат



Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу

Карта курса

5 Архитектор

1 Инфраструктурные паттерны

3 Распределенные системы

2 Коммуникационные паттерны

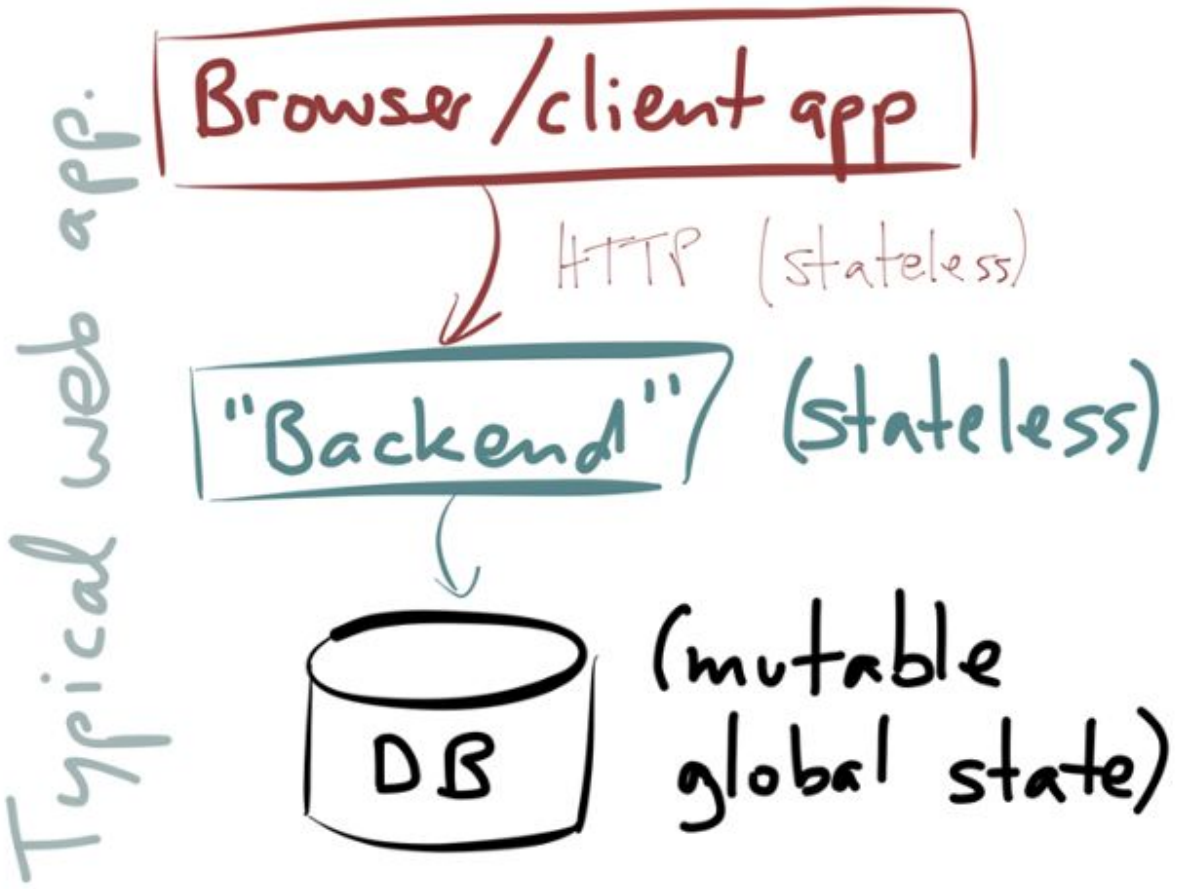
4 Децентрализованные системы

Опрос по программе - каждый месяц в ЛК

Карта вебинара

- Stream Processing
- Демо
- Надежная отправка сообщений

Database inside out



https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

Типичные базы данных хранят состояние внутри себя.

Но в рамках микросервисной архитектуры какая-часть этого состояния должна быть пошарена (shared) с другими микросервисами.

По большому счету одно и то же состояние, но имеющее разные представления, может хранится в разных сервисах.

Database inside out

https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

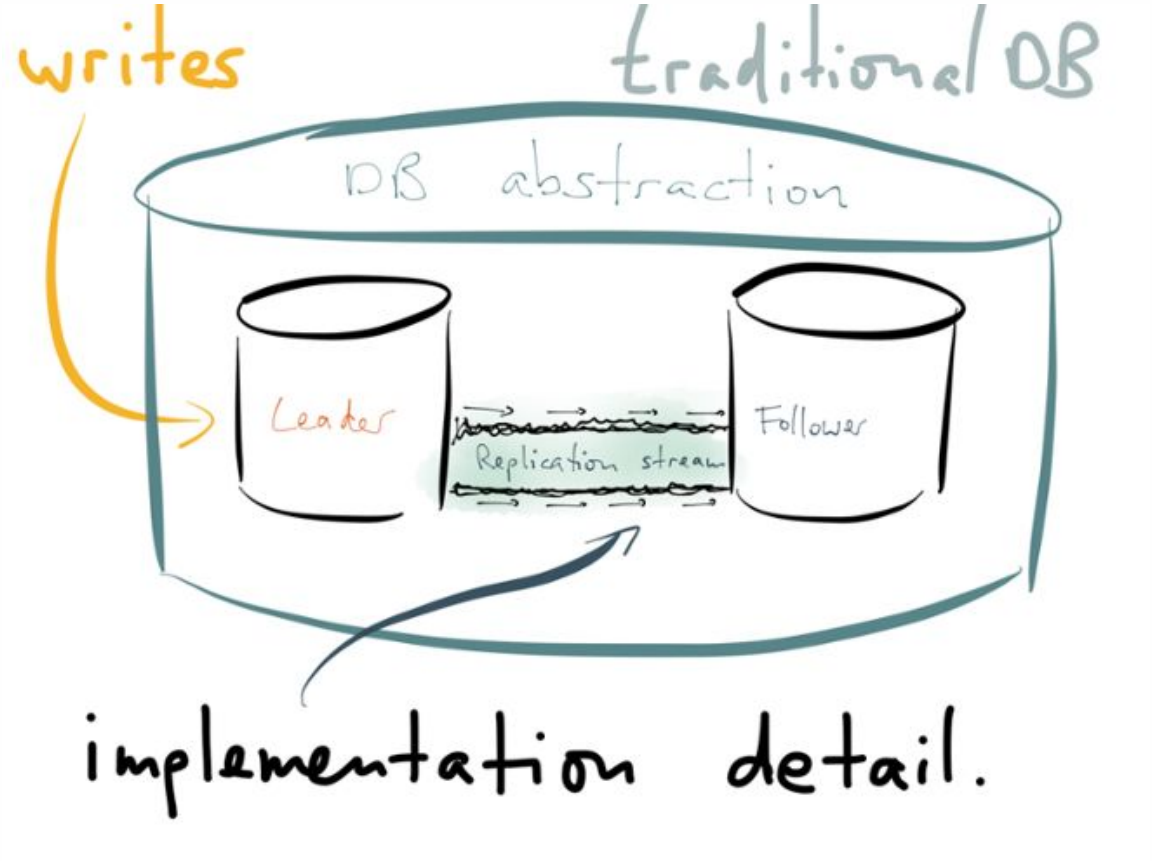
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

Давайте посмотрим на то, как устроена репликация в обычной БД с ведущим узлом.

Для того, чтобы поддерживать в консистентном состоянии мастер и реплику используется репликация лога транзакций, который из себя представляет бесконечный поток событий об изменениях в ведующем узле.

Database inside out



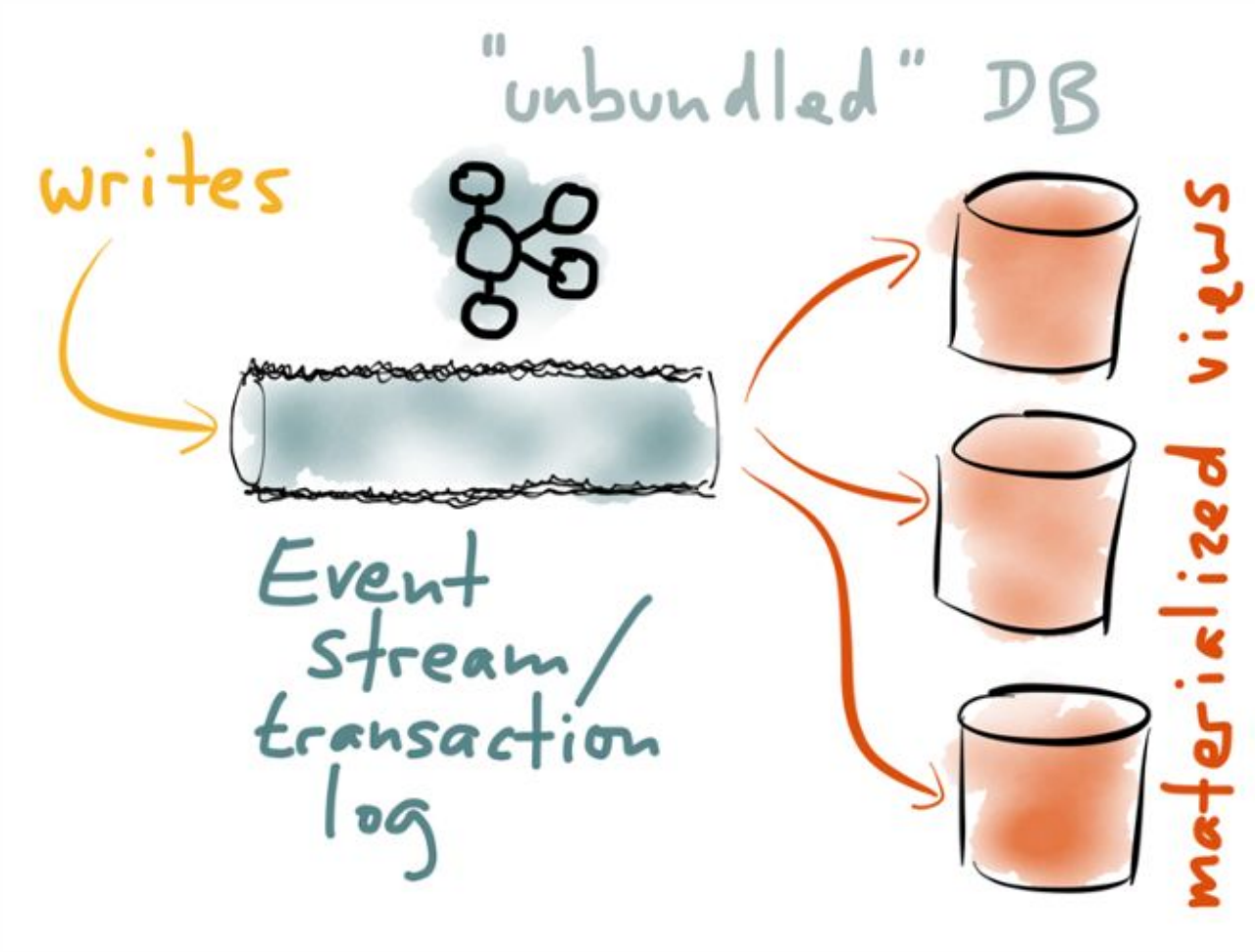
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

Давайте вывернем нашу архитектуру, и replication log из деталей реализации сделаем корневым элементом.

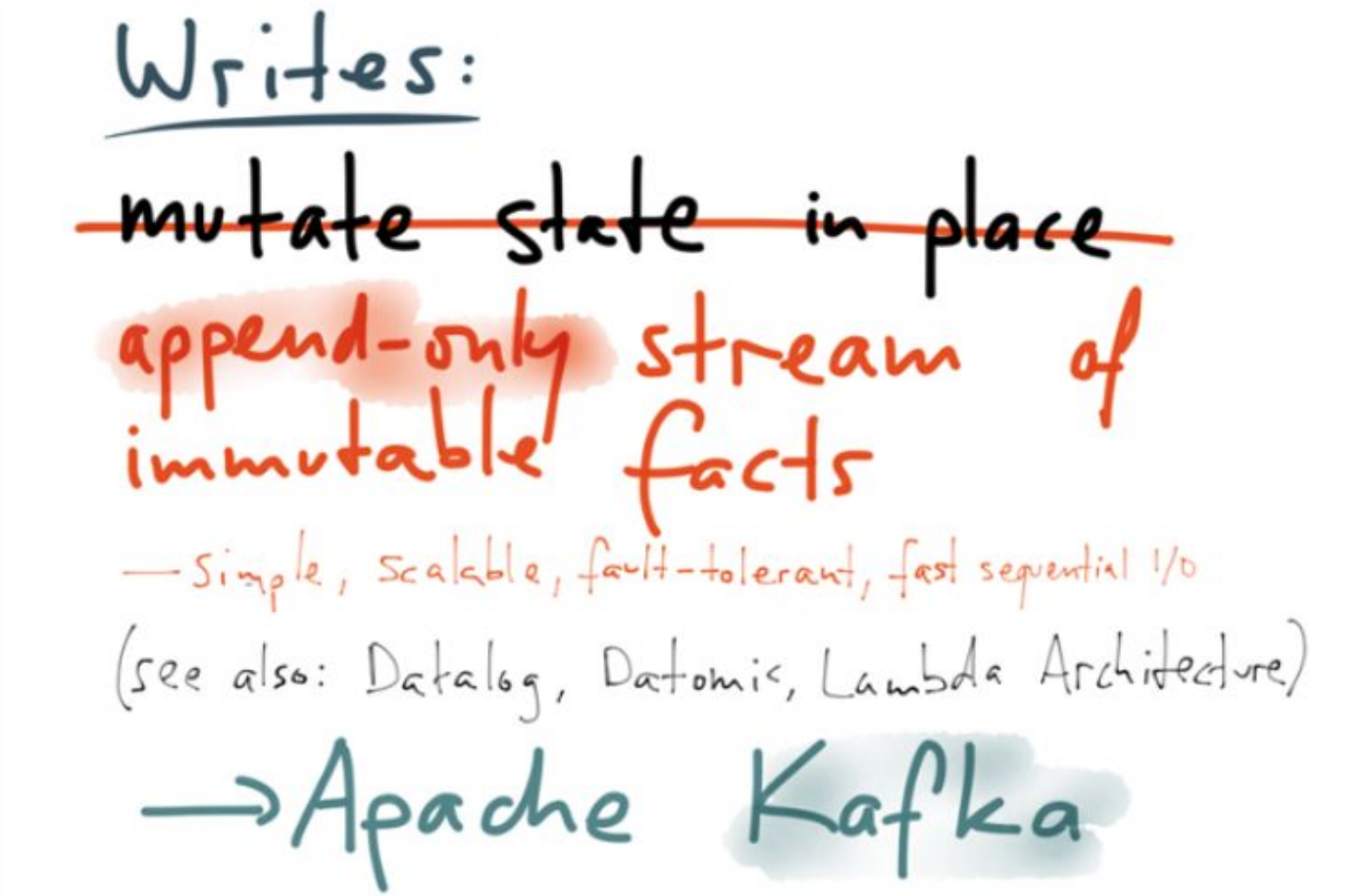
Давайте хранить состояние, которое шарят друг с другом разные микросервисы хранить не в виде состояния (проекции, представления и т.д) а в виде лога событий это состояние меняющих.

Database inside out



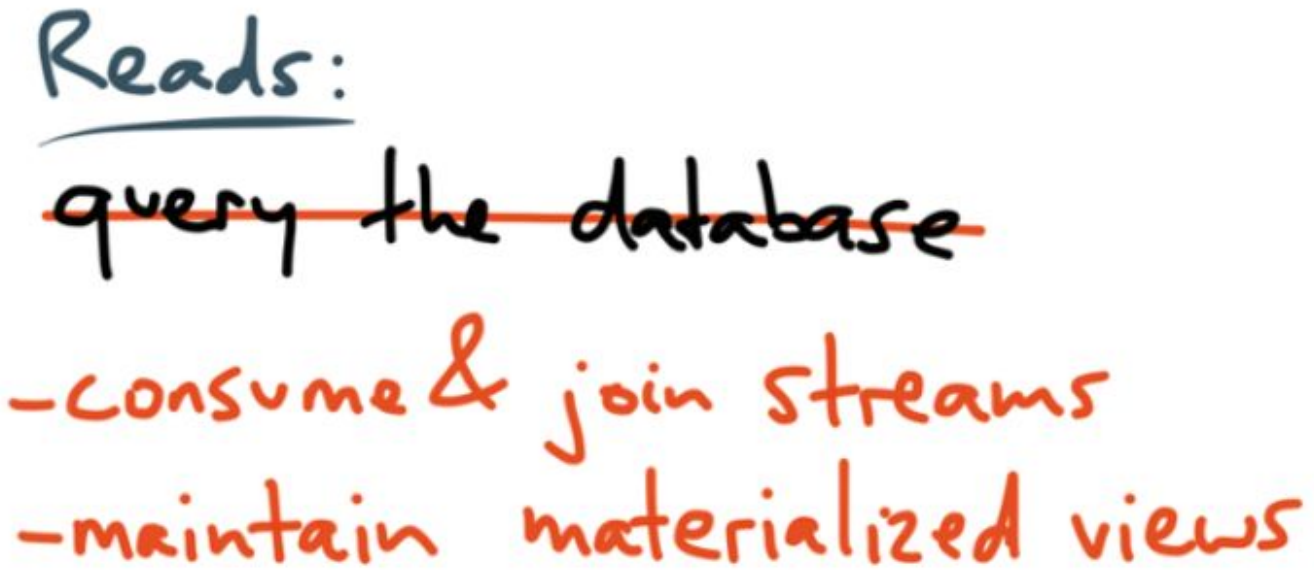
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out



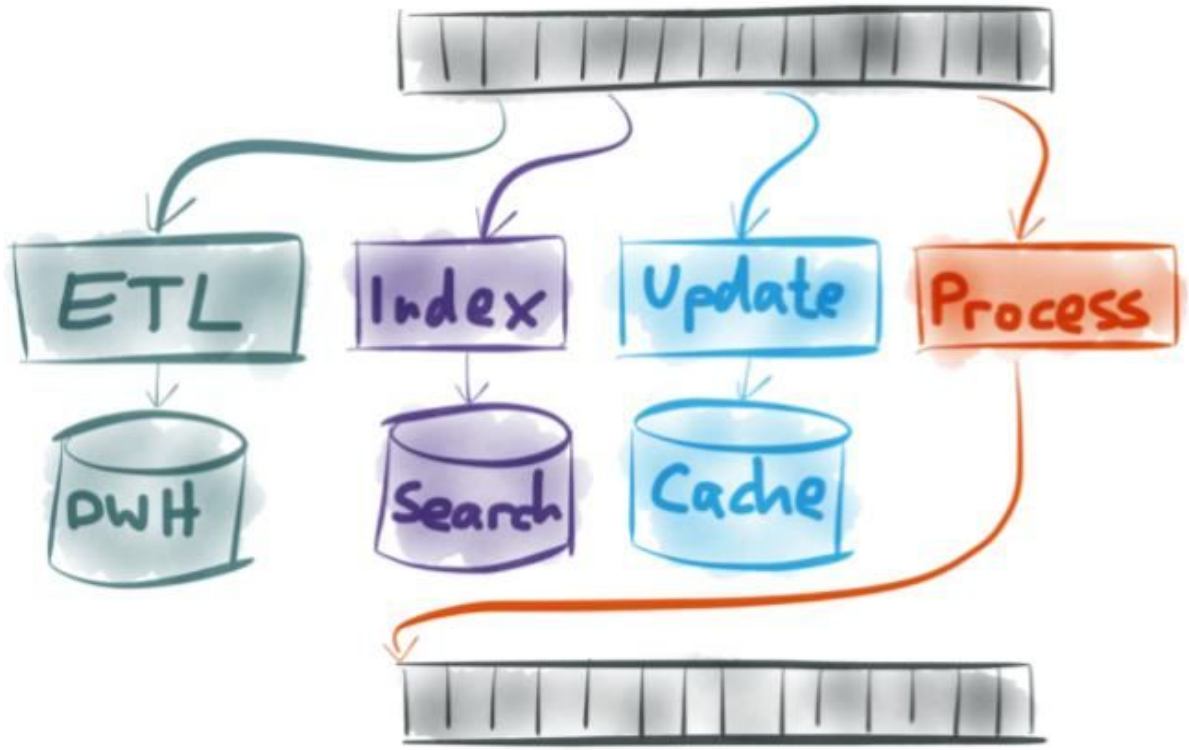
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out



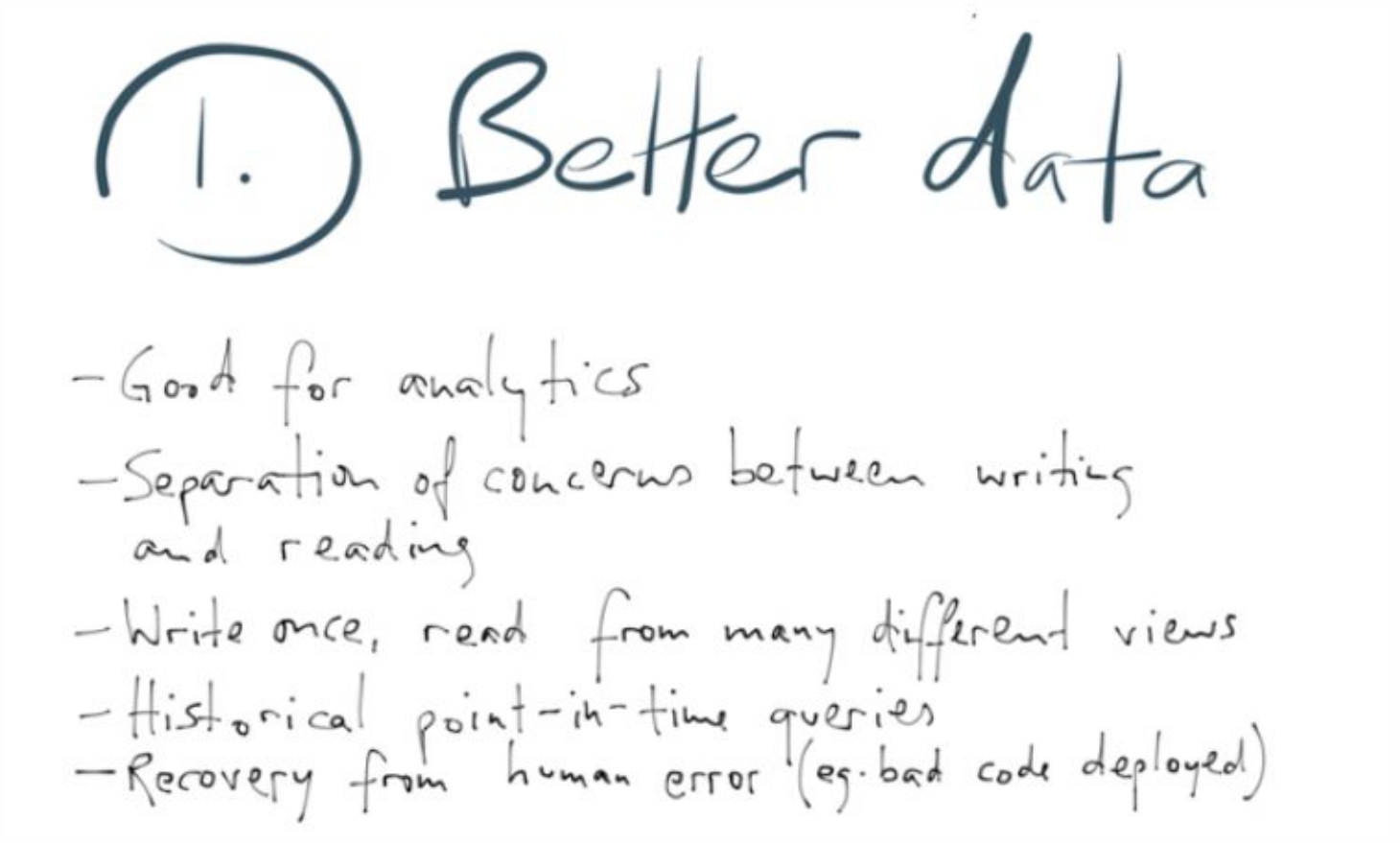
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out



https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out



https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

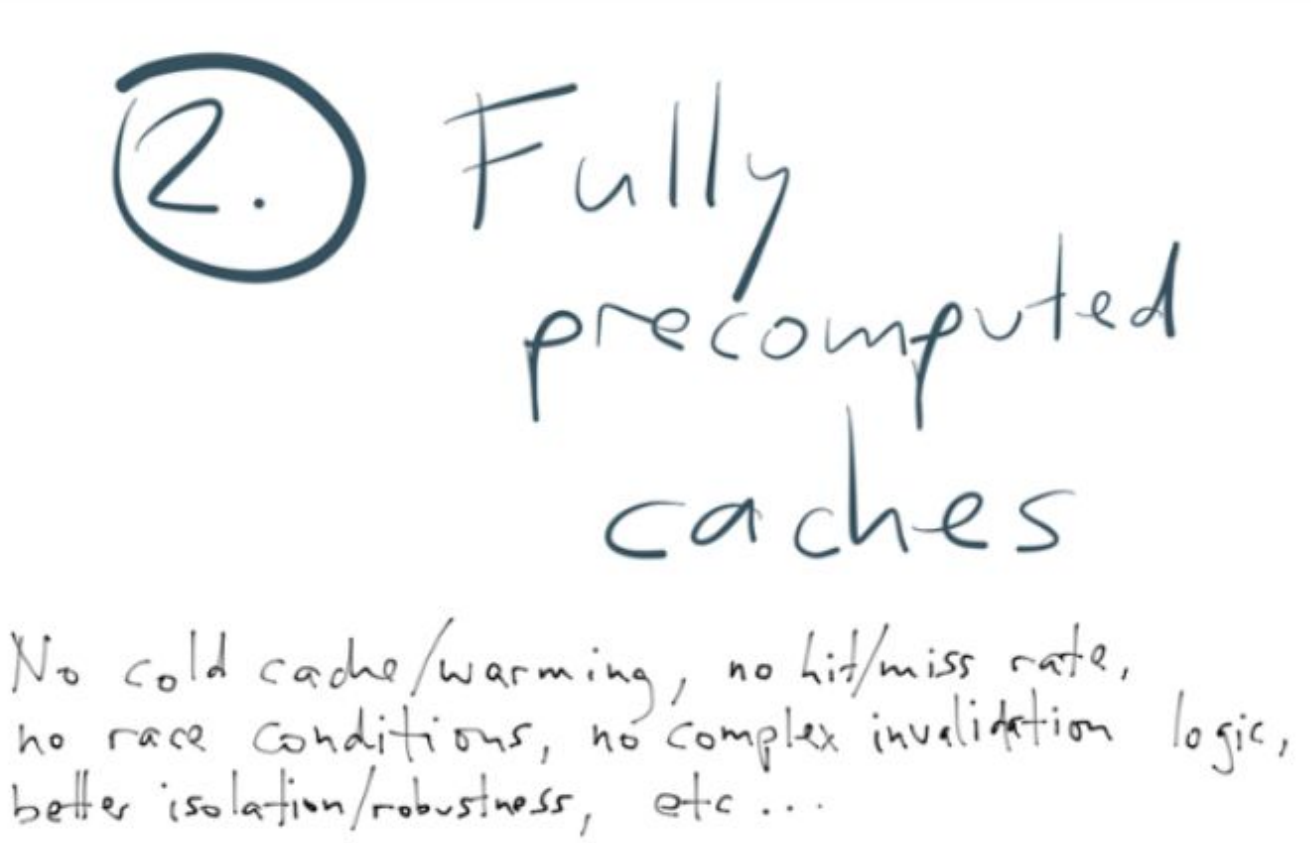
Что мы получаем?

Лучшее представление данных

- Поток событий для аналитиков – намного лучше просто состояния
- Разделение данных для записи и чтения
- Пиши один раз, и можешь читать из множества разных представлений для чтений
- Лучшая защита от ошибок (человеческий фактор)
- Темпоральные запросы и восстановление

https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out



https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

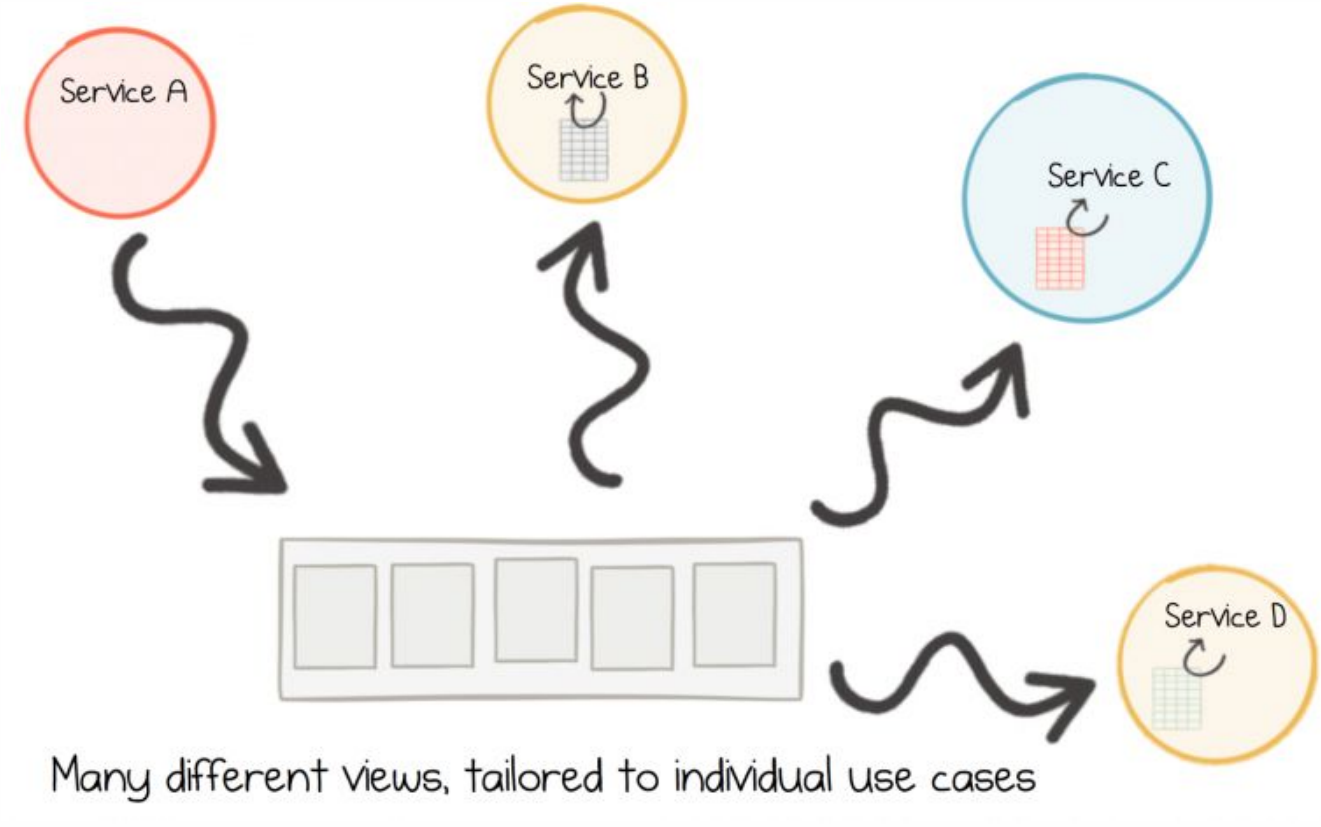
Database inside out

Что мы получаем?

Хорошие кэши:
Больше никакого разогрева кэшей, hit/miss rate, никаких гонок, никакой больше сложной логики инвалидации, лучшая изоляция и надежность.

https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

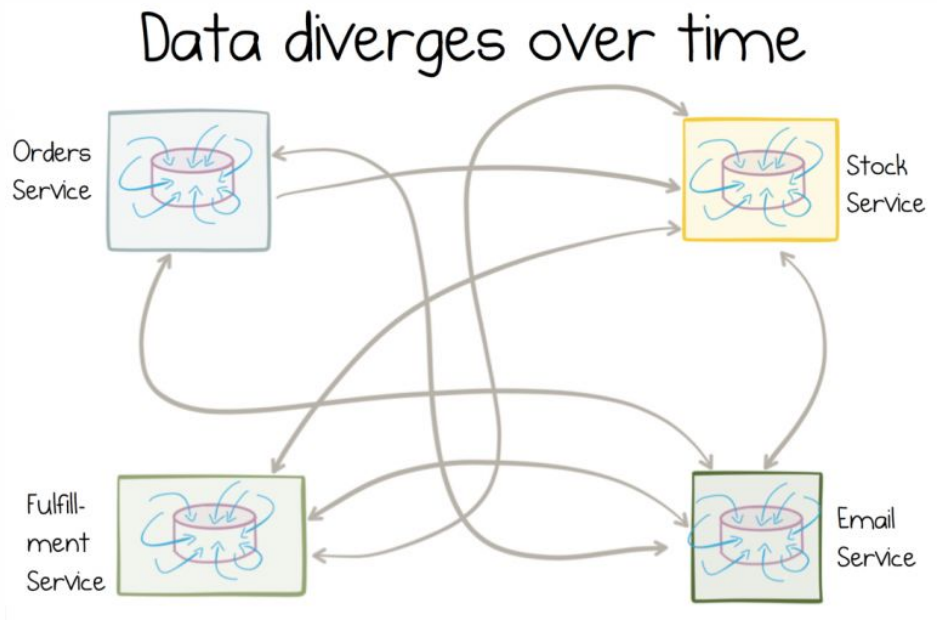


https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

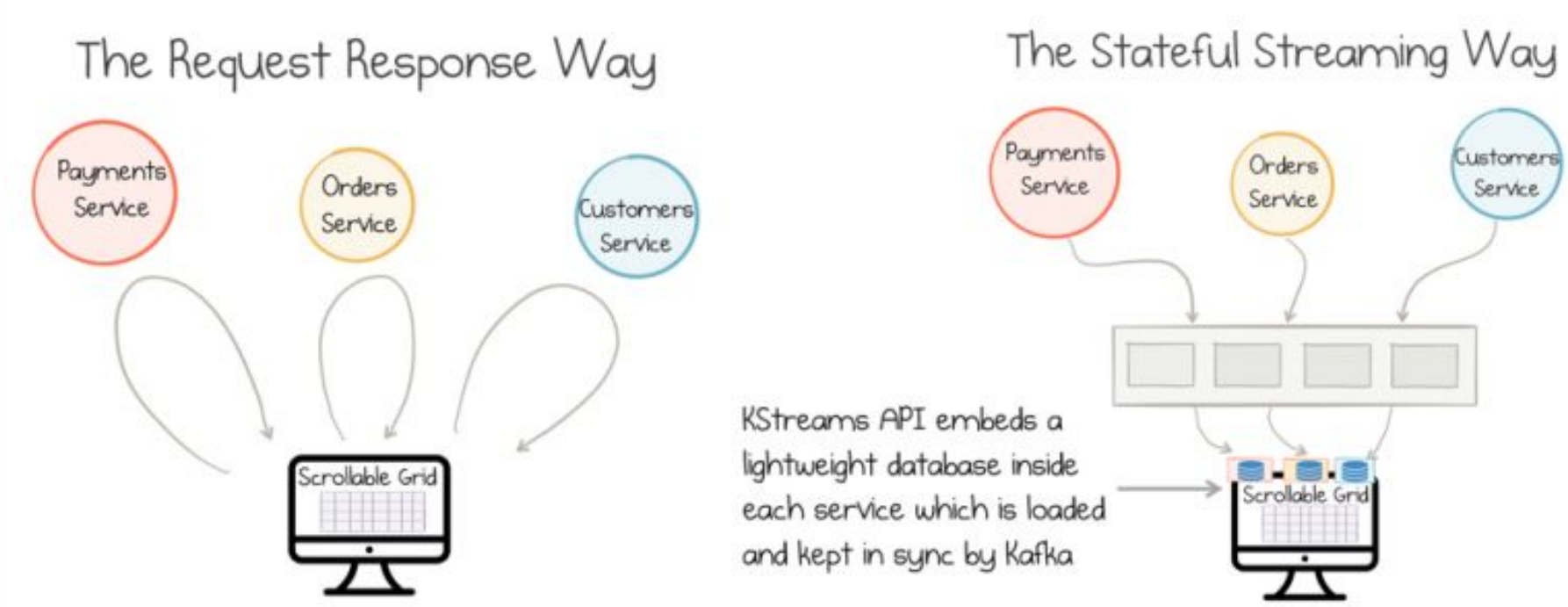
Database inside out

Как получить состояние из другого сервиса? Сделать синхронный запрос к сервису (типичный api composition).

Но с течением времени мы понимаем, что в разных сервисах нам нужно разное представление этих данных и синхронный API может превратиться в big ball mud.



Database inside out



https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Database inside out

Соответственно, нам нужна специализированная БД, которая бы могла хранить поток событий, и могла бы эти изменения транслировать дальше.

Такой БД является, например, Kafka.

Kafka – не только брокер сообщений, но и база данных.
Kafka предназначена для того, чтобы быть хранилищем, и брокером событий (сообщений).

https://www.confluent.io/blog/okay-store-data-apache-kafka/

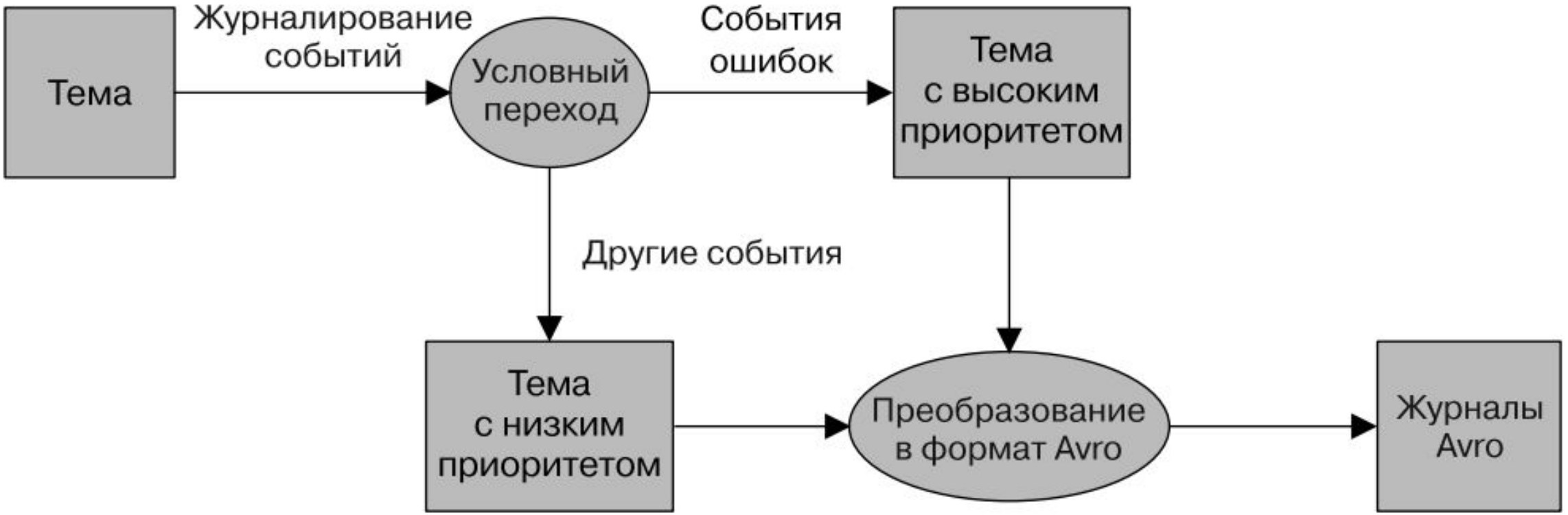
Характеристики потоков событий

- Неограниченность
- Упорядоченность
- Неизменяемость
- Повторяемость

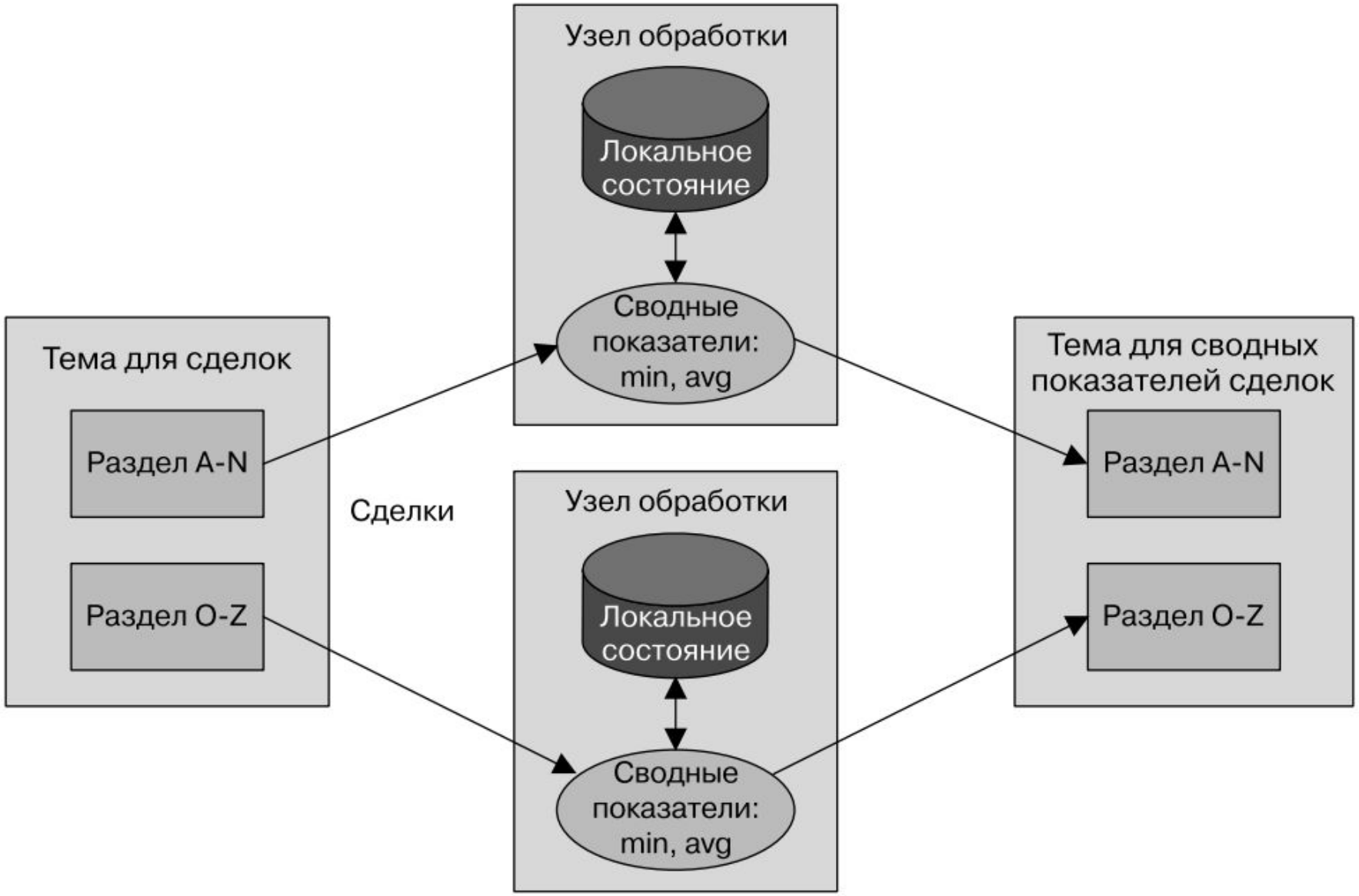
Основные понятия потоковой обработки

- Время - создания, записи, обработки
- Состояние обработки - stateless и statefull
- Временные окна
  - Размер окна
  - Насколько часто окно сдвигается
  - Когда можно обновлять

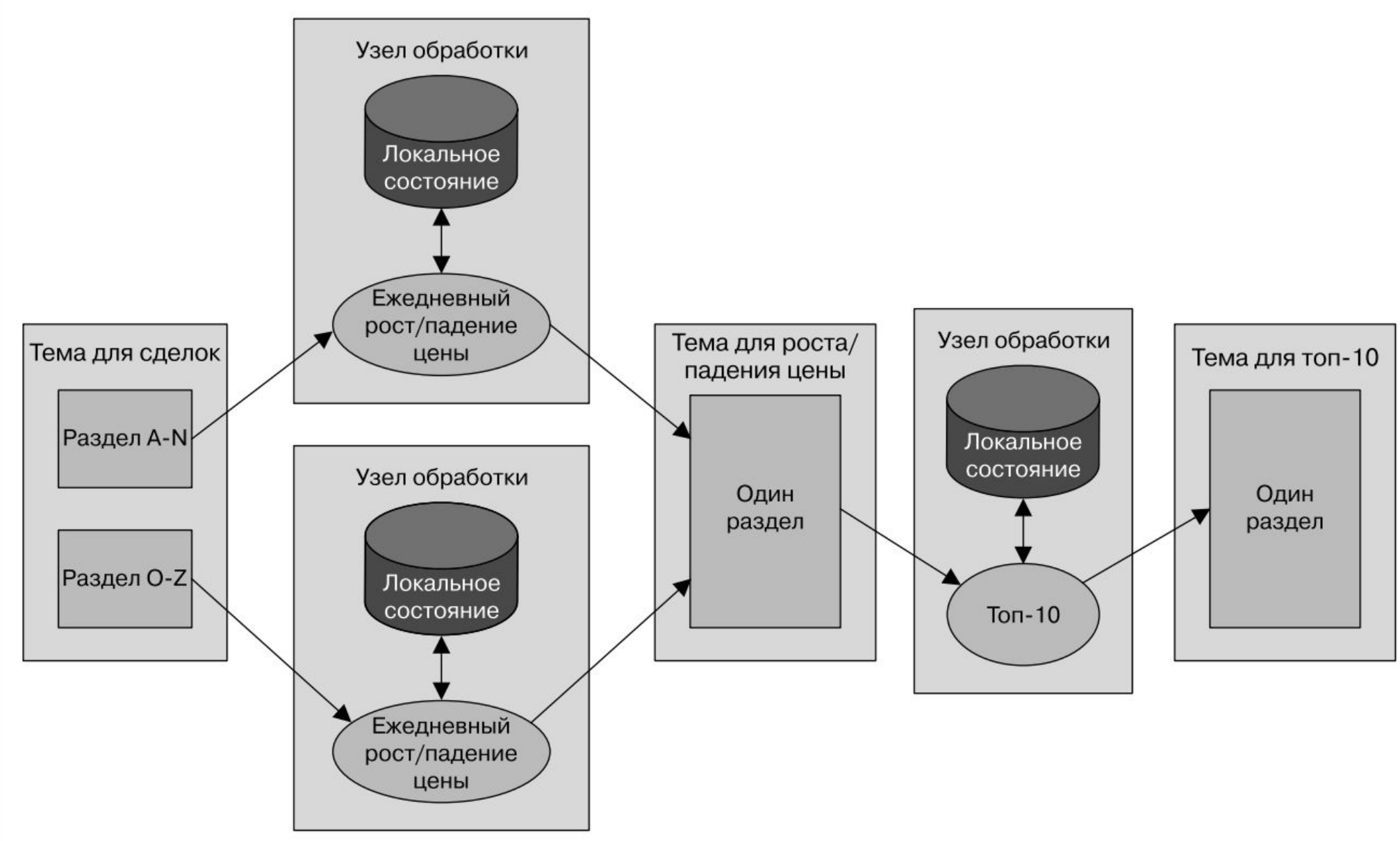
Паттерны проектирования - обработка по отдельности



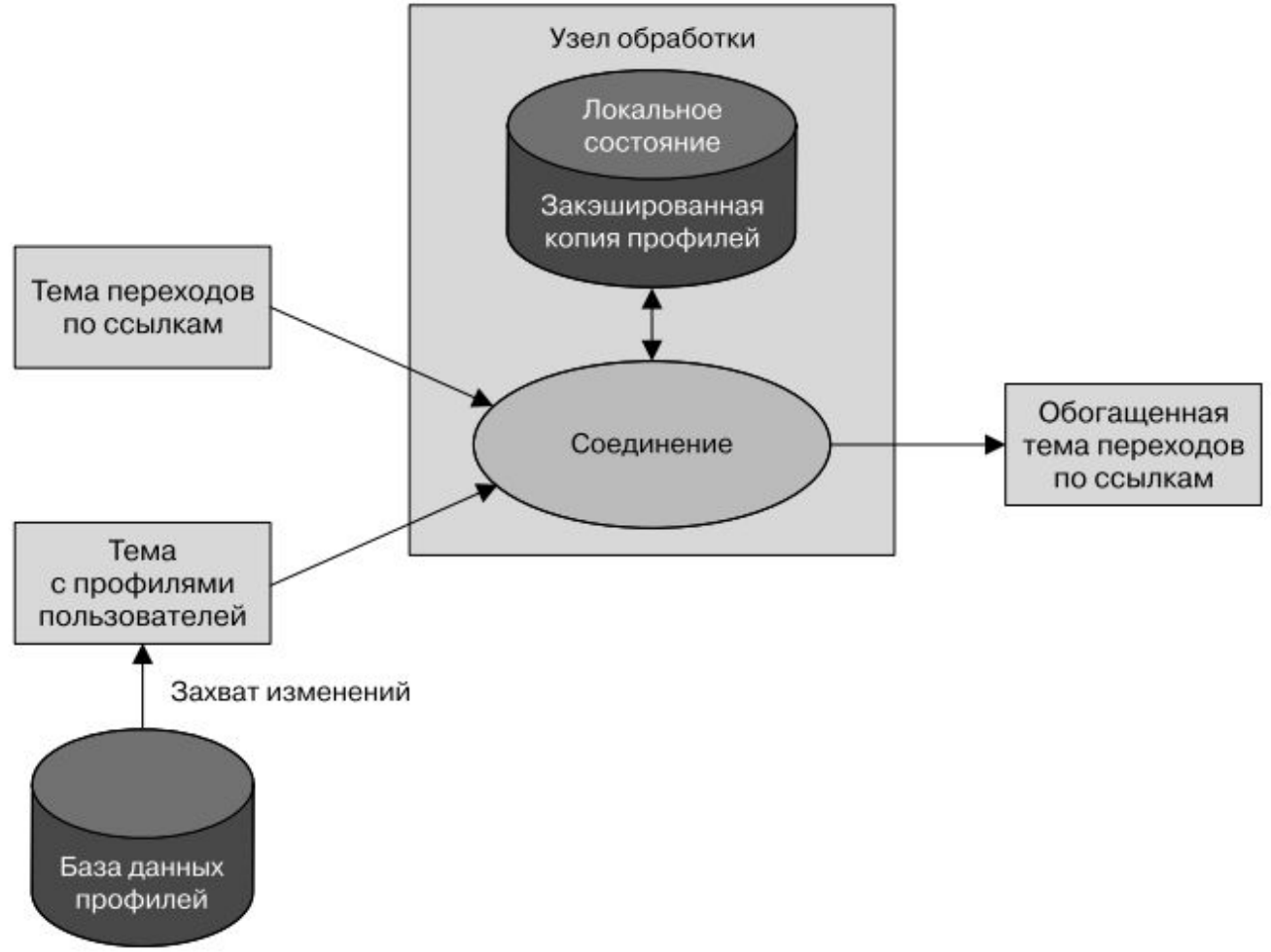
Паттерны проектирования - локальное состояние



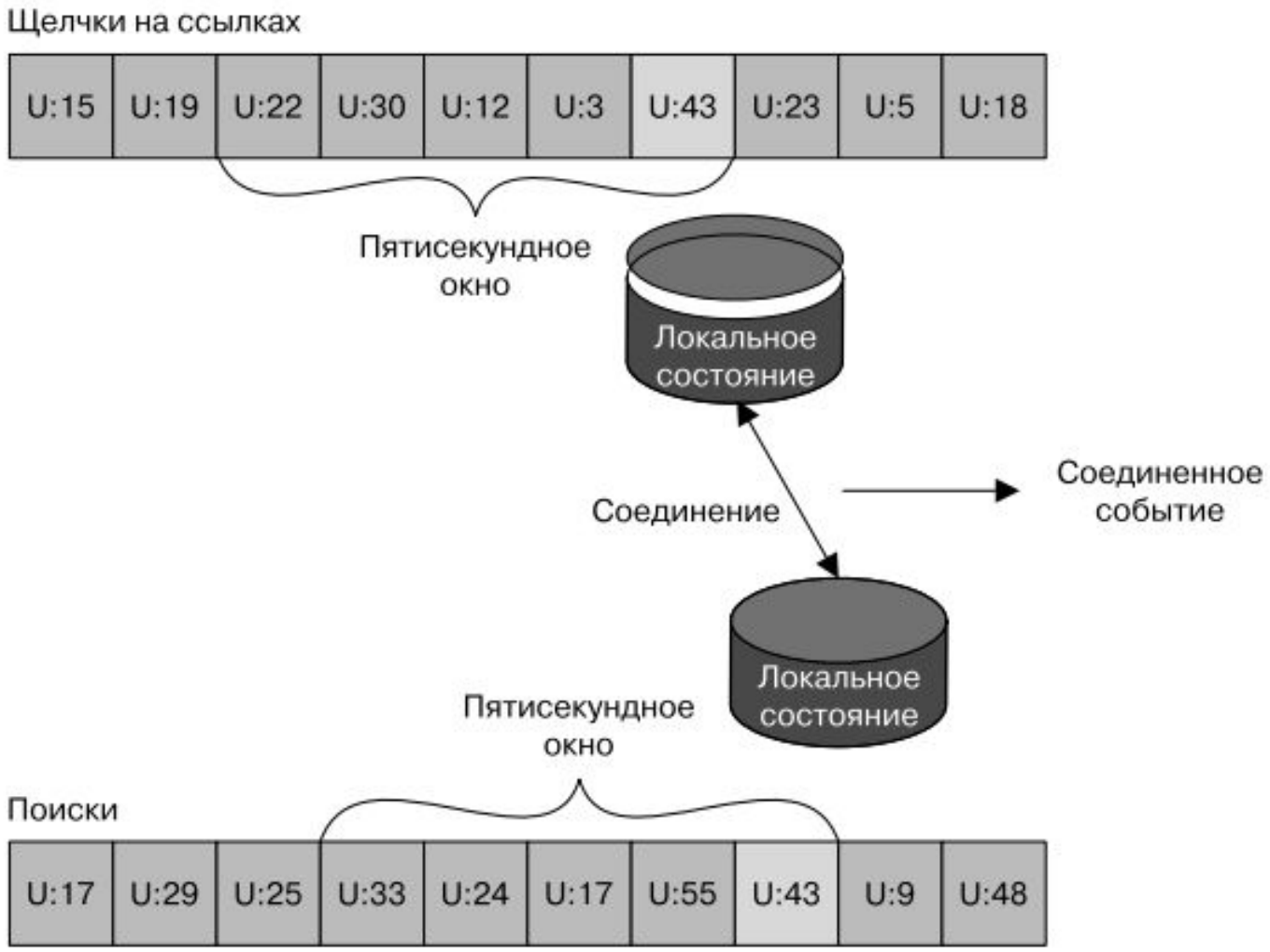
Паттерны проектирования - многоэтапная обработка



Паттерны проектирования - внешний справочник



Паттерны проектирования - соединение потоков

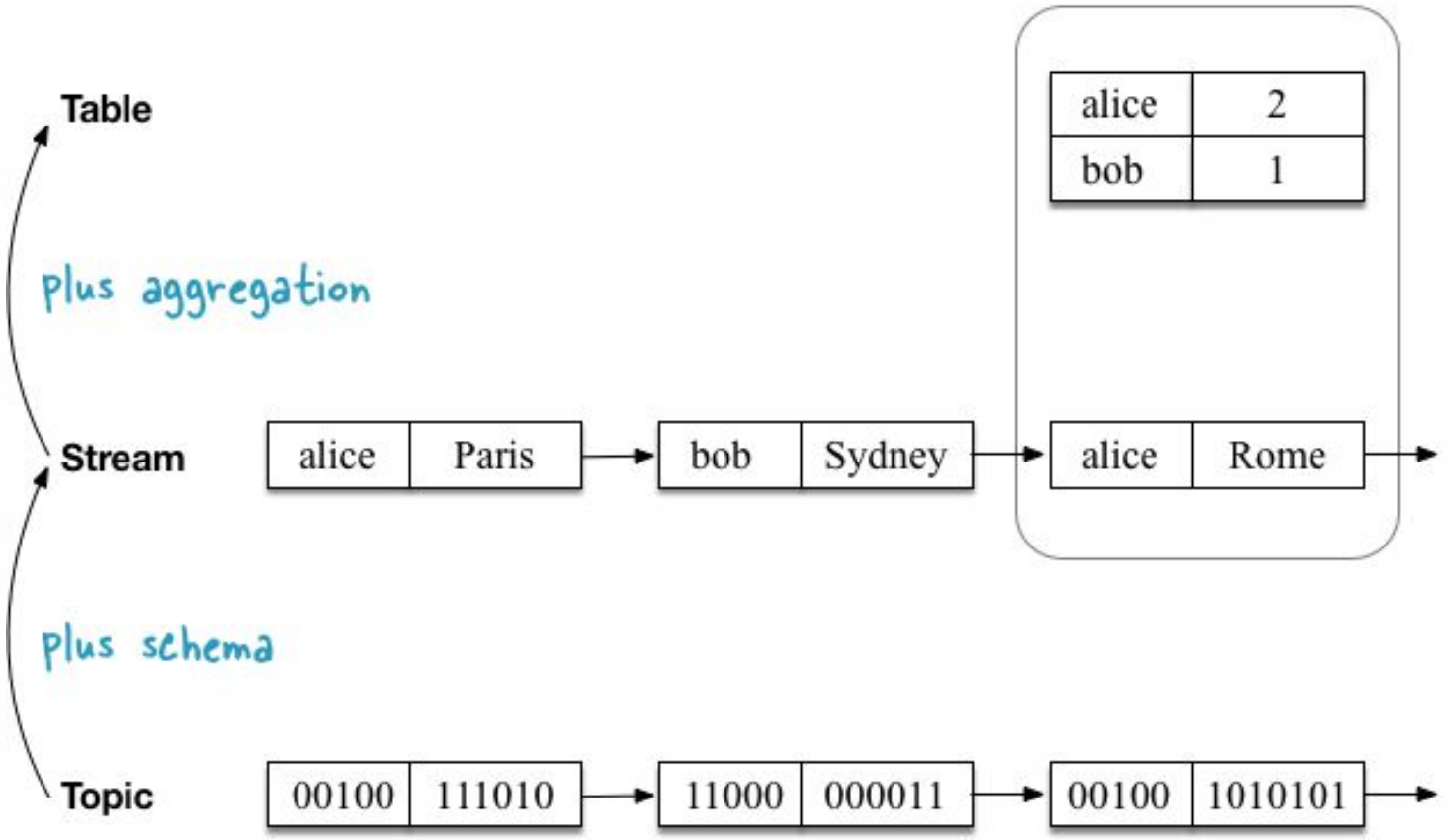


Паттерны проектирования - несвоевременное поступление

- Игнорировать или обрабатывать?
- Обработать - у нас нет перезапуска
- Обновление результатов

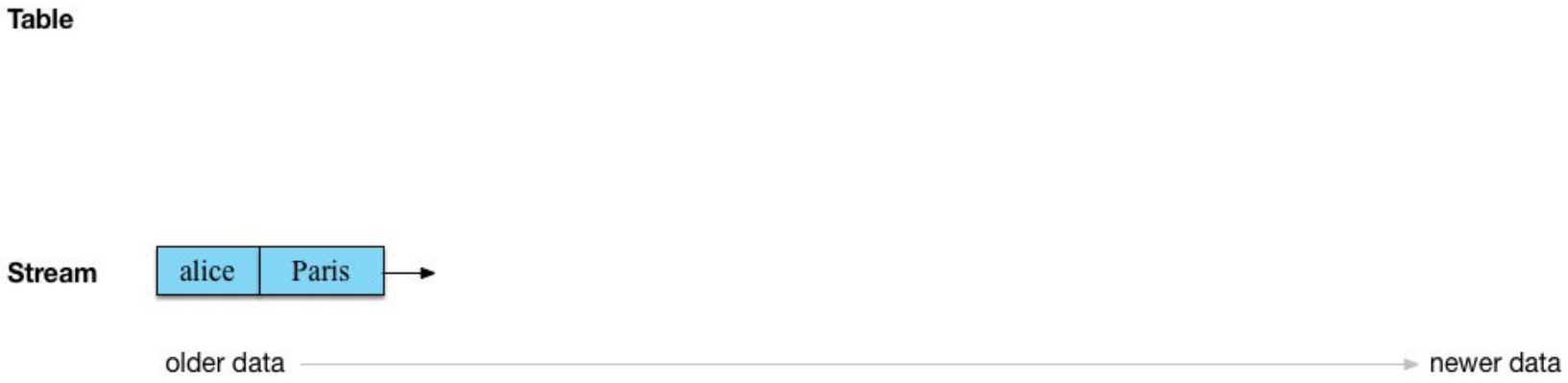
Kafka Streams

Stream – это поток типизированых сообщений (событий).
Table – это агрегация потока (стрима)



https://kafka.apache.org/documentation/streams/

Kafka Streams



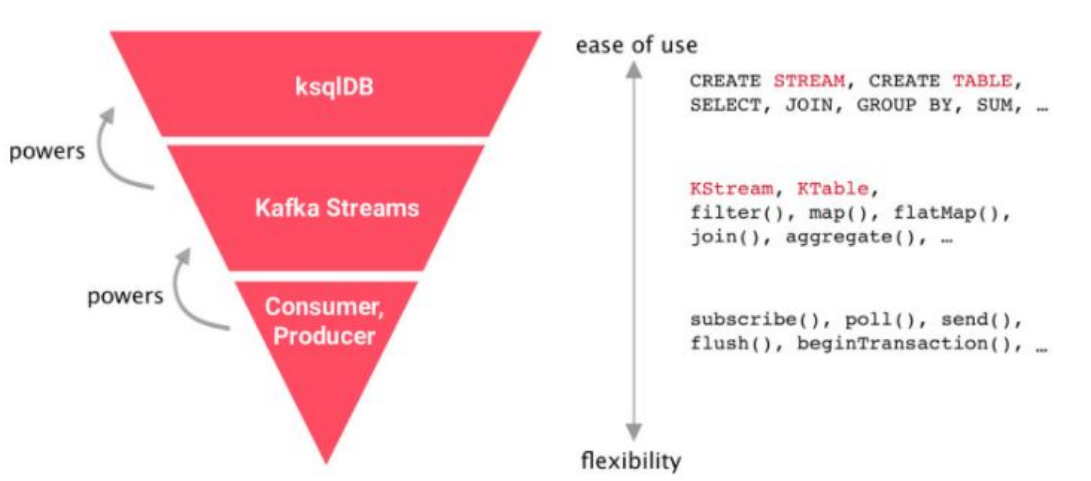
https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

Stream processing

Очевидно, потоки рождаются в разных сервисах, обрабатываются, передаются дальше и т.д.

Обработку эти потоков в микросервисах можно

1) делать «руками» внутри самого сервиса
2) воспользоваться клиентскими библиотеками для stream processing
3) использовать внешние сервисы для хранения состояния и stream processing



Kafka Streams

Kafka Streams API – это библиотека для Java, которая позволяет сделать работу с потоками событий или сообщений проще.

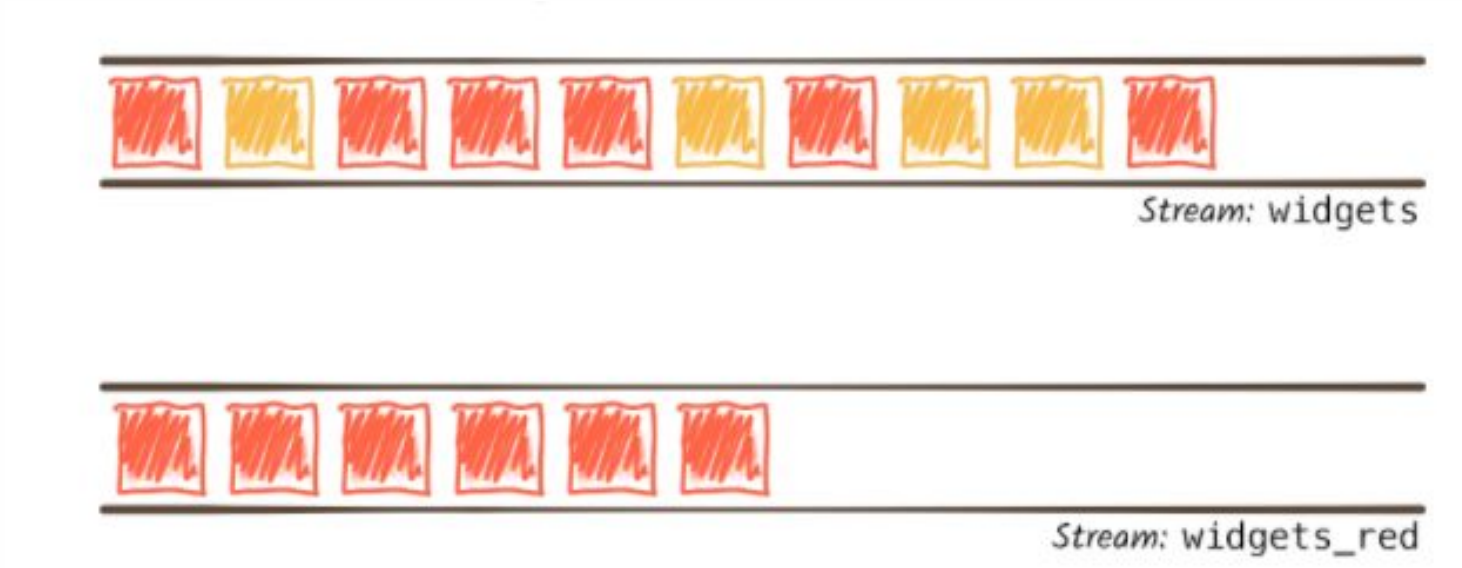
https://kafka.apache.org/documentation/streams/

Stream SQL

В последнее время получает популярность обработка стримов в SQL-like синтаксе.

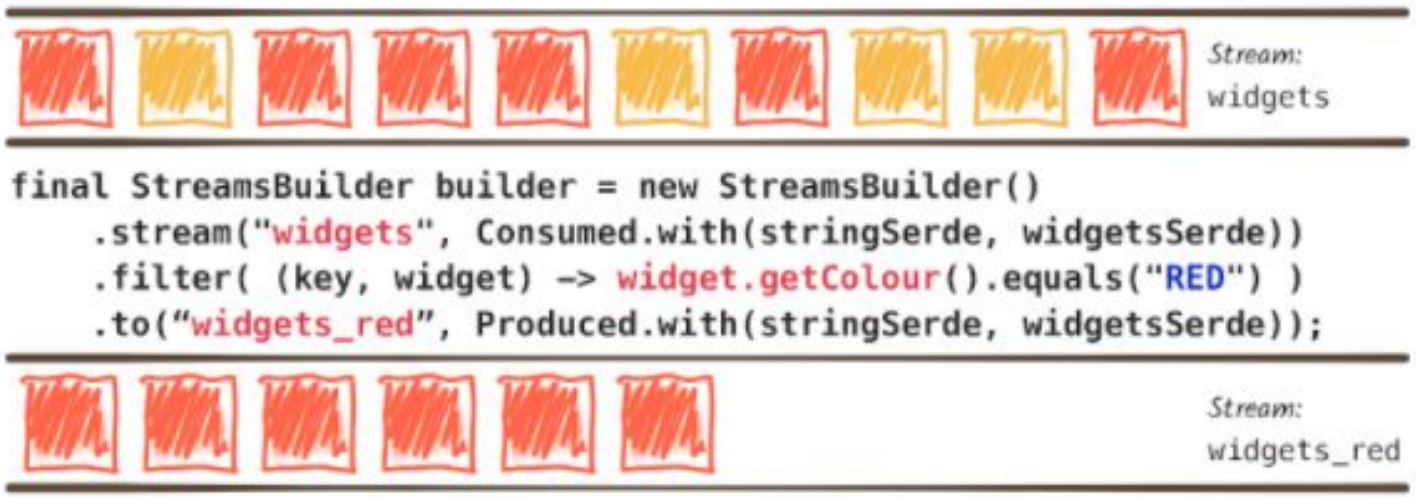
https://talks.rmoff.net/LjZAS7/slides

Stream SQL



https://talks.rmoff.net/LjZAS7/slides

Stream SQL

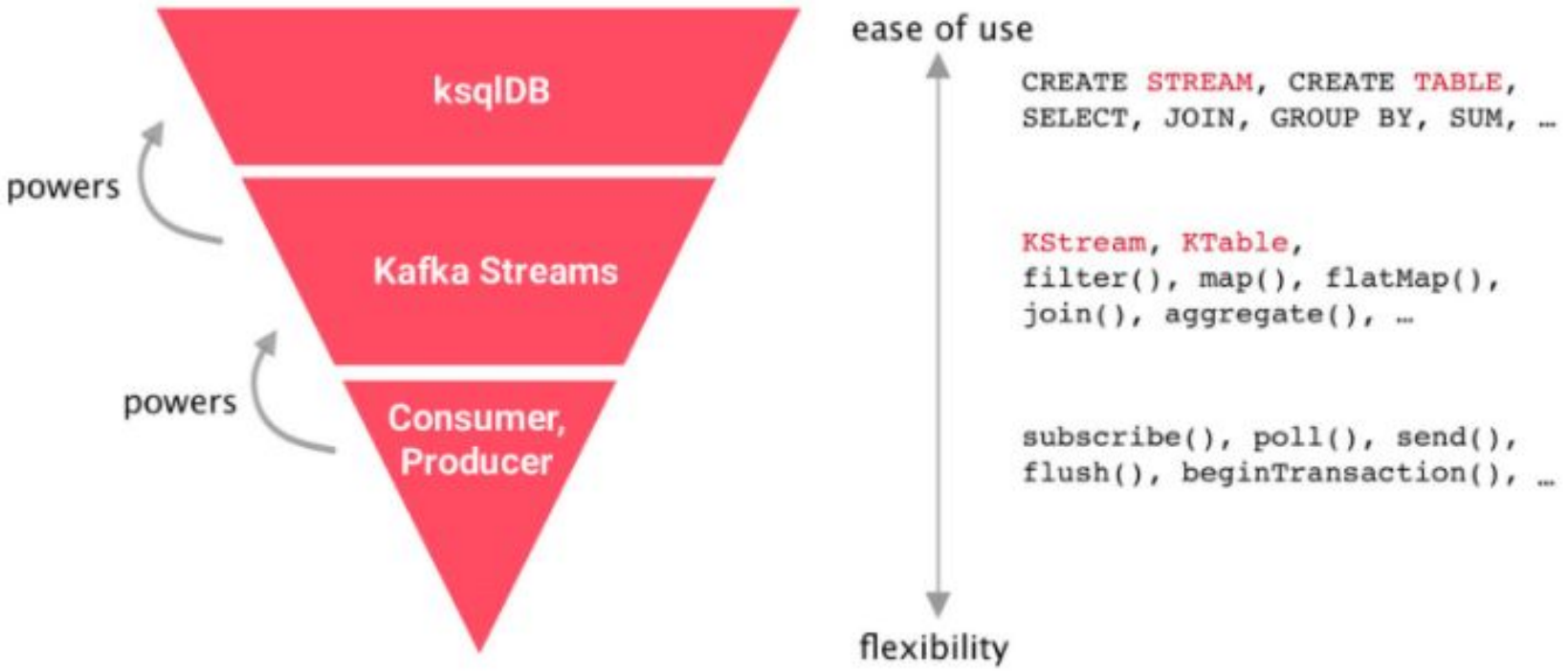


https://talks.rmoff.net/LjZAS7/slides

Stream SQL

https://talks.rmoff.net/LjZAS7/slides

Stream SQL

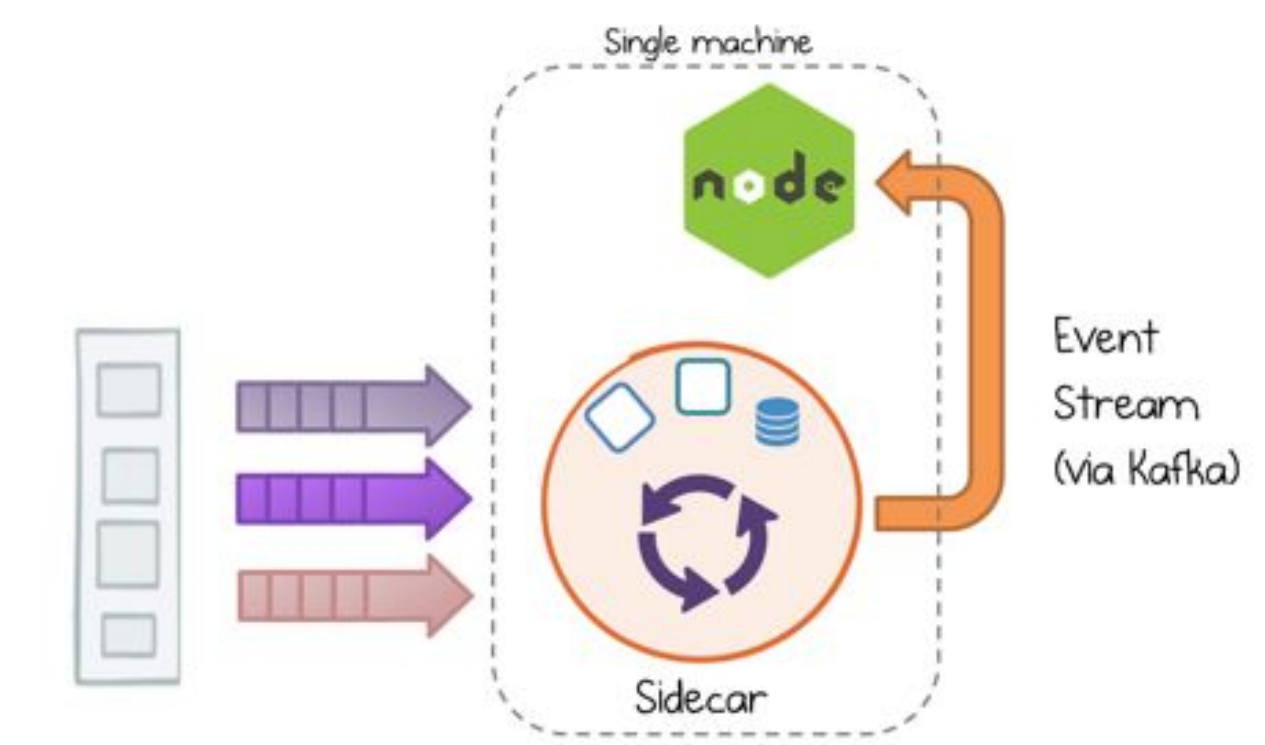


https://talks.rmoff.net/LjZAS7/slides

ksqlDB

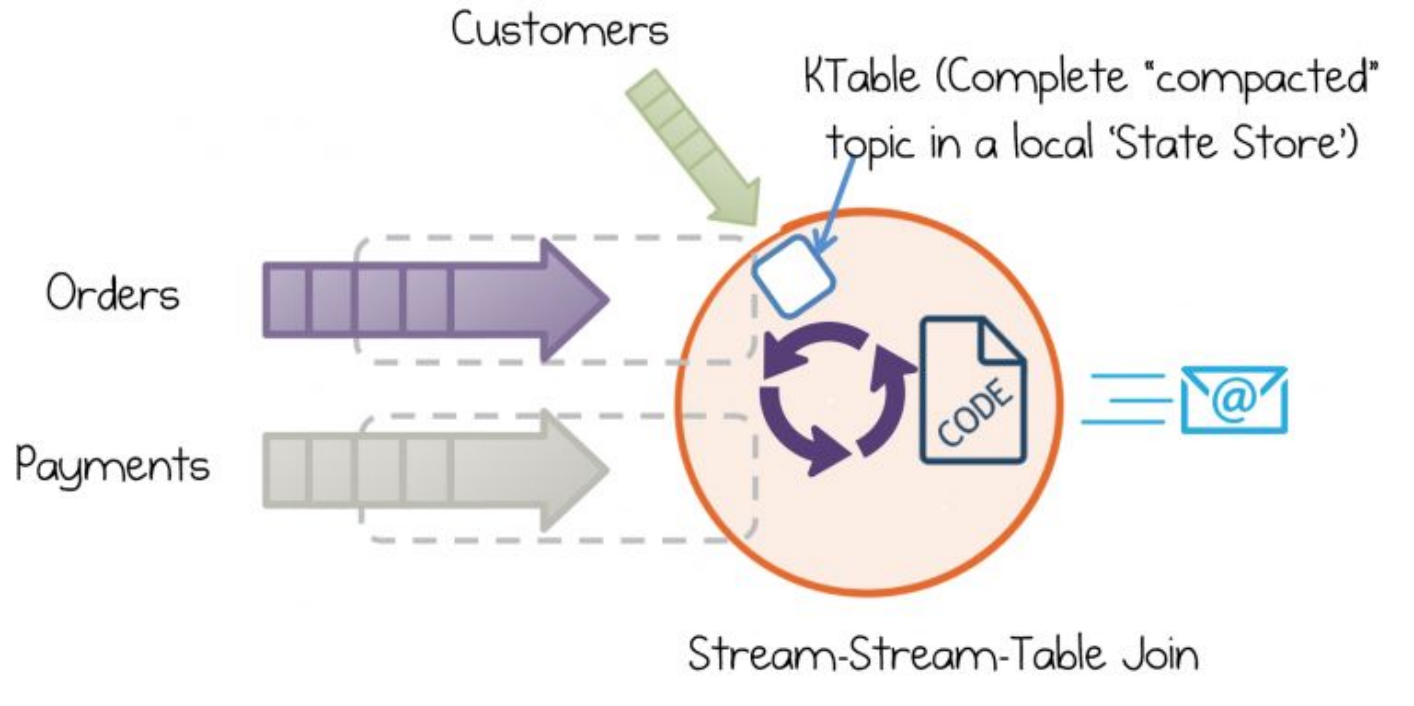
https://talks.rmoff.net/LjZAS7/slides

ksqlDB



https://www.confluent.io/blog/data-dichotomy-rethinking-the-way-we-treat-data-and-services/

ksqlDB



https://talks.rmoff.net/LjZAS7/slides

Демо

https://github.com/schetinnikov-otus/arch-labs/tree/master/stream-processing

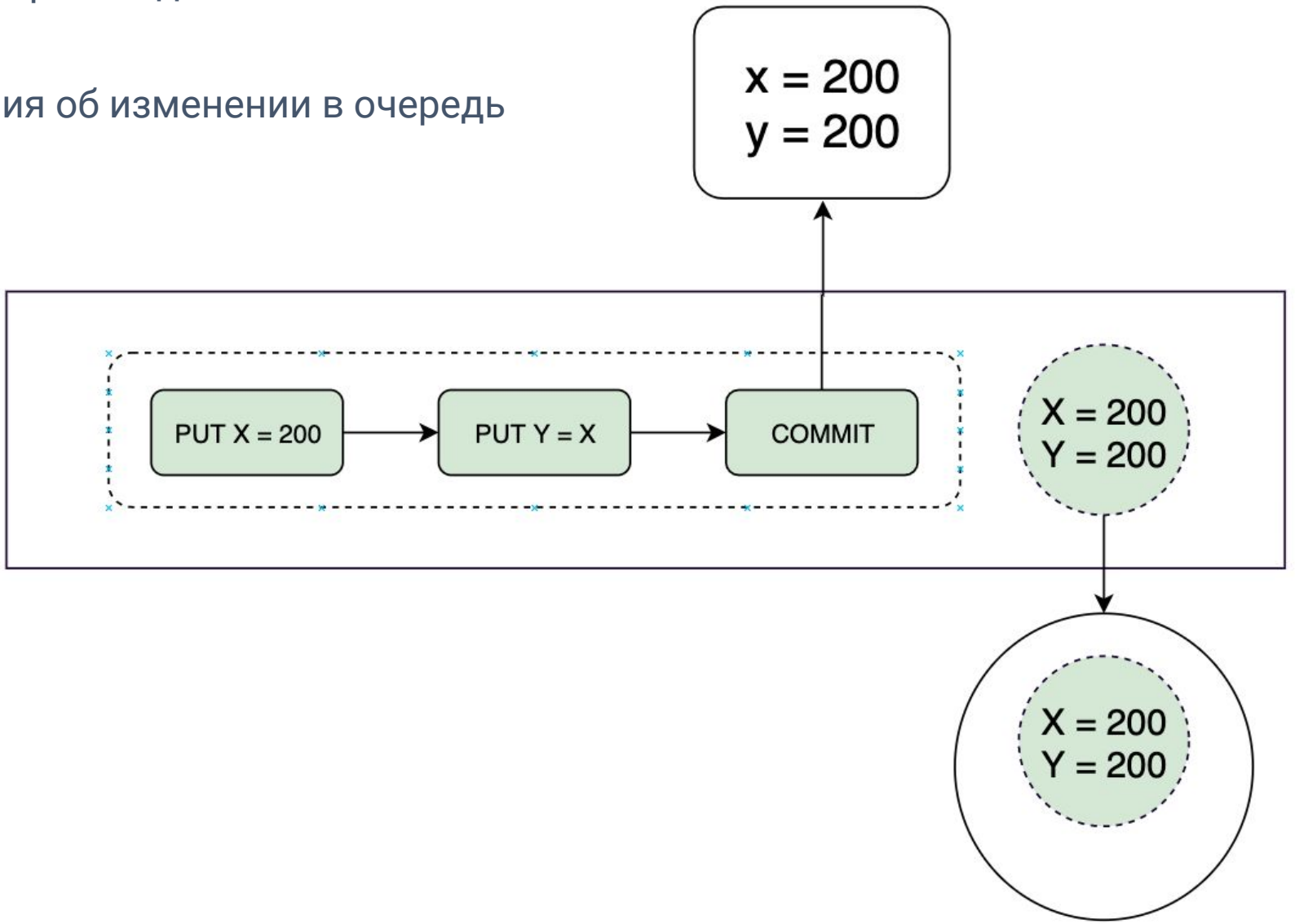
02

Надежная отправка сообщений

Кейс

Отправка сообщения происходит обычно в 2 этапа:

- Запись в БД
- Отправка сообщения об изменении в очередь



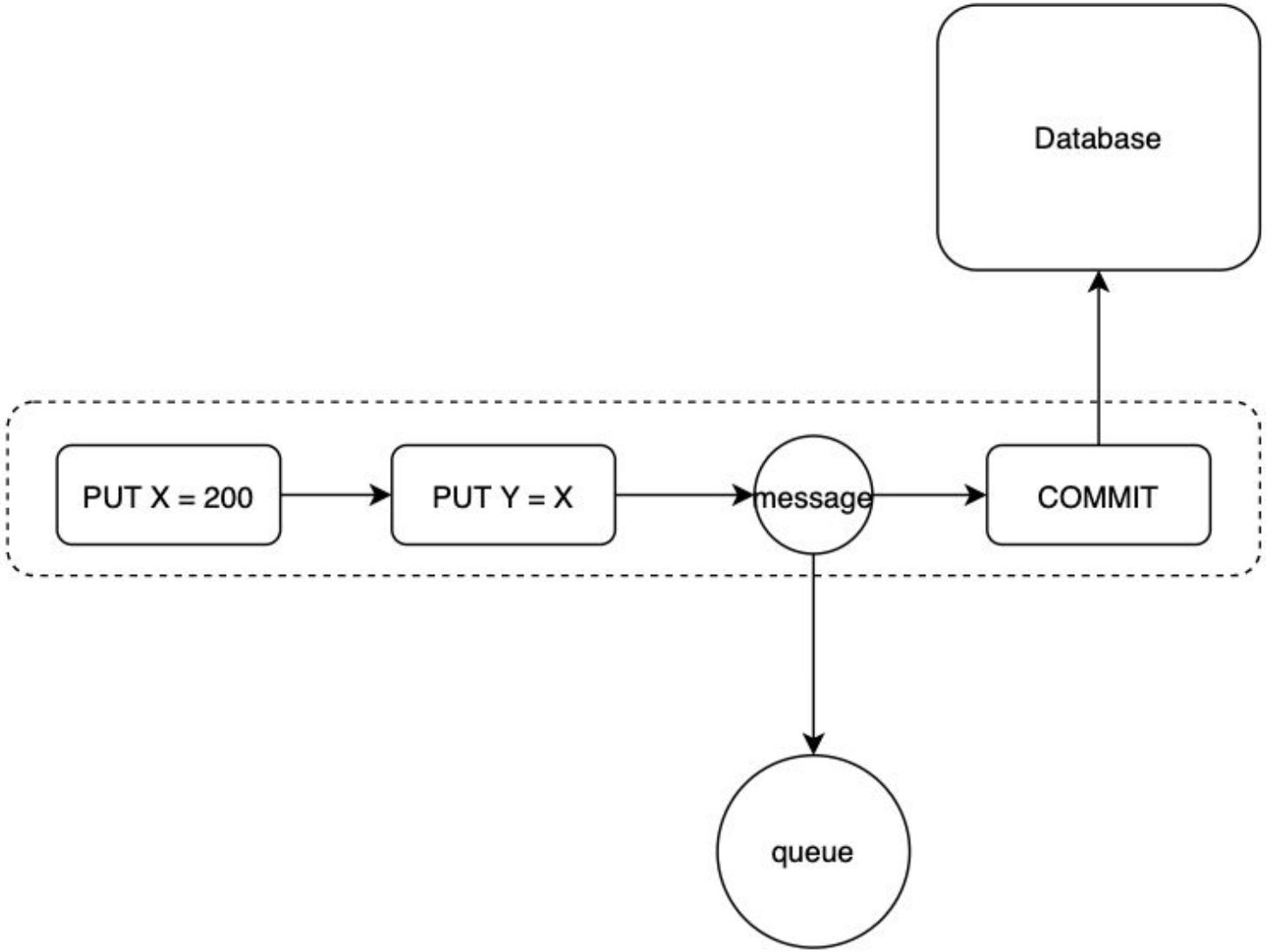
Отправка сообщений

Поскольку отправка сообщения и запись в базу не являются транзакцией, то возможно различные проблемы связанные с этим.

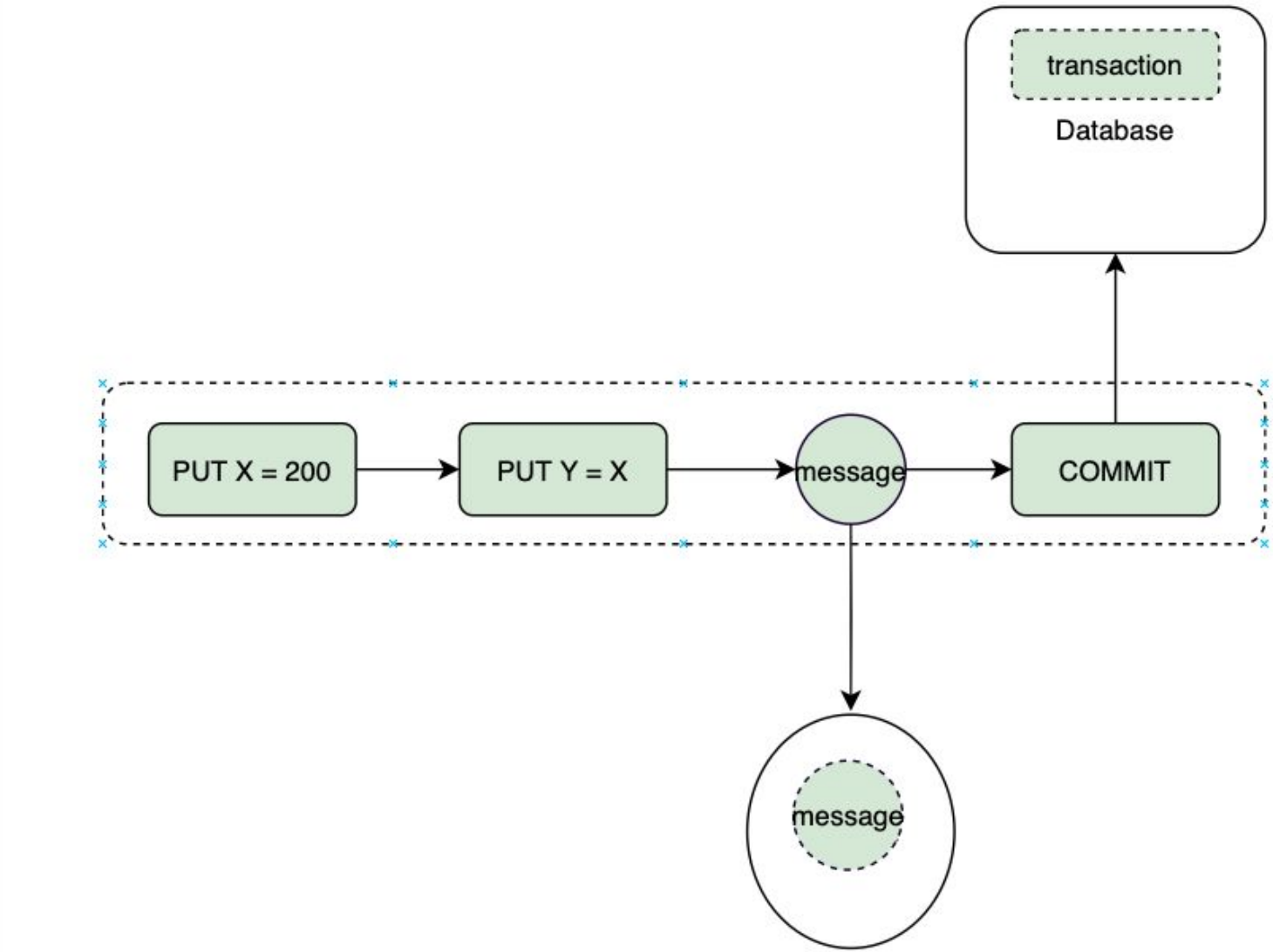
Чаще всего используется 2 паттерна отправки сообщений

- Отправка сообщения до коммита в базу
- Отправка сообщения после коммита в базу

Отправка сообщений до коммита



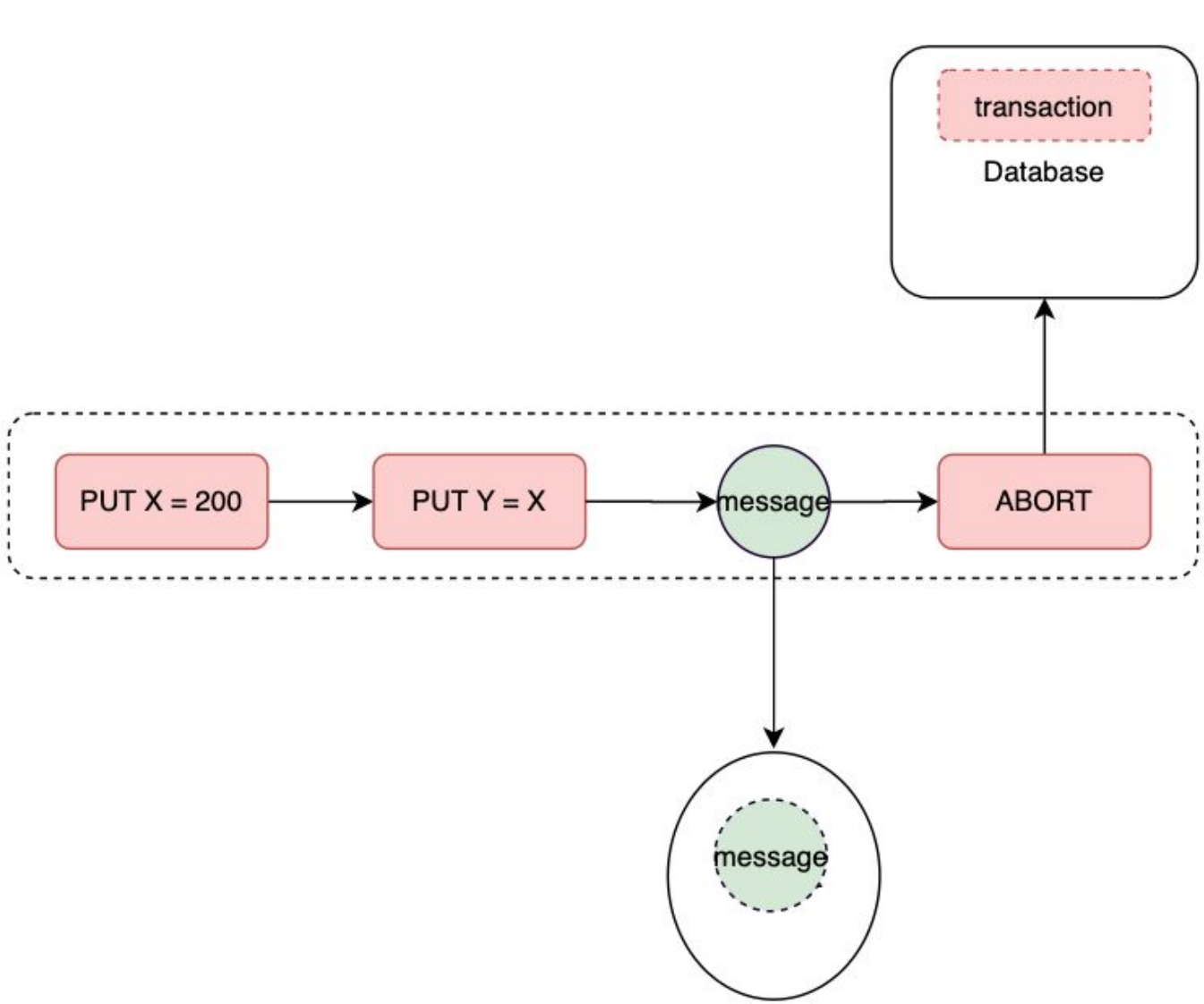
Отправка сообщений до коммита



Проблемы отправки сообщения до COMMITa

Даже в случае позитивного сценария такое решение может приводить к race –condition, в том случае, когда сообщение успевают обработать другие сервисы БЫСТРЕЕ, чем сервис сделал COMMIT. Тогда другие сервисы могут встречаться с ошибками в API при попытке обработать сообщение.

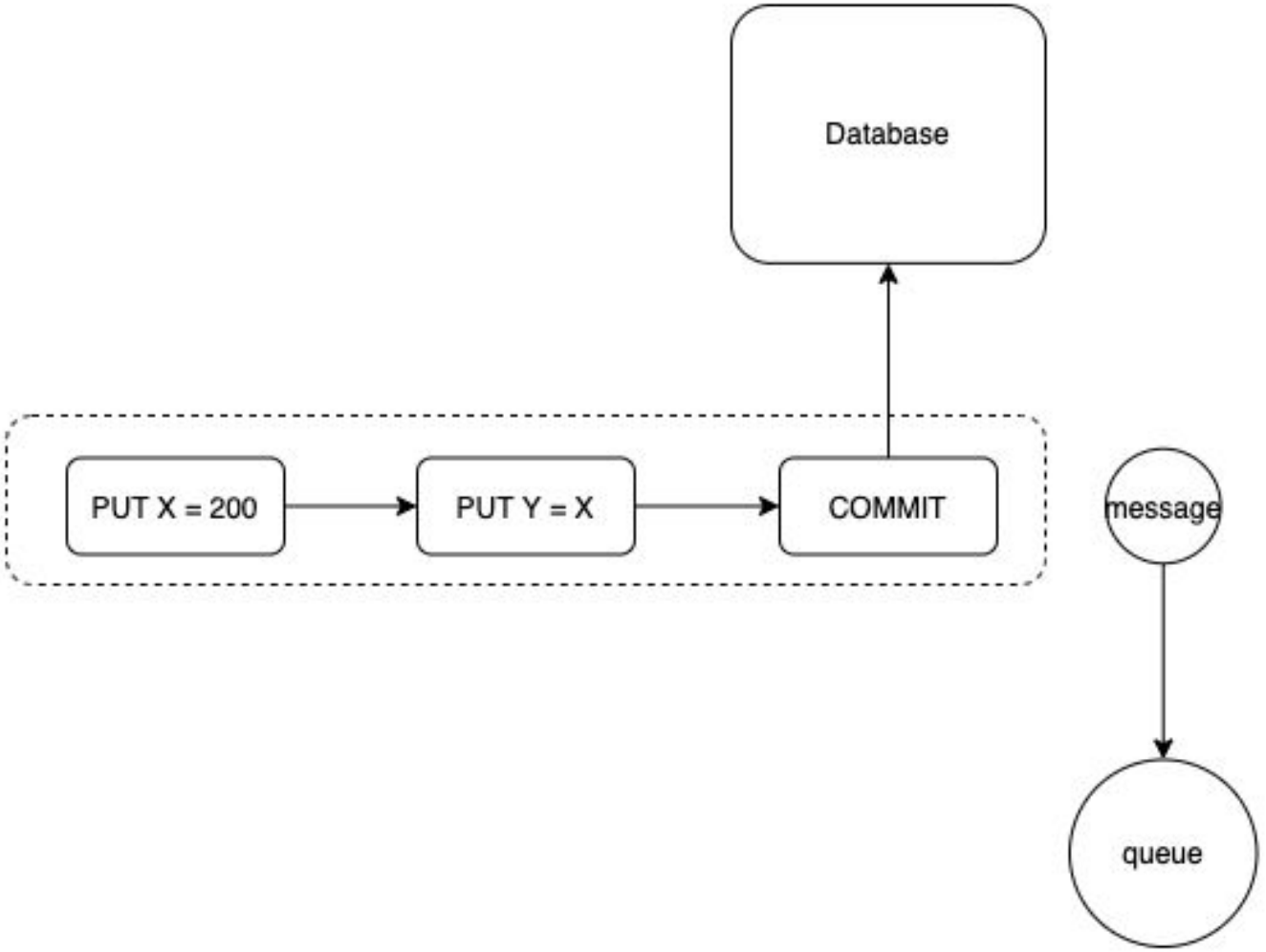
Отправка сообщений до коммита



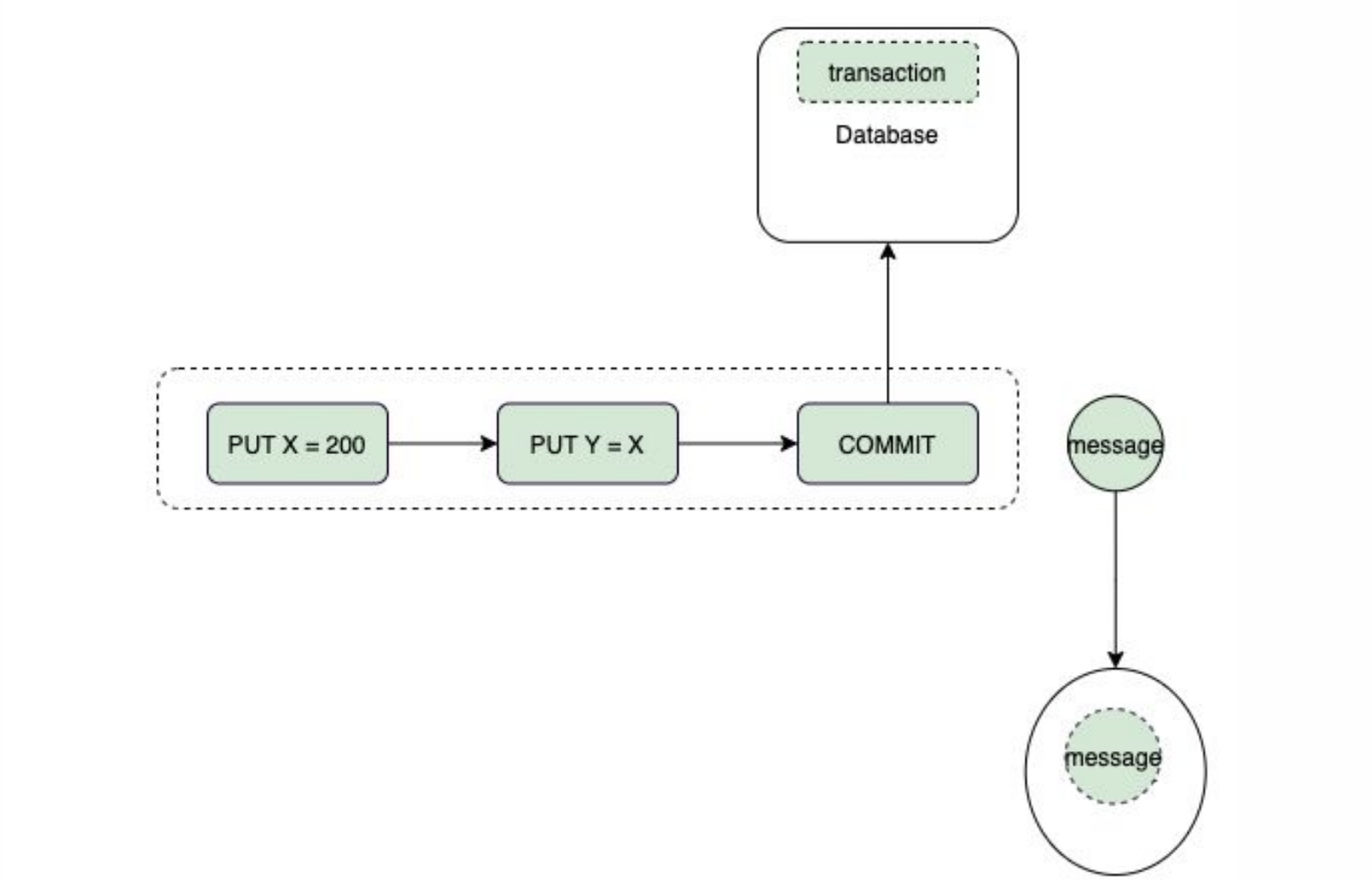
Проблемы отправки сообщения до COMMITa

В случае негативного сценария мы опять-таки отправляем событие, которого не случилось, что в некоторых ситуациях может приводить к неконсистентному состоянию в других сервисах.

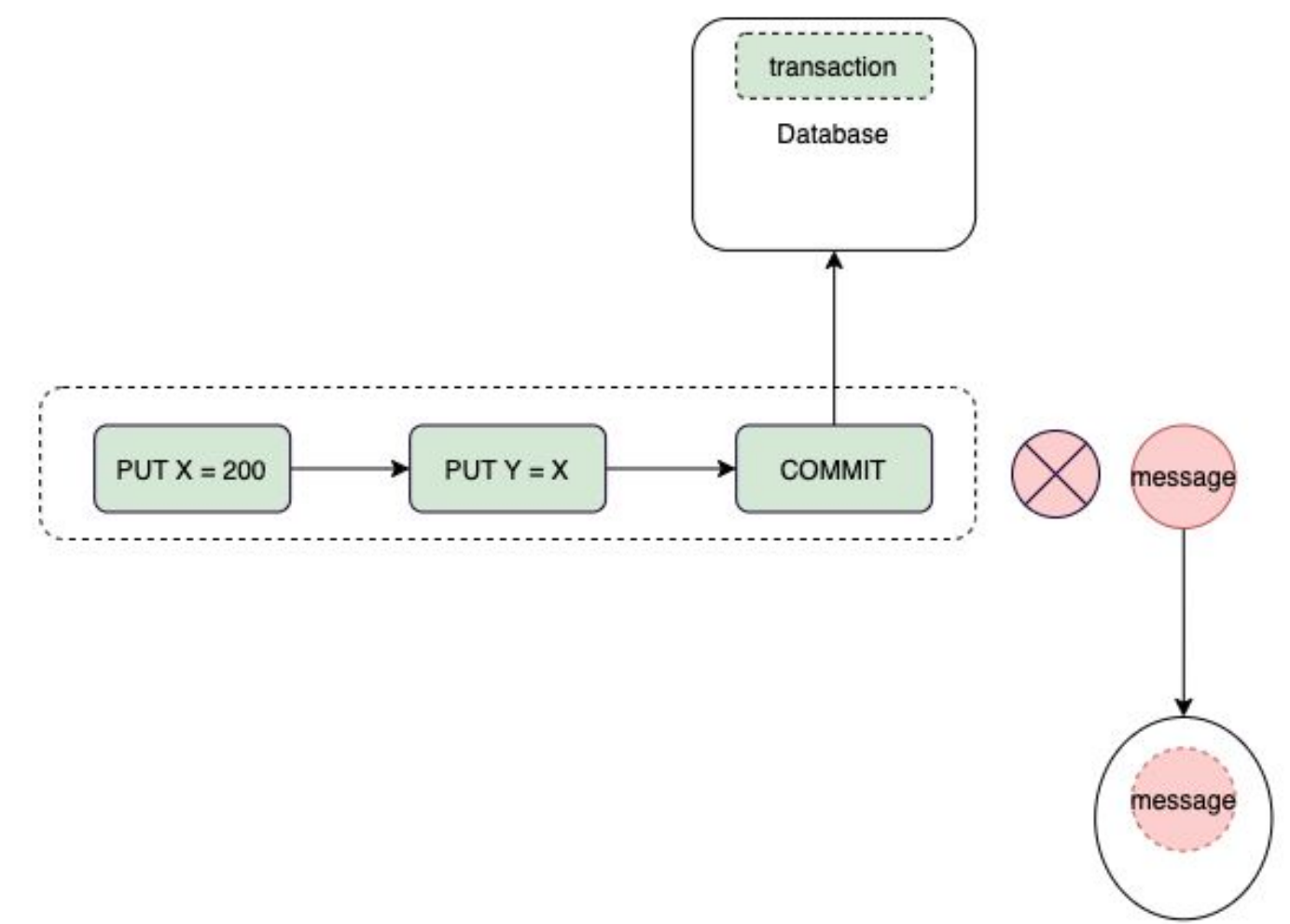
Отправка сообщений после коммита



Отправка сообщений после коммита



Отправка сообщений после коммита



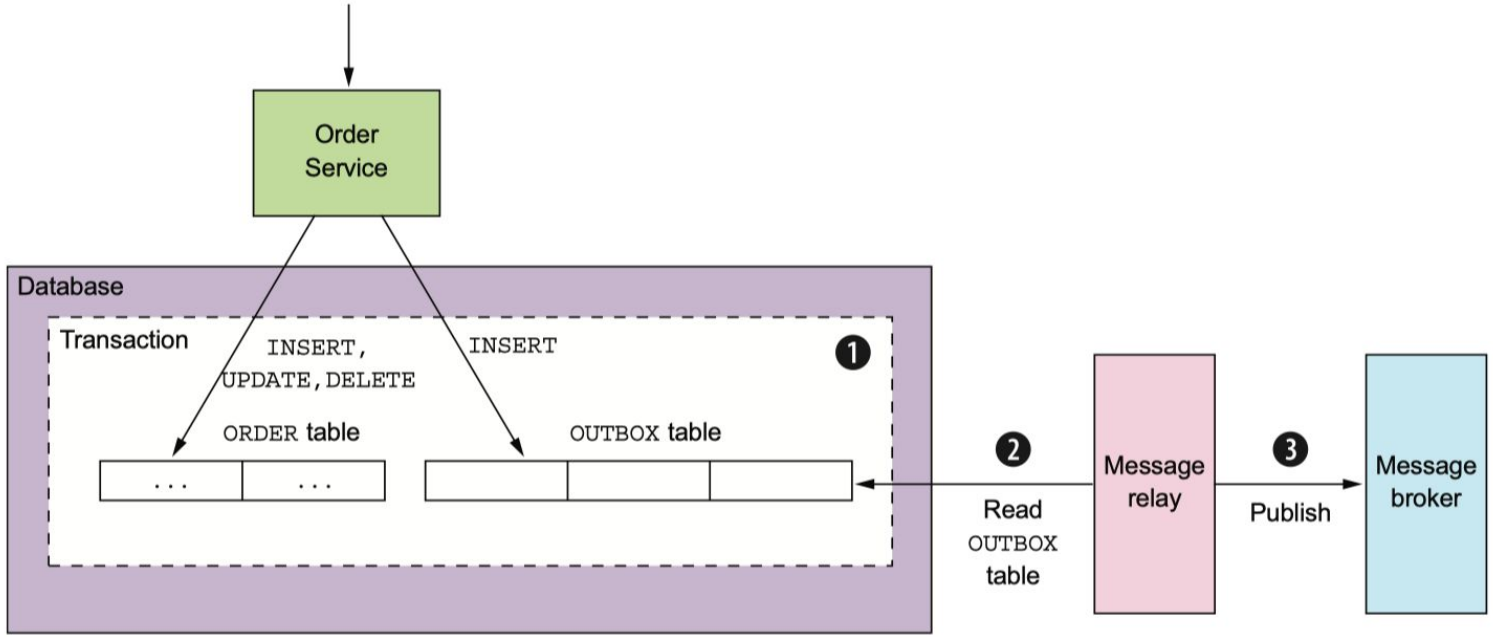
Проблемы отправки сообщения после COMMITa

В негативном сценарии сообщение может быть не отправлено.

Transactional Outbox

Если мы очень хотим иметь надежную доставку, то можно использовать паттерн transactional outbox

- Завести табличку с событиями (outbox table).
- Коммиты в табличку с данными и outbox table происходят транзакционно (1)
- Специальный сервис читает outbox table (2)
- Публикует сообщения в брокер сообщений (3)



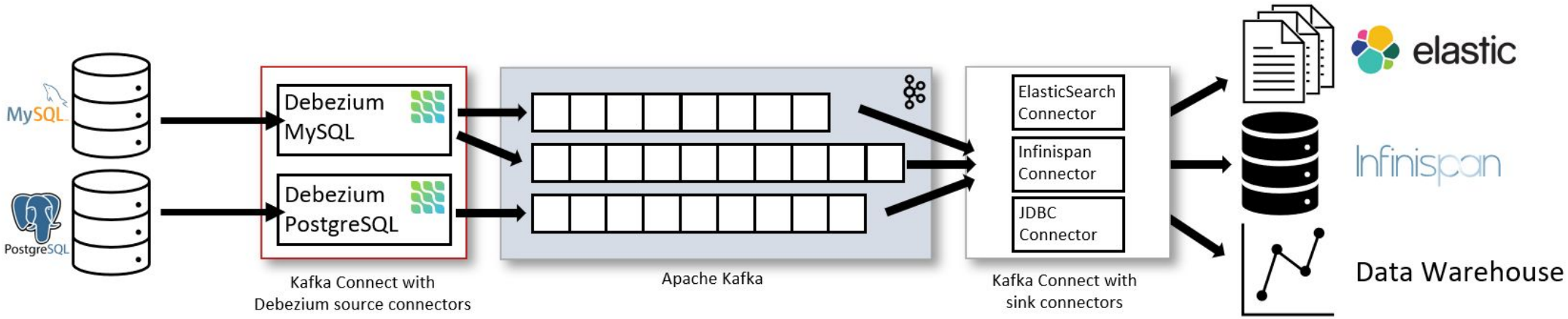
Transaction Log Miner

Такая схема используется в Rambler - https://www.youtube.com/watch?v=oByOmhOmOh4

Change Data Capture

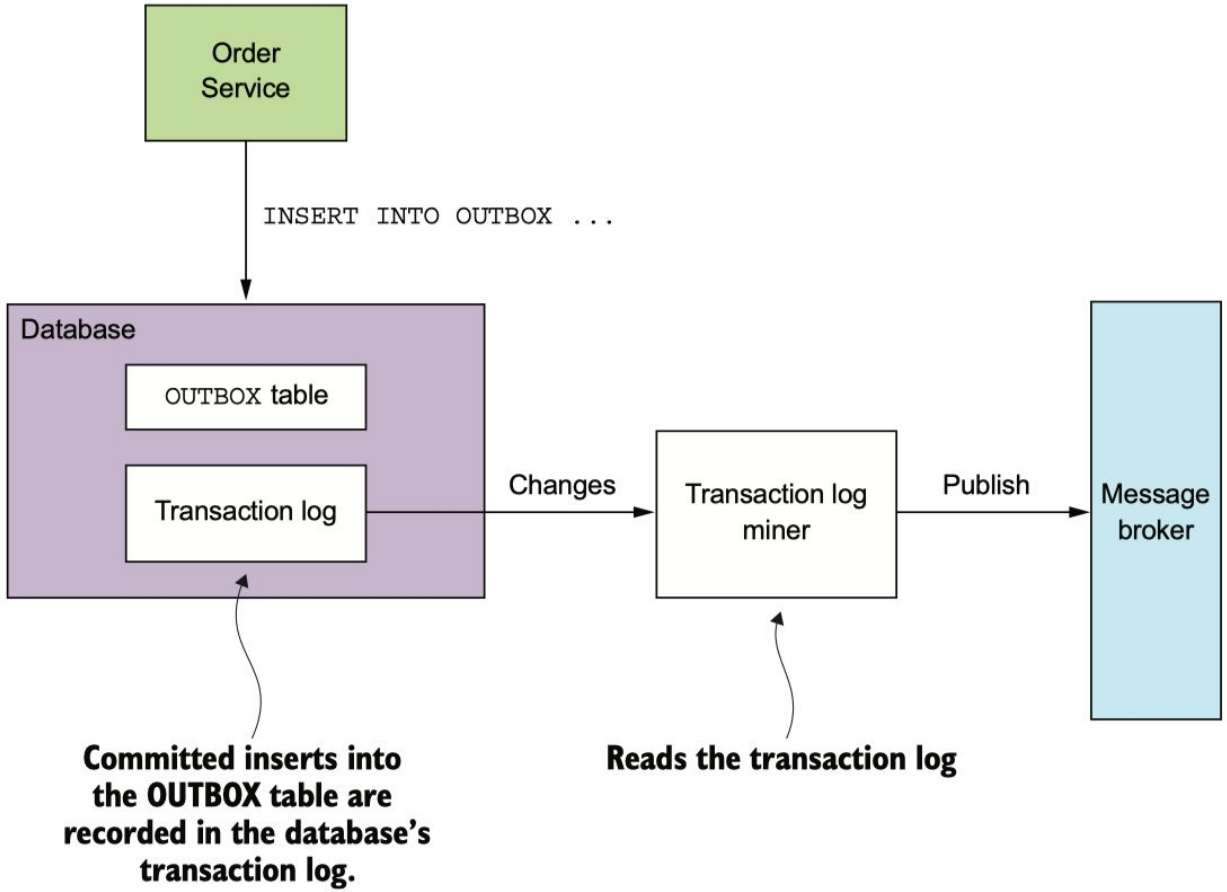
А почему бы не читать изменения из БД, а не из самого приложения?

Давайте читать данные из (WAL) лога БД и транслировать их в очередь, и оттуда читать.



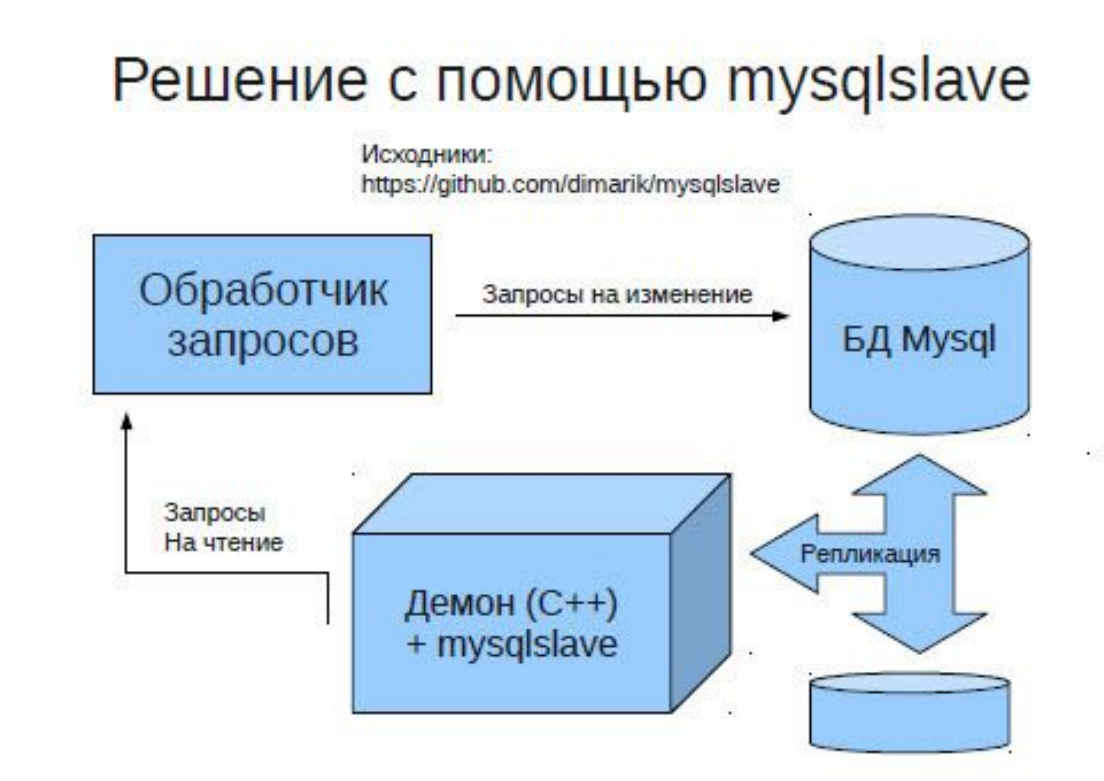
Transaction Log Miner

- Читаем WAL-лог базы данных или прикидываемся репликой
- Можем читать outbox table, можем читать сразу данные

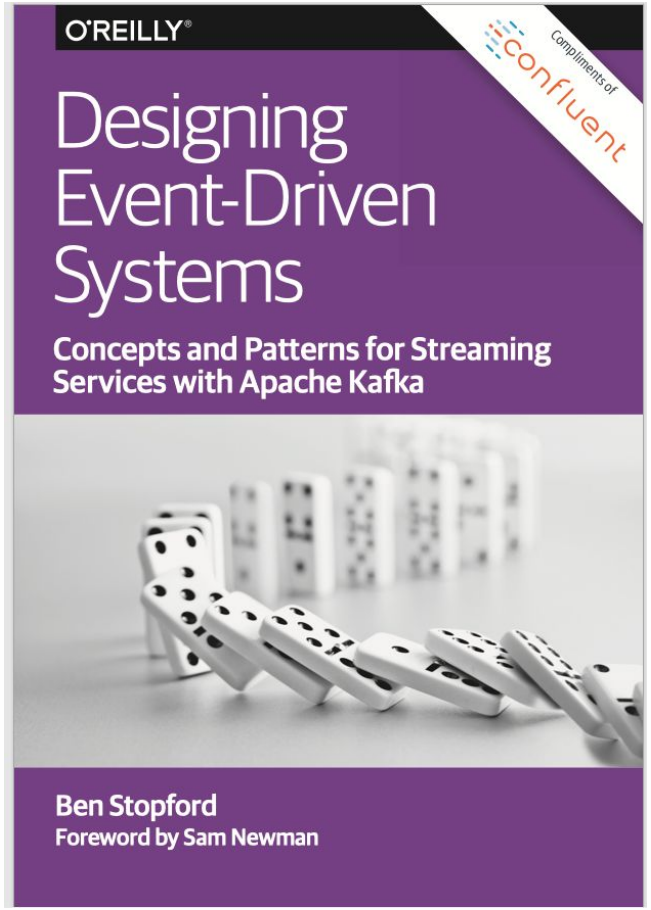


Transaction Log Miner

Такая схема используется в Mail.Ru (mytarget) - https://habr.com/ru/company/mailru/blog/219015/

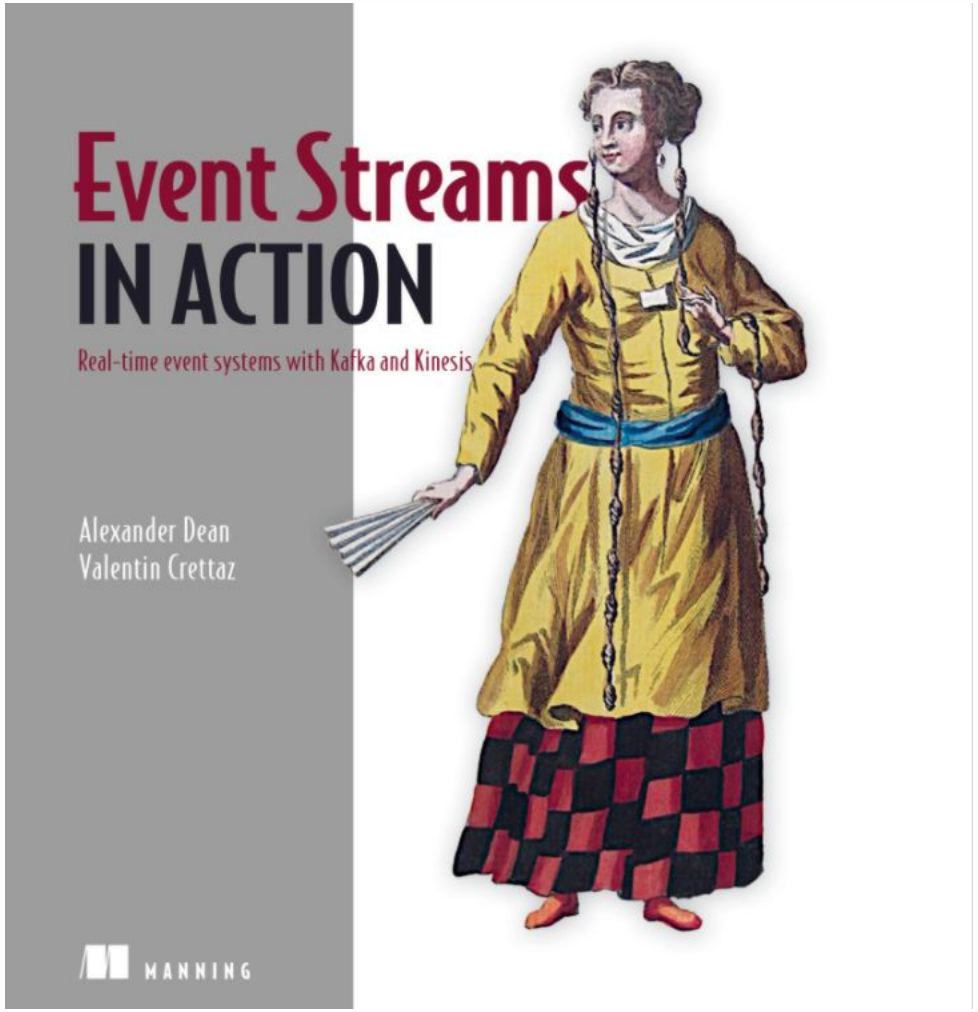


EDD и Stream Processing



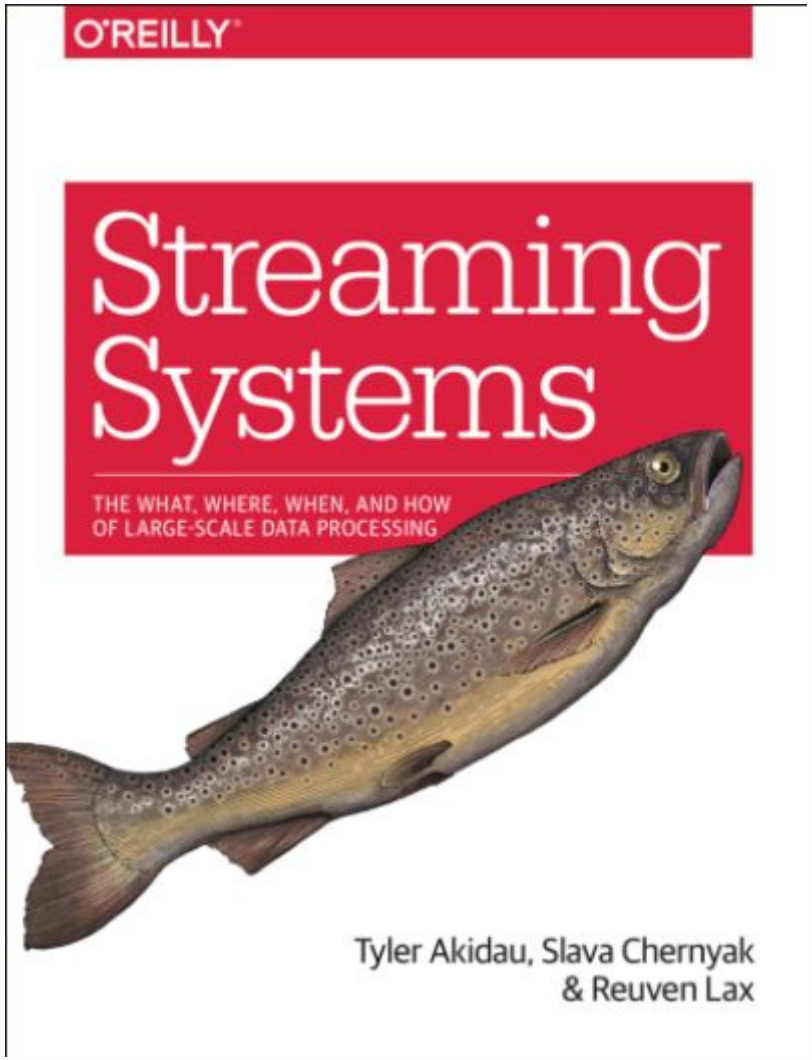
https://www.confluent.io/designing-event-driven-systems/

EDD и Stream Processing



https://livebook.manning.com/book/event-streams-in-action/about-this-book/0

EDD и Stream Processing



http://streamingsystems.net/

Домашнее задание

Опрос

https://otus.ru/polls/31956/

Спасибо
за внимание!