



ARC-015

Микросервисы

Организация хранения данных

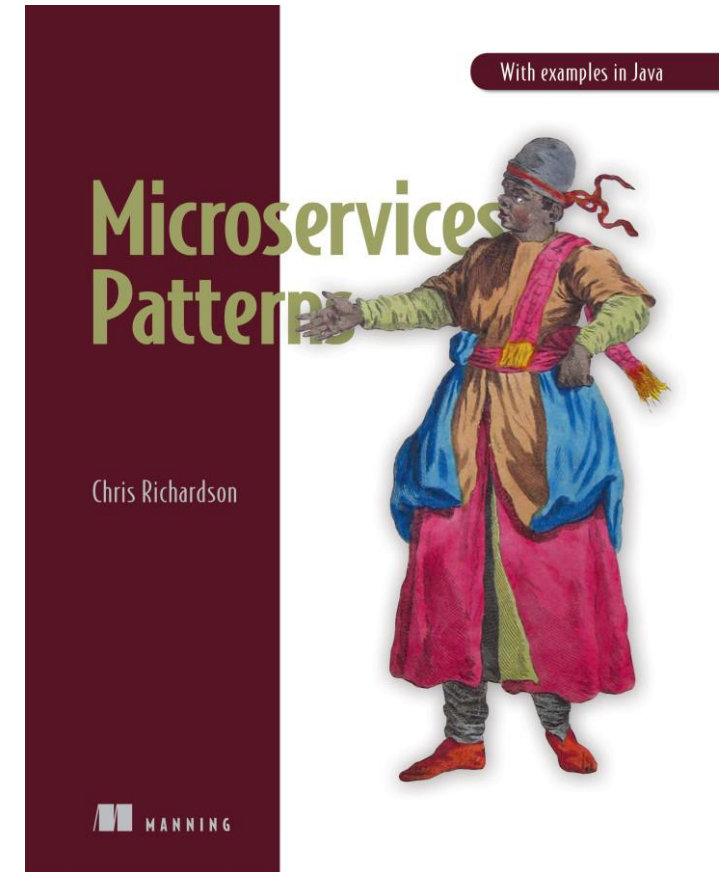
Владислав Родин

luxoft
A DXC Technology Company

Паттерны организации работы с данными

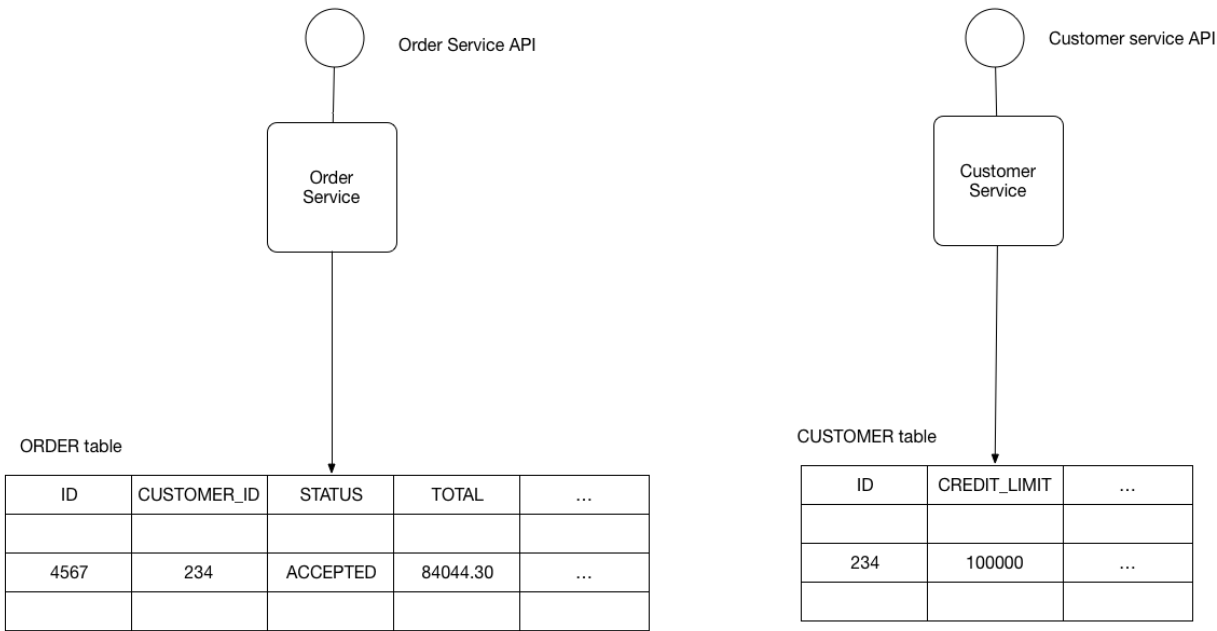
Крис Ричардсон

- **Общая база данных**
(Shared database)
- **Только один сервис на базу данных**
(Database per Service)
- **Порождение событий**
(Event Sourcing)



Общая база данных (Shared database)

Используем общую базу данных на несколько сервисов

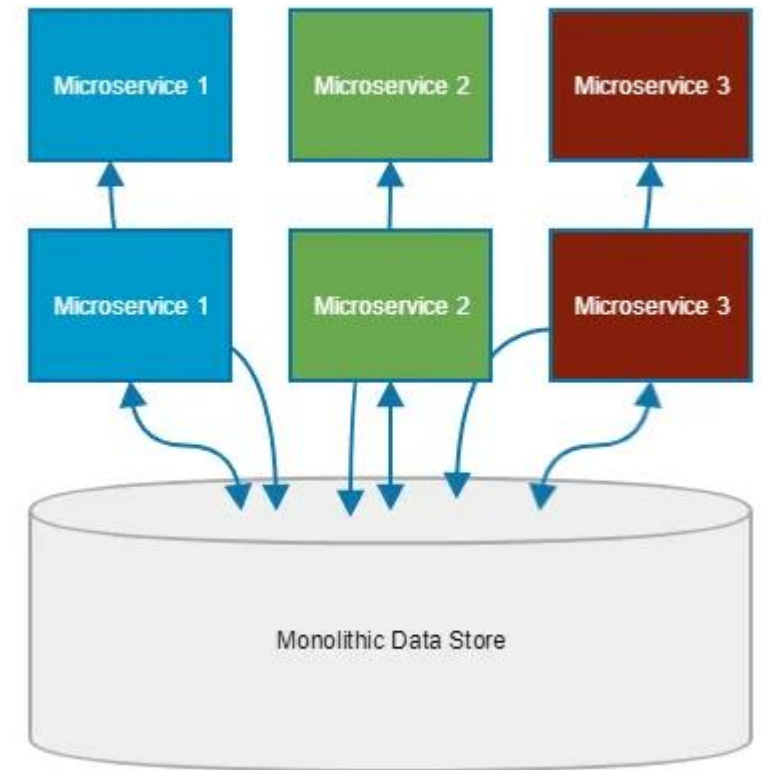


Общая база данных: Результат

- **Преимущества:**
 - Разработчик использует знакомые и простые транзакции ACID для обеспечения согласованности данных.
 - С одной базой данных проще работать.
 - Производительность (уменьшается число сетевых запросов).
- **Недостатки:**
 - Связывание времени разработки – разработчику, необходимо согласовывать изменения схемы с разработчиками других сервисов, которые обращаются к тем же таблицам.
 - Связывание во время выполнения – поскольку все службы обращаются к одной и той же базе данных, они могут потенциально создавать помехи друг другу.
 - Единая база данных может не соответствовать требованиям хранения данных и доступа различных служб.

Анти-паттерн Everything Micro (Except for the Data)

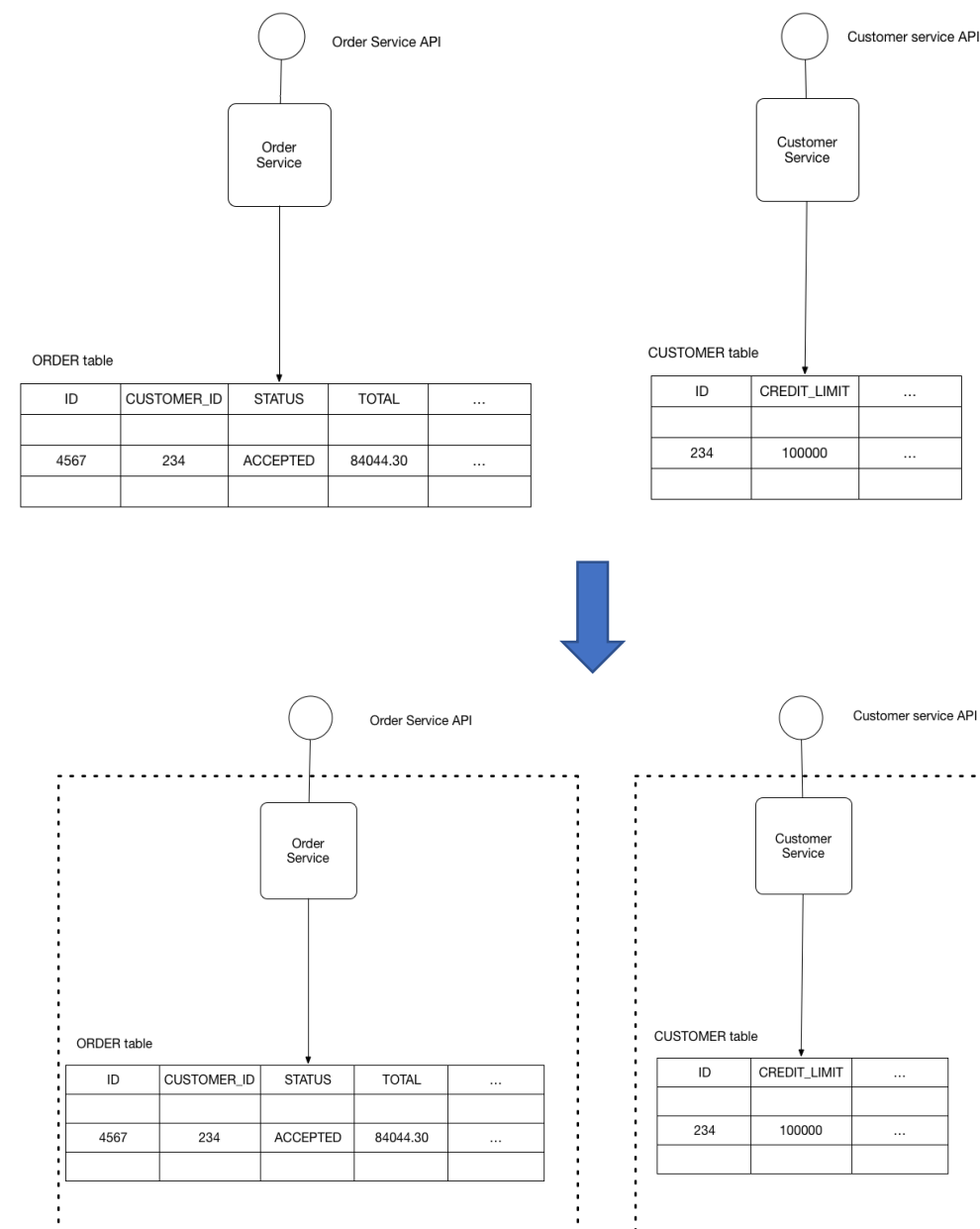
- Наиболее распространено при использовании мощных серверов Microsoft SQL Server, Oracle и DB2, главным образом потому, что их модели лицензирования не позволяют легко реализовать базу данных для каждой службы.



Только один сервис на базу данных (Database per Service)

Варианты

- ***Private-tables-per-service*** – каждому сервису принадлежит набор таблиц, к которым должен обращаться только этот сервис
- ***Schema-per-service*** - каждый сервис имеет свою схему данных
- ***Database-server-per-service*** – каждый сервис имеет свой собственный сервер базы данных



Только один сервис на базу данных: Результат

- **Преимущества:**

- Обеспечивает слабую связанность сервисов.

Изменения в базе данных одного сервиса не влияют на другие сервисы.

- Каждый сервис может использовать тот тип базы данных, который лучше всего соответствует его потребностям. Например, сервис, выполняющий текстовый поиск, может использовать Elasticsearch. Сервис, который манипулирует социальным графом, может использовать Neo4j.

- **Недостатки:**

- Реализация бизнес-транзакций, охватывающих несколько сервисов, не проста (теорема CAP).
- Реализация запросов, объединяющих данные, которые находятся в нескольких базах данных, является сложной задачей.
- Сложность управления несколькими базами данных SQL и NoSQL.

Общая база с одним владельцем

- При решении с общей базой данных выделяется владелец для каждого элемента (всей базы данных, схемы, определенного набора таблиц)
- Устанавливаются следующие ограничения:
 - Владелец данных имеет право читать и записывать данные
 - Владелец данных отвечает за миграцию данных
 - Остальные сервисы могут только читать данные
- Это решение позволяет использовать базу данных в качестве инструмента взаимодействия
 - Требуется организация взаимодействия между командами разработчиков по установлению контрактов

Порождение событий (Event Sourcing)

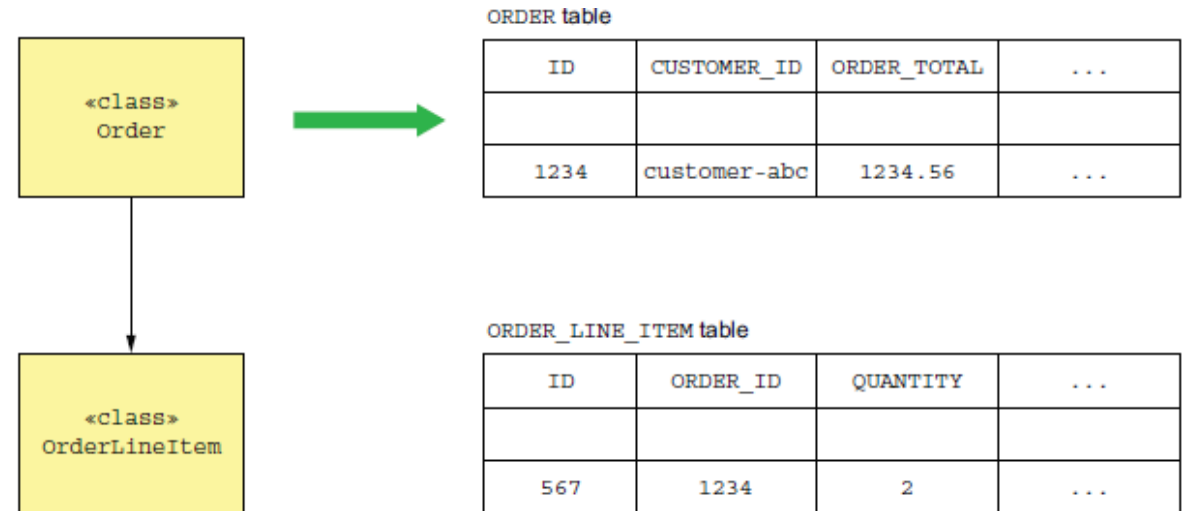
Порождение событий (Event Sourcing)

Способ структурирования бизнес-логики и сохранения агрегатов.

- **Агрегаты сохраняются в виде последовательности событий, каждое из которых представляет изменение состояния агрегата.**
- **Приложение воссоздает текущее состояние, воспроизводя записанные события (регидратация, rehydration).**

Проблемы традиционного сохранения данных

- Объектно-реляционный разрыв.
- Отсутствие истории агрегатов.
- Реализация журналирования для аудита требует много усилий и чревата ошибками.
- Публикация событий является лишь дополнением к бизнес-логике.



Порождение событий: Результаты

- **Преимущества:**
 - Позволяет реализовать запросы, которые определяют состояние объекта в любой момент времени.
 - Выполняет надежную публикацию доменных событий.
 - Решает проблему ORM.
 - Обеспечивает 100% надежный журнал аудита изменений.
- **Недостатки:**
 - Сложен в изучении, поскольку это совершенно другой способ написания бизнес-логики.
 - Обращение к хранилищу событий часто затруднительно и требует использования шаблона CQRS.
 - Приходится ограничиваться итоговой согласованностью.

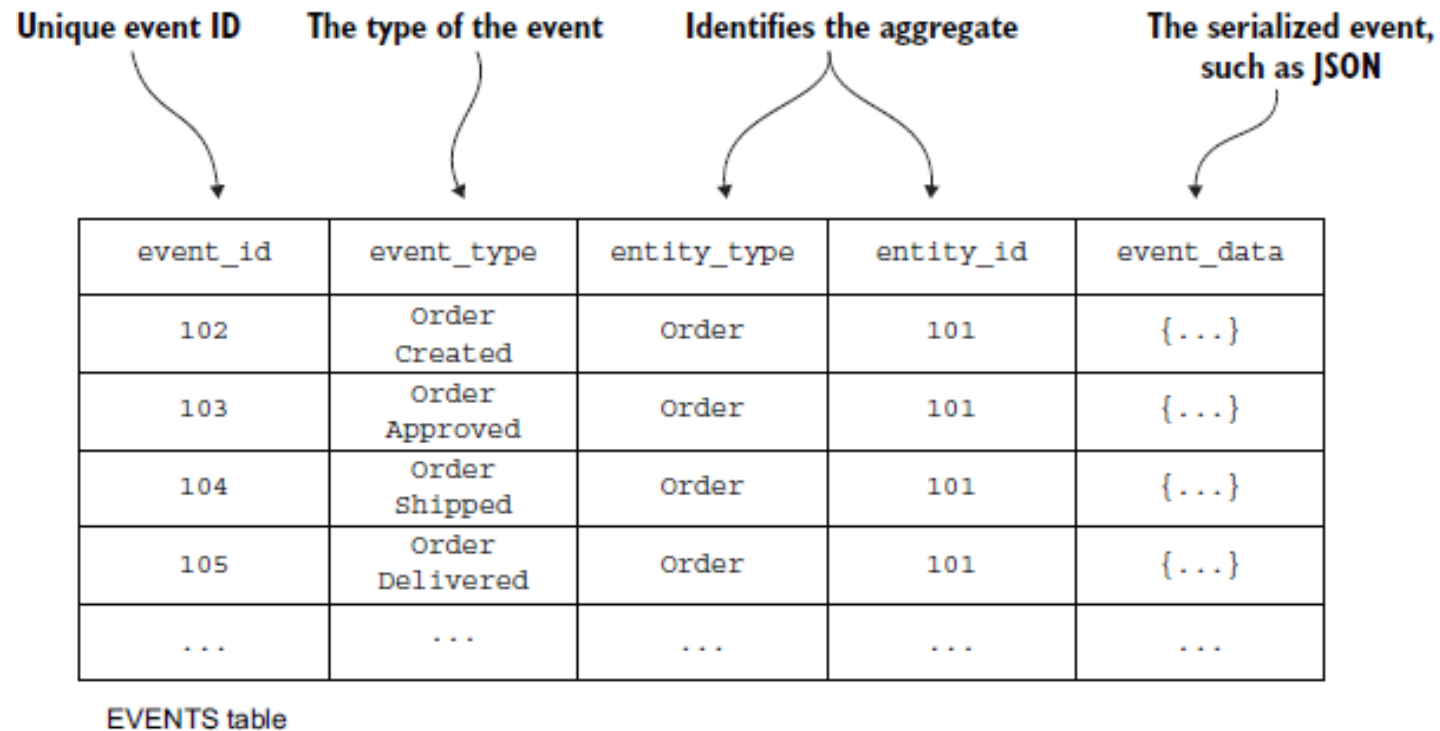
Объектно-реляционное отображение — Вьетнамская война в мире информатики

Тед Ньюард (Ted Neward)



Порождение событий: Запись

- Оно сохраняет каждый агрегат в базе данных, так называемом хранилище событий в виде последовательности событий.



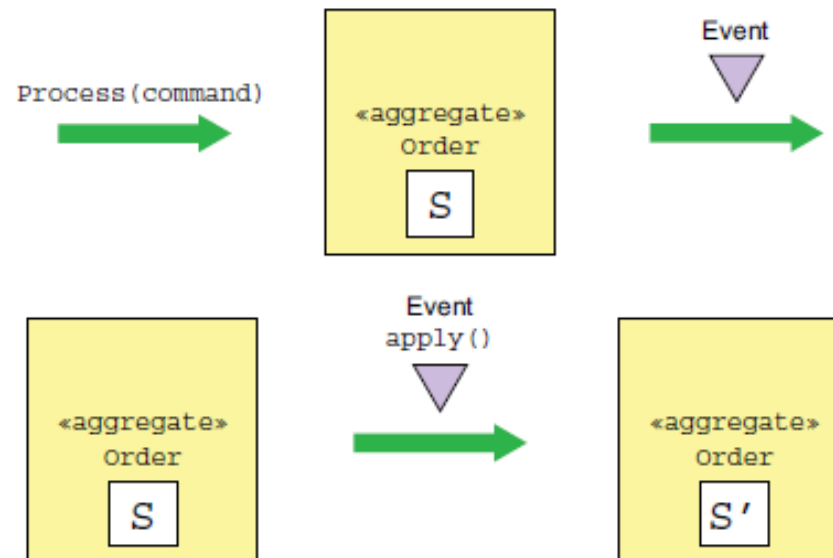
Порождение событий: Чтение (регидратация, rehydration)

Шаги:

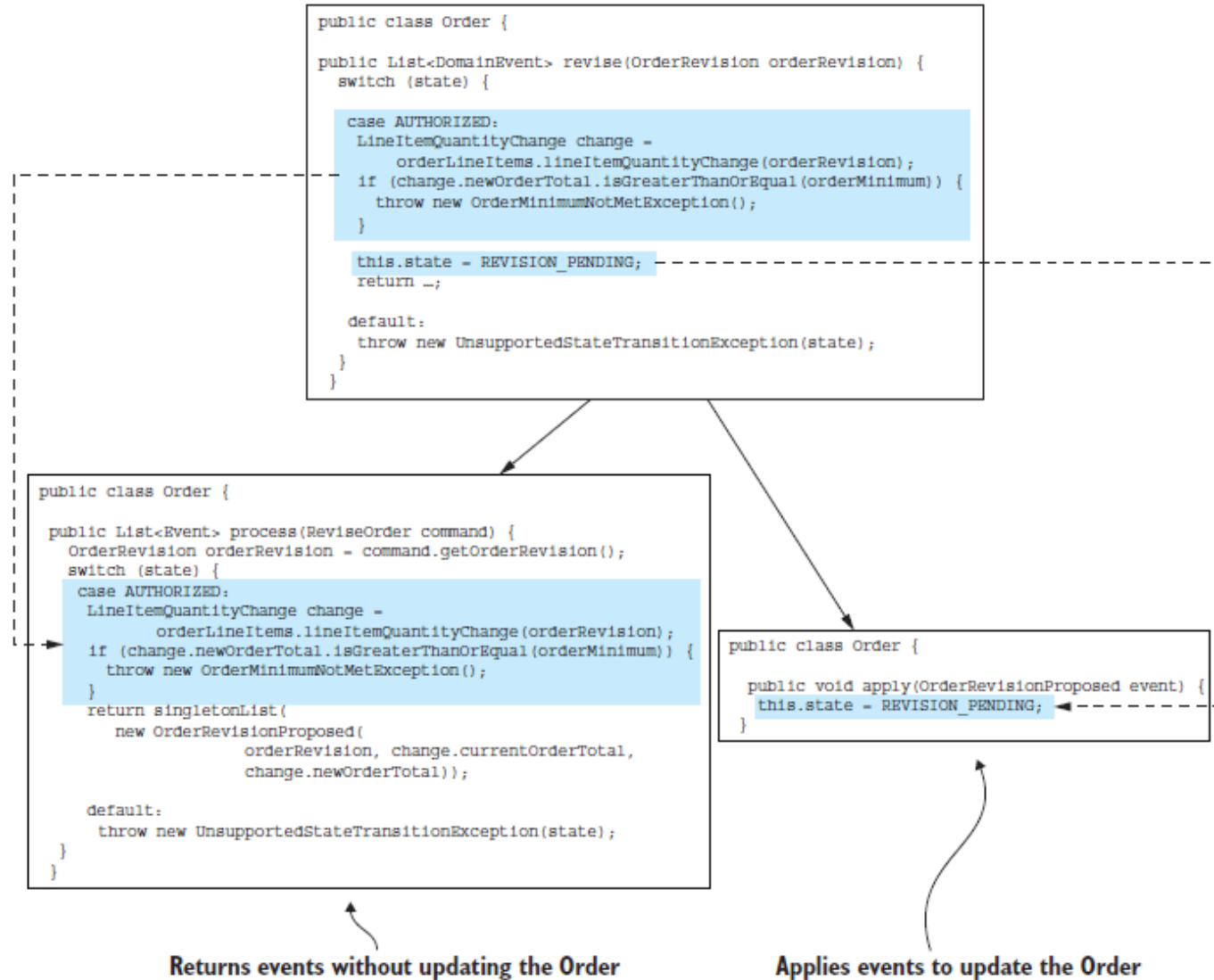
1. Загрузка событий агрегата.
2. Создание экземпляра агрегата с помощью конструктора по умолчанию.
3. Перебор событий с вызовом `apply()`.

Порождение событий: События

- События порождаются не только по потребности клиентов, но и на каждое изменение агрегата
- События содержат всю информацию, необходимую для восстановления агрегата



Порождение событий: Расщепление метода обработки команды



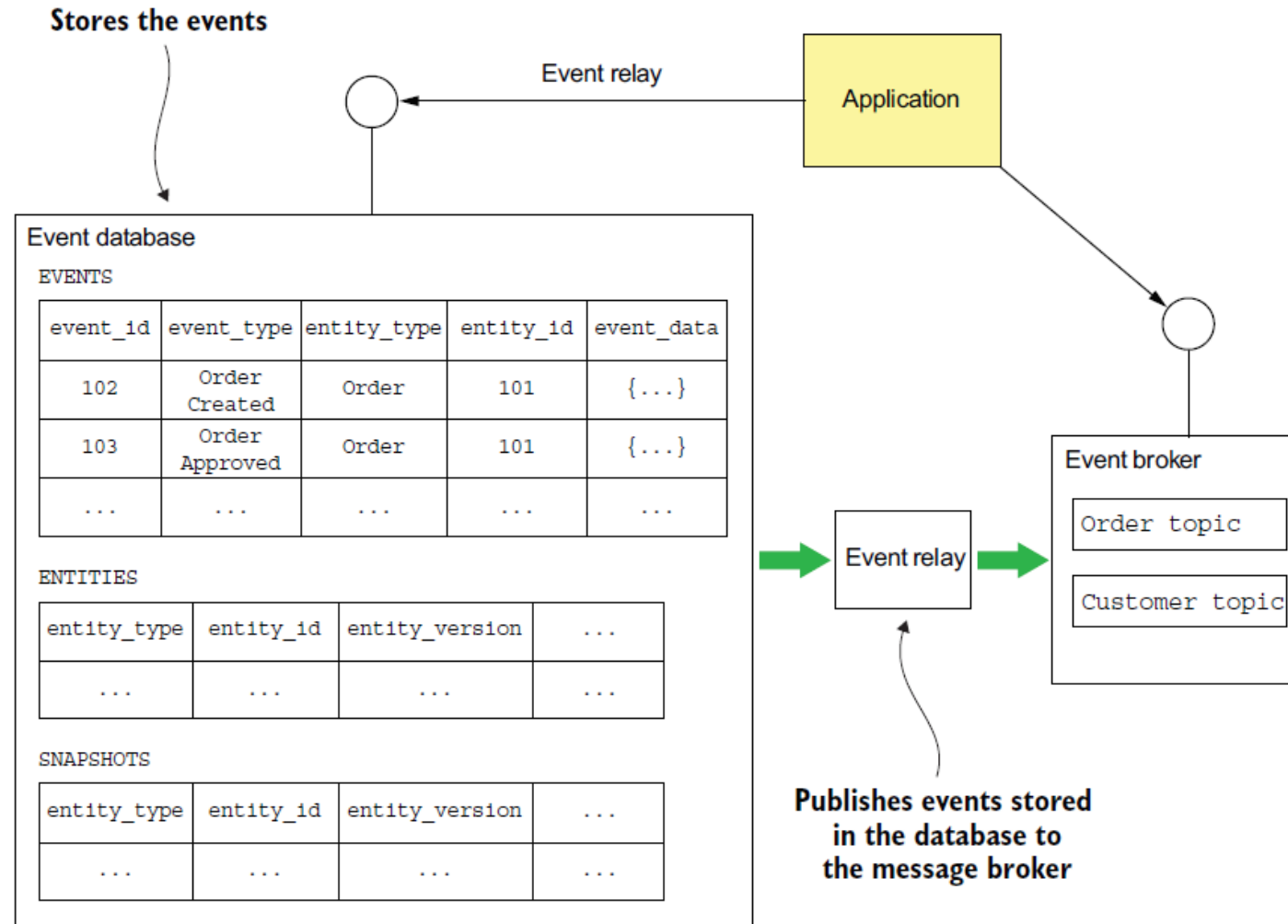
Конфликт при обновлении агрегата

- **Для разрешения конфликтов используется оптимистическая блокировка**
 - **Перед записью новых событий определяется версия агрегата (версия последнего события)**
 - **При наличии конфликта запись не осуществляется**

Реализация хранилища событий

- ***Event Store*** — хранилище событий с открытым исходным кодом на основе .NET от Грегга Янга (Greg Young), пионера в области порождения событий (<https://eventstore.org/>).
- ***Lagom*** — микросервисный фреймворк от компании Lightbend, ранее известной как Typesafe (www.lightbend.com/lagom-framework).
- ***Axon*** — фреймворк с открытым исходным кодом на языке Java для разработки событийных приложений, которые используют порождение событий и CQRS (www.axonframework.org).
- ***Eventuate*** — фреймворк, разработанный стартапом Eventuate (eventuate.io).
 - Eventuate SaaS — облачный сервис
 - Eventuate Local — открытый проект на основе Apache Kafka/СУРБД.

Реализация хранилища событий: Eventuate Local



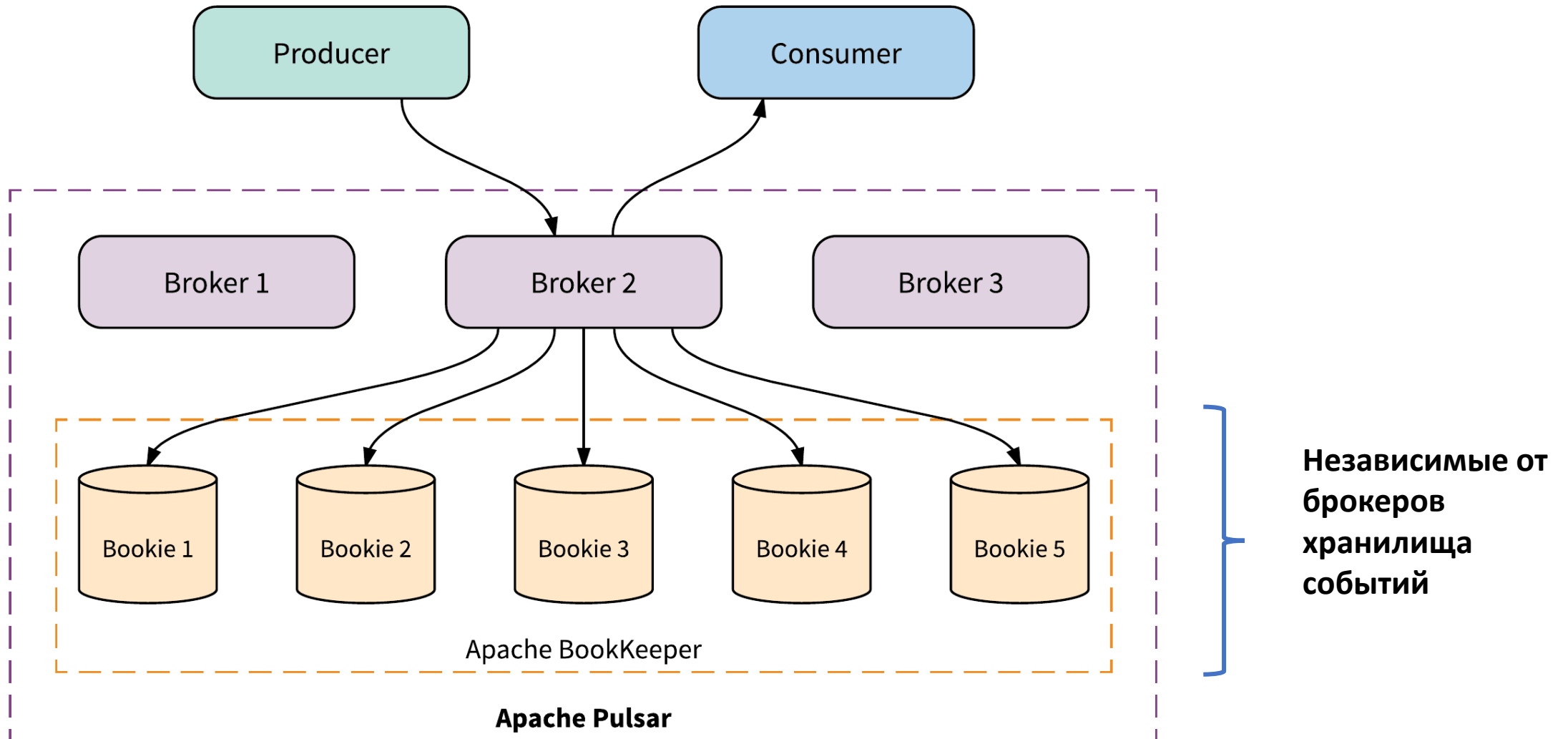
Журнал событий как хранилище

- В качестве хранилища событий возможно использовать журнал событий (например Apache Kafka)
 - Источник событий включает в себя поддержание неизменной последовательности событий, на которую могут подписаться несколько приложений.
 - Kafka - это высокопроизводительный, с малыми задержками, масштабируемый и долговечный журнал, который используется тысячами компаний по всему миру и испытан в боевых условиях.
 - Следовательно, Kafka является естественной основой для хранения событий при переходе к архитектуре приложений на основе источников событий.

Apache Kafka как хранилище событий: Недостатки

- Для получения данных по одному агрегату нужно вытянуть все события топика
 - Решение: один топик на экземпляр (требуется множество топиков)
- Невозможно восстановить систему «переиграв» все события, так как они находятся в разных топиках
 - Решение: один топик на все события (при небольшом количестве событий)
- Нет поддержки оптимистичной блокировки при записи событий агрегата
 - Решение: блокировка отдельный механизм
- Журналы событий (хранилища) жестко привязаны к брокерам очередей
Это создает проблемы при необходимости масштабировать хранилище или при выходе из строя одного из разделов
 - Решение: переход на журнал событий с другой архитектурой, например Apache Pulsar

Apache Pulsar

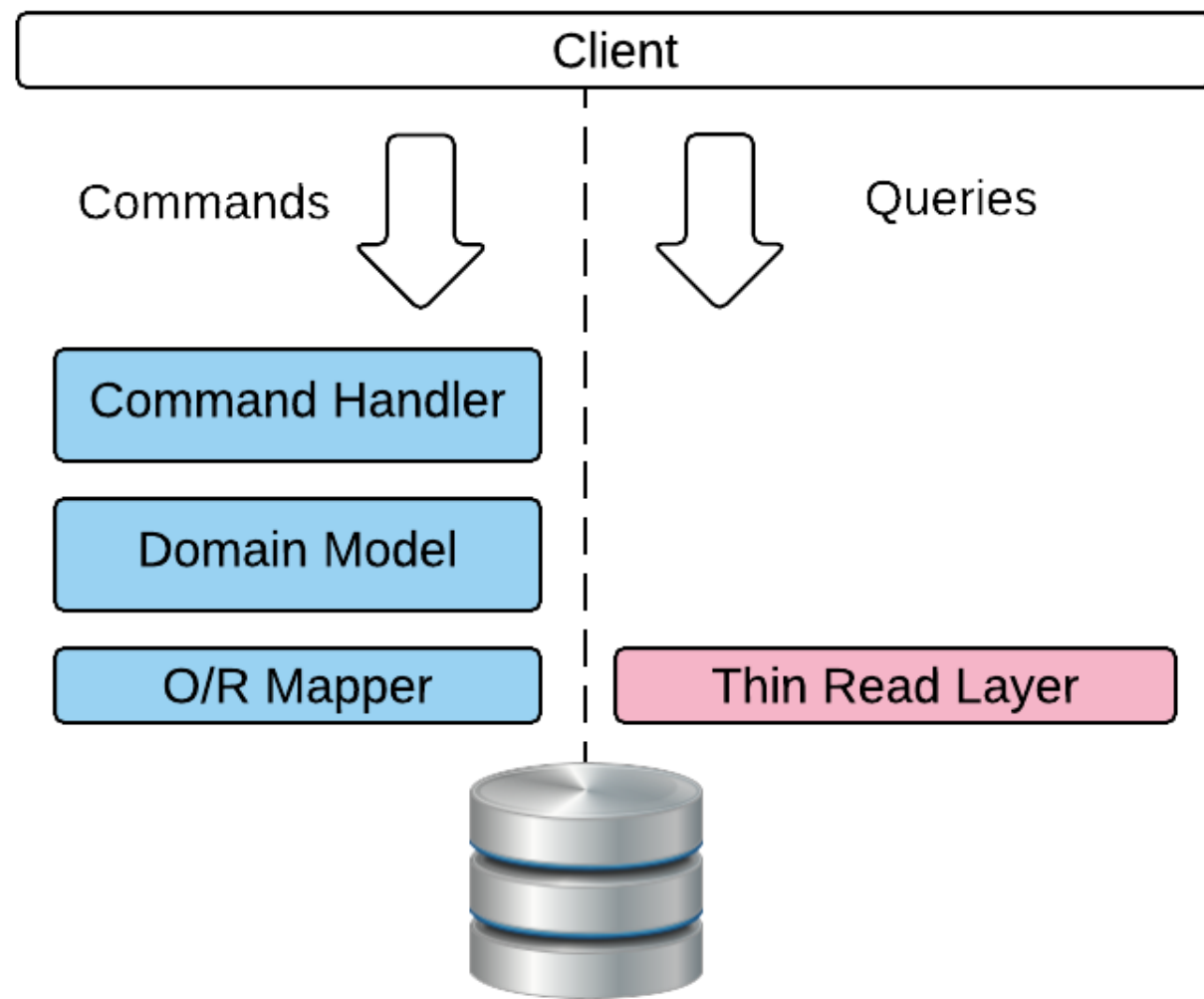


Журнал событий в качестве хранилища: Проблемы

- **Загрузка текущего состояния**
 - Для того, чтобы сформировать один агрегат, надо перебрать множество событий, выбирая из них связанные с нашим агрегатом
- **Последовательность записей**
 - Необходимо обеспечить оптимистическую блокировку при записи событий по одному агрегату
 - Можно держать блокировку отдельно в транзакционной базе
 - Можно обеспечить единственный поток записи данных одного агрегата

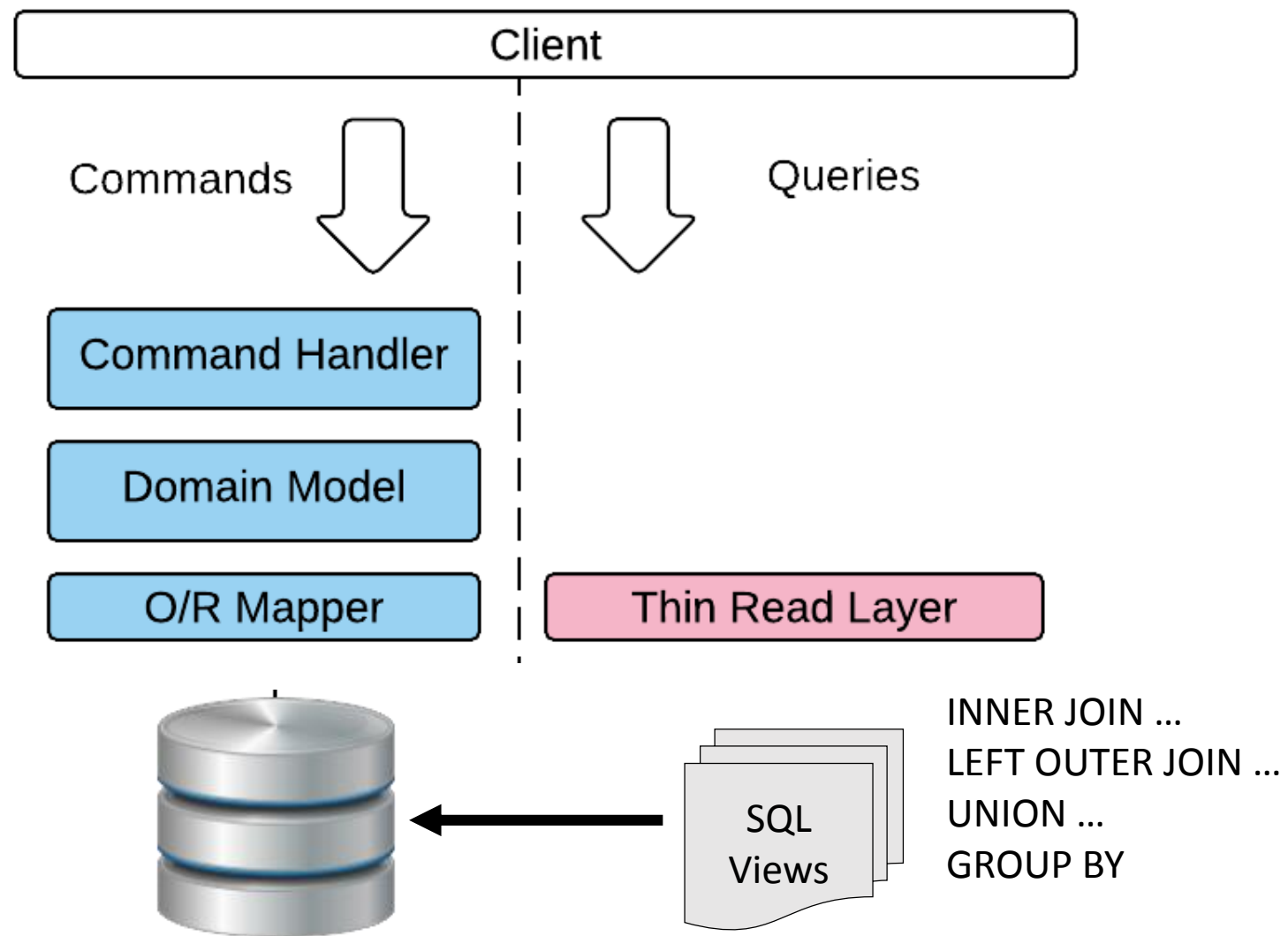
Event Sourcing + CQRS

Шаг 1



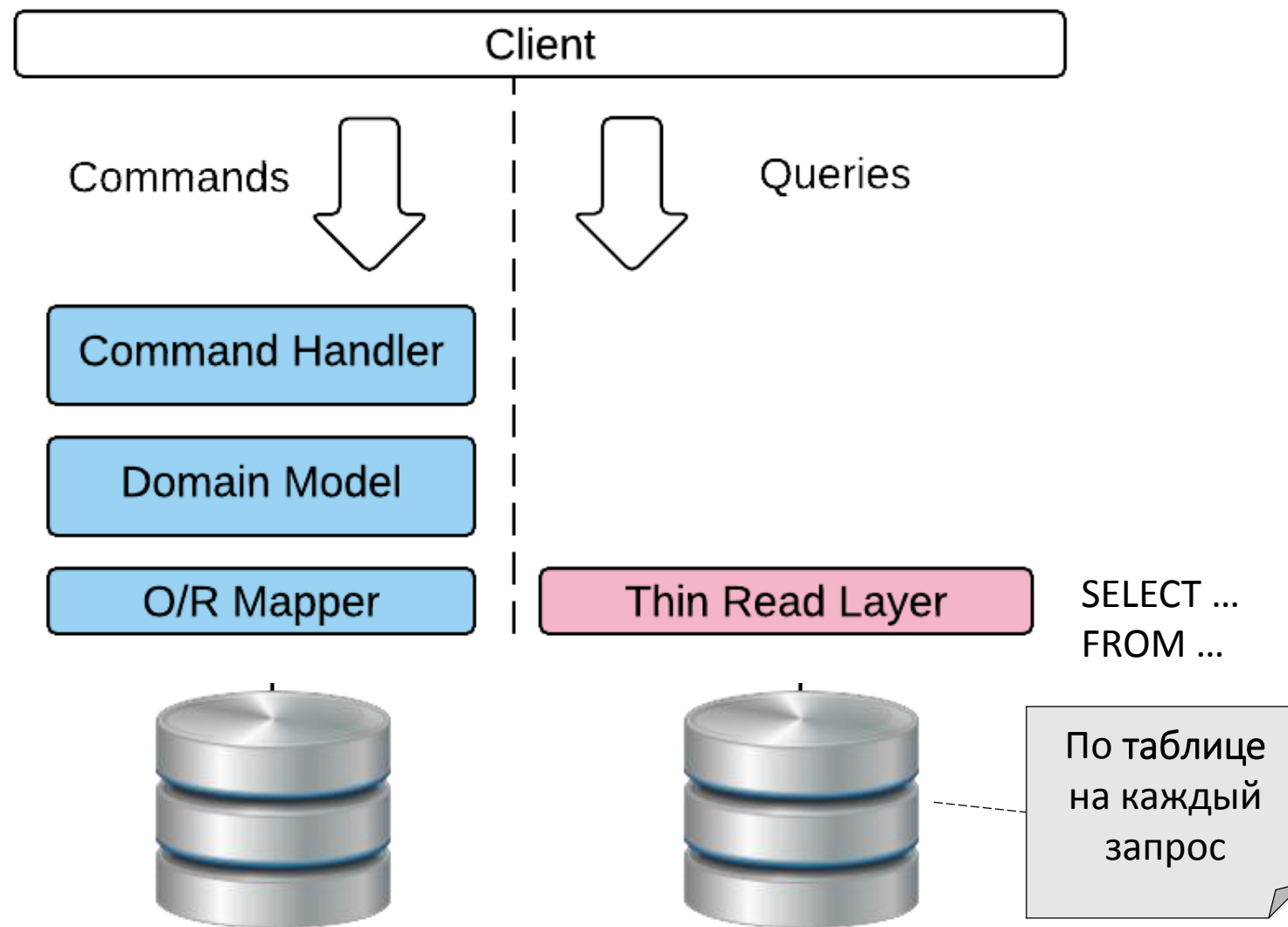
Event Sourcing + CQRS

Шаг 2



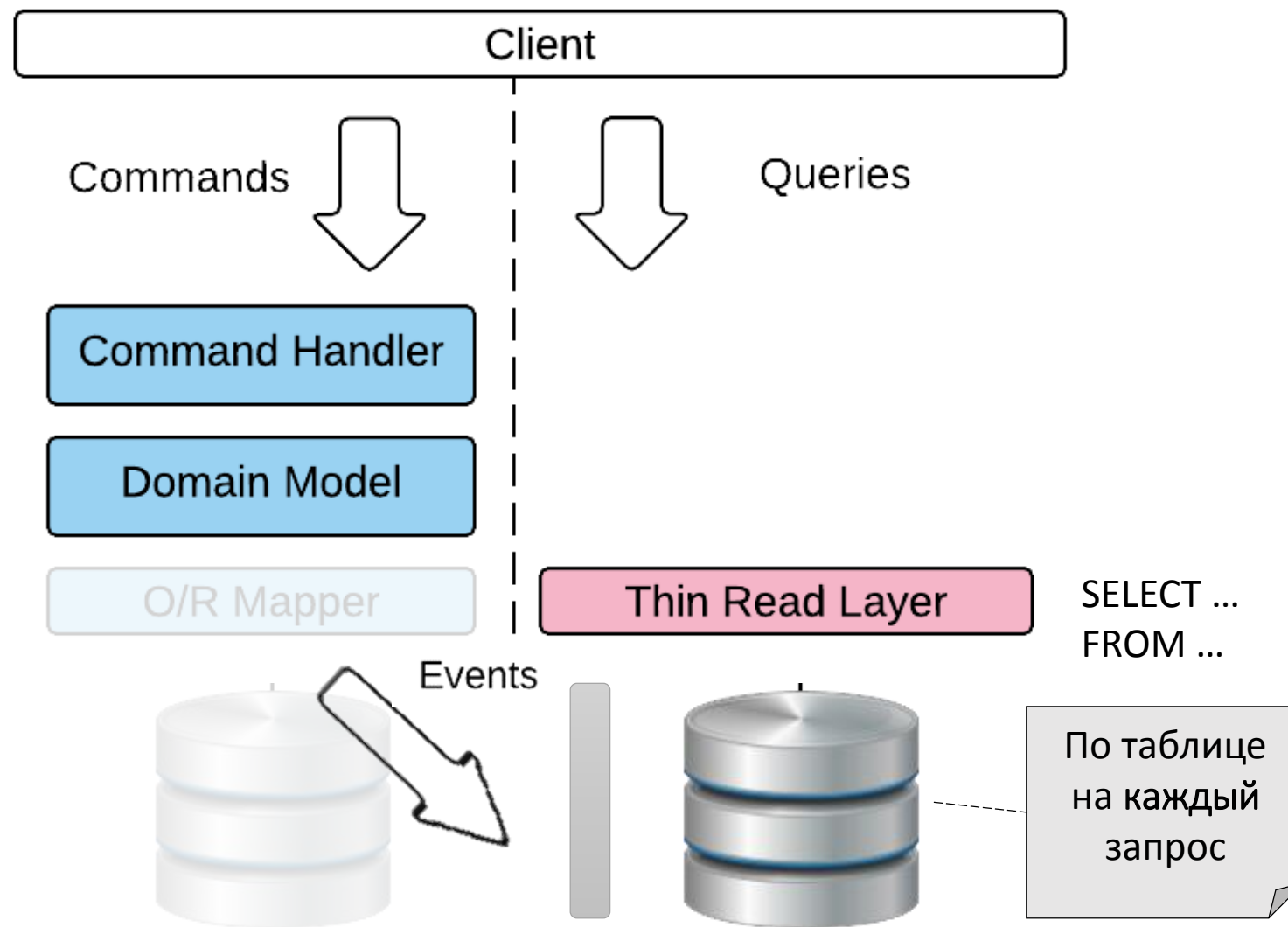
Event Sourcing + CQRS

War 3

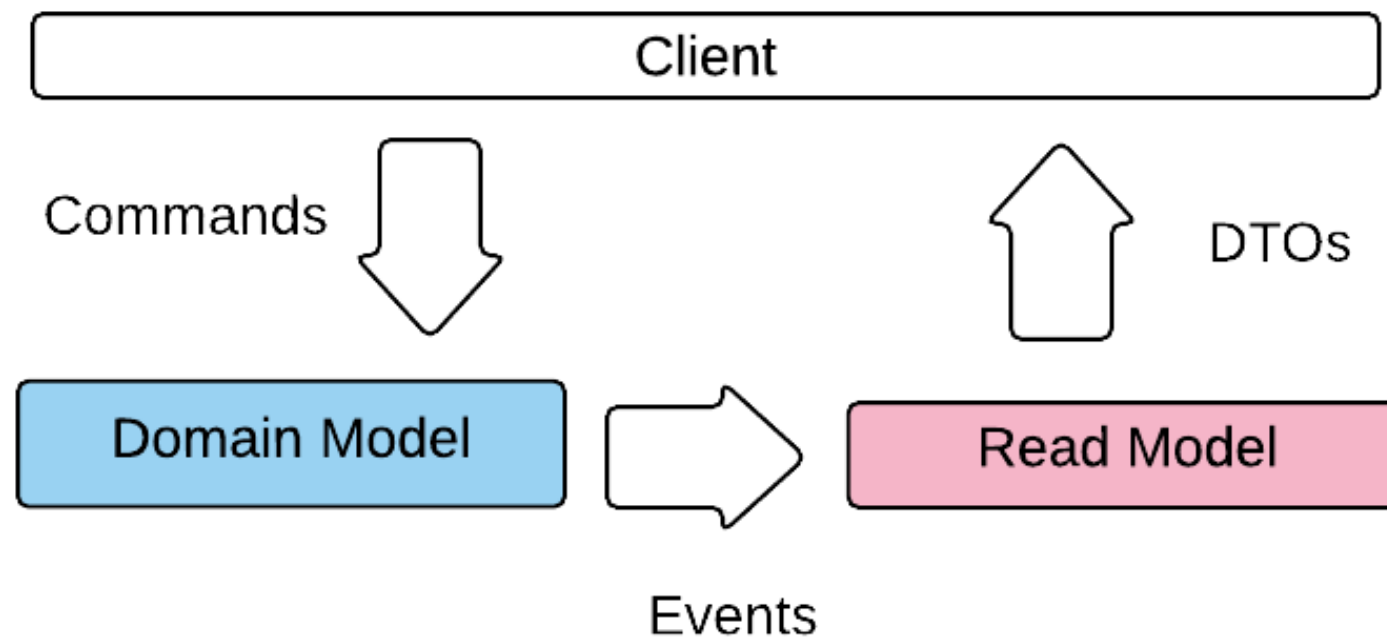


Event Sourcing + CQRS

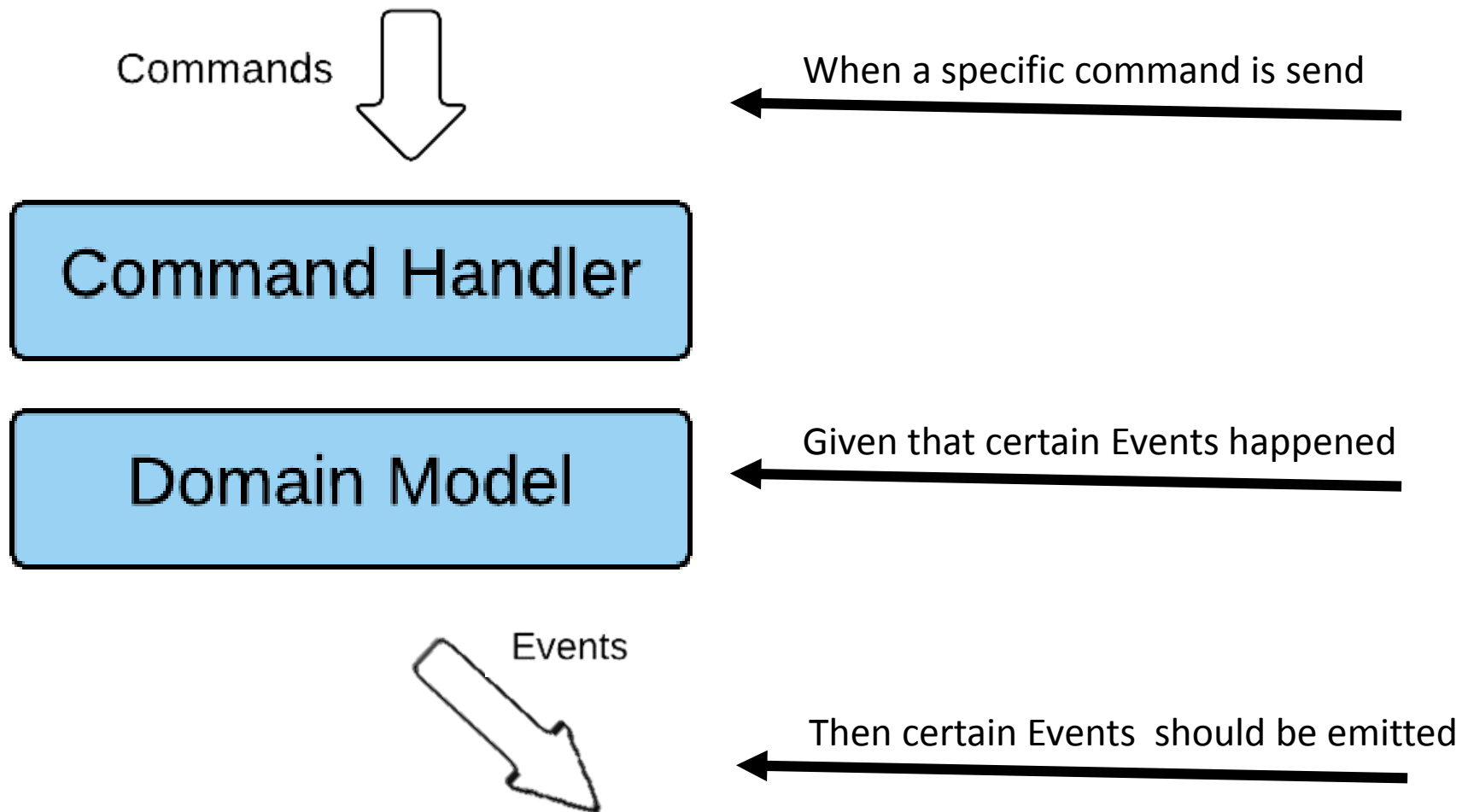
Шаг 4



Упрощенная модель

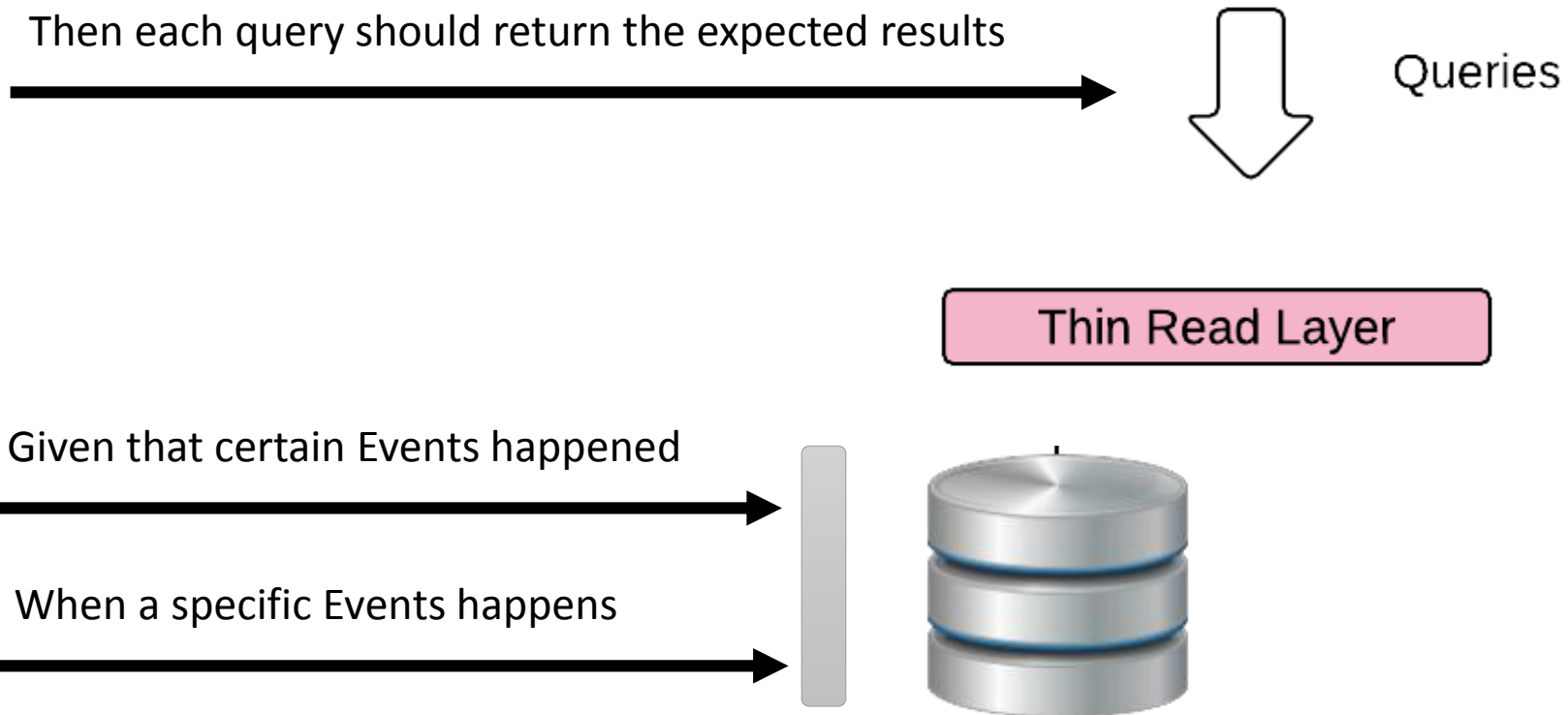


Тестируемость Модель записи

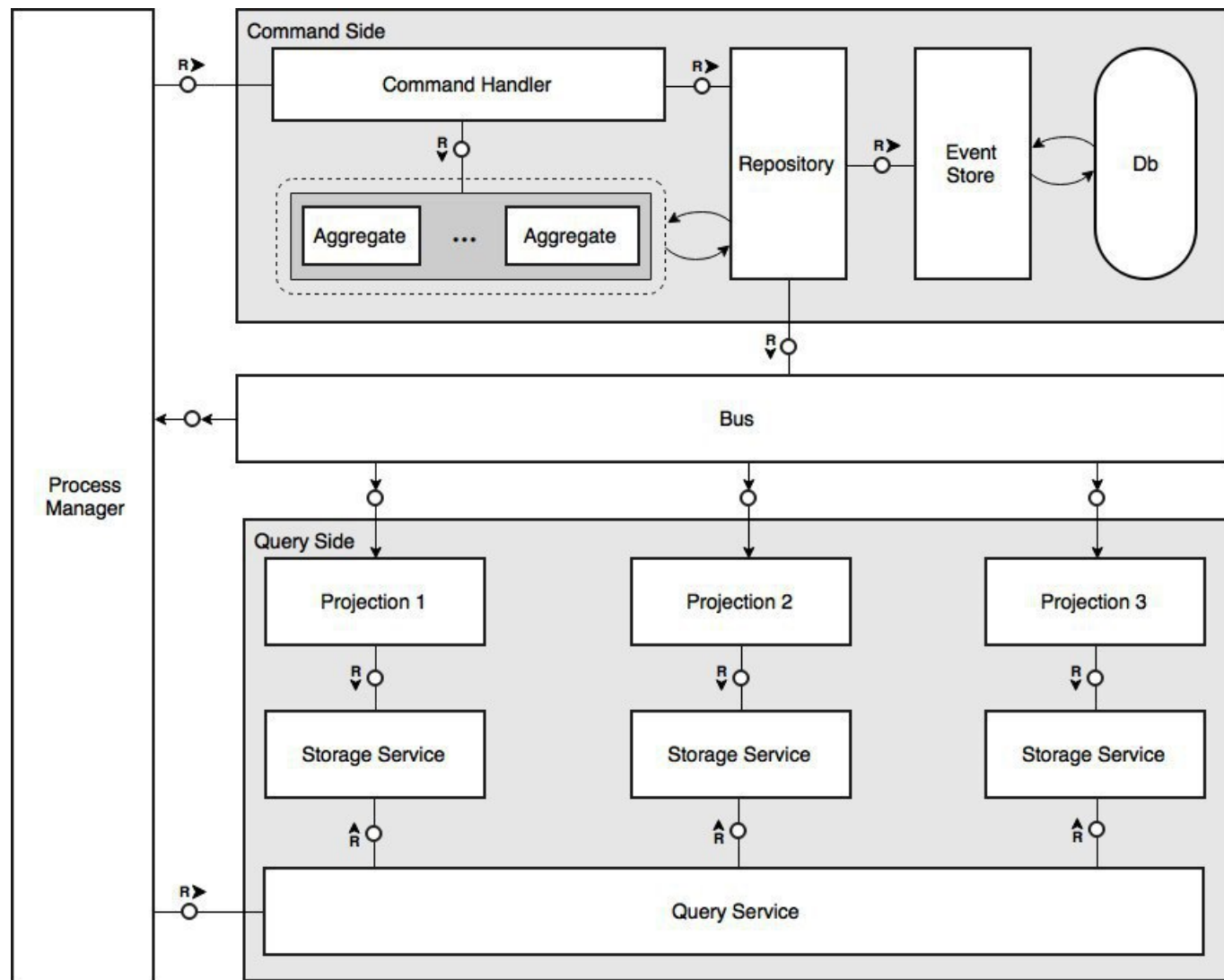


Тестируемость

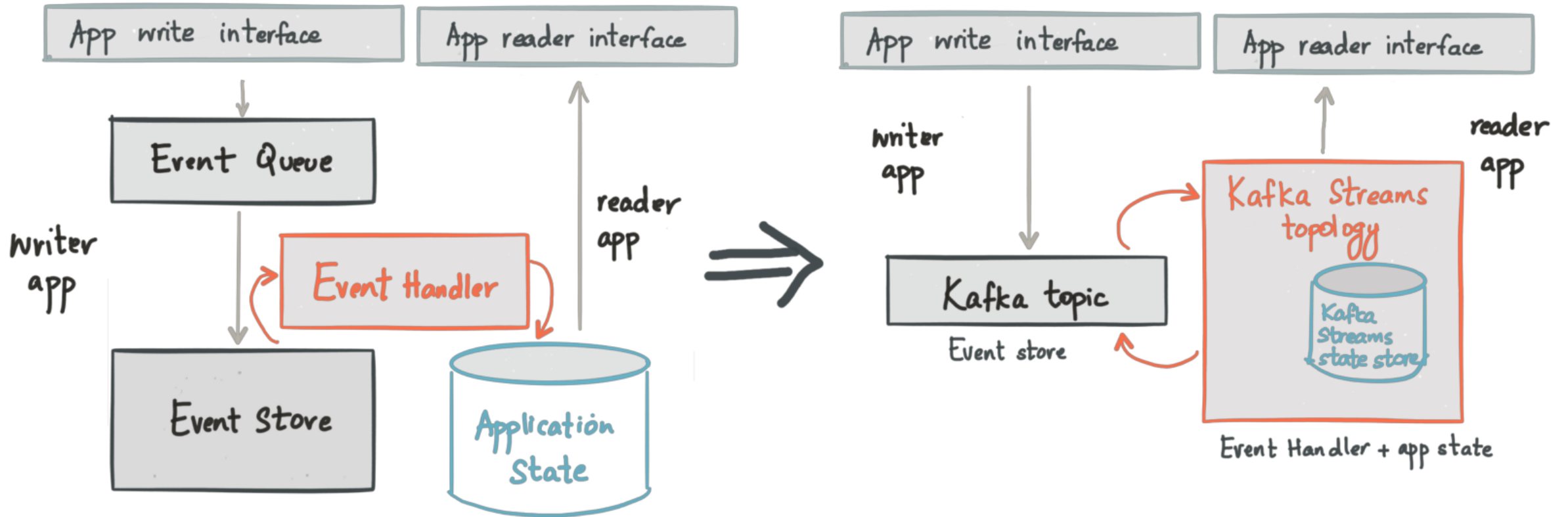
Модель чтения



CQRS + ES



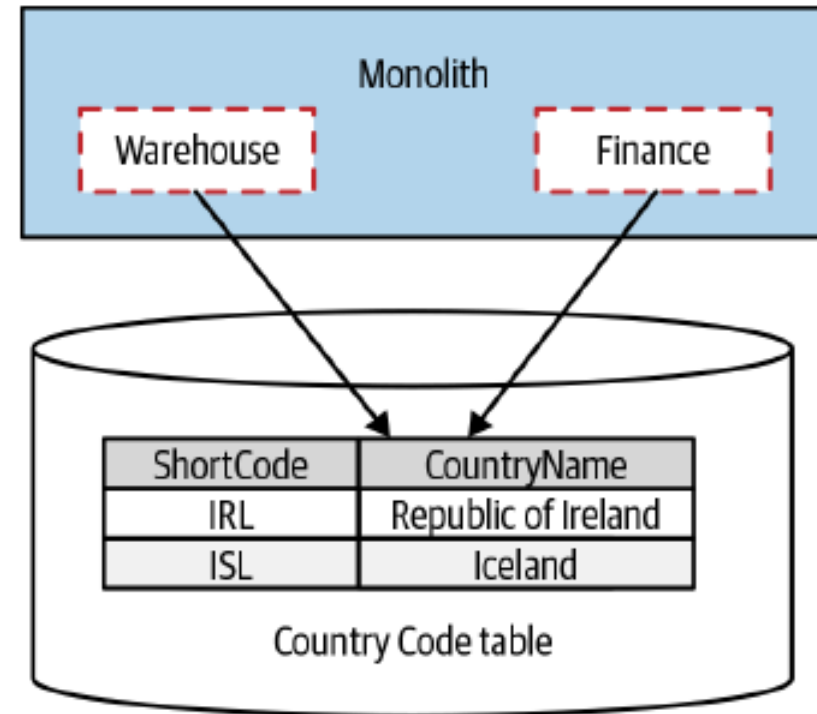
Использование журнала событий в модели CQRS



Справочные данные

Справочные данные (reference data)

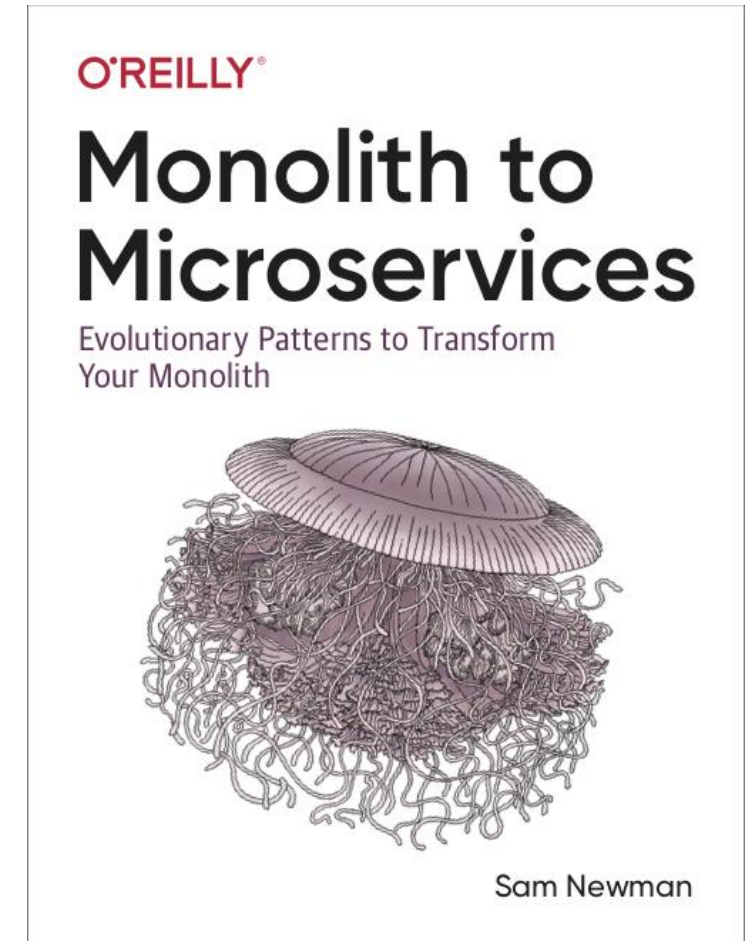
- **Характеристики**
 - Немногочисленны
 - Редко изменяются
 - Востребованы различными сервисами
- **Например**
 - Код страны



Паттерны поддержки справочных данных

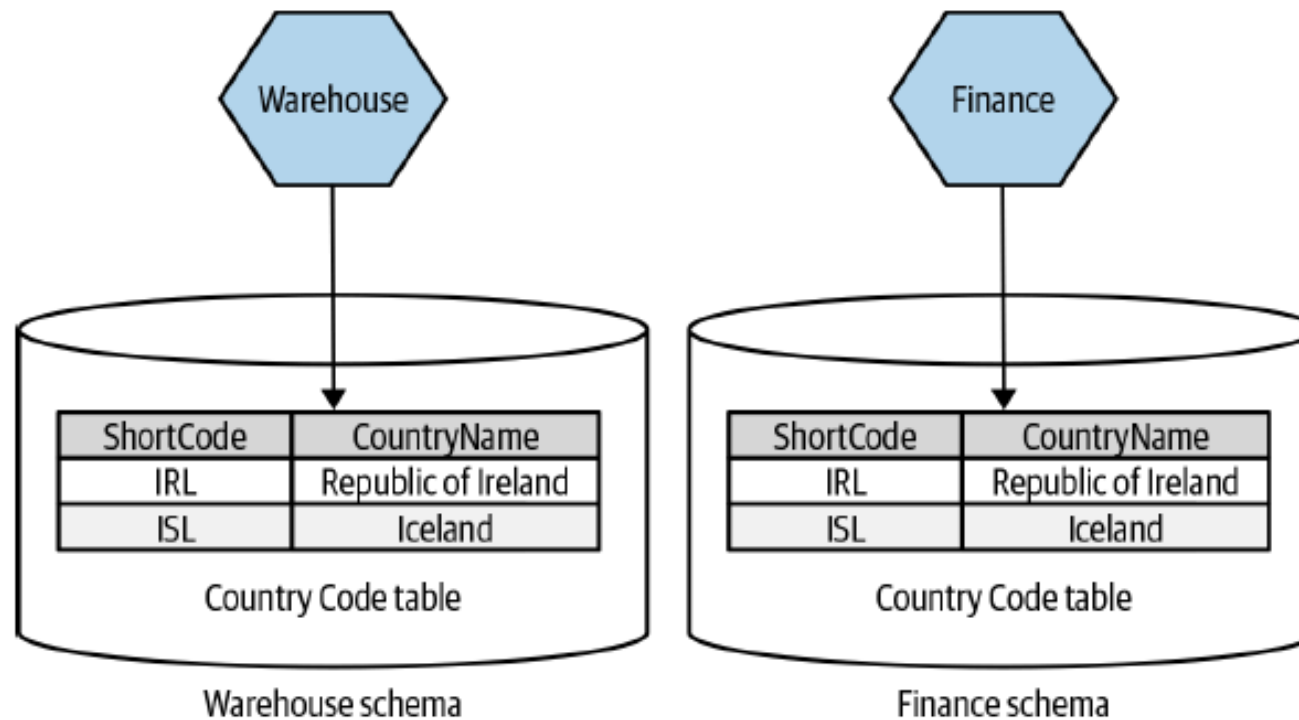
Сэм Ньюман

- Дублирование справочных данных
(Duplicate static reference data)
- Выделение схемы (Dedicated reference data schema)
- Общая библиотека (Static reference data library)
- Справочный сервис (Static reference data service)



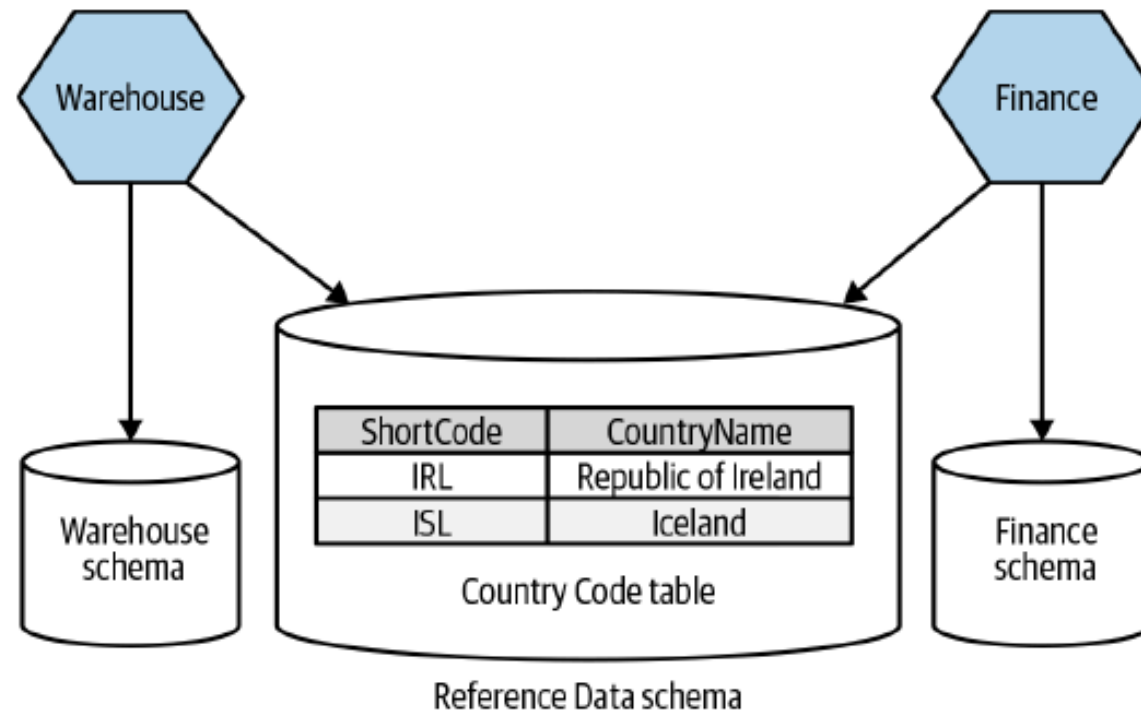
Дублирование справочных данных (Duplicate static reference data)

- Редкое решение
- Проблема синхронизации копий



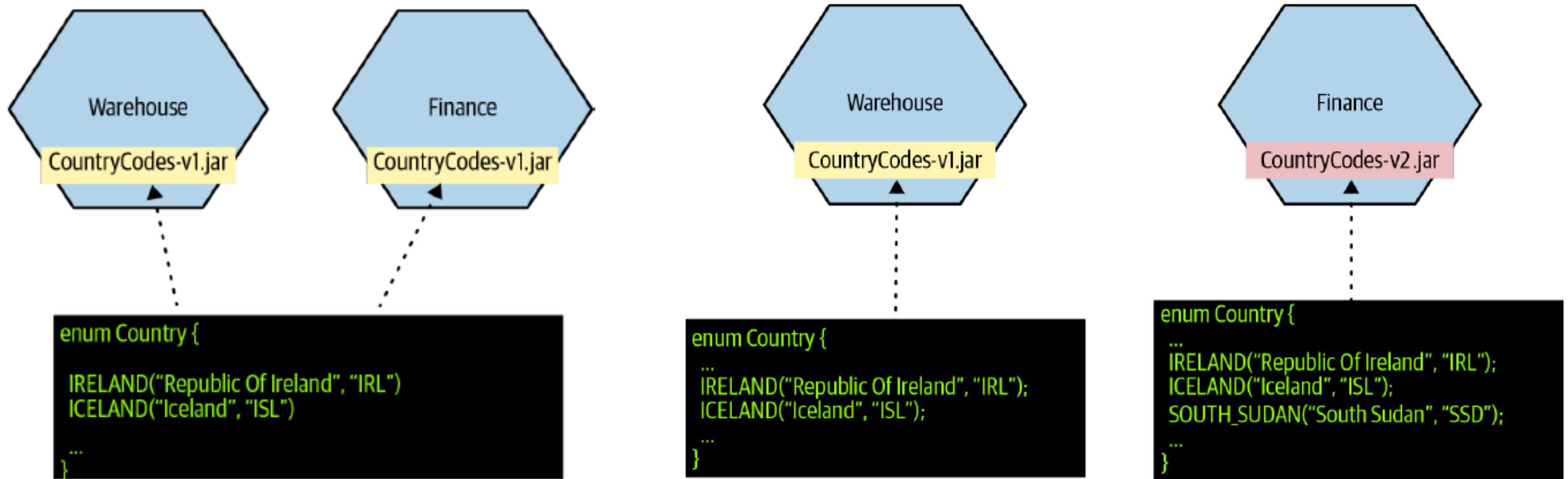
Выделение схемы (Dedicated reference data schema)

- **Высокая связанность**
 - Смена формата повлияет на все сервисы



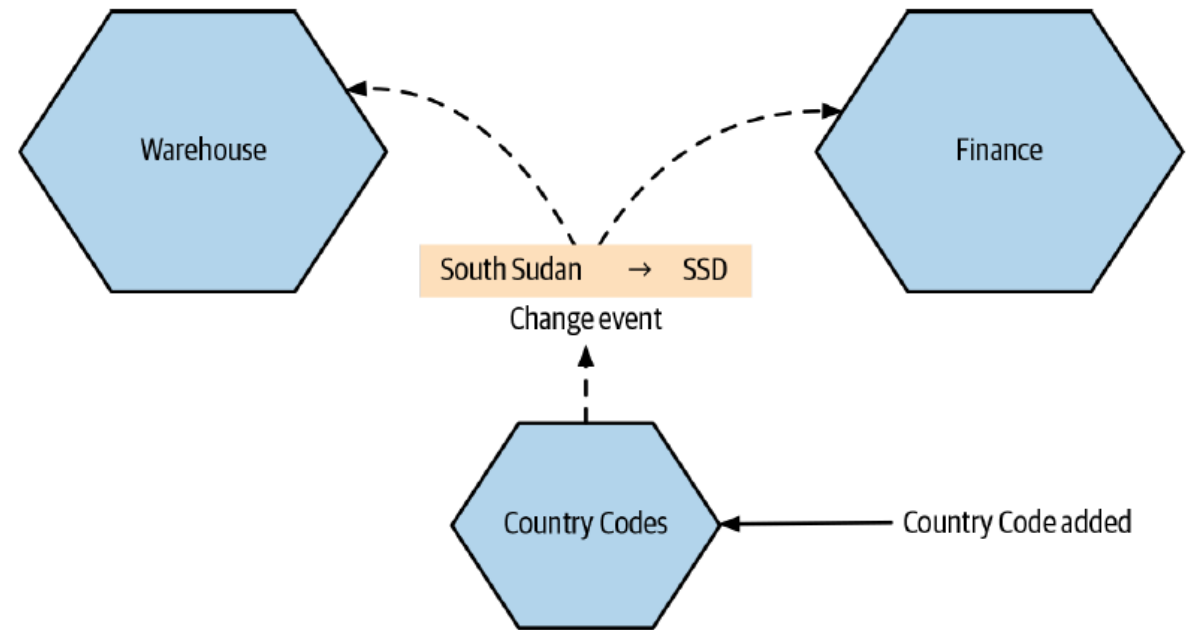
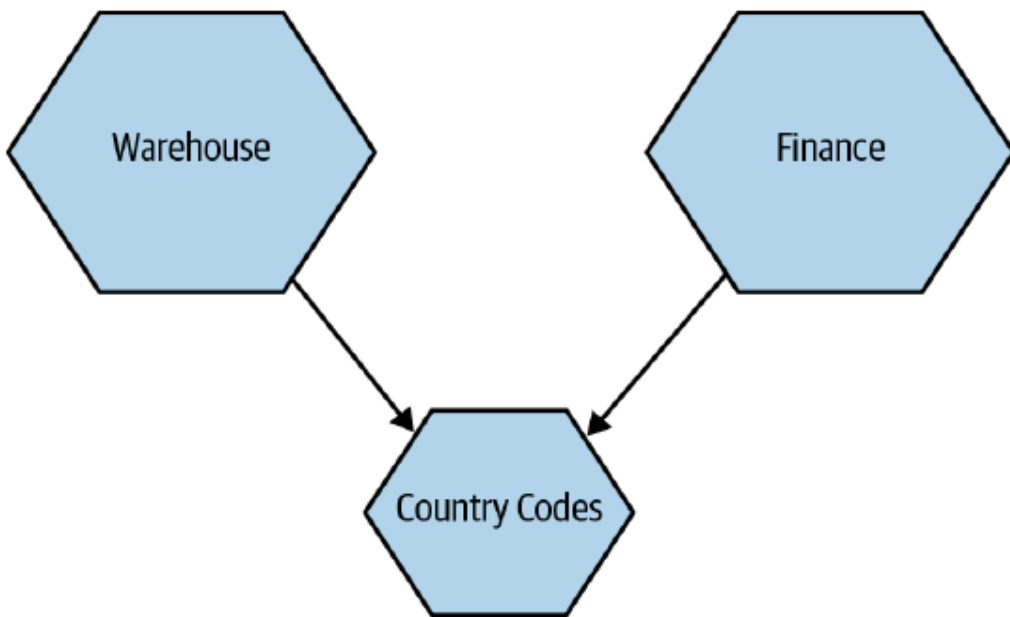
Общая библиотека (Static reference data library)

- Возможно нарушение согласованности



Справочный сервис (Static reference data service)

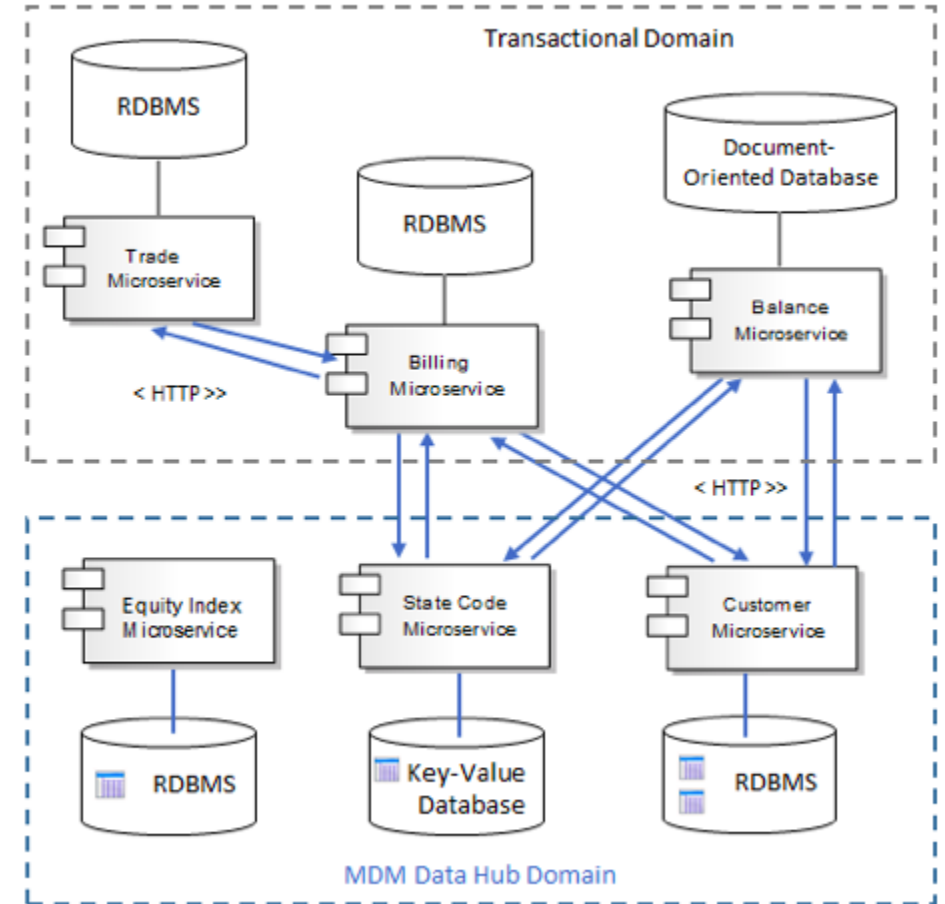
- Проблема производительности (сетевые запросы)
 - Решение: подогреваемые кэши

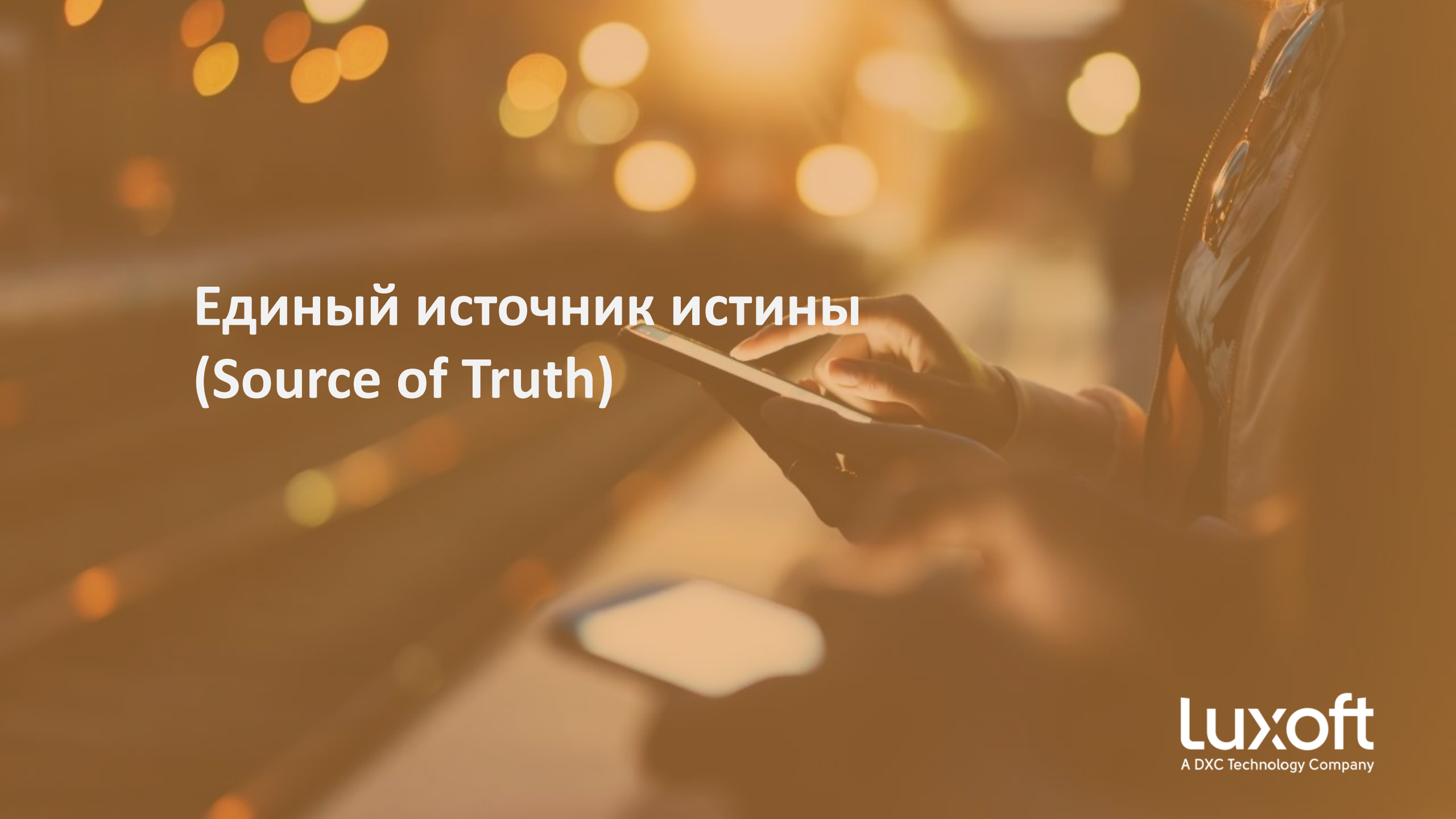


Нормативно-справочная информация (НСИ) Master Data Management, MDM

Совокупность процессов и инструментов для постоянного определения и управления основными данными компании (в том числе справочными)

- Основные задачи:
 - устранение дублирования в различных подсистемах
 - введение единой идентификации
 - база для сервиса авторизации
- Объединяем все справочники в одну систему, но оставляем возможность различных технологических решений для каждого справочника





Единый источник истины (Source of Truth)

Единый источник истины (Source of Truth, SoT)

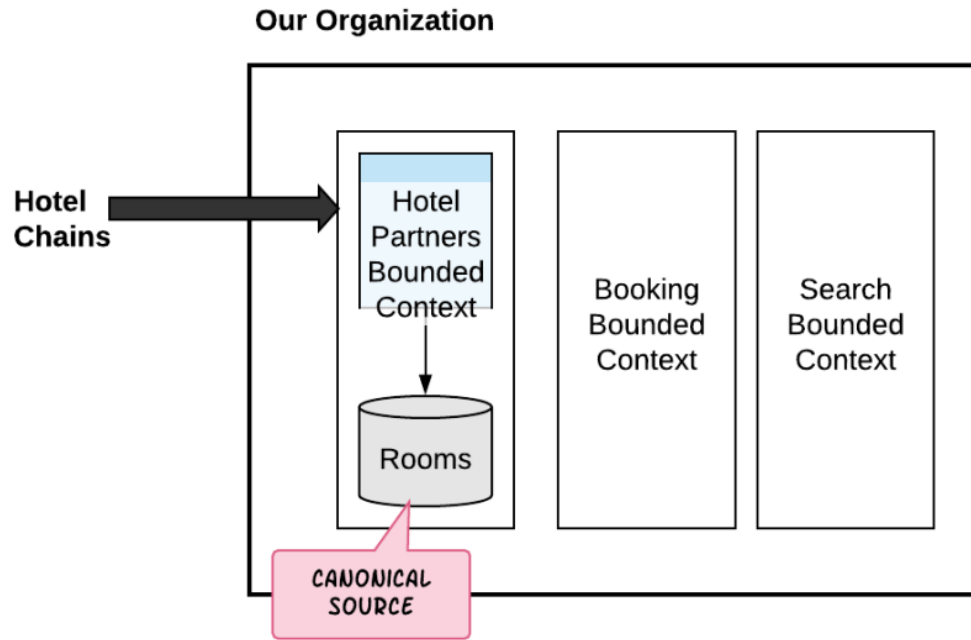
- Практика структурирования информационных моделей и связанных с ними схемы данных таким образом, что каждый элемент данных хранится ровно один раз.
- Любые возможные связи с этим элементом данных осуществляются только по ссылке.

Единый источник истины: Проблемы

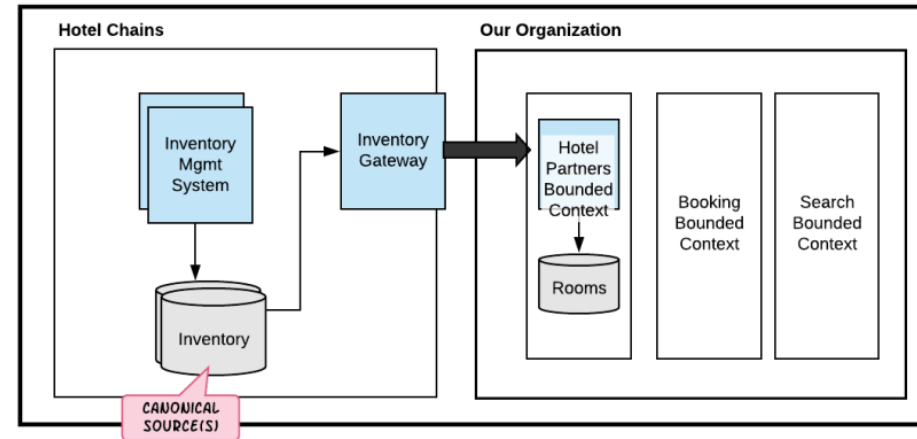
- Единый источник истины формирует зависимости между командами и ограниченными контекстами
- Единый источник истины – единая точка отказа (single points of failure)
- Единый источник истины сдерживает развитие архитектуры
- Единый источник истины **анти-паттерн** в MSA

Единый источник истины: Пример

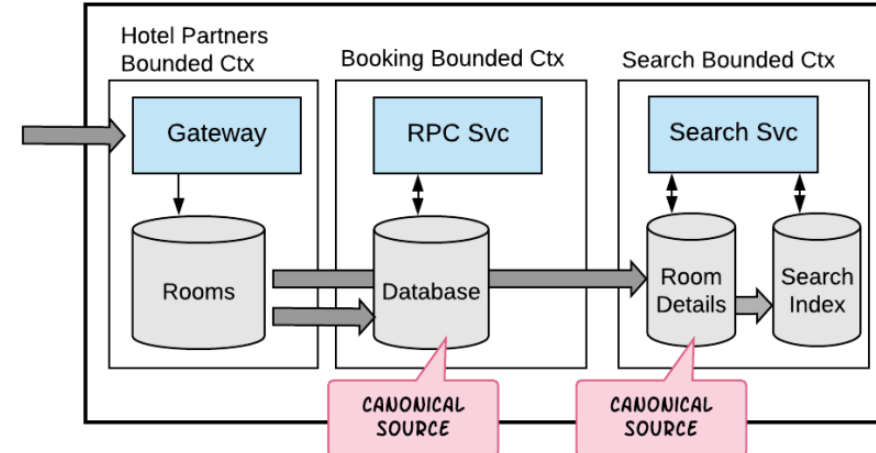
■ Система бронирования отелей



The Industry

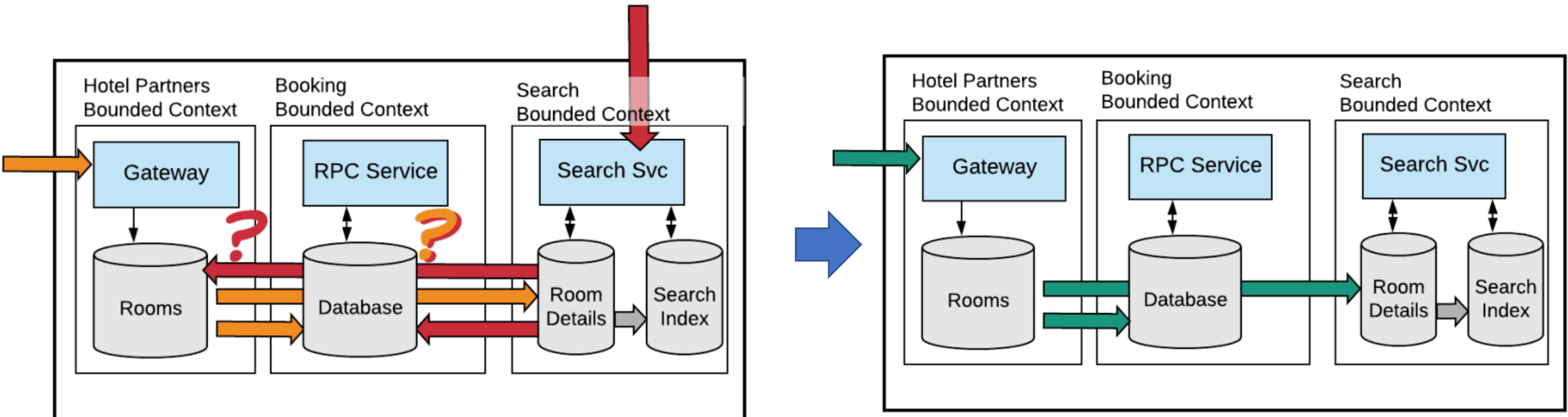


Our Organization



Направление обновления

- Однако, чтобы избежать проблем «зацикливания» данных при обновлении из разных источников, лучше всегда вести обновление в одном направлении
- Это так же решит проблему возникновения конфликтов



Master Data Management и SoT

- **MDM не относится к SoT и не является анти-паттерном,**
в случае, если остается достаточно легковесным и не содержит информацию,
необходимую только в некоторых контекстах

Единая версия истины (Single Version of Truth, SVoT)

- Если некоторые приложения производят одни и те же расчеты, основываясь на одинаковых данных, их результат должен быть одинаков.
- Решения:
 - Общие библиотеки
Недостатки: в случае изменения переразвертываем все связанные сервисы, привязываем себя к одному языку
 - Выделенная служба
Получаем результат аналогичный SoT со всеми вытекающими проблемами
 - Публикатор расчетов (Calculation Publisher)
При изменении данных, сервис проводит расчеты и публикует результаты

Буду рад ответить на ваши вопросы

