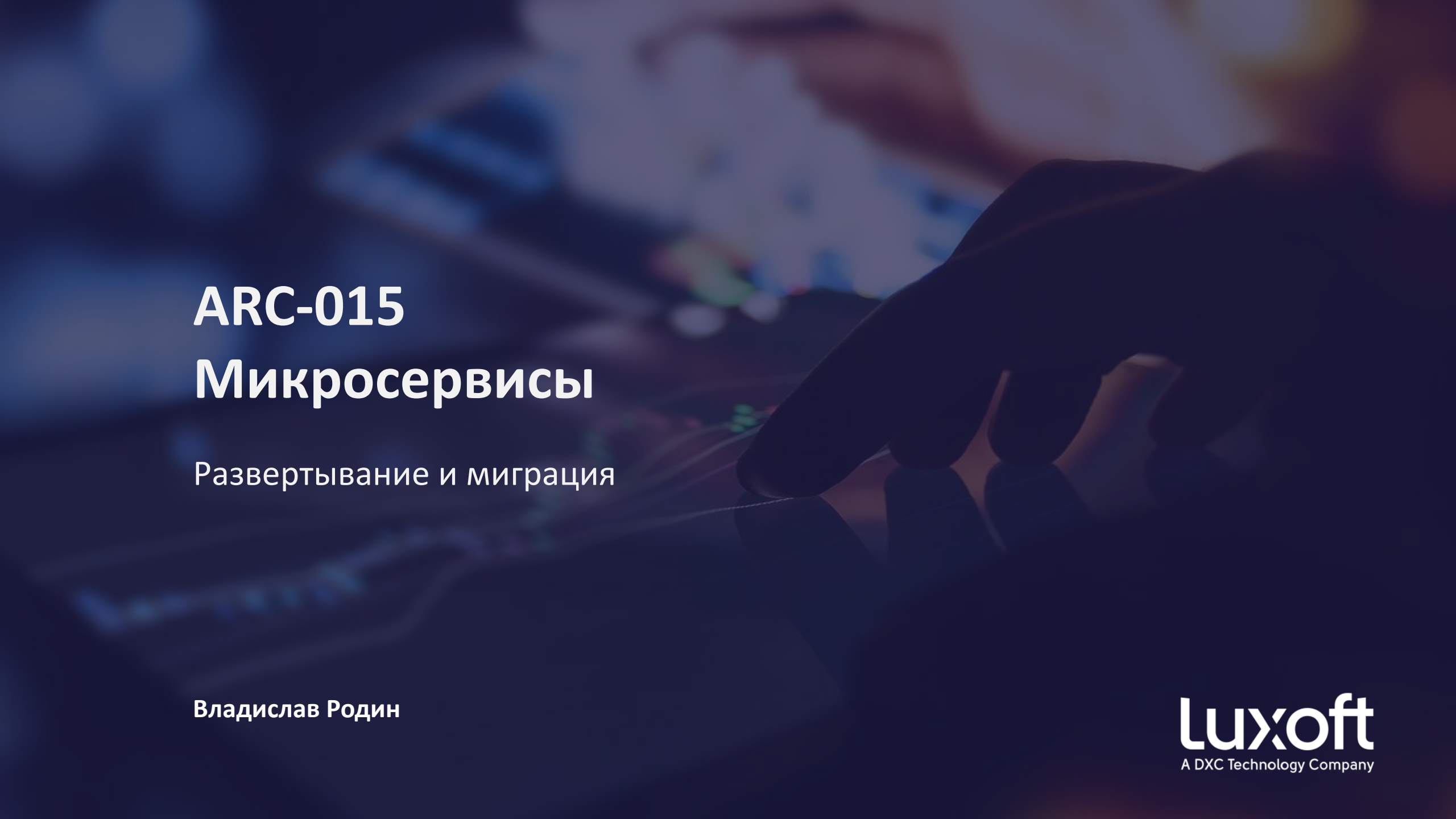


ARC-015
Микросервисы

Развертывание и миграция

Владислав Родин

Luxoft
A DXC Technology Company

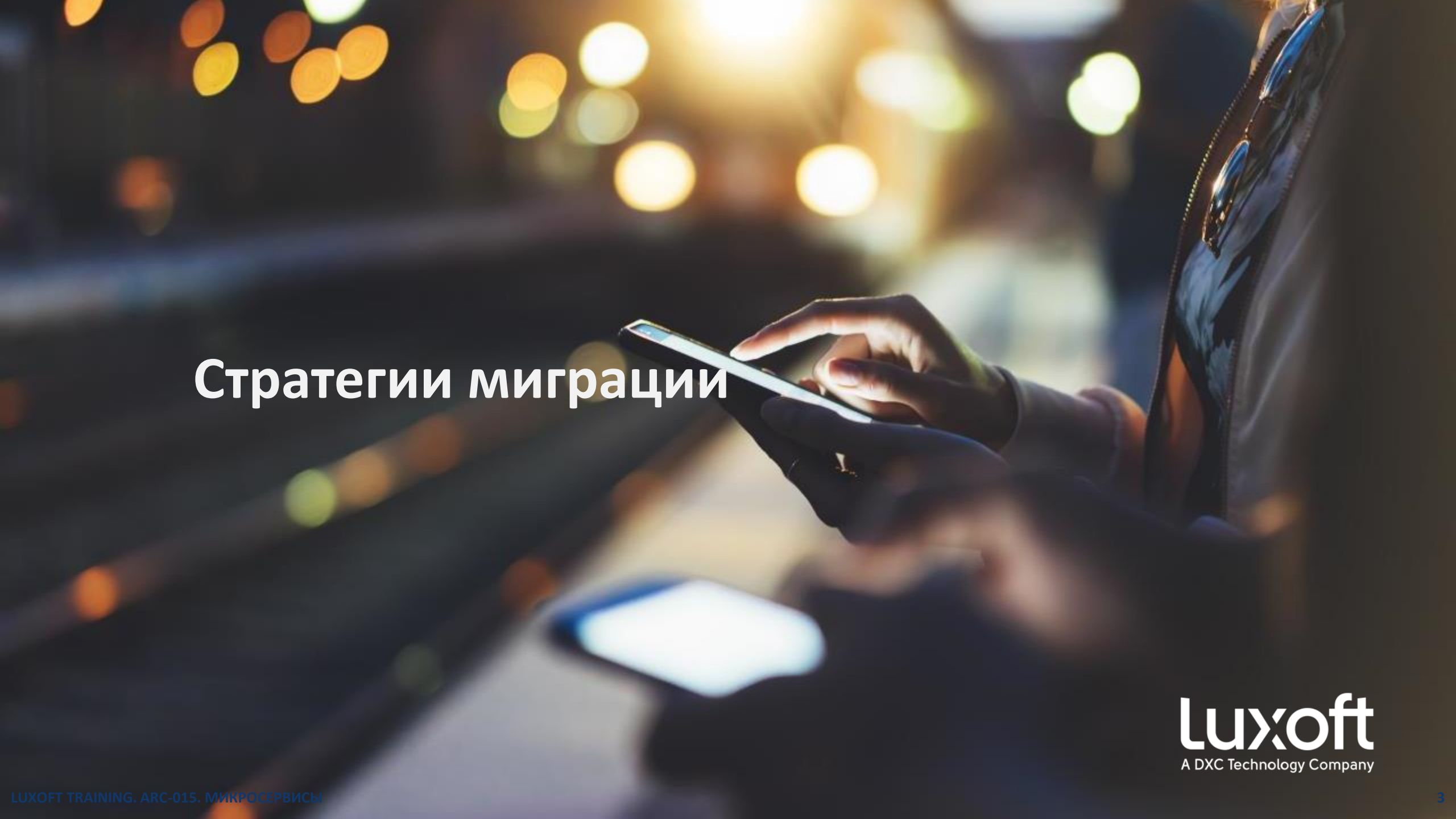
Проблема

- Предположим, что имеется система, построенная в стиле «монолит»
- Микросервисная архитектура является целевой
- Каким образом можно осуществить переход, не переписывая всю систему?
  - Начать с чистого листа и оставить старую кодовую базу в прошлом — звучит заманчиво.

    Но это чрезвычайно рискованный подход, который, скорее всего, закончится неудачей

Миграция: принципы

- **Минимизация изменений, вносимых в монолит**
  - **При переходе на микросервисную архитектуру не стоит вносить масштабные изменения в монолит.**
- **Инфраструктура развертывания: не все сразу**
  - **Как бы соблазнительно не выглядела подготовка инфраструктуры, делайте как можно меньше предварительных инвестиций в ее построение.**
  - **Единственное, без чего нельзя обойтись, — это процесс развертывания с автоматическим тестированием.**



Стратегии миграции

luxoft
A DXC Technology Company

Стратегии перехода с монолита на микросервисы

- **Реализация новых возможностей в виде сервисов.**
- **Разделение уровня представления и внутренних компонентов.**
- **Разбиение монолита путем оформления функциональности в виде сервисов.**

Если вы оказались в яме, прежде всего перестаньте копать

Закон ямы

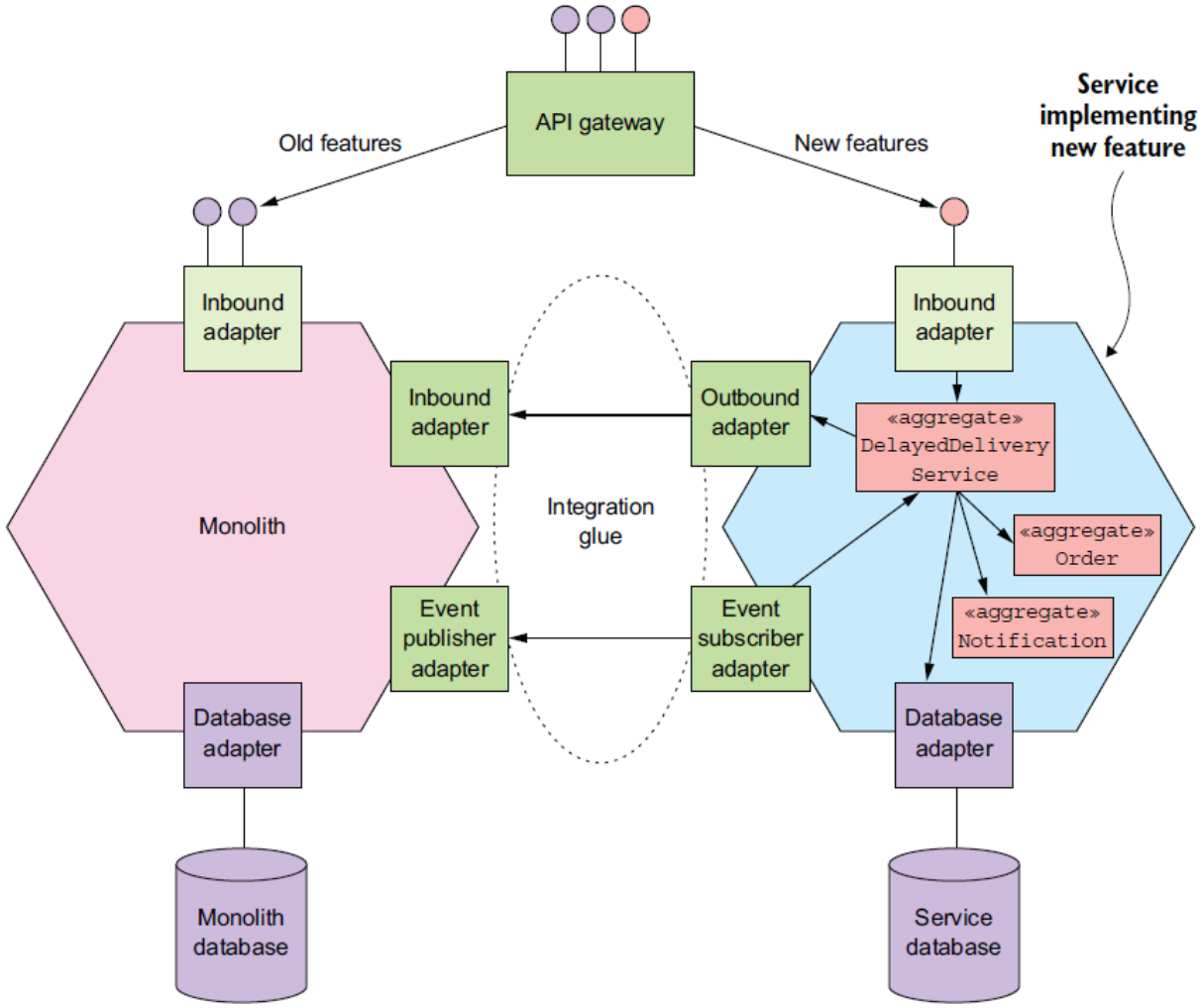
Реализация новых возможностей в виде сервисов

Если у вас есть большой и сложный монолитный проект, прекратите добавлять в него новые возможности, иначе он станет еще более крупным и неуправляемым. Вместо этого новые функции следует реализовывать в виде сервисов

- **Проблема:**
  - **Не всегда возможно реализовать новый функционал как сервис**
    - **В связи с высокой связанностью монолита и потребности в согласованности данных, новые функции можно оставлять в границах монолита для последующего извлечения**
  - **Интеграция новых сервисов с монолитом**

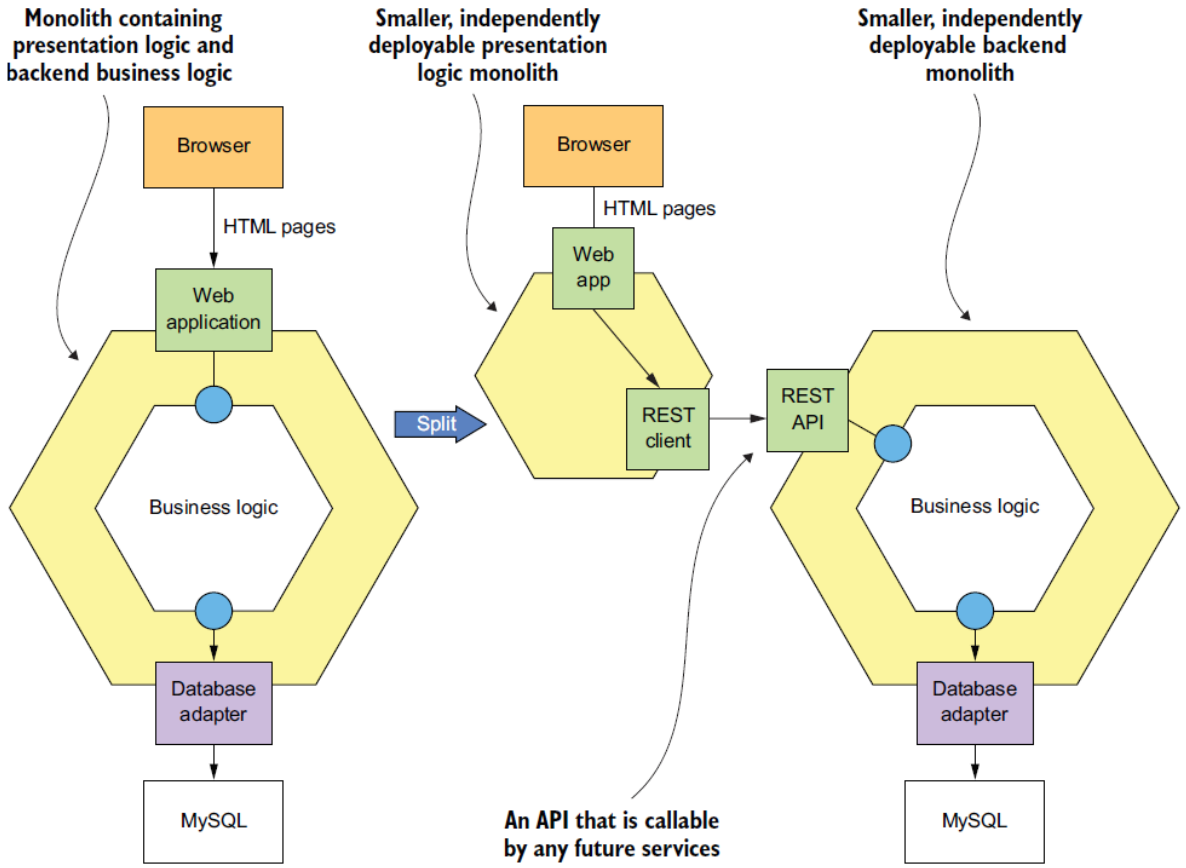
Интеграция нового сервиса с монолитом

- **API-шлюз — направляет запросы новой функциональности к новым сервисам, а старые запросы — к монолиту**
- **Интеграционный связующий код — интегрирует сервисы в монолит. Позволяет сервису обращаться к данным и функциям, принадлежащим монолиту**
  - **Не является самостоятельным компонентом**



Разделение уровня представления и внутренних компонентов

Отделение уровня представления от бизнес-логики и слоя доступа к данным.

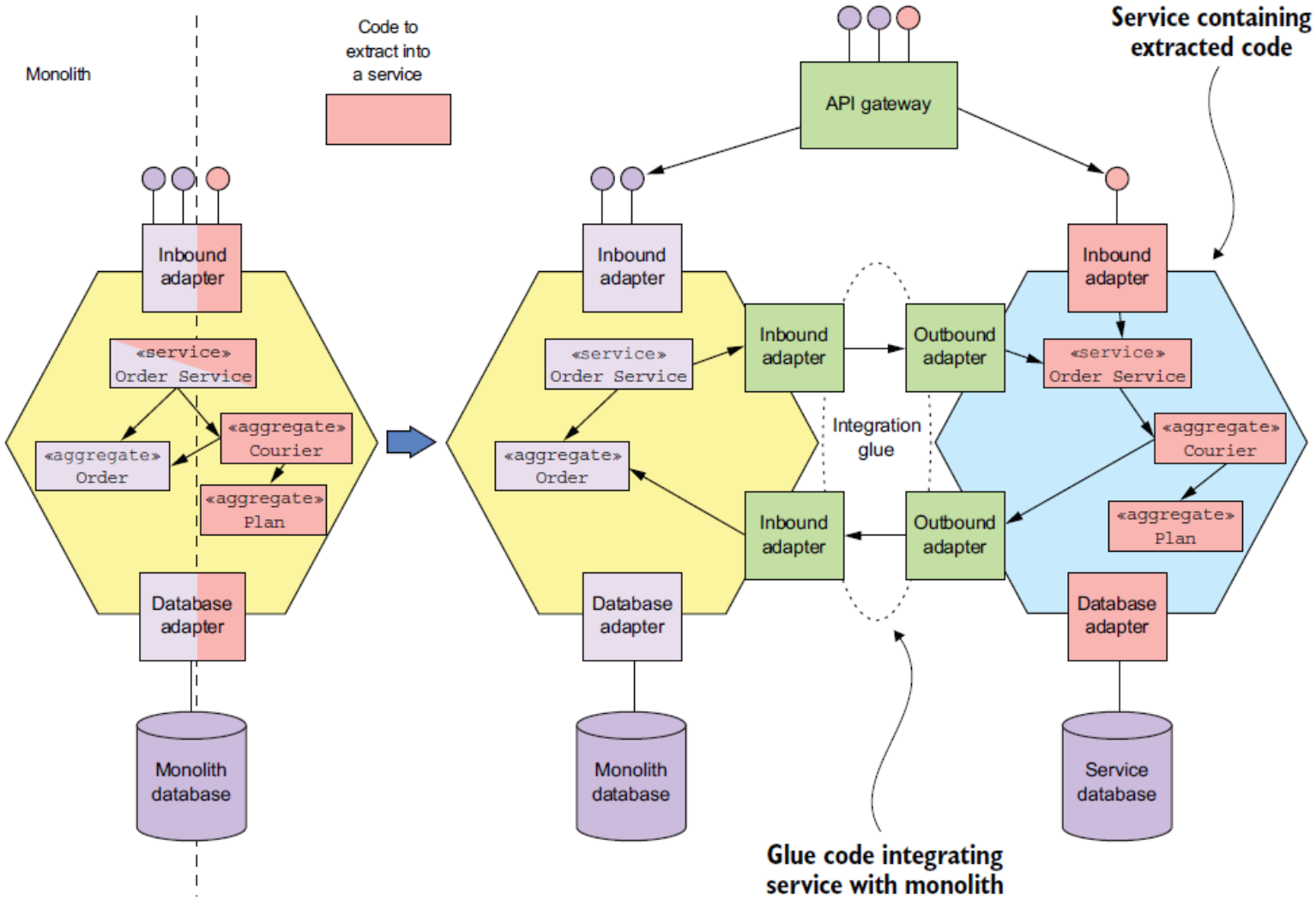


Извлечение бизнес-возможностей в сервисы

Разбейте монолит на части, постепенно перенося его бизнес-возможности в сервисы

- **Для извлечения функций в сервисы необходимо брать вертикальный срез монолита, который состоит из следующих компонентов:**
  - **входящих адаптеров, реализующих конечные точки API;**
  - **доменной логики;**
  - **исходящих адаптеров, таких как логика доступа к БД;**
  - **схемы базы данных монолита**

Извлечение бизнес-возможностей в сервисы: Схема



Какие сервисы и в какой момент нужно извлекать

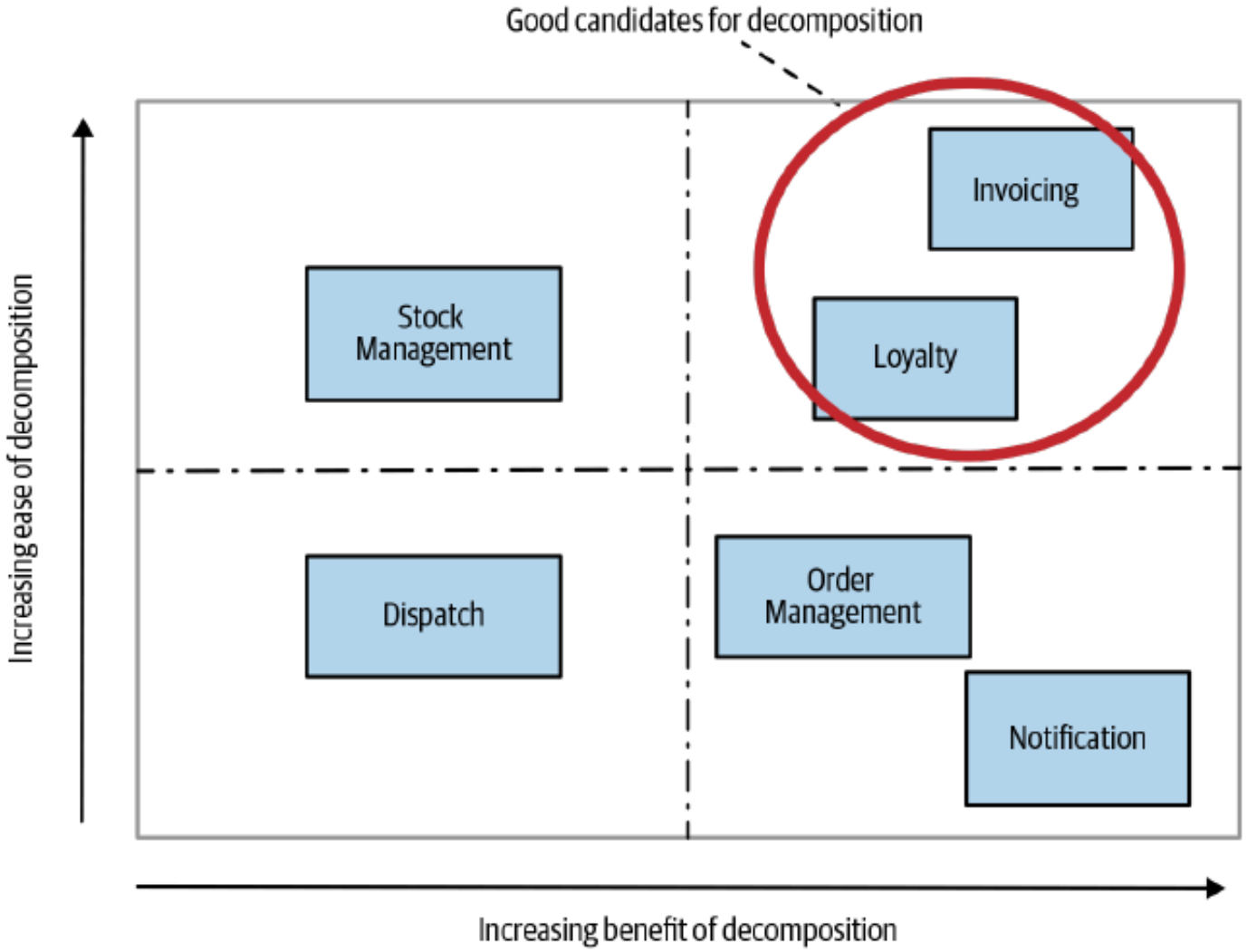
- **Стратегии:**
  - **Заморозка работы над монолитом и извлечение сервиса, в котором требуются изменения**
  - **Планирование разбиения исходя из максимальной выгоды**
    - **Ускорение разработки.**

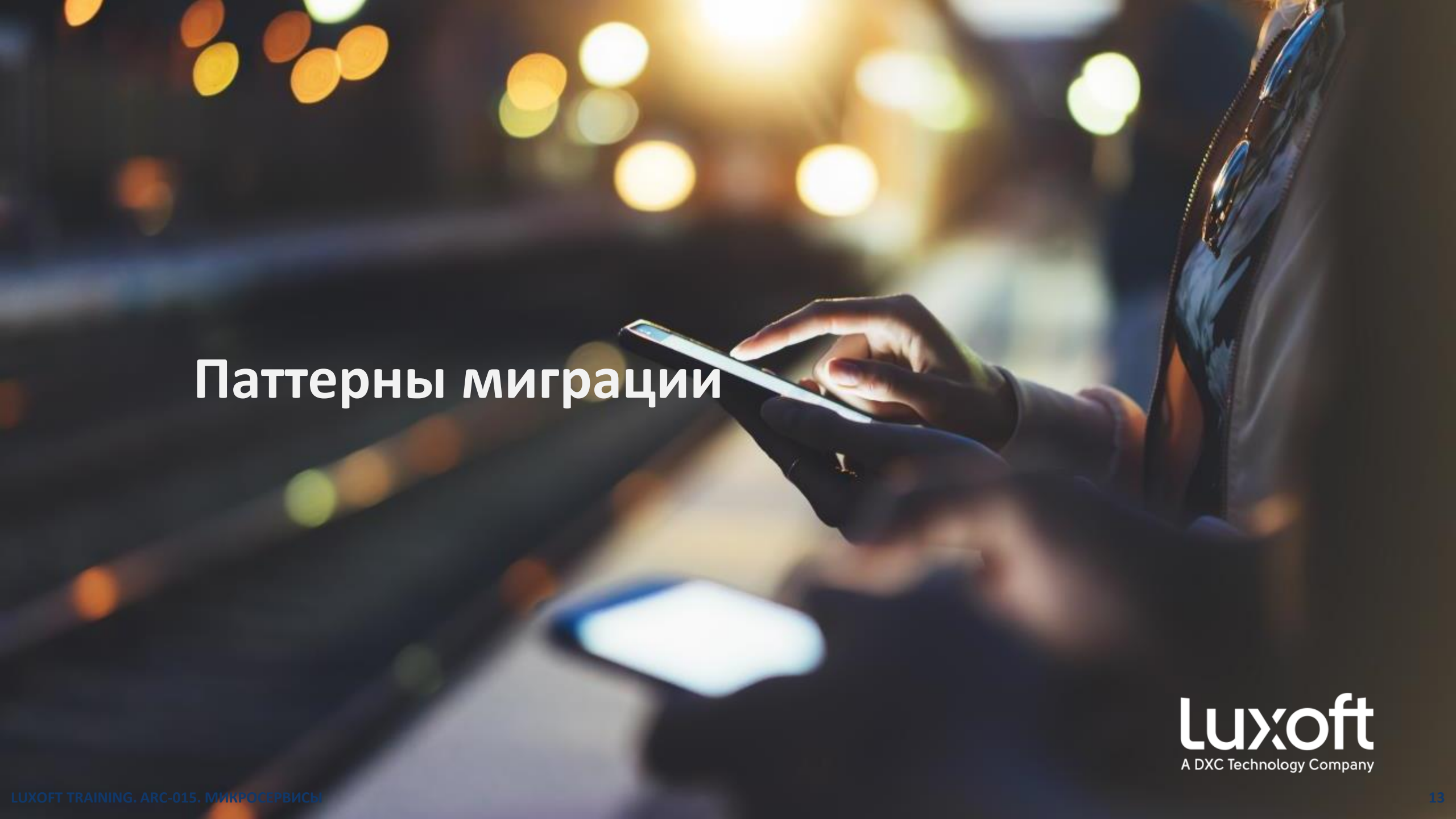
      Если согласно плану какая-то часть вашего приложения будет активно развиваться на протяжении следующего года, разработку можно ускорить путем извлечения ее в сервис.
    - **Решение проблем с производительностью, масштабируемостью и надежностью.**

      Если определенная часть вашего приложения ненадежна или имеет проблемы с производительностью или масштабируемостью, будет полезно преобразовать ее в сервис.
    - **Возможность извлечь какие-то другие сервисы.**

      Иногда из-за зависимостей между модулями извлечение одного сервиса упрощает извлечение другого.

Квадрант приоритезации



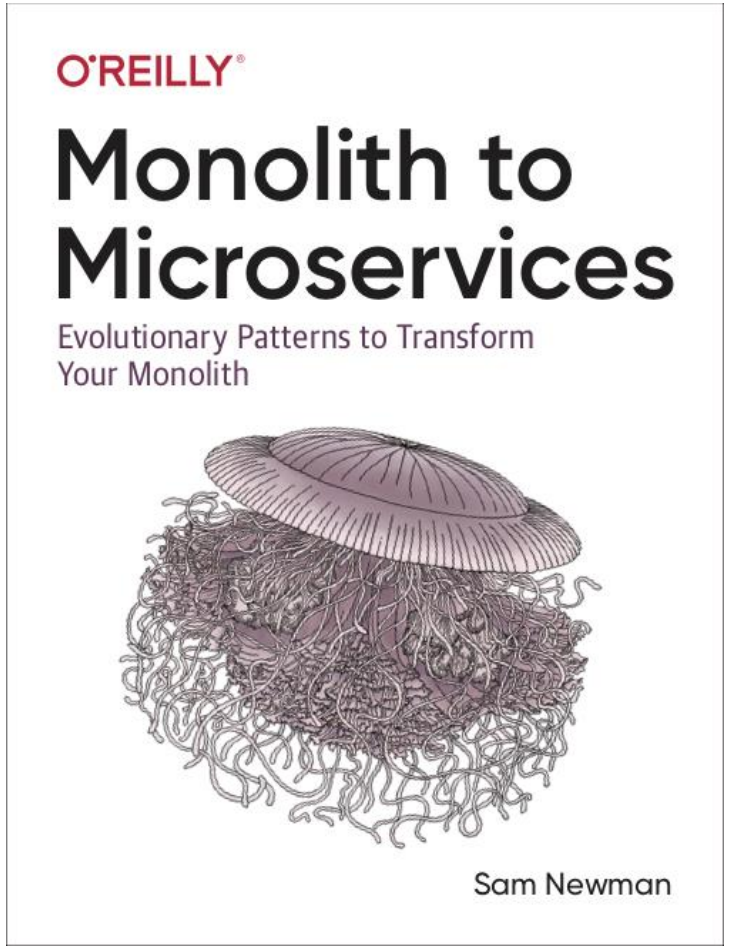


Паттерны миграции

luxoft
A DXC Technology Company

Паттерны миграции
Сэм Ньюман

- Удушение монолита (Strangler Fig Application)
- UI компоновка (UI Composition)
- Отделение через абстракцию (Branch by Abstraction)
- Паралельная работа (Parallel Run)
- Декорация участника (Decorating Collaborator)
- Захват изменения данных (Change Data Capture)



Удушение монолита (Strangler Fig Application, Strangler monolith)

- **Удушающее приложение состоит из микросервисов, работающих в связке с монолитным кодом.**

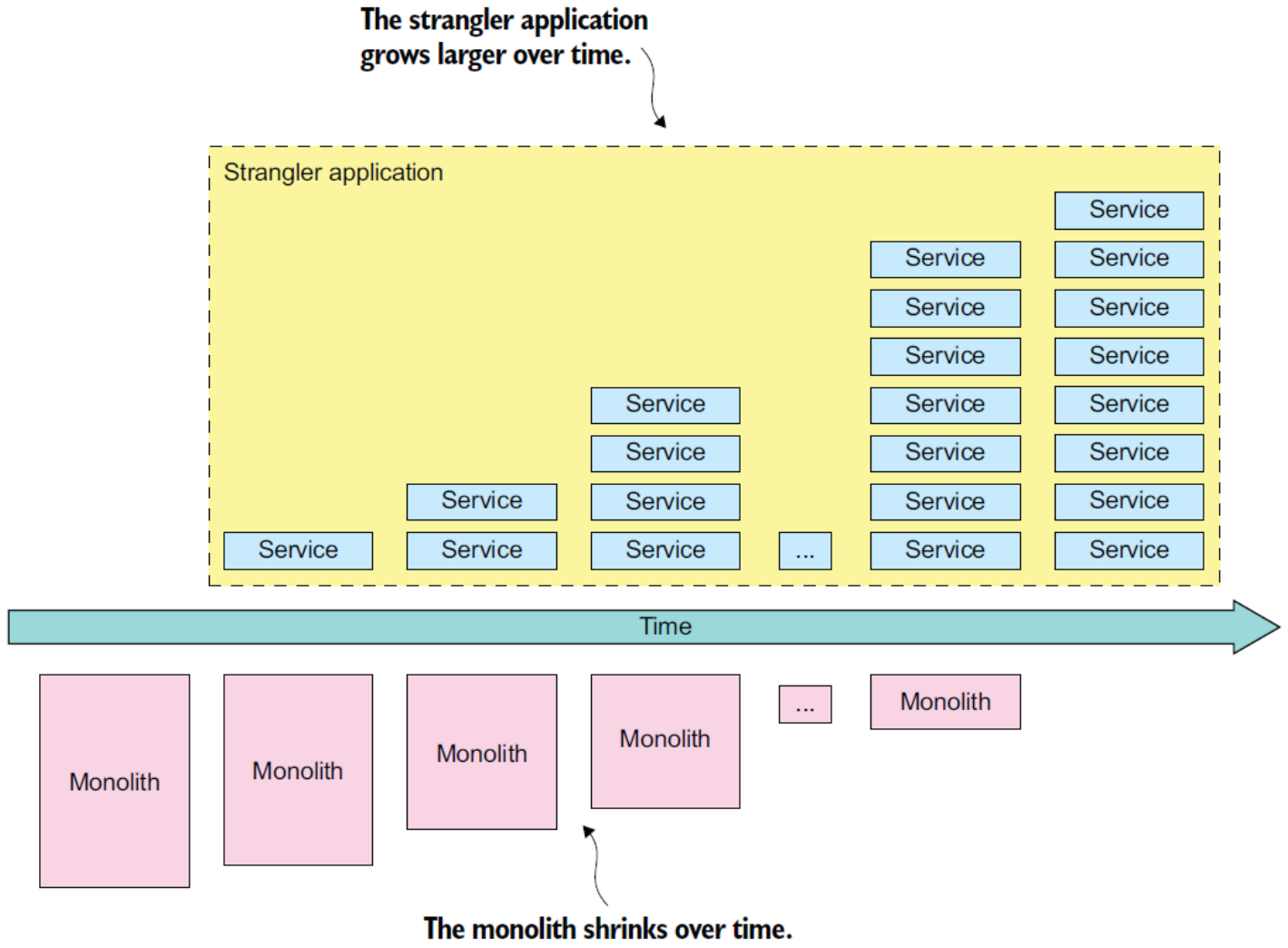
  Со временем монолитное приложение будет реализовывать все меньше и меньше функций, пока полностью не исчезнет или не превратится в еще один микросервис.

  - **Эта стратегия похожа на то, как если бы вы пытались ремонтировать свою машину прямо на ходу.**
  - **Это непросто, но куда менее рискованно, чем попытка переписывания с нуля.**

Переписывание с нуля гарантирует лишь одно — ноль!

Martin Fowler

Шаблон «Удушающее приложение» (Strangler application)

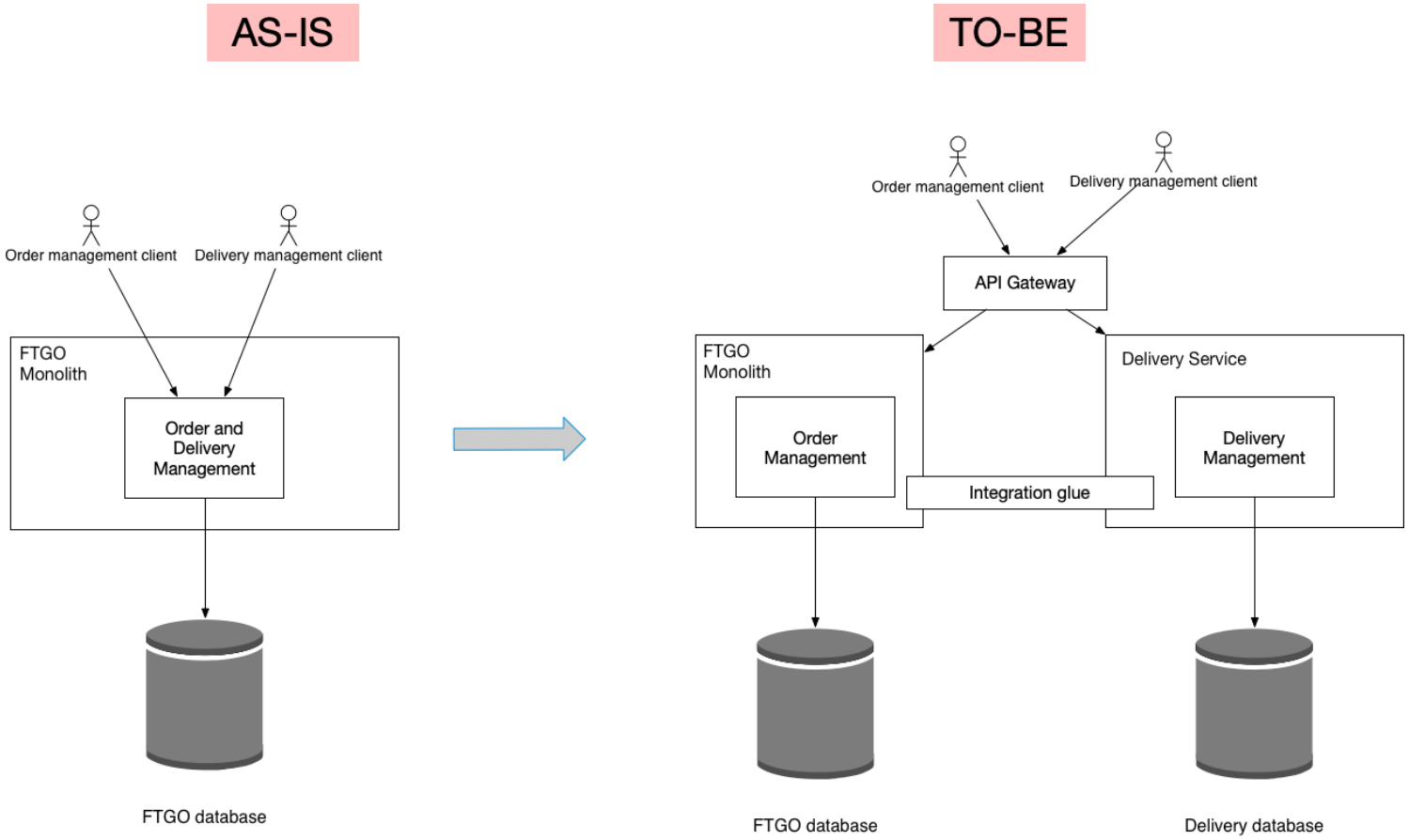


«Удушение» монолита: длительность

- Рефакторинг может занимать годы (на примере Amazon)
- Трансформация может никогда не завершиться, уступив место более важным задачам

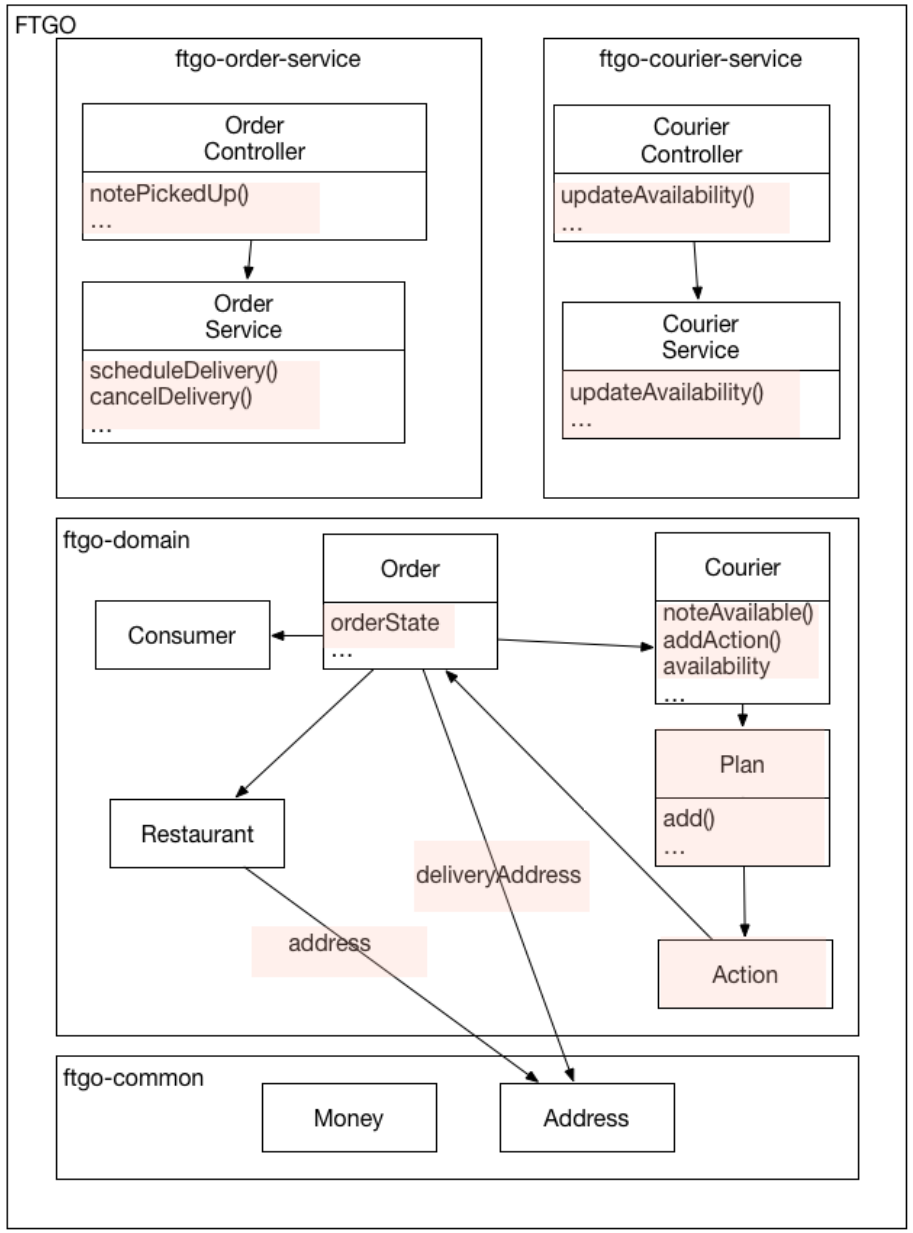
Извлечение сервисов

- **Хотя реализация новых функций в качестве сервисов чрезвычайно полезна, единственный способ устранить монолит – это постепенно извлекать модули из монолита и преобразовывать их в сервисы**



Шаг 1. Ревью кода

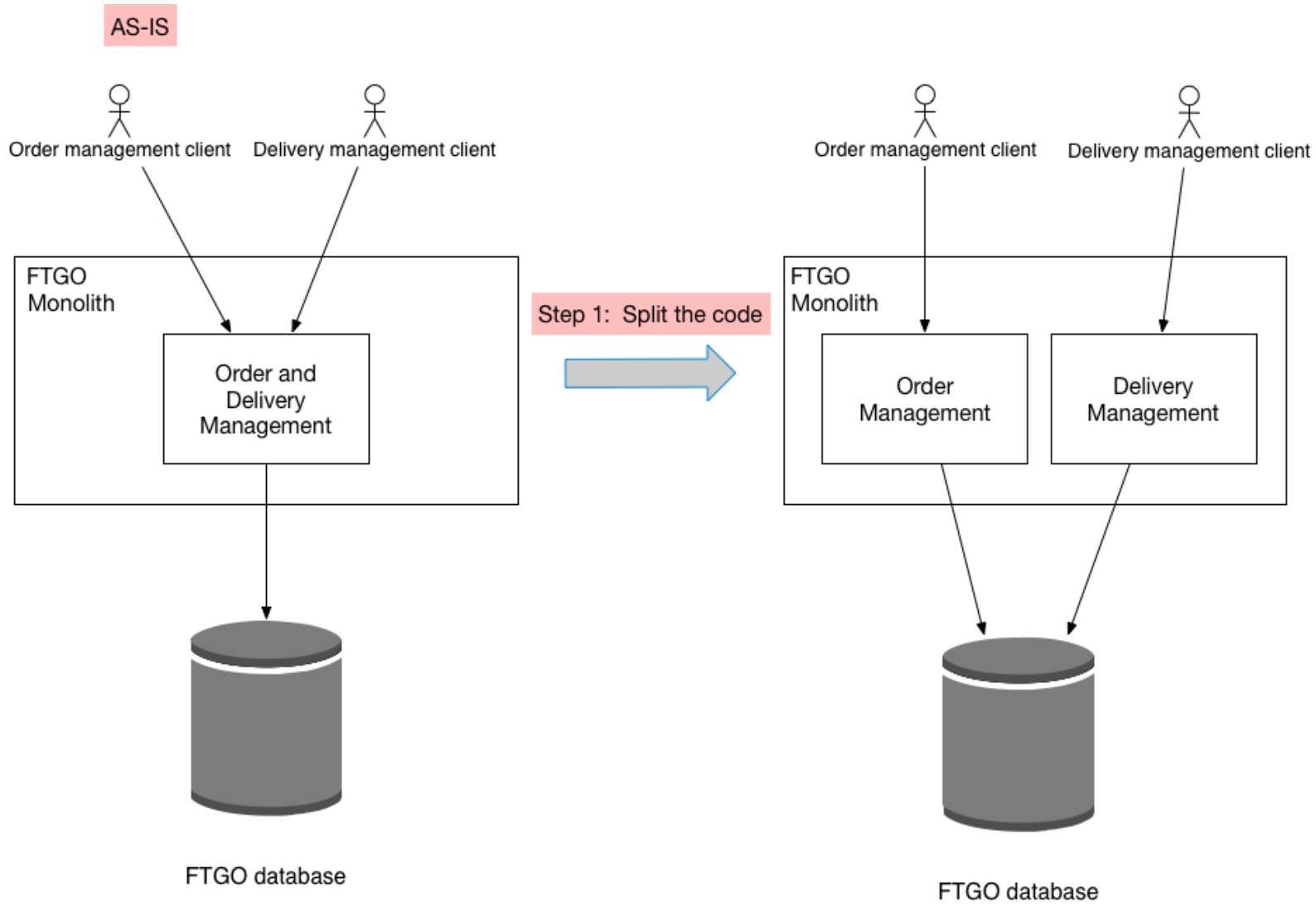
Delivery
Management logic



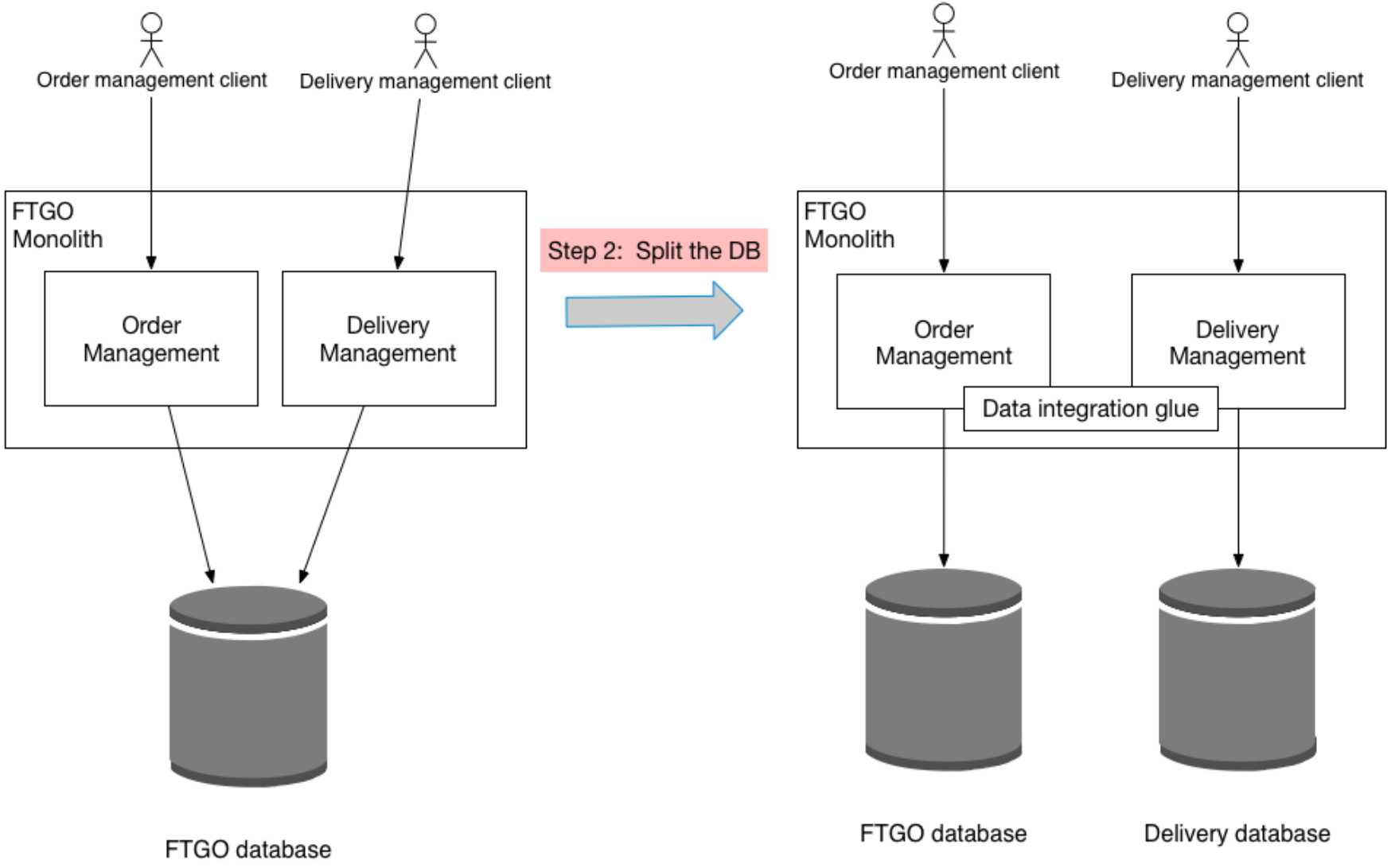
FTGO database
ORDERS table

ID	STATE	ASSIGNED_COURIER_ID	…
…	…	…	…

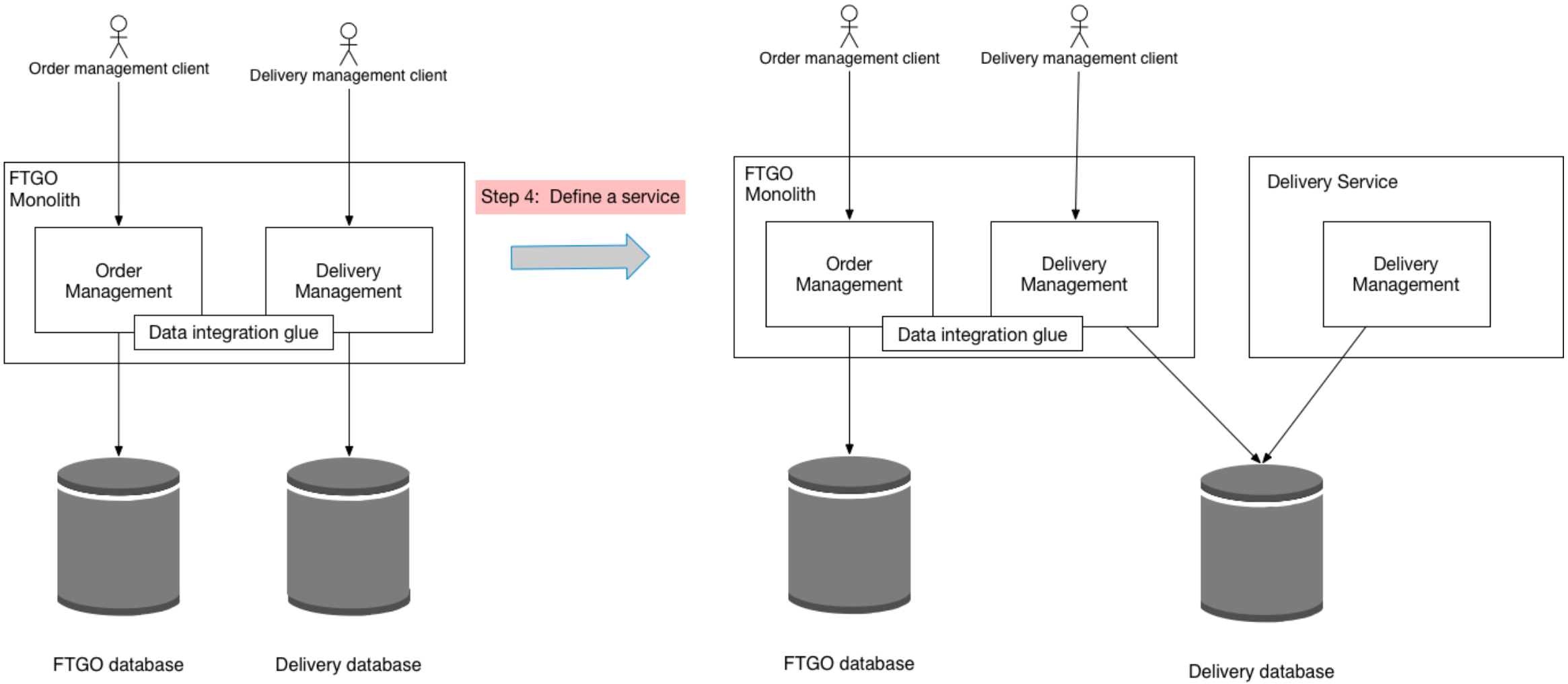
Шаг 2. Разделение кода



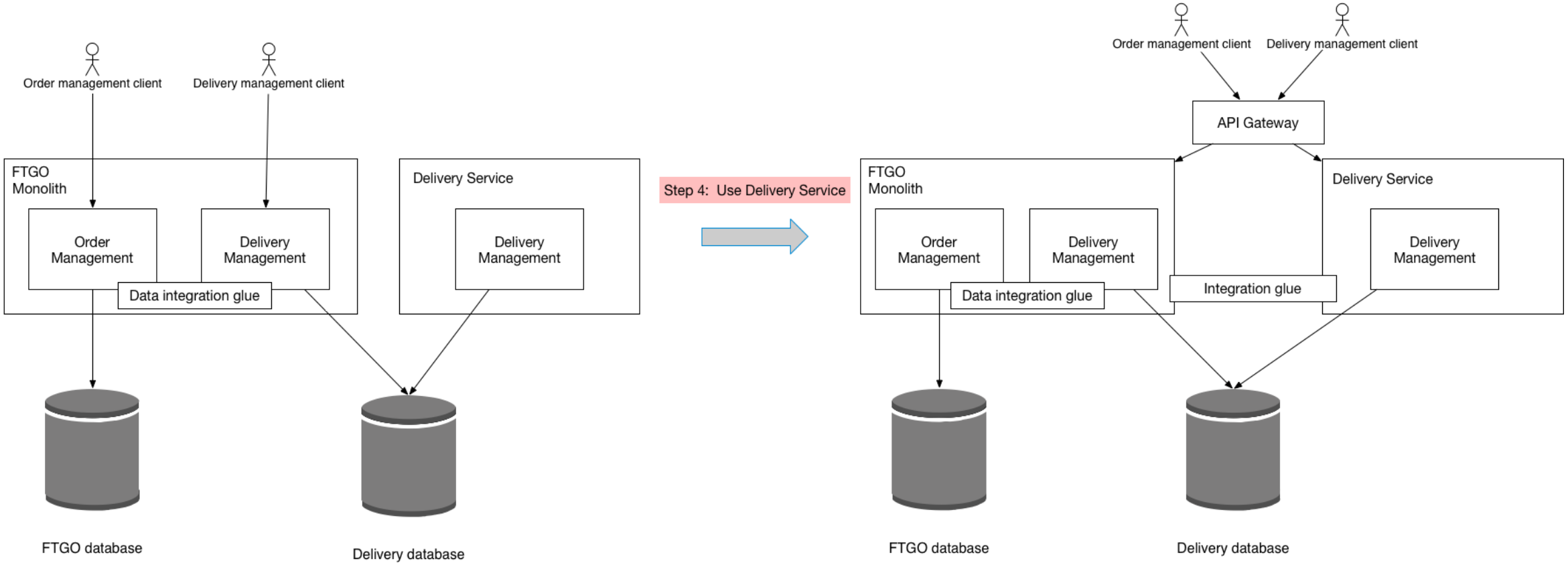
Шаг 3. Разделение базы данных



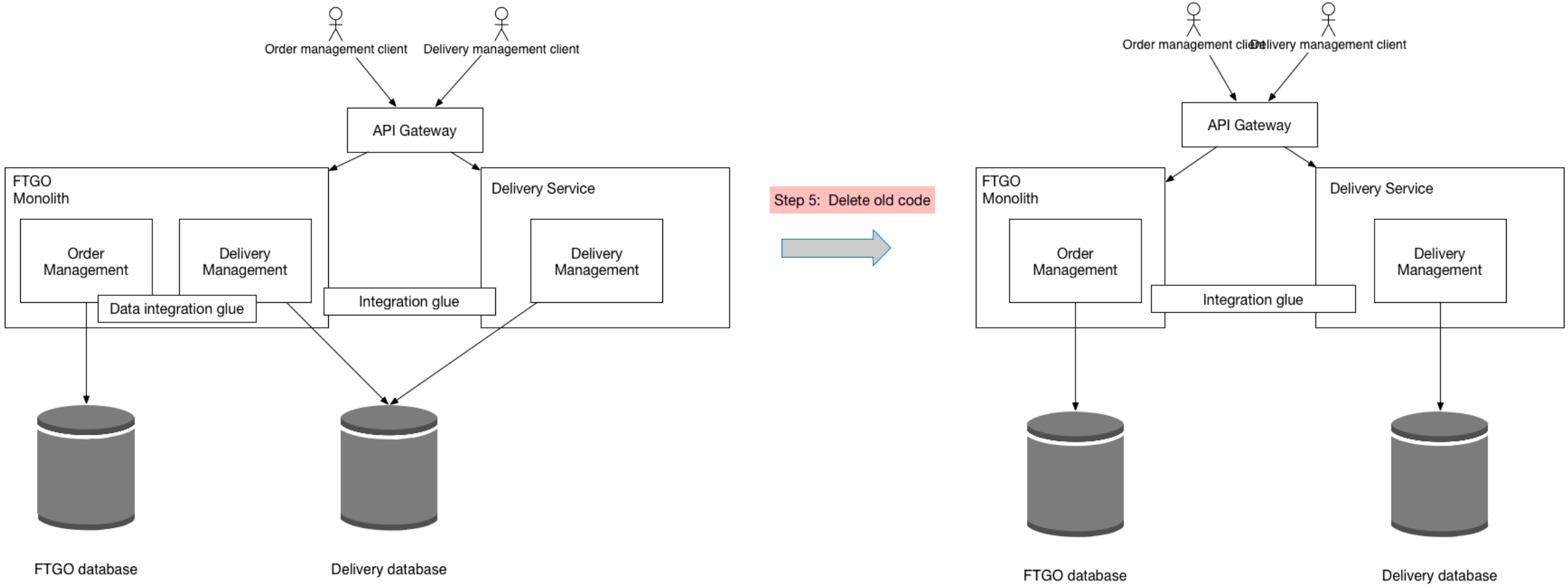
Шаг 4. Выделение сервиса



Шаг 5. Использование выделенного сервиса

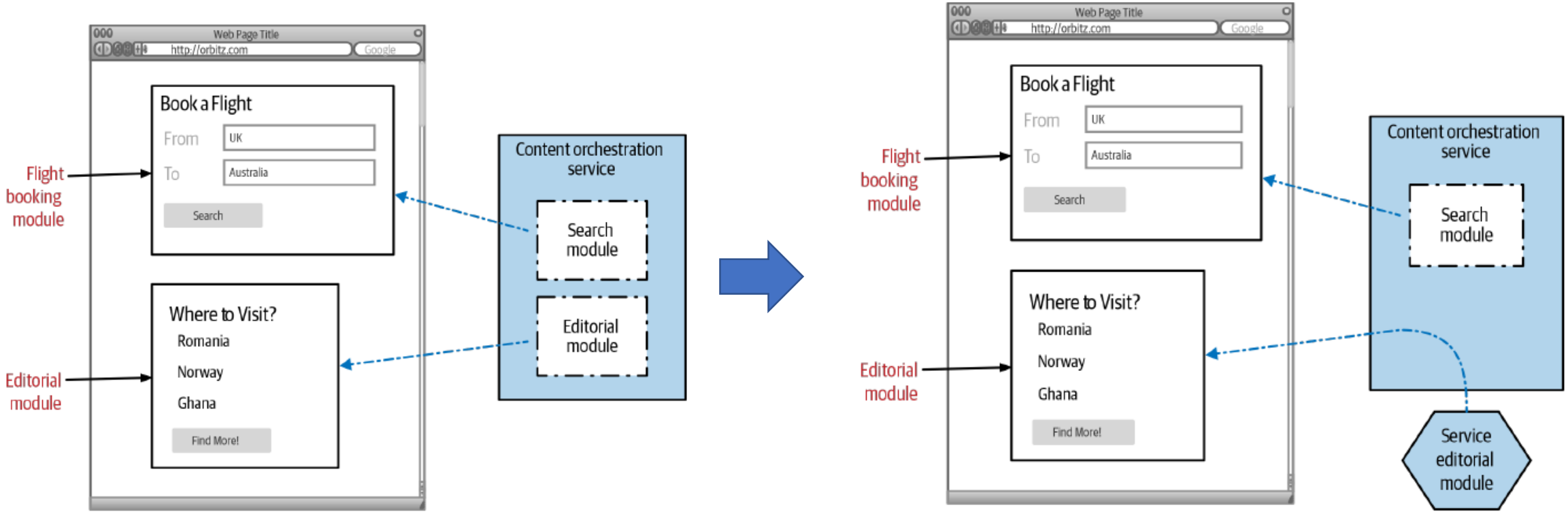


Шаг 6. Удаление выделенного функционала из монолита



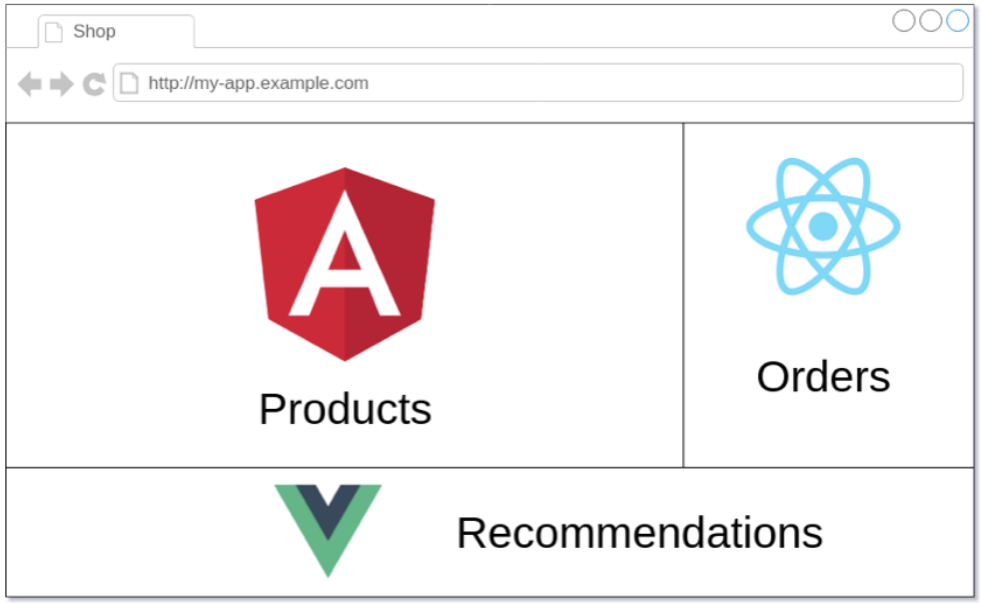
UI компоновка (UI Composition)

- Компоновку микросервисов осуществляем на UI
- Две техники:
  - Постраничная компоновка
  - Компоновка на виджетах



Микрофронтенд (Micro Frontends)

- **Дает возможность объединить в одном приложении разные виджеты или страницы, написанные разными командами с использованием разных фреймворков.**



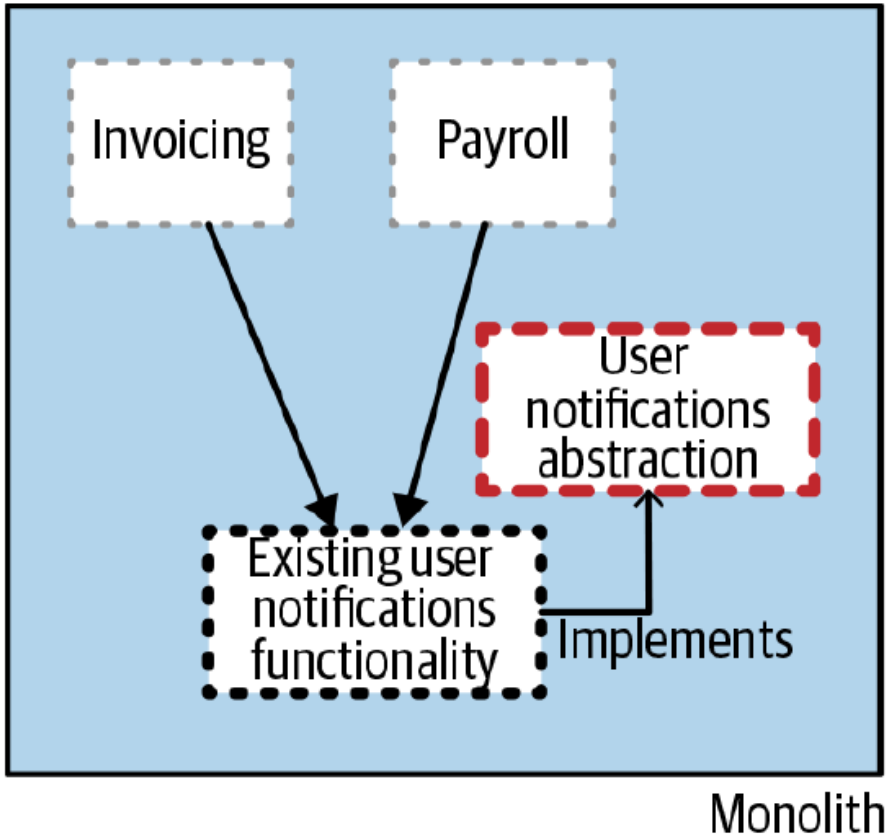
Отделение через абстракцию (Branch by Abstraction)

Отгораживаем разработчиков монолита от изменений вносимых рефакторингом, выполняя следующие шаги:

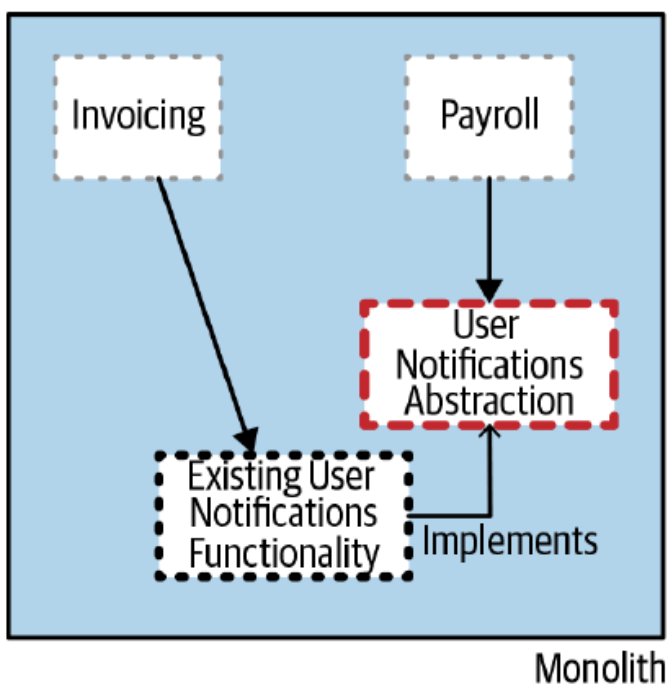
1. **Создайте абстракцию для заменяемой функциональности.**
2. **Переключите клиентов, использующих эту функциональность на абстракцию.**
3. **Создайте новую реализацию абстракции с переработанным функционалом.**

   В нашем случае эта новая реализация вызовет наш новый микросервис.
4. **Переключите абстракцию, чтобы использовать нашу новую реализацию.**
5. **Очистите абстракцию и удалите старую реализацию**

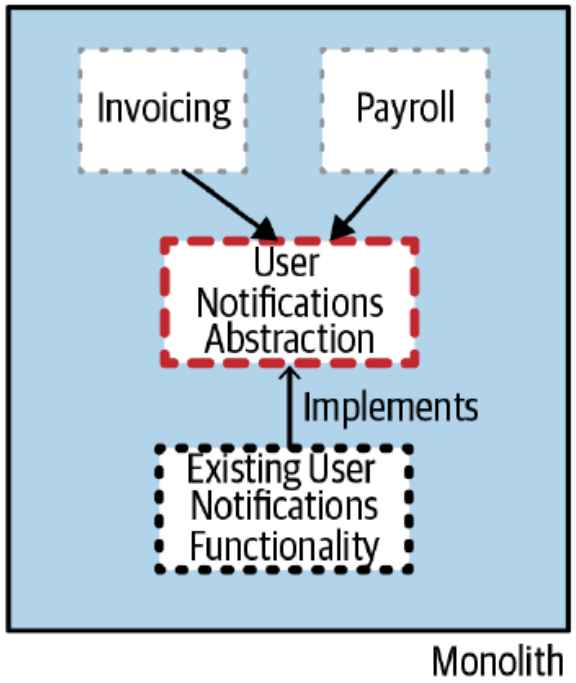
Шаг 1. Создайте абстракцию



Шаг 2. Переключите клиенты

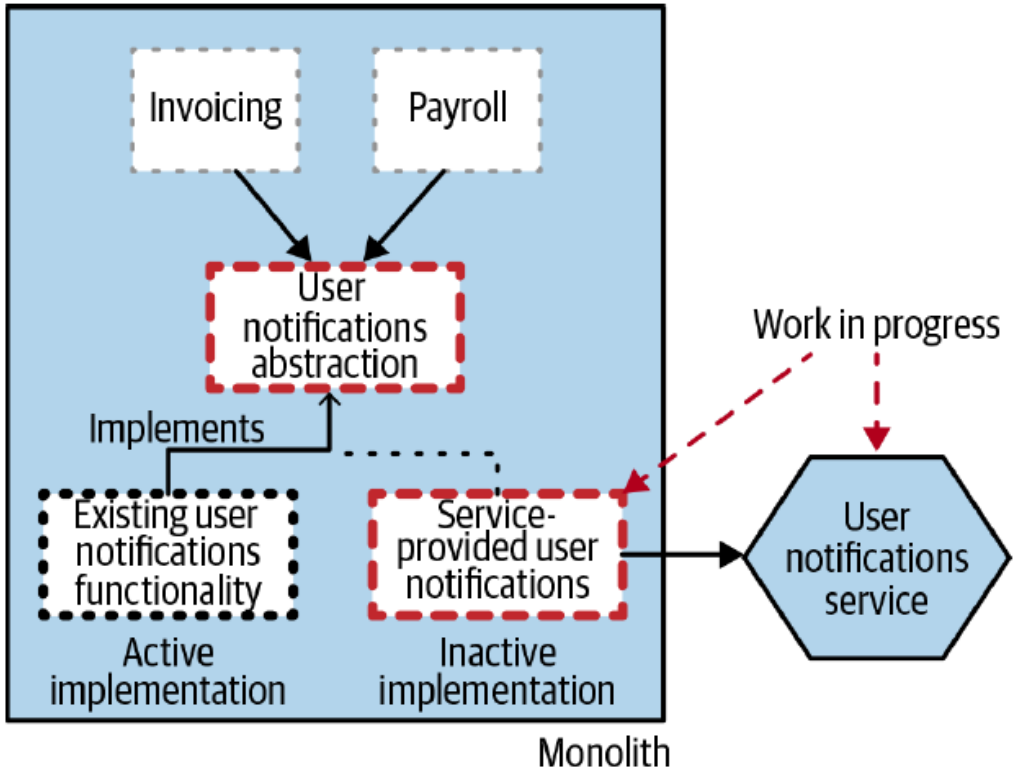


Change Payroll to use the new abstraction

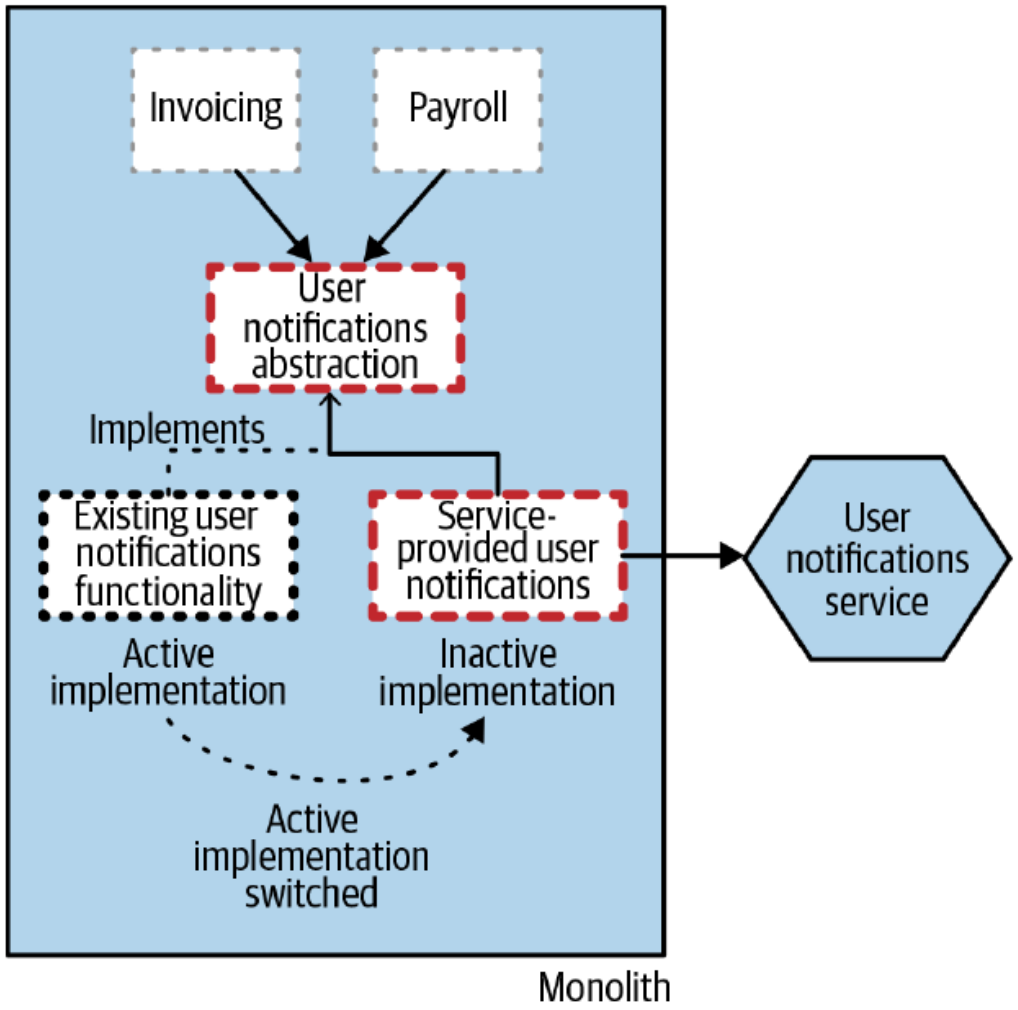


Change Invoicing to use the new abstraction

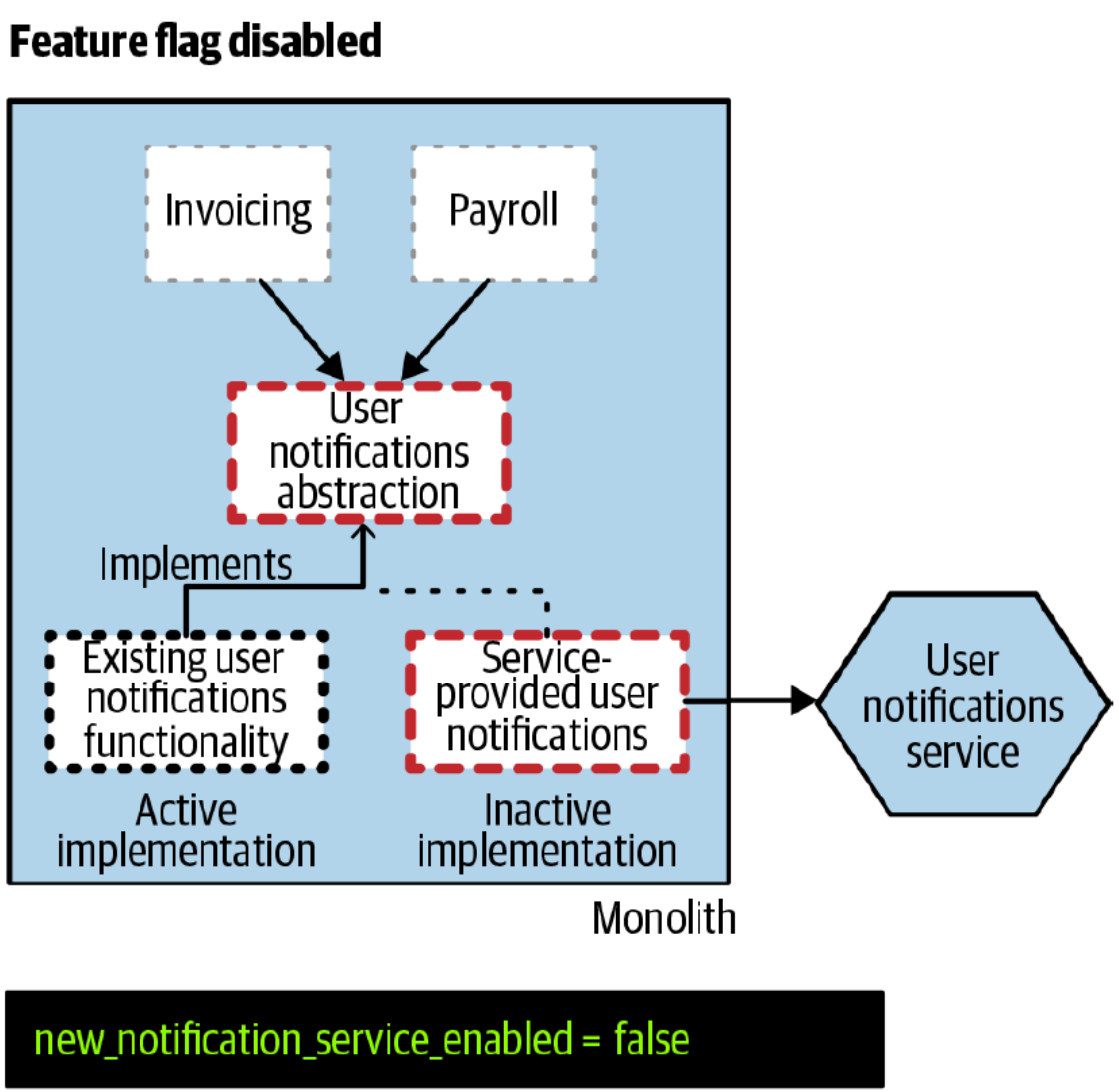
Шаг 3. Создайте новую реализацию

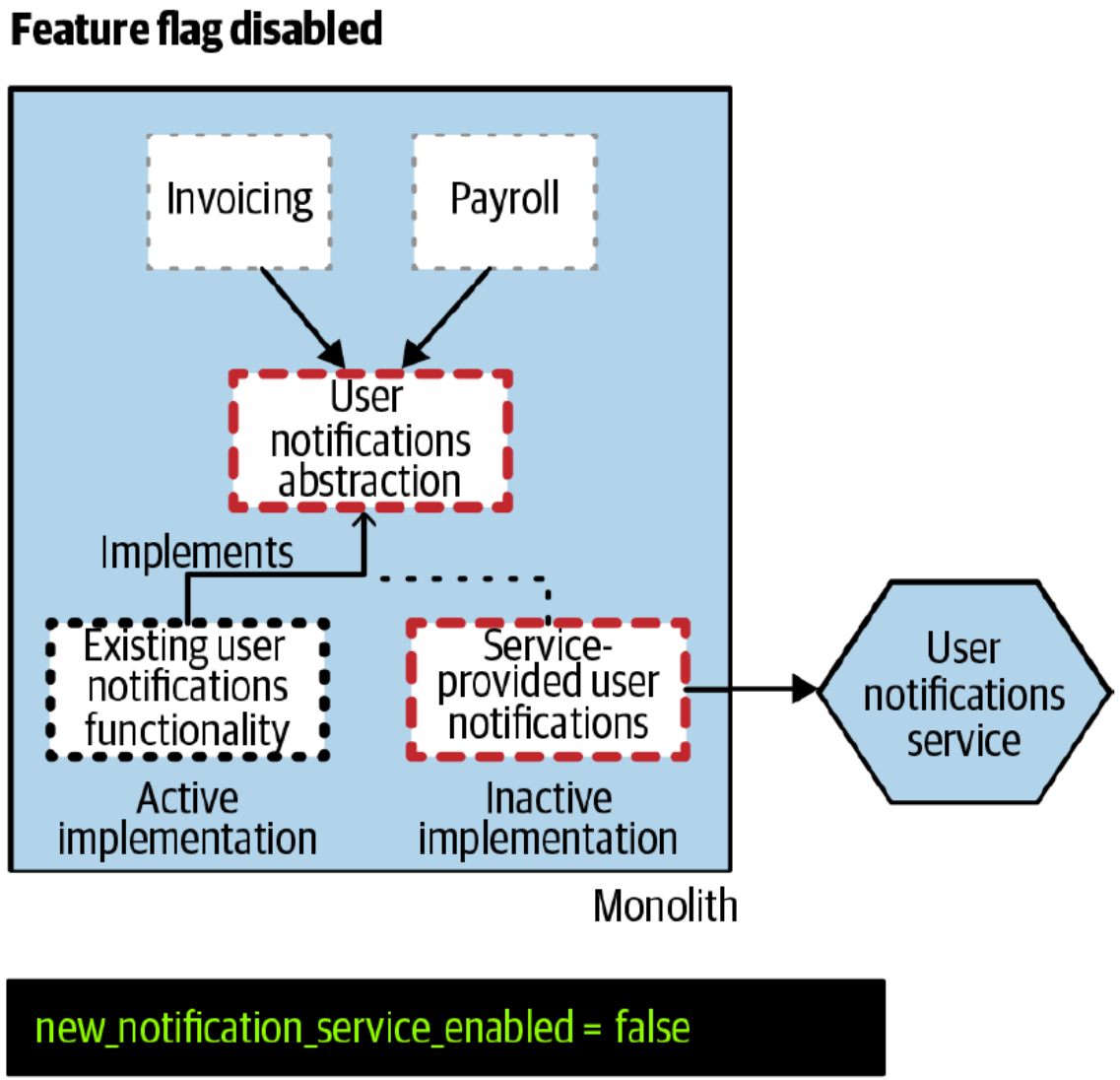


Шаг 4. Переключите абстракцию

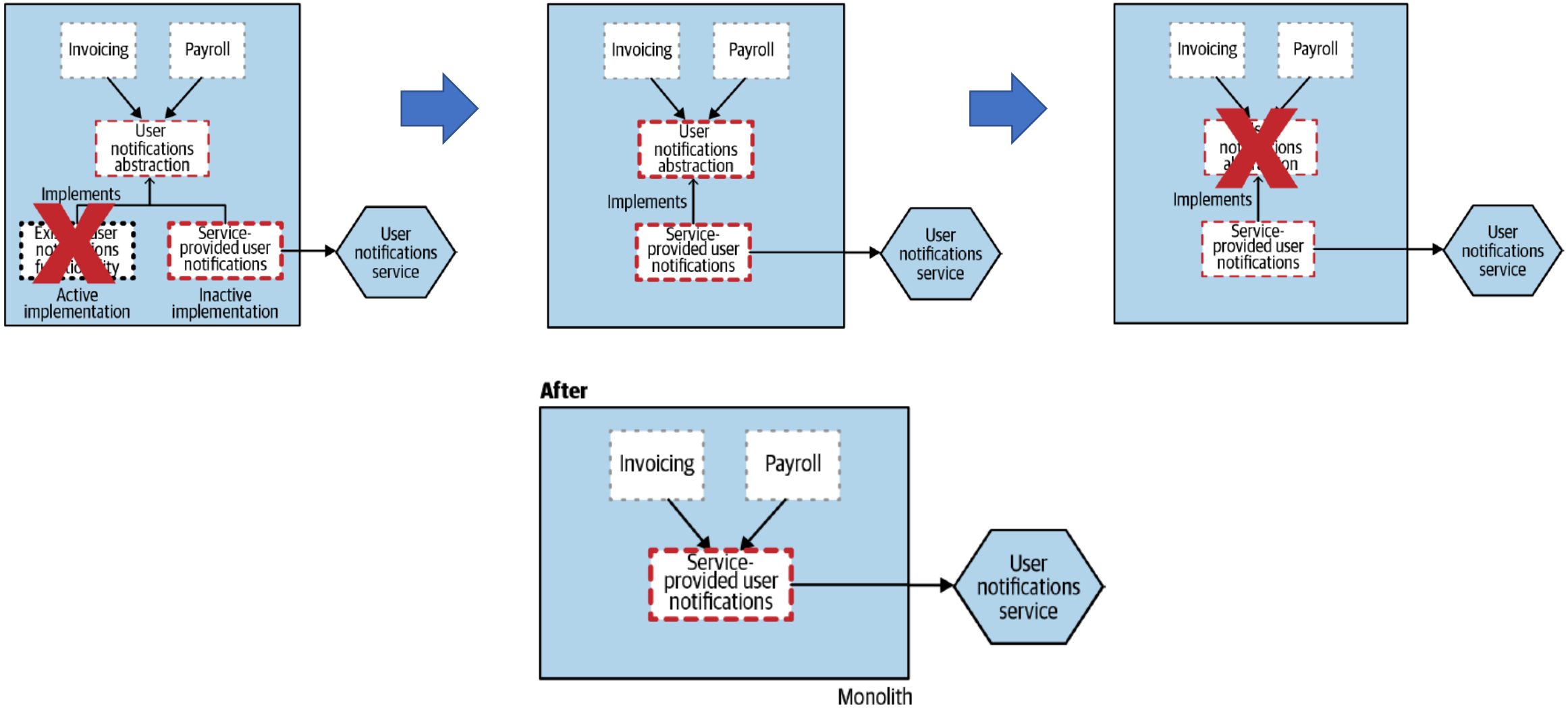


Использование feature flag для переключения



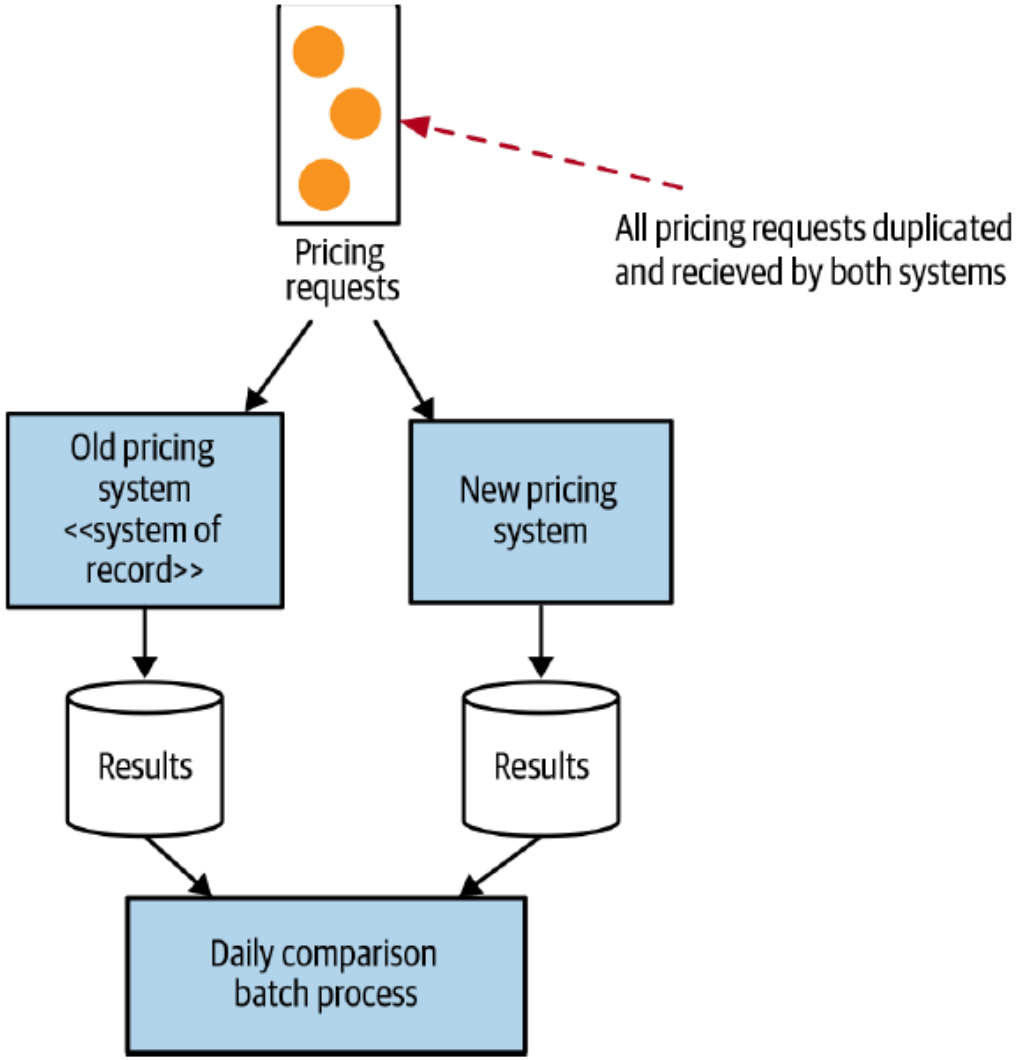


Шаг 5. Очистка



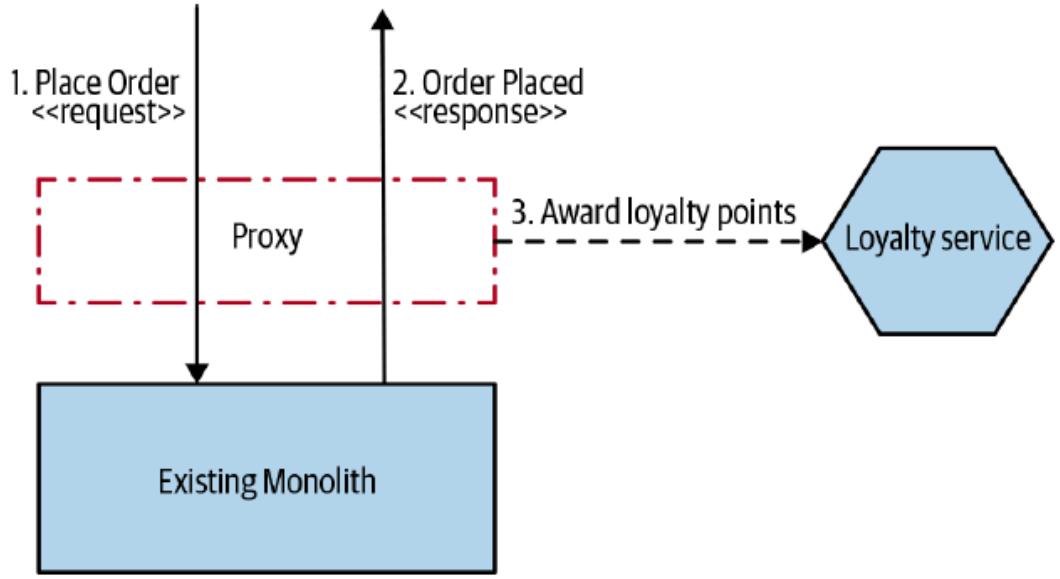
Параллельная работа (Parallel Run)

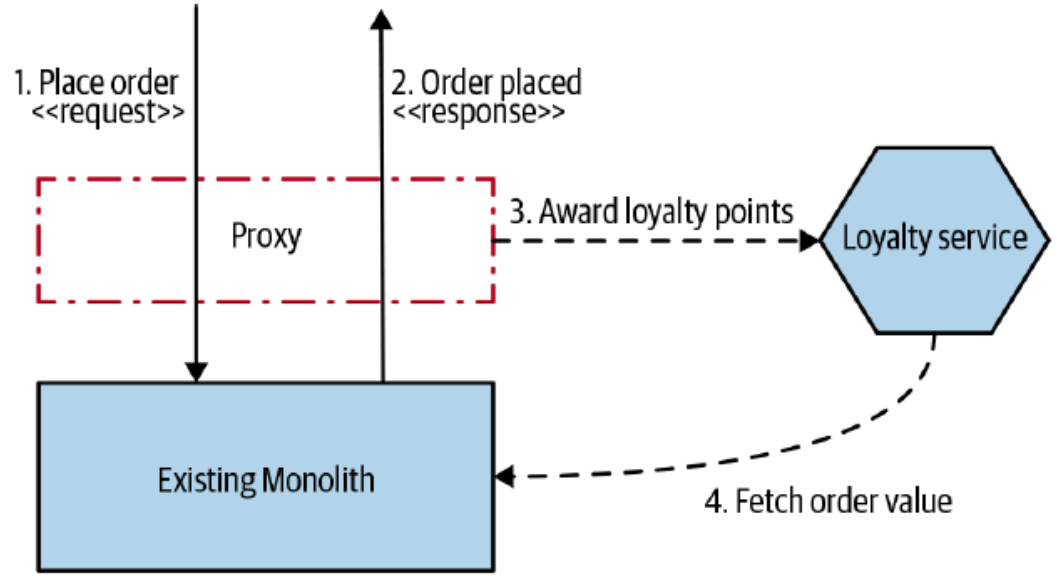
- **Запускаем обе версии одновременно и сравниваем результаты их работы**



Декорация участника (Decorating Collaborator)

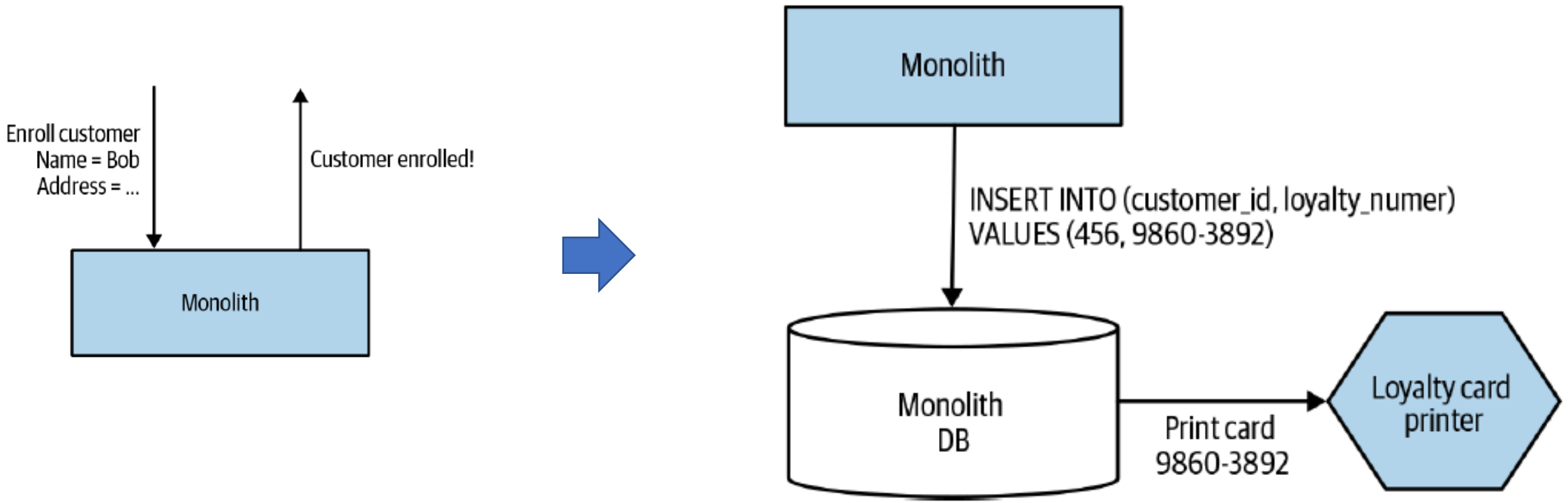
- **Декорируем поведение монолита новой функциональностью от микросервиса**





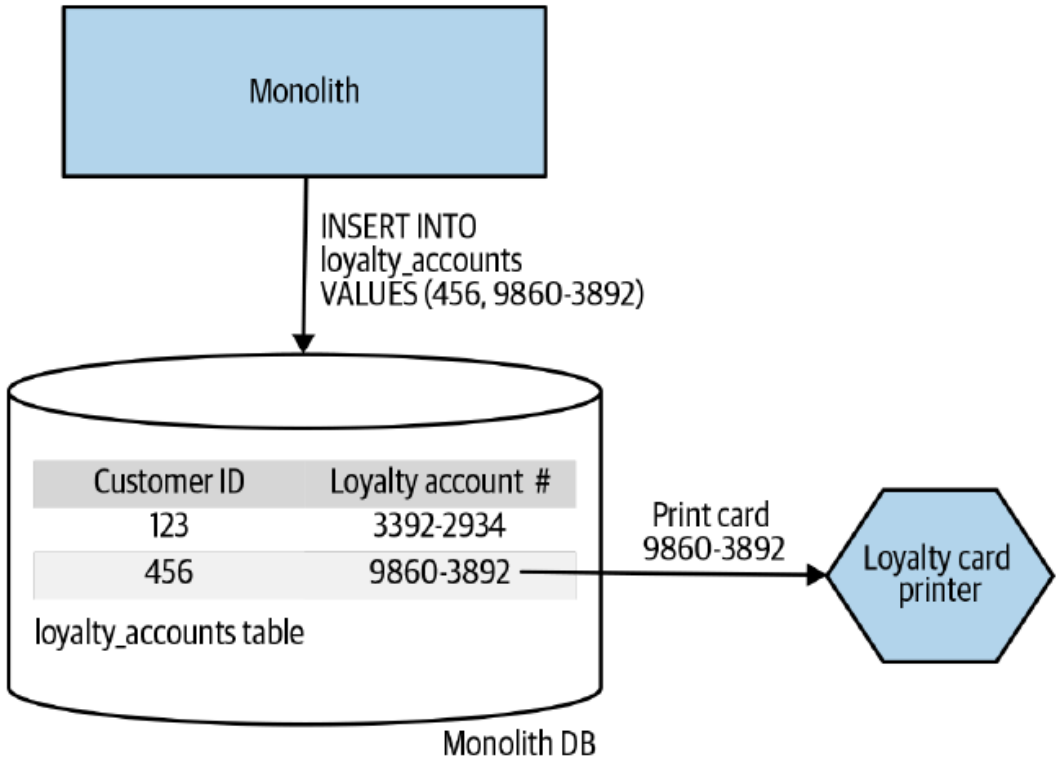
Захват изменения данных (Change Data Capture)

- **Используем механизм CDC для вызова нового сервиса**

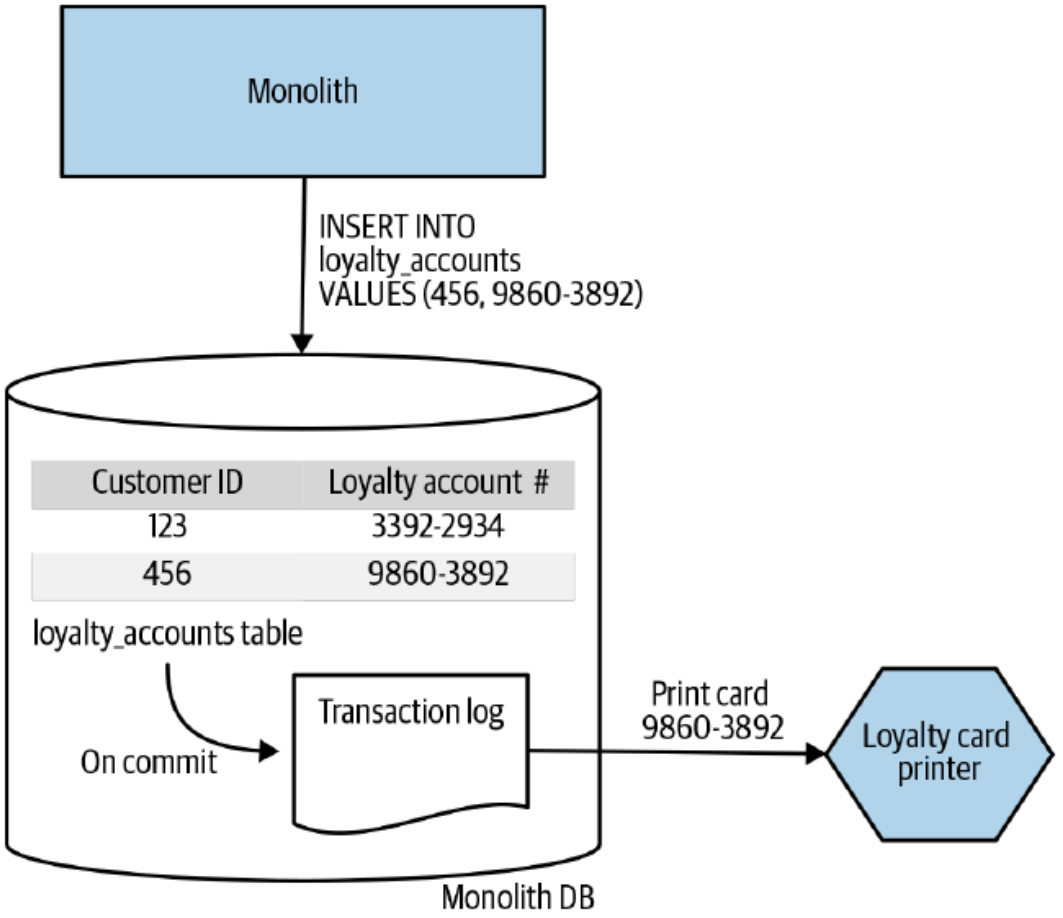


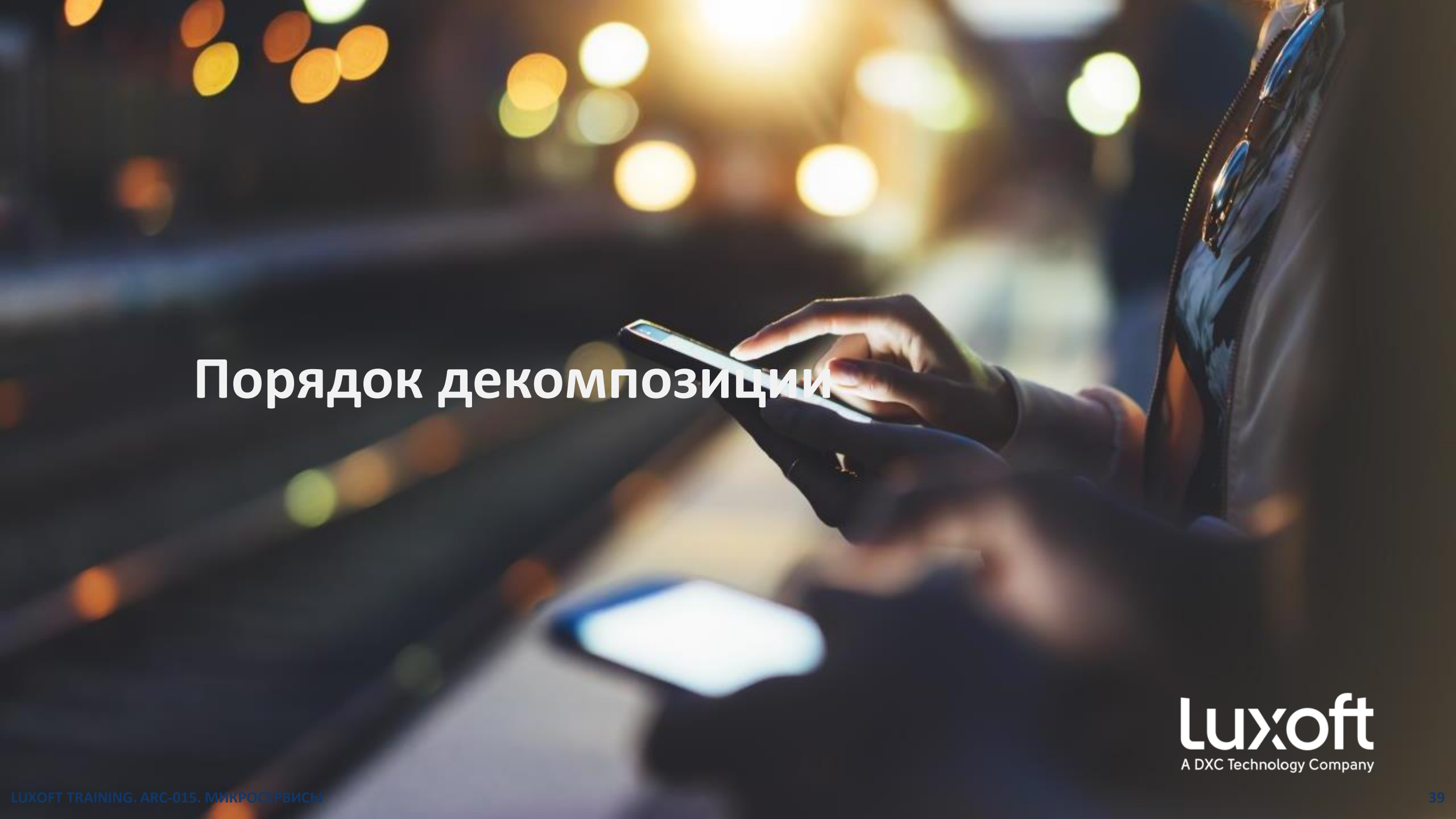
Реализации CDC

Триггер



Отслеживание журнала транзакций



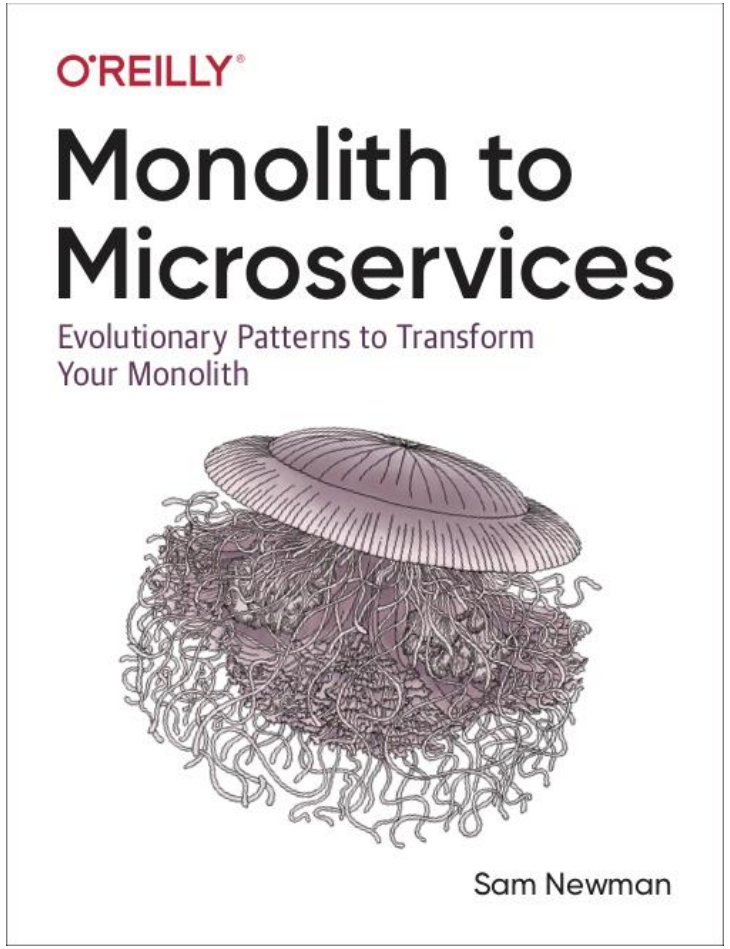


Порядок декомпозиции

luxoft
A DXC Technology Company

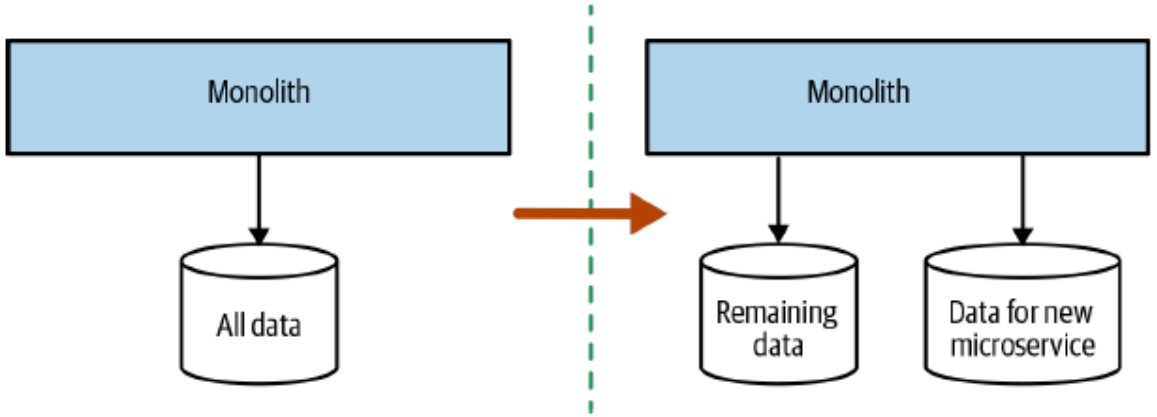
Порядок декомпозиции

- **Вначале данные**
  - **Паттерн «Репозиторий на контекст» (Repository per bounded context)**
  - **Паттерн «База данных на контекст» (Database per bounded context)**
- **Вначале код**
  - **Паттерн «Монолит как слой доступа к данным» (Monolith as data access layer)**
  - **Паттерн «Хранеништель множества схем» (Multischema storage)**
- **Одновременно код и данные**

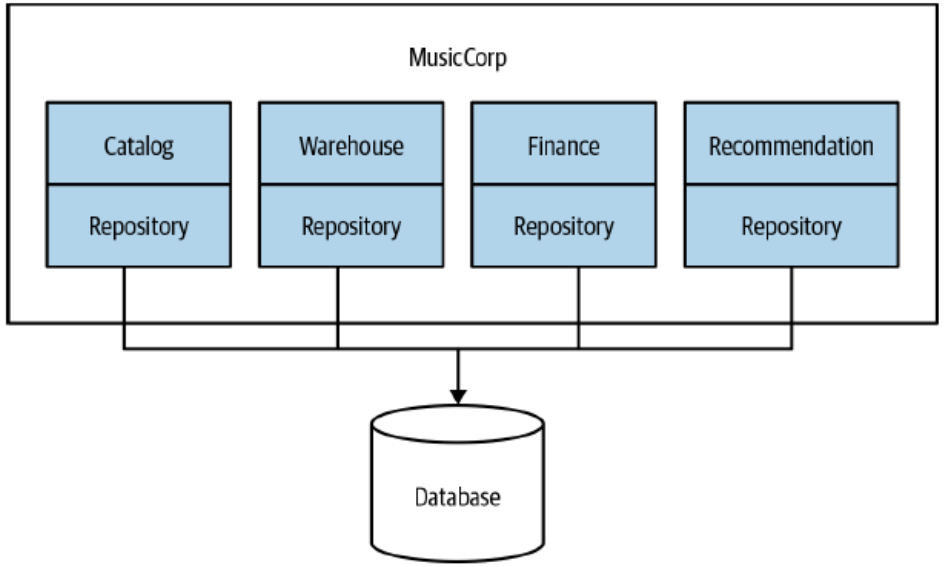


Вначале данные

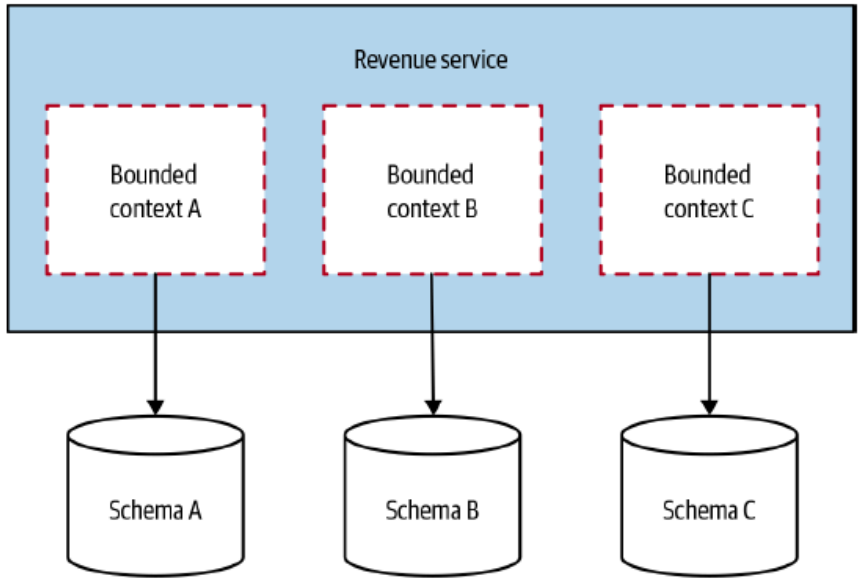
Если основные наши проблемы производительность и согласованность данных



Репозиторий на контекст

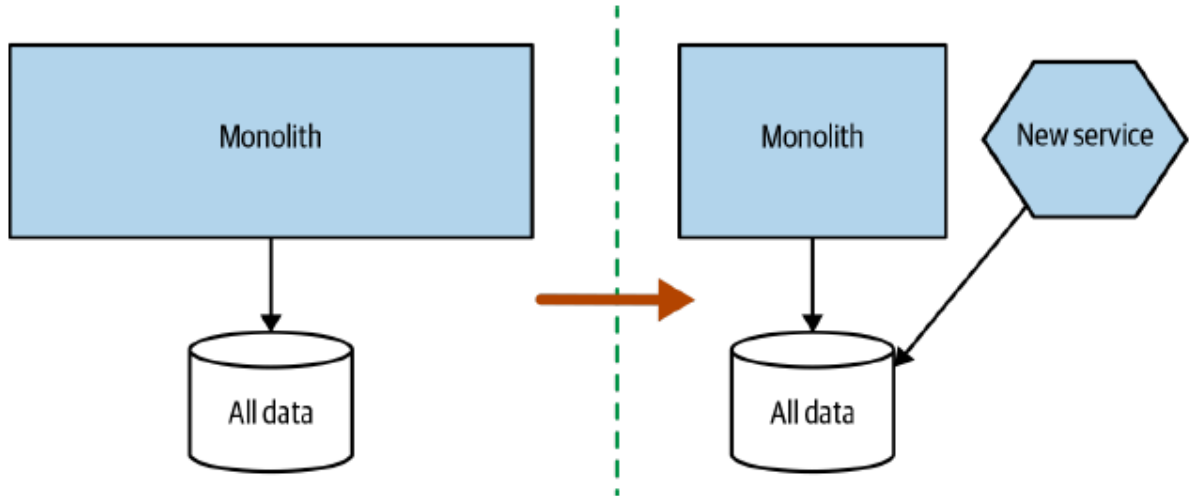


База данных на контекст

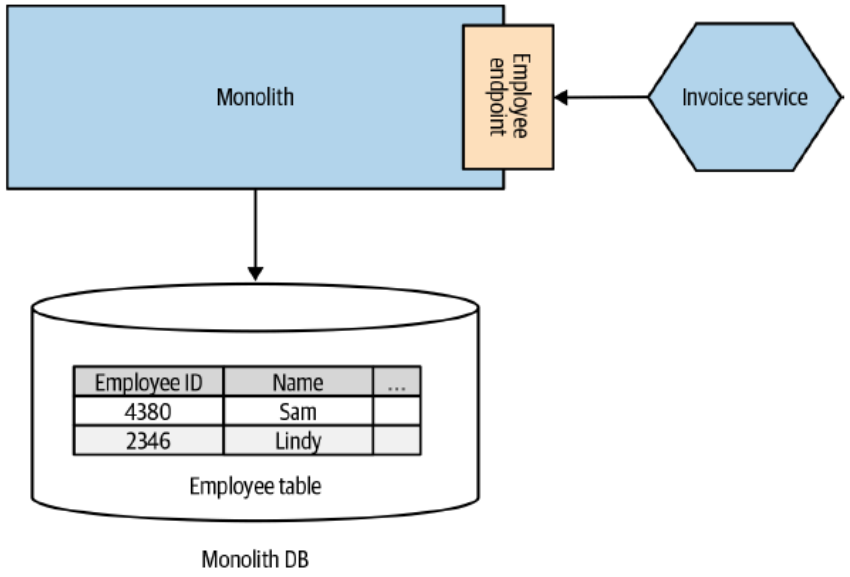


Вначале код

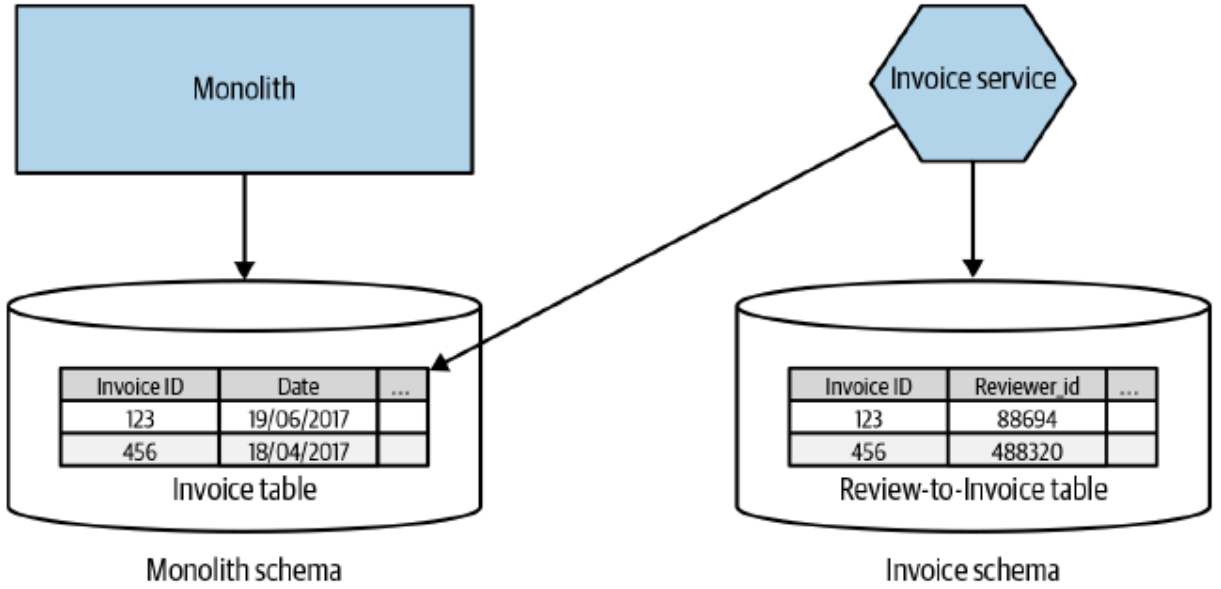
В противном предыдущему случае



Монолит как слой доступа к данным

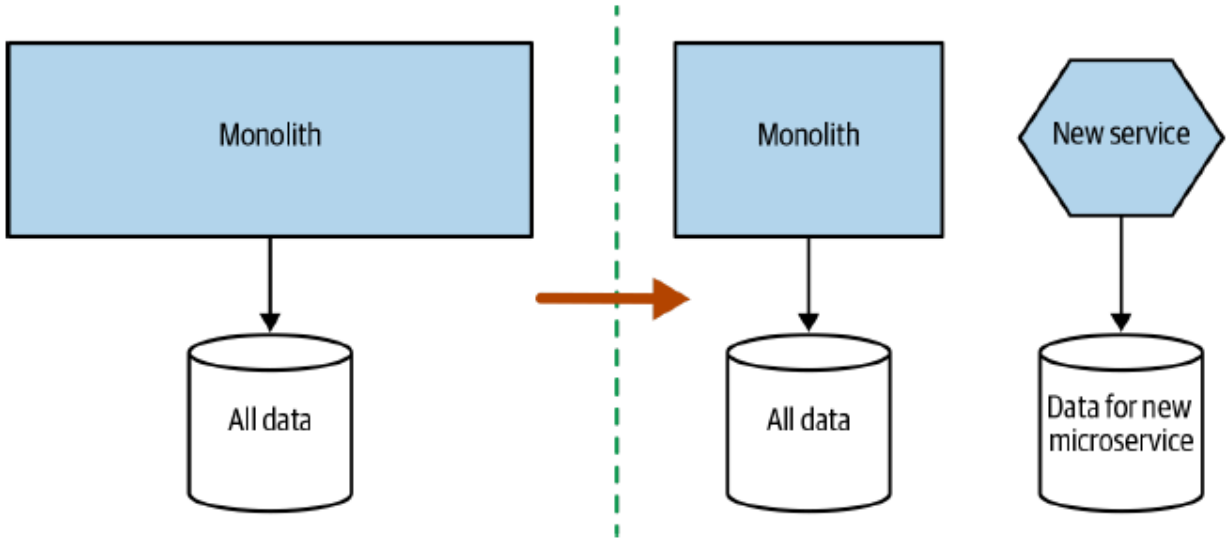


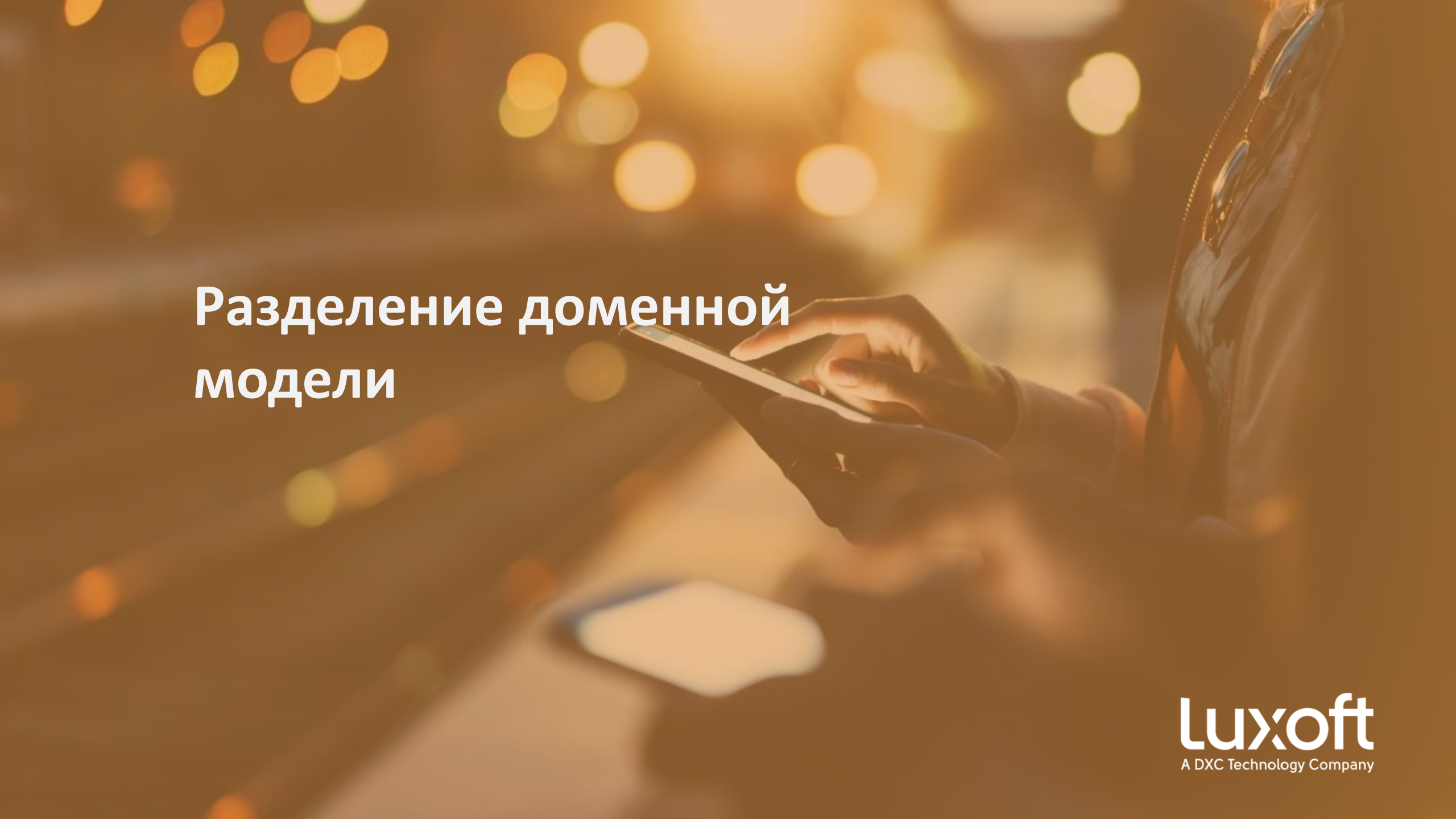
Хранеритель множества схем



Одновременно код и данные

Редкий вариант



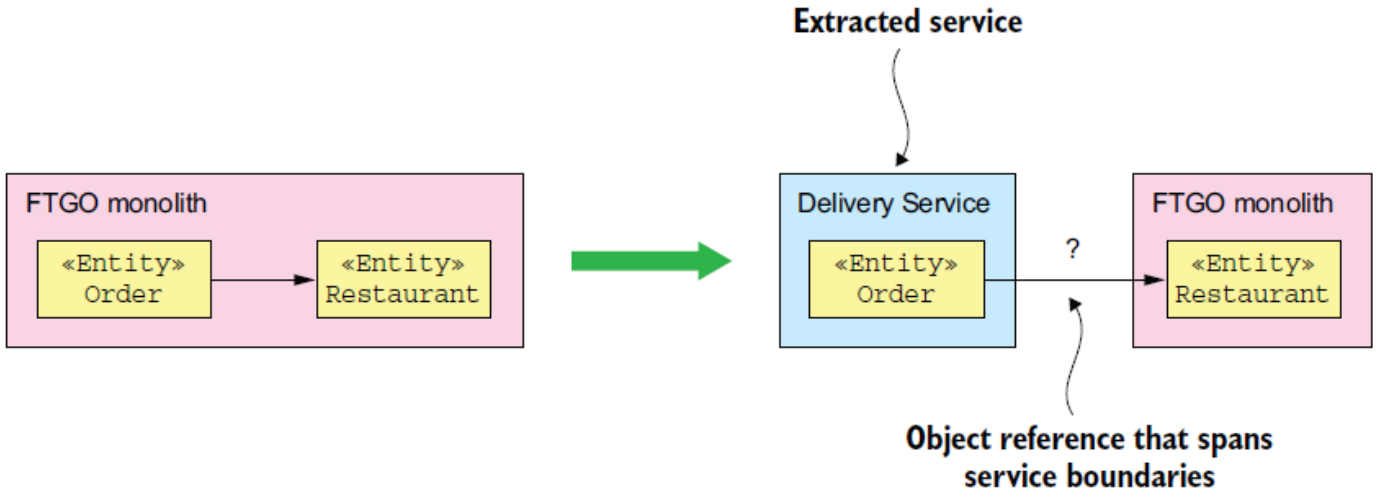


Разделение доменной модели

Luxoft
A DXC Technology Company

Разделение доменной модели: Проблемы

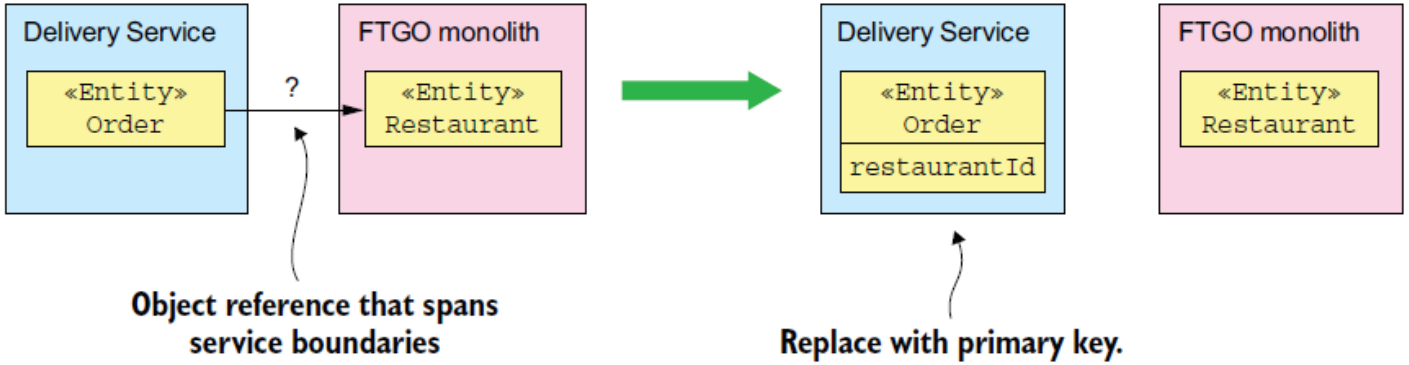
- Разделение функциональности в классах, имеющих очень много ответственностей (анти-паттерн «класс-бог»)
- Наличие объектных ссылок, которые могут выйти за границы сервиса



Разделение доменной модели: Решение

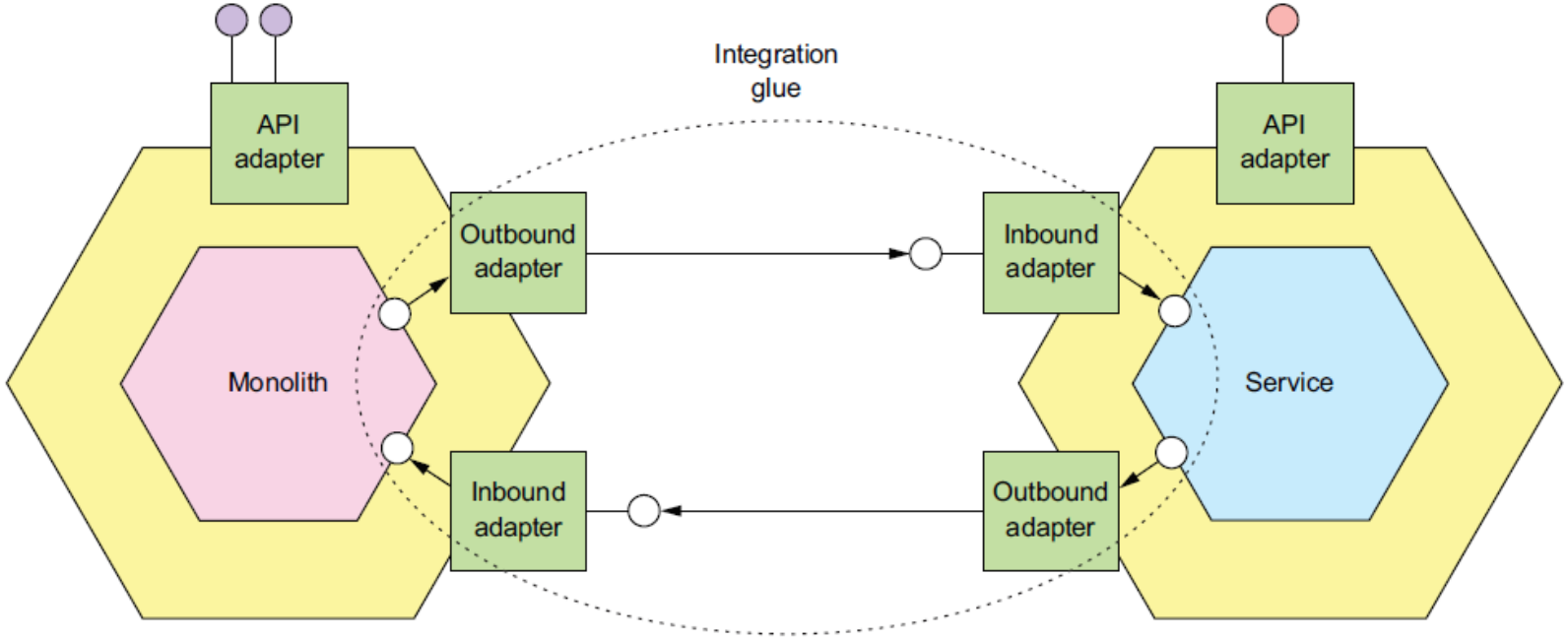
- **Объектные ссылки заменяются ключами (используем паттерн DDD «Агрегат»)**
  - **При этом возникает проблема для клиентов сервиса, работающих со ссылкой**

    Извлечение сервиса более трудоемкий процесс чем извлечение класса в ограниченном контексте



Взаимодействие сервиса и монолита: Проблема

- Сервисы редко бывают автономными
  - Иногда сервису нужно обратиться к монолиту
  - Иногда монолиту нужно обратиться к сервису
- При этом зачастую важно поддерживать согласованность между данными сервиса и монолита

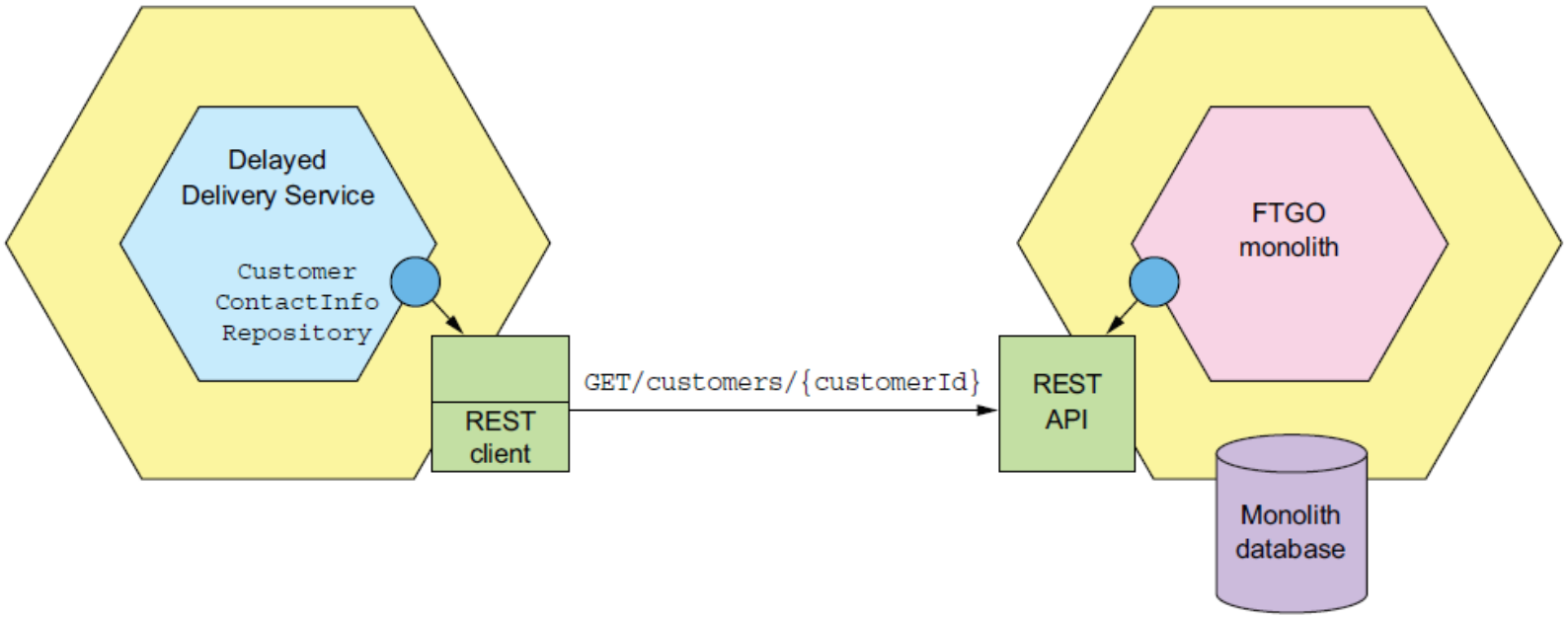


Проектирование интеграционного слоя

- Проектирование API интеграционного слоя
- Выбор стиля и механизма взаимодействия
  - Взаимодействие «запрос-ответ»
  - Проекция данных (CQRS)
  - Паттерн DDD «Предохранительный слой» (Anti-corruption layer, ACL)

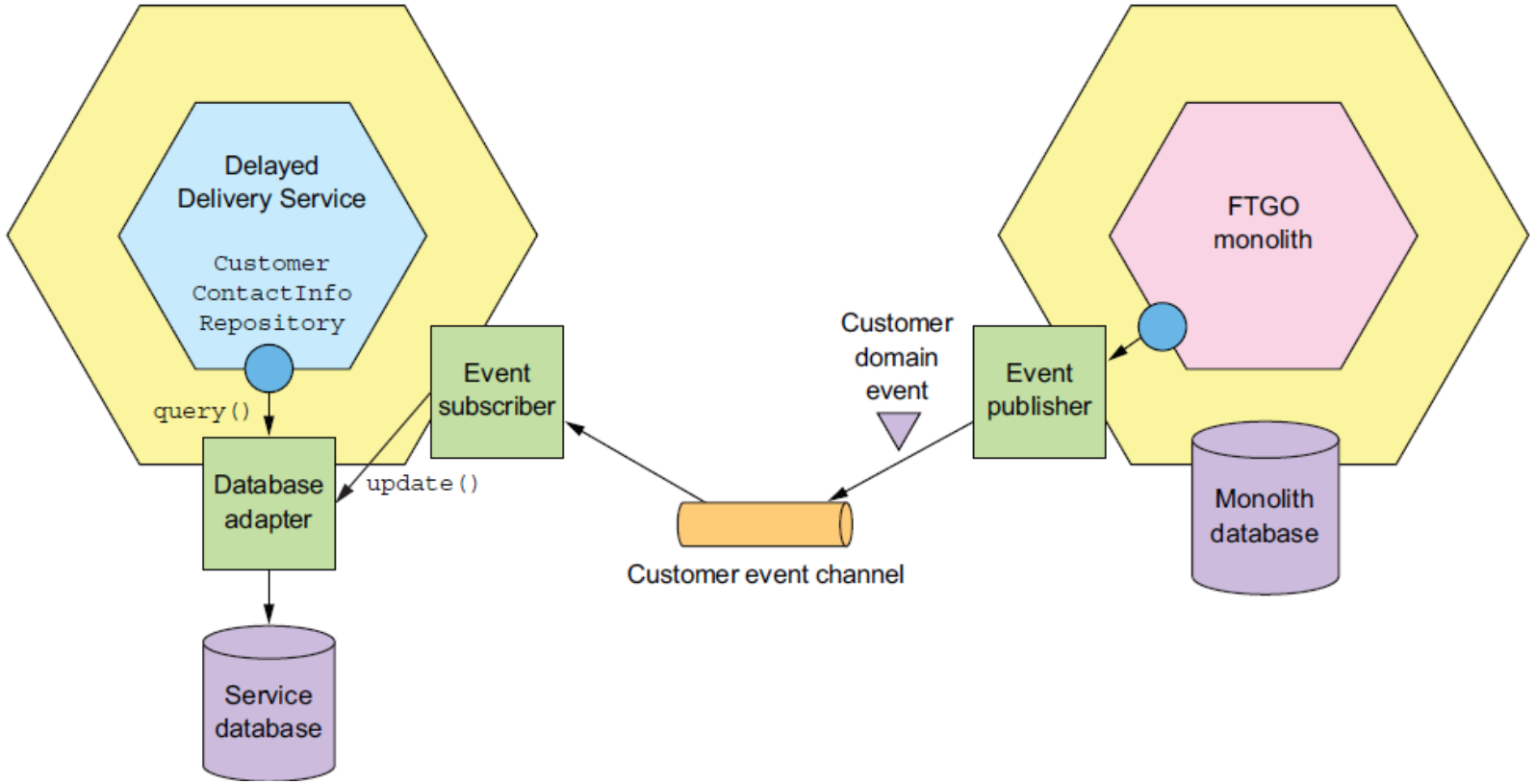
Взаимодействие «запрос-ответ»

- **Основное достоинство – простота**
- **Основной недостаток – неэффективность**



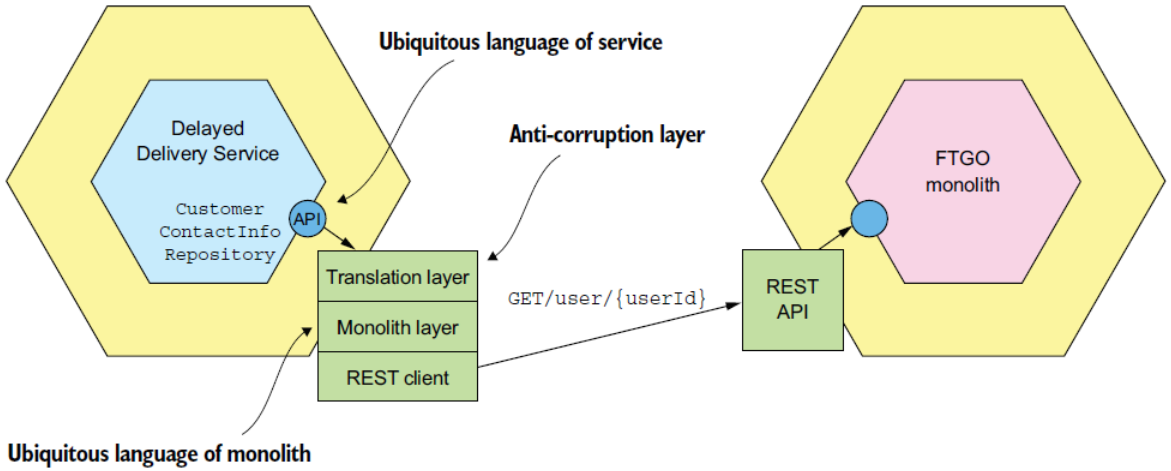
Проекция данных (CQRS)

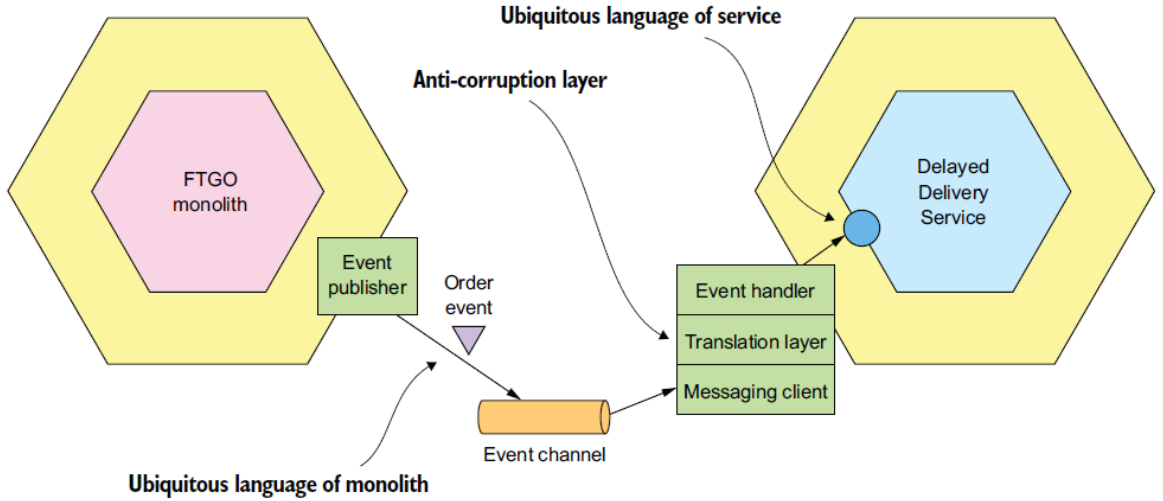
- **Эффективное решение**
- **Основной недостаток – сложно поддерживать реплику**



Предохранительный слой (ACL)

Программный слой, выступающий посредником между двумя доменными моделями, не позволяющий им засорить друг друга своими концептами





Как монолит публикует и подписывается на события

- **Стратегии публикации событий**
  - **Найти все участки кода, связанные с изменениями, и вставить туда вызовы API**

    Не всегда просто найти все участки, тем более что они могут находится в хранимых процедурах
  - **Публикация доменных событий на уровне базы данных.**

    Не нужно модифицировать монолит

    Не всегда ясна причина изменения данных и трудно определить событие
- **Проблема потребления событий**
  - **Обычно подписаться на события в монолите бывает достаточно просто, но случается так, что язык, на котором написан монолит, не имеет клиента для нужного брокера**
    - **В этом случае пишется дополнительное приложение**

Обеспечение согласованности данных между сервисом и монолитом

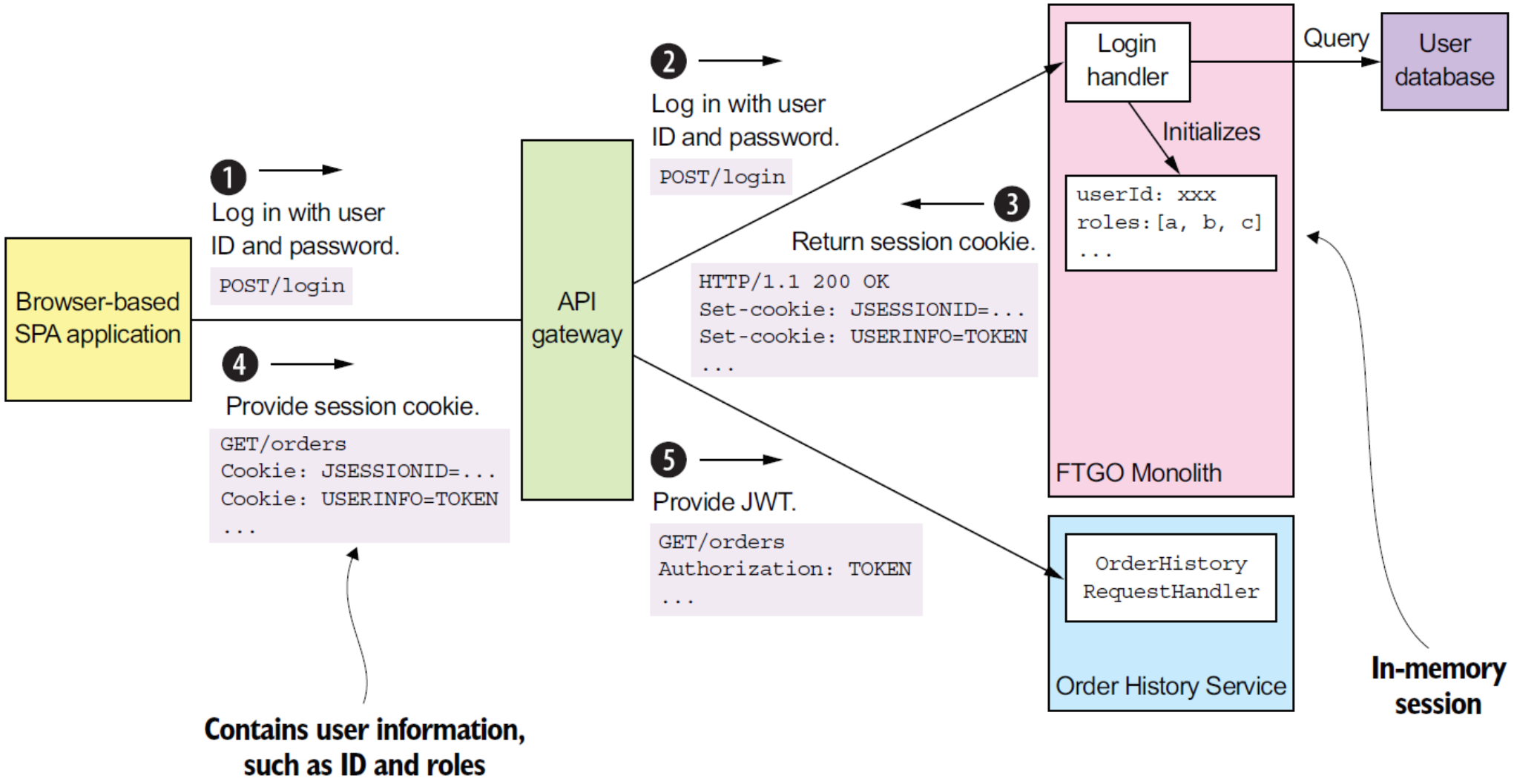
- Для поддержания согласованности данных в микросервисных архитектурах используется паттерн «Сага»
- Проблемы паттерна в монолите:
  - Трудности изменения монолита для поддержки компенсируемых транзакций

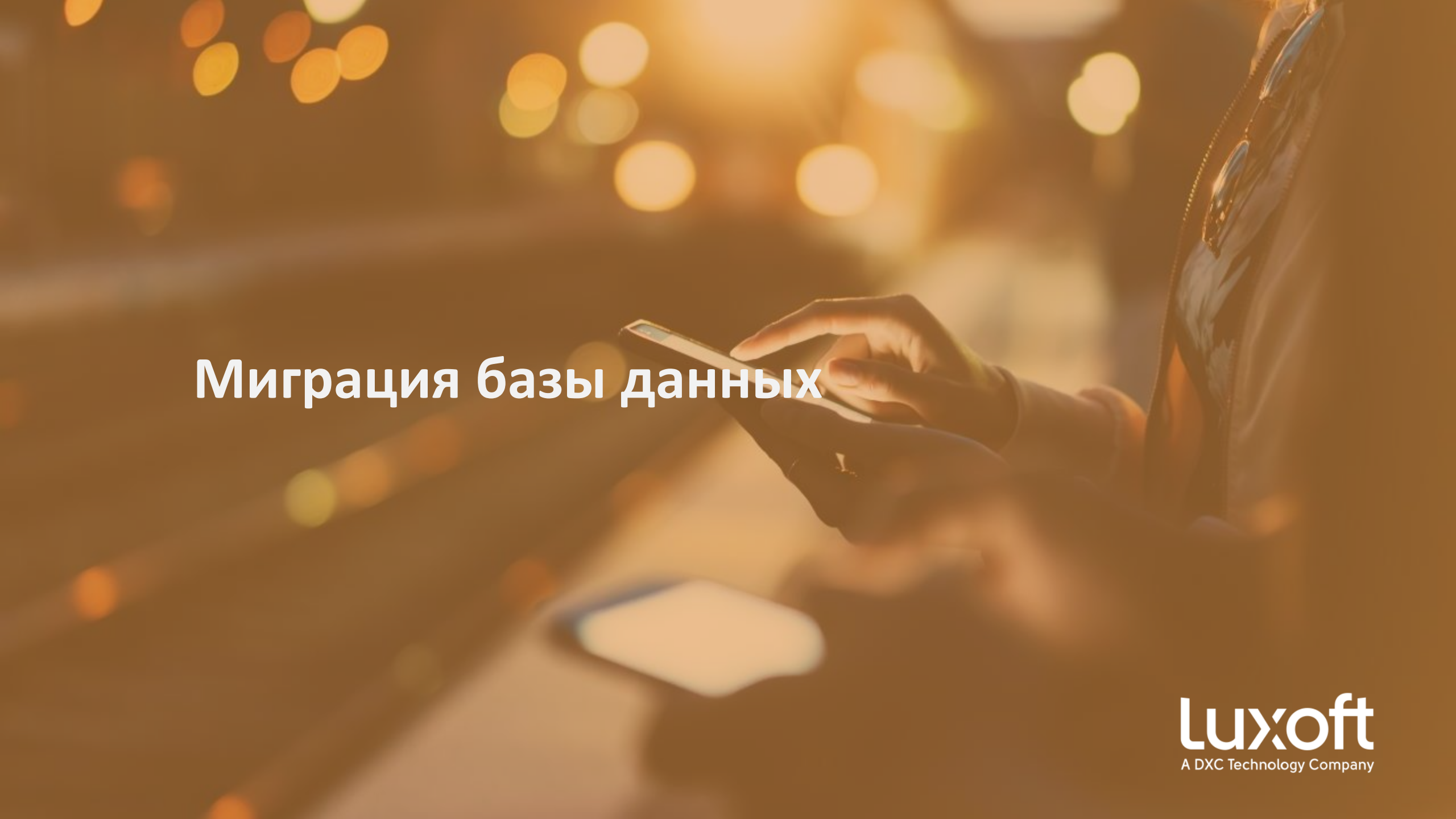
    Для каждого действия в рамках разделяемой транзакции должно быть написано компенсирующее действие, что сложно в масштабах монолита
    - Если все транзакции в монолите являются либо поворотными, либо повторяемыми, то писать компенсирующие транзакции не нужно
    - Тщательно планируем порядок операций в распределенных транзакциях

Аутентификация и авторизация: Проблема

- Требуется одновременная поддержка двух механизмов
- Адаптация монолита
  - Обработчик входа в систему возвращает дополнительный cookie-файл, который содержит информацию о пользователе, такую как его ID и роли
  - Браузер включает этот cookie в каждый запрос
  - API-шлюз извлекает содержимое cookie и добавляет его в HTTP-запрос, направленный к сервису
  - В итоге каждый сервис получает доступ к необходимой пользовательской информации

Адаптация механизма безопасности монолита





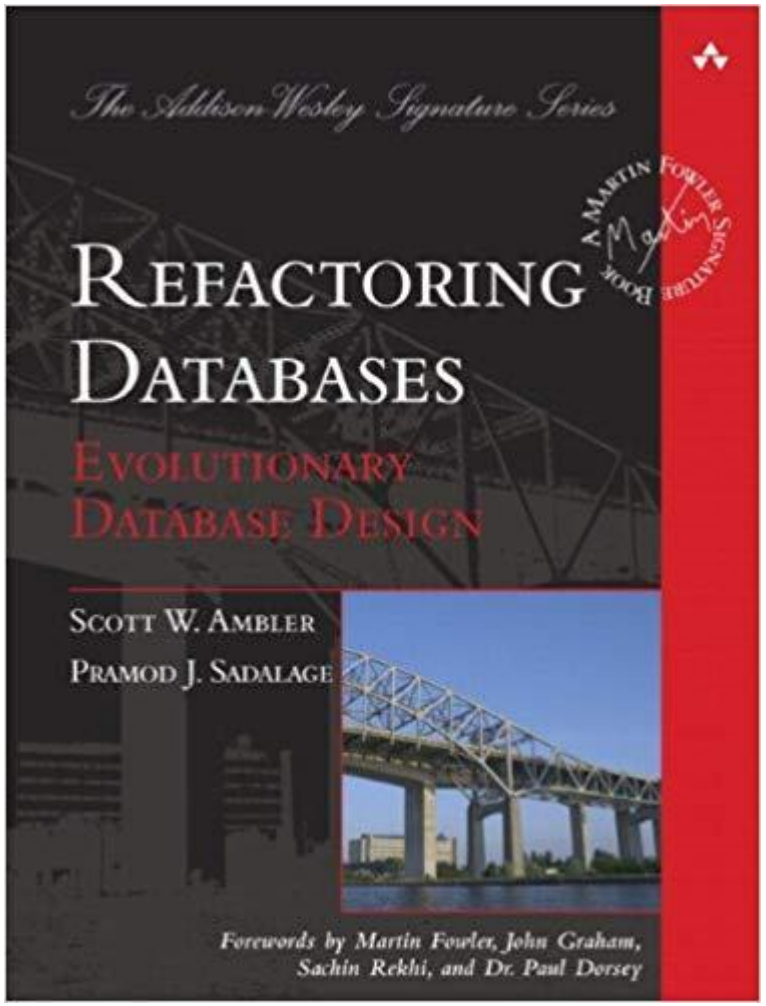
Миграция базы данных

Luxoft
A DXC Technology Company

Рекомендуемая литература

- **Refactoring Databases: Evolutionary Database Design**

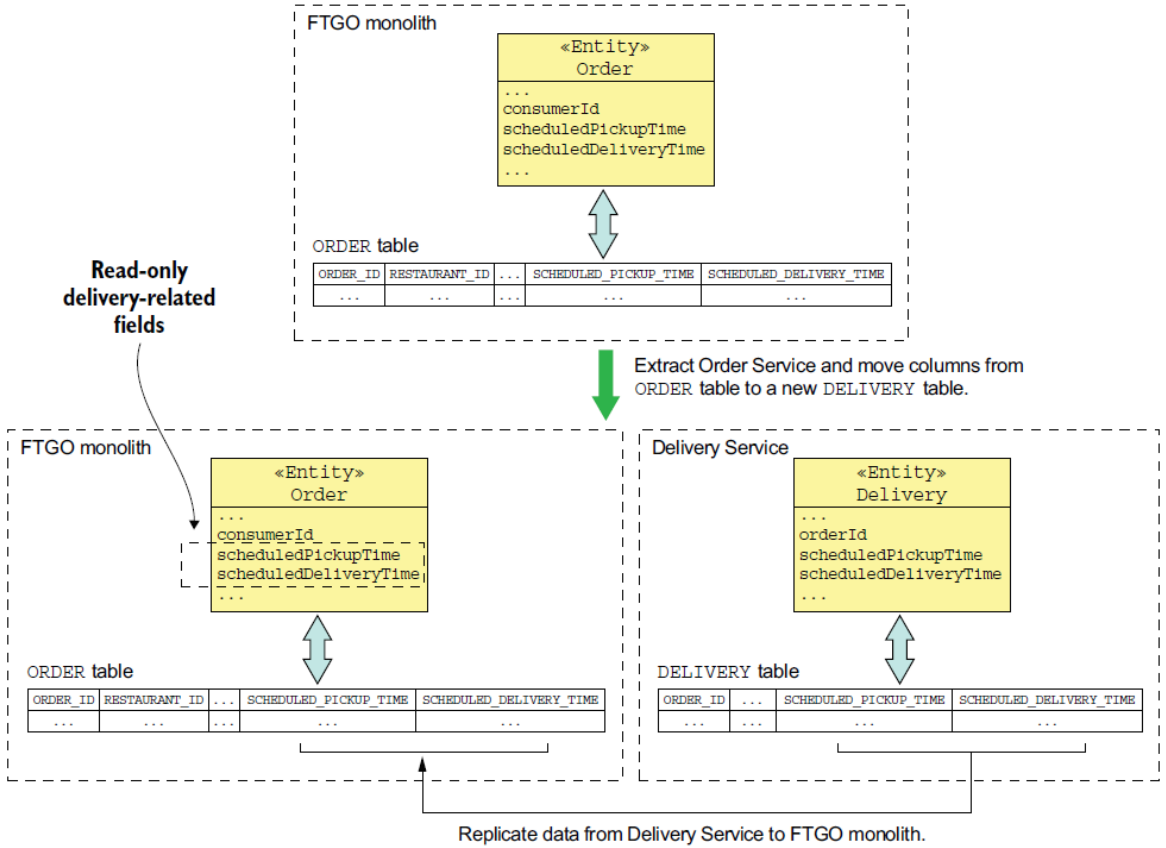
Scott J Ambler, Pramod J. Sadalage



Репликация данных для ограничения области изменений

Сохраняем исходную схему на время переходного периода и используем триггеры для ее синхронизации с новыми схемами.

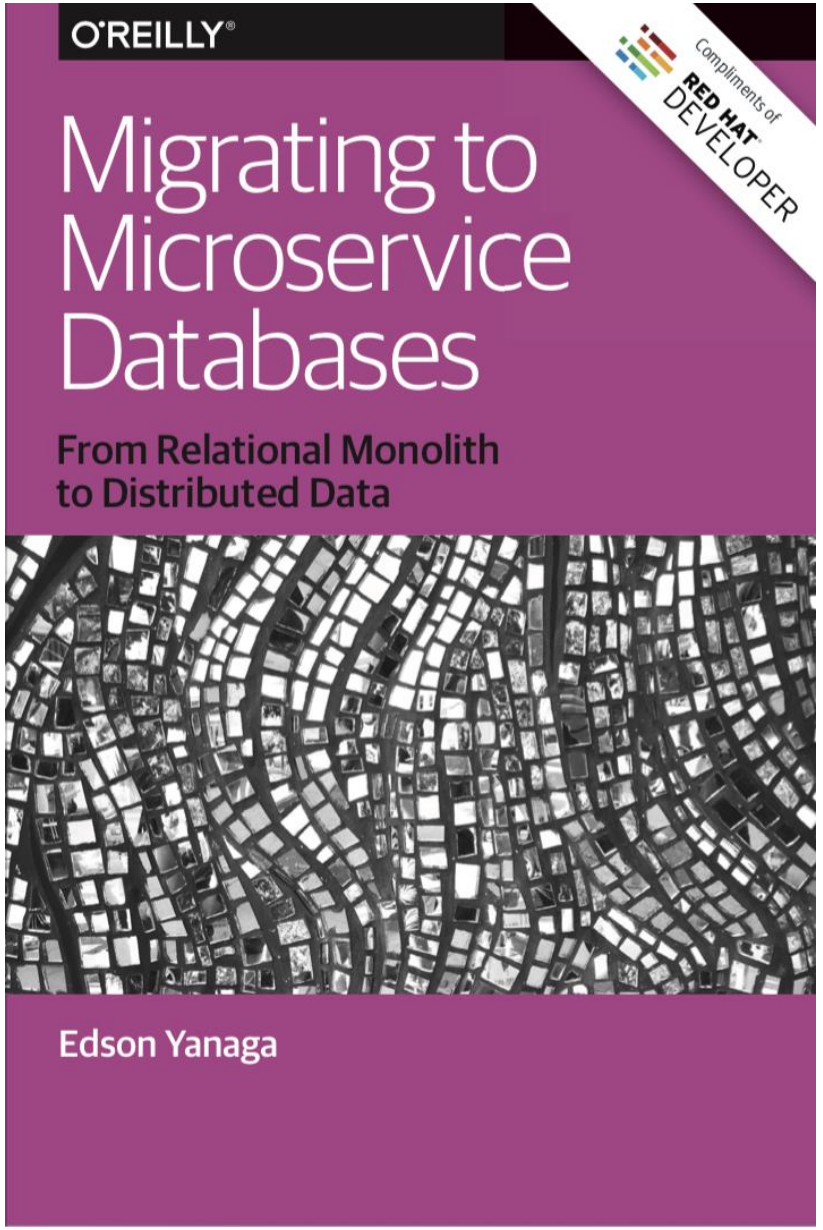
При этом клиенты постепенно мигрируют со старой схемы на новую.



Рекомендуемая литература

Microservice database migration guide

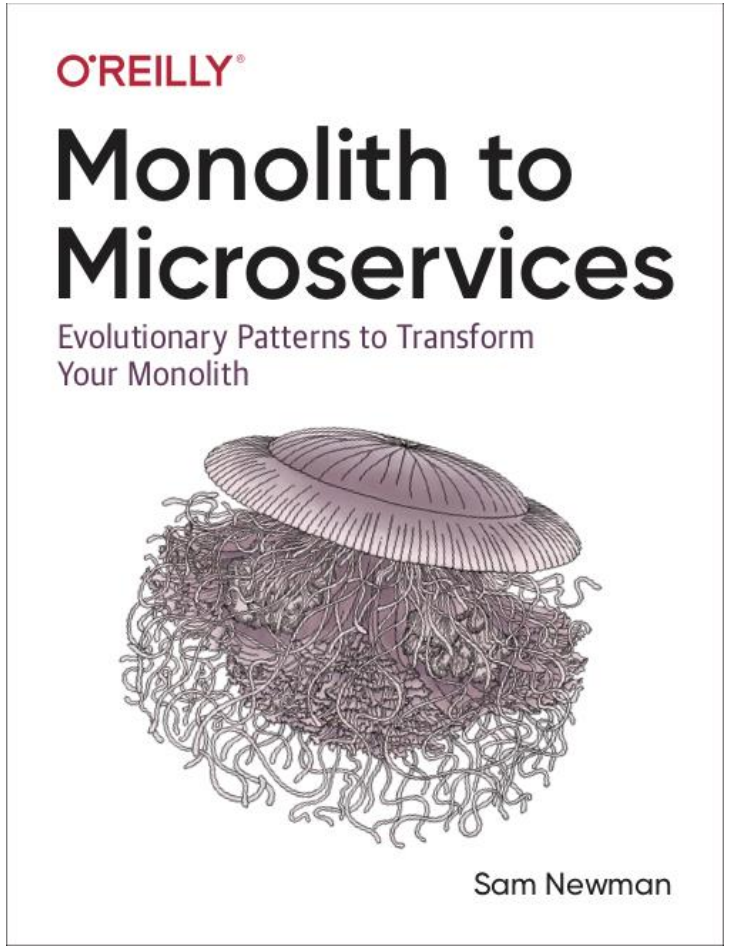
Edson Yanaga



Паттерны декомпозиции
Сэм Ньюман

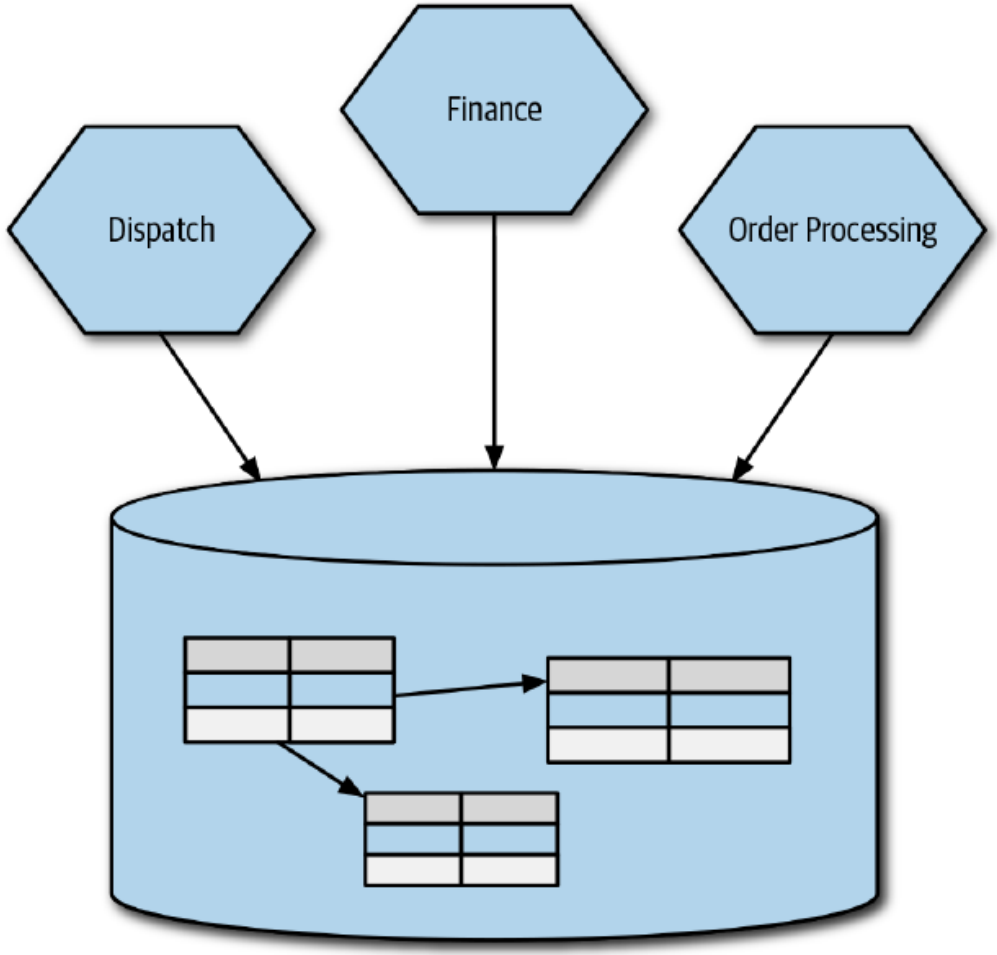
- Общая база данных (The Shared Database)
- Представления (Database View)
- Сервис-обертка (Database Wrapping Service)
- База данных как сервис

  (Database-as-a-Service Interface)



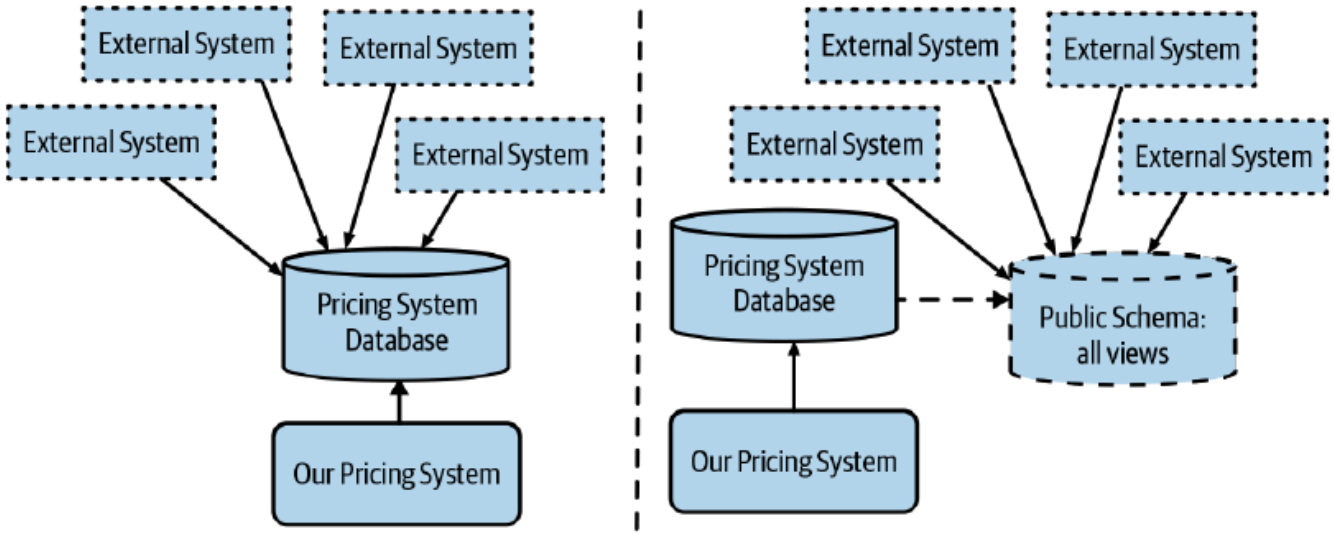
Общая база данных (The Shared Database)

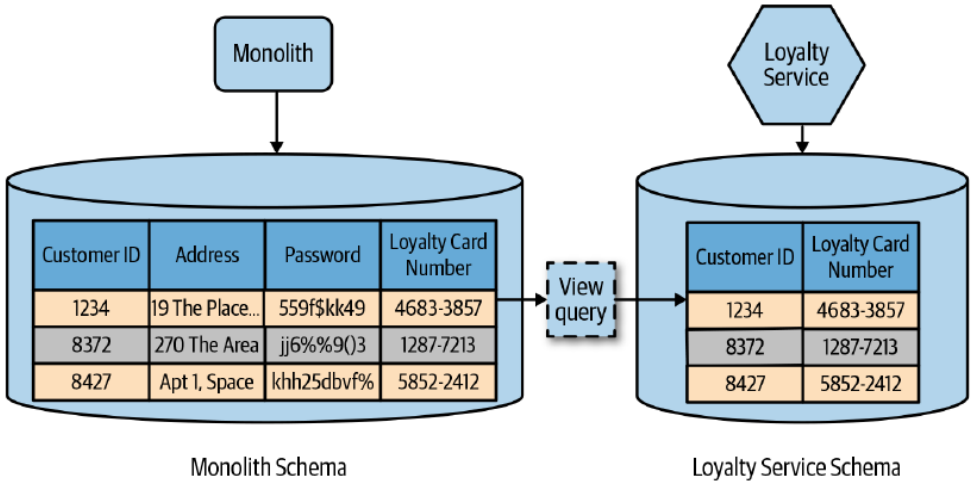
- **Разделяем бизнес логику, но временно оставляем общую базу данных**



Представления (Database View)

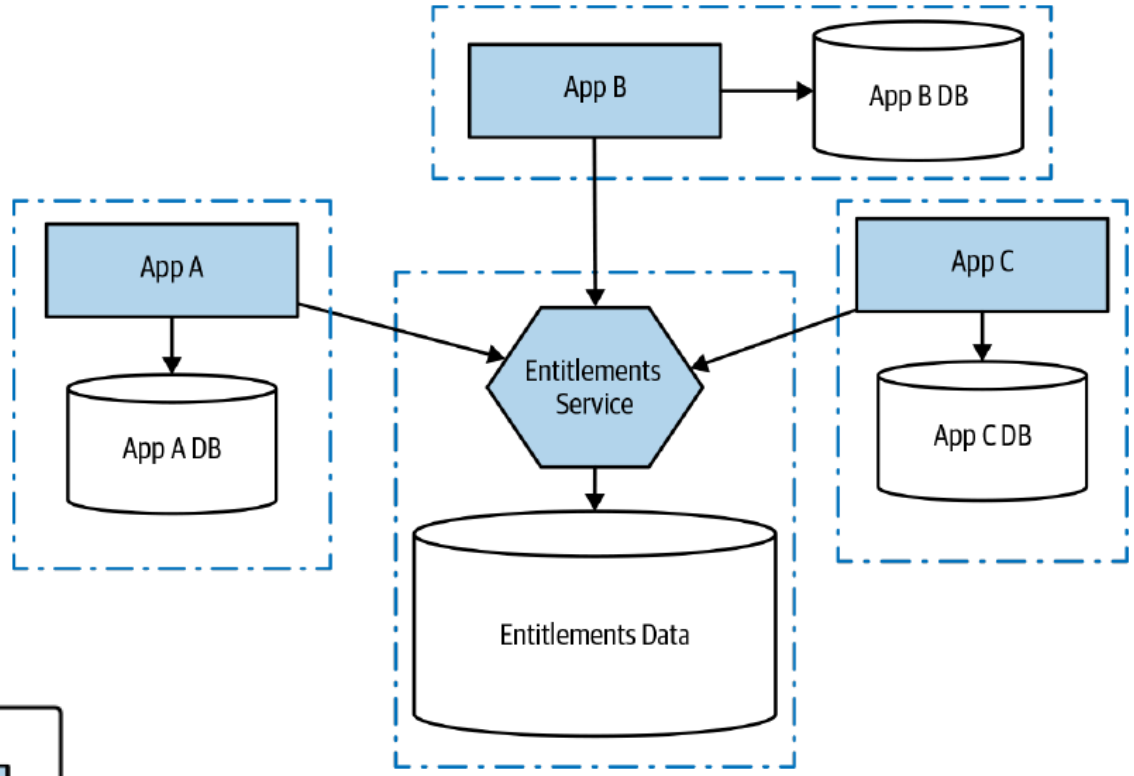
- Разделяем схемы данных и предоставляем доступ к представлениям (view)
- Представление будет являться проекцией данных

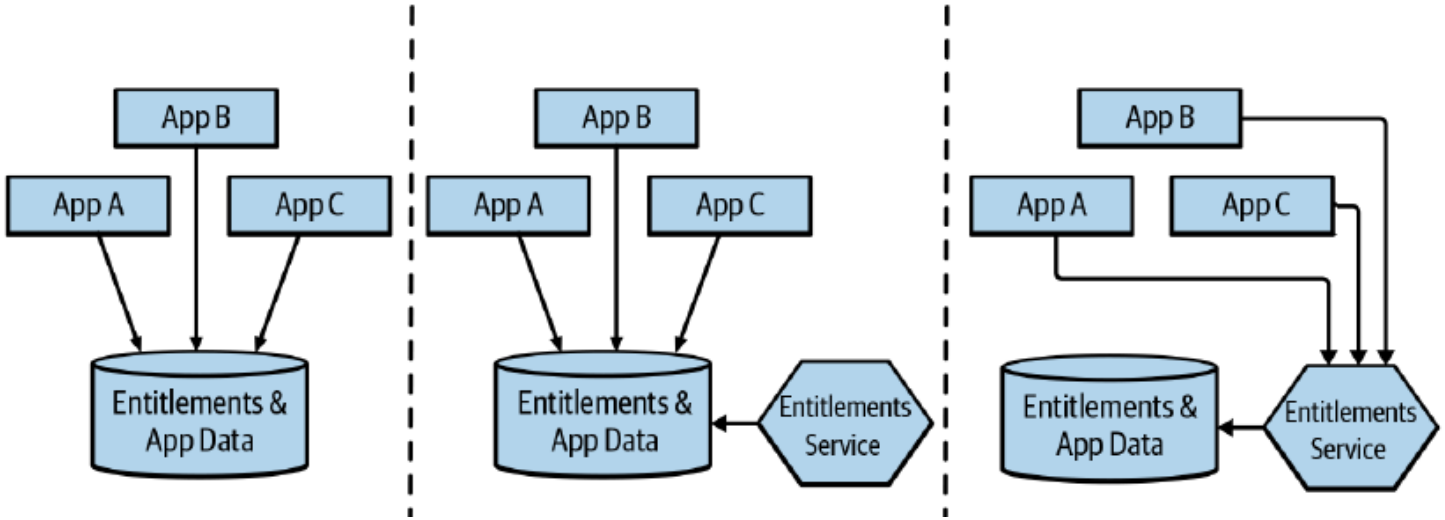




Сервис-обертка (Database Wrapping Service)

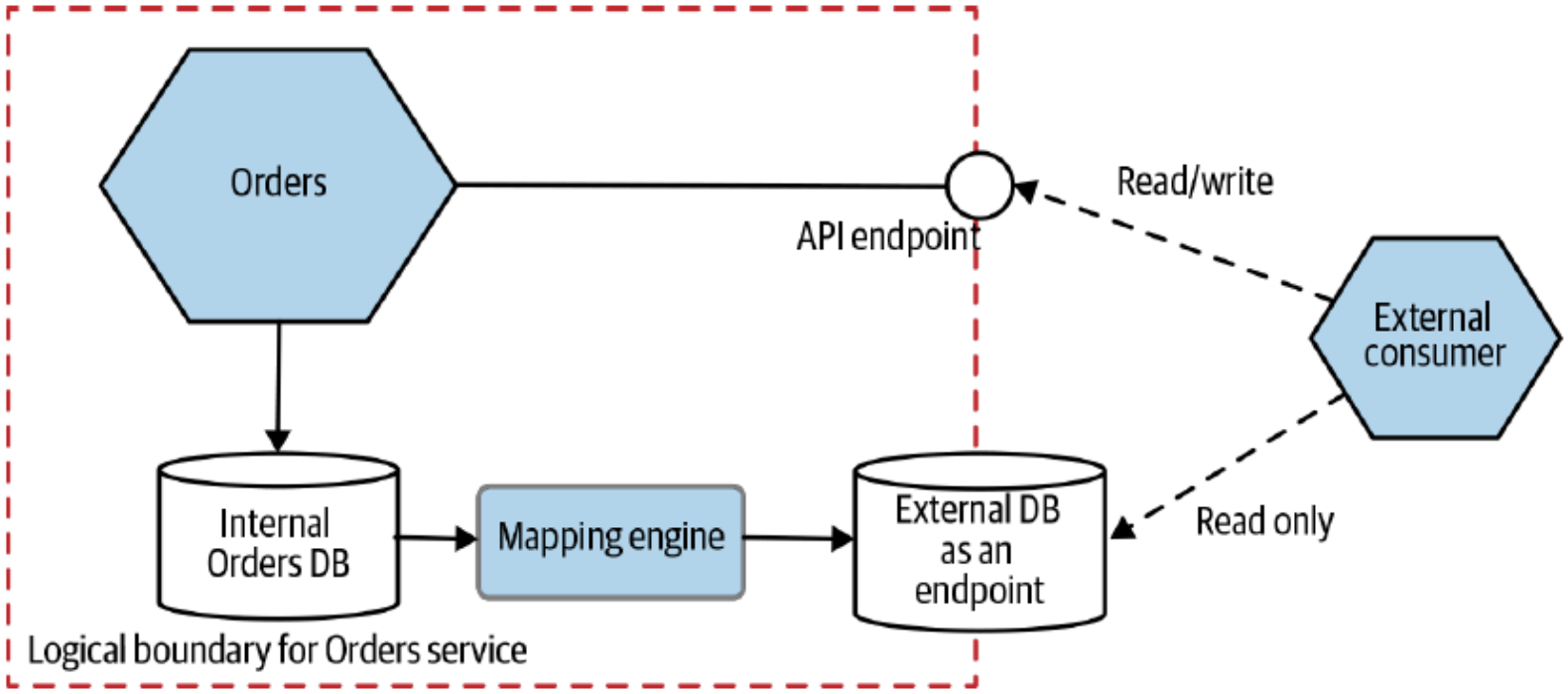
- **Инкапсулируем общие данные посредством сервиса-обертки**
  - **Предотвращает дальнейший рост БД**
  - **Упрощает постановку заглушек**





База данных как сервис (Database-as-a-Service Interface)

- **Если внешние сервисы нуждаются только в чтении данных, мы можем им предоставить копию данных для чтения (см. Паттерн «Reporting Database»)**

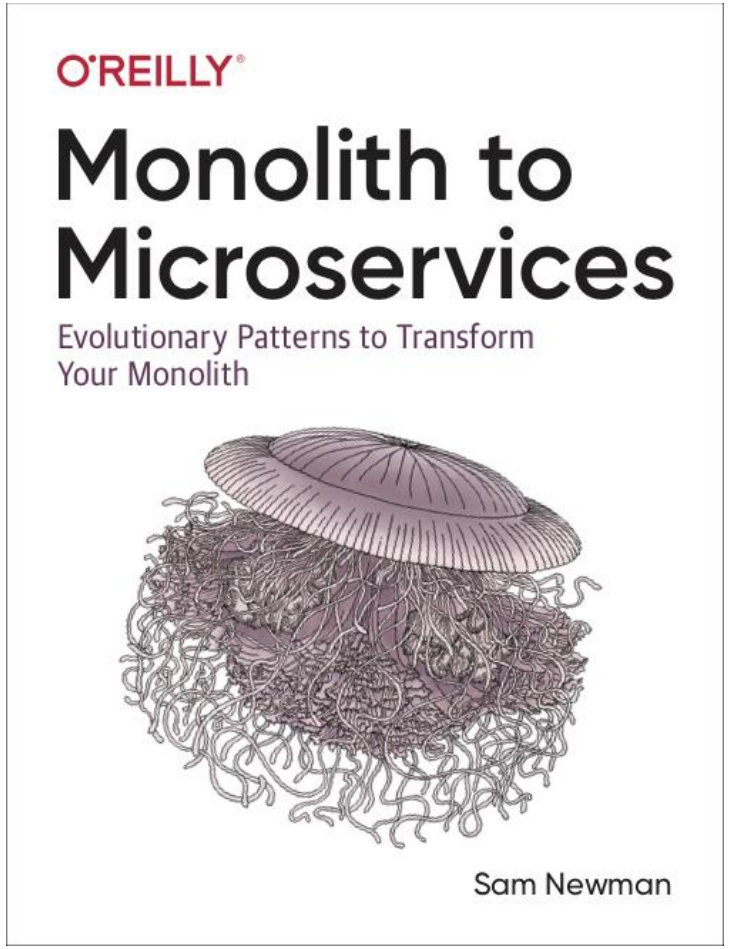


Передача данных

- **Если необходимые нашему сервису данные принадлежат монолиту, как мы можем осуществить передачу данных нашему сервису?**

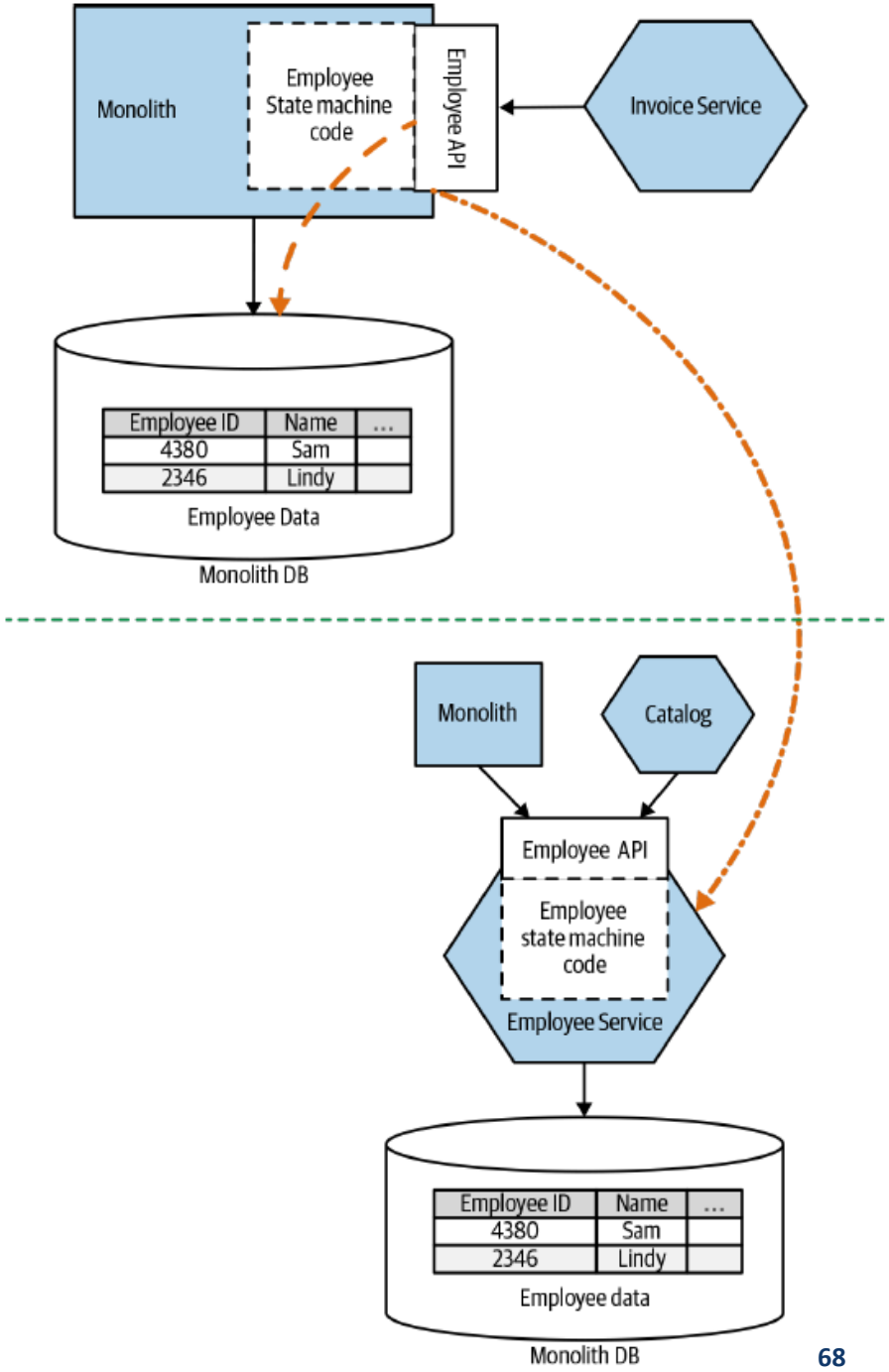
Паттерны передачи данных
Сэм Ньюман

- **Монолит, предоставляющий агрегат (Aggregate Exposing Monolith)**
- **Смена владельца (Change Data Ownership)**



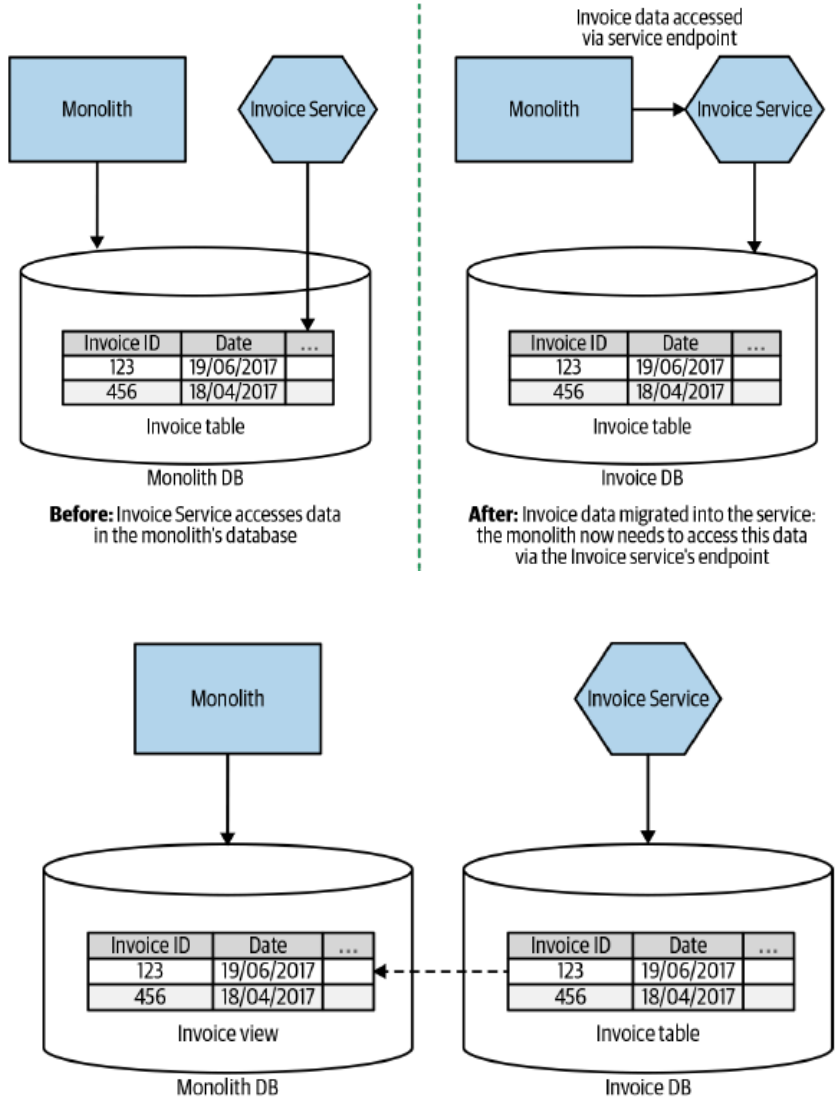
Монолит, предоставляющий агрегат (Aggregate Exposing Monolith)

- Если нужные нам данные представлены агрегатом в монолите, можем выделить агрегат в отдельный сервис



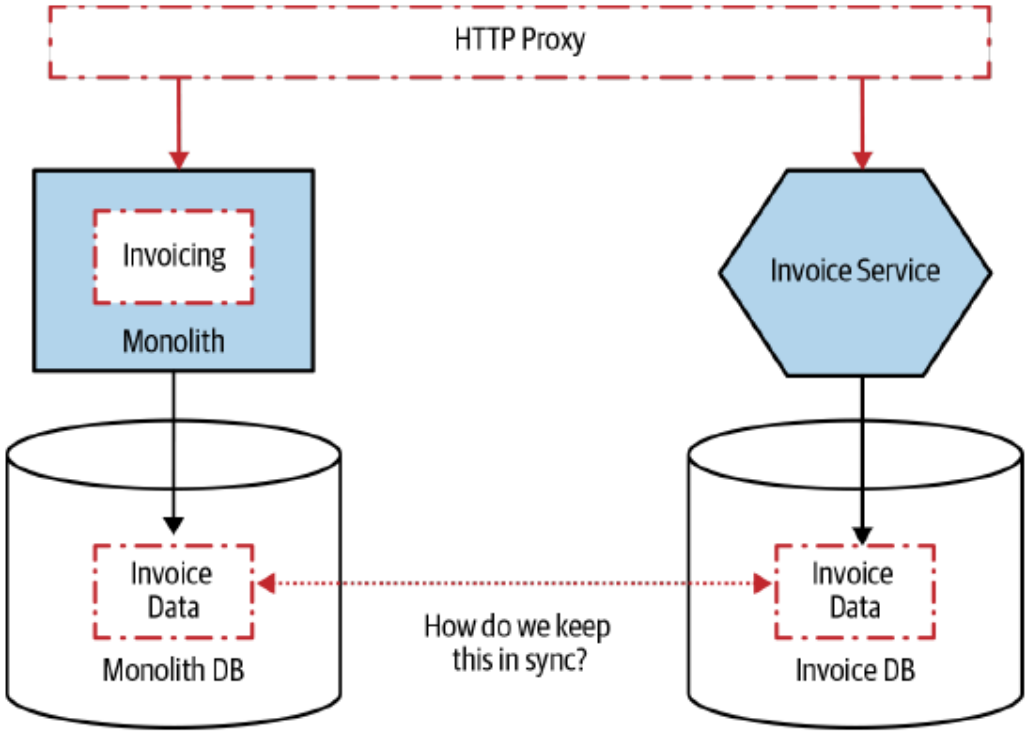
Смена владельца (Change Data Ownership)

- Если нужные нам данные представлены в монолите, передаем владение нашему сервису
- При необходимости монолит может получить данные через представление (view)



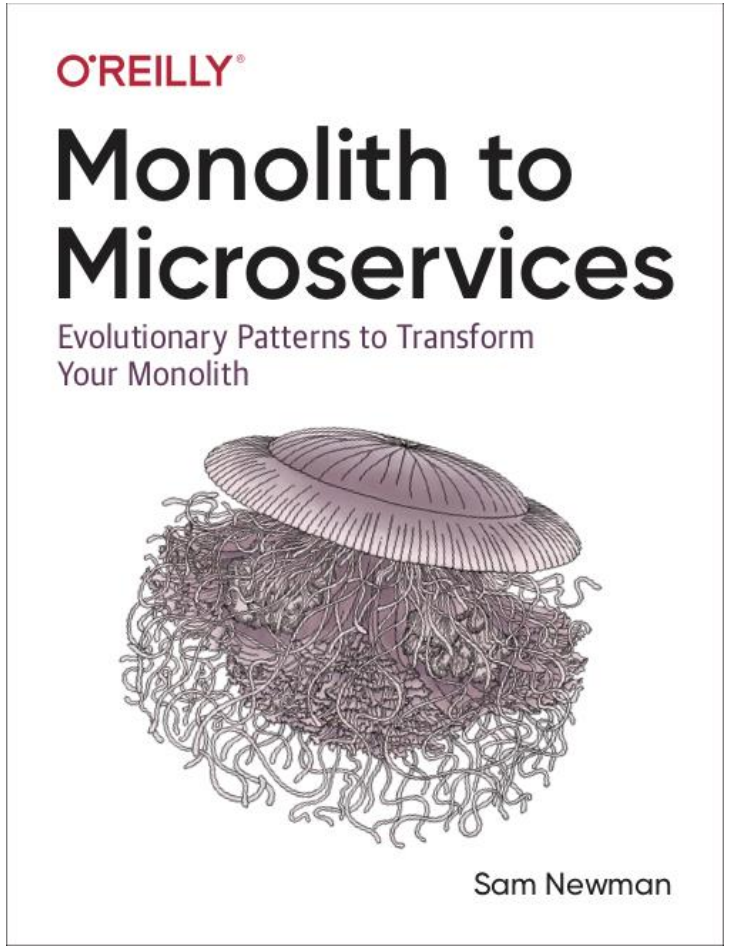
Синхронизация данных

- **Если мы проводим удушение монолита, как поддерживать синхронизацию данных между выделенным сервисом и монолитом?**



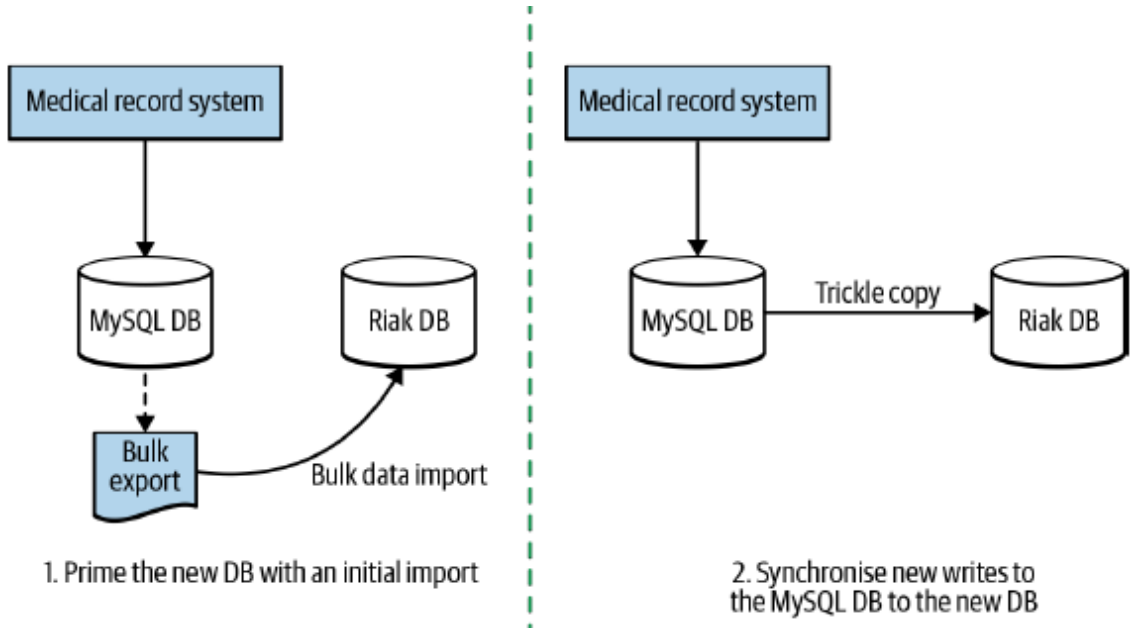
Паттерны синхронизации данных
Сэм Ньюман

- Синхронизация в приложении (Synchronize Data in Application)
- Трасирующая запись (Tracer Write)

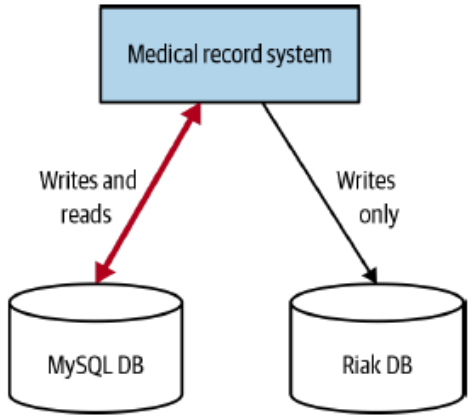


Синхронизация в приложении (Synchronize Data in Application)

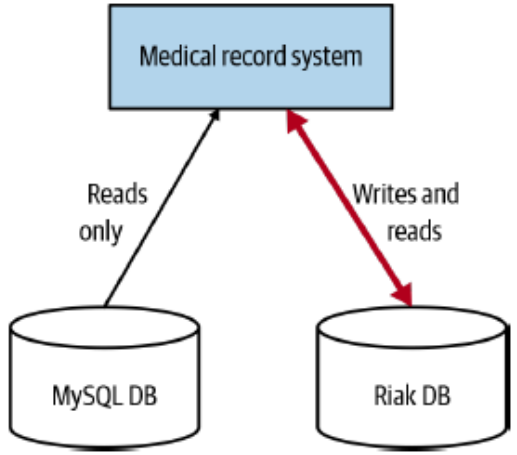
Шаг 1. Перенос данных в новую БД



Step 2: Пишем в обе БД, читаем из старой



Step 2: Пишем в обе БД, читаем из новой

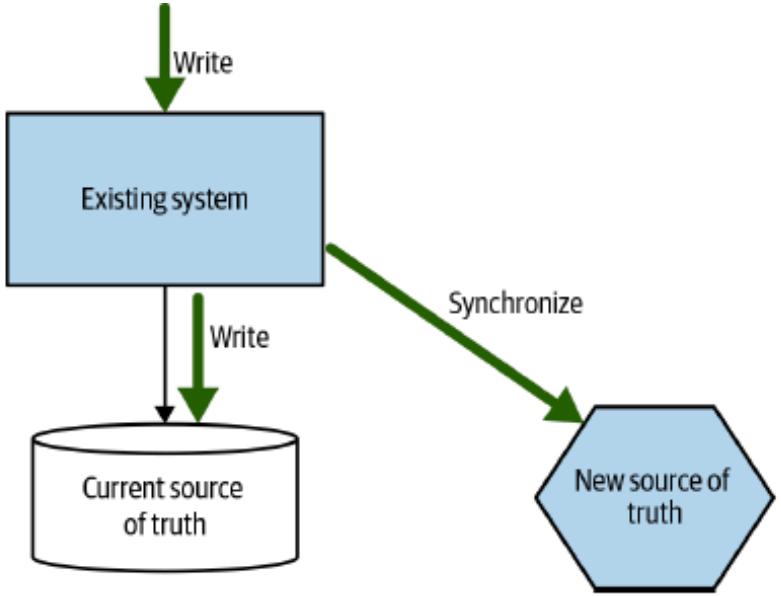


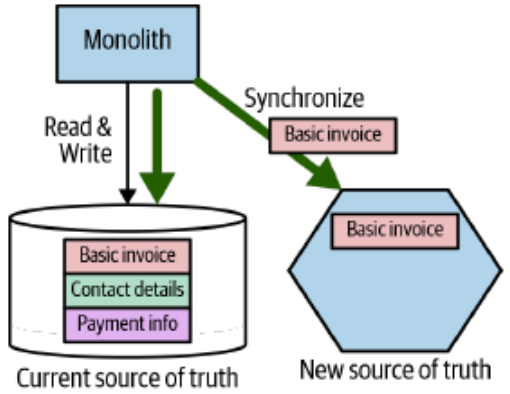
Трасирующая запись (Tracer Write)

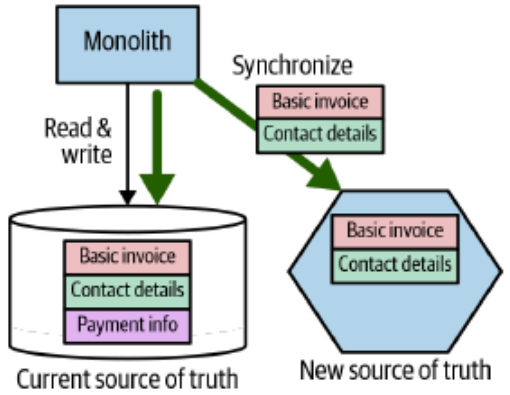
- **Пишем данные в старую базу и одновременно передаем их в новый сервис**

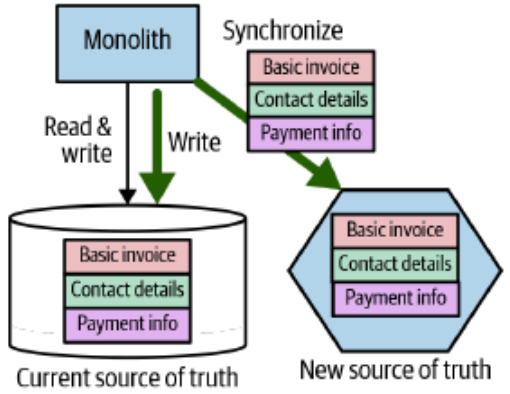
  Обе системы работают параллельно

- **Можем переносить данные инкрементально**









Буду рад ответить на ваши вопросы