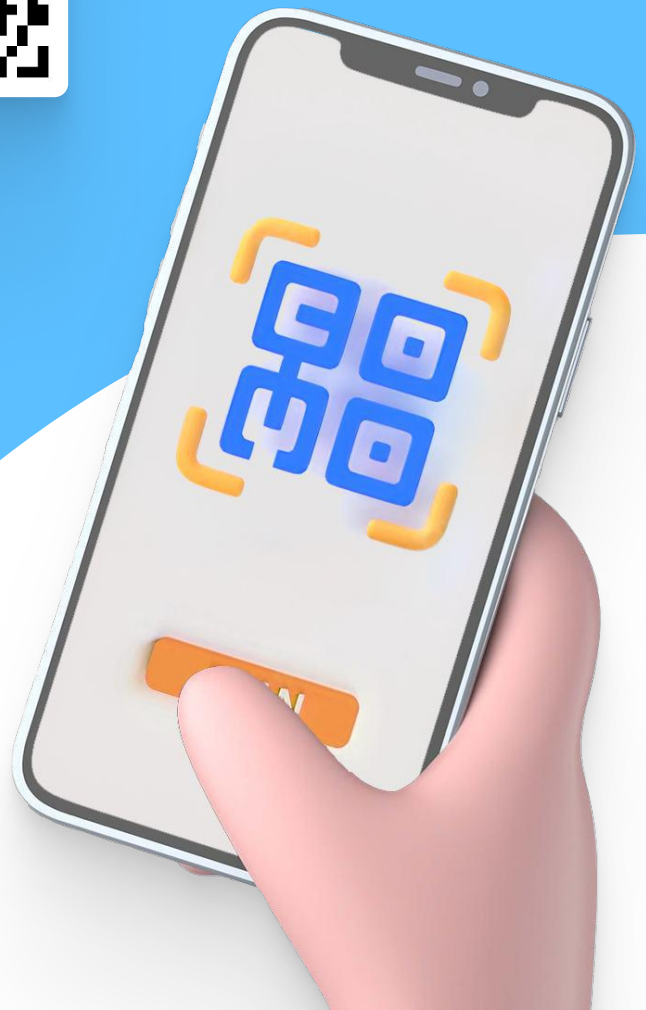
СКЛАДЧИК

# КУПЛЕНО НА SKLADCHIK.ORG

Все самые свежие
курсы по лучшей цене!

SKL.TO

Отсканируйте
QR-код телефоном



# Микросервисная архитектура

# Обо мне

МегаФон

TELE2

Team Leader

Код, структура сервиса

BCS BROKER

Lead of 6 Teams

Распределение сервисов по командам, интеграции

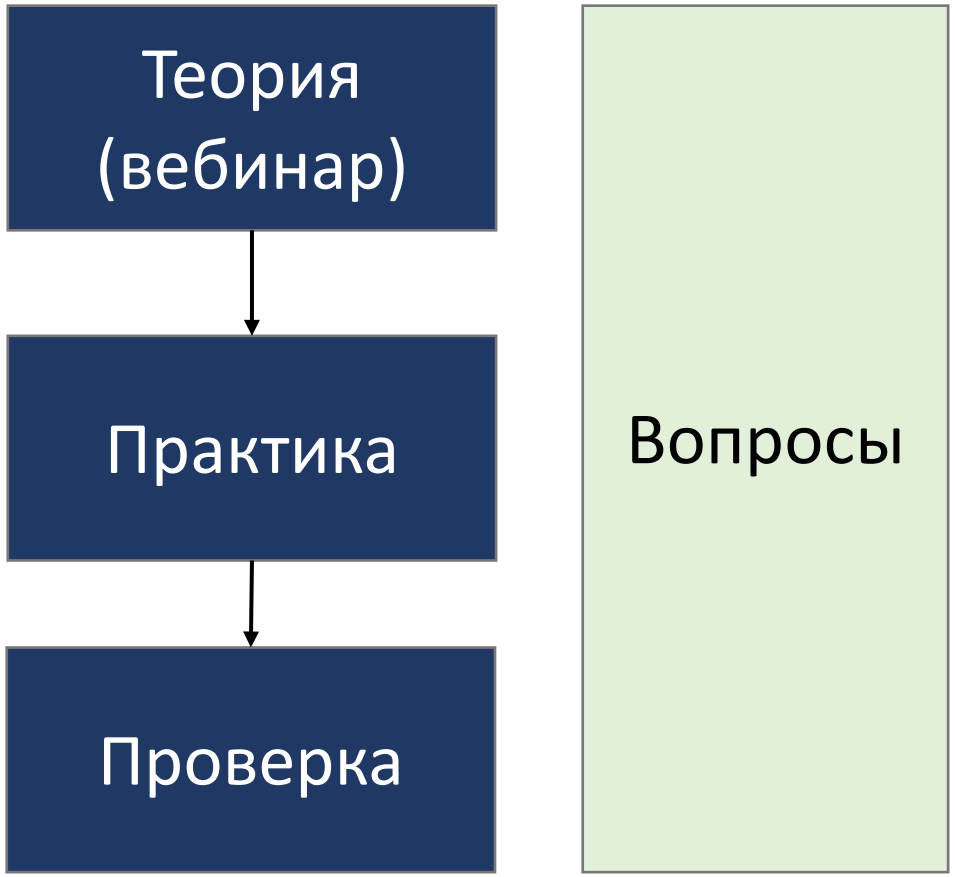
СБЕР МАРКЕТ

Architect

Глобальные решения, точечные решения, в компании с 900 IT специалистами.


microarch

# Как будем работать



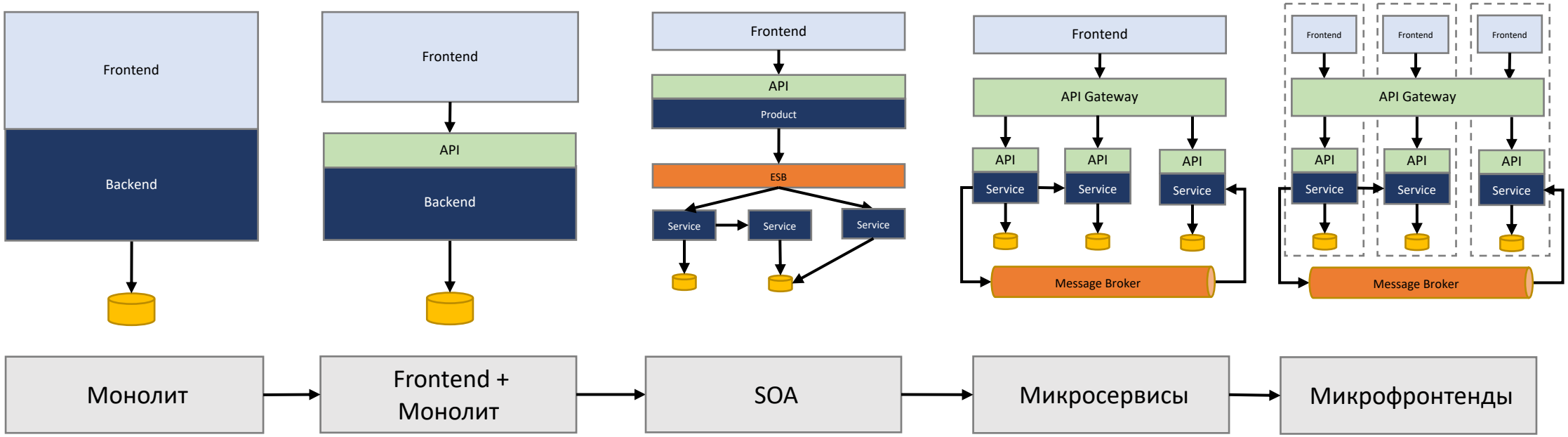
# Подготовка

# Цели занятия

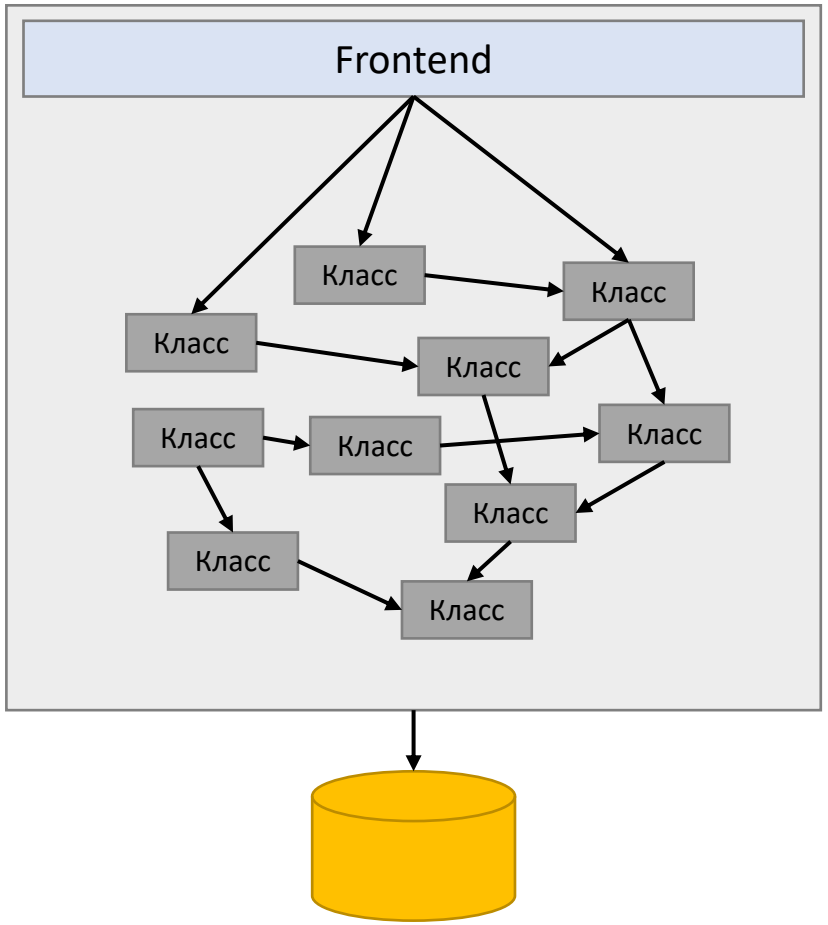
- Познакомиться с разными архитектурными подходами
- Посмотреть на плюсы и минусы микросервисной архитектуры
- Разобраться, когда стоит применять микросервисы, а когда лучше подойдёт монолит
- Определить основные цели внедрения микросервисной архитектуры в компании

# Сравнение архитектур

# Хронология развития архитектур

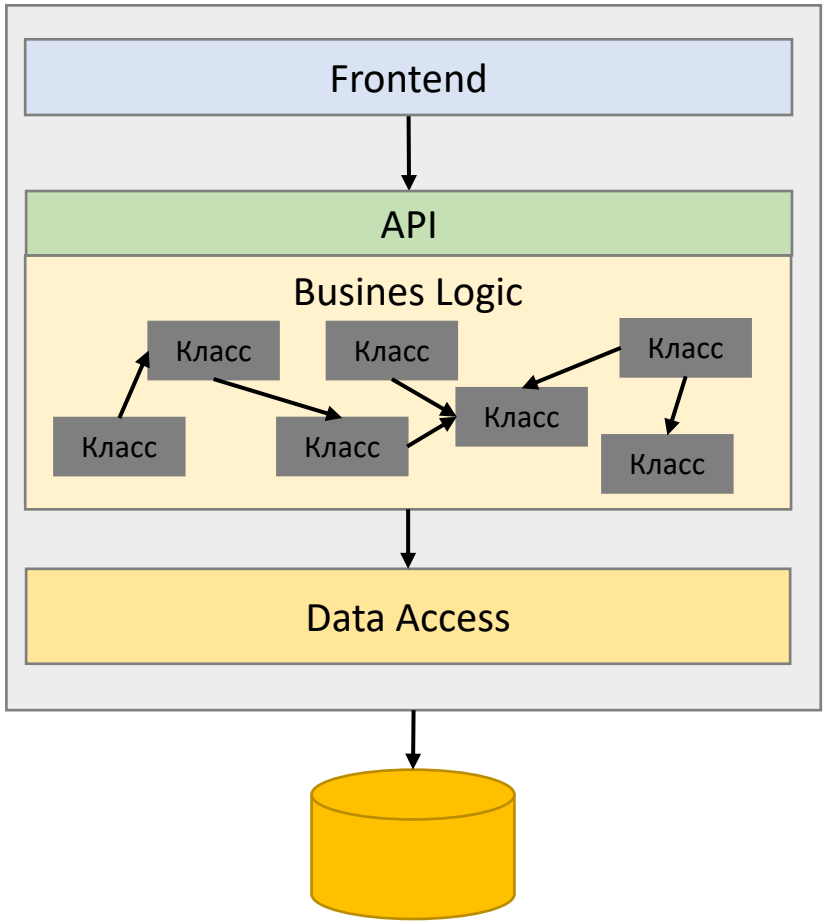


# Big Ball of Mud




- Нет четко определенного разделения на части
- Каждая часть системы связана со всеми остальными частями
- В эту архитектуру сложно вносить какие-либо изменения. Изменение порождает сбой.

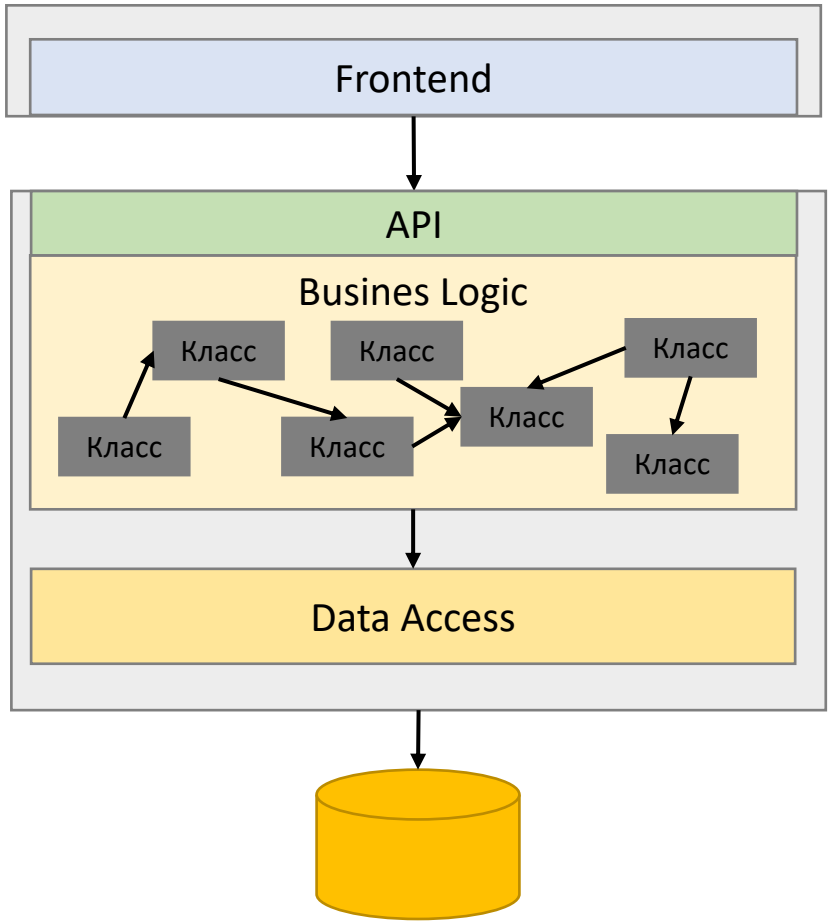
# Многоуровневая архитектура




- Каждый слой изолирован от остальных
- Каждый слой отвечает за определенную часть
- Но изменения зачастую пронизывают все слои системы

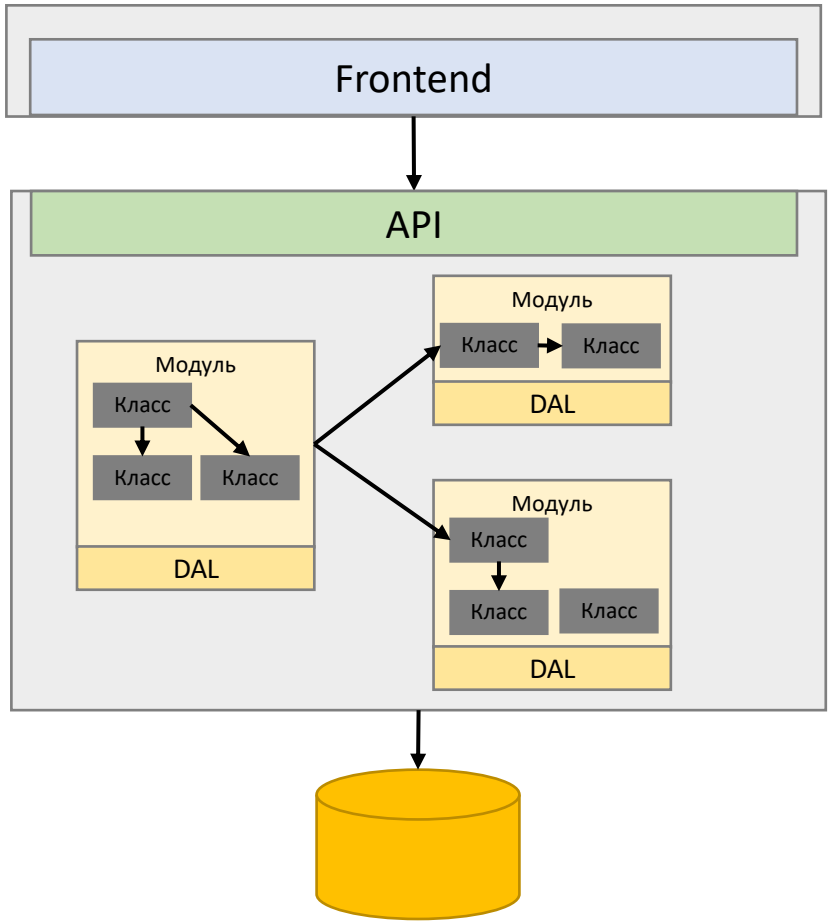
SKLADCHIK.ORG

# Frontend + Монолит



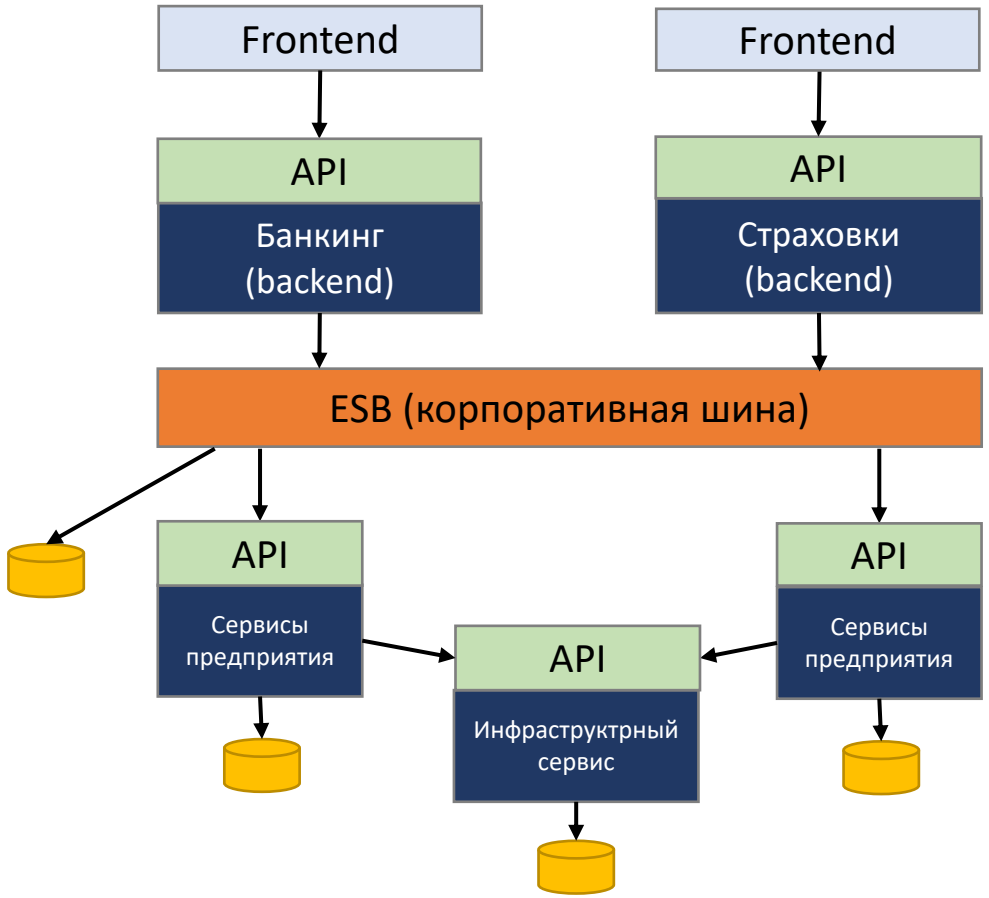

- Можно независимо деплоить Frontend и Backend
- Но проблемы не исчезли

# Модульный монолит



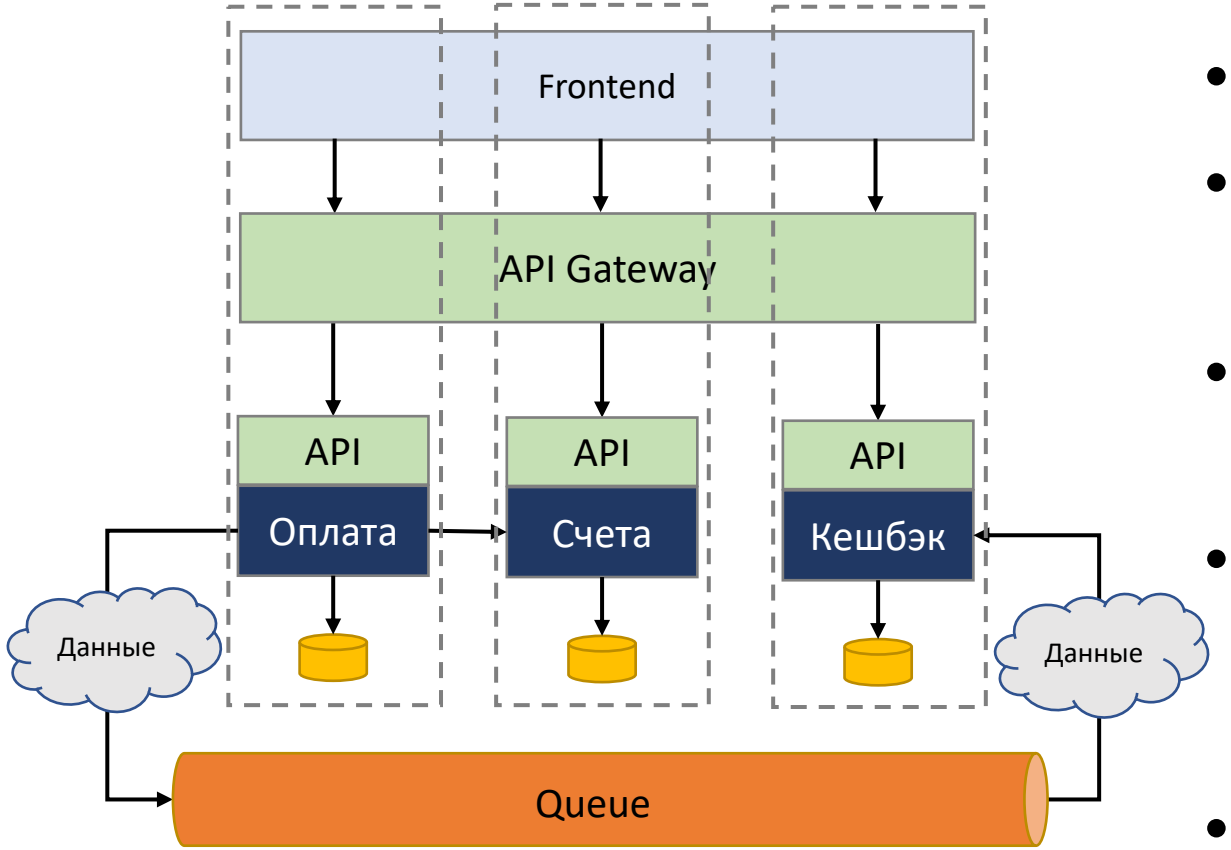

- Использует логическое группирование функциональности в модули
- Модуль может содержать все необходимые слои

# Сервис ориентированная архитектура (SOA)



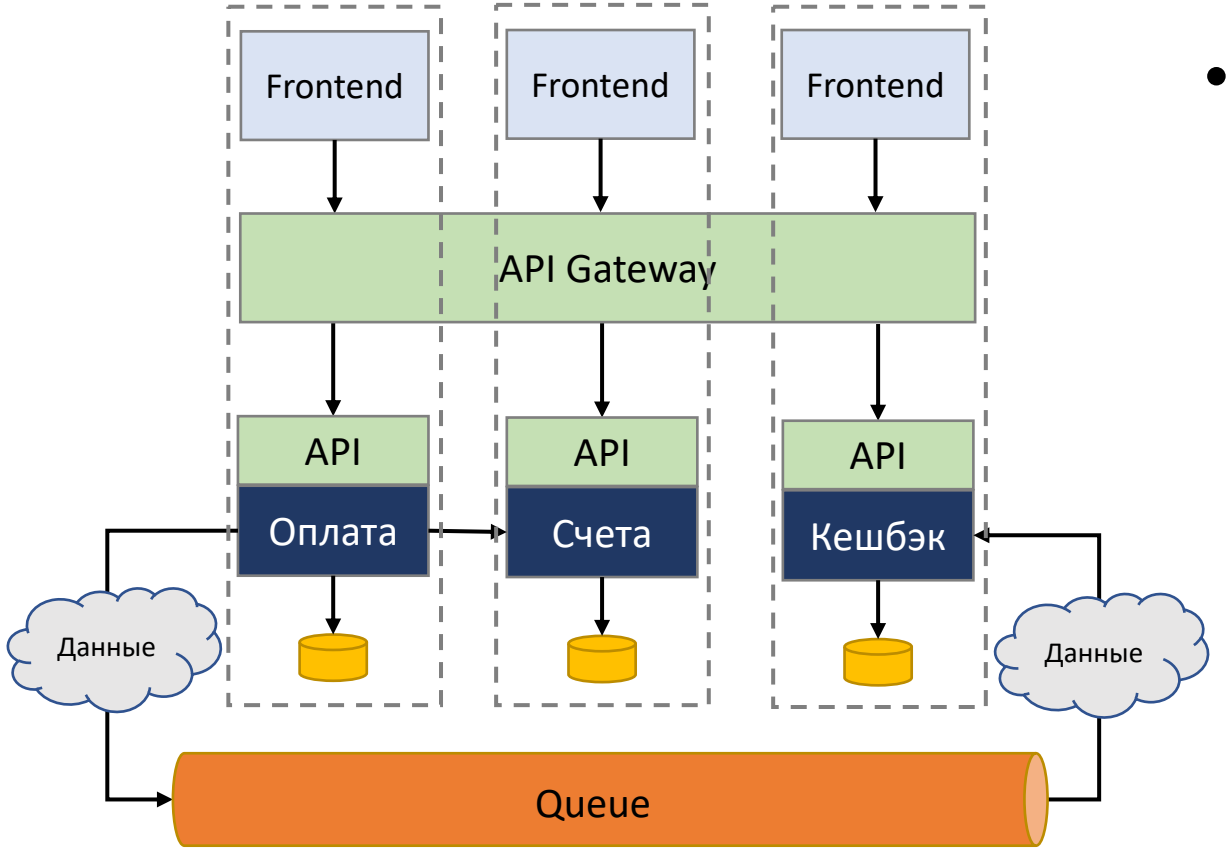

- Распределенная система
- Централизация (из-за ESB)
- Стремление к повторному переиспользованию сервисов и разделению ресурсов

# Микросервисная архитектура (MSA)




- Распределенная система
- Разделение по Subdomain (предметным областям)
- Автономность важнее переиспользования
- Легковесные каналы коммуникации (Message Broker вместо ESB)
- Без разделения ресурсов

# Микрофронтенды




- Frontend декомпозирован в соответствии с микросервисами

# Монолит vs Микросервисы

# Монолит

**Минусы**

- Сложность понимания
- Сложность внесения изменений
- Сложность масштабирования разработки
- Привязанность к технологическому стеку
- Хрупкость
- Сложность тестирования

**Плюсы**

- Меньше зависимостей от других команд
- Простота развертывания
- Меньшие требования к компетенциям
- ACID транзакционность

# Микросервисы

**Минусы**

- Сложность правильной декомпозиции
- Трудозатраты на интеграции
- Отсутствие транзакционности между сервисами
- Порой бывают зависимости от других команд

**Плюсы**

- Легкое масштабирование разработки
- Легче вносить изменения
- Лояльность к инновациям
- Легче тестировать
- Оптимизация ресурсов

# Область применения микросервисов

# Нет лучшей архитектуры

| Монолит (Big Ball of Mud) | Монолит (модульный) | SOA | Микросервисы |
|---|---|---|---|
| Идея / Стартап | Продукт до 50-70 разработчиков | Не актуально на сегодняшний день | Большой продукт, больше 50-70 разработчиков |

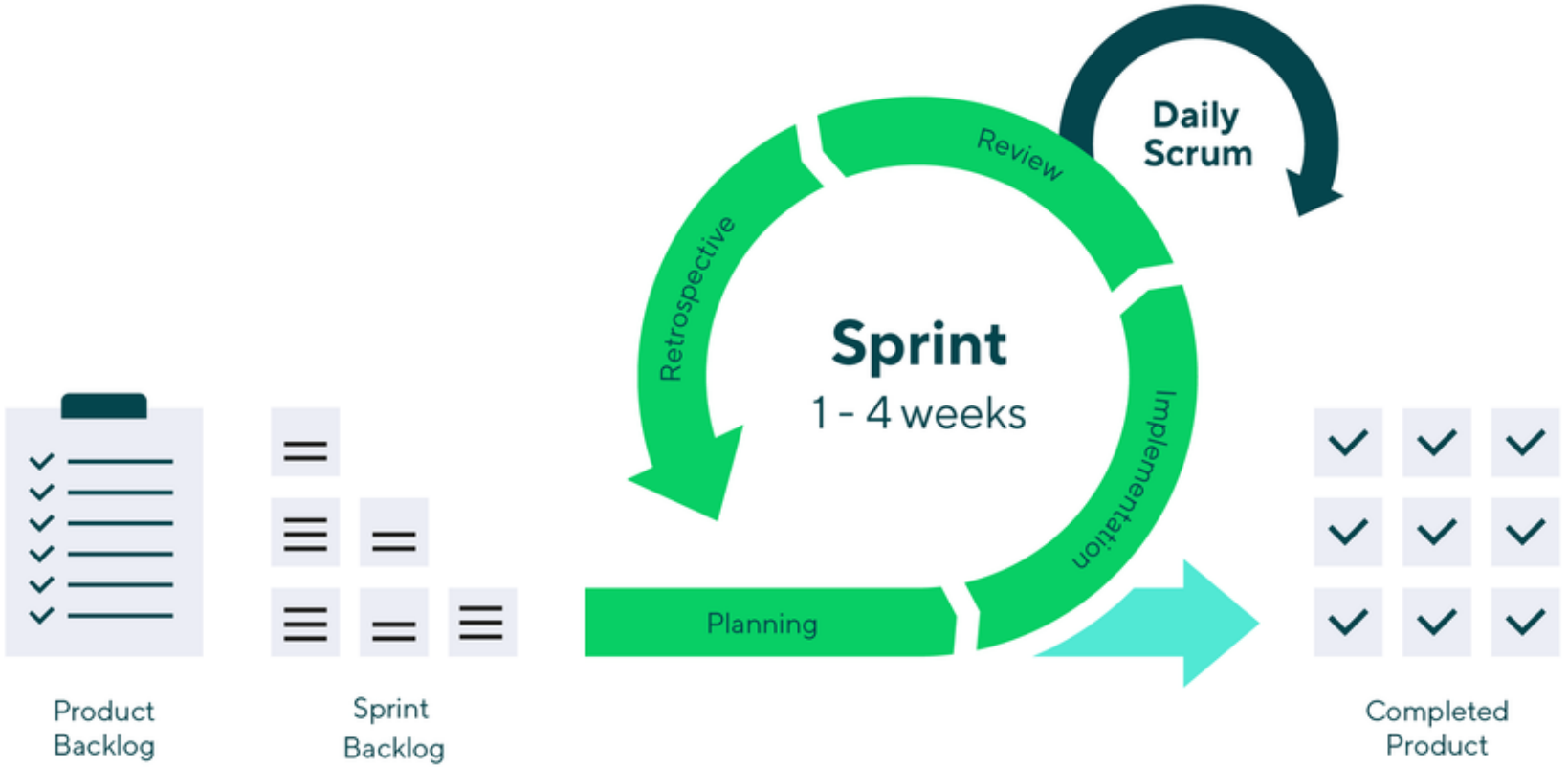
SKLADCHIK.ORG

# Вывод

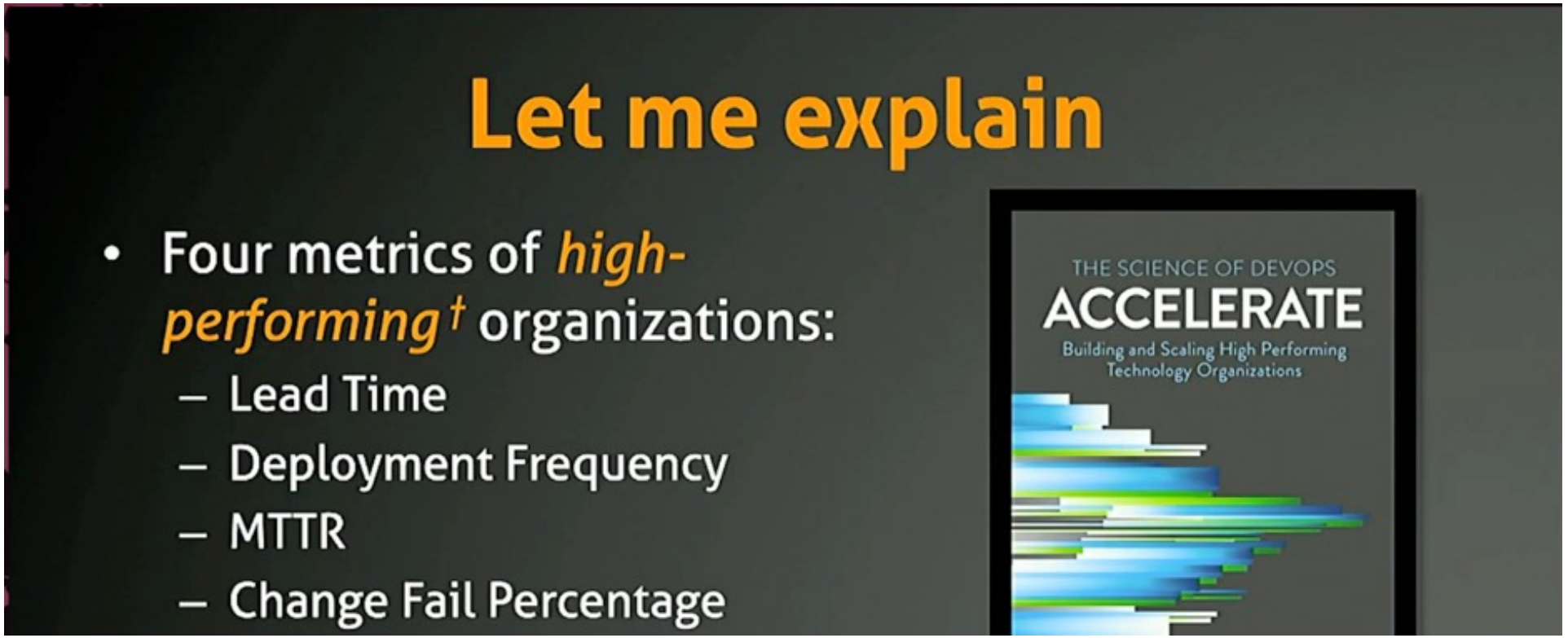
Микросервисы применимы для больших систем и большого штата разработчиков, в небольших системах нет проблем, которые они решают.

# Цели внедрения MSA

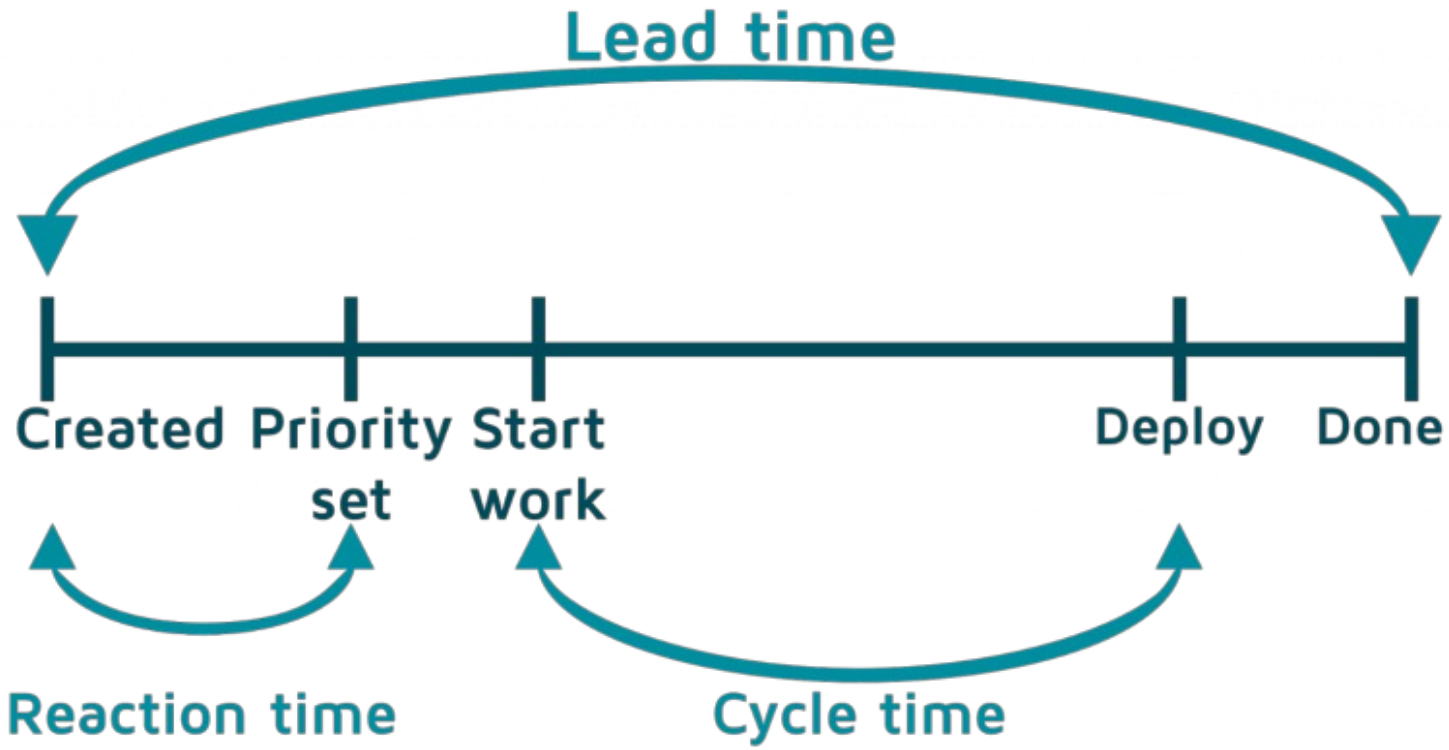
# Что хочет бизнес?



# Метрики DORA

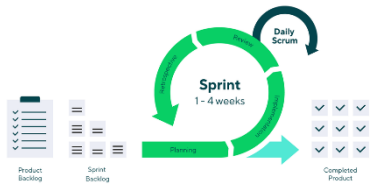


# Метрики процесса разработки



# Цель внедрения микросервисов

**Цель бизнеса**




1. Быстрая реакция на изменения рынка
2. Частая поставка

**Кто**

IT департамент

**Как**

Сократить Lead time на N%

**Что**

- Оптимизация процессов разработки
- Изменение орг. структуры
- Внедрение DevOps
- Внедрение микросервисов

# Вывод

Первичная цель внедрения микросервисов – сокращение или удержание Lead Time в больших системах.

# Вывод

Если после внедрения микросервисов Lead time не снижается - вы делаете что-то не так.

# О чем узнали

- Микросервисная архитектура содержит не только плюсы, но и минусы. Не всем проектам она подходит
- Технические цели внедрения микросервисной архитектуры важны, но первостепенная цель – это высокая скорость разработки в проектах с большим количеством участников
- Преждевременное применение микросервисной архитектуры может негативно сказаться на скорости разработки.

# Основы
# Domain Driven Design

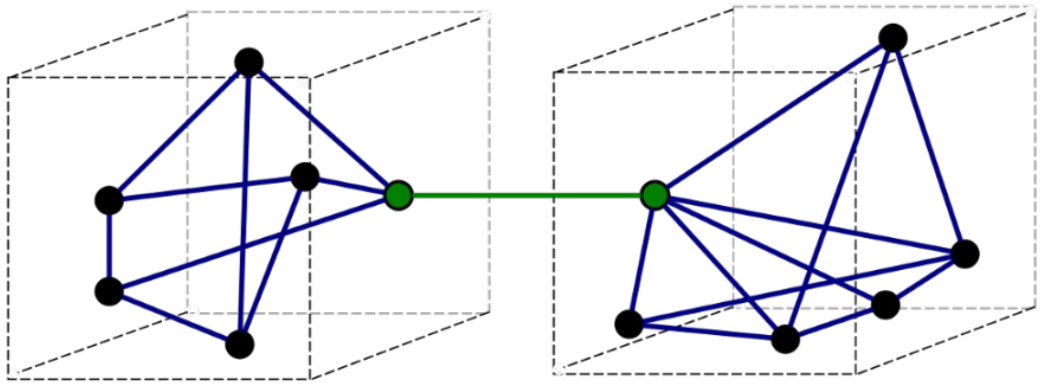
SKLADCHIK.ORG

# Цели занятия

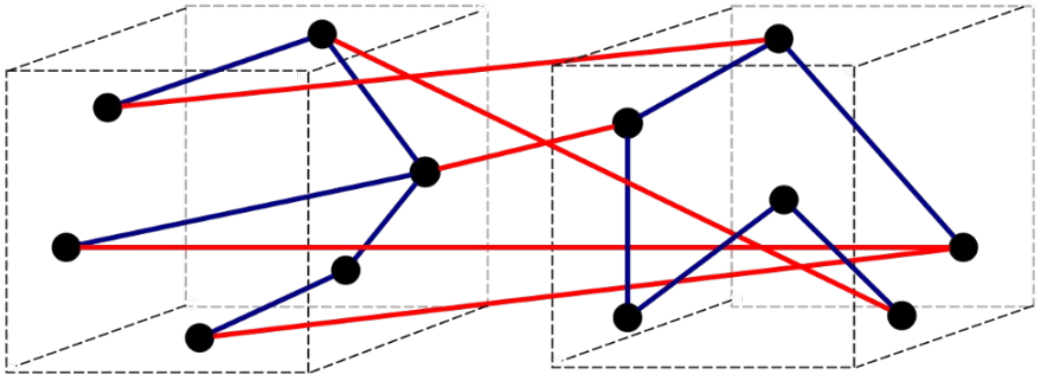
- Понять основной принцип хорошей декомпозиции
- Познакомиться с Domain Driven Design (DDD)
- Изучить стратегические и тактические паттерны DDD

# Правило хорошей декомпозиции

# Правило хорошей декомпозиции



a) Low Coupling (низкая связанность) и High Cohesion (высокое зацепление)



b) High Coupling (высокая связанность) и Low Cohesion (низкое зацепление)

# Микросервис

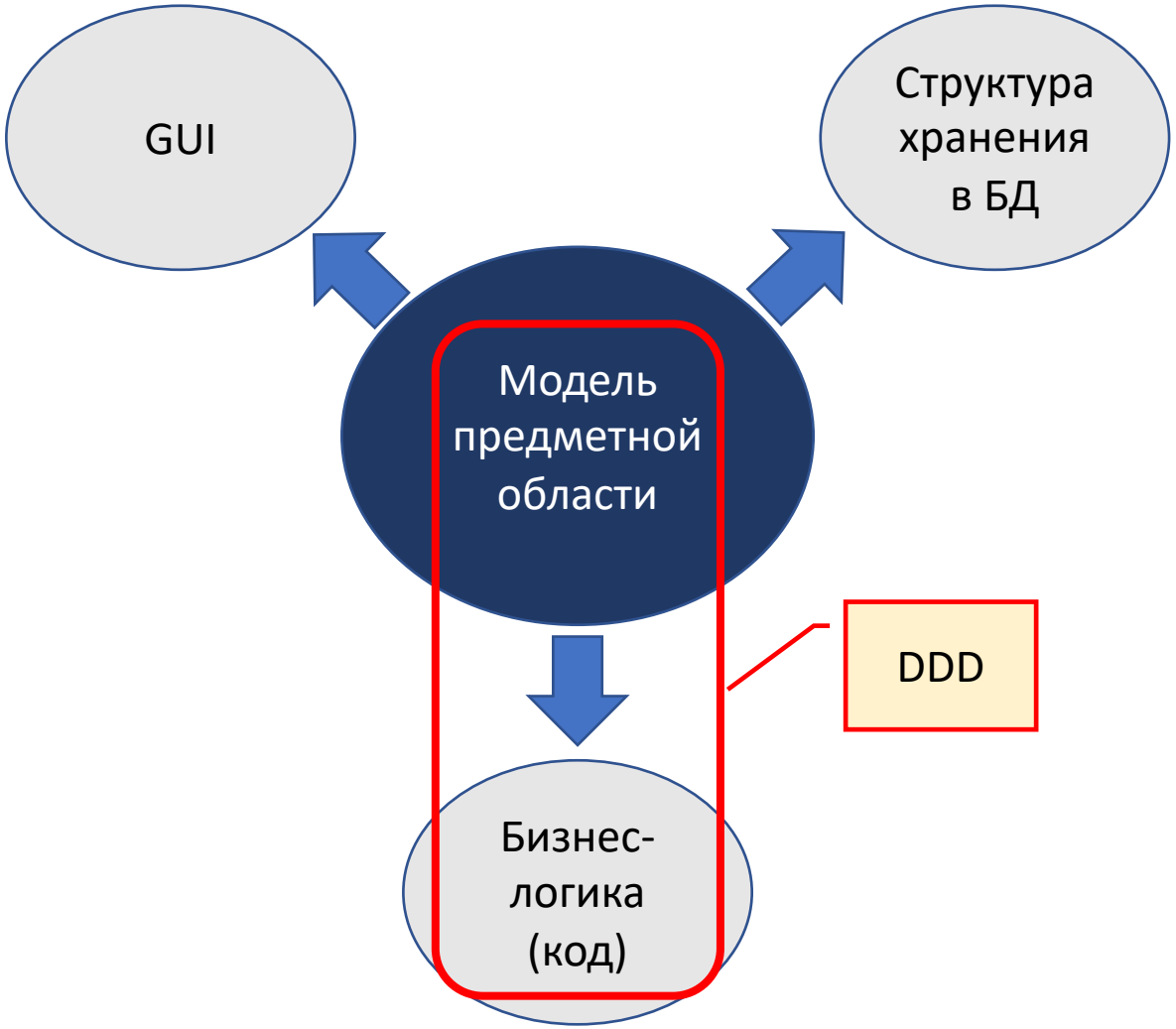
Микросервис — это **автономный**, независимо развертываемый программный компонент, который реализует определенные полезные функции. Границы сервисов формируются **на основе бизнес-границ**.

# Domain Driven Design
# (DDD)

# Суть подхода

**DDD (Domain Driven Design)** - предметно-ориентированная парадигма проектирования.

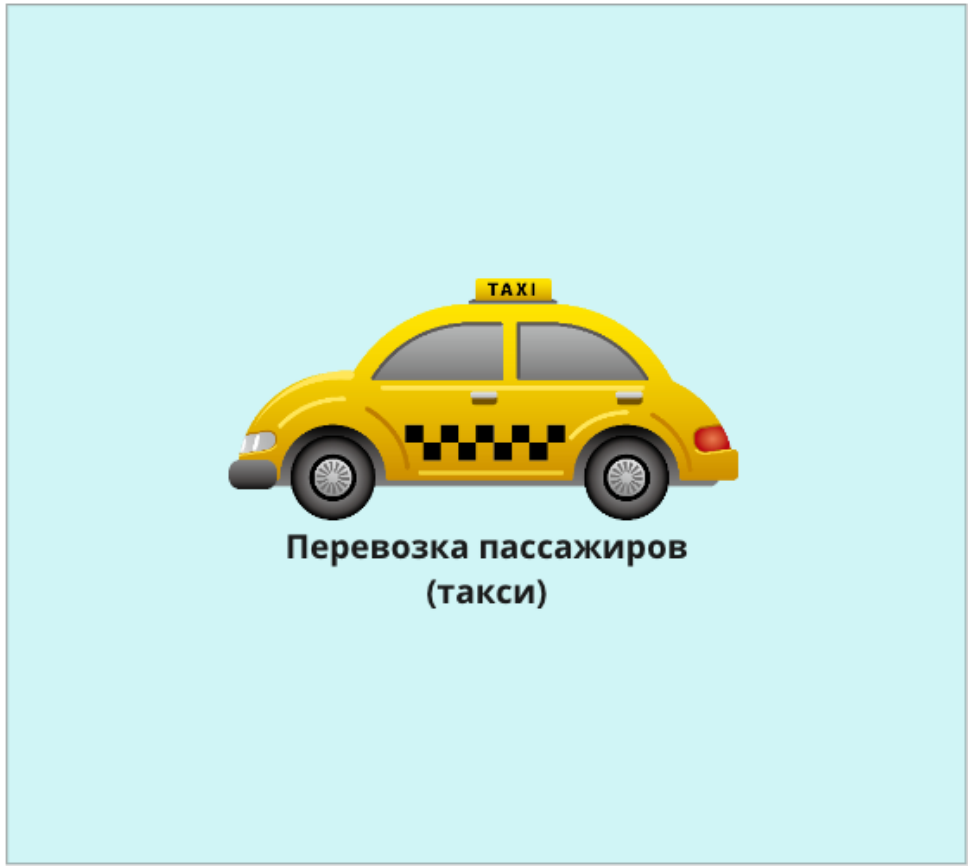
В данной парадигме ключевым понятием является логика (бизнес-логика) вокруг которой строится приложение.



# Стратегические паттерны

# Business Domain

Компания




**Business Domain** определяет основную область деятельности компании. Это услуга, которую компания предоставляет своим клиентам.

# Domain expert

**Domain expert (эксперт в предметной области)** — это человек, который глубоко разбирается в бизнес-сфере, начиная с ее политик и рабочих процессов, заканчивая ее особенностями.

# Subdomain


**Subdomain** — это подобласть деловой активности (бизнес возможности).

Все **Subdomain** компании образуют ее Business Domain.

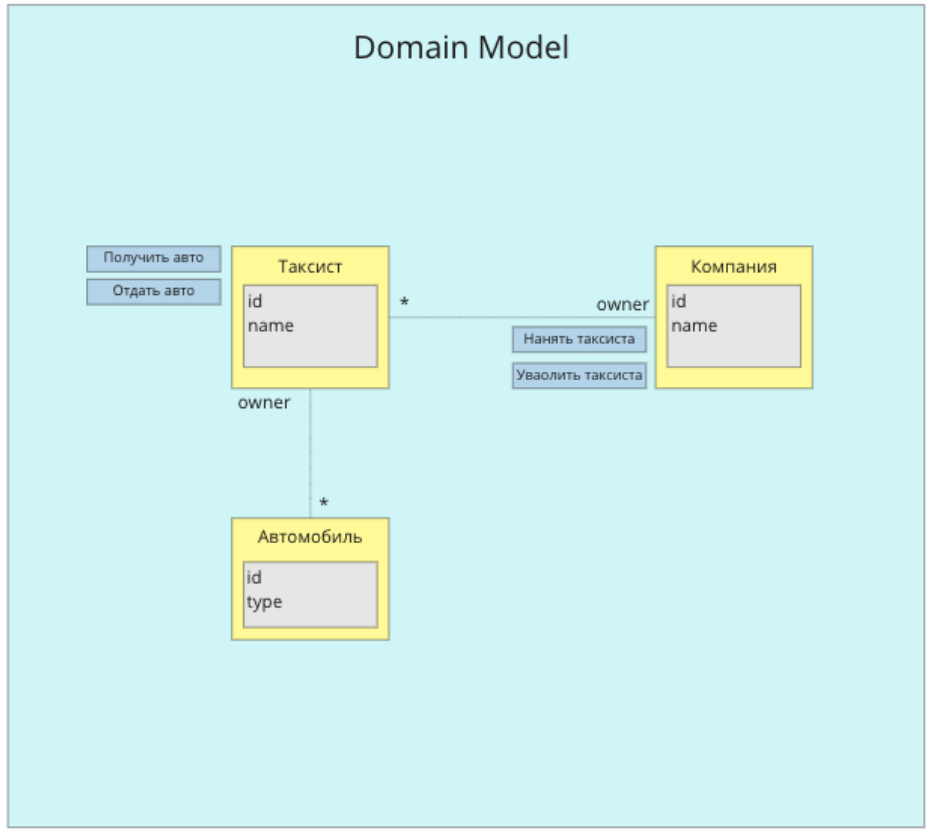
**Subdomain** обычно достаточно автономен.

SKLADCHIK.ORG

# Subdomain и Domain Expert


В разных **Subdomain** могут быть разные **Domain expert’ы.**

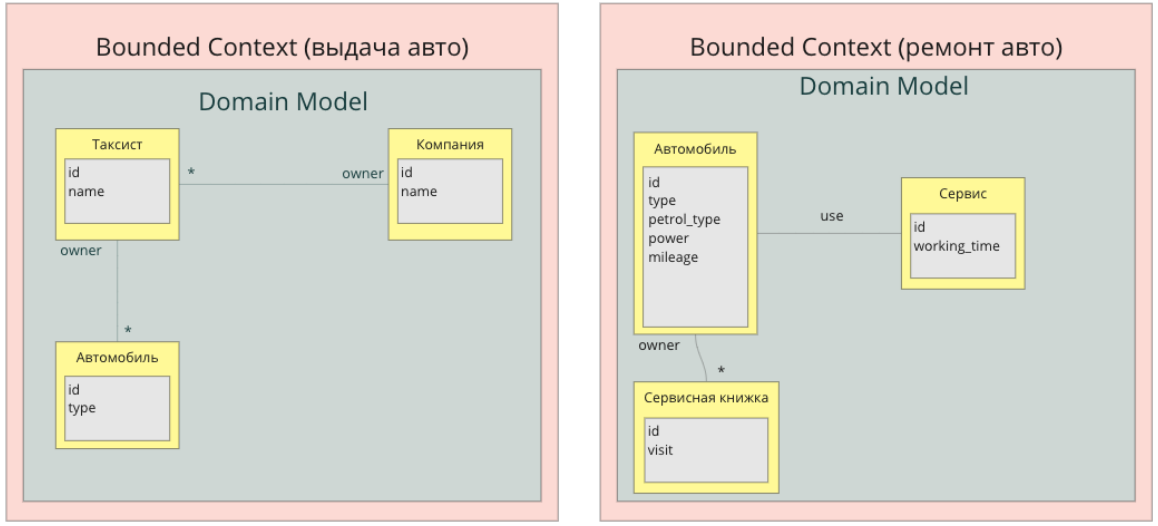
# Domain Model




**Domain Model -** это система абстракций, включающая только те аспекты предметной области, которые преобладают при решении конкретных бизнес-задачи.

**Domain Model -** это не модель реального мира.

# Bounded Context







**Bounded Context** — это граница, внутри которой существует Domain Model.

Если ваша Domain Model стала большой и противоречивой, то создайте вместо неё несколько отдельных непротиворечивых Domain Model. Каждая Domain Model - это **Bounded Context.**

# Subdomain и Bounded Context


**Subdomain** — это область проблемы.

**Bounded Context** — это область решения.

**Subdomain** — мы обнаруживаем.

**Bounded Context** — мы формируем**.**

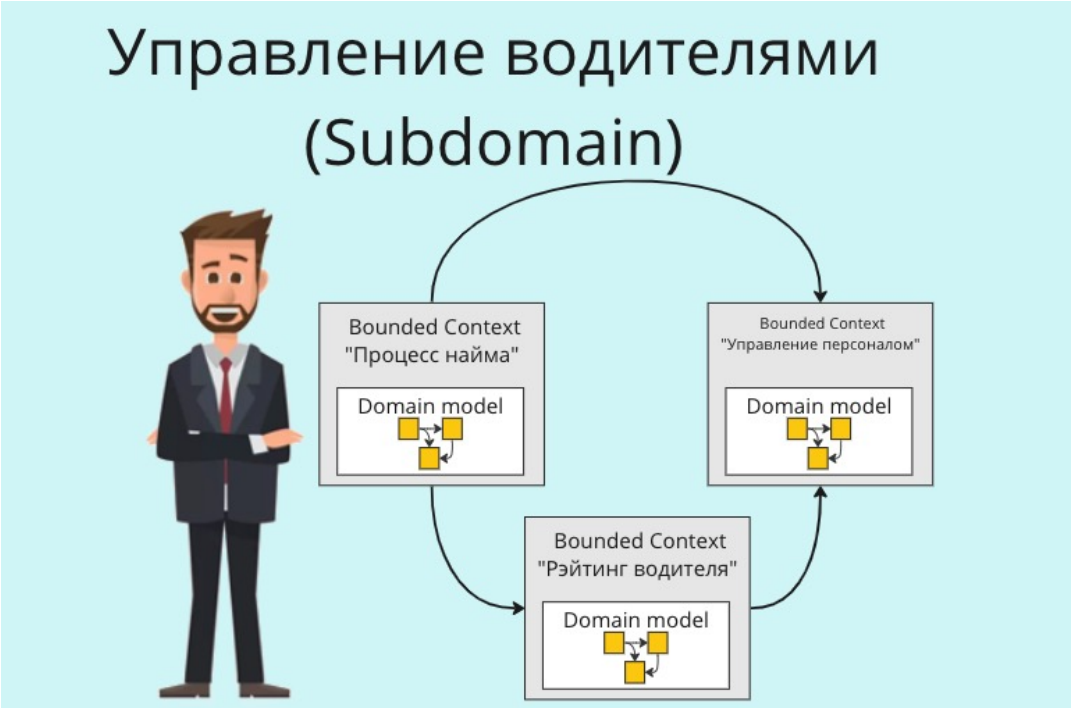
Обычно они соотносятся как 1 к 1. Но не всегда.

# Bounded Context и команды


**Bounded Context** — граница владения команды.

Над одним **Bounded Context** должна работать только одна команда.

# Subdomain и несколько Bounded Context



# Subdomain и несколько Bounded Context

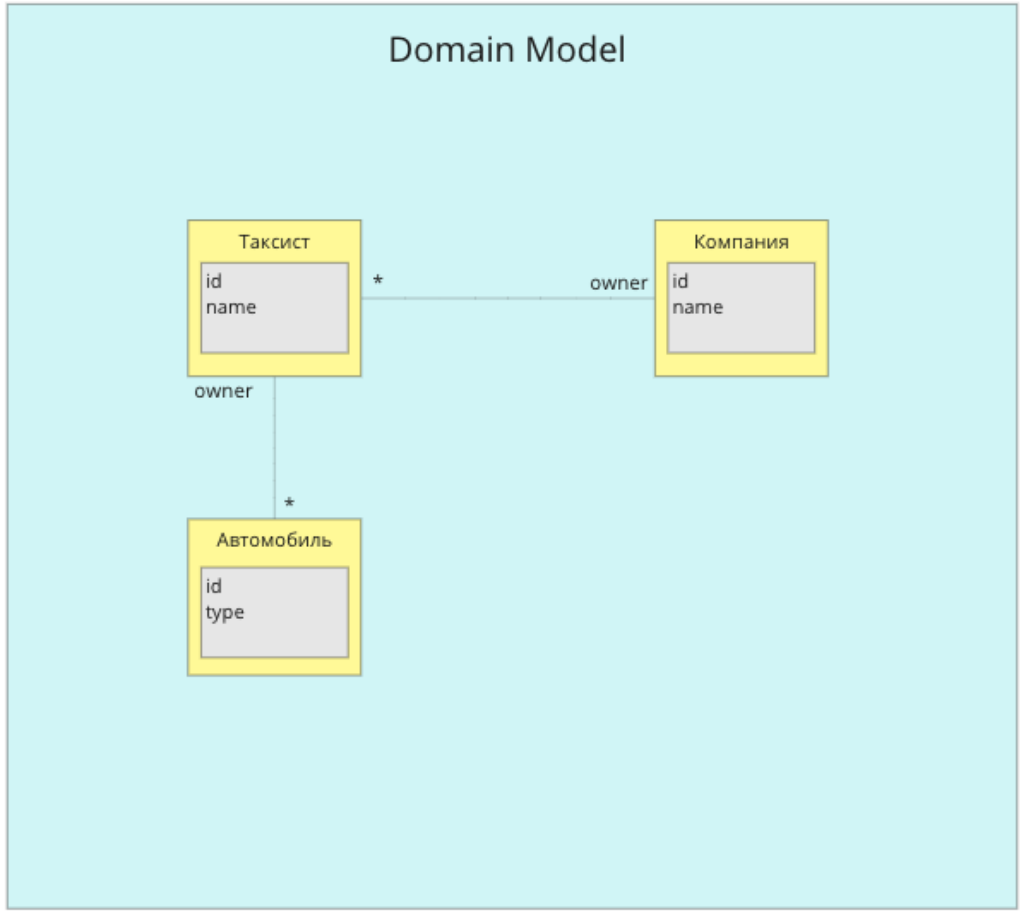
# Проблема единой Domain Model
# (что если не применять Bounded Context)

# Use case 1

Таксист компании может владеть автомобилем, если он ему выдан

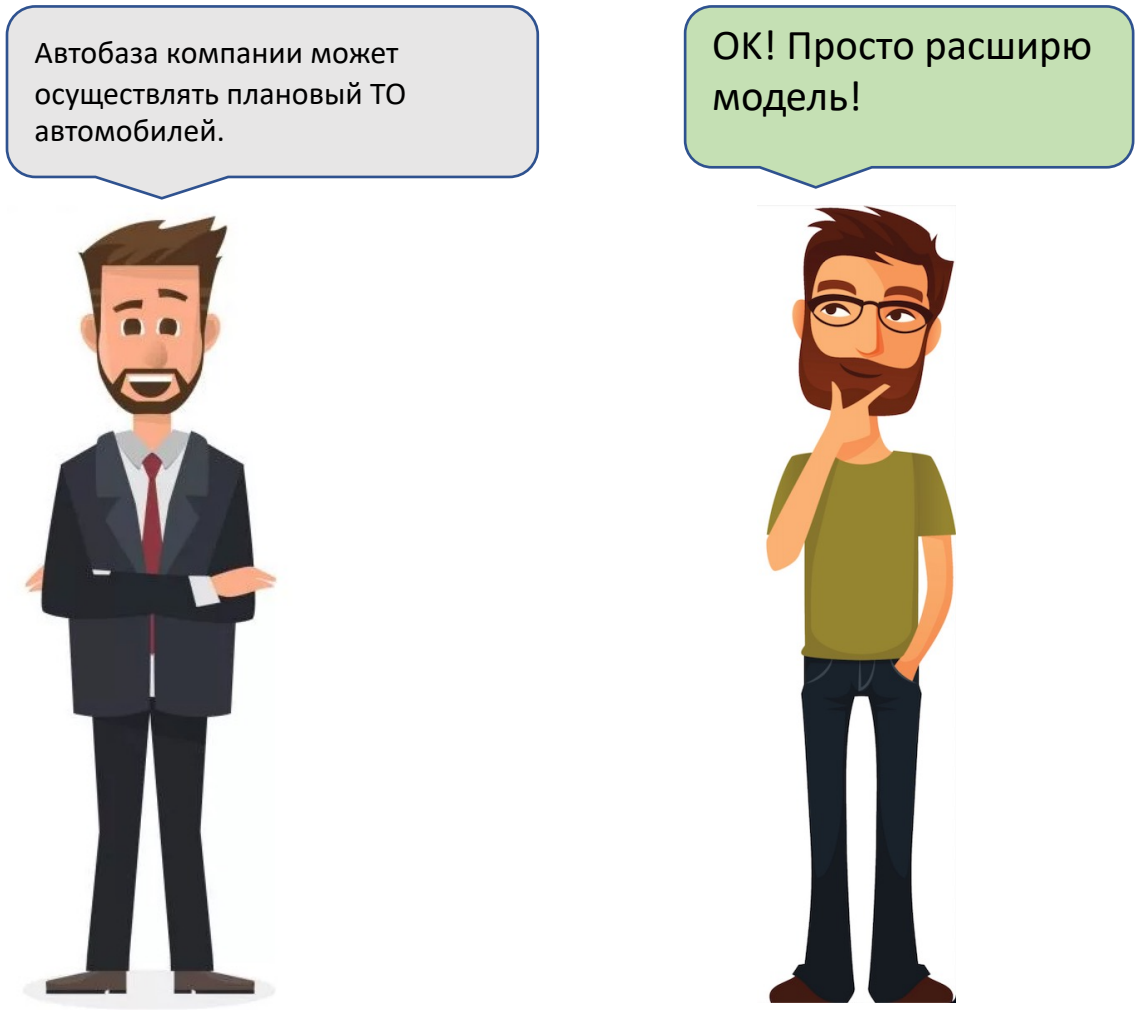
OK! Учту это в Domain Model

# Domain Model

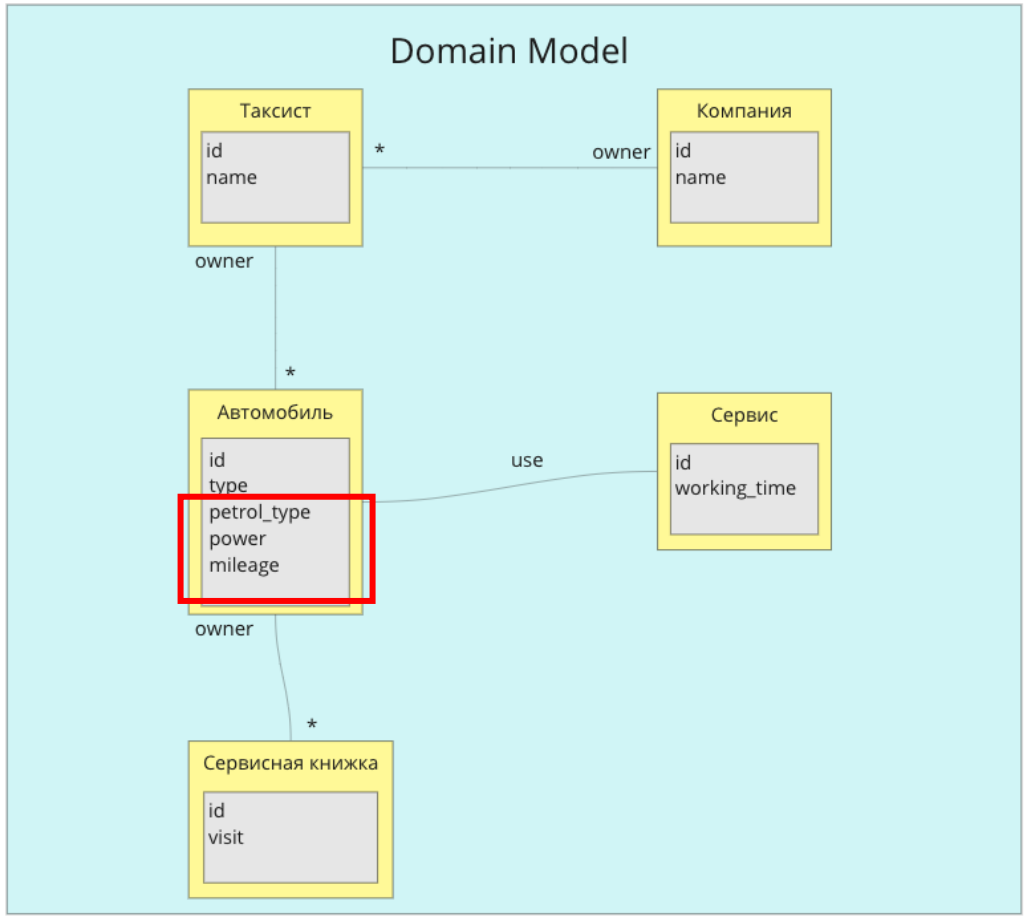



SKLADCHIK.ORG

# Use case 2



# Domain Model

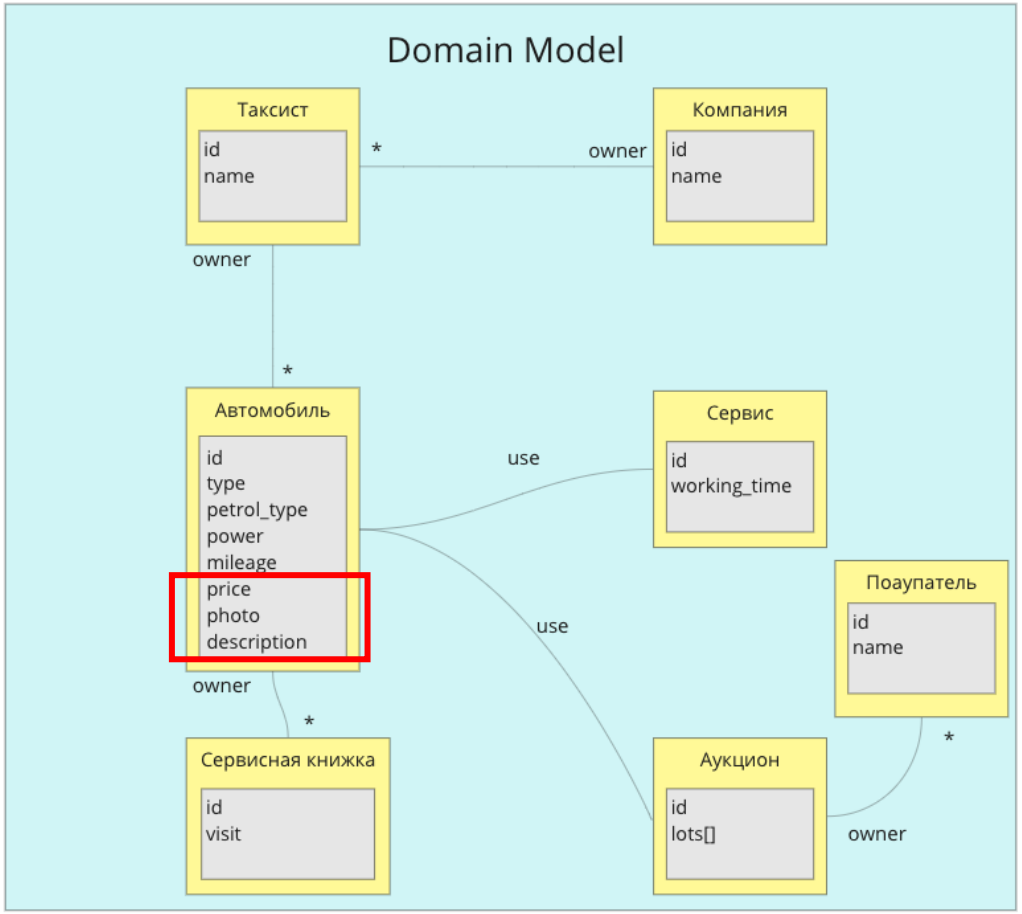



# Use case 3

Автомобили компании, которые старше 5 лет продаются на определенных условиях, а новые мы закупаем на аукционе

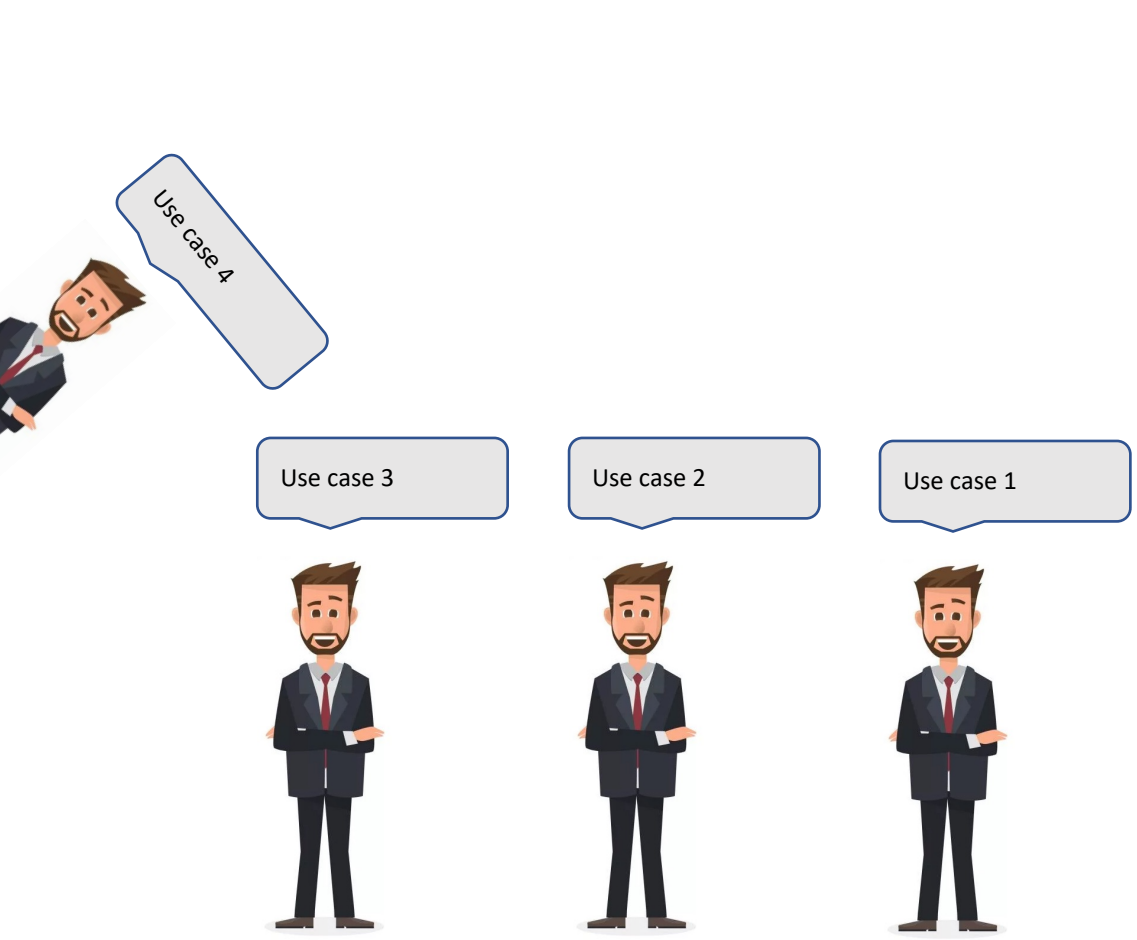
OK! Просто расширю модель!

# Domain Model

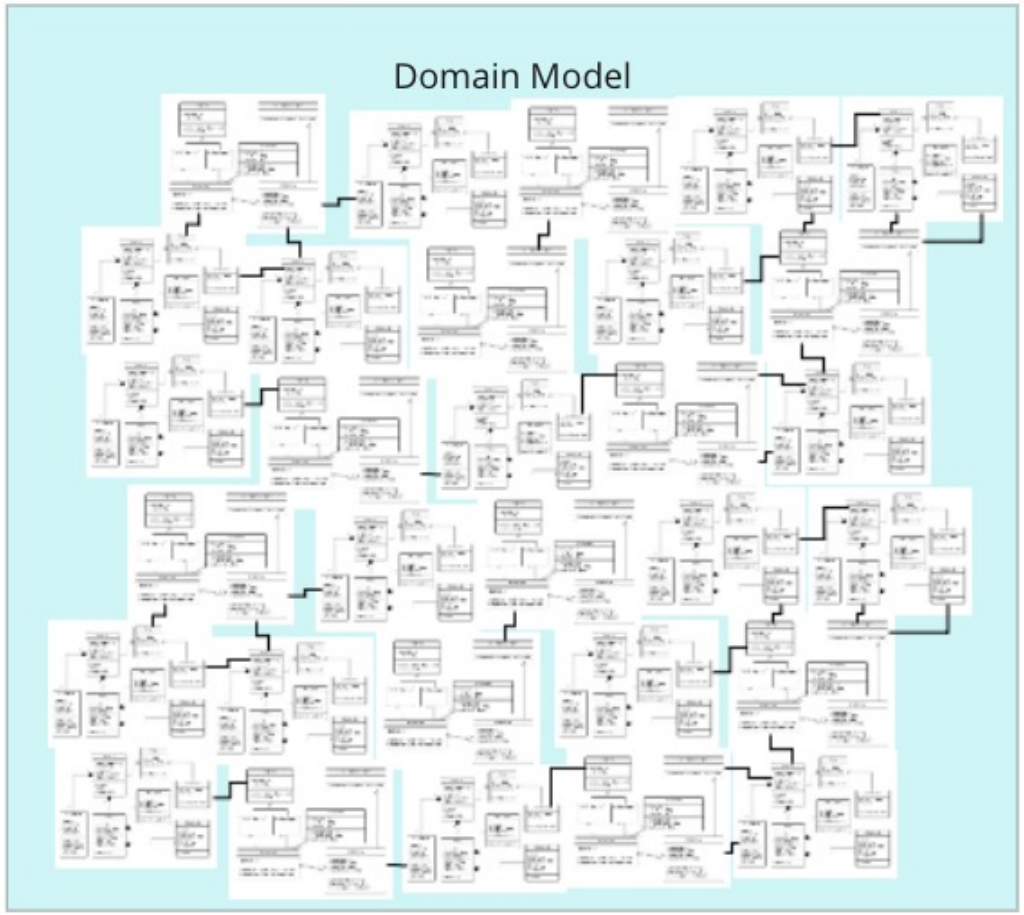



Готово!

# И таких Use case сотни…

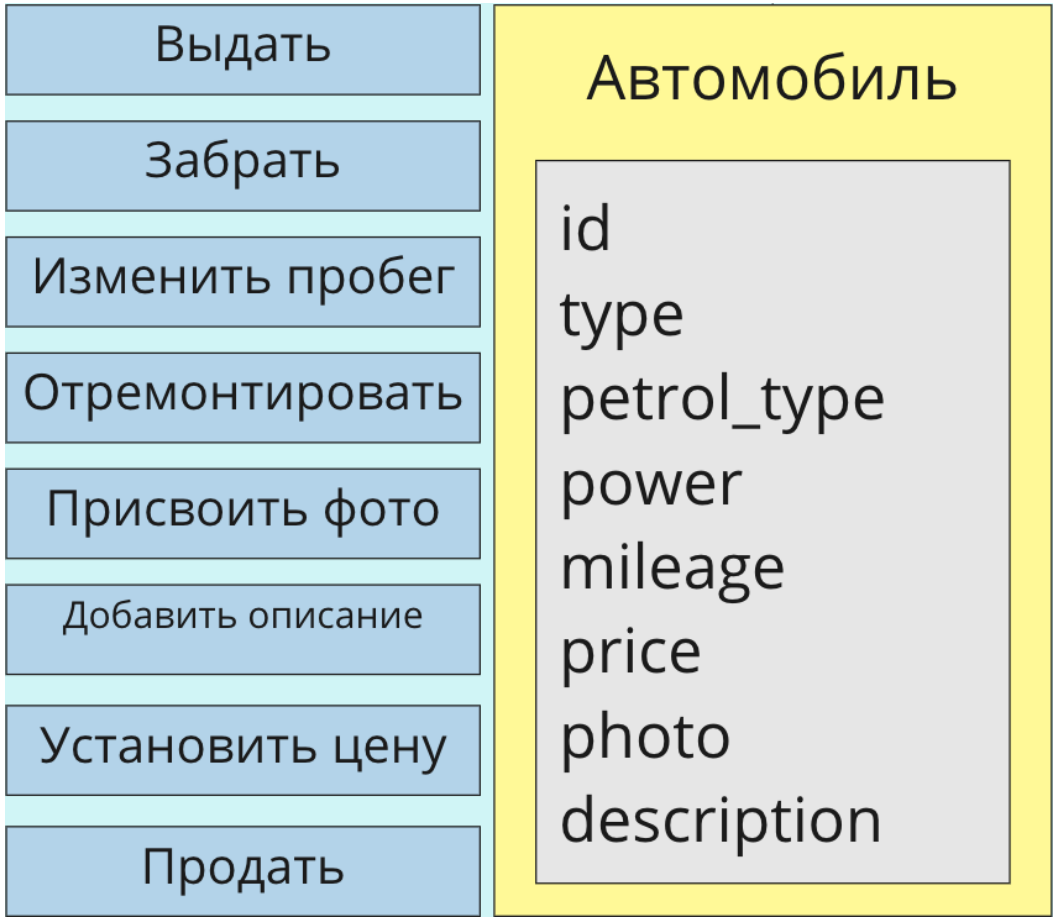


# Прошло пол года…




Что-то стало сложно..похоже у меня монолит типа *BBoM((*

# God Object




**God Object** - антипаттерн объектно-ориентированного программирования, описывающий объект, который хранит в себе «слишком много».

**SRP (Single responsibility principle)** - каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс.

**У нас проблема!**

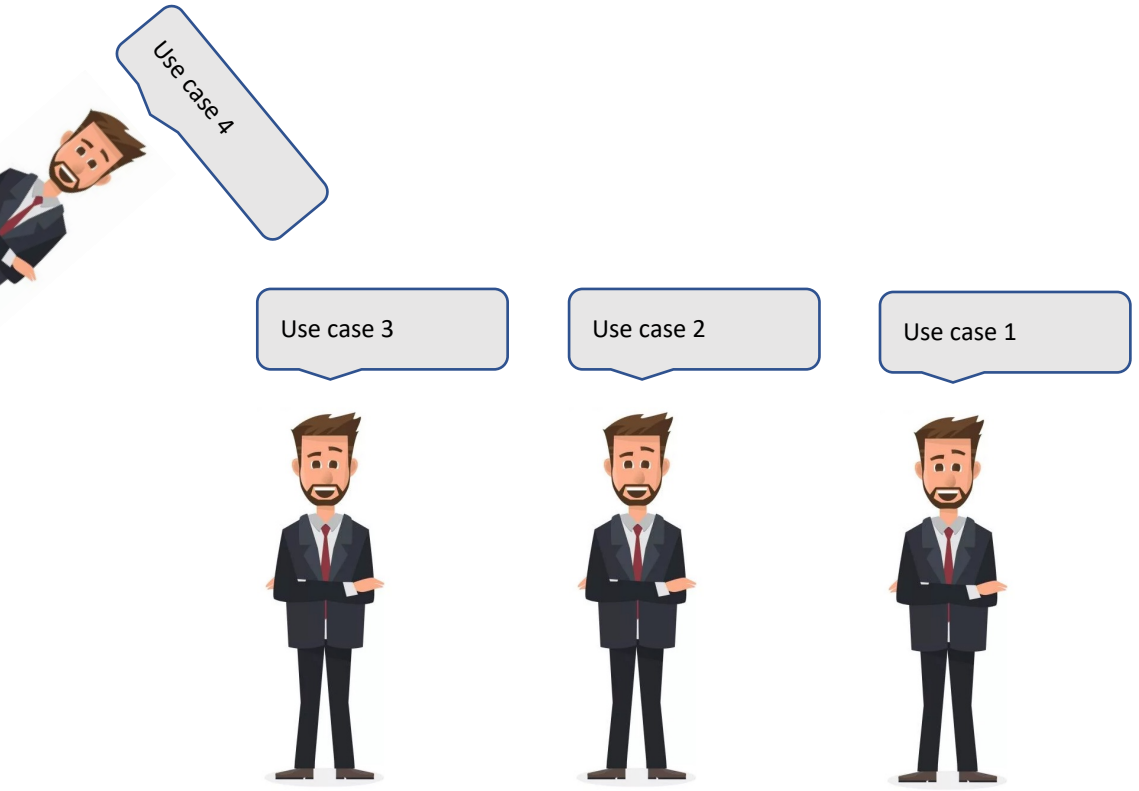
# Вывод

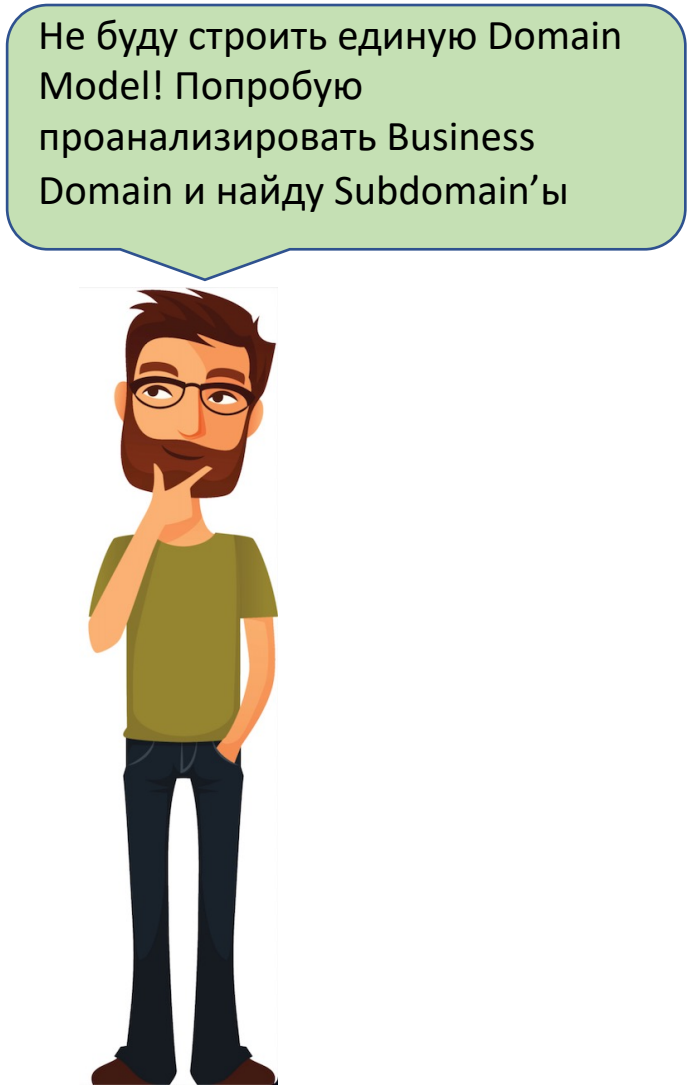
Попытка построить единую Domain Model - приводит к противоречивому и связанному коду.

Давайте строить N отдельных и непротиворечивых Domain Model, то есть выделять Bounded Context’ы.

Попробуем
применить DDD

# Use cases


Use case 4
Use case 3
Use case 2
Use case 1


Не буду строить единую Domain Model! Попробую проанализировать Business Domain и найду Subdomain’ы

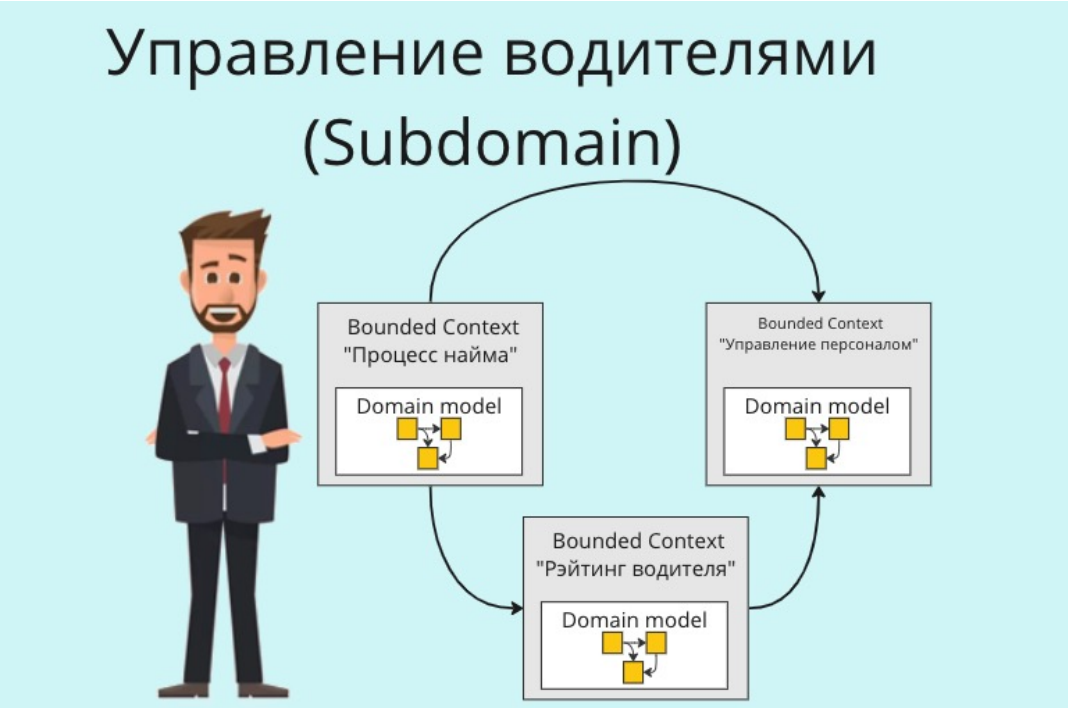
SKLADCHIK.ORG

# Нашел все Subdomain


# Domain Expert’ов может быть несколько

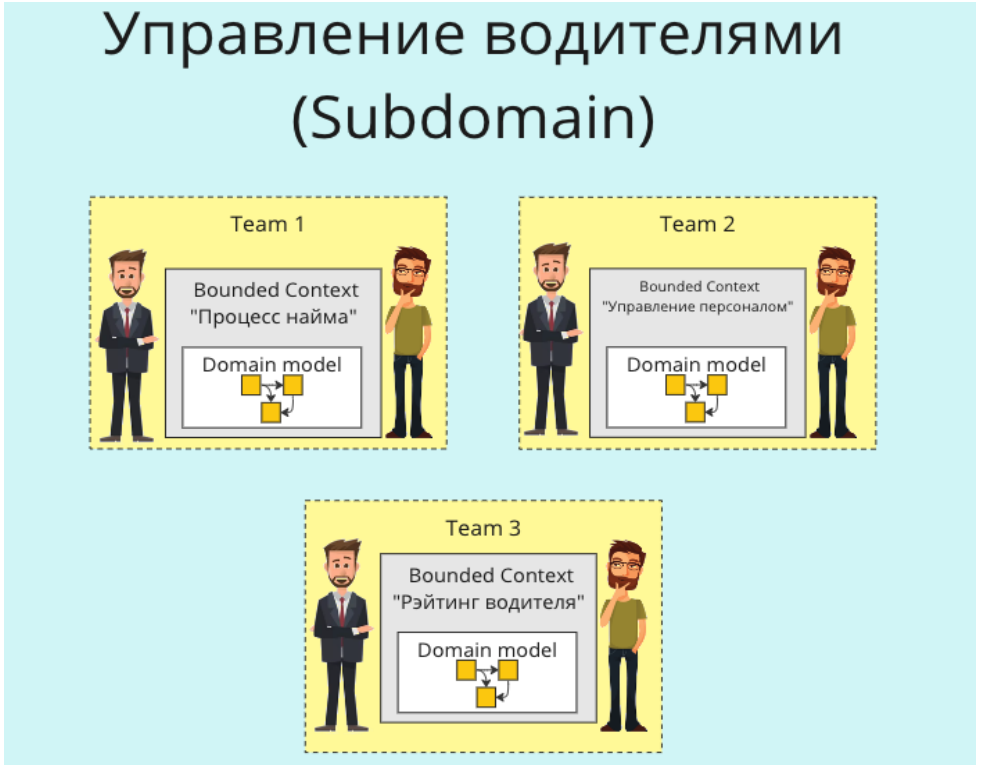

# Выделил Bounded Context


# Или Bounded Context поменьше




А для крупных Subdomain я, как архитектор, выделю несколько слабо связанных Bounded Context.

# Bounded Context и команды




Bounded Context теперь еще является границей ответственности команд. Я могу раздать разные Bounded Context – разным командам и работать параллельно!

# Избавляемся от God Object «Автомобиль»

# Автомобиль в разных Bounded Context

В контексте управления поездками
– это точка на карте

В контексте покупки/продажи
– это лот

В контексте сдачи в аренду
– это материальная ценность

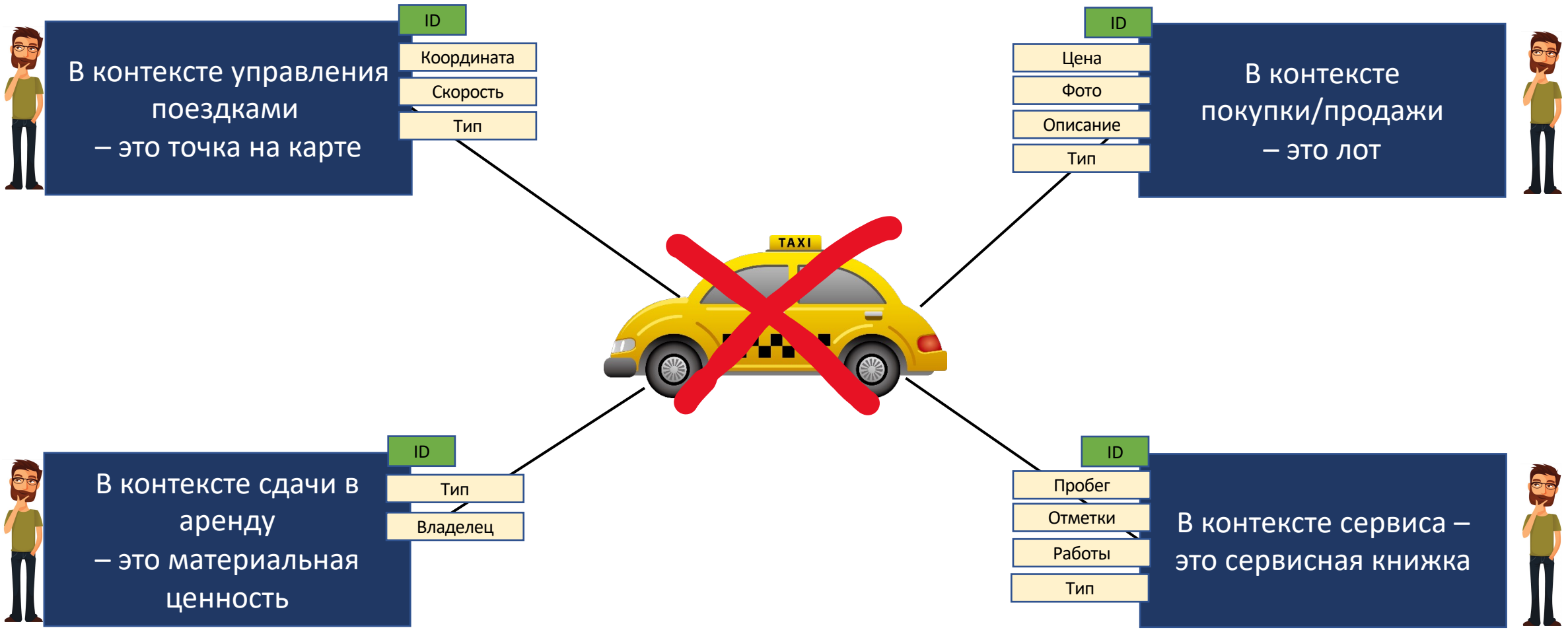
В контексте сервиса –
это сервисная книжка


Выдать
Забрать
Изменить пробег
Отремонтировать
Присвоить фото
Добавить описание
Установить цену
Продать

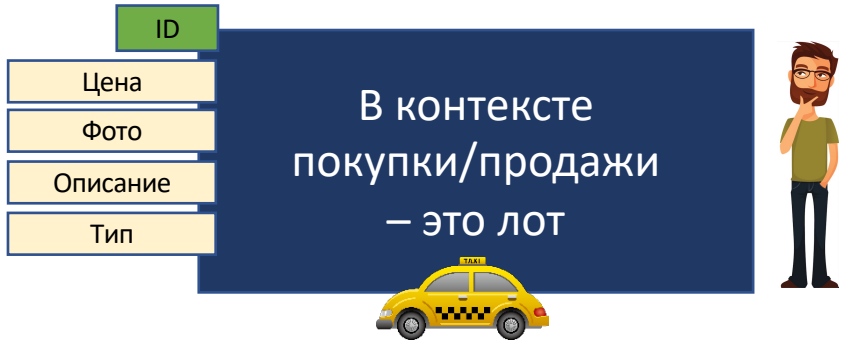
Автомобиль

id
type
petrol_type
power
mileage
price
photo
description

# Автомобиль в разных Bounded Context

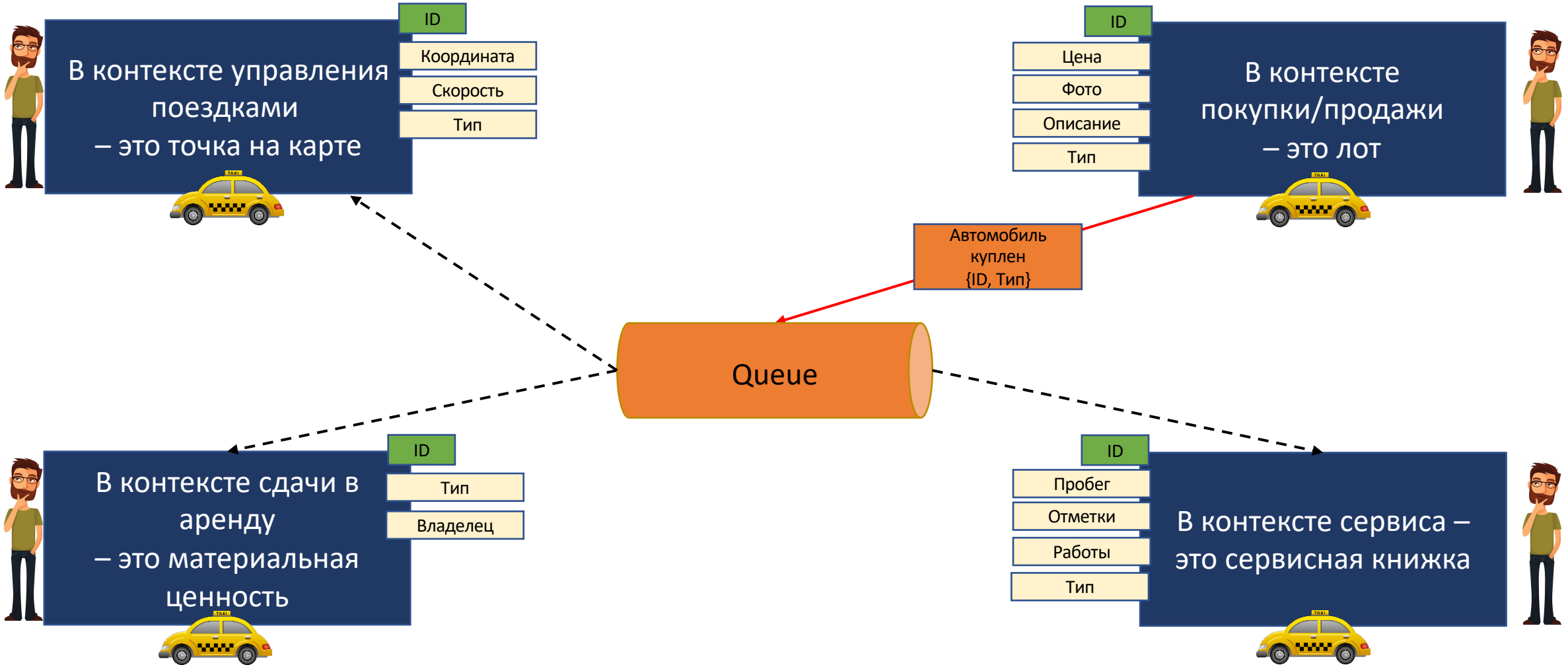


# Автомобиль в разных Bounded Context



# Связь Bounded Context



SKLADCHIK.ORG

# Тактические паттерны

# Aggregate


**Aggregate** - это кластер доменных сущностей, которые рассматриваются как единое целое с точки зрения изменения данных.

# Объект значения
# (Value Object)

# Value Object

```
public class Color
{
    public int R { get; set; }
    public int G { get; set; }
    public int B { get; set; }
}
```

**Value Object** — это объект, который можно идентифицировать по составу его значений.

# Пример

| red | green | blue |
|---|---|---|
| 255 | 255 | 0 |
| 0 | 128 | 128 |
| 0 | 0 | 255 |
| 0 | 0 | 255 |

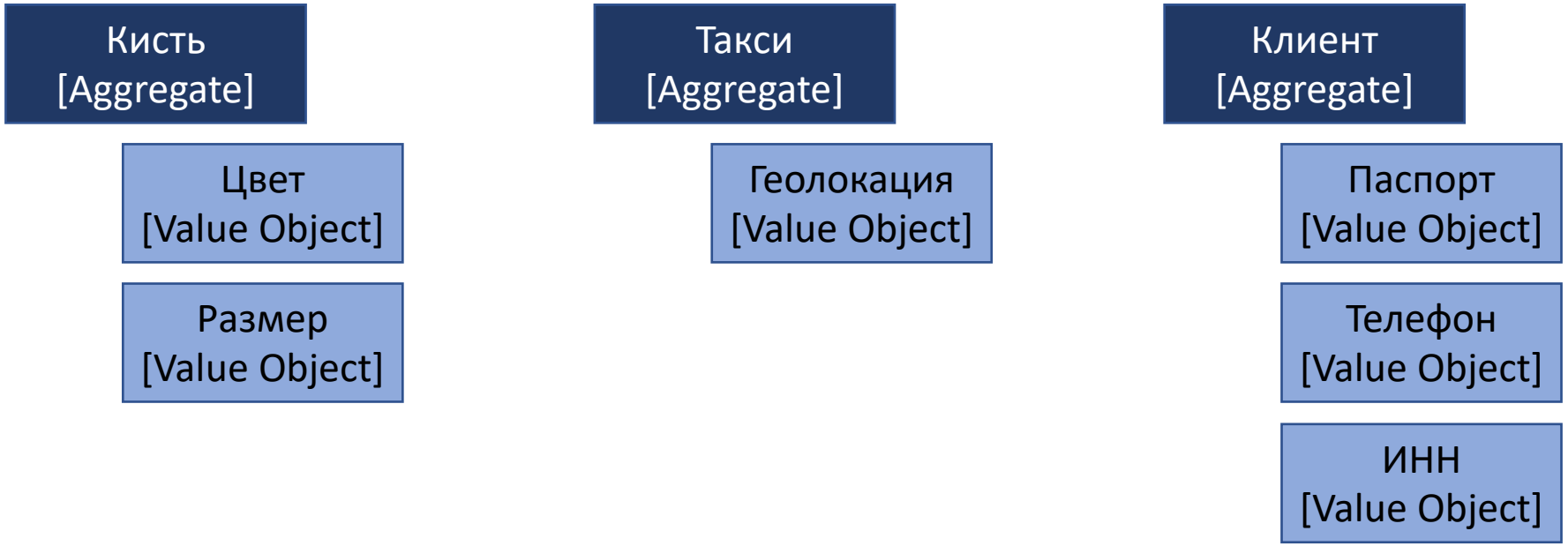
# Сущность с использованием Value Object

```
public class Brush
{
    public int ID { get; set; }
    public Size Size { get; set; }
    public Color Color { get; set; }
}

public class Size
{
    public int with { get; set; }
    public int height { get; set; }

    public static readonly Size Big = new Size(10,10);
    public static readonly Size Small = new Size(5,5);
}

public class Color
{
    public int R { get; set; }
    public int G { get; set; }
    public int B { get; set; }
}
```

```
Color.Parse("#CC00FF")
Color.IsDark(125,125,0)
Color grey = Color.Black + Color.White

Size size = Size.Big

Brush brush = new Brush(grey, size)
```

# Когда использовать Value Object?

Используйте Value Object для элементов домена, описывающих свойства других объектов.



# Сущность
# (Entity)

# Entity

```
0 references
public class Person
{
    1 reference
    public readonly Guid Id { get; set; }
    1 reference
    public Name Name { get; set; }

    0 references
    public new Person(Name name)
    {
        Id = Guid.NewGuid();
        Name = name;
    }
}
```

**Entity** является противоположностью Value Object. **Entity** требует явного идентификатора, чтобы различать разные экземпляры объекта.

Тривиальным примером сущности является человек.

# Пример

| ID | FirstName | LastName |
|---|---|---|
| 1 | Ivan | Ivanov |
| 2 | Denis | Petrov |
| 3 | Oleg | Smirnov |
| 4 | Oleg | Smirnov |

SKLADCHIK.ORG

# Когда использовать Entity

Используйте Entity для элементов домена, являющихся дочерними по отношению к Aggregate и идентичность которых определяется по ID, а не совокупности их свойств.

# Агрегат
# (Aggregate)

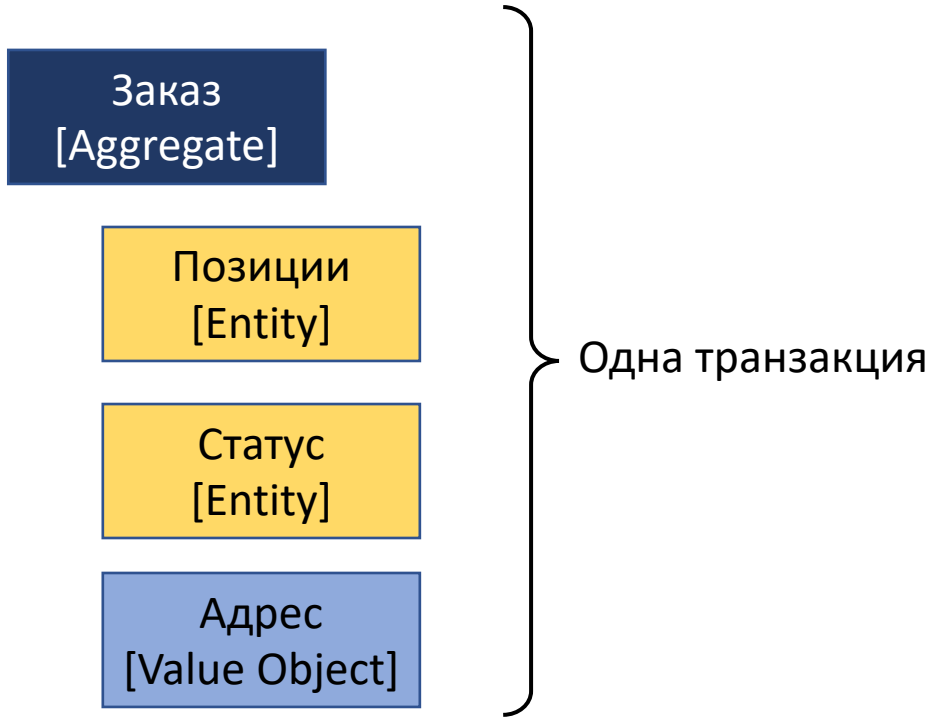
# Aggregate


**Aggregate** - это кластер доменных сущностей, которые рассматриваются как единое целое с точки зрения изменения данных.

# Aggregate отвечает за соблюдение инвариантов

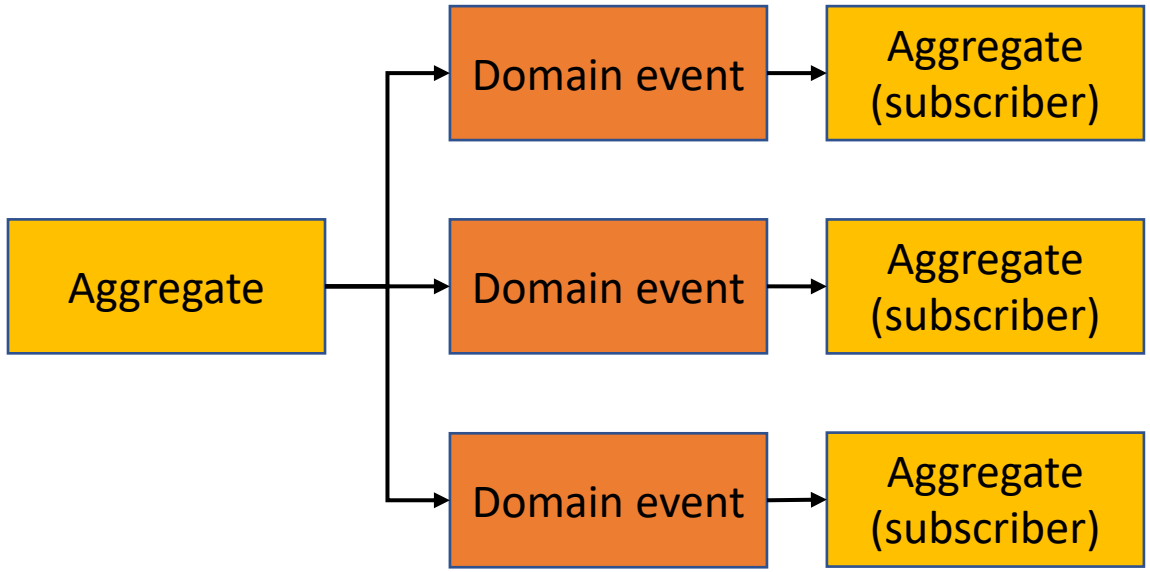
```
0 references
public class Order
{
    1 reference
    public readonly Guid Id { get; set; }
    1 reference
    public Guid BuyerId { get; set; }
    0 references
    public Item[] Items { get; set; }

    1 reference
    public Address Address { get; set; }
    6 references
    public Status Status { get; set; }

    0 references
    public new Order(Buyer buyer)
    {
        Id = Guid.NewGuid();
        BuyerId =  buyer.Id;
        Status = Status.New;
    }

    0 references
    public new AddAddress(Address address)
    {
        Address = address;
        Status = Status.Filled;
    }

    0 references
    public new Checkout()
    {
        if(items.count==0)
        {
            throw new Exception();
        }
        //логика рассчета
        //...
        //конец логики расчета
        Status = Status.Checkout;
    }
}
```

# Aggregate - граница транзакции



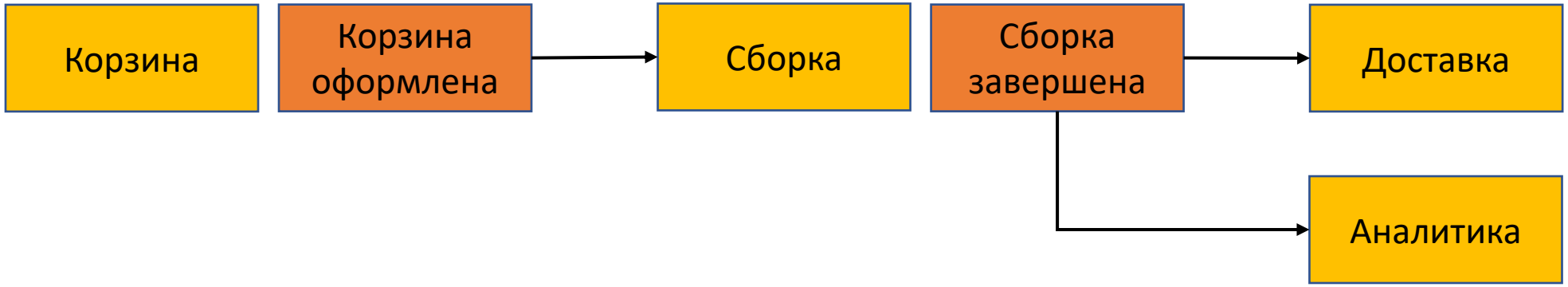
# Доменное событие
# (Domain Event)

# Domain Event

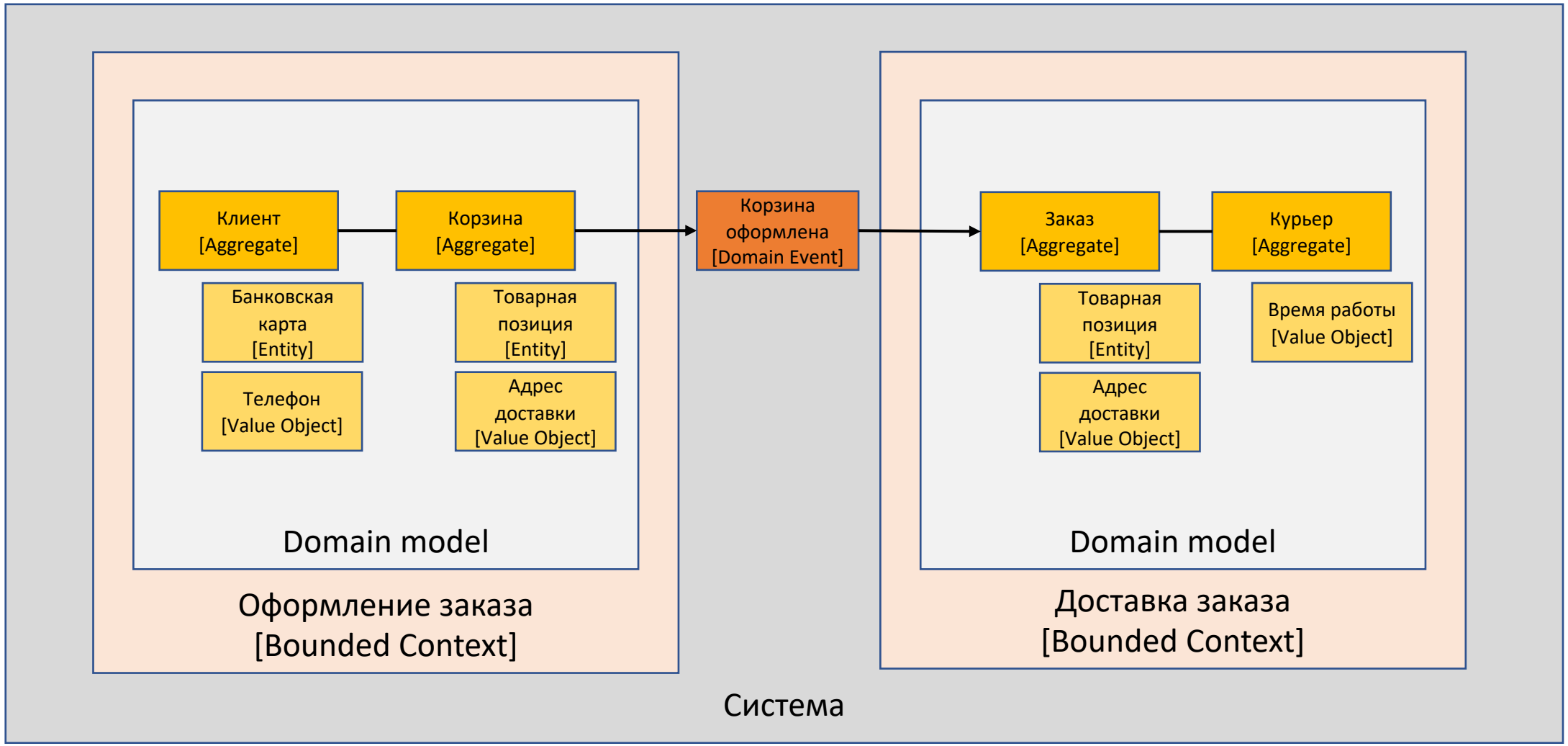



**Domain Event** — это то, что произошло в прошлом. Логически, событие предметной области — это то, что произошло в конкретной предметной области, и то, о чем должны быть в курсе и на что должны реагировать другие части системы.

# Пример Domain Event



# Итог


Клиент
[Aggregate]
Корзина
[Aggregate]
Банковская
карта
[Entity]
Товарная
позиция
[Entity]
Телефон
[Value Object]
Адрес
доставки
[Value Object]
Domain model
Оформление заказа
[Bounded Context]
Корзина
оформлена
[Domain Event]
Заказ
[Aggregate]
Курьер
[Aggregate]
Товарная
позиция
[Entity]
Время работы
[Value Object]
Адрес
доставки
[Value Object]
Domain model
Доставка заказа
[Bounded Context]
Система

# О чем узнали

- Слабая связь между сервисами – основной критерий успешной декомпозиции
- В основе декомпозиции системы на микросервисы лежит Domain Driven Design
- Bounded Context – способ борьбы со сложностью.  Множество Bounded Context удобнее чем единая (большая и противоречивая) Domain Model

SKLADCHIK.ORG

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

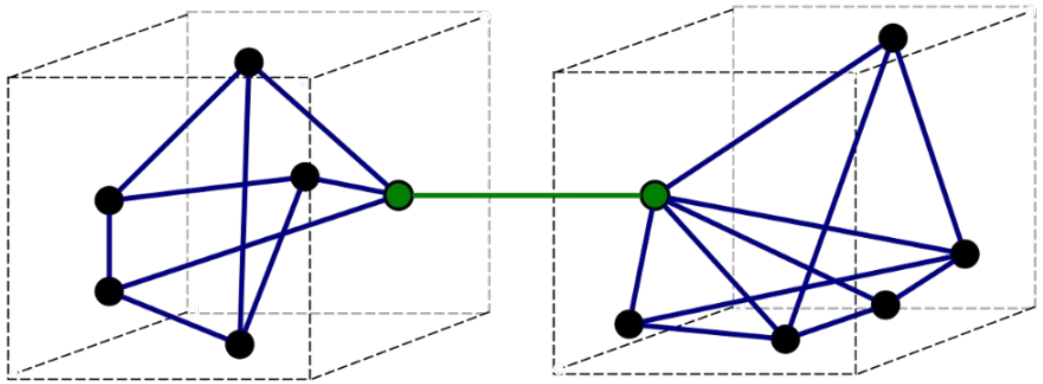

# Вебинар

# Цели занятия

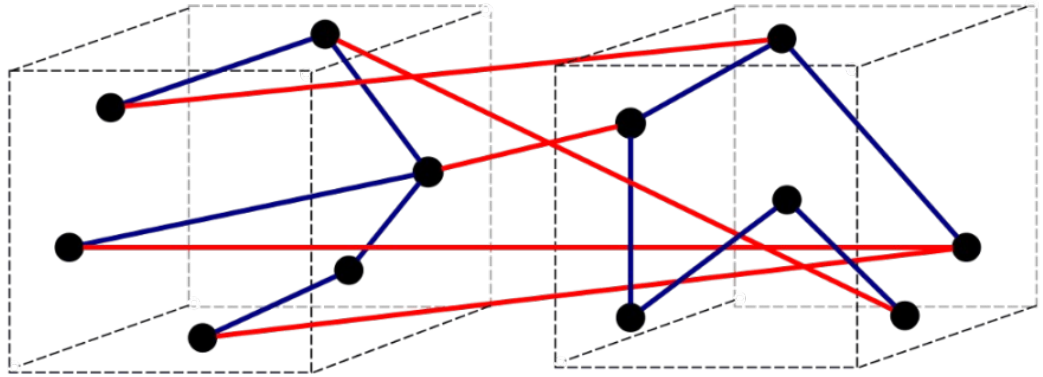
- Разобраться, что есть микросервис и какого он размера
- Понять оптимальное количество микросервисов в команде
- Разобраться, какой основной критерий хорошей декомпозиции
- Познакомится с Event Storming и научиться его проводить

# Декомпозиция на микросервисы

# Правило хорошей декомпозиции

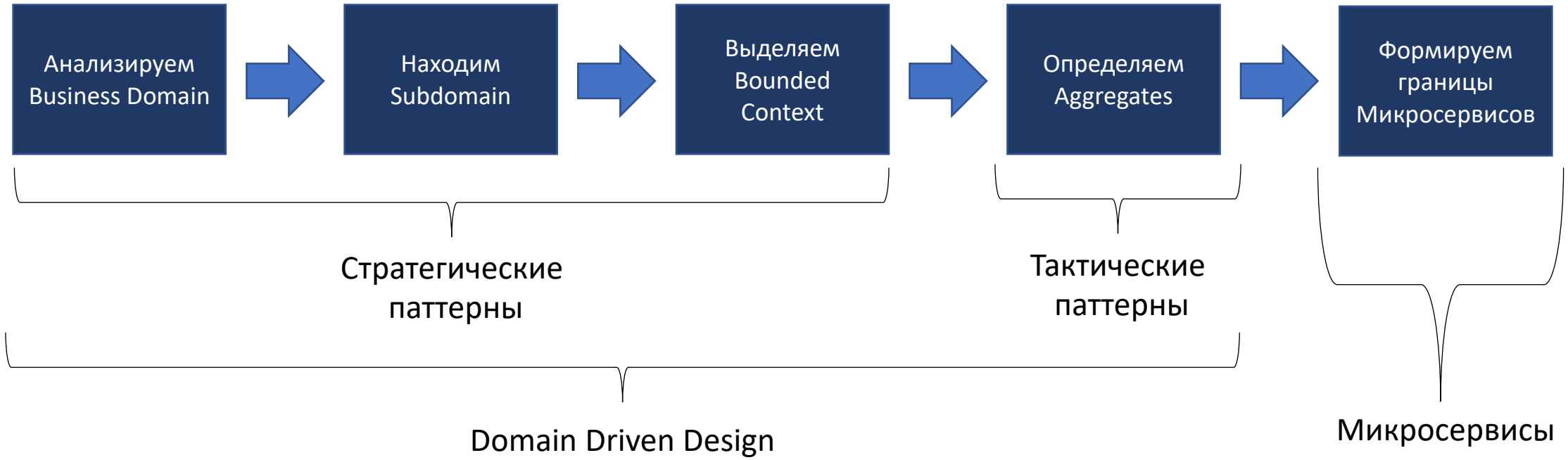


a) Low Coupling (низкая связанность) и High Cohesion (высокое зацепление)

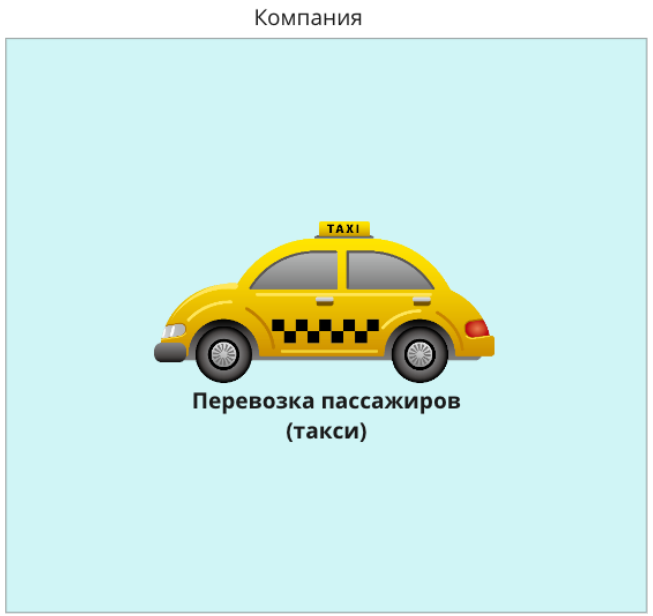


b) High Coupling (высокая связанность) и Low Cohesion (низкое зацепление)

# Стратегия выделения микросервисов



# 1. Анализируем Business Domain



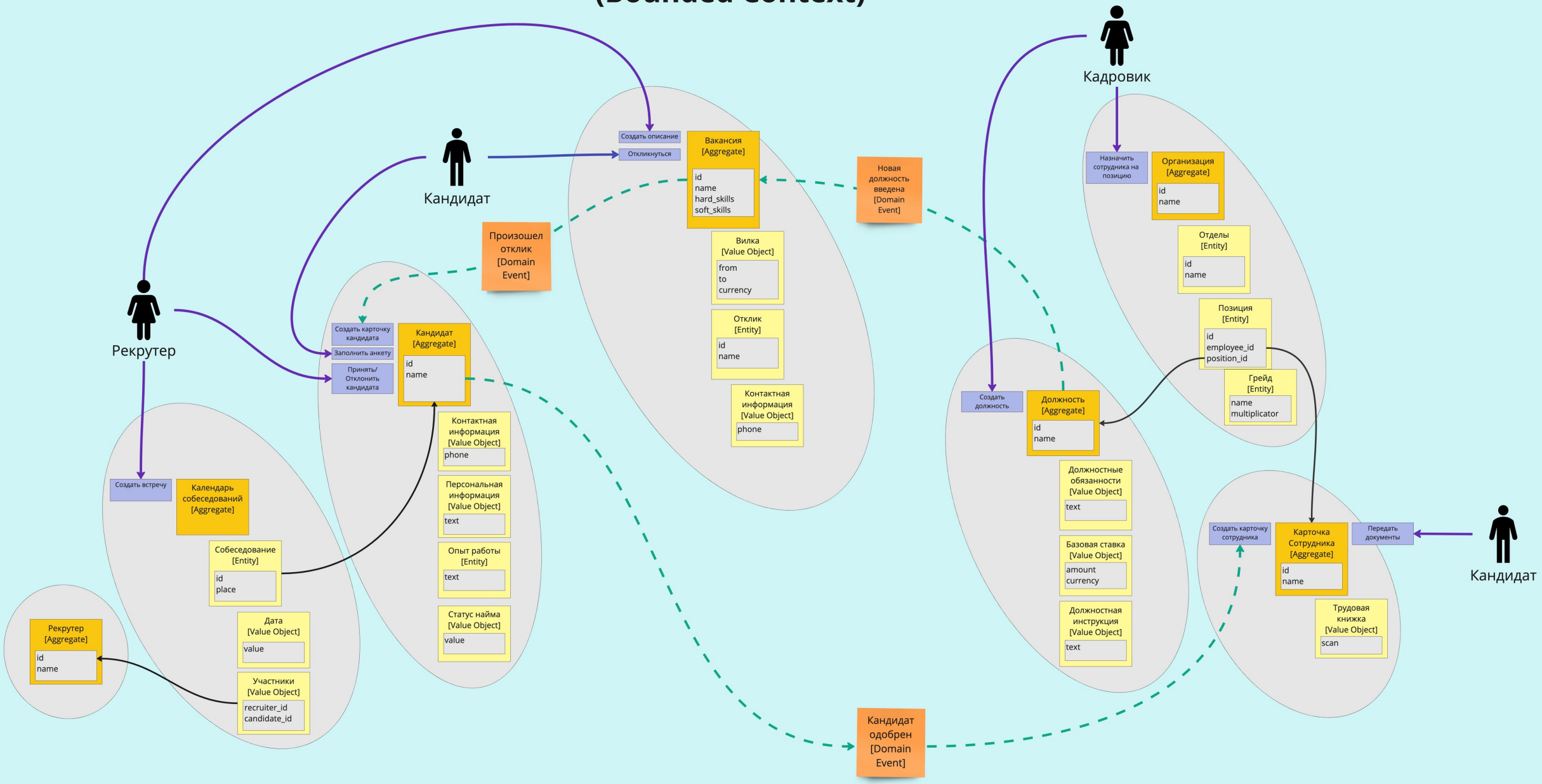

# 2. Находим Subdomain


# 3. Выделяем Bounded Context

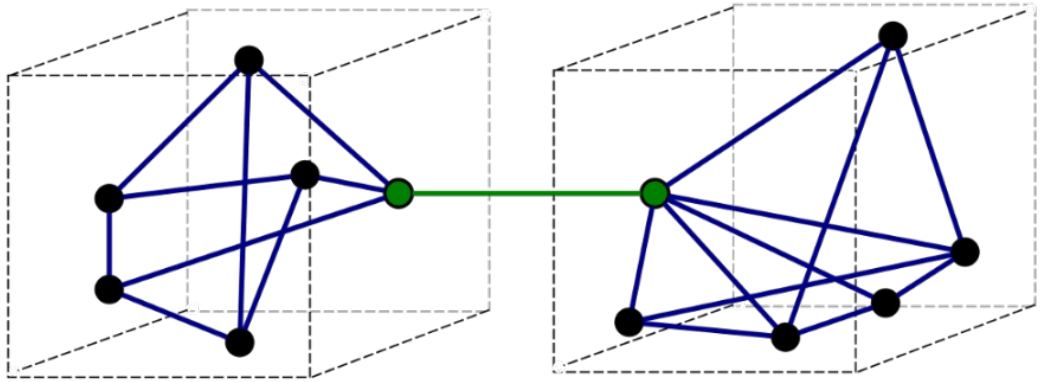

# 4. Определяем Aggregates


SKLADCHIK.ORG

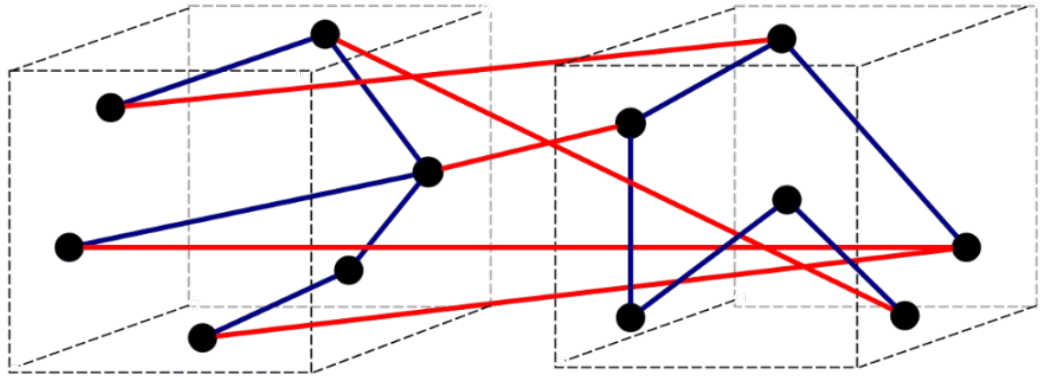
# Процесс найма (Bounded Context)



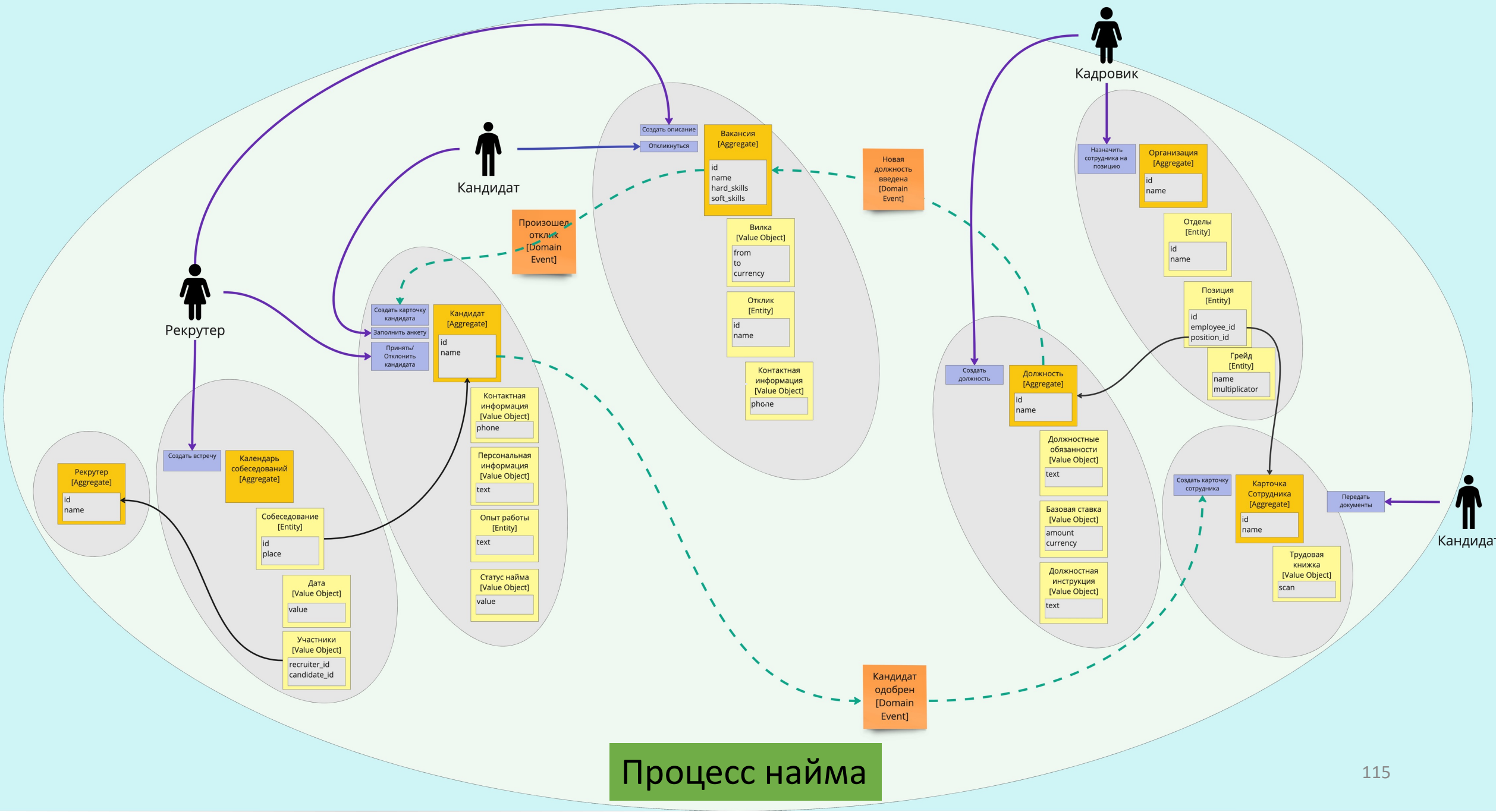
# 5. Формируем границы Микросервисов



a) Low Coupling (низкая связанность) и High Cohesion (высокое зацепление)

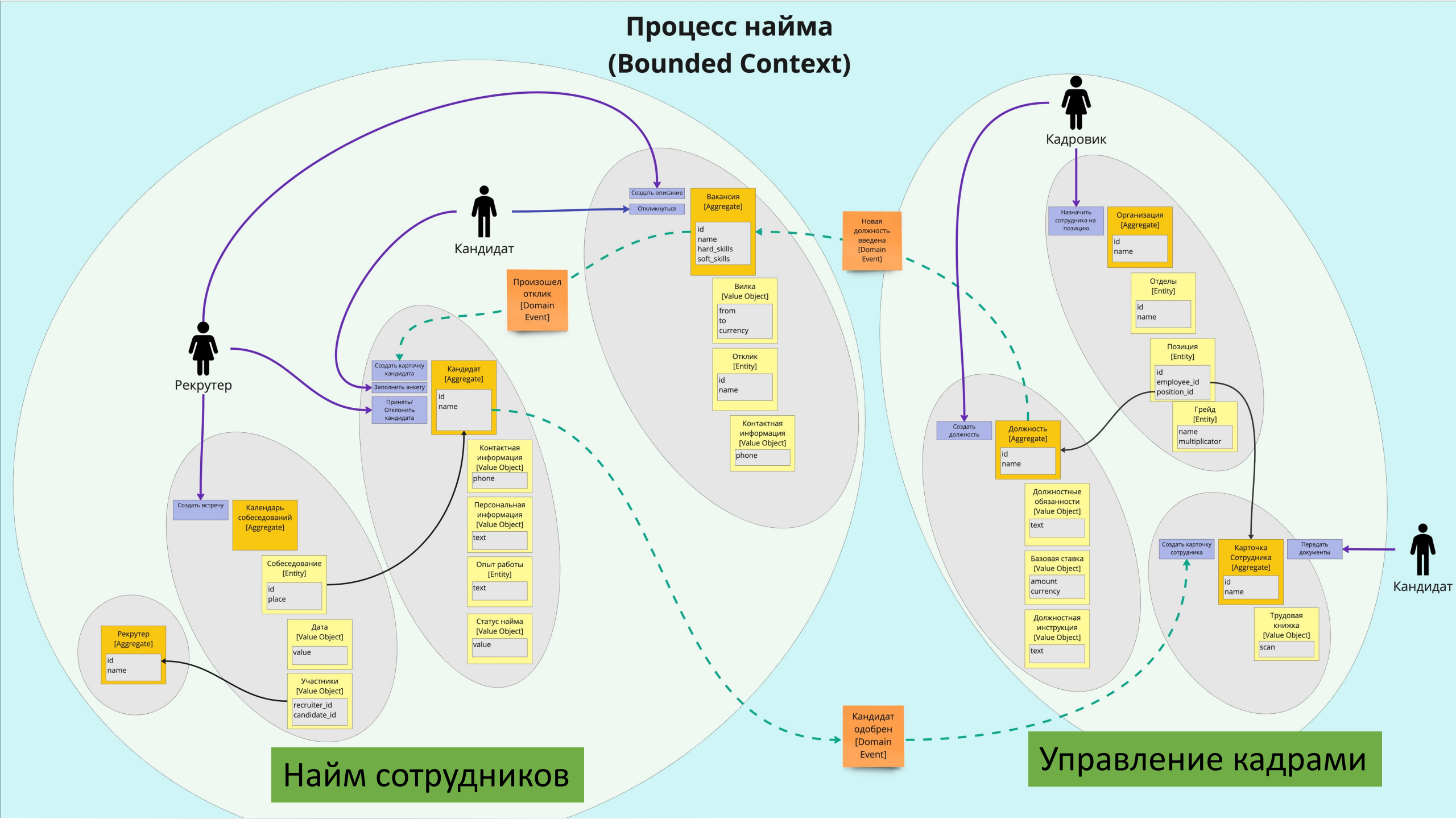


b) High Coupling (высокая связанность) и Low Cohesion (низкое зацепление)

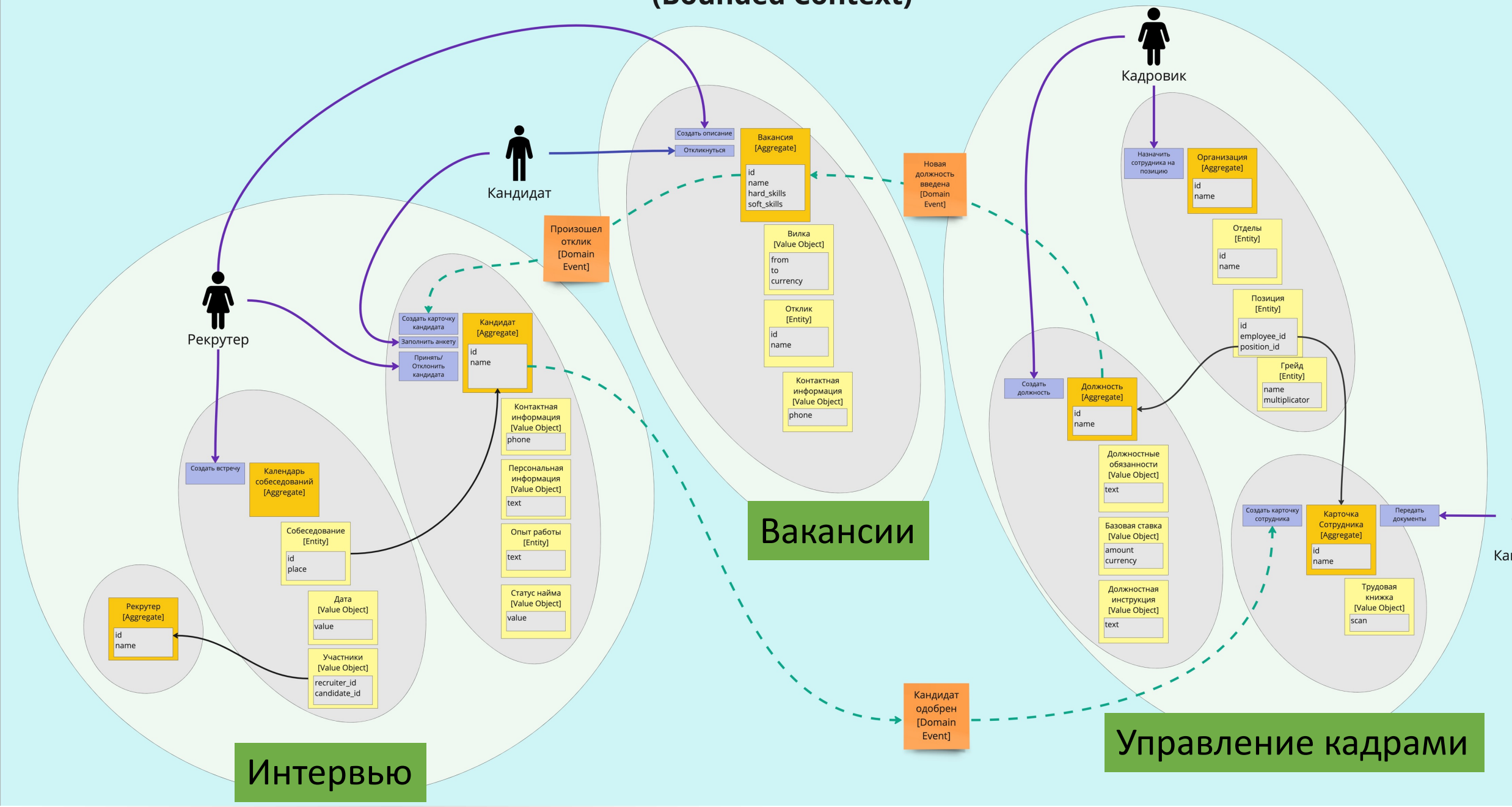



# Процесс найма

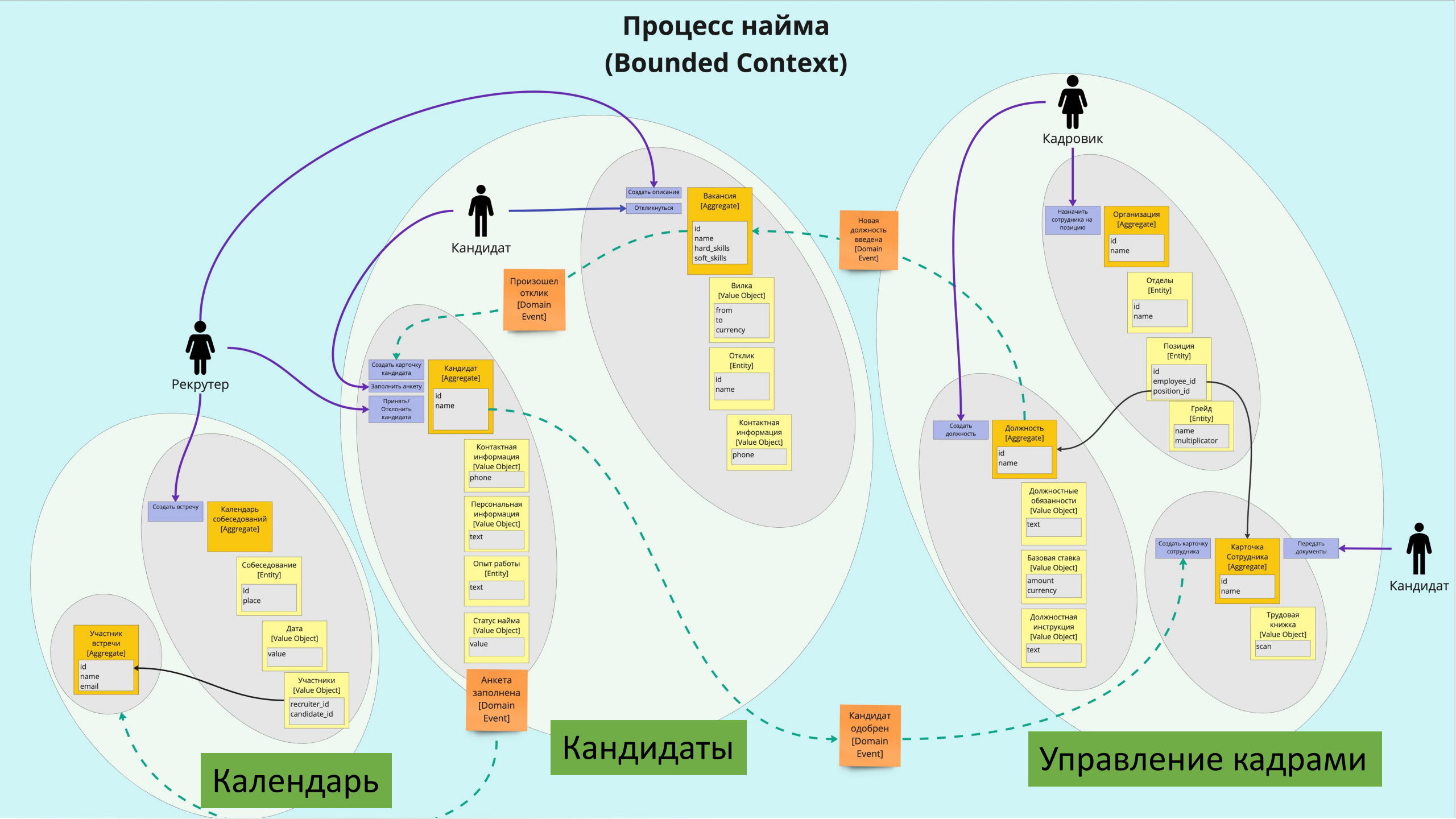
# Процесс найма
# (Bounded Context)



# Процесс найма
# (Bounded Context)

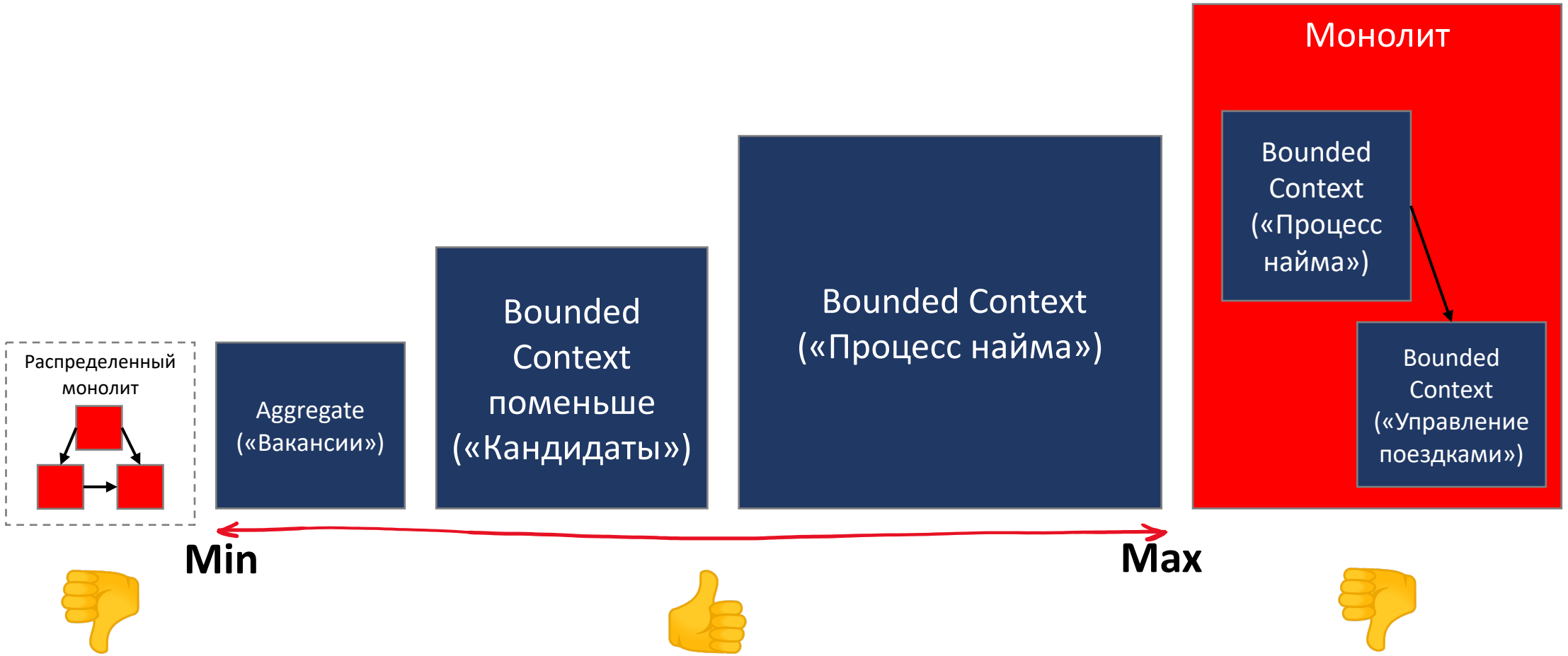


# Процесс найма
# (Bounded Context)



# Допустимый размер микросервиса

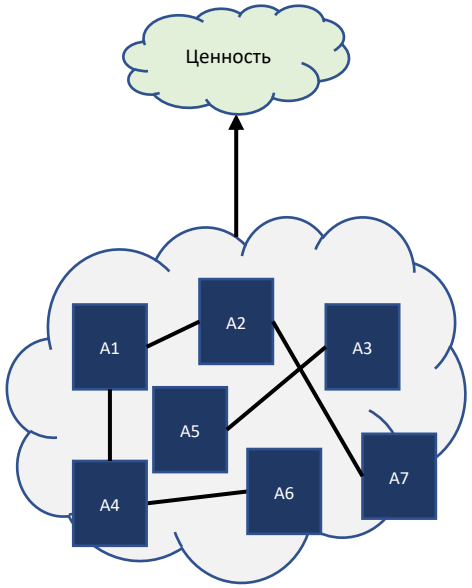
# Размер микросервиса



# Размер микросервиса и команда

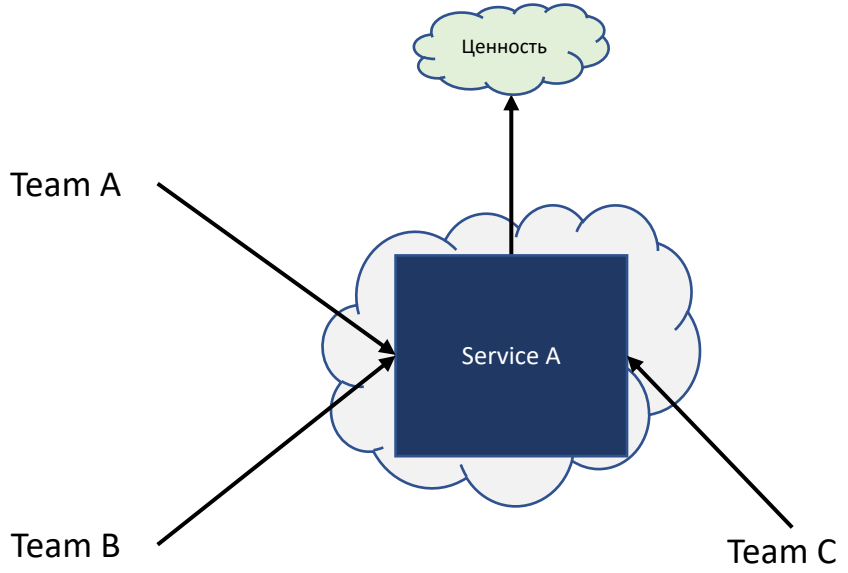
# Много маленьких или один огромный

## Много маленьких



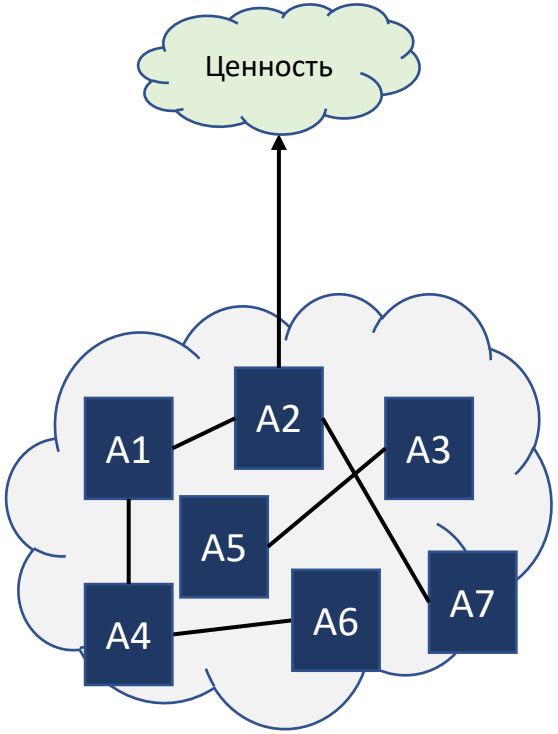

Team A

## 1 огромный



SKLADCHIK.ORG

# Когда у одной команды много сервисов




Team A

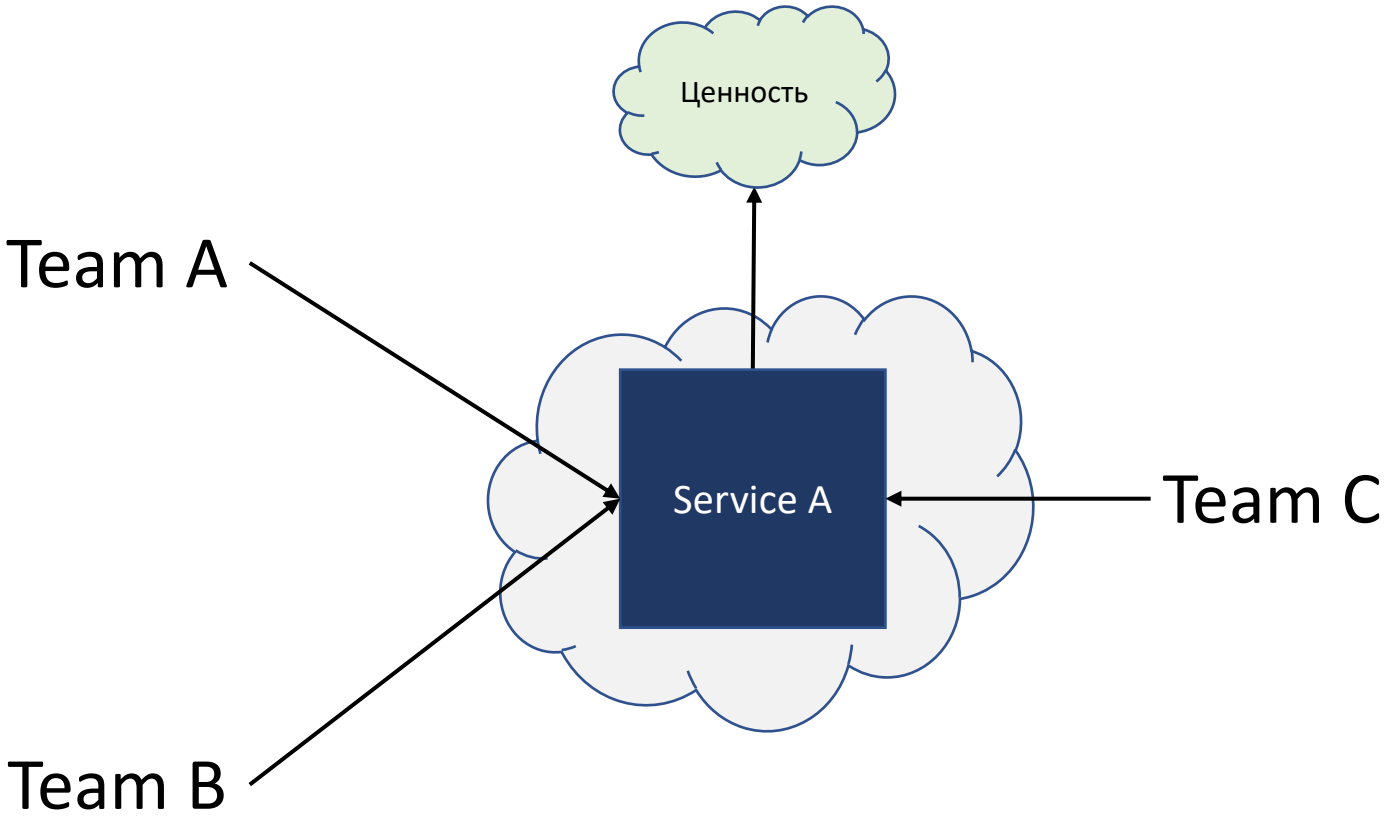
**Потери:**

1. Инициализация проекта(git, ci/cd)
2. Базовая архитектура
3. Логирование(код + elk)
4. Мониторинг(Prometheus + Grafana + Slack)
5. Развертывание тест
6. Развертывание прод
7. Документация
8. Реализация базовых конструкций(API, обработка ошибок и тп)
9. Инициализация тестового фреймворка

# Вывод

Lead Time растет т.к. каждый дополнительный сервис добавляет сложности и накладных расходов.

# Когда 1 сервис на несколько команд



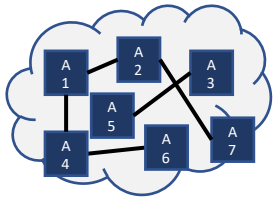

**Потери:**

1. Ожидание доработок от других команд
2. Затраты на коммуникацию(синхронизацию)
3. Затраты на ожидание стабилизации
4. Затраты на ожидание принятия пул реквеста

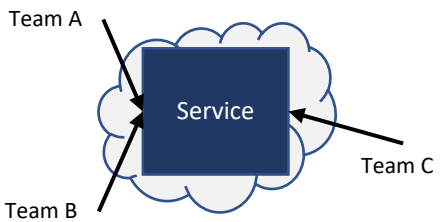
# Вывод

Lead Time растет так как каждая дополнительная команда добавляет сложности коммуникации.

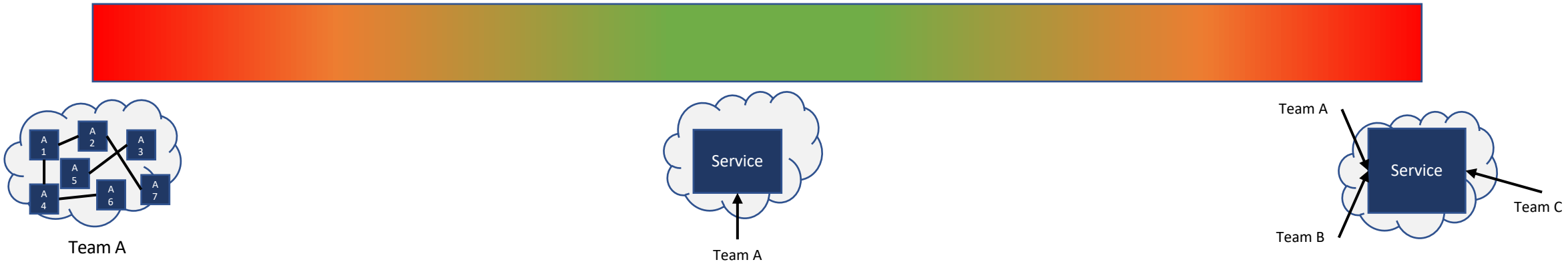
# Количество сервисов в 1 команде




Team A



# Количество сервисов в 1 команде



# Вывод

Размер микросервиса = объему кода, который в состоянии создать и комфортно развивать

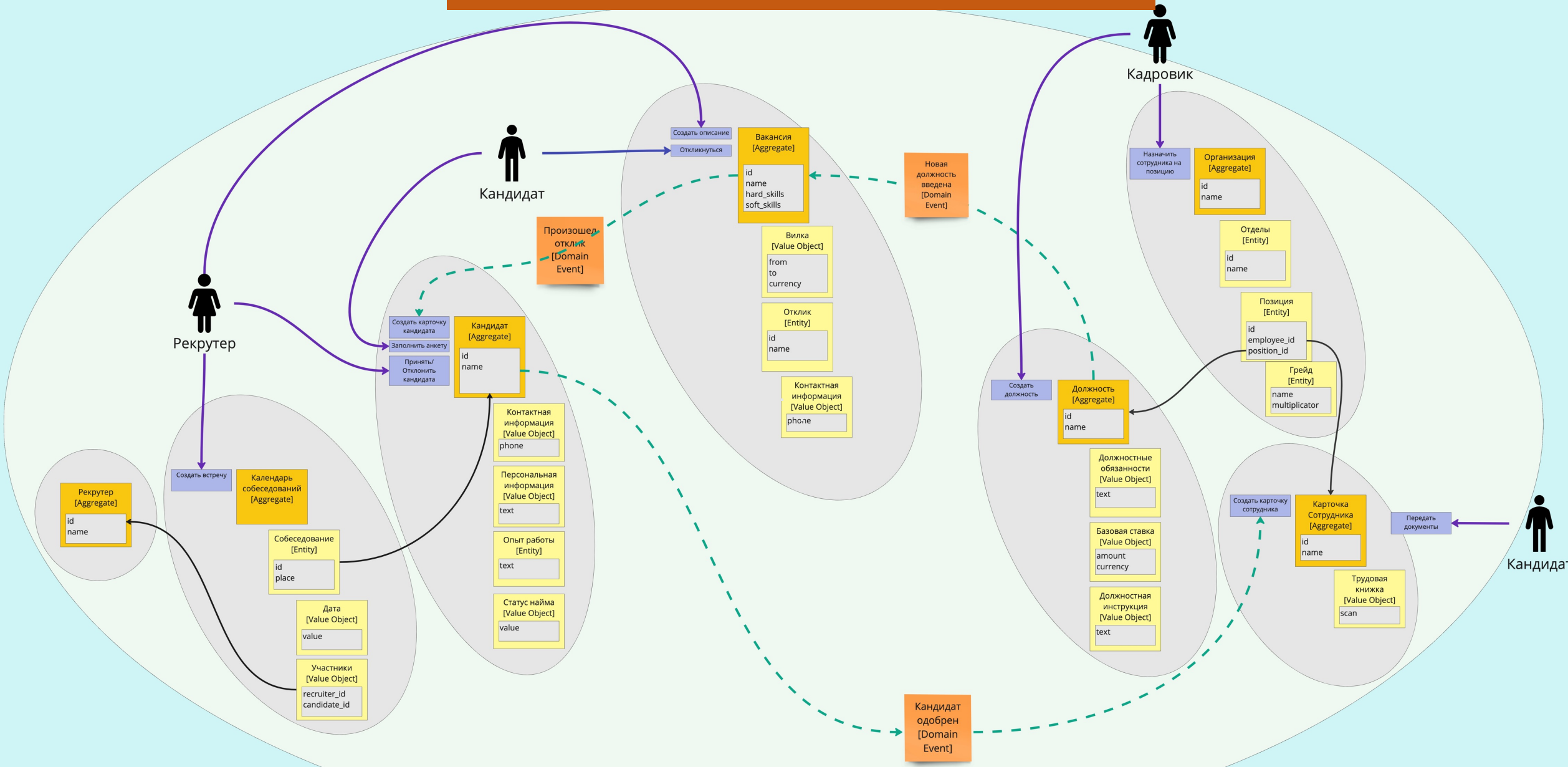
кросс-функциональная команда (~9-10 человек).

# Вывод

Слишком мелкое разбиение повышает риск создать «распределенный монолит».

# Учтем команды

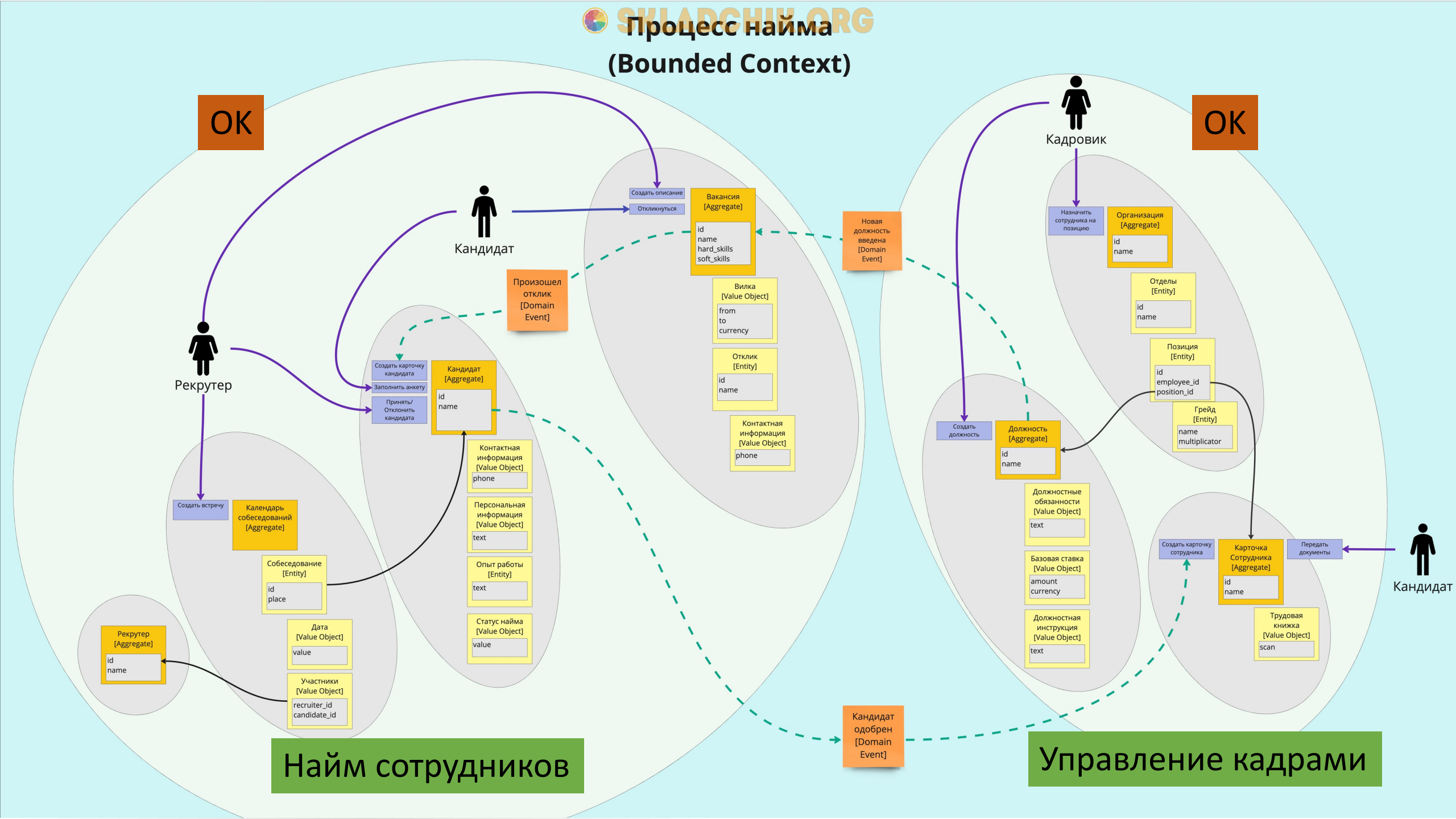
# Слишком большой для одной команды



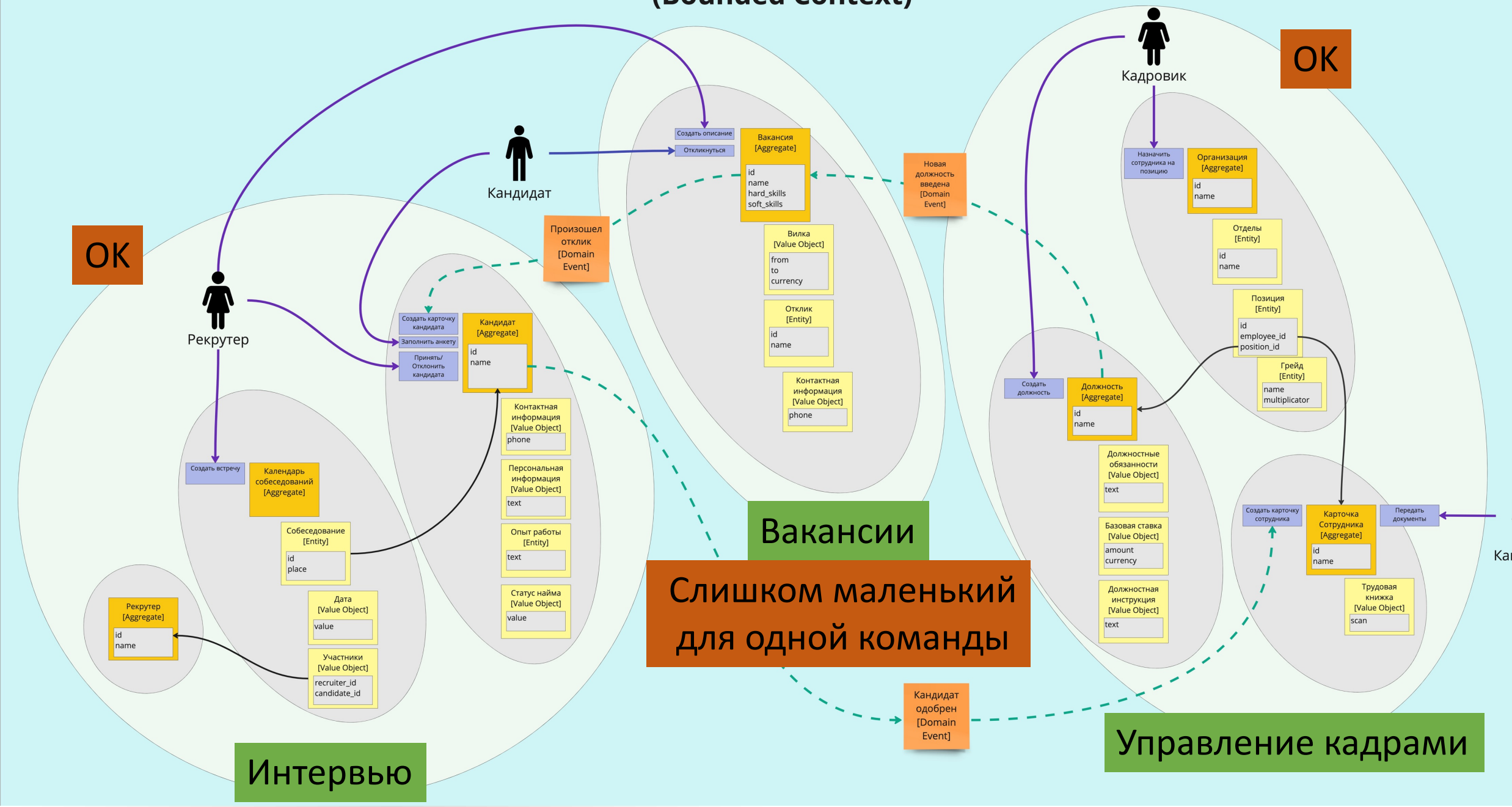

# Процесс найма

SKLADCHIK.ORG

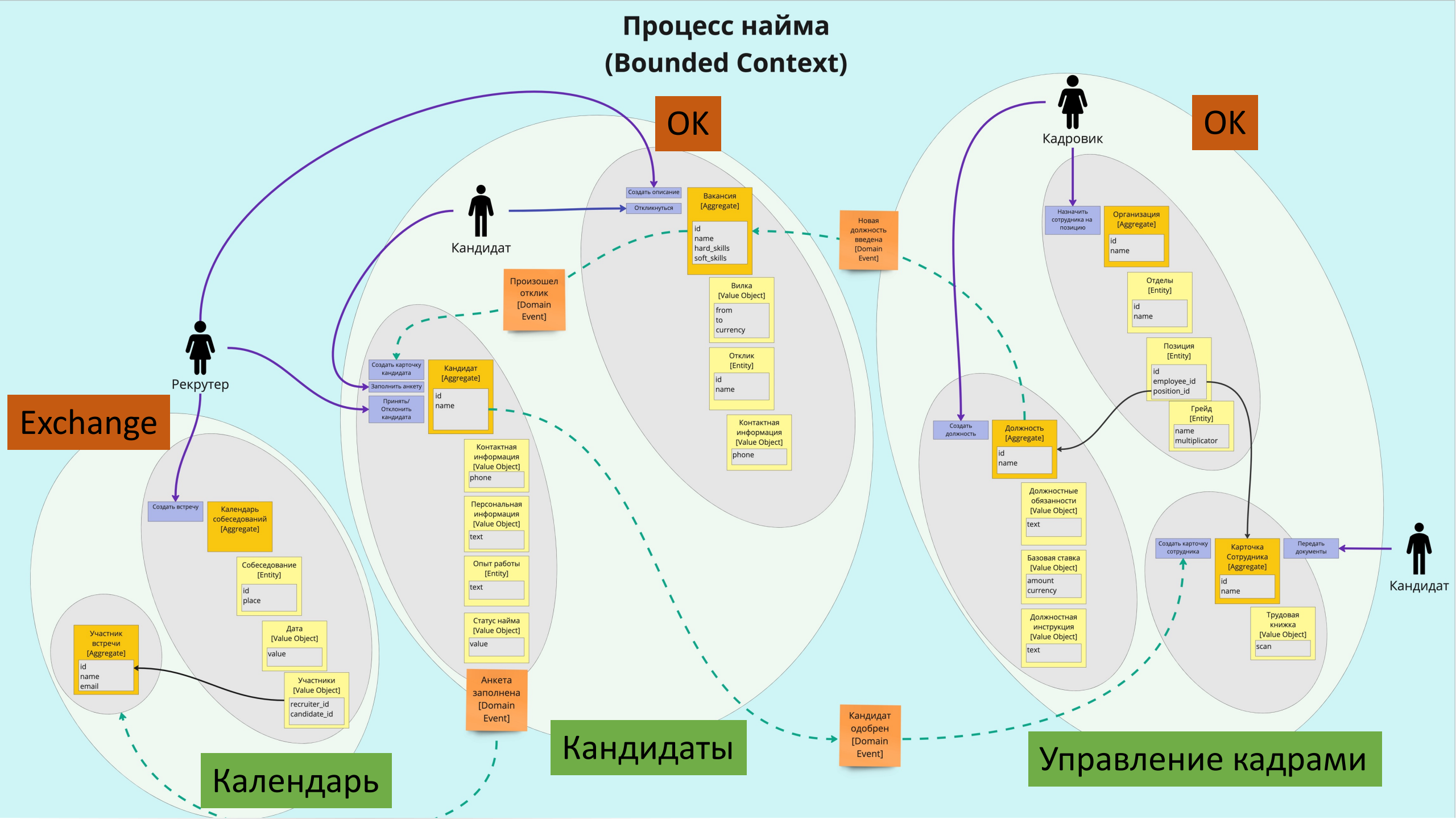
# Процесс найма (Bounded Context)



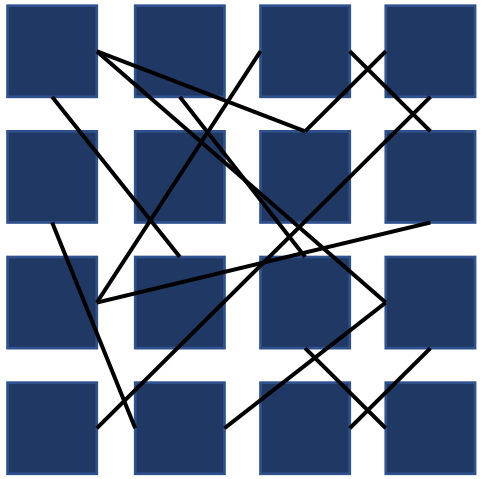
# Процесс найма
# (Bounded Context)



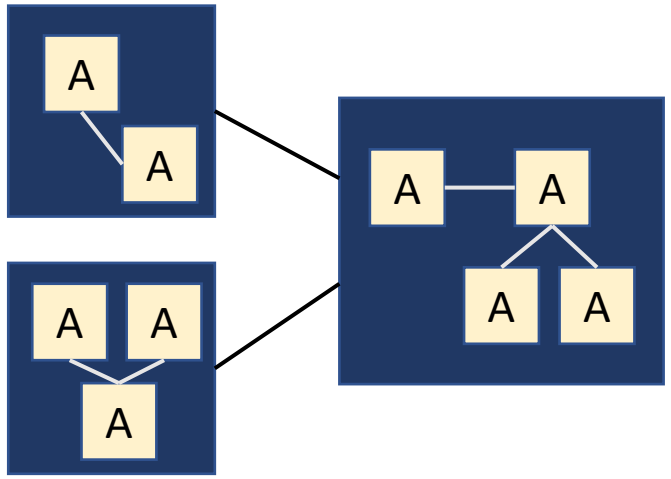
# Процесс найма
# (Bounded Context)



# Итого



👎 Anti-pattern "Распределенный монолит"



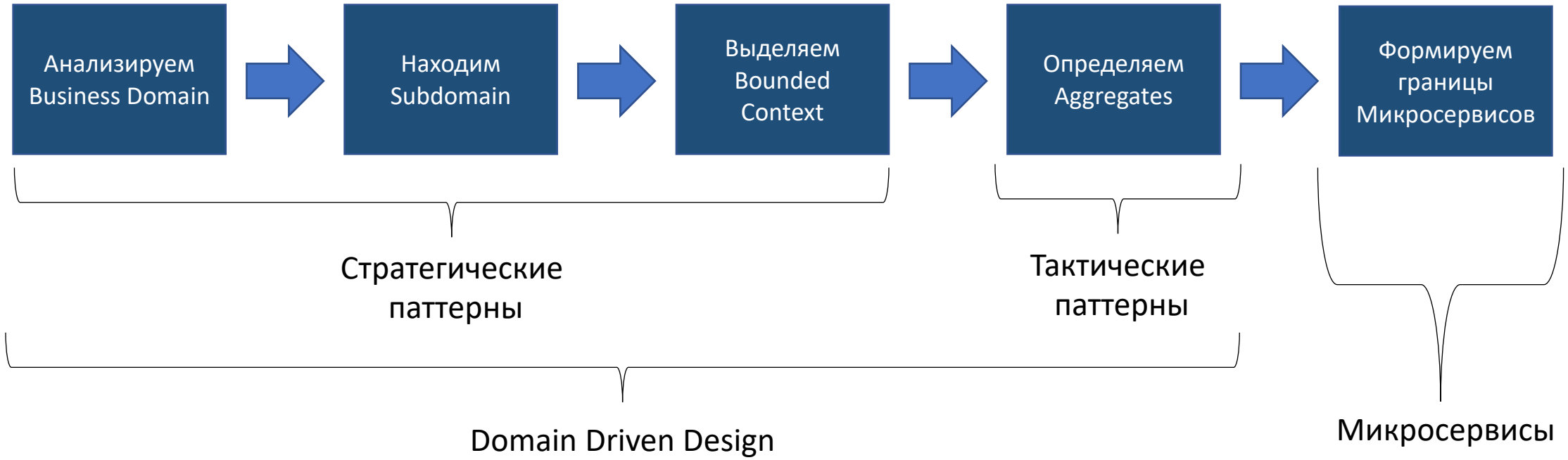

Микросервисы 👍

# Event Storming

# Цели занятия

- Познакомиться с Event Storming и научиться его проводить

# Тактика декомпозиции



# Проблема

# Проблема ТЗ


https://youtube.com/shorts/23oXoBXxbyA?feature=share

# Проблема непонимания

МНОЖЕСТВО ЛЮДЕЙ РАБОТАЮТ НАД ПРОЕКТОМ, КОТОРЫЙ В ДЕЙСТВИТЕЛЬНОСТИ НЕ ПОНИМАЮТ

SKLADCHIK.ORG

Как обнаружить Subdomain?

Как выделить Bounded Context / Aggregate?

# Демонстрация в Miro

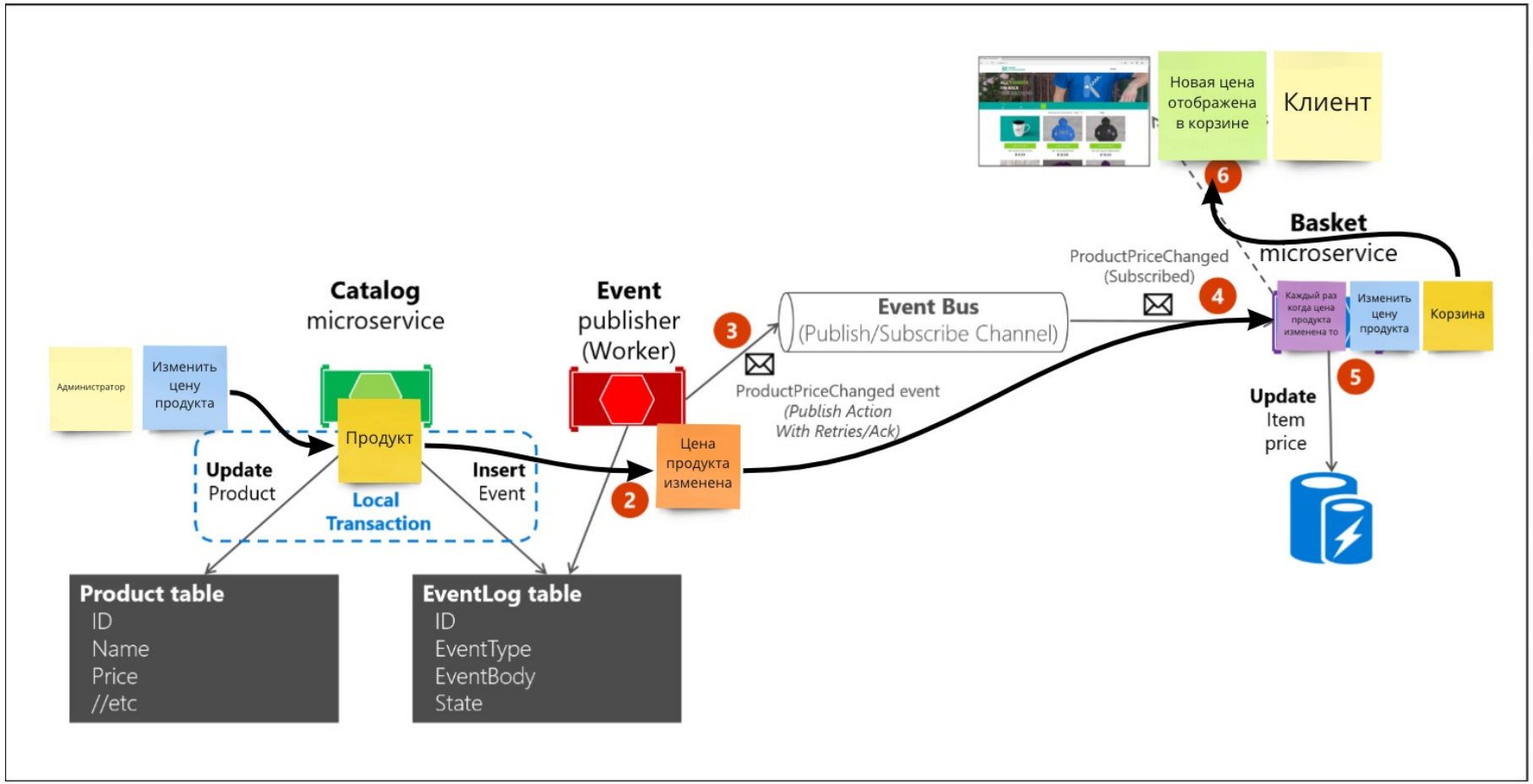
# Когда использовать Event Storming

- Вы моделируете бизнес-процесс
- Вы изучаете новые бизнес-требования
- Вы хотите восстановить знание домена
- Вы хотите улучшить существующий бизнес-процесс
- Вы хотите поделиться знаниями с новыми членами команды

# Когда НЕ использовать Event Storming

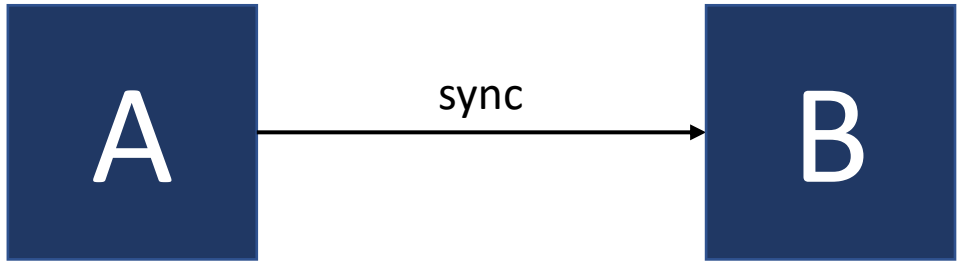
- Если бизнес-процесс, который вы изучаете, прост или очевиден

# Связь с ИТ

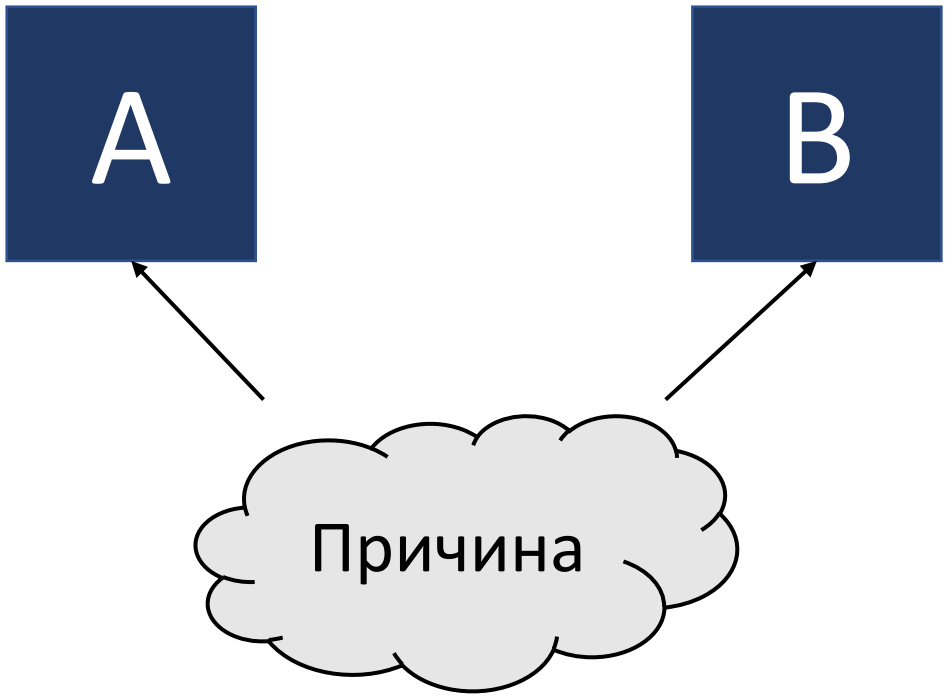

Новая цена отображена в корзине
Клиент
6
Basket microservice
ProductPriceChanged (Subscribed)
4
Каждый раз когда цена продукта изменена то
Изменить цену продукта
Корзина
5
Update Item price
Catalog microservice
Event publisher (Worker)
3
Event Bus (Publish/Subscribe Channel)
ProductPriceChanged event (Publish Action With Retries/Ack)
Администратор
Изменить цену продукта
Продукт
Update Product
Local Transaction
Insert Event
Цена продукта изменена
2
Product table
ID
Name
Price
//etc
EventLog table
ID
EventType
EventBody
State

# Сигналы неправильного разбиения

# Сервис A не может работать без сервиса B

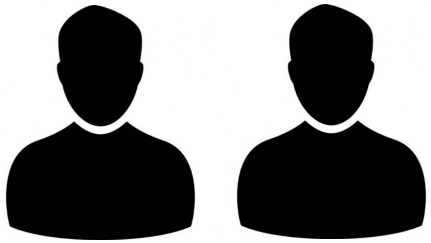


# Сервис A и сервис B меняются по одной причине в большинстве случаев



# Вам нужны ACID транзакции

# Слишком большая команда на один сервис



Команда > 9 человек

Сервис

SKLADCHIK.ORG

# Чек-лист

- У каждого сервиса есть одна обязанность.
- Между сервисами нет излишней коммуникации. Если разделение функций на два сервиса приводит к чрезмерной «болтливости», это может быть признаком того, что эти функции принадлежат одному и тому же сервису.
- Каждый сервис достаточно мал, чтобы его могла создать небольшая команда, работающая независимо. Но не настолько мал, чтобы в команде было много сервисов.
- Нет взаимозависимостей, которые потребуют синхронного развертывания двух или более сервисов. Всегда должна быть возможность развернуть сервис без повторного развертывания каких-либо других сервисов.
- Сервисы не являются тесно связанными и могут развиваться независимо друг от друга.
- Границы ваших сервисы не создают проблем с согласованностью или целостностью данных. Иногда важно поддерживать согласованность данных, помещая функциональные возможности в один микросервис. Тем не менее, подумайте, действительно ли вам нужна строгая согласованность. Существуют стратегии обеспечения отложенной согласованности в распределенной системе, и преимущества декомпозиции сервисов часто перевешивают проблемы связанные с отложенной согласованностью (отложенная согласованность будет обсуждаться чуть позже).

# О чем узнали

- Микросервис не обязательно должен быть очень маленьким, достаточно чтобы он был автономным (слабосвязанным) по отношению к другим микросервисам
- 1-2 микросервиса на команду, это оптимальное количество
- Слабая связанность и автономность сервисов – признак хорошей декомпозиции
- Если ИТ-специалисты, участвующие в декомпозиции, не понимают как устроен бизнес-домен, то декомпозиция на сервисы будет некорректной с большой долей вероятности
- Автономность важнее переиспользования

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.


# Вебинар

# Цели занятия

- Понять, как формируются команды при применении микросервисной архитектуры
- Разобраться в связи Scrum и микросервисной архитектуры
- Понять роль архитектора. За что он отвечает, а за что нет
- Разобраться, какие ещё виды команд бывают, кроме продуктовых

# Команды и организационная трансформация

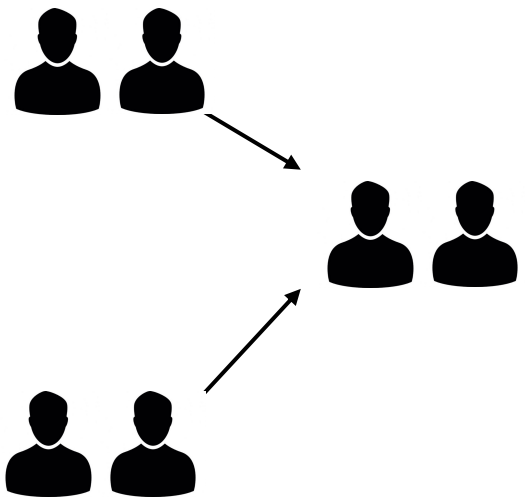
# Организация команд

# Закон Конвея

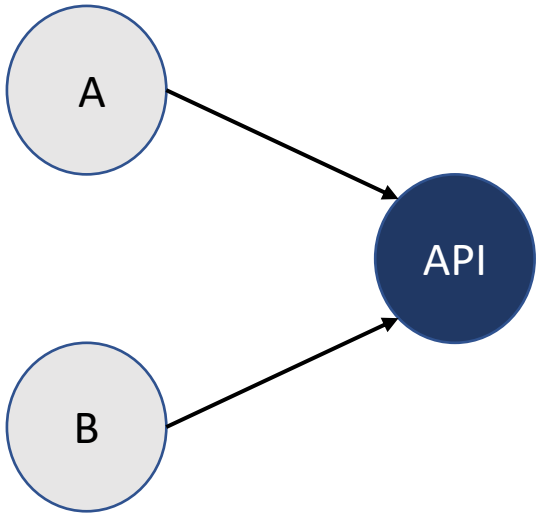
Любая организация, проектирующая систему, неизбежно создаст такую модель, которая будет повторять коммуникационную структуру самой организации.

# Закон Конвея

**Структура организации**

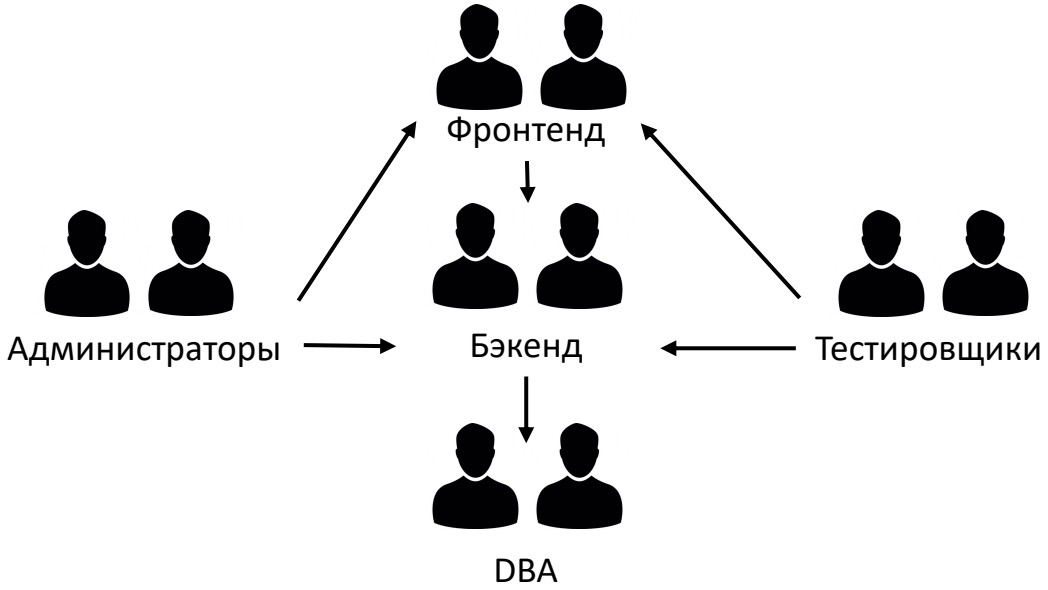


**Система**

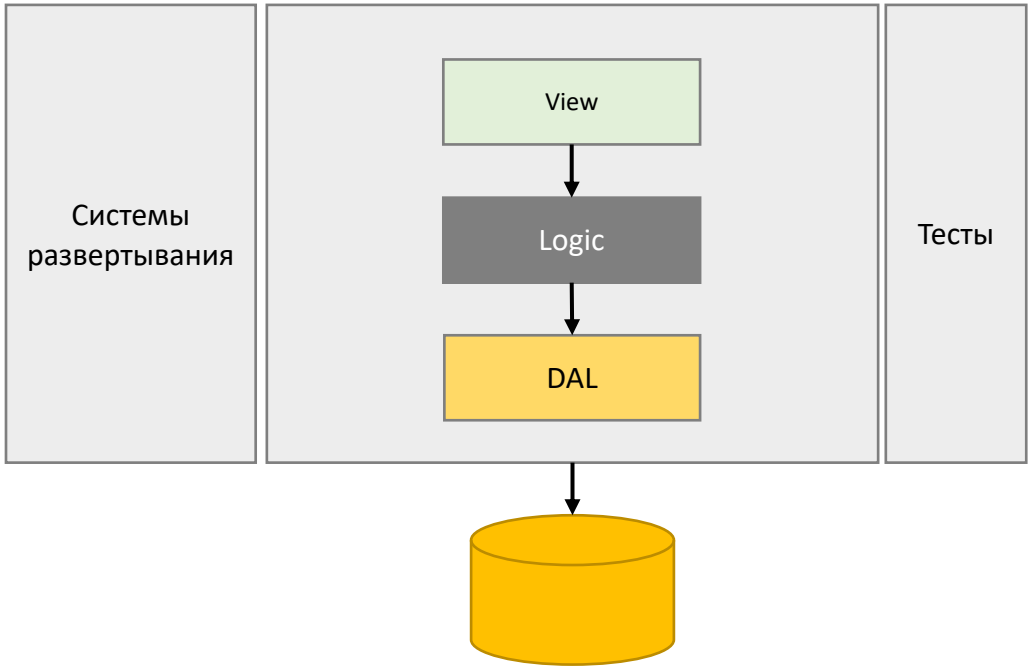


# Пример

**Структура организации**




**Монолит**



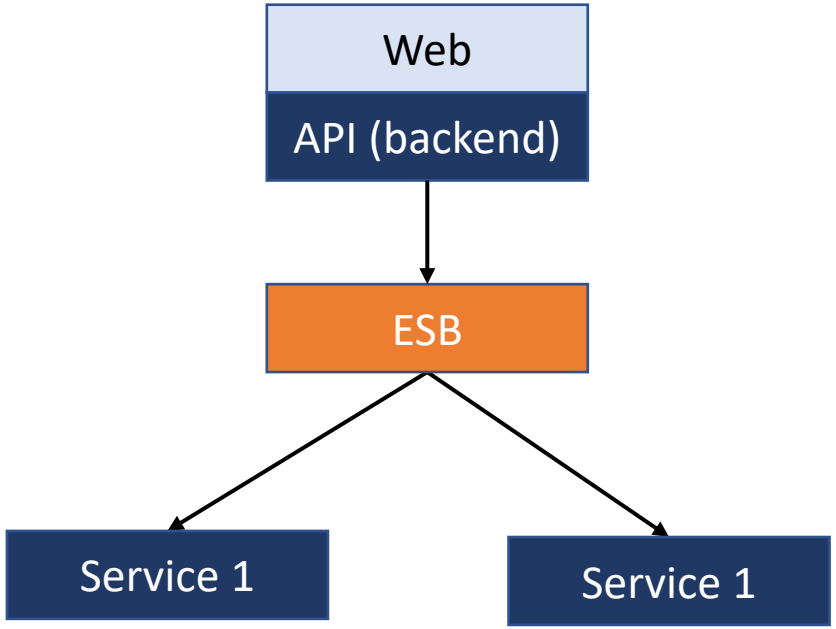
SKLADCHIK.ORG

# Обратный закон Конвея

Организации, использующие программные системы ограничены структурами коммуникации, которые копируют эту систему.

# Пример

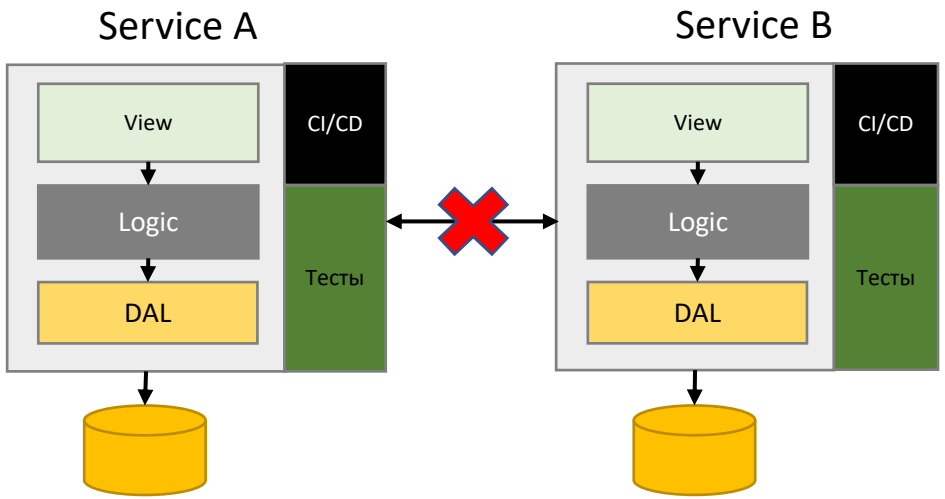
**SOA**




**Структура организации**

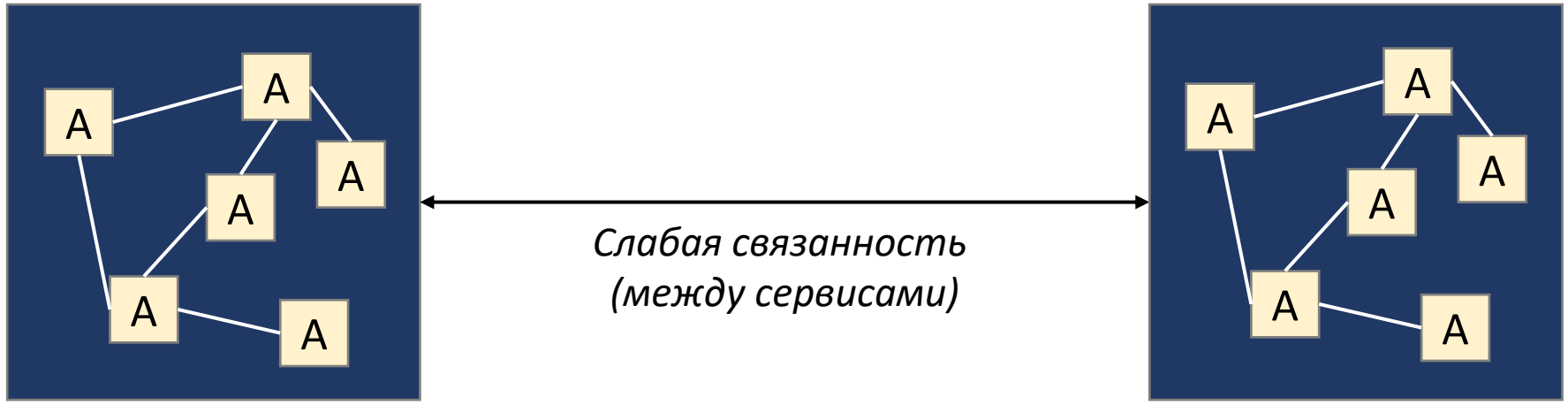
# Структура команд в MSA

**MSA**




**Структура команд**

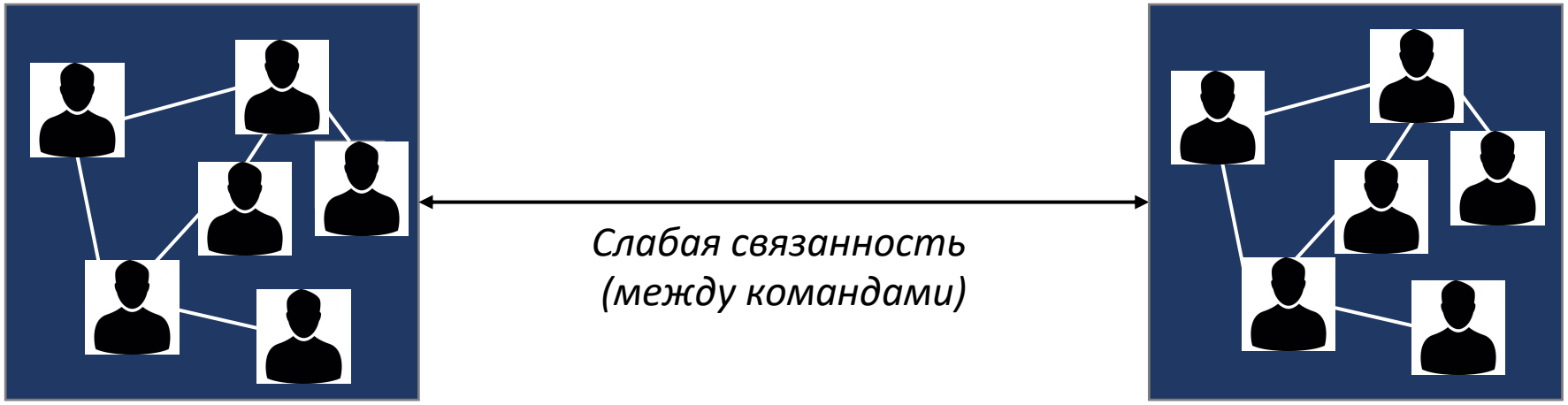
# Тактика декомпозиции на сервисы




Сильное
зацепление
(внутри сервиса)

Сильное
зацепление
(внутри сервиса)

# Тактика декомпозиции на команды

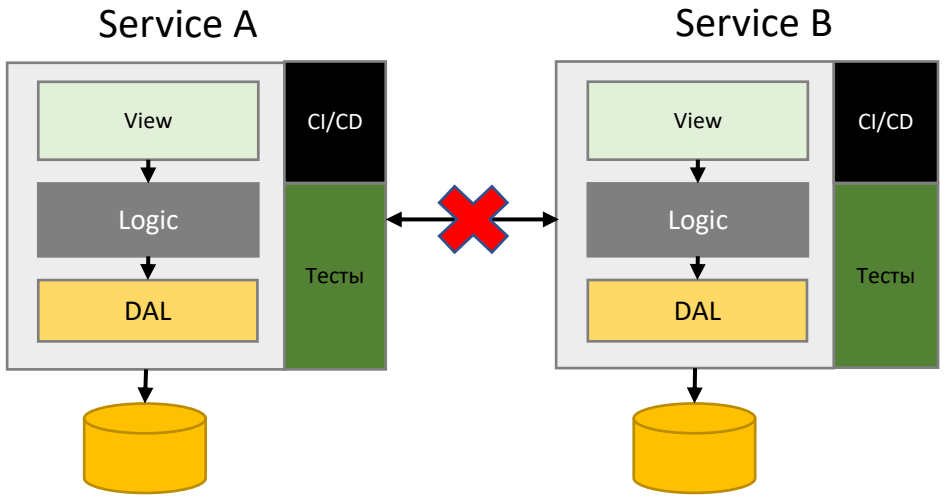



Сильное
зацепление
(внутри команды)

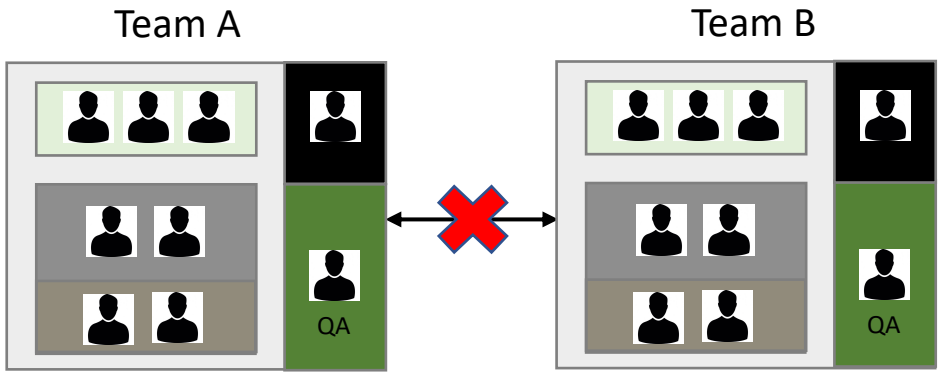
Сильное
зацепление
(внутри команды)

# Структура команд в MSA

**MSA**




**Структура команд**

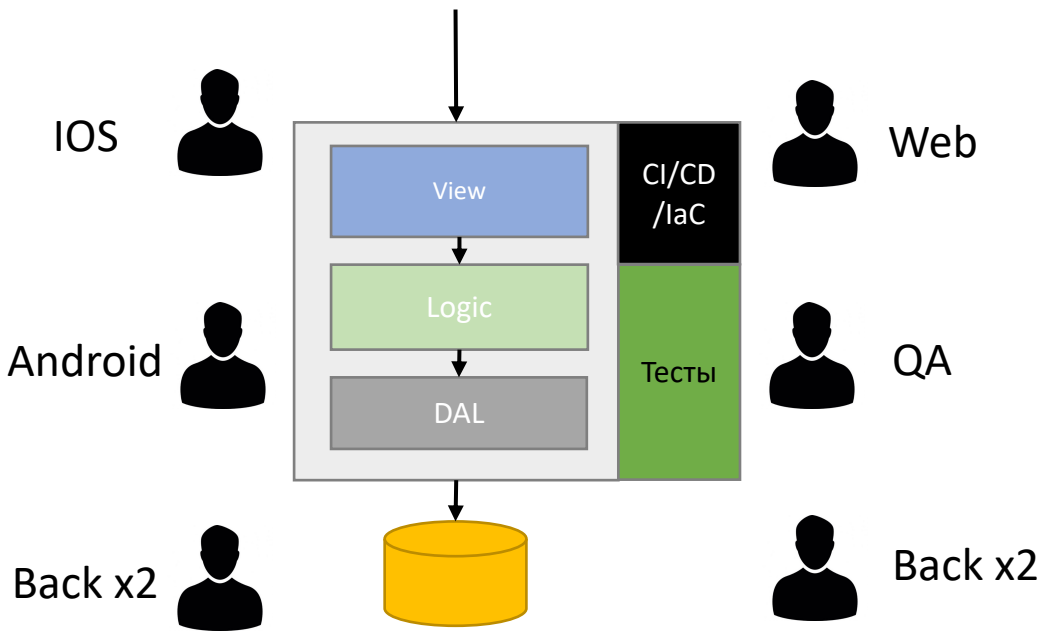


# Владение сервисом

# Service per team pattern

# Service per team pattern

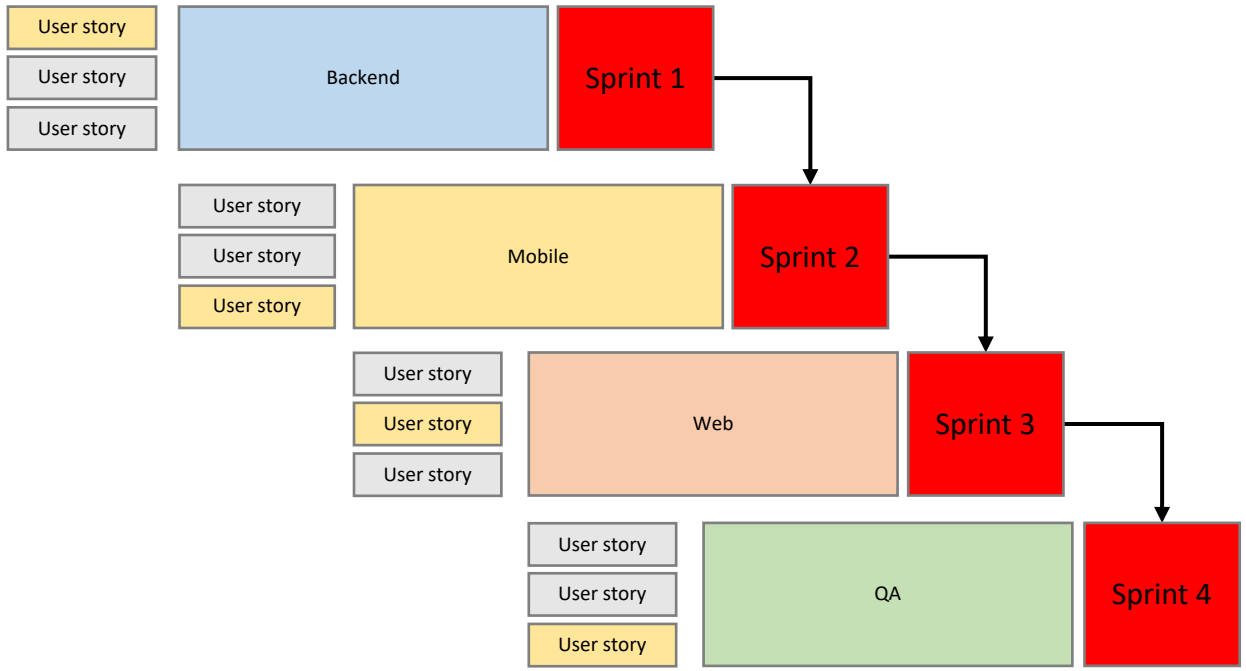
**MSA**




**Структура**

- Организована вокруг предметной области
- Небольшая (9+- человек)
- Кросс-функциональная
- Автономная
- Долгоживущая
- Владеет 1-2 сервисами
- Несет полную ответственность за развитие и поддержку сервиса

# Последствия неправильного разбиения на команды




Lead Time = больше 4 Sprint’ов

SKLADCHIK.ORG

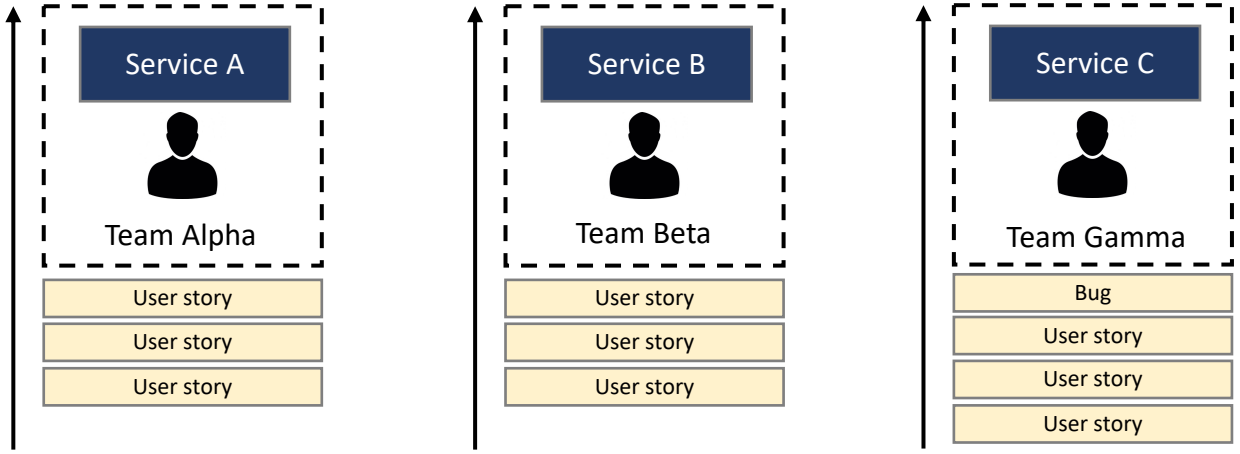
# Вывод

Если вы внедрили микросервисы, автономные и независимые, то орг. структура должна им соответствовать – иметь автономные и независимые команды.

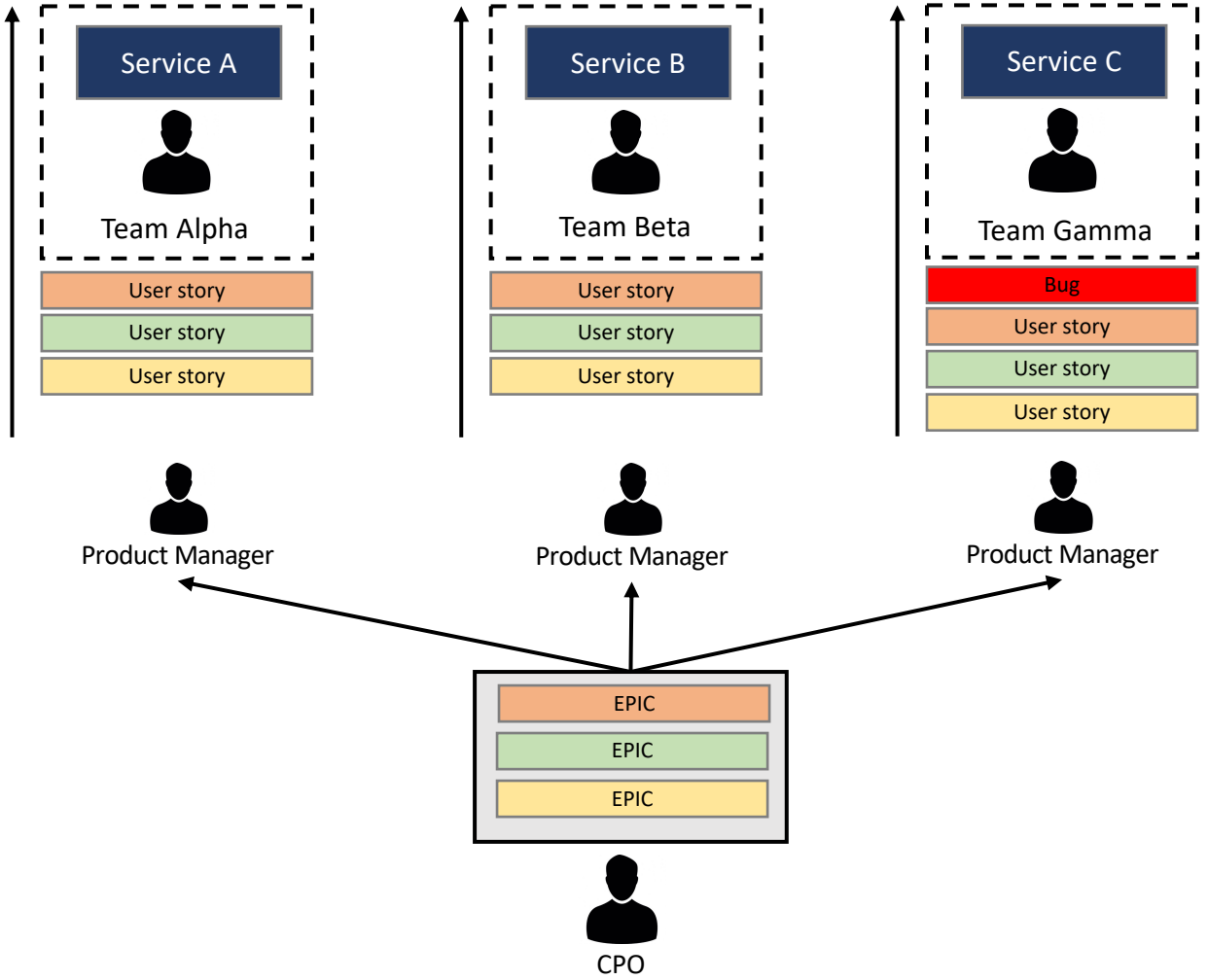
# Управление продуктом в MSA

# Сочетание MSA со Scrum

# Сочетание MSA со Scrum



# Сочетание MSA со Scrum



# Роль архитектора в MSA

# Взгляд изнутри и снаружи

**Команда и лес**

**Архитектор и лес**

# Роль архитектора в MSA

# Если нет архитектора

# Зоны контроля

**Архитектор**

- Определение границ сервисов
- Определение оптимальных способов межсервисного взаимодействия
- Построение микросервисной платформы (Auth server, Api Gateway  и т.п.)
- Формирование стандартов

**Команда**

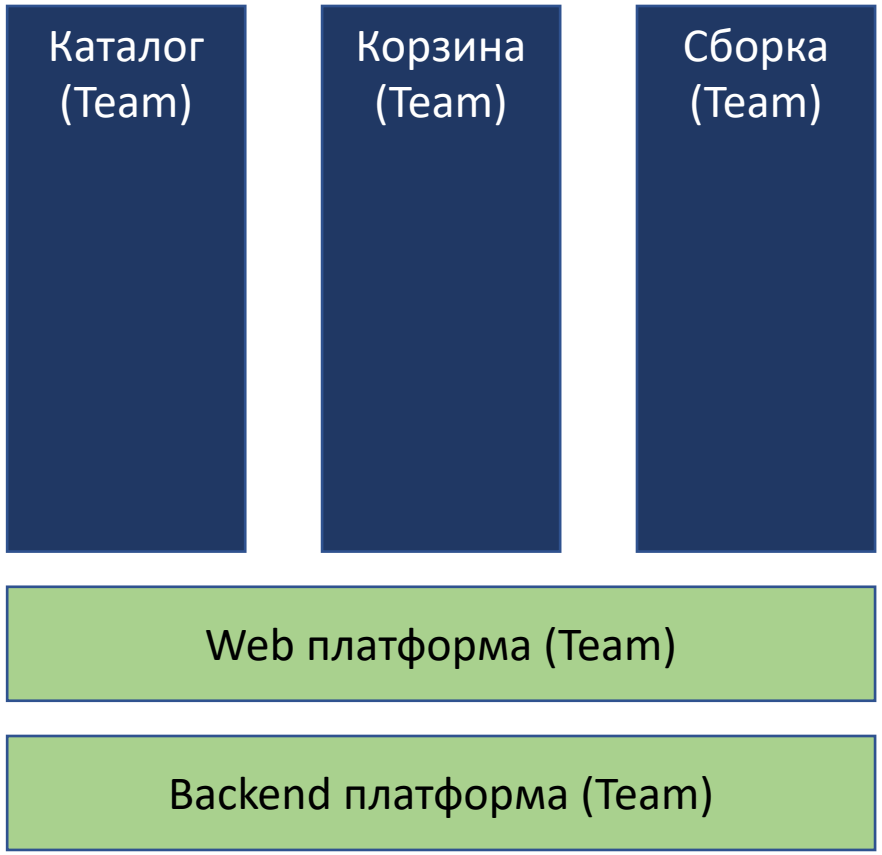
- Глубокое понимание бизнес области сервиса
- Архитектура внутри сервиса
- Технологии внутри сервиса (в рамках TechRadar компании)
- Реализация и развитие сервиса

SKLADCHIK.ORG

# Платформенные команды

# Платформенные команды

- Самые опытные разработчики
- Контролируют стек в рамках своей платформы
- Делают функции, системы и библиотеки, которые помогают продуктовым командам работать быстрее



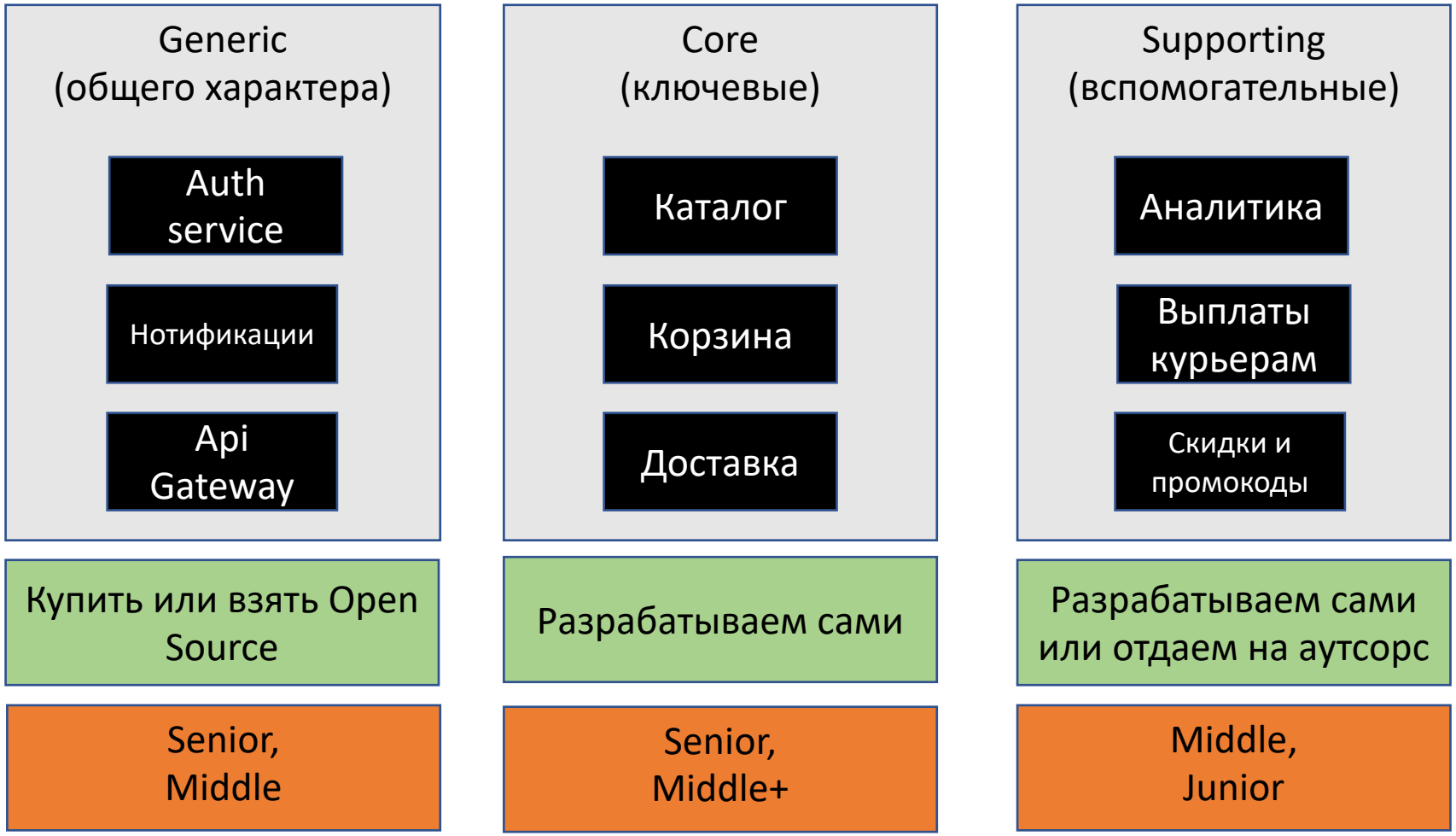
# Команды разработки общих сервисов

# Команды разработки общих сервисов

- Опытные разработчики
- Делают/поддерживают инфраструктурные сервисы, которые не имеют продуктовой составляющей, но нужны для работы системы

# Как распределять разработчиков на сервисы

# Как распределять разработчиков



# О чем узнали

- Команды организованные вокруг технологии будут негативно влиять на Lead Time
- Не все команды работают на продукт, некоторые решают платформенные задачи
- Архитектор не отвечает за внутреннюю архитектуру сервиса, это ответственность команды

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

# Раскрытие API

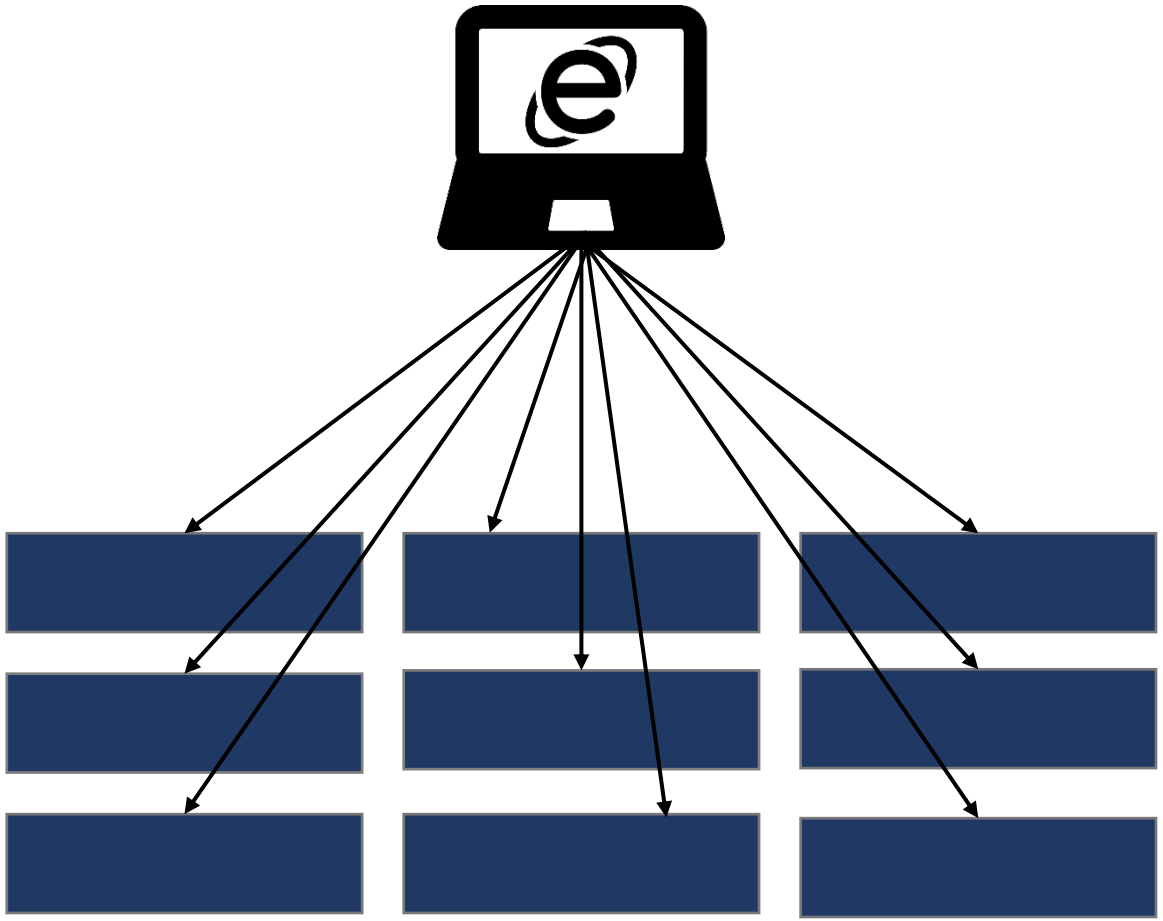
# Цели занятия

- Понять, как правильно раскрывать доступ до API сервисов для фронтенд приложений
- Узнать, что делать, если данные находятся в разных сервисах, но нужны на одном экране
- Разобраться, как настроить аутентификацию в микросервисной архитектуре

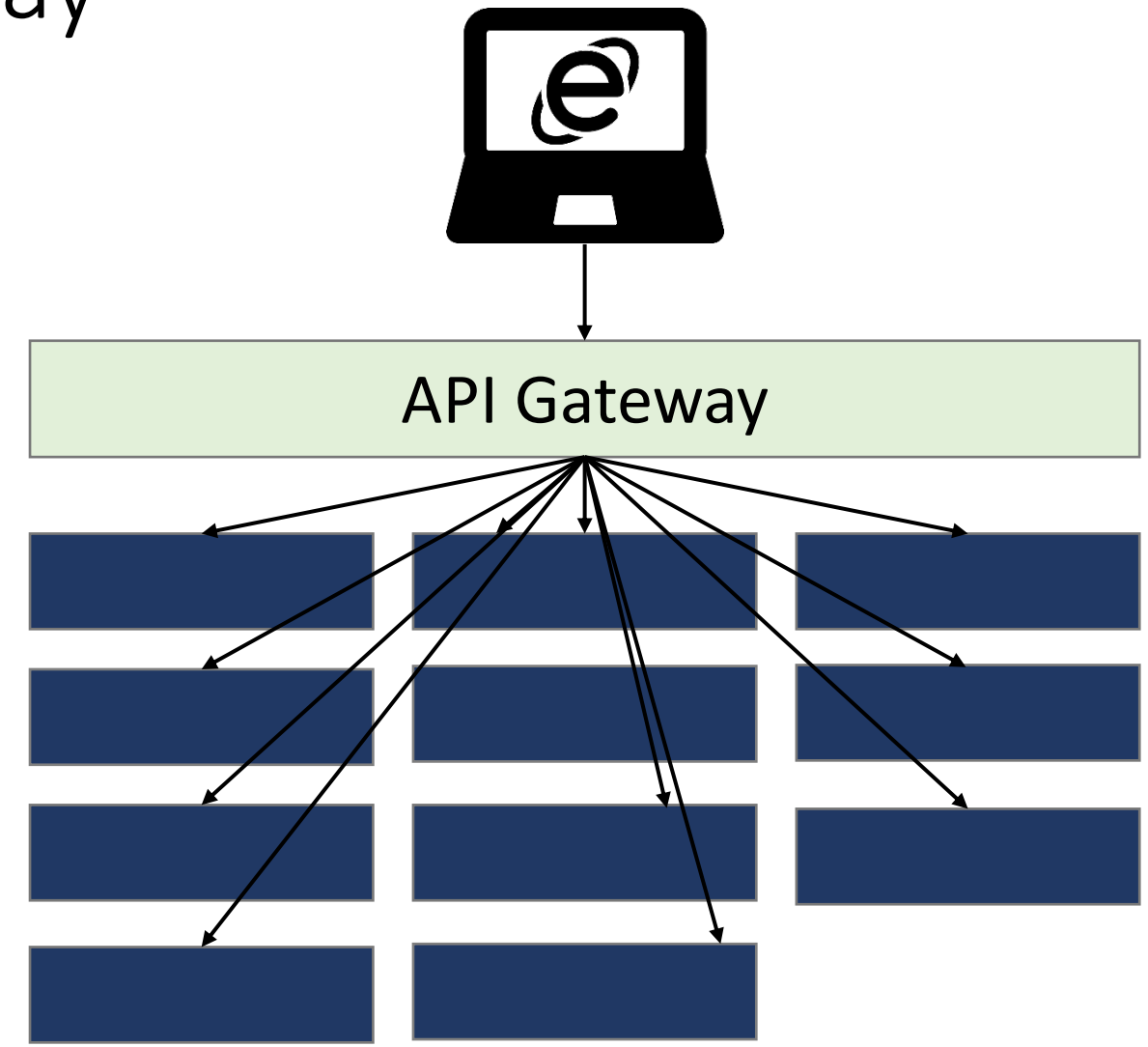
SKLADCHIK.ORG

# API Gateway pattern

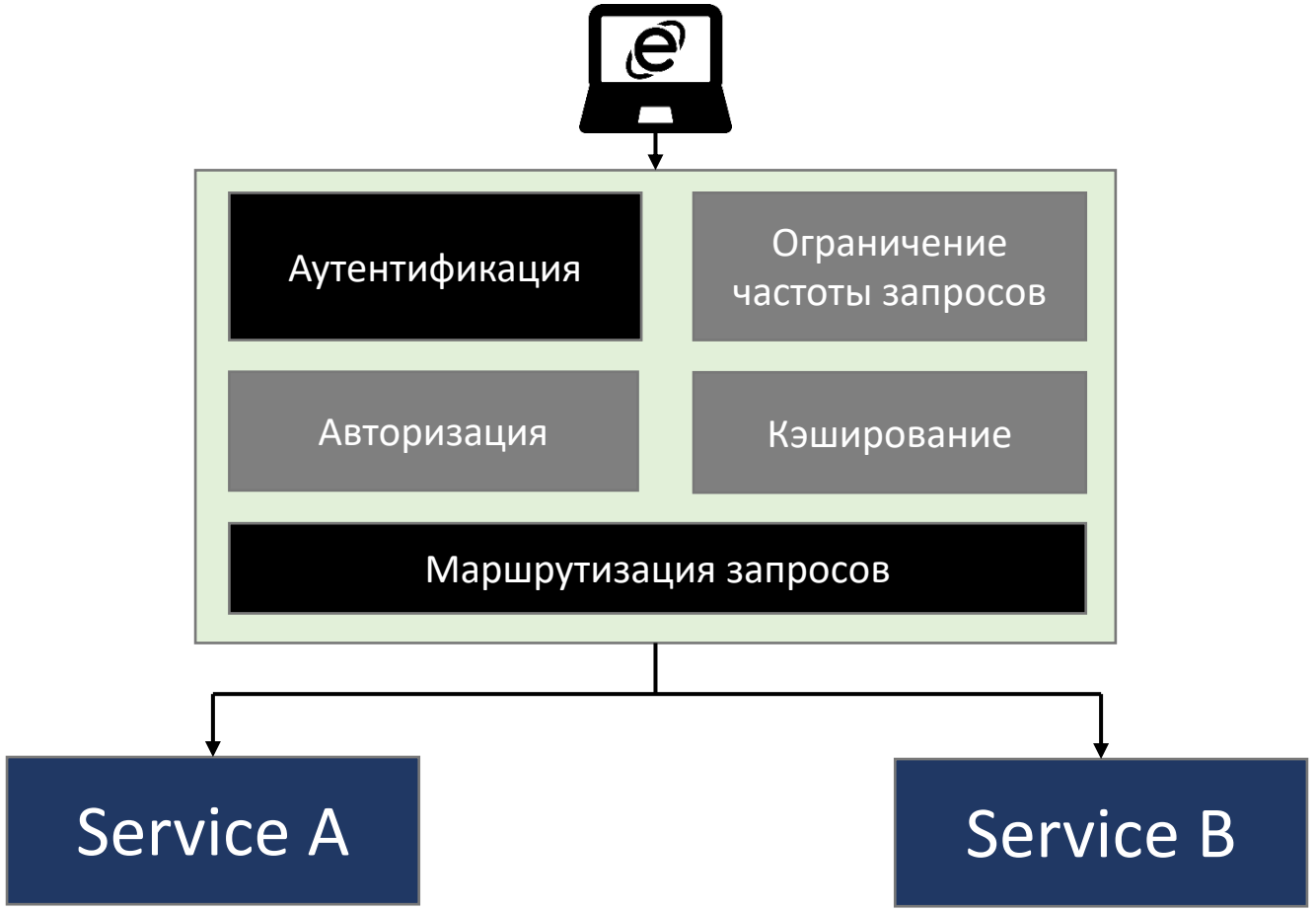
# Использование API сервисов напрямую



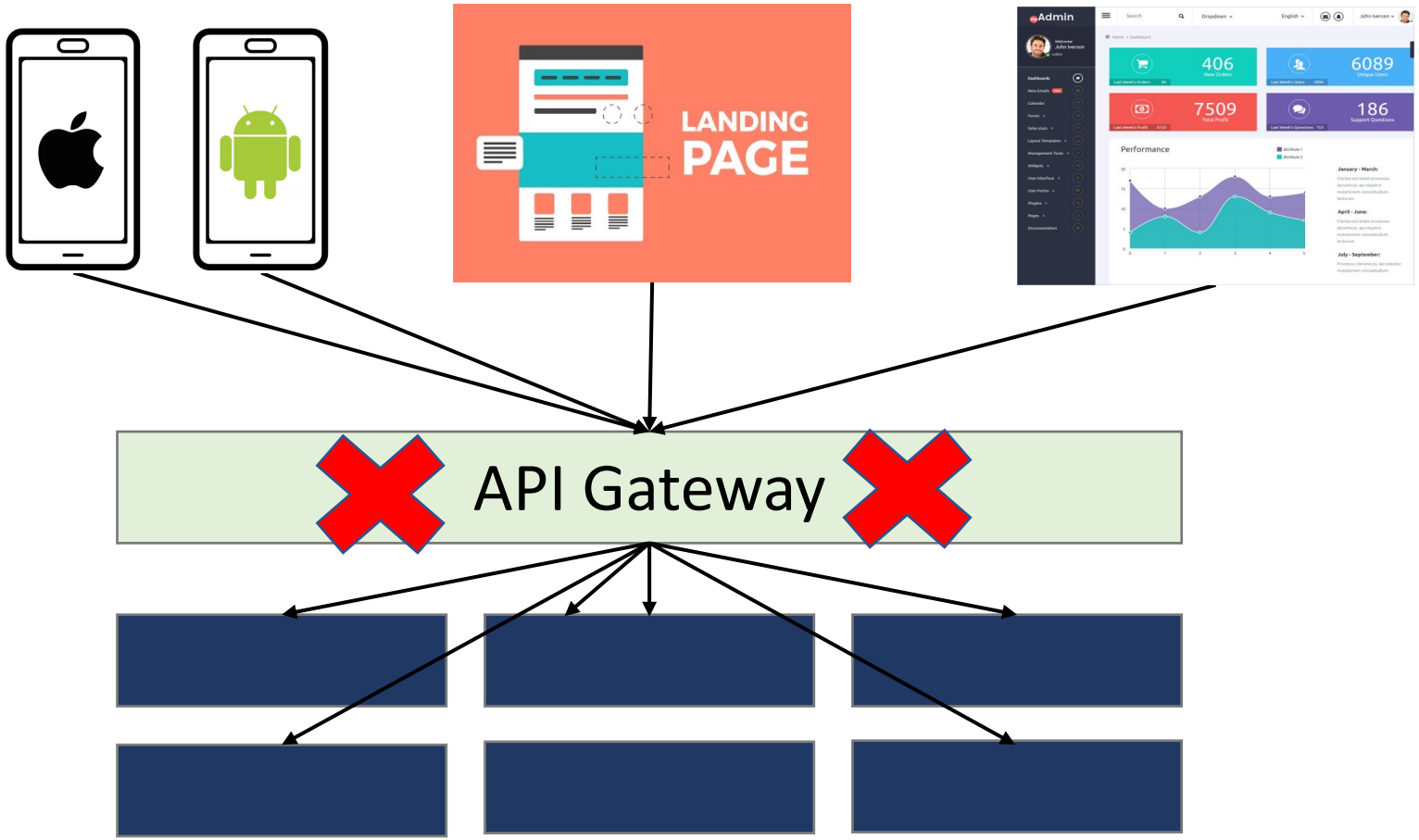
# API Gateway



# Задачи API Gateway

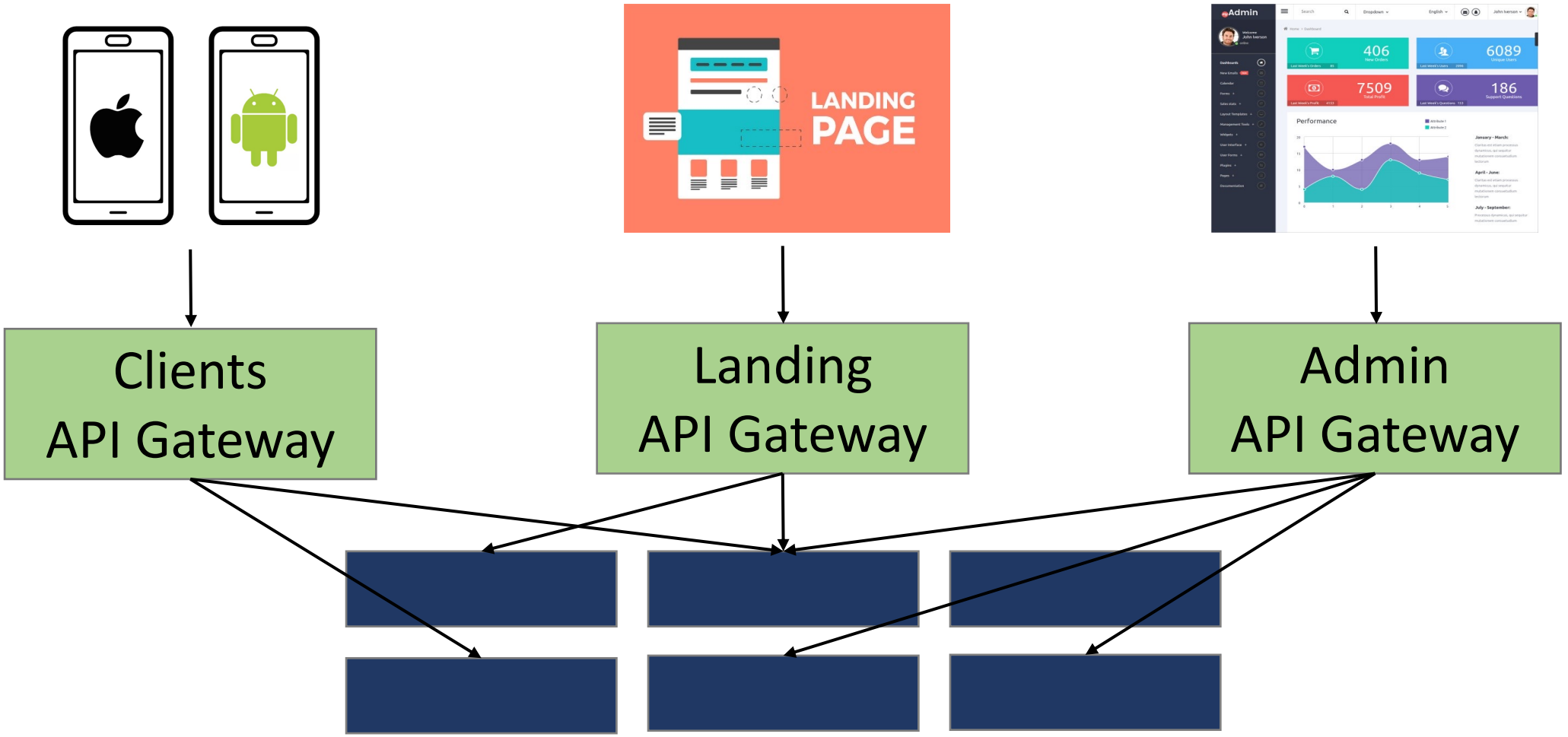


# Проблема единого API Gateway



# BFF (Backend for front-end) pattern

# BFF


LANDING
PAGE
Admin
406
6089
7509
186
Performance
Clients
API Gateway
Landing
API Gateway
Admin
API Gateway

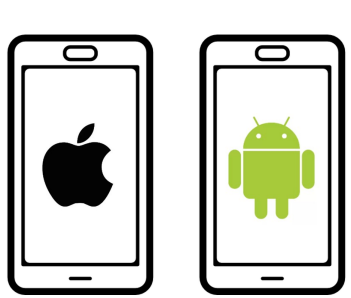
# Существующие решения

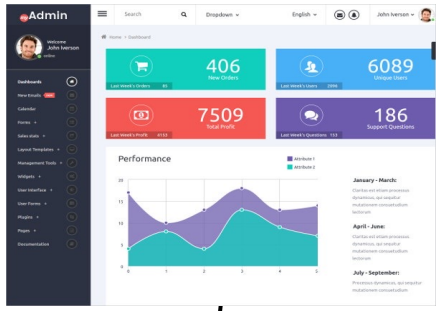




# Пример

# Пример




[POST] api/v1/pay
[GET] api/v1/cards/{id}

Clients
API Gateway

[POST] api/v1/cards

Landing
API Gateway

[POST] api/v1/cards/{id}/block

Admin
API Gateway

[POST] api/v1/pay

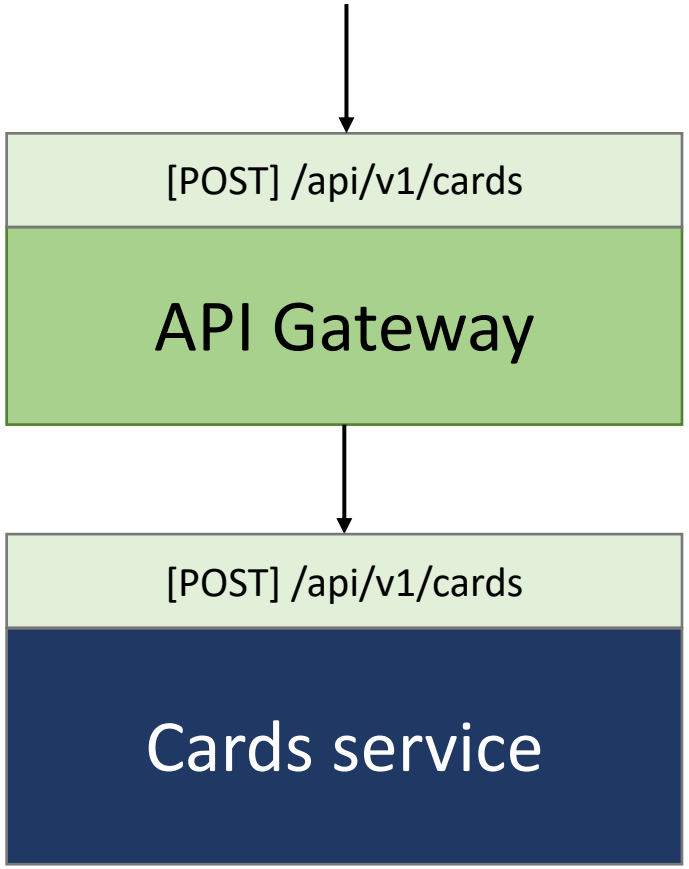
Payment service

[POST] api/v1/cards
[GET] api/v1/cards/{id}
[POST] api/v1/cards/{id}/block

Cards service

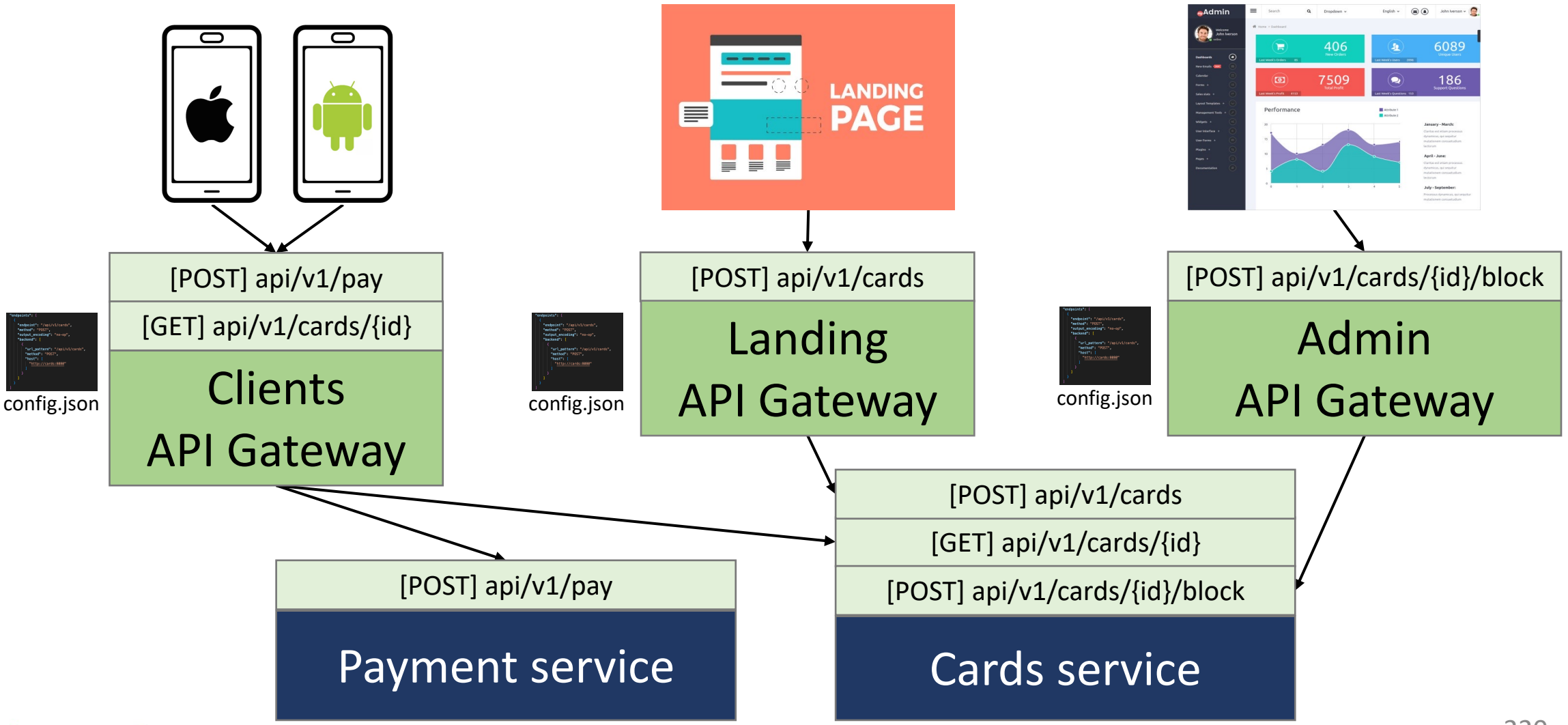
SKLADCHIK.ORG

# Конфигурация



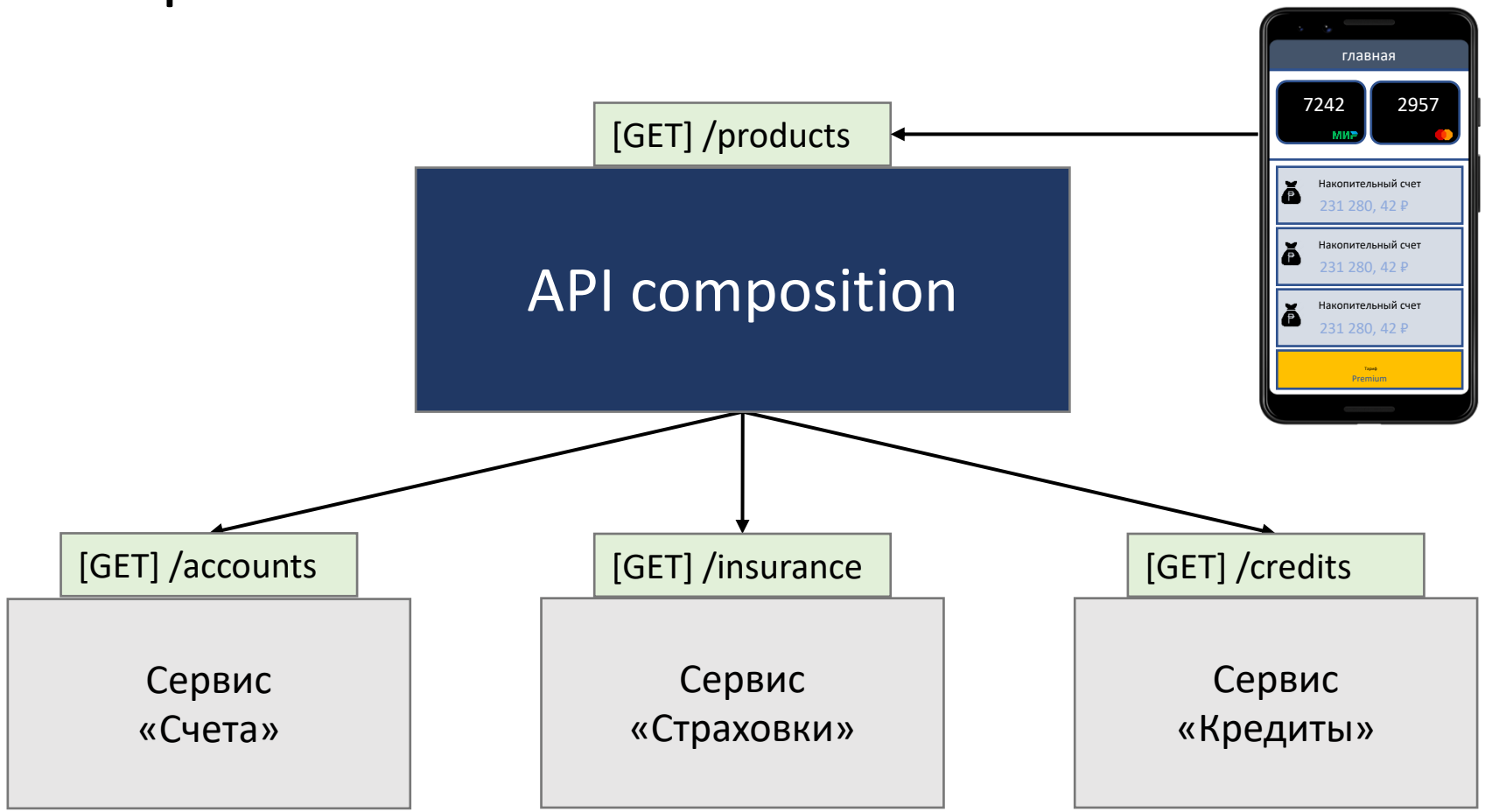

```
"endpoints": [
  {
    "endpoint": "/api/v1/cards",
    "method": "POST",
    "output_encoding": "no-op",
    "backend": [
      {
        "url_pattern": "/api/v1/cards",
        "method": "POST",
        "host": [
          "http://cards:8080"
        ]
      }
    ]
  }
]
```

# Результат

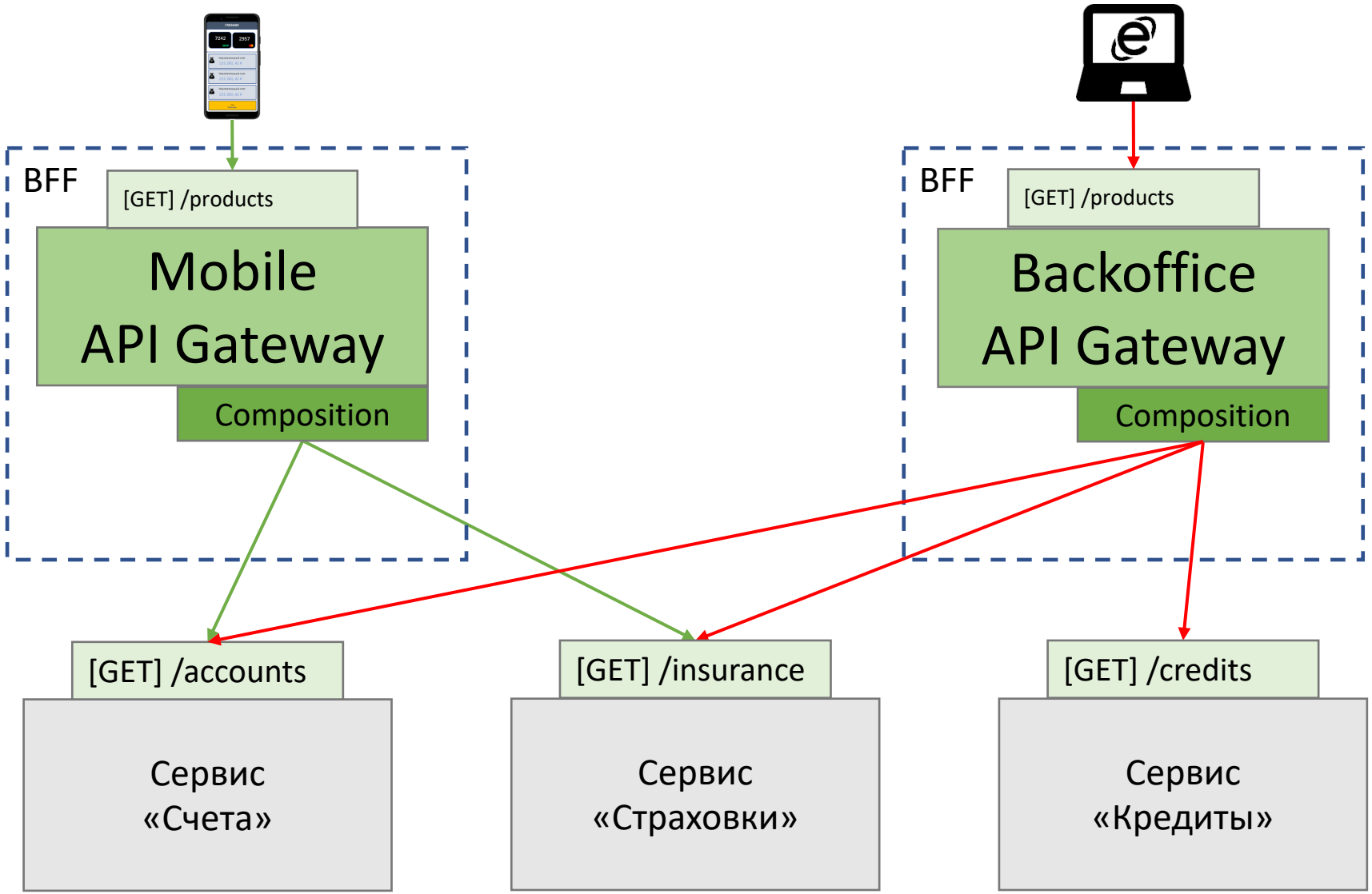

LANDING PAGE
[POST] api/v1/pay
[GET] api/v1/cards/{id}
Clients API Gateway
config.json
[POST] api/v1/cards
Landing API Gateway
config.json
[POST] api/v1/cards/{id}/block
Admin API Gateway
config.json
[POST] api/v1/pay
Payment service
[POST] api/v1/cards
[GET] api/v1/cards/{id}
[POST] api/v1/cards/{id}/block
Cards service

# API Composition pattern

# API Composition



# API Composition

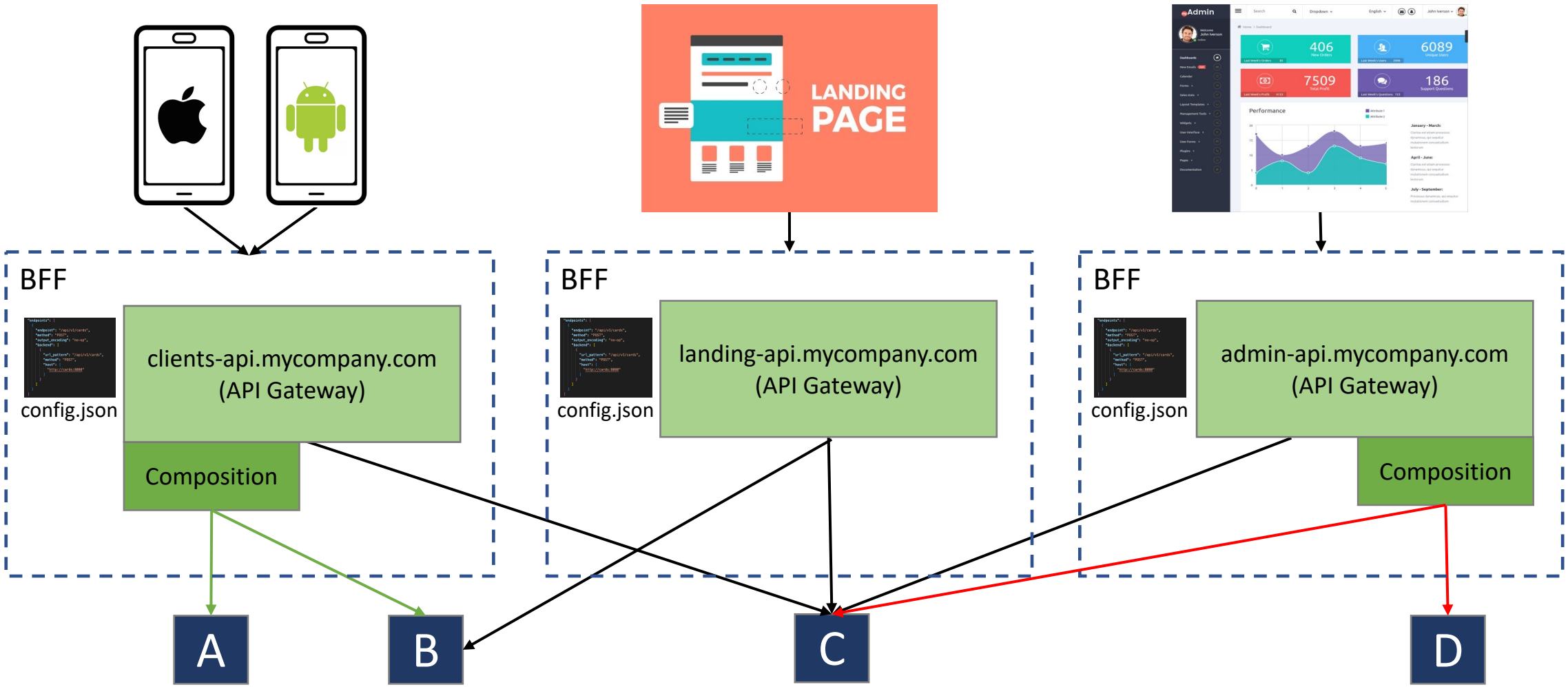


# Конфигурация

```
{
    "endpoints": [
        {
            "endpoint": "/abc",
            "timeout": "800ms",
            "method": "GET",
            "backend": [
                {
                    "url_pattern": "/a",
                    "encoding": "json",
                    "host": [
                        "http://service-a.company.com"
                    ]
                },
                {
                    "url_pattern": "/b",
                    "encoding": "xml",
                    "host": [
                        "http://service-b.company.com"
                    ]
                },
                {
                    "url_pattern": "/c",
                    "encoding": "json",
                    "host": [
                        "http://service-c.company.com"
                    ]
                }
            ]
        }
    ]
}
```

https://www.krakend.io/docs/endpoints/response-manipulation/

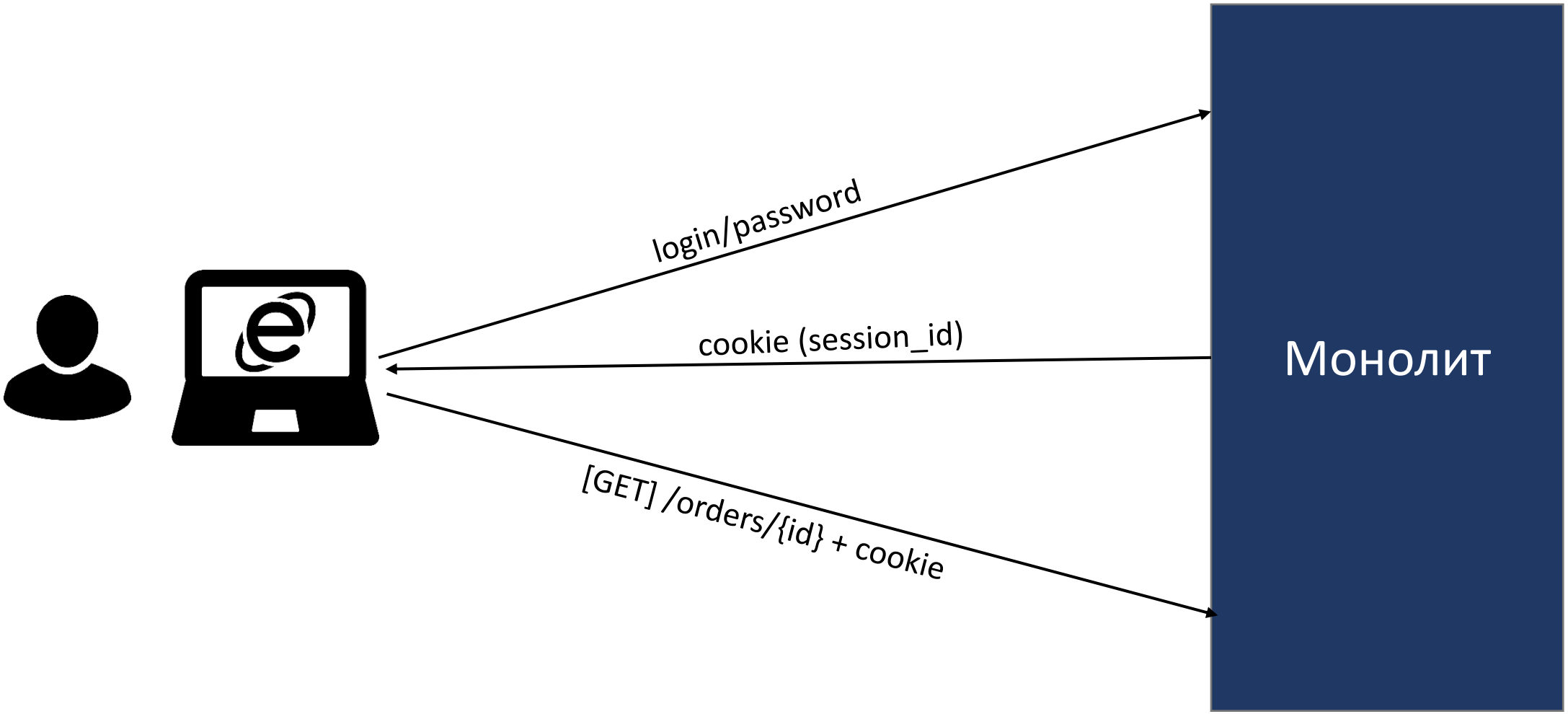
# Итоговая схема



# Аутентификация, авторизация

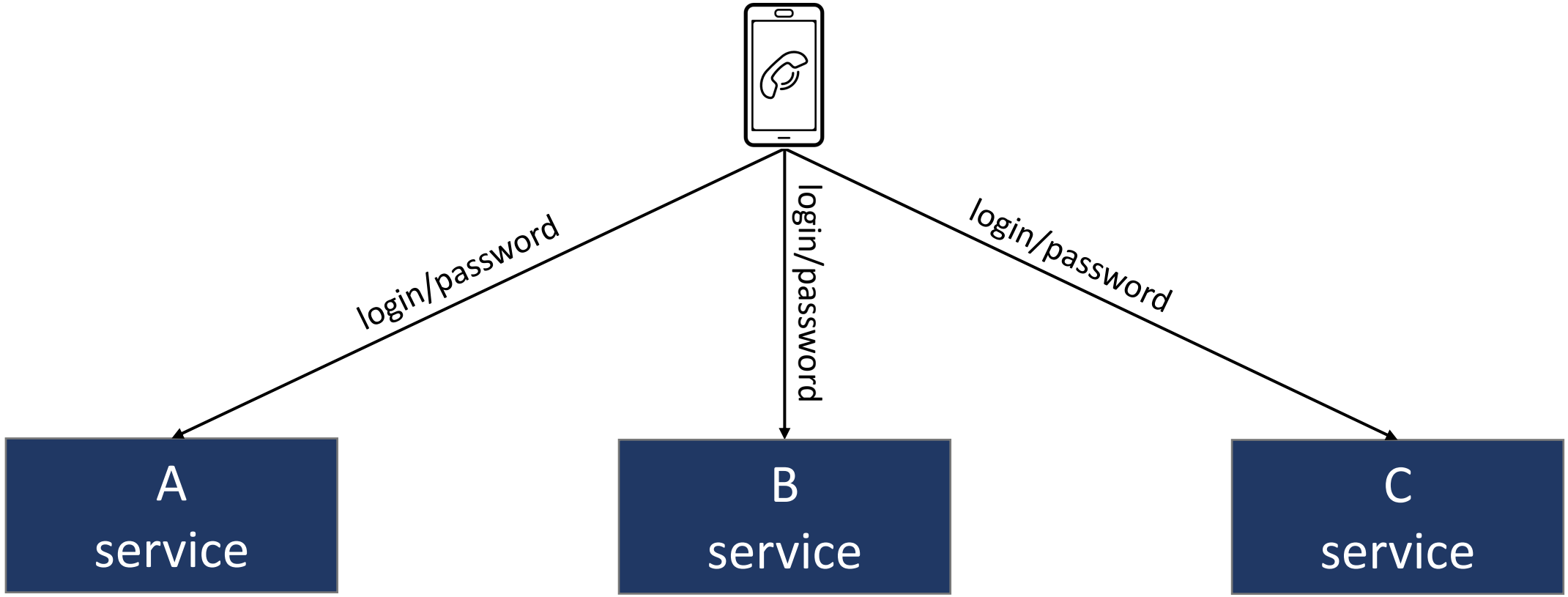
# Access token pattern

# Аутентификация в монолите

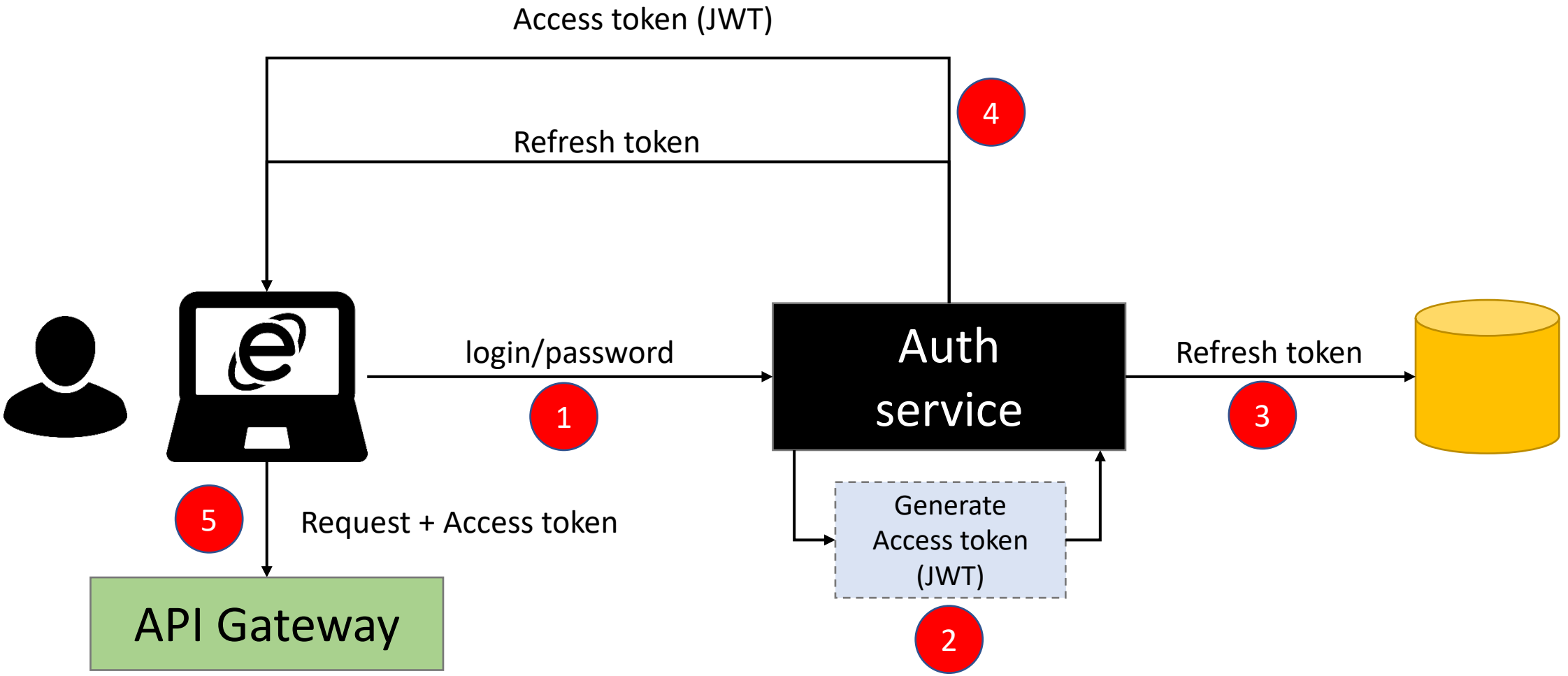


SKLADCHIK.ORG

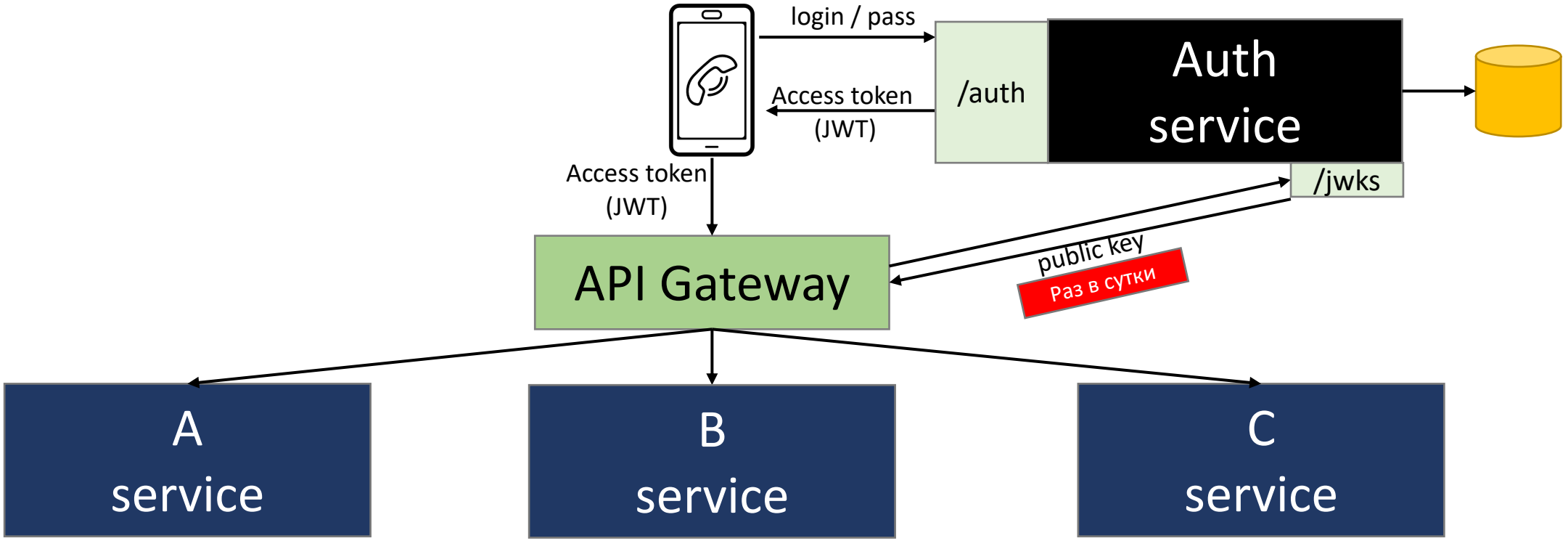
# Аутентификация в сервисах – плохо



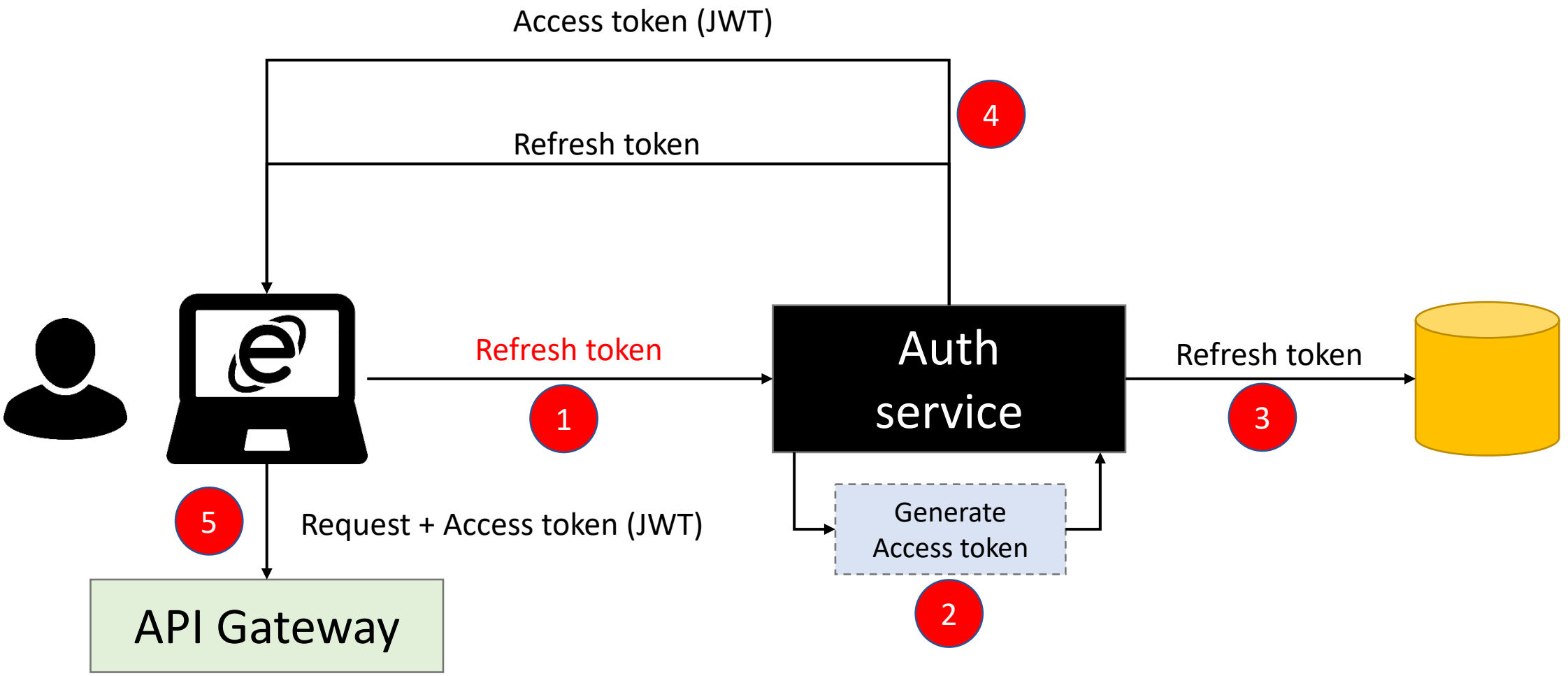
# Сервер аутентификации и JWT



# Проверка JWT токена



# Перевыпуск JWT токена



# Популярные сервера аутентификации

# Популярные решения


https://www.keycloak.org/


https://www.ory.sh/docs/hydra

# О чем узнали

- Количество Api Gateway определяется количеством уникальных фронтенд приложений или внешних потребителей. Этот подход называют BFF
- Настраивать Api Gateway очень легко, для этого не нужен администратор или выделенная команда
- Аутентификация с помощью JWT токенов позволяет выдерживать высокие нагрузки

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

# Архитектура сервиса и хранение состояния

# Цели занятия

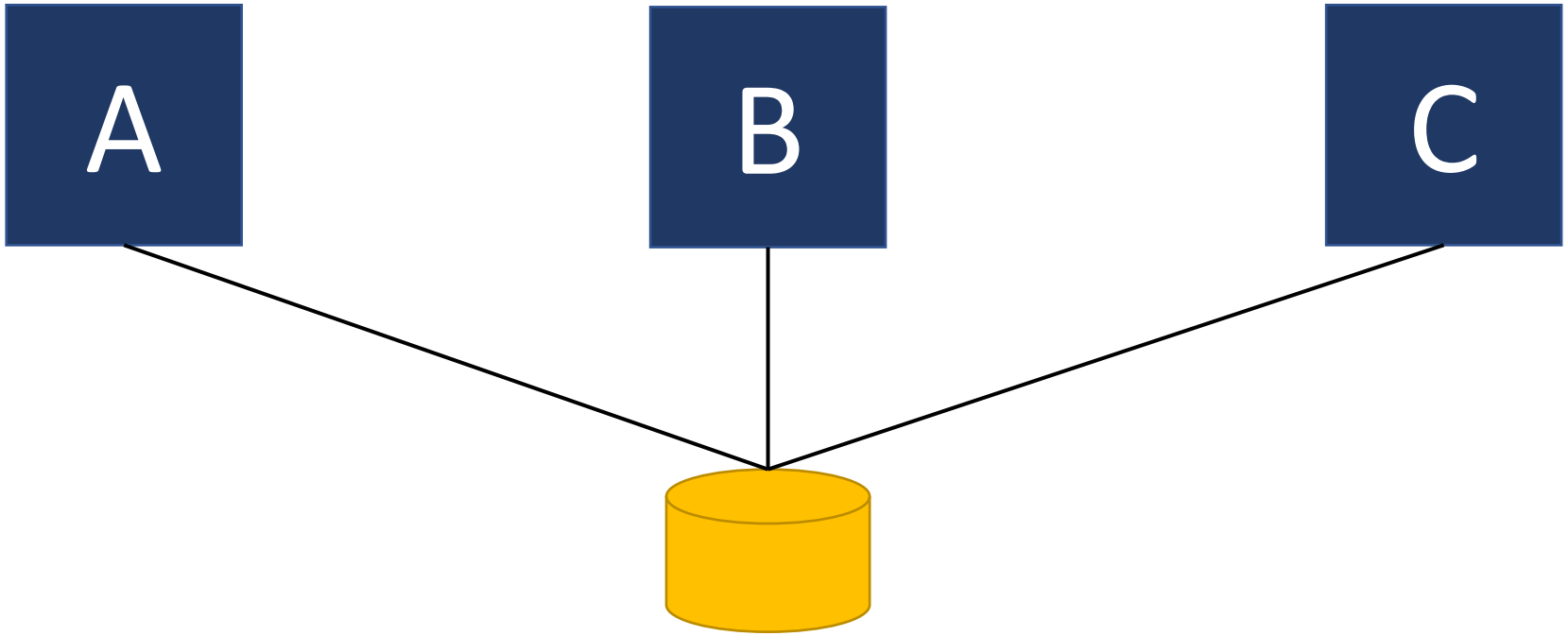
- Узнать, как микросервисы сохраняют своё состояние
- Разобраться, какая архитектура может быть внутри микросервиса
- Понять, какой код можно выносить в общие библиотеки, а какой нет

SKLADCHIK.ORG

# Организация работы с данными

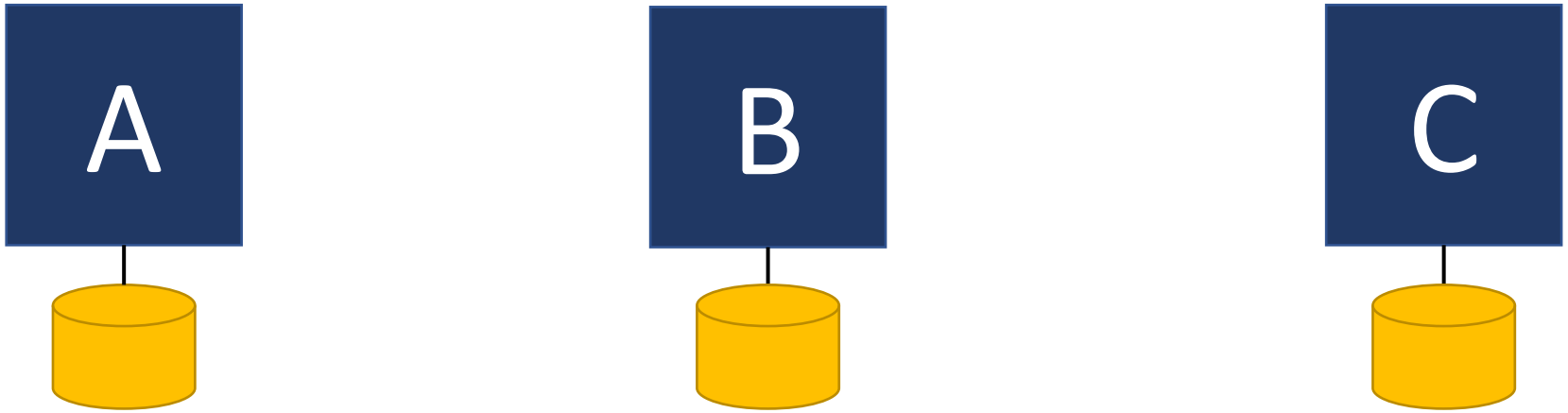
# Shared database anti-pattern

# Shared database anti-pattern

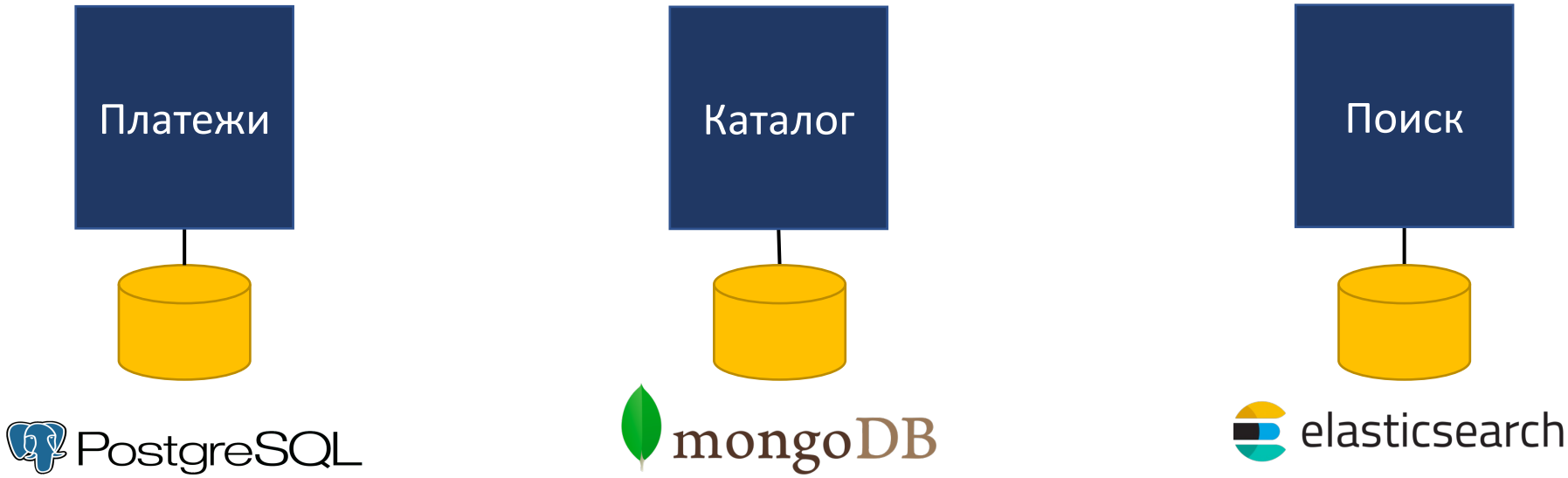


# Database per service pattern

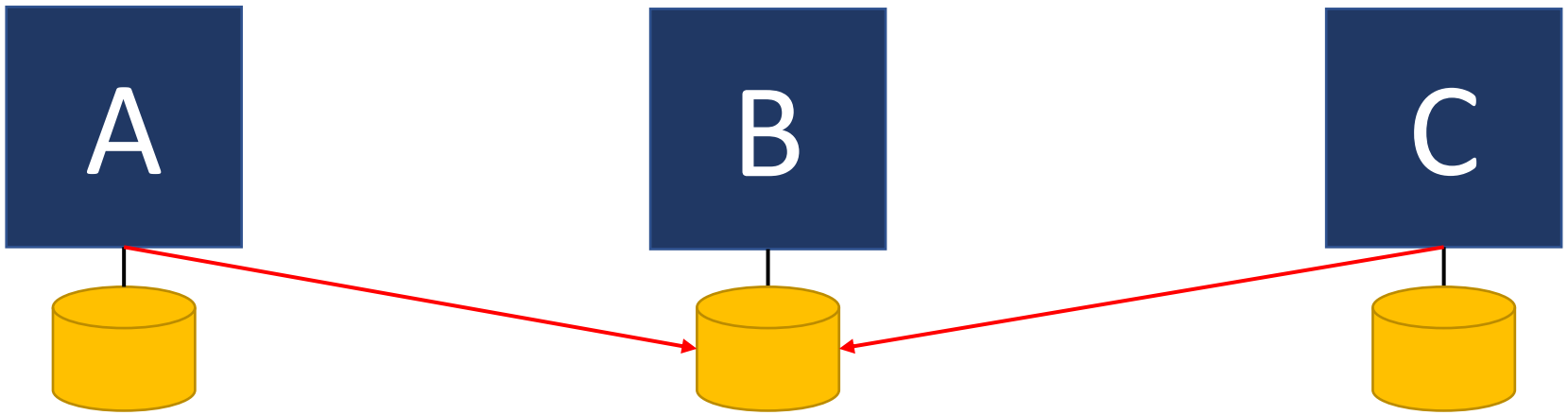
# У каждого сервиса своя БД



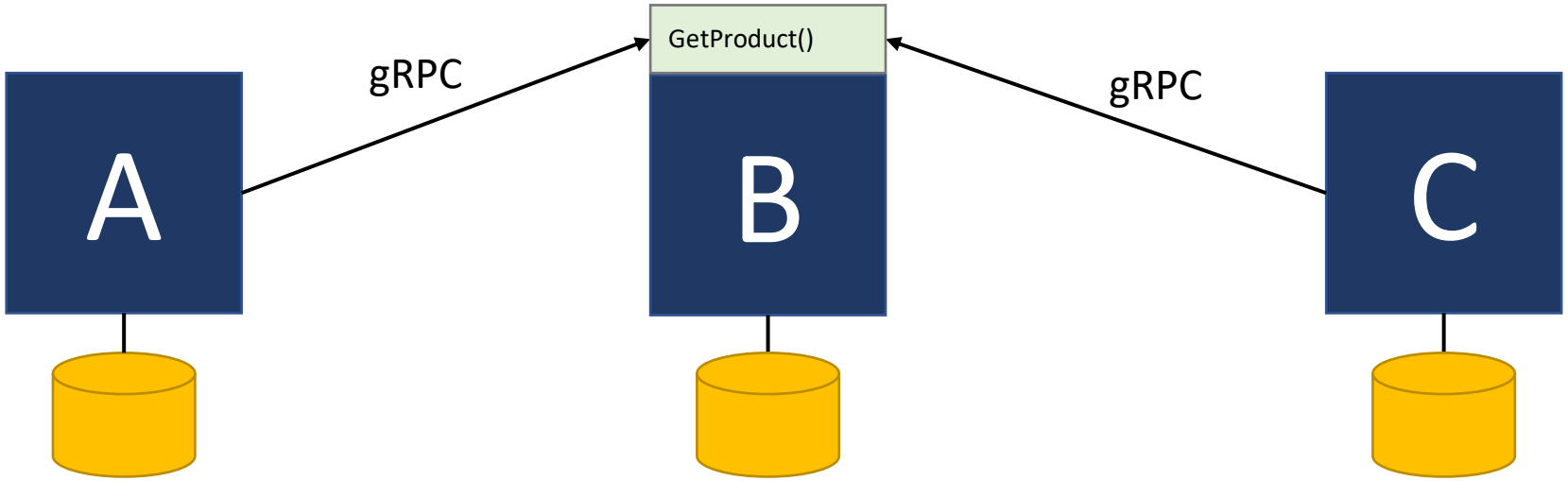
# Лучшая технология для задачи



# Так нельзя



# Так можно

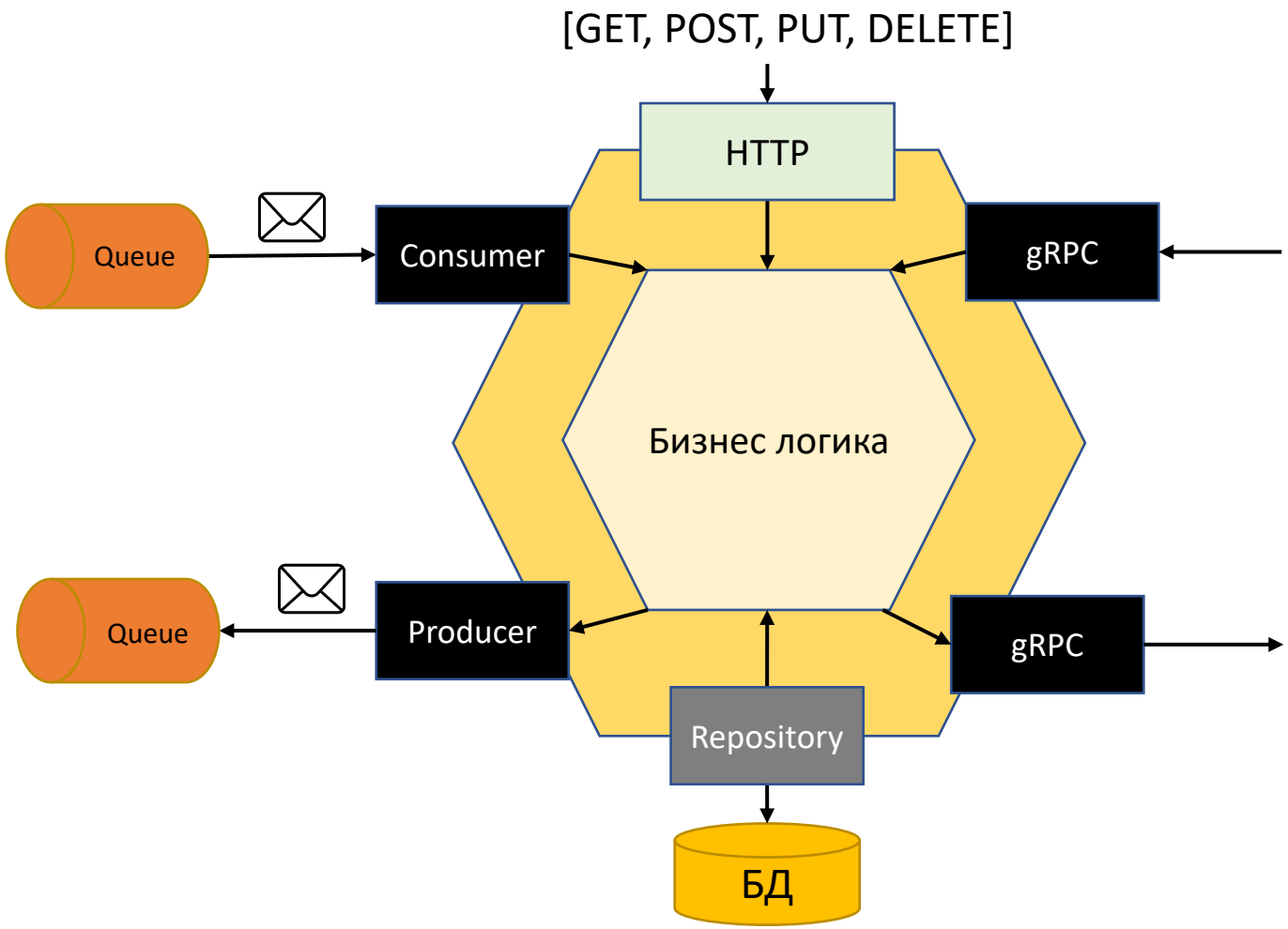


# Внутренняя архитектура сервиса

# Для разных задач - разная архитектура

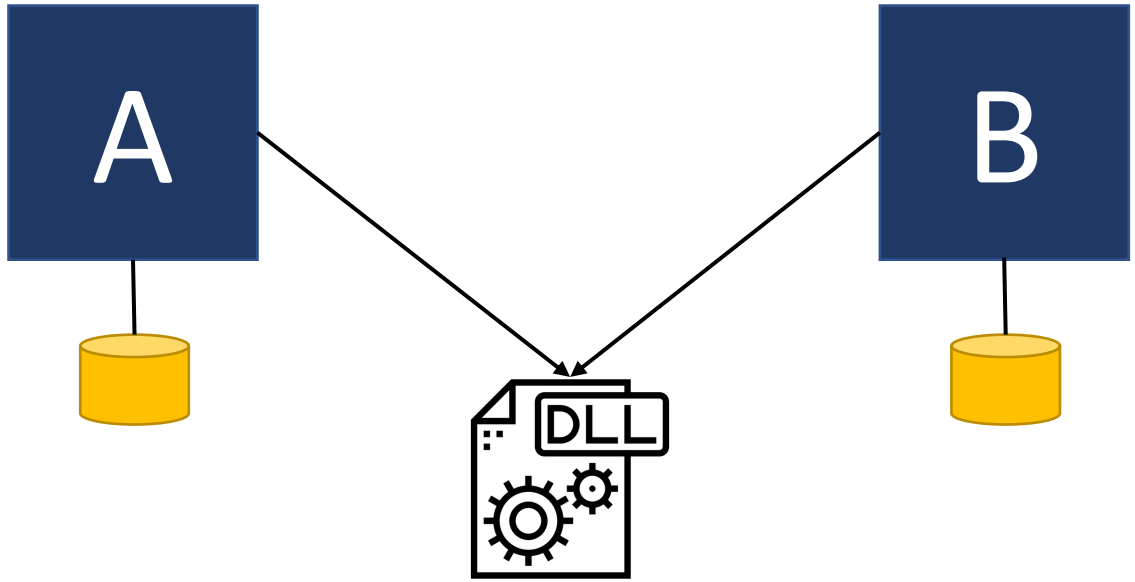
SKLADCHIK.ORG

# Гексагональная архитектура



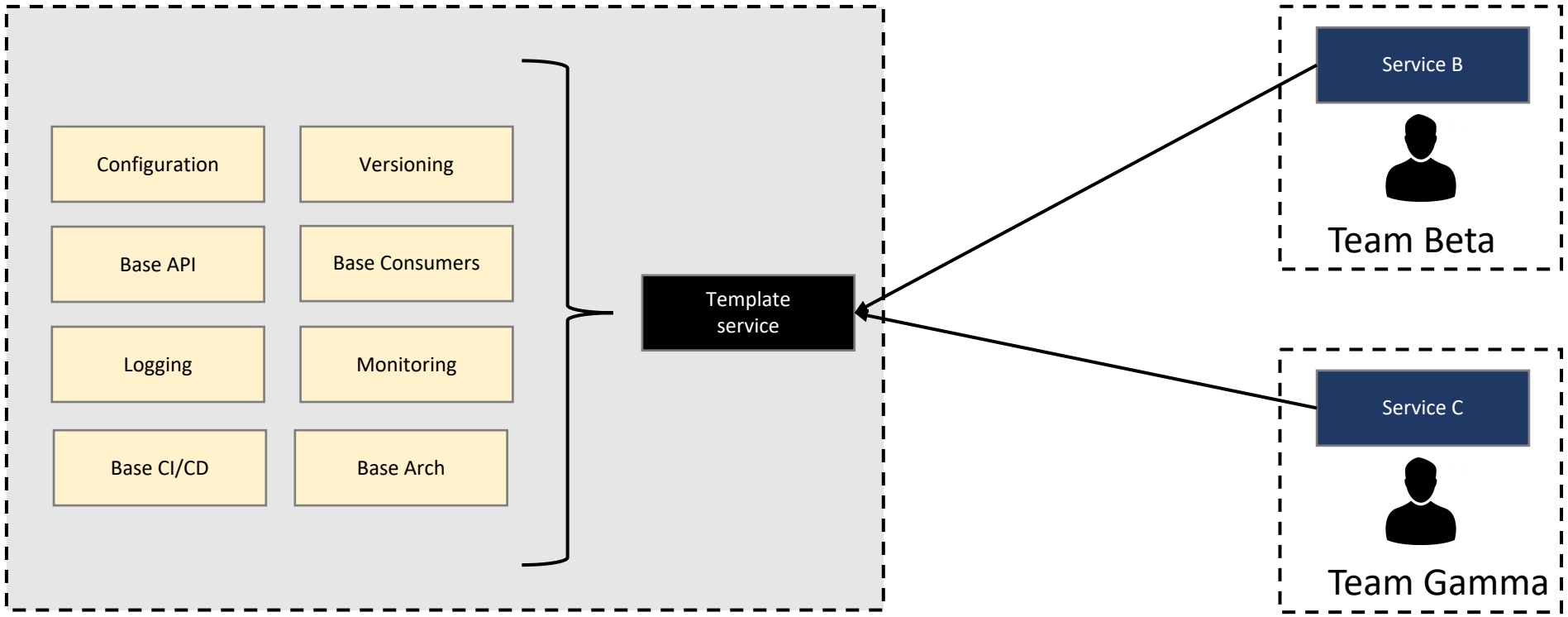
# Повторное использование кода

# Shared library anti-pattern



# Service template pattern

# Шаблон сервиса



# О чём узнали

- Использовать общую базу данных для интеграции разных сервисов - плохая идея
- Использовать общую библиотеку с бизнес сущностями в разных сервисах - плохая идея
- Новый сервис лучше и дешевле начинать с Service Template

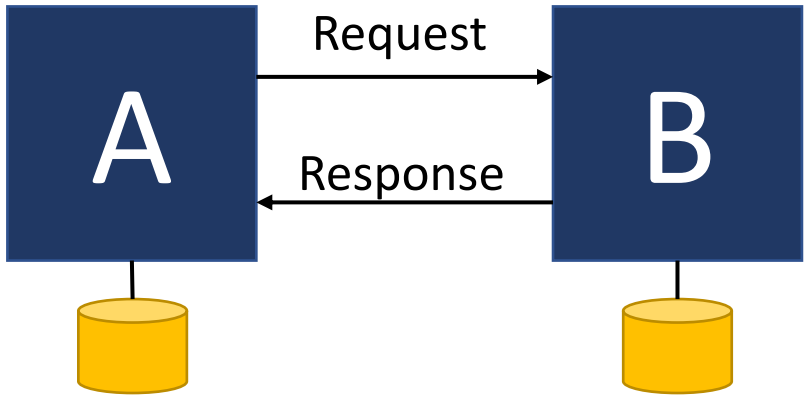
# Взаимодействие между микросервисами

# Цели занятия

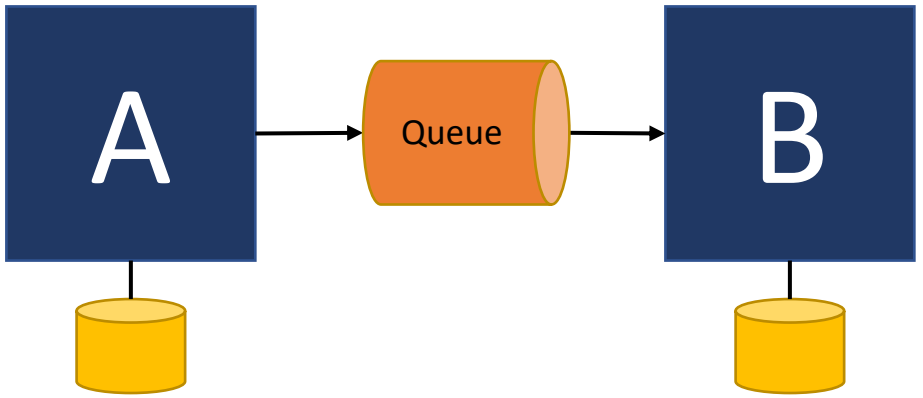
- Разобраться какие бывают способы межсервисного взаимодействия
- Разобраться какие способы межсервисного взаимодействия делают систему более отказоустойчивой и быстрой
- Понять, как обеспечить согласованность данных между микросервисами
- Разобраться, как управлять сквозными процессами в микросервсной архитектуре

# Синхронное/Асинхронное взаимодействие

**Синхронное**




**Асинхронное**

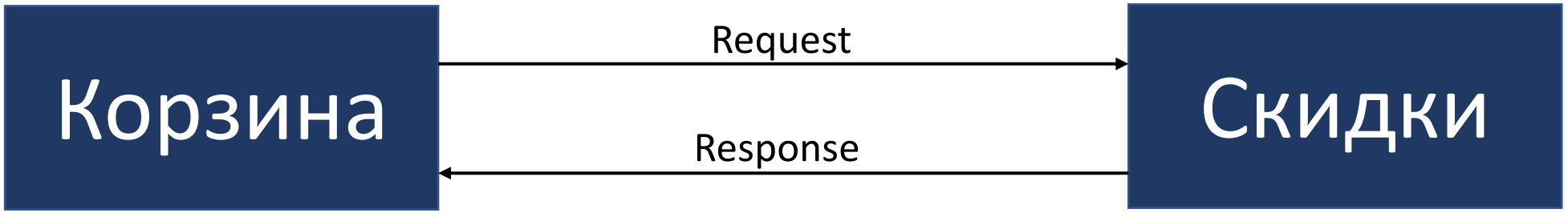


# Синхронное взаимодействие

SKLADCHIK.ORG

# Remote Procedure Call (RPC) pattern

# Пример



# Когда выбирать

- Если требуется мгновенный отклик и задержки недопустимы, синхронное взаимодействие может быть предпочтительным.
- Если сервису A нужны данные из сервиса B, но не целесообразно их асинхронно реплицировать из B в A

# Плюсы и минусы

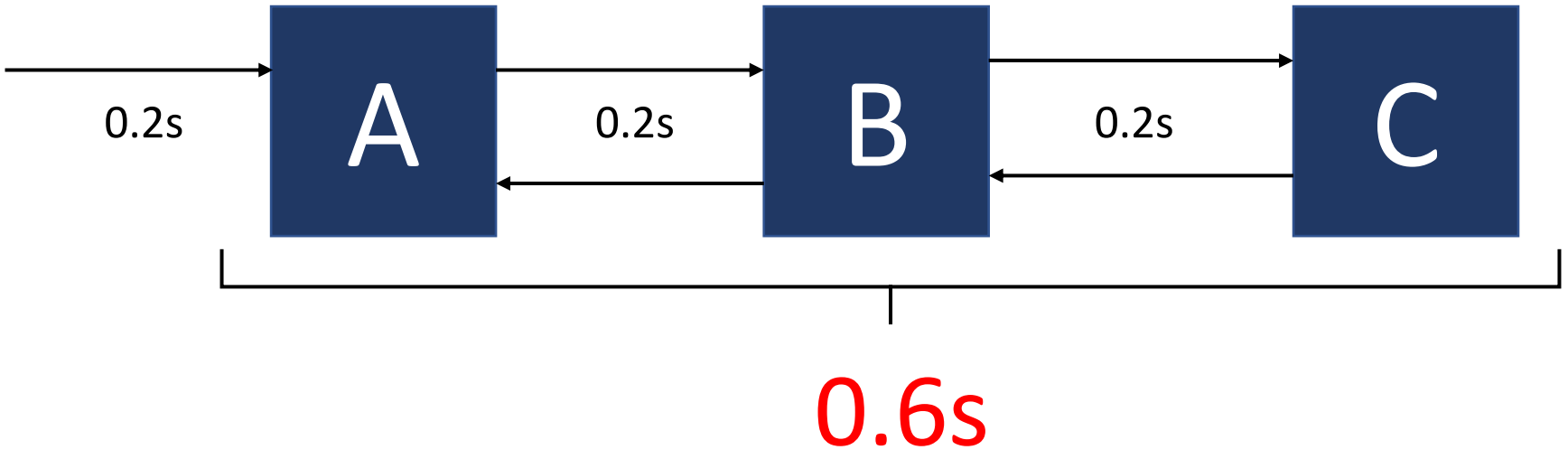
**Плюсы**

- **Простота**. Синхронное взаимодействие проще в реализации и отладке.
- **Прозрачность**. Синхронное взаимодействие позволяет легко отслеживать и управлять последовательностью выполнения операций.

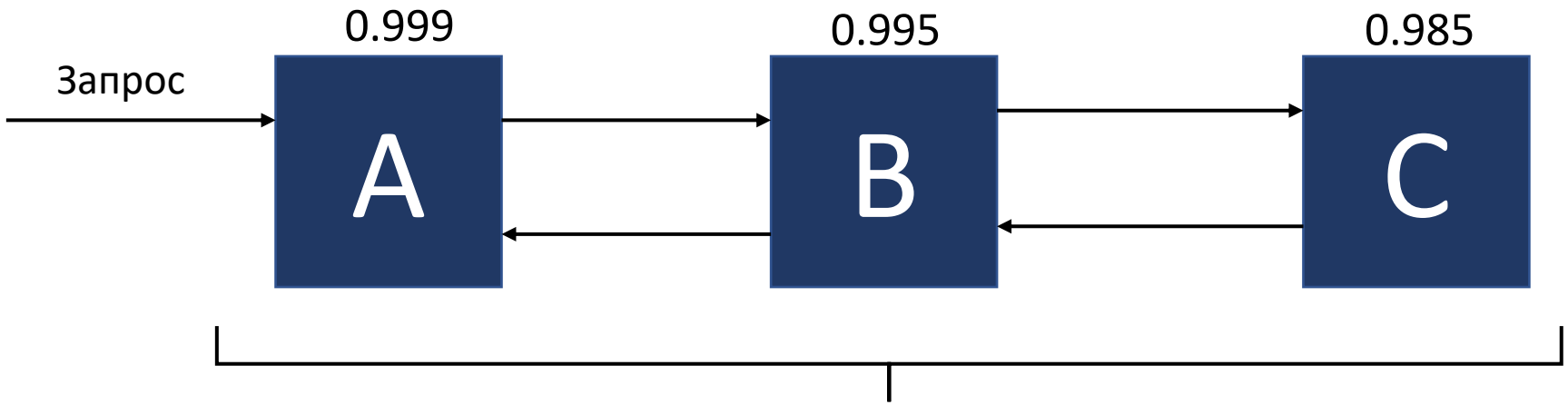
**Минусы**

- **Снижение доступности.** Если вызываемый микросервис недоступен или работает медленно, это может привести к задержкам и блокировкам в клиентском микросервисе.
- **Узкое место производительности.** Если синхронные вызовы выполняются последовательно, это может стать узким местом производительности.

# Узкое место производительности



# Снижение доступности

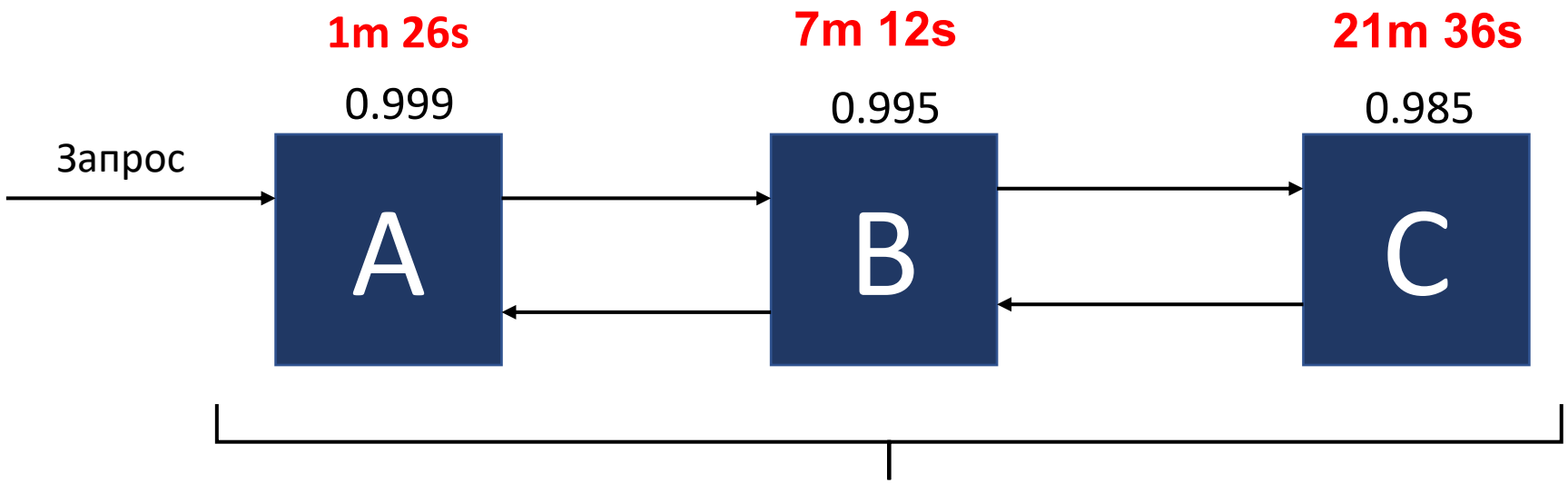



Request SLA = 0.999 * 0.995 * 0.985

0.979

(97,9%)

# Downtime запроса в день




Request SLA = 0.999 * 0.995 * 0.985

**30m 14s**

0.979
(97,9%)

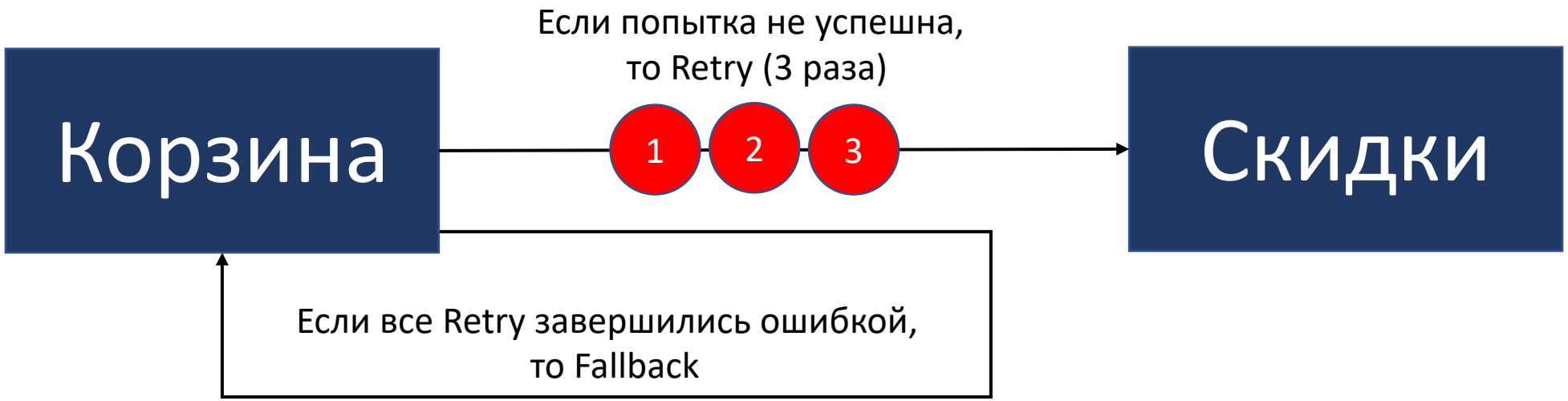
# Downtime в день

| Сервисов в запросе | SLA | Downtime в день |
|---|---|---|
| 1 | 0.999 (99,9%) | 1m 26s |
| 2 | 0.994 (99,4%) | 8m 38s |
| 3 | 0.979 (97,9%) | 30m 14s |
| 9 | 0.938 (93.8%) | 1h 29m 17s |

# Защита от сбоев

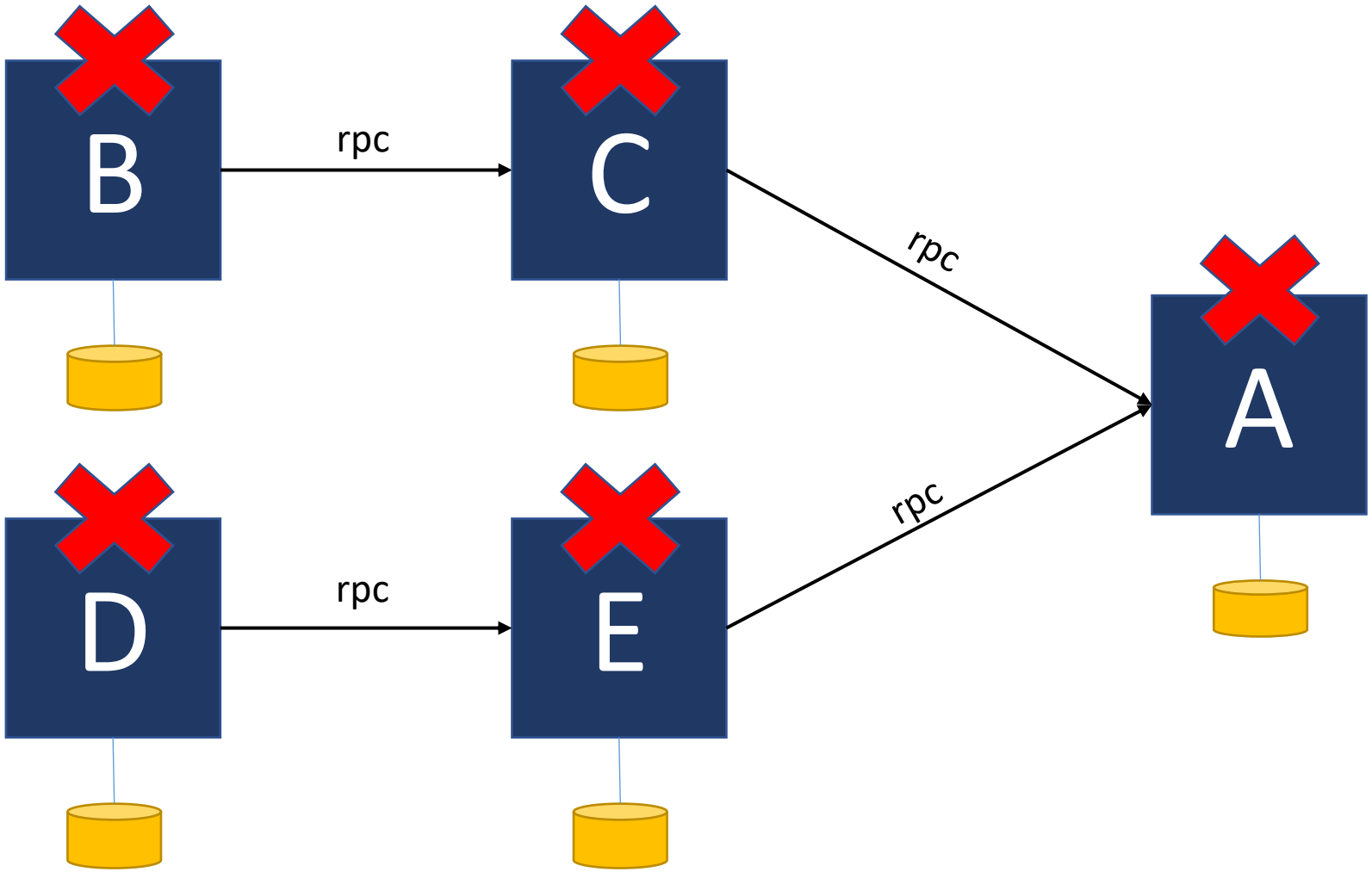
# Fallback + Retry

```
Policy<Discount>
    .Handle<SomeException>()
    .OrResult(null)
    .Retry(3)
    .Fallback<Discount>(() => Discount.GetRandomDiscount())
```



SKLADCHIK.ORG

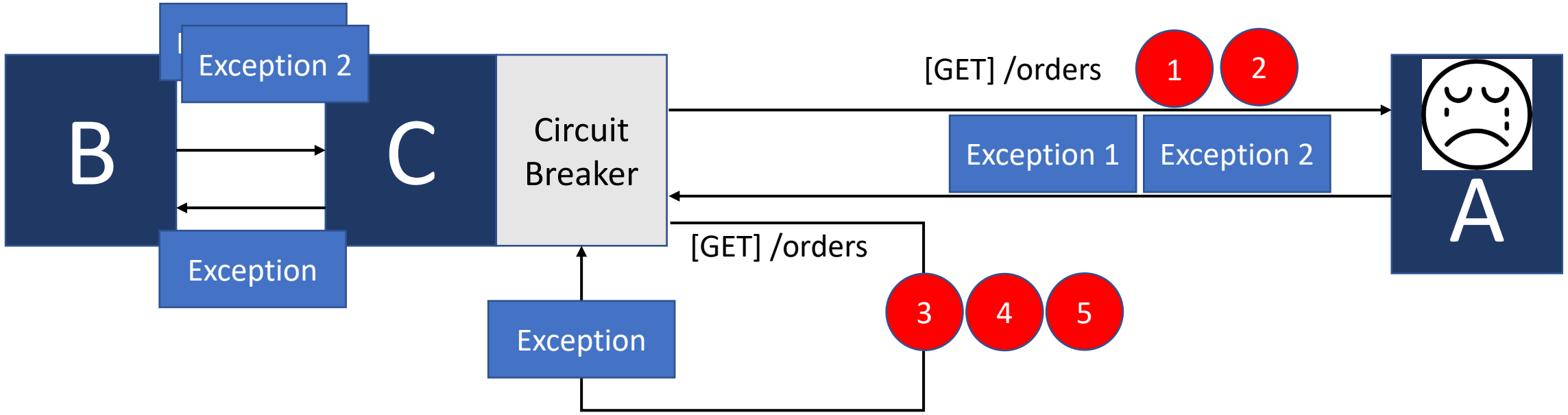
# Каскадные сбои



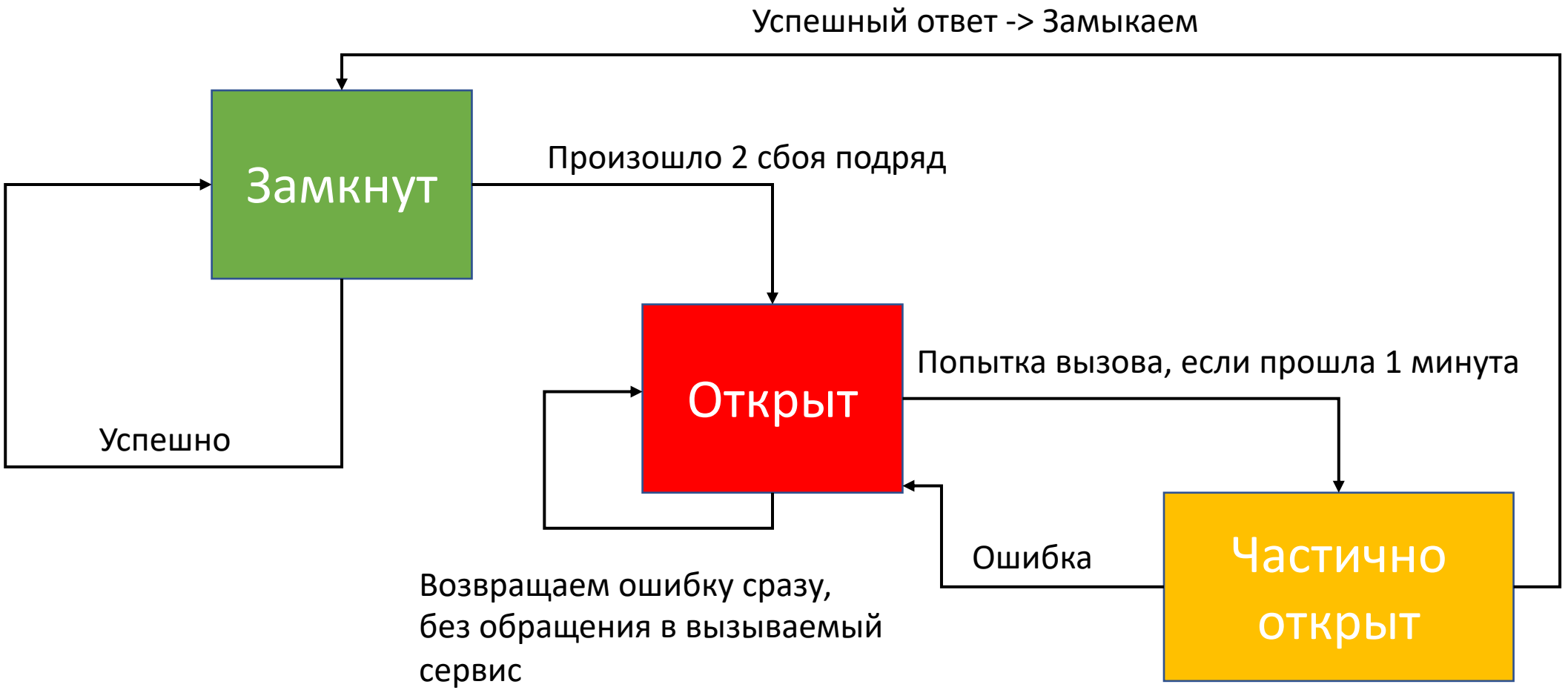
# Circuit Breaker pattern

# Circuit Breaker (предохранитель)

```
Policy
    .Handle<TimeOutException>()
    .CircuitBreaker(2, TimeSpan.FromMinutes(1));
```



# Circuit Breaker (принцип работы)



# Популярные решения

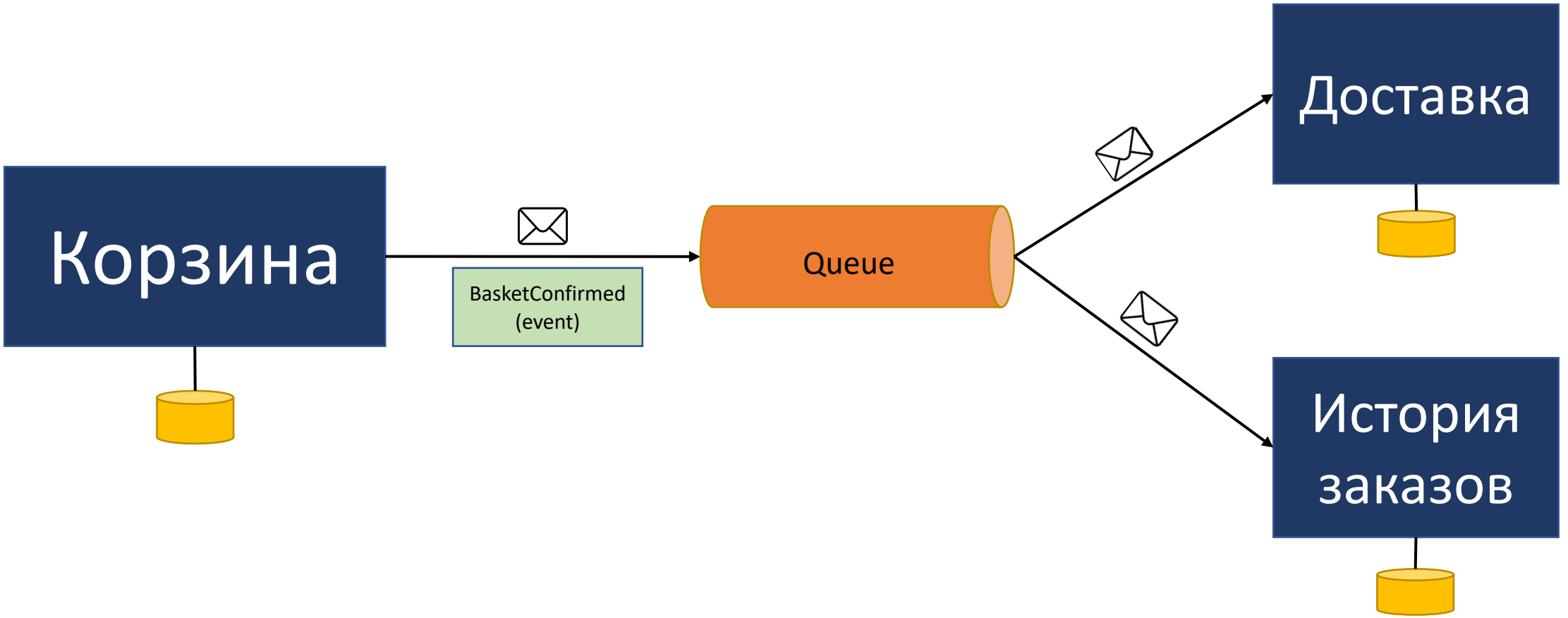

Polly


Istio Circuit Breaking

# Асинхронное взаимодействие

# Messaging pattern

# Пример


Корзина
BasketConfirmed
(event)
Queue
Доставка
История
заказов

# Когда выбирать

- Если НЕ требуется мгновенный отклик и задержки допустимы, асинхронное взаимодействие может быть предпочтительным.
- **Проще говоря – всегда, кроме случаев, когда синхронная связь более предпочтительна.**

# Плюсы и минусы

## Плюсы

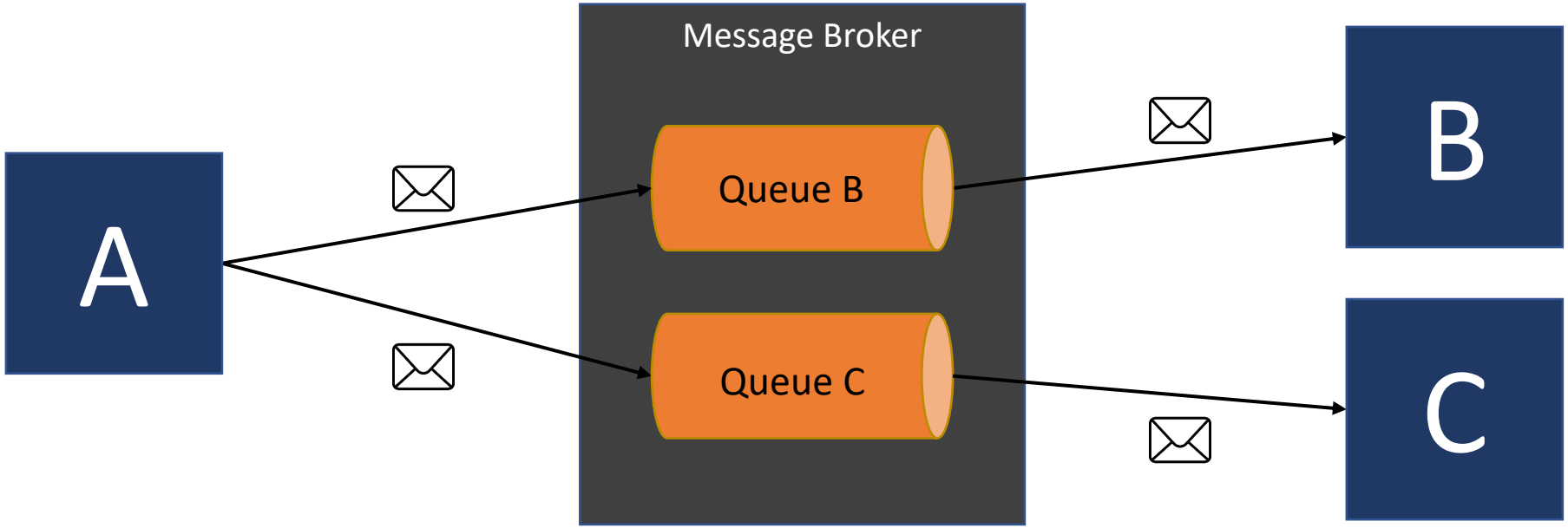
- **Отказоустойчивость**. Асинхронное взаимодействие позволяет избежать блокировки клиентского микросервиса при недоступности вызываемого микросервиса.
- **Масштабируемость**. Асинхронное взаимодействие может быть параллельным, что способствует лучшей масштабируемости системы.

## Минусы

- **Сложность.** Асинхронное взаимодействие требует более сложной реализации, так как необходимо обрабатывать асинхронные ответы и управлять состоянием запросов.
- **Усложнение отладки.** Отслеживание и отладка асинхронного взаимодействия может быть сложнее из-за распределения запросов и ответов во времени.

SKLADCHIK.ORG

# Брокер сообщений



# Плюсы и минусы

**Плюсы**

- Слабая связанность
- Буферизация сообщений
- Гибкое взаимодействие

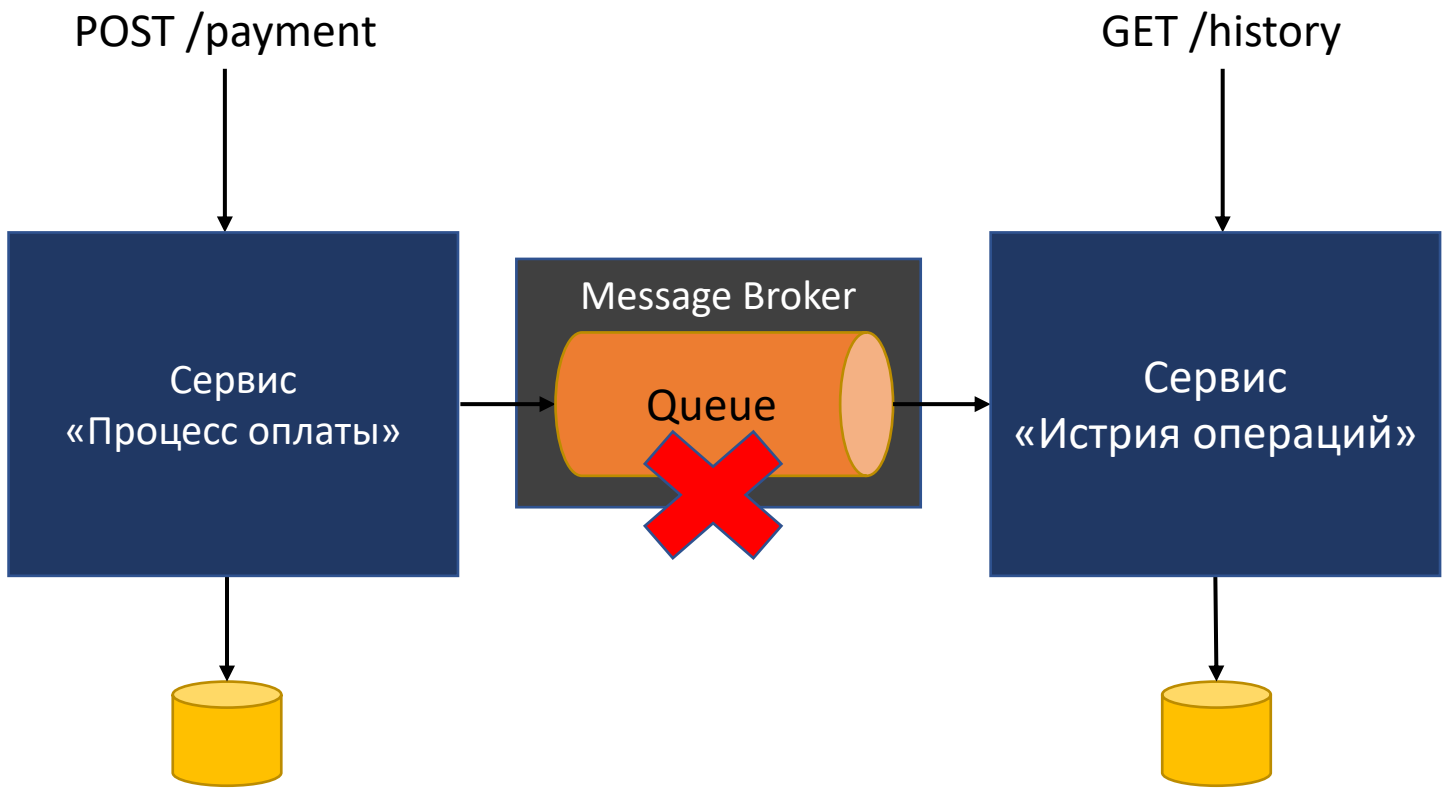
**Минусы**

- Потенциальное узкое место производительности
- Потенциальная единая точка отказа
- Дополнительная сложность в администрировании

Отложенная согласованность

(eventual consistency)

# Проблема


POST /payment
GET /history
Сервис
«Процесс оплаты»
Message Broker
Queue
Сервис
«Истрия операций»

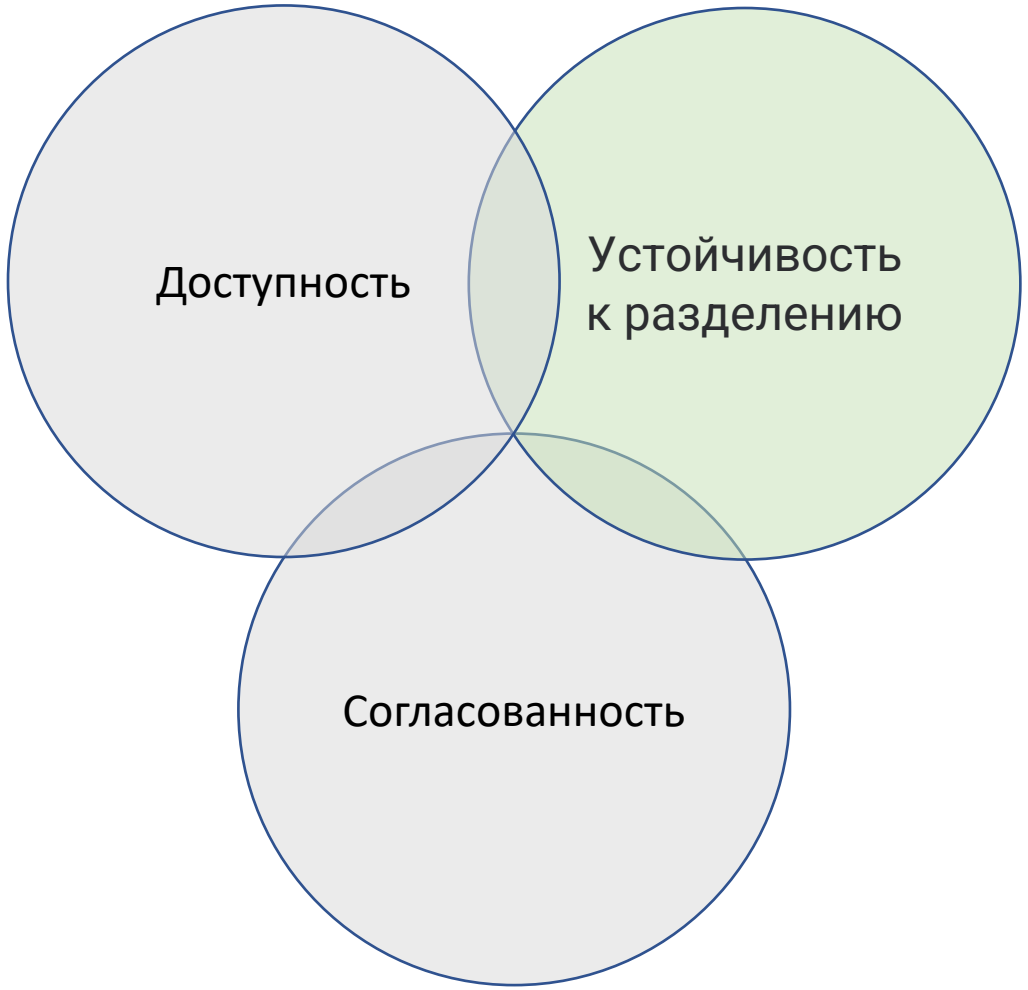
# CAP теорема

# CAP теорема


В случае сбоя в распределённой системе можно обеспечить только два свойства из трёх:

1. согласованность
2. доступность
3. устойчивость к разделению.

# CAP теорема

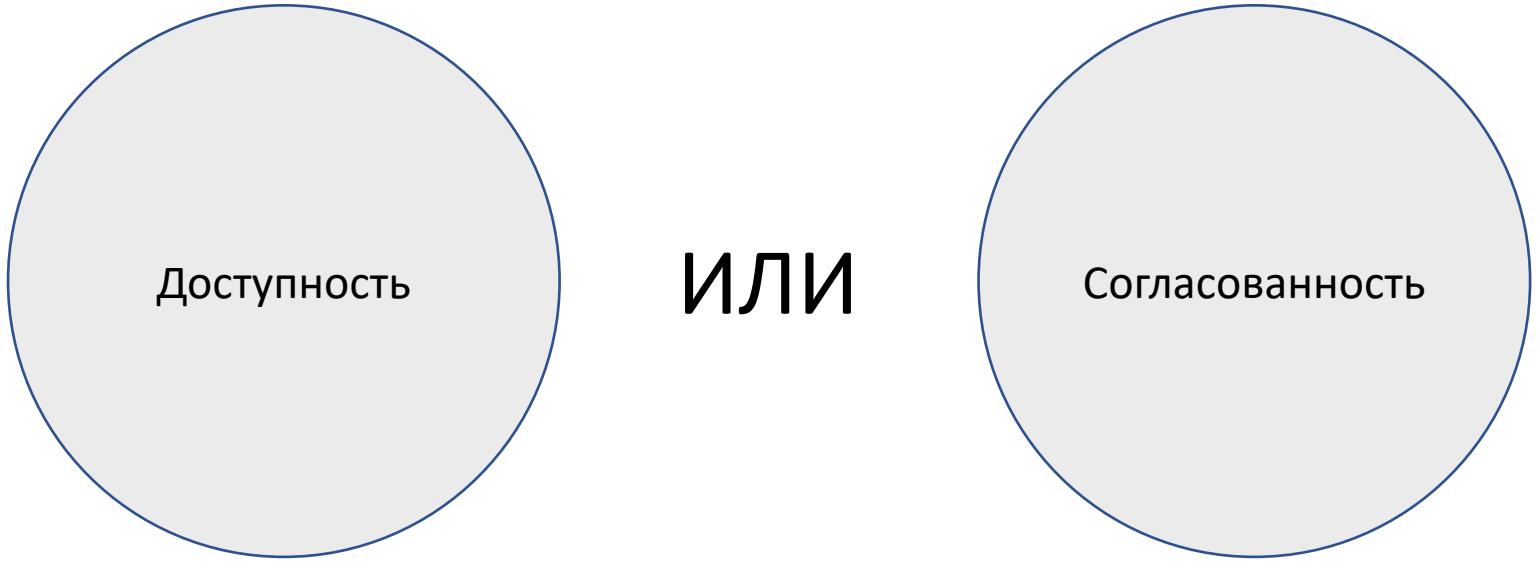



В случае сбоя в распределенной системе можно обеспечить только два свойства из трех:

1. согласованность
2. доступность
3. ✓ устойчивость к разделению.

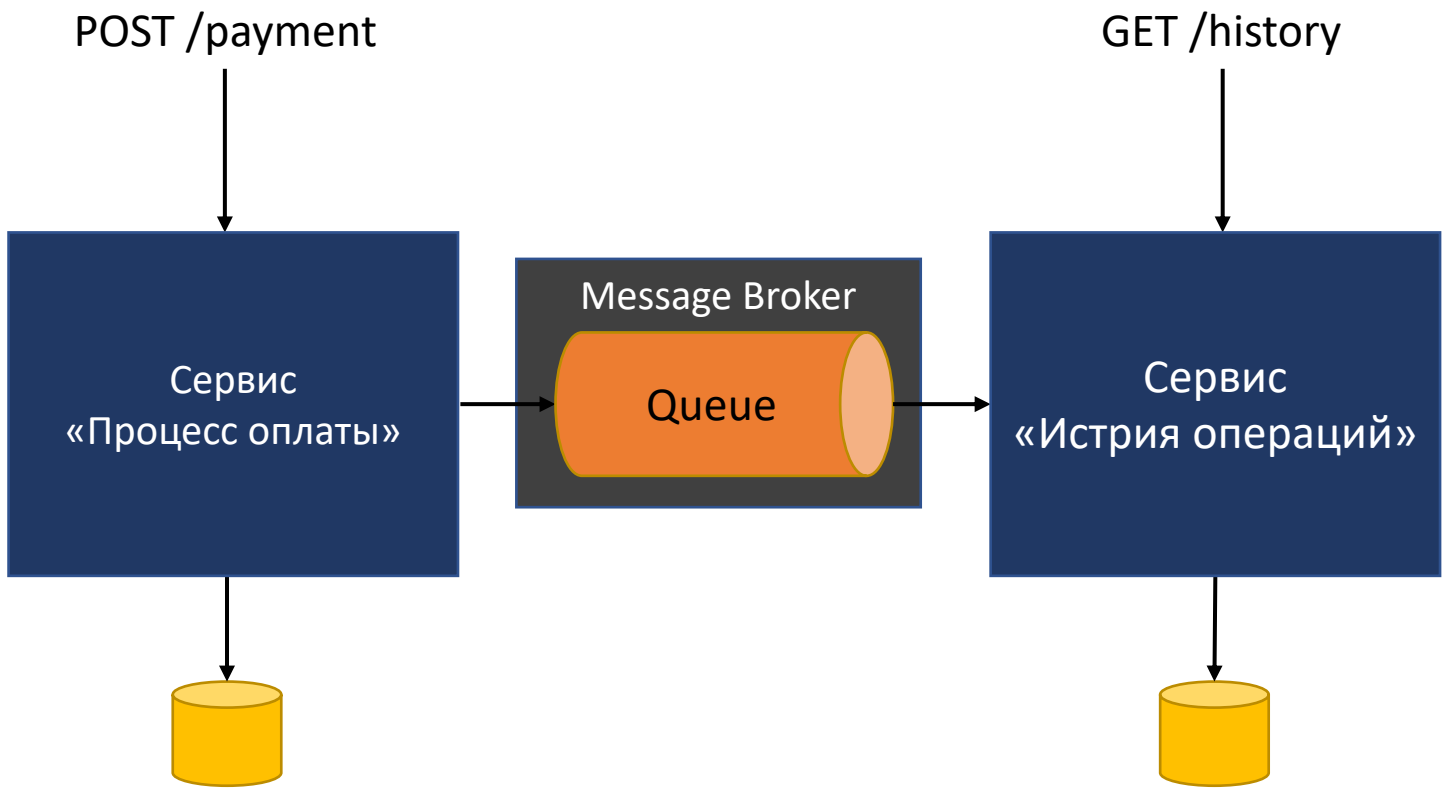
# Выбор

**При отказе в распределенной системе у нас стоит выбор каким свойством системы пожертвовать.**



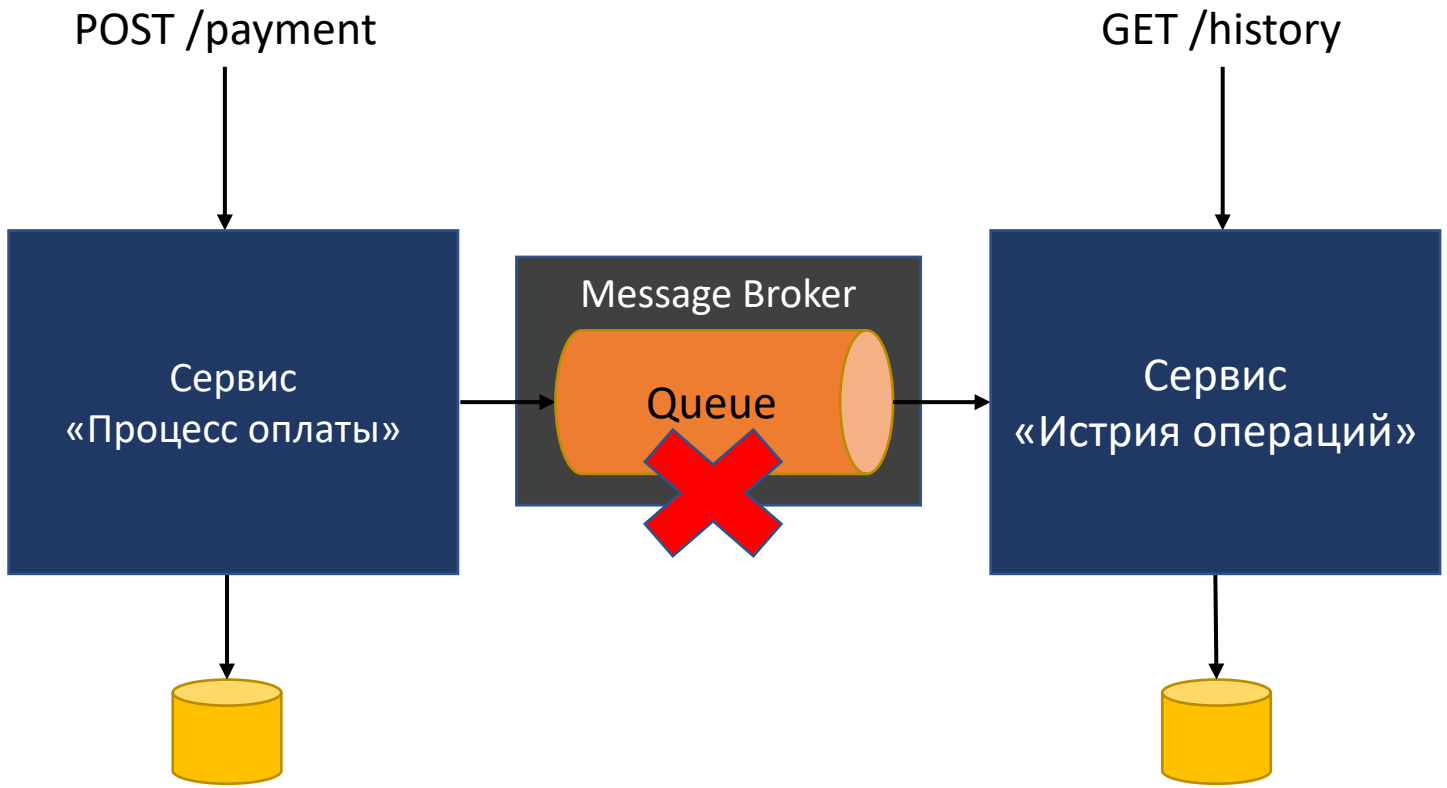
# Пример

# Пример

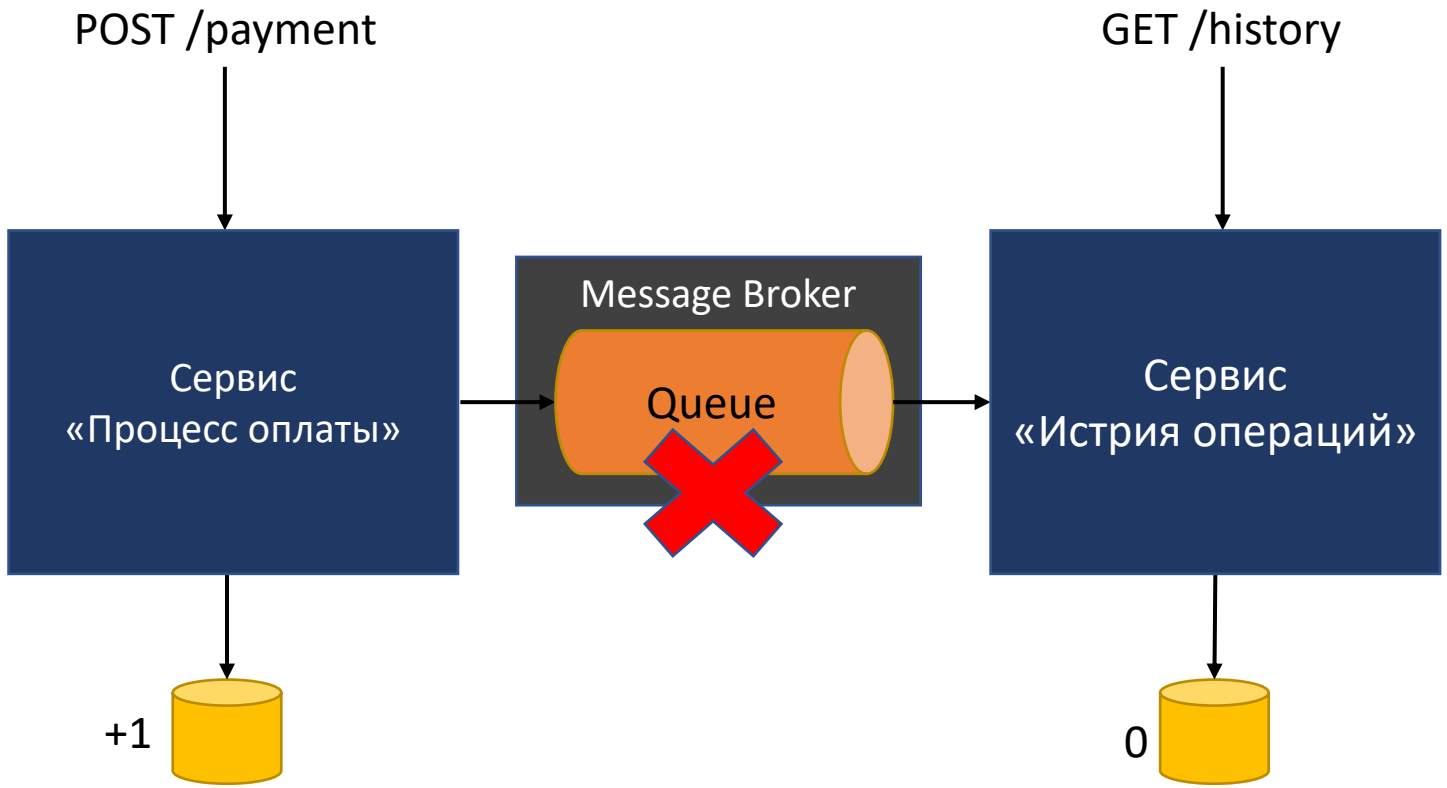

POST /payment
GET /history
Сервис
«Процесс оплаты»
Message Broker
Queue
Сервис
«Истрия операций»

SKLADCHIK.ORG

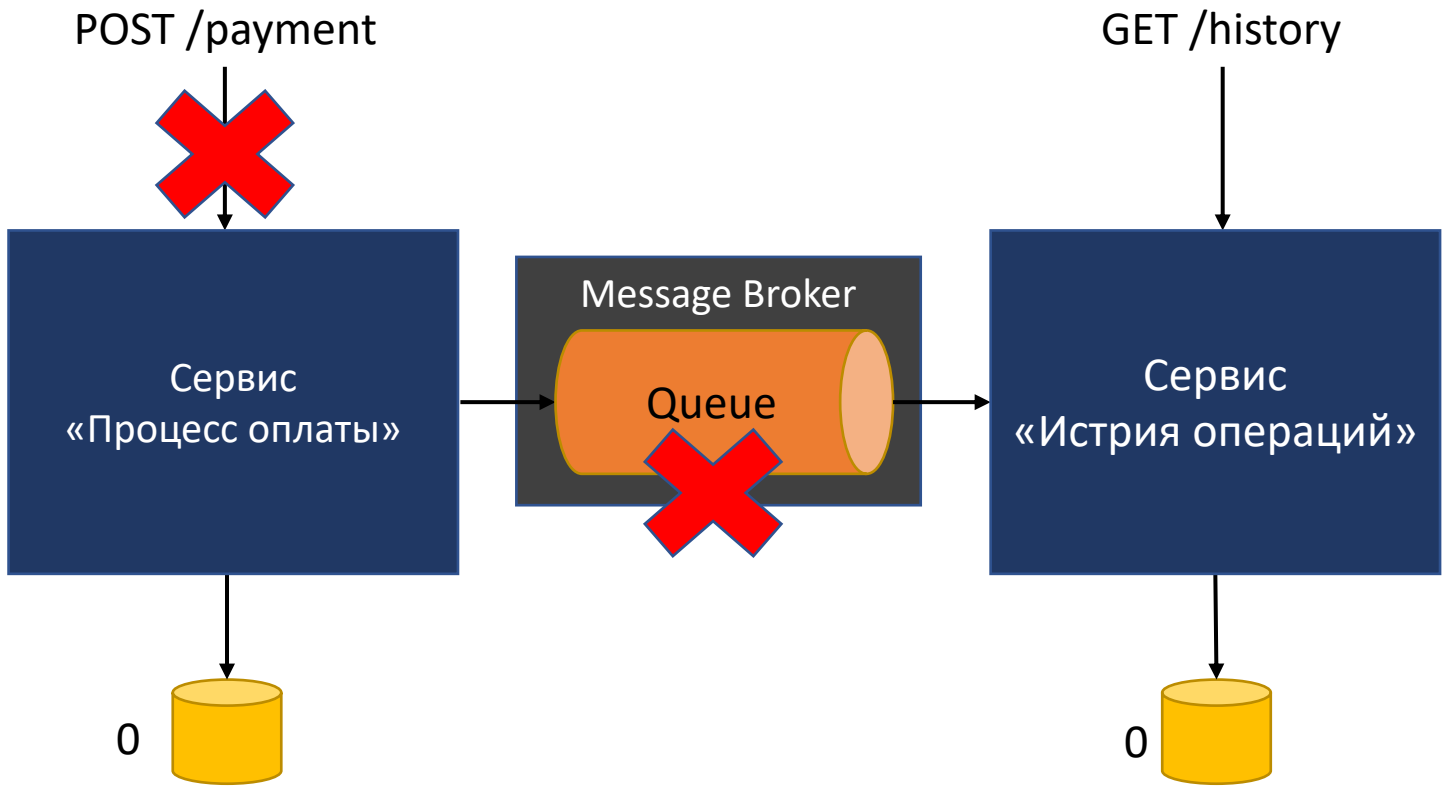
# Пример



# Жертвуем согласованностью

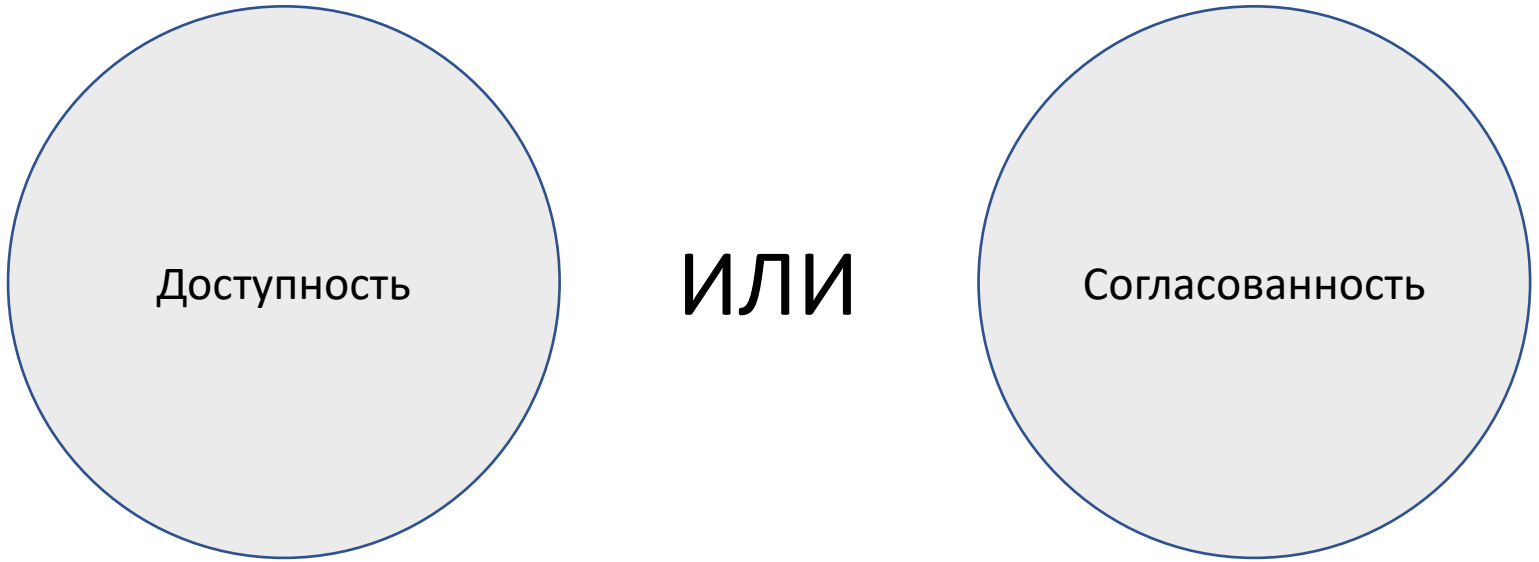


# Жертвуем доступностью



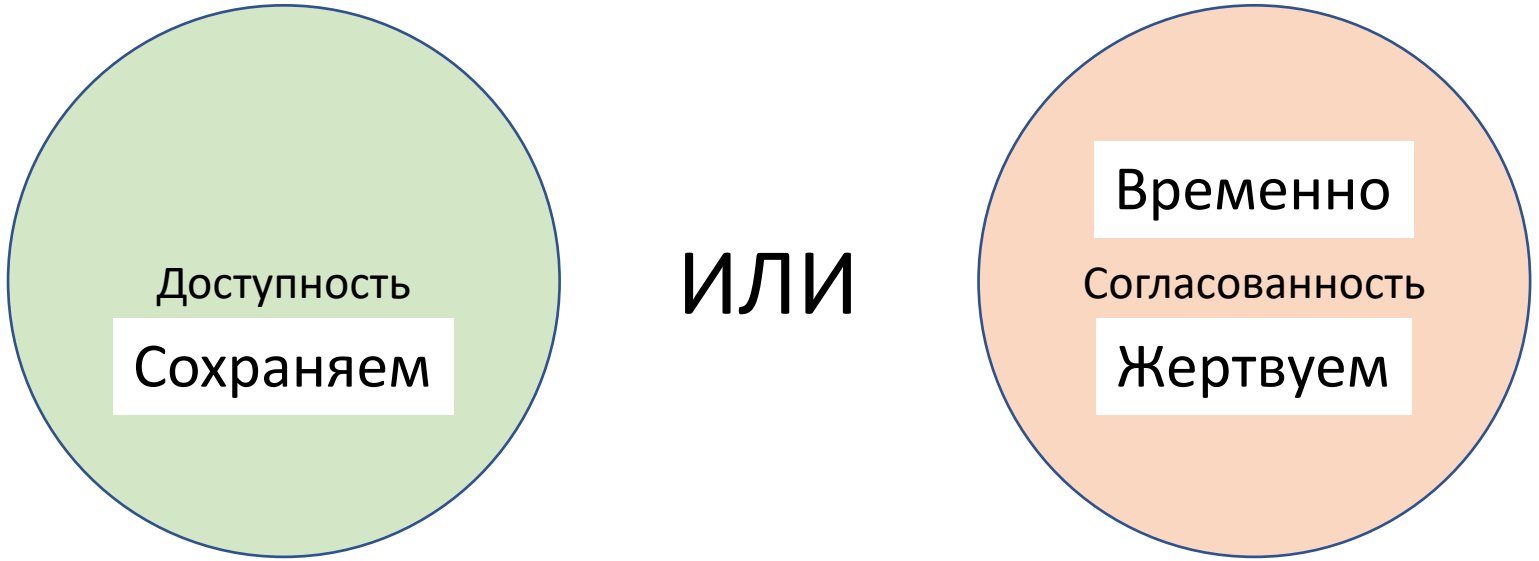
# Что выбрать?

**При отказе в распределенной системе у нас стоит выбор каким свойством системы пожертвовать.**



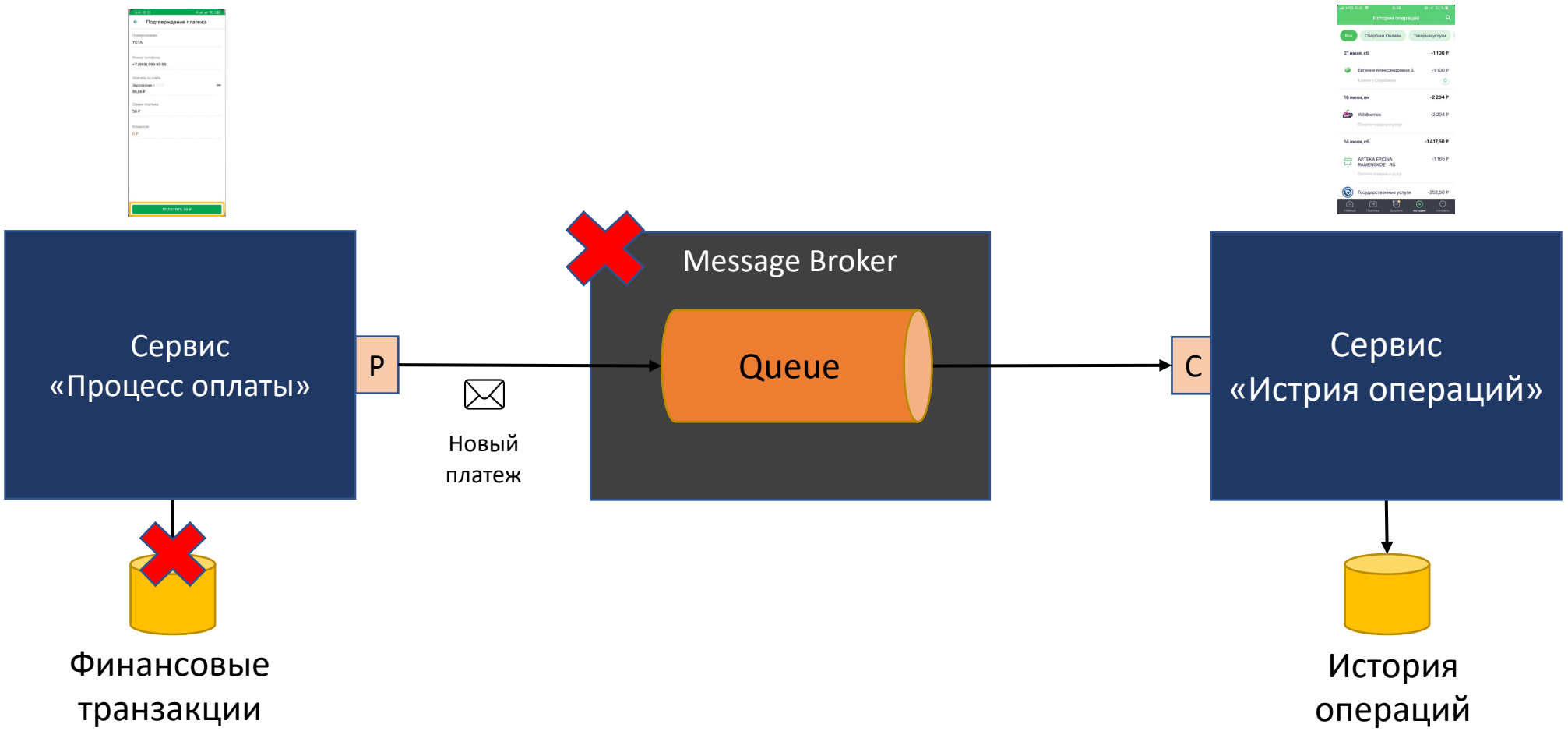
# Что выбрать?

**При отказе в распределенной системе у нас стоит выбор каким свойством системы пожертвовать.**



# Обеспечение
# отложенной согласованности

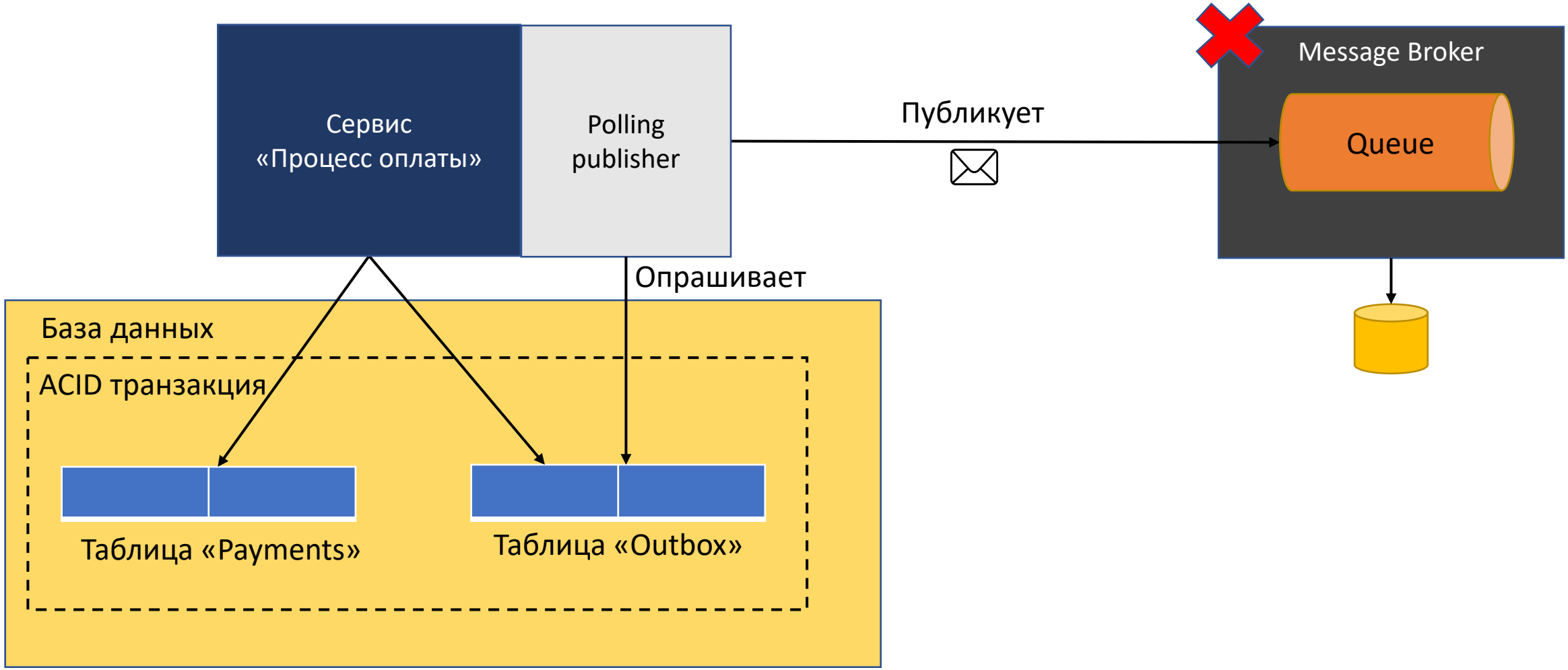
# Обеспечение согласованности



Transactional outbox pattern

Polling publisher pattern

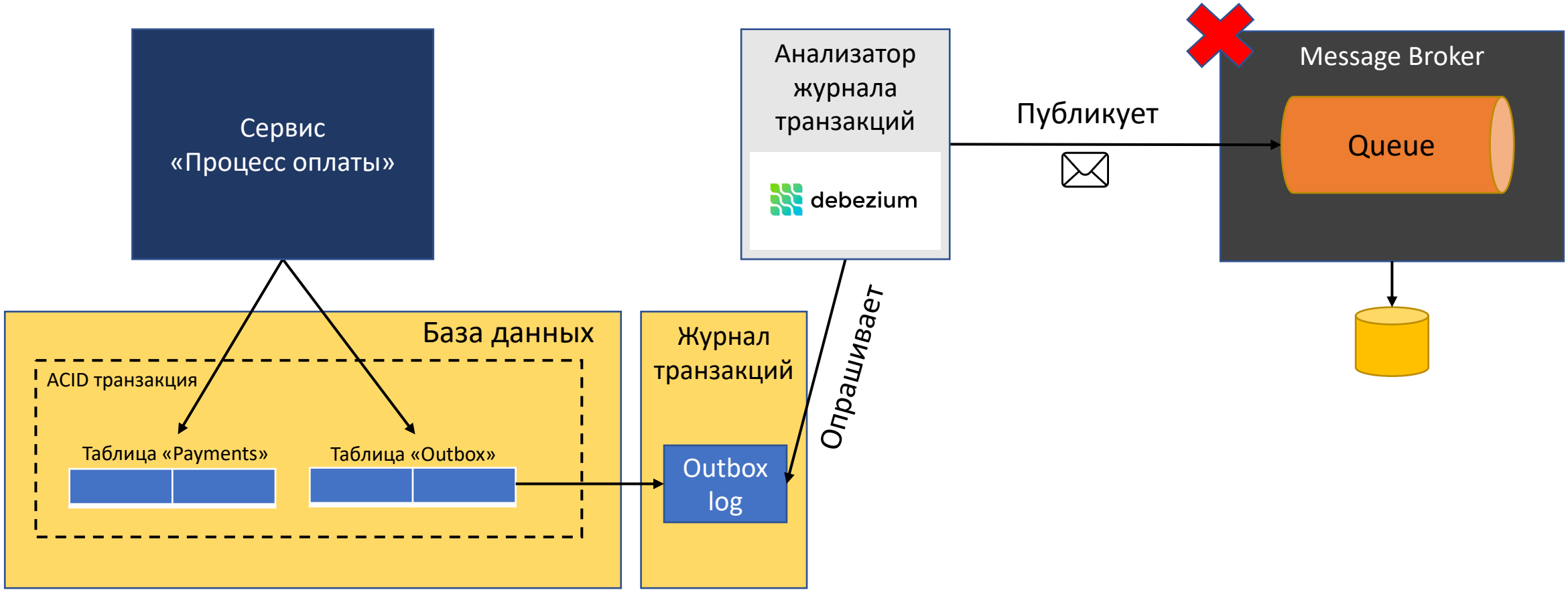
# Transactional outbox + Polling publisher



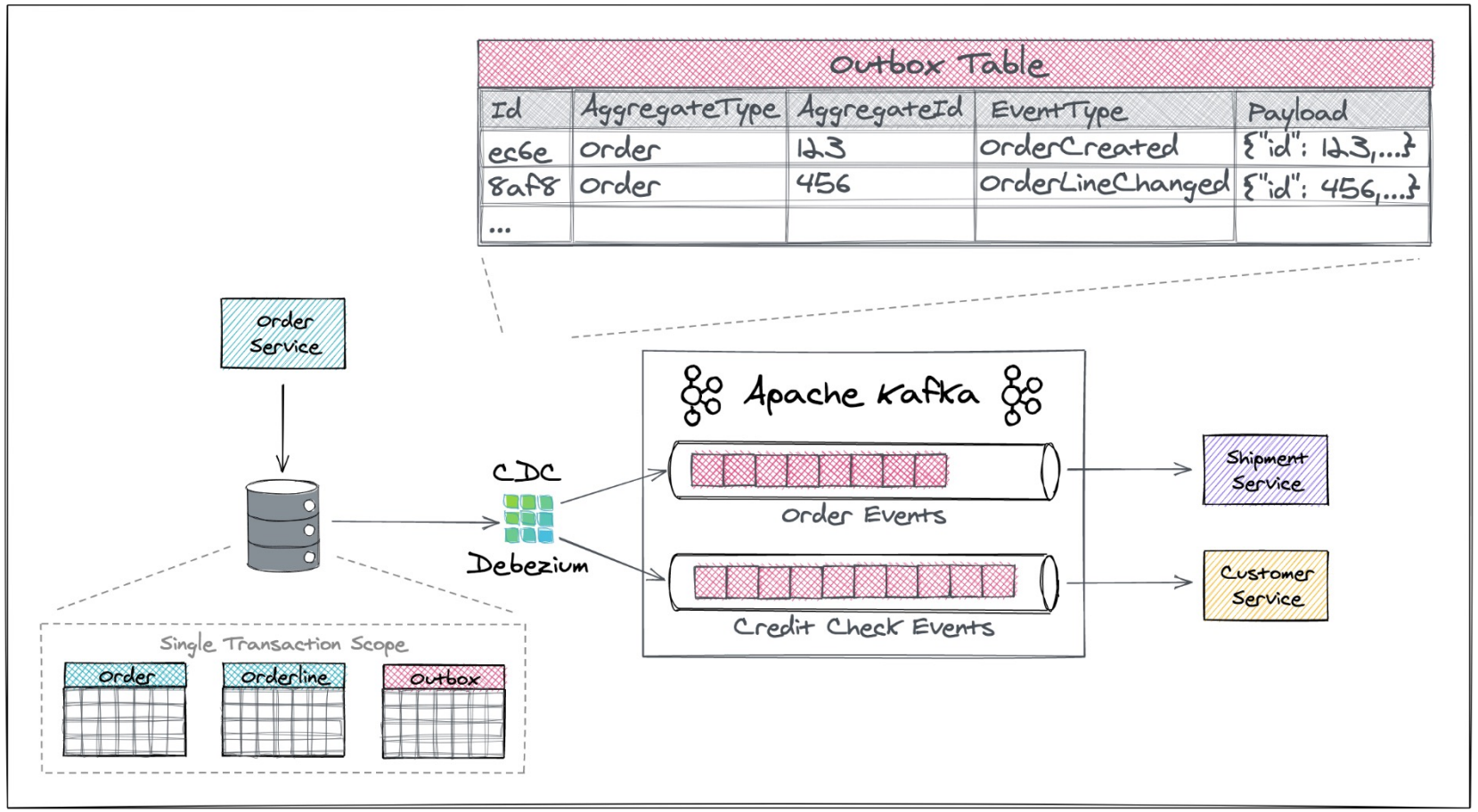
# Transaction log tailing pattern

SKLADCHIK.ORG

# Transaction log tailing + Outbox



# Структура таблицы Outbox



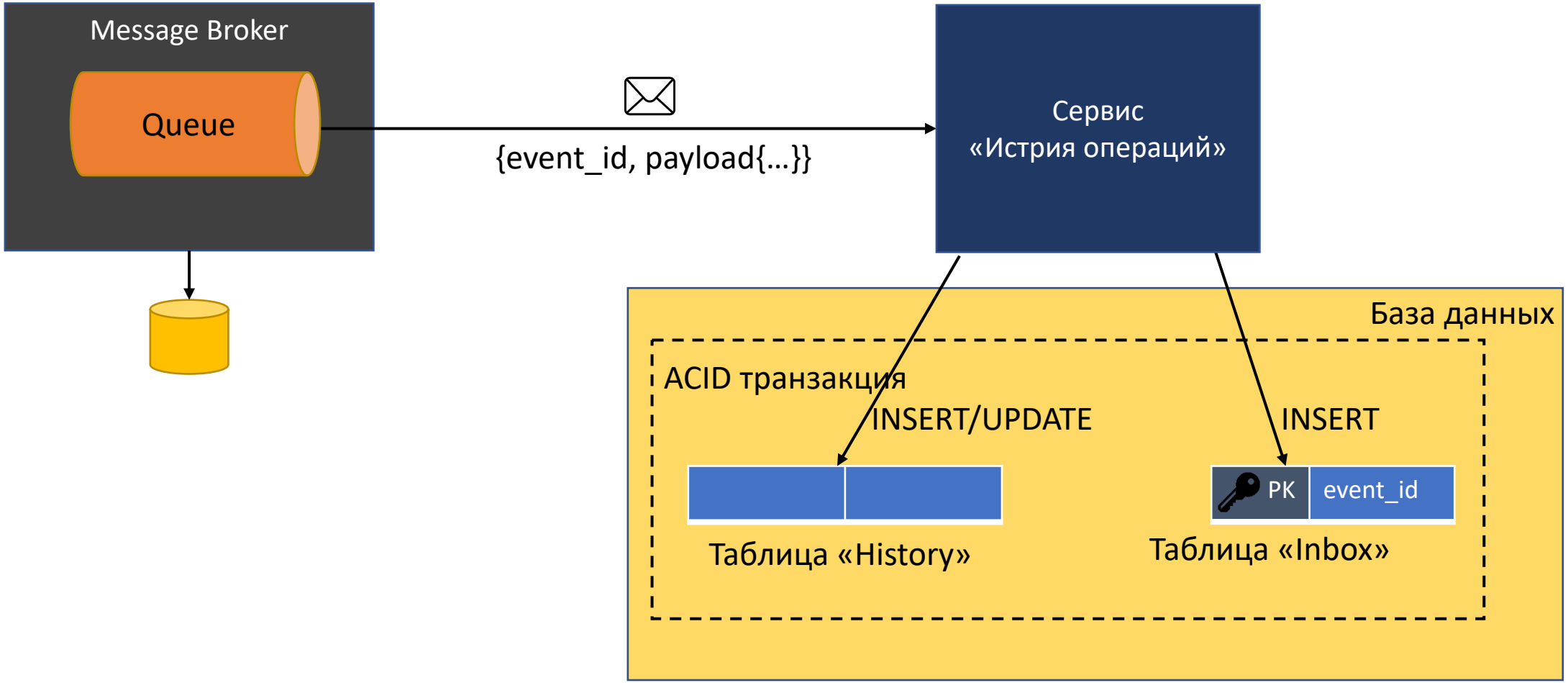
# Популярные решения


https://debezium.io/


debezium + outbox

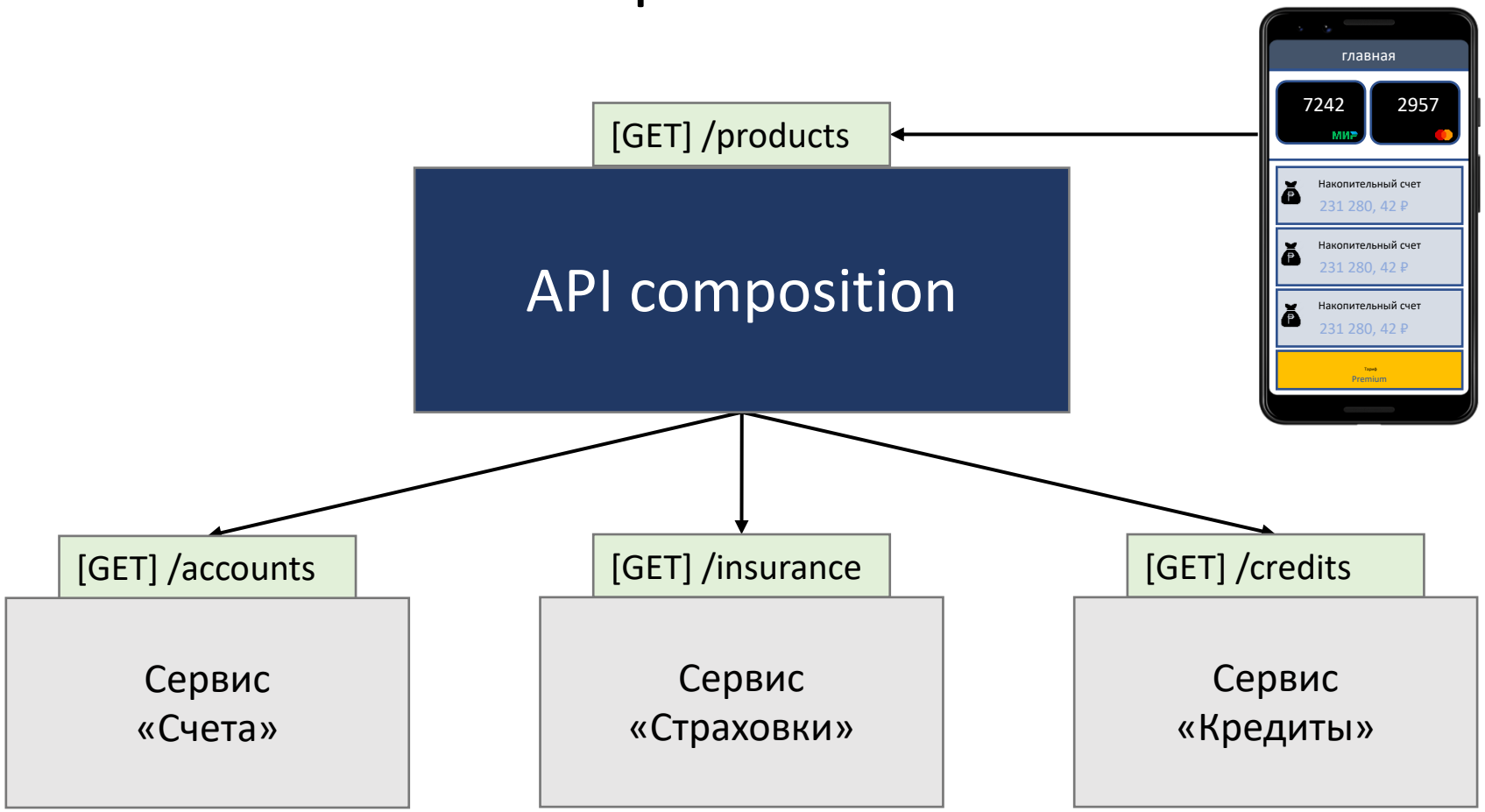
# Idempotent Consumer (inbox) pattern

# Idempotent consumer (inbox) pattern

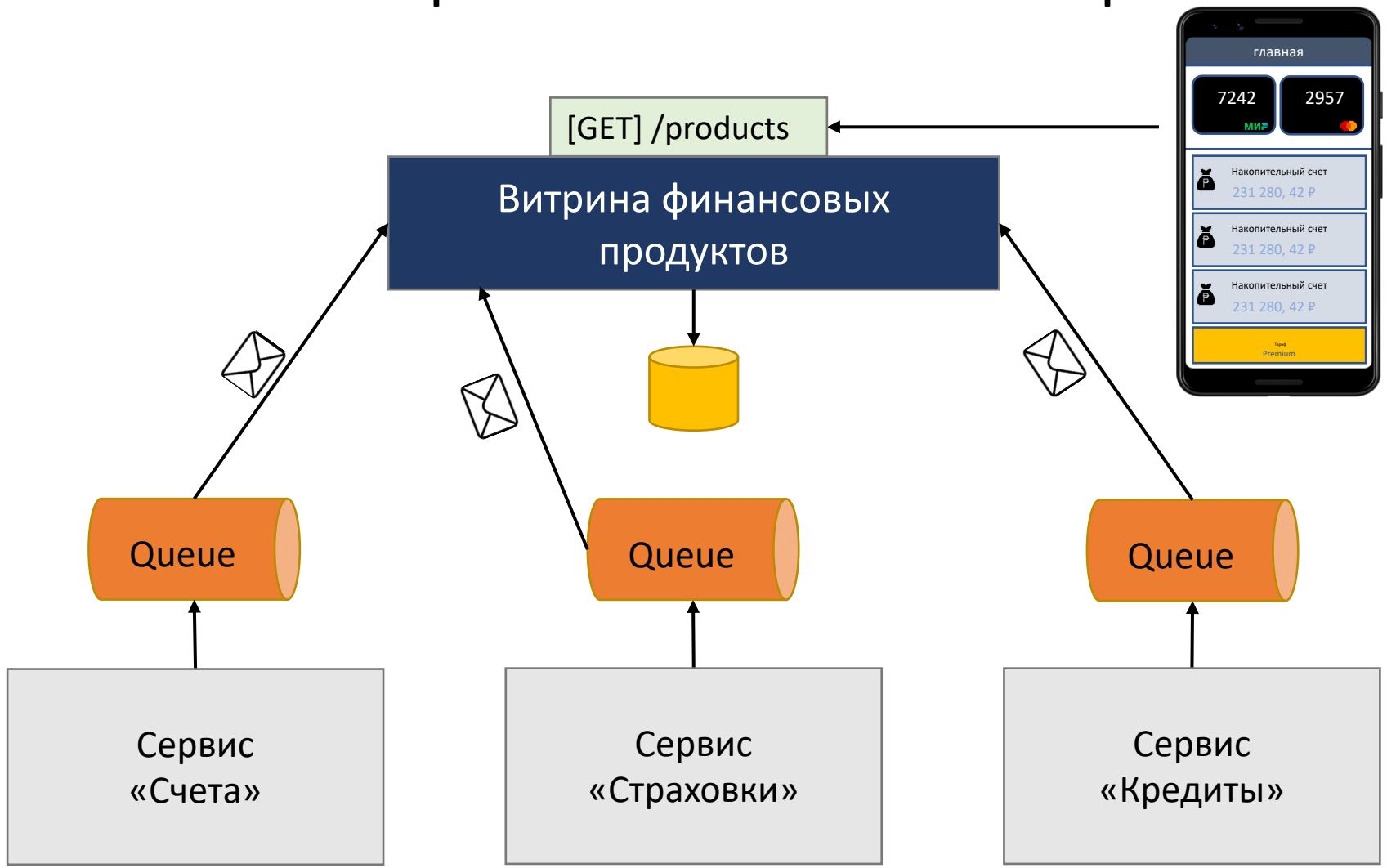


# Command Query Responsibility Segregation (CQRS) pattern

# Вспомним API Composition



# CQRS как альтернатива API Composition



# Плюсы и минусы CQRS

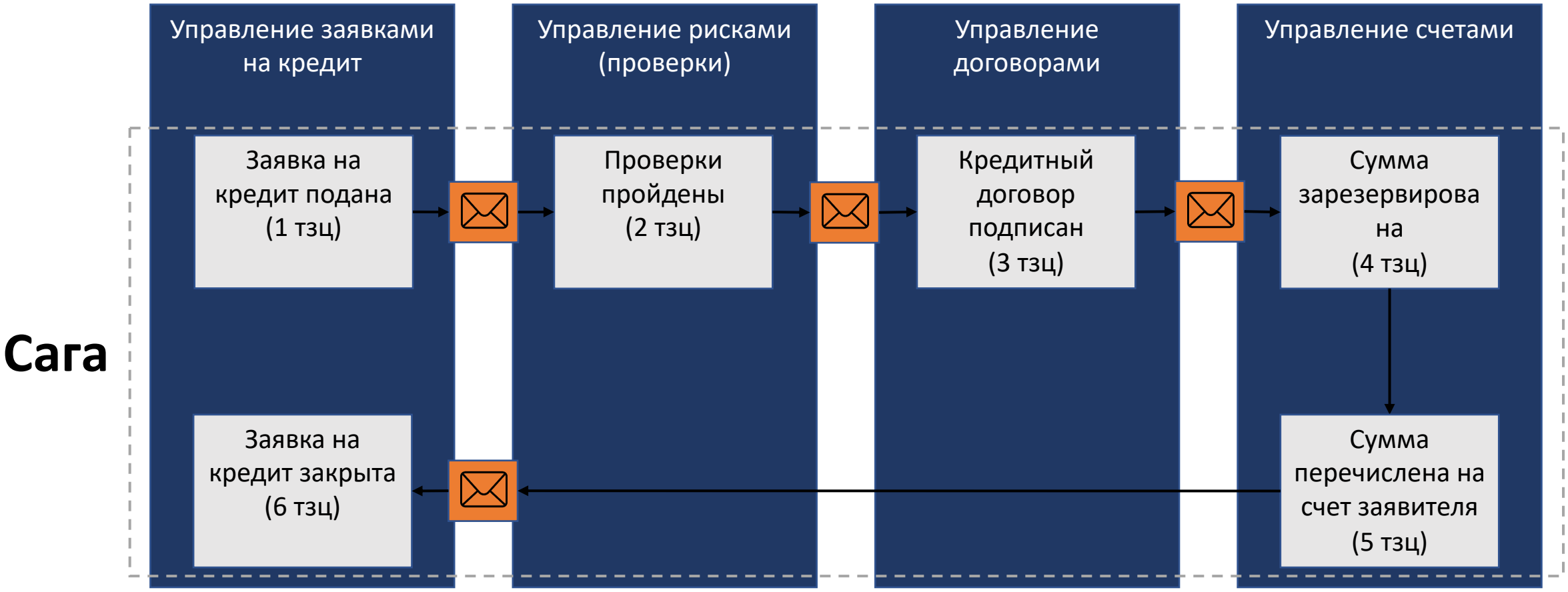
| **Плюсы** | **Минусы** |
| --- | --- |
| • Возможность эффективной реализации запросов<br>• Возможность эффективной реализации сложных/разнородных запросов<br>• Улучшенное разделение ответственности | • Более сложная архитектура<br>• Отставание репликации |

# Управление сквозными процессами

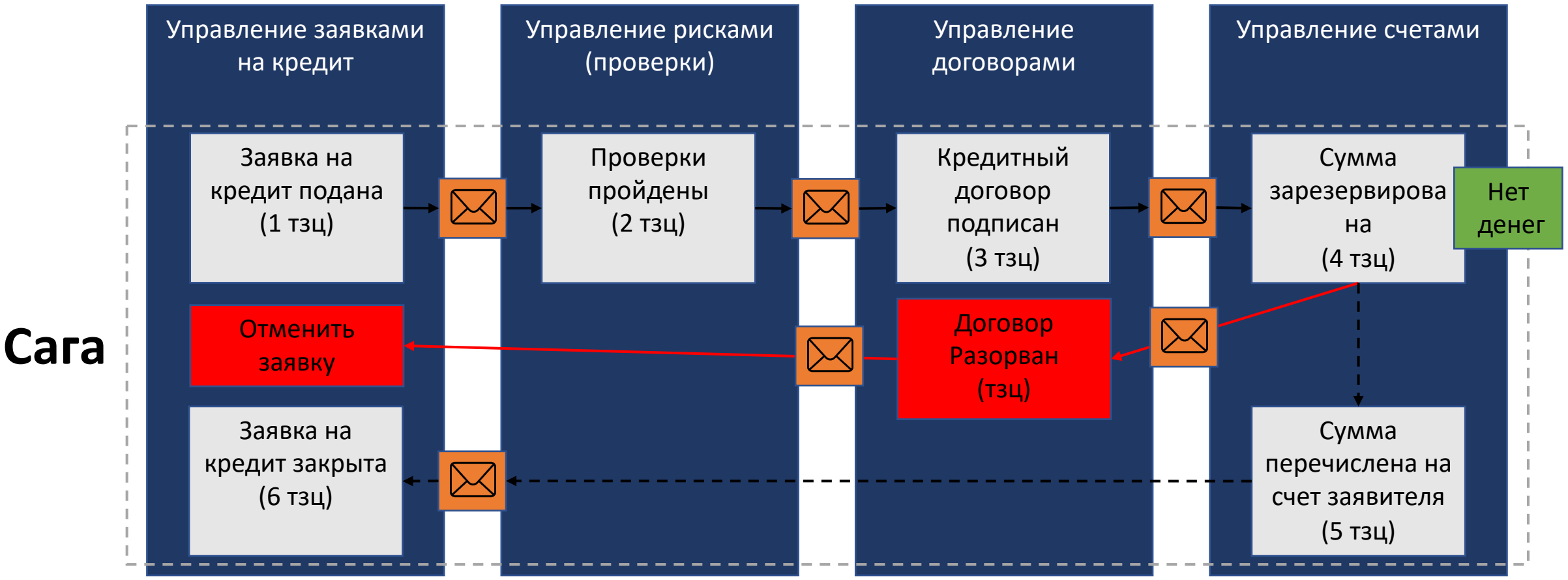
SKLADCHIK.ORG

# Saga pattern

# Пример


Сага
Управление заявками на кредит
Управление рисками (проверки)
Управление договорами
Управление счетами
Заявка на кредит подана (1 тзц)
Проверки пройдены (2 тзц)
Кредитный договор подписан (3 тзц)
Сумма зарезервирована (4 тзц)
Сумма перечислена на счет заявителя (5 тзц)
Заявка на кредит закрыта (6 тзц)

# Компенсирующие транзакции

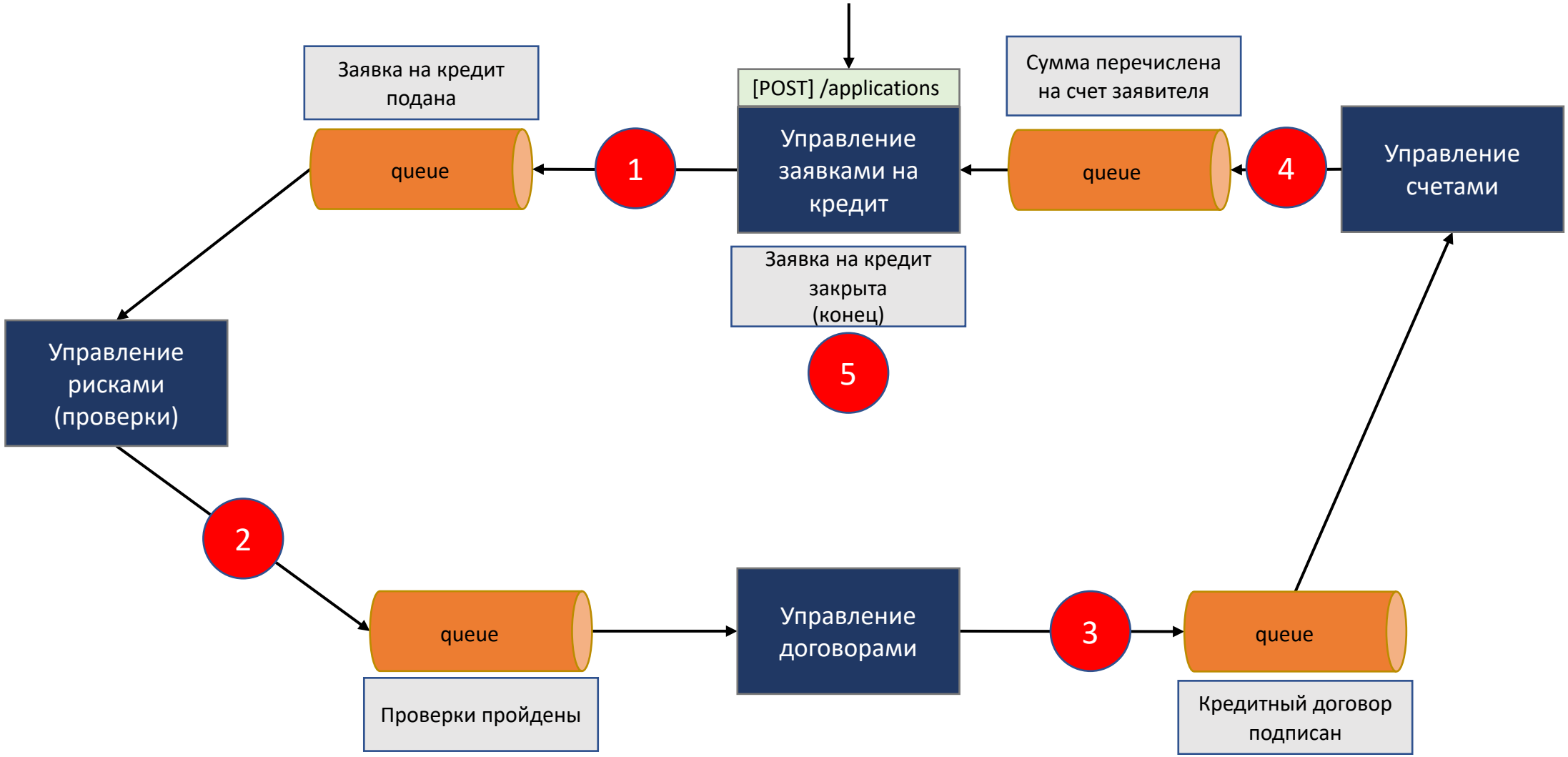


# Координация Saga

Хореография

Оркестрация

# Хореография



# Плюсы и минусы хореографии

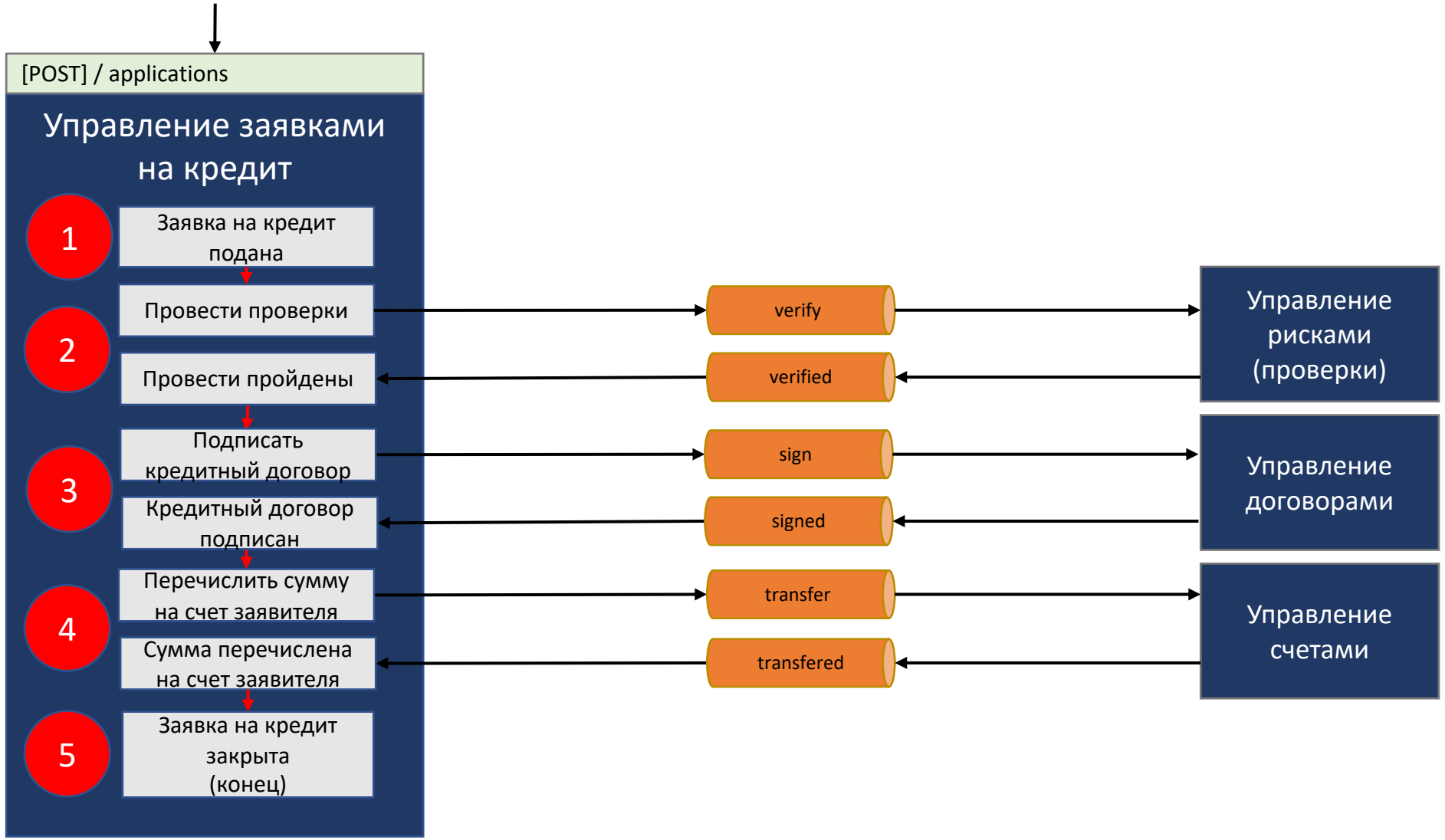
**Плюсы**

- Простота
- Слабая связанность

**Минусы**

- Сложнее диагностика проблем
- Риск циклических зависимостей между сервисами

# Оркестрация



# Плюсы и минусы оркестрации

**Плюсы**

- Не создает циклических зависимостей
- Улучшенное разделение ответственности, вся координирующая логика находится в оркестраторе

**Минусы**

- Риск избыточной централизации бизнес-логики в оркестраторе

# О чём узнали

- Синхронное взаимодействие допустимо, но содержит ряд недостатков. Лучше использовать асинхронное взаимодействие
- Временная несогласованность данных в микросервисах зачастую не является проблемой, но позволяет строить более стабильные и масштабируемые решения
- Каналы коммуникации ненадежны. Мы должны использовать подходы, защищающие от потери данных (outbox)

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

SKLADCHIK.ORG

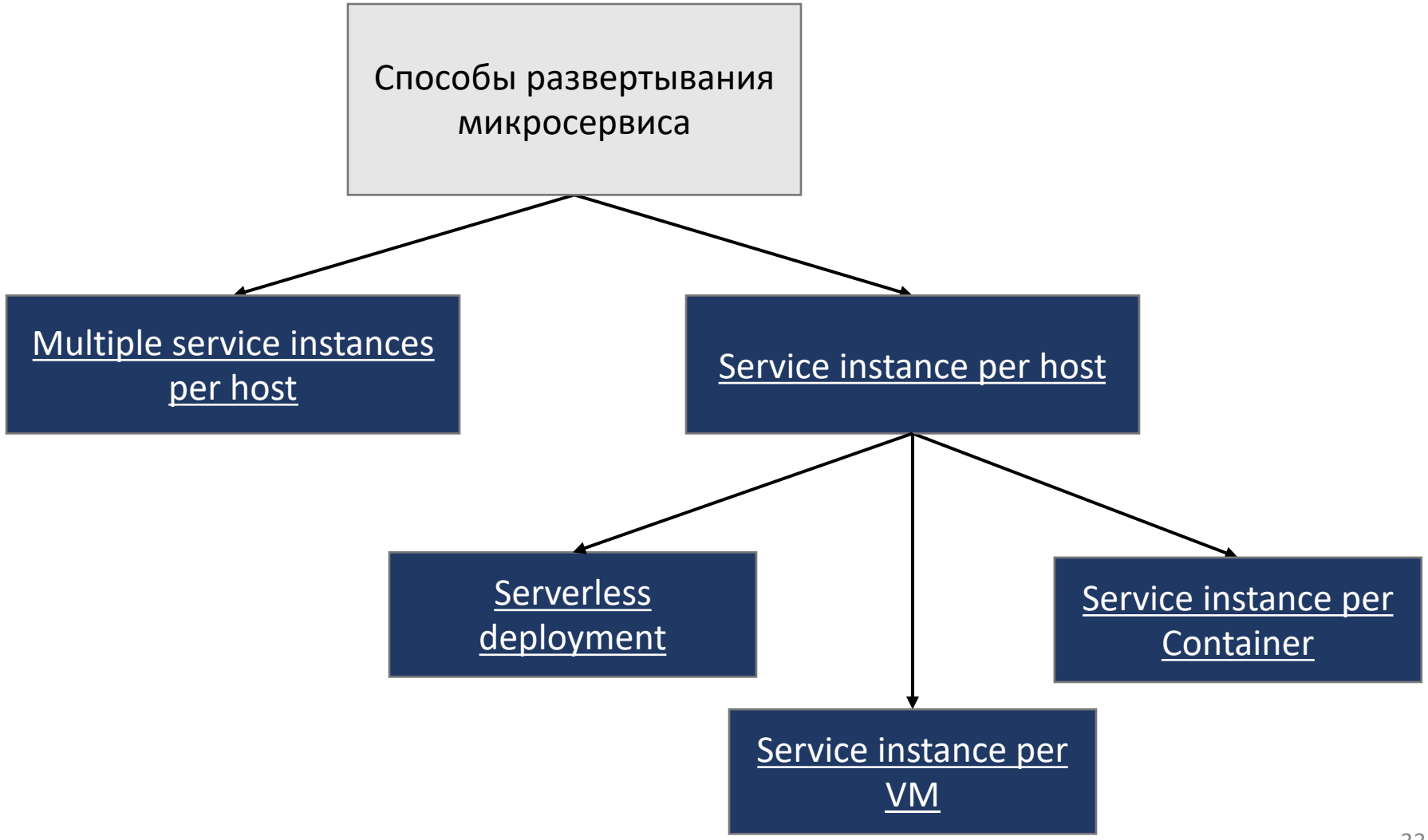
# Развертывание

# Цели занятия

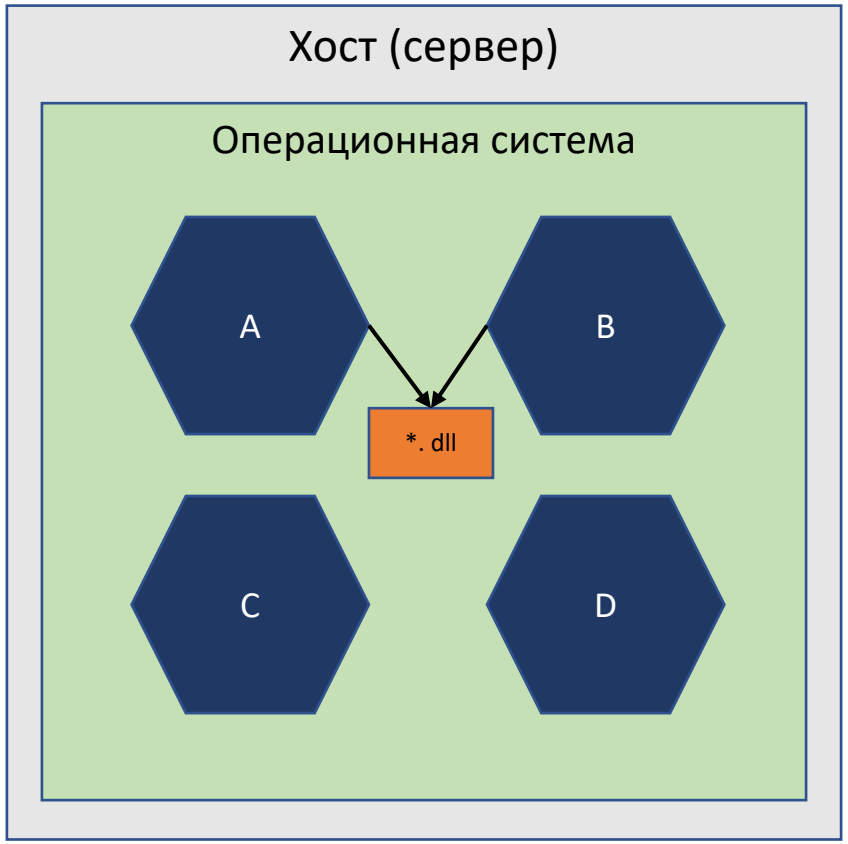
- Узнать, какие способы развёртывания микросервисов бывают
- Разобраться, как организовать GitLab репозитории
- Рассмотреть разные подходы получения конфигурации сервисом

# Способы развертывания микросервиса

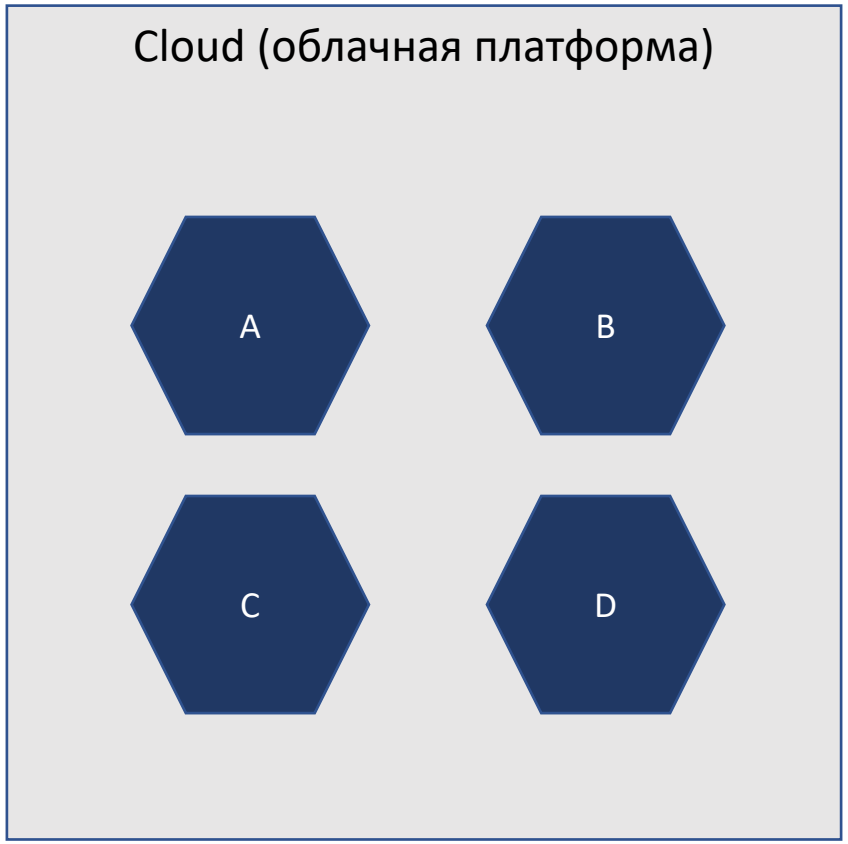
# Способы развертывания микросервиса



# Multiple service instances per host

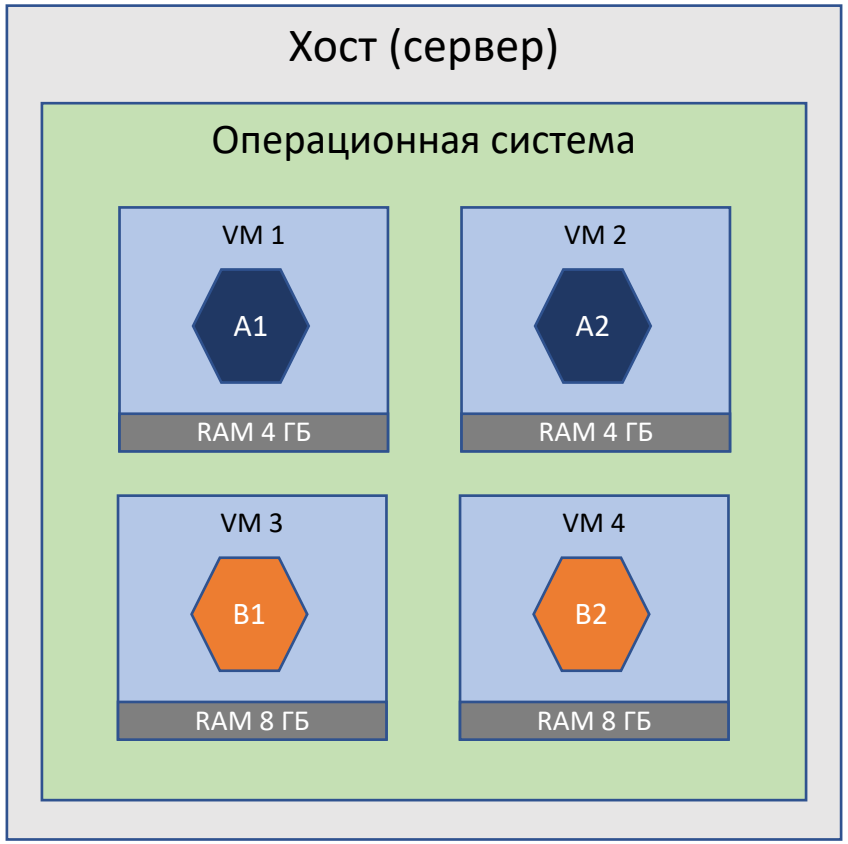


# Serverless deployment

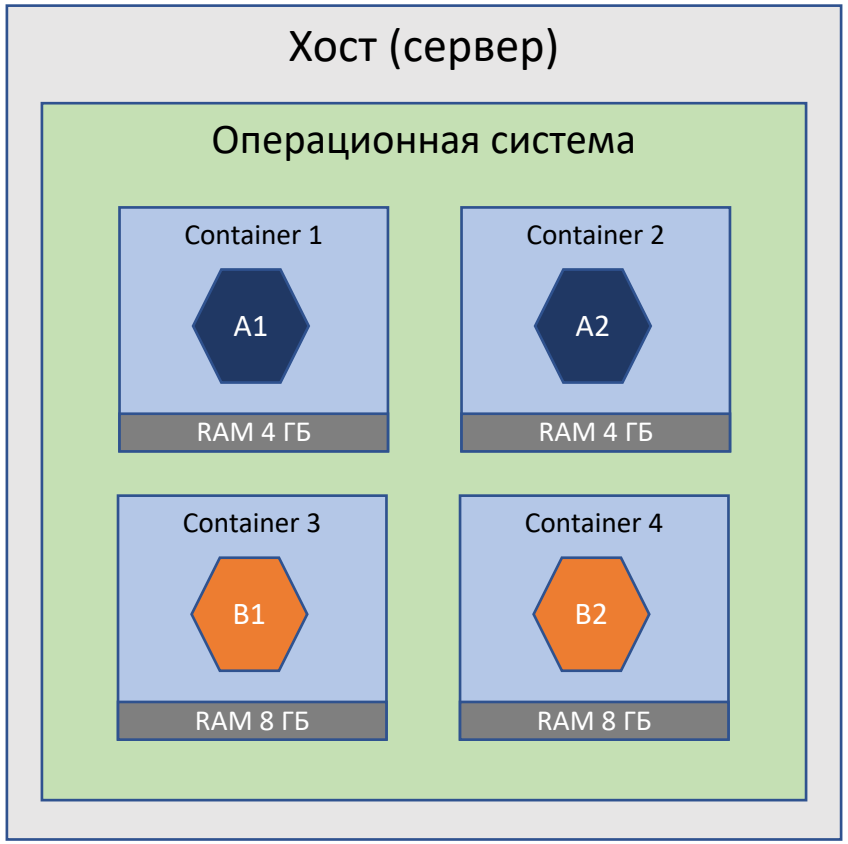



**Serverless** — бессерверные вычисления, которые помогают разрабатывать приложения, хранить данные и настраивать интеграцию с другими платформами без создания виртуальных машин и обслуживания инфраструктуры.

# Service instance per VM



# Service instance per Container



# VM or Container

**VM**

- Долго поднимается
- Инфраструктура настраивается
- Медленно «переезжает»

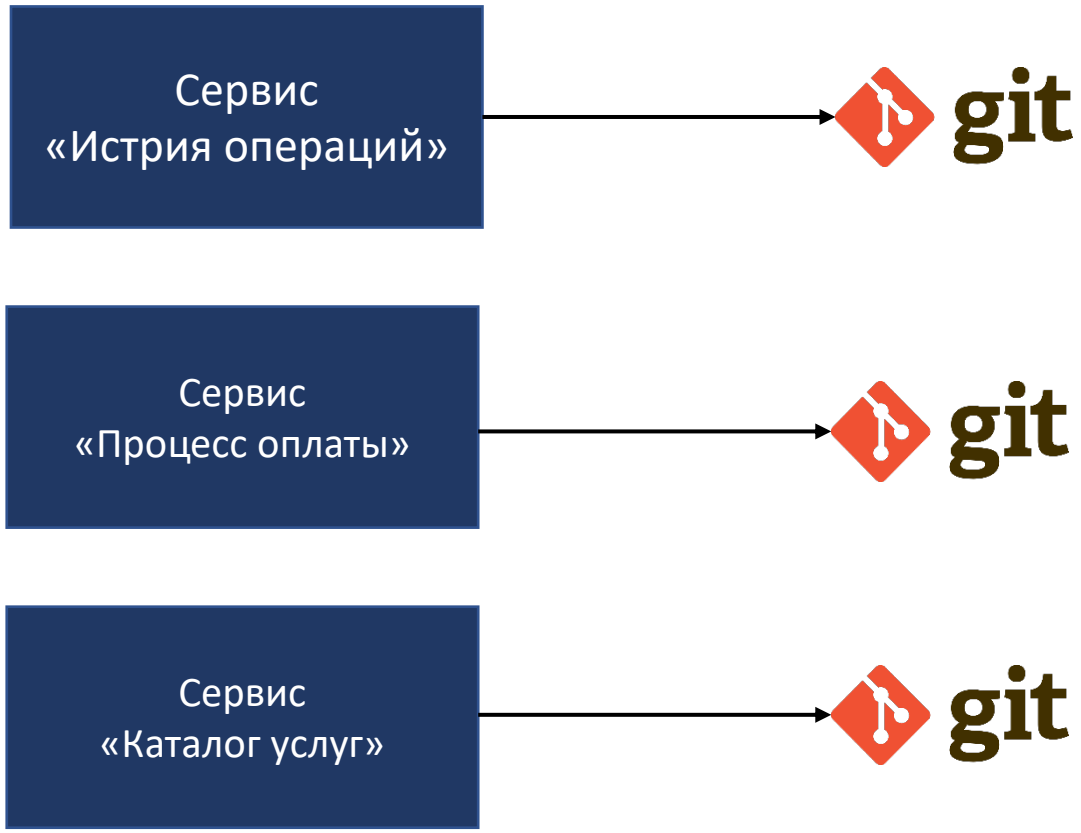
**Container**

- Быстро поднимается
- Инфраструктура внутри
- Легко «переезжает»

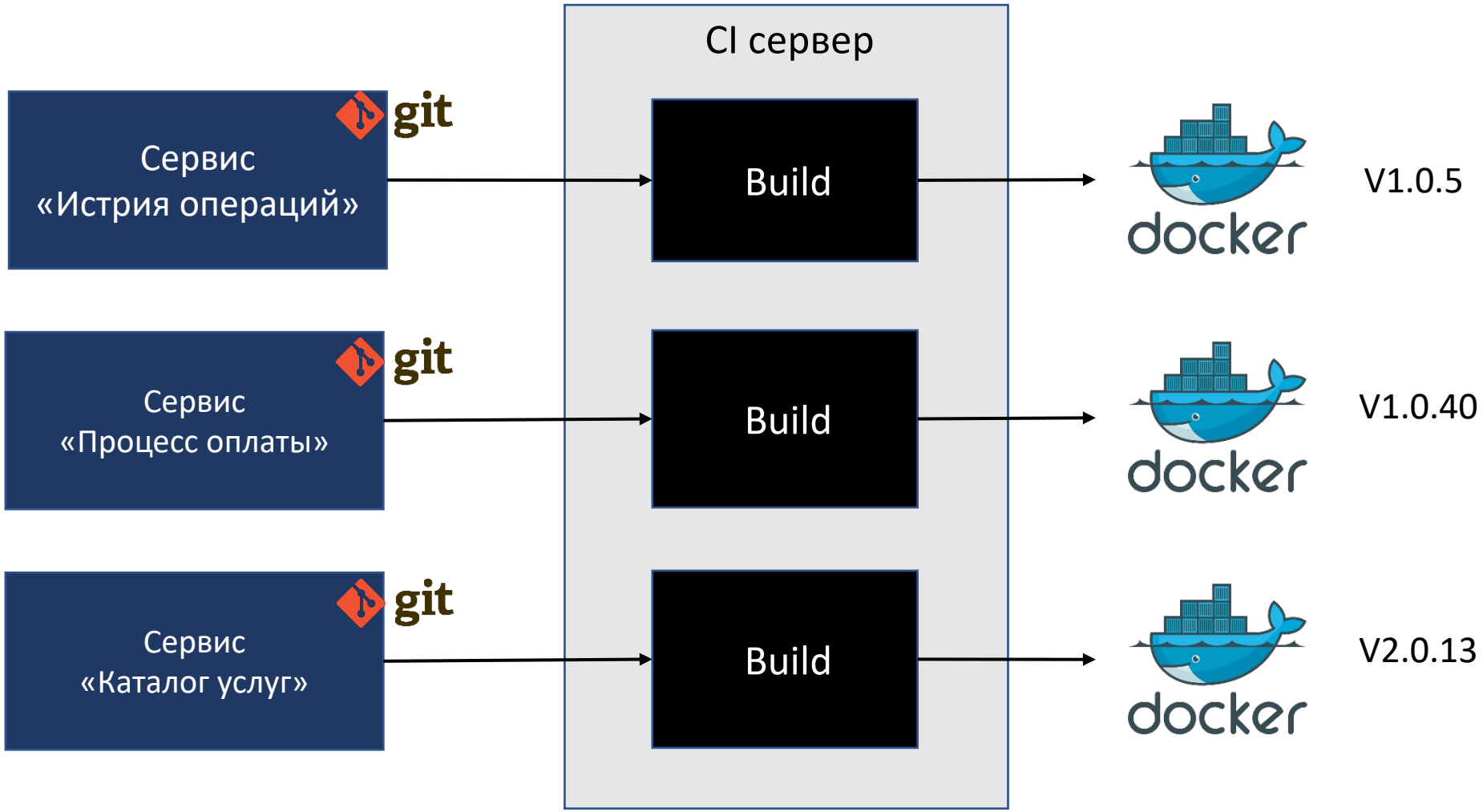
# Стратегия автономности поставки

SKLADCHIK.ORG

# Один микросервис - один репозиторий

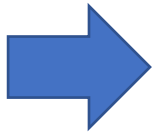


# Один микросервис - один CI конвейер



# Типовой CI/CD конвейер сервиса

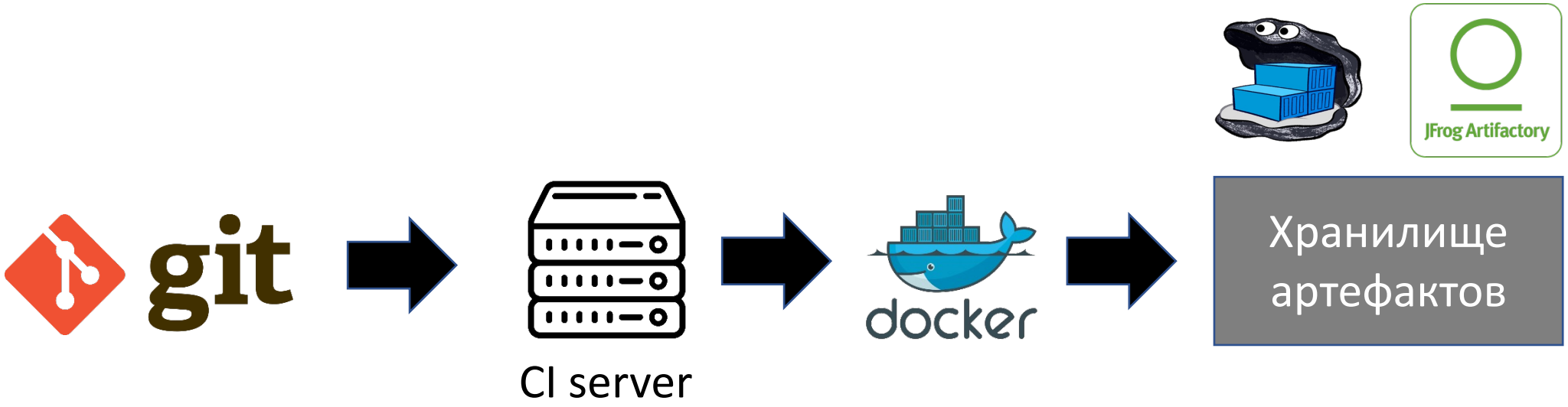
# Continuous Integration

CI server

- Статический анализ кода
- Unit тесты
- Сборка артефакта
- Интеграционные тесты
- Компонентные тесты
- Отправка в хранилище
- Запуск триггеров

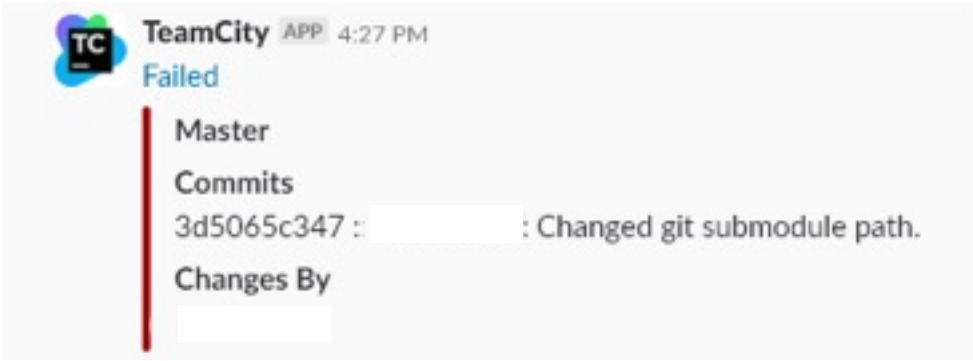
# Хранилище артефактов



# Быстрая обратная связь

## Офлайн


## Онлайн

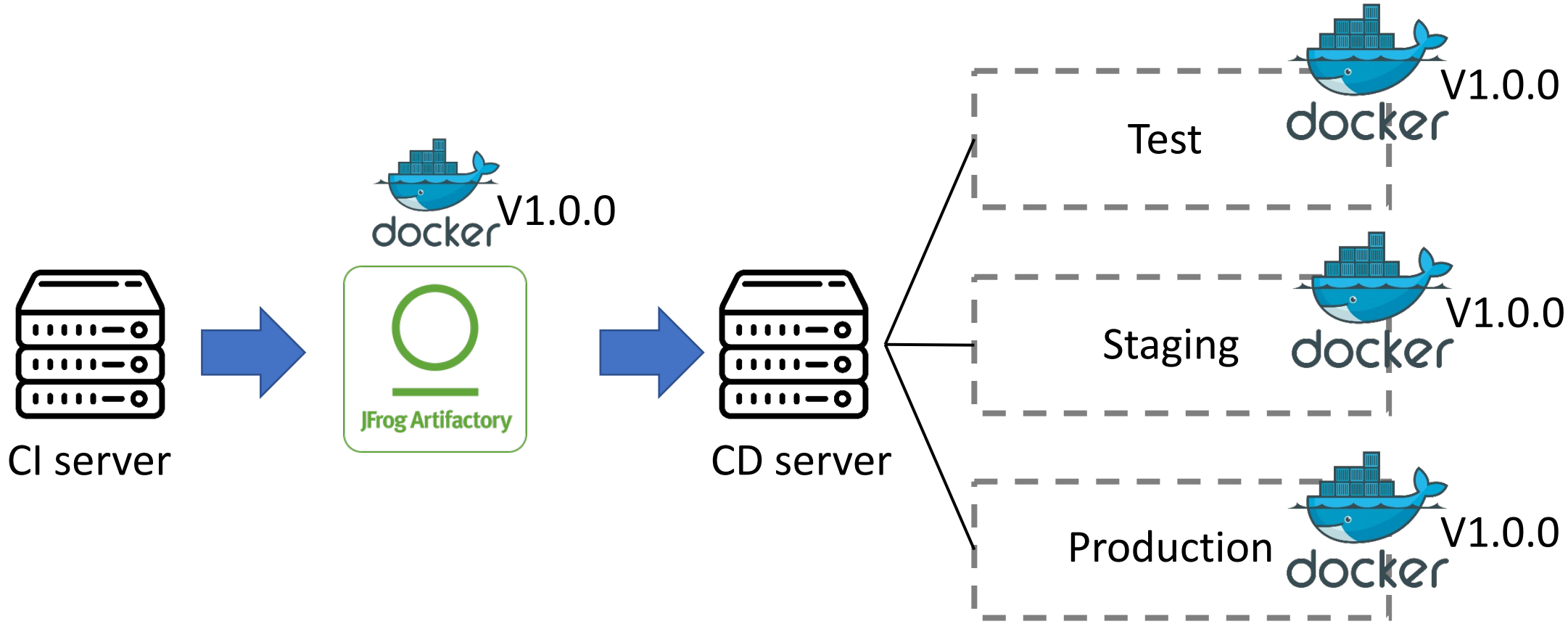



slack

# Continuous Delivery

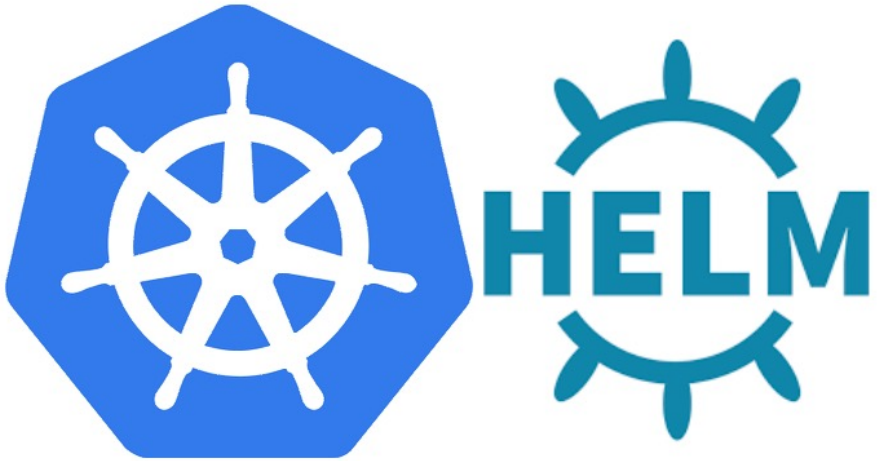
- Автоматизировать развертывание артефактов
- Единый конвейер развертывания для любой среды
- Минимизировать человеческий фактор

# Выпуск новой версии микросервиса



# CD сервиса - шаги

# Deploy - инструменты

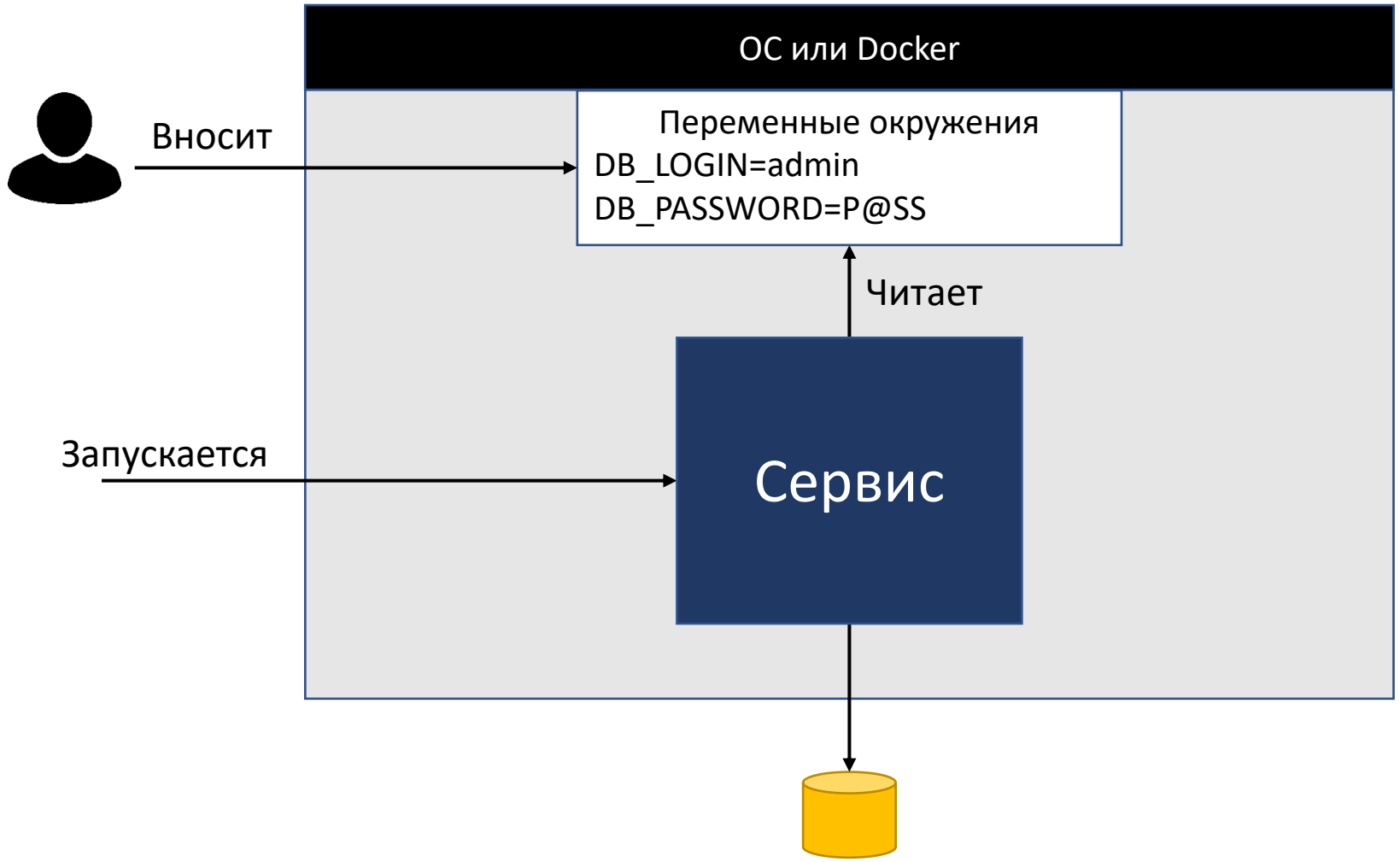



SKLADCHIK.ORG

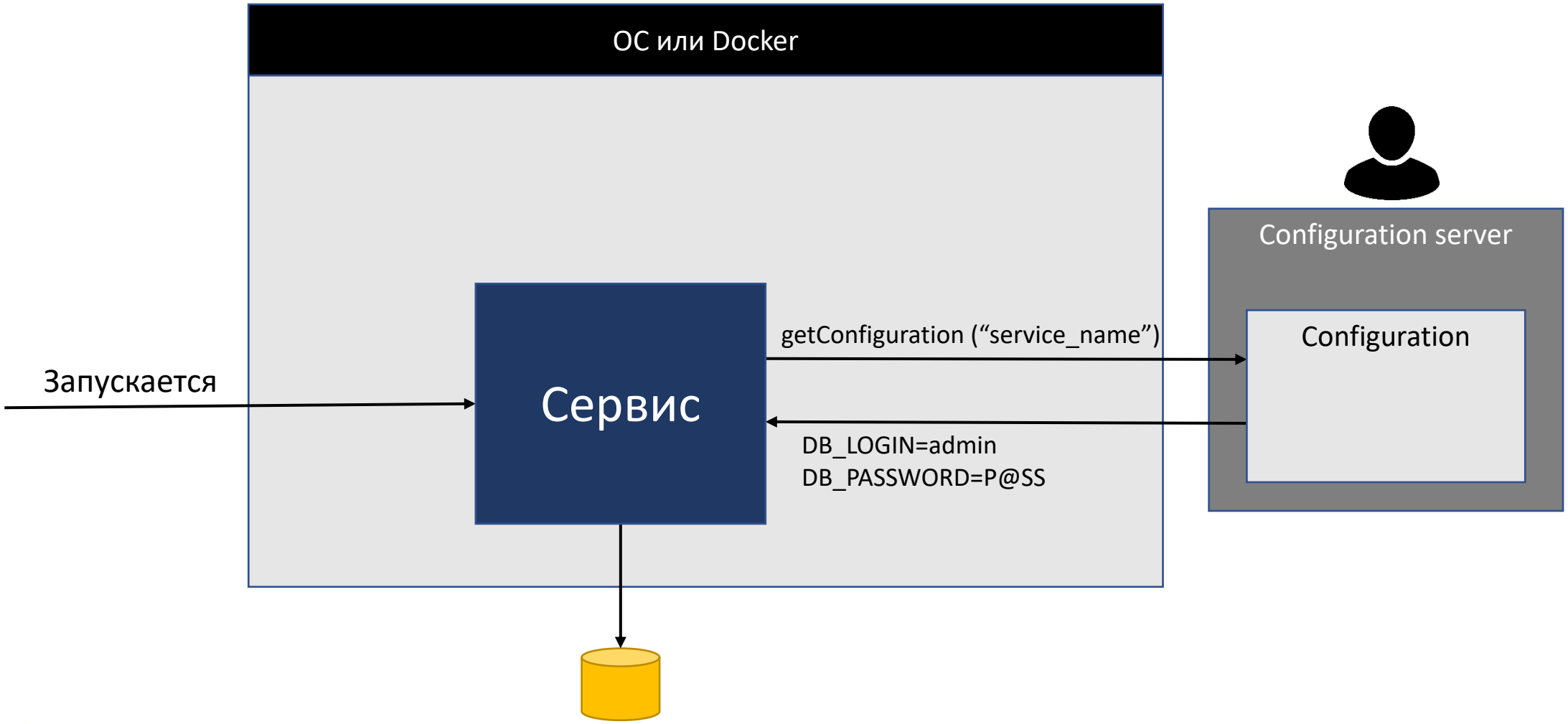
# Способы получения конфигурации

# Externalized configuration pattern

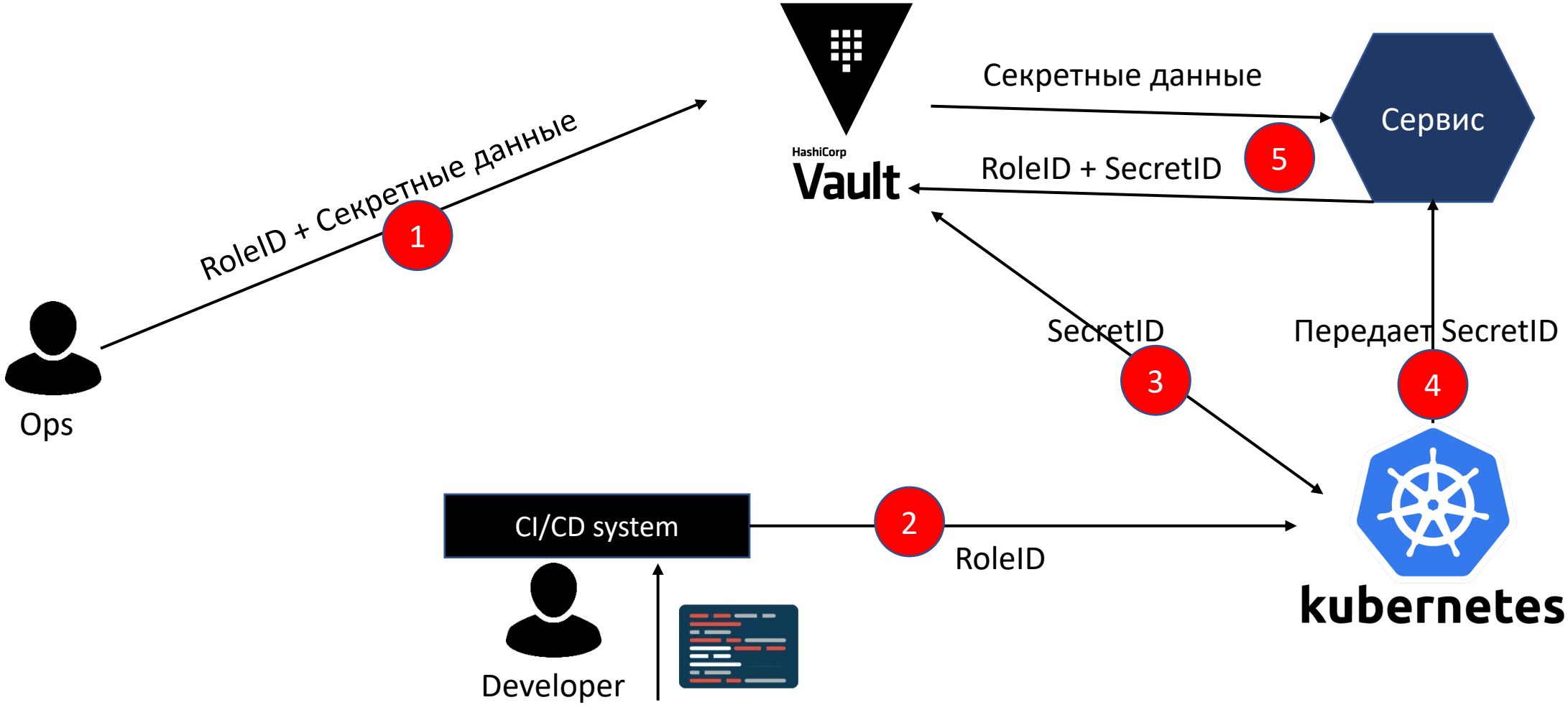
# Пассивная модель



# Активная модель



# Хранение/получение секретных данных (Vault)



# О чём узнали

- При развёртывании микросервисов важно добиться того, чтобы сервисы обладали автономностью
- Пересобираем и деплоим только тот микросервис, в котором были изменения. Зависимые деплои сервисов считаются плохой практикой

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

# Тестирование

# Цели занятия

- Разобраться в особенностях тестирования распределенных систем
- Познакомиться с основными видами тестов
- Научиться формировать стратегию тестирования микросервисной системы

# Как обычно происходит тестирование системы?

SKLADCHIK.ORG

# Выделенный специалист или команда

QA отдел

End-to-end тесты

API

Система

# И все вроде бы ничего, но..
# у нас распределенная система

# End-to-end

QA отдел

End-to-end тесты

API

Система

Сервис 1

Сервис 2

Сервис 3

4

8

9

5

7

6

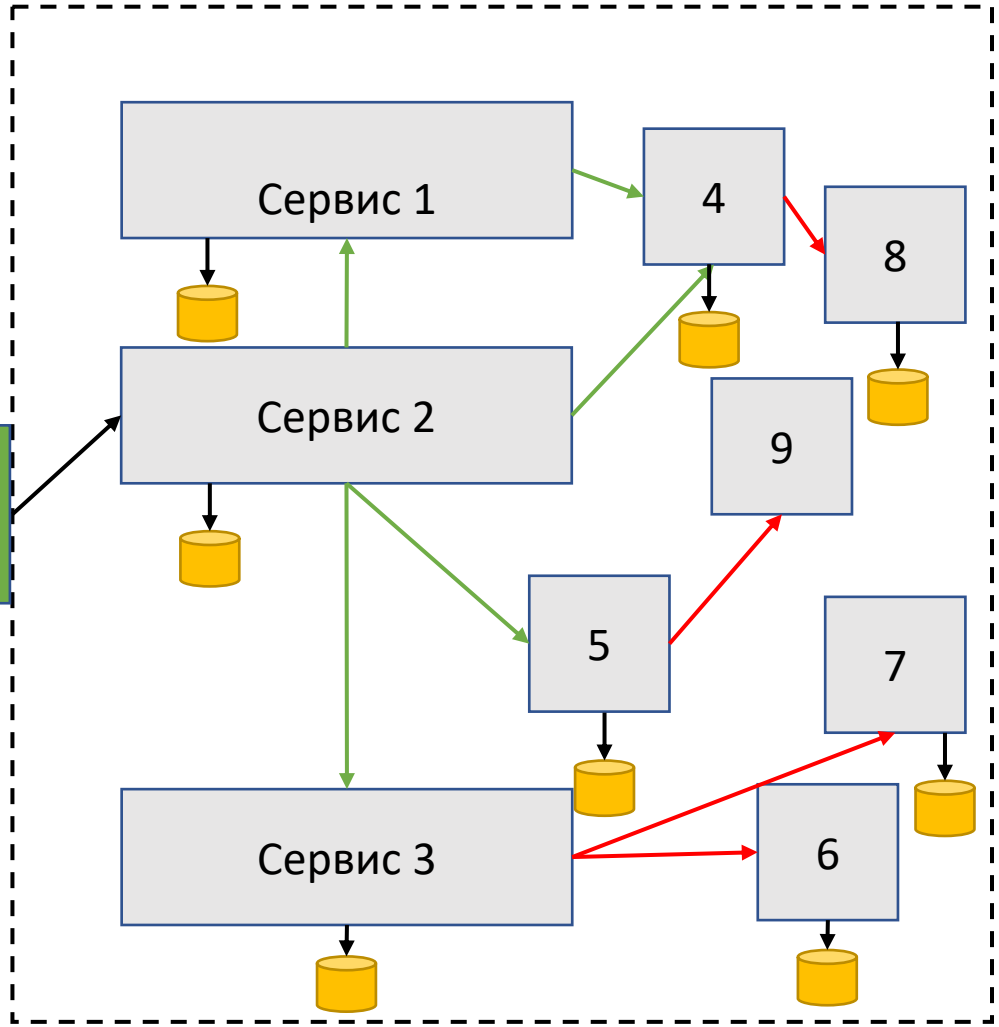
# End-to-end

QA отдел

End-to-end тесты

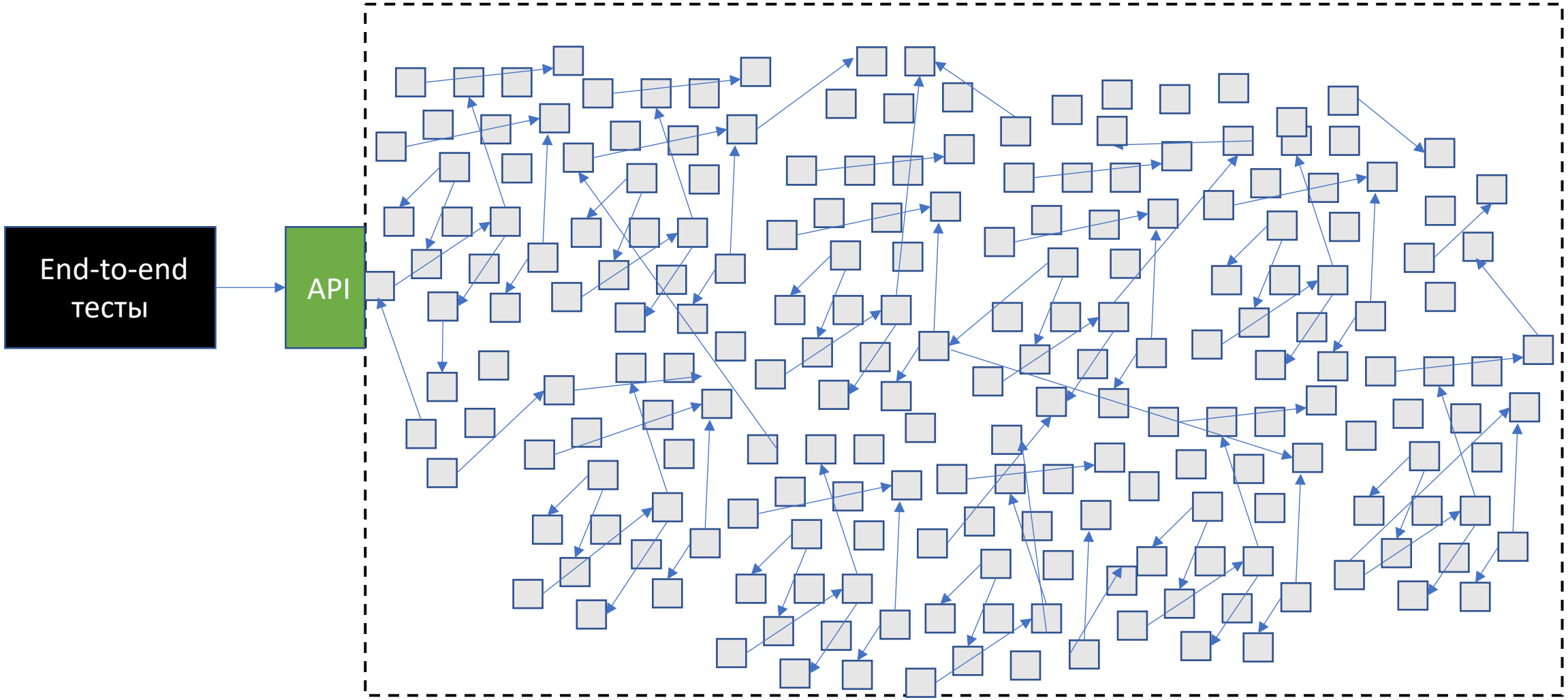
API

Система

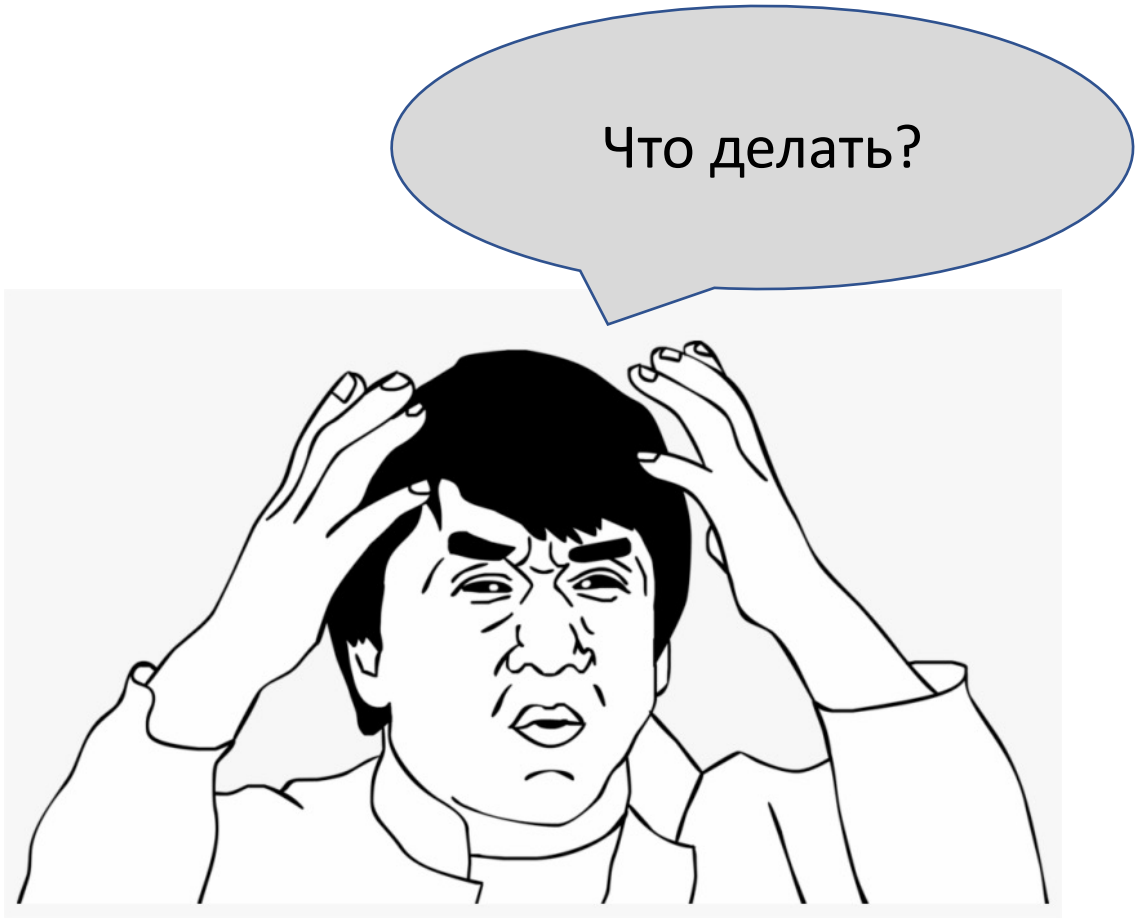


И все вроде бы ничего, но..
в реальности оно выглядит так

# Пример реальной системы



# Писать End-to-end тесты становится очень сложно



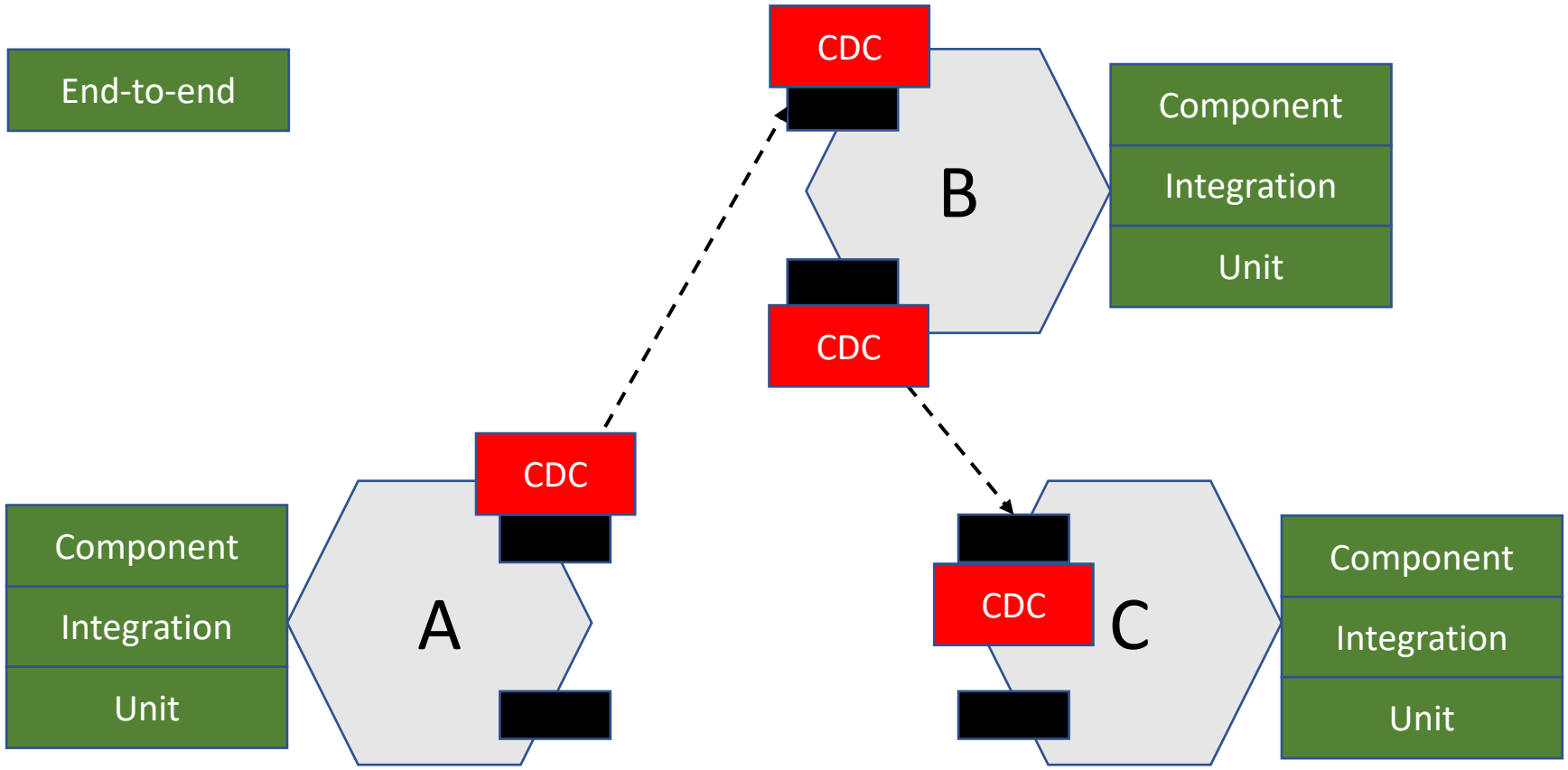
# Решение в лоб

Нам нужно нанять  больше тестировщиков чтобы писать End-to-end тесты!

QA отдел

# Альтернативное решение

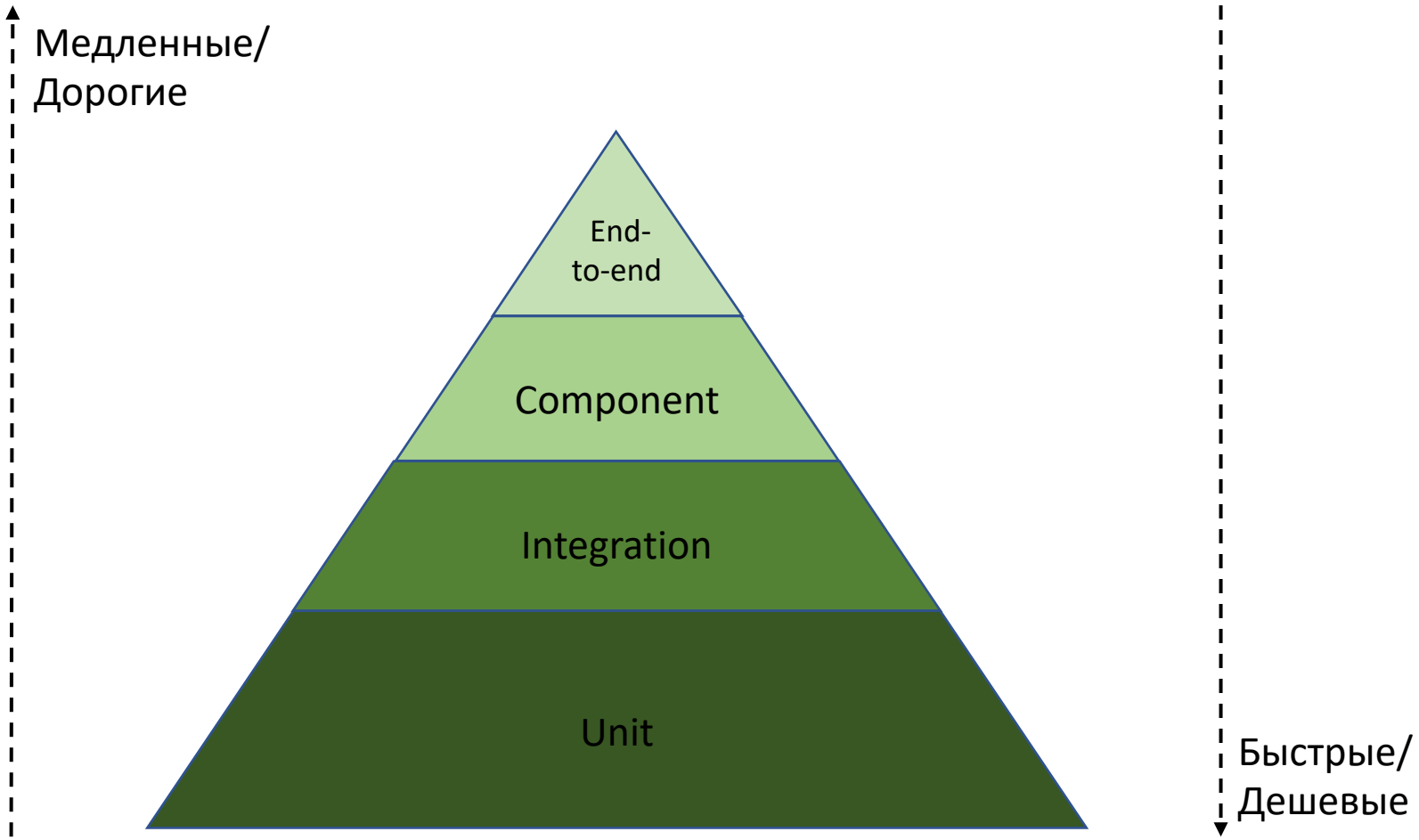
# Тестирование в распределенной системе



SKLADCHIK.ORG

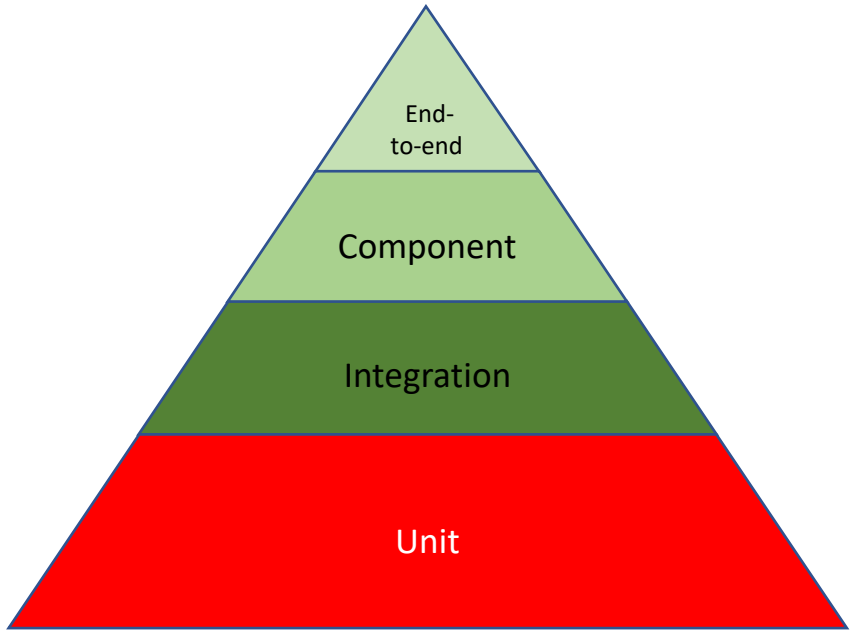
# Пирамида тестирования

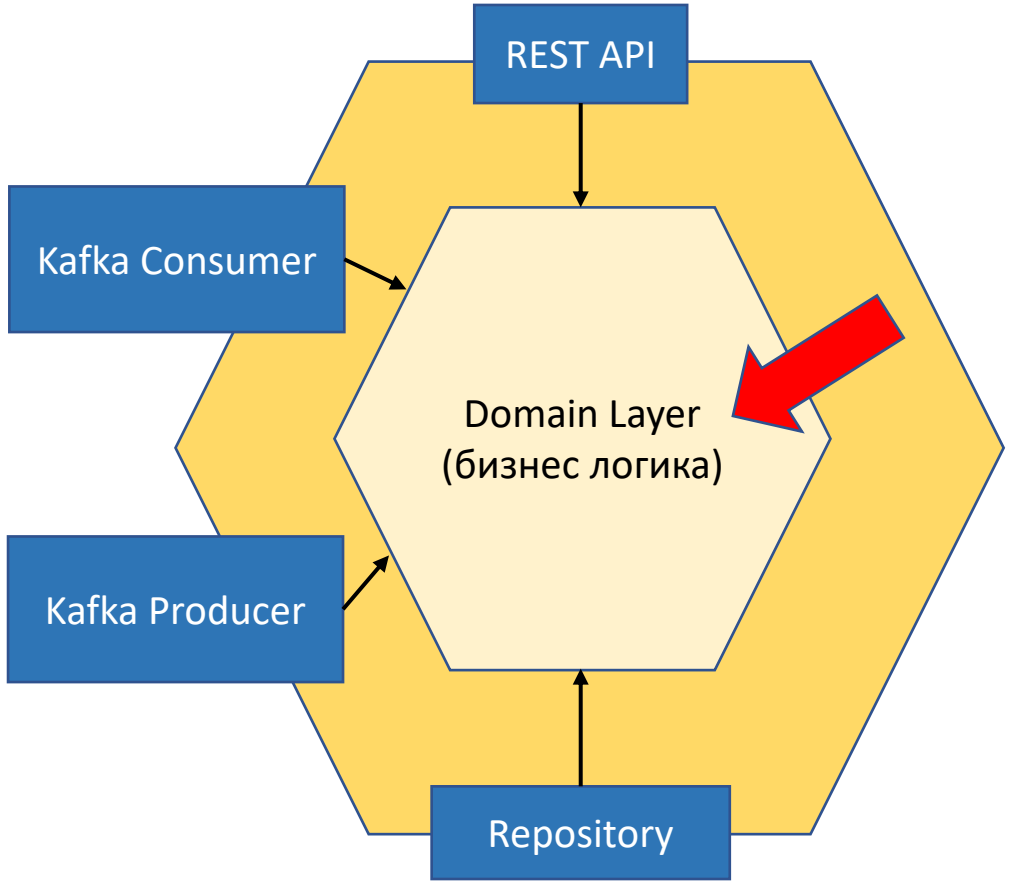
# Пирамида тестирования



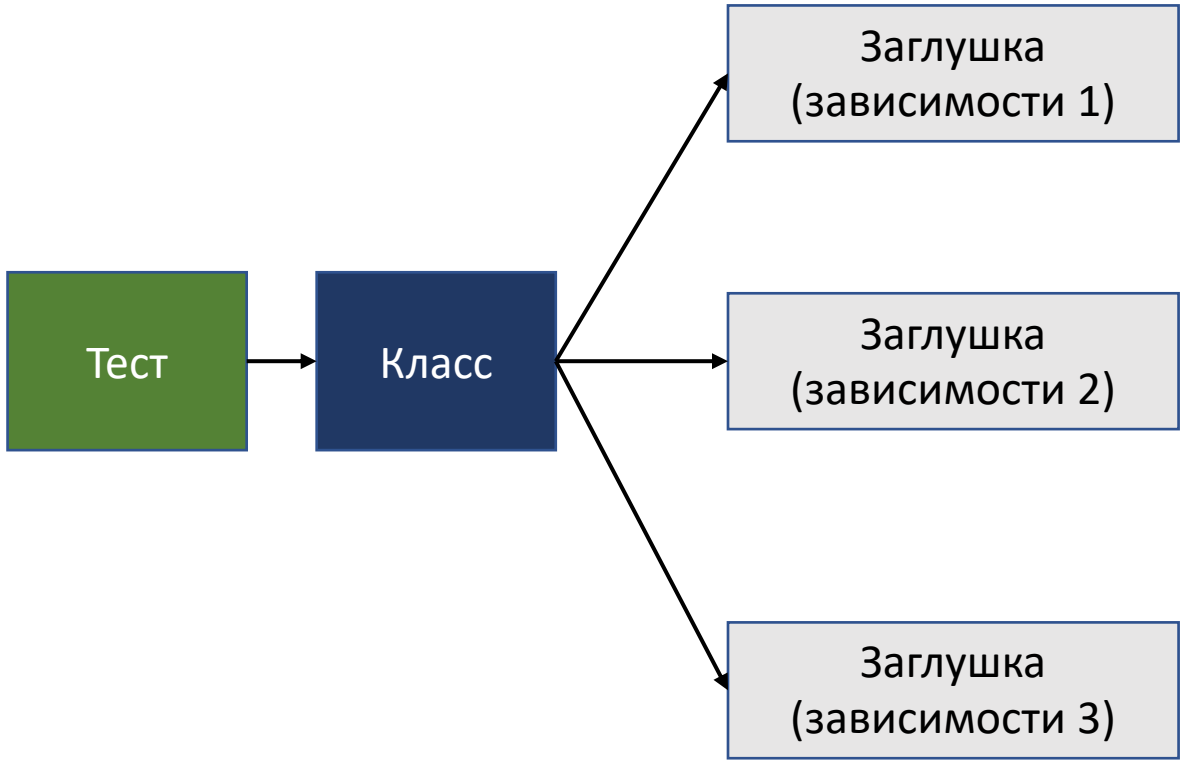
# Unit тесты

# Unit тесты


End-
to-end
Component
Integration
Unit


REST API
Kafka Consumer
Domain Layer
(бизнес логика)
Kafka Producer
Repository

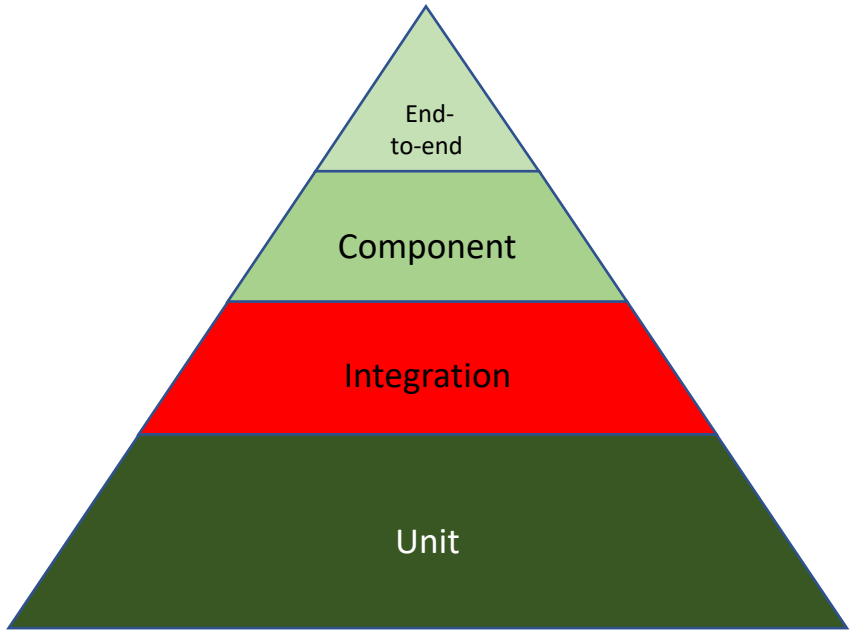
# Unit тесты

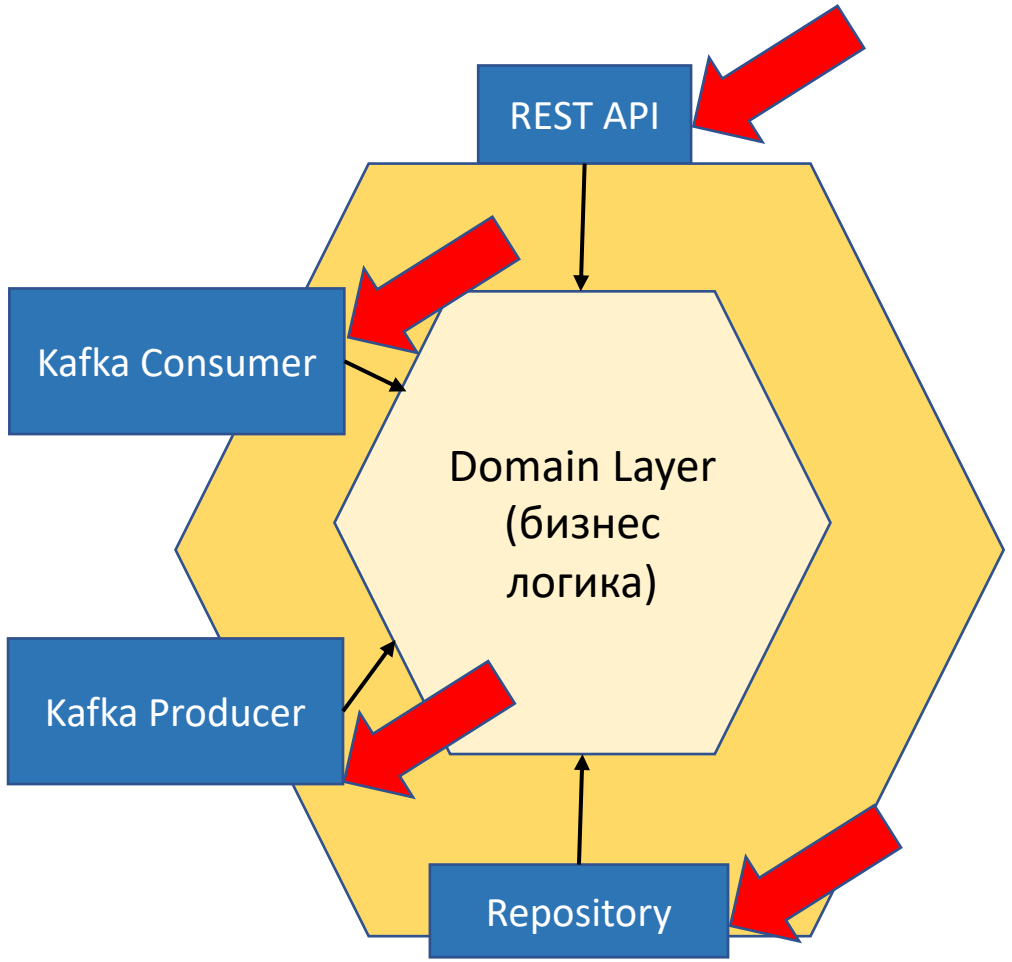

Тест
Класс
Заглушка
(зависимости 1)
Заглушка
(зависимости 2)
Заглушка
(зависимости 3)

# Место в CI

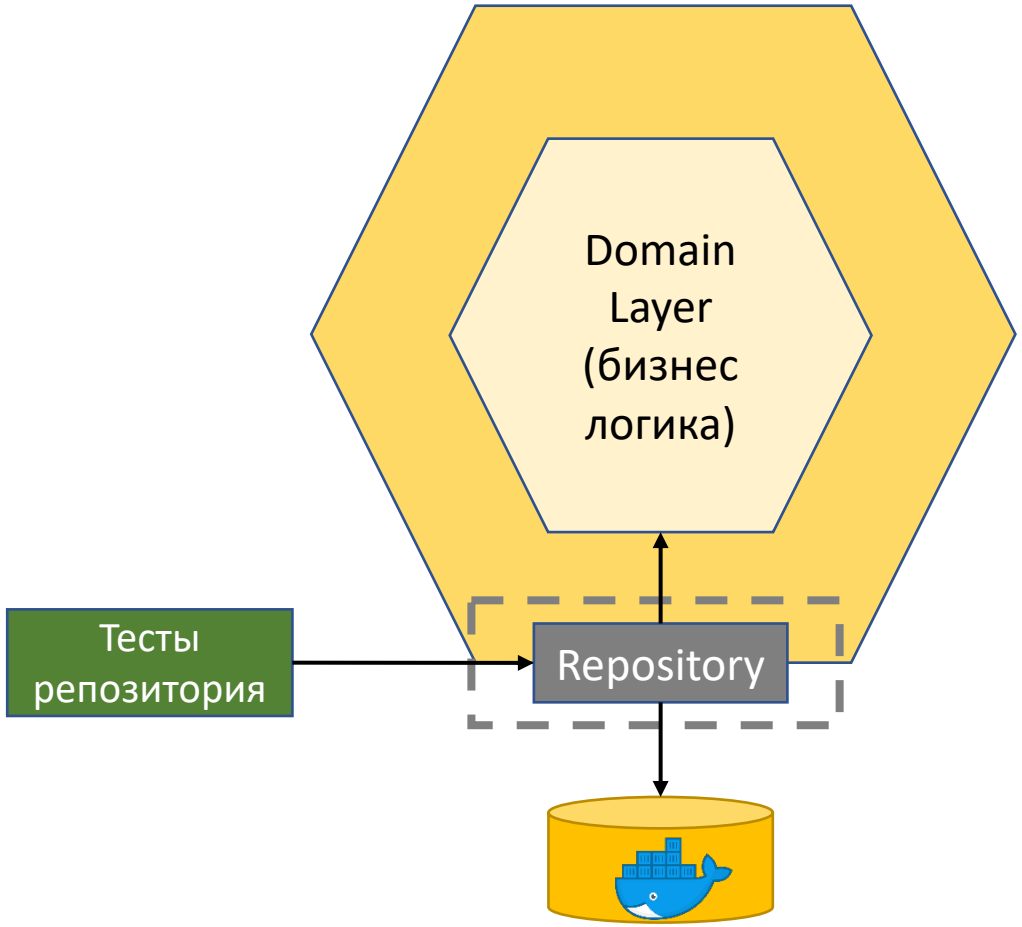
# Integration тесты

# Интеграционные тесты



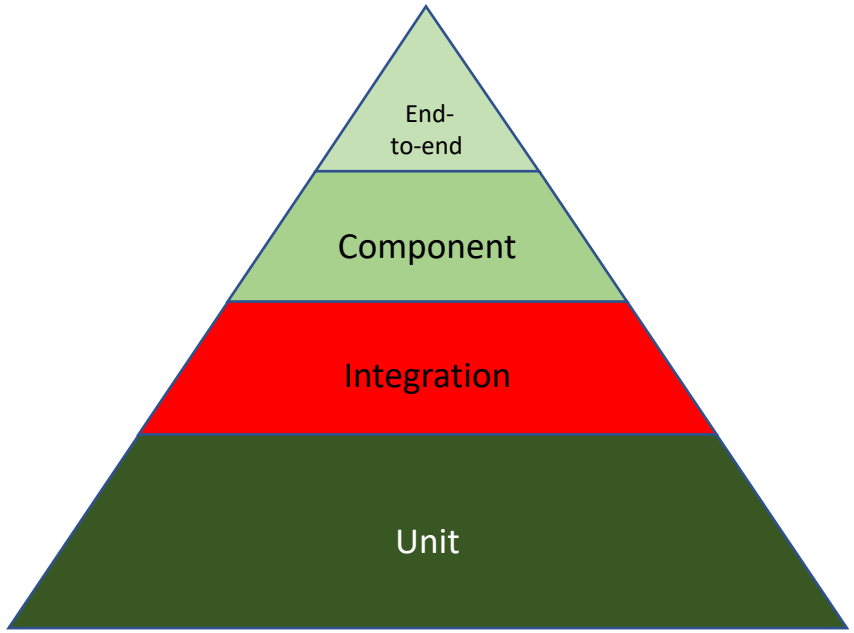
# Тестирование репозитиория

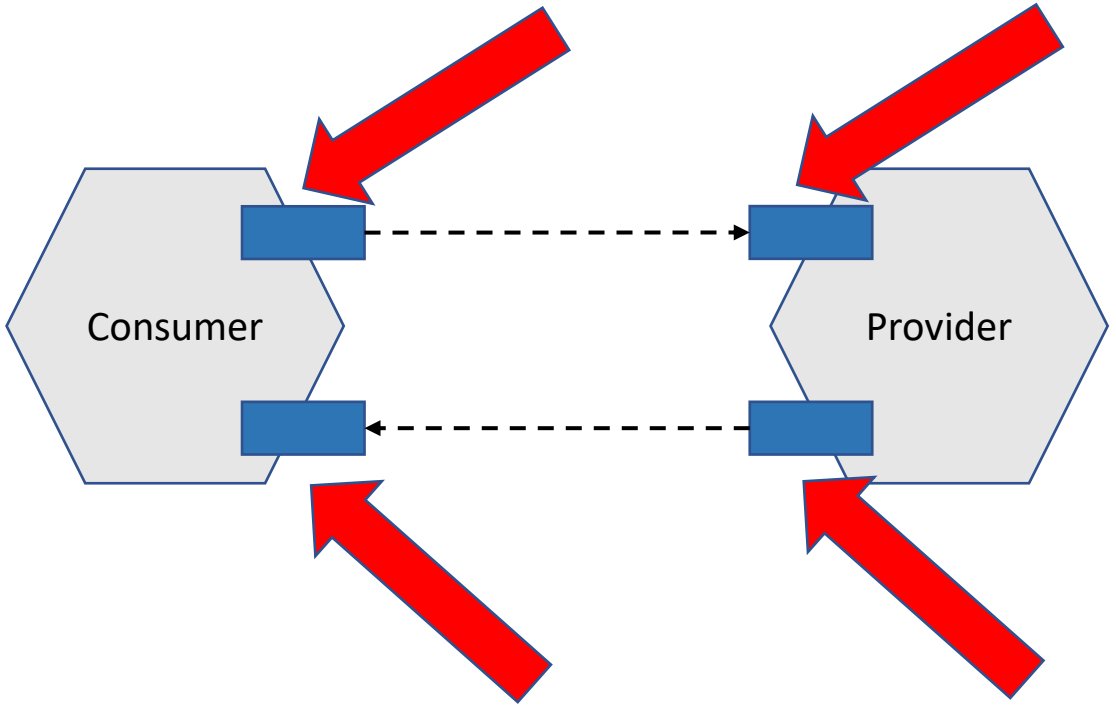


# Consumer-side contract test pattern

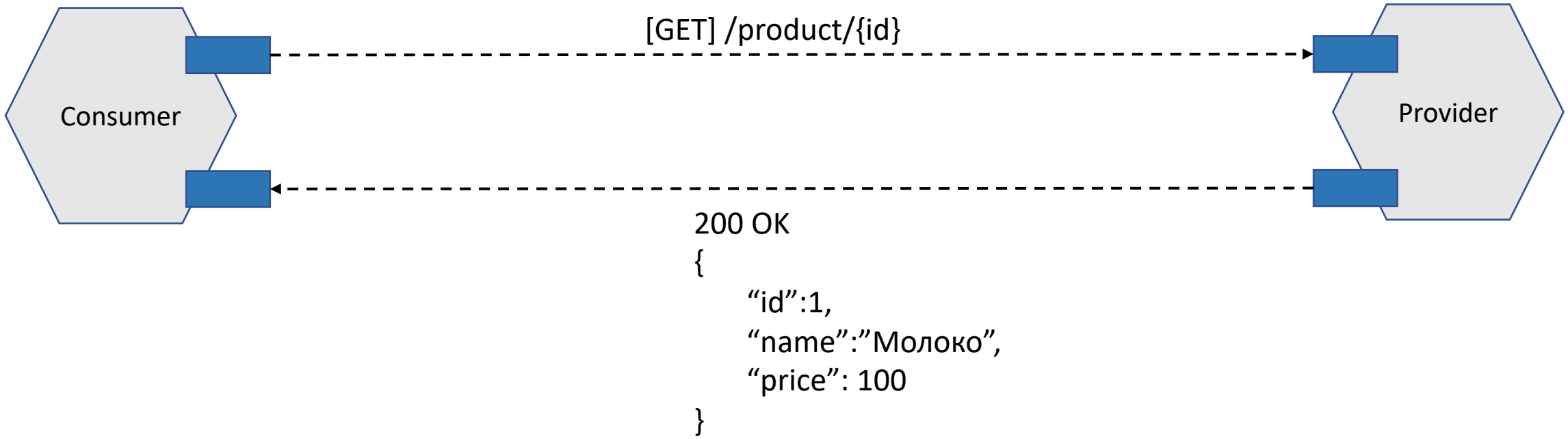
SKLADCHIK.ORG

# CDC тесты (контрактов)



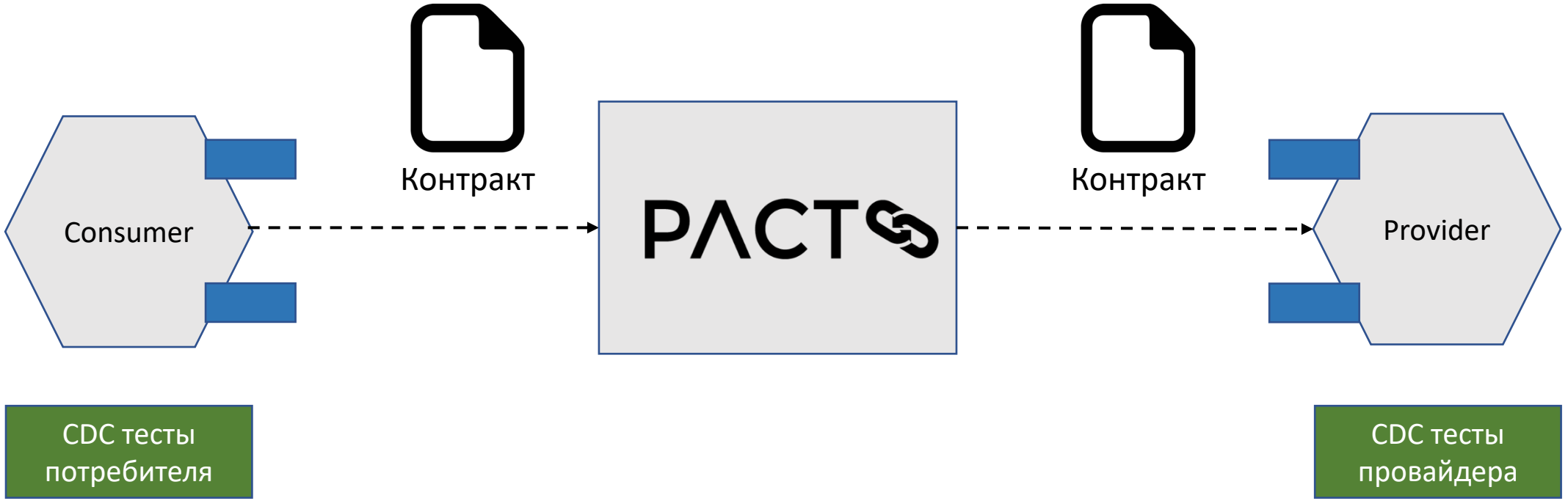



# Интеграционное тестирование взаимодействия в стиле RPC

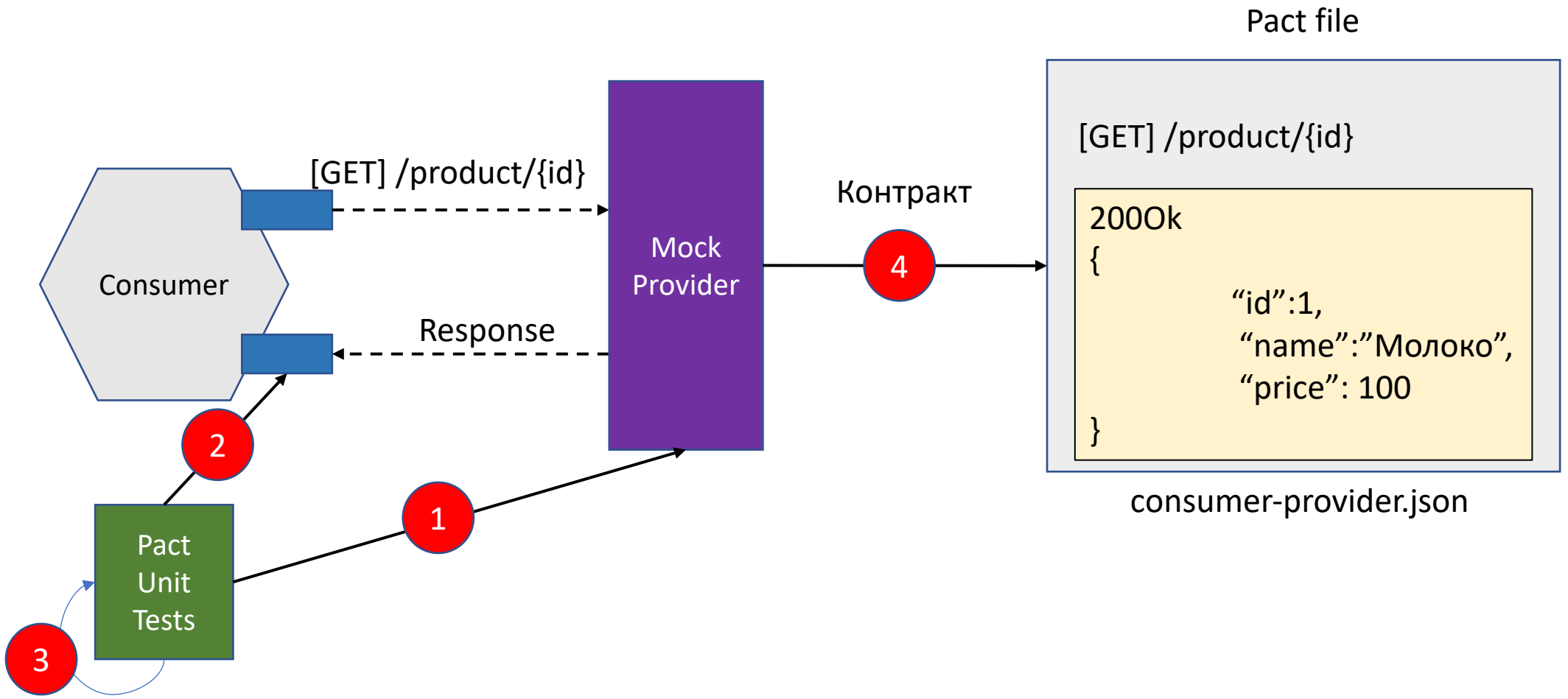


# Популярные решения

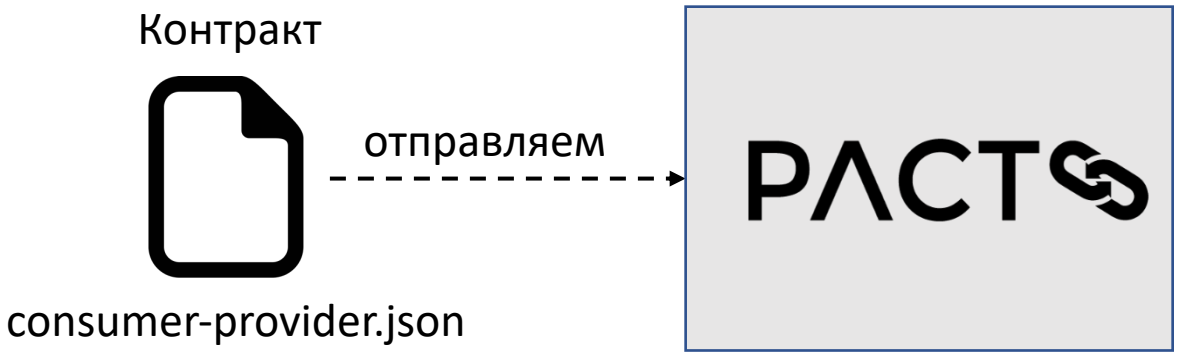
# Принцип работы



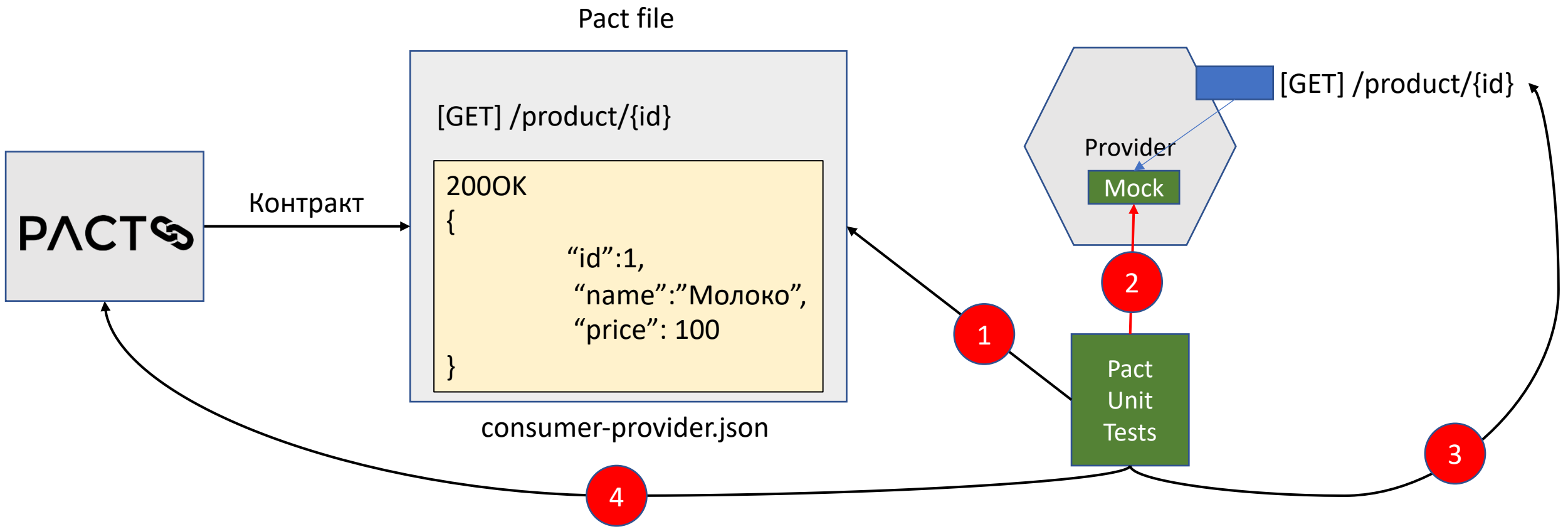
# Тестирование Consumer



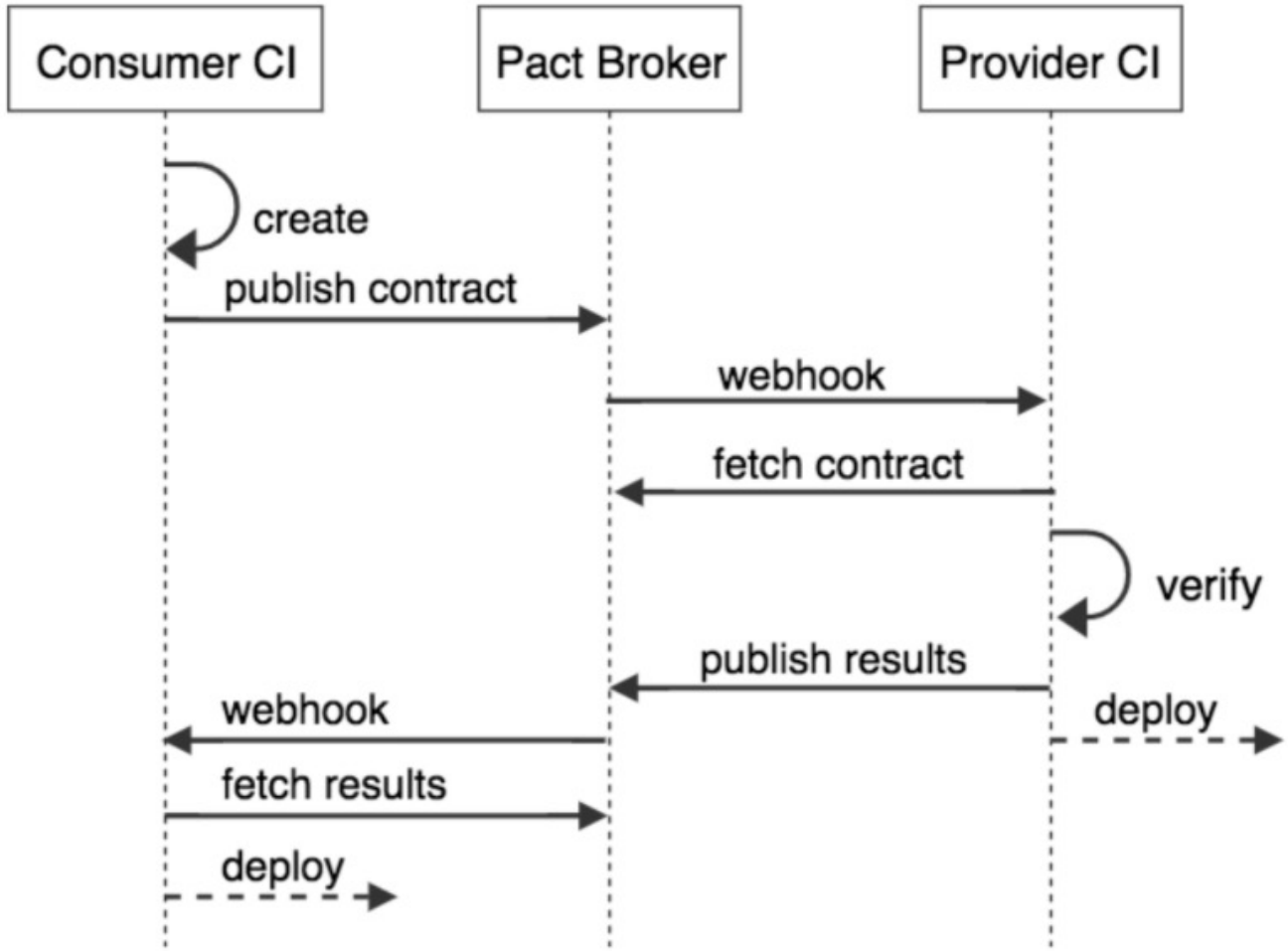
# Публикация контракта



# Тестирование Provider



# Связь pipeline’ов через webhook



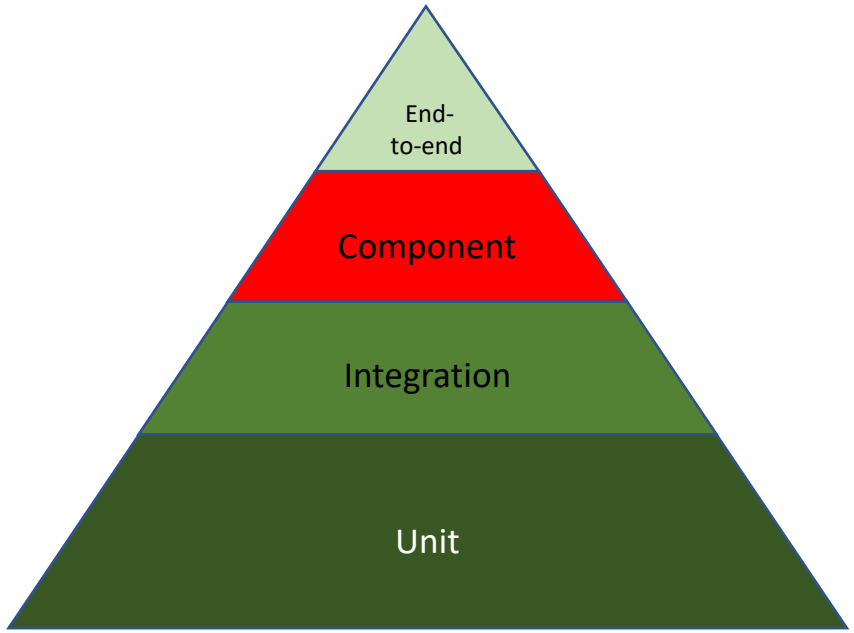
# Место в CI

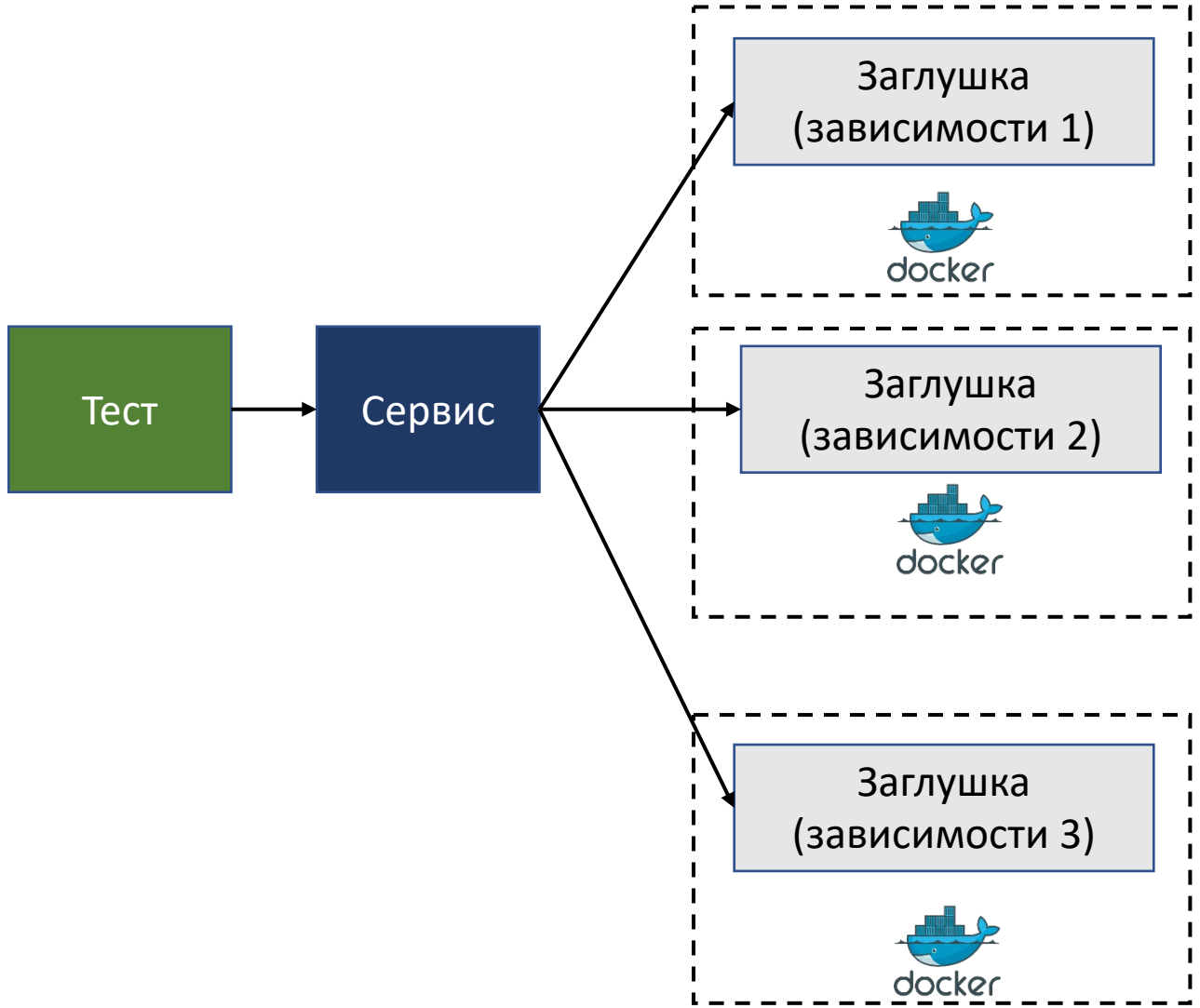
# Component тесты

SKLADCHIK.ORG

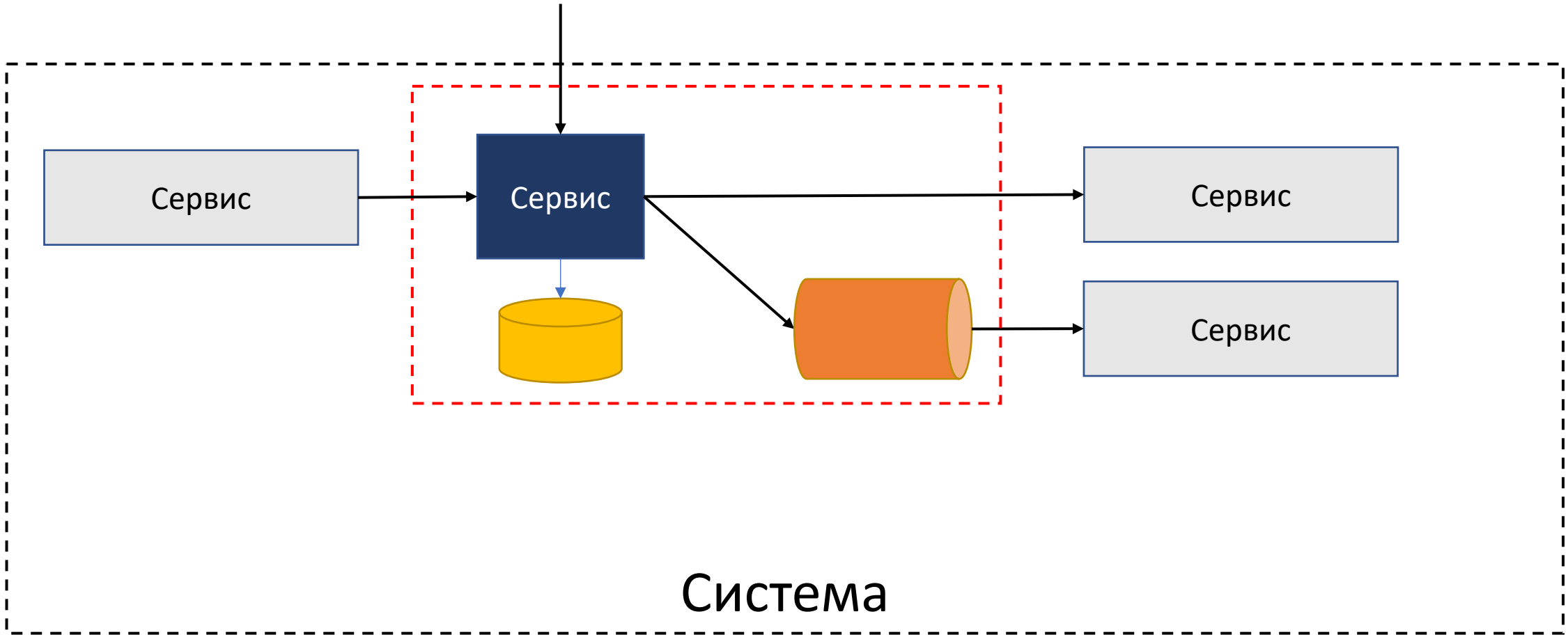
# Service Component Test pattern

# Component

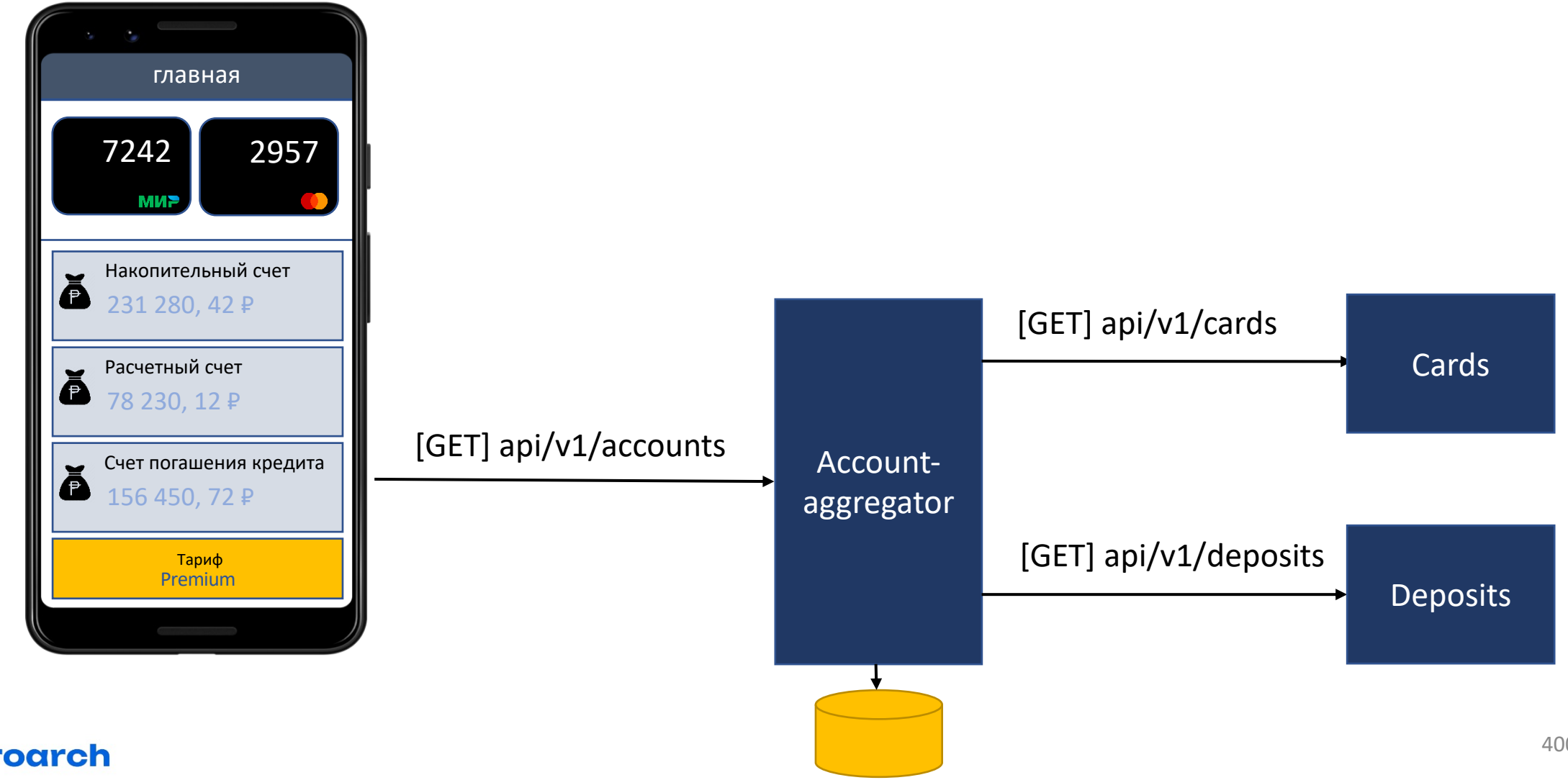

End-
to-end
Component
Integration
Unit


Тест
Сервис
Заглушка
(зависимости 1)
docker
Заглушка
(зависимости 2)
docker
Заглушка
(зависимости 3)
docker

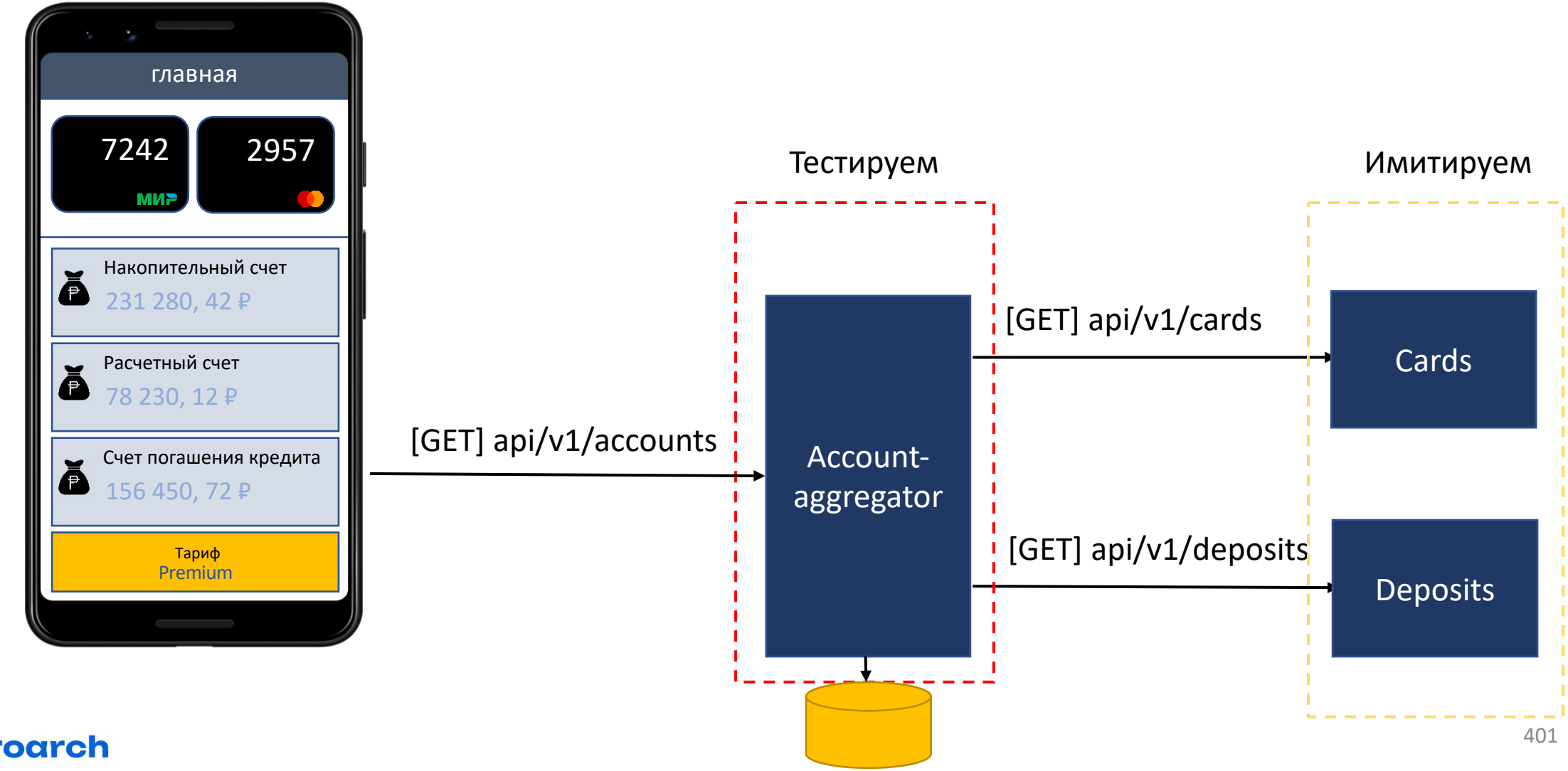
# Тестируем запущенный сервис в изоляции



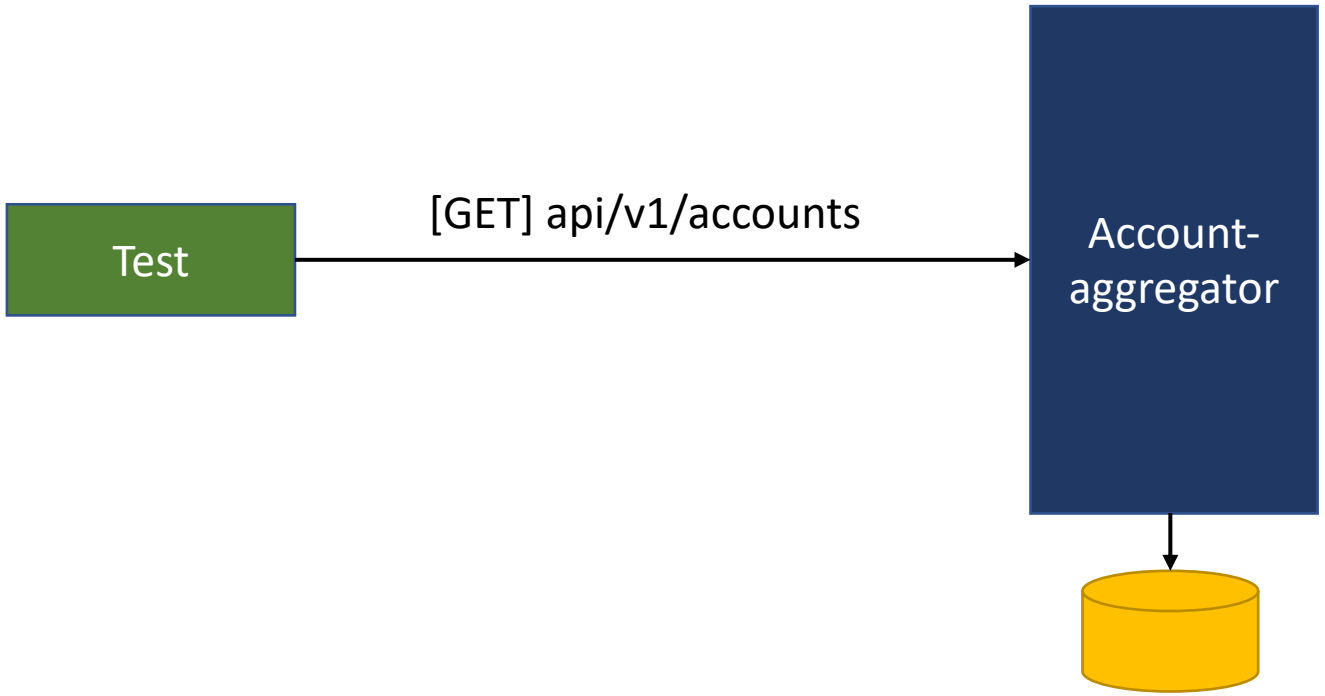
# Пример


главная
7242
МИР
2957
Накопительный счет
231 280, 42 ₽
Расчетный счет
78 230, 12 ₽
Счет погашения кредита
156 450, 72 ₽
Тариф
Premium
[GET] api/v1/accounts
Account-aggregator
[GET] api/v1/cards
Cards
[GET] api/v1/deposits
Deposits

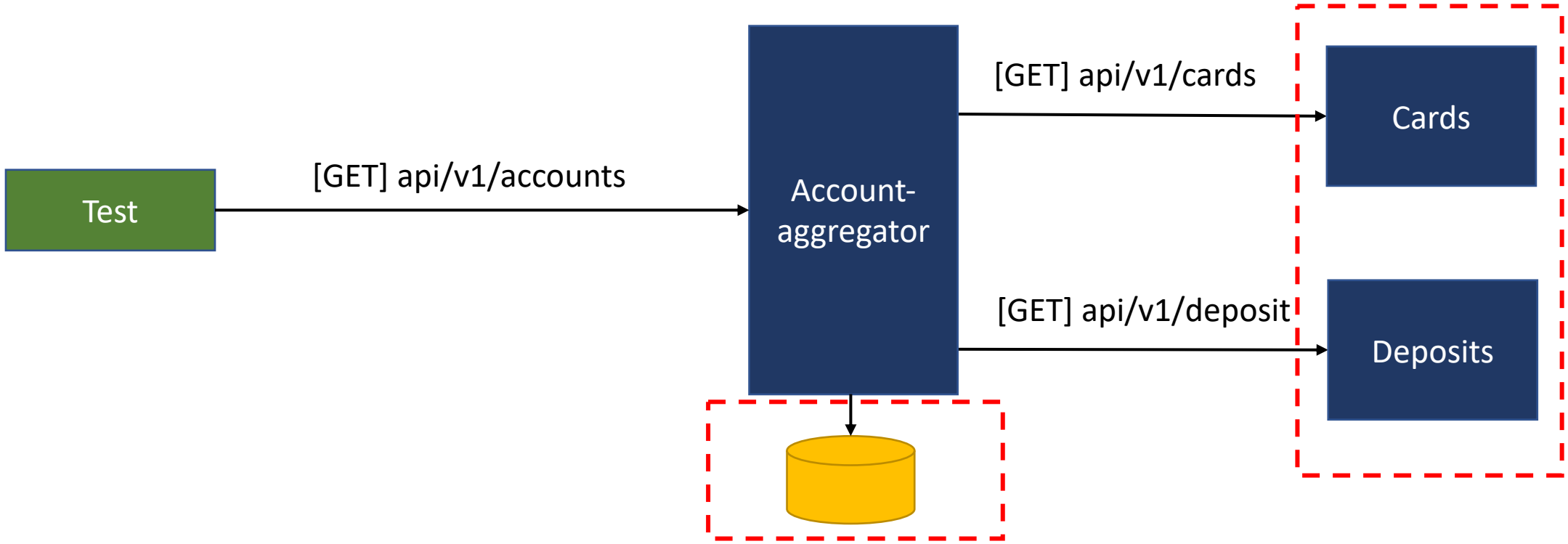
# Что тестируем



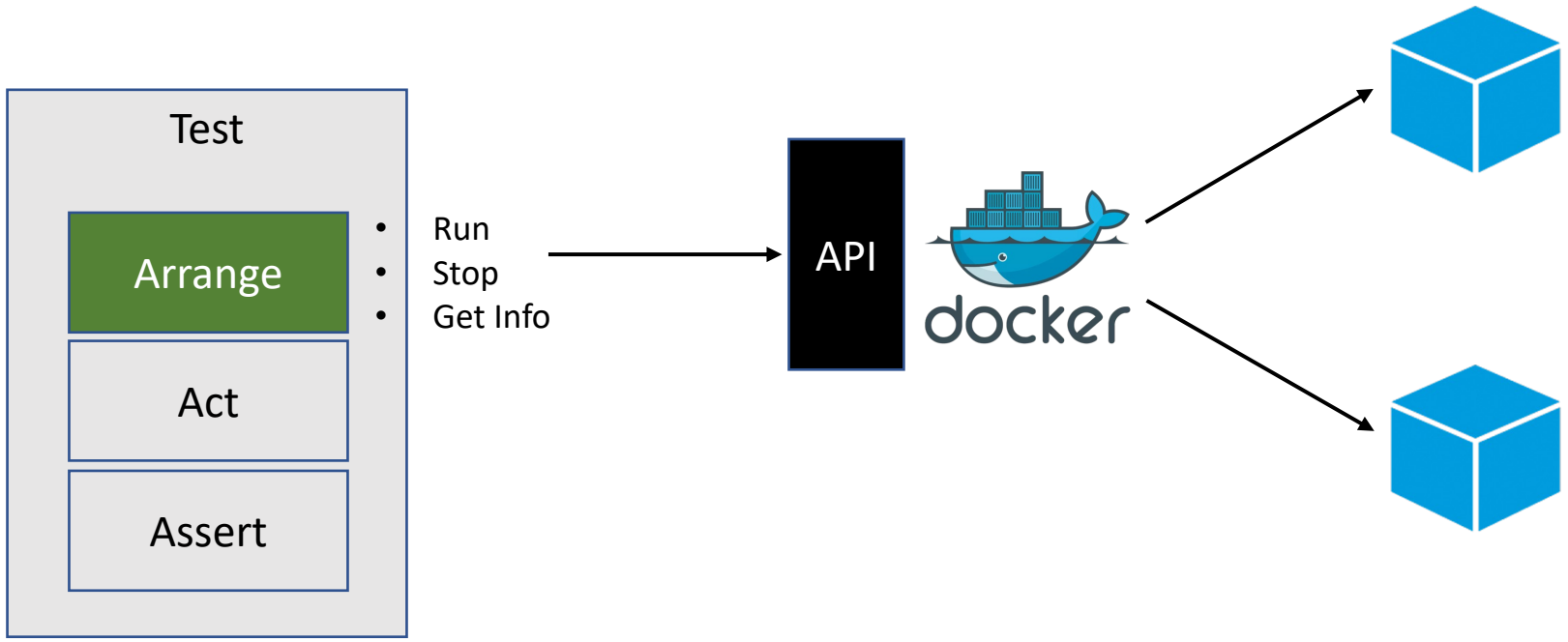
# Тесты направлены на API методы



# Что делать с зависимостями



# Библиотеки управления Docker



# Имплементация идеи (Java, Go lang, C#)

JAVA

https://www.testcontainers.org/

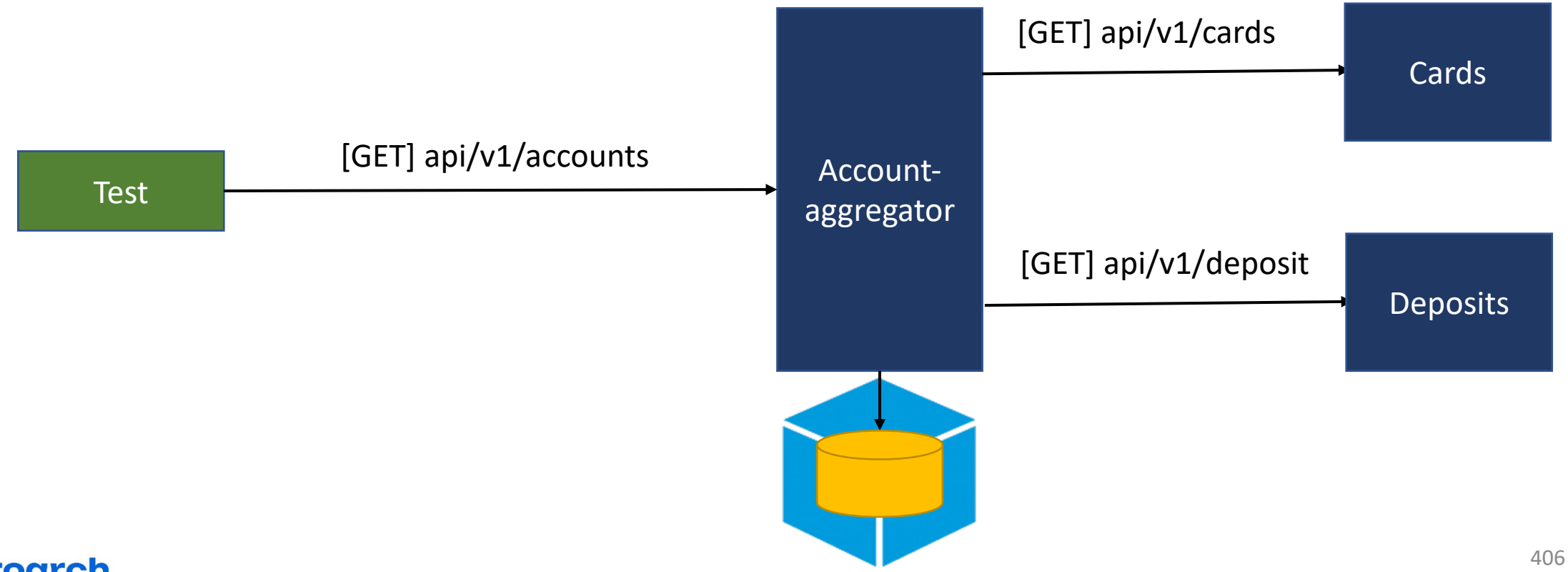

Go lang

https://github.com/testcontainers/testcontainers-go


C#

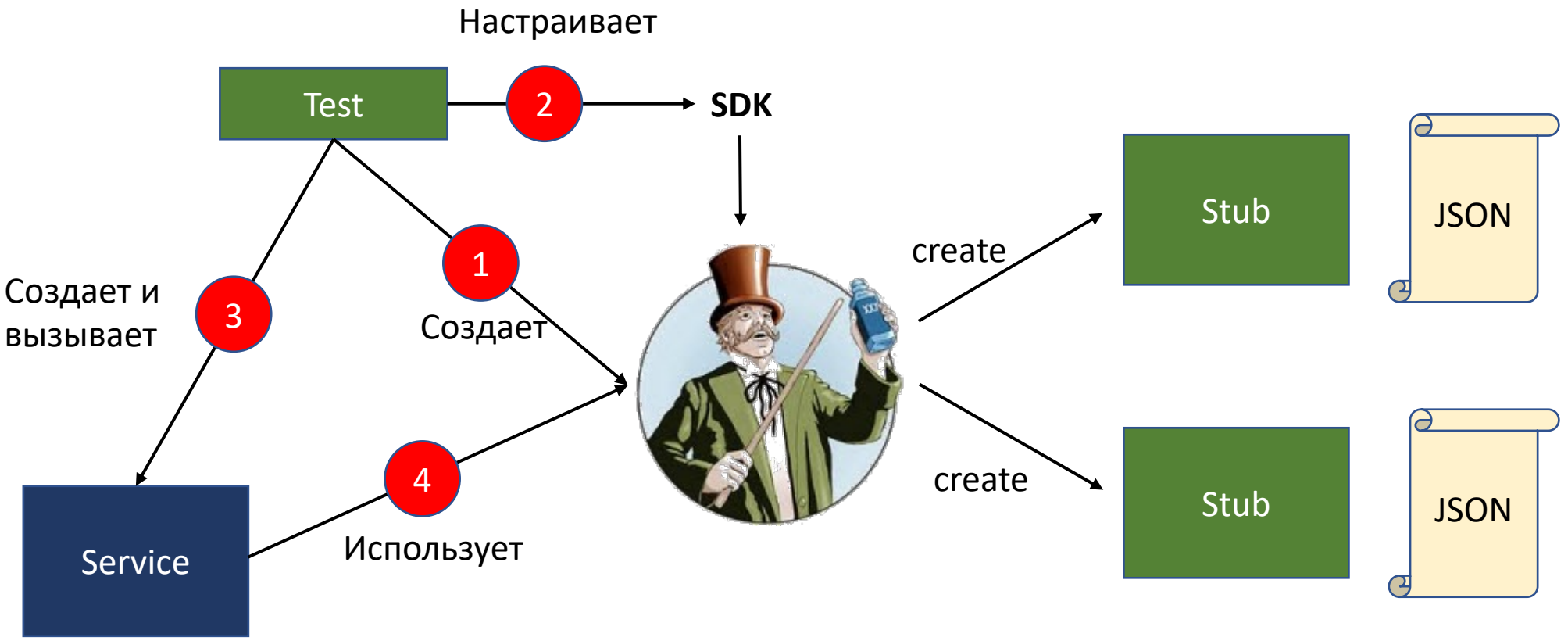
https://github.com/HofmeisterAn/dotnet-testcontainers

# БД в Docker

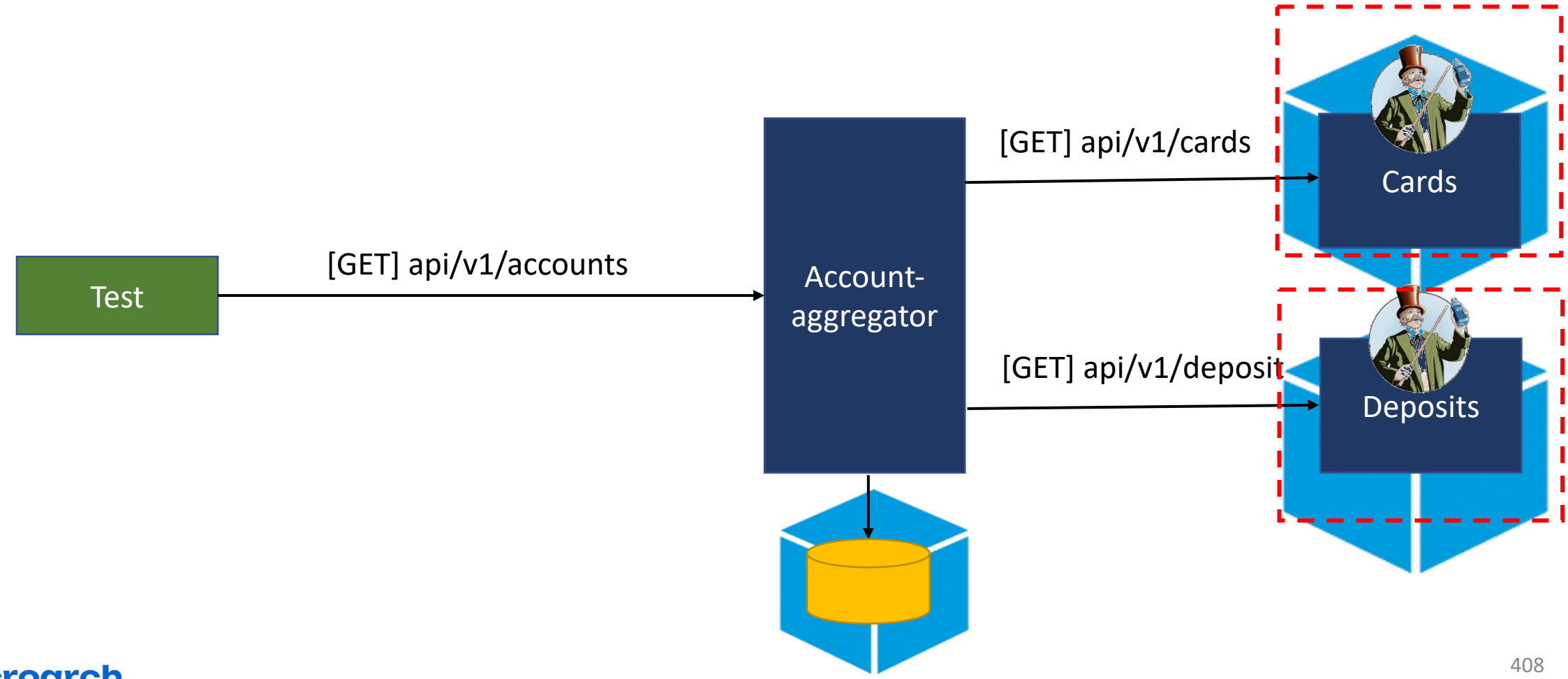

Test
[GET] api/v1/accounts
Account-aggregator
[GET] api/v1/cards
Cards
[GET] api/v1/deposit
Deposits

SKLADCHIK.ORG

# Конфигурация Mountebank



# БД в Docker + Mountebank в Docker



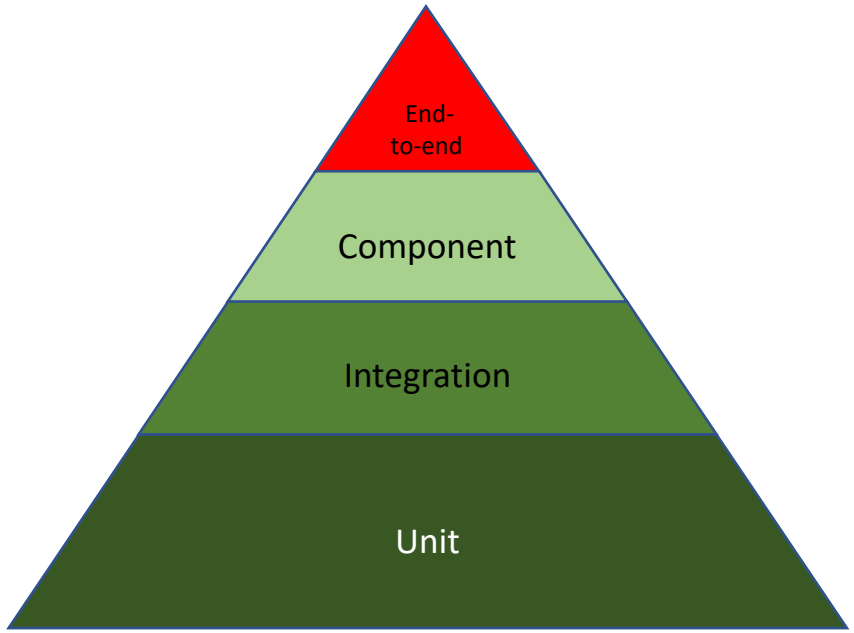
# Как это работает

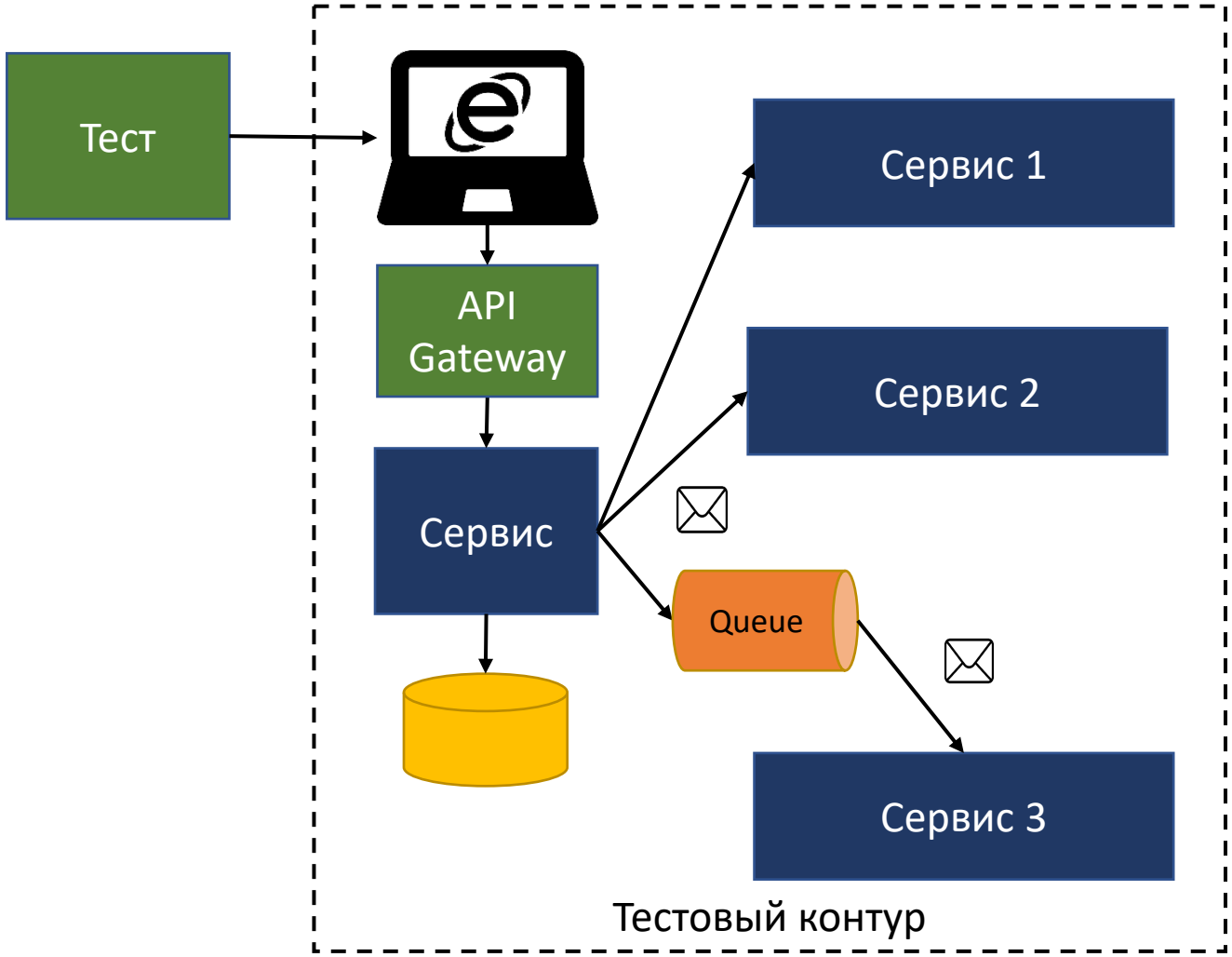
| | |
|---|---|
| Arrange | • Создание Docker контейнеров<br>• Подготовка состояния БД и Mountebank |
| Act | • Прогон сценария(вызов методов) |
| Assert | • Все ок?(да/нет)<br>• Утилизация контейнеров |

# Место в CI

# End-to-end тесты

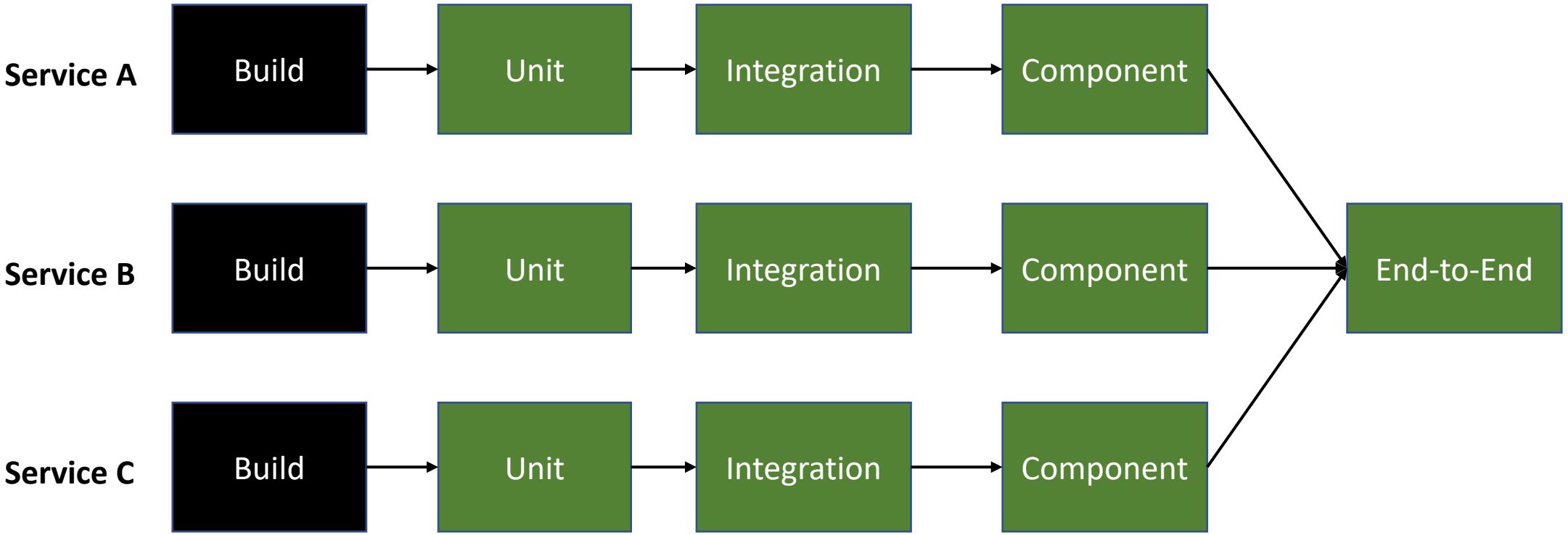
# End-to-end


End-
to-end
Component
Integration
Unit

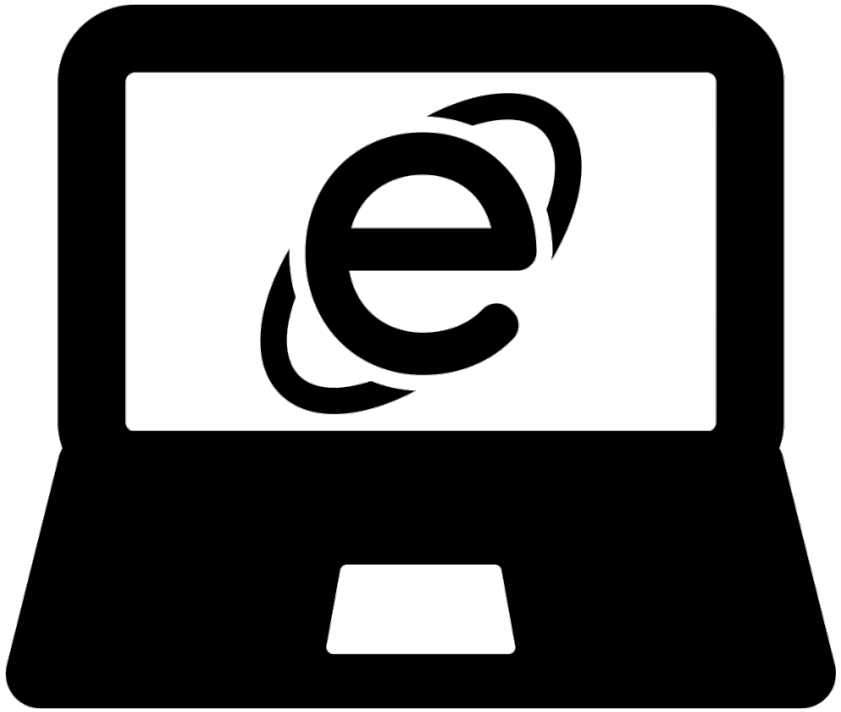

Тест
API
Gateway
Сервис
Сервис 1
Сервис 2
Queue
Сервис 3
Тестовый контур

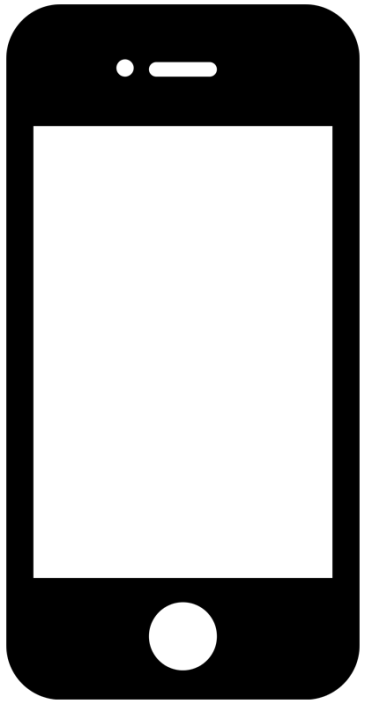
# Тестируйте CJM, а не User Story

# Когда запускать End-to-End тесты

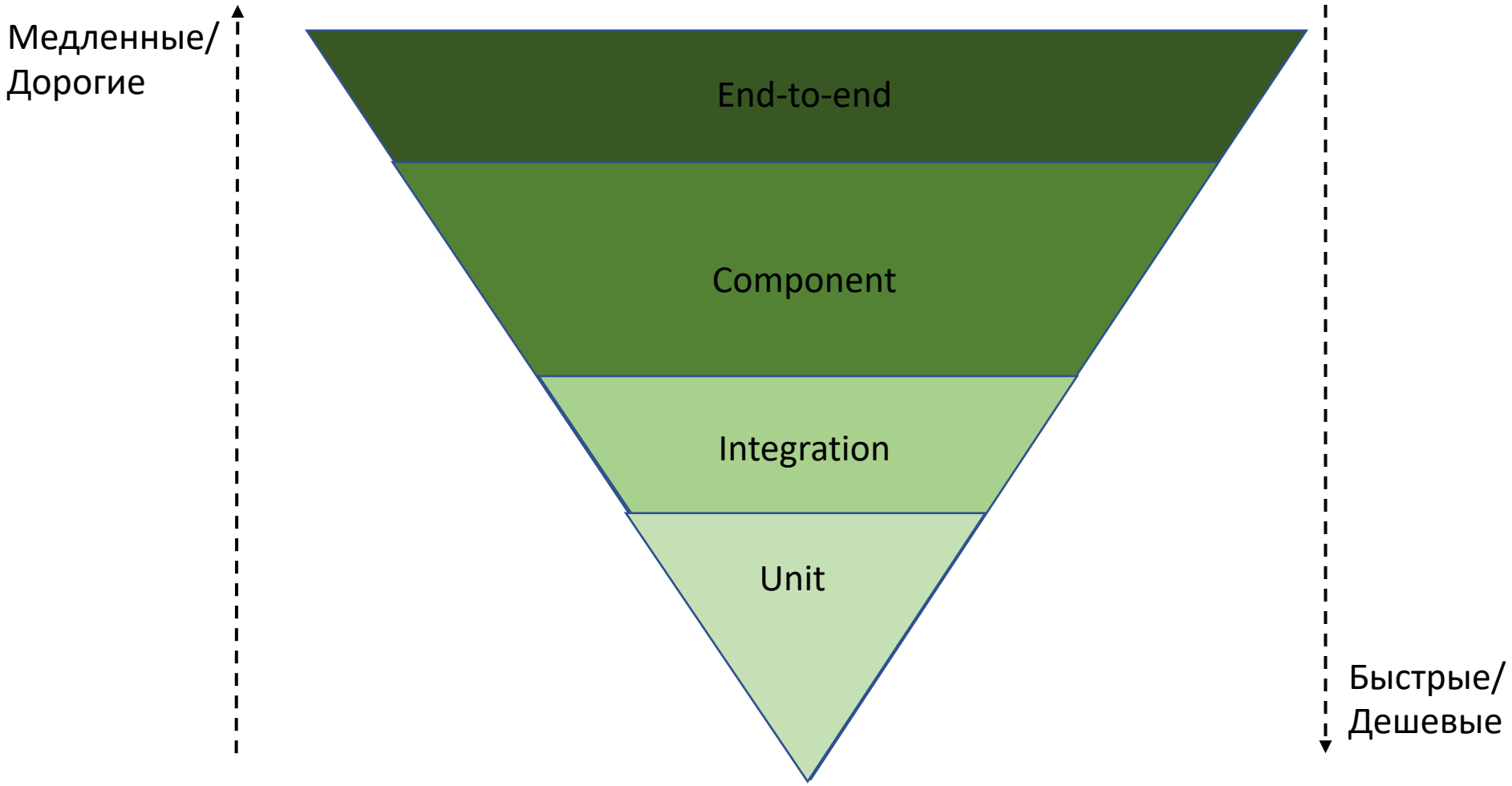


# А что с фронтами?



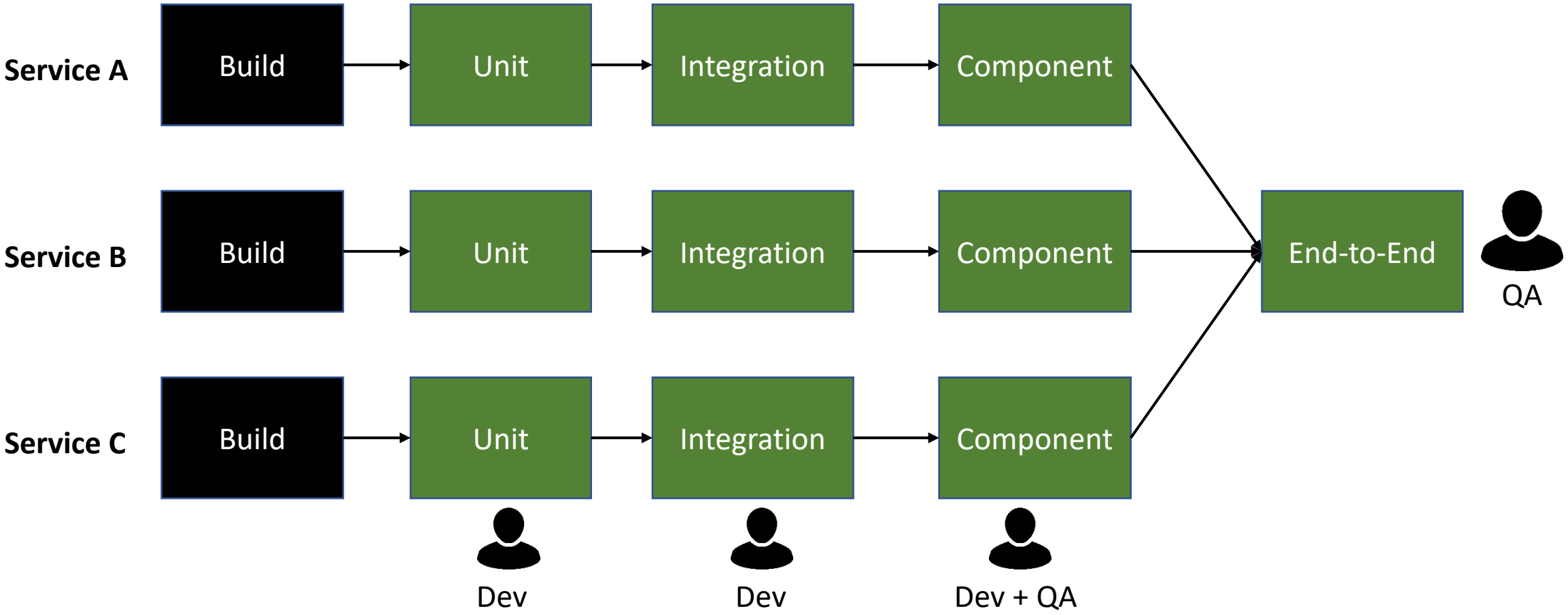


# Антипаттерн «рожок с мороженным»

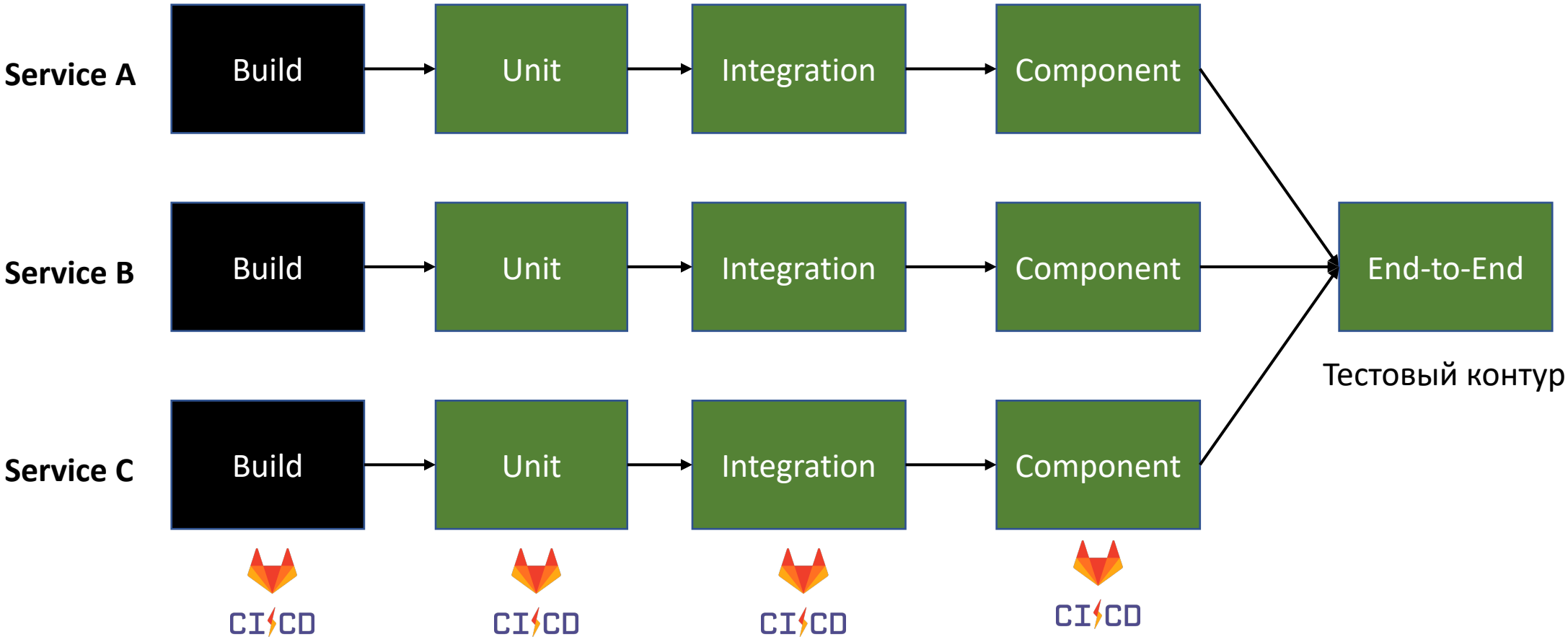


SKLADCHIK.ORG

# Кто создает все эти тесты



# Где проходят тесты



# О чём узнали

- End-to-End тесты мало пригодны для тестирования распределённых систем, лучше их заменять группой более низкоуровневых тестов
- Если тесты отвечают на вопрос – «Можно ли развёртывать сервис в Production?», то это хорошие тесты
- Тестовый контур нужен, но уже не так важен, все основные тесты можно запускать прямо в Pipeline
- Если разработчики пишут и код, и тесты, то это сокращает Lead Time

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

# Мониторинг и поддержка

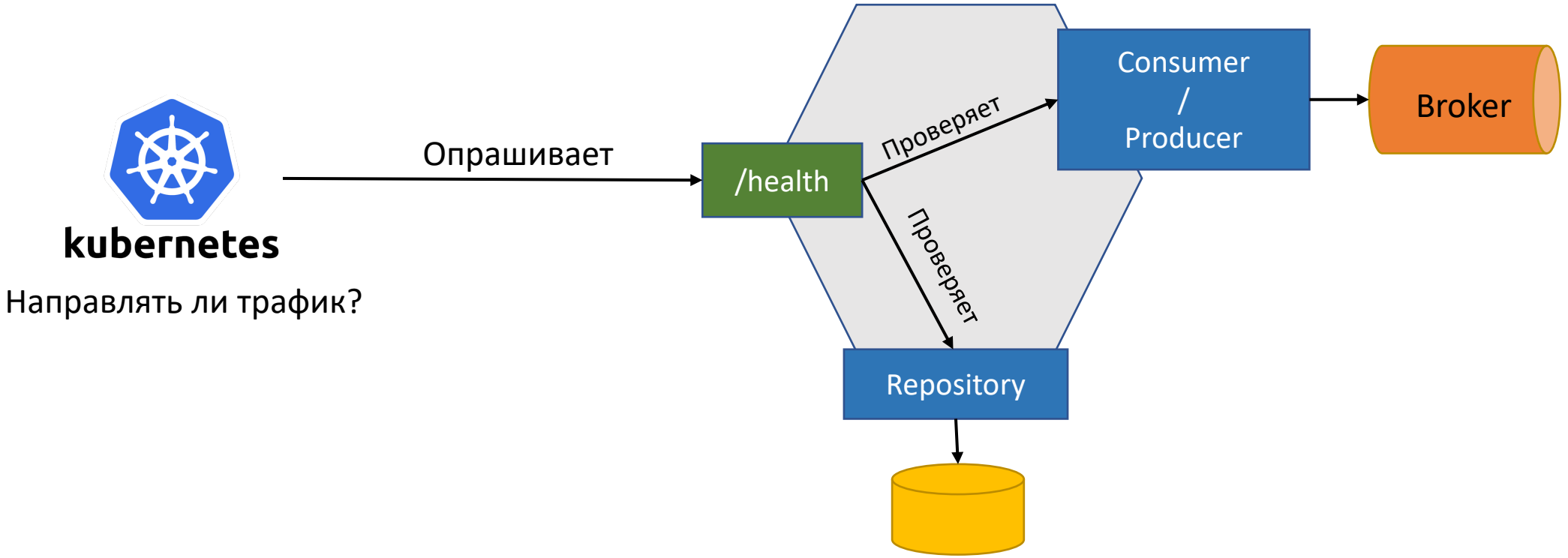
# Цели занятия

- Узнать, как осуществлять мониторинг микросервисной системы
- Разобраться, как диагностировать ошибки с помощью логов
- Познакомиться с основными способами масштабирования

# Мониторинг

# Health Check API pattern

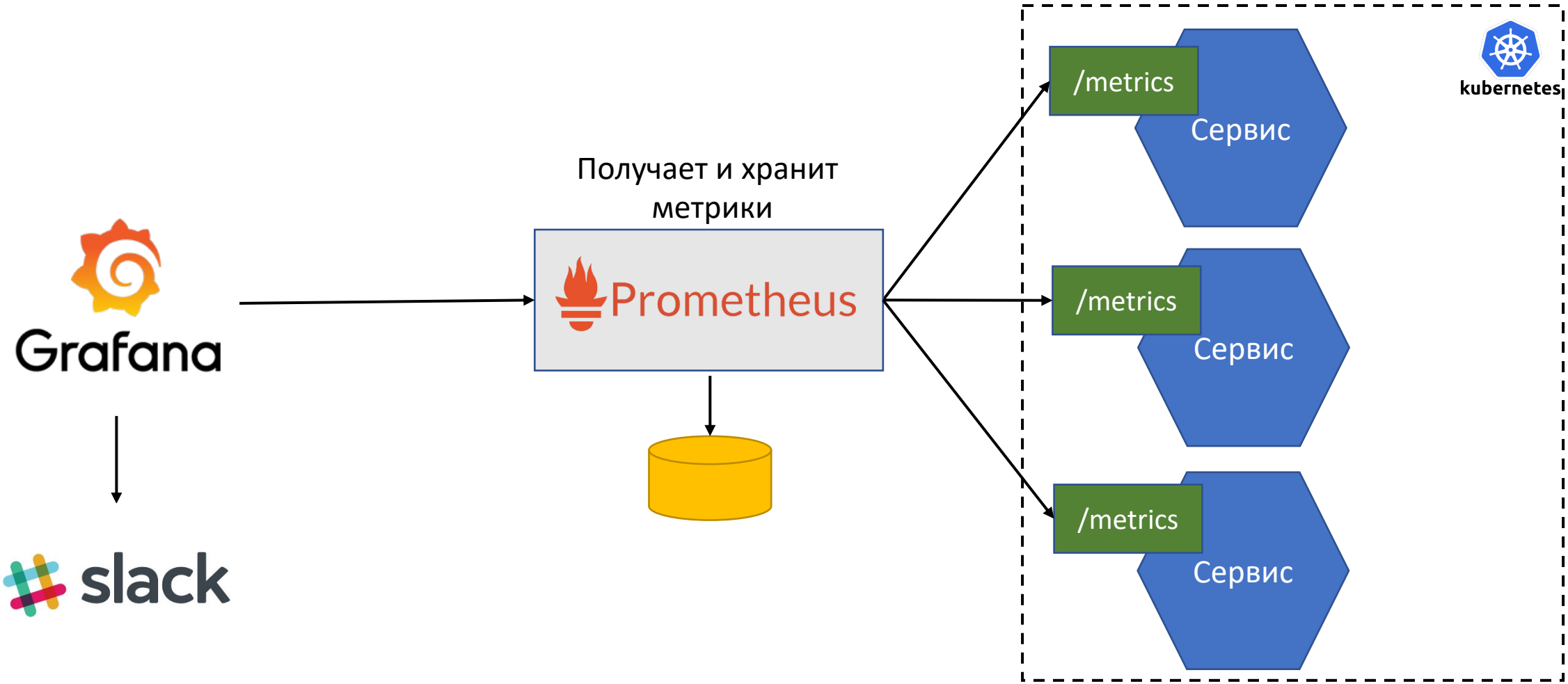
# API проверки работоспособности



# Application metrics pattern

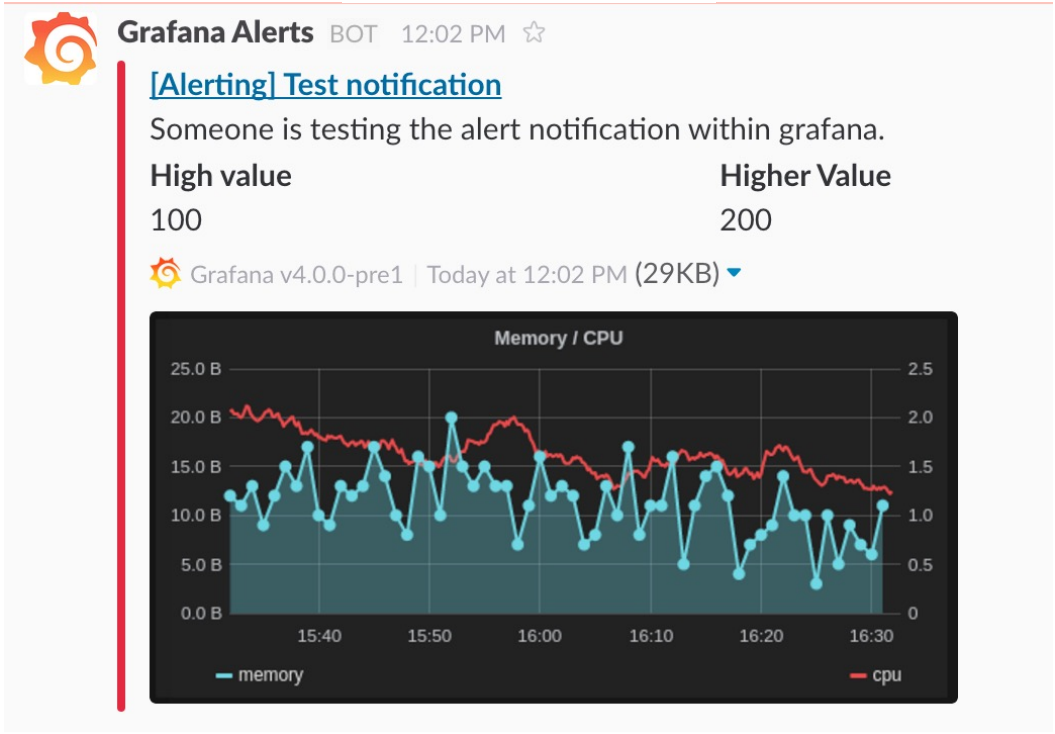
SKLADCHIK.ORG

# Мониторинг состояния сервиса



# Alerts и Grafana

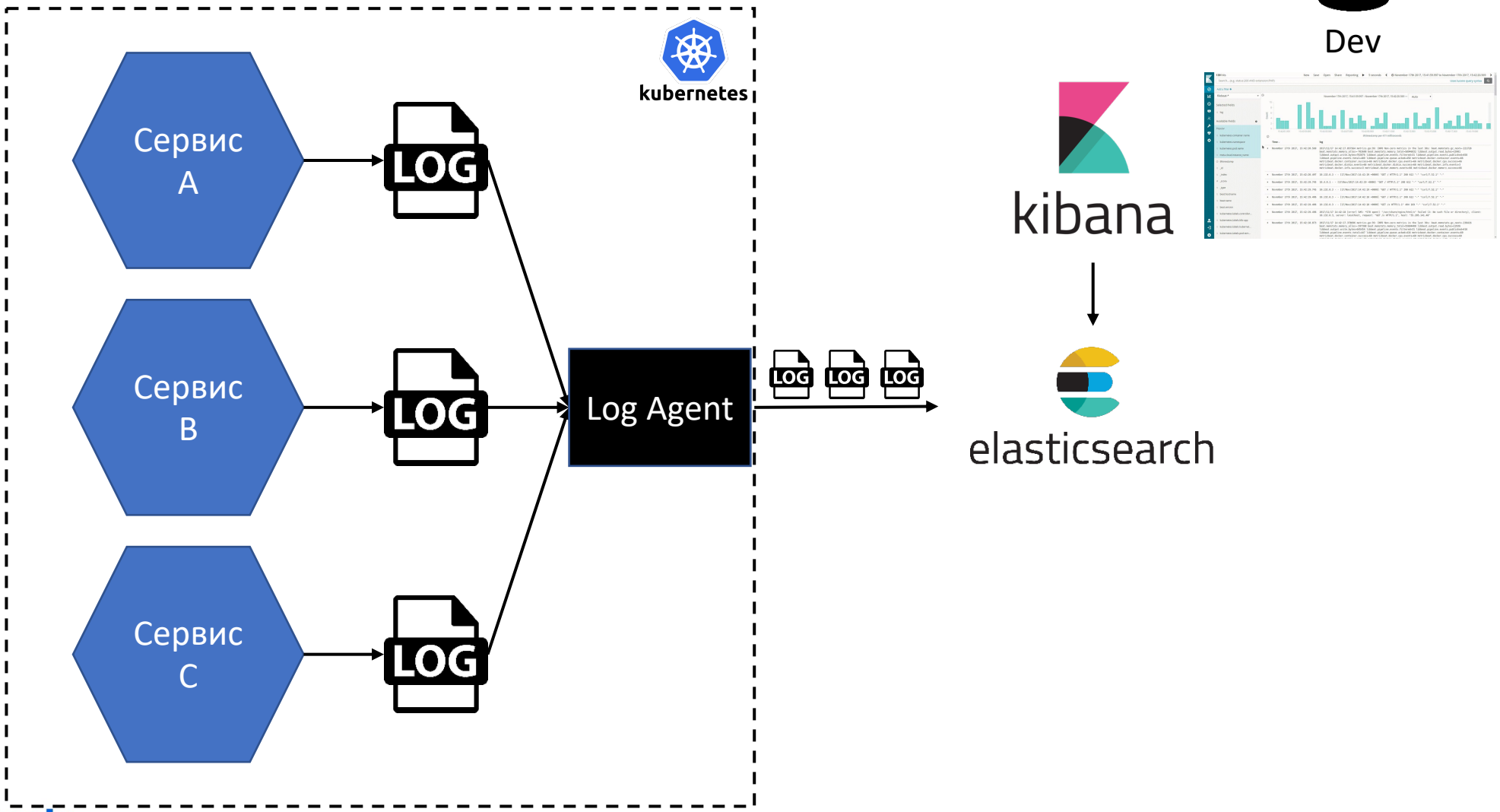
slack



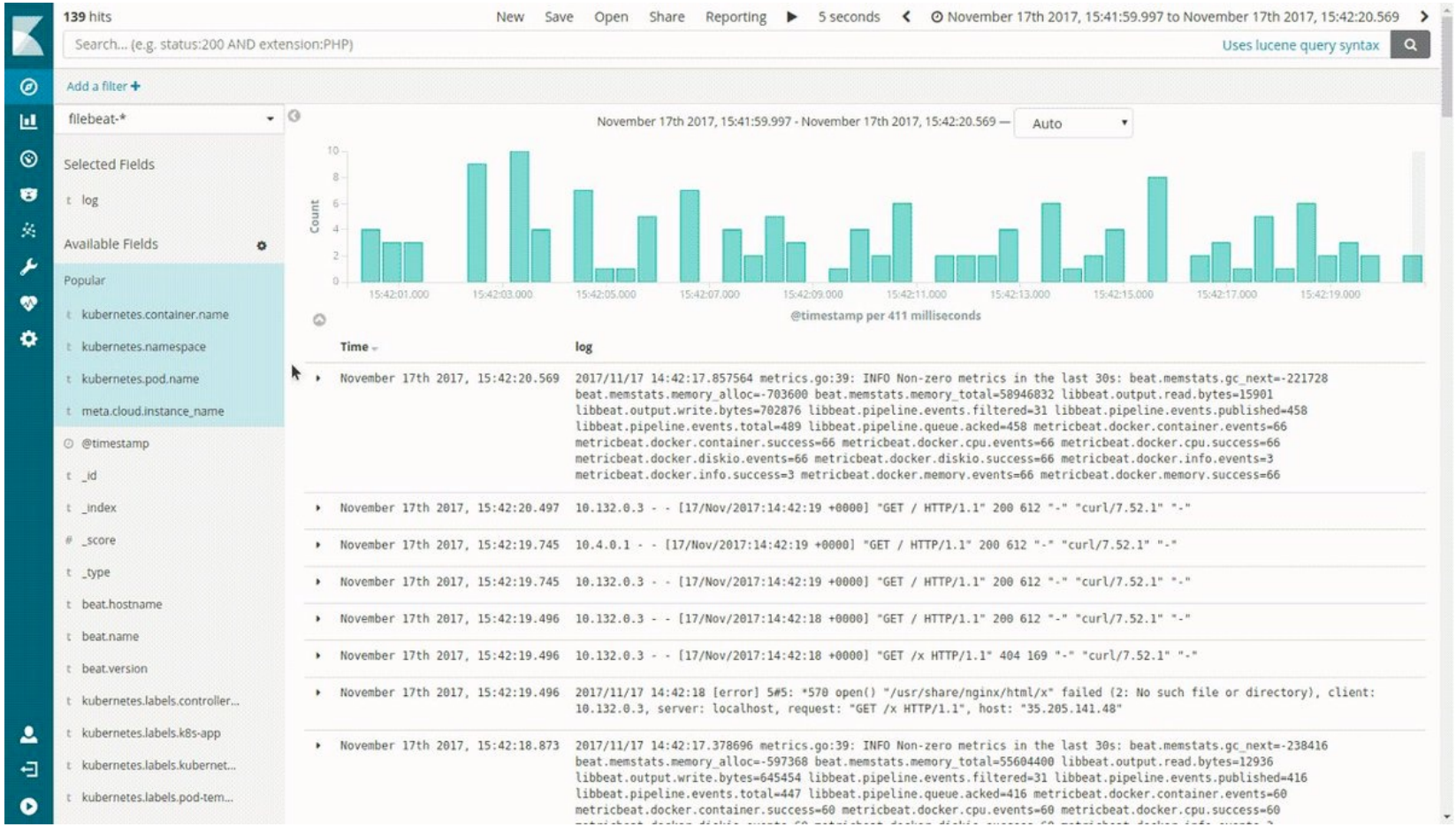

# Сбор логов

# Log aggregation pattern

# Агрегация логов



# Поиск логов


139 hits
New
Save
Open
Share
Reporting
5 seconds
November 17th 2017, 15:41:59.997 to November 17th 2017, 15:42:20.569
Search... (e.g. status:200 AND extension:PHP)
Uses lucene query syntax
Add a filter +
filebeat-*
November 17th 2017, 15:41:59.997 - November 17th 2017, 15:42:20.569 —
Auto
Selected Fields
t log
Available Fields
Popular
t kubernetes.container.name
t kubernetes.namespace
t kubernetes.pod.name
t meta.cloud.instance_name
@timestamp
t _id
t _index
# _score
t _type
t beat.hostname
t beat.name
t beat.version
t kubernetes.labels.controller...
t kubernetes.labels.k8s-app
t kubernetes.labels.kubernet...
t kubernetes.labels.pod-tem...
Count
@timestamp per 411 milliseconds
Time
log
November 17th 2017, 15:42:20.569 2017/11/17 14:42:17.857564 metrics.go:39: INFO Non-zero metrics in the last 30s: beat.memstats.gc_next=-221728 beat.memstats.memory_alloc=-703600 beat.memstats.memory_total=58946832 libbeat.output.read.bytes=15901 libbeat.output.write.bytes=702876 libbeat.pipeline.events.filtered=31 libbeat.pipeline.events.published=458 libbeat.pipeline.events.total=489 libbeat.pipeline.queue.acked=458 metricbeat.docker.container.events=66 metricbeat.docker.container.success=66 metricbeat.docker.cpu.events=66 metricbeat.docker.cpu.success=66 metricbeat.docker.diskio.events=66 metricbeat.docker.diskio.success=66 metricbeat.docker.info.events=3 metricbeat.docker.info.success=3 metricbeat.docker.memory.events=66 metricbeat.docker.memory.success=66
November 17th 2017, 15:42:20.497 10.132.0.3 - - [17/Nov/2017:14:42:19 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.52.1" "-"
November 17th 2017, 15:42:19.745 10.4.0.1 - - [17/Nov/2017:14:42:19 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.52.1" "-"
November 17th 2017, 15:42:19.745 10.132.0.3 - - [17/Nov/2017:14:42:19 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.52.1" "-"
November 17th 2017, 15:42:19.496 10.132.0.3 - - [17/Nov/2017:14:42:18 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.52.1" "-"
November 17th 2017, 15:42:19.496 10.132.0.3 - - [17/Nov/2017:14:42:18 +0000] "GET /x HTTP/1.1" 404 169 "-" "curl/7.52.1" "-"
November 17th 2017, 15:42:19.496 2017/11/17 14:42:18 [error] 5#5: *570 open() "/usr/share/nginx/html/x" failed (2: No such file or directory), client: 10.132.0.3, server: localhost, request: "GET /x HTTP/1.1", host: "35.205.141.48"
November 17th 2017, 15:42:18.873 2017/11/17 14:42:17.378696 metrics.go:39: INFO Non-zero metrics in the last 30s: beat.memstats.gc_next=-238416 beat.memstats.memory_alloc=-597368 beat.memstats.memory_total=55604400 libbeat.output.read.bytes=12936 libbeat.output.write.bytes=645454 libbeat.pipeline.events.filtered=31 libbeat.pipeline.events.published=416 libbeat.pipeline.events.total=447 libbeat.pipeline.queue.acked=416 metricbeat.docker.container.events=60 metricbeat.docker.container.success=60 metricbeat.docker.cpu.events=60 metricbeat.docker.cpu.success=60

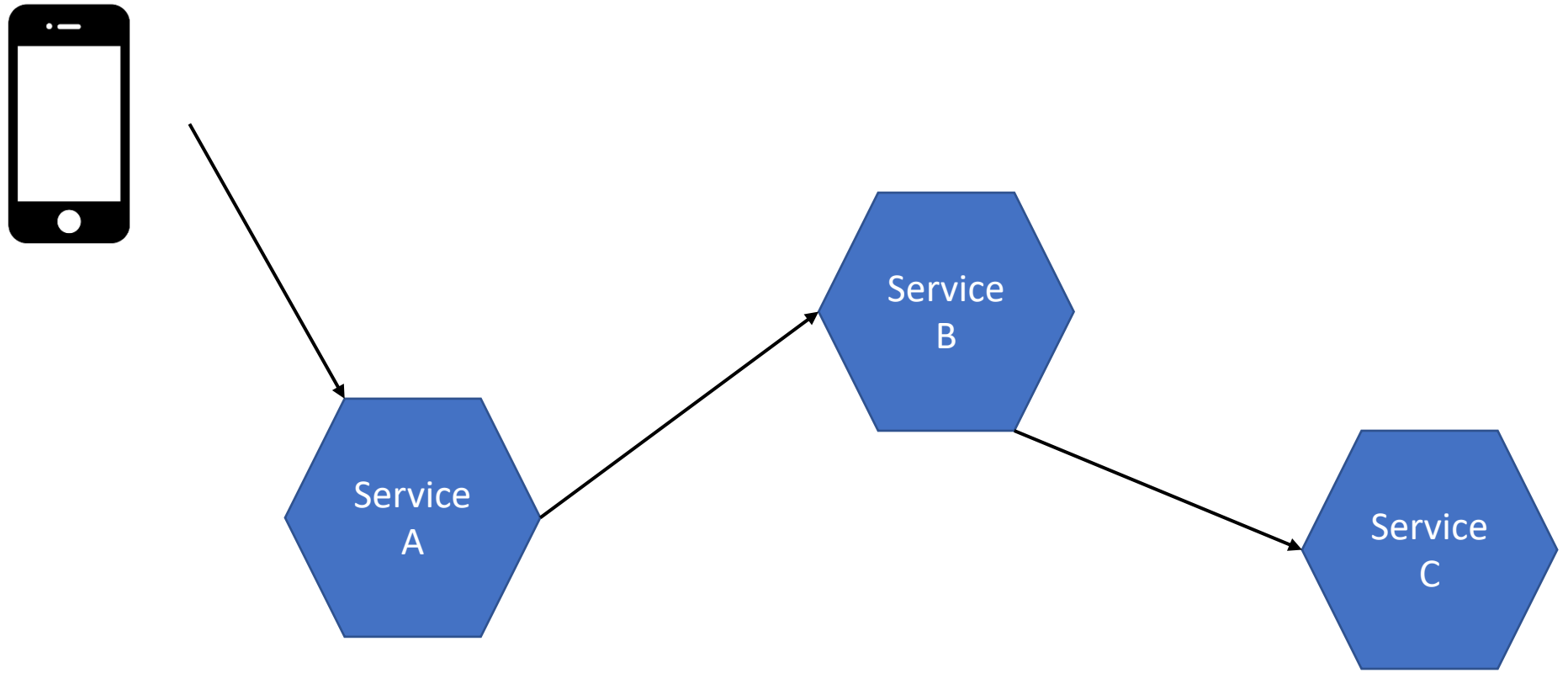
# Популярные решения


ELK

# Распределенная трассировка

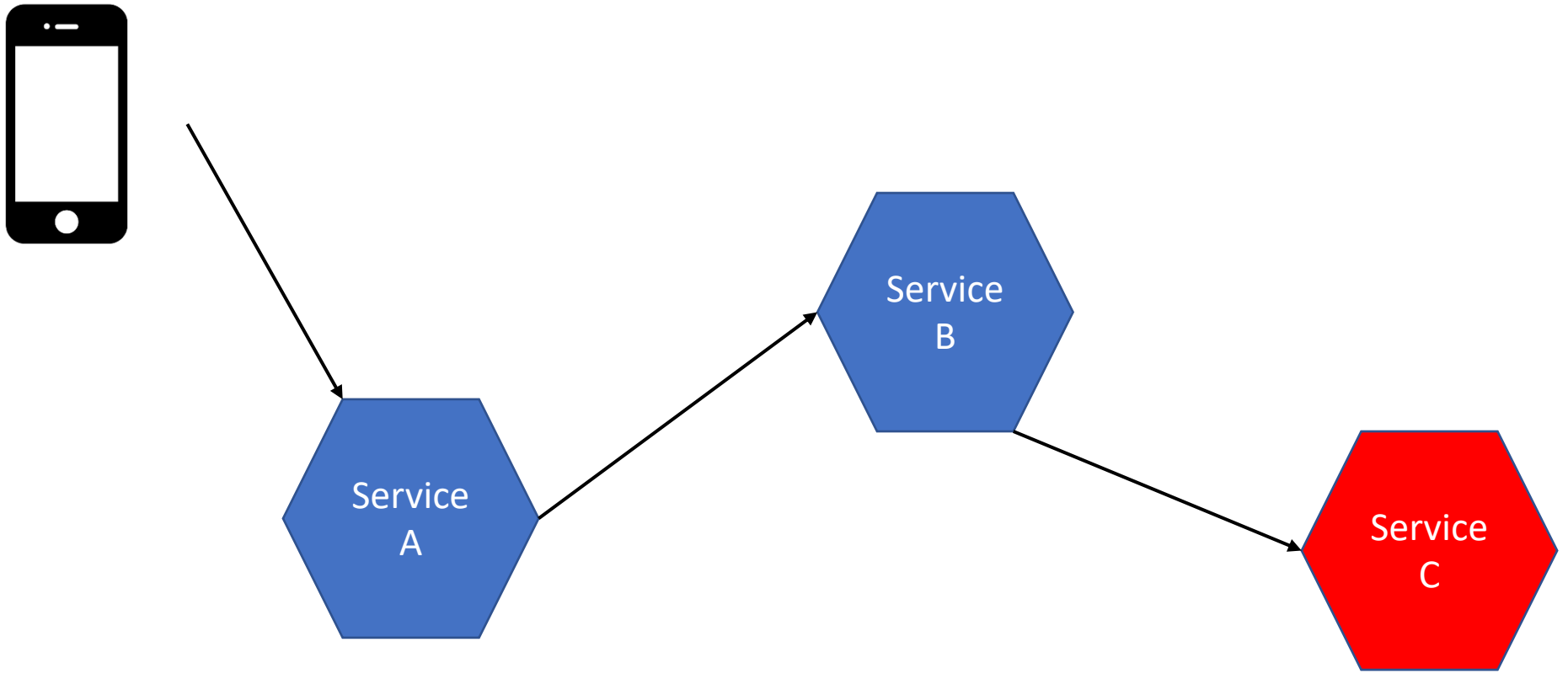
# Distributed tracing pattern

# Трассировка

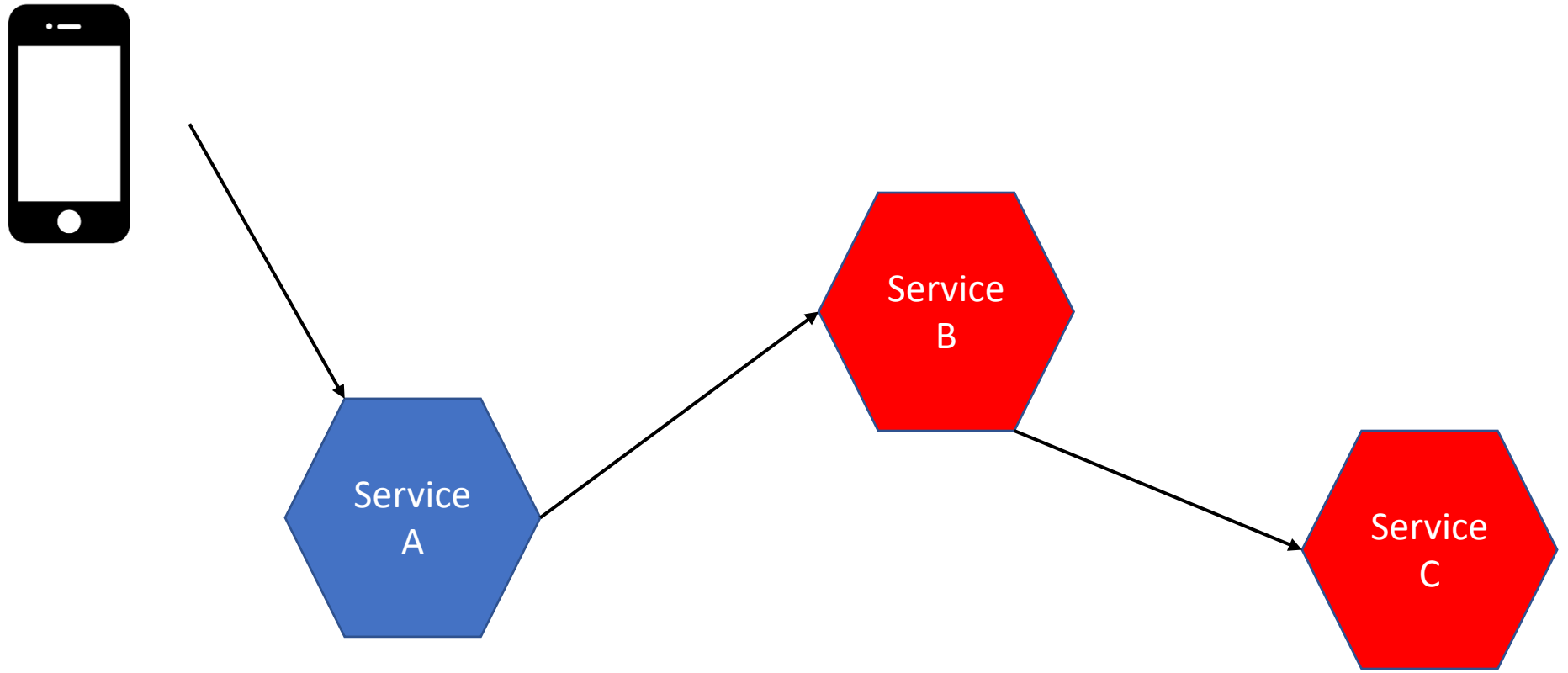


SKLADCHIK.ORG

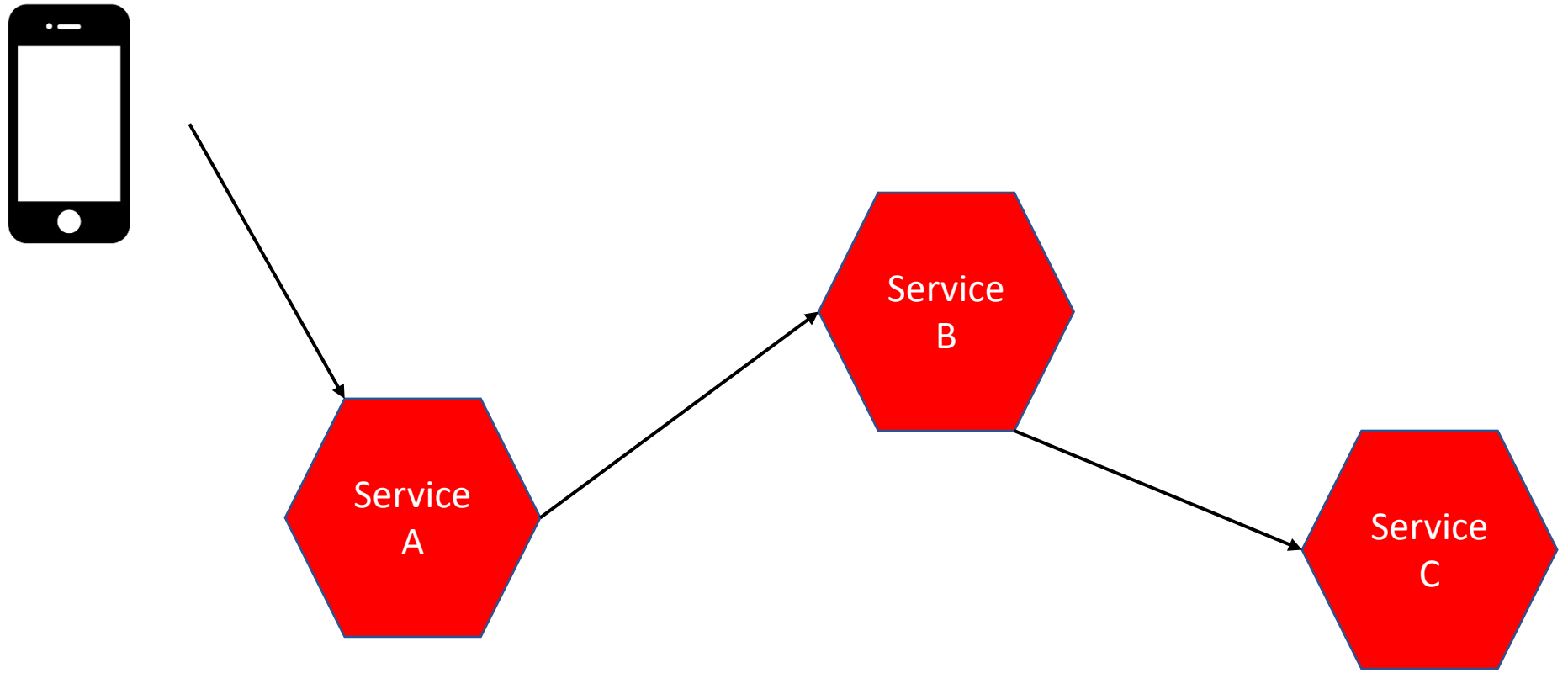
# Трассировка



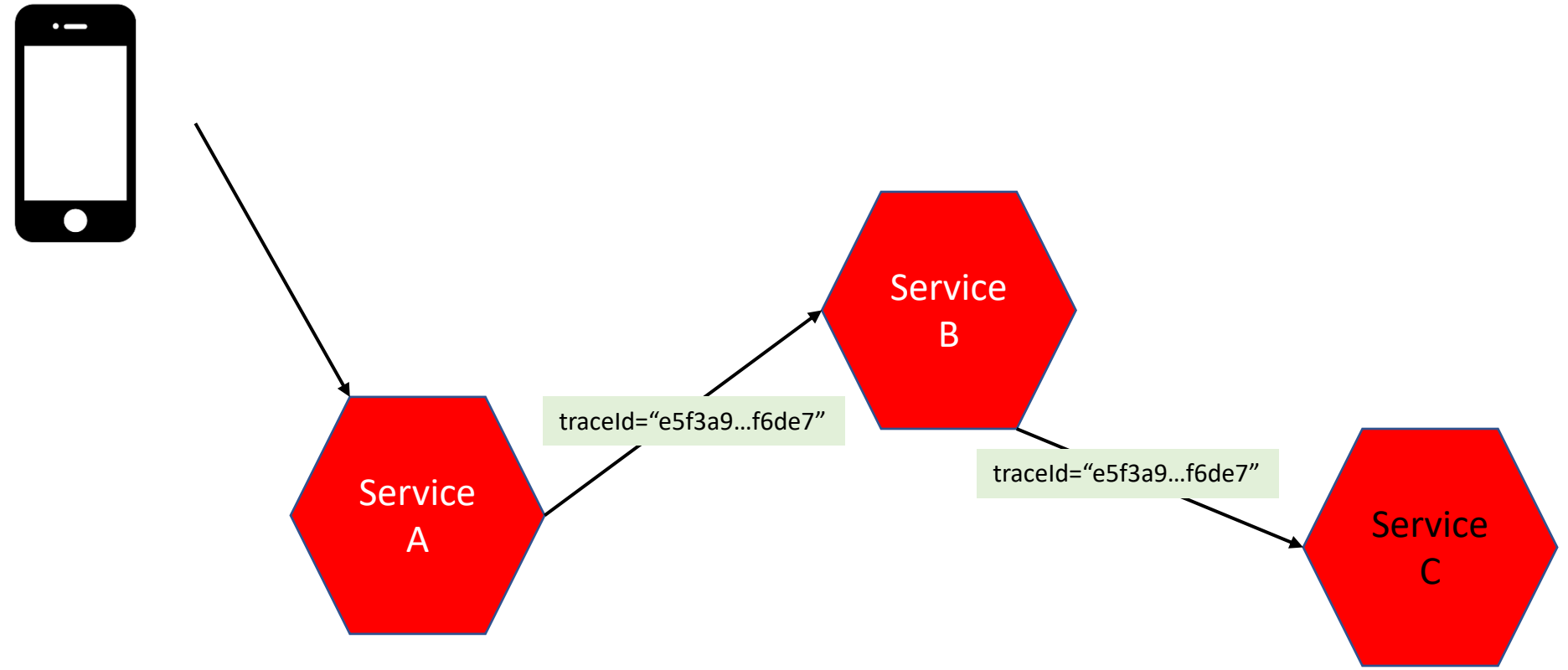
# Трассировка



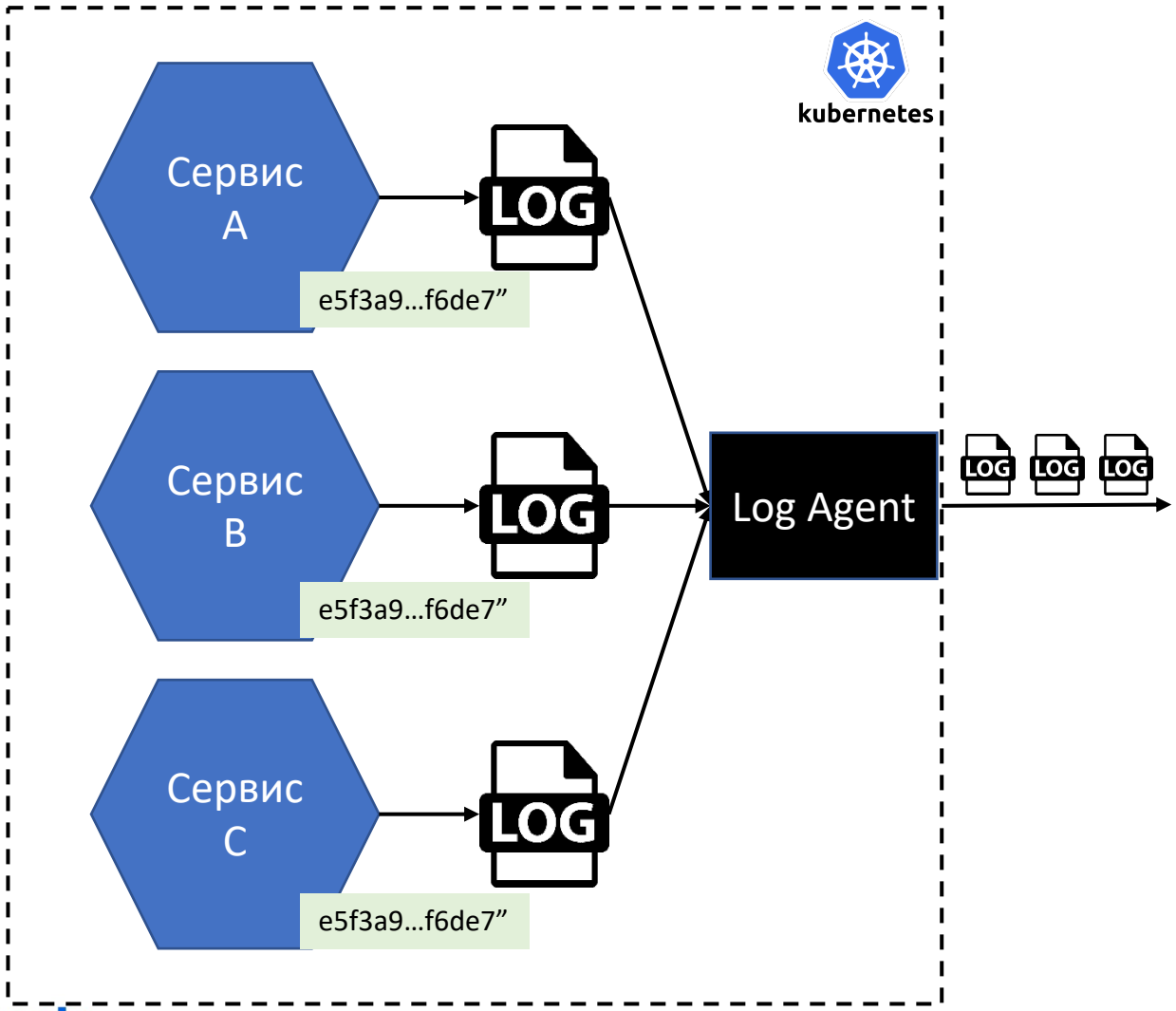
# Трассировка



# Трассировка

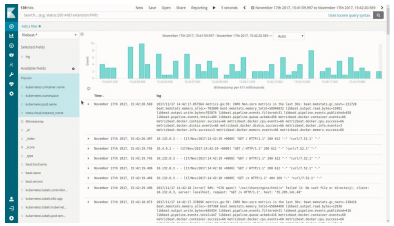


# Агрегация логов с трассировкой




Dev

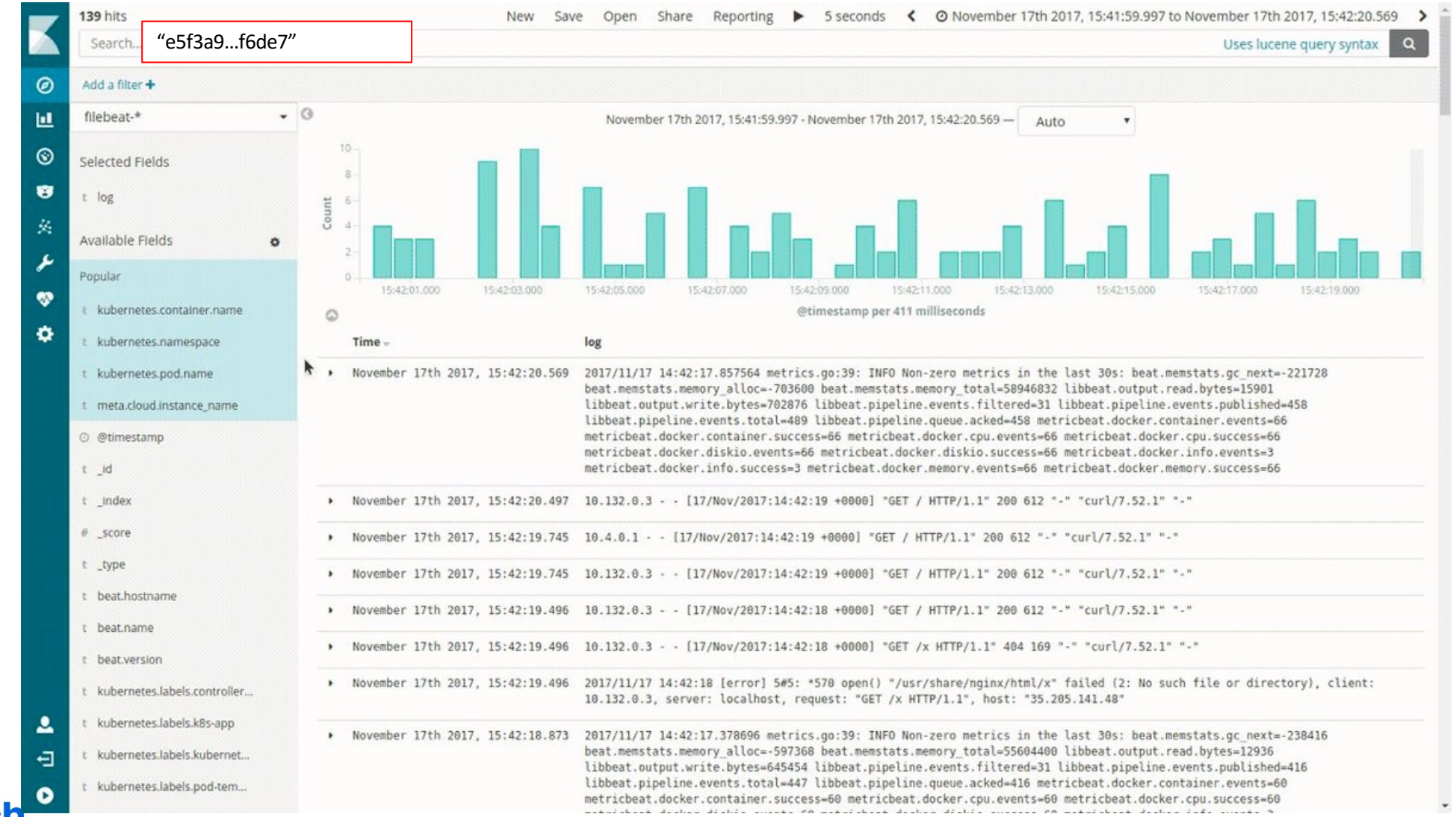
kibana



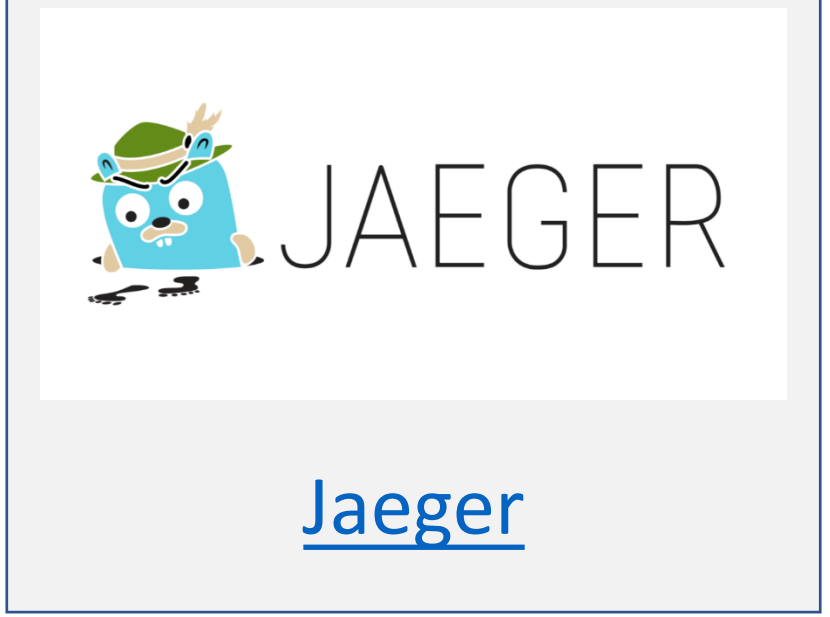
elasticsearch

| DateTime | App | Level | Message | TraceId |
|---|---|---|---|---|
| 15-02-2023 16:01:03 | Service A | INFO | Some Meassage | e5f3a9…f6de7 |
| 15-02-2023 16:01:02 | Service B | INFO | Some Meassage | e5f3a9…f6de7 |
| 15-02-2023 16:01:01 | Service C | ERROR | Exception Message | e5f3a9…f6de7 |

# Трассировка



# Популярные решения

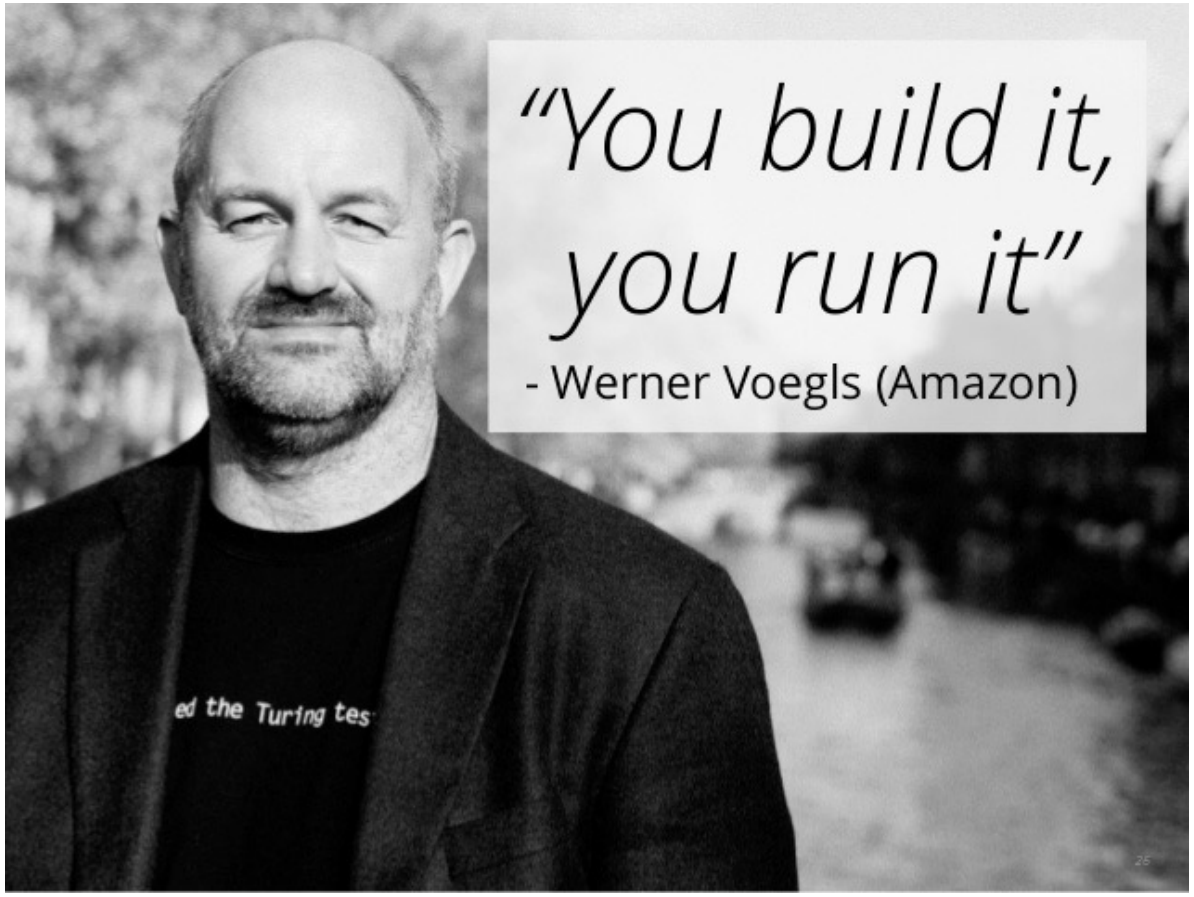



Jaeger

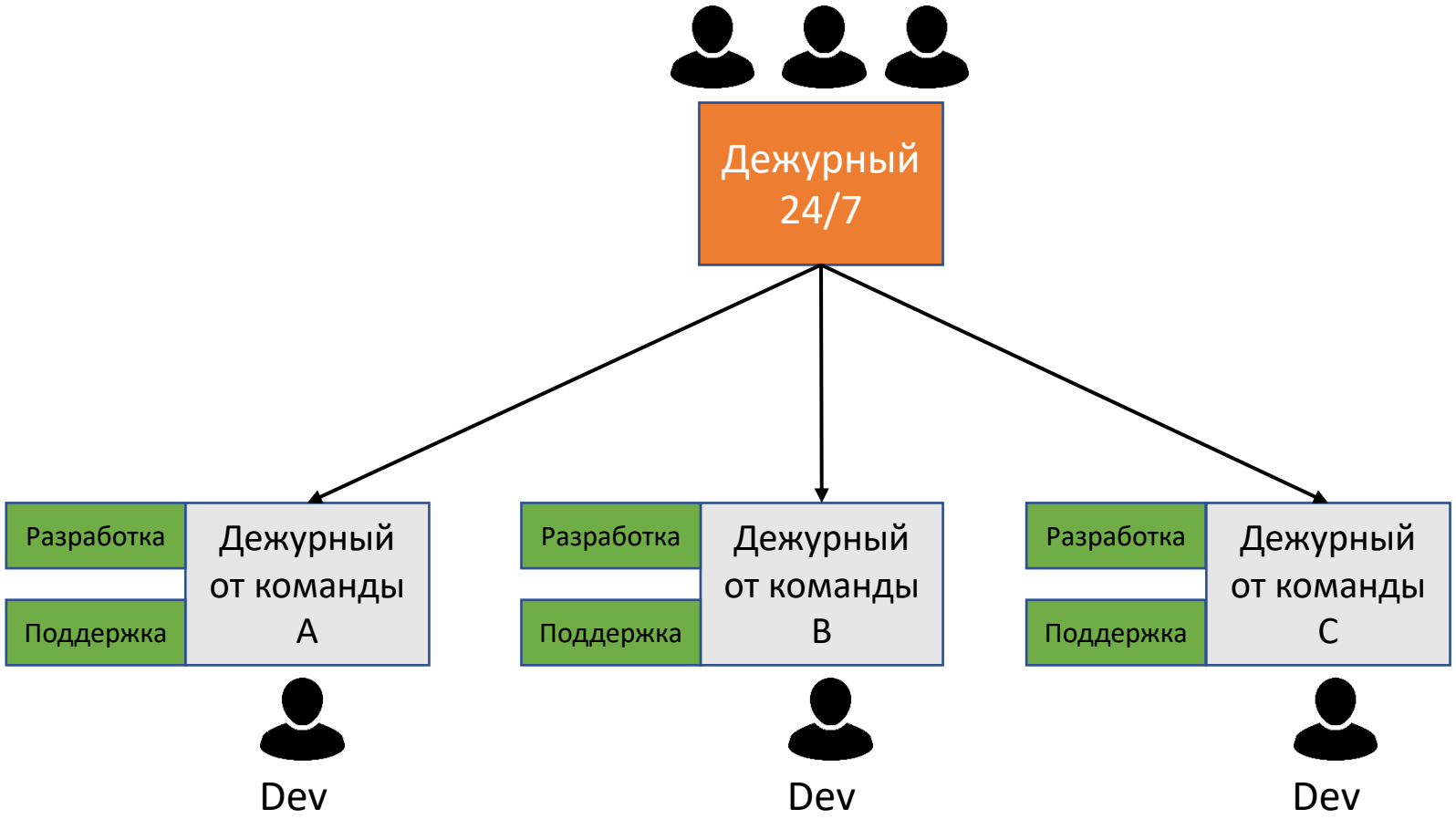

Zipkin

# Поддержка

# Подход "кто разработал тот и поддерживает"



# Реакция


Дежурный 24/7
Разработка
Поддержка
Дежурный от команды A
Dev
Разработка
Поддержка
Дежурный от команды B
Dev
Разработка
Поддержка
Дежурный от команды C
Dev

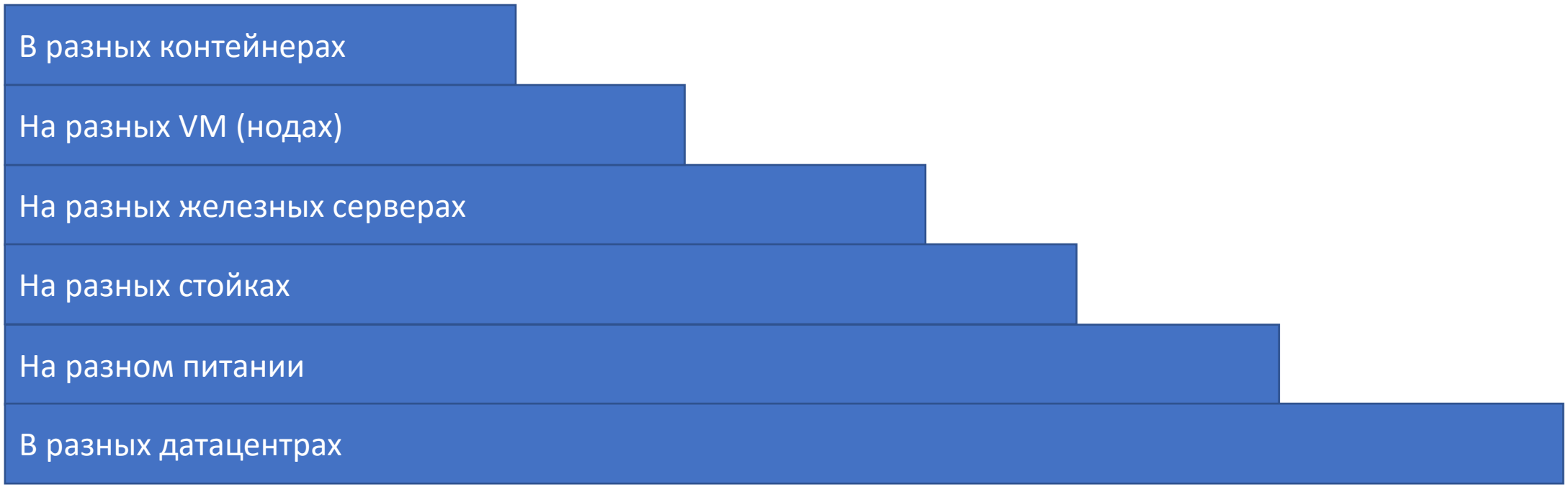
SKLADCHIK.ORG

# Отказоустойчивость

# Изолированность

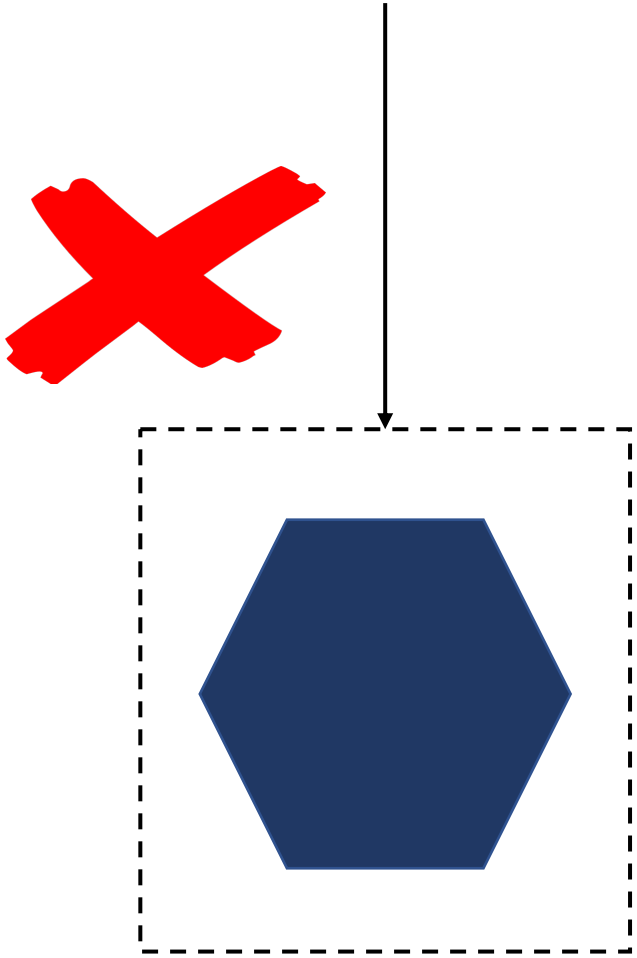
# Не класть все сервисы в 1 место, которое может отказать

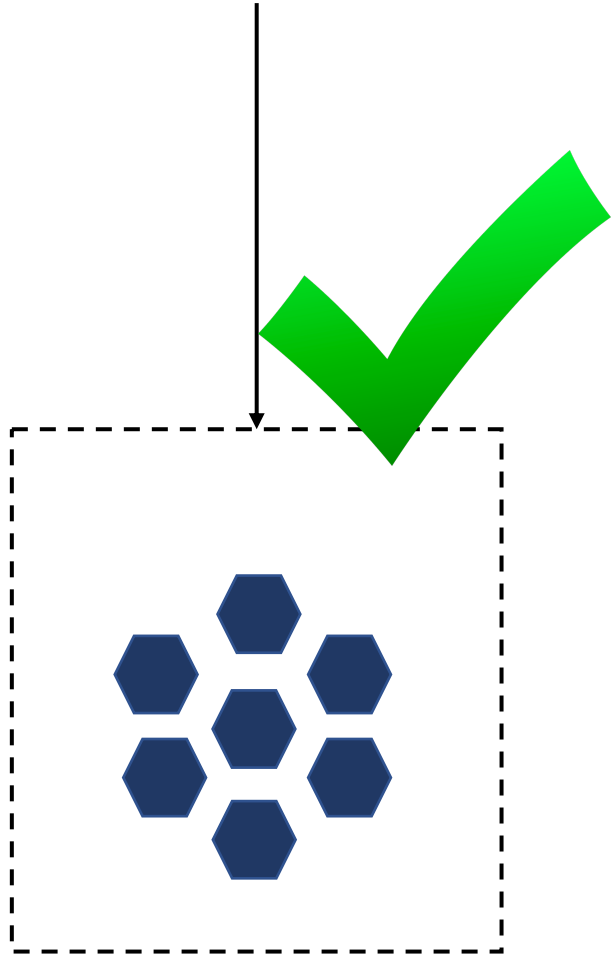
# Изолированность



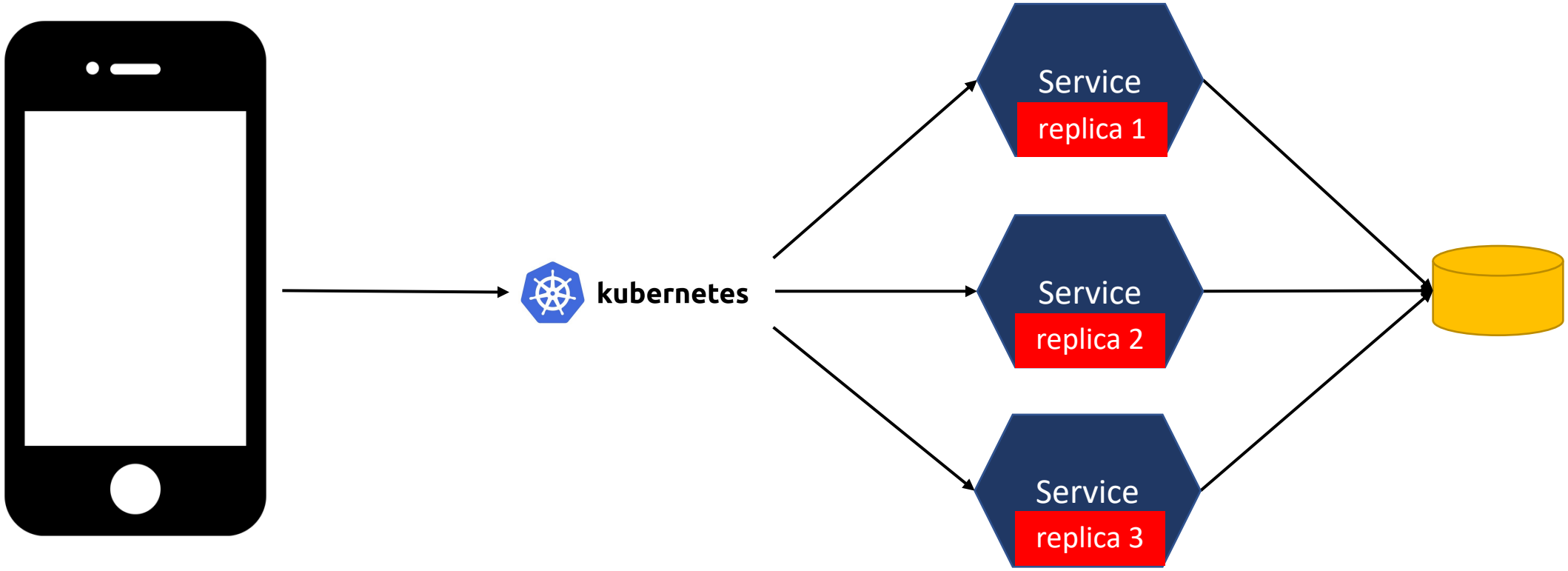
# Масштабирование

# Масштабирование

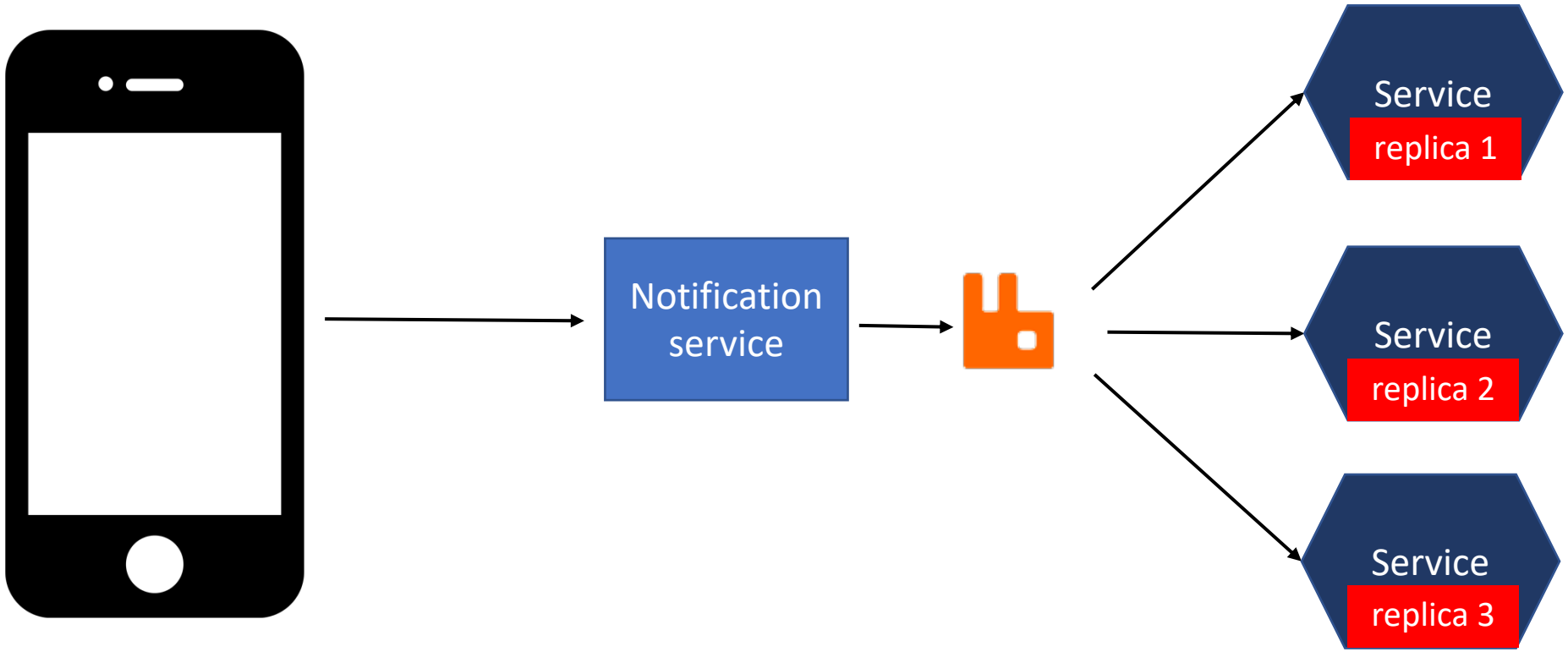




# Балансировка нагрузки RPC



# Балансировка нагрузки Event



# О чём узнали

- Метрики сервиса формирует сама команда, так как лучше всех знает особенности сервиса
- Метрики бывают не только технические, но и бизнес
- Трассировка крайне важна для быстрой диагностики проблем
- Увеличение количества экземпляров сервиса – основная стратегия масштабирования системы

# Спасибо за внимание!

Задавайте вопросы. Обо всем, что связано с текущей темой.

SKLADCHIK.ORG

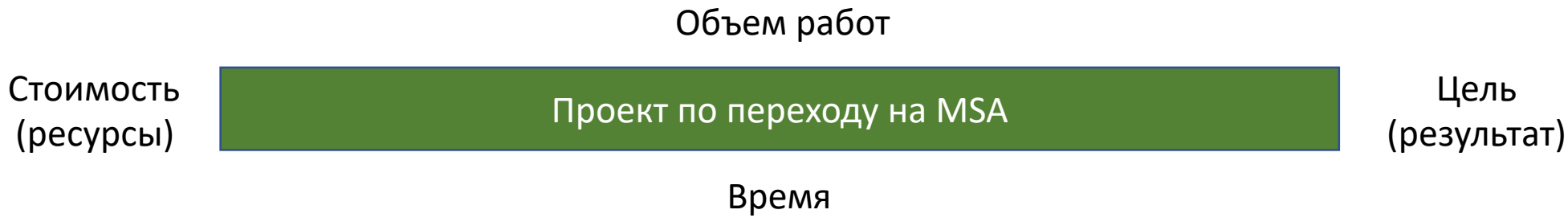
# Стратегии разбиения монолита

# Цели занятия

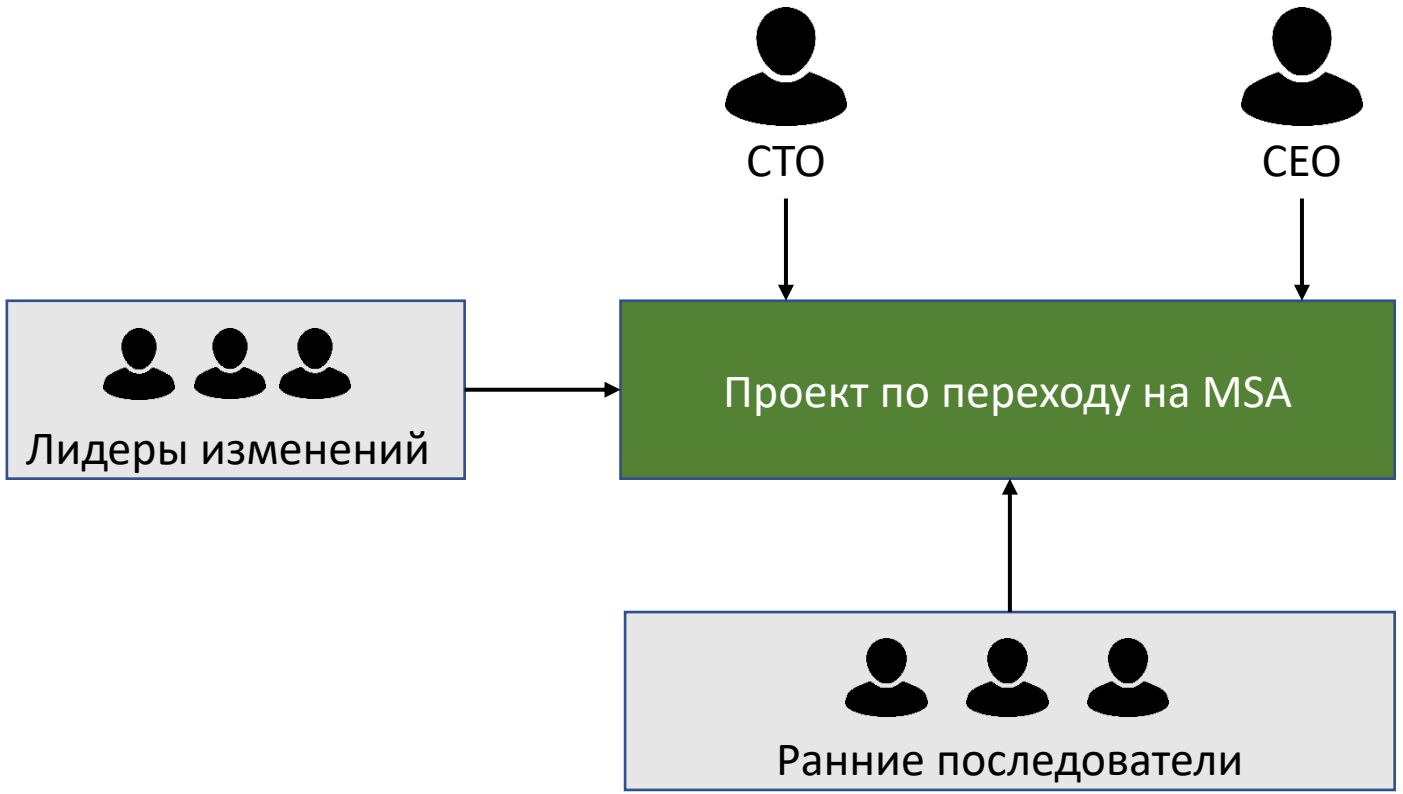
- Узнать три основные стратегии разбиения монолита
- Разобраться, из каких этапов состоит проект по разбиению монолита
- Познакомиться с основными техническими подходами

# Roadmap распила монолита

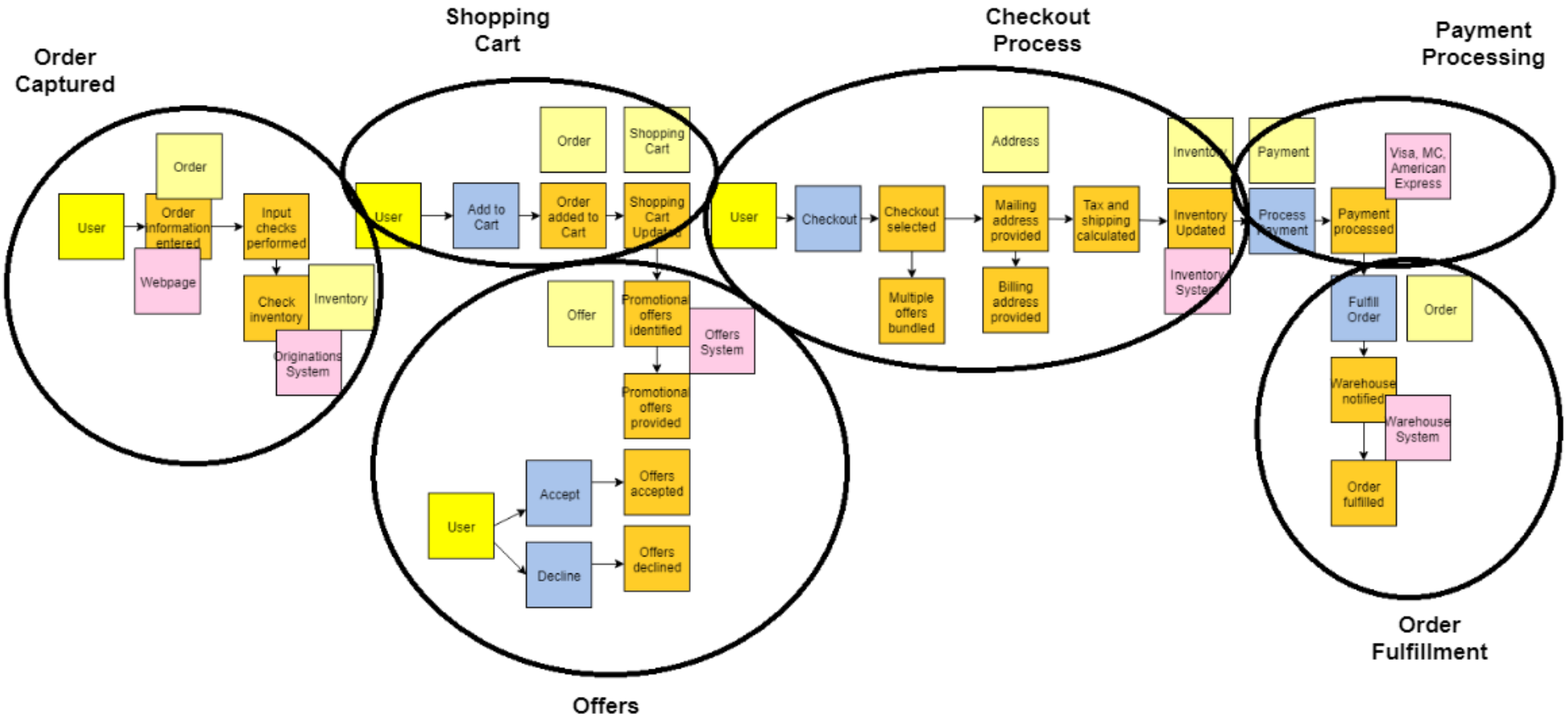
# Распил как проект



# 1 Шаг. Инициализация проекта

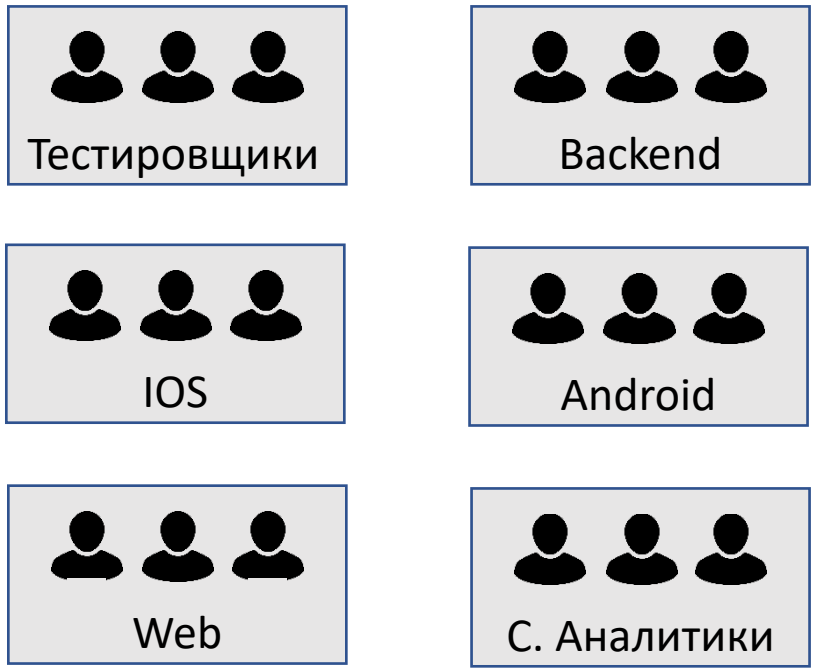


# 2 Шаг. Проведение Event Storming



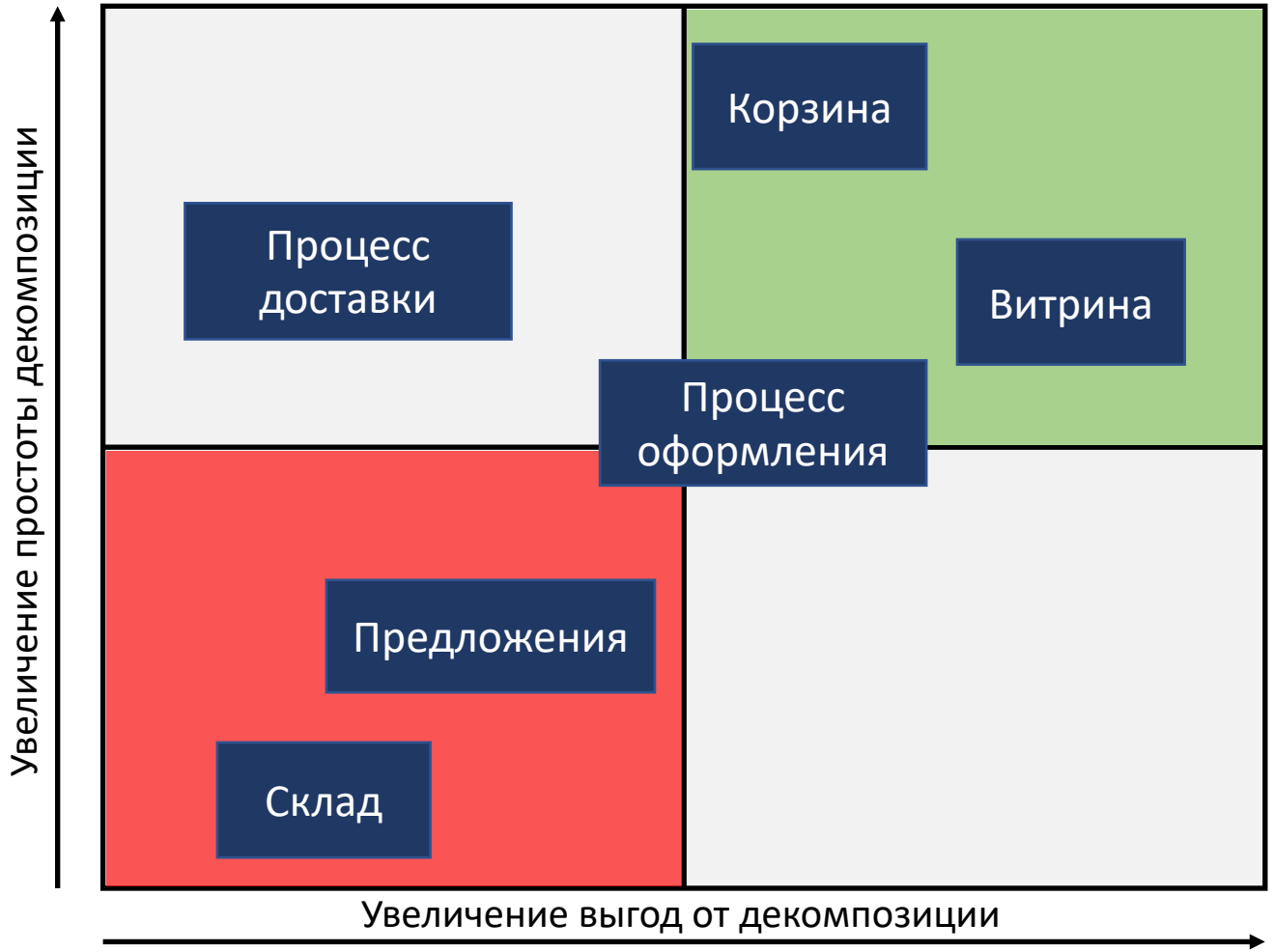
# 3 Шаг. Реорганизация команд

**Было**




**Стало**

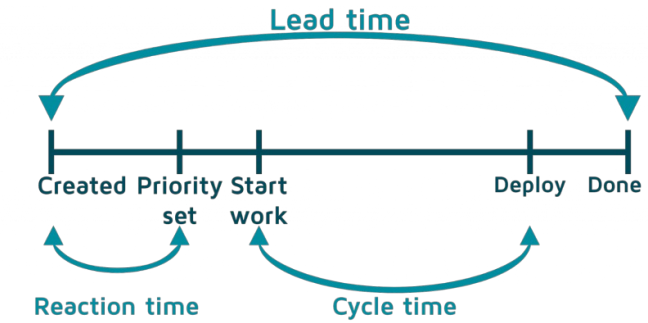
# 4 Шаг. Определение приоритетов

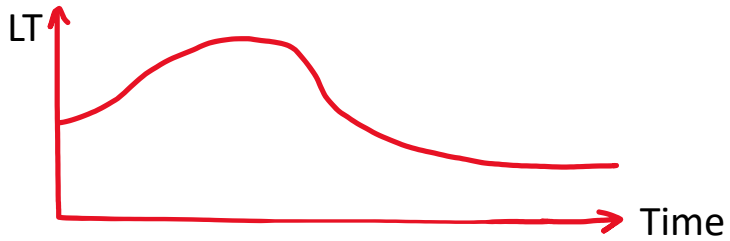


# 5 Шаг. Определение метрик успеха

**Количественные**

- Lead Time

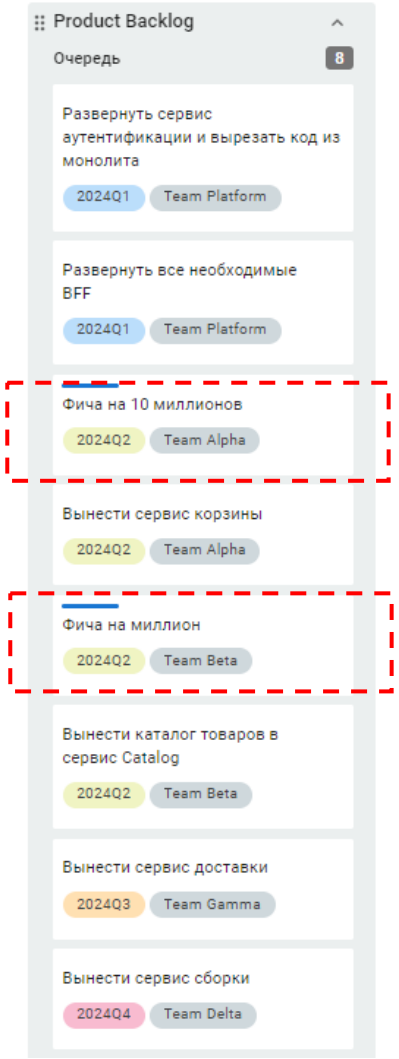


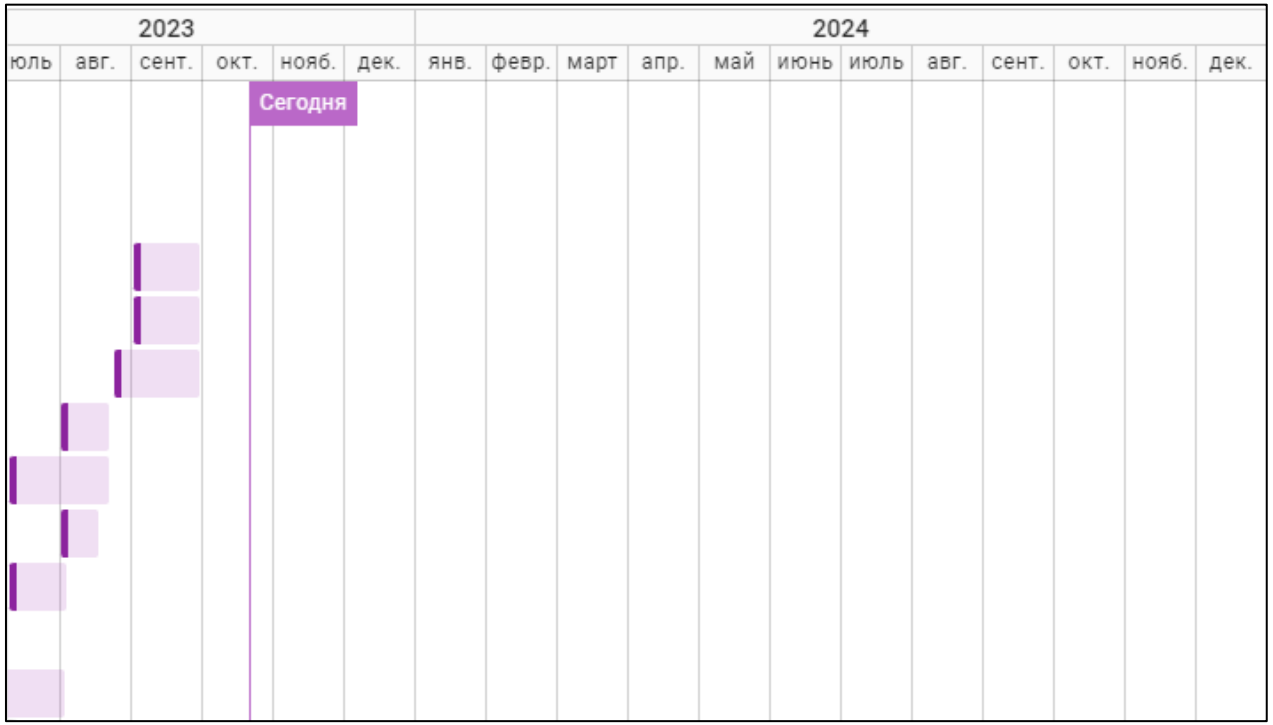




**Качественные**

- Happy Index

# 6 Шаг. Планирование

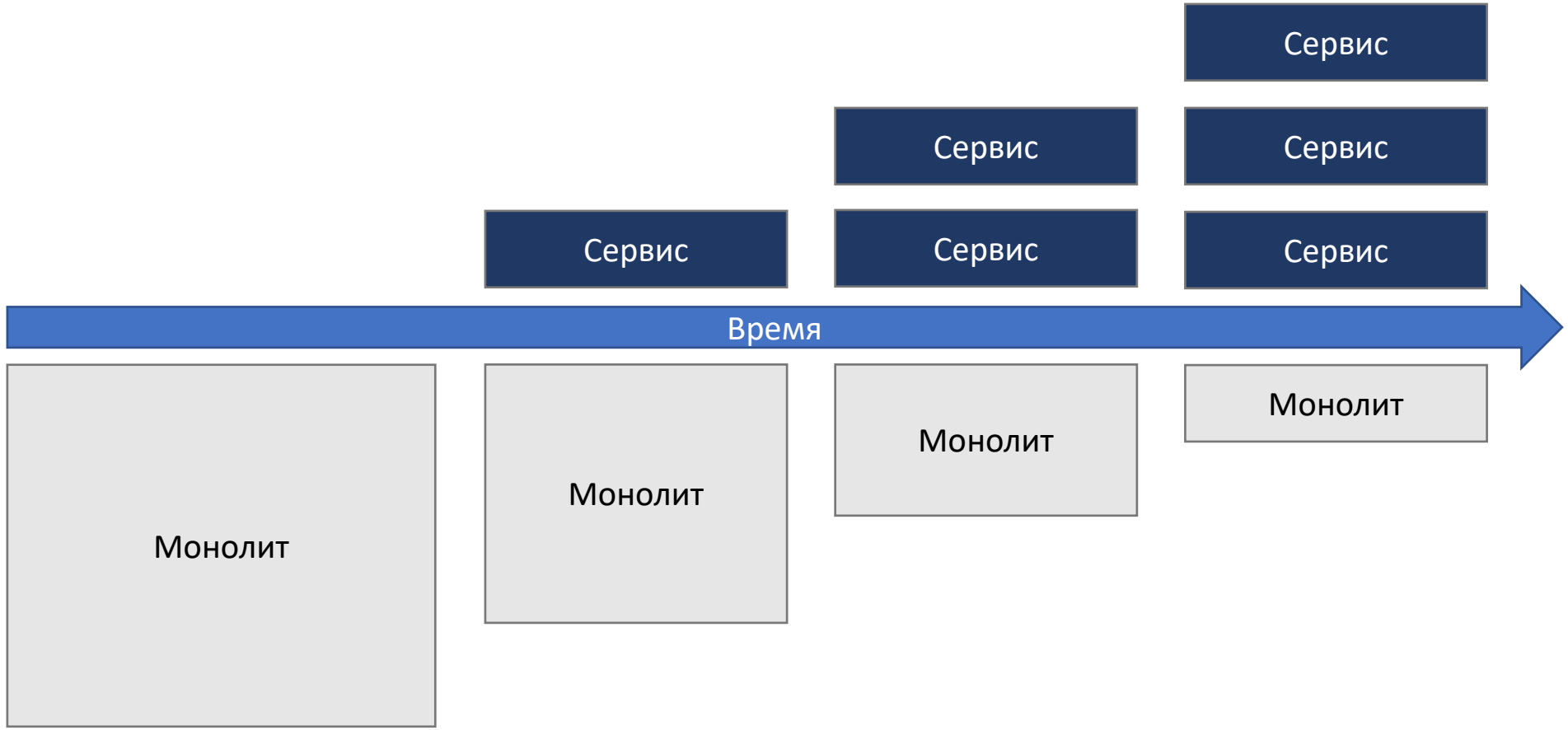


SKLADCHIK.ORG

# Основные стратегии декомпозиции

# Базовая идея



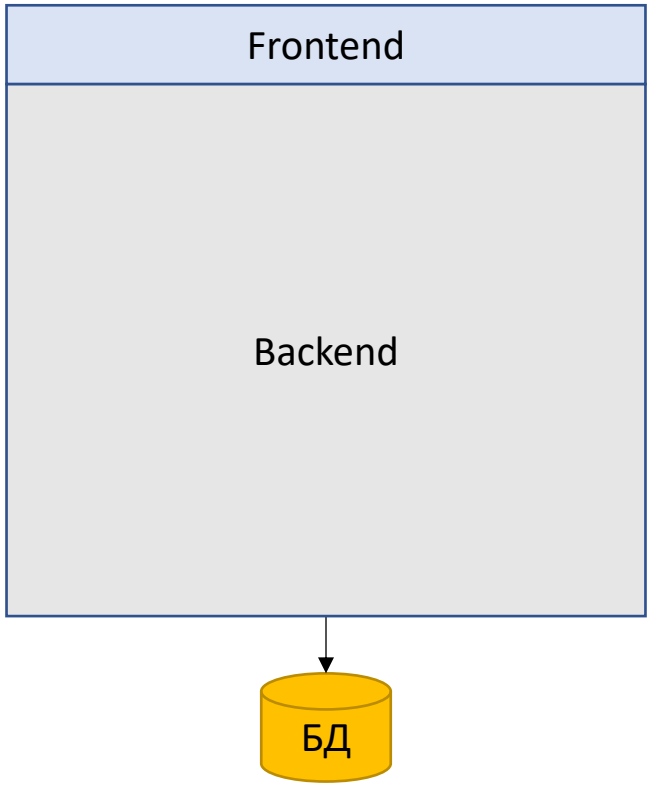
# Стратегии

1. Отделение Frontend от Backend

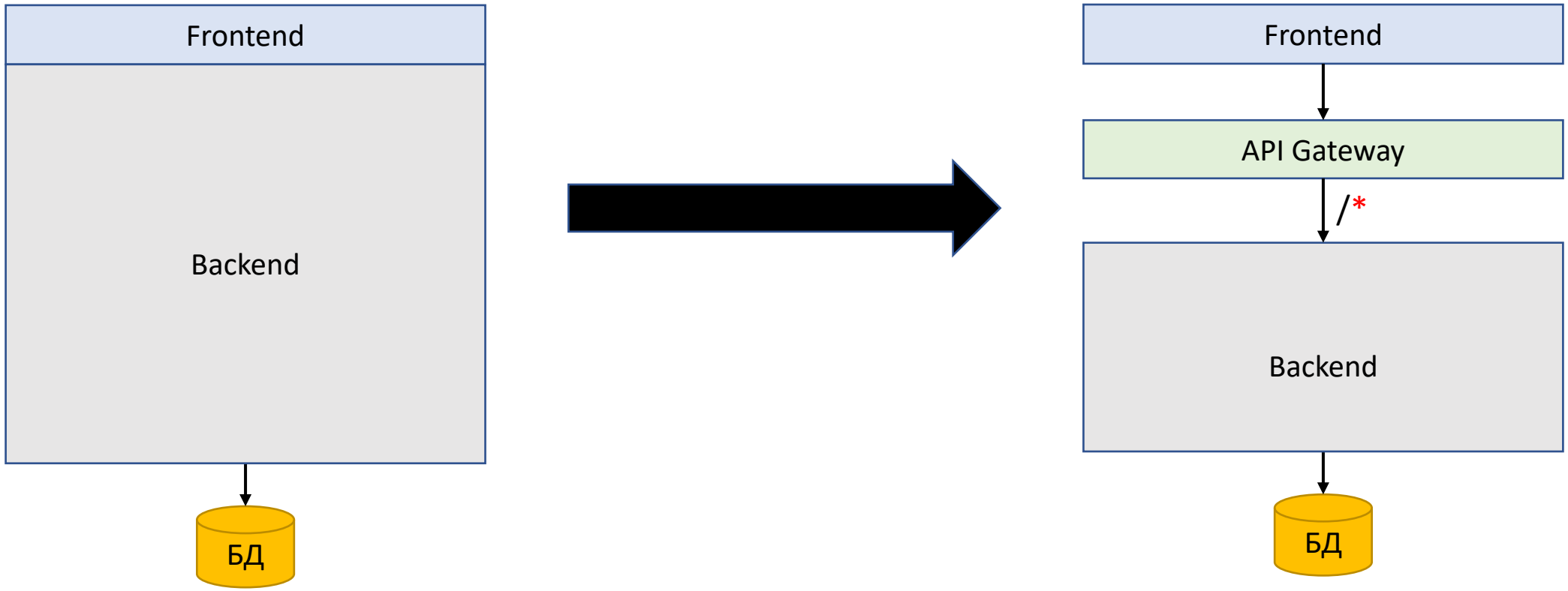
2. Реализация новых бизнес возможностей в виде сервисов

3. Разбиение монолита путем выделения сервисов

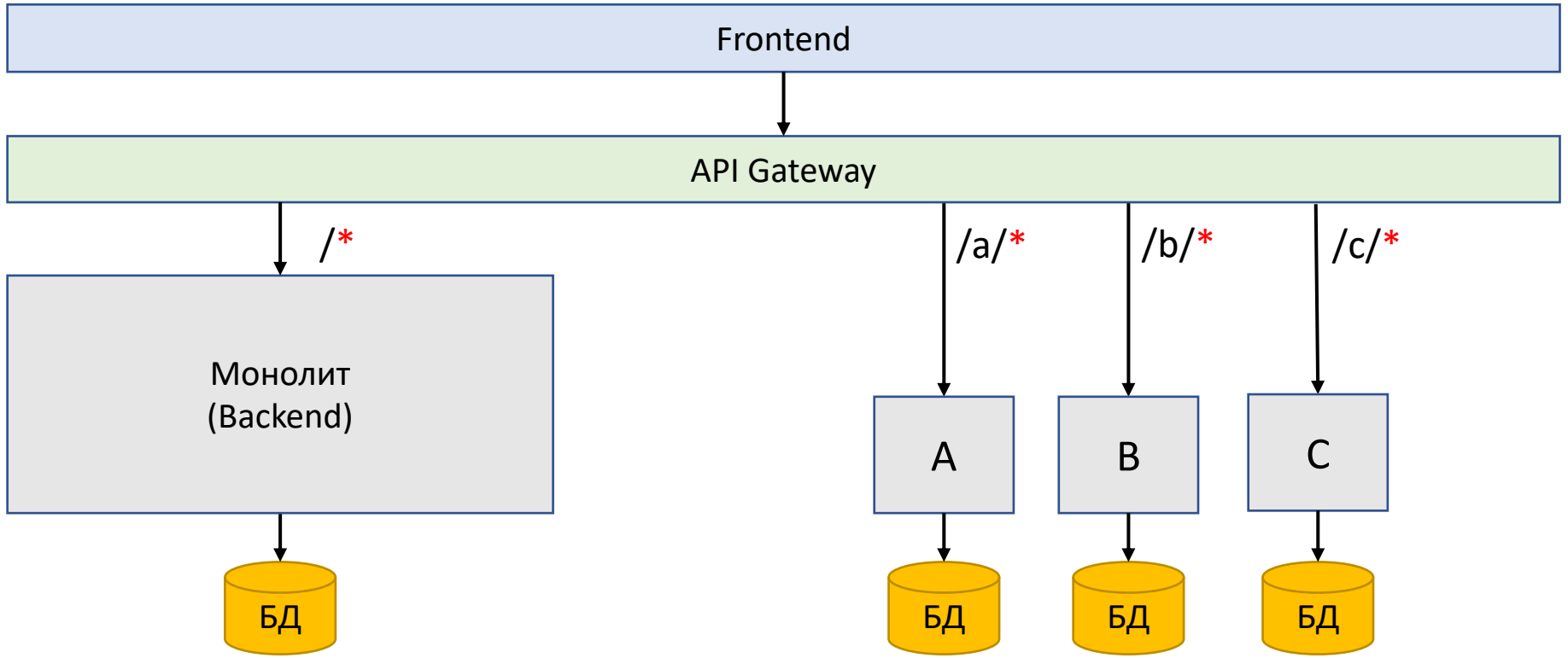
# 1. Разделение Frontend и Backend



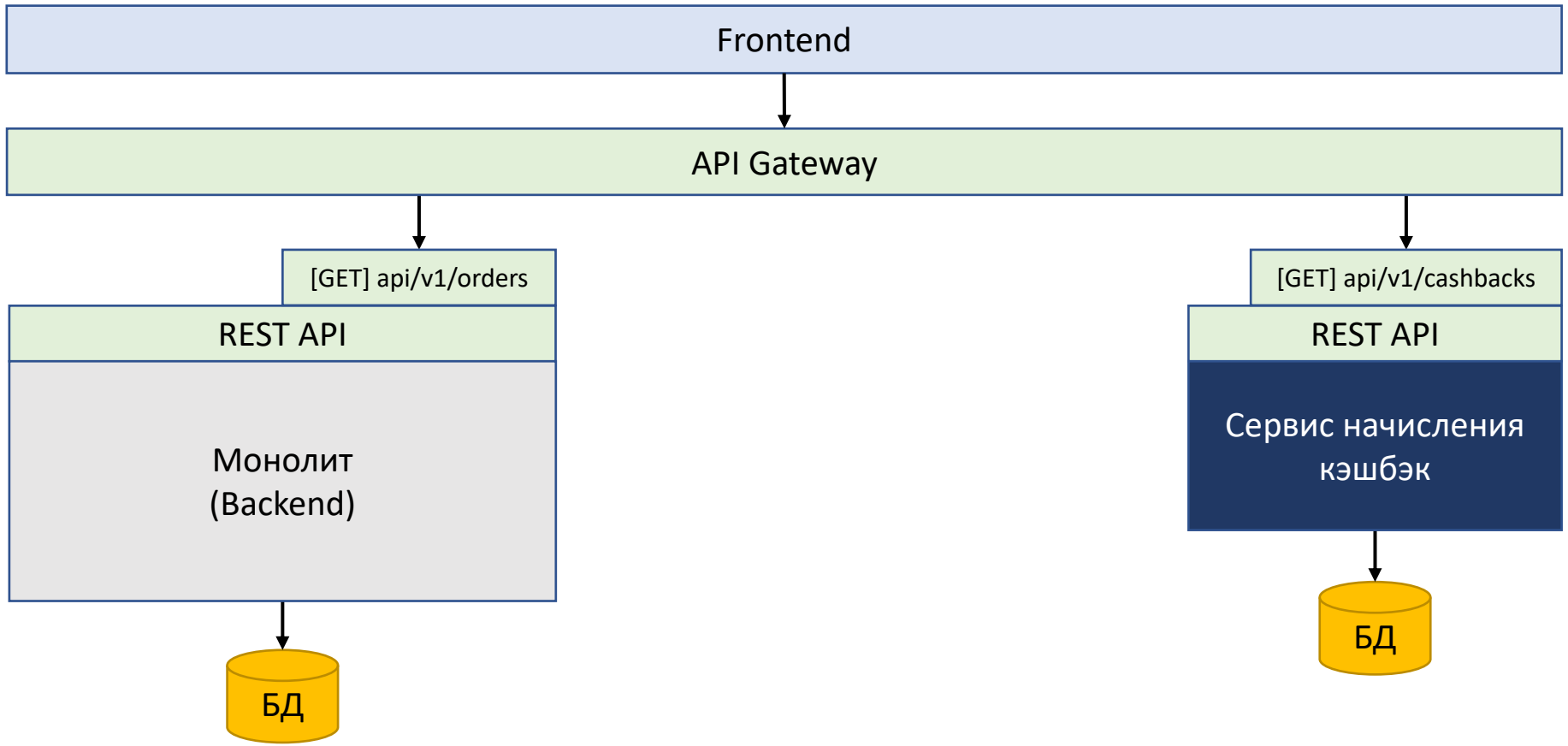
# 1. Разделение Frontend и Backend



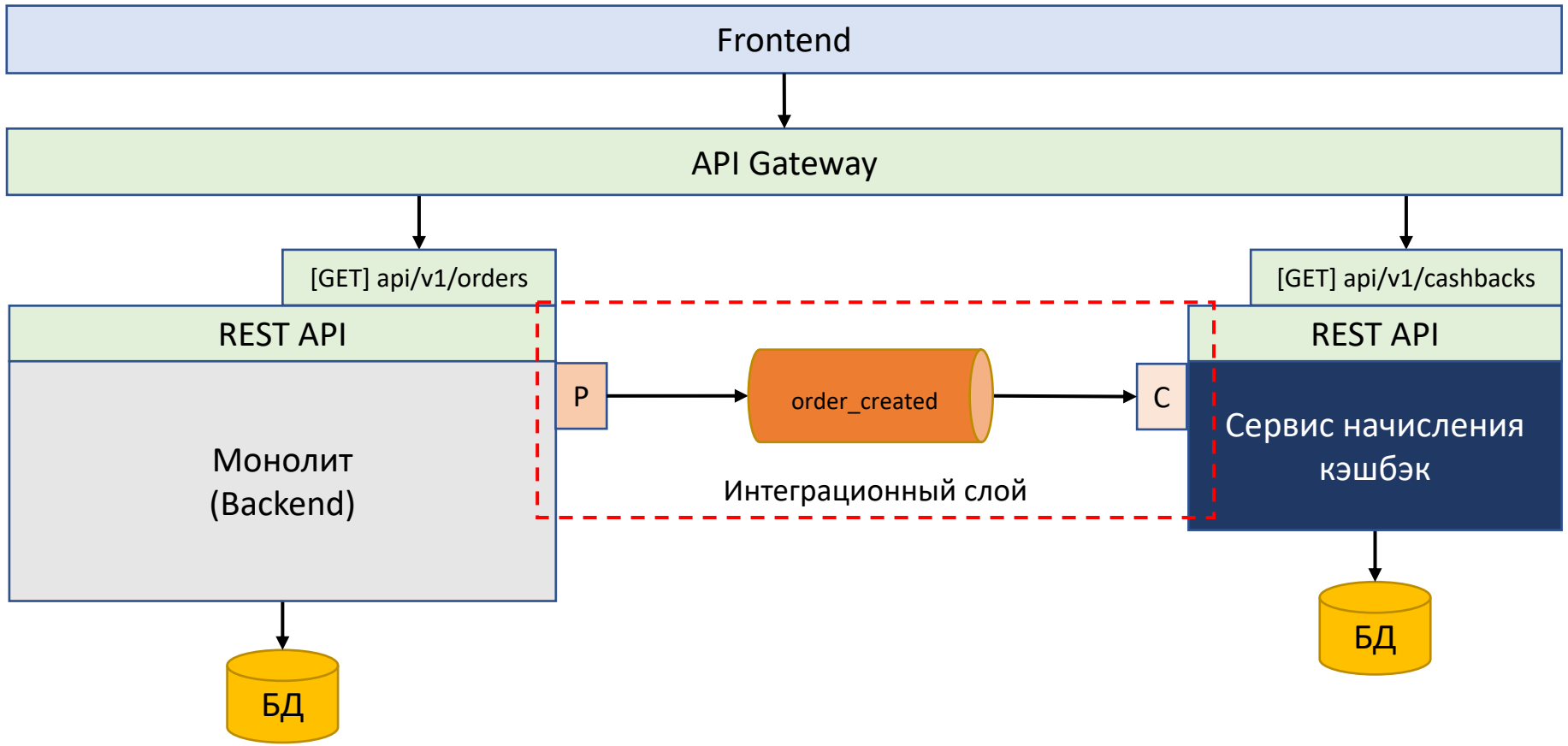
# 1. Разделение Frontend и Backend



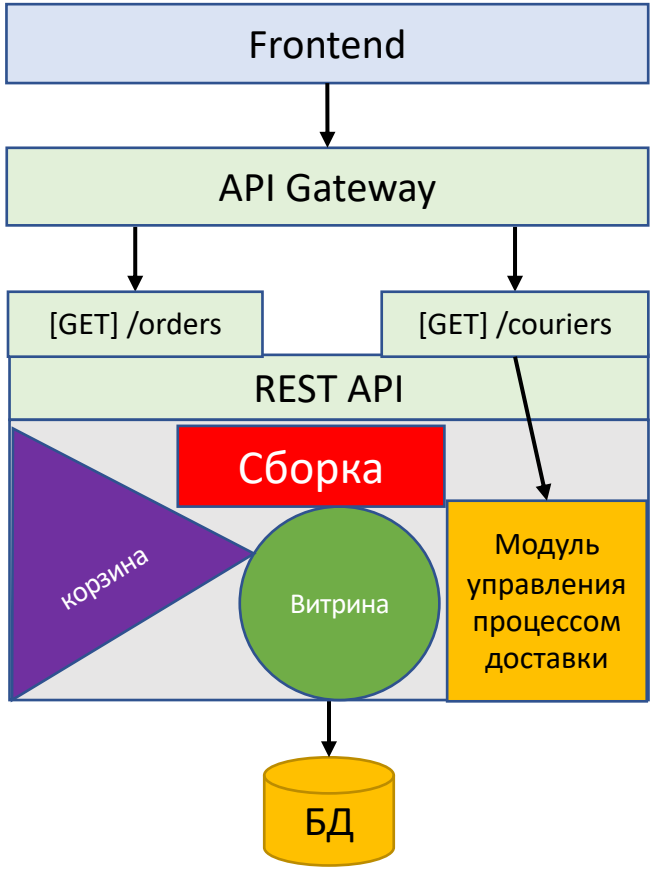
# 2. Реализация новых бизнес возможностей в виде сервисов



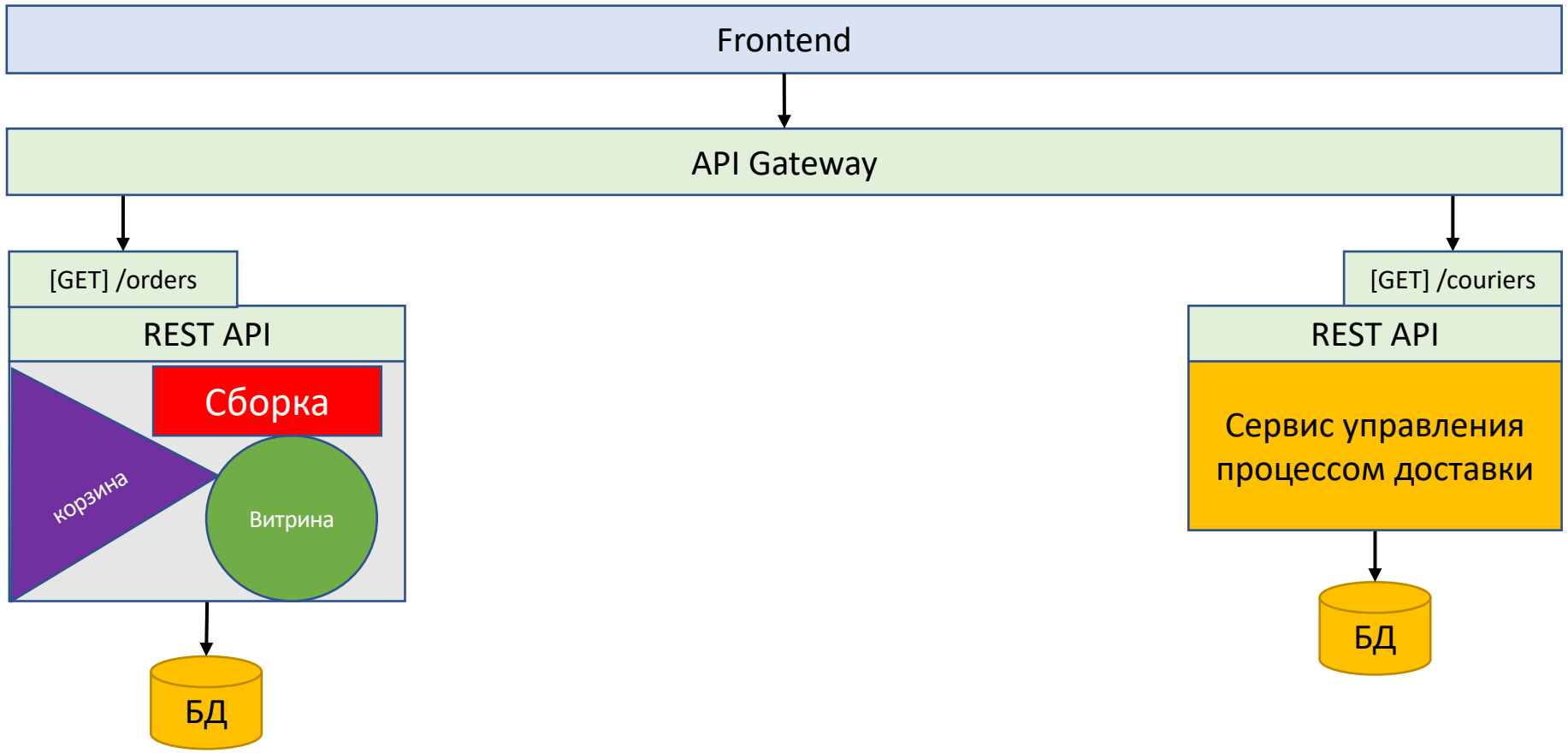
# 2. Реализация новых бизнес возможностей в виде сервисов



# 3. Разбиение монолита путем выделения сервисов

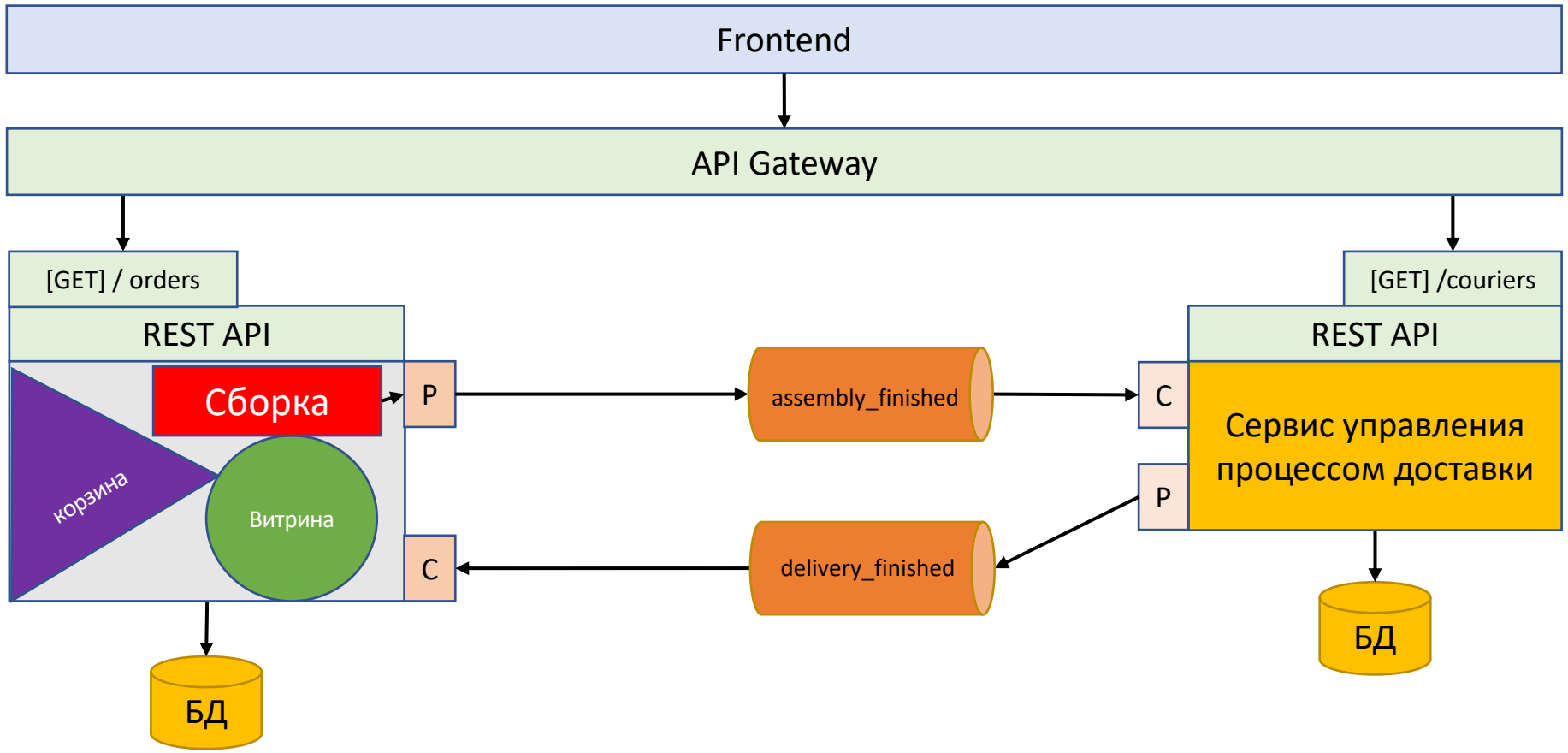


# 3. Разбиение монолита путем выделения сервисов



SKLADCHIK.ORG

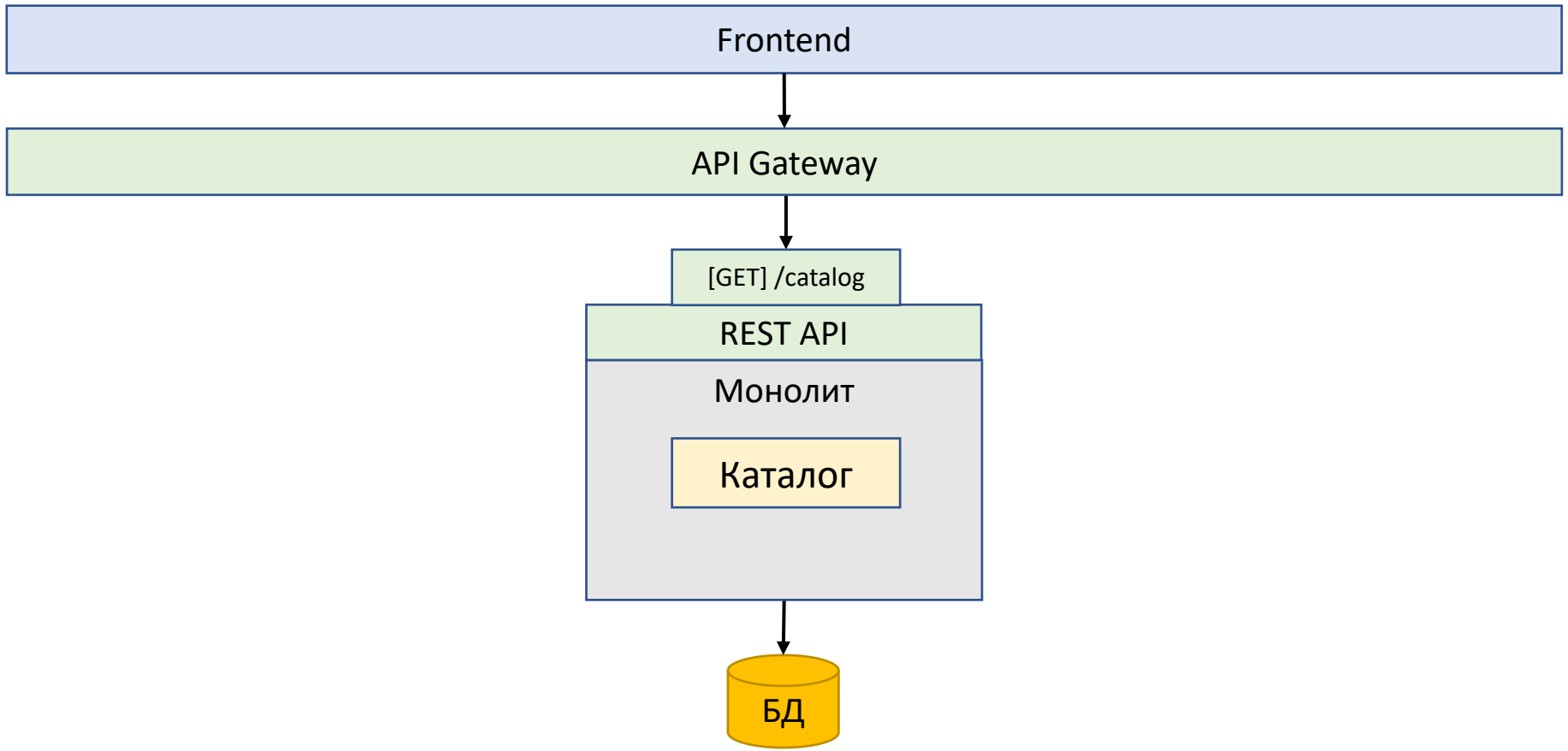
# 3. Разбиение монолита путем выделения сервисов



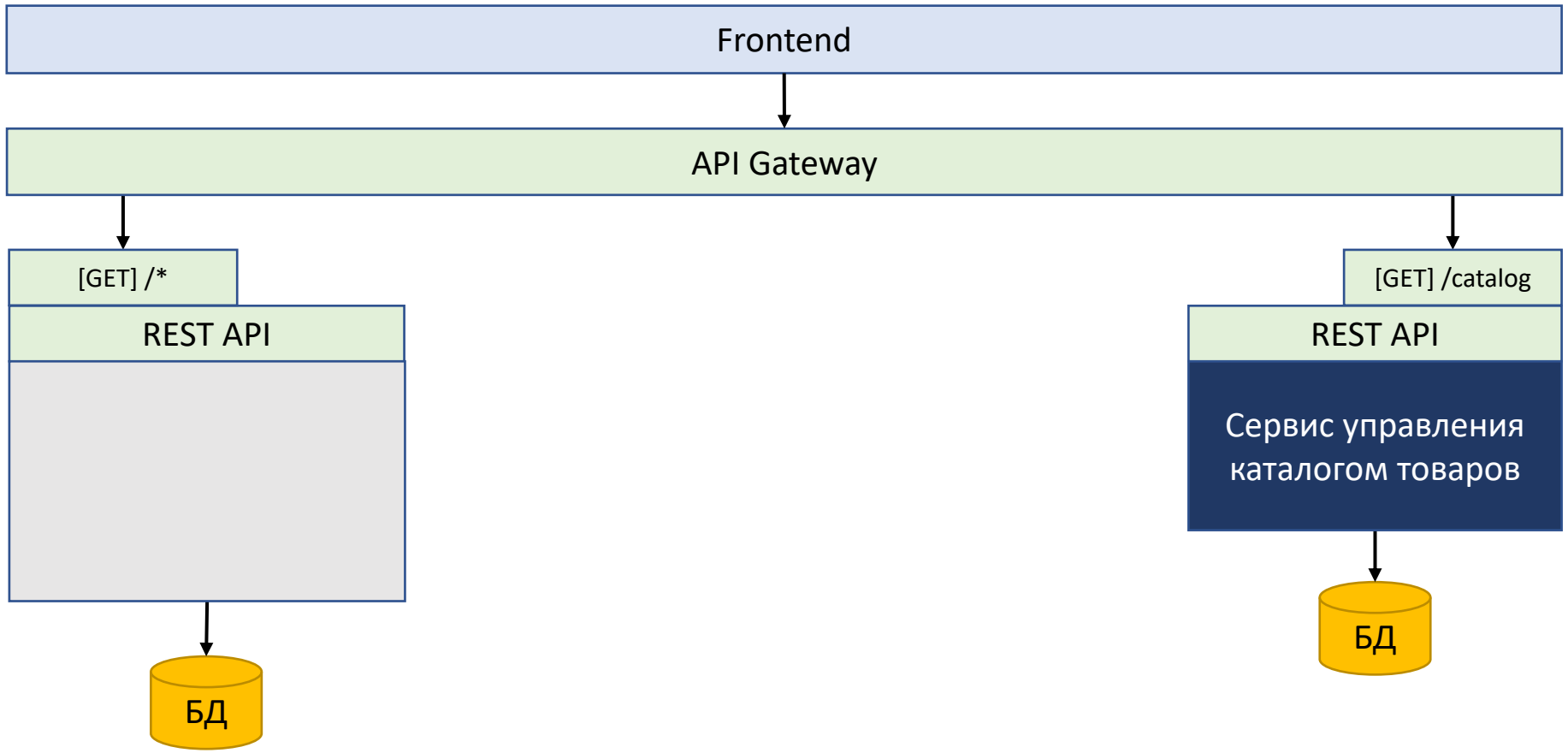
# Тактические паттерны

# Strangler application pattern

# Strangler application pattern

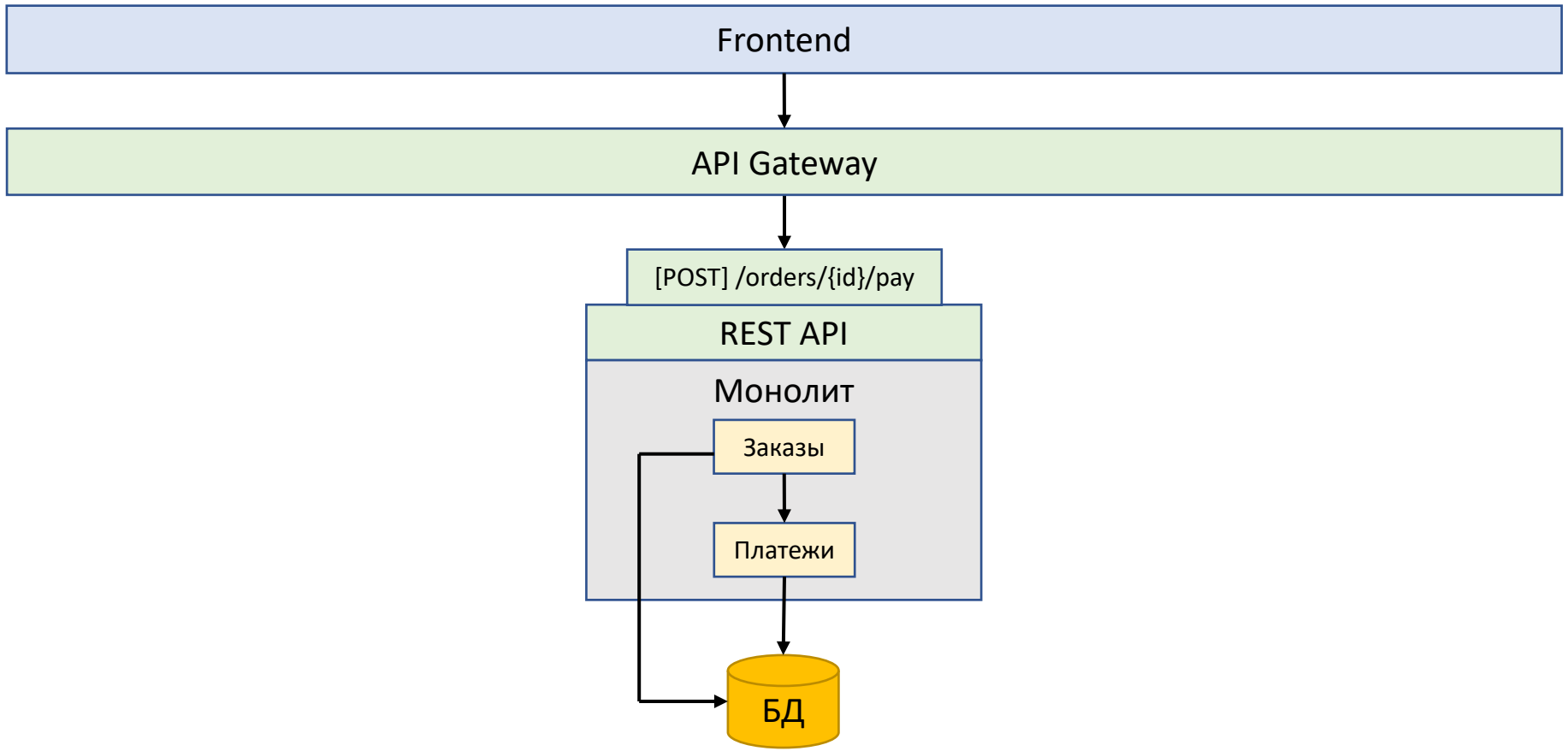


# Strangler application pattern

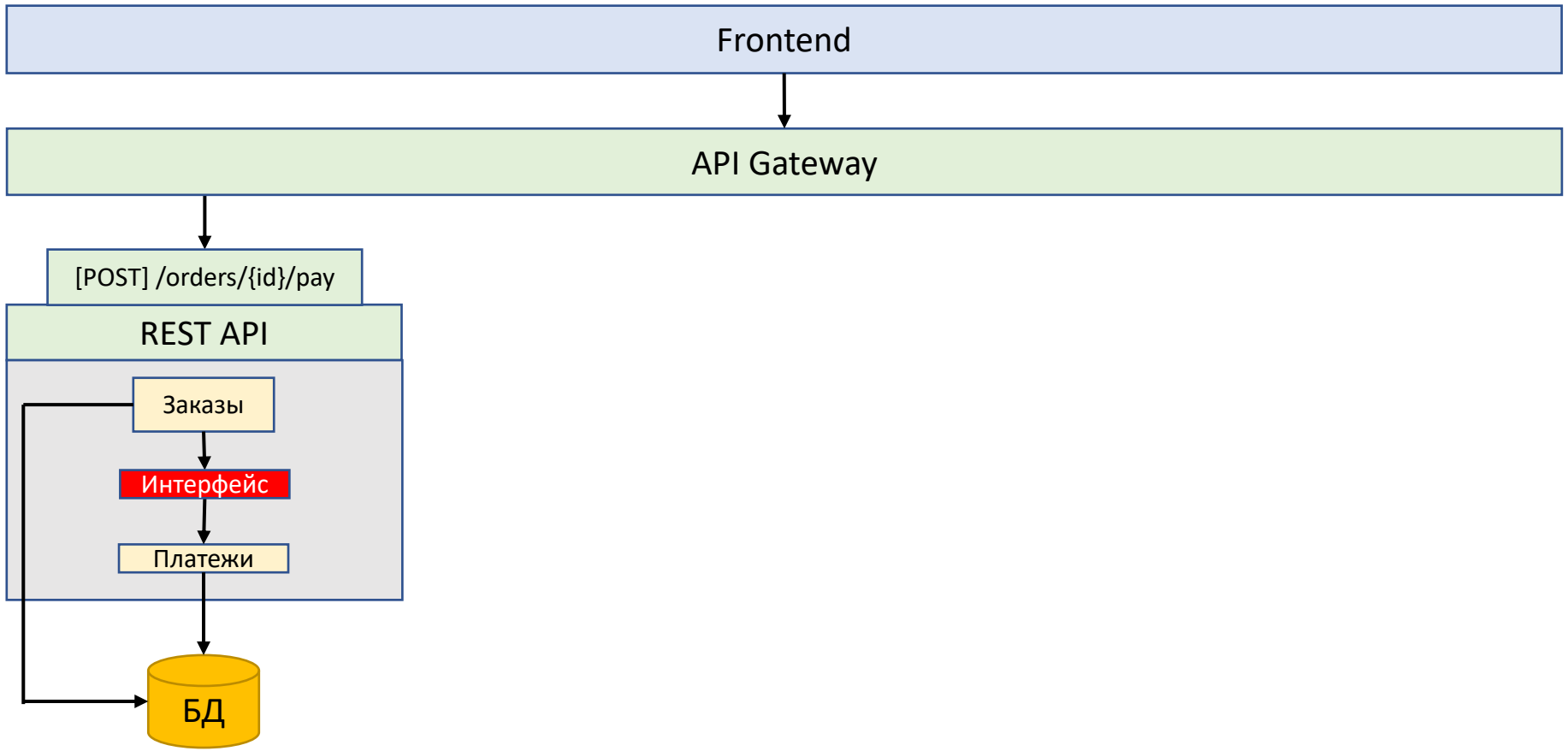


# Branch by abstraction pattern

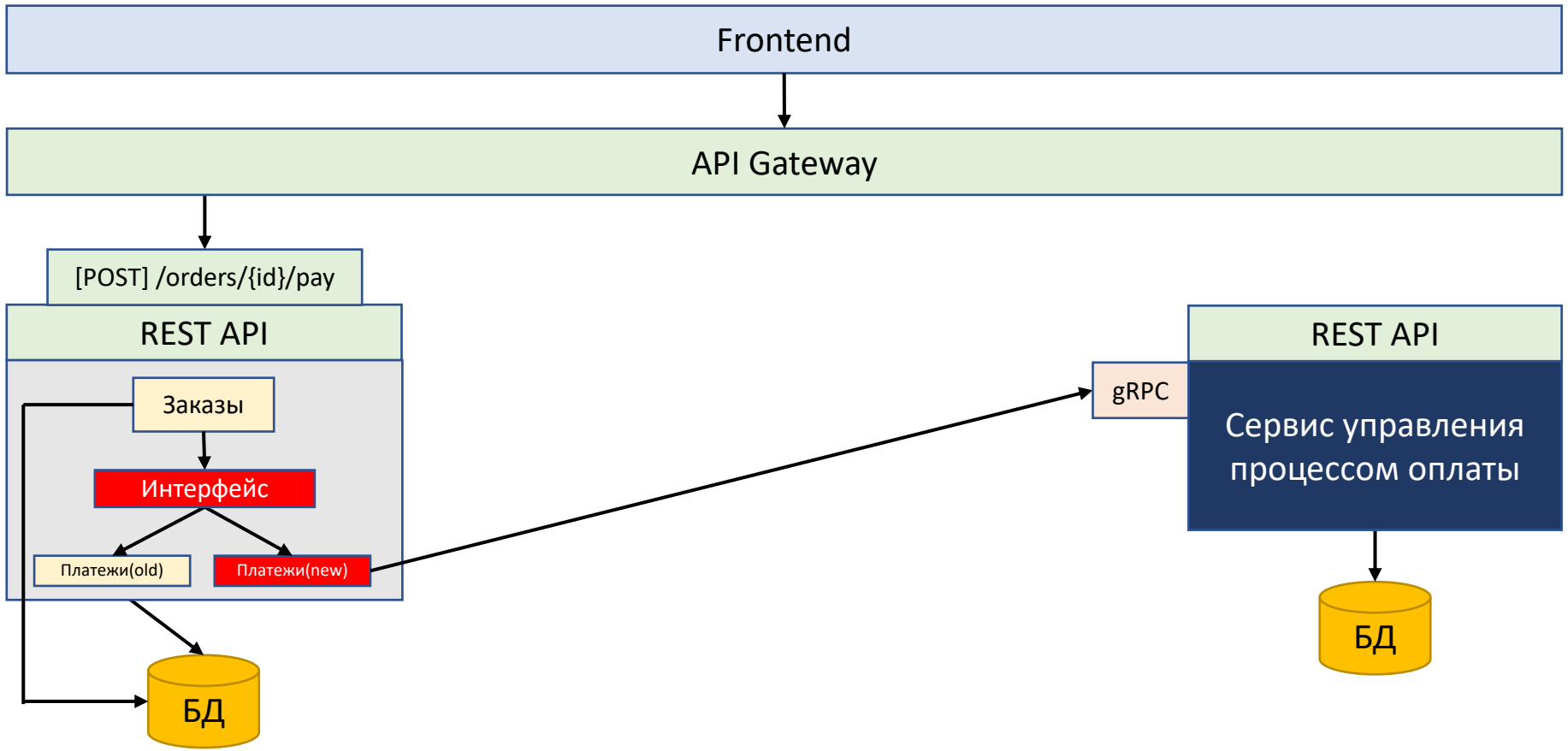
# Branch by abstraction pattern



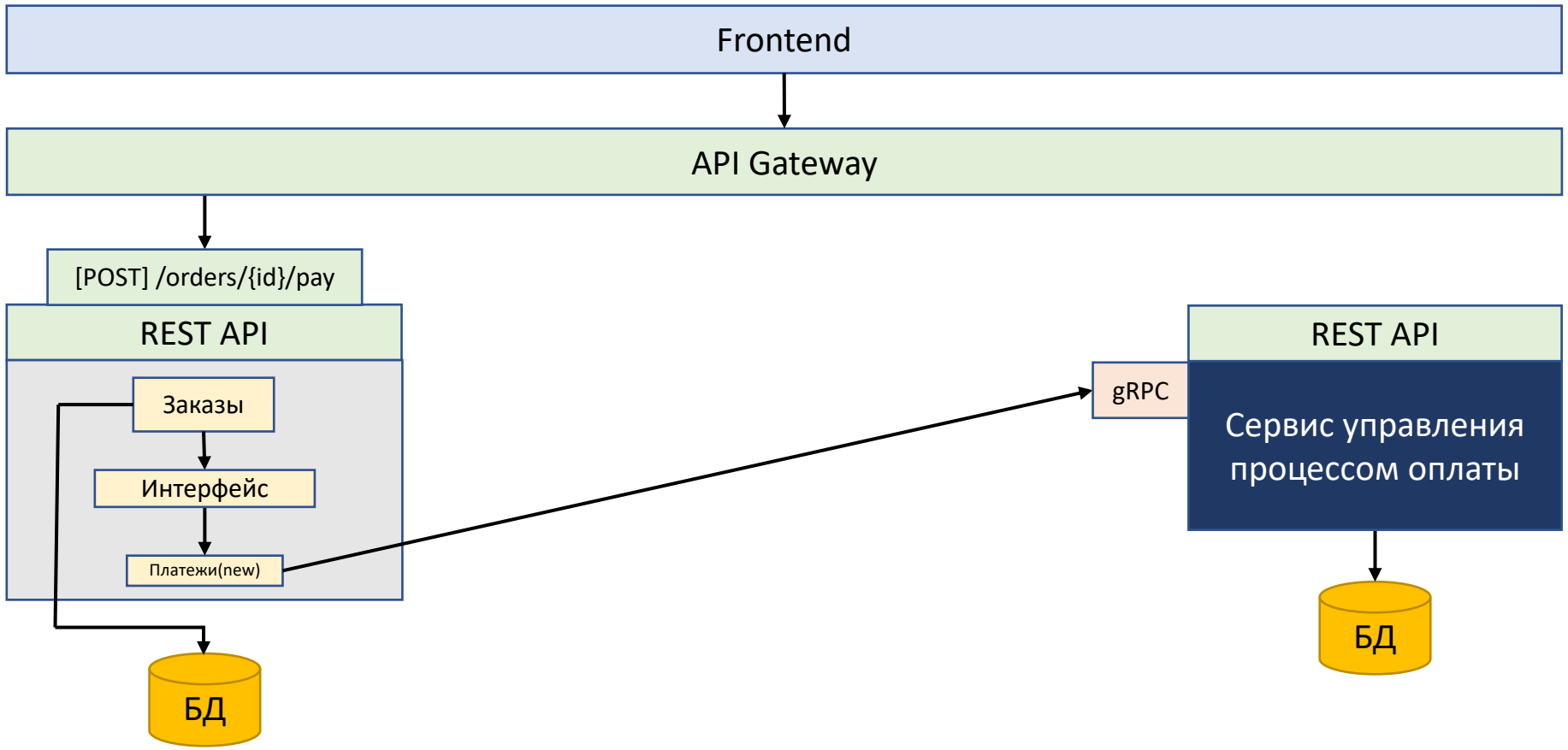
# Branch by abstraction pattern



# Branch by abstraction pattern



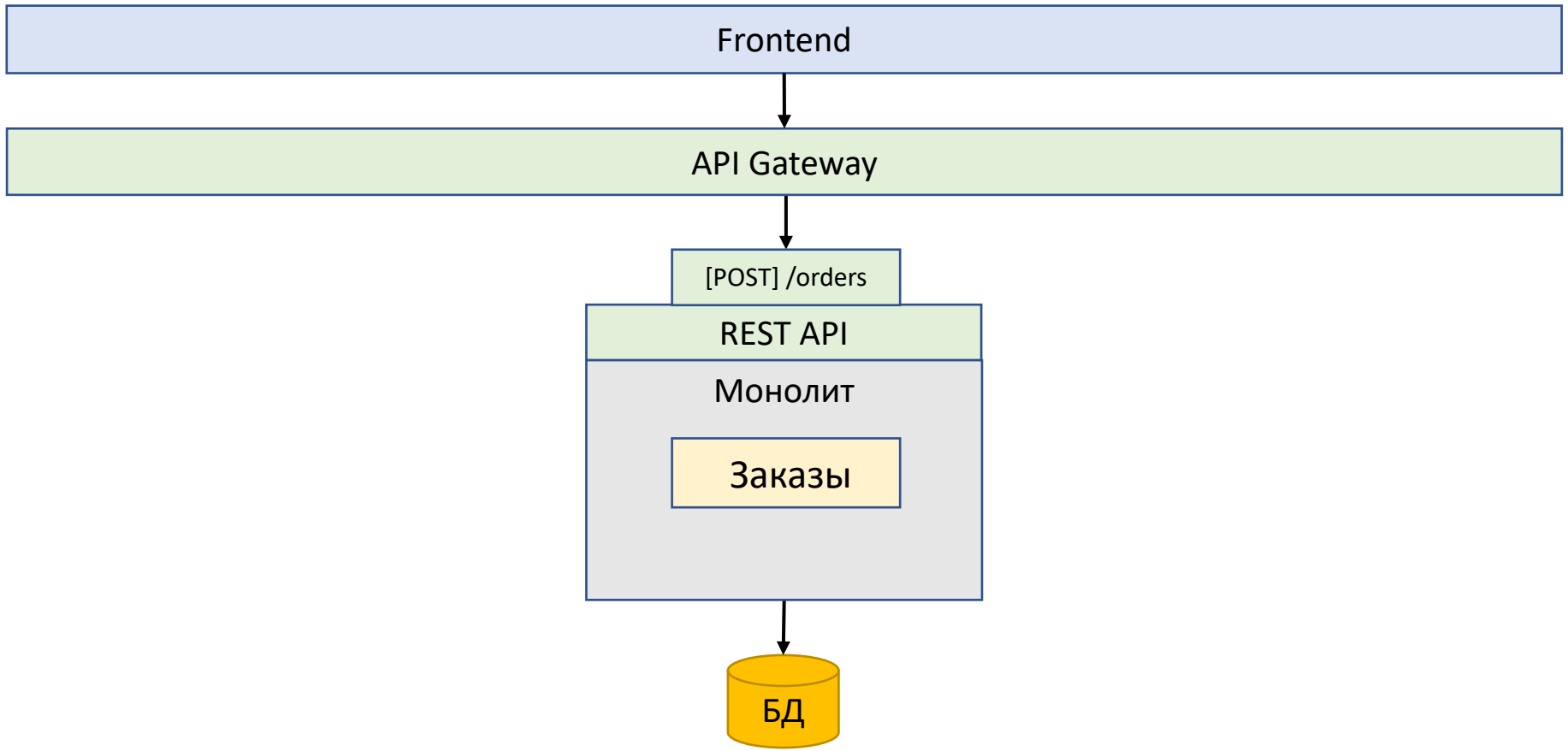
# Branch by abstraction pattern



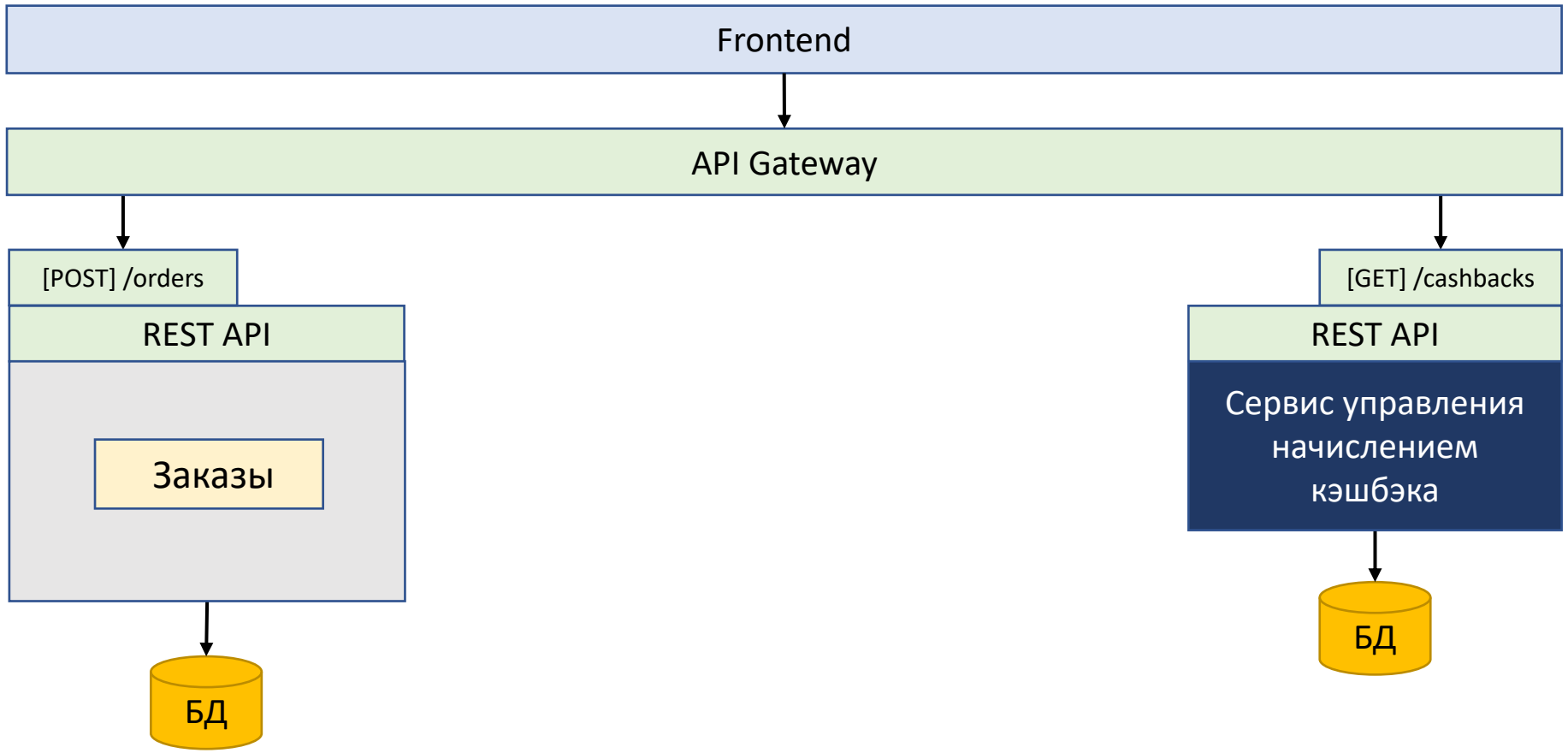
SKLADCHIK.ORG

# Change data capture pattern

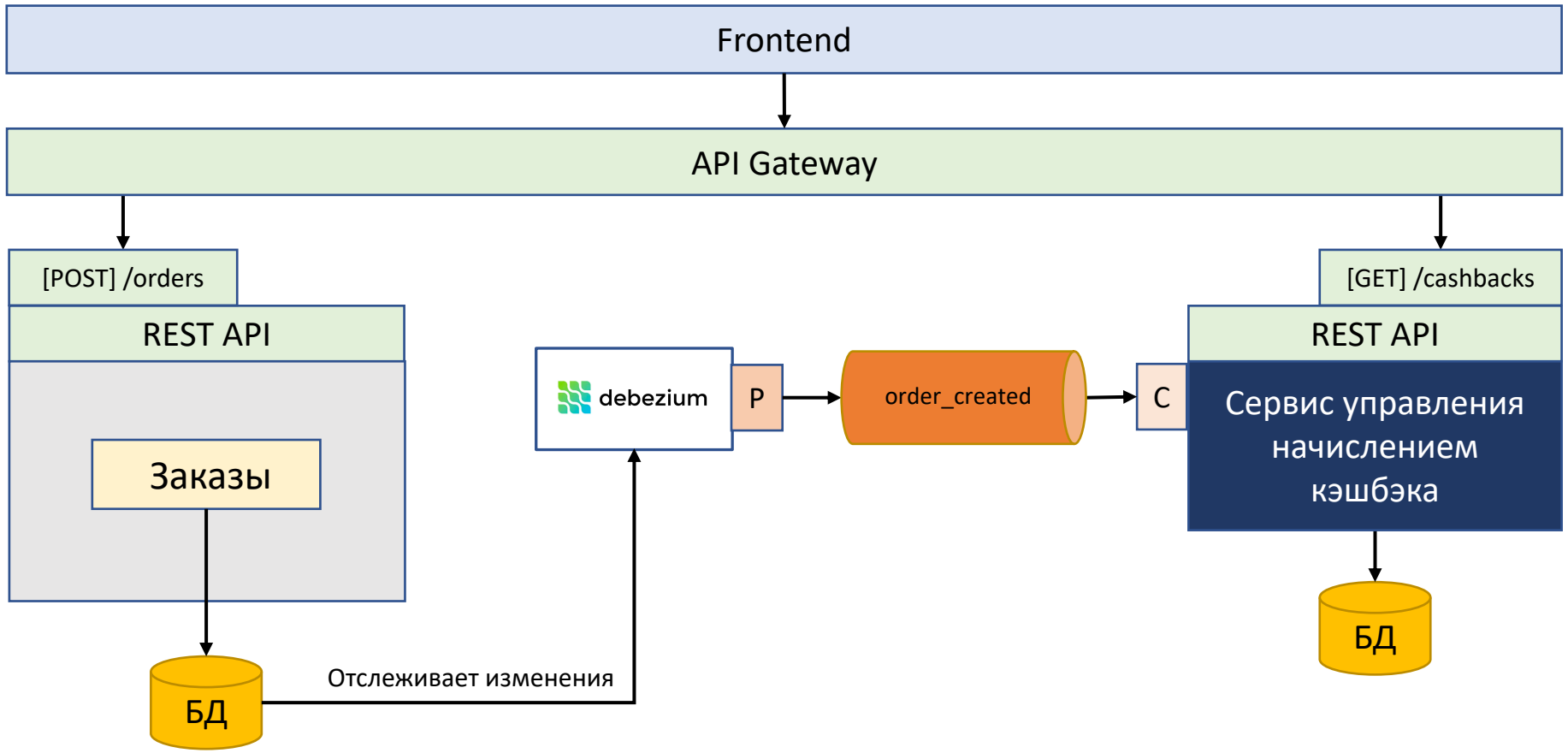
# Change data capture pattern



# Change data capture pattern

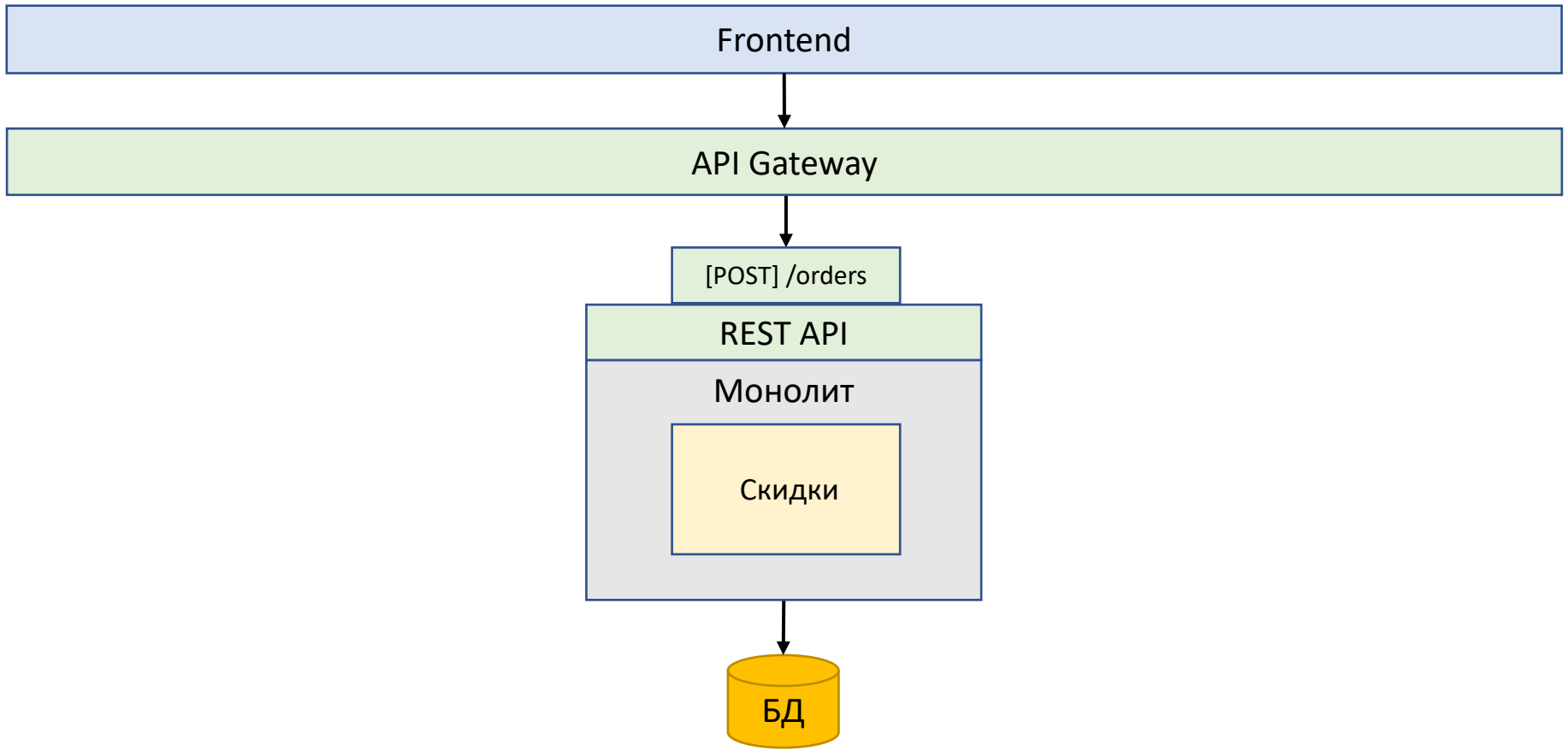


# Change data capture pattern

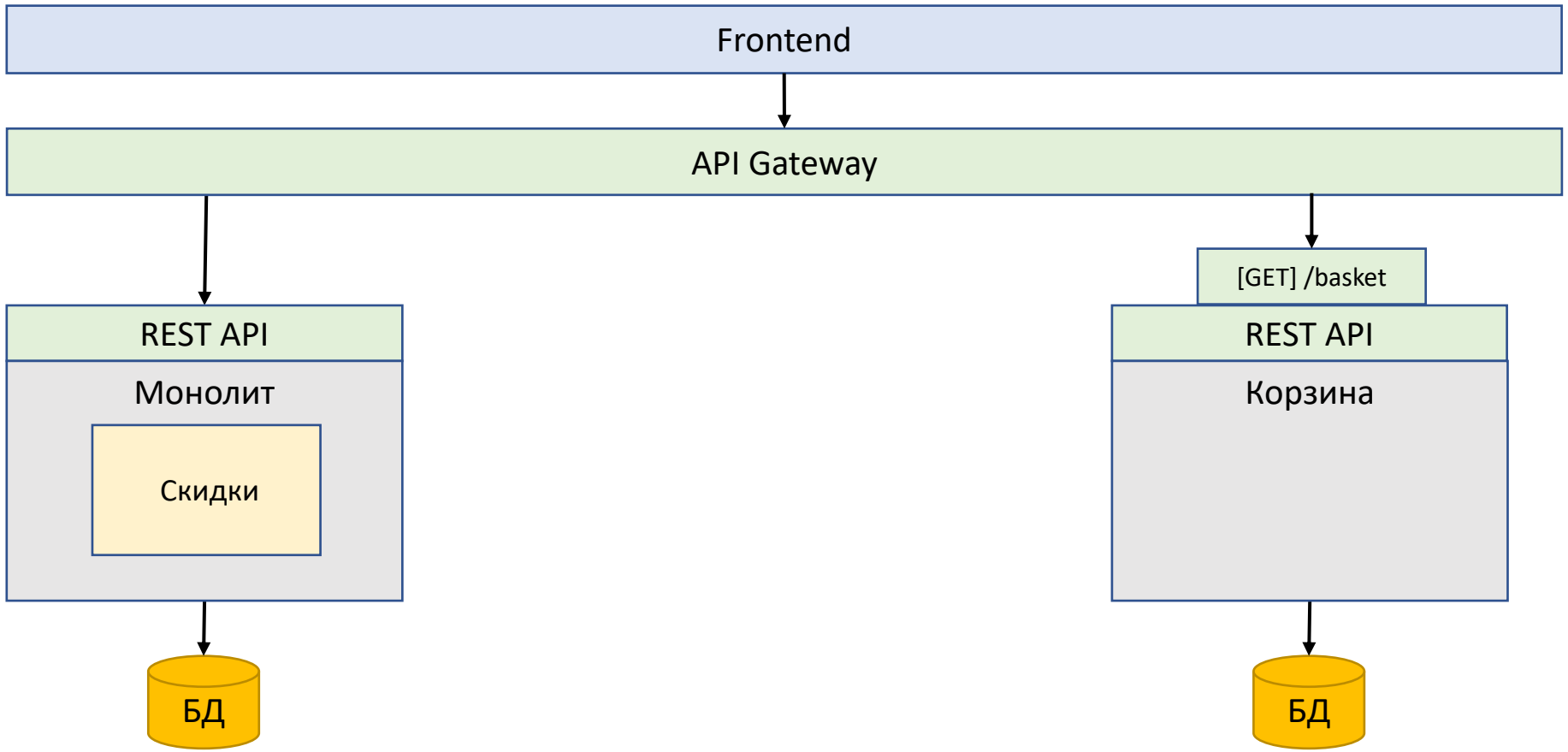


# Доступ к Aggregate через API

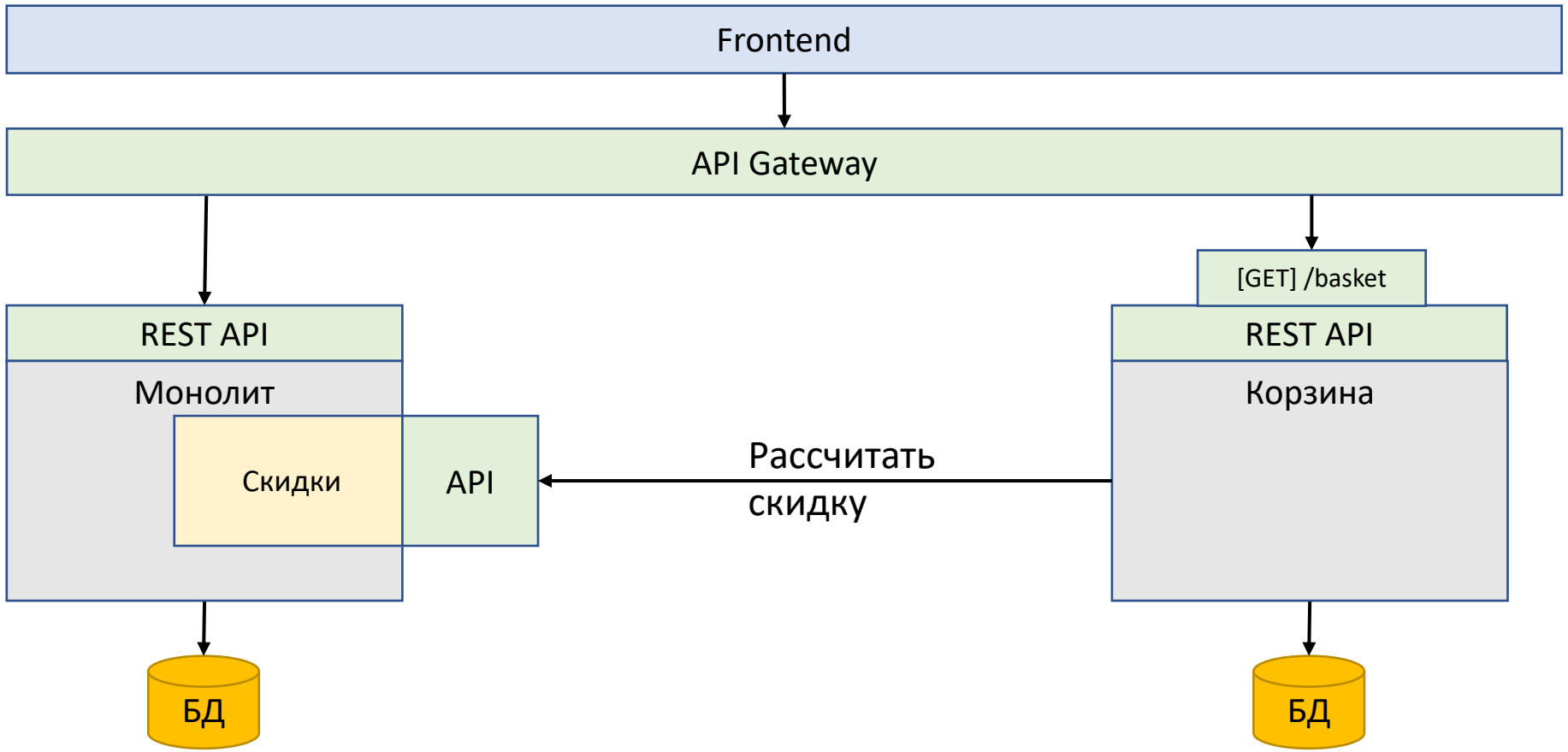
# Доступ к Aggregate через API



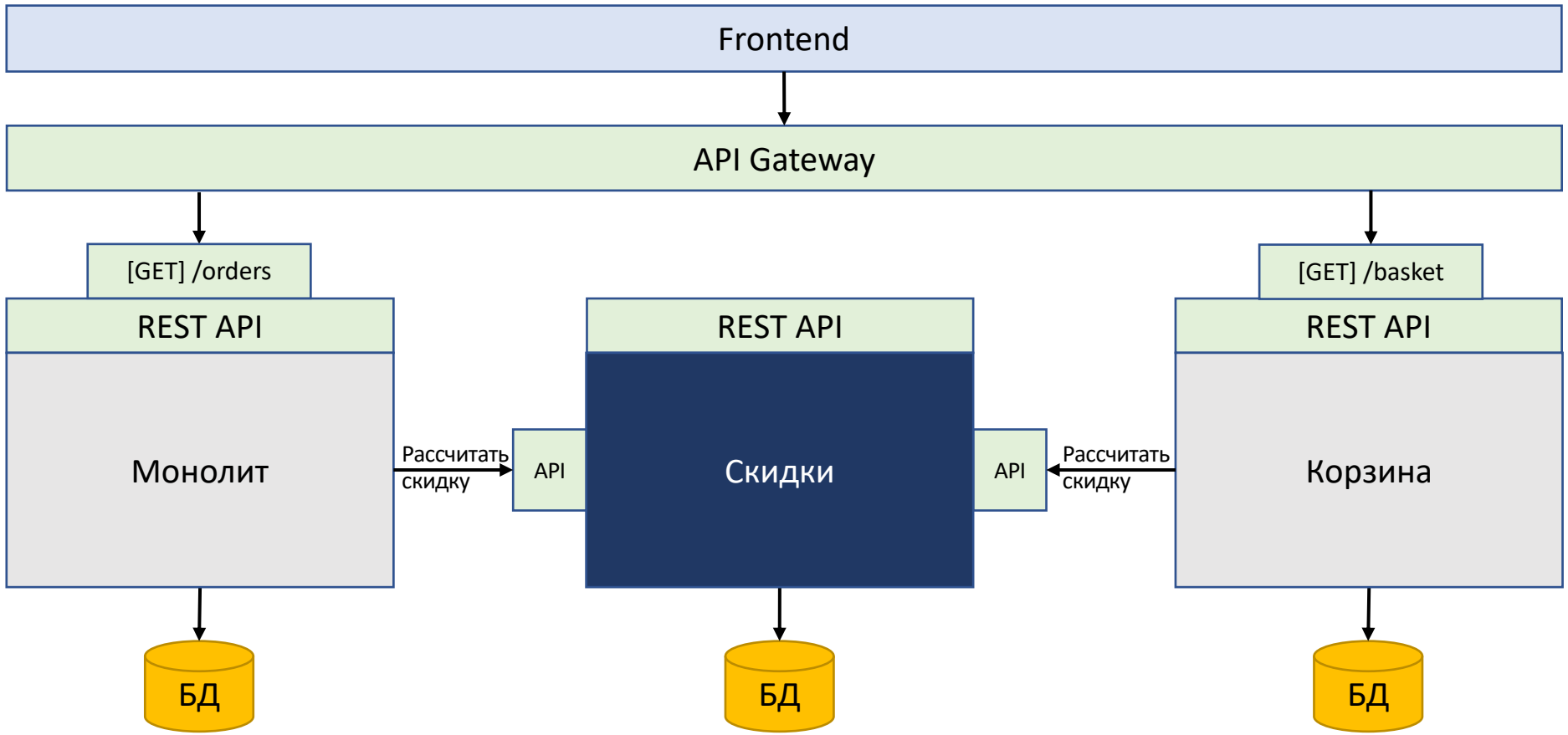
# Доступ к Aggregate через API



# Доступ к Aggregate через API



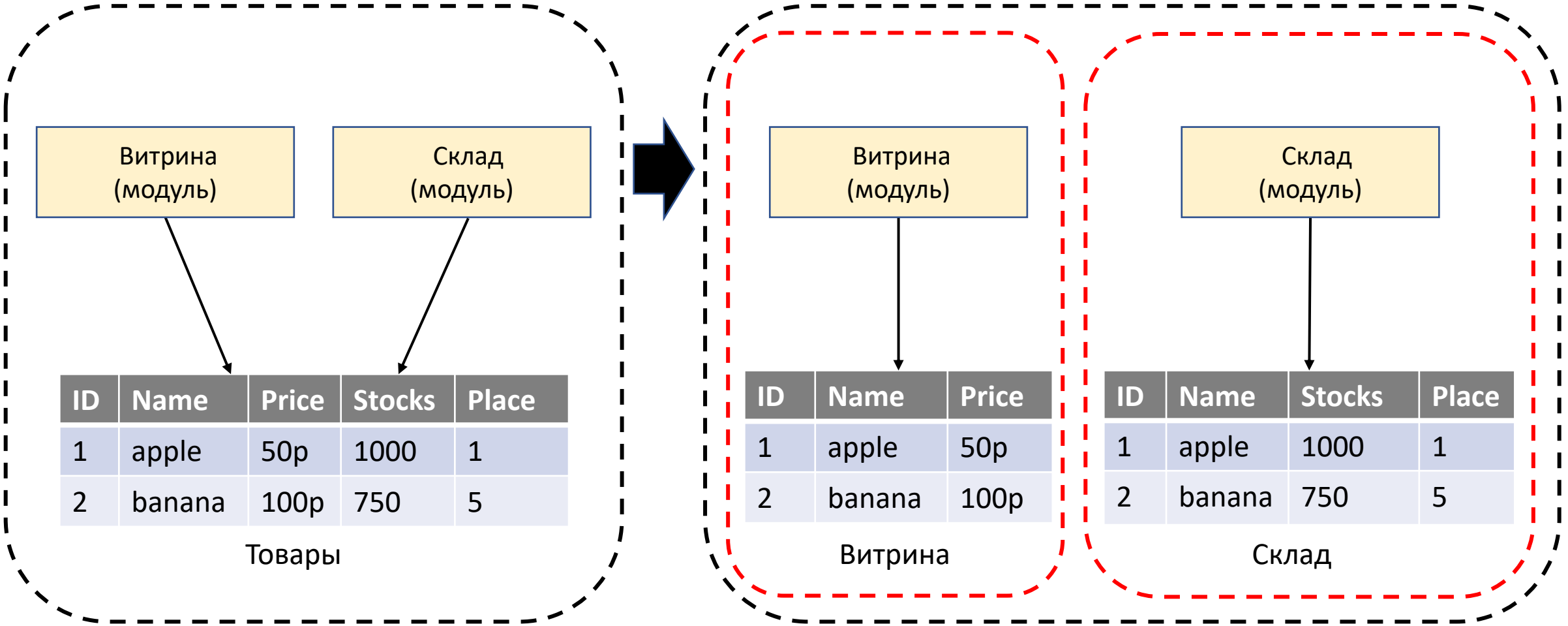
# Доступ к Aggregate через API



# Split Table pattern

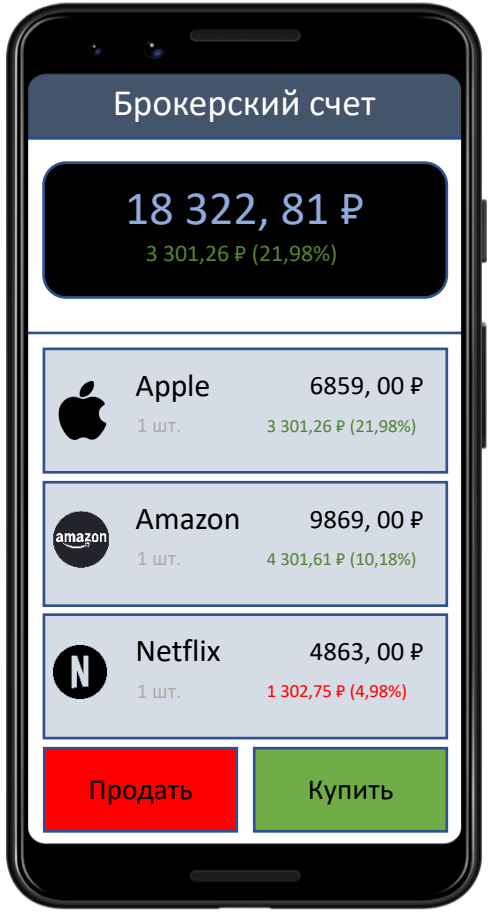
SKLADCHIK.ORG

# Split Table pattern

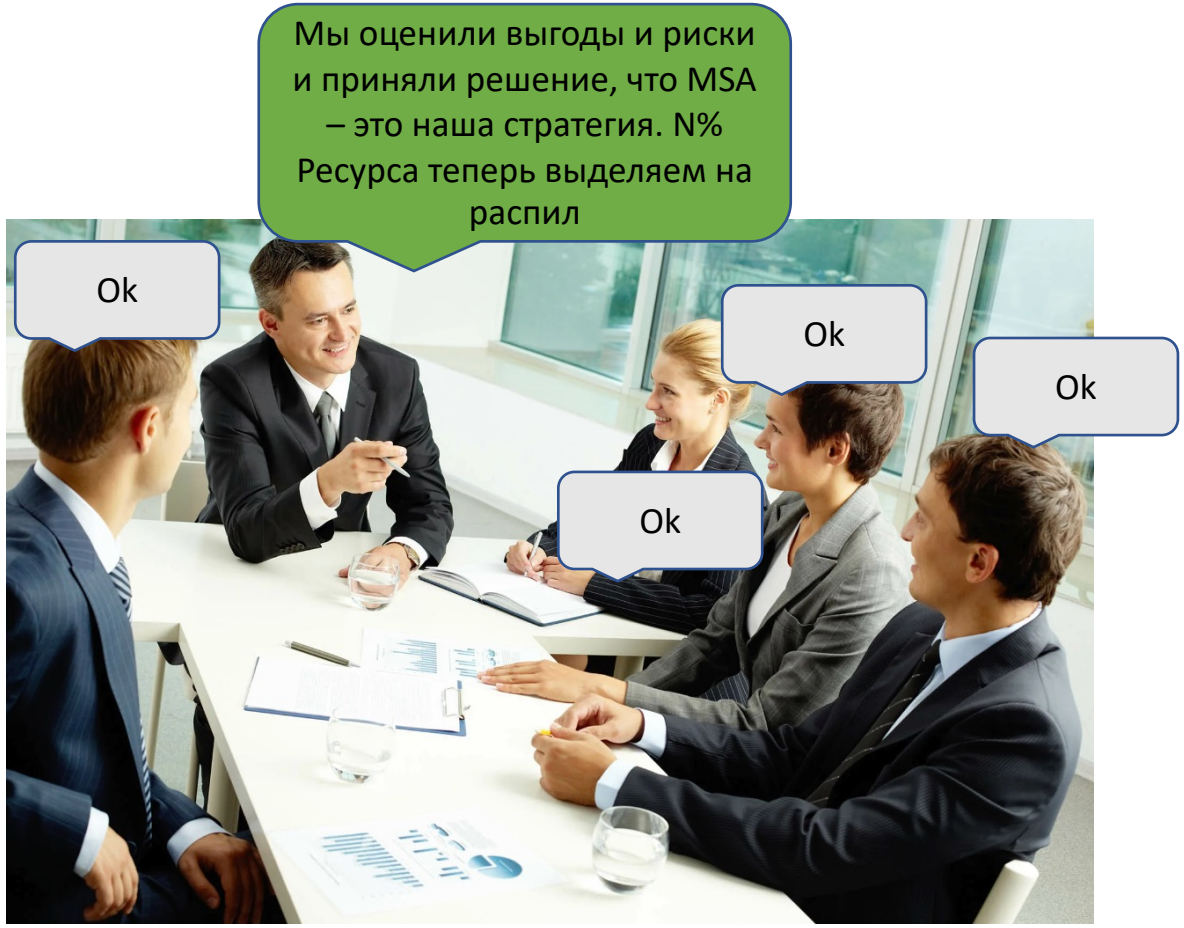


# Пример

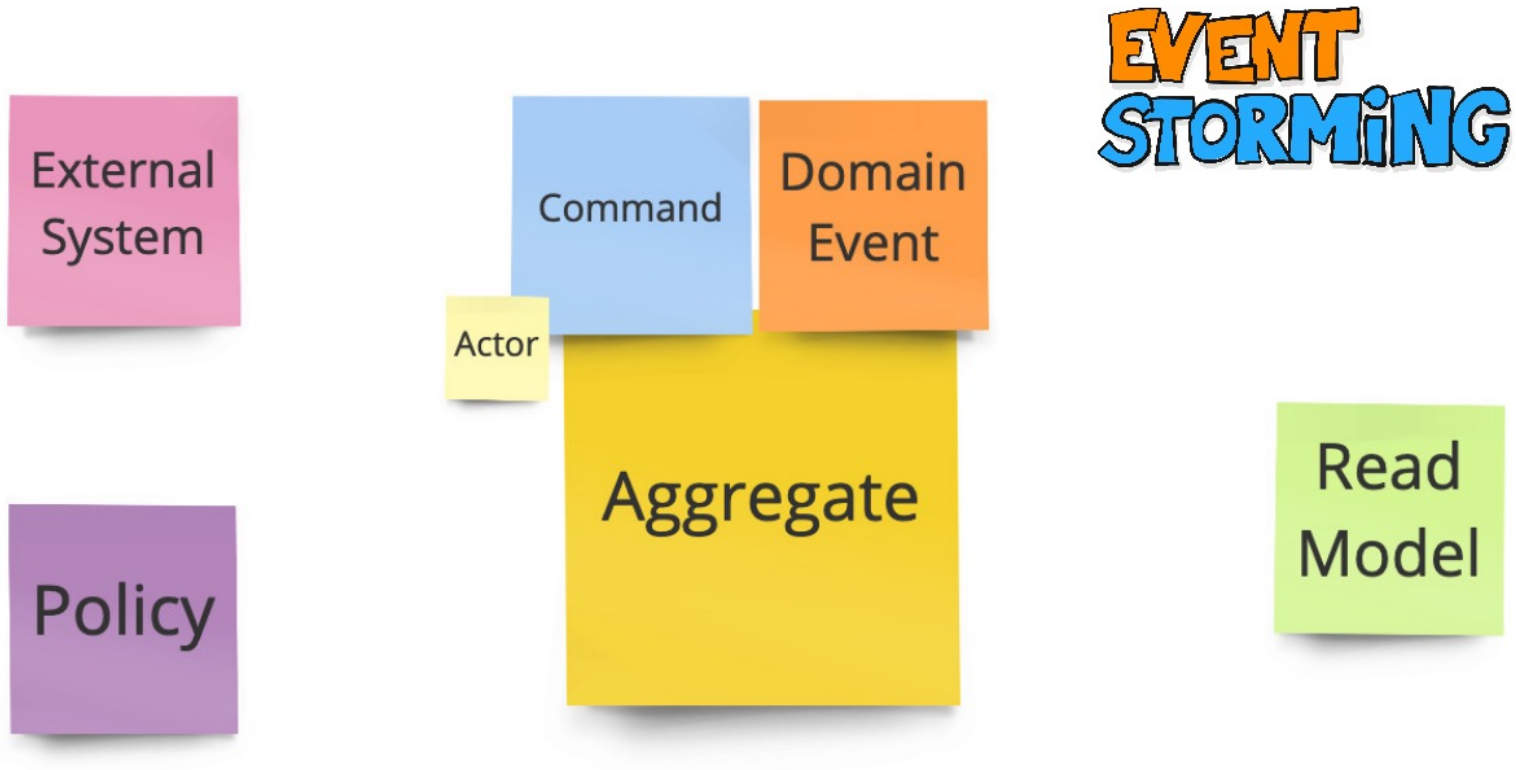
# Система для инвестирования



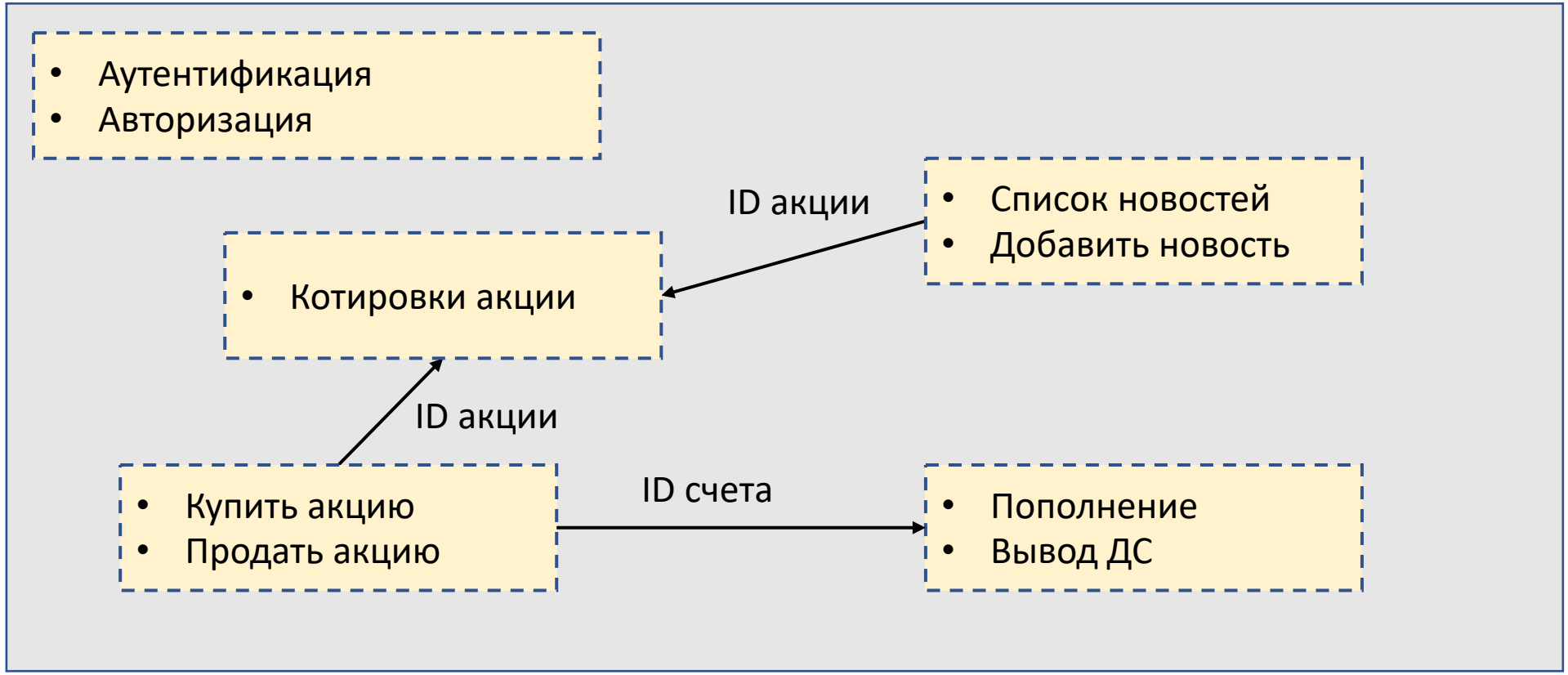

# 1 Шаг. Инициализация проекта



# 2 Шаг. Проведение Event Storming

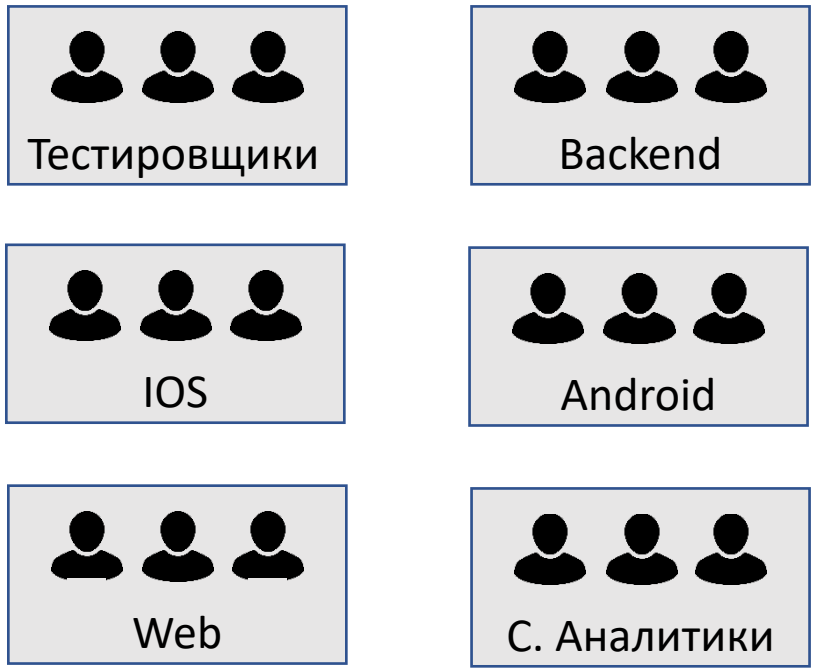


# 2 Шаг. Проведение Event Storming



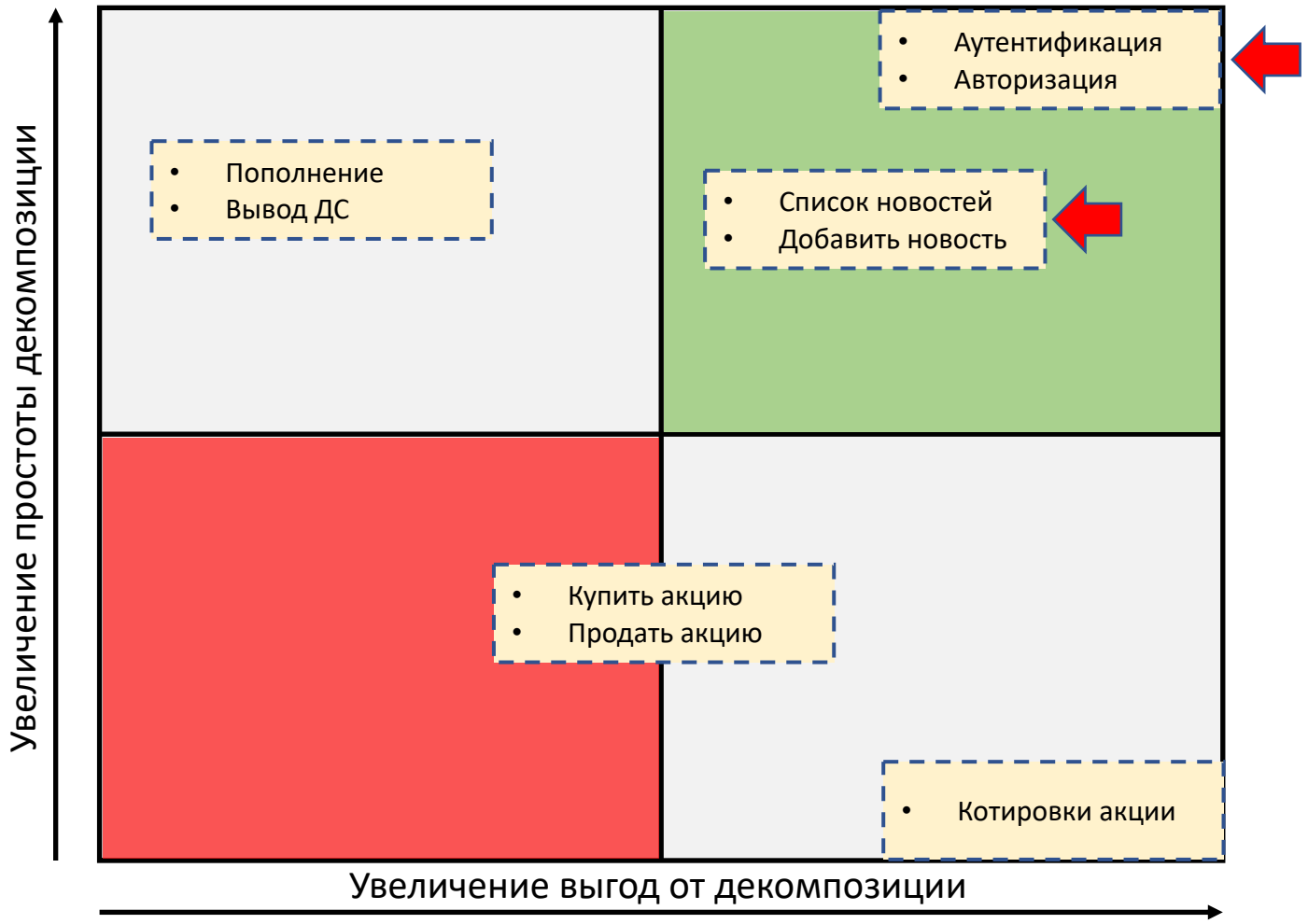
# 3 Шаг. Реорганизация команд

**Было**




**Стало**

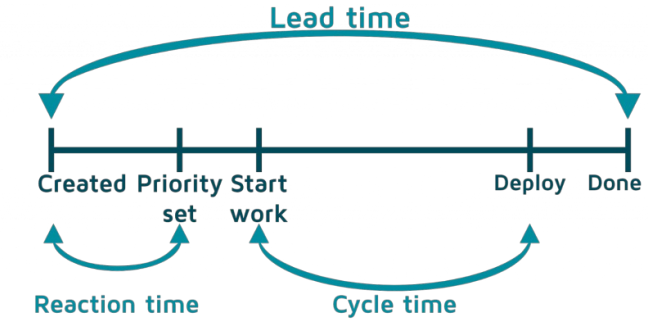
# 4 Шаг. Определение приоритетов

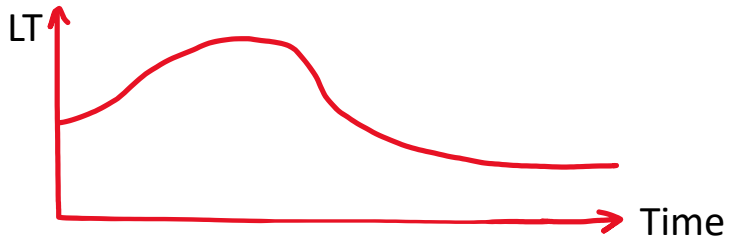


# 5 Шаг. Определение метрик успеха

**Количественные**

- Lead Time

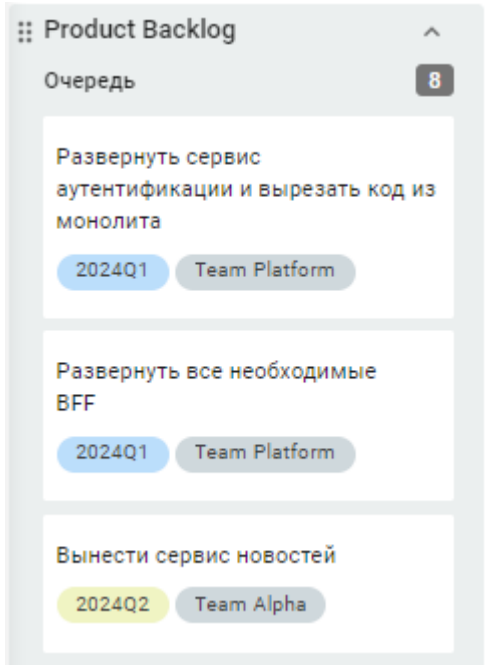


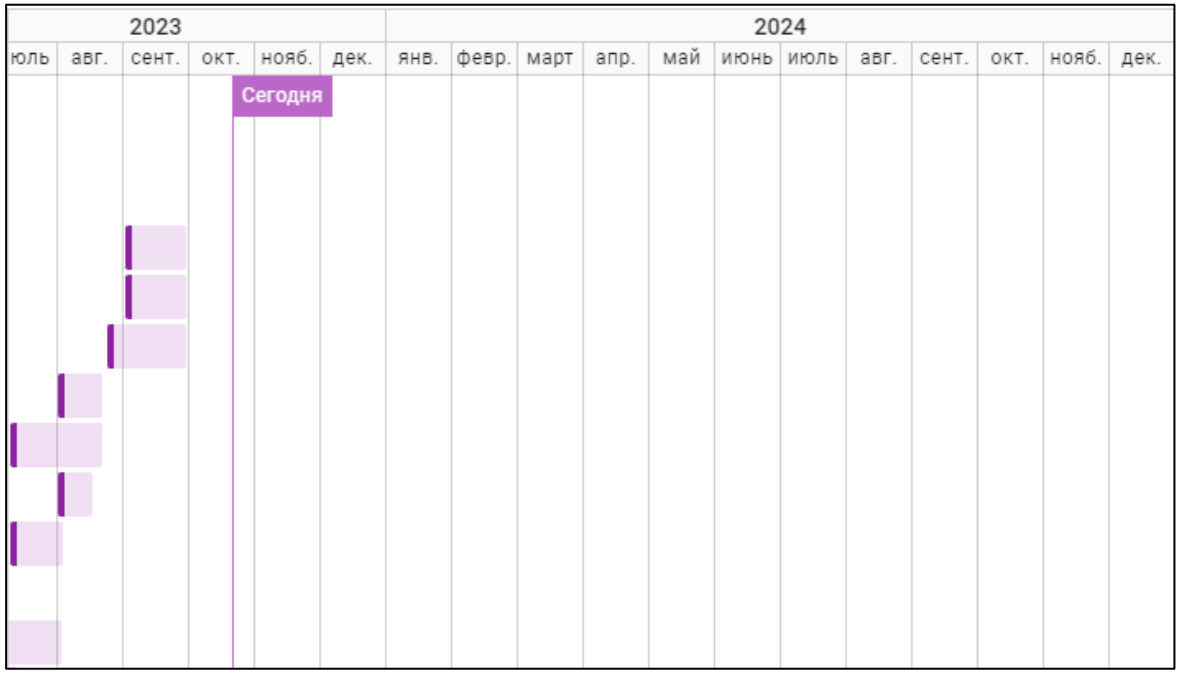




**Качественные**

- Happy Index

# 6 Шаг. Планирование

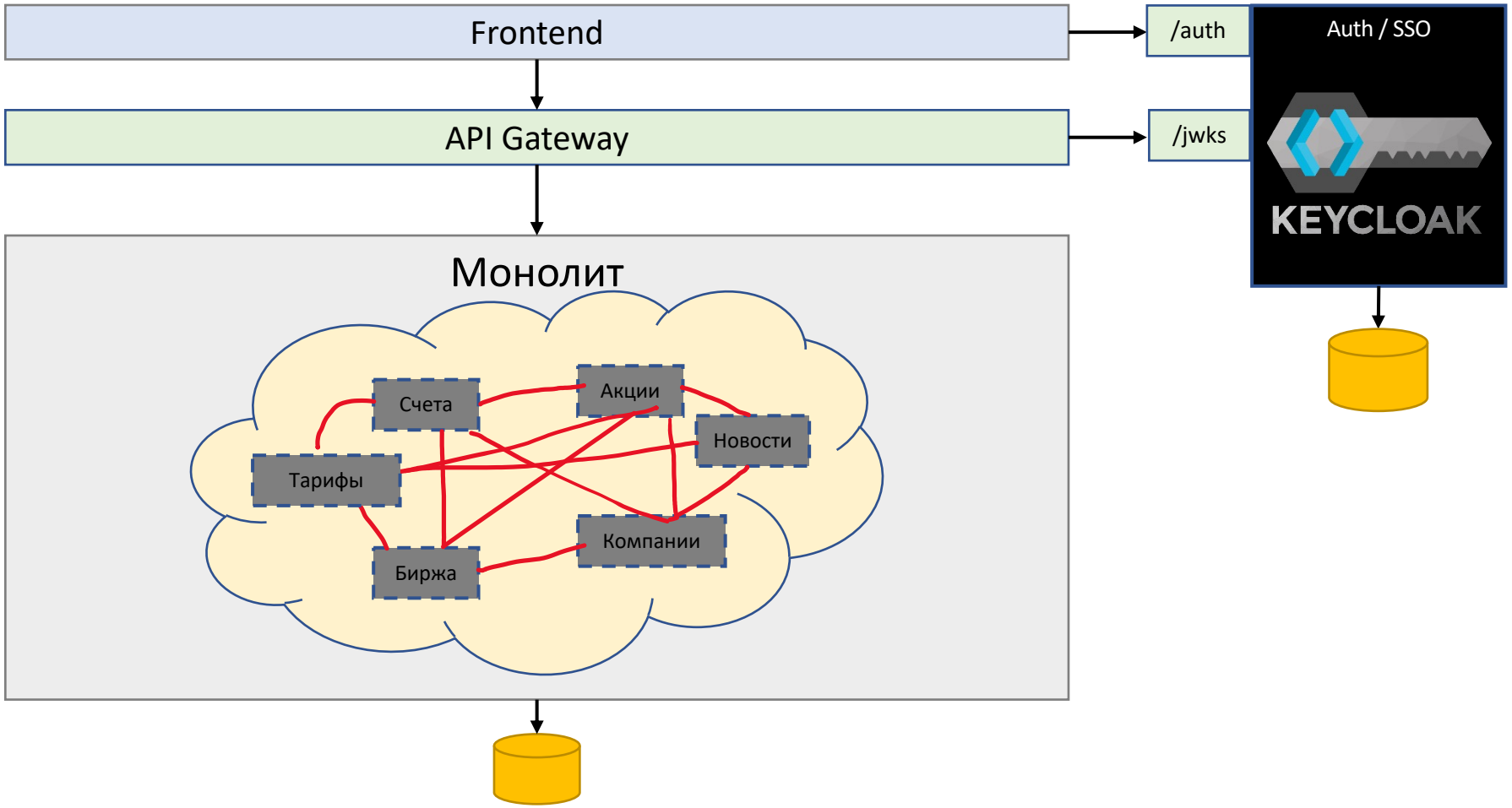


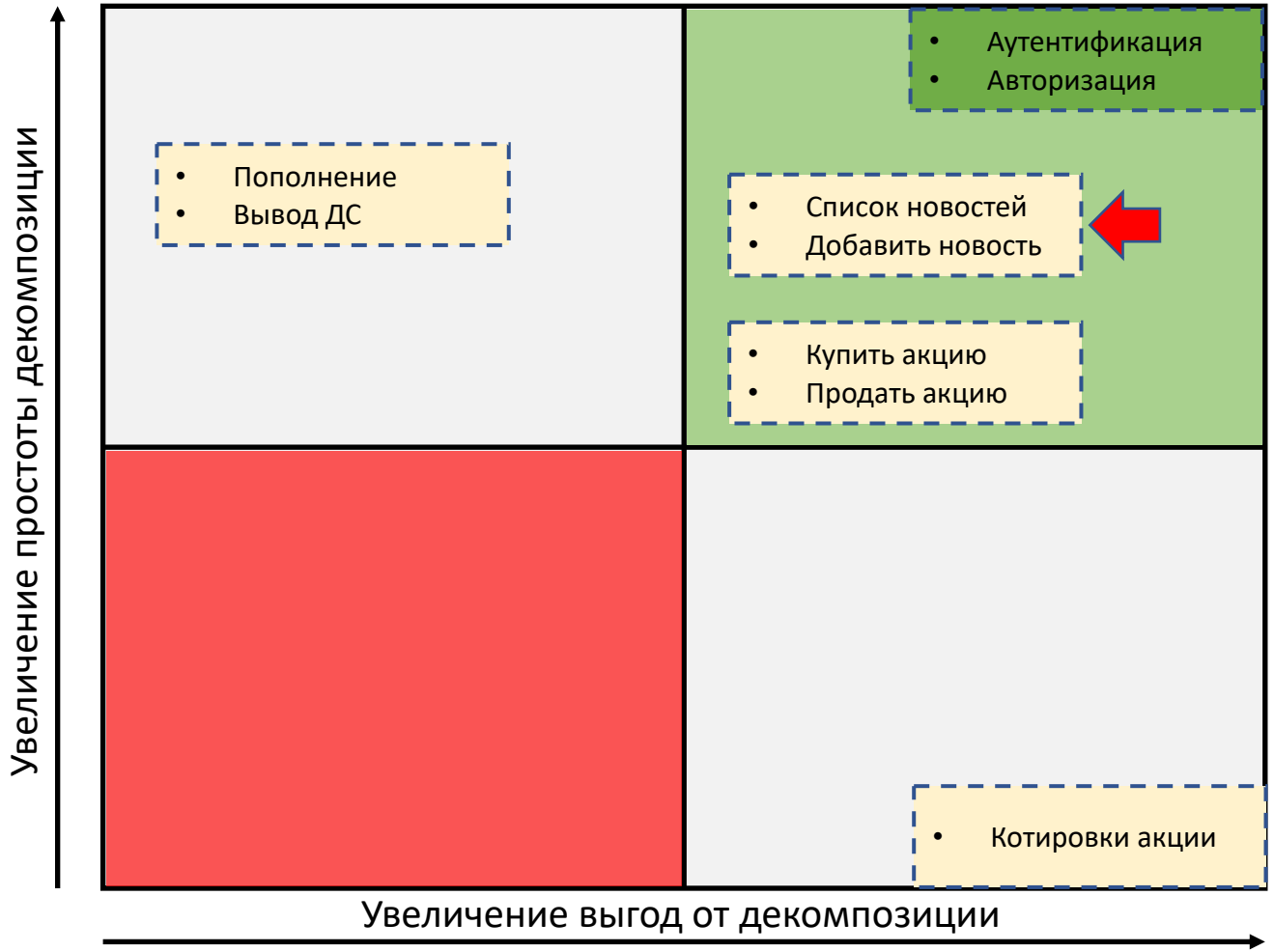
SKLADCHIK.ORG

# Приступаем к работе

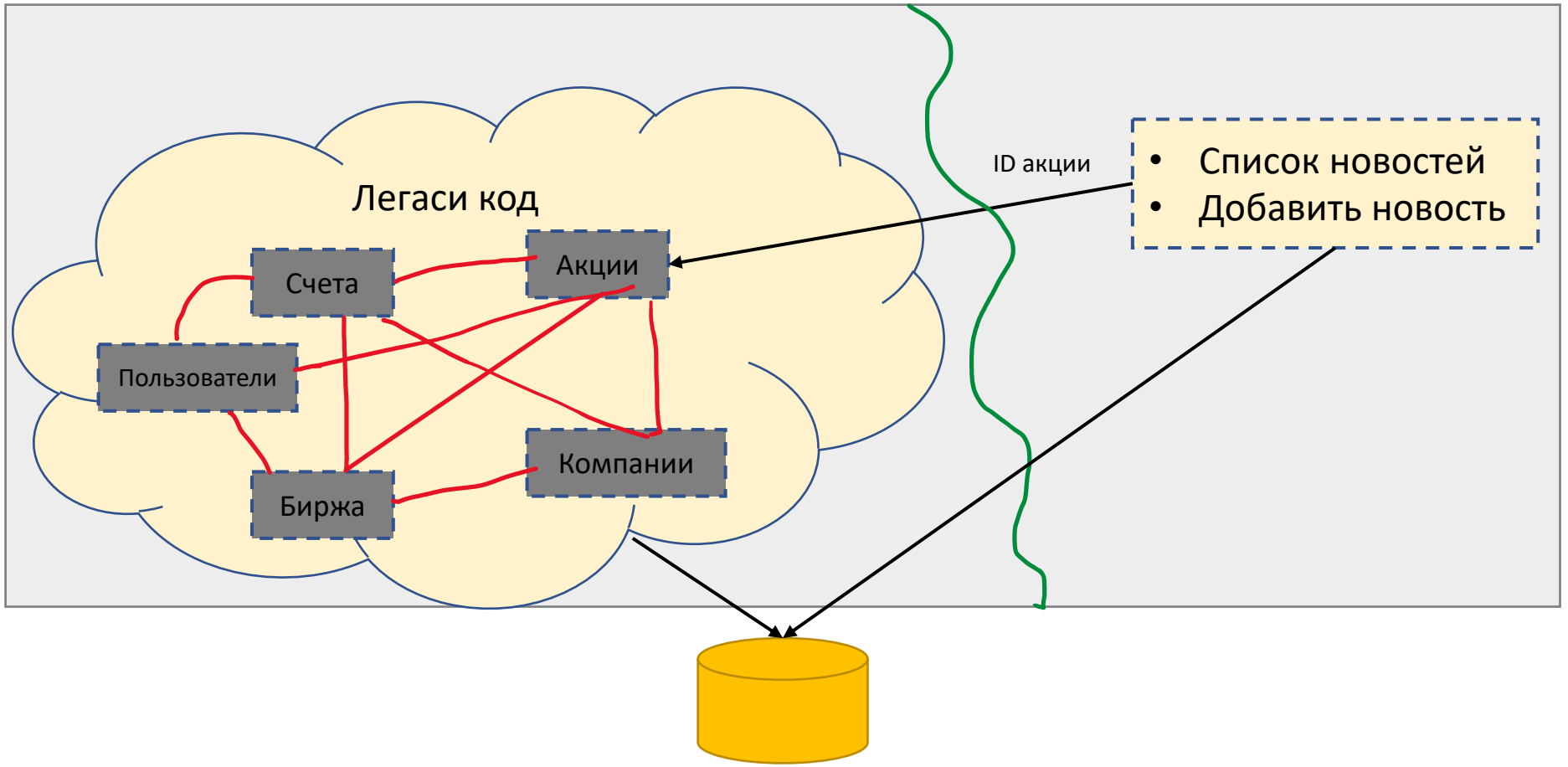
# 1. Готовим «микросервисную платформу»



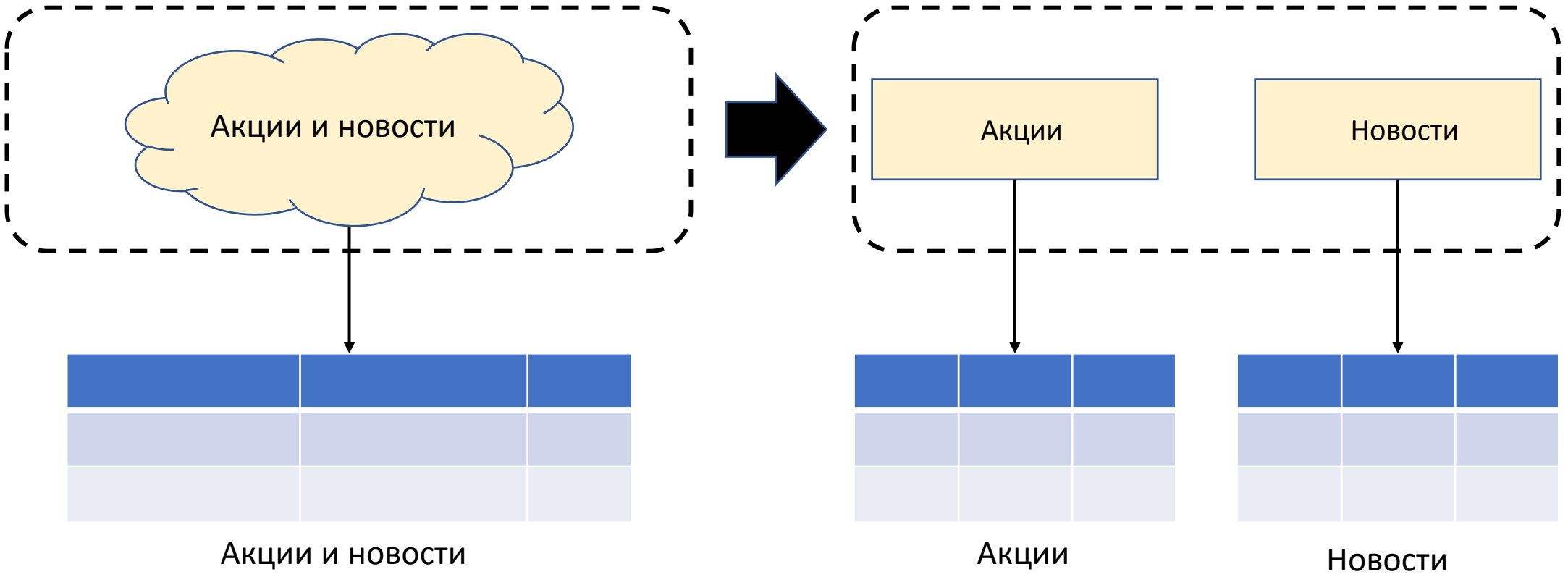
# 2. Выбираем следующий сервис



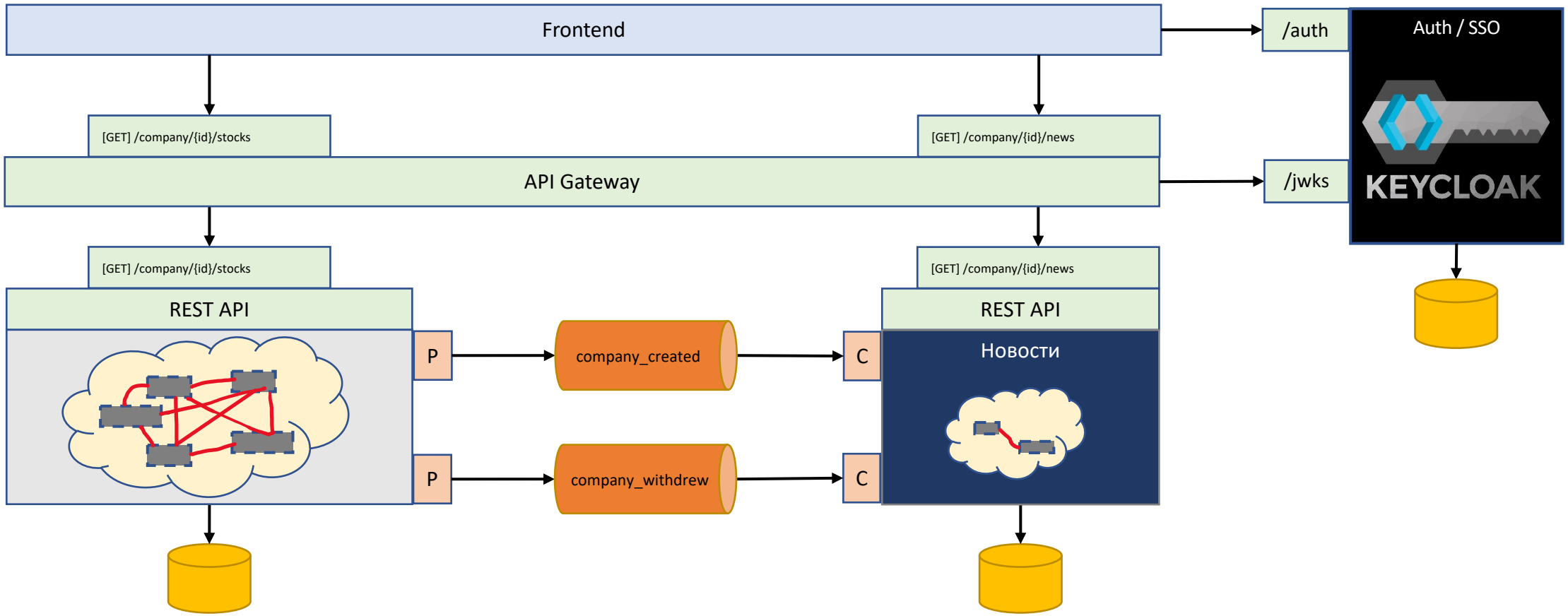
# 3. Выделяем модуль «Новости» в монолите



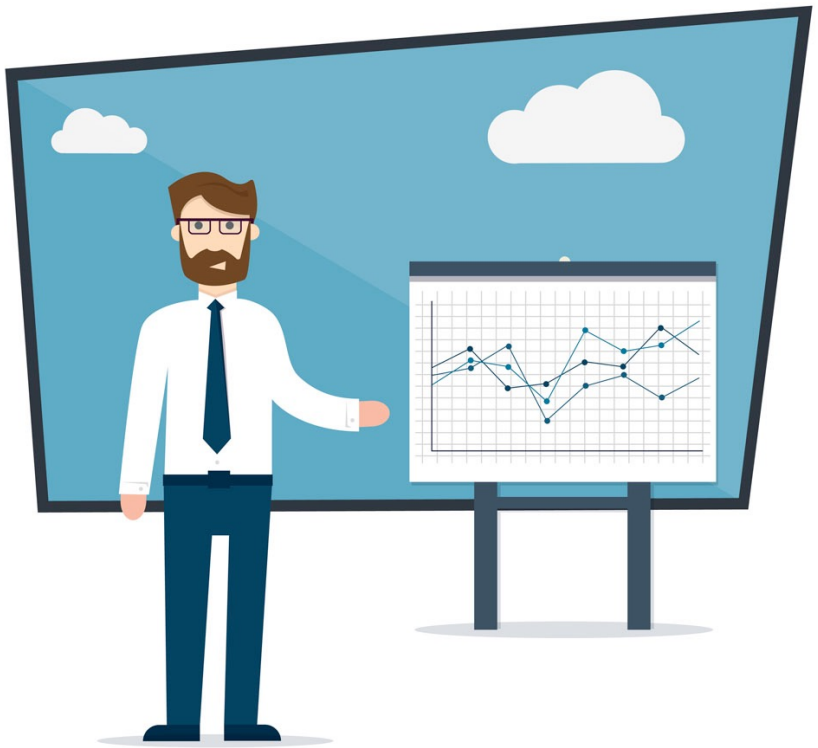
# 4. Декомпозируем БД

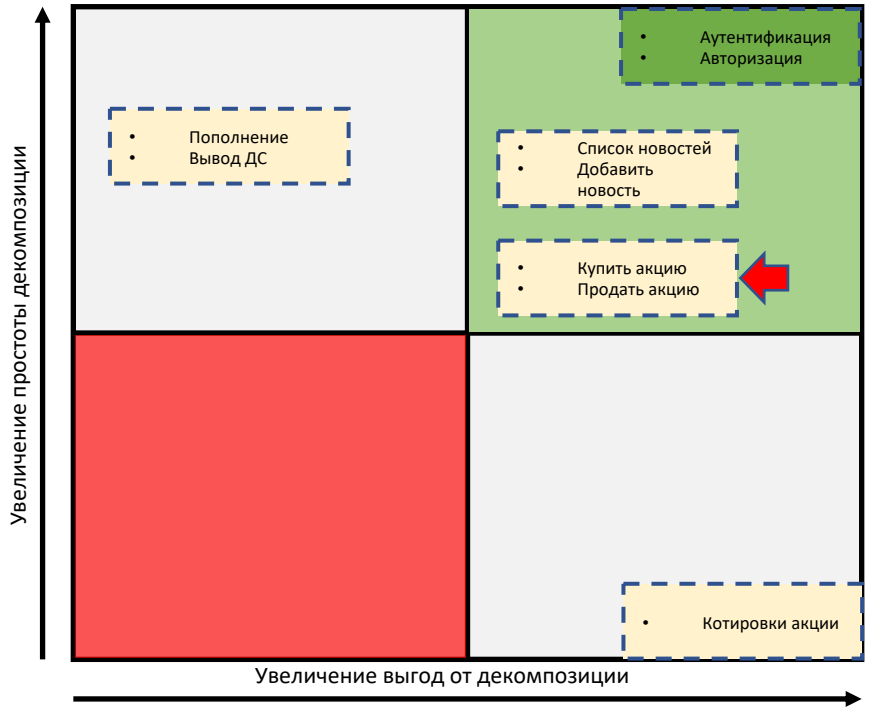


# 5. Выделяем сервис



# 6. Демонстрируем успех бизнесу

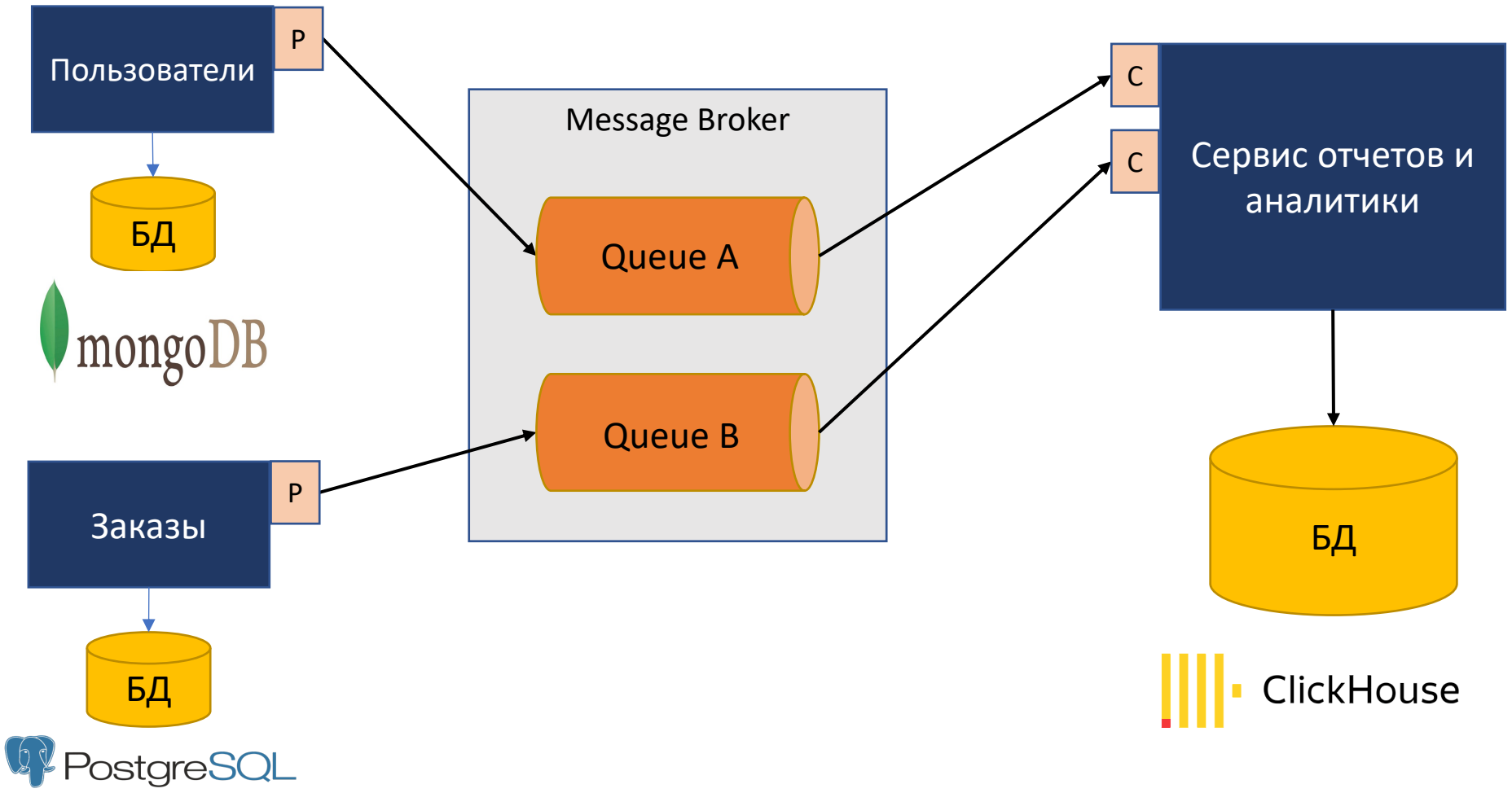




# Отчеты и аналитика

# Репликация БД

# Перекачка данных на основе событий



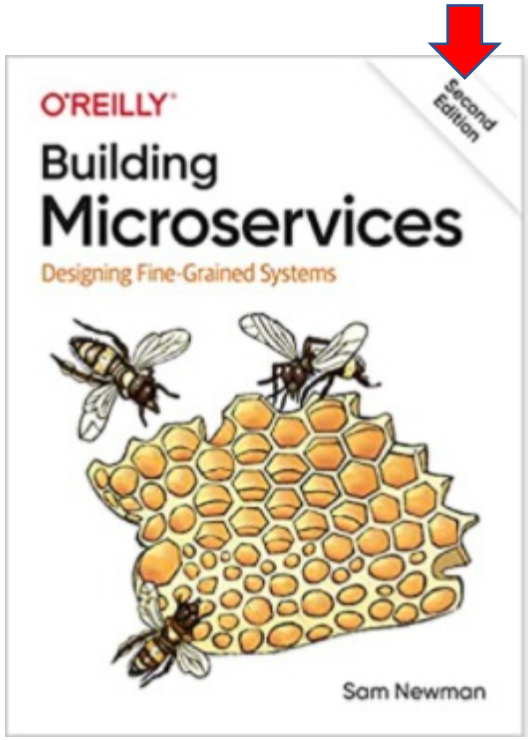
SKLADCHIK.ORG

# О чём узнали

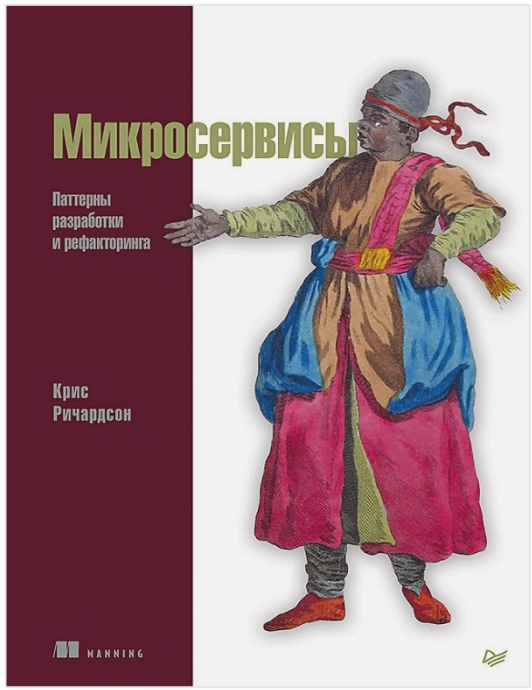
- Если у вас есть монолит – вам повезло
- К задаче декомпозиции монолита можно и нужно относиться как к проекту с конкретным результатом. Project Managment отлично подходит для инициализации и координации процесса декомпозиции монолита
- Если топ-менеджмент вас не поддерживает, то шансы на успех не велики
- Event Storming прекрасно подходит для анализа процессов в монолите, помогает определить верные границы сервисов
- Ещё до начала декомпозиции вы должны подготовить микросервисную платформу
- Отслеживайте метрики и держите бизнес в курсе своих успехов.  Если достигли успеха – покажите его всем.

# Книги и статьи

# Микросервисы







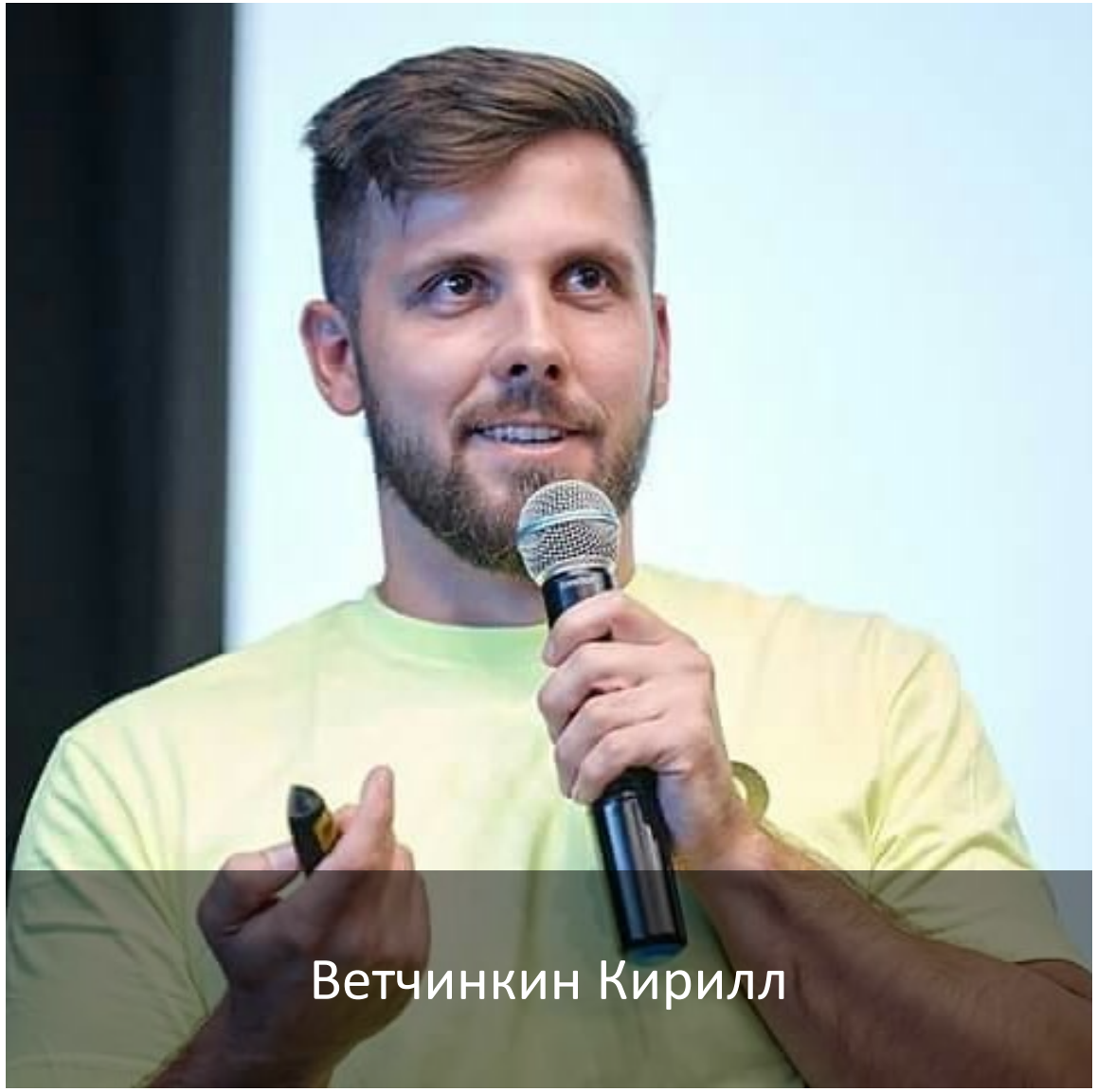

microservices.io

# Контакты

https://microarch.ru

info@microarch.ru

https://t.me/kirill_vet



СКЛАДЧИК

# ЭТОТ КОНТЕНТ КУПЛЕН НА САЙТЕ SKLADCHIK.ORG

ТЫ ЕЩЕ НЕ В КЛУБЕ?
ПРИСОЕДИНЯЙСЯ И НАЧИНАЙ ЭКОНОМИТЬ!

SKL.TO

## ЧТО ТАКОЕ КЛУБ «СКЛАДЧИК»?

### Платформа
Это платформа, где каждый день тысячи людей собираются вместе, чтобы общими усилиями находить, приобретать и изучать курсы интересные именно им.

### Сообщество
Это сообщество из 500 000 людей, открывших для себя выгодный способ быть в тренде самых актуальных и получать ценные знания по минимальной цене.

### Библиотека
Это крупнейшая библиотека инфопродуктов в которой можно найти практически любой курс или тренинг продававшийся за последние 10 лет, а также уникальные авторские инфопродукты, которые не получиться найти больше нигде.