

# Микросервисы: принципы

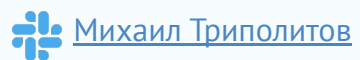


Михаил  
Триполитов



**Михаил Триполитов**

Technical Lead





## План занятия

1. [Проектирование системы](#)
2. [Разделение на сервисы](#)
3. [Взаимодействие между сервисами](#)
4. [Двенадцать факторов](#)
5. [Итоги](#)
6. [Домашнее задание](#)



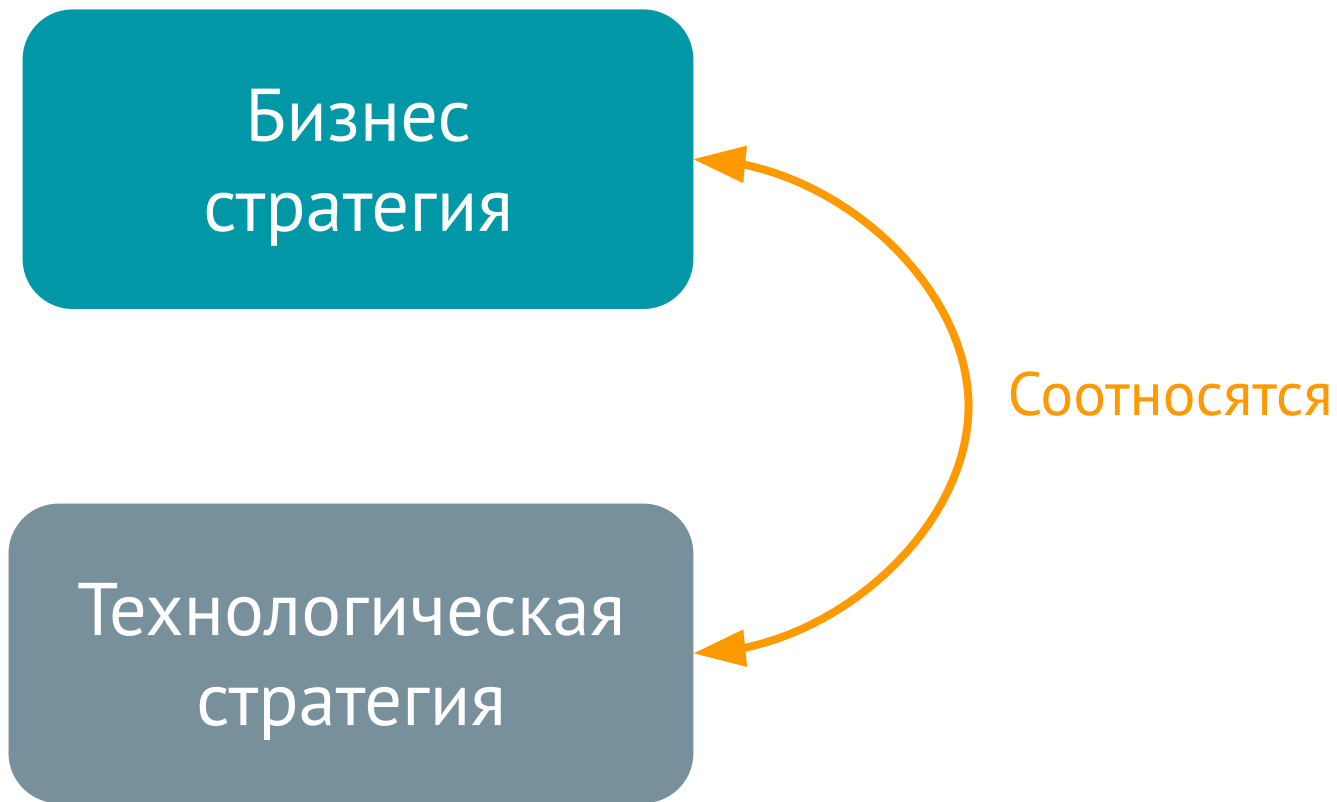
# Проектирование системы

---

# Архитектура

- ✓ Правила
- ✓ Принципы
- ✓ Практики
- ✓ Ограничения
- ✗ Детальный план
- ✗ Подробная документация

# Стратегические цели



---

# Принципы



Принципы

# Практики

## Цели

Масштабирование  
бизнеса

Сокращение  
используемого  
операторами  
программного  
обеспечения

Оптимизация  
ресурсов и затрат

## Принципы

Обеспечивать  
согласованные  
интерфейсы и потоки  
данных

Выбирать решения с  
быстрым циклом  
обратной связи

Уменьшать  
избыточную  
сложность, заменяя  
дублирующие  
системы

## Практики

Использовать HTTP для  
межсервисных интеграций

Исключать интеграционные  
базы данных

Использовать независимые  
сервисы

Применять непрерывную  
интеграцию и постоянное  
развертывание

Использовать мониторинг  
для получения состояния  
сервисов

Автоматически генерировать  
клиентские библиотеки для  
всех публикуемых  
интерфейсов



---

## Практики: необходимый минимум

- Централизованный мониторинг
- Централизованный сбор логов
- Ограниченный набор допустимых интерфейсов
- Безопасное поведение



# Разделение на сервисы



# Признаки хорошего сервиса

## **Loose Coupling**

Минимизировать влияние изменений в одном сервисе на всю систему

## **High Cohesion**

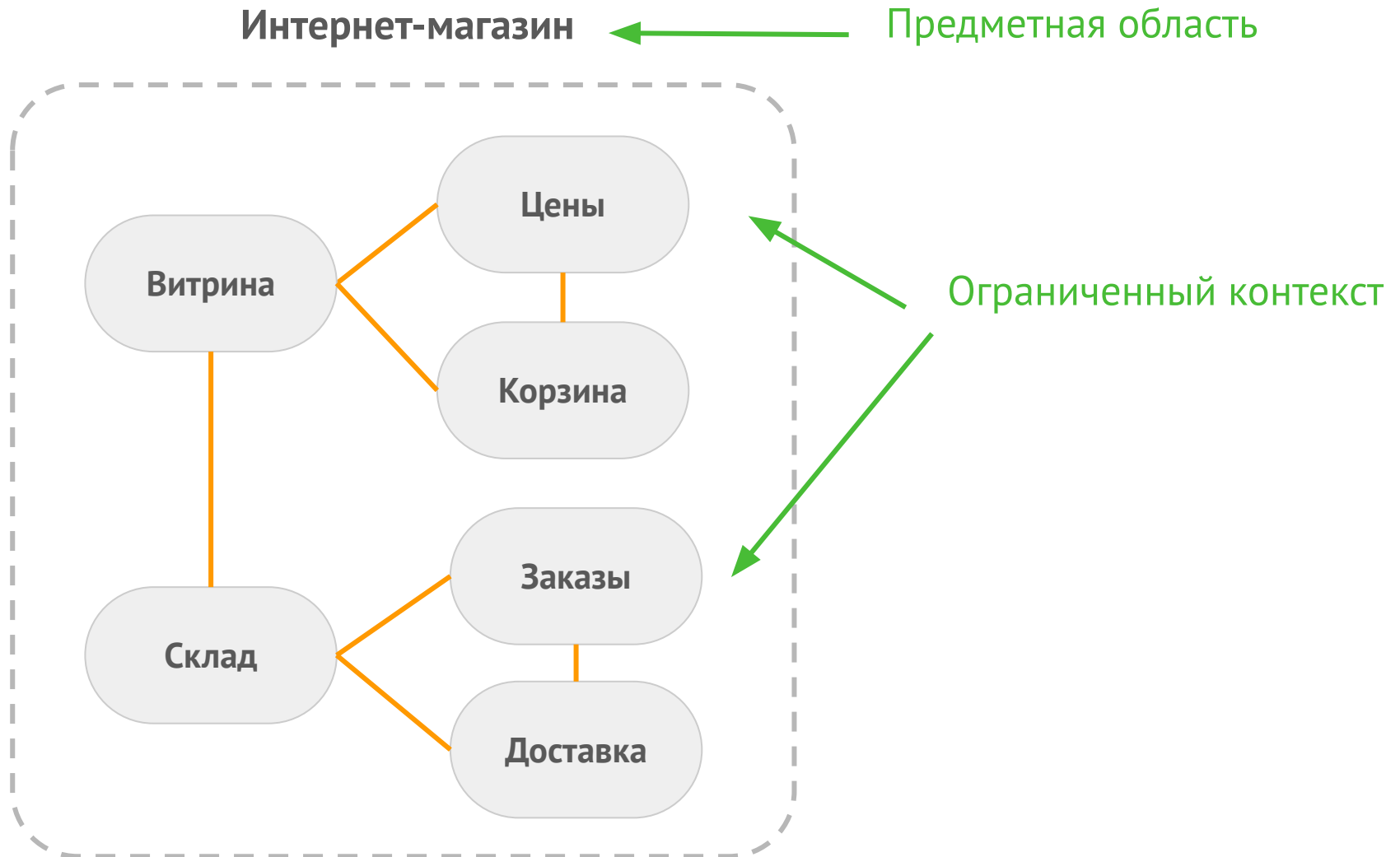
Минимизировать необходимость менять несколько сервисов при изменении поведения системы

# Bounded Context

Предметная область



# Bounded Context



---

# Один контекст. Один сервис. Одна команда.

**Ограниченный контекст - это:**

- Отдельная команда
- Отдельный репозиторий
- Отдельная схема базы данных
- Отдельная процедура тестирования
- Отдельная процедура выкладки

---

# Причины уменьшать


- Частота изменений
- Масштабирование
- Независимость
- Сложность



# Bounded Context

Использование ограниченных контекстов для разбиения системы на сервисы позволит **обеспечить слабую связность и сильное зацепление**, соблюдая баланс размера сервисов.





# Взаимодействие между сервисами

# Выбор

## Какой протокол выбрать?

- XML-RPC
- JSON-RPC
- gRPC
- REST
- GraphQL
- SOAP
- ...

## Что еще выбрать?

- Синхронно / Асинхронно
- Оркестрация / Хореография
- RPC / Команды и события
- ....

---

# Выбор

- Обратная совместимость
- Технологическая независимость
- Забота о потребителях
- Четкий интерфейс



# Обратная совместимость

Избегайте обратно несовместимых изменений



# Технологическая независимость

Избегайте интеграционных технологий и подходов, которые привязаны к какой-то конкретной технологической платформе

---

# Забота о потребителях

Стремитесь снизить требования к клиентам и упростить использование интерфейсов

- Документация
- Простое и понятное API
- Готовые клиентские библиотеки



## Четкий интерфейс

Прячьте детали реализации от потребителей интерфейсов.  
Избегайте интеграционных подходов и технологий, которые раскрывают детали внутренней реализации потребителям.

---

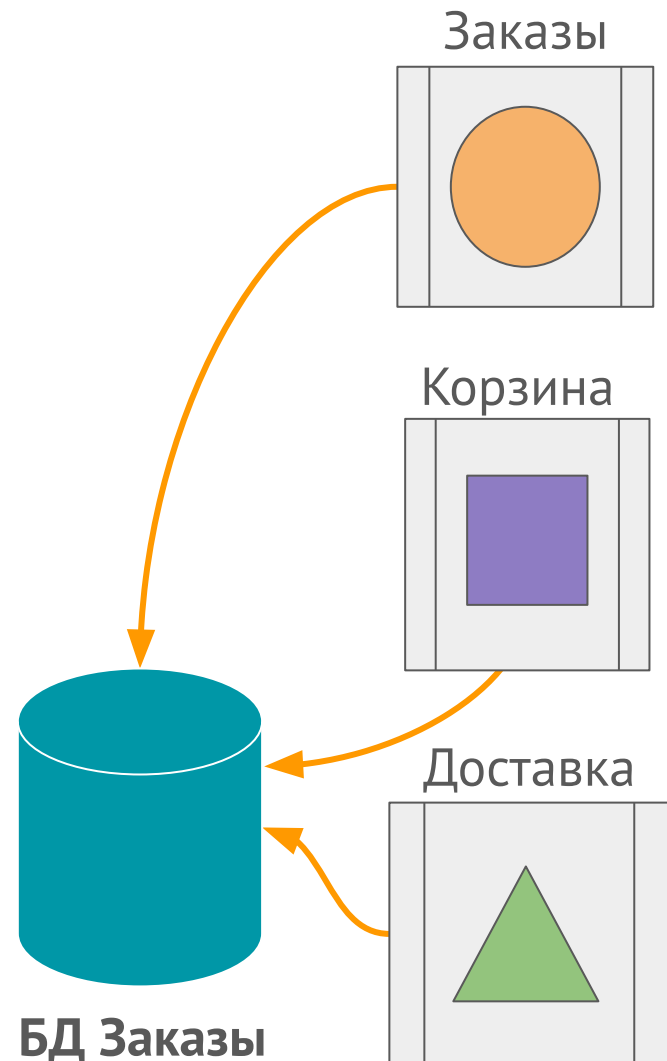
# Выбор

- Обратная совместимость
- Технологическая независимость
- Забота о потребителях
- Четкий интерфейс

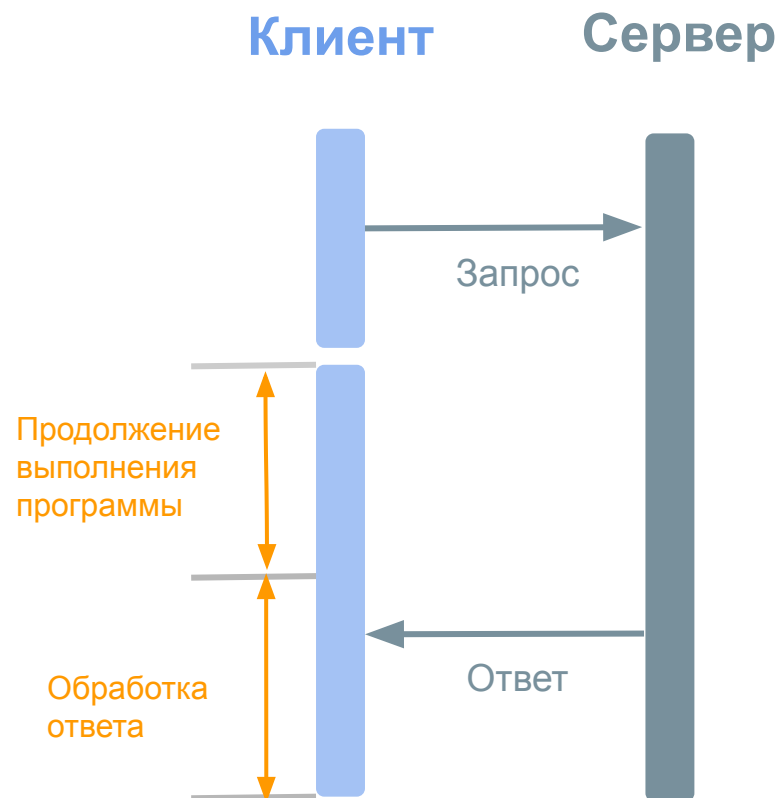
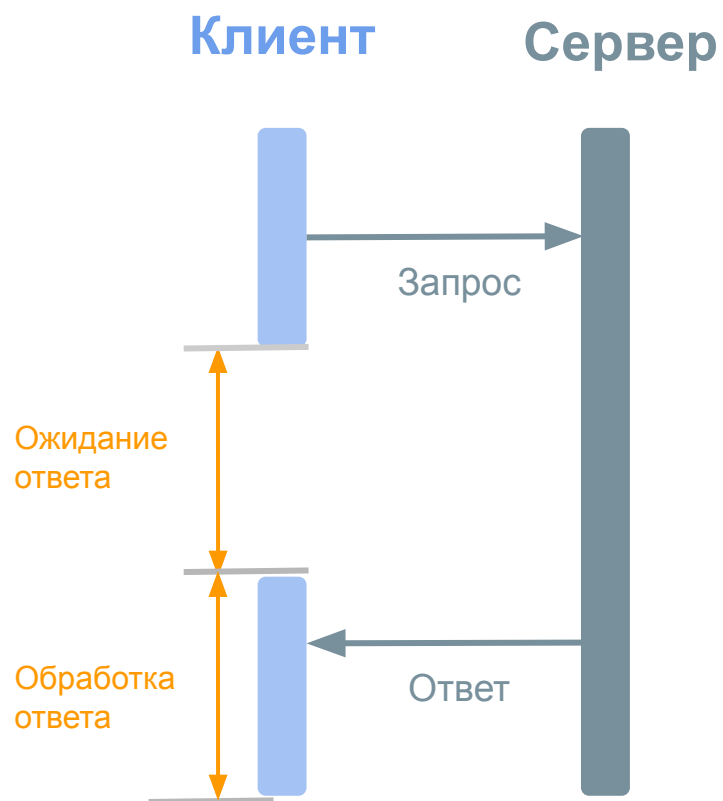


# Общая база данных

- ❌ Детали реализации доступны другим сервисам
- ❌ Потребители ограничены выбранной технологией БД
- ❌ У данных нет единого владельца — логика по манипуляции данными распределена по разным сервисам



# Синхронное или асинхронное взаимодействие



---

# Запрос / Ответ или События

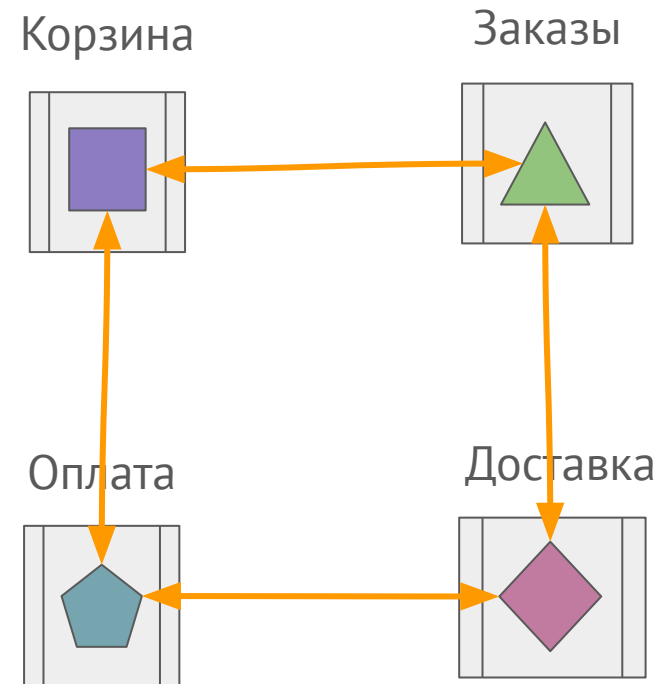
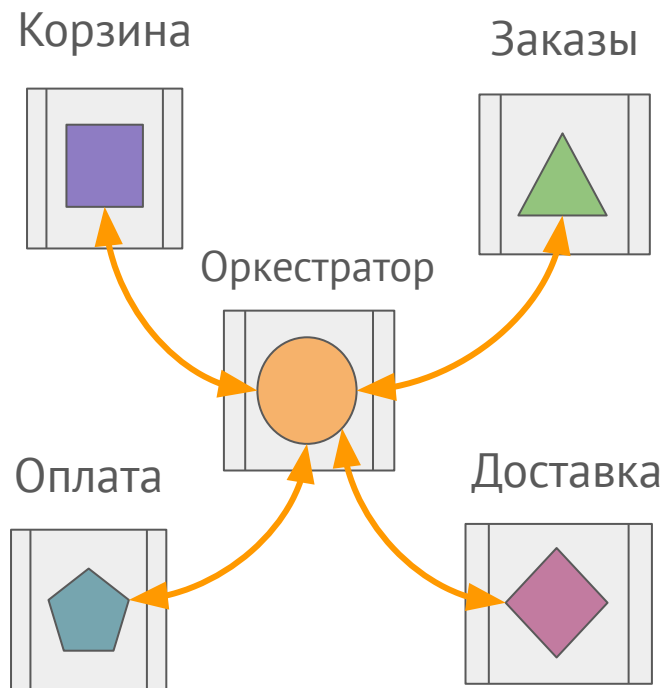
## Запрос / Ответ

- ✓ Проще для понимания
- ✓ Синхронное и асинхронное взаимодействие
- ✗ Централизует бизнес логику
- ✗ Увеличивает связность

## Событийная модель

- ✓ Уменьшает связность
- ✓ Децентрализация логики
- ✓ Простота расширения системы
- ✗ Общая сложность системы
- ✗ Только асинхронный подход

# Оркестрация или Хореография



---

# Remote Procedure Calls

- ✓ Детали реализации недоступны другим сервисам
- ✓ При правильном выборе технологии потребители не ограничены одним стеком
- ✓ Простота использования
- ✗ Расширение моделей возможно только через добавление полей
- ✗ Ограниченная поддержка инфраструктурными инструментами

---

# REST ( REpresentation State Transfer )

- ✓ Детали реализации недоступны другим сервисам
- ✓ Потребители не ограничены одним стеком
- ✓ Простота использования
- ✓ Хорошая поддержка инфраструктурными инструментами
- ✓ Текстовый формат данных JSON, XML
- ✗ Расширение моделей возможно только через добавление полей
- ✗ Не всегда возможно описать модель в терминах протокола HTTP

---

# GraphQL

- ✓ Детали реализации недоступны другим сервисам
- ✓ Потребители не ограничены одним стеком
- ✓ Возможность получить несколько ресурсов одним запросом
- ✓ Возможность указать необходимые данные
- ✗ Расширение моделей возможно только через добавление полей
- ✗ Нет поддержки кэширования со стороны инфраструктуры

---

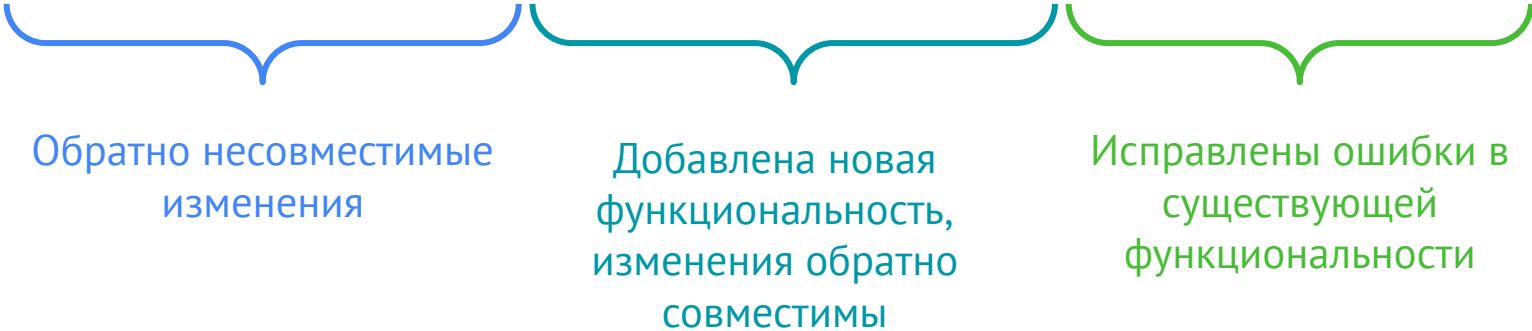
## Событийная модель

- ✓ Детали реализации недоступны другим сервисам
- ✓ Потребители не ограничены одним стеком
- ✓ Возможность получить несколько ресурсов одним запросом
- ✓ Низкая общая связность решения
- ✓ Высокая гибкость и способность к расширению
- ✗ Потребители ограничены выбранной технологией
- ✗ Высокая общая сложность системы
- ✗ Повышенные требования к инфраструктуре



## Версионирование: SemVer

# MAJOR.MINOR.PATCH



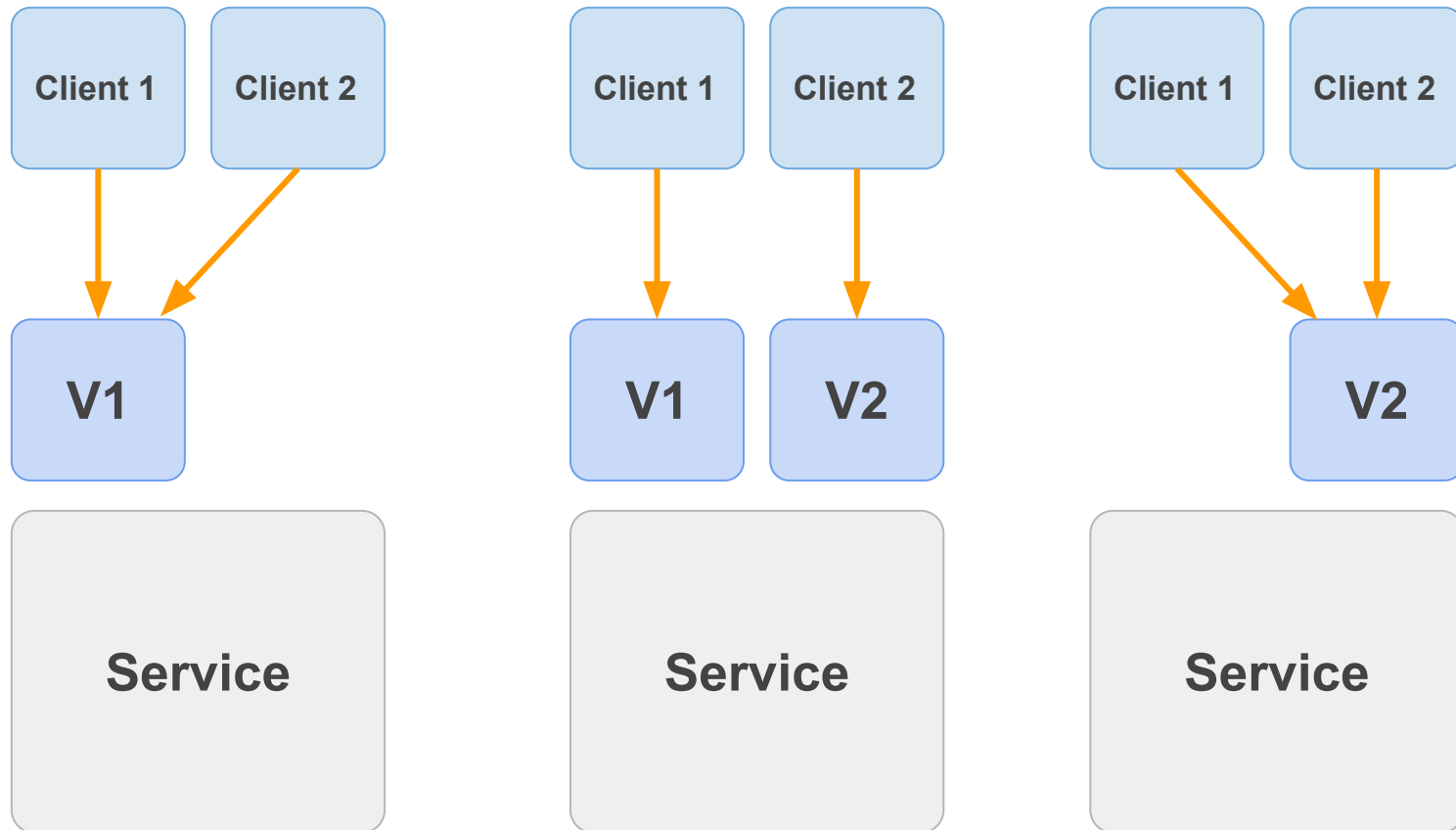
Обратно несовместимые  
изменения

The diagram illustrates the three components of a Semantic Version (SemVer) string: MAJOR, MINOR, and PATCH. Each component is bracketed and associated with a specific type of change. The MAJOR component is associated with 'Обратно несовместимые изменения' (Backward incompatible changes) in blue. The MINOR component is associated with 'Добавлена новая функциональность, изменения обратно совместимы' (Added new functionality, backward compatible changes) in teal. The PATCH component is associated with 'Исправлены ошибки в существующей функциональности' (Fixed bugs in existing functionality) in green.

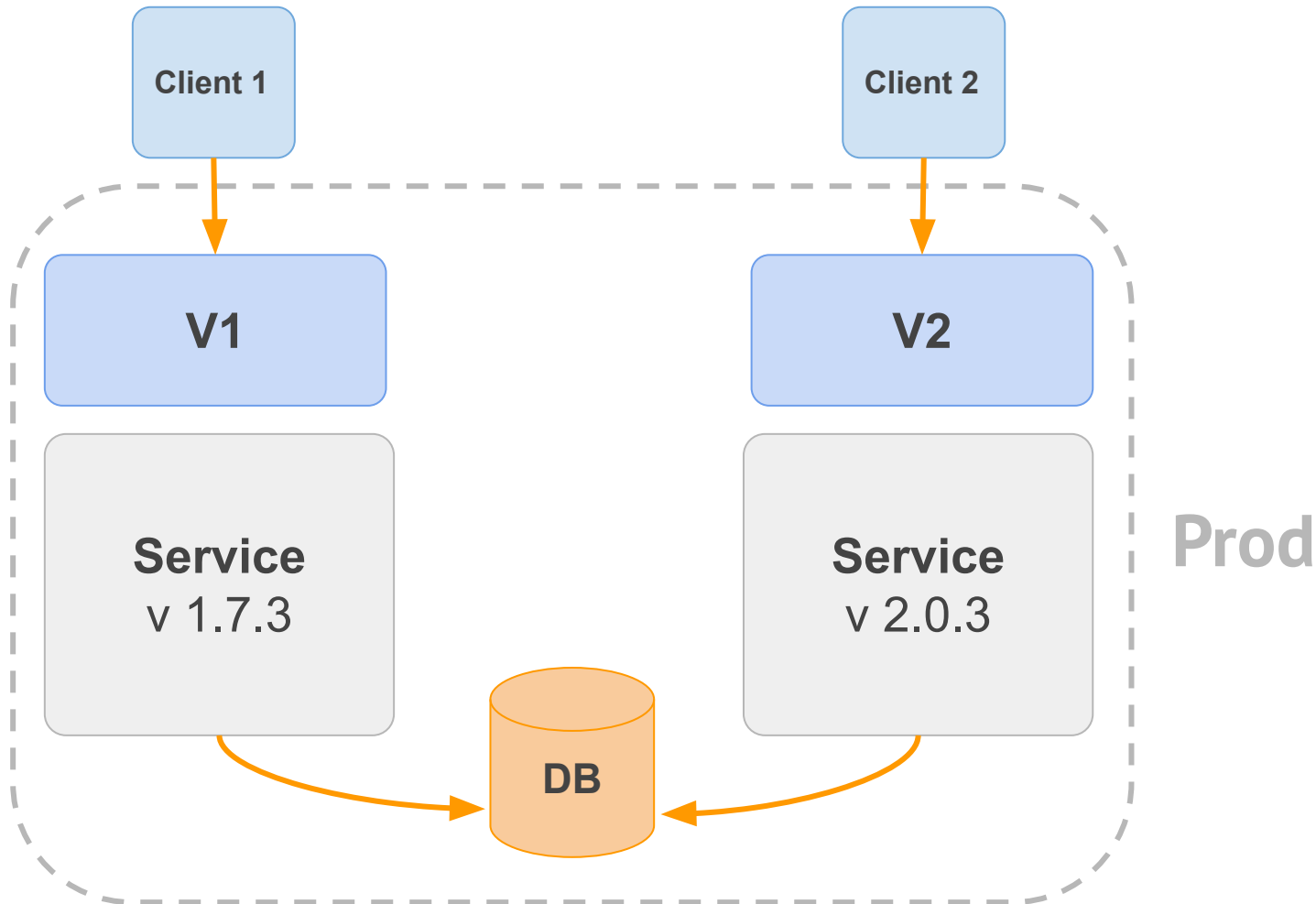
Добавлена новая  
функциональность,  
изменения обратно  
совместимы

Исправлены ошибки в  
существующей  
функциональности

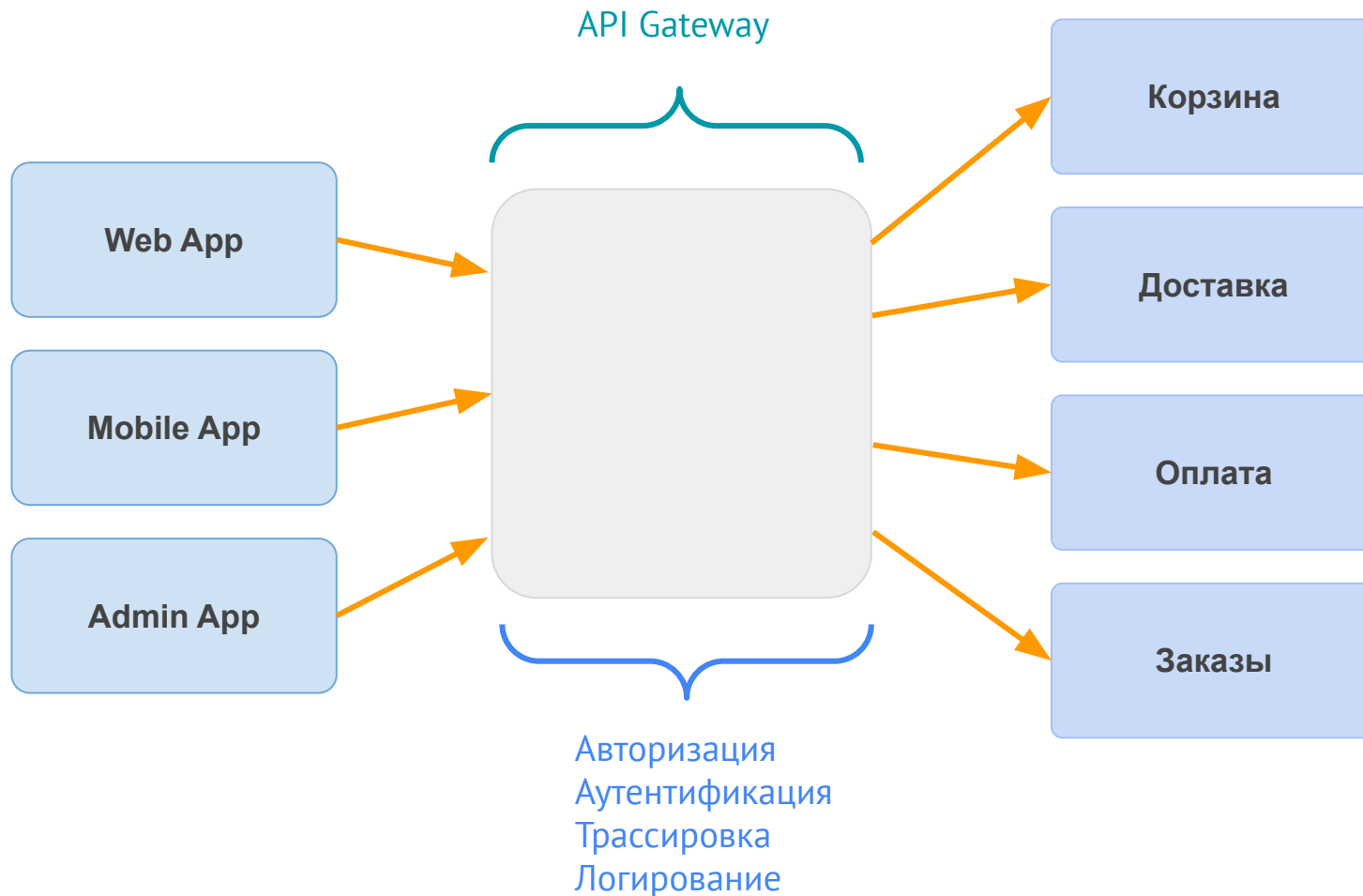
# Версионирование: версии эндпоинтов



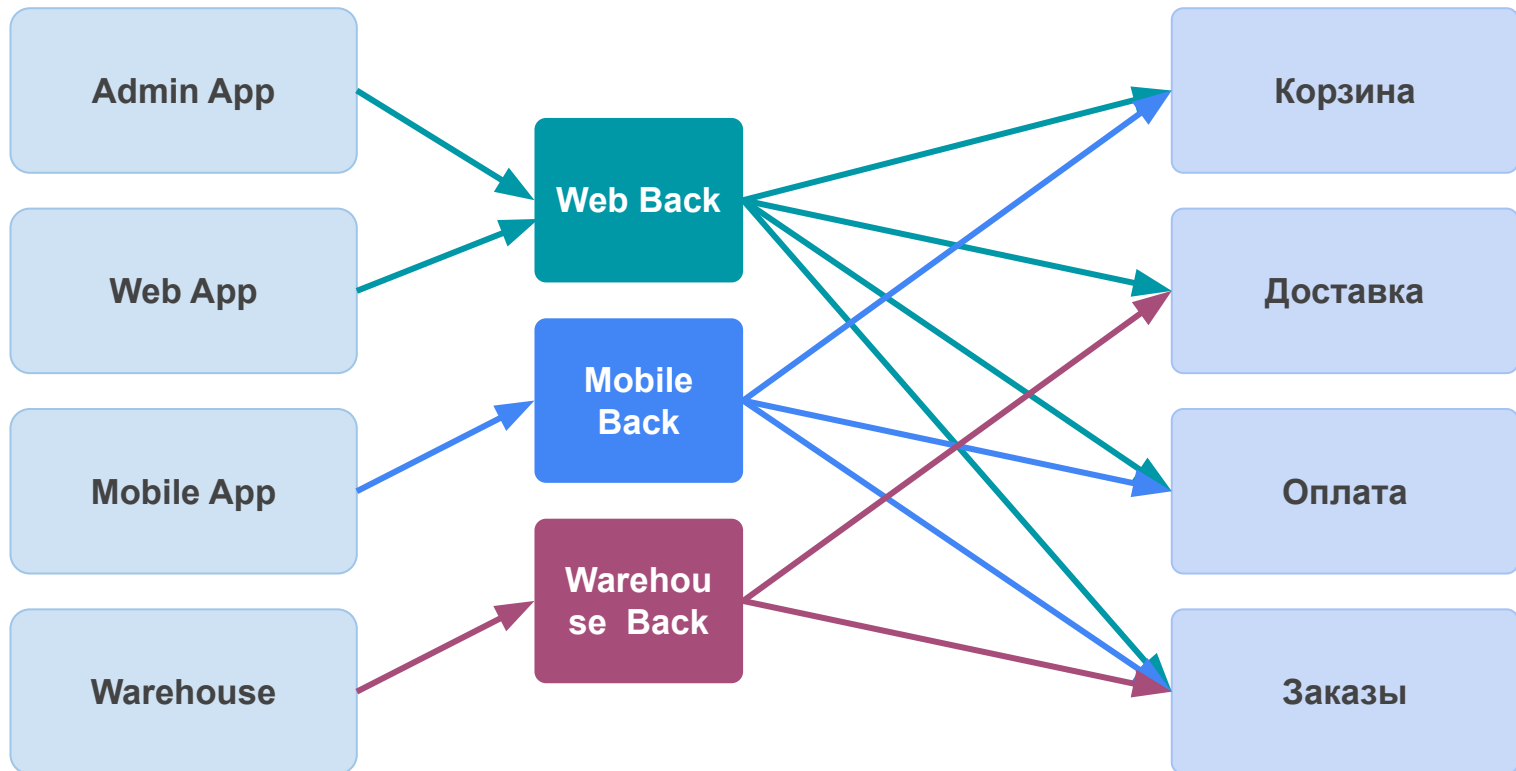
# Версионирование: версии сервисов



# API Gateway



# Backend for frontend



---

## Выводы

- Не используйте интеграцию через общую базу данных
- Начинайте с REST для request/response интеграций
- Хореография предпочтительнее чем оркестрация
- Избегайте обратно несовместимых изменений и необходимости версионировать эндпоинты



# Двенадцать факторов

---

## Двенадцать факторов



### THE TWELVE-FACTOR APP

Читать тут: <https://12factor.net/ru/>



---

# 1-4

## **I. Кодовая база**

Одна кодовая база, отслеживаемая в системе контроля версий, – множество развёртываний

## **II. Зависимости**

Явно объявляйте и изолируйте зависимости

## **III. Конфигурация**

Сохраняйте конфигурацию в среде выполнения

## **IV. Сторонние службы (Backing Services)**

Считайте сторонние службы (backing services) подключаемыми ресурсами



## 5-8

### **V. Сборка, релиз, выполнение**

Строго разделяйте стадии сборки и выполнения

### **VI. Процессы**

Запускайте приложение как один или несколько процессов не сохраняющих внутреннее состояние (stateless)

### **VII. Привязка портов (Port binding)**

Экспортируйте сервисы через привязку портов

### **VIII. Параллелизм**

Масштабируйте приложение с помощью процессов



## 9-12

### **IX. Утилизируемость (Disposability)**

Максимизируйте надёжность с помощью быстрого запуска и корректного завершения работы

### **X. Паритет разработки/работы приложения**

Держите окружения разработки, промежуточного развёртывания (staging) и рабочего развёртывания (production) максимально похожими

### **XI. Журналирование (Logs)**

Рассматривайте журнал как поток событий

### **XII. Задачи администрирования**

Выполняйте задачи администрирования/управления с помощью разовых процессов

---

# Итоги

- Разобрались с вариантами разбиения системы на сервисы
- Узнали какие бывают варианты организации взаимодействия между сервисами
- Познакомились с принципами создания независимых приложений



---

# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Михаил Триполитов**