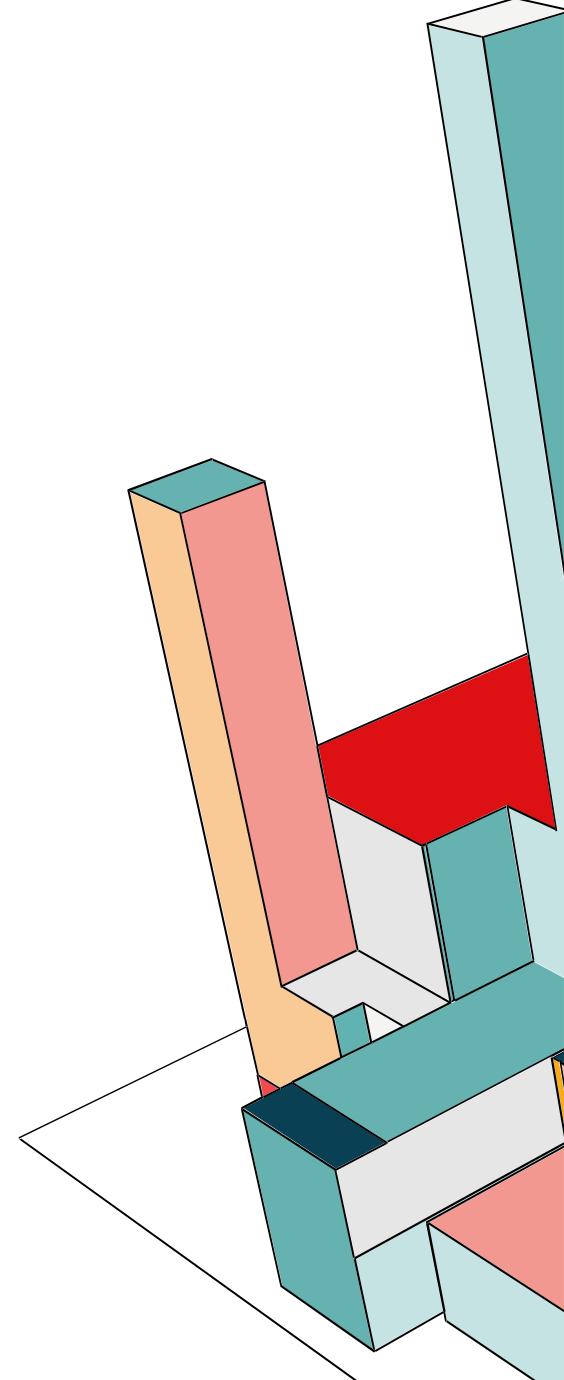
The background features a collection of 3D-style geometric shapes in various colors like red, blue, green, and yellow, arranged in a somewhat abstract, overlapping manner.

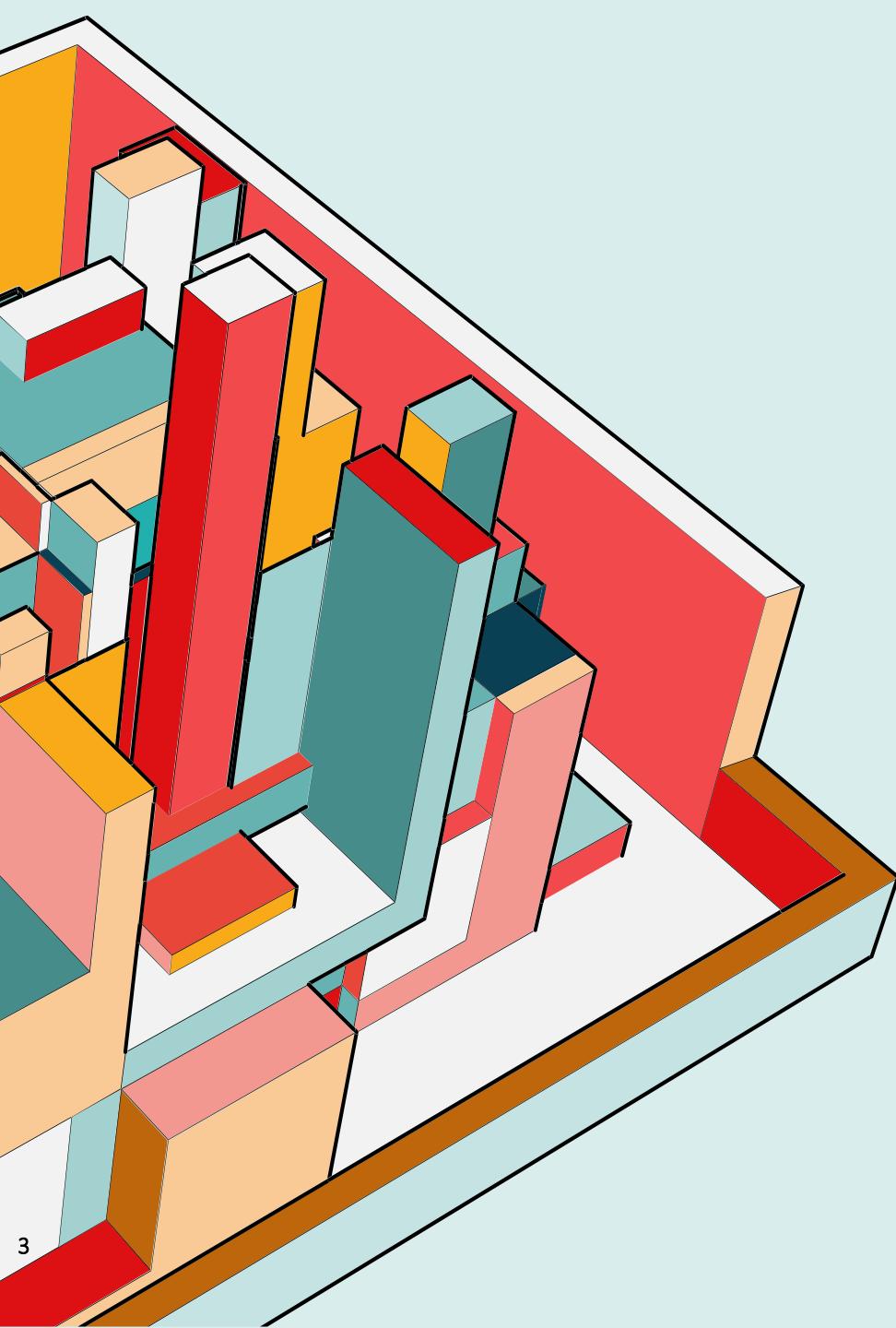
# КОНСТРУИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

# ПЛАН ЛЕКЦИИ № 10

## Клиент-серверное и межсервисное взаимодействие

1. Синхронное и асинхронное взаимодействие
2. HTTP, протоколы, архитектурные стили
3. REST API, принципы, правила
4. SOAP, что это такое, сравнение с REST
5. GraphQL
6. WebSockets, для чего и когда использовать
7. RPC (gRPC, tRPC) - удаленный вызов процедур



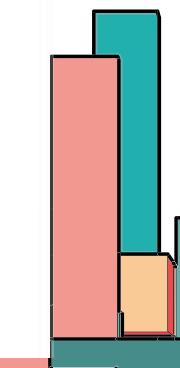
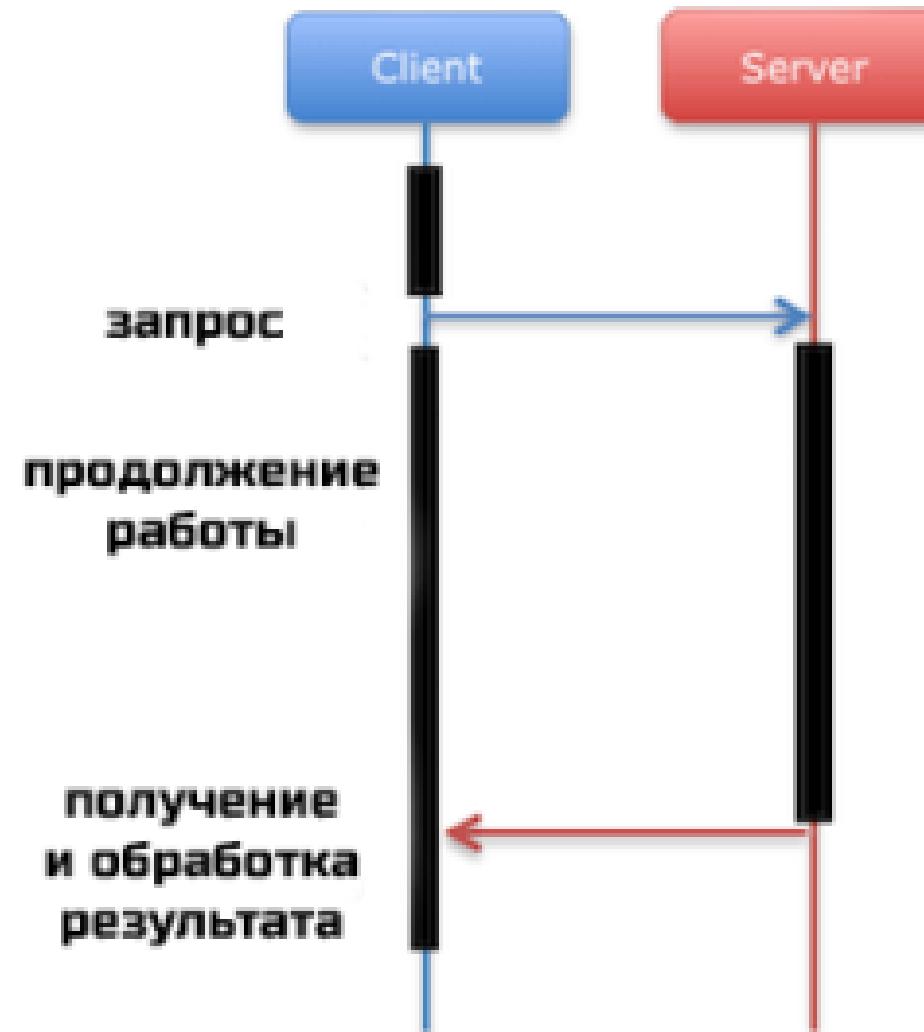


# **СИНХРОННОЕ И АСИНХРОННОЕ ВЗАЙМОДЕЙСТВИЕ**

## СИНХРОННО



## АСИНХРОННО



# СИНХРОННО



## Плюсы синхронного взаимодействия:

- Простота в реализации и отладке.
- Прозрачность. Легко отслеживать и управлять последовательностью выполнения операций.

## Минусы синхронного взаимодействия:

- Зависимость от доступности.
- Узкое место.



# СИНХРОННО



## Пример использования.

Приложение электронной коммерции может синхронно вызывать микросервис для проверки наличия товара перед оформлением заказа.

Клиентский микросервис блокируется, пока не получит ответ о наличии товара.



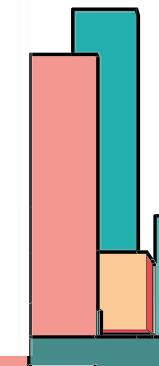
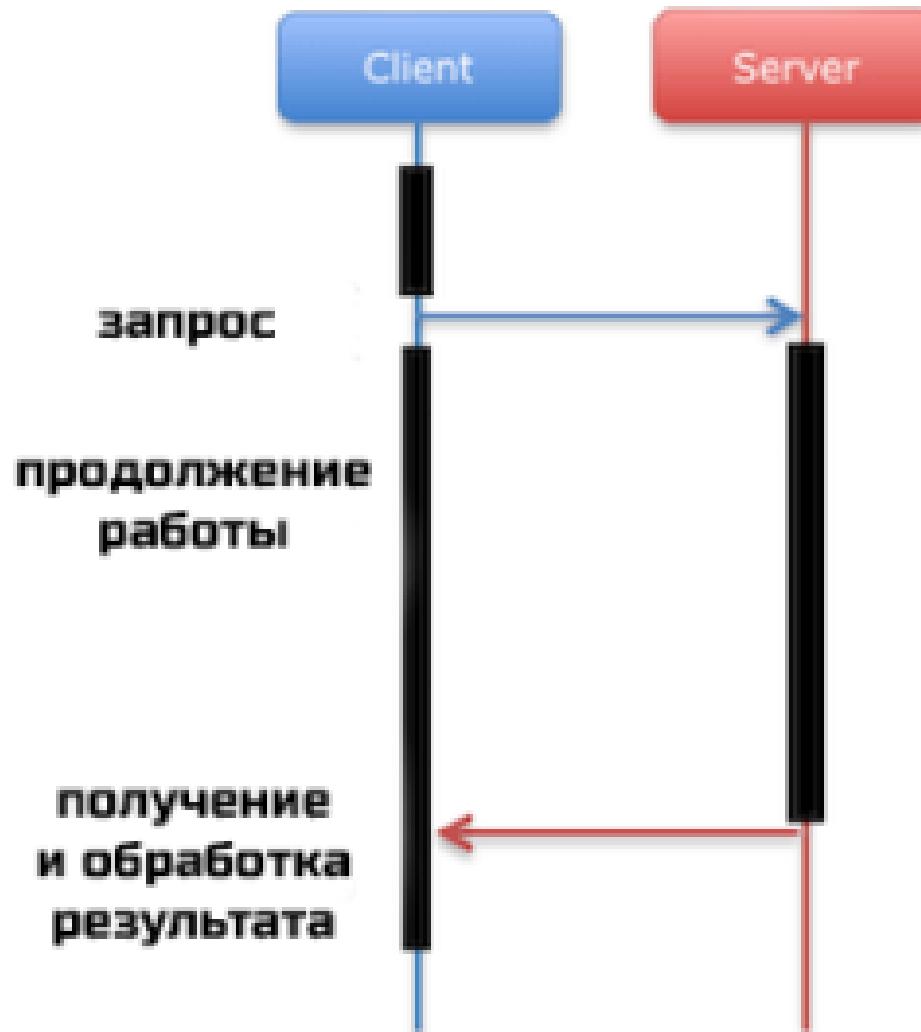
# асинхронно

## Плюсы асинхронного взаимодействия:

- Отказоустойчивость. Позволяет избежать блокировки клиентского микросервиса.
- Масштабируемость. Асинхронное взаимодействие может быть параллельным.

## Минусы асинхронного взаимодействия:

- Сложность в разработке.
- Усложнение отладки.

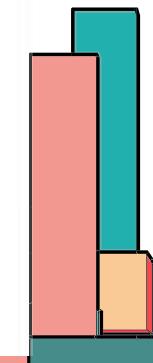
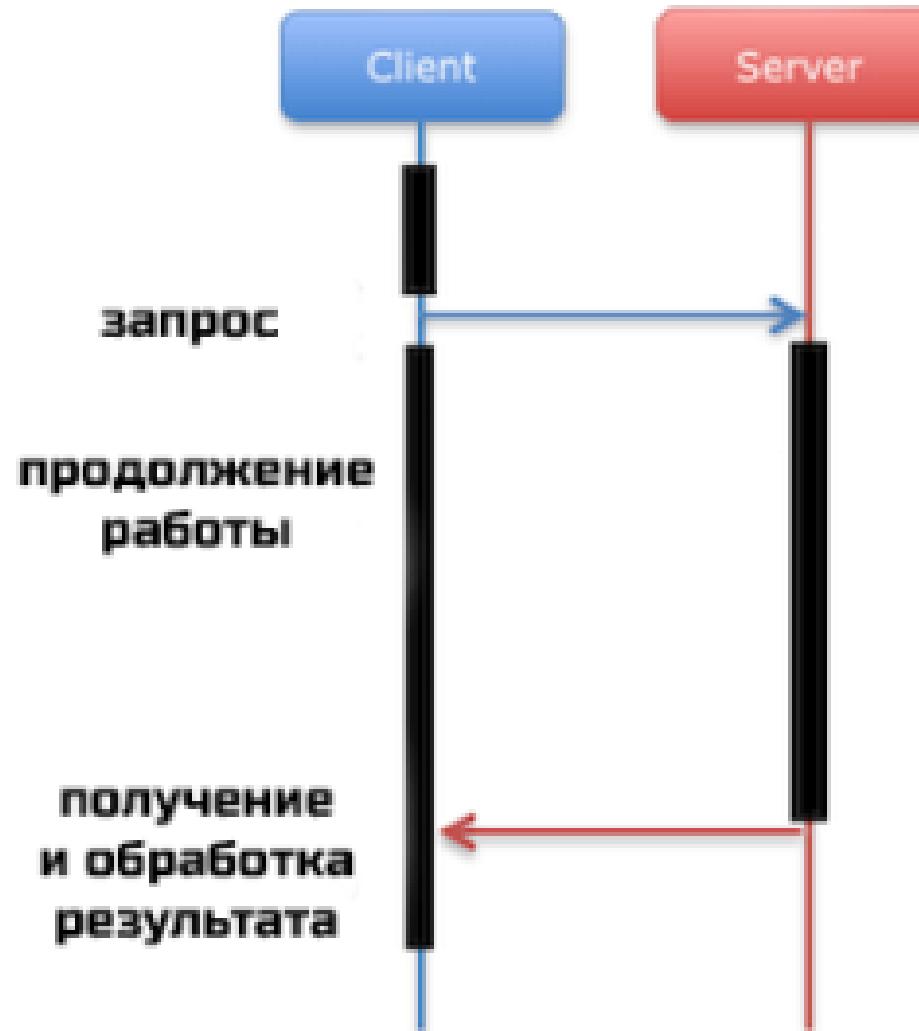


# асинхронно

## Пример использования.

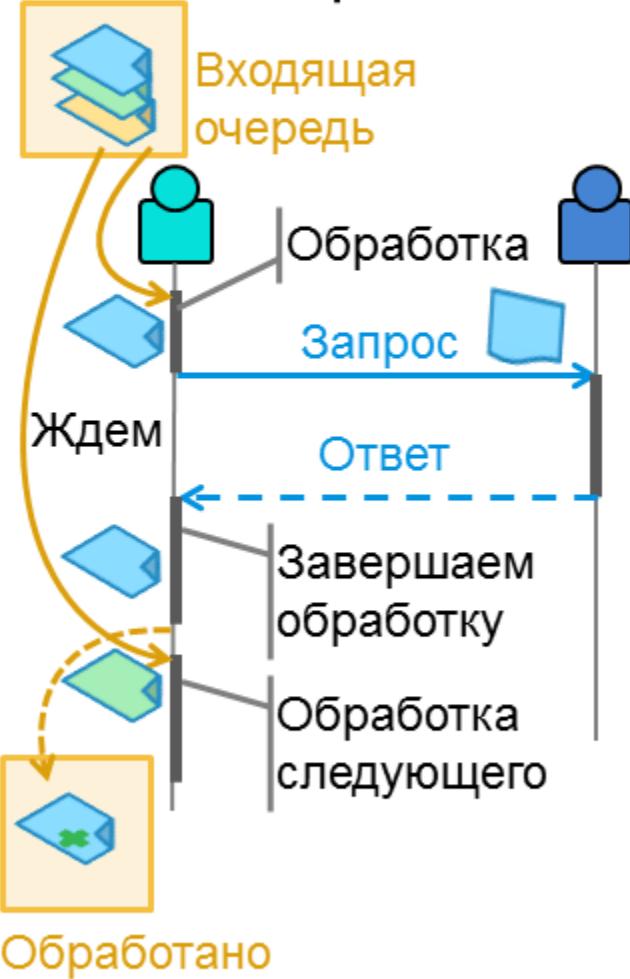
Система обработки платежей может асинхронно отправлять запрос на обработку платежа, а затем получать ответ о статусе платежа через сообщения.

Такой подход позволяет клиентскому микросервису продолжать работу без ожидания ответа платежного сервиса.

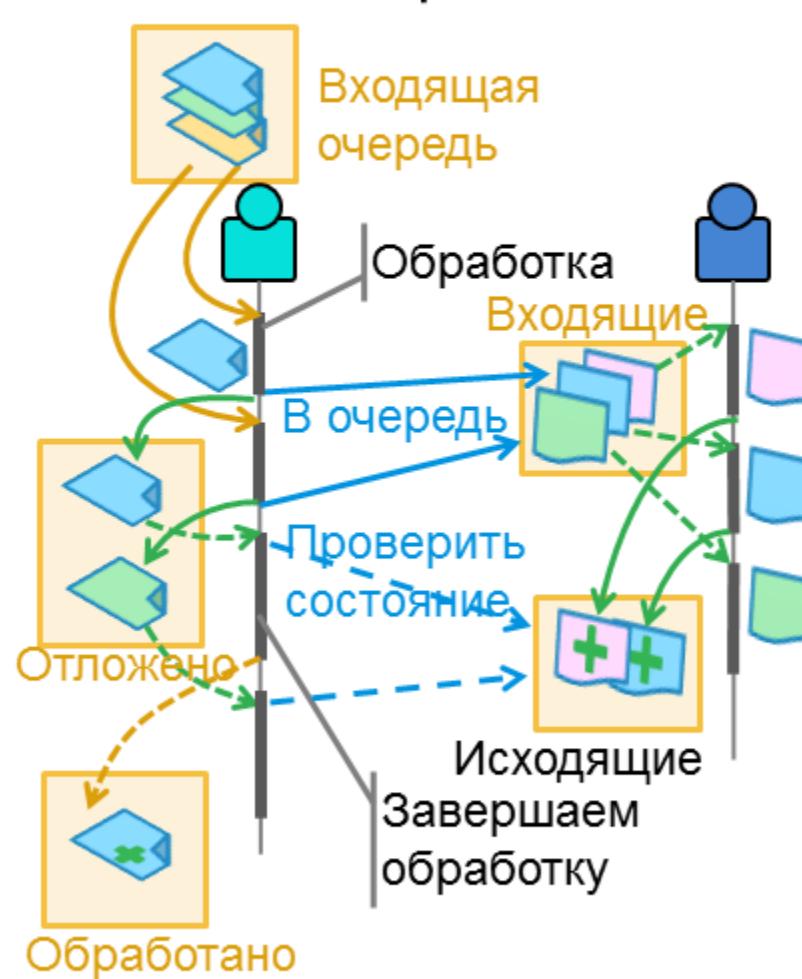


# Варианты межсервисного взаимодействия

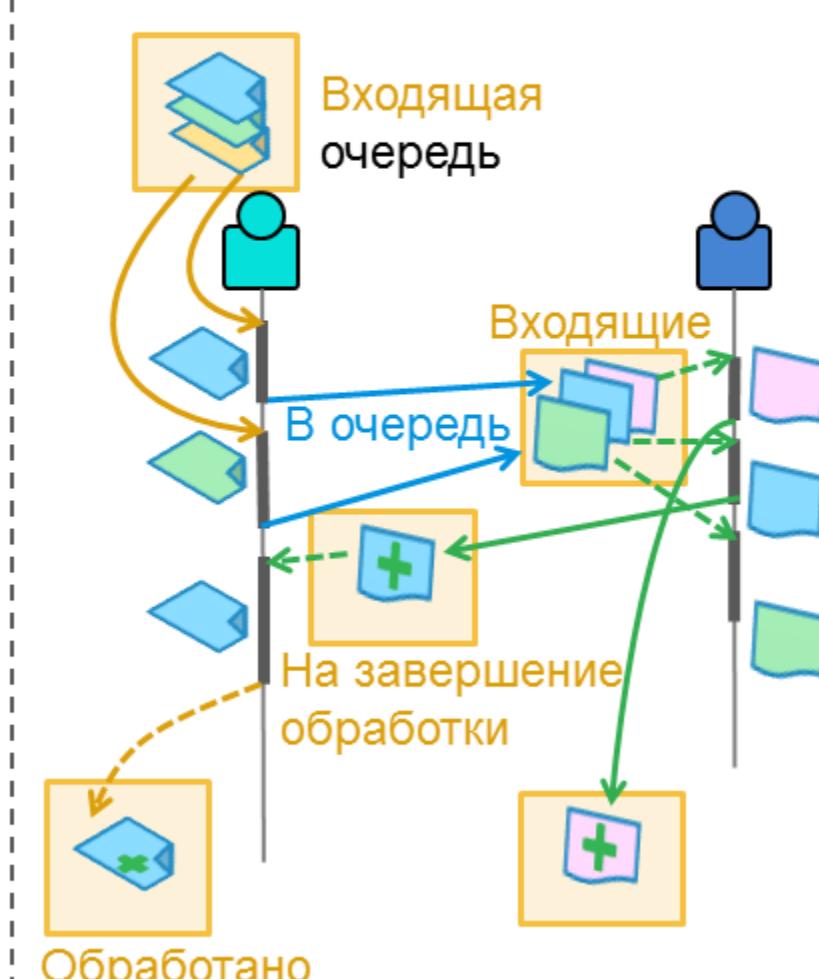
Синхронное



Асинхронное



Реактивное



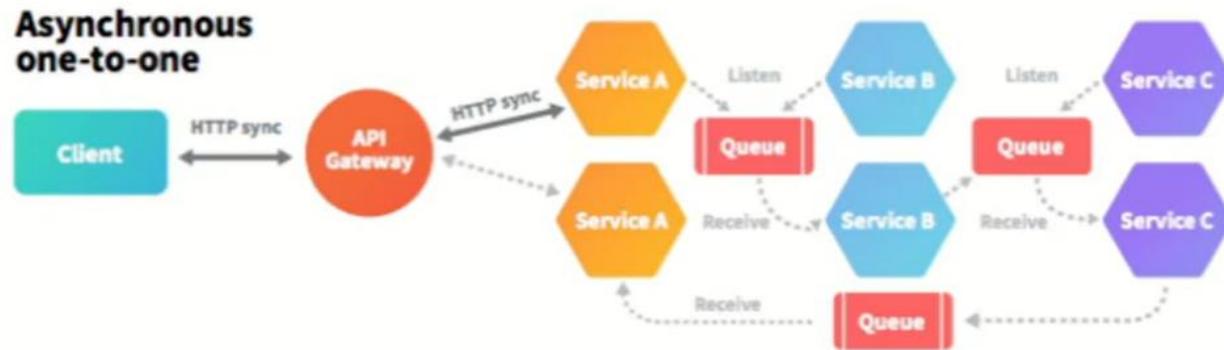
# Request-response vs message based

## Request-response:

- Проще дебаг
- Latency ниже

## Message based

- Дебаг сложнее
- Latency за счет посредников выше
- Разделение сервисов
- Роутинг
- Массовая отправка (kafka)



# ЧТО ВЫБРАТЬ?

Правильный выбор зависит от следующих факторов:

- **Время отклика.** Если требуется отклик и задержки недопустимы, синхронное взаимодействие может быть предпочтительным.
- **Надежность.** Если надежность и отказоустойчивость важны, асинхронное взаимодействие может быть предпочтительным, так как избегает блокировок и позволяет более гибко обрабатывать ошибки и отказы.
- **Производительность.** Если система требует высокой производительности и параллельной обработки запросов, асинхронное взаимодействие может быть более эффективным.

Иногда комбинация синхронного и асинхронного взаимодействия может быть наиболее эффективной.

Например, клиентский микросервис может асинхронно отправлять запрос на выполнение длительной операции и получать уведомление о завершении через сообщения, в то время как продолжает синхронно выполнять другие операции.



# Семантика взаимодействия

Следует различать семантику взаимодействия и транспортный протокол.

Request-Reply, Async, Pub/Sub – это «транспортный» уровень взаимодействия.



# Синхронное взаимодействие

**Синхронное взаимодействие** – это такое взаимодействие, при котором семантика сервиса А такова, что он не может продолжить выполнение запроса, если не получил ответа от Б.

## Например

- Сервис «заказ» не может продолжить свою работу, пока не убедится, что пользователь обладает необходимыми правами для удаления заказа
- Сервис «оформления заказа» не может закончить свою работу, пока не придет подтверждение от биллинга об оплате.

Еще это называется **функциональная зависимость**.

Когда результаты одной функции необходимы для работы другой.



# Асинхронное взаимодействие

**Асинхронное взаимодействие** – это такое взаимодействие, при котором сервис А в соответствии с семантикой предметной области может продолжить выполнение, не дожидаясь ответа от Б.

Например

- Сервису «регистраций» достаточно сообщить о факте создания нового пользователя, ему не нужен ответ от сервиса «нотификаций».
- Формирование отчетов брокера или налоговой



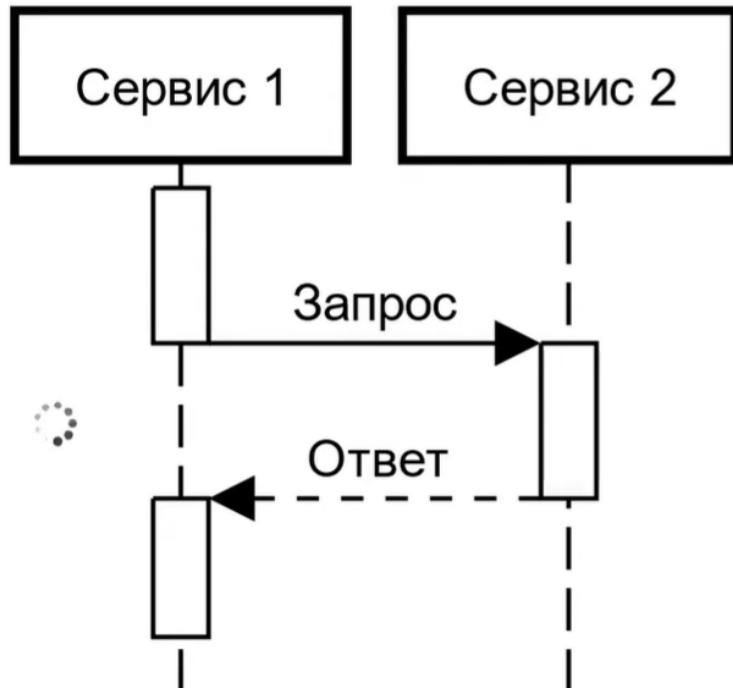
# Синхронное и асинхронное взаимодействие

Синхронное взаимодействие нельзя поменять на асинхронное просто поменяв транспортный уровень. Например, заставив всех общаться через общую очередь или по REST-у.

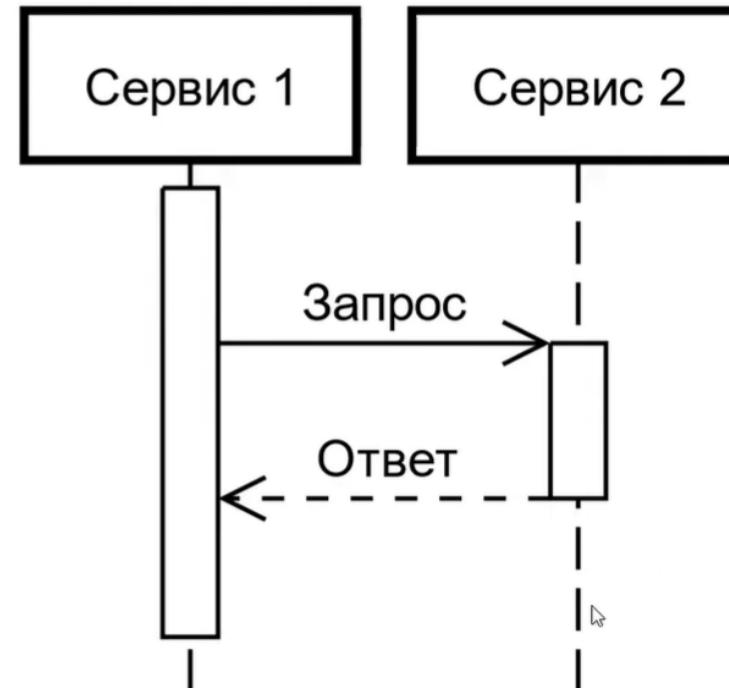
Такое изменение возможно только с изменением семантики работы сервисов, которое приводит к изменению и других характеристик системы – транзакционность, консистентность, время ответа, надежность, простота разработки.



# Что такое синхронный и асинхронный запрос?

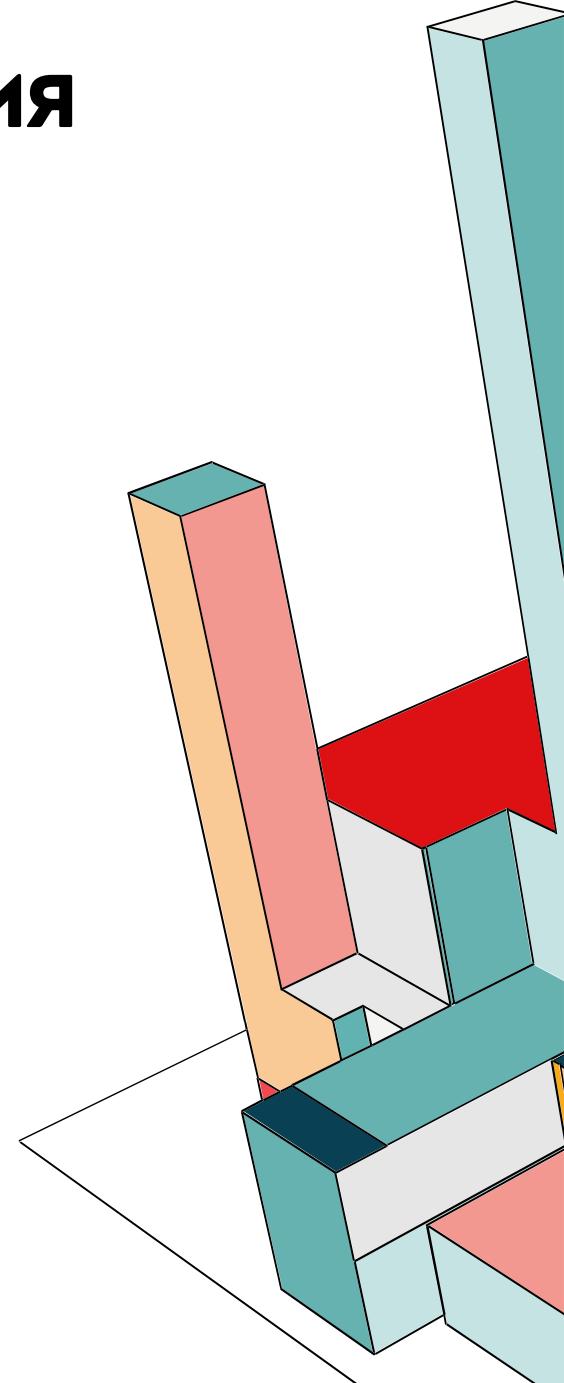
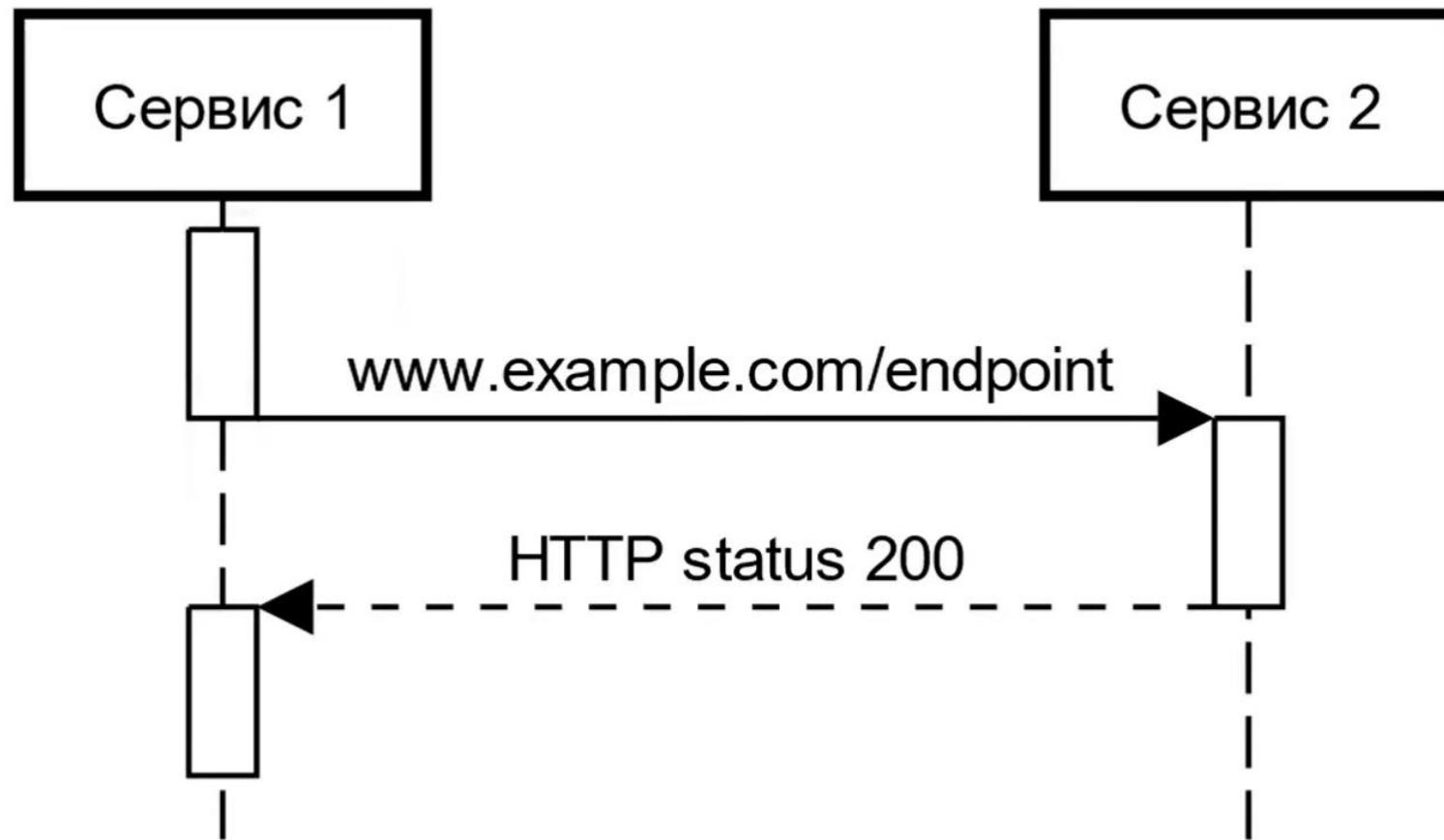


Синхронный

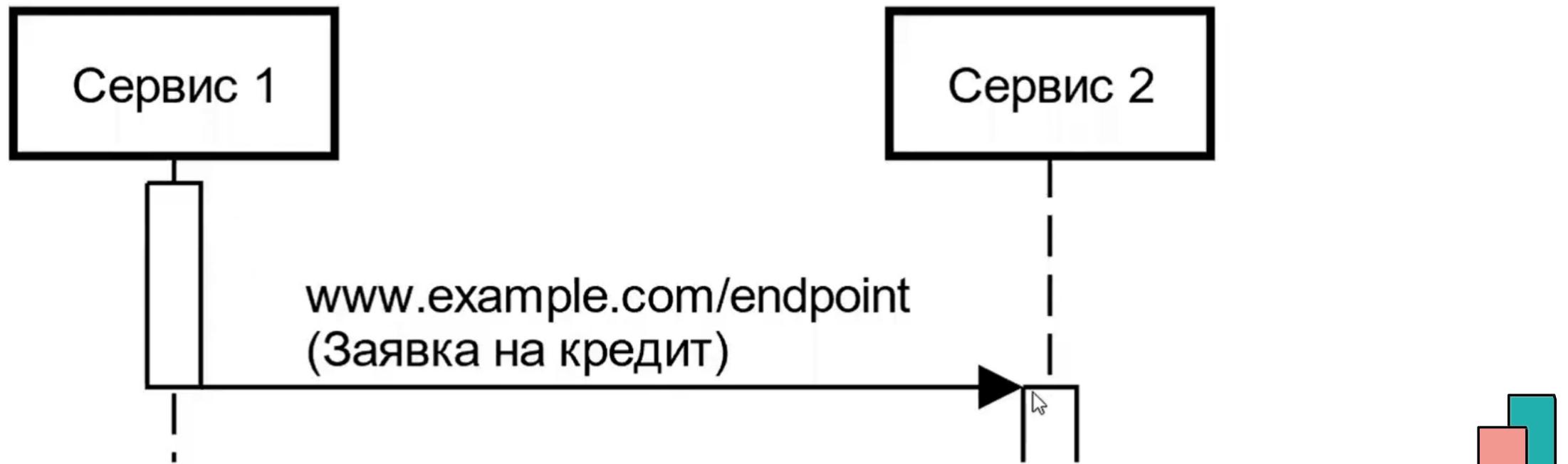


Асинхронный

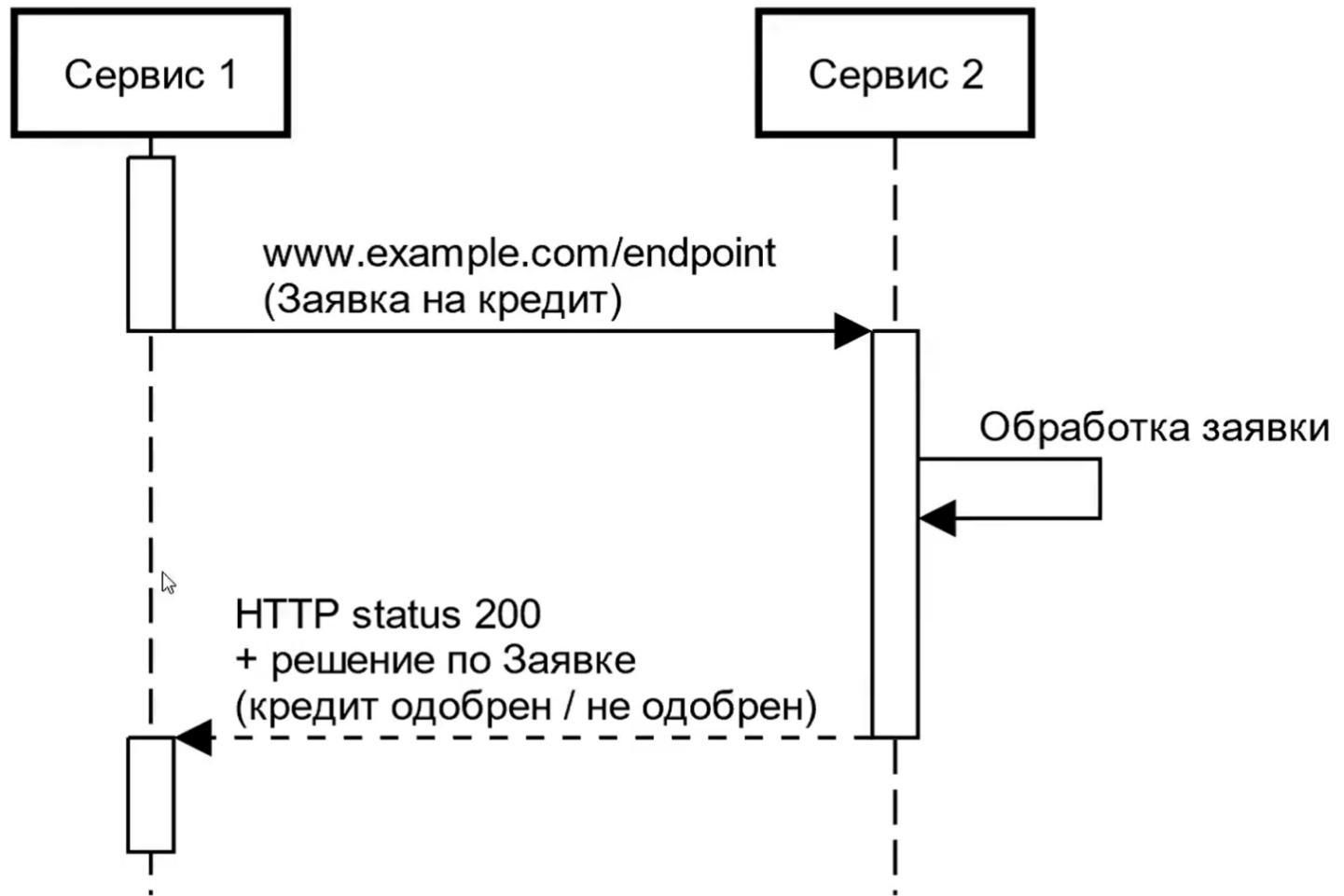
# СИНХРОННАЯ РЕАЛИЗАЦИЯ ВЗАЙМОДЕЙСТВИЯ СЕРВИСОВ ПО HTTP ПРОТОКОЛУ



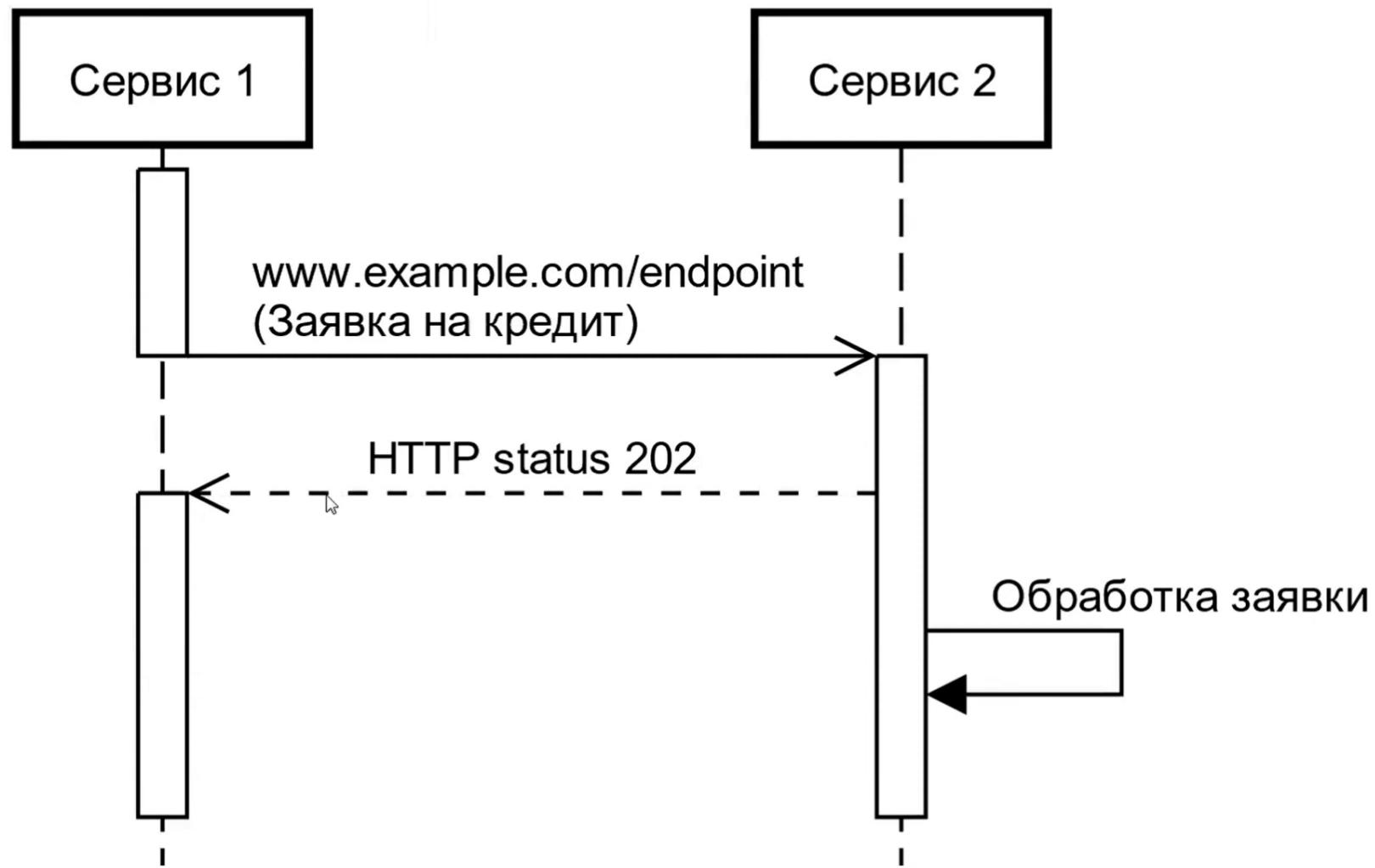
# Пример: заявка на кредит



# Заявка на кредит: синхронная реализация



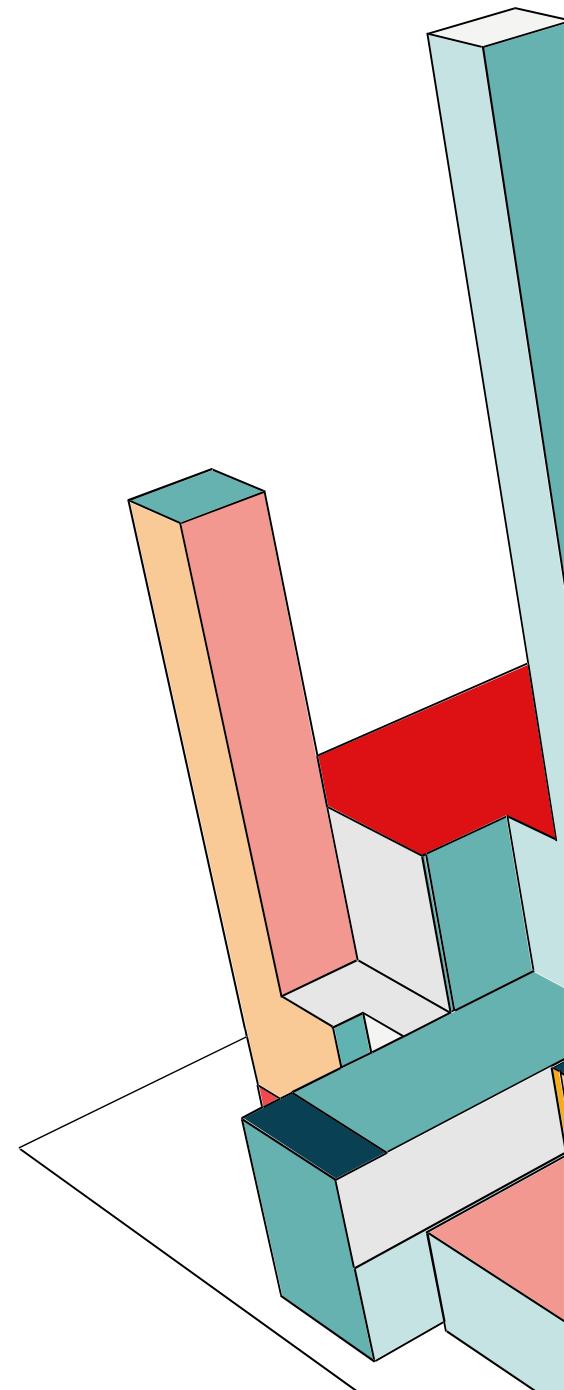
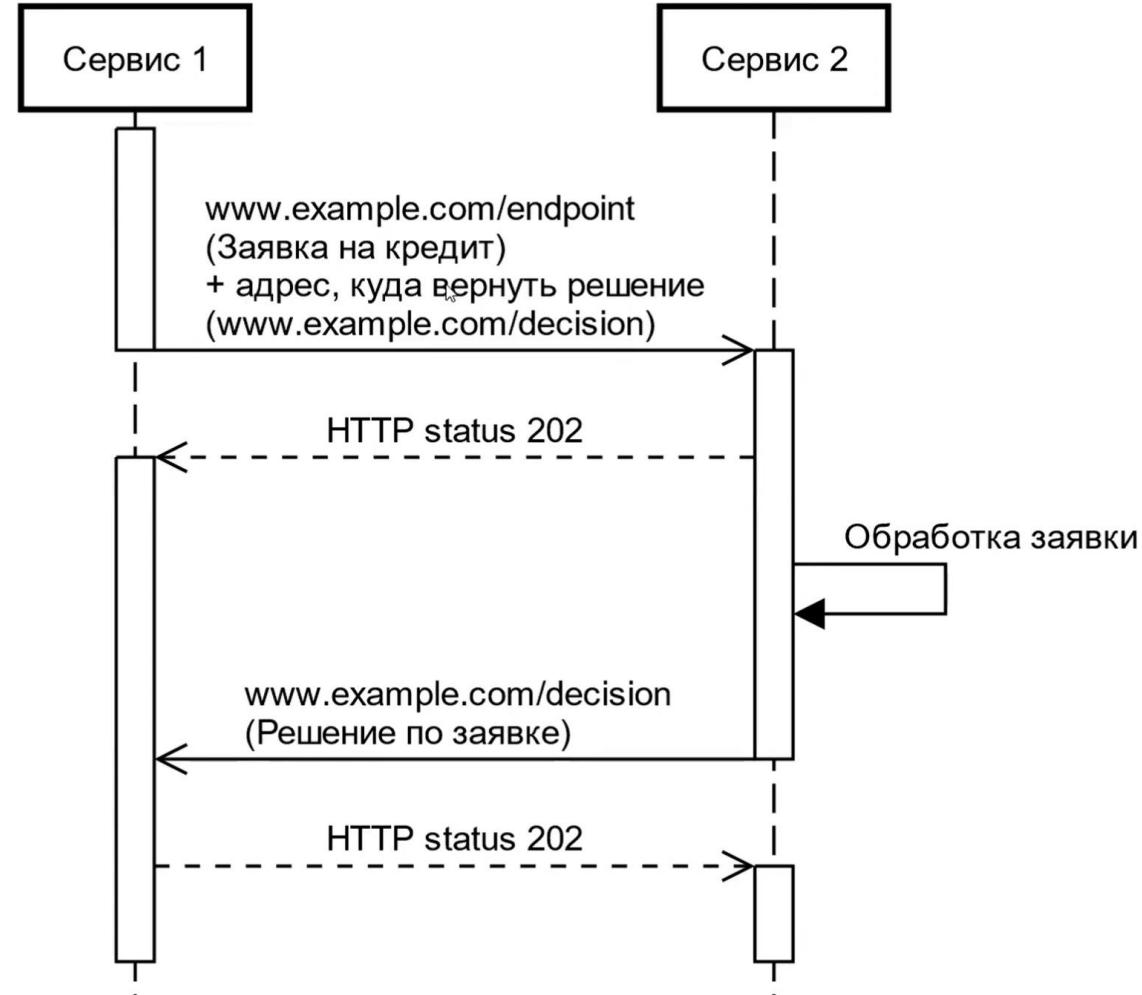
# Заявка на кредит: асинхронная реализация



# ВАРИАНТ 1

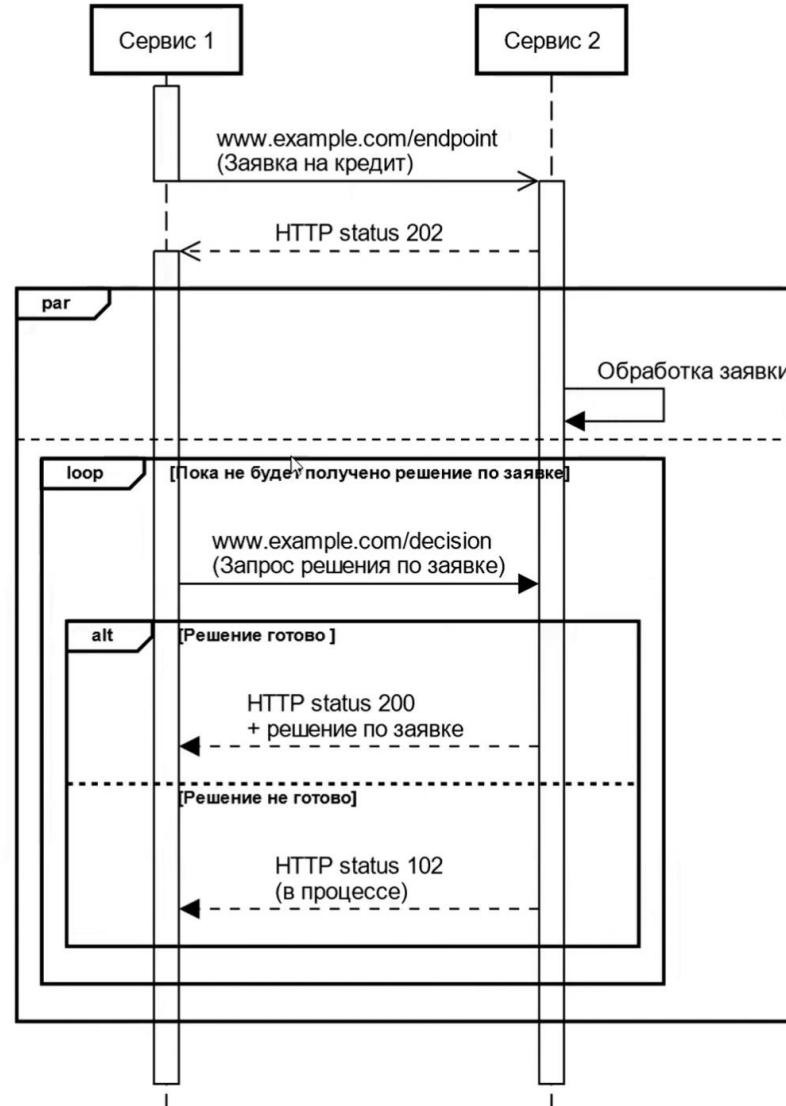
## РЕАЛИЗАЦИЯ АСИНХРОННОГО ВЗАЙМОДЕЙСТВИЯ ЧЕРЕЗ HTTP

Callback  
(обратный вызов)



# ВАРИАНТ 2

## РЕАЛИЗАЦИЯ АСИНХРОННОГО ВЗАЙМОДЕЙСТВИЯ ЧЕРЕЗ HTTP



- 1) Принять запрос и сразу вернуть HTTP status 202
- 2) Обработать запрос
- 3) В другом endpoint вернуть результат обработки запроса

# Рекомендации

- сначала определитесь с семантикой, затем уже выбирайте транспорт и протокол
- если есть возможность используйте семантически асинхронные взаимодействия
- для асинхронных взаимодействий чаще всего подходят message-base протоколы, для синхронных request-reply. Но не всегда
- лучше использовать ту технологию, в которой у вас есть экспертиза, даже если на бумаге она не идеально подходит
- service discovery и балансировка иногда могут быть болью
- трасировки очень сильно помогают



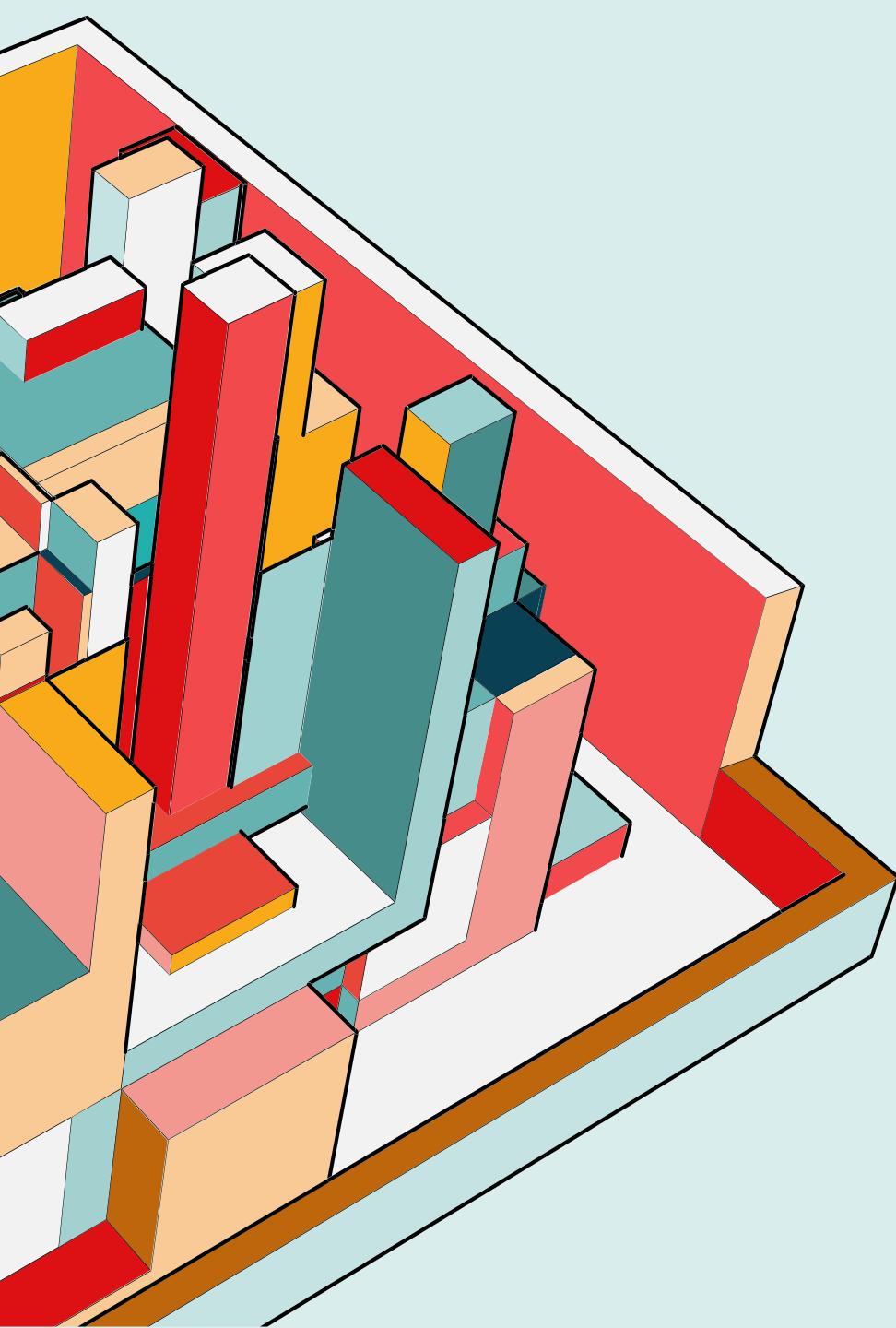
# API

**API (программный интерфейс приложения) –**

описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

API реализуется с помощью каких-то протоколов обмена

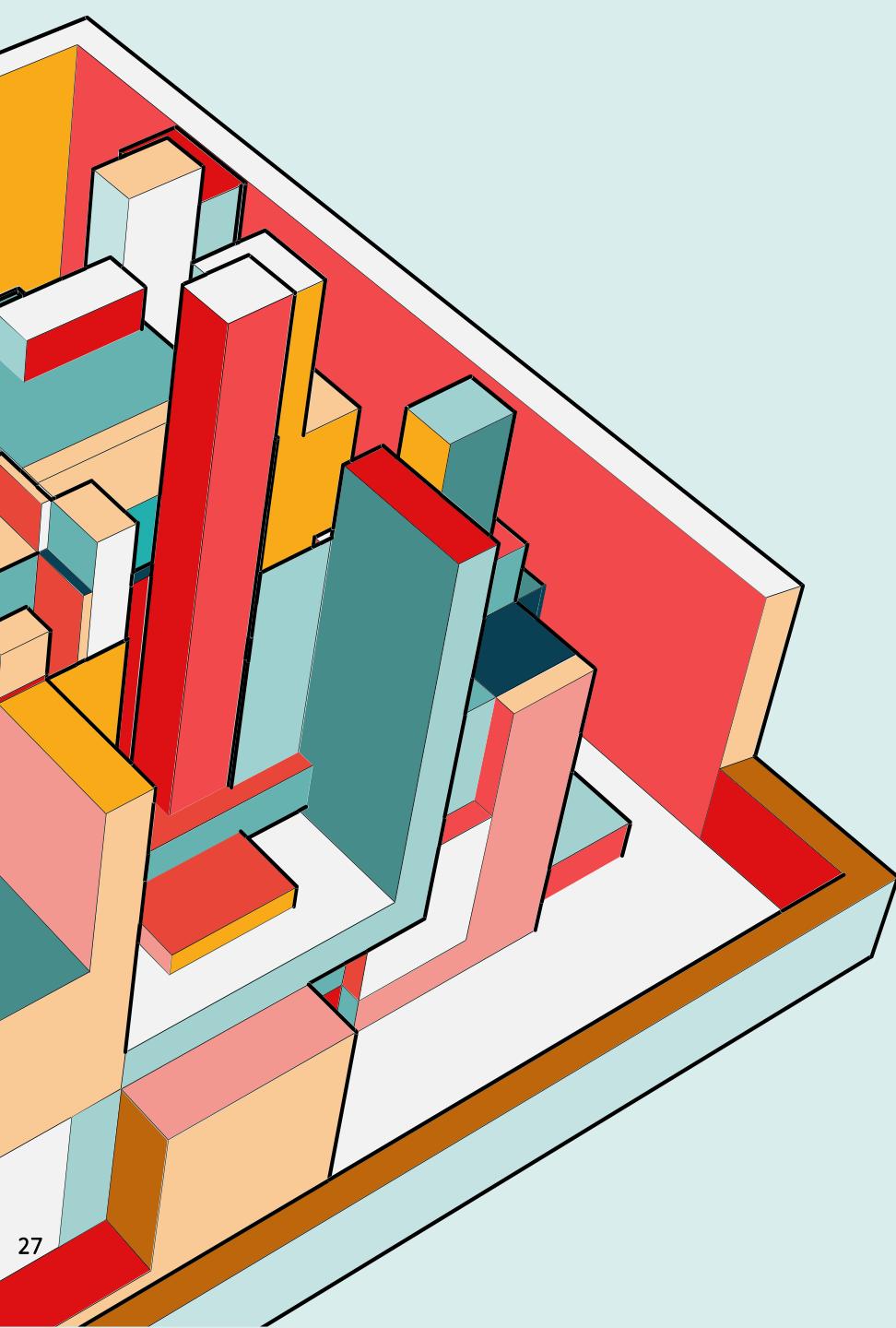




# **REST API, HTTP, SOAP, GRAPHQL, WEBSOCKETS RPC (GRPC, TRPC)**

Что это всё такое?

- 1. HTTP - СЕТЕВОЙ ПРОТОКОЛ ПРИКЛАДНОГО УРОВНЯ
- 2. REST API - АРХИТЕКТУРНЫЙ СТИЛЬ
- 3. SOAP - ПРОТОКОЛ ОБМЕНА СТРУКТУРИРОВАННЫМИ СООБЩЕНИЯМИ
- 4. GRAPHQL - ЯЗЫК ЗАПРОСОВ
- 5. WEBSOCKETS - СЕТЕВОЙ ПРОТОКОЛ
- 6. RPC (GRPC, TRPC) - УДАЛЕННЫЙ ВЫЗОР ПРОЦЕДУР



**HTTP**

протокол

# HTTP

HTTP,DNS,DHCP,FTP	Уровень приложений
TCP,UDP	Транспортный уровень
IPv4,IPv6, ICMPv4,ICMPv6	Межсетевой уровень
PPP,Frame Relay, Ethernet	Уровень сетевого доступа

# HTTP

Тексты

1

Файлы

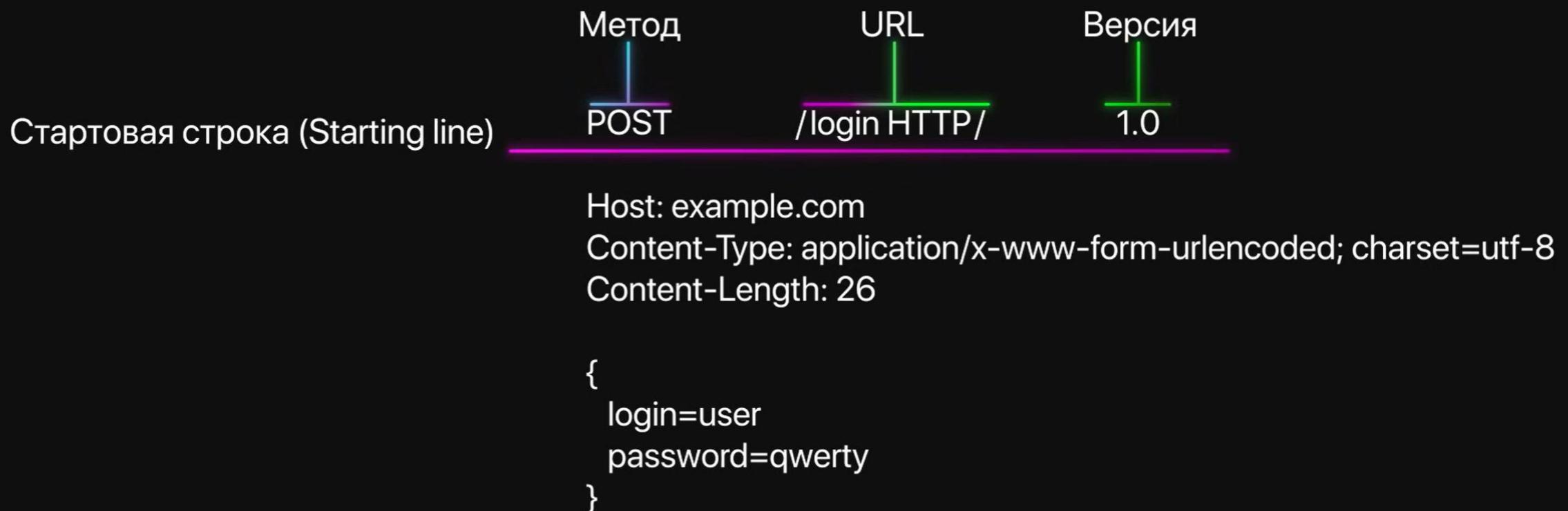
2

html/xml/json

3

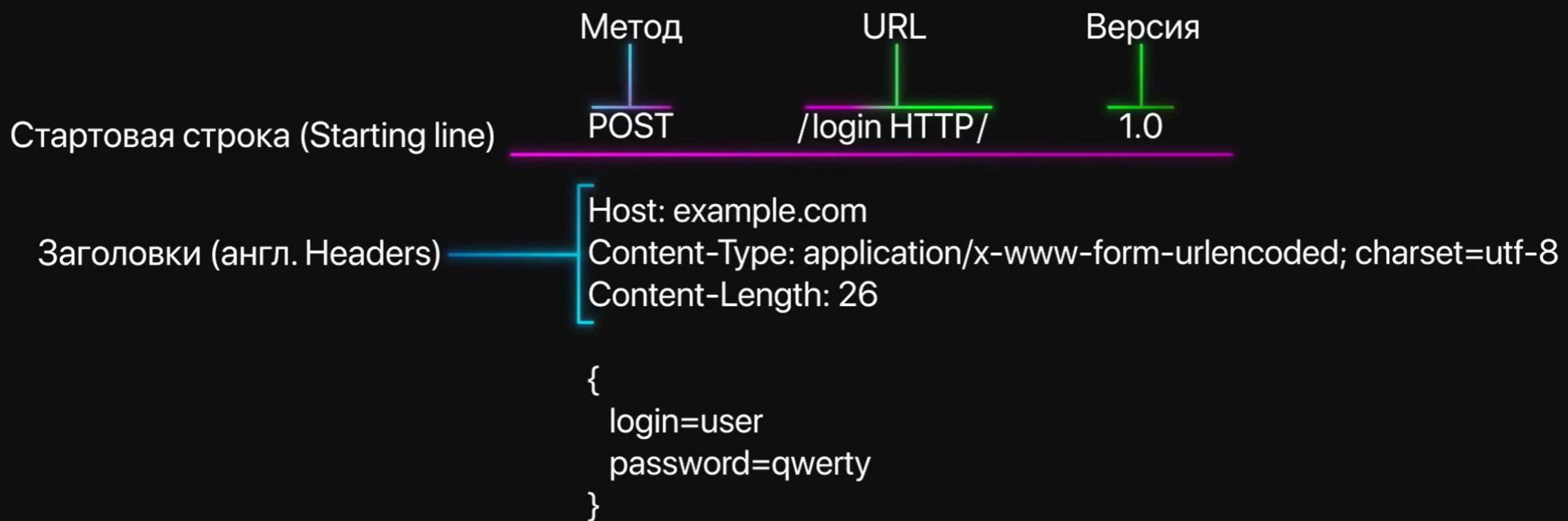
# HTTP - структура

## Пример запроса



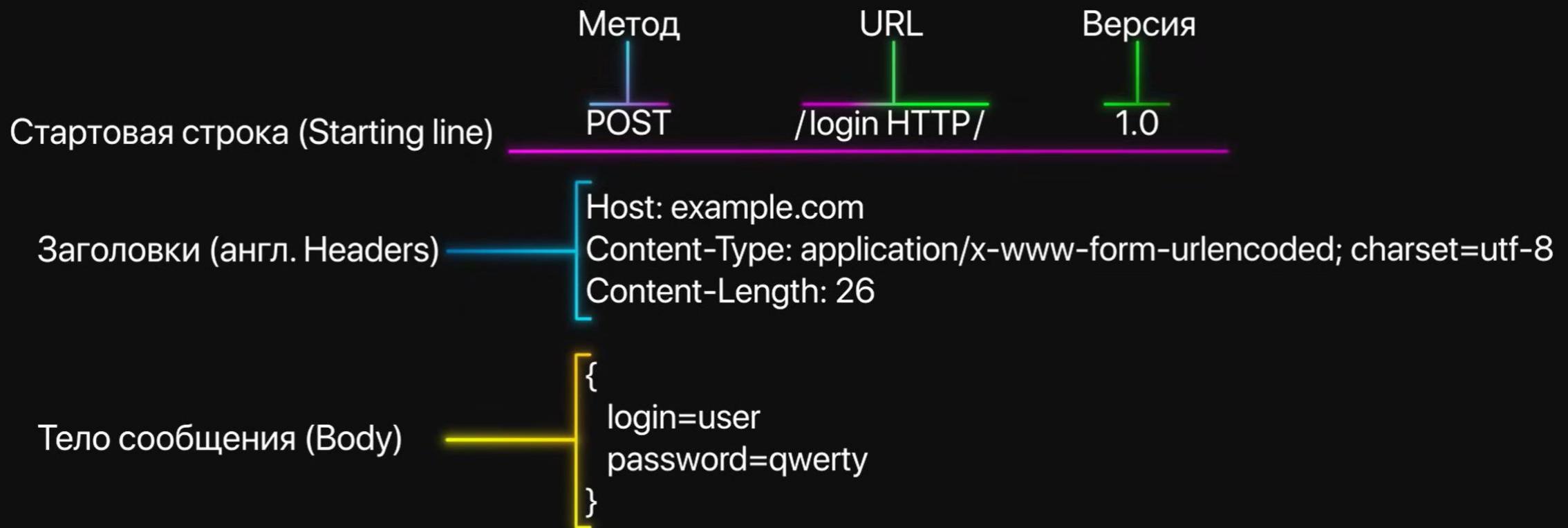
# HTTP - структура

## Пример запроса



# HTTP – структура

## Пример запроса



# Запрос



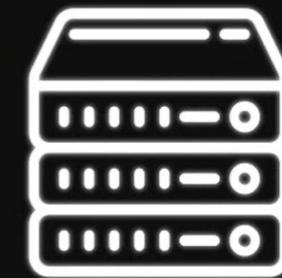
POST /login HTTP/1.0

Host: example.com  
Content-Type: application/json  
Content-Length: 26

```
{  
    login=user  
    password=qwerty  
}
```

Стартовая строка (Starting line)

# Сервер



HTTP/1.1 200 OK

Content-Type: application/json  
Connection: Closed

```
{  
    message: 'success login',  
    user : {id: '1', username ... }  
}
```

Строка статуса (status line)

# Ответ



## Основные методы

GET - получение ресурса

POST - передача данных (создание ресурса)

PUT - обновление ресурса

PATCH - обновление фрагмента ресурса (частичное)

DELETE - удаление ресурса

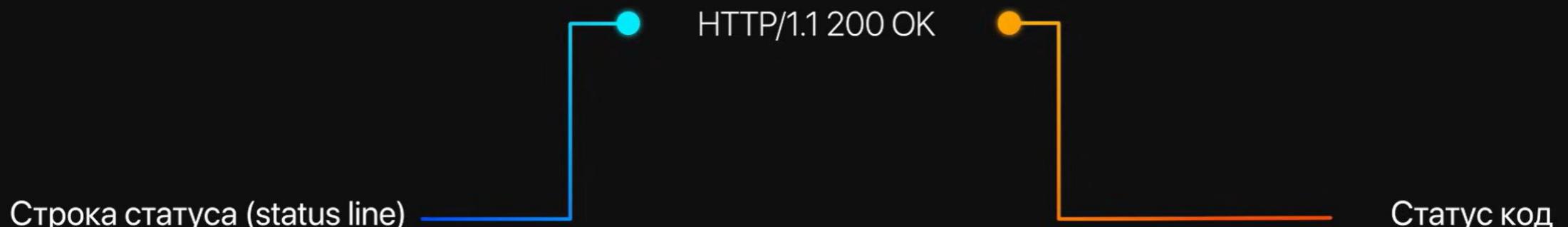
1xx: Informational (информационные)

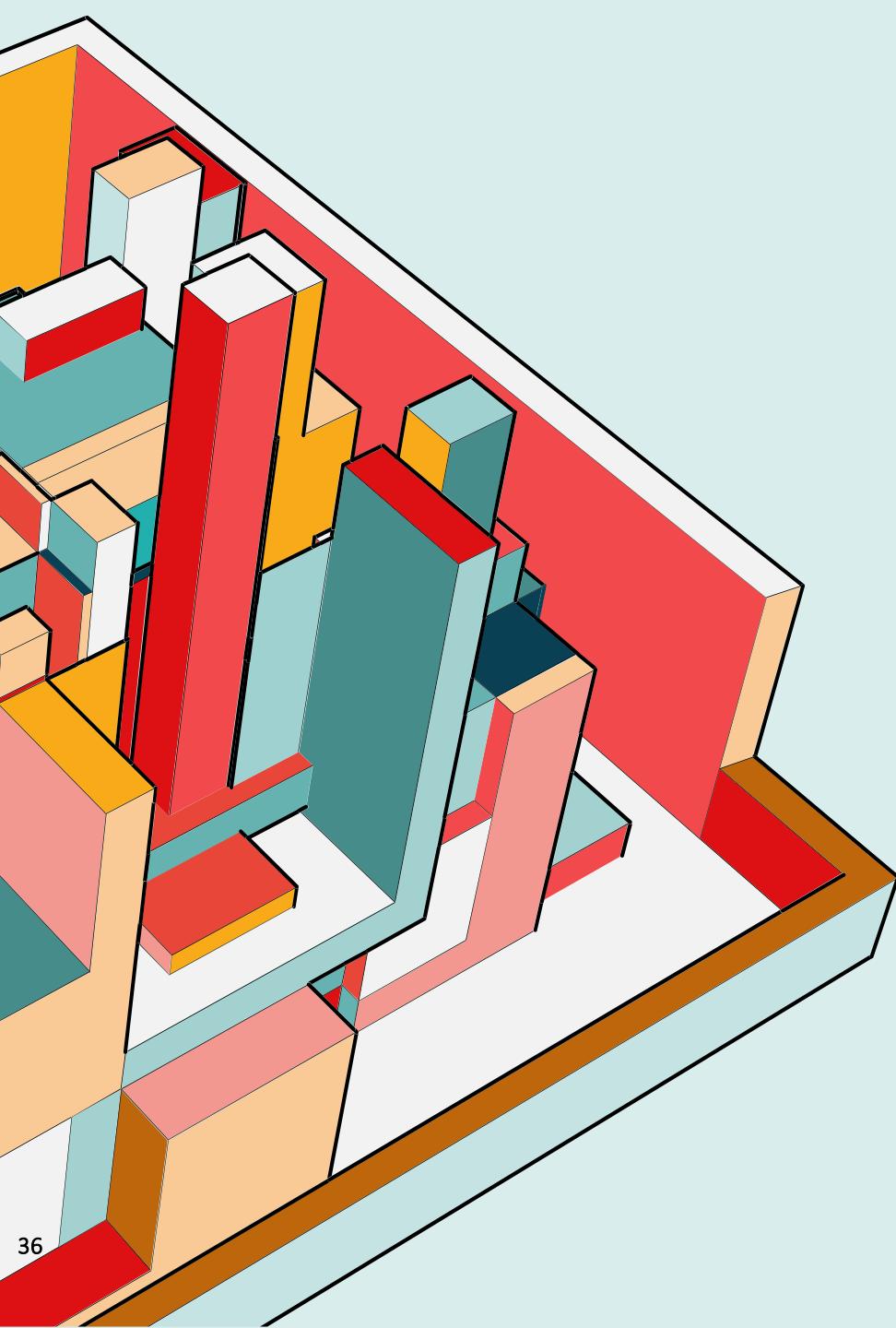
2xx: Success (успешно)

3xx: Redirection (перенаправление)

4xx: Client Error (ошибка клиента)

5xx: Server Error (ошибка сервера)





# **REST API**

архитектурные стили

# API

[ Application Programming Interface ]



Программа 1

Общение происходит с помощью API



Программа 2

# API

## [ Application Programming Interface ]

Получить актуальный  
курс валют

Конвертирация валют

Получить прогноз погоды  
на неделю



Программа 2

# API

## [ Application Programming Interface ]

Получить прогноз погоды  
на неделю

**method и url:**

GET /weather

**query params:**

date - string

city - string

**response body:**

```
{  
  weather: [  
    {  
      time: string,  
      temperature: string,  
      rainfall: boolean  
    }  
  ]  
}
```



Программа 2

# REST API

Протокол



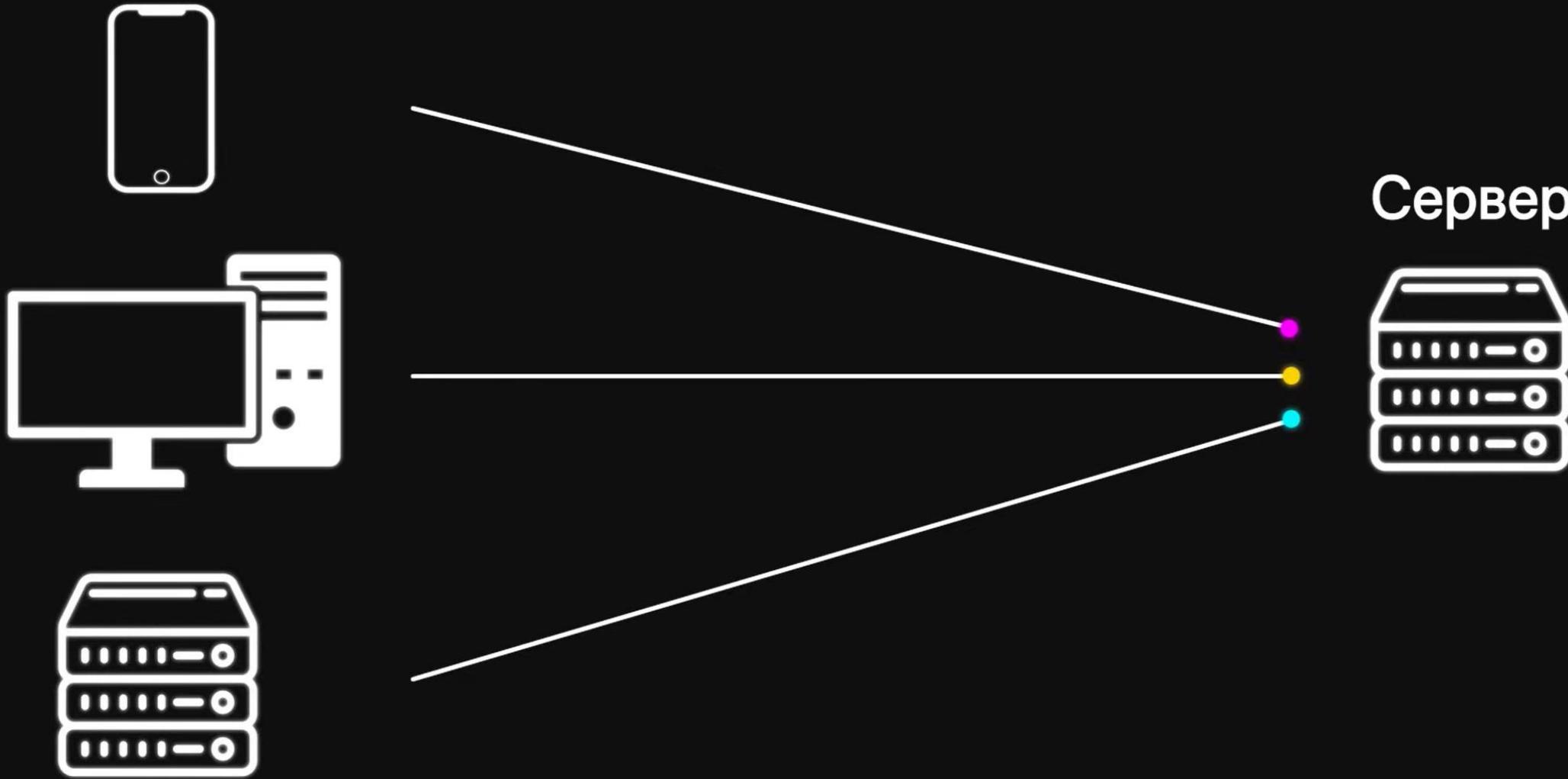
# REST API

## Архитектурный стиль

Набор правил  
для серверного приложения



# REST API Концепции



# REST API Концепции

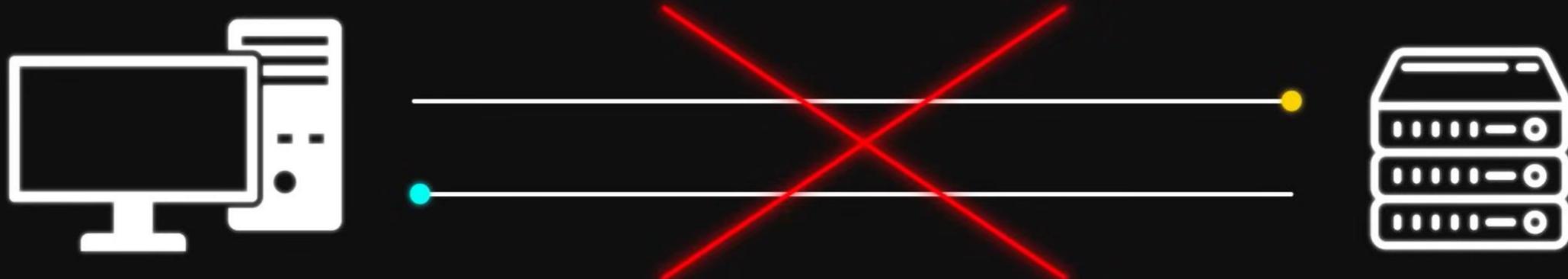


# REST API Концепции

**Stateless**  
**Отсутствие состояния**

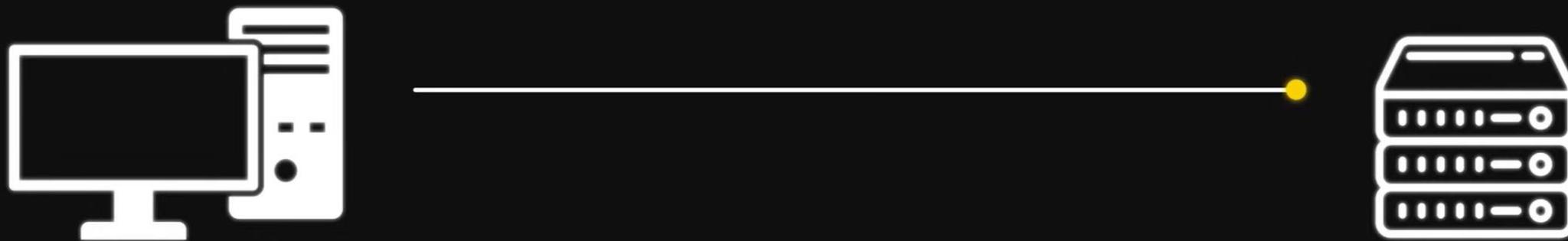
# REST API Концепции

Сервер не запоминает состояние клиента



# REST API Концепции

Клиент должен отправить вновь всю информацию, необходимую для выполнения запроса



# REST API Концепции

Единообразный  
унифицированный интерфейс

# REST API Концепции

CRUD (create, read, update, delete)

1.Добавить новый товар

2.Удалить товар

3.Обновить информацию о товаре

4.Получить все товары

5.Получить полную информацию  
по конкретному товару

Интернет магазин



# REST API Концепции

CRUD (create, read, update, delete)

	http method	URI
1.Добавить новый товар	POST	/products
2.Удалить товар	DELETE	/products
3.Обновить информацию о товаре	PUT/PATCH	/products
4.Получить все товары	GET	/products
5.Получить полную информацию по конкретному товару	GET	/products/{id}

# REST API Концепции

С помощью GET запроса  
получить данные



# REST API Концепции

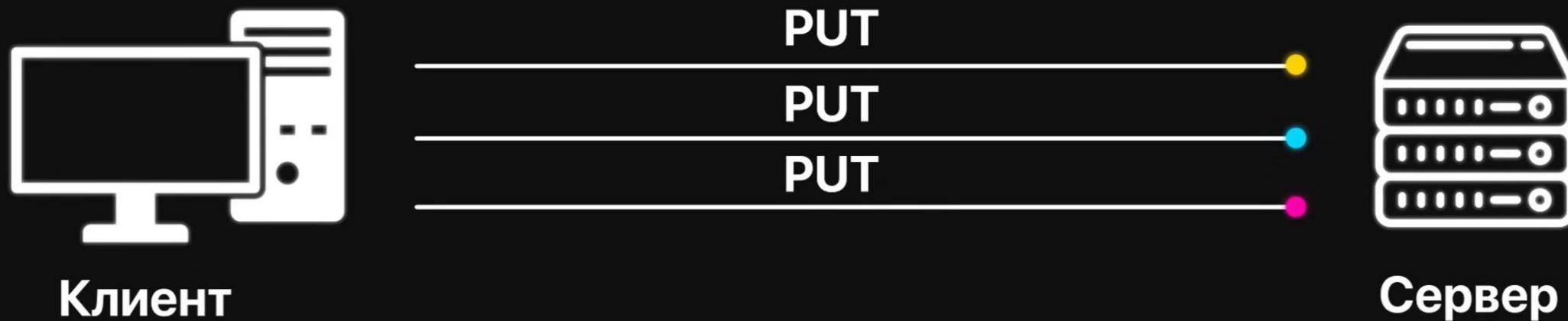
С помощью GET запроса  
**ИЗМЕНИТЬ/УДАЛИТЬ** данные



# REST API Концепции

PUT - идемпотентен

# REST API Концепции



# REST API Концепции

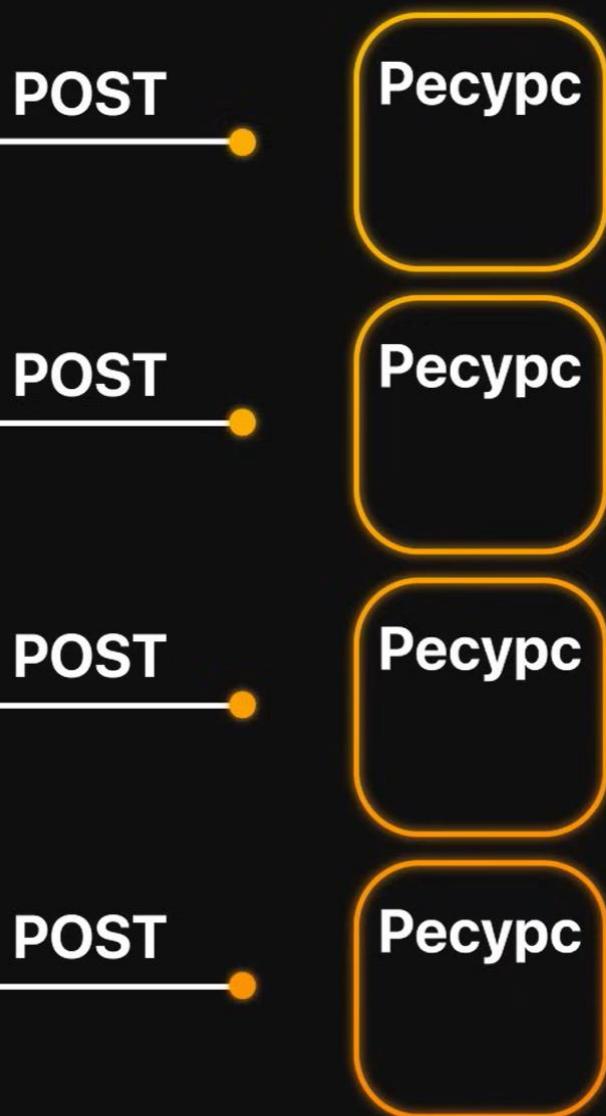
Сервер должен менять ресурс, но  
не создавать дубликаты



# REST API Концепции

Идемпотентность — это свойство, которое означает, что повторный идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект, не изменяющий состояние сервера. Корректно реализованные методы GET, PUT и DELETE идемпотентны, но не метод POST.

# REST API Концепции



# REST API Концепции

PUT age: 20

PUT age: 20

PUT age: 20

Ресурс

age: 20

# REST API Концепции

## Кеширование

5

# REST API Концепции

GET и POST запросы  
могут быть кэшируемыми

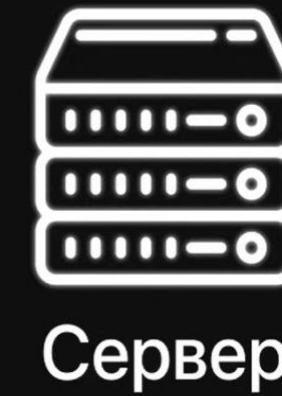
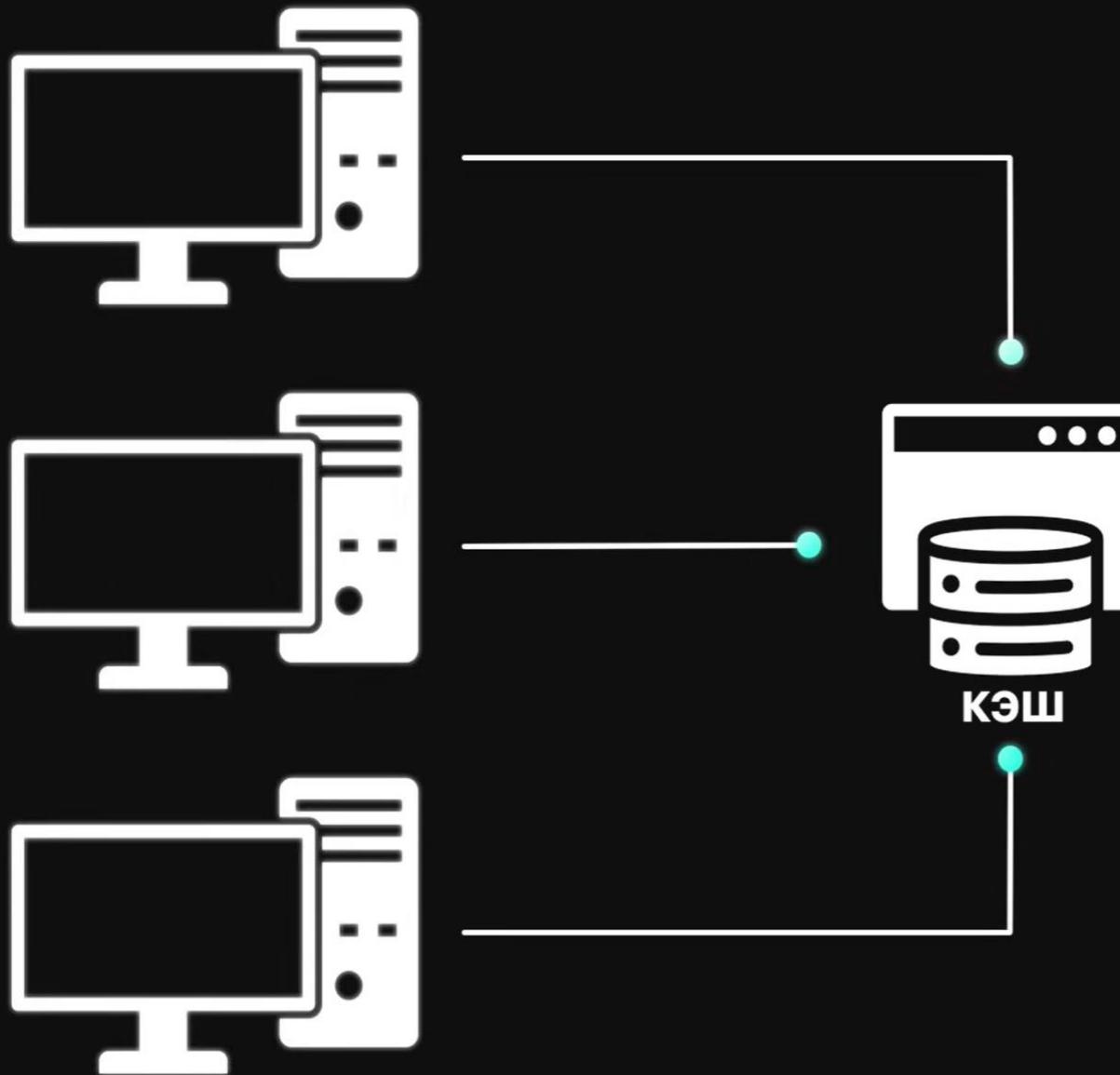
PUT и DELETE  
не кэшируются



# REST API Концепции



# REST API Концепции



5

# REST API Концепции

**Формат обмена данными**

# REST API Концепции

## JSON

```
{"empinfo":  
{  
    "employees": [  
        {  
            "name": "James Kirk",  
            "age": 40,  
        },  
        {  
            "name": "Jean-Luc Picard",  
            "age": 45,  
        },  
        {  
            "name": "Wesley Crusher",  
            "age": 27,  
        }  
    ]  
}
```

## XML

```
<empinfo>  
    <employees>  
        <employee>  
            <name>James Kirk</name>  
            <age>40</age>  
        <employee>  
            <name>Jean-Luc Picard</name>  
            <age>45</age>  
        </employee>  
        <employee>  
            <name>Wesley Crusher</name>  
            <age>27</age>  
        </employee>  
    </employees>  
</empinfo>
```

# REST API Концепции

Версионирование

# REST API Концепции

/api/users GET

/api/users POST

/api/users PUT

/api/users DELETE

/api/users GET

/api/users POST

/api/users PUT

~~/api/users DELETE~~

# REST API Концепции

/api/users GET

/api/users POST

/api/users PUT

/api/users DELETE

/api/v2/users GET

/api/v2/users POST

/api/v2/users PUT

меняется формат данных

/api/**v3**/users GET

/api/**v3**/users POST

/api/**v3**/users PUT

Сохраняем обратную совместимость за счет версионирования

# REST API Концепции

## Документирование

# REST API Концепции

## Open API Спецификация

```
openapi: "3.0.0"
info:
  title: Simple API overview
  version: 2.0.0
paths:
  /:
    get:
      operationId: listVersionsv2
      summary: List API versions
      responses:
        "200":
          description: |-  
          200 response
          content:
            application/json:
              examples:
                foo:
                  value:
                    {
                      "versions": [
                        {
                          "status": "CURRENT",
                          "updated": "2011-01-21T12:33:21Z",
                          "id": "v2.0",
                          "links": [
                            ]
                          }
                        ]
                      ]
                    }
```

# REST API Концепции

## Swagger

Набор инструментов использующий  
спецификацию Open API

# REST API Концепции

**pet : Operations about pets**

Show/Hide | List Operations | Expand Operations | Raw

**GET** /pet/{petId} Find pet by ID

**DELETE** /pet/{petId} Deletes a pet

**PATCH** /pet/{petId} partial updates to a pet

**POST** /pet/{petId} Updates a pet in the store with form data

OFF !

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
petId	(required)	ID of pet that needs to be updated	path	string
name		Updated name of the pet	form	string
status		Updated status of the pet	form	string

**Response Messages**

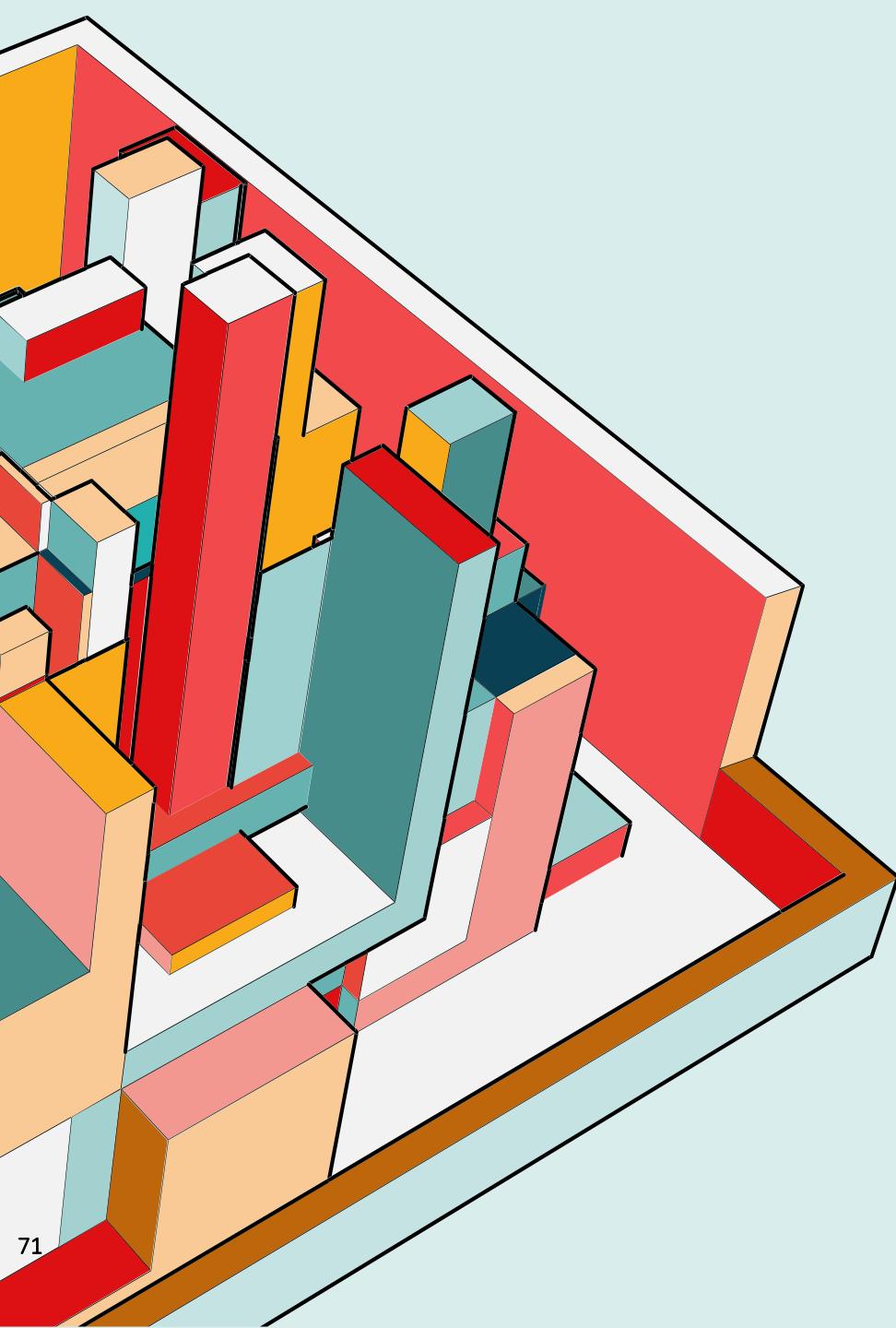
HTTP Status Code	Reason	Response Model
405	Invalid input	

[Try it out!](#)

**POST** /pet Add a new pet to the store

**PUT** /pet Update an existing pet

**GET** /pet/findByStatus Finds Pets by status



**SOAP**

протокол

# SOAP

Rest

Soap

Архитектурный стиль

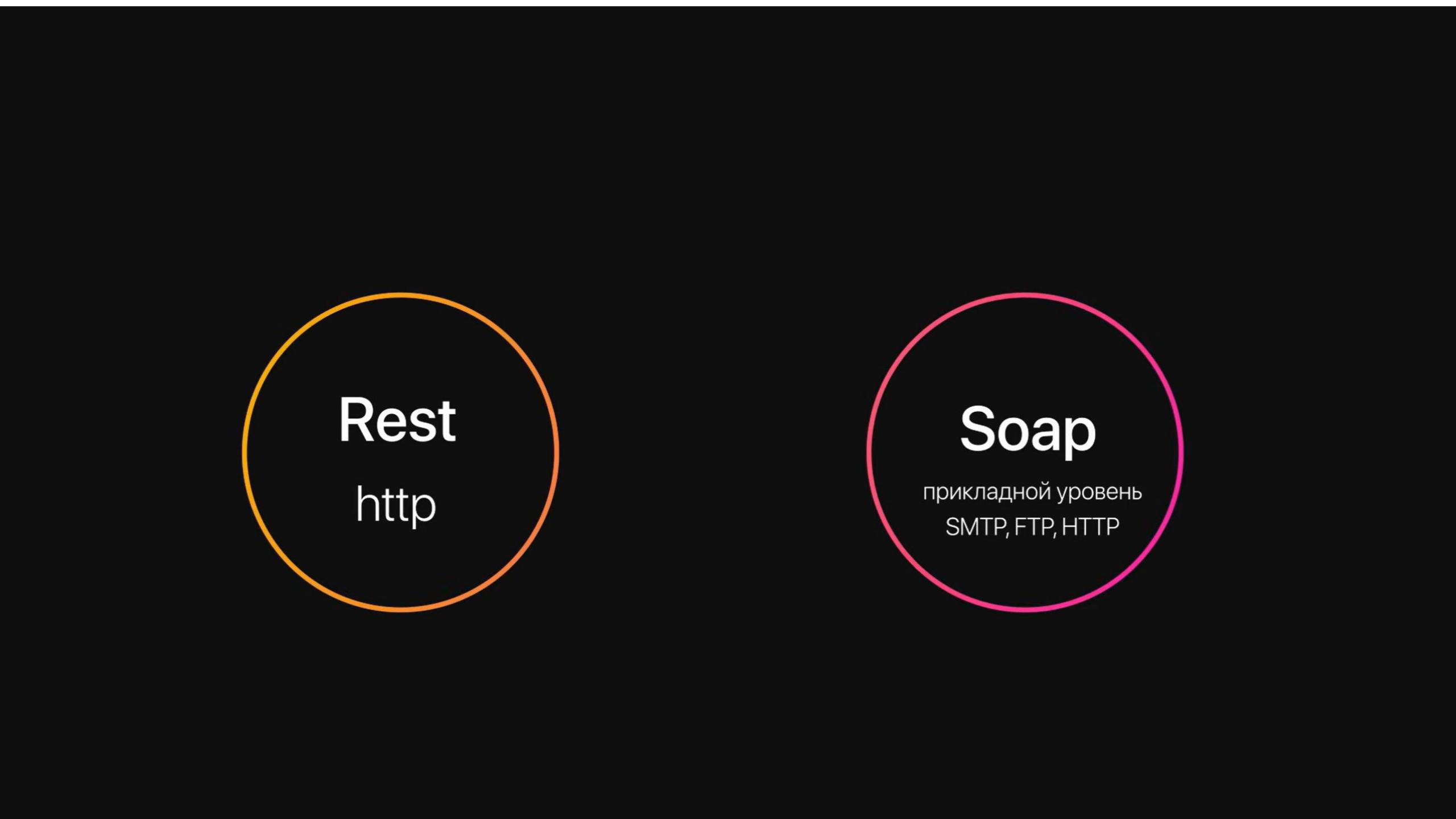
протокол обмена структурированными  
сообщениями

JSON/XML

XML (Soap-XML)

# SOAP





**Rest**

http

**Soap**

прикладной уровень  
SMTP, FTP, HTTP

# Soap

использует WSDL (Web Services Description Language) язык  
описания веб-сервисов и доступа к ним,  
основанный на языке XML

```
<binding type="bookPortType" name="bookBind">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="getBook">
        <soap:operation soapAction="getBook"/>
        <input> <soap:body use="literal"/> </input>
        <output> <soap:body use="literal"/> </output>
    </operation>
</binding>

<service name="Hello Service">
    <port binding="bookBind" name="bookPort">
        <soap:address location="http://localhost/bookservice"/>
    </port>
</service>
```

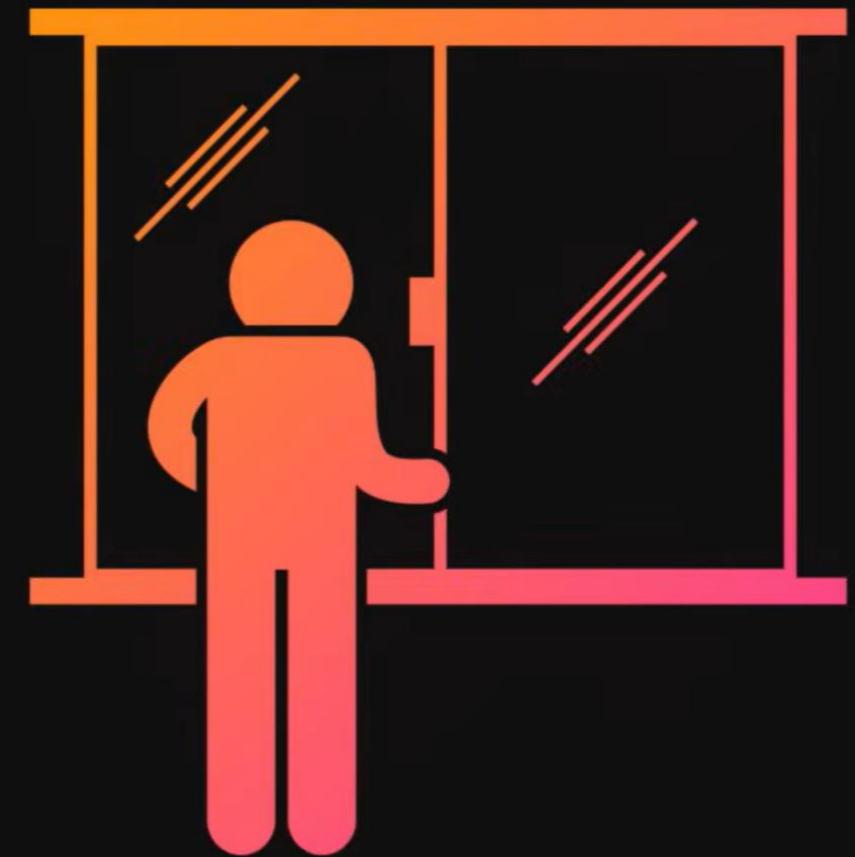
# Rest

endpoint endpoint endpoint endpoint



endpoint endpoint endpoint endpoint

# Soap



Название процедуры

# Сообщение SOAP

Envelope

Header

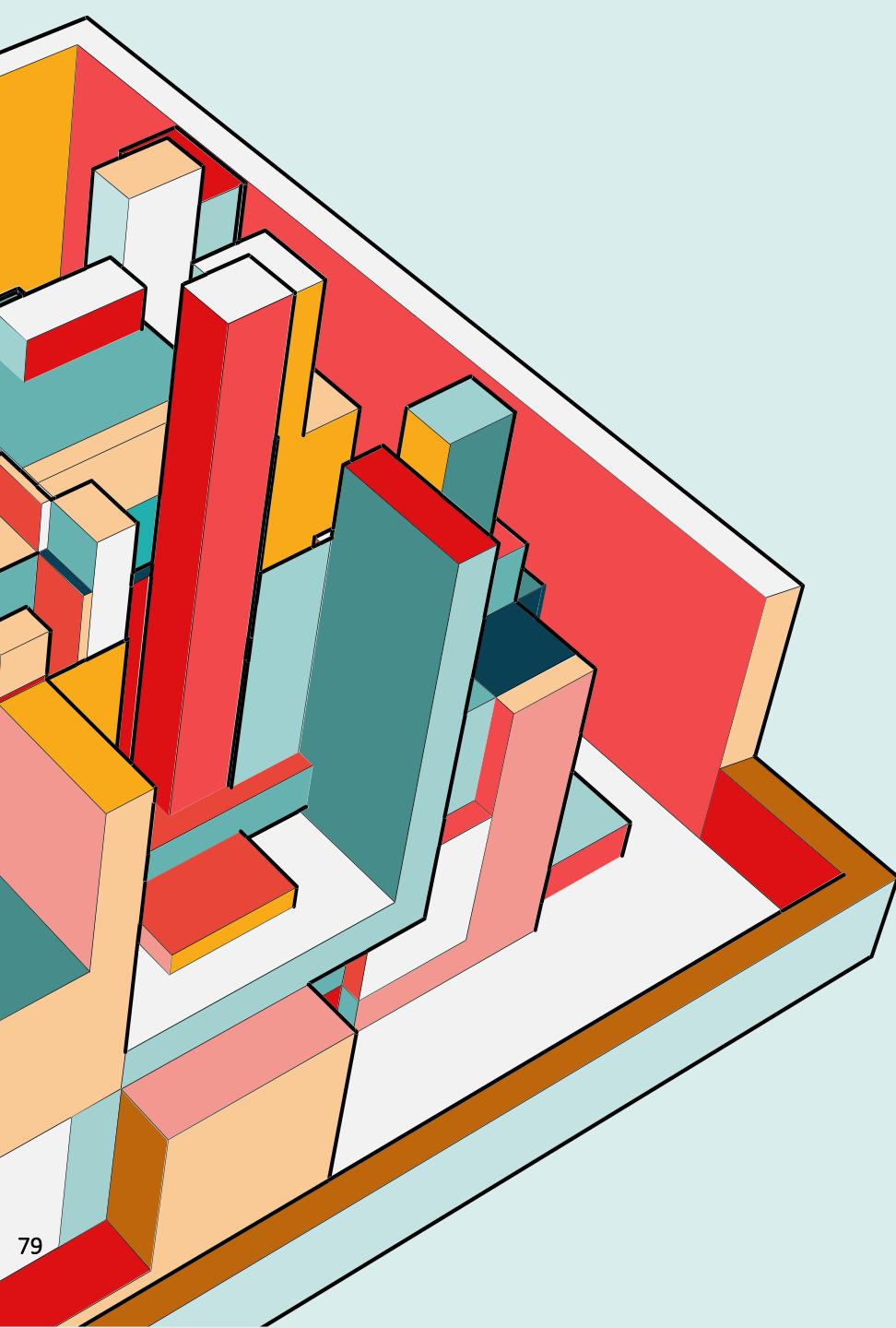
Body

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"  
xmins:demo="http://fto.com.ru/demo">  
    <soap:Header/>  
    <soap:Body>  
        <demo:СлучайноеЧисло>  
            <demo:Минимум>10</demo:Минимум>  
            <demo:Максимум>30</demo:Максимум>  
        </demo:СлучайноеЧисло>  
    </soap:Body>  
</soap:Envelope>
```

**Rest - набор рекомендаций (свободная структура)**

---

**SOAP - задает строгую структуру**



# **GRAPHQL**

Язык запросов

КупиДевайс

Админ Выйти



Iphone 12 pro max

4.5

от 100 000 руб

Добавить в корзину

Характеристики:

Оперативная память: 5гб  
Камера: 12мп  
Процессор: пентиум 5  
Объем памяти: 256гб  
Оперативная память: 5гб  
Камера: 12мп  
Процессор: пентиум 5  
Объем памяти: 256гб  
Оперативная память: 5гб  
Камера: 12мп  
Процессор: пентиум 5  
Объем памяти: 256гб

Запрашивается много  
данных

КупиДевайс

Админ Выйти

Samsung

Холодильники	Смартфон Sam.. 5 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51
Телефоны	Смартфон Sam.. 5 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51
Стиральные машины	Смартфон Sam.. 5 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51
Ноутбуки	Смартфон Sam.. 5 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51
Планшеты	Смартфон Sam.. 5 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51
Компьютеры	Смартфон Sam.. 5 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51	Смартфон Sam.. 4.2 ⭐ Galaxy a51

1 2 3 ... 7

Запрашивается мало  
данных

## **Классический подход**

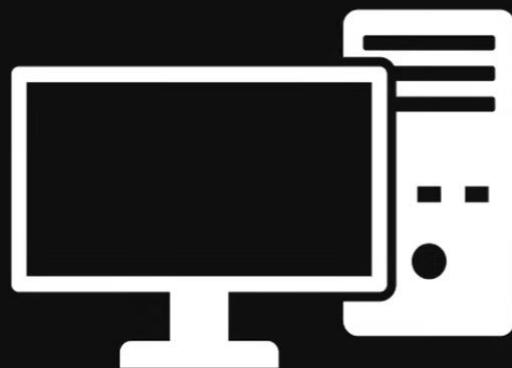
/products/little\_data  
/products/big\_data

## **Graphql**

query /products

Дай мне эти данные

```
{  
  id  
  name  
  price  
}
```



Клиент

query / products



Сервер

```
{  
  id  
  name  
  price  
}
```



## Классический подход

Сервер определяет схему  
и формат возвращаемых данных



## GraphQL

Сервер определяет схему,  
клиент запрашивает нужные  
ему данные

QUERY  
[GET]



MUTATION  
[POST]

SUBSCRIPTION  
[real time]



```
type Query {  
    hero: Character  
}  
  
type Character {  
    name: String  
    friends: [Character]  
    homeWorld: Planet  
    species: Species  
}  
  
type Planet {  
    name: String  
    climate: String  
}  
  
type Species {  
    name: String  
    lifespan: Int  
    origin: Planet  
}
```

Описывается схема данных



# QUERY

```
{  
  hero {  
    name  
  }  
}
```

```
{  
  "data": {  
    "hero" {  
      "name": "RD-D2"  
    }  
  }  
}
```

# Преимущества

## Частичная самодокументация

Преимущества

КОДОГЕНЕРАЦИЯ

# Преимущества

```
// apollo-remote-state/client/src/operations/queries/getAllTodos
import { gql } from "@apollo/client";

export const GET_ALL_TODOS = gql`query GetAllTodos {
  todos {
    edges {
      node {
        id
        text
        completed
      }
    }
  }
}
```

npx apollo codegen:generate --localSchemaFile=graphql-schema.json --target=typescript

# Преимущества

```
/* tslint:disable */
/* eslint-disable */
// This file was automatically generated and should not be

// =====
// GraphQL query operation: GetAllTodos
// =====

export interface GetAllTodos.todos_edges_node {
  __typename: "Todo";
  id: number;
  text: string;
  completed: boolean;
}

export interface GetAllTodos.todos_edges {
  __typename: "TodosEdge";
  node: GetAllTodos.todos_edges_node;
}

export interface GetAllTodos.todos {
  __typename: "TodosConnection";
  edges: (GetAllTodos.todos_edges | null)[];
}

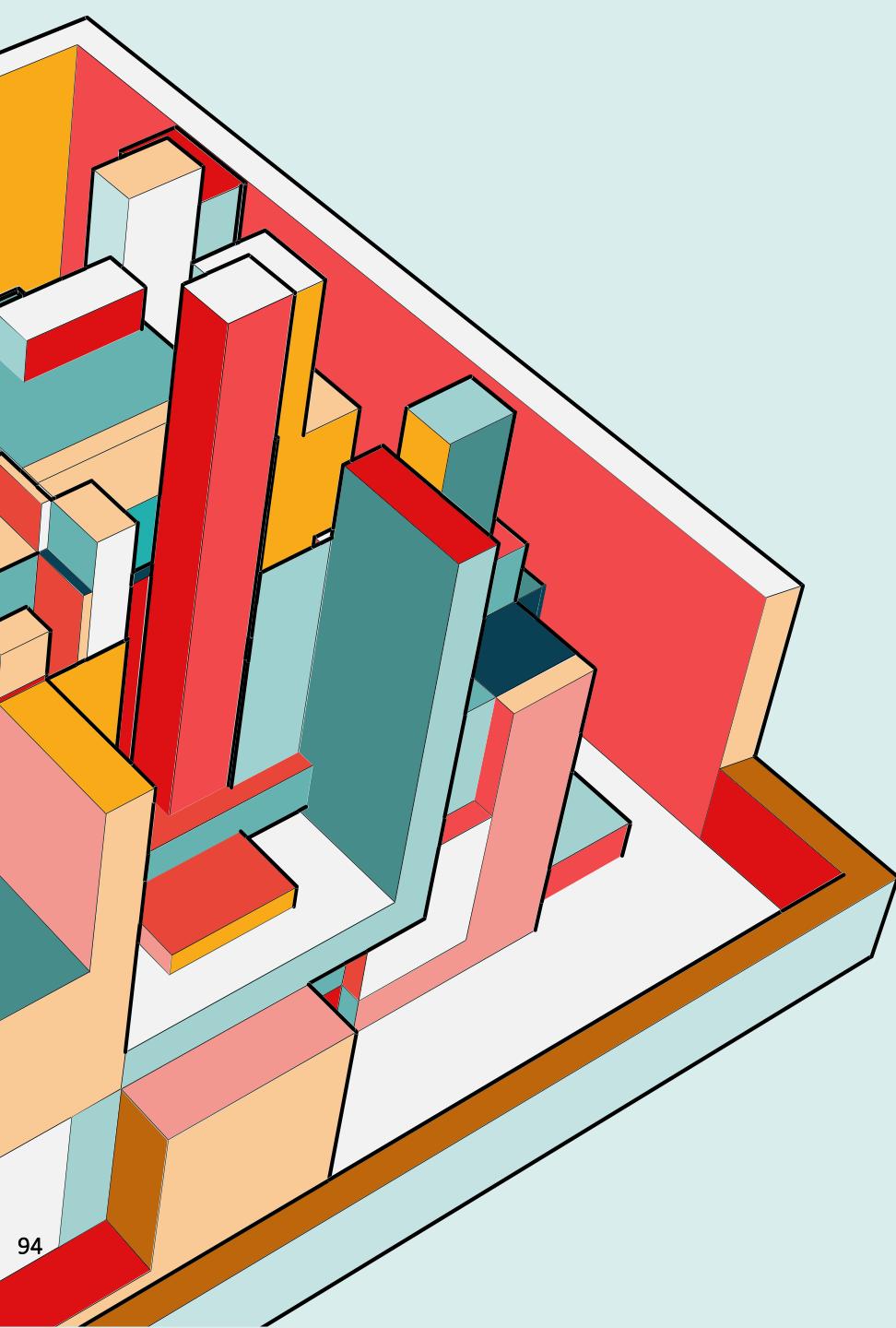
export interface GetAllTodos {
  todos: GetAllTodos.todos;
}
```

# Преимущества

КЛИЕНТ ЗАПРАШИВАЕТ ТОЛЬКО  
НУЖНЫЕ ДАННЫЕ

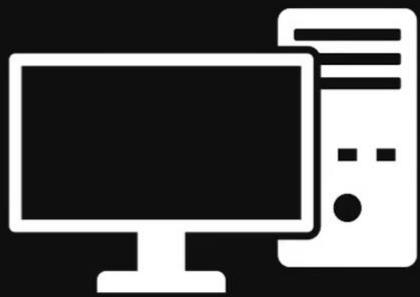
# Преимущества

МЕНЬШЕ ТРАФИКА ГОНЯЕТСЯ ПО СЕТИ



# WEBSOCKETS

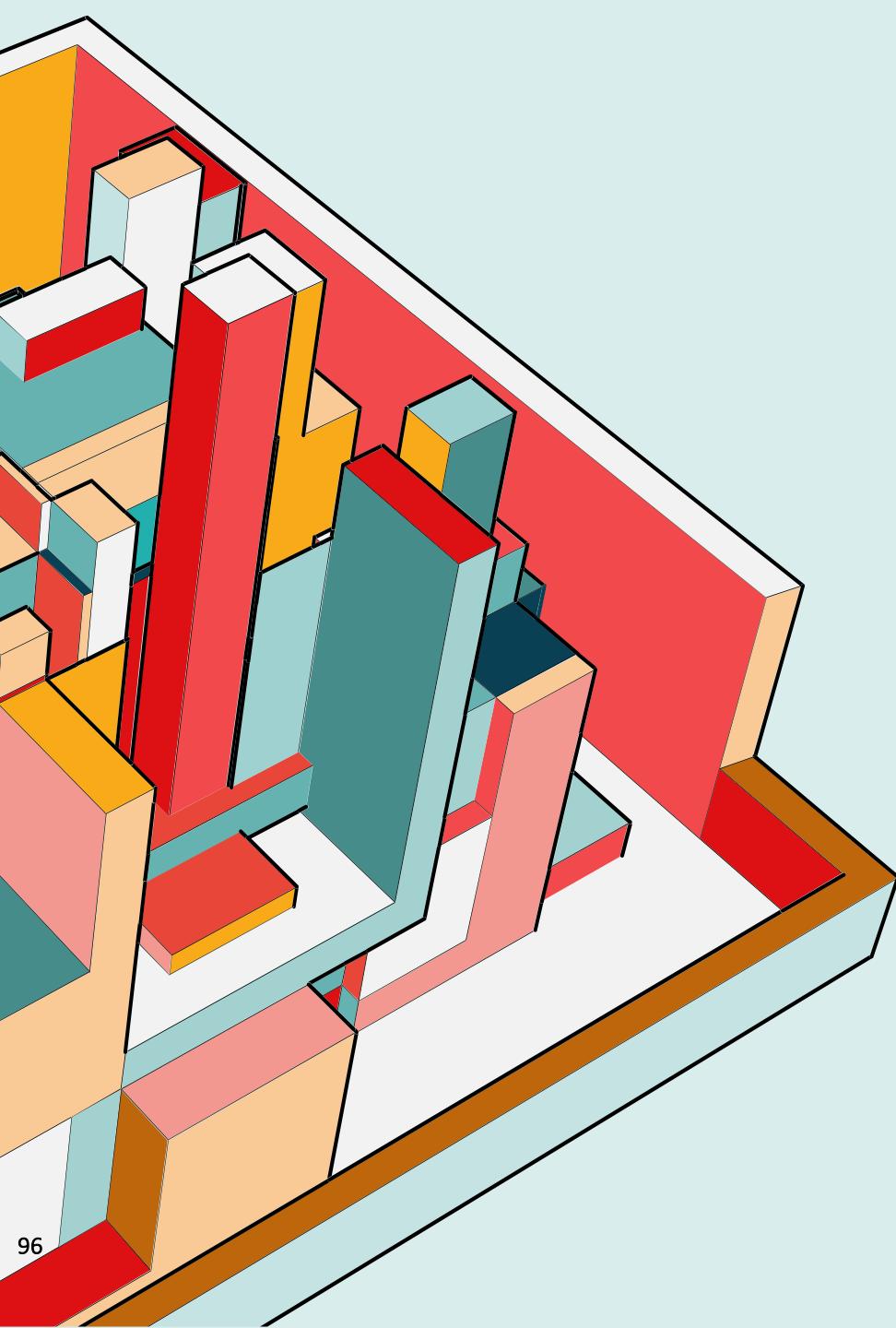
Протокол



Клиент



Сервер



**RPS**

Протокол

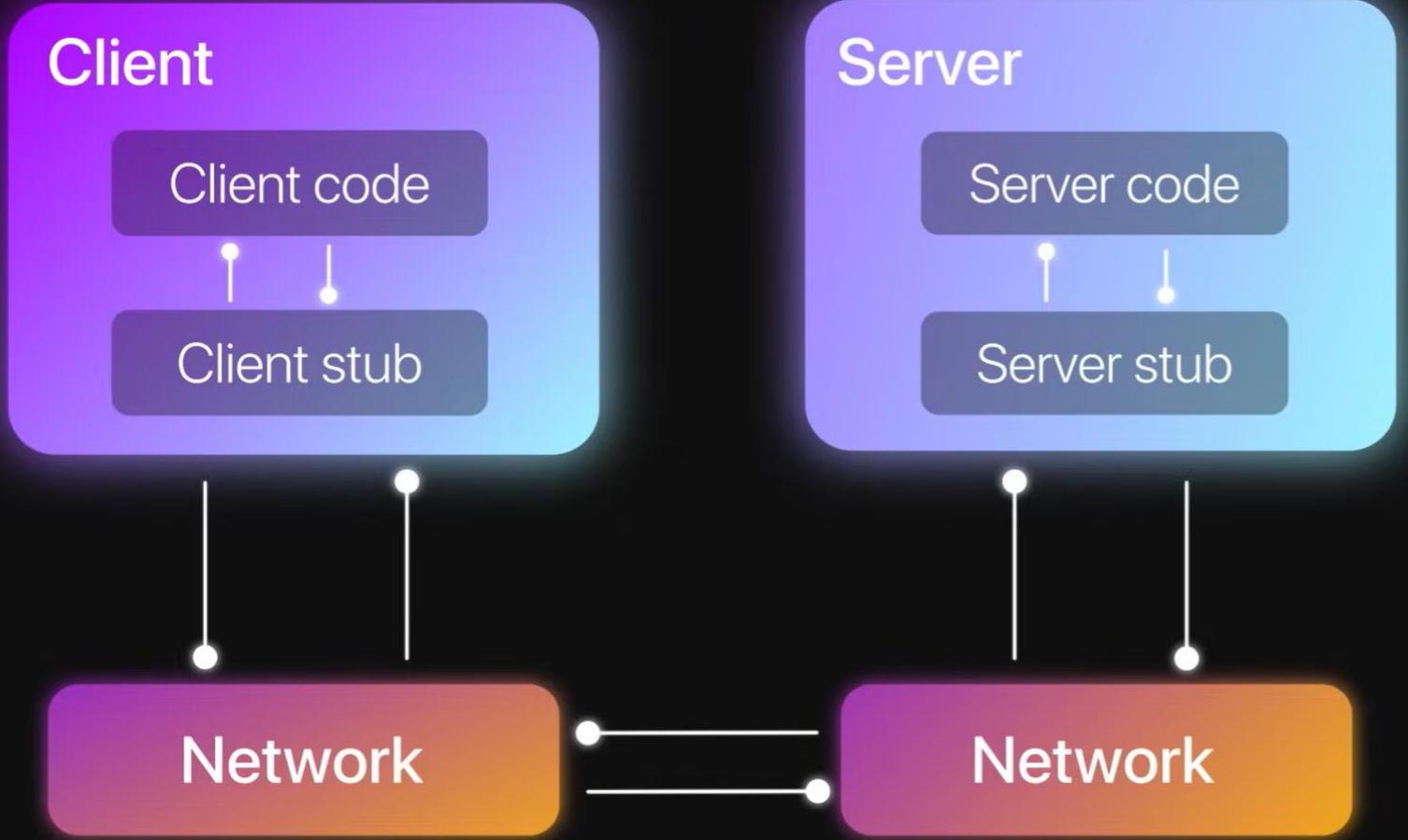


gRPC

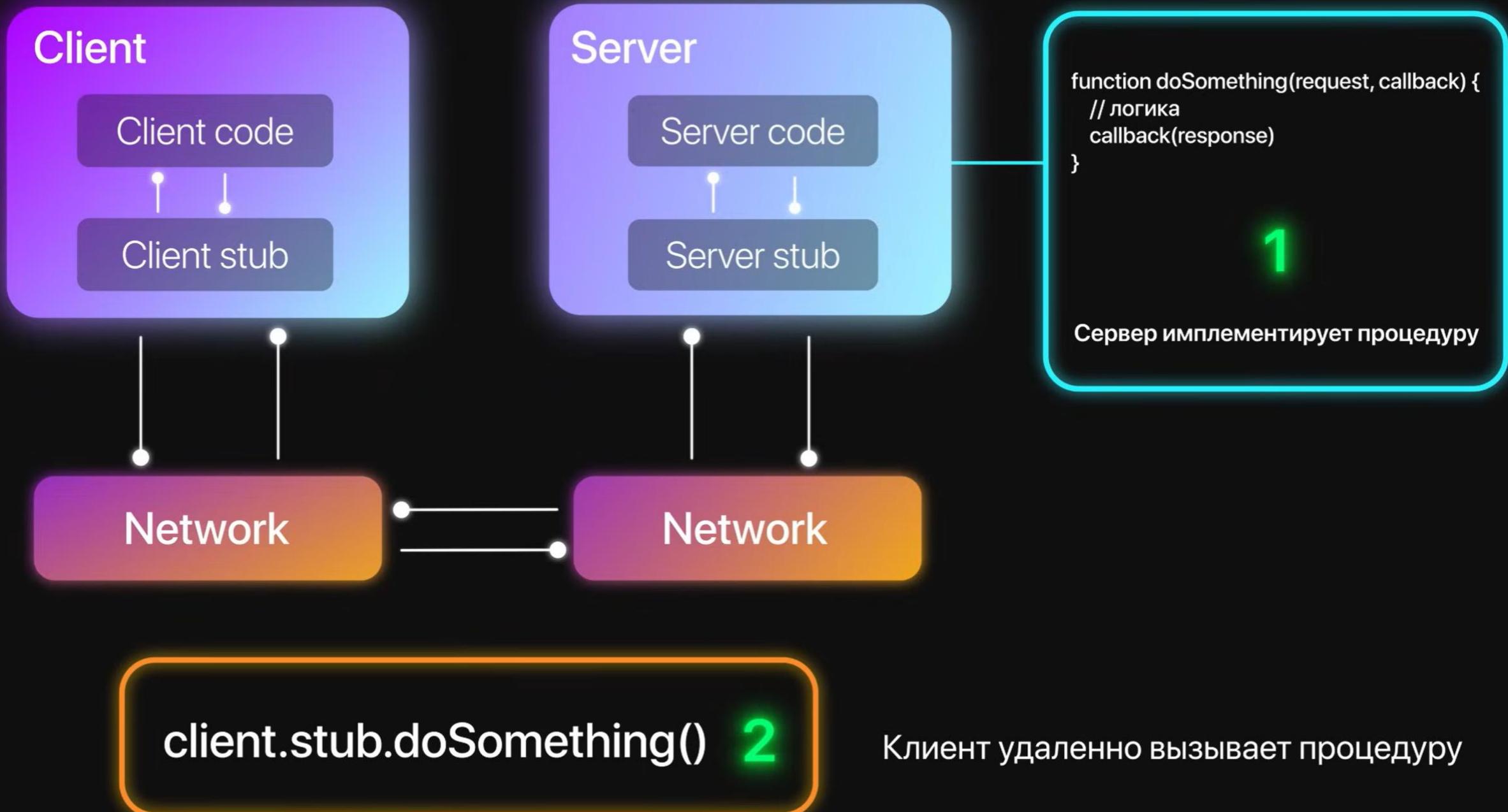
RPC

gRPC

tRPC

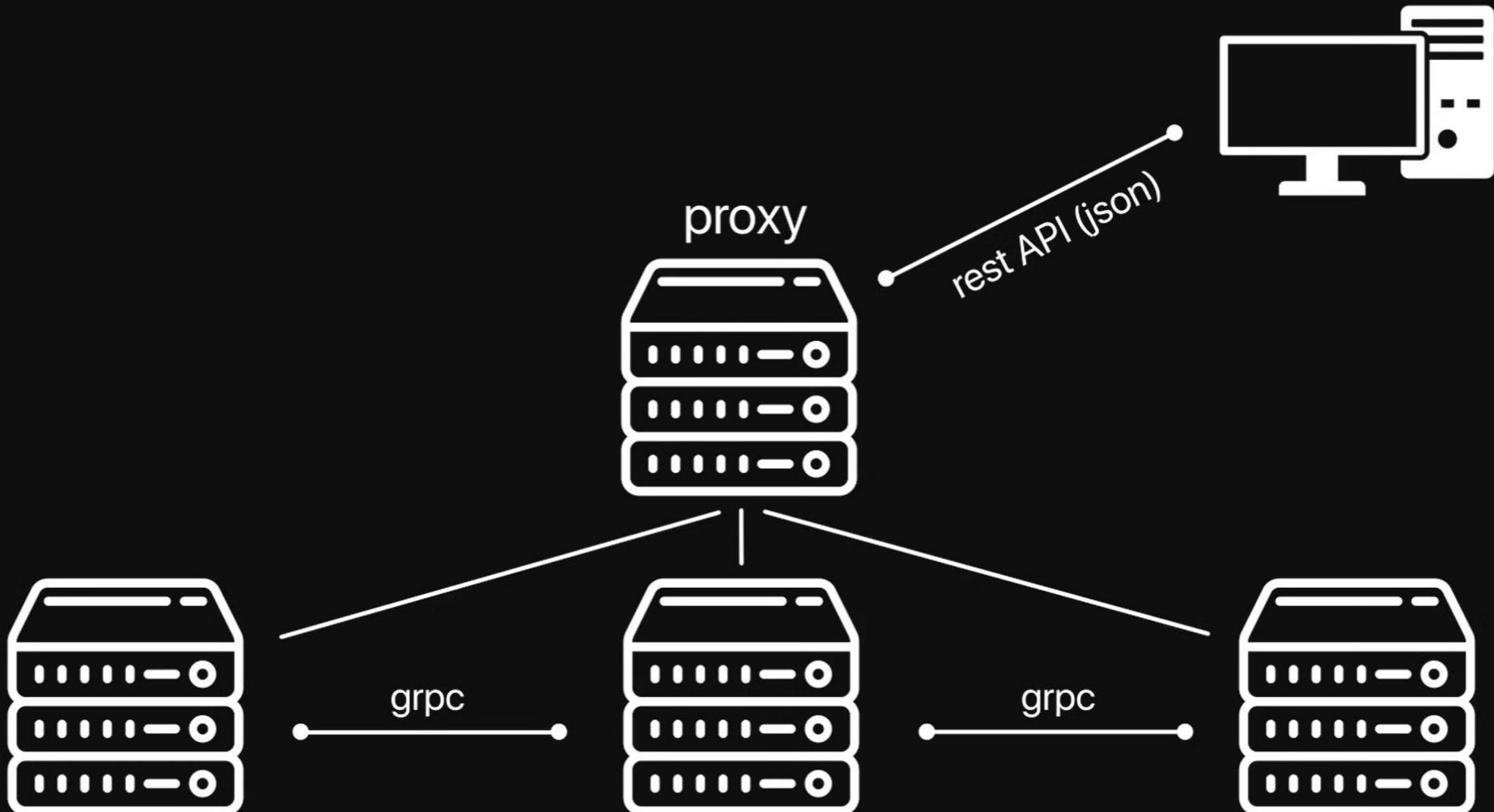


`client.stub.doSomething()`



# gRPC

Фреймворк/набор инструментов/платформа  
от Google (Исходный код открыт в 2015)



Микросервис 1

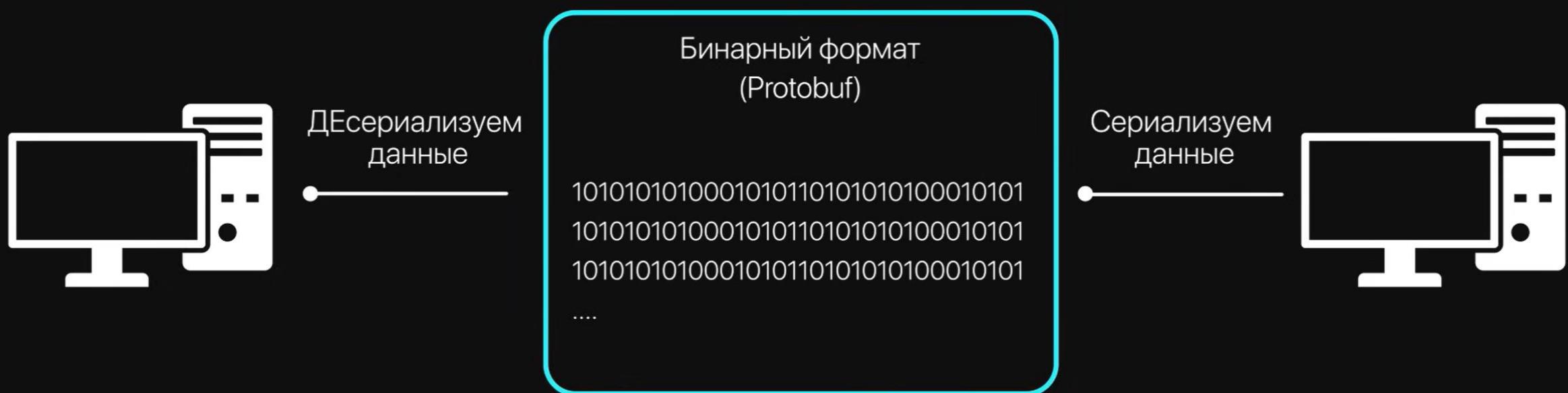
Микросервис 2

Микросервис 3

## Что отличает gRPC?

- Используется HTTP 2 вместо HTTP 1/1
- Работает в разы быстрее
- Простой и быстрый стриминг данных
- Вместо json используется бинарный формат (protobuf)
- Инструментарий из коробки:
  - Генерация кода для многих ЯП
  - Аутентификация
  - Потоковая передача данных (в том числе двунаправленная)
  - ....
- Удобный вызов процедур

# Формат данных



.proto файл

```
package helloworld;

service Greeter {
    rpc SayHello (HelloRequest) returns (HelloReply) {}

    rpc SayHelloStreamReply (HelloRequest) returns (stream HelloReply) {}
}

message HelloRequest {
    string name = 1;
}

message HelloReply {
    string message = 1;
}
```

protoc



# Используемая литература:

## **Синхронное и асинхронное взаимодействие**

- Для введения: <https://vc.ru/id1788045/709274-sinhronnoe-vs-asinhronnoe-vybiraem-podhod-k-vzaimodeistviyu-mikroservisov>
- Для погружения: <https://habr.com/ru/companies/oleg-bunin/articles/543946/>

## **Клиент-серверное и межсервисное взаимодействие (синхронная семантика)**

- Очень кратно <https://university.ylab.io/articles/tpost/j9l5xdbxs1-mezhservisnoe-vzaimodeistvie>
- Чуть подробнее <https://habr.com/ru/articles/729528/>

## **О каждом из вариантов подробно**

- REST API <https://blog.skillfactory.ru/glossary/rest-api/>
- gRPC [https://yandex.cloud/ru/docs/glossary/grpc?utm\\_referrer=https%3A%2F%2Fwww.google.com%2F](https://yandex.cloud/ru/docs/glossary/grpc?utm_referrer=https%3A%2F%2Fwww.google.com%2F)
- SOAP API <https://blog.skillfactory.ru/glossary/soap-api/>
- GraphQL <https://blog.skillfactory.ru/kak-novichku-nachat-polzovatsya-graphql-i-zachem-eto-nuzhno/>
- WebSocket <https://blog.skillfactory.ru/glossary/websocket/>
- Webhooks <https://www.mango-office.ru/journal/for-marketing/osnovy/webhook-i-kak-ego-ispolzovat/>
- REST API vs. GraphQL <https://academy.mediasoft.team/article/rest-api-vs-graphql-vybiraem-arkhitekturu-obsheniya-fronta-i-beka-dlya-prilozheniya/>

# СПАСИБО!

Виденин Сергей

@videninserg

