

gRPC Introduction

gRPC vs. REST API, Protobuf

Неделя 14

Вспоминаем разницу между HTTP 1.1 и 2

Характеристики HTTP/1.1:

- Текстовый формат
- Заголовки в текстовом формате
- TCP-соединение требует «трехстороннего рукопожатия» (three-way handshake) – один запрос и ответ с одним единственным TCP-соединением

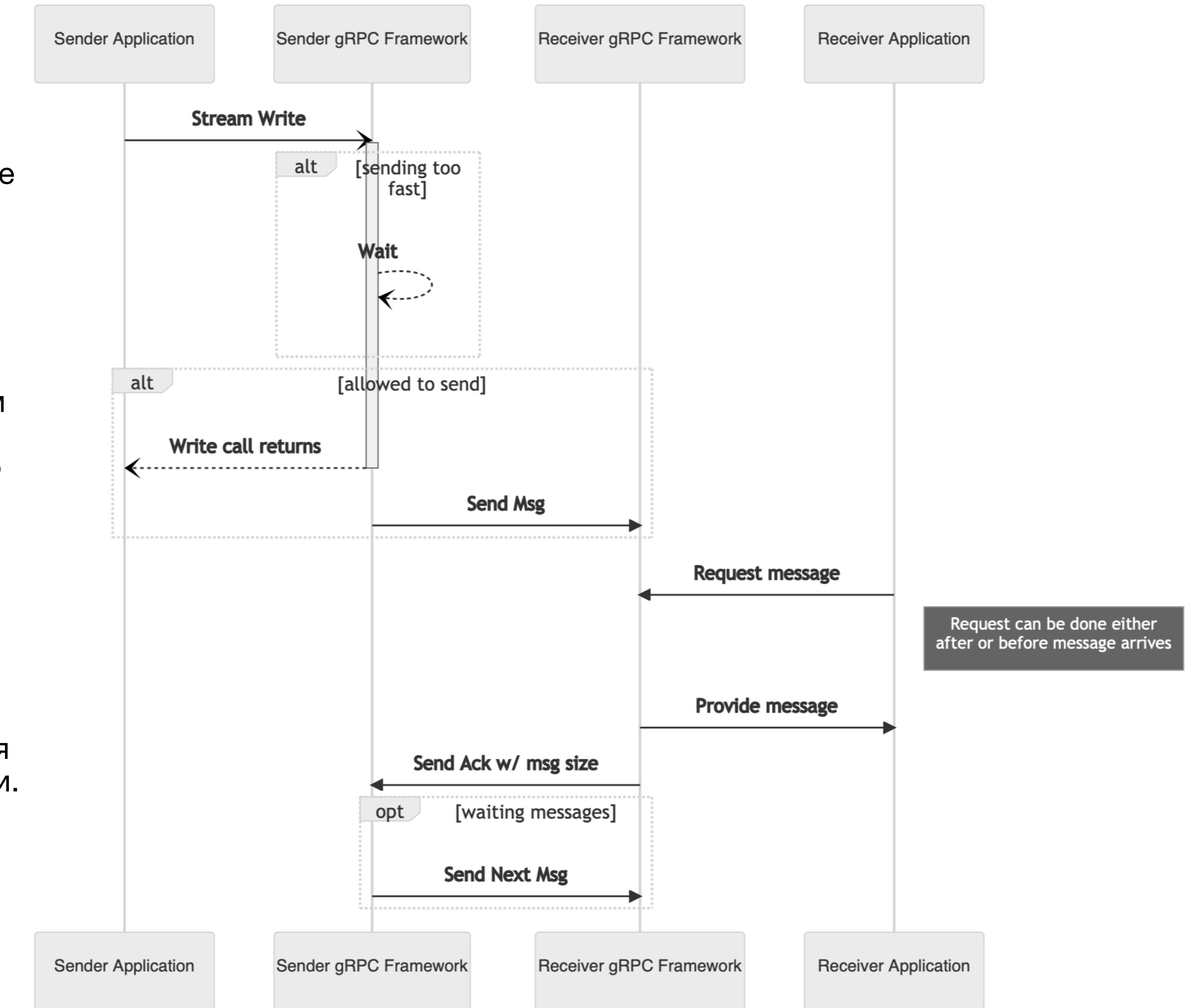
Характеристики HTTP/2:

- Бинарный формат
- Сжатие заголовков
- Управление потоком
- Мультиплексирование (одно и то же TCP-соединение может быть повторно использовано для мультиплексирования. Потоковая передача с сервера – потоковая передача от клиента – возможна двунаправленная потоковая передача)

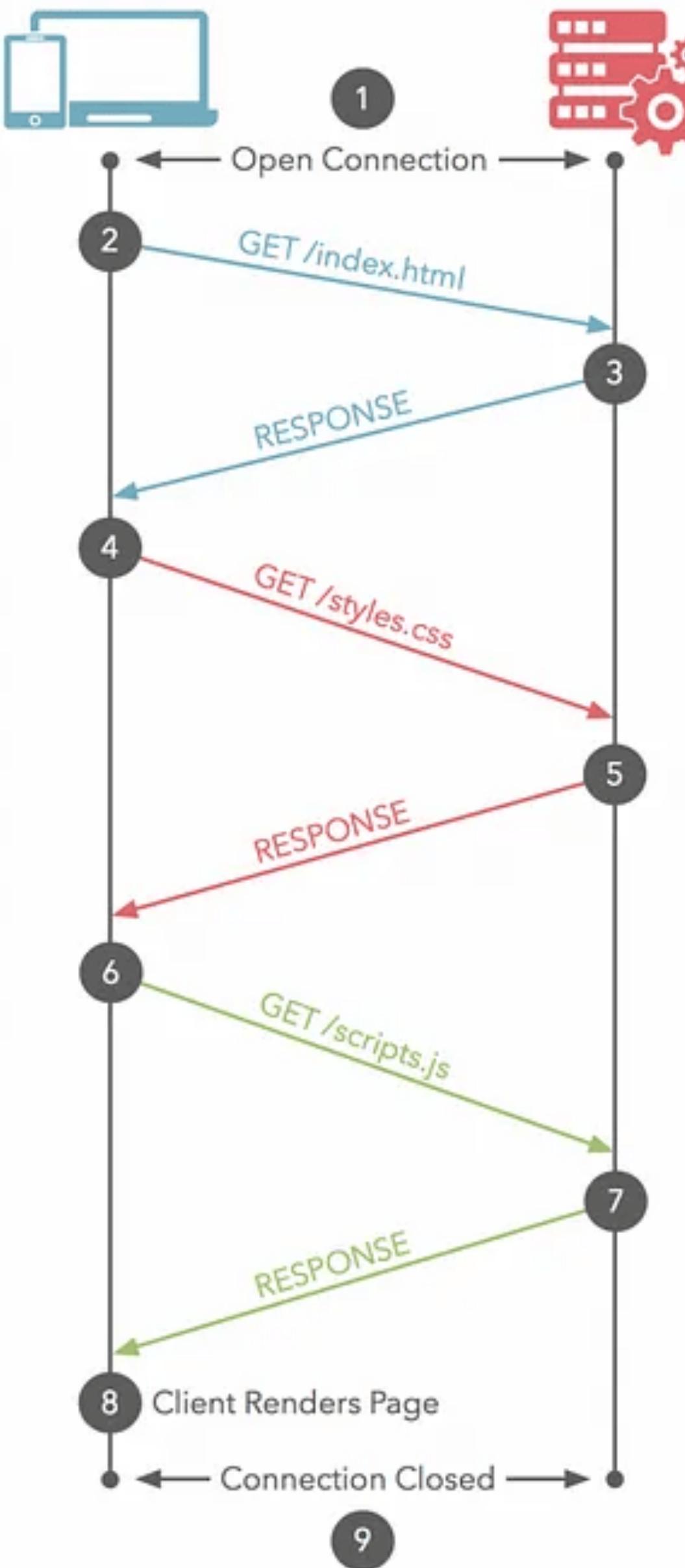
Источник: <https://habr.com/ru/companies/otus/articles/730740/>

Flow Control

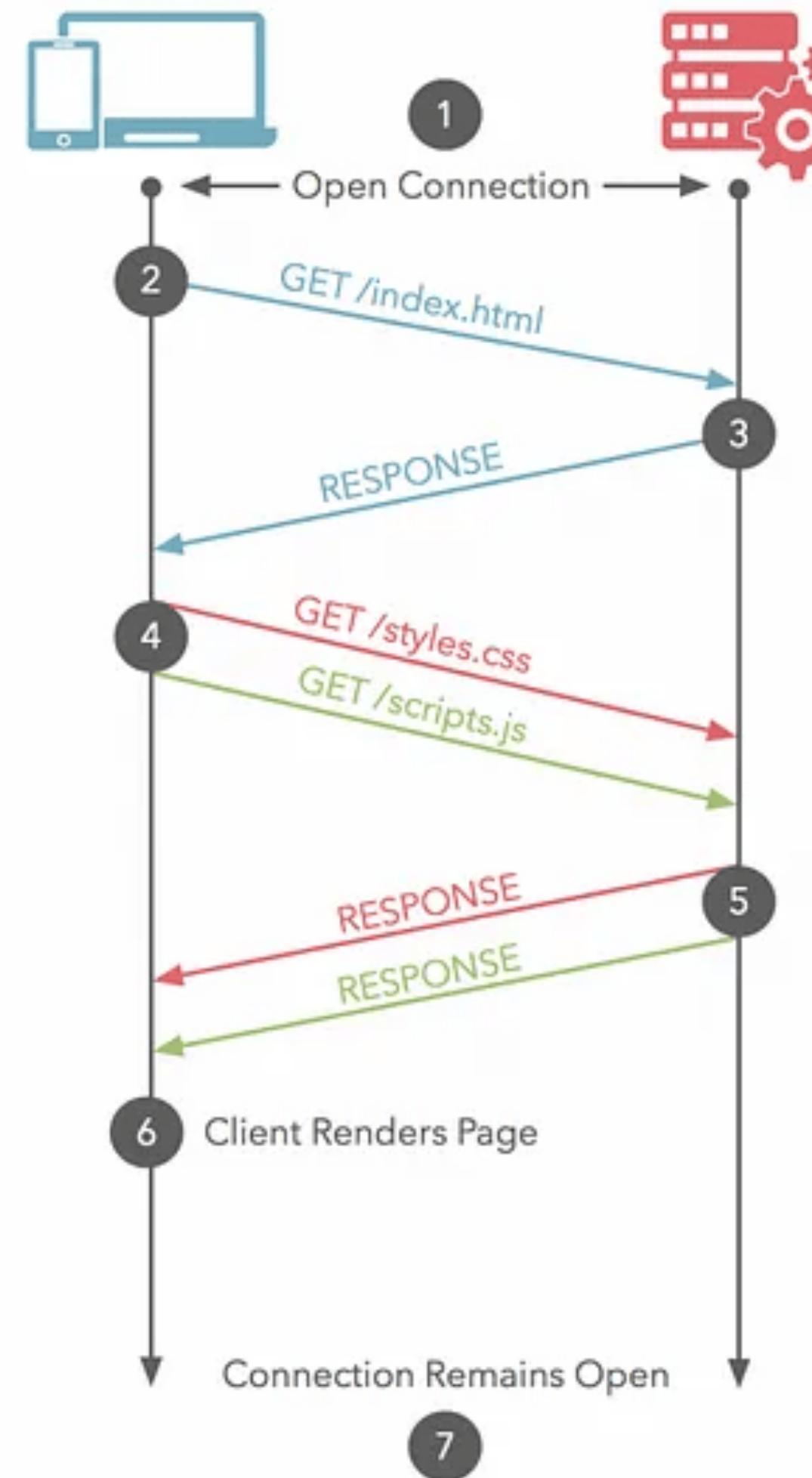
Управление потоком - это механизм, гарантирующий, что получатель сообщений не будет перегружен быстрым отправителем. Управление потоком предотвращает потерю данных, улучшает производительность и повышает надежность. Он применяется к потоковым RPC и не имеет отношения к унарным RPC. По умолчанию gRPC обрабатывает взаимодействие с управлением потоком за вас, хотя некоторые языки позволяют вам переопределить поведение по умолчанию и взять управление на себя.



HTTP/1.1 Baseline



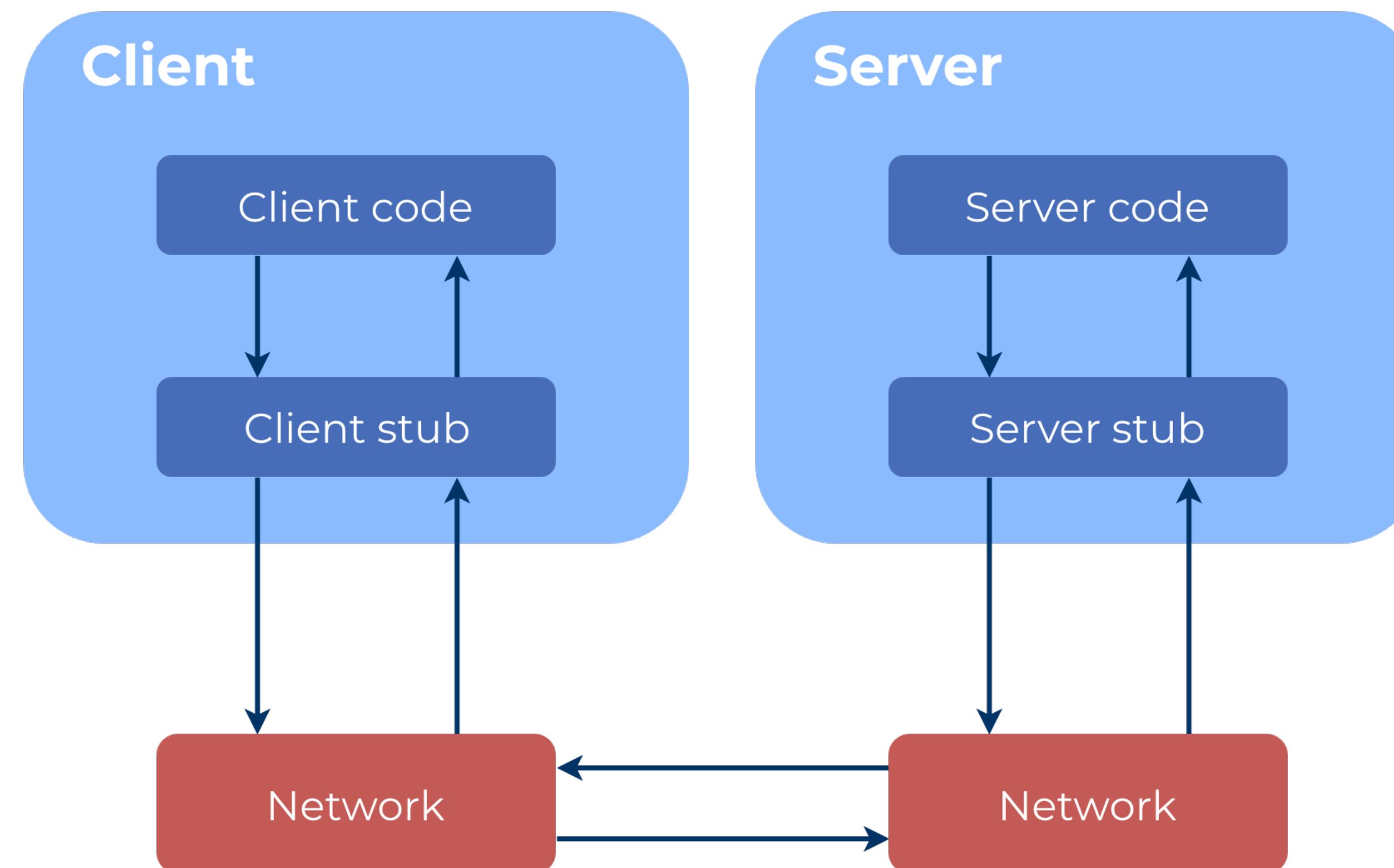
HTTP/2 Multiplexing



RPC

RPC (Remote Procedure Call) – удаленный вызов процедур.

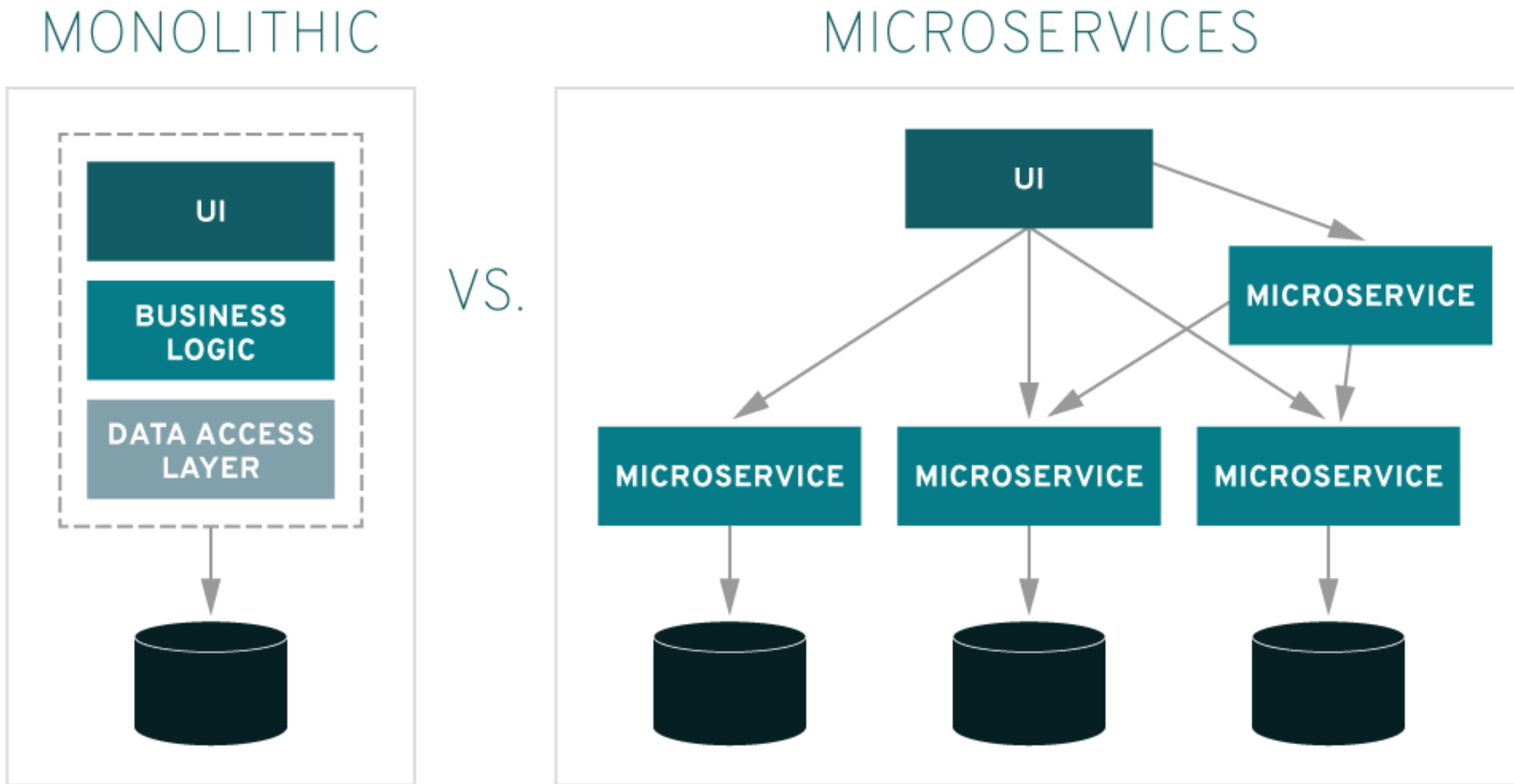
Клиент может вызывать методы сервера, при этом не воспринимая это как вызов по сети.



gRPC

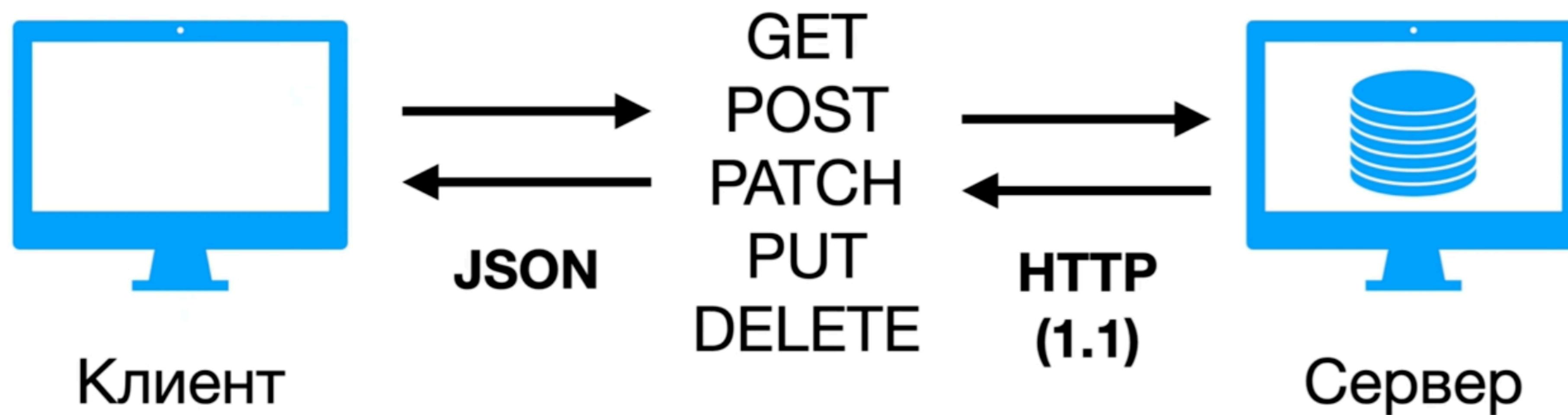
gRPC – реализация подхода RPC, разработанная компанией Google

Служит заменой REST, особенно в микросервисных системах



REST API

Стандартный способ общения приложений



Недостатки REST API

- Для каждого языка программирования необходимо писать свой код для API
- JSON, который используется в REST API для передачи данных - не бинарный формат данных, никак не сжимается при пересылке
- Чаще всего используется протокол HTTP 1.1

Преимущества gRPC

- Вместо JSON-а для передачи данных используется бинарный формат Protobuf (меньше размер сообщения, быстрее передача данных)
- HTTP 2 вместо HTTP 1.1
- Генерация кода для самых популярных ЯП стандартными средствами

Источник: <https://www.youtube.com/watch?v=SMy4CaxizbA&t=1626s>

Protobuf

Бинарный формат данных

JSON

```
{  
  "timestamp": 1503053477,  
  "url": "http://example.com/"  
}
```

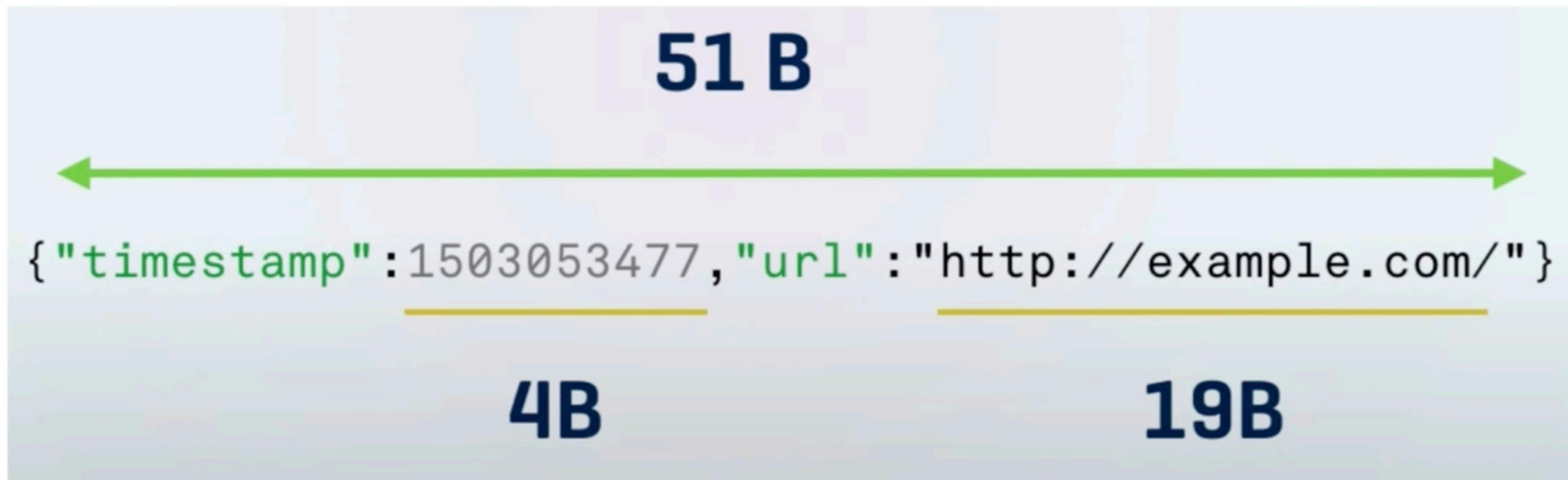
- + Читаемый и простой
- + Нет необходимости кодировать и декодировать
 - Не сжимается при передаче, передается как текст (большой размер сообщения)
 - Избыточный (ключи повторяются)
 - Нет строгой типизации

Protobuf



- + Бинарный и эффективный (использует сжатие)
- + Имеет строгую типизацию
 - Нечитаемый
 - Необходимо кодировать и декодировать данные

Избыточность JSON'a



Источник: <https://www.youtube.com/watch?v=uGYZn6xk-hA>

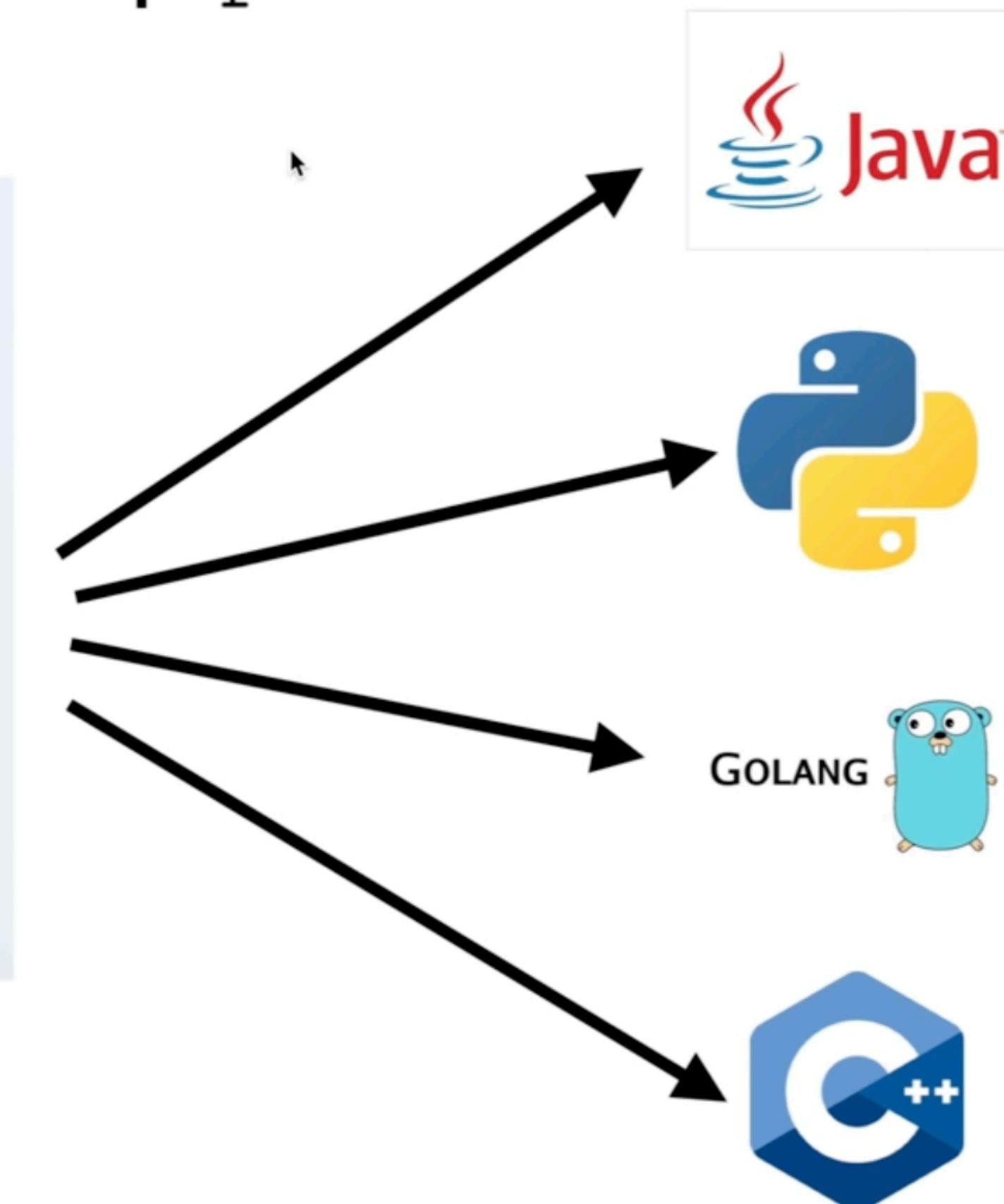
Генерация кода

Компилятор protoc

```
syntax = "proto3";  
  
package example;  
  
message WebsiteVisit {  
    int32 timestamp = 1;  
    string url = 2;  
}
```

.proto файл

Описывает типы данных, формат
сообщений, RPC операции



Когда использовать gRPC

- Микросервисы, которые общаются друг с другом
- Много разных языков программирования - gRPC (или REST API)
- Нужен стриминг данных
- Критически важна скорость передачи данных (огромное количество запросов или узкий канал)

Когда использовать REST API

- Монолитное приложение, к которому должны иметь доступ извне или через браузер

Пример proto файла

```
1 syntax = "proto3";
2 package com.example.grpc;
3
4 message HelloRequest {
5
6     string name = 1;
7
8     repeated string hobbies = 2;
9
10}
11
12 message HelloResponse {
13
14     string greeting = 1;
15
16 service GreetingService {
17     rpc greeting(HelloRequest) returns (HelloResponse);
18 }
```

Пример вызова в коде

```
public class GreetingServiceImpl extends GreetingServiceGrpc.GreetingServiceImplBase {  
  
    @Override  
    public void greeting(GreetingServiceOuterClass>HelloRequest request,  
                         StreamObserver<GreetingServiceOuterClass>HelloResponse> responseObserver) {  
  
        System.out.println(request);  
  
        GreetingServiceOuterClass>HelloResponse response = GreetingServiceOuterClass.  
            HelloResponse.newBuilder().setGreeting("Hello from server, " + request.getName())  
            .build();  
  
        responseObserver.onNext(response);  
  
        responseObserver.onCompleted();  
    }  
}
```

Дальше (в следующем модуле) поговорим о микросервисах и попробуем использовать gRPC на практике...