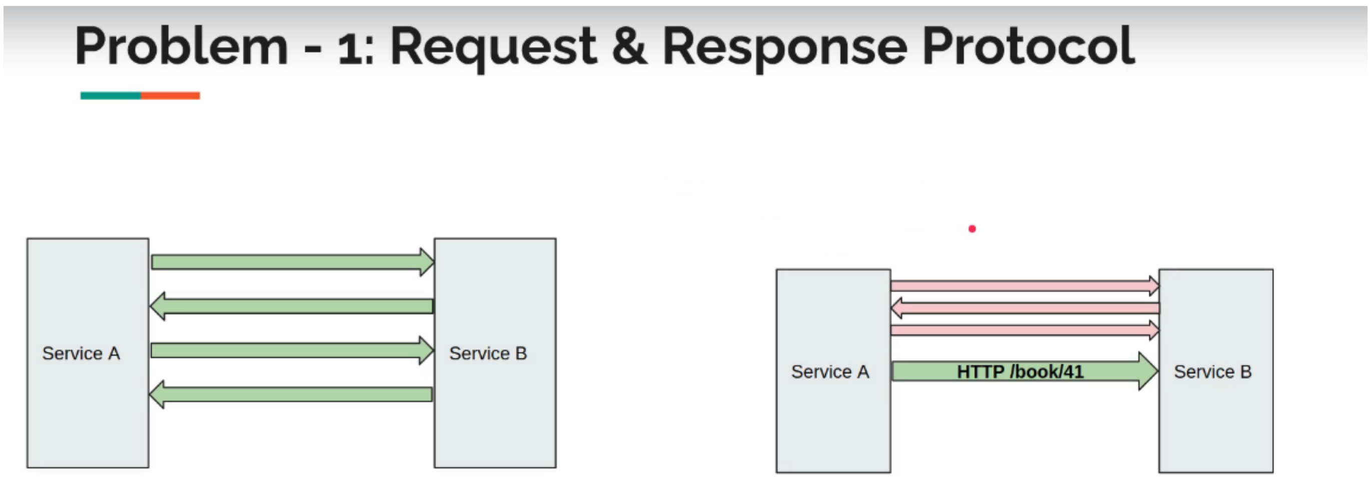
gRPC Application

1. Overview

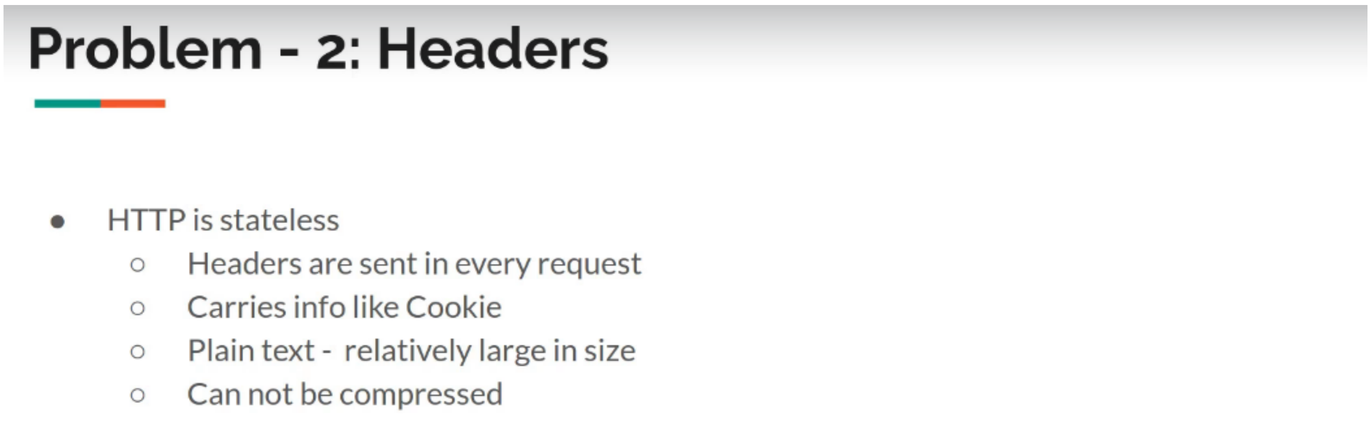
1.1 Problems with HTTP Rest

- TCP Connection create and destroy



Problem - 1: Request & Response Protocol

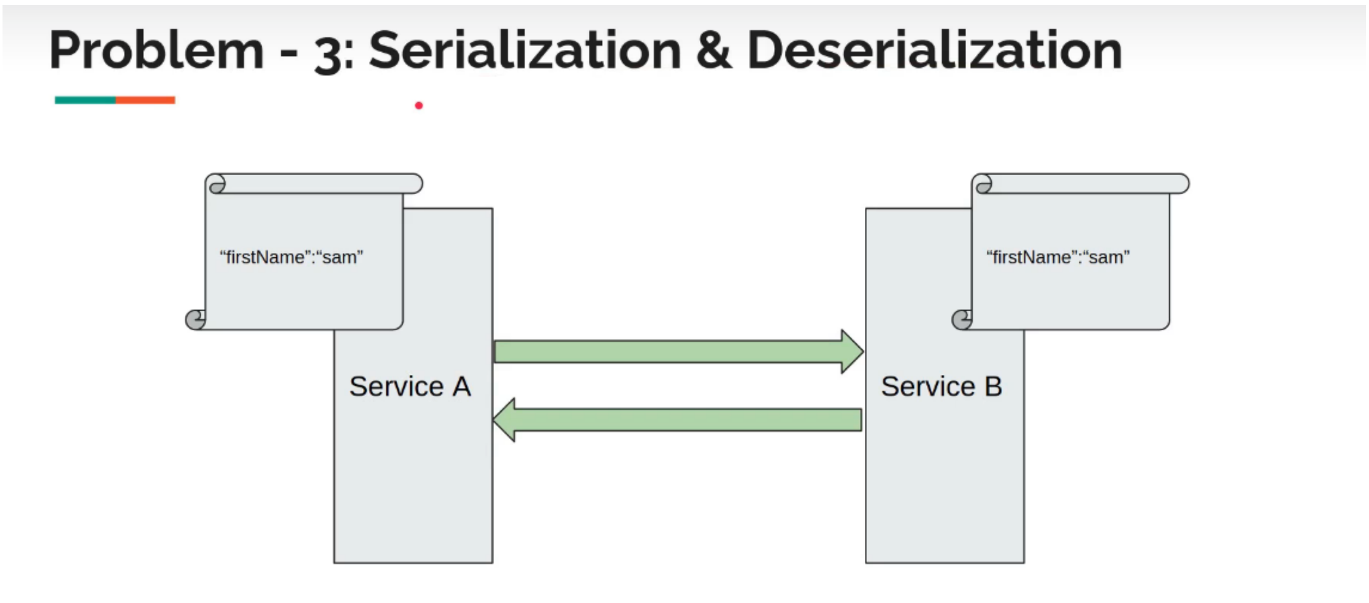
- **Headers**: heavily carried and can not decompress.



Problem - 2: Headers

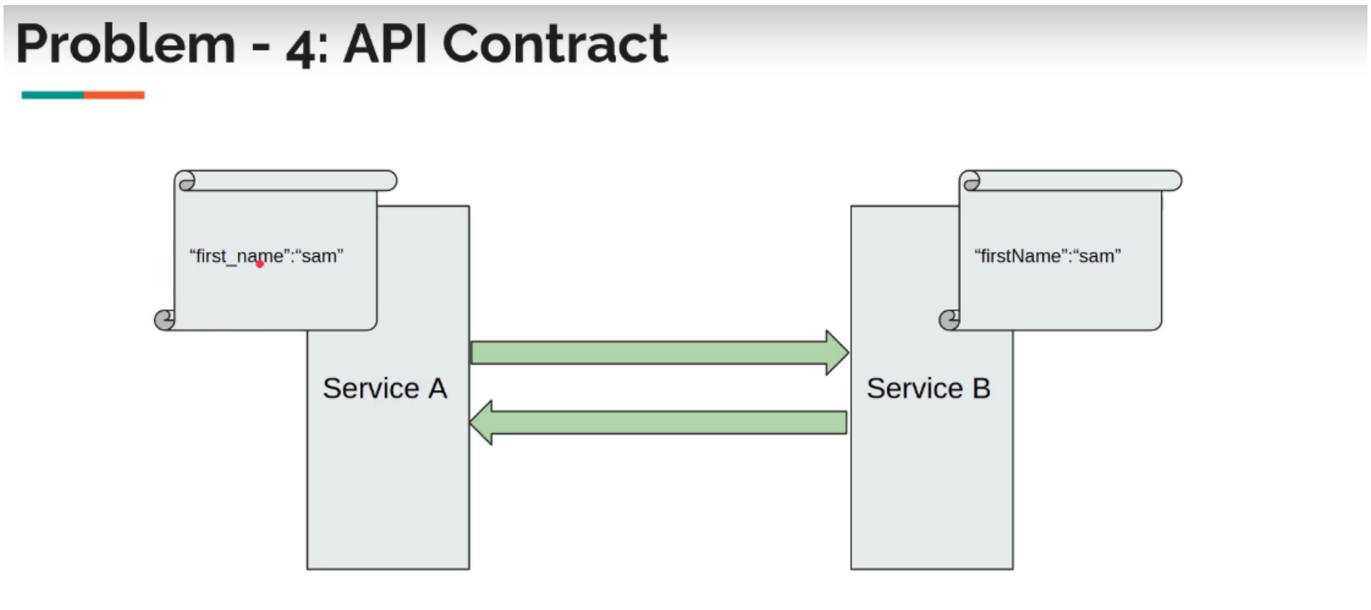
- HTTP is stateless
  - Headers are sent in every request
  - Carries info like Cookie
  - Plain text - relatively large in size
  - Can not be compressed

- Serialize & Deserialize overhead



Problem - 3: Serialization & Deserialization

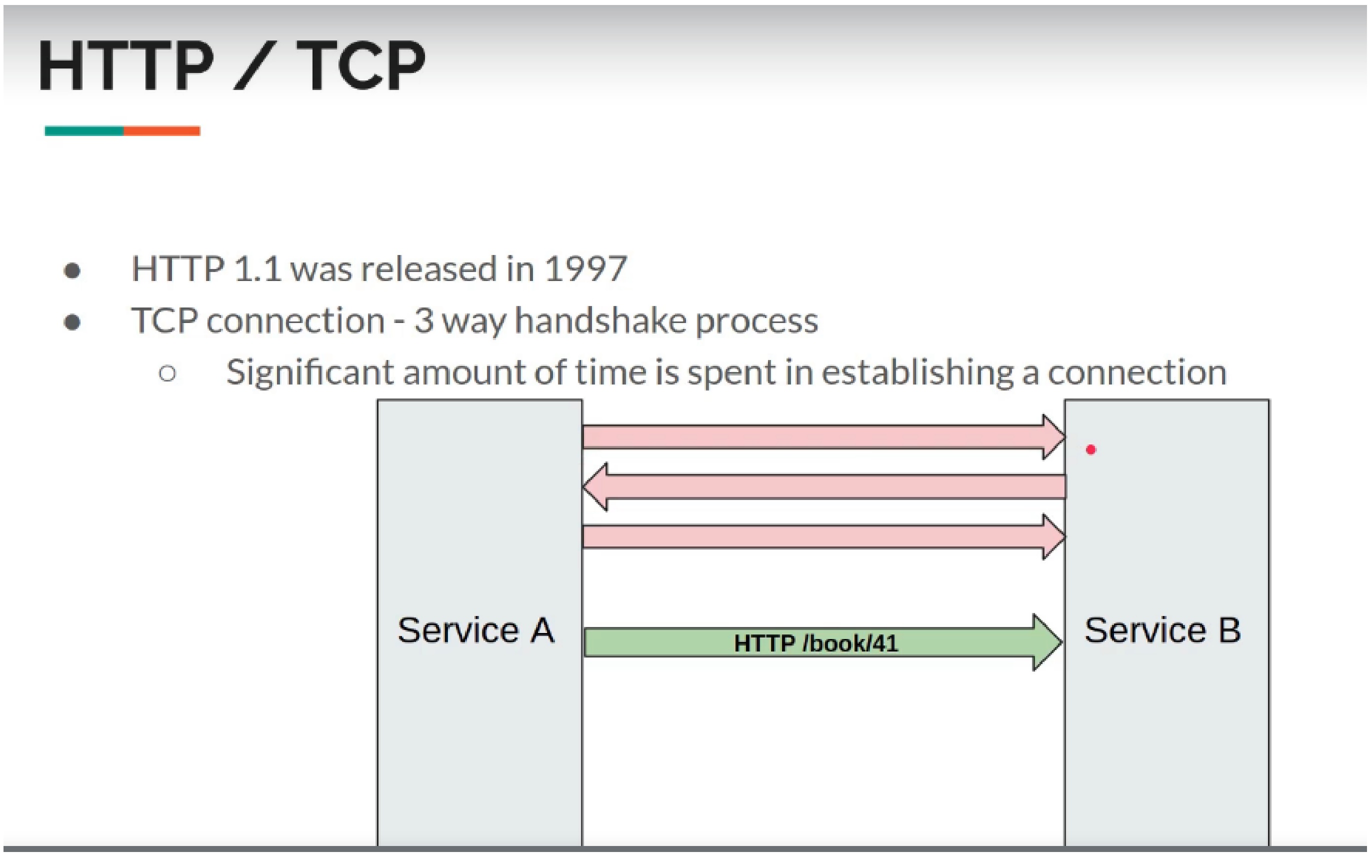
- API contract: no strict type



Problem - 4: API Contract

1.2 HTTP/2 vs HTTP/1.1

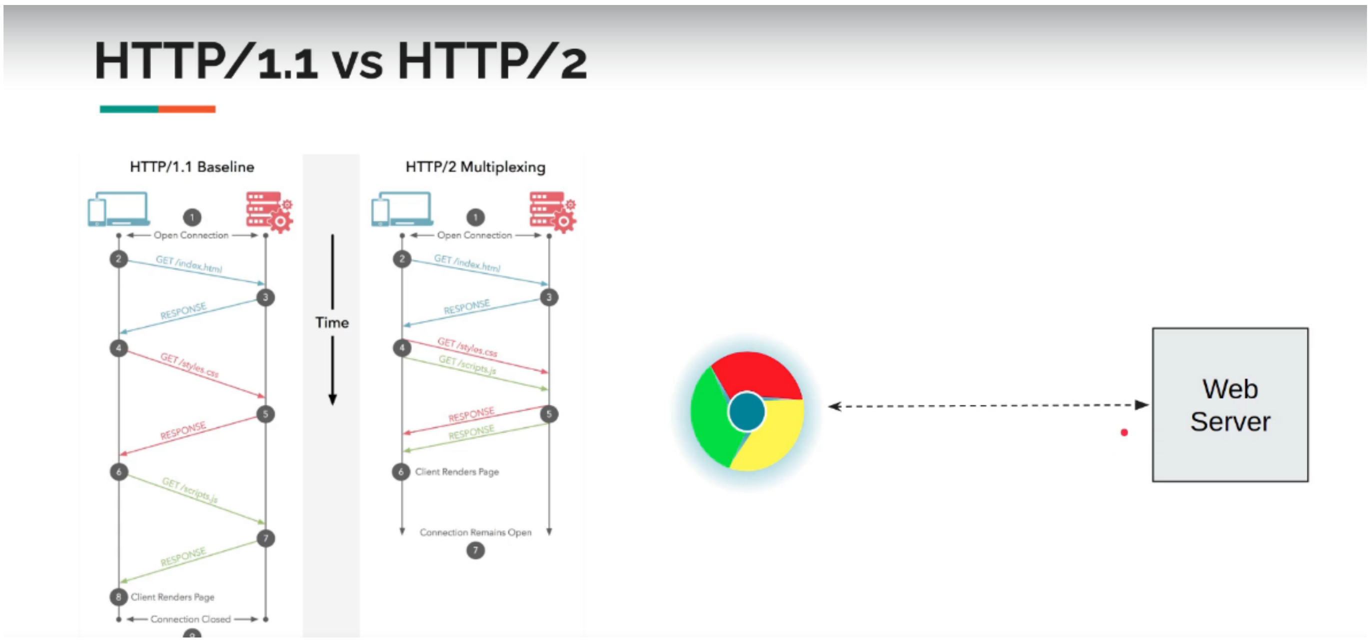
HTTP take a lot of time to establish connection



HTTP / TCP

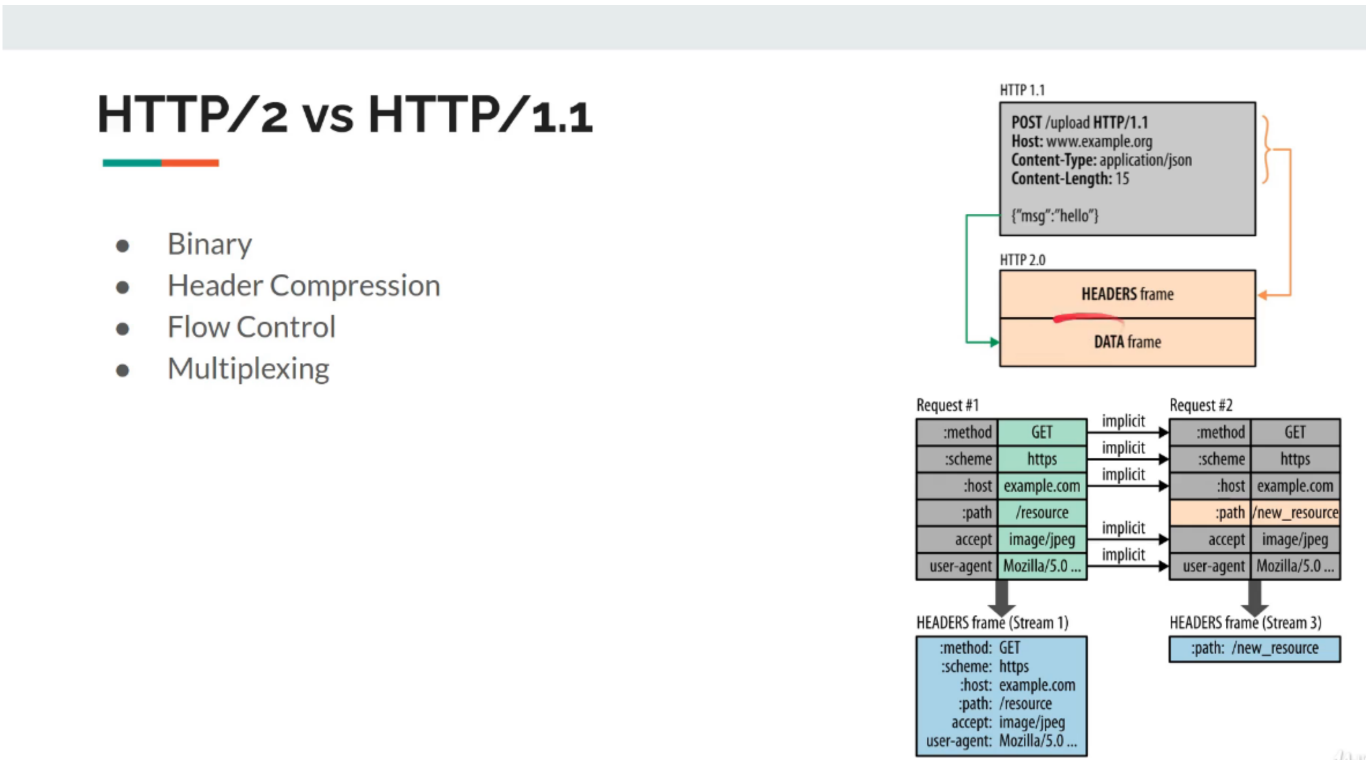
- HTTP 1.1 was released in 1997
- TCP connection - 3 way handshake process
  - Significant amount of time is spent in establishing a connection

- Multiplexing, similar to channel in rabbitMQ



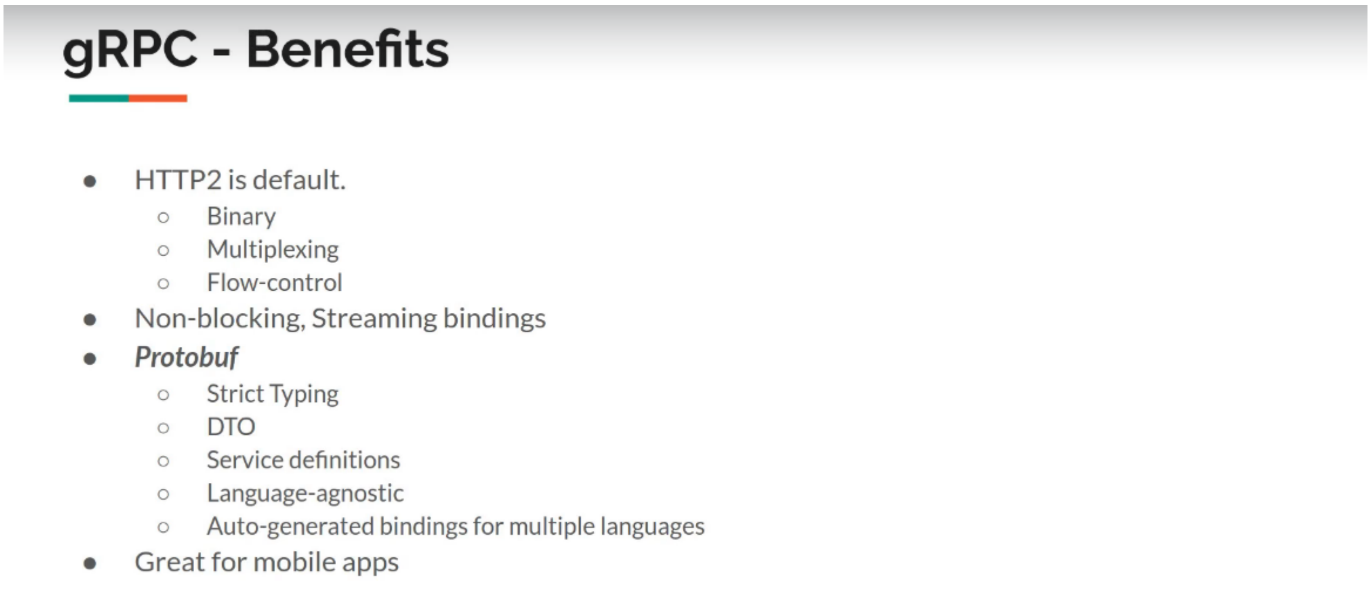
HTTP/1.1 vs HTTP/2

- HTTP/2 vs HTTP/1.1



HTTP/2 vs HTTP/1.1

- Binary
- Header Compression
- Flow Control
- Multiplexing

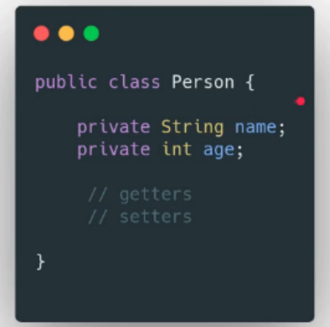


gRPC - Benefits

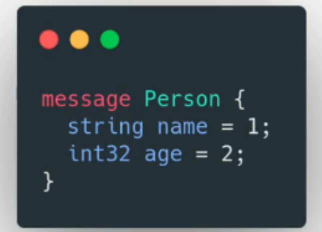
- HTTP2 is default.
  - Binary
  - Multiplexing
  - Flow-control
- Non-blocking, Streaming bindings
- *Protobuf*
  - Strict Typing
  - DTO
  - Service definitions
  - Language-agnostic
  - Auto-generated bindings for multiple languages
- Great for mobile apps

1.3 REST vs gRPC

Sample Proto



```java
public class Person {

    private String name;
    private int age;

     // getters
     // setters

}
```



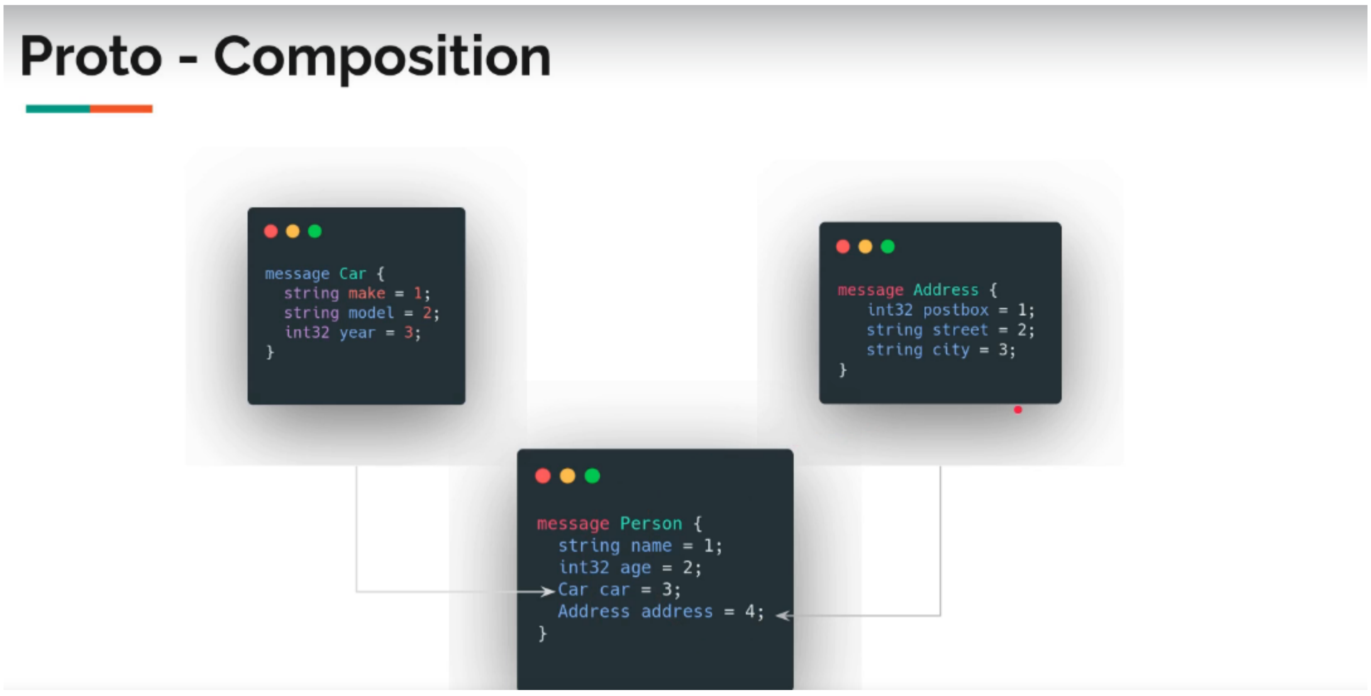
```proto
message Person {
  string name = 1;
  int32 age = 2;
}
```

```
public class PerformanceTest {

    public static void main(String[] args) {

        // JSON with Jackson
        ObjectMapper mapper = new ObjectMapper();
        JsonPerson person = new JsonPerson("A", 10);
        Runnable jsonTask = () -> {
            try {
                byte[] bytes = mapper.writeValueAsBytes(person);
                JsonPerson parsedPerson = mapper.readValue(bytes, JsonPerson.class);
            } catch (IOException jsonProcessingException) {
                jsonProcessingException.printStackTrace();
            };
        };

        // Protobuf
        Person pPerson = Person.newBuilder().setAge(10).setName("A").build();
        Runnable protoTask = () -> {
            try {
                byte[] bytes = pPerson.toByteArray();
                Person parsedPerson = Person.parseFrom(bytes);
            } catch (IOException jsonProcessingException) {
                jsonProcessingException.printStackTrace();
            };
        };

        benchMark(jsonTask, "Parse JSON");
        benchMark(protoTask, "Parse Protobuf");

    }

    static void benchMark(Runnable runnable, String method) {
        Long start = System.currentTimeMillis();
        runnable.run();
        Long end = System.currentTimeMillis();
        System.out.printf("%s take %d ms %n", method, end-start);
    }

}
```

Result:

```
Parse JSON take 88 ms
Parse Protobuf take 48 ms
```

1.4 Composition & Collection



Proto - Composition

```
message Car {
  string make = 1;
  string model = 2;
  int32 year = 3;
}
```

```
message Address {
    int32 postbox = 1;
    string street = 2;
    string city = 3;
}
```

```
message Person {
  string name = 1;
  int32 age = 2;
  Car car = 3;
  Address address = 4;
}
```

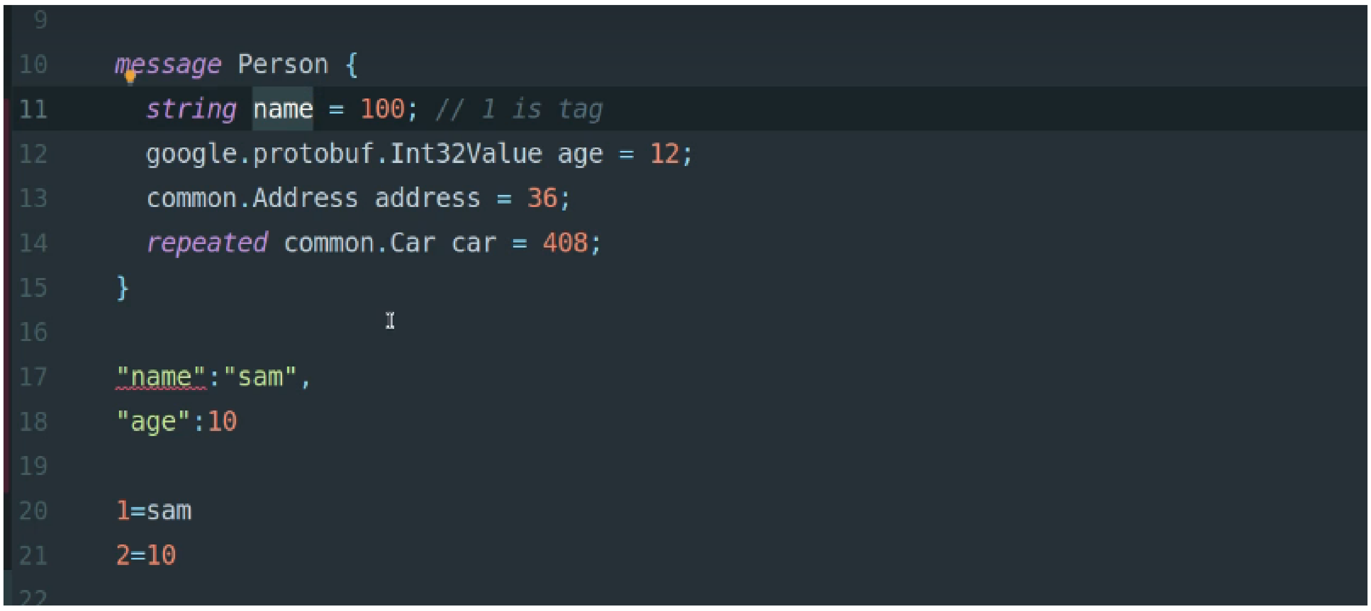
```
public static void main(String[] args) {
    Person person = Person.newBuilder()
                            .setName("AA")
                            .setAddress(Address.newBuilder()
                                                   .setLine("100")
                                                   .build())
                            .addAccount(1000)
                            .addAccount(2000)
                            .build();
    System.out.println(person);
}
```

1.5 Interface

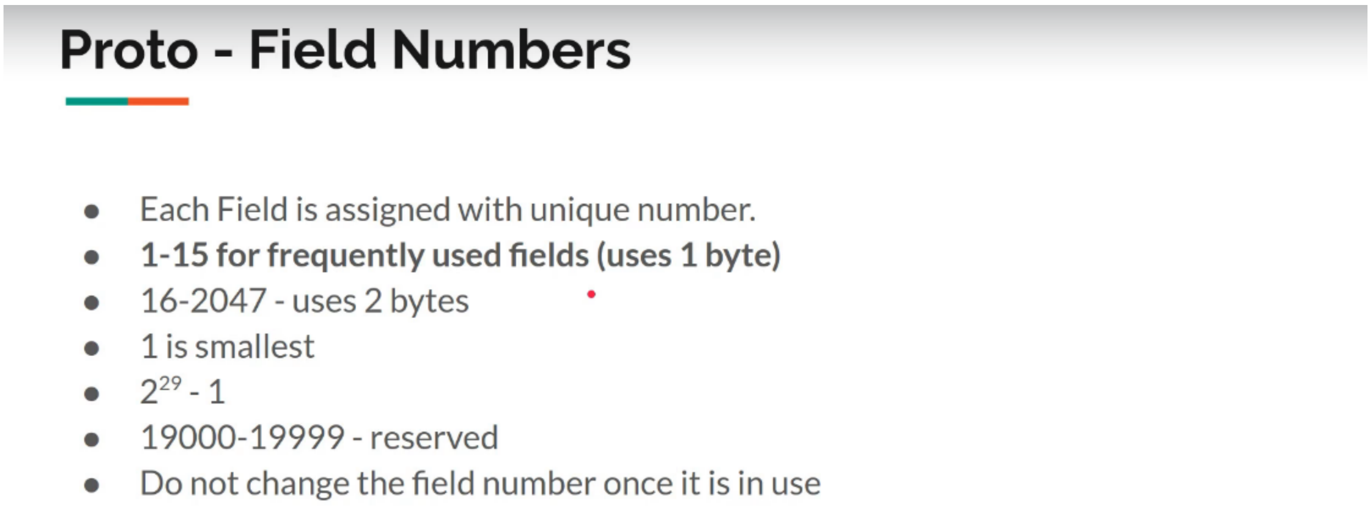
```
message PhoneOTP {
  string phone = 1;
  int32 code = 2;
}


message EmailOTP {
  string email = 1;
  int32 code = 2;
}

message Credentials {
  oneof mode {
    PhoneOTP phone = 1;
    EmailOTP email = 2;
  }
}
```

1.6 Why protobuf is faster

Unlike JSON, client or server have to parse object (require key), protobuf using tag as key indentify position between client and server (both using the same proto file).



```

message Person {
  string name = 100; // 1 is tag
  google.protobuf.Int32Value age = 12;
  common.Address address = 36;
  repeated common.Car car = 408;
}

"name":"sam",
"age":10

1=sam
2=10
```


Proto - Field Numbers

- Each Field is assigned with unique number.
- **1-15 for frequently used fields (uses 1 byte)**
- 16-2047 - uses 2 bytes
- 1 is smallest
- 2^{29} - 1
- 19000-19999 - reserved
- Do not change the field number once it is in use

1.7 Message Changes compability

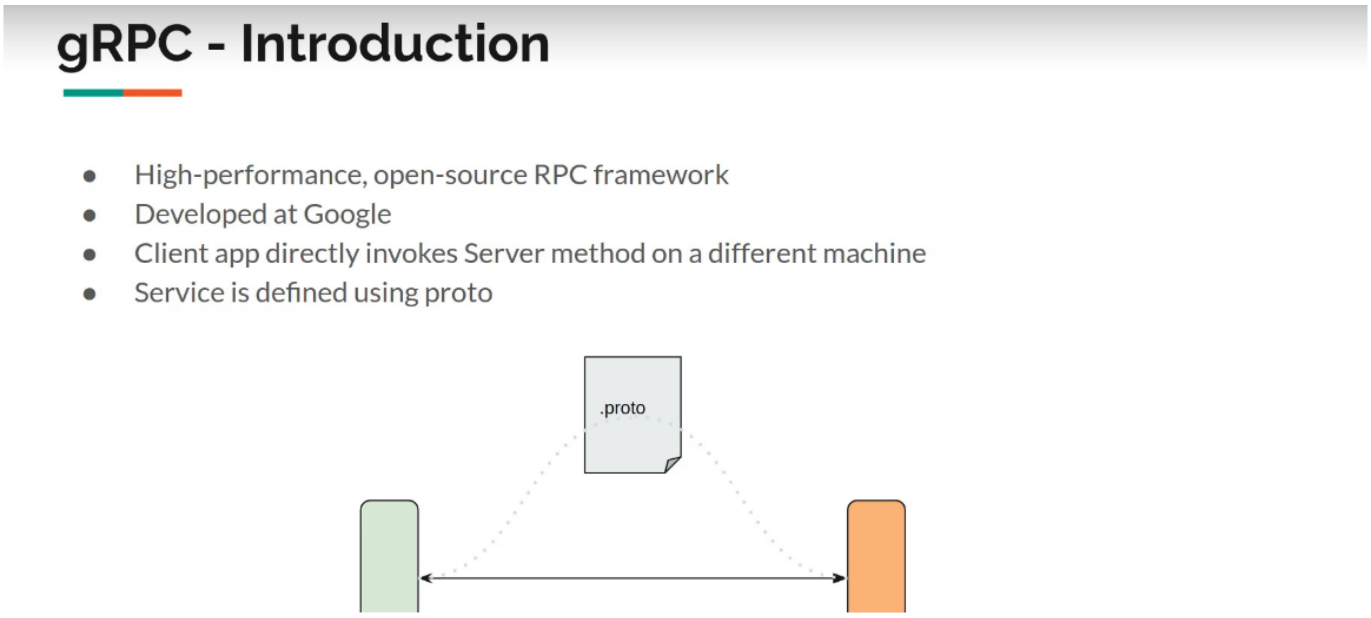
- Default empty string = empty

```
message Employee {
    string first_name = 1;
    int32 age = 2;
    int32 salary = 3;
}
```

Currently use as above, there is requirement to get rid of salary:

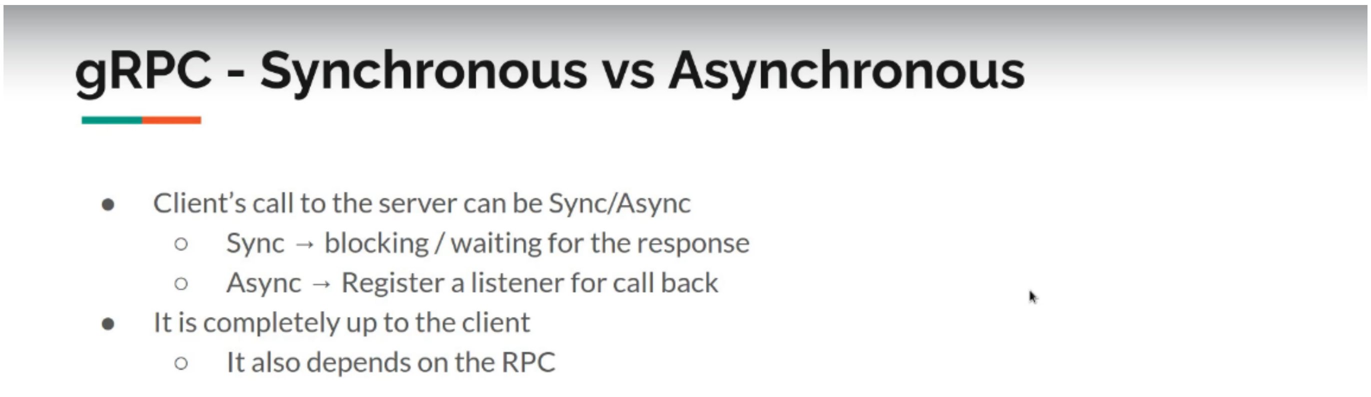
```
message Employee {
    string first_name = 1;
    int32 age = 2;
    reversed 3;
    reversed "salary";
}
```

2. Unary gRPC



gRPC - Introduction

- High-performance, open-source RPC framework
- Developed at Google
- Client app directly invokes Server method on a different machine
- Service is defined using proto

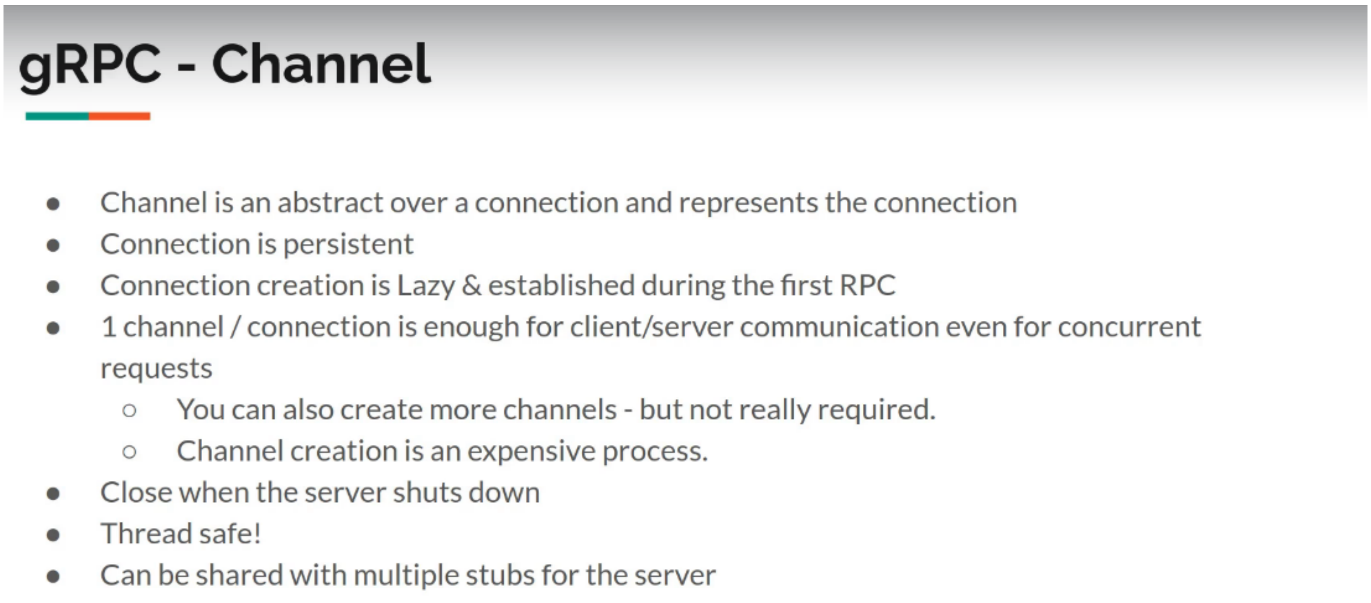


gRPC - Synchronous vs Asynchronous

- Client's call to the server can be Sync/Async
  - Sync → blocking / waiting for the response
  - Async → Register a listener for call back
- It is completely up to the client
  - It also depends on the RPC

3. Stream client - server

4. Load balancing

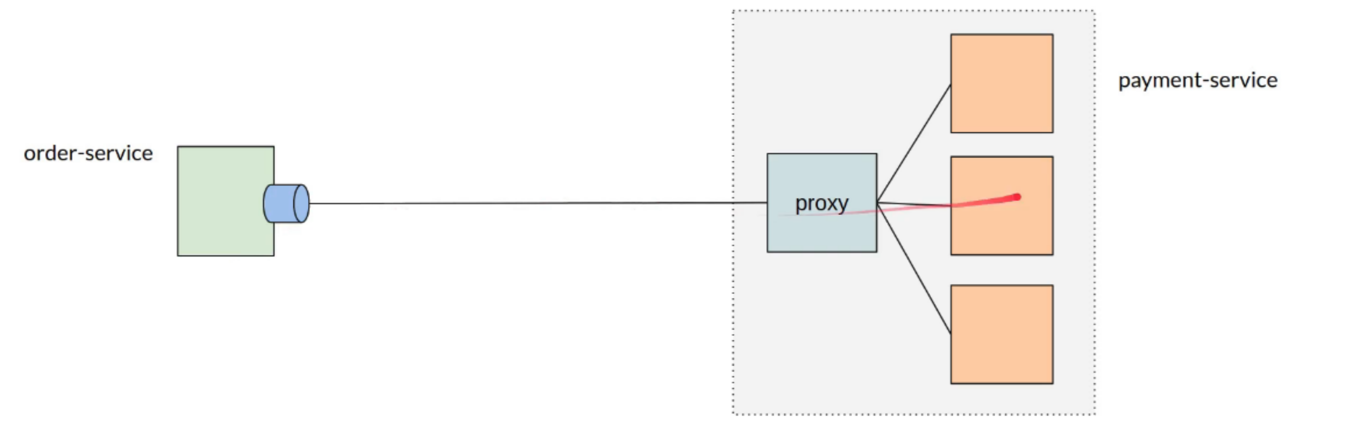


gRPC - Channel

- Channel is an abstract over a connection and represents the connection
- Connection is persistent
- Connection creation is Lazy & established during the first RPC
- 1 channel / connection is enough for client/server communication even for concurrent requests
  - You can also create more channels - but not really required.
  - Channel creation is an expensive process.
- Close when the server shuts down
- Thread safe!
- Can be shared with multiple stubs for the server

- Server side load balanacing, client have to idea of multiple server, it just know the endpoint.

gRPC - Server-Side Load Balancing


order-service
proxy
payment-service