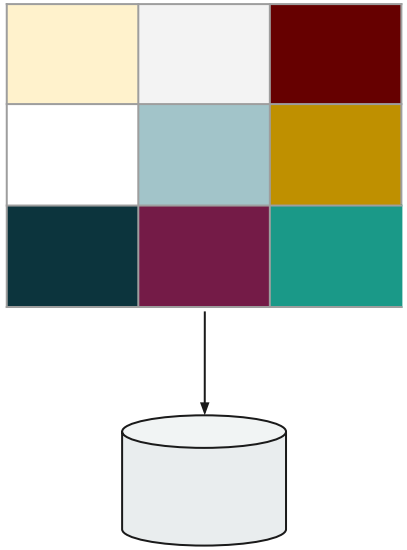
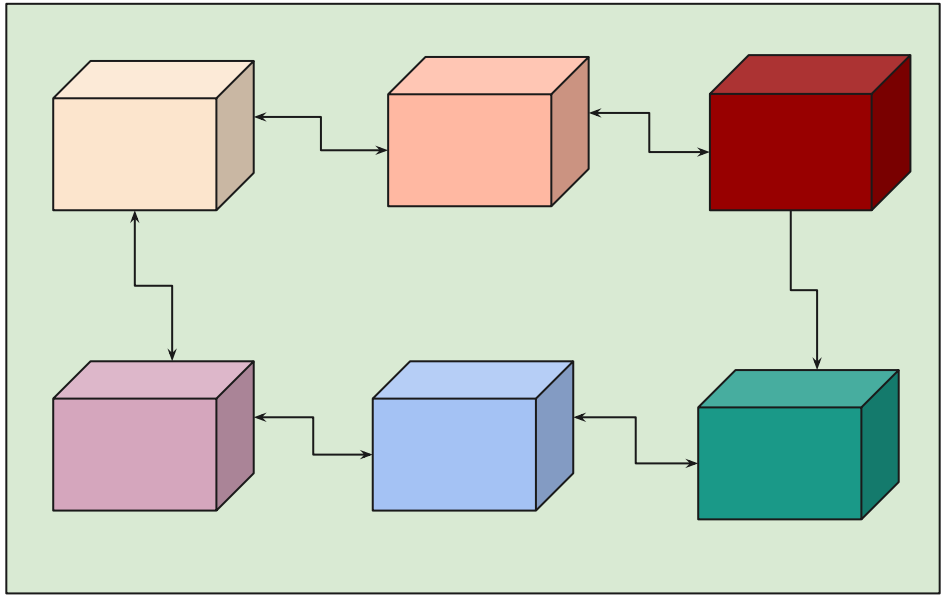
A Complete Guide To gRPC

For Java Developers

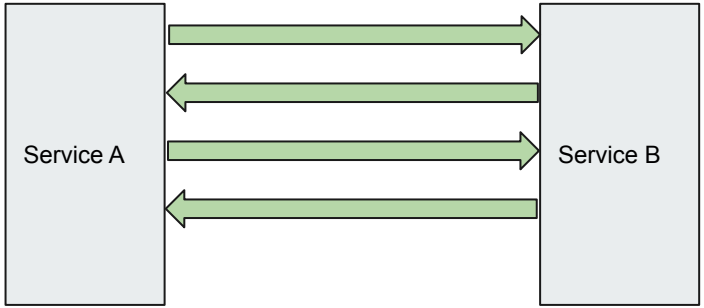
Monolithic Application

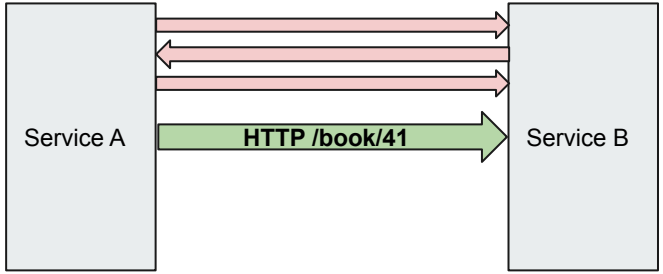


MicroServices



Problem - 1: Request & Response Protocol

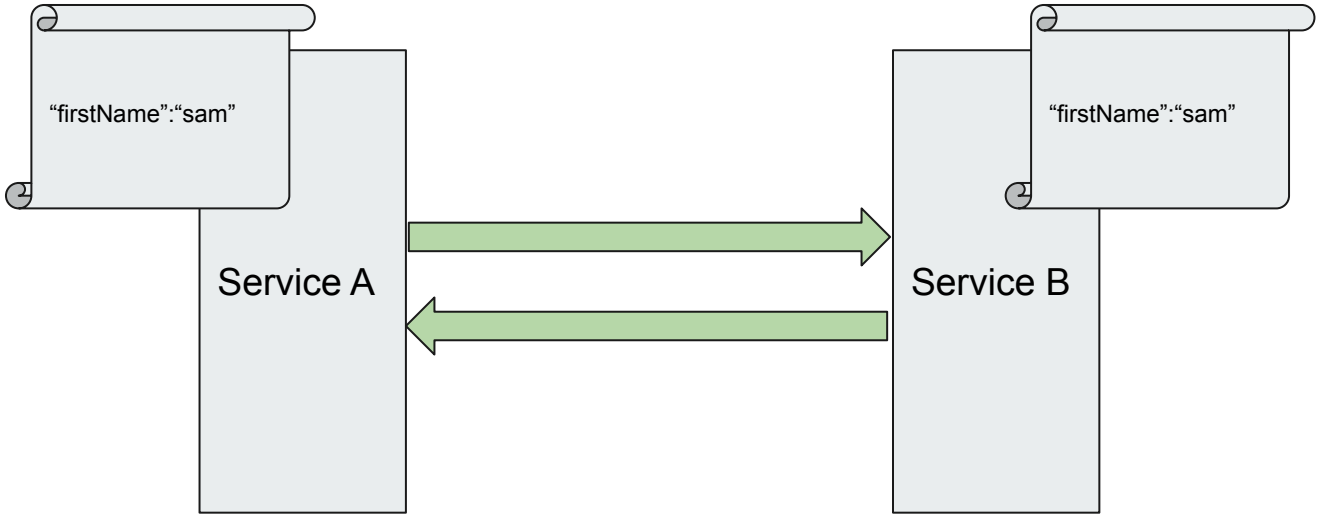




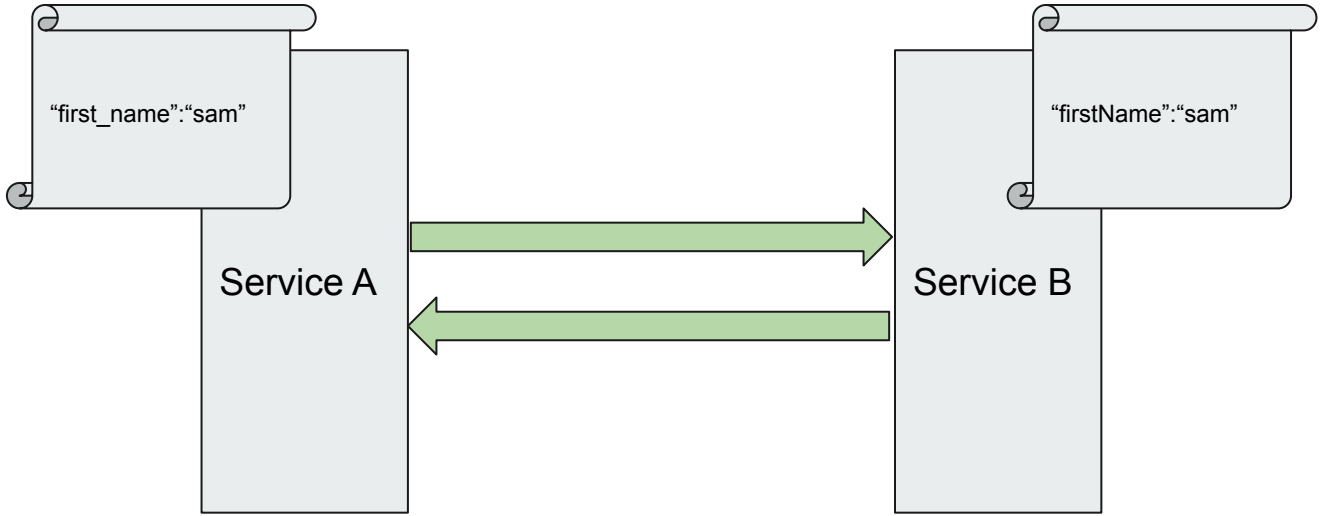
Problem - 2: Headers

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

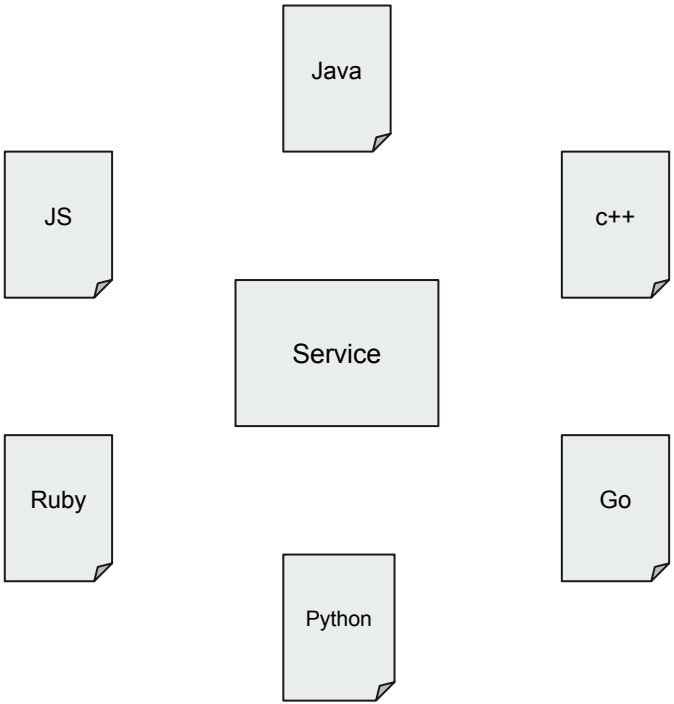
Problem - 3: Serialization & Deserialization



Problem - 4: API Contract

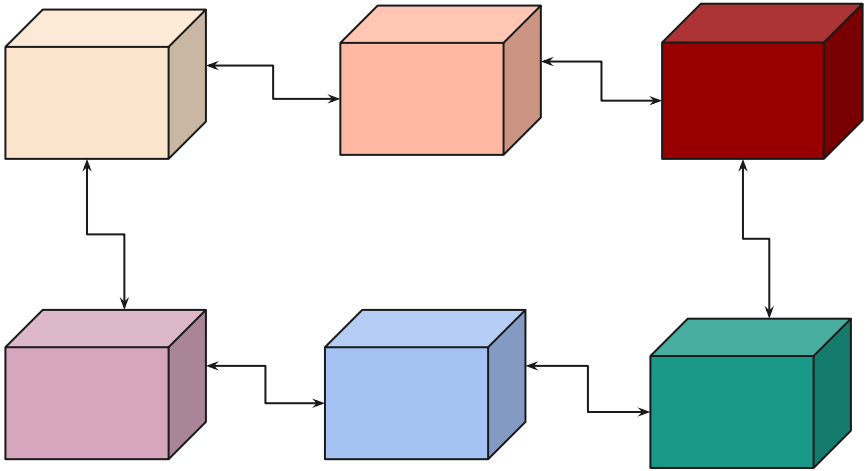


Problem - 5: Client SDK



Stubby

- RPC Framework from Google
- 15 years
- 10 billions reqs / sec!!!!
- Cross-platform
- *Tightly coupled with infrastructure*

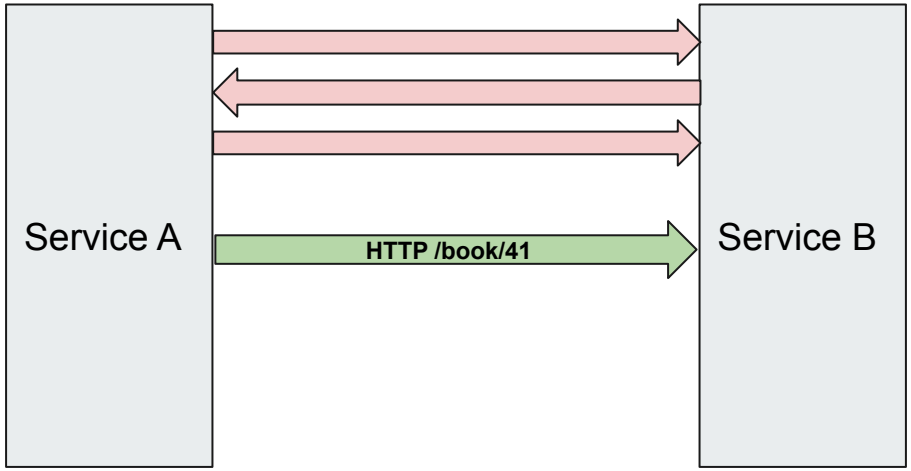


gRPC

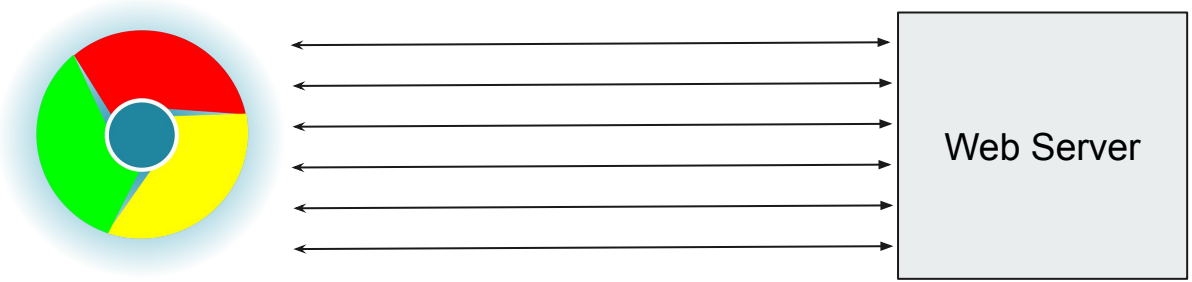
- Developed at Google
- Inspired by Stubby
- Released in 2016
- Adopted by
  - Netflix
  - Microsoft
- Belongs to **CNCF**

HTTP / TCP

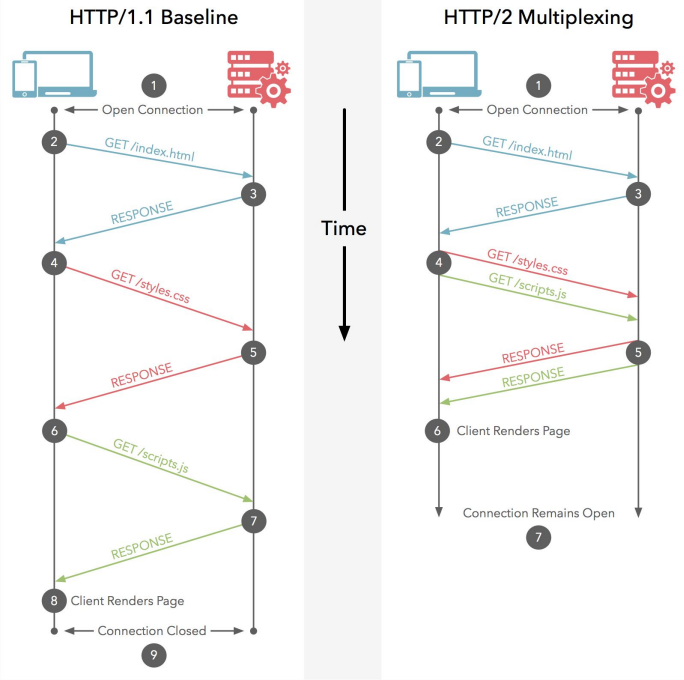
- HTTP 1.1 was released in 1997
- TCP connection - 3 way handshake process
  - Significant amount of time is spent in establishing a connection

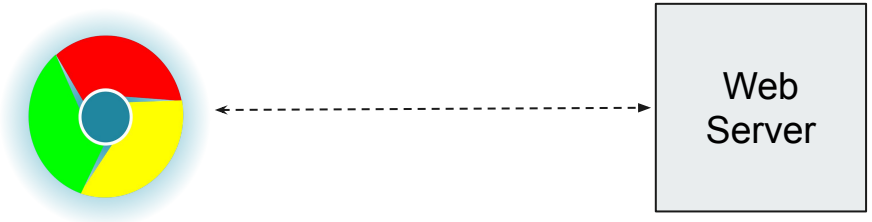


HTTP / TCP



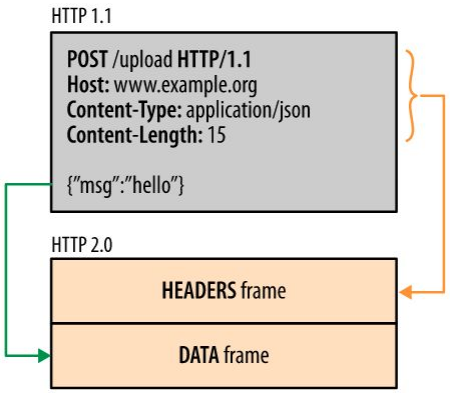
HTTP/1.1 vs HTTP/2

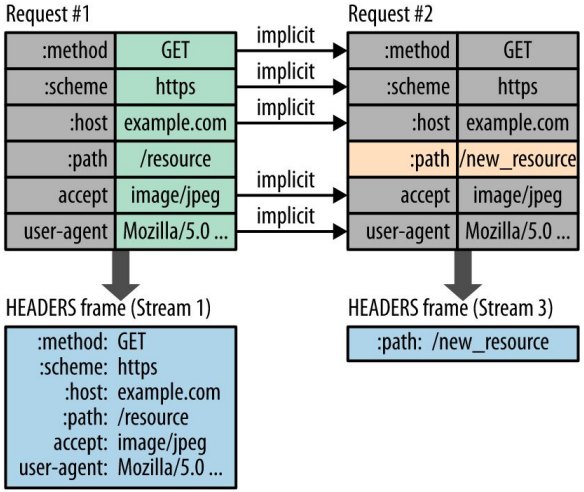




HTTP/2 vs HTTP/1.1

- Binary
- Header Compression
- Flow Control
- Multiplexing





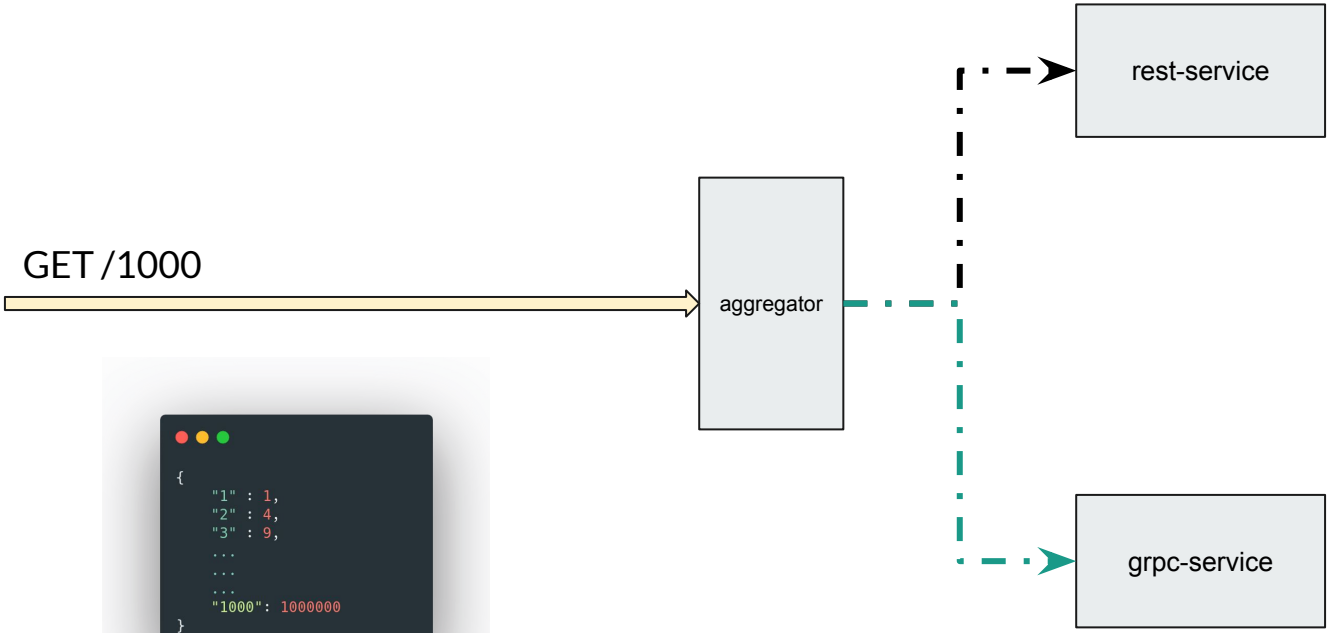
gRPC - Benefits

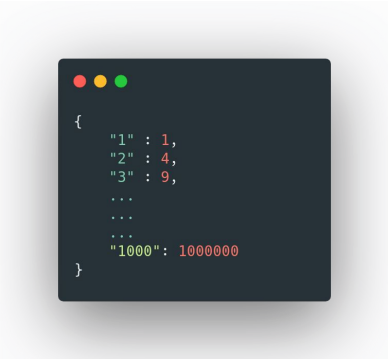
- HTTP2 is default.
  - Binary
  - Multiplexing
  - Flow-control
- Non-blocking, Streaming bindings
- ***Protobuf***
  - Strict Typing
  - DTO
  - Service definitions
  - Language-agnostic
  - Auto-generated bindings for multiple languages
- Great for mobile apps

gRPC vs REST

- REST
  - Architectural style
  - Resource oriented
  - ~JSON + HTTP
- gRPC
  - A RPC framework
  - More flexible & Action oriented
  - Specific to inter-services communication

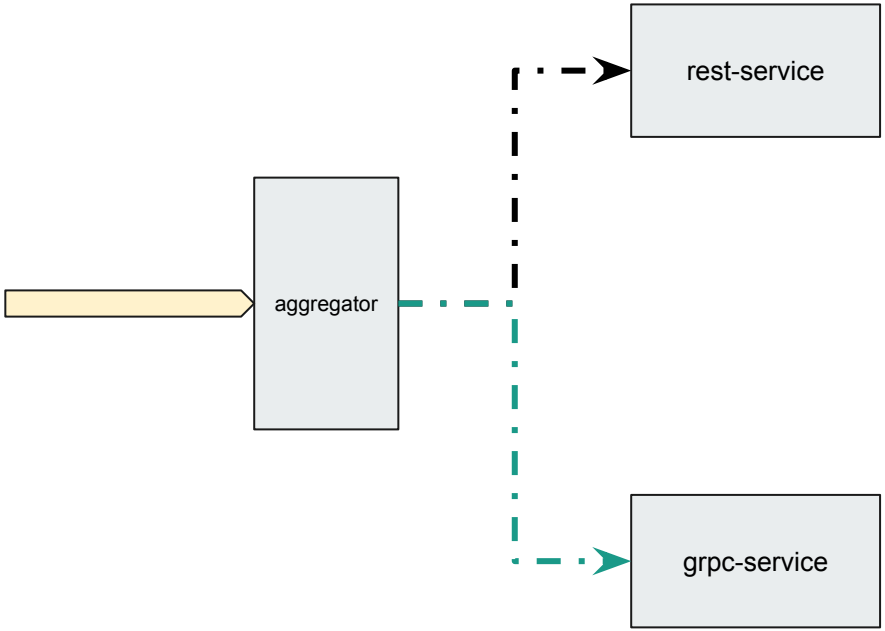
gRPC vs REST


GET /1000
aggregator
rest-service
grpc-service


{
"1" : 1,
"2" : 4,
"3" : 9,
...
...
...
"1000": 1000000
}

gRPC vs REST

```
// REST
rest/unary/1000

// gRPC
grpc/unary/1000
grpc/stream/1000
```



Course Structure

- Protocol Buffers / protobuf
- gRPC
  - Client/Server application
  - Different types of RPC
  - Client/Server - Game assignment
  - Load Balancing
  - Authentication / Metadata / Contexts
  - Error Handling
  - Deadline
  - Intercepting the requests
- ***Bonus***: gRPC - Spring Boot Integration

Note

- Assumption: Experience with Java 8+
- Pace could be slow or fast depending on your experience :(
    - **But You would be comfortable with gRPC**
- Would like to show things from scratch

Protocol Buffers

Introduction

- IDL (interface description language) for API
- Platform neutral
- Language neutral
- Serializing/Deserializing structured data
- Very Fast / Optimized for interservices communication
- Provides client libraries automatically for many languages!
  - Java
  - C++
  - Javascript
  - Go
  - Ruby
  - C#
  - Python

Sample Proto

```
public class Person {

    private String name;
    private int age;

     // getters
     // setters

}
```

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

Protobuf vs JSON - Performance Comparison

Types

Java Type	Proto Type
int	int32
long	int64
float	float
double	double
boolean	bool
String	string
byte[]	bytes

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;
}
```

Proto - Collections & Map

Java Type	Proto Type
Collection / List	repeated
Map	map

Proto - Enum

```
enum Payment {
    CREDITCARD = 0;
    CASH = 1;
    BITCOIN = 2;
}
```

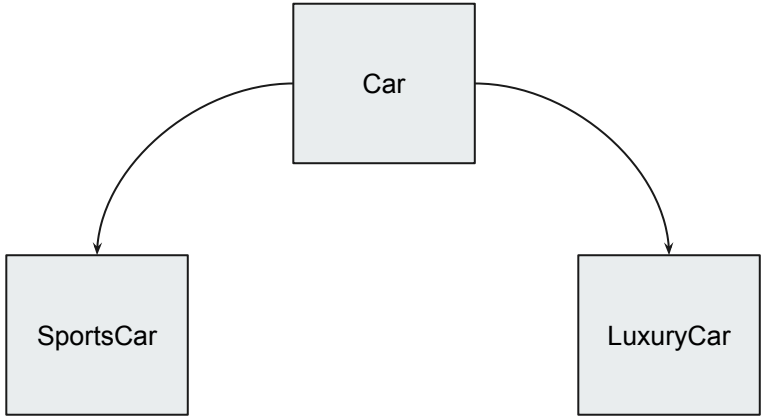
Default Values

Proto Type	Default
int32 / any number type	0
bool	false
string	empty string
enum	first value
repeated	empty list
map	wrapper / empty map

Proto - Modules / Import

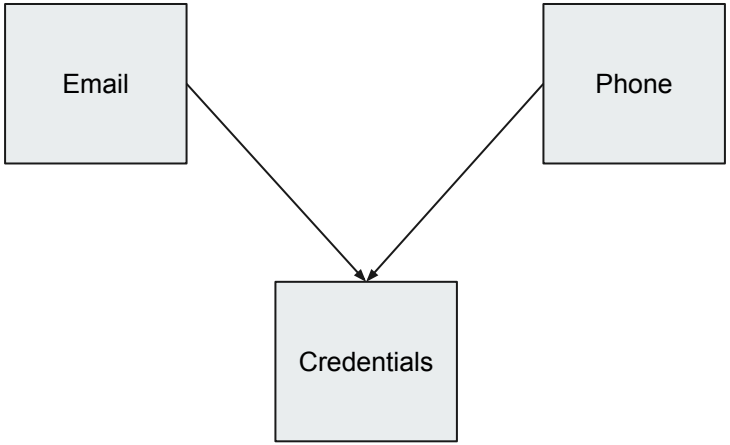
- Proto files can be packaged & imported

Java - Interface / Implementations




```java
drive(new LuxuryCar());
drive(new SportsCar());

//...
//...

public void drive(Car car){
    car.drive();
}
```

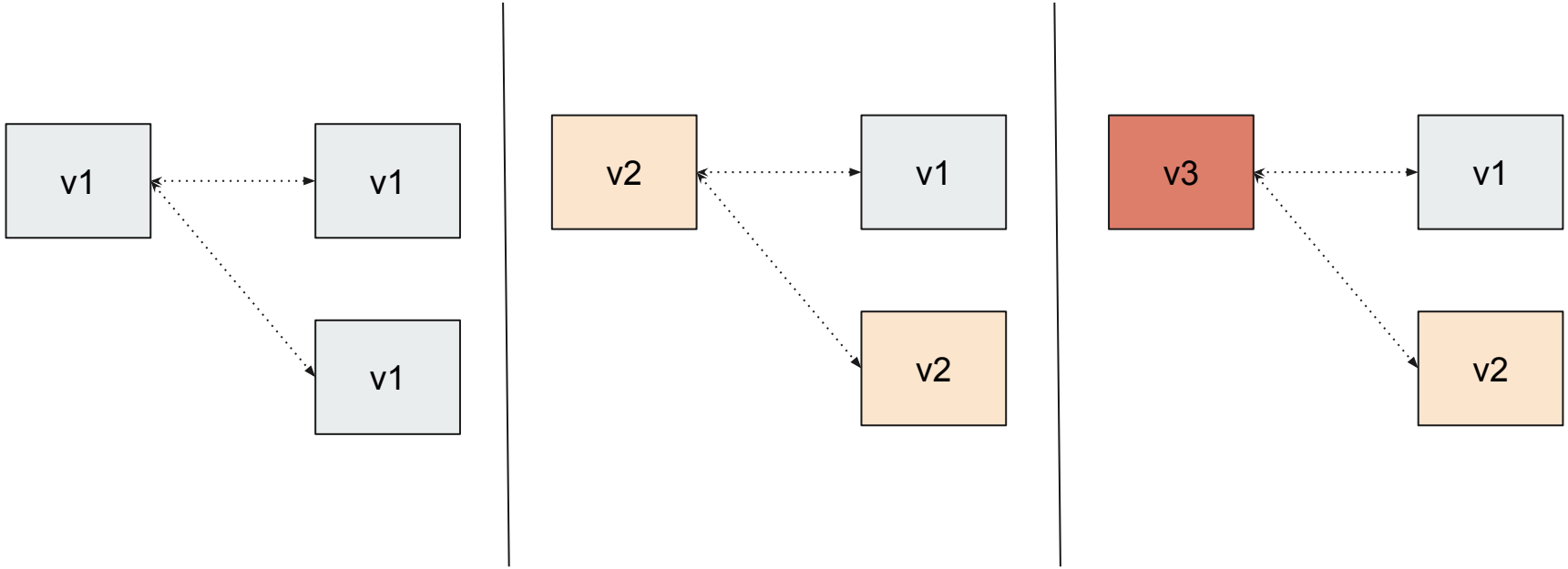
Proto - OneOf



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

Proto - Message Format Changes



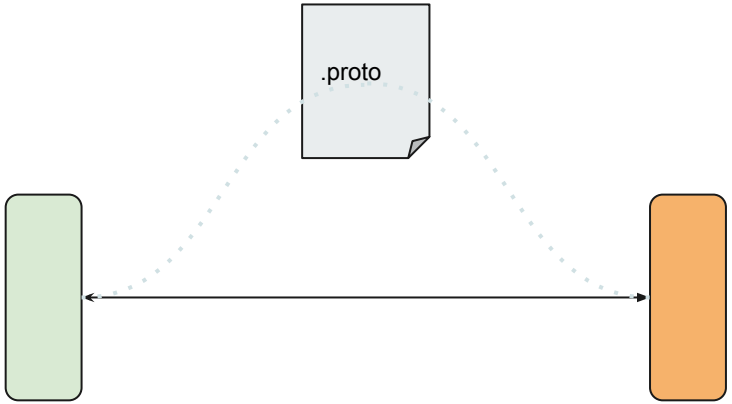
Proto - Guidelines

- 1-15 for frequently used fields
- Do not reorder the fields once it is in use
- Adding New fields will not break old proto
- Removing fields will not break old proto
  - Use reserved
- Changing Type
  - int32 → int64 is OK
  - int64 → int32 might be a problem
- Renaming is OK. but be cautious.
- Keep the protos as separate maven-module and add them as dependency in other modules

gRPC

gRPC - Introduction

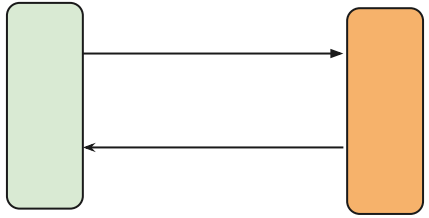
- High-performance, open-source RPC framework
- Developed at Google
- Client app directly invokes Server method on a different machine
- Service is defined using proto



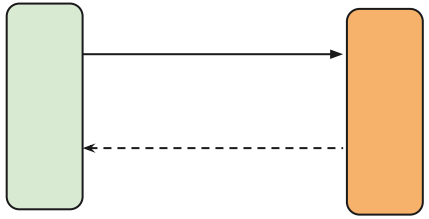
gRPC - Synchronous vs Asynchronous

- 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

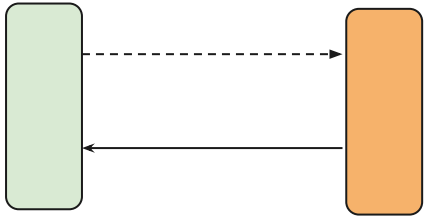
RPC - Types



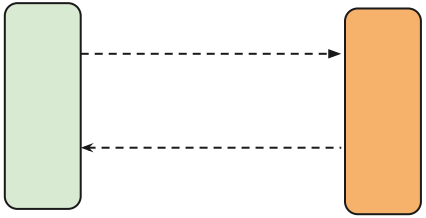
Unary



Server-streaming

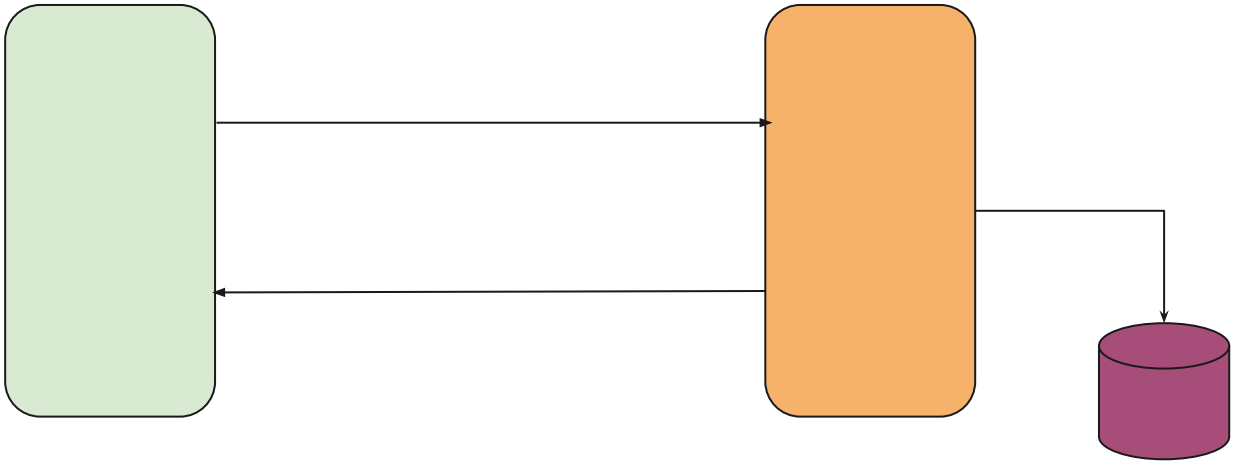


Client-streaming

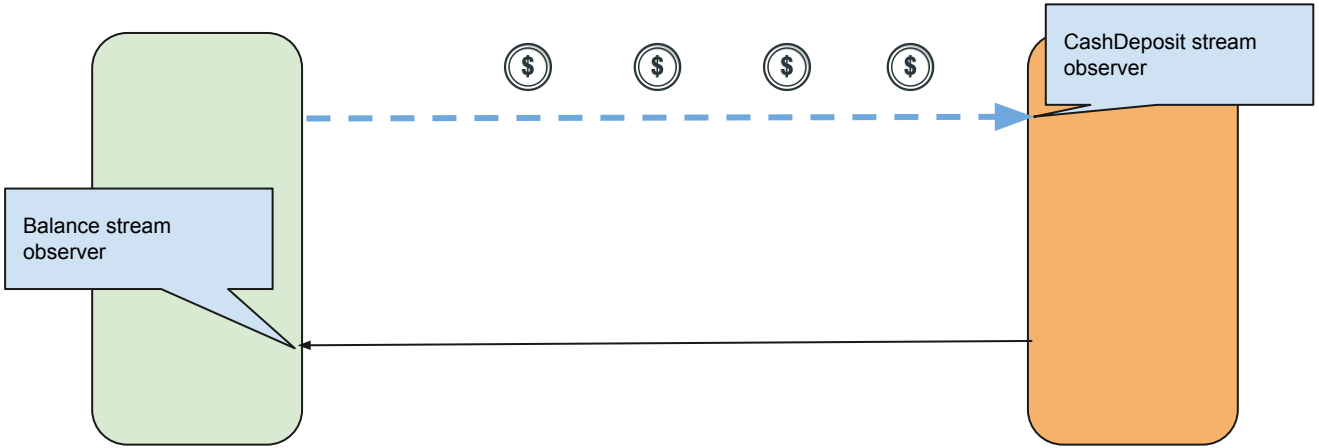


Bidirectional-streaming

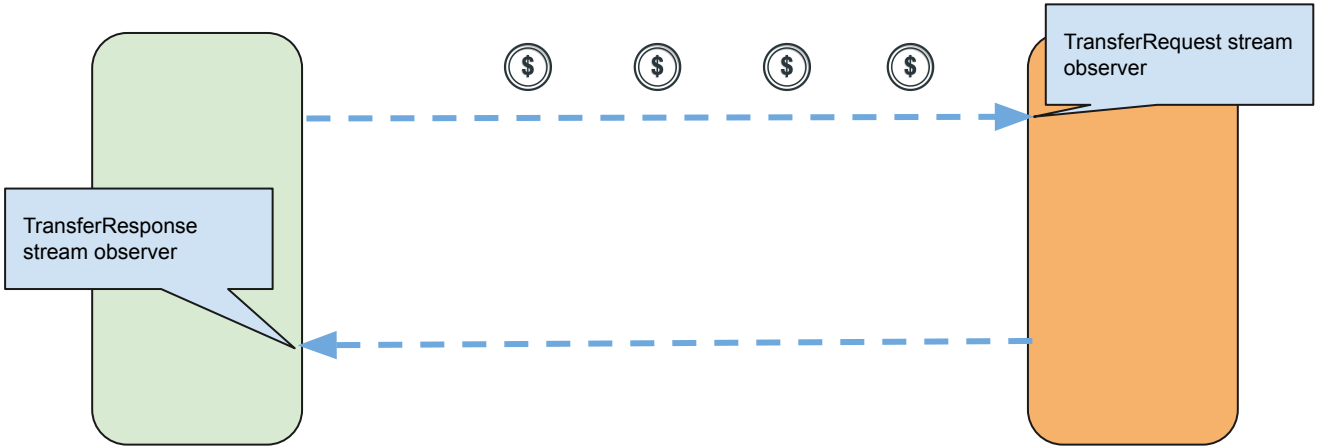
gRPC - Services Communication



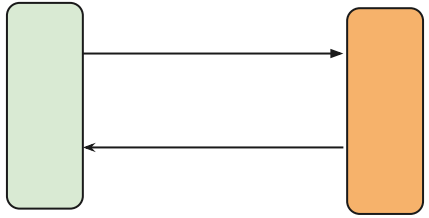
StreamObserver



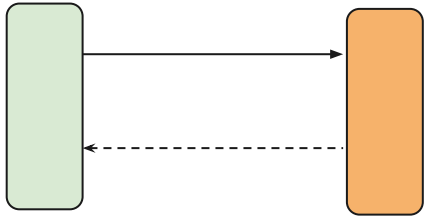
Bi-directional Stream



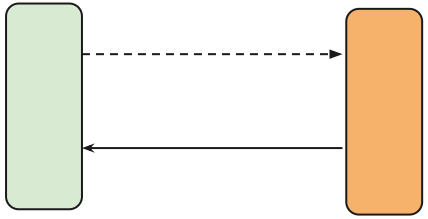
RPC - Types



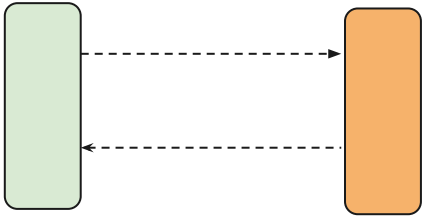
Unary



Server-streaming



Client-streaming



Bidirectional-streaming

Response - Streaming vs Single

- Streaming
  - Size - Potentially large / unknown
    - Ex: Pagination
    - Uber-eats
  - Receiving side might take too much time to process
    - File upload
  - More efficient than multiple RPC calls
  - Bi-Directional Stream → Client/Server streams are completely independent
- Single
  - More efficient than streaming RPC
  - Size is small

REST - CRUD

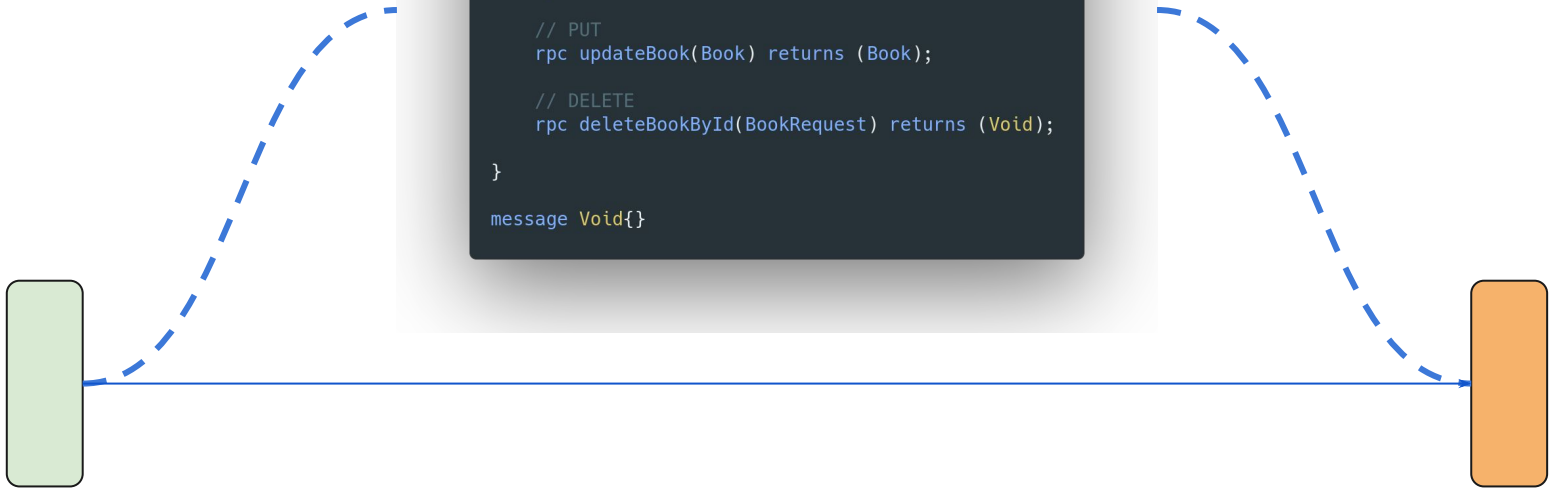
GET

POST

```java
@RestController
public class BookResource {

    @GetMapping("/book/{id}")
    public Book getBook(@PathVariable int id){
        return bookService.getBookById(id);
    }

    @PostMapping("/book")
    public Book saveBook(@RequestBody Book book){
        return bookService.saveBook(book);
    }

    @PutMapping("/book")
    public void updateBook(@RequestBody Book book){
        return bookService.updateBook(book);
    }

    @DeleteMapping("/book/{id}")
    public void delete(@PathVariable int id){
        return bookService.deleteBookById(id);
    }
}
```

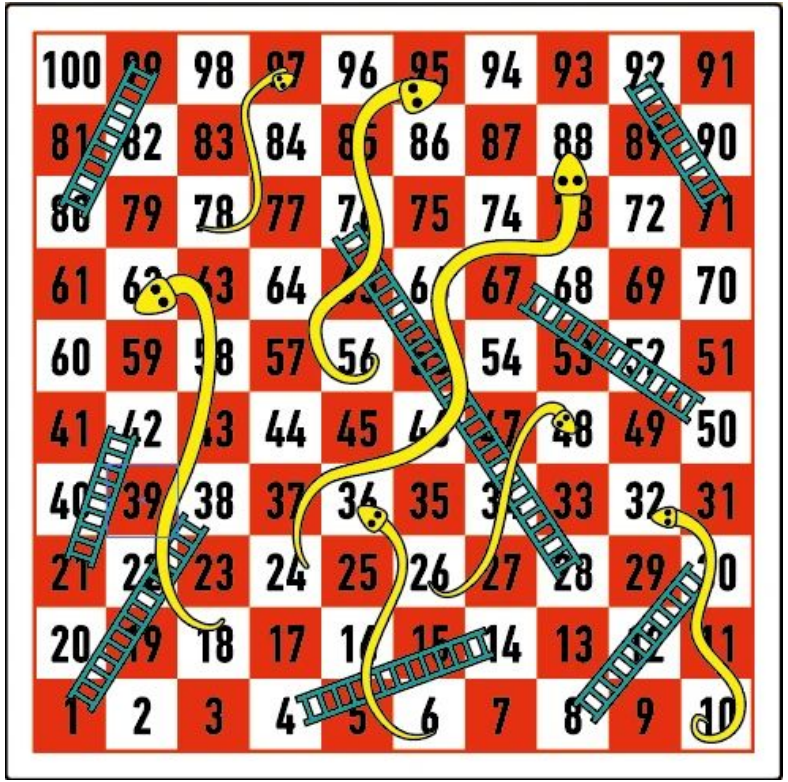
gRPC - CRUD

```
service BookService {

    // GET
    rpc getBookById(BookRequest) returns (Book);

    // POST
    rpc saveBook(Book) returns (Book);

    // PUT
    rpc updateBook(Book) returns (Book);

    // DELETE
    rpc deleteBookById(BookRequest) returns (Void);

}

message Void{}
```



Snakes & Ladders

Snakes & Ladders

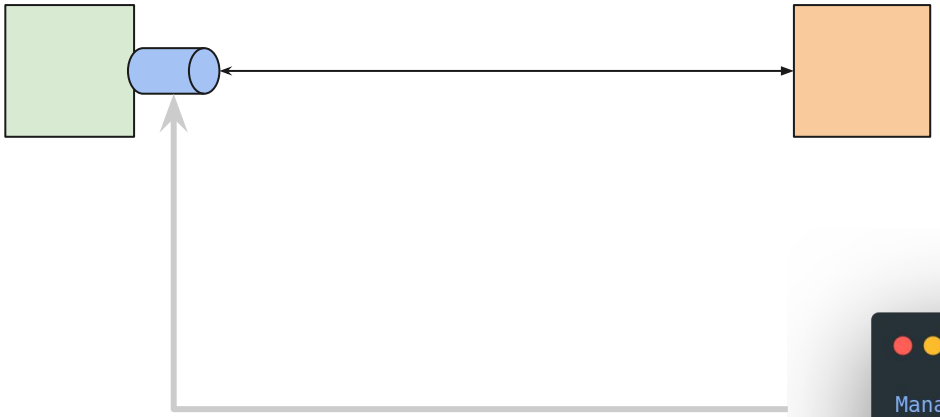


Hints

- Client and Server positions start from 0
- Server maintains both client's and server's position
- Client takes the first turn always!
- Client sends its die value to the server
- Server updates the client's position
- Server rolls the die and updates its position
- Server sends the GameState - both client's and server's position once the Server's turn is over or Client wins.
- Client prints the game state as and when it receives the game state
- Game is over when either of them wins!

gRPC - Load Balancing

gRPC - Channel

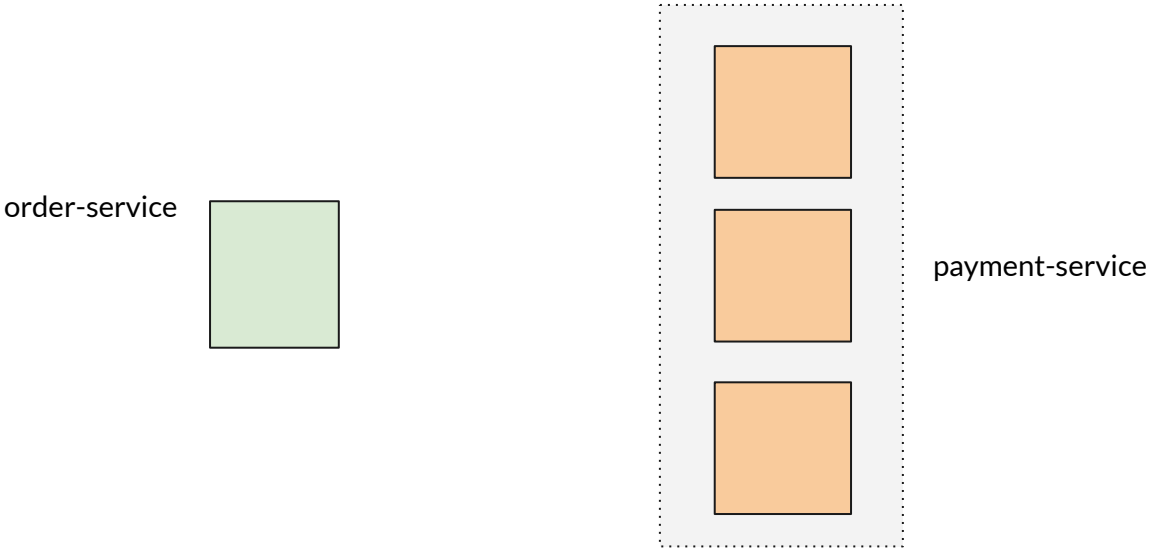


```
ManagedChannelBuilder.forAddress("localhost", 6565)
                .usePlaintext()
                .build();
```

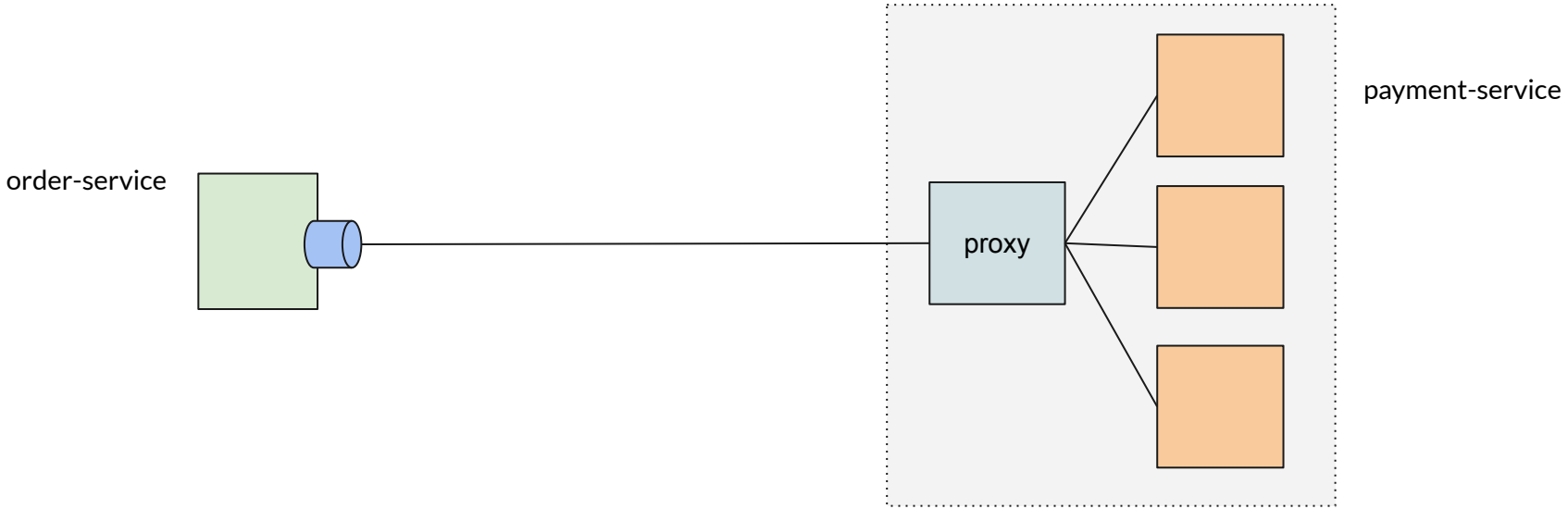
gRPC - Channel

- 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

gRPC - Load Balancing



gRPC - Server-Side Load Balancing

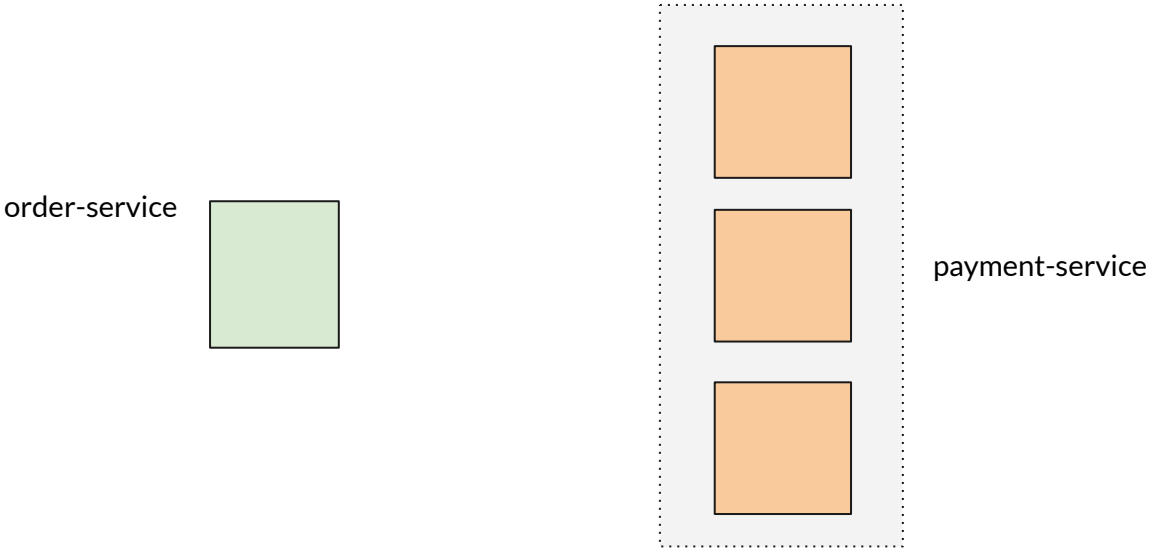


nginx

```
cd c:\
unzip nginx-1.19.3.zip
cd nginx-1.19.3
start nginx
```

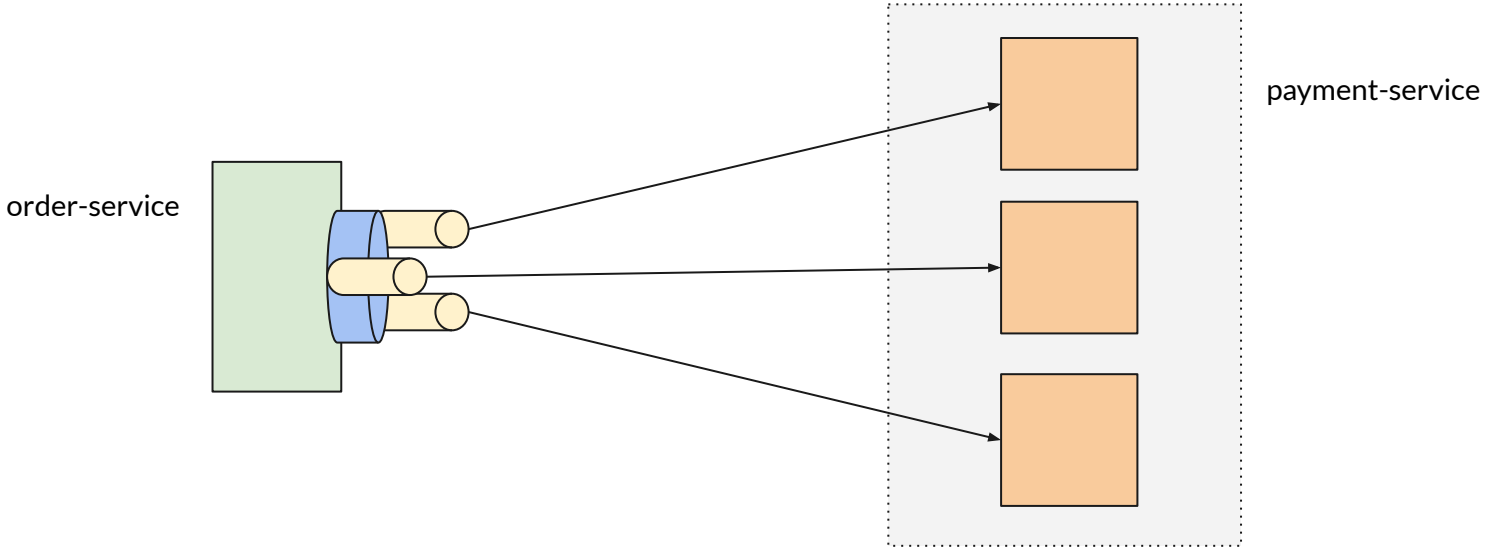
```
// install
brew install nginx

// conf
/usr/local/etc/nginx/nginx.conf
```

gRPC - Load Balancing

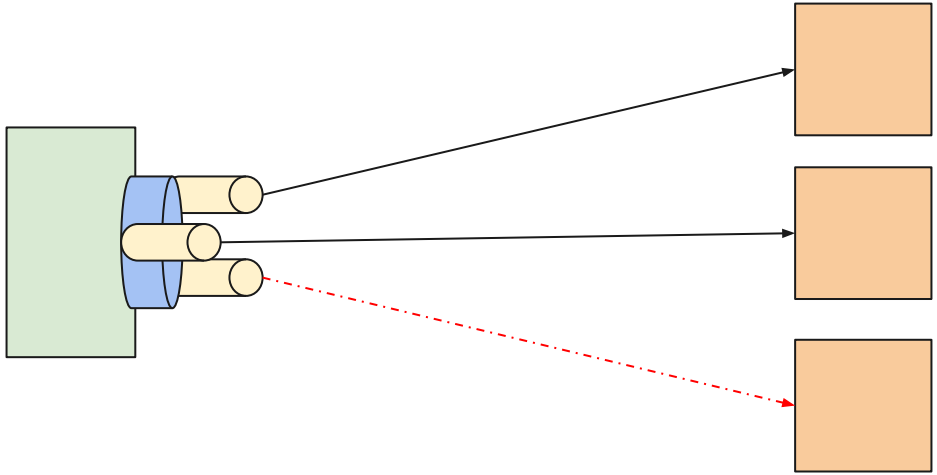


gRPC - SubChannels

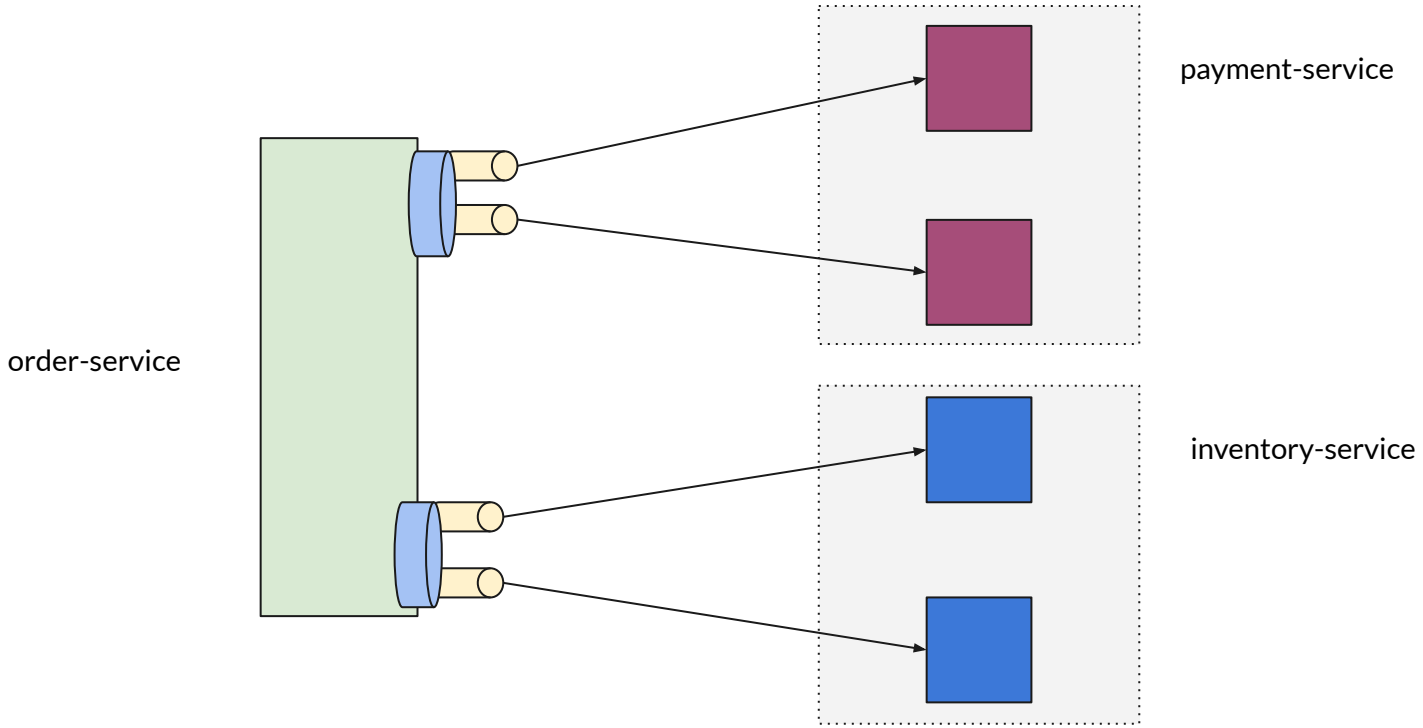
- A Channel can have many subchannels - each subchannel represents a connection to the server.
- Channel chooses subchannel in round robin fashion *(not default)*



gRPC - SubChannels



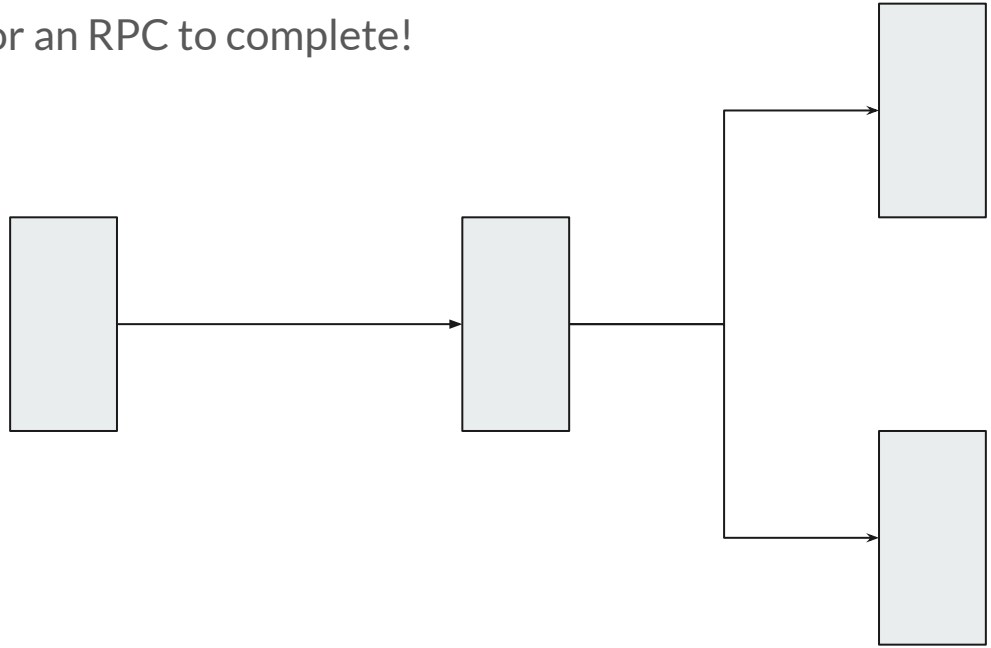
gRPC - Channels / SubChannels



gRPC - Advanced Topics

gRPC - Deadline

- Timeout for an RPC to complete!

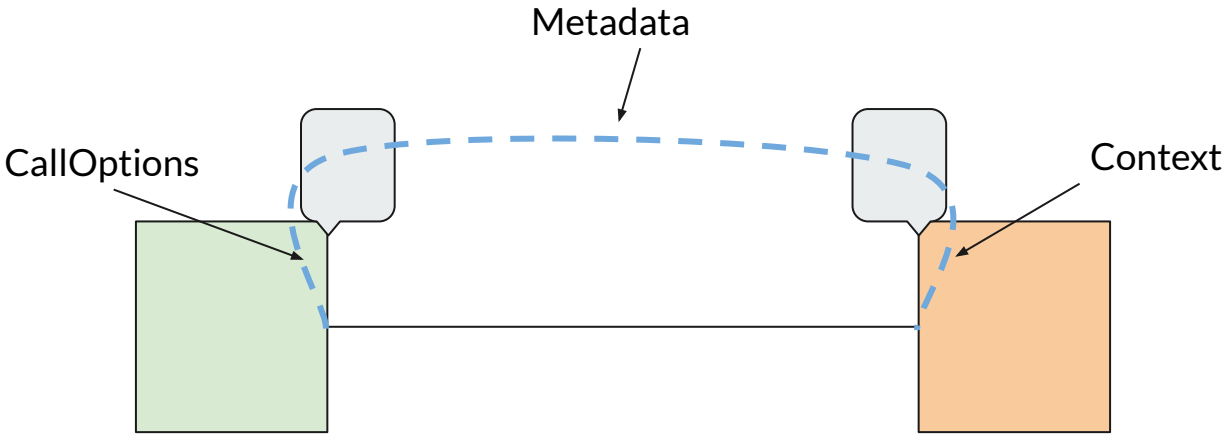


gRPC - Interceptor

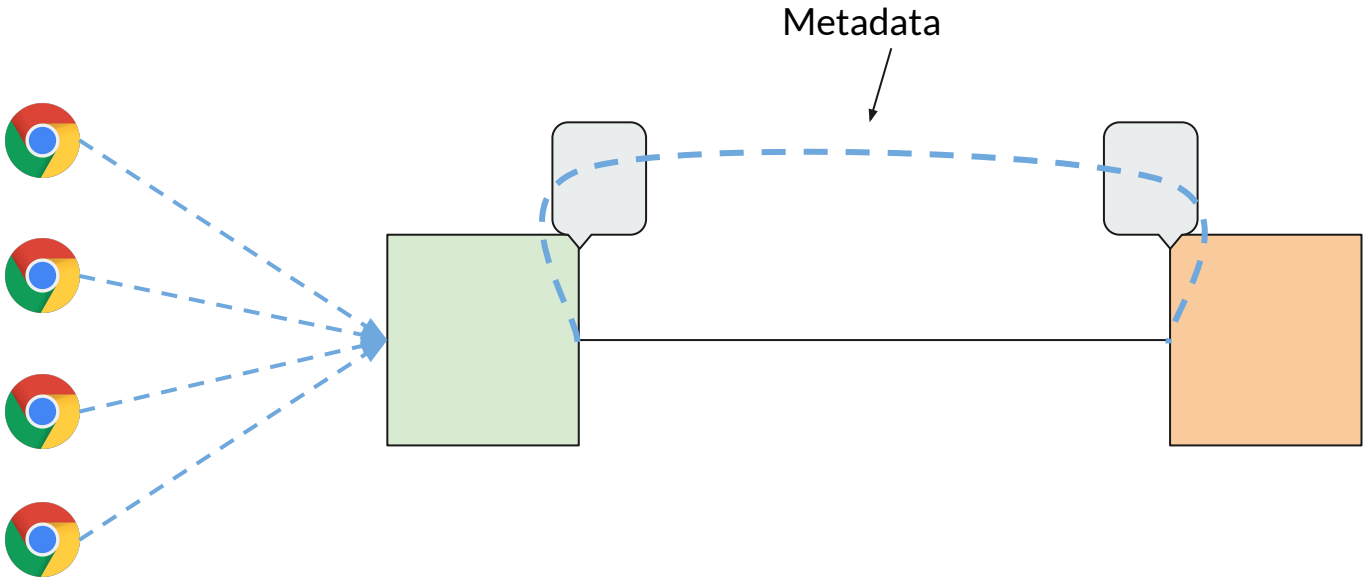
- We can intercept RPCs both at client and server side.
- To handle Cross-cutting Concerns

gRPC - Interceptor

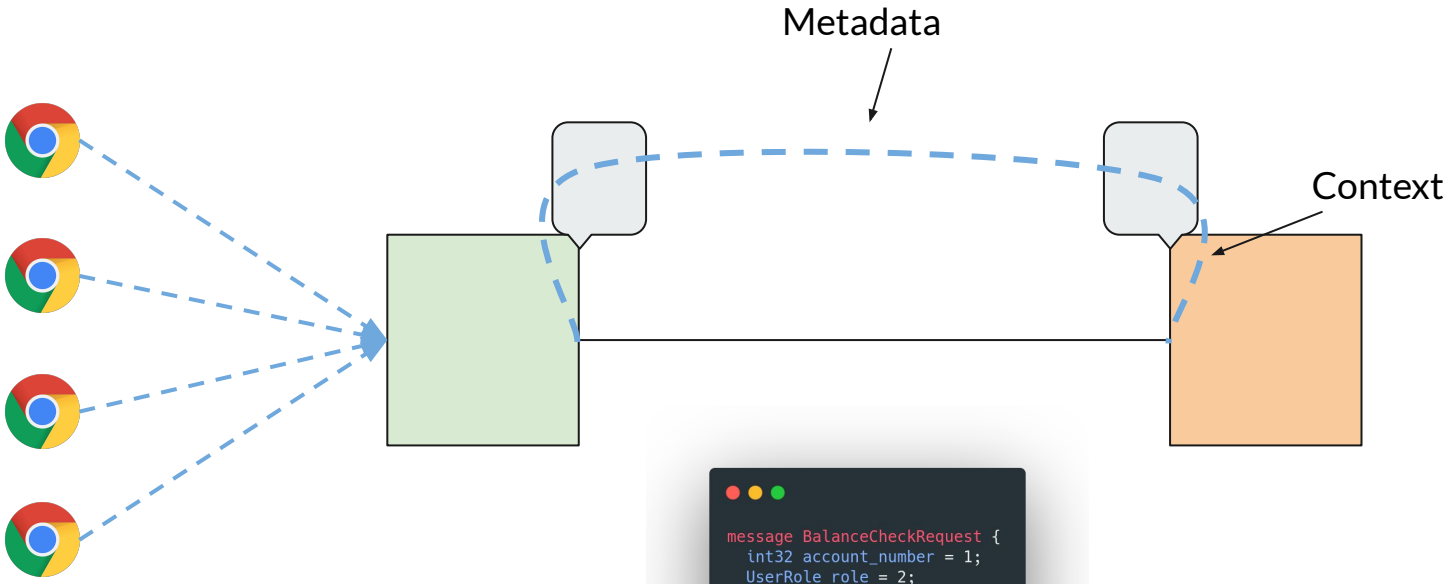
gRPC - Key/Value Pairs



gRPC - Attaching User Token



gRPC - Passing User Role via Context




```
message BalanceCheckRequest {
  int32 account_number = 1;
  UserRole role = 2;
}
```

gRPC - Metadata

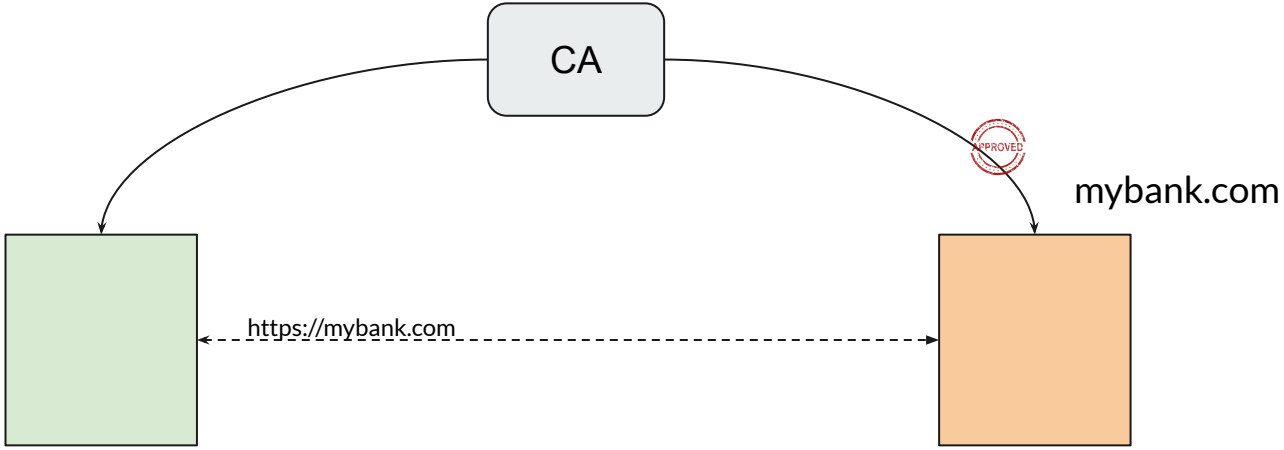
- Key/Value pairs
- Value can be a String or Binary data or an Object
- To pass information between Client & Server

gRPC - Error Handling

- Error channel
  - Status codes
  - Metadata
- Data channel
  - OneOf

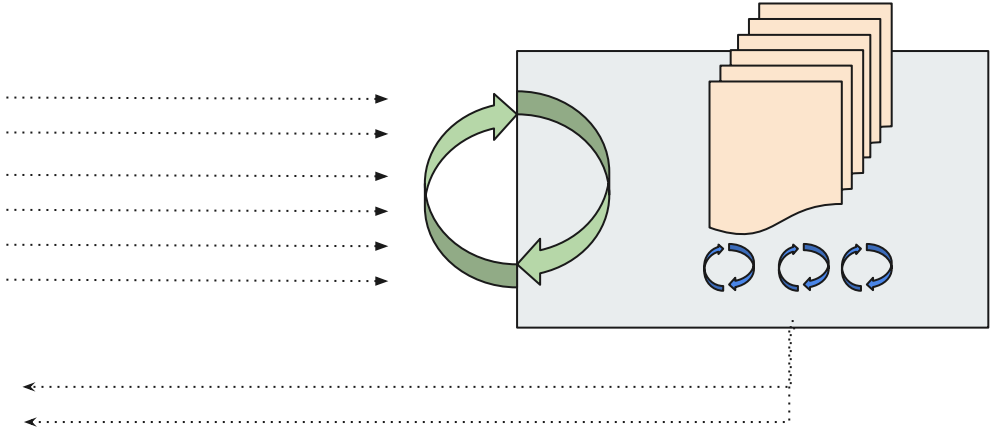
```
service BankService {

  rpc withdraw(WithdrawRequest) returns(WithdrawResponse);

}

message WithdrawResponse {
    oneof response {
        Money money = 1;
        Error error = 2;
    }
}
```

gRPC - SSL/TLS



gRPC - Executor

- CachedThreadPool - default
- ***directExecutor*** → Better performance for non-blocking app
- FixedThreadPool → Preferred



gRPC - Executor

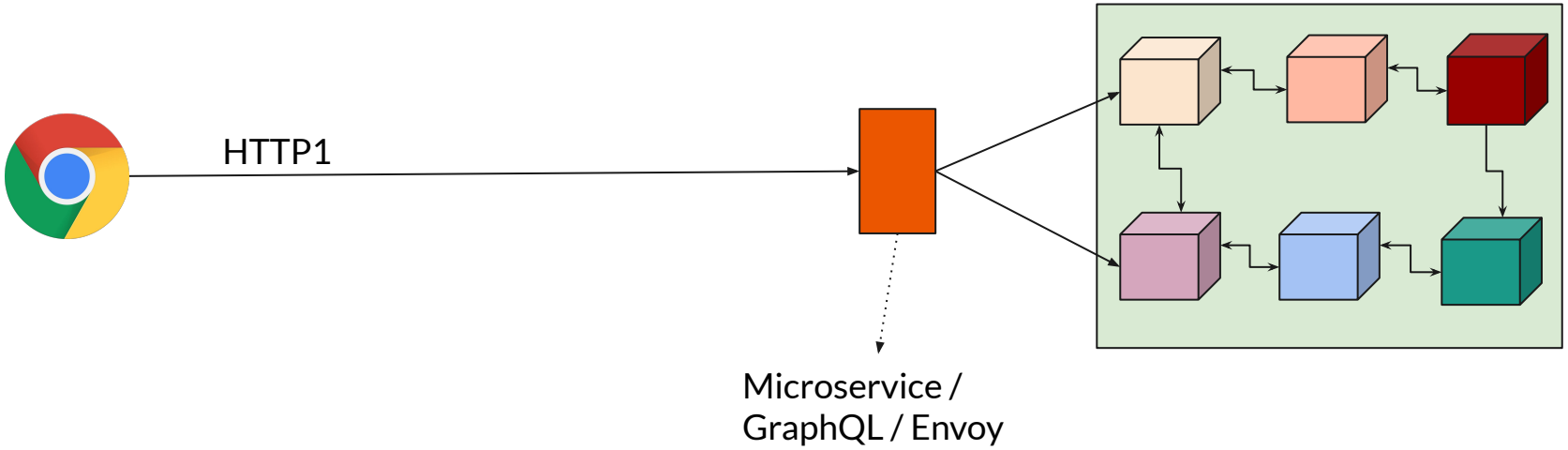
```
ServerBuilder.forPort(6565)
              .directExecutor()
              .addService(new GameService())
              .build();
```

```
ServerBuilder.forPort(6565)
              .executor(Executors.newFixedThreadPool(20))
              .addService(new GameService())
              .build();
```

gRPC - Unit Testing

- InProcessServerBuilder
- InProcessChannelBuilder

gRPC - Web



gRPC - Spring Boot Integration

gRPC - Spring Boot Integration - Server

```
<dependency>
    <groupId>net.devh</groupId>
    <artifactId>grpc-server-spring-boot-starter</artifactId>
    <version>2.9.0.RELEASE</version>
</dependency>
```

```
grpc.server.port=6565
```

```
@GrpcService
public class CalculatorService extends CalculatorServiceGrpc.CalculatorServiceImplBase {

    @Override
    public void getAllDoubles(Input request, StreamObserver<Output> responseObserver) {
        int index  = request.getNumber();
        IntStream.rangeClosed(1, index)
                .map(i -> i * 2) // add Thread.sleep to simulate time consuming operation
                .mapToObj(i -> Output.newBuilder().setResult(i).build())
                .forEach(responseObserver::onNext);
        responseObserver.onCompleted();
    }
}
```

gRPC - Spring Boot Integration - Client

```
<dependency>
    <groupId>net.devh</groupId>
    <artifactId>grpc-client-spring-boot-starter</artifactId>
    <version>2.9.0.RELEASE</version>
</dependency>
```

```
grpc:
  client:
    calculator-service:
      address: static://localhost:6565
      negotiationType: plaintext
```

```
@GrpcClient("calculator-service")
private CalculatorServiceGrpc.CalculatorServiceBlockingStub blockingStub;
```

gRPC - Spring Boot Integration