Introduction to grpc

Ramveer Singh
Software Engineer Athenahealth Bangalore
Dated: 17th Oct 2019

Agenda

- REST
- HTTP/1 and HTTP2
- JSON and protobuff
- RPC
- Learn GRPC

How do we make computers talks to each others

How do we make Apps talks to each others

How do we build web APIs

SOAP


SOAP

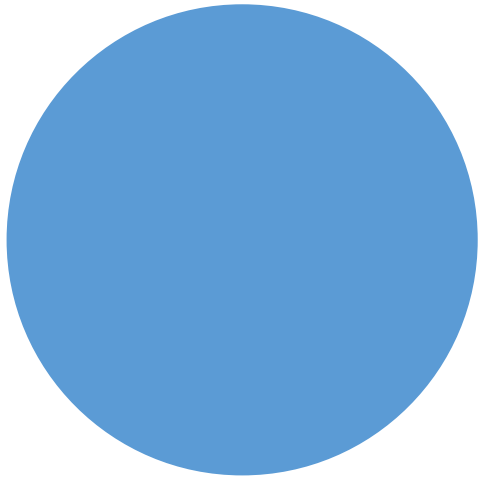
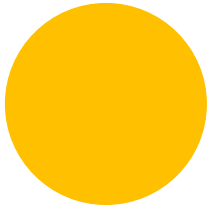
REST

HTTP+JSON

(REST)

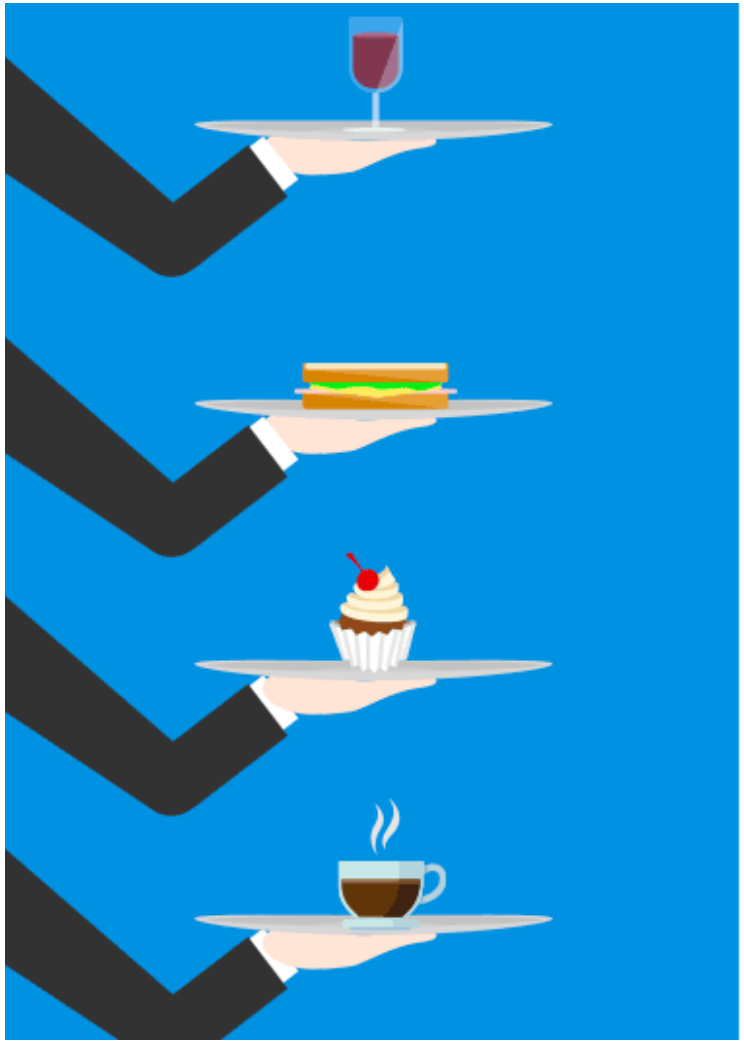
REST APIs

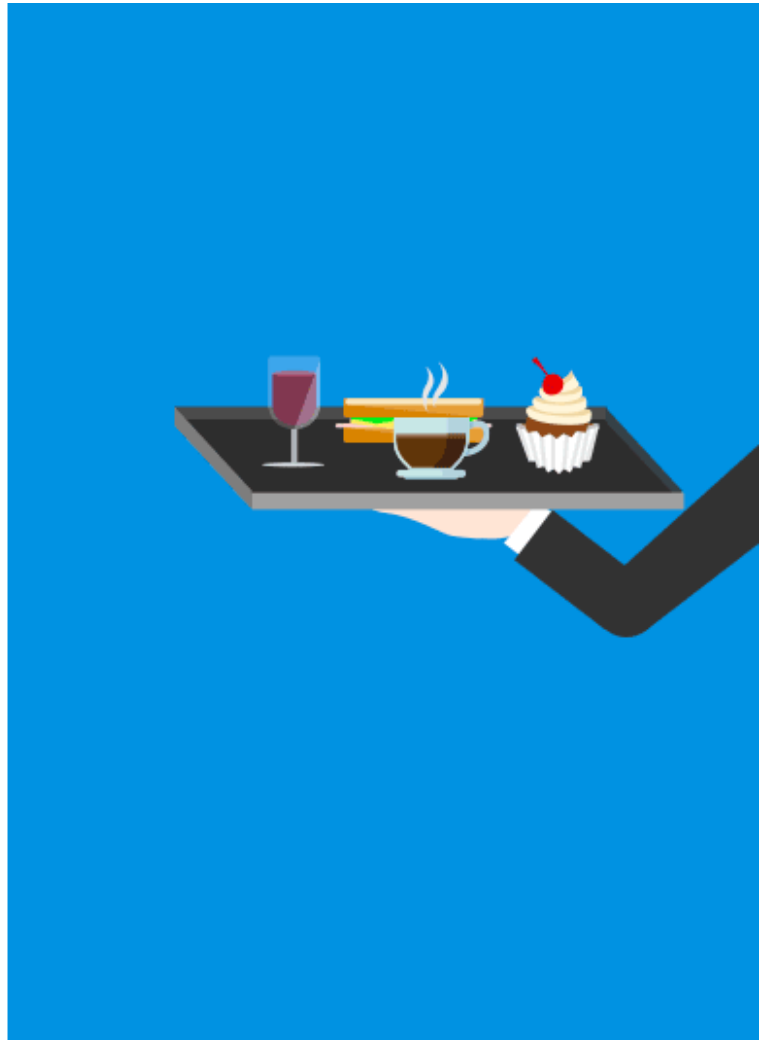
- No formal(machine-readable) API Contract
  - Writing/updating client library for different clients
- Streaming is difficult
- Need semantic versioning whenever the api contract needs to be changed.
- Bidirectional streaming not possible until you use pub/sub
- Uses HTTP/1
- Text base protocol(JSON)

HTTP 1 vs HTTP 2

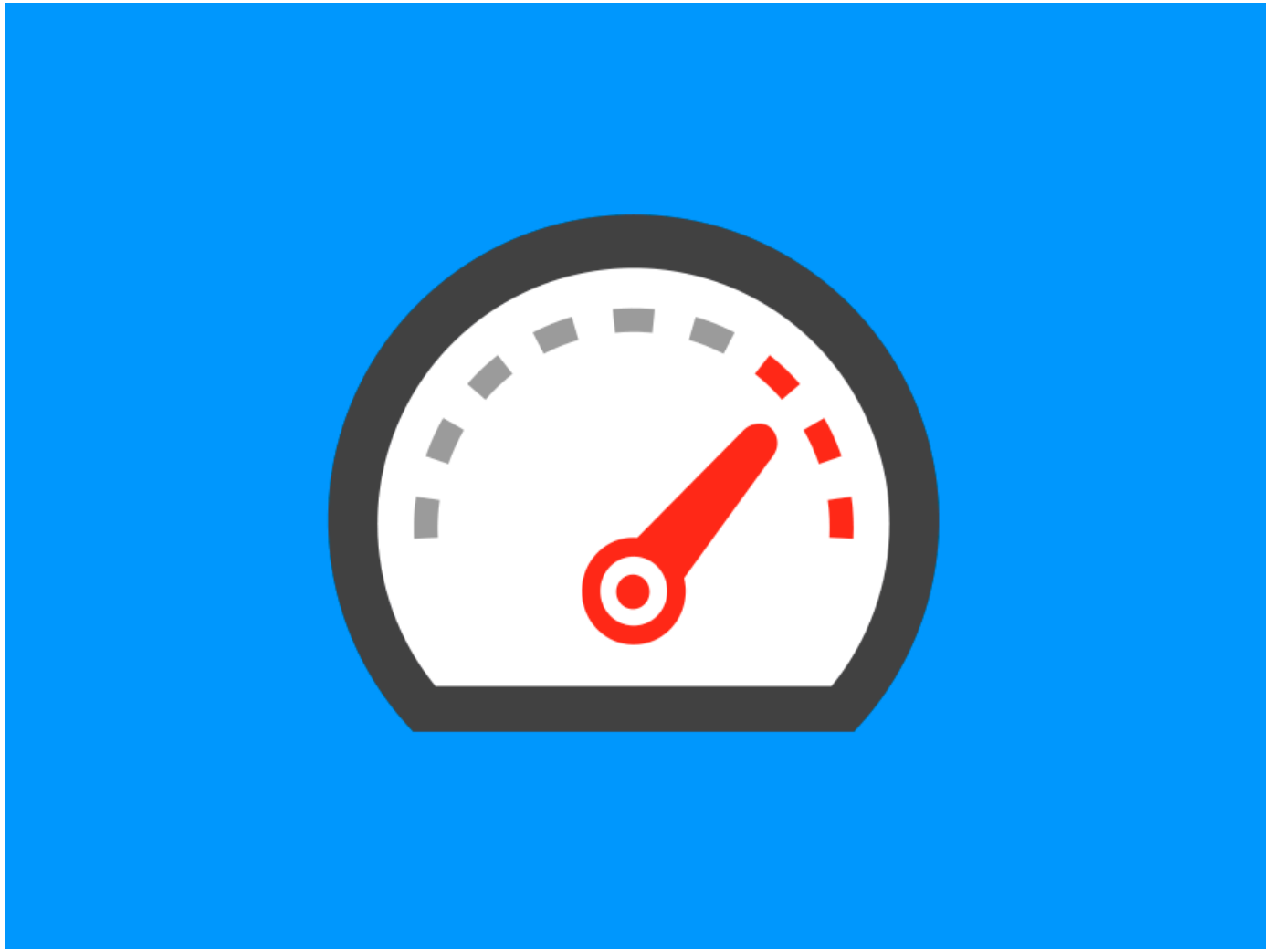
1. Multiplexing

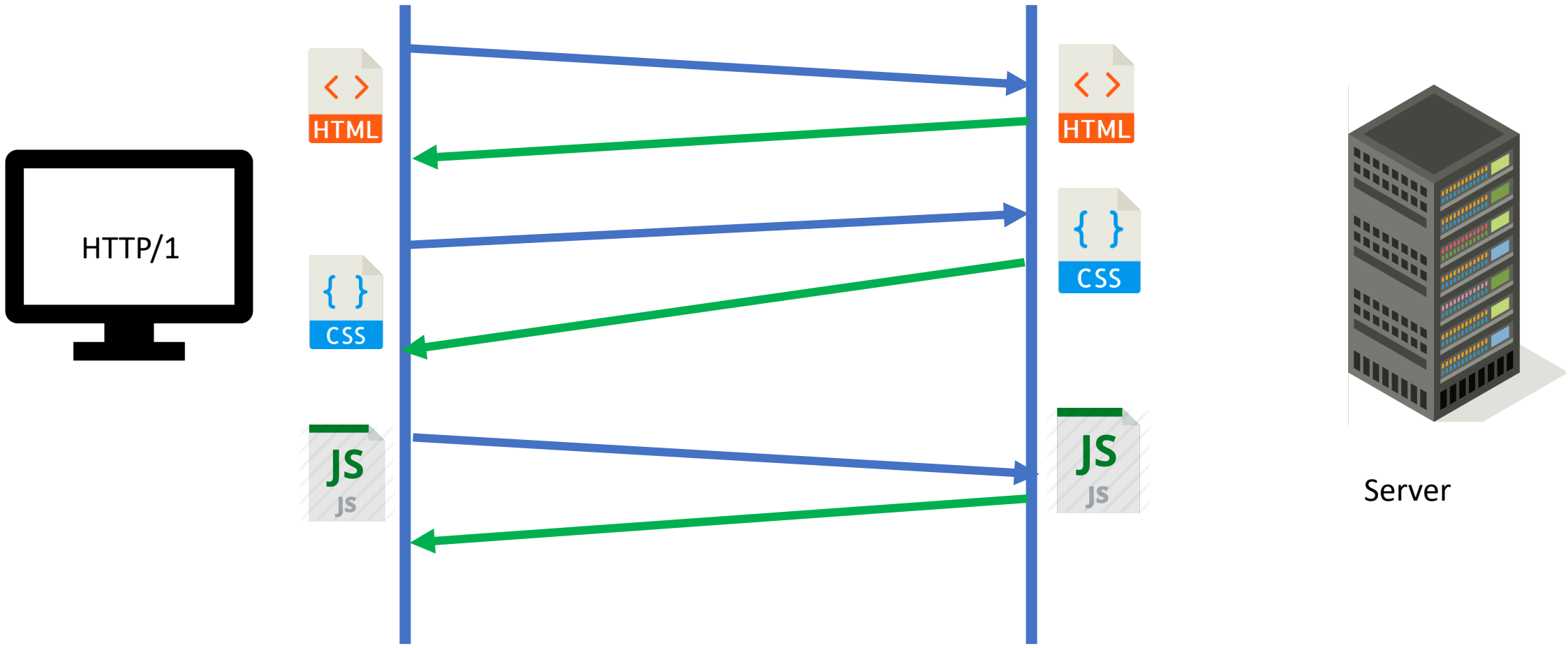


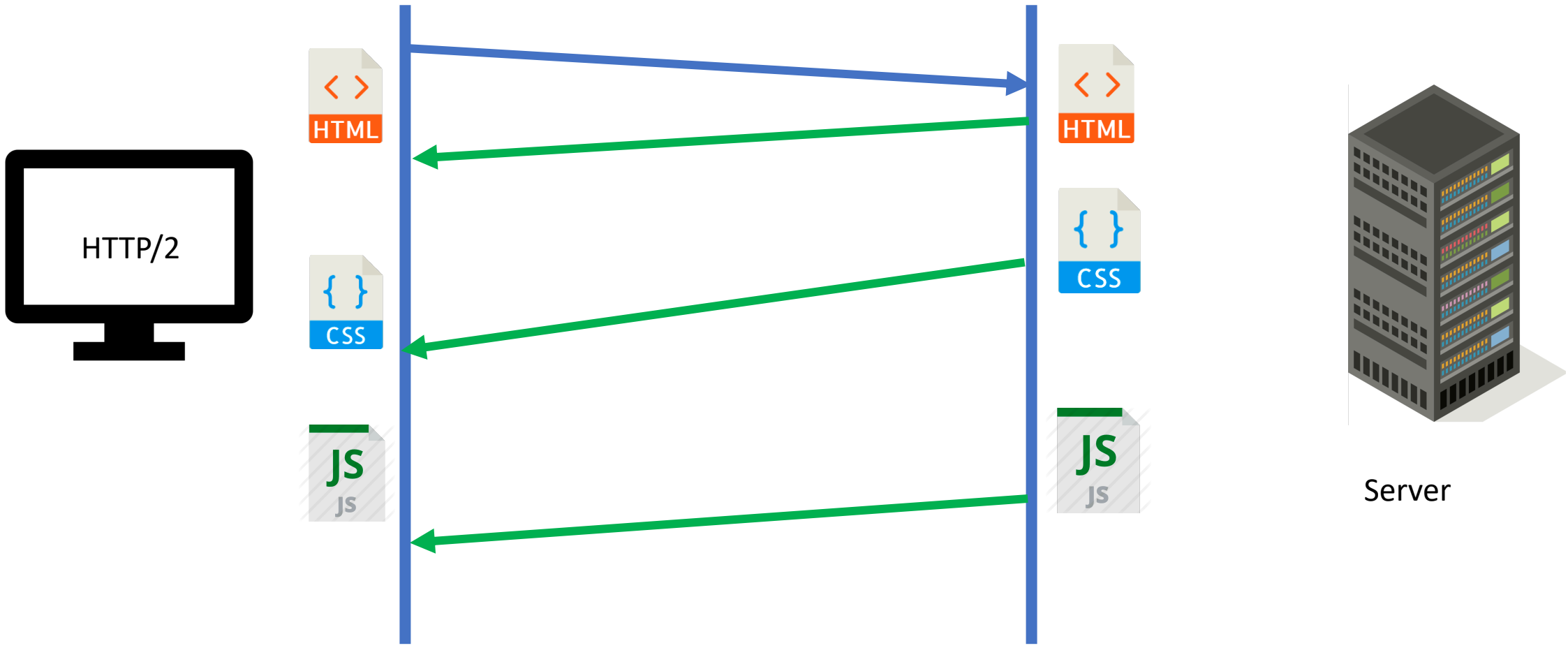



HTTP/1.1 Baseline
1
Open Connection
2
GET /index.html
3
RESPONSE
4
GET /styles.css
5
RESPONSE
6
GET /scripts.js
7
RESPONSE
8
Client Renders Page
Connection Closed
9
Time
HTTP/2 Multiplexing
1
Open Connection
2
GET /index.html
3
RESPONSE
4
GET /styles.css
GET /scripts.js
5
RESPONSE
RESPONSE
6
Client Renders Page
Connection Remains Open
7

1. Server Push



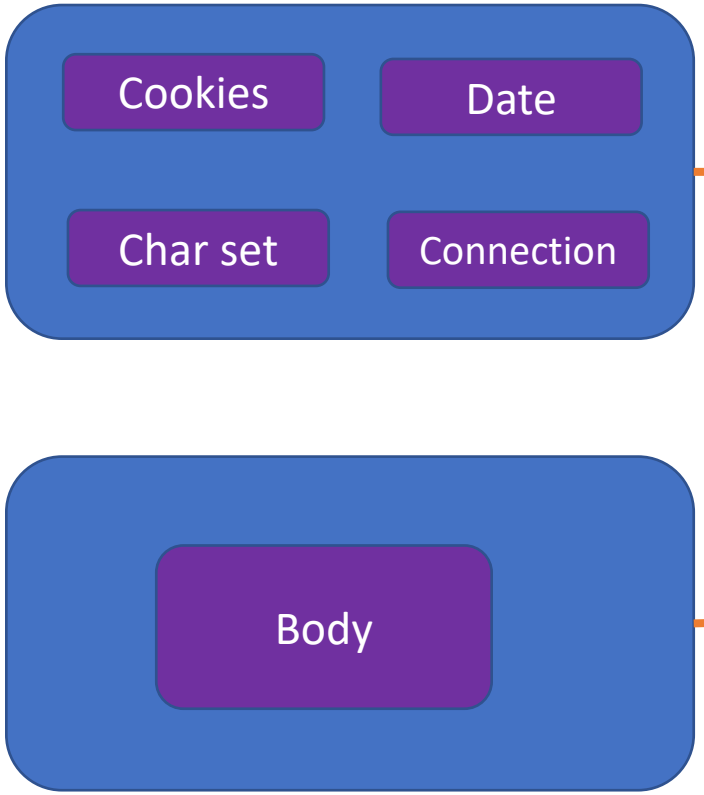

HTTP/1
HTML
CSS
JS
JS
HTML
CSS
JS
Server


HTTP/2
HTML
CSS
JS
JS
HTML
CSS
JS
Server

2.Header compression

HTTP Message

- Header



Example	
Accept-Encoding:	gzip, deflate
Cookie	ATHENATEXT_BROWSER_UUID=uuid:b96b8c3e-ab99-4f46-b62d-0505f60871ba;
Date	Wed, 16 Oct 2019 06:33:00 GMT
Connection	keep-alive
Host	Devcon.com
Body	<html><body> Hello, World <body></html>

HPACK COMPRESSION

HTTP HEADER	
:method	Get
:scheme	https
:host	Devcon.com
:user-agent	Mozilla/5.0

59 character

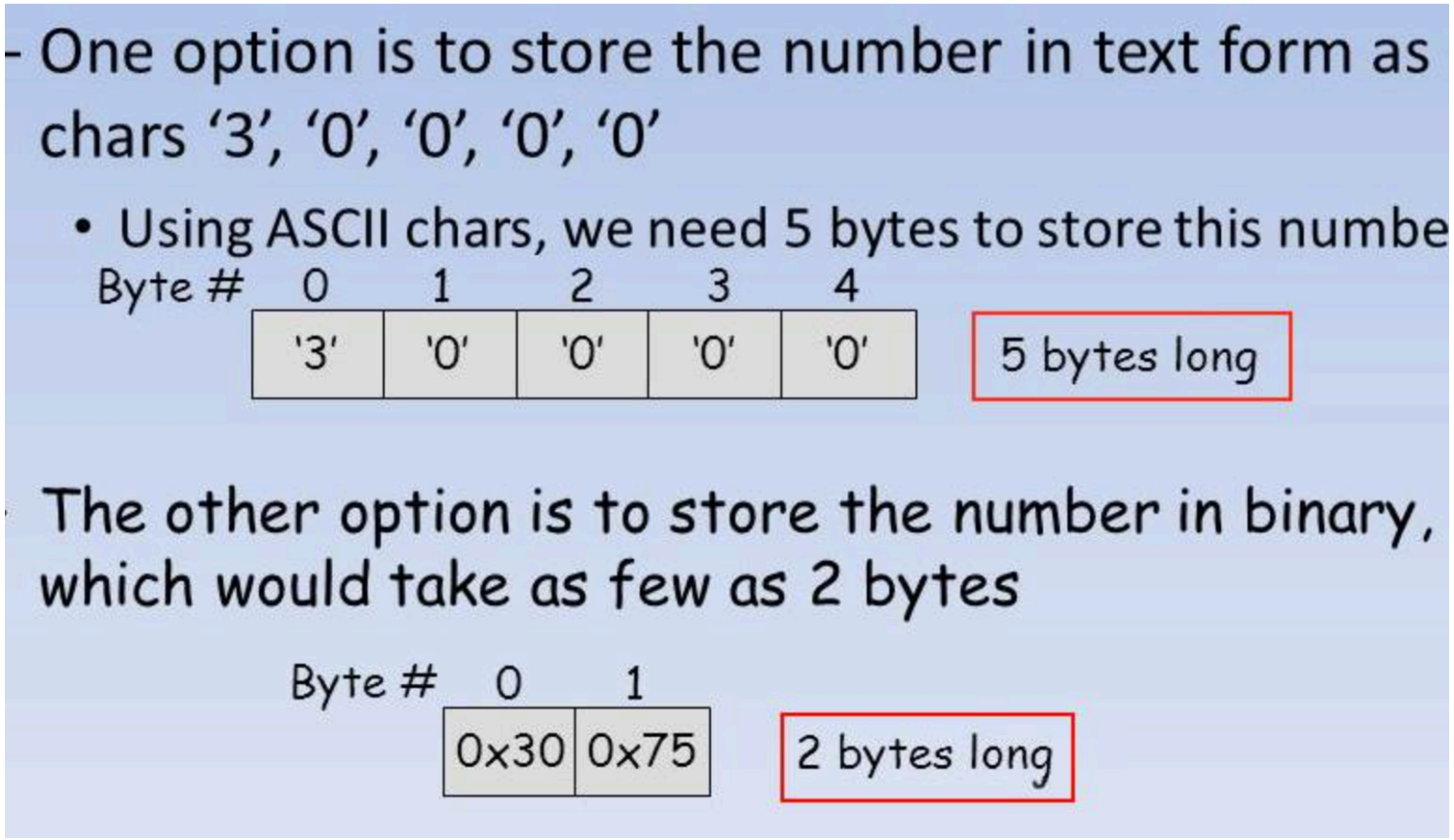
STATIC TABLE		
1	:method	Get
2	:scheme	https
3	…….	…….
4	…….	…..

DYNAMIC TABLE		
50	:host	Devcon.com
51	:user-agent	Mozilla/5.0
…..		
……		

ENCODED HEADER
1,2,50,51 Huffman(“www.Devcon.com”)

6 character

3.Binary protocol



Binary protocol

- HTTP/1 human readable frames
- HTTP/2 binary protocol , Not suitable for humans only for machines

HTTP/2 vs HTTP/1 demo

- http://www.http2demo.io/

Protobuff vs JSON

JSON
{"status":"OK","message":"Hello JSON!"}

JSON object length	Information length	Non-information length
39	13	26(wastage)

Serialized msg:

{"status":"OK","message":"Hello JSON!"}

PROTO BUFF

```
message ProtoVsJson{
    string status = 1;
    string message  = 2;
}
```

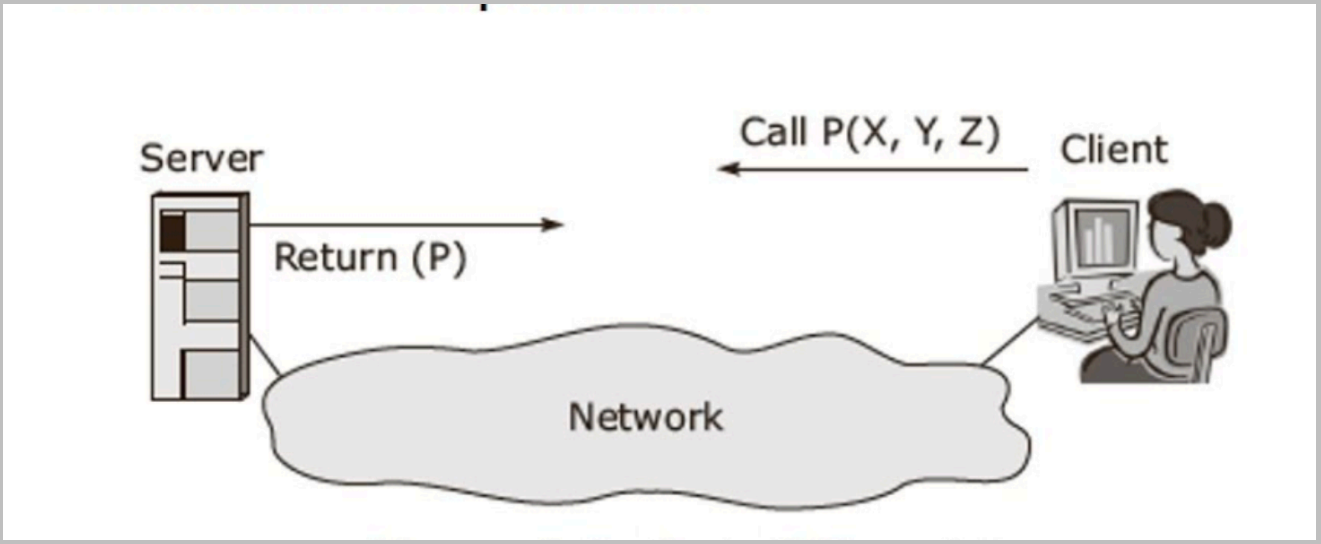
Serialized data length	Information length	Non-information length
17	13	4(wastage)

Serialized msg:

{
1: OK
2: Hello JSON!
}

```
10 2 79 75 18 11 72 101 108 108 111 32 74 83 79 78 33
    0  K       H   e   l   l   o    J  S  O  N  !
```

RPC



1. client uses stub to call the method on server side
2. Stub does packing and unpacking of input/output data

Server

```
class xyz{
 int P(int X,int Y,int Z);
}
```

Client

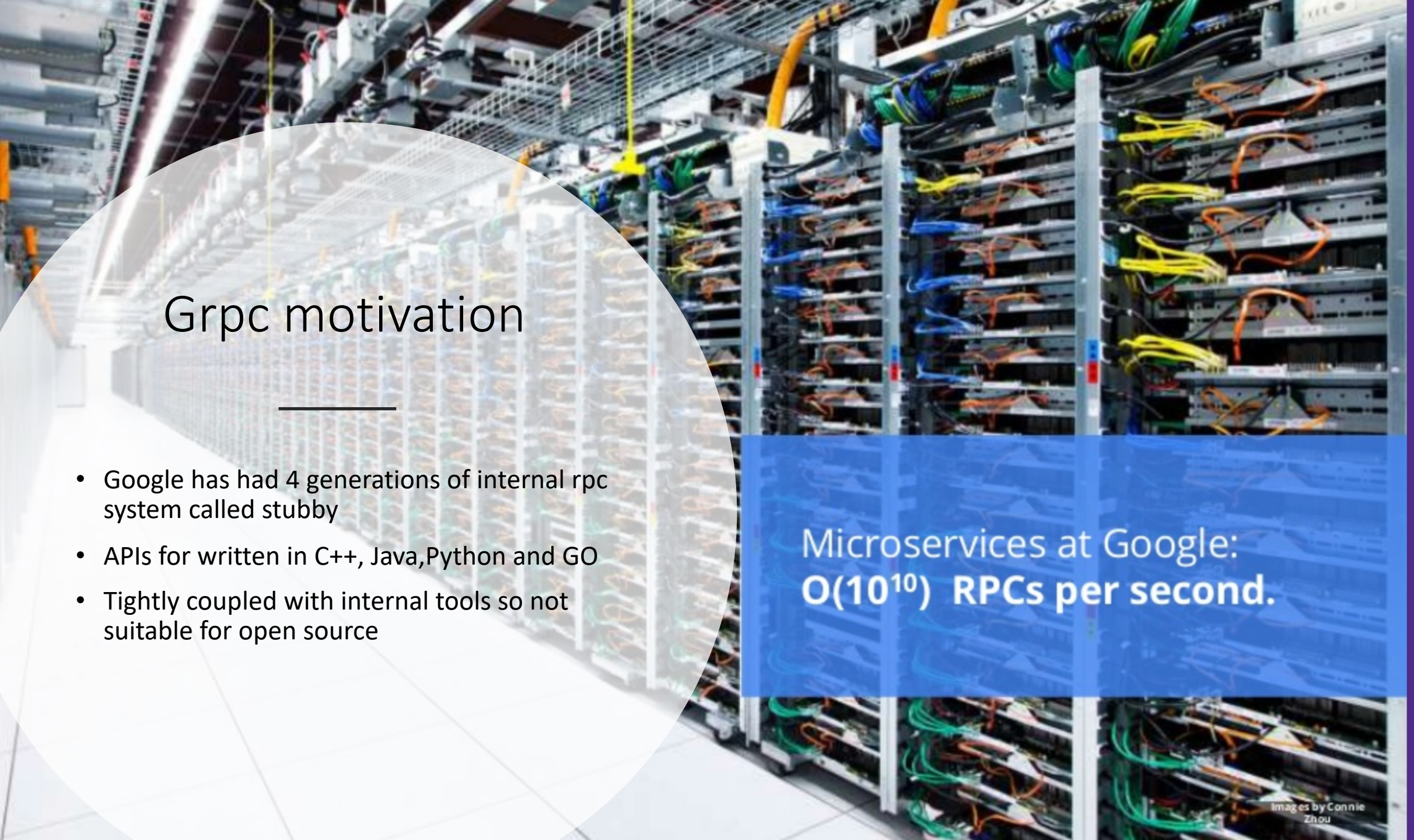
```
RemoteObjStub ro = new RemoteObjStub("destinationAdd")
ro.P(X,Y,Z);
```

http://grpc.io

Open source on Github for C, C++, Java, Node.js, Python, Ruby, Go, C#, PHP, Objective-C

Grpc motivation

- Google has had 4 generations of internal rpc system called stubby
- APIs for written in C++, Java,Python and GO
- Tightly coupled with internal tools so not suitable for open source



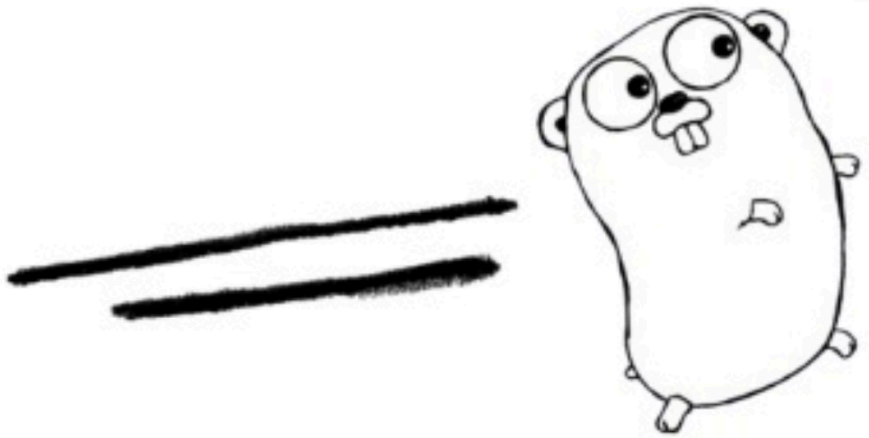
grpc

- Binary protocol(HTTP/2)
- Multiplexing many requests on one connection (HTTP/2)
- Strongly typed service and message definition (Protobuf for serialization)
- client/server library implementations in 10+ languages
- Monitoring ,auth, load balancing

Multiple Languages

C/C++

python™



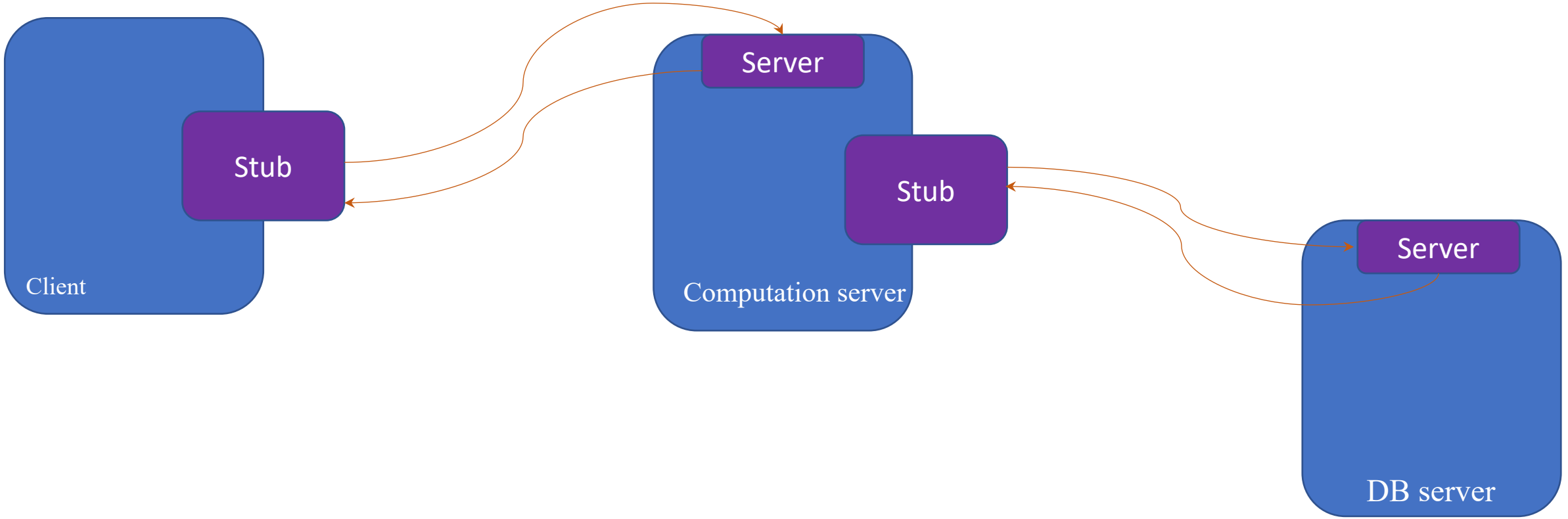
Java™

php

C#

node.js™

Calling Microservices with grpc



Stub: Service definition and IDL(POJO)

Protobuff

```
syntax = "proto3";

option java_package = "com.employee.proto";

service Employee{

    rpc addEmployee(EmployeeRequest) returns (EmployeeResponse);
    rpc getEmployees(EmployeeRequest) returns (EmployeesList);

}

enum EmployeeDepartment{
    ADMIN = 0;
    DEVELOPMENT = 1;
    QA =2;
    TRANSPORT = 3;
}
message EmployeeRequest{
    string employeeName = 1;
    int32 employeeId = 2;
    EmployeeDepartment employeeDepartment = 3;
    repeated string languages = 4;
    string address = 5;
}

message EmployeeResponse{
    string message = 1;
    int32 statusCode = 2;
    EmployeeRequest employee = 3;

}

message EmployeesList{
    repeated EmployeeRequest employeeRequest = 1;

}
```

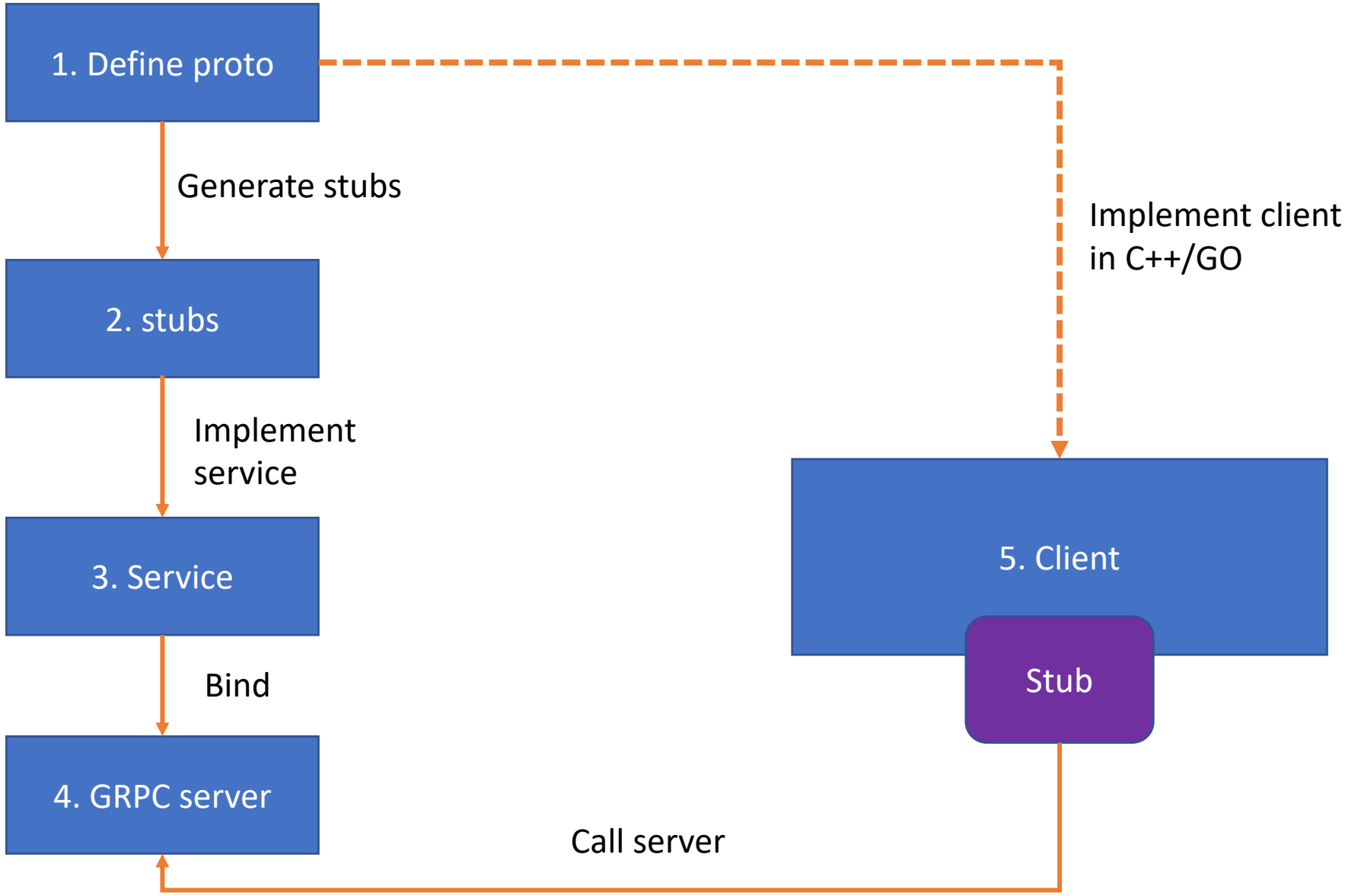
Service

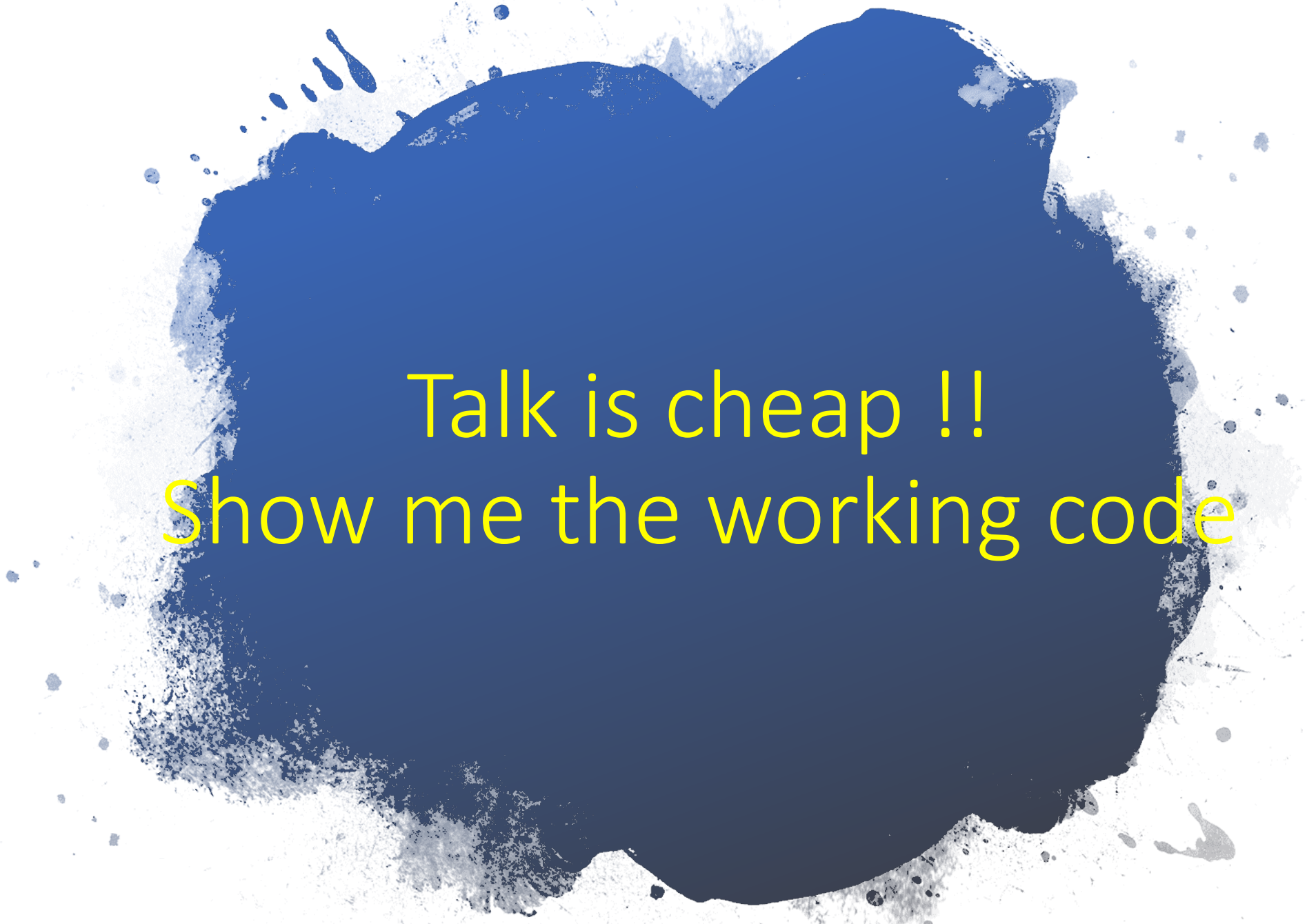
```
static Map<Integer,EmployeeRequest> map = new HashMap<>();

@Override
public void addEmployee(EmployeeRequest request, StreamObserver<EmployeeResponse> responseObserver) {
    map.put(request.getEmployeeId(), request);
    EmployeeResponse.Builder response = EmployeeResponse.newBuilder();
    response.setMessage("Successfully added employee with id: "+request.getEmployeeId()+"Now the map size is :"+map.size());
    response.setStatusCode(201);
    response.setEmployee(request);
    responseObserver.onNext(response.build());
    responseObserver.onCompleted();
}

@Override
public void getEmployees(EmployeeRequest request, StreamObserver<EmployeesList> responseObserver) {
    EmployeesList.Builder employeeList = EmployeesList.newBuilder();
    Set<Integer> keys = map.keySet();
    int i = 0;
    for(Integer key:keys) {
        EmployeeRequest employee = map.get(key);
        employeeList.addEmployeeRequest(i++, employee);
    }
    responseObserver.onNext(employeeList.build());
    responseObserver.onCompleted();
}
```

Client

```
public static void main(String args[]) {
    ManagedChannel channel  =  ManagedChannelBuilder.forAddress("localhost", 9094).usePlaintext().build();
    EmployeeBlockingStub blockingStub = EmployeeGrpc.newBlockingStub(channel);
    List<String> languages = new ArrayList<>();
    languages.add("JAVA");
    languages.add("GO");
    languages.add("JS");
    EmployeeRequest request = EmployeeRequest.newBuilder()
            .setEmployeeId(1)
            .setEmployeeDepartment(EmployeeDepartment.ADMIN)
            .setEmployeeName("Ramveer")
            .addAllLanguages(languages)
            .setAddress("Test Address123")
            .build();
    EmployeeResponse response = blockingStub.addEmployee(request);
    System.out.println("Response...."+response);
}
```

grpc flow


1. Define proto
Generate stubs
2. stubs
Implement
service
3. Service
Bind
4. GRPC server
Implement client
in C++/GO
5. Client
Stub
Call server


Talk is cheap !!
Show me the working code

drawbacks

- No browser support
- No curl support
- Not great documentation
- Standardization across languages

Grpc in production

- Netflix
- Cisco
- Juniper network
- Google
- spotify

Load Balancing

- https://github.com/grpc/grpc/blob/master/doc/load-balancing.md
- https://grpc.io/blog/loadbalancing/

- https://github.com/grpc/grpc/blob/master/doc/service_config.md