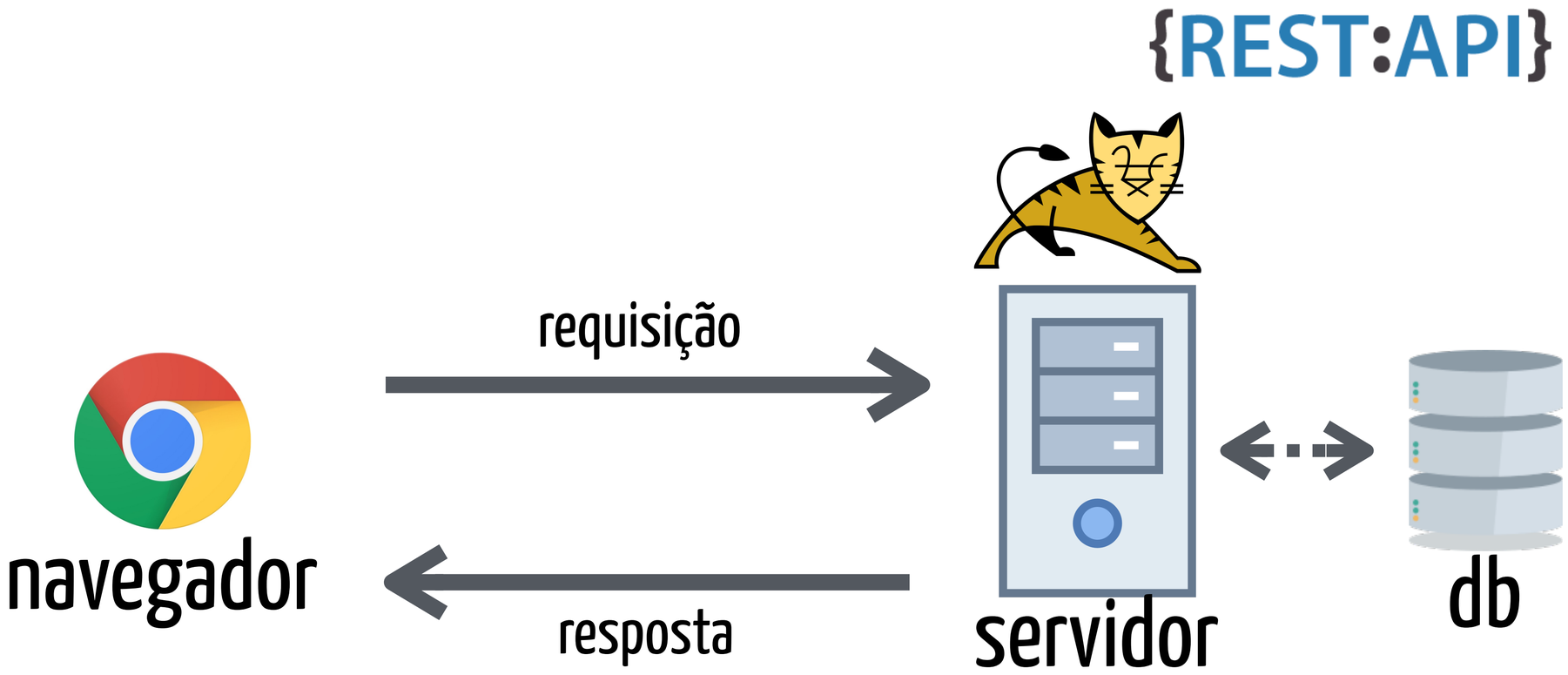
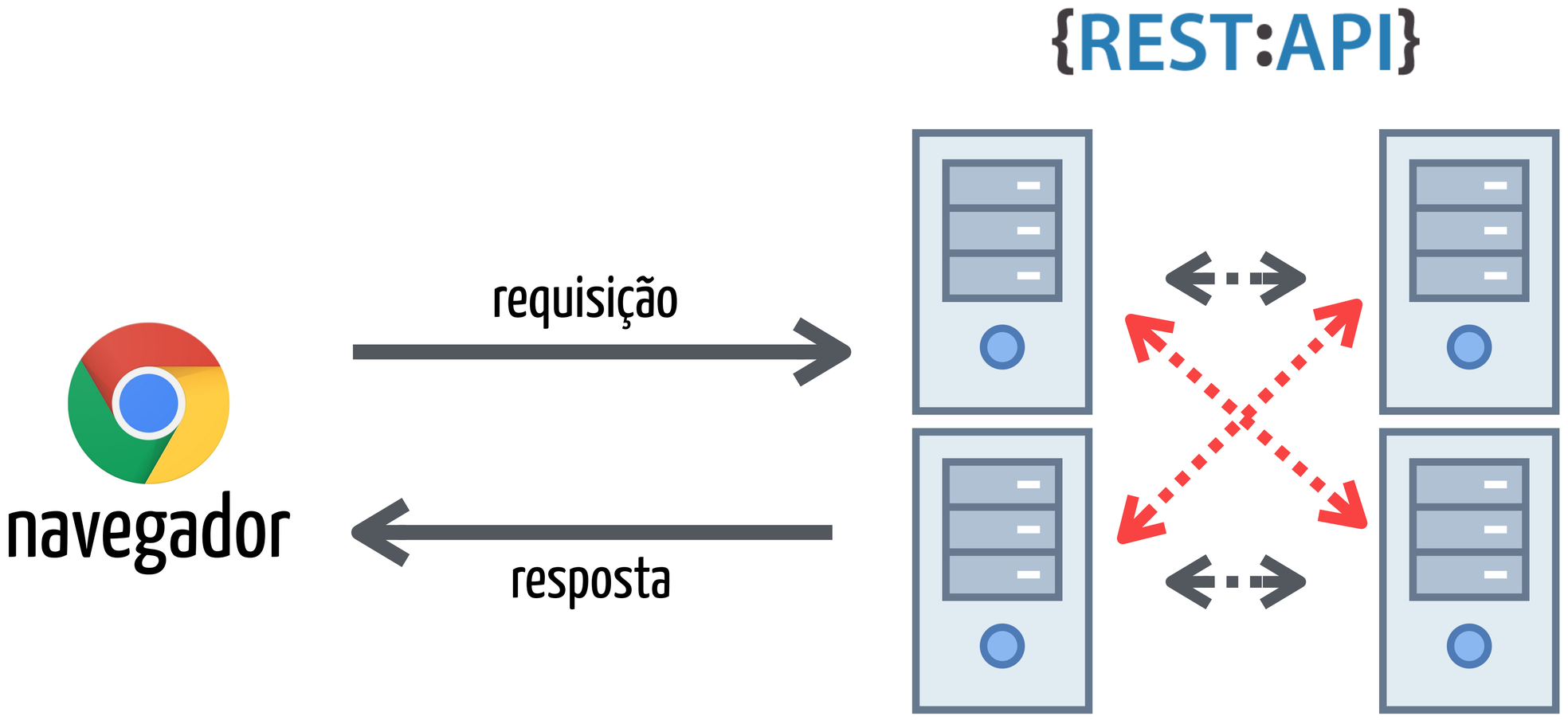
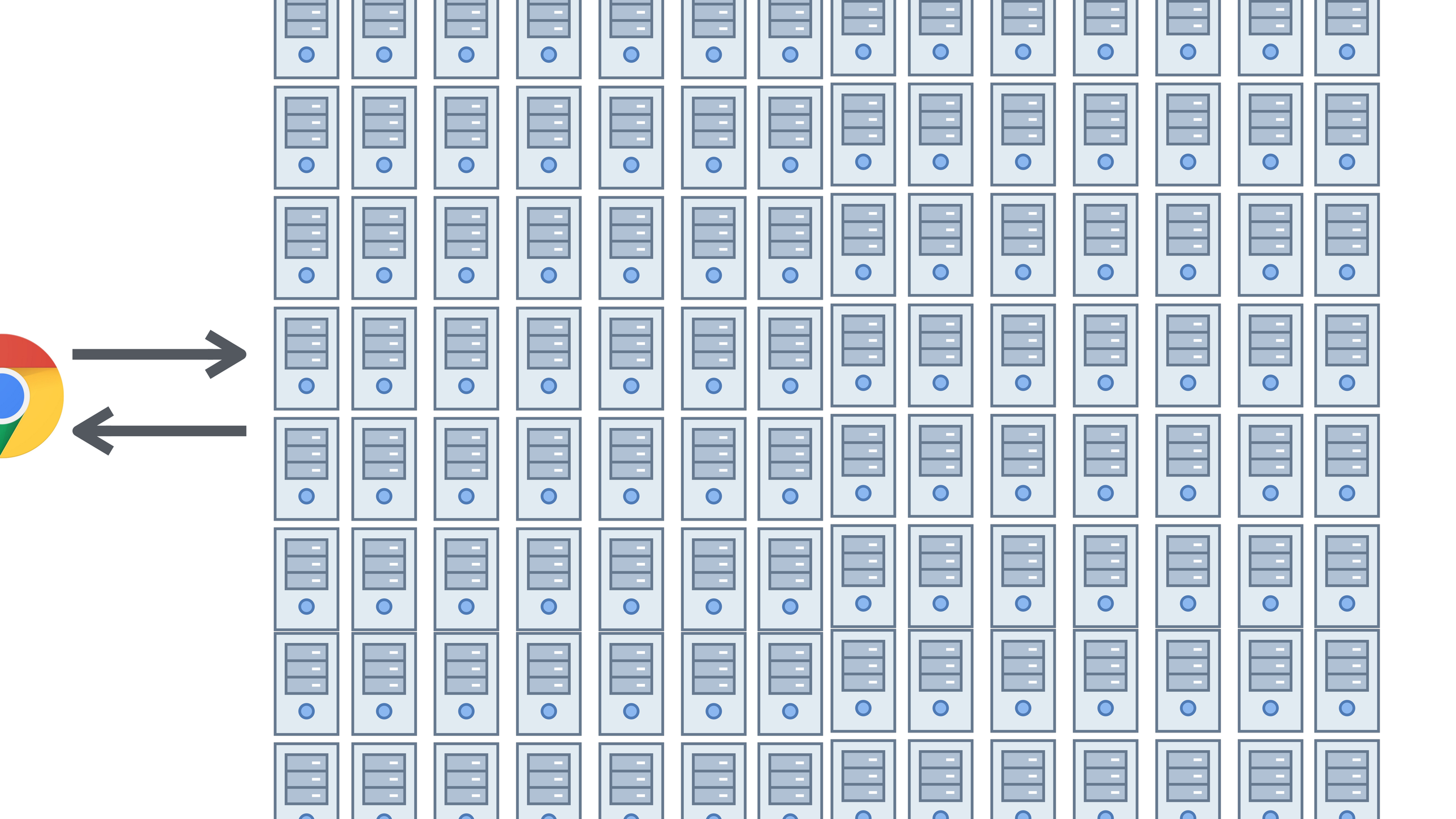
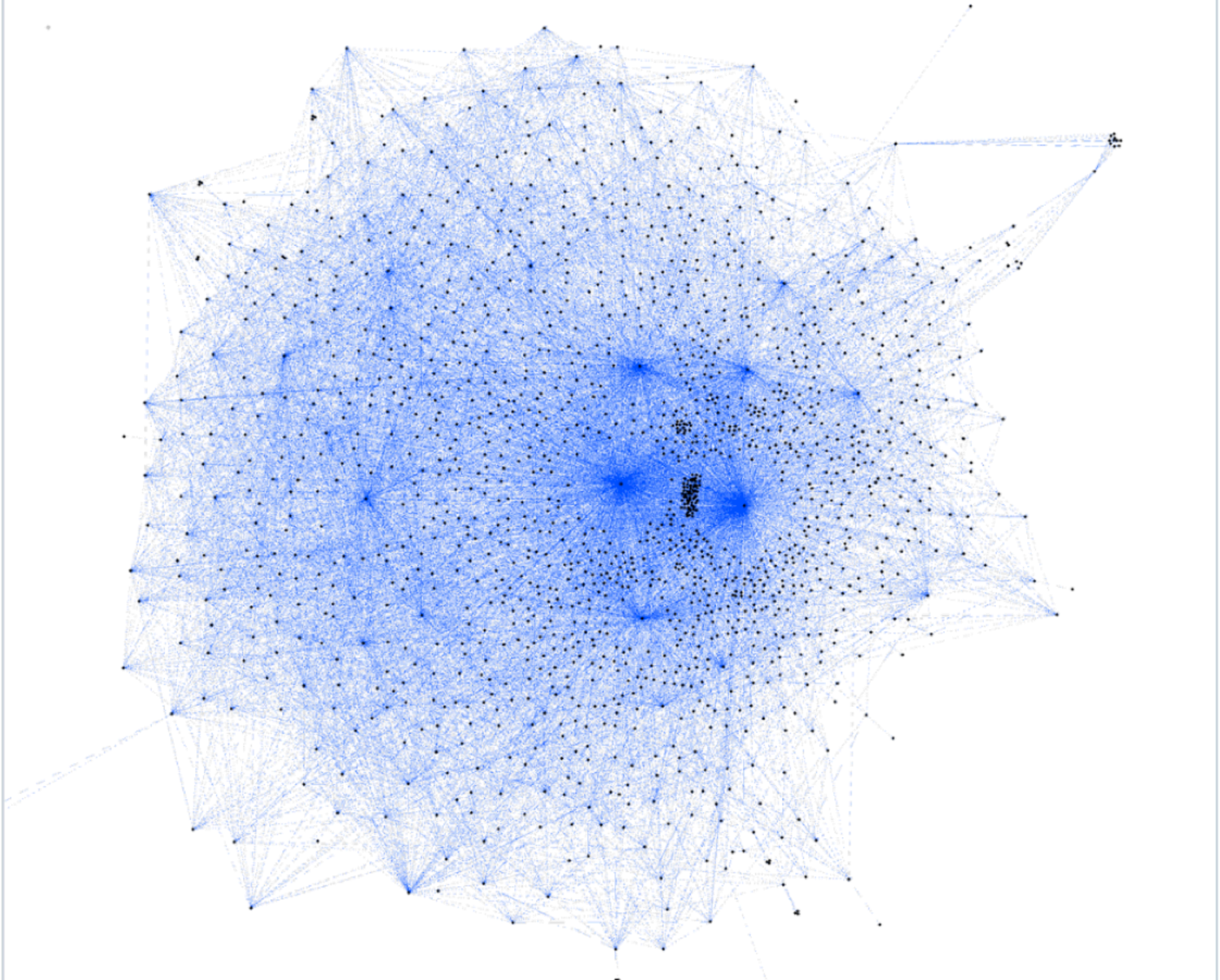
um framework moderno e performático para
comunicação entre microsserviços

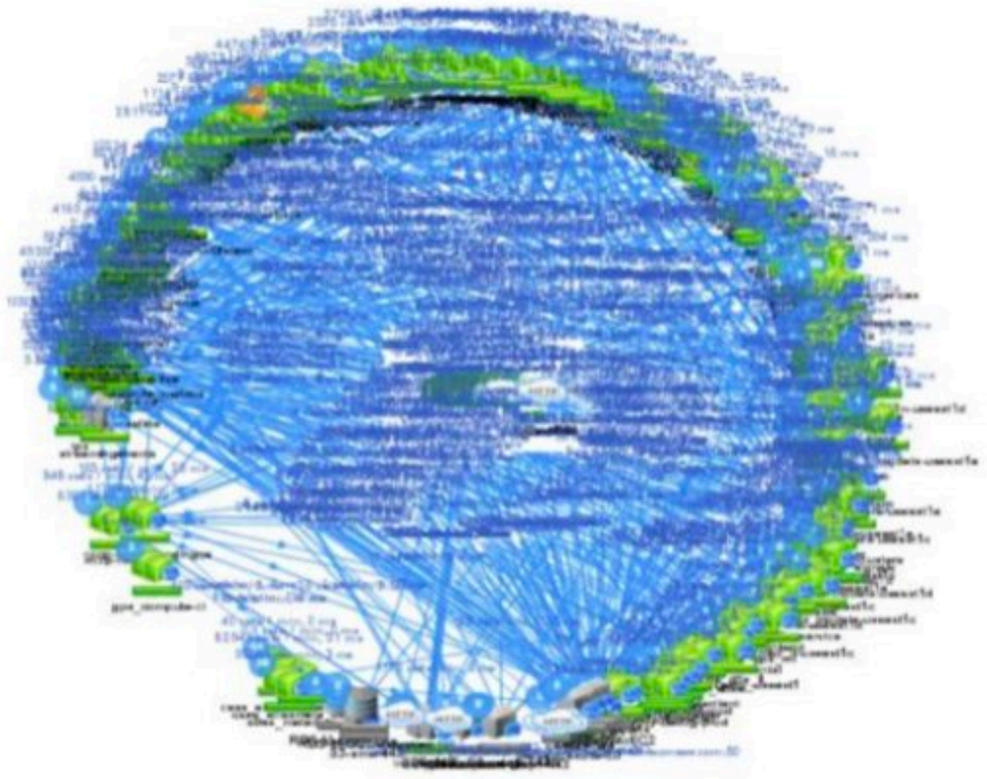

{REST:API}
requisição
resposta
navegador
servidor
db

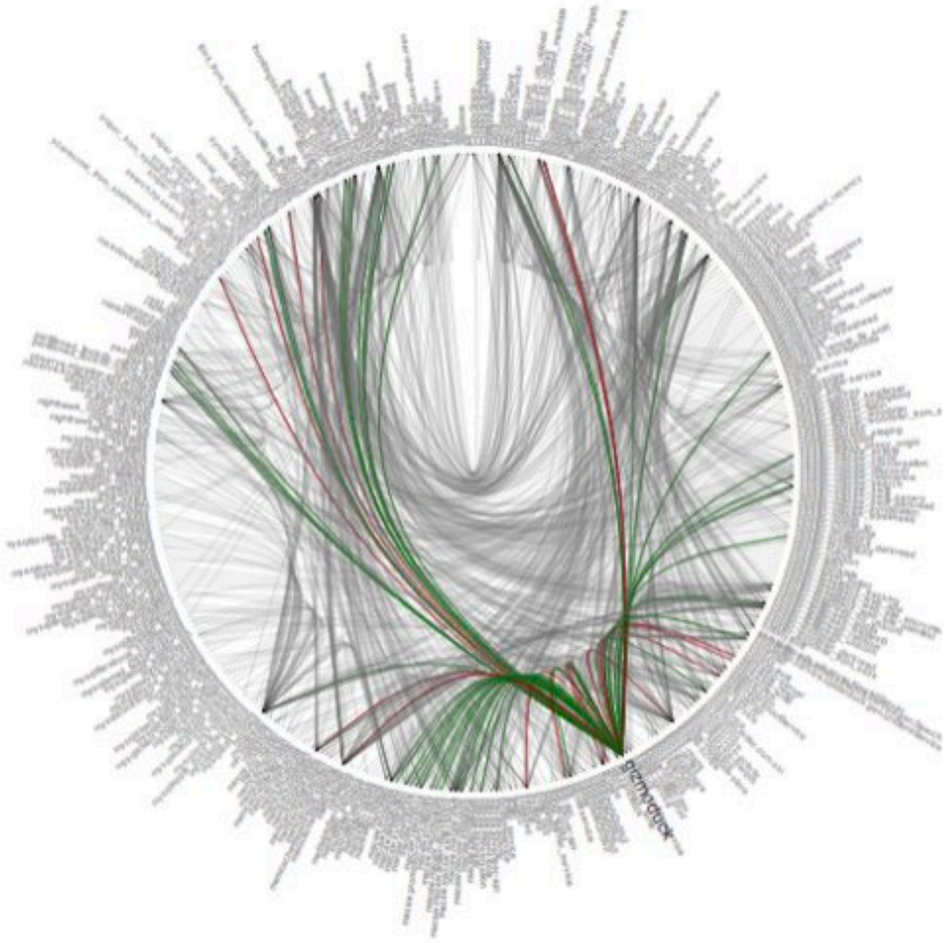

{REST:API}
requisição
navegador
resposta



Pode piorar...





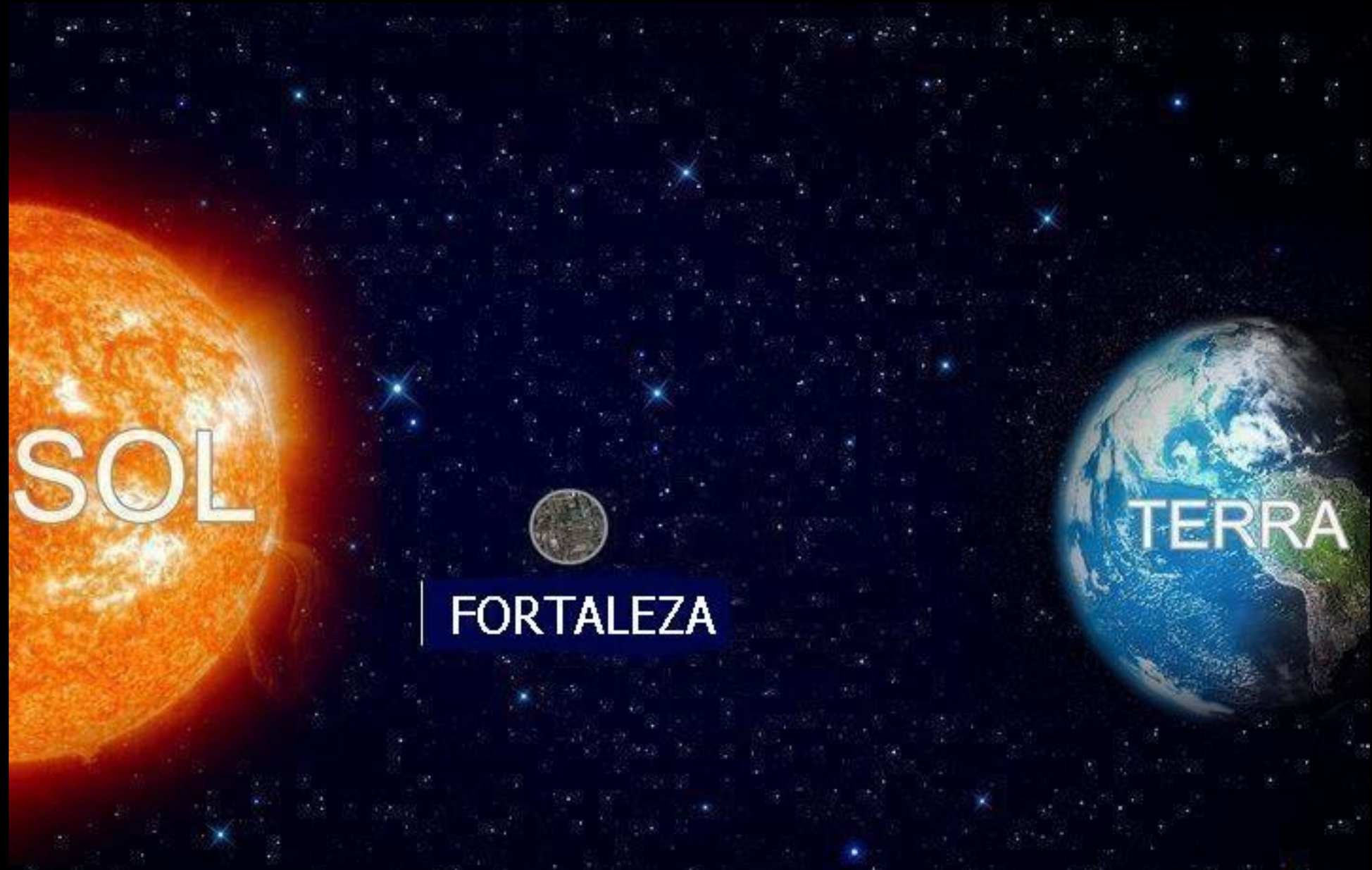


NETFLIX


GRPC


Rafael Ponte
@rponte


ZUP



Fortaleza - Terra do Sol

devs cansados

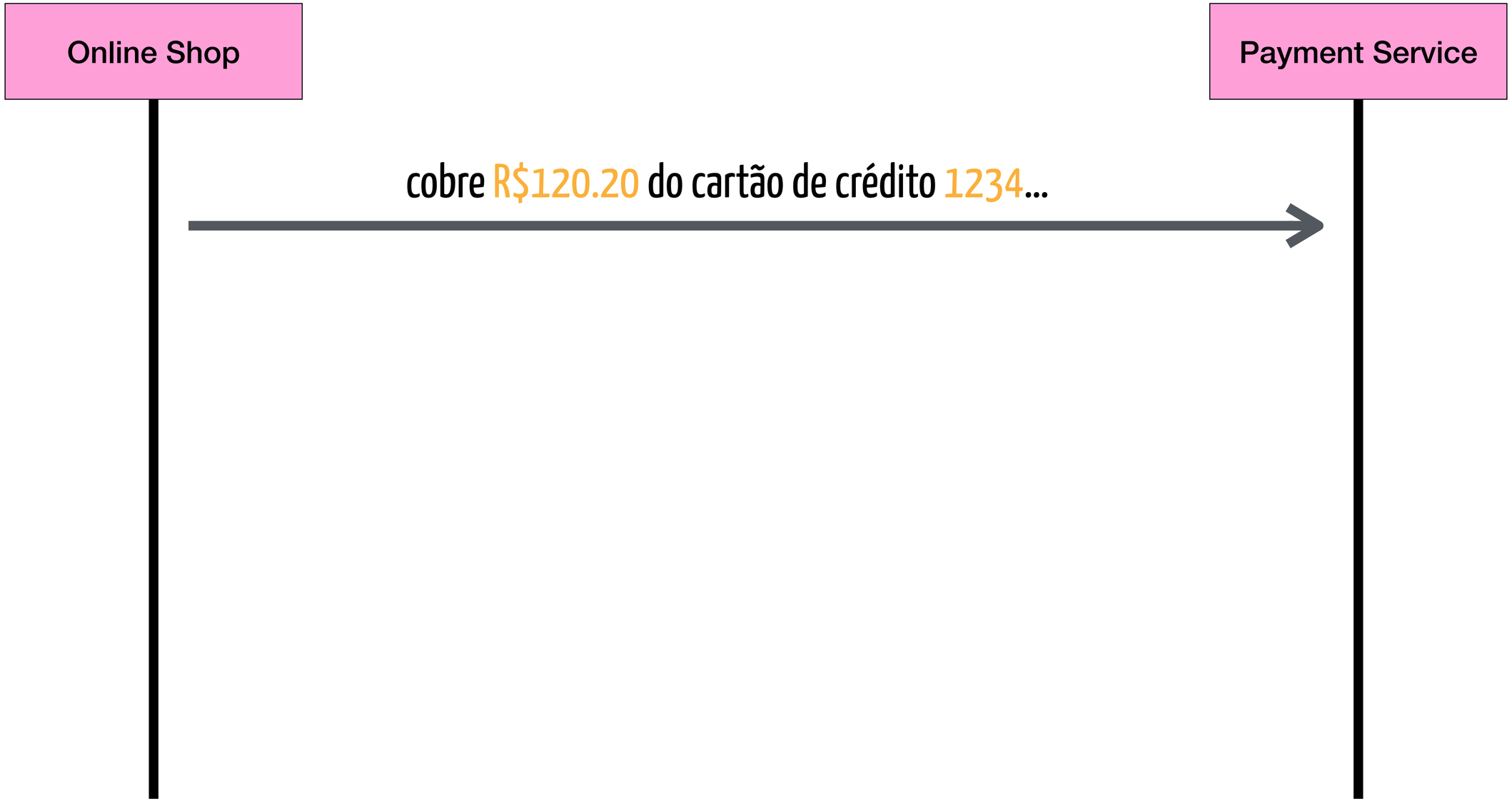
gRPC Remote Procedure Call framework

Remote Procedure Call

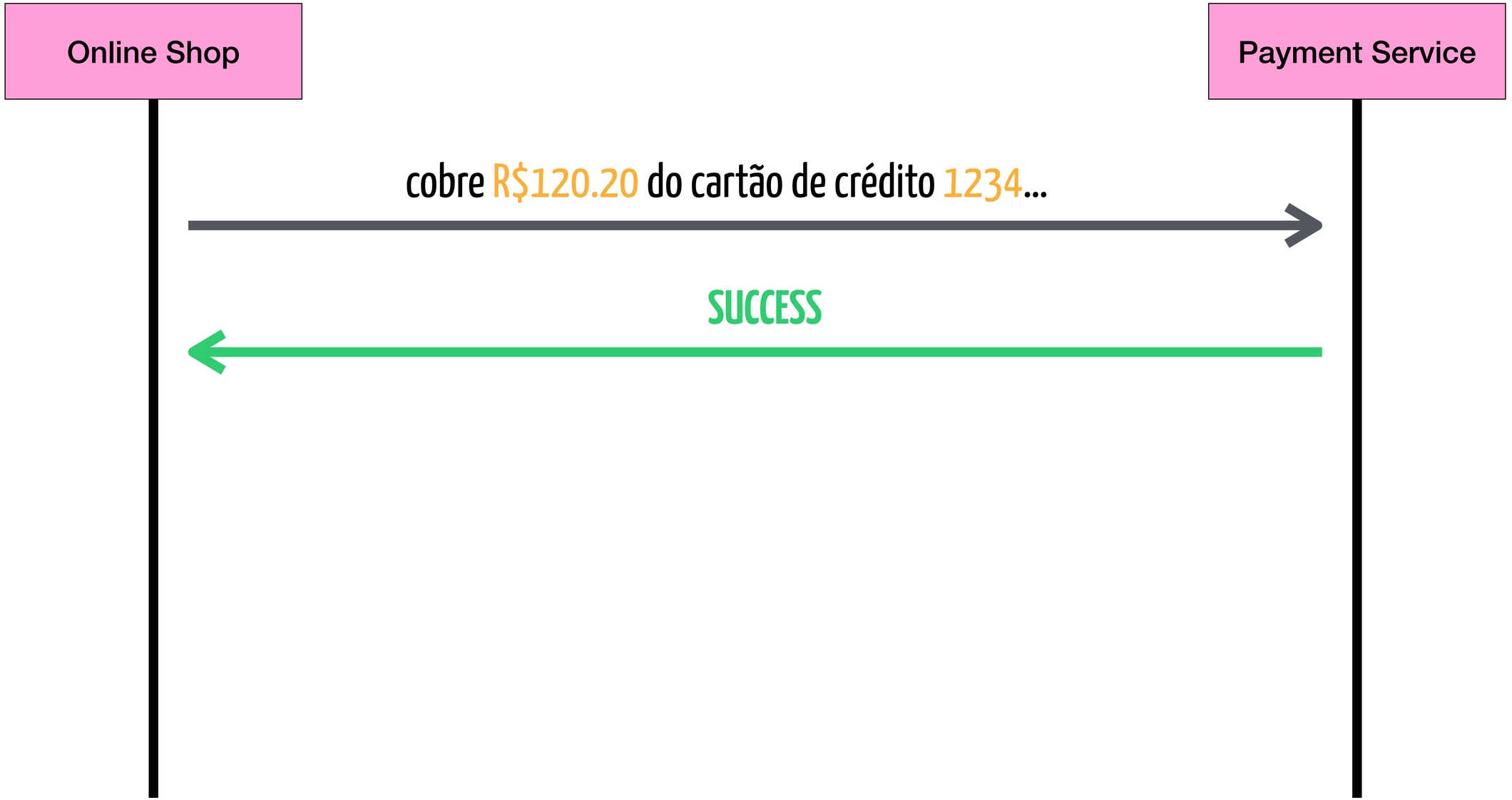
Loja Virtual

Pagamento Online

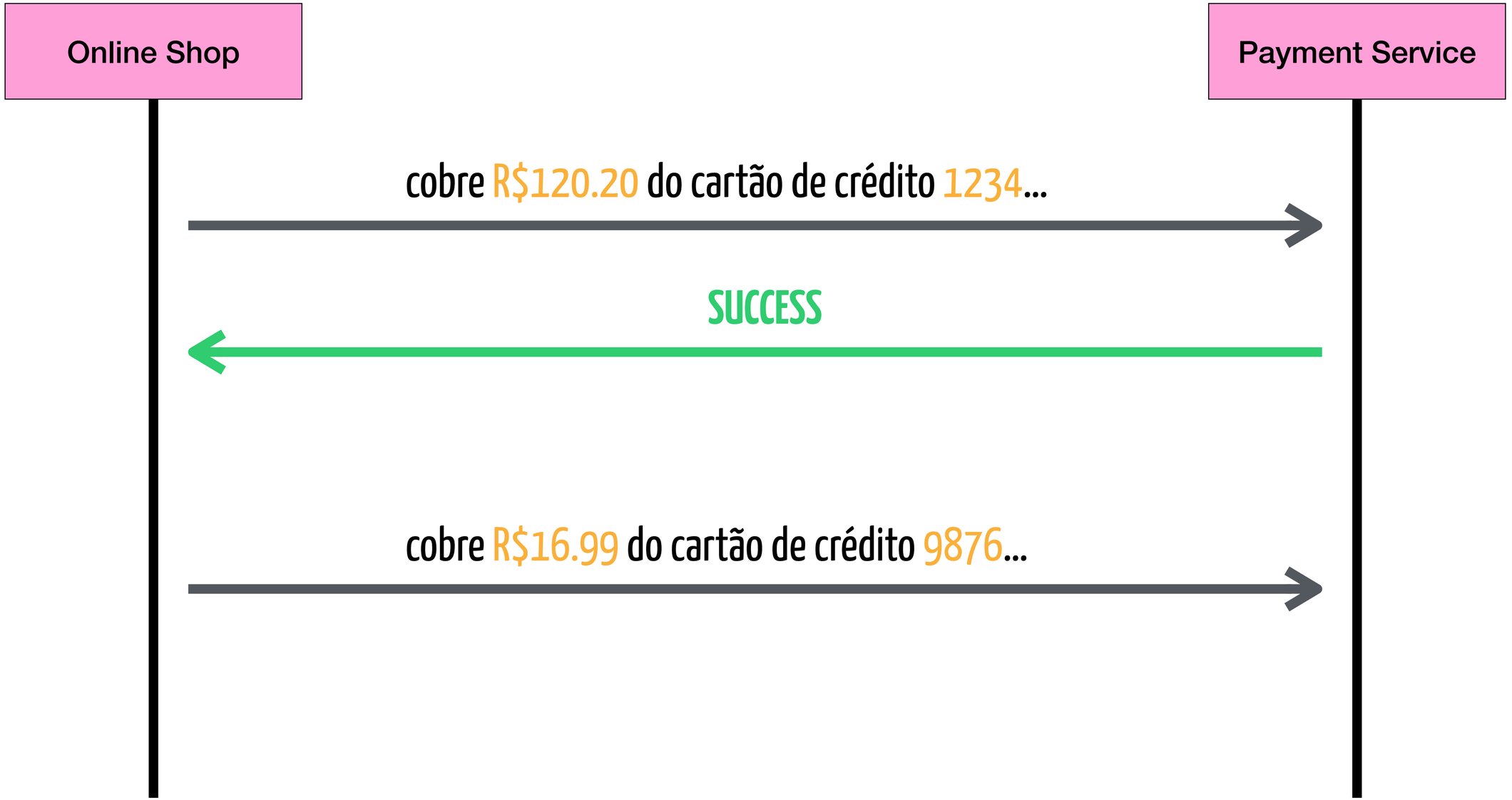
Pagamento Online



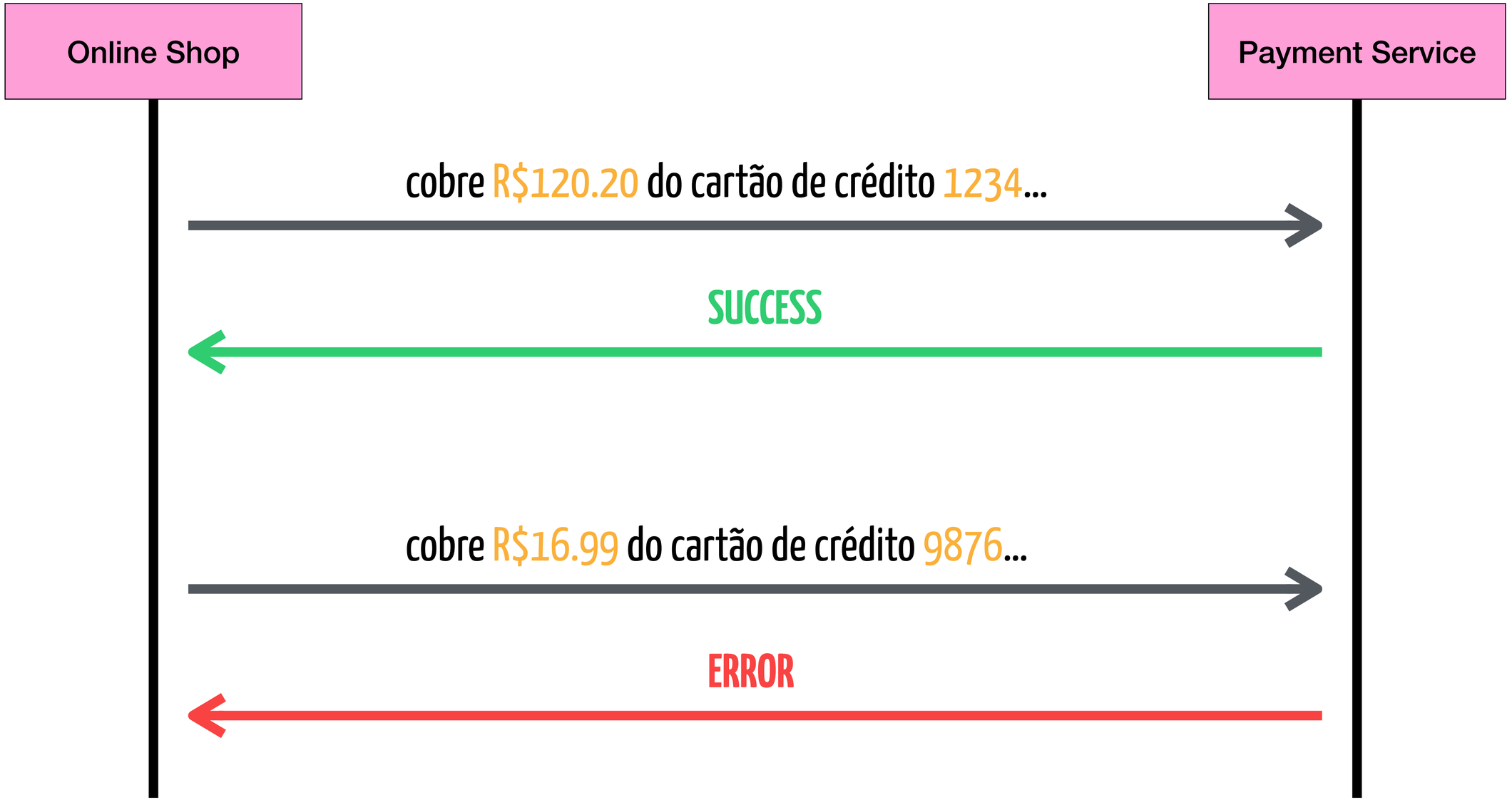
Pagamento Online



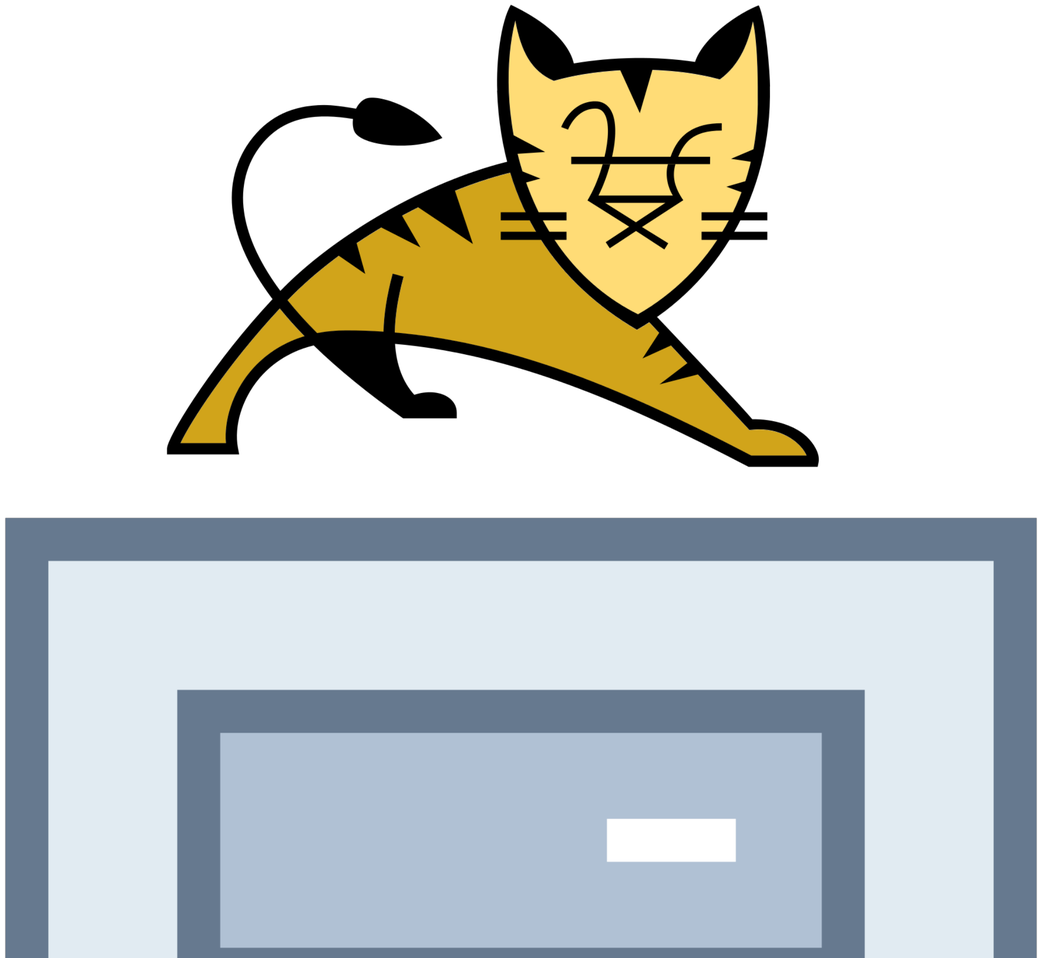
Pagamento Online

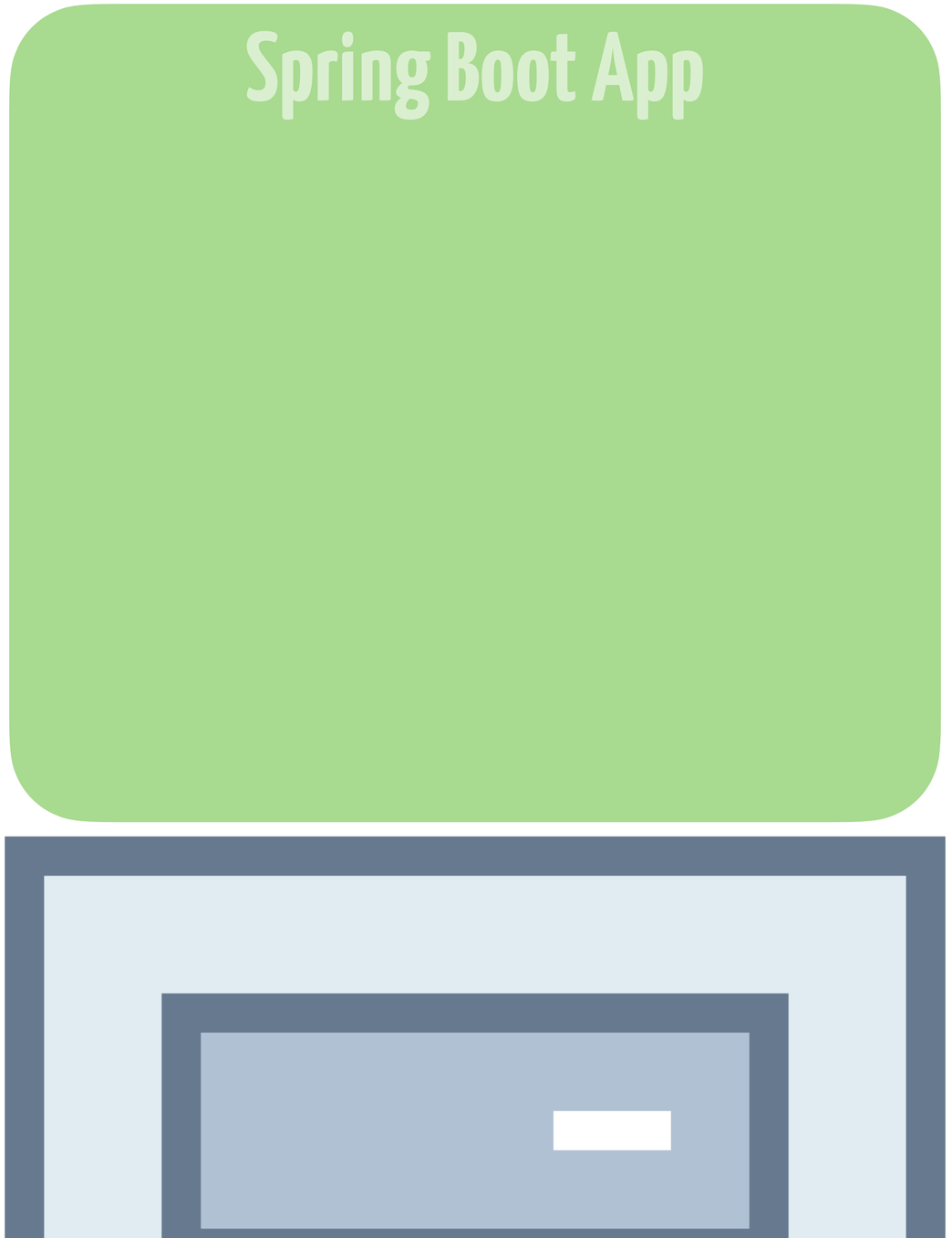


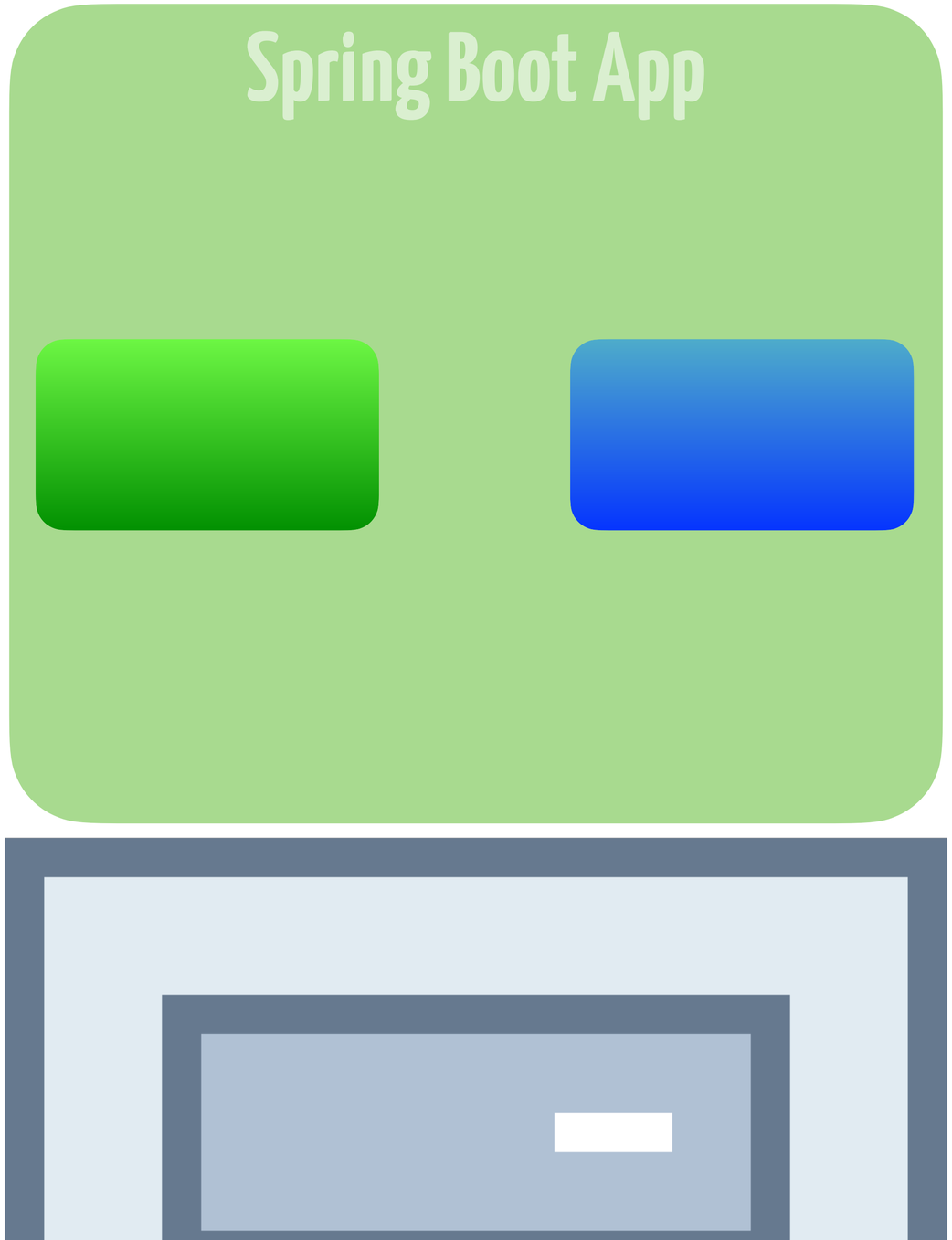
Pagamento Online

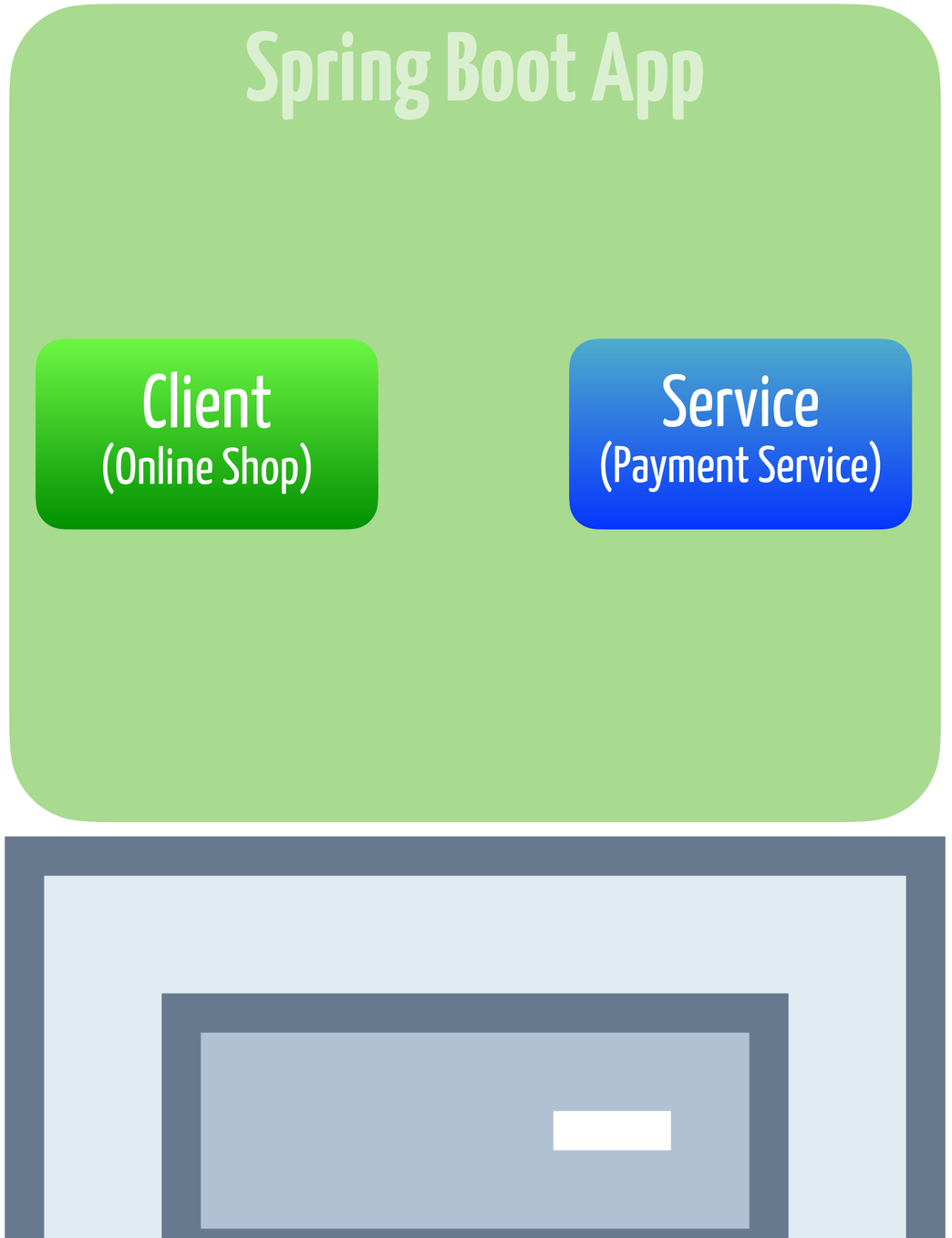


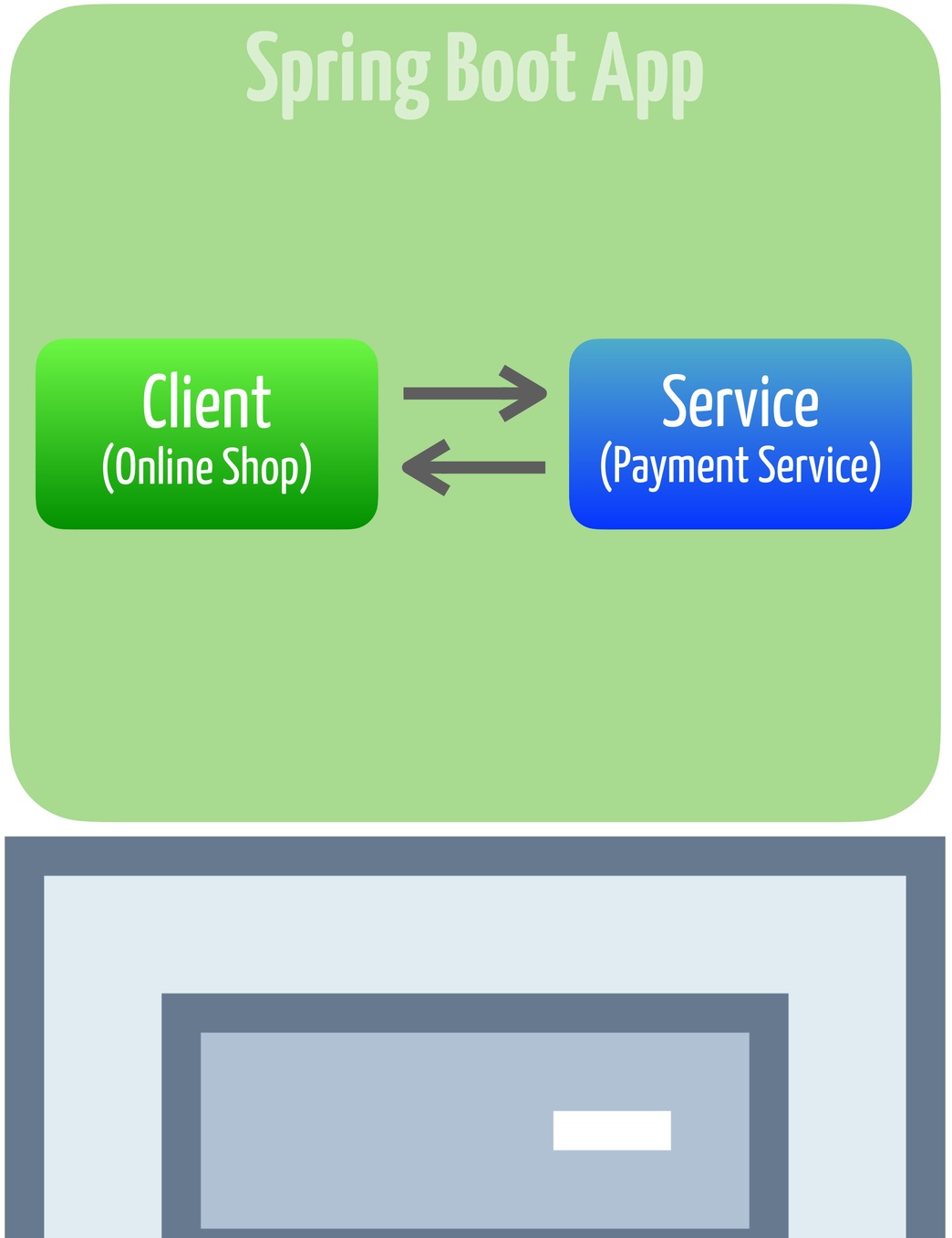
Como implementamos essa lógica?



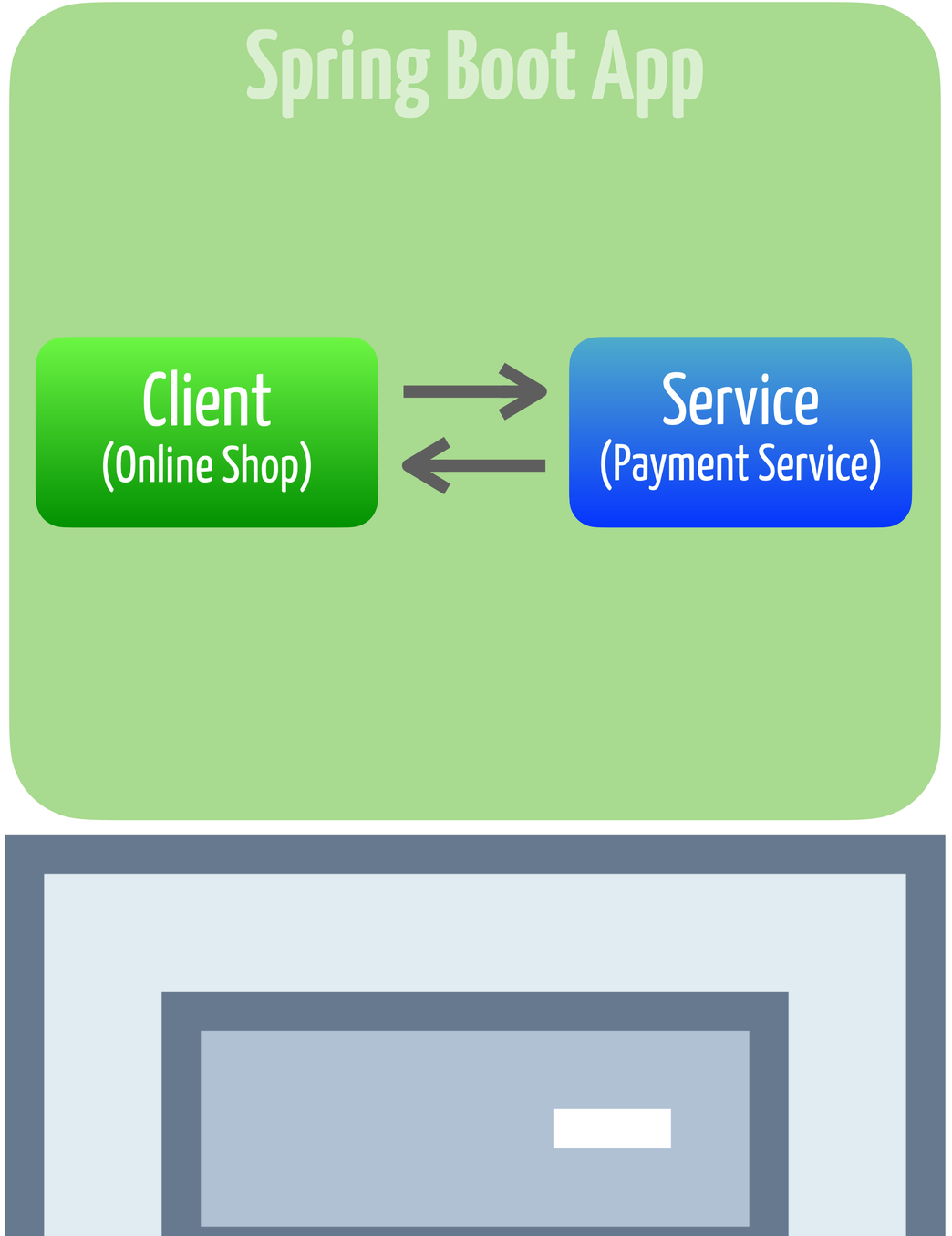

Spring Boot App


Spring Boot App


Spring Boot App
Client
(Online Shop)
Service
(Payment Service)


Spring Boot App
Client
(Online Shop)
Service
(Payment Service)

Local Calls


Spring Boot App
Client
(Online Shop)
Service
(Payment Service)

Local Calls


Spring Boot App
Client
(Online Shop)
Service
(Payment Service)

```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

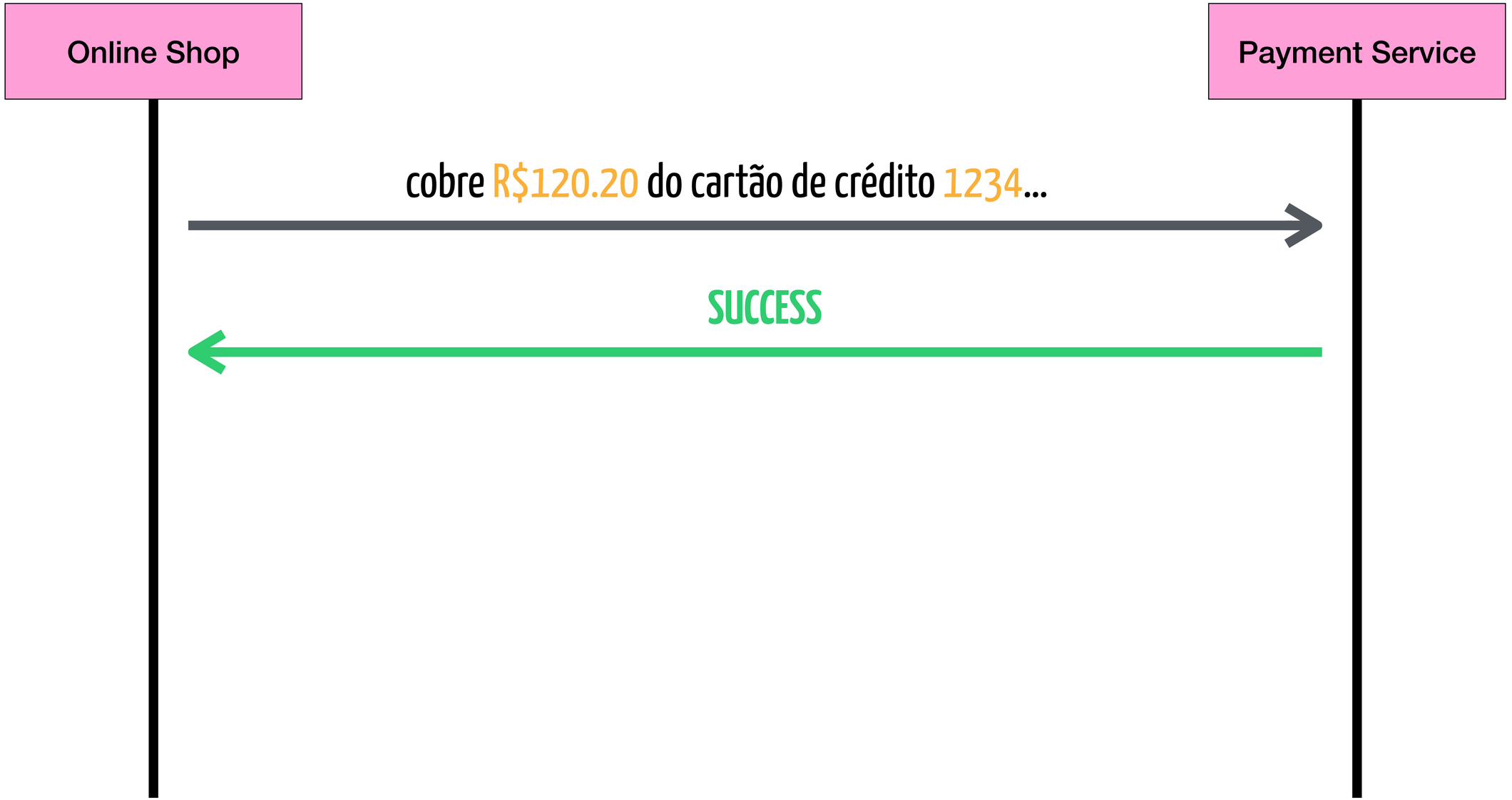
```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

```java
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

Mas quando trabalhamos com sistemas distribuídos...

Pagamento Online



Pagamento Online

Online Shop

Payment Service

cobre R$120.20 do cartão de crédito 1234...

SUCCESS

Pagamento Online

Online Shop

Payment Servic

cobre R$120.20 do cartão de crédito 1234...

SUCCESS

Spring Boot App

Spring Boot App

Spring Boot App

Client
(Online Shop)

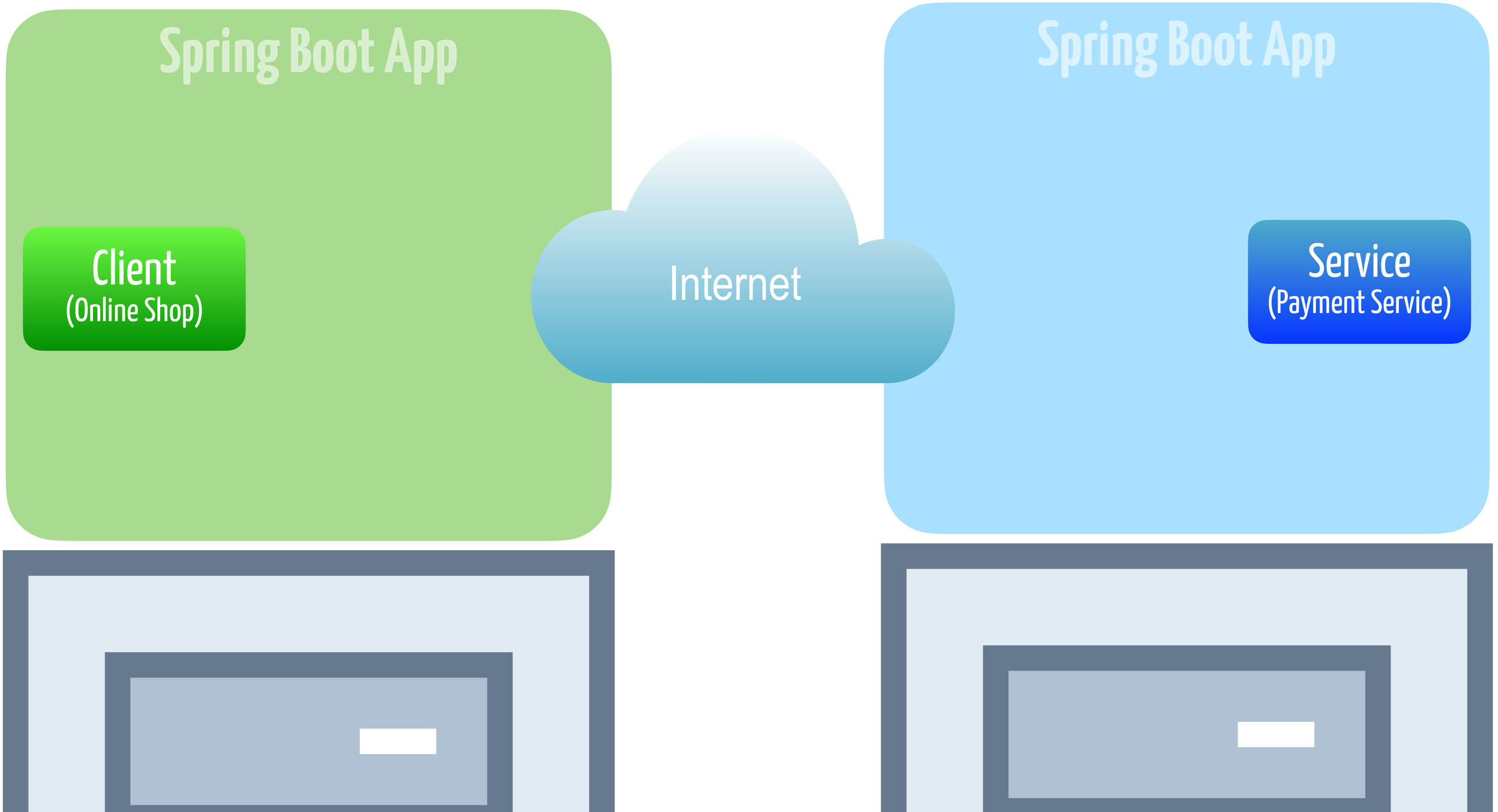
Spring Boot App

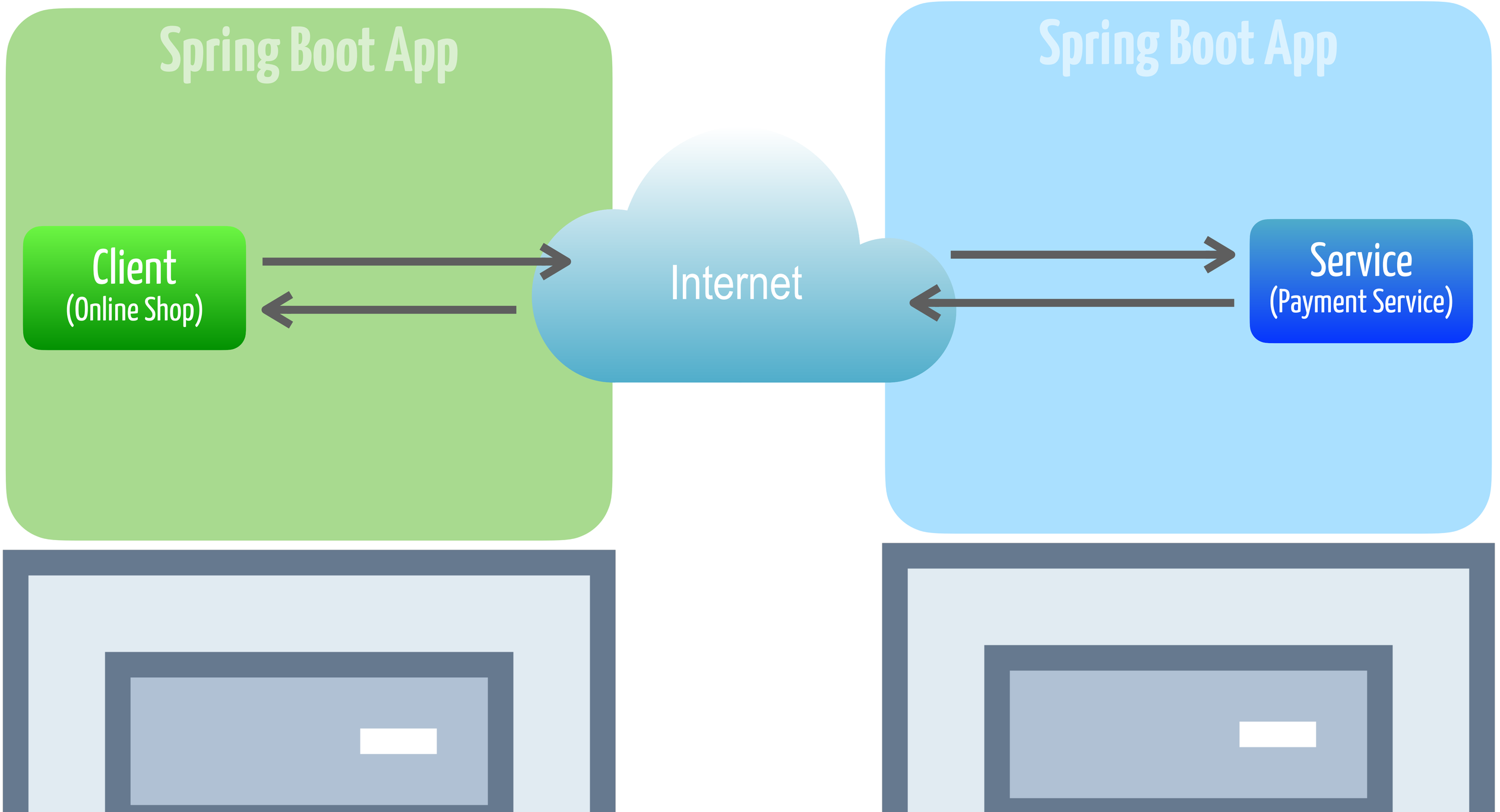
Spring Boot App

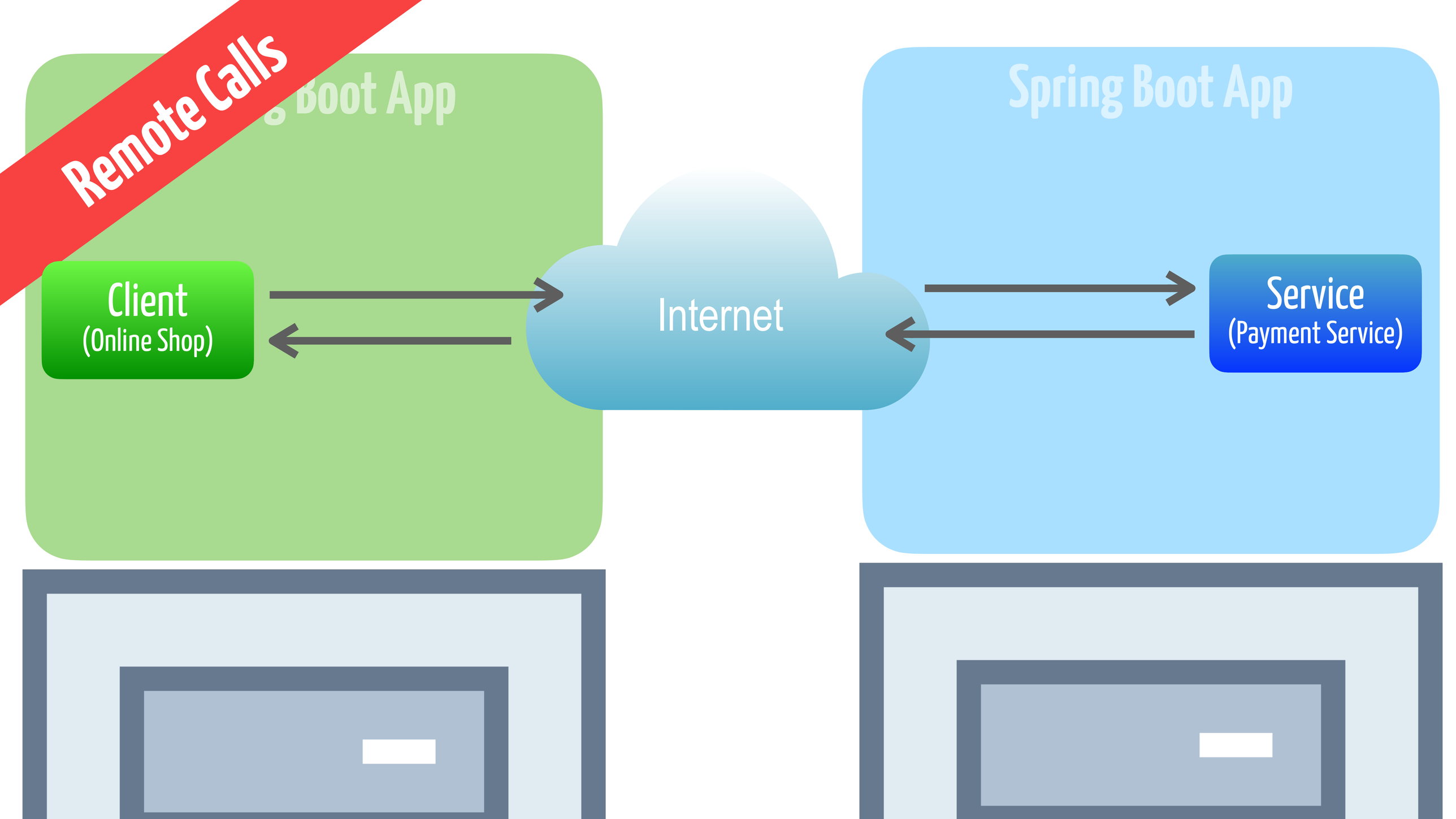
Client
(Online Shop)

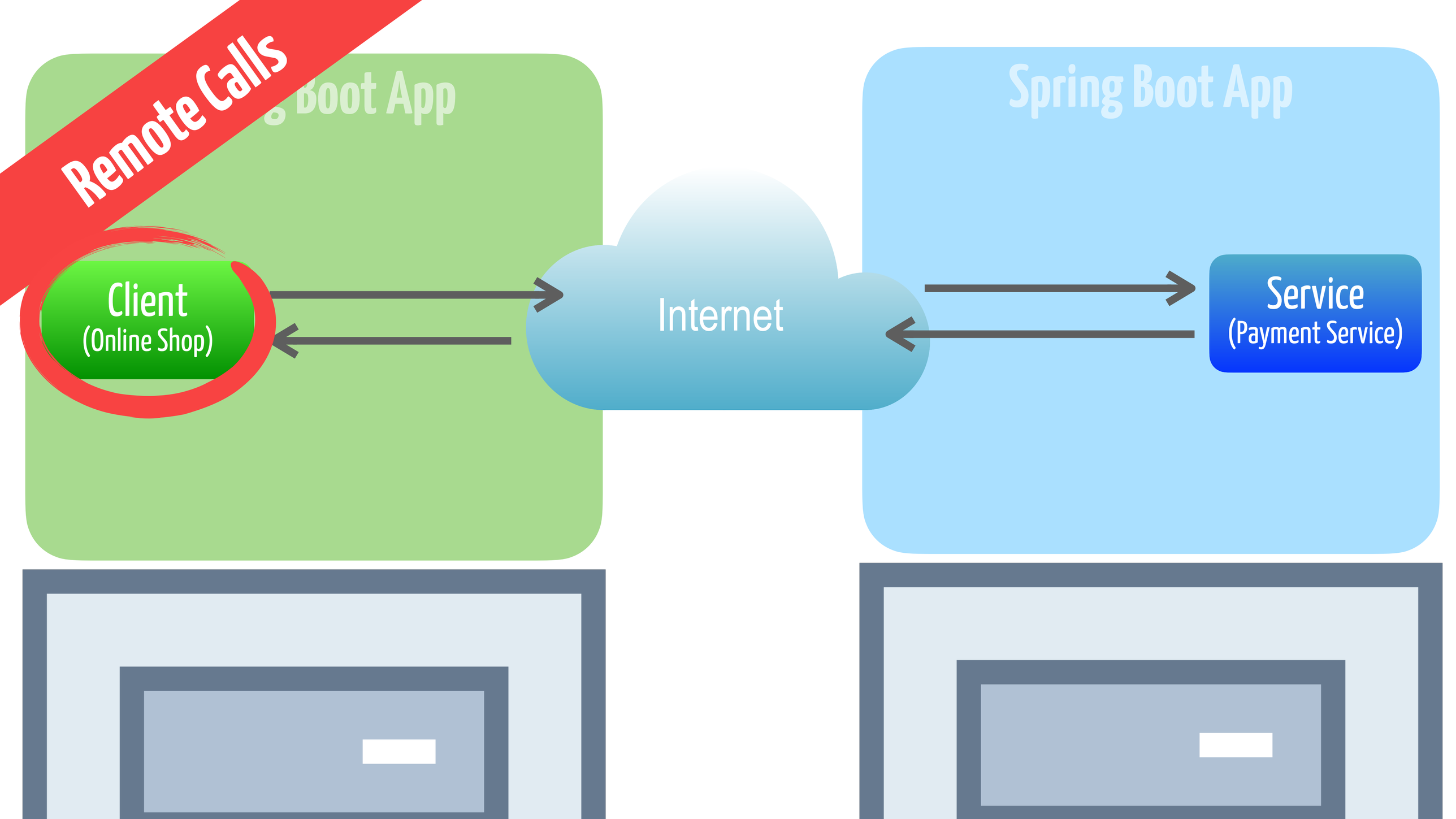
Spring Boot App

Service
(Payment Service)


Spring Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)


Spring Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)


Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)


Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)

```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

RPC framework

RPC Framework

Online Shop

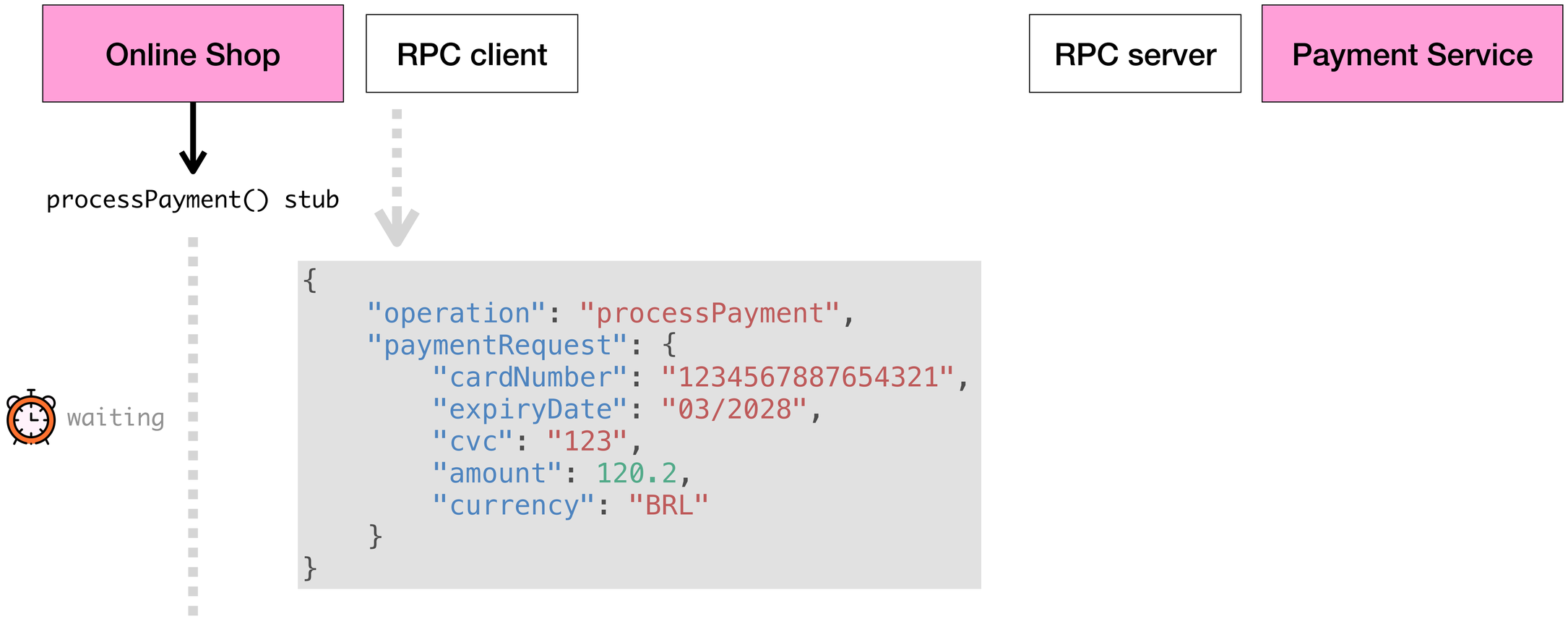
Payment Service

RPC Framework

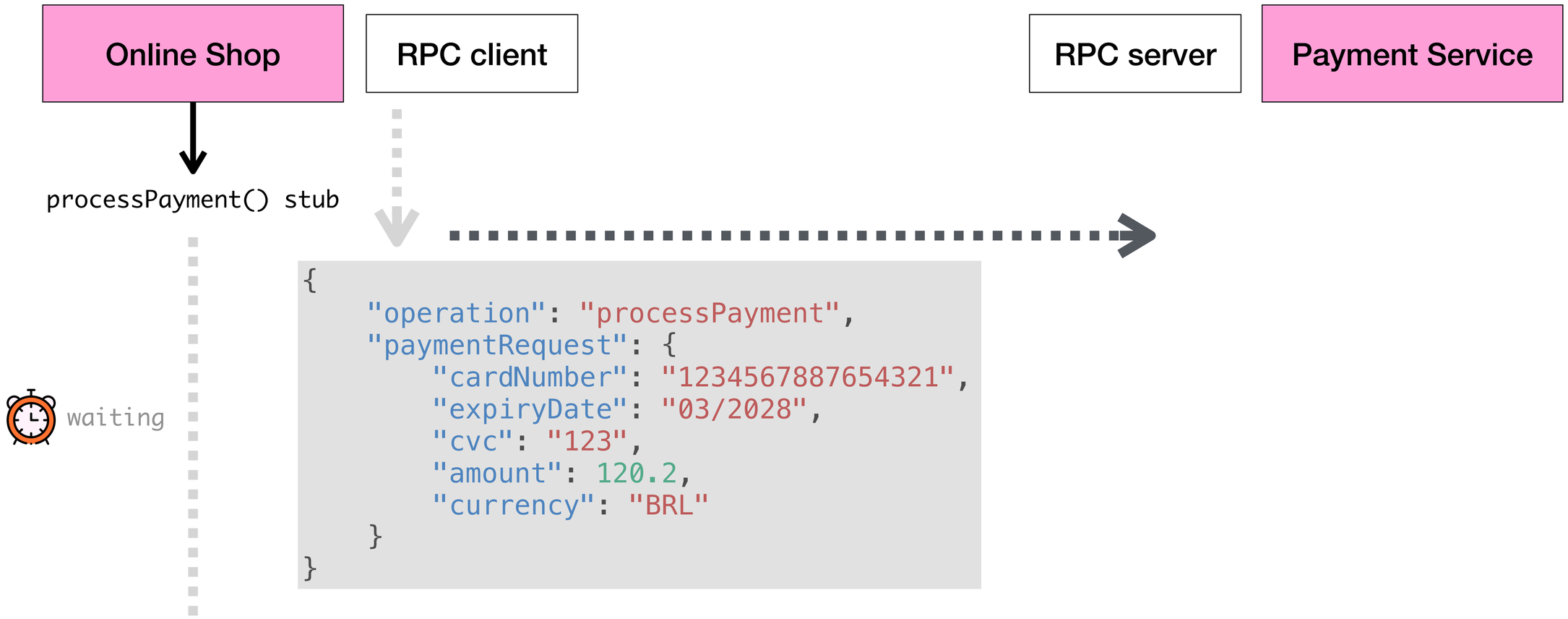
RPC Framework

RPC Framework

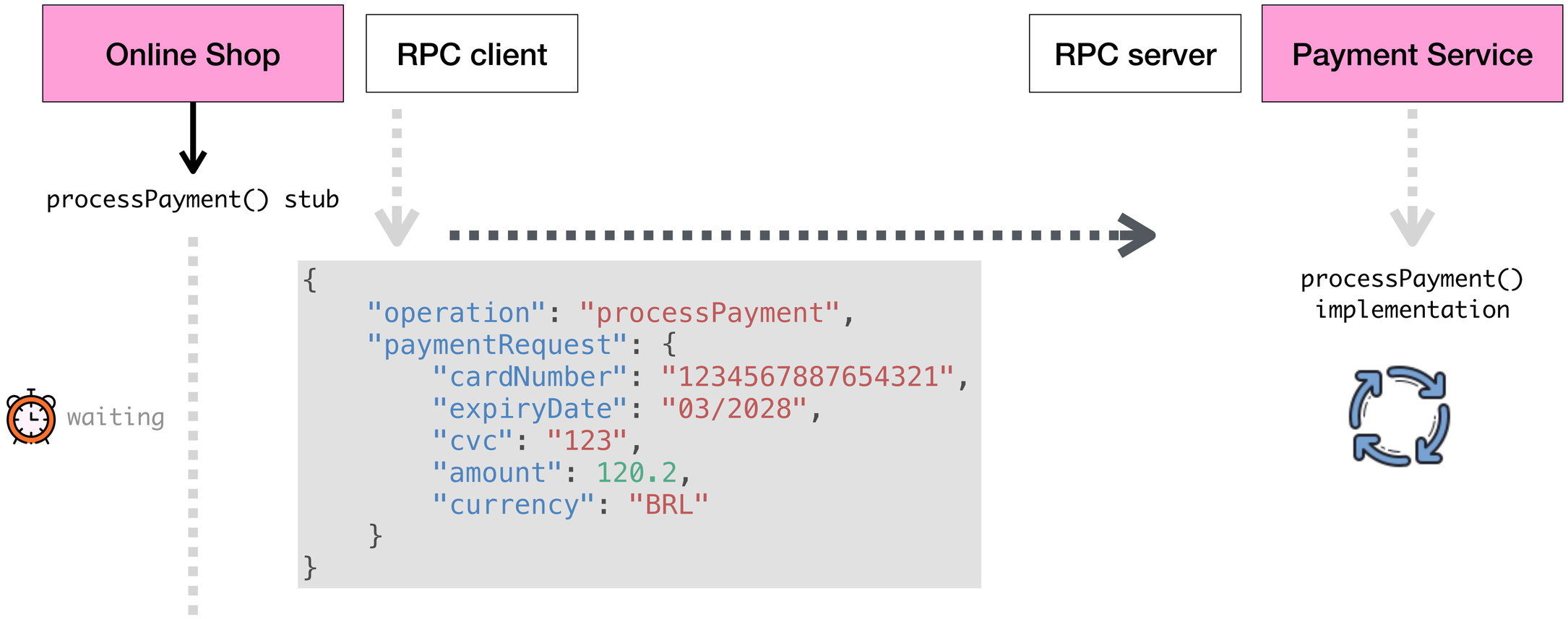
RPC Framework



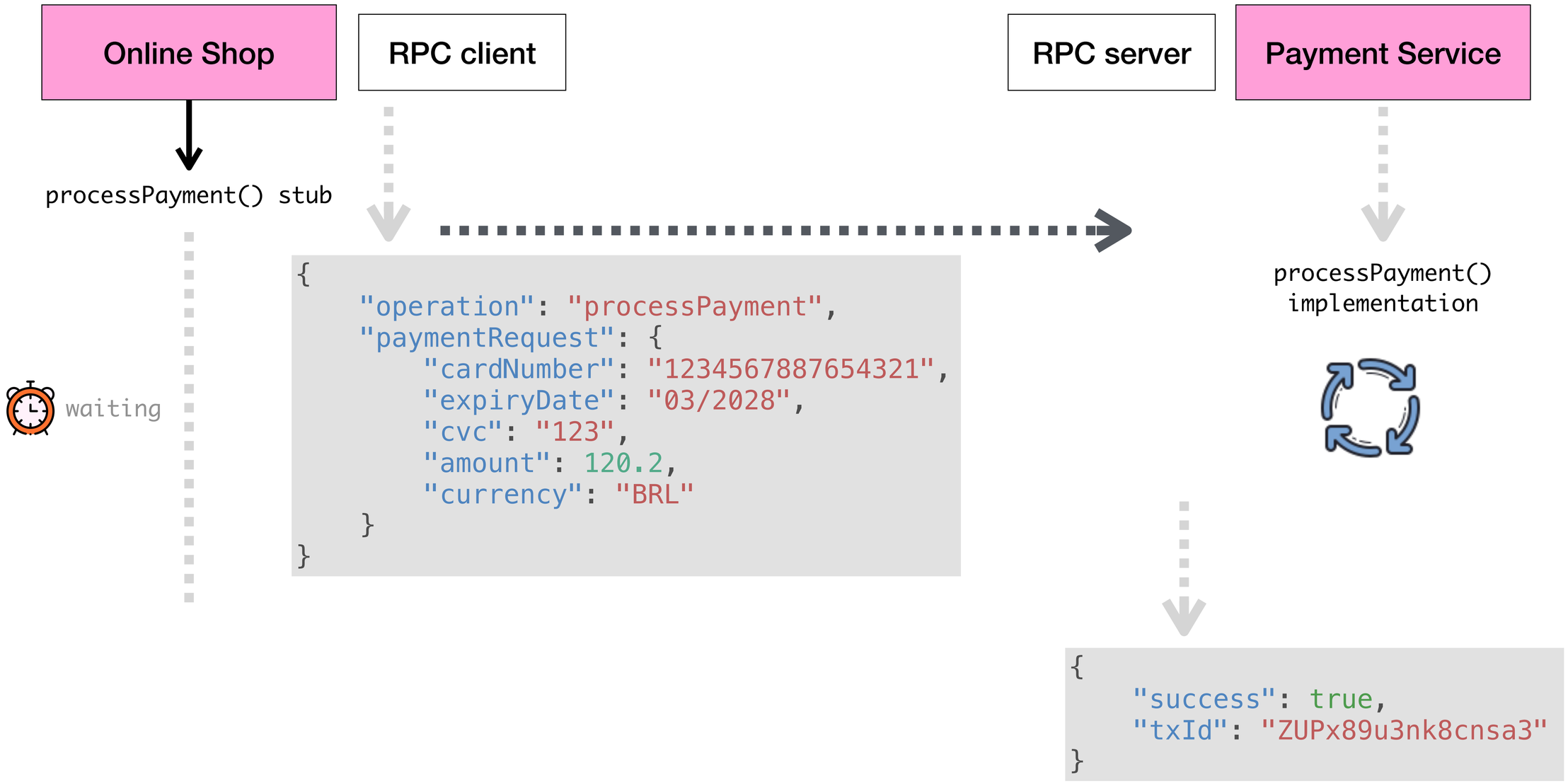
RPC Framework



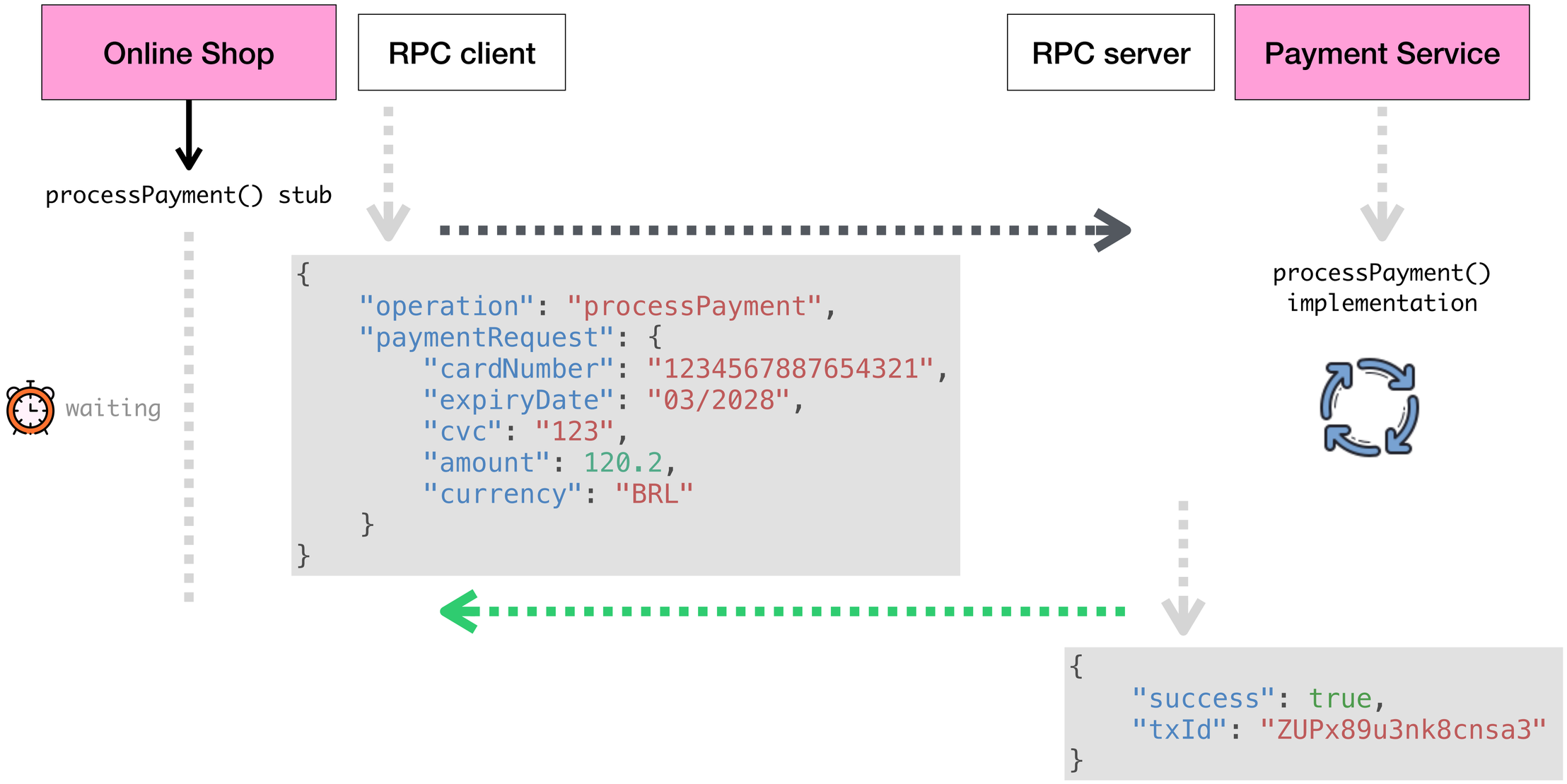
RPC Framework



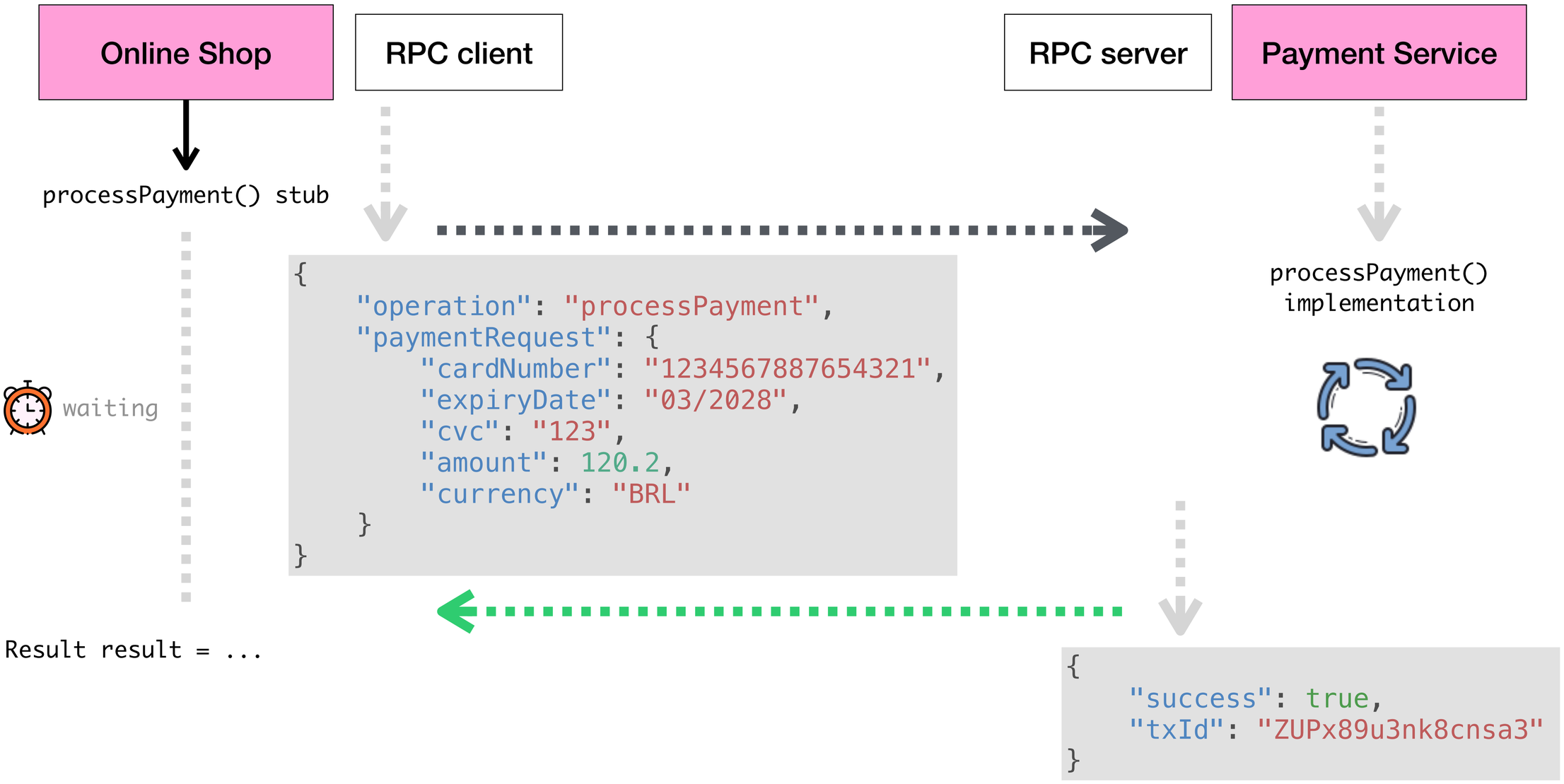
RPC Framework



RPC Framework

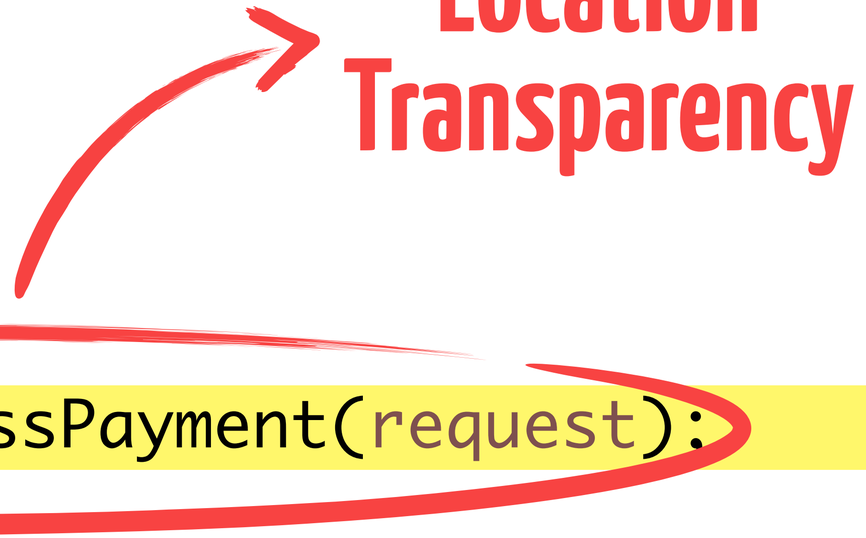


RPC Framework

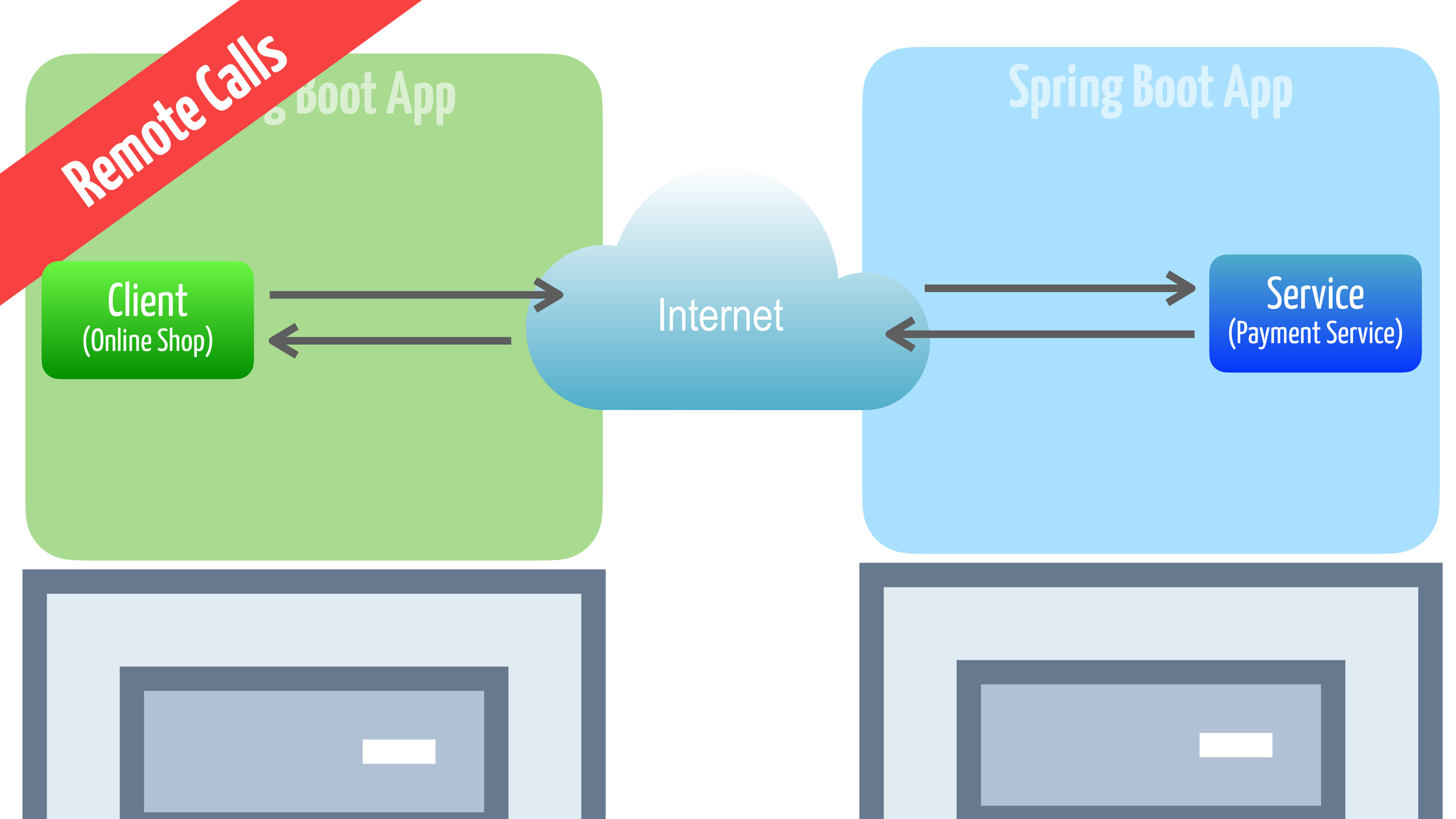


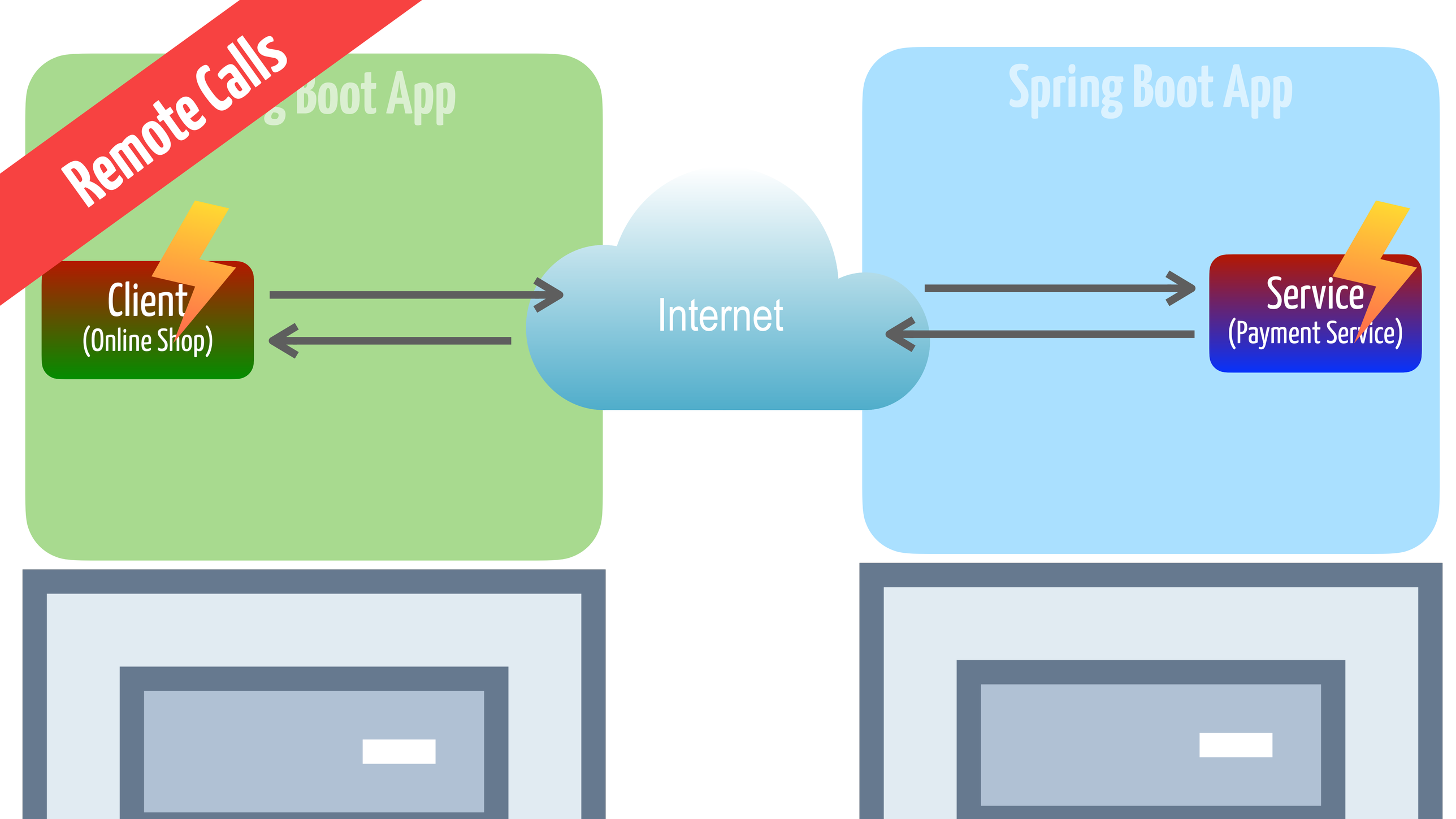
```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

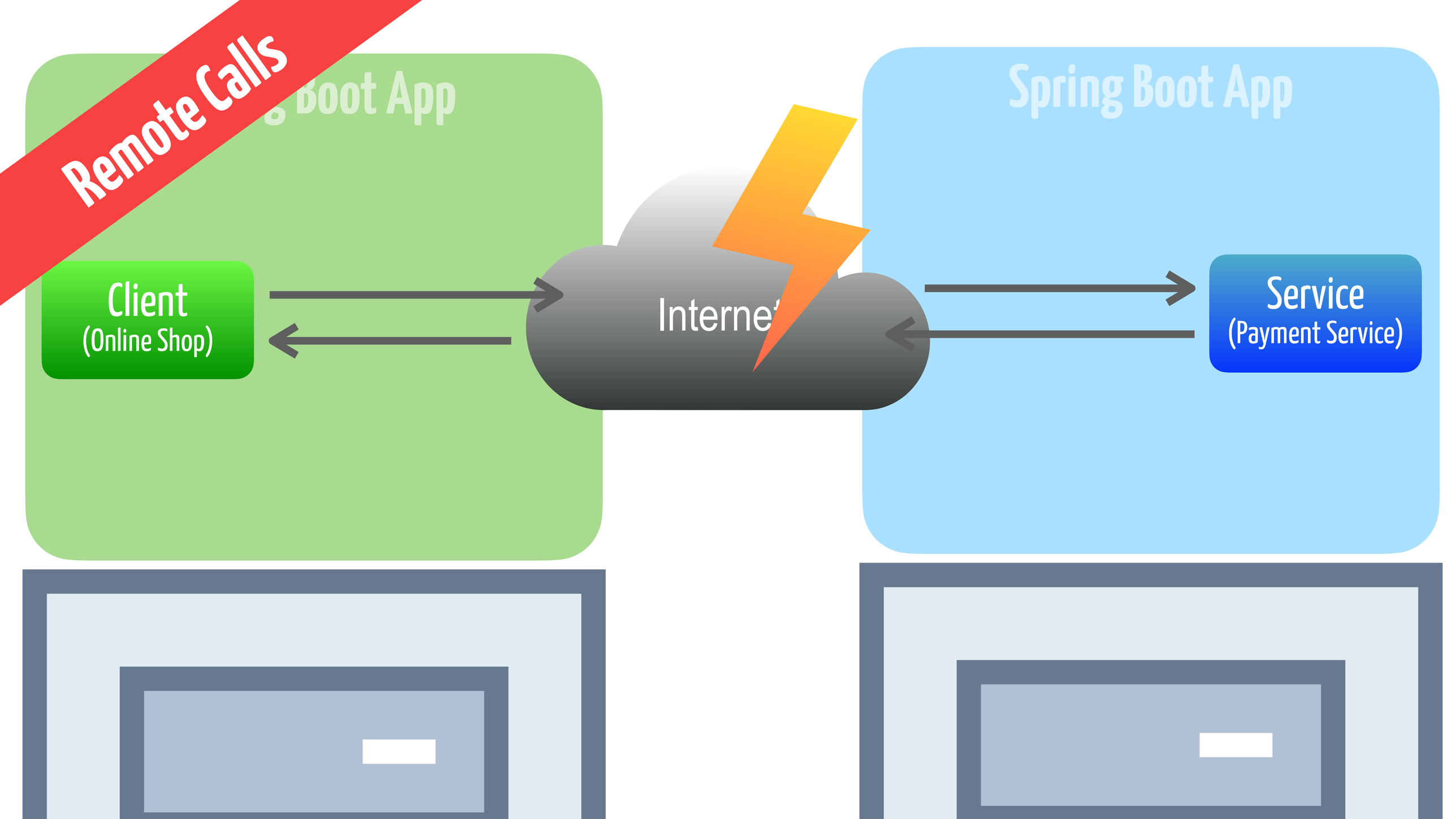
```
PaymentRequest request = new PaymentRequest();
request.setCardNumber("1234 5678 8765 4321");
request.setExpiryDate("03/2028");
request.setCvc("123");
request.setAmount(120.2);
request.setCurrency(Currency.BRL);

Result result = paymentService.processPayment(request);
if (result.isSuccess()) {
    // processa fluxo do pedido...
}
```

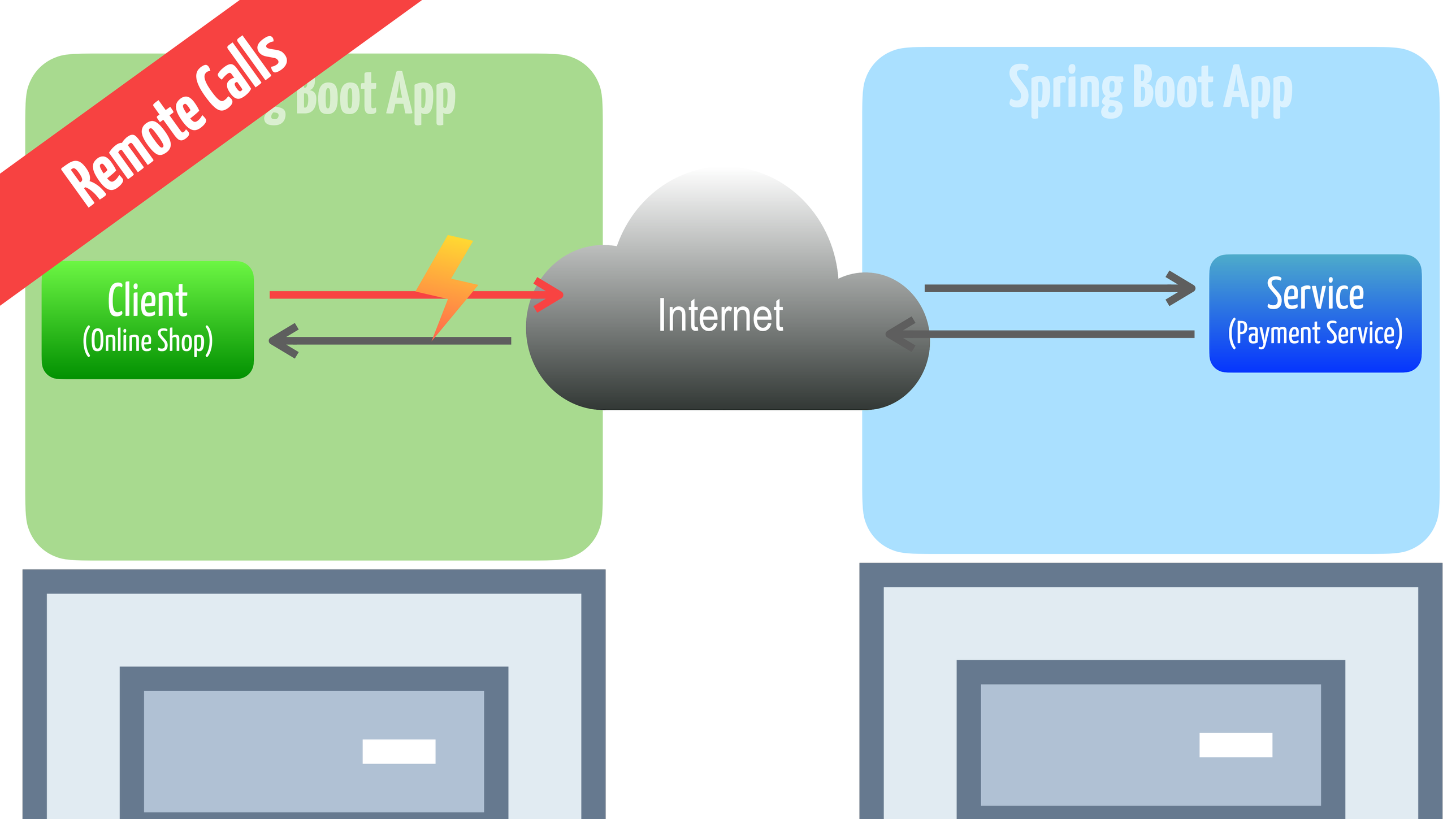


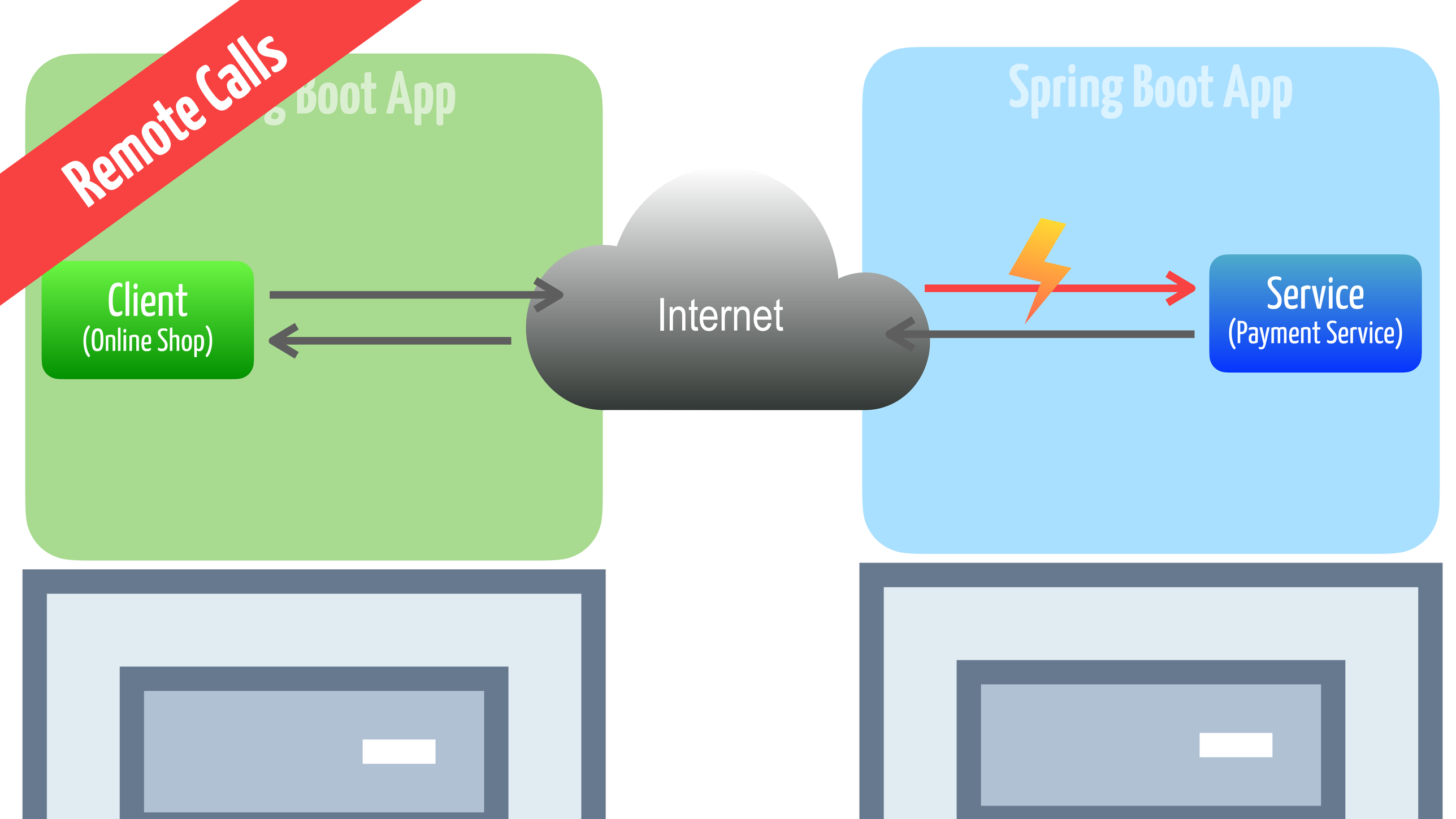
Mas na prática...

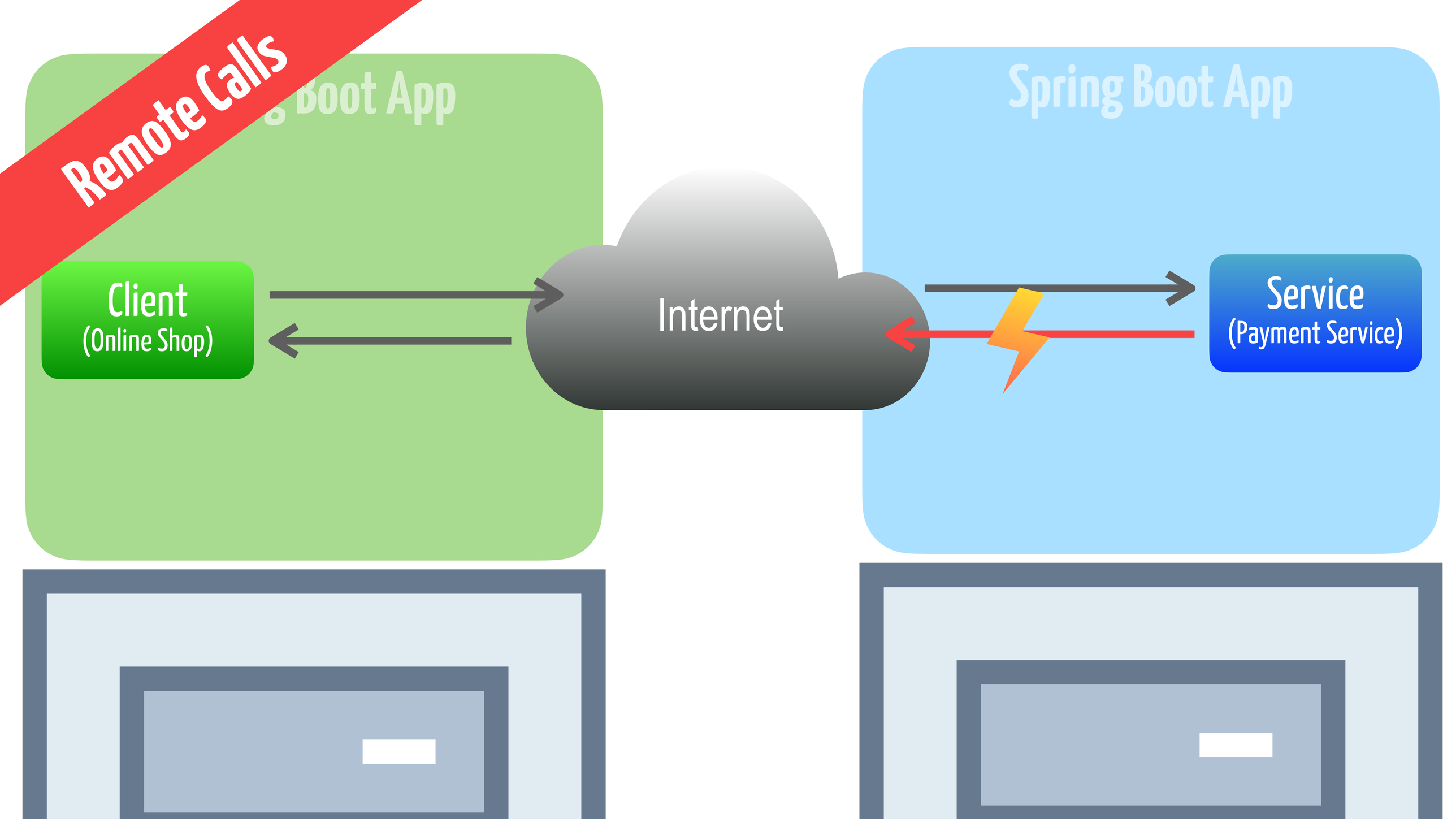

Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)

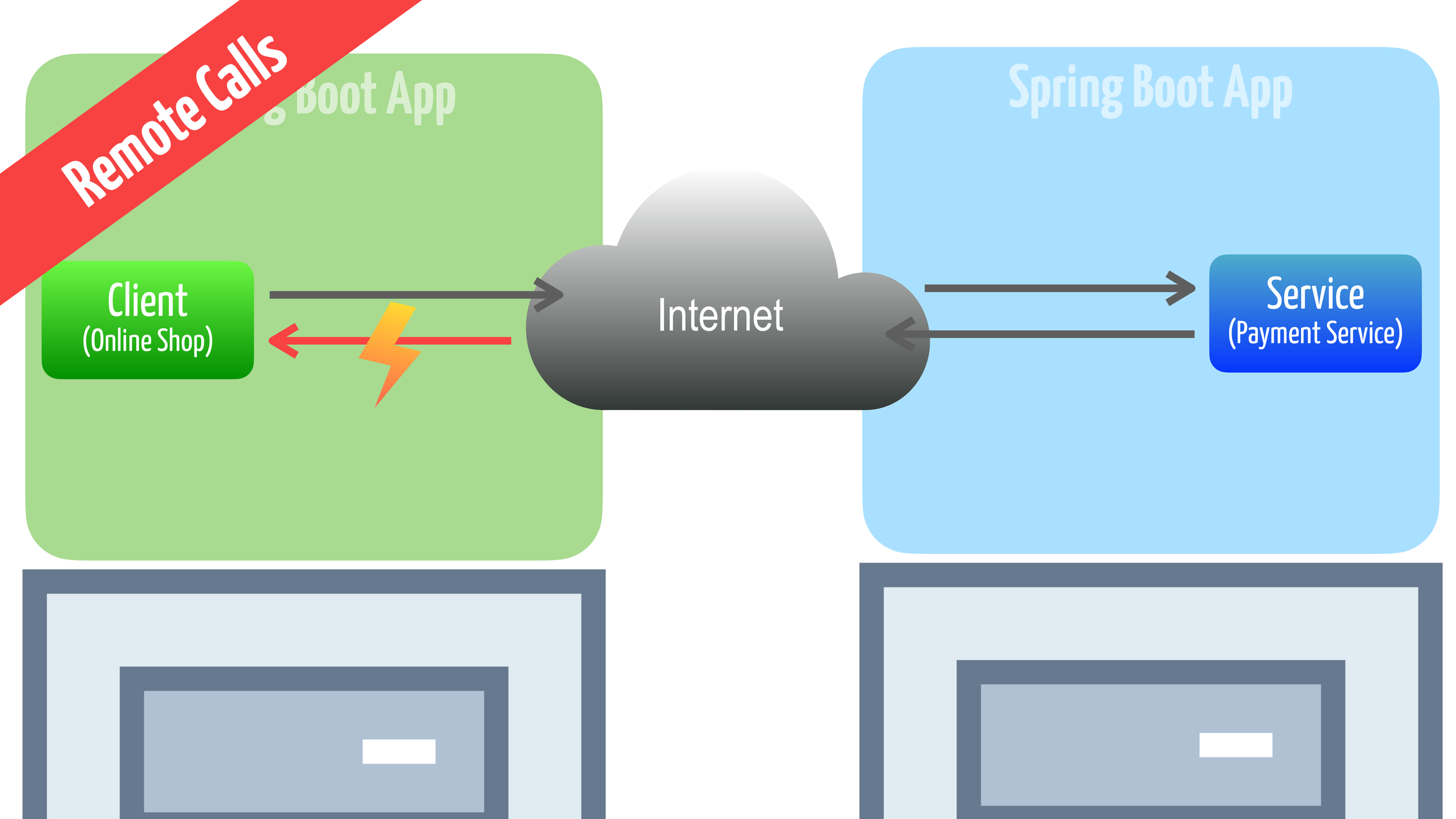

Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)

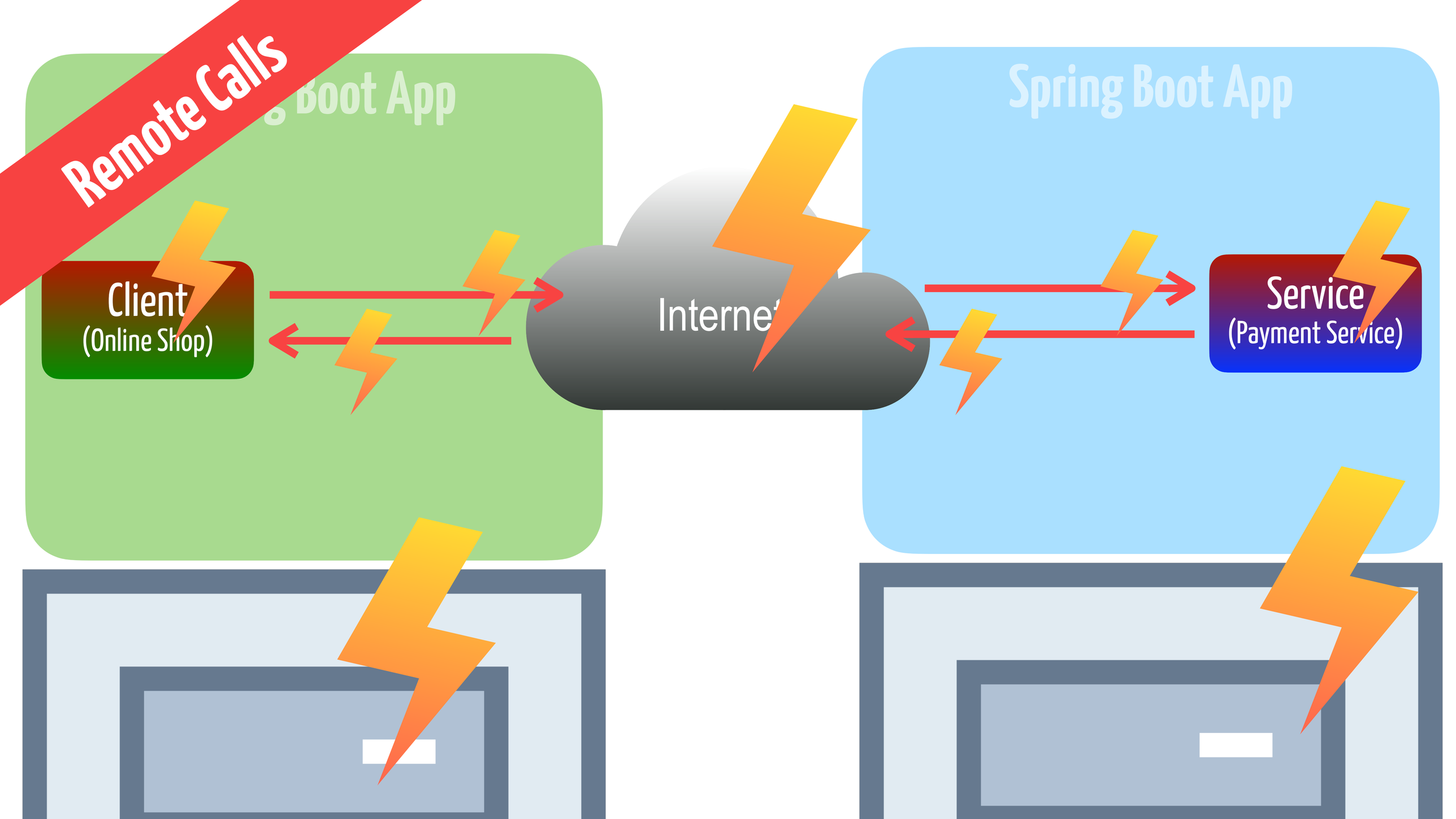

Remote Calls
Boot App
Spring Boot App
Client
(Online Shop)
Internet
Service
(Payment Service)


Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)


Remote Calls
Boot App
Spring Boot App
Client
(Online Shop)
Internet
Service
(Payment Service)


Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)


Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)

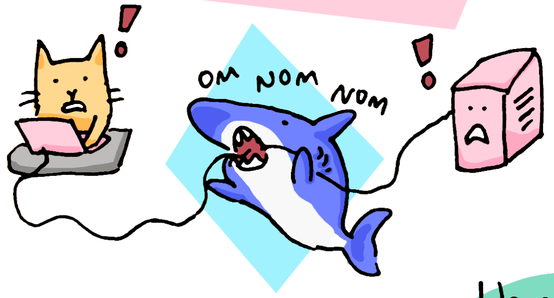

Remote Calls
Boot App
Client
(Online Shop)
Internet
Spring Boot App
Service
(Payment Service)

the 8 Fallacies of Distributed Computing

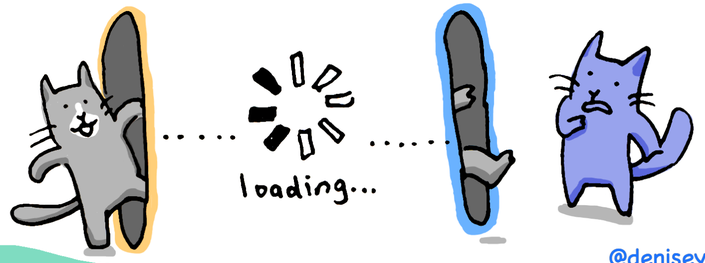
@deniseyu21

Originally formulated by L. Peter Deutsch & Colleagues at Sun Microsystems in 1994; #8 added in 1997 by James Gosling

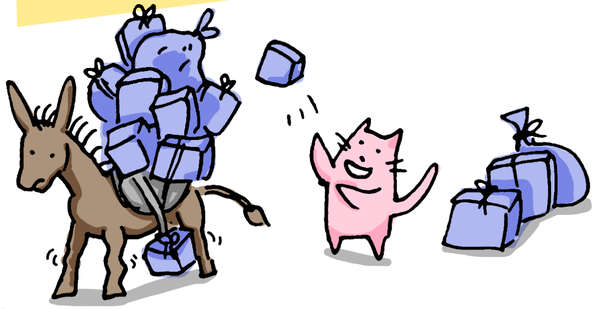
(1) The network is reliable



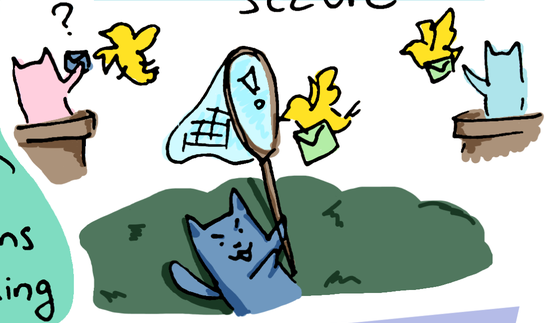
(2) Latency is ZERO



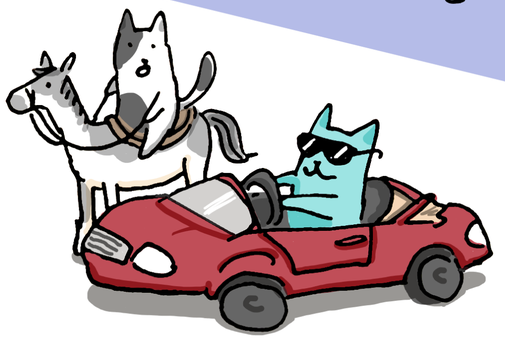
(3) Bandwidth is infinite



(4) The network is secure



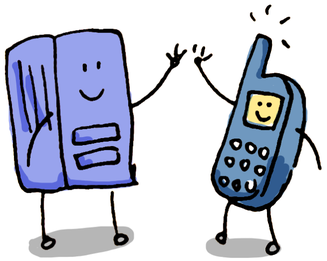
(5) Topology doesn't change



(6) There is only one administrator

(7) Transport costs $0

(8) The network is homogeneous

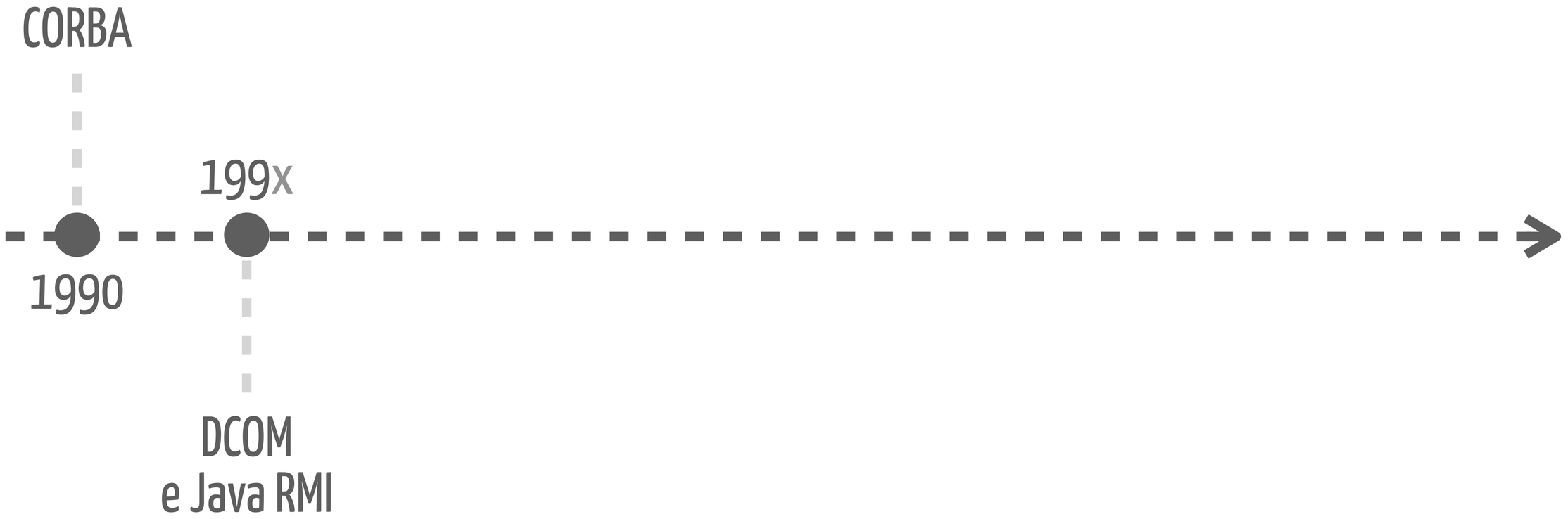


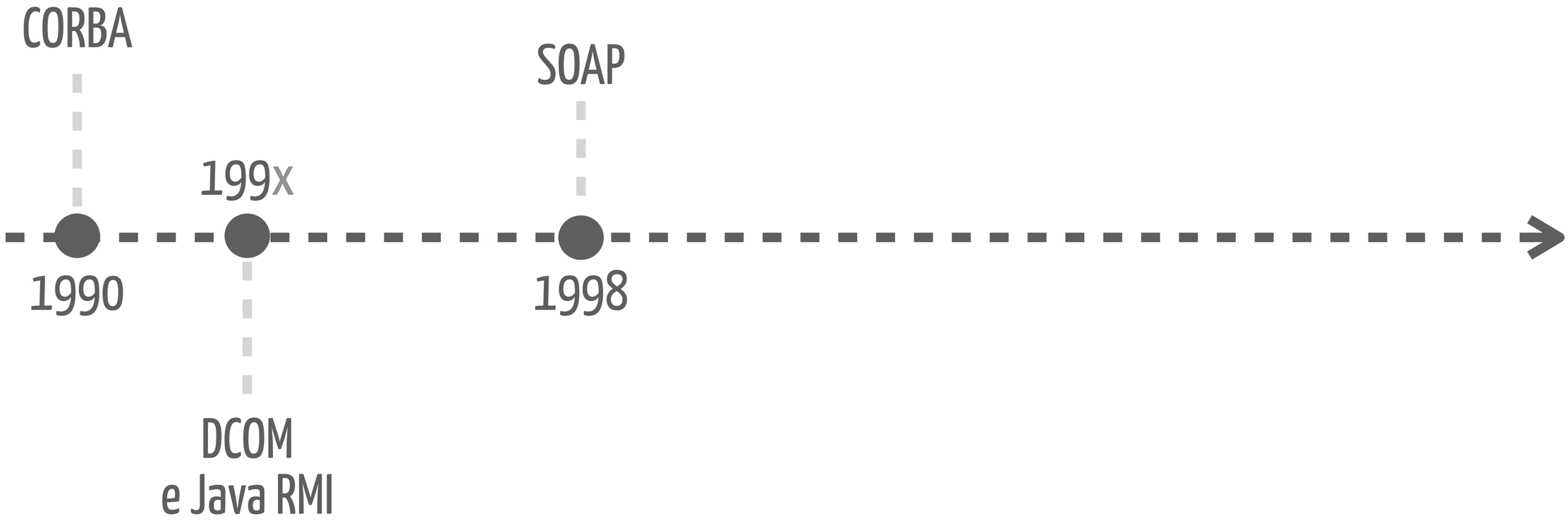
https://deniseyu.io/art/

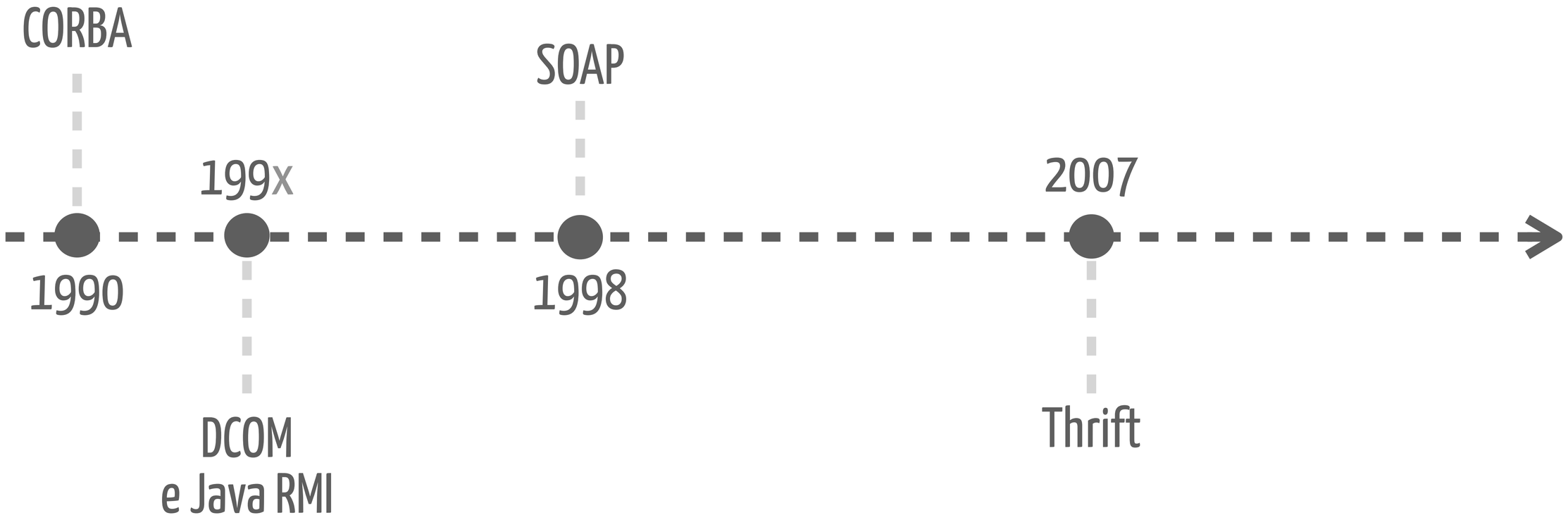
Falando em frameworks...

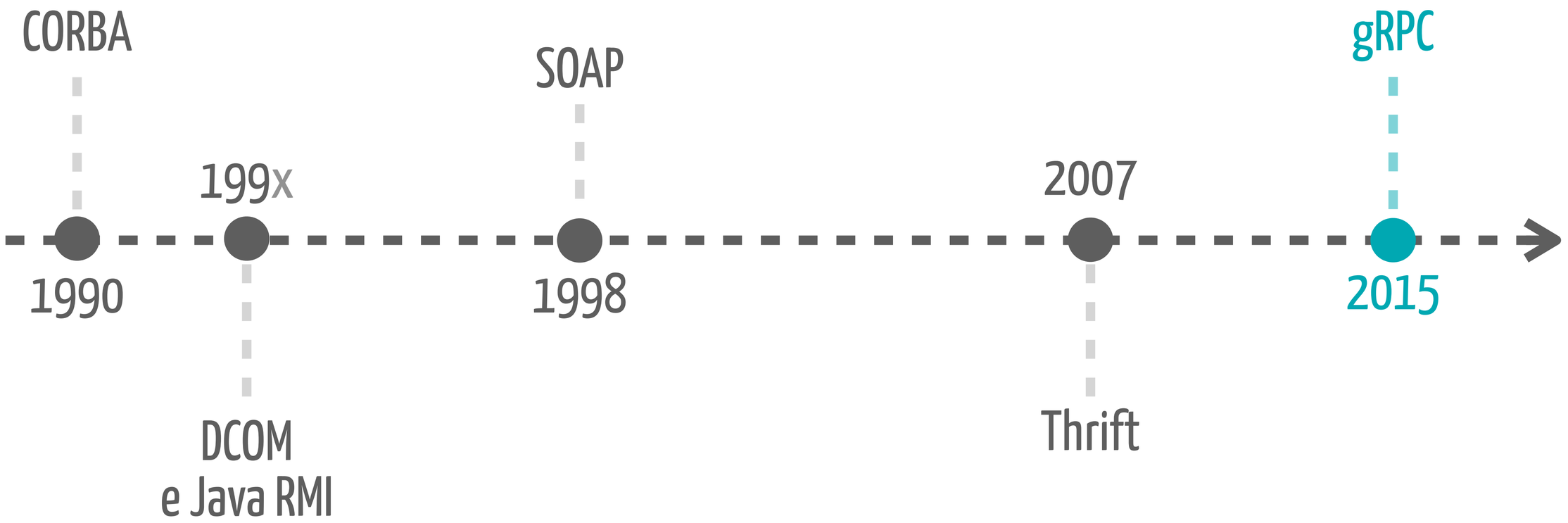

1990


CORBA
1990

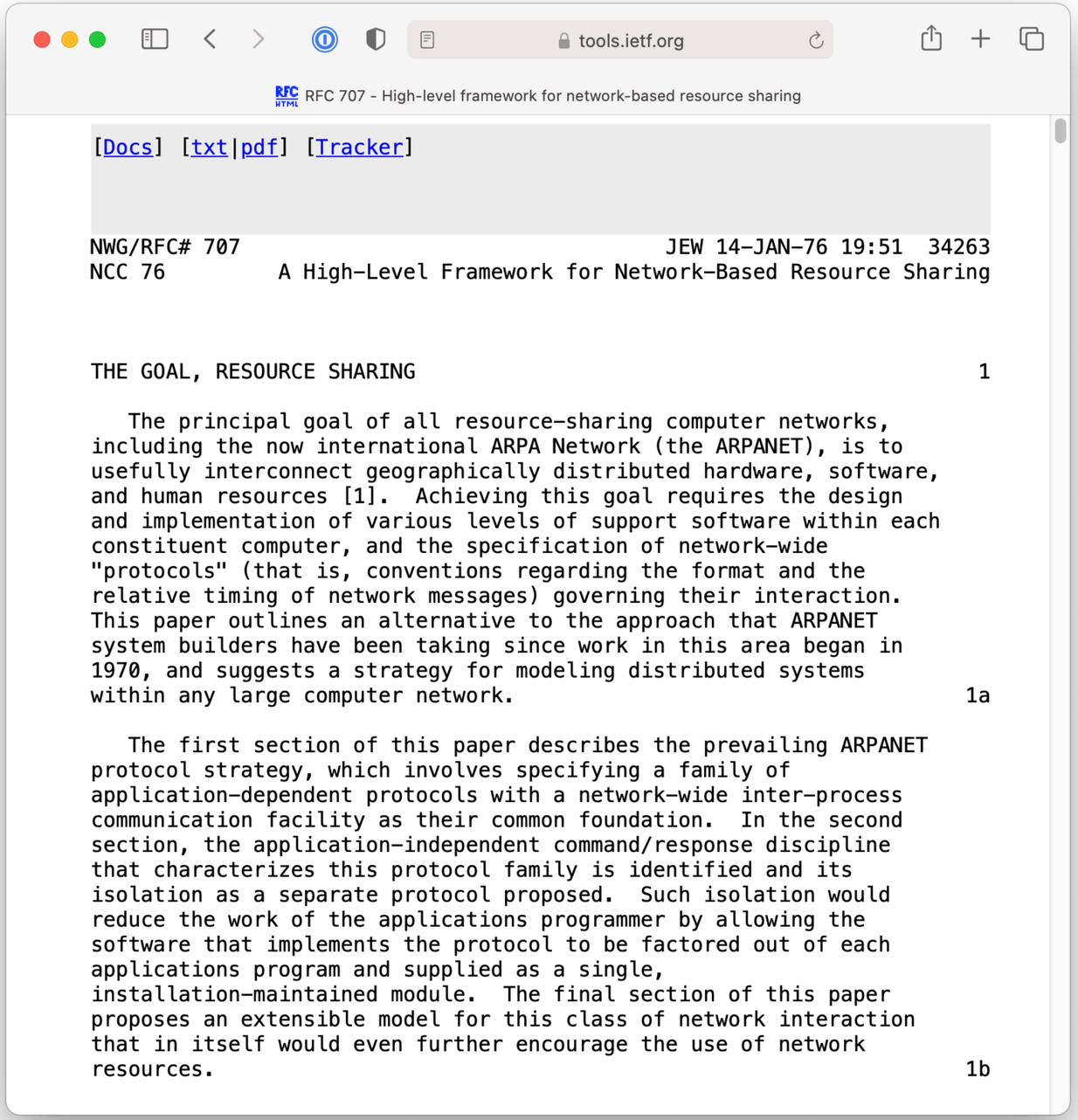

CORBA
199x
1990
DCOM
e Java RMI


CORBA
SOAP
199x
1990
1998
DCOM
e Java RMI

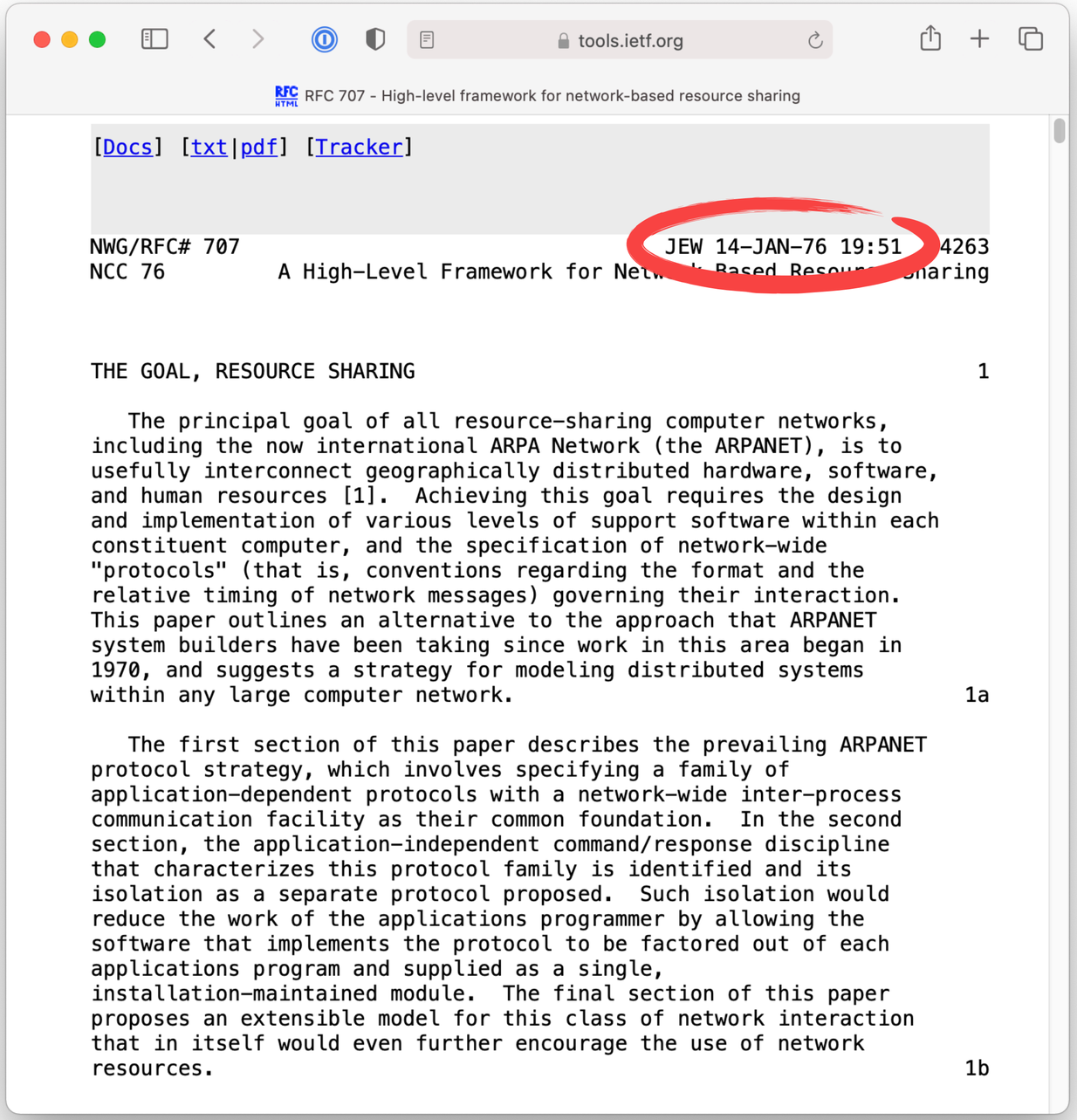

CORBA
SOAP
199x
2007
1990
1998
DCOM
e Java RMI
Thrift


CORBA
199x
SOAP
2007
gRPC
1990
1998
2015
DCOM
e Java RMI
Thrift

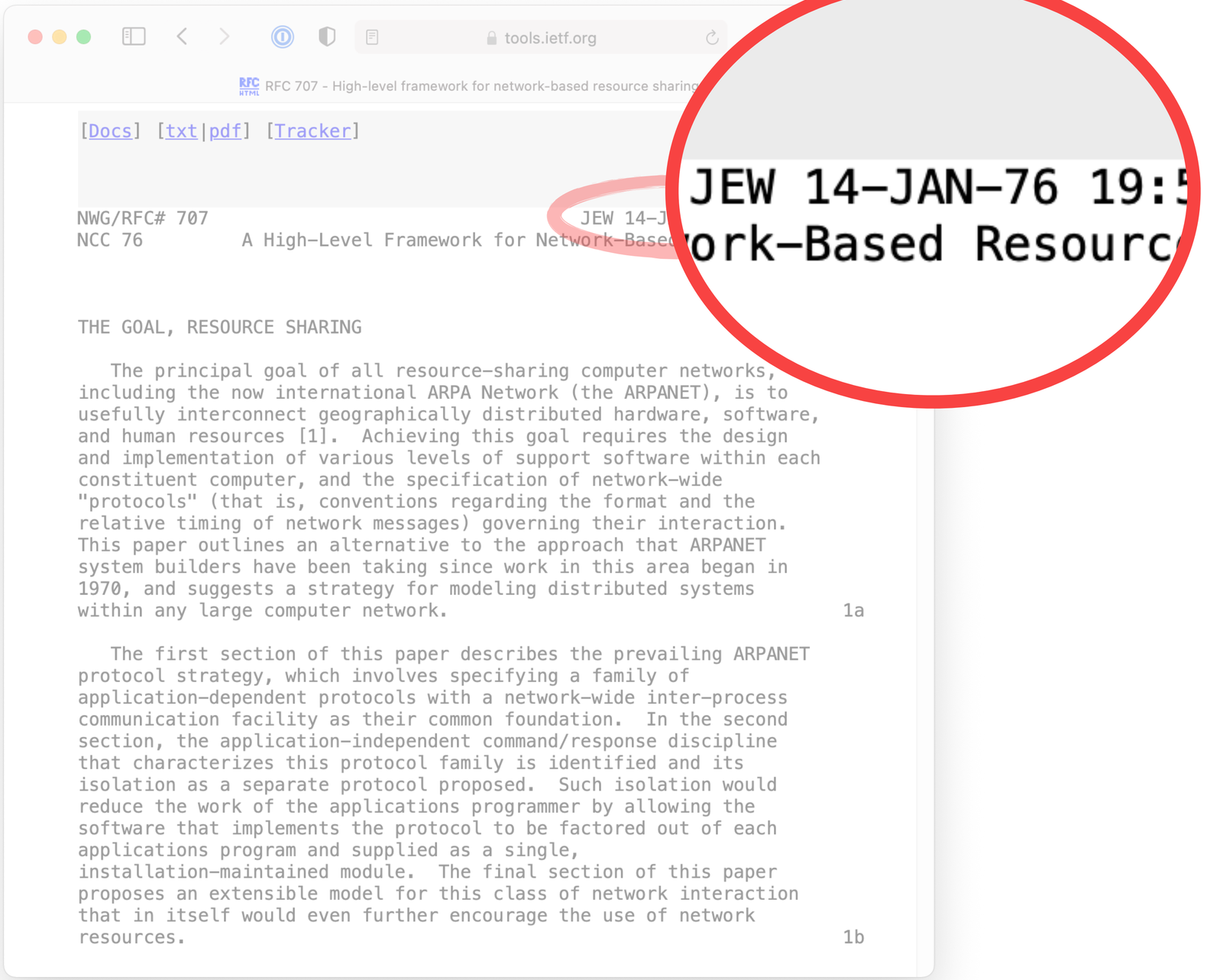
https://tools.ietf.org/html/rfc707



https://tools.ietf.org/html/rfc707



https://tools.ietf.org/html/rfc707



gRPC Remote Procedure Call framework

gRPC Remote Procedure Call framework

gRPC Remote Procedure Call framework

simple & idiomatic

gRPC Remote Procedure Call framework

simple & idiomatic

performant & scalable

gRPC Remote Procedure Call framework

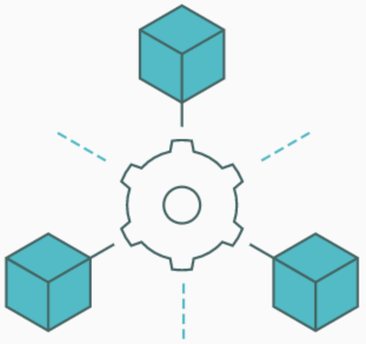
simple & idiomatic

performant & scalable

interoperable & extensible

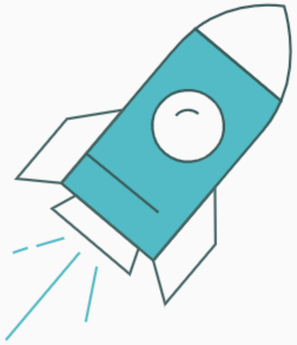
Simple service definition

Define your service using Protocol Buffers, a powerful binary serialization toolset and language



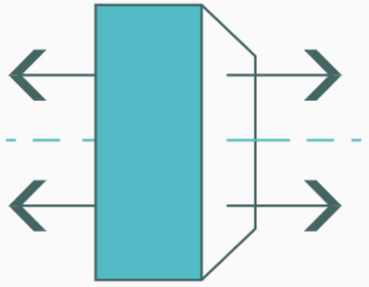
Start quickly and scale

Install runtime and dev environments with a single line and also scale to millions of RPCs per second with the framework



Works across languages and platforms

Automatically generate idiomatic client and server stubs for your service in a variety of languages and platforms



Bi-directional streaming and integrated auth

Bi-directional streaming and fully integrated pluggable authentication with HTTP/2-based transport

gRPC vs REST API with JSON

High-level comparison

The following table offers a high-level comparison of features between gRPC and HTTP APIs with JSON.

Feature	gRPC	HTTP APIs with JSON
Contract	Required (*.proto*)	Optional (OpenAPI)
Protocol	HTTP/2	HTTP
Payload	Protobuf (small, binary)	JSON (large, human readable)
Prescriptiveness	Strict specification	Loose. Any HTTP is valid.
Streaming	Client, server, bi-directional	Client, server
Browser support	No (requires grpc-web)	Yes
Security	Transport (TLS)	Transport (TLS)
Client code-generation	Yes	OpenAPI + third-party tooling

https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-5.0

gRPC Remote Procedure Call framework

simple & idiomatic

performant & scalable

interoperable & extensible

gRPC Remote Procedure Call framework

IDL

performant & scalable

interoperable & extensible

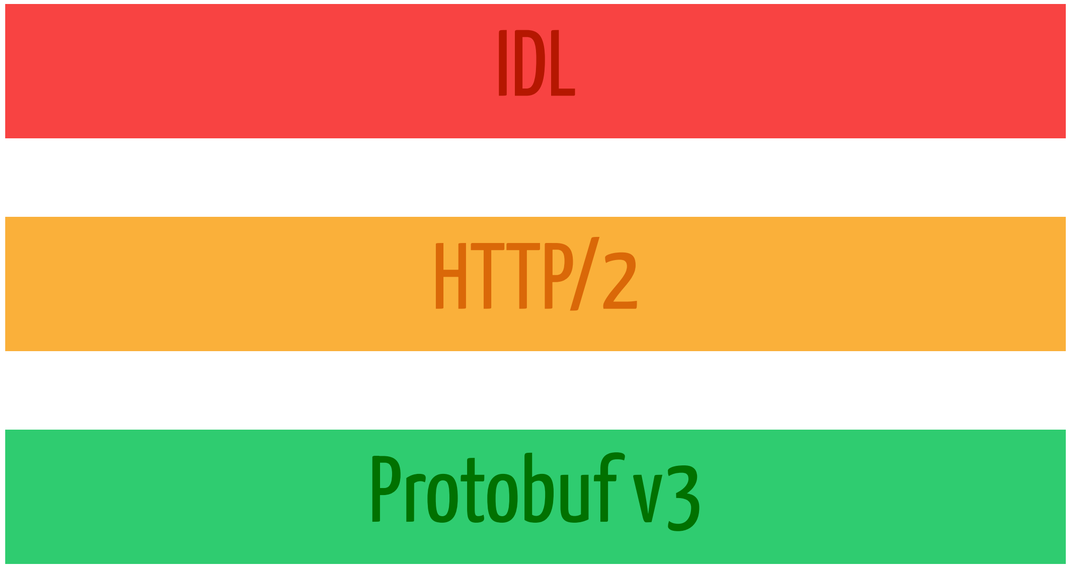
gRPC Remote Procedure Call framework

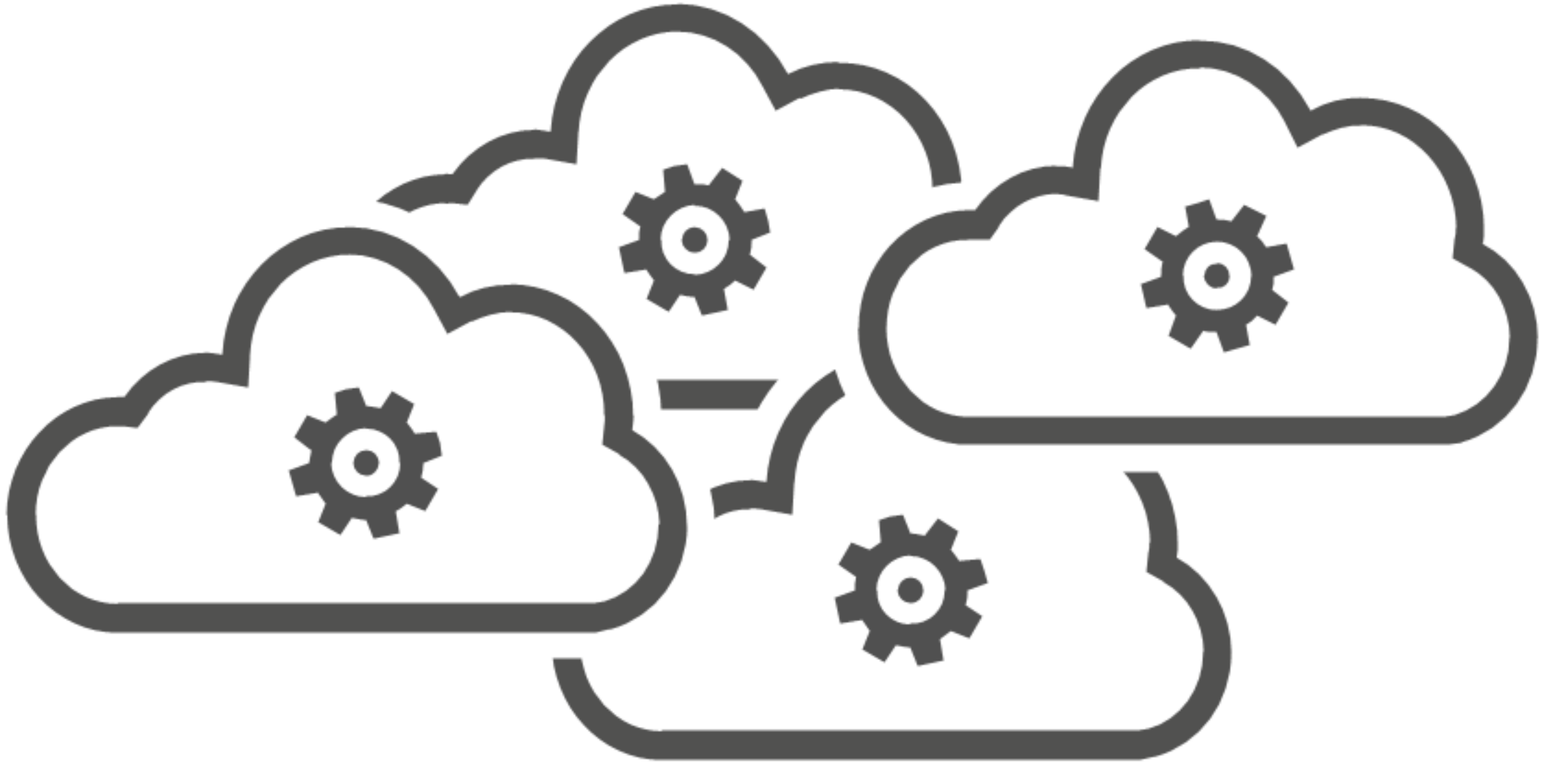
IDL

HTTP/2

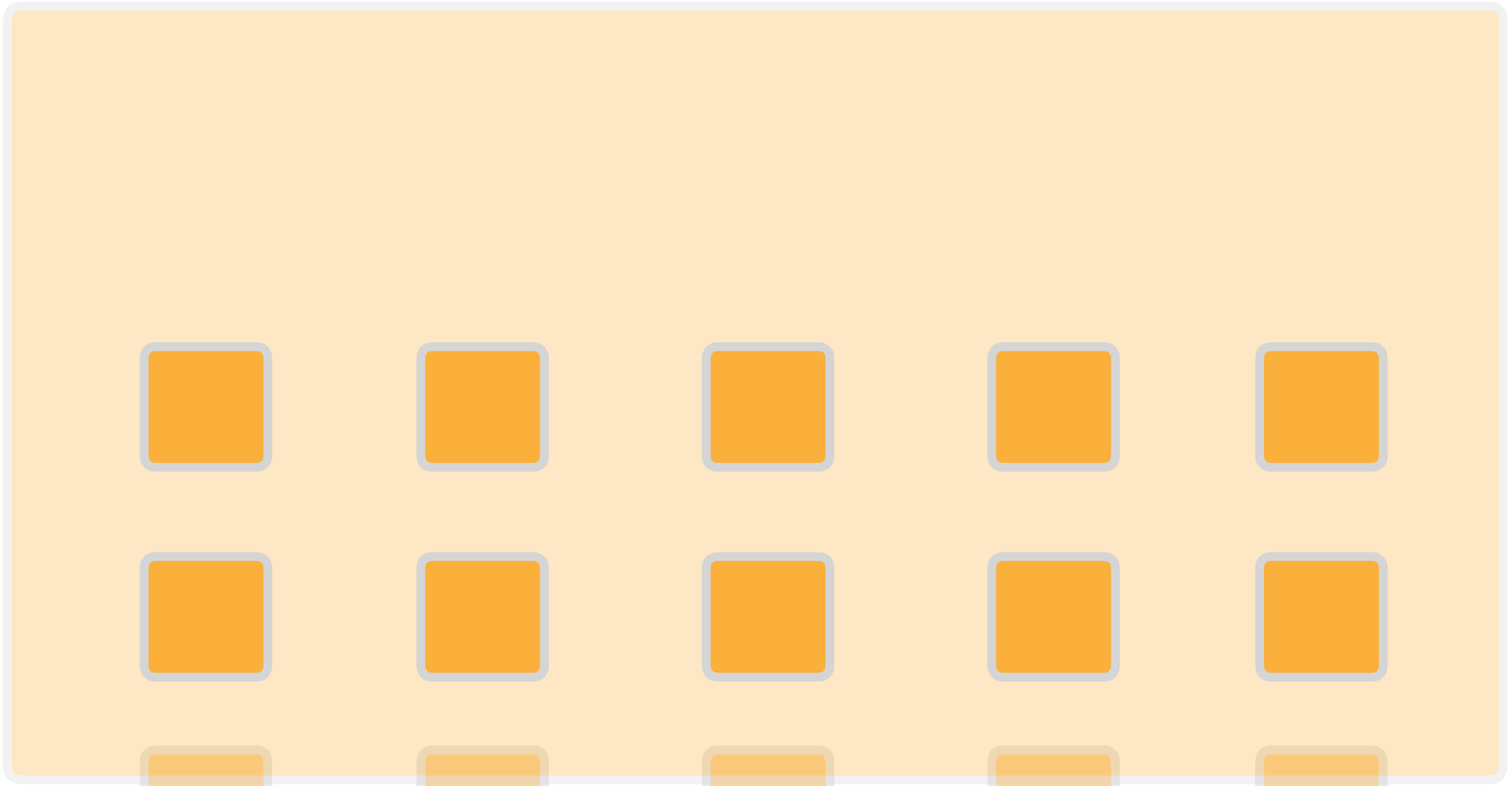
interoperable & extensible

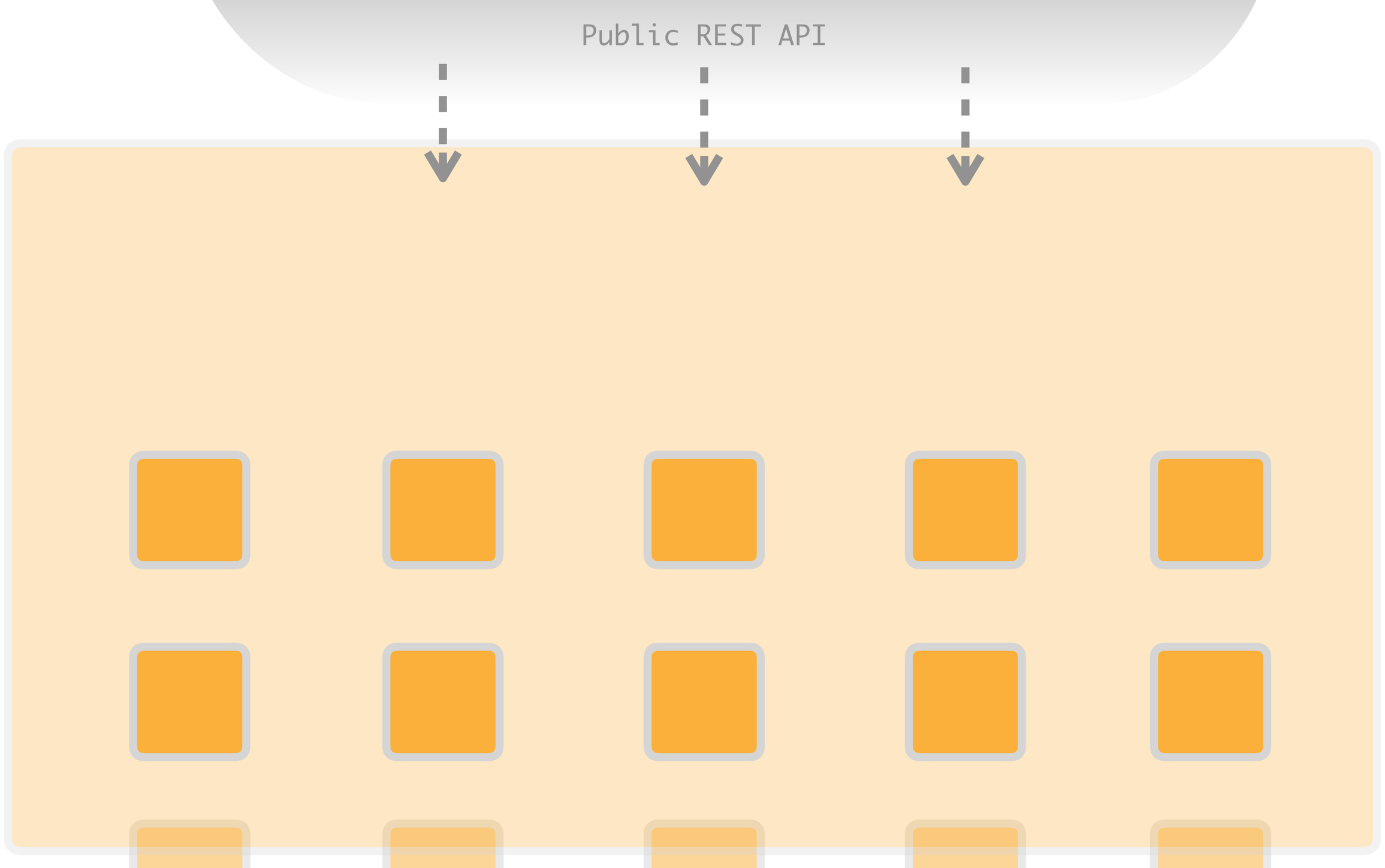
gRPC Remote Procedure Call framework

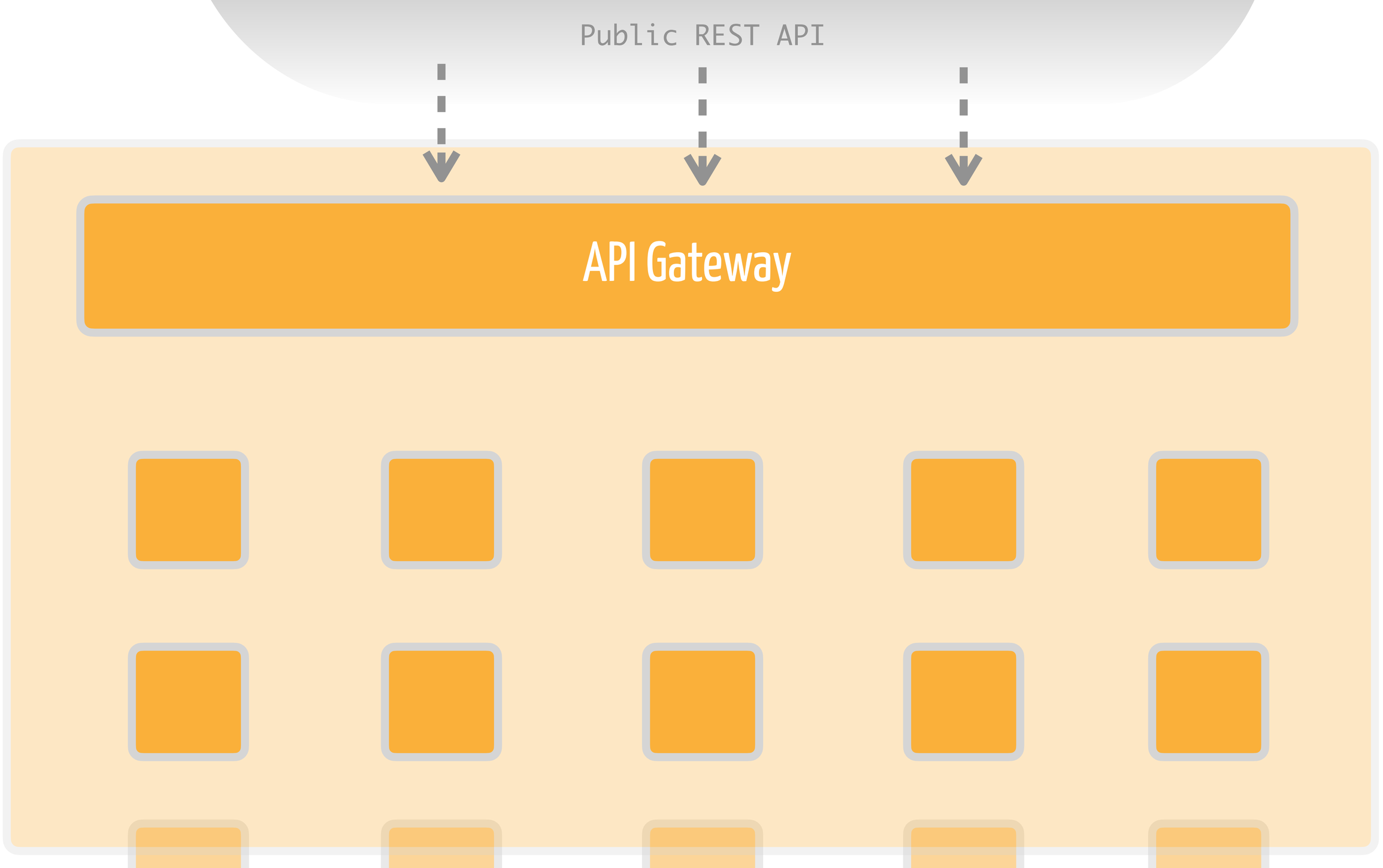


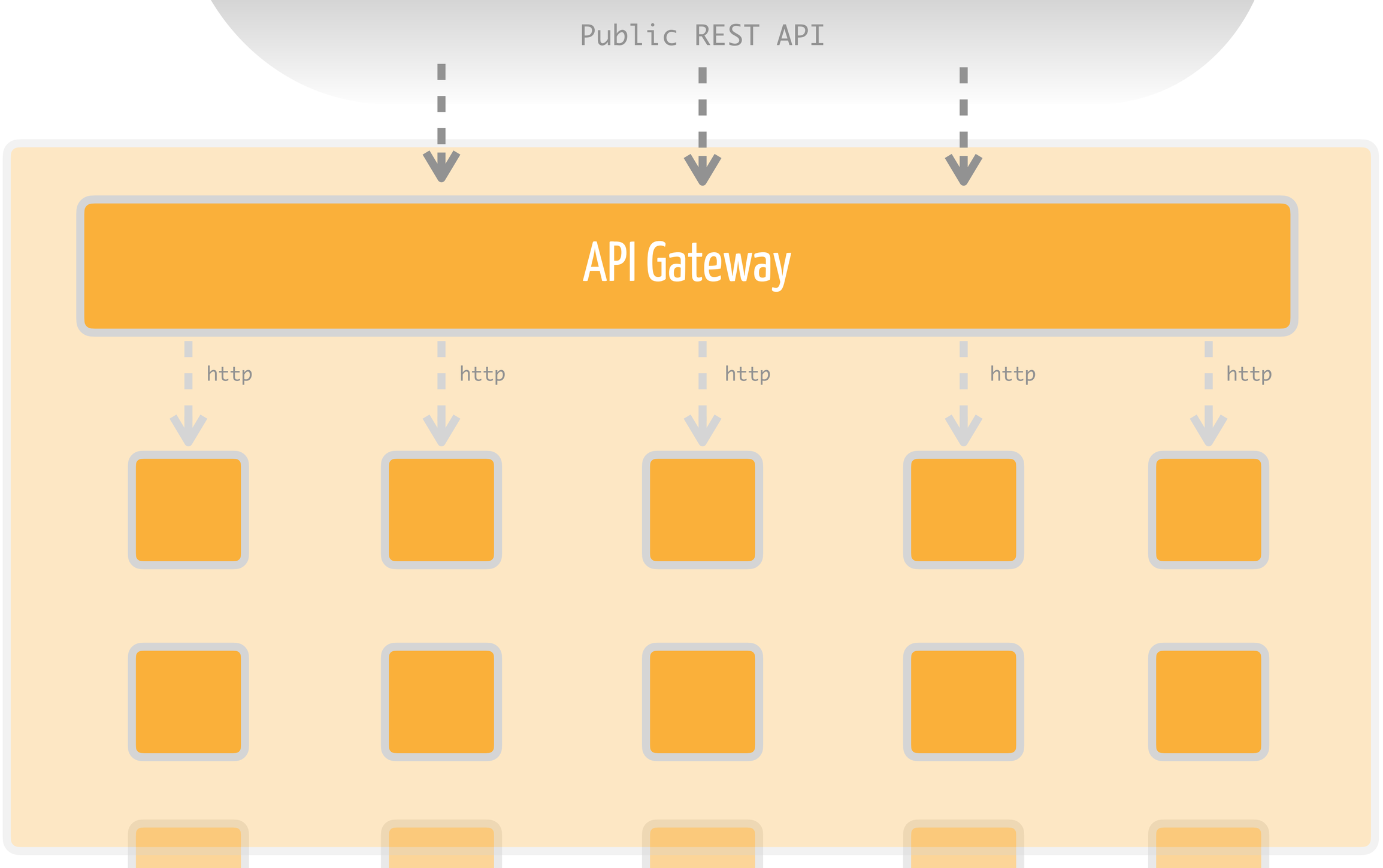


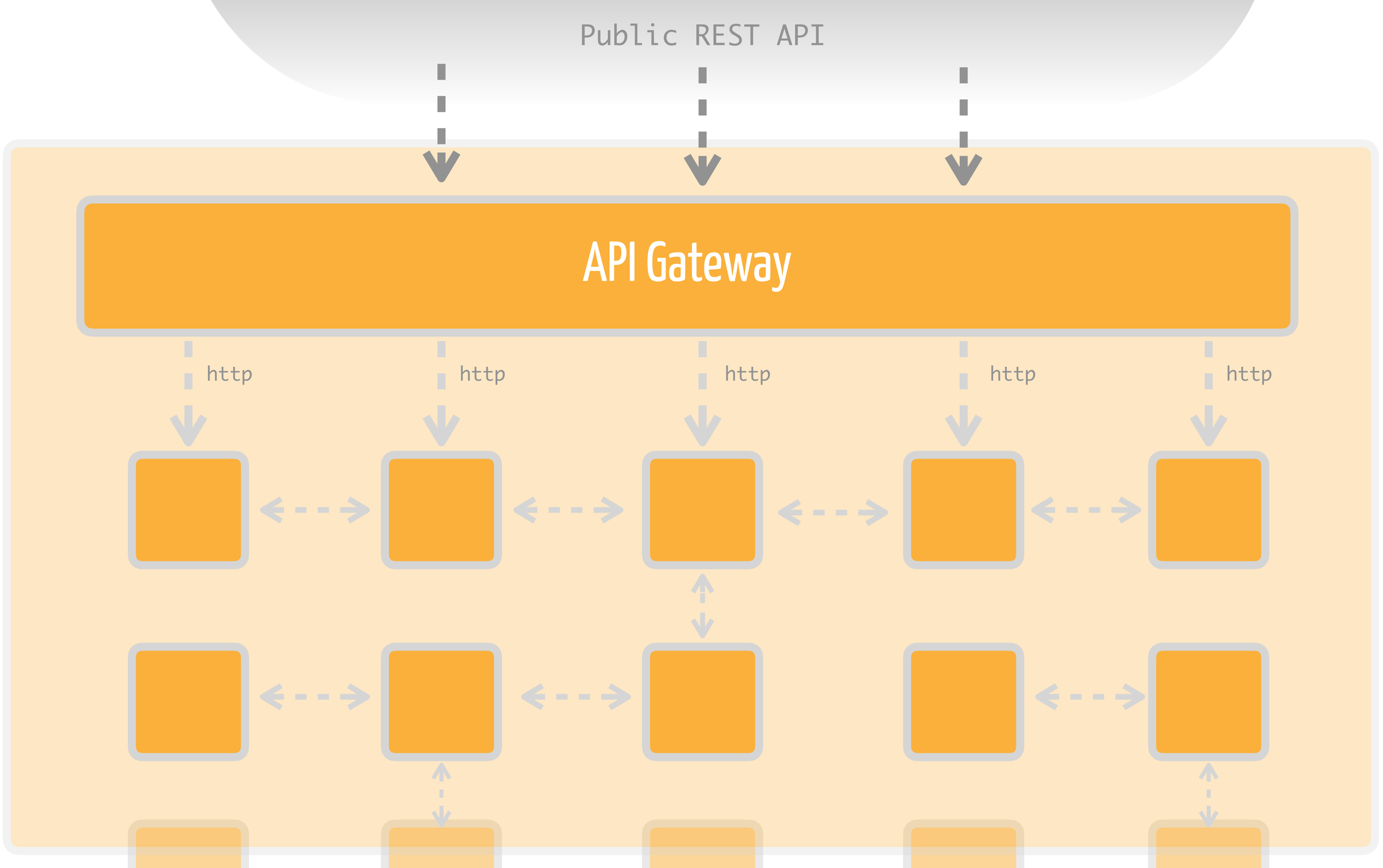
Microservices

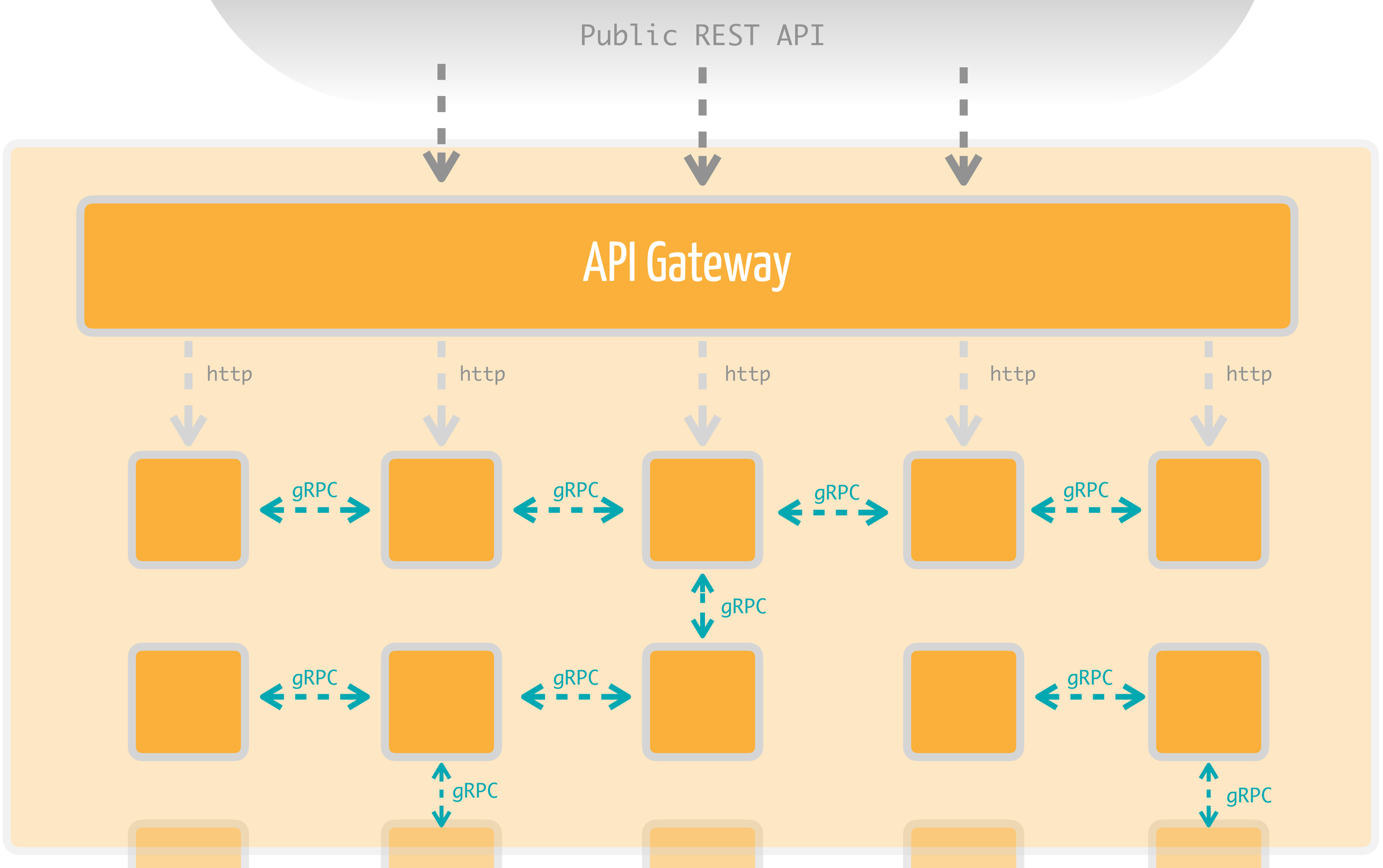


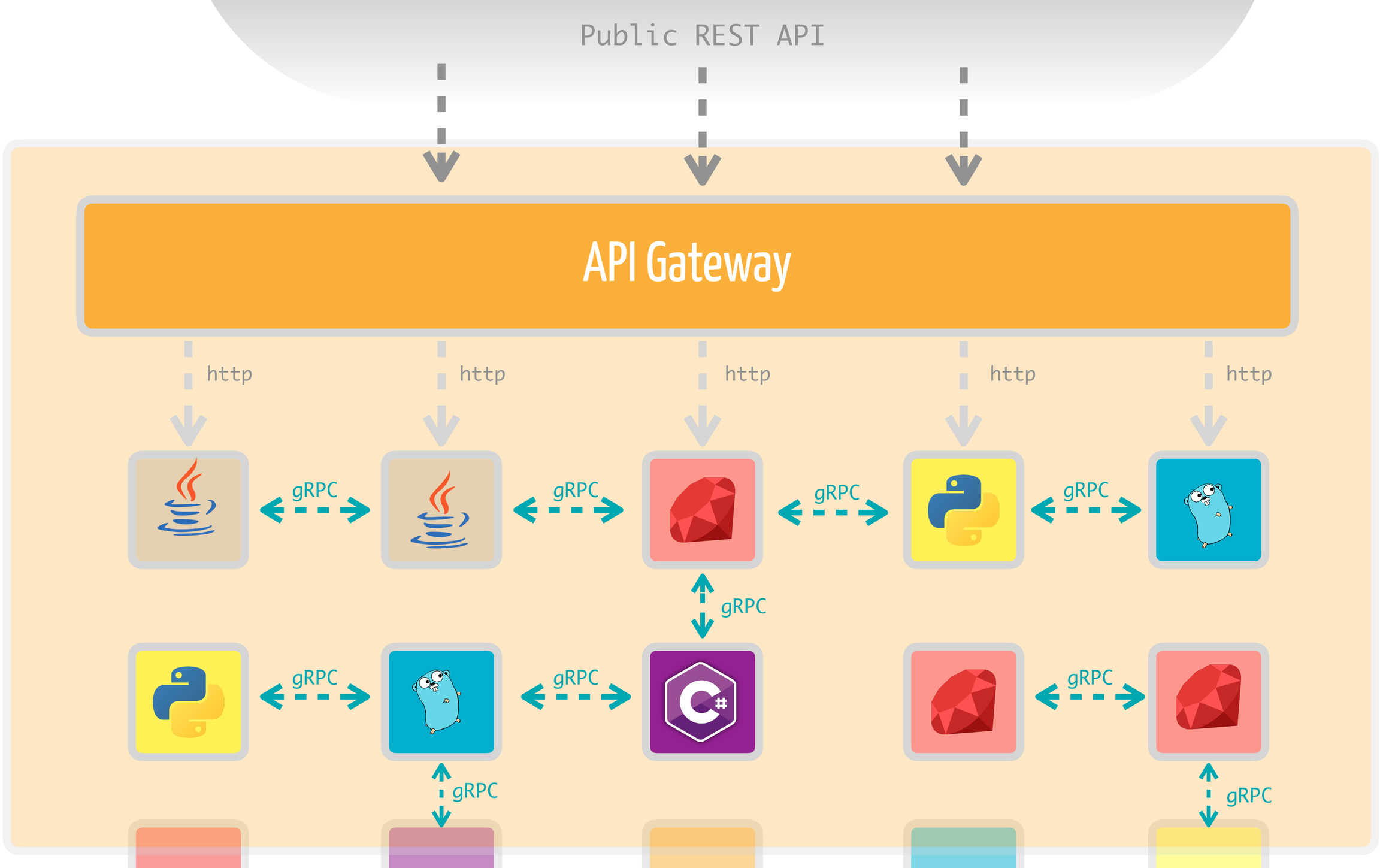

Public REST API

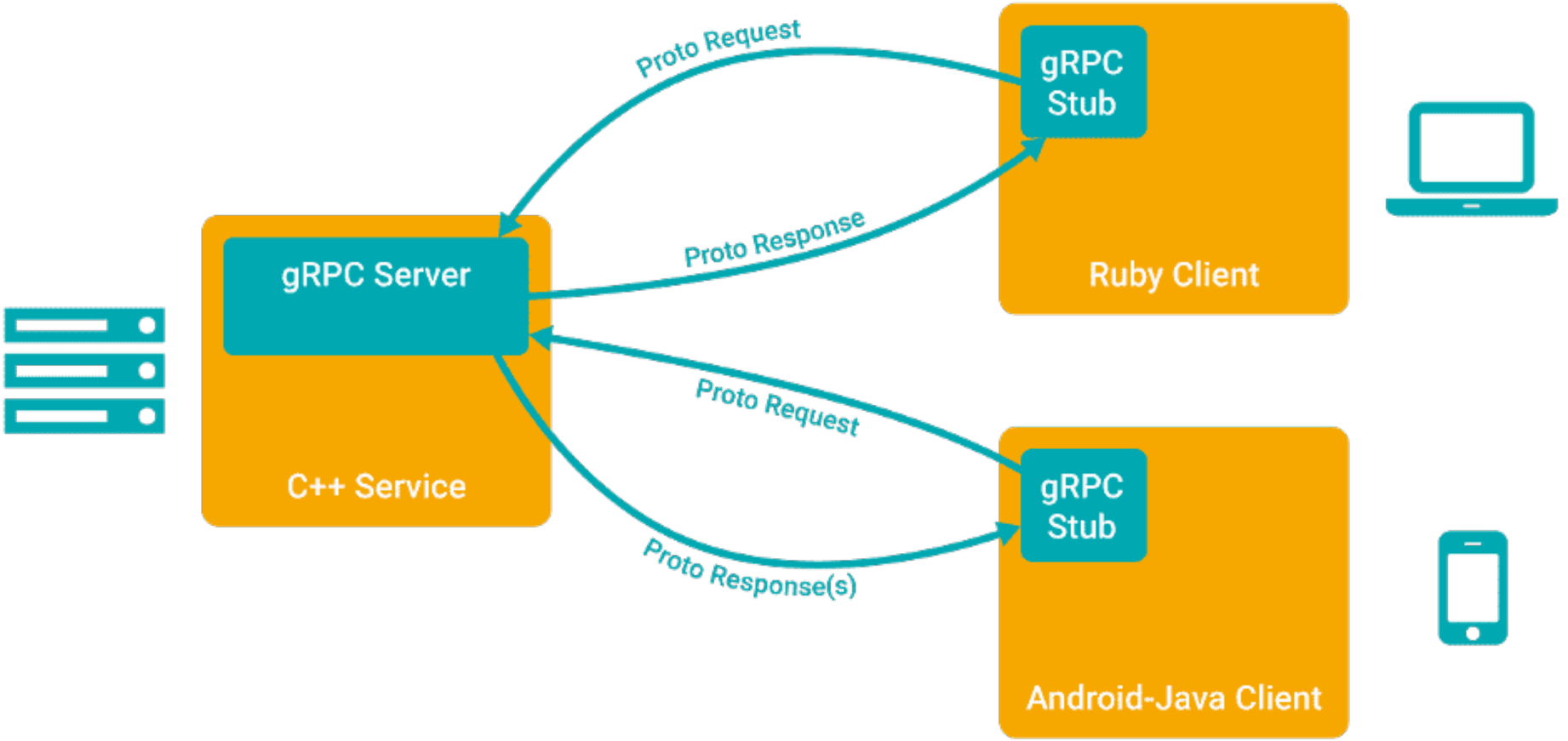

Public REST API
API Gateway


Public REST API
API Gateway
http
http
http
http
http


Public REST API
API Gateway
http
http
http
http
http

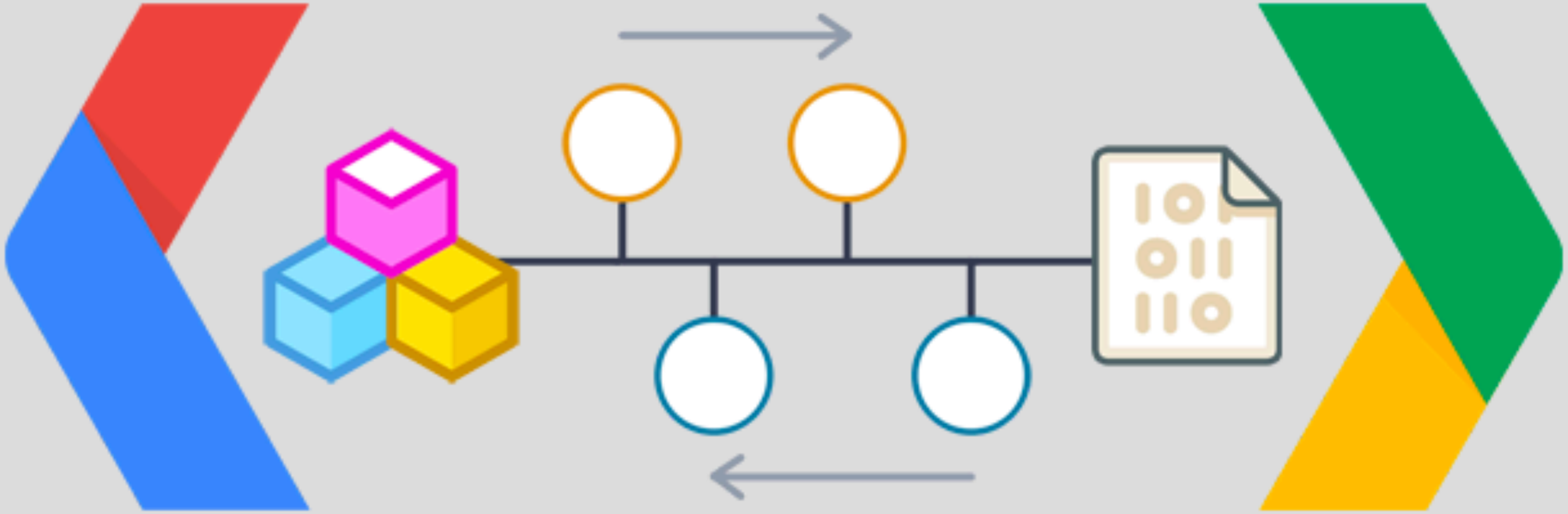

Public REST API
API Gateway
http
http
http
http
http
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC


Public REST API
API Gateway
http
http
http
http
http
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC
gRPC


Proto Request
gRPC Stub
Ruby Client
Proto Response
gRPC Server
C++ Service
Proto Request
gRPC Stub
Android-Java Client
Proto Response(s)


protobuf
Protocol Buffers

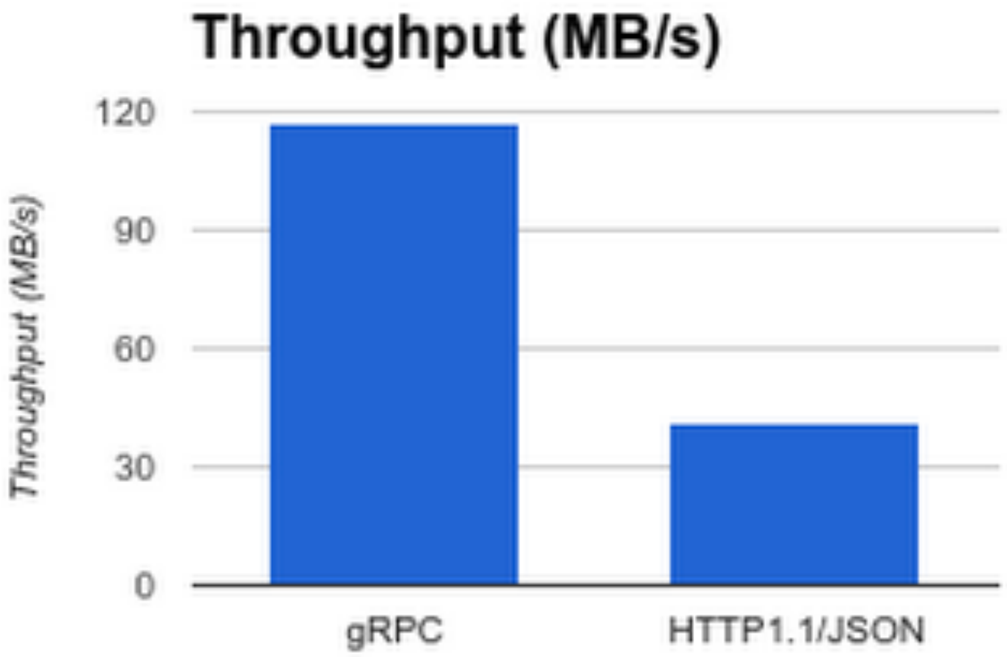
Binary Serialization with



Google Protocol Buffers

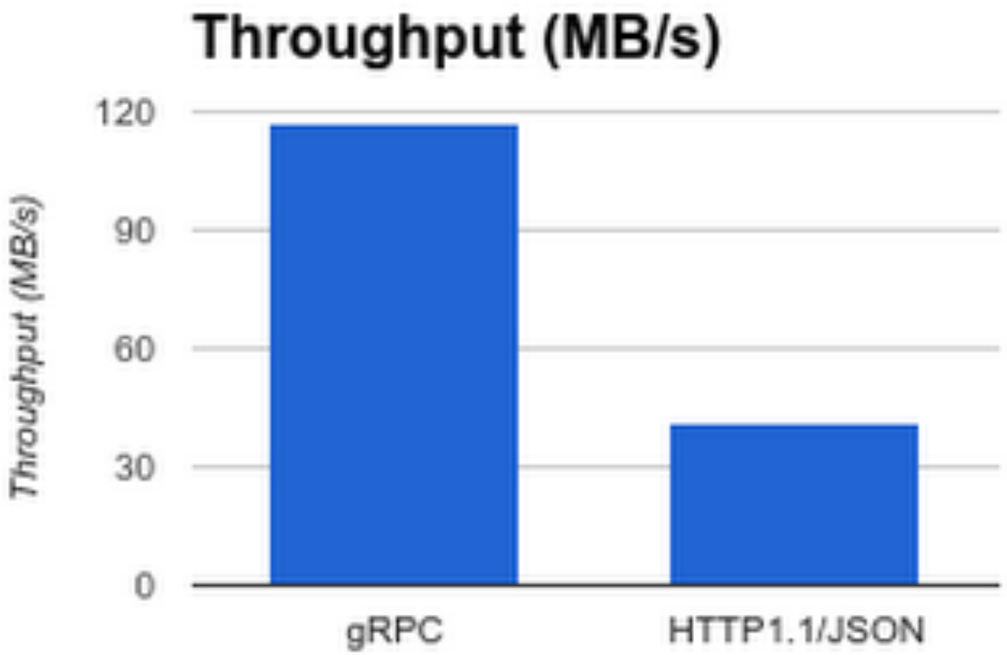
Performance

Performance

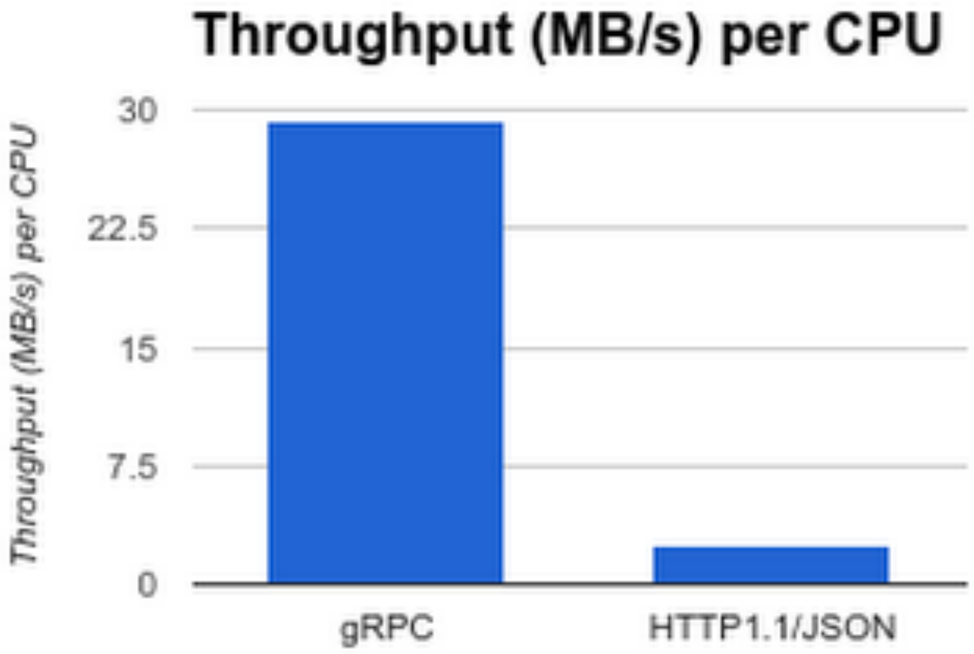


3x increase in throughput

Performance



3x increase in throughput



11x difference per CPU

https://cloud.google.com/blog/products/gcp/announcing-grpc-alpha-for-google-cloud-pubsub

Seu smartphone agradece!

IDL

(Interface Definition Language)

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

Compilando o arquivo .proto

```
$ protoc --proto_path=src --java_out=build/gen src/paymentService.proto
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                          .setCardNumber("1234 5678 8765 4321")
                          .setExpiryDate("03/2028")
                          .setCvc("123")
                          .setAmount(120.2)
                          .setCurrency(Currency.BRL)
                          .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                          .setCardNumber("1234 5678 8765 4321")
                          .setExpiryDate("03/2028")
                          .setCvc("123")
                          .setAmount(120.2)
                          .setCurrency(Currency.BRL)
                          .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                        .setCardNumber("1234 5678 8765 4321")
                        .setExpiryDate("03/2028")
                        .setCvc("123")
                        .setAmount(120.2)
                        .setCurrency(Currency.BRL)
                        .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                         .setCardNumber("1234 5678 8765 4321")
                         .setExpiryDate("03/2028")
                         .setCvc("123")
                         .setAmount(120.2)
                         .setCurrency(Currency.BRL)
                         .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                         .setCardNumber("1234 5678 8765 4321")
                         .setExpiryDate("03/2028")
                         .setCvc("123")
                         .setAmount(120.2)
                         .setCurrency(Currency.BRL)
                         .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                         .setCardNumber("1234 5678 8765 4321")
                         .setExpiryDate("03/2028")
                         .setCvc("123")
                         .setAmount(120.2)
                         .setCurrency(Currency.BRL)
                         .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

```
Channel channel = ManagedChannelBuilder
                    .forAddress("paymentservice.com", 50051)
                    .build();

PaymentServiceBlockingStub paymentService = PaymentServiceGrpc.newBlockingStub(channel);

PaymentRequest request = newBuilder()
                          .setCardNumber("1234 5678 8765 4321")
                          .setExpiryDate("03/2028")
                          .setCvc("123")
                          .setAmount(120.2)
                          .setCurrency(Currency.BRL)
                          .build();

PaymentResponse result = paymentService.processPayment(request);
if (result.getSuccess()) {
    // processa fluxo do pedido...
}
```

No final das contas...

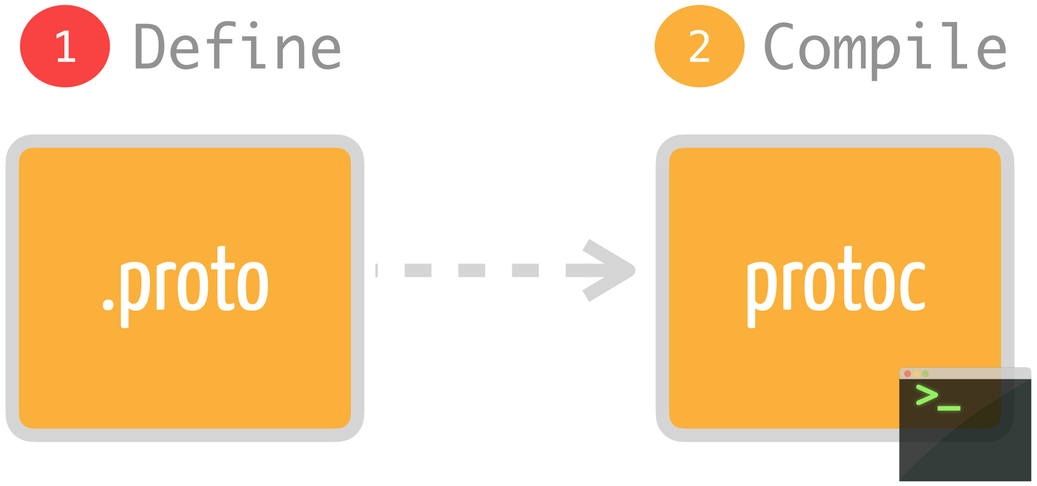
Protobuf Workflow

Protobuf Workflow

1 Define

.proto

Protobuf Workflow



Protobuf Workflow

1 Define

2 Compile

.proto

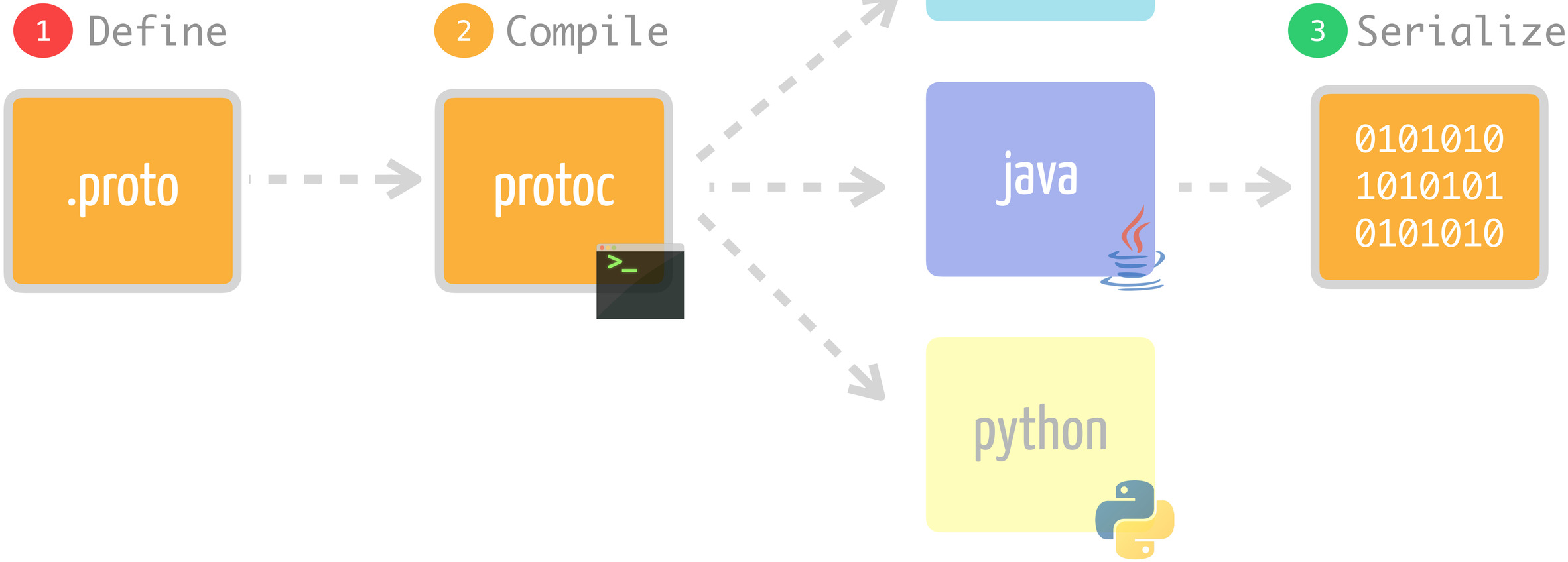
protoc

go

java

python

Protobuf Workflow

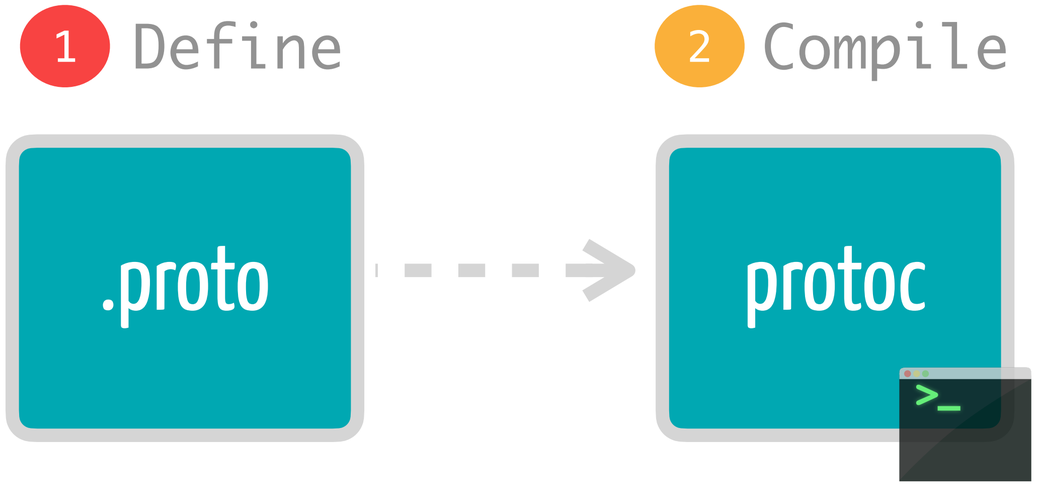


gRPC Workflow

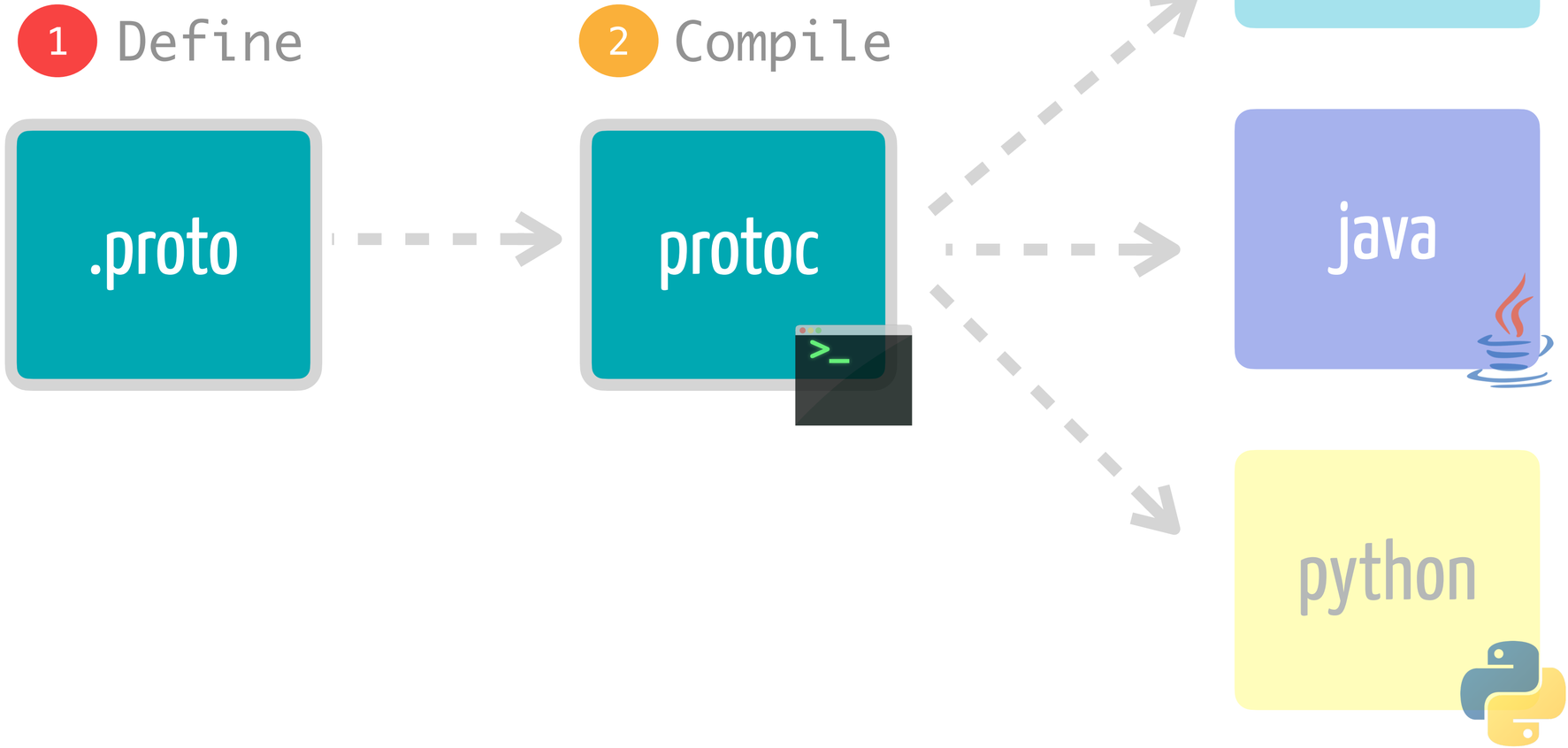
gRPC Workflow

1 Define

gRPC Workflow



gRPC Workflow



gRPC Workflow

1 Define

.proto

2 Compile

protoc

go

java

python

3 Implement

server

client

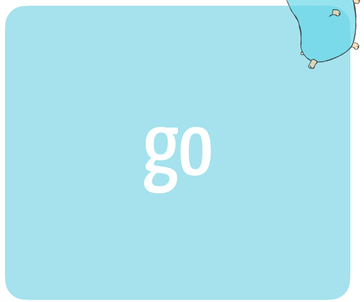
gRPC Workflow

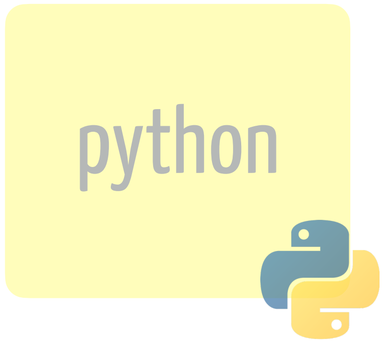
1 Define

.proto

2 Compile

protoc





3 Implement

server

client

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
}

message PaymentRequest {
    enum Currency {
        BRL = 0;
        USD = 1;
    }

    string cardNumber = 1;
    string expiryDate = 2;
    string cvc        = 3;
    double amount     = 4;
    Currency currency = 5;
}

message PaymentResponse {
    bool success = 1;
    string txId  = 2;
}
```


```
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```


```java
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @Override
    public void processPayment(PaymentRequest request,
                               StreamObserver<PaymentResponse> responseObserver) {

        // lógica de negócio
        Payment payment = toPayment(request); // converte para modelo de domínio
        UUID transactionId = new PaymentService().process(payment);

        // envia resposta para o client
        responseObserver.onNext(PaymentResponse.newBuilder()
                                    .setTxId(transactionId.toString())
                                    .setSuccess(true)
                                    .build());
        responseObserver.onCompleted();
    }
}
```

Padrões de comunicação

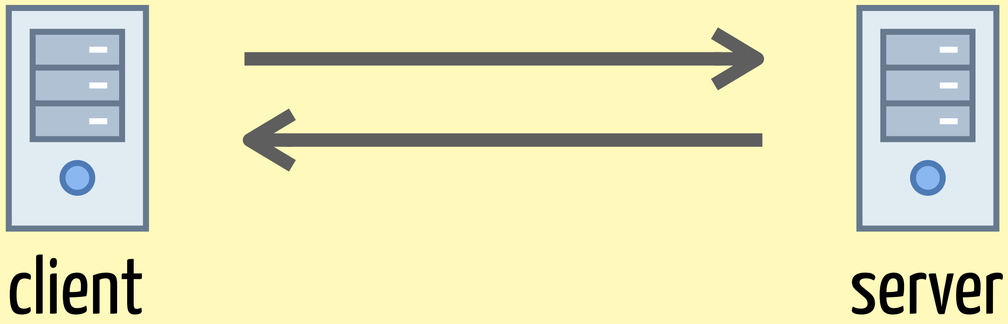
1

2

3

4

Simple RPC (Unary RPC)



2

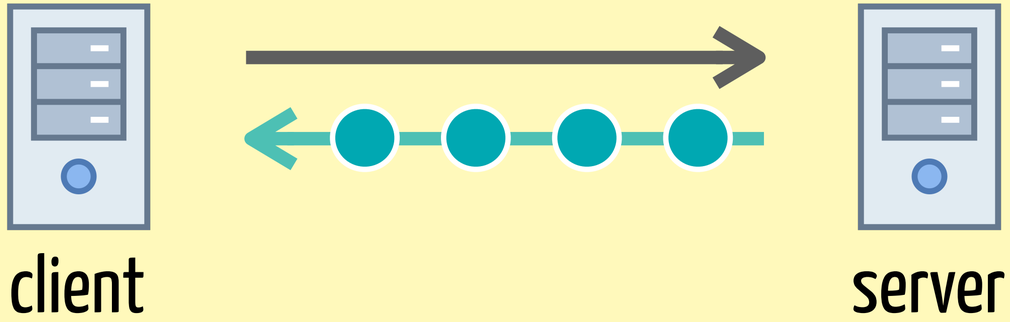
3

4

Simple RPC (Unary RPC)

3

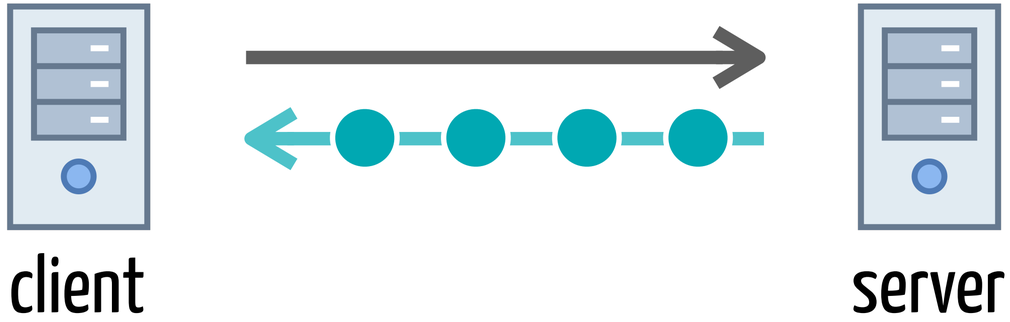
Server-side Streaming RPC

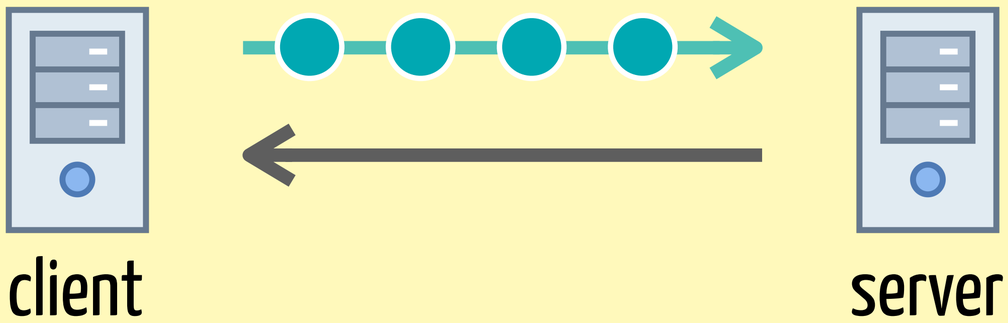


4

Simple RPC (Unary RPC)

Server-side Streaming RPC



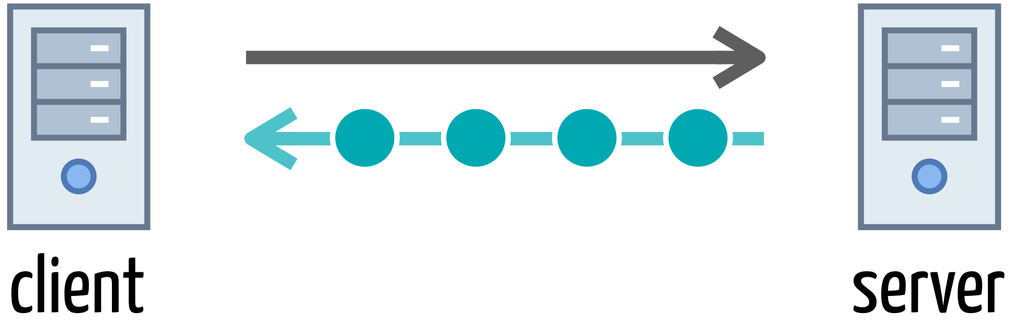


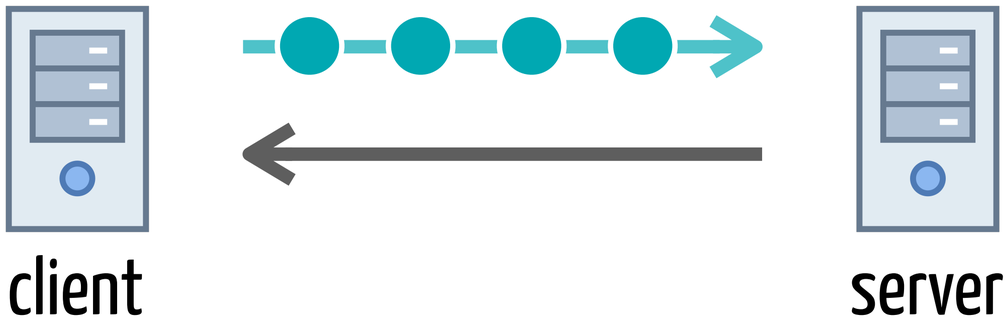
Client-side Streaming RPC

4

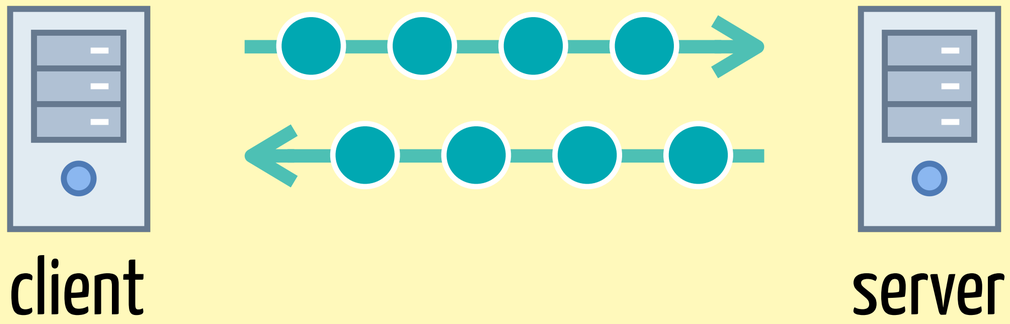
Simple RPC (Unary RPC)

Server-side Streaming RPC





Client-side Streaming RPC



Bidirectional Streaming RPC

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}




}
```

paymentService.proto

Simple RPC (Unary RPC)

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}

}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}

  rpc listenToTransactions(TransactionRequest) returns (stream TransactionResponse) {}



}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest)

  rpc listenToTransactions(TransactionRequest) returns (stream TransactionResponse) {}



}
```

Server-side Streaming RPC

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}

  rpc listenToTransactions(TransactionRequest) returns (stream TransactionResponse) {}

  rpc processPaymentInBatch(stream PaymentInBatchRequest) returns (PaymentInBatchResponse) {}


}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc pr                                                    {}

  rpc li                                                    TransactionResponse) {}

  rpc processPaymentInBatch(stream PaymentInBatchRequest) returns (PaymentInBatchResponse) {}


}
```

Client-side Streaming RPC

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}

  rpc listenToTransactions(TransactionRequest) returns (stream TransactionResponse) {}

  rpc processPaymentInBatch(stream PaymentInBatchRequest) returns (PaymentInBatchResponse) {}

  rpc validateCreditCard(stream CreditCardRequest) returns (stream ValidateCCResponse) {}

}
```

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}

  rpc listenToTra                                          ponse) {}

  rpc processPaym                                          atchResponse) {}

  rpc validateCreditCard(stream CreditCardRequest) returns (stream ValidateCCResponse) {}

}
```

Bidirectional Streaming RPC

paymentService.proto

```
syntax = "proto3";

service PaymentService {

  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}

  rpc listenToTransactions(TransactionRequest) returns (stream TransactionResponse) {}

  rpc processPaymentInBatch(stream PaymentInBatchRequest) returns (PaymentInBatchResponse) {}

  rpc validateCreditCard(stream CreditCardRequest) returns (stream CreditCardResponse) {}

}
```

gRPC Workflow

1 Define

.proto

2 Compile

protoc

go

java

python

3 Implement

server

client

CONCLUINDO

Microservices

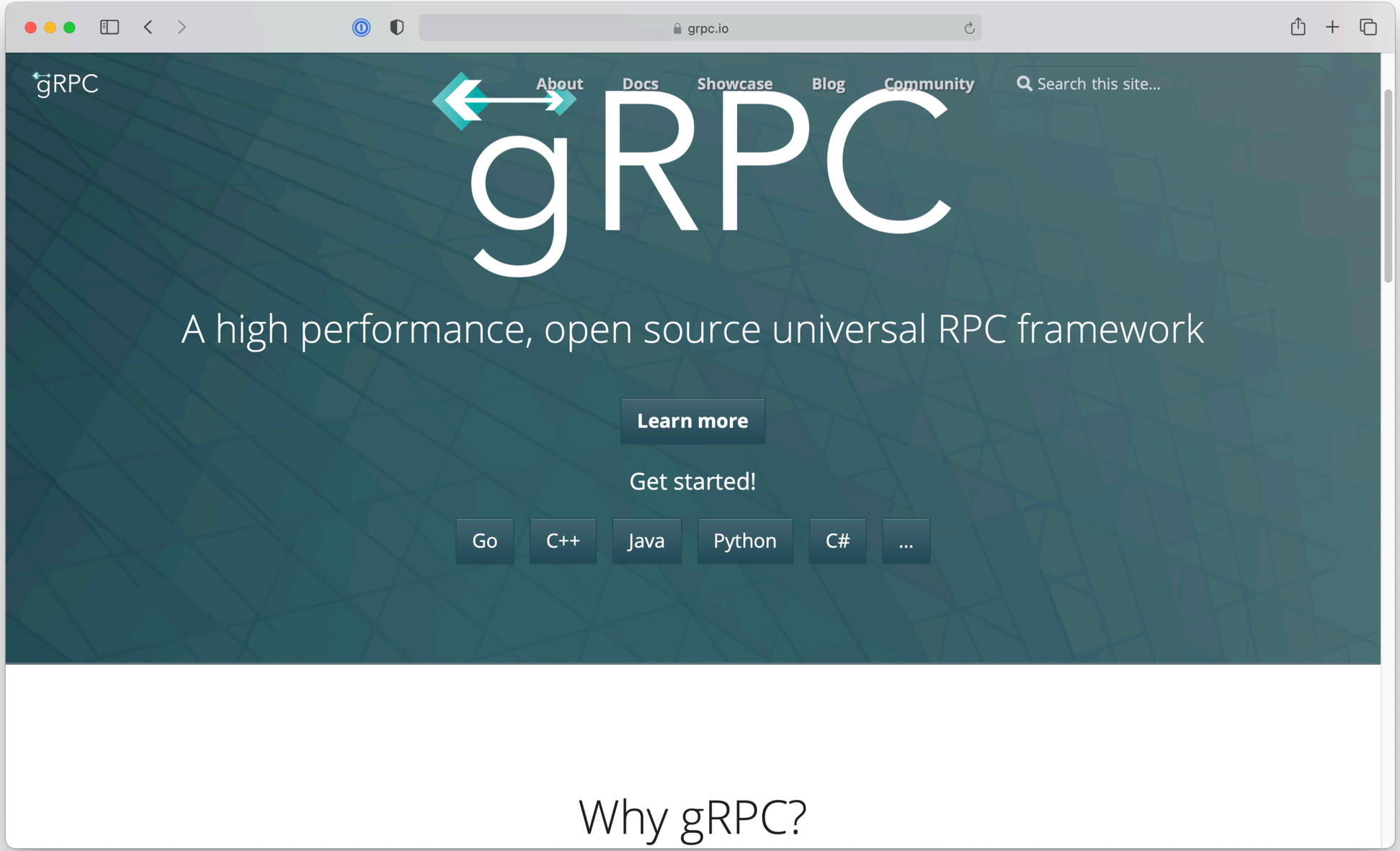
Microservices

Mobile

Microservices

Mobile

Realtime



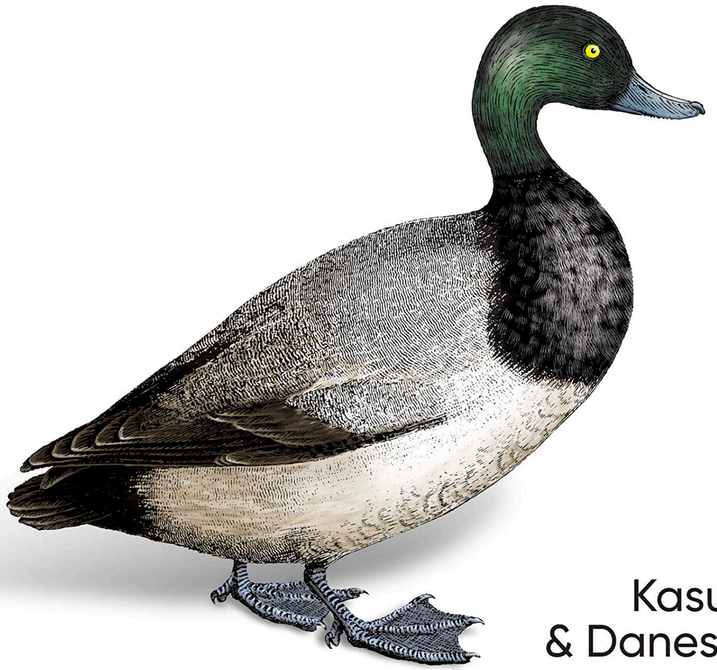
Why gRPC?

https://grpc.io/

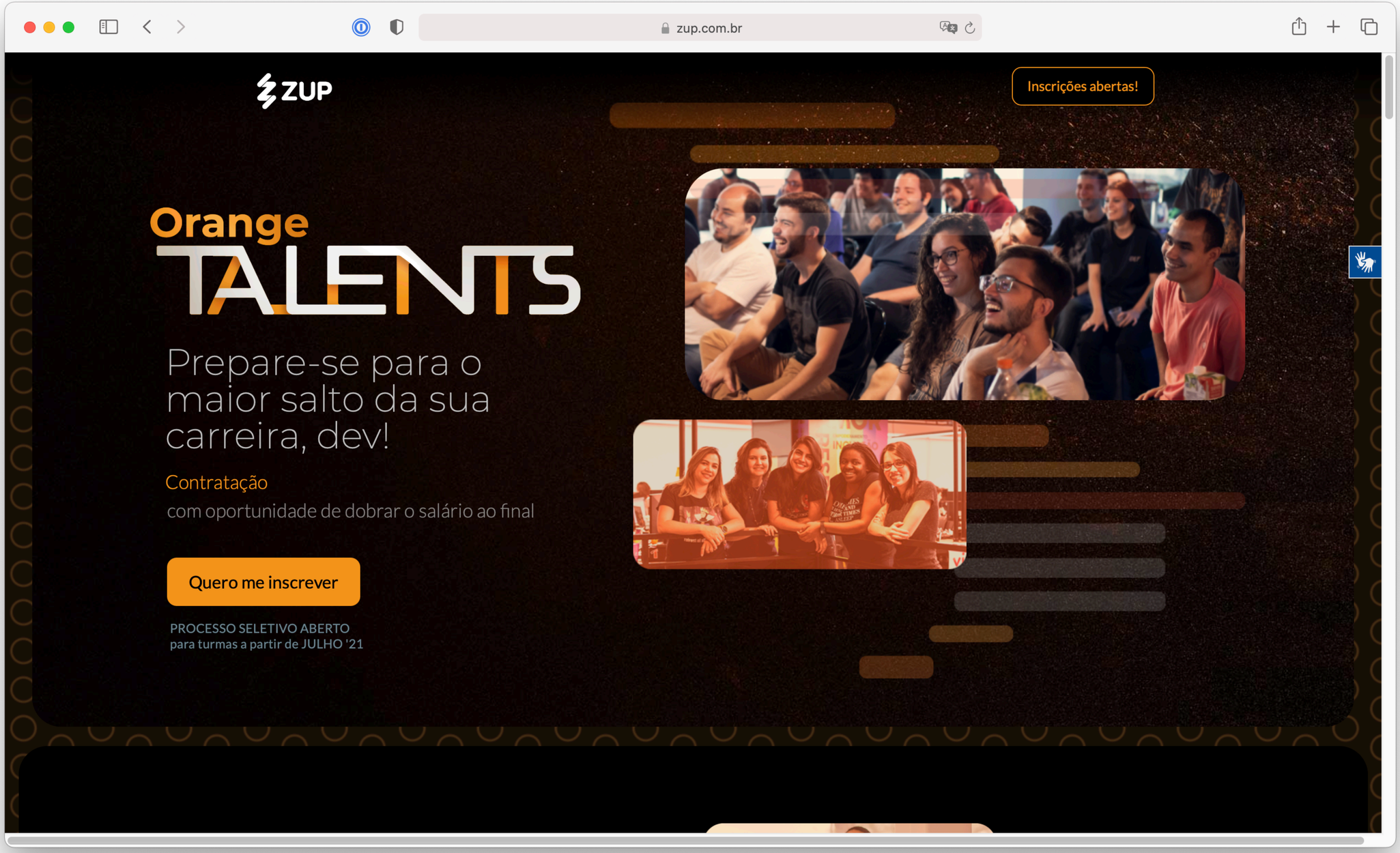
O'REILLY®

gRPC Up & Running

Building Cloud Native Applications with Go and Java for Docker and Kubernetes



Kasun Indrasiri
& Danesh Kuruppu



https://www.zup.com.br/orange-talents

Rafael Ponte
@rponte