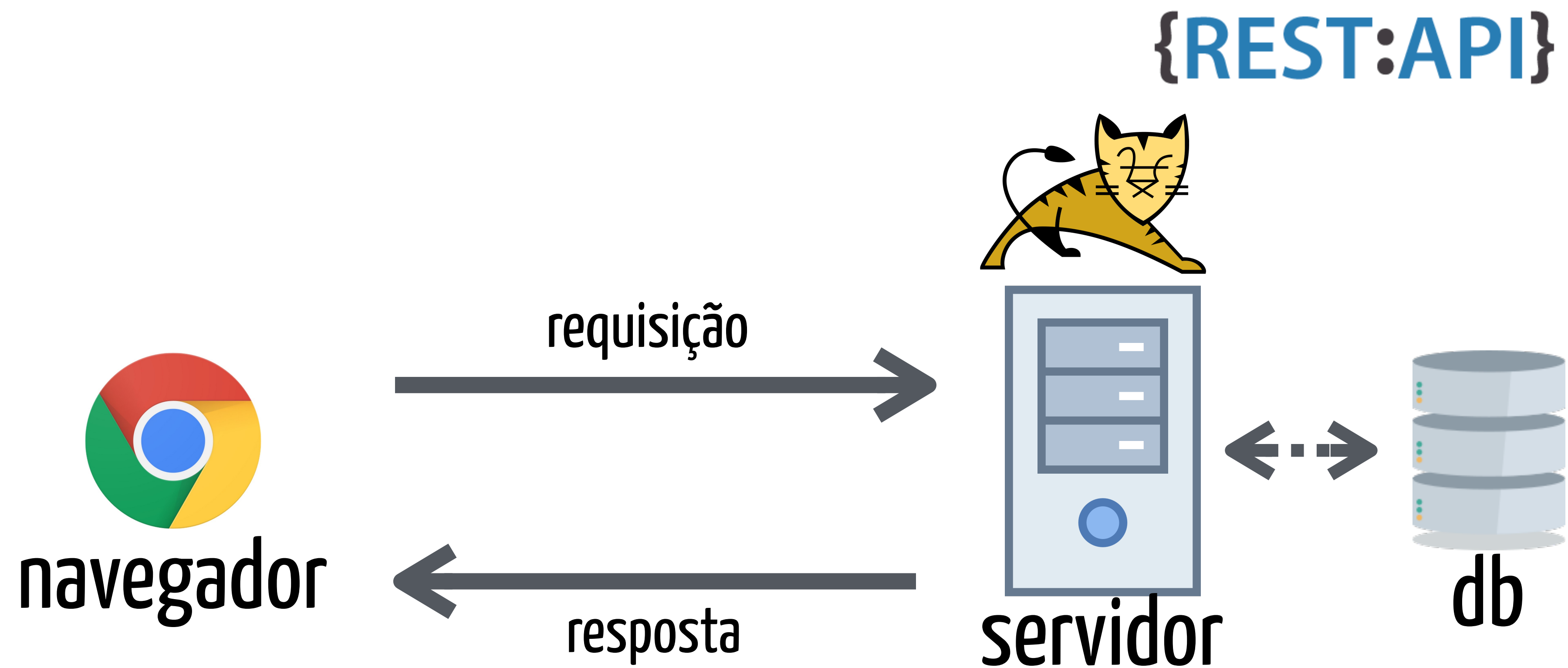


The logo for gRPC features the text "gRPC" in a bold, teal, sans-serif font. The letter "g" is stylized with an upward-pointing arrow on its left side, and the letter "C" is stylized with a downward-pointing arrow on its right side. The entire logo is set against a light blue, horizontally-oriented oval background.

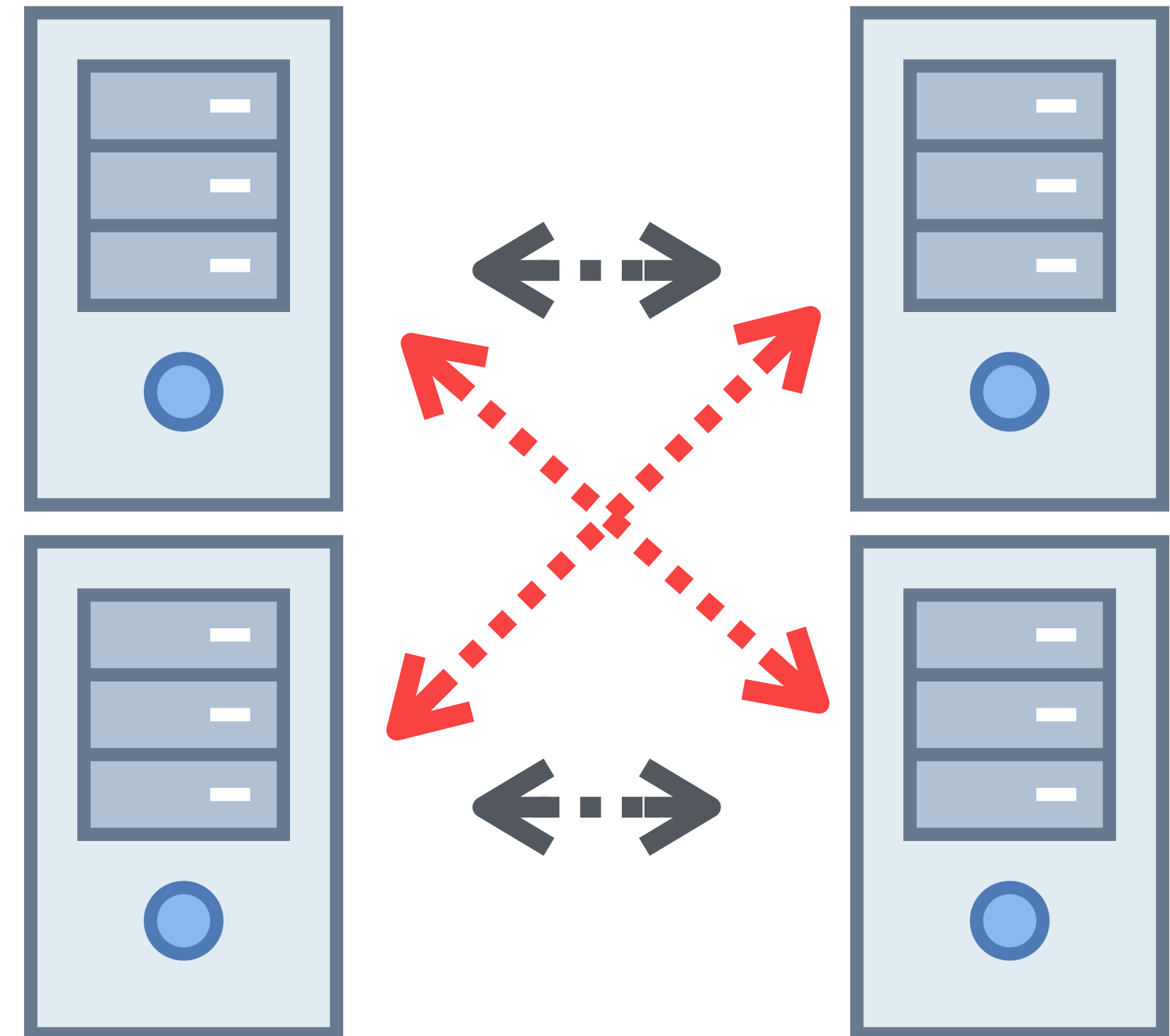
gRPC

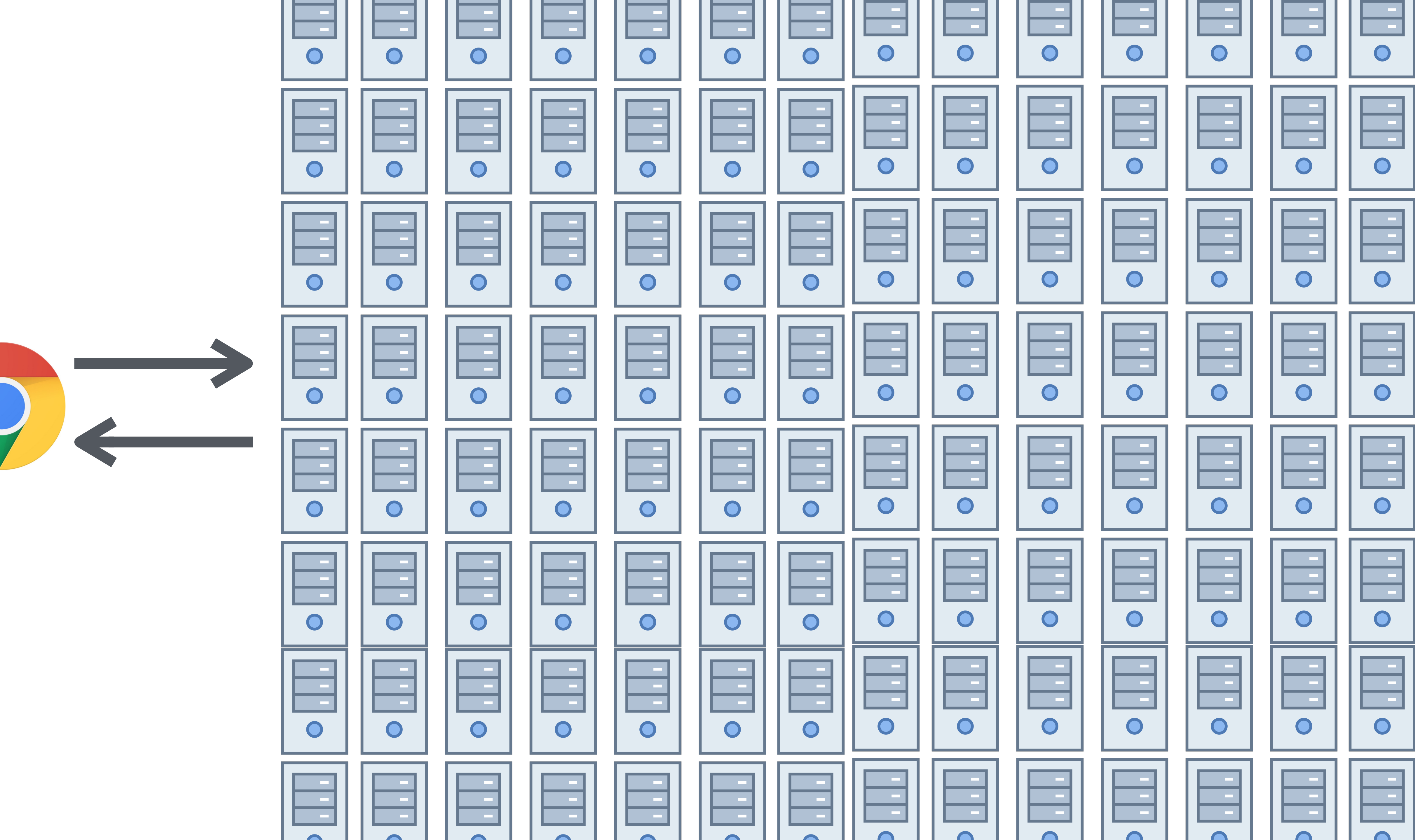
**um framework moderno e performático para
comunicação entre microsserviços**



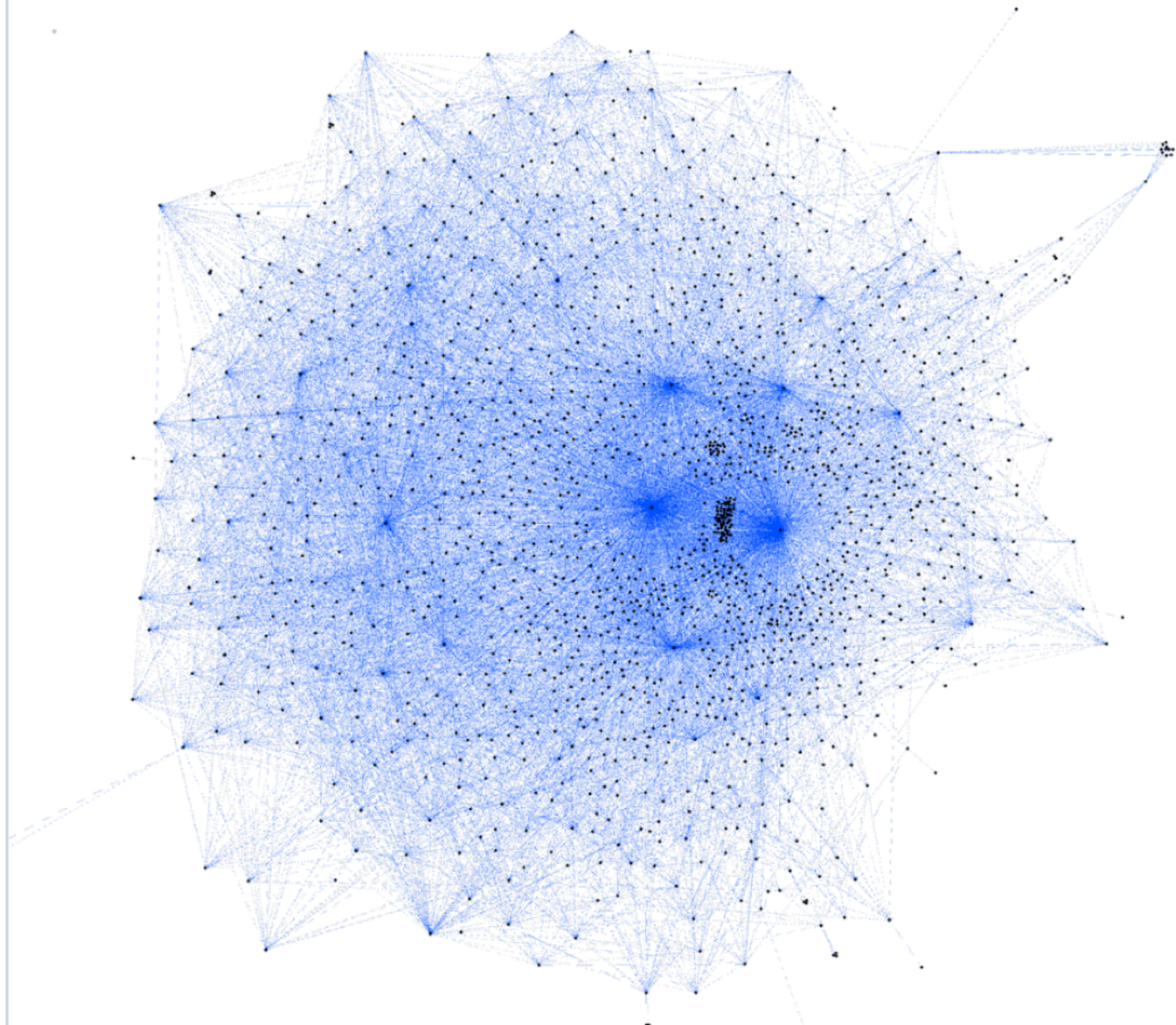


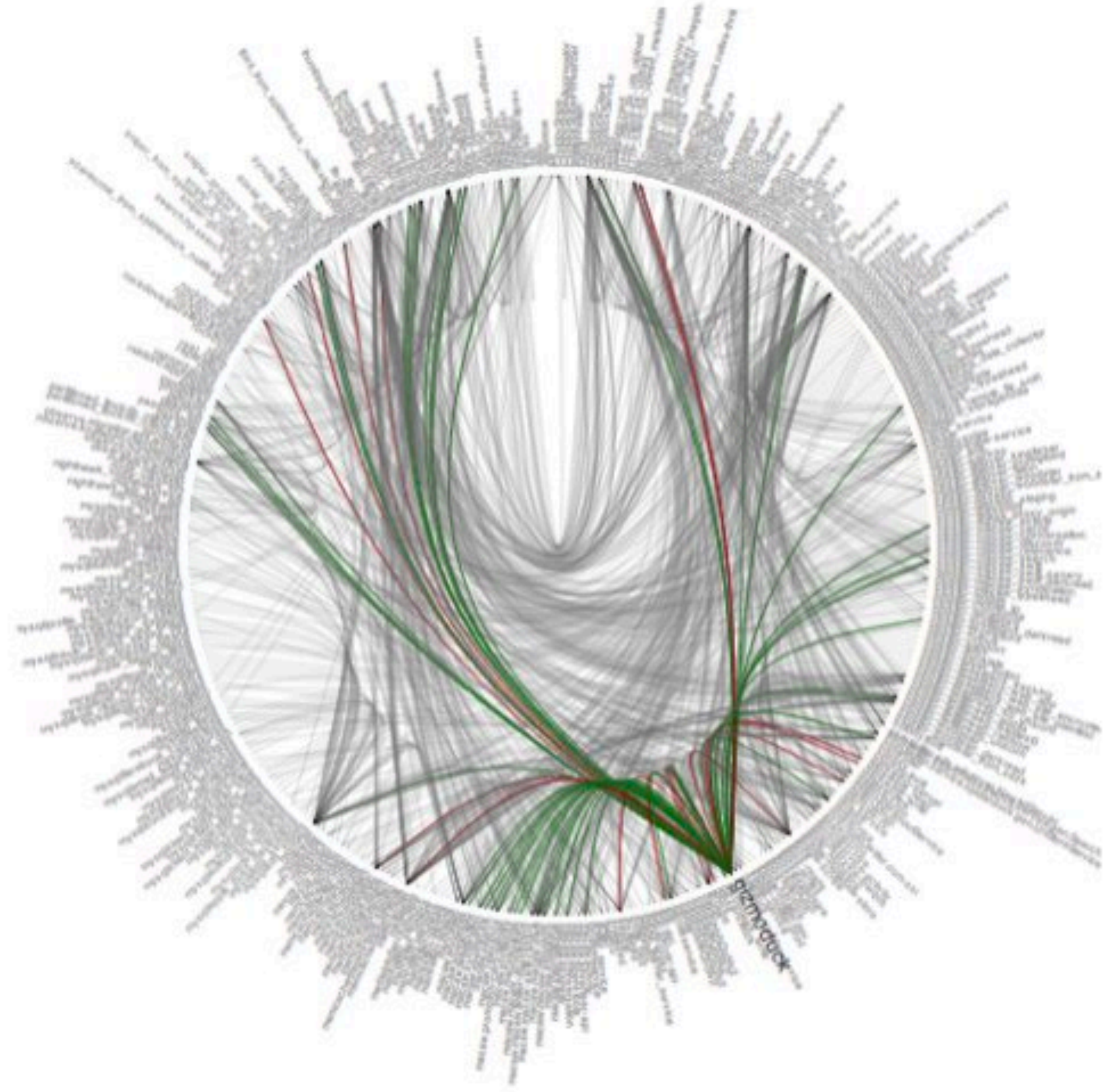
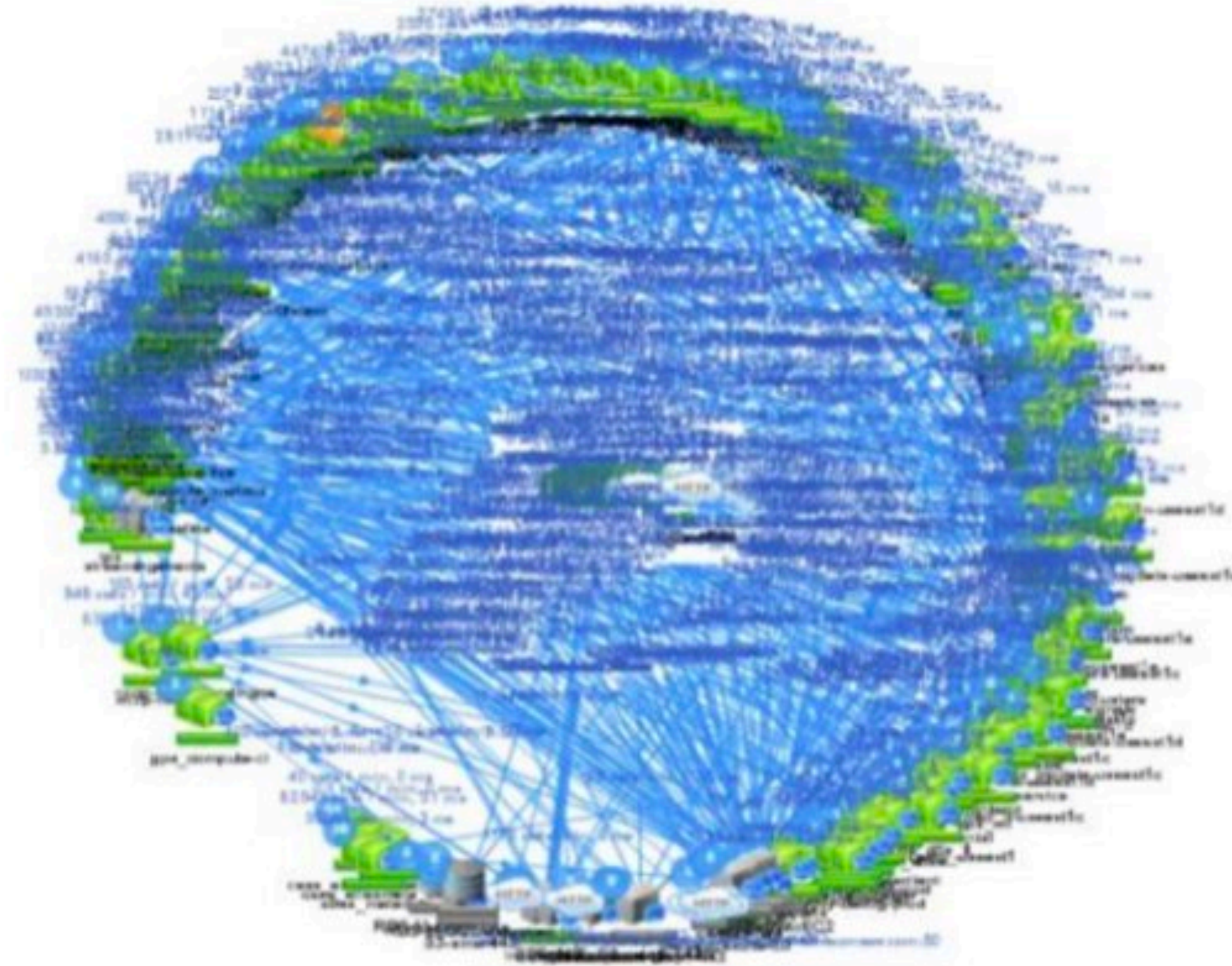
{REST:API}





Pode piorar...





NETFLIX





GRPC



a





Rafael Ponte

 @rponte



A diagram of the Sun-Earth system. On the left is a large, glowing orange-yellow sphere representing the Sun. In the center is a small grey sphere representing a satellite in orbit. On the right is a blue and green sphere representing Earth. The background is a dark blue space filled with numerous small white stars. The labels 'SOL', 'FORTALEZA', and 'TERRA' are overlaid on the image in white capital letters.

SOL

FORTALEZA

TERRA

Fortaleza - Terra do Sol



devs cansados

gRPC

gRPC Remote Procedure Call framework

The image features the acronym 'gRPC' in a large, bold, sans-serif font. The 'g' is light blue, while 'RPC' is a darker teal. A light blue arrow on the left points upwards and to the right, and another on the right points downwards and to the left, framing the text.

gRPC

Remote Procedure Call



Loja Virtual

Pagamento Online

Online Shop

```
graph TD; A[Online Shop] --> B[Payment Service];
```

The diagram illustrates the online payment process. It features a central title 'Pagamento Online' at the top. Below the title, on the left, is a pink rectangular box labeled 'Online Shop'. A thick black vertical line extends downwards from the bottom center of this box. On the right, there is another pink rectangular box labeled 'Payment Service'. A thick black vertical line extends downwards from the bottom center of this box. The two vertical lines are parallel and represent the flow of the payment process from the online shop to the payment service.

Payment Service

Pagamento Online

Online Shop

Payment Service

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



Pagamento Online

Online Shop

Payment Service

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

SUCCESS

Pagamento Online

Online Shop

Payment Service

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

SUCCESS

cobre R\$16.99 do cartão de crédito 9876...

Pagamento Online

Online Shop

Payment Service

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

SUCCESS

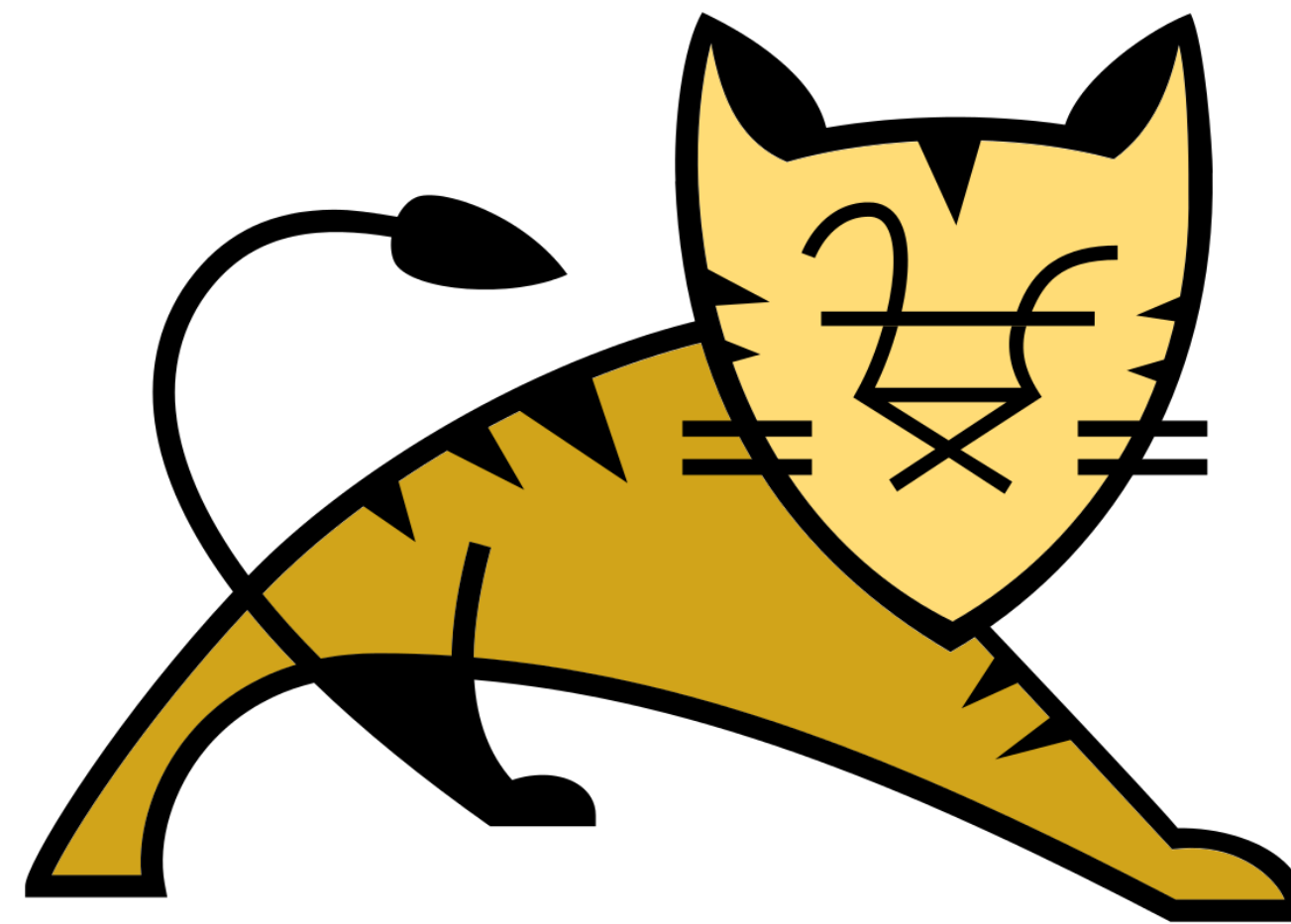
cobre R\$16.99 do cartão de crédito 9876...

ERROR

Como
implementamos
essa lógica?









Spring Boot App



Spring Boot App



Spring Boot App

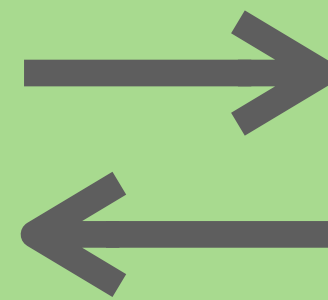
Client
(Online Shop)

Service
(Payment Service)



Spring Boot App

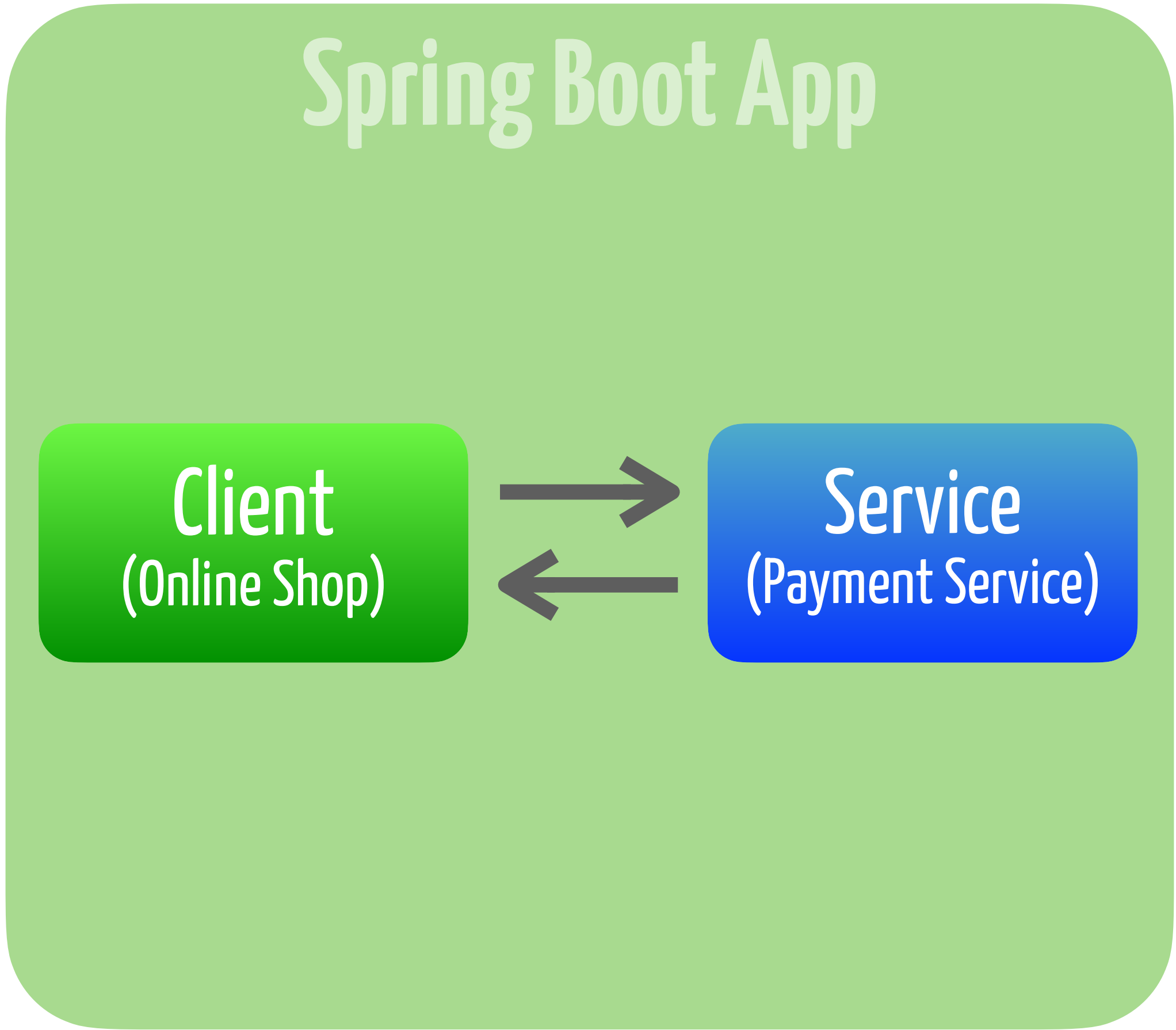
Client
(Online Shop)



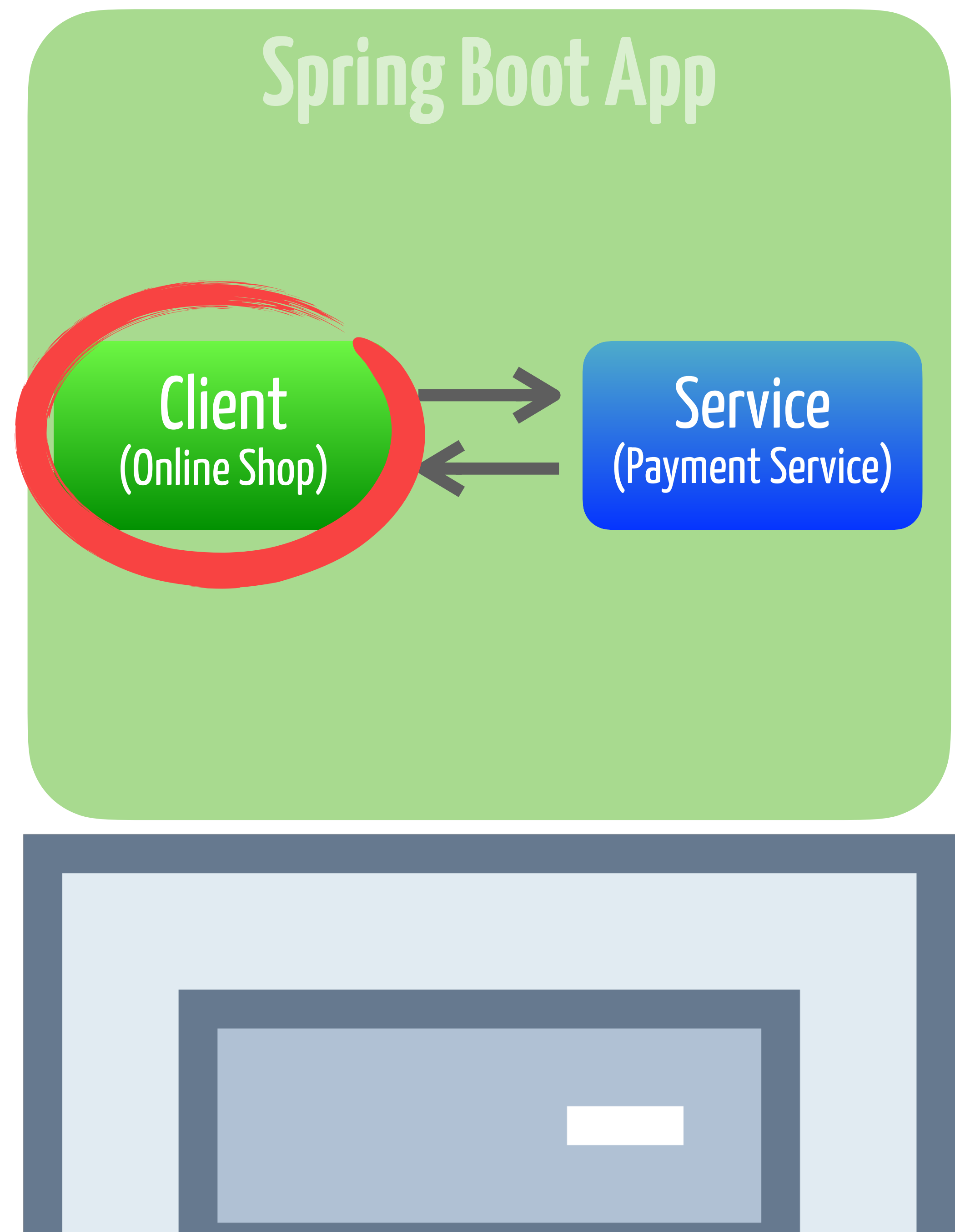
Service
(Payment Service)



Local Calls



Local Calls




```
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...  
}
```



```
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

Online Shop

Payment Service

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

SUCCESS

Pagamento Online



Online Shop

Payment Service

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

SUCCESS

Pagamento Online



Online Shop



Payment Service

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)



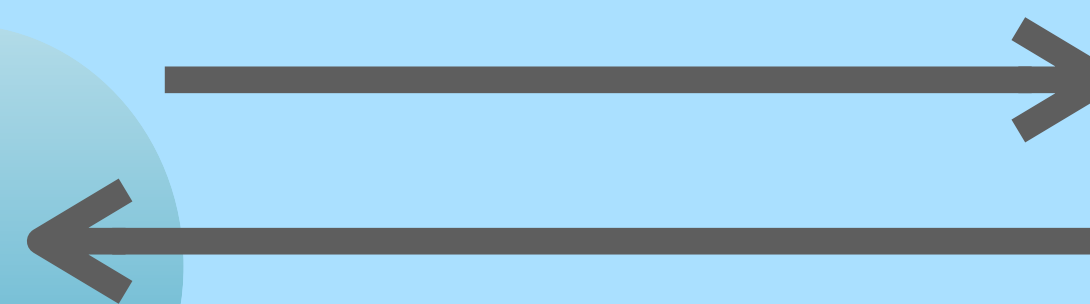
Spring Boot App

Client
(Online Shop)

Internet

Spring Boot App

Service
(Payment Service)



Remote Calls

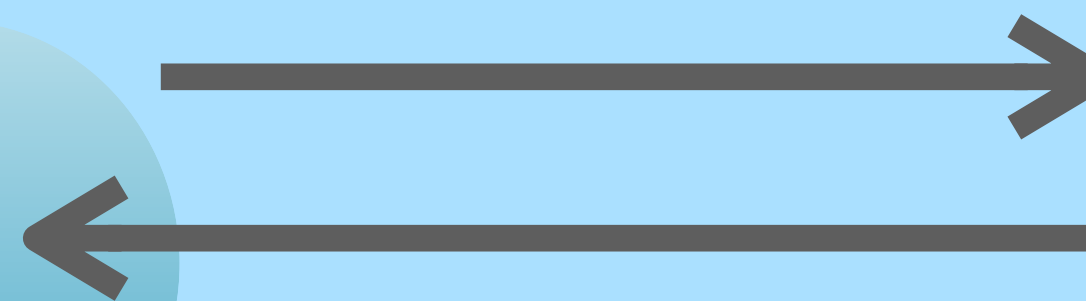
Spring Boot App

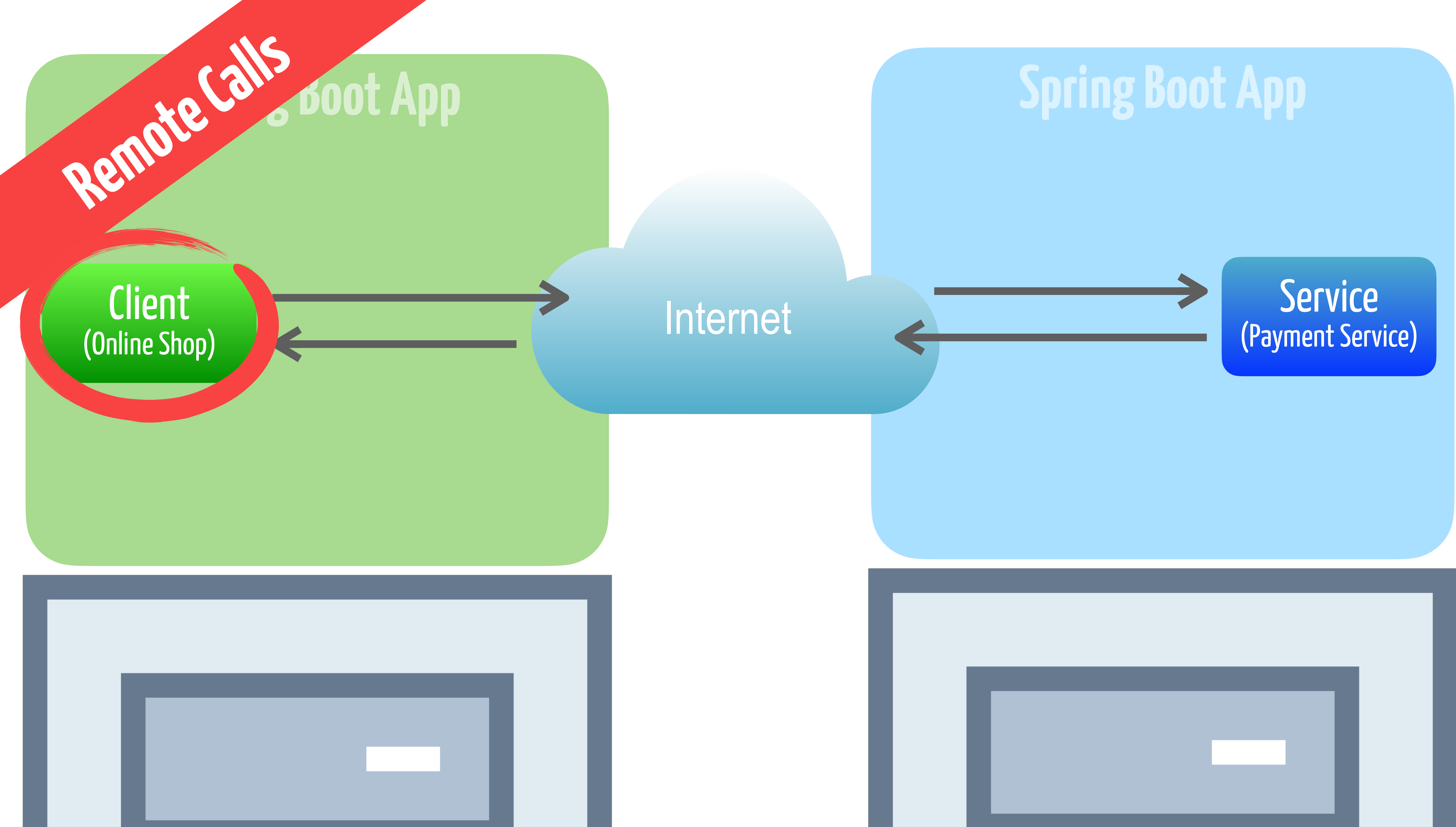
Client
(Online Shop)

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

Online Shop

RPC client

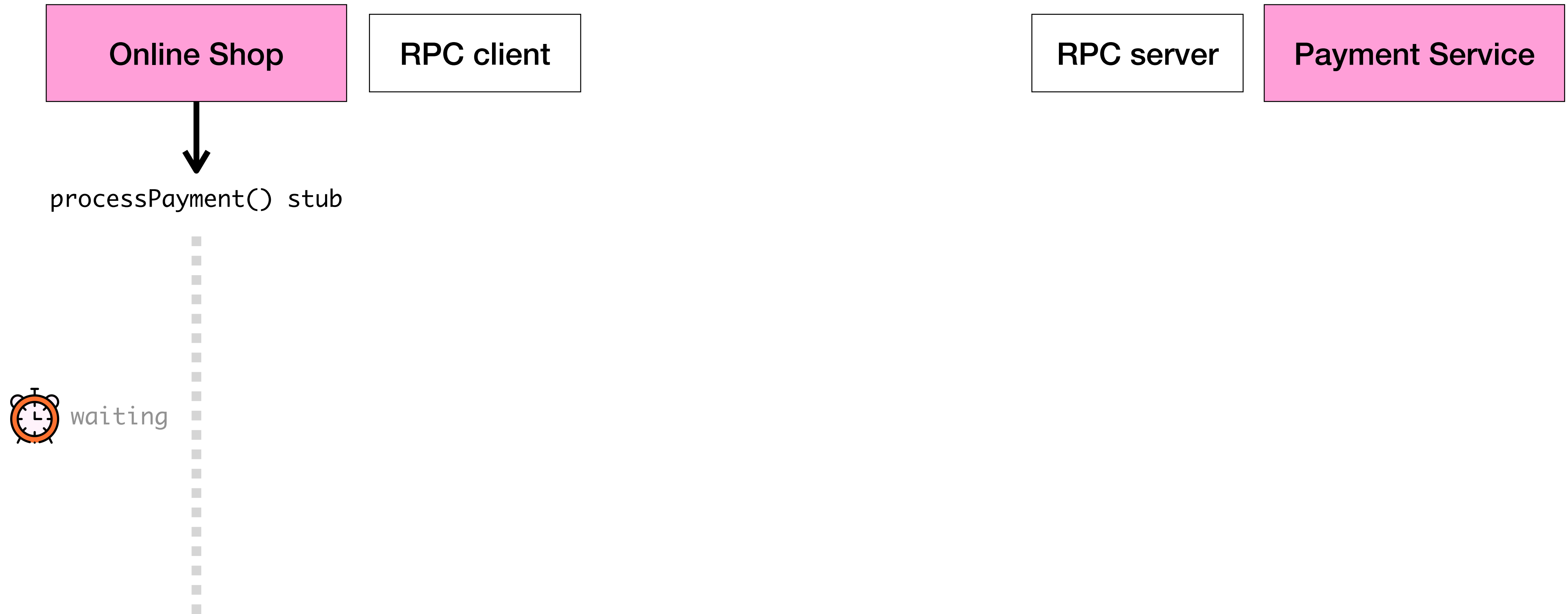
RPC server

Payment Service

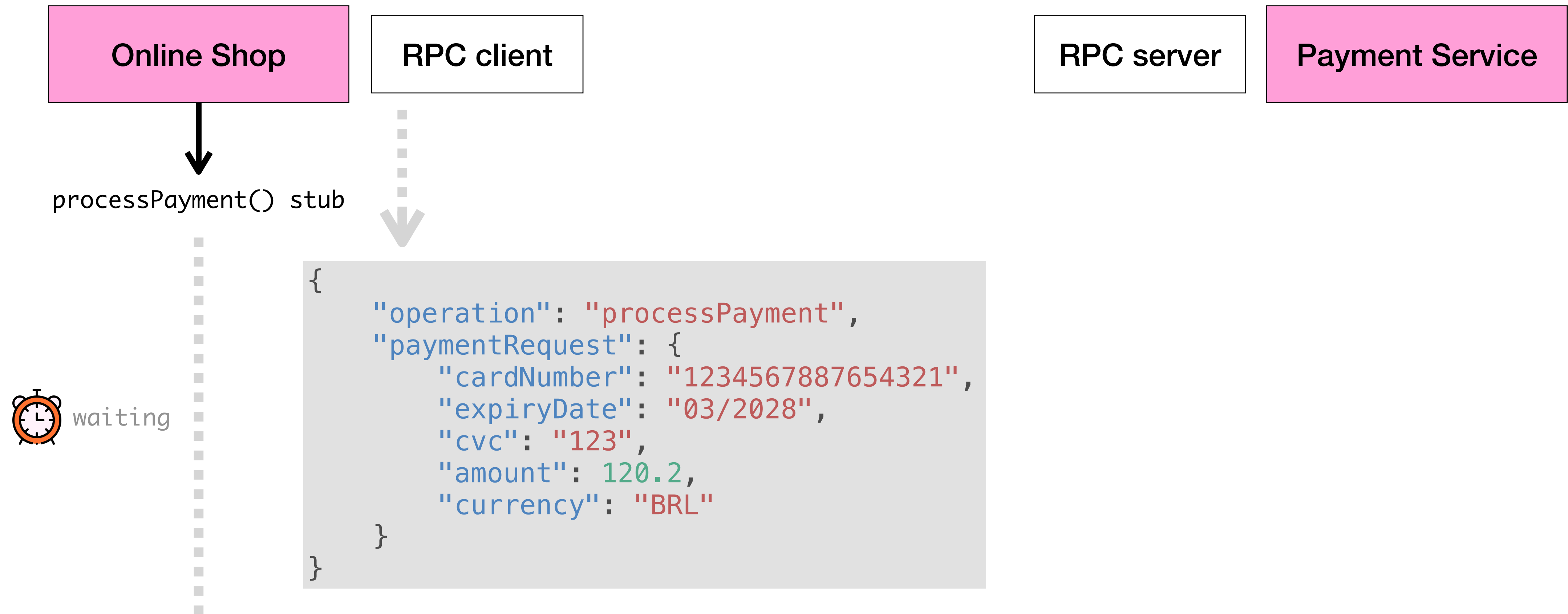
RPC Framework



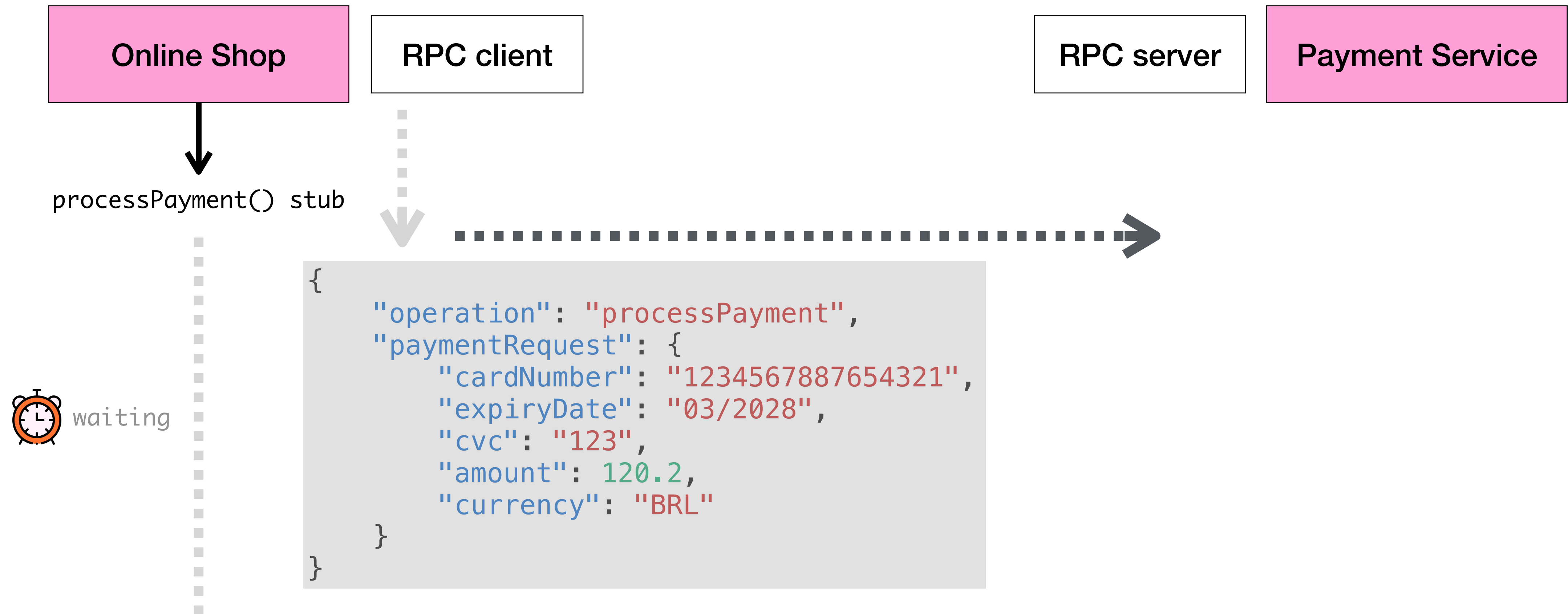
RPC Framework



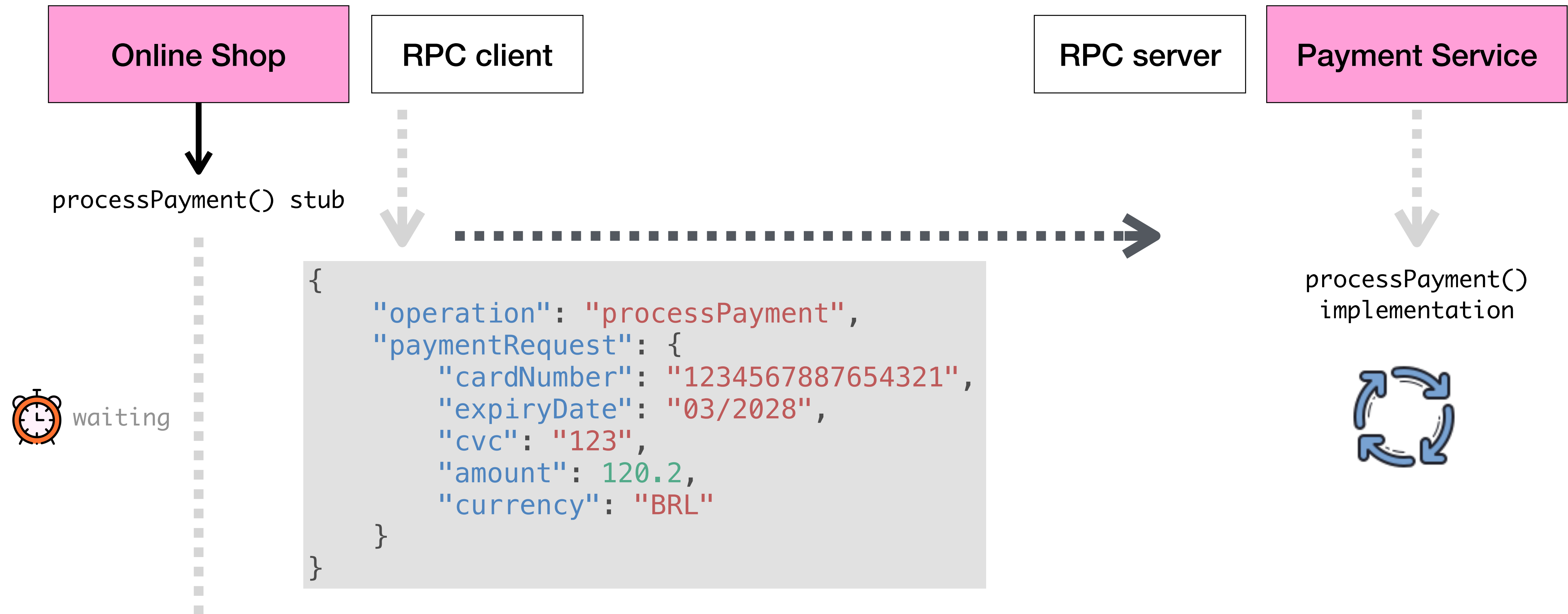
RPC Framework



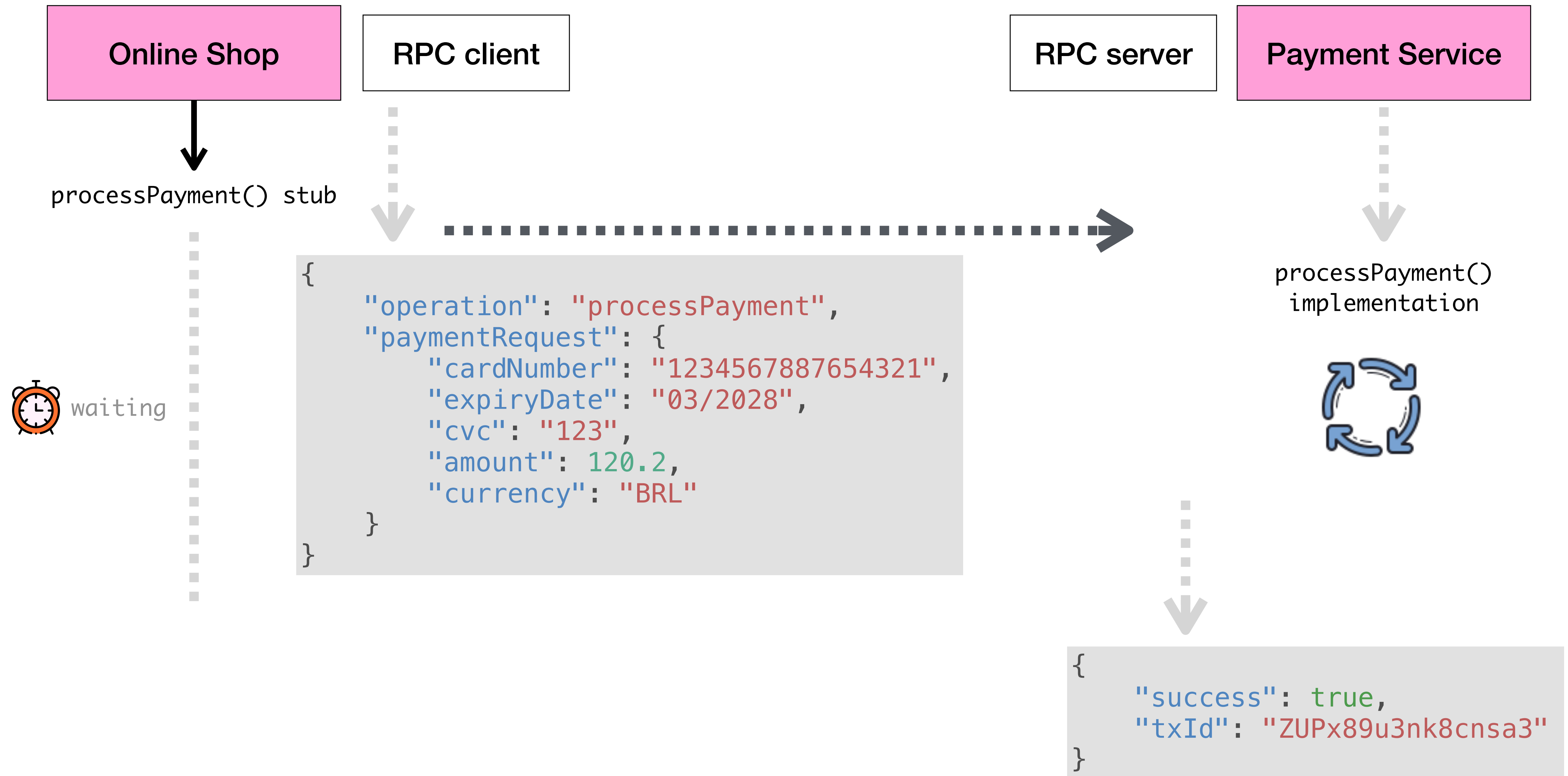
RPC Framework



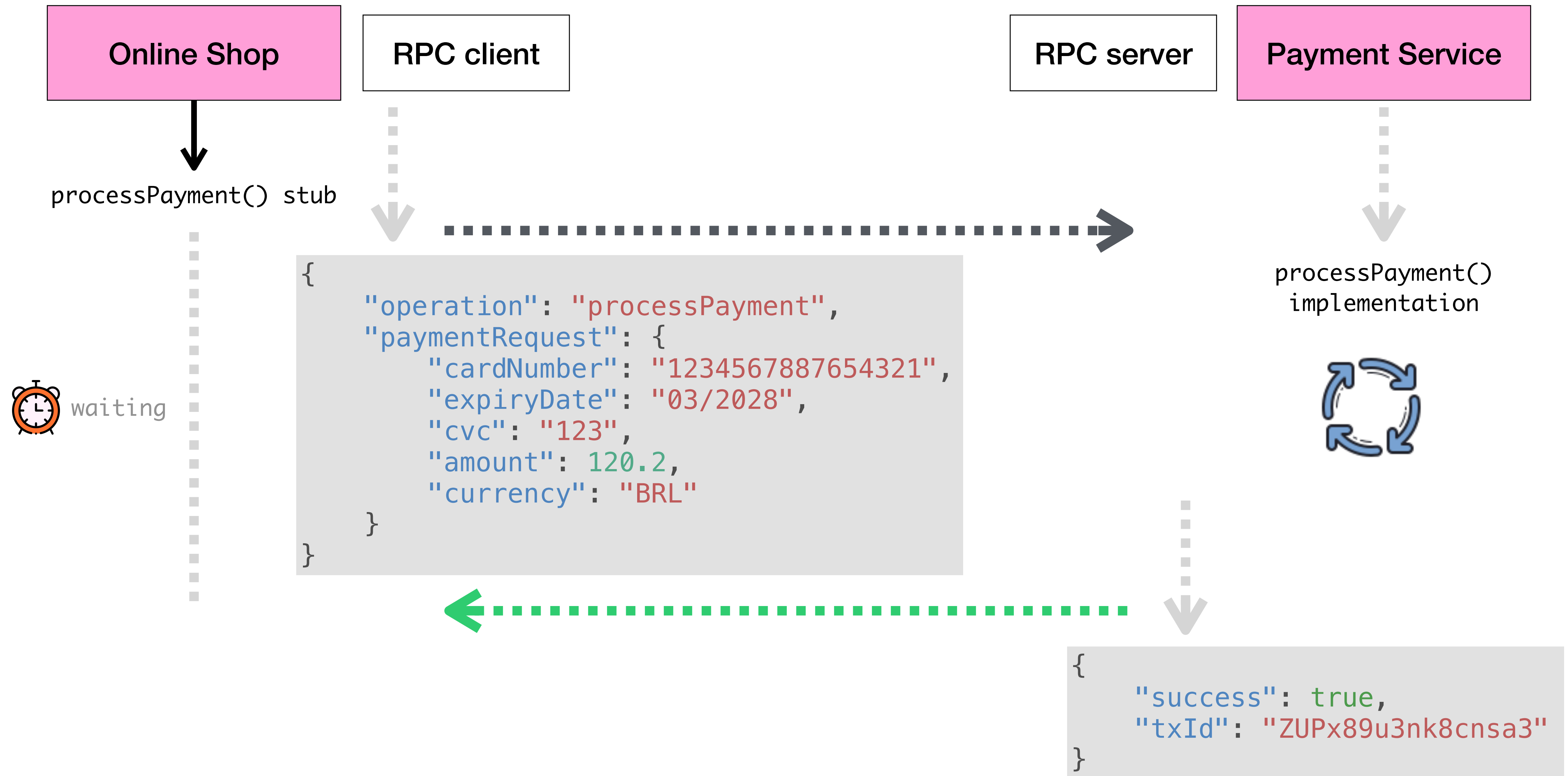
RPC Framework



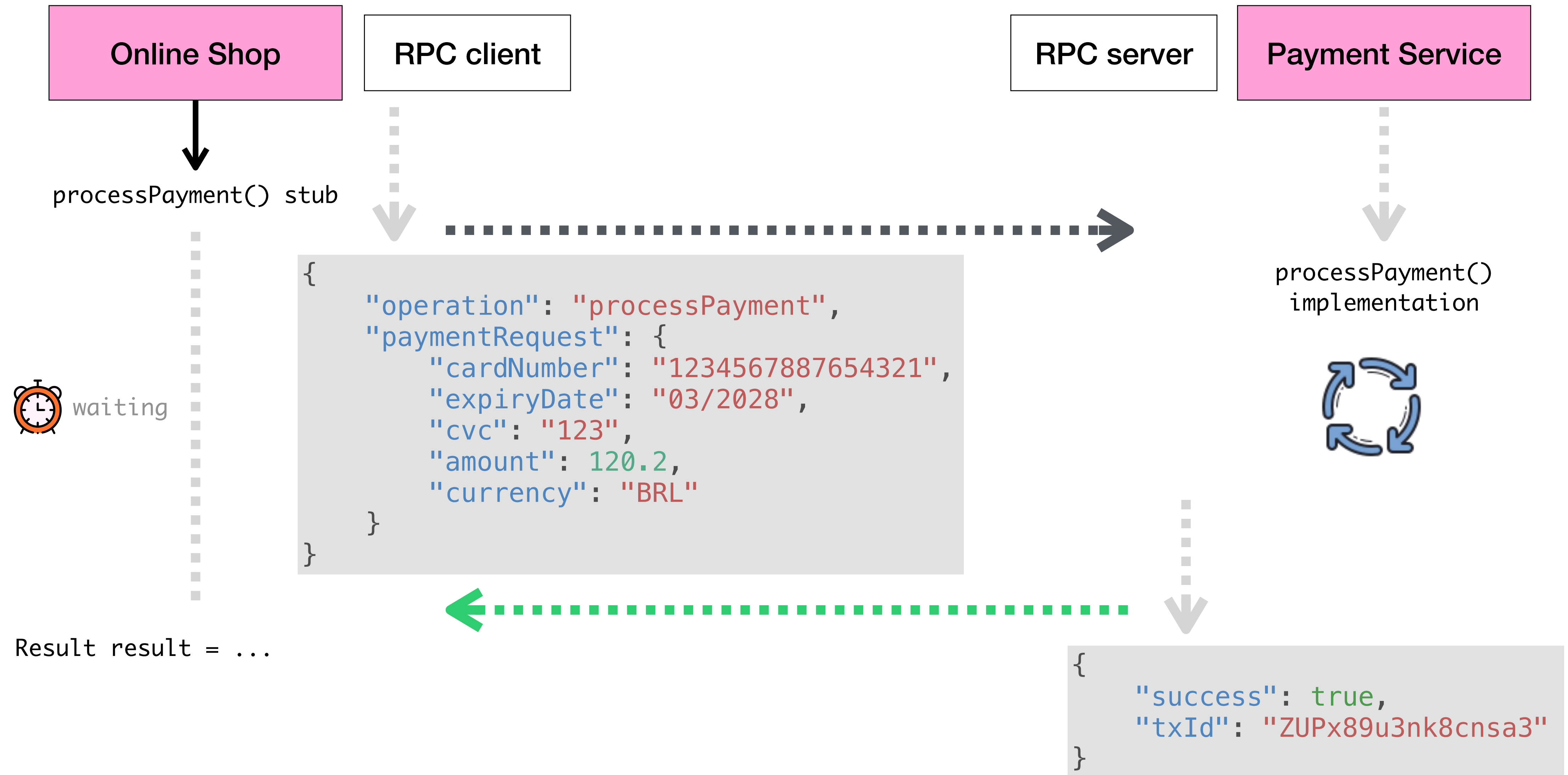
RPC Framework



RPC Framework



RPC Framework

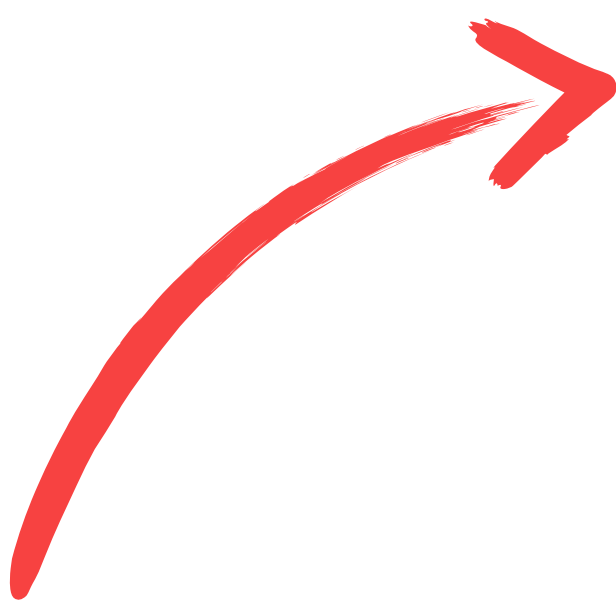


```
PaymentRequest request = new PaymentRequest();  
request.setCardNumber("1234 5678 8765 4321");  
request.setExpiryDate("03/2028");  
request.setCvc("123");  
request.setAmount(120.2);  
request.setCurrency(Currency.BRL);
```

```
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);
```

**Location
Transparency**



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


**Mas na
prática...**

Remote Calls

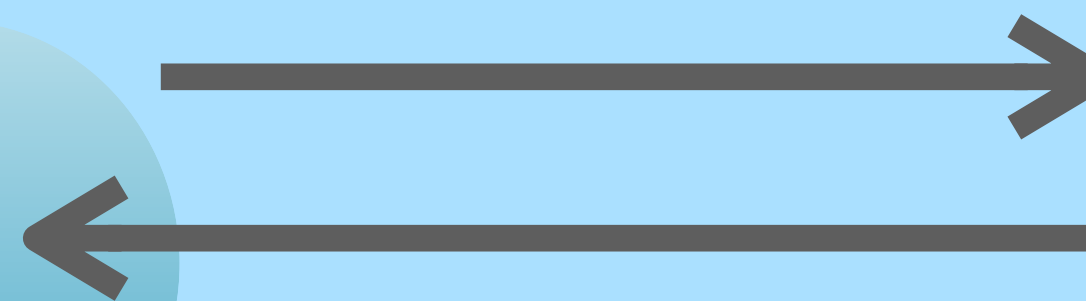
Spring Boot App

Client
(Online Shop)

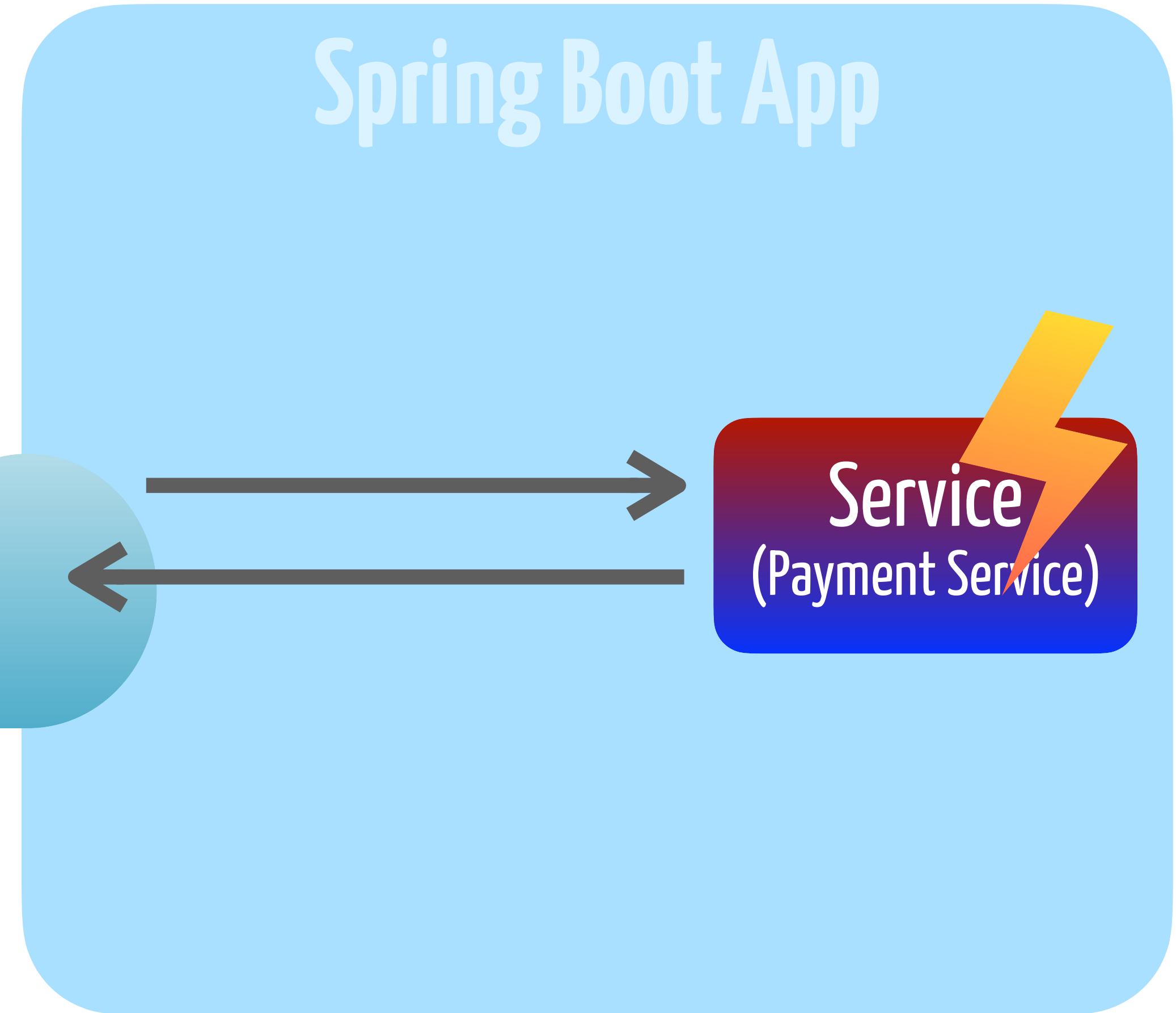
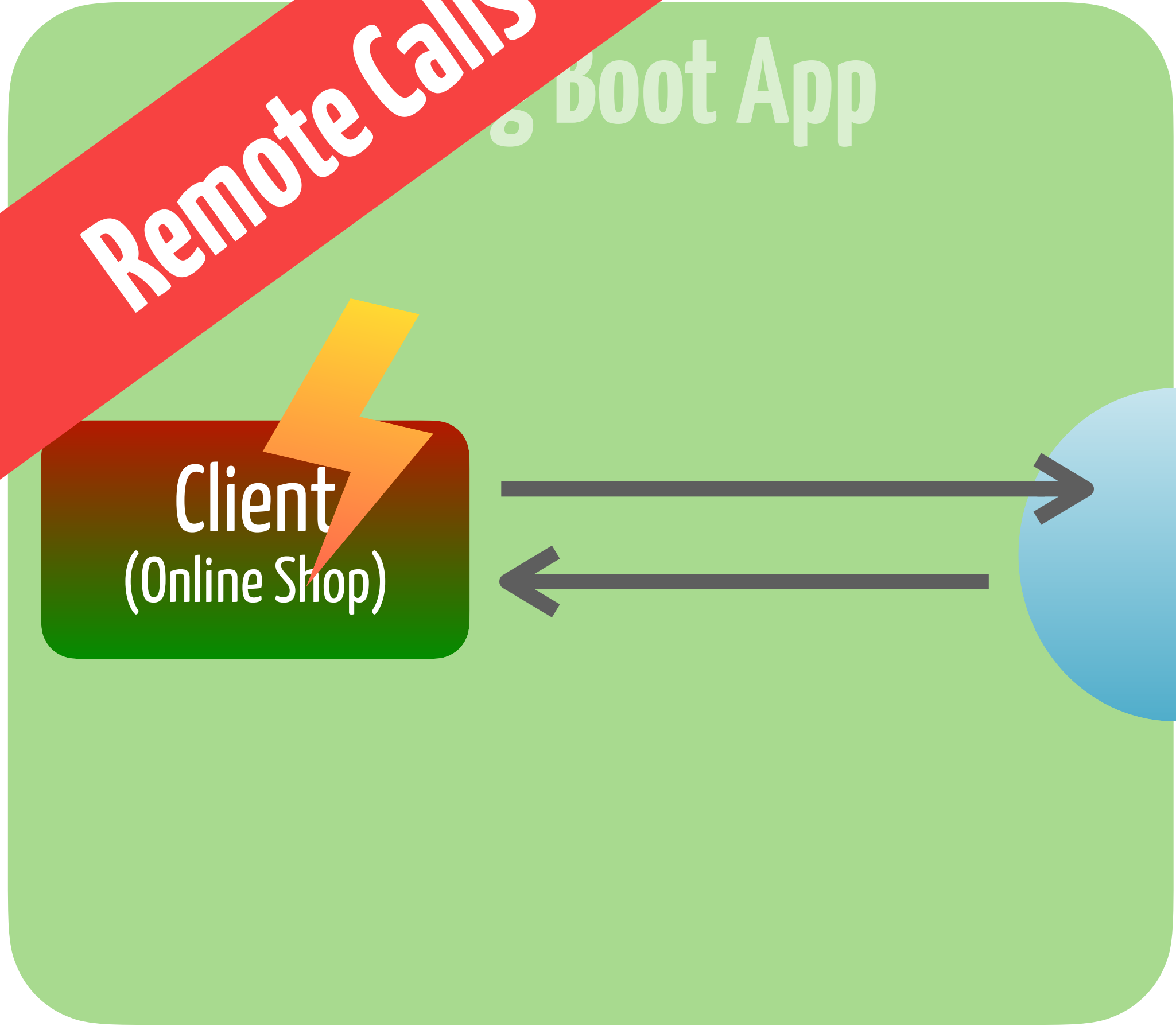
Internet

Spring Boot App

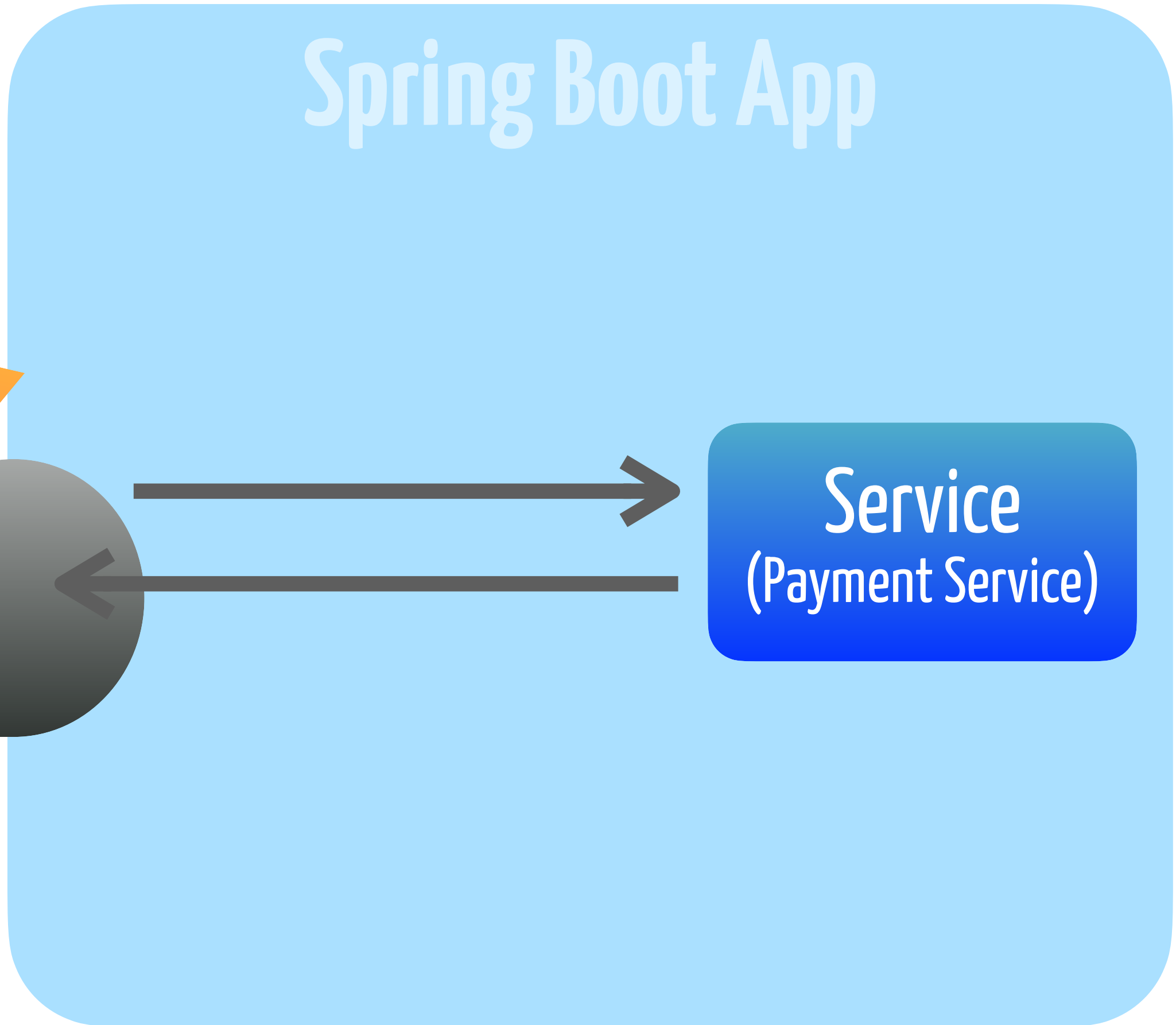
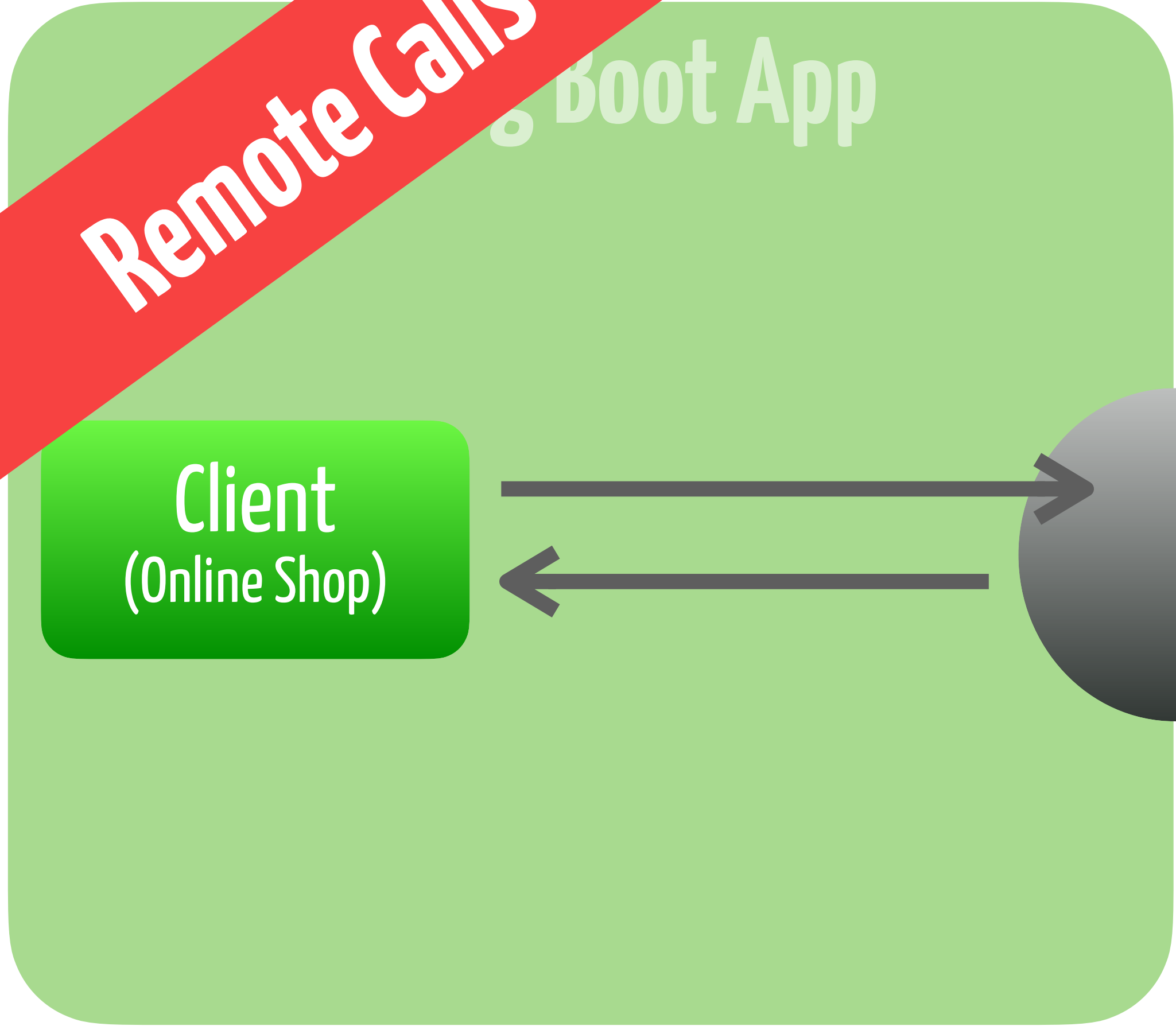
Service
(Payment Service)



Remote Calls



Remote Calls



Remote Calls

Spring Boot App

Client
(Online Shop)

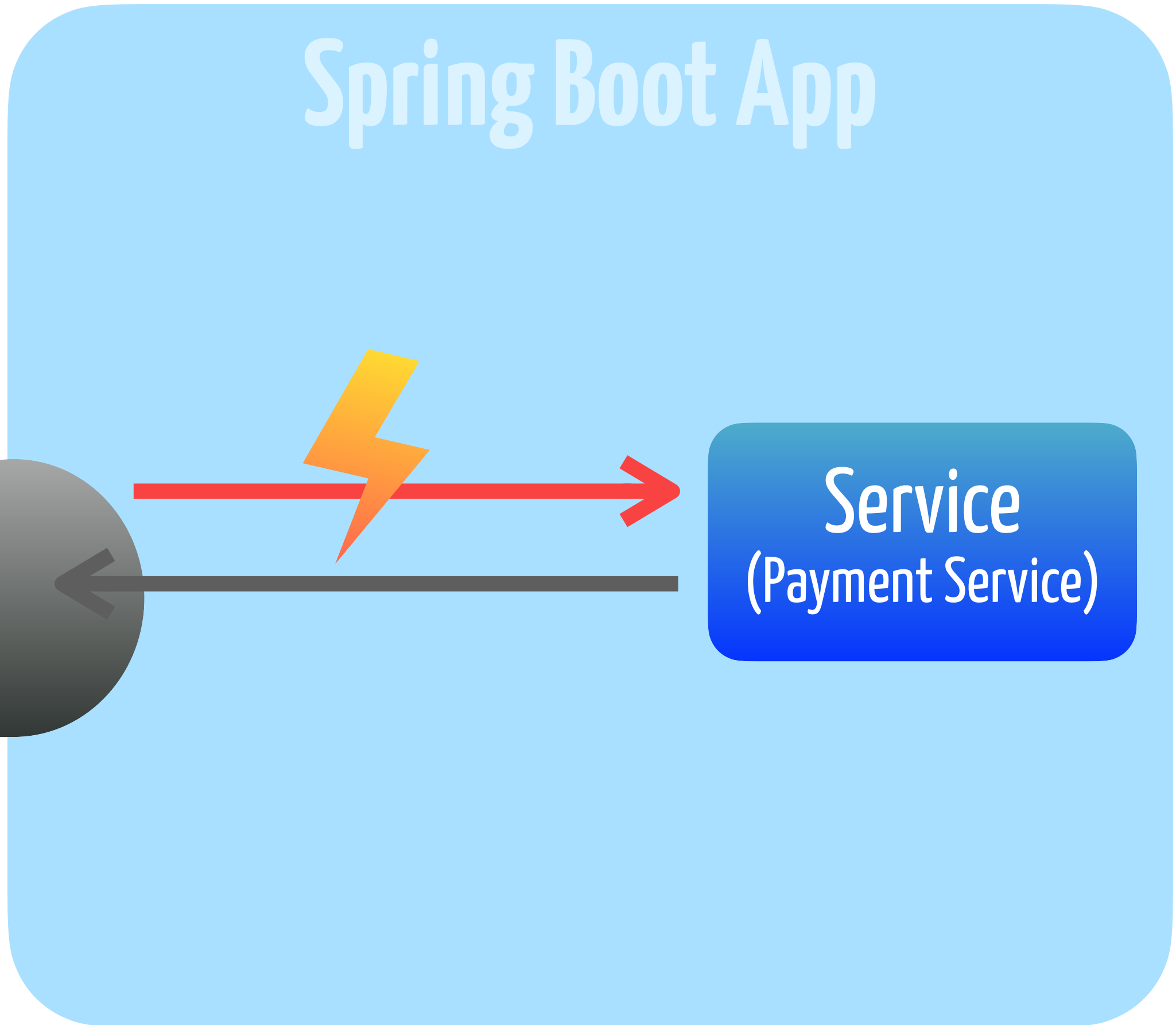
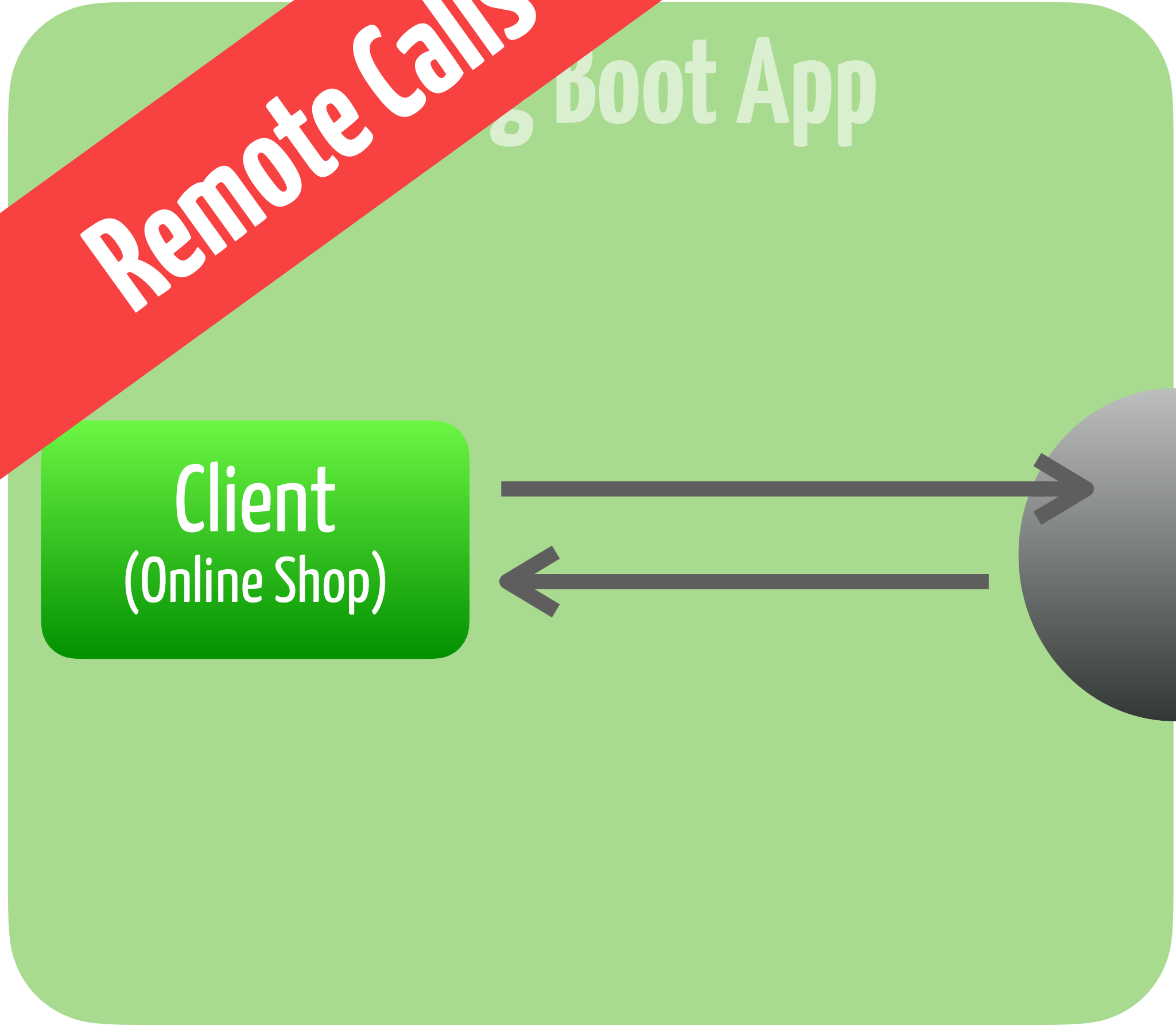
Internet

Spring Boot App

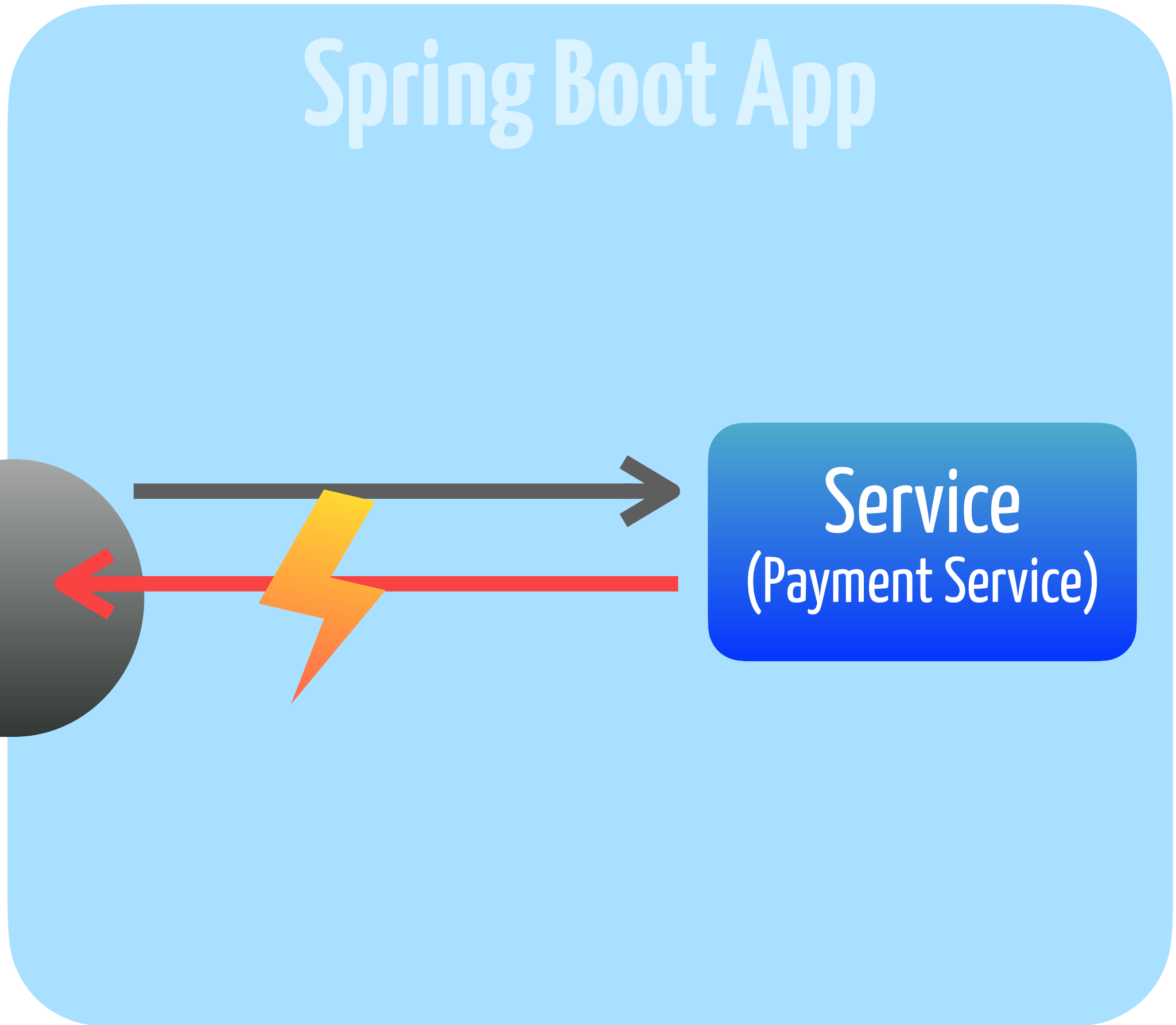
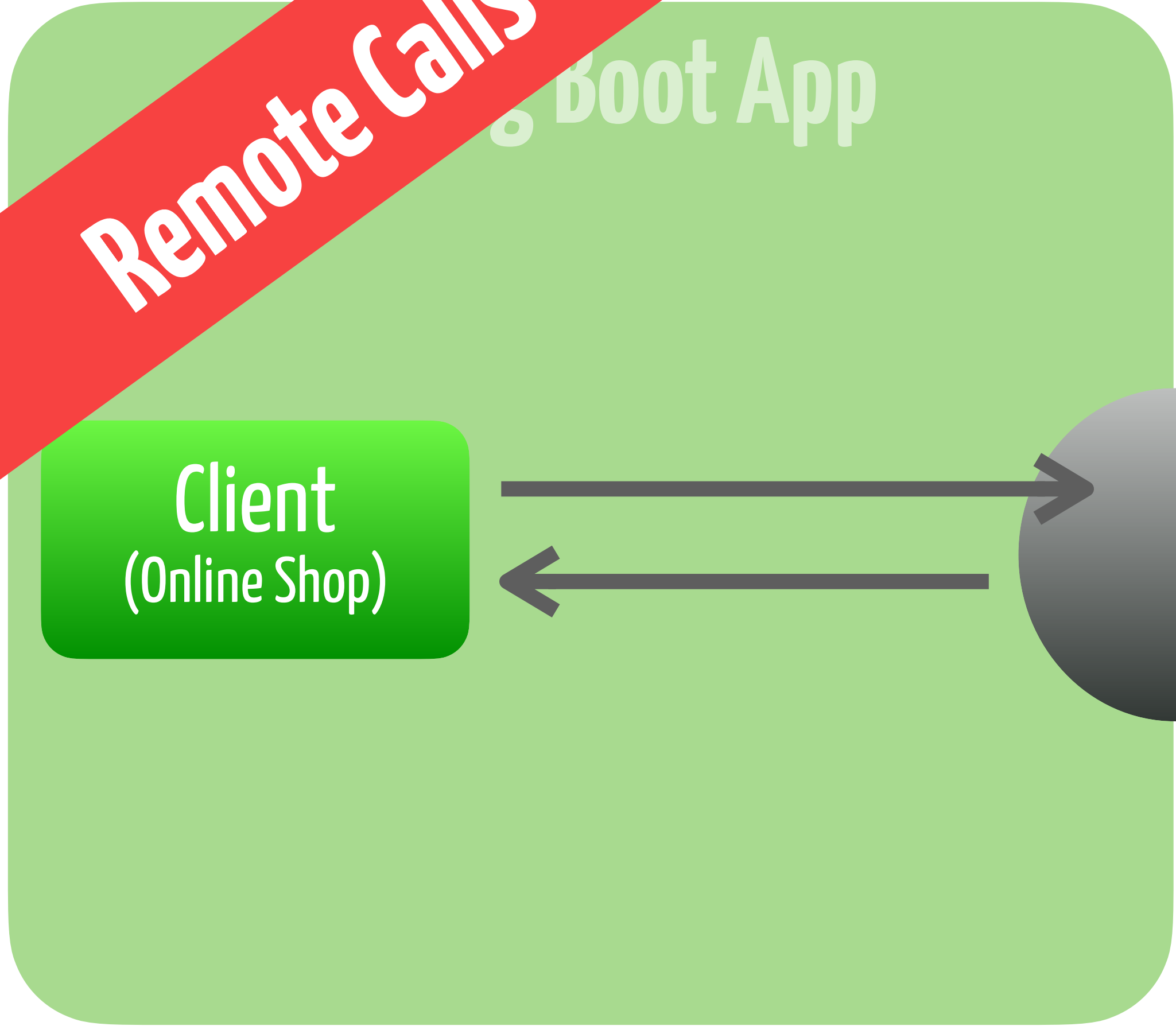
Service
(Payment Service)



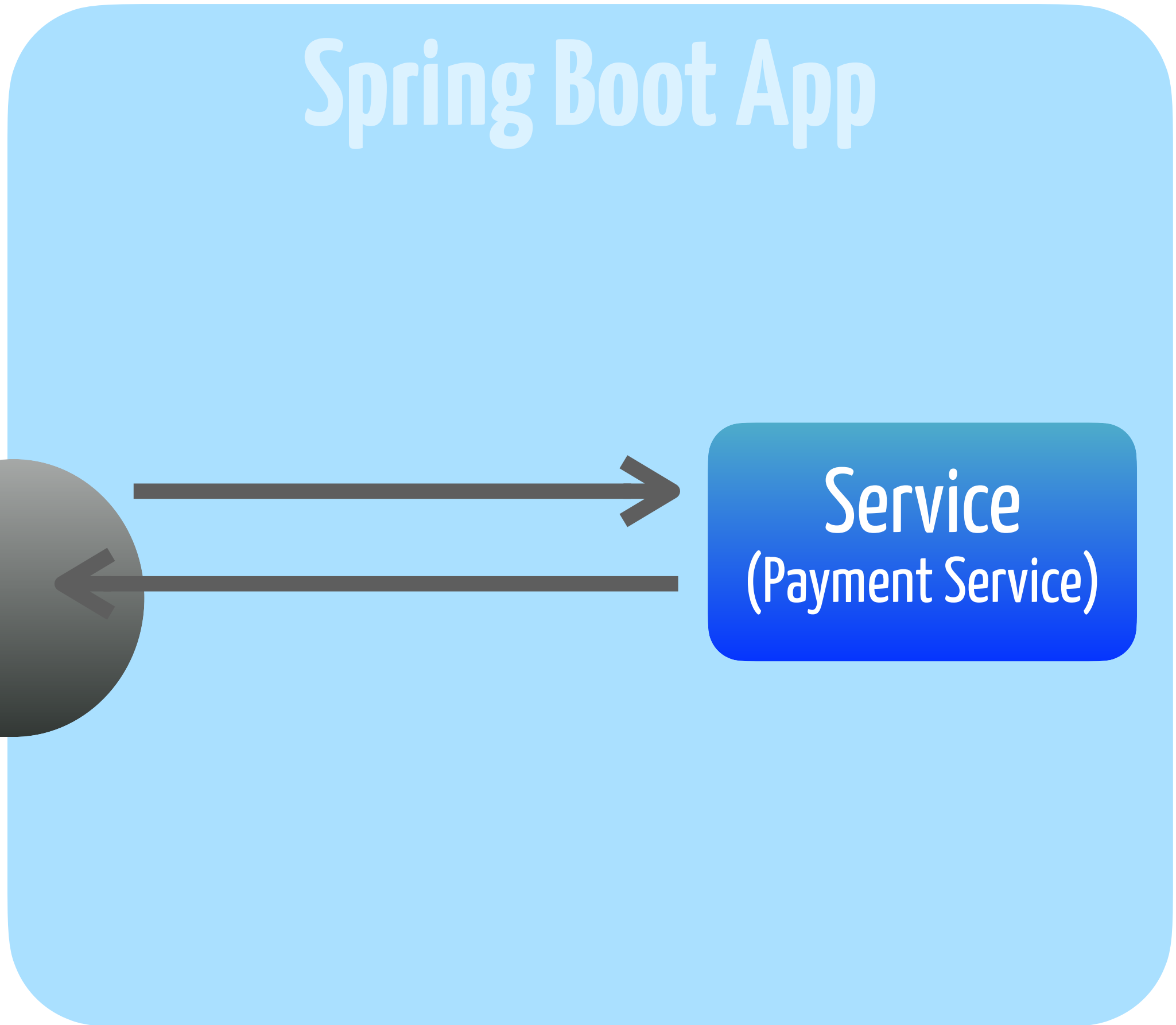
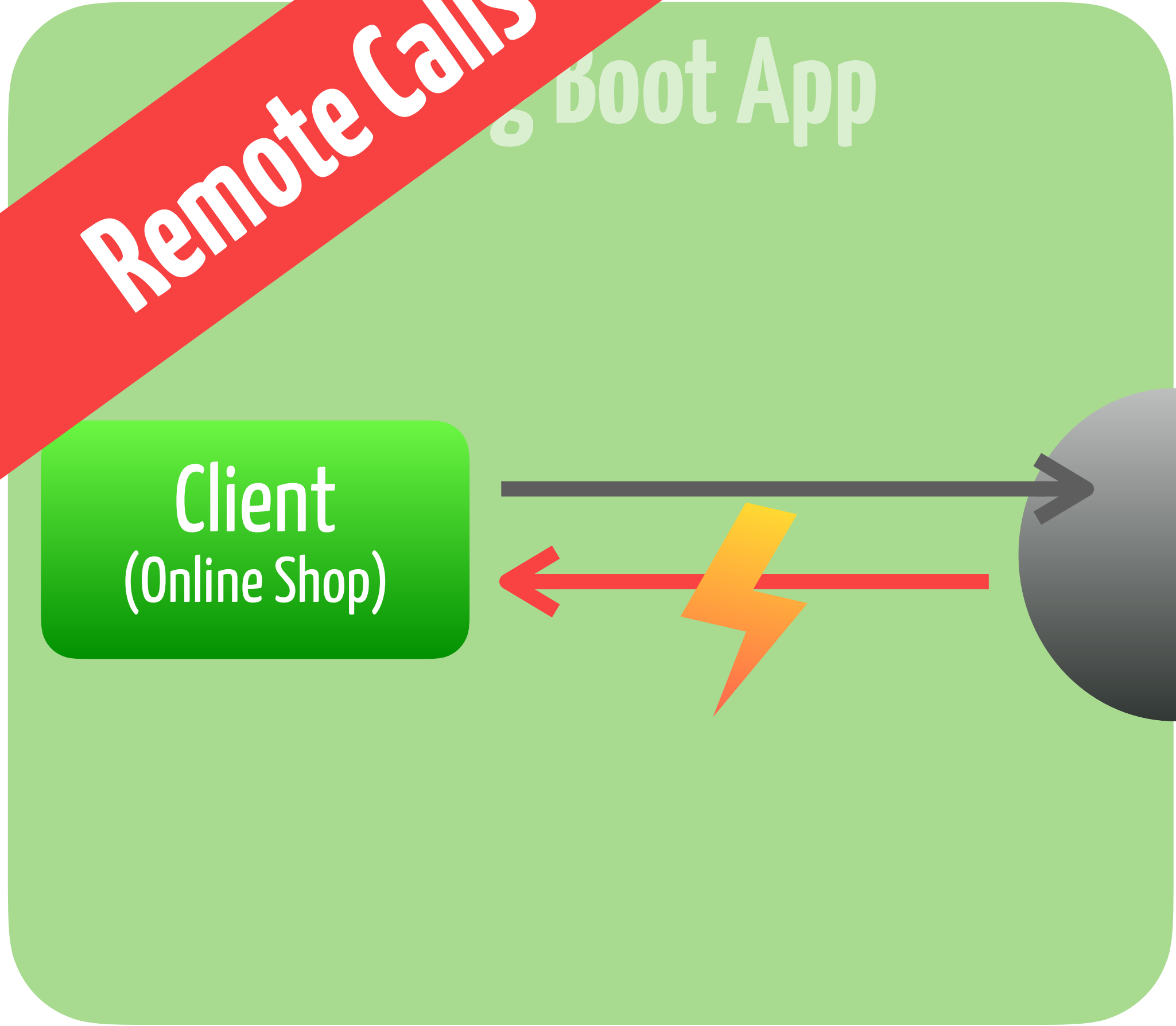
Remote Calls



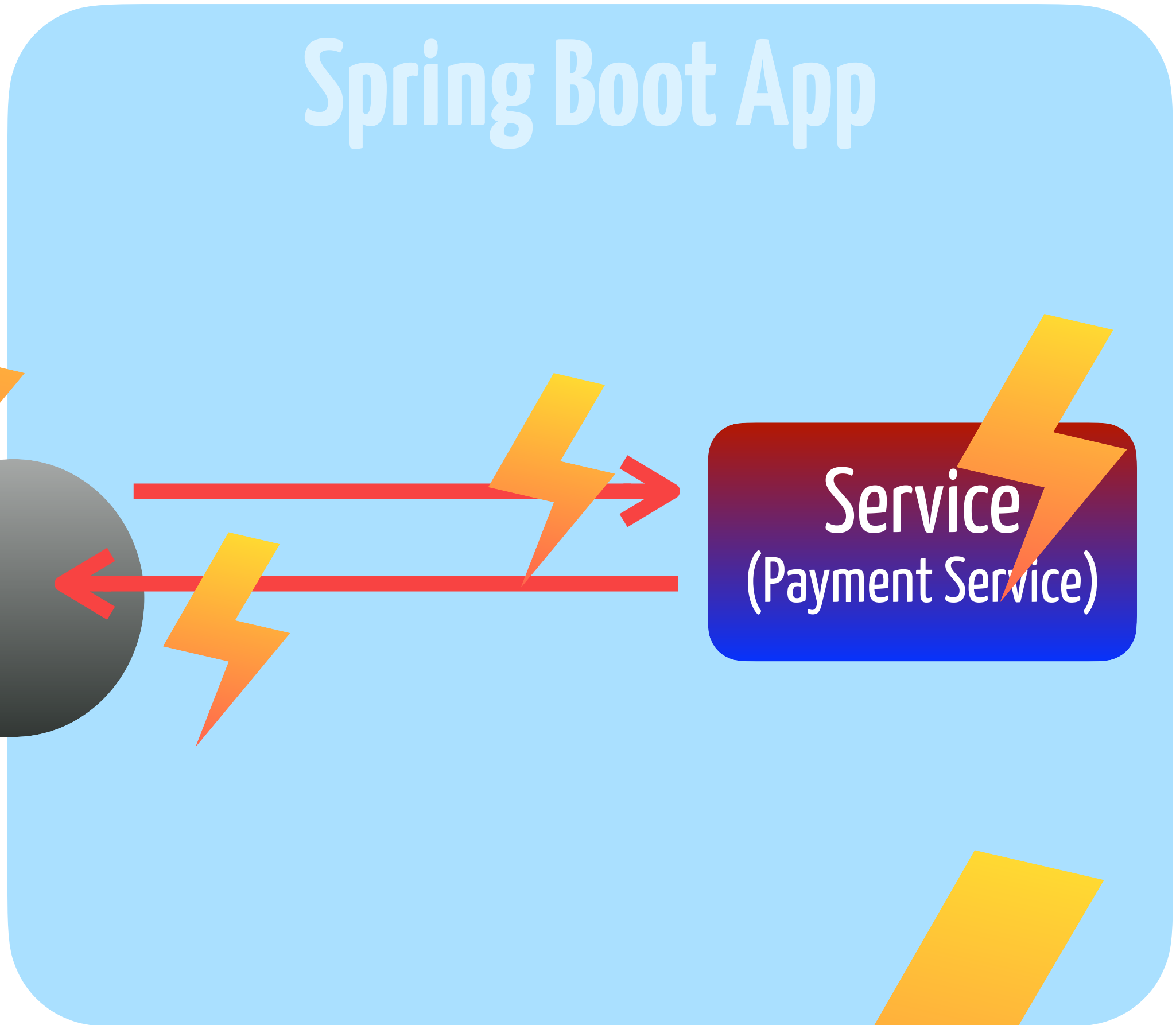
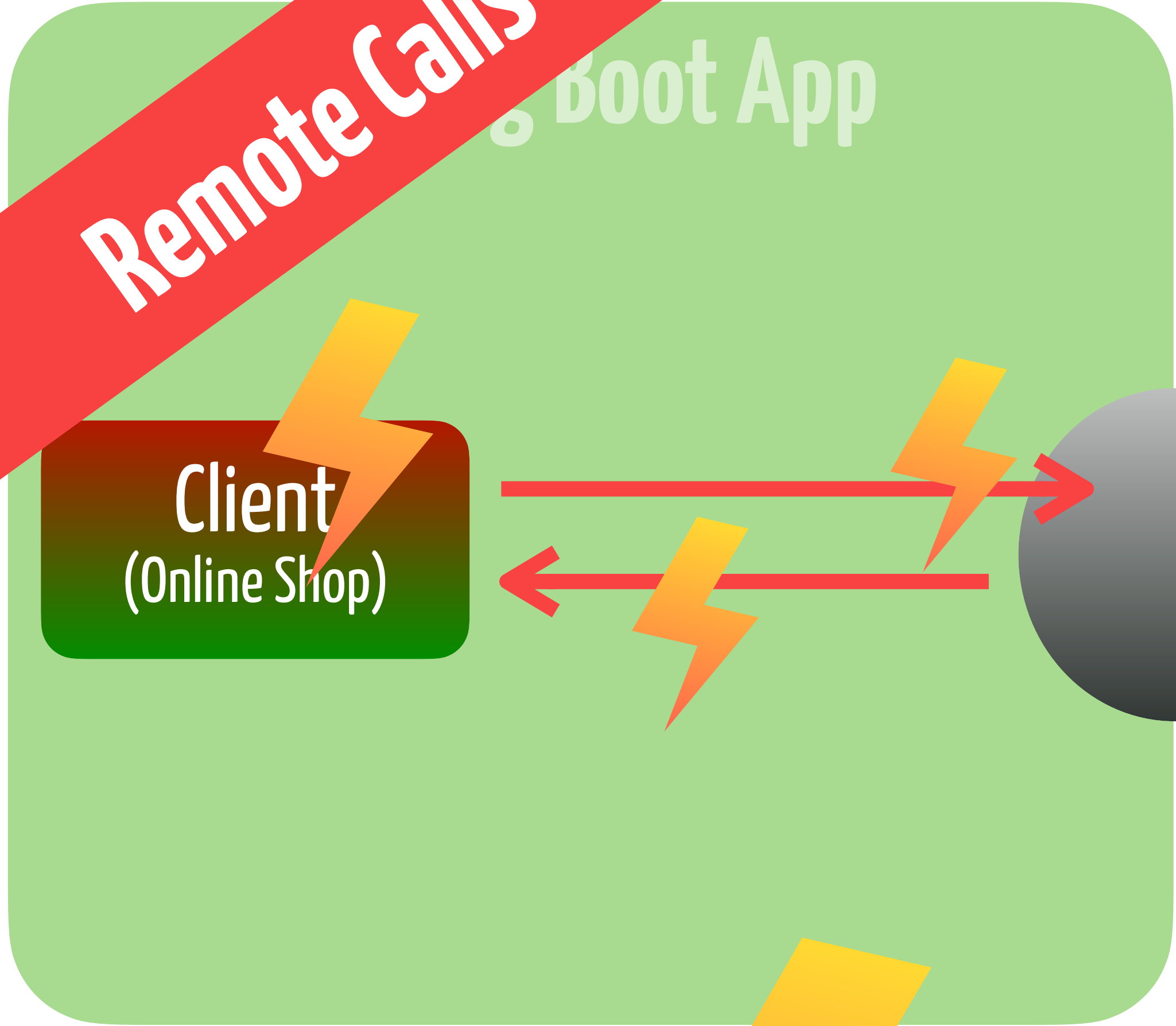
Remote Calls



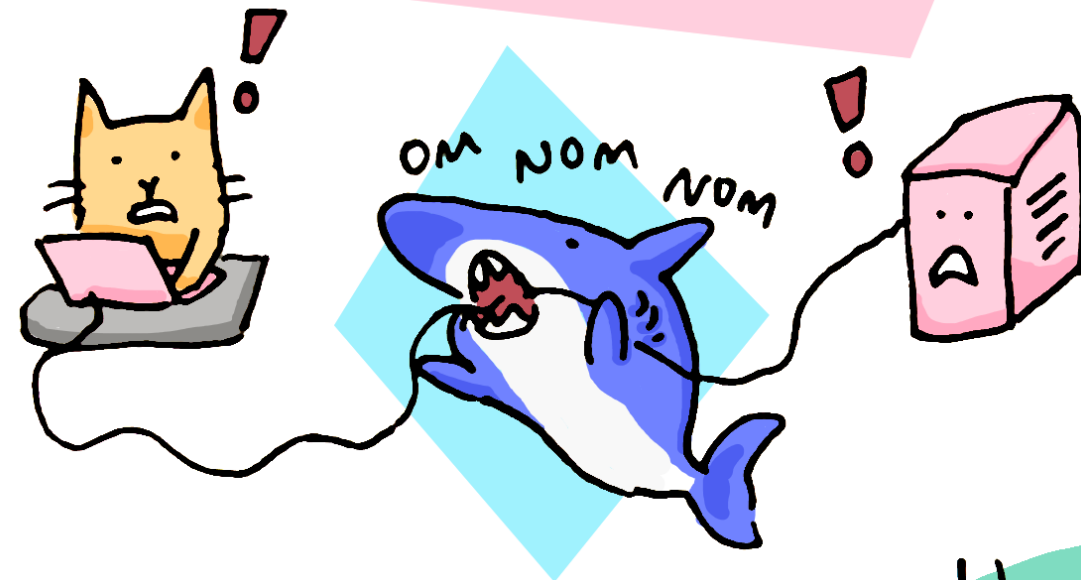
Remote Calls



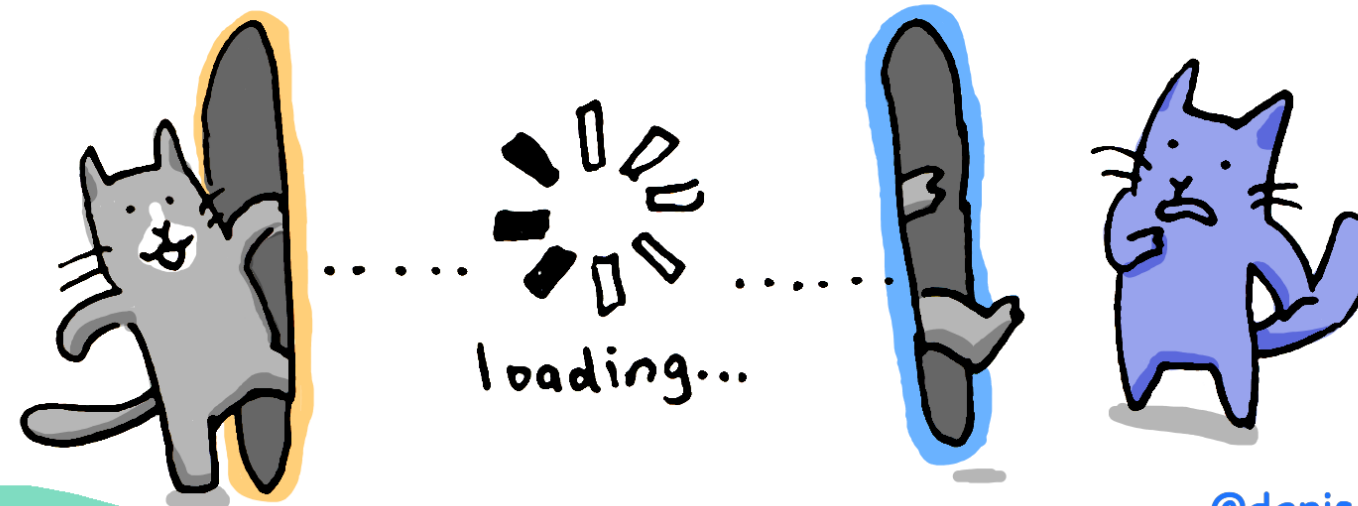
Remote Calls



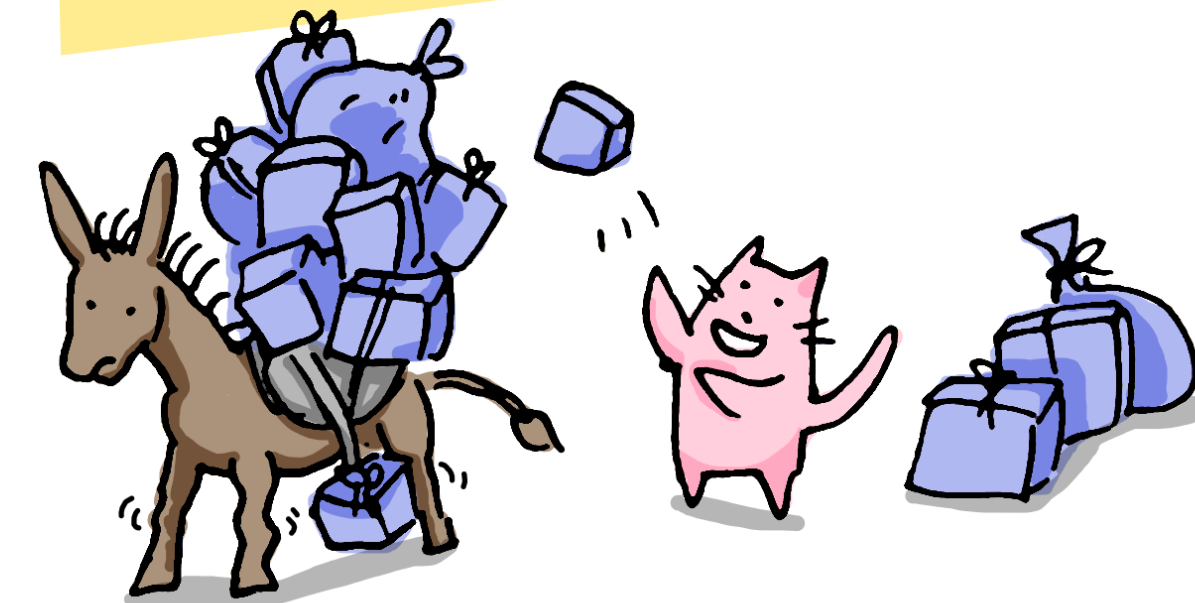
① The network is reliable



② Latency is ZERO



③ Bandwidth is infinite



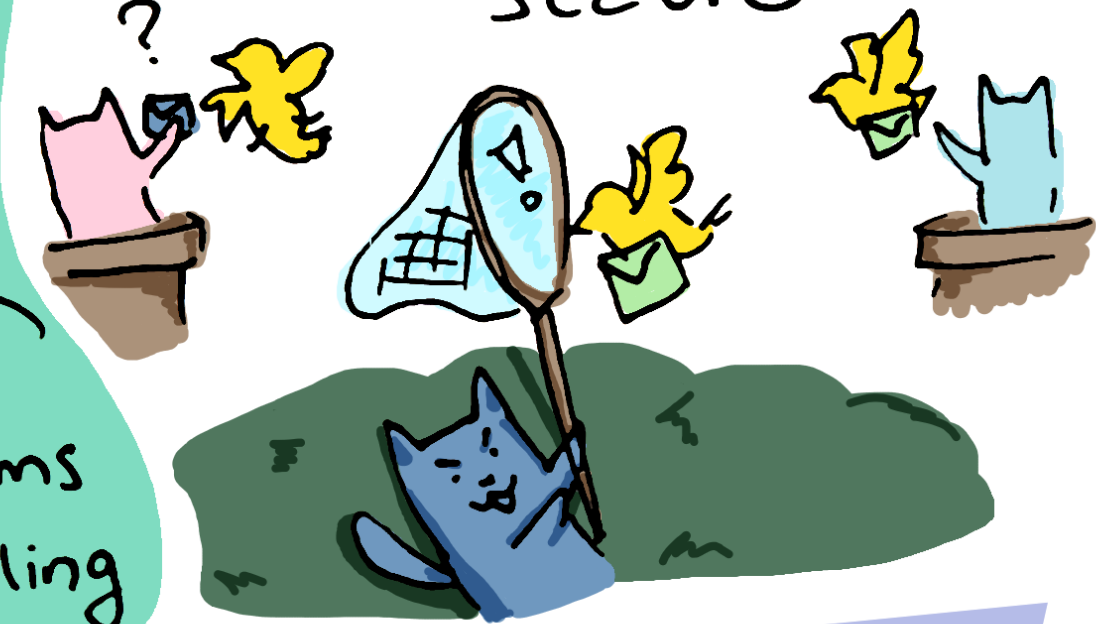
⑧ The network is homogeneous



the 8 Fallacies of Distributed Computing

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

④ The network is secure



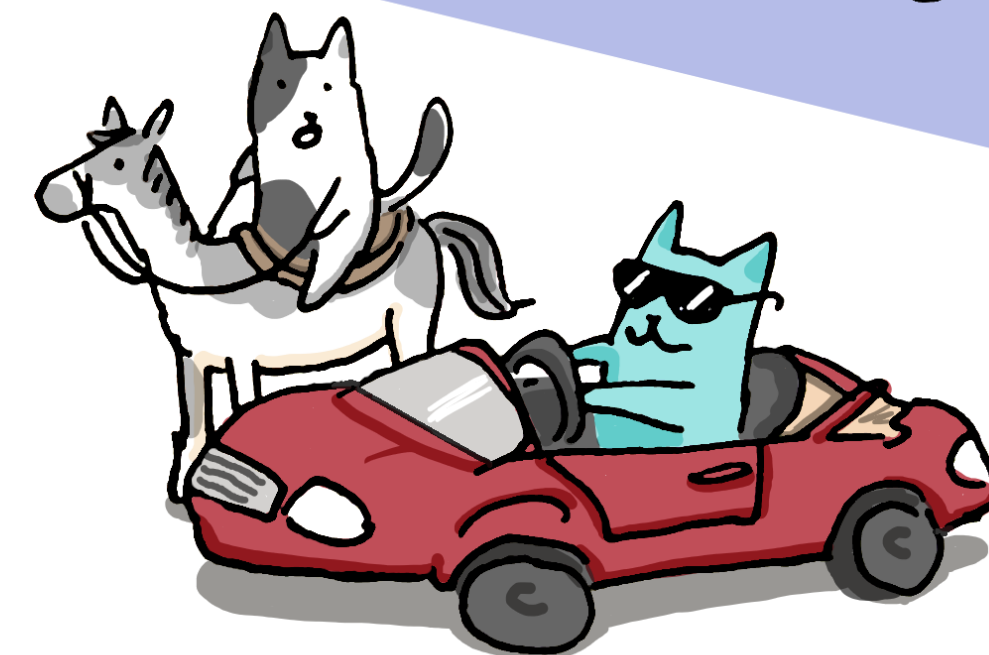
⑦ Transport costs \$0



⑥ There is only one administrator



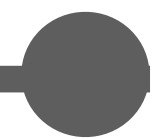
⑤ Topology doesn't change



**Falando em
frameworks...**

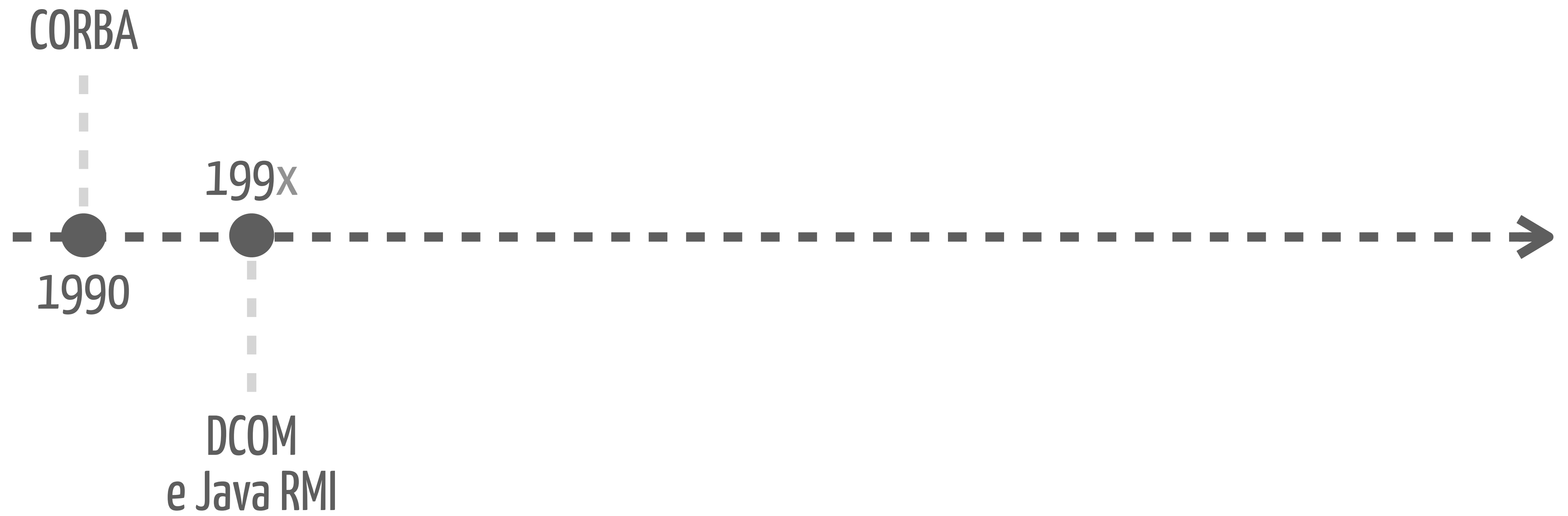


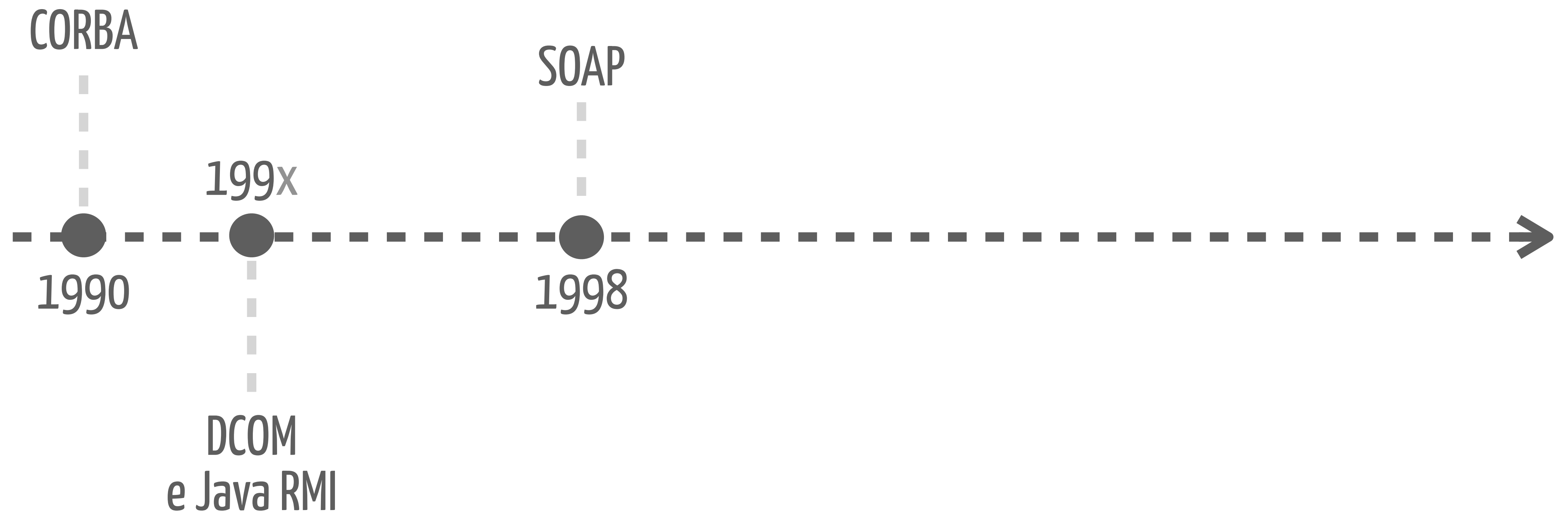
CORBA

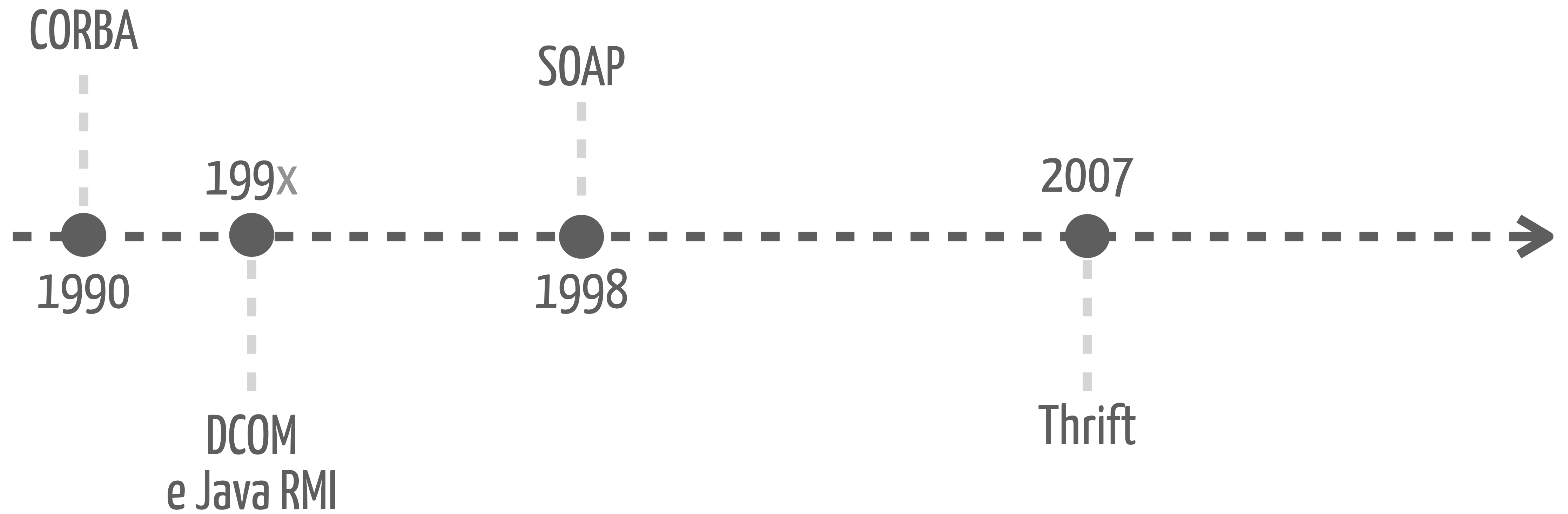


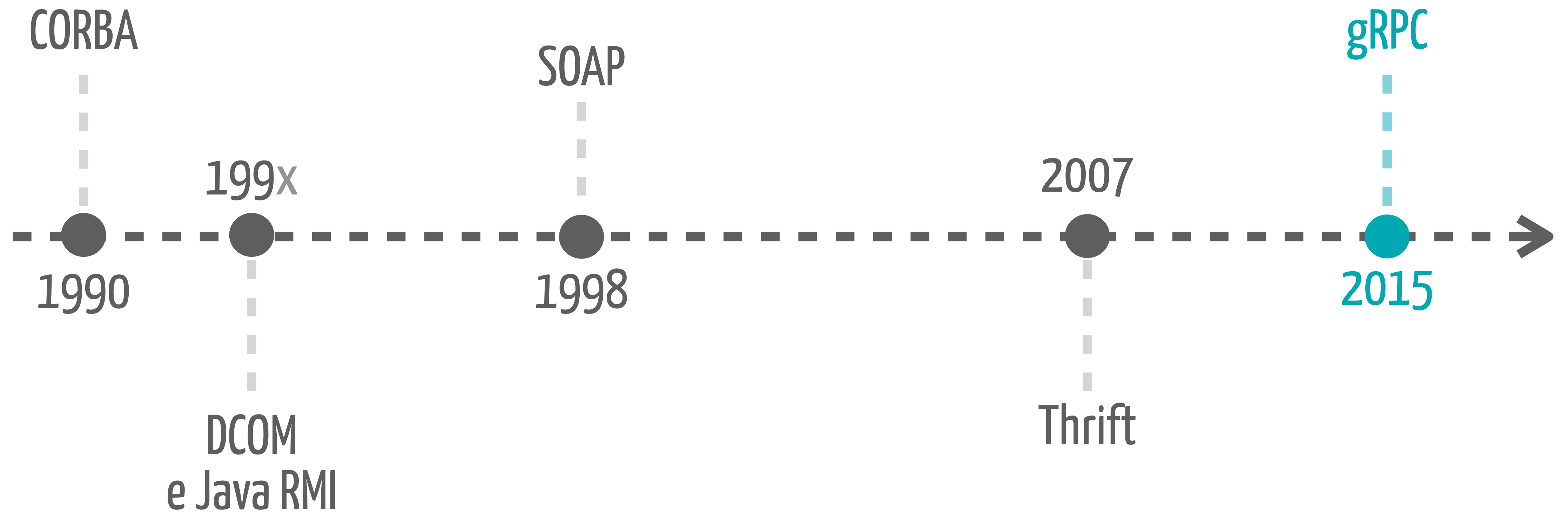
1990











tools.ietf.org

RFC 707 - High-level framework for network-based resource sharing

[\[Docs\]](#) [\[txt|pdf\]](#) [\[Tracker\]](#)

NWG/RFC# 707

NCC 76

JEW 14-JAN-76 19:51

4263

A High-Level Framework for Network-Based Resource Sharing

THE GOAL, RESOURCE SHARING

1

The principal goal of all resource-sharing computer networks, including the now international ARPA Network (the ARPANET), is to usefully interconnect geographically distributed hardware, software, and human resources [1]. Achieving this goal requires the design and implementation of various levels of support software within each constituent computer, and the specification of network-wide "protocols" (that is, conventions regarding the format and the relative timing of network messages) governing their interaction. This paper outlines an alternative to the approach that ARPANET system builders have been taking since work in this area began in 1970, and suggests a strategy for modeling distributed systems within any large computer network.

1a

The first section of this paper describes the prevailing ARPANET protocol strategy, which involves specifying a family of application-dependent protocols with a network-wide inter-process communication facility as their common foundation. In the second section, the application-independent command/response discipline that characterizes this protocol family is identified and its isolation as a separate protocol proposed. Such isolation would reduce the work of the applications programmer by allowing the software that implements the protocol to be factored out of each applications program and supplied as a single, installation-maintained module. The final section of this paper proposes an extensible model for this class of network interaction that in itself would even further encourage the use of network resources.

1b

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

The image shows a web browser window displaying the RFC 707 document. The browser's address bar shows the URL `https://tools.ietf.org/html/rfc707`. The page title is "RFC 707 - High-level framework for network-based resource sharing". Below the title, there are links for "[Docs]", "[txt|pdf]", and "[Tracker]". The document header includes "NWG/RFC# 707", "NCC 76", and "A High-Level Framework for Network-Based Resource Sharing". The main text begins with "THE GOAL, RESOURCE SHARING" and describes the principal goal of all resource-sharing computer networks. A red magnifying glass is positioned over the header information, highlighting the text "JEW 14-JAN-76 19:5" and "Network-Based Resource".

[Docs] [txt|pdf] [Tracker]

NWG/RFC# 707
NCC 76

A High-Level Framework for Network-Based Resource Sharing

JEW 14-JAN-76 19:5

Network-Based Resource

THE GOAL, RESOURCE SHARING

The principal goal of all resource-sharing computer networks, including the now international ARPA Network (the ARPANET), is to usefully interconnect geographically distributed hardware, software, and human resources [1]. Achieving this goal requires the design and implementation of various levels of support software within each constituent computer, and the specification of network-wide "protocols" (that is, conventions regarding the format and the relative timing of network messages) governing their interaction. This paper outlines an alternative to the approach that ARPANET system builders have been taking since work in this area began in 1970, and suggests a strategy for modeling distributed systems within any large computer network.

1a

The first section of this paper describes the prevailing ARPANET protocol strategy, which involves specifying a family of application-dependent protocols with a network-wide inter-process communication facility as their common foundation. In the second section, the application-independent command/response discipline that characterizes this protocol family is identified and its isolation as a separate protocol proposed. Such isolation would reduce the work of the applications programmer by allowing the software that implements the protocol to be factored out of each applications program and supplied as a single, installation-maintained module. The final section of this paper proposes an extensible model for this class of network interaction that in itself would even further encourage the use of network resources.

1b

gRPC

gRPC Remote Procedure Call framework



gRPC Remote Procedure Call framework

gRPC

gRPC Remote Procedure Call framework

simple & idiomatic



gRPC Remote Procedure Call framework

simple & idiomatic

performant & scalable



gRPC Remote Procedure Call framework

simple & idiomatic

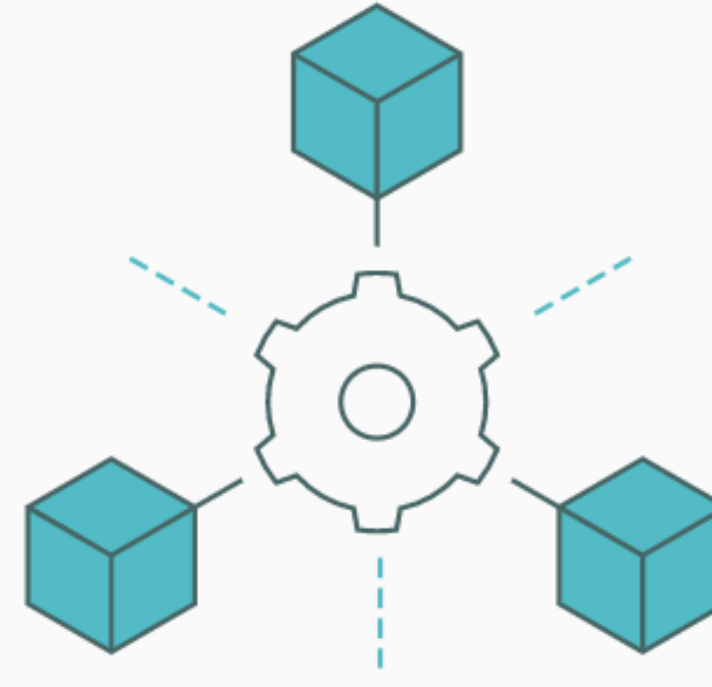
performant & scalable

interoperable & extensible



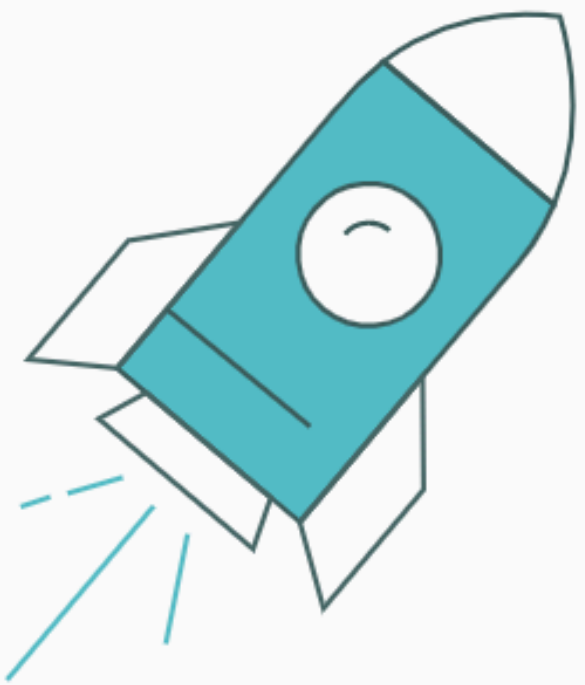
Simple service definition

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



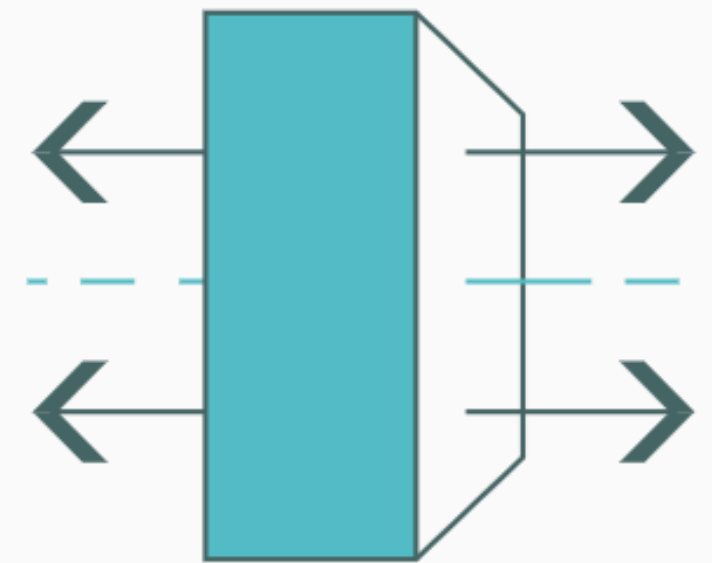
Start quickly and scale

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 (<i>.proto</i>)	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



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

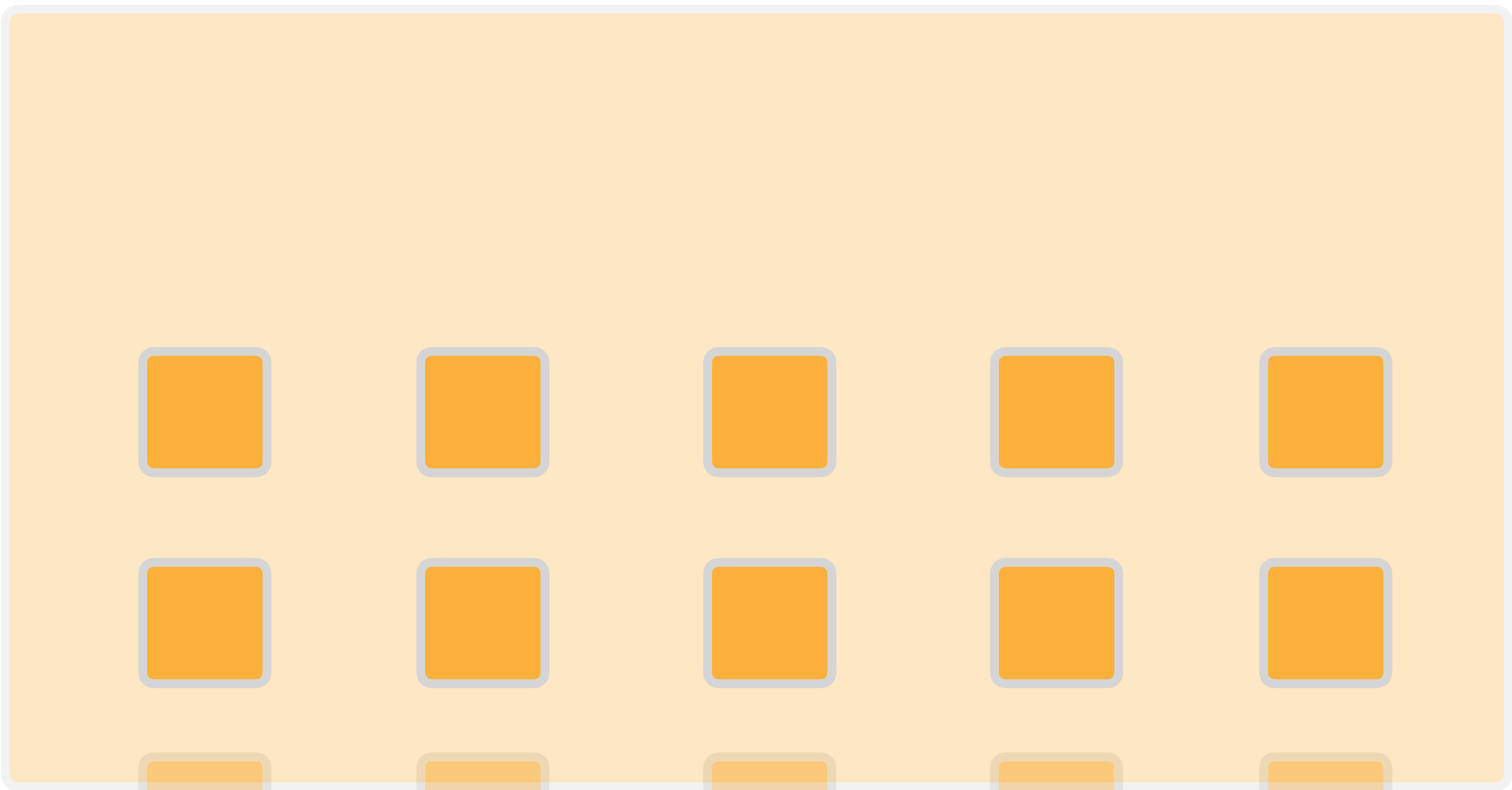
IDL

HTTP/2

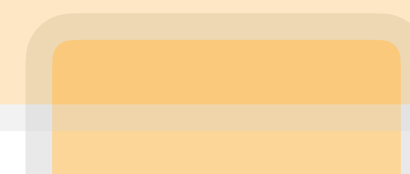
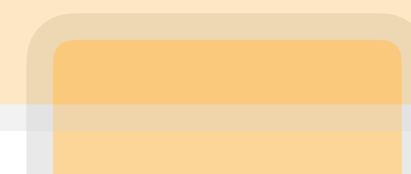
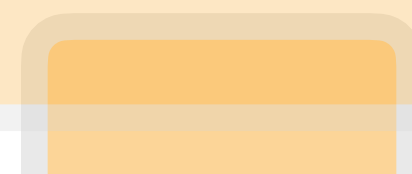
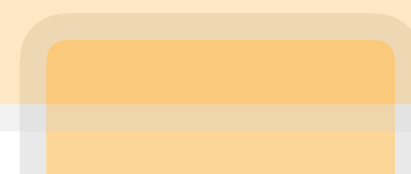
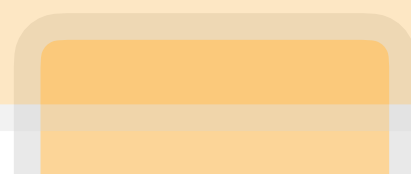
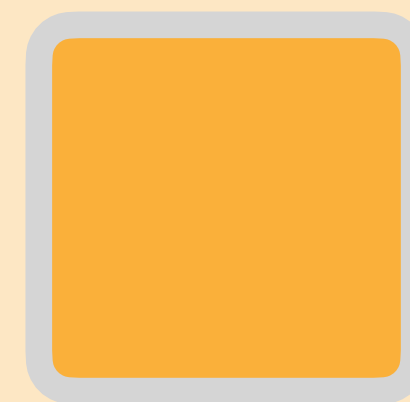
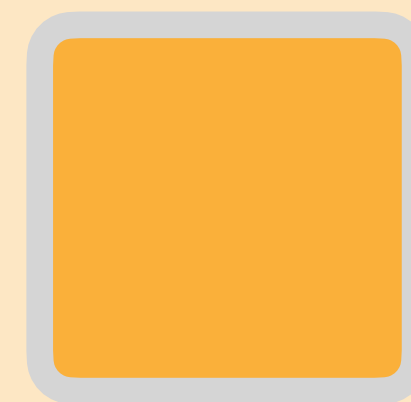
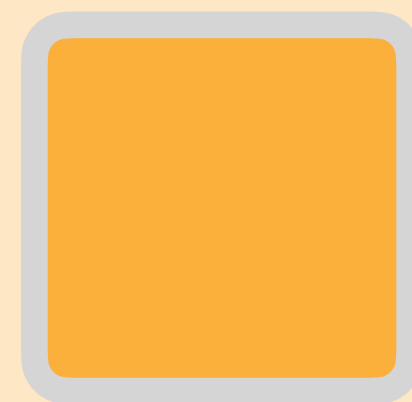
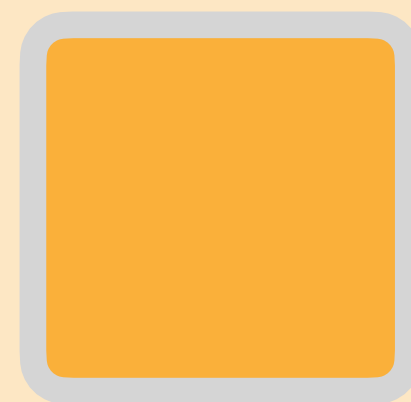
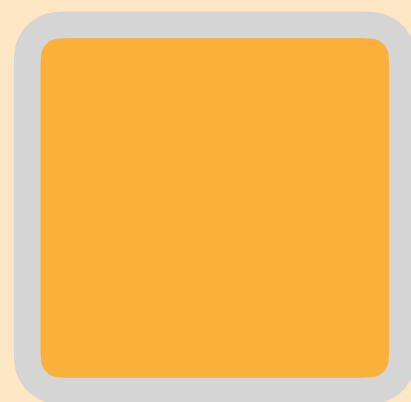
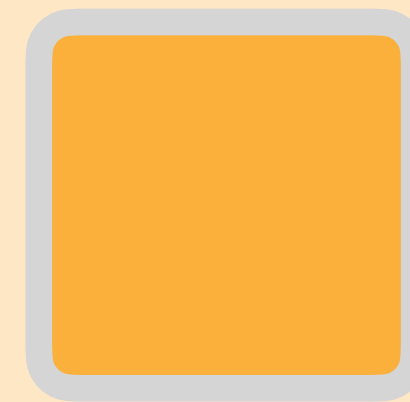
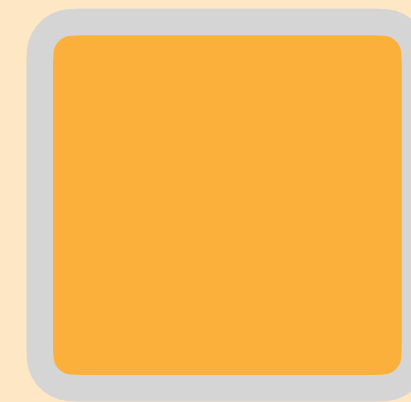
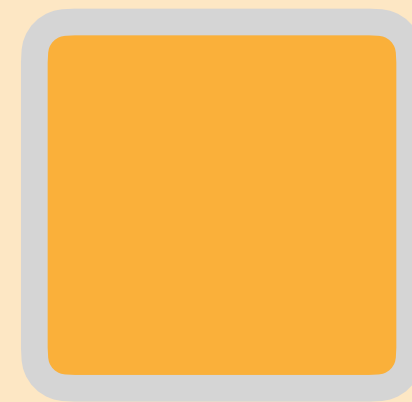
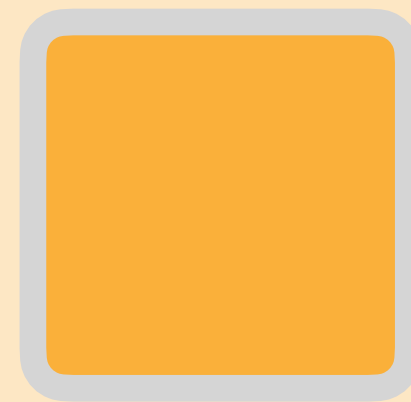
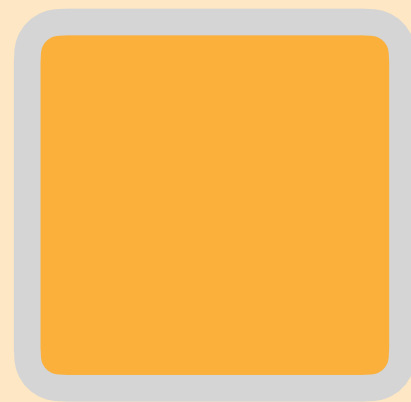
Protobuf v3



Microservices



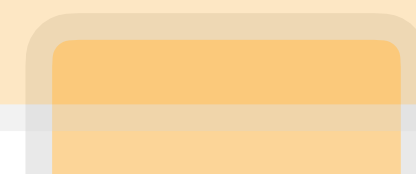
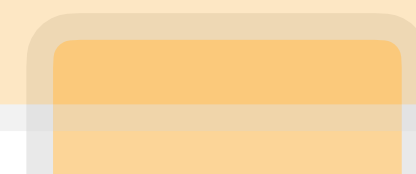
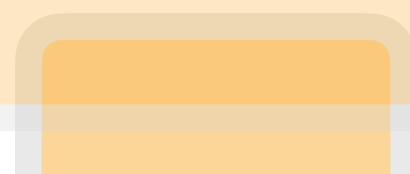
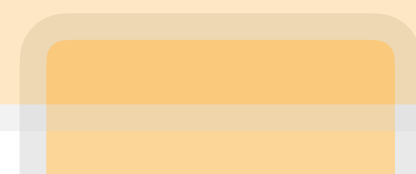
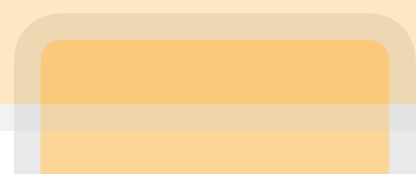
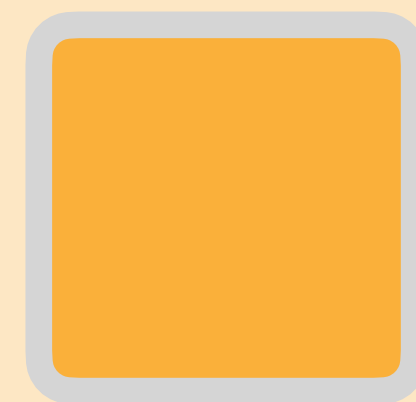
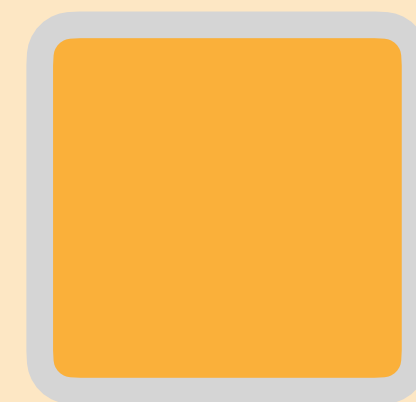
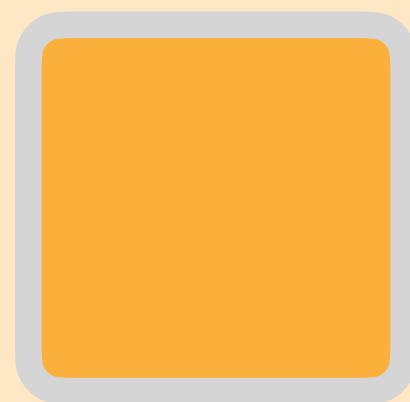
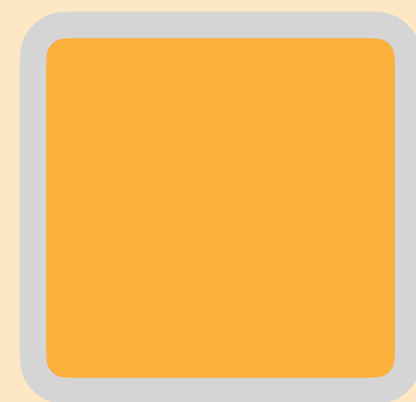
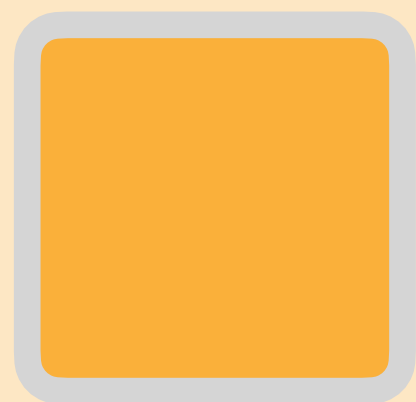
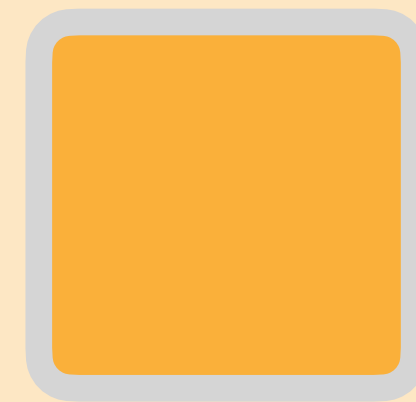
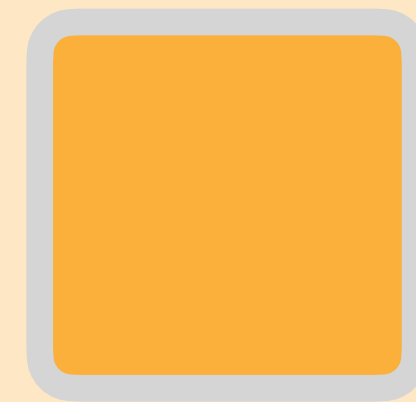
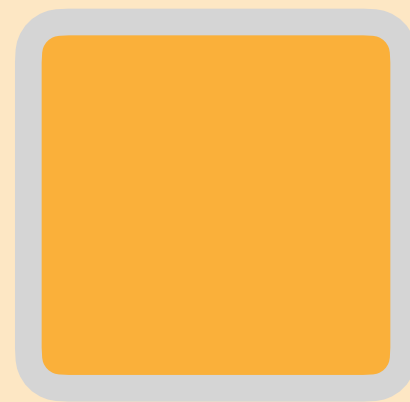
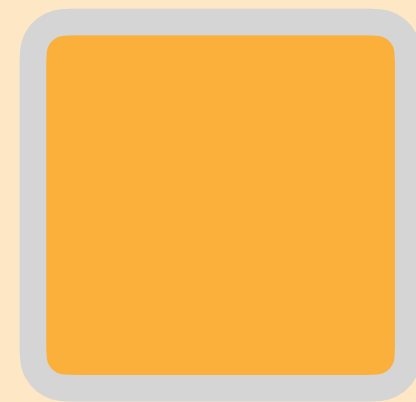
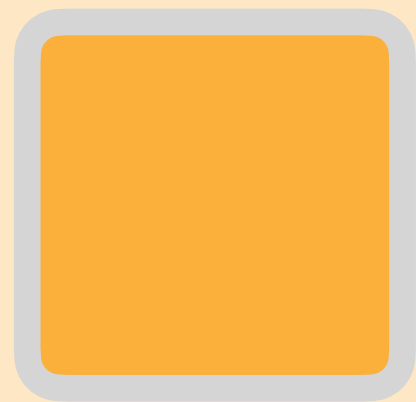
Public REST API



Public REST API



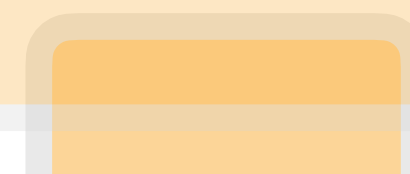
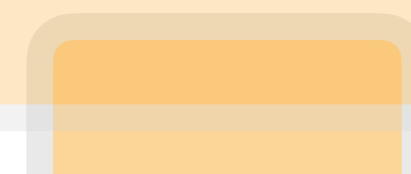
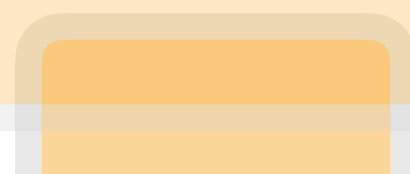
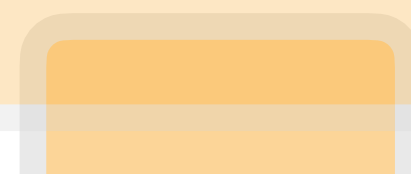
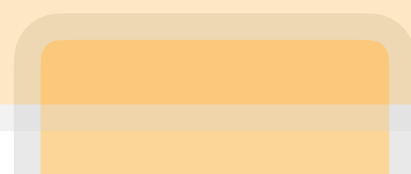
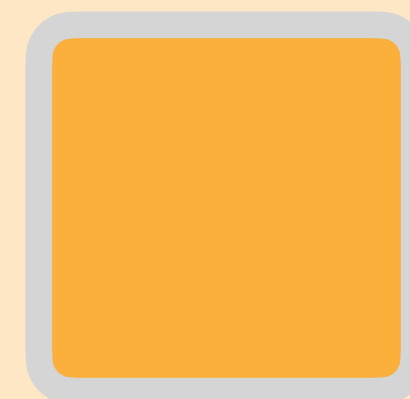
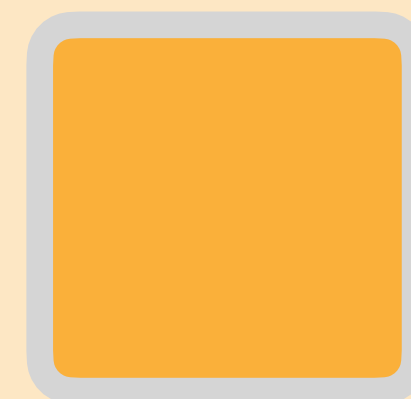
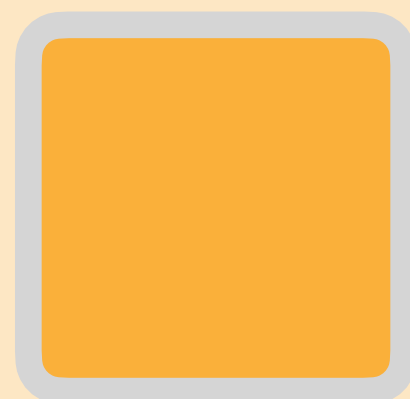
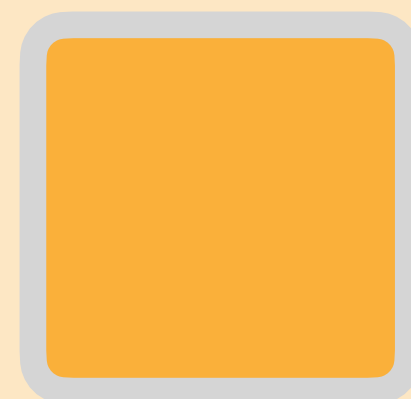
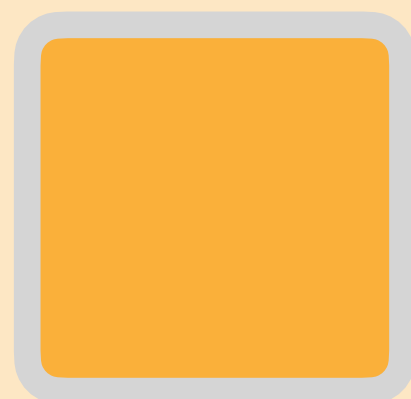
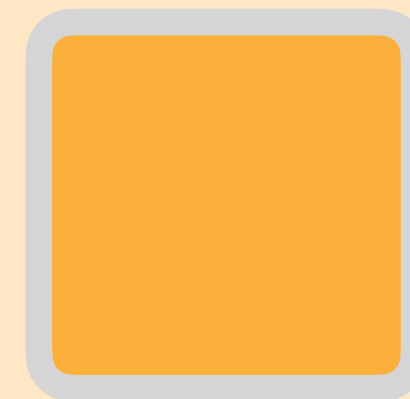
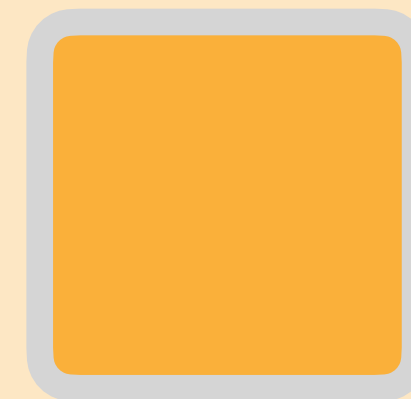
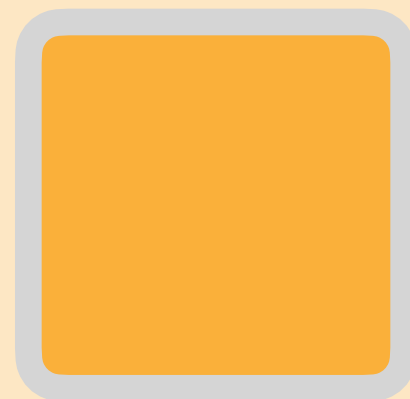
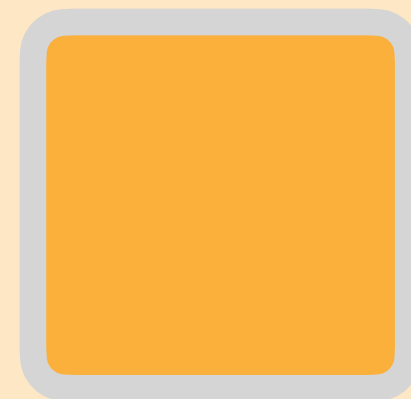
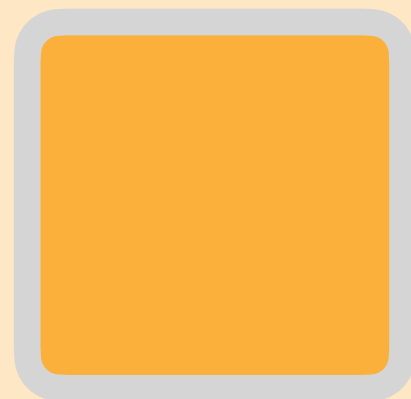
API Gateway



Public REST API



API Gateway



Public REST API

API Gateway

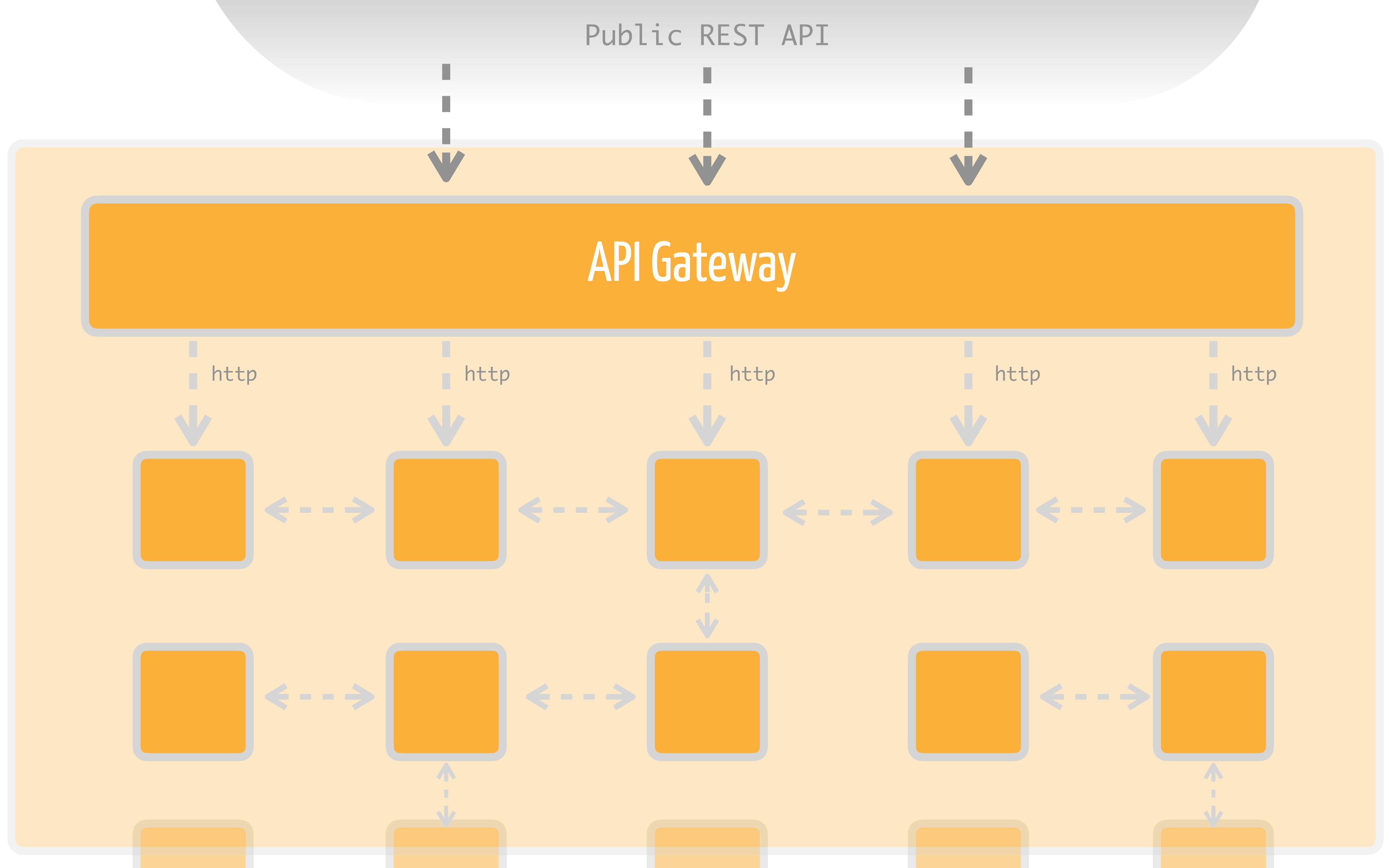
http

http

http

http

http



Public REST API

API Gateway

http

http

http

http

http

gRPC

gRPC

gRPC

gRPC

gRPC

gRPC

gRPC

gRPC

gRPC

gRPC

Public REST API

API Gateway

http

http

http

http

http

gRPC

gRPC

gRPC

gRPC

gRPC

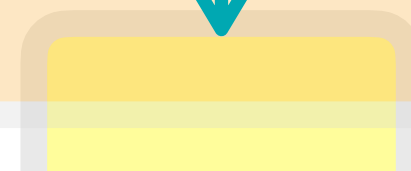
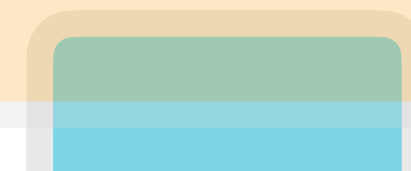
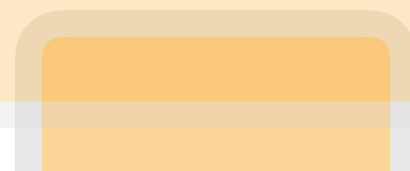
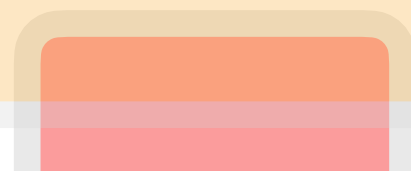
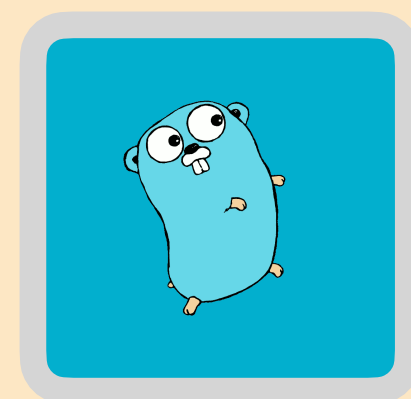
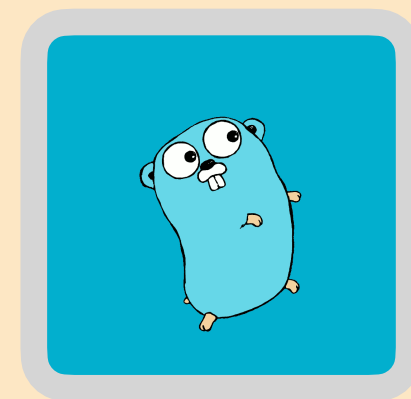
gRPC

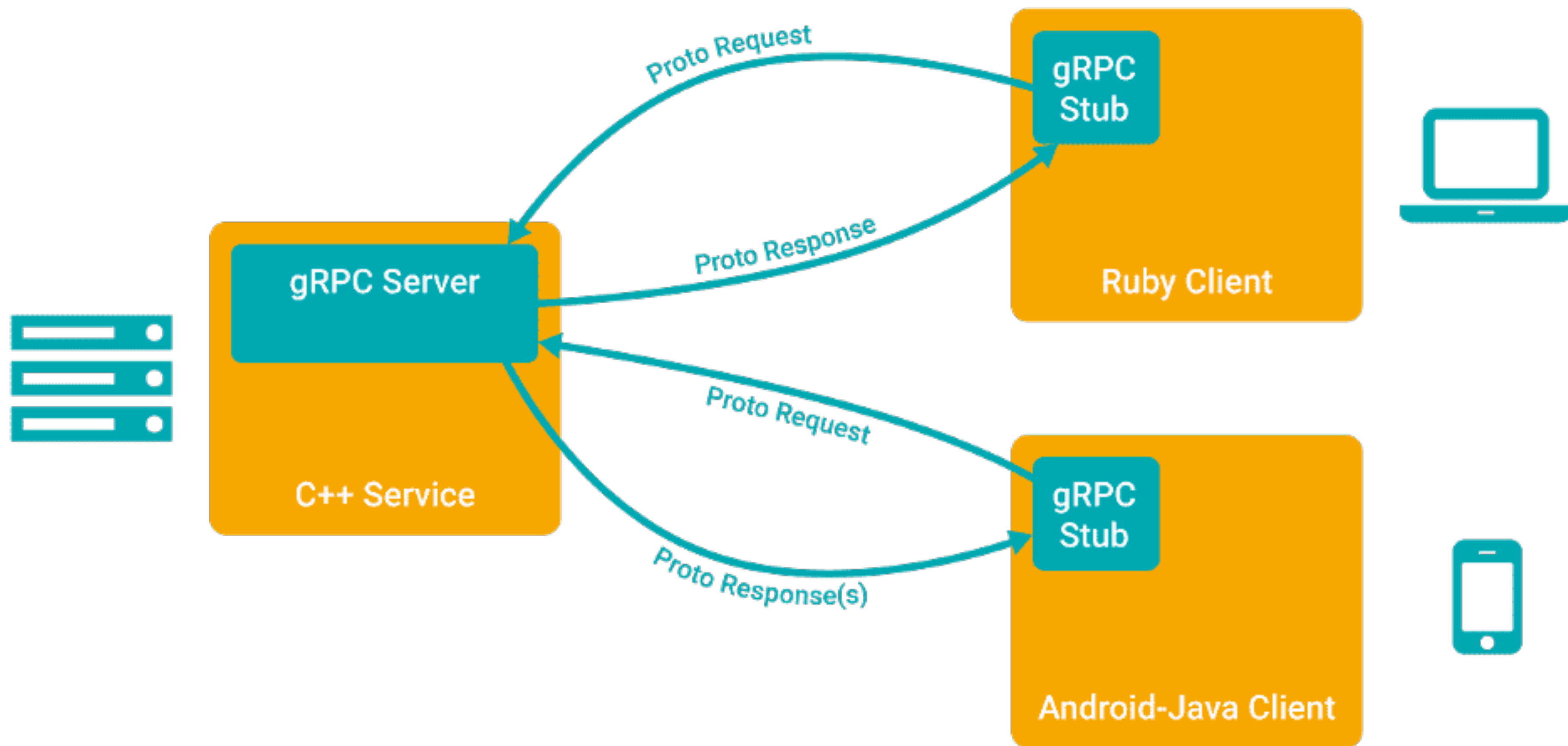
gRPC

gRPC

gRPC

gRPC





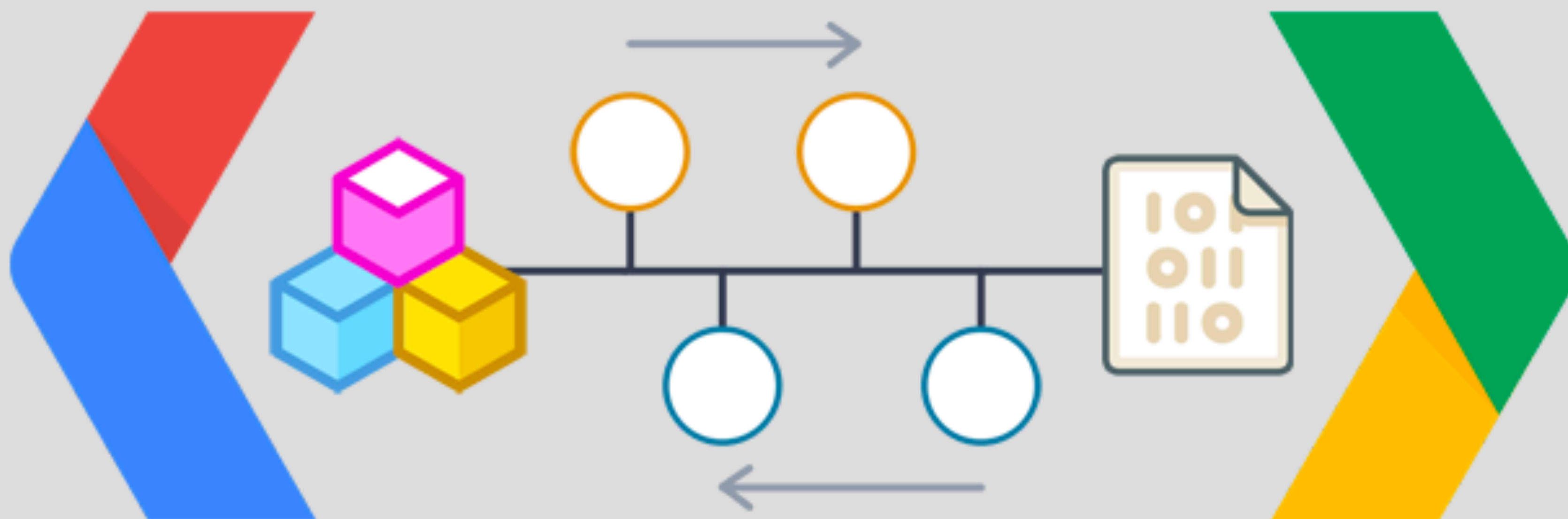


protobuf

Protocol Buffers



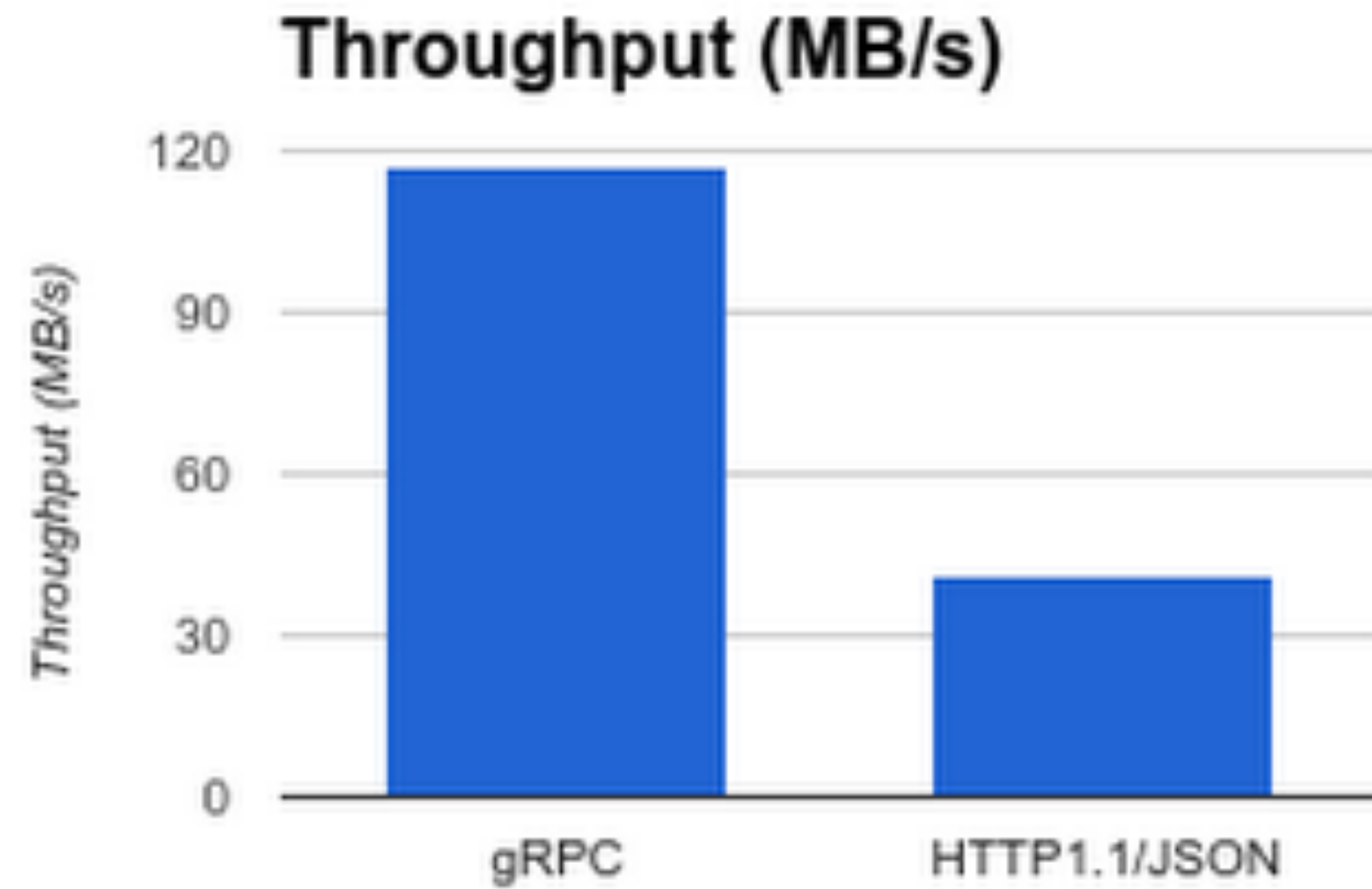
Binary Serialization with



Google Protocol Buffers

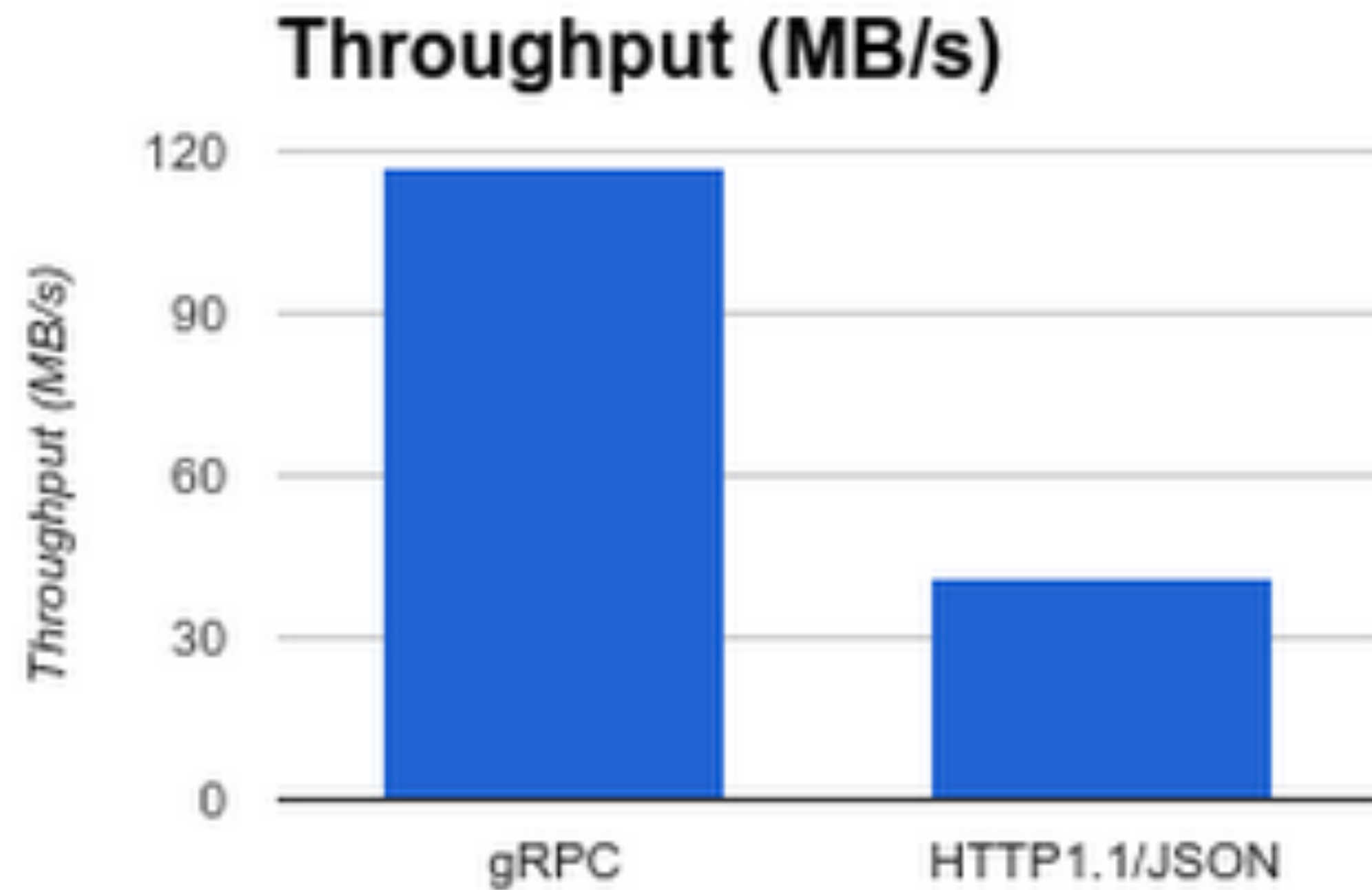
Performance

Performance

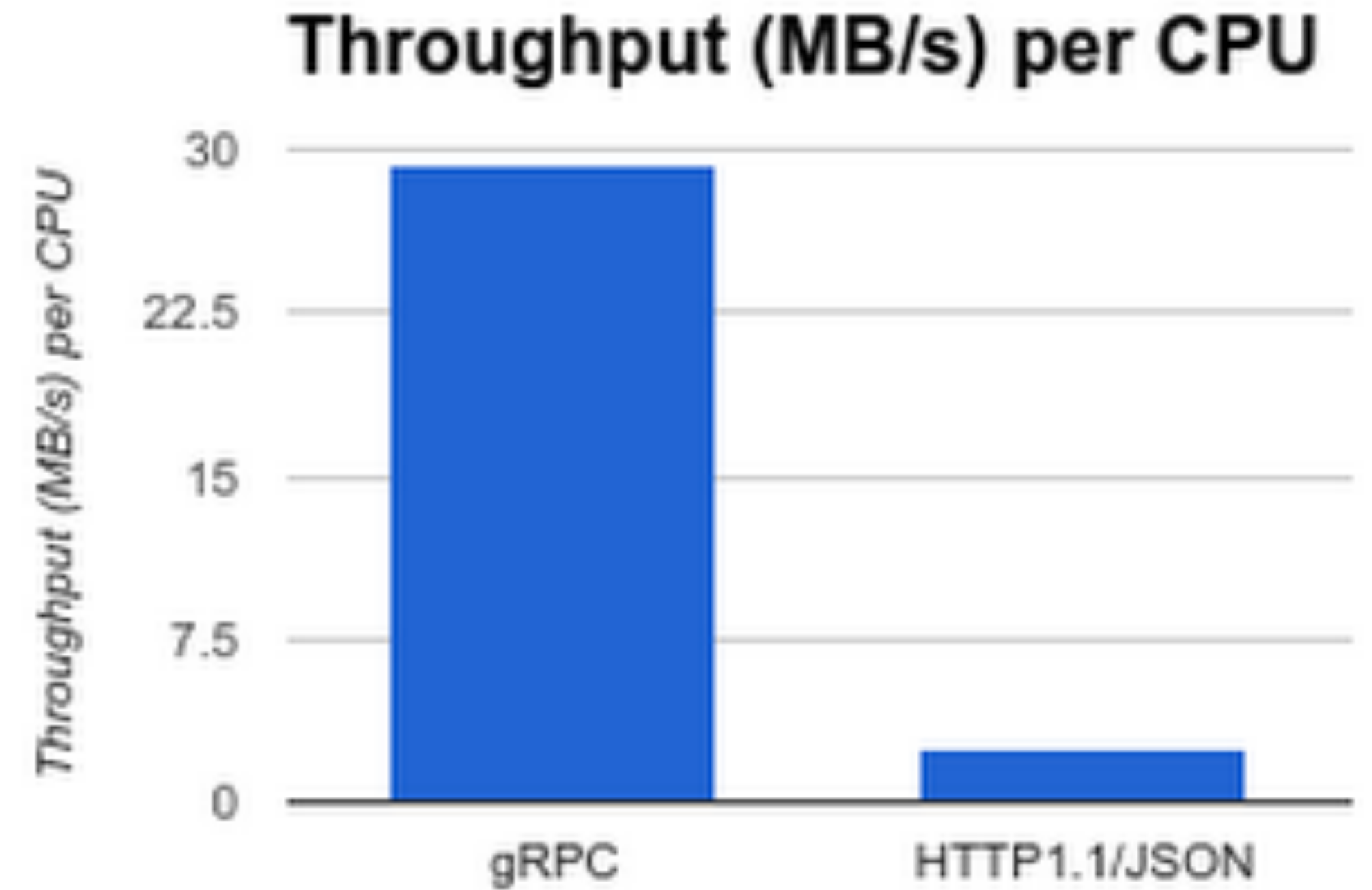


3x increase in throughput

Performance



3x increase in throughput



11x difference per CPU



Seu smartphone agradece!

IDL

(Interface Definition Language)


```
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;  
}
```



```
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;  
}
```

```
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;  
}
```

```
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;  
}
```



```
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;  
}
```

```
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;  
}
```

```
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;  
}
```



```
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;  
}
```

```
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;  
}
```



```
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;  
}
```

```
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;  
}
```

```
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;  
}
```



```
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

1 Define



2 Compile

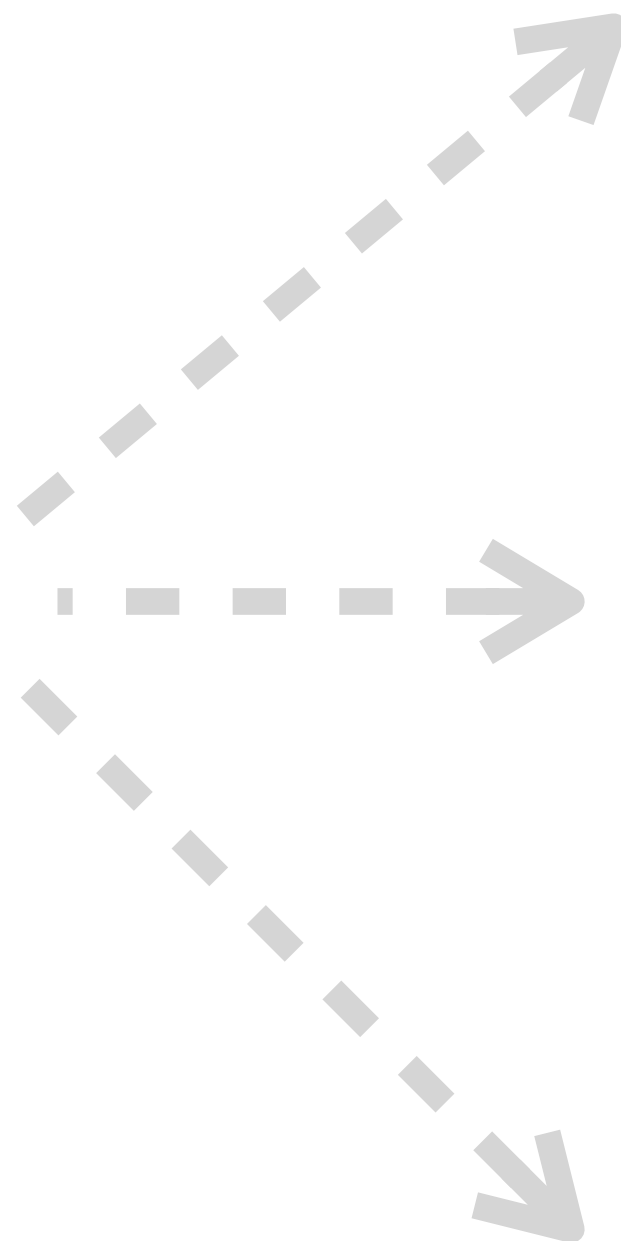
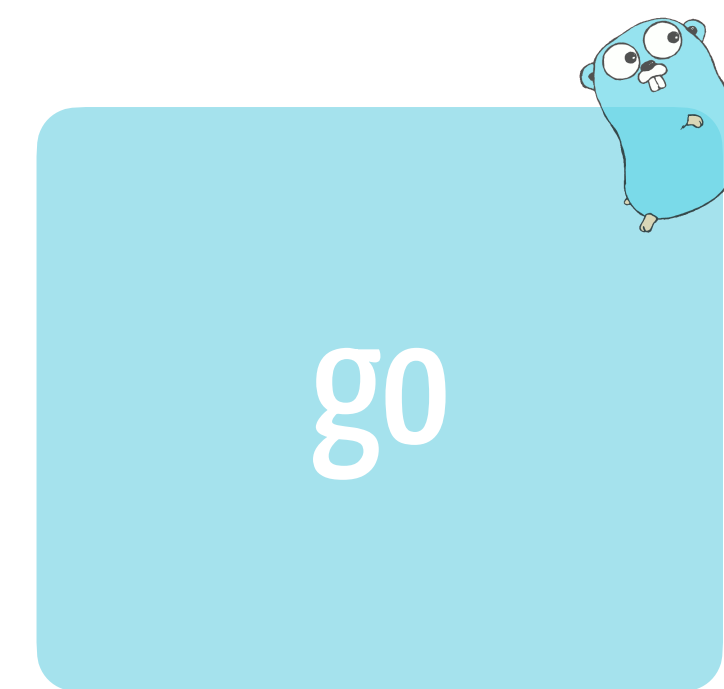


Protobuf Workflow

1 Define

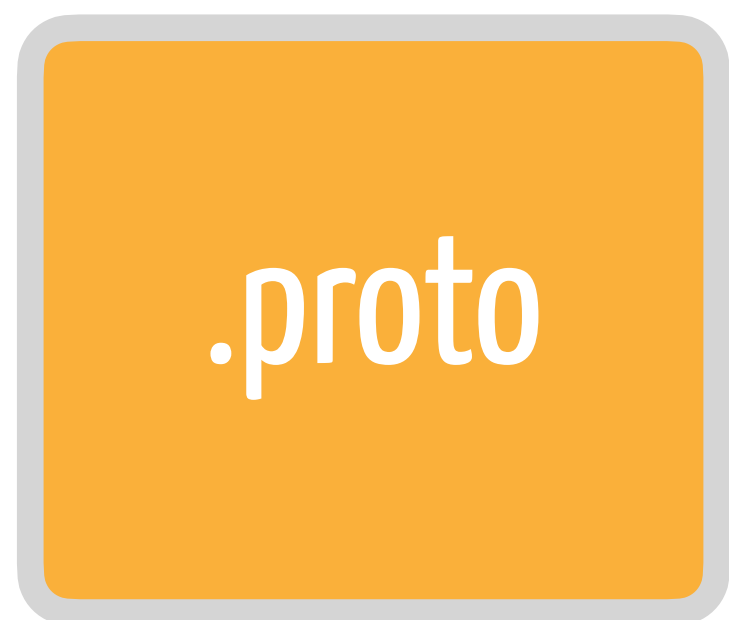


2 Compile

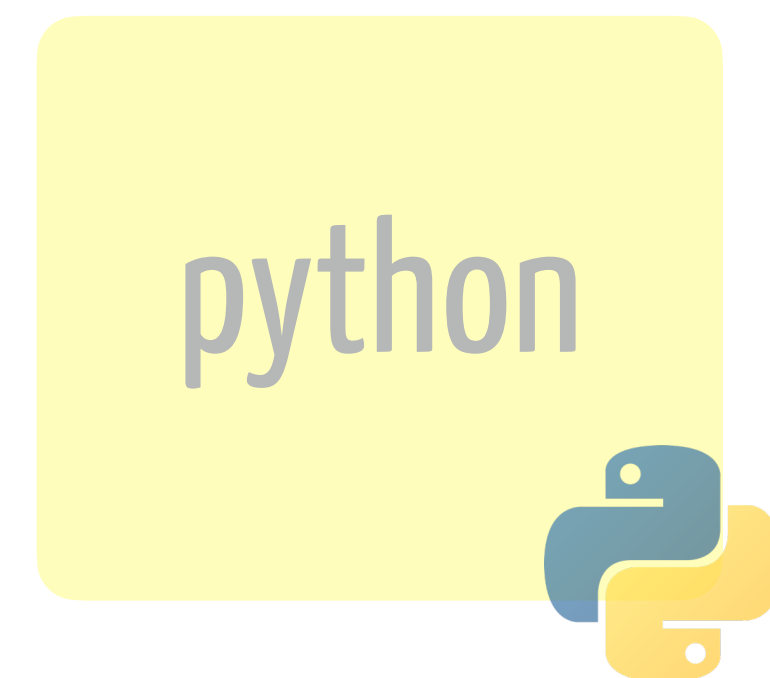
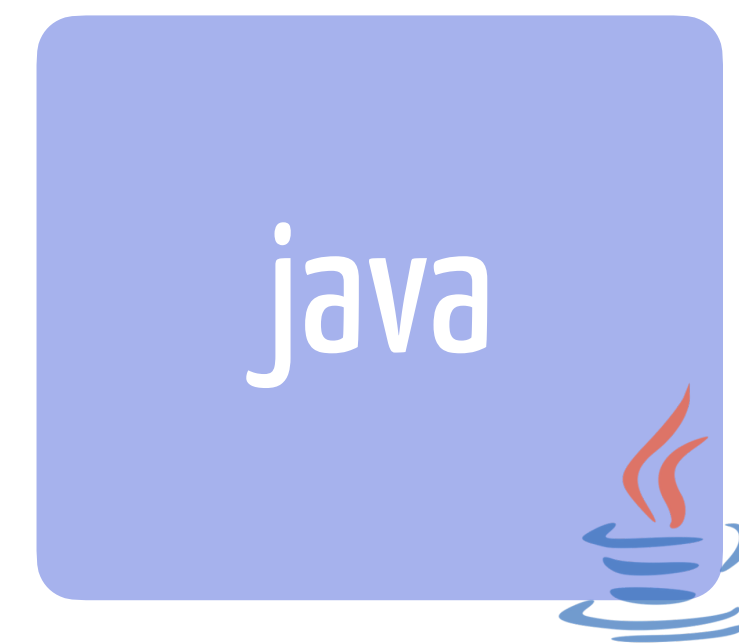
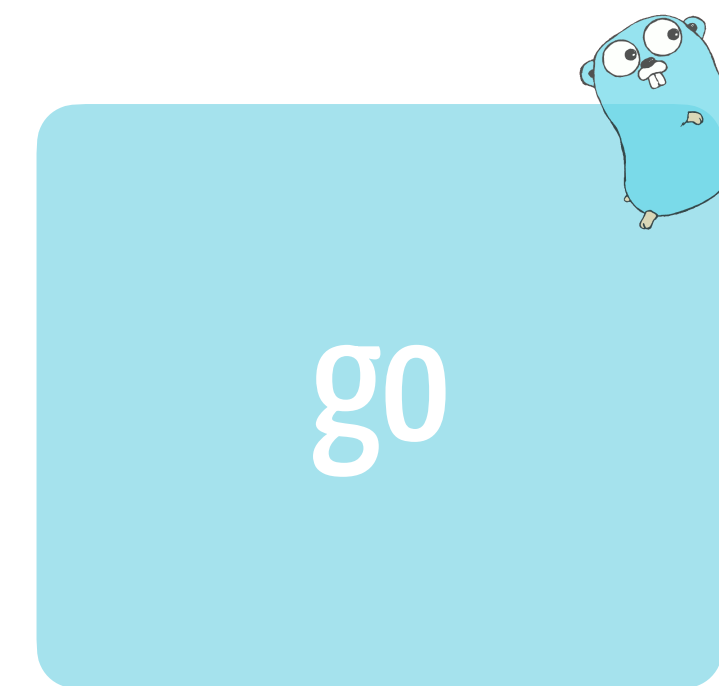
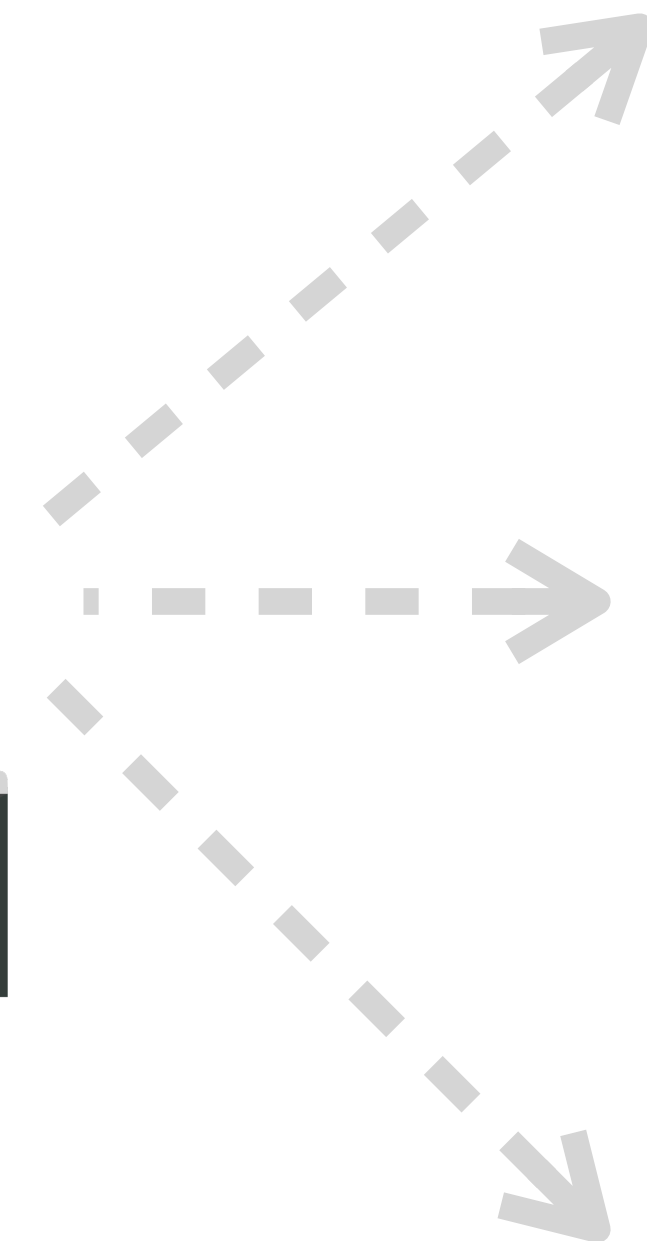


Protobuf Workflow

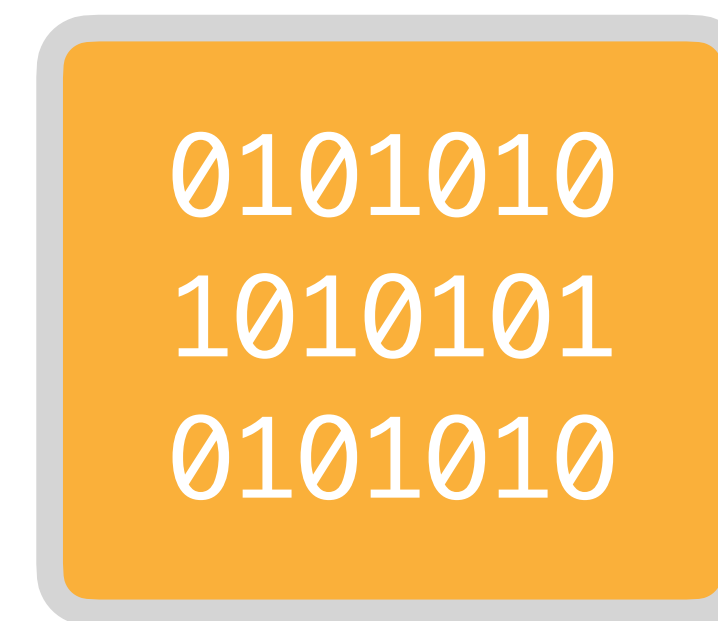
1 Define



2 Compile



3 Serialize



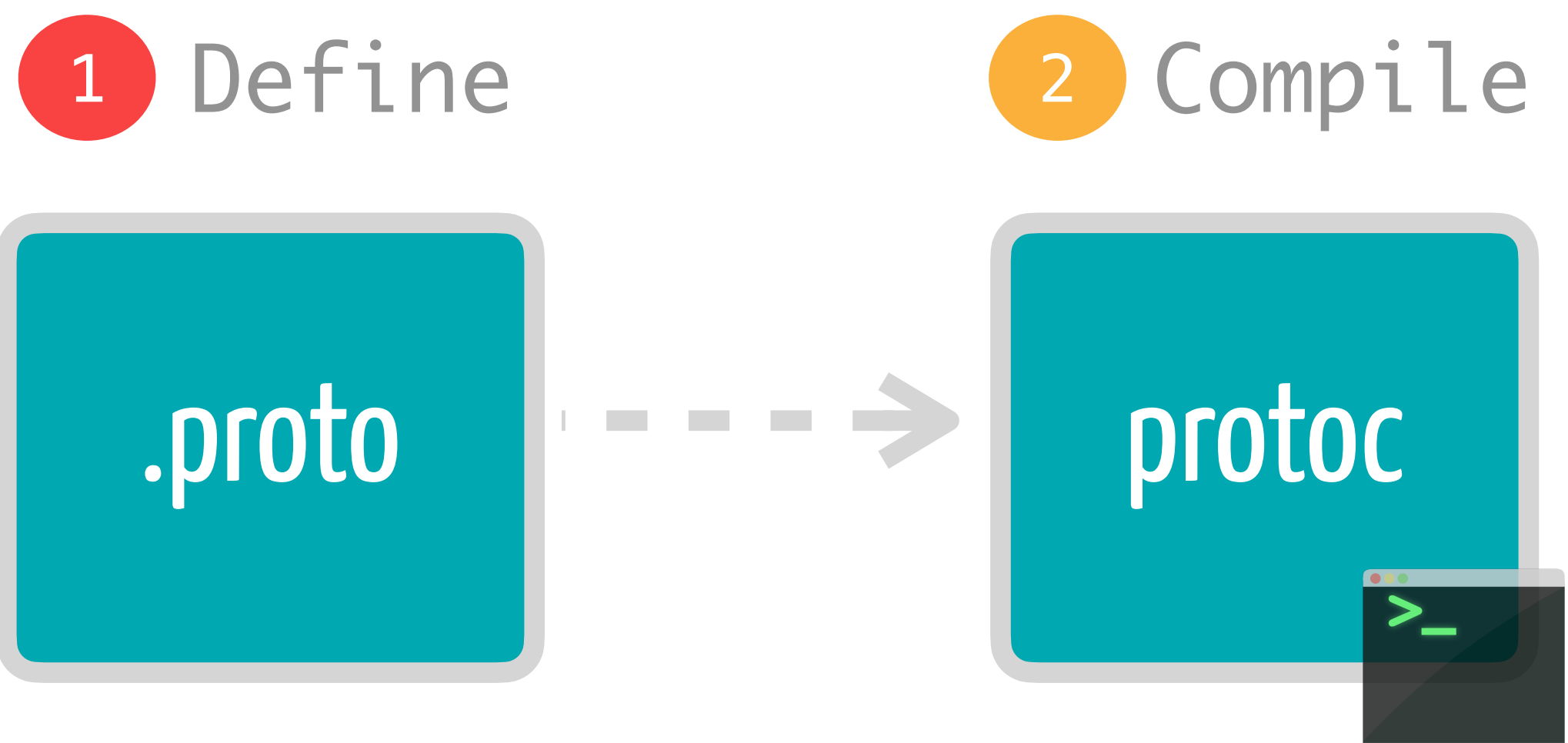
gRPC Workflow

gRPC Workflow

1 Define



gRPC Workflow

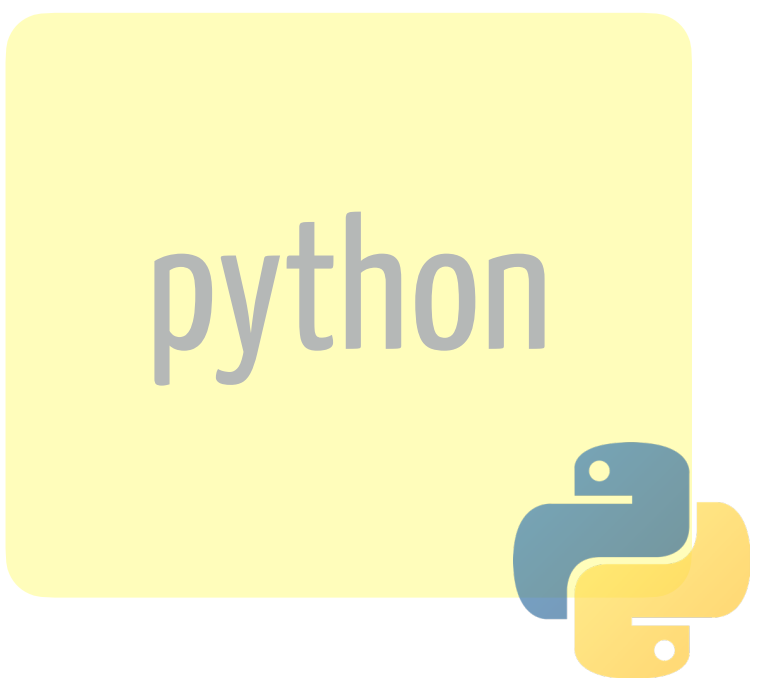
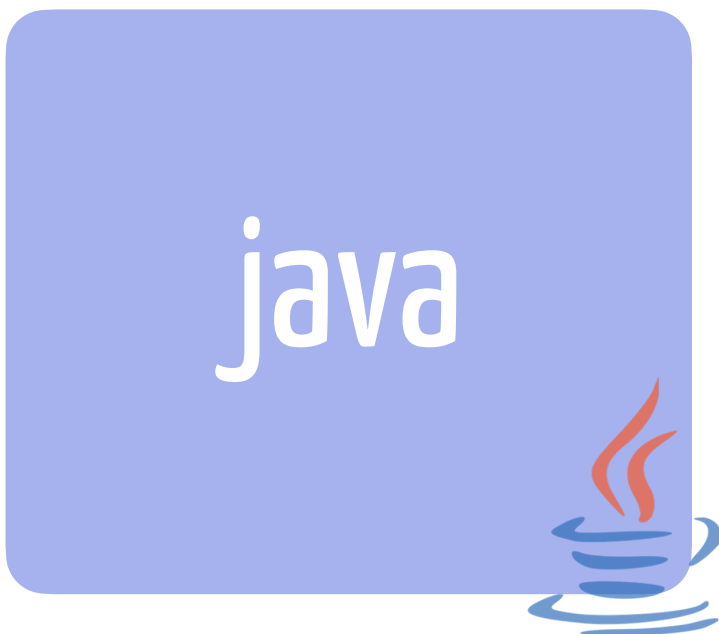
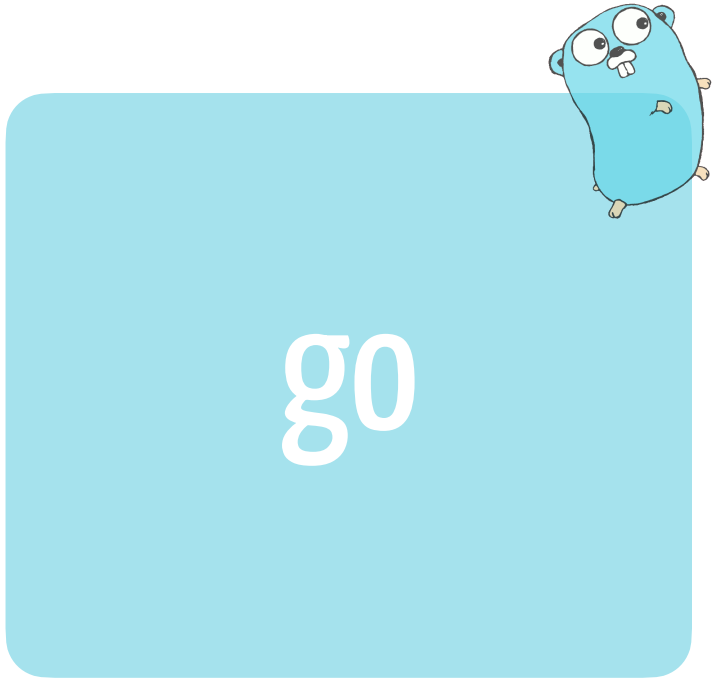
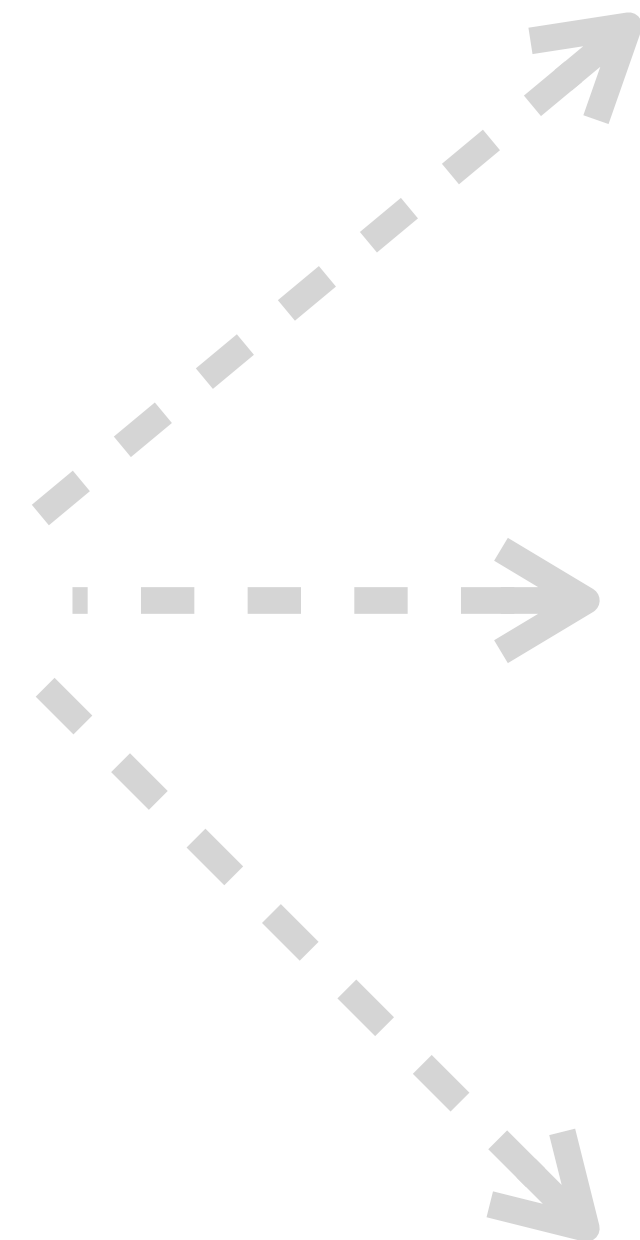
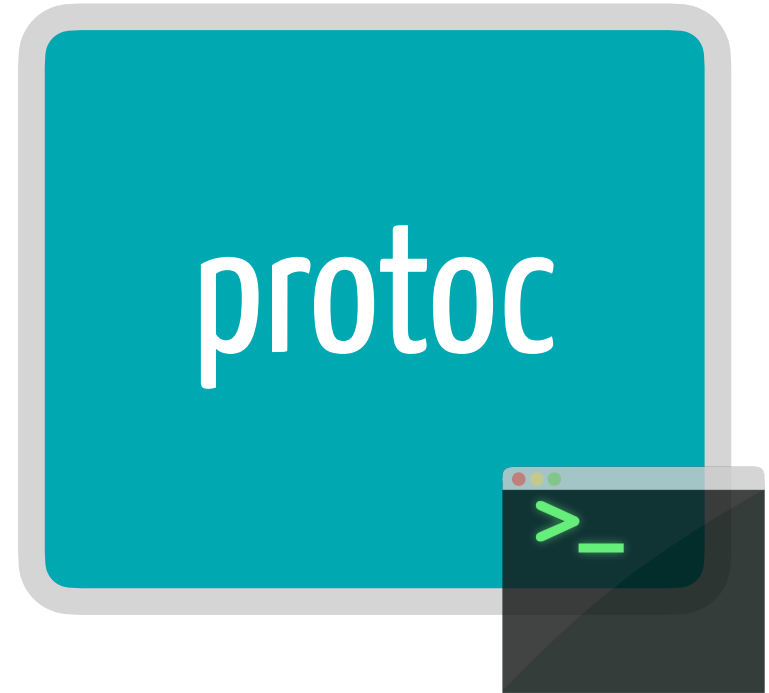


gRPC Workflow

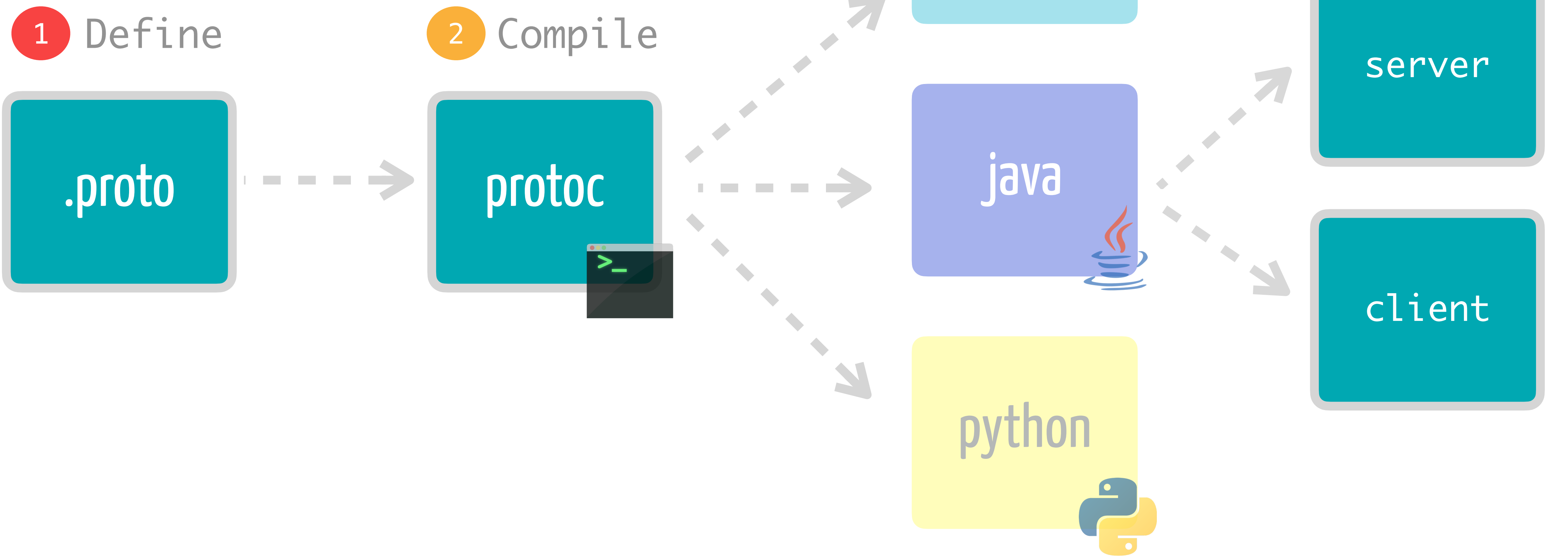
1 Define



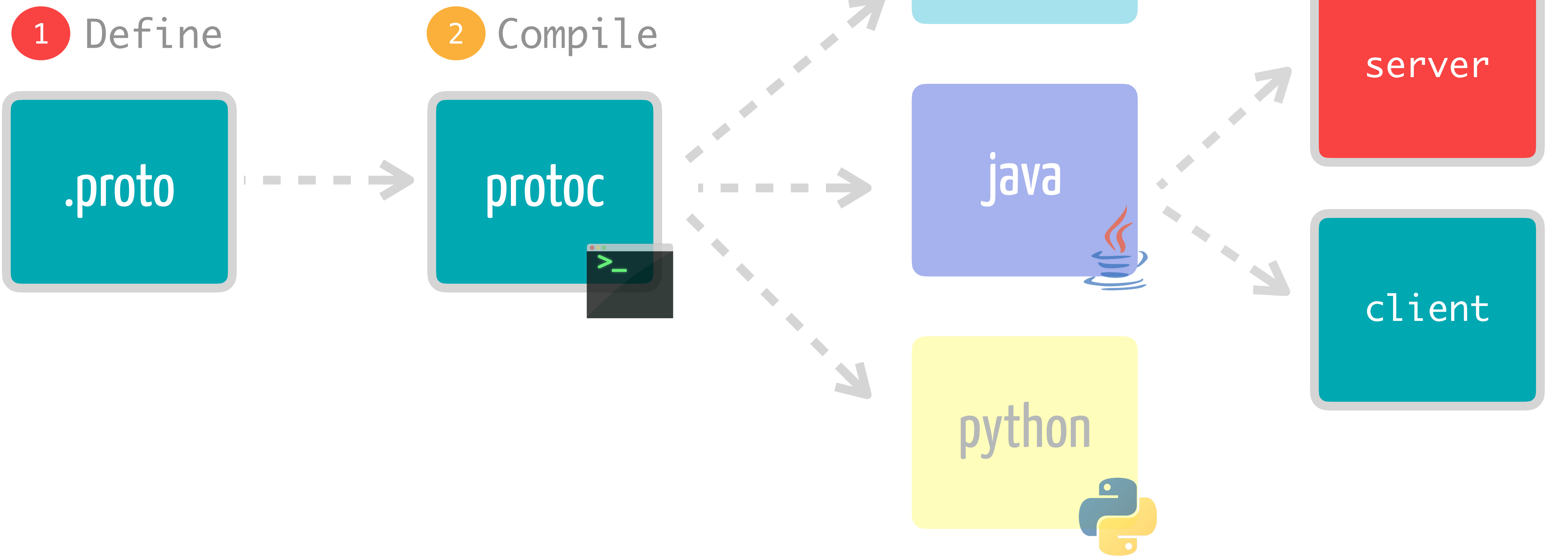
2 Compile



gRPC Workflow



gRPC Workflow




```
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;  
}
```

```
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();  
    }  
}
```



```
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();  
    }  
}
```




```
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();
    }
}
```




```
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();
    }
}
```



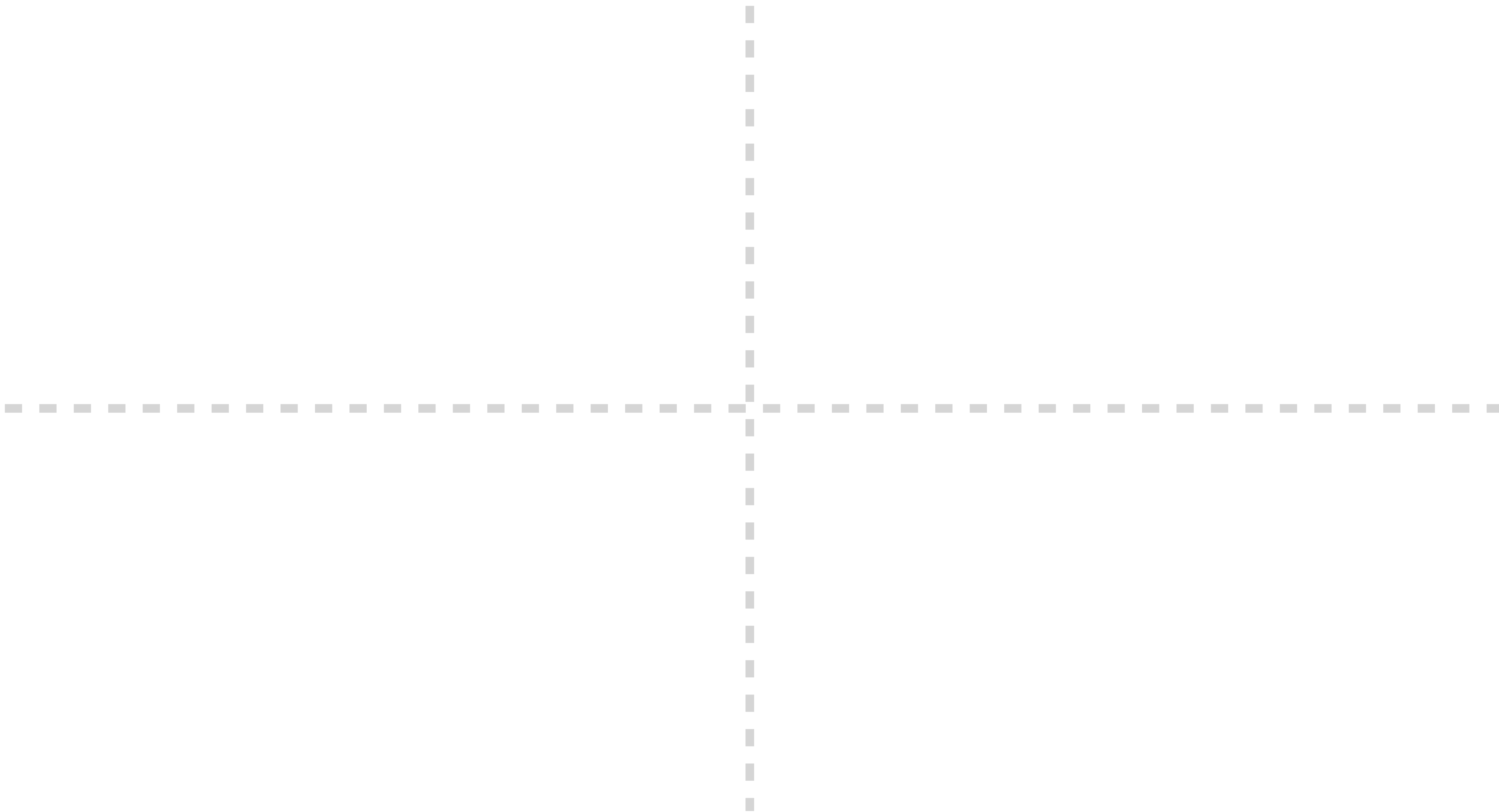
```
public class PaymentEndpoint extends PaymentServiceGrpc.PaymentServiceImplBase {

    @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();
    }
}
```

Padrões de comunicação



1

2

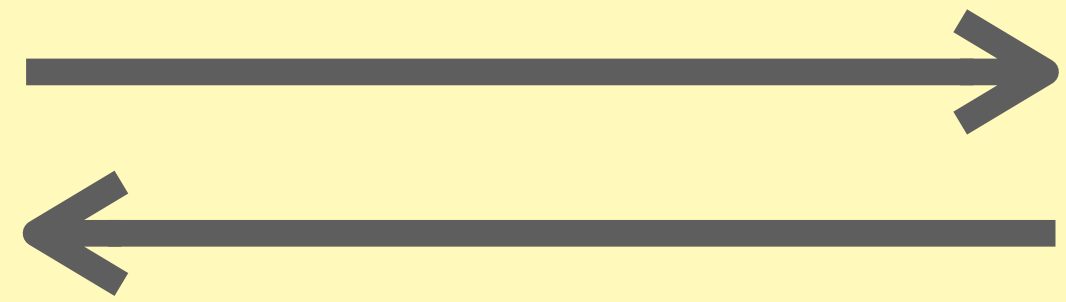
3

4

Simple RPC (Unary RPC)



client



server

2

3

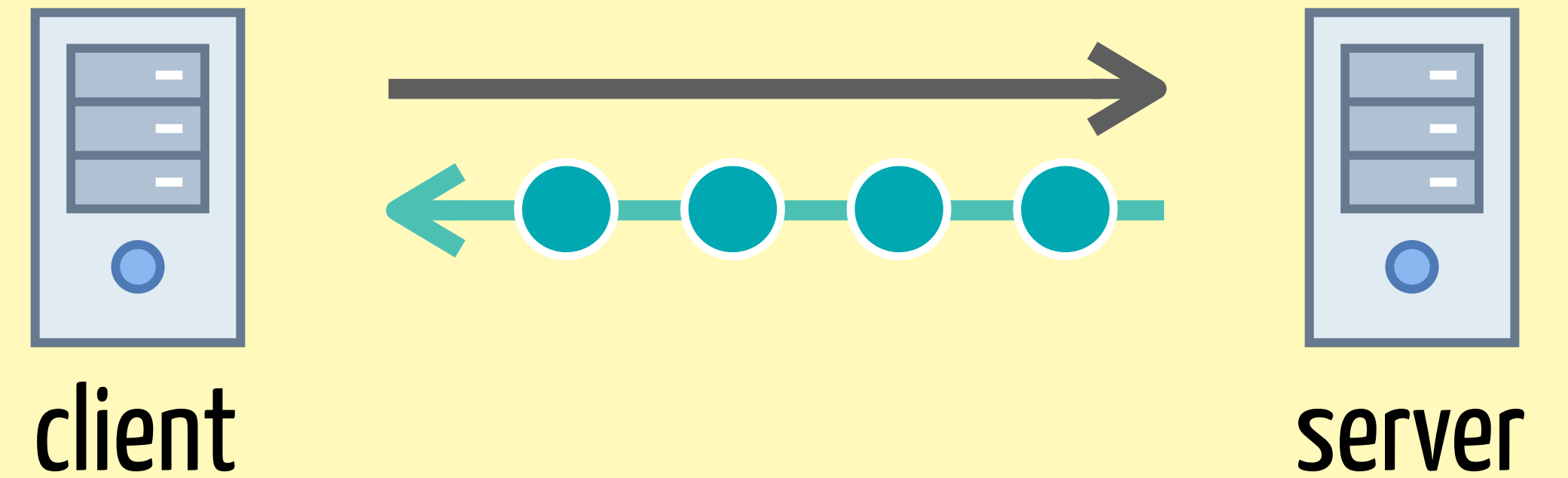
4

Simple RPC (Unary RPC)



3

Server-side Streaming RPC

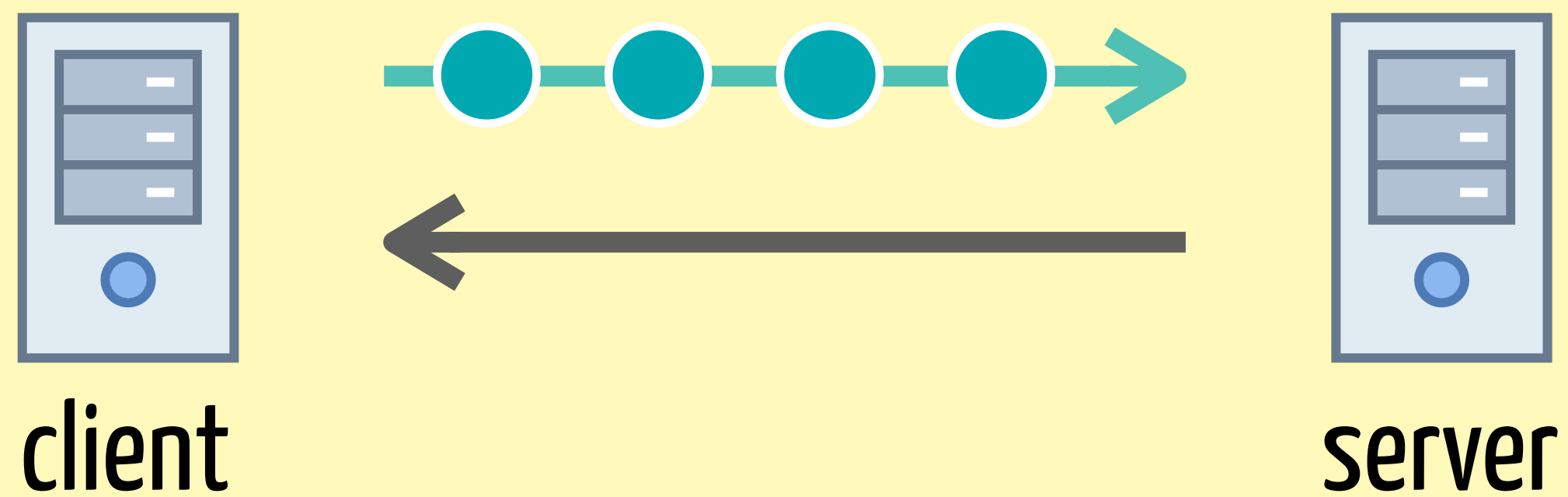
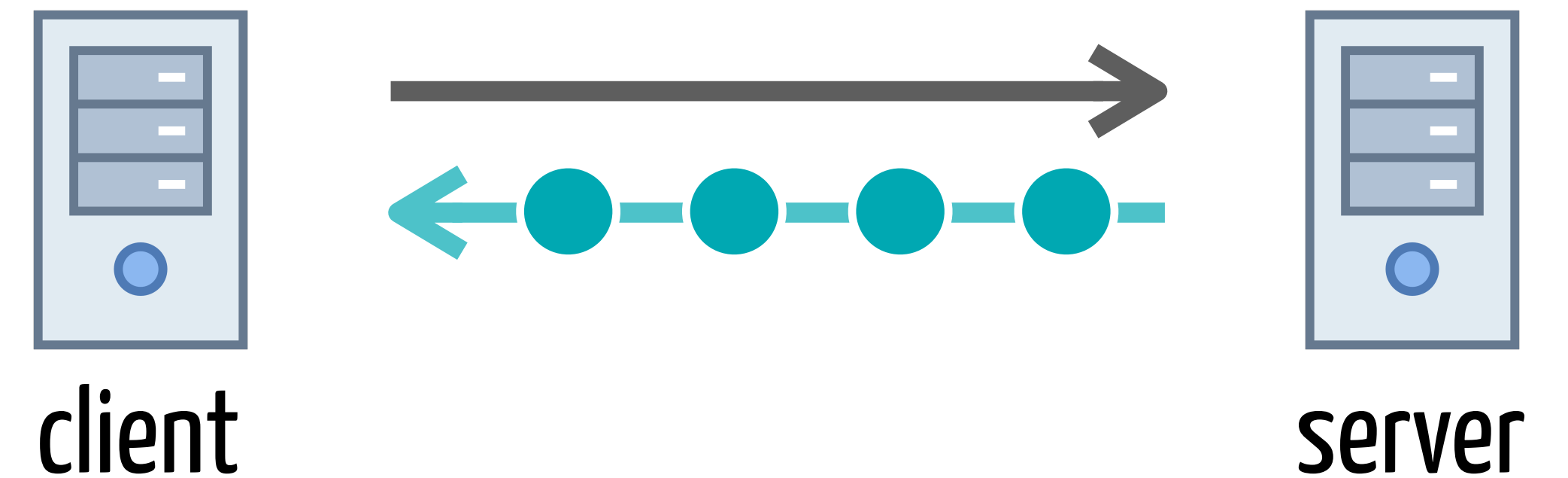


4

Simple RPC (Unary RPC)



Server-side Streaming RPC



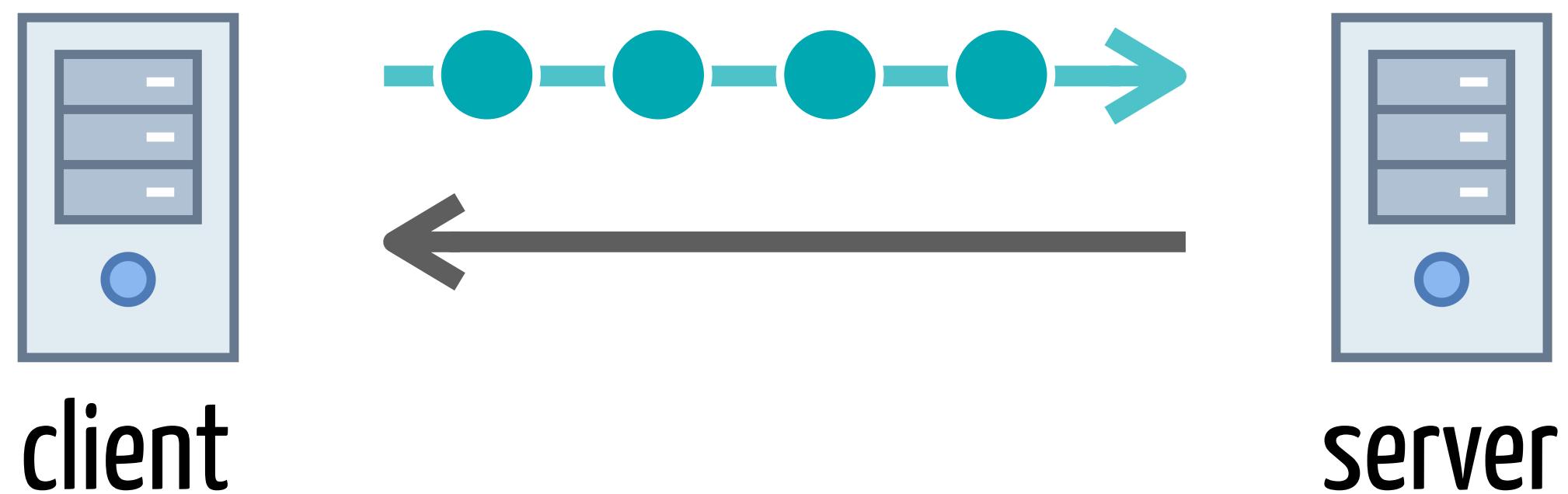
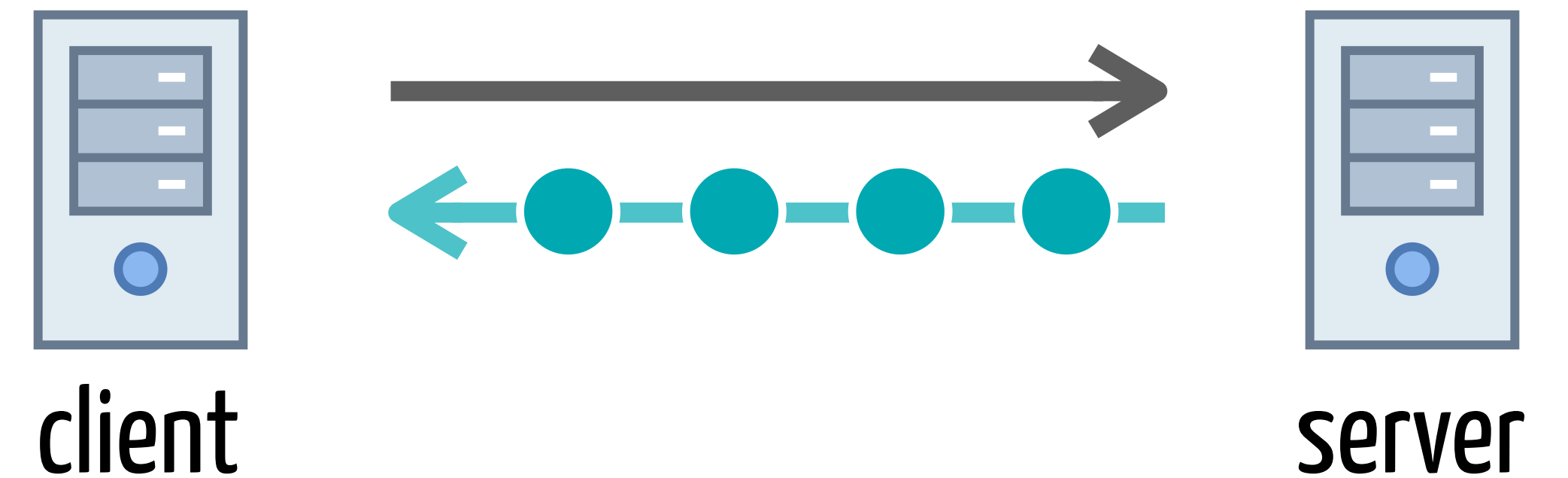
Client-side Streaming RPC

4

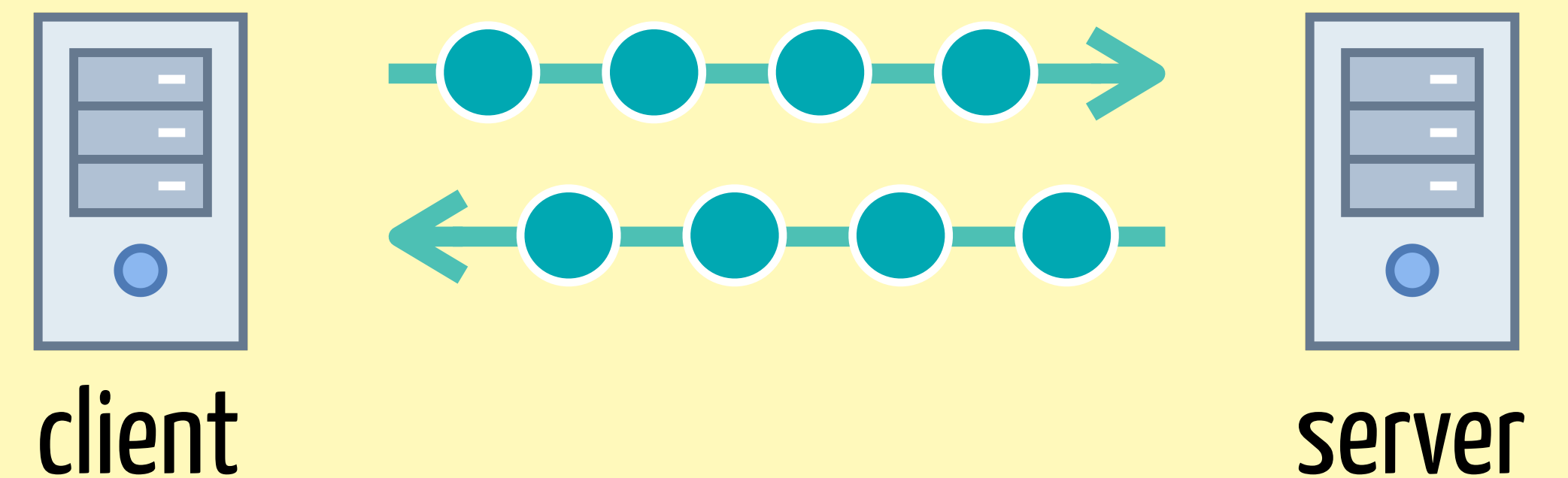
Simple RPC (Unary RPC)



Server-side Streaming RPC



Client-side Streaming RPC



Bidirectional Streaming RPC

```
syntax = "proto3";
```

```
service PaymentService {
```

```
    rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
```

```
}
```

```
syntax = "proto3";  
  
service PaymentService {
```

Simple RPC (Unary RPC)

```
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
```

```
}
```



```
syntax = "proto3";
```

```
service PaymentService {
```

```
    rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
```

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

```
}
```

```
syntax = "proto3";
```

```
service PaymentService {
```

```
  rpc processPayment(PaymentRequest)
```

Server-side Streaming RPC

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

```
}
```



```
syntax = "proto3";
```

```
service PaymentService {
```

```
    rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
```

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

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

```
}
```

```
syntax = "proto3";
```

```
service PaymentService {
```

```
  rpc pr
```

```
  rpc li
```

Client-side Streaming RPC

```
  {}
```

```
TransactionResponse) {}
```

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

```
}
```

```
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) {}
```

```
}
```



```
syntax = "proto3";
```

```
service PaymentService {
```

```
  rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
```

```
  rpc listenToTransactions returns (stream PaymentResponse) {}
```

```
  rpc processPaymentBatch returns (stream BatchResponse) {}
```

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

```
}
```

Bidirectional Streaming RPC

```
syntax = "proto3";
```

```
service PaymentService {
```

```
    rpc processPayment(PaymentRequest) returns (PaymentResponse) {}
```

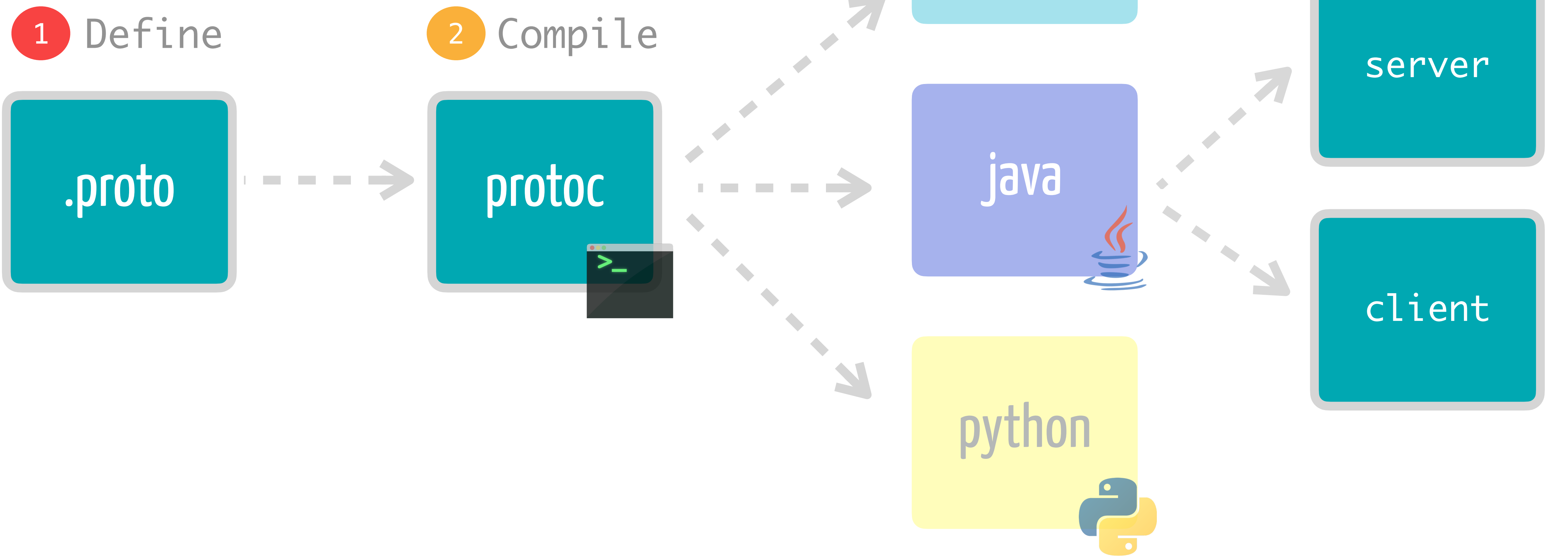
```
    rpc listenToTransactions(TransactionRequest) returns (stream TransactionResponse) {}
```

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

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

```
}
```

gRPC Workflow



CONCLUINDO



Microservices





Microservices



Mobile



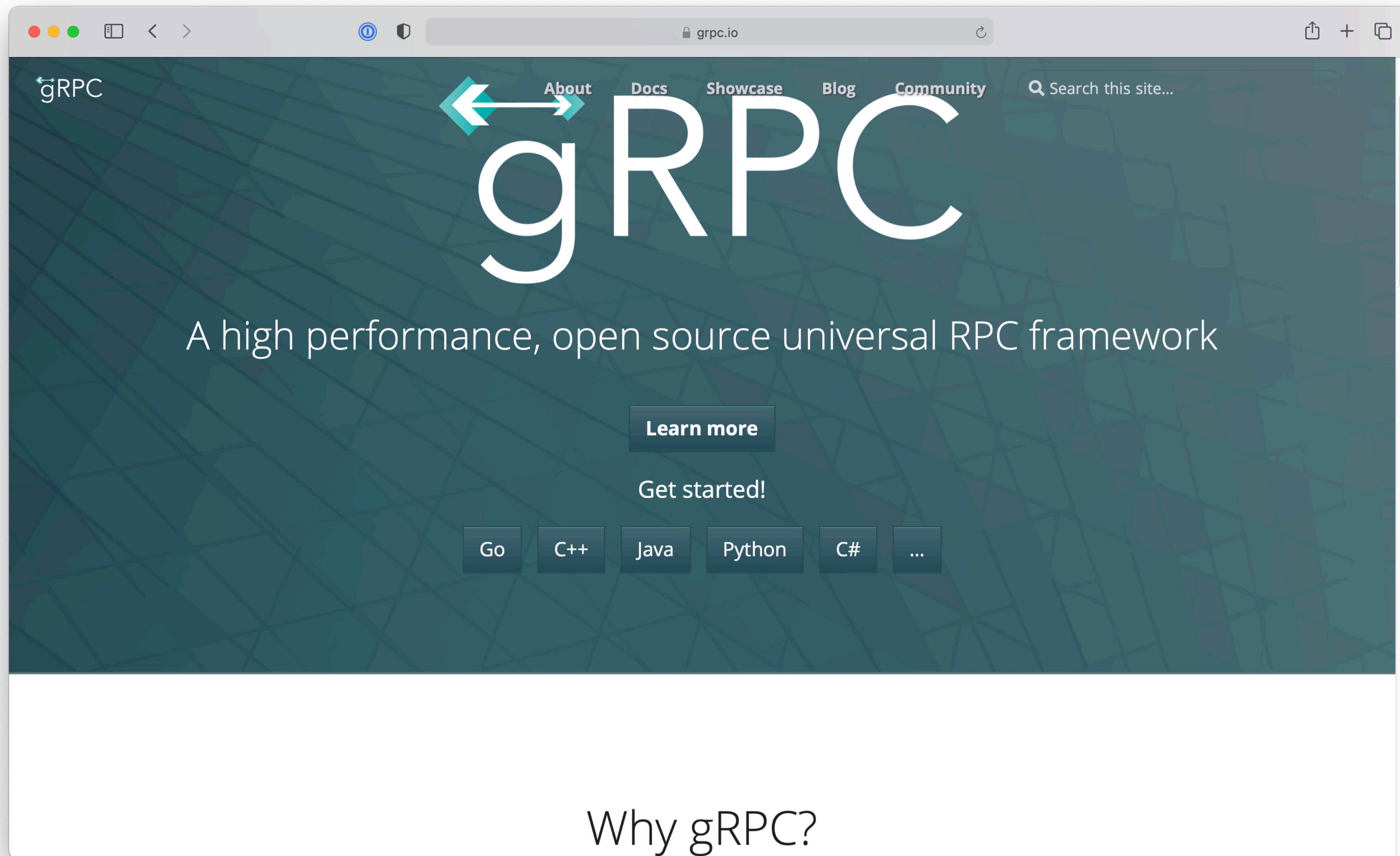
Microservices



Mobile



Realtime



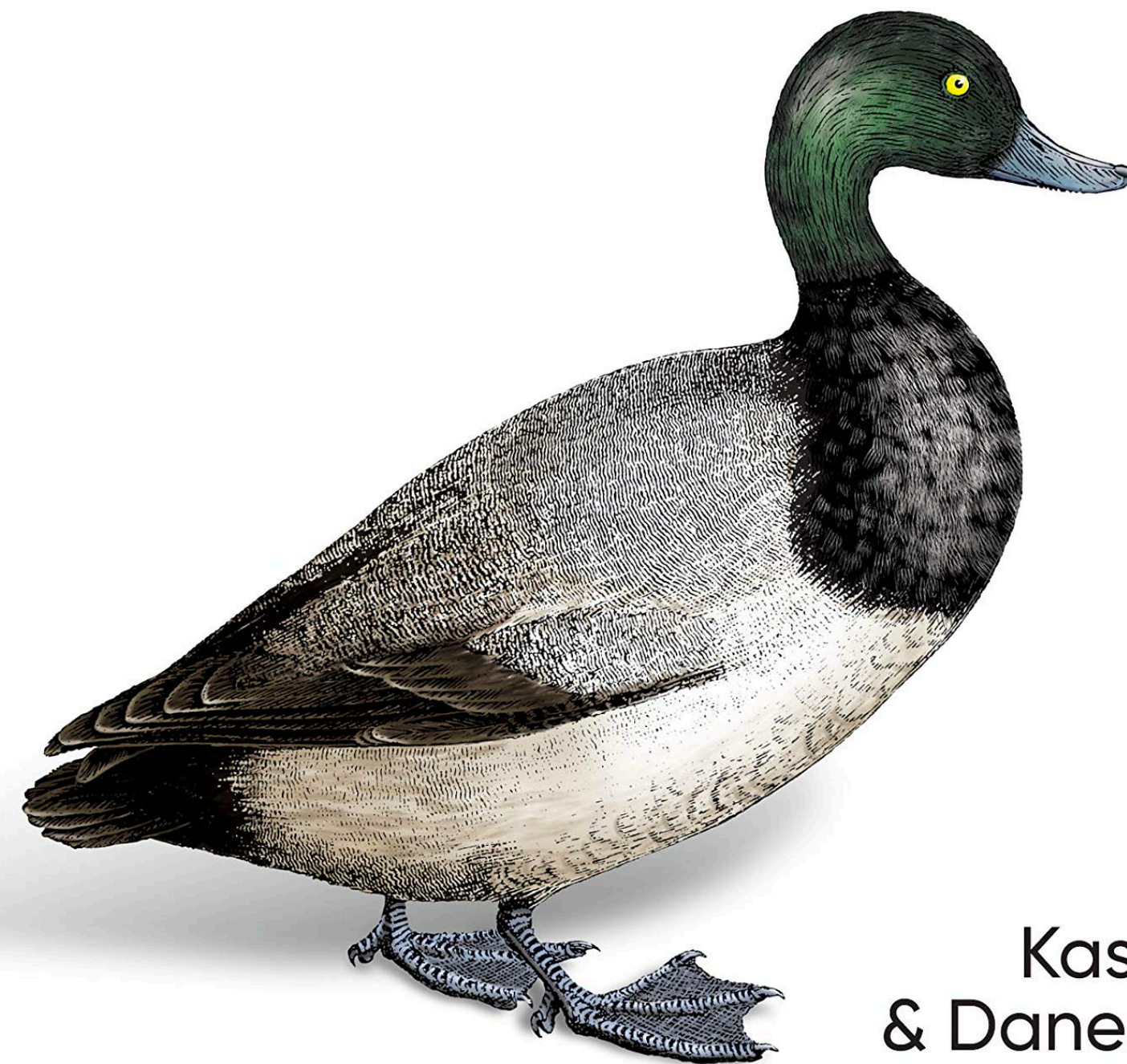
Why gRPC?

<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

