Full Stack Application Development

gRPC

Akshaya Ganesan

Agenda

- gRPC

gRPC- An RPC library and framework

- Problems with RPC- laid the motivation for gRPC
- In 2015, Google released gRPC , a modern, open source, high-performance remote procedure call (RPC) framework that can run anywhere.
- An interprocess communication technology that allows you to connect, invoke, operate, and debug distributed heterogeneous applications as easily as making a local function call.

gRPC

- What does the "g" in gRPC actually stand for?
  - Its not Google
  - Google changes the meaning of the "g" for each version
  - Refer to the README doc

gRPC- Architecture

- A service Definition is created.
- Using that service definition, the server-side code known as a *server skeleton* is generated
- On the server side, the server implements the service, and runs a gRPC server to handle client calls.
- On the client side, the client has a stub that provides the same methods as the server.
- gRPC uses **Protocol Buffers** as
  - Interface Definition Language (**IDL**) and
  - message interchange format.

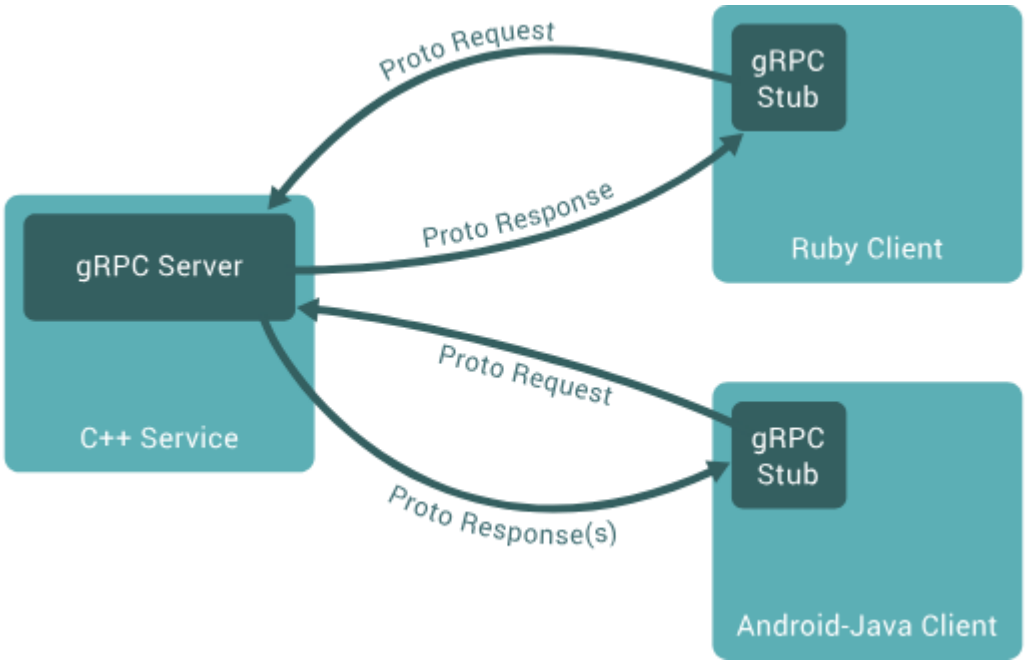


Image Source: https://grpc.io/docs/what-is-grpc/introduction/

Protocol Buffers

- gRPC uses protocol buffers as the Interface Definition Language to define the service interface
- Protocol Buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.
- The service interface definition is specified in a proto file -an ordinary text file with a .proto extension.
- The service definition specified in the .proto file, is used by both the server and client sides to generate the code.

Protocol Buffers

- Protocol buffer data is structured as messages.
- Each message is a small logical record of information containing a series of name-value pairs called fields.
- Example:

```
message Person {
  string name = 1;
  int id = 2;
  string description = 3;
}
```

Protocol Buffers

- You define gRPC services in ordinary proto files, with RPC method parameters and return types specified as protocol buffer messages.

Example:

```
service ProductInfo {
   rpc addProduct(Product) returns (ProductID);
   rpc getProduct(ProductID) returns (Product);
}
message Product {
   string id = 1;
   string name = 2;
   string description = 3;
}
message ProductID {
   string value = 1;
}
```

gRPC Service

- The service definition can be used to generate the server or client-side code using the protocol buffer compiler protoc.
- Since gRPC service definitions are language agnostic, you can generate clients and servers for any supported language

gRPC Service

- Server Side:
  - Implement the service logic of the generated service skeleton by overriding the service base class.
  - Run a gRPC server to listen for requests from clients and return the service responses.
- Client Side:
  - The client stub provides the same methods as the server, which your client code can invoke.
  - The client stub translates them to remote function invocation network calls that go to the server side.

Why gRPC?

- Efficient for inter-process communication
  - gRPC implements protocol buffers on top of HTTP/2
- Simple, well-defined service interfaces and schema
  - contract-first approach
- Strongly typed- protocol buffers to define gRPC services
- Polyglot
- Duplex streaming
- Built-in commodity features

gRPC in Real World

- With the adoption of gRPC, Netflix has seen a massive boost in developer productivity.
- Etcd uses a gRPC user-facing API to leverage the full power of gRPC.
- Dropbox has switched to gRPC

gRPC- Disadvantages

- It may not be suitable for external-facing services.
- Drastic service definition changes are a complicated development process
- The ecosystem is still evolving.

gRPC- Flow

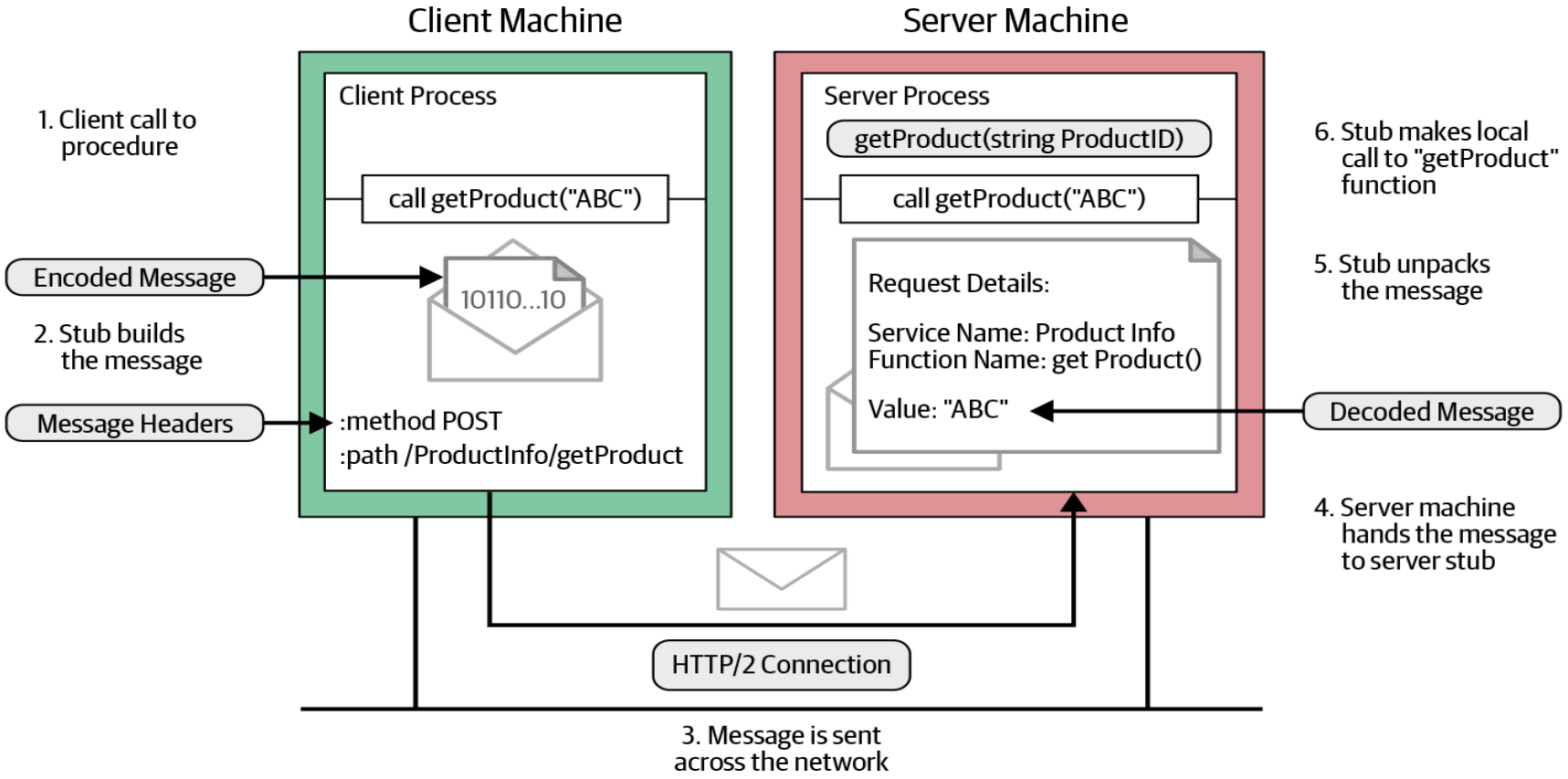


Image Source: gRPC: Up and Running by Kasun Indrasiri, Danesh Kuruppu

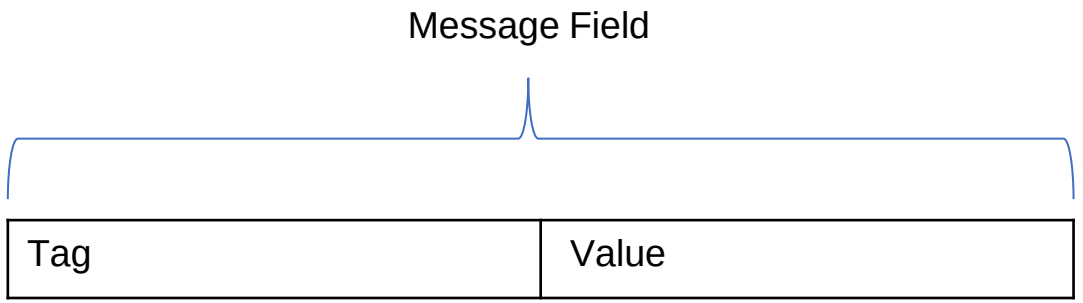
Message Encoding

message ProductID {

string value = 1;

}

Let’s say we need to get product details for product ID 15;



Message Encoding

- Tag value = (field_index << 3) | wire_type
- **Wire type**
  - 0 for Varint int32, int64, uint32, uint64, sint32, sint64, bool, enum
  - 1 for 64-bit fixed64, sfixed64, double
  - 2 for Length-delimited string, bytes, embedded messages, packed repeated fields
  - 3 for Start group groups (deprecated)
  - 4 for End group groups (deprecated)
  - 5 for 32-bit fixed32, sfixed32, float

- Tag value = (00000001 << 3) | 00000010
- = 000 1010

Length-Prefixed Message Framing

- gRPC uses a message-framing technique called length-prefix framing.
- Length-prefix is a message-framing approach that writes the size of each message before writing the message itself.

gRPC over HTTP/2

- gRPC uses HTTP/2 as its transport protocol to send messages over the network.
- This is one of the reasons why gRPC is a high-performance RPC framework.
- The gRPC channel represents a connection to an endpoint, which is an HTTP/2 connection.
- Messages that are sent in the remote call are sent as HTTP/2 frames.
- A frame may carry one gRPC length-prefixed message, or if a gRPC message is quite large it might span multiple data frames.

Communication patterns

- The four fundamental communication patterns used in gRPC-based applications are
  - unary RPC (simple RPC),
  - server-side streaming,
  - client-side streaming, and
  - bidirectional streaming

Simple RPC (Unary RPC)

- In simple RPC, the client invokes a remote function of a server.
- The client sends a single request to the server and gets a single response that is sent along with status details and trailing metadata.
- Example:
- OrderManagement service for an online retail application
- rpc getOrder(google.protobuf.StringValue) returns (Order);

```
message Order {
string id = 1;
repeated string items = 2;
string description = 3;
float price = 4;
}
```

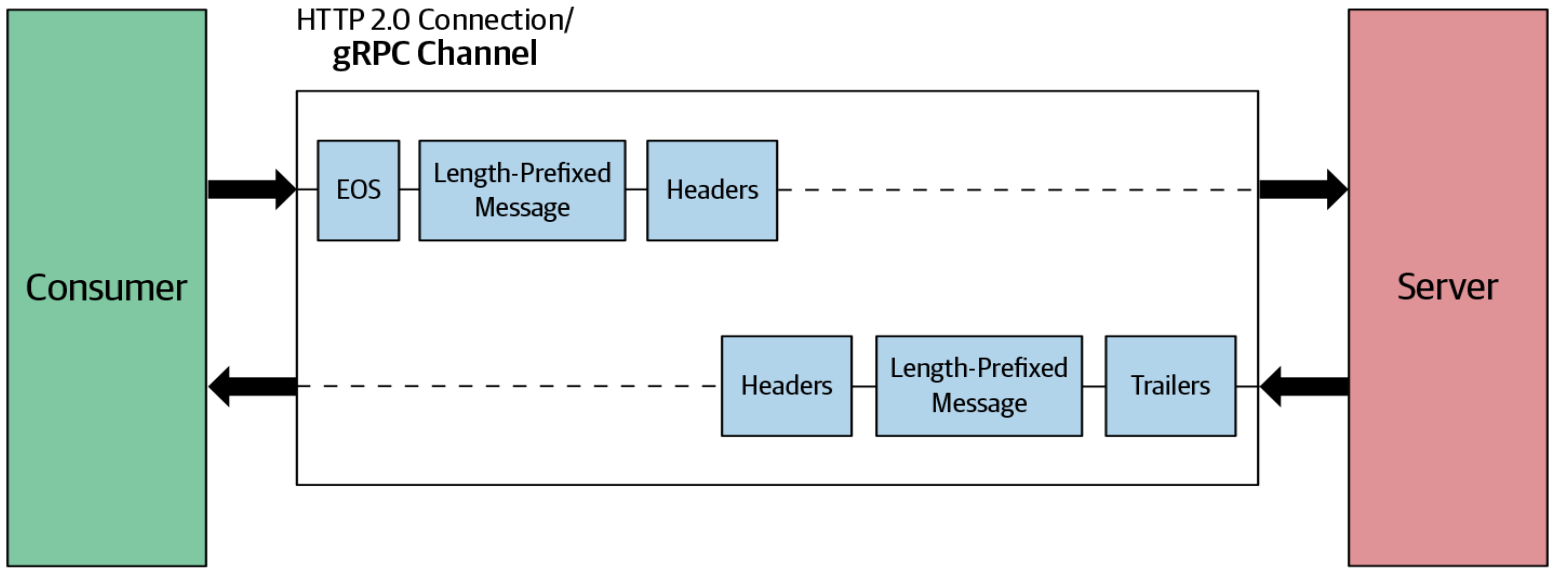


Image Source: gRPC: Up and Running by Kasun Indrasiri, Danesh Kuruppu

Server-Streaming RPC

- In server-side streaming RPC, the server sends back a sequence of responses after getting the client's request message.
- This sequence of multiple responses is known as a "stream."
- After sending all the server responses, the server marks the end of the stream by sending the server's status details as trailing metadata to the client.
- Example: searchOrder method of the Order Management service.
- rpc searchOrders(google.protobuf.StringValue) returns (stream Order);

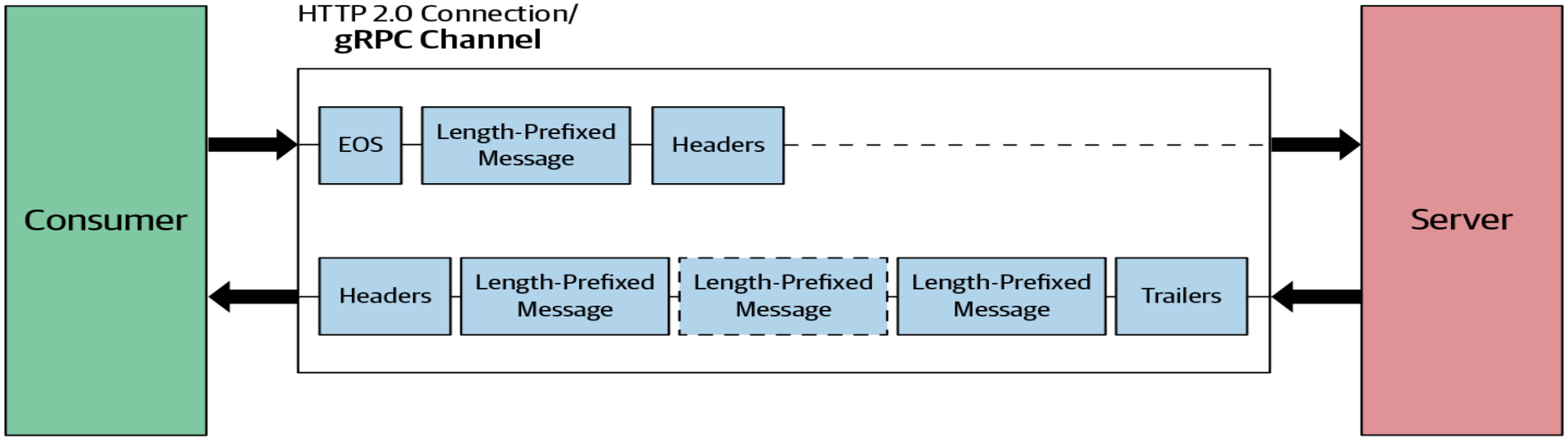


Image Source: gRPC: Up and Running by Kasun Indrasiri, Danesh Kuruppu

Client-Streaming RPC

- In client-streaming RPC, the client sends multiple messages to the server instead of a single request.
- The server sends back a single response to the client.
- However, the server does not necessarily have to wait until it receives all the messages from the client side to send a response.
- Example: updateOrders method of the Order Management service.
- **rpc** updateOrders(stream Order) **returns** (google.protobuf.StringValue);

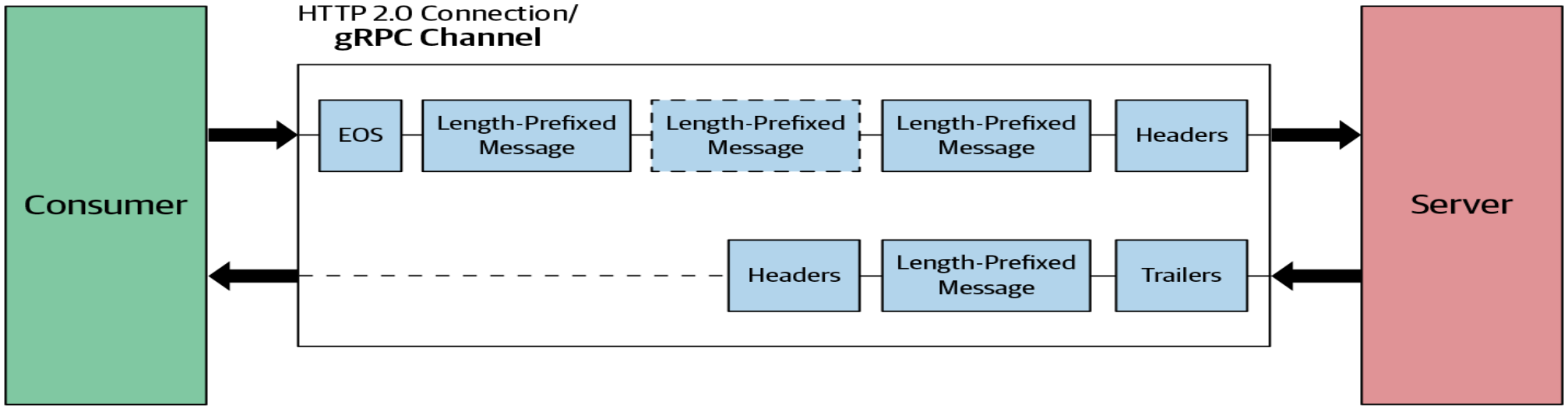


Image Source: gRPC: Up and Running by Kasun Indrasiri, Danesh Kuruppu

Bidirectional-streaming RPC

- In bidirectional-streaming RPC, the client is sending a request to the server as a stream of messages.
- The server also responds with a stream of messages.
- The call has to be initiated from the client side,
- Example: updateOrders method of the Order Management service.
- rpc processOrders(stream google.protobuf.StringValue) returns (stream CombinedShipment);

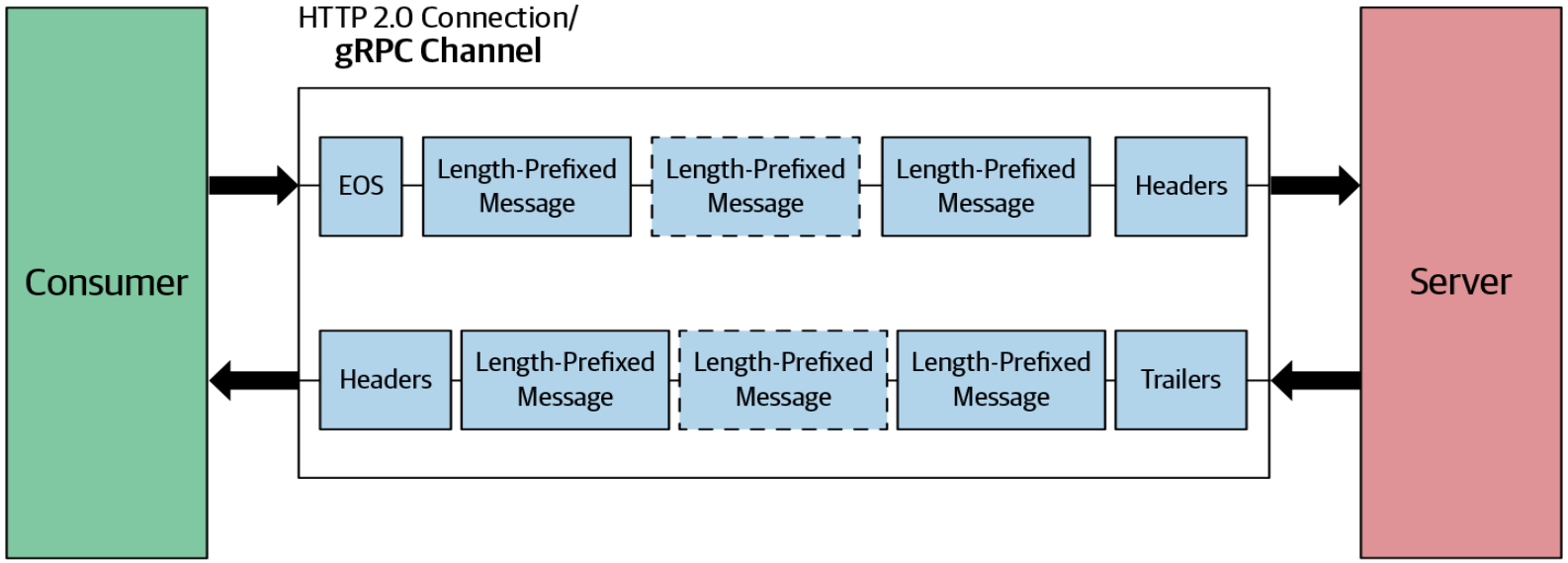


Image Source: gRPC: Up and Running by Kasun Indrasiri, Danesh Kuruppu

Summary

- gRPC
- Protobuffers
- gRPC Communication Patterns

Thank You!