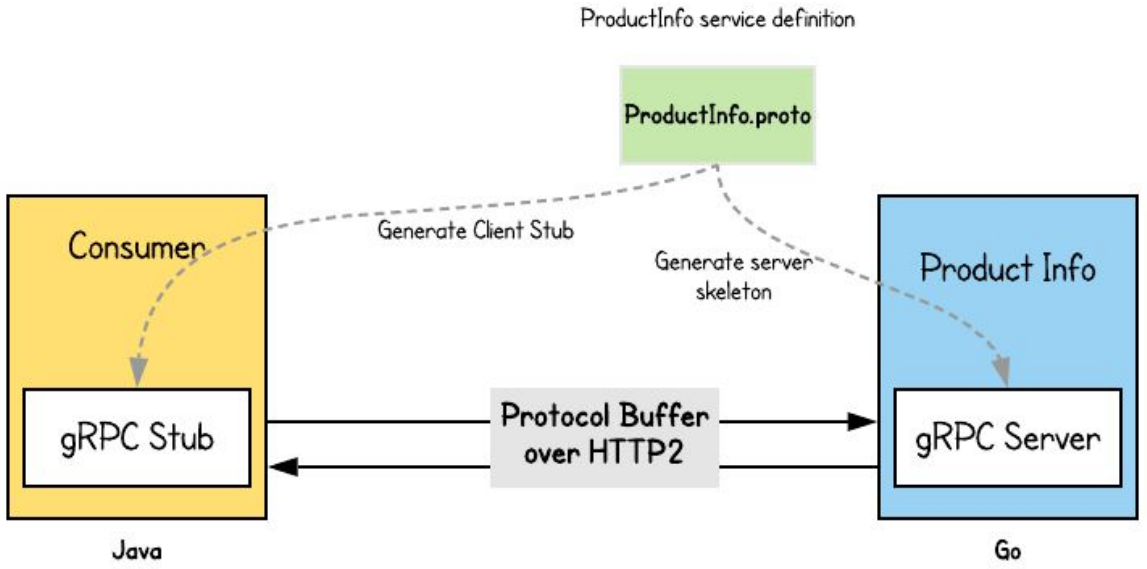
gRPC Communication Patterns – A Deep Dive

Kasun Indrasiri
Author “gRPC Up and Running”,
and “Microservices for Enterprise”

Danesh Kuruppu
Author “gRPC Up and Running”,
Associate Tech Lead @WSO2

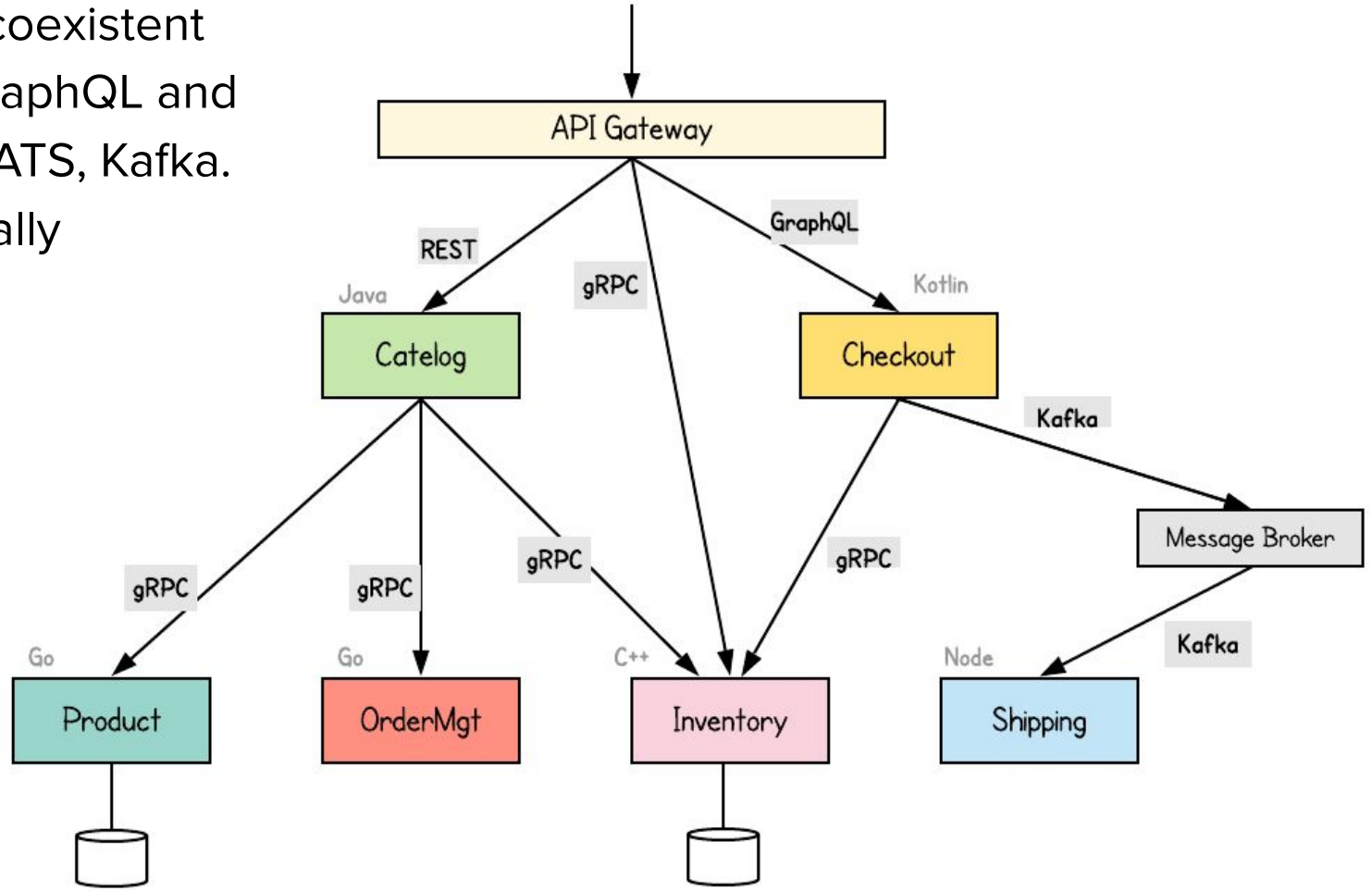
gRPC in a nutshell

- **What is gRPC?**
  - Modern inter-process communication technology.
  - Invoking remote functions as easy as making a local function invocation.
  - Contract First.
  - Using Protocol Buffers IDL
  - Binary Messaging on the wire on top of HTTP2.
- **Why gRPC?**
  - Efficient, Strongly Typed, Polyglot, Duplex Streaming.

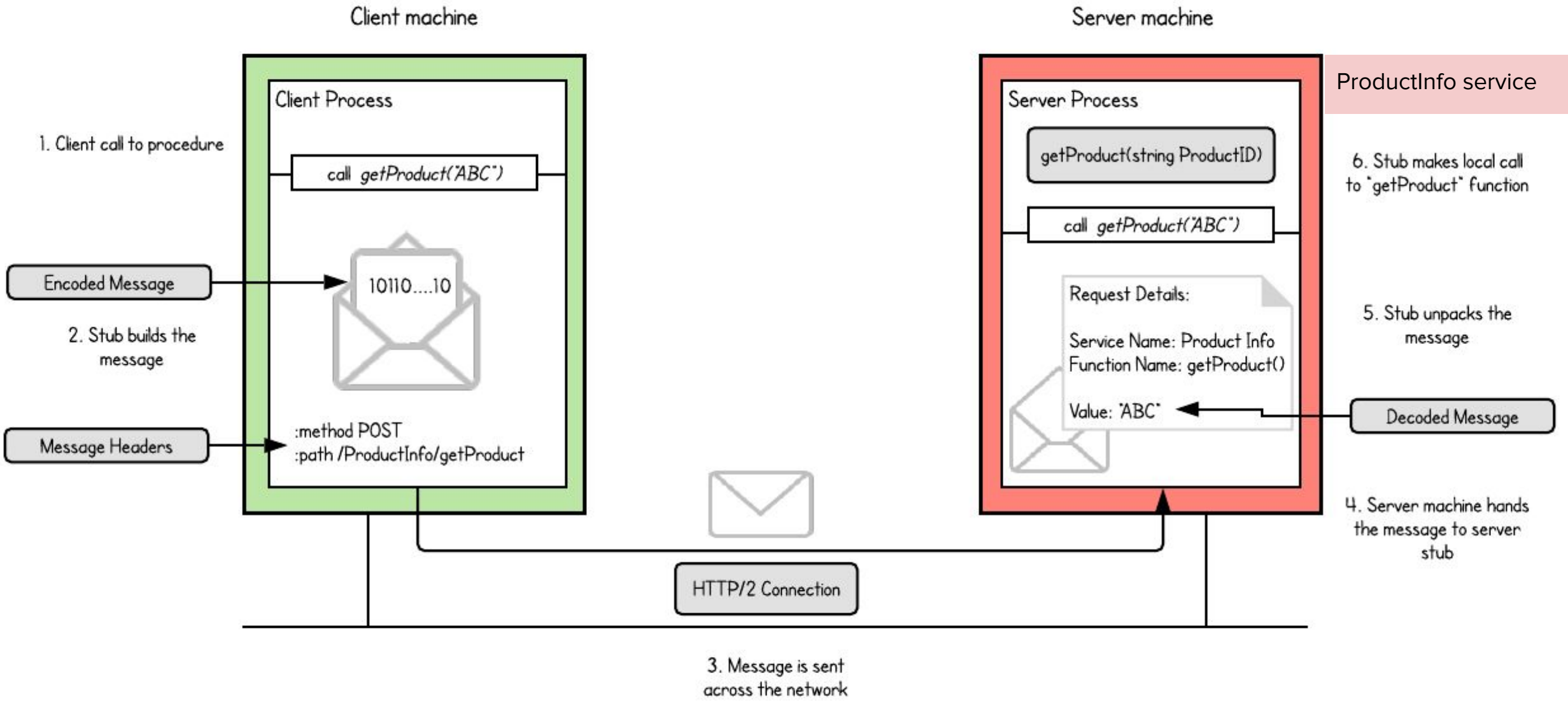


gRPC with other technologies

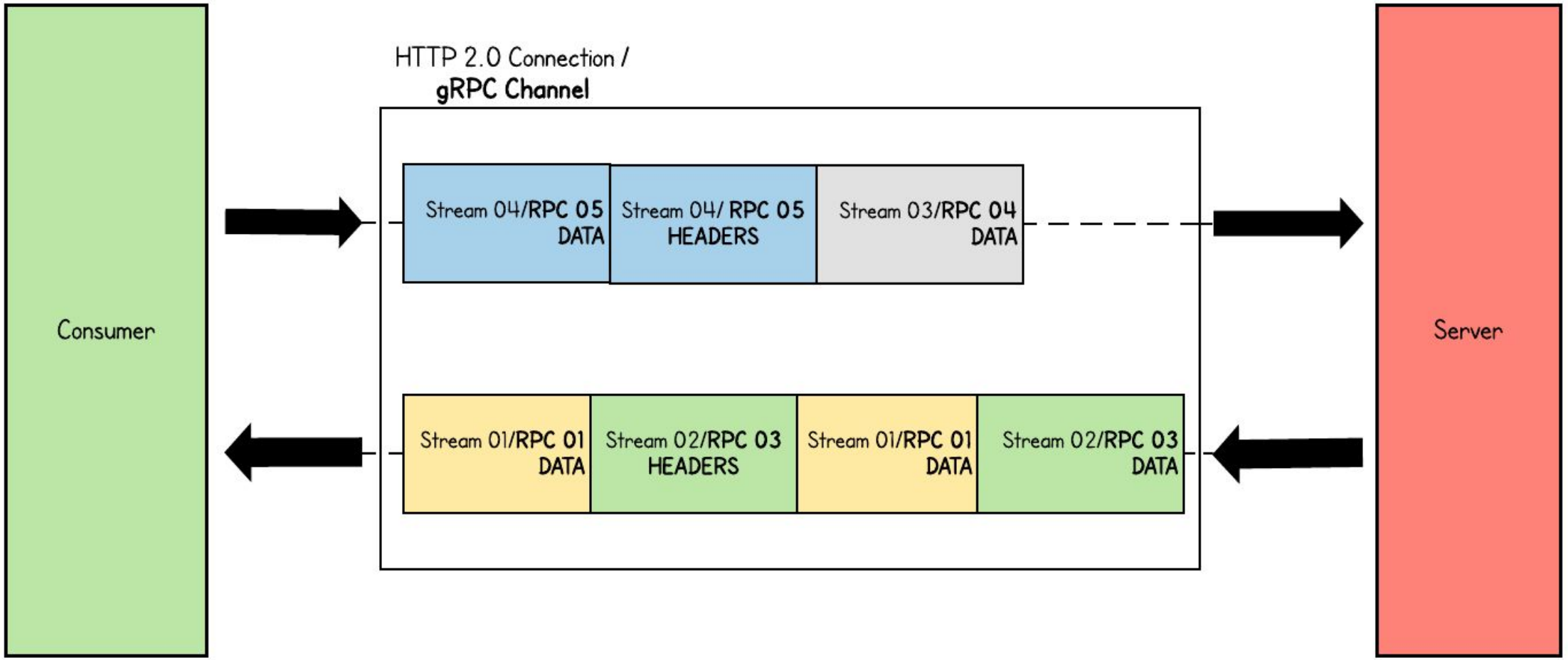
- Typical deployment, gRPC need to coexistent with transport protocol like REST, GraphQL and also with messaging protocol like NATS, Kafka.
- External client facing APIs are normally controlled by API Gateway.
- gRPC can exists in any place in the deployment.



RPC Flow

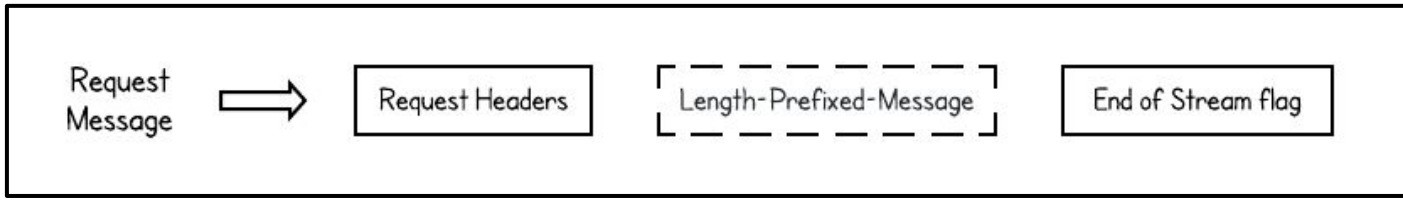

Client machine
Server machine
Client Process
Server Process
ProductInfo service
1. Client call to procedure
call getProduct("ABC")
getProduct(string ProductID)
call getProduct("ABC")
6. Stub makes local call to "getProduct" function
Encoded Message
10110....10
2. Stub builds the message
Request Details:
Service Name: Product Info
Function Name: getProduct()
Value: "ABC"
5. Stub unpacks the message
Decoded Message
Message Headers
:method POST
:path /ProductInfo/getProduct
4. Server machine hands the message to server stub
HTTP/2 Connection
3. Message is sent across the network

gRPC over HTTP/2

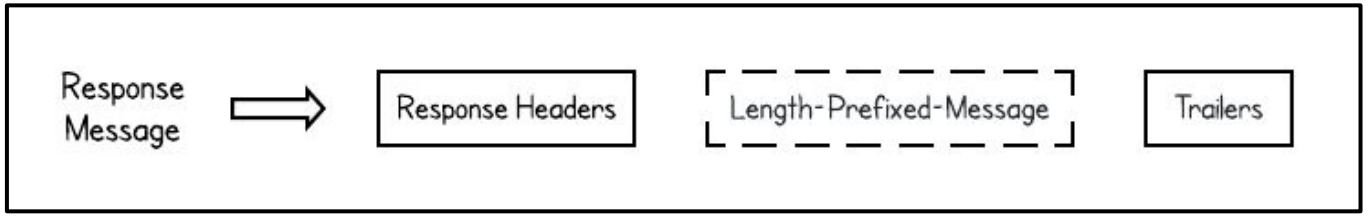


Request/Response Message

- **Request Message** contains:
  - Header frame
  - Framed message which spans one or more data frames
  - End of Stream(EOS) flag in the last data frame.

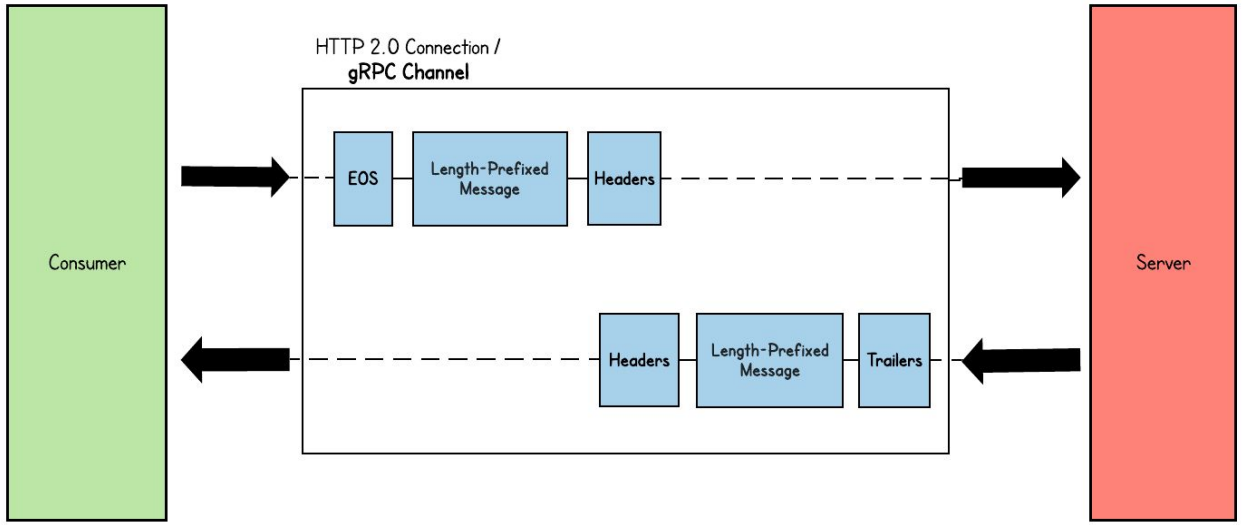


- **Response Message** contains
  - a Header frame,
  - one or more framed messages
  - Trailer headers carrying the status of the request at the end.



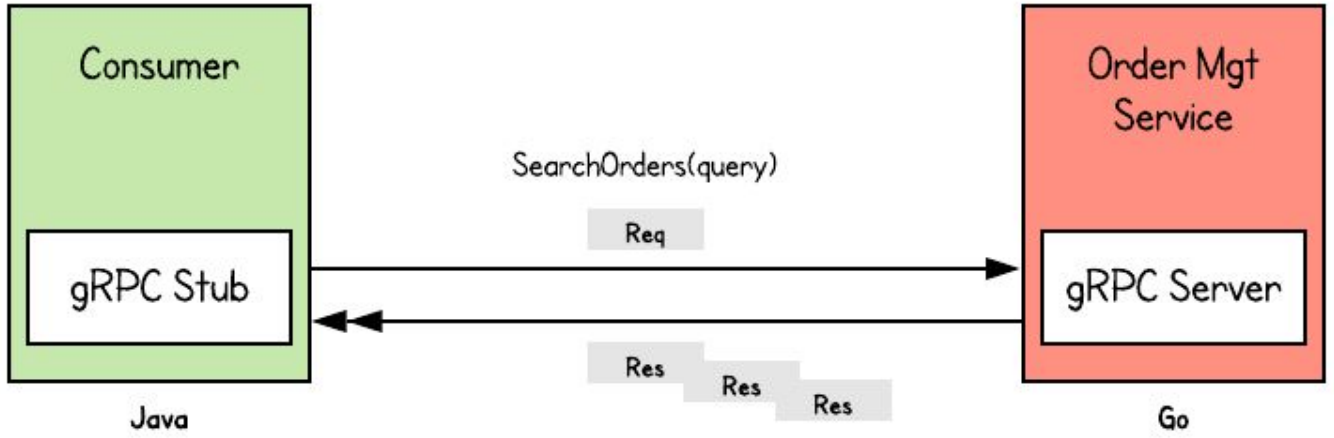
Unary/Simple RPC

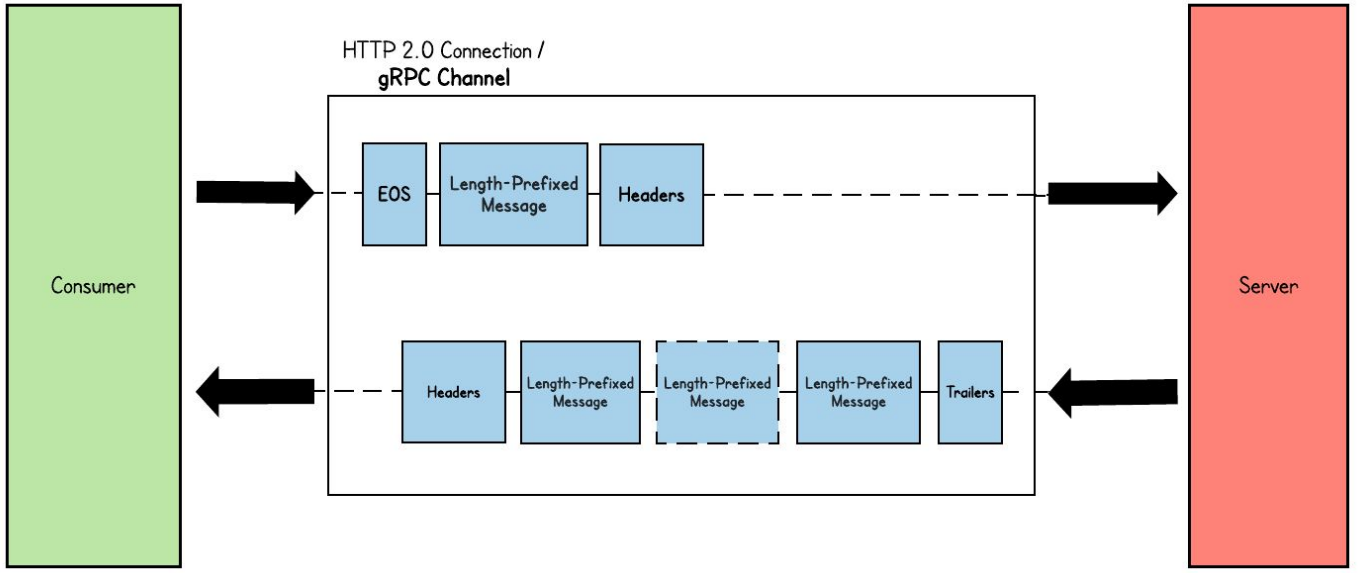
- Client sends a single request to the server and gets a single response.
- **Request Message** contains a Header frame, a framed message which spans one or more data frames and End of Stream(EOS) flag in the last data frame.
- **Response Message** contains a Header frame, a framed message and Trailer headers carrying the status of the request.



Server Streaming RPC

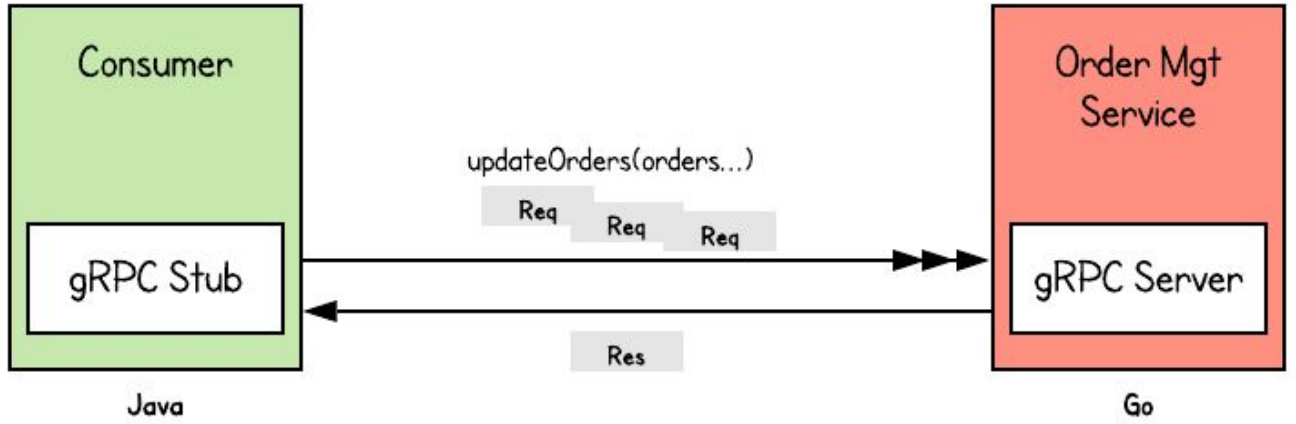
- Server sends back a sequence of responses(stream) after getting the client's request message.
- After sending all the responses server marks the end of stream.
- **Request Message** contains a Header frame, a framed message which spans one or more data frames and End of Stream(EOS) flag in the last data frame.
- **Response Message** contains a Header frame, one or more framed messages and Trailer headers carrying the status of the request.

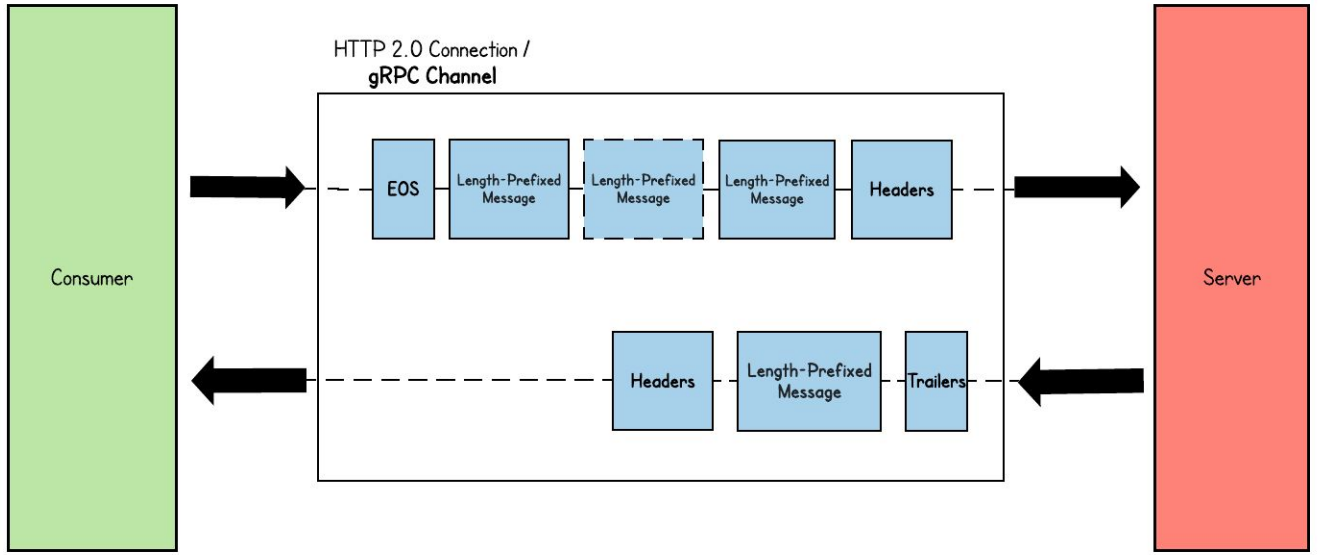




Client Streaming RPC

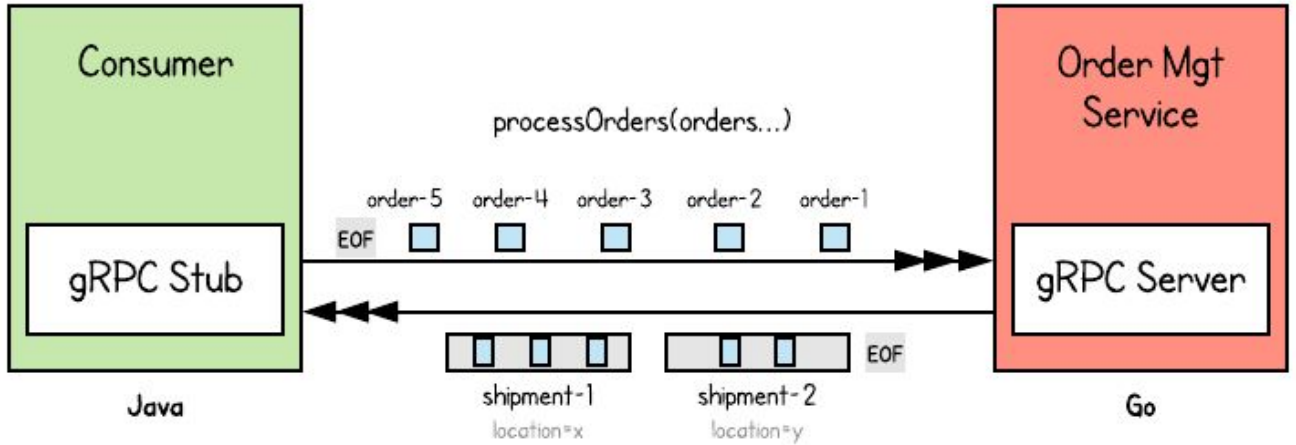
- Client sends multiple messages to the server instead of a single request.
- Server sends back a single response to the client.
- **Request Message** contains a Header frame, one or more framed messages which spans one or more data frames and End of Stream(EOS) flag in the last data frame.
- **Response Message** contains a Header frame, a framed message and Trailer headers carrying the status of the request.

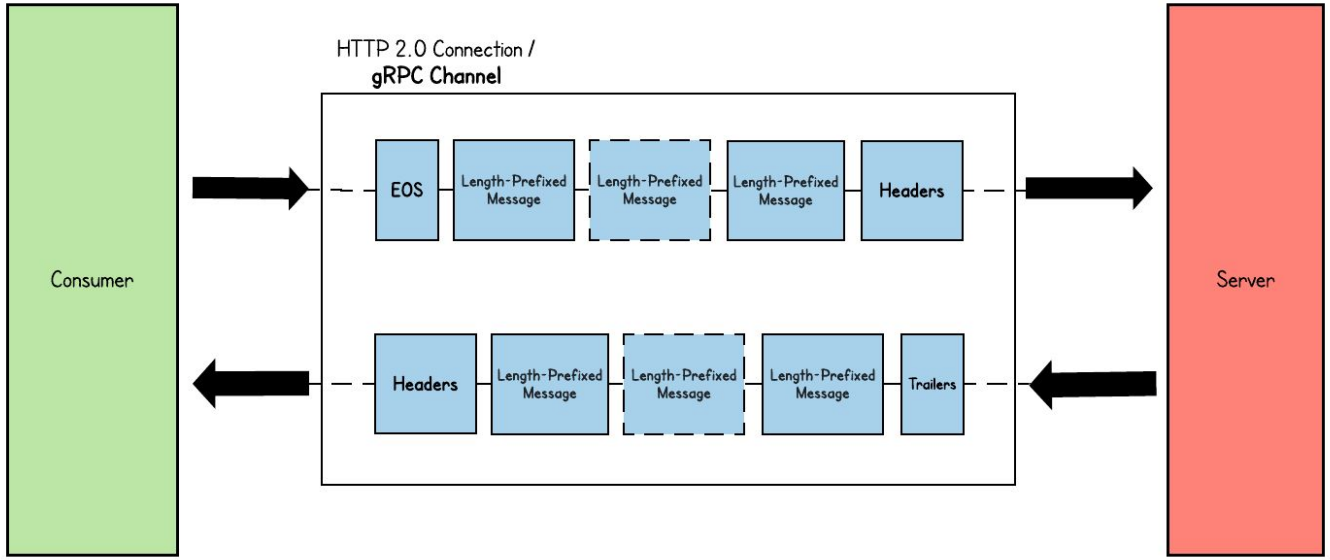




Bidirectional Streaming RPC

- Client is sending a request to the server as a stream of messages.
- Server also responds with a stream of messages.
- Client has to initiated the RPC.
- **Request Message** contains a Header frame, one or more framed messages which spans one or more data frames and End of Stream(EOS) flag in the last data frame.
- **Response Message** contains a Header frame, one or more framed messages and at the end Trailer headers carrying the status of the request.





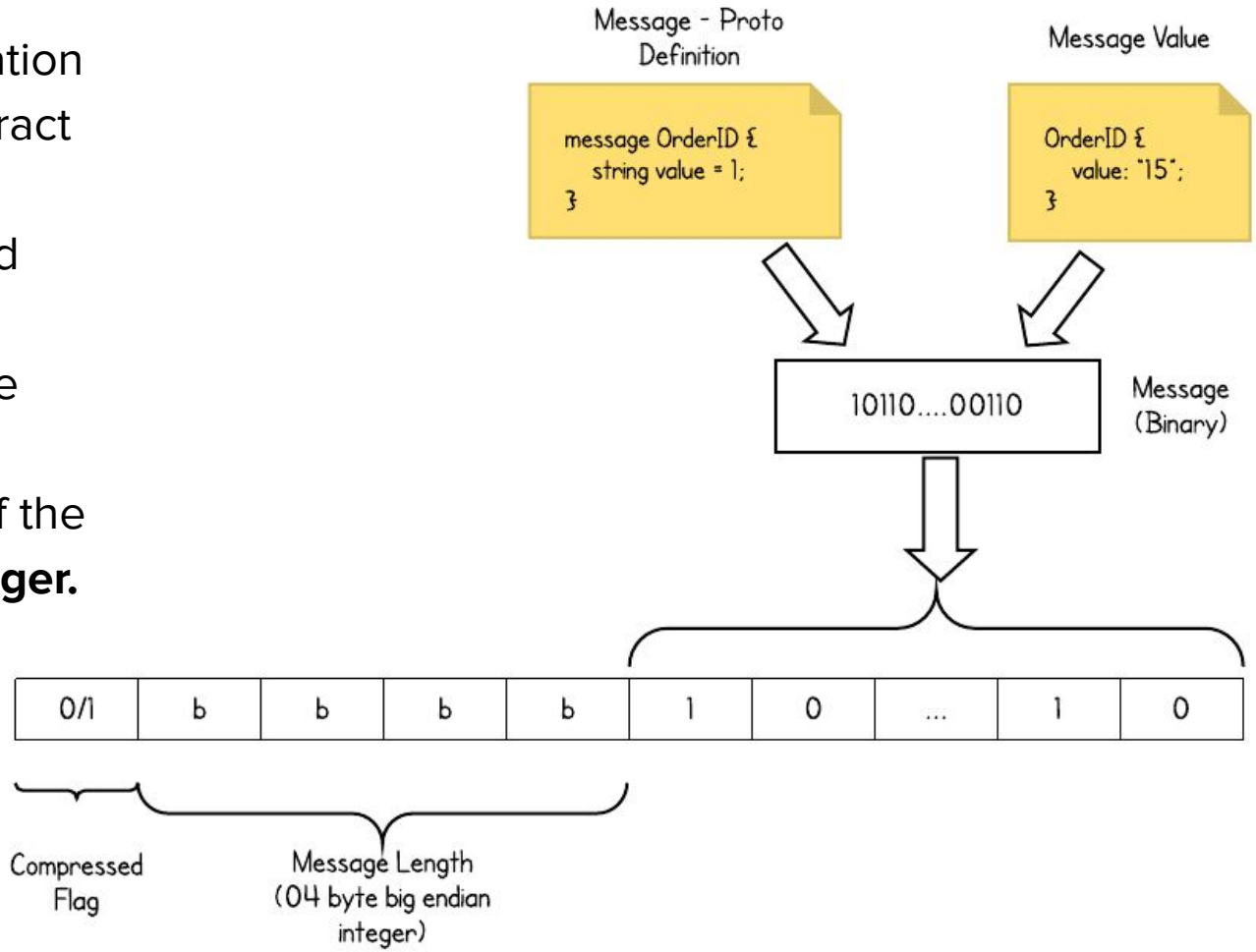
Request/Response Headers

- There are two types of headers used in gRPC
  - *Call-definition headers*
  - *Custom metadata*
- Call-definition headers are predefined headers supported by HTTP/2.
- Header names starting with `:` are called reserved headers. HTTP/2 requires these headers to appear before.
- Custom Metadata is an arbitrary set of key-value pair defined by the application layer.
- Use **Metadata** to share information about the RPC calls that are not related to the business context of the RPC (e.g. Security Headers)
- When defining custom metadata. avoid prefix `**grpc-**`. It is reserved for gRPC core.

Header key	**Header value**
:method	POST
:scheme	http
:path	/ProductInfo/getProduct
:authority	abc.com
te	trailers
grpc-timeout	1s
content-type	application/grpc
grpc-encoding	gzip
authorization(custom)	Bearer xxxxx

Length-Prefixed Message

- **Message-framing** approach constructs information such that the intended audience can easily extract the information.
- gRPC uses a message-framing technique called **length-prefix framing.**
- **Length-prefixed** approach writes the size of the message before writing the message itself.
- In gRPC, 4 bytes are allocated to set the size of the message and size is written as **Big-endian integer.**

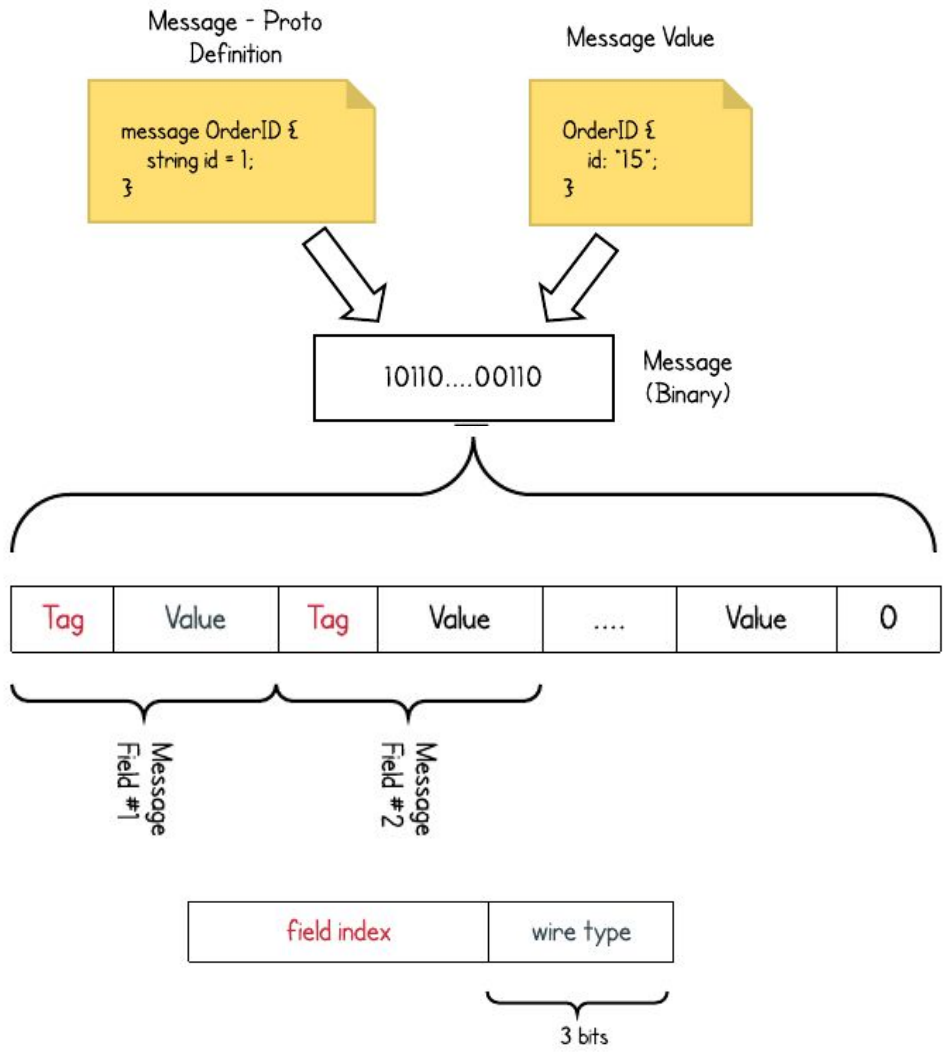


Encoded Binary Message

- By default gRPC uses **Protocol Buffers** to encode the message.
- Protocol Buffers encodes the message based on the **message structure** defined in service contract.
- Encoded Binary Message consists of **Tag-Value pairs** and Message ends with **0**
- Each Message field value is represented by tag-value in binary format.
- Tag value is constructed using **field index** defined in the contract and **wire type** based on field type.

```
Tag value = (field_index << 3) | wire_type
```

- Field value is encoded using different techniques based on field type.



Error Handling

- Errors are first class concept in gRPC.
- For every RPC call, the response will be either payload message or an error.
- The error includes **status code** which is unified across all languages and the **status message**.
- Errors are sent as response **trailing headers.**
- Do not include the error details in response payload in most cases.
- At server side, returns all errors to the caller. unless internal state is compromised.

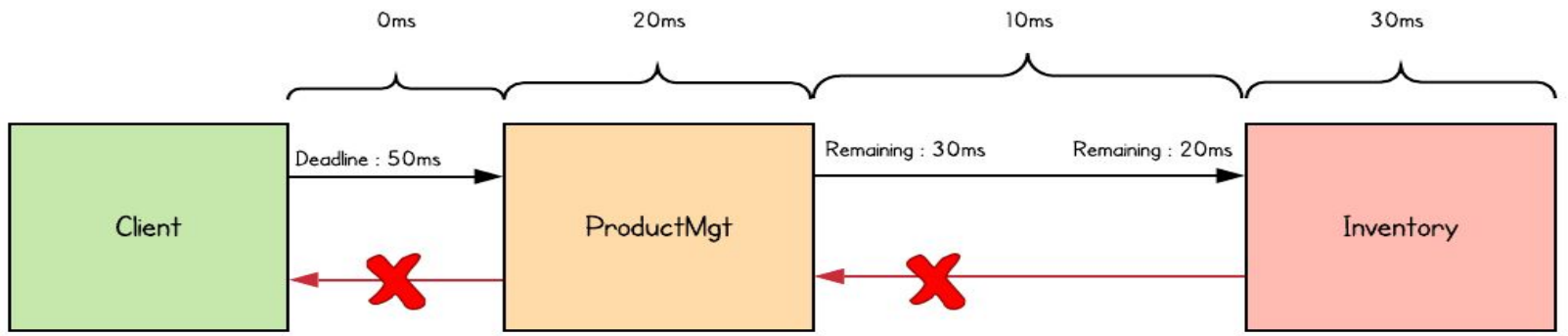
Header key	Header value
grpc-status	0 #OK
grpc-message	

Code	Number	Description
OK	0	Success status
CANCELLED	1	The operation cancelled
UNKNOWN	2	Unknown error
INVALID_ARGUMENT	3	Invalid argument
DEADLINE_EXCEEDED	4	deadline expired before the operation complete
...	..	...

Reference:
https://github.com/grpc/grpc/blob/master/doc/statuscodes.md

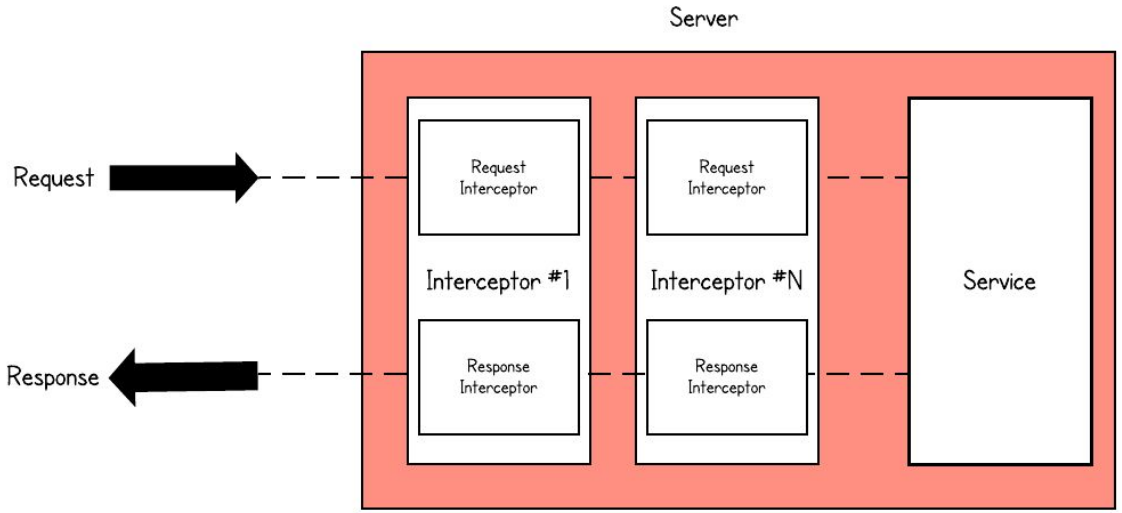
Deadlines

- Deadlines allow both clients and services to know when to abort an operation.
- Clients are responsible for setting deadlines.
- Allows use deadlines.
- Deadline normally sets as an absolute time.
- If the service is talking with another service, propagate the deadline to other services.



Interceptors

- Mechanism to execute some common logic before or after the execution of the remote function for server or client application.
- Server Side and Client Side interceptors.
- Unary Interceptors
  - Phases : preprocessing, invoking the RPC method, and postprocessing
- Streaming interceptors
  - Intercepts any streaming RPC
- Useful for logging, authentication, metrics etc.



Service Versioning with gRPC

- Services should strive to remain backwards compatible with old clients.
- Service versioning allows us to introduce breaking changes to the gRPC service.
- gRPC package ➔ specify a version number for your service and its messages.

order_mgt.proto

```
syntax = "proto3";
package ecommerce.v1;
service OrderManagement {
   rpc addOrder(Order) returns
(google.protobuf.StringValue);
   rpc getProduct(google.protobuf.StringValue) returns
(Order);
}
```

```
:method POST
:path   /<package_name>.<service_name>/<method_name>


E.g: AddOrder Remote Call:

:method POST
:path   /ecommerce.v1.OrderManagement>/addOrder
```

Extending Service Definition

- Service level, method level and field level options in service definition.
- Access those options at runtime/implementation.

Service Options

```
import "google/protobuf/descriptor.proto" ;

// custom service options
extend google.protobuf.ServiceOptions {
 string oauth2Provider = 50003;
}

service OrderManagement {
 option(oauth2Provider) =
"https://localhost:9444/oauth2/introspect";
}
```

Field Options

```
import "google/protobuf/descriptor.proto" ;

// custom field options
extend google.protobuf.FieldOptions {
 bool sensitive = 50000;
}

message Order {
 string id = 1;
 string destination = 5 [(ecommerce.sensitive) = true];
}
```

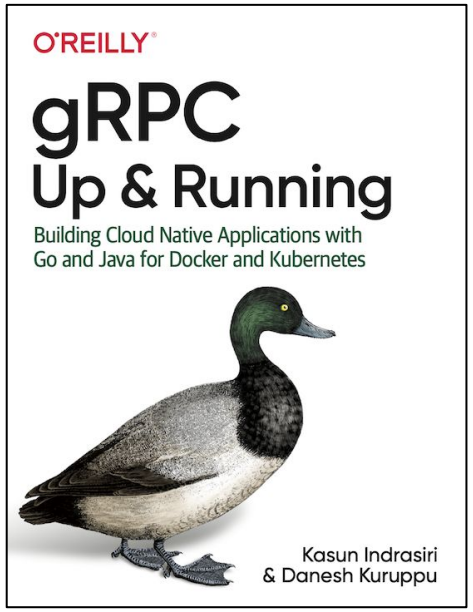
Method Options

```
import "google/protobuf/descriptor.proto" ;

// custom method options
extend google.protobuf.MethodOptions {
 int32 throttling_tier_per_min = 50001;
}
service OrderManagement {
 rpc addOrder(Order) returns (google.protobuf.StringValue) {
   option(throttling_tier_per_min) = 10000;
 }
}
```

Resources

- gRPC Up and Running Book.
  - Gives you a comprehensive understanding of gRPC.
  - gRPC communication patterns and advanced concepts.
  - Running gRPC in production.
  - Dozens of samples written in Go and Java.
- Use cases and source code in Java and Go - *https://grpc-up-and-running.github.io/*
- grpc.io




KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual