gRPC with Kotlin Coroutines

Mohit Sarveiya

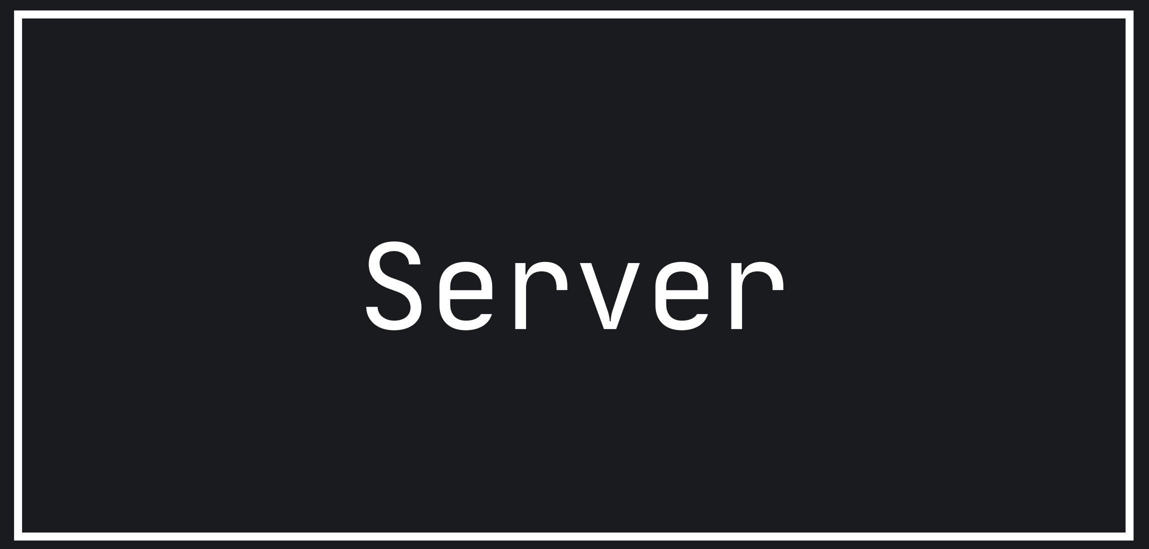
www.codingwithmohit.com

www.twitter.com/heyitsmohit

gRPC with Kotlin Coroutines

- Build gRPC Server
- Using Channels & Flow with gRPC
- Build gRPC Client in Kotlin

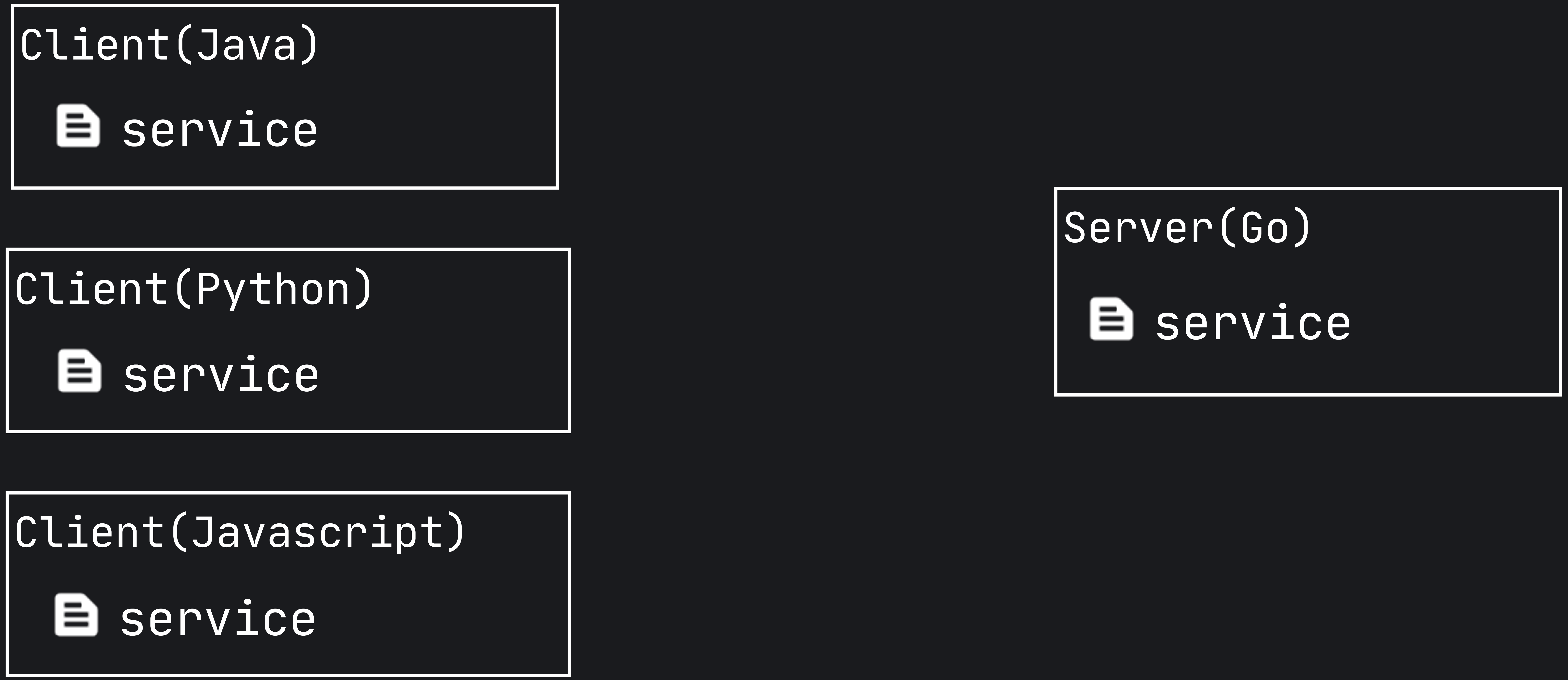
gRPC



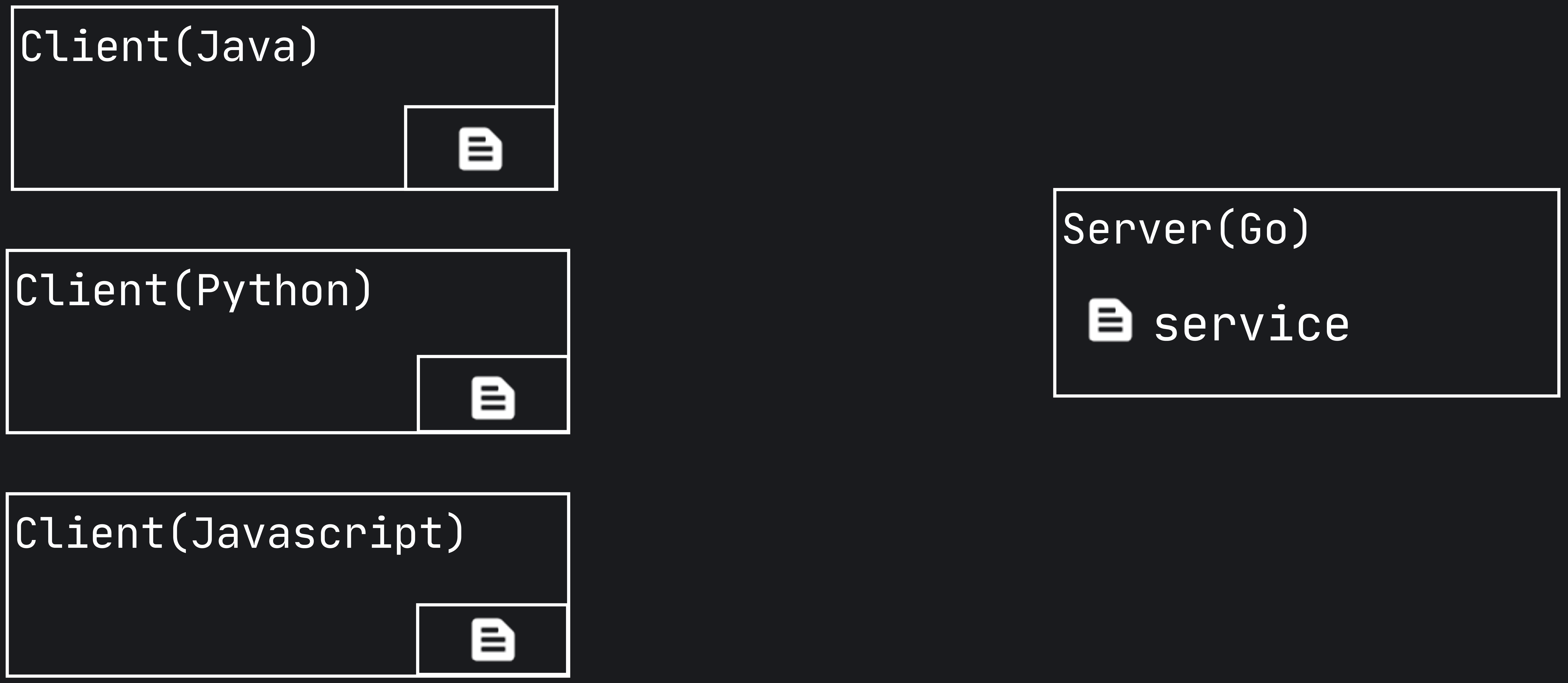
gRPC

Server
service

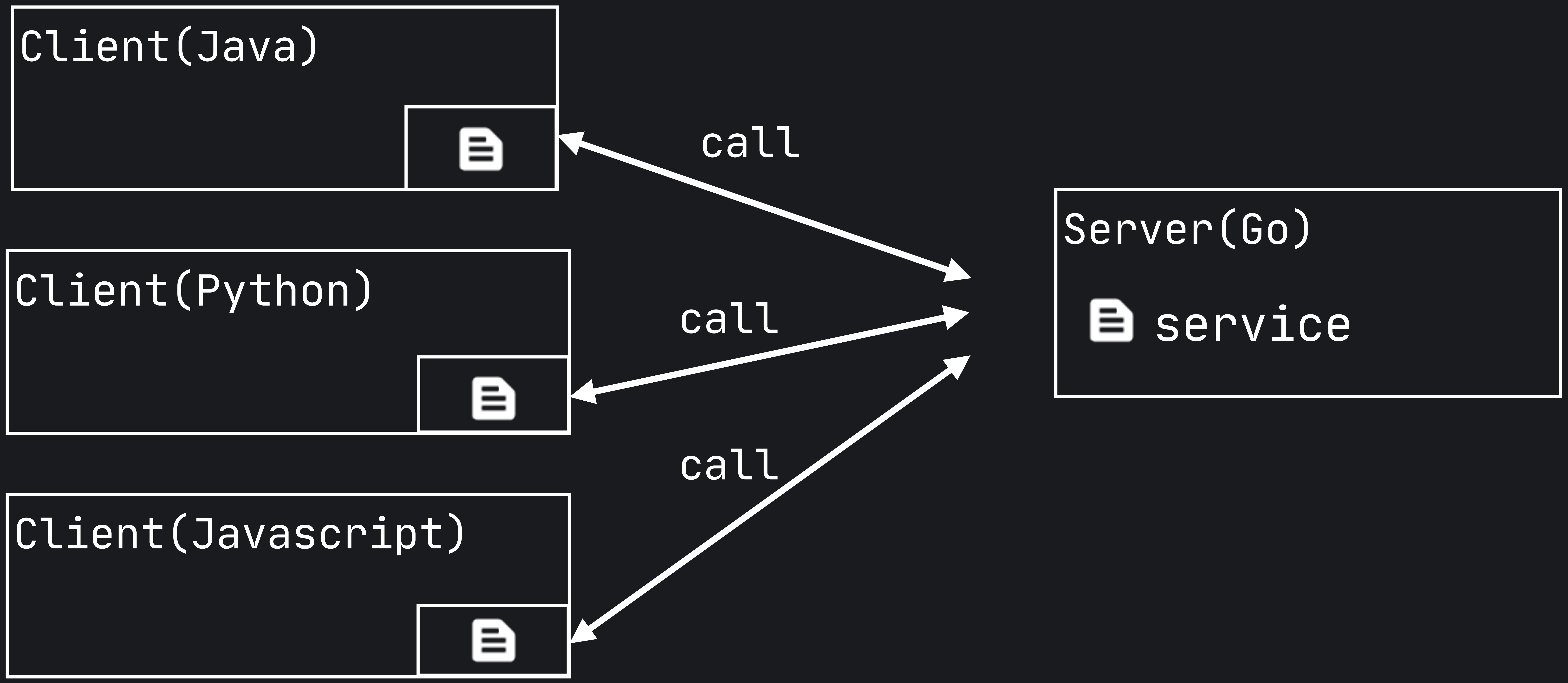
gRPC


Client(Java)
service
Client(Python)
service
Client(Javascript)
service
Server(Go)
service

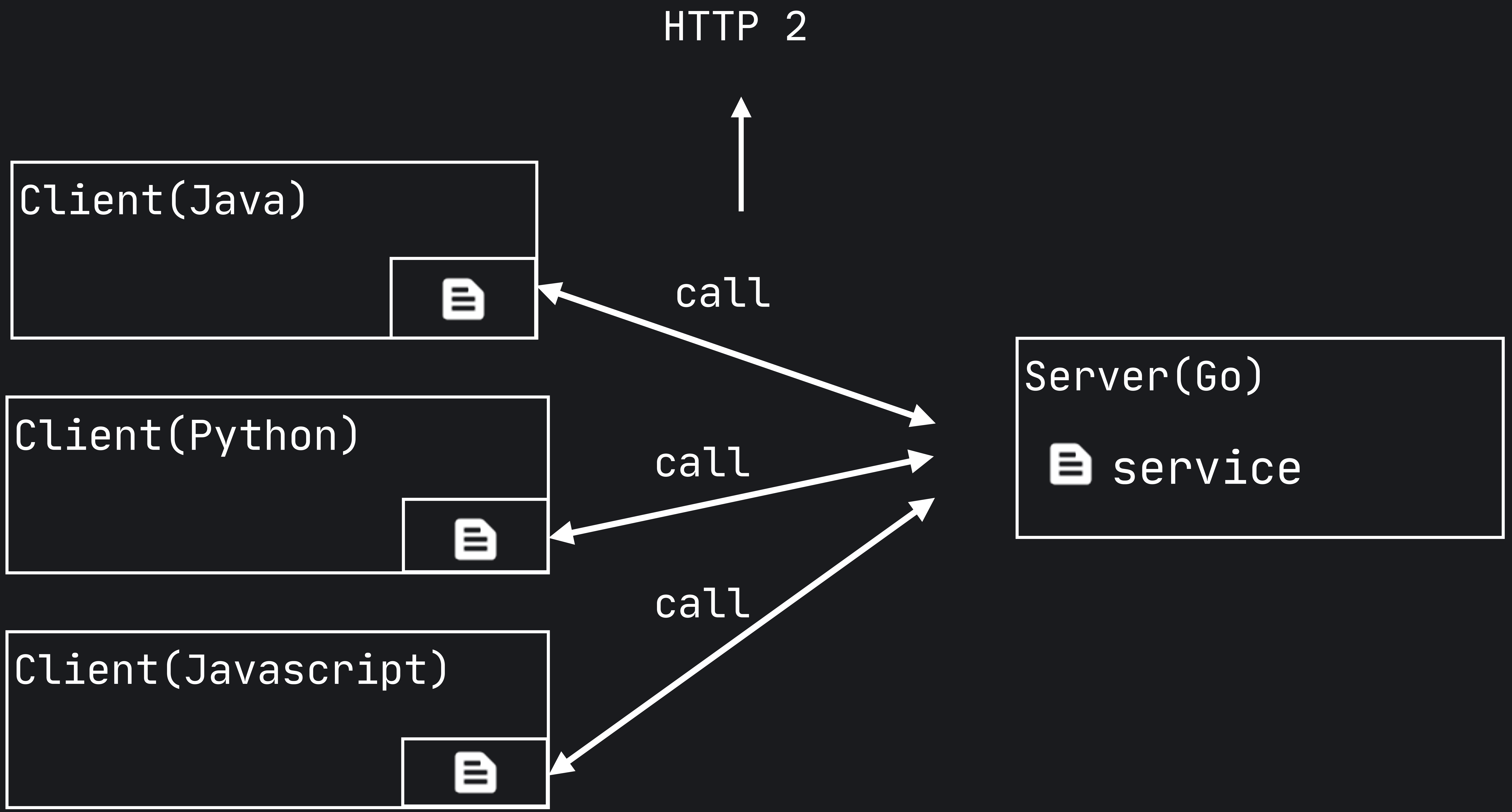
gRPC

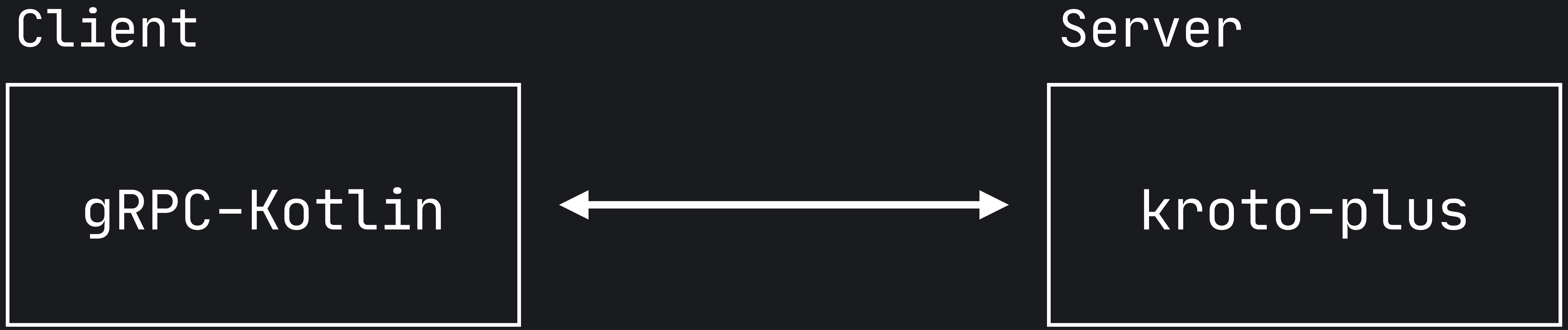

Client(Java)
Client(Python)
Client(Javascript)
Server(Go)
service

gRPC

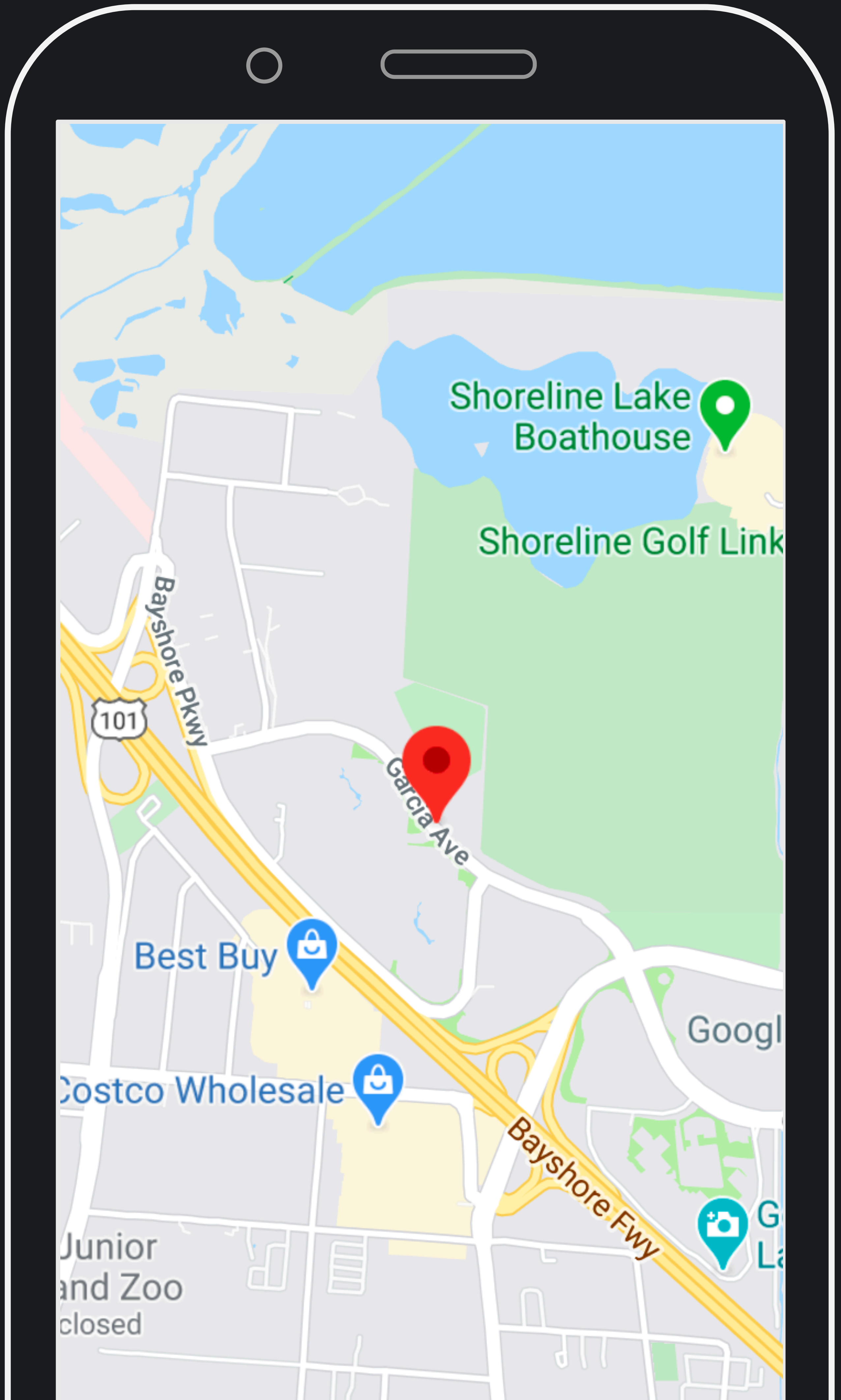

Client(Java)
Client(Python)
Client(Javascript)
call
call
call
Server(Go)
service

gRPC


HTTP 2
Client(Java)
Client(Python)
Client(Javascript)
call
call
call
Server(Go)
service

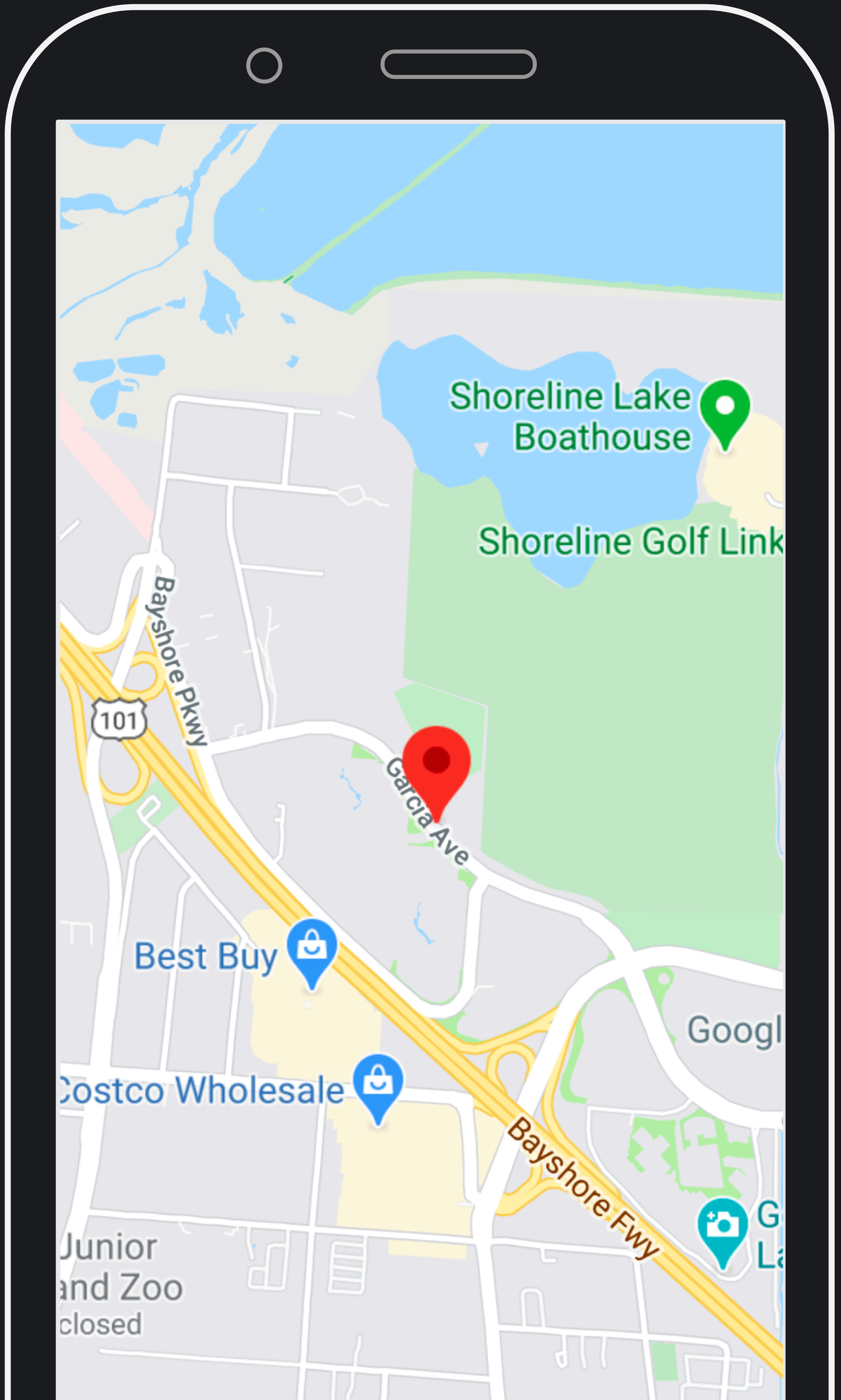

Client
gRPC-Kotlin
Server
kroto-plus

Use Case



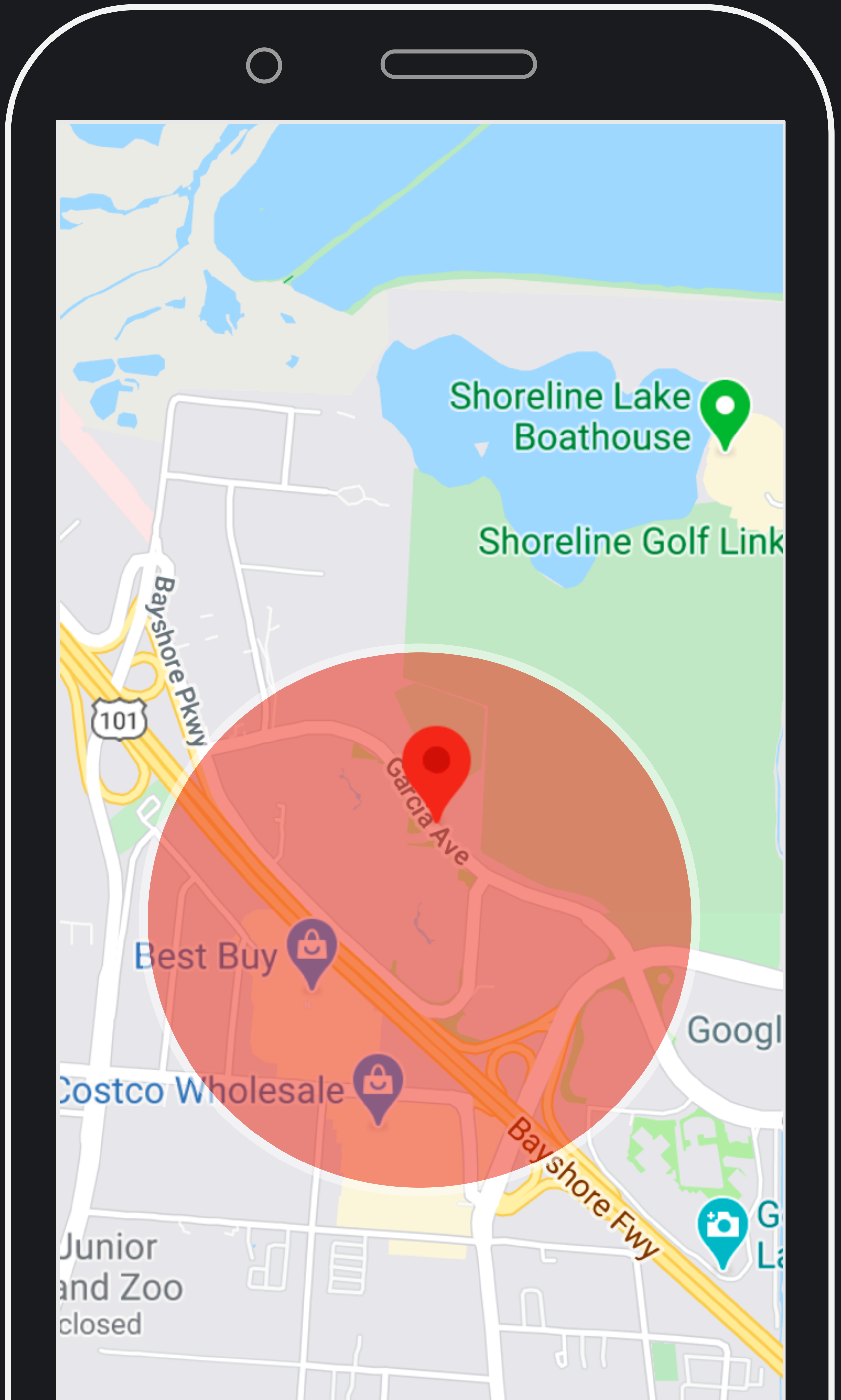
- Tracks your route.

Use Case



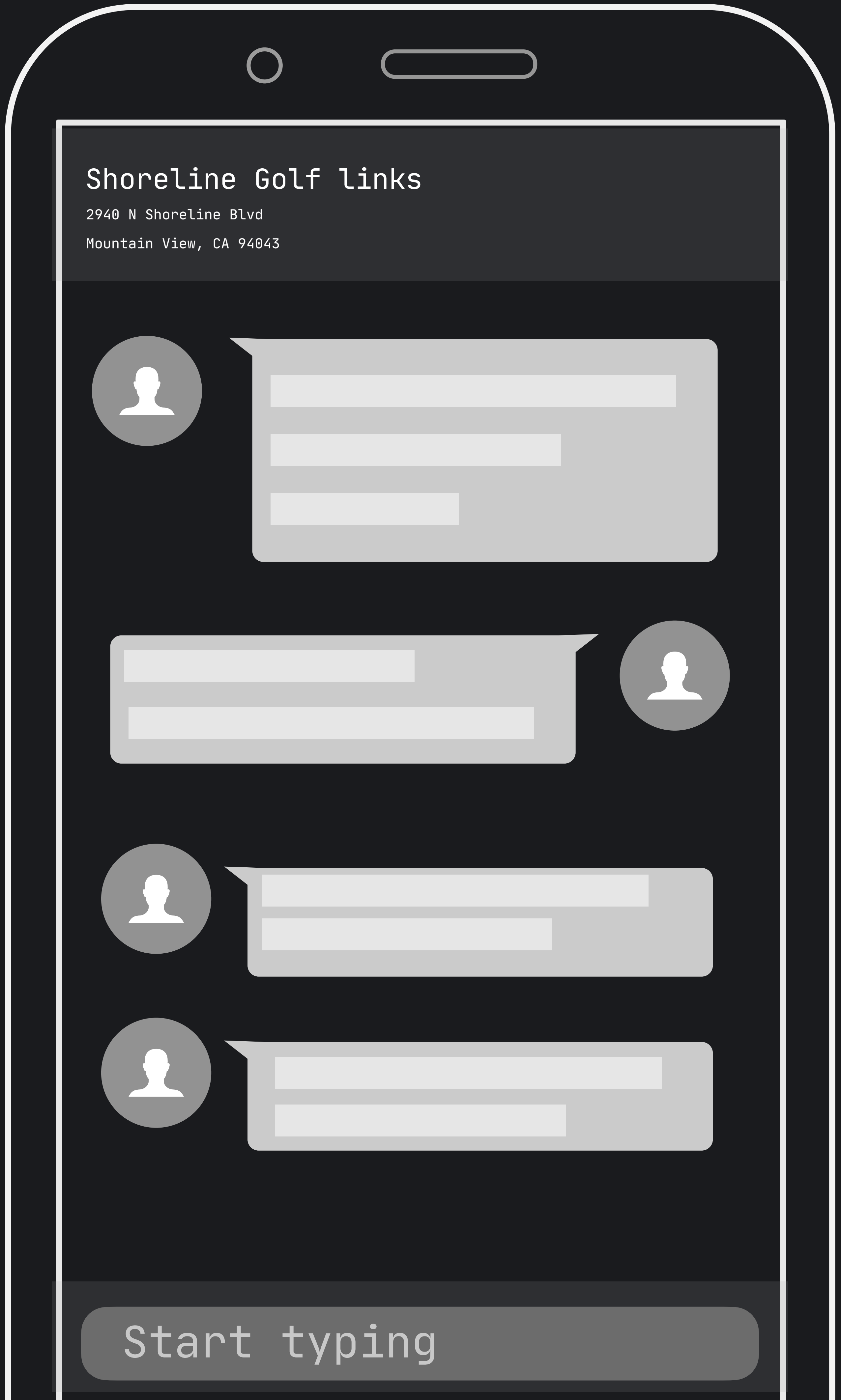
- Tracks your route.
- Get a place from location.

Use Case



- Tracks your route.
- Get a place from location.
- List Places around location.

Use Case



- Tracks your route.
- Get a place from location.
- List Places around location.
- Chat with others at location.

Building gRPC Server

Building gRPC Server

- Configure Server with Kroto-Plus
- Create Service with Protocol Buffers
- Implement Service
- Start Server

marcoferrer/kroto-plus

Kroto-Plus+

Build passing   Download 0.6.1

gRPC Kotlin Coroutines

Protobuf DSL

Scripting for Protoc

gRPC Server Modules

- 🗂 Server
  - 🗂 src
  - 🗂 database
  - 🗂 api

gRPC Server Modules

- Server
  - src ➡ Server Startup / Read from RPC
  - database
  - api

gRPC Server Modules

- Server
    - src
    - database ➡ Reading/Writing from DB
    - api

gRPC Server Modules

- Server
  - src
  - database
  - api → Define RPC Calls and Messages

gRPC Server Modules

```
Server
    src
    database
    api
      └─proto  ➡  Proto Buff Service
```

gRPC Server Modules

```
Server
    src
    database
    api
      └─proto
          └─places.proto  ➡  Create Proto Buff File
```

Service

```
syntax = “proto3”;
```

⬅ Protocol Buffer Version

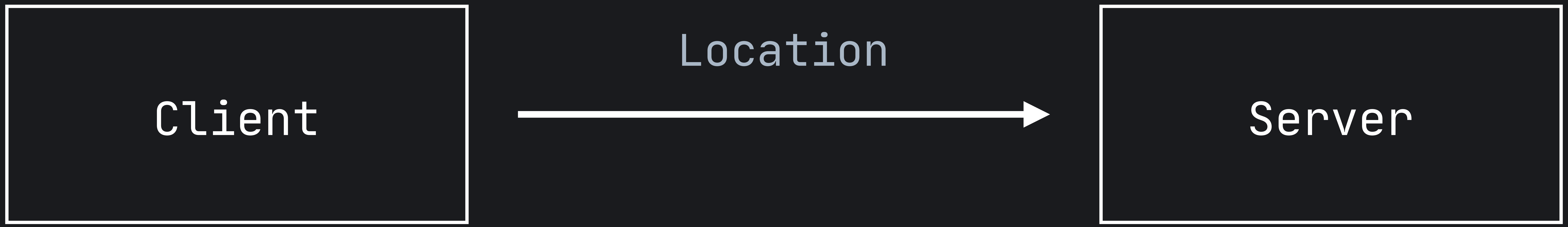
Service

```
syntax = “proto3”;

service Places {        ⇦ Declare Service







}
```

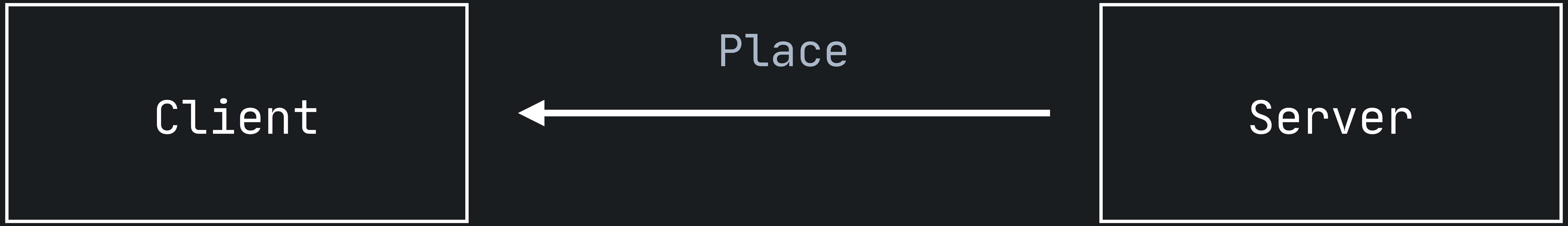
RPC Call Types

- Unary
- Server Streaming
- Client Streaming
- Bidirectional

Unary RPC Call



Unary RPC Call



Unary RPC Call

```
syntax = “proto3”;

service Places {

   rpc GetPlace(Location) returns (Place) {};

}
```

Unary RPC Call

Unary RPC Call

```
syntax = “proto3”;

service Places {

    rpc GetPlace(Location) returns (Place) {};
```

⇧

keyword

```
}
```

Unary RPC Call

```
syntax = “proto3”;

service Places {

    rpc GetPlace(Location) returns (Place) {};
```

Name of RPC call

```
}
```

Unary RPC Call

```
syntax = "proto3";

service Places {

    rpc GetPlace(Location) returns (Place) {};

}
```

Take a Location

Unary RPC Call

```
syntax = “proto3”;

service Places {

   rpc GetPlace(Location) returns (Place) {};



}
```

Returns Place

Unary RPC Call

```
syntax = “proto3”;

service Places {

   rpc GetPlace(Location) returns (Place) {};
```

↑

Unary RPC Call

```
}
```

Unary RPC Call

```
syntax = “proto3”;

service Places {

    rpc GetPlace(Location) returns (Place) {};
```

How do we define our messages?

```
}
```

Unary RPC Call

```
service Places {

  message Location {
     double latitude = 1;
     double longitude = 2;
  }




}
```

⬅ Location message

Unary RPC Call

```
service Places {

  message Location {
     double latitude = 1;
     double longitude = 2;
  }


}
```

Keyword

Unary RPC Call

Message name

```
service Places {

  message Location {
     double latitude = 1;
     double longitude = 2;
  }





}
```

Unary RPC Call

```
service Places {

  message Location {
     double latitude = 1;
     double longitude = 2;
  }
```

Fields

```
}
```

Unary RPC Call

```
service Places {

  message Location {
      double latitude = 1;
      double longitude = 2;
  }
```

Type

Unary RPC Call

```
service Places {

  message Location {
     double latitude = 1;
     double longitude = 2;
  }
```

Type ➡

.proto Type	C++	Java	Go
double	double	Double	*float64
int32	int32	int	*int32
Int64	long	int/long	*int64

https://developers.google.com/protocol-buffers/docs/overview#scalar

```
}
```

Unary RPC Call

```
service Places {

  message Location {
     double latitude = 1;
     double longitude = 2;
  }
```

Field numbers

Unary RPC Call

```
syntax = "proto3";

service Places {

   rpc GetPlace(Location) returns (Place) {};




}
```

How do we define Place message?

Unary RPC Call

```
message Place {
  string name = 1;
  Location location = 2;

}
```

⬅ Name & Location

Unary RPC Call

```
message Place {
  string name = 1;
  Location location = 2;

  PlaceType placeType = 3;  ⬅  Enum

}
```

Unary RPC Call

```
message Place {
  string name = 1;
  Location location = 2;

  PlaceType placeType = 3;

  enum PlaceType {
     Landmark = 0;
     Driving_Range = 1;
     Golf_Course = 2;
     Restaurant = 3;
     Retail = 4;
  }
```

Enum

Unary RPC Call

```
message Place {
  string name = 1;
  Location location = 2;

  PlaceType placeType = 3;

  enum PlaceType { … }

  int64 checkins = 4;
  int64 comments = 5;

}
```

Int Scaler Types

Unary RPC Call

```
syntax = “proto3”;

service Places {

   rpc GetPlace(Location) returns (Place) {};

}
```

⬆

Unary RPC Call

Unary RPC Call

```
syntax = “proto3”;

service Places {

  rpc CheckIn(Place) returns () {};

}
```

↑

Check in to place

Unary RPC Call

```
syntax = “proto3”;

service Places {

  rpc CheckIn(Place) returns () {};




}
```

How do we return an empty?

Unary RPC Call

protocolbuffers/protobuf

protobuf/src/google/protobuf/**empty.proto**

```
message Empty { }
```

Unary RPC Call

```
syntax = “proto3”;
import "google/protobuf/empty.proto";

service Places {

  rpc CheckIn(Place) returns () {};




}
```

⬅ Import

Unary RPC Call

```
syntax = “proto3”;
import "google/protobuf/empty.proto";

service Places {

  rpc CheckIn(Place) returns (google.protobuf.Empty) {};

}
```

⬆

Empty

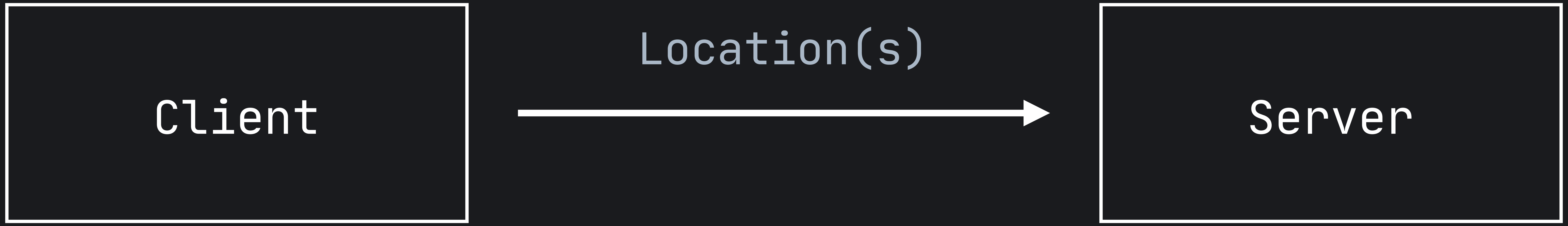
Unary RPC Call

```
syntax = “proto3”;
import "google/protobuf/empty.proto";

service Places {

  rpc CheckIn(Place) returns (google.protobuf.Empty) {};




}
```

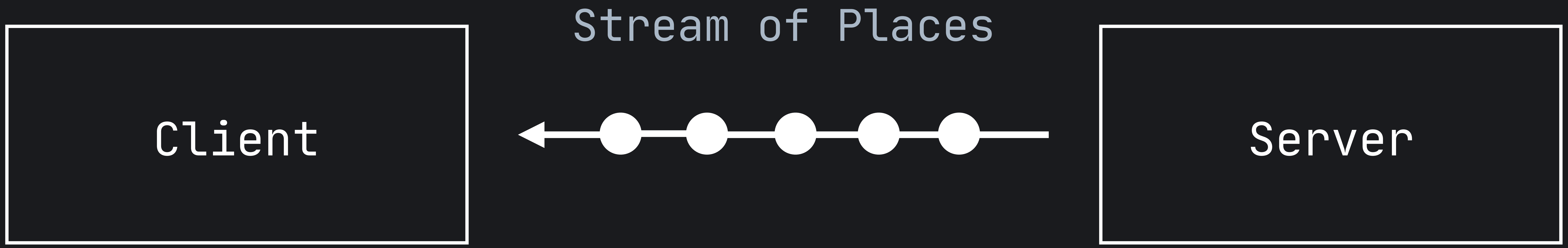
RPC Call Types

- Unary ✓
- Server Streaming ⬅
- Client Streaming
- Bidirectional

Server Streaming



Server Streaming



Server Streaming

```
service Places {

  rpc ListPlaces(Area) returns (stream Place) {};

}
```

Server Streaming RPC Call

Server Streaming

```
service Places {

  rpc ListPlaces(Area) returns (stream Place) {};

}
```

↑

Cluster of Locations

Server Streaming

```
message Area {

  Location lo = 1;

  Location hi = 2;

}
```

⬅ Cluster of Locations

Server Streaming

```
service Places {

  rpc ListPlaces(Area) returns (stream Place) {};
```

⬆

stream keyword

```
}
```

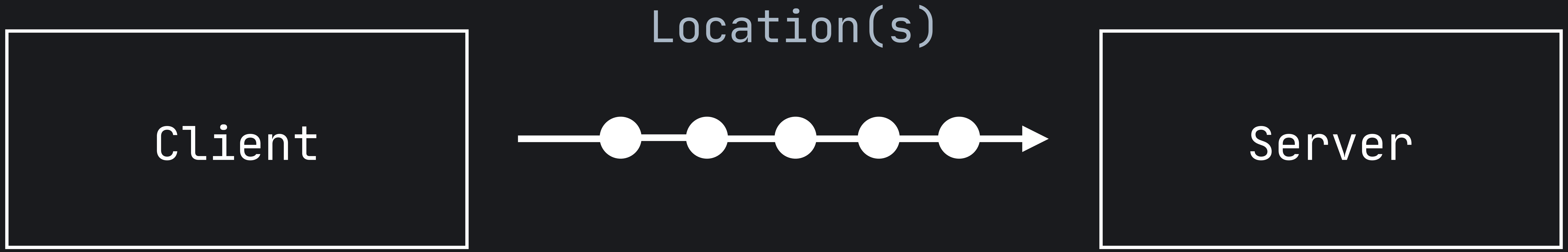
Server Streaming

```
service Places {

  rpc ListPlaces(Area) returns (stream Place) {};

                    ⬆

          Server Streaming RPC Call



}
```

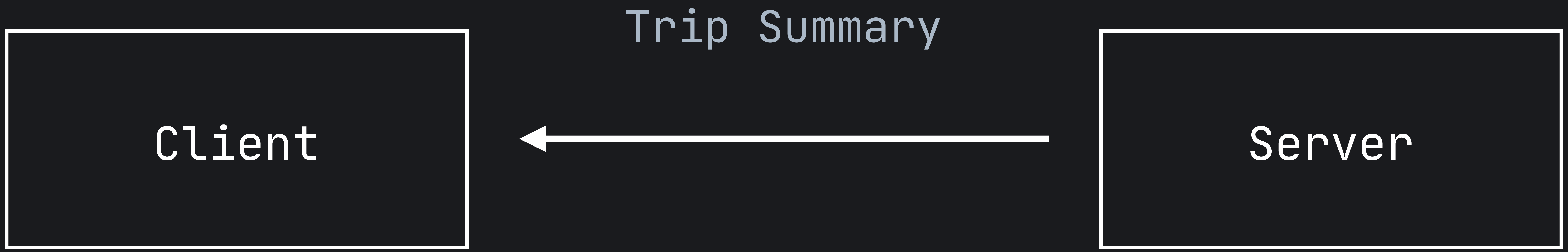
RPC Call Types

- Unary ✓
- Server Streaming ✓
- Client Streaming ⇦
- Bidirectional

Client Streaming



Client Streaming



Client Streaming

```
service Places {

  rpc RecordTrip(stream Location) returns (TripSummary) {};







}
```

Client Streaming

```
service Places {

  rpc RecordTrip(stream Location) returns (TripSummary) {};

                ⬆

           Client Stream



}
```

Client Streaming

```
service Places {

  rpc RecordTrip(stream Location) returns (TripSummary) {};

}
```

⬆

Return single message

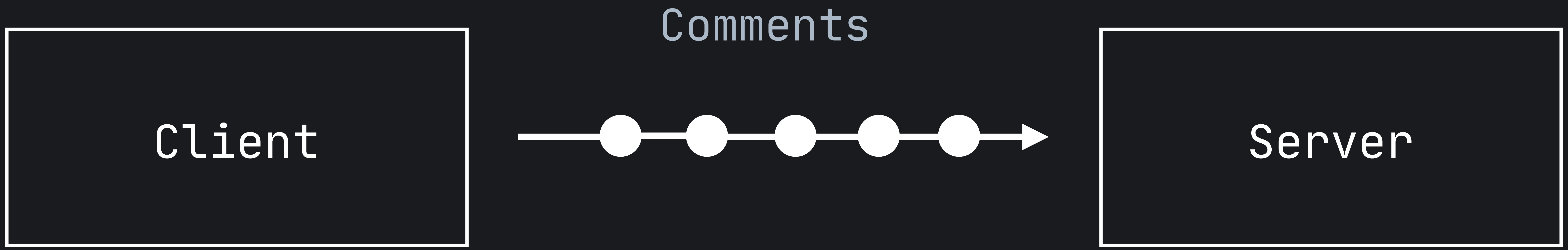
Client Streaming

```
service Places {

  rpc RecordTrip(stream Location) returns (TripSummary) {};






}
```

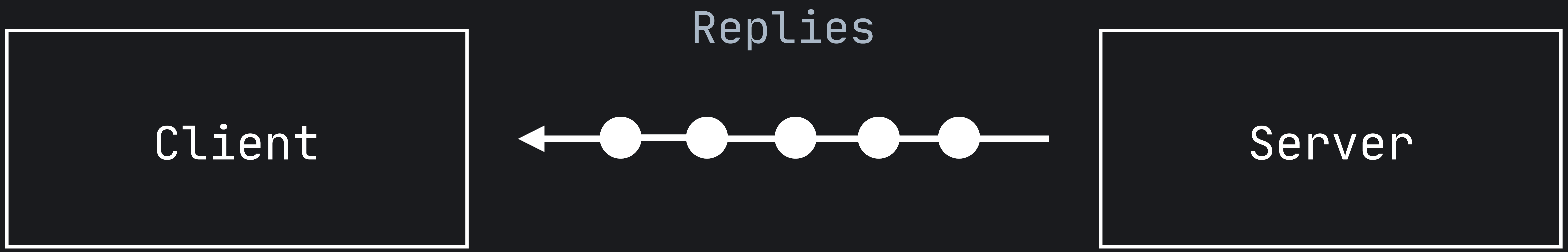
RPC Call Types

- Unary ✓
- Server Streaming ✓
- Client Streaming ✓
- Bidirectional ⬅

Bidirectional RPC Call



Bidirectional RPC Call



Bidirectional RPC Call

```
service Places {

 rpc Chat(stream Comment) returns (stream Comment) {};
```

Bidirectional RPC Call

```
}
```

Bidirectional RPC Call

```
service Places {

 rpc Chat(stream Comment) returns (stream Comment) {};
```

stream keyword

```
}
```

RPC Call Types

- Unary ✓
- Server Streaming ✓
- Client Streaming ✓
- Bidirectional ✓

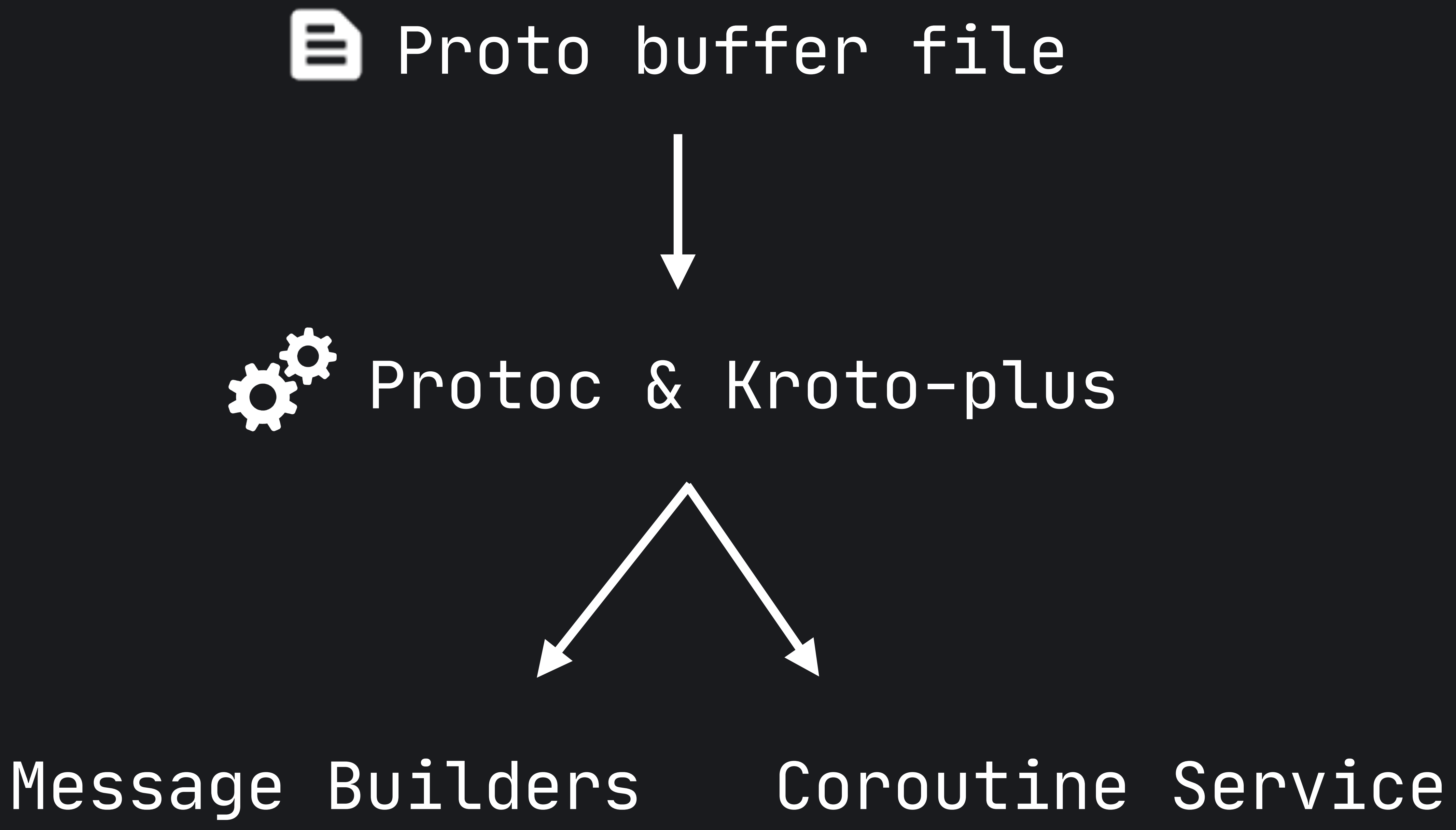
RPC Call Types

```
service Places {

  rpc GetPlace(Location) returns (Place) {};

  rpc ListPlaces(Area) returns (stream Place) {};

  rpc CheckIn(Place) returns (google.protobuf.Empty) {};

  rpc Chat(stream Comment) returns (stream Comment) {};

}
```

Building gRPC Server

- Configure Server with Kroto-Plus ✓
- Create Service with Protocol Buffers ✓
- Implement Service ⬅
- Start Server

Generate Service



Creating Messages

```
message Location {
  double latitude = 1;
  double longitude = 2;
}
```

$\longrightarrow$

```
class Location {

  static class Builder {

    Builder setLatitude(double value)

    Builder setLongitude(double value)

    Location build()
  }

}
```

Creating Messages

```
message Location {
  double latitude = 1;
  double longitude = 2;
}
```

→

```
Location.newBuilder()
        .setLatitude(40.9888341)
        .setLongitude(-73.8502007)
        .build()
```

Kotlin-friendly

api

src / **krotoPlusConfig.yml**

```
protoBuilders:
- unwrapBuilders: true
```

Create inline functions

Creating Messages

```
message Location {
  double latitude = 1;
  double longitude = 2;
}
```

$\downarrow$

```
inline fun Location(
   block: Location.Builder.() → Unit
): Location
   = Location.newBuilder()
     .apply(block)
     .build()
```

Creating Messages

```
message Location {
  double latitude = 1;
  double longitude = 2;
}
```

↓

```
Location {
    longitude = 40.9888341
    latitude = -73.8502007
}
```

Kotlin-friendly

api

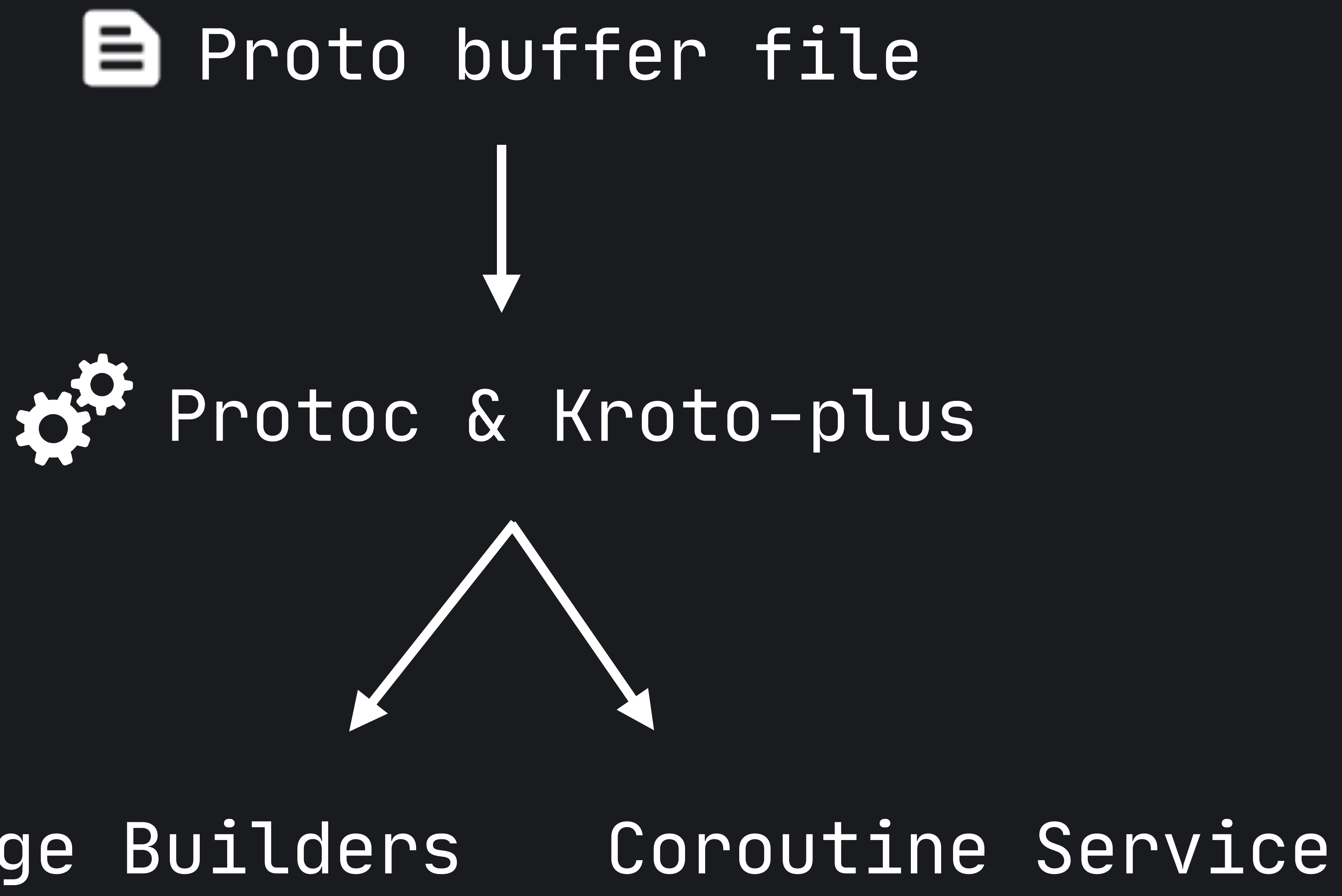
src / **krotoPlusConfig.yml**

```yaml
protoBuilders:
- unwrapBuilders: true
- useDslMarkers: true
```

Creating Messages

```kotlin
@DslMarker
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.BINARY)
annotation class PlacesProtoDslMarker

@PlacesProtoDslMarker
interface PlacesProtoDslBuilder
```

Generate Coroutines



gRPC with Coroutines

api

src / **krotoPlusConfig.yml**

```
grpcCoroutines: [{}]     ⇦ Generate Coroutines

protoBuilders:
- unwrapBuilders: true
- useDslMarkers: true
```

Generate Coroutines

```
abstract class PlacesImplBase








}
```

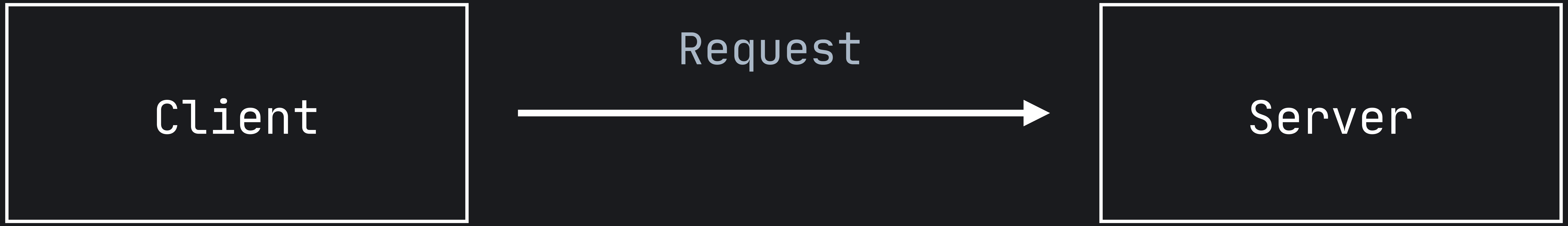
Unary RPC Call

```
abstract class PlacesImplBase

   rpc GetPlace(Location) returns (Place) {};







}
```

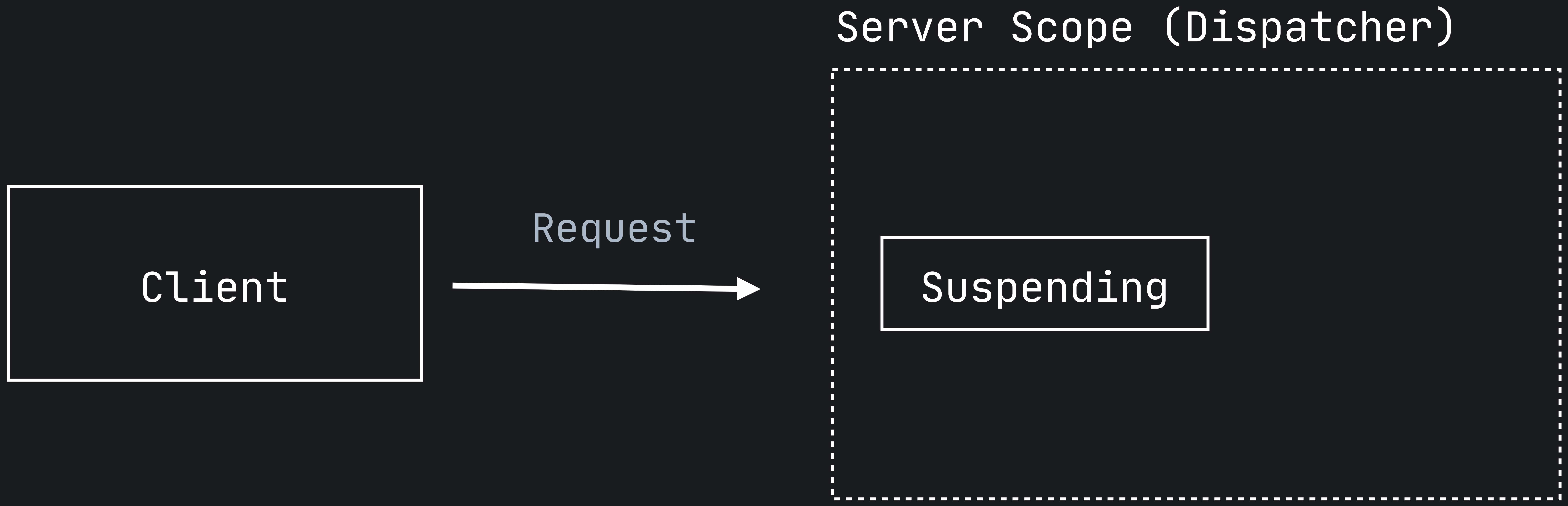
Unary RPC Call

```
abstract class PlacesImplBase

    rpc GetPlace(Location) returns (Place) {};

                        ↓

    suspend fun getPlace(request: Location): Place



}
```

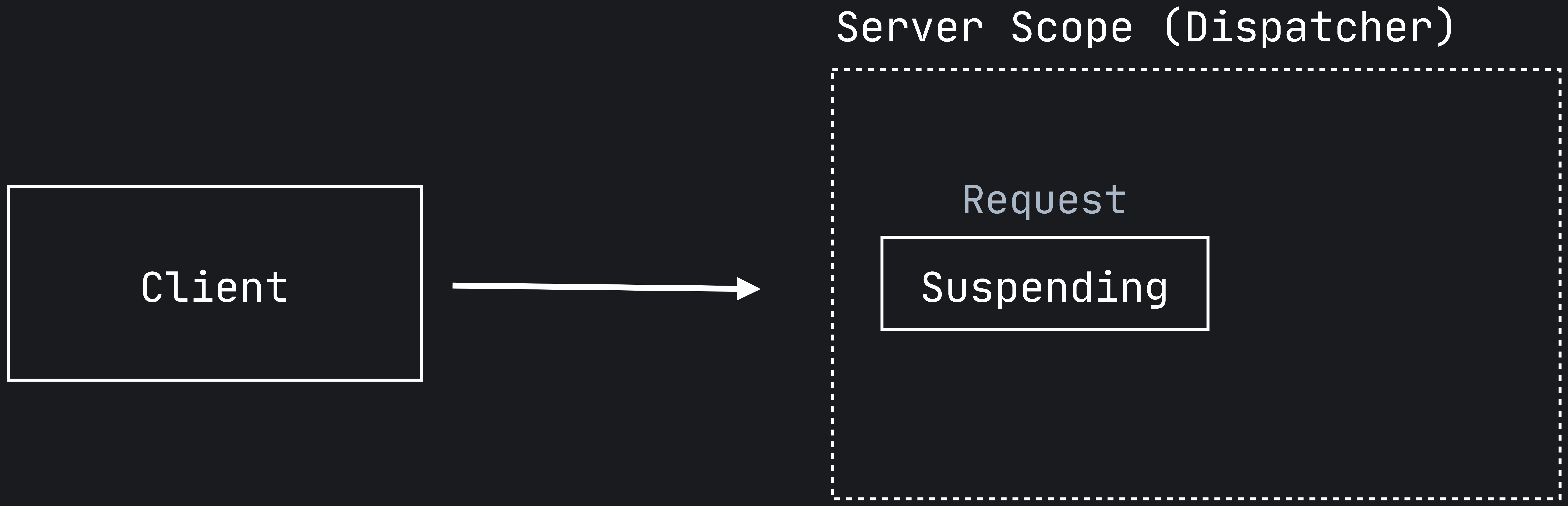
Unary RPC Call



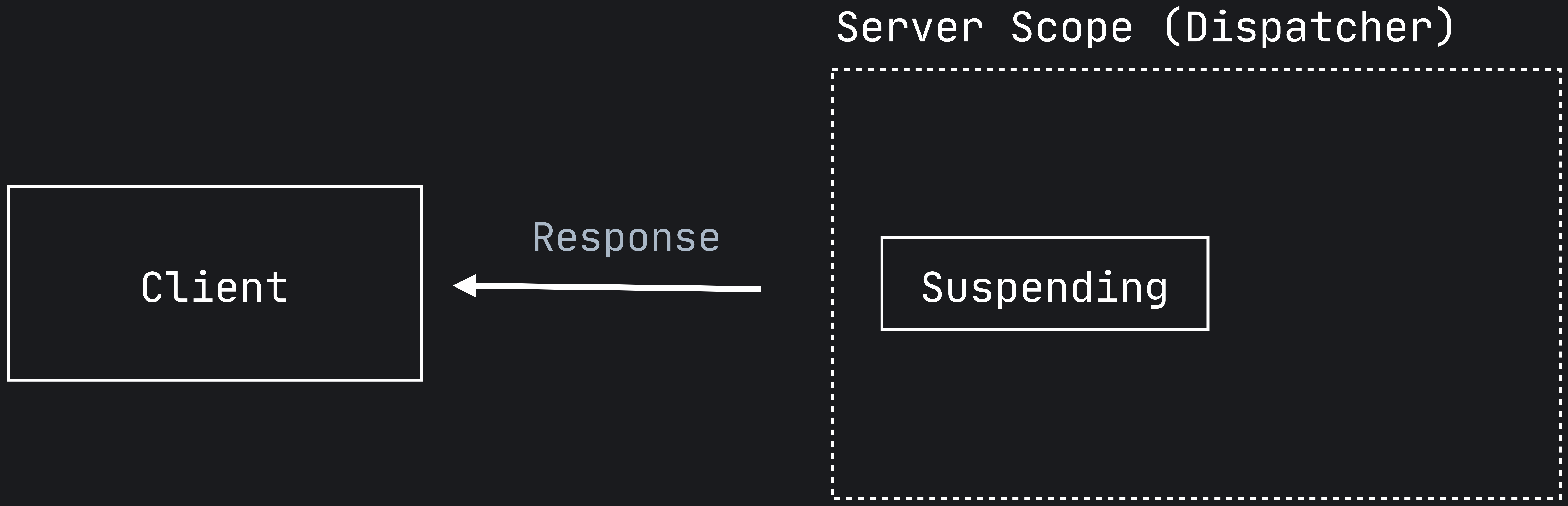
Unary RPC Call



Unary RPC Call



Unary RPC Call



Unary RPC Call

```kotlin
fun serverCallUnary(…) {
    with(newRpcScope(initialContext)) {
        launch(start = CoroutineStart.ATOMIC) {
            handleRequest()
        }
    }
}
```

Server Streaming

```
abstract class PlacesImplBase

    rpc ListPlaces(Area) returns (stream Place) {};







}
```

Server Streaming

```
abstract class PlacesImplBase

    rpc ListPlaces(Area) returns (stream Place) {};
```

$\downarrow$

```
    suspend fun listPlaces(
            request: Area,
            responseChannel: SendChannel<Place>
    )
```

Server Streaming

```
abstract class PlacesImplBase

    rpc ListPlaces(Area) returns (stream Place) {};



    suspend fun listPlaces(
            request: Area,
            responseChannel: SendChannel<Place>
    )
```

Server Streaming

```
abstract class PlacesImplBase

    rpc ListPlaces(Area) returns (stream Place) {};



    suspend fun listPlaces(
            request: Area,
            responseChannel: SendChannel<Place>
    )
```

Channel Coroutine

```kotlin
interface Channel<E> : SendChannel<E>, ReceiveChannel<E>


interface SendChannel<in E> {

   suspend fun send(element: E)
}
```

Server Streaming

```
abstract class PlacesImplBase

    rpc ListPlaces(Area) returns (stream Place) {};
```

$$\downarrow$$

```
    suspend fun listPlaces(
            request: Area,
            responseChannel: SendChannel<Place>
    )
```

Bidirectional

```
abstract class PlacesImplBase

    rpc Chat(stream Comment) returns (stream Comment) {};
```

Bidirectional

```
abstract class PlacesImplBase

   rpc Chat(stream Comment) returns (stream Comment) {};
```

↓

```
  suspend fun chat(
     requestChannel: ReceiveChannel<Comment>,
     responseChannel: SendChannel<Comment>
  )
```

Bidirectional

```
abstract class PlacesImplBase

   rpc Chat(stream Comment) returns (stream Comment) {};




  suspend fun chat(
     requestChannel: ReceiveChannel<Comment>,
     responseChannel: SendChannel<Comment>
  )
```

Channel Coroutine

```kotlin
interface Channel<E> : SendChannel<E>, ReceiveChannel<E>


interface ReceiveChannel<out E> {

    suspend fun receive(): E
}
```

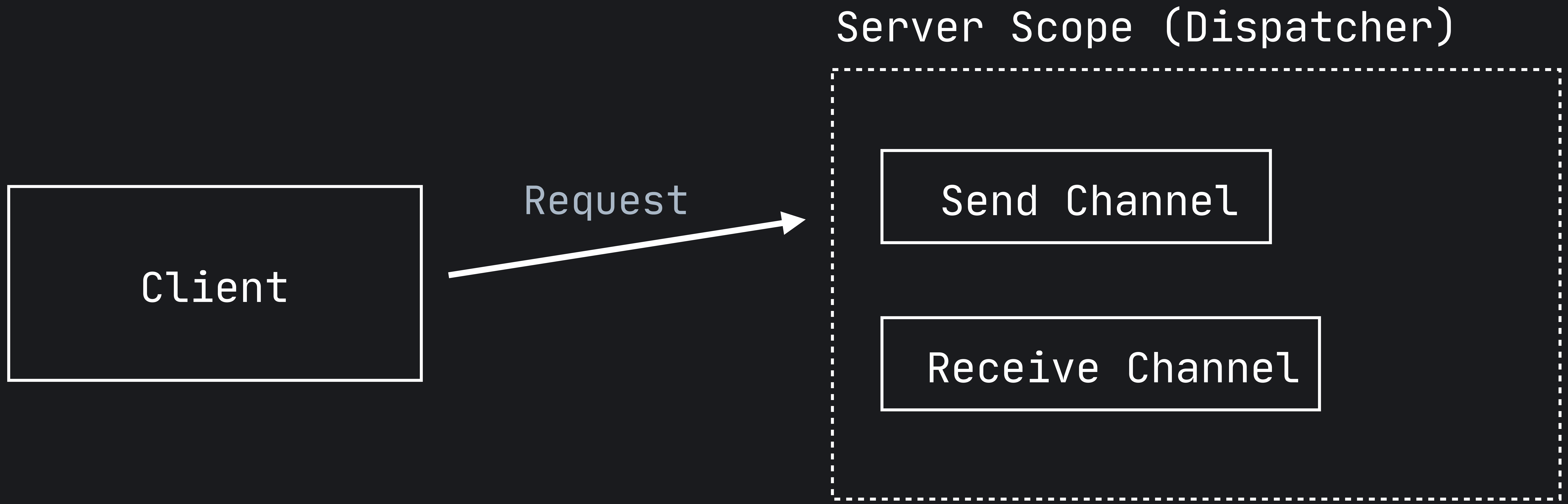
Bidirectional

```
abstract class PlacesImplBase

   rpc Chat(stream Comment) returns (stream Comment) {};
```

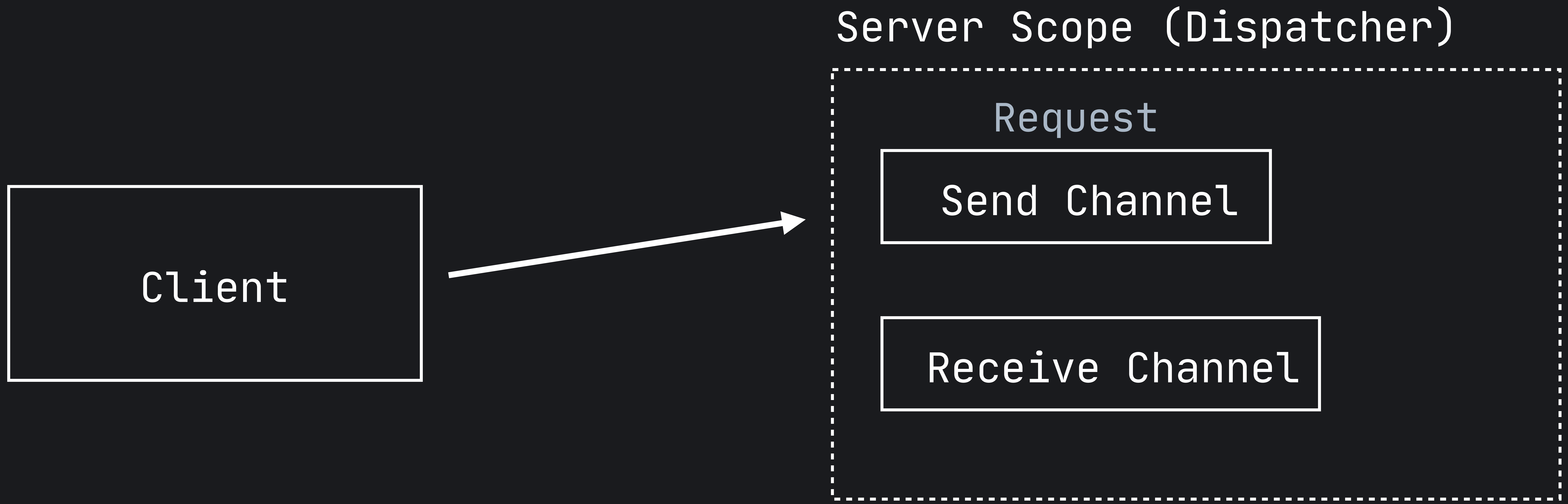
$$\downarrow$$

```
  suspend fun chat(
     requestChannel: ReceiveChannel<Comment>,
     responseChannel: SendChannel<Comment>
  )
```

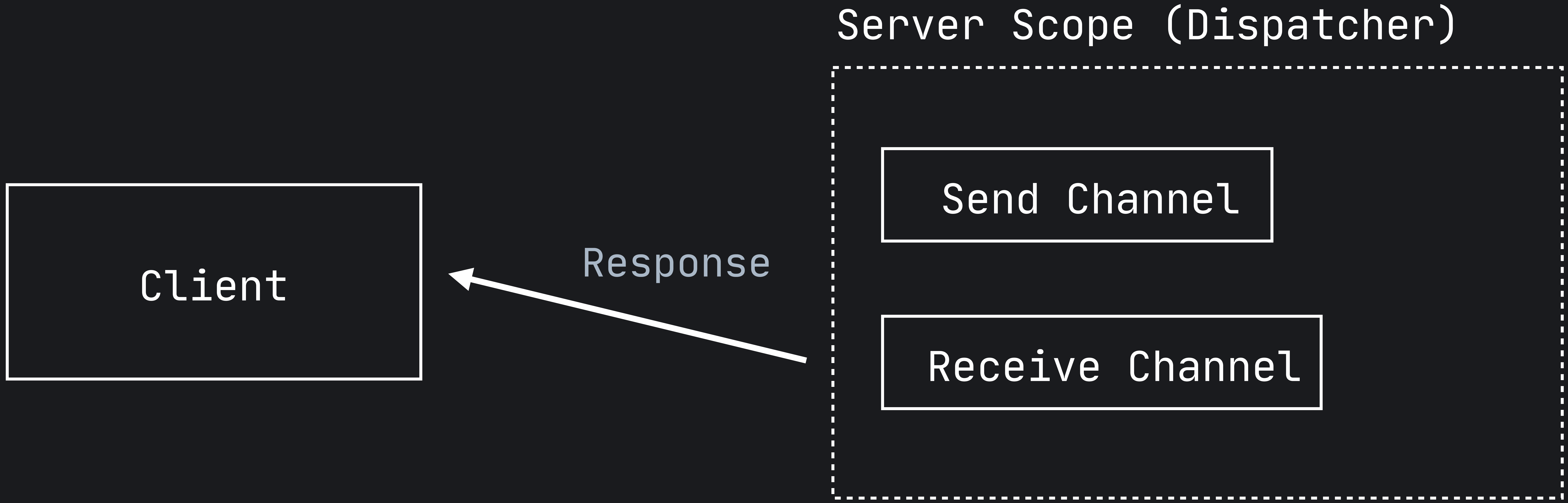
Streaming


Server Scope (Dispatcher)
Client
Request
Send Channel
Receive Channel

Streaming


Server Scope (Dispatcher)
Request
Send Channel
Receive Channel
Client

Streaming


Server Scope (Dispatcher)
Send Channel
Receive Channel
Client
Response

Generate Service

```
abstract class PlacesImplBase {

    open suspend fun getPlace(…): Place

    open suspend fun listPlaces(…)

    open suspend fun chat(…)

    open suspend fun recordTrip(…)

}
```

Implement Service

```
class PlacesService(): PlacesImplBase() {

                          ⬆
                 Inherit base Implementation










}
```

Implement Service

```
class PlacesService(): PlacesImplBase() {

  override val initialContext: CoroutineContext
    get() = Dispatchers.IO




}
```

Specify Dispatcher

Implement Service

```
class PlacesService(dispatcher): PlacesImplBase() {

  override val initialContext: CoroutineContext
     get() = Dispatchers.IO









}
```

Implement Service

```
class PlacesService(dispatcher): PlacesImplBase() {

  override val initialContext: CoroutineContext
    get() = dispatcher




}
```

⬆

Specify Dispatcher

Implement Service

```kotlin
class PlacesService(dispatcher): PlacesImplBase() {

  override suspend fun getPlace(request: Location): Place {


  }


}
```

Implement Service

```
class PlacesService(dispatcher): PlacesImplBase() {


  override suspend fun getPlace(request: Location): Place {
      val placeFromDB = getPlacesFromDb()
                          .first { it.location == request }


   }


}
```

Get Place from DB

Implement Service

```
class PlacesService(dispatcher): PlacesImplBase() {


  override suspend fun getPlace(request: Location): Place {
      val placeFromDB = getPlacesFromDb()
                          .first { it.location == request }
      return Place {
          name = placeFromDb.name
          ...
      }
  }
```

Map it to Proto Message

Implement Service

```
abstract class PlacesImplBase

    rpc ListPlaces(Area) returns (stream Place) {};
```

$$\downarrow$$

```
    suspend fun listPlaces(
            request: Area,
            responseChannel: SendChannel<Place>
    )
```

Implement Service

```
abstract class PlacesImplBase

  suspend fun listPlaces(
        area: Area,
        responseChannel: SendChannel<Place>
   ) {
        val places = getPlaces(area)
        places.forEach { responseChannel.send(it) }
   }
}
```

Send to Channel

Implement Service

```
abstract class PlacesImplBase

  suspend fun listPlaces(
        area: Area,
        responseChannel: SendChannel<Place>
   ) {
        val places = getPlaces(area)
        places.forEach { responseChannel.send(it) }
   }
}
```

How does channel close?

Close Channel

kroto-plus

kroto-plus/ServerCalls.kt

```
rpcScope.launch {
    block(responseChannel)
    responseChannel.close()    ⬅   Closes Channel
}
```

Errors

```
abstract class PlacesImplBase

   suspend fun listPlaces(
          area: Area,
          responseChannel: SendChannel<Place>
    ) {
          val places = getPlaces(area)
          places.forEach { responseChannel.send(it) }
    }
}
```

Exception occurs?

Errors

kroto-plus

kroto-plus/ServerCalls.kt

```
rpcScope.launch {
  try {
    block(responseChannel)
    responseChannel.close()
  } finally {
   cancelScope()
  }
```

Clean up on error

Implement Service

```
class PlacesService(dispatcher): PlacesImplBase {

    suspend fun getPlace(…): Place

    suspend fun listPlaces(…)

    suspend fun chat(…)

    suspend fun recordTrip(…)

}
```

Building gRPC Server

- Configure Server with Kroto-Plus ✓
- Create Service with Protocol Buffers ✓
- Implement Service ✓
- Start Server ⇦

grpc-java

grpc-java/ServerBuilder.java

```
class ServerBuilder {

    forPort(port)

    addService(service)

    interceptor(interceptor)
```

Configure gRPC Server

```
val server: Server = ServerBuilder
        .forPort(port)
        .addService(PlacesService())
        .build()
```

Configure gRPC Server

```
val server: Server = ServerBuilder
        .forPort(port)
        .addService(PlacesService())
        .build()
```

Start gRPC Server

```
fun start() {
    server.start()
    Runtime.getRuntime().addShutdownHook(
            Thread {
                server.shutdown()
            }
    )
}
```

Start gRPC Server

```kotlin
fun main() {
    val port = 50051
    val server = configureServer(port)
    server.start()
    server.blockUntilShutdown()
}
```

Building gRPC Server

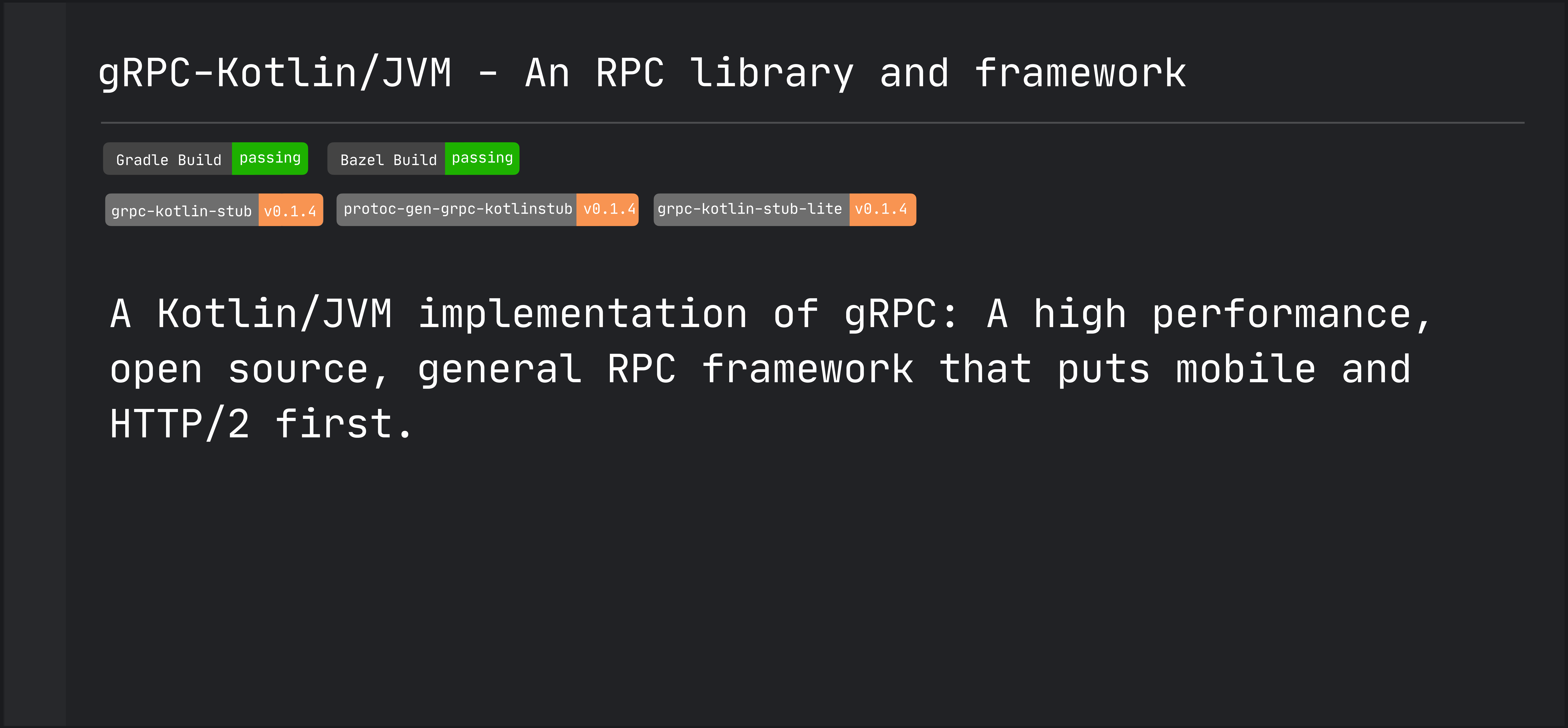
- Configure Server with Kroto-Plus ✓
- Create Service with Protocol Buffers ✓
- Implement Service ✓
- Start Server ✓

Resources

- **gRPC Java**
  https://github.com/grpc/grpc-java
- **Kroto-plus**
  https://github.com/marcoferrer/kroto-plus
- **Protocol Buffers**
  https://developers.google.com/protocol-buffers/docs/overview
- **Micronaut**
  https://micronaut.io/

Building gRPC Client

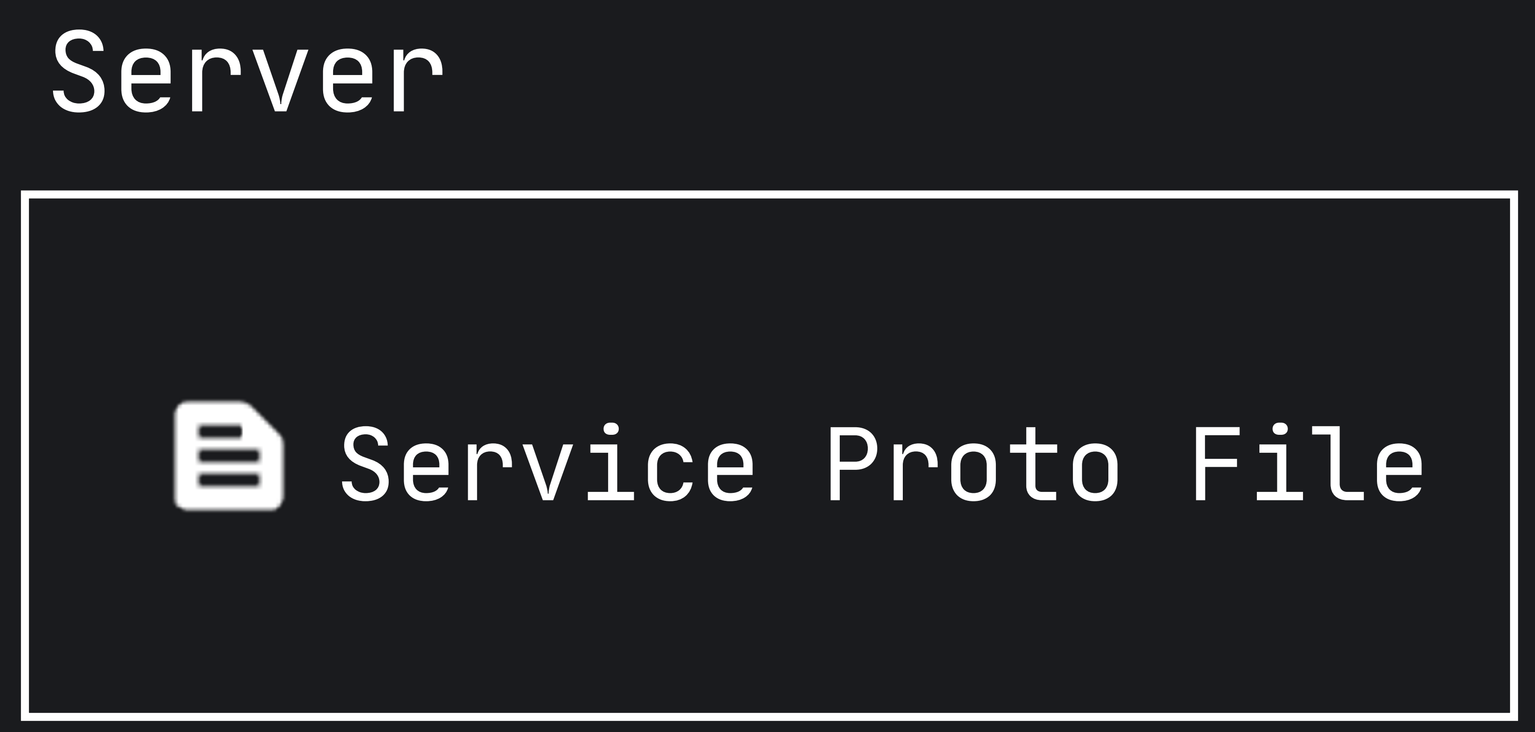
grpc/grpc-kotlin

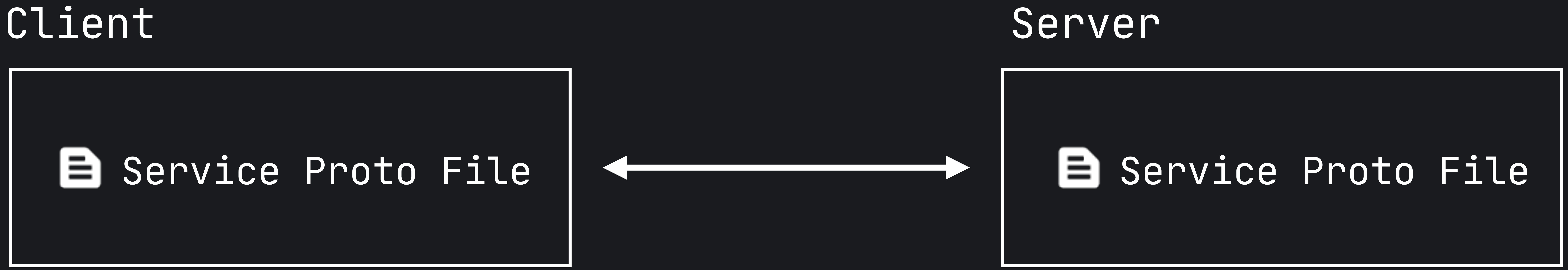


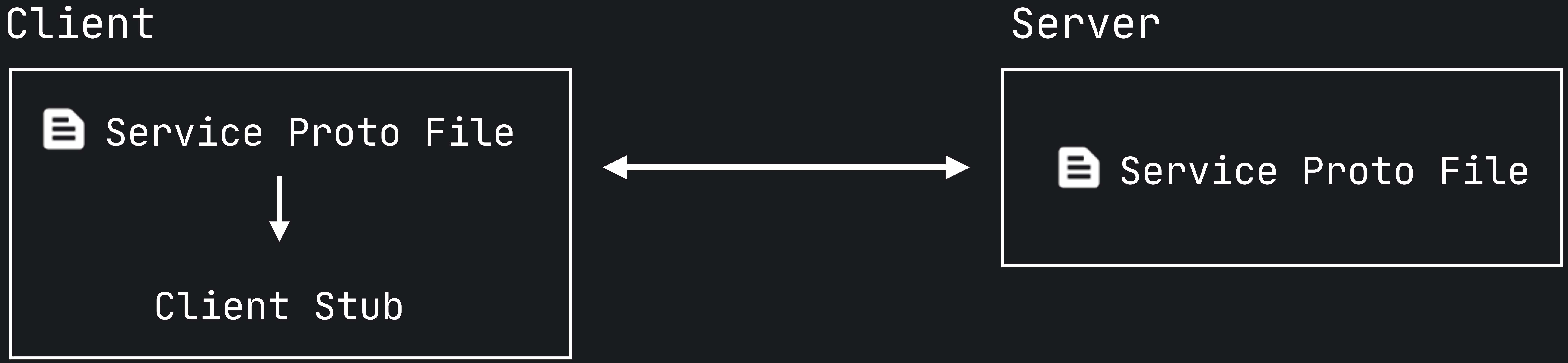
gRPC-Kotlin/JVM - An RPC library and framework

Gradle Build passing
Bazel Build passing
grpc-kotlin-stub v0.1.4
protoc-gen-grpc-kotlinstub v0.1.4
grpc-kotlin-stub-lite v0.1.4

A Kotlin/JVM implementation of gRPC: A high performance, open source, general RPC framework that puts mobile and HTTP/2 first.


Server
Service Proto File


Client
Service Proto File
Server
Service Proto File


Client
Service Proto File
Client Stub
Server
Service Proto File

Generated Client

```
class CoroutineStub {







}
```

Generated Client

```
class CoroutineStub {

    rpc GetPlace(Location) returns (Place) {};





}
```

Generated Client

```
class CoroutineStub {

    rpc GetPlace(Location) returns (Place) {};
                          ↓
    suspend fun getPlace(request: Location): Place


}
```

Generated Client

```
class CoroutineStub {

    rpc ListPlaces(Area) returns (stream Place) {};
                       ↓
    fun listPlaces(request: Area): Flow<Place>
                                      ⬆

                          Stream is map to Flow
}
```

Generated Client

```
class CoroutineStub {

    rpc Chat(stream Comment) returns (stream Comment) {};
                                  ↓
    fun chat(requests: Flow<Comment>): Flow<Comment>
                       ⇧
             Stream is map to Flow

}
```

Generated Client

```
class CoroutineStub {

    suspend fun getPlace(request: Location): Place

    fun recordTrip(requests: Flow<Location>): TripSummary

    fun listPlaces(request: Area): Flow<Place>

    fun chat(requests: Flow<Comment>): Flow<Comment>

}
```

Using Client

```
val managedChannel = ManagedChannelBuilder
    .forAddress(host, port)
    .useTransportSecurity()
    .build()
```

⬅ Specify host and port

Using Client

```
val managedChannel = ManagedChannelBuilder
    .forAddress(host, port)
    .useTransportSecurity()
    .build()
```

for https

Using Client

```
val managedChannel = ManagedChannelBuilder
    .forAddress(host, port)
    .intercept(object : ClientInterceptor {
        fun interceptCall(method, callOptions, channel) {


    })
  .build()
```

Intercept RPC Calls

Using Client

```
val managedChannel = ManagedChannelBuilder
    .forAddress(host, port)
    .useTransportSecurity()
    .build()
```

Using Client

```kotlin
val managedChannel = ManagedChannelBuilder
    .forAddress(host, port)
    .useTransportSecurity()
    .build()

val client = CoroutineStub(managedChannel)
```

Using Client

```
class GrpcViewModel(val client: CoroutineStub): ViewModel() {







}
```

Using Client

```kotlin
class GrpcViewModel(val client: CoroutineStub): ViewModel() {

    fun getPlace(location: Location) {
        viewModelScope.launch {
            val place = client.getPlace(location)
        }
    }


}
```

Using Client

```
class GrpcViewModel(val client: CoroutineStub): ViewModel() {

    fun listPlaces(area: Area) {
        viewModelScope.launch {
            val places: Flow<Place> = client.listPlaces(area)
            places.collect {

            }
        }
    }

}
```

Using Client

Stream of comments

```
class GrpcViewModel(val client: CoroutineStub): ViewModel() {

    fun chat(comments: ReceiveChannel<Comment>) {
        viewModelScope.launch {
            client.chat(comments.consumeAsFlow()).collect { }
        }
    }


}
```

Using Client

```
class GrpcViewModel(val client: CoroutineStub): ViewModel() {

    fun chat(comments: ReceiveChannel<Comment>) {
        viewModelScope.launch {
            client.chat(comments.consumeAsFlow())
                .collect {

                }
        }
    }
}
```

Convert Channel to Flow

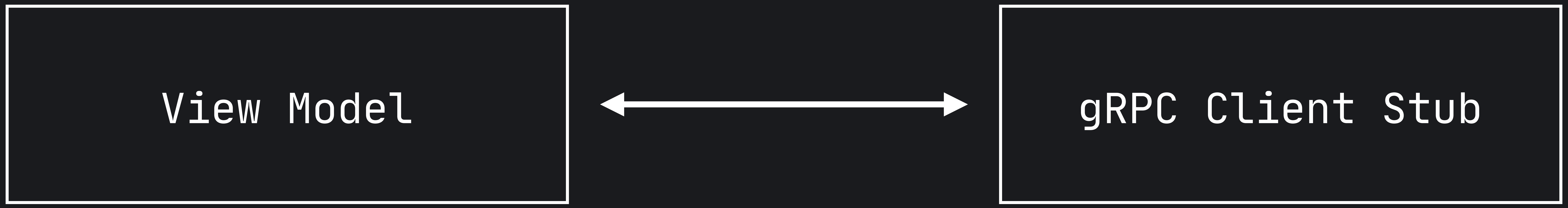
Using Client

```
class GrpcViewModel(val client: CoroutineStub): ViewModel() {

    fun chat(comments: ReceiveChannel<Comment>) {
        viewModelScope.launch {
            client.chat(comments.consumeAsFlow())
                  .collect {

                  }
            }
        }
}
```

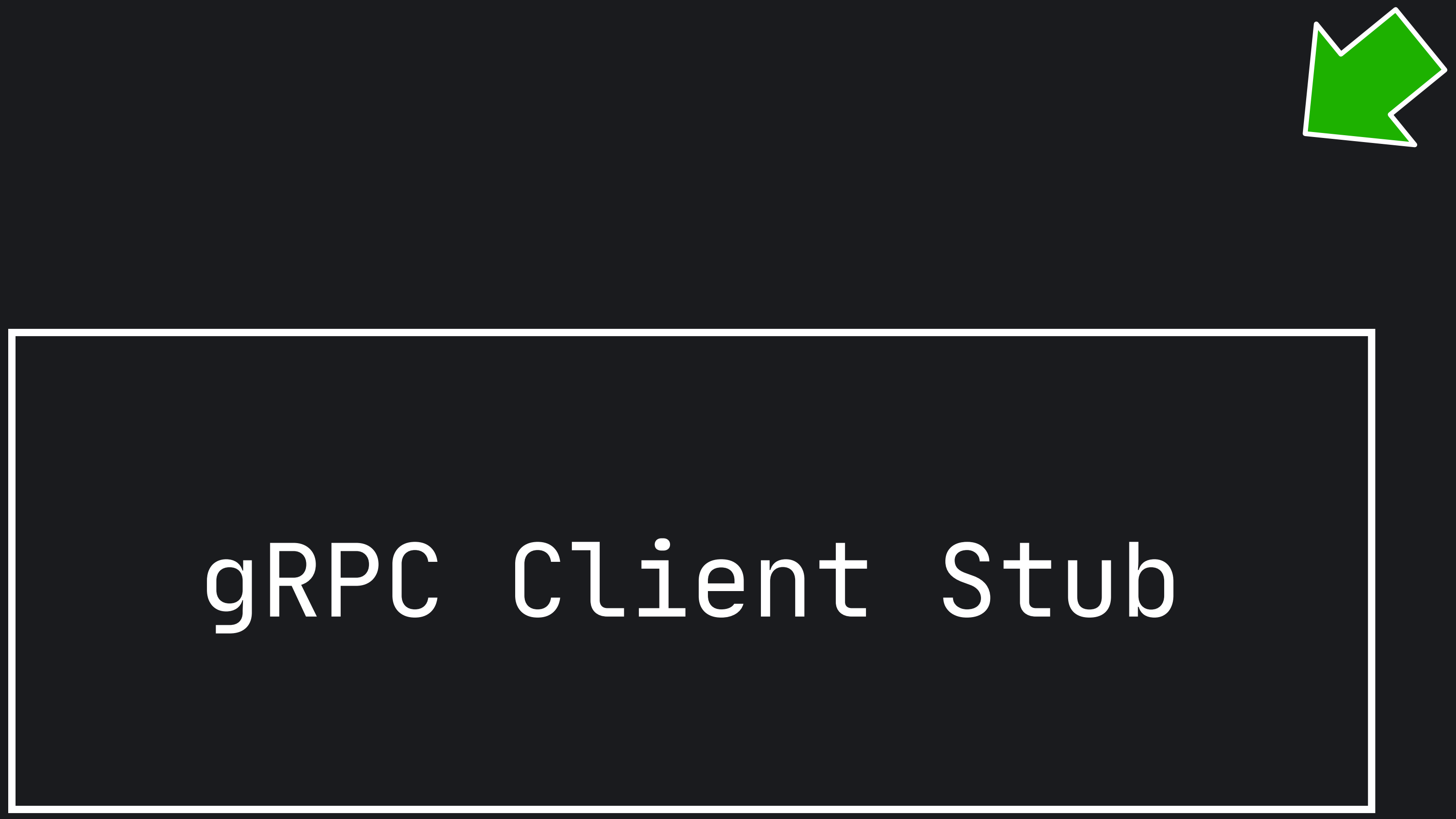
Collect from Flow

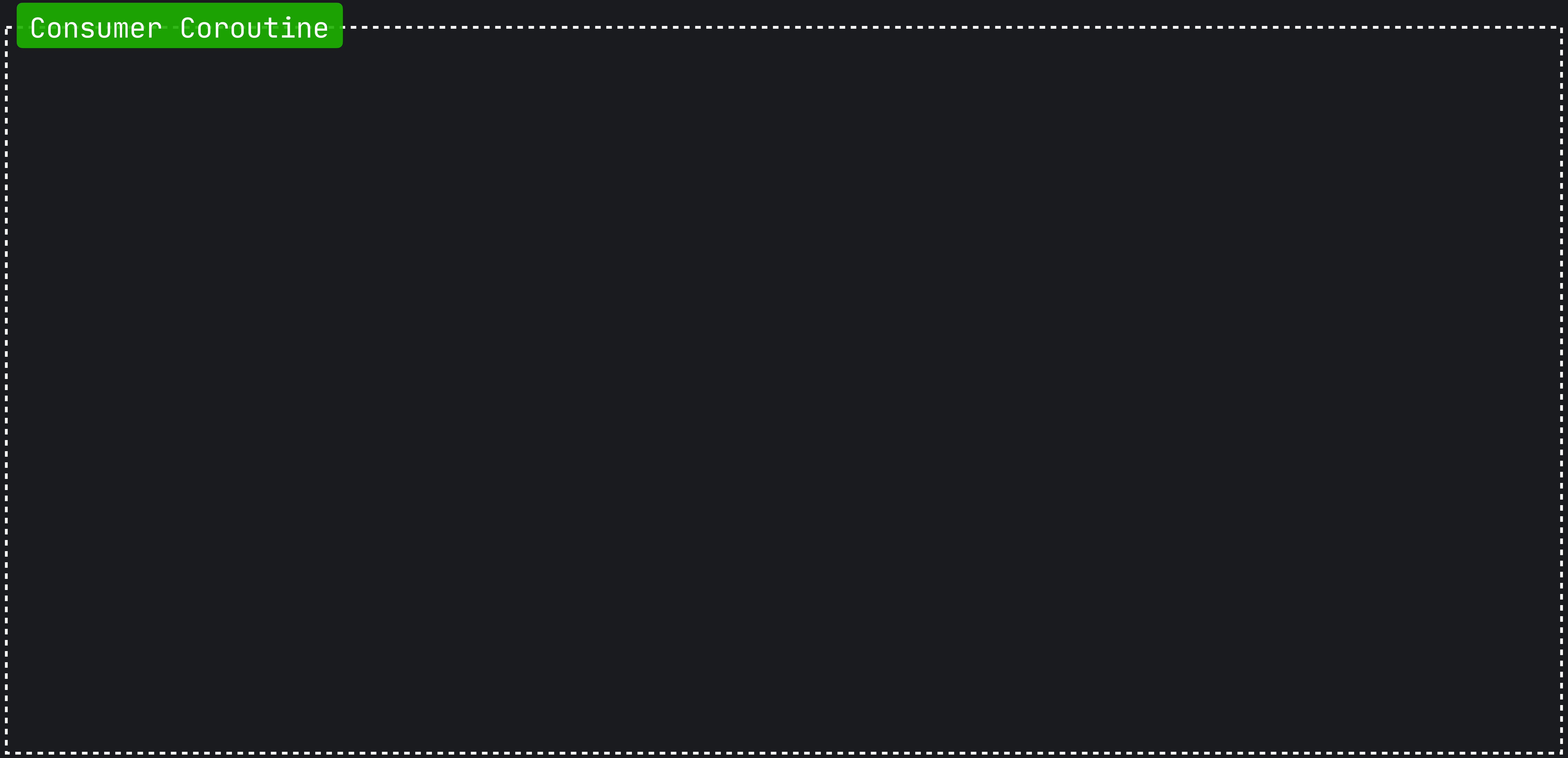
Using Client

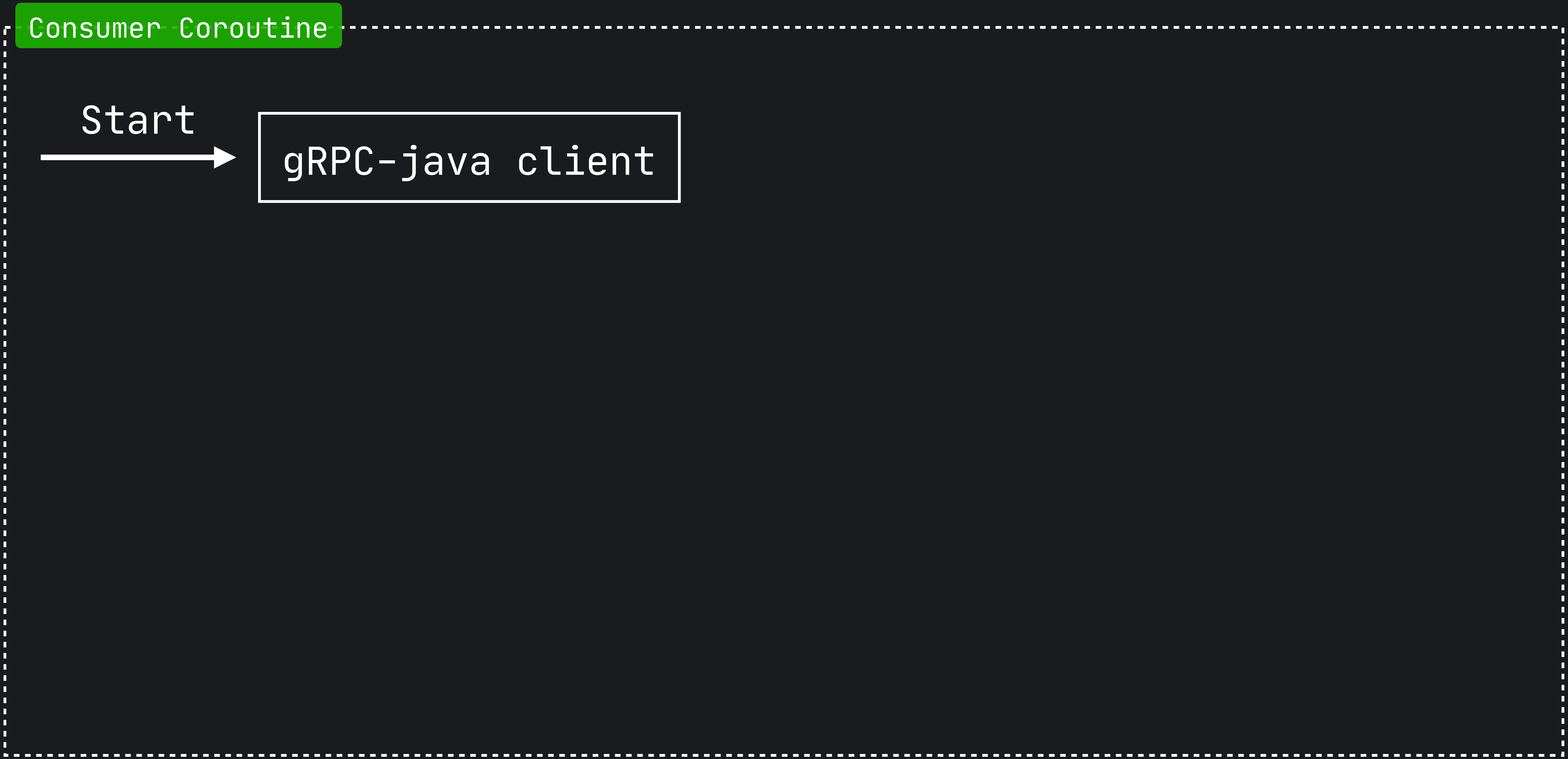
```kotlin
class GrpcViewModel(val client: CoroutineStub): ViewModel() {

    fun getPlace(location: Location)

    fun listPlaces(area: Area)

    fun chat(chat: ReceiveChannel<Comment>)

}
```

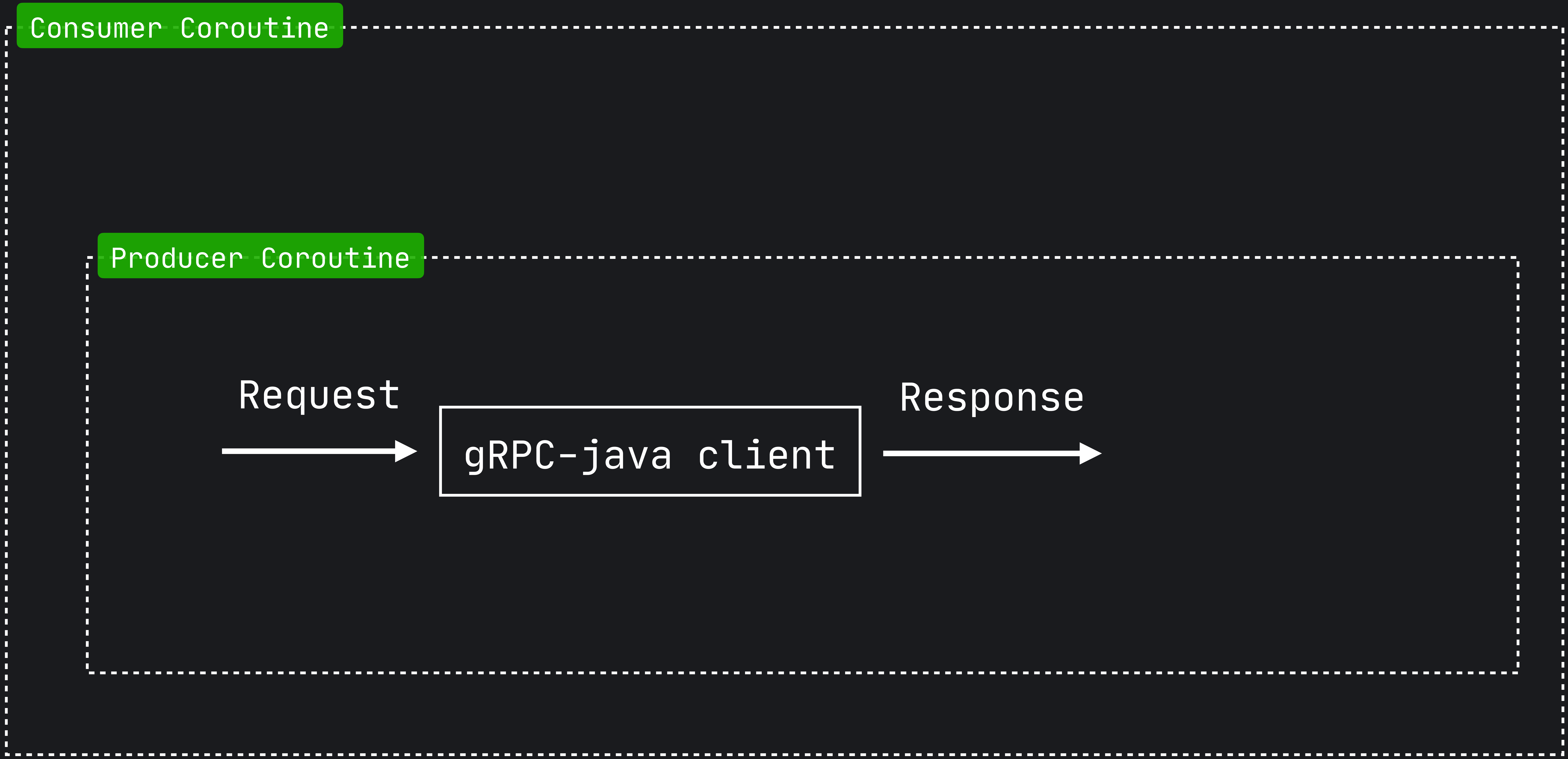

View Model
gRPC Client Stub

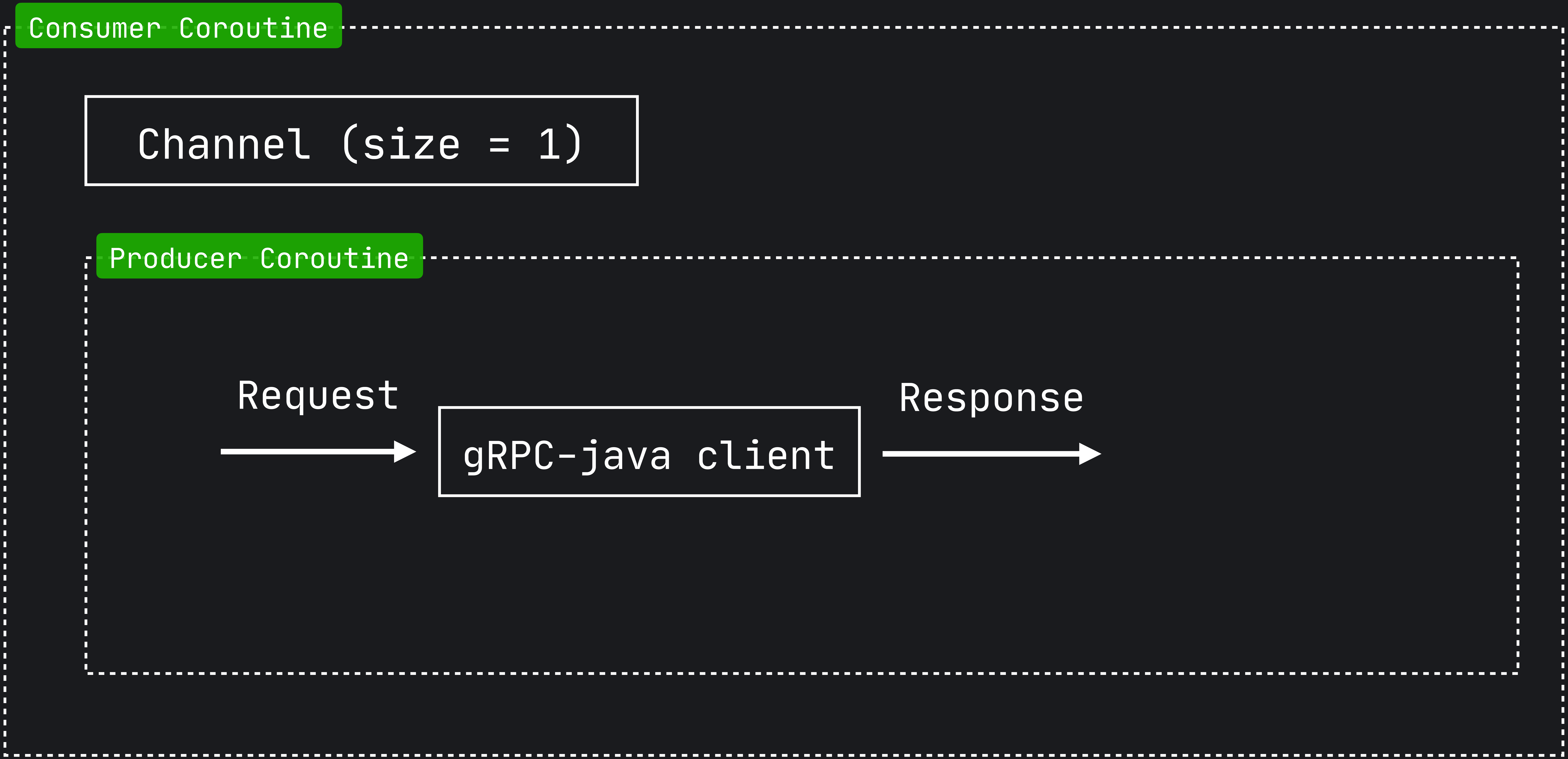
How does it use coroutines?

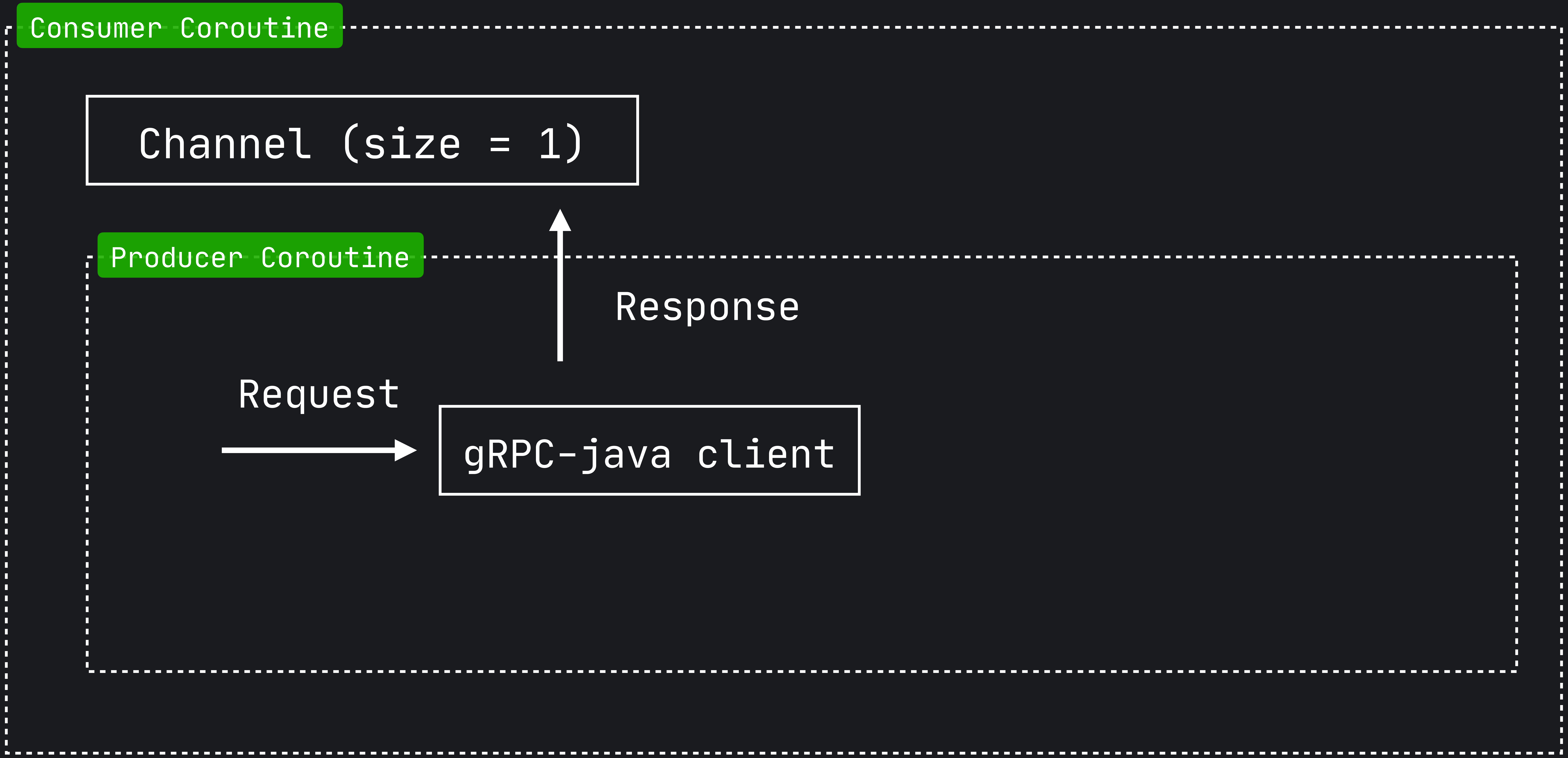


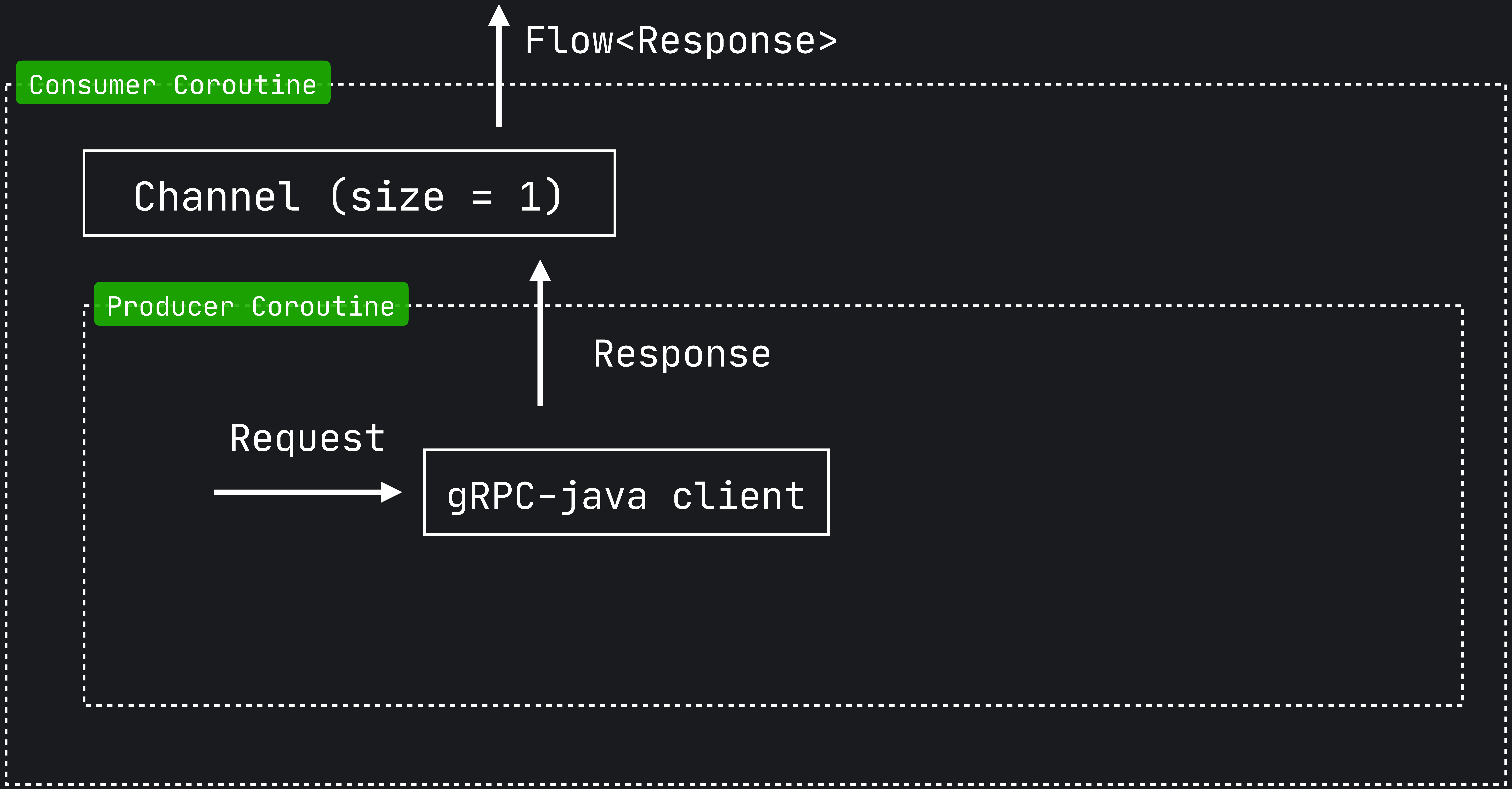

Consumer Coroutine

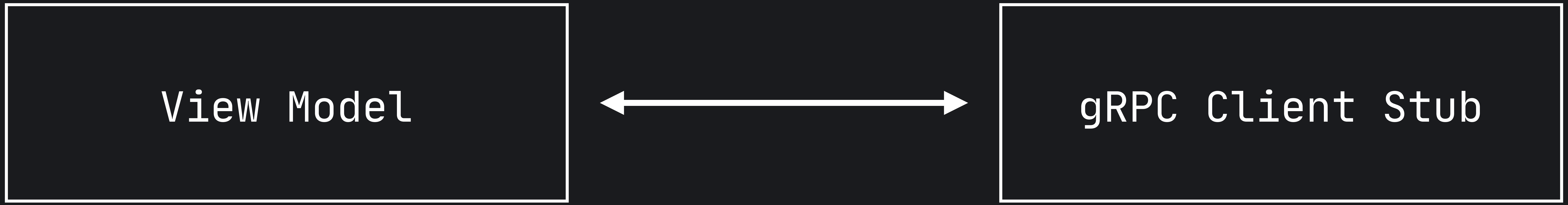

Consumer Coroutine
Start
gRPC-java client


Consumer Coroutine
Producer Coroutine
Request
gRPC-java client
Response


Consumer Coroutine
Channel (size = 1)
Producer Coroutine
Request
gRPC-java client
Response

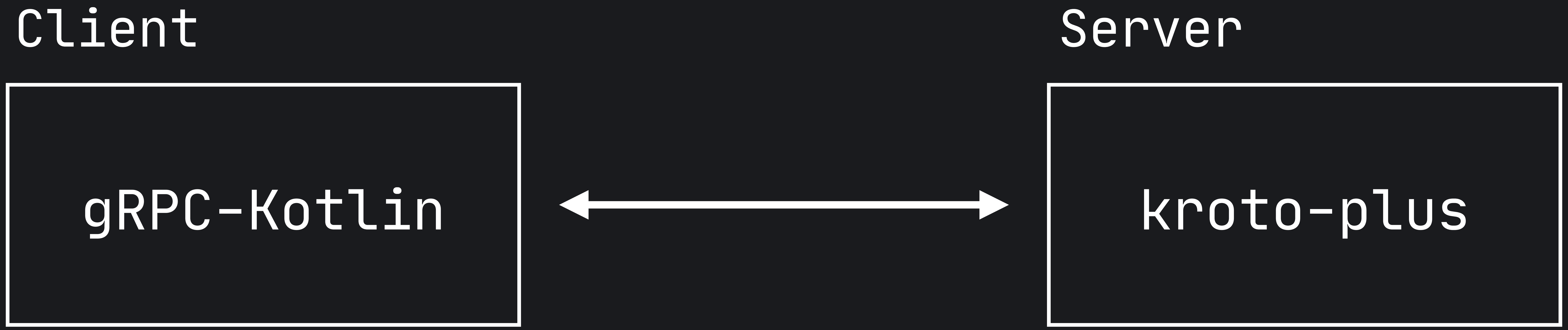

Consumer Coroutine
Channel (size = 1)
Producer Coroutine
Response
Request
gRPC-java client


Flow<Response>
Consumer Coroutine
Channel (size = 1)
Producer Coroutine
Response
Request
gRPC-java client

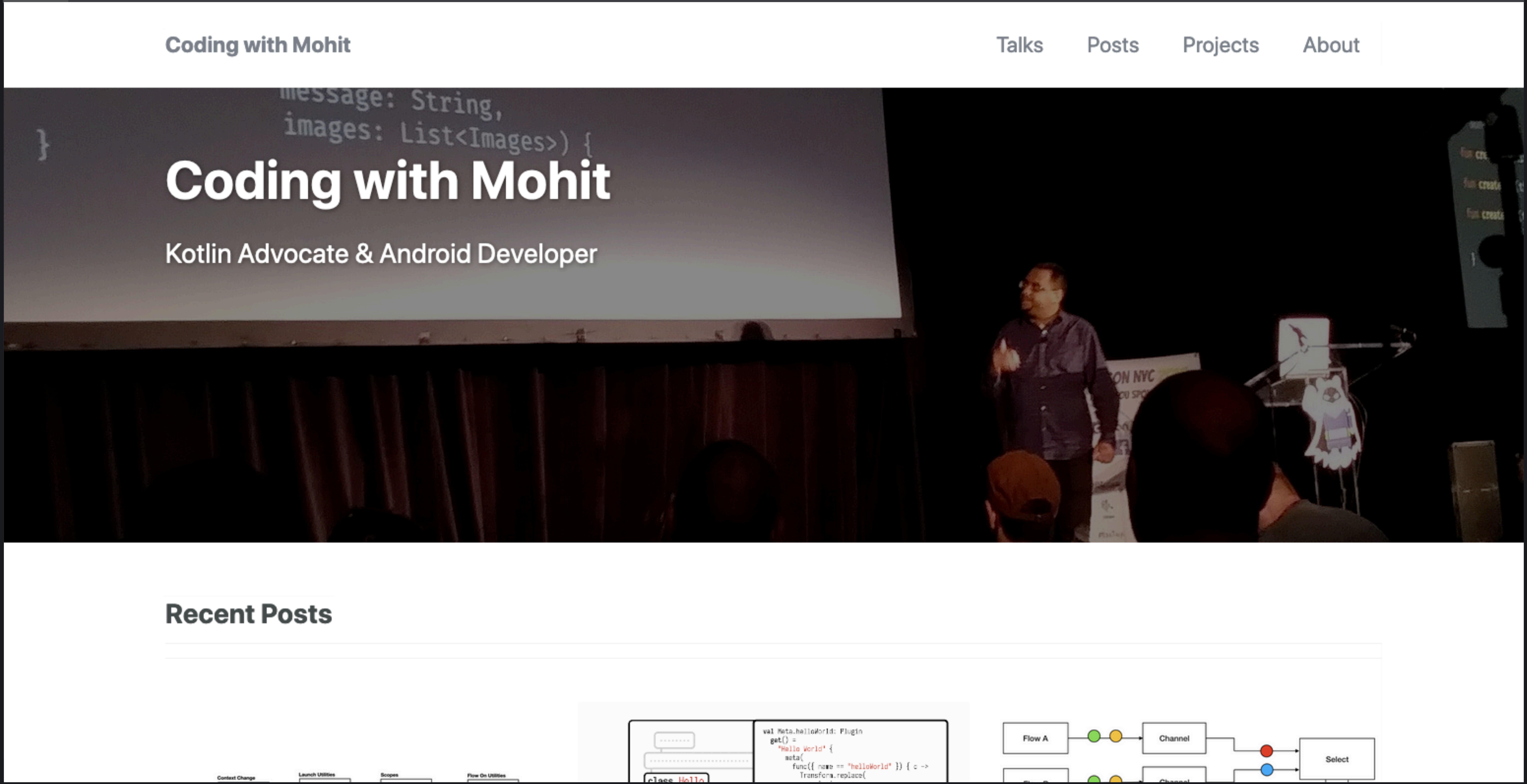

View Model
gRPC Client Stub

Resources

- gRPC Kotlin
  https://github.com/grpc/grpc-kotlin
- Wire
  https://github.com/square/wire


Client
gRPC-Kotlin
Server
kroto-plus

https://codingwithmohit.com/



Coding with Mohit

Talks Posts Projects About

Coding with Mohit

Kotlin Advocate & Android Developer

Recent Posts

Resources

- **Unit Testing Delays, Errors & Retries with Kotlin Flows**

  https://codingwithmohit.com/coroutines/unit-testing-delays-errors-retries-with-kotlin-flows/

- **Kotlin Assert Flow Delight**

  https://codingwithmohit.com/coroutines/kotlin-assert-flow-delight/

- **Channels & Flows in Practice**

  https://speakerdeck.com/heyitsmohit/channels-and-flows-in-practice

Thank You!

www.codingwithmohit.com

www.twitter.com/heyitsmohit