Building microservices with Kotlin and gRPC

Marharyta Nedzelska

@jMargaritaN

Copenhagen

Denmark

Who am I?

Marharyta Nedzelska

- Software Engineer @ Wix
- KKUG & KNight Kyiv, Devoxx UA
- https://medium.com/@margoqueen95
- @jMargaritaN twitter

One more thing you should know

I like boxing!

So think twice before asking tricky questions!

AGENDA

Let’s talk about gRPC

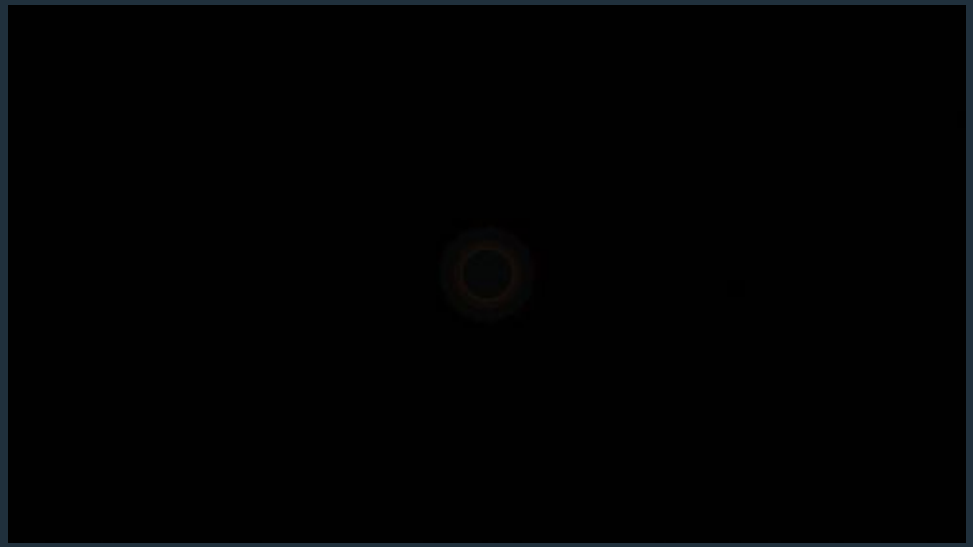
Implement Service

gRPC and coroutines

gRPC Kotlin libs

00

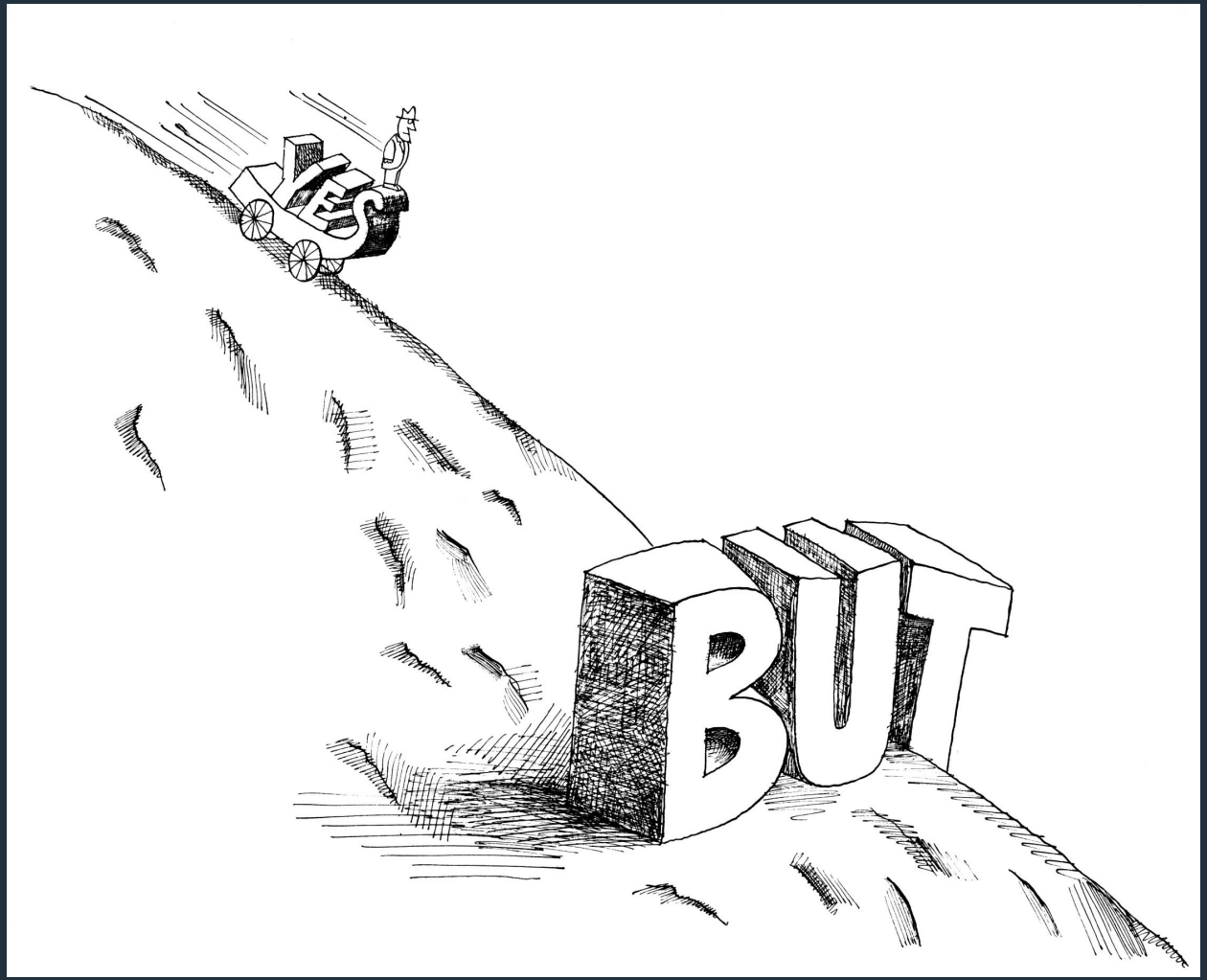
The very beginning...



Who writes microservices today, please, raise your hands?

Microservices Rock!

- Independent
- Scalable
- Best suitable tool
- Parallel development
- etc...


YES
BUT

Btw, who said this?

The biggest issue in changing a monolith into microservices lies in changing the communication pattern.

Btw, who said this?

The biggest issue in changing a monolith into microservices lies in changing the communication pattern.

Martin Fowler

Who uses HTTP 1.1 and REST today, please, raise your hands?


Clients

01

Let’s discover gRPC

Riddle

What does ‘gRPC’ stand for?

~~*“gRPC” means gRPC Remote Procedure Call*~~

'g' stands for something different every gRPC release:

- 1.0 'g' stands for 'gRPC'
- 1.1 'g' stands for 'good'
- 1.2 'g' stands for 'green'
- 1.3 'g' stands for 'gentle'
- 1.4 'g' stands for 'gregarious'
- 1.6 'g' stands for 'garcia'
- 1.7 'g' stands for 'gambit'
- 1.8 'g' stands for 'generous'
- 1.9 'g' stands for 'glossy'
- 1.10 'g' stands for 'glamorous'
- 1.11 'g' stands for 'gorgeous'
- 1.12 'g' stands for 'glorious'
- 1.13 'g' stands for 'gloriosa'
- 1.14 'g' stands for 'gladiolus'
- 1.15 'g' stands for 'glider'
- 1.16 'g' stands for 'gao'
- 1.17 'g' stands for 'gizmo'
- 1.18 'g' stands for 'goose'
- 1.19 'g' stands for 'gold'
- 1.20 'g' stands for 'godric'
- 1.21 'g' stands for 'gandalf'
- 1.22 'g' stands for 'gale'
- 1.23 'g' stands for 'gangnam'
- 1.24 'g' stands for 'ganges'
- 1.25 'g' stands for 'game'

That's 'g'. What the hell is 'RPC'?

In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure to execute in a different address space, which is coded as if it was a normal (local) procedure call.

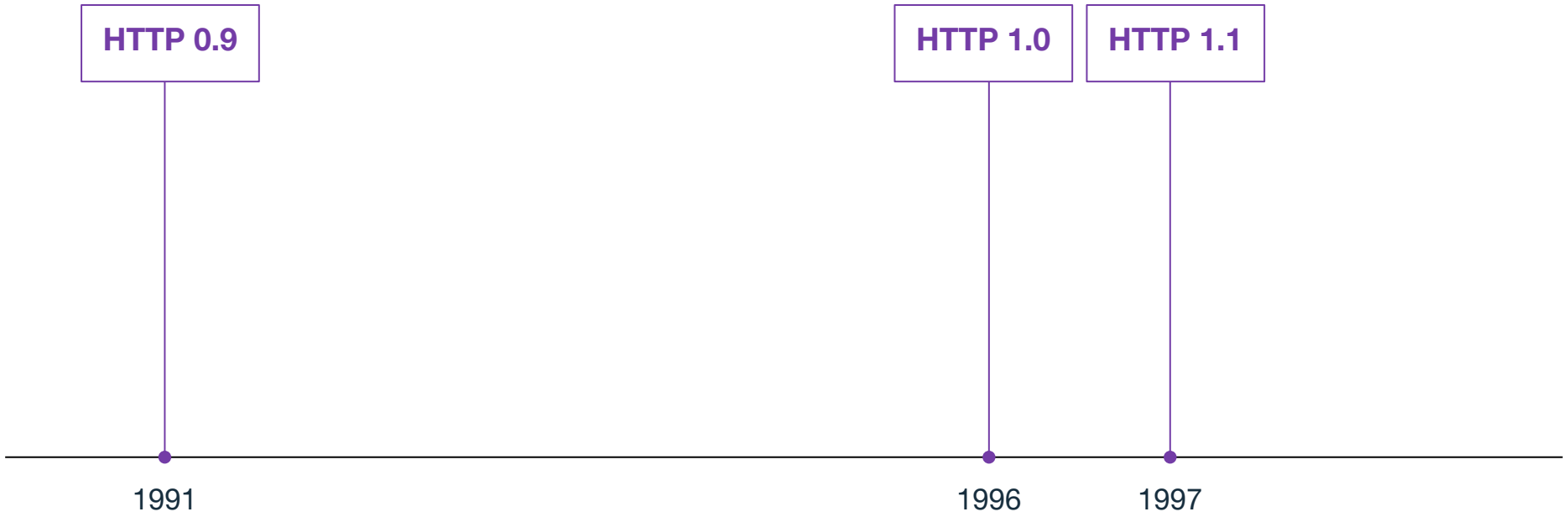
https://en.wikipedia.org/wiki/Remote_procedure_call

gRPC Features

gRPC Features

- **Use HTTP/2**

HTTP protocol evolution



HTTP protocol evolution

No progress here

1998 1999 2000

HTTP protocol evolution

And here

2001 2002 2003

HTTP protocol evolution

And still no progress...

2004 2005 2006

HTTP protocol evolution

Are they alive?

2007 2008 2009

HTTP protocol evolution

I guess no...

2010 2011 2012

HTTP protocol evolution

And suddenly!

2013 2014 2015

HTTP protocol evolution

HTTP/2

2015

gRPC Features

- Use HTTP/2
  - Single TCP connection

gRPC Features

- Use HTTP/2
  - Single TCP connection
  - Bidirectional streaming

gRPC Features

- Use HTTP/2
  - Single TCP connection
  - Bidirectional streaming
  - Flow control

gRPC Features

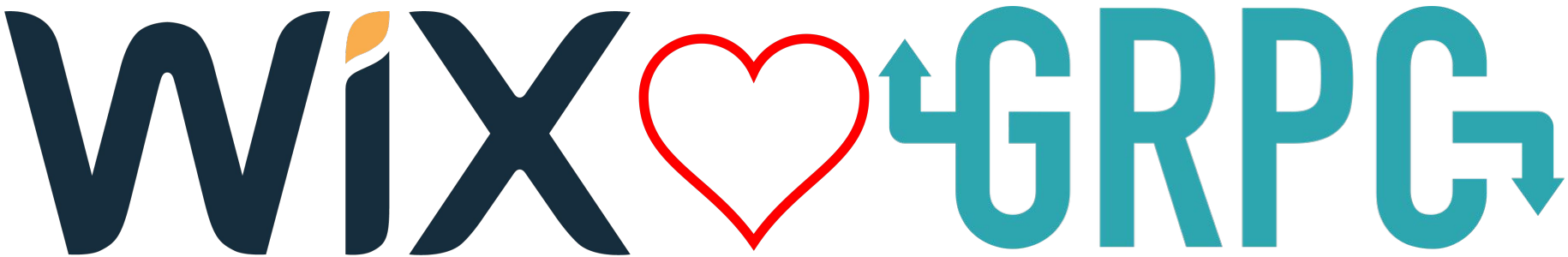
- Use HTTP/2
  - Single TCP connection
  - Bidirectional streaming
  - Flow control
- Supports multiple languages

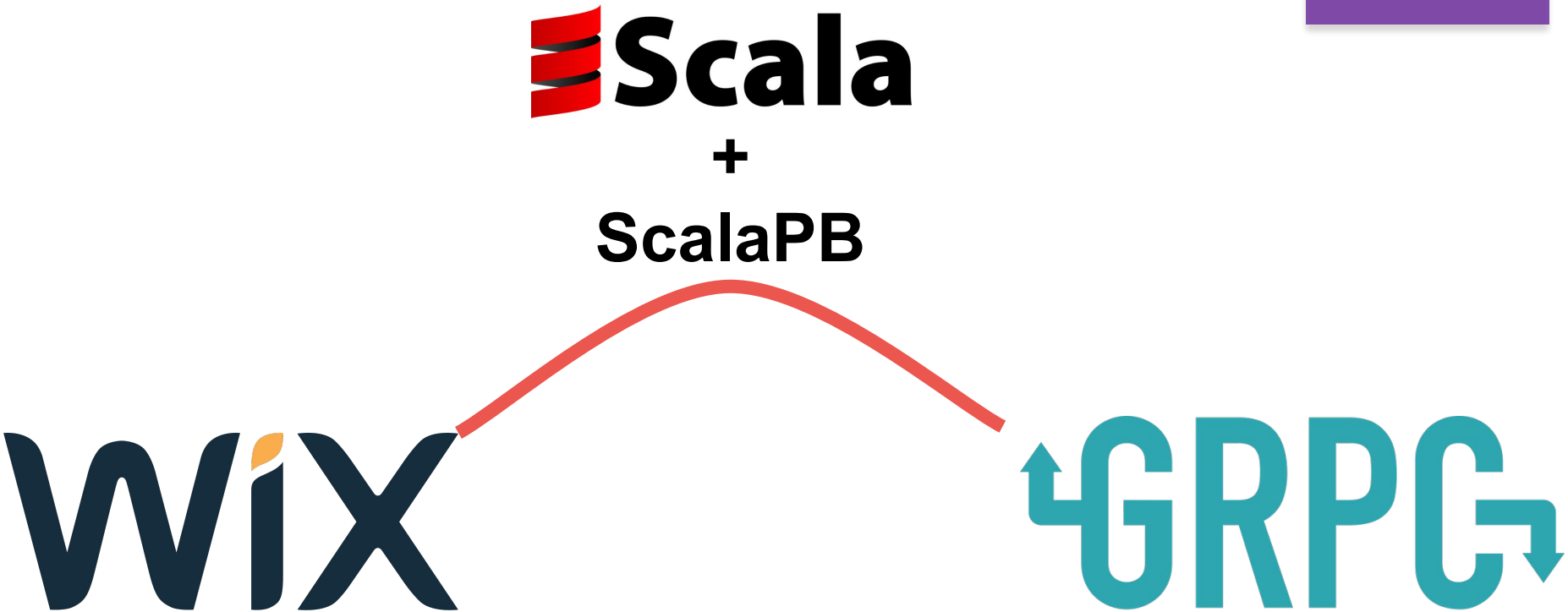
gRPC Features

- Use HTTP/2
  - Single TCP connection
  - Bidirectional streaming
  - Flow control
- Supports multiple languages
- Binary, uses protobuf


Java


GRPC


WIX
GRPC


Scala
+
ScalaPB
WIX
GRPC

?


WIX


GRPC

gRPC Discovered!

- Use HTTP/2
  - Single TCP connection
  - Bidirectional streaming
  - Flow control
- Supports multiple languages
- Binary, uses protobuf

02

Protocol Buffers

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data — think XML, but smaller, faster, and simpler.

https://developers.google.com/protocol-buffers/

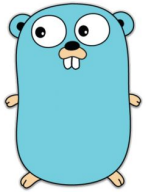
Sample .proto file

```
syntax = "proto3";

package helloworld;

service Greeter {
    rpc SayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {
    string name = 1;
}

message HelloReply {
    string message = 1;
}
```

protobuf
Protocol Buffers

compiling...



Dart

Go





Protobuf summary

- Type Safety

Protobuf summary

- Type Safety
- No schema violations

Protobuf summary

- Type Safety
- No schema violations
- Fast serialization/deserialization

Protobuf summary

- Type Safety
- No schema violations
- Fast serialization/deserialization
- Backward compatibility

Protobuf summary

- ✓ Type Safety
- ✓ No schema violations
- ✓ Fast serialization/deserialization
- ✓ Backward compatibility
- ✗ Human readability

03

Implement Service

We’re building a Death Star!

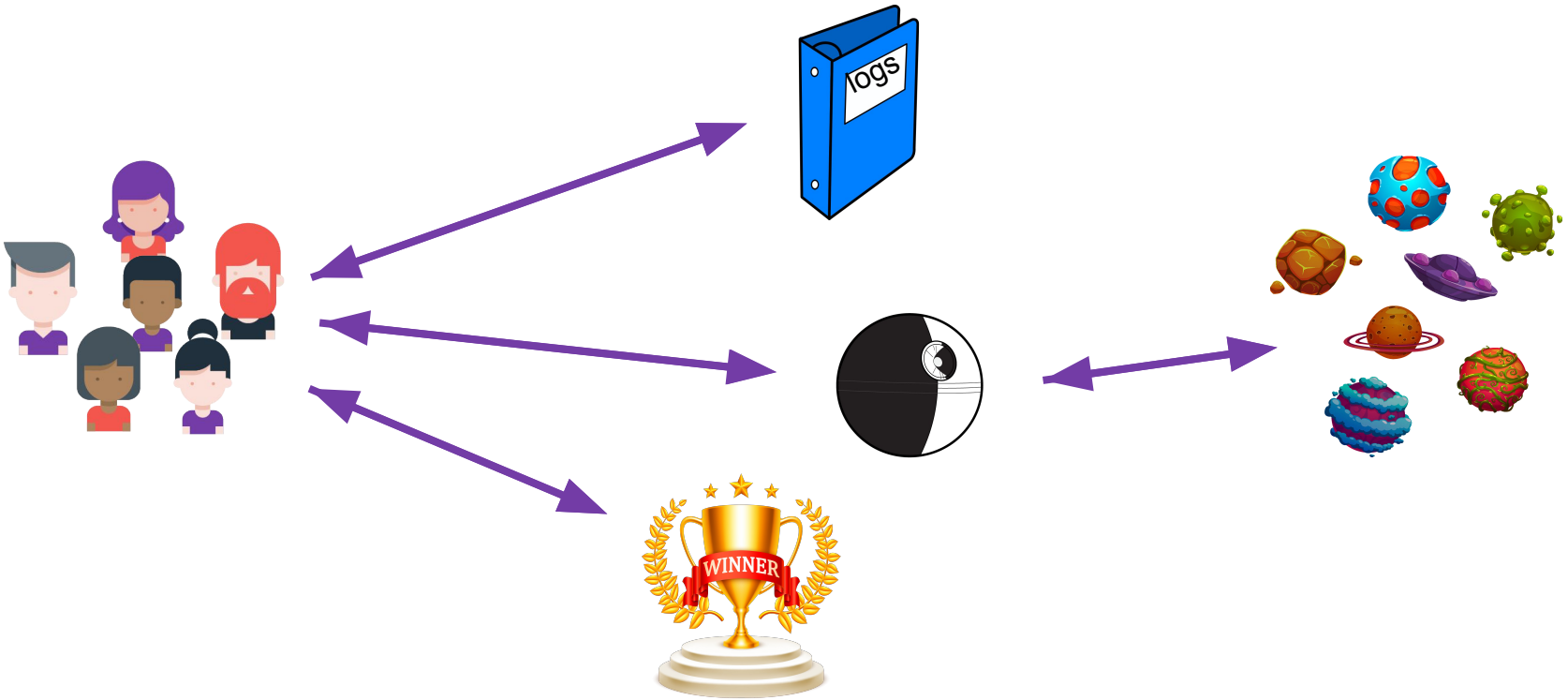
Task for this session

- Write a “Death Star”
- Destroy as much planets as you can
- Tweet with **#kotlinconf #gRPC** and **#Kotlin** tags
- Greet winners at the end of our session!
- Enjoy!

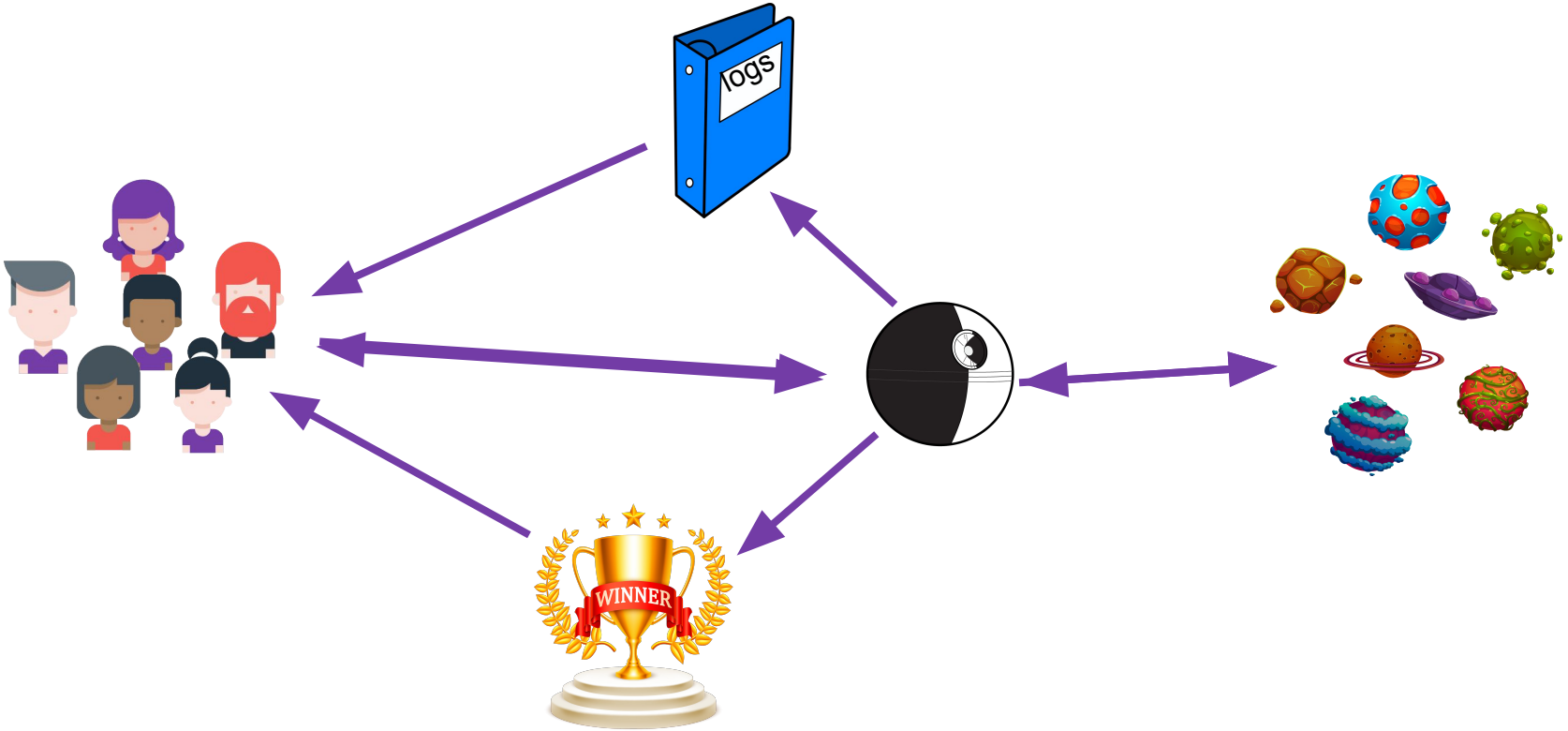
Application is here!

http://104.198.212.208:8080

When user joins ...



When user destroys planet ...



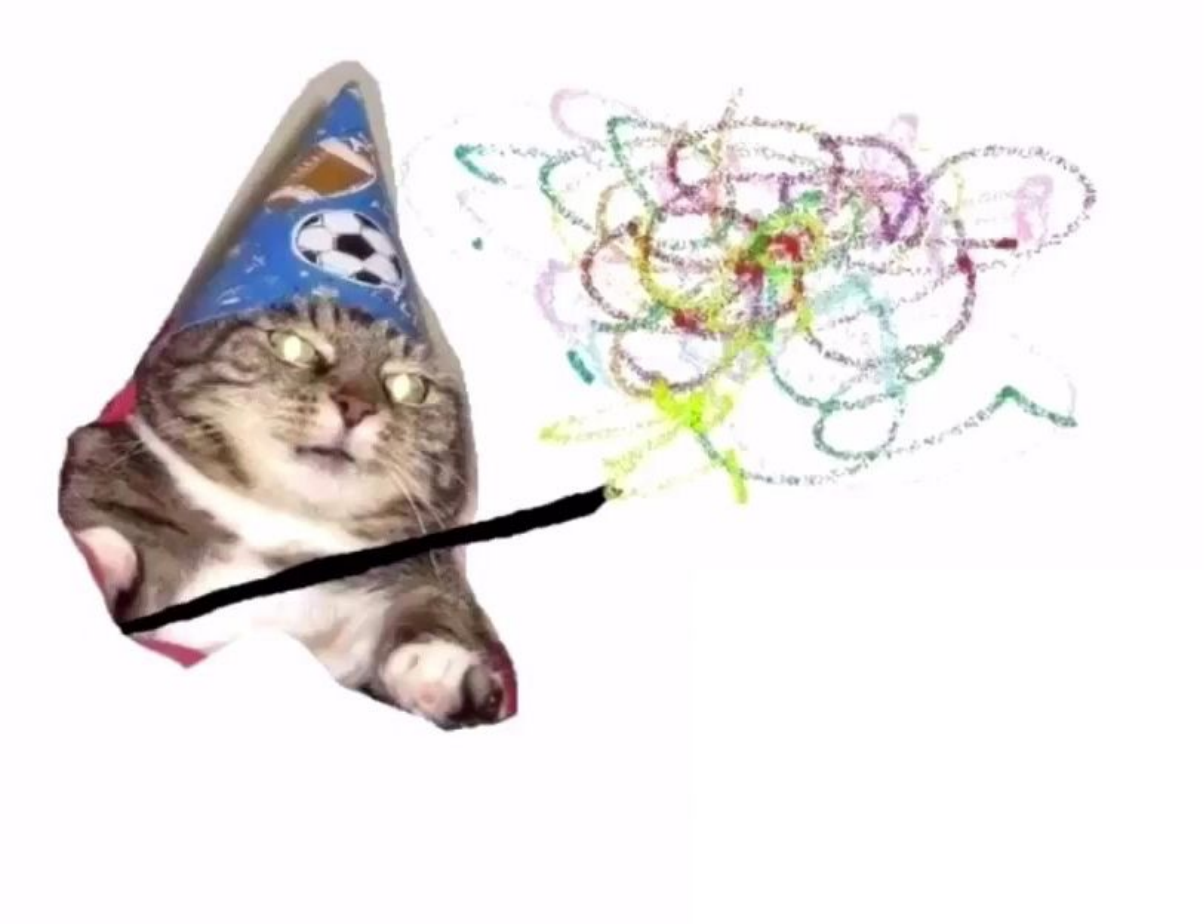
Let’s implement!

First look at proto

```
syntax = "proto3";

option java_package = "ua.nedz.grpc";
option java_outer_classname = "DeathStarProto";
option objc_class_prefix = "DSP";

package deathstar;

import "planet.proto";

service DeathStarService {
  rpc Destroy (stream DestroyPlanetRequest) returns (stream
                                                     Planets) {}
}
```

First look at proto

```
message Planet {
   int64 planetId = 1;
   string name = 2;
   int64 weight = 3;
   int32 img = 4;
}
message Planets {
   repeated Planet planets = 1;
}
message DestroyPlanetRequest {
   string userName = 1;
   int64 planetId = 2;
   int64 weight = 3;
}
```



And we have generated

```
@javax.annotation.Generated(
   value = "by gRPC proto compiler (version 1.16.1)",
   comments = "Source: death-star.proto")
public final class DeathStarServiceGrpc {...}

public final class DeathStarProto {...}

public final class PlanetProto {...}

                                                ...
```

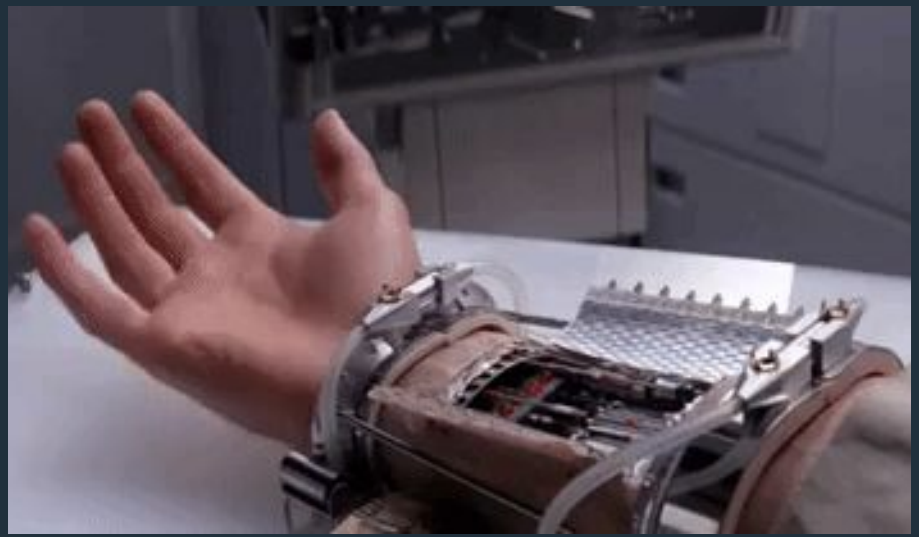
Server is

```
class DeathStarServer (private val port: Int = 50051,
 private val serverBuilder: ServerBuilder<*> =
                                        ServerBuilder.forPort(port)) {
   private lateinit var server: Server
   fun start() {
      server = serverBuilder
          .addService(DeathStarServiceImpl())
          .build()
          .start()
      println("Server started!")
   }
}
```


WELL DONE
MY YOUNG APPRENTICE
memegenerator.net

04

gRPC and coroutines



The whole destroy method...

```
override fun destroy(responseObserver: StreamObserver<Planets>): StreamObserver<DestroyPlanetRequest> {
  listeners.add(responseObserver)
  planetStub.getAllPlanets(Empty.getDefaultInstance(),
        object : StreamObserver<Planets> by DefaultStreamObserver() {
          override fun onNext(planets: Planets) {
            responseObserver.onNext(populateWithCoordinnates(planets))
          }
        })
  return object : StreamObserver<DestroyPlanetRequest> by DefaultStreamObserver() {
    override fun onNext(destroyPlanetRequest: DestroyPlanetRequest) {
      planetStub.removePlanet(RemovePlanetRequest { planetId = destroyPlanetRequest.planetId },
            object : StreamObserver<PlanetServiceProto.RemovePlanetResponse> by DefaultStreamObserver() {
              override fun onNext(removePlanetResponse: PlanetServiceProto.RemovePlanetResponse) {
                if (removePlanetResponse.result) {
                  scoreStub.addScore(AddScoreRequest {
                    userName = destroyPlanetRequest.userName
                    toAdd = destroyPlanetRequest.weight
                  }, object : StreamObserver<Empty> by DefaultStreamObserver() {})
                  logStub.destroyedPlanet(destroyPlanetRequest, object : StreamObserver<Empty>
                  by DefaultStreamObserver() {})
                  planetStub.generateNewPlanet(Empty.getDefaultInstance(),
                        object : StreamObserver<PlanetProto.Planet> by DefaultStreamObserver() {
                          override fun onNext(planet: PlanetProto.Planet) {
                            logStub.newPlanet(planet, object : StreamObserver<Empty>
                            by DefaultStreamObserver() {})
                            listeners.forEach {
                              it.onNext(Planets {
                                addPlanets(populateWithCoordinates(planet,
                                      destroyPlanetRequest.coordinates.x,
                                      destroyPlanetRequest.coordinates.y))
                              })
                            }
                          }
                        })
                }
              }
            })
    }
  }
}
```

The whole destroy method...

```
override fun destroy(responseObserver: StreamObserver<Planets>): StreamObserver<DestroyPlanetRequest> {
  listeners.add(responseObserver)
  planetStub.getAllPlanets(Empty.getDefaultInstance(),
        object : StreamObserver<Planets> by DefaultStreamObserver() {
          override fun onNext(planets: Planets) {
            responseObserver.onNext(populateWithCoordinnates(planets))
          }
        })
  return object : StreamObserver<DestroyPlanetRequest> by DefaultStreamObserver() {
    override fun onNext(destroyPlanetRequest: DestroyPlanetRequest) {
      planetStub.removePlanet(RemovePlanetRequest { planetId = destroyPlanetRequest.planetId },
            object : StreamObserver<PlanetServiceProto.RemovePlanetResponse> by DefaultStreamObserver() {
              override fun onNext(removePlanetResponse: PlanetServiceProto.RemovePlanetResponse) {
                if (removePlanetResponse.result) {
                  scoreStub.addScore(AddScoreRequest {
                    userName = destroyPlanetRequest.userName
                    toAdd = destroyPlanetRequest.weight
                  }, object : StreamObserver<Empty> by DefaultStreamObserver() {})
                  logStub.destroyedPlanet(destroyPlanetRequest, object : StreamObserver<Empty>
                  by DefaultStreamObserver() {})
                  planetStub.generateNewPlanet(Empty.getDefaultInstance(),
                        object : StreamObserver<PlanetProto.Planet> by DefaultStreamObserver() {
                          override fun onNext(planet: PlanetProto.Planet) {
                            logStub.newPlanet(planet, object : StreamObserver<Empty>
                            by DefaultStreamObserver() {})
                            listeners.forEach {
                              it.onNext(Planets {
                                addPlanets(populateWithCoordinates(planet,
                                      destroyPlanetRequest.coordinates.x,
                                      destroyPlanetRequest.coordinates.y))
                              })
                            }
                          }
                        })
                }
              }
            })
    }
  }
}
```

The whole destroy method...

```
override fun destroy(responseObserver: StreamObserver<Planets>): StreamObserver<DestroyPlanetRequest> {
  listeners.add(responseObserver)
  planetStub.getAllPlanets(Empty.getDefaultInstance(),
        object : StreamObserver<Planets> by DefaultStreamObserver() {
          override fun onNext(planets: Planets) {
            responseObserver.onNext(populateWithCoordinnates(planets))
          }
        })
  return object : StreamObserver<DestroyPlanetRequest> by DefaultStreamObserver() {
    override fun onNext(destroyPlanetRequest: DestroyPlanetRequest) {
      planetStub.removePlanet(RemovePlanetRequest { planetId = destroyPlanetRequest.planetId },
            object : StreamObserver<PlanetServiceProto.RemovePlanetResponse> by DefaultStreamObserver() {
              override fun onNext(removePlanetResponse: PlanetServiceProto.RemovePlanetResponse) {
                if (removePlanetResponse.result) {
                  scoreStub.addScore(AddScoreRequest {
                    userName = destroyPlanetRequest.userName
                    toAdd = destroyPlanetRequest.weight
                  }, object : StreamObserver<Empty> by DefaultStreamObserver() {})
                  logStub.destroyedPlanet(destroyPlanetRequest, object : StreamObserver<Empty>
                  by DefaultStreamObserver() {})
                  planetStub.generateNewPlanet(Empty.getDefaultInstance(),
                        object : StreamObserver<PlanetProto.Planet> by DefaultStreamObserver() {
                          override fun onNext(planet: PlanetProto.Planet) {
                            logStub.newPlanet(planet, object : StreamObserver<Empty>
                            by DefaultStreamObserver() {})
                            listeners.forEach {
                              it.onNext(Planets {
                                addPlanets(populateWithCoordinates(planet,
                                      destroyPlanetRequest.coordinates.x,
                                      destroyPlanetRequest.coordinates.y))
                              })
                            }
                          }
                        })
                }
              }
            })
    }
  }
}
```

CALLBACK HELL

Why 

ures?

Destroy method is

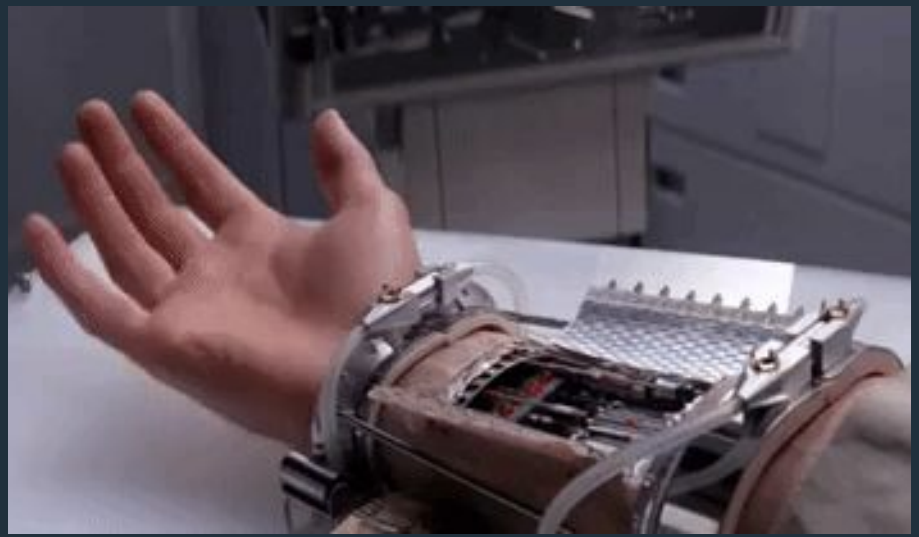
```
override fun destroy(responseObserver: StreamObserver<PlanetProto.Planets>): StreamObserver<DestroyPlanetRequest> {
  listeners.add(responseObserver)
  GlobalScope.launch {
    val allPlanets = planetStub.getAllPlanets(Empty.getDefaultInstance()).await()
    responseObserver.onNext(populateWithCoordinnates(allPlanets))
  }
  return object : StreamObserver<DestroyPlanetRequest> by DefaultStreamObserver() {
    override fun onNext(destroyPlanetRequest: DestroyPlanetRequest) {
      GlobalScope.launch {
        val removePlanet = planetStub.removePlanet(
            RemovePlanetRequest { planetId = destroyPlanetRequest.planetId }).await()
        if (removePlanet.result) {
          scoreStub.addScore(AddScoreRequest {
            userName = destroyPlanetRequest.userName
            toAdd = destroyPlanetRequest.weight
          })
          logStub.destroyedPlanet(destroyPlanetRequest)
          val newPlanet = planetStub.generateNewPlanet(Empty.getDefaultInstance()).await()
          logStub.newPlanet(newPlanet)
          listeners.forEach {
            it.onNext(Planets {
              addPlanets(populateWithCoordinates(newPlanet,
                  destroyPlanetRequest.coordinates.x,
                  destroyPlanetRequest.coordinates.y))
            })
          }
        }
      }
    }
  }
}
```

Destroy method is

```
override fun destroy(responseObserver: StreamObserver<PlanetProto.Planets>): StreamObserver<DestroyPlanetRequest> {
  listeners.add(responseObserver)
  GlobalScope.launch {
    val allPlanets = planetStub.getA
    responseObserver.onNext(popul
  }
  return object : StreamObserver<D
    override fun onNext(destroyPla
      GlobalScope.launch {
        val removePlanet = planet
            RemovePlanetReques
        if (removePlanet.result) {
          scoreStub.addScore(Ad
            userName = destroyP
            toAdd = destroyPlanet
          })
          logStub.destroyedPlane
          val newPlanet = planetS
          logStub.newPlanet(new
          listeners.forEach {
            it.onNext(Planets {
              addPlanets(populat
                  destroyPlanetR
                  destroyPlanetR
            })
          }
        }
      }
    }
  }
}
```

05

gRPC and coroutines II




WHO ARE WE?
KOTLIN DEVS!
WHAT DO WE WANT
CHANNELS IN GRPC!
imgflip.com

We need a new hero!

grpc-kotlin

Destroy method is

```
override suspend fun destroy(requests: ReceiveChannel<DestroyPlanetRequest>): ReceiveChannel<Planets> {
  val channel = Channel<Planets>()
  listeners.add(channel)
  channel.send(populateWithCoordinnates(planetStub.getAllPlanets()))
  for (request in requests) {
    val wasRemoved = planetStub.removePlanet(RemovePlanetRequest { planetId = request.planetId })
    if (wasRemoved.result) {
      scoreStub.addScore(AddScoreRequest {
        userName = request.userName
        toAdd = request.weight
      })
      logStub.destroyedPlanet(request)
      val newPlanet = planetStub.generateNewPlanet()
      logStub.newPlanet(newPlanet)
      listeners.forEach {
        it.send(Planets {
          addPlanets(populateWithCoordinates(newPlanet, request.coordinates.x, request.coordinates.y))
        })
      }
    }
  }
  return channel
}
```

Destroy method is

```
override fun destroy(reque                                              ets> {
  val channel = Channel<Pl
  listeners.add(channel)
  channel.send(populateWit
  for (request in requests) {
    val wasRemoved = pla                                              anetId })
    if (wasRemoved.result)
      scoreStub.addScore
        userName = reque
        toAdd = request.w
      })
      logStub.destroyedPl
      val newPlanet = plan
      logStub.newPlanet(r
      listeners.forEach {
        it.send(Planets {
          addPlanets(pop                                              dinates.y))
        })
      }
    }
  }
  return channel
}
```


Wait a minute...

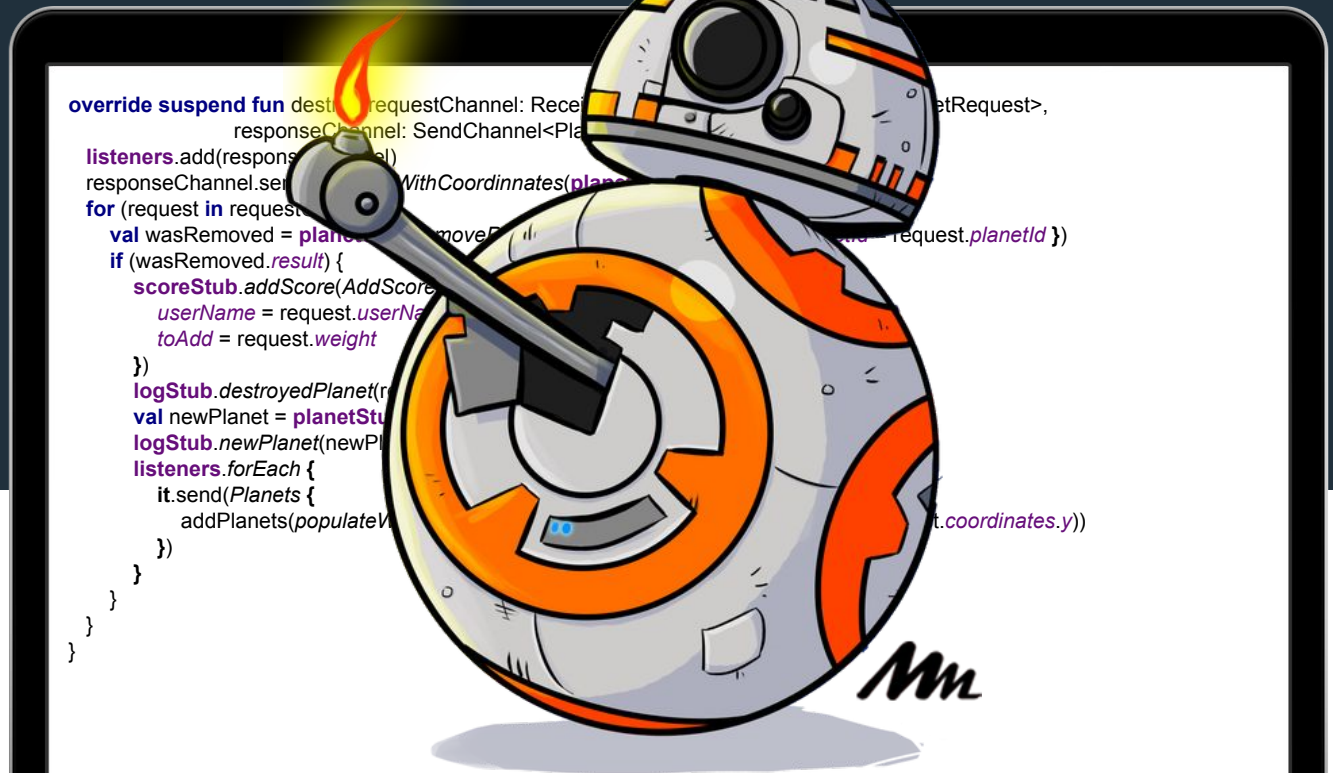
We have another hero!

kroto+

Destroy method is

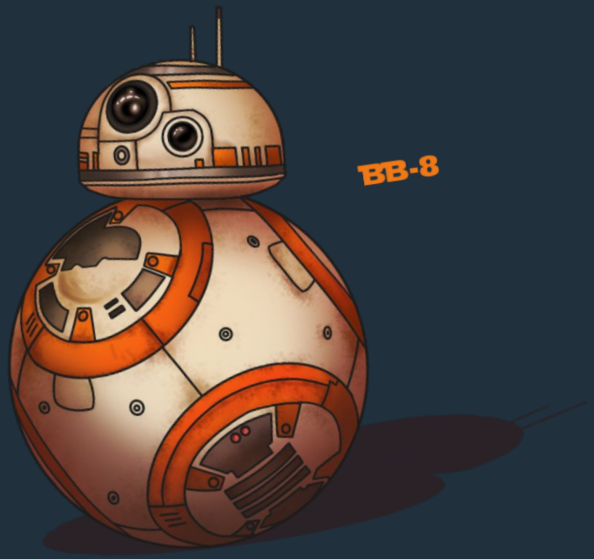
```
override suspend fun destroy(requestChannel: ReceiveChannel<PlanetProto.DestroyPlanetRequest>,
                             responseChannel: SendChannel<PlanetProto.Planets>) {
  listeners.add(responseChannel)
  responseChannel.send(populateWithCoordinnates(planetStub.getAllPlanets()))
  for (request in requestChannel) {
    val wasRemoved = planetStub.removePlanet(RemovePlanetRequest { planetId = request.planetId })
    if (wasRemoved.result) {
      scoreStub.addScore(AddScoreRequest {
        userName = request.userName
        toAdd = request.weight
      })
      logStub.destroyedPlanet(request)
      val newPlanet = planetStub.generateNewPlanet()
      logStub.newPlanet(newPlanet)
      listeners.forEach {
        it.send(Planets {
          addPlanets(populateWithCoordinates(newPlanet, request.coordinates.x, request.coordinates.y))
        })
      }
    }
  }
}
```

Destroy method is



06

More libs for Kotlin



Expectation

Reality

https://github.com/rouzwawi/grpc-kotlin.git

gRPC Kotlin - Coroutine based gRPC for Kotlin

gRPC Kotlin is a protoc plugin for generating native Kotlin bindings using coroutine primitives for gRPC services.

https://github.com/rouzwawi/grpc-kotlin.git

gRPC Kotlin - Coroutine based gRPC for Kotlin

gRPC Kotlin is a protoc plugin for generating native Kotlin bindings using coroutine primitives for gRPC services.

https://github.com/rouzwawi/grpc-kotlin.git

gRPC Kotlin - Coroutine based gRPC for Kotlin

gRPC Kotlin is a protoc plugin for generating native Kotlin bindings using coroutine primitives for gRPC services.

> This project is an early prototype and has not been tested in production. But don't hesitate to try it out and open up issues in the project if you run into any problems. PR's are welcome!

https://github.com/rouzwawi/grpc-kotlin.git

gRPC Kotlin - Coroutine based gRPC for Kotlin

build passing maven-central v0.1.2

gRPC Kotlin is a protoc plugin for generating native Kotlin bindings using coroutine primitives for gRPC services.

https://github.com/rouzwawi/grpc-kotlin.git

gRPC Kotlin - Coroutine based gRPC for Kotlin

gRPC Kotlin is a protoc plugin for generating native Kotlin bindings using coroutine primitives for gRPC services.

https://github.com/rouzwawi/grpc-kotlin.git

gRPC Kotlin - Coroutine based gRPC for Kotlin

build passing maven-central v0.1.2

gRPC Kotlin is a protoc plugin for generating native Kotlin bindings using coroutine primitives for gRPC services.

https://github.com/marcoferrer/kroto-plus.git

Kroto+

Protoc plugin for bringing together Kotlin, Protobuf, Coroutines, and gRPC

A year ago

https://github.com/marcoferrer/kroto-plus.git

gRPC Kotlin Coroutines, Protobuf DSL, Scripting for Protoc

Community Contributions are Welcomed

https://github.com/marcoferrer/kroto-plus.git

KROTO+

gRPC Kotlin Coroutines, Protobuf DSL, Scripting for Protoc

Community Contributions are Welcomed

Useful links

- https://github.com/salesforce/grpc-java-contrib/
- https://github.com/salesforce/reactive-grpc
- https://github.com/cretz/pb-and-k
- https://github.com/leveretka/grpc-death-star
- https://medium.com/@bimeshde/grpc-vs-rest-performance-simplified-fd35d01bbd4

One day we'll have a stable kotlin support

We have a stable kotlin support!

Time to summarize

Takeaways

- Use gRPC for effective communications

Takeaways

- Use gRPC for effective communications
- gRPC + Kotlin can be used today

Takeaways

- Use gRPC for effective communications
- gRPC + Kotlin can be used today
- gRPC + Kotlin make things easier

Takeaways

- Use gRPC for effective communications
- gRPC + Kotlin can be used today
- gRPC + Kotlin make things easier
- Waiting for 1.0 release, flows...

Takeaways

- Use gRPC for effective communications
- gRPC + Kotlin can be used today
- gRPC + Kotlin make things easier
- Waiting for 1.0 release, flows...
- Feel free to contribute :)

Takeaways

- Use gRPC for effective communications
- gRPC + Kotlin can be used today
- gRPC + Kotlin make things easier
- Waiting for 1.0 release, flows...
- Feel free to contribute :)
- Don’t use grpc!

Takeaways

- Use gRPC for effective communications
- gRPC + Kotlin can be used today
- gRPC + Kotlin make things easier
- Waiting for 1.0 release, flows...
- Feel free to contribute :)
- Don’t use grpc! (when don’t need it)

Time to define the
Death Star master

Let’s say “Thank you”

Without them I wouldn't have done it!

Ray Tsang

Rouzbeh Delavari

Marco Ferrer

Without them I wouldn't have done it!

Alex Borysov

Andrii Petryk

George Lucas

THANK YOU
AND REMEMBER TO VOTE

Marharyta Nedzelska @jMargaritaN
#KotlinConf