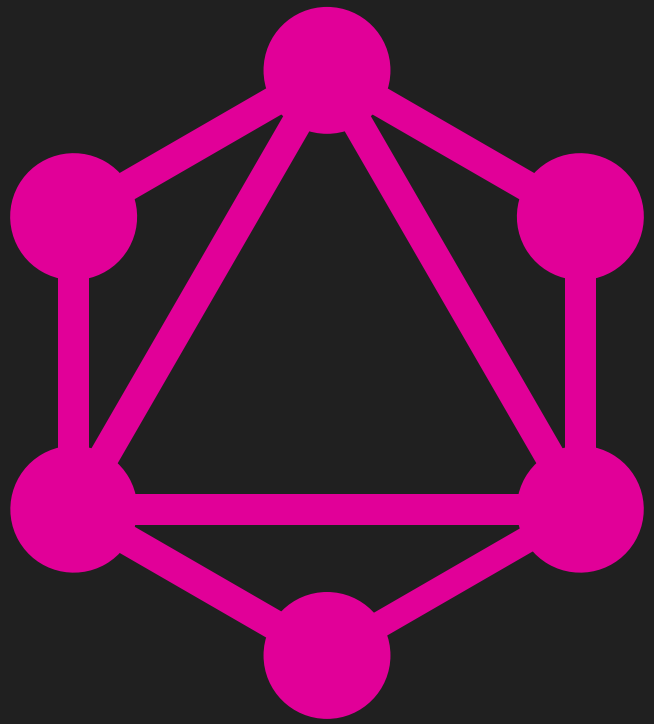


# GraphQL: IoC makes its way to HTTP

# Hello

@dustinwhitney

---

ny-scala / Northeast Scala Symposium

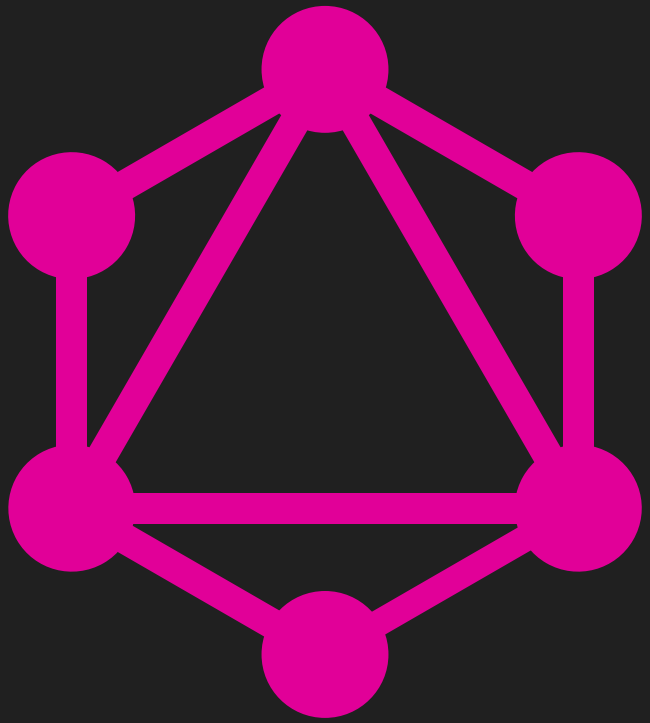
---

Pellucid Analytics / Project September

Freek (@mandubian)

gl-react / gl-react-native (@greweb, @mrspeaker)

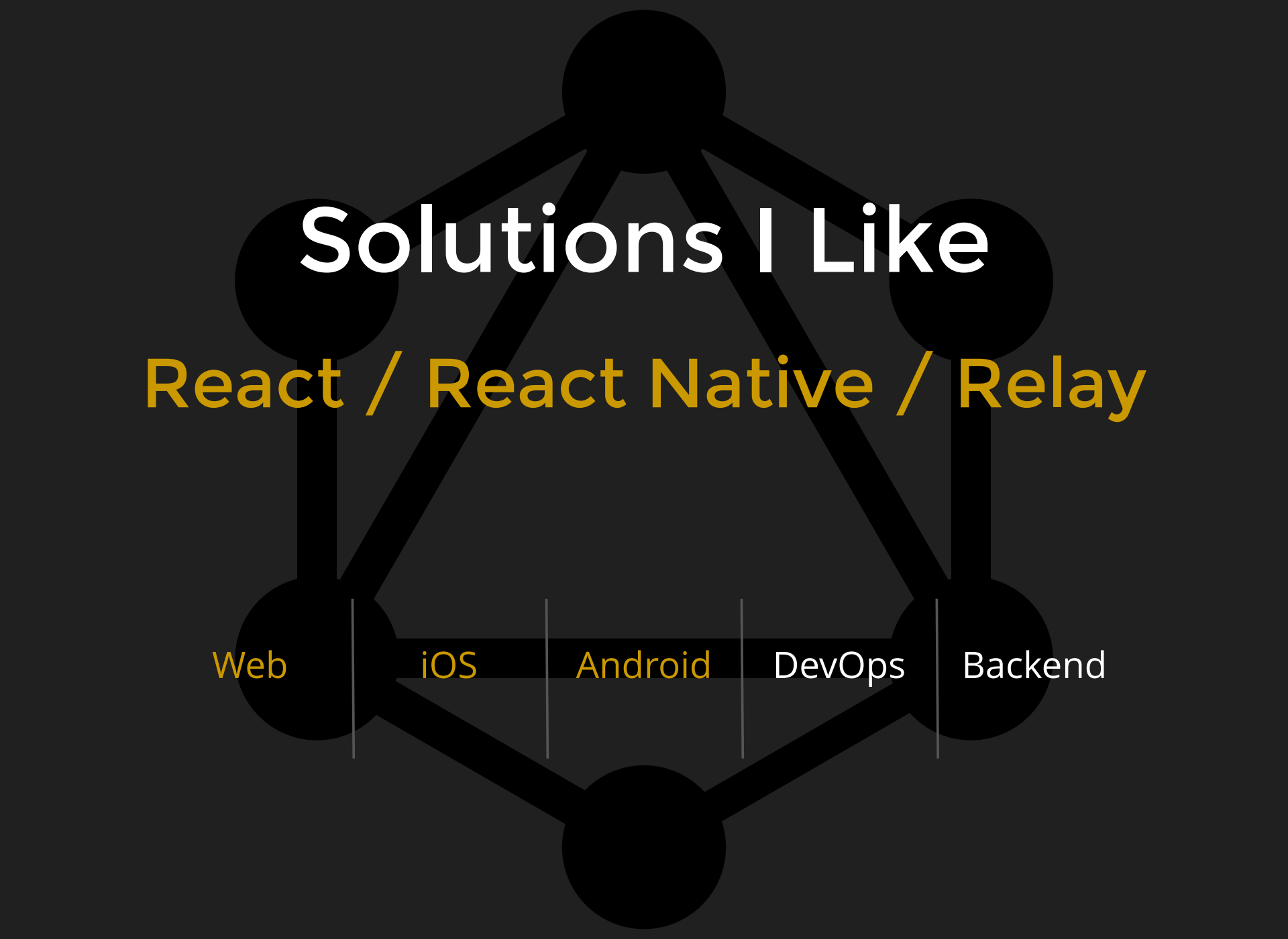
---

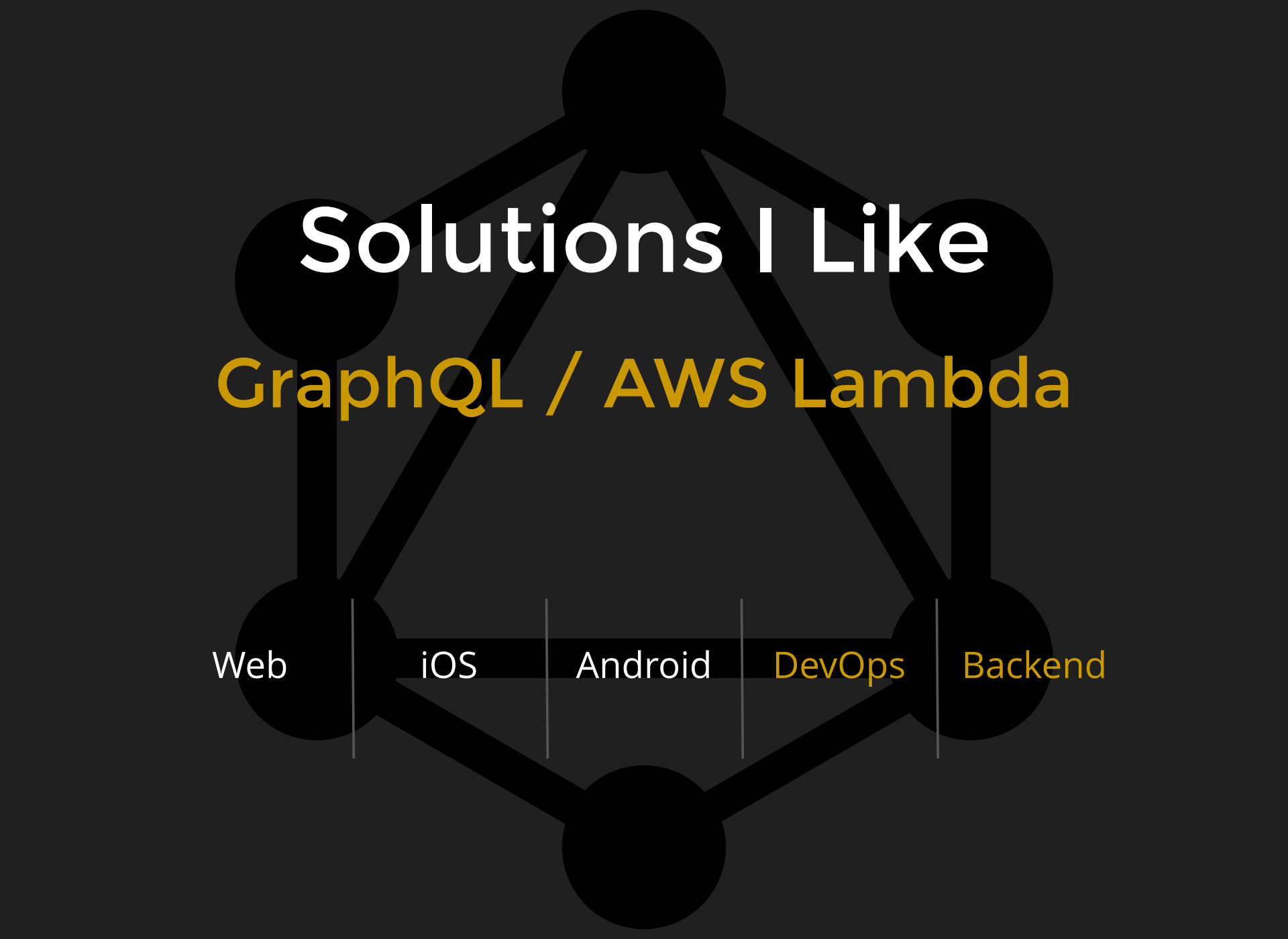


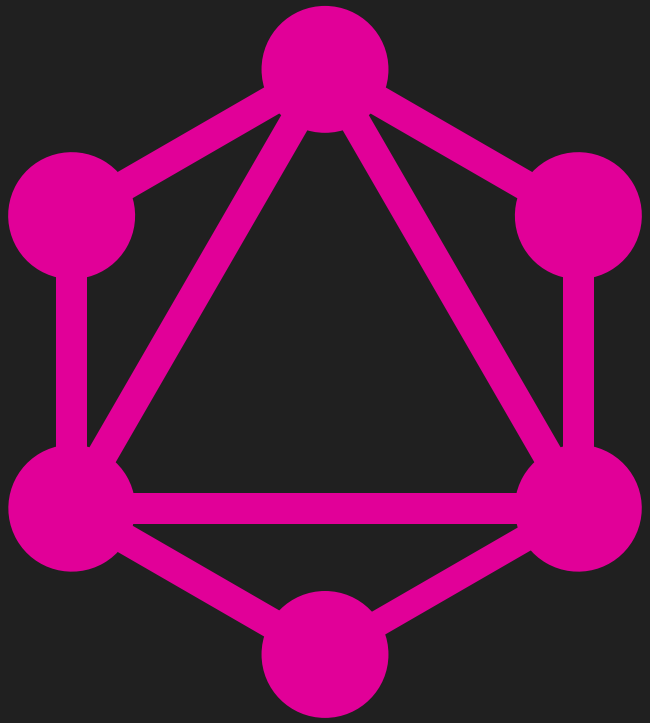
# Background

# Solving a Startups Biggest Problem

At a startup, people are your biggest problem


Solutions I Like
React / React Native / Relay
Web
iOS
Android
DevOps
Backend


Solutions I Like
GraphQL / AWS Lambda
Web
iOS
Android
DevOps
Backend



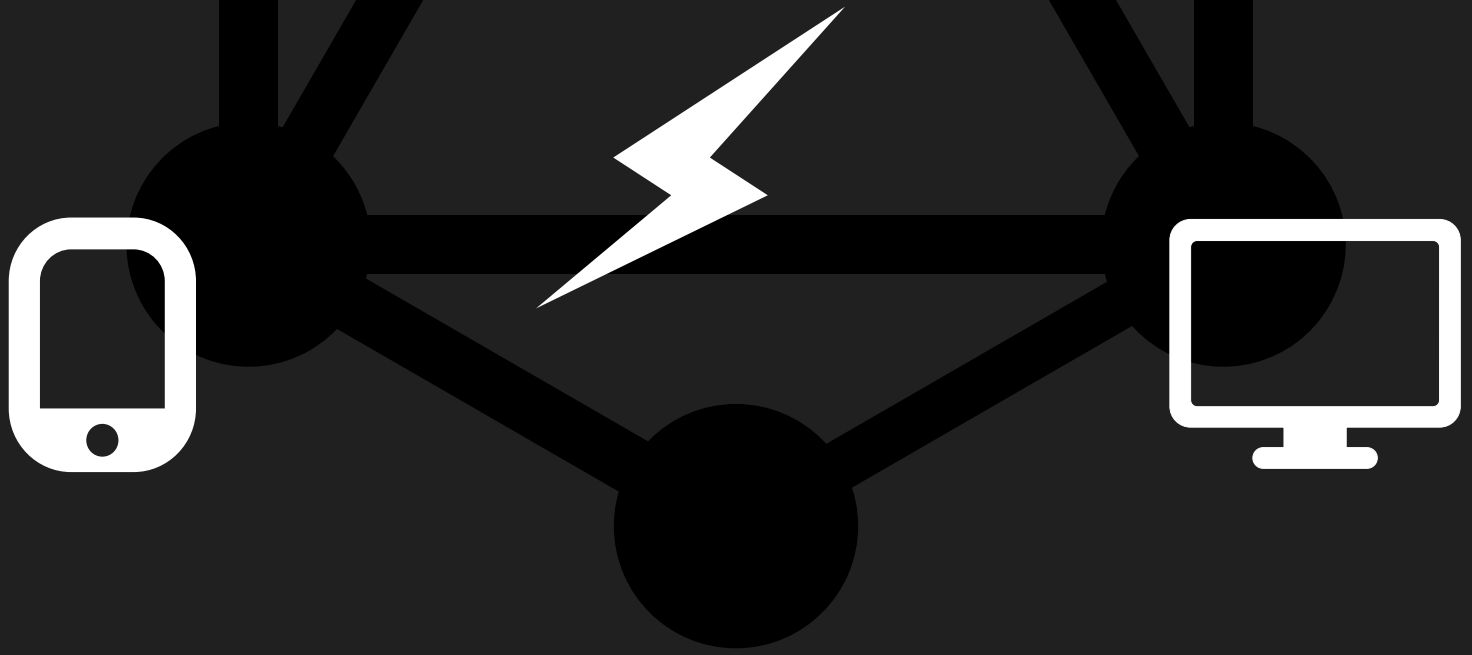
# GraphQL: Inversion of Control

# Inversion of Control?

http://example.com/api/everything/odersky

---

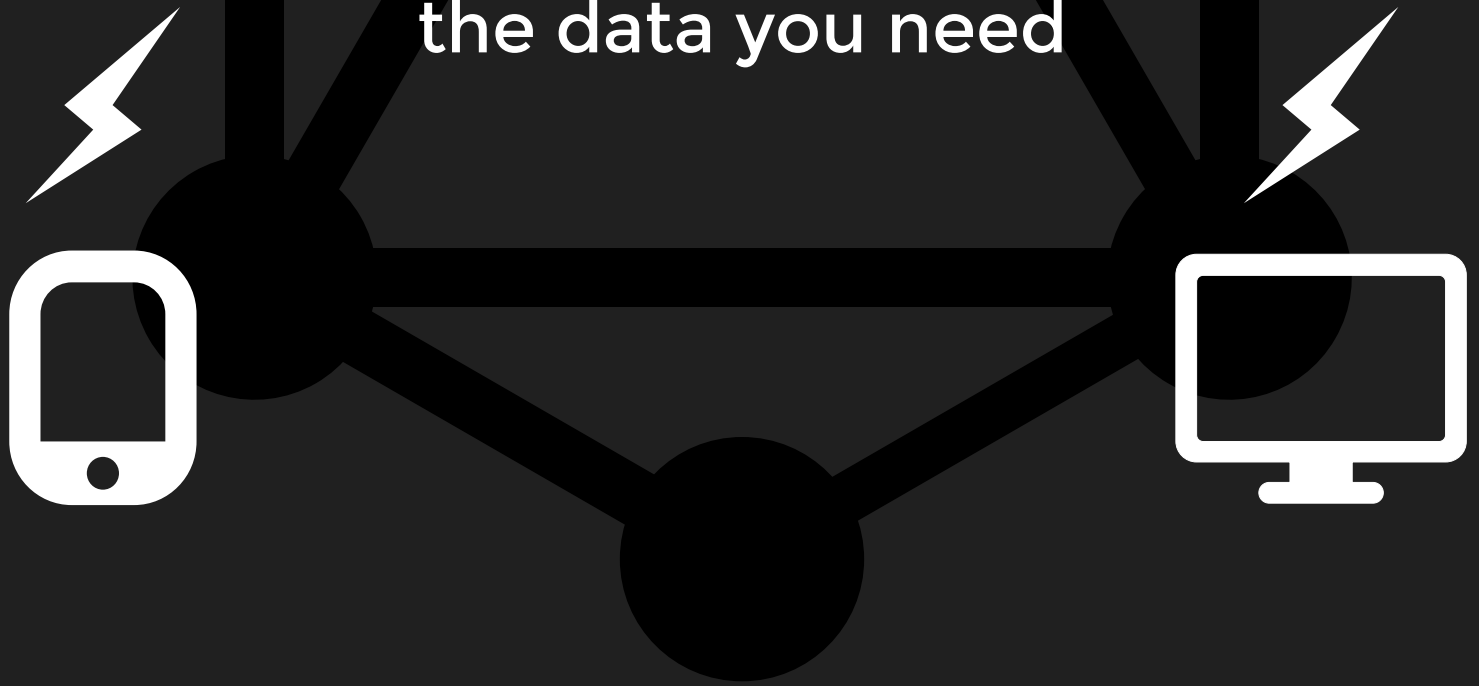
http://example.com/api/feed/odersky
http://example.com/api/avatar/odersky
http://example.com/api/comments/odersky



# Inversion of Control?

GraphQL is a query language for your API

Query for exactly - and only - the data you need



# What is GraphQL?

You construct a schema and query much like SQL

```
# SQL

CREATE TABLE User (
  username VARCHAR(30) PRIMARY KEY
)
```

```
# SQL                  # username |
select * from User     #----------|
                       # odersky  |
                       # d6       |
                       # rit      |
```

```
// GraphQL

type User {
  username: String!
}

// Arguments
type Root {
  users: [User]
}
```

```
// GraphQL Query
{
  users{
    username
  }
}

// Result
{
  users:[
    {username:"odersky"},
    {username:"d6"},
    {username:"rit"}
  ]
}
```

# What is GraphQL?

## GraphQL Schemas and Types

```
// Object types and fields
type User {
  username: String! // non-null
  comments: [String] // List
}

// Scalar types: Int, Float, String, Boolea

// Arguments
type Root {
  user(username: String!): User
}

// Query and Mutation
schema {
  query: Root
  mutation: Mutation
}
```

```
// Enumeration types
enum UserFlag {
  STANDARD
  ADMIN
}

// Interfaces
interface Mammal {
  hasHair: Boolean!
}

type Dog implements Mammal{
  hasHair: Boolean!
  barks: Boolean!
}

// Union Types
union Animal = Dog | Cat
```

# How Do I do it in Scala?

## Build a schema from your object graph

```
case class User(
  username: String
)

case class Avatar(
  url: String
, mobileOptimizedURL: String
)

trait AvatarRepository{
  def getAvatar(
    username: String
  ): Option[Avatar]
}

trait UserRepository{
  def getUser(
    username: String
  ): Option[User]
```

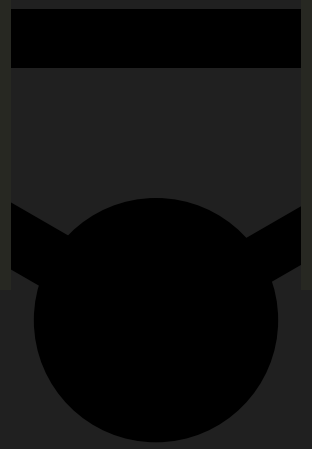
```
type User {
  username: String!
  avatar: Avatar
}

type Avatar {
  url: String!
  mobileOptimizedURL: String!
}

type Root {
  user(username: String!): User
}
```

# How does it work?

## Goal Query

```
{
  user(username:"odersky"){
    username
    avatar{
      mobileOptimizedURL
    }
  }
}
```

```
{
  "data" : {
    "user" : {
      "username" : "odersky",
      "avatar" : {
        "mobileOptimizedURL" : "...
      }
    }
  }
}
```



# How does it work?

## Use existing model and services

```
case class User(
  username: String
)

case class Avatar(
  url: String
, mobileOptimizedURL: String
)

trait AvatarRepository{
  def getAvatar(
    username: String
  ): Option[Avatar]
}

trait UserRepository{
  def getUser(
    username: String
  ): Option[User]
```

```
class Cake
  extends AvatarRepository
  with UserRepository{

  val users = Map("odersky" -> User("odersky"))

  val avatars = Map(
    "odersky" -> Avatar(
                   url = "..."
                 , mobileOptimizedURL = "..."
                 )
  )

  def getAvatar(username: String): Option[Avatar]
    avatars.get(username)

  def getUser(username: String): Option[User] =
    users.get(username)
}
```

# How does it work?

Build a schema using Sangria by
@easyangel (http://sangria-graphql.org/)

```
/*case class User(
  username: String
)*/

case class Avatar(
  url: String
, mobileOptimizedURL: String
)

/*trait AvatarRepository{
  def getAvatar(
    username: String
  ): Option[Avatar]
}

trait UserRepository{
  def getUser(
    username: String
  ): Option[User]
```

```
val avatarObj: ObjectType[Cake, Avatar] = ObjectTyp
  "Avatar"
, fields[Cake, Avatar](
    Field("url", StringType, resolve = _.value.url)
  , Field("mobileOptimizedURL"
    , StringType
    , resolve = _.value.mobileOptimizedURL
  )
)
```

```
// for less boilerplate use derive with macros!
val avatarObj = deriveObjectType[Cake, Avatar]()
```

```
// Resulting GraphQL
type Avatar {
  url: String!
  mobileOptimizedURL: String!
}
```

# How does it work?

## More about Fields

```
case class User(
  username: String
)

/*case class Avatar(
  url: String
, mobileOptimizedURL: String
)*/

/*trait AvatarRepository{
  def getAvatar(
    username: String
  ): Option[Avatar]
}

trait UserRepository{
  def getUser(
    username: String
  ): Option[User]
```

```
val userObj = deriveObjectType[Cake, User](
  AddFields(
    Field("avatar"
      , OptionType(avatarObj)
      , resolve = { context: Context[Cake, User] =>
          context.ctx.getAvatar(ctx.value.username)
    })))
```

```
case class Context[Ctx, Val](
  value: Val,
  ctx: Ctx,
  args: Args,
  ...)
```

```
type User {
  username: String!
  avatar: Avatar
}
```

# How does it work?

## Finishing up the schema

```
val queryObj = ObjectType(
  "Root"
 , fields[Cake, Unit](
    Field(
      "user"
    , OptionType(userObj)
    , arguments =
        Argument("username", StringType) :: Ni
    , resolve = { ctx =>
        ctx.ctx.getUser((ctx arg "username"))
      }
    )))

val schema = Schema(queryObj)
println(SchemaRenderer.renderSchema(schema))
```

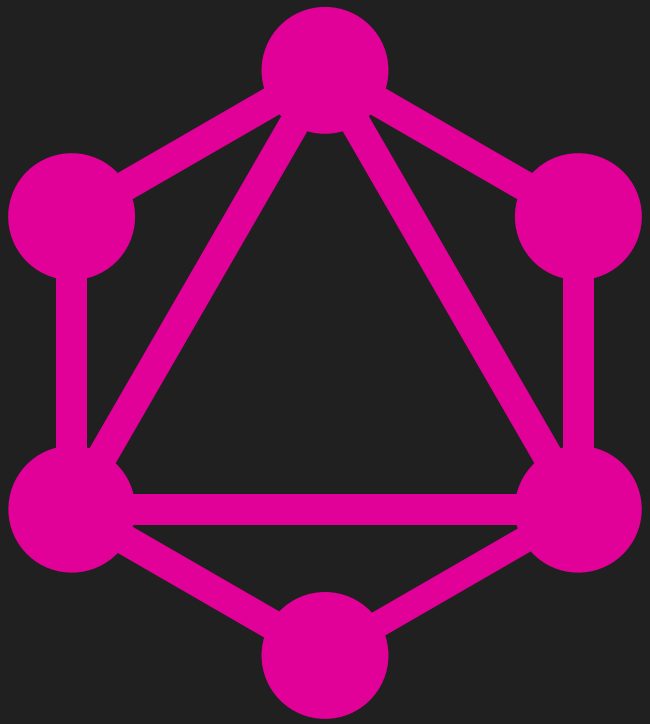
```
schema {
  query: Root
}

type Avatar {
  url: String!
  mobileOptimizedURL: String!
}

type Root {
  user(username: String!): User
}

type User {
  username: String!
  avatar: Avatar
}
```

# How does it work?

## Executing our query

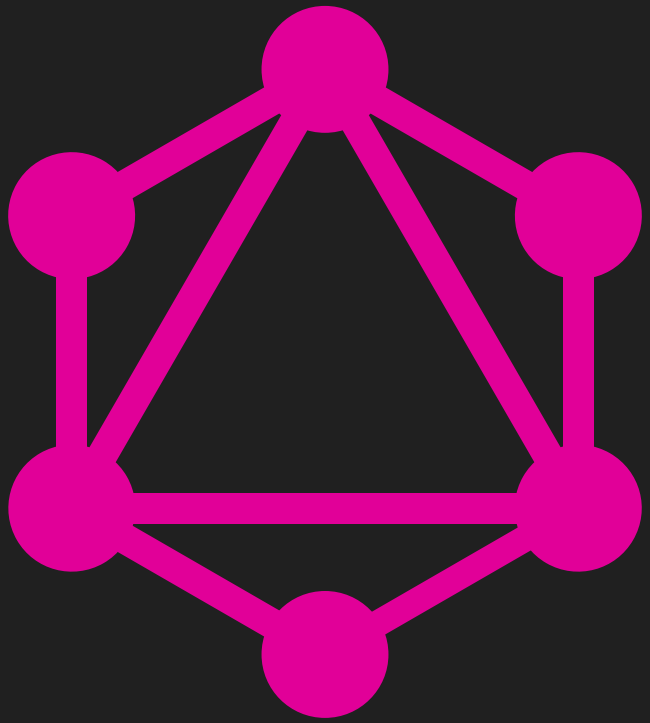
```
val query =
"""{
  user(username:"odersky"){
    username
    avatar{
      mobileOptimizedURL
    }
  }
}"""

val Success(queryAst) =
    QueryParser.parse(query)

import sangria.marshalling.circe._
println(Await.result(

    Executor.execute(
        schema
    , queryAst
    , new Cake

  ), 5 seconds))
```

```
// Result
{
  "data" : {
    "user" : {
      "username" : "odersky",
      "avatar" : {
        "mobileOptimizedURL" : "..."
      }
    }
  }
}
```



https://github.com/dwhitney/sbtb

Demo: Graphiql



Questions?