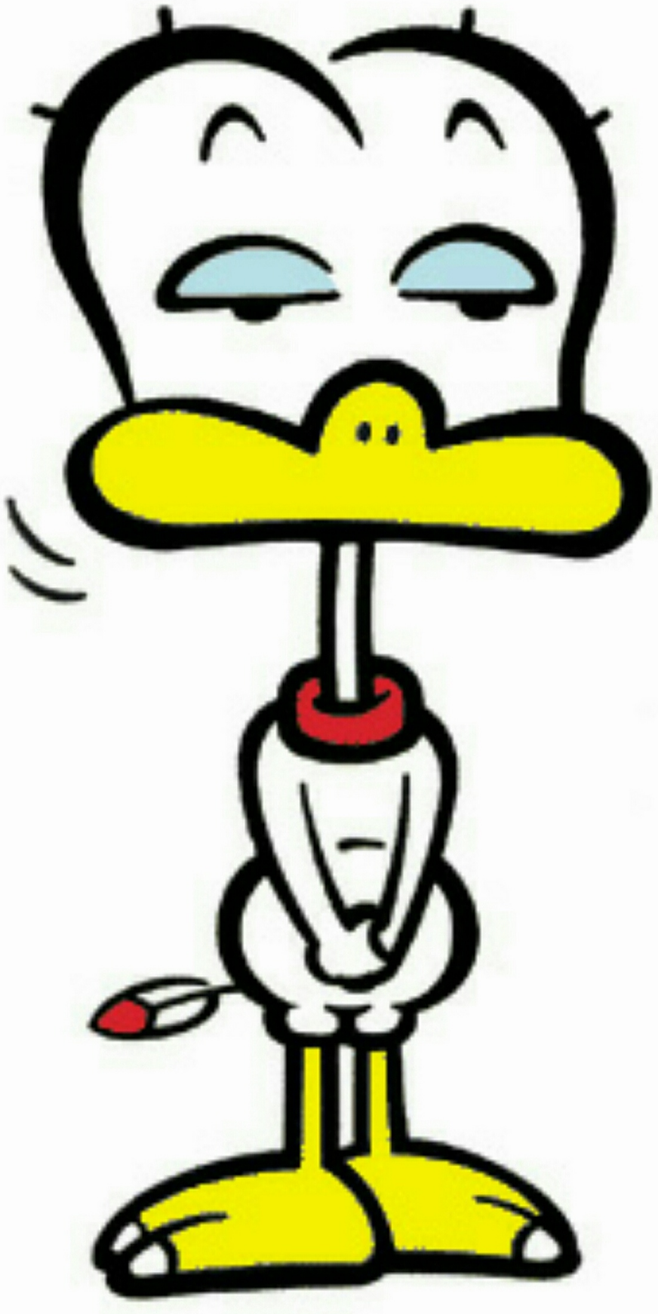
DESIGNING A FUNCTIONAL GRAPHQL LIBRARY

FUNCTIONAL SCALA 2019

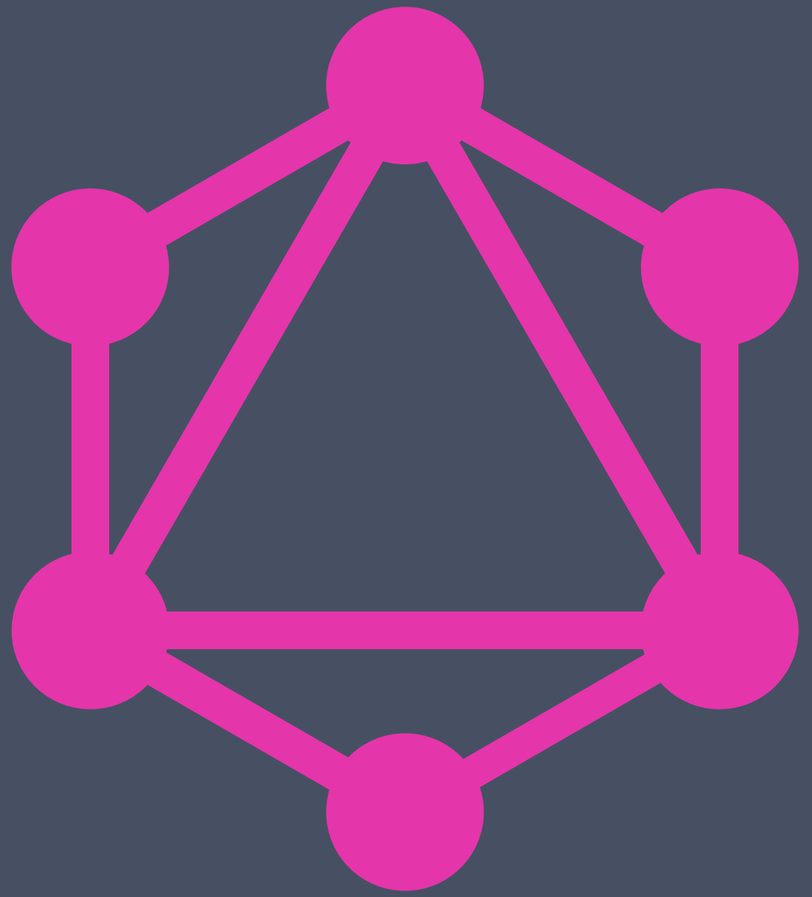


WHO AM I?

- PIERRE RICADAT AKA @GHOSTDOGPR
- 🇫🇷 EXILED IN 🇰🇷
- CONTRIBUTOR TO ZIO
- CREATOR OF CALIBAN

bpd HOLDINGS

GRAPHQL ?



GRAPHQL IN A NUTSHELL

- QUERY LANGUAGE FOR APIS
- SERVER EXPOSES A TYPED SCHEMA
- CLIENT ASKS FOR WHAT THEY WANT
  - QUERIES
  - MUTATIONS
  - SUBSCRIPTIONS

WHY?

WHY CREATE A GRAPHQL LIBRARY?

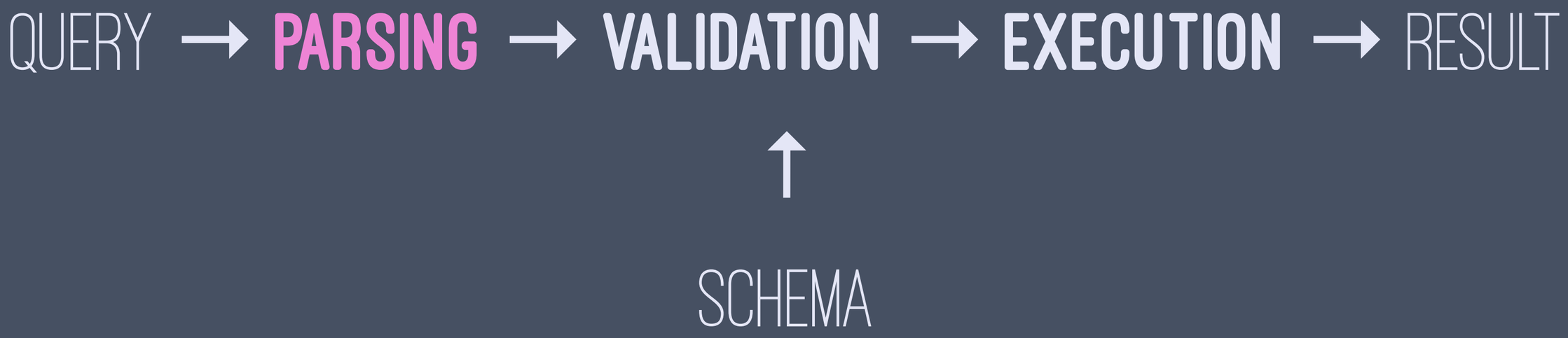
- SANGRIA IS GREAT BUT...
  - LOTS OF BOILERPLATE
  - FUTURE-BASED
  - SCHEMA AND RESOLVER TIED TOGETHER
- INTERESTING APPROACH BY MORPHEUS (HASKELL)
- IT'S FUN!

GOALS

- MINIMAL BOILERPLATE
- PURELY FUNCTIONAL
  - STRONGLY TYPED
  - EXPLICIT ERRORS
  - ZIO
- SCHEMA / RESOLVER SEPARATION

PLAN OF ACTION

QUERY PARSING



QUERY PARSING

GRAPHQL SPEC: HTTPS://GRAPHQL.GITHUB.IO/GRAPHQL-SPEC/

2.8 Fragments

FragmentSpread :
    `...` *FragmentName* *Directives*$_{opt}$

FragmentDefinition :
    `fragment` *FragmentName* *TypeCondition* *Directives*$_{opt}$ *SelectionSet*

FragmentName :
    Name but not **`on`**

QUERY PARSING

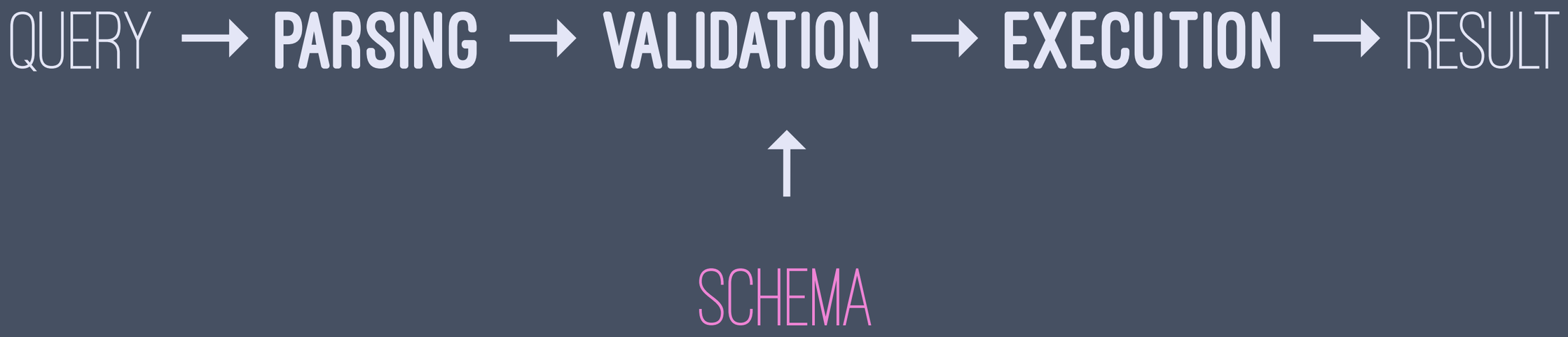
FASTPARSE: FAST. EASY TO USE. GOOD DOCUMENTATION

```
def name[_: P]: P[String] = P(CharIn("_A-Za-z") ~~ CharIn("_0-9A-Za-z").repX).!

def fragmentName[_: P]: P[String] = P(name).filter(_ != "on")

def fragmentSpread[_: P]: P[FragmentSpread] =
  P("..." ~ fragmentName ~ directives).map {
    case (name, dirs) => FragmentSpread(name, dirs)
  }
```

SCHEMA



SCHEMA

IDEA FROM MORPHEUS:

DERIVE GRAPHQL SCHEMA FROM BASIC TYPES

SIMPLE API

```
type Queries {
  pug: Pug!
}

case class Queries(pug: Pug)
```

OBJECTS

```
type Pug {
  name: String!
  nicknames: [String!]!
  favoriteFood: String
}
```

```
case class Pug(
  name: String,
  nicknames: List[String],
  favoriteFood: Option[String])
```

ENUMS

```
enum Color {
  FAWN
  BLACK
  OTHER
}

sealed trait Color
case object FAWN  extends Color
case object BLACK extends Color
case object OTHER extends Color
```

ARGUMENTS

```
type Queries {
  pug(name: String!): Pug
}

case class PugName(name: String)
case class Queries(pug: PugName => Option[Pug])
```

EFFECTS

```
type Queries {
  pug(name: String!): Pug
}
```

```
case class PugName(name: String)
case class Queries(pug: PugName => Task[Pug])
```

RESOLVER

SIMPLE VALUE

```
case class PugName(name: String)
case class Queries(
  pug: PugName => Task[Pug]
  pugs: Task[List[Pug]]
)

val resolver = Queries(
  args => PugService.findPug(args.name),
  PugService.getAllPugs
)
```

SCHEMA

```
trait Schema[-R, T] {

  // describe the type T
  def toType(isInput: Boolean = false): __Type

  // describe how to resolve a value of type T
  def resolve(value: T): Step[R]
}
```

WHAT'S A STEP

- PURE VALUE (LEAF)
- LIST
- OBJECT
- EFFECT
- STREAM

SCHEMA FOR STRING

```
implicit val stringSchema = new Schema[Any, String] {

  def toType(isInput: Boolean = false): __Type =
    __Type(__TypeKind.SCALAR, Some("String"))

  def resolve(value: String): Step[R] =
    PureStep(StringValue(value))
}
```

SCHEMA FOR CASE CLASSES AND SEALED TRAITS?

MAGNOLIA

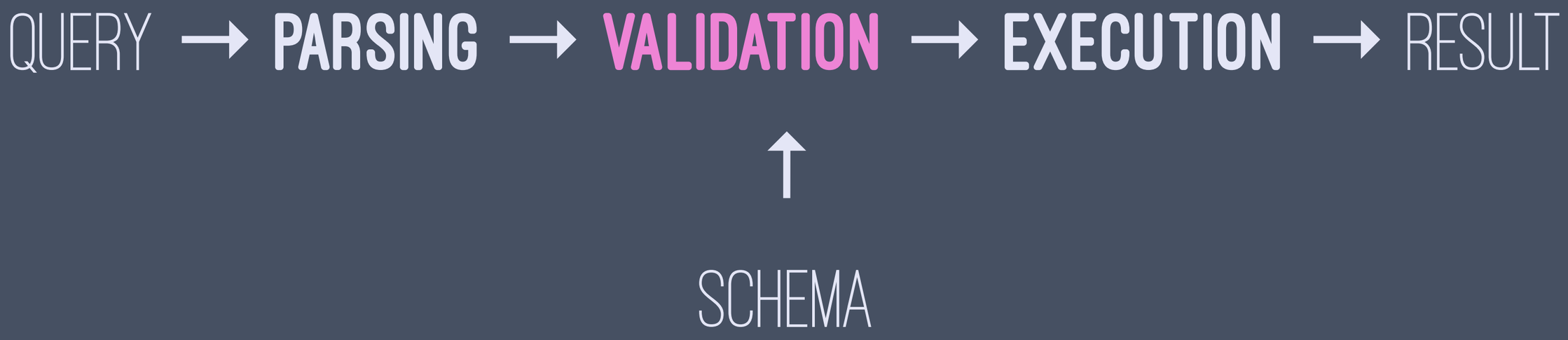
- 👍
  - EASE OF USE
  - COMPILE TIME
  - DOCUMENTATION, EXAMPLES
- 🤔
  - ERROR MESSAGES (GETTING BETTER!)

MAGNOLIA DERIVATION

```
def combine[T](
  caseClass: CaseClass[Typeclass, T]): Typeclass[T]

def dispatch[T](
  sealedTrait: SealedTrait[Typeclass, T]): Typeclass[T]
```

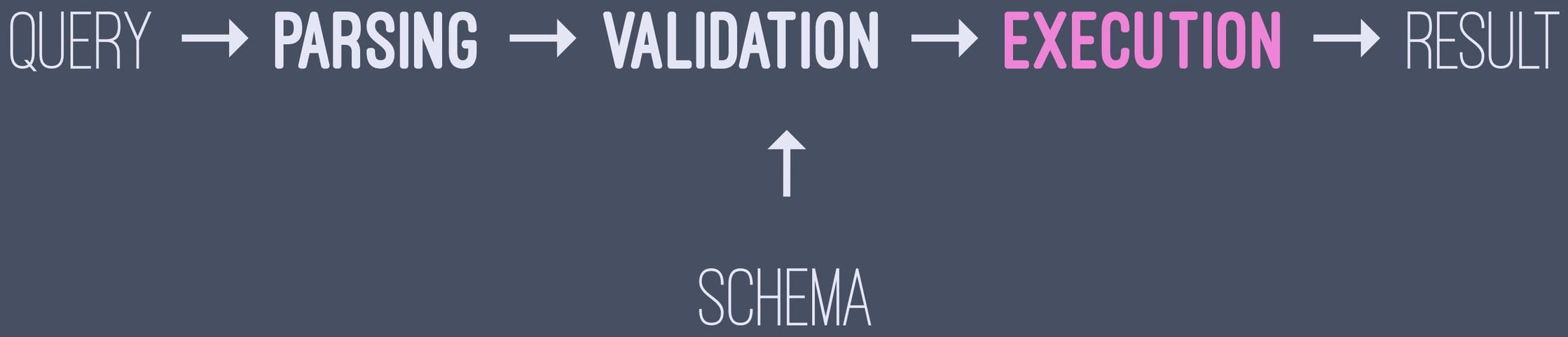
VALIDATION



VALIDATION

BACK TO THE SPECS 😒

EXECUTION



EXECUTION

1. SCHEMA + RESOLVER => EXECUTION PLAN (TREE OF STEPS)
2. FILTER PLAN TO INCLUDE ONLY REQUESTED FIELDS
3. REDUCE PLAN TO PURE VALUES AND EFFECTS
4. RUN EFFECTS

N+1 PROBLEM

```
{
  orders {          # fetches orders (1 query)
    id
    customer {  # fetches customer (n queries)
      name
    }
  }
}
```

QUERY OPTIMIZATION

NAIVE ZIO[R, E, A] VERSION

```
val getAllUserIds: ZIO[Any, Nothing, List[Int]] = ???
def getUserNameById(id: Int): ZIO[Any, Nothing, String] = ???

for {
  userIds   <- getAllUserIds
  userNames <- ZIO.foreachPar(userIds)(getUserNameById)
} yield userNames
```

QUERY OPTIMIZATION

MEET ZQUERY[R, E, A]

```
val getAllUserIds: ZQuery[Any, Nothing, List[Int]] = ???
def getUserNameById(id: Int): ZQuery[Any, Nothing, String] = ???

for {
  userIds   <- getAllUserIds
  userNames <- ZQuery.foreachPar(userIds)(getUserNameById)
} yield userNames
```

ZQUERY BENEFITS

- PARALLELIZE QUERIES
- CACHE IDENTICAL QUERIES (DEDUPLICATION)
- BATCH QUERIES IF BATCHING FUNCTION PROVIDED

ZQUERY

- COURTESY OF @ADAMGFRASER
- BASED ON PAPER ON HAXL
  - "THERE IS NO FORK: AN ABSTRACTION FOR EFFICIENT, CONCURRENT, AND CONCISE DATA ACCESS"
- WILL BE EXTRACTED TO ITS OWN LIBRARY AT SOME POINT

INTROSPECTION

DOGFOODING

```
case class __Introspection(
  __schema: __Schema,
  __type: __TypeArgs => __Type)
```

USAGE

1. DEFINE YOUR QUERIES / MUTATIONS / SUBSCRIPTIONS
2. PROVIDE A SCHEMA FOR CUSTOM TYPES
3. PROVIDE A RESOLVER
4. PROFIT

1. DEFINE YOUR QUERIES

```
case class Pug(name: String, nicknames: List[String], favoriteFood: Option[String])
case class PugName(name: NonEmptyString)
case class Queries(pugs: Task[List[Pug]], pug: PugName => Task[Pug])

type Pug {
  name: String!
  nicknames: [String!]!
  favoriteFood: String
}

type Queries {
  pugs: [Pug!]
  pug(name: String!): Pug
}
```

2. PROVIDE A SCHEMA FOR CUSTOM TYPES

```
implicit val nesSchema: Schema[NonEmptyString] =
  Schema.stringSchema.contramap(_.value)
```

3. PROVIDE A RESOLVER

```
def getPugs: Task[List[Pug]] = ???
def getPug(name: String): Task[Pug] = ???

val queries = Queries(
  getPugs,
  args => getPug(args.name))

val resolver = RootResolver(queries)
```

4. PROFIT

```
val interpreter = graphQL(resolver)

for {
  result <- interpreter.execute(query)
  _      <- zio.console.putStrLn(result.toString)
} yield ()
```

HTTP4S MODULE

```
val route: HttpRoutes[RIO[R, *]] =
  Http4sAdapter.makeRestService(interpreter)

val wsRoute: HttpRoutes[RIO[R, *]] =
  Http4sAdapter.makeWebSocketService(interpreter)
```

SCALAJS

FEW AND MODERN DEPENDENCIES

SCALAJS SUPPORT FOR FREE

CATS COMPATIBILITY

CALIBAN-CATS MODULE

```
def executeAsync[F[_]: Async](...)(
  implicit runtime: Runtime[Any]): F[GraphQLResponse[E]]

def makeRestServiceF[F[_]: Effect, Q, M, S, E](...)(
  implicit runtime: Runtime[Any]): HttpRoutes[F]
```

PERFORMANCE

CALIBAN IS FAST

CHECK BENCHMARKS MODULE ON GITHUB

FUTURE

- QUERY ANALYSIS
- MIDDLEWARE (QUERY-LEVEL, FIELD-LEVEL)
- CODE GENERATION
- OTHER HTTP ADAPTERS (AKKA HTTP) <- HELP NEEDED
- CLIENT SUPPORT

WANNA KNOW MORE?

- WEBSITE: https://ghostdogpr.github.io/caliban/
- ZIO DISCORD: #caliban

THANKS!

QUESTIONS?