Introduction to Kyo
Adam Hearn

What is Kyo?

- Kyo is a powerful toolkit for developing with Scala
- Built from a series of standalone modules:
  - `kyo-data`: Low allocation, performant structures
  - `kyo-prelude`: Side-effect free **Algebraic effects**
  - `kyo-core`: Effects for IO, Async, & Concurrency
  - `kyo-scheduler`: high performance adaptive scheduler
    - `kyo-scheduler-zio`: run your ZIO App!

What are Algebraic Effects?

What are ~~Algebraic~~ Effects?

- Effects
  - Descriptions of what you want
  - Produce what you want *when run*
  - Programs as values!
- Effects are backed by suspension
  - Suspension defers a computation until later
  - Separation of execution from definition:
    - Flexibility in execution (Retry, Delay, Interrupt)
    - Delayed implementation (Clock.live vs Clock.test)

What are Algebraic Effects?

- Extensible & Composable Effects!
  - Fine-grained control over effect handling
  - Trivial combination of various abilities
  - Separation of effect declaration and implementation
  - User defined effects!
- Handlers: Define how effects are interpreted

Why use Algebraic Effects?

Why use Kyo?

- Includes flexible algebraic effects in **Scala**
- Designed for simplicity and performance
- Core effect handling is not restricted to included effects
  - User defined effects are

Kyo Syntax

```
val _: String < IO = IO("Hello scala.io!")
```

- Infix 'Pending' Type: Result < Effects
    - String < IO
- Effects are represented as unordered set:
    - File < (IO & Resource)'

8

IO: Side-Effect Suspension

Kyo enables labeling side effects via `IO`:

```scala
object DB:
  def run(query: SQL[Result]): Result < IO = ???

object MyApp:
  val _: Chunk[Person] < Any = IO.run(DB.run(sql"select * from person"))
```

- `IO` must be handled individually (`IO.run`)
- The above expression is not fully evaluated and may be pending further suspensions.

Abort: Short Circuit

```scala
object User:
  case class User(email: String)
  enum UserError:
    case InvalidEmail
    case AlreadyExists

  def from(email: String): User < Abort[UserError] =
    if !email.contains('@') then Abort.fail(Invalid)
    else User(names.head, names.tail)

val x: Unit < IO =
  Abort.run(from("adam@veak.co")).map:
    case Result.Success(user) => Console.println(s"Success! $user")
    case Result.Fail(Invalid)                    => Console.println(s"Bad email!")
```

Env: Dependency Injection

```
trait DB:
  def apply(id: Int): String < IO

val program: Record < (Env[DB] & IO) =
  for
    config <- Env.get[DB]
    result <- IO(s"Connecting to ${config.url}")
  yield result

// Usage
val config = Config("http://example.com")
val result = program.provideEnv(config).eval
```

Kyo: Effect Widening

```scala
val a: Int < IO = IO(42)
val b: Int < (IO & Abort[Exception]) = a
val c: Int < (IO & Async & Abort[Exception]) = b
```

- Computations can be widened to include more effects
- Allows for flexible and composable code
- Plain values can be widened to Kyo computation
  - Values aren't suspended & don't allocate [1]

[1] Primitives widened to Kyo will box as Scala 3 does not support proper specialization

Kyo: Flattening

- Effects can be easily combined using `map` and `flatMap`
- Resulting type includes all unique pending effects

Kyo: Effect Handling

```
val a: Int < Abort[Exception] = 42
val b: Result[Exception, Int] < Any = Abort.run(a)
val c: Result[Exception, Int] = b.eval
```

- Effects are handled explicitly
- Order of handling can affect the result type and value

KyoApp

Direct Syntax in Kyo

```scala
val a: String < (Abort[Exception] & IO) =
    defer {
        val b: String = await(IO("hello"))
        val c: String = await(Abort.get(Right("world")))
        b + " " + c
    }
```

- defer and await provide a more intuitive syntax

Conclusion

- Kyo provides a powerful yet simple way to work with algebraic effects
- Offers composability, type safety, and performance
- Enables cleaner, more modular functional programming in Scala

Questions?

- !all effects suspend!
- Suspension + Handling
- ZIO each map/flatMap introduces suspension/resumption
- Kyo continues execution until an effect needs to be handled
  https://koka-lang.github.io/koka/doc/index.html

Kyo supports a wide set of audience
- Library Authors
- Novices

Wrap up:

- Effects
- Kyo:
  - map vs flatMap
  - widening
  - IO/Abort/Async/Env/
  - Running your App

TODO: Kyo Data
kyo-scheduler