ALEX BERGERON
PERISCOPE DATA (WE'RE HIRING)

EFFECTS TYPE IN SCALA

WHAT I'M NOT TALKING ABOUT

- Which library is "the best"
- Performance
- Codes of conduct
- Things people say on Twitter

WHY USE AN EFFECT TYPE

- Allows engineer a greater control over what effects are happening
- Effects can compose together
- Easier to reason about than threads
- Easy to run asynchronously, avoiding to block threads
- Allows to write stack safe recursive functions through trampolining

SEMANTICS

- (Typically) running and memoized computation
- Futures composes - you can create a new `Future[B]` through an existing `Future[A]` and a `A => B / A => Future[B]`
- Meant to be used for asynchronous, non-blocking computations, allowing you to keep your thread pools small
- You can await on the `Future` and block your current thread - but why would you reasonably do that?

IMPLEMENTATIONS

- Scala: Backed by `ExecutionContext`, non cancellable. Supported by many libraries. Somewhat of a default asynchronous default type in Scala
- Twitter: Backed by Finagle, cancellable. Mostly supported in Twitter's stack.

WHY FUTURES ARE NOT AN EFFECT TYPE

- Represents a computation that is already running, not a computation that can be executed
- Breaks referential transparency - replacing a future instance with the code it executes might yield a different result.
- Couples a computation, a result, and error handling
- Since there's no way of knowing whether the last transformation was effectful or not, every invocation of `map`/`foreach` creates a new computation to be scheduled

THE END OF THE WORLD

- Ultimately, you do want your effects to be evaluated, but at a time where they are absolutely necessary, and in an order that could be optimized, compared to how they were written
- This should happen as late as necessary, by calling the appropriate unsafe method
  - When serving a request on a web server?
  - When running a database transaction?
  - When running your whole application?
- Ultimately, engineers should decide when effects needs to be resolved, and how they are executed

SCALAZ-7 IO

- Implements the IO Monad - wraps all effectful computations in an `IO[A]` which only gets executed at the end of the world
- Gets evaluated in a trampoline to ensure stack safety
- Bound to be evaluated on the current thread - hence you should only call `unsafePerformIO` on your main thread, ideally through `SafeApp`
- Adoption remained small, as the community more generally adopted Task

SCALAZ-7 TASK

- `Task[A]` is implemented by a scalaz `Future[Throwable \/ A]`
  - Note: scalaz `Future[A]` does not provide error handling, and is lazy - not running until required
- Allows for asynchronous evaluation through a callback - `unsafePerformAsync((Throwable \/ A) => Unit): Unit`
- Executed through a `Strategy`, which is typically implemented through a Java `ExecutorService`
- Usage has dropped in recent years, as the community moved on to cats-effect and ZIO

CATS-EFFECT IO

- Cancellable, asynchronous execution with explicit error handling (`Either[Throwable, A]`)
- Platform-specific execution - you cannot run a blocking computation on JavaScript
  - In Java - executed through an `ExecutorService`.
    In JavaScript, will use `setTimeout`
- Great support from the Typelevel community
- Inspired a lot by Monix Task - its author is a major contributor to cats-effect

ZIO

- Cancellable, asynchronous, with specified exception types
  - `IO[Nothing, A]` *considered infallible*
  - `IO[Throwable, A]` considered to potentially have an exception
  - `IO[String, A]` allows for an error message without an exception
  - `IO[B, Nothing]` cannot terminate
  - Note: cats-bio implements cats-effect typeclasses as a Bifunctor
- Focus on purity - hard to invoke unsafe functions
- More recent that cats-effect, documentation is not as mature

INTERACTING WITH FUTURES

- Bots cats-effect and ZIO provides ways of converting a scala `Future[A]` to an `IO[A]` lazily, so that it gets executed at the end of the world
- Both cats-effect and ZIO provides ways of executing an `IO[A]` as a scala `Future[A]`
- This makes it easy to integrate in an existing codebase dependent on `Future` and integrate an effect type into your team's codebase (as long as your colleagues are willing to learn)

CATS-EFFECT TYPECLASSES

- The cats-effect project provides an extensive typeclass hierarchy that allows to build asynchronous computation for an arbitrary effect type.
- ZIO provides typeclasses instances to its `IO` type
- djspiewak/shims provides typeclasses instances for both scalaz-7 `IO` and `Task`

THANK YOU