ТИНЬКОФФ

Scala

I

O

ТИНЬКОФФ

Scala

Cats Effect

ZIO

Cats Effect vs ZIO

slido.com with #077428

https://app.sli.do/event/ki8594ms/embed/polls/4ef212bd-c196-42ad-8407-d9f62d29ff12

Haskell

Haskell

```haskell
countZeros :: [Int] -> Int
countZeros numbers =
      var count = 0
      do
        num <- numbers
        if num == 0 then
          count = count + 1
        return count
```

Haskell

```haskell
countZeros :: [Int] -> Int
countZeros numbers =
      var count = 0
      do
        num <- numbers
        if num == 0 then
          count = count + 1
        return count
```

Haskell

```haskell
countZeros :: [Int] -> Int
countZeros numbers =
    let go accum (0 : rest) = go (accum + 1) rest
        go accum (_ : rest) = go accum rest
        go accum []         = accum
    in go 0 numbers
```

Haskell

```haskell
countZeros :: [Int] -> Int
countZeros numbers =
    let go accum (0 : rest) = go (accum + 1) rest
        go accum (_ : rest) = go accum rest
        go accum []         = accum
    in go 0 numbers
```

```haskell
countZeros :: [Int] -> Int
countZeros = length . filter ( == 0)
```

Haskell

```haskell
main :: ?
main = ?
```

Haskell

```haskell
main :: ?
main = ?
```

```haskell
type Dialogue = [Request] -> [Response]
main :: Dialogue
```

Haskell

```haskell
main :: ?
main = ?
```

```haskell
type Dialogue = [Request] -> [Response]
main :: Dialogue
```

```haskell
main :: [String] -> [String]
main (firstLine : nextLine : ...) = let

        in firstOutLine : nextOutLine : ...
```

Haskell

```haskell
main :: ?
main = ?
```

```haskell
type Dialogue = [Request] -> [Response]
main :: Dialogue
```

```haskell
main :: [String] -> [String]
main (firstLine : nextLine : ...) = let

        in firstOutLine : nextOutLine : ...
```

```scala
def main(input: LazyList[String]): LazyList[String] = ???

enum LazyList[+A]:
   case Empty
   case Cons(head: () => A, tail: () => LazyList[A])
```

Haskell

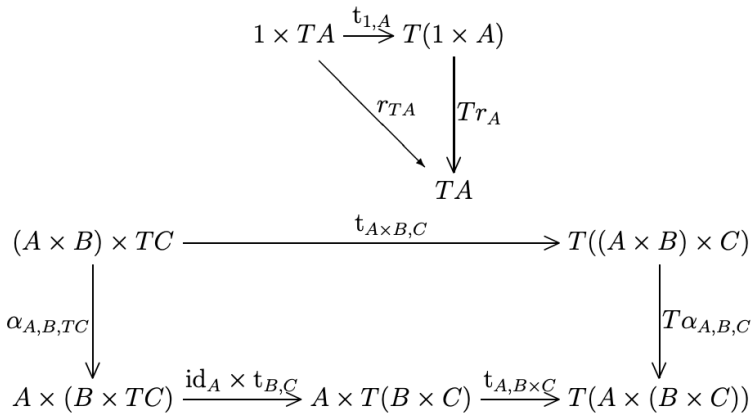
Computational lambda-calculus and monads

Eugenio Moggi*
LFCS
Dept. of Comp. Sci.
University of Edinburgh
EH9 3JZ Edinburgh, UK
em@lfcs.ed.ac.uk

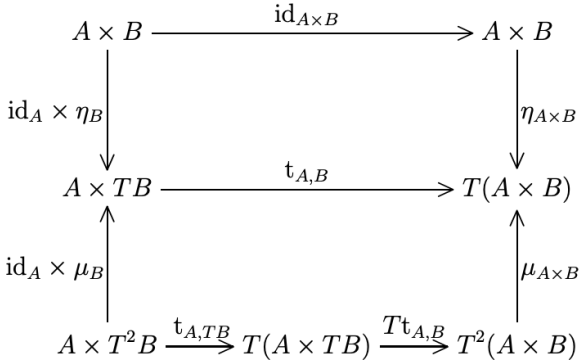
October 1988

Haskell

Computational lambda-calculus and monads



satisfying the following diagrams:



RULE	SYNTAX		SEMANTICS
var	$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash x_i : \tau_i$	$=$	$\pi_i^n ; \eta_{[\![\tau_i]\!]}$
let	$\Gamma \vdash e_1 : \tau_1$	$=$	g_1
	$\Gamma, x_1 : \tau_1 \vdash e_2 : \tau_2$	$=$	g_2
	$\Gamma \vdash (\mathrm{let}\, x_1 = e_1 \,\mathrm{in}\, e_2) : \tau_2$	$=$	$\langle \mathrm{id}_{[\![\Gamma]\!]}, g_1 \rangle ; t_{[\![\Gamma]\!],[\![\tau_1]\!]} ; Tg_2 ; \mu_{[\![\tau_2]\!]}$
*	$\Gamma \vdash * : 1$	$=$	$!_{[\![\Gamma]\!]} ; \eta_1$
	where $!_A$ is the only morphism from A to 1		
$\langle\rangle$	$\Gamma \vdash e_1 : \tau_1$	$=$	g_1
	$\Gamma \vdash e_2 : \tau_2$	$=$	g_2
	$\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2$	$=$	$\langle g_1, g_2 \rangle ; \psi_{[\![\tau_1]\!],[\![\tau_2]\!]}$
π_i	$\Gamma \vdash e : \tau_1 \times \tau_2$	$=$	g
	$\Gamma \vdash \pi_i(e) : \tau_1$	$=$	$g ; T(\pi_i)$

Monads

Monads and composable continuations

PHILIP WADLER (*wadler@dcs.glasgow.ac.uk*)

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ, Scotland

Keywords: Monads, Continuations, Continuation-passing style, Types

Monads

Monads and composable continuations

PHILIP WADLER (*wadler@dcs.glasgow.ac.uk*)

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ, Scotland

$$\text{type } M\ a = (a \to O) \to O$$

$$\begin{array}{lcl} \mathit{unit} & :: & a \to M\ a \\ (\star) & :: & M\ a \to (a \to M\ b) \to M\ b \\ \mathit{eval} & :: & M\ O \to O \end{array}$$

$$\begin{array}{lcl} \mathit{unit}\ v & = & \lambda c.\ c\ v \\ m \star k & = & \lambda c.\ m\ (\lambda v.\ k\ v\ c) \\ \mathit{eval}\ m & = & m\ \mathit{id} \end{array}$$

Continuation-passing style, Types

$$\begin{array}{lcl} \mathit{shift} & :: & ((a \to M\ O) \to M\ O) \to M\ a \\ \mathit{reset} & :: & M\ O \to M\ O \end{array}$$

$$\begin{array}{lcl} \mathit{shift}\ h & = & \lambda c.\ h\ (\lambda v.\ \lambda c'.\ c'\ (c\ v))\ \mathit{id} \\ \mathit{reset}\ m & = & \lambda c.\ c\ (m\ \mathit{id}) \end{array}$$

Monads

Monads and composable continuations

PHILIP WADLER (*wadler@dcs.glasgow.ac.uk*)

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ, Scotland

await

Continuation-passing style, Types

$$\begin{array}{lcl} \text{type } M\ a & = & (a \to O) \to O \\ \\ unit & :: & a \to M\ a \\ (\star) & :: & M\ a \to (a \to M\ b) \to M\ b \\ eval & :: & M\ O \to O \\ \\ unit\ v & = & \lambda c.\ c\ v \\ m \star k & = & \lambda c.\ m\ (\lambda v.\ k\ v\ c) \\ eval\ m & = & m\ id \end{array}$$

$$\begin{array}{lcl} shift & :: & ((a \to M\ O) \to M\ O) \to M\ a \\ reset & :: & M\ O \to M\ O \\ \\ shift\ h & = & \lambda c.\ h\ (\lambda v.\ \lambda c'.\ c'\ (c\ v))\ id \\ reset\ m & = & \lambda c.\ c\ (m\ id) \end{array}$$

async

Monads

```
1 (>>=) :: IO a -> (a -> IO b) -> IO b
```

Monads

How to make *ad-hoc* polymorphism less *ad hoc*

Philip Wadler and Stephen Blott
University of Glasgow*

October 1988

Monads

How to make *ad-hoc* polymorphism less *ad hoc*

Philip Wadler and Stephen Blott
University of Glasgow*

October 1988

Comprehending monads†

PHILIP WADLER

Department of Computing Science, University of Glasgow, G12 8QQ, Scotland.
wadler@dcs.glasgow.ac.uk.

Received 2 January 1991; revised 2 June 1992

Monads

How to make *ad-hoc* polymorphism less *ad hoc*

Philip Wadler and Stephen Blott
University of Glasgow*

October 1988

Comprehending monads†

PHILIP WADLER

Department of Computing Science, University of Glasgow, G12 8QQ, Scotland.

1 2 June 1992

Imperative functional programming

Simon L Peyton Jones Philip Wadler

Dept of Computing Science, University of Glasgow
Email: {simonpj,wadler}@dcs.glagsow.ac.uk

October 1992

IO

```haskell
type IO a = World -> IORes a
data IORes a = MkIORes a World
```

IO

```haskell
type IO a = World -> IORes a
data IORes a = MkIORes a World
```

```haskell
countZeros :: [Int] -> IO Int
countZeros nums = do
  count <- newIORef 0
  forM_ nums \num ->
    when (num == 0) $
      modifyIORef count (+ 1)
  readIORef count
```

Concurrency

Concurrent Haskell

Simon Peyton Jones
University of Glasgow

Andrew Gordon
University of Cambridge

Sigbjorn Finne
University of Glasgow

```haskell
forkIO :: IO () -> IO ThreadId
throwTo :: Exception e => ThreadId -> e -> IO ()

atomicModifyIORef :: IORef a -> (a -> (a, b)) -> IO b
atomicWriteIORef :: IORef a -> a -> IO ()

takeMVar :: MVar a -> IO a
putMVar :: MVar a -> a -> IO ()
readMVar :: MVar a -> IO a

writeChan :: Chan a -> a -> IO ()
readChan :: Chan a -> IO a
```

Concurrency

Concurrent Haskell

Simon Peyton Jones
University of Glasgow

Andrew Gordon
University of Cambridge

Sigbjorn Finne
University of Glasgow

```haskell
forkIO :: IO () -> IO ThreadId
throwTo :: Exception e => ThreadId -> e -> IO ()

atomicModifyIORef :: IORef a -> (a -> (a, b)) -> IO b
atomicWriteIORef :: IORef a -> a -> IO ()

takeMVar :: MVar a -> IO a
putMVar :: MVar a -> a -> IO ()
readMVar :: MVar a -> IO a

writeChan :: Chan a -> a -> IO ()
readChan :: Chan a -> IO a
```

Concurrency

Concurrent Haskell

Simon Peyton Jones
University of Glasgow

Andrew Gordon
University of Cambridge

Sigbjorn Finne
University of Glasgow

```haskell
forkIO :: IO () -> IO ThreadId
throwTo :: Exception e => ThreadId -> e -> IO ()

atomicModifyIORef :: IORef a -> (a -> (a, b)) -> IO b
atomicWriteIORef :: IORef a -> a -> IO ()

takeMVar :: MVar a -> IO a
putMVar :: MVar a -> a -> IO ()
readMVar :: MVar a -> IO a

writeChan :: Chan a -> a -> IO ()
readChan :: Chan a -> IO a
```

STM

Beautiful concurrency

to appear in "Beautiful code", ed Greg Wilson, O'Reilly 2007

Simon Peyton Jones, Microsoft Research, Cambridge

May 1, 2007

```haskell
newTVar :: a -> STM (TVar a)
readTVar :: TVar a -> STM a
writeTVar :: TVar a -> a -> STM ()

atomically :: STM a -> IO a
retry :: STM a

```

STM

Beautiful concurrency

to appear in "Beautiful code", ed Greg Wilson, O'Reilly 2007

Simon Peyton Jones, Microsoft Research, Cambridge

May 1, 2007

```haskell
newTVar :: a -> STM (TVar a)
readTVar :: TVar a -> STM a
writeTVar :: TVar a -> a -> STM ()

atomically :: STM a -> IO a
retry :: STM a

```

Async

```haskell
async :: IO a -> IO (Async a)
wait :: Async a -> IO a
```

Resource

```
1 bracket
2         :: IO a           -- ^ computation to run first (\"acquire resource\")
3         -> (a -> IO b)    -- ^ computation to run last (\"release resource\")
4         -> (a -> IO c)    -- ^ computation to run in-between
5         -> IO c           -- returns the value from the in-between computation
```

Resource

```haskell
bracket
        :: IO a          -- ^ computation to run first (\"acquire resource\")
        -> (a -> IO b)   -- ^ computation to run last (\"release resource\")
        -> (a -> IO c)   -- ^ computation to run in-between
        -> IO c          -- returns the value from the in-between computation
```

```haskell
allocate :: IO a -> IO () -> ResIO (ReleaseKey, a)
runResourceT :: ResIO a -> IO a
```

Streams

- Conduit
- Streaming
- Streamly
- ...

Scala

Akka

2010

```
1 package akka.actor
2
3 trait Actor {
4     abstract def receive: PartialFunction[Any, Unit]
```

Future

2011

```scala
package scala.concurrent

trait Future[+T]{
    def onComplete[U](f: Try[T] => U)(implicit ec: ExecutionContext): Unit

    def map[S](f: T => S)(implicit ec: ExecutionContext): Future[S] = ...

    def flatMap[S](f: T => Future[S])(implicit ec: ExecutionContext): Future[S] =
```

Future

2011

```scala
package scala.concurrent

trait Future[+T]{
    def onComplete[U](f: Try[T] => U)(implicit ec: ExecutionContext): Unit

    def map[S](f: T => S)(implicit ec: ExecutionContext): Future[S] = ...

    def flatMap[S](f: T => Future[S])(implicit ec: ExecutionContext): Future[S] =
```

```scala
def businessFeature(arg: Arg, input: Input): Future[Result] = for {
       info   <- getInfo(arg)
       _      <- doSomething(input)
       result <- getResult(info, input)
    } yield result
```

Future

2011

```scala
package scala.concurrent

trait Future[+T]{
    def onComplete[U](f: Try[T] => U)(implicit ec: ExecutionContext): Unit

    def map[S](f: T => S)(implicit ec: ExecutionContext): Future[S] = ...

    def flatMap[S](f: T => Future[S])(implicit ec: ExecutionContext): Future[S] =
```

```scala
def businessFeature(arg: Arg, input: Input): Future[Result] = for {
       info   <- getInfo(arg)
       _      <- doSomething(input)
       result <- getResult(info, input)
    } yield result
```

Monifu

2013

```scala
trait Scheduler extends ExecutionContext with Executor {
  def execute(command: Runnable): Unit
  def scheduleOnce(initialDelay: Long, unit: TimeUnit, r: Runnable): Cancelable
  def scheduleWithFixedDelay(
        initialDelay: Long, delay: Long, unit: TimeUnit, r: Runnable): Cancelable
}

trait Observer[-A] extends Any with Serializable {
  def onNext(elem: A): Future[Ack]

  def onError(ex: Throwable): Unit

  def onComplete(): Unit
}

abstract class Observable[+A]{
  def subscribe(observer: Observer[A])(implicit s: Scheduler): Cancelable
}
```

Monifu

2013

```scala
trait Scheduler extends ExecutionContext with Executor {
  def execute(command: Runnable): Unit
  def scheduleOnce(initialDelay: Long, unit: TimeUnit, r: Runnable): Cancelable
  def scheduleWithFixedDelay(
        initialDelay: Long, delay: Long, unit: TimeUnit, r: Runnable): Cancelable
}

trait Observer[-A] extends Any with Serializable {
  def onNext(elem: A): Future[Ack]

  def onError(ex: Throwable): Unit

  def onComplete(): Unit
}

abstract class Observable[+A]{
  def subscribe(observer: Observer[A])(implicit s: Scheduler): Cancelable
}
```

Monifu

2013

```scala
trait Scheduler extends ExecutionContext with Executor {
  def execute(command: Runnable): Unit
  def scheduleOnce(initialDelay: Long, unit: TimeUnit, r: Runnable): Cancelable
  def scheduleWithFixedDelay(
        initialDelay: Long, delay: Long, unit: TimeUnit, r: Runnable): Cancelable
}

trait Observer[-A] extends Any with Serializable {
  def onNext(elem: A): Future[Ack]

  def onError(ex: Throwable): Unit

  def onComplete(): Unit
}

abstract class Observable[+A]{
  def subscribe(observer: Observer[A])(implicit s: Scheduler): Cancelable
}
```

Monix

2016

```scala
abstract class Callback[-A] {
  def onSuccess(value: A): Unit
  def onError(ex: Throwable): Unit
}

sealed abstract class Task[+A]{
  def runAsync(cb: Callback[A])(implicit s: Scheduler): Cancelable
}

abstract class MVar[A] {
  def put(a: A): Task[Unit]
  def take: Task[A]
  def read: Task[A]
}
```

scalaz

2010

```
1 package scalaz
2 package effect
3
4 sealed abstract class IO[A] {
5   private[effect]
6         def apply(rw: World[RealWorld]): Trampoline[(World[RealWorld], A)]
```

scalaz

2010

```
1 package scalaz
2 package effect
3
4 sealed abstract class IO[A] {
5   private[effect]
6         def apply(rw: Tower[IvoryTower]): Trampoline[(Tower[IvoryTower], A)]
```

scalaz-stream

2013

```scala
trait Process[+F[_], +O]

object Process {
  case class Await[F[_], A, +O] private[stream](
    req: F[A], recv: A => Process[F,O],
    fallback: Process[F,O],
    cleanup: Process[F,O]) extends Process[F, O]

  case class Emit[F[_], O] private[stream](
    head: Seq[O],
    tail: Process[F,O]) extends Process[F,O]

  case object Halt extends Process[Nothing, Nothing]
```

2015

scalaz → cats

scalaz-stream

↓ ↓ ↓

Functional Streams For Scala

2015

scalaz → cats

scalaz-stream

↓ ↓ ↓

F S F S

2015

scalaz ⟶ cats

scalaz-stream

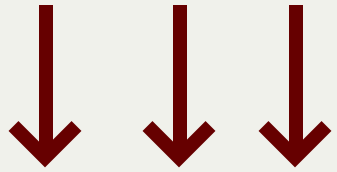
↓ ↓ ↓

$(F\ S)^2$

2015

scalaz ⟶ cats

scalaz-stream



FS2

FS2

```scala
final class Task[+A](private[fs2] val get: Future[Attempt[A]])

trait Semaphore[F[_]] {
  def decrementBy(n: Long): F[Unit]
  def incrementBy(n: Long): F[Unit]
}

trait Queue[F[_], A] {
  def enqueue1(a: A): F[Unit]
  def dequeue1: F[A]
}

trait Topic[F[_], A] {
  def publish1(a: A): F[Unit]
  def subscribe(maxQueued: Int): Stream[F, A]
}

trait Signal[F[_], A] {
  def get: F[A]
  def set(a: A): F[Unit]
}
```

cats-effect

2017

```
1 sealed abstract class IO[+A]
2
3 case class Pure[+A](a: A)                                          extends IO[A]
4 case class Delay[+A](thunk: () => A)                               extends IO[A]
5 case class RaiseError(e: Throwable)                                extends IO[Nothing]
6 case class Suspend[+A](thunk: () => IO[A])                         extends IO[A]
7 case class Bind[E, +A](source: IO[E], f: E => IO[A])               extends IO[A]
8 case class Async[+A](
9    k: (IOConnection, Either[Throwable, A] => Unit) => Unit) extends IO[A]
```

cats-effect

2017

```scala
trait Semaphore[F[_]] {
  def decrementBy(n: Long): F[Unit]
  def incrementBy(n: Long): F[Unit]
}

abstract class MVar[A] {
  def put(a: A): Task[Unit]
  def take: Task[A]
  def read: Task[A]
}

trait Deferred[F[_], A] // Topic {
  def get: F[A]
  def complete(a: A): F[Unit]
}

abstract class Ref[F[_], A] // Signal {
  def get: F[A]
  def set(a: A): F[Unit]
}
```

scalaz8-effect

2017

scalaz8-effect

2017

scalaz-ioeffect

2018

scalaz8-effect

2017

scalaz-ioeffect

2018

scalaz-zio

2018

scalaz8-effect

2017

scalaz-ioeffect

2018

scalaz-zio

2018

ZIO

2019

10

https://app.sli.do/event/ki8594ms/embed/polls/4ef212bd-c196-42ad-8407-d9f62d29ff12

IO

```
1 trait IO[+A]:
2   def run(world: World): (A, World)
3
4   def flatMap[B](f: A => IO[B]): IO[B] = world1 =>
5     val (a, world2) = run(world1)
6     f(a).run(world2)
7
8   def unsafeRun() = run(new World)._1
```

IO

```scala
trait IO[+A]:
  def run(world: World): (A, World)

  def flatMap[B](f: A => IO[B]): IO[B] = world1 =>
    val (a, world2) = run(world1)
    f(a).run(world2)

  def unsafeRun() = run(new World)._1
```

```scala
def countZeros(xs: LazyList[Int]): IO[Int] =
  def go(xs: LazyList[Int], ref: IORef[Int]): IO[Unit] = xs match
    case 0 #:: rest => ref.update(_ + 1).flatMap(_ => go(rest, ref))
    case _ #:: rest => go(rest, ref)
    case _          => IO(())
  for
    ref <- IO.newIORef(0)
    _   <- go(xs, ref)
    x   <- ref.get
  yield x
```

IO

```scala
trait IO[+A]:
  def run(world: World): (A, World)

  def flatMap[B](f: A => IO[B]): IO[B] = world1 =>
    val (a, world2) = run(world1)
    f(a).run(world2)

  def map[B](f: A => B): IO[B] =
    flatMap(a => IO(f(a)))

def go(xs: LazyList[IO[Unit]]): IO[Unit] =
  xs match
    case _ #:: rest => go(rest)
    case _          => IO(())
for
  ref <- IO.newIORef(0)
  _   <- go(xs, ref)
  x   <- ref.get
yield x
```

```
Exception in thread "main" java.lang.StackOverflowError
    at scala.collection.immutable.LazyList$.$anonfun$fill$1(LazyList.scala:1223)
    at scala.collection.immutable.LazyList.scala$collection$immutable$LazyList$$state$lzycompute(LazyList.scala:259)
    at scala.collection.immutable.LazyList.scala$collection$immutable$LazyList$$state(LazyList.scala:252)
    at scala.collection.immutable.LazyList.isEmpty(LazyList.scala:269)
    at scala.collection.IterableOnceOps.nonEmpty(IterableOnce.scala:800)
    at scala.collection.IterableOnceOps.nonEmpty$(IterableOnce.scala:800)
    at scala.package$$hash$colon$colon$.unapply(package.scala:94)
    at playground.talks.iomonad.iomonad$package$.go$3(iomonad.scala:39)
    at playground.talks.iomonad.iomonad$package$.go$2$$anonfun$2(iomonad.scala:39)
    at playground.talks.iomonad.IO.flatMap$$anonfun$1(iomonad.scala:12)
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B] = FlatMap(this, f)

  case Pure(a: A)
  case FlatMap[A, +B](a: IO[A], f: A => IO[B]) extends IO[B]

  @tailrec final def unsafeRun(): A = this match
    case Pure(a)         => a
    case FlatMap(fx, f) =>
      fx match
        case Pure(x)         => f(x).unsafeRun()
        case FlatMap(fy, g) =>
          fy.flatMap(y => g(y).flatMap(f)).unsafeRun()
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B] = FlatMap(this, f)

  case Pure(a: A)
  case FlatMap[A, +B](a: IO[A], f: A => IO[B]) extends IO[B]

  @tailrec final def unsafeRun(): A = this match
    case Pure(a)         => a
    case FlatMap(fx, f) =>
      fx match
        case Pure(x)         => f(x).unsafeRun()
        case FlatMap(fy, g) =>
          fy.flatMap(y => g(y).flatMap(f)).unsafeRun()
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B] = FlatMap(this, f)

  case Pure(a: A)
  case FlatMap[A, +B](a: IO[A], f: A => IO[B]) extends IO[B]

  @tailrec final def unsafeRun(): A = this match
    case Pure(a)         => a
    case FlatMap(fx, f) =>
      fx match
        case Pure(x)         => f(x).unsafeRun()
        case FlatMap(fy, g) =>
          fy.flatMap(y => g(y).flatMap(f)).unsafeRun()
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B]                    =
    Continue(this, _.fold(Throw(_), f))
  def handleWith[A1 >: A](h: Throwable => IO[A1]): IO[A1] =
    Continue(this, _.fold(h, Pure(_)))

  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]

  @tailrec private def unsafeRun1(): Try[A] = this match
    case Pure(a)          => Success(a)
    case Throw(err)       => Failure(err)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1()
        case Throw(err)       => f(Failure(err)).unsafeRun1()
        case Continue(fx, g) => Continue(fx, ta => Continue(g(ta), f)).unsafeRun1()

  final def unsafeRun(): Try[A]             =
    try unsafeRun1()
    catch { case NonFatal(err) => Failure(err) }
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B]                        =
    Continue(this, _.fold(Throw(_), f))
  def handleWith[A1 >: A](h: Throwable => IO[A1]): IO[A1] =
    Continue(this, _.fold(h, Pure(_)))

  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]

  @tailrec private def unsafeRun1(): Try[A] = this match
    case Pure(a)          => Success(a)
    case Throw(err)       => Failure(err)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1()
        case Throw(err)       => f(Failure(err)).unsafeRun1()
        case Continue(fx, g) => Continue(fx, ta => Continue(g(ta), f)).unsafeRun1()

  final def unsafeRun(): Try[A]                    =
    try unsafeRun1()
    catch { case NonFatal(err) => Failure(err) }
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B]                        =
    Continue(this, _.fold(Throw(_), f))
  def handleWith[A1 >: A](h: Throwable => IO[A1]): IO[A1] =
    Continue(this, _.fold(h, Pure(_)))

  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]

  @tailrec private def unsafeRun1(): Try[A] = this match
    case Pure(a)          => Success(a)
    case Throw(err)       => Failure(err)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1()
        case Throw(err)       => f(Failure(err)).unsafeRun1()
        case Continue(fx, g) => Continue(fx, ta => Continue(g(ta), f)).unsafeRun1()

  final def unsafeRun(): Try[A]                    =
    try unsafeRun1()
    catch { case NonFatal(err) => Failure(err) }
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B]                        =
    Continue(this, _.fold(Throw(_), f))
  def handleWith[A1 >: A](h: Throwable => IO[A1]): IO[A1] =
    Continue(this, _.fold(h, Pure(_)))

  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]

  @tailrec private def unsafeRun1(): Try[A] = this match
    case Pure(a)          => Success(a)
    case Throw(err)       => Failure(err)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1()
        case Throw(err)       => f(Failure(err)).unsafeRun1()
        case Continue(fx, g) => Continue(fx, ta => Continue(g(ta), f)).unsafeRun1()

  final def unsafeRun(): Try[A]                    =
    try unsafeRun1()
    catch { case NonFatal(err) => Failure(err) }
```

Scala IO

```scala
enum IO[+A]:
  def flatMap[B](f: A => IO[B]): IO[B]                        =
    Continue(this, _.fold(Throw(_), f))
  def handleWith[A1 >: A](h: Throwable => IO[A1]): IO[A1] =
    Continue(this, _.fold(h, Pure(_)))

  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]

  @tailrec private def unsafeRun1(): Try[A] = this match
    case Pure(a)          => Success(a)
    case Throw(err)       => Failure(err)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1()
        case Throw(err)       => f(Failure(err)).unsafeRun1()
        case Continue(fx, g) => Continue(fx, ta => Continue(g(ta), f)).unsafeRun1()

  final def unsafeRun(): Try[A]                    =
    try unsafeRun1()
    catch { case NonFatal(err) => Failure(err) }
```

Scala IO

```scala
enum IO[+A]:
  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]
  case Async(cont: (Try[A] => Unit) => Unit, ec: ExecutionContext)

  @tailrec private def unsafeRun1(callback: Try[A] => Unit): Unit = this match
    case Pure(a)          => callback(Success(a))
    case Throw(err)       => callback(Failure(err))
    case Async(cont, _)   => cont(callback)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1(callback)
        case Throw(err)       => f(Failure(err)).unsafeRun1(callback)
        case Continue(fx, g) => Continue(fx, u => Continue(g(u), f)).unsafeRun1(callback)
        case Async(cont, ec) => cont(e => ec.execute(() => f(e).unsafeRun(callback)))

  final def unsafeRun(callback: Try[A] => Unit): Unit                =
    try unsafeRun1(callback)
    catch { case NonFatal(err) => callback(Failure(err)) }
```

Scala IO

```scala
enum IO[+A]:
  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]
  case Async(cont: (Try[A] => Unit) => Unit, ec: ExecutionContext)

  @tailrec private def unsafeRun1(callback: Try[A] => Unit): Unit = this match
    case Pure(a)          => callback(Success(a))
    case Throw(err)       => callback(Failure(err))
    case Async(cont, _)   => cont(callback)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1(callback)
        case Throw(err)       => f(Failure(err)).unsafeRun1(callback)
        case Continue(fx, g) => Continue(fx, u => Continue(g(u), f)).unsafeRun1(callback)
        case Async(cont, ec) => cont(e => ec.execute(() => f(e).unsafeRun(callback)))

  final def unsafeRun(callback: Try[A] => Unit): Unit                =
    try unsafeRun1(callback)
    catch { case NonFatal(err) => callback(Failure(err)) }
```

Scala IO

```scala
enum IO[+A]:
  case Pure(a: A)
  case Throw(err: Throwable)
  case Continue[A, +B](a: IO[A], f: Try[A] => IO[B]) extends IO[B]
  case Async(cont: (Try[A] => Unit) => Unit, ec: ExecutionContext)

  @tailrec private def unsafeRun1(callback: Try[A] => Unit): Unit = this match
    case Pure(a)          => callback(Success(a))
    case Throw(err)       => callback(Failure(err))
    case Async(cont, _)   => cont(callback)
    case Continue(fa, f) =>
      fa match
        case Pure(a)          => f(Success(a)).unsafeRun1(callback)
        case Throw(err)       => f(Failure(err)).unsafeRun1(callback)
        case Continue(fx, g) => Continue(fx, u => Continue(g(u), f)).unsafeRun1(callback)
        case Async(cont, ec) => cont(e => ec.execute(() => f(e).unsafeRun(callback)))

  final def unsafeRun(callback: Try[A] => Unit): Unit                =
    try unsafeRun1(callback)
    catch { case NonFatal(err) => callback(Failure(err)) }
```

Common

https://app.sli.do/event/ki8594ms/embed/polls/4ef212bd-c196-42ad-8407-d9f62d29ff12

Common::Ref

```scala
abstract class Ref[F[_], A] {
  def get: F[A]
  def set(a: A): F[Unit]
  def modify[B](f: A => (A, B)): F[B]
  def tryModify[B](f: A => (A, B)): F[Option[B]]
 }

 sealed abstract class Ref[A]{
  def get: UIO[A]
  def set(a: A): UIO[Unit]
  def modify[B](f: A => (B, A)): UIO[B]
  def setAsync(a: A): UIO[Unit]
 }
```

Common::Ref

```scala
abstract class Ref[F[_], A] {
  def get: F[A]
  def set(a: A): F[Unit]
  def modify[B](f: A => (A, B)): F[B]
  def tryModify[B](f: A => (A, B)): F[Option[B]]
 }

 sealed abstract class Ref[A]{
  def get: UIO[A]
  def set(a: A): UIO[Unit]
  def modify[B](f: A => (B, A)): UIO[B]
  def setAsync(a: A): UIO[Unit]
 }
```

Common::Ref

```scala
abstract class Ref[F[_], A] {
  def get: F[A]
  def set(a: A): F[Unit]
  def modify[B](f: A => (A, B)): F[B]
  def tryModify[B](f: A => (A, B)): F[Option[B]]
 }

 sealed abstract class Ref[A]{
  def get: UIO[A]
  def set(a: A): UIO[Unit]
  def modify[B](f: A => (B, A)): UIO[B]
  def setAsync(a: A): UIO[Unit]
 }
```

Common::Ref

```scala
abstract class Ref[F[_], A] {
  def get: F[A]
  def set(a: A): F[Unit]
  def modify[B](f: A => (A, B)): F[B]
  def tryModify[B](f: A => (A, B)): F[Option[B]]
 }

 sealed abstract class Ref[A]{
  def get: UIO[A]
  def set(a: A): UIO[Unit]
  def modify[B](f: A => (B, A)): UIO[B]
  def setAsync(a: A): UIO[Unit]
 }
```

Common::Promise

```scala
abstract class Deferred[F[_], A] {
  def get: F[A]
  def complete(a: A): F[Unit]
 }

final class Promise[E, A]{
  def await: IO[E, A]
  def completeWith(io: IO[E, A]): UIO[Boolean]
}
```

Common::Promise

```scala
abstract class Deferred[F[_], A] {
  def get: F[A]
  def complete(a: A): F[Unit]
 }

final class Promise[E, A]{
  def await: IO[E, A]
  def completeWith(io: IO[E, A]): UIO[Boolean]
}
```

Common::Semaphore

```scala
abstract class Semaphore[F[_]] {
  def acquire: F[Unit]
  def release: F[Unit]
  def permit: Resource[F, Unit]
}

final class Semaphore{
  def withPermits[R, E, A](n: Long)(task: ZIO[R, E, A]): ZIO[R, E, A]
  def withPermitManaged[R, E]: ZManaged[R, E, Unit]
}
```

Common::Semaphore

```scala
abstract class Semaphore[F[_]] {
  def acquire: F[Unit]
  def release: F[Unit]
  def permit: Resource[F, Unit]
}

final class Semaphore{
  def withPermits[R, E, A](n: Long)(task: ZIO[R, E, A]): ZIO[R, E, A]
  def withPermitManaged[R, E]: ZManaged[R, E, Unit]
}
```

Common::Queue

```scala
abstract class Queue[F[_], A] {
  def offer(a: A): F[Unit]
  def take: F[A]
}

abstract class Queue[A]{
  def offer(a: A): UIO[Boolean]
  def take: UIO[A]
}
```

Common::Queue

```scala
abstract class Queue[F[_], A] {
  def offer(a: A): F[Unit]
  def take: F[A]
}

abstract class Queue[A]{
  def offer(a: A): UIO[Boolean]
  def take: UIO[A]
}
```

Common::Resource

```scala
sealed abstract class Resource[F[_], +A]  {
  def use[B](f: A => F[B])(implicit F: BracketThrow[F]): F[B]
  def flatMap[B](f: A => Resource[F, B]): Resource[F, B]
}

object Resource{
  def make[F[_]: Functor, A](
    acquire: F[A])(
    release: A => F[Unit]): Resource[F, A]
}

abstract class ZManaged[-R, +E, +A]{
  def use[B](f: A => ZIO[R, E, B]): ZIO[R, E, B]
  def flatMap[R, E, B](f: A => ZManaged[R, E, B]): ZManaged[R, E, B]
}

object ZManaged{
  def make[R, E, A](
    acquire: ZIO[R, E, A])(
    release: A => ZIO[R, Nothing, Any]): ZManaged[R, E, A]
}
```

Common::Resource

```scala
sealed abstract class Resource[F[_], +A]  {
  def use[B](f: A => F[B])(implicit F: BracketThrow[F]): F[B]
  def flatMap[B](f: A => Resource[F, B]): Resource[F, B]
}

object Resource{
  def make[F[_]: Functor, A](
    acquire: F[A])(
    release: A => F[Unit]): Resource[F, A]
}

abstract class ZManaged[-R, +E, +A]{
  def use[B](f: A => ZIO[R, E, B]): ZIO[R, E, B]
  def flatMap[R, E, B](f: A => ZManaged[R, E, B]): ZManaged[R, E, B]
}

object ZManaged{
  def make[R, E, A](
    acquire: ZIO[R, E, A])(
    release: A => ZIO[R, Nothing, Any]): ZManaged[R, E, A]
}
```

Common::Resource

```scala
sealed abstract class Resource[F[_], +A]  {
  def use[B](f: A => F[B])(implicit F: BracketThrow[F]): F[B]
  def flatMap[B](f: A => Resource[F, B]): Resource[F, B]
}

object Resource{
  def make[F[_]: Functor, A](
    acquire: F[A])(
    release: A => F[Unit]): Resource[F, A]
}

abstract class ZManaged[-R, +E, +A]{
  def use[B](f: A => ZIO[R, E, B]): ZIO[R, E, B]
  def flatMap[R, E, B](f: A => ZManaged[R, E, B]): ZManaged[R, E, B]
}

object ZManaged{
  def make[R, E, A](
    acquire: ZIO[R, E, A])(
    release: A => ZIO[R, Nothing, Any]): ZManaged[R, E, A]
}
```

Common::Resource

```
1  val application: Resource[IO, Unit] = for {
2     config <- readConfig
3     state  <- initializeState(config)
4     db     <- initializeDB(config.db)
5     kafka  <- connectToKafka(config.kafka)
6     _      <- startKafkaListeners(config.topics, kafka, db)
7     _      <- runBackroundWorkers(config.bg)
8     _      <- startHttpServer(config.http, db)
9     _      <- IO.never
10 } yield ()
```

Common::Fiber

```scala
sealed abstract class IO[+A]{
  def start: IO[Fiber[IO, Throwable, A]]
}

trait GenSpawn[F[_], E]{
  def start[A](fa: F[A]): F[Fiber[F, E, A]]
}

trait Fiber[F[_], E, A]{
  def join: F[Outcome[F, E, A]]
  def cancel: F[Unit]
}

sealed trait ZIO[-R, +E, +A]{
  final def fork: URIO[R, Fiber[E, A]]
}

sealed abstract class Fiber[+E, +A]{
  def await: UIO[Exit[E, A]]
  def interrupt: UIO[Exit[E, A]]
}
```

Common::Fiber

```scala
sealed abstract class IO[+A]{
  def start: IO[Fiber[IO, Throwable, A]]
}

trait GenSpawn[F[_], E]{
  def start[A](fa: F[A]): F[Fiber[F, E, A]]
}

trait Fiber[F[_], E, A]{
  def join: F[Outcome[F, E, A]]
  def cancel: F[Unit]
}

sealed trait ZIO[-R, +E, +A]{
  final def fork: URIO[R, Fiber[E, A]]
}

sealed abstract class Fiber[+E, +A]{
  def await: UIO[Exit[E, A]]
  def interrupt: UIO[Exit[E, A]]
}
```

Common::Fiber

```scala
sealed abstract class IO[+A]{
  def start: IO[Fiber[IO, Throwable, A]]
}

trait GenSpawn[F[_], E]{
  def start[A](fa: F[A]): F[Fiber[F, E, A]]
}

trait Fiber[F[_], E, A]{
  def join: F[Outcome[F, E, A]]
  def cancel: F[Unit]
}

sealed trait ZIO[-R, +E, +A]{
  final def fork: URIO[R, Fiber[E, A]]
}

sealed abstract class Fiber[+E, +A]{
  def await: UIO[Exit[E, A]]
  def interrupt: UIO[Exit[E, A]]
}
```

Common::FiberLocal

```scala
sealed abstract class IOLocal[+A]{
  def get: IO[A]
  def set(value: A): IO[Unit]
  def reset: IO[Unit]
}

object IOLocal {
  def apply[A](default: A): IO[IOLocal[A]]
}

final class FiberRef[A]{
  val get: UIO[A]
  def set(value: A): UIO[Unit]
}

object FiberRef {
  def make[A](initial: A,
    fork: A => A = (a: A) => a,
    join: (A, A) => A = ((_: A, a: A) => a)): UIO[FiberRef[A]] =
}
```

Common::FiberLocal

```scala
sealed abstract class IOLocal[+A]{
  def get: IO[A]
  def set(value: A): IO[Unit]
  def reset: IO[Unit]
}

object IOLocal {
  def apply[A](default: A): IO[IOLocal[A]]
}

final class FiberRef[A]{
  val get: UIO[A]
  def set(value: A): UIO[Unit]
}

object FiberRef {
  def make[A](initial: A,
    fork: A => A = (a: A) => a,
    join: (A, A) => A = ((_: A, a: A) => a)): UIO[FiberRef[A]] =
}
```

Common::FiberLocal

```scala
sealed abstract class IOLocal[+A]{
  def get: IO[A]
  def set(value: A): IO[Unit]
  def reset: IO[Unit]
}

object IOLocal {
  def apply[A](default: A): IO[IOLocal[A]]
}

final class FiberRef[A]{
  val get: UIO[A]
  def set(value: A): UIO[Unit]
}

object FiberRef {
  def make[A](initial: A,
    fork: A => A = (a: A) => a,
    join: (A, A) => A = ((_: A, a: A) => a)): UIO[FiberRef[A]] =
}
```

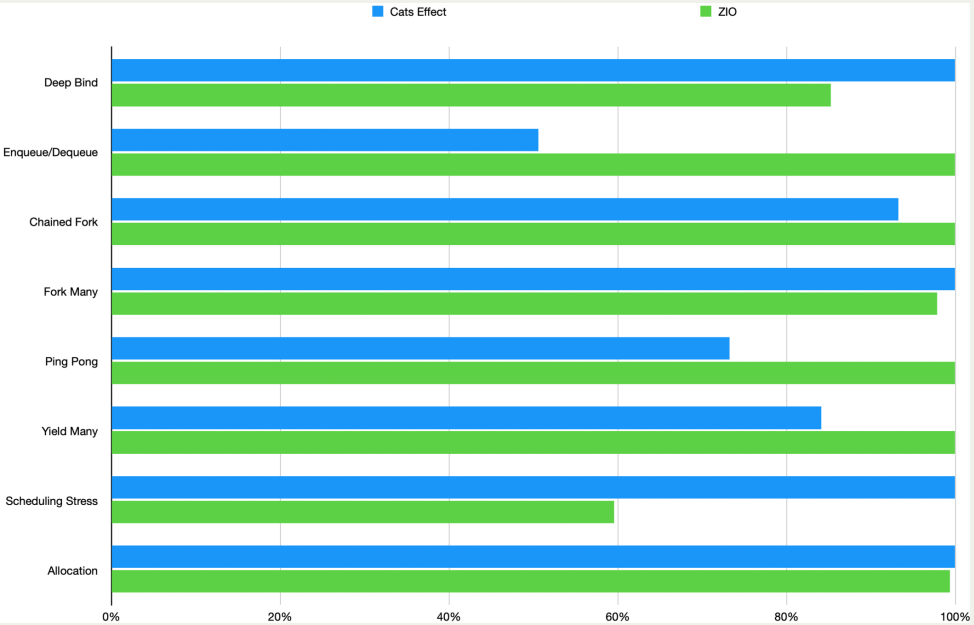
Common

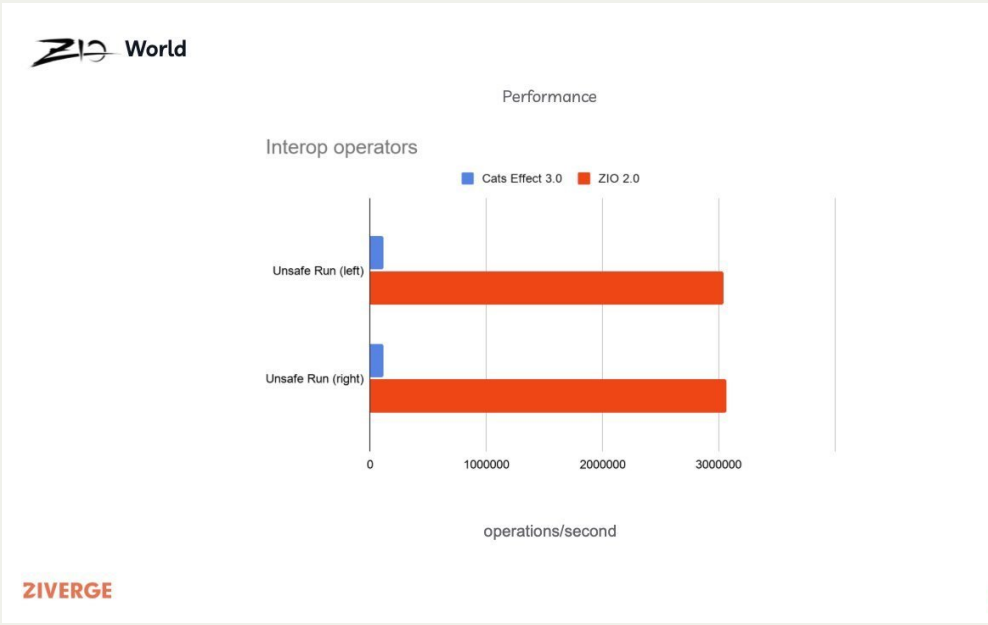
- IO монада
- Структуры коммуникации
  - Ref
  - Queue
  - Promise
  - Semaphore
  - FiberLocals
- Fiber concurrency
- Структурный контроль за ресурсами

Compete

https://app.sli.do/event/ki8594ms/embed/polls/4ef212bd-c196-42ad-8407-d9f62d29ff12

Compete::Performance





Compete::Streams

FS2

- Pull-based
- Doobie
- Http4s
- ....

zio-streams

- iterator-like
- ...:)??

Compete::Streams

FS2

- Pull-based
- Doobie
- Http4s
- ....

zio-streams

- iterator-like
- ...:)??

Compete::Context

```
case class ApplicationContext(
   traceId: TraceId,
   spanId: SpanId,
   currentUser: User,
   pgPool: PostgresPool[IO],
   config: Config
)
```

Compete::Context

```scala
case class ApplicationContext(
   traceId: TraceId,
   spanId: SpanId,
   currentUser: User,
   pgPool: PostgresPool[IO],
   config: Config
)
```

Cats-Effect

```scala
type F[A] = ReaderT[ApplicationContext, IO, A]

final case class Kleisli[F[_], -A, B](run: A => F[B]){
  def local[C](f: C => A): Kleisli[F, C, B]
  def ask[F[_], A](implicit F: Applicative[F]): Kleisli[F, A, A]
}
```

Compete::Context

```scala
case class ApplicationContext(
   traceId: TraceId,
   spanId: SpanId,
   currentUser: User,
   pgPool: PostgresPool[IO],
   config: Config
)
```

Cats-Effect

```scala
type F[A] = ReaderT[ApplicationContext, IO, A]

final case class Kleisli[F[_], -A, B](run: A => F[B]){
  def local[C](f: C => A): Kleisli[F, C, B]
  def ask[F[_], A](implicit F: Applicative[F]): Kleisli[F, A, A]
}
```

ZIO

```scala
sealed trait ZIO[-R, +E, +A]{
  def provide(r: R): IO[E, A] = ZIO.provide(r)(self)
  def provideSome[C](f: C => R): ZIO[C, E, A]
}

object ZIO{
  def environment[R]: URIO[R, R] = access(r => r)
}
```

Compete::Context

```scala
case class ApplicationContext(
   traceId: TraceId,
   spanId: SpanId,
   currentUser: User,
   pgPool: PostgresPool[IO],
   config: Config
)
```

Cats-Effect

```scala
type F[A] = ReaderT[ApplicationContext, IO, A]

final case class Kleisli[F[_], -A, B](run: A => F[B]){
  def local[C](f: C => A): Kleisli[F, C, B]
  def ask[F[_], A](implicit F: Applicative[F]): Kleisli[F, A, A]
}
```

ZIO

```scala
sealed trait ZIO[-R, +E, +A]{
  def provide(r: R): IO[E, A] = ZIO.provide(r)(self)
  def provideSome[C](f: C => R): ZIO[C, E, A]
}

object ZIO{
  def environment[R]: URIO[R, R] = access(r => r)
}
```

Compete::Unsafe Execution

Cats-Effect 2

```
trait Effect[F[_]]
     def runAsync[A](fa: F[A])(cb: Either[Throwable, A] => IO[Unit]): SyncIO[Unit]
}
```

Compete::Unsafe Execution

Cats-Effect 2

```
trait Effect[F[_]]
     def runAsync[A](fa: F[A])(cb: Either[Throwable, A] => IO[Unit]): SyncIO[Unit]
}
```

Cats-Effect 3

```
trait Dispatcher[F[_]]{
     def unsafeToFuture[A](fa: F[A]): Future[A]
}
object Dispatcher{
     def apply[F[_]](implicit F: Async[F]): Resource[F, Dispatcher[F]]
}
```

Compete::Unsafe Execution

Cats-Effect 2

```
trait Effect[F[_]]
     def runAsync[A](fa: F[A])(cb: Either[Throwable, A] => IO[Unit]): SyncIO[Unit]
}
```

Cats-Effect 3

```
trait Dispatcher[F[_]]{
     def unsafeToFuture[A](fa: F[A]): Future[A]
}
object Dispatcher{
     def apply[F[_]](implicit F: Async[F]): Resource[F, Dispatcher[F]]
}
```

ZIO

```
object ZIO{
   def runtime[R]: URIO[R, Runtime[R]]
}
trait Runtime[R]{
  def unsafeRunAsync[E, A](zio: => ZIO[R, E, A])(k: Exit[E, A] => Any): Unit
}
```

Compete::Unsafe Execution

Cats-Effect 2

```
trait Effect[F[_]]
     def runAsync[A](fa: F[A])(cb: Either[Throwable, A] => IO[Unit]): SyncIO[Unit]
}
```

Cats-Effect 3

```
trait Dispatcher[F[_]]{
     def unsafeToFuture[A](fa: F[A]): Future[A]
}
object Dispatcher{
     def apply[F[_]](implicit F: Async[F]): Resource[F, Dispatcher[F]]
}
```

ZIO

```
object ZIO{
   def runtime[R]: URIO[R, Runtime[R]]
}
trait Runtime[R]{
  def unsafeRunAsync[E, A](zio: => ZIO[R, E, A])(k: Exit[E, A] => Any): Unit
}
```

Compete::DomainErrors

Vanilla

```
1 trait UserRepository{
2   def findUser(id: UserId): IO[Either[UserNotFound, User]]
3   def createUser(user: User): IO[Either[AlreadyExists, UserId]]
4 }
```

Compete::DomainErrors

Vanilla

```
1 trait UserRepository{
2   def findUser(id: UserId): IO[Either[UserNotFound, User]]
3   def createUser(user: User): IO[Either[AlreadyExists, UserId]]
4 }
```

Cats-Effect

```
1 trait UserRepository{
2   def findUser(id: UserId): EitherT[IO, UserNotFound, User]]
3   def createUser(user: User): EitherT[IO, AlreadyExists, UserId]]
4 }
```

Compete::DomainErrors

Vanilla

```scala
trait UserRepository{
  def findUser(id: UserId): IO[Either[UserNotFound, User]]
  def createUser(user: User): IO[Either[AlreadyExists, UserId]]
}
```

Cats-Effect

```scala
trait UserRepository{
  def findUser(id: UserId): EitherT[IO, UserNotFound, User]]
  def createUser(user: User): EitherT[IO, AlreadyExists, UserId]]
}
```

ZIO

```scala
trait UserRepository{
  def findUser(id: UserId): IO[UserNotFound, User]
  def createUser(user: User): IO[AlreadyExists, UserId]
}
```

Compete::DomainErrors

Vanilla

```
trait UserRepository{
  def findUser(id: UserId): IO[Either[UserNotFound, User]]
  def createUser(user: User): IO[Either[AlreadyExists, UserId]]
}
```

Cats-Effect

```
trait UserRepository{
  def findUser(id: UserId): EitherT[IO, UserNotFound, User]]
  def createUser(user: User): EitherT[IO, AlreadyExists, UserId]]
}
```

ZIO

```
trait UserRepository{
  def findUser(id: UserId): IO[UserNotFound, User]
  def createUser(user: User): IO[AlreadyExists, UserId]
}
```

Cats Effect Specials

Cats Effect Specials

- F[_] + Type-class hierarchy

Cats Effect Specials

- F[_] + Type-class hierarchy
- Fiber-aware thread pool (zio 2?)

ZIO Specials

ZIO Specials

- STM (cats-stm?)

ZIO Specials

- STM (cats-stm?)
- Module-pattern
  - Module-pattern 2.0 + ZLayer
  - Distage

ZIO Specials

- STM (cats-stm?)
- Module-pattern
  - Module-pattern 2.0 + ZLayer
  - Distage
- ZSchedule (fs-timeseries?)

Итоги

Итоги

- Используйте cats-effect

Итоги

- Используйте cats-effect
- Используйте ZIO

Итоги

- Используйте cats-effect
- Используйте ZIO
- Используйте Tofu

github.com/tofu-tf/tofu

Благодарю за внимание

t.me/odomontois

t.me/scala_ru

o.nizhnikov@tinkoff.ru

Эти слайды:

slides.com/olegnizhnik/scala_ios

ТИНЬКОФФ

https://app.sli.do/event/ki8594ms/embed/polls/4ef212bd-c196-42ad-8407-d9f62d29ff12