Tour of Typed Error Handling

Jacob Wang

Feb 2026, London Scala User Group

Hello 👋

- Writing Scala since 2015
- Maintainer of libraries like Difflicious, Doobie, etc
- @jatcwang (GitHub, mas.to, Bluesky)

Today’s itinerary

- Why typed error handling?
- Which errors should we track with types?
- Tour of error handling libraries

Errors! Failures! Faults! 💥

Errors! Failures! Faults! 💥

- Errors are a fact of life for (almost) any program

Errors! Failures! Faults! 💥

- Errors are a fact of life for (almost) any program
- The basics: `try-catch`, `IO/Future`'s `recoverWith`

Errors! Failures! Faults! 💥

- Errors are a fact of life for (almost) any program
- The basics: `try-catch`, `IO/Future`'s `recoverWith`
- Untyped
  - Read the documentation, or implementation!

```
def uploadFile(userId: UserId, parentPath: Path, file: File): IO[Unit] =
 ...
```

The case for typed errors

The case for typed errors

Putting possible errors in the types:

- Exhaustive error handling, checked by the compiler.
- Free documentation!

The case for typed errors

Putting possible errors in the types:

- Exhaustive error handling, checked by the compiler.
- Free documentation!

```
def uploadFile(userId: UserId, parentPath: Path, file: File): IO[Unit]
```

```
enum FileUploadError {
  case UnauthorizedUpload( ... )
  case NotEnoughStorageQuota( ... )
  case FileAlreadyExist( ... )
}
```

Typed errors

But which errors?

Typed errors

But which errors?

- Can the errors be **meaningfully** handled by the function caller?
  - Additional actions on error
  - Different code paths for different errors

Typed errors

But which errors?

- Can the errors be **meaningfully** handled by the function caller?
  - Additional actions on error
  - Different code paths for different errors

When NOT to use typed errors?

- The caller can’t do much with it other than rethrowing it

Typed errors - but which ones?

What errors should we track with types?

- **Fatal errors** (OutOfMemoryError, StackOverflowError)
- **Bugs** (AssertionError)
- **Expected-unexpected failures** (Network / IO exceptions)
- **Domain errors** (Validation / Incorrect state)

Typed errors - but which ones?

What errors should we track with types?

- ~~Fatal errors (OutOfMemoryError, StackOverflowError)~~
- ~~Bugs (AssertionError)~~
- **Expected-unexpected failures** (Network / IO exceptions) Depends...?
- **Domain errors** (Validation / Incorrect state) Great!

What’s a good error handling mechanism?

What’s a good error handling mechanism?

- Succinct
  - Good type inference

What’s a good error handling mechanism?

- Succinct
  - Good type inference
- Precise
  - Each function expresses its possible errors

What’s a good error handling mechanism?

- Succinct
  - Good type inference
- Precise
  - Each function expresses its possible errors
- Safe
  - ...from refactorings and user mistakes

Example

```
def upload(user: User, path: Path, file: File)
```

The function either succeeds or fail with `FileUploadError`:

```
enum FileUploadError:
  case UnauthorizedUpload()
  case NotEnoughStorageQuota()
  case FileAlreadyExist()
```

Or if modeled using union types:

```
type FileUploadError =
  UnauthorizedUpload |
  NotEnoughStorageQuota |
  FileAlreadyExist
```

 *Let the tour begin!* 

IO[Either[E, A]]

IO[Either[E, A]]

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  getQuotaForPath(user, path).flatMap {
    case Left(e) ⇒ IO.pure(Left(e))
    case Right(bytesQuota) ⇒
      if (bytesQuota - bytes.length ⩾ 0) IO.pure(Left(NotEnoughStorageQuota()))
      else
        getFile(path).flatMap {
          case Some(f) ⇒ IO.pure(Left(FileAlreadyExist()))
          case None    ⇒ doUpload(path, bytes).map(_ ⇒ Right(()))
        }
  }
```

IO[Either[E, A]]

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  getQuotaForPath(user, path).flatMap {
    case Left(e) ⇒ IO.pure(Left(e))
    case Right(bytesQuota) ⇒
      if (bytesQuota - bytes.length ⩾ 0) IO.pure(Left(NotEnoughStorageQuota()))
      else
        getFile(path).flatMap {
          case Some(f) ⇒ IO.pure(Left(FileAlreadyExist()))
          case None    ⇒ doUpload(path, bytes).map(_ ⇒ Right(()))
        }
  }
```

IO[Either[E, A]]

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  getQuotaForPath(user, path).flatMap {
    case Left(e) ⇒ IO.pure(Left(e))
    case Right(bytesQuota) ⇒
      if (bytesQuota - bytes.length ⩾ 0) IO.pure(Left(NotEnoughStorageQuota()))
      else
        getFile(path).flatMap {
          case Some(f) ⇒ IO.pure(Left(FileAlreadyExist()))
          case None    ⇒ doUpload(path, bytes).map(_ ⇒ Right(()))
        }
  }
```

IO[Either[E, A]]

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  getQuotaForPath(user, path).flatMap {
    case Left(e) ⇒ IO.pure(Left(e))
    case Right(bytesQuota) ⇒
      if (bytesQuota - bytes.length ⩾ 0) IO.pure(Left(NotEnoughStorageQuota()))
      else
        getFile(path).flatMap {
          case Some(f) ⇒ IO.pure(Left(FileAlreadyExist()))
          case None    ⇒ doUpload(path, bytes).map(_ ⇒ Right(()))
        }
  }
```

IO[Either[E, A]]

```
1  def step1(): IO[Either[Err, A]] = ...
2  def step2(): IO[Either[Err, B]] = ...
3  def step3(): IO[Either[Err, C]] = ...
4
5  def run(): IO[Either[E, A]] =
6    step1().flatMap {
7      case Left(err) ⇒ IO.pure(Left(err))
8      case Right(a) ⇒ step2().flatMap {
9        case Left(err) ⇒ IO.pure(Left(err))
10       case Right(b) ⇒ step3().map(Right(_))
11     }
12   }
```

IO[Either[E, A]]

```scala
def step1(): IO[Either[Err, A]] = ...
def step2(): IO[Either[Err, B]] = ...
def step3(): IO[Either[Err, C]] = ...

def run(): IO[Either[E, A]] =
  step1().flatMap {
    case Left(err) ⇒ IO.pure(Left(err))
    case Right(a) ⇒ step2().flatMap {
      case Left(err) ⇒ IO.pure(Left(err))
      case Right(b) ⇒ step3().map(Right(_))
    }
  }
```

EitherT

EitherT

```
case class EitherT[F[_], E, A](value: F[Either[E, A]])
```

- A thin wrapper around e.g. `IO[Either[E, A]]`
- `EitherT`'s `flatMap` handles the short-circuiting

EitherT

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  (for {
    bytesQuota ← EitherT(getQuotaForPath(user, path))
    _ ← EitherT.cond[IO](bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← EitherT(
      getFile(path).flatMap {
        case None    ⇒ IO(Right(()))
        case Some(_) ⇒ IO(Left(FileAlreadyExist()))
      },
    )
    _ ← EitherT.liftF[IO, FileUploadError, Unit](doUpload(path, bytes))
  } yield ()).value
}
```

EitherT

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  (for {
    bytesQuota ← EitherT(getQuotaForPath(user, path))
    _ ← EitherT.cond[IO](bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← EitherT(
      getFile(path).flatMap {
        case None    ⇒ IO(Right(()))
        case Some(_) ⇒ IO(Left(FileAlreadyExist()))
      },
    )
    _ ← EitherT.liftF[IO, FileUploadError, Unit](doUpload(path, bytes))
  } yield ()).value
}
```

EitherT

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  (for {
    bytesQuota ← EitherT(getQuotaForPath(user, path))
    _ ← EitherT.cond[IO](bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← EitherT(
      getFile(path).flatMap {
        case None    ⇒ IO(Right(()))
        case Some(_) ⇒ IO(Left(FileAlreadyExist()))
      },
    )
    _ ← EitherT.liftF[IO, FileUploadError, Unit](doUpload(path, bytes))
  } yield ()).value
}
```

EitherT

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  (for {
    bytesQuota ← EitherT(getQuotaForPath(user, path))
    _ ← EitherT.cond[IO](bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← EitherT(
      getFile(path).flatMap {
        case None    ⇒ IO(Right(()))
        case Some(_) ⇒ IO(Left(FileAlreadyExist()))
      },
    )
    _ ← EitherT.liftF[IO, FileUploadError, Unit](doUpload(path, bytes))
  } yield ()).value
}
```

EitherT

```scala
def getQuotaForPath(user: User, path: Path): IO[Either[UnauthorizedUpload, Int]] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  (for {
    bytesQuota ← EitherT(getQuotaForPath(user, path))
    _ ← EitherT.cond[IO](bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← EitherT(
      getFile(path).flatMap {
        case None    ⇒ IO(Right(()))
        case Some(_) ⇒ IO(Left(FileAlreadyExist()))
      },
    )
    _ ← EitherT.liftF[IO, FileUploadError, Unit](doUpload(path, bytes))
  } yield ()).value
}
```

ZIO

ZIO

`ZIO[-R, +E, +A]`

- Integrates typed error directly into the effect type (`E`)

ZIO

```scala
import zio.*

def getQuotaForPath(user: User, path: Path): ZIO[Any, UnauthorizedUpload, Int] = ???
// UIO is an alias for ZIO[Any, Nothing, A]
def getFile(path: Path): UIO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): UIO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): ZIO[Any, FileUploadError, Unit] = {
  for {
    bytesQuota ← getQuotaForPath(user, path)
    _ ← ZIO.cond(bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← getFile(path).filterOrFail(_.isEmpty)(FileAlreadyExist())
    _ ← doUpload(path, bytes)
  } yield ()
}
```

ZIO

```scala
import zio.*

def getQuotaForPath(user: User, path: Path): ZIO[Any, UnauthorizedUpload, Int] = ???
// UIO is an alias for ZIO[Any, Nothing, A]
def getFile(path: Path): UIO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): UIO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): ZIO[Any, FileUploadError, Unit] = {
  for {
    bytesQuota ← getQuotaForPath(user, path)
    _ ← ZIO.cond(bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← getFile(path).filterOrFail(_.isEmpty)(FileAlreadyExist())
    _ ← doUpload(path, bytes)
  } yield ()
}
```

ZIO

```scala
import zio.*

def getQuotaForPath(user: User, path: Path): ZIO[Any, UnauthorizedUpload, Int] = ???
// UIO is an alias for ZIO[Any, Nothing, A]
def getFile(path: Path): UIO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): UIO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): ZIO[Any, FileUploadError, Unit] = {
  for {
    bytesQuota ← getQuotaForPath(user, path)
    _ ← ZIO.cond(bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← getFile(path).filterOrFail(_.isEmpty)(FileAlreadyExist())
    _ ← doUpload(path, bytes)
  } yield ()
}
```

ZIO

```scala
import zio.*

def getQuotaForPath(user: User, path: Path): ZIO[Any, UnauthorizedUpload, Int] = ???
// UIO is an alias for ZIO[Any, Nothing, A]
def getFile(path: Path): UIO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): UIO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): ZIO[Any, FileUploadError, Unit] = {
  for {
    bytesQuota ← getQuotaForPath(user, path)
    _ ← ZIO.cond(bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← getFile(path).filterOrFail(_.isEmpty)(FileAlreadyExist())
    _ ← doUpload(path, bytes)
  } yield ()
}
```

ZIO

```scala
import zio.*

def getQuotaForPath(user: User, path: Path): ZIO[Any, UnauthorizedUpload, Int] = ???
// UIO is an alias for ZIO[Any, Nothing, A]
def getFile(path: Path): UIO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): UIO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): ZIO[Any, FileUploadError, Unit] = {
  for {
    bytesQuota ← getQuotaForPath(user, path)
    _ ← ZIO.cond(bytesQuota - bytes.length ⩾ 0, (), NotEnoughStorageQuota())
    _ ← getFile(path).filterOrFail(_.isEmpty)(FileAlreadyExist())
    _ ← doUpload(path, bytes)
  } yield ()
}
```

cats-mtl / IOHandle

cats-mtl

- `Raise[F, E]`

```
def doThings(param: Int)(given Raise[IO, PossibleErrors]): IO[Int]
```

cats-mtl / IOHandle

cats-mtl:

- `Raise[F, -E]` - Capability to raise errors of type `E` in effect `F`
- `Handle[F, E]` - Extends `Raise`, and can incepcept errors of type `E` in effect `F`

cats-mtl / IOHandle

cats-mtl:

- `Raise[F, -E]` - Capability to raise errors of type `E` in effect `F`
- `Handle[F, E]` - Extends `Raise`, and can incepcept errors of type `E` in effect `F`

IOHandle

- A library specializes the above for `cats.effect.IO` (+ many more conveniences!)
- `IORaise` and `IOHandle`

```
def doThings(param: Int)(given IORaise[PossibleErrors]): IO[Int]
```

cats-mtl / IOHandle

IOHandle Usage:

- Call `ioHandling[E]` to open a scope
- Call `ioAbort(err)` to abort with an error
- Handle the result with methods like:
  - `.toEither`: Converts to `IO[Either[E, A]]`
  - `.rescueWith`: Handle the error directly

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 2:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(implicit raise: IORaise[UnauthorizedUpload]): IO[Int
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
  ioHandling[FileUploadError] { implicit handle: IORaise[FileUploadError] ⇒
    for {
      bytesQuota ← getQuotaForPath(user, path)
      _ ← if (bytesQuota - bytes.length < 0)
            ioAbort(NotEnoughStorageQuota())
          else IO.unit
      _ ← getFile(path).flatMap(maybeFile ⇒ ioAbortIf(maybeFile.nonEmpty, FileAlreadyExist()))
      _ ← doUpload(path, bytes)
    } yield ()
  }.toEither
```

IOHandle

Scala 3 + more helper methods:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(using IORaise[UnauthorizedUpload]): IO[Int] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
    ioHandling[FileUploadError]:
      for
        bytesQuota ← getQuotaForPath(user, path)
        _ ← ioAbortIf(bytesQuota - bytes.length < 0, NotEnoughStorageQuota())
        _ ← getFile(path).abortIf(_.nonEmpty, FileAlreadyExist())
        _ ← doUpload(path, bytes)
      yield ()
    .toEither
```

IOHandle

Scala 3 + more helper methods:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(using IORaise[UnauthorizedUpload]): IO[Int] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
    ioHandling[FileUploadError]:
      for
        bytesQuota ← getQuotaForPath(user, path)
        _ ← ioAbortIf(bytesQuota - bytes.length < 0, NotEnoughStorageQuota())
        _ ← getFile(path).abortIf(_.nonEmpty, FileAlreadyExist())
        _ ← doUpload(path, bytes)
      yield ()
    .toEither
```

IOHandle

Scala 3 + more helper methods:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(using IORaise[UnauthorizedUpload]): IO[Int] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
    ioHandling[FileUploadError]:
      for
        bytesQuota ← getQuotaForPath(user, path)
        _ ← ioAbortIf(bytesQuota - bytes.length < 0, NotEnoughStorageQuota())
        _ ← getFile(path).abortIf(_.nonEmpty, FileAlreadyExist())
        _ ← doUpload(path, bytes)
      yield ()
    .toEither
```

IOHandle

Scala 3 + more helper methods:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(using IORaise[UnauthorizedUpload]): IO[Int] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
    ioHandling[FileUploadError]:
      for
        bytesQuota ← getQuotaForPath(user, path)
        _ ← ioAbortIf(bytesQuota - bytes.length < 0, NotEnoughStorageQuota())
        _ ← getFile(path).abortIf(_.nonEmpty, FileAlreadyExist())
        _ ← doUpload(path, bytes)
      yield ()
    .toEither
```

IOHandle

Scala 3 + more helper methods:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(using IORaise[UnauthorizedUpload]): IO[Int] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
    ioHandling[FileUploadError]:
      for
        bytesQuota ← getQuotaForPath(user, path)
        _ ← ioAbortIf(bytesQuota - bytes.length < 0, NotEnoughStorageQuota())
        _ ← getFile(path).abortIf(_.nonEmpty, FileAlreadyExist())
        _ ← doUpload(path, bytes)
      yield ()
    .toEither
```

IOHandle

Scala 3 + more helper methods:

```
import iohandle.*

def getQuotaForPath(user: User, path: Path)(using IORaise[UnauthorizedUpload]): IO[Int] = ???
def getFile(path: Path): IO[Option[File]] = ???
def doUpload(path: Path, bytes: Array[Byte]): IO[Unit] = ???

def upload(user: User, path: Path, bytes: Array[Byte]): IO[Either[FileUploadError, Unit]] =
    ioHandling[FileUploadError]:
      for
        bytesQuota ← getQuotaForPath(user, path)
        _ ← ioAbortIf(bytesQuota - bytes.length < 0, NotEnoughStorageQuota())
        _ ← getFile(path).abortIf(_.nonEmpty, FileAlreadyExist())
        _ ← doUpload(path, bytes)
      yield ()
    .toEither
```

How it works

How it works

- `ioHandling` : Creates a unique marker, carried in `IORaise` capability object

How it works

- `ioHandling` : Creates a unique marker, carried in `IORaise` capability object
- `ioAbort` : throws `IOHandleErrorWrapper` with the marker
  - `class IOHandleErrorWrapper[E](error: E, marker: AnyRef) extends RuntimeException`

How it works

- `ioHandling` : Creates a unique marker, carried in `IORaise` capability object
- `ioAbort` : throws `IOHandleErrorWrapper` with the marker
  - `class IOHandleErrorWrapper[E](error: E, marker: AnyRef) extends RuntimeException`
- Handler methods (`.toEither`) catch and handle `IOHandleErrorWrapper` with the expected marker

Caveats

- `IOHandleErrorWrapper` can be unintentionally caught & swallowed by user code!
- Solution: use `handleUnexpectedWith` instead of `IO#handleErrorWith`

```
ioHandling[MyError]:
  checkSomething()
    .flatMap(succeeded ⇒ ioAbortIf(!succeeded, BadResult(..)))
    .handleErrorWith: e ⇒
        IO.println("something bad happened!") // swallowed! :(
```

Caveats

- `IOHandleErrorWrapper` can be unintentionally caught & swallowed by user code!
- Solution: use `handleUnexpectedWith` instead of `IO#handleErrorWith`

```
ioHandling[MyError]:
  checkSomething()
    .flatMap(succeeded ⇒ ioAbortIf(!succeeded, BadResult(..)))
    .handleUnexpectedWith: e ⇒
        IO.println("something bad happened!")
```

Ox

Ox

- A library for **direct-style** concurrency
- Error-handling utility built on top of **boundary-break**

Ox

- A library for **direct-style** concurrency
- Error-handling utility built on top of **boundary-break**
- Usage:
  - `ox.either` to start a scope
  - Unwrap `Either`s with `.ok()`
  - `someError.fail()` to abort with an error

Ox

```
1  import ox.either
2  import ox.either.{fail, ok}
3
4  def getQuotaForPath(user: User, path: Path): Either[UnauthorizedUpload, Int] = ???
5  def getFile(path: Path): Option[File] = ???
6  def doUpload(path: Path, bytes: Array[Byte]): Unit = ???
7
8  def upload(user: User, path: Path, bytes: Array[Byte]): Either[FileUploadError, Unit] =
9    either:
10     val bytesQuota: Int = getQuotaForPath(user, path).ok()
11     if bytesQuota - bytes.length < 0 then NotEnoughStorageQuota().fail()
12     if getFile(path).nonEmpty then FileAlreadyExist().fail()
13     doUpload(path, bytes)
```

Ox

```scala
import ox.either
import ox.either.{fail, ok}

def getQuotaForPath(user: User, path: Path): Either[UnauthorizedUpload, Int] = ???
def getFile(path: Path): Option[File] = ???
def doUpload(path: Path, bytes: Array[Byte]): Unit = ???

def upload(user: User, path: Path, bytes: Array[Byte]): Either[FileUploadError, Unit] =
  either:
    val bytesQuota: Int = getQuotaForPath(user, path).ok()
    if bytesQuota - bytes.length < 0 then NotEnoughStorageQuota().fail()
    if getFile(path).nonEmpty then FileAlreadyExist().fail()
    doUpload(path, bytes)
```

Ox

```scala
import ox.either
import ox.either.{fail, ok}

def getQuotaForPath(user: User, path: Path): Either[UnauthorizedUpload, Int] = ???
def getFile(path: Path): Option[File] = ???
def doUpload(path: Path, bytes: Array[Byte]): Unit = ???

def upload(user: User, path: Path, bytes: Array[Byte]): Either[FileUploadError, Unit] =
  either:
    val bytesQuota: Int = getQuotaForPath(user, path).ok()
    if bytesQuota - bytes.length < 0 then NotEnoughStorageQuota().fail()
    if getFile(path).nonEmpty then FileAlreadyExist().fail()
    doUpload(path, bytes)
```

Ox

```scala
import ox.either
import ox.either.{fail, ok}

def getQuotaForPath(user: User, path: Path): Either[UnauthorizedUpload, Int] = ???
def getFile(path: Path): Option[File] = ???
def doUpload(path: Path, bytes: Array[Byte]): Unit = ???

def upload(user: User, path: Path, bytes: Array[Byte]): Either[FileUploadError, Unit] =
  either:
    val bytesQuota: Int = getQuotaForPath(user, path).ok()
    if bytesQuota - bytes.length < 0 then NotEnoughStorageQuota().fail()
    if getFile(path).nonEmpty then FileAlreadyExist().fail()
    doUpload(path, bytes)
```

Ox

```scala
import ox.either
import ox.either.{fail, ok}

def getQuotaForPath(user: User, path: Path): Either[UnauthorizedUpload, Int] = ???
def getFile(path: Path): Option[File] = ???
def doUpload(path: Path, bytes: Array[Byte]): Unit = ???

def upload(user: User, path: Path, bytes: Array[Byte]): Either[FileUploadError, Unit] =
  either:
    val bytesQuota: Int = getQuotaForPath(user, path).ok()
    if bytesQuota - bytes.length < 0 then NotEnoughStorageQuota().fail()
    if getFile(path).nonEmpty then FileAlreadyExist().fail()
    doUpload(path, bytes)
```

Ox

```
1  import ox.either
2  import ox.either.{fail, ok}
3
4  def getQuotaForPath(user: User, path: Path): Either[UnauthorizedUpload, Int] = ???
5  def getFile(path: Path): Option[File] = ???
6  def doUpload(path: Path, bytes: Array[Byte]): Unit = ???
7
8  def upload(user: User, path: Path, bytes: Array[Byte]): Either[FileUploadError, Unit] =
9    either:
10     val bytesQuota: Int = getQuotaForPath(user, path).ok()
11     if bytesQuota - bytes.length < 0 then NotEnoughStorageQuota().fail()
12     if getFile(path).nonEmpty then FileAlreadyExist().fail()
13     doUpload(path, bytes)
```

Summary

Summary

How do the libraries compare when it comes to typed-errors?

Library	Precise	Succint?	Footguns / edgecases?
EitherT	Yes	Not really	Some*
cats-mtl/IOHandle	Yes	Decent	Some
ox	Yes	Great	Some
ZIO	Yes	Great	None

Honourable mentions

- **Kyo**: Mix-and-match effects, including typed error handling
- **raise4s/yaes**: For direct-style scala
- `cats.ApplicativeError`: Equivalent to `cats.mtl.Handle` but a bit less ergonomic

Acknowledgments

- **Daniel Spiewak & Thanh Le** for innovation in `cats-mtl` `Raise/Handle`
- IOHandle contributors: **Alex, Dmitryo, Francesco, Pavel, David**
- **Noel Welsh** for reviewing this talk!

Thank you

Here are some relevant links

- Monad Transformer issues with concurrency
- https://typelevel.org/blog/2025/09/02/custom-error-types.html
- https://github.com/jatcwang/iohandle
  - `"com.github.jatcwang" %% "iohandle" % "0.1.0"`

Bonus: Effectful error accumulation?

```
import iohandle.ioscreen.*

def validatePackage(id: String, width: Int, height: Int): IO[Either[BadPackageError, Package]] =
  ioScreen[String]:
    (
      checkWidthAllowedRemotely(width).reportIf(_ == false, "Too wide"),
      checkHeightAllowedRemotely(width).reportIf(_ == false, "Too tall"),
    ).parZip: (width, height) ⇒
      Right(Package(id, width, height))
  .handleErrors: (errors: NonEmptyVector[String]) ⇒
    Left(BadPackageError(id, errors))
```

Bonus: Effectful error accumulation?

```
import iohandle.ioscreen.*

def validatePackage(id: String, width: Int, height: Int): IO[Either[BadPackageError, Package]] =
  ioScreen[String]:
    (
      checkWidthAllowedRemotely(width).reportIf(_ == false, "Too wide"),
      checkHeightAllowedRemotely(width).reportIf(_ == false, "Too tall"),
    ).parZip: (width, height) ⇒
      Right(Package(id, width, height))
  .handleErrors: (errors: NonEmptyVector[String]) ⇒
    Left(BadPackageError(id, errors))
```

Bonus: Effectful error accumulation?

```
import iohandle.ioscreen.*

def validatePackage(id: String, width: Int, height: Int): IO[Either[BadPackageError, Package]] =
  ioScreen[String]:
    (
      checkWidthAllowedRemotely(width).reportIf(_ == false, "Too wide"),
      checkHeightAllowedRemotely(width).reportIf(_ == false, "Too tall"),
    ).parZip: (width, height) ⇒
      Right(Package(id, width, height))
  .handleErrors: (errors: NonEmptyVector[String]) ⇒
    Left(BadPackageError(id, errors))
```

Bonus: Effectful error accumulation?

```
import iohandle.ioscreen.*

def validatePackage(id: String, width: Int, height: Int): IO[Either[BadPackageError, Package]] =
  ioScreen[String]:
    (
      checkWidthAllowedRemotely(width).reportIf(_ == false, "Too wide"),
      checkHeightAllowedRemotely(width).reportIf(_ == false, "Too tall"),
    ).parZip: (width, height) ⇒
      Right(Package(id, width, height))
  .handleErrors: (errors: NonEmptyVector[String]) ⇒
    Left(BadPackageError(id, errors))
```