Scala for Java Developers

a presentation for the

Tampa Java Users Group

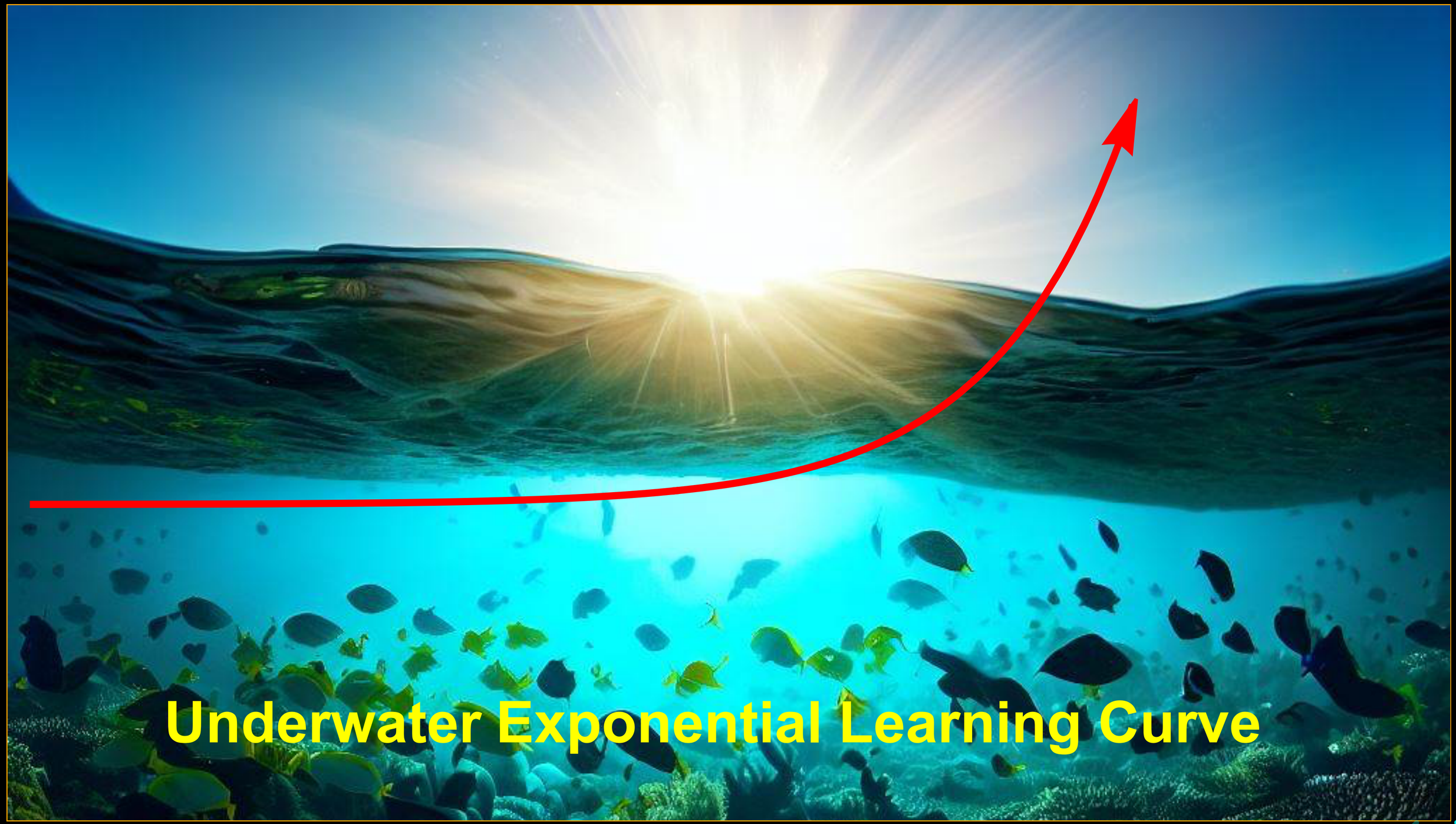
by

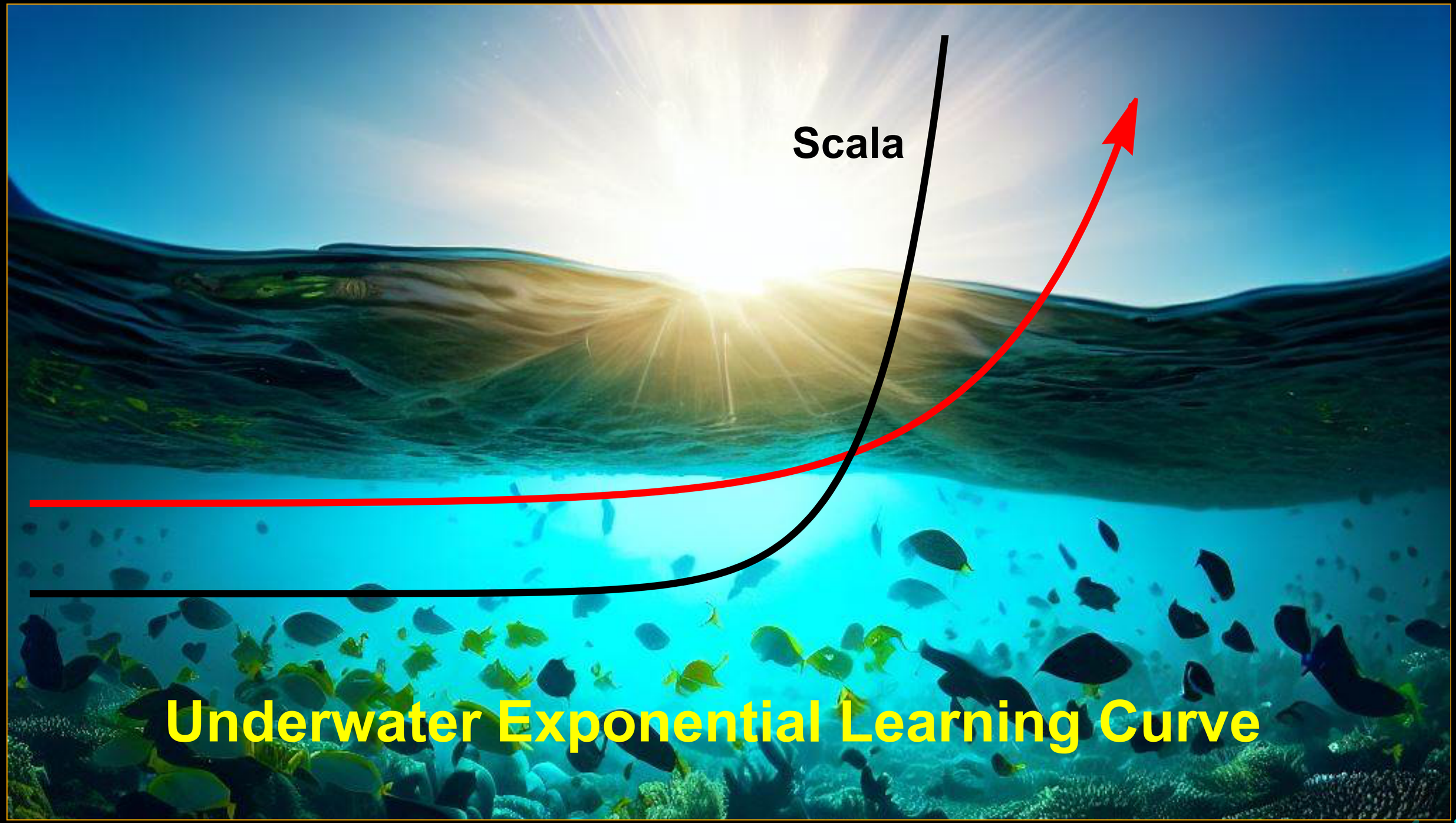
Steve Waldman

2023-05-11 @ KForce (Thanks!)

=> swaldman@mchange.com
=> @interfluidity@fosstodon.org
=> https://github.com/swaldman

Preliminaries


Underwater Exponential Learning Curve


Scala
Underwater Exponential Learning Curve

Scala as bleeding-edge Java

Scala Feature	Eventual Java Feature
lambdas, `map(...)`, etc.	lambdas, `map(...)`, etc.
`scala.Option`	`java.lang.Optional`
`case class`	`record`
pattern matching	enhanced `switch`
`sealed` classes and traits	`sealed` classes and interfaces
ubiquitous type inference	inferred `var` type
concurrency via `Future`	concurrency via `Future`
concrete `trait` methods	`default` methods of `interface`
`_` for unnamed variables	`_` for unnamed variables

Scala as bleeding-edge Java

- *All these features are more idiomatic to and deeply integrated into Scala.*
- *Scala arguably exists because Martin Odersky got frustrated with the slow pace of Java evolution.*
- See *pizza*, the 1997 prototype of generic Java

Why Scala?

Concision

Scala

- encourages programmers to express themselves **cleanly and concisely** with very little boilerplate.

- successfully blends multiple programming-language paradigms (**functional**, **object-oriented**), enabling whichever style best models a particular problem or domain.

- combines **flexible syntax** with extraordinary **type safety**, enabling construction of "dialects" that function as domain-specific languages.

Concision — Primes in Java

```java
public final class Primes {
  public static boolean isPrime(int n) {
    var isPrime = true;
    var max = (int) Math.floor(Math.sqrt(n));
    for (int i = 2; i <= max; ++i) {
      if (n % i == 0) {
        isPrime = false;
        break;
      }
    }
    return isPrime;
  }
}
```

Concision — Primes in Scala

```
def isPrime( n : Int ) =
  !(2 to math.floor(math.sqrt(n)).toInt).exists( n % _ == 0 )
```

or, perhaps more readably...

```
def isPrime( n : Int ) =
  val max = math.floor(math.sqrt(n)).toInt
  !(2 to max).exists( n % _ == 0 )
```

⇒ *I am using Scala 3 syntax!*

Concision — Primes in Java, functional style

```java
import java.util.stream.IntStream;

public final class Primes {
  public static boolean isPrime(int n) {
    var max = (int) Math.floor(Math.sqrt(n));
    return !IntStream.rangeClosed( 2, max )
                     .anyMatch(check -> n % check == 0 );
  }
}
```

⇒ *Pretty close!*

Safety

```
def sendmail(to: String, subject: String, text: String)  = ???
```

Yuck. 🤮

Enforce everything you know at compile-time!

```
def sendmail(to: Email, subject: String, text: String)  = ???
```

⇒ *The ??? function is a wonderful convenience!*

Safety

Elsewhere...

```
opaque type Email = String

class BadEmail(msg: String, cause: Throwable = null) extends Exception(msg, cause)

// modified from https://www.regular-expressions.info/email.html
val EmailRegex = """(?i)^\s*([A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,})\s*$""".r

def toEmail(s: String): Email =
  s match {
    case EmailRegex(trimmed) => trimmed
    case _                   => throw new BadEmail(s)
  }

// toString() will give us back our String
```

⇒ *No boxing or runtime overhead! One-line custom Exception!*

Abstraction

e.g. "tagless final" style — *very optional! love it or hate it!*

```scala
// from Practical FP in Scala, 2nd Ed, by Gabriel Volpe (2021)
final case class Checkout[F[_]: Background: Logger: MonadThrow: Retry](
    payments: PaymentClient[F],
    cart:     ShoppingCart[F],
    orders:   Orders[F],
    policy:   RetryPolicy[F]
){
  def process(userId: UserId, card: Card): F[OrderId] =
    cart.get(userId).flatMap {
      case CartTotal(items, total)  =>
        for {
          its <- ensureNonEmpty(items)
          pid <- processPayment(Payment(userId, total, card))
          oid <- createOrder(userId, pid, its, total)
          _   <- cart.delete(userId).attempt.void
        } yield oid
    }
  def processPayment(in: Payment): F[PaymentId] =
    Retry[F] // calls no-arg apply methods of Retry, finds implicit instance
      .retry(policy, Retriable.Payments)(payments.process(in))
      .adaptError {
        case e => PaymentError( Option(e.getMessage).getOrElse("Unknown") )
      }
  // ...
}
```

Hyperscale

The Scala ecosystem includes libraries for building concurrent, resilient, incredibly scalable services based on transformable "functional effects".

- ZIO
- Cats effects / FS2

See also Tapir, an infrastructure agnostic endpont definition library.

These are libraries are challenging. End up here, but don't start your Scala journey with these tools, unless you are working with a team to mentor you.

Simplicity

- Scala's best kept secret is how beautifully you can accomplish simple things very simply, while getting all the benefits of strong type safety.

- Scala's built-in collections library is a superpower.

Simplicity

Li Haoyi writes beautiful libraries that emphasize simplicity. Some examples I enjoy include

- os-lib for simple file-system and process handling
- requests-scala, a simple blocking HTTP client
- upickle, simple typesafe JSON processing
- utest, beautifully concise testing
- cask, HTTP services without ceremony

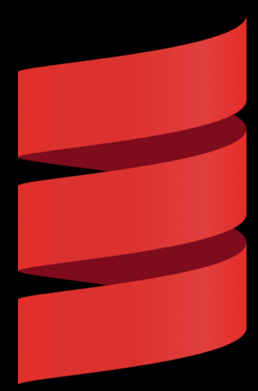
Check out all of his libraries here.

Haoyi is also the author of the mill build tool, which like all build tools in Scala is a Swiss Army knife with which you do much more than build.

Haoyi's book *Hands-On Scala Programming* is a great introduction for programmers interested in getting stuff done fast.

Compatible

- The **full range of Java libraries** is accessible from Scala code
- You can **write Java-compatible facades** to make your Scala libs available to Java and other JVM language clients.



Just effing interesting

Scala is...

- a kitchen sink / mad scientists' lab of programming language ideas

- a metaprogramming language as much as programming language

Commune with the compiler.

paque types • top-level declarations • inline, macros, and metaprogramming • flexible, concise syntax • multiplatform: Scala JVM, Scala JS, Scala native • rich collections library • typeclasses / ad hoc polymorphism • `apply(...)` method makes anything callable • rich regex support • for comprehensions, convenient monad sequencing • strong immutability preference • by-name params + multiple arg lists => user-defined control flow constructs • infix function notation => operator overloading • `String` interpolation • `case class` • strong expressive typing with convenient type inference • extension methods • `if/then` and codeblocks as expressions • multiline `String` • pattern matching with deep destructuring • higher kinded types • `Nothing` as type hierarchy bottom • singleton `object`, much saner than `static` • opaque types • top-level declarations • inline, macros, and metaprogramming • flexible, concise syntax • multiplatform: Scala JVM, Scala JS, Scala native • rich collections library • typeclasses / ad hoc polymorphism • `apply(...)` method makes anything callable • rich regex support • for comprehensions, convenient monad sequencing • strong immutability preference • by-name params + multiple arg lists => user-defined control flow constructs • infix

Word Salad

scala-cli for Java programmers

scala-cli

- a shell command, **minimal-config build tool** in the style of golang's multifarious `go` command.
- will soon become the default `scala` command
  - replaces existing command modeled on the `java` command
- supports **compiling**, **running**, **packaging**, and **publishing** libraries and applications, and running a **REPL**
- supports `graalvm` native image generation
- supports **scripting in Scala**
  - `#!/usr/bin/env -S scala-cli shebang`

scala-cli

- supports **Java** code *as well and easily* as **Scala** code!
  - On the Java side, `jbang` is the closest competition, but `scala-cli` is in some respects easier to use, more "batteries included"
  - and of course, `scala-cli` supports Scala too

- Consider prototyping your Java projects with `scala-cli`
  - https://scala-cli.virtuslab.org/
  - maybe mix in a bit of Scala!

Demos

Demos

1. `HelloWorld.java` in `scala-cli`
   - output to native
2. `HelloPDF.java` in `scala-cli`
   - let's use Apache PDFBox, dependency resolution is so easy!
   - output to JVM-dependent package
3. `HelloWorld.scala`
   - cool regex handling
4. `poemalyzer`
   - hit a JSON API with requests-scala to grab a random poem
   - deserialize result to `case class` with upickle
   - build a sorted word frequency table with Scala's Collections API
   - print a nice report

Demos — cheatsheet

- Library coordinates
  - `org.apache.pdfbox:pdfbox:2.0.28` ← Java
  - `com.lihaoyi::upickle:3.1.0` ← Scala
  - `com.lihaoyi::requests:0.8.0` ← Scala
  - understand the tricky double colon for Scala libs!
- PDFBox Hello World to steal
  - `https://pdfbox.apache.org/1.8/cookbook/documentcreation.html`
- requests breadcrumbs
  - `requests.get(...).text()`
- upickle breadcrumbs
  - `upickle.default.{ReadWriter, read}`
- PoetryDB API random poem endpoint
  - `https://poetrydb.org/random`

Epilogue

Learn Scala

Quickstart:

- Free *Scala at Light Speed* two-hour video course
  rockthejvm.com

Recommended books:

- *Hands on Scala Programming* by Li Haoyi
  https://lihaoyi.gumroad.com/l/DNJPR
- *Programming Scala, 3ed* by Dean Wampler
  https://www.oreilly.com/library/view/programming-scala-3rd/9781492077886/
- *Programming in Scala, 5ed* by Odersky et al
  https://www.artima.com/shop/programming_in_scala_5ed

Thank You!

Colophon

Tools

- reveal.js — web presentation library
- unstatic — Scala static-site generator
- untemplate — Scala templating library

Images

- Cover image, weird server farm — *Midjourney*
- Over-under sunny ocean split shot — *Bing Image Generator*
- Goofy Superhero — *Bing Image Generator*
- Steampunk mad scientist's lab — *Bing Image Generator*
- Headshots of Martin Odersky and Li Hayoi scraped from online profile pics

Fonts

- Raleway
- Inconsolata

You can find the build of this presentation here.