



learning Scalaz
eugene yokota (@eed3si9n)

about eugene

- nights-and-weekend Scala programmer
- scalaxb, treehugger, scopt
- sbt-(assembly, buildinfo, etc.)
- contributor to n8han/pamflet, etc.
- translator of docs.scala-lang.org

My first Scala code. Tetris of course
<http://is.gd/PXs8> (Happy 25th bd
Tetris)

Reply Delete ★

4:02 AM - 6 Jun 2009

renaming my fork of schema2src to
#scalaxb to
<http://is.gd/>



eugene yokota @eed3si9n · 25 Jul 2010
settled on writing plain HttpServlet in **scala** 2.8 to use on **appengine**
also giving netbeans a try again.



Nathan Han @n8han



Following

**Started Scala code in 2009.
By 2010, started scalaxb,
joined ny-scala.**



eugene yokota
@eed3si9n

i'm going. RT



Jason Zaugg
@retronym

I'm presenting **#scalaz** the New York
#scala meetup on 21/9. You are
hereby cordially invited!

@eed3si9n You should try this thing,
it's better:
n8han/unfiltered

More

Hill, New York

A blurry, out-of-focus photograph of two people. On the left, a man with dark hair and a white shirt is looking towards the right. On the right, a woman with long dark hair and glasses is looking towards the left. They appear to be at a social event or party.

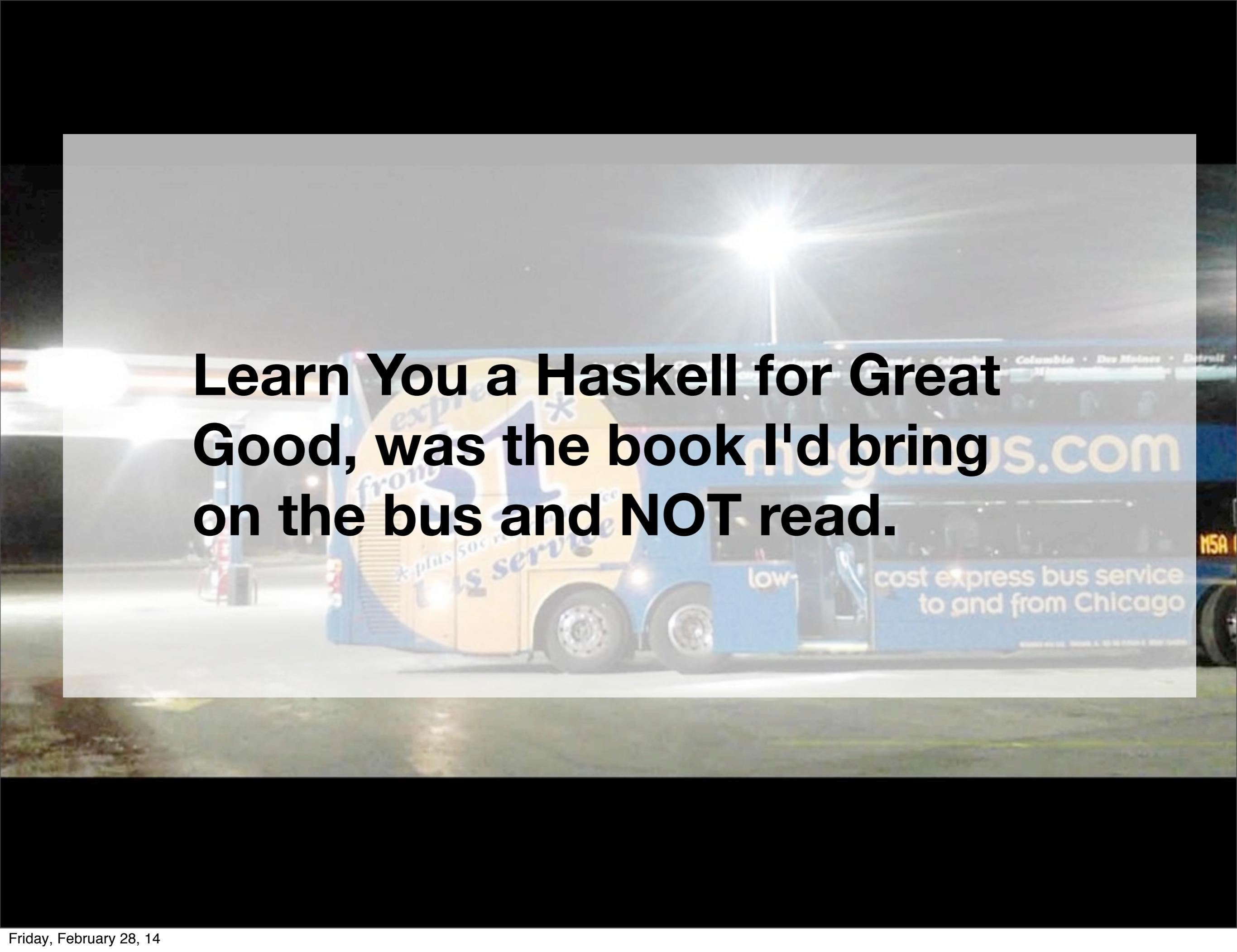
**twitter and Meetup
changed my life**

A stack of several red nametags. The top nametag clearly shows the word "HELLO" in large letters, with "my name is" written below it. Other nametags in the stack are partially visible, showing the same text.

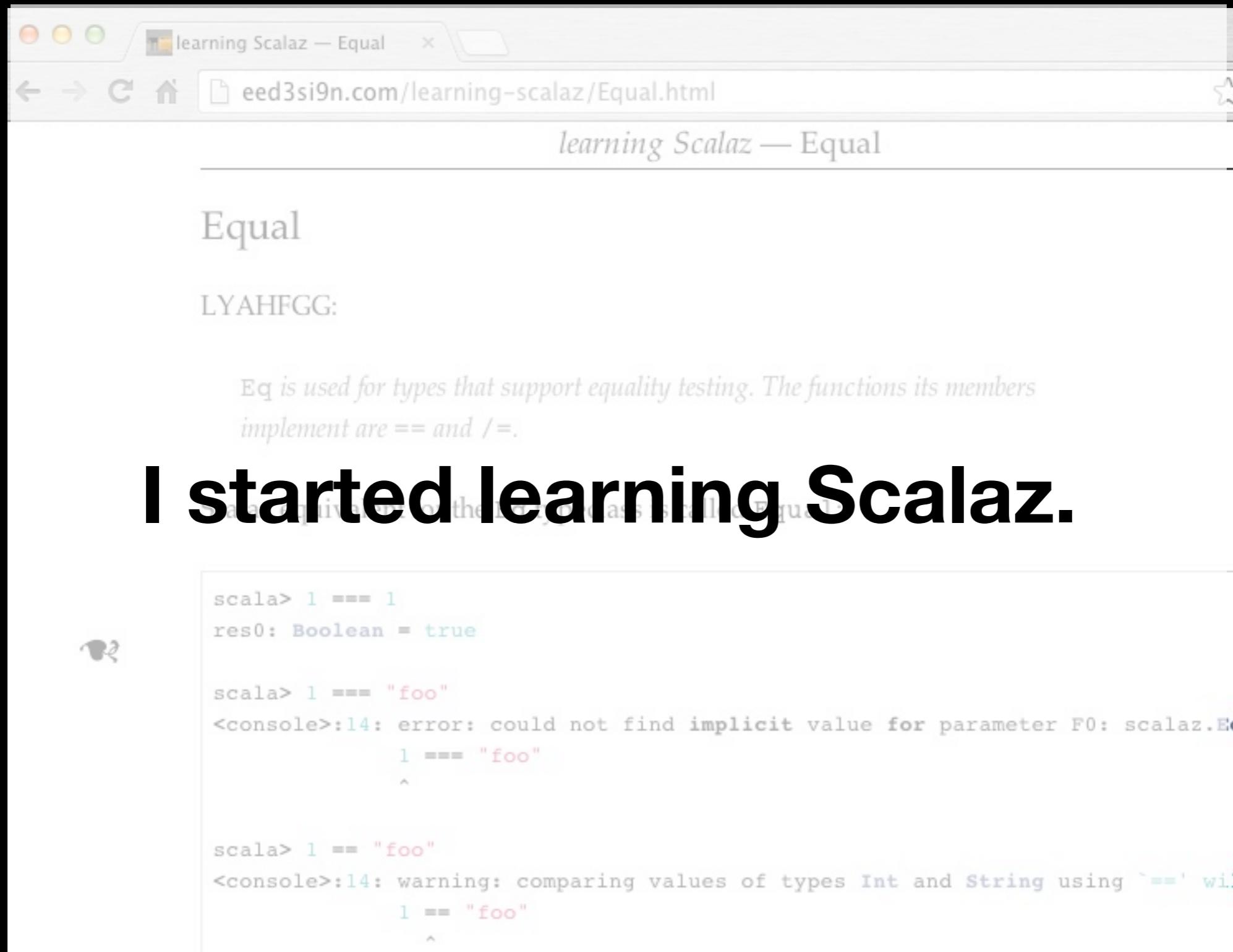
**Joined more meetup groups.
tinkered with Haskell and Lisp.**

A photograph of a person sitting at a desk in a room. The person is wearing a yellow and black patterned shirt and is looking down at a laptop. On the desk, there is a white computer monitor displaying a dark screen, a keyboard, and some papers. Behind the person, there is a window with white horizontal blinds. The room has green walls and a wooden floor.

**Then I moved to Amherst,
Massachusetts in 2011.**



**Learn You a Haskell for Great
Good, was the book I'd bring
on the bus and NOT read.**



A screenshot of a web browser window titled "learning Scalaz — Equal". The URL in the address bar is "eed3si9n.com/learning-scalaz/Equal.html". The page content includes a heading "Equal", a section titled "LYAHFGG:", and a large bold heading "I started learning Scalaz.". Below the heading is a Scala REPL session showing type safety issues with equality operators.

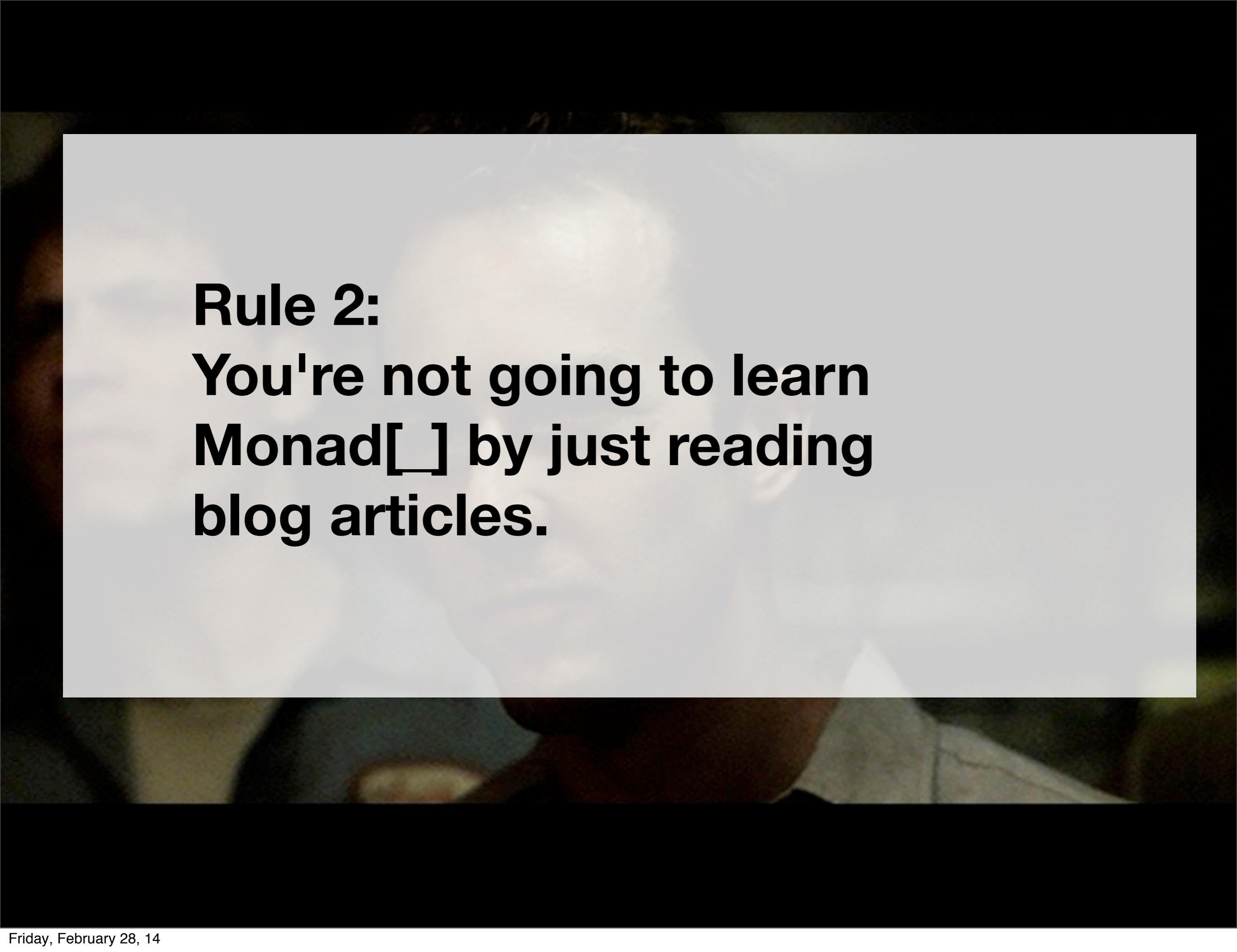
```
scala> 1 === 1
res0: Boolean = true

scala> 1 === "foo"
<console>:14: error: could not find implicit value for parameter F0: scalaz.Eq[Int]
          1 === "foo"
          ^
          ^

scala> 1 == "foo"
<console>:14: warning: comparing values of types Int and String using `==` will
          1 == "foo"
          ^
```



Rule 1:
You're not going to learn
Scalaz by just reading my blog
articles.



Rule 2:
You're not going to learn
Monad[] by just reading
blog articles.

Get your hands dirty

- coding exercise
- try it on REPL
- write blog
- social feedback

A black and white photograph of a man with glasses, wearing a light-colored shirt, sitting at a desk and reading a book. He is holding a magnifying glass over the book. In the background, there are more books and papers, suggesting a library or study environment.

Rule 3:
Buy books.

A comprehensive step-by-step guide

Programming in
Scala

Second Edition



Updated for Scala 2.8

artima

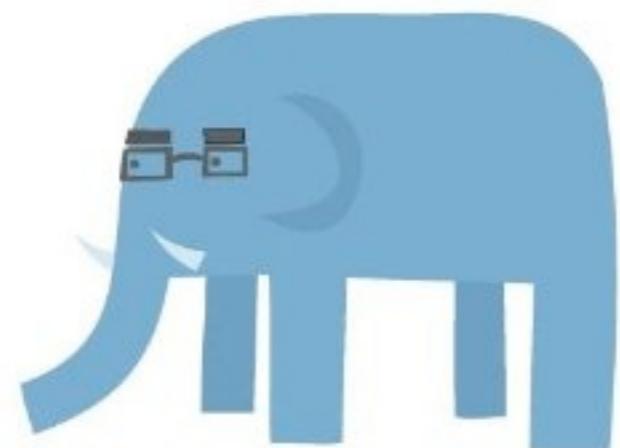
Martin Odersky
Lex Spoon
Bill Venners

Programming in Scala, 2nd ed Odersky, Spoon, Venners

Copyrighted Material

Learn You a Haskell for Great Good!

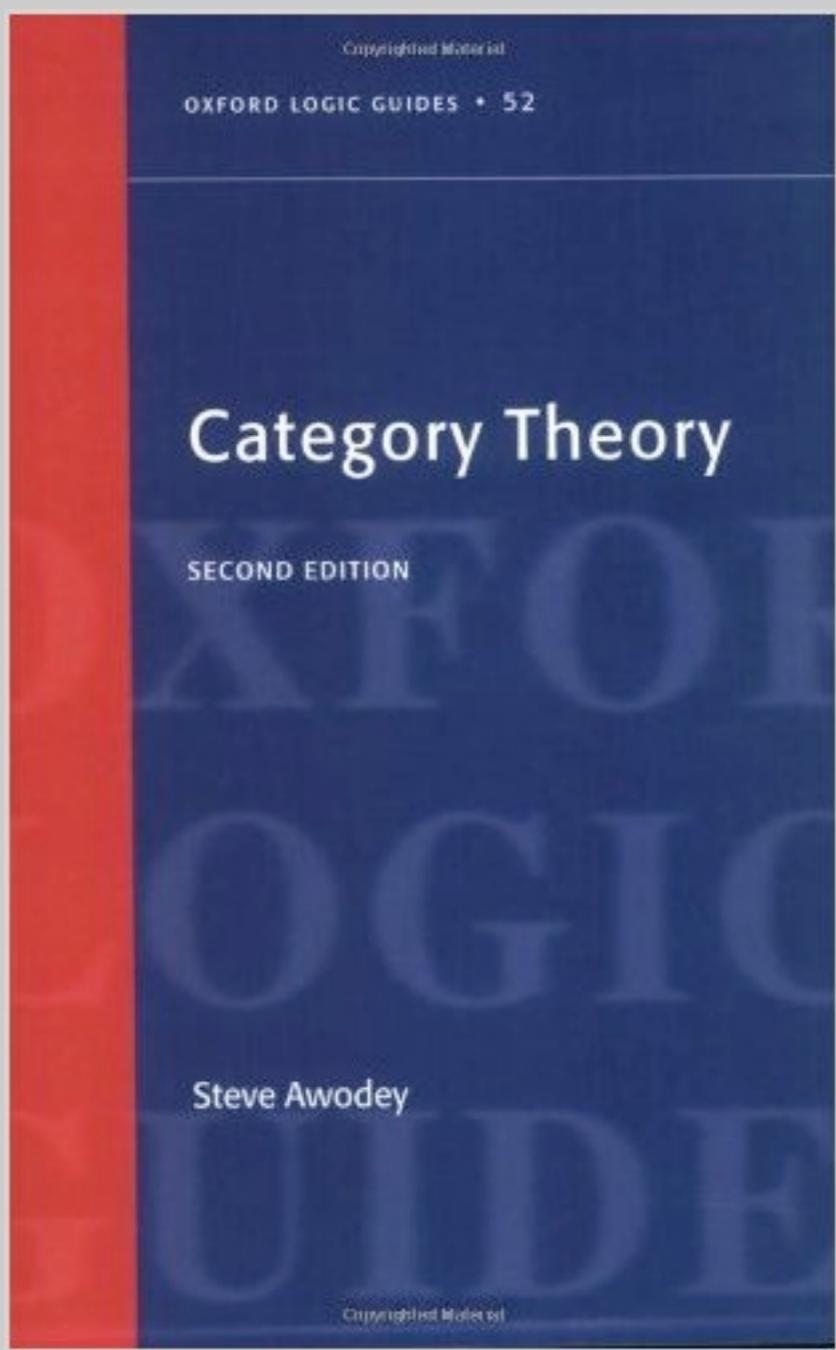
A Beginner's Guide



Miran Lipovača

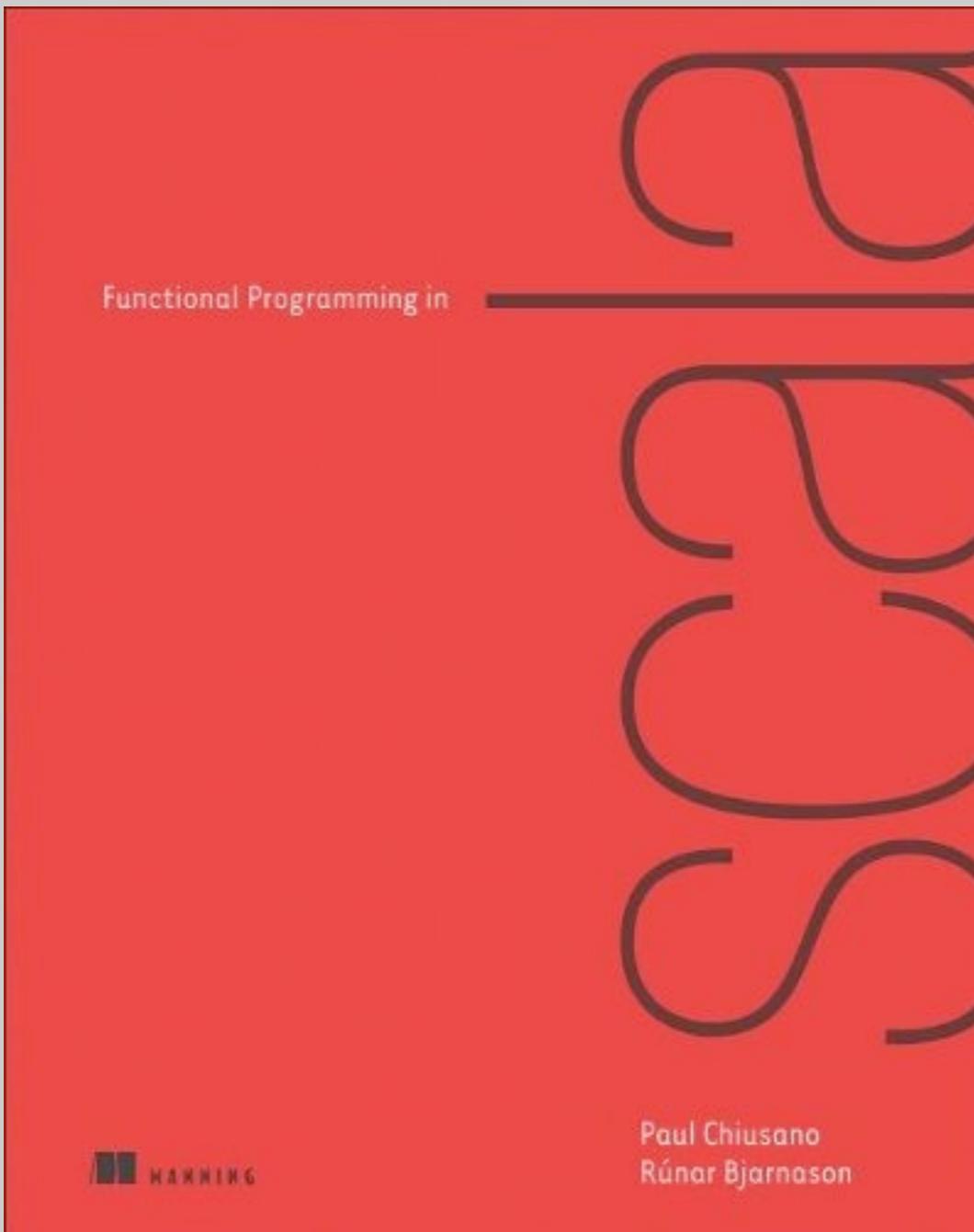
Copyrighted Material

Learn You a
Haskell for Great
Good!
Lipovača

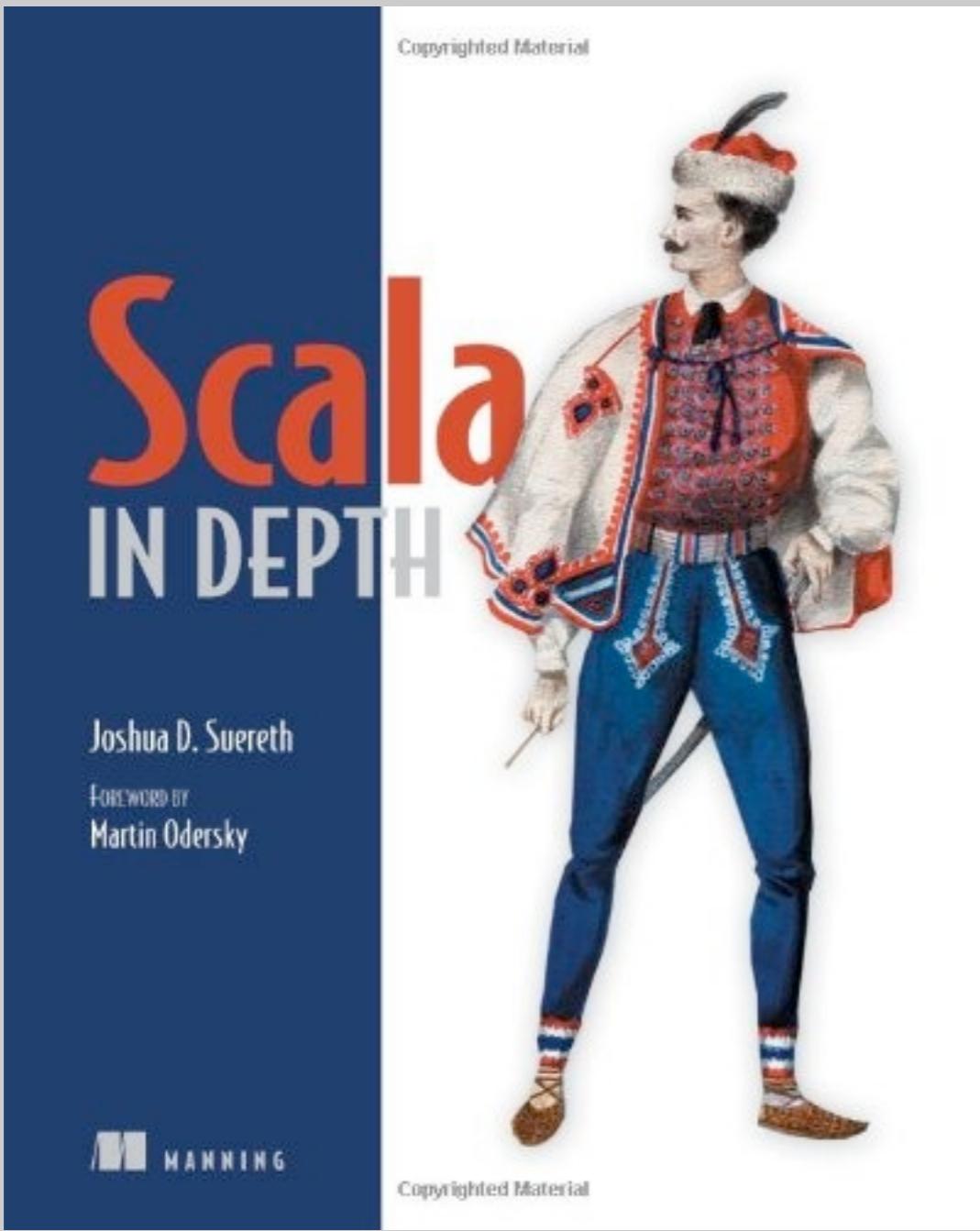


Category Theory

Awodey

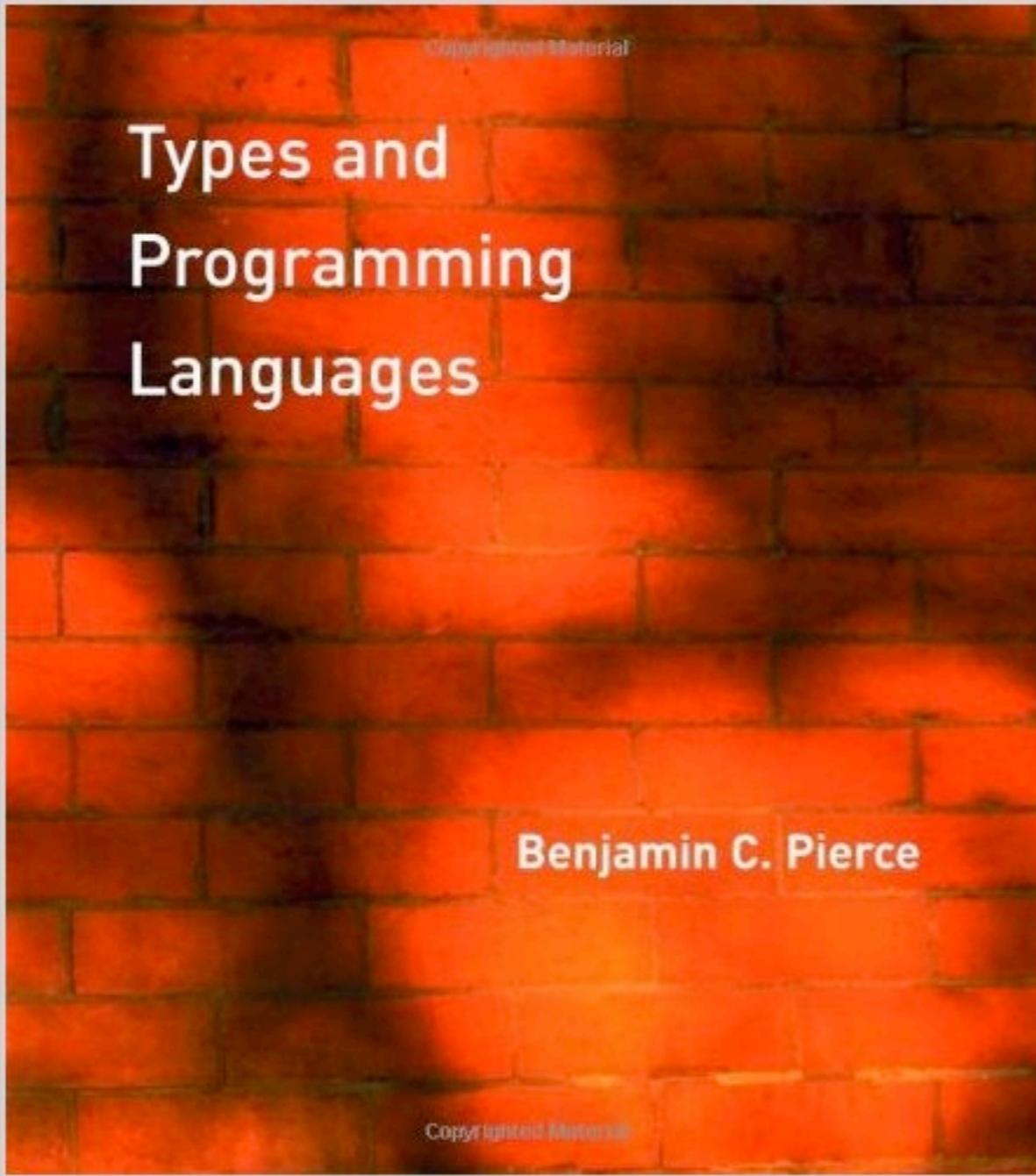


Functional Programming in Scala Chiusano and Rúnar

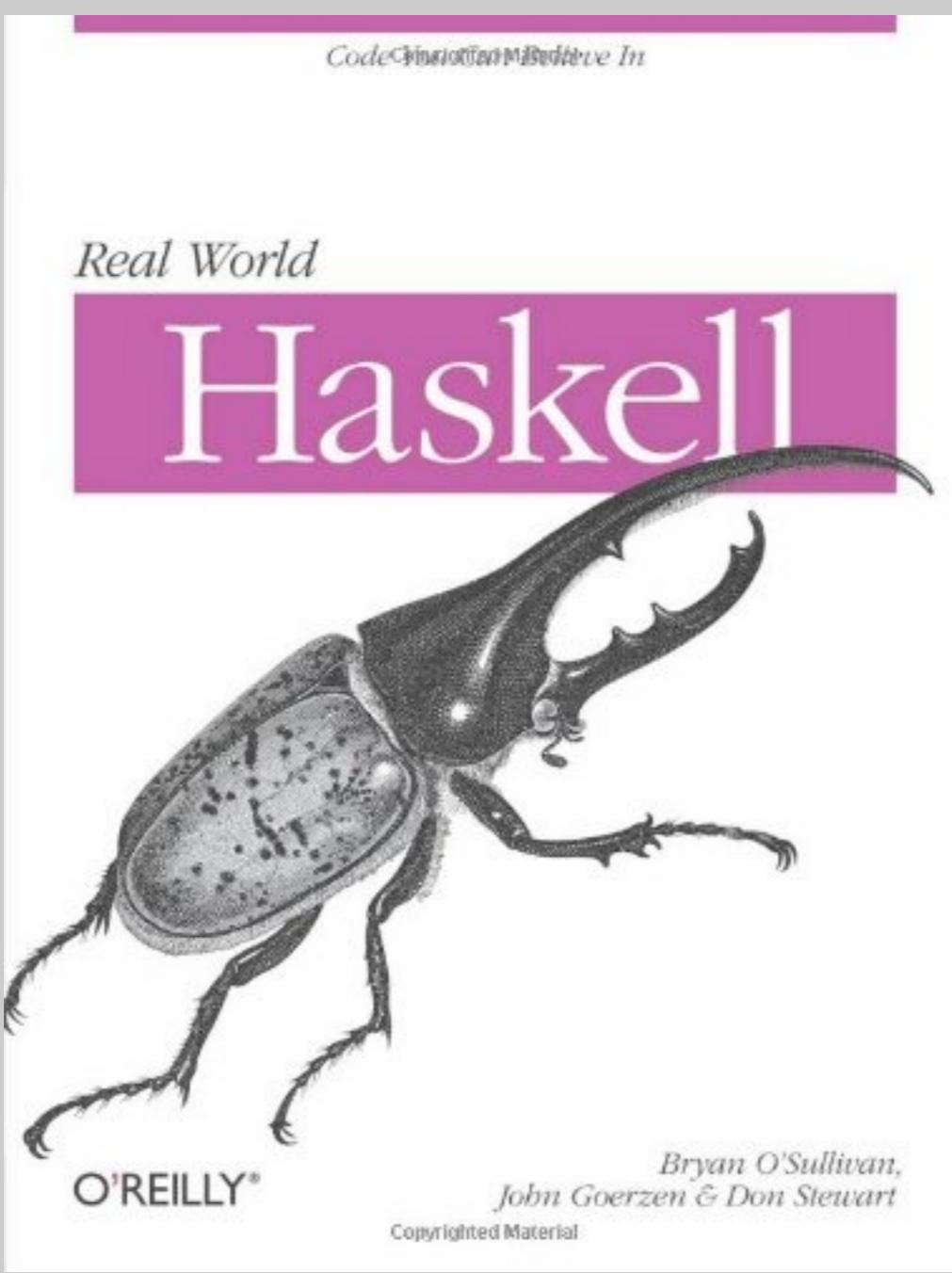


Scala in Depth

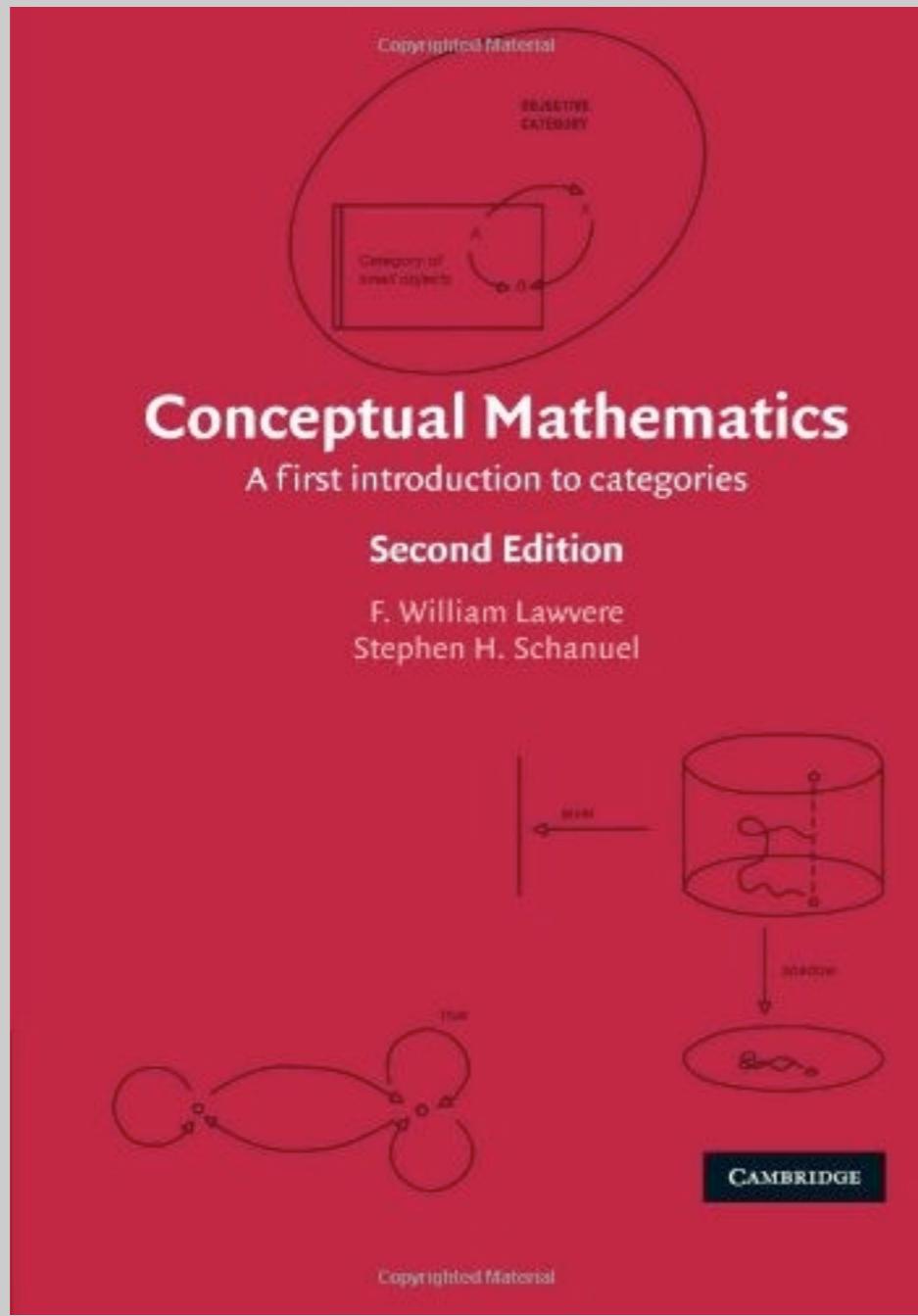
Suereth



Types and Programming Languages Pierce

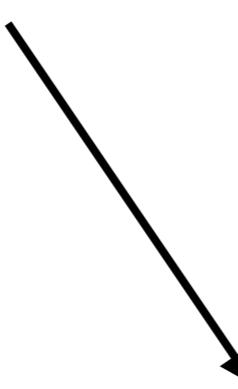
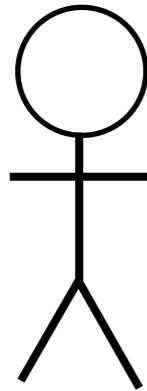


Real World Haskell O'Sullivan, Goerzen, Stewart

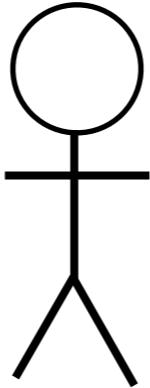


Conceptual Mathematics, 2nd ed Lawvere, Schanuel

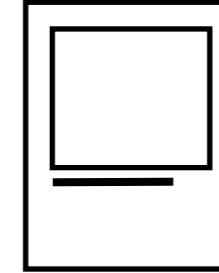
Problem Domain



Model

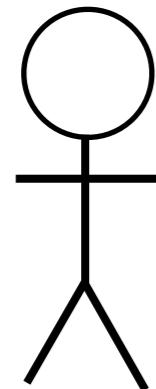


Computer



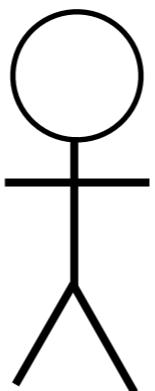
Problem

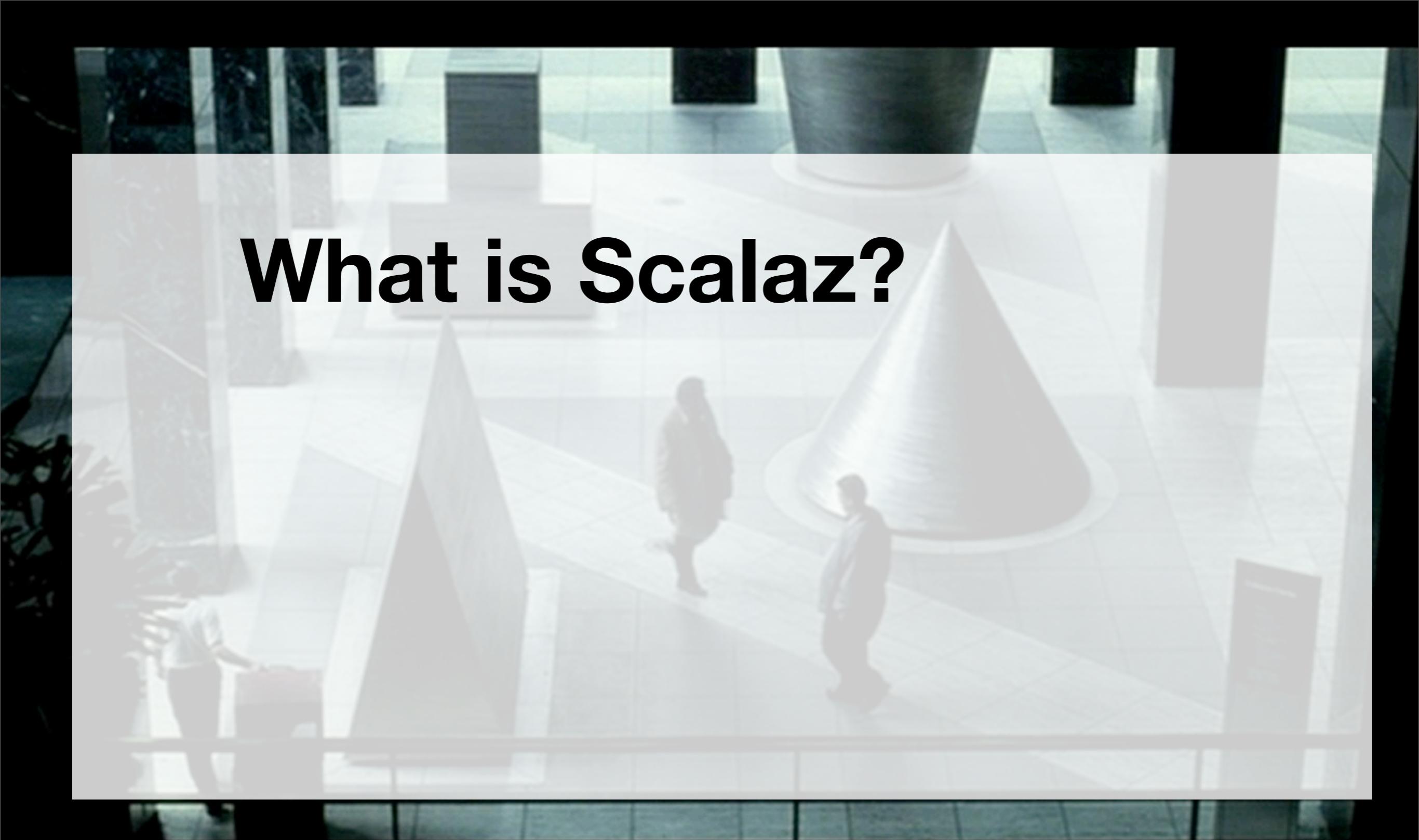
Domain



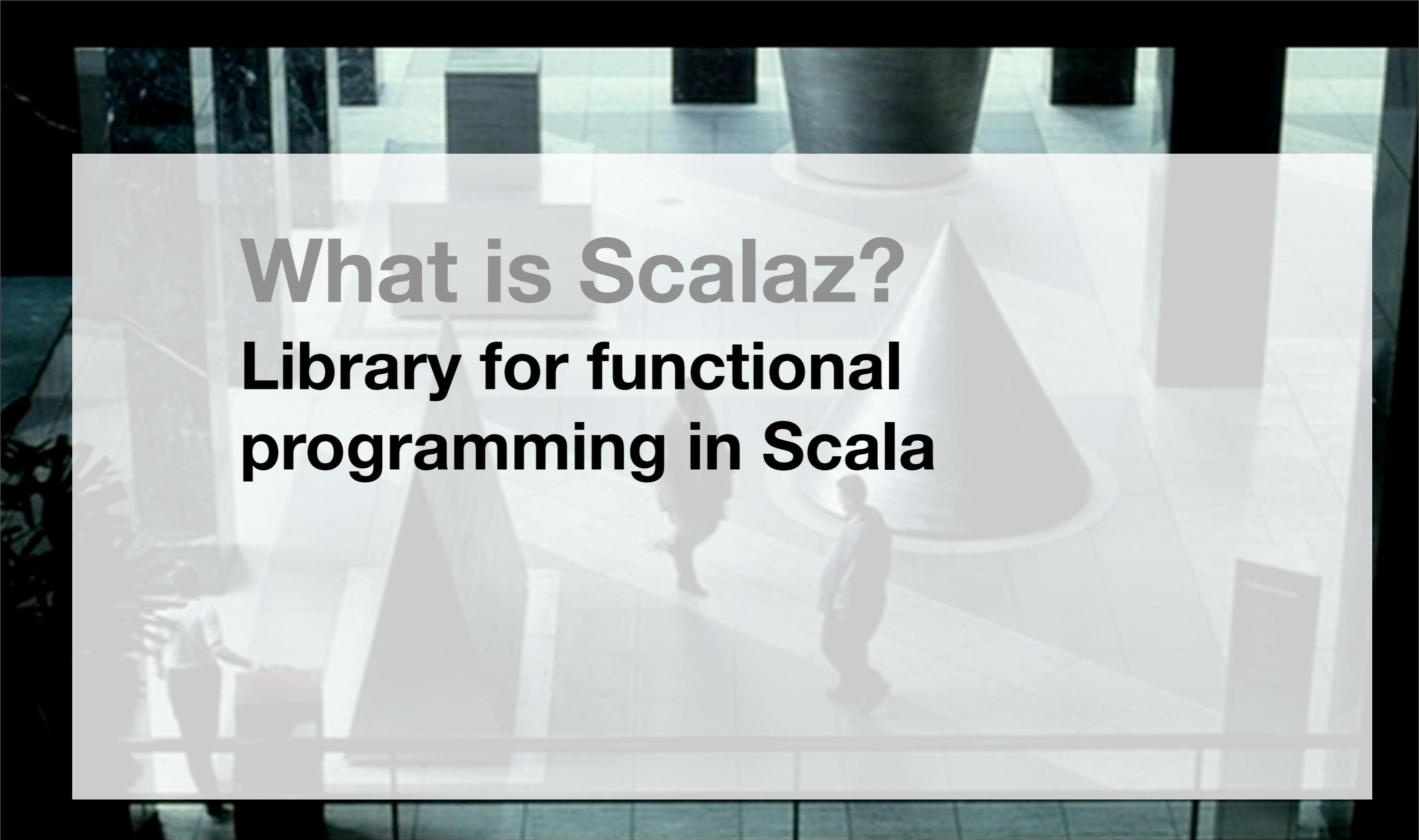
Operators

Computer



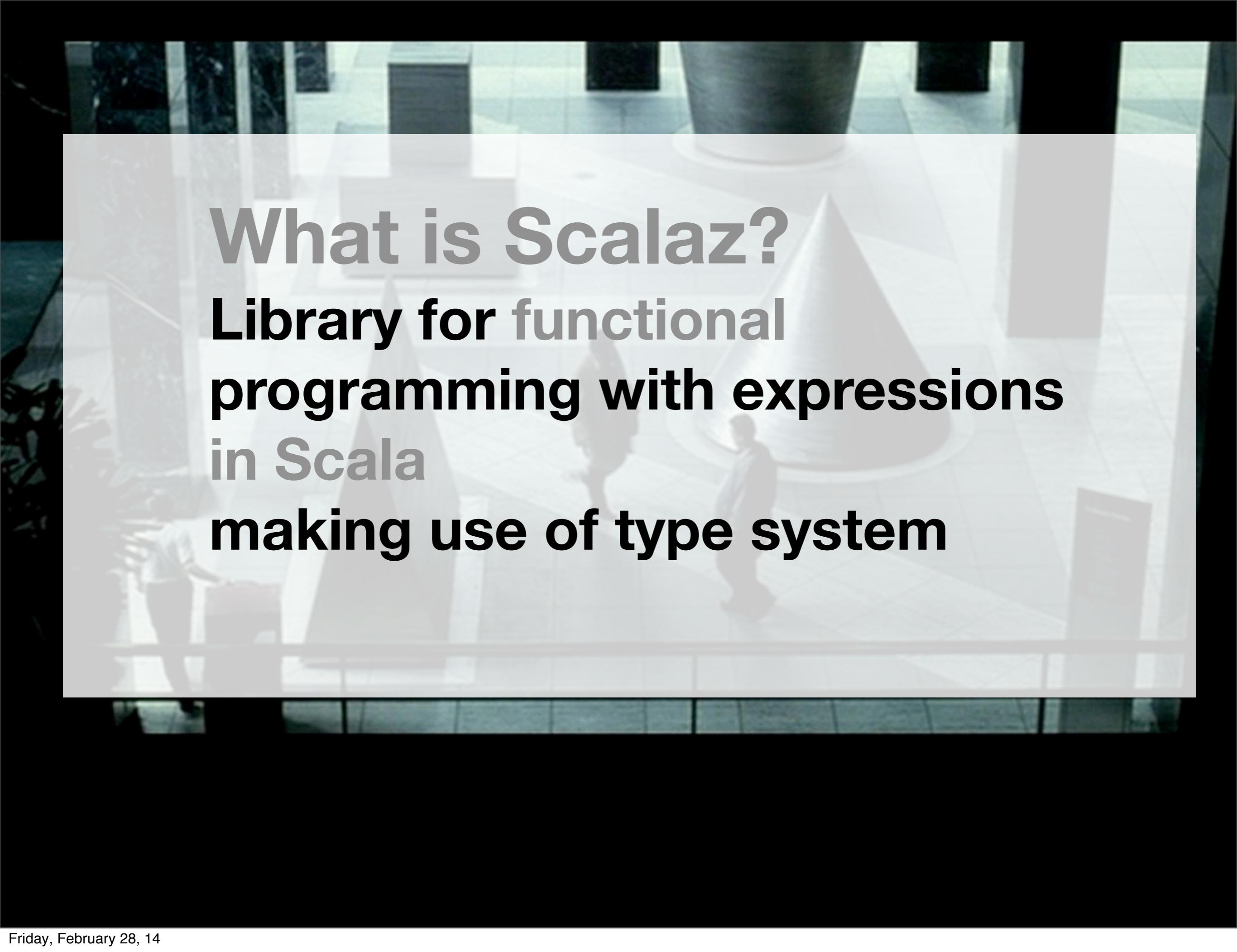


What is Scalaz?



What is Scalaz?

Library for functional programming in Scala



What is Scalaz?
Library for functional
programming with expressions
in Scala
making use of type system

Soundness

- Argument is valid

$$P \rightarrow Q$$

- All premise is true

$$P \sim \text{true}$$

Soundness

A *type system* is a syntactic method for automatically checking the absence of certain erroneous behaviors...

Soundness?

```
scala> "1" == 1  
res0: Boolean = false
```

This should *not* compile

Equality

```
trait Equal[A] {  
    def equal(a1: A, a2: A):  
        Boolean  
}
```

Oxygen Supply

Equality

```
scala> import scalaz._,  
           Scalaz._  
scala> "1" === 1  
<console>:20: error: could not  
implicit value for parameter F0  
scalaz.Equal[Any]  
          "1" === 1  
          ^
```

Expression Problem

Given implementation of Int
addition: $a + b$

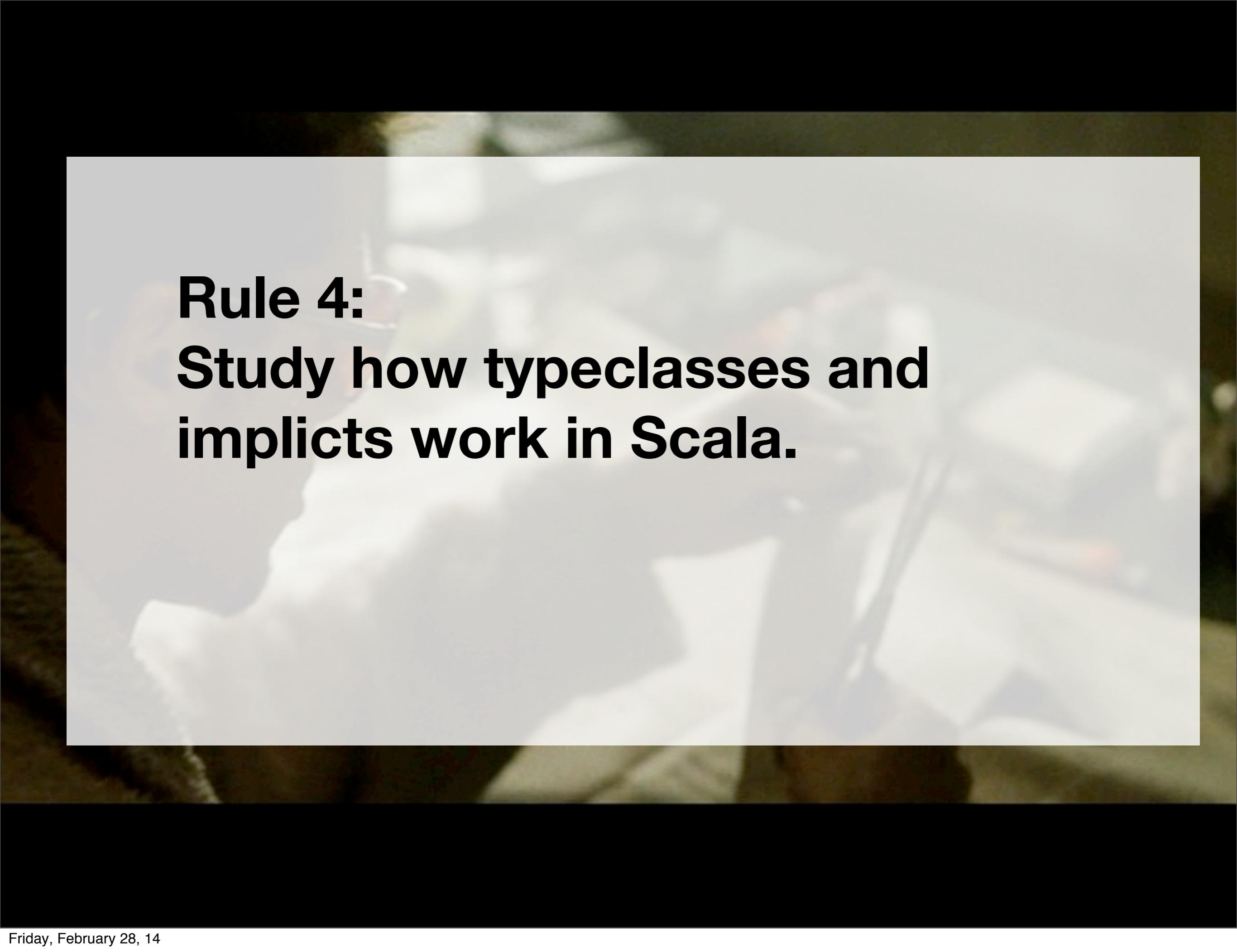
- Add multiplication of Int: $a * b$
- Add addition of Double: $c + d$

without recompilation or casting.

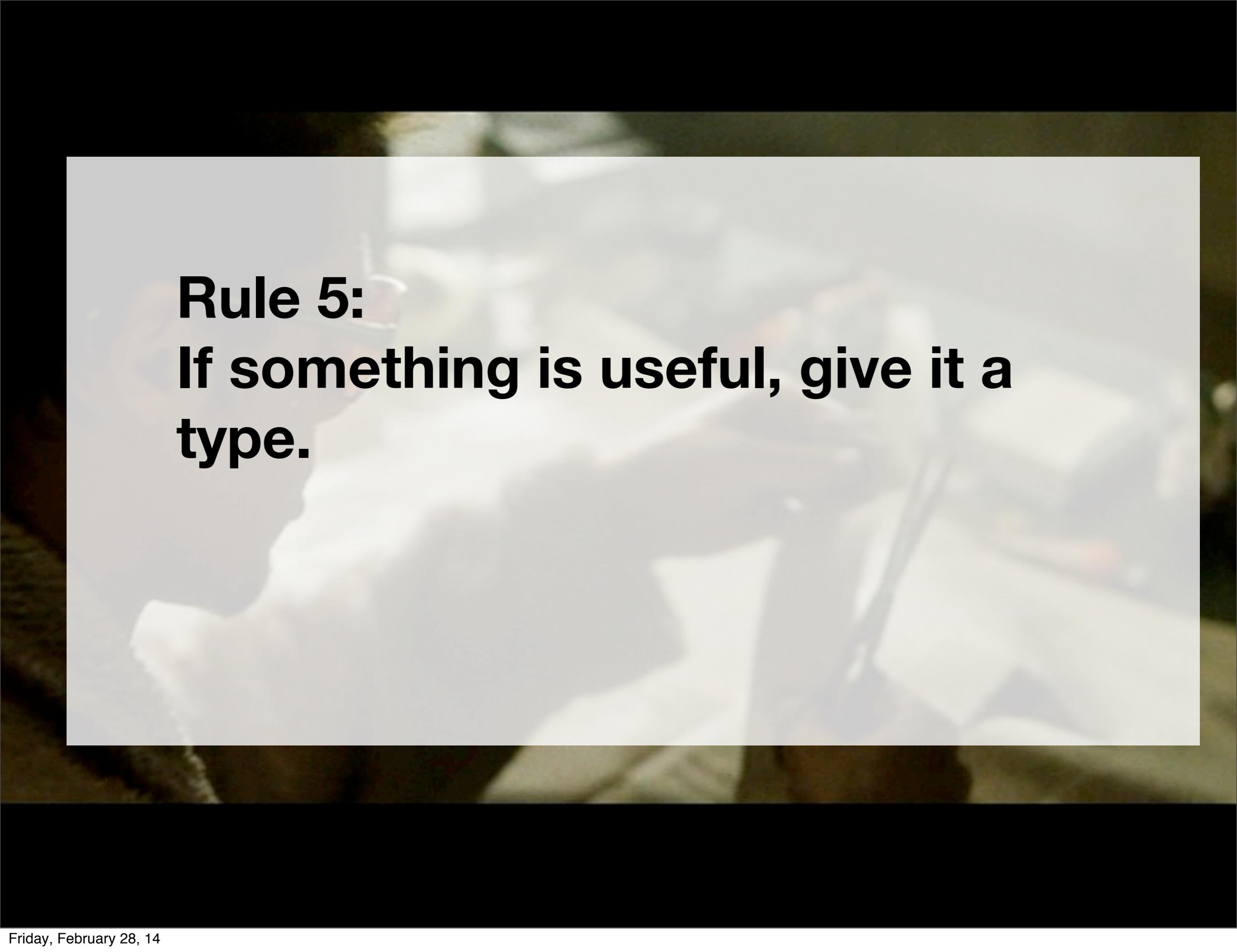
Expression Problem

- Add multiplication of Int: $a * b$
- Add addition of Double: $c + d$

Statically typed plugin system

A person is sitting in a chair, seen from behind, looking at a computer screen. The person is wearing a light-colored shirt and dark trousers. The background is a blurred indoor setting.

Rule 4:
Study how typeclasses and
implicits work in Scala.

A close-up photograph of a person's face in profile, facing right. The person has short, light-colored hair and is wearing a dark, possibly black, garment. The background is blurred, showing what appears to be a natural setting with greenery and possibly a body of water.

Rule 5:
If something is useful, give it a
type.

Parametric Polymorphism

```
def head[A] (xs: List[A]): A  
= xs(0)
```

```
scala> head(1 :: 2 :: Nil)  
res0: Int = 1
```

Subtype Polymorphism

```
class Foo extends Equal[Foo] {  
    def equal(a2: Foo): Foo  
        = ???  
}
```

Ad-hoc Polymorphism

```
trait Equal[A] {  
    def equal(a1: A, a2: A): A  
}  
  
def equal[A: Equal](a1: A,  
a2: A): A =  
    implicitly[Equal[A]].  
    equal(a1, a2)
```

Typeclass contract

```
trait Equal[A] {  
    def equal(a1: A, a2: A): A  
}
```

```
def equal[A: Equal](a1: A,  
a2: A): A =  
    implicitly[Equal[A]]  
    .equal(a1, a2)
```

Defining typeclass instance

```
implicit val fooEqual: Equal[Foo]  
= Equal.equal { _ == _ }  
// = Equal.equalA
```

```
implicit val fooEqual: Equal[Foo]  
= Equal.equal { _ == _ }  
// = Equal.equalA
```

Using typeclass instance

```
import val fooEqual: Equal[Foo]  
= Equal.equal { _ == _ }  
// = Equal.equalA
```

```
def equal[A: Equal](a1: A,  
a2: A): A =  
implicitly[Equal[A]].  
equal(a1, a2)
```

Stop stereotyping

Inheritance ~ "is born as"

Typeclass ~ "is capable of"

```
def equal[A: Equal](a1: A,  
a2: A): A =  
  implicitly[Equal[A]].  
  equal(a1, a2)
```

Scalaz consists of:

- 1. Typeclasses**
 - a. Instances**
 - b. Syntax (ops)**
- 2. Functional data structures**
- 3. Syntax (ops) on standard classes**

Extracting a typeclass

```
def sum(xs: List[Int]): Int =  
  xs.foldLeft(0) { _ + _ }
```

```
scala> sum(List(1, 2, 3, 4))  
res3: Int = 10
```

Extracting a typeclass

```
def sum(xs: List[Int]): Int =  
  xs.foldLeft(0) { _ + _ }  
  
val intAddition extends Monoid[Int]  
  def zero: Int = 0  
  def append(a: Int, b: Int)  
    : Int = a + b  
}
```

Extracting a typeclass

```
def sum[A: Monoid]
  (xs: List[A]): A = {
  val m = implicitly[Monoid[A]]
  xs.foldLeft(m.zero)
  { m.append(_, _) }
}
```

We have abstracted out (Int, +).

Extracting a typeclass

```
scala> sum(List(1, 2, 3, 4))  
res3: Int = 10
```

```
scala> sum(List("1", "2", "3"))  
res4: String = 123
```

```
scala>  
sum(List(Tags.Disjunction(true),  
Tags.Disjunction(false)))  
res5: scalaz.@@[...] = true
```

Ad-hoc polymorphism + Method injection (Enrich-my)

```
scala> mzero[Int] mappend 1  
res16: Int = 1
```

```
scala> Ø[Int] |+| 1  
res17: Int = 1
```

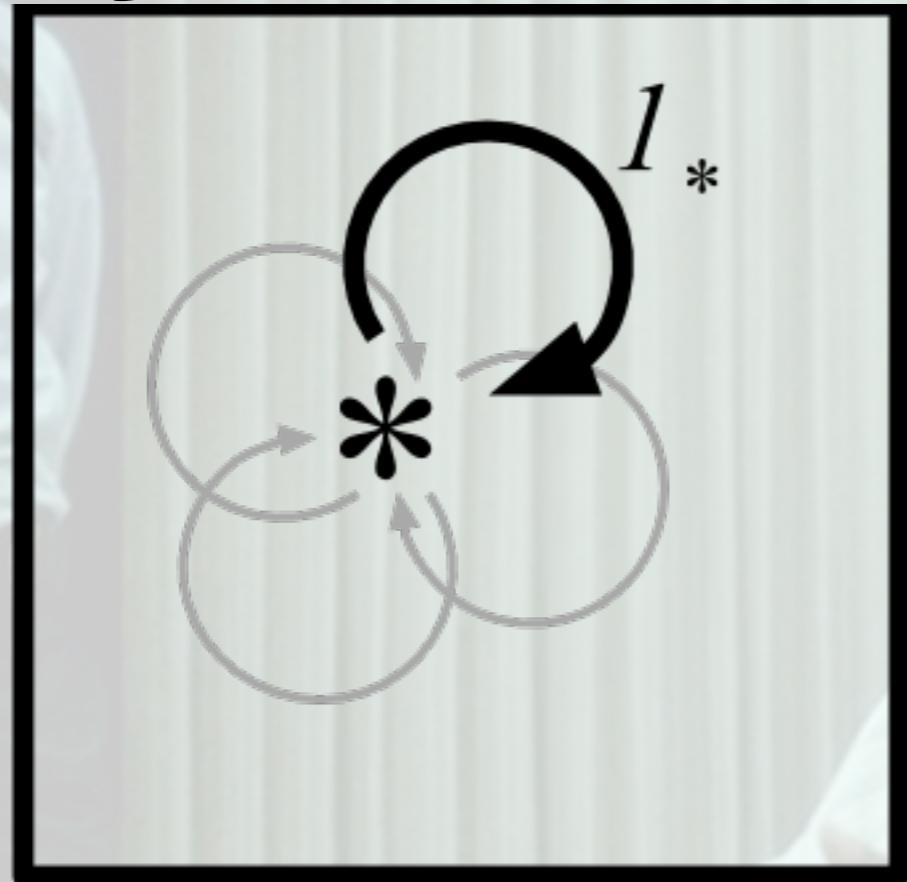
```
scala> Ø[String] |+| "x"  
res18: String = x
```

Monoid Laws

- Left Identity
- Right Identity
- Associativity



Monoid ~ Category with one object



C

- $\text{zero} \mid+ \mid x = x$
- $x \mid+ \mid \text{zero} = x$
- $(x \mid+ \mid y) \mid+ \mid z =$
 $x \mid+ \mid (y \mid+ \mid z)$

Syntax for Options

`(1 < 10) option 1`

`==== 1.some`

`1.some`

`==== (Some(1): Option[Int])`

`none[Int]`

`==== (None: Option[Int])`

`1.some? 'x' | 'y' === 'x'`

`1.some | 2`

`==== 1 // getOrElse`

Validation

```
( "first ok".successNel[String] |@|
  "last failed!".failureNel[String] |@|
  "password failed!".failureNel[String]) {
  _ + _ + _
res25: scalaz.Unapply... =
Failure(NonEmptyList(last failed!,
  password failed!))
```

Memo

```
val memoizedFib: Int => Int =  
  Memo.mutableHashMapMemo {  
    case 0 => 0  
    case 1 => 1  
    case n => memoizedFib(n - 2) +  
               memoizedFib(n - 1)  
  }
```

Tagged type

```
sealed trait KiloGram
def KiloGram[A] (a: A) :
  A @@ KiloGram =  
  Tag[A, KiloGram](a)
def f[A] (mass: A @@ KiloGram) :
  A @@ KiloGram = ???
```

Tagged type

```
sealed trait KiloGram
def KiloGram[A] (a: A) :
  A @@ KiloGram =  
  Tag[A, KiloGram](a)
def f[A] (mass: A @@ KiloGram) :
  A @@ KiloGram = ???
```

State

```
(for {
    xs <- get[List[Int]]
    _   <- put(xs.tail)
} yield xs.head).run(1 :: Nil)
assert_ === (Nil, 1)
```

**Biggest Win:
Changes the way you look at the
world programmatically.**



<http://eed3si9n.com/>