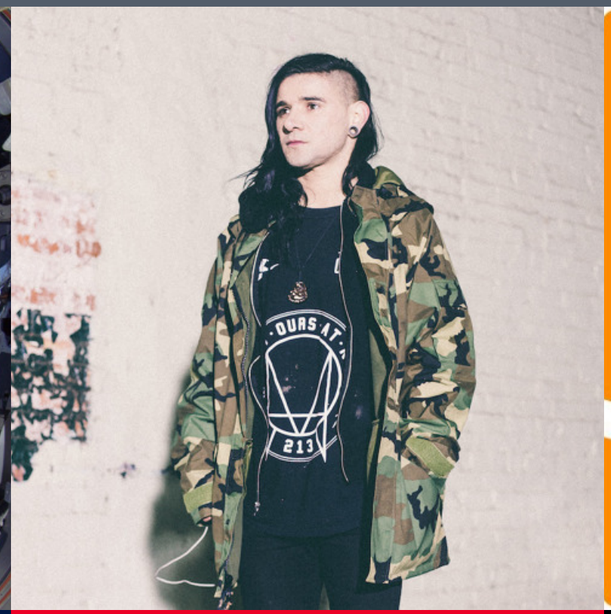
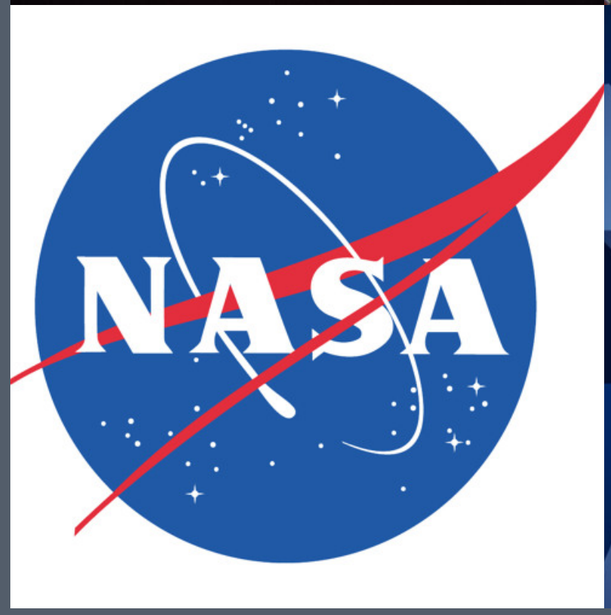
My name is @folone

https://github.com/folone/scalaua


SOUNDCLOUD


AC/DC
ROCK OR BUST




NASA


Haskell Cast


LED ZEPPELIN


#300K

→ 12 hours uploaded every minute

→ ~35k listening years every month

→ >125M tracks (including content from majors: Sony/Universal/Warner)

→ ~170M monthly active users

Search for topics

Groups

POST REPLY

Data Center & Network Engineering ›

Maintenance postponed by rapper diss war. :-/

3 posts by 1 author

g+1

ryan

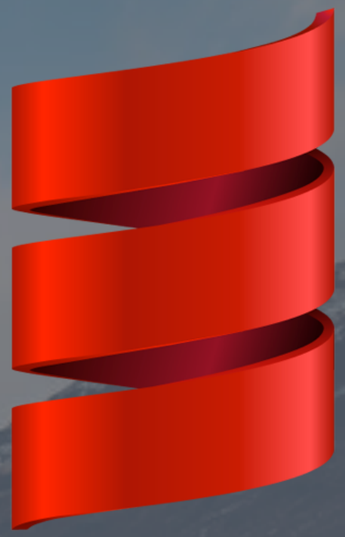
Other recipients: tkau...@batblue.com, bpa...@batblue.co


SOUNDCLOUD

TLC

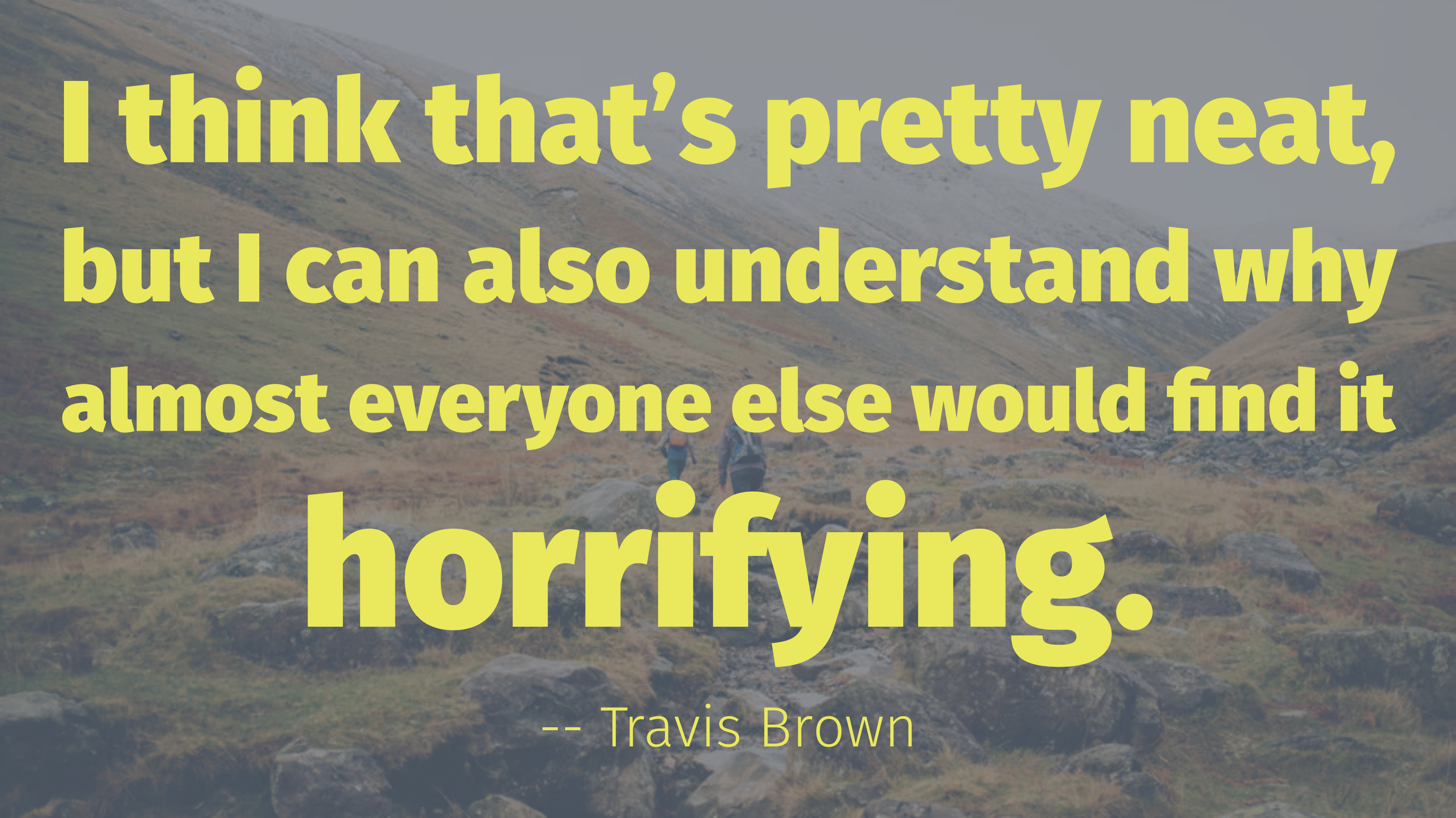
T-L-what now?[0]

[0] http://typelevel.org/blog/2014/09/02/typelevel-scala.html

We ❤ 

👻

Roll your own Scala


I think that's pretty neat,
but I can also understand why
almost everyone else would find it
horrifying.
-- Travis Brown

TLC

```
scalaVersion := "2.11.7"
scalaOrganization := "org.typelevel"
```

[some] Features

→ Type lambdas

→ `@implicitAmbiguous` (coming to 2.12 #4673)

→ Singleton types

→ -Zirrefutable-generator-patterns

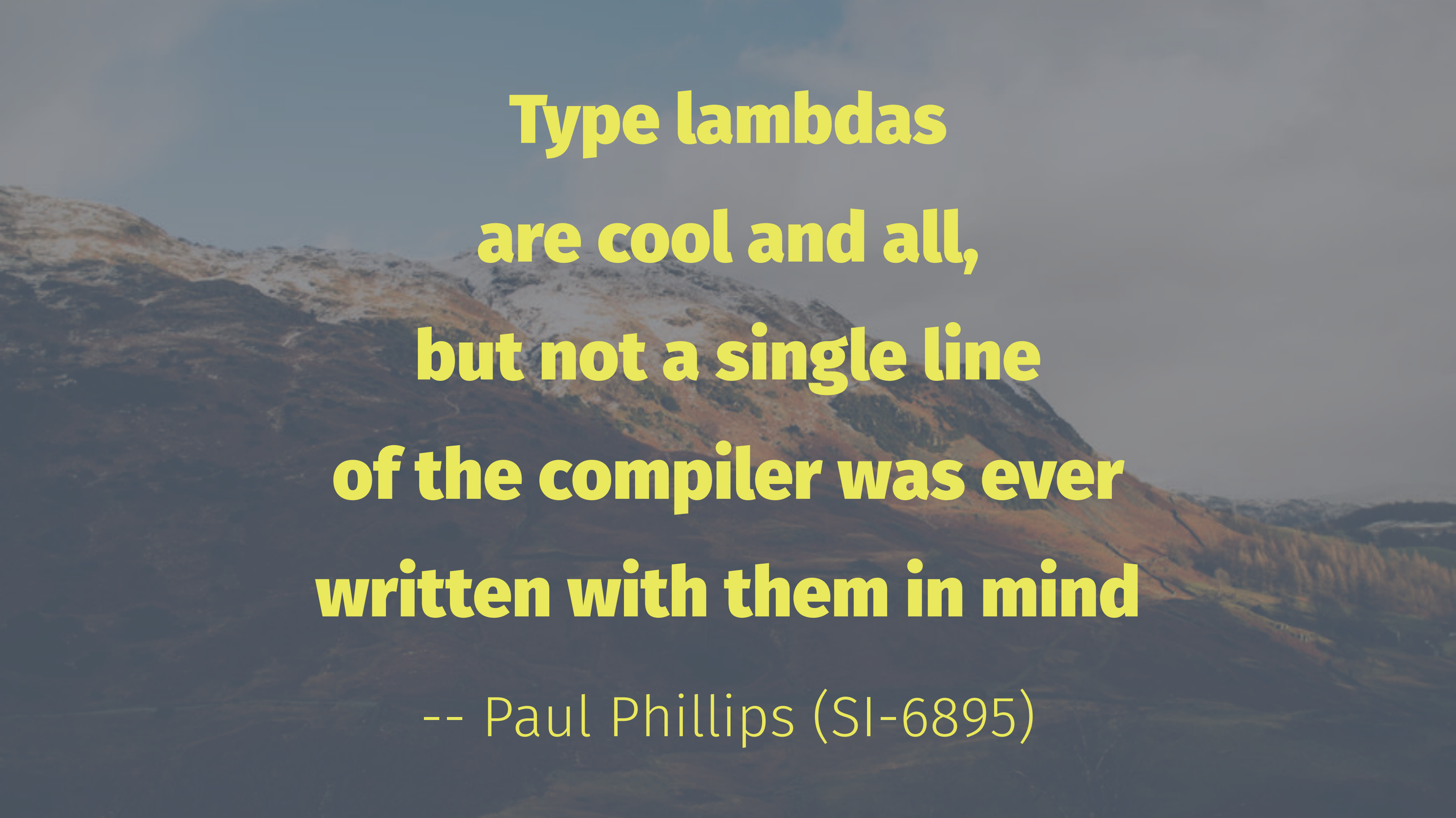
→ Nifties

Type lambdas

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(fn: A => B): F[B]
}

trait LeftFunctor[R] extends
  Functor[({type U[x] = Either[x, R]})#U]

trait RightFunctor[L] extends
  Functor[[y] => Either[L, y]]
```

Type lambdas

```
[x] => (x, x)
[x, y] => (x, Int) => y
[x[_]] => x[Double]
[+x, -y] => Function1[y, x]
```


Type lambdas
are cool and all,
but not a single line
of the compiler was ever
written with them in mind
-- Paul Phillips (SI-6895)

@implicitAmbiguous[1]

```
// Encoding for "A is not a subtype of B"
trait <:!<[A, B]

// Uses ambiguity to rule out the cases we're trying to exclude
implicit def nsub[A, B] : A <:!< B = null
@typelevel.annotation.implicitAmbiguous("Returning ${B} is forbidden.")
implicit def nsubAmbig1[A, B >: A] : A <:!< B = null
implicit def nsubAmbig2[A, B >: A] : A <:!< B = null

// Type alias for context bound
type |¬|[T] = {
  type λ[U] = U <:!< T
}
```

[1] https://gist.github.com/milessabin/c9f8befa932d98dcc7a4

@implicitAmbiguous

```
def foo[T, R : |¬|[Unit]#λ](t: T)(f: T => R) = f(t)

foo(23)(_ + 1)   // OK
foo(23)(println) // Doesn't compile: "Returning Unit is forbidden."
```

Singleton types[2]

```
trait Assoc[K] { type V ; val v: V }

def mkAssoc[K, V0](k: K, v0: V0): Assoc[k.type] { type V = V0 } =
  new Assoc[k.type] {type V = V0 ; val v = v0}
def lookup[K](k: K)(implicit a: Assoc[k.type]): a.V = a.v
```

[2] requeires `-Xexperimental` flag

Singleton types

```
implicit def firstAssoc = mkAssoc(1, "Panda!")
//> firstAssoc : Assoc[Int(1)]{type V = String}
implicit def secondAssoc = mkAssoc(2, 2.0)
//> secondAssoc : Assoc[Int(2)]{type V = Double}

implicit def ageAssoc = mkAssoc("Age", 3)
//> ageAssoc : Assoc[String("Age")]{type V = Int}
implicit def nmAssoc = mkAssoc("Name", "Jane")
//> nmAssoc : Assoc[String("Name")]{type V = String}
```

Singleton types

```
lookup(1)
// > res1: String = Panda!
lookup(2)
// > res2: Double = 2.0
lookup("Age")
// > res3: Int = 3
lookup("Name")
// > res4: String = Jane
```

Irrefutable generator patterns

```
for {
  (x, _) <- Option((1, 2))
} yield x
```

Desugars to

```
Some(scala.Tuple2(1, 2))
  .withFilter(((check$ifrefutable$1) =>
    check$ifrefutable$1: @scala.unchecked match {
      case scala.Tuple2((x @ _), _) => true
      case _ => false
    })).map(((x$1) => x$1: @scala.unchecked match {
      case scala.Tuple2((x @ _), _) => x
    }))
```

With irrefutable patterns

```
Some(scala.Tuple2(1, 2)).map(((x$1) => x$1 match {
  case scala.Tuple2((x @ _), _) => x
}))
```

Няшки

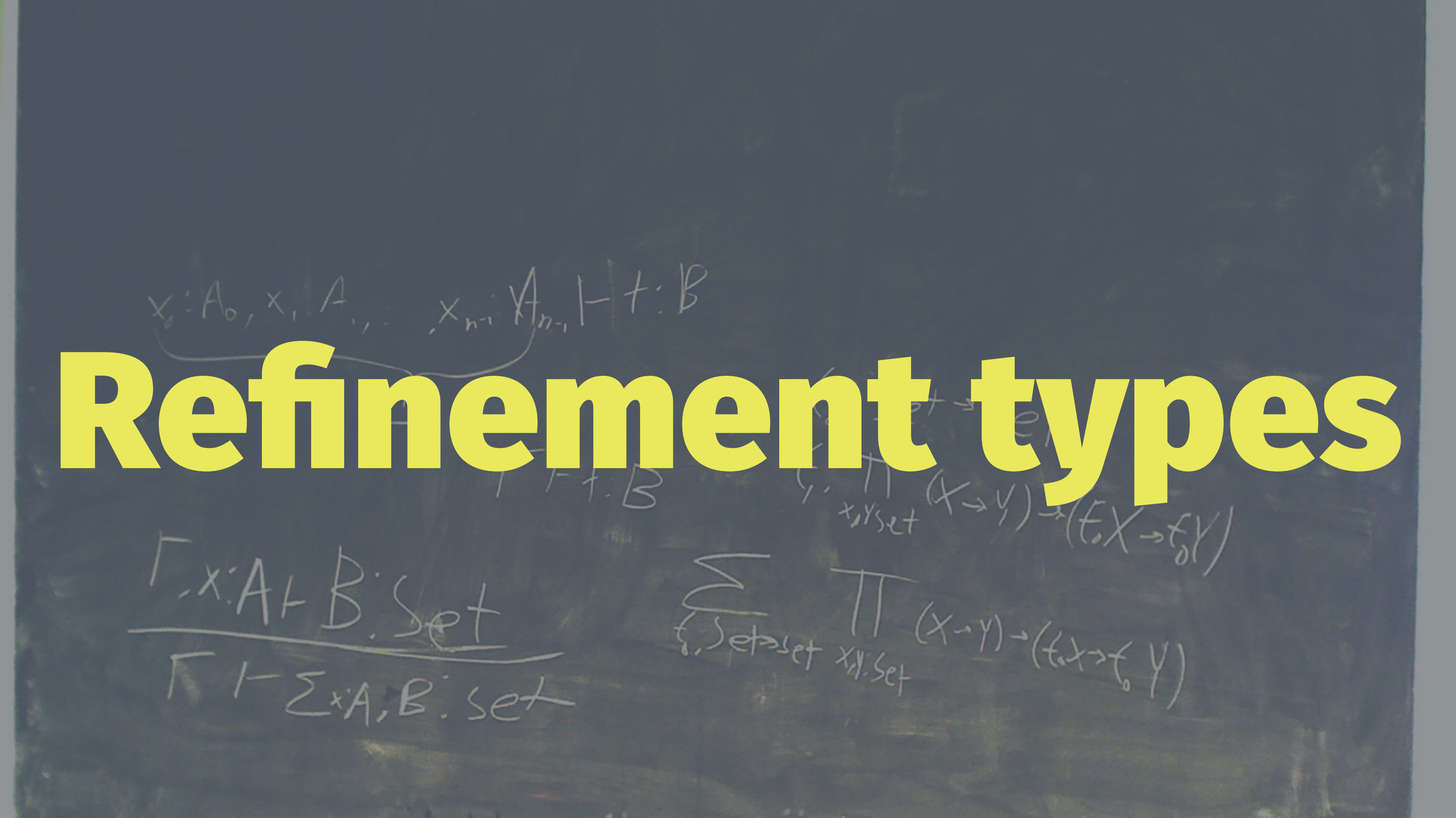
```
def fib(n: Int) = {
  def fib'(n: Int, next: Int, £: Int): Int =
    n match {
      case 0 => £
      case _ => fib'(n - 1, next + £, next)
    }
  fib'(n, 1, 0)
}

val £': Byte  = 127z
```

Vision

Low-hanging fruits

- → converting partest tests to junit
- → documentation
- → Reporting bugs
- → Fixing bugs
- → Backporting changes from typesafe scala

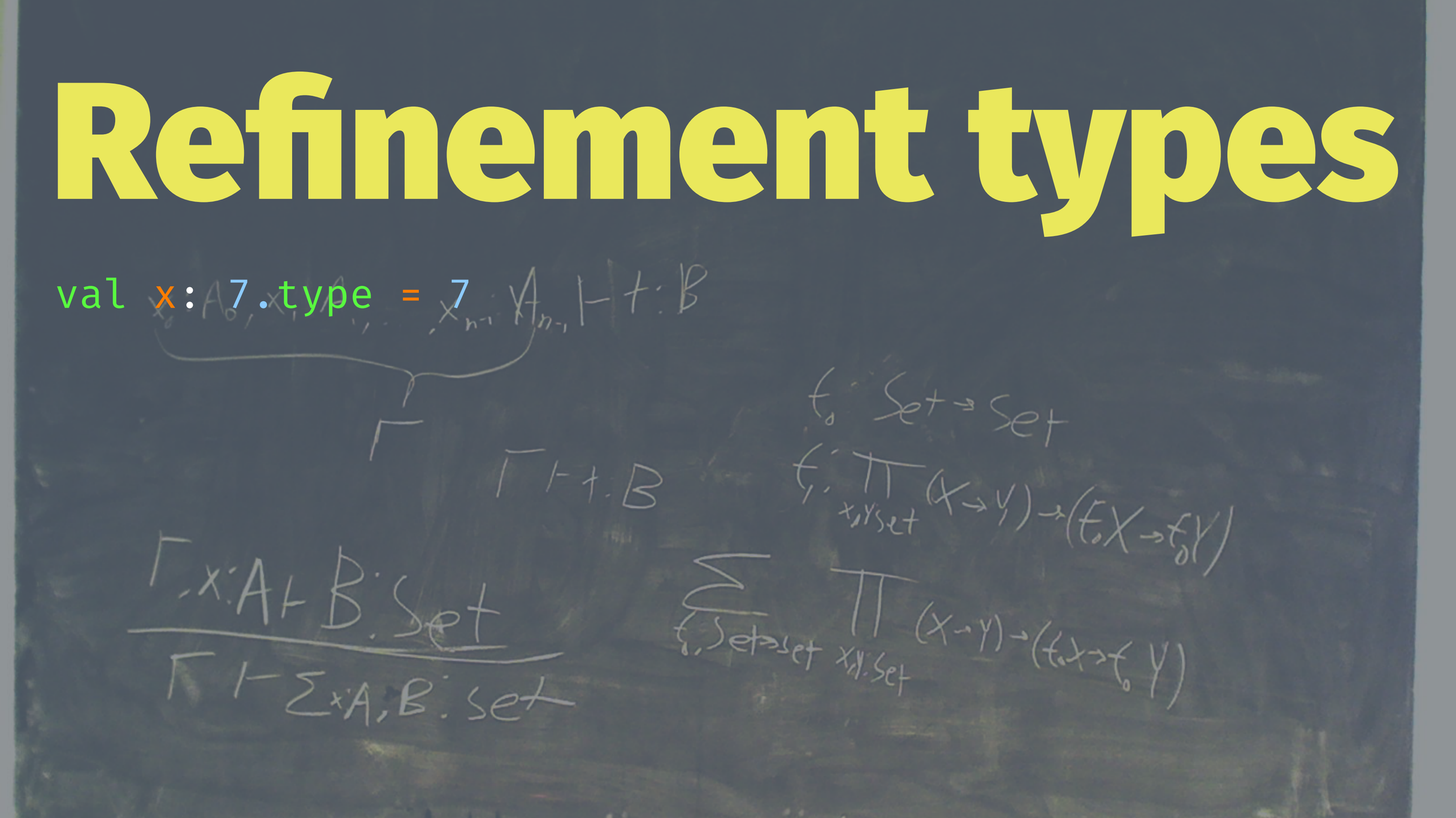
Let's fantasize a bit now



Refinement types

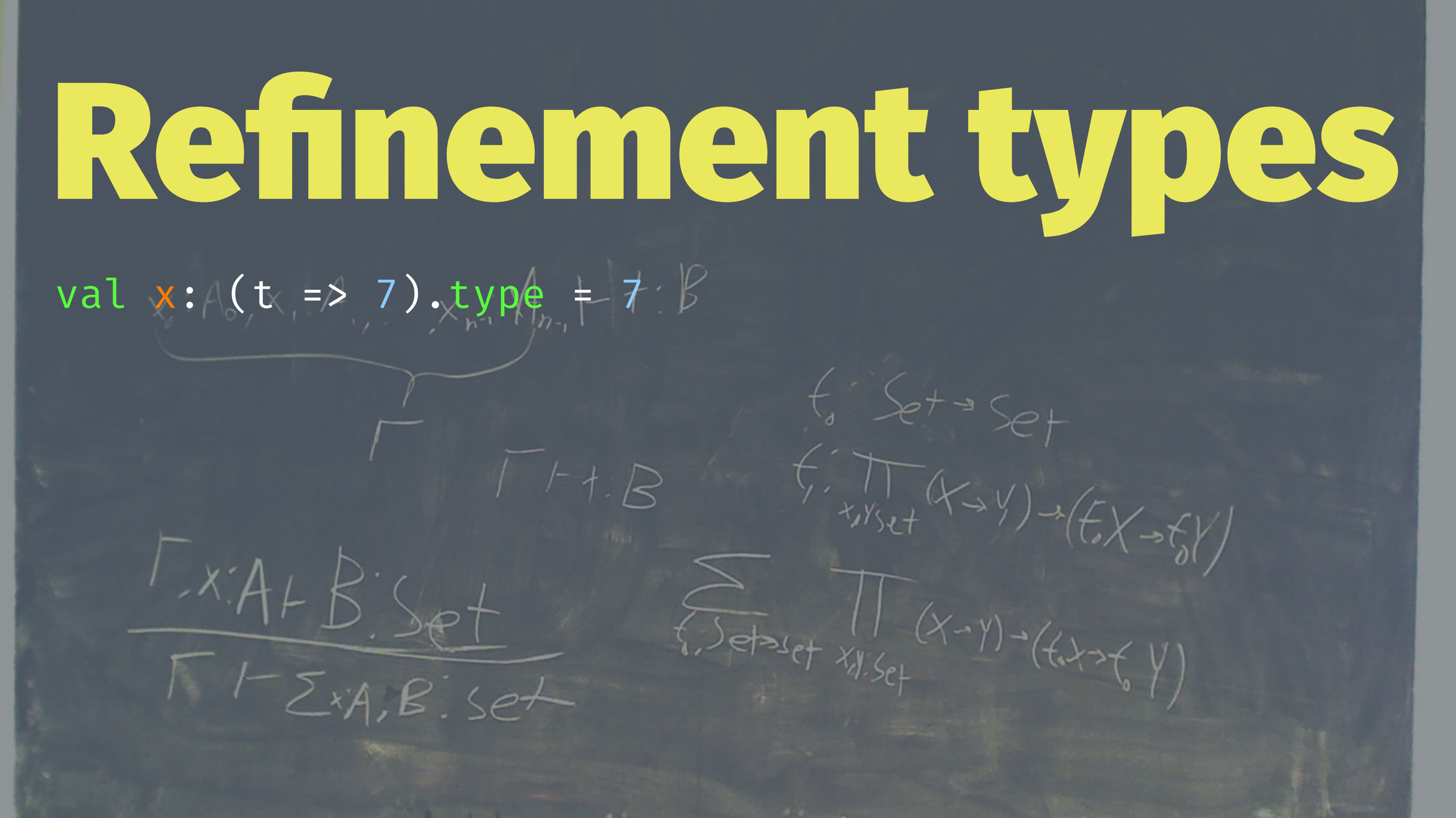
Refinement types

```
val x: 7.type = 7
```



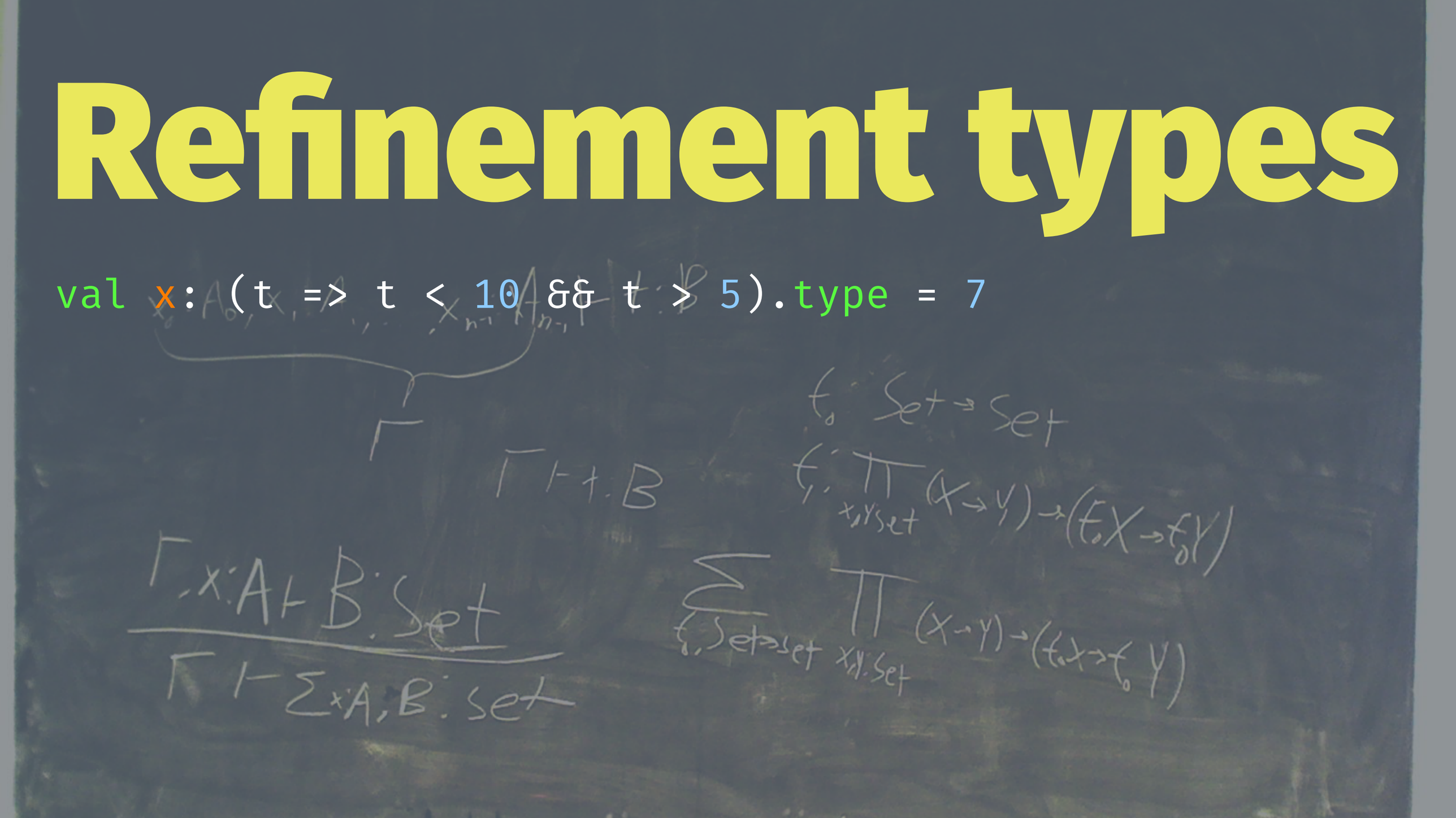
Refinement types

```
val x: (t => 7).type = 7
```



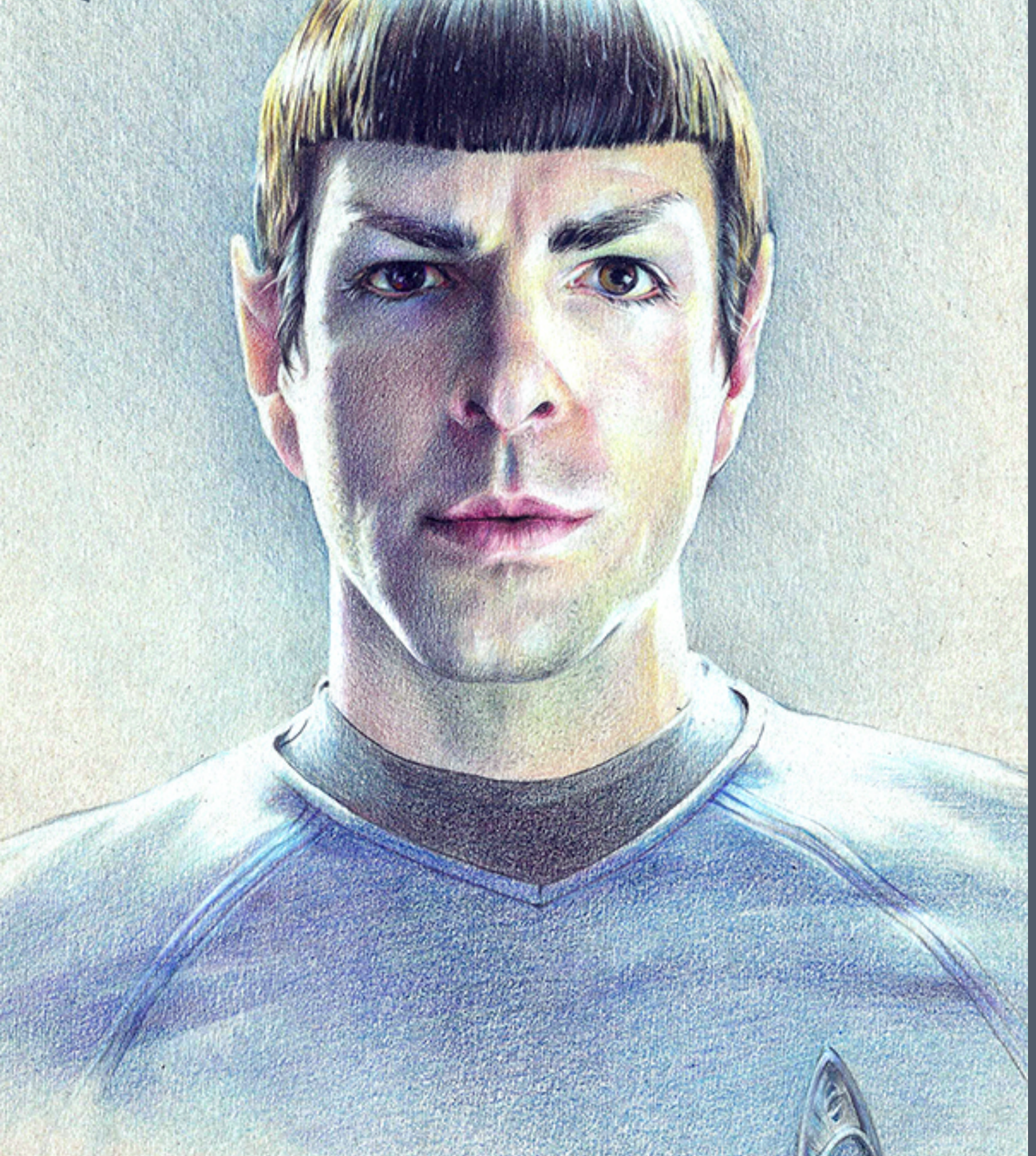
Refinement types

```
val x: (t => t < 10 && t > 5).type = 7
```



Experiment more with stdlib

Integrating alternative repl



Rethinking the role of implicits[3]

There's a Prolog in your Scala:
http://j.mp/prolog-scala-talk

[3] https://github.com/typelevel/scala/issues/28 -- `@implicitWeight`

Hands-on

```
scalaVersion       := "2.11.7"
scalaOrganization := "org.typelevel"
```

What do I do once I check out the repo?

```
sbt compile
sbt test
sbt partest
```


./tools/
partest-ack


To summarize


Danke!
@folone
https://soundcloud.com/jobs