@ochafik

Scalaxy/Streams

Faster collections, now !

github.com/ochafik/Scalaxy

Olivier Chafik

@ochafik

Disclaimer: I'm not doing Scala @ Google.

Views & Copyright are my own.

- ScalaCL: Scala on GPU
  - GPU doesn't like libraries…
  - Rewriting loops helps regular Scala
  - 4+ years of compiler plugin / macro hacks.

- Native bindings: BridJ, JNAerator, JavaCL…

github.com/ochafik/Scalaxy

for comprehension

```
for (i <- 0 to n)
  println(i)
```

```
scala.Predef.intWrapper(0).to(n).foreach(i => {
 println(i)
})
```

for comprehension

```
for (i <- 0 to n;
     ii = i * i;
     j <- i to n;
     jj = j * j;
     if (ii - jj) % 2 == 0;
     k <- (i + j) to n)
  yield ii * jj + k
```

map,

flatMap,

filter ?

for *incomprehension*

```
for (i <- 0 to n;
     ii = i * i;
     j <- i to n;
     jj = j * j;
     if (ii - jj) % 2 == 0;
     k <- (i + j) to n)
  yield ii * jj + k
```

```
(0 to n).map(i => {
  val ii = i * i
  (i, ii)
}).flatMap(_ match {
  case (i, ii) =>
    (i to n).map(j => {
      val jj = j * j
      (j, jj)
    }).withFilter(_ match {
      case (j, jj) =>
        ii - jj % 2 == 0
    }).flatMap(_ match {
      case (j, jj) =>
        ((i + j) to n)
          .map(k => ii * jj + k)
    })
})
```

github.com/ochafik/Scalaxy

for -> while

```
for (i <- 0 to n;
     ii = i * i;
     j <- i to n;
     jj = j * j;
     if (ii - jj) % 2 == 0;
     k <- (i + j) to n)
  yield ii * jj + k
```

```
val out = VectorBuilder[Int]()
var i = 0
while (i <= n) {
 val ii = i * i
 var j = i
 while (j <= n) {
   val jj = j * j
   if ((ii - jj) % 2 == 0) {
     var k = i + j
     while (k <= n) {
       out += (ii * jj + k)
       k += 1
     }
   }
   j += 1
 }
 i += 1
}
```

github.com/ochafik/Scalaxy

for -> while

```
for (i <- 0 to n;
     ii = i * i;
     j <- i to n;
     jj = j * j;
     if (ii - jj) % 2 == 0;
     k <- (i + j) to n)
  yield ii * jj + k
```

```
val out = VectorBuilder[Int]()
var i = 0
while (i <= n) {
 val ii = i * i
 var j = i
 while (j <= n) {
   val jj = j * j
   if ((ii - jj) % 2 == 0) {
     var k = i + j
     while (k <= n) {
       out += (ii * jj + k)
       k += 1
     }
   }
   j += 1
 }
 i += 1
}
```

github.com/ochafik/Scalaxy

Not an isolated case

```
(0 until n)
  .filter(v => (v % 2) == 0)
  .map(_ * 2)
  .forall(x => x < n / 2)
```

github.com/ochafik/Scalaxy

Not an isolated case

```
(0 until n)
  .filter(v => (v % 2) == 0)    // -> Vector
  .map(_ * 2)                   // -> Vector
  .forall(x => x < n / 2)       // -> Boolean
```

Intermediate collections, really?

Iterators can only help so much…

```
(0 until n).toIterator
  .filter(v => (v % 2) == 0)    // -> Iterator
  .map(_ * 2)                   // -> Iterator
  .forall(x => x < n / 2)       // -> Boolean
```

Closure elim, megamorphic methods, inlining...

github.com/ochafik/Scalaxy

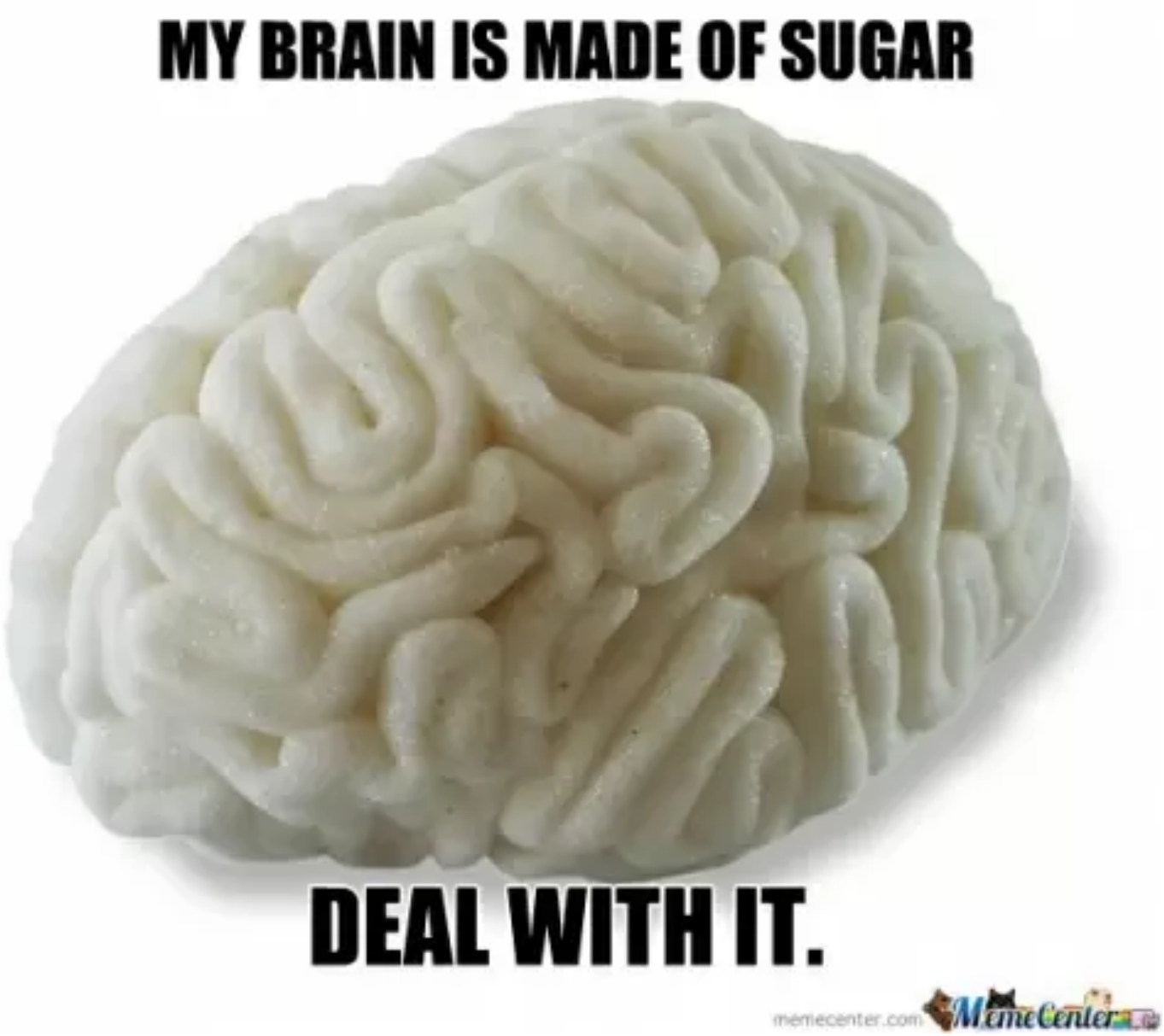
Scala is fast… with while loops

spectral-norm Scala program source code

```
/* The Computer Language Benchmarks Game
   http://benchmarksgame.alioth.debian.org/
   contributed by Isaac Gouy
   modified by Meiko Rachimow
   updated for 2.8 by Rex Kerr
*/

object spectralnorm {
  def main(args: Array[String]) = {
    val n = (if (args.length>0) args(0).toInt else 100)
    printf("%.09f\n", (new SpectralNorm(n)).approximate())
  }
}

class SpectralNorm(n: Int) {

  // Ordinary and transposed versions of infinite matrix
  val A = (i: Int, j: Int) => 1.0/((i+j)*(i+j+1)/2 +i+1)
  val At = (j: Int, i: Int) => 1.0/((i+j)*(i+j+1)/2 +i+1)

  // Matrix multiplication w <- M*v
  def mult(v: Array[Double], w: Array[Double], M: (Int,Int)=> Double ) {
    var i = 0
    while (i < n) {
     var s = 0.0
     var j = 0
     while (j < n) { s += M(i,j)*v(j); j += 1 }
     w(i) =  s
     i += 1
    }
  }
```

http://benchmarksgame.alioth.debian.org/

github.com/ochafik/Scalaxy



github.com/ochafik/Scalaxy

Scalaxy in a slide...

- Faster for comprehensions / collections
- Safe by default
- Easy to use

```
libraryDependencies += "com.nativelibs4java" %% "scalaxy-streams" %
  "0.3.2" % "provided"
```

```
import scalaxy.streams.optimize
optimize {
  for (i <- 0 to 10)    // -> while loop
    println(i)
}
```

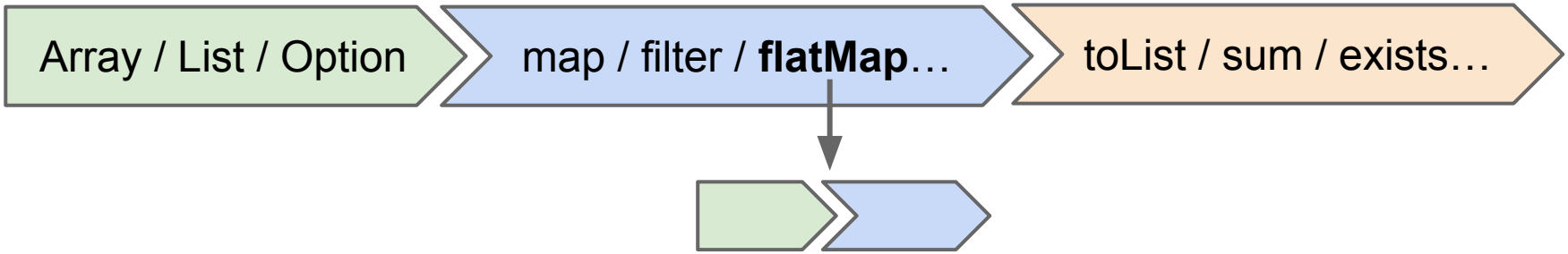
github.com/ochafik/Scalaxy

… or in a couple of slides

- Sources / operations / sinks + nesting
- Tuples elimination
- Side-effect detection
- Optimization strategies
- Macro + plugin

github.com/ochafik/Scalaxy

What is a stream? (in this context)



Quasiquotes matching on ASTs:

```
def unapply(tree: Tree) = Option(tree) collect {
  case q"$target.filter($param => $body)" =>
    FilterOp(target, param, body)
}
```

Tuples are nice… and often skippable

No tuples were harmed (or created) here:

```
list.zipWithIndex
    .filter(_._2 % 2 == 0)
    .map({ case (v, i) => v + i })
```

Sometimes we want to materialize them:

```
list.zipWithIndex
    .filter(_._2 % 2 == 0)
    .foreach(tup => println(tup))
```

Side-effects...

```
(1 to 2)
        .map(i => { println("first map, " + i); i })
        .map(i => { println("second map, " + i); i })
        .take(1)

        //   first map, 1
        //   first map, 2
        //   second map, 1
        //   second map, 2
```

(there's worse side-effects than `println` :-))

Aggressive optimization: ≠ semantics

```
(1 to 2).toIterator
        .map(i => { println("first map, " + i); i })
        .map(i => { println("second map, " + i); i })
        .take(1)
        .toSeq

        //   first map, 1
        //   second map, 1
```

Scalaxy/Streams doesn’t do that *by default* !

github.com/ochafik/Scalaxy

Trusted APIs vs. side-effects

- "*Almost safe*":
  `toString`, `hashCode`, `equals`, `+`, `++`
- Immutable classes: `Int`, `String`…
- Whitelisted `Predef` utils / classes: `RichInt`…
- Immutable collections (except `toString`…)

github.com/ochafik/Scalaxy

Some cases are already “optimal”

```
@noinline
final override def flatMap[B, That](f: A => GenTraversableOnce[B])(implicit bf: CanBuildFrom[List[A], B, That]): That = {
 if (bf eq List.ReusableCBF) {
   if (this eq Nil) Nil.asInstanceOf[That] else {
     var rest = this
     var found = false
     var h: ::[B] = null
     var t: ::[B] = null
     while (rest ne Nil) {
       f(rest.head).foreach{ b =>
         if (!found) {
           h = new ::(b, Nil)
           t = h
           found = true
         }
         else {
           val nx = new ::(b, Nil)
           t.tl = nx
           t = nx
         }
       }
       rest = rest.tail
     }
     (if (!found) Nil else h).asInstanceOf[That]
   }
 }
 else super.flatMap(f)
}
```

`List.flatMap`: hand-optimized

Rule thumb: optimize `List` when there’s more than 1 lambda

github.com/ochafik/Scalaxy

Optimization strategies

	safer	**safe**	aggressive	foolish
worthiness	X	X	X	
side-effects	X	X	*(warn)*	*(warn)*
don't trust .toString	X			

github.com/ochafik/Scalaxy

Macro vs. plugin

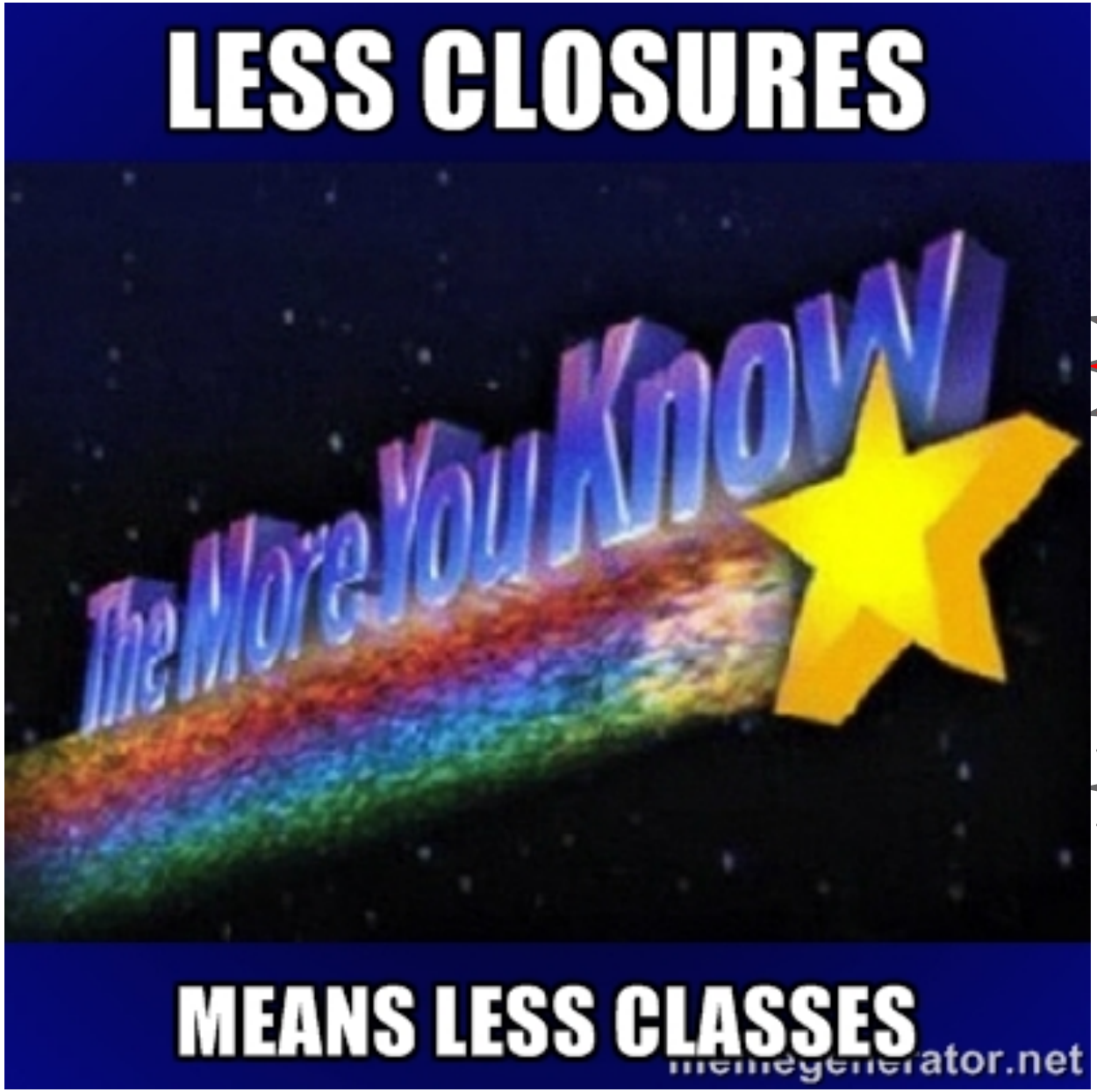
```
import scalaxy.streams.optimize
import scalaxy.streams.strategy.aggressive


    optimize {
      for (i <- 0 until n) {
        println(i)
      }
    }
```

github.com/ochafik/Scalaxy

Bottom line...

Fast'n'Easy, baby !

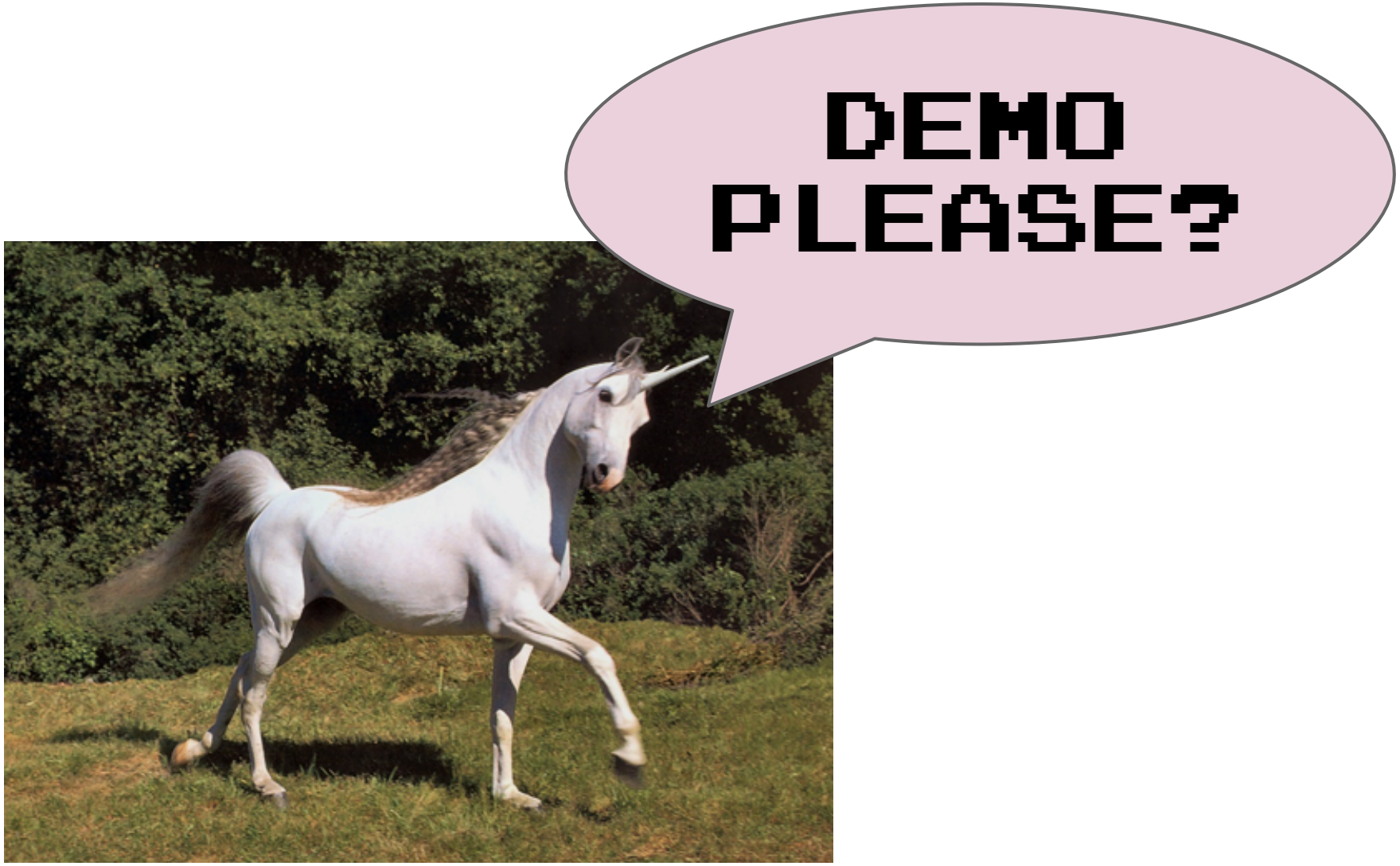
github.com/ochafik/Scalaxy



-5% size scala-lib

-60% size example

github.com/ochafik/Scalaxy

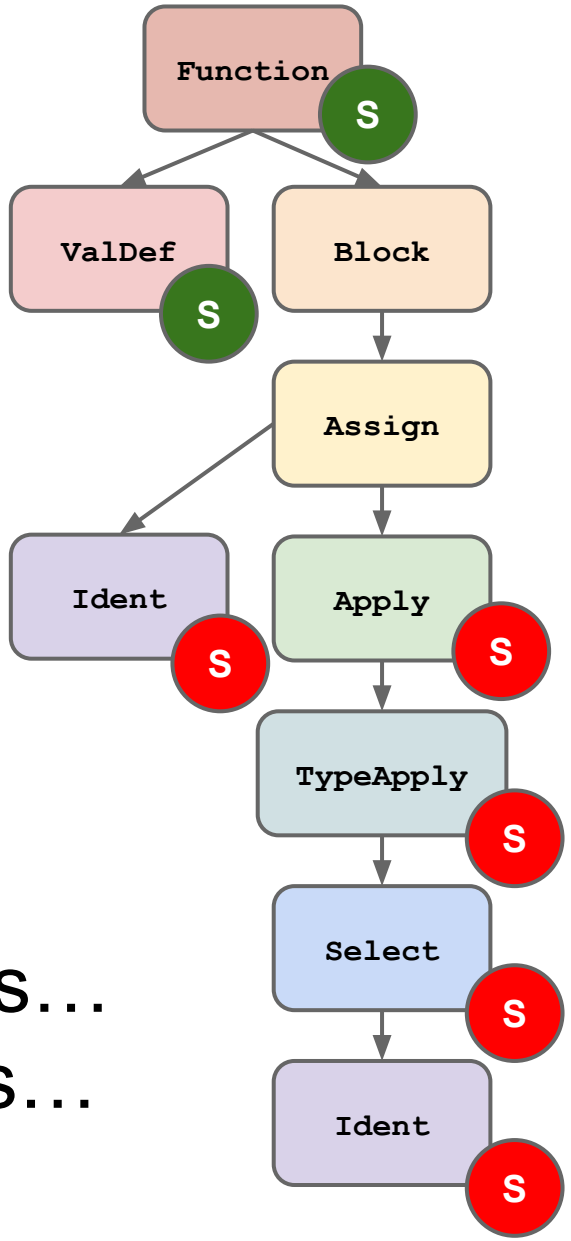


github.com/ochafik/Scalaxy

No unicorns for macro programmers

A peek at the grungy stuff

- Types, Symbols on Trees: crucial for matching
- Typer stops at first typed tree
  - Cannot *reliably* untype trees
  - Synthetic code must be typed
- Typer bring hierarchical symbols…
- Loop rewriter removes functions…



github.com/ochafik/Scalaxy

Removing functions is neat...

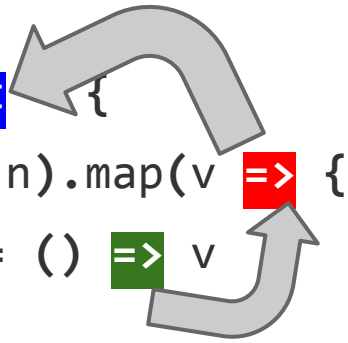
```
def parent = {
 (0 until n).map(v => {
   val f = () => v
   f
 })
}
```

```
def parent = {
 val out = VectorBuilder[() => Int]()
 var i = 0
 while (i < n) {
   val f = () => v
   out += f
   i += 1
 }
 out.result
}
```

github.com/ochafik/Scalaxy

But… every symbol has an owner

```
def parent {
 (0 until n).map(v => {
   val f = () => v
   f
 })
}
```




```
def parent = {
 val out = VectorBuilder[() => Int]()
 var i = 0
 while (i < n) {
   val f = () => v
   out += f
   i += 1
 }
 out.result
}
```

Optimizations delete functions

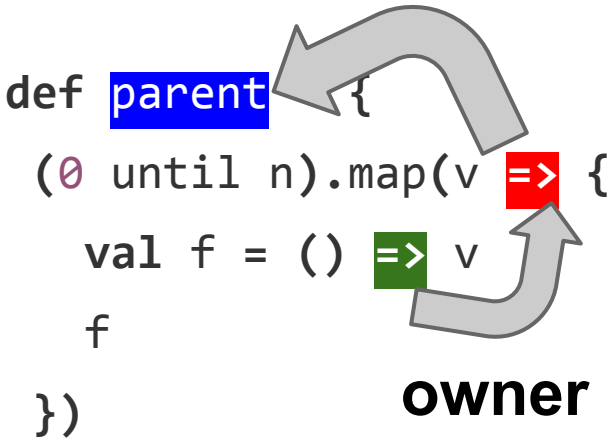
```
def parent {
 (0 until n).map(v => {
   val f = () => v
   f
 })
}
```

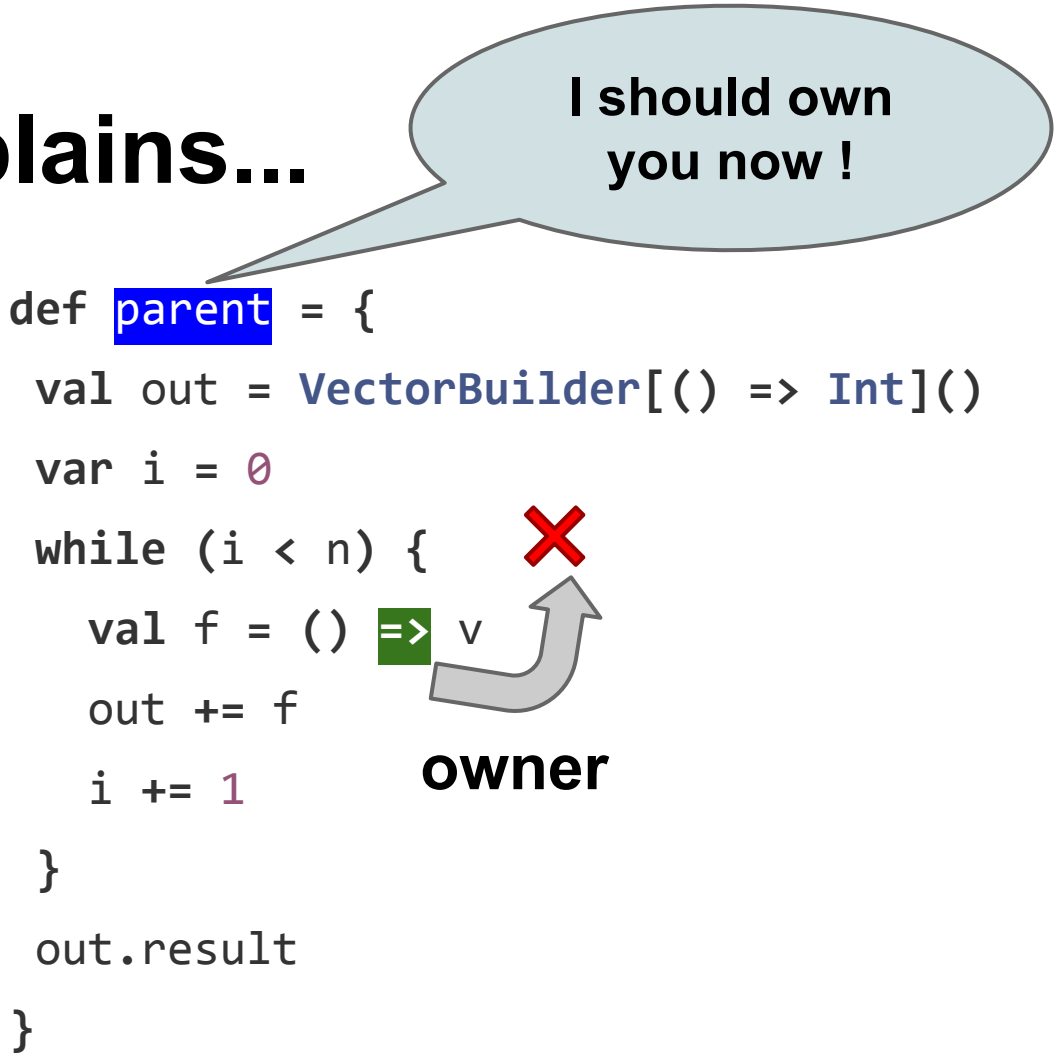
owner

```
def parent = {
 val out = VectorBuilder[() => Int]()
 var i = 0
 while (i < n) {
   val f = () => v
   out += f
   i += 1
 }
 out.result
}
```

owner

Compiler complains...





github.com/ochafik/Scalaxy

Adoption by grand-parent

```
def parent = {
 (0 until n).map(v => {
   val f = () => v
   f
 })
}
```

owner

```
def parent = {
 val out = Vect  uilder[() => Int]()
 var i = 0
 while (i < n)
   val f = () => v
   out += f
   i += 1
 }
 out.result
}
```

new owner

Official APIs don't help: reflection does :-)

github.com/ochafik/Scalaxy

Ahead-of-time typing

Can’t create symbols in macros?

```
val Block(List(startValDef, endValDef, ...), _) = typed(q"""
    val $startVal: $tpe = ${transform(start)};
    val $endVal: $tpe = ${transform(end)};
    ...
""")
```

Are you ready for Scalaxy/Streams?

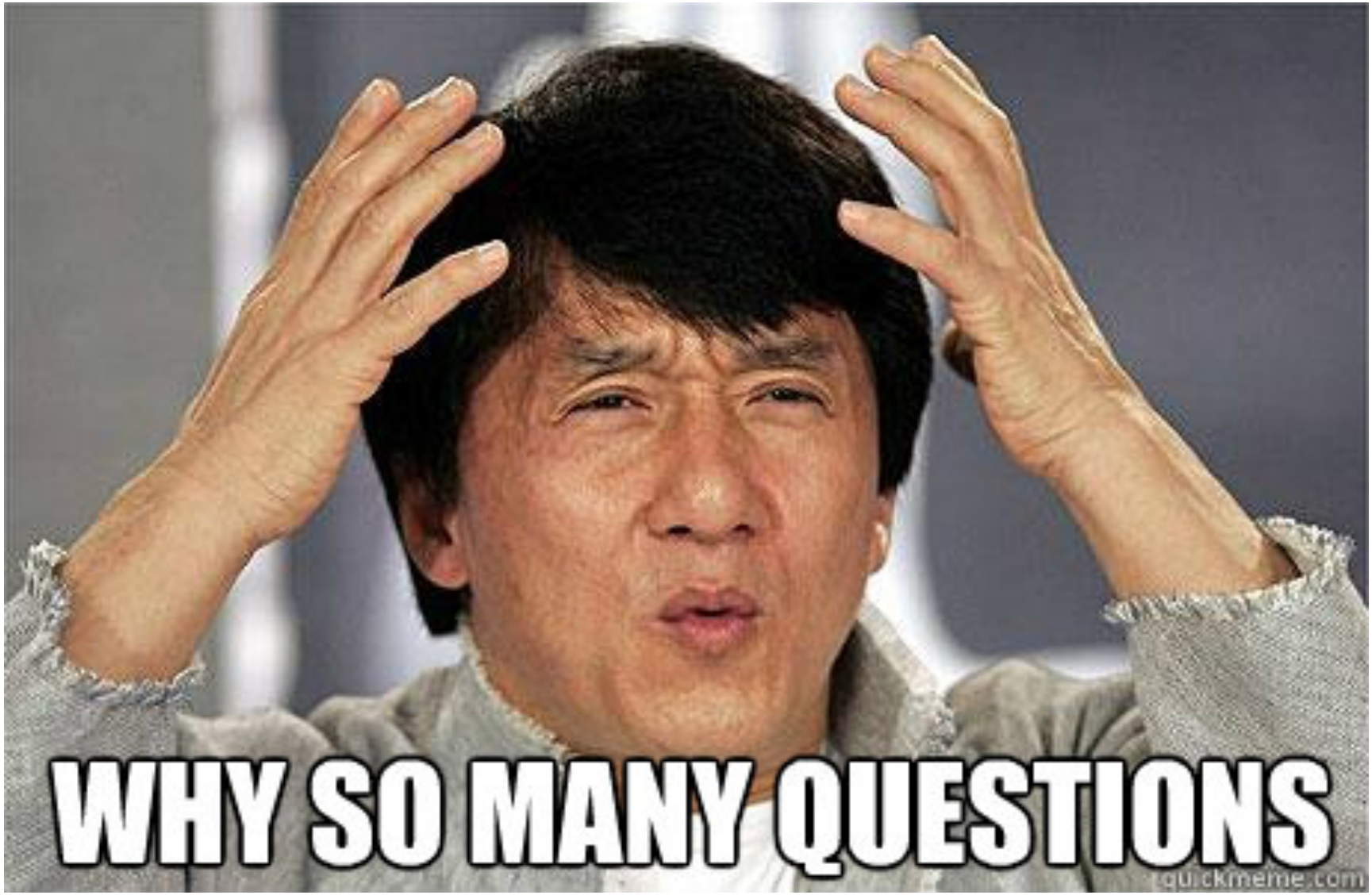
- 0.3.x: *reasonably experimental*
- Macro => minimize risks
  Plugin => maximize gains

```
optimize { … here … }
```

- Mostly builds scala-library.jar (**-5%** JAR size)
- *Note*: ScalaBlitz = concurrent project
  (parallelizes, no operation fusion / side-effect analysis)

github.com/ochafik/Scalaxy

https://github.com/ochafik/Scalaxy/issues/new



github.com/ochafik/Scalaxy