Metaprogramming 2.0

Eugene Burmako (@xeno_by)

École Polytechnique Fédérale de Lausanne
http://scalameta.org/

17 June 2016

This talk was presented and recorded at ScalaDays Berlin 2016:
https://www.youtube.com/watch?v=IPnd_SZJ1nM

Today's talk

- What is scala.meta?
- What can scala.meta do?
- Why should you care?
- What will happen next?

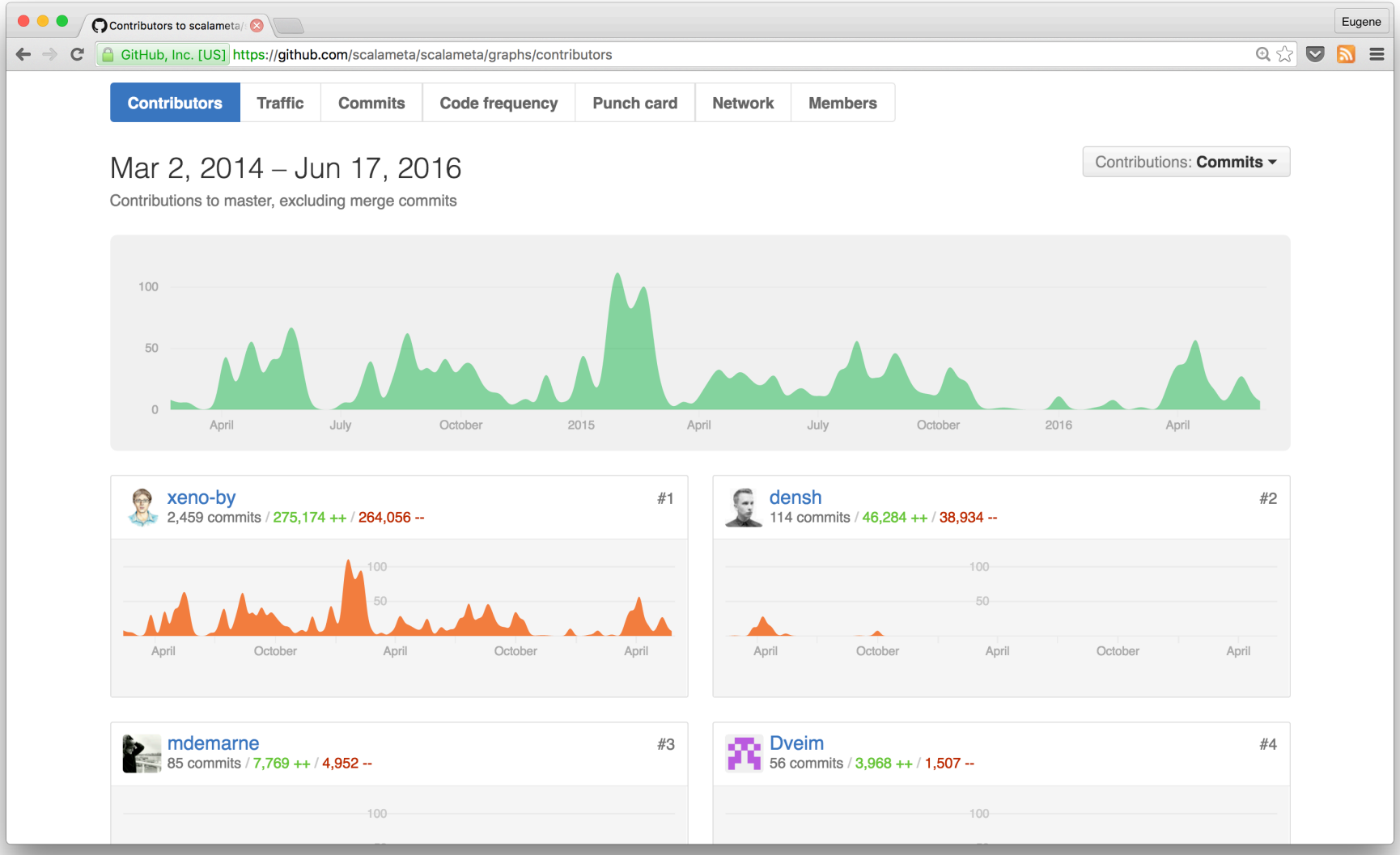
Easy Metaprogramming For Everyone!

Eugene Burmako (@xeno_by)
Denys Shabalin (@den_sh)

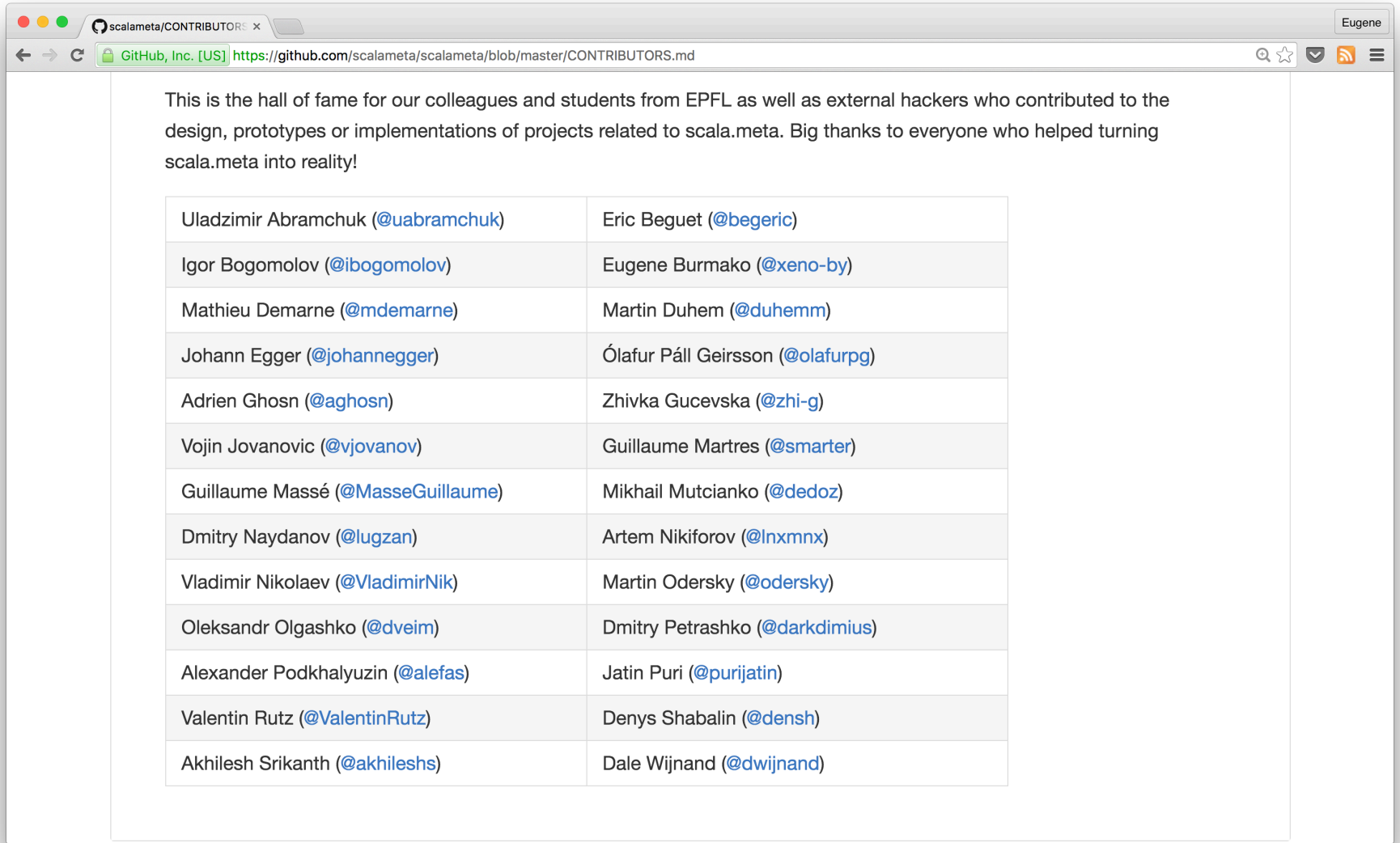
École Polytechnique Fédérale de Lausanne
http://scalameta.org/

17 June 2014

scala.meta is an active project



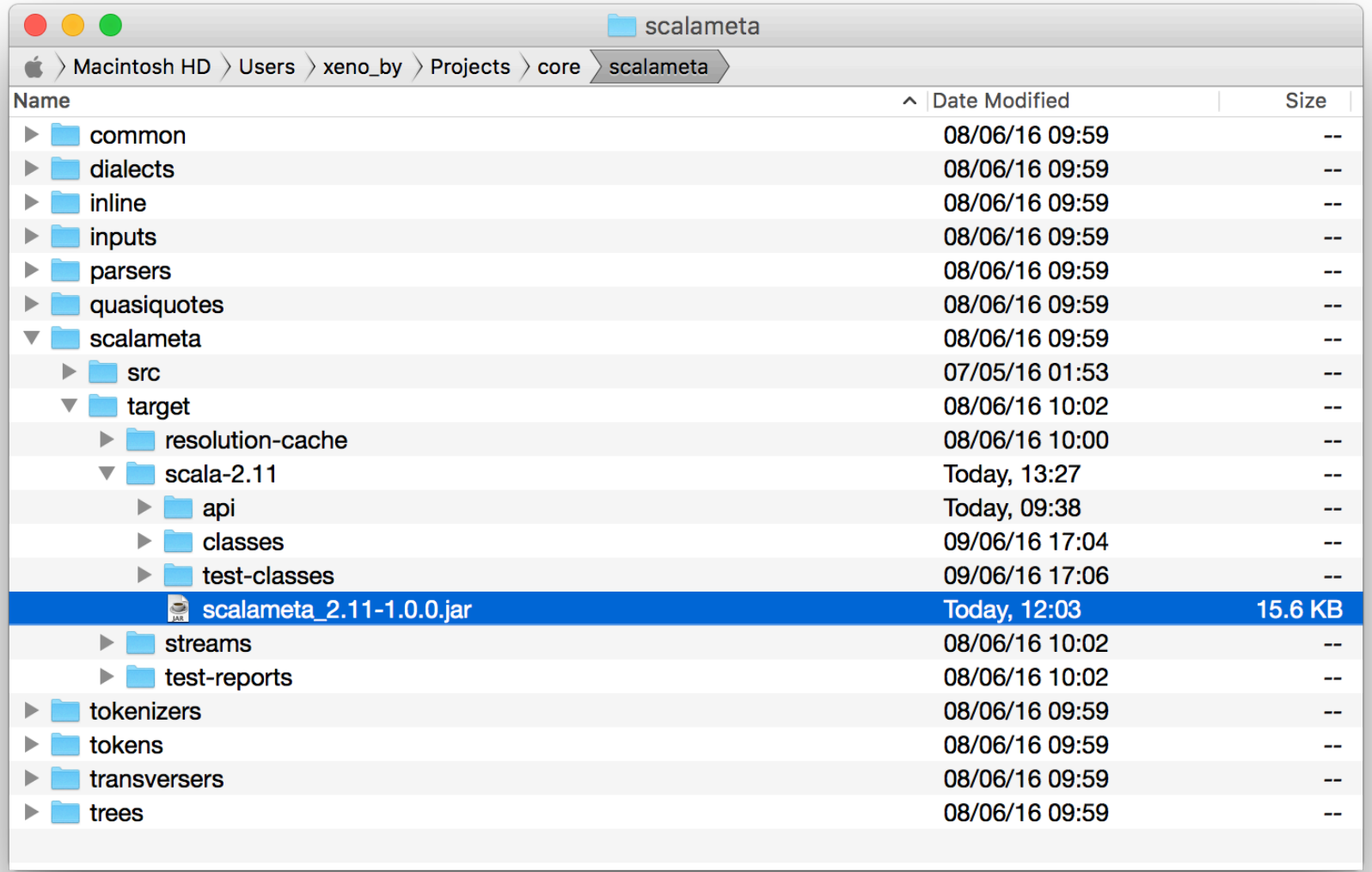
scala.meta is a community



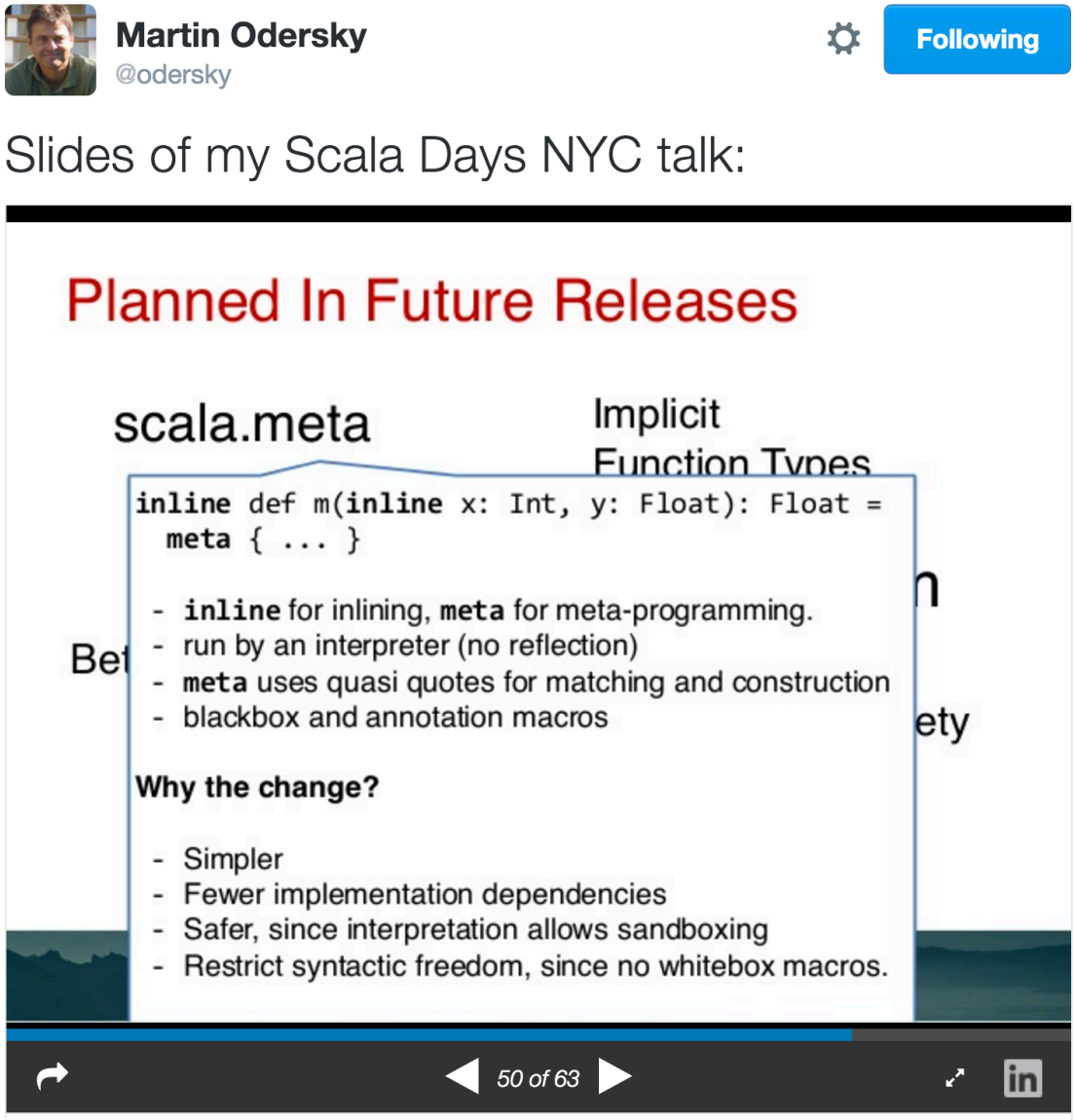
This is the hall of fame for our colleagues and students from EPFL as well as external hackers who contributed to the design, prototypes or implementations of projects related to scala.meta. Big thanks to everyone who helped turning scala.meta into reality!

Uladzimir Abramchuk (@uabramchuk)	Eric Beguet (@begeric)
Igor Bogomolov (@ibogomolov)	Eugene Burmako (@xeno-by)
Mathieu Demarne (@mdemarne)	Martin Duhem (@duhemm)
Johann Egger (@johannegger)	Ólafur Páll Geirsson (@olafurpg)
Adrien Ghosn (@aghosn)	Zhivka Gucevska (@zhi-g)
Vojin Jovanovic (@vjovanov)	Guillaume Martres (@smarter)
Guillaume Massé (@MasseGuillaume)	Mikhail Mutcianko (@dedoz)
Dmitry Naydanov (@lugzan)	Artem Nikiforov (@lnxmnx)
Vladimir Nikolaev (@VladimirNik)	Martin Odersky (@odersky)
Oleksandr Olgashko (@dveim)	Dmitry Petrashko (@darkdimius)
Alexander Podkhalyuzin (@alefas)	Jatin Puri (@purijatin)
Valentin Rutz (@ValentinRutz)	Denys Shabalin (@densh)
Akhilesh Srikanth (@akhileshs)	Dale Wijnand (@dwijnand)

scala.meta is a product



scala.meta is officially endorsed



Part 1: What can scala.meta do?

Project status

- 17 Jun 2014: “Easy metaprogramming for everyone!”

Project status

- 17 Jun 2014: “Easy metaprogramming for everyone!”
- 17 Jun 2016: 3191 commits and 27 milestones later...

Project status

- 17 Jun 2014: “Easy metaprogramming for everyone!”
- 17 Jun 2016: 3191 commits and 27 milestones later...
- We’re finally releasing v1.0!

Supported functionality

- Vendor-neutral tree interchange format
- High-fidelity parsing
- First-class tokens

Parsing: easy to get started

```
scala> import scala.meta._
import scala.meta._

scala> "x + y".parse[Term]
res0: scala.meta.parsers.Parsed[scala.meta.Term] = x + y
```

Parsing: remember all syntactic details

```
scala> import scala.meta._
import scala.meta._

scala> "x + y".parse[Term]
res0: scala.meta.parsers.Parsed[scala.meta.Term] = x + y

scala> "x + y // hello world".parse[Term]
res1: scala.meta.parsers.Parsed[scala.meta.Term] =
x + y // hello world
```

Tokens: remember all syntactic details

```
scala> val add = "x + y // hello world".parse[Term].get
add: scala.meta.Term = x + y // hello world

scala> add.tokens
res2: scala.meta.tokens.Tokens =
Tokens(, x,  , +,  , y,  , // hello world, )
```

Tokens: remember all syntactic details

```
scala> val add = "x + y // hello world".parse[Term].get
add: scala.meta.Term = x + y // hello world

scala> add.tokens
res2: scala.meta.tokens.Tokens =
Tokens(, x,  , +,  , y,  , // hello world, )

scala> add.tokens.structure
res3: String = Tokens(BOF [0..0), x [0..1),  [1..2),
+ [2..3), [3..4), y [4..5),  [5..6),
// hello world [6..20), EOF [20..20))
```

Parsing: support for dialects

```
scala> val sbtBuild = new File(".../project/plugins.sbt")
sbtBuild: java.io.File = .../project/plugins.sbt
```

Parsing: support for dialects

```
scala> val sbtBuild = new File(".../project/plugins.sbt")
sbtBuild: java.io.File = .../project/plugins.sbt

scala> scala.meta.dialects.Sbt0136(sbtBuild).parse[Source]
res4: scala.meta.parsers.Parsed[scala.meta.Source] =

addSbtPlugin("com.typesafe.sbt" % "sbt-pgp" % "0.8.1")

addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.11.2")
...
```

Quasiquotes: stealing better parts of scala.reflect

```
scala> q"x + y"
res5: scala.meta.Term.ApplyInfix = x + y

scala> val q"$a + $b" = res5
a: scala.meta.Term = x
b: scala.meta.Term.Arg = y
```

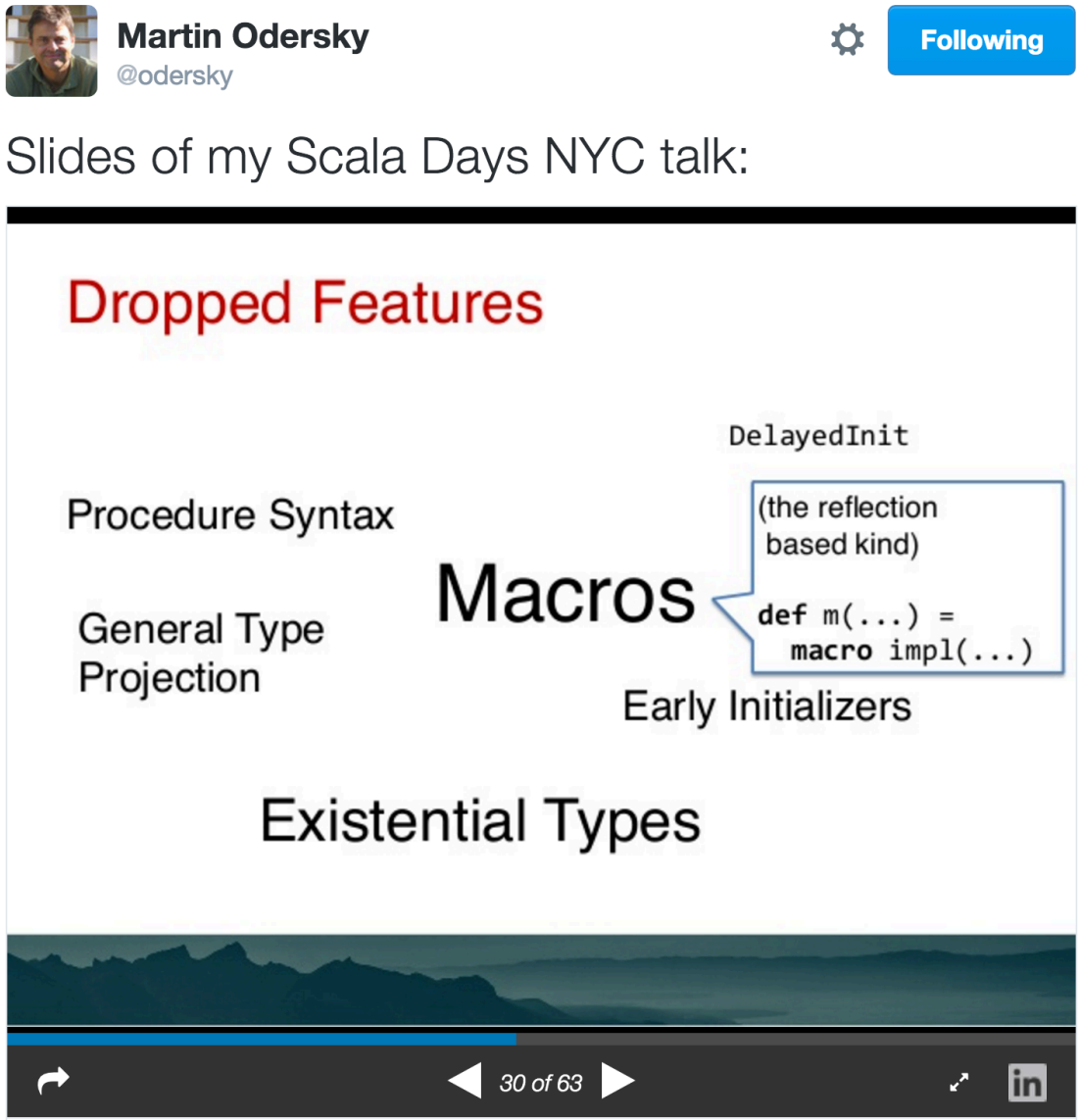
Part 2: Why should you care?

Why should you care?

Scala.meta provides unique features that enable much better tools

Case study: Macros

Macros are dead?



Macros are bad

- Hard to write
- Don't work with tools well

Example: the @main macro

```
@main object Test {
  println("hello world")
}
```

```
object Test {
  def main(args: Array[String]): Unit = {
    println("hello world")
  }
}
```

Example: the @main macro

```
class main extends scala.annotation.StaticAnnotation {
  def macroTransform(annottees: Any*): Any =
    macro Macros.impl
}
```

Example: the @main macro

```
class main extends scala.annotation.StaticAnnotation {
  def macroTransform(annottees: Any*): Any =
    macro Macros.impl
}

import scala.reflect.macros.whitebox.Context
object Macros {
  def impl(c: Context)(annottees: c.Tree*): c.Tree = {
    import c.universe._
    val q"object $name { ..$stats }" = annottees.head
    val main = q"""
      def main(args: Array[String]): Unit = { ..$stats }
    """
    q"object $name { $main }"
  }
}
```

Really hard to write!

```
class main extends scala.annotation.StaticAnnotation {
  def macroTransform(annottees: Any*): Any =
    macro Macros.impl
}

import scala.reflect.macros.whitebox.Context
object Macros {
  def impl(c: Context)(annottees: c.Tree*): c.Tree = {
    import c.universe._
    val q"object $name { ..$stats }" = annottees.head
    val main = q"""
      def main(args: Array[String]): Unit = { ..$stats }
    """
    q"object $name { $main }"
  }
}
```

Don't work with tools well

- If we make a typo in macro code, IDEs won't help us
- If we change macro code, SBT isn't going to recompile

On the other hand, macros are great

- Enable unique functionality
- Frequently used by library authors
- We can't really do without them!

The new future for macros

- We thought real hard how to make macros better
- We found that most of the complexity is incidental
- This is how `scala.meta` was conceived two years ago
- Now that v1.0 is here, it's time to unveil our plans


The Return of the King

The new future for macros

- We got to the essence of macros and found two orthogonal concepts
- `inline` for inlining and `meta` for metaprogramming
- For details: SIP NN: Inline Definitions and Meta Expressions

Example: the new @main macro

```
import scala.meta._

class main extends scala.annotation.StaticAnnotation {
  inline def apply(defn: Any) = meta {
    val q"object $name { ..$stats }" = defn
    val main = q"""
      def main(args: Array[String]): Unit = { ..$stats }
    """
    q"object $name { $main }"
  }
}
```

Example: the new @main macro

```
@main object Test {
  println("hello world")
}
```

Example: the new @main macro

```
@main object Test {
  println("hello world")
}
```

```
meta {
  val q"object $name { ..$stats }" = q"object Test { ... }"
  val main = q"def main(args: Array[String]): Unit = { ..$stats }"
  q"object $name { $main }"
}
```

Example: the new @main macro

```
@main object Test {
  println("hello world")
}
```

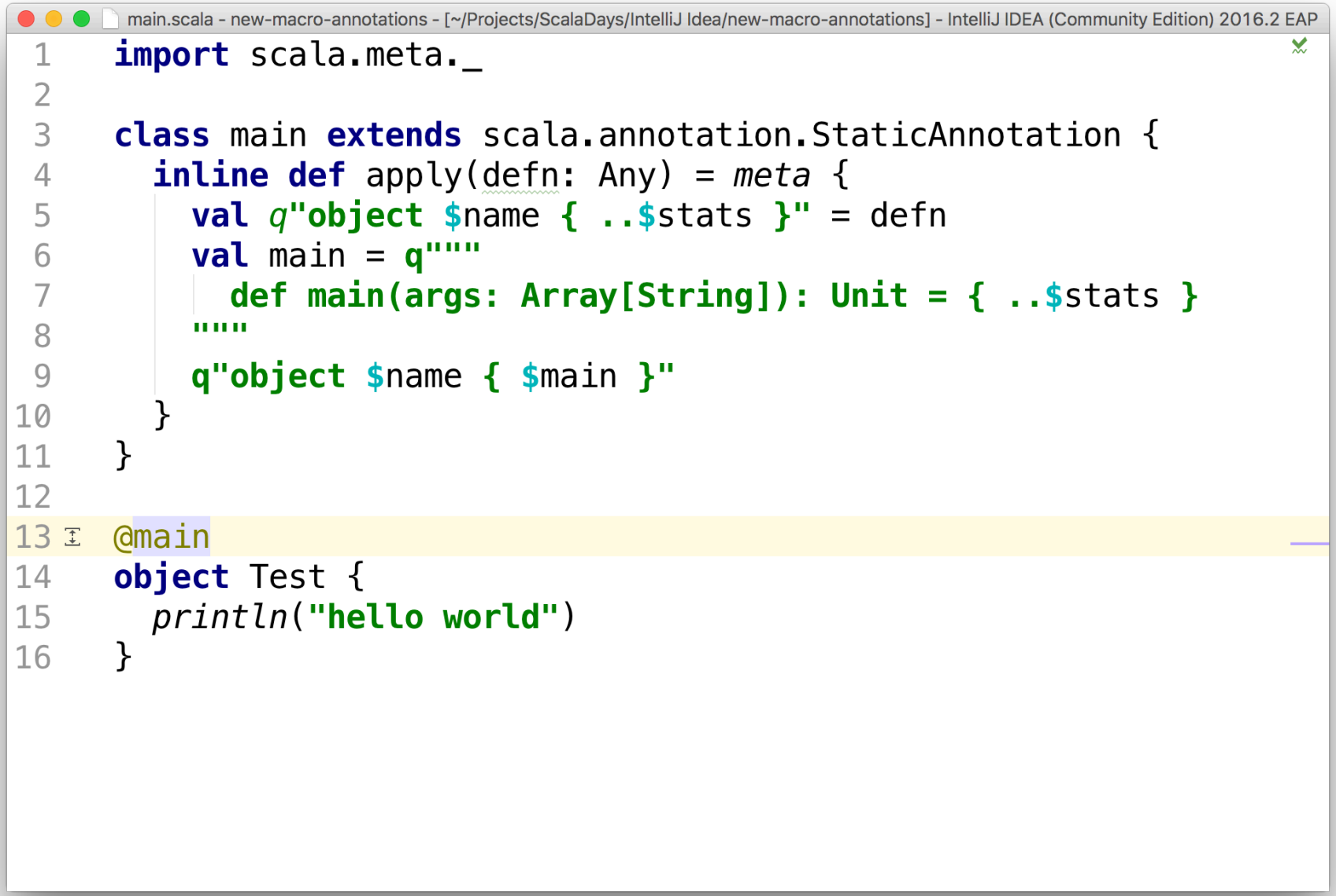
```
meta {
  val q"object $name { ..$stats }" = q"object Test { ... }"
  val main = q"def main(args: Array[String]): Unit = { ..$stats }"
  q"object $name { $main }"
}
```

```
object Test {
  def main(args: Array[String]): Unit = {
    println("hello world")
  }
}
```

Macro support in IntelliJ IDEA

- We've been collaborating since the early days of Scala macros
- Some results are already available in production builds
- Scala.meta provides a principled foundation for macro support

Live demo!


main.scala - new-macro-annotations - [~/Projects/ScalaDays/IntelliJ Idea/new-macro-annotations] - IntelliJ IDEA (Community Edition) 2016.2 EAP
1 import scala.meta._
2
3 class main extends scala.annotation.StaticAnnotation {
4 inline def apply(defn: Any) = meta {
5 val q"object $name { ..$stats }" = defn
6 val main = q"""
7 def main(args: Array[String]): Unit = { ..$stats }
8 """
9 q"object $name { $main }"
10 }
11 }
12
13 @main
14 object Test {
15 println("hello world")
16 }

How IntelliJ macro support works

- IntelliJ provides typechecking and name resolution
- This information flows into scala.meta via an integration layer
- Macro expansions are fed back into IntelliJ
- Live feedback, precise error messages, and other goodies!

Future plans for IntelliJ macro support

- Live resolve of transformed trees
- Looking forward to scala.meta's semantic API
- Interpreting scala.meta programs

Summary

- Macros are not going away
- Will be replaced with a better version based on scala.meta
- Easier to write, better IDE support
- We've got a prototype working, will be upgrading it in the near future

Case study: Dotty linker

Dotty linker

- Experimental whole program optimizer for Scala
- Auto specialization
- Recent addition: rewrite rules
- Check out Dmitry's talk for details

Rewrite rules

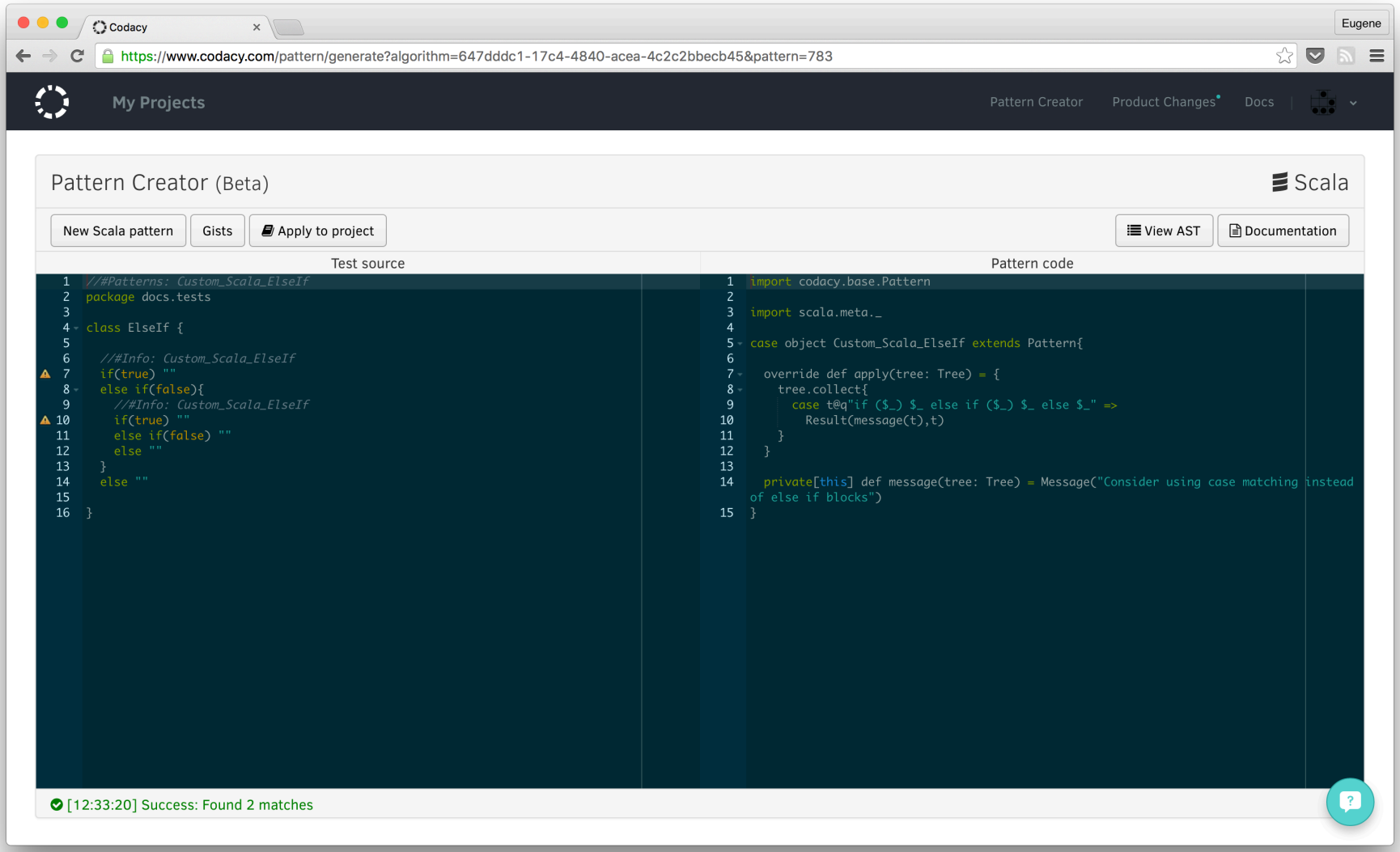
```
import dotty.linker._

@rewrites
object Rewrites {
  def metaRule[T](x: ExistingClass) =
    Rewrite(from = x.existingSlowMethod(...),
            to = meta { /* entry point to scala.meta */ })
}
```

Case study: Codacy

Codacy

- Codacy is a Portuguese startup doing static code analysis
- Web application that connects to GitHub/Bitbucket projects
- Supporting various languages including Scala via open-source tools

Live demo!


Codacy
Eugene
https://www.codacy.com/pattern/generate?algorithm=647dddc1-17c4-4840-acea-4c2c2bbecb45&pattern=783
My Projects
Pattern Creator
Product Changes
Docs
Pattern Creator (Beta)
Scala
New Scala pattern
Gists
Apply to project
View AST
Documentation
Test source
Pattern code
//#Patterns: Custom_Scala_ElseIf
package docs.tests
class ElseIf {
//#Info: Custom_Scala_ElseIf
if(true) ""
else if(false){
//#Info: Custom_Scala_ElseIf
if(true) ""
else if(false) ""
else ""
}
else ""
}
import codacy.base.Pattern
import scala.meta._
case object Custom_Scala_ElseIf extends Pattern{
override def apply(tree: Tree) = {
tree.collect{
case t@q"if ($_) $_ else if ($_) $_ else $_" =>
Result(message(t),t)
}
}
private[this] def message(tree: Tree) = Message("Consider using case matching instead of else if blocks")
}
[12:33:20] Success: Found 2 matches

How Codacy's Scala engine works

- Used to run on scala.reflect toolboxes
- Now using a much more precise scala.meta's parser thanks to the joint work of Johann Egger and Mathieu Demarne
- Code patterns are written with quasiquotes
- More than 80 patterns implemented

Future plans

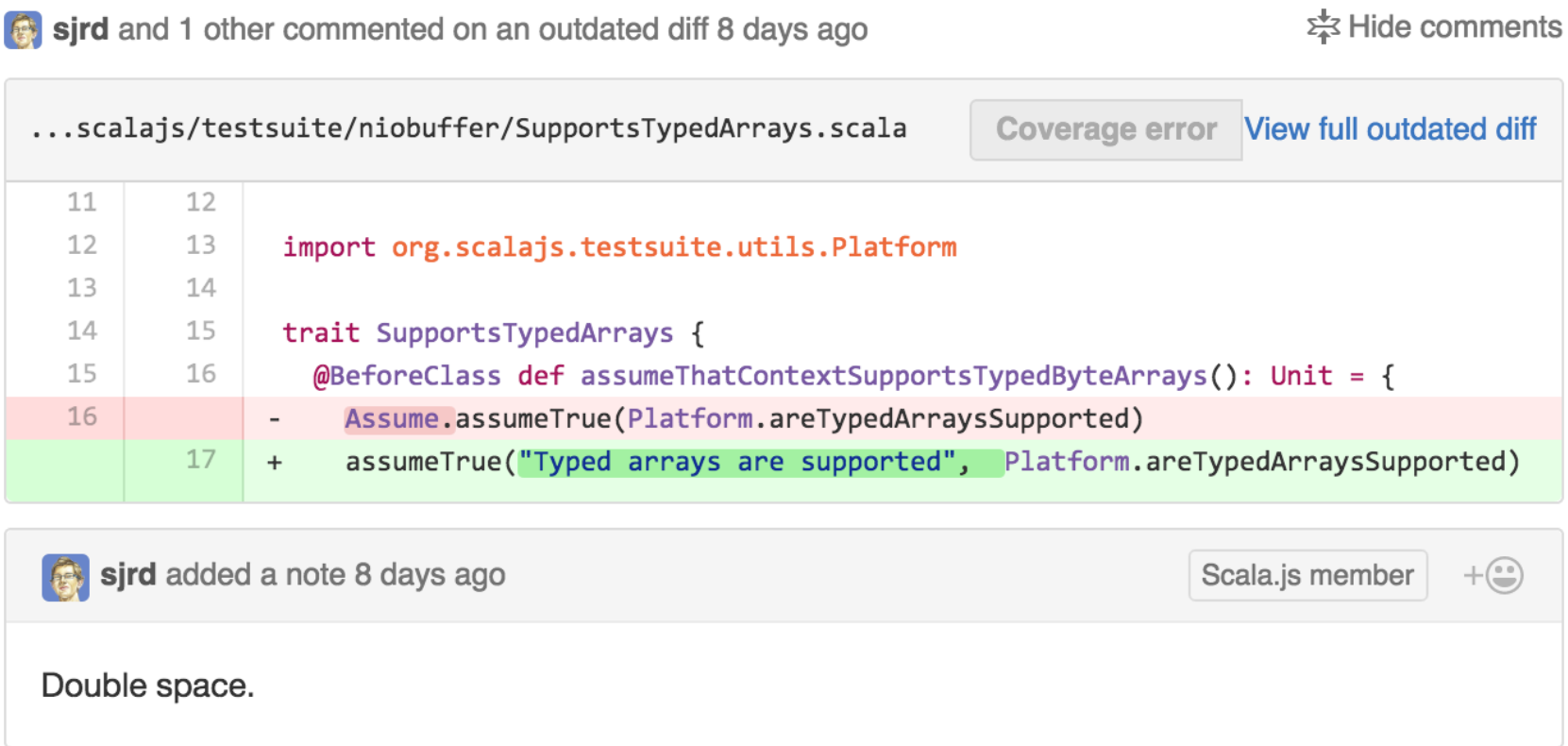
- More patterns, focused on security
- Looking forward to scala.meta's semantic API

Now open-source!

- Most of the Scala patterns were open-sourced yesterday
- Automatic deployment of pull requests is on its way
- Check it out at https://github.com/codacy/codacy-scalameta

Case study: Scalafmt

Better things to do than argue about formatting



Code formatting should be automated

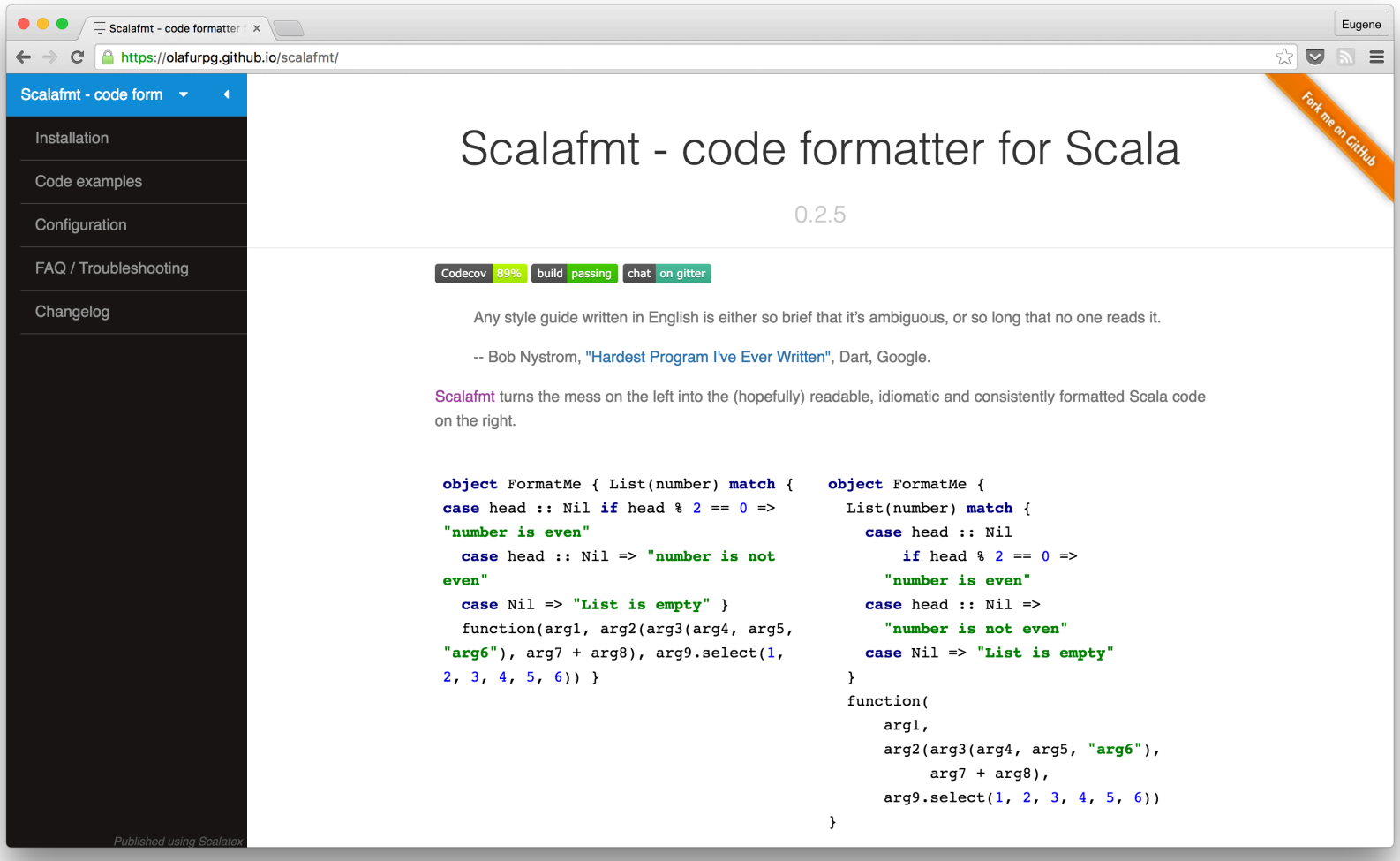
Klangmeister
@viktorklang

Code style should not be enforced by review, but by automate rewriting. Evolve the style using PRs against the rewriting config.

RETWEETS 23 LIKES 36

9:00 AM - 7 Feb 2016

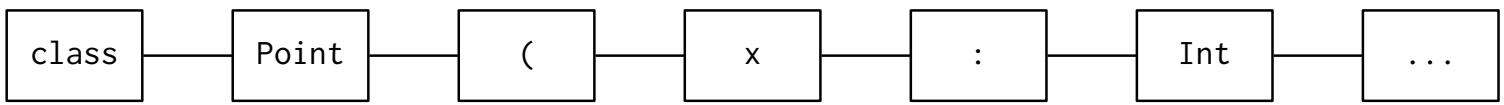
Live demo!


Scalafmt - code formatter
Eugene
https://olafurpg.github.io/scalafmt/
Scalafmt - code form
Installation
Code examples
Configuration
FAQ / Troubleshooting
Changelog
Published using Scalatex
Fork me on GitHub
Scalafmt - code formatter for Scala
0.2.5
Codecov 89%
build passing
chat on gitter
Any style guide written in English is either so brief that it's ambiguous, or so long that no one reads it.
-- Bob Nystrom, "Hardest Program I've Ever Written", Dart, Google.
Scalafmt turns the mess on the left into the (hopefully) readable, idiomatic and consistently formatted Scala code on the right.
object FormatMe { List(number) match { case head :: Nil if head % 2 == 0 => "number is even" case head :: Nil => "number is not even" case Nil => "List is empty" } function(arg1, arg2(arg3(arg4, arg5, "arg6"), arg7 + arg8), arg9.select(1, 2, 3, 4, 5, 6)) }
object FormatMe { List(number) match { case head :: Nil if head % 2 == 0 => "number is even" case head :: Nil => "number is not even" case Nil => "List is empty" } function(arg1, arg2(arg3(arg4, arg5, "arg6"), arg7 + arg8), arg9.select(1, 2, 3, 4, 5, 6)) }

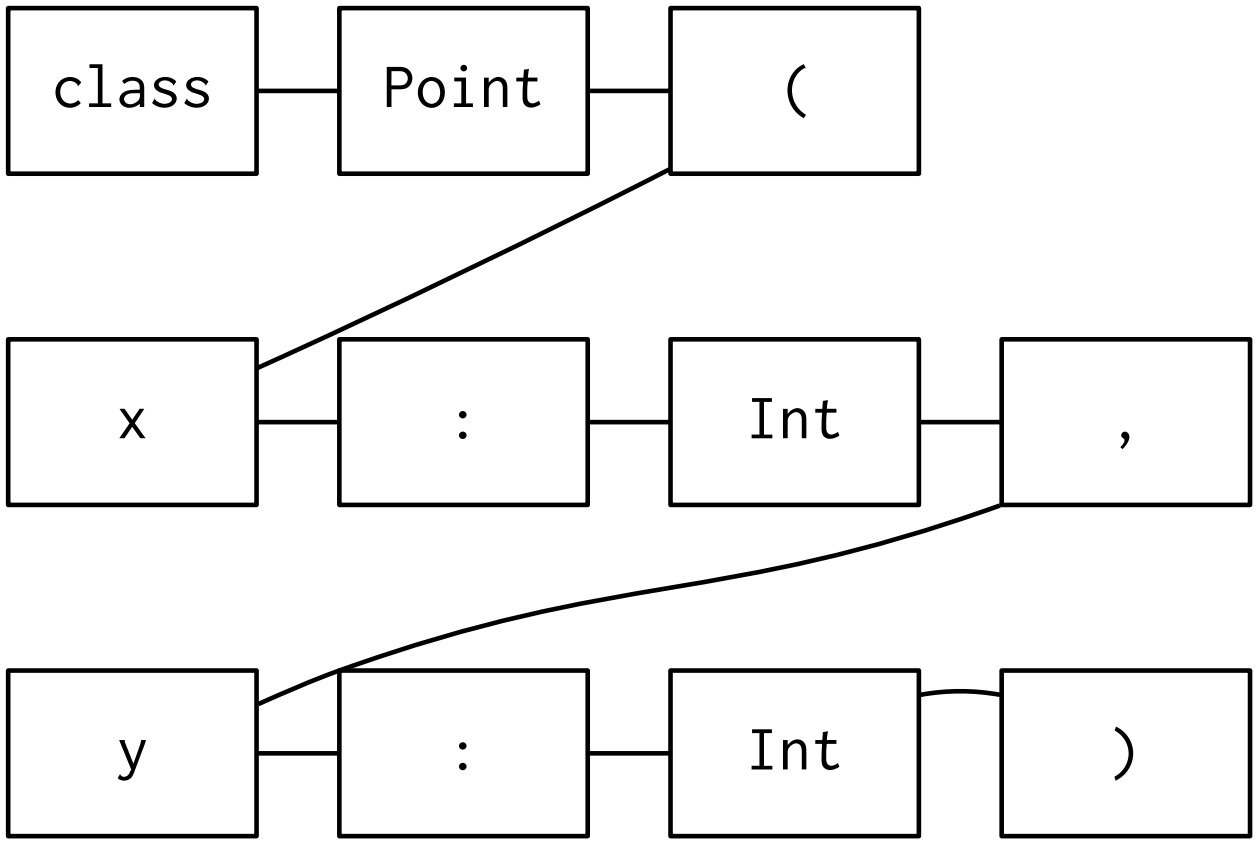
How scalafmt works: example snippet

```
class Point(x: Int, y: Int)
```

How scalafmt works: unformatted tokens



How scalafmt works: formatted tokens



Future plans

- Out-of-the-box presets supporting popular code styles
- Support for diff formatting
- Performance improvements

Part 3: What will happen next?

Graduation

- It's been a fun five PhD years at EPFL
- But it's time to wrap it up
- Thesis defense incoming in 3... 2... 1...

My next adventure

Eugene Burmako @xeno_by · Mar 7

Totally psyched to announce that this fall I'll be joining Twitter's Engineering Effectiveness group: scalamacros.org/news/2016/03/0 …

58 161

The arrangement with Twitter

- 50% of the time: working on my open-source Scala projects
- 50% of the time: making Scala tooling at Twitter better

Next steps with scala.meta

- Semantic API: typechecking, name resolution, implicit inference, etc
- New-style ("inline") macros
- Other execution environments (2.10, 2.12, scala.js)

Summary

Scala Macros

- Macros are not going away
- Will be replaced with a better version based on scala.meta
- I'm graduating soon, but still going to be the project lead

Scala Meta

- The project is officially endorsed and funded
- Just released v1.0
- Jetbrains, Codacy and Scalafmt are already building cool stuff with it
- Join us in our quest for next-generation tooling!