Version 2.0
(24th June 2014)

Macros for the rest of us

Dave Gurnell, Underscore
@davegurnell, http://underscore.io

Macros for the rest of us

Scala 2.11
(also Scala 2.10 via Macro Paradise)

Blackbox def macros

No prior knowledge of macros
(just basic Scala)

Full code available at…

underscore.io

What is a macro?

What is a macro?

A ***method-like construct*** that executes ***at compile time*** and ***transforms the code*** in our program.

What is a macro?

Useful for ***code generation***, ***static checking***, and ***domain specific languages***.

Eugene Burmako -- What are macros good for?
https://skillsmatter.com/skillscasts/4920-what-are-macros-good-for

Example: Maximum

project maximum in the code

Example: Maximum

```scala
def maximum(a: Int, b: Int): Int =
  macro maximumMacro


def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  q"if($a > $b) $a else $b"
}
```

Example: Maximum

```
def maximum(a: Int, b: Int): Int =
  macro maximumMacro


def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  q"if($a > $b) $a else $b"
}
```

Example: Maximum

```
def maximum(a: Int, b: Int): Int =
  macro maximumMacro
```

```
def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  q"if($a > $b) $a else $b"
}
```

Example: Maximum

```
def maximum(a: Int, b: Int): Int =
  macro maximumMacro
```

```
def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  q"if($a > $b) $a else $b"
}
```

Example: Maximum

```
def maximum(a: Int, b: Int): Int =
  macro maximumMacro


def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  q"if($a > $b) $a else $b"
}
```

Example: Maximum

```scala
def maximum(a: Int, b: Int): Int =
  macro maximumMacro


def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  q"if($a > $b) $a else $b"
}
```

Example: Maximum

```
val x = 1
val y = 2

println(maximum(x, y))
```

Example: Maximum

```scala
val x = 1
val y = 2

println(maximum(x, y))


println(if(x > y) x else y)
```

Example: Maximum

```
val x = 1
val y = 2

println(maximum(x + 10, y * 3))
```

Example: Maximum

```
val x = 1
val y = 2

println(maximum(x + 10, y * 3))

println(
  if((x + 10) > (y * 3))
    x + 10
  else
    y * 3
)
```

Example: Maximum

```
val x = 1
val y = 2

println(maximum(x + 10, y * 3))

println(
  if((x + 10) > (y * 3))
    x + 10
  else
    y * 3
)
```

Can you spot the bug?!

Example: Maximum

```
val x = 1

println(maximum(x, randomInt()))
```

Example: Maximum

```
val x = 1


println(maximum(x, randomInt()))

println(
  if(x > randomInt())
    x
  else
    randomInt()
)
```

Example: Maximum

Version 2.0
(errata fixed)

```scala
def maximum(a: Int, b: Int): Int =
  macro maximumMacro


def maximumMacro(c: Context)(a: c.Tree, b: c.Tree): c.Tree = {
  import c.universe._
  val temp1 = c.freshName(TermName("temp"))
  val temp2 = c.freshName(TermName("temp"))
  q"""
  val $temp1 = $a
  val $temp2 = $b
  if($temp1 > $temp1) $temp1 else $temp2
  """
}
```

Ask the compiler to allocate variable names

There are gotchas here: see here for info

See the following (steps 10 onwards) for a discussion of name generation:
https://github.com/scalamacros/macrology201

Example: Maximum

```
val x = 1

println(maximum(x, randomInt()))
```

Example: Maximum

```
val x = 1


println(maximum(x, randomInt()))

println {
  val temp1 = x
  val temp2 = randomInt()
  if(temp1 > temp2) temp1 else temp2
}
```

Example: Maximum

Take home points

Macros — generate code

Implementation — is just tree substitution

Semantics are important — make sure you generate what developers would expect

Setup

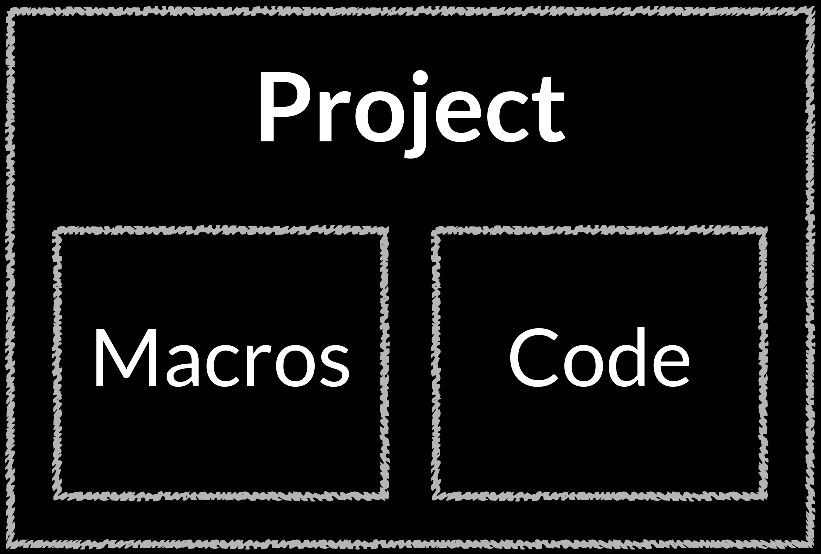
Setup: Projects

Setup: Projects

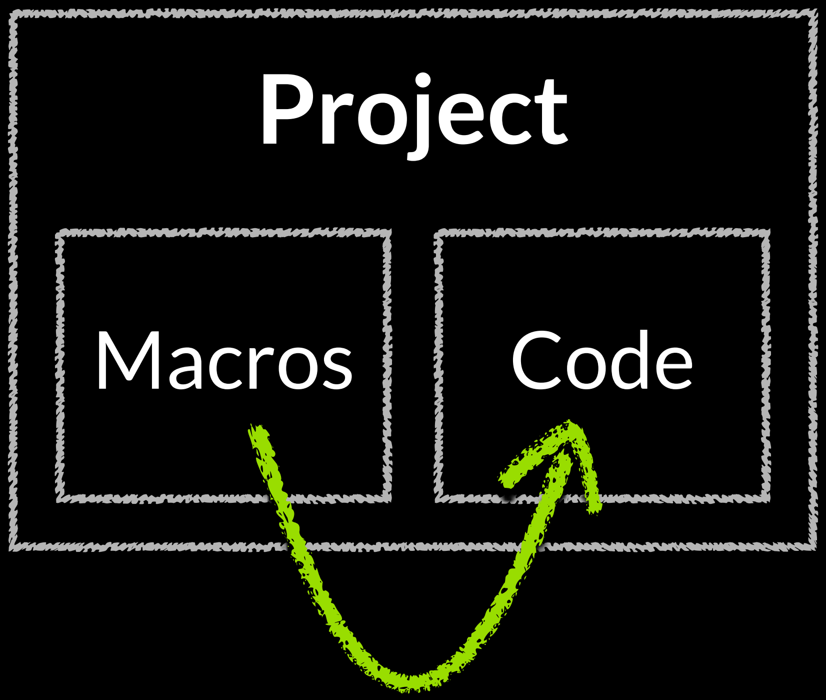
Separate Compilation

Macros cannot be ***defined*** and ***used***
in the same ***compilation unit***.

Setup: Projects



Setup: Projects

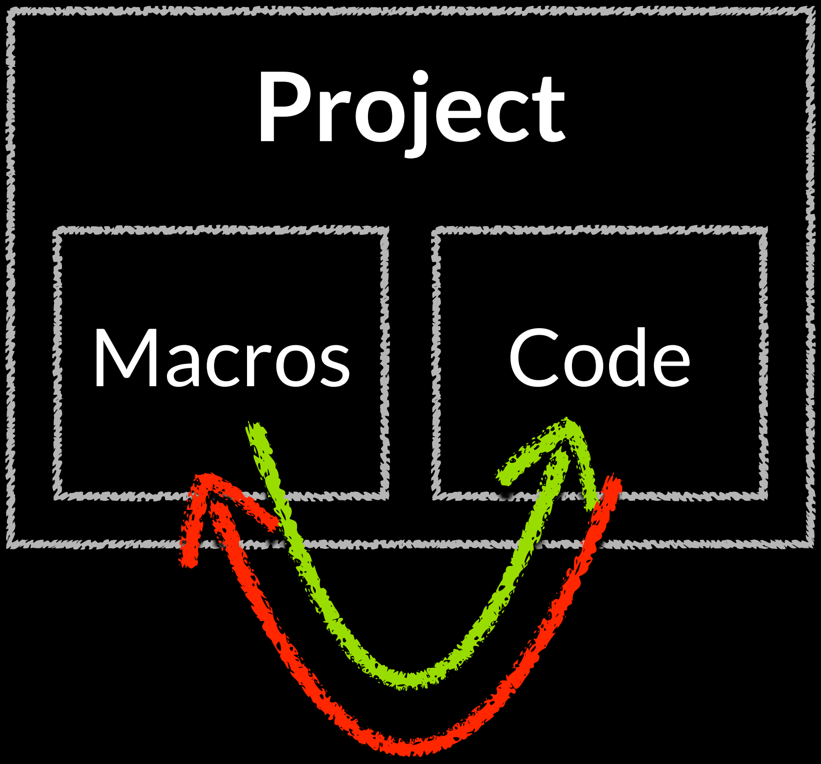


Project

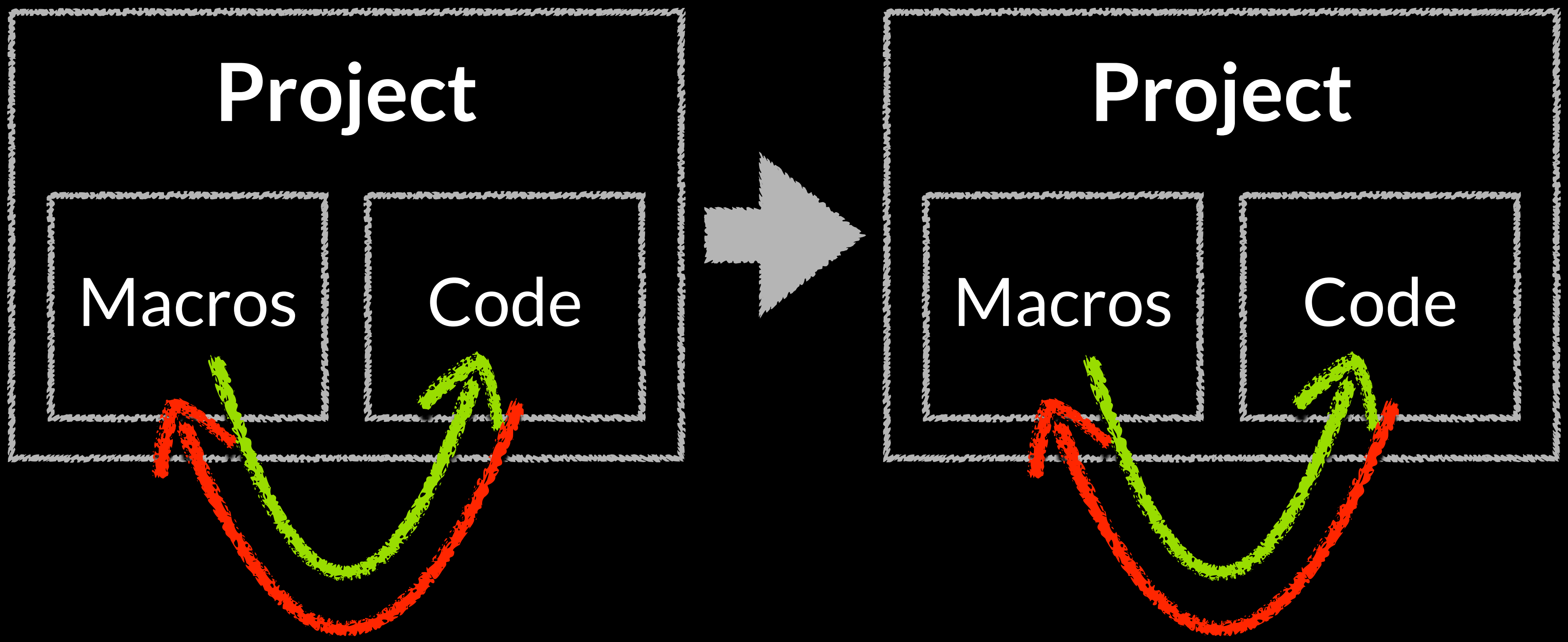
Macros

Code

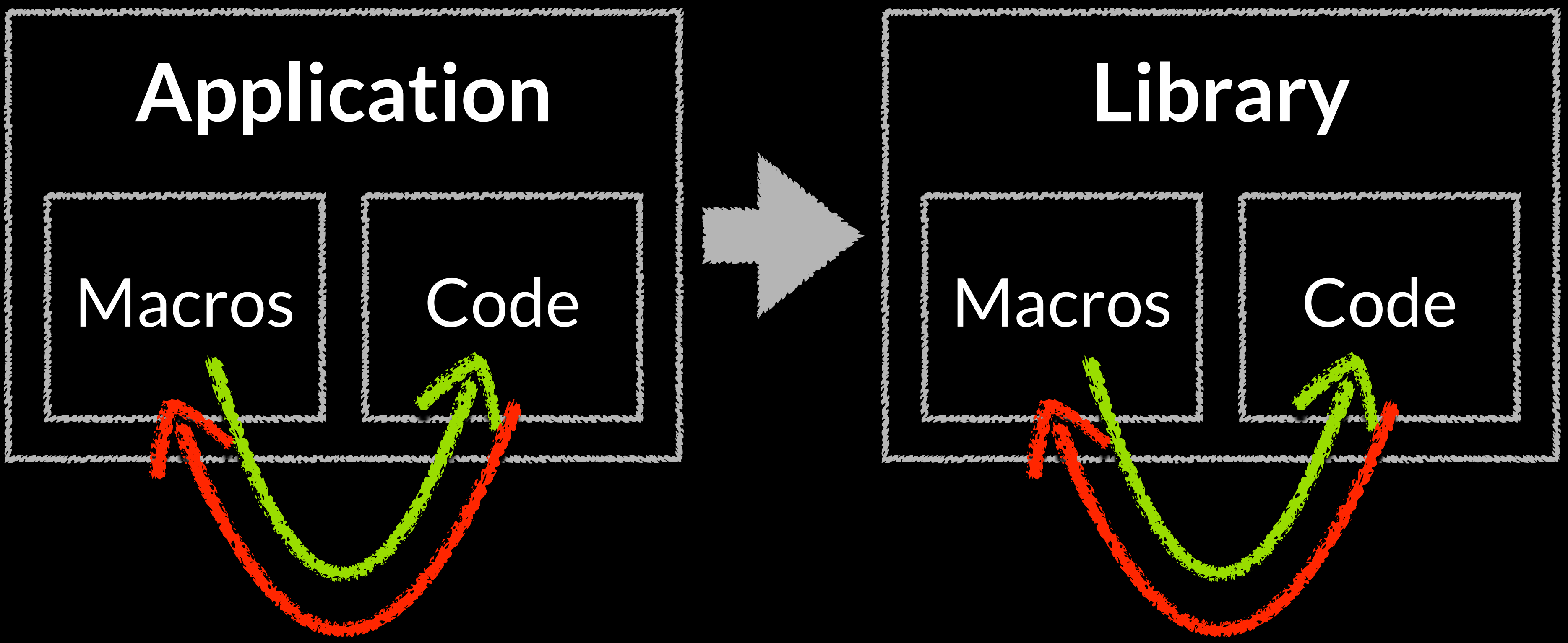
Setup: Projects



Setup: Projects



Setup: Projects



Application

Macros

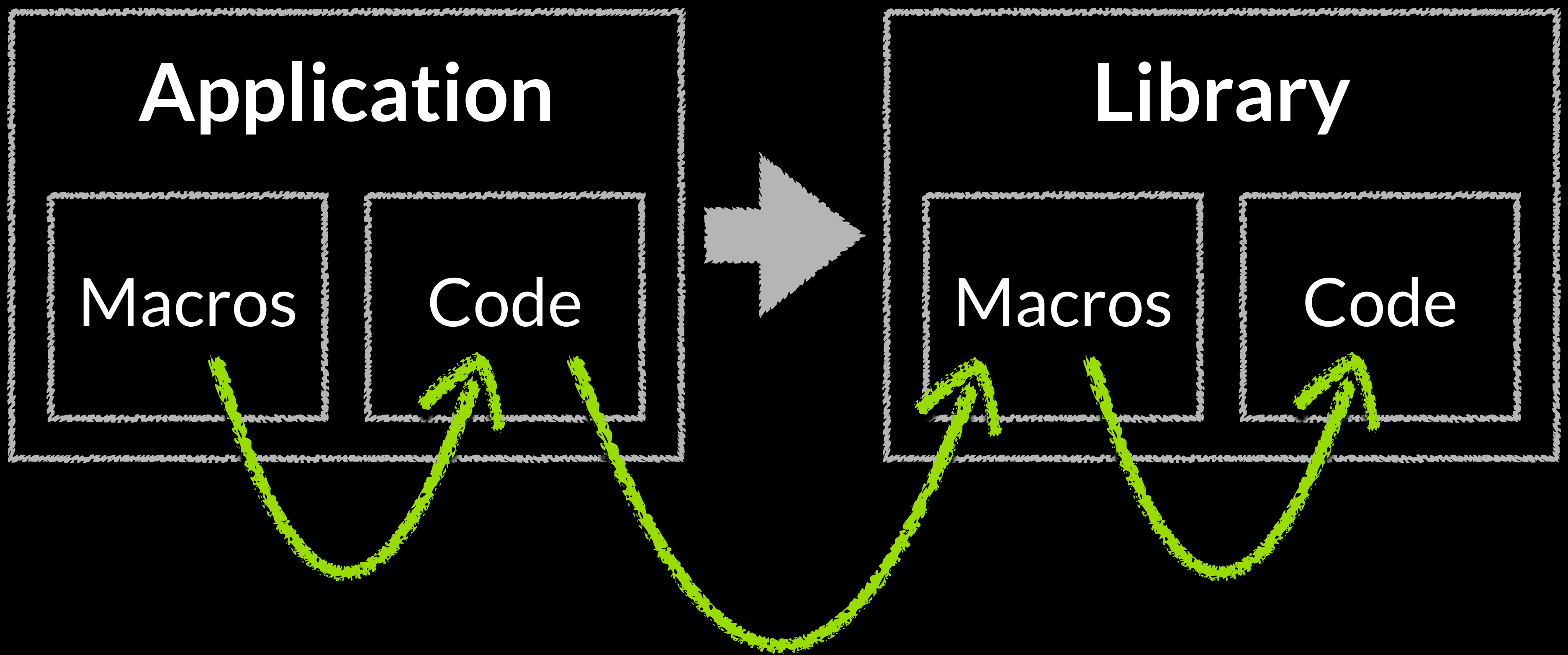
Code

Library

Macros

Code

Setup: Projects



Application

Macros

Code

Library

Macros

Code

Setup: Code

Setup: Code

```
import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object Macros {
  def maximum(a: Int, b: Int): Int =
    macro maximumMacro

  def maximumMacro(c: Context)(a: c.Tree, b: c.Tree) = {
    import c.universe._
    // ...
  }
}
```

Setup: Code

```
import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object Macros {
  def maximum(a: Int, b: Int): Int =
    macro maximumMacro

  def maximumMacro(c: Context)(a: c.Tree, b: c.Tree) = {
    import c.universe._
    // ...
  }
}
```

Setup: Code

```
import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object Macros {
  def maximum(a: Int, b: Int): Int =
    macro maximumMacro

  def maximumMacro(c: Context)(a: c.Tree, b: c.Tree) = {
    import c.universe._
    // ...
  }
}
```

Setup: Code

```scala
import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object Macros {
  def maximum(a: Int, b: Int): Int =
    macro maximumMacro

  def maximumMacro(c: Context)(a: c.Tree, b: c.Tree) = {
    import c.universe._
    // ...
  }
}
```

Setup: Code

```
import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object Macros {
  def maximum(a: Int, b: Int): Int =
    macro maximumMacro

  def maximumMacro(c: Context)(a: c.Tree, b: c.Tree) = {
    import c.universe._
    // ...
  }
}
```

Setup: Code

```scala
import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object Macros {
  def maximum(a: Int, b: Int): Int =
    macro maximumMacro

  def maximumMacro(c: Context)(a: c.Tree, b: c.Tree) = {
    import c.universe._
    // ...
  }
}
```

Setup

Take home points

Separate compilation — deal with it!

Macro definitions — come in two parts:
declaration and *implementation*

Macro API — import from *Context* and *Universe*

Trees

Creating Trees

Creating Trees

```
if(x > y) x else y
```

Creating Trees

```
if(x > y) x else y
```

```
val tree: Tree =
  If(
    Apply(
      Select(
        Ident(TermName("x")),
        TermName("$greater")),
      List(Ident(TermName("y")))),
    Ident(TermName("x")),
    Ident(TermName("y")))
```

Example: Printing Trees

project printtree in the code

Jason Zaugg — Macrocosm — “desugar” macro
https://github.com/retronym/macrocosm

Example: Printing Trees

```scala
def printTree(title: String)(expr: Any): Unit =
  macro printTreeMacro

def printTreeMacro(c: Context)(title: c.Tree)(expr: c.Tree) = {
  import c.universe._

  val code : String = showCode(expr)
  val ast  : String = showRaw(expr)

  q"""
  println(
    $title.toUpperCase + "\n\n" +
    $code              + "\n\n" +
    $raw               + "\n\n"
  )
  """
}
```

Example: Printing Trees

```
def printTree(title: String)(expr: Any): Unit =
  macro printTreeMacro

def printTreeMacro(c: Context)(title: c.Tree)(expr: c.Tree) = {
  import c.universe._

  val code : String = showCode(expr)
  val ast  : String = showRaw(expr)

  q"""
  println(
    $title.toUpperCase + "\n\n" +
    $code              + "\n\n" +
    $raw               + "\n\n"
  )
  """
}
```

Example: Printing Trees

```
def printTree(title: String)(expr: Any): Unit =
  macro printTreeMacro

def printTreeMacro(c: Context)(title: c.Tree)(expr: c.Tree) = {
  import c.universe._

  val code : String = showCode(expr)
  val ast  : String = showRaw(expr)

  q"""
  println(
    $title.toUpperCase + "\n\n" +
    $code              + "\n\n" +
    $raw               + "\n\n"
  )
  """
}
```

Example: Printing Trees

```scala
def printTree(title: String)(expr: Any): Unit =
  macro printTreeMacro

def printTreeMacro(c: Context)(title: c.Tree)(expr: c.Tree) = {
  import c.universe._

  val code : String = showCode(expr)
  val ast   : String = showRaw(expr)

  q"""
  println(
    $title.toUpperCase + "\n\n" +
    $code              + "\n\n" +
    $raw               + "\n\n"
  )
  """
}
```

Example: Printing Trees

```
printTree("Integer literal") {
  123
}

printTree("Simple block") {
  123
  234
  345
}

printTree("Simple expression") {
  x > y
}
```

Code

```
printTree("Integer literal") {
  123
}

printTree("Simple block") {
  123
  234
  345
}

printTree("Simple expression") {
  x > y
}
```

Tree

```
Literal(Constant(123))


Block(
  List(
    Literal(Constant(123)),
    Literal(Constant(234))
  ),
  Literal(Constant(345)))

Apply(
  Select(
    Ident(TermName("x")),
    TermName("$greater")),
  List(Ident(TermName(“y"))))
```

Printing Trees

Take home points

Trees — are the bread and butter of macros

showCode / *showRaw* — are useful
for understanding common tree structures

printTree — (or equivalent)
is an essential part of your toolchain

Quasiquotes

Quasiquotes

```
val tree: Tree =
  If(
    Apply(
      Select(
        Ident(TermName("x")),
        TermName("$greater")),
      List(Ident(TermName("y")))),
    Ident(TermName("x")),
    Ident(TermName("y")))
```

Quasiquotes

```
val tree: Tree =
  q"""
   if(x > y) x else y
   """
```

Quasiquotes

```
val a: Tree = q"x"

val b: Tree = q"y"


val tree: Tree =
  q"""
   if($a > $b) $a else $b
   """



// => q"if(x > y) x else y"
```

Quasiquotes

```
val exprs = List(
  q"x",
  q"x*2",
  q"x+10")

val tree =
  q"println(Seq(..$exprs))"

// => println(Seq(x, x*2, x+10))
```

Quasiquotes

```
val a: Int    = 1

val b: Double = 2.0


val tree: Tree =
  q"""
   if($a > $b) $a else $b
   """

// => if(1 > 2.0) 1 else 2.0
```

Quasiquotes

```
val a: Int    = 1

val b: Double = 2.0


val tree: Tree =
  q"""
   if($a > $b) $a else $b
   """

// => if(1 > 2.0) 1 else 2.0
```

Liftable[Int]
Liftable[Double]

Quasiquotes

```
val tree           = q"Foo[Bar]"
```

What does this mean?

Quasiquotes

```
val tree            = q"Foo[Bar]"


val useCase1 = q"val a: $tree  = ..."
```

Quasiquotes

```
val tree           = q"Foo[Bar]"
```

```
val useCase1 = q"val a: $tree  = ..."
val useCase2 = q"val b = $tree"
```

Quasiquotes

```
val tree           = q"Foo[Bar]"



val useCase1 = q"val a: $tree  = ..."    ✗

val useCase2 = q"val b = $tree"          ✓
```

Quasiquotes

```
val exprTree       = q"Foo[Bar]"
               // == Foo.apply[Bar]
```

Quasiquotes

```
val exprTree       = q"Foo[Bar]"
               // == Foo.apply[Bar]

val typeTree       = tq"Foo[Bar]"

val patternTree    = pq"a: Int"

val caseClauseTree = cq"a: Int => println(a)"

val forEnumTree    = fq"i <- 1 to 10"
```

Quasiquotes

Take home points

Quasiquotes — are a quick way to build trees

Substitution — “$” and “..$”

Liftables — automatically convert basic data types to trees for you

Interpolators — q“…”, tq“…”, pq“…”, cq“…”, fq“…”

Inspecting Trees

Example: Simple Assert

project simpleassert in the code

Example: Simple Assert

```
val a = 123
val b = 234

assert(a == b)
```

Example: Simple Assert

```scala
val a = 123
val b = 234

assert(a == b)

// java.lang.AssertionError: assertion failed
//   at scala.Predef$.assert(Predef.scala:165)
//   etc...
```

Example: Simple Assert

```
val a = 123
val b = 234

assert(a == b)

// java.lang.AssertionError: 123 != 234
//   at scala.Predef$.assert(Predef.scala:165)
//   etc...
```

Example: Simple Assert

```scala
def assert(expr: Boolean): Unit =
  macro assertMacro

def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  // ...
}
```

Example: Simple Assert

```
def assert(expr: Boolean): Unit =
  macro assertMacro

def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  // ...
}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  // ...














}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """





    case other =>
      // ...
  }
}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """





    case other =>
      // ...
  }
}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """






    case other =>
      // ...
  }
}
```

Example: Simple Assert

```scala
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """





    case other =>
      // ...
  }
}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """




    case other =>
      // ...
  }
}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """




    case other =>
      // ...
  }
}
```

Example: Simple Assert

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      q"""
      if($a != $b) {
        throw new AssertionError($a + " != " + $b)
      }
      """




    case other =>
      // ...
  }
}
```

Can you spot the bug?!

Example: Simple Asser

Version 2.0
(errata fixed)

```
def assertMacro(c: Context)(expr: c.Tree) = {
  import c.universe._

  expr match {
    case q"$a == $b" =>
      val temp1 = c.freshName(TermName("temp"))
      val temp2 = c.freshName(TermName("temp"))
      q"""
      val $temp1 = $a
      val $temp2 = $b
      if($temp1 != $temp2) {
        throw new AssertionError($temp1 + "!= " + $temp2)
      }
      """

    case other =>
      // ...
  }
}
```

See the following (steps 10 onwards) for a discussion of name generation:
https://github.com/scalamacros/macrology201

Example: Simple Assert

```
val a = 123
val b = 234

assert(a == b)

// java.lang.AssertionError: 123 != 234
//   etc…



assert(false)

// java.lang.AssertionError: assertion failed
//   etc...
```

Example: Better Assert

project betterassert in the code

Example: Better Assert

```
assert(a.b == c.d(e, f))

// java.lang.AssertionError: ???
```

Example: Better Assert

```
assert(a.b == c.d(e, f))

// java.lang.AssertionError:
//   a.b       = ...
//   c.d(e, f) = ...
```

Example: Better Assert

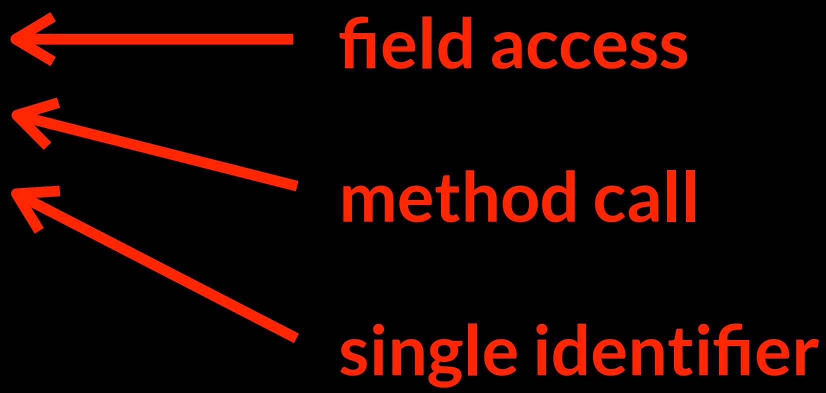
```
assert(a.b == c.d(e, f))

// java.lang.AssertionError:
//   a.b       = ...
//   c.d(e, f) = ...
//   a         = ...
//   c         = ...
//   e         = ...
//   f         = ...
```

Example: Better Assert

```
assert(a.b < c.d(e, f))

// java.lang.AssertionError:
//   a.b       = ...
//   c.d(e, f) = ...
//   a         = ...
//   c         = ...
//   e         = ...
//   f         = ...
```

Example: Better Assert

```
assert(a.b < c.d(e, f))

// java.lang.AssertionError:
//   a.b        = ...
//   c.d(e, f)  = ...
//   a          = ...
//   c          = ...
//   e          = ...
//   f          = ...
```



field access

method call

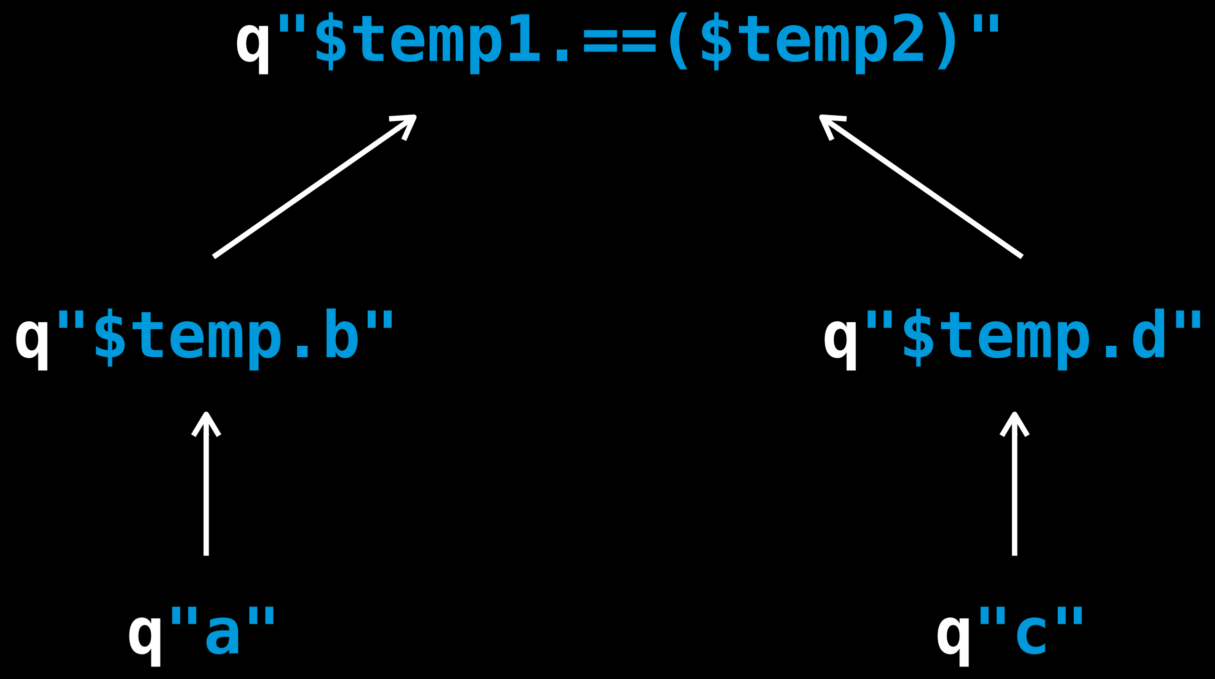
single identifier

Example: Better Assert

```
expr match {
  case q"$recv.$method(..$args)" =>
    // ... method call ...

  case q"$recv.$field" =>
    // ... field access ...

  case ident: Ident =>
    // ... single identifier ...

  case other => other
}
```

Example: Better Assert

```
assert(a.b == c.d)
```



```
q"$temp1.==($temp2)"
```

```
q"$temp.b"
```

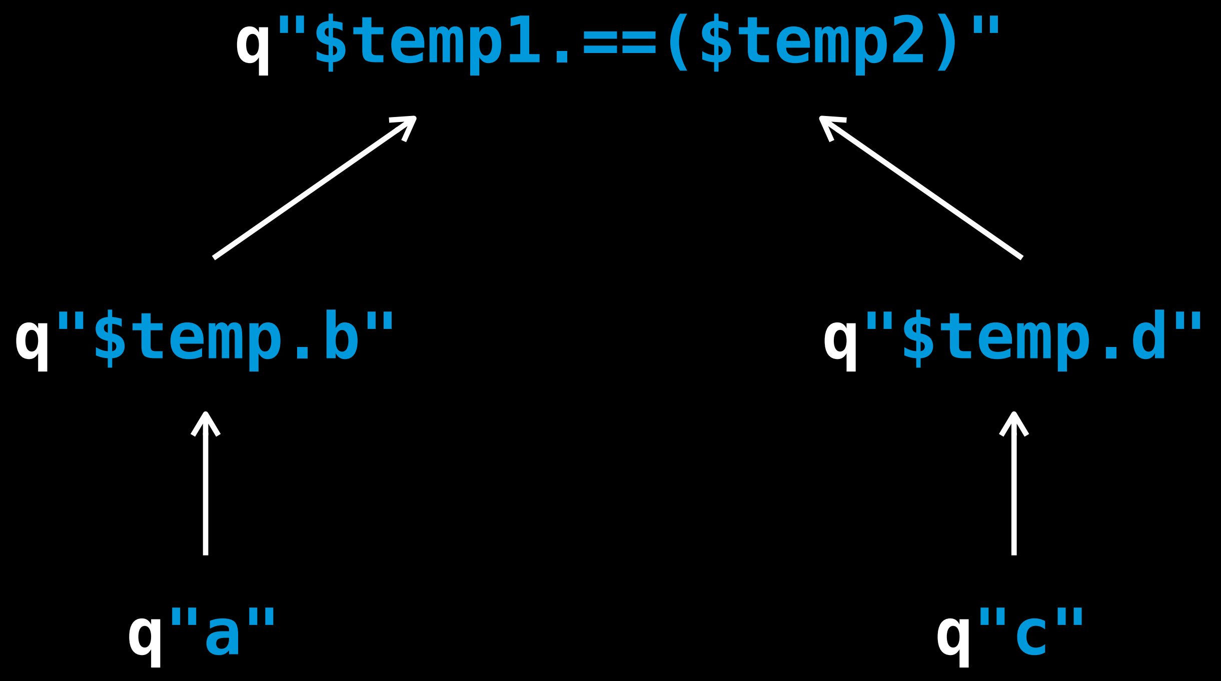
```
q"$temp.d"
```

```
q"a"
```

```
q"c"
```

Example: Better Assert

```
assert(a.b == c.d)
```



```
q"$temp1.==($temp2)"

q"$temp.b"        q"$temp.d"

q"a"              q"c"
```

```
val temp1 = a
val temp2 = temp1.b
val temp3 = c
val temp4 = temp3.d
val temp5 = temp2 == temp4

if(!temp5)
  throw new AssertionError(
    "a   = " + temp1 +
    "a.b = " + temp2 +
    /* etc... */
```

Example: Better Assert

```
assert(a.b < c.d(e, f))

// java.lang.AssertionError:
//   a         = ...
//   a.b       = ...
//   c         = ...
//   e         = ...
//   f         = ...
//   c.d(e, f) = ...
```

Example: Better Assert

Take home points

Quasiquotes — can be used to
pattern match against tree structures

Tree traversal — can be used to
pick apart complex expressions

Error handling / passthrough — is important!
(we often can't anticipate all possible scenarios)

Types

Types

```scala
def foo[A]: Unit = macro fooMacro[A]

def fooMacro[A: c.WeakTypeTag](c: Context): c.Tree = {
  import c.universe._

  val tpe: Type = c.weakTypeOf[A]

  // ...
}
```

Types

```
def foo[A]: Unit = macro fooMacro[A]

def fooMacro[A: c.WeakTypeTag](c: Context): c.Tree = {
  import c.universe._

  val tpe: Type = c.weakTypeOf[A]

  // ...
}
```

Types

Type tags — concretely known types

Weak type tags — partially known types

Types

Type tags — concretely known types

`String` or `List[Int]`

Weak type tags — partially known types

`T` or `List[A]`

Types

Type tags — concretely known types

`String` or `List[Int]`

```
val t: Type = c.typeOf[Option[Int]]
```

Weak type tags — partially known types

`T` or `List[A]`

```
val t: Type = c.weakTypeOf[A]
```

Types

```
def foo[A]: Unit = macro fooMacro[A]

def fooMacro[A: c.WeakTypeTag](c: Context): c.Tree = {
  import c.universe._

  val tpe: Type = c.weakTypeOf[A]

  // ...
}
```

Types

```
foo[List[Int]]

foo[MyCaseClass]
```

Types

fully known

```
foo[List[Int]]

foo[MyCaseClass]
```

Types

fully known

```
foo[List[Int]]

foo[MyCaseClass]
```

```
def someMethod[X] =
  foo[X]
```

Types

fully known

```
foo[List[Int]]

foo[MyCaseClass]
```

```
def someMethod[X] =
  foo[X]
```

partially known

Types

```
val a: Type = c.weakTypeOf[A]

t.decls
```

Types

```
val a: Type = c.weakTypeOf[A]
```

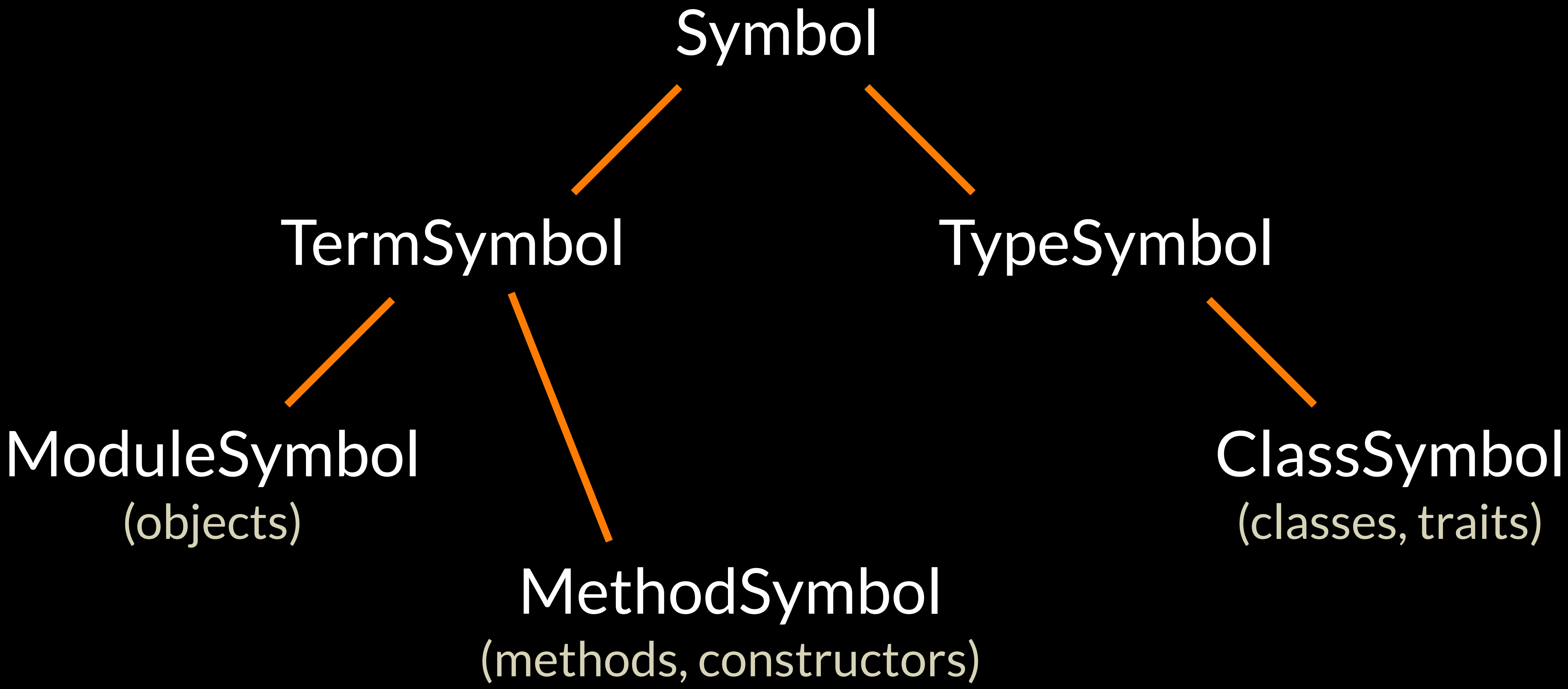
```
t.decls
```

Symbols

Declarations

(constructors, methods, fields, etc)

Inspecting Types



Symbol

TermSymbol

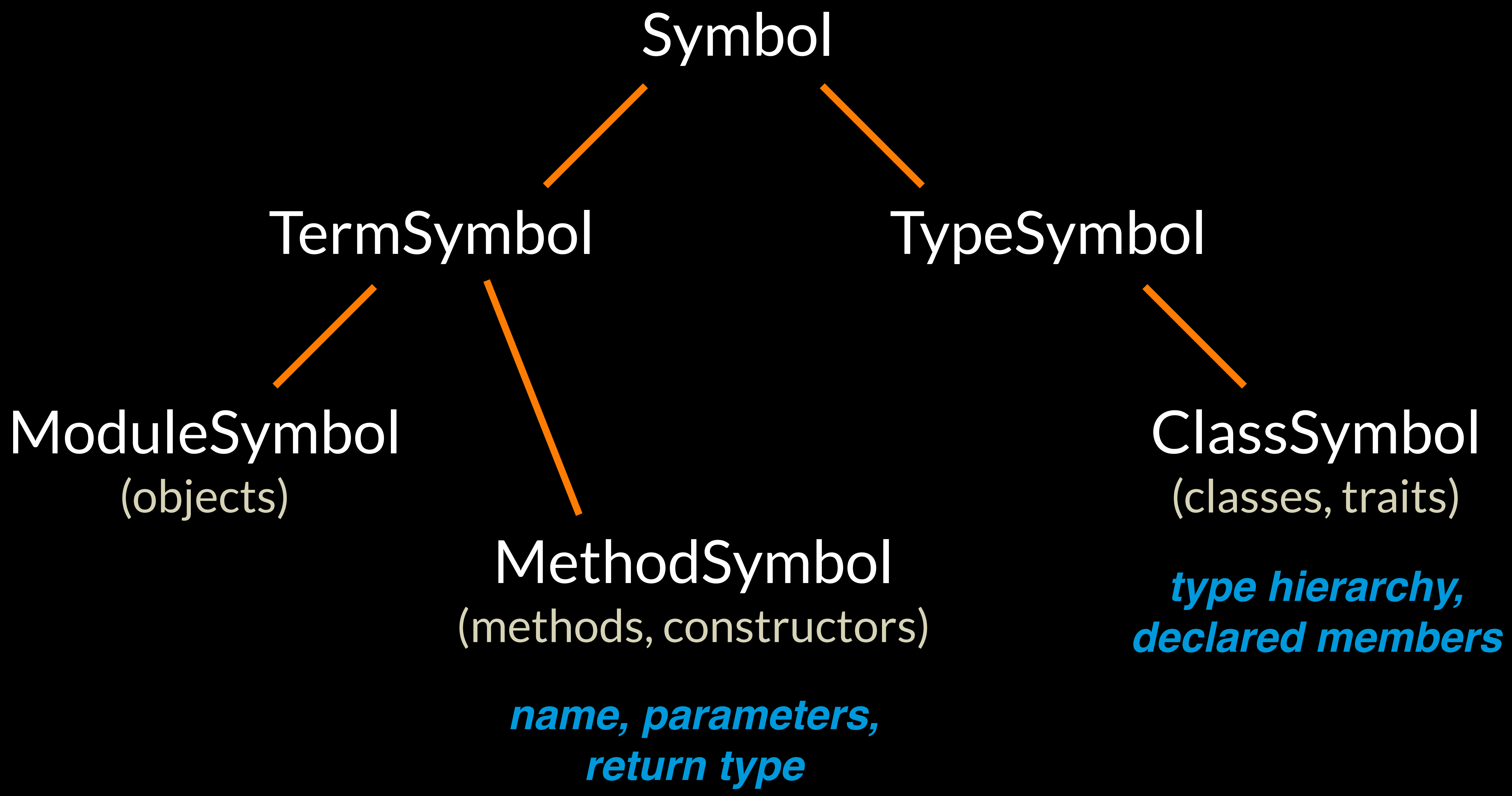
TypeSymbol

ModuleSymbol
(objects)

MethodSymbol
(methods, constructors)

ClassSymbol
(classes, traits)

Inspecting Types



Symbol

TermSymbol

TypeSymbol

ModuleSymbol
(objects)

MethodSymbol
(methods, constructors)

name, parameters,
return type

ClassSymbol
(classes, traits)

type hierarchy,
declared members

Example: Orderings

project orderings in the code

Example: Orderings

```scala
case class Person(name: String, age: Int)

val people = List(
  Person("Anne",    35),
  Person("Bob",     45),
  Person("Charlie", 20))
```

Example: Orderings

```scala
case class Person(name: String, age: Int)

val people = List(
  Person("Anne",    35),
  Person("Bob",     45),
  Person("Charlie", 20))
```

http://api.example.com/people**?sort=name**

Example: Orderings

```scala
case class Person(name: String, age: Int)

val people = List(
  Person("Anne",    35),
  Person("Bob",     45),
  Person("Charlie", 20))



people.sorted(/* Ordering[Person] */)
```

Example: Orderings

```
case class Person(name: String, age: Int)

val people = List(
  Person("Anne",    35),
  Person("Bob",     45),
  Person("Charlie", 20))

def by(field: String): Ordering[Person] =
  ???

people.sorted(/* Ordering[Person] */)
```

Example: Orderings

```
case class Person(name: String, age: Int)

val people = List(
  Person("Anne",    35),
  Person("Bob",     45),
  Person("Charlie", 20))

def by(field: String): Ordering[Person] =
  ???

people sorted by("name")
```

Example: Orderings

```
def by(field: String) = field match {
  case "name" =>
    // name ordering...

  case "age" =>
    // age ordering...
}
```

Example: Orderings

```
def by(field: String) = field match {
  case "name" =>
    Ordering.by[Person, String](_.name)

  case "age" =>
    Ordering.by[Person, Int](_.age)
}
```

Example: Orderings

```
def by(field: String) = field match {
  case "name" =>
    Ordering.by[Person, String](_.name)

  case "age" =>
    Ordering.by[Person, Int](_.age)
}
```

Example: Orderings

```
def by(field: String) = field match {
  case "name" =>
    Ordering.by[Person, String](_.name)

  case "age" =>
    Ordering.by[Person, Int](_.age)
}
```

Example: Orderings

```
def orderings[A]: String => Ordering[A] =
    macro orderingsMacro[A]

def orderingsMacro[A: c.WeakTypeTag](c: Context) = {
  import c.universe._

  val tpe   = c.weakTypeOf[A]
  val cases = tpe.decls collect {
    case method: MethodSymbol if method.isCaseAccessor =>
      val ret = method.returnType
      cq"""
      ${method.name.toString} =>
        Ordering.by[$tpe, $ret](_.${method.name})
      """
  }

  q"(name: String) => name match { case ..$cases }"
}
```

Example: Orderings

```
def orderings[A]: String => Ordering[A] =
    macro orderingsMacro[A]

def orderingsMacro[A: c.WeakTypeTag](c: Context) = {
  import c.universe._

  val tpe   = c.weakTypeOf[A]
  val cases = tpe.decls collect {
    case method: MethodSymbol if method.isCaseAccessor =>
      val ret = method.returnType
      cq"""
      ${method.name.toString} =>
        Ordering.by[$tpe, $ret](_.${method.name})
      """
  }

  q"(name: String) => name match { case ..$cases }"
}
```

Example: Orderings

```
def orderings[A]: String => Ordering[A] =
  macro orderingsMacro[A]

def orderingsMacro[A: c.WeakTypeTag](c: Context) = {
 import c.universe._

 val tpe   = c.weakTypeOf[A]
 val cases = tpe.decls collect {
  case method: MethodSymbol if method.isCaseAccessor =>
   val ret = method.returnType
   cq"""
   ${method.name.toString} =>
    Ordering.by[$tpe, $ret](_.${method.name})
   """
 }

 q"(name: String) => name match { case ..$cases }"
}
```

Example: Orderings

```
def orderings[A]: String => Ordering[A] =
    macro orderingsMacro[A]

def orderingsMacro[A: c.WeakTypeTag](c: Context) = {
  import c.universe._

  val tpe   = c.weakTypeOf[A]
  val cases = tpe.decls collect {
    case method: MethodSymbol if method.isCaseAccessor =>
      val ret = method.returnType
      cq"""
      ${method.name.toString} =>
        Ordering.by[$tpe, $ret](_.${method.name})
      """
  }

  q"(name: String) => name match { case ..$cases }"
}
```

Example: Orderings

```
def orderings[A]: String => Ordering[A] =
    macro orderingsMacro[A]

def orderingsMacro[A: c.WeakTypeTag](c: Context) = {
  import c.universe._

  val tpe   = c.weakTypeOf[A]
  val cases = tpe.decls collect {
    case method: MethodSymbol if method.isCaseAccessor =>
      cq"""
      ${method.name.toString} =>
        implicitly[Ordering[${method.returnType}]].
          on[$tpe](_.${method.name})
      """
  }

  q"(name: String) => name match { case ..$cases }"
}
```

Example: Orderings

```
case class Person(name: String, age: Int)

val people = List(
  Person("Anne",    35),
  Person("Bob",     45),
  Person("Charlie", 20))

val by = orderings[Person]

people sorted by("name")
```

Example: Orderings

Take home points

Generic macros — take type parameters and are useful for code generation

Types — can be inspected and traversed (provided they are sufficiently grounded)

Error handling is important — we may not be able to de-alias or inspect the type!

Implicits

Example: CSV

project csv in the code

Example: CSV

```
case class Person(name: String, age: Int,
  address: Address)

case class Address(house: Int, street: String)

val people = List(
  Person("Anne",    35, Address(1, "High Street")),
  Person("Bob",     45, Address(2, "Bristol Road")),
  Person("Charlie", 20, Address(3, "Acacia Avenue”)))
```

Example: CSV

```scala
case class Person(name: String, age: Int,
  address: Address)

case class Address(house: Int, street: String)

val people = List(
  Person("Anne",    35, Address(1, "High Street")),
  Person("Bob",     45, Address(2, "Bristol Road")),
  Person("Charlie", 20, Address(3, "Acacia Avenue")))

println(writeCsv(people))
```

Example: CSV

```
trait CsvFormat[A] extends (A => Seq[String])

def writeCsv[A: CsvFormat](values: Seq[A]): String =
  // ...
```

Example: CSV

```
trait CsvFormat[A] extends (A => Seq[String])

def writeCsv[A: CsvFormat](values: Seq[A]): String =
  // ...




implicit val addressFormat = new CsvFormat[Address] // ...

implicit val personFormat = new CsvFormat[Person] // ...

println(writeCsv(people))
```

Example: CSV

```
trait CsvFormat[A] extends (A => Seq[String])

def writeCsv[A: CsvFormat](values: Seq[A]): String =
  // ...

def csvFormat[A]: CsvFormat[A] =
  macro csvFormatMacro[A]


implicit val addressFormat = new CsvFormat[Address] // ...

implicit val personFormat = new CsvFormat[Person] // ...

println(writeCsv(people))
```

Example: CSV

```
trait CsvFormat[A] extends (A => Seq[String])

def writeCsv[A: CsvFormat](values: Seq[A]): String =
  // ...

def csvFormat[A]: CsvFormat[A] =
  macro csvFormatMacro[A]


implicit val addressFormat = csvFormat[Address]

implicit val personFormat = csvFormat[Person]

println(writeCsv(people))
```

Example: CSV

```
trait CsvFormat[A] extends (A => Seq[String])

def writeCsv[A: CsvFormat](values: Seq[A]): String =
  // ...

implicit def csvFormat[A]: CsvFormat[A] =
  macro csvFormatMacro[A]


implicit val addressFormat = csvFormat[Address]

implicit val personFormat = csvFormat[Person]

println(writeCsv(people))
```

Example: CSV

```
trait CsvFormat[A] extends (A => Seq[String])

def writeCsv[A: CsvFormat](values: Seq[A]): String =
  // ...

implicit def csvFormat[A]: CsvFormat[A] =
  macro csvFormatMacro[A]




println(writeCsv(people))
```

Example: CSV

```
case class Person(name: String, age: Int,
  address: Address)

case class Address(house: Int, street: String)

val people = List(
  Person("Anne",    35, Address(1, "High Street")),
  Person("Bob",     45, Address(2, "Bristol Road")),
  Person("Charlie", 20, Address(3, "Acacia Avenue”)))

println(writeCsv(people))
```

Example: CSV

```
trait LowPriorityCsvImplicits {
  implicit def csvFormat[A]: CsvFormat[A] =
    macro csvFormatMacro[A]
}

trait CsvImplicits extends LowPriorityCsvImplicits {
  implicit val stringFormat : CsvFormat[String] = // ...
  implicit val intFormat    : CsvFormat[Int]    = // ...
  // ...
}
```

Example: CSV

Take home points

Implicit macros — can be used
to generate implicit values

Low/high priority traits — let us choose predefined
or macro-generated values based on type

Honorary Mention

Example: Validation

project validation in the code

~~Example Validation~~

~~***project validation*** is the code~~

Example: Validation

```
implicit val personValidator =
  validate[Person].
  field(_.name)(nonBlank and maxLength(40)).
  field(_.age)(nonNegative).
  field(_.address)
```

Example: Validation

```
implicit val personValidator =
  validate[Person].
  field(_.name)(nonBlank and maxLength(40)).
  field(_.age)(nonNegative).
  field(_.address)
```

Summary

Summary

Def macros with quasiquotes —in Scala 2.11

Setup — project / code structure (***maximum***)

Trees — creating / inspecting (***printtree***, ***assert***)

Types — inspecting / code generation (***ordering***)

Implicits — type class generation (***csv***)

Philosophy

Your library must be usable without macros

(over-reliance leads to brittle code)

Scala has types and implicits — use them!

(most problems are solvable without macros)

Use macros to tidy up or provide defaults

(make things easier or prettier in 80% of cases)

Be wary of the code you generate

(don’t subvert developers’ expectations)

References

Version 2.0
(new slide)

Macros Guide in the Scala Documentation
http://docs.scala-lang.org/overviews/macros/usecases.html

Eugene Burmako, flatMap(Oslo) — Macrology 201 Workshop
https://github.com/scalamacros/macrology201

Scala Reflect API Documentation
http://www.scala-lang.org/api/2.11.1/scala-reflect/#scala.reflect.package

Eugene Burmako, Scala eXchange — What are macros good for?
http://www.parleys.com/play/520a25c7e4b06de8a0ad962d/chapter0/about

Any questions?

Grab the code from:

_.io

underscore.io

Dave Gurnell, @davegurnell