Coding in Style

J. Suereth

Agenda

- Simple Scala Style guidelines
- Highlight Scala 2.10.x new features
- Discussion

Embrace Operator Notation

```scala
def privs(user: Option[User]): Privileges =
  user.map(findUserPrivileges).getOrElse(
    NoPrivs)
```

```
def privs(user: Option[User]): Privileges =
  (user map findUserPrivileges
   getOrElse NoPrivs)
```

Favor Expressions

```
def isAwesome(data: Data): Boolean = {
  if(data.awesome) return true
  if(data.lame) return false
  return data.contains("Tasty Sandwich")
```

```
def isAwesome(data: Data): Boolean =
  if(data.awesome) true
  else if(data.lame) false
  else (data contains "Tasty Sandwich")
```

```
def isAwesome(data: Data): Boolean =
    (data.awesome ||
    (!data.lame)  ||
    (data contains "Tasty Sandwich"))
```

Let the language do the work

SUDO Make me a variable

```
def slurp(file: File): String = {
  var input: InputStream = null
  var output: OutputStream = null
  var read = 0
  try {
    input = new FileInputStream(file)
    output = new StringOutputStream()
    val buf = new Array[Byte](BUF_SIZE)
    read = input.read(buf)
    while(read > 0) {
       output.write(buf, 0, read)
       read = input.read(buf)
    }
  } finally {
    if(input != null) input.close()
    if(output != null) output.close()
 }
 if(output != null) return output.toString
 return null
}
```

```scala
def slurp(in: File): String = {
  val in = new FileInputStream(in)
  val out = new StringWriter()
  val buf = new Array[Byte](BUF_SIZE)
  def read(): Unit = (in read buf) match {
    case 0 => ()
    case n =>
      out.write(buf, 0, n)
      read()
  }
  try read()
  finally in.close()
  out.toString
}
```

Tail-recursioN!

```
@tailrec
def read(): Unit = (in read buf) match {
  case 0 => ()
  case n =>
    out.write(buf, 0, n)
    read()
}
```

If it seems useful, look for it

```
def countEm(counts: Seq[Int]): Int = {
  var total = 0
  for(count <- counts) total += count
  total
}
```

```
def countEm(counts: Seq[Int]): Int =
  counts.sum
```

Aim Higher

higher-order-functions, that is

```
def sort[T](
  data: Seq[T]
)(
  lessThan: (T,T) => Boolean
): Seq[T] = ...
```

Build strings the nice way

interpolators

```
val HAI = "HI"
val name = "SQUARE"
val height = 100
val width = 20

scala> val x = s"${HAI}"
x: String = HI

scala> f"$name is $height%04d meters by
$width%04d meters"
res0: String = SQUARE is 0100 meters by
0020 meters
```

Keep the abstract….

…. well abstract.

```scala
trait Node[N] {
  def value: N
}
trait Edge[N,E] {
  def value: E
  def from: Node[N]
  def to: Node[N]
}
```

```scala
trait Graph[N,E] {
  type Nd = Node[N]
  type Ed = Edge[N,E]

  def nodes: Set[Nd]
  def edges(n: Nd): Seq[Ed]
}
```

Add behavior

with shiny new value classes and implicit classes

```scala
object Graph {
  @inline implicit class Ops[N,E](val g: Graph[N,E])
    extends AnyVal {

    def isCyclic: Boolean = !isAcyclic

    def isAcyclic: Boolean =
      cycles forall (c => (c drop 1).isEmpty)

    def cycles: Set[Set[Node[N]]] =
      ... compute using tarjan or other algorithm ...

    def topological: Set[Node[N]] =
      ... topo-sort ...
  }
```

~~Cheat~~ ***Import*** to win

```
import language.{ _, reflectiveCalls => _ }
import language.experimental.macros
```

Hidden language features

- dynamics
- postfixOps
- reflectiveCalls
- implicitConversions
- higherKinds
- existentials
- experimental.macros

Cheat in an Emergency

With Dynamic Types

```
class MapHelper(json: Map[String,AnyRef])
    extends Dynamic {

  def selectDynamic(name: String) =
    (json get name
     getOrElse sys.error(s"Attribute $name
not found in json: $json"))

}
```

```
scala> val m = new
MapHelper(Map[String,AnyRef]("name" ->
"Jimmy", "age" -> "10"))
m: MapHelper = MapHelper@683f67e0

scala> m.name
res1: AnyRef = Jimmy

scala> m.sex
java.lang.RuntimeException: Attribute sex
not found in json: Map(name -> Jimmy, age
-> 10)
```

Promise the Future!

Make sure to deliver

```
import scala.concurrent._
import scala.concurrent.ExecutionContext.Implicits.global

scala> val x = promise[Int]
scala> val y = x.future
scala> y.isCompleted
res5: Boolean = false

scala> x success 5
scala> y.isCompleted
res7: Boolean = true
scala> y.value
res9: Option[Either[Throwable,Int]] = Some(Right(5))
```

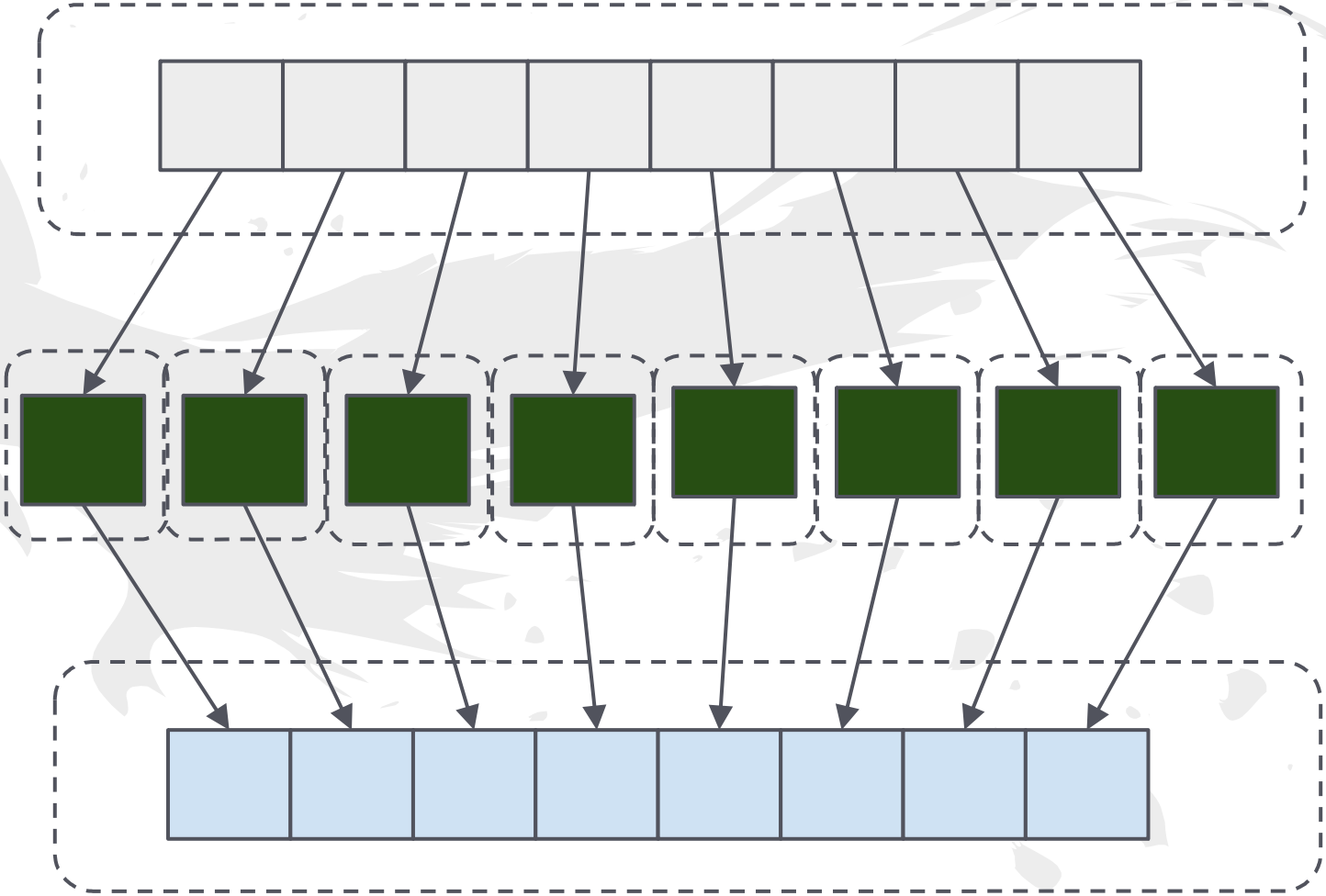
Staying Async!

See http://jsuereth.com/intro-to-fp/

```scala
trait AsynchService {
  def users: Future[Seq[User]]
  def projects(user: User):Future[Seq[Project]]
}
```

```
def allprojects(api: AsynchService):
Future[Set[Project]] =
  (Future.traverse(api.users)(api.projects)
   map (_.flatten.toSet))
```

```scala
def traverse[A](
  seq: Future[Seq[A]]
)(
  op: A => Future[B]
): Future[Seq[B]] = ....
```



Feeling Crazy?

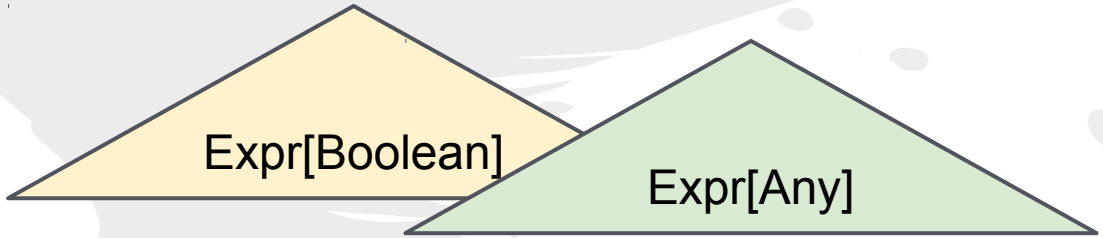
Manipulate some ASTs

```scala
object MacroAsserts {
  def assert(cond: Boolean, expr: String) =
    macro assertImpl
  def assertImpl(c: Context)(
    cond: c.Expr[Boolean],
    msg: c.Expr[String]
  ): c.Expr[Unit] =
    if(sys.env("SCALA_ASSERT") == "true")
      c.reify(if(cond.splice)
                sys.error(msg.splice))
    else c.reify(())
}
```

```
object Bar {
    assert(false,
      "THIS IS A BAD OBJECT")
}
```

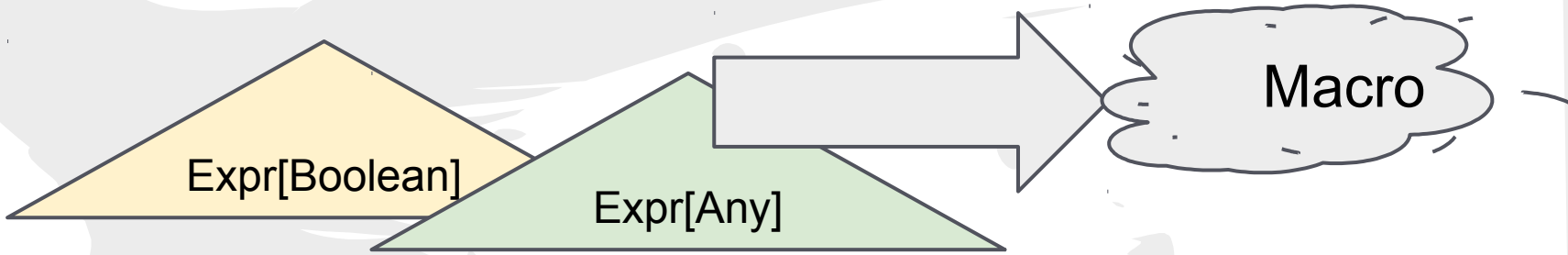
```
object Bar {

}
```



```
object Bar {

}
```



```
object Bar {
```

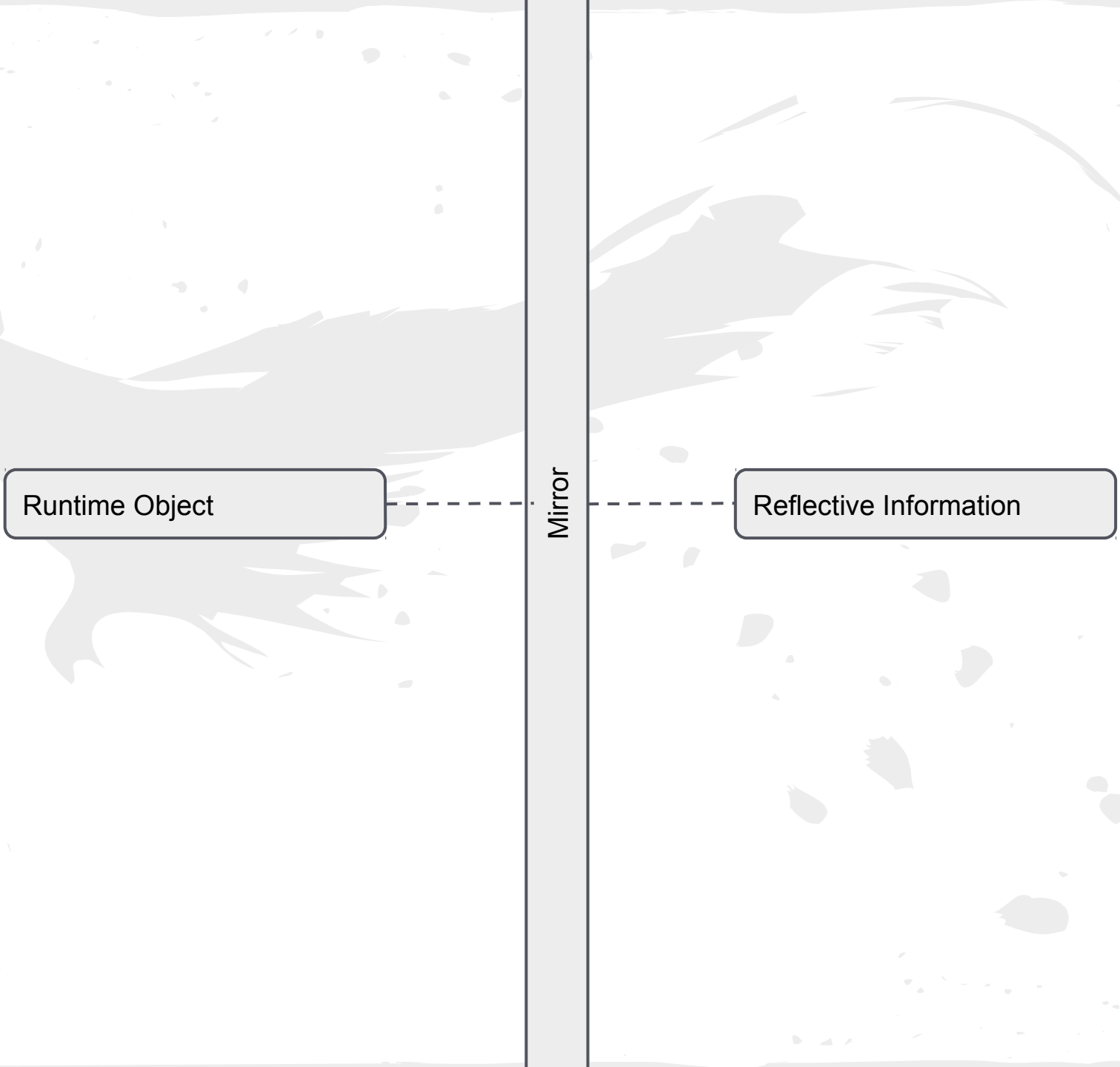

```
}
```

```
object Bar {
    ()

}
```

Feeling Crazier?

Take time to reflect


Runtime Object
Mirror
Reflective Information

```
scala> val m =  reflect.runtime.currentMirror
m: reflect.runtime.universe.Mirror = JavaMirror with
scala.tools.nsc.interpreter.IMain$TranslatingClassLoader@4
3540a77 of type class
scala.tools.nsc.interpreter.IMain$TranslatingClassLoader
with classpath [(memory)] and parent being
scala.tools.nsc.util.ScalaClassLoader$URLClassLoader@427b2
d29 of type class
scala.tools.nsc.util.ScalaClassLoader$URLClassLoader with
classpath [file:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/resources.jar,file:/usr/lib/jvm/java-6-
openjdk-amd64/jre/lib/rt.jar,file:/usr/lib/jvm/java-6-
openjdk-amd64/jre/lib/jsse.jar,file:/usr/lib/jvm/java-6-
openjdk-amd64/jre/lib/jce.jar,file:/usr/lib/jvm/java-6-
openjdk-amd64/jre/lib/charsets.jar,file:/usr/lib/jvm/java-
6-openjdk-
amd64/jre/lib/rhino.jar,file:/usr/share/java/scala-
compiler.jar,file:/usr/share/java/scala-reflect.jar,...
```

```
scala> import mirror.universe._
import m.universe._

scala> typeOf[String]
res23: m.universe.Type = String
```

A rich API

```
scala> val stringTpe = typeOf[String]
stringTpe: m.universe.Type = String

scala> stringTpe.
=:=                asInstanceOf       asSeenFrom         baseClasses
baseType           contains           declaration        declarations
erasure            exists             find               foreach
isConcrete         isHigherKinded     isInstanceOf       isSpliceable
kind               map                member             members
narrow             nonPrivateMember   nonPrivateMembers  normalize
parents            resultType         substituteSymbols  substituteTypes
```

How bout something useful?

```
scala> val mirror =
reflect.runtime.universe.runtimeMirror(
  classOf[Int].getClass.getClassLoader)
mirror: reflect.runtime.universe.Mirror = JavaMirror with primordial
classloader with boot classpath [/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/resources.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/rt.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/sunrsasign.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/jsse.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/jce.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/charsets.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/netx.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/plugin.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/rhino.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/lib/modules/jdk.boot.jar:/usr/lib/jvm/java-6-openjdk-
amd64/jre/classes:/usr/share/java/scala-compiler.jar:/usr/share/java/scala-
reflect.jar:/usr/share/java/scala-library.jar:/usr/shar.
```

How bout something useful?

```
val classSymbol: ClassSymbol =
  mirror classSymbol classOf[Int].getClass

val rc: ClassMirror = mirror reflectClass classSymbol

scala> rc.companion
res1: Option[reflect.runtime.universe.ModuleMirror] =
Some(scala.reflect.runtime.JavaMirrors$JavaMirror$JavaModu
leMirror@7efc0795)

scala> res1.get.runtimeClass
res2: Class[_] = class scala.Int$
```

What are trees/symbols/types?

```
class Foo(arg: String) {
  def bar = "String"
}
```

Trees

- Short lived, Mostly immutable, Mostly plain case-classes

Symbols

- Linking references and definitions

Long lived, mostly mutable

Types

- Immutable, long-lived
- Stores full information

Building stuff

In the alternative universe

```
val tb =
tools.reflect.ToolBox(reflect.runti
me.currentMirror).mkToolBox()

scala>
tb2.parseExpr("println(\"HI\")")
res0: tb2.u.Tree = println("HI")

scala> tb2.runExpr(res0)
HI
res1: Any = ()
```

What can a toolbox do?

- expression -> tree
- type checking
- tree evaluation
- implicits

Contexting your Macros

It's all about reflection

Macro Contexts

A macro context is an amalgamation of:

- Reflection Mirror
- Reflection Toolbox
- Unicorns
- Rainbows

You know what?

JUST READ THIS:

http://scalamacros.org/talks/201