

Scala: why do I care?

Asheville Coders League, August 28, 2013

Havoc Pennington

typesafe.com

twitter.com/havocp github.com/havocp

This Talk

A sample of Scala...

- May be easier to understand if you know Java
- Why is Scala interesting? Is this worth your time?
- My personal take. YMMV

Some quick history and background, then a bunch of sample code.

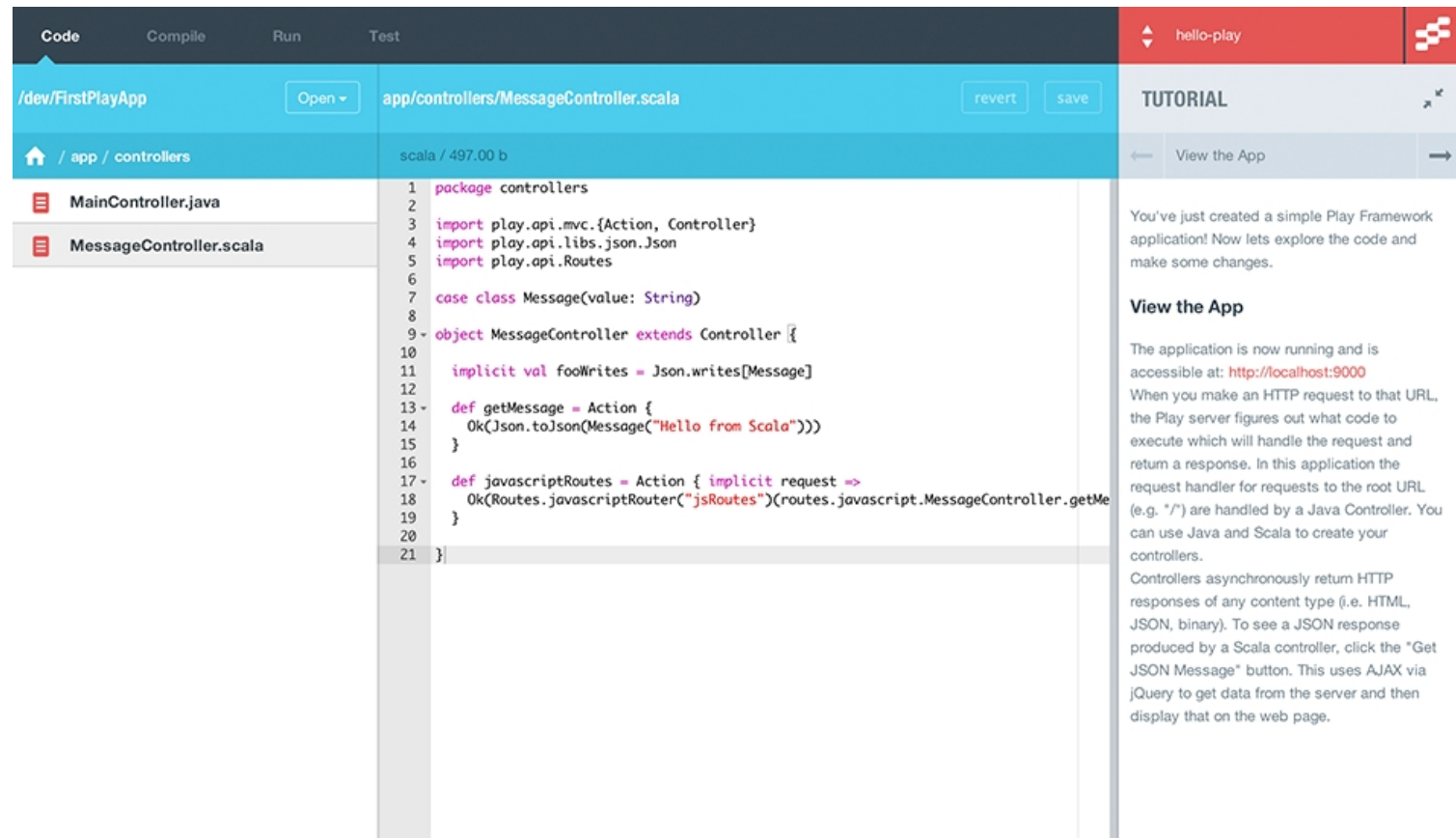
A Sampler

No way to teach you Scala in 45 minutes.

But you might be inspired to learn more based on what you see.

Typesafe Activator

My current work project



The screenshot displays the Typesafe Activator IDE interface. The top bar includes tabs for 'Code', 'Compile', 'Run', and 'Test'. Below this, the file explorer on the left shows the project structure with files like 'MainController.java' and 'MessageController.scala'. The main editor area displays the code for 'app/controllers/MessageController.scala', which includes package declarations, imports, a case class 'Message', and an object 'MessageController' extending 'Controller'. The code defines a 'getMessage' action and 'javascriptRoutes'. The right sidebar contains a 'TUTORIAL' section with instructions on how to view the application and understand the request handling process.

```
1 package controllers
2
3 import play.api.mvc.{Action, Controller}
4 import play.api.libs.json.Json
5 import play.api.Routes
6
7 case class Message(value: String)
8
9 object MessageController extends Controller {
10
11   implicit val fooWrites = Json.writes[Message]
12
13   def getMessage = Action {
14     Ok(Json.toJson(Message("Hello from Scala")))
15   }
16
17   def javascriptRoutes = Action { implicit request =>
18     Ok(Routes.javascriptRouter("jsRoutes")(routes.javascript.MessageController.getMe
19   )
20
21 }
```

TUTORIAL

You've just created a simple Play Framework application! Now let's explore the code and make some changes.

View the App

The application is now running and is accessible at: <http://localhost:9000>. When you make an HTTP request to that URL, the Play server figures out what code to execute which will handle the request and return a response. In this application the request handler for requests to the root URL (e.g. "/") are handled by a Java Controller. You can use Java and Scala to create your controllers.

Controllers asynchronously return HTTP responses of any content type (i.e. HTML, JSON, binary). To see a JSON response produced by a Scala controller, click the "Get JSON Message" button. This uses AJAX via jQuery to get data from the server and then display that on the web page.

What does Typesafe do?

Reactive apps on the Java Virtual Machine

- Scala: practical, superior alternative to Java with incremental migration path
- Akka: proven Actor model gives horizontal scale for both Java and Scala, without the pain points of explicit threads
- Play: popular Rails-style convention-over-configuration lightweight web framework for both Java and Scala
- We support BOTH Java and Scala, can use Akka/Play a la carte
- See <http://typesafe.com> and <http://reactivemanifesto.org/> for more

Scala History and Community

Where it comes from

Scala has established itself as one of the main alternative languages on the JVM.

Created by Martin Odersky, who also designed generics in Java 5 and implemented the javac compiler. He works with a team at EPFL (École polytechnique fédérale de Lausanne) in addition to the team at Typesafe.

Prehistory:

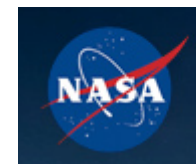
1996 – 1997: Pizza

1998 – 2000: GJ, Java generics, javac (*“make Java better”*)

Timeline:

2003 – 2006: The Scala “Experiment”

2006 – 2009: An industrial strength programming language
(*“make a better Java”*)



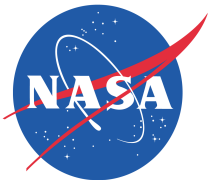
Select Commercial Users



Novell.

Migrated core messaging service from Ruby to sustain 3 orders of magnitude growth

Scala drives its social graph service: 380-400 M transactions/day



UBS

Entire web site and all services written in Scala

EU's largest energy firm migrated a 300K lines contract modeling platform from Java to Scala

Approved for general production use



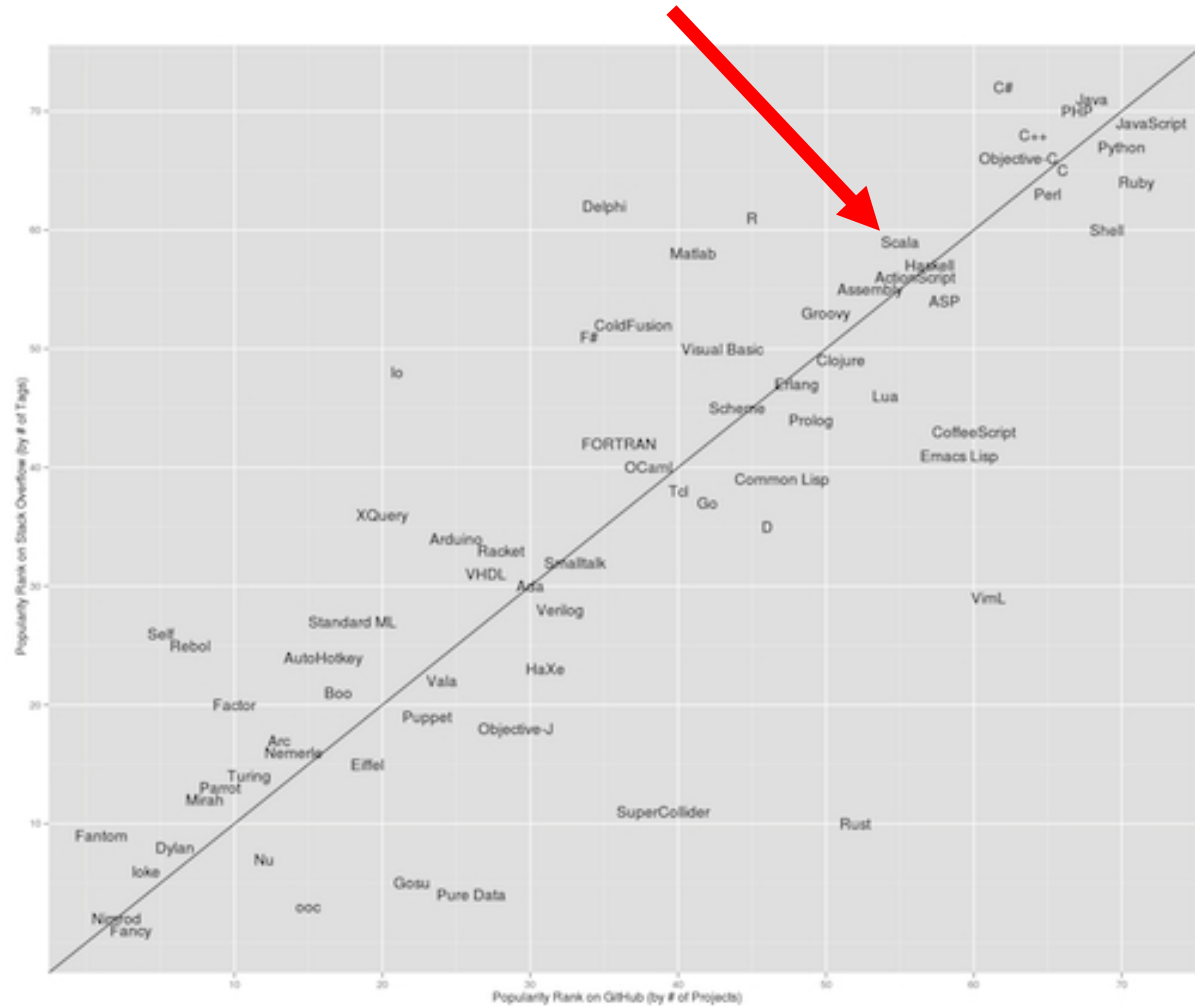
Community Traction

Open source ecosystem with

- Hundreds of contributors
- 20+ books
- 40+ active user groups
- Plenty of github/stackoverflow activity



RedMonk GitHub / StackOverflow Analysis



StackOverflow Popularity

GitHub Popularity



The Case for Scala

Why Scala?

Here's what I hope you'll see as we look at some Scala code:

- **Beautiful interoperability** with Java means this is a *practical* way to move to a more modern language
- **Conciseness** means dramatic improvement in maintainability and productivity
- Support for **immutable data structures** (functional programming style) *eliminates* whole classes of concurrency bugs
- **Makes previously-too-hard patterns feasible**, in particular async (nonblocking) code
- **Expressive** (hard to define, but we all like it)
- **Fun** to do your work in a less tedious and more elegant way

Less To Type, So What?

Reading and refactoring matter more than writing the first draft.

Less Code

When going from Java to Scala, expect at least a factor of 2 reduction in LOC.

Guy Steele: “Switching from Java to Scala reduced size of Fortress typechecker by a factor of 4”.

But does it matter?

Doesn't Eclipse write these extra lines for me?

This does matter. Eye-tracking experiments* show that for program comprehension, average time spent per word of source code is constant.

So, roughly, **half the code means half the time necessary to understand it.**

*G. Dubochet. Computer Code as a Medium for Human Communication: Are Programming Languages Improving? In 21st Annual Psychology of Programming Interest Group Conference, pages 174-187, Limerick, Ireland, 2009.

Let's See Some Scala

Finally!

A fluent, clean look

```
val people = getPeople()  
  
val adults = people filter (_.age >= 18)  
  
val agesOfAdultsStartingWithA =  
    adults filter(_.name.startsWith("A")) map (_.age)
```

OR, equivalently

```
getPeople() filter (_.age >= 18) filter (_.name startsWith "A") map (_.age)
```

For now just note the lack of type-annotation boilerplate... it doesn't look like the typesafe languages you may be used to

Type Inference

- Omit explicit types when it's obvious

```
// type String is inferred  
val s = "hello"
```

```
// but spell it out if you want to  
val s2: String = "hello"
```

Scala puts the type *after* the name, to allow omitting it

Hello.scala

```
object Hello extends App {  
  println("Hello, World")  
}
```

Hello.java

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World");  
    }  
}
```

Red = noise that adds no value

Imagine reading a book with 50% irrelevant words!

Noise Reduction

```
object Hello extends App {  
  println("Hello, World")  
}
```

Notice anything missing?

- println is a standalone function so no object to invoke it on
- “object” keyword creates a singleton, no need for singleton machinery or “static”
- “App” provides main() boilerplate
- Class itself can have a body, in this case run from main()
- “public” is the default
- No semicolons

Something More Involved

A simple “data” class (bean?) to store a Person with “name” and “age” fields.

A class ...

... in Java:

```
public class Person {  
    public final String name;  
    public final int age;  
    Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

... in Scala:

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

Don't Repeat Yourself?

```
public class Person {  
    public final String name;  
    public final int age;  
    Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

- We had to say “name” and “age” four times each.
- We had to say “Person” two times.
- We had to say “public” three times and “final” twice.
- We had to say “String” and “int” twice each.

Scala says it once

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

- A case class is an immutable (final) set of fields
- It automatically defines equals, hashCode, toString sensibly, the Java version does not (without even more code...)
- Automatically supports pattern matching (more on next slide)

Remember: Scala puts the type *after* the name

Pattern matching

Why is it called a case class?

```
def selectCategory(person: Person) = person match {  
  case Person("Havoc", _) => "Slow Race"  
  case Person(_, age) if age > 60 => "Masters"  
  case _ => "Regular"  
}
```

(Stuff to notice: “_” as wildcard, no “return” keyword, implicit return type)

Tuples

Case classes are a “named tuple” but the name is optional (Scala is not too opinionated):

```
def selectCategory(person: (String, Int)) = person match {  
  case ("Havoc", _) => "Slow Race"  
  case (_, age) if age > 60 => "Masters"  
  case _ => "Regular"  
}
```

(Stuff to notice: “_” as wildcard, no “return” keyword, implicit return type)

Working with collections

... in Java:

```
import java.util.ArrayList;

...
Person[] people;
Person[] minors;
Person[] adults;
{ ArrayList<Person> minorsList = new ArrayList<Person>();
  ArrayList<Person> adultsList = new ArrayList<Person>();
  for (int i = 0; i < people.length; i++)
    (people[i].age < 18 ? minorsList : adultsList)
      .add(people[i]);
  minors = minorsList.toArray(people);
  adults = adultsList.toArray(people);
}
```

... in Scala:

```
val people: Array[Person]
val (minors, adults) = people.partition (_.age < 18)
```

Anonymous functions

```
val people: Array[Person]  
val (minors, adults) = people partition (_.age < 18)
```

```
// the short form
```

```
_.age < 18
```

```
// with named parameter
```

```
person => person.age < 18
```

```
// non-anonymous version
```

```
def isMinor(person: Person): Boolean =  
    person.age < 18
```

From “Too Hard” to “Actually Happens”

A great point from Play developer James Roper (see his blog post at http://jazzy.id.au/default/2012/11/02/scaling_scala_vs_java.html)

Writing asynchronous, nonblocking code is a known way to greatly improve scalability. But people *typically do not* write it in Java, and they *typically do* write it in Scala.

It's not about whether you *can* - in any Turing-complete language you can - but whether you *will*, and will it be maintainable.

Async is one example.

Async code, Java vs. Scala

Java

```
1 Promise<User> user = getUserById(id);
2 Promise<List<Order>> orders = user.flatMap(new Function<User, List<Order>>>() {
3     public Promise<List<Order>> apply(User user) {
4         return getOrdersForUser(user.email);
5     }
6 }
7 Promise<List<Product>> products = orders.flatMap(new Function<List<Order>, List<Product>>>() {
8     public Promise<List<Product>> apply(List<Order> orders) {
9         return getProductsForOrders(orders);
10    }
11 }
12 Promise<List<Stock>> stock = products.flatMap(new Function<List<Product>, List<Stock>>>() {
13     public Promise<List<Stock>> apply(List<Product> products) {
14         return getStockForProducts(products);
15    }
16 }
```

Scala

```
1 for {
2   user <- getUserById(id)
3   orders <- getOrdersForUser(user.email)
4   products <- getProductsForOrders(orders)
5   stock <- getStockForProducts(products)
6 } yield stock
```

OR ...

(Java 8 lambda is helpful, of course, but not as helpful as Scala)

New async/await syntax

Scala with async/await macros (like C#):

```
async {  
  val user = await getUserById(id)  
  val orders = await getOrdersForUser(user.email)  
  val products = await getProductsForOrders(orders)  
  await getStockForProducts(products)  
}
```

Traditional Scala way for comparison:

```
1 | for {  
2 |   user <- getUserById(id)  
3 |   orders <- getOrdersForUser(user.email)  
4 |   products <- getProductsForOrders(orders)  
5 |   stock <- getStockForProducts(products)  
6 | } yield stock
```


Less Is More

Greatly Reducing Useless Noise =
Much More Maintainable Code

Don't Repeat Yourself

Touring a Few More Tricks

- Code Blocks
- Traits
- Flexible Method Call Syntax
- Implicits
- Java interoperability

(this is a somewhat arbitrary sample of cool stuff)

Code Blocks

- Code blocks let you elegantly factor out, for example, exception handling

```
def ignoringIOException[T](block: => T): Unit = {  
  try {  
    block  
  } catch {  
    case e: IOException => ()  
  }  
}  
  
ignoringIOException {  
  in.close()  
}
```

Traits: Interface + Code

- Scala has “traits” rather than interfaces; traits optionally contain some implementation
- With no implementation, they compile to interfaces
- Multiple inheritance Just Works (Google “Scala class linearization” for gory details that don't matter)

```
trait Quacker {  
  def quack = println("Quack")  
}
```

```
trait Barker {  
  def bark = println("Woof")  
}
```

```
class DuckDog extends Quacker with Barker
```

Flexible method call syntax

- To read Scala code you need to know: the dot and the parens can be left out when there's no ambiguity

```
val s = "Hello"
```

```
val len1 = s.length()
```

```
val len2 = s.length
```

```
val len3 = s length
```

```
val s1 = s.substring(1)
```

```
val s2 = s substring 1
```

Few restrictions on method names

- You can name a method with any Unicode characters
- Some people call this “operator overloading” but it is NOT like C++ where “operators” are special. In Scala, it's just that Scala doesn't restrict method names.
- Yes you can name a method “P†■” but please use common sense!

```
val a = 1  
val b = 2  
// “+” with method syntax  
println(a.+(b))  
// “+” with dot and parens omitted  
println(a + b)
```

Implicit conversion

For example, the standard library contains:

```
// Convert a Byte to a wider numeric type  
implicit def byte2short(x: Byte): Short = x.toShort  
implicit def byte2int(x: Byte): Int = x.toInt  
implicit def byte2long(x: Byte): Long = x.toLong  
implicit def byte2float(x: Byte): Float = x.toFloat  
implicit def byte2double(x: Byte): Double = x.toDouble
```

This mechanism is available for any user-defined type, while other languages have special-case built-in rules for numeric primitives.

Implicit conversion to find a method

If a method doesn't exist, maybe a conversion can make it exist. Safe alternative to “monkey patching” (as in Ruby, JavaScript).

// Silly example

```
class Quacker {  
  def quack = println("quack")  
}
```

```
implicit def stringToQuacker(s: String): Quacker = new Quacker
```

```
"hello".quack
```


Java Interoperability

Practical Migration and Interoperability

- Scala differs from Java *only on the source code level*
- Once you have a .jar or .war, it just looks like Java
- Scala code can seamlessly import and use any Java class
- Projects can have a mix of Java and Scala files
- Deploy Scala to any cloud or container that supports Java

Java:

```
import java.net.URL;
import java.io.InputStream;

URL u = new URL("http://foo.com");
InputStream s = u.openStream();
s.close();
```

Scala:

```
import java.net.URL

val u = new URL("http://foo.com")
val s = u.openStream()
s.close()
```

Scala Has a Java Core

While Scala improves on Java, it keeps Java as the core

- the syntax is in the same general C++-like family
- on JVM level, classes are classes, methods are methods, etc.
- support for object-oriented programming in exactly Java style
- objects have toString, equals
- even annotations, generics, and other “advanced” features correspond between Java and Scala
- collections are different but easy no-copy conversions are provided
- Compile-time type checks

You can implement the same API in Scala that you would in Java; all the same once it's compiled to class files.

Scala can be used as “concise Java” right away, short learning curve.

Immutable Data

Immutable Data

- Concise, readable code is one advantage of Scala
- Another huge one: support for immutable data

Immutability: Key to Functional Programming

- Emphasizes transformation (take a value, return a new value) over mutable state (take a value, change the value in-place)
- Think $f(x)$
- Advantages include:
 - Inherently parallelizable and thread-safe
 - Enables many optimizations, such as lazy evaluation
 - Can make code more flexible and generic, for example by supporting composition

Mutable Data, Imperative Style (Java)

```
public static void addOneToAll(ArrayList<Integer> items) {  
    for (int i = 0; i < items.size(); ++i) {  
        items.set(i, items.get(i) + 1);  
    }  
}
```

Mutates (modifies) the list in-place

Immutable Data, Functional Style (Java)

```
public static List<Integer> addOneToAll(List<Integer> items) {  
    ArrayList<Integer> result = new ArrayList<Integer>();  
    for (int i : items) {  
        result.add(i + 1);  
    }  
    return result;  
}
```

Returns a new list, leaving original untouched


Mutable Data, Imperative Style (Scala)

```
def addOneToAll(items : mutable.IndexedSeq[Int]) = {  
  var i = 0  
  while (i < items.length) {  
    items.update(i, items(i) + 1)  
    i += 1  
  }  
}
```

Mutates (modifies) the list in-place

Immutable Data, Functional Style (Scala)

```
def addOneToAll(items : Seq[Int]) = items map { _ + 1 }
```



Anonymous function applied to
each item in the list

Returns a new list, leaving original untouched

Language and library support

- Most importantly, library APIs all support use of immutable collections and other data
- But many small language features support immutability.

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

```
// case classes automatically have “copy”
```

```
def celebrateBirthday(person: Person) =  
  person.copy(age=person.age+1)
```

Enabling Composition

```
public static List<Integer> addTwoToAll(List<Integer> items) {  
    return addOneToAll(addOneToAll(items));  
}
```

(Composition is great for HTTP request handlers, by the way.)

Automatic Thread Correctness

The official Oracle tutorial on concurrency recommends immutable data, but Java APIs don't support it, and doing it properly in Java means lots of ceremony.

In Scala it's the default, as it should be.

Processors have more cores every day.

Switch to Immutable Style!

Even if you don't switch to Scala, this will improve your Java (or JavaScript, or C, or ...)

Reserve mutability for hotspots revealed by profiling.

My config lib is an example of a Java library with an immutable API:

- <https://github.com/typesafehub/config/>

Why not Scala?

Some Pitfalls

- Do the usual new-technology due diligence:
 - does it make sense for your team/situation?
 - did you leave time for training and learning?
 - walk before you run: start smaller, more Java-like
- Build times
- IDEs are usable but not as mature
- The complexity debate...

How I think about complexity...

- *Where* is the complexity (for example, library or application)
- *What are we getting* for the complexity
- *What happens* when we don't understand the complexity

Every library, language, or line of code adds complexity, but what do we get for it? Is it *worth it*?

Scala and Complexity

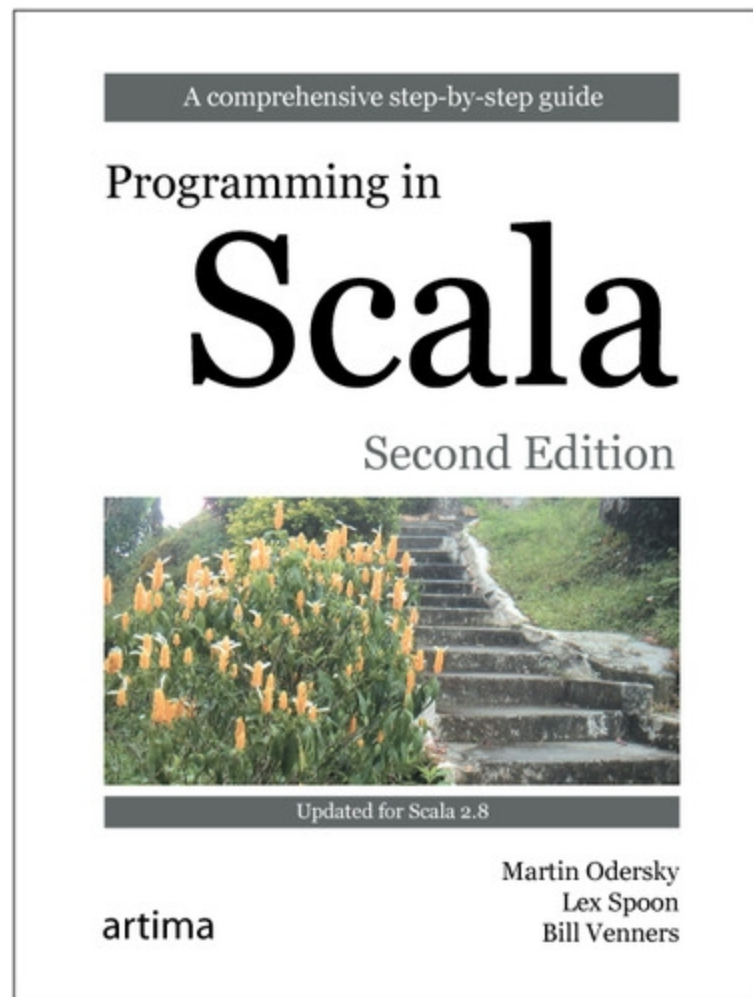
- Clearly Scala has more language features than Java (though it isn't out of line with say C#)
- But:
 - Scala is often **more consistent** (e.g. no static methods, primitives can be used like other objects, functions can be nested, every statement evaluates to a value, etc.)
 - Scala **doesn't require routine type casts**
 - Scala removes the need for **extralinguistic “fixes”** such as AspectJ, dependency injection, annotation/reflection-based tricks
 - Reading the Scala collections library may blow your mind, but **application code** written with it is **shorter and easier to read**
 - Getting Scala wrong usually means “won't compile” (compare to C++, where it means “explode”)
 - Power can be misused, but can also be used well. Expressiveness requires freedom.

Scala in Summary

- Beautiful interoperability with Java: mix Scala and Java as desired, no huge rewrites
- Conciseness means dramatic improvement in maintainability and productivity
- Formerly “too hard” patterns such as async IO become possible
- Functional style with immutable data seamlessly supports parallel computation and thread safety
- Vibrant community, hockey-stick growth curve, and available commercial support

Recommended: Read the Book

- Very clearly-written, by Scala's creator and designer
- Highly recommended to start here



Some Next Steps

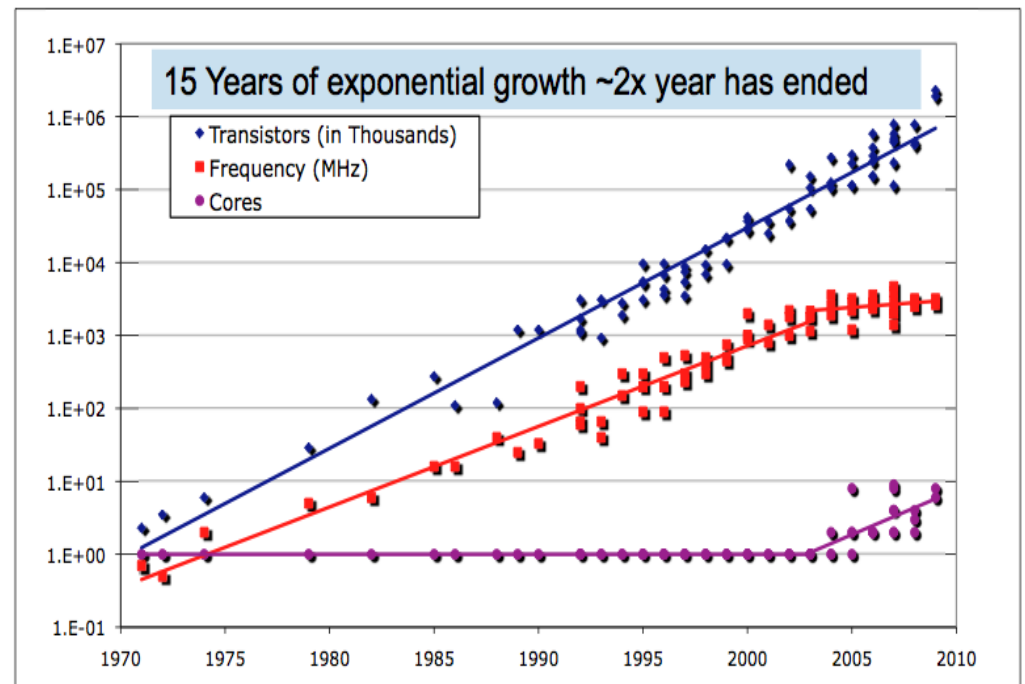
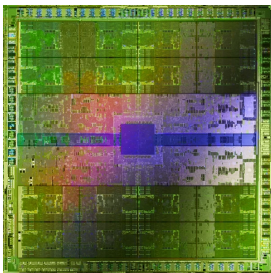
- Read *Programming in Scala* by Martin Odersky
- Try Typesafe Activator (my project!) for tutorials
- Get the Eclipse-based Scala IDE (or Scala support for your favorite IDE/editor)
- Get the Scala support for your favorite build tool
- Try writing a discrete module or two in Scala, unit tests are a safe place to start
- Consider a training course online or in-person, especially if you have a group to bring up to speed
- Join the community: Hendersonville Scala User Group, follow people on Twitter, mailing lists, StackOverflow
- Two Coursera courses: Functional Programming in Scala and **NEW** Principles of Reactive Programming

Scala backup slides

Horizontal Scale

The world of mainstream software is changing:

- Moore's law now achieved by increasing # of cores not clock cycles
- Huge volume workloads that require horizontal scaling



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanovic

Concurrency is too hard

Almost every program that uses
threads is full of bugs.

The Root of The Problem

- *Non-determinism* caused by *concurrent threads* accessing *shared mutable state*.
- It helps to encapsulate state in actors or transactions, but the fundamental problem stays the same.
- So,

```
var x = 0
async { x = x + 1 }
async { x = x * 2 }

// can give 0, 1, 2
```

non-determinism = *parallel processing* + *mutable state*

- To get deterministic processing, avoid the mutable state!
- Avoiding mutable state means programming *functionally*.

Remember this code from before...

... in Java:

```
import java.util.ArrayList;

...
Person[] people;
Person[] minors;
Person[] adults;
{ ArrayList<Person> minorsList = new ArrayList<Person>();
  ArrayList<Person> adultsList = new ArrayList<Person>();
  for (int i = 0; i < people.length; i++)
    (people[i].age < 18 ? minorsList : adultsList)
      .add(people[i]);
  minors = minorsList.toArray(people);
  adults = adultsList.toArray(people);
}
```

... in Scala:

```
val people: Array[Person]
val (minors, adults) = people.partition (_.age < 18)
```

Let's make it parallel...



... in Java:

... in Scala:

```
val people: Array[Person]
val (minors, adults) = people.par partition (_.age < 18)
```

Quick Overview of Scala

- Vibrant open source community
- Significant and growing commercial adoption
- Catching fire last couple years, but has been maturing many years
- Supports both object-oriented and functional styles; “not opinionated”
- Great interoperability with Java
- Compile-time type safety
- Type inference makes it concise and expressive, more like a dynamic language
- “Scala” implies a “scalable language”
 - Horizontal scale across cores
 - Developer scale: manage complex projects