面向对象分析与设计

齐琦
海南大学

课程简介

- 课程网页
  - http://qiqi789.github.io/teaching/OO/
- 我的电子邮箱
  - qqi@hainu.edu.cn

面向对象分析与设计（2014年秋季学期）

更新通知

September 22, 2014

> 欢迎来到这学期的课程!
>
> 请浏览以下网页以了解这门课程的信息。

课程信息

上课时间地点

第4至7周：周二，1-3节课，地点5-420；周五，1-3节课，地点5-220

第8至18周：周二，1-3节课，地点5-420。

课程形式：

以下安排可能会有变化。

- 讲课：计划18次（如下面的课程安排所列），每次3节课时
- 作业：可能有笔答和编程两种形式
- 考试：1次期末考试
- 编程设计：1个最终编程设计，以小组完成（1-3人一组），最后有一个答辩

评分标准：

以下安排可能会有变化。

课堂出勤和参与	10%
课后作业	20%
期末考试	30%
最终编程设计	40%
总共	100%

课本参考：

Odersky, M., L. Spoon and B. Venners (2011). Programming in Scala: [a comprehensive step-by-step guide; updated for scala 2.8]. Walnut Creek, Calif, Artima Press.

课程交流：

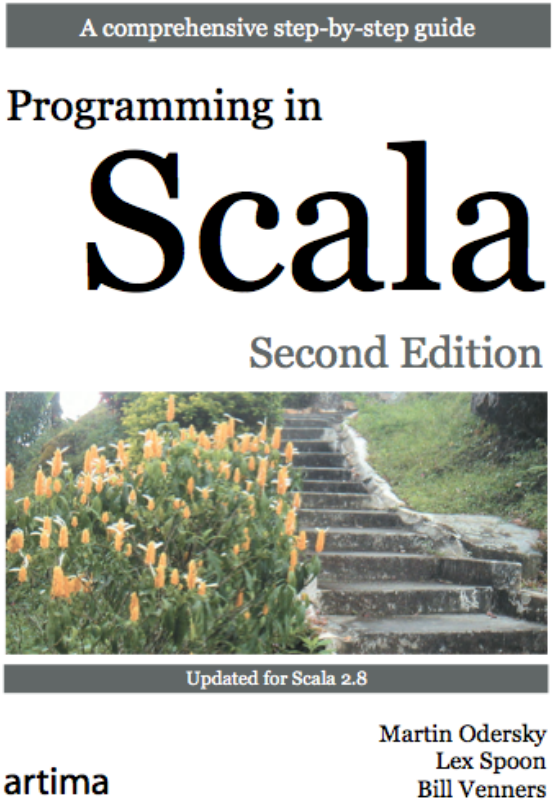
我的电子邮件地址是：qqi@hainu.edu.cn

课程安排

随课程进展，以下安排可能会有变化。

课程简介

- 课本参考
  - Odersky, M., L. Spoon and B. Venners (2011). Programming in Scala: [a comprehensive step-by-step guide; updated for scala 2.8]. Walnut Creek, Calif, Artima Press.
  - 不必须购买
- Scala 编程语言为载体



课程简介

- 课程形式
  - 讲课：计划18次，每次3节课时
  - 作业：可能有笔答和编程两种形式
  - 考试：1次期末考试
  - 编程设计：1个最终编程设计，以小组完成（1-3人一组），最后有一个答辩
- 评分标准
  - 课堂出勤和参与 10%
  - 课后作业 20%
  - 期末考试 30%
  - 最终编程设计 40%
  - 总共100%

今天的内容

- 编程范型
- Scala是什么？
- 为什么用Scala ？
- 如何开始Scala编程？

编程范型

- 命令式 (Imperative programing)
- 函数式 (Functional programming)
- 面向对象式 (Object-Oriented programming)
- 逻辑式 (Logic programming)
- 混合式 (Hybrid programming)

How OOP got started

First popular OO language:

Simula 67

Second popular OO language:

Smalltalk

来自 Martin Odersky: Scala with Style

Why Did OOP become popular?

Encapsulation?	No
Code Re-use?	Don't think so
Dynamic Binding	Not directly
Dependency inversion?	Came much later
Liskov substition principle?	
Open-closed principle?	You gotta be kidding!

It was because of the things you could **do** with OOP!

How OOP got started

New challenge: Simulation

Fixed number of operations:

```
nextStep
toString
aggregate
```

Unbounded number of implementations:

```
Car, Road, Molecule, Cell, Person, Building, City, ...
```

How OOP got started

New challenge: GUI widgets

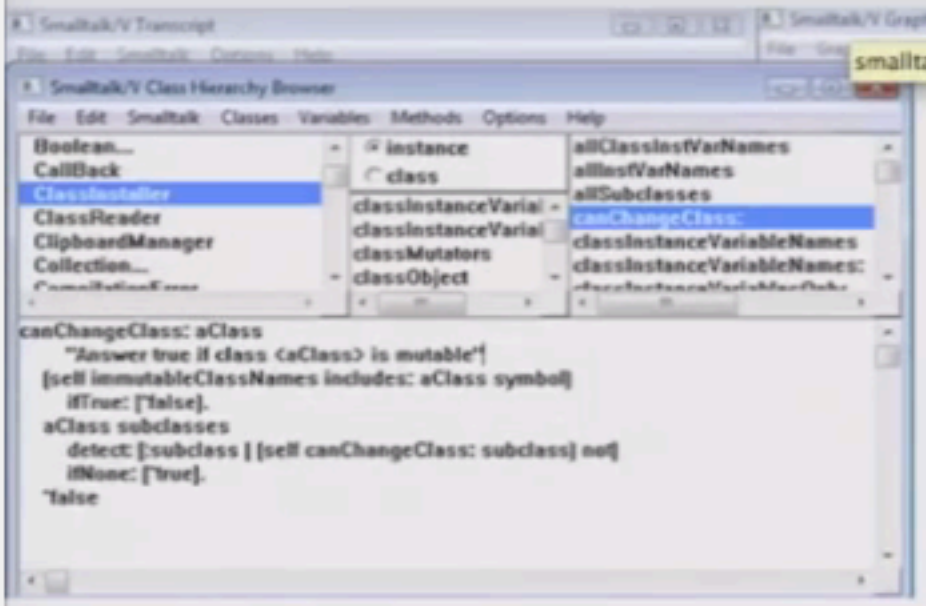
Fixed number of operations:

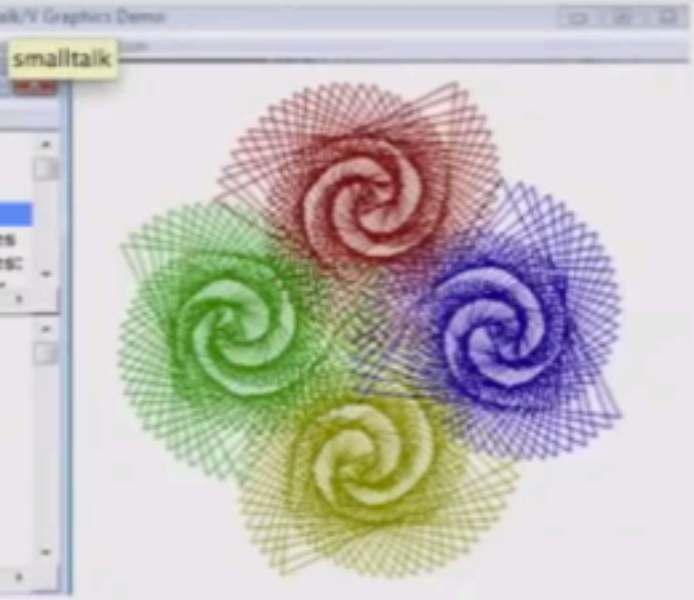
```
redraw
boundingRectangle
move
```

Unbounded number of implementations:

```
Window, Menu, Letter, Shape, Curve, Image, Video, ...
```

What Simulation and GUIs have in common





Both need a way to execute a *fixed* API with an *unknown* implementation.

It's *possible* to do this in a procedural language such as C.

But it's too *cumbersome*, so people wanted object-oriented languages instead.

What does this have to do with FP?

Just like OOP did then, FP has lots of methodological advantages:

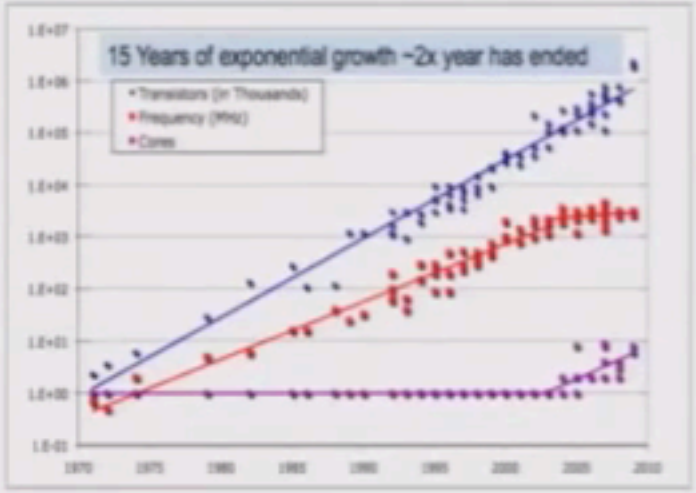
- Fewer errors
- Better modularity
- Higher-level abstractions
- Shorter code
- Increased developer productivity

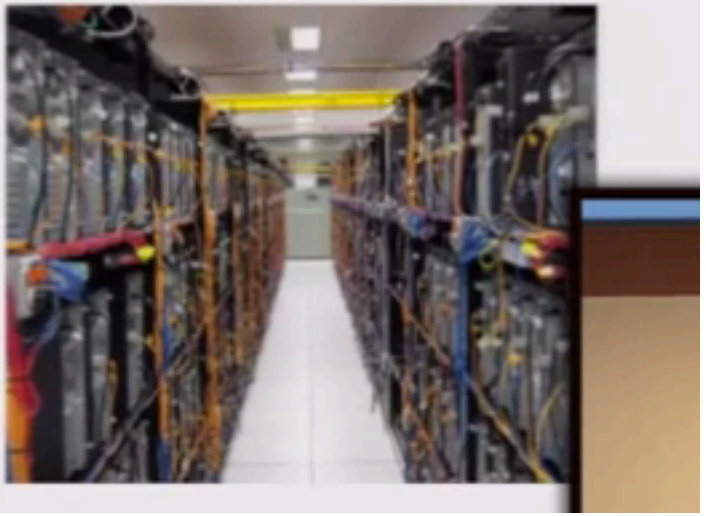
But these alone are not enough for mainstream adoption (after all FP has been around for 50 years)

Need a catalyzer, something that sparks initial adoption until the other advantages become clear to everyone.

A Catalyzer

- Two forces driving software complexity:
  - Multicore (= parallel programming)
  - Cloud computing (= distributed programming)
- Current languages and frameworks have trouble keeping up (locks/threads don't scale)
- Need better tools with the right level of abstraction





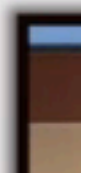
Parallel – how to make use of multicore, GPUs, clusters?

Async – how to deal with asynchronous events

Distributed – how to deal with delays and failures?

Mutable state is a liability for each one of these!

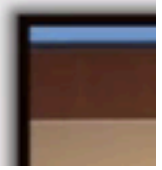
- Cache coherence
- Races
- Versioning



The Essence of Functional Programming

Concentrate on **transformations** of **immutable** values

instead of **stepwise modifications** of **mutable** state.



What about Objects?

So, should we forget OO and all program in functional programming languages?

No! What the industry learned about OO decomposition in analysis and design stays valid.

Central question: What goes where?

In the end, need to put things somewhere, or risk unmanageable global namespace.

New Objects

Previously:

> *"Objects are characterized by state, identity, and behavior." (Booch)*

Now:

Eliminate or reduce mutable state.

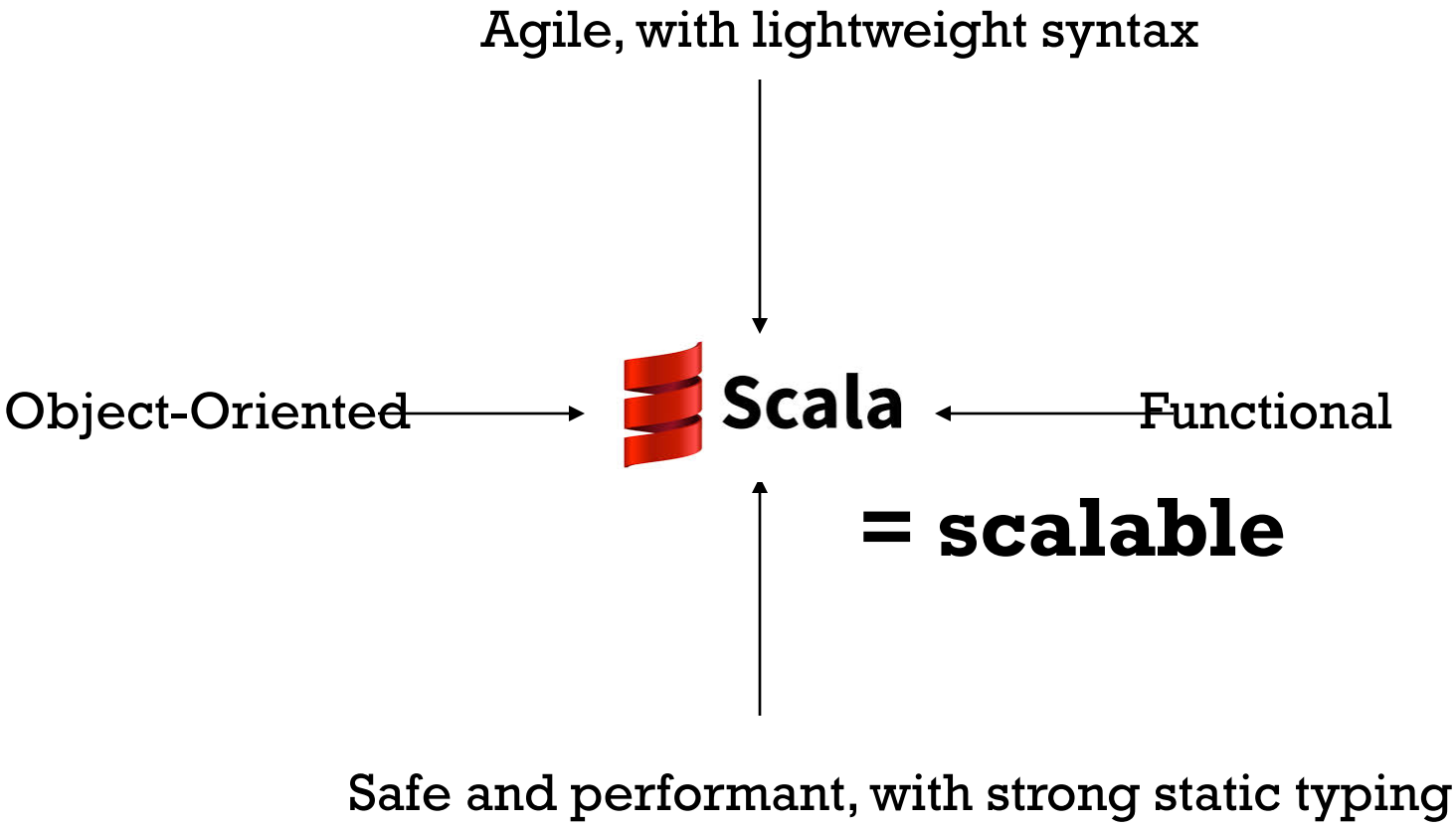
Structural equality instead of reference identity.

Concentrate on functionality (behavior)

Scala 是什么编程语言?

- 一种可扩展的(scalable)语言
- 运行在JVM上，和Java函数库无缝互操作
- 混合了面向对象和函数式编程的概念
- 静态强制类型语言
- 清晰、简洁的编程风格

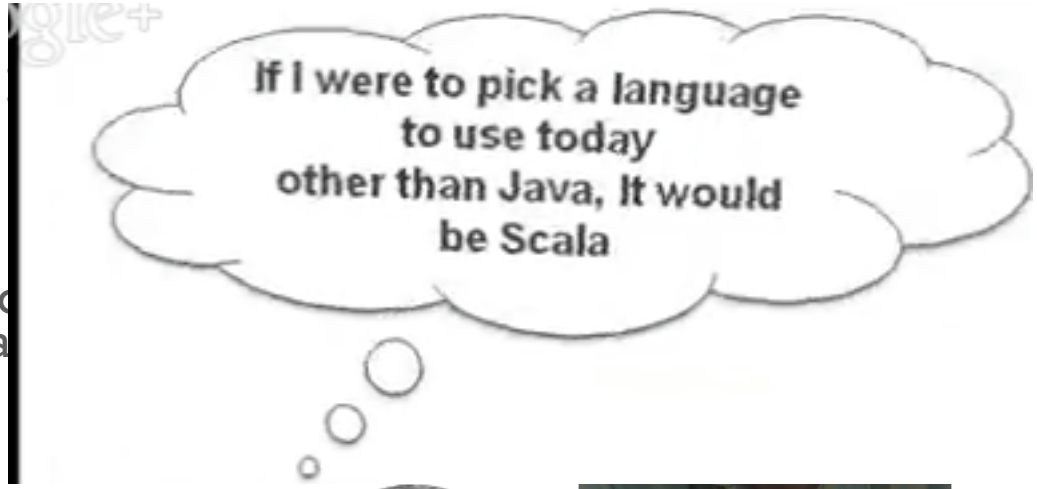
Scala 是什么编程语言?



SCALA

Pascal
Turing Award

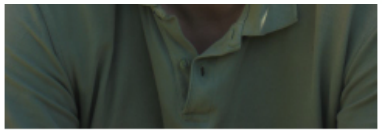
- 开始于2001, 首次发布于 2004.
- 发明人：**Martin Odersky**
  - Programming Methods Labo
    Institute of Technology in La
  - ACM Fellow
- Typesafe




Martin Odersky
Chairman & Co-Founder


James Gosling


Rod Johnson

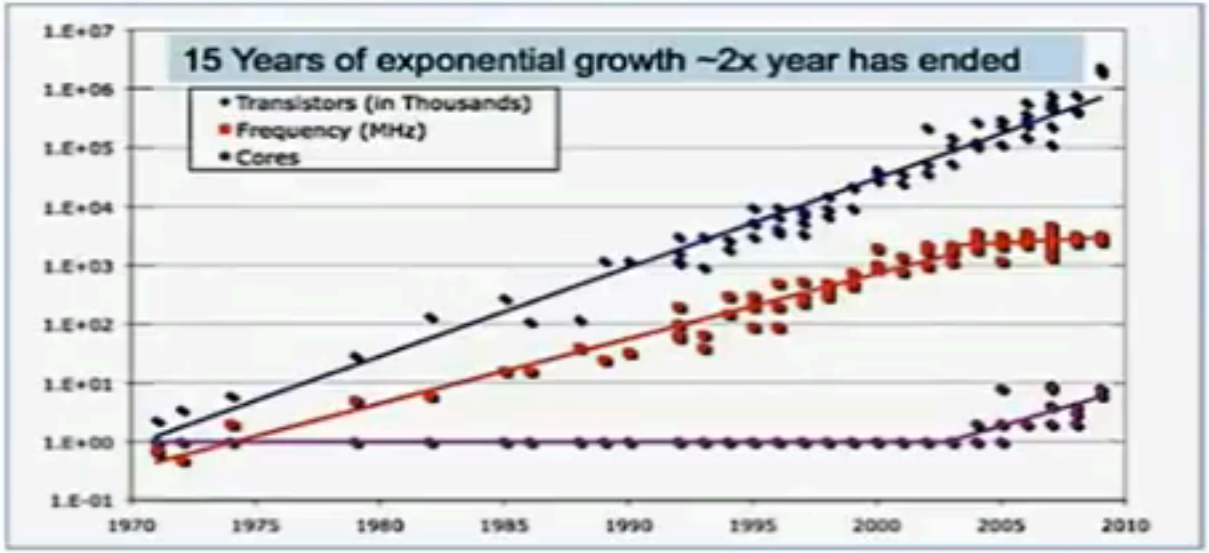


- Backed by Greylock Partners, Shasta Ventures, Bain Capital Ventures and Juniper Networks, Typesafe is headquartered in San Francisco with offices in Switzerland and Sweden.

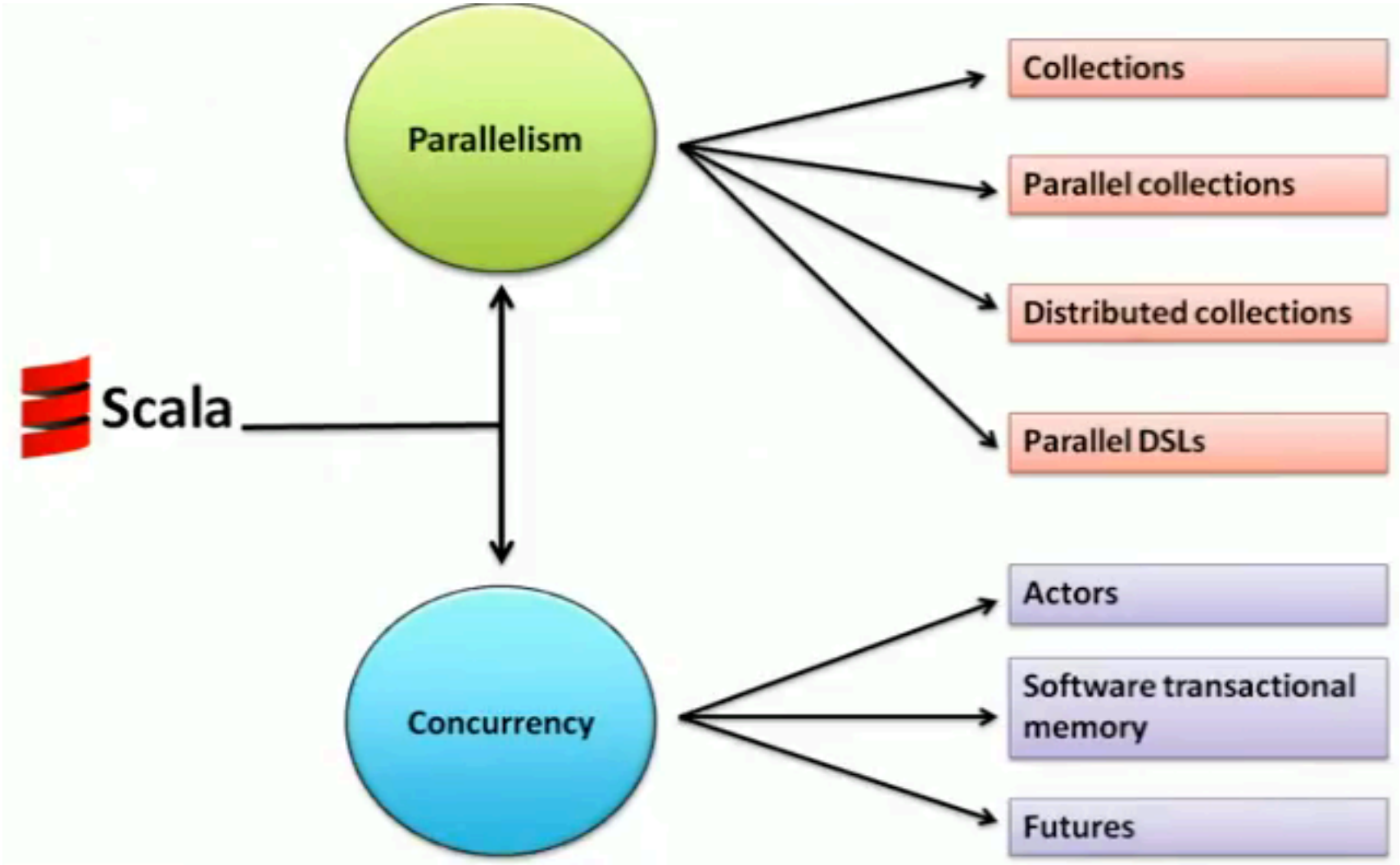
James Stratchan

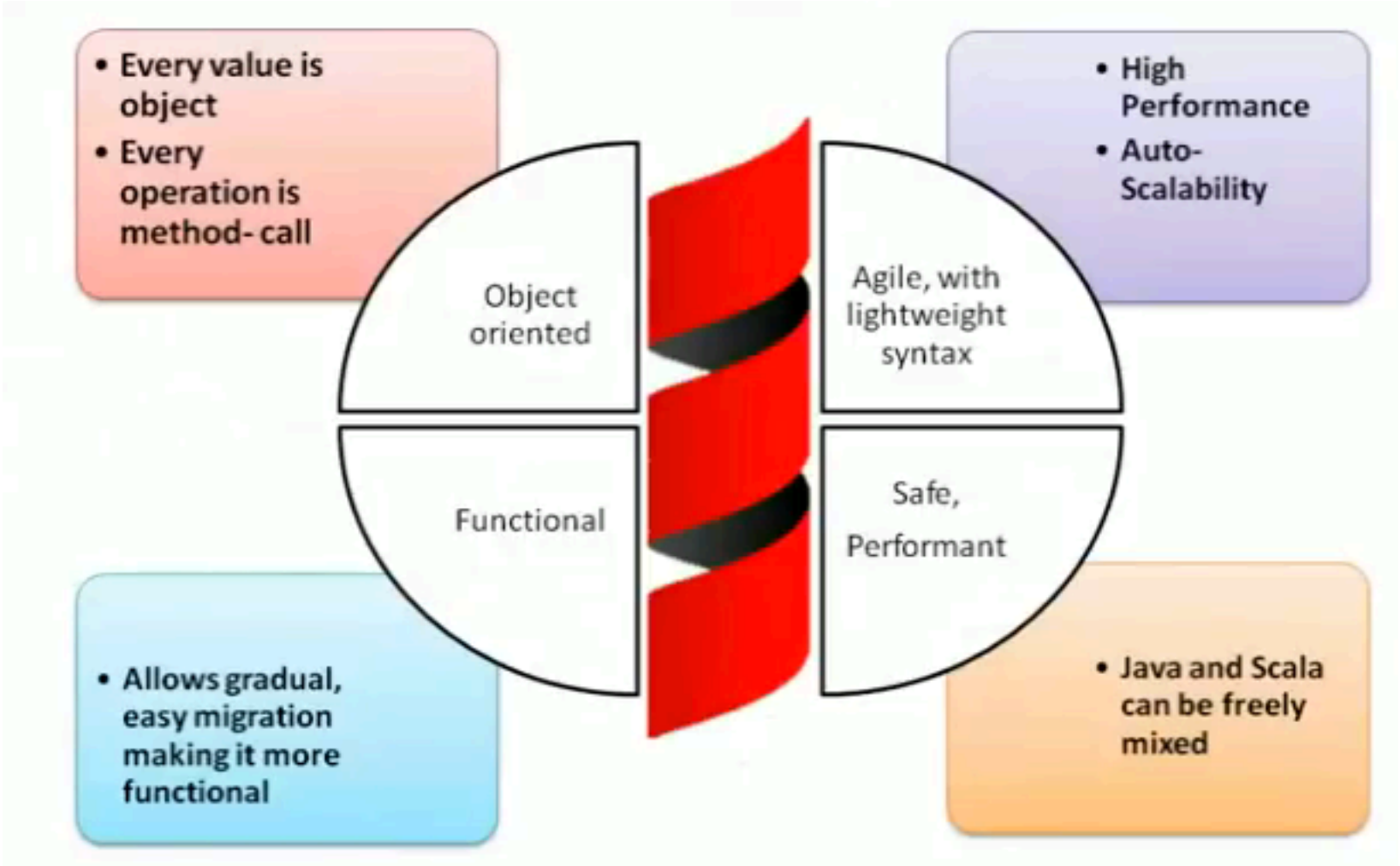
(Father of Groovy)

The world of mainstream software is changing

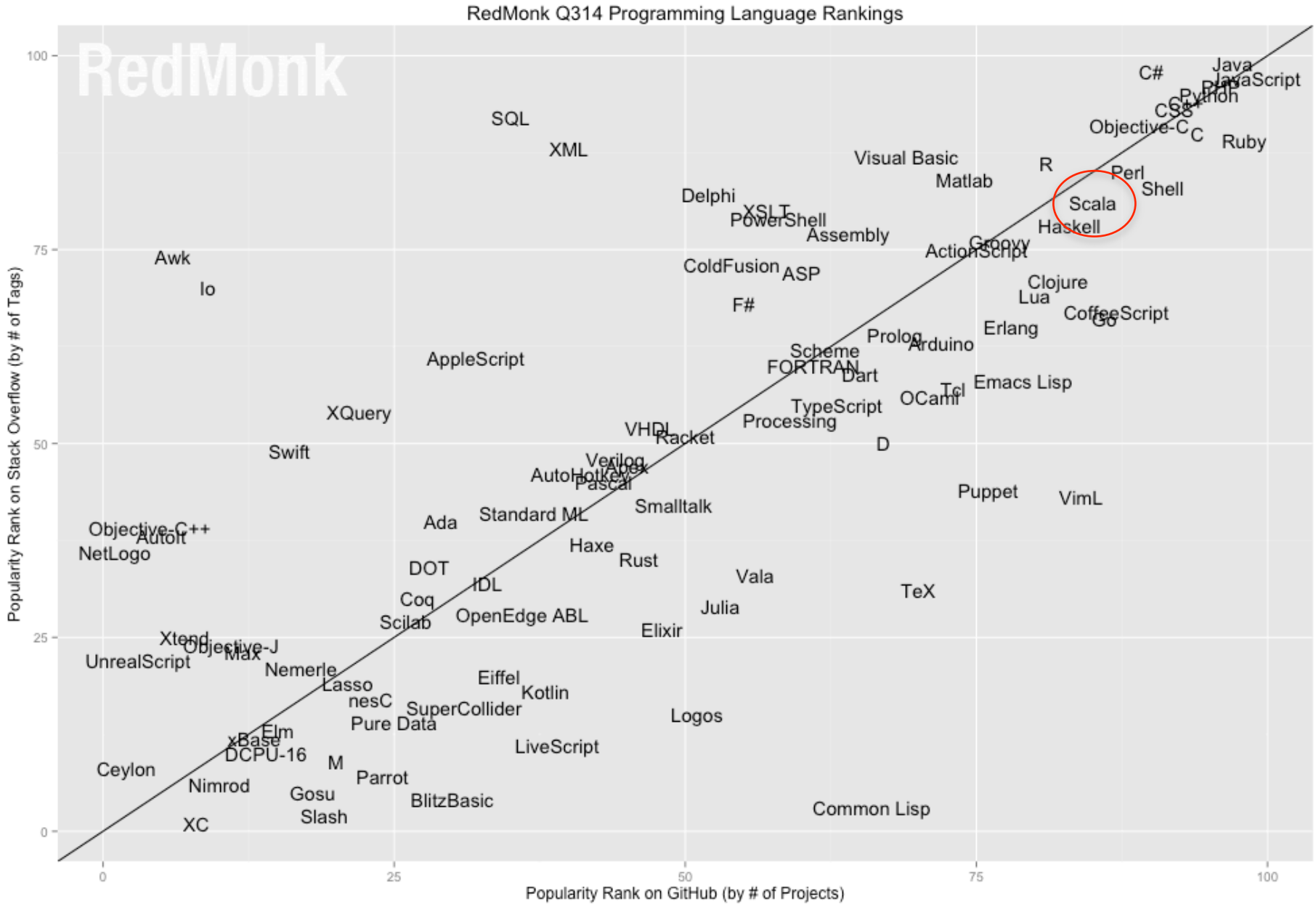


Moore's law now achieved by increasing # of cores not cloak cycles	**Huge volume workloads that require horizontal scaling**
"PPP" Grand Challenge	Parallel programming and Concurrent programming


Parallelism
Collections
Parallel collections
Distributed collections
Parallel DSLs
Scala
Concurrency
Actors
Software transactional memory
Futures


• Every value is object
• Every operation is method- call
• High Performance
• Auto-Scalability
Object oriented
Agile, with lightweight syntax
Functional
Safe, Performant
• Allows gradual, easy migration making it more functional
• Java and Scala can be freely mixed

RedMonk ranking 2014/6



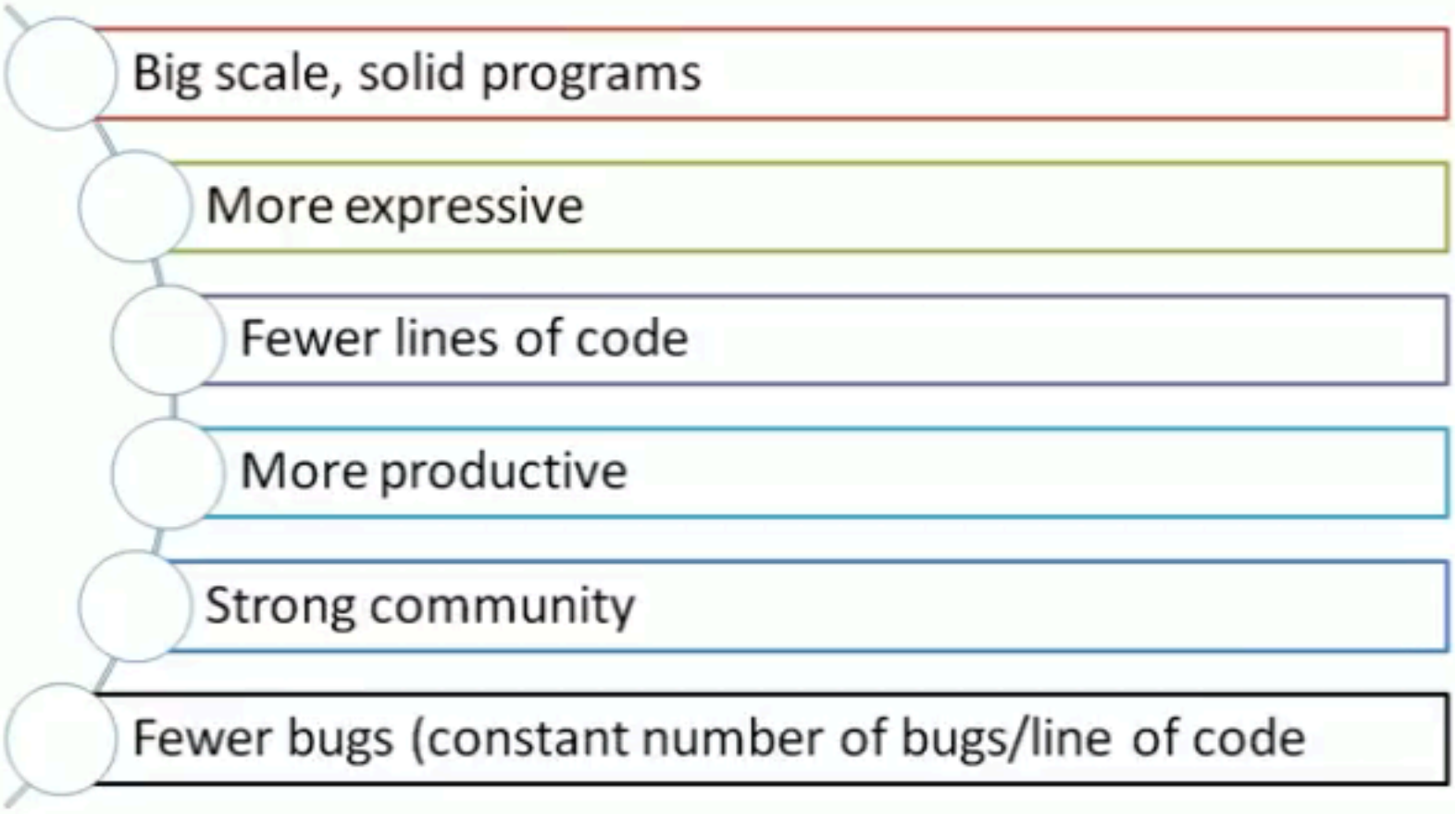
工业界应用广泛


Scala


Genentech
IN BUSINESS FOR LIFE
LinkedIn
Walmart
box
JUNIPER NETWORKS
NCL
NORWEGIAN CRUISE LINE
workday.
DIRECTV
GILT GROUPE
Angie's list.
white pages
here
Scotiabank
Saks Fifth Avenue
hp
THE HUFFINGTON POST
TOMAX retail.net


coursera
simpleBI
foursquare
primetalk
theguardian
flowdock
abiquo

Why In the Enterprise?



现代软件开发者所选择的10大技术之一

SOURCE: Typesafe

July 23, 2014 17:54 ET

Scala Named a "Top 10 Technology for Modern Developers"

RebelLabs Study Based on Developer Feedback and Market Data Affirms Popularity of Scala as Powerful Language for the JVM

SAN FRANCISCO, CA--(Marketwired - July 23, 2014) - Typesafe, provider of the world's leading Reactive platform and the company behind Play Framework, Akka, and Scala, today announced that Scala has been selected as one of the "10 Kick-Ass Technologies Modern Developers Love" in an independent study conducted by RebelLabs, the research and content think tank of ZeroTurnaround.

The study set out to reveal the technologies most "loved and chosen specifically by geeks" based on selection criteria of "Market Data" (raw % of market, relative size of market, level of fragmentation), "Developer Feedback" (i.e. explicit responses from surveys), "Amount of Buzz" (press coverage, social media share, community presence), and "Anecdotal Evidence" (personal conversations, stories, gut feeling). In being recognized, Scala joined other hot developer technologies that got a nod, including (in alphabetical order): Confluence, Git, Gradle, Groovy, IntelliJ IDEA, Jenkins, JIRA, MongoDB and Tomcat + TomEE.

"Scala has evolved a lot since its inception in 2004," said Martin Odersky, creator of Scala. "Its user base has grown to a size I could have never imagined back then, and has been picked up in industries ranging from major enterprises and financial institutions to startups. Being recognized by RebelLabs is a huge honor and validates the work we're doing to create an elegant and powerful programming language."

According to the study, when developers were asked "which alternative JVM (Java Virtual Machine) language they would be most interested in learning," 47% listed Scala as their next choice. The study cites high interest in Scala's sbt, and Typesafe's "focus and the community backing of the Scala ecosystem of tools, including Akka and Play" among key contributing factors for the rise of Scala's popularity as a language.

Additional Resources:

- Full Report: 10 Kick-Ass Technologies Modern Developers Love
- Typesafe Reactive Platform / Scala

Scala的风格

Bridging the Gap

- Scala acts as a bridge between the two paradigms.
- To do this, it tries to be:
  - orthogonal
  - expressive
  - un-opinionated
- It naturally adapts to many different programming styles.

A Question of Style

Because Scala admits such a broad range of styles, the question is which one to pick?

Certainly, Scala is:

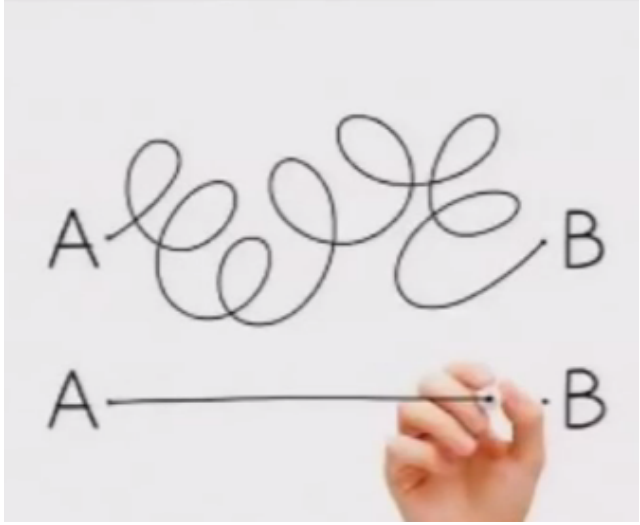
- Not a better Java
- Not a Haskell on the JVM

But what is it then?

I expect that a new fusion of functional and object-oriented will emerge.

Scala 风格

#1 Keep it Simple



#2 Don't pack too much in one expression

- I sometimes see stuff like this:

```
jp.getRawClasspath.filter(
  _.getEntryKind == IClasspathEntry.CPE_SOURCE).
  iterator.flatMap(entry =>
    flatten(ResourcesPlugin.getWorkspace.
      getRoot.findMember(entry.getPath)))
```

- It's amazing what you can get done in a single statement.
- But that does not mean you have to do it.

#3 Prefer Functional

Prefer Functional ...

- By default:
  - use vals, not vars.
  - use recursions or combinators, not loops.
  - use immutable collections
  - concentrate on transformations, not CRUD.
- When to deviate from the default:
  - Sometimes, mutable gives better performance.
  - Sometimes (but not that often!) it adds convenience.

#4 ... But don't diabolize local state

Why does mutable state lead to complexity?

It interacts with different program parts in ways that are hard to track.

=> Local state is less harmful than global state.

More local state examples

The canonical functional solution uses a `foldLeft`:

```
val (totalPrice, totalDiscount) =
  items.foldLeft((0.0, 0.0)) {
  case ((tprice, tdiscount), item) =>
    (tprice + item.price,
     tdiscount + item.discount)
  }
}
```

Whereas the canonical imperative solution looks like this:

```
var totalPrice, totalDiscount = 0.0
for (item <- items) {
  totalPrice += item.price
  totalDiscount += item.discount
}
```

#5 Careful with mutable objects

Mutable objects tend to encapsulate global state.

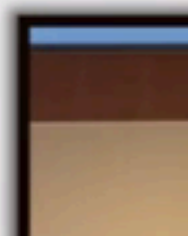
"Encapsulate" sounds good, but it does not make the global state go away!

=> Still a lot of potential for complex entanglements.

#6 Don't stop improving too early

- You often can shrink code by a factor of 10, and make it more legible at the same time.
- But the cleanest and most elegant solutions do not always come to mind at once.
- That's OK. It's great to experience the pleasure to find a better solution multiple times.

So, keep going ...



SCALA 特点

- 结合面向对象和函数式编程模式，抽象性能好，组件重用性强
  - 每个值是一个对象
  - 每个函数也是一个值，函数也可作为输入输出参数
- 类对象组合灵活（composition by mixing with traits）
- 数据类型解析，通过模式匹配（case class, pattern matching）
- 类型自动分析，软件健壮性好（type inference）
- 面向JVM，和Java可以自由结合联合使用

SCALA优势

- 并行分布计算（parallel, concurrent, distributed computing）
  - 数据集合上的并行计算
  - 对未来事件的抽象，支持异步并发处理（Futures, Promises, async）
  - 基于消息传递的并行分布计算（Actors，Akka）
- 构建组件系统，可扩展性强（component systems, scalable）
- 面向JVM，可自由利用Java库
- 语法简洁，开发效率高，软件测试和可维护性强

软件安装

- 开发环境 Eclipse IDE http://scala-ide.org
- Typesafe reactive platform (Web-based Tool; Templates)
  http://www.typesafe.com/platform
- SBT(Scala Building Tool; Command-line-based)
  http://www.scala-sbt.org/release/docs/Getting-Started/Setup.html
- 相关网页
  - http://www.scala-lang.org
  - http://www.typesafe.com
  - http://akka.io
  - http://www.playframework.com

如何开始一个SCALA项目

1. 建立项目模版
2. Scala Eclipse-based IDE导入项目进行编辑
3. SBT 修改更新项目设置
4. 重复2，3步进行维护调整

自学参考

- Scala Standard Library API http://www.scala-lang.org/api/
- Scala School, a tutorial by Twitter http://twitter.github.com/scala_school/
- A Tour of Scala http://docs.scala-lang.org/tutorials/tour/tour-of-scala.html
- Scala Overview on StackOverflow http://stackoverflow.com/tags/scala/info
- Scala By Example http://www.scala-lang.org/docu/files/ScalaByExample.pdf
- Scala Cheatsheet http://docs.scala-lang.org/cheatsheets/
- Books

Programming in Scala. Martin Odersky, Lex Spoon and Bill Venners. 2nd edition. Artima 2010

Scala for the Impatient. Cay Horstmann. Addison-Wesley 2012

Scala in Depth. Joshua D. Suereth. Manning 2012.