Why Scala?
Alex Payne and Dean Wampler
@al3x, @deanwampler
Programming Scala

programmingscala.com
Why do we need a new language?
#1
We need *Succinct Code* ...
.... for productivity.
.... for less debugging.
.... for less overhead.
Infix Operator Notation

"hello".+("world")

\textit{same as}

"hello" + "world"
Not Just for “Operators”

"hello".compareTo("world")

"hello" compareTo "world"

Great for DSLs!
Type *Inference*

// Java
```java
HashMap<String, Person> persons =
    new HashMap<String, Person>();
```

VS.

// Scala
```
val persons =
    new HashMap[String, Person]
```
// Scala
val persons = new HashMap[String, Person]

no () needed.
Semicolons inferred.
val meaningOfLife = 42
val bestScalaBook = "Programming Scala"
val bestPeopleOnTwitter = "@al3x" :: "@deanwampler" :: Nil
class Point {
    private double x;
    private double y;

    public Point(double x, double lastName, int age) {
        this.x = x;
        this.lastName = lastName;
    }

    public void double getX() { return this.x; }
    public void setX(double x) {
        this.x = x;
    }

    public void double getY() { return this.y; }
    public void setY(double y) {
        this.y = y;
    }
}

Typical Java
class Point(
    var x: Double,
    var y: Double)
class Point(
    val x: Double,
    val y: Double)
case class Point(
  x: Double,
  y: Double)
We need Functional Programming
... for concurrency.
... for concise code.
... for correctness.
Example with Akka Actors, REST, Pattern Matching, ...
// "Atmosphere" REST annot.
@Path("/ajax")
class RestfulDataPublisher {
  @GET
  @Path("{command}")
  @Produces(Array("application/json"))
  def handleGET(...
def handleGET(@PathParam("command") command: String): String = {
  command match {
    case "service_list" =>
      getAvailableServices
    case "heartbeats" =>
      getHeartBeats
    case _ =>
      unknown(command)
  }
}
def getHeartBeats = {
  val futures = for {
    s <- services
    server <- allServersFor(s)
  }
  yield (server !!! HeartBeat)
  Futures.awaitAll(futures)
  ...
}
Futures.awaitAll(futures)
val results = for {
  future <- futures
  result <- future.result
} yield result
val all = results reduceLeft(
  (r1, r2) => r1 merge r2 )
  compact(render(all))
}
#3

We need a better *Object Model*
... for **composability**.

... for **scalable designs**.
Traits

class MyActor extends Actor with Logging {
  ...
  log.error("fail")
  ...
}


Scala’s *Thesis:* Functional Programming Complements Object-Oriented Programming

*Despite surface contradictions...*
But we want to keep our investment in Java/C#.
Scala is...

- A JVM and .NET language.
- Functional and object oriented.
- Statically typed.
- An improved Java/C#.
Scala Summit
Lineup
Akka: Simpler Scalability, Fault-Tolerance, Concurrency & Remoting through Actors

Jonas Bonér
Simple Build Tool

Mark Harrah
Specs & Scala,
Tips and Tricks for a Friendly DSL Syntax

Eric TorreBorre
Lift: Quick and Fun

David Pollak
Rewiring Android with Scala

Nathan Hamblen
Scala in Practice

Alex Payne, Coda Hale
Scala’s Powerful Language Features
Don’t forget:
Ignite OSCON, O’Reilly Open Source Awards

Tonight: 7-9
Thanks!

al3x@alex.net
@alex
dean@deanwampler.com
@deanwampler

programmingscala.com