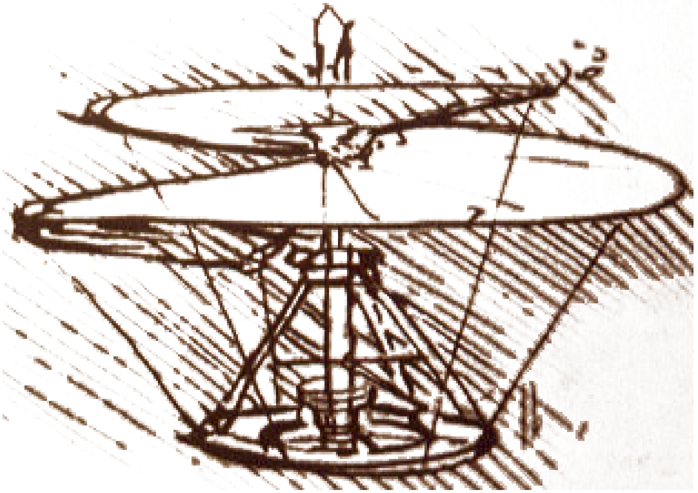
Evolving the Java platform



Ola Bini
JRuby Core Developer
ThoughtWorks Studios

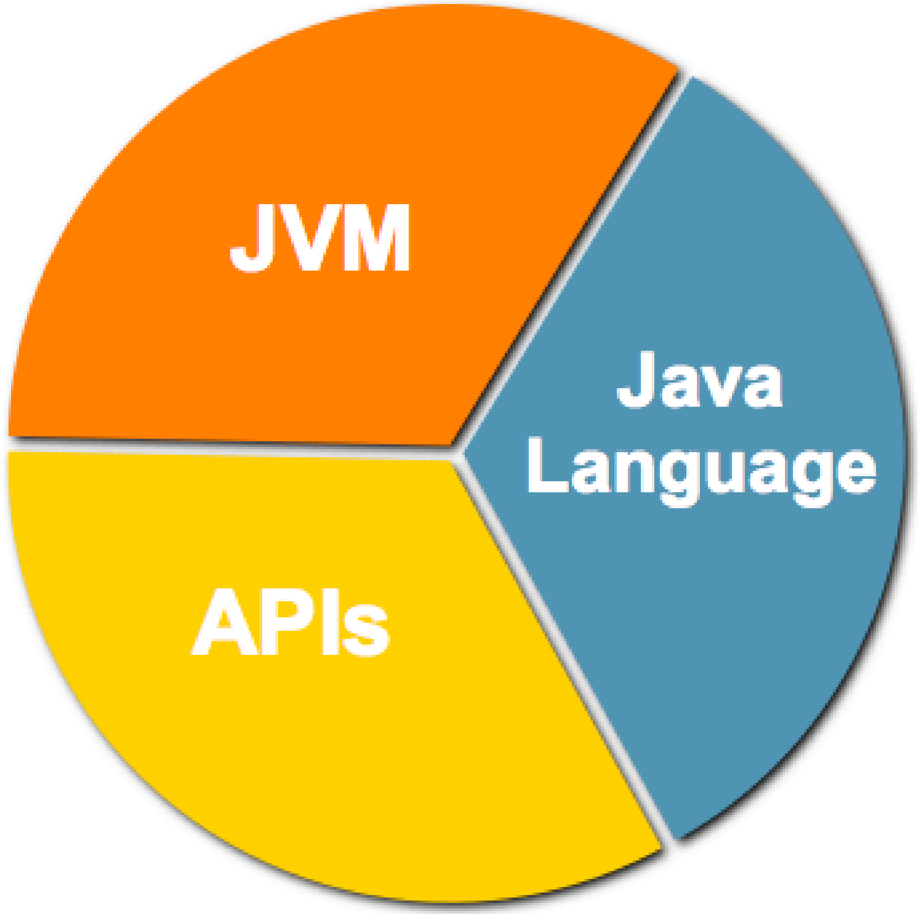
About me

- Ola Bini
- From Stockholm, Sweden
- JRuby Core Developer
- ThoughtWorks Studios
- Member of the JSR292 expert group
- Programming language geek

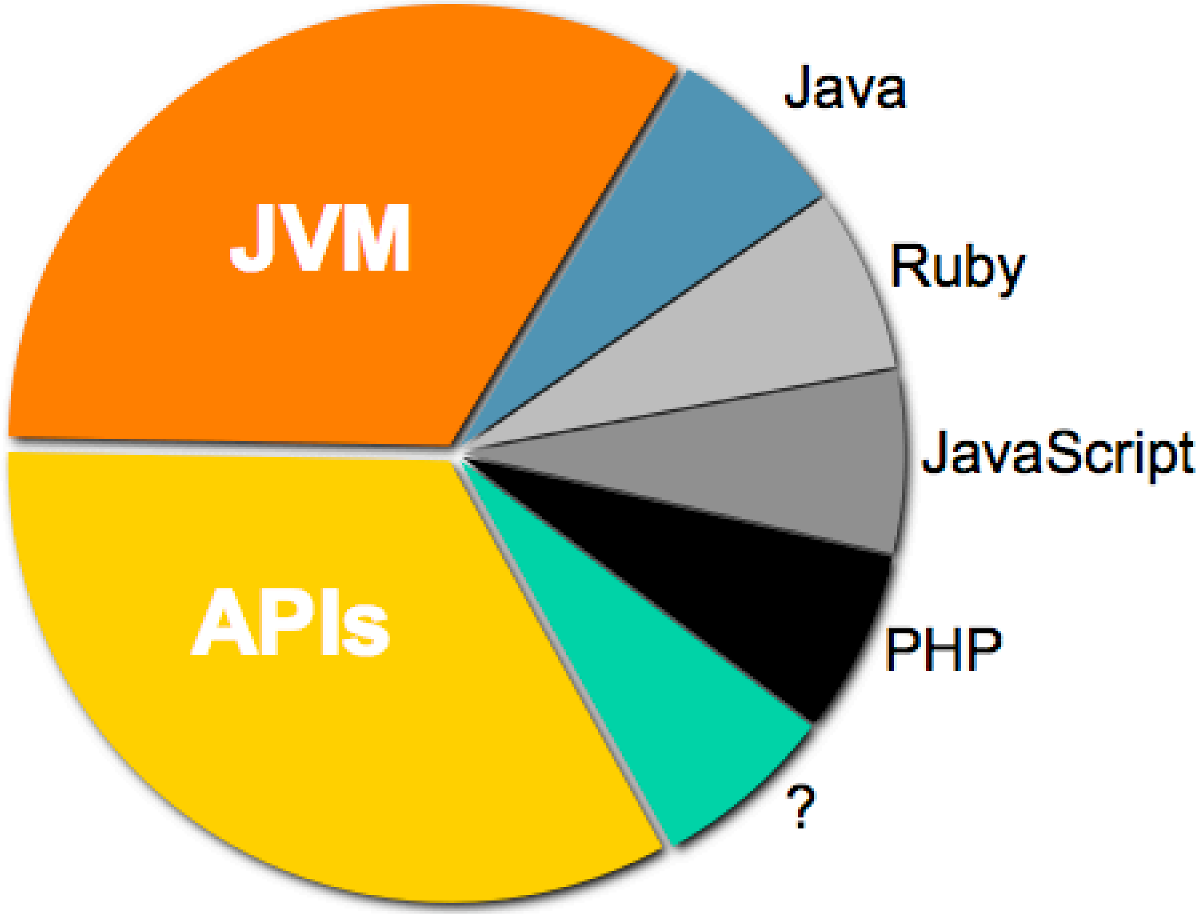
Agenda

- Other languages?
- The Java Virtual Machine
- New language features
- The DaVinci machine
- Java and Ruby
- Q&A

The Java platform



The Java platform



Other languages

Hecl
Jacl
Clojure
Ync/Javascript
JoyJ
v-language
CAL
Aardappel
Funnel
Mini
PLAN
Sixx
BDC Scheme
ABCL
Lili
Jatha
Bigloo
SISC
Lisp
PS3i

HotScheme
webLISP
Jaja
JScheme
Skij
Kawa
uts
JBasic
Mapyrus
CONVERT
HotTEA
COCOA
NetLogo
StarLogo
AJLogo
Turtle Tracks
rLogo
Yoyo
TermWare
XProlog

tuProlog
JLog
LL
javalog
SmallWorld
Bistro
Talks2
Obol
Groovy
Nice
Scala
Anvil
dSelf
Hojo
Correlate
MetaJ
Sather
Quercus
FScript
Sleep

WLShell
JudoScript
JRuby
Jickle
Rhino
BeanShell
Resin
Jython
Pnuts
Janino
Join Java
JMatch
iScript
Yassl
Yoix
W4F
PERCobol
Bex Script
Demeter/Java
CKI Prolog

Other languages: Clojure

- Lisp dialect (dynamic, code as data)
- Designed for the JVM
- Powerful macros
- Good interoperability with Java
- Functional programming language (mostly)
  - Immutable data structures
- Concurrency
  - Shared transactional memory
  - Actors

Other languages: Groovy

- Dynamically, strongly typed
- Object oriented
- Designed for the JVM
- Inspired by Python, Ruby and Smalltalk
- Good integration with Java
- Mostly precompiled

Other languages: Scala

- Multiparadigm language
  - Object orientedness
  - Functional programming natural
- Designed for the JVM
- Concurrency: Actors
- Includes many advanced language features
  - Pattern matching, closures, parametric polymorphism
  - Sequence comprehensions, mixins, infix or postfix statements

The Java Virtual Machine

- Virtual machines are the norm
- CPU cycles are cheap enough for JIT, GC, RTT, etc
- The JVM is a great virtual machine
  - Flexible online code loading (with safe bytecodes)
  - GC & object structure
    - Mature and provides lots of algorithms and tuning opportunities
  - Reflective access to classes & objects
  - Tools (JMM, JVMTI, dtrace)
  - Good libraries & a nice language to write more

The Java Virtual Machine

- Optimizing Just-In-Time compiler
- Clever performance techniques
  - Type inference
  - Customization
  - Profiling
  - Deoptimizing
  - Fast/slow paths
  - etc.
- The JVM is <u>mature</u>

Needs of higher level languages

- High level languages often require:
    - Very late binding (runtime linking, typing, code gen)
    - Automatic storage management (GC)
    - Environmental queries (reflection, stack walking)
    - Exotic primitives (tailcalls, bignums, call/cc)
    - Code management integrated with execution
    - Robust handling of incorrect inputs
    - Helpful runtime support libraries (regexps, math, ...)
    - A compiler (JIT and/or AOT) that understands it all
- The JVM has some of this, but not all

What's missing?

- Dynamic invocation
- As always, higher performance

What’s missing?

- Dynamic invocation
- As always, higher performance
- Lightweight method objects
- Lightweight bytecode loading
- Continuations and stack introspection
- Tails calls and tail recursion
- Tuples and value-oriented types
- Immediate wrapper types
- Symbolic freedom (non-Java names)

Dynamic invocation

- Non-Java call site in the bytecodes
- Language-specific handler
  - Determines linking at runtime
  - Works in a reflective style
  - Installs direct (non-reflective) methods
- Stateful: can be updated or revoked over time
- Any dynamic language will benefit greatly

ThoughtWorks

Lightweight method handles

- Method handle = lightweight reference to a method
- Like `java.lang.reflect.Method`, but much, much lighter
- Caller invokes without knowing method’s name, etc
- Call runs at nearly the speed of Java call
- Required to glue together dynamic call sites
- Requires VM and/or library support for common adaptation patterns (currying, receiver check, varargs, etc)

Lightweight bytecode loading

- Anonymous classes
- Faster and more reliable loading and unloading
- Little interaction with system dictionary or class loaders
  - “class names considered harmful”
- Library-directed code customization
- No more one-classloader-per-class

Continuations

- Stack manipulation (call/cc)
- Extremely powerful
- Allows computations to be paused and resumed
- Could be implemented using **`copyStack`** and **`resumeStack`**.

```
(+ 1 (call/cc
          (lambda (k)
             (+ 2 (k 3)))))     ; => 4
```

Tail calls

- Allows iteration to be modeled as recursion
  - Without the performance problems of this
- Common pattern in many languages
- Allow computations to be more closely modeled on mathematical formulas
- Factorial in Scheme:

```
(define (factorial n)
  (define (fac-times n acc)
    (if (= n 0)
        acc
        (fac-times (- n 1) (* acc n))))
  (fac-times n 1))
```

Tuples and value types

- Quite common pattern in Java:

```
return new Object[]{42, “something”};
```

- Tuples are basically a named struct
- Ordered pairs, etc
- Other value objects: Lisp-style bignums?

Symbolic freedom

- Allow any identifier as name
- JVM identifiers originally based on the Java language
- No real reason for this
- Support for Ruby style names
  - empty?
  - value=
  - clear!
- Canonical name mangling

Interface injection

- Give existing classes a new supertype
- Either an interface
- ... or an interface plus new method implementations
- Or Mixins
- There are several tactics to make this quite simple for the VM

Performance

- Bytecode analysis
  - Less-static bytecode shapes
  - Class.isInstance, Arrays.copyOf
- Faster reflection
- Faster closure-type objects
- Escape analysis to remove auto-boxing

What about Closures?

- There are several closures proposals right now
- All of them except CICE benefits from method handles
- Interface injection would also be beneficial
- But - closures doesn’t require any of this
- The machinery is already there
- It will just be simpler to implement with this available

The DaVinci Machine

- Evolutionary adaptation of the present JVM
- Open-ended experiment
  - Wild ideas are considered, but most prove useful
  - While incubating, features are disabled by default
- Eventual convergence
- Prototype JVM extensions to run non-Java languages efficiently
- First class architectural support (no hack or side-cars)
- New languages to co-exist gracefully with Java in the JVM

The DaVinci Machine

- Most of the features mentioned above have or will be implemented here
- Will eventually decide what makes it in Java 7
- Why do this?
  - Language implementers know what they want
    - ...and how to simulate it at 100x slowdown
  - VM implementors know what VMs can do
    - ...and how to make their languages sing
  - Let’s bring them together

Case study: Ruby on the JVM

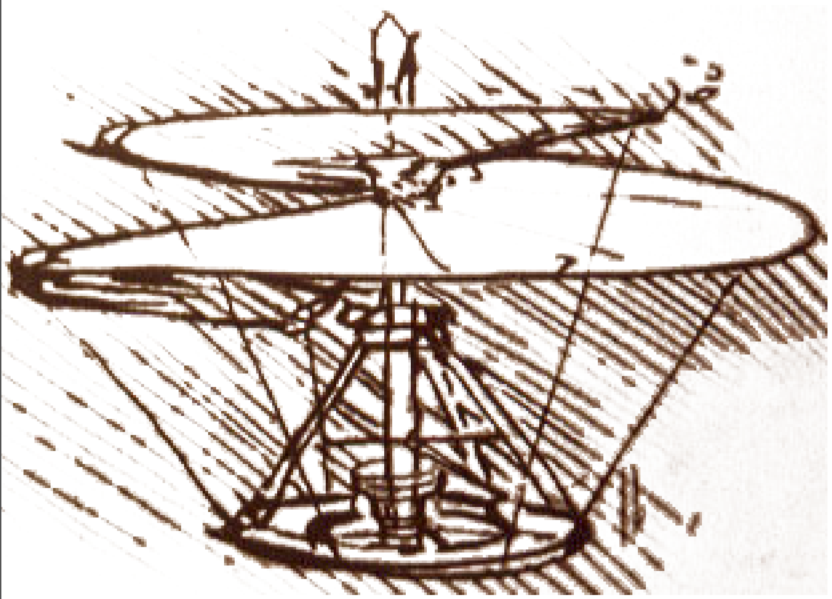
- JRuby
- Java implementation of the Ruby language
- Interpreter and Compiler
- Interpreter: slow
- Compiler: fast
    - But cumbersome
- Ruby is a complex language

JRuby Compiler pain

- AOT pain
  - Code bodies as Java methods need method handles
    - Are generated as adapter methods right now
  - Ruby is terse - compiled output extremely verbose
  - Mapping symbols safely
- JIT pain
  - Method body must live on a class
    - Class must live in separate ClassLoader to GC
    - Class name must be unique within that classloader
    - Gobs of memory used up working around all this

Compiler optimization pain

- Build-your-own dynamic invocation
  - Naive approach doesn’t perform (hash lookup, reflection)
- B-y-o reflective method handle logic
  - Handle-per-method means class+classloader per method
  - Overloaded signatures means more handles
  - Non-overloading languages introduce arg boxing cost
- B-y-o call site optimizations
  - ... and must make sure they don’t interfere with JVM optz
- We shouldn’t have to worry about all this



DEMO

Compilation output

JSR 292

- Supporting Dynamically Typed Languages
- Main feature:
  - invoke_dynamic
  - Hotswapping
- Representatives from JRuby, Groovy, Jython, among others
- Focus on VM support

The JVM Languages group

- Focus on library level support for languages running on the JVM
- Discussions about current painpoints
- Meta-object protocol
- Java method overload resolution at runtime
- Representatives from JRuby, Jython, Groovy, Pnuts, Nice, Ng, Scala, Clojure, and many more

Resources

- http://openjdk.java.net/projects/mlvm
- http://blogs.sun.com/jrose
- http://groups.google.com/group/jvm-languages
- http://lambda-the-ultimate.org
- http://www.scala-lang.org
- http://clojure.sourceforge.net
- http://groovy.codehaus.org

Q&A