



JS, Rust, Python and the Holy Graal

Lars Hupel
BOB
2020-02-28

INNOQ

The origins of Java

“

1996 – James Gosling invents Java. Java is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Sun loudly heralds Java's novelty. ”

– James Iry



Java: a short history

1994 first JVM version



Java: a short history

1994 first JVM version

2001 first versions of Jython and JRuby



Java: a short history

1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy



Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

2004 first version of Scala

Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

2004 first version of Scala

2006 Java 6 with JSR 223 (scripting)

JSR 223

Problem

Servlets are cool, but we want to write our web pages in PHP.

JSR 223

Problem

Servlets are cool, but we want to write our web pages in PHP.

Solution

Define an API for scripting languages to exchange objects with Java.

Scripting languages

- JSR 223 does not define "scripting language"
- wording implies untyped languages
- fundamental problem: most calls require reflection (slow)



Available implementations

- AppleScript
- BeanShell
- Groovy
- Jaskell
- Java
- JavaFX Script
- JavaScript
 - ▶ Rhino
 - ▶ Nashorn
- JUEL
- PHP
- Python
- Ruby
- Scheme
- Smalltalk
- Tcl
- ...

Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

2004 first version of Scala

2006 Java 6 with JSR 223 (scripting)

Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

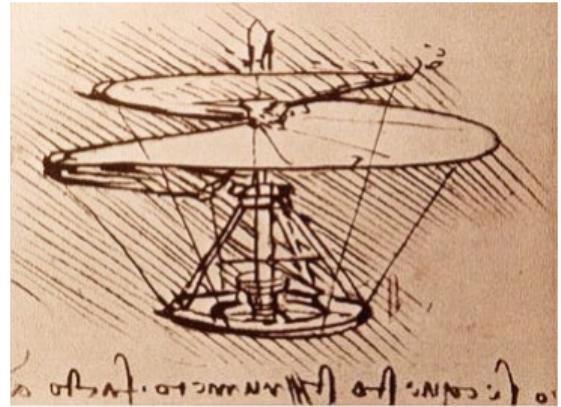
2004 first version of Scala

2006 Java 6 with JSR 223 (scripting)

2011 Java 7 with JSR 292 (invokedynamic)

Da Vinci Machine

- started in 2007 as a VM playground
- subprojects: dynamic invocation, continuations, tail-calls, ...





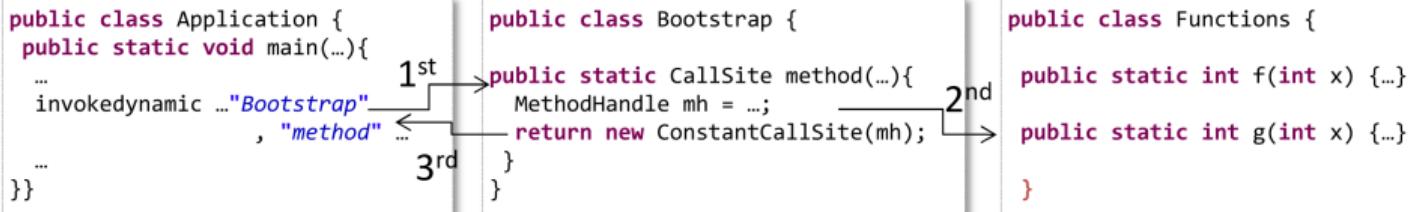
the Da Vinci Machine

*a multi-language renaissance
for the Java™ Virtual Machine
architecture*

[http://openjdk.java.net/
projects/mlvm/](http://openjdk.java.net/projects/mlvm/)



An invokedynamic instruction is *linked* just before its first execution.



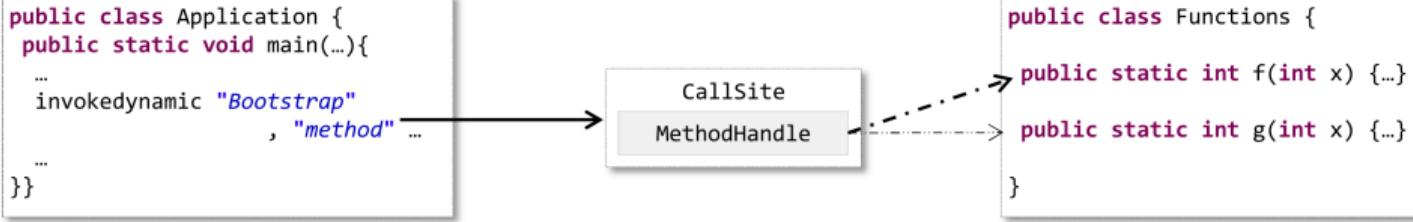
1

The call site then becomes *permanently linked* to the invokedynamic instruction.



2

A non-constant call site may be *relinked* by changing its method handle.



3

Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

2004 first version of Scala

2006 Java 6 with JSR 223 (scripting)

2011 Java 7 with JSR 292 (invokedynamic)

Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

2004 first version of Scala

2006 Java 6 with JSR 223 (scripting)

2011 Java 7 with JSR 292 (invokedynamic)

2014 Java 8 with Nashorn

Java: a short history



1994 first JVM version

2001 first versions of Jython and JRuby

2003 first version of Groovy

2004 first version of Scala

2006 Java 6 with JSR 223 (scripting)

2011 Java 7 with JSR 292 (invokedynamic)

2014 Java 8 with Nashorn

2019 first version of GraalVM

What is Truffle?

“ The Truffle framework allows you to run programming languages efficiently on GraalVM. It simplifies language implementation by automatically deriving high-performance code from interpreters. ”

Partial evaluation

A => B => C



Partial evaluation

A => B => C

+

A



Partial evaluation

A => B => C

+

A

=

B => C



Compilers perform partial evaluation all the time!

Constant folding

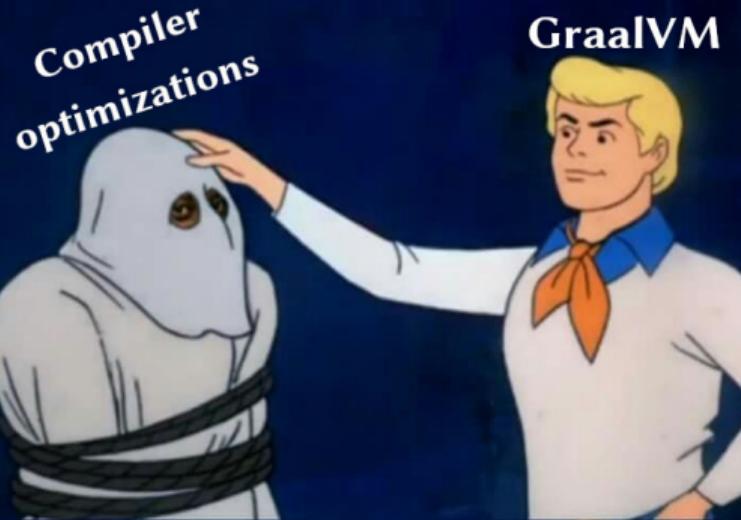
Bounds-checking elimination

Inlining

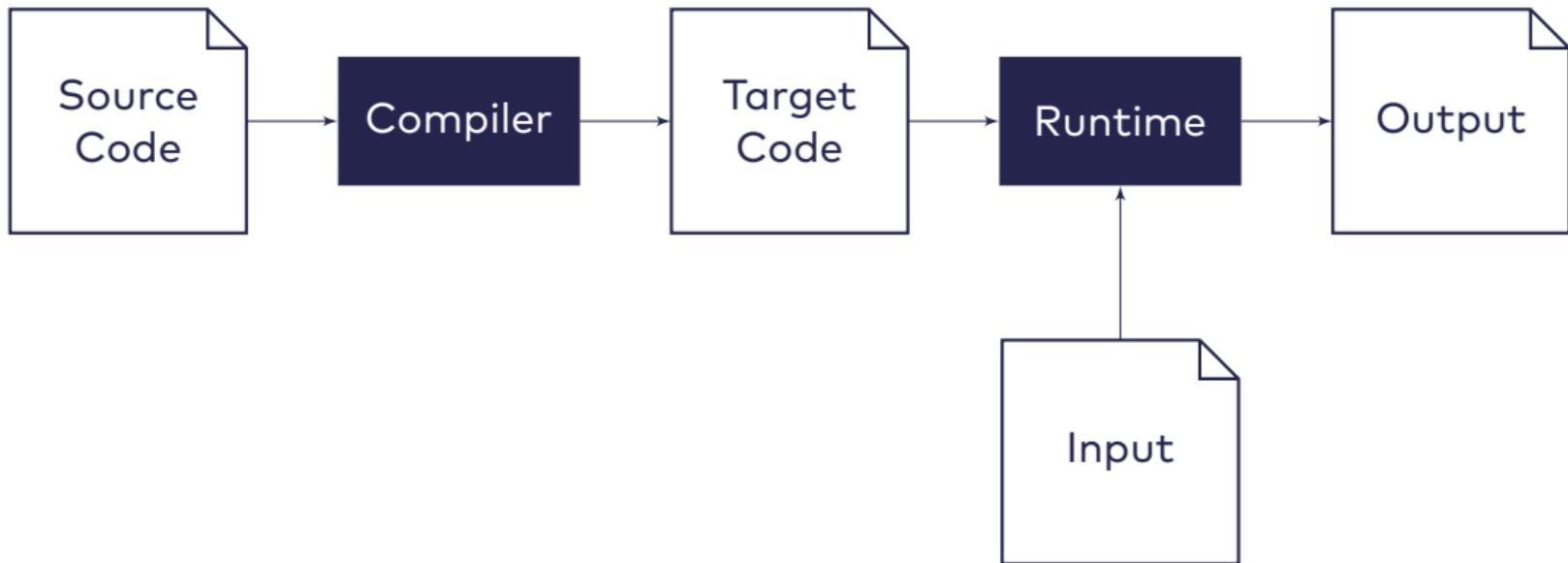
Loop unrolling

Deforestation

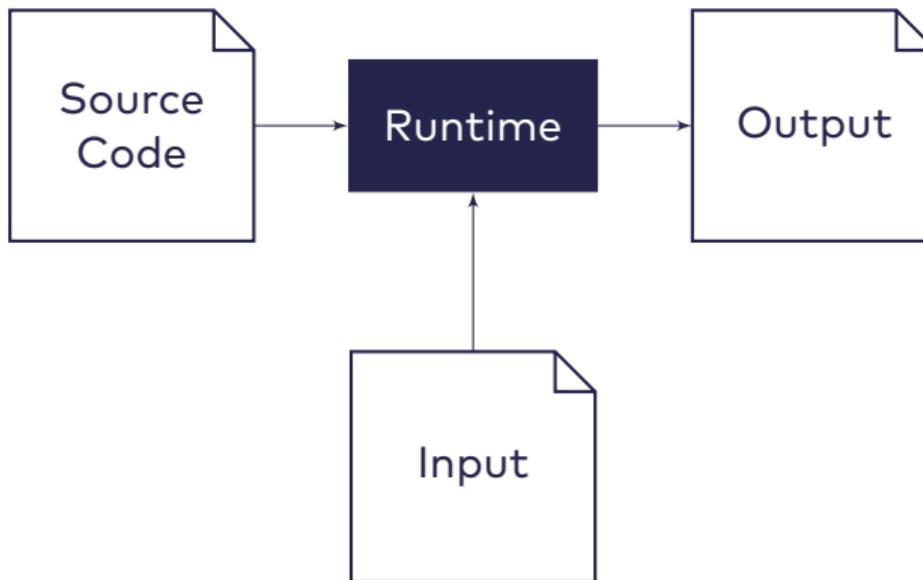
Dead code elimination



Compiler



Interpreter



Partial Evaluation of Computation Process— An Approach to a Compiler-Compiler

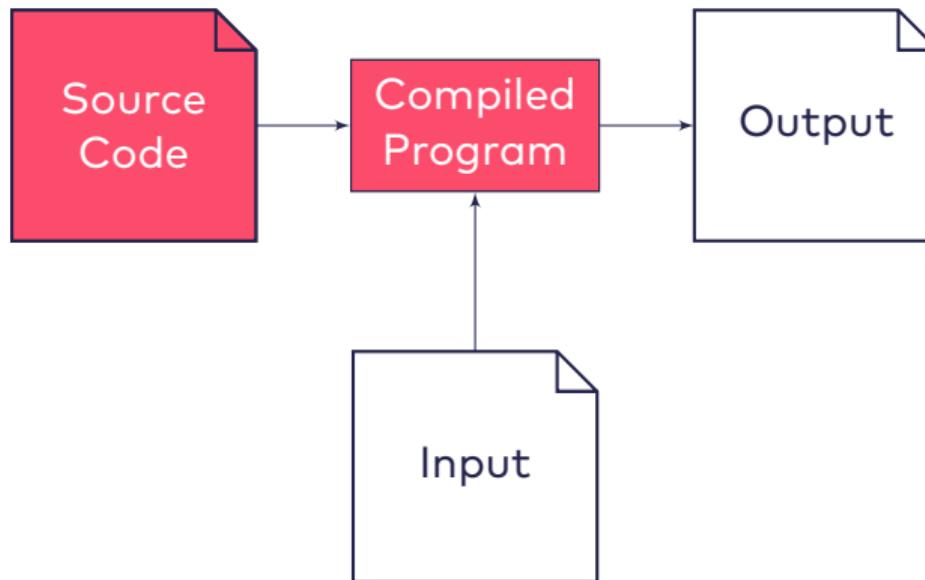
YOSHIHIKO FUTAMURA

Central Research Laboratory, Hitachi, Ltd., Kokubunji, Tokyo, Japan 185

Abstract. This paper reports the relationship between formal description of semantics (i.e., interpreter) of a programming language and an actual compiler. The paper also describes a method to automatically generate an actual compiler from a formal description which is, in some sense, the partial evaluation of a computation process. The compiler-compiler inspired by this method differs from conventional ones in that the compiler-compiler based on our method can describe an evaluation procedure (interpreter) in defining the semantics of a programming language, while the conventional one describes a translation process.

FUTAMURA

Futamura projection





Code →
Result



Code →
Result



(Code,
Interpreter) →
Executable



Code →
Result



Interpreter →
Compiler



(Code,
Interpreter) →
Executable



Code →
Result



Interpreter →
Compiler



(Code,
Interpreter) →
Executable



(Interpreter →
Compiler) →
Supercompiler

Futamura projections

conjectured by Yoshihiko Futamura in the 1980's



Applying Futamura projections to compose Languages and Tools in GraalVM

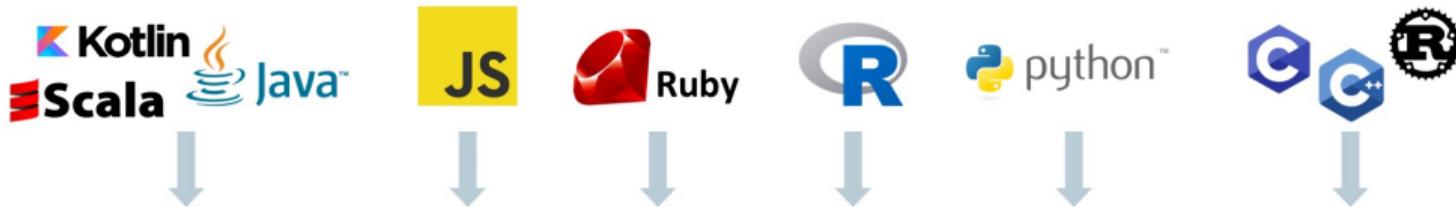
PEPM 2019

Christian Humer

VM Research Group, Oracle Labs, Zurich

GraalVM™





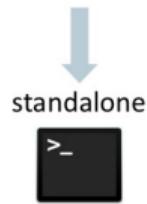
Automatic transformation of interpreters to compilers

ORACLE®

GraalVM

Enterprise Edition

Embeddable in native or managed applications



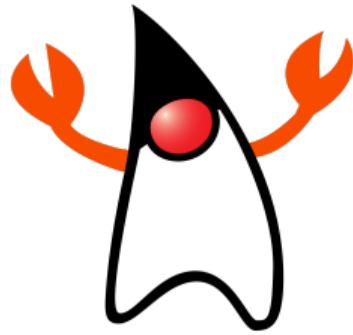
```
public abstract class JSMultiplyNode extends JSBinaryNode {  
  
    public abstract Object execute(Object a, Object b);  
  
    @Specialization(guards = "b > 0", rewriteOn = ArithmeticException.class)  
    protected int doIntBLargerZero(int a, int b) { /* ... */ }  
  
    @Specialization(rewriteOn = ArithmeticException.class)  
    protected int doInt(int a, int b) { /* ... */ }  
  
    @Specialization  
    protected double doDouble(double a, double b) {  
        return a * b;  
    }  
  
    // ...  
}
```

JVM + Polyglot = Profit

- languages may call each other
- languages benefit from the JVM:
 - ▶ parallelism
 - ▶ tooling
 - ▶ libraries
 - ▶ ...



Native code



Q & A



Lars Hupel

 lars.hupel@innoq.com

 @larsr_h

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim a. Rh.
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany

Ludwigstr. 180 E
63067 Offenbach
Germany

Kreuzstr. 16
80331 München
Germany

c/o WeWork
Hermannstrasse 13
20095 Hamburg
Germany

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 01 11

Albulastr. 55
8048 Zürich
Switzerland



LARS HUPEL

Consultant
innoQ Deutschland GmbH

Lars enjoys programming in a variety of languages, including Scala, Haskell, and Rust. He is known as a frequent conference speaker and one of the founders of the Typelevel initiative which is dedicated to providing principled, type-driven Scala libraries.

Image sources

- James Gosling: Peter Campbell, CC-BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?title=File:James_Gosling_2008.jpg&oldid=149207971
- Conde-Clemente, Patricia & Ortin, Francisco. (2014). JINDY: A java library to support invokedynamic.
- Ducks: <https://unsplash.com/photos/b7nK3aECcRI>
- Spices: <https://pixabay.com/photos/spices-spice-mix-market-73776/>
- GraalVM architecture: <https://blogs.oracle.com/graalvm/announcement>
- GraalVM slide: <https://popl19.sigplan.org/details/pepm-2019-papers/2/ Applying-Futamura-Projections-to-Compose-Languages-and-Tools-in-GraalVM-Invited-Talk>
- Rainbow: <https://pixabay.com/photos/rainbow-seaside-coast-beach-sky-675832/>
- Futurama logo: <https://de.wikipedia.org/w/index.php?title=Datei:Futurama-logo.svg&oldid=88835467>
- Da Vinci VM slide: <https://openjdk.java.net/projects/mlvm/pdf/LangNet20080128.pdf>
- LLVM logo: Apple
- Duke: Oracle
- Field: <https://unsplash.com/photos/4miBe6zg5r0>