Das eierlegende Truffleschwein

Lars Hupel
JUG Ingolstadt
2019-09-10

INNOQ

Java, eine kurze Geschichte

1994 erste Version der JVM

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

2004 erste Version von Scala

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

2004 erste Version von Scala

2006 Java 6 mit Scripting, Java Compiler API

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

2004 erste Version von Scala

2006 Java 6 mit Scripting, Java Compiler API

2011 Java 7 mit `invokedynamic`

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

2004 erste Version von Scala

2006 Java 6 mit Scripting, Java Compiler API

2011 Java 7 mit `invokedynamic`

2014 Java 8 mit Nashorn

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

2004 erste Version von Scala

2006 Java 6 mit Scripting, Java Compiler API

2011 Java 7 mit `invokedynamic`

2014 Java 8 mit Nashorn

2017 Java 9 mit jshell, AOT, jlink

Java, eine kurze Geschichte

1994 erste Version der JVM

2001 erste Versionen von Jython und JRuby

2003 erste Version von Groovy

2004 Java 5 mit Generics

2004 erste Version von Scala

2006 Java 6 mit Scripting, Java Compiler API

2011 Java 7 mit `invokedynamic`

2014 Java 8 mit Nashorn

2017 Java 9 mit jshell, AOT, jlink

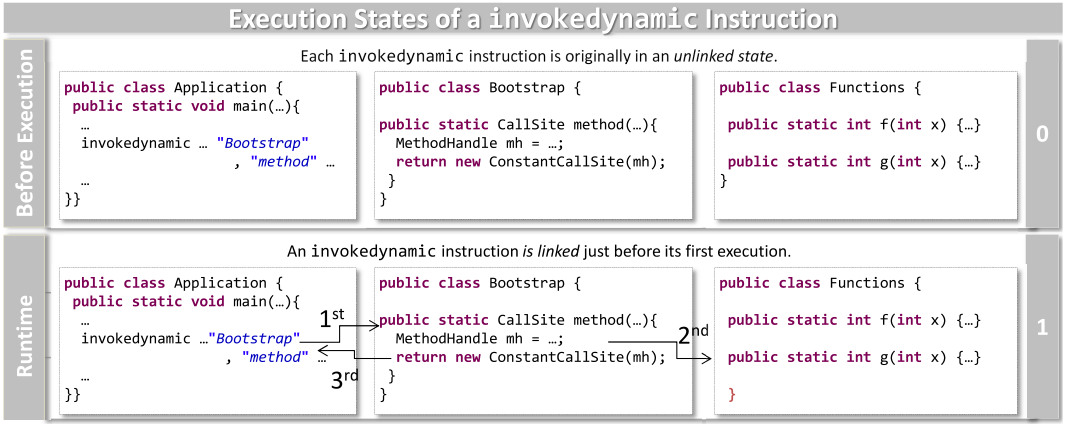
2019 erste Version von GraalVM

JSR 223

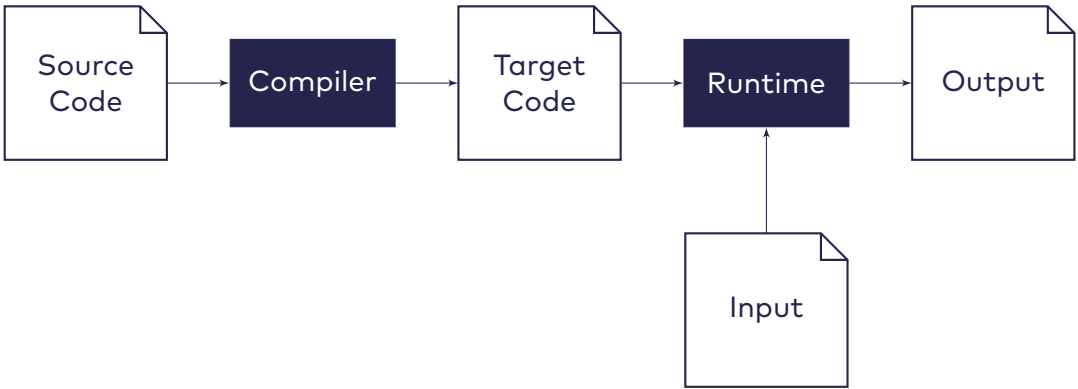
- AppleScript
- BeanShell
- Groovy
- Jaskell
- Java
- JavaFX Script
- JavaScript
  - Rhino
  - Nashorn
- JUEL
- PHP
- Python
- Ruby
- Scheme
- Smalltalk
- Tcl
- ...

JSR 292

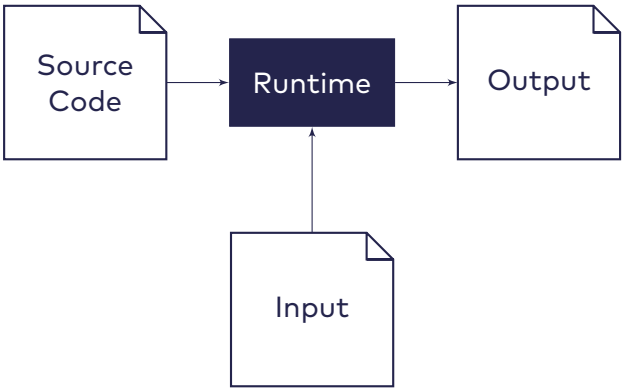
JSR 292


Execution States of a invokedynamic Instruction
Before Execution
Each invokedynamic instruction is originally in an unlinked state.
public class Application {
public static void main(…){
…
invokedynamic … "Bootstrap"
, "method" …
…
}}
public class Bootstrap {
public static CallSite method(…){
MethodHandle mh = …;
return new ConstantCallSite(mh);
}
}
public class Functions {
public static int f(int x) {…}
public static int g(int x) {…}
}
0
Runtime
An invokedynamic instruction is linked just before its first execution.
public class Application {
public static void main(…){
…
invokedynamic …"Bootstrap"
, "method" …
…
}}
1st
2nd
3rd
public class Bootstrap {
public static CallSite method(…){
MethodHandle mh = …;
return new ConstantCallSite(mh);
}
}
public class Functions {
public static int f(int x) {…}
public static int g(int x) {…}
}
1

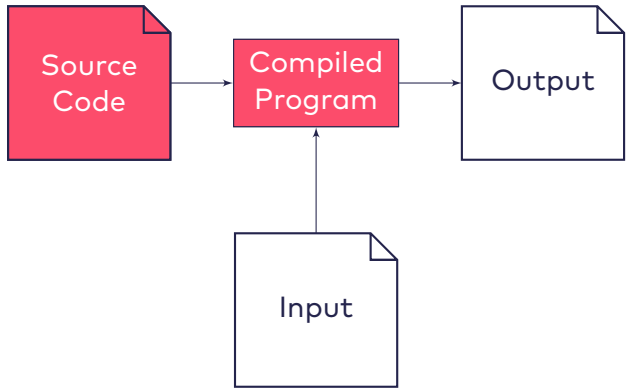
Compiler vs. Interpreter

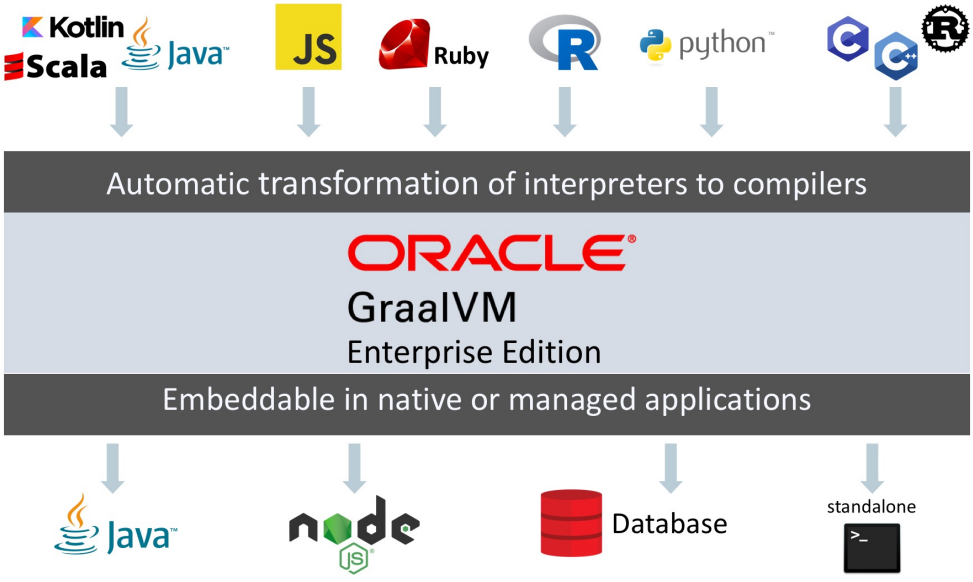


Compiler vs. Interpreter



Futamura-Projektion



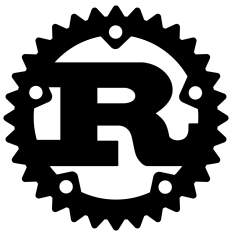

Kotlin
Scala
Java
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compilers
ORACLE
GraalVM
Enterprise Edition
Embeddable in native or managed applications
Java
node
JS
Database
standalone
>_

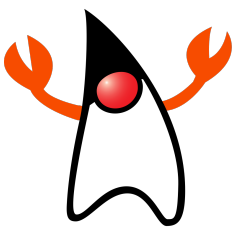
```
public abstract class JSMultiplyNode extends JSBinaryNode {

    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

- Sprachen können sich gegenseitig aufrufen
- Sprachen profitieren von der JVM:
  - Nebenläufigkeit
  - Tooling
  - Bibliotheken
  - ...

Nativer Code





Q & A

INNOQ
www.innoq.com

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.

Bildnachweise

- Chercheur de Truffes: `https://commons.wikimedia.org/wiki/File:Lot_chercheur_de_truffes.JPG` (gemeinfrei)
- Conde-Clemente, Patricia & Ortin, Francisco. (2014). JINDY: A java library to support invokedynamic.
- GraalVM architecture: `https://blogs.oracle.com/graalvm/announcement`
- Regenbogen: `https://pixabay.com/photos/rainbow-seaside-coast-beach-sky-675832/`
- LLVM-Logo: Apple
- Duke: Oracle