
# GraalVM

https://github.com/azatsatklichov/Java-Features

Azat Satklichov
azats@seznam.cz,
http://sahet.net/htm/java.html

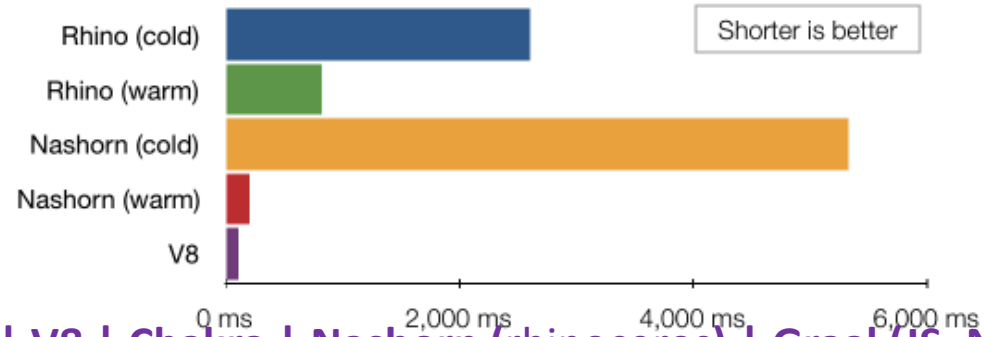
# New Features in JAVA 15

## Deprecations and Removals

- As a part of Java 15 removals/deprecations, Nashorn engine is removed (Nashorn supports ECMAScript 5.1 specification) …. Impacts/alternatives

- **Nashorn JS Engine (old one was Rhino engine)** – removed, also the tool 'jjs' is removed.

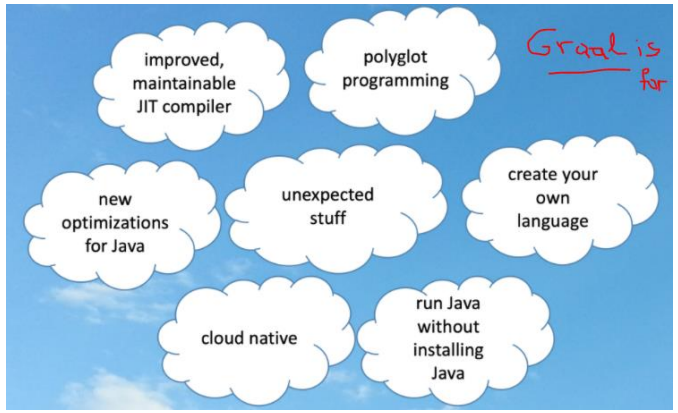
This JEP also removed the below two modules:
jdk.scripting.nashorn.shell – contains the jjs tool and
jdk.scripting.nashorn – contains jdk.nashorn.api.scripting and
jdk.nashorn.api.tree packages.




**NashornInsteadOfRhino.java ?** **Rhino | SpiderMonkey | V8 | Chakra | Nashorn** (rhinoceros) **| Graal (JS, NodeJS)**

**JDK has (will) no JS anymore (maintenance cost, also Graal offers more, and also JS hast testing-frameworks ).**
See Migration Guide

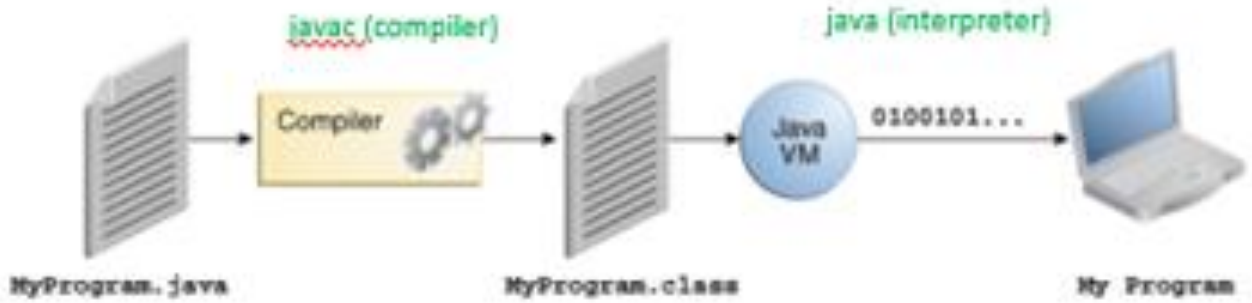
```
C:\Users\as892333>jjs
Warning: The jjs tool is planned to be removed from a future JDK release
jjs>
```



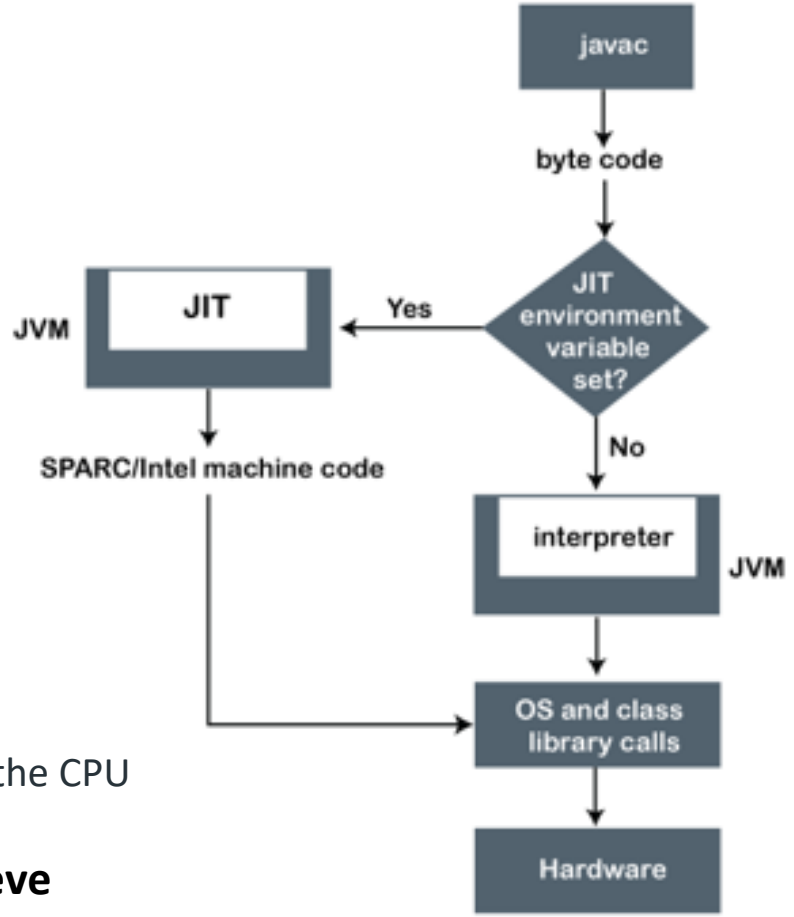

**Alternative** see: **GraalVM**, Project Detroit (V8) – bring V8 JS engine to OpenJDK

# Java Compiler, Interpreter, and JIT

Overview of the Java Software development process




JIT Compilation Process




src-programs are compiled (**javac,** regular Java compiler) ahead of time (**AOT**) and stored as machine independent code (bytecode, *.class files, platform independent), which is then linked at run-time and executed by an interpreter. Interpreting the bytecode, the standard implementation of the JVM slows the execution of the programs.

JIT compilers (not regular compiler) interact with JVM at runtime to improve performance and compile appropriate bytecode (JVM instruction set) sequences into native machine code that the CPU executes it directly. Default strategy used by the HotSpot during normal program execution, called **tiered compilation (C1 & C2) . It is a mix of C1 and C2 compilers in order to achieve both fast startup and good long-term performance**.

C1 (client), C2 (server compiler) JIT compilers. Not improved lately, hard-to-maintain,.. written **in C++**.

Also Java 9 has AOT - https://openjdk.java.net/jeps/295

The JIT is **enabled by default**. To enable Explicitly: set JAVA_COMPILER=jitc or java -Djava.compiler=jitc <class>
To disable the JIT: Set theset **JAVA_COMPILER=NONE**, Or java **-Djava.compiler=NONE** <class>

# New Feature in JAVA 15


GraalVM is an umbrella term, consists of: **Graal** the JIT compiler, and **Truffle** (lang. runtimes), **Substrate VM** (Native Image).

| Linux AMD64 | Linux ARM64 | macOS | Windows |
|---|---|---|---|

**GraalVM Enterprise** (based on **Oracle JDK**) and **GraalVM Community** (on **OpenJDK**) editions, supports **Java 8, 11, 16. Releases:** 1-release GraalVM 19.0 is based on top of JDK version 8u212, latest-21.2.0.1

**History:** GraalVM has its roots in the Maxine Virtual Machine project at Sun Microsystems Laboratories (now Oracle Labs). The **goal was** removing dependency to C++ & its problems … & written in modular, maintainable and extendable fashion **in Java itself**. Moreover you can deeply understand, learn it looking the code as well.

**Project goals (improve performance, reduce startup time, native images, extending JVM app. Or native app. Own lang.)**

- To improve the performance of Java virtual machine-based languages to match the performance of native languages.
- To reduce the startup time of JVM-based apps by compiling them AOT with GraalVM Native Image technology.
- To enable GraalVM integration into the OracleDB, OpenJDK, Node.js, Android/iOS, and to support similar custom embeddings.
- To allow freeform mixing of code from any programming language in a single program, billed as "polyglot applications".
- Create your own language. To include an easily extended set of "polyglot programming tools".

**E.g: Twitter use Graal to run Scala app., Facebook uses it to accelerate its Spark workloads, …**

# GraalVM Architecture

## Core Components

**Runtimes:** Java HotSpot VM, Javascript runtime, LLVM runtime.

**Runtime Modes:** JVM runtime mode, Native Image, Java on Truffle (on AST)

**Libraries (JAR files):**

- Graal compiler
- *Polyglot API*

**Utilities:** JS REPL & JS interpreter, **lli** tool to run LLVM app., GraalVM Updater

**Truffle Language Impl. framework and the GraalVM SDK**

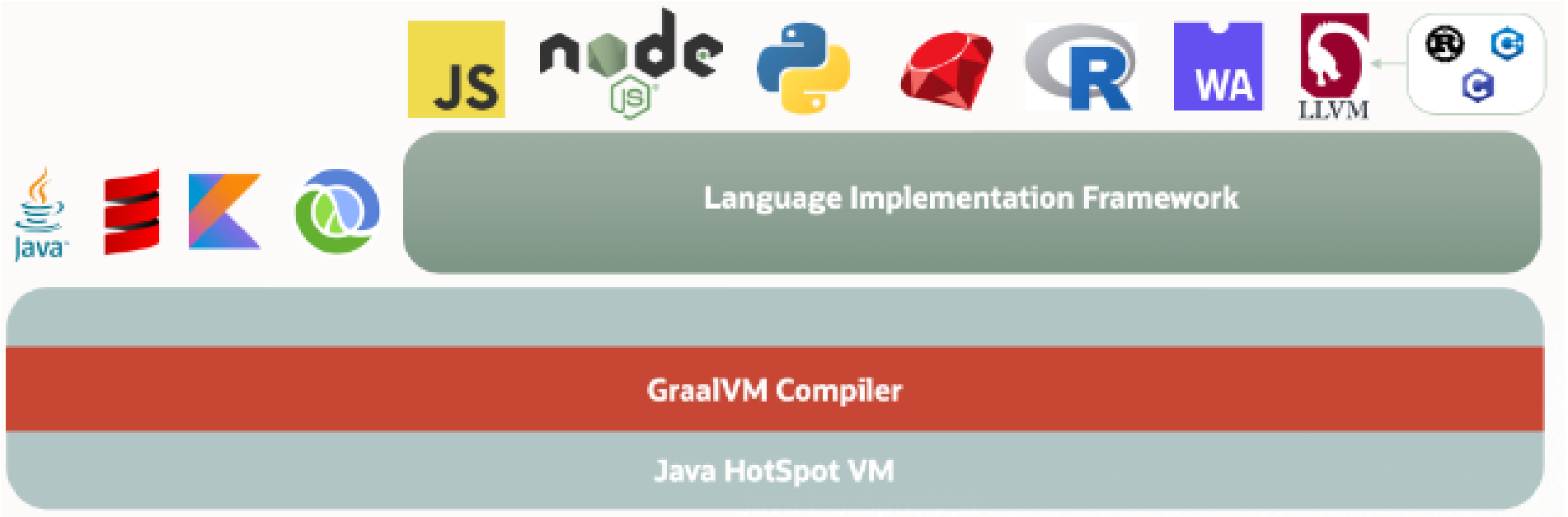
## Additional Components

core installation can be extended with:

**Tools/Utilities:**

- *Native Image*
- *LLVM toolchain*
- *Java on Truffle*

**Runtimes: Node.js, Python, Ruby, R, GraalWasm(Web Assembly), Combined Languages, ..**

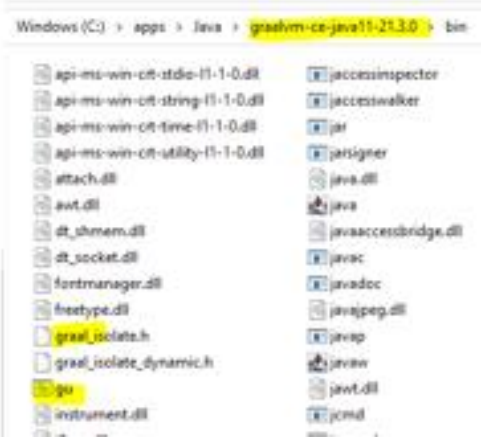



**See: Repository Structure, APIs**

GraalVM Native Image is officially supported by the Fn, Gluon, Helidon, Micronaut, Picocli, Quarkus, Vert.x and Spring Boot Java frameworks

# Language and Runtime Support

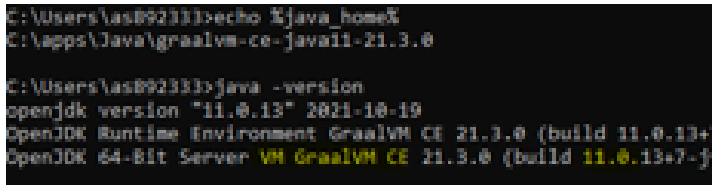
**See: Download, APIs**




GraalVM's /bin directory, as like standard JDK, with additional launchers and utilities:

- **js** a JavaScript launcher
- **lli** a LLVM bitcode launcher
- **gu** the GraalVM Updater tool to install additional language runtimes and utilities

**>gu list** - to see installed launchers/utils




Besides JVM langs. Java, Scala, Kotlin, … additional languages can be supported in GraalVM based on Truffle Language Implementation framework: GraalVM JavaScript, TruffleRuby:, FastR, GraalVM Python, GraalVM LLVM Runtime , & GraalWasm

Run **Java** as it is:
>javac…. Then how to add other runtimes? Node.js, Ruby, R, WebAssembly?

# Graal Compiler

**Graal is a high-performance JIT compiler. See repo**

- It accepts the JVM bytecode and produces the machine code.
- Uses JVMCI to communicate with the VM. **JVM Compiler Interface**: JVMCI excludes the standard tiered compilation and plug in our brand new compiler (i.e. Graal) without the need of changing anything in the JVM

- Graal compiler is alternative to C2. Unlike C2, it can run in both **JIT** and **AOT** compilation **modes** to produce native code

**Advantages of writing a compiler in Java?**
Safety, no crash but ex., no mem.leak, tool support (debuggers, profilers). ..

To enable the use of the new JIT compiler: -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler

**We can run a simple program in three different ways**: via regular C1/C2, with the JVMCI version of Graal on Java 10, or with the GraalVM

High-performance modern Java: **E.g. Demo1, Demo2.java**
Low-footprint, fast-startup Java: Currently suffer from longer startup time and high memory usage for short-running process

Tooling support: GraalVM includes VisualVM with the standard jvisualvm command.
>jvisualvm &> /dev/null &

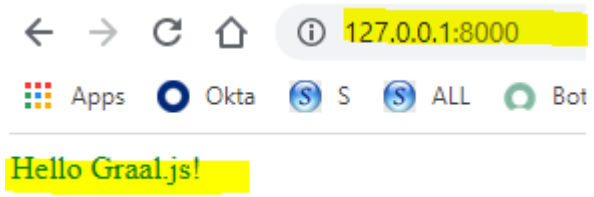
Extend a JVM-based application: A new **org.graalvm.polyglot** API lets you load and run code in other languages

# Language and Runtime Support

GraalVM JS Implementation provides an ECMAScript-compliant (See Kangax table) runtime to execute JS and Node.js apps. To migrate the code previously targeted to the **Nashorn** or **Rhino** engines, see migration guides.

**Why GraalVM JavaScript than Nashorn?** Higher performance, better tooling, and interoperability (polyglot) **It can execute Node.js apps, .. ibs..tools**

The languages in GraalVM aim to be drop-in replacements for your existing languages. E.g. we can install a Node.js module




**Run JS & Node.js. Debugging (Tooling support)**

>C:\apps\Java\graalvm-ce-java11-21.3.0\bin>**js**

**>1+3** **or DEBUG:>** js --inspect fiz.js (--inspect)

>**gu** install **nodejs**

>node -v

v14.17.6

**100,000 libs** < npm packages are compatible with GraalVM, e.g. express, react, async, mocha, etc.

//run Node JS app with domains_list.txt

>node validate-domain-async1.js

>npm install colors ansispan express >npm list

>node z-node-app.js →open: http://127.0.0.1:8000/

>graalpython --inspect fizzbuzz.py

>ruby --inspect fizzbuzz.rb

**Run Python, R, LLVM Langs (C/C++, Rust, .. ), ..**

- **gu install R**

- gu install **python**
- graalpython

**>1+3**

> gu install **llvm-toolchain** >gu install native-image

# Language and Runtime Support

WebAssembly (Wasm) is an universal low level bytecode that runs on the web with near-native performance. It is a compilation target for languages like Rust, AssemblyScript (Typescript-like), Emscripten (C/C++), C# and much more!

## Run WebAssembly

**>gu install wasm**

>emcc -o floyd.wasm floyd.c

```
> wasm --Builtins=wasi_snapshot_preview1 floyd.wasm
1 .
2 3 .
4 5 6 .
7 8 9 10 .
11 12 13 14 15 .
16 17 18 19 20 21 .
22 23 24 25 26 27 28 .
29 30 31 32 33 34 35 36 .
37 38 39 40 41 42 43 44 45 .
46 47 48 49 50 51 52 53 54 55 .
```

## Creating HTML and JavaScript

//compile by Emscripten compiler, steps to perform

**> emcc hello.c -s WASM=1 -o hello.html**

//'emcc' is not recognized as an internal or external command

//open html in running server

## Run WebAssembly in Java

```
> WebAssemplyByJava.java
```

# Polyglot API - Combine Languages

GraalVM allows you to call one programming language into another and exchange data between them. To enable interoperability, GraalVM provides the --polyglot flag

## Run Combine Languages – Polyglot (Embedding Languages)

The GraalVM Polyglot API lets you embed and run code from guest languages in JVM-based host applications.
Allow interoperability with Java code the *--jvm* **flag**

**JAVA[JS, Ruby, Python, LLVM…]: >**Polyglot.java
**JS[Java, Python, Ruby, LLVM]: >**js --polyglot --jvm polyglot.js

**C[…]: >** gu install **llvm-toolchain**

**Python[…]: >** graalpython --polyglot --jvm polyglot.py

## Passing options:

You can configure a language engine for better throughput or startup
- Through launchers : js, python, llvm, r, ruby
- Programmatically
- Using JVM arguments

# Native Images

Native Image is a technology to AOT Java code to a standalone executable, called a native image. Native Image supports JVM-based languages, e.g., Java, Scala, Clojure, Kotlin. The resulting image can, optionally, execute dynamic languages like JavaScript, Ruby, R or Python . GraalVM Native Image is officially supported by the Fn, Gluon, Helidon, Micronaut, Picocli, Quarkus, Vert.x and Spring Boot Java frameworks. The **jaotc** command creates a Native Image. The experimental -XX:+EnableJVMCIProduct flag enables the use of Graal JIT

**Native Images** (read more)

> gu install native-image
>javac NativeImage.java
>java NativeImage
>native-image NativeImage


# Truffle Language Implementation Framework

In association with GraalVM, Oracle Labs developed a language abstract syntax tree interpreter called "Truffle" which would allow it to implement languages (or language-agnostic tools like debuggers, profilers, and other instrumentations) on top of the GraalVM. The Truffle framework and its dependent part, GraalVM SDK, are released under the Universal Permissive License

Truffle is a Java library that helps you to write an *abstract syntax tree* (AST) interpreter for a language. An AST interpreter is probably the simplest way to implement a language, because it works directly on the output of the parser and doesn't involve any bytecode or conventional compiler techniques, but it is often slow.

```
<dependency>
    <groupId>org.graalvm.truffle</groupId>
    <artifactId>truffle-api</artifactId>
    <version>21.3.0</version>
</dependency>
<dependency>
    <groupId>org.graalvm.truffle</groupId>
    <artifactId>truffle-dsl-processor</artifactId>
    <version>21.3.0</version>
    <scope>provided</scope>
</dependency>
```

## Instrumentation-based Tool Support

The core GraalVM installation provides a language-agnostic debugger, profiler, heap viewer, and others based on instrumentation and other VM support.


# THANK YOU

**References**

https://github.com/oracle/grail, https://www.graalvm.org/docs/introduction/#graalvm-architecture
https://www.graalvm.org/reference-manual/ https://medium.com/graalvm/graalvm-in-2018-b5fa7ff3b917
https://www.devonblog.com/software-development/graalvm-a-polyglot-and-faster-virtual-machine-getting-started/
https://www.baeldung.com/graal-java-jit-compiler https://www.graalvm.org/reference-manual/js/FAQ/
https://www.baeldung.com/jvm-tiered-compilation https://www.beyondjava.net/category/graalvm
https://www.graalvm.org/docs/getting-started/ https://www.devonblog.com/software-development/graalvm-a-polyglot-and-faster-virtual-machine-getting-started/