ORACLE

JAVAFORUM stuttgart September 2021

# GraalVM 21.2 Feature Update der universellen VM

## Merkmale und Einsatzgebiete der GraalVM

**Wolfgang Weigend**
Master Principal Solution Engineer | global Java Team
Java Technology & GraalVM and Architecture

## Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

## GraalVM  Native Image early adopter status

GraalVM Native Image technology (including SubstrateVM) is Early Adopter technology.  It is available only under an early adopter license and remains subject to potentially significant further changes, compatibility testing and certification.




# Agenda

- GraalVM in the Java SE Subscription
- GraalVM Enterprise Intro
- GraalVM Just-in-Time Compiler
- GraalVM Polyglot support for multiple languages
- GraalVM Enterprise Native Image
- GraalVM Enterprise Components
- GraalVM Release Notes
- GraalVM Espresso – Java on Truffle
- Summary




# GraalVM Enterprise with Java SE Subscription

- Oracle Java SE Subscription now entitles customers to use Oracle GraalVM Enterprise at no additional cost
- Key benefits for Java SE Subscribers:
  - Native Image utility to compile Java to native executables that start almost instantly for containerized workloads
  - High-performance Java runtime with optimizing compiler that can improve application performance

ORACLE

# GraalVM Enterprise

# GraalVM Enterprise

High-performance runtime that provides significant improvements in application performance and efficiency


**High-performance optimizing Just-in-Time (JIT) compiler**


**Multi-language support for the JVM**

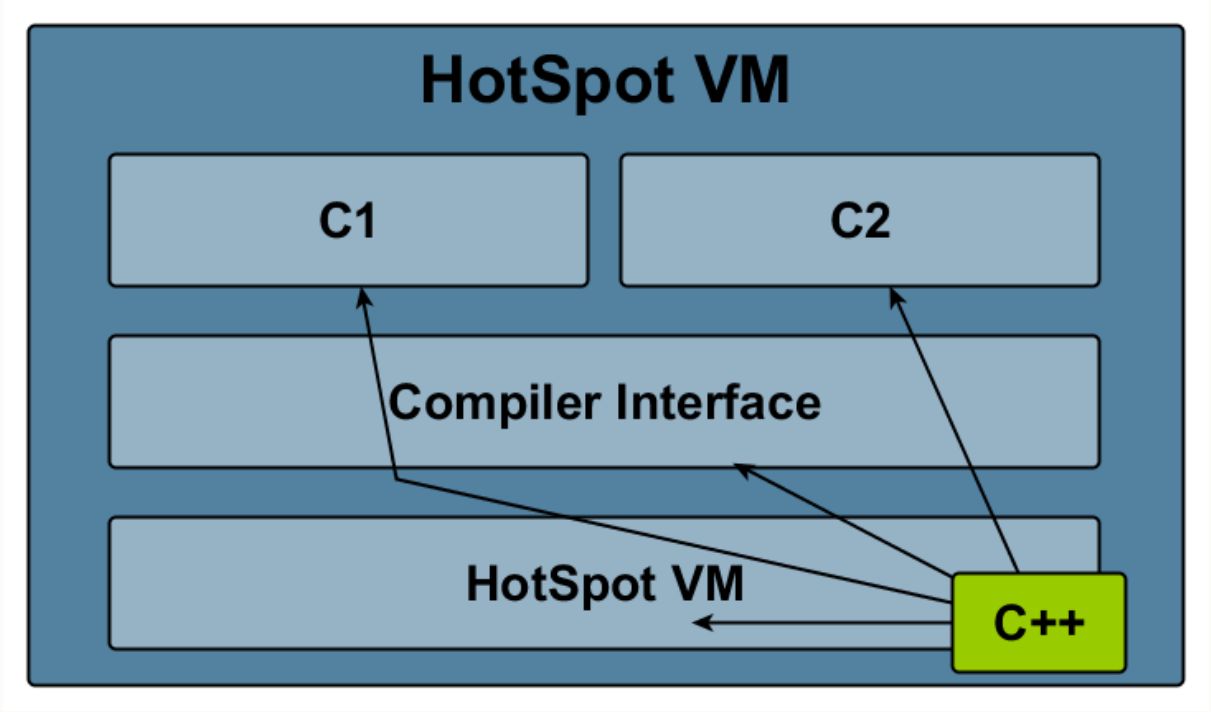

**Ahead-of-Time (AOT) “native image” compiler**

# GraalVM JIT Compiler working

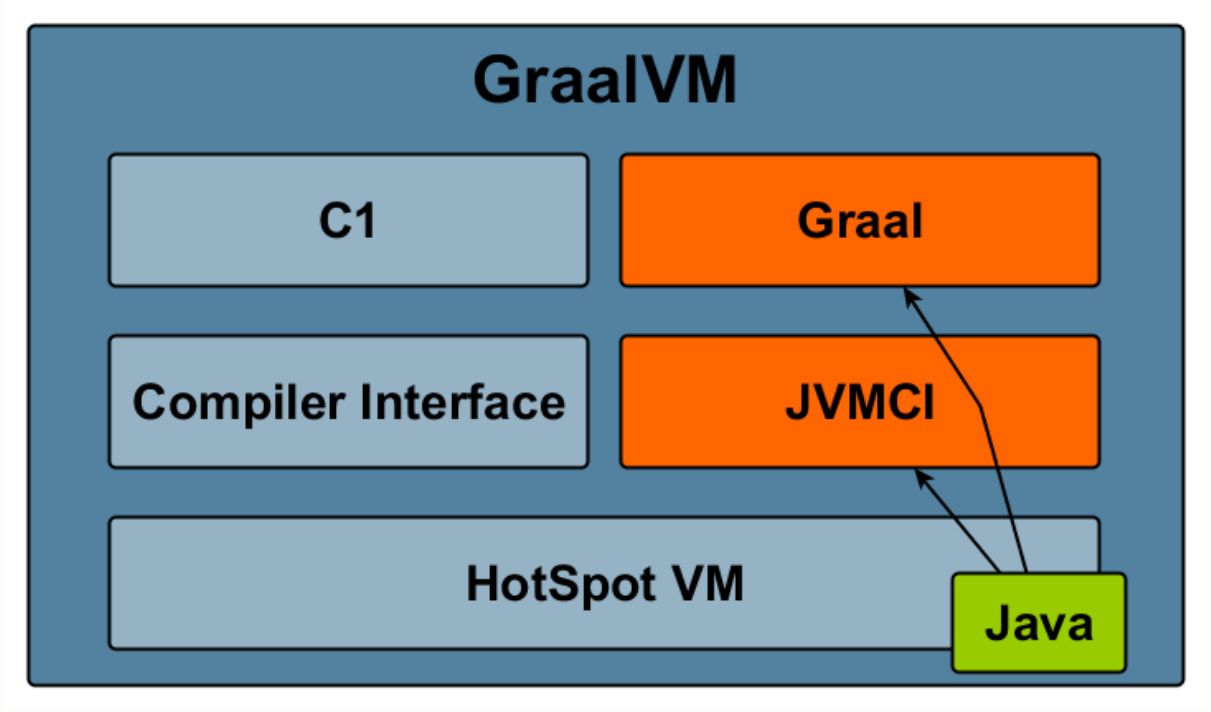
- **Inlining**
  - Code der aufzurufenden Methode/Funktion anstelle des Aufrufs
- **On-Stack Replacement**
  - Loop-Compilation, ohne auf den Methodenaufruf zu warten
- **Escape Analysis**
  - Automatische Stack-Allokation, ohne GC
- **De-Optimierung**
  - Optimierung rückgängig machen




# JIT Compiler written in C++



# JIT Compiler written in Java

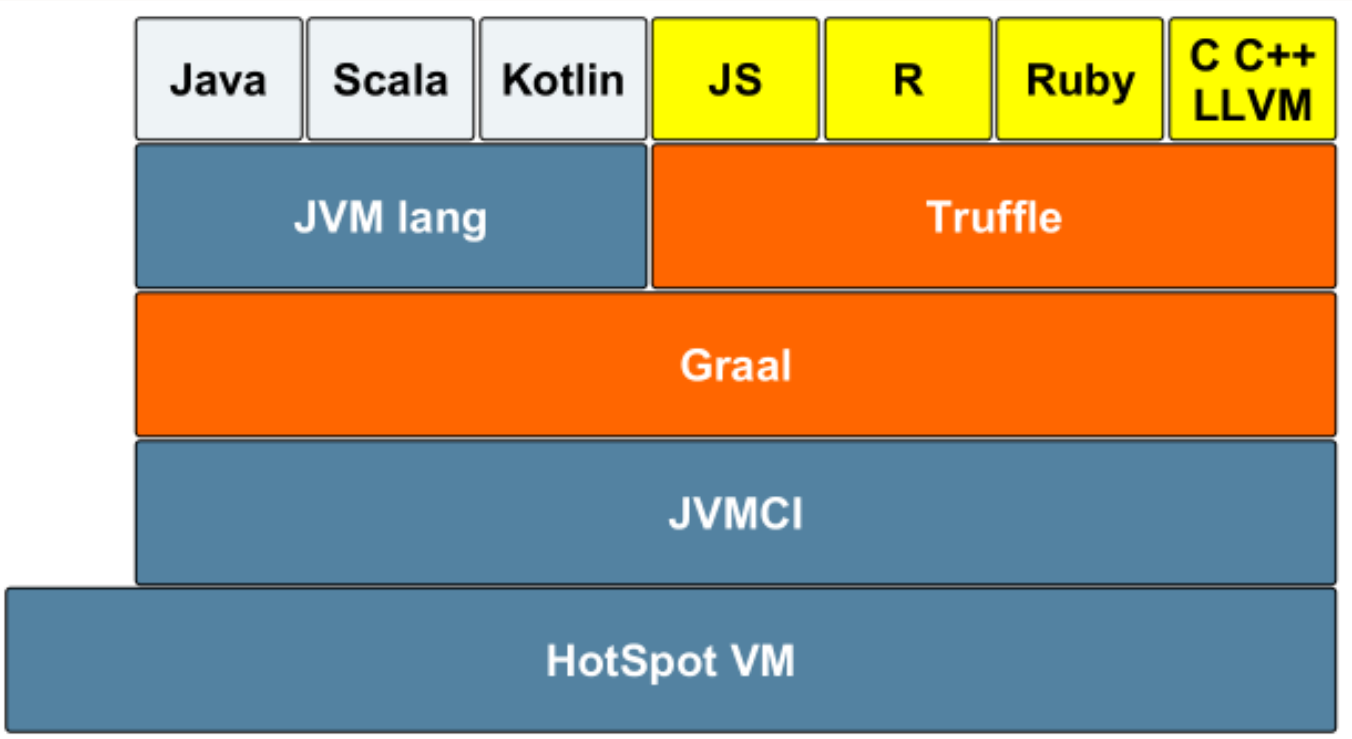


# GraalVM

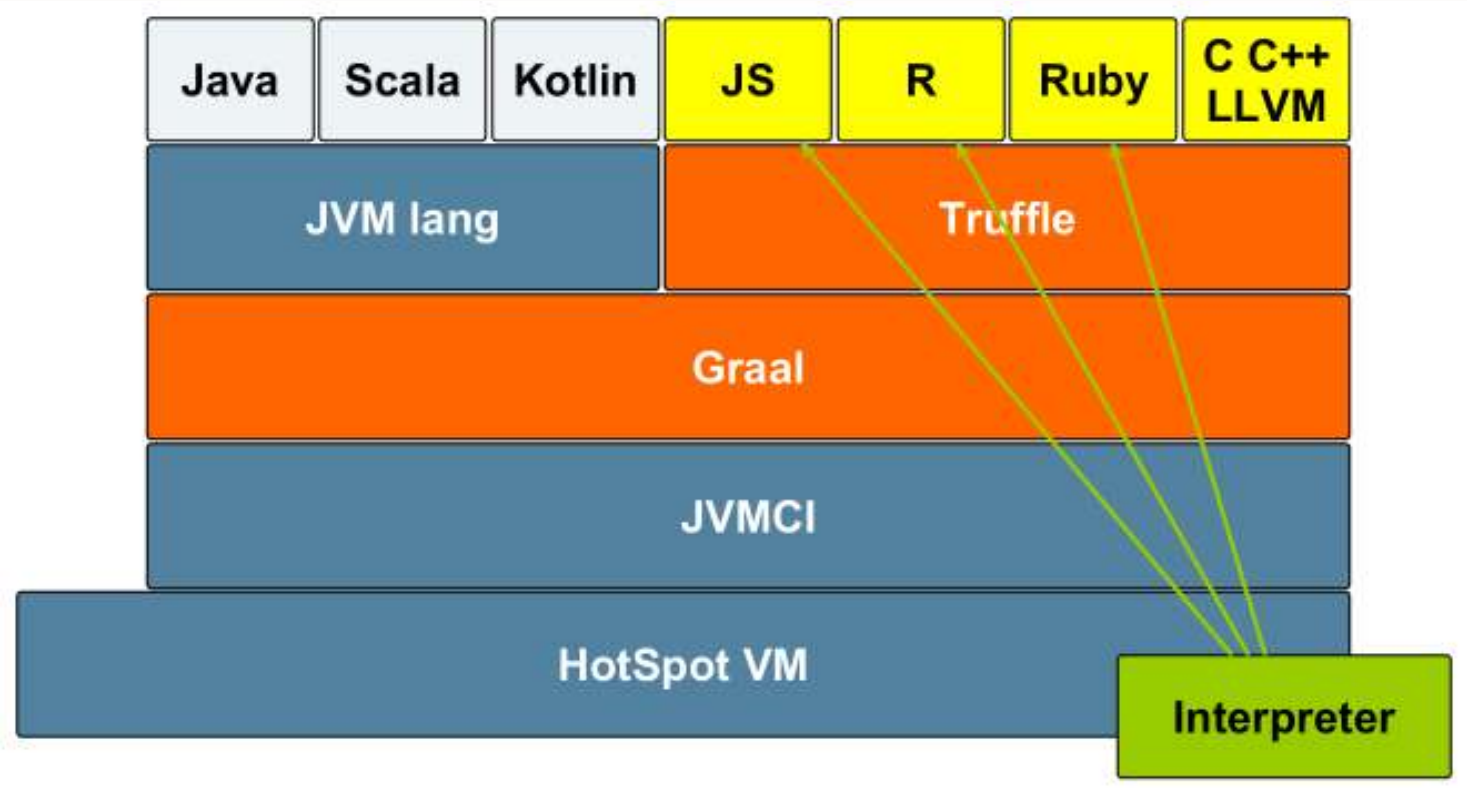
- **Graal**
  - **JIT Compiler**
    - Graal in GraalVM - A new Java JIT Compiler
  - **Graal integrated via Java Virtual Machine Compiler Interface (JVM CI)**
  - **Use a JDK with Graal (jdk.internal.vm.compiler)**
- **Truffle**
  - **Language Implementation Framework**
- **Substrate VM**
  - **Runtime Library and a set of tools for building Java AOT compiled code**




# GraalVM - Polyglot (1)

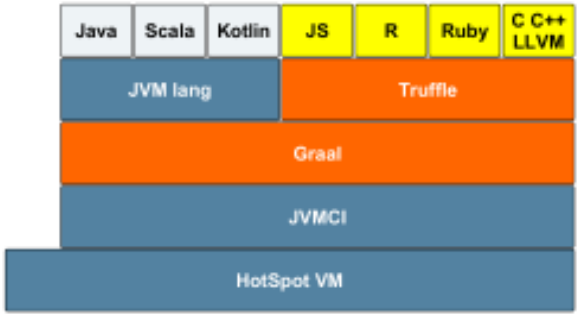


# GraalVM - Polyglot (2)



## GraalVM - Language Usability

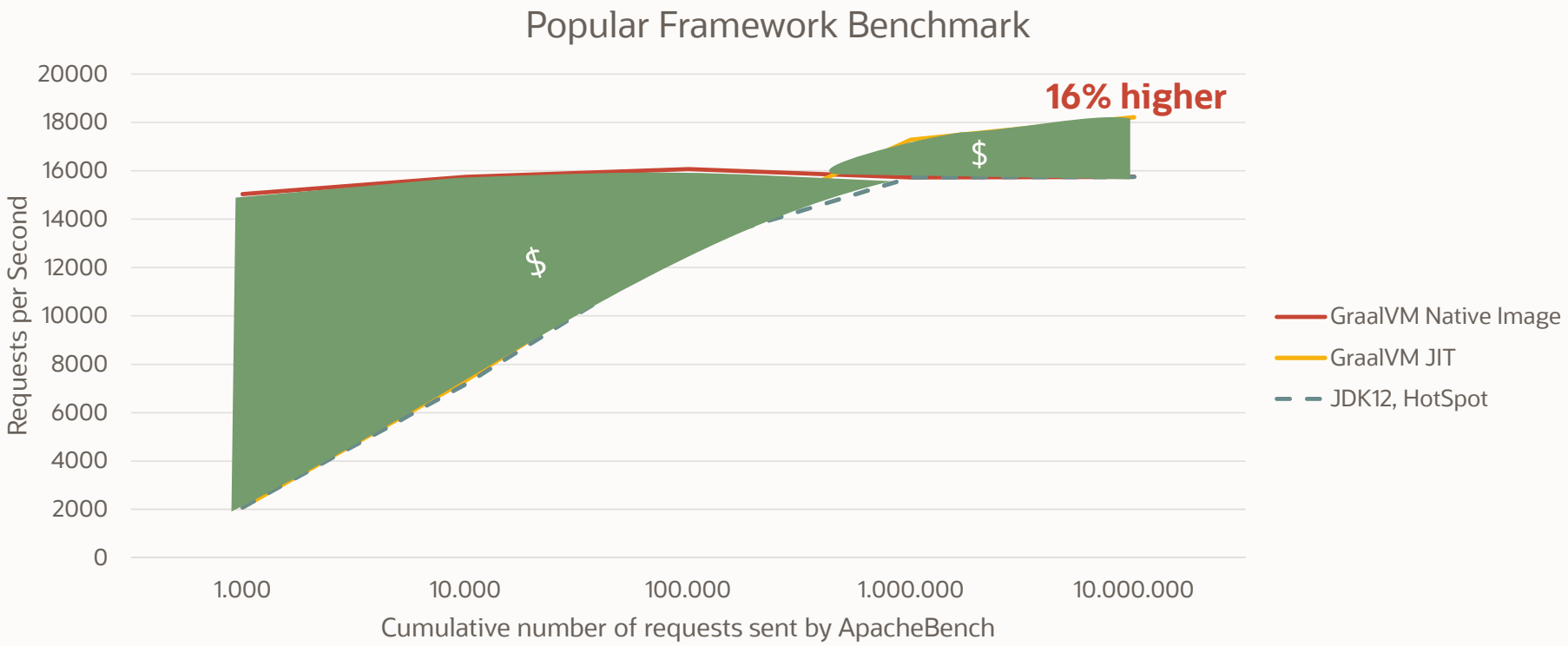
| Production-Ready | Experimental | Visionary |
|---|---|---|
| Java | Ruby | Python |
| Scala, Groovy, Kotlin | R | VSCode Plugin |
| JavaScript | LLVM Tool Chain | GPU Integration |
| Node.js | | WebAssembly |
| Native Image | | LLVM Backend |
| VisualVM | | |



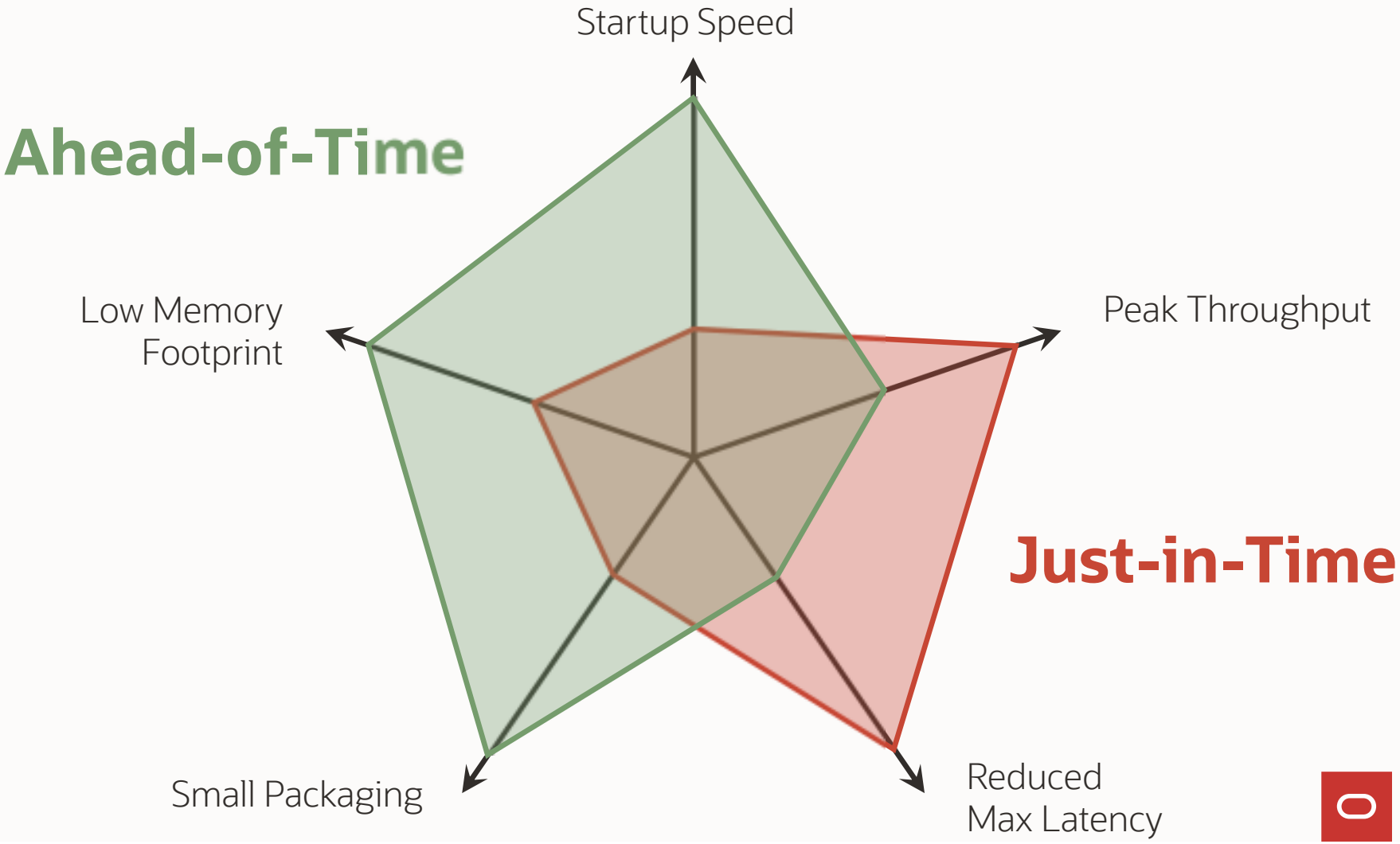
# GraalVM - Language Usability for the Platform

| Feature | Linux AMD64 | Linux ARM64 | MacOS | Windows |
|---|---|---|---|---|
| Native Image | supported | experimental | supported | supported |
| LLVM Runtime | supported | experimental | supported | not available |
| LLVM Toolchain | supported | experimental | supported | not available |
| JavaScript | supported | experimental | supported | supported |
| Node.js | supported | experimental | supported | supported |
| Java on Truffle | experimental | not available | experimental | experimental |
| Python | experimental | not available | experimental | not available |
| Ruby | experimental | not available | experimental | not available |
| R | experimental | not available | experimental | not available |
| WebAssembly | experimental | experimental | experimental | experimental |

# GraalVM Enterprise throughput

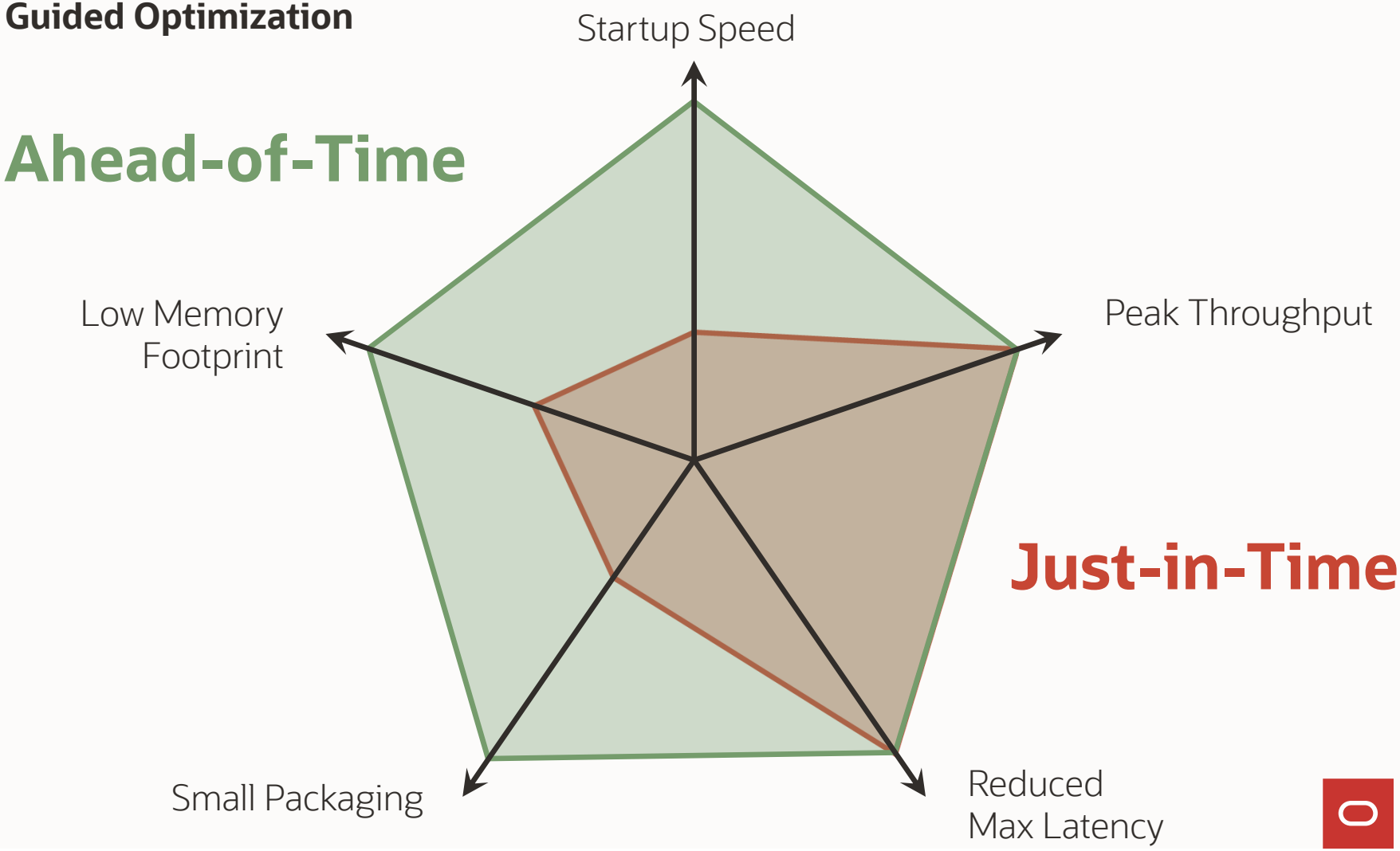


# GraalVM Enterprise compilation performance characteristics



# GraalVM Enterprise compilation performance characteristics

**Profile Guided Optimization**



ORACLE

# GraalVM Native Image

# GraalVM Enterprise Native Image—Ahead-of-time compiler & runtime

Microservices and Containers


**Up to 5x less memory**
**100x faster startup**

# Closed World Assumption

- **The points-to analysis needs to see all bytecode**
  - Otherwise aggressive AOT optimizations are not possible
  - Otherwise unused classes, methods, and fields cannot be removed
  - Otherwise a class loader / bytecode interpreter is necessary at run time
- **Dynamic parts of Java require configuration at build time**
  - Reflection, JNI, Proxy, resources, …
- **No loading of new classes at run time**



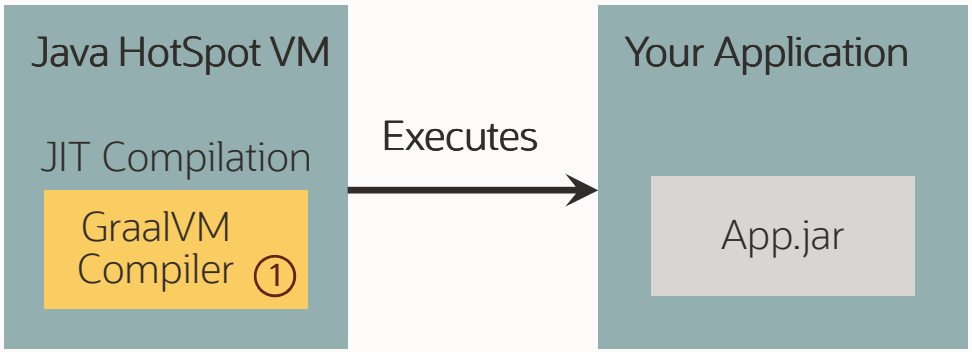

# Image Heap

- **Execution at run time starts with an initial heap: the “image heap”**
  - Objects are allocated in the Java VM that runs the image generator
  - Heap snapshotting gathers all objects that are reachable at run time
- **Do things once at build time instead at every application startup**
  - Class initializers, initializers for static and static final fields
  - Explicit code that is part of a so-called “Feature”
- **Examples for objects in the image heap**
  - java.lang.Class objects, Enum constants




# One Compiler, Many Configurations

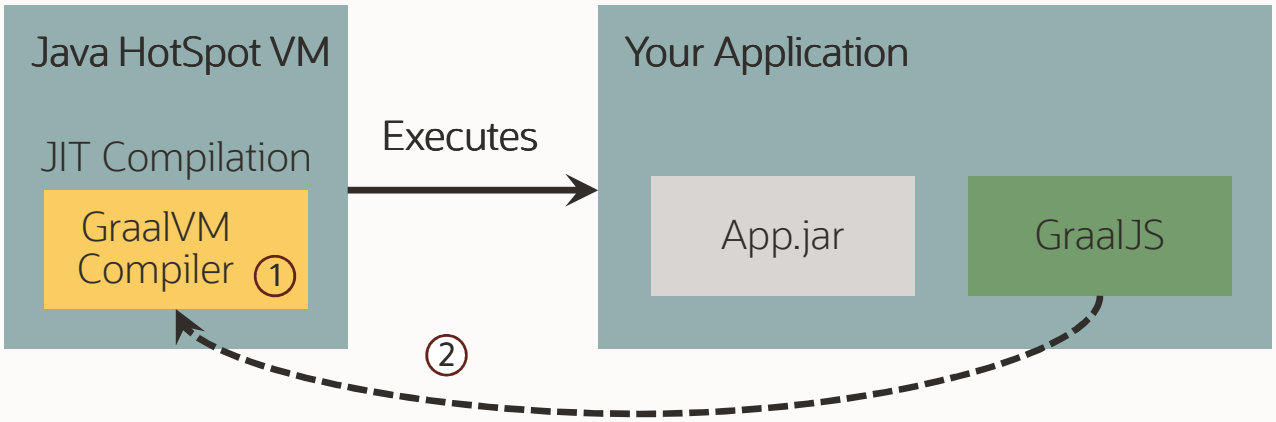
GraalVM




① Compiler configured for just-in-time compilation inside the Java HotSpot VM

# One Compiler, Many Configurations

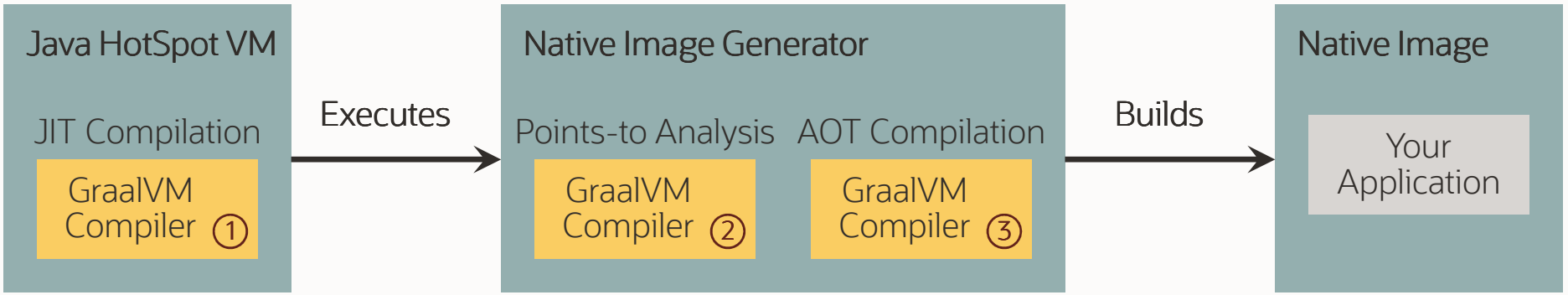
GraalVM




1. Compiler configured for just-in-time compilation inside the Java HotSpot VM
2. Compiler also used for just-in-time compilation of JavaScript code

# One Compiler, Many Configurations

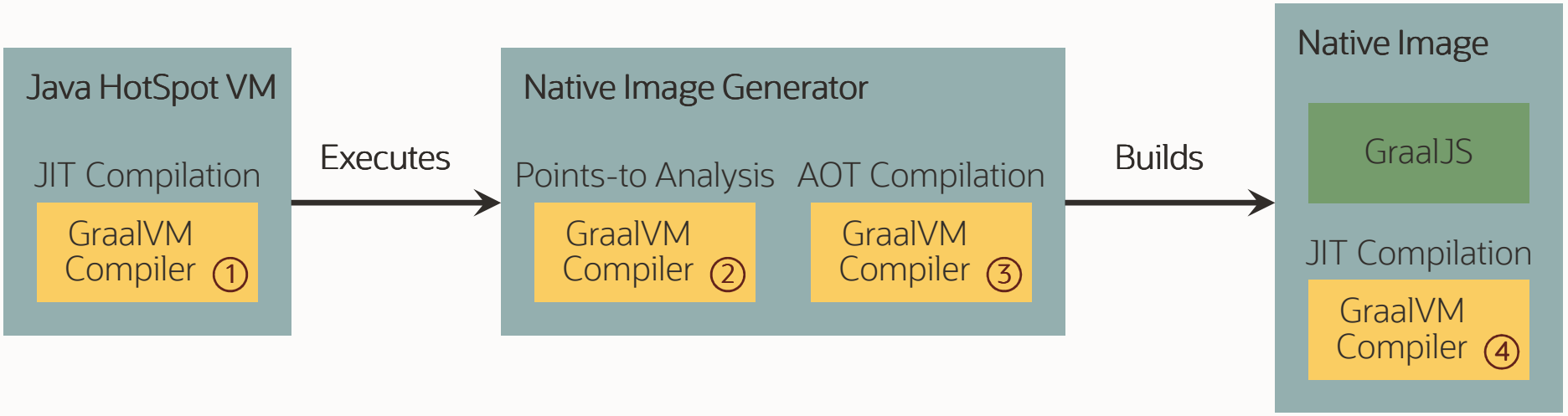
GraalVM




① Compiler configured for just-in-time compilation inside the Java HotSpot VM
② Compiler configured for static points-to analysis
③ Compiler configured for ahead-of-time compilation

# One Compiler, Many Configurations

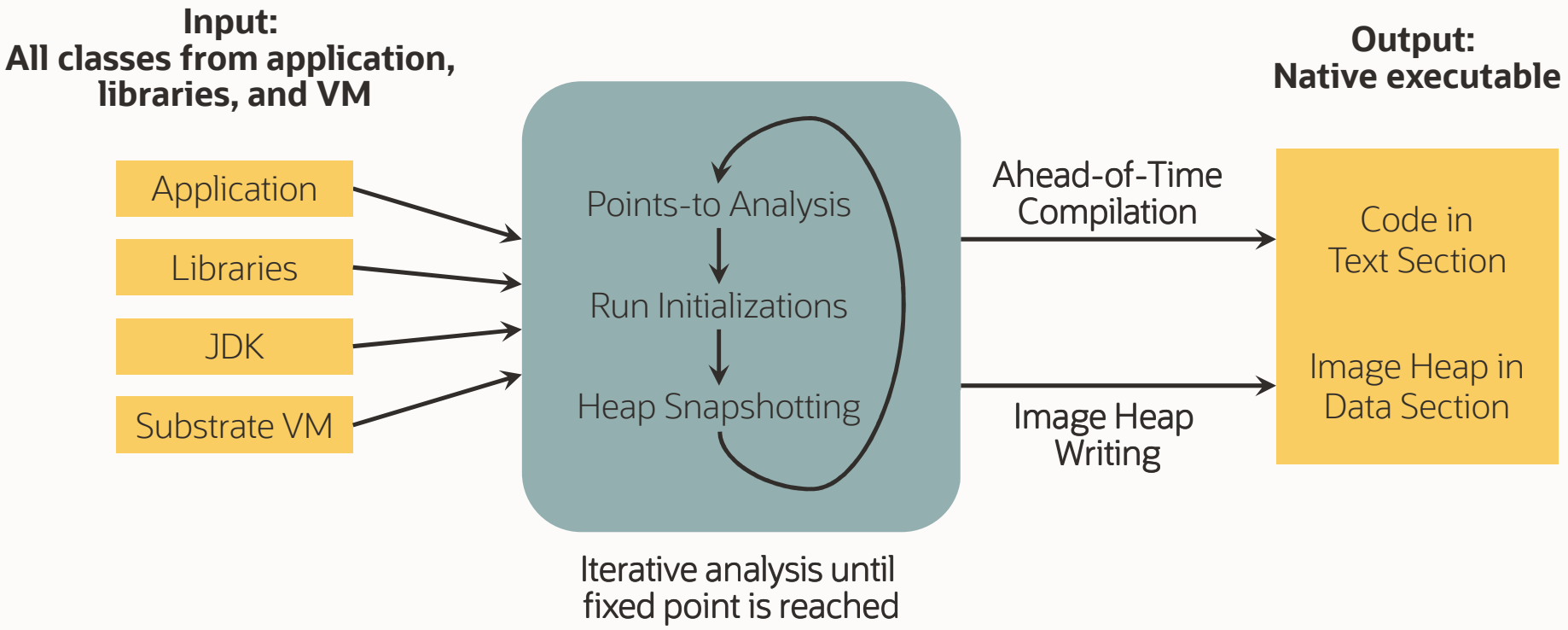
GraalVM




① Compiler configured for just-in-time compilation inside the Java HotSpot VM
② Compiler configured for static points-to analysis
③ Compiler configured for ahead-of-time compilation
④ Compiler configured for just-in-time compilation inside a Native Image

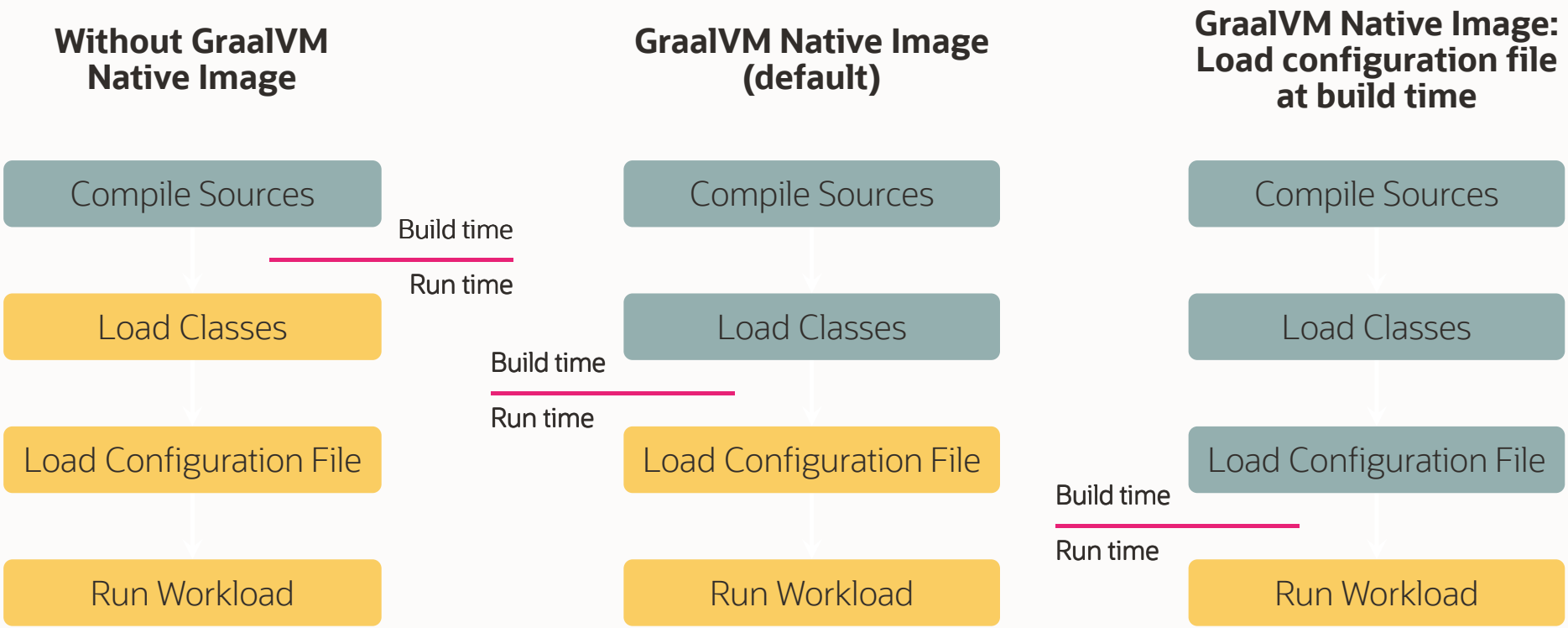
# Native Image - Details

GraalVM



# Benefits of the Image Heap

GraalVM

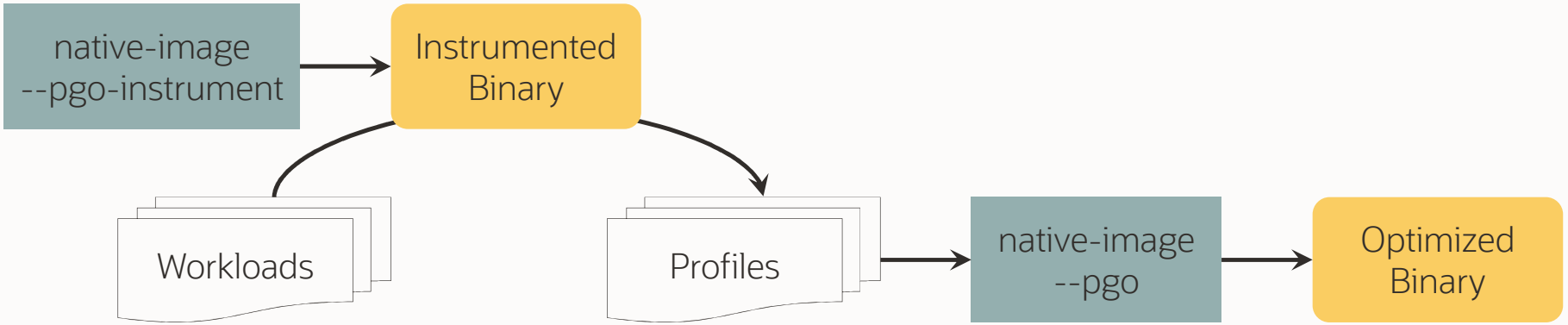


# Profile-Guided Optimizations (PGO)

GraalVM

## Out of Band Optimization

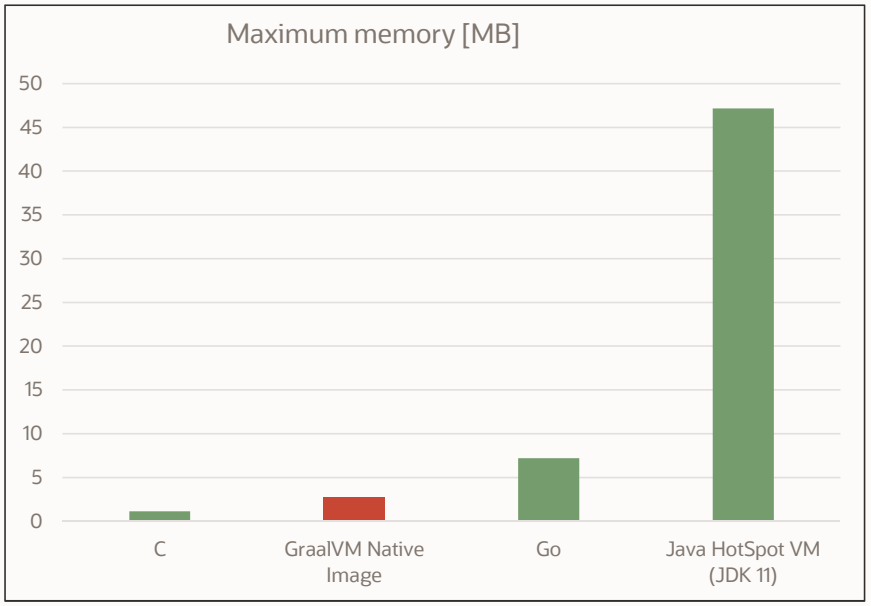
- AOT compiled code cannot optimize itself at run time (no “hot spot” compilation)
- PGO requires representative workloads
- Optimized code runs immediately at startup, no “warmup” curve



# GraalVM Enterprise Native Image

Lower cloud costs for containerized workloads, and microservices


Competitive startup time

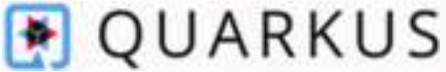



Significantly reduced memory requirements

# GraalVM Enterprise Native Image

## Supported by leading frameworks







# GraalVM Enterprise Native Image - Spring Native

Which version of Spring Boot is certified or official supported with GraalVM EE 21 native image?

- We don't currently offer certification of Spring Boot but we are discussing it.
- But Spring will declare Spring Native 1.0, which is essentially "certification" for Native Image.

When do we expect the declaration of Spring Native 1.0, which is the essentially "certification" for the GraalVM native image?

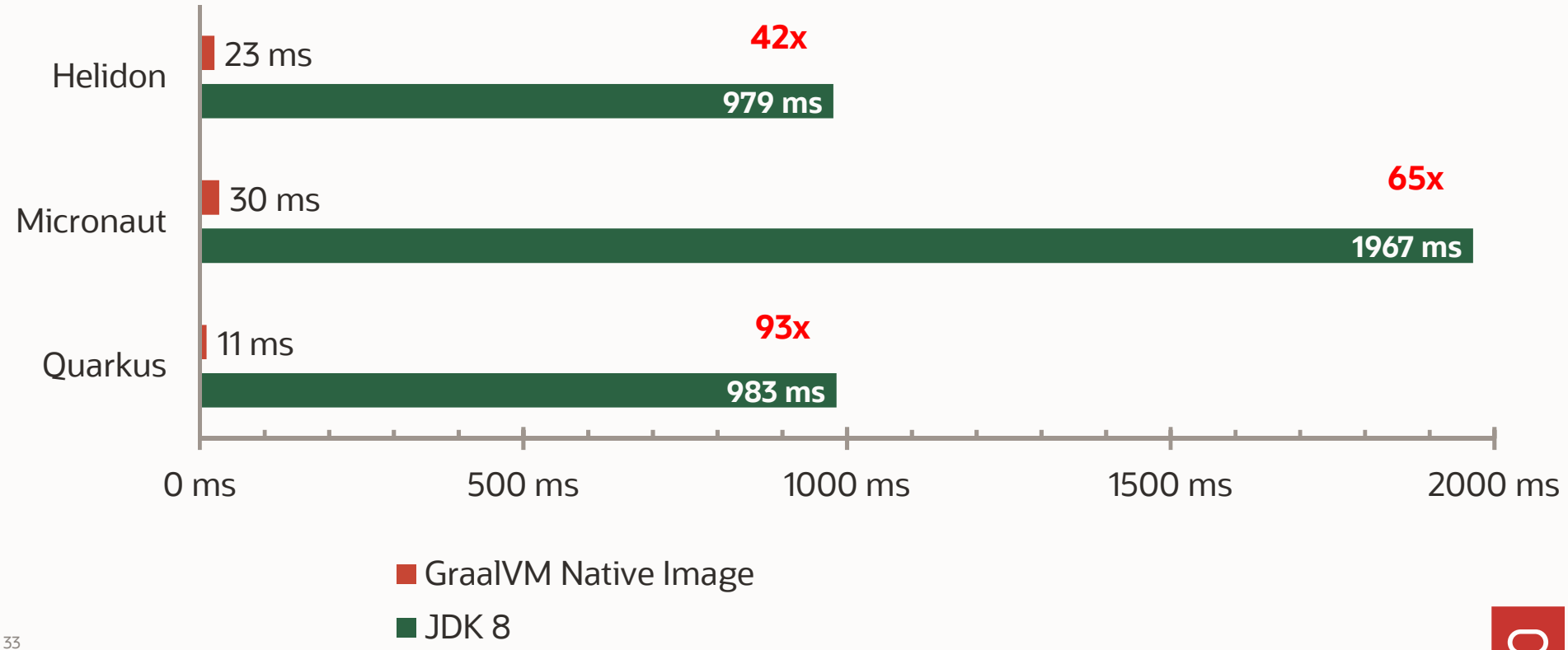
- Spring Native is in beta now, as the latest 0.10.0
- Spring Native 0.9.0 supports Spring Boot 2.4.3
- Spring Native 0.9.1 will support Spring Boot 2.4.4
- Spring Native beta 0.10.0, based on GraalVM 21.1.0, will support Spring Boot 2.5, etc.
- https://spring.io/blog/2021/06/14/spring-native-0-10-0-available-now

Spring Native should be GA in the next few months.

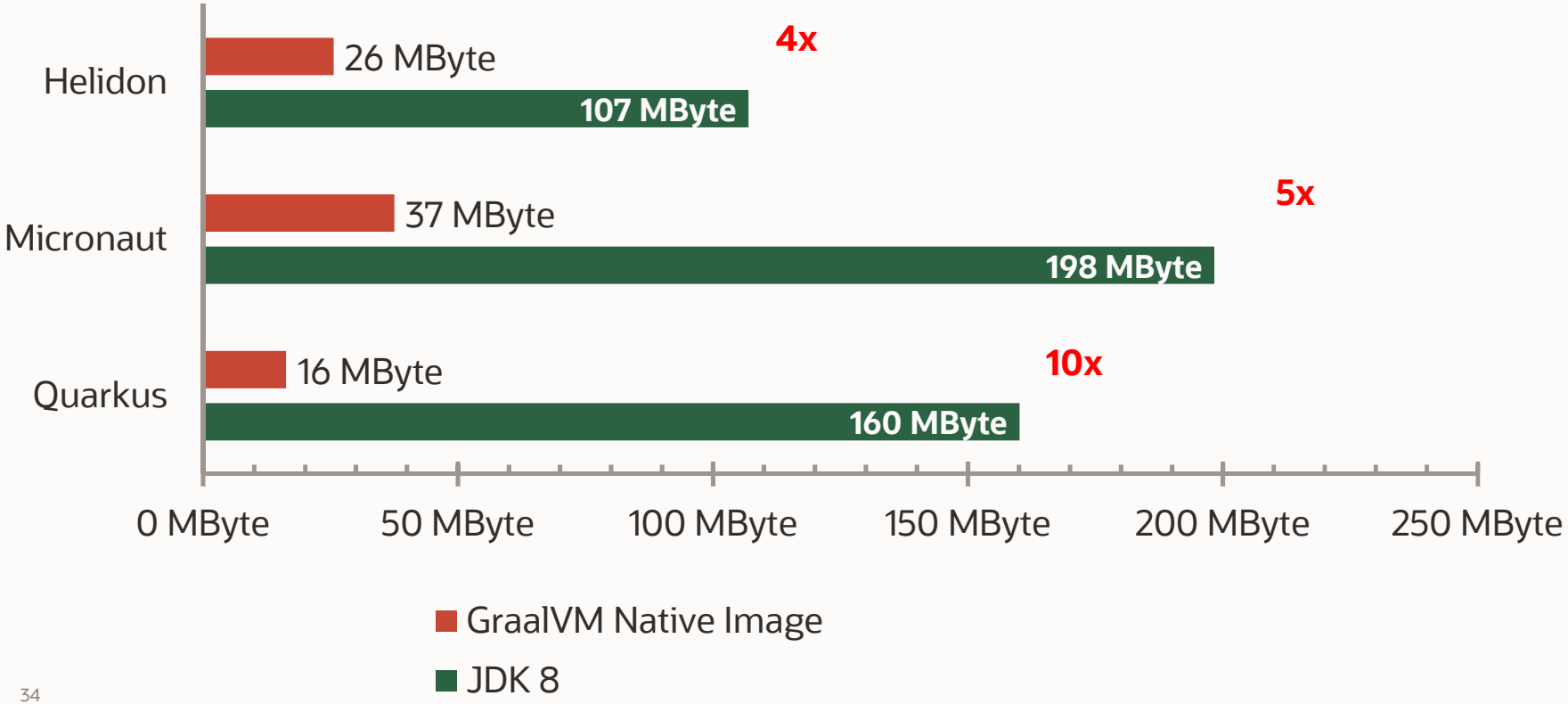
# What GraalVM is for Microservices and Cloud Runtime

## Up to 5x Less Memory
## 100x Faster Startup

# Cloud Services – Startup Time



# Cloud Services – Memory Footprint



ORACLE

# GraalVM Enterprise Components

# GraalVM Enterprise Components – Version 21.2.0.1
MacOS

```
wolfgangweigend@wolfgangweigend-mac hello % gu list
ComponentId              Version         Component name                Stability       Origin
------------------------------------------------------------------------------------------------------------
graalvm                  21.2.0.1        GraalVM Core                  -
R                        21.2.0          FastR                         Experimental    github.com
espresso                 21.2.0.1        Java on Truffle               Experimental    oca.opensource.oracle.com
js                       21.2.0.1        Graal.js                      Supported
llvm-toolchain           21.2.0.1        LLVM.org toolchain            Supported       oca.opensource.oracle.com
native-image             21.2.0.1        Native Image                  Early adopter   oca.opensource.oracle.com
nodejs                   21.2.0.1        Graal.nodejs                  Supported       oca.opensource.oracle.com
python                   21.2.0.1        Graal.Python                  Experimental    oca.opensource.oracle.com
wasm                     21.2.0.1        GraalWasm                     Experimental    oca.opensource.oracle.com
```




ORACLE

# GraalVM 21.2.0 Release Notes

## Summary

# GraalVM Enterprise Edition 21.2.0 Release Notes (1)

- **Java and Compiler Updates**

The Oracle JDK release that GraalVM Enterprise Edition is built on was updated to:

- 8u301 for Java 8 based GraalVM Enterprise
  - ❖ Java SE 8 release notes
- 11.0.12 for Java 11 based GraalVM Enterprise
  - ❖ Java SE 11 release notes
- 16.0.2 for Java 16 based GraalVM Enterprise
  - ❖ Java SE 16 release notes




***https://docs.oracle.com/en/graalvm/enterprise/21/docs/overview/release-notes/***

# GraalVM Enterprise Edition 21.2.0 Release Notes (2)

- Java and Compiler Updates
- Platform Updates
- Native Image
- Polyglot Runtime
- Java on Truffle
- JavaScript
- WebAssembly
- LLVM Runtime
- Ruby
- Python
- R
- Tools
- Polyglot Embedding
- Truffle Language and Tool Implementations




https://docs.oracle.com/en/graalvm/enterprise/21/docs/overview/release-notes/

ORACLE

# GraalVM Espresso

## Java on Truffle

# GraalVM Enterprise Edition 21.0.0 – Espresso (1)



- A meta-circular Java bytecode interpreter for the GraalVM
- Espresso is a fully meta-circular implementation of *The Java Virtual Machine Specification, Java SE 8 and 11 Edition*, written in Java, capable of running non-trivial programs at speed
- A Java bytecode interpreter at its core, turned Just-In-Time (JIT) compiler by leveraging Truffle and the Graal compiler on the GraalVM
- It highlights the sublime potential of the GraalVM as a platform for implementing high-performance languages and runtimes




***https://github.com/oracle/graal/tree/master/espresso***

# GraalVM Enterprise Edition 21.0.0 – Espresso (2)

- Espresso is still an early prototype, but it already passes the Java Compatibility Kit (a.k.a. the JCK or TCK for Java SE) 8c and 11 runtime suite
- Espresso can compile itself with both javac and (the Eclipse Java Compiler) ecj

  It features complete meta-circularity: it can run itself any amount of layers deep, preserving all the capabilities (Unsafe, JNI, Reflection…) of the base layer. Running HelloWorld on three nested layers of Espresso takes ~15 minutes
- Espresso is similar to HotSpot Express, the same codebase can run either an 8 or 11 guest JVM, on either an 8 or 11 host JVM
- The development of Espresso happens mostly on HotSpot, but this configuration (Espresso on HotSpot) is only supported on Linux
- Espresso's native image runs on Linux, MacOS and Windows




***https://github.com/oracle/graal/tree/master/espresso***

# GraalVM Enterprise Edition 21.2.0 Release Notes (3)

- **Java on Truffle**
  - Introduced Truffle on Java HotSwap Plugin API which allows to reload the code without the need for restarting a running application. It is meant for framework developers to reflect changes to, e.g., annotations, framework-specific updates such as implemented services or beans. For more details, check the documentation.
  - Improved bytecode dispatch, yielding a 15-30% interpreter speed-up.
  - Added a new static object model implementation that dynamically generates host classes at run time. It is experimental at the moment.
  - Added fixes to prevent crashes when external threads enter Espresso and call native code.
  - Fixed crashes when running with older versions of GNU libc (<= 2.17).
  - Added support for additional interoperability messages for guest objects implementing Map, Map.Entry, List, Iterator, or Iterable.

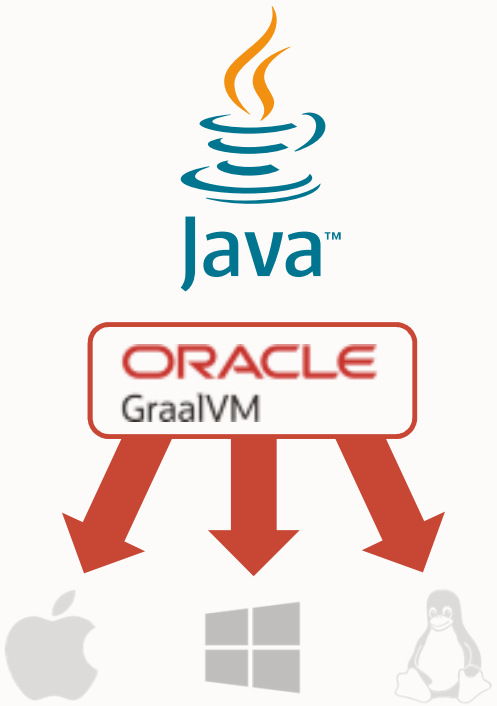



***https://docs.oracle.com/en/graalvm/enterprise/21/docs/overview/release-notes/***

# GraalVM Enterprise — Summary

## High-performance optimizing Just-in-Time (JIT) compiler


## Ahead-of-Time (AOT) “native image” compiler




## Multilingual Virtual Machine


- ***Test your applications with GraalVM***
  - ***Documentation and downloads***
- ***Connect your technology with GraalVM***
  - ***Integrate GraalVM into your application***

# Thanks!

## GraalVM Enterprise

**Wolfgang.Weigend@oracle.com**

**Twitter: @wolflook**