ORACLE

GraalVM 21
Feature Update der universellen VM

26. Mai 2021

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 for cloud native development
  - Java in Containers
- Summary

GraalVM Enterprise with Java SE Subscription

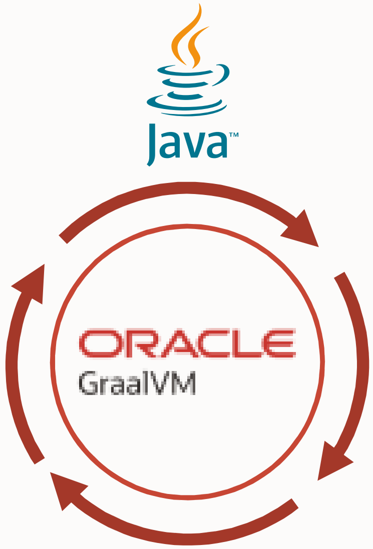
- Oracle Java SE Subscription now entitles customers to use Oracle GraalVM Enterprise at no additional cost
- Key benefits for Java SE Subscribers:
  - Native Image utility to compile Java to native executables that start almost instantly for containerized workloads
  - High-performance Java runtime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

ORACLE

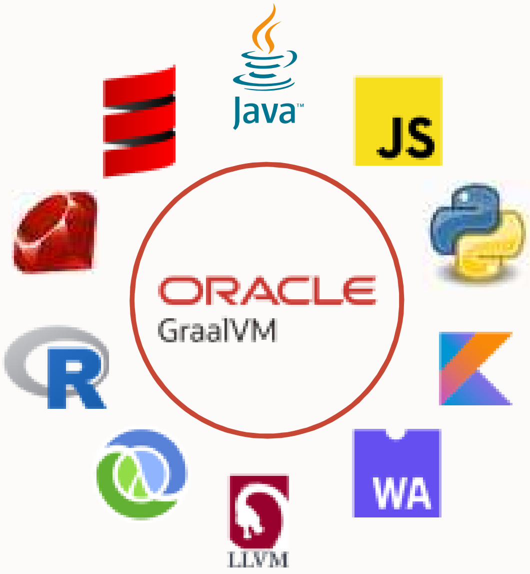
GraalVM Enterprise

GraalVM Enterprise

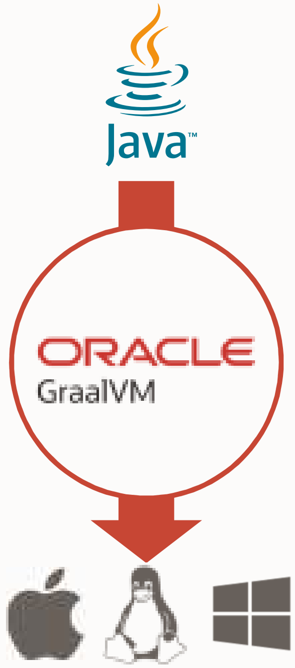
High-performance runtime that provides significant improvements in application performance and efficiency



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

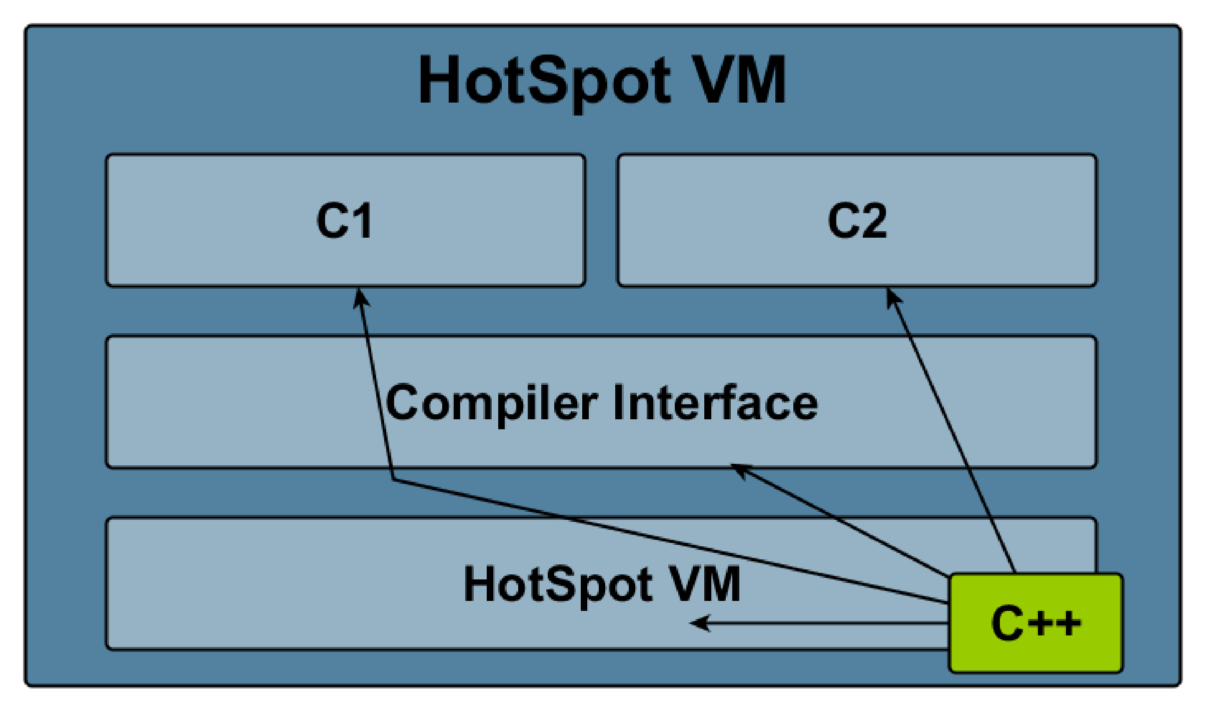


Multi-language support for the JVM

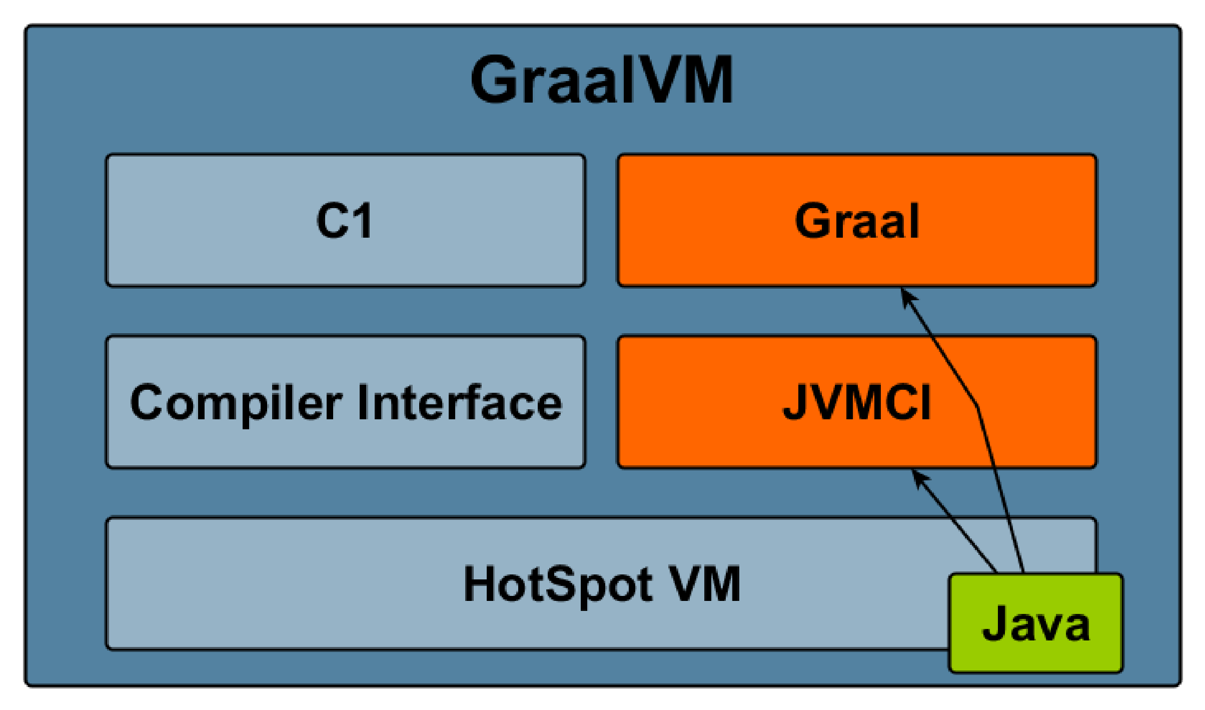


Ahead-of-Time (AOT) "native image" compiler

JIT Compiler written in C++



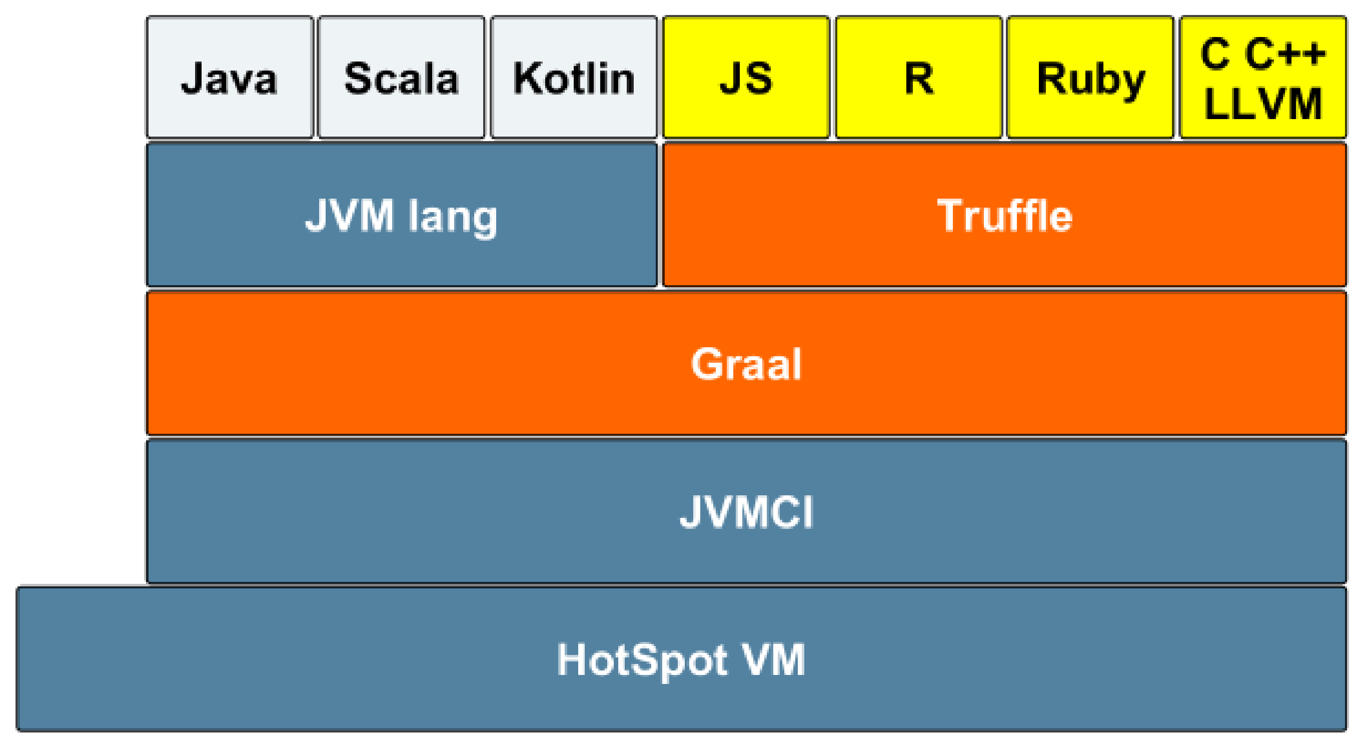
JIT Compiler written in Java



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)



Java Scala Kotlin JS R Ruby C C++ LLVM

JVM lang

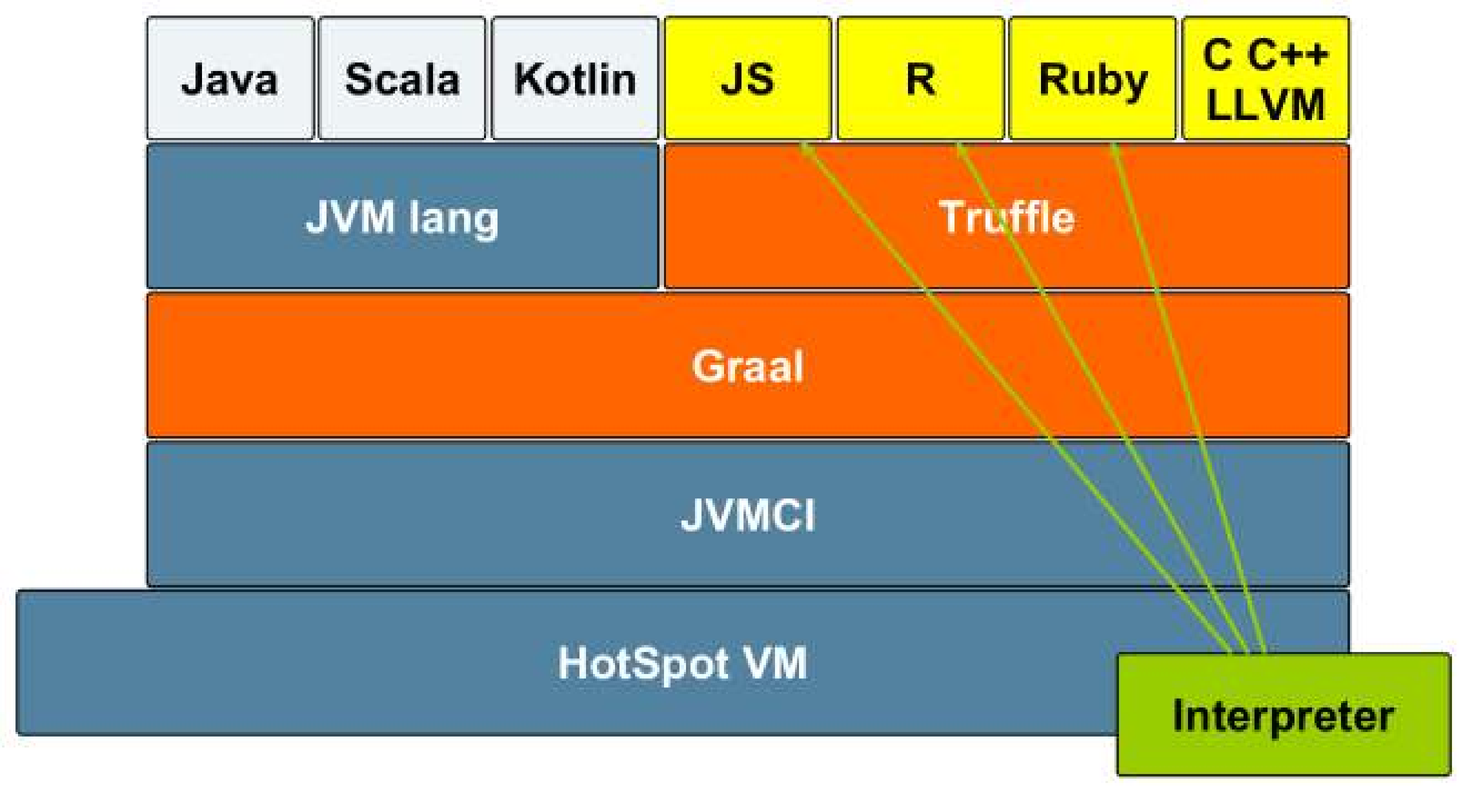
Truffle

Graal

JVMCI

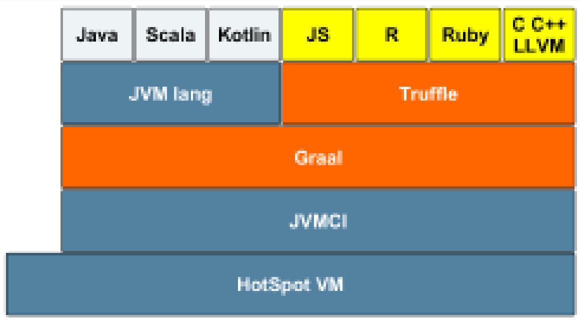
HotSpot VM

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		



Java Scala Kotlin JS R Ruby C C++ LLVM

JVM lang Truffle

Graal

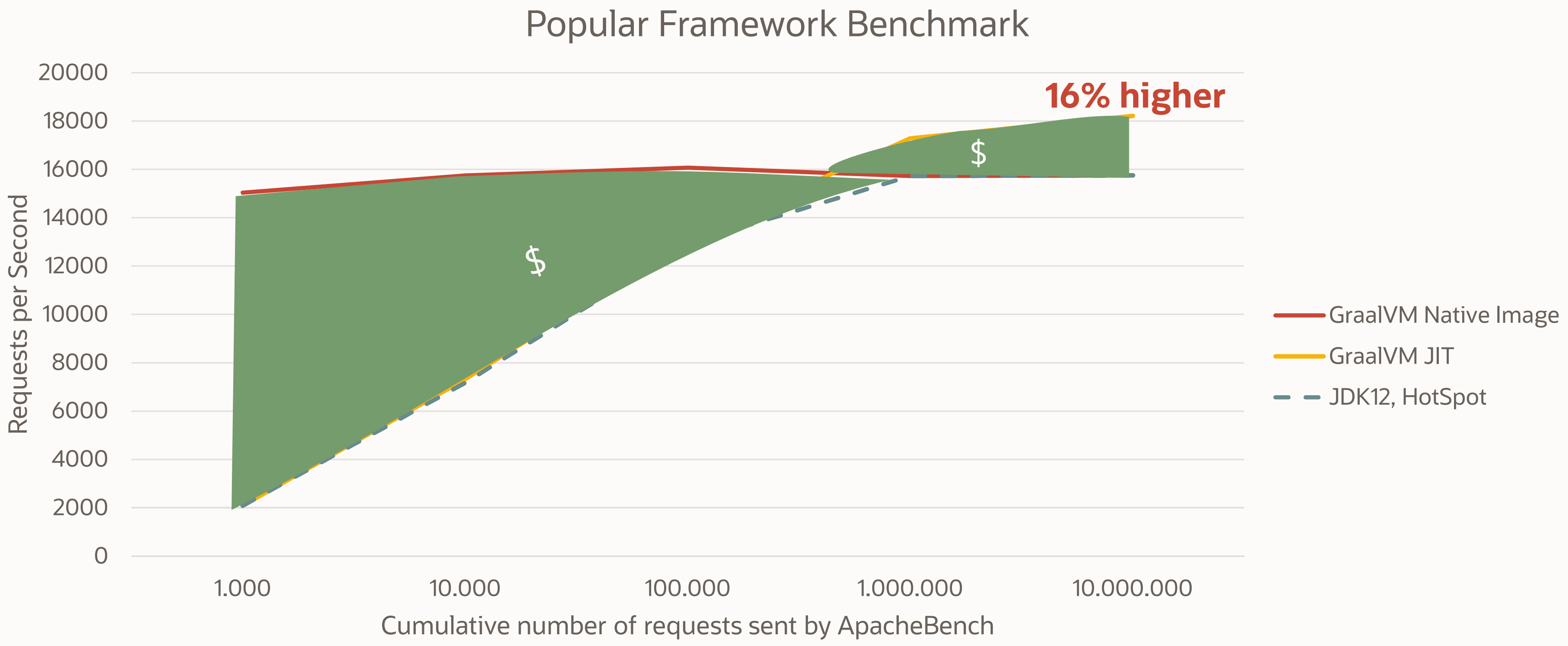
JVMCI

HotSpot VM

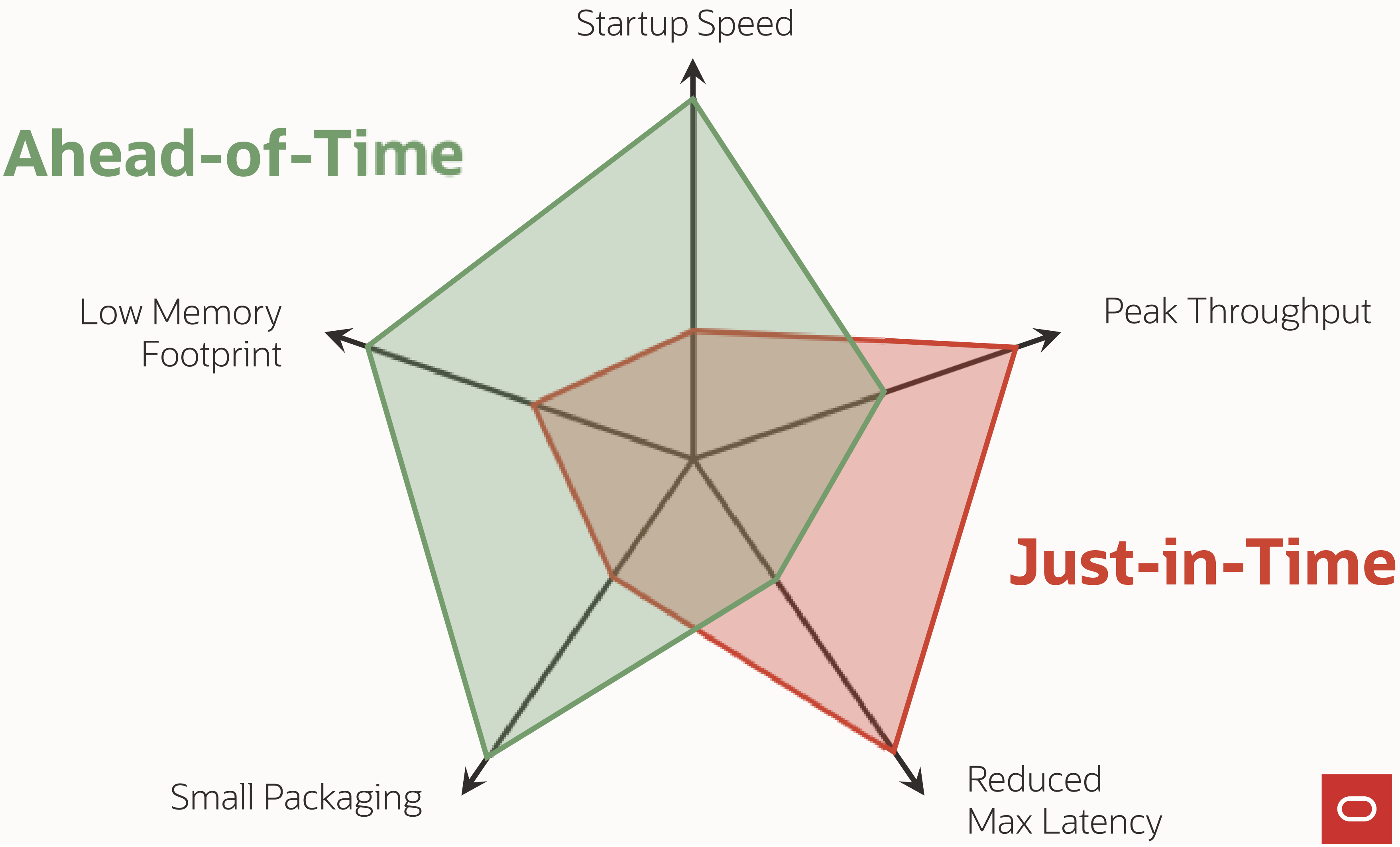
GraalVM - Language Usability for the Platform

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

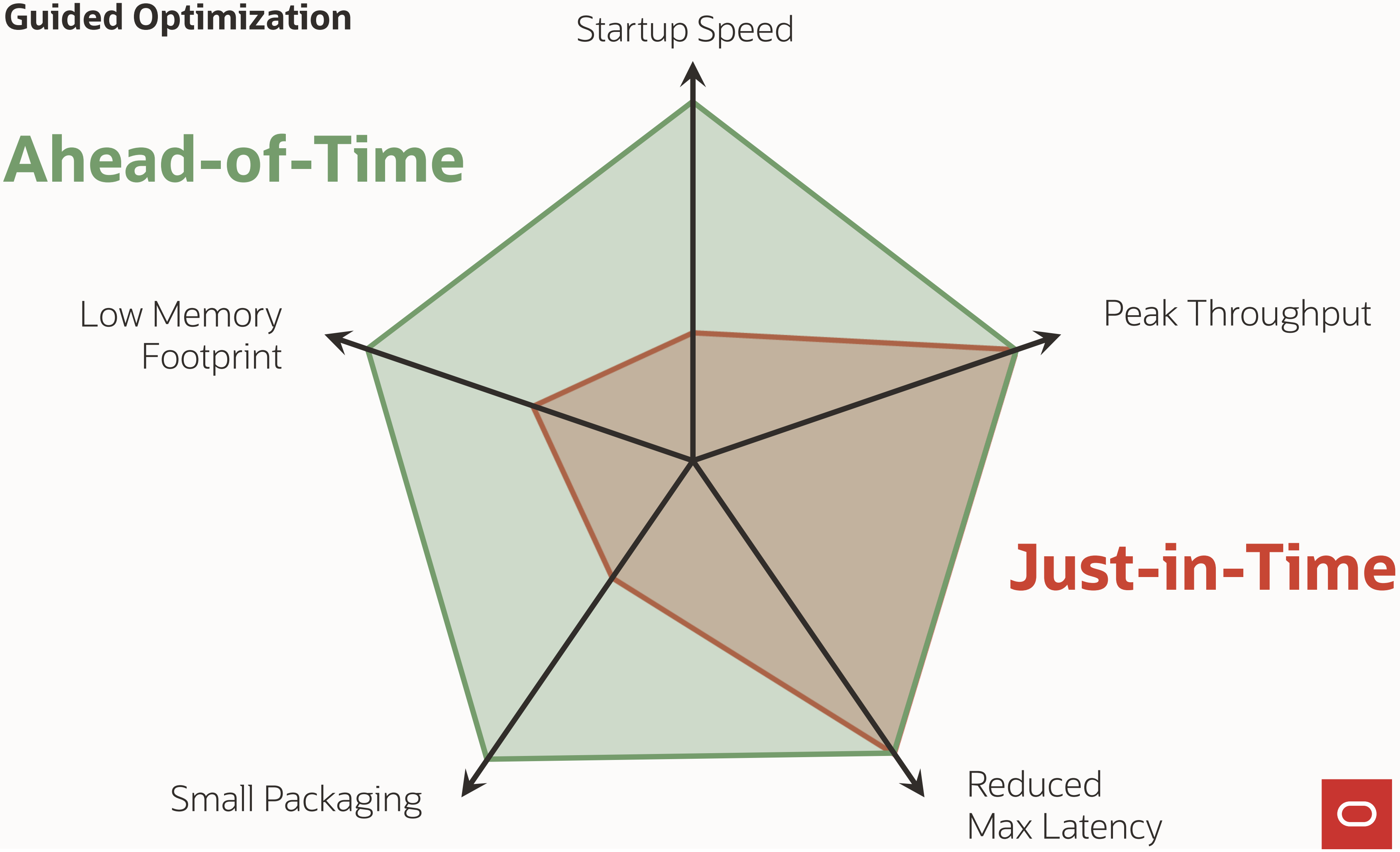


GraalVM Enterprise compilation performance characteristics



GraalVM Enterprise compilation performance characteristics

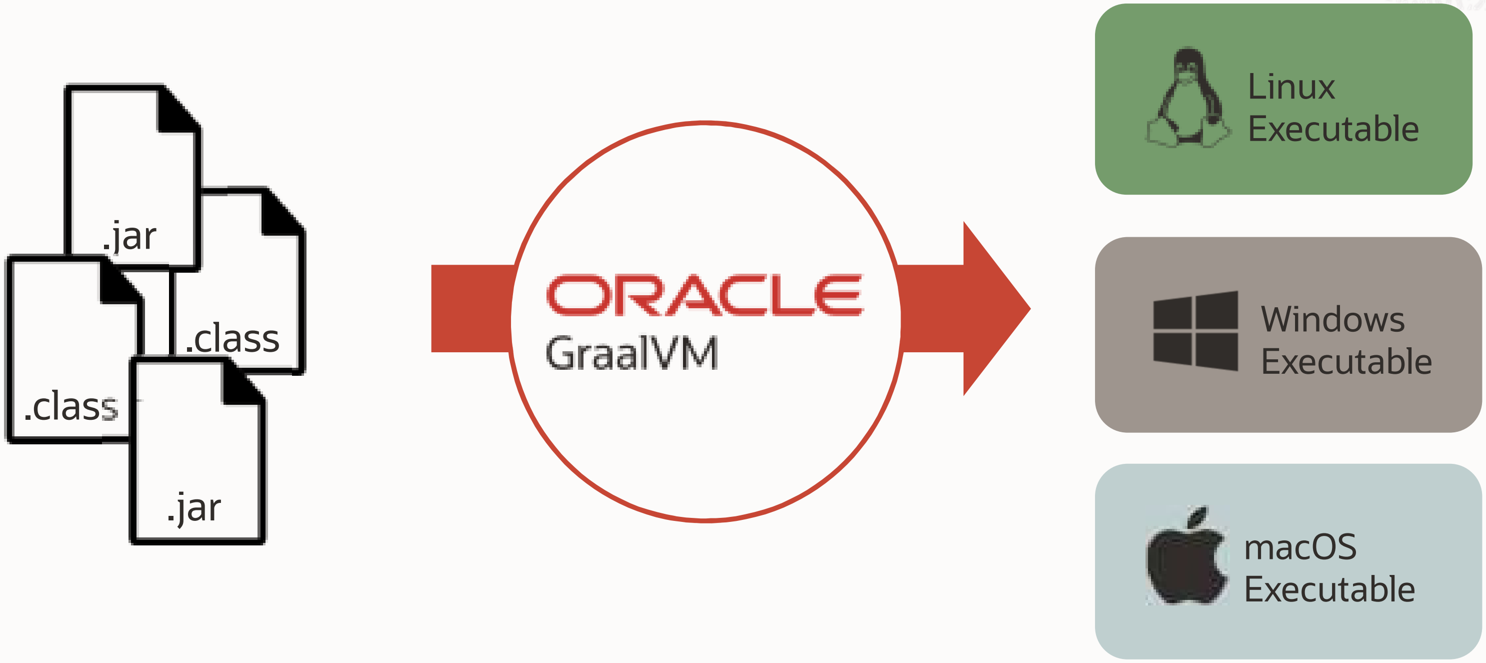
Profile Guided Optimization



GraalVM Native Image

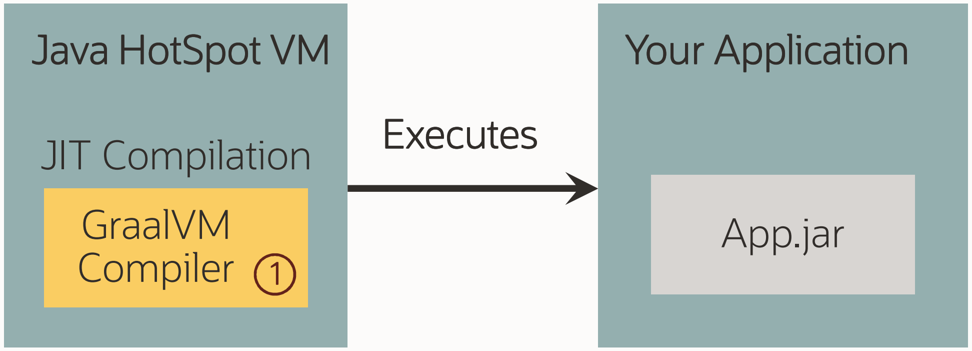
GraalVM Enterprise Native Image—Ahead-of-time compiler & runtime

Microservices and Containers



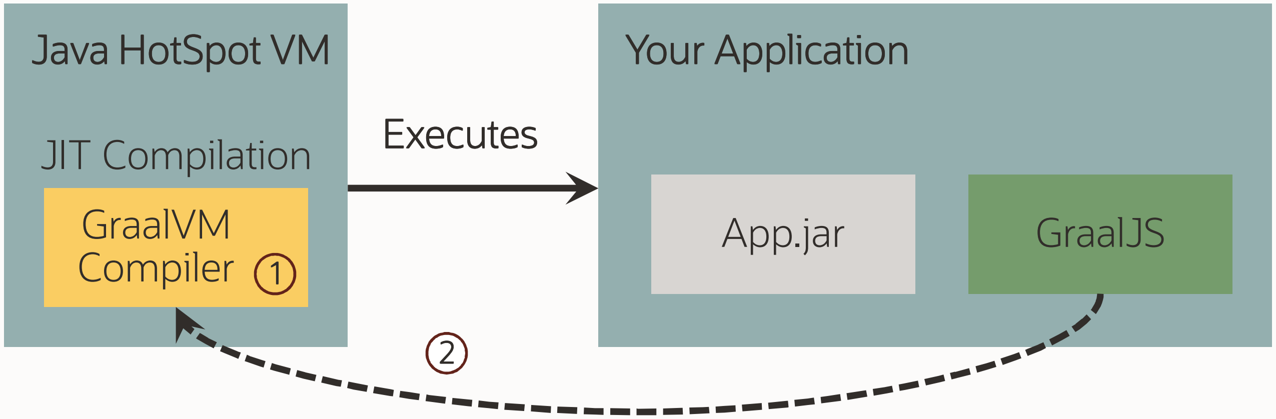
Up to 5x less memory
100x faster startup

One Compiler, Many Configurations



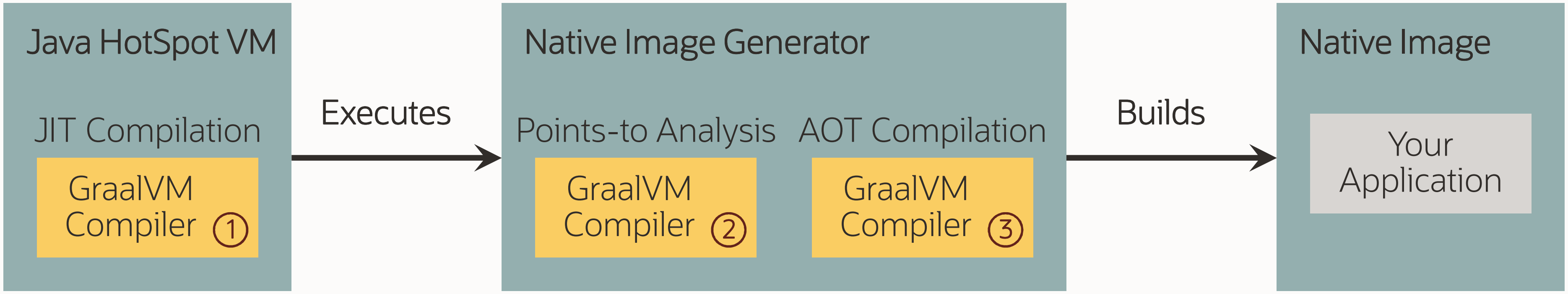
① Compiler configured for just-in-time compilation inside the Java HotSpot VM

One Compiler, Many Configurations



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

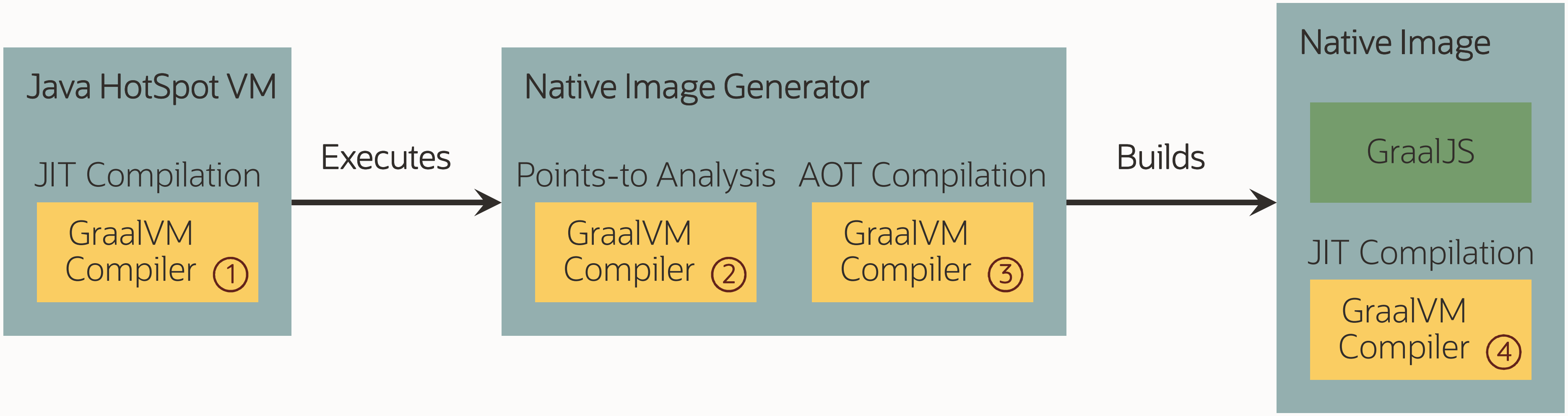
One Compiler, Many Configurations



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

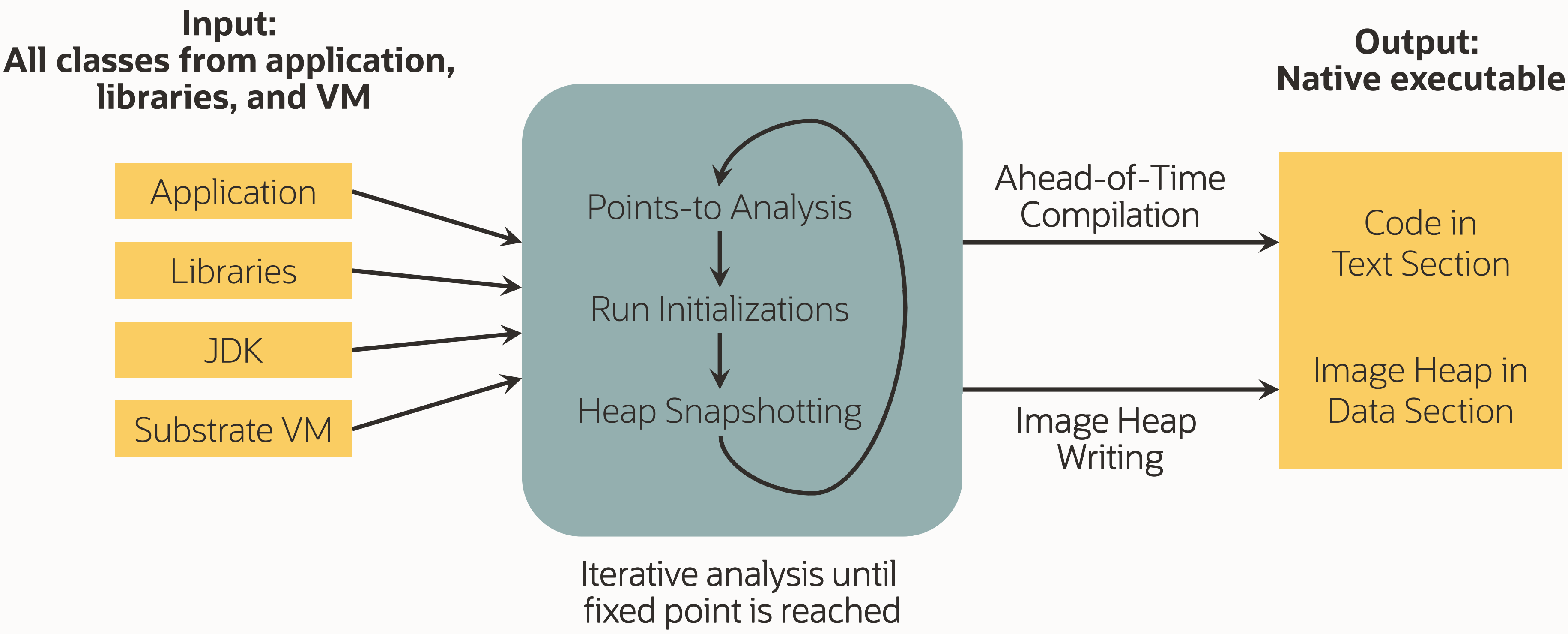
One Compiler, Many Configurations

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



Input:
All classes from application, libraries, and VM

Application

Libraries

JDK

Substrate VM

Points-to Analysis

Run Initializations

Heap Snapshotting

Iterative analysis until fixed point is reached

Ahead-of-Time Compilation

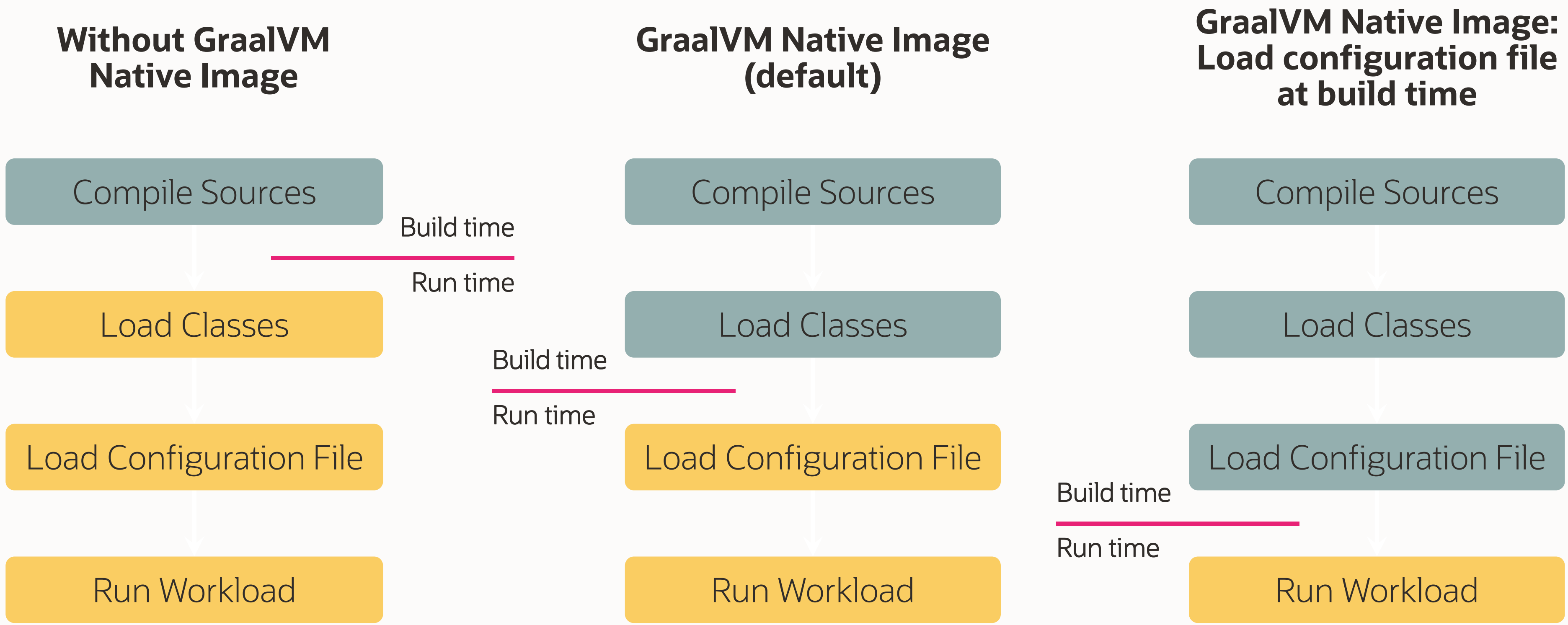
Image Heap Writing

Output:
Native executable

Code in Text Section

Image Heap in Data Section

Benefits of the Image Heap



Without GraalVM Native Image

Compile Sources

Build time

Run time

Load Classes

Load Configuration File

Run Workload

GraalVM Native Image (default)

Compile Sources

Load Classes

Build time

Run time

Load Configuration File

Run Workload

GraalVM Native Image: Load configuration file at build time

Compile Sources

Load Classes

Load Configuration File

Build time

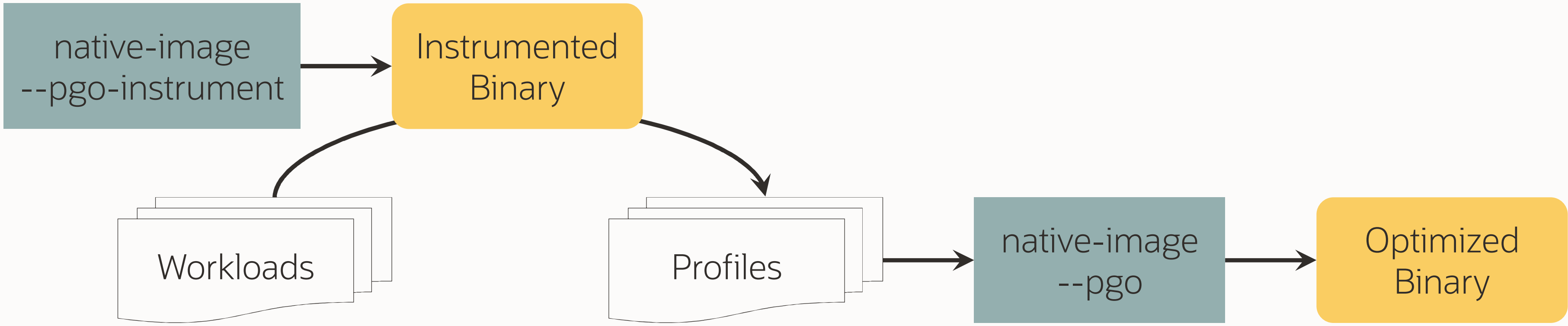
Run time

Run Workload

Profile-Guided Optimizations (PGO)

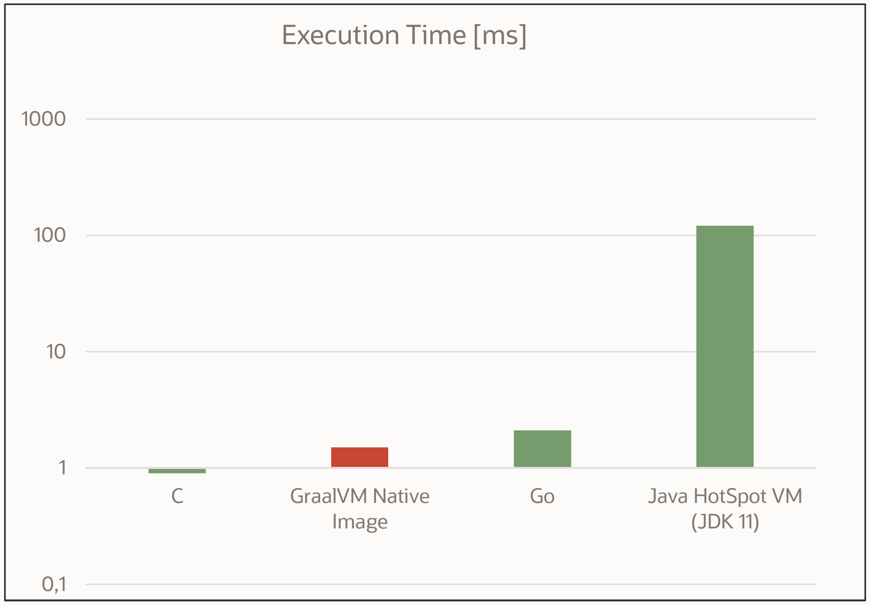
Out of Band Optimization

- 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

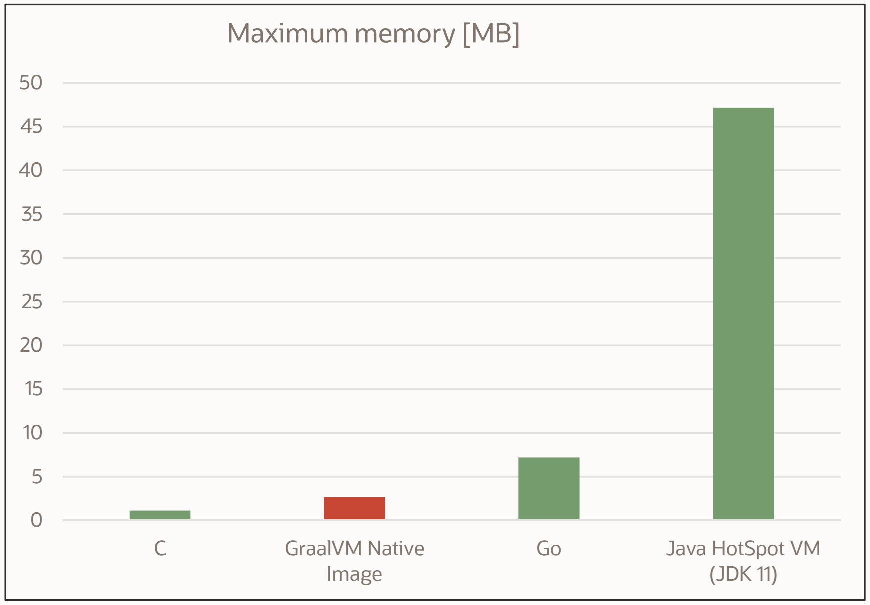


GraalVM Enterprise Native Image

Lower cloud costs for containerized workloads, and microservices



Competitive startup time



Significantly reduced memory requirements

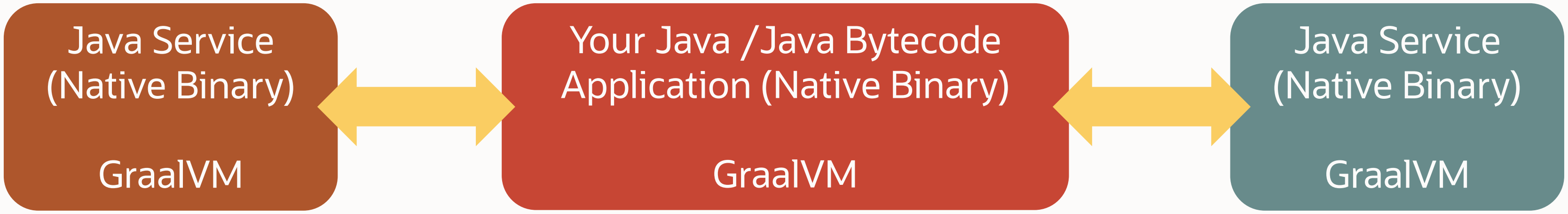
GraalVM Enterprise Native Image

Supported by leading frameworks

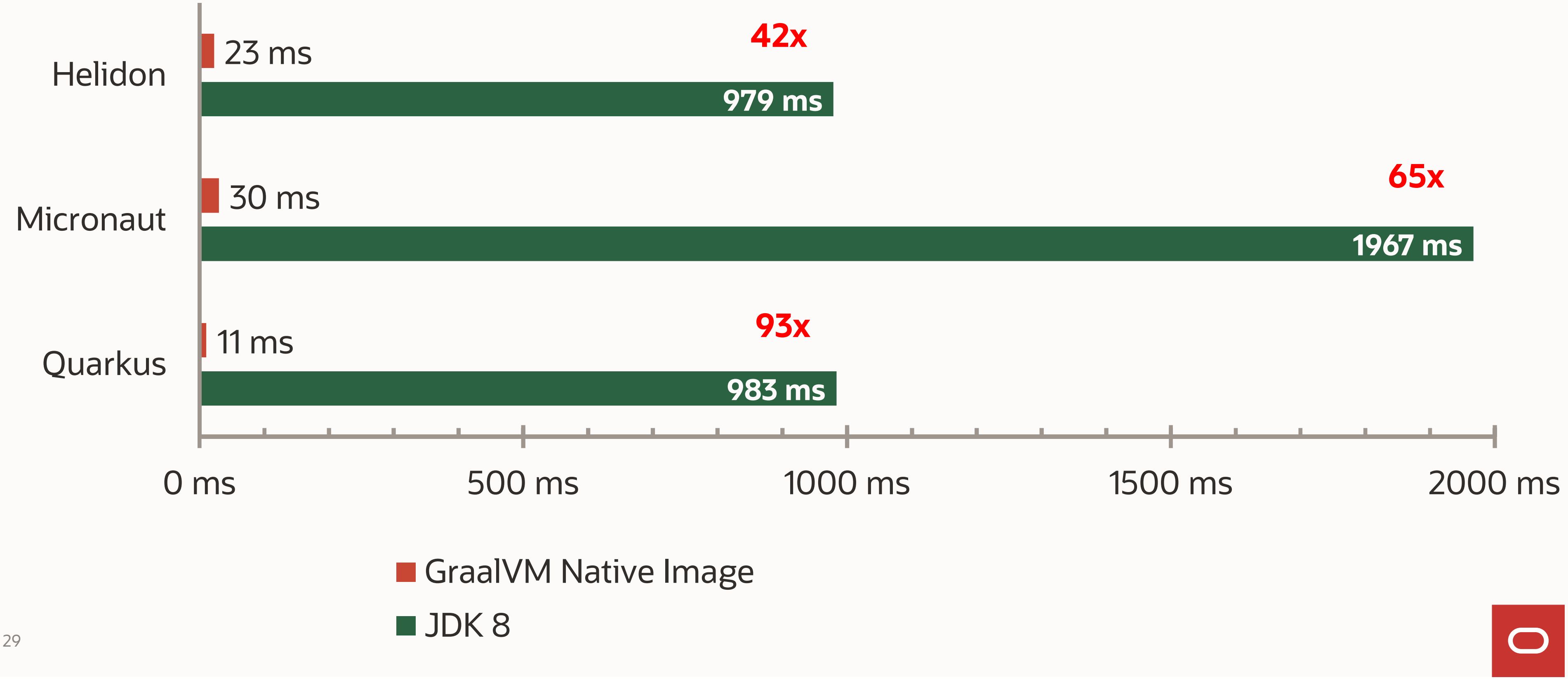
spring
by Pivotal
Coming Soon!
MICRONAUT™
™
QUARKUS
Netty
helidon

What GraalVM is for Microservices and Cloud Runtime

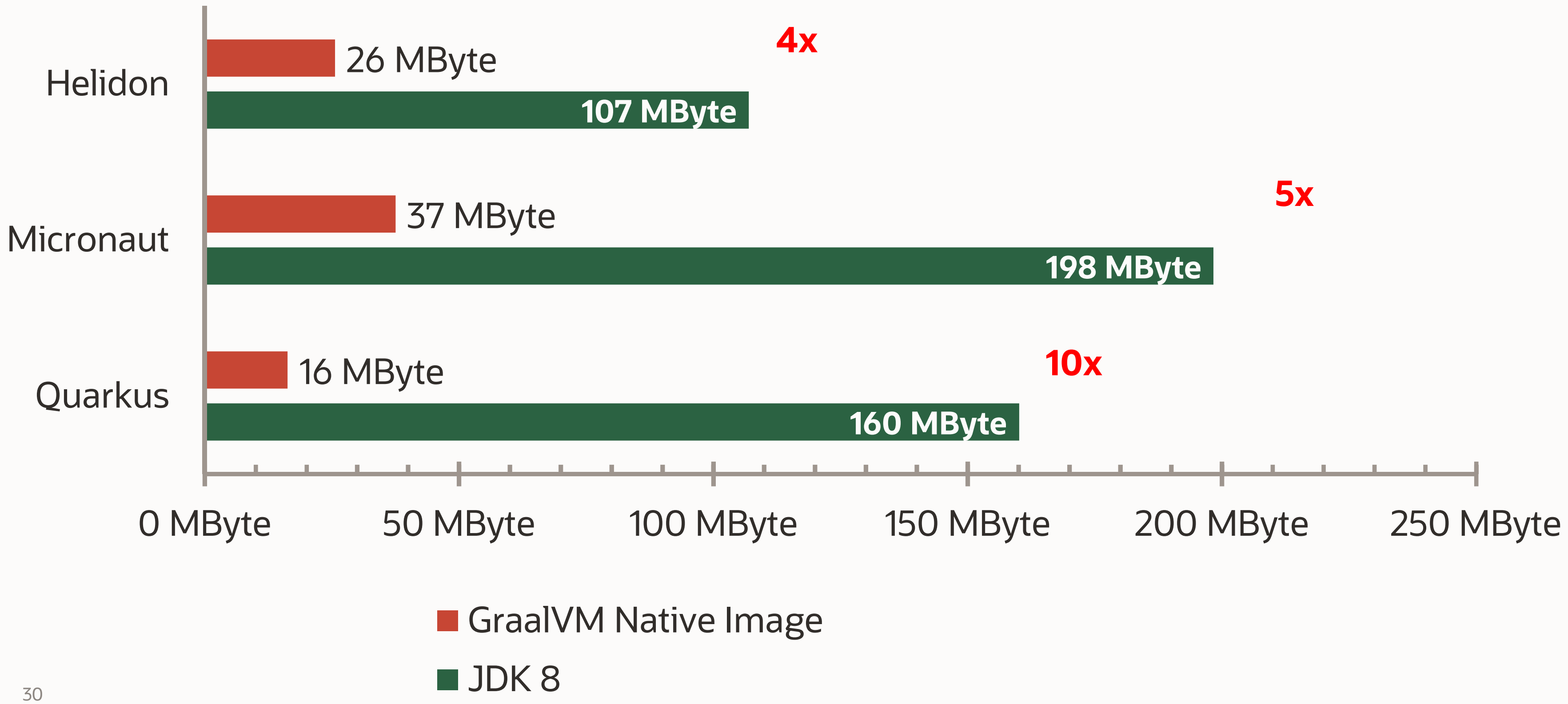
Up to 5x Less Memory
100x Faster Startup



Cloud Services – Startup Time



Cloud Services – Memory Footprint



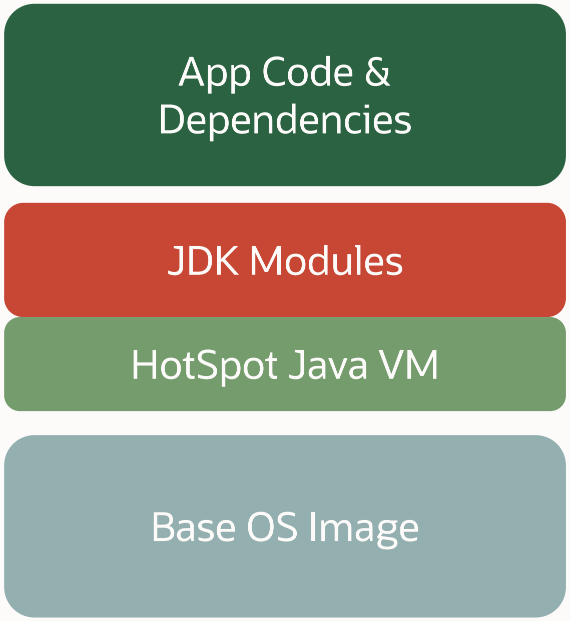
GraalVM Enterprise Native Image based on JDK 11 with microservices frameworks

Build Profiles

1. **Executable Jar**
   - Hollow jar
   - All third-party dependencies are stored separately to take advantage of Docker layering
2. **Jlink image**
   - Jlink optimized JRE + your application
   - Faster startup time and smaller image size with no code restrictions
3. **GraalVM native-image**
   - Fastest startup time and smallest memory consumption
   - Introduces some code restrictions related to usage of runtime operations

Java in Containers

Java in Container



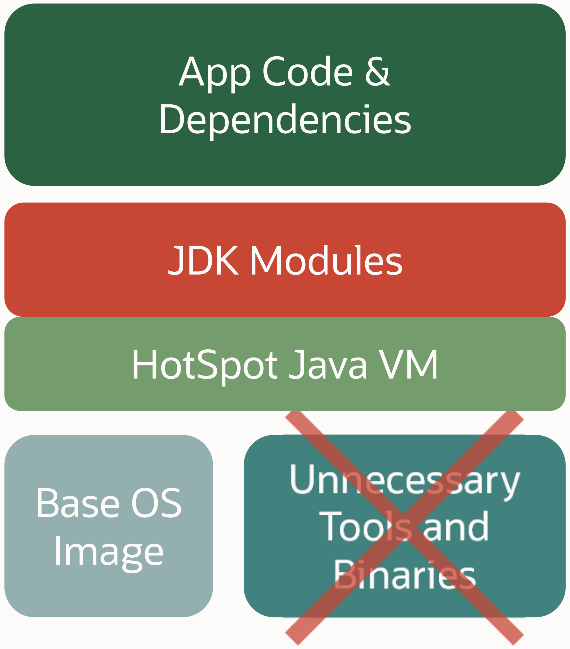
App Code & Dependencies

JDK Modules

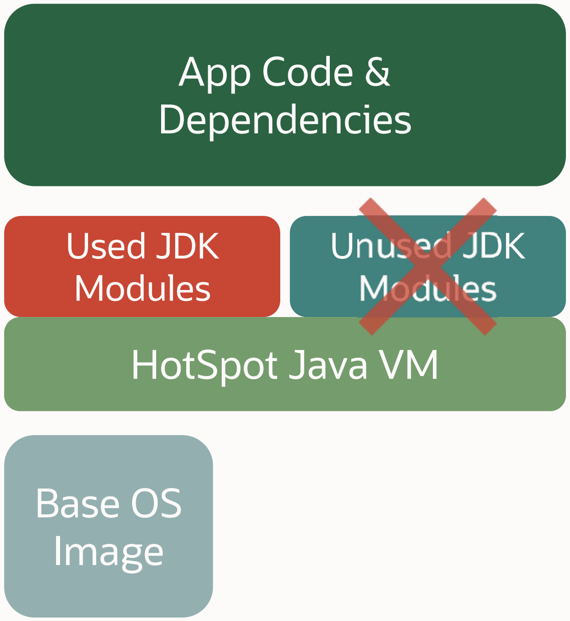
HotSpot Java VM

Base OS Image

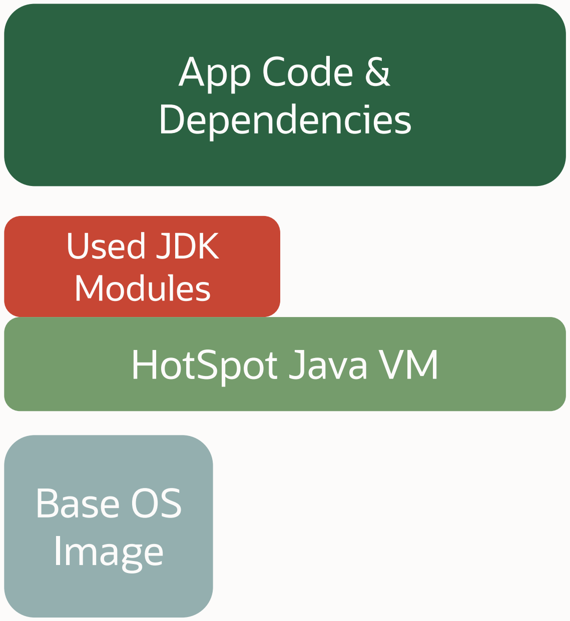
Java in a Slim/Distroless Container



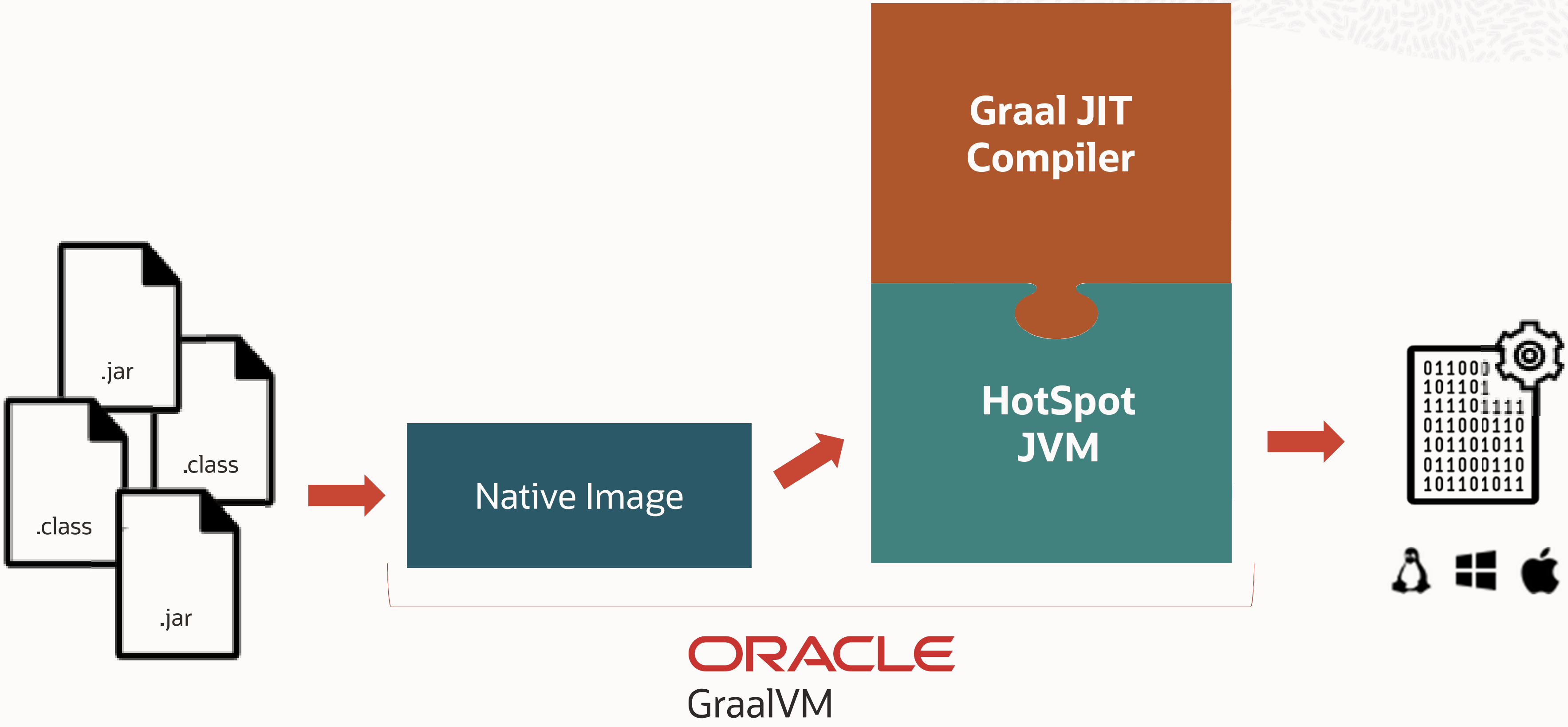
Java using `jlink` in a Slim/Distroless Container



Java using `jlink` in a Slim/Distroless Container



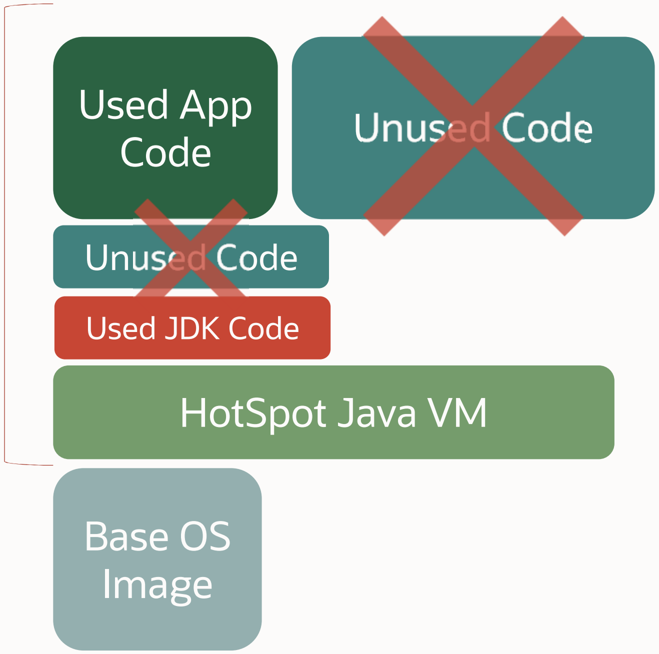
GraalVM Native Image — built on optimizing Graal compiler technology



Java Native Executable in Scratch Container w/ GraalVM Native Image

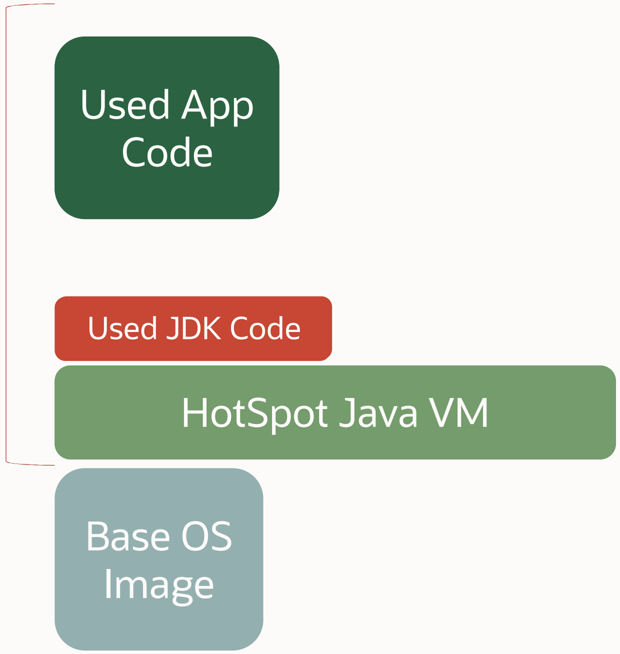
```
native-image --static -jar <jar> <app name>
```

ORACLE
GraalVM



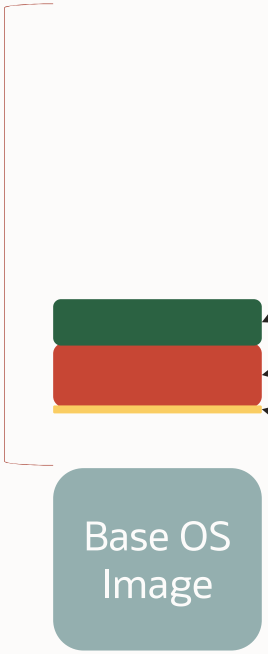
Java Native Executable in Scratch Container w/ GraalVM Native Image

ORACLE GraalVM



Java Native Executable in Scratch Container w/ GraalVM Native Image

ORACLE
GraalVM



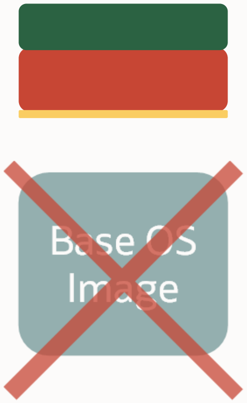
machine code for app, dependencies, & JDK code

thin runtime services layer & GC

libc statically linked into executable

Java Native Executable in Scratch Container w/ GraalVM Native Image

```
FROM scratch
COPY helloworld app
ENTRYPOINT ["/app"]
```



Java Native Executable in Scratch Container w/ GraalVM Native Image

Compile, generate executable, build Container

```
$ javac HelloWorld.java
```

```
$ native-image --static HelloWorld hello
-rwxrwxr-x. 1 opc opc  11M Jan 26 18:54 hello
```

```
$ docker build . -t hello:scratch
REPOSITORY           TAG              IMAGE ID         CREATED              SIZE
hello                static           7afc946a849e     About a minute ago   10.7MB
```

Java Hello World container image size **~ 11 MB**

JVMs in Containers — recap

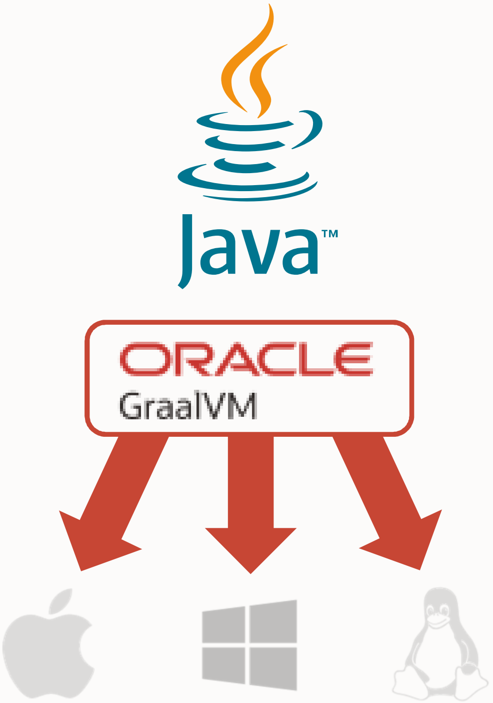
- JVM behaves as a good (Container) citizen
- Reduce “latency”
  - Container Startup
  - Application Startup
- All OpenJDK investments “leaks” into containers!
  - New Java languages
  - New JDK Features
  - Performance improvements
  - Footprint improvements
  - Etc.
- GraalVM offers unparalleled startup and container size reductions

GraalVM Enterprise — Summary

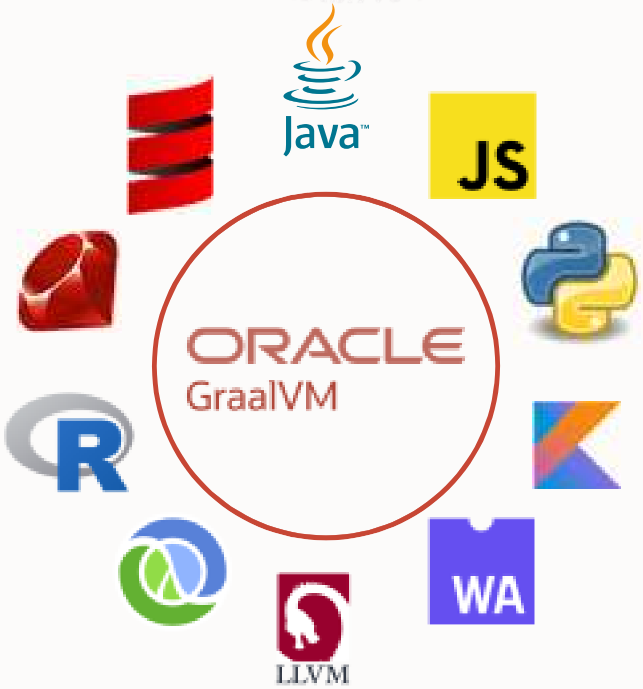
High-performance optimizing Just-in-Time (JIT) compiler

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

Ahead-of-Time (AOT) "native image" compiler



Multilingual Virtual Machine



Thanks!

GraalVM Enterprise

Wolfgang.Weigend@oracle.com

Twitter: @wolflook