ORACLE

GraalVM Enterprise 22.2 Feature Update und Verbesserungen beim Native Image

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
  - Native Image Performance
- GraalVM Enterprise Performance
- GraalVM Enterprise Components
- 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

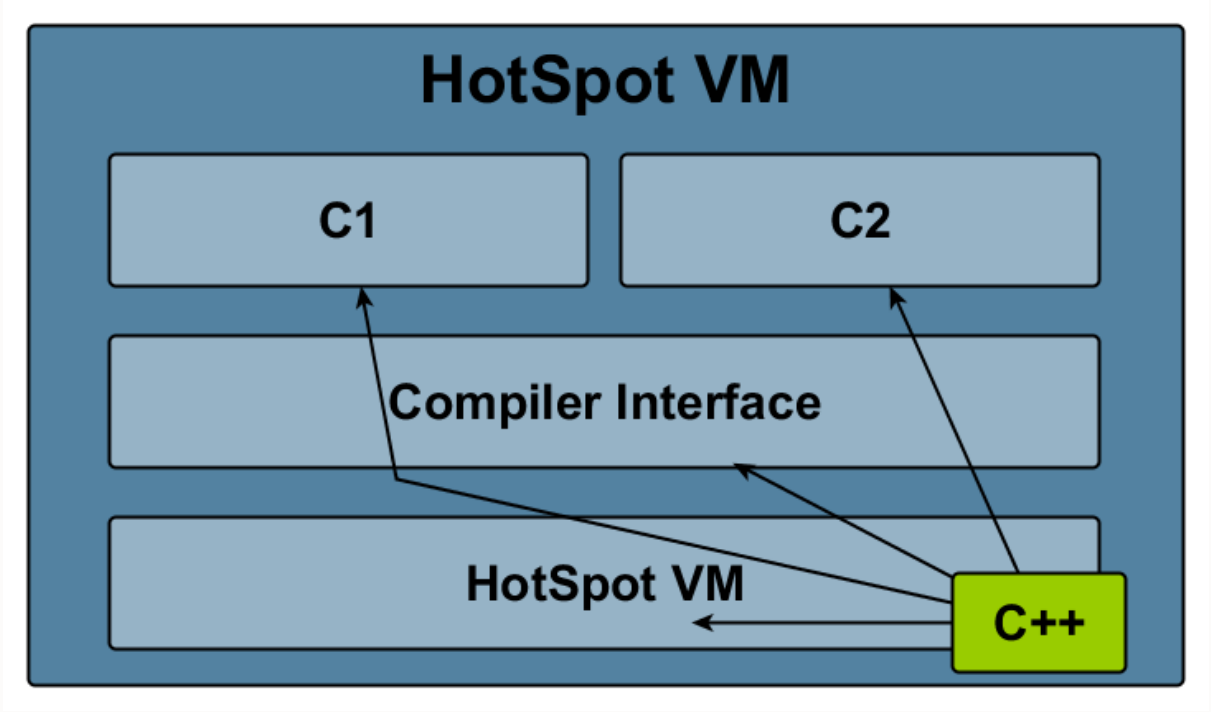
High-performance runtime that provides significant improvements in application performance and efficiency

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

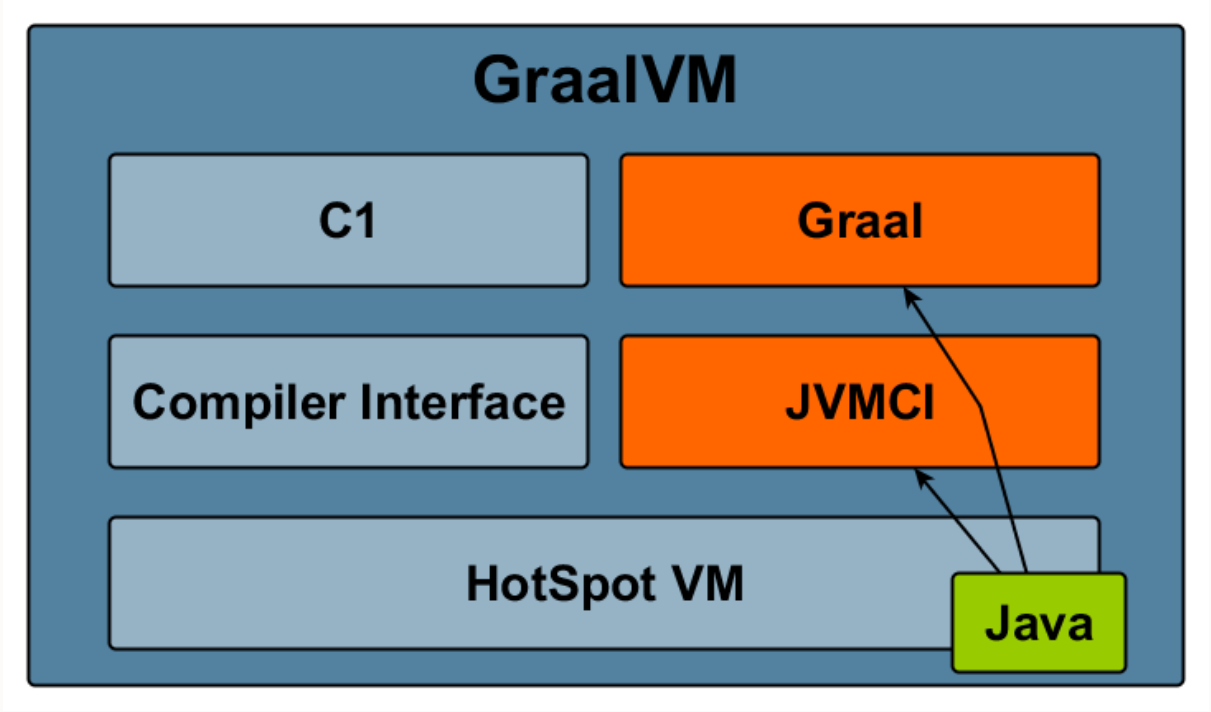
Multi-language support for the JVM

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

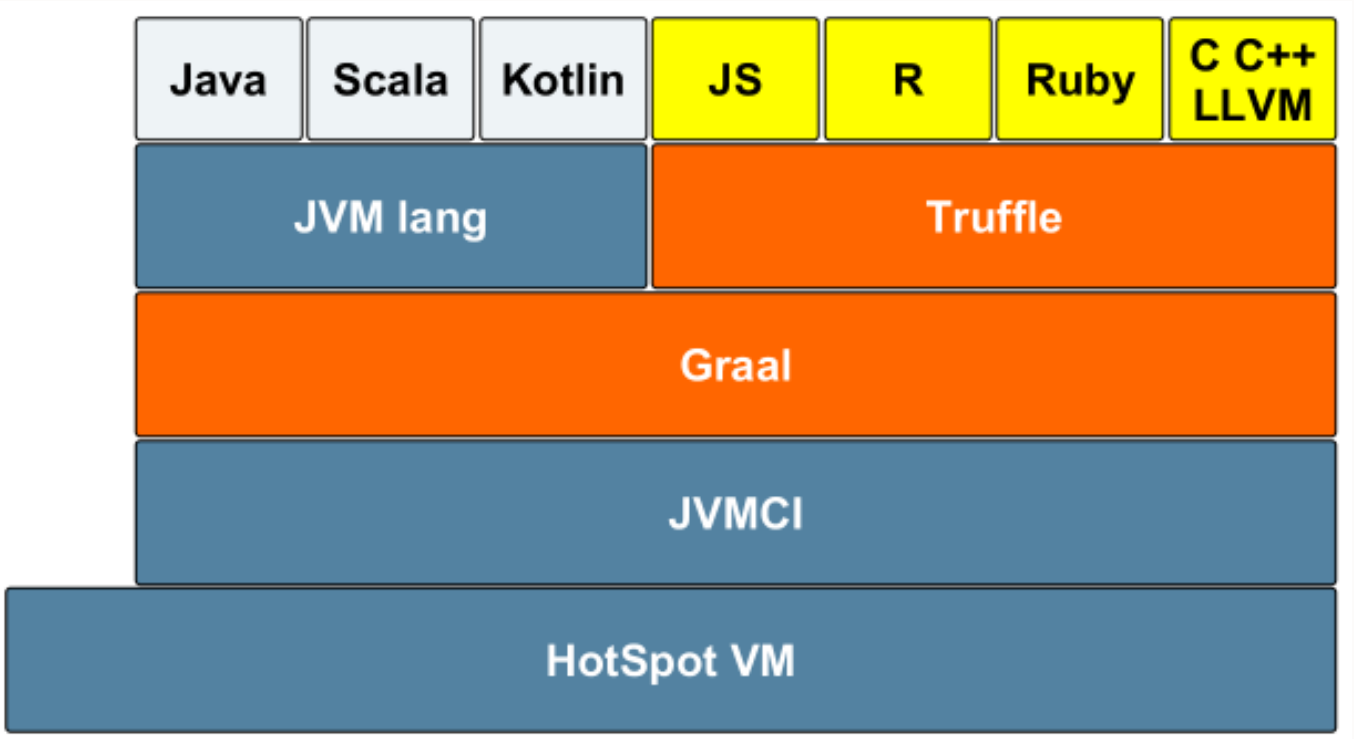
JIT Compiler written in C++



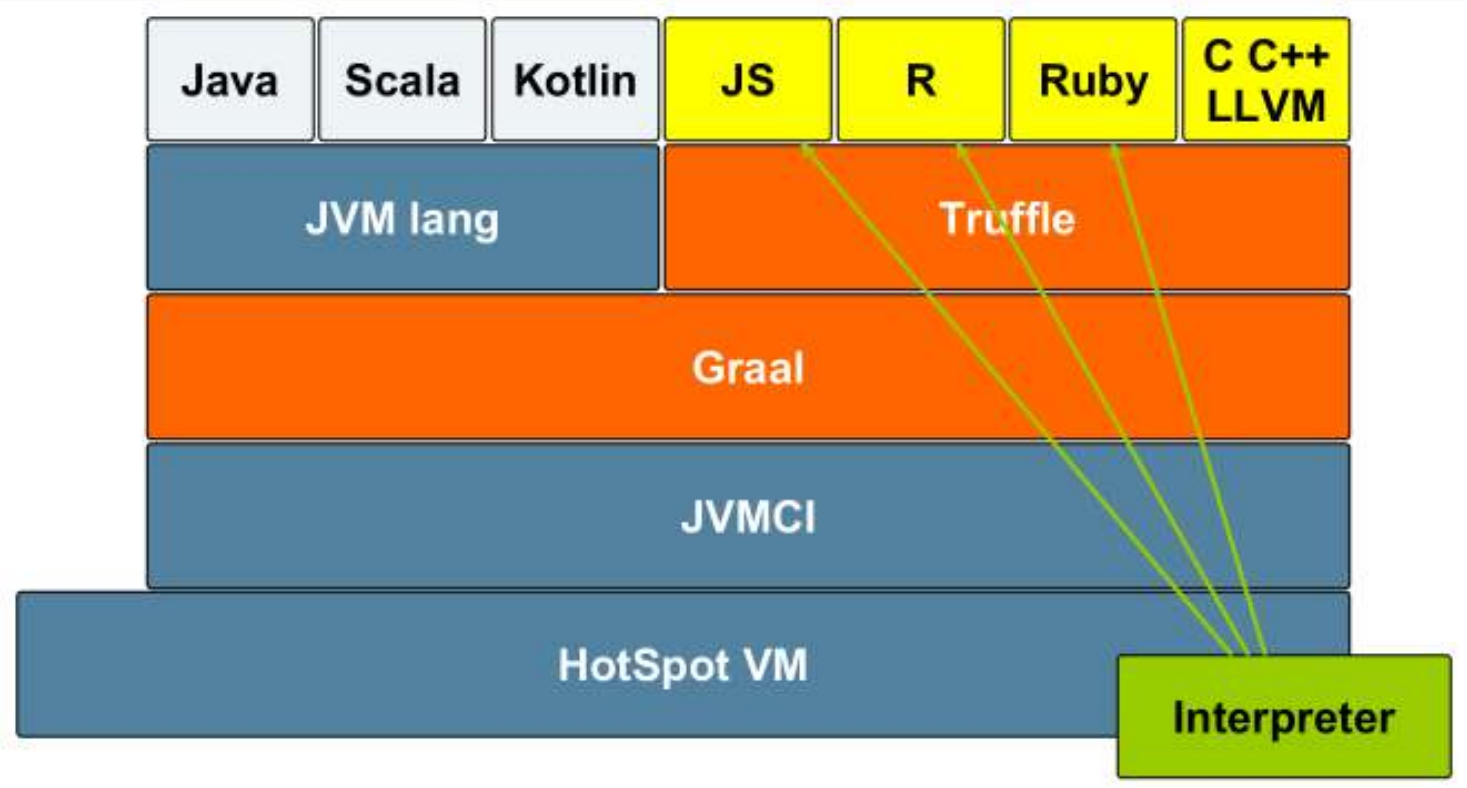
JIT Compiler written in Java



GraalVM - Polyglot (1)



GraalVM - Polyglot (2)



Production-ready and experimental features in GraalVM CE 22.1 by platform

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

ORACLE

GraalVM Native Image

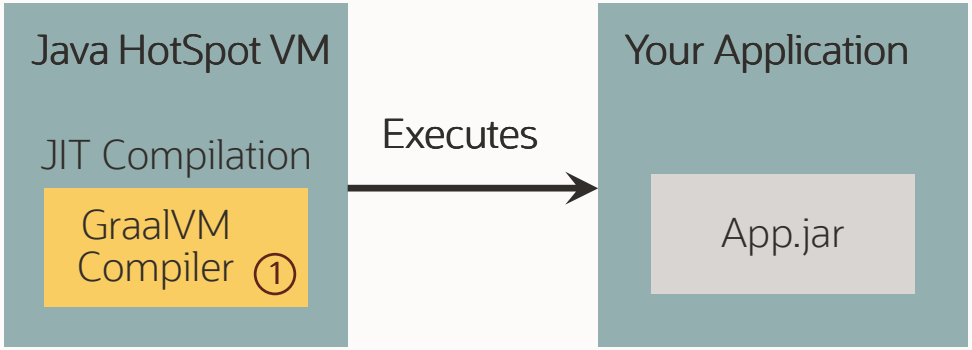
GraalVM Enterprise Native Image—Ahead-of-time compiler & runtime

Microservices and Containers

Up to 5x less memory
100x faster startup

One Compiler, Many Configurations

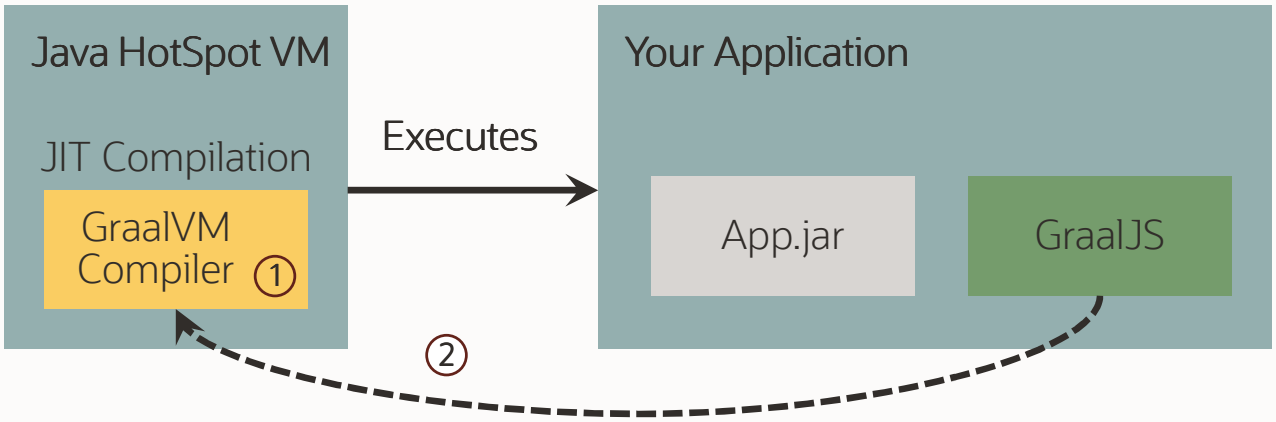
GraalVM



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

One Compiler, Many Configurations

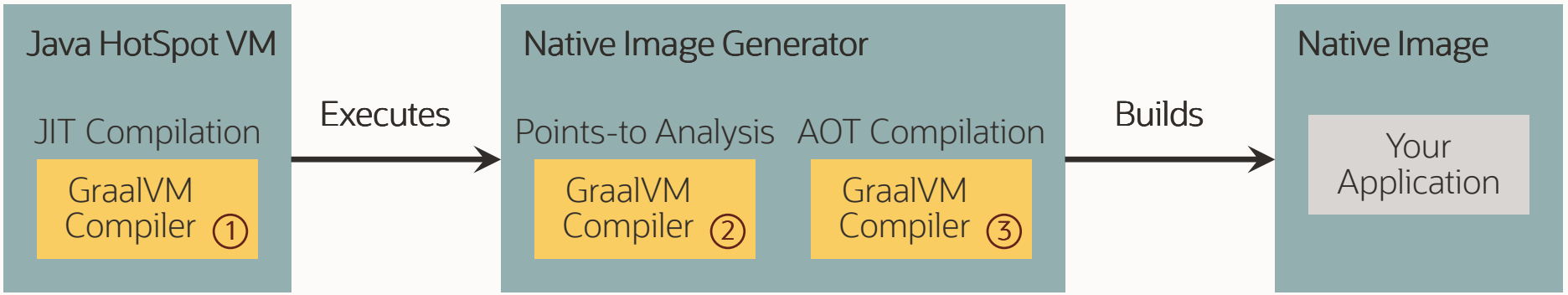
GraalVM



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

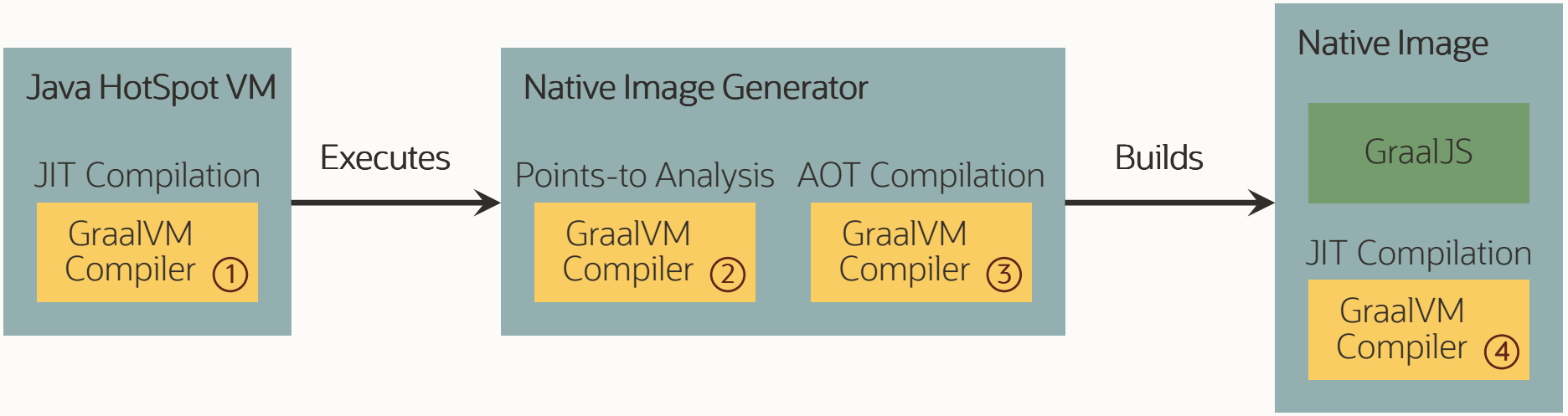
GraalVM



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

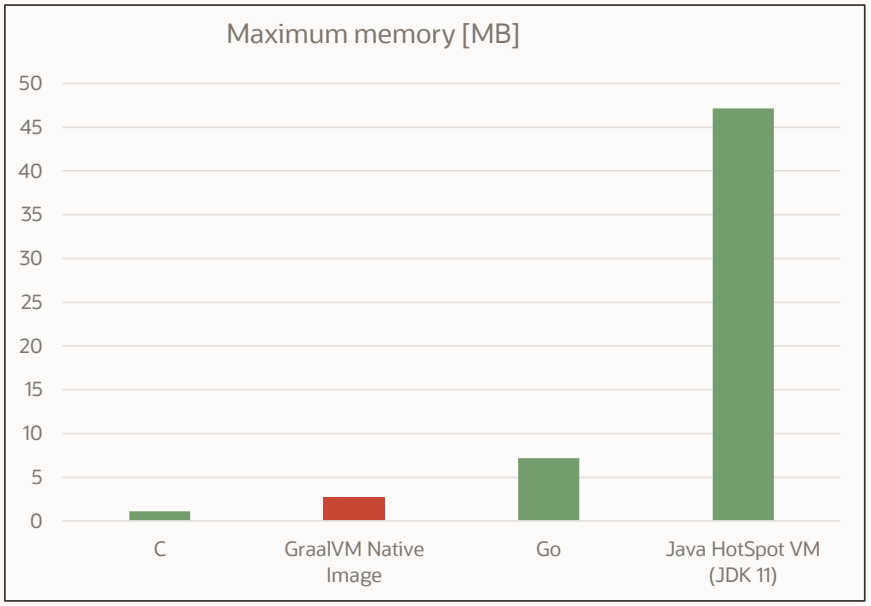


① Compiler configured for just-in-time compilation inside the Java HotSpot VM
② Compiler configured for static points-to analysis
③ Compiler configured for ahead-of-time compilation
④ Compiler configured for just-in-time compilation inside a Native Image

GraalVM Enterprise Native Image

Lower cloud costs for containerized workloads, and microservices

Competitive startup time



Significantly reduced memory requirements

GraalVM Enterprise Native Image

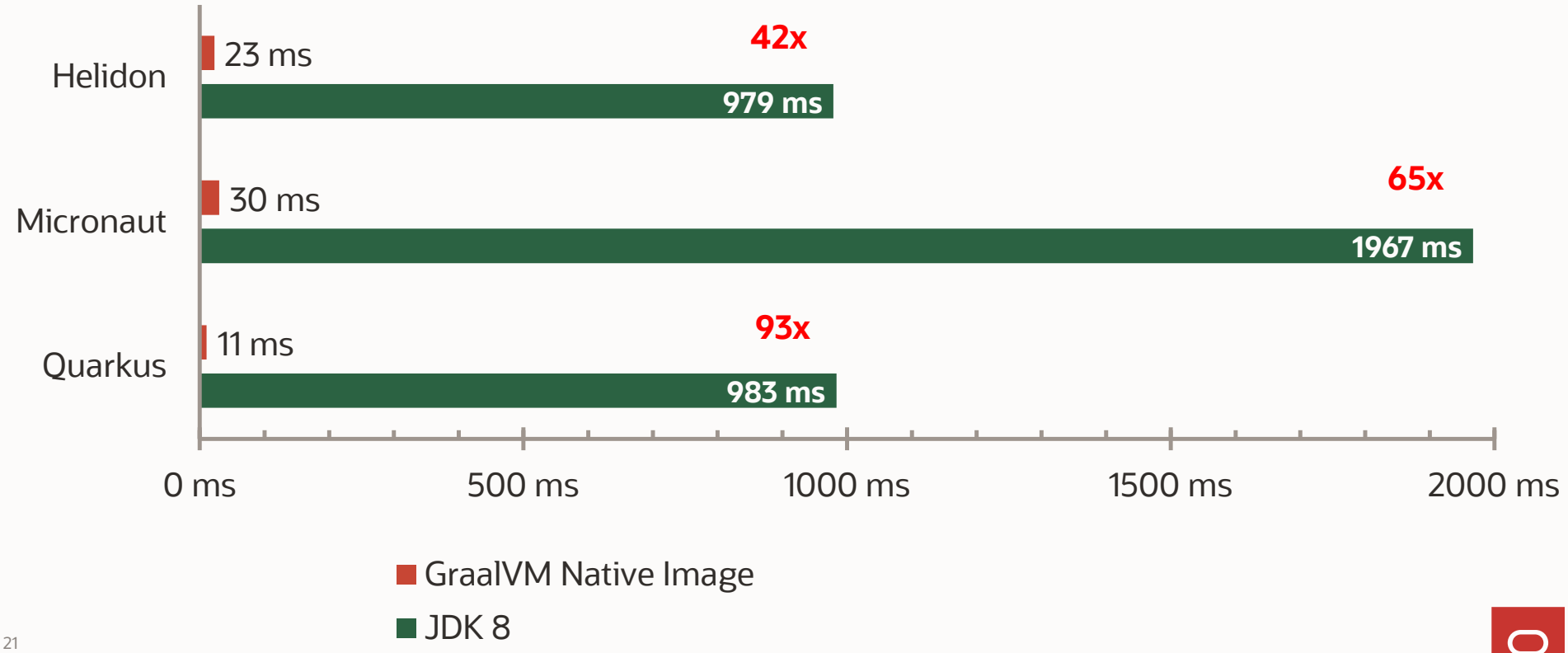
Supported by leading frameworks

GraalVM Enterprise Native Image - Spring Native

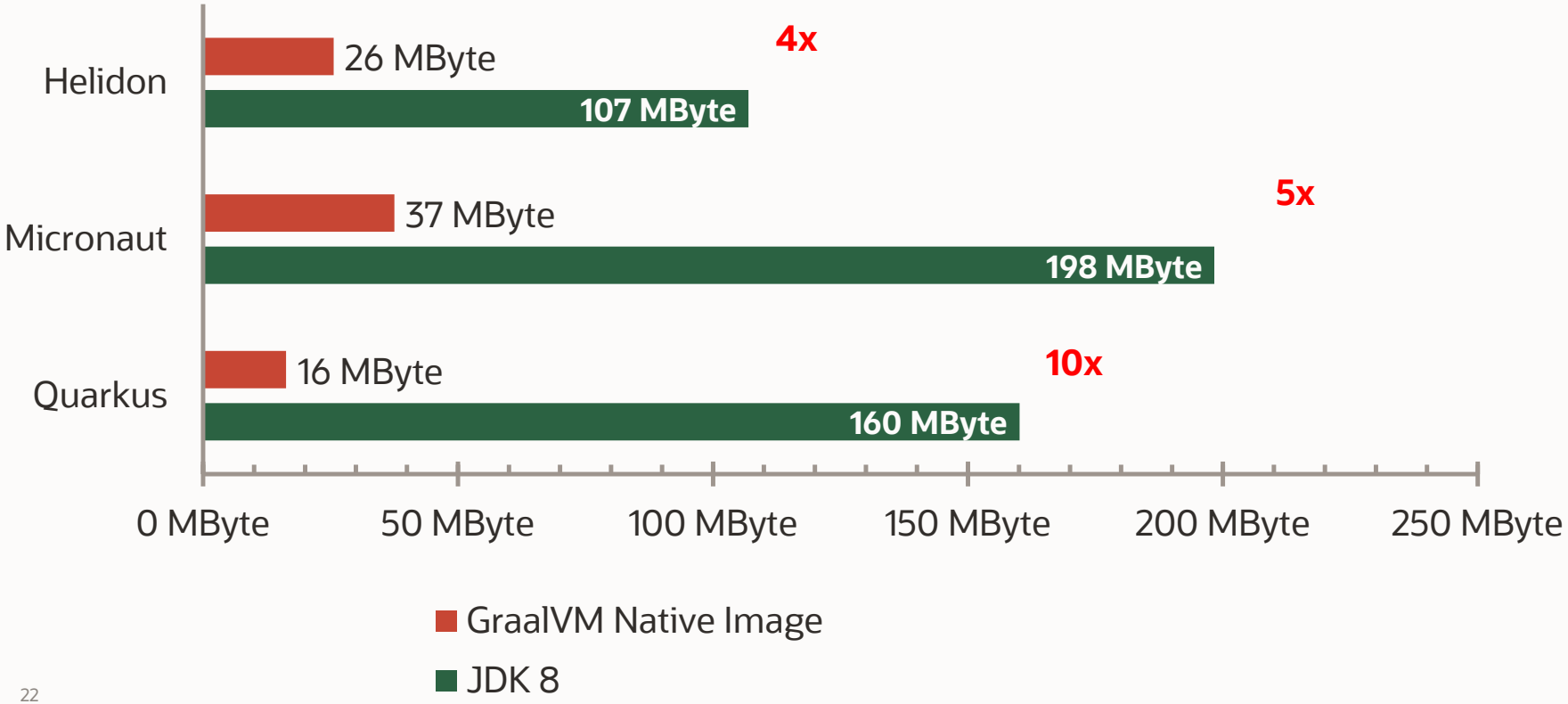
- Which version of Spring Boot is certified or official supported with GraalVM EE 21 native image?
  - We don't currently offer certification of Spring Boot but we are discussing it.
  - But Spring will declare Spring Native 1.0, which is essentially "certification" for Native Image.
- When do we expect the declaration of Spring Native 1.0, which is the essentially "certification" for the GraalVM native image?
  - Spring Native beta 0.10.0, based on GraalVM 21.1.0, will support Spring Boot 2.5, etc.
  - Spring Native 0.11.0-M1 available for early adopters
    - https://github.com/spring-projects-experimental/spring-native/milestone/15
    - Milestone 0.11.0-M1 closed on 5th of October 2021 which is 100% complete.
      - It introduces a new AOT engine which generates a programmatic application context at build-time, allowing between 17% and 25% of memory footprint reduction.
    - Milestone 0.11.0-M2 spring-builds released on 3rd of November 2021
      - Introduces GraalVM 21.3.0 and Java 17 support.
    - Milestone 0.11.x ..
      - AOT ApplicationContext transformations, Spring Boot 2.6.0
    - Milestone 0.12.0 ..
      - https://github.com/spring-projects-experimental/spring-native/milestone/32
      - Upgrade to Spring Boot 2.7.0
      - .. GraalVM 22.1

Spring Native should be GA in the release of Spring Boot 3.0 - tba SpringOne

Cloud Services – Startup Time



Cloud Services – Memory Footprint



GraalVM Native Image – Capabilities

- GraalVM Native Image is an ahead-of-time compilation technology that generates native platform executables
- Native executables are ideal for containers and cloud deployments as they are small, start very fast, and require significantly less CPU and memory
- Deploy native executables on distroless and even Scratch container images for reduced size and improved security
- With profile-guided optimization and the G1 garbage collector, native executables built with GraalVM Native Image can achieve peak throughput on par with the JVM
- GraalVM Native Image enjoys significant adoption with support from leading Java frameworks such as Spring Boot, Micronaut, Quarkus, Gluon Substrate, others

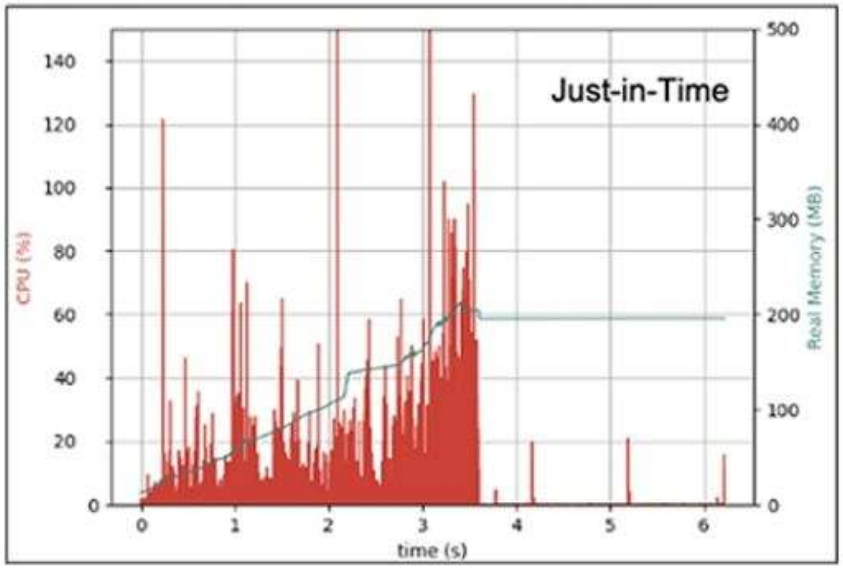
GraalVM - Startup Time Effect of JIT and Native Image Modes

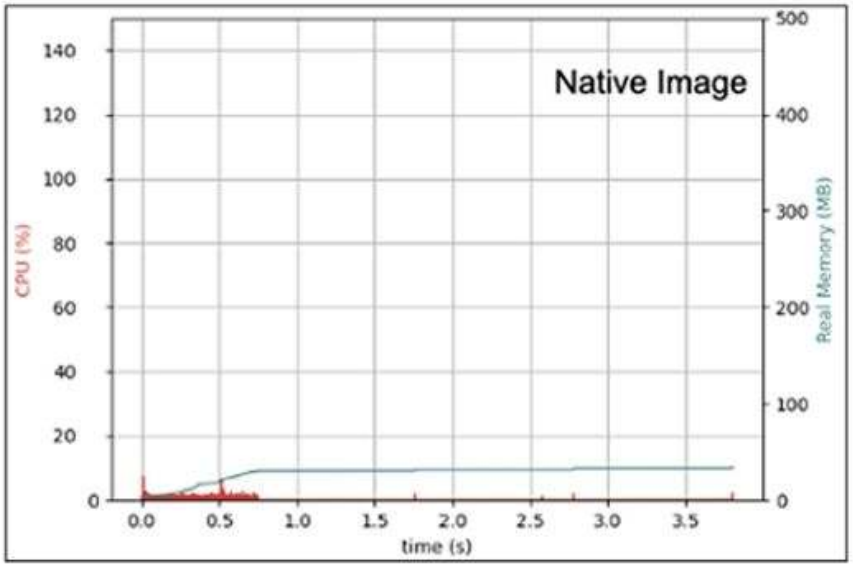
JIT	AOT
• Operating system loads the JVM executable • VM loads classes from the file system • Bytecode is verified • Bytecode interpretation starts • Static initializers run • First-tier compilation (C1) • Profiling metrics gathered • ... (after some time) • Second-tier compilation (C2/ Graal compiler) • Finally running with optimized machine code	• Operating system loads executable with prepared heap • Application starts immediately with optimized machine code

GraalVM - Memory and CPU Usage in JIT and Native Image Modes

Memory efficiency

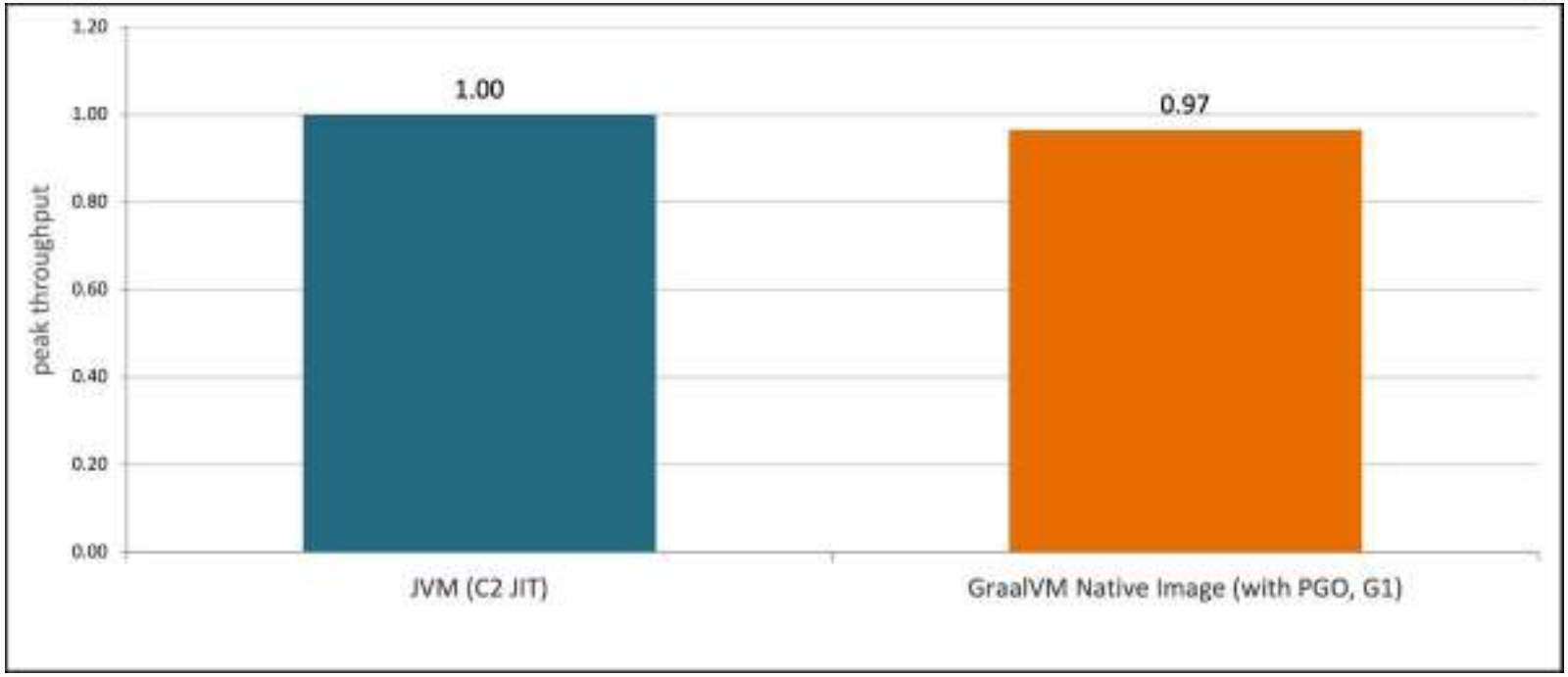
- A native executable requires neither the JVM and its JIT compilation infrastructure nor memory for compiled code, profile data, and bytecode caches.
- All it needs is memory for the executable and the application data.





GraalVM PGO and the G1 GC - Native executables achieve peak performance on par with the JVM

Geomean of Renaissance and DaCapo Benchmarks



With these options, you can maximize every performance dimension of your application with Native Image: startup time, memory efficiency, and peak throughput

GraalVM Native Image – Thoughts & ideas for new features and improvements

- Supporting more platforms
- Simplifying configuration and compatibility for Java libraries
- Continuing with peak performance improvements
- Keep working with Java framework teams to leverage all Native Image features, develop new ones, improve performance, and ensure a great developer experience
- Introducing a faster development compilation mode
- Supporting virtual threads from Project Loom
- IDE support for Native Image configuration and agent-based configuration
- Further improving GC performance and adding new GC implementations

GraalVM 22.2 – Smaller GraalVM JDK base distribution

- Runtime size impacts download times and the developer experience
- Starting with GraalVM 22.2, the base GraalVM JDK is more modular and no longer includes the JavaScript runtime, the LLVM runtime, or VisualVM
- To install those components, use gu install js, gu install llvm, or gu install visualvm in the same way that you already install Native Image, Python, Ruby, or other GraalVM components
  - ➢ This means the base GraalVM JDK download is much smaller
- If you are using GraalVM to run Java applications on the JVM or using Native Image, there is no change to the way you set up GraalVM and run your applications, except for significantly smaller JDK downloads
- Comparison of 22.1 versus 22.2 for the JDK 17 artifacts:

GraalVM version and platform	JDK size in 22.1	JDK size in 22.2	Difference
GraalVM Community / Java 17 / Linux AMD64	431 MB	251 MB	-42%
GraalVM Community / Java 17 / Darwin AMD64	425 MB	247 MB	-42%
GraalVM Enterprise / Java 17 / Darwin AMD64	495 MB	271 MB	-45%

Changes intend to make CI/CD setups more efficient and improve developer experience

GraalVM 22.2 – Making third-party libraries support Native Image

- During the build process, Native Image compiles only the code that is reachable from your application's main entry point
- This approach optimizes for the best startup performance and usage of resources, but poses challenges for dynamic Java features, such as reflection and serialization
- If some element of your application is not reachable, it won't be included in the native executable
  - But this can lead to failures when the element is required and accessed via reflection at run time
  - To include elements that Native Image deems unreachable, you must provide Native Image with metadata (configuration information)
  - The metadata can be automatically provided by a framework or created manually, for example, with the help of the tracing agent
  - To simplify this task further, we are introducing the GraalVM Reachability Metadata Repository — a centralized repository that library and framework maintainers (as well as Native Image users) can use to share metadata for Native Image
- The metadata repository is integrated with the GraalVM Native Build Tools
  - For example, to enable automatic use of the metadata repository in a Gradle project,

    add the following to your `build.gradle` config:

```
graalvmNative {
metadataRepository {
enabled = true
 }
}
```


GraalVM 22.2 – Smaller memory footprint of Native Image builds

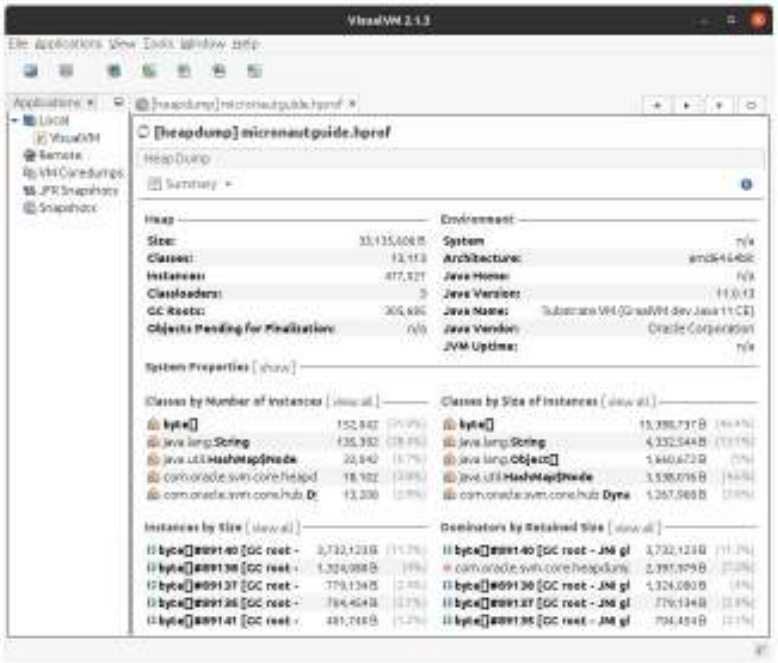
- Several improvements in internal data structures lead to, significantly less memory is required by Native Image when it builds a native executable
- The reduction of memory usage is particularly beneficial in memory-constrained environments, such as cloud services and GitHub Actions
- Starting with release 22.2, the Native Image tool can successfully build many larger native executables with only 2 GB of Java heap

GraalVM 22.2 – Generating heap dumps in Native Image

- Dumping the heap of native executables at runtime is now a supported feature in GraalVM Community
- There are different ways to dump the heap, including **`-XX:+DumpHeapAndExit`** – a new command-line option that will dump the initial heap of a native executable

```
$GRAALVM_HOME/bin/native-image HelloWorld -H:+AllowVMInspection
./helloworld -XX:+DumpHeapAndExit
Heap dump created at '/path/to/helloworld.hprof'.
```

- The heap dumps can be analyzed with Java heap dump tools such as VisualVM



GraalVM 22.2 – Other updates in Native Image

- Added support for Software Bill of Materials (SBOM) to GraalVM Enterprise Native Image. SBOM is a list of components used in a software artifact
  - GraalVM Native Image can now optionally include a SBOM into a native executable to aid vulnerability scanners
  - Currently we support the CycloneDX format, which could be enabled by using the -H:IncludeSBOM=cyclonedx command-line option during compilation
  - After embedding the compressed SBOM into the executable, use the Native Image Inspection tool (available in GraalVM Enterprise) to extract the compressed SBOM with this command:

```
$GRAALVM_HOME/bin/native-image-inspect — sbom <path_to_binary>
```

- Improved debugging support on Linux
  - The Dwarf information now contains parameters and local variables — thanks to Red Hat for this contribution
- New experimental support for Native Image debugging in IntelliJ IDEA 2022.2 EAP 5

GraalVM 22.2 – Compiler updates (1)

➢ Reduced memory usage in JIT mode

- The GraalVM compiler now uses memory more efficiently at runtime
- When an application warms up and reaches a stable state with few or no compilations needed, the Graal compiler releases the unused memory back to the system
- Try it on your own workload to evaluate memory impact by running the following command:

```
ps aux --sort --rss
```

➢ New strip mining optimization for counted loops

- Strip mining converts a single long-running loop into a nested loop where the inner body runs for a bounded time
- This enables putting a safepoint in the outer loop to reduce the overhead of safepoint polling
- By choosing the right value for the outer loop stride, we ensure reasonable time-to-safepoint latency
- The latter is particularly important for low-pause-time garbage collectors such as ZGC and Shenandoah

GraalVM 22.2 – Compiler updates (2)

➢ New strip mining optimization for counted loops - example

```
for (long i = init; i < limit; i += stride) {
        use(i);
}
```

becomes

```
final long stripMax = (long) CountedStripMiningInnerLoopTrips;
for (long i = init; i < limit;) {
        long innerTrips = i < limit - stripMax ? stripMax : limit - i;
        long i_ = i;
        for (long j = 0; j |<| innerTrips; j++) {
                use(i_);
                i_ += stride;
        }
        i = i_;
}
```

- As a result, we see around 20% increase in speed on workloads that exercise long range checks, such as code using the foreign-memory access API
- This optimization is also beneficial for Truffle languages
- In GraalVM 22.2 this optimization is experimental — enable it with the command-line option `-Dgraal.StripMineCountedLoops=true`

GraalVM 22.2 – GraalVM Enterprise for Apple Silicon and new components for GraalVM Community

- **Apple Silicon users can now develop applications using GraalVM Enterprise**
  - You can get the builds
  - Support for Apple silicon was one of the most requested features on GraalVM's GitHub, and now GraalVM Enterprise users can benefit from it too
  - Note that support is experimental in this release
  - We would appreciate your issue reports and feedback
- **In addition to the JDK and Native Image, we also added Apple Silicon support for a number of components in both GraalVM Community and Enterprise Edition:**
  - JavaScript
  - the LLVM toolchain and runtime
  - Ruby
  - Java on Truffle
  - Webassembly

GraalVM 22.2 – GraalPython: faster startup and extended library support

- Added an experimental bytecode interpreter to GraalPython for faster startup and better interpreter performance
- It's not enabled by default in this release — enable it by using the command-line option

```
--python.EnableBytecodeInterpreter
```

- Additionally, we updated to HPy version 0.0.4, which adds support for the (completed) HPy port of Kiwi, and the in-progress ports of Matplotlib and NumPy

GraalVM 22.2 – Improved interoperability in GraalJS

- Starting with 22.2, objects from other languages are assigned a proper JavaScript prototype by default
- This feature, previously experimental, increases the portability of code by letting foreign objects appear as arrays, functions, and other types from JavaScript

GraalVM 22.2 – Community Contributions

- Oracle Labs are grateful to the community and our partners in the ecosystem for working with us on this release
- There were many helpful reports and contributions on GitHub, and by sharing feedback and helping other community members on GraalVM's platforms, we can collectively make GraalVM better for everyone
- In particular:
  - Improved debugging support on Linux was contributed by Red Hat
  - The metadata repository was built as a collaboration between the GraalVM, Micronaut, Quarkus, and the Spring Boot teams

GraalVM Project Advisory Board .. and the Members

Advisory committee, composed of key project contributors and partners in the ecosystem with a shared goal of driving the GraalVM project forward

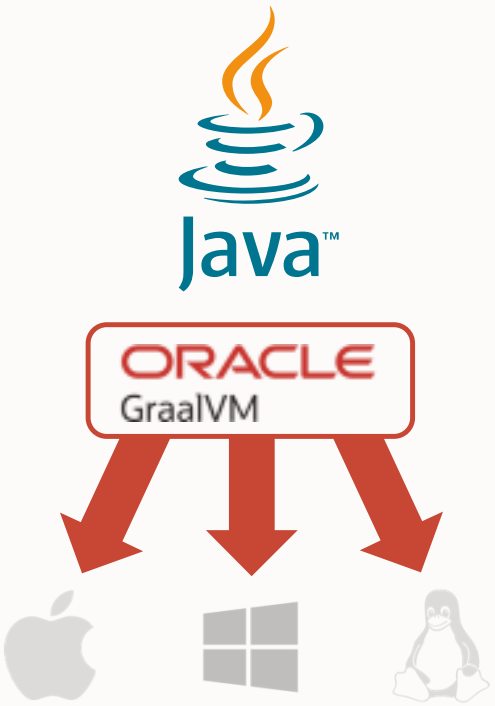
- *Such organizations can nominate their Representative for the Board*
- *The Representative will act as the main contact point for all project-related matters associated with their company*

- Aleksei Voitylov, BellSoft. Works on bringing musl libc support to GraalVM and enhancing GraalVM on ARM platforms.
- Bruno Caballero, **Microdoc**. **Work on GraalVM integrations in the embedded space**.
- Chris Seaton, **Shopify**. Contributors to TruffleRuby – GraalVM Ruby implementation.
- James Kleeh, **Object Computing, Inc. Developers of Micronaut** – a framework for building microservice and serverless applications, integrated with GraalVM.
- Johan Vos, **Gluon**. Work on the JavaFX and mobile/embedded platform support for GraalVM native images.
- Max Rydahl Andersen, **Red Hat**. Developer on Quarkus - A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards.
- Michael Simons, **Neo4j**. Integrate with GraalVM to support polyglot dynamic languages for user-defined-procedures in Neo4j, a JVM-based graph database.
- Paul Hohensee, **Amazon**. Interested in GraalVM Community Edition, GraalVM Native Image, and AWS Lambda on GraalVM.
- San-Hong Li, **Alibaba**. Contribute to the project and share their experience with the community.
- Sébastien Deleuze, **VMware. Spring Framework committer**, works on Spring Native for GraalVM.
- **Thomas Wuerthinger, Oracle. Key developers and maintainers of GraalVM and related projects.**
- Uma Srinivasan, **Twitter. Run GraalVM Community Edition in production** on a large scale system and share their experience with the community.
- Xiaohong Gong, **Arm Technology China**. Works on GraalVM Compiler Optimizations On AArch64.

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