ORACLE

ORACLE

GraalVM Native Image

Past, Present, and Future

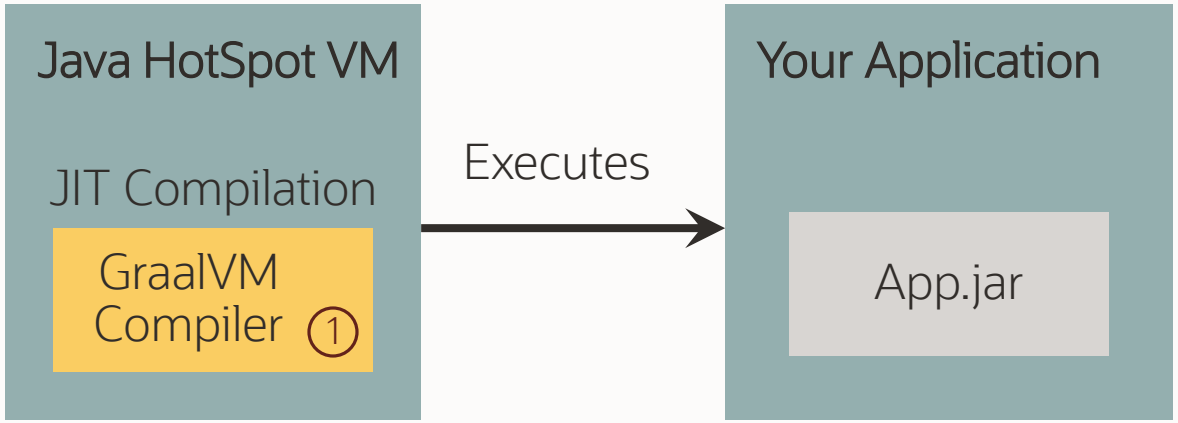
Christian Wimmer
GraalVM Native Image Project Lead
christian.wimmer@oracle.com

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.

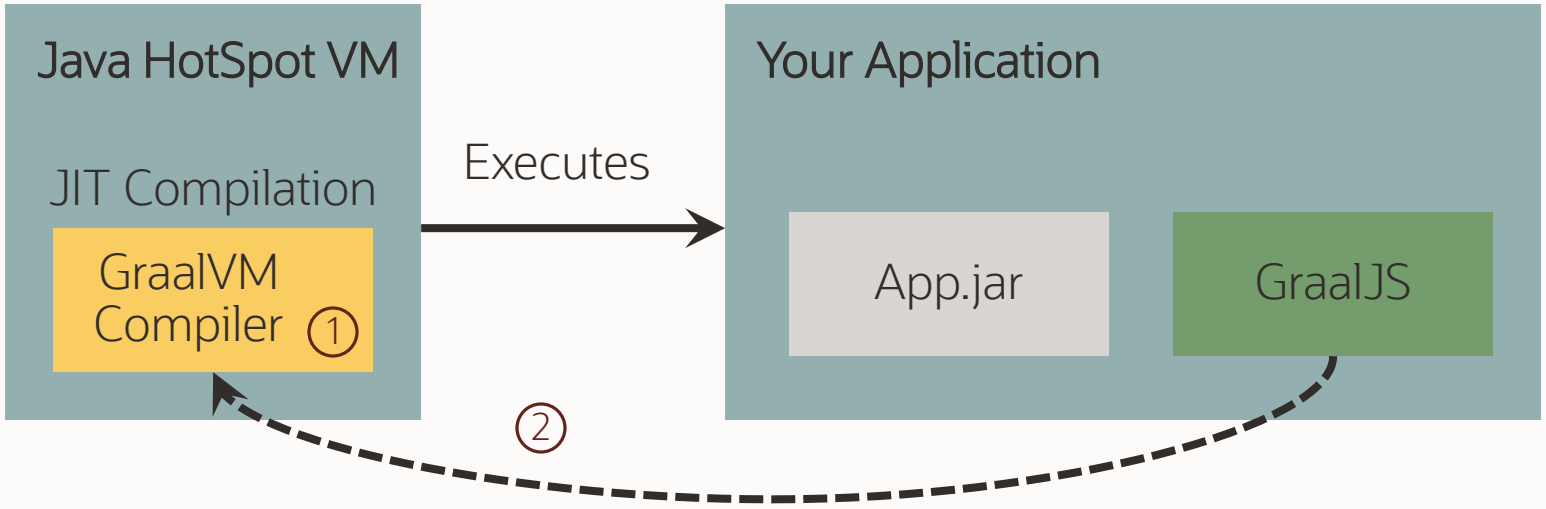
GraalVM Native Image technology (including Substrate VM) is Early Adopter technology. It is available only under an early adopter license and remains subject to potentially significant further changes, compatibility testing and certification.

One Compiler, Many Configurations



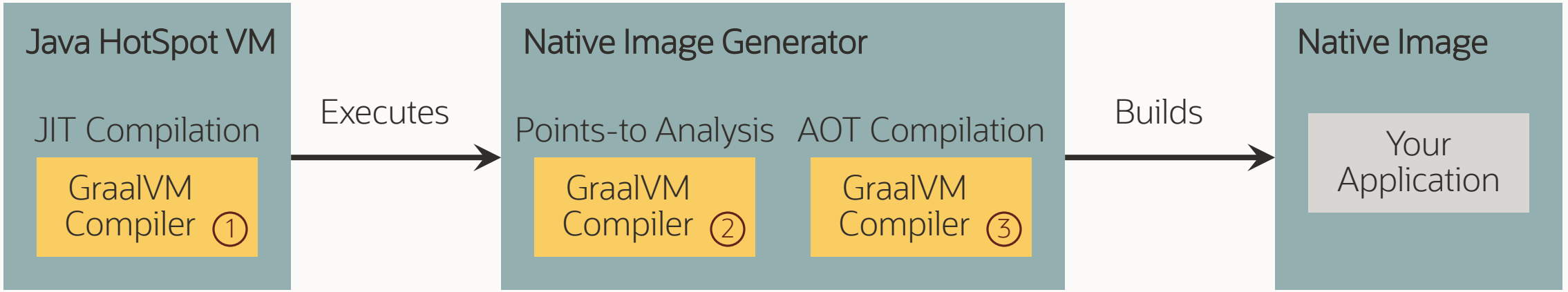
1 Compiler configured for just-in-time compilation inside the Java HotSpot VM

One Compiler, Many Configurations



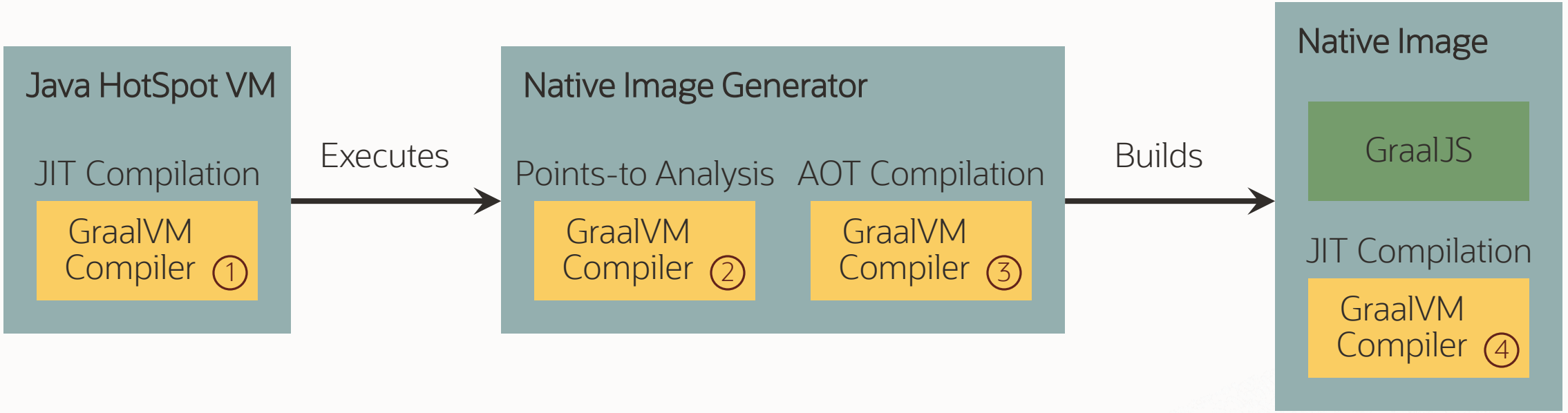
1. Compiler configured for just-in-time compilation inside the Java HotSpot VM
2. Compiler also used for just-in-time compilation of JavaScript code

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



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

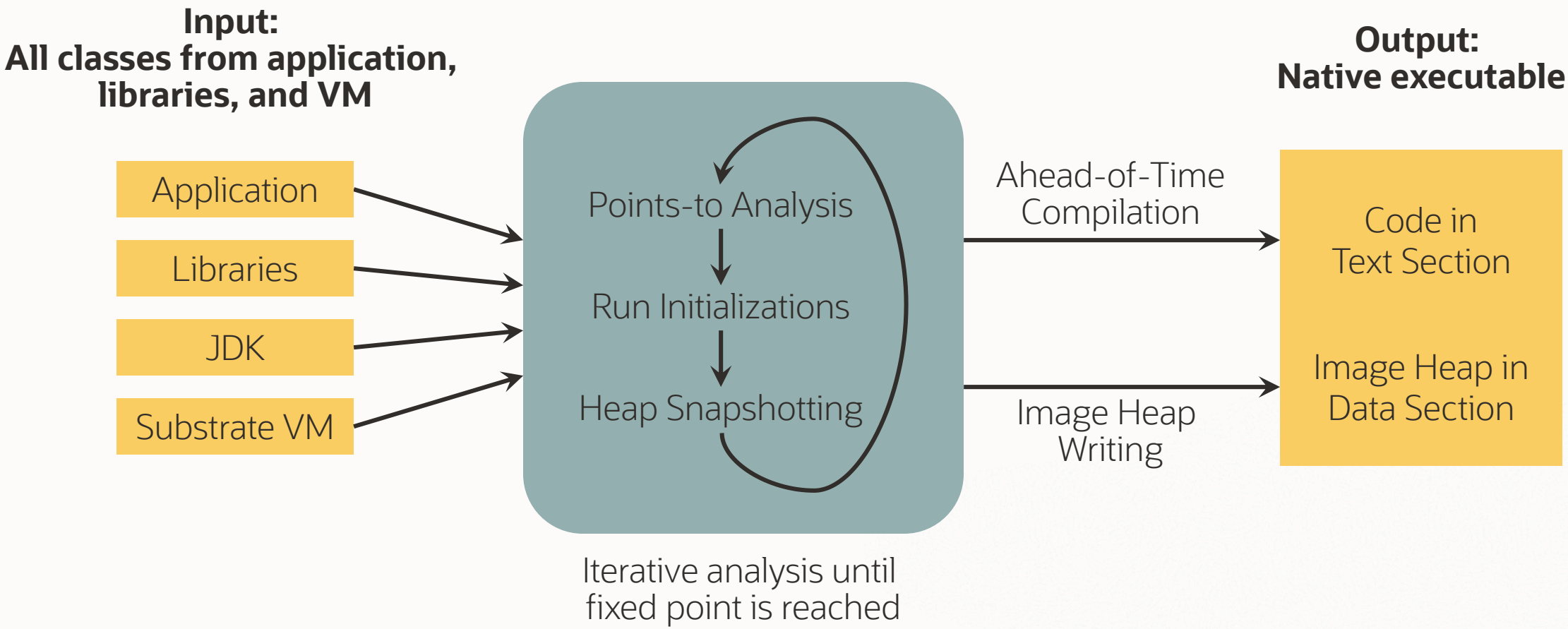
Initial Design Decisions

- Original use case: Oracle Database Multilingual Engine
  - Integration of the GraalVM language framework (Truffle) into the Oracle database
- Assumptions
  - All code apart from the JDK is developed by same team and can adapt to restrictions
  - One production OS: the OS abstraction layer of the database
    - Linux and MacOS support for development
    - Only basic file system and network support is necessary.
  - No Java reflection or JNI support is needed
  - Fast initialization of a database session and low memory footprint are essential
- Result: a quite restrictive but very well performing system

More Use Cases

- Many other uses cases profit from fast startup and low memory footprint
  - Microservices
  - Command line applications
- “Real-world code” does not follow our initial restrictions
  - How far can we go and lift restrictions without sacrificing the main benefits
- We already finished a lot, but still have more work to do

Native Image: Details



Paper with Details, Examples, Benchmarks

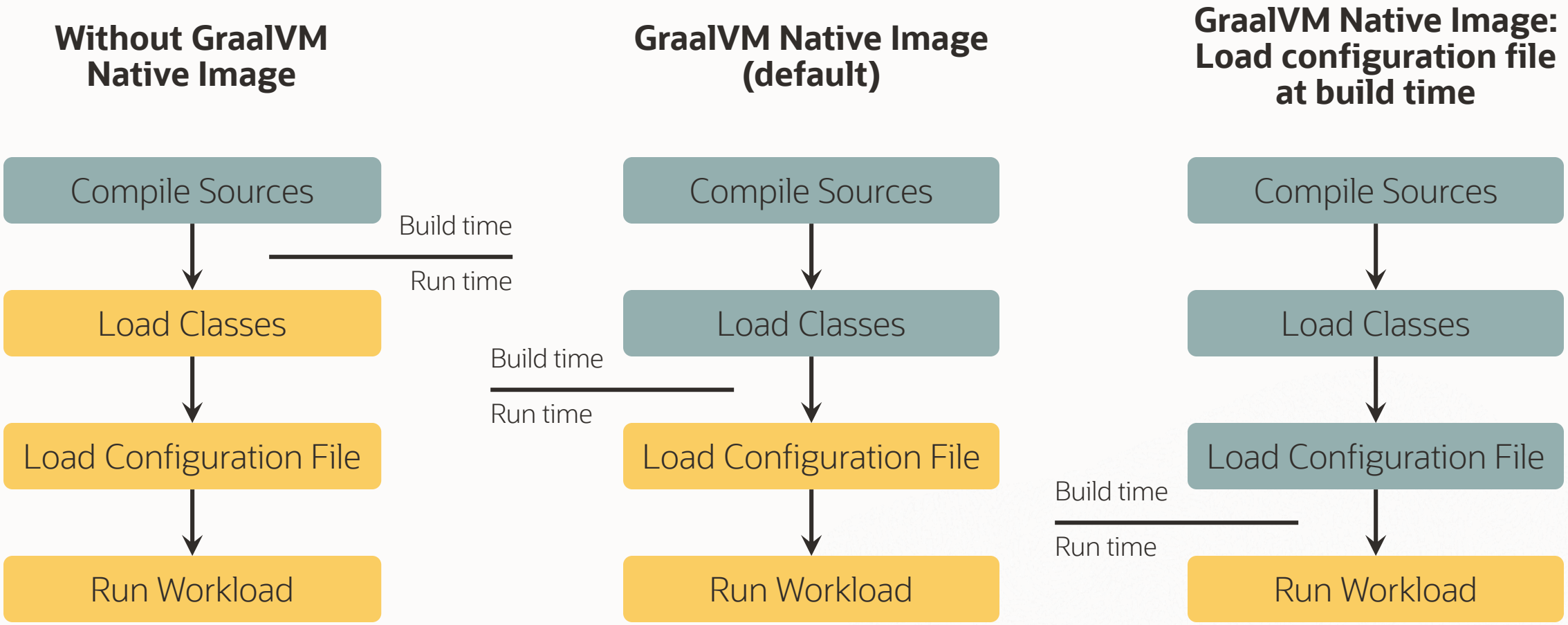
http://www.christianwimmer.at/Publications/Wimmer19a/Wimmer19a.pdf

Initialize Once, Start Fast: Application Initialization at Build Time

CHRISTIAN WIMMER, Oracle Labs, USA
CODRUT STANCU, Oracle Labs, USA
PETER HOFER, Oracle Labs, Austria
VOJIN JOVANOVIC, Oracle Labs, Switzerland
PAUL WÖGERER, Oracle Labs, Austria
PETER B. KESSLER, Oracle Labs, USA
OLEG PLISS, Oracle Labs, USA
THOMAS WÜRTHINGER, Oracle Labs, Switzerland

Arbitrary program extension at run time in language-based VMs, e.g., Java's dynamic class loading, comes at a startup cost: high memory footprint and slow warmup. Cloud computing amplifies the startup overhead. Microservices and serverless cloud functions lead to small, self-contained applications that are started often. Slow startup and high memory footprint directly affect the cloud hosting costs, and slow startup can also break service-level agreements. Many applications are limited to a prescribed set of pre-tested classes, i.e., use a closed-world assumption at deployment time. For such Java applications, GraalVM Native Image offers fast startup and stable performance.

Benefits of the Image Heap



Native Image Support in Libraries

- Configuration options should be provided by libraries
  - Library and framework developers know best what their code needs
- Configuration files in META-INF/native-image are automatically picked up
  - native-image.properties for command line options like class initialization options
  - reflect-config.json, jni-config.json, … for configuration files created by tracing agent
  - Use subdirectories to make files composeable (inspired by Maven)
    - META-INF/native-image/your.group.id/artifactId/

Major Improvements in 2019

Class initialization
- Initialization at run time is the default for application classes (GraalVM 19.0)
- -H:+TraceClassInitialization to diagnose class initialization at build time (GraalVM 19.2)
- Fixing corner cases, e.g., lambda interface initialization (GraalVM 19.3)

Reflection and JNI
- Agent to record usage and produce configuration files for native images
- java -agentlib:native-image-agent=config-output-dir=META-INF/native-image ...

Platform support
- Statically link in C code of JDK instead of manual rewrite to Java (GraalVM 19.3)
- A bit like changing out the engine while flying
- Eliminates 14,000 lines of code, plus 12,000 lines of comments (original C code in comments)
- Enables JDK 11, JDK 14+, Windows support

Native Image API

Project org.graalvm.nativeimage contains the stable API
- We will try our best to only deprecate but never remove
- Therefore we are careful what we add

All other projects are internal API and subject to change at any time
- Use at your own risk
- We want to know what you are using and why so that we can expand the API

Examples that are not in the API
- Substitutions: very powerful but impossible to make safe. Modify the original code if possible

Things Not To Do or Use

Do not use internal iteration of all classes on the class path
- FeatureImpl. findSubclasses, findAnnotatedClasses, …
- We are looking for ways to remove the “classlist” phase because it is expensive

Do not use --report-unsupported-elements-at-runtime for production
- Intended to jump-start a project and see if Native Image is feasible
- Hides many issues at build time that can then fail at run time

Planned Features: Platforms and Performance

- Windows support
- AArch64 support

- Improve peak performance: more compiler optimizations for AOT compilation
- Low-latency GC: similar approach to G1
- Improvement to the single-threaded high-throughput GC
  - Tenuring, mark-and-compact for old generation

- Sampling-based profiling for profile guided optimizations
  - Always-on low-overhead profiling across all deployments of an application

- Improve memory footprint and time of native image generation
  - At the cost of analysis precision?

Planned Features

- Class initialization: only initialize parts of the JDK at build time that is explicitly tested
  - AWT, Swing, SQL, … currently initialized at build time but known to not work
- Make --allow-incomplete-classpath the default (= accept reality)

- Support for Java module system on JDK 11 (currently there is no module path for the image generator and only stub module information at run time)
- Support for MethodHandle and invokedynamic: will be slow (no JIT compilation) but complete

Eventually Planned Features

- Support for Project Panama: new C interface for Java
- Support for Project Loom: continuations for Java

- Full support for localization
  - Currently the Locale is fixed during image generation
  - Resource bundles only available for this one Locale
- Fundamental question: how much should be included in a native image by default
  - All timezones? https://github.com/oracle/graal/issues/1833
  - All charsets?
  - All locales?

- Better support for JSR/RET bytecodes
  - How important is it to support libraries compiled for Java 5?

Community

Please help make libraries ready for Native Image
- Push native-image.properties
- Push changes to make code AOT friendly: reduce reflection usage, improve class initializer, …

Provide feedback for missing API
- Do not rely on internals that are subject to change

Provide feedback about priorities for missing features

Summary

Past
- One use case: Oracle Database Multilingual Engine

Present
- 2019: the year of Java microservice frameworks on GraalVM Native Image
- Micronaut, Quarkus, Helidon, Spring Boot

Future
- Same peak performance and latency as Java HotSpot VM
- Expand completeness based on customer priorities

Thank you