Going Native: Fast and Lightweight Spring Boot Applications with GraalVM

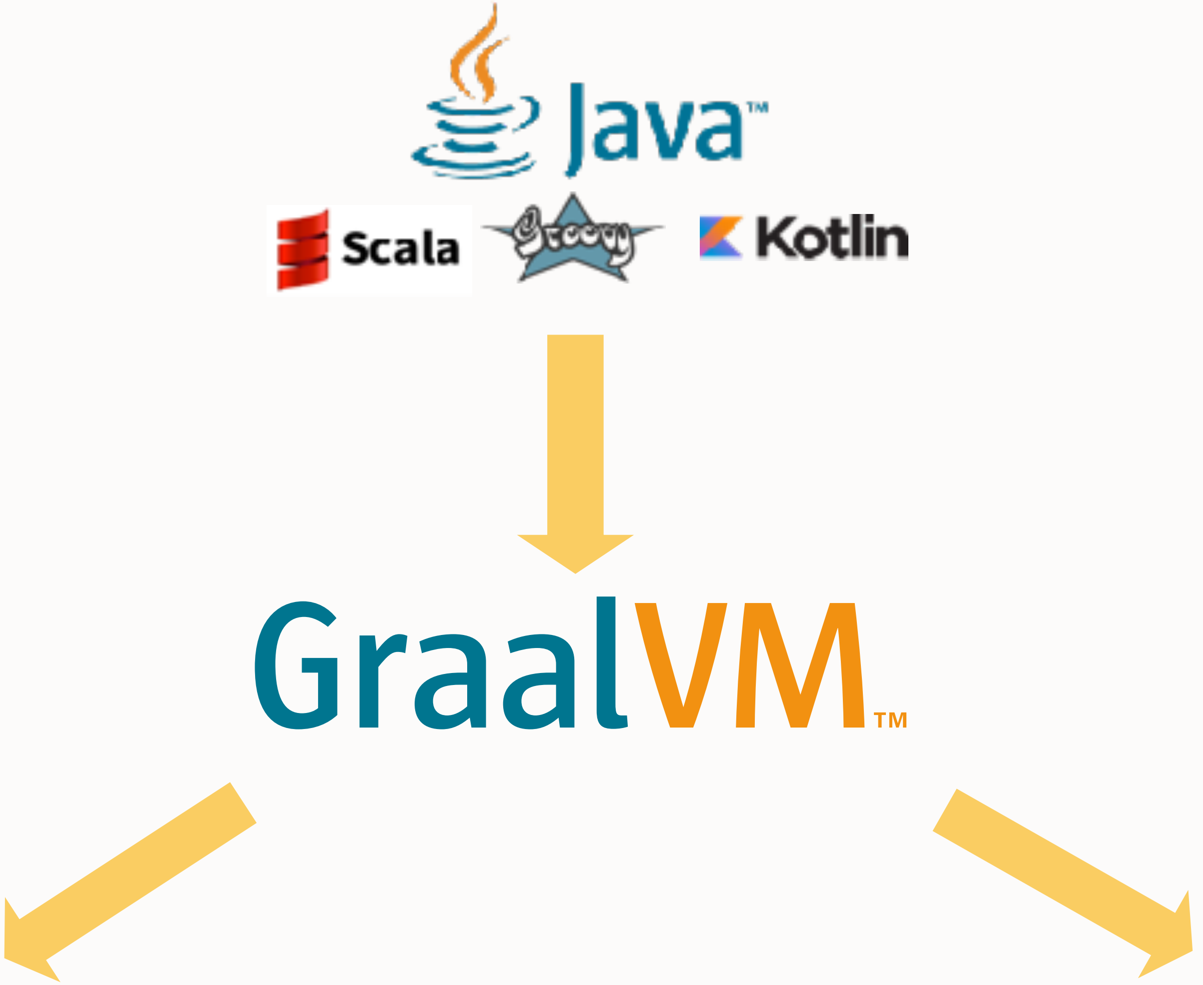
Alina Yurenko

Developer Advocate for GraalVM

Oracle Labs

Spring I/O

Logan Armstrong @ Unsplash

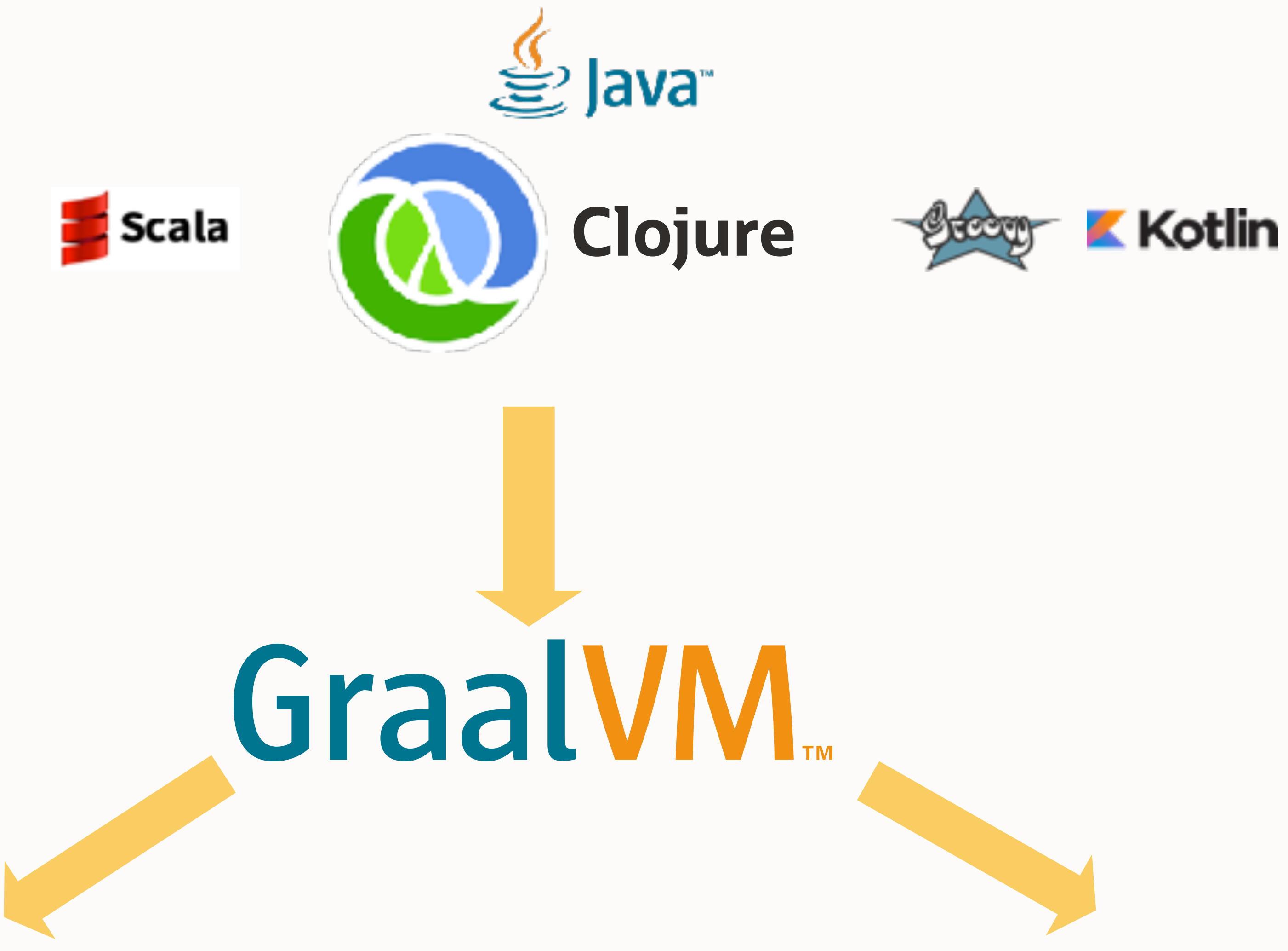


JIT

```
java MyMainClass
```

AOT

```
native-image MyMainClass
./mymainclass
```



JIT

```
java MyMainClass
```

AOT

```
native-image MyMainClass
./mymainclass
```

Native Image deployments

Start Fast

Low Resource Usage

Minimize Vulnerability

Compact Packaging

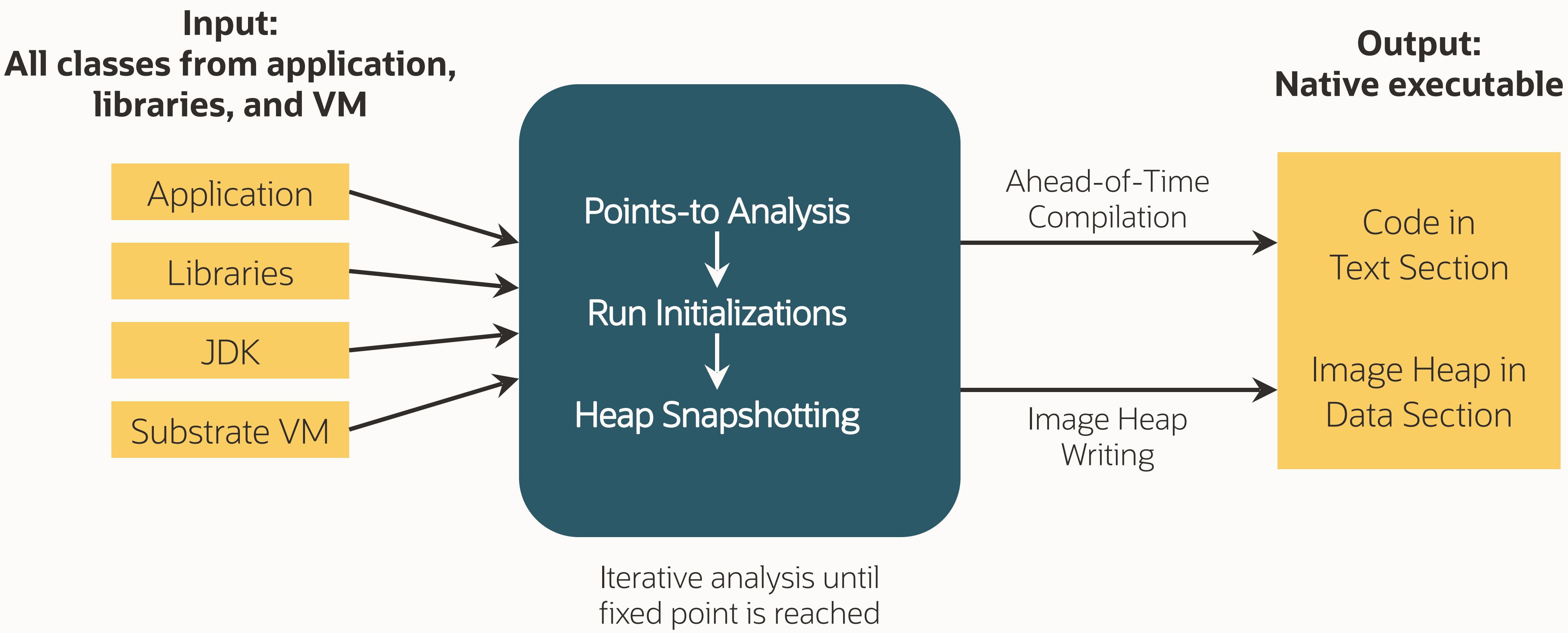
Start Fast

Low Resource Usage

Minimize Vulnerability

Compact Packaging

Native Image Build Process



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

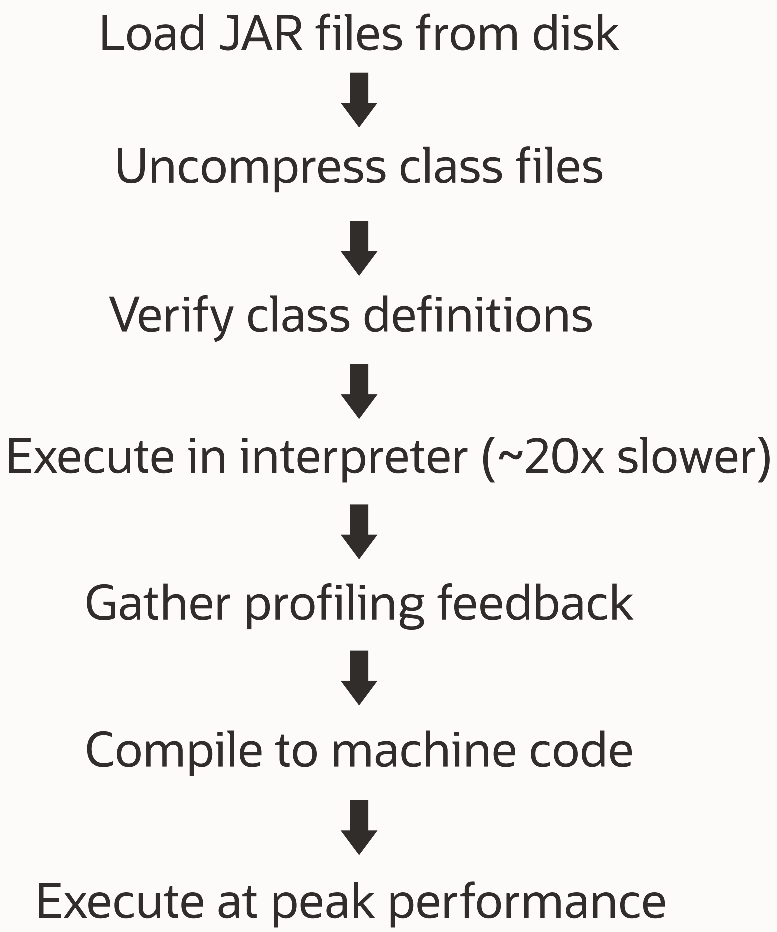
Image Heap Writing

Output:
Native executable

Code in Text Section

Image Heap in Data Section

JIT


Load JAR files from disk

↓

Uncompress class files

↓

Verify class definitions

↓

Execute in interpreter (~20x slower)

↓

Gather profiling feedback

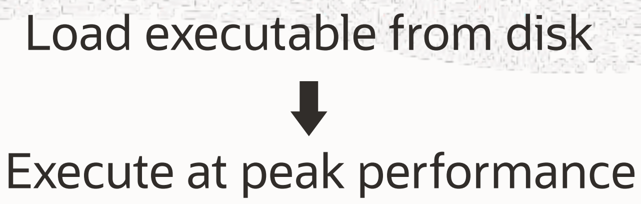
↓

Compile to machine code

↓

Execute at peak performance

AOT


Load executable from disk

↓

Execute at peak performance

Start Fast

Low Resource Usage

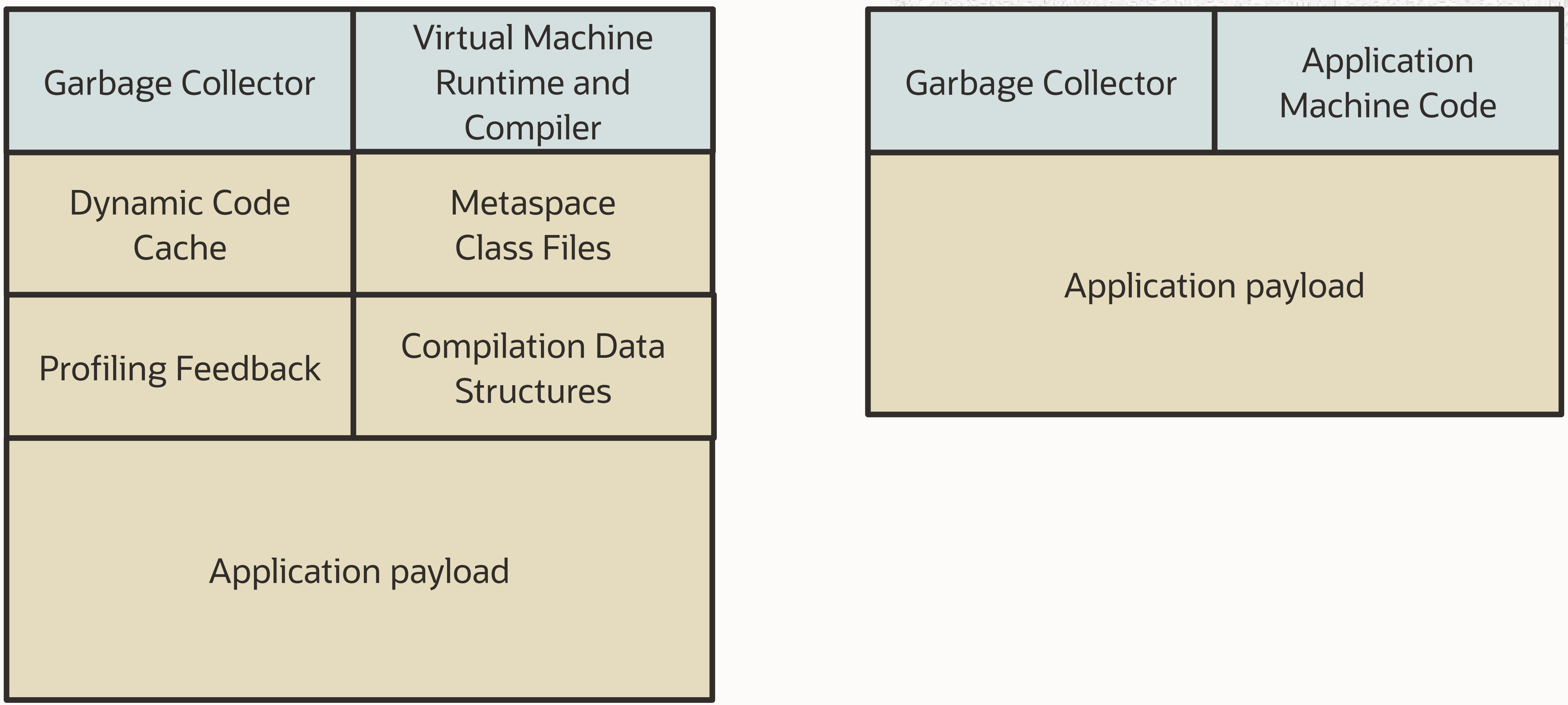
Minimize Vulnerability

Compact Packaging

Memory

JIT

AOT



JIT

Garbage Collector

Virtual Machine Runtime and Compiler

Dynamic Code Cache

Metaspace Class Files

Profiling Feedback

Compilation Data Structures

Application payload

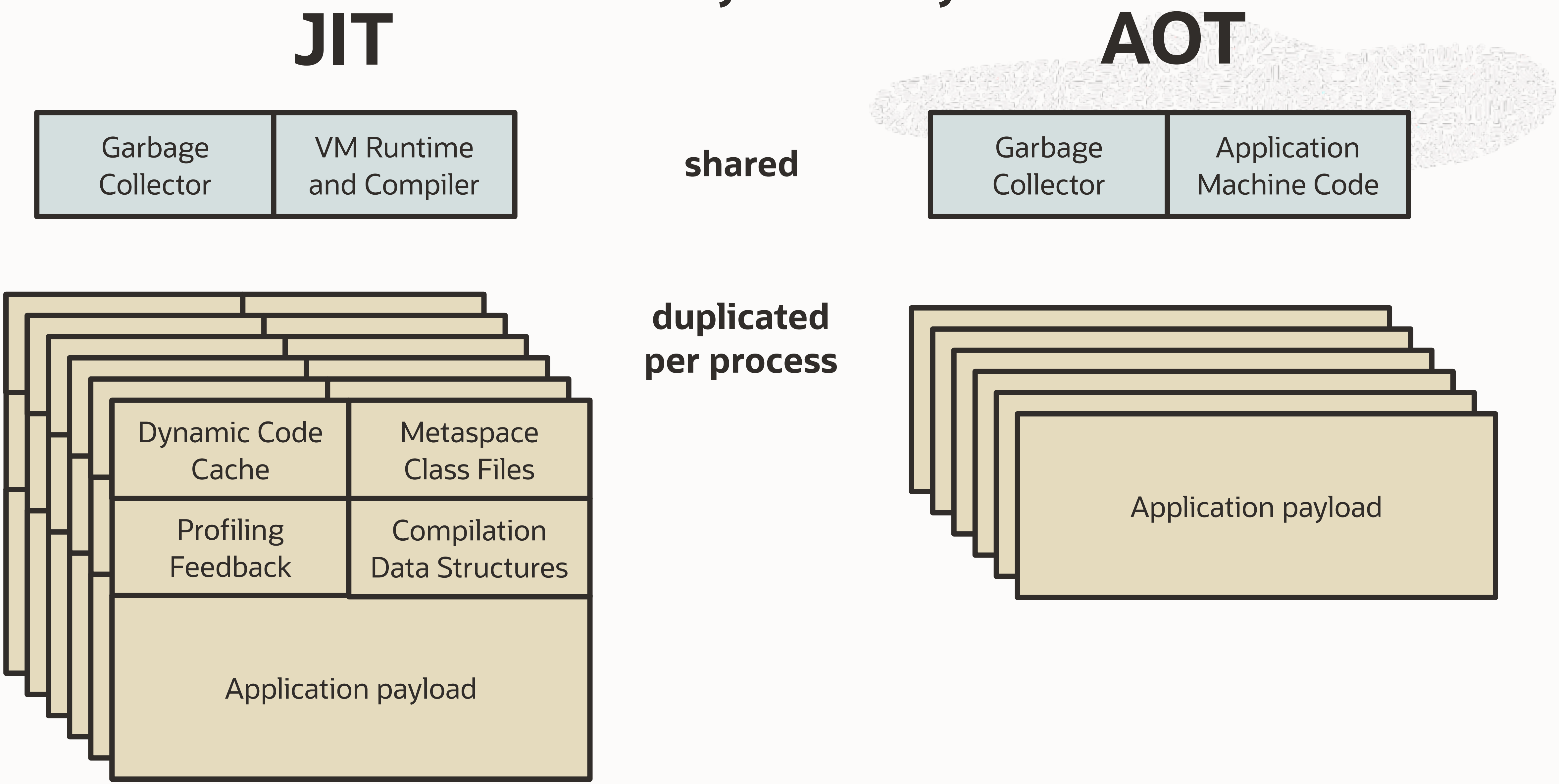
AOT

Garbage Collector

Application Machine Code

Application payload

Memory Scalability



JIT

Garbage Collector

VM Runtime and Compiler

shared

AOT

Garbage Collector

Application Machine Code

duplicated per process

Dynamic Code Cache

Metaspace Class Files

Profiling Feedback

Compilation Data Structures

Application payload

Application payload

Demo: startup and performance

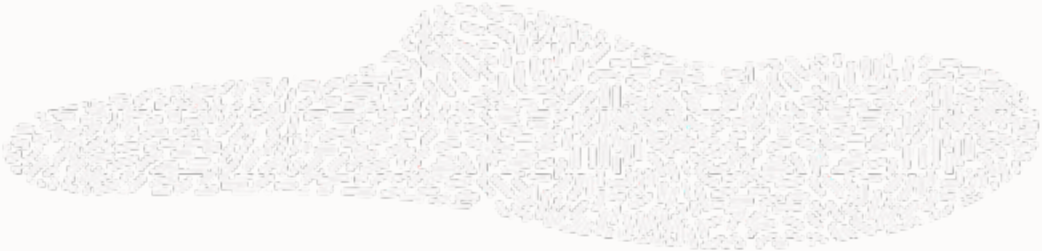
Start Fast

Low Resource Usage

Minimize Vulnerability

Compact Packaging

Reduced Attack Surface



- No new unknown code can be loaded at run time
- Only paths proven reachable by the application are included in the image
- Reflection is disabled by default and needs an explicit include list
- Deserialization only enabled for specified list of classes
- Just-in-time compiler crashes, wrong compilations, or “JIT spraying” to create machine code gadgets are impossible

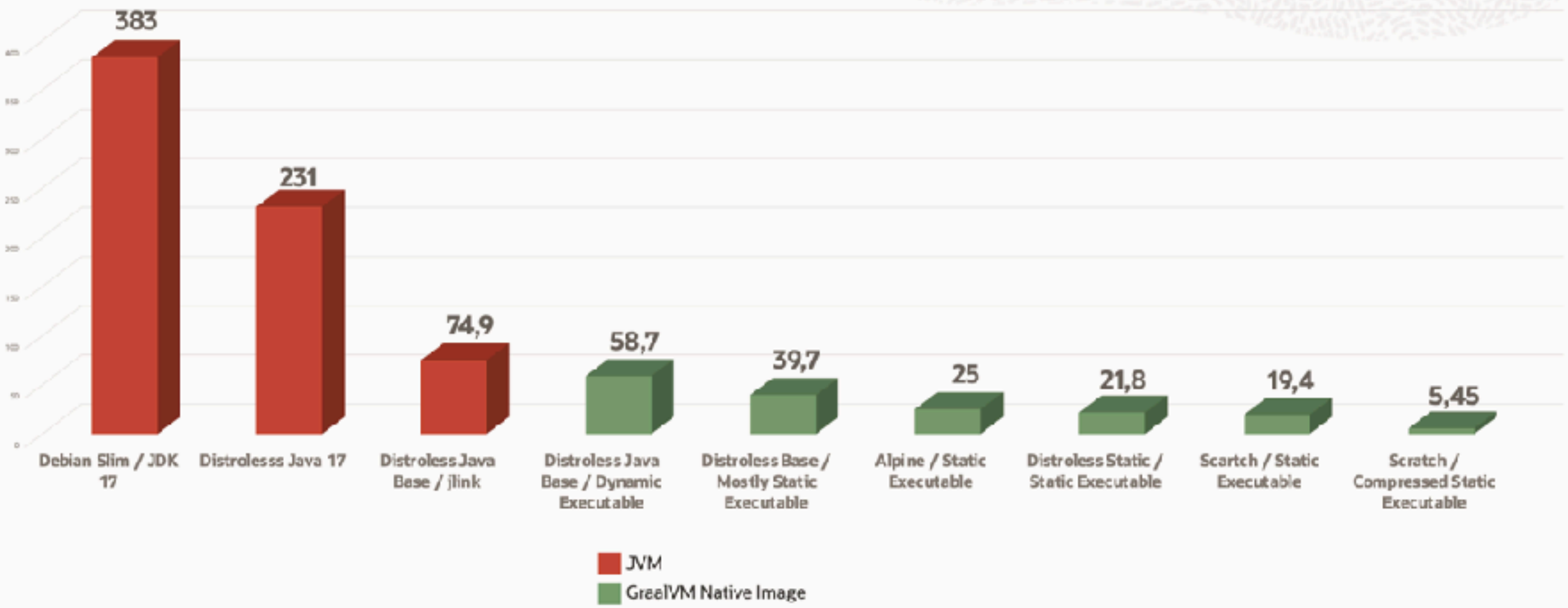
Start Fast

Low Resource Usage

Minimize Vulnerability

Compact Packaging

Lightweight containerized applications



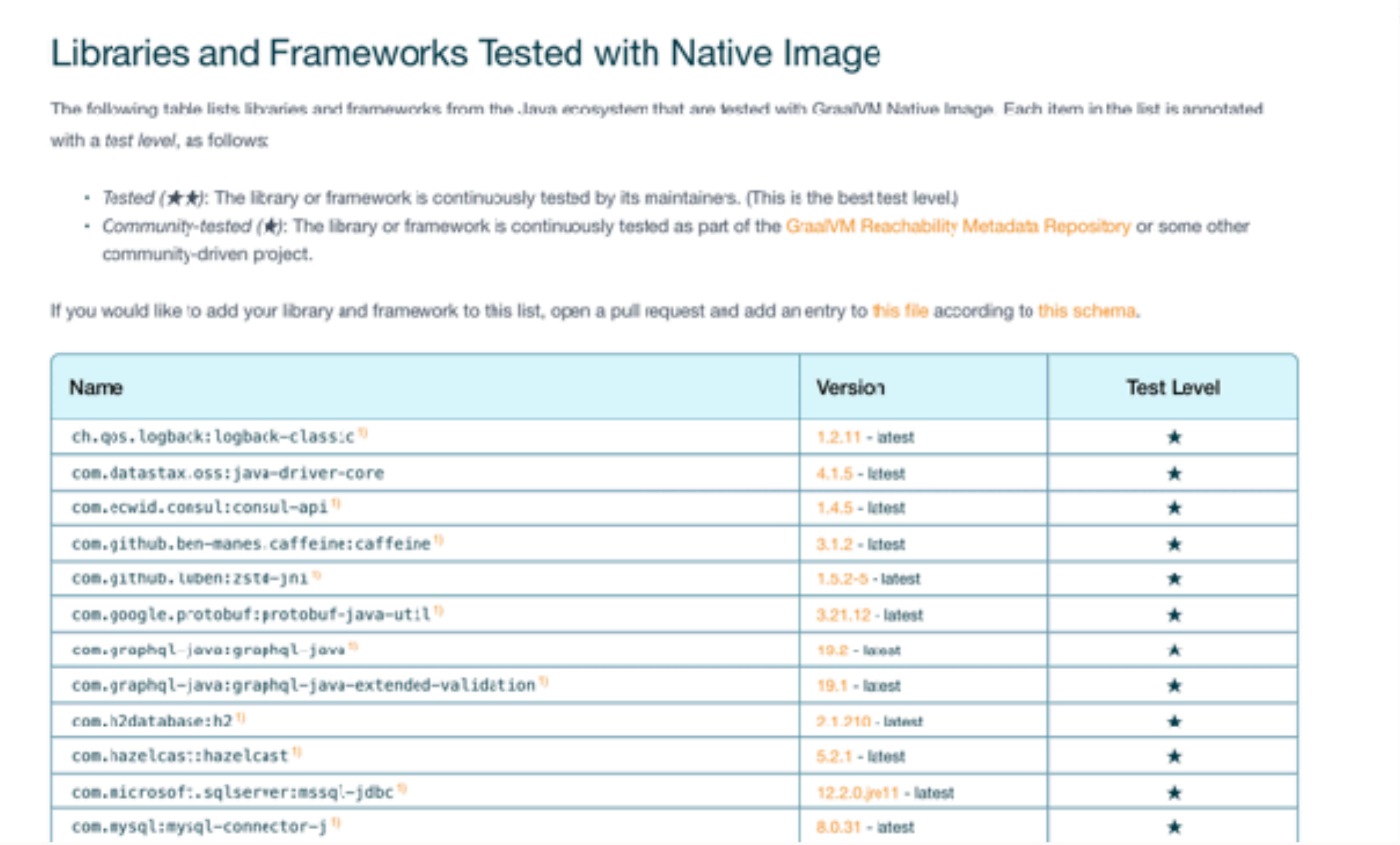
YouTube: A 1.5MB Java Container App? Yes you can! by Shaun Smith

What's the catch?

GraalVM & Reflection?

- GraalVM 🤝 Reflection!
- Native Image tries to resolve the target elements through a static analysis that detects calls to the Reflection API
  - If the analysis can not automatically detect your use of reflection, you might need additional configuration
- Trace reflection, JNI, resource usage on the JVM with the tracing agent
  - Manual adjustment / addition might still be necessary

Reflection in 3rd-party libraries



Libraries and Frameworks Tested with Native Image

The following table lists libraries and frameworks from the Java ecosystem that are tested with GraalVM Native Image. Each item in the list is annotated with a *test level*, as follows:

- *Tested (★★)*: The library or framework is continuously tested by its maintainers. (This is the best test level.)
- *Community-tested (★)*: The library or framework is continuously tested as part of the GraalVM Reachability Metadata Repository or some other community-driven project.

If you would like to add your library and framework to this list, open a pull request and add an entry to this file according to this schema.

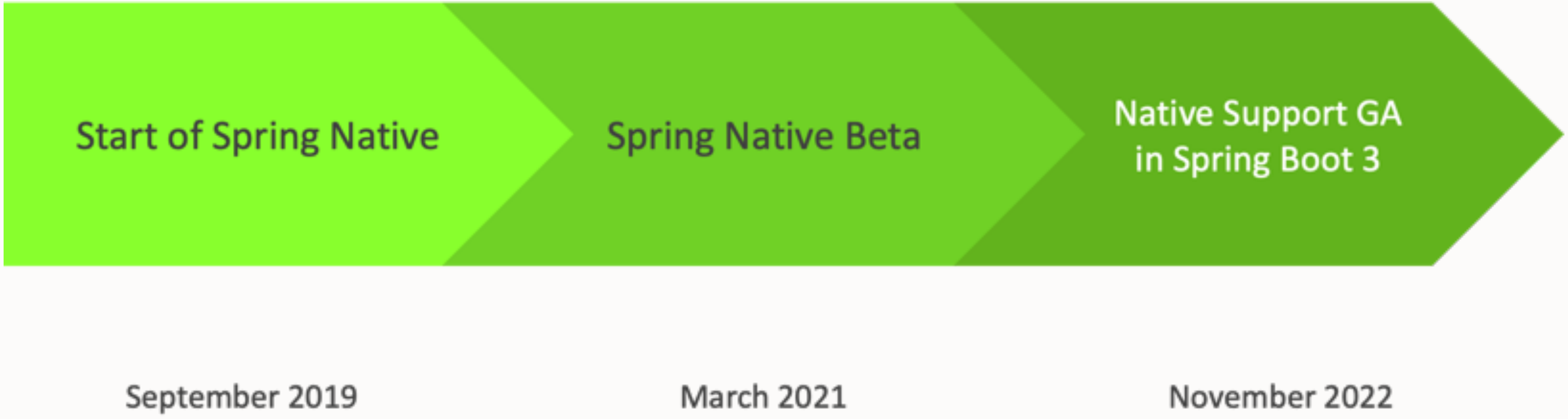
Name	Version	Test Level
`ch.qos.logback:logback-classic` [1]	1.2.11 - latest	★
`com.datastax.oss:java-driver-core`	4.1.5 - latest	★
`com.ecwid.consul:consul-api` [1]	1.4.5 - latest	★
`com.github.ben-manes.caffeine:caffeine` [1]	3.1.2 - latest	★
`com.github.luben:zstd-jni` [1]	1.5.2-5 - latest	★
`com.google.protobuf:protobuf-java-util` [1]	3.21.12 - latest	★
`com.graphql-java:graphql-java` [1]	19.2 - latest	★
`com.graphql-java:graphql-java-extended-validation` [1]	19.1 - latest	★
`com.h2database:h2` [1]	2.1.210 - latest	★
`com.hazelcast:hazelcast` [1]	5.2.1 - latest	★
`com.microsoft.sqlserver:mssql-jdbc` [1]	12.2.0.jre11 - latest	★
`com.mysql:mysql-connector-j` [1]	8.0.31 - latest	★

Required Build Time Step

- Computational effort necessary at build time
- Need a powerful machine with the same target architecture & OS
  - Use GraalVM with GitHub Actions: github.com/marketplace/actions/github-action-for-graalvm
  - Many larger apps can build with 2 GB of memory
- Develop in JIT mode for fast development, only use AOT for final deployment
- For best throughput, use profile-guided optimizations

GraalVM & Spring Boot tips and tricks

Native Image support evolution in Spring Boot



Start of Spring Native

Spring Native Beta

Native Support GA
in Spring Boot 3

September 2019

March 2021

November 2022

AOT processing

- New app lifecycle phase that AOT optimizes and transforms your code for native compilation
- Operates on bean definitions
- Produces the following:
  - Java source code
  - Configuration files for Native Image (META-INF/native-image/*.json)

Registering hints for Native Image

```
  @Bean
  @RegisterReflectionForBinding(Person.class)
         public ItemProcessor<Person, Person> processor() {
                 return item -> new Person(item.firstName().toUpperCase(),
  item.lastName().toUpperCase());

         }


static class BatchApplicationRuntimeHints implements RuntimeHintsRegistrar {

               @Override
               public void registerHints(RuntimeHints hints, ClassLoader classLoader) {
                       hints.resources().registerPattern("persons.csv");
               }

       }
```

Native Build tools: Official Gradle and Maven Plugins 🏗️

- Build, test and run Java applications as native executables
- Out-of-the-box support for native JUnit 5 testing
  - testing Java code with *JUnit 5* behaves in the same way in native execution as with the JVM
  - allows libraries in the JVM ecosystem to run their test suites via GraalVM Native Image

```
plugins {
id 'org.graalvm.buildtools.native' version “0.9.22” // or a newer version
}
```

GraalVM Native Image & JUnit

- `@EnabledInNativeImage`
  - used to signal that the annotated test class or test method is only *enabled* when executing within GraalVM native images
  - when applied at the class level, all test methods within that class will be enabled within a native image

- `@DisabledInNativeImage`
  - used to signal that the annotated test class or test method is only *disabled* when executing within a GraalVM native image.

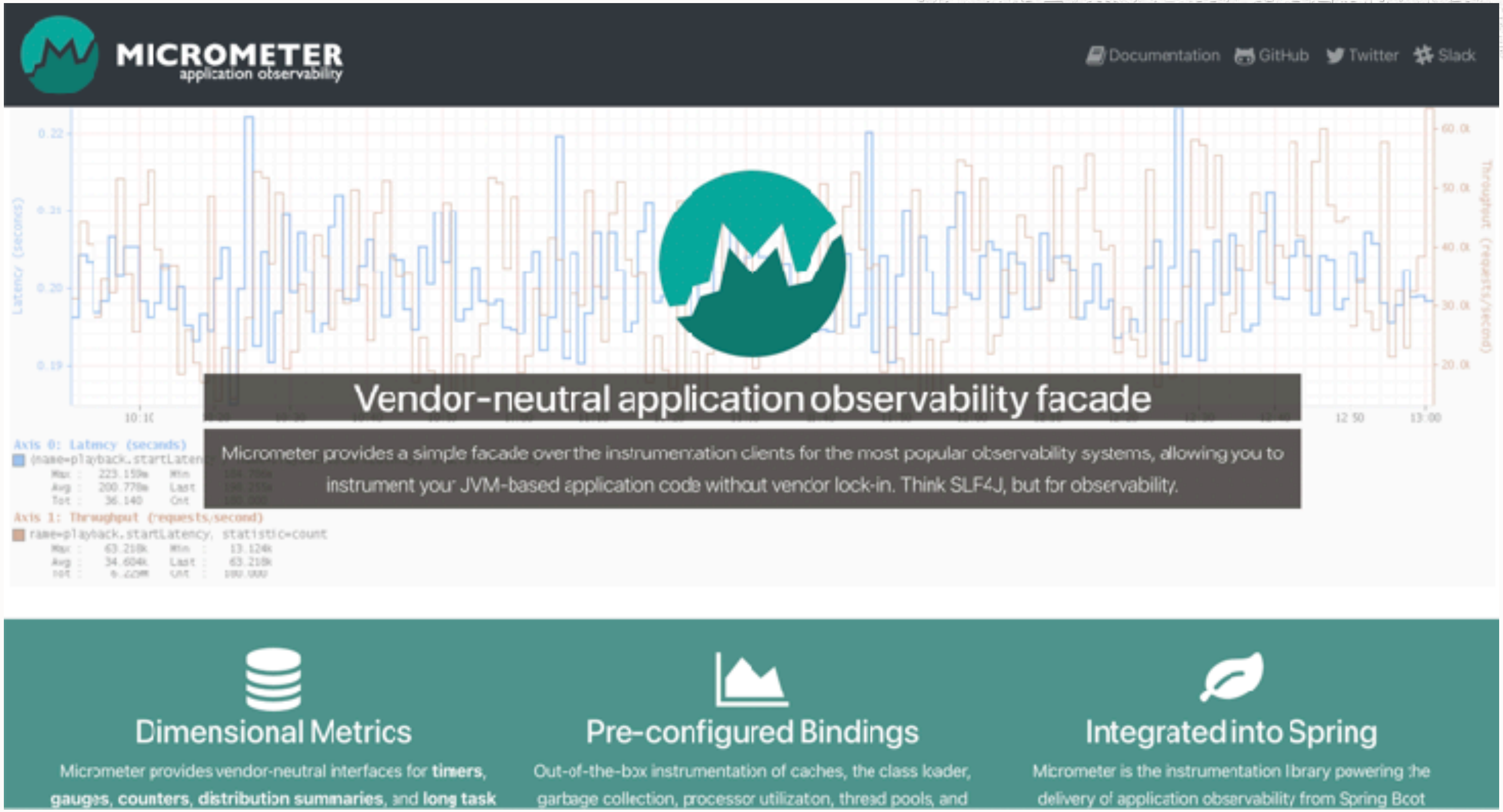
What’s new in GraalVM

GraalVM™
JDK 20
dev builds
now available! 🎉

New monitoring features in GraalVM Native Image 📈

- `-H:+AllowVMInspection -> --enable-monitoring`
  - `--enable-monitoring=<all,heapdump,jfr,jvmstat>`
- added support for jvmstat in Native Image
- keep building out the JFR support in Native Image (thanks to Red Hat for their contributions!)

Micrometer 🤝 Native Image



Vendor-neutral application observability facade

Micrometer provides a simple facade over the instrumentation clients for the most popular observability systems, allowing you to instrument your JVM-based application code without vendor lock-in. Think SLF4J, but for observability.

Dimensional Metrics

Micrometer provides vendor-neutral interfaces for **timers**, **gauges**, **counters**, **distribution summaries**, and **long task**

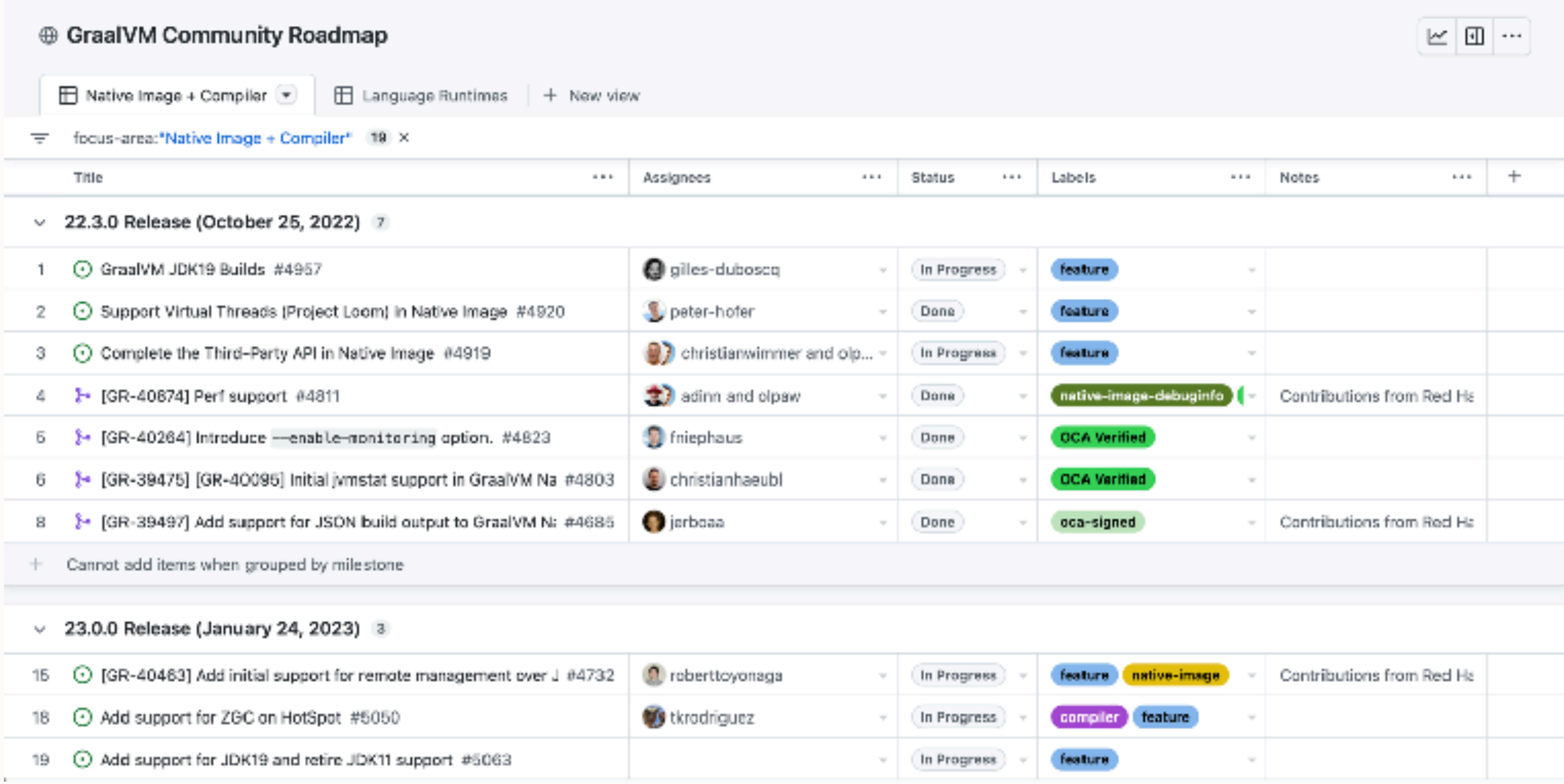
Pre-configured Bindings

Out-of-the-box instrumentation of caches, the class loader, garbage collection, processor utilization, thread pools, and

Integrated into Spring

Micrometer is the instrumentation library powering the delivery of application observability from Spring Boot

GraalVM Community roadmap on GitHub



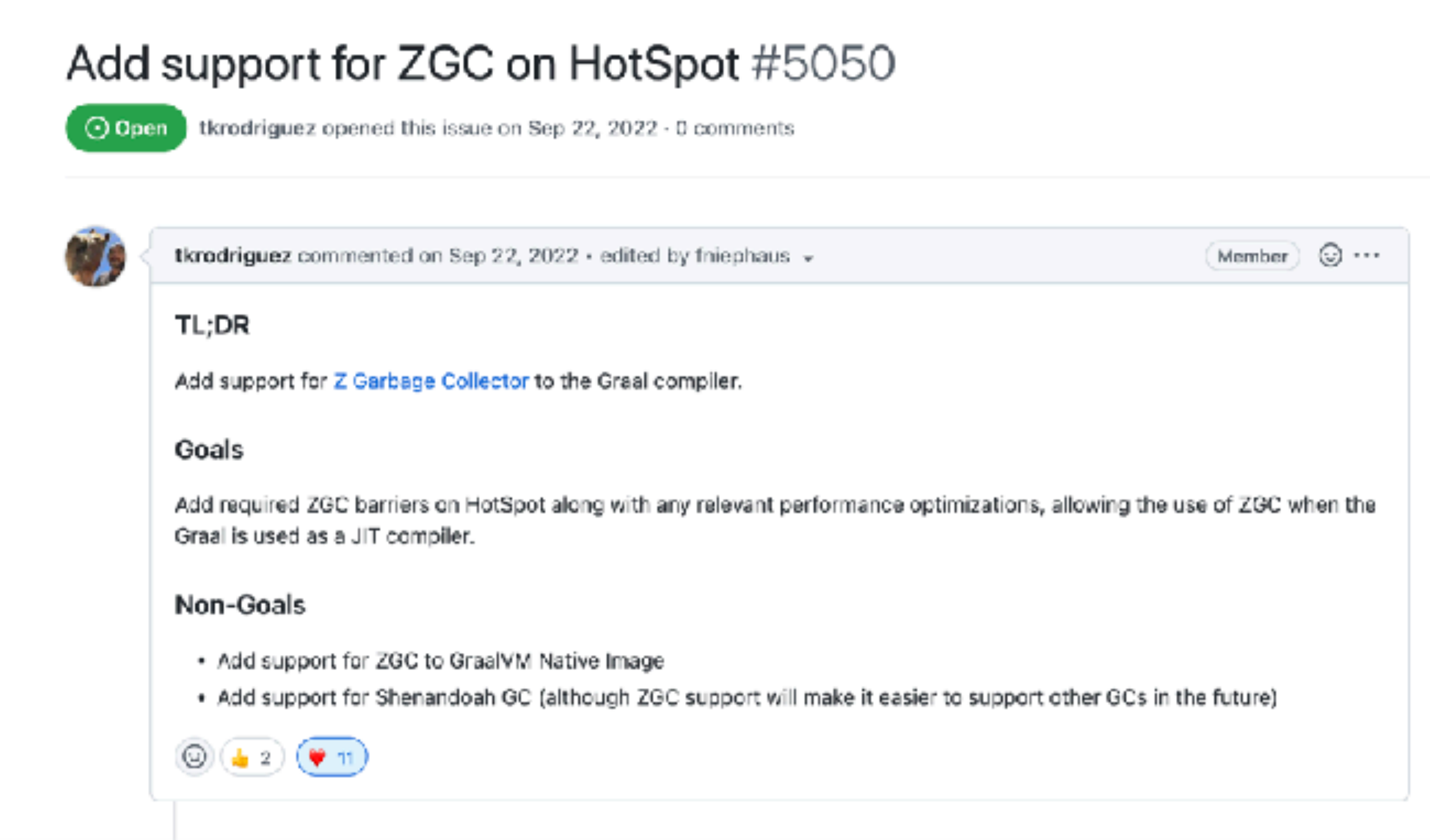
GraalVM Community Roadmap

focus-area:"Native Image + Compiler" 19 ×

	Title	Assignees	Status	Labels	Notes
	22.3.0 Release (October 25, 2022) 7				
1	GraalVM JDK19 Builds #4957	gilles-duboscq	In Progress	feature	
2	Support Virtual Threads (Project Loom) in Native Image #4920	peter-hofer	Done	feature	
3	Complete the Third-Party API in Native Image #4919	christianwimmer and olp...	In Progress	feature	
4	[GR-40674] Perf support #4811	adinn and olpaw	Done	native-image-debuginfo	Contributions from Red Ha
5	[GR-40264] Introduce `--enable-monitoring` option. #4823	fniephaus	Done	OCA Verified	
6	[GR-39475] [GR-40095] Initial jvmstat support in GraalVM Na #4803	christianhaeubl	Done	OCA Verified	
8	[GR-39497] Add support for JSON build output to GraalVM N: #4685	jerboaa	Done	oca-signed	Contributions from Red Ha
+	Cannot add items when grouped by milestone				
	23.0.0 Release (January 24, 2023) 3				
15	[GR-40463] Add initial support for remote management over J #4732	roberttoyonaga	In Progress	feature native-image	Contributions from Red Ha
18	Add support for ZGC on HotSpot #5050	tkrodriguez	In Progress	compiler feature	
19	Add support for JDK19 and retire JDK11 support #5063		In Progress	feature	

https://github.com/orgs/oracle/projects/6

What’s next for GraalVM



Add support for ZGC on HotSpot #5050

Open tkrodriguez opened this issue on Sep 22, 2022 · 0 comments

tkrodriguez commented on Sep 22, 2022 • edited by fniephaus Member

TL;DR

Add support for Z Garbage Collector to the Graal compiler.

Goals

Add required ZGC barriers on HotSpot along with any relevant performance optimizations, allowing the use of ZGC when the Graal is used as a JIT compiler.

Non-Goals

- Add support for ZGC to GraalVM Native Image
- Add support for Shenandoah GC (although ZGC support will make it easier to support other GCs in the future)

👍 2 ❤️ 11

What’s next for Native Image

- 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
- Further reduce build time and footprint of the Native Image builder
- IDE support for Native Image configuration and agent-based configuration
- Further improving GC performance and adding new GC implementations

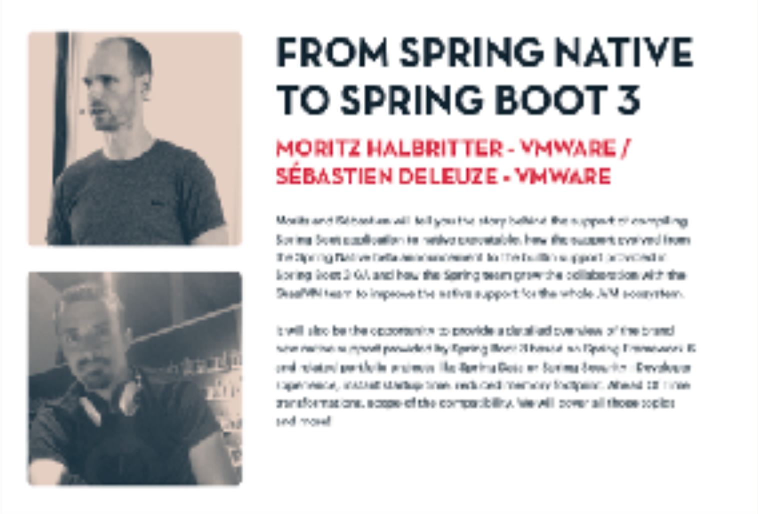
Get started with GraalVM

Get started with GraalVM

```
bash <(curl -sL https://get.graalvm.org/jdk)\
graalvm-ce-java19-22.3.0

sdk install java 22.3.r19-grl
```

More about GraalVM @ Spring I/O

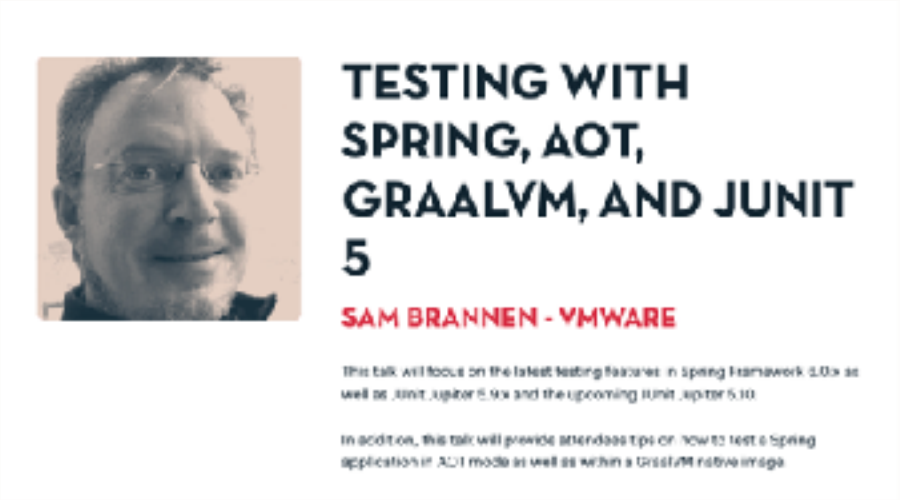


FROM SPRING NATIVE TO SPRING BOOT 3

MORITZ HALBRITTER - VMWARE / SÉBASTIEN DELEUZE - VMWARE

Moritz and Sébastien will tell you the story behind the support of compiling Spring Boot application to native executable, how the support evolved from the Spring Native beta announcement to the built-in support provided in Spring Boot 3 GA and how the Spring team grew the collaboration with the GraalVM team to improve the native support for the whole JVM ecosystem.

It will also be the opportunity to provide a detailed overview of the brand new native support provided by Spring Boot 3 based on Spring Framework 6 and related portfolio projects like Spring Data or Spring Security: Developer experience, instant startup time, reduced memory footprint, Ahead Of Time transformations, scope of the compatibility. We will cover all those topics and more!

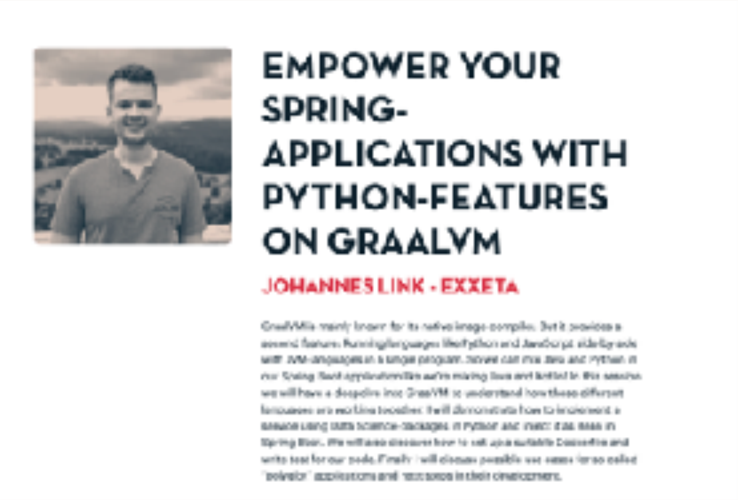


TESTING WITH SPRING, AOT, GRAALVM, AND JUNIT 5

SAM BRANNEN - VMWARE

This talk will focus on the latest testing features in Spring Framework 6.0.x as well as JUnit Jupiter 5.9.x and the upcoming JUnit Jupiter 5.10.

In addition, this talk will provide attendees tips on how to test a Spring application in AOT mode as well as within a GraalVM native image.



EMPOWER YOUR SPRING-APPLICATIONS WITH PYTHON-FEATURES ON GRAALVM

JOHANNES LINK - EXXETA

GraalVM is mainly known for its native image compiler. But it provides a second feature: Running languages like Python and JavaScript side by side with JVM-languages in a single program. So we can mix Java and Python in our Spring Boot applications like we're mixing Java and Kotlin. In this session we will have a deep dive into GraalVM to understand how these different languages are working together. It will demonstrate how to implement a service using data science packages in Python and inject it as bean in Spring Boot. We will also discover how to set up a suitable Devcontainer and write test for our code. Finally I will discuss possible use cases for so called "polyglot" applications and restrictions in their environment.

Thank you!

Alina Yurenko

@alina_yurenko