ORACLE

ORACLE

GraalVM Native Image Deep Dive

Part I

Alina Yurenko
Developer Advocate for GraalVM
@alina_yurenko

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.

Universal Virtual Machine

GraalVM™

1. Run programs more efficient
2. Make developers more productive

GraalVM Project Goals

1. High performance for abstractions of any language
2. Low-footprint ahead-of-time mode for JVM-based languages
3. Convenient language interoperability and polyglot tooling
4. Simple embeddability in native and managed programs

GraalVM™

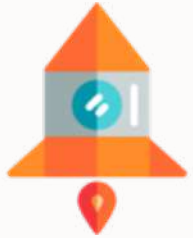
database

standalone

What GraalVM offers

High Performance

Optimize application performance with GraalVM compiler



Fast Startup

Compile your application AOT and start instantly

Polyglot

Mix & match languages with seamless interop

Open Source

See what's inside, track features progress, contribute

Production-ready!🎉

Community Edition

GraalVM Community is available for free for evaluation, development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for Linux, macOS X, and Windows platforms on x86 64-bit systems. Windows support is experimental.

DOWNLOAD FROM GITHUB

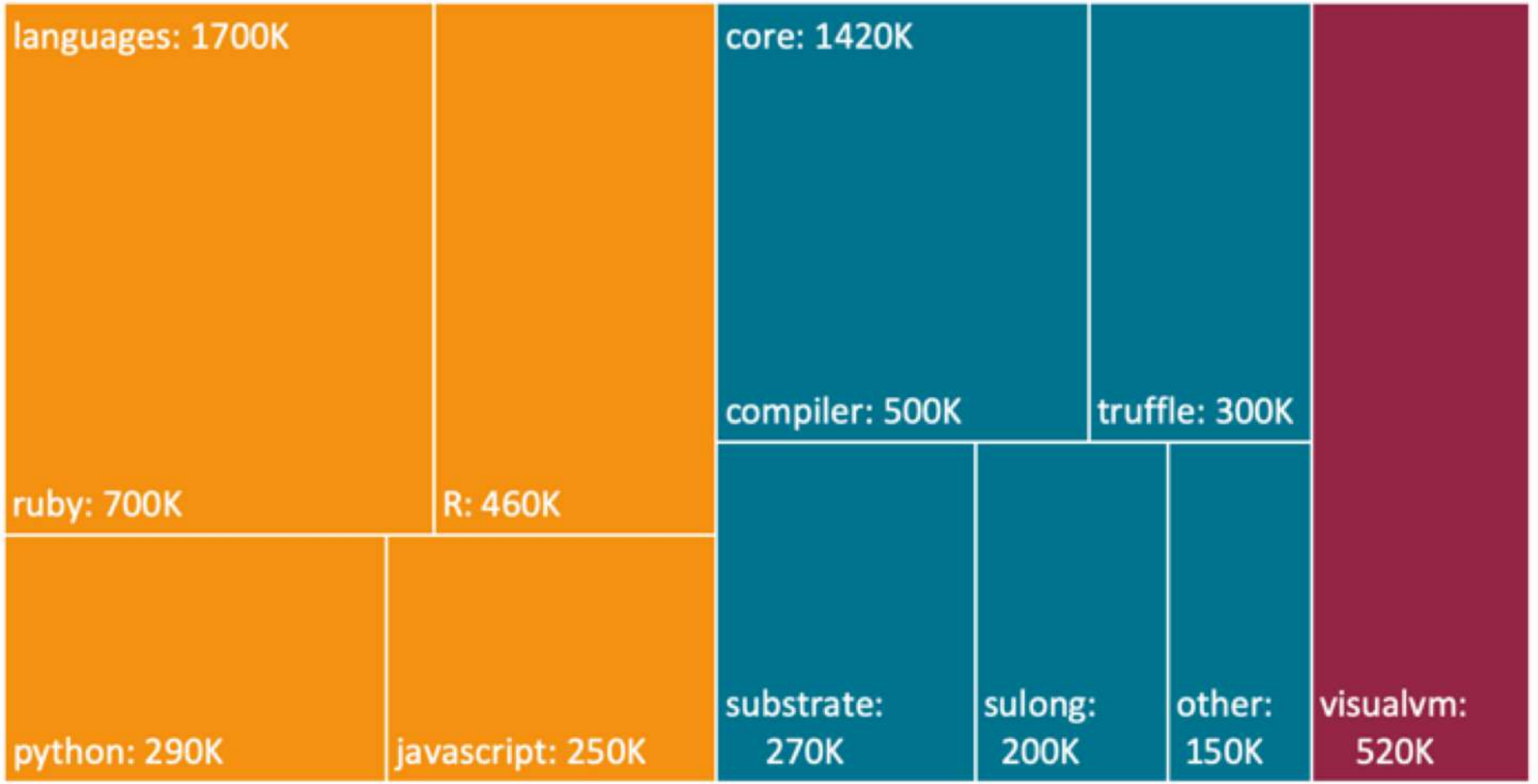
Enterprise Edition

GraalVM Enterprise provides additional performance, security, and scalability relevant for running applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for Linux, macOS X, and Windows platforms on x86 64-bit systems. Windows support is experimental.

DOWNLOAD FROM OTN

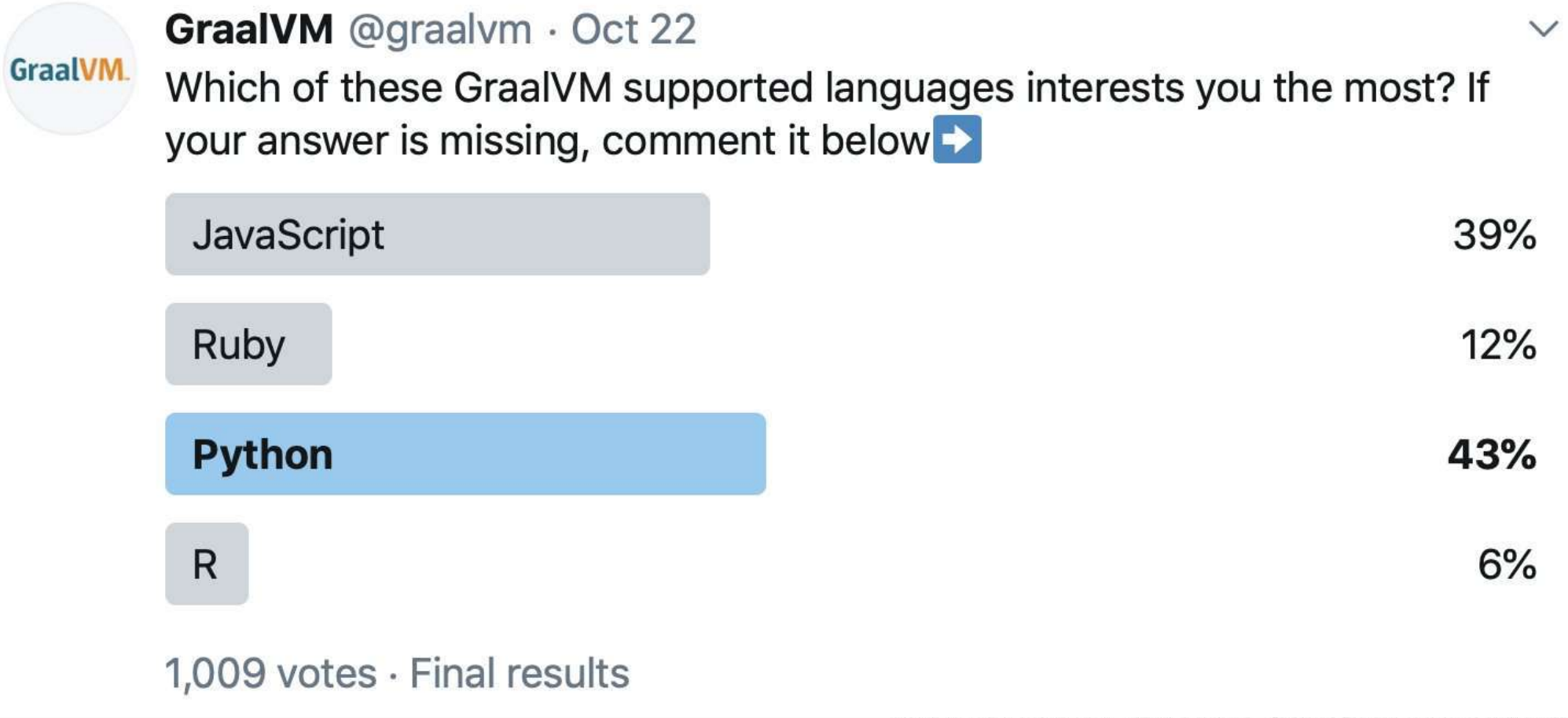
get both: graalvm.org

Open Source LOC actively maintained for GraalVM



Total: 3,640,000 lines of code

GraalVM Language Ecosystem


GraalVM
GraalVM @graalvm · Oct 22
Which of these GraalVM supported languages interests you the most? If your answer is missing, comment it below
JavaScript
39%
Ruby
12%
Python
43%
R
6%
1,009 votes · Final results

Multiplicative Value-Add of GraalVM Ecosystem

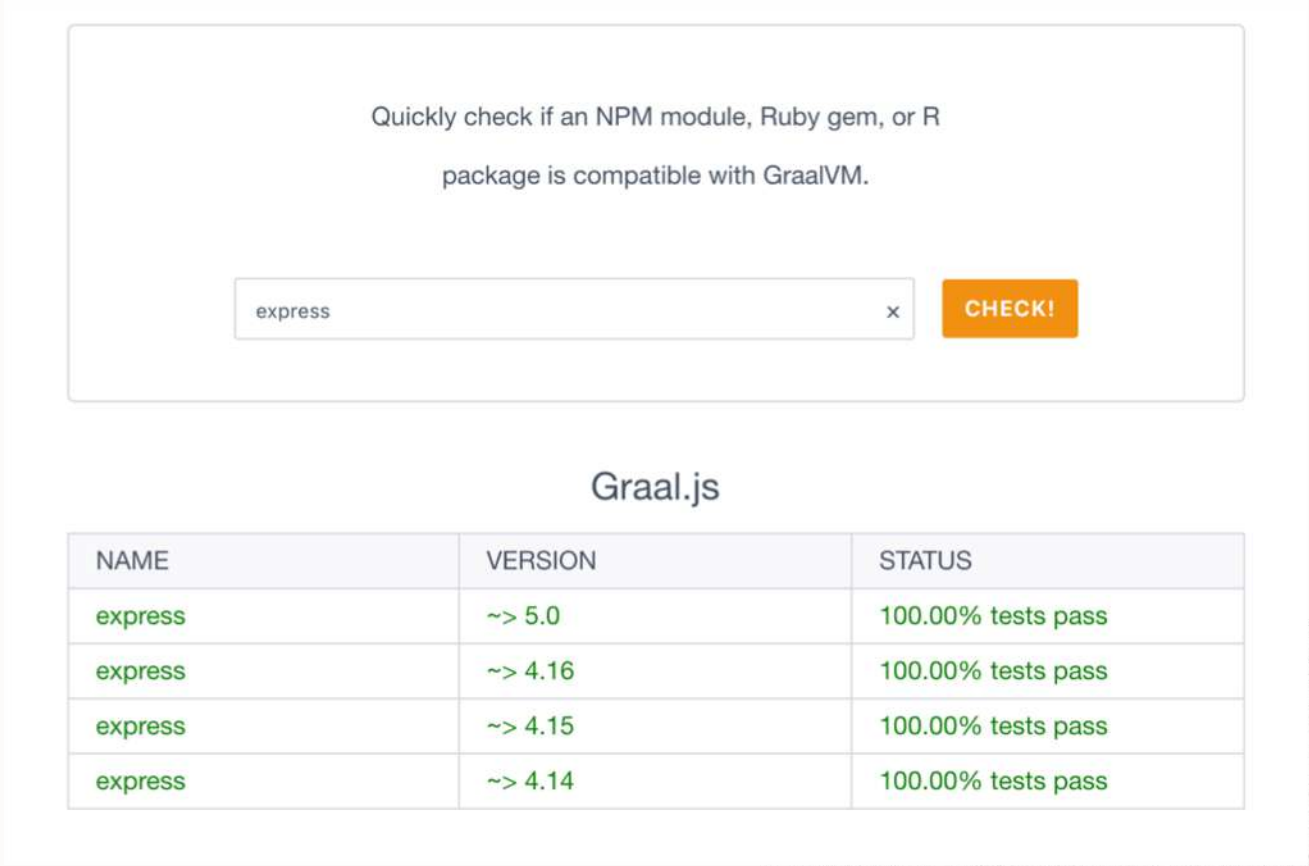
Languages	*	GraalVM	*	Embeddings
Java JavaScript Ruby R Python C/C++, FORTRAN, …		Optimizations Tooling Interoperability Security		HotSpot JVM Oracle RDBMS Node.js Standalone Spark …

Add your own language or embedding or language-agnostic tools!

JavaScript & Node.js

- ECMAScript 2019 complaint JavaScript engine;
- Access to GraalVM language interoperability and common tooling;
- Constantly tested against 90,000+ npm modules, `including express, react, async, request`

Compatibility Tool



https://www.graalvm.org/docs/reference-manual/compatibility

Nashorn Migration Guide

Migration guide from Nashorn to GraalVM JavaScript

This document serves as migration guide for code previously targeted to the Nashorn engine. See the JavaInterop.md for an overview of supported Java interoperability features.

Both Nashorn and GraalVM JavaScript support a similar set of syntax and semantics for Java interoperability. The most important differences relevant for migration are listed here.

Nashorn features available by default:

- `Java.type`, `Java.typeName`
- `Java.from`, `Java.to`
- `Java.extend`, `Java.super`
- Java package globals: `Packages`, `java`, `javafx`, `javax`, `com`, `org`, `edu`

Nashorn compatibility mode

GraalVM JavaScript provides a Nashorn compatibility mode. Some of the functionality necessary for Nashorn compatibility is only available when the `js.nashorn-compat` option is enabled. This is the case for Nashorn-specific extensions that GraalVM JavaScript does not want to expose by default. Note that you have to enable [experimental options](Options.md#Stable and Experimental options) to use this flag.

The `js.nashorn-compat` option can be set using a command line option:

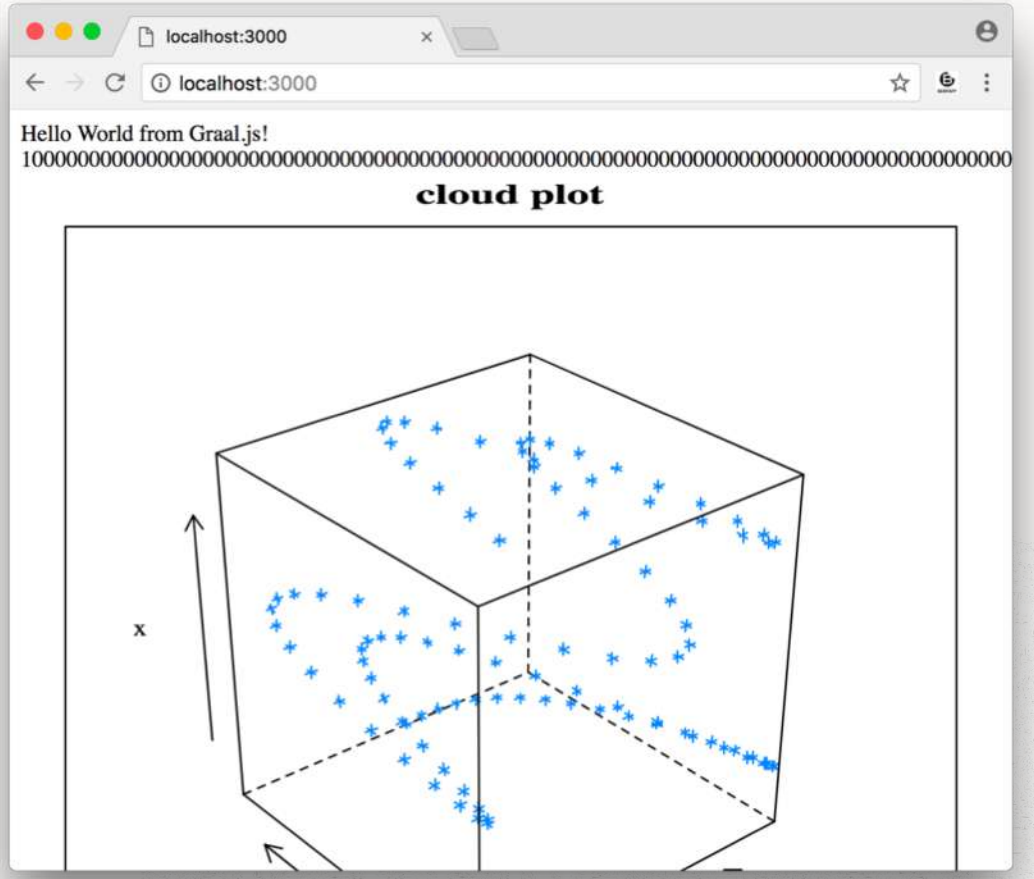
```
$ js --experimental-options --js.nashorn-compat=true
```

https://github.com/graalvm/graaljs/blob/master/docs/user/NashornMigrationGuide.md

JavaScript + Java + R

server.js

```
41
42  const express = require('express')
43  const app = express()
44
45  const BigInteger = Java.type('java.math.BigInteger')
46
47
48  app.get('/', function (req, res) {
49    var text = '<h1>Hello from Graal.js!</h1>'
50
51    // Using Java standard library classes
52    text += BigInteger.valueOf(10).pow(100)
53                .add(BigInteger.valueOf(43)).toString() + '<br>'
54
55    // Using R methods to return arrays
56    text += Polyglot.eval('R',
57        'ifelse(1 > 2, "no", paste(1:42, c="|"))') + '<br>'
58
59    // Using R interoperability to create graphs
60    text += Polyglot.eval('R',
61      `svg();
62      require(lattice);
63      x <- 1:100
64      y <- sin(x/10)
65      z <- cos(x^1.3/(runif(1)*5+10))
66      print(cloud(x~y*z, main="cloud plot"))
67      grDevices:::svg.off()
68    `);
```

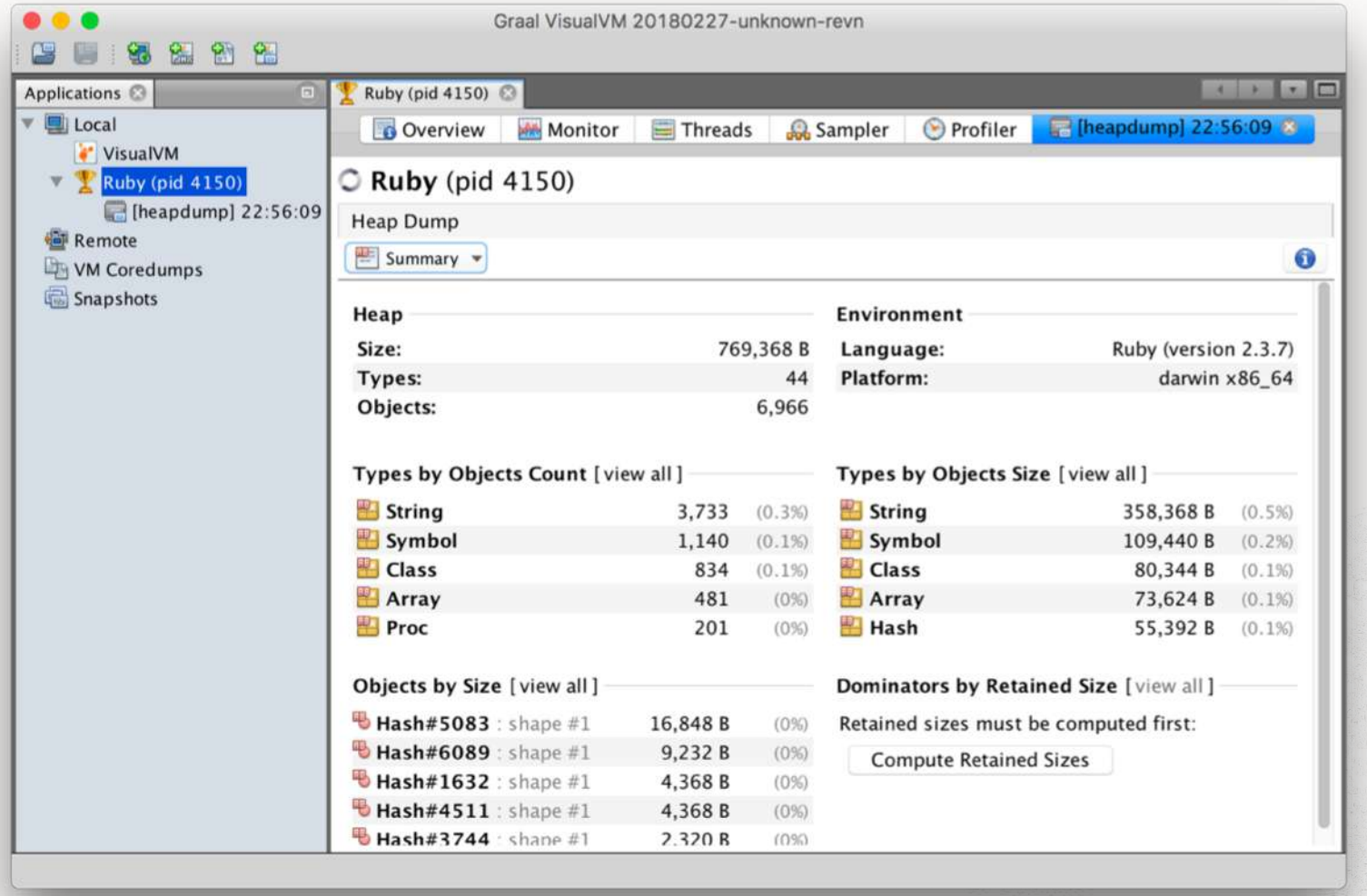


Polyglot in a Database

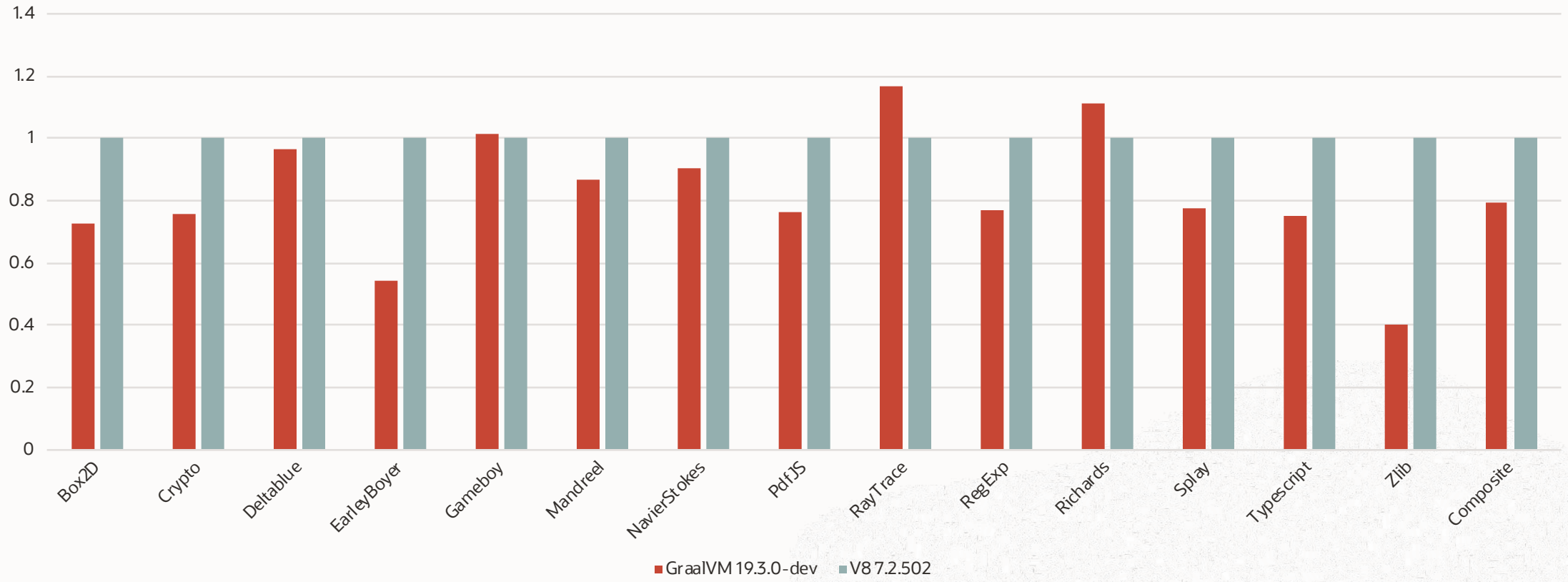
```
$ npm install validator
$ npm install @types/validator
$ dbjs deploy -u scott -p tiger -c localhost:1521/ORCLCDB validator
$ sqlplus scott/tiger@localhost:1521/ORCLCDB
```

```
SQL> select validator.isEmail('hello.world@oracle.com') from dual;

VALIDATOR.ISEMAIL('HELLO.WORLD@ORACLE.COM')
-------------------------------------------
                                          1


SQL> select validator.isEmail('hello.world') from dual;

VALIDATOR.ISEMAIL('HELLO.WORLD')
--------------------------------
                               0
```

Polyglot tools: GraalVM VisualVM



Graal.js Performance (versus V8)



FastR

- GNU-R compatible R implementation
  - Including the C/Fortran interface
- Built on top of the GraalVM platform
  - Leverages GraalVM optimizing compiler
  - Integration with GraalVM dev tools
  - Zero overhead interop with other GraalVM languages

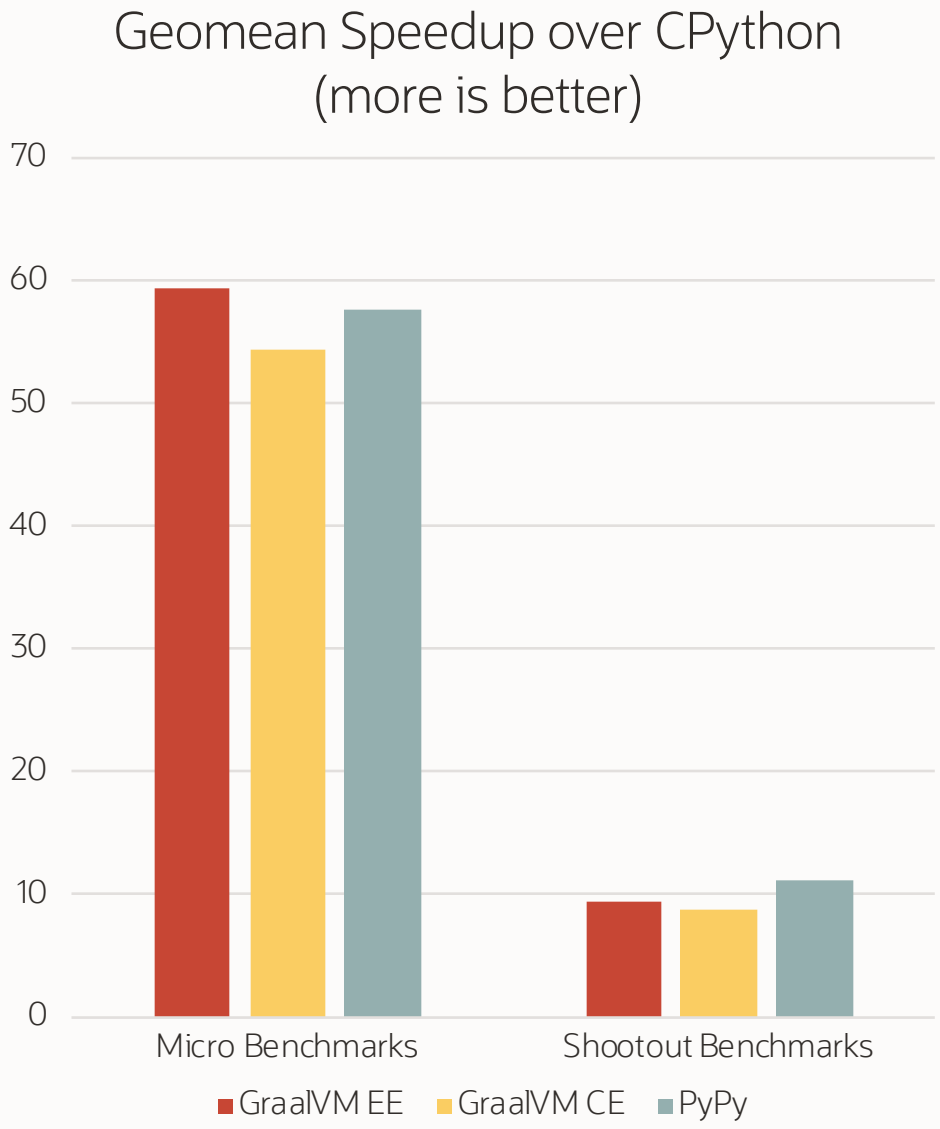
GraalVM in practice at the Dutch National Police

JavaZone

https://vimeo.com/360837119

Python Performance

Comparable to PyPy, the fastest alternative

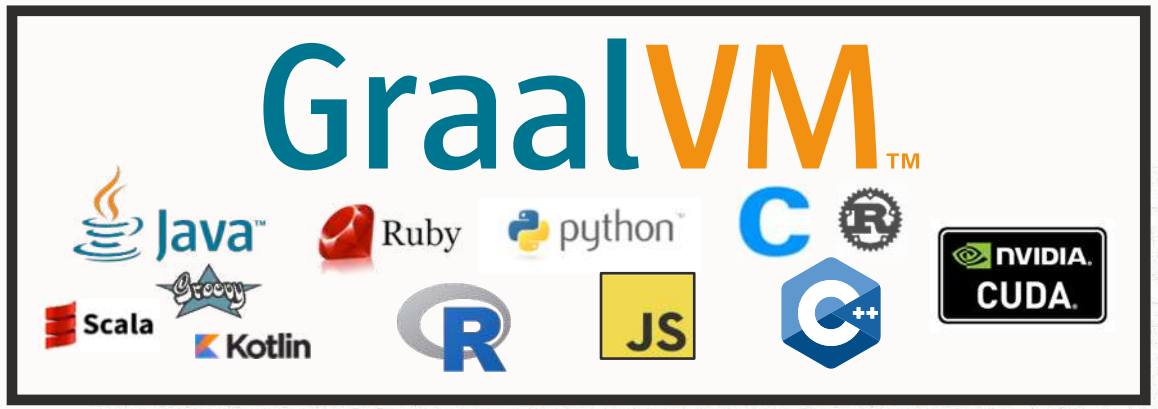


Using grCUDA to Access Nvidia GPUs

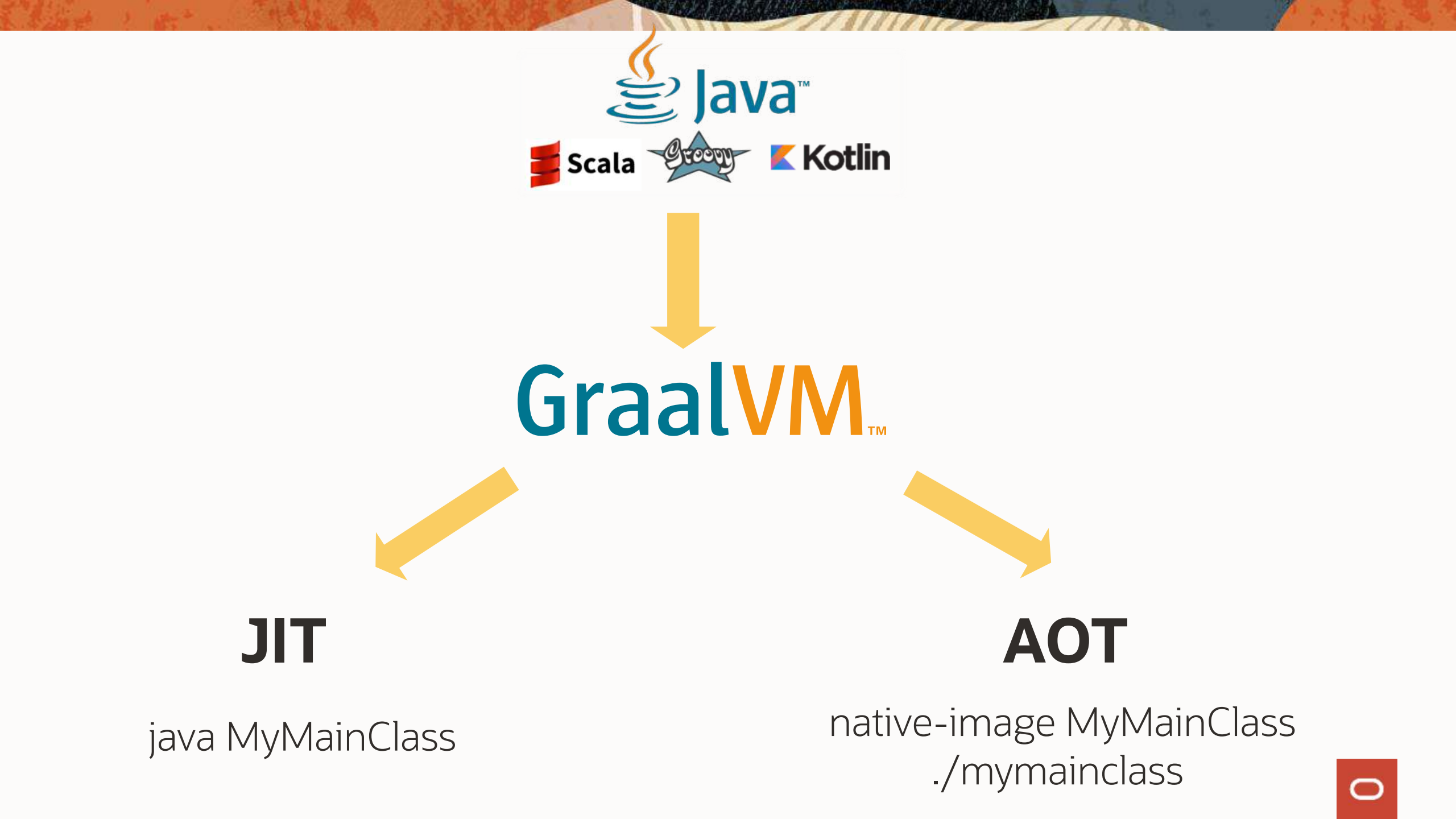
- Efficient exchange of data between host language and GPU without burdening the programmer
- Expose GPU resources in ways that are native in the host language, e.g., as arrays
- Allow programmers to invoke existing GPU code from their host language
- Allow programmers to define new GPU kernels on the fly
- Polyglot interface: uniform bindings across several programming languages

- Implemented as a “Truffle Language”
  (although “CUDA” is a platform, not a language)

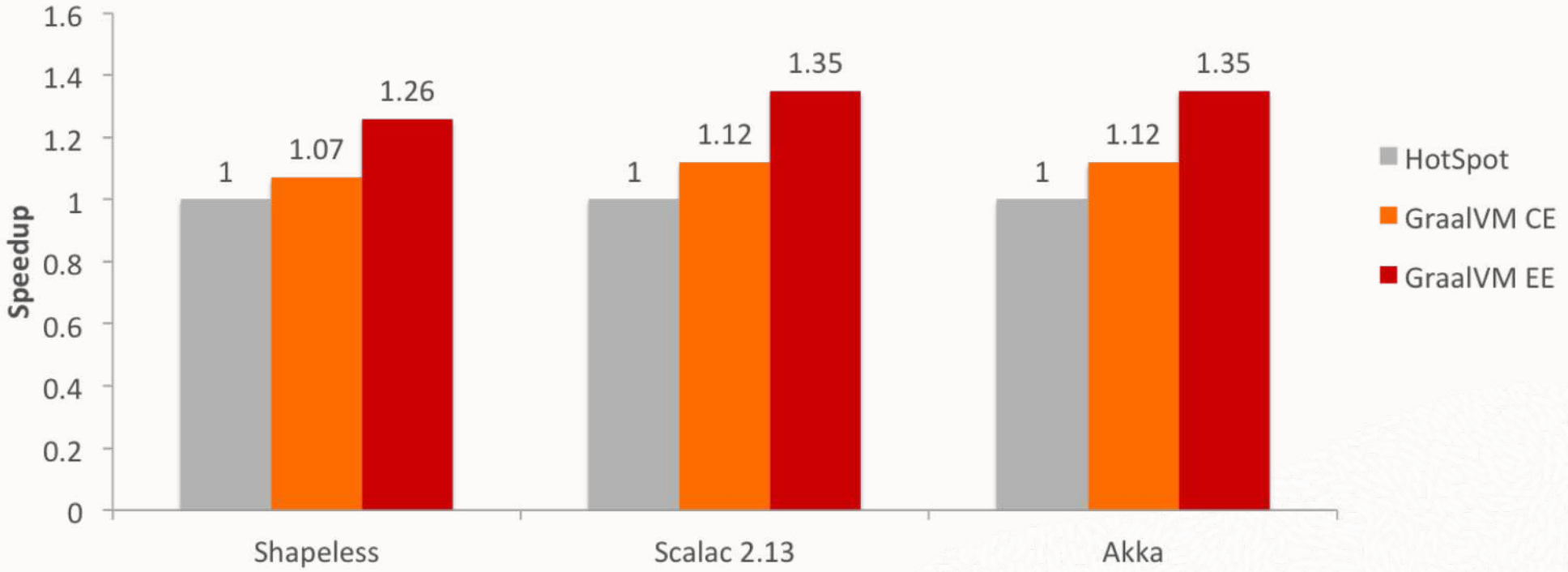
- Developed by NVIDIA in collaboration
  with Oracle Labs
- BSD 3-clause license



GraalVM for Java


Java
Scala
Groovy
Kotlin
GraalVM
JIT
java MyMainClass
AOT
native-image MyMainClass
./mymainclass

Scala performance



https://medium.com/graalvm/compiling-scala-faster-with-graalvm-86c5c0857fa3

GraalVM Native Images

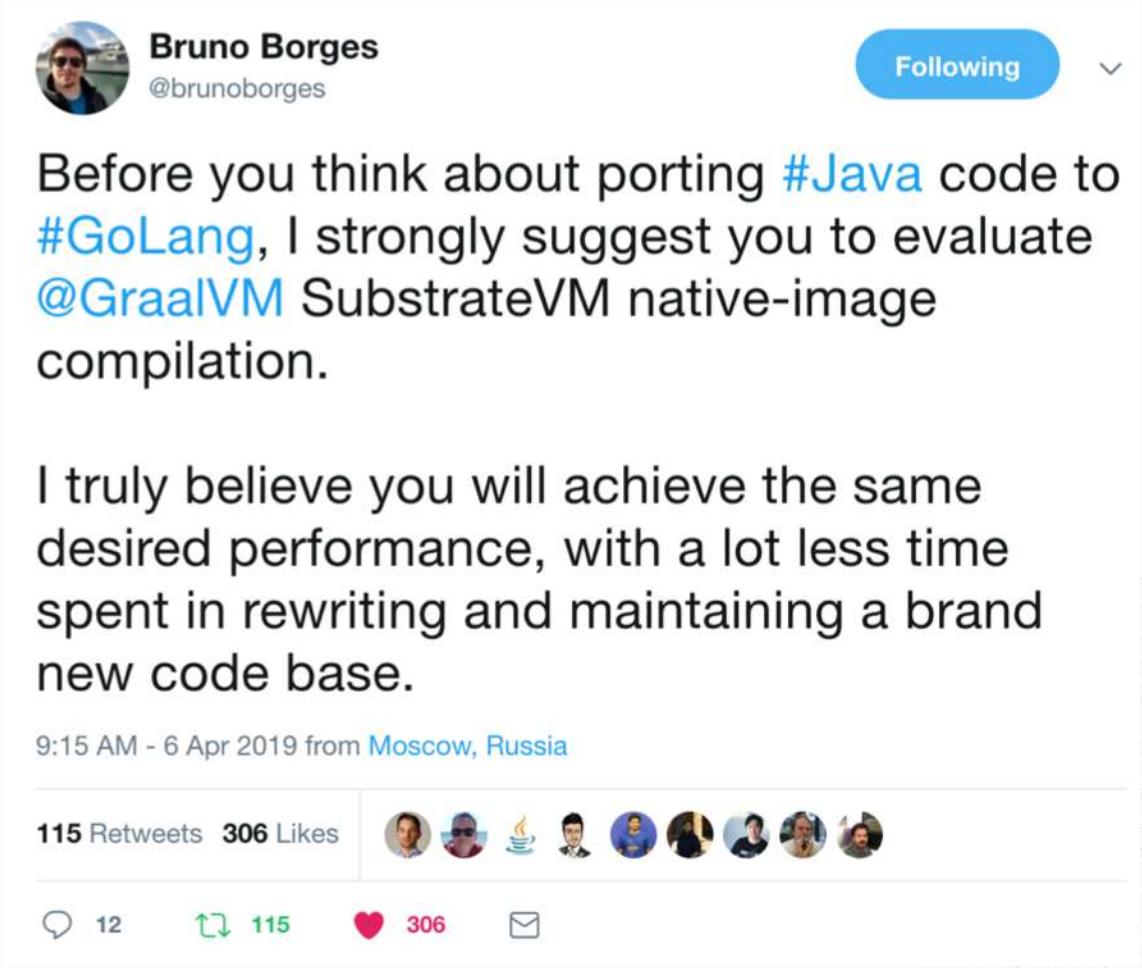
GraalVM Native Images

- Java program, compiled into a standalone native executable;
- Instant startup;
- Low memory footprint;
- AOT-compiled using the GraalVM compiler.

Get Ready for the Cloud and Microservices

Important evaluation metrics:

- Startup time
- Memory footprint
- Peak requests per MByte-second

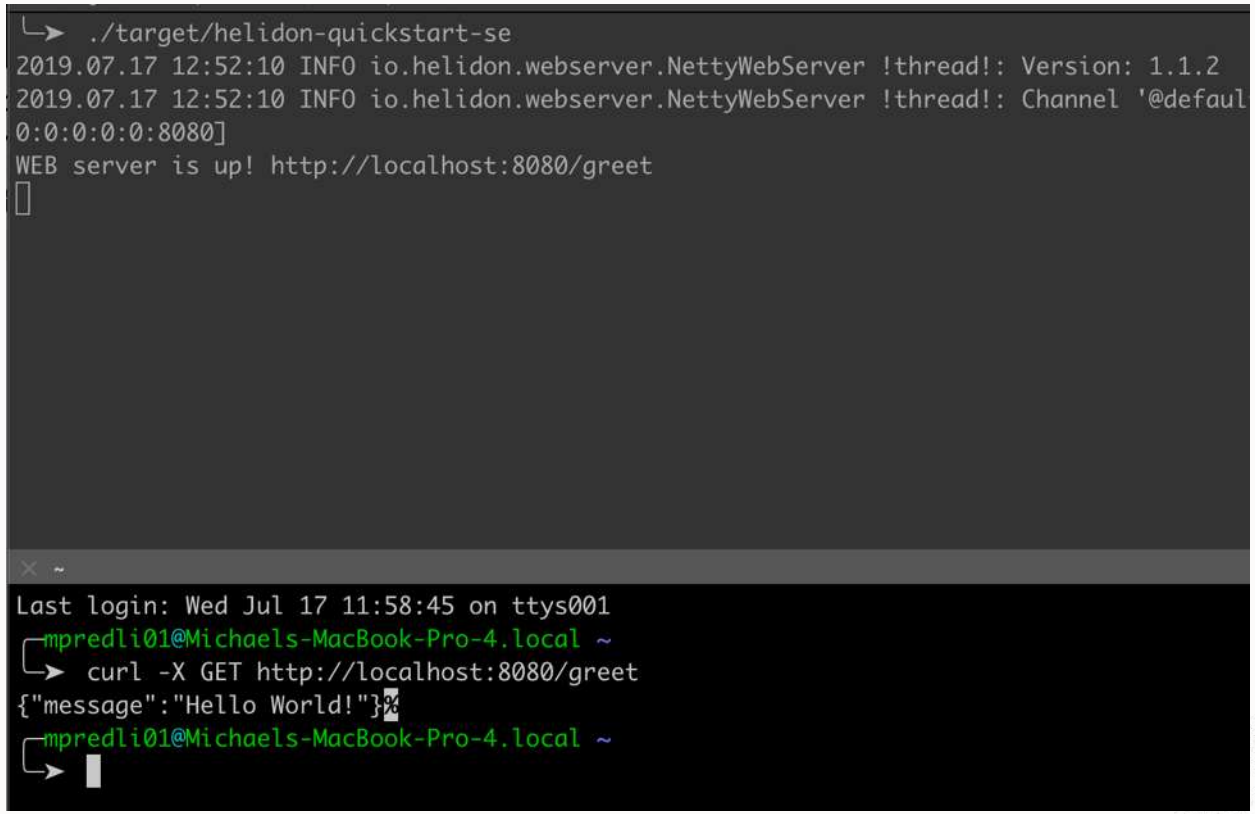


Micronaut

Create your first Micronaut GraalVM application:

https://guides.micronaut.io/micronaut-creating-first-graal-app/guide/index.html

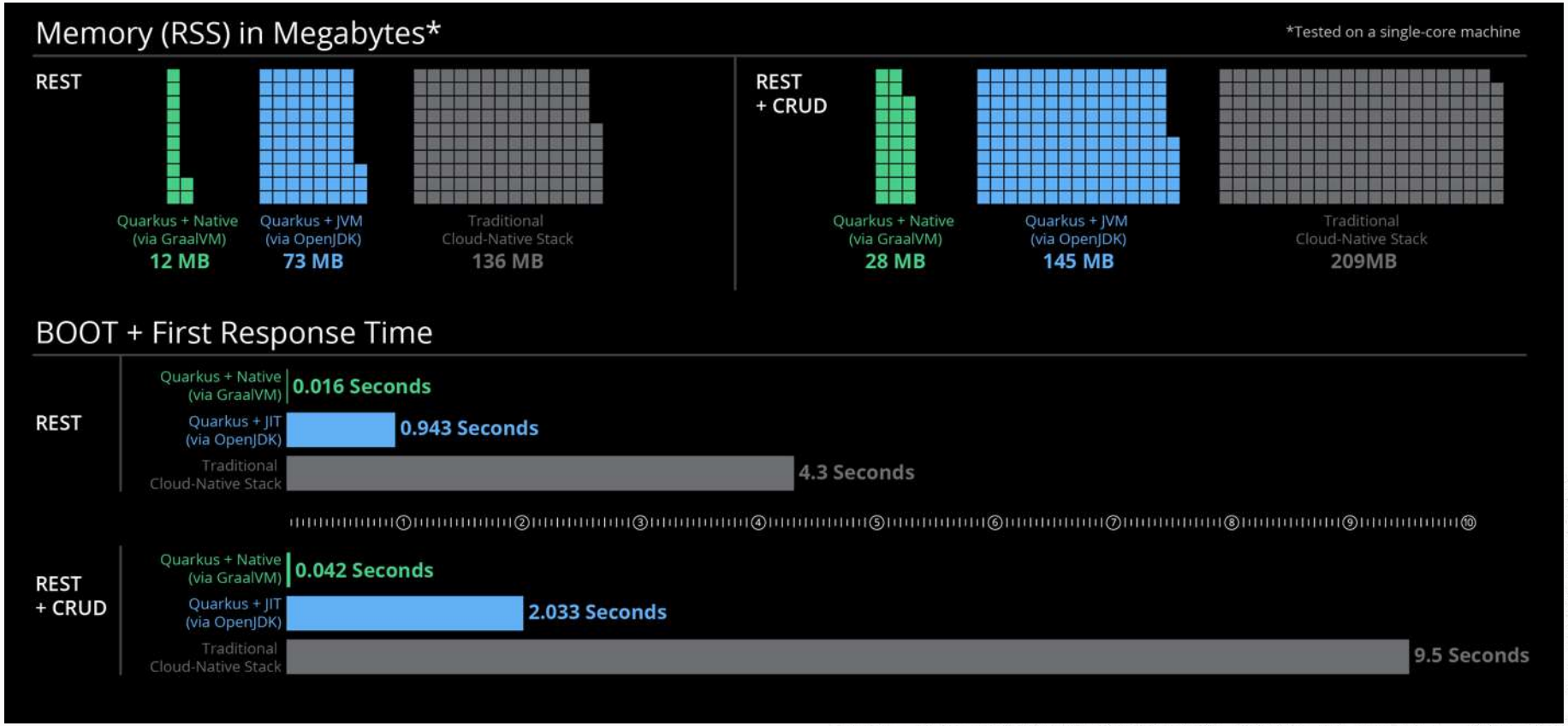
Helidon



Helidon and GraalVM:

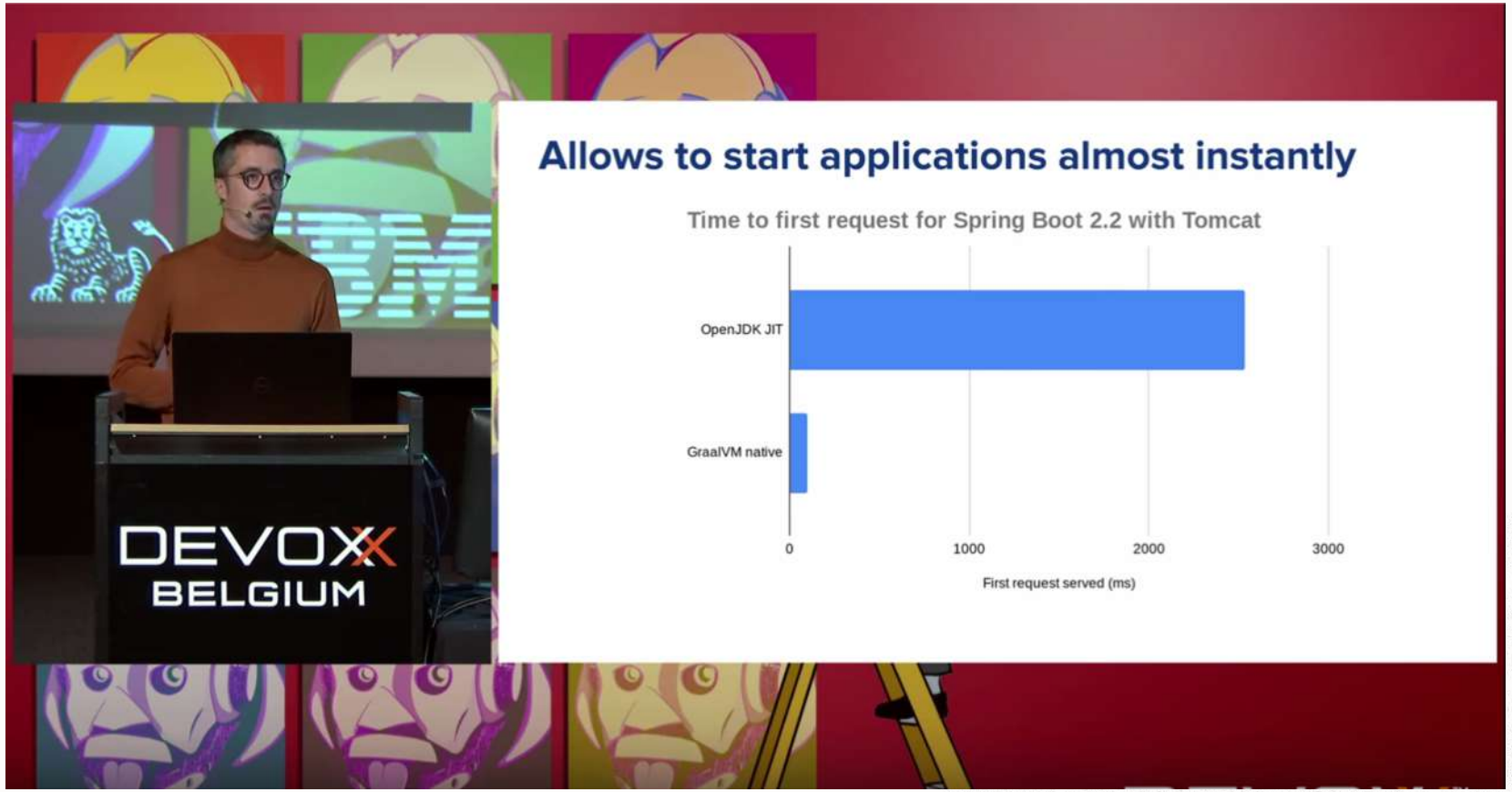
https://helidon.io/docs/latest/#/guides/36_graalnative

Quarkus



https://quarkus.io/guides/building-native-image

Spring Boot Applications as GraalVM Native Images



https://www.youtube.com/watch?v=3eoAxphAUIg

Spring Boot Applications as GraalVM Native Images

```
Alinas-MacBook-Pro:~/spring-graal-native/spring-graal-native-samples$ ls
commandlinerunner       spring-petclinic-jpa    vanilla-orm2
commandlinerunner-maven springmvc-tomcat        vanilla-rabbit
kotlin-webmvc           vanilla-grpc            vanilla-thymeleaf
logger                  vanilla-jpa             vanilla-tx
messages                vanilla-orm             webflux-netty
```

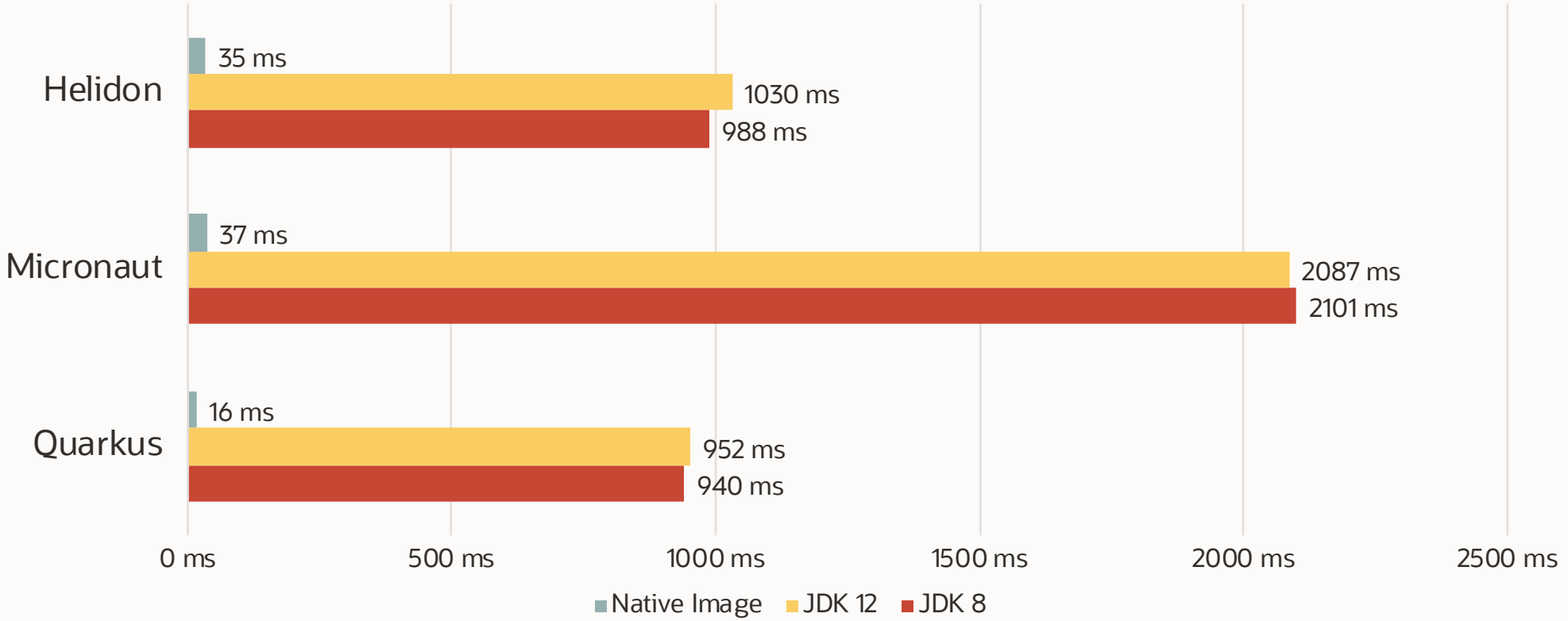
"Spring Graal Native" project: https://github.com/spring-projects-experimental/spring-graal-native

https://www.youtube.com/watch?v=3eoAxphAUIg

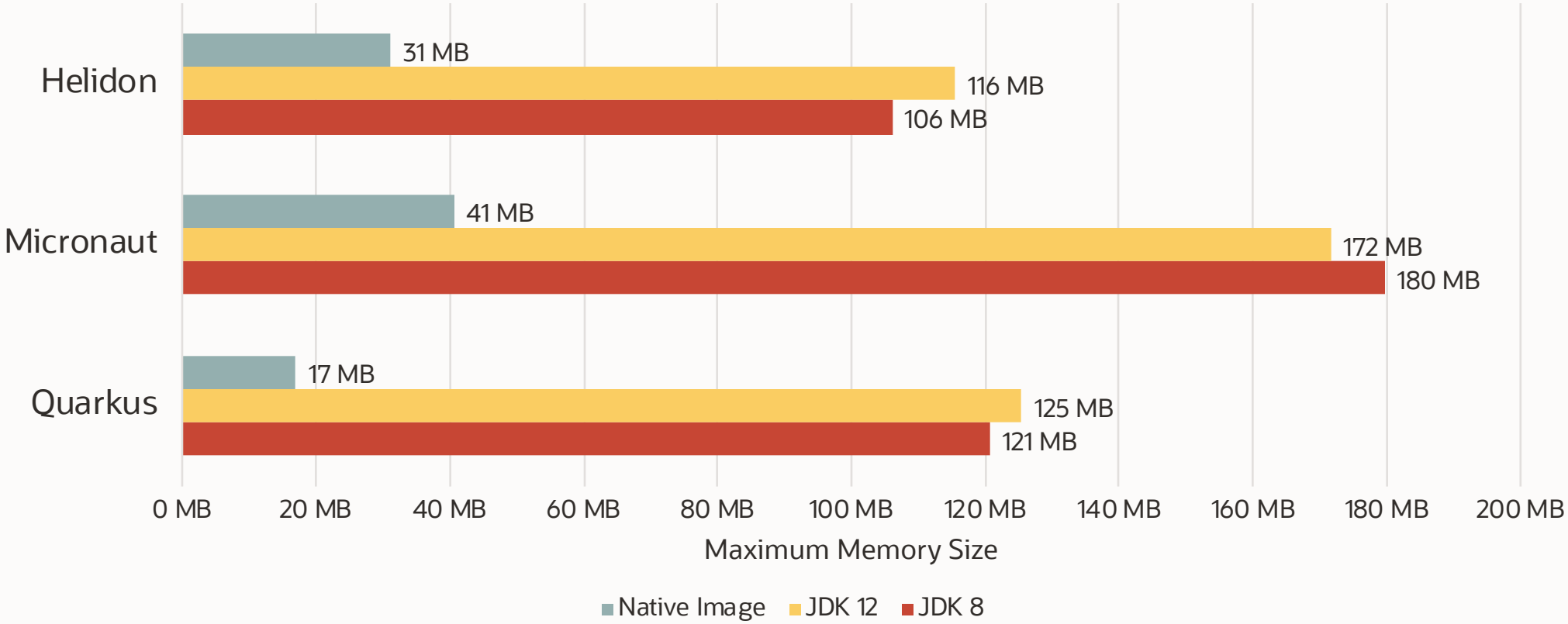
Demo time

https://github.com/spring-projects-experimental/spring-graal-native/

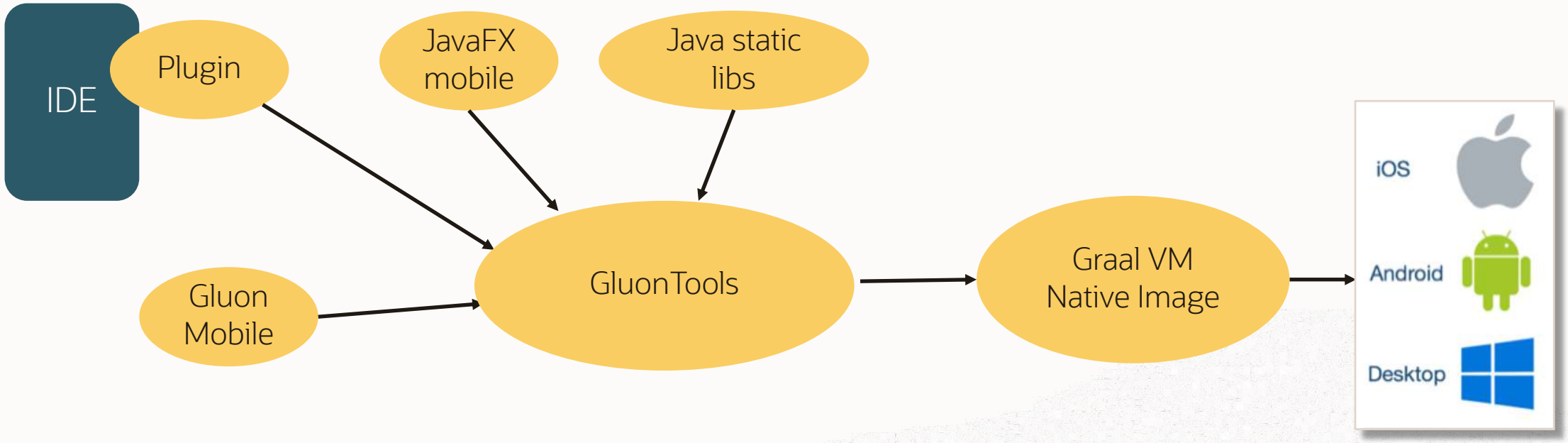
Microservice Frameworks: Startup Time



Microservice Frameworks: Memory Usage



Do even more with GraalVM: Cross-Platform Development



GraalVM native image for real-world projects

Jump Start Your Project

- How do I know quickly if my application will run as a native image?
- Disable fallback image generation
    - --no-fallback
- Report unsupported features at run time
    - --report-unsupported-elements-at-runtime
- Allow incomplete class path: throw linking errors at run time
    - --allow-incomplete-classpath
- Trace reflection, JNI, resource, … usage on Java HotSpot VM
    - java -agentlib:native-image-agent=config-output-dir=META-INF/native-image …
- Initialize all application classes at run time: default since GraalVM 19.0

Tracing Agent

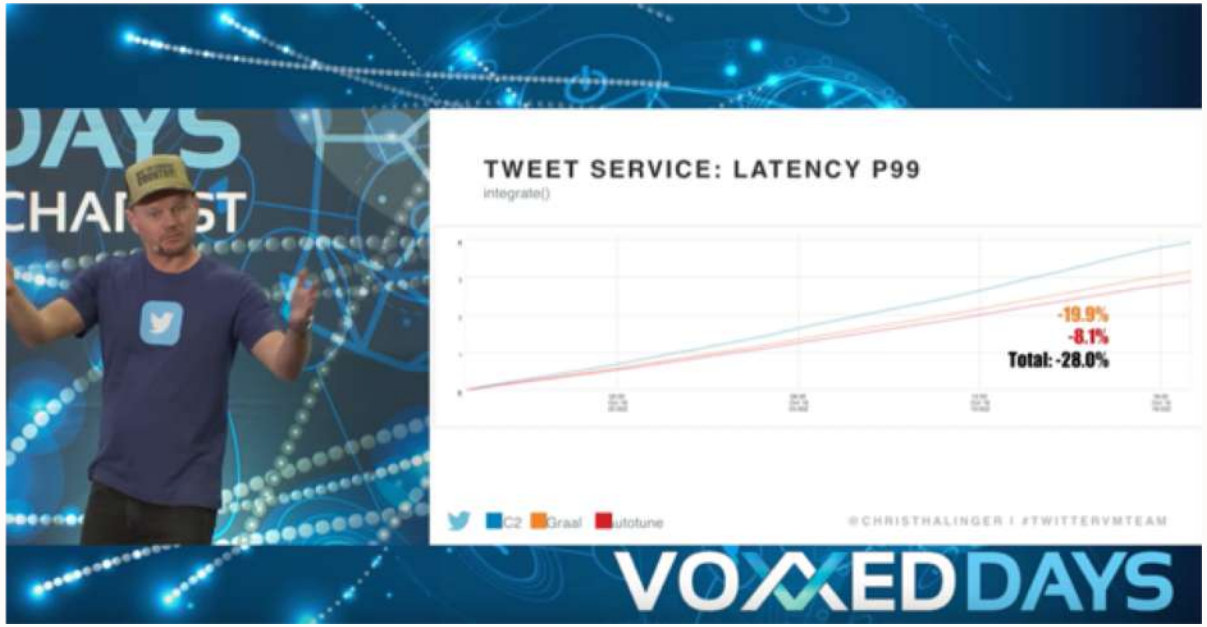
- Trace reflection, JNI, resource usage on Java HotSpot VM
  - Agent to record usage and produce configuration files for native images
    - java -agentlib:native-image-agent=config-output-dir=META-INF/native-image ...
  - Simplify the getting-started process
    - Everything that was executed on the Java HotSpot VM also works in the native image
  - Manual adjustment / addition will still be necessary
    - Unless you have an excellent test suite for your application
- Fun fact: Agent is a Java Native Image
  - JVMTI interface implemented using the low-level C interface of Native Image

GraalVM Native Image vs GraalVM JIT

- Use GraalVM Native Image when
  - Startup time matters
  - Memory footprint matters
    - Small to medium-sized heaps (100 MByte – a few GByte)
  - All code is known ahead of time

- Use GraalVM JIT when
  - Heaps size is large
    - Multiple GByte – TByte heap size
  - Classes are only known at run time

Twitter uses GraalVM compiler in production to run their Scala microservices



- Peak performance: +10%
- Garbage collection time: -25%
- Seamless migration

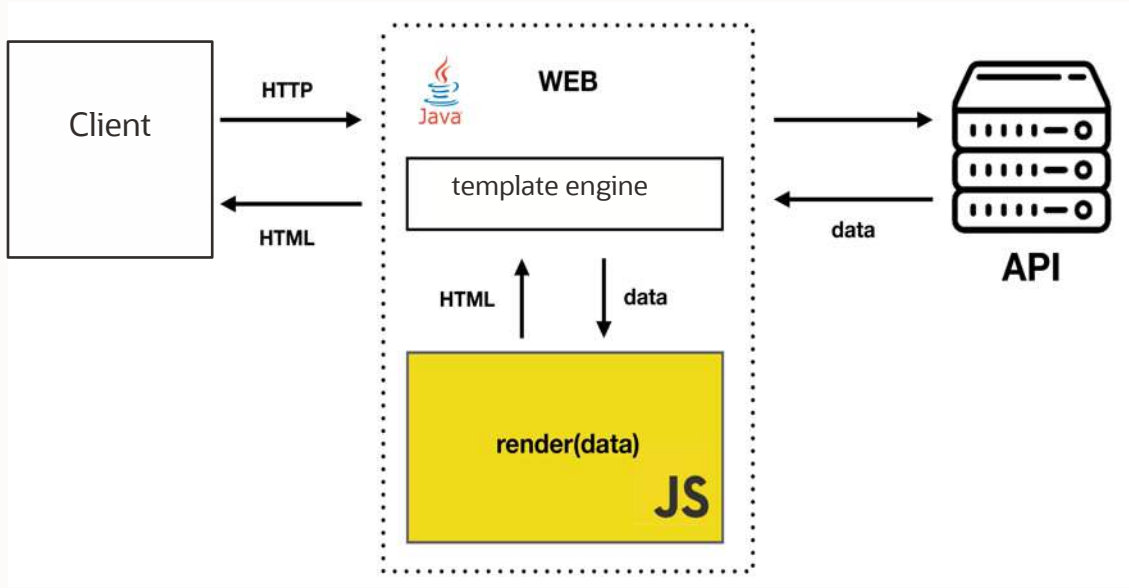
ORACLE
Cloud Infrastructure

The rich ecosystem of CUDA-X libraries is now available for GraalVM applications.

GPU kernels can be directly launched from GraalVM languages such as R, JavaScript, Scala and other JVM-based languages.

Learn more: https://devblogs.nvidia.com/grcuda-a-polyglot-language-binding-for-cuda-in-graalvm/

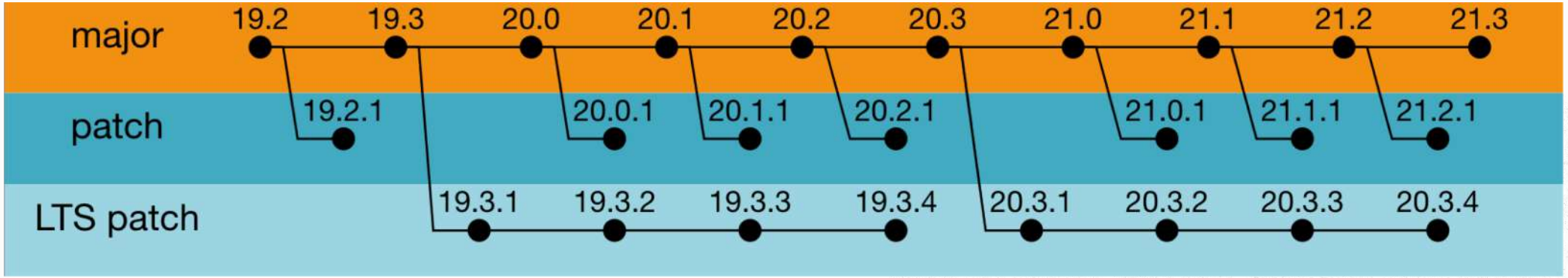
Odnoklassniki use GralVM in a production Java workload (70 mln users, ~600K req/min, ~7K servers) for React sever-side rendering



Project Roadmap

Version Roadmap

- Predictable release schedule;
- LTS releases: last major release of the year.



https://www.graalvm.org/docs/release-notes/version-roadmap

Recent Updates

- JDK-11 based builds;
- WebAssembly support;
- Support for JFR in Graal VisualVM;
- Throughput improvements in native images;
- LLVM toolchain;
- VS Code plugin preview;
- Class Initialization changes in native images.

What's next for GraalVM

- Extended ARM64 and Windows support;
- Low-latency, high-throughput, and parallel GC for native images;
- Work with the community to support important libraries;
- New languages and platforms;
- Your choice – contribute!

Contributions are welcome!

- How to contribute:
- Report an issue: https://github.com/oracle/graal/issues
- Submit your PR: https://github.com/oracle/graal/pulls
- Extend libraries support: graalvm.org/docs/reference-manual/compatibility/
- Contribute to documentation: https://www.graalvm.org/docs/

When you need GraalVM

1. High performance for abstractions of any language
2. Low footprint ahead-of-time mode for JVM-based languages
3. Convenient language interoperability and polyglot tooling
4. Simple embeddability in native and managed programs

What's next for you

- Download:

  graalvm.org/downloads

- Follow updates:

  @GraalVM / #GraalVM

- Get help:
- graalvm.org/slack-invitation/
- graalvm-users @oss.oracle.com

Thank you!

Alina Yurenko

@alina_yurenko