ORACLE

Maximizing Applications Performance with GraalVM

Alina Yurenko
Developer Advocate for GraalVM
@alina_yurenko
February 05, 2020

Safe harbor statement

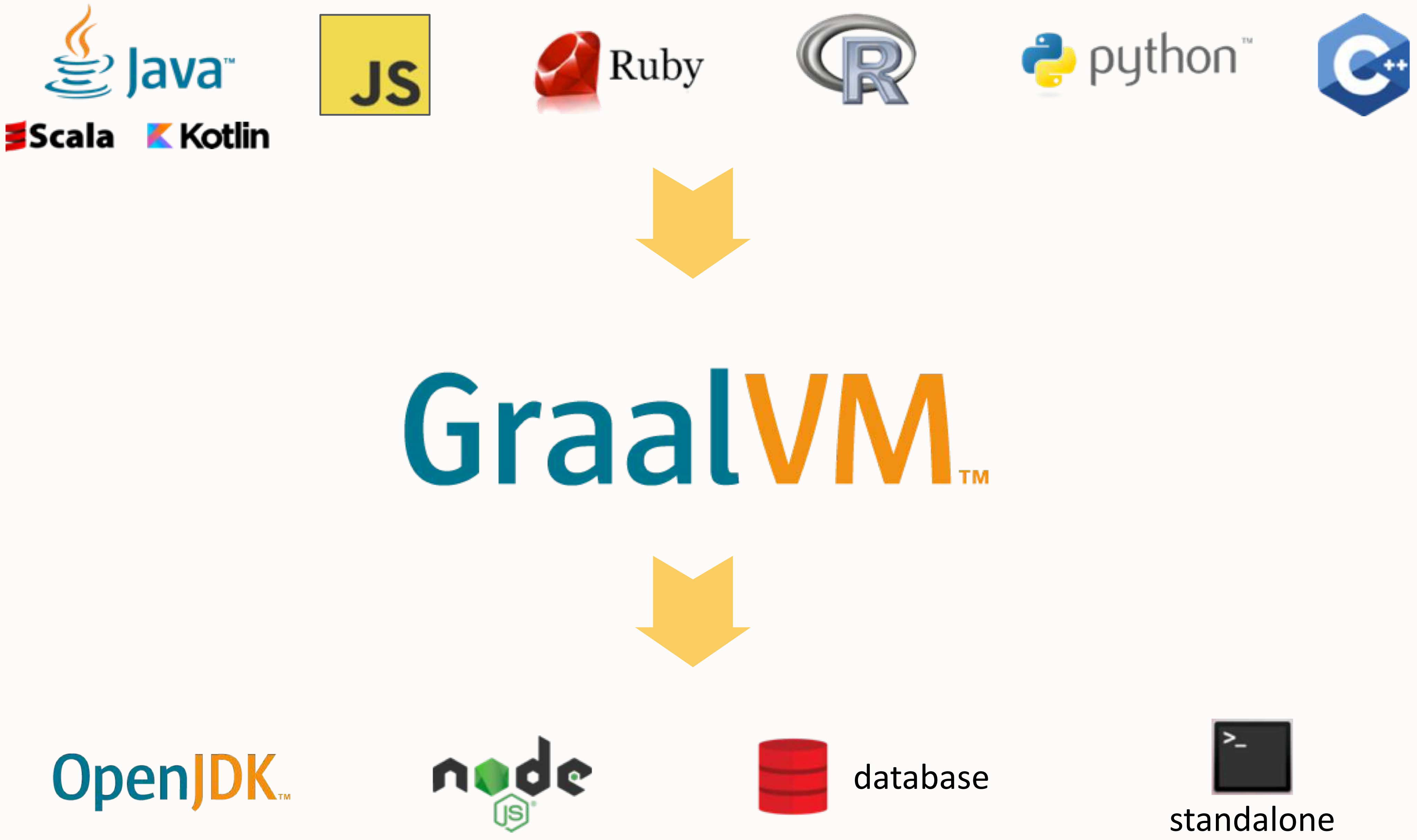
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, 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.

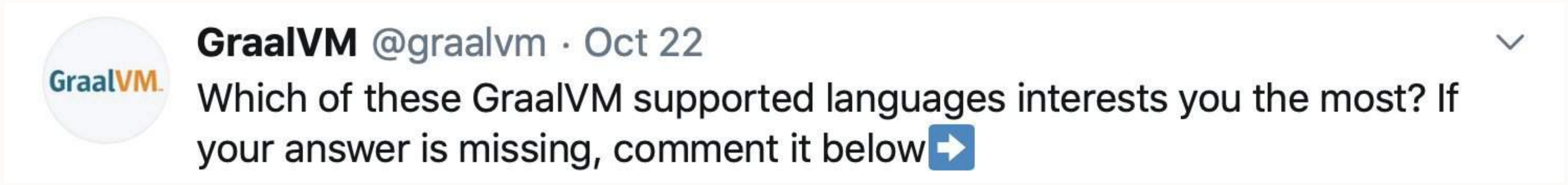
GraalVM Native Image technology (including Substrate VM) is Early Adopter technology. It is available only under an early adopter license and remains subject to potentially significant further changes, compatibility testing and certification.

Run programs faster anywhere

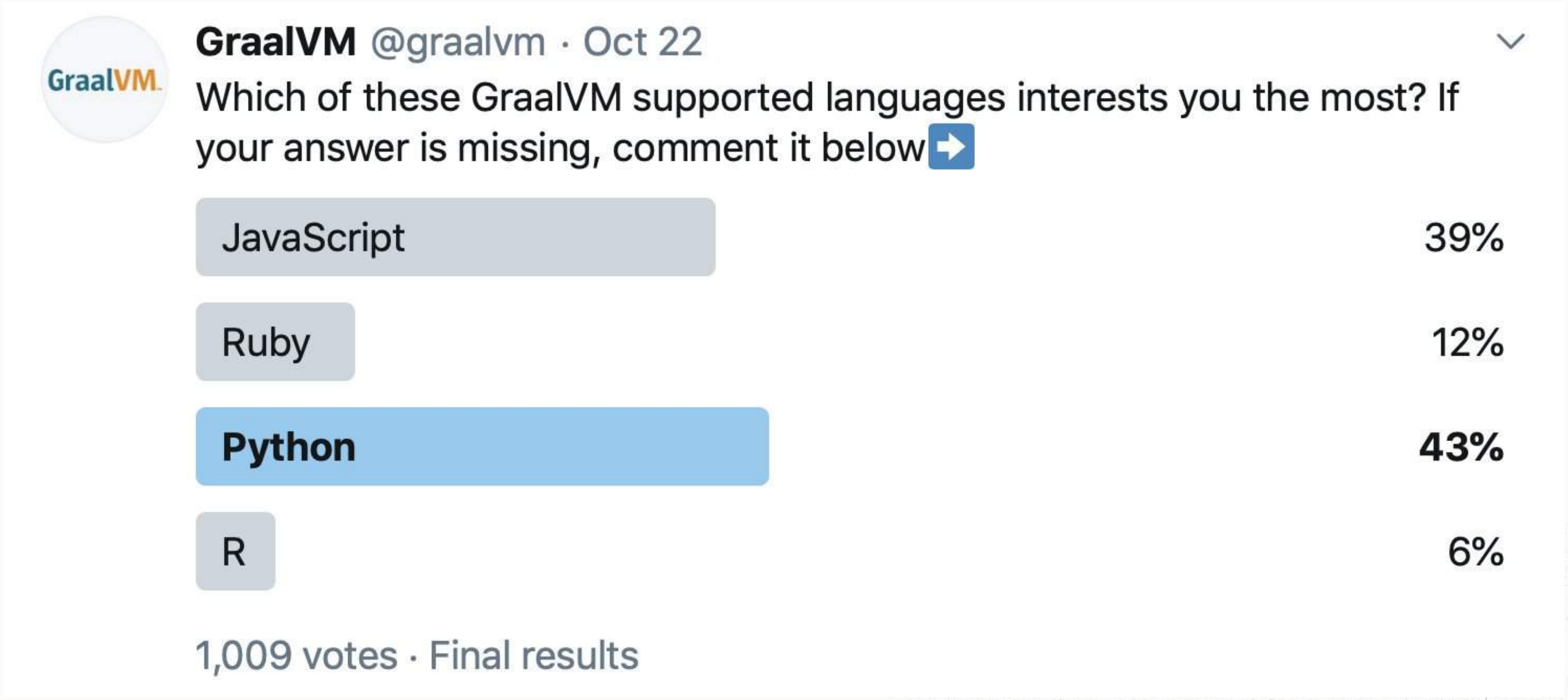
GraalVM™

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


Java
JS
Ruby
R
python
C++
Scala
Kotlin
GraalVM
OpenJDK
node JS
database
standalone


GraalVM @graalvm · Oct 22

Which of these GraalVM supported languages interests you the most? If your answer is missing, comment it below➡️



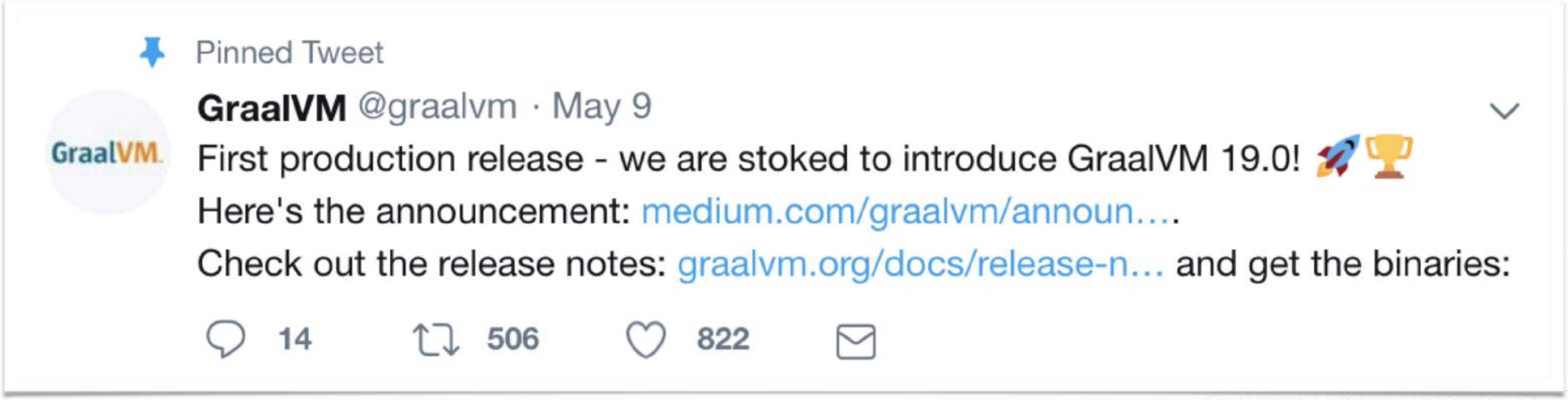
GraalVM @graalvm · Oct 22

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

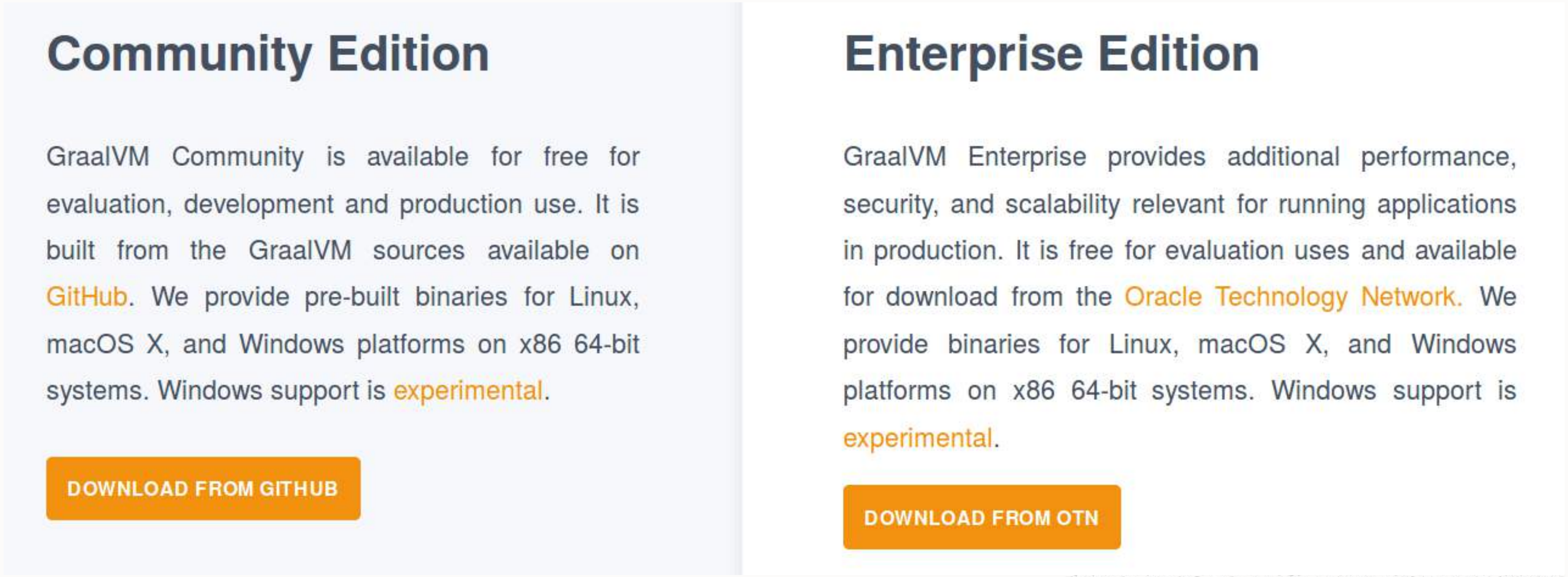
Production-ready!🎉


Pinned Tweet

GraalVM @graalvm · May 9

First production release - we are stoked to introduce GraalVM 19.0! 🚀🏆
Here's the announcement: medium.com/graalvm/announ….
Check out the release notes: graalvm.org/docs/release-n… and get the binaries:

14 506 822



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

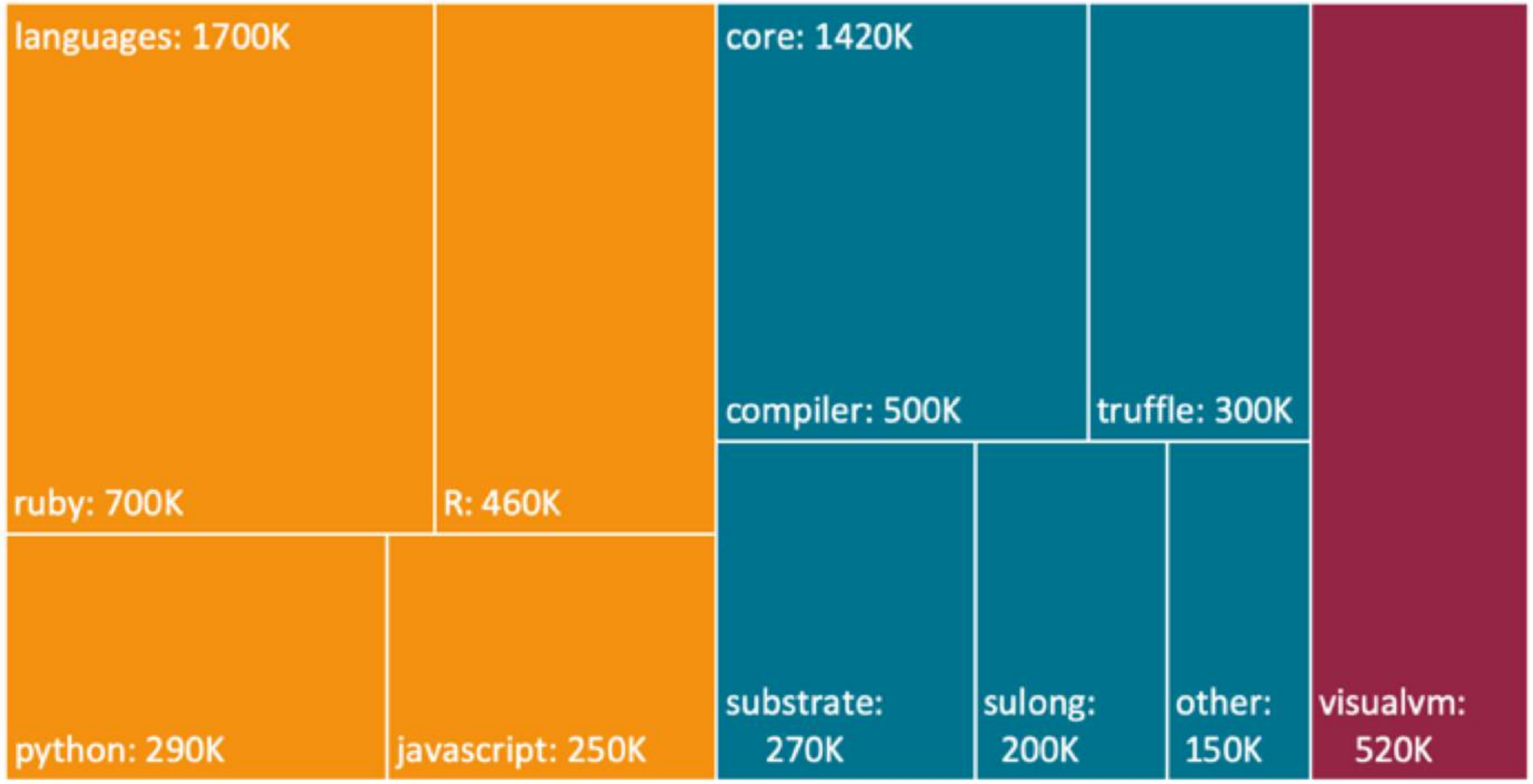
Enterprise Edition

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

Performance Metrics

How to measure performance?

- Throughput
- Latency
- Capacity
- Utilization
- Efficiency
- Scalability
- Degradation

https://learning.oreilly.com/library/view/optimizing-java -
by Chris Newland, James Gough, Benjamin J Evans

Optimizing performance with GraalVM

Java™

Scala Groovy Kotlin

GraalVM™

JIT

```
java MyMainClass
```

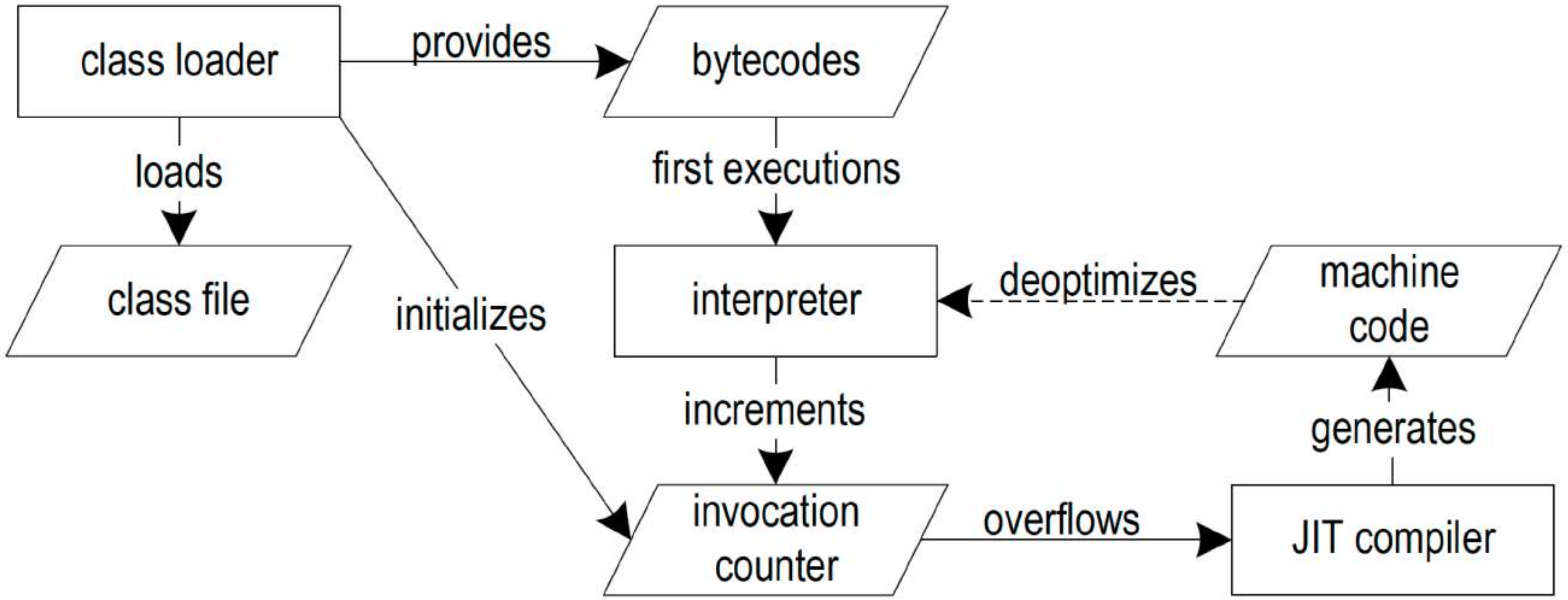
AOT

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

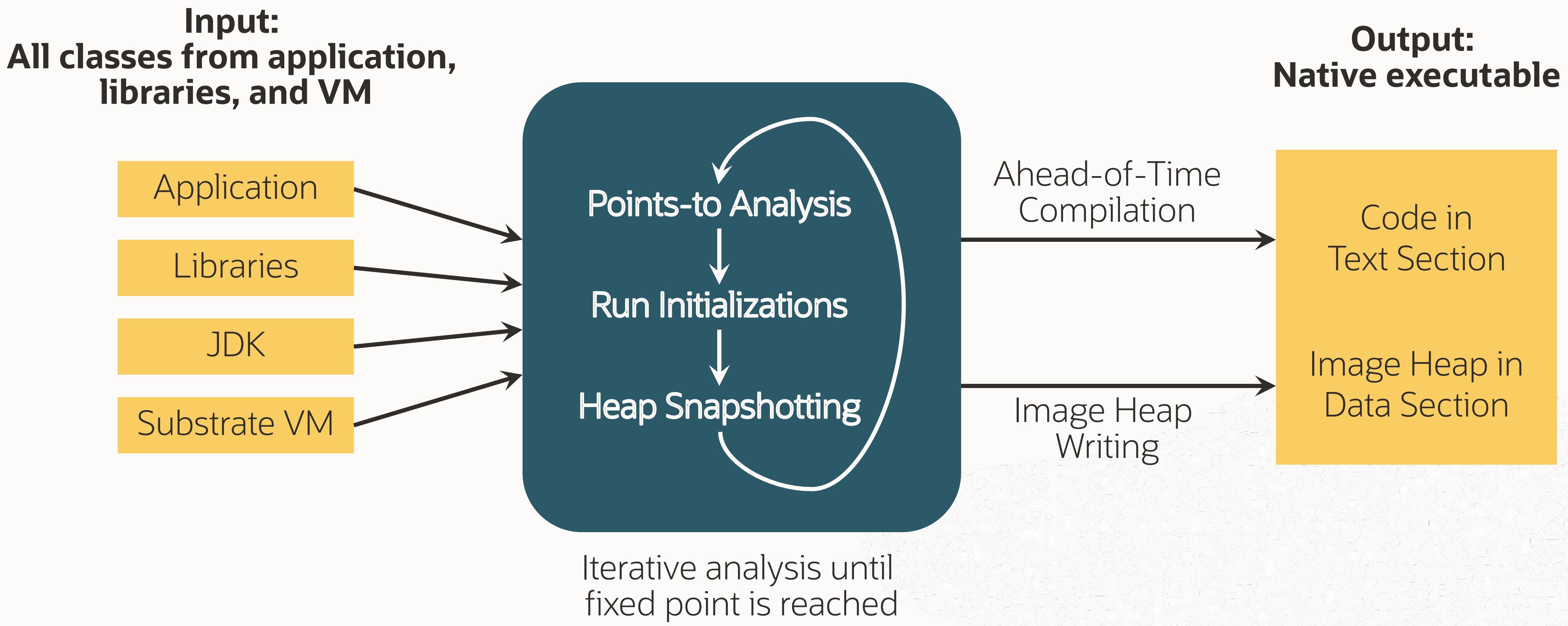
GraalVM Native Images

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

Java Dynamic Execution



Native Image Build Process



Startup Performance

AOT vs JIT: Startup Time

JIT

- Load JVM executable
- Load classes from file system
- Verify bytecodes
- Start interpreting
- Run static initializers
- First tier compilation (C1)
- Gather profiling feedback
- Second tier compilation (GraalVM or C2)
- Finally run with best machine code

AOT

- Load executable with prepared heap
- Immediately start with best machine code

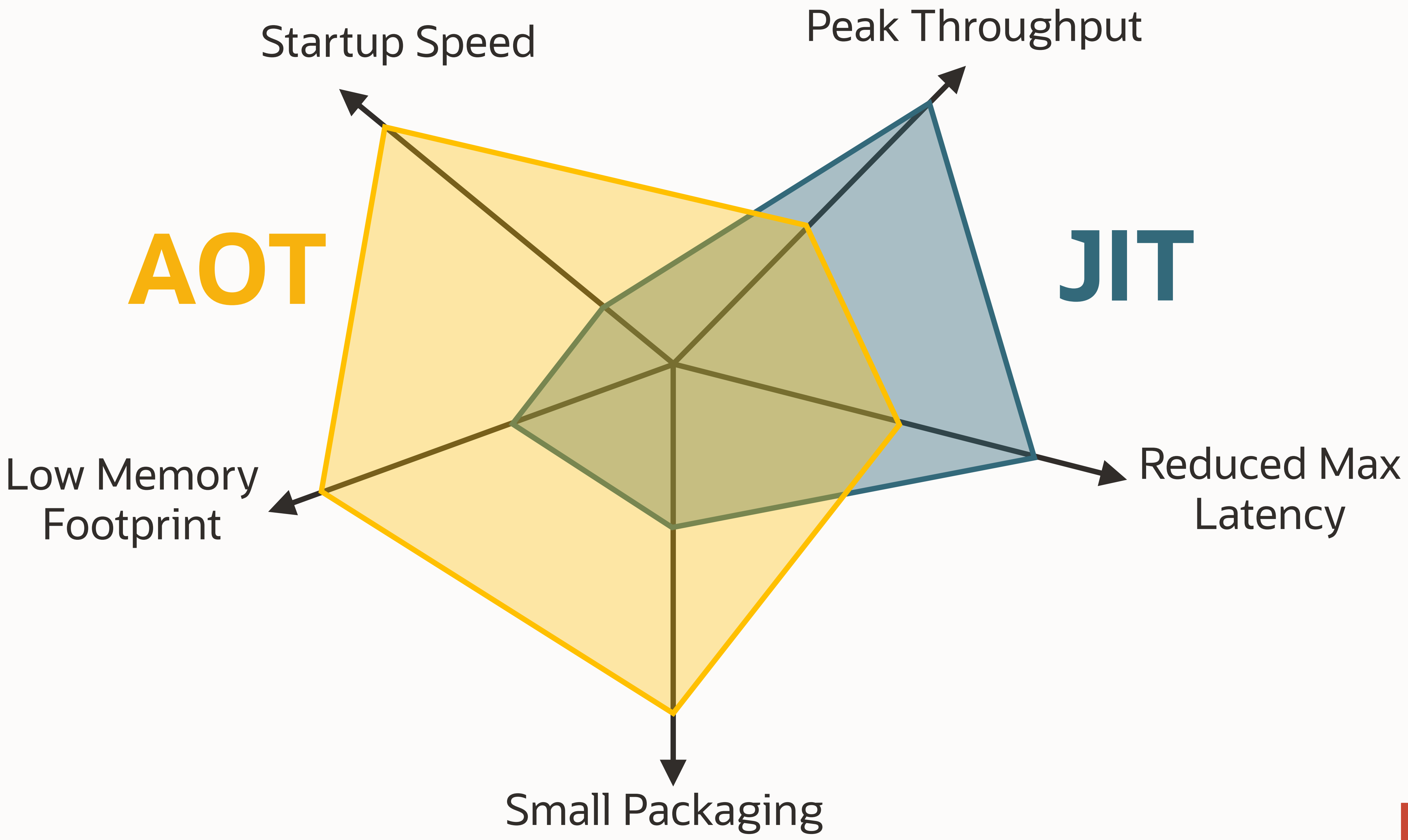
AOT vs JIT: Memory Footprint

JIT

- Loaded JVM executable
- Application data
- Loaded bytecodes
- Reflection meta-data
- Code cache
- Profiling data
- JIT compiler data structures

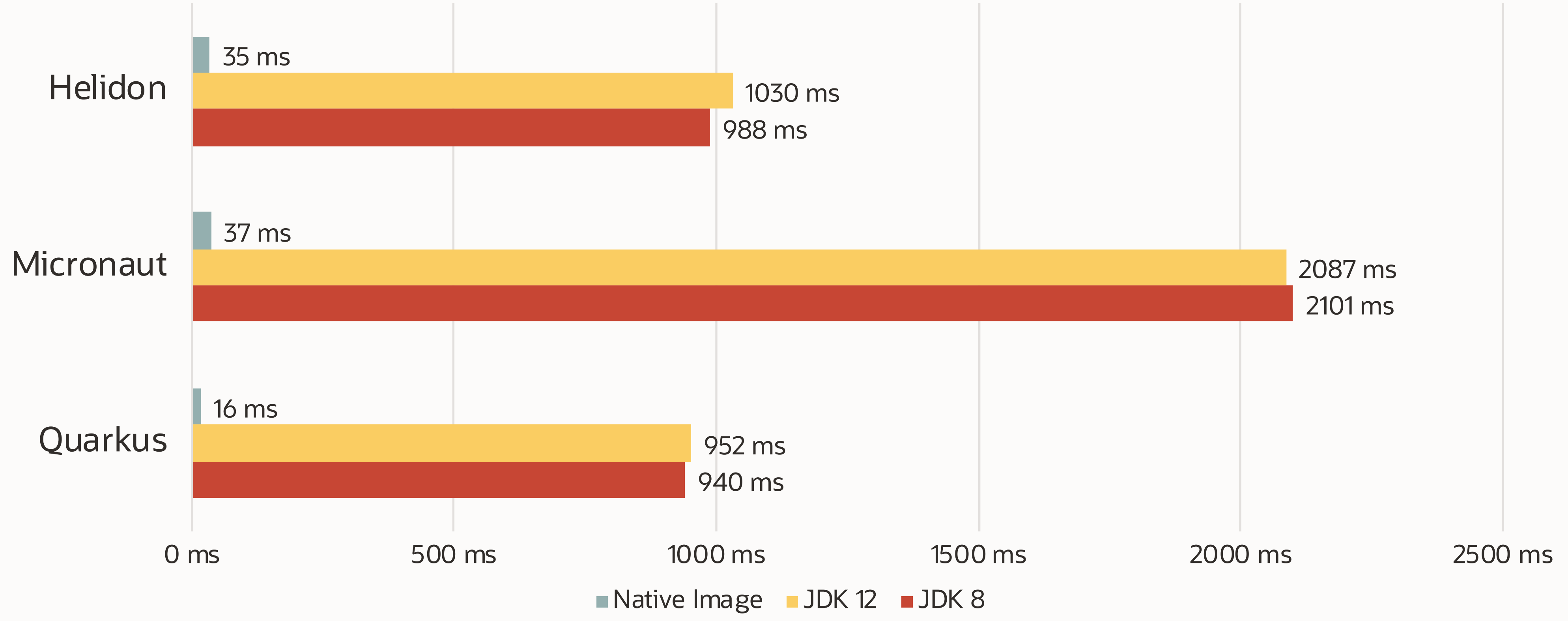
AOT

- Loaded application executable
- Application data

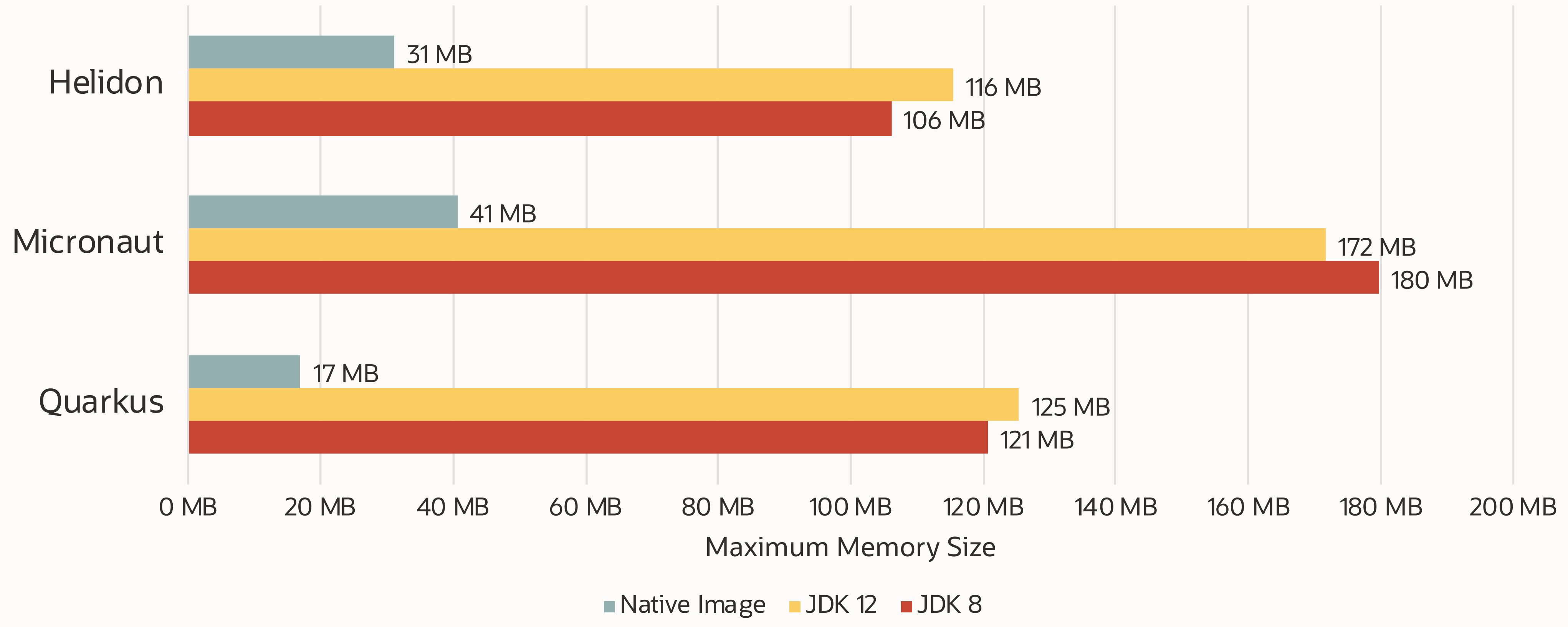

Startup Speed
Peak Throughput
AOT
JIT
Low Memory Footprint
Reduced Max Latency
Small Packaging

Demo: startup and memory footprint

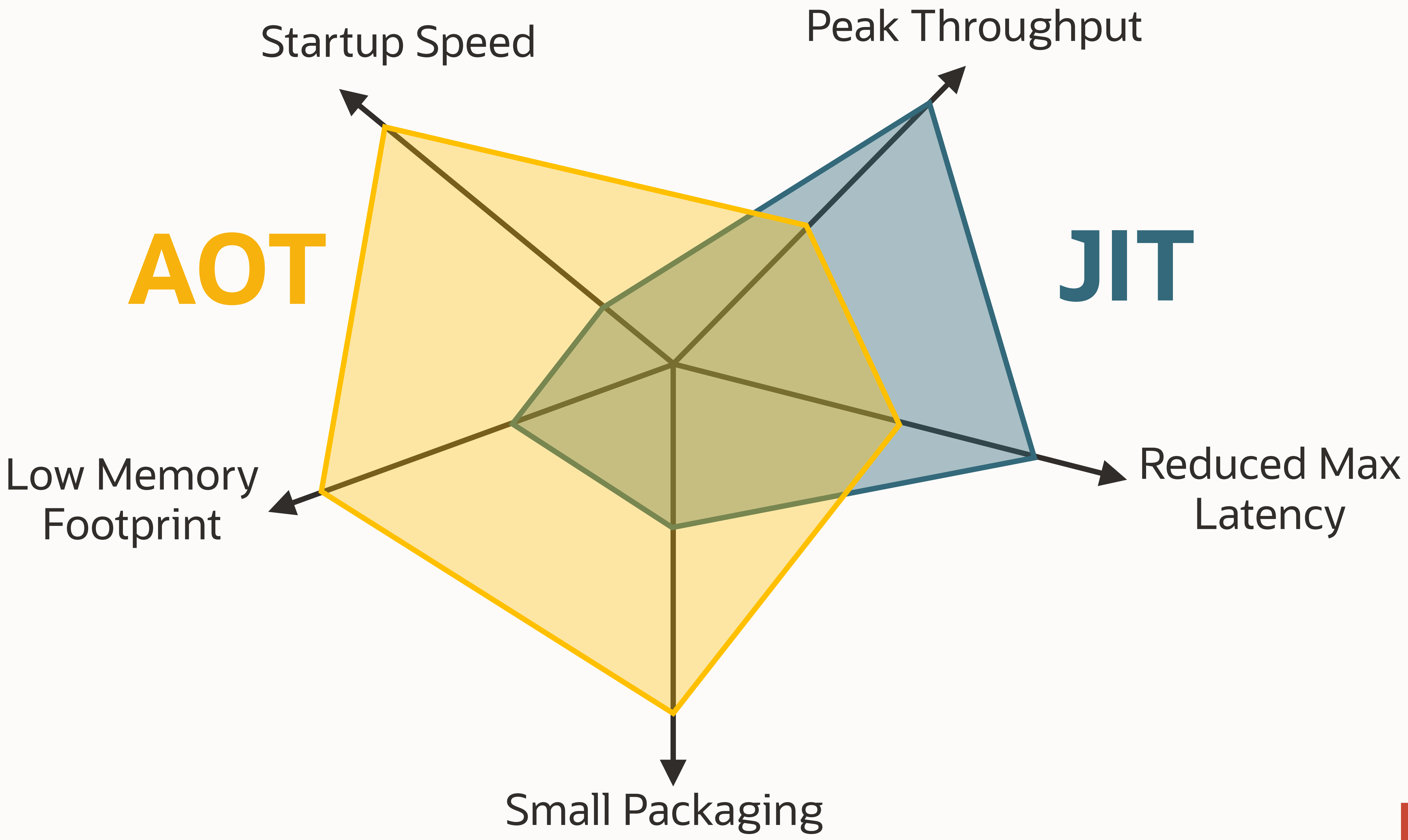
Microservice Frameworks: Startup Time



Microservice Frameworks: Memory Usage



Peak Performance


Startup Speed
Peak Throughput
AOT
JIT
Low Memory Footprint
Reduced Max Latency
Small Packaging

AOT vs JIT: Peak Throughput

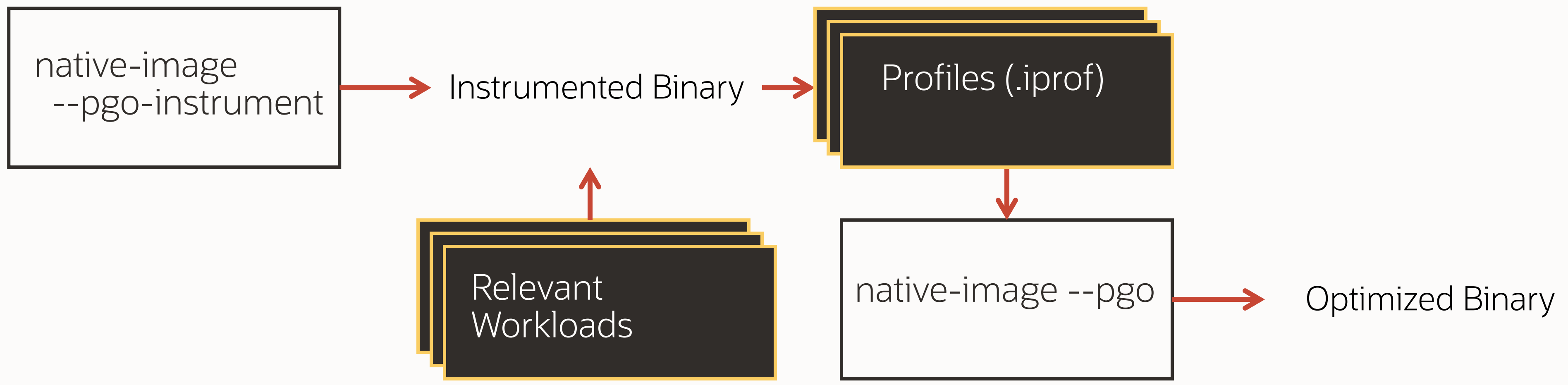
JIT

- Profiling at startup enables better optimizations
- Can make optimistic assumptions about the profile and deoptimize

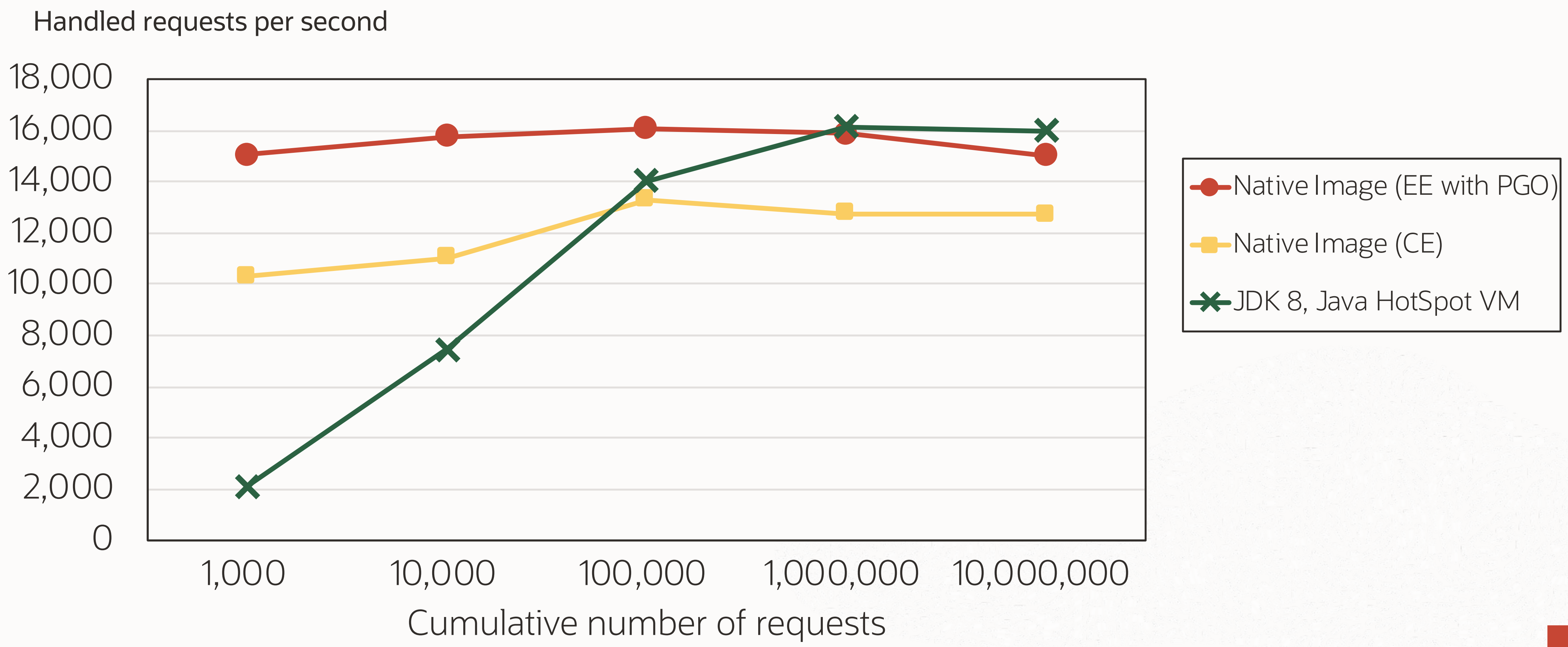
- AOT

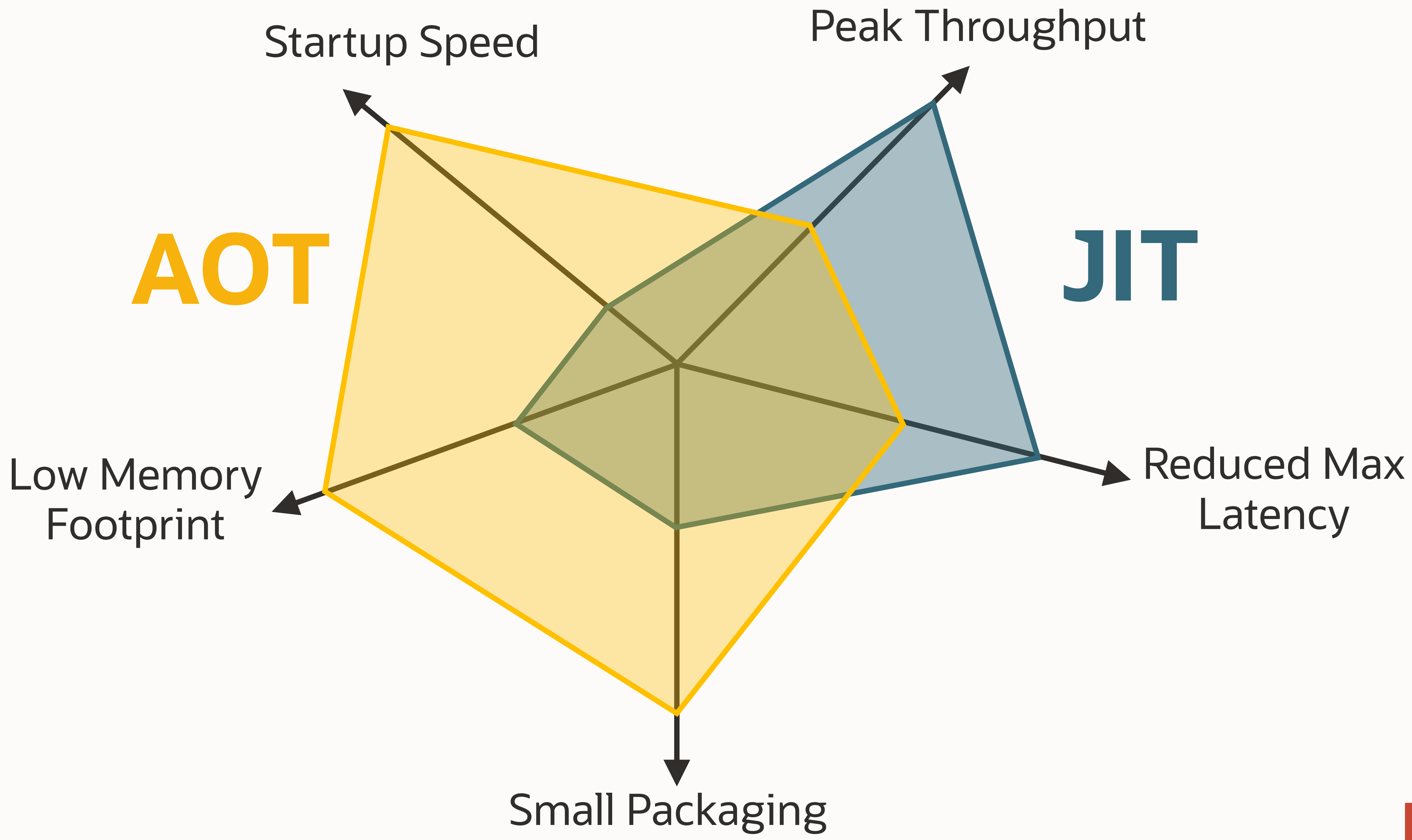
- Needs to handle all cases in machine code
- Predictable performance
- Profile-guided optimizations help

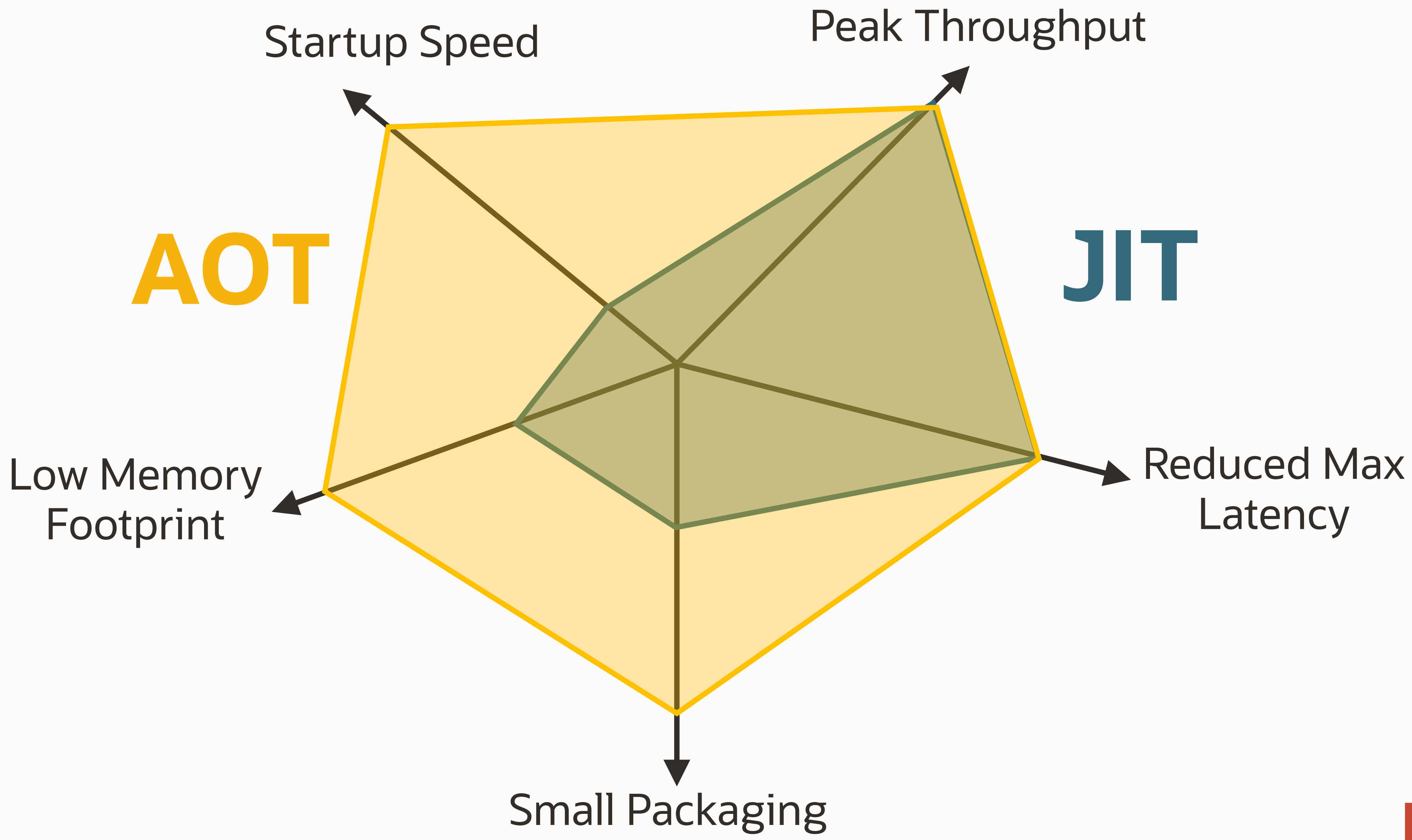
Profile-Guided Optimizations (PGO)



AOT vs JIT: Peak Throughput

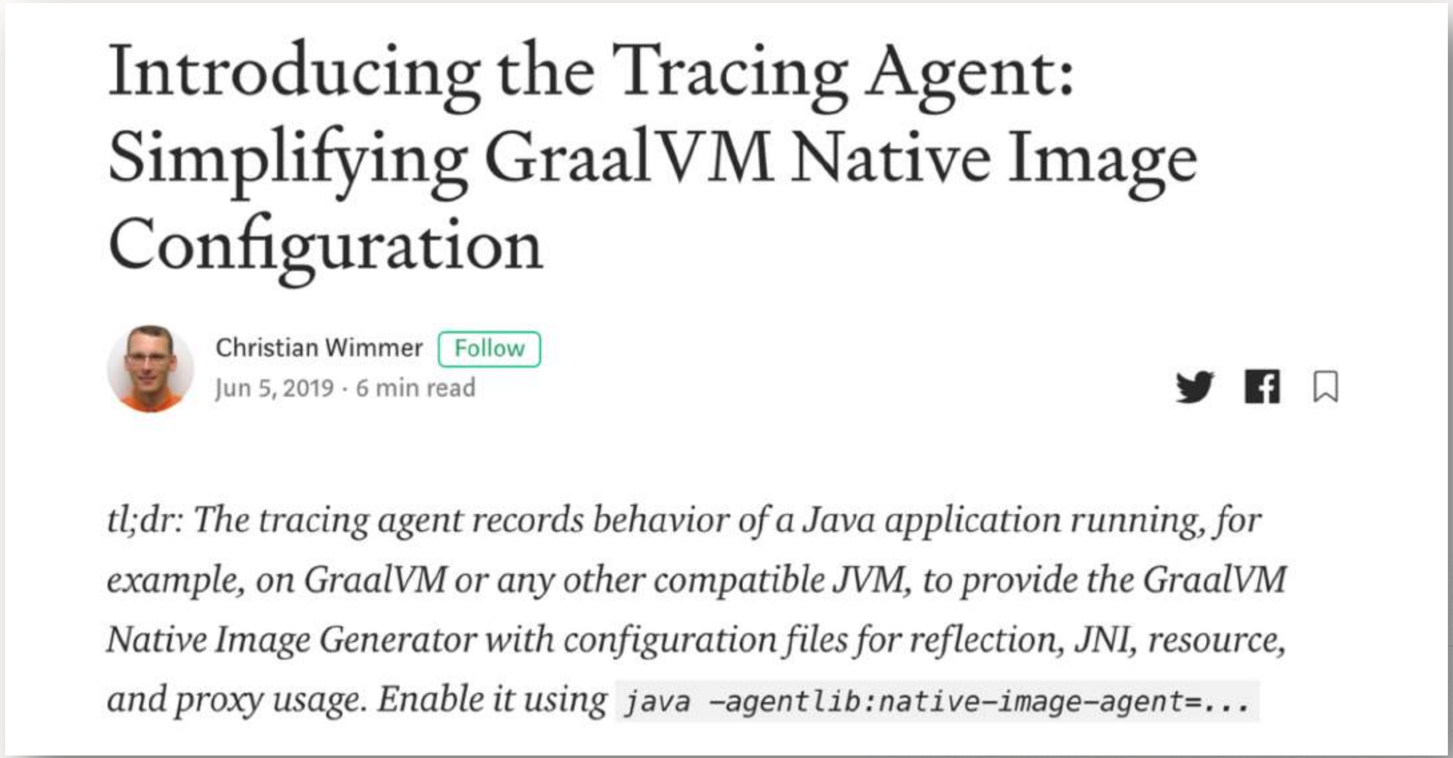



Startup Speed
Peak Throughput
AOT
JIT
Low Memory Footprint
Reduced Max Latency
Small Packaging


Startup Speed
Peak Throughput
AOT
JIT
Low Memory Footprint
Reduced Max Latency
Small Packaging

GraalVM native image for real-world projects

Simplify native image configuration



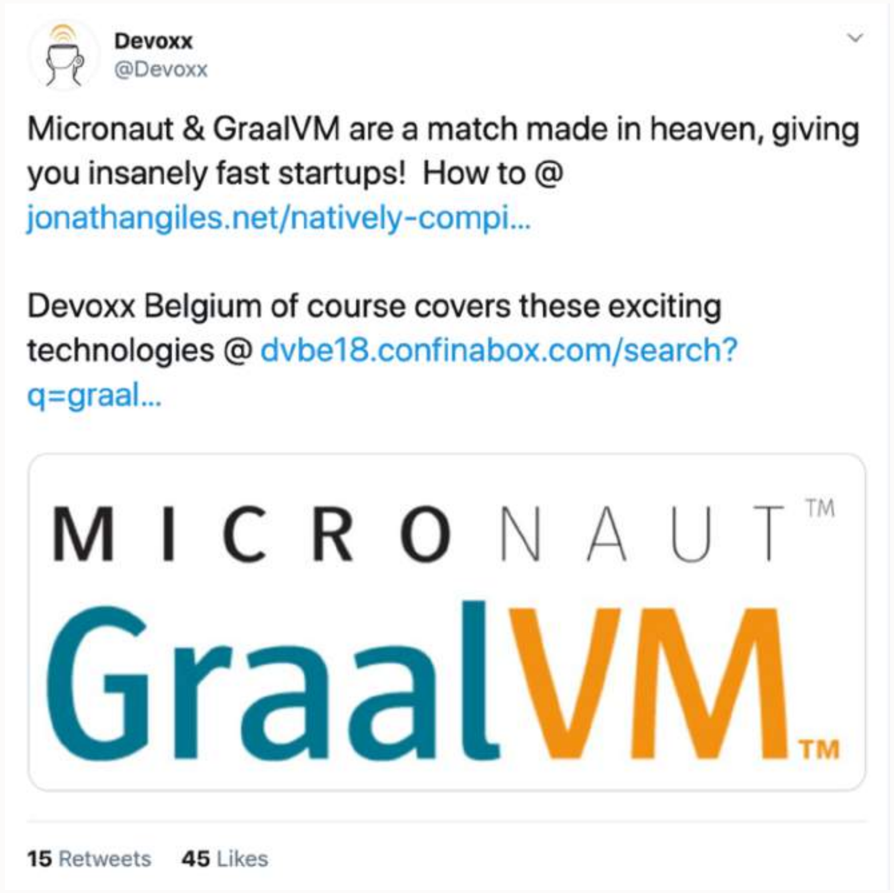
Introducing the Tracing Agent: Simplifying GraalVM Native Image Configuration

Christian Wimmer Follow
Jun 5, 2019 · 6 min read

tl;dr: The tracing agent records behavior of a Java application running, for example, on GraalVM or any other compatible JVM, to provide the GraalVM Native Image Generator with configuration files for reflection, JNI, resource, and proxy usage. Enable it using `java -agentlib:native-image-agent=...`

medium.com/graalvm/introducing-the-tracing-agent-simplifying-graalvm-native-image-configuration-c3b56c486271

Micronaut


Devoxx
@Devoxx

Micronaut & GraalVM are a match made in heaven, giving you insanely fast startups! How to @ jonathangiles.net/natively-compi...

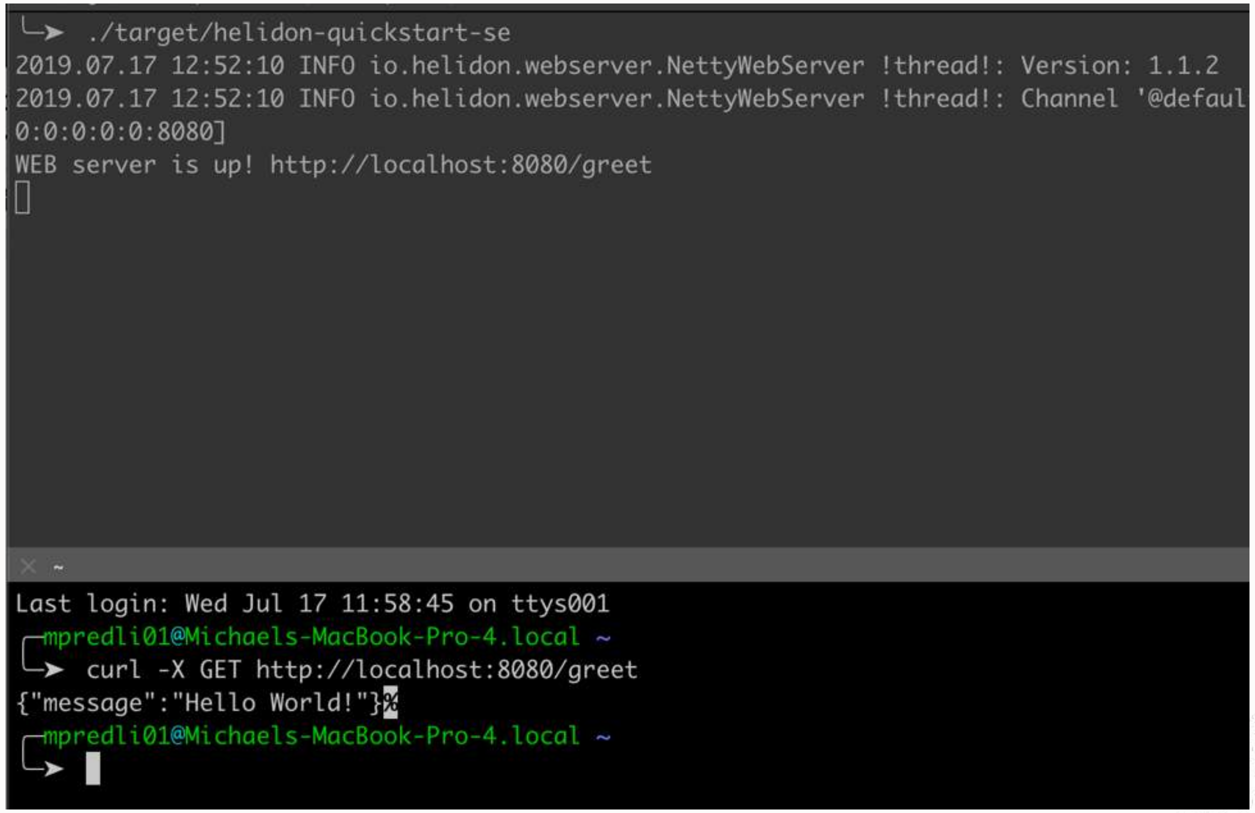
Devoxx Belgium of course covers these exciting technologies @ dvbe18.confinabox.com/search?q=graal...

15 Retweets 45 Likes

Create your first Micronaut GraalVM application:

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

Helidon



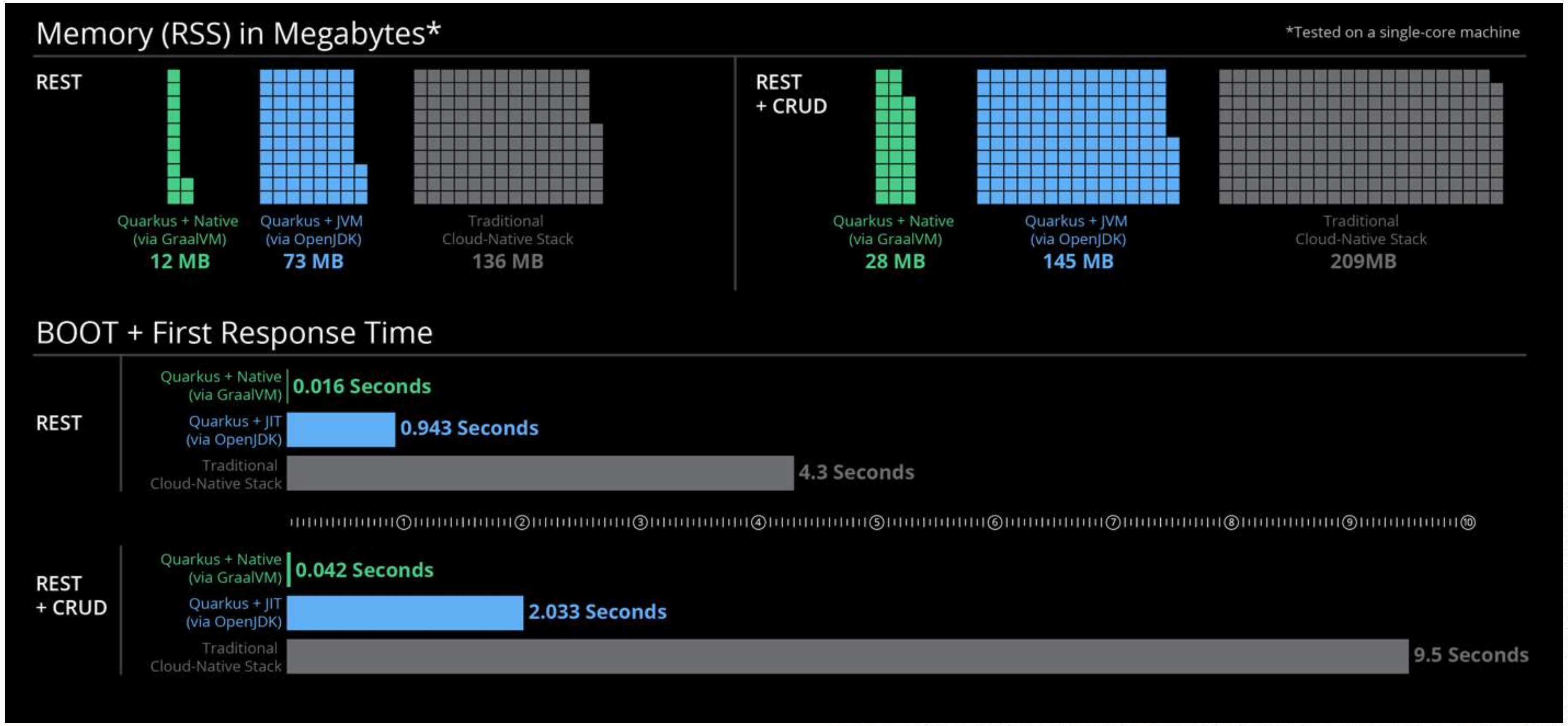
```
└➤  ./target/helidon-quickstart-se
2019.07.17 12:52:10 INFO io.helidon.webserver.NettyWebServer !thread!: Version: 1.1.2
2019.07.17 12:52:10 INFO io.helidon.webserver.NettyWebServer !thread!: Channel '@defaul
0:0:0:0:0:8080]
WEB server is up! http://localhost:8080/greet
```

```
Last login: Wed Jul 17 11:58:45 on ttys001
┌mpredli01@Michaels-MacBook-Pro-4.local ~
└➤  curl -X GET http://localhost:8080/greet
{"message":"Hello World!"}%
┌mpredli01@Michaels-MacBook-Pro-4.local ~
└➤
```

Helidon and GraalVM:

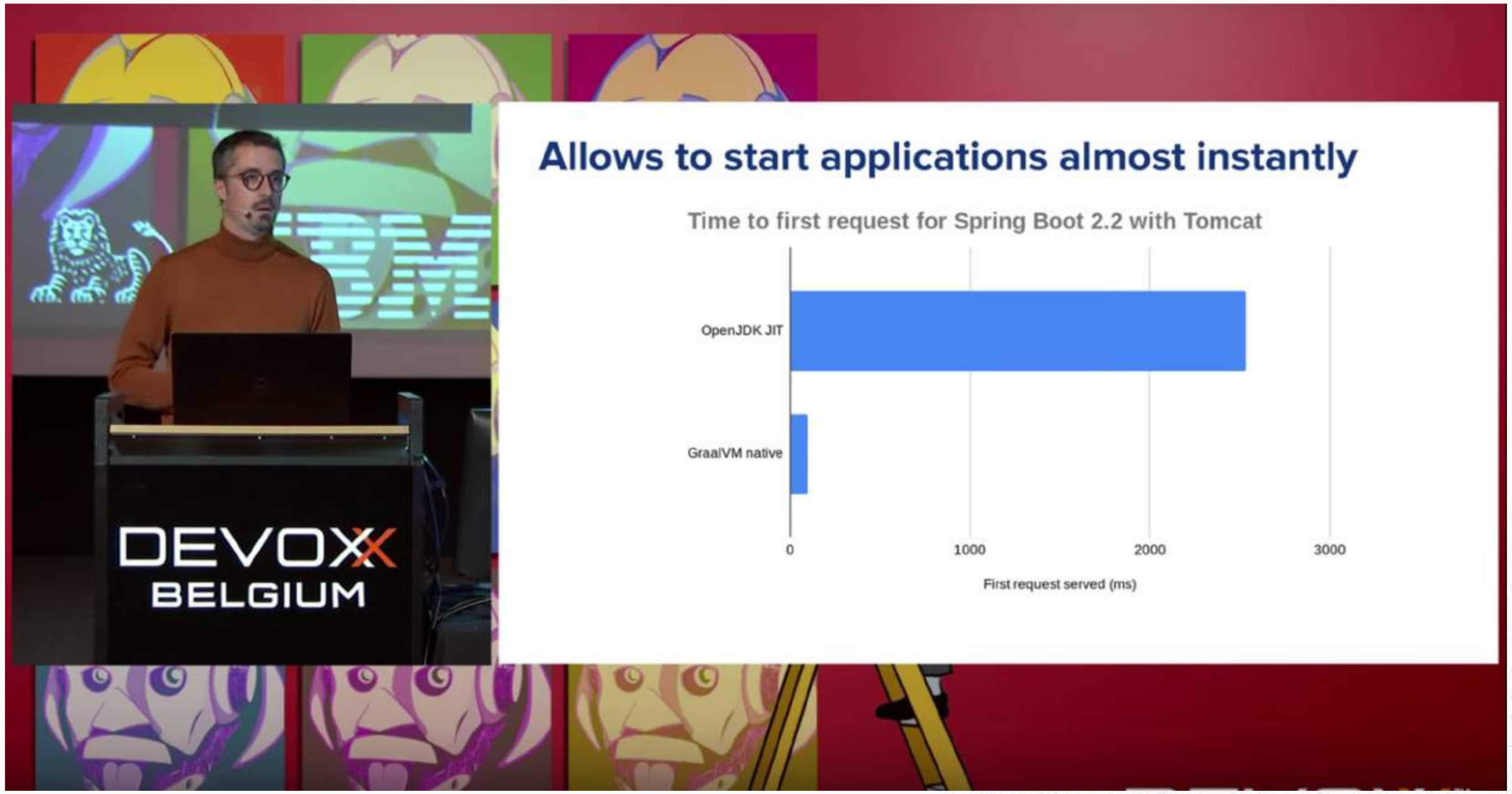
https://helidon.io/docs/latest/#/guides/36_graalnative

Quarkus



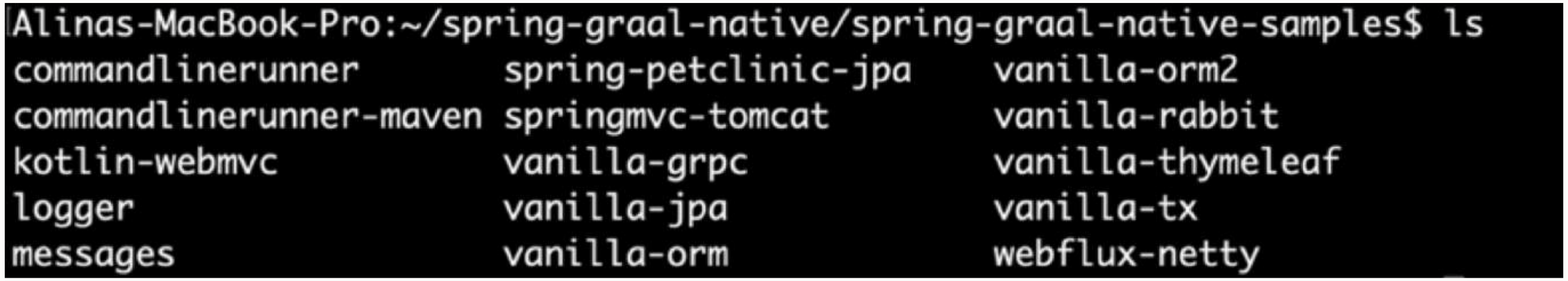
https://quarkus.io/guides/building-native-image

Spring Boot Applications as GraalVM Native Images


DEVOXX
BELGIUM
Allows to start applications almost instantly
Time to first request for Spring Boot 2.2 with Tomcat
OpenJDK JIT
GraalVM native
0
1000
2000
3000
First request served (ms)

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

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

GraalVM Language Ecosystem

Multiplicative Value-Add of GraalVM Ecosystem

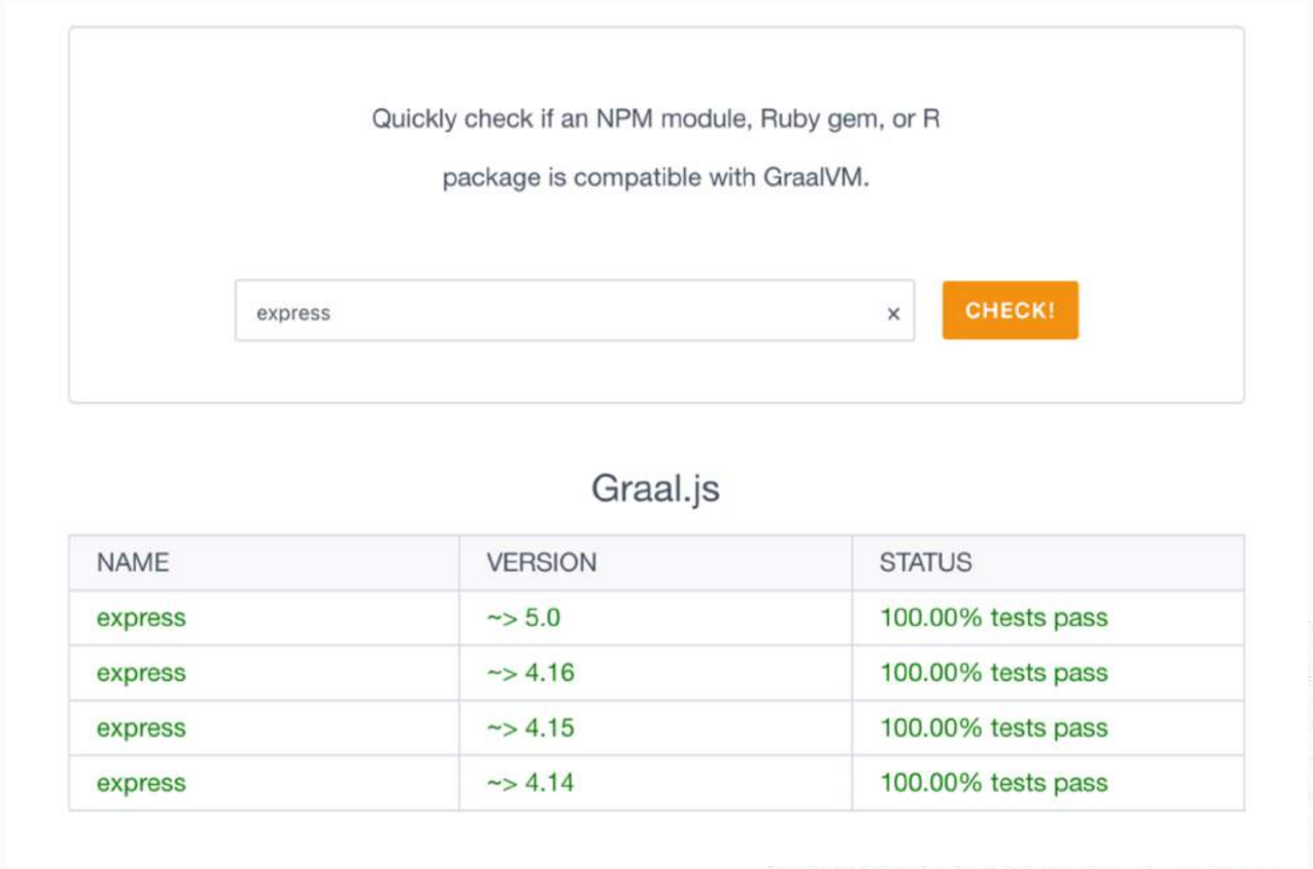
Languages	*	GraalVM	*	Embeddings
Java JavaScript Ruby R Python C/C++, FORTRAN, …	❯	Optimizations Tooling Interoperability Security	❮	HotSpot JVM Oracle RDBMS Node.js Standalone Spark …

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

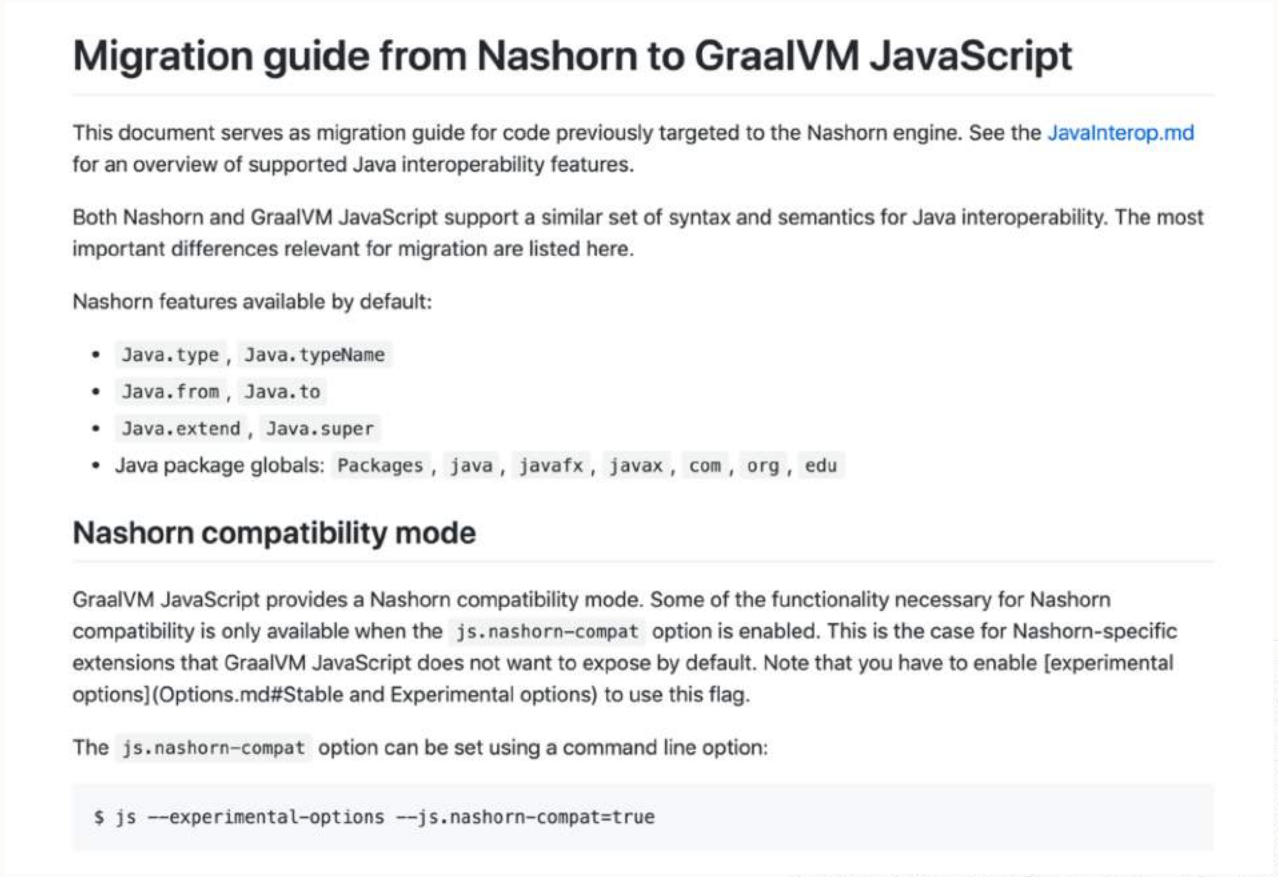

Quickly check if an NPM module, Ruby gem, or R package is compatible with GraalVM.

Graal.js

NAME	VERSION	STATUS
express	~> 5.0	100.00% tests pass
express	~> 4.16	100.00% tests pass
express	~> 4.15	100.00% tests pass
express	~> 4.14	100.00% tests pass

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

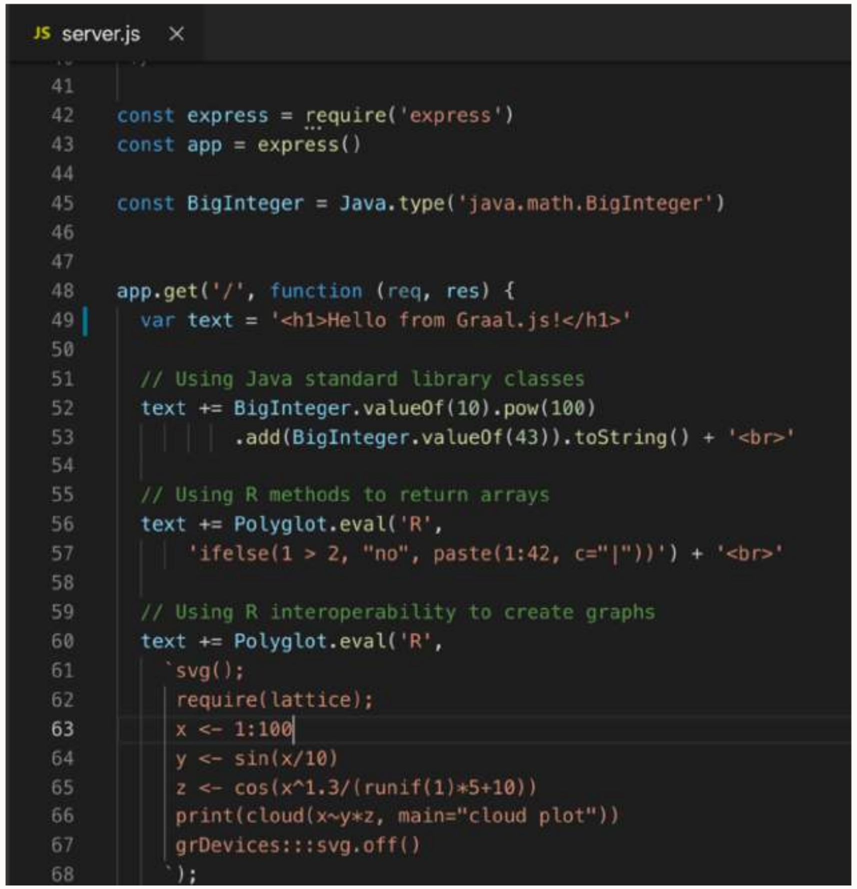
GraalVM JavaScript provides a Nashorn compatibility mode. Some of the functionality necessary for Nashorn compatibility is only available when the `js.nashorn-compat` option is enabled. This is the case for Nashorn-specific extensions that GraalVM JavaScript does not want to expose by default. Note that 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:

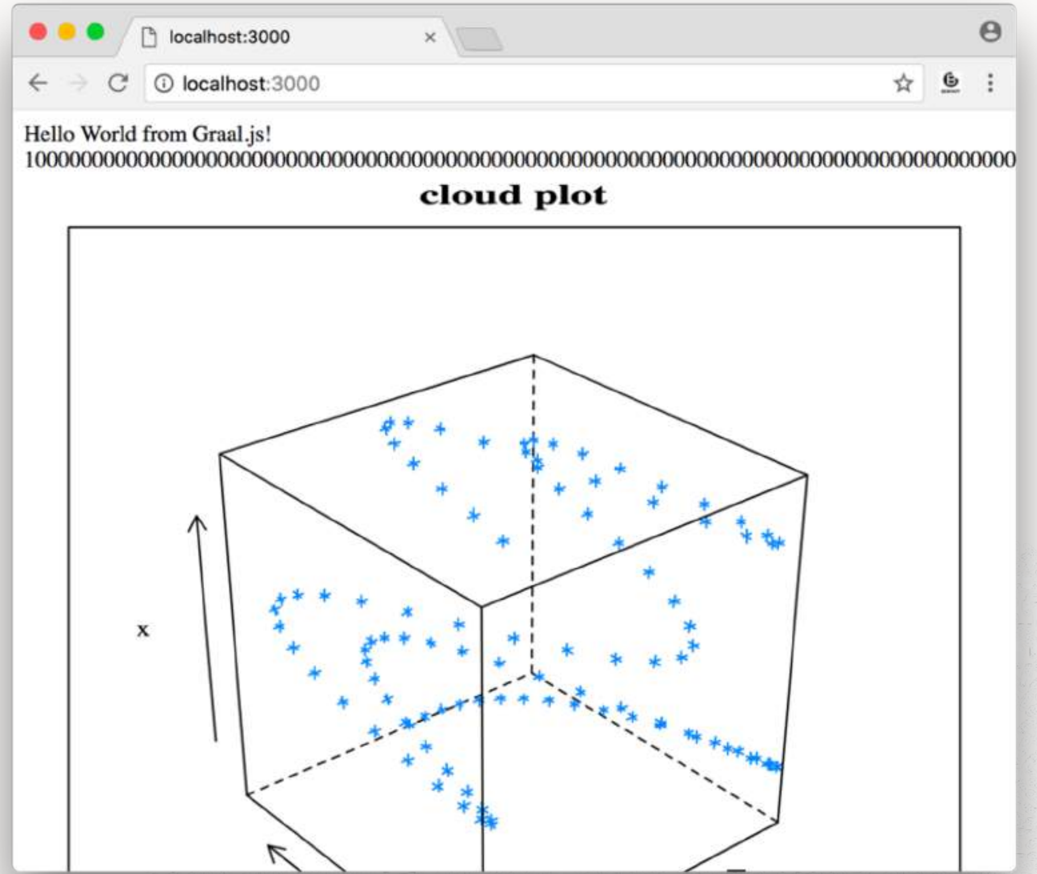
```
$ js --experimental-options --js.nashorn-compat=true
```

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

JavaScript + Java + R



```
const express = require('express')
const app = express()

const BigInteger = Java.type('java.math.BigInteger')


app.get('/', function (req, res) {
  var text = '<h1>Hello from Graal.js!</h1>'

  // Using Java standard library classes
  text += BigInteger.valueOf(10).pow(100)
          .add(BigInteger.valueOf(43)).toString() + '<br>'

  // Using R methods to return arrays
  text += Polyglot.eval('R',
      'ifelse(1 > 2, "no", paste(1:42, c="|"))') + '<br>'

  // Using R interoperability to create graphs
  text += Polyglot.eval('R',
    `svg();
     require(lattice);
     x <- 1:100
     y <- sin(x/10)
     z <- cos(x^1.3/(runif(1)*5+10))
     print(cloud(x~y*z, main="cloud plot"))
     grDevices:::svg.off()
    `);
```



localhost:3000

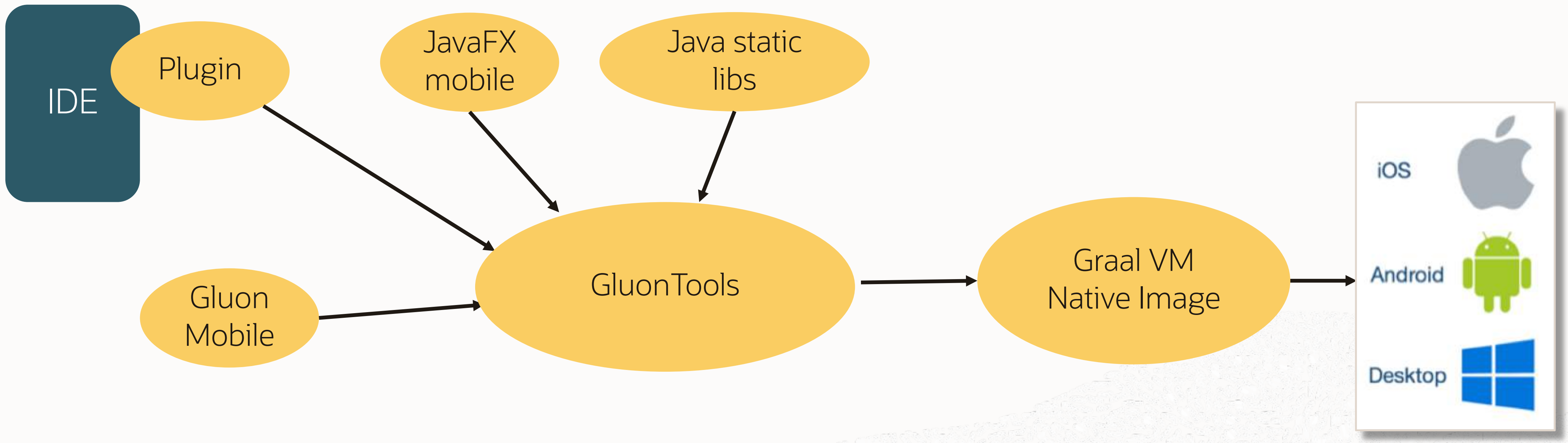
```
Hello World from Graal.js!
10000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

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
```

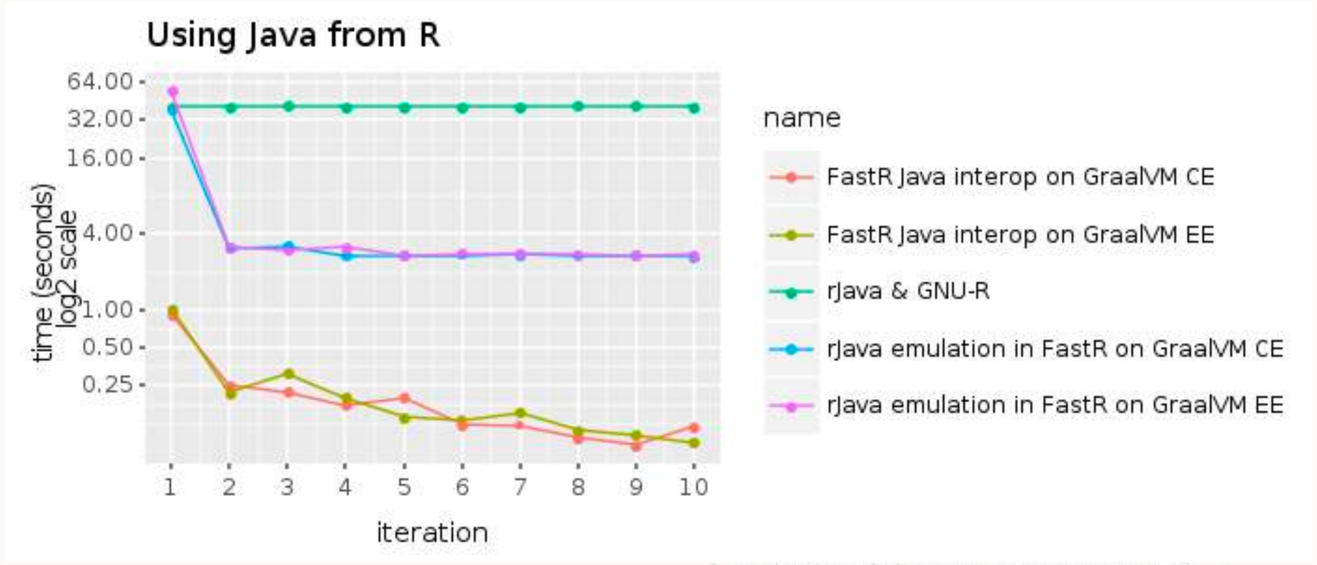
```
SQL> select validator.isEmail('hello.world@oracle.com') from dual;

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

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

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

Do even more with GraalVM: Cross-Platform Development



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



warm-up curves, i.e. lower is better, of rJava on GNU-R, FastR and the native Java interoperability in FastR

GraalVM Python

- Python 3 implementation;
- High performance;
- Focus on supporting SciPy and its constituent libraries;
- Easy interop with Java and the rest of GraalVM languages.

```
$ graalpython [options] [-c cmd | filename]
```

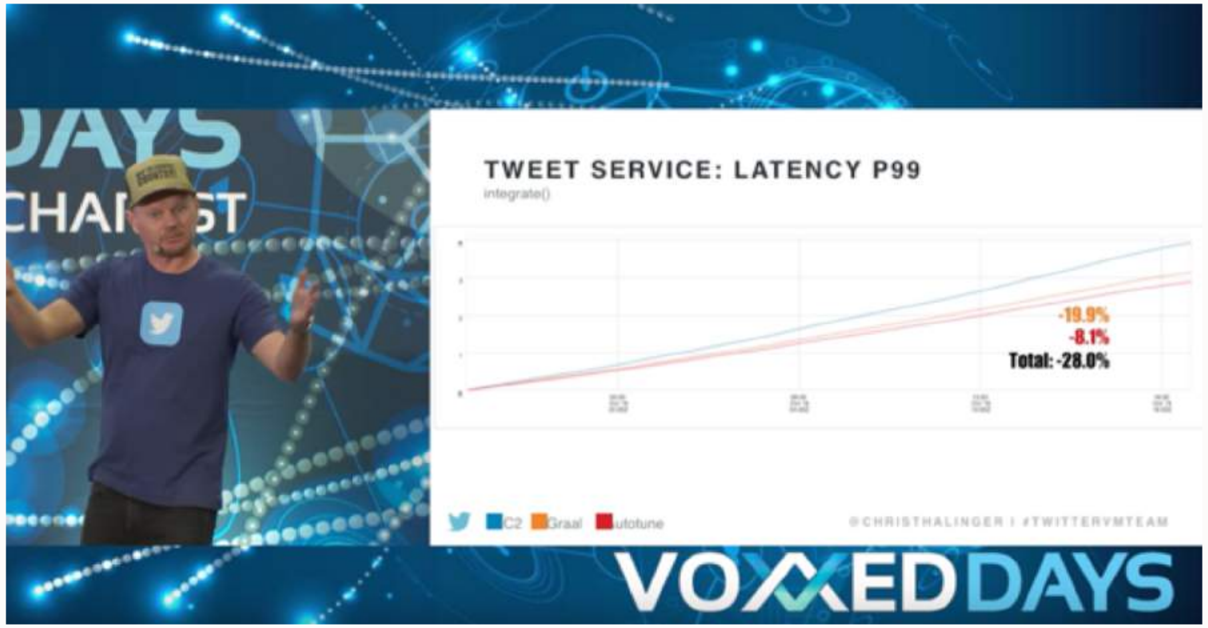
Using grCUDA to Access Nvidia GPUs

- 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

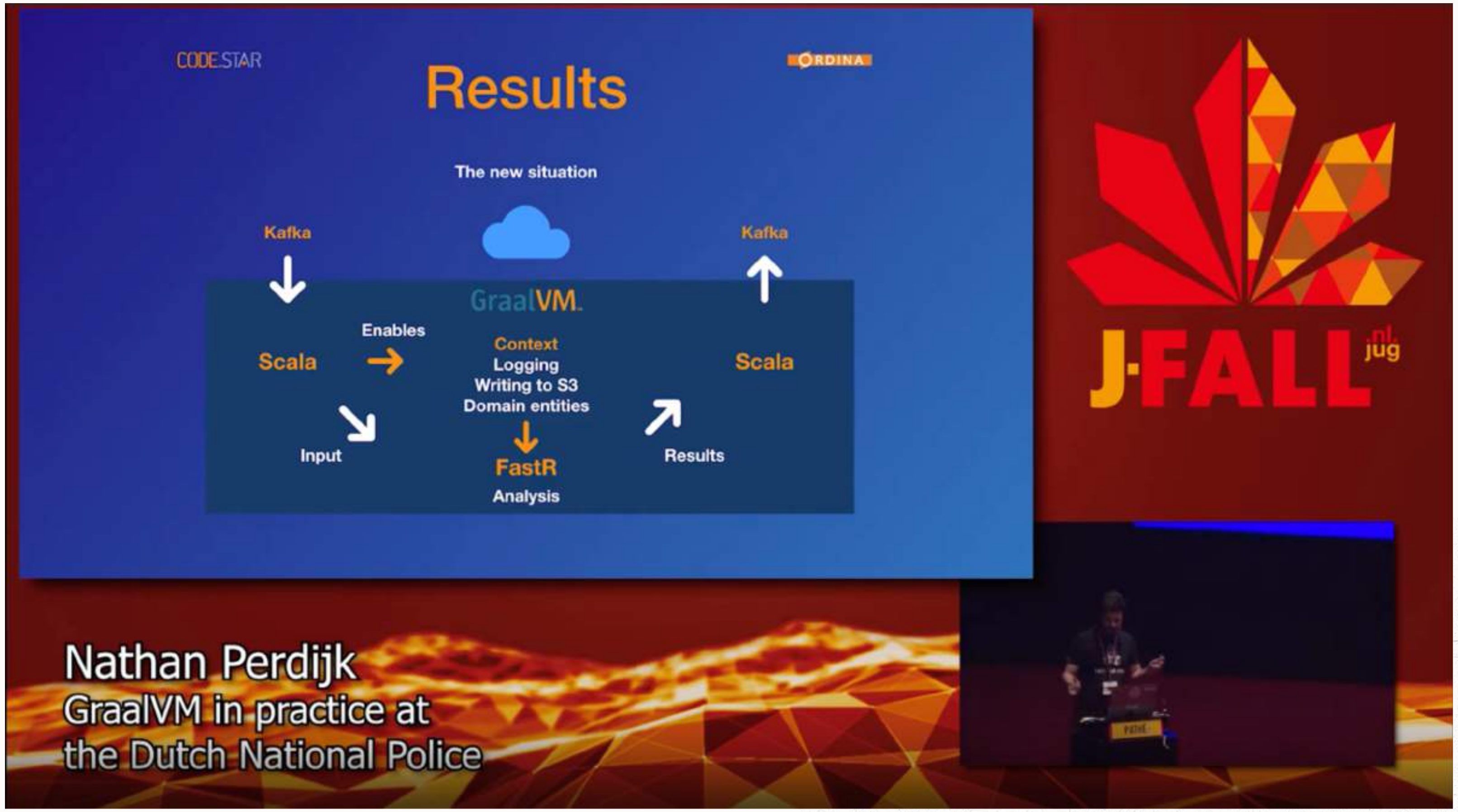
- Implemented as a “Truffle Language” (although “CUDA” is a platform, not a language)

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

Twitter uses GraalVM compiler in production to run their Scala microservices



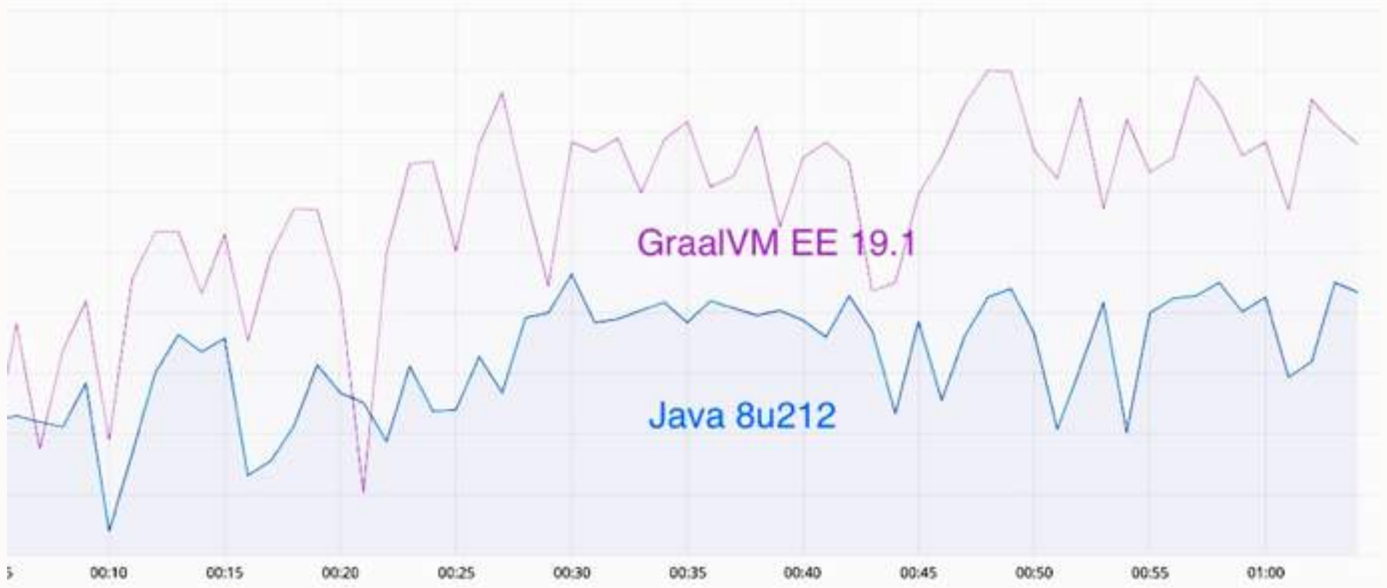
GraalVM in practice at the Dutch National Police



https://www.youtube.com/watch?v=poNIwZjoYjs

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

ORACLE
Cloud Infrastructure



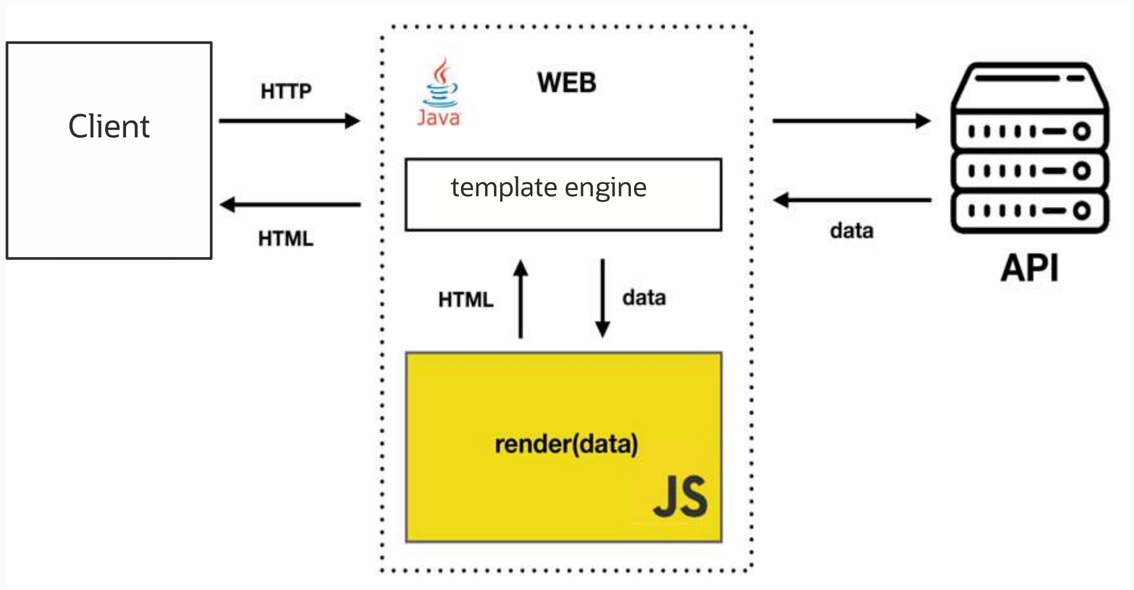
The rich ecosystem of CUDA-X libraries is now available for GraalVM applications.

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/

NVIDIA®

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

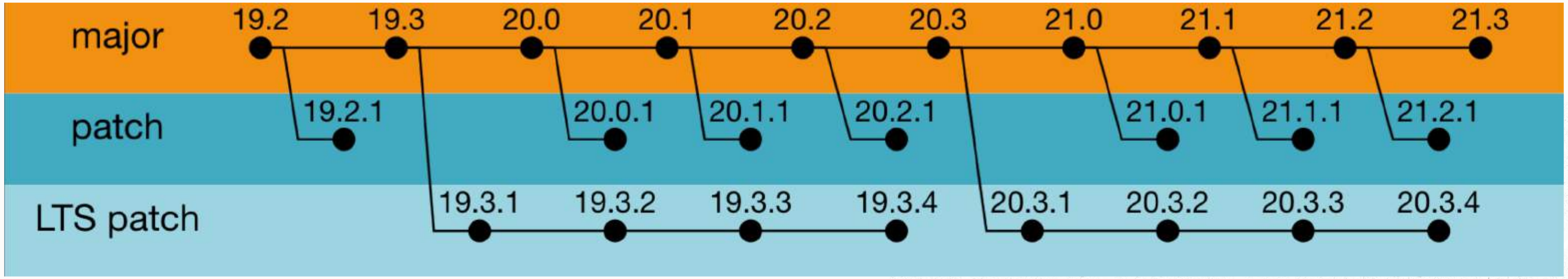


https://prog.world/new-odnoklassniki-frontend-launching-react-in-java-part-i/

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 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 to consider 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