ORACLE

GraalVM

Thomas Wuerthinger
GraalVM Project Lead
Oracle Labs
@thomaswue

December 2019

Safe harbor statement

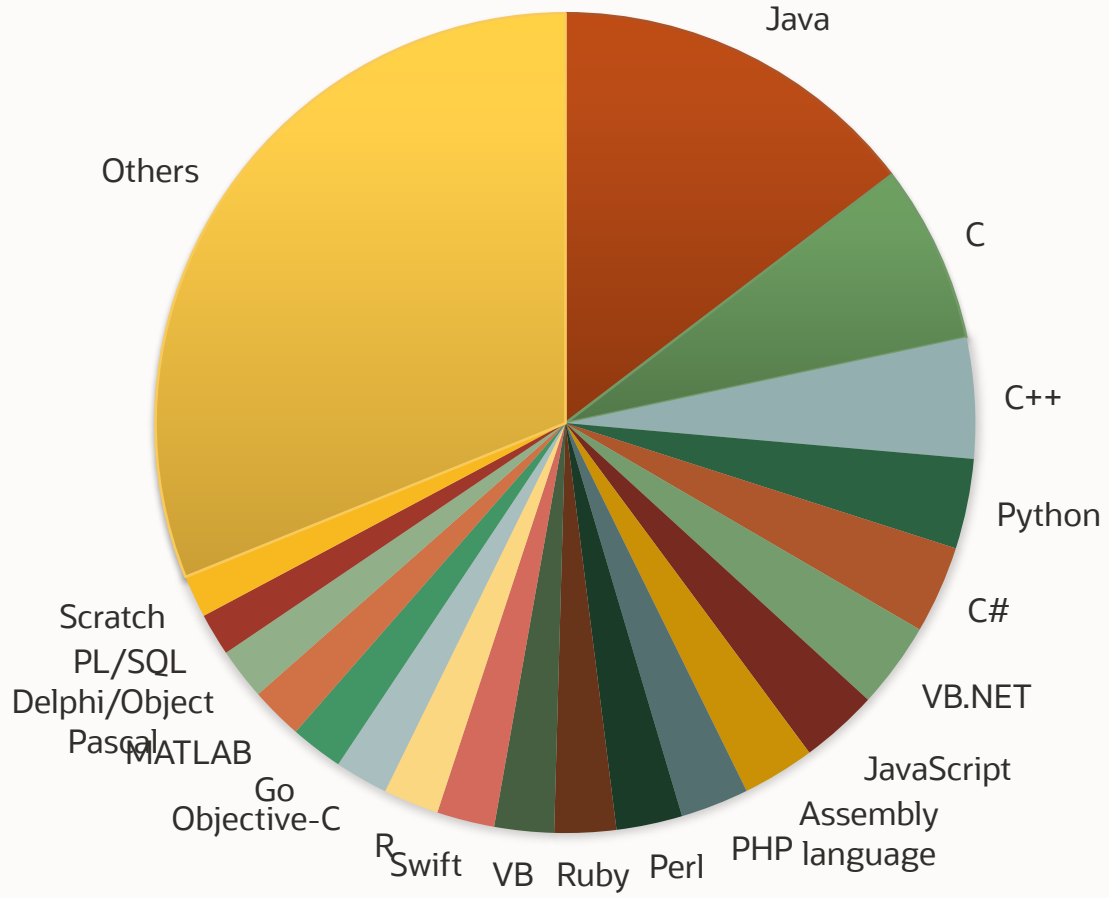
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development,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

© 2019 Oracle

The World is Polyglot!

Programming Language Popularity
(TOP 20 Languages From May 2018 Tiobe INDEX)



© 2019 Oracle

~8 years ago, we started a little research project…

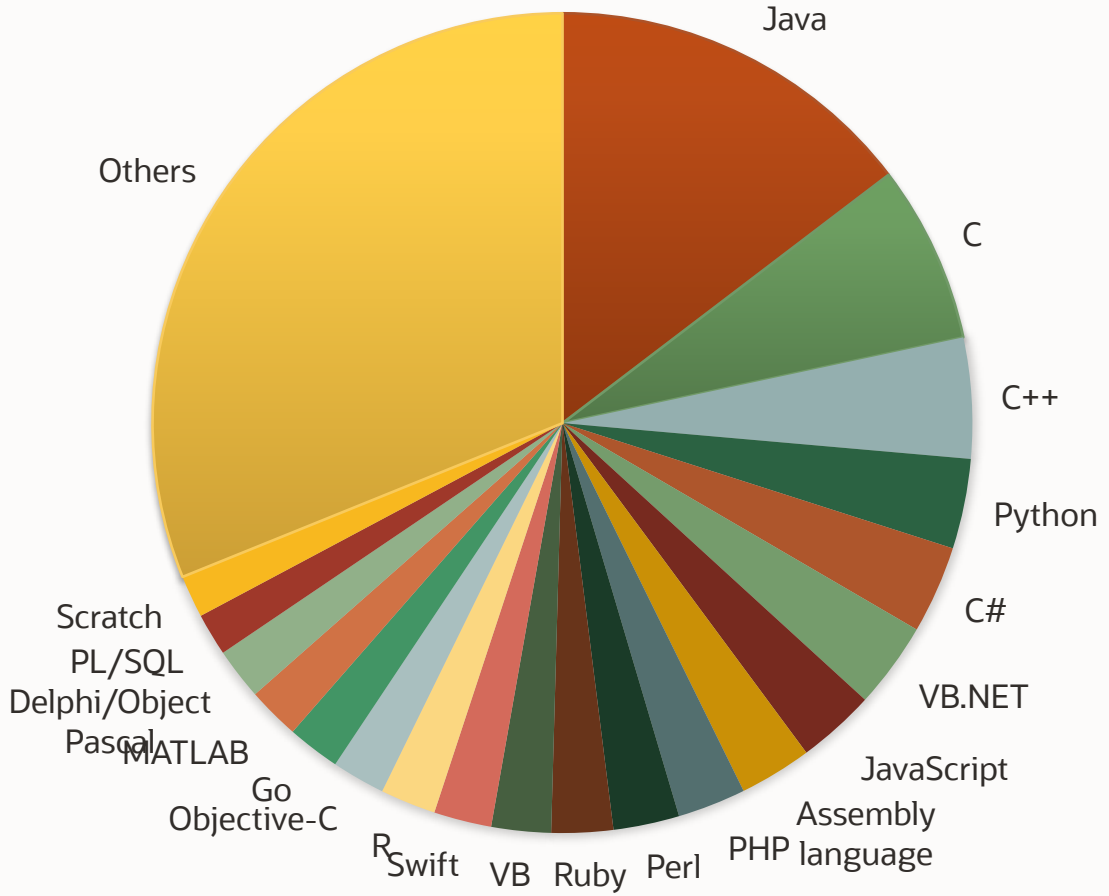
One VM to Rule Them All

Thomas Würthinger* Christian Wimmer* Andreas Wöß† Lukas Stadler†
Gilles Duboscq† Christian Humer† Gregor Richards§ Doug Simon* Mario Wolczko*

*Oracle Labs †Institute for System Software, Johannes Kepler University Linz, Austria §S³ Lab, Purdue University
{thomas.wuerthinger, christian.wimmer, doug.simon, mario.wolczko}@oracle.com

© 2019 Oracle

Programming Language Popularity
(TOP 20 Languages From May 2018 Tiobe INDEX)



© 2019 Oracle

GraalVM will be the Universal VM

ubiquitous across the cloud stack

1. run programs more efficient
2. make developers more productive

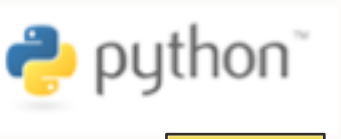
© 2019 Oracle

What is Graal VM?

- Drop-in replacement for Oracle Java 8 (and soon Java 11)
  - Run your Java application faster
- Ahead-of-time compilation for Java
  - Create standalone binaries with low footprint
- High-performance JavaScript, Python, Ruby, R, ...
  - The first VM for true polyglot programming
  - Implement your own language or DSL

© 2019 Oracle

GraalVM™



OpenJDK

ORACLE
Database

Native Image

© 2019 Oracle

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.

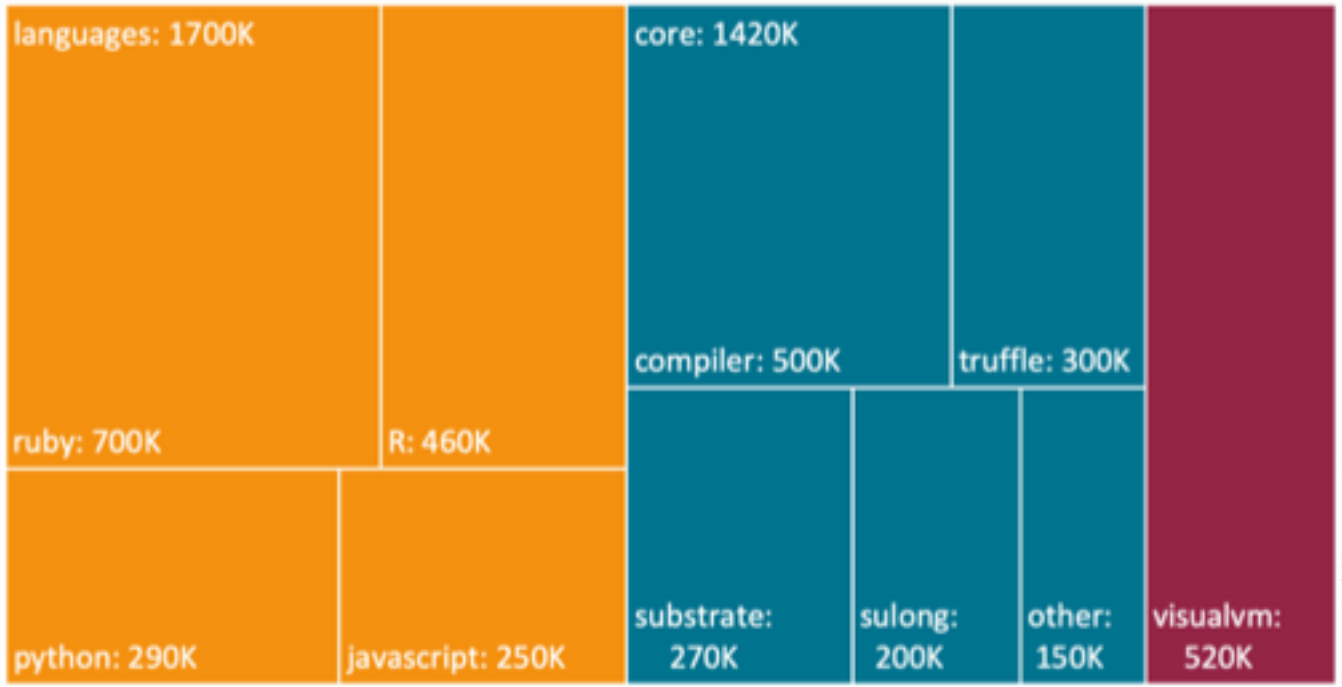
DOWNLOAD FROM OTN

FREE on Oracle Cloud!

© 2019 Oracle

GraalVM Open Source

Open Source LOC actively maintained by GraalVM team



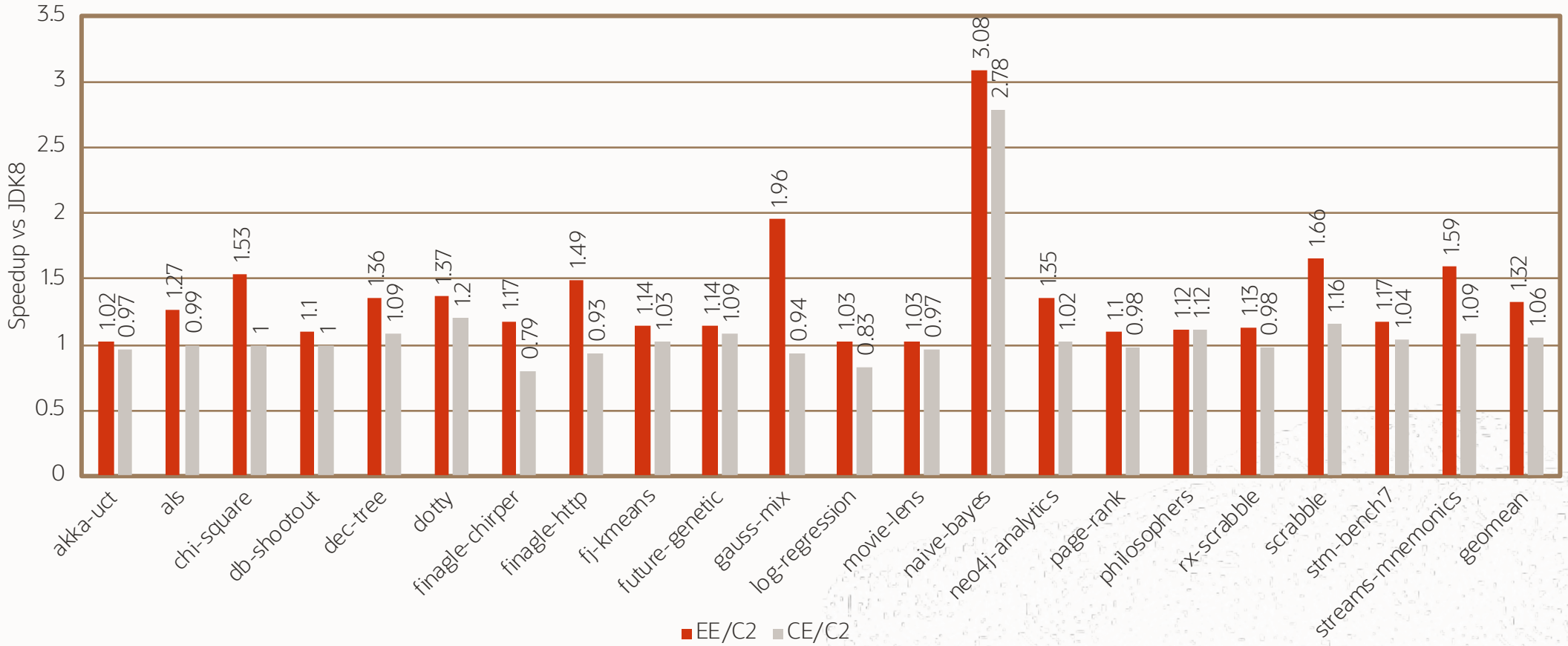
Total: 3,640,000 lines of code

© 2019 Oracle

GraalVM Vision:
Abstractions should be without performance regret!

© 2019 Oracle

GraalVM JIT Performance: Renaissance.dev



© 2019 Oracle

More Benchmarks…

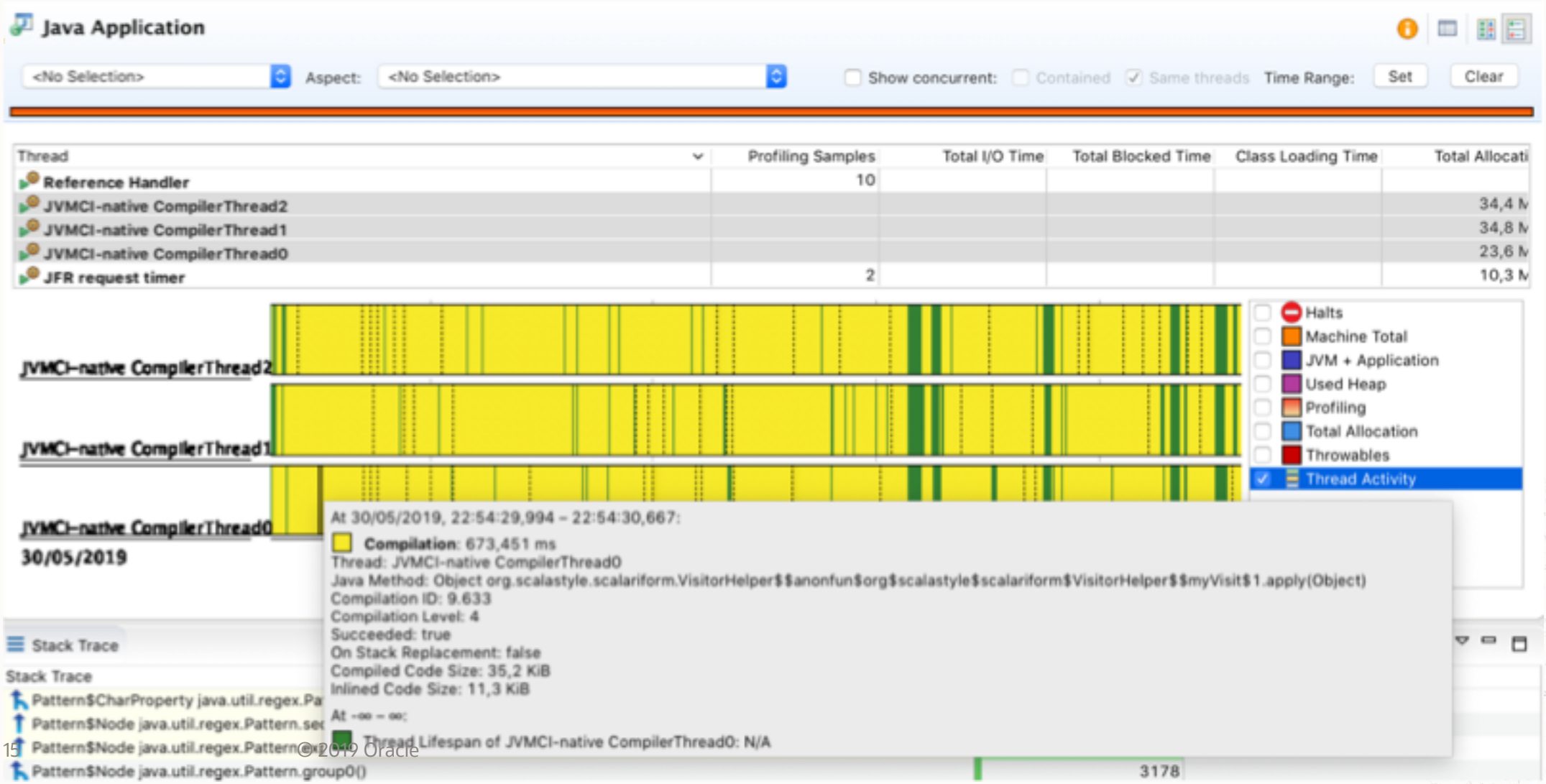
Optimizing for too few benchmarks is like overfitting a machine learning algorithm.

Therefore we started together with academic collaborators https://renaissance.dev

All benchmark data can be interesting; careful with conclusions though.

© 2019 Oracle

Java Flight Recorder Compilation Information



©2019 Oracle

ORACLE

GraalVM Native Image

© 2019 Oracle

GraalVM™

JIT

java MyMainClass

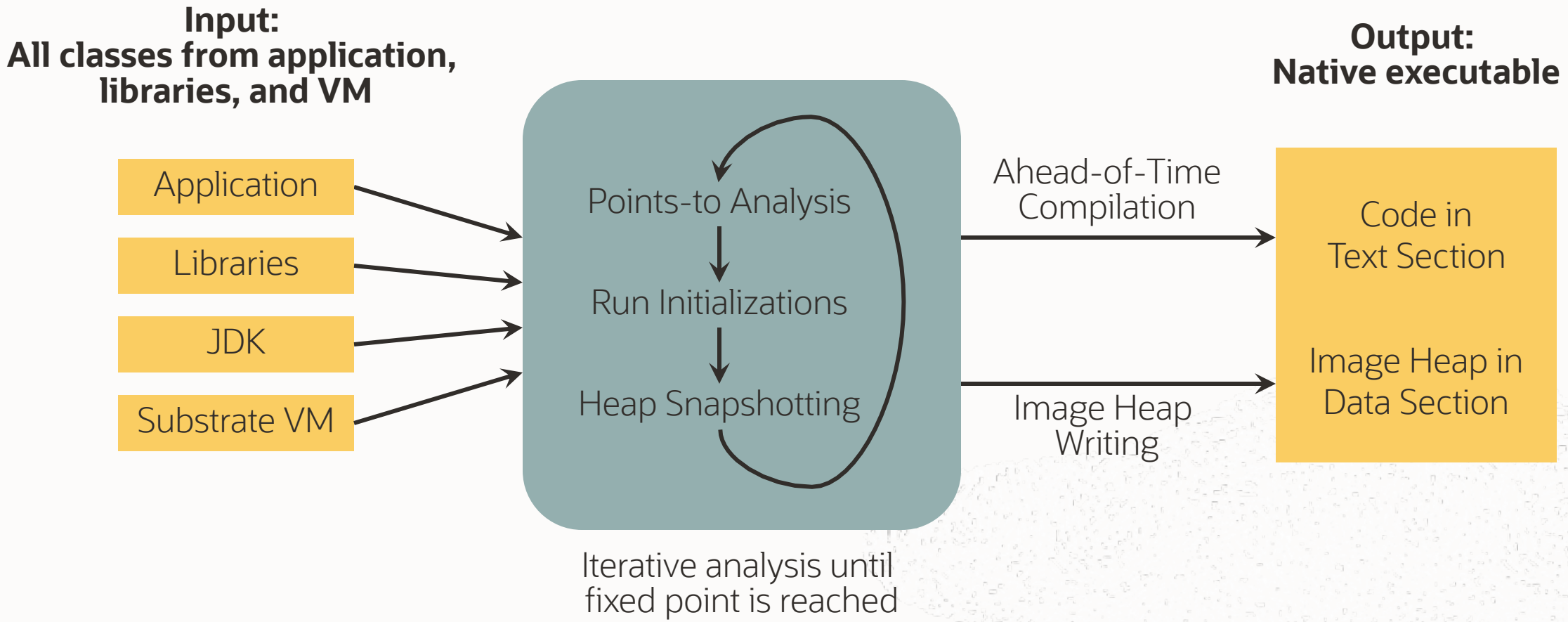
OpenJDK™

AOT

native-image MyMainClass
./mymainclass

© 2019 Oracle

Native Image Architecture



© 2019 Oracle

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.

©

Closed World Assumption

- The points-to analysis needs to see all bytecode
  - Otherwise aggressive AOT optimizations are not possible
  - Otherwise unused classes, methods, and fields cannot be removed
  - Otherwise a class loader / bytecode interpreter is necessary at run time
- Dynamic parts of Java require configuration at build time
  - Reflection, JNI, Proxy, resources, …
  - That's what this talk is about
- No loading of new classes at run time

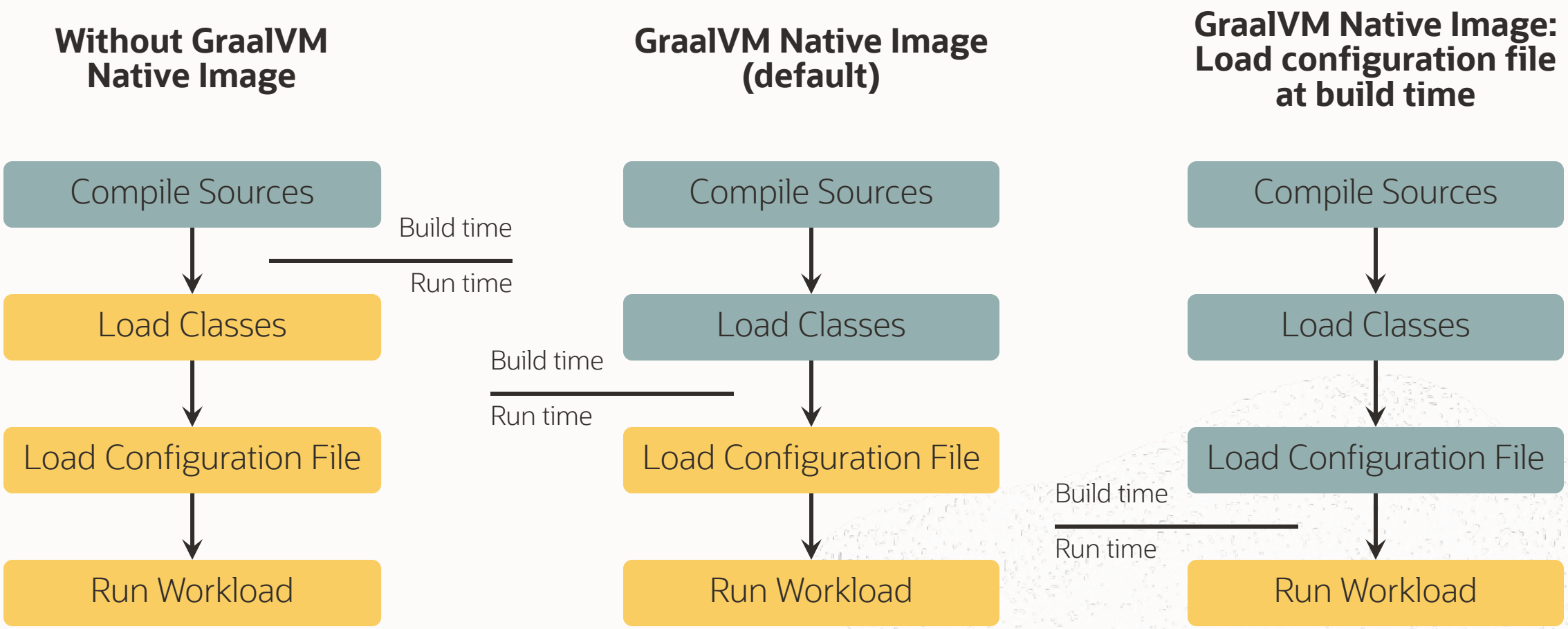
© 2019 Oracle

Image Heap

- Execution at run time starts with an initial heap: the “image heap”
  - Objects are allocated in the Java VM that runs the image generator
  - Heap snapshotting gathers all objects that are reachable at run time
- Do things once at build time instead at every application startup
  - Class initializers, initializers for static and static final fields
  - Explicit code that is part of a so-called “Feature”
- Examples for objects in the image heap
  - java.lang.Class objects, Enum constants

© 2019 Oracle

Benefits of the Image Heap



© 2019 Oracle

Get VMs Ready for the Cloud and Microservices

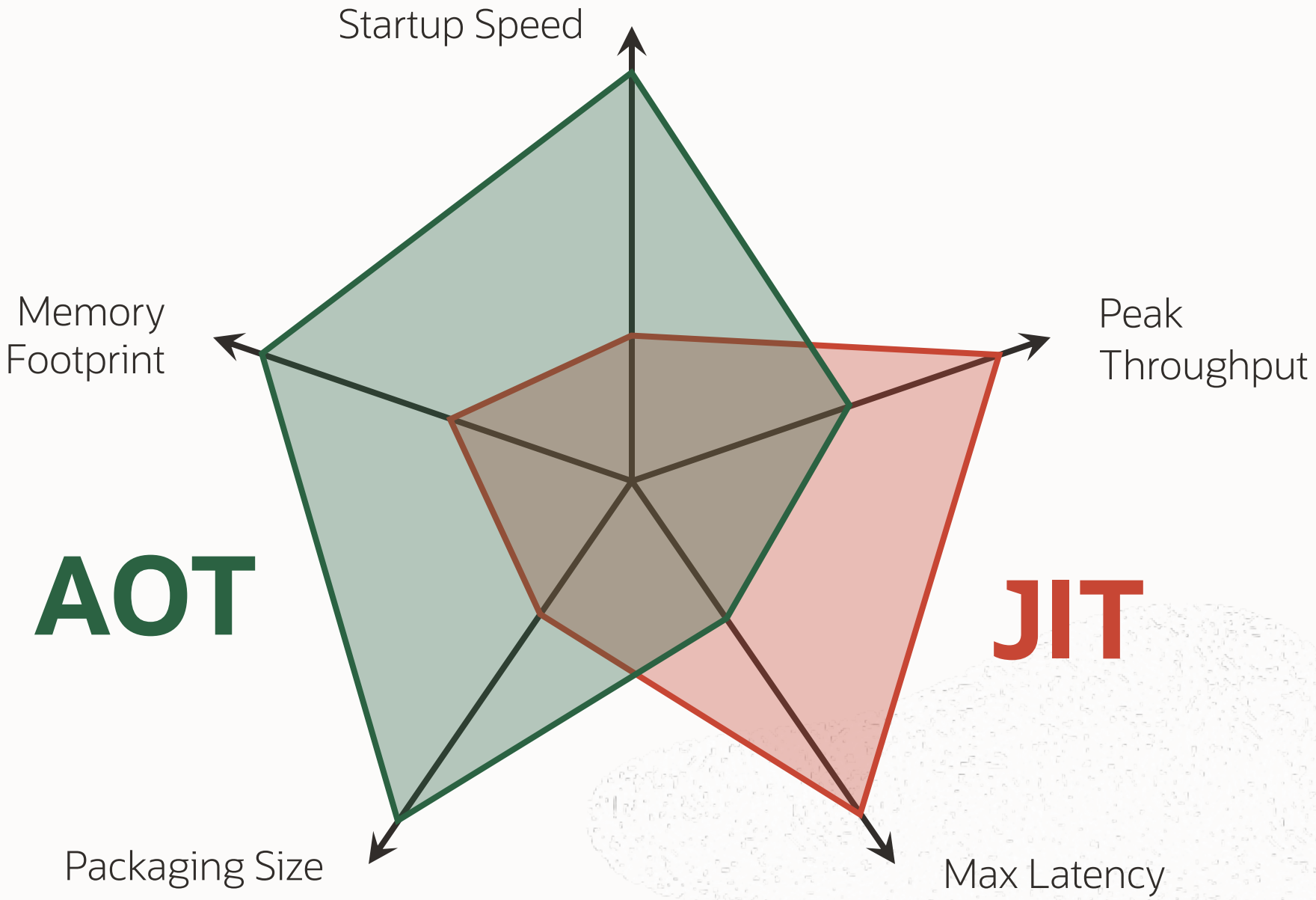
Important evaluation metrics:

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

Bruno Borges,
Microsoft Azure Advocate

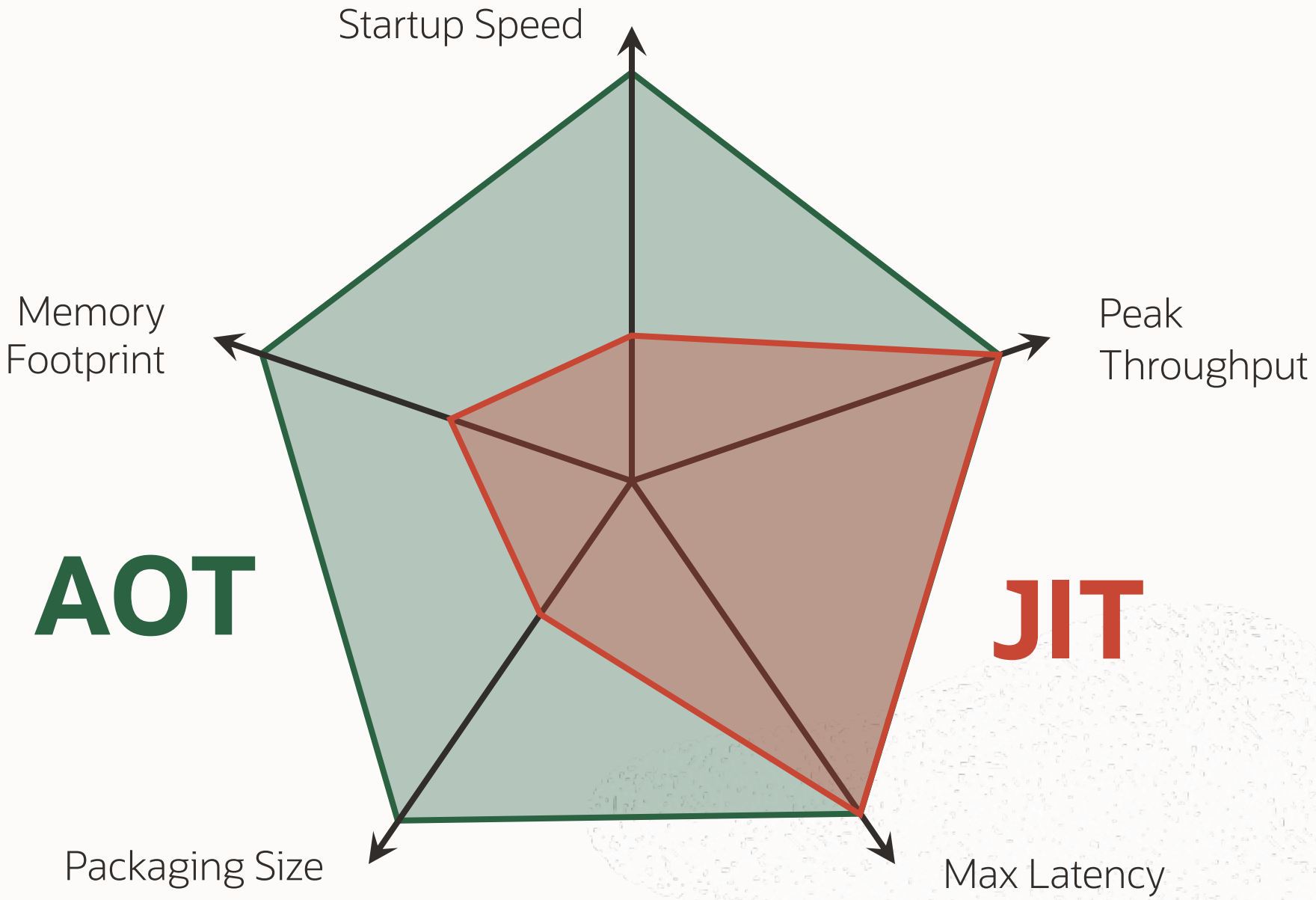
© 2019 Oracle

Currently



© 2019 Oracle

Goal



© 2019 Oracle

Java Microservice Frameworks with GraalVM Native Image Support

https://micronaut.io

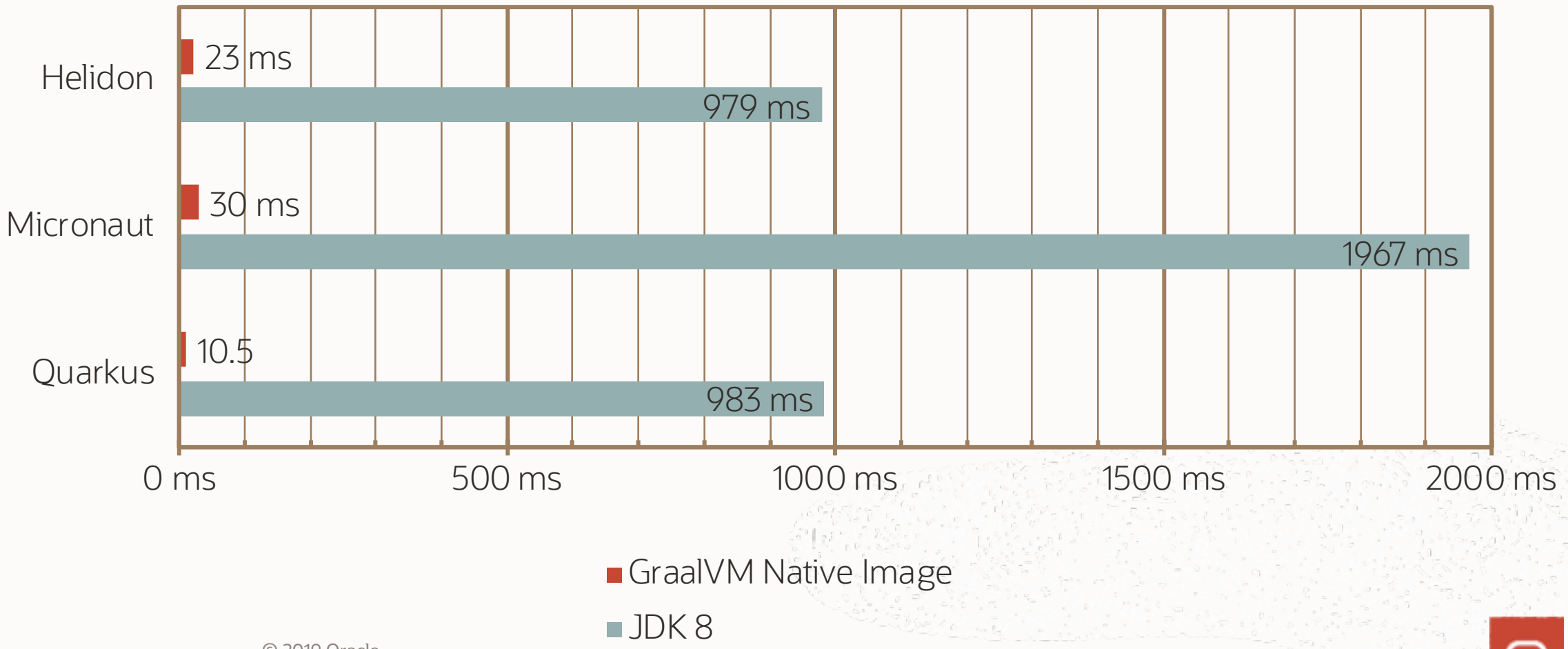
https://helidon.io

https://quarkus.io

Soon also Spring support
https://github.com/spring-projects-experimental/spring-graal-native

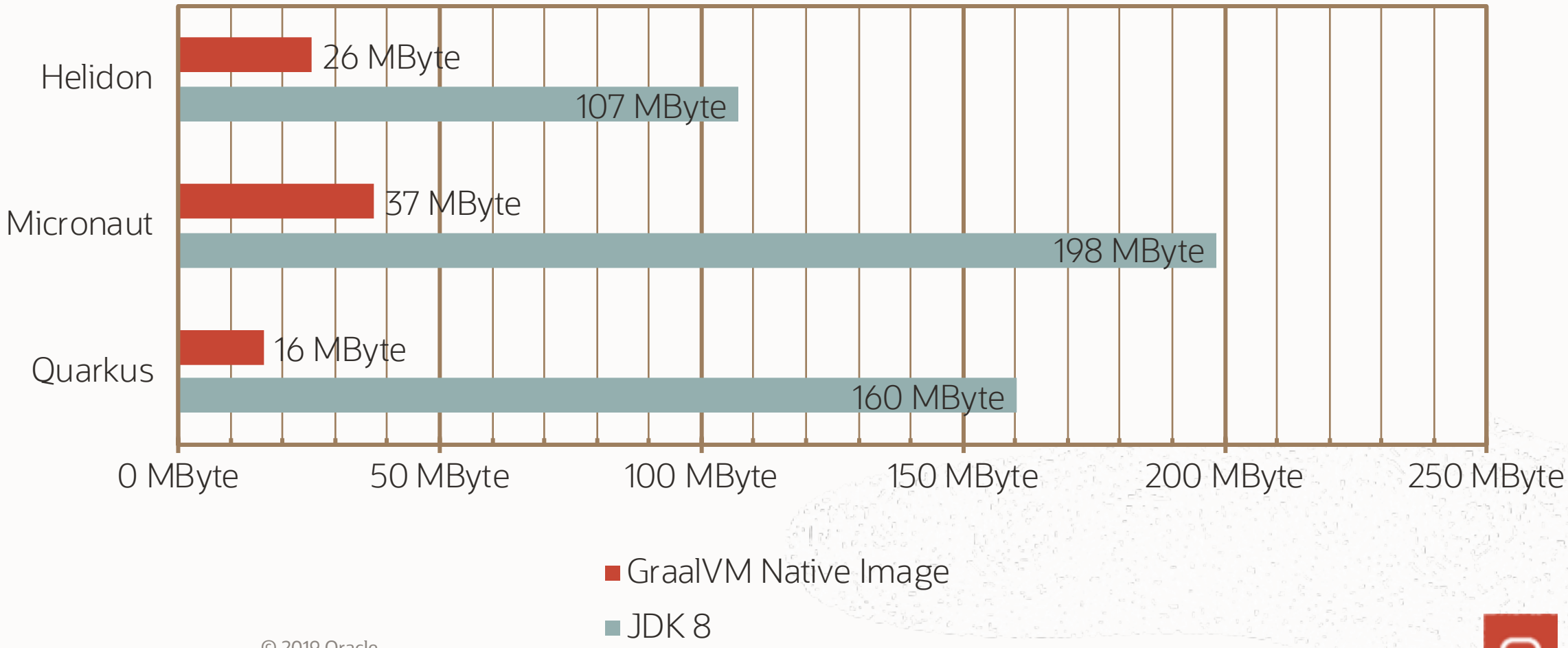
© 2019 Oracle

Startup Time



© 2019 Oracle

Memory Footprint



© 2019 Oracle

Native Image vs. Java HotSpot VM

- 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 Java HotSpot VM when
  - Heaps size is large
    - Multiple GByte – TByte heap size
  - Classes are only known at run time

© 2019 Oracle

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

© 2019 Oracle

Reflection and JNI

- Need configuration at image build time
  - Classes, methods, and fields that are reflectively visible
  - Necessary to keep the metadata small and to avoid too conservative points-to analysis

Example: Gson library to serialize / deserialize Java objects

Data class:

```
class Element {
  String value;
}
```

Serialize Java object to JSON:

```
Element element = new Element();
element.value = "Hello World";
String json = new Gson().toJson(element);
```

Deserialize JSON to Java object:

```
String json = "{\"value\":\"Hello World\"}";
Element element = new Gson().fromJson(json, Element.class);
```

Reflection configuration:

```
[
  {
    "name" : "com.oracle.test.Element",
    "fields" : [
      { "name" : "value" }
    ],
    "methods" : [
      { "name" : "<init>" }
    ]
  }
]
```


© 2019 Oracle

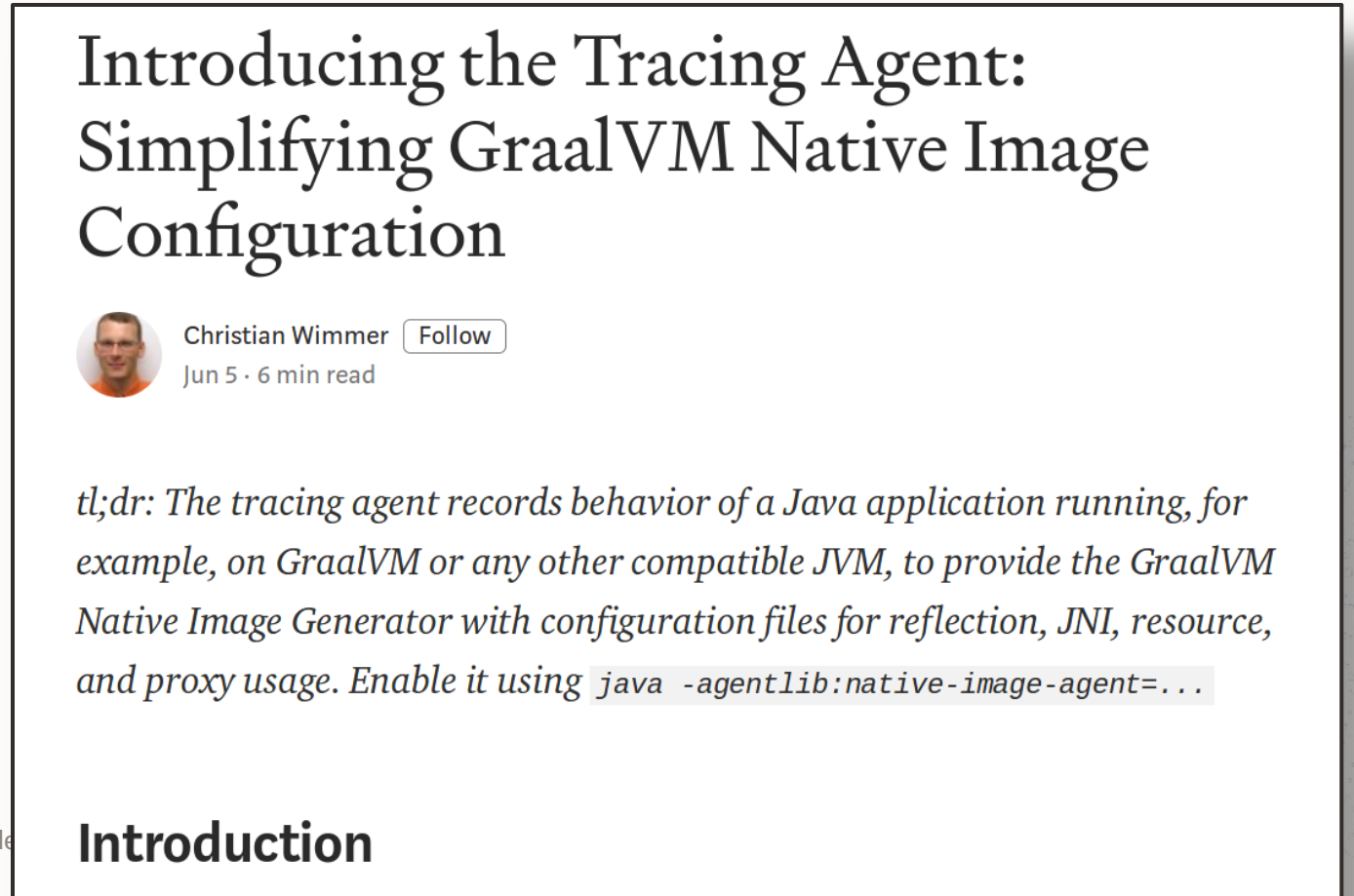
Tracing Agent

- 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

© 2019 Oracle

Blog Article with Details and Examples

https://medium.com/graalvm/c3b56c486271



Introducing the Tracing Agent: Simplifying GraalVM Native Image Configuration

Christian Wimmer Follow
Jun 5 · 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=...`

Introduction

© 2019 Oracle

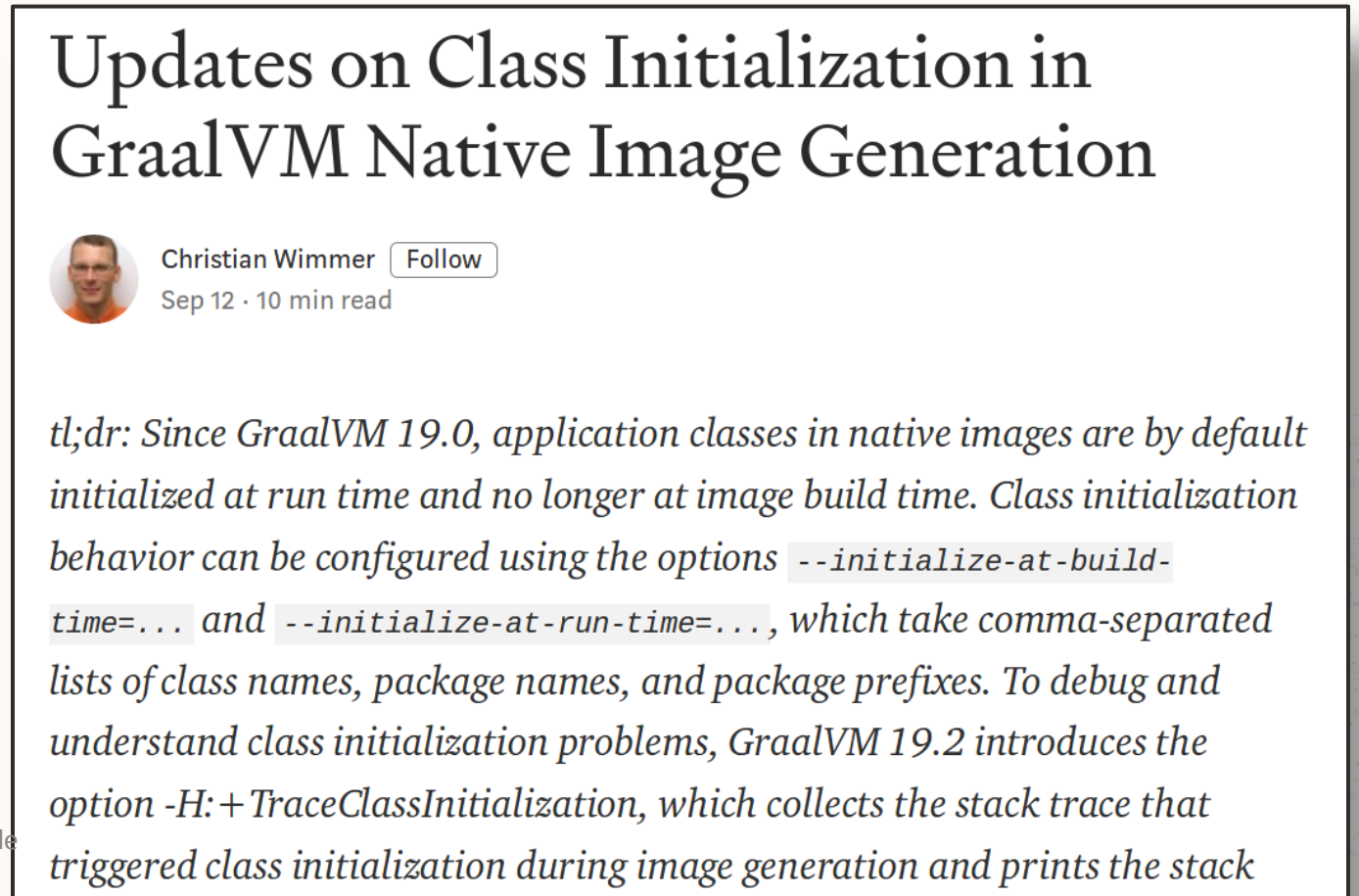
Class Initialization

- Class initialization at image build time improves application startup
- Configurable per class / package / package prefix
  - --initialize-at-build-time=... --initialize-at-run-time=...
  - By default, application classes are initialized at run time
  - Most JDK classes are initialized at image build time
  - Static analysis finds class initializers that can run at image build time
- Performance implications of class initialization at run time
  - If a class is initialized at run time, no instances can be in the image heap
  - Runtime checks before static method calls, static field accesses, and allocations

© 2019 Oracle

Blog Article with Details and Examples

https://medium.com/graalvm/c61faca461f7



Updates on Class Initialization in GraalVM Native Image Generation

Christian Wimmer Follow

Sep 12 · 10 min read

tl;dr: Since GraalVM 19.0, application classes in native images are by default initialized at run time and no longer at image build time. Class initialization behavior can be configured using the options `--initialize-at-build-time=...` *and* `--initialize-at-run-time=...`*, which take comma-separated lists of class names, package names, and package prefixes. To debug and understand class initialization problems, GraalVM 19.2 introduces the option -H:+TraceClassInitialization, which collects the stack trace that triggered class initialization during image generation and prints the stack*

© 2019 Oracle

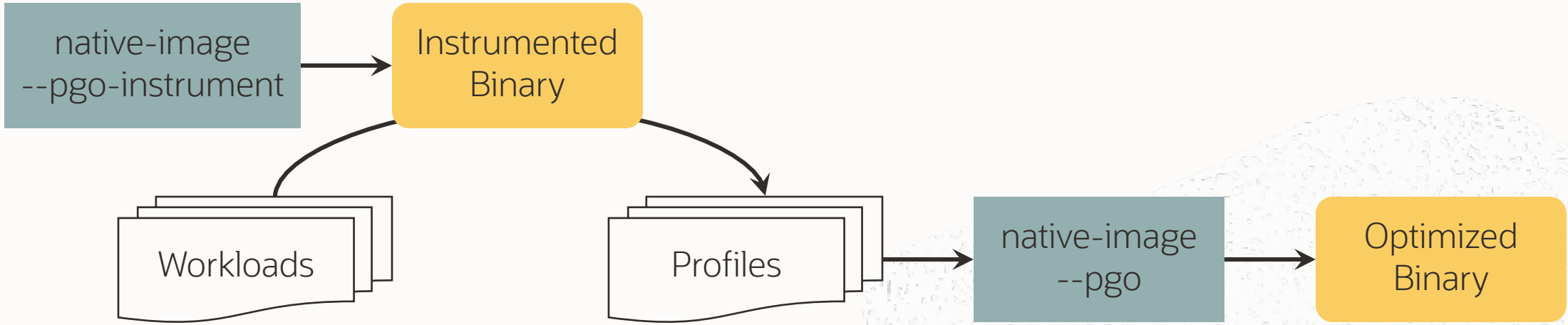
Native Image Support in Libraries

- 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/

© 2019 Oracle

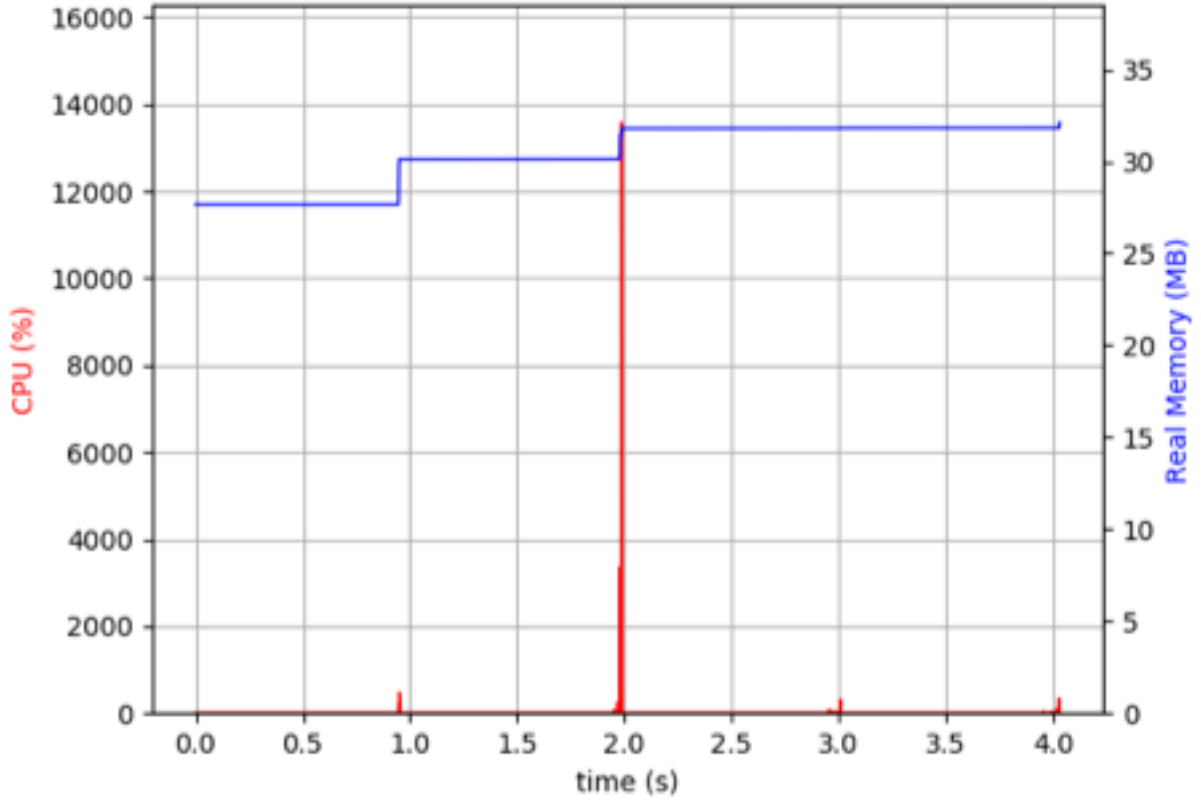
Profile-Guided Optimizations (PGO)

- AOT compiled code cannot optimize itself at run time
  - No dynamic “hot spot” compilation
- PGO requires relevant workloads at build time
- Optimized code runs immediately at startup, no “warmup” curve

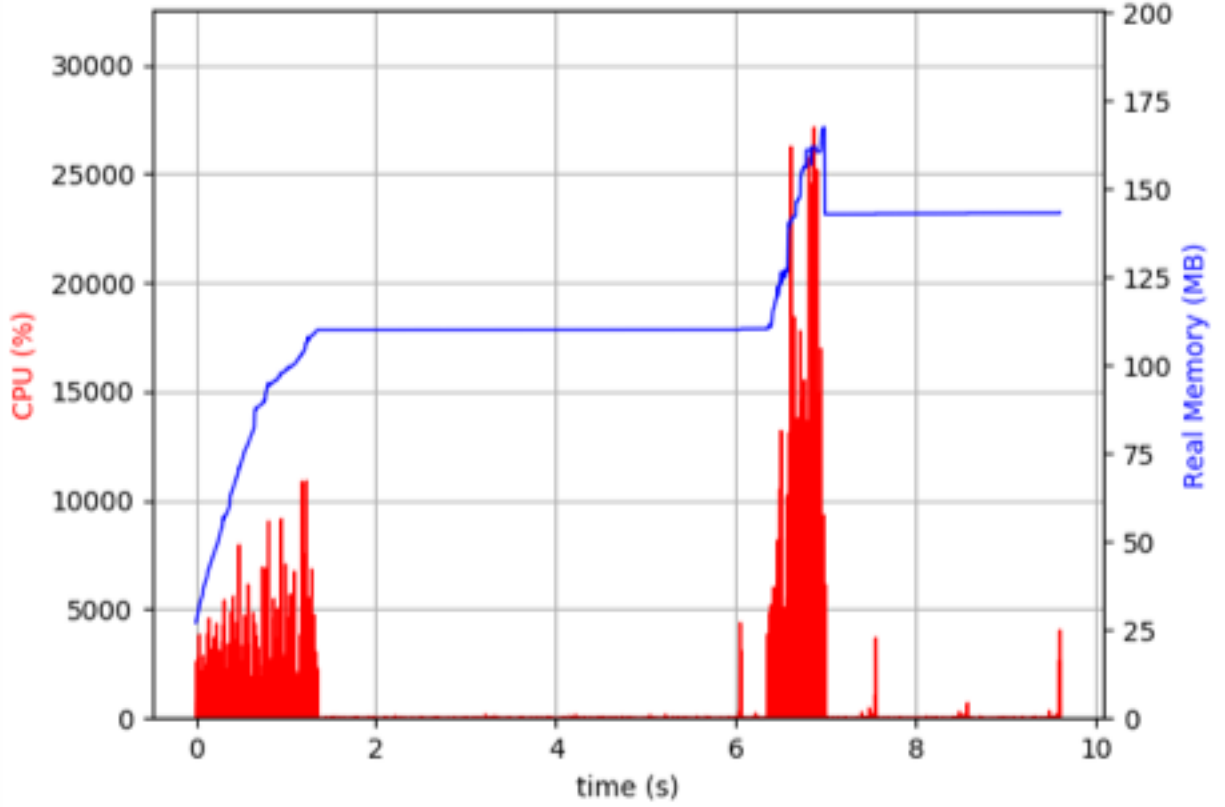


© 2019 Oracle

AOT



JIT



© 2019 Oracle

Tuning Native Image Footprint

-Xmx (maximum memory), default infinite

-Xmn (new generation size), default 256m

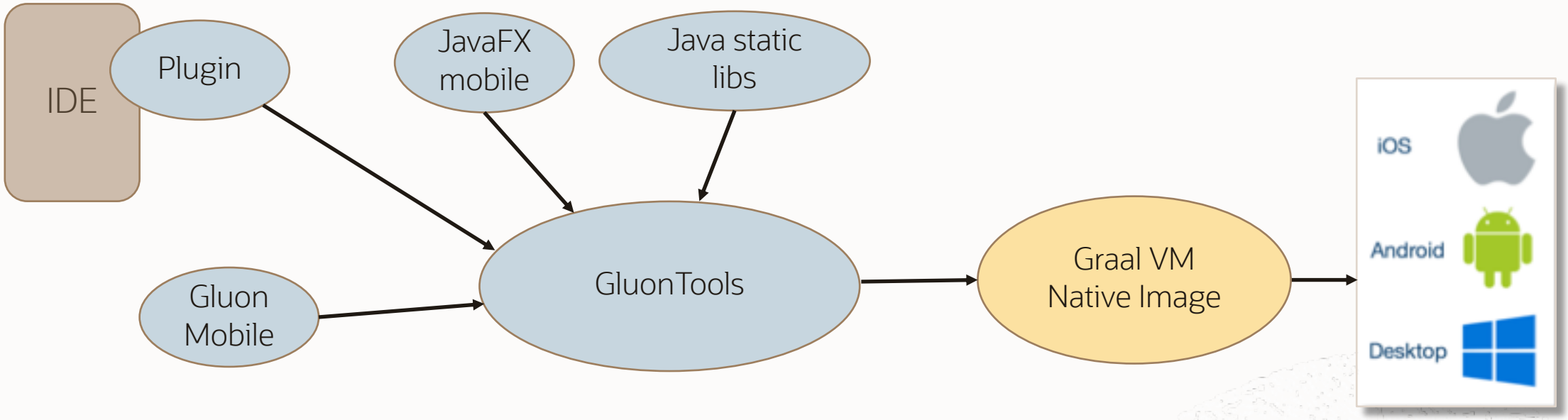
© 2019 Oracle

Native Image:
Support any Platform via LLVM

- Implemented an LLVM backend for GraalVM Native Image
  - In case Apple starts requiring LLVM for their apps
  - Easy to port Scala to niche architectures
- Using LLVM comes with a price
  - Moving GC in LLVM is still experimental
  - Slow at compilation
  - Slower run-time generated code

© 2019 Oracle

Cross-Platform GraalVM



- JavaFX extensions for mobile
- Integration with mobile functionality (e.g. GPS/camera)
- Mobile-specific connectivity

© 2019 Oracle

Key Performance Takeaways

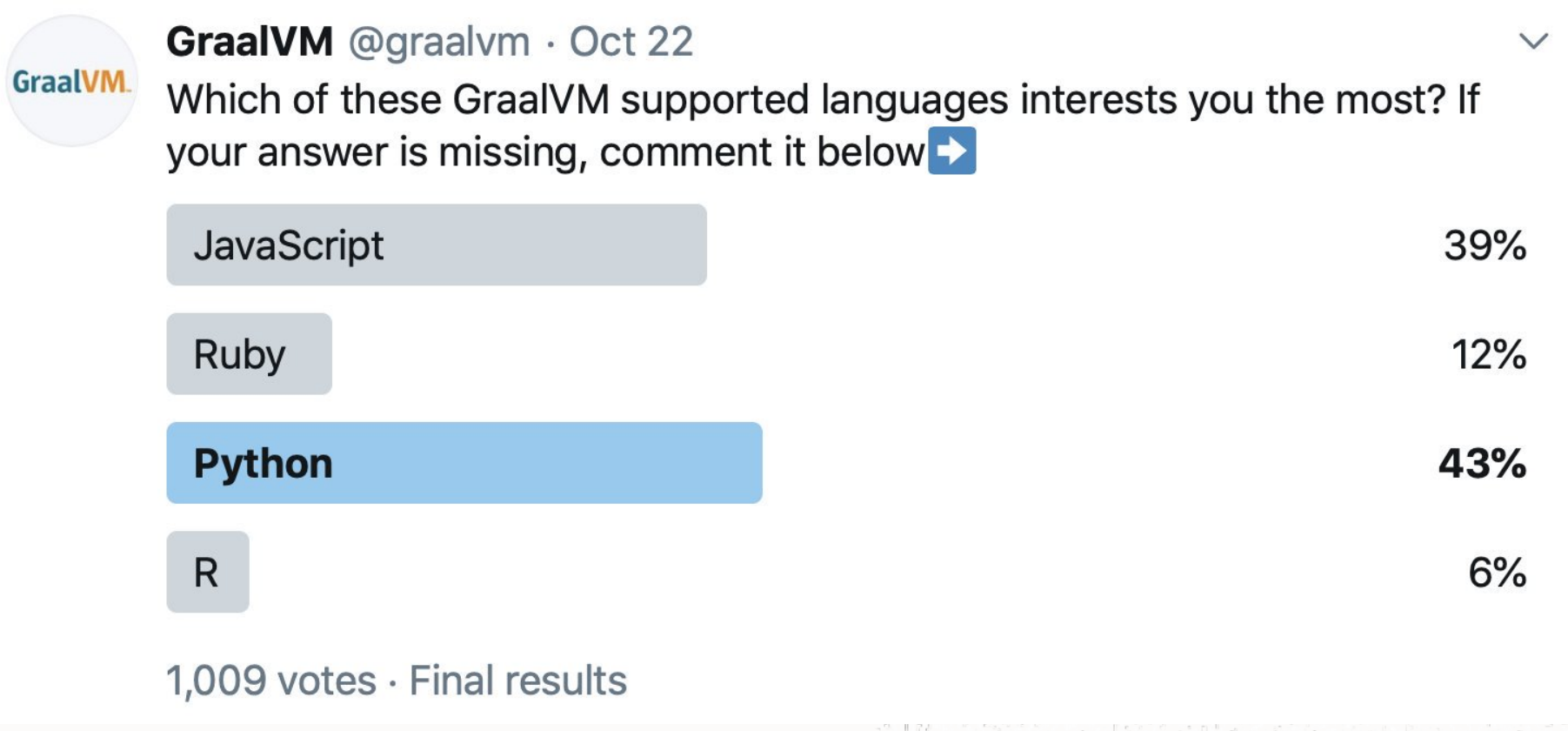
- Write small methods
- Local allocations are free, global data structures expensive
- Don’t hand optimize, unless you have studied the compiler graph
  - Make a pull request (or at least an issue) for the GraalVM project!

- For best throughput use GraalVM JIT,
  for best startup & footprint use GraalVM AOT (native images)

© 2019 Oracle

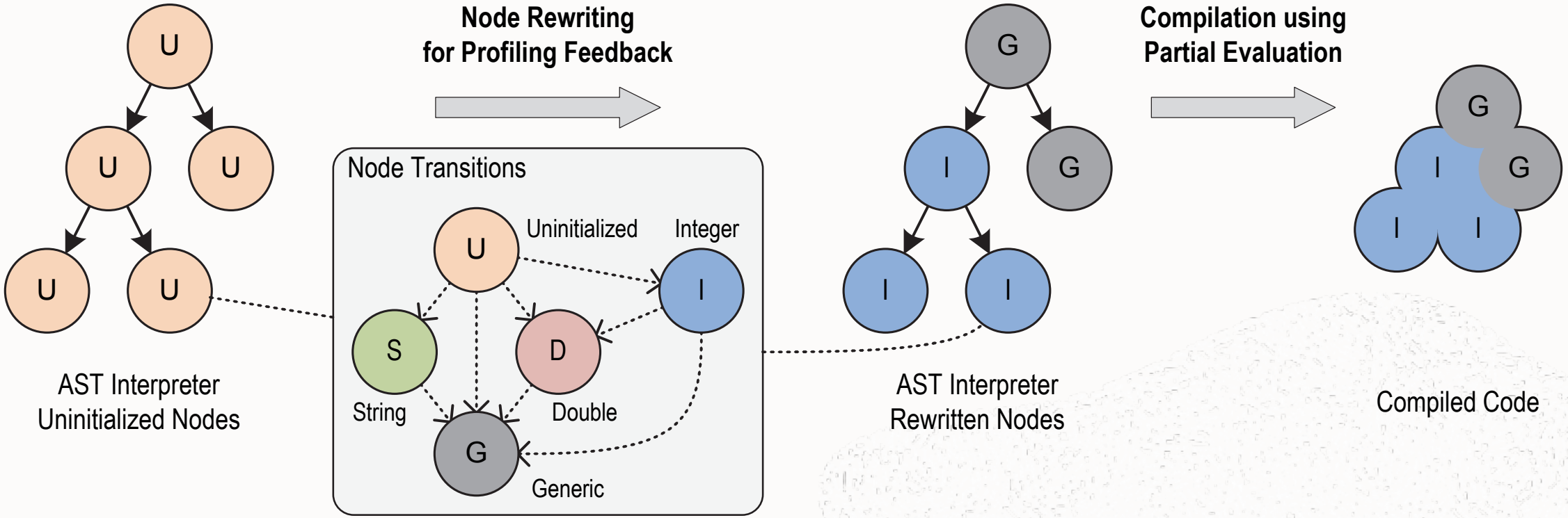
Polyglot and Embeddability

© 2019 Oracle


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

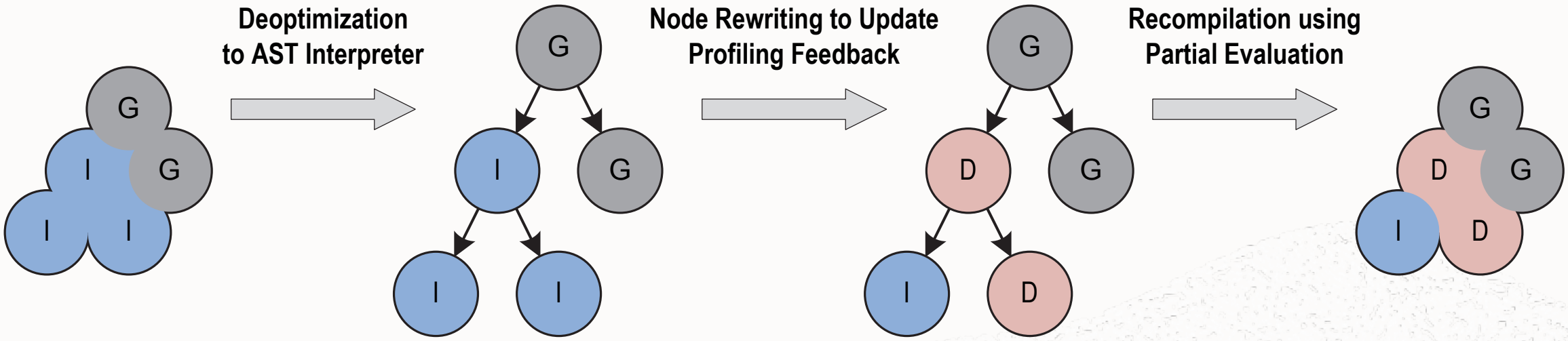
© 2019 Oracle

Optimization and Speculation…



© 2019 Oracle

And Deoptimization!



© 2019 Oracle

Simple Embeddability

```
public class Main {
    public static void main(String[] args) {
        try (Engine engine = Engine.create()) {
            Source source = Source.create("js", "21 + 21");
            try (Context context = Context.newBuilder()
                                          .engine(engine)
                                          .build()) {
                int v = context.eval(source).asInt();
                assert v == 42;
            }
            try (Context context = Context.newBuilder()
                                          .engine(engine)
                                          .build()) {
                int v = context.eval(source).asInt();
                assert v == 42;
            }
        }
    }
}
```


© 2019 Oracle

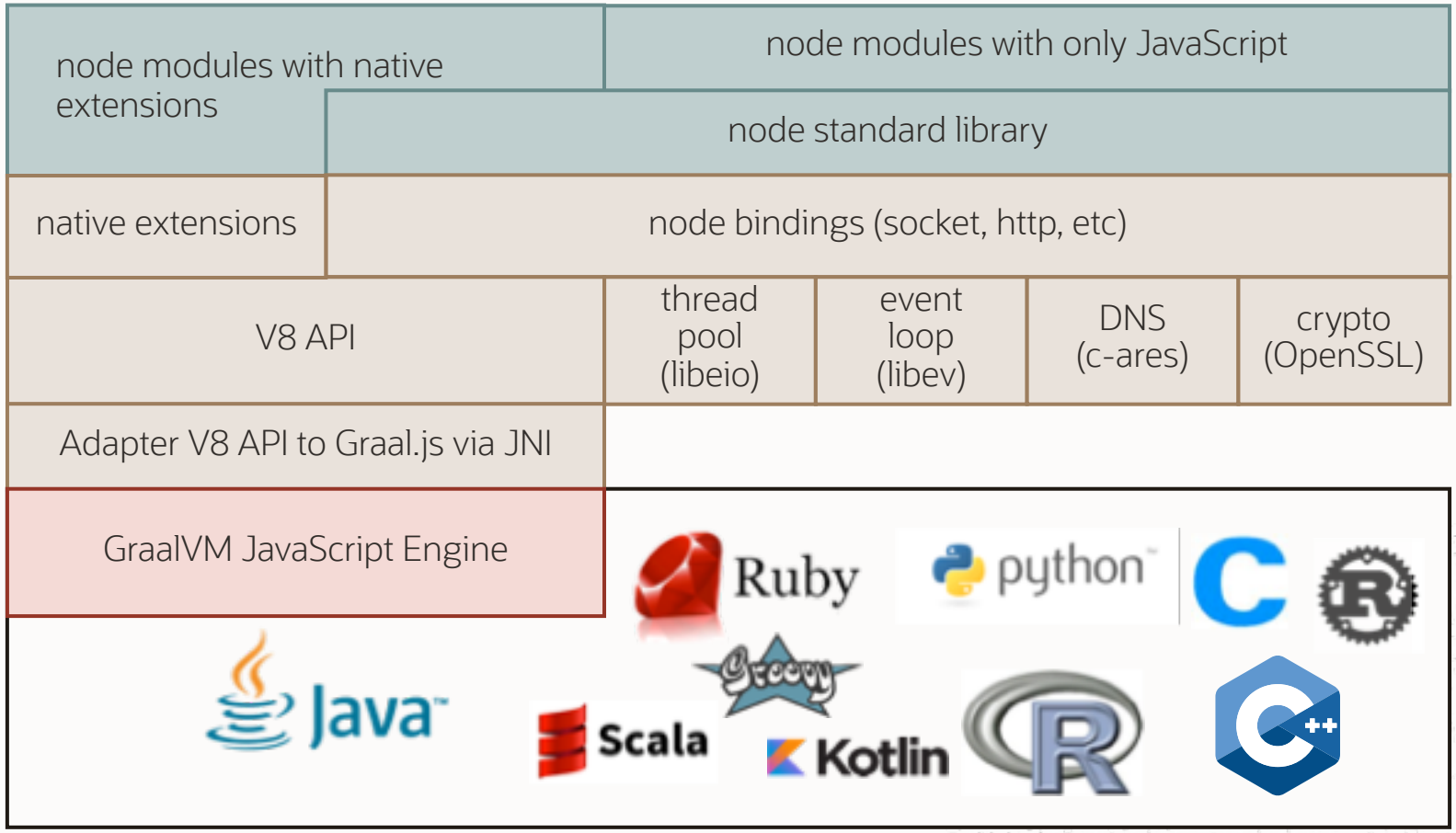
Full separation of logical and physical data layout, enabling virtual data structures

```
static class ComputedArray implements ProxyArray {
    public Object get(long index) {
        return index * 2;
    }
    public void set(long index, Value value) {
        throw new UnsupportedOperationException();
    }
    public long getSize() {
        return Long.MAX_VALUE;
    }
}

public static void main(String[] args) {
    try (Context context = Context.create()) {
        ComputedArray arr = new ComputedArray();
        context.getBindings("js").putMember("arr", arr);
        long result = context.eval("js",
                        "arr[1] + arr[1000000000]")
                    .asLong();
        assert result == 2000000002L;
    }
}
```

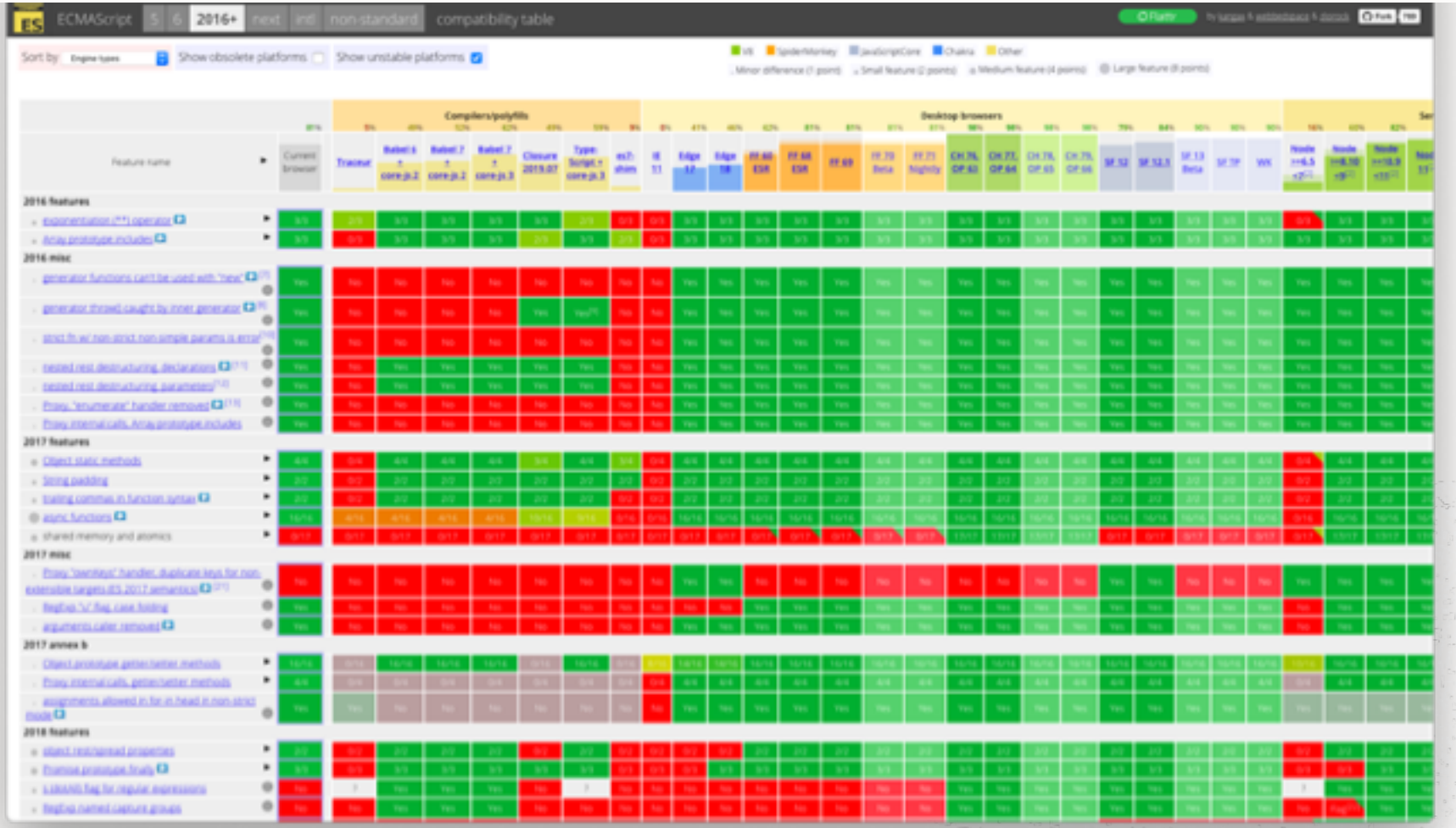

© 2019 Oracle

Architecture of Node.js running via GraalVM

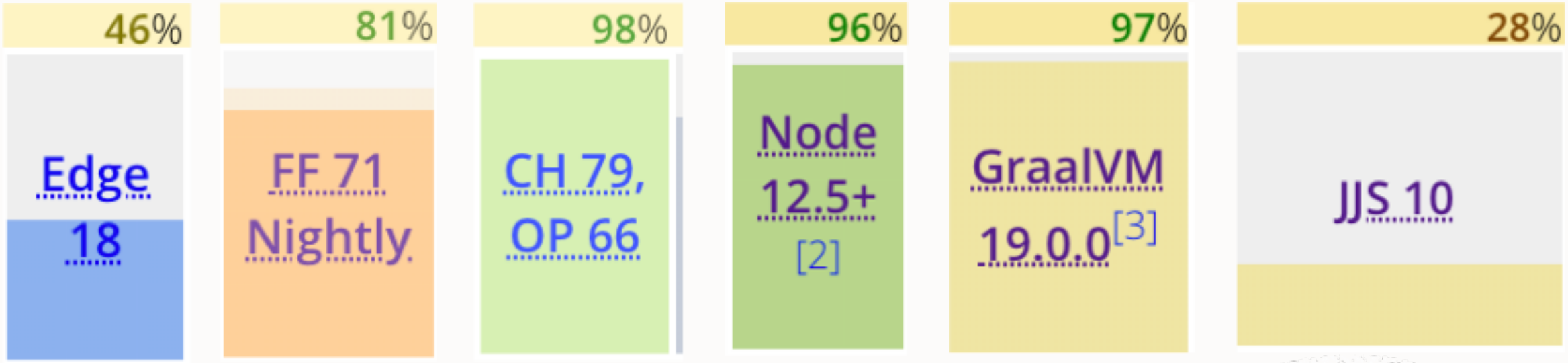


© 2019 Oracle

https://kangax.github.io



© 2019 Oracle


46%
Edge 18
81%
FF 71 Nightly
98%
CH 79, OP 66
96%
Node 12.5+ [2]
97%
GraalVM 19.0.0[3]
28%
JJS 10

© 2019 Oracle

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

© 2019 Oracle

Trade-Offs

Advantages

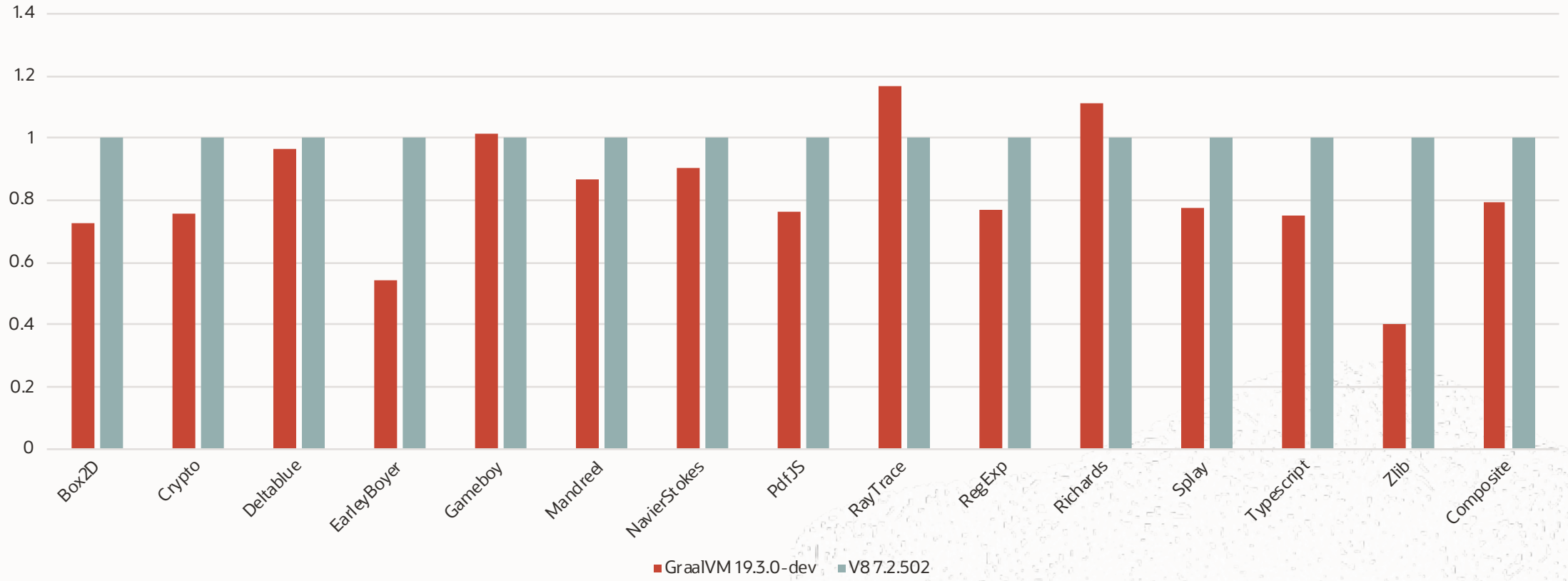
- Java interoperability: Use any Java library or framework
- Polyglot capabilities (Python, Ruby, R)
- Run with large heaps and JVM garbage collectors

Disadvantages

- Longer start-up time
- Higher memory footprint

© 2019 Oracle

Graal.js Performance (versus V8)



© 2019 Oracle

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

© 2019 Oracle

fastRCluster package

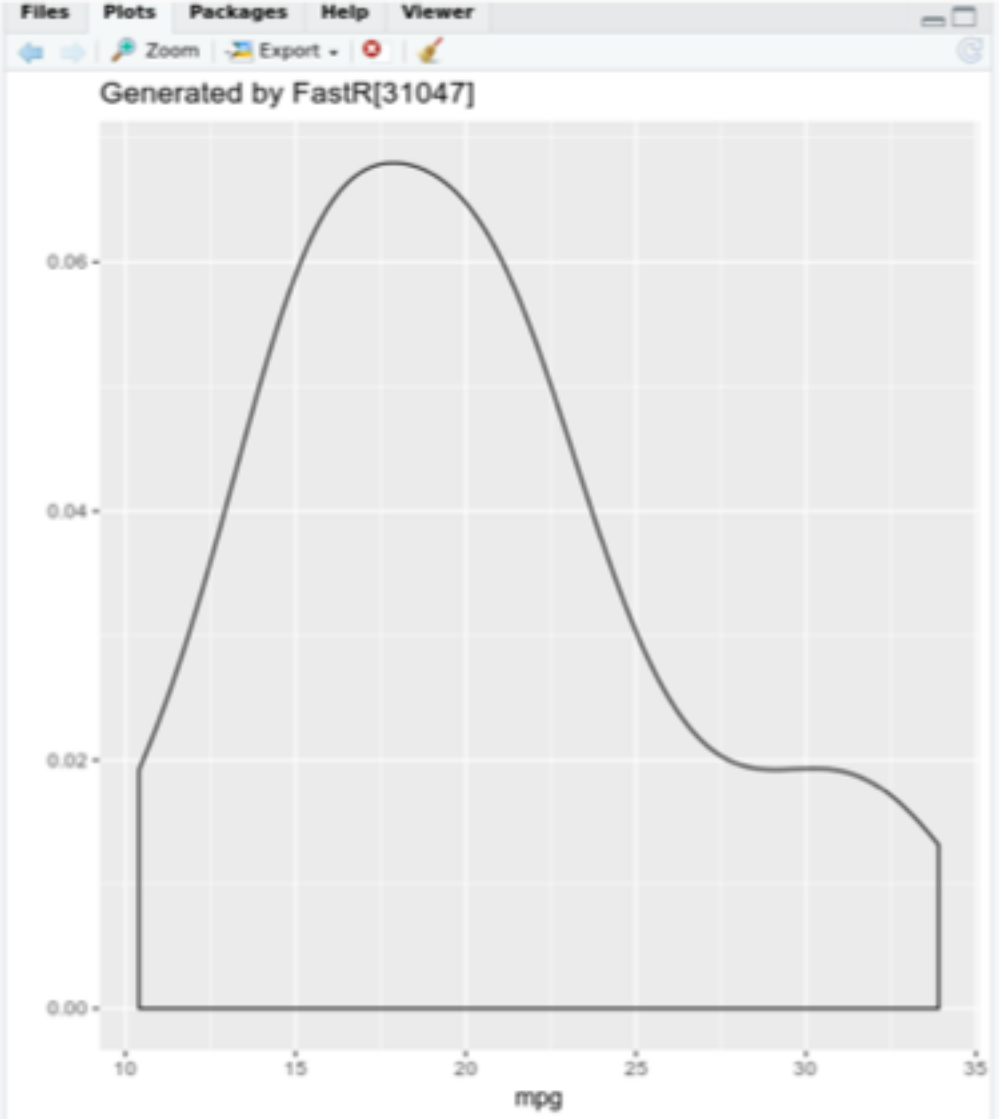
Console | Terminal | Jobs

~/dev/Graal/shinybrot/

```
[[1]]

               _
platform       x86_64-unknown-linux-gnu
arch           amd64
os             linux5.0.0-20-generic
system         amd64, linux5.0.0-20-generic
major          3
minor          5.1
year           2018
month          07
day            02
svn rev
language       R
engine         FastR
version.string FastR version 3.5.1 (2018-07-02)

> fastr(fastrNode, R.version$engine)
[1] "FastR"
> gg <- fastr(fastrNode, ggplot2::qplot(mpg, data=mtcars,
 geom="density", main=paste0("Generated by ", R.version$en
gine, "[", Sys.getpid(), "]")))
> plot(gg)
>
```



Python Polyglot

- Use code from Java or any other GraalVM language
- Embed into JVM languages using GraalVM Embedder API

```
# load the R package lattice and open a window for plotting
polyglot.eval(string="library(lattice); awt()", language="R")

# Create R function that draws a histogram
plot = polyglot.eval(string="""
function(y) {
  print(histogram(~as.vector(y),
                  main='Hello from Python to R',
                  xlab='Python List'))
}
""", language="R")

# invoke the R function with data from Python
plot(numpy.array(mydata))
```

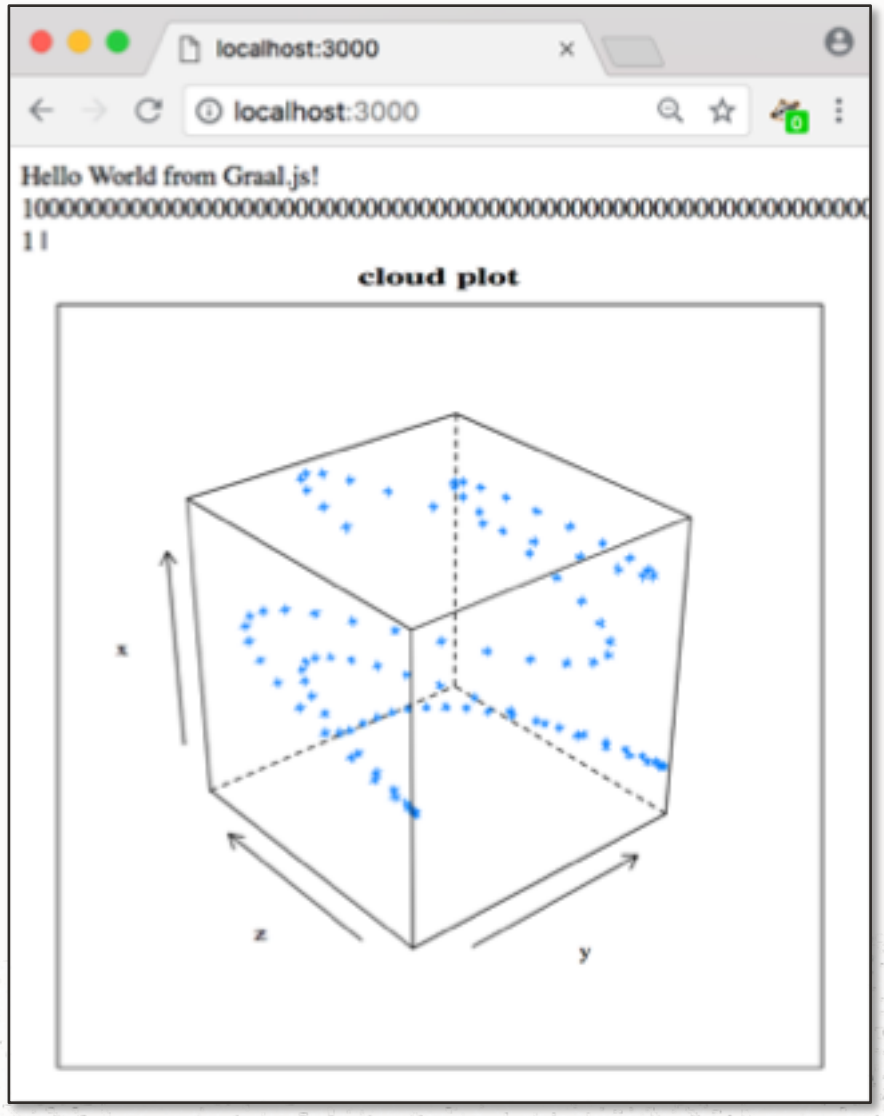

© 2019 Oracle

Python Package Support

- Pip installer is not available, yet (c.f.: no sockets)
  - We ship our own installer ginstall
- Pandas and NumPy can be installed work for a wide range of code in the latest builds
- More and more pure Python packages “just work” and compatibility is improving

© 2019 Oracle

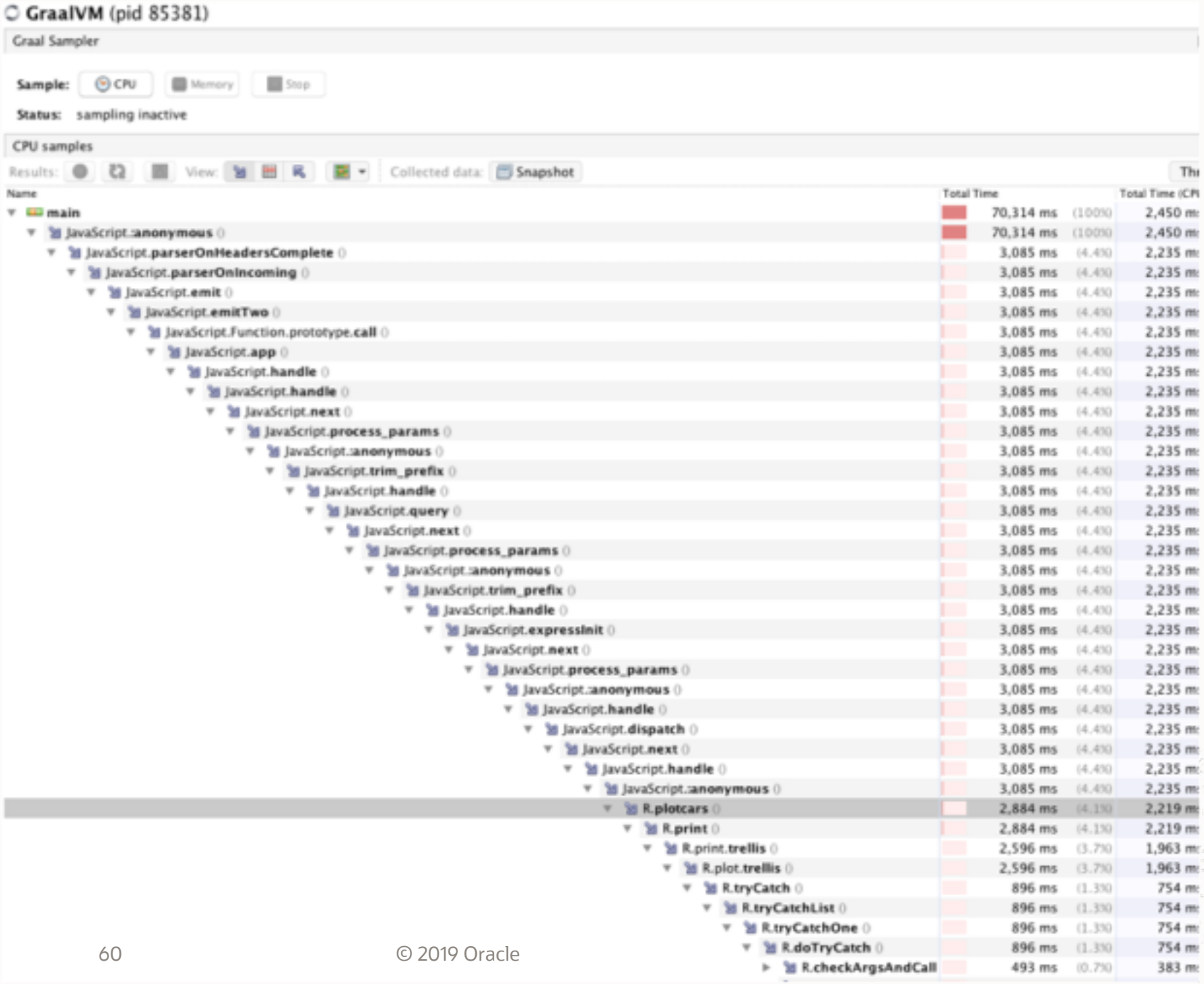
```
const express = require('express');
const app = express();
app.listen(3000);
app.get('/', function(req, res) {
  var text = 'Hello World!';
  const BigInteger = Java.type(
    'java.math.BigInteger');
  text += BigInteger.valueOf(2)
    .pow(100).toString(16);
  text += Polyglot.eval(
    'R', 'runif(100)')[0];
  res.send(text);
})
```



Enable Polyglot interoperability:

node --polyglot

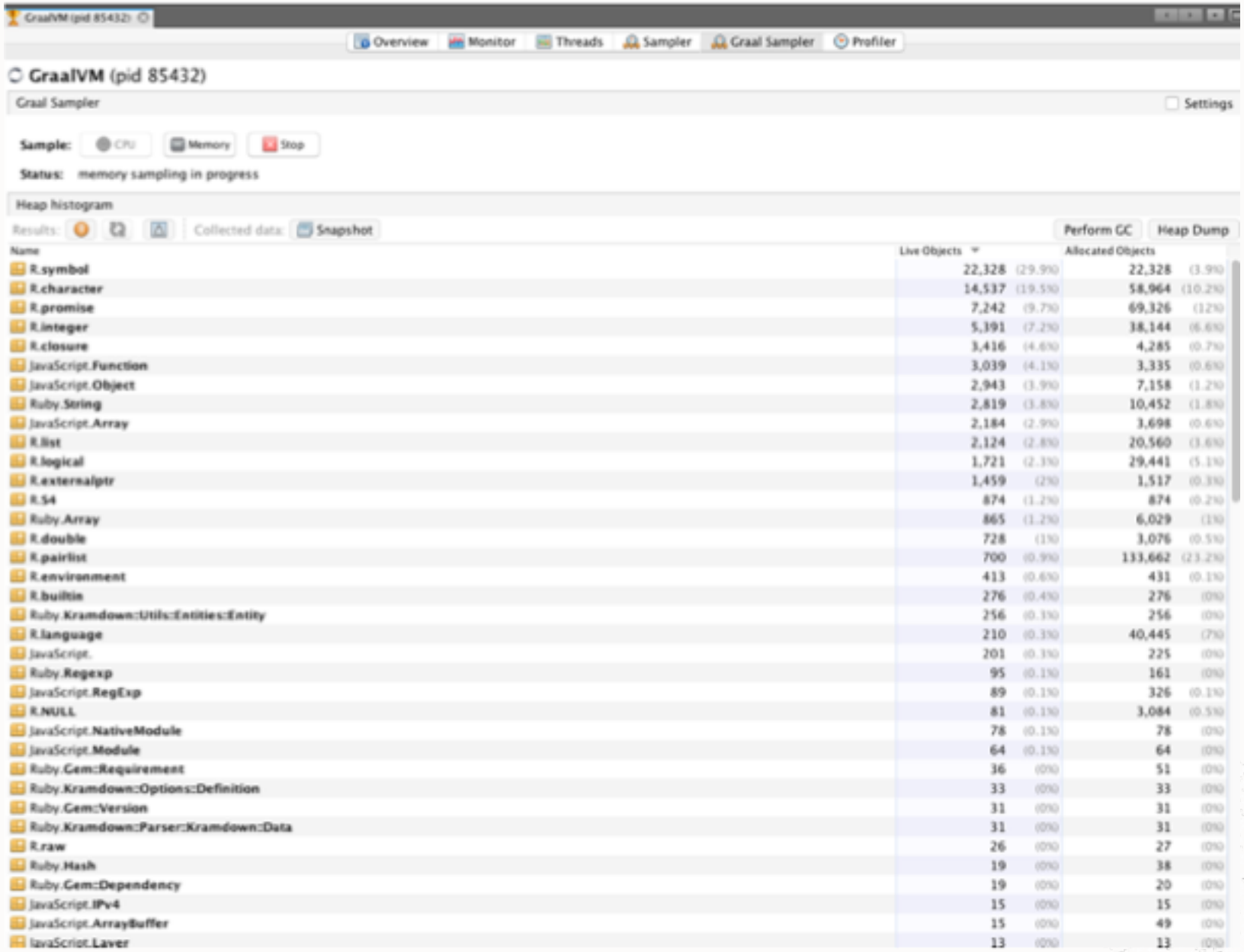
© 2019 Oracle



VisualVM

Polyglot Stack Trace

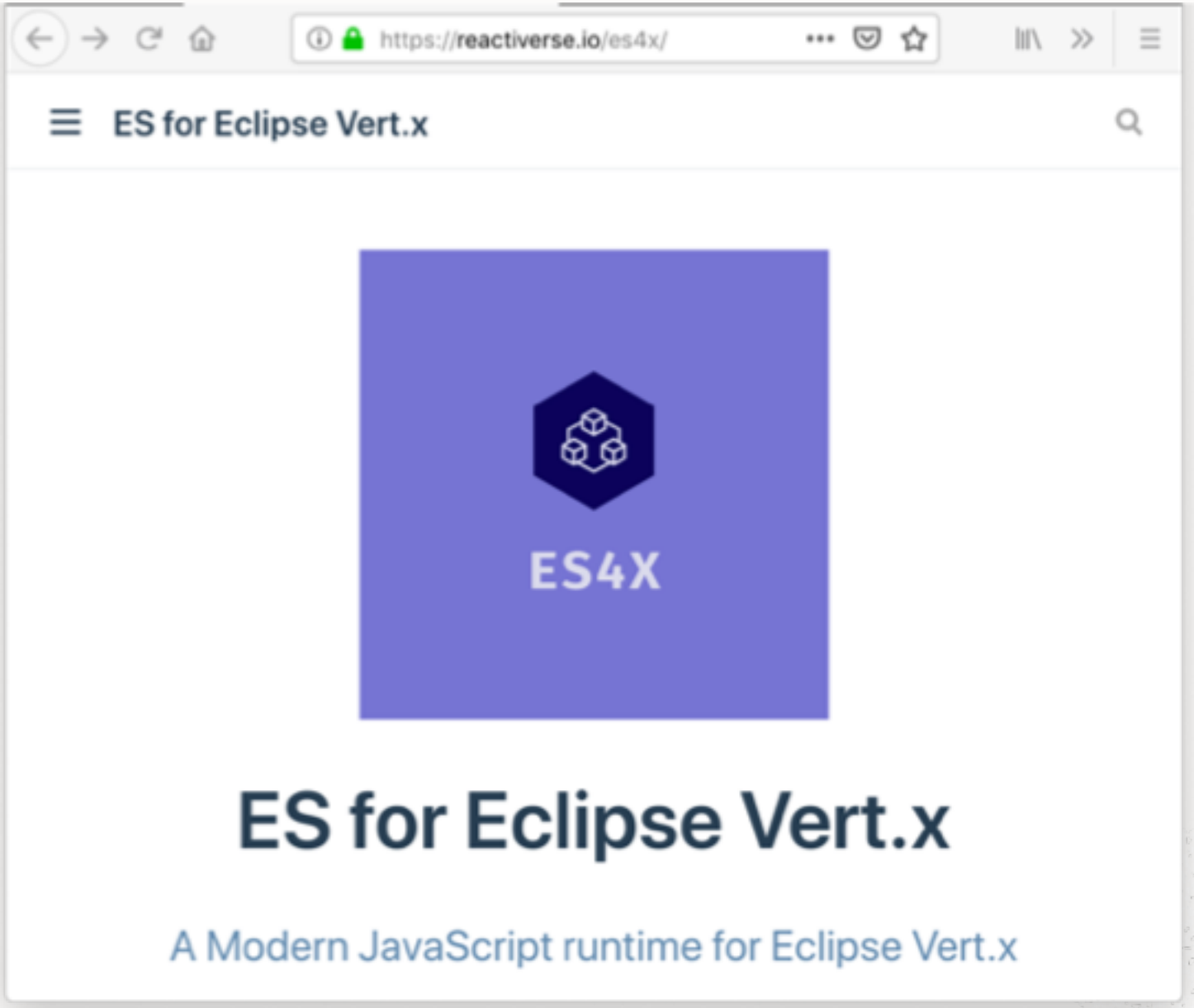
© 2019 Oracle



VisualVM

Polyglot Heap Dump

© 2019 Oracle



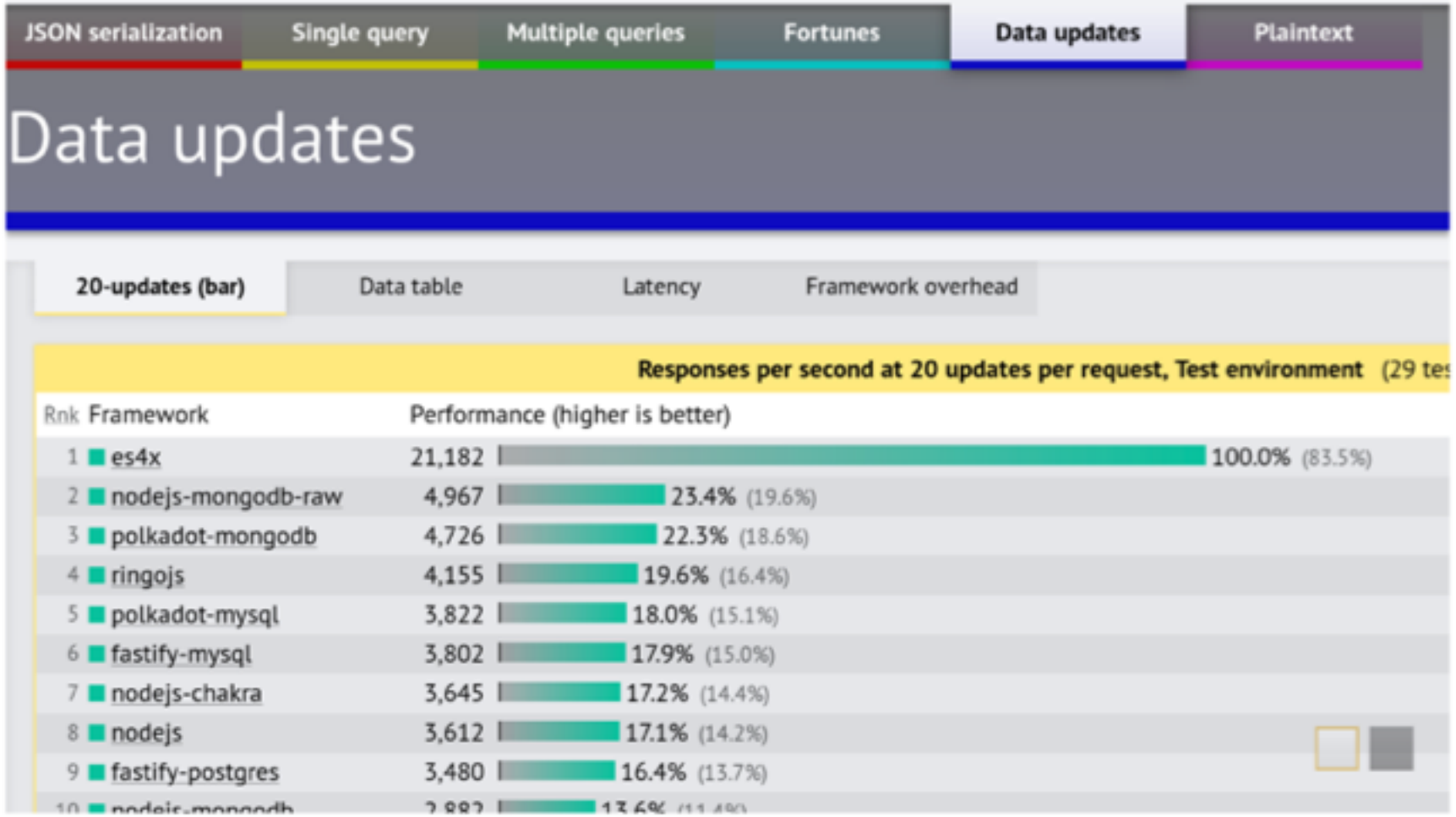
Performant

ES4X runs on top of GraalVM offering a great performance for JavaScript

© 2019 Oracle

Performance

ES4X is **the fastest** `JavaScript` according to TechEmpower Frameworks Benchmark Round #18.
ES4X is the fastest on all tests when compared to `JavaScript` frameworks:



© 2019 Oracle

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.

https://github.com/NVIDIA/grcuda

© 2019 Oracle

ORACLE

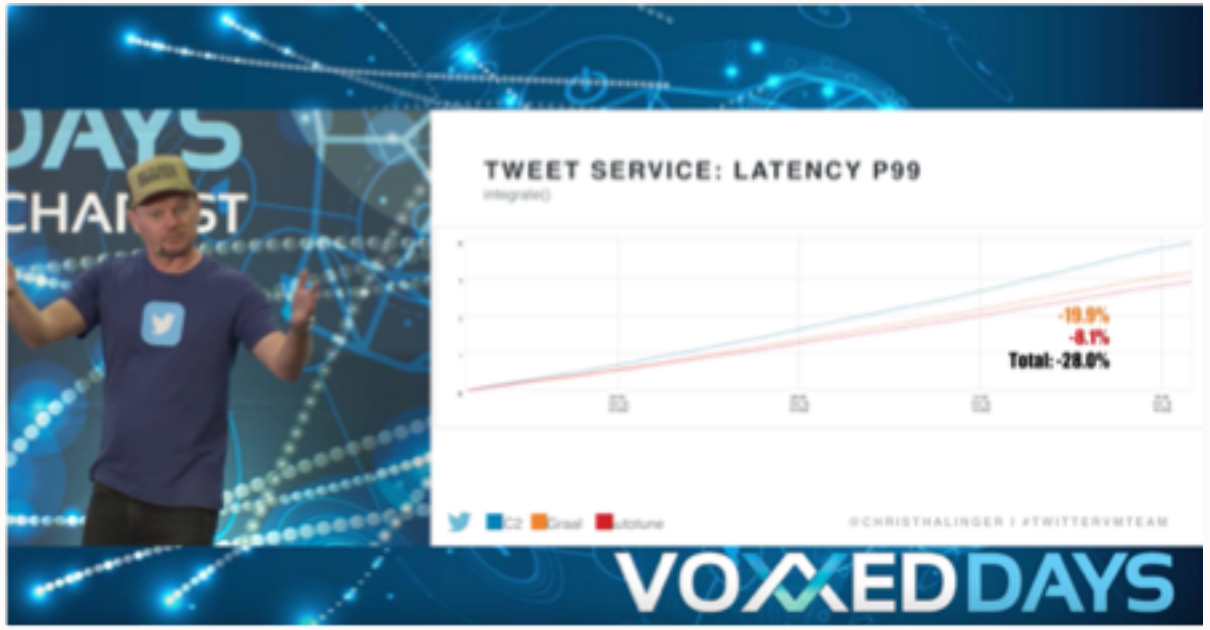
Summary

© 2019 Oracle

Production-Ready	Experimental	Visionary
Java	Ruby	Python
Scala, Groovy, Kotlin	R	VSCode Plugin
JavaScript	LLVM Toolchain	GPU Integration
Node.js		Webassembly
Native Image		LLVM Backend
VisualVM		

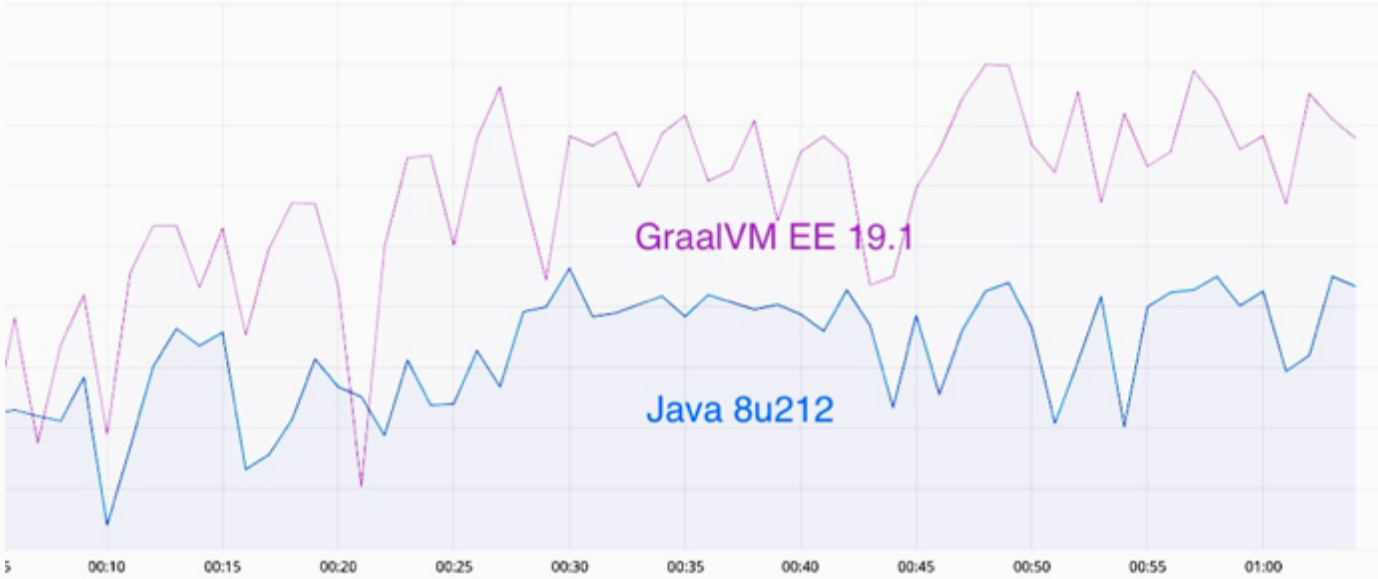
© 2019 Oracle

Twitter uses GraalVM compiler in production to run their Scala microservices



© 2019 Oracle
87
© 2019 Oracle

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



ORACLE
Cloud Infrastructure

© 2019 Oracle
© 2019 Oracle

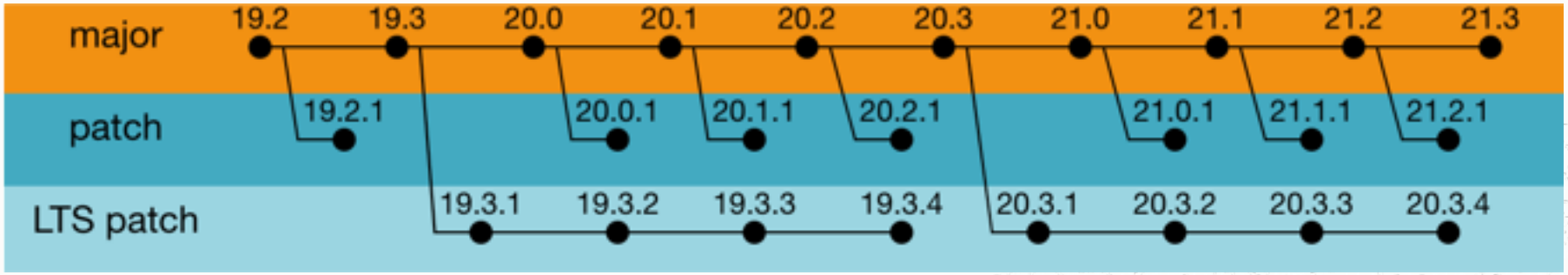
Oracle GraalVM Enterprise Edition

- Higher performance
- Smaller footprint
- Enhanced security for native code
- Oracle Enterprise Support 7x24x365
  - Support directly from the GraalVM Team

© 2019 Oracle

Versioning

- Major release every 3 months named YEAR.x
- Both major and CPU releases on predictable dates
- Last major release of a year receives patches for the following year



© 2019 Oracle

GraalVM Value Proposition

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

© 2019 Oracle

Multiplicative Value-Add of GraalVM Ecosystem

- **Languages**

Java
JavaScript
Ruby
R
Python
C/C++, FORTRAN, …

*

GraalVM

Optimizations
Tooling
Interoperability
Security

*

Embeddings

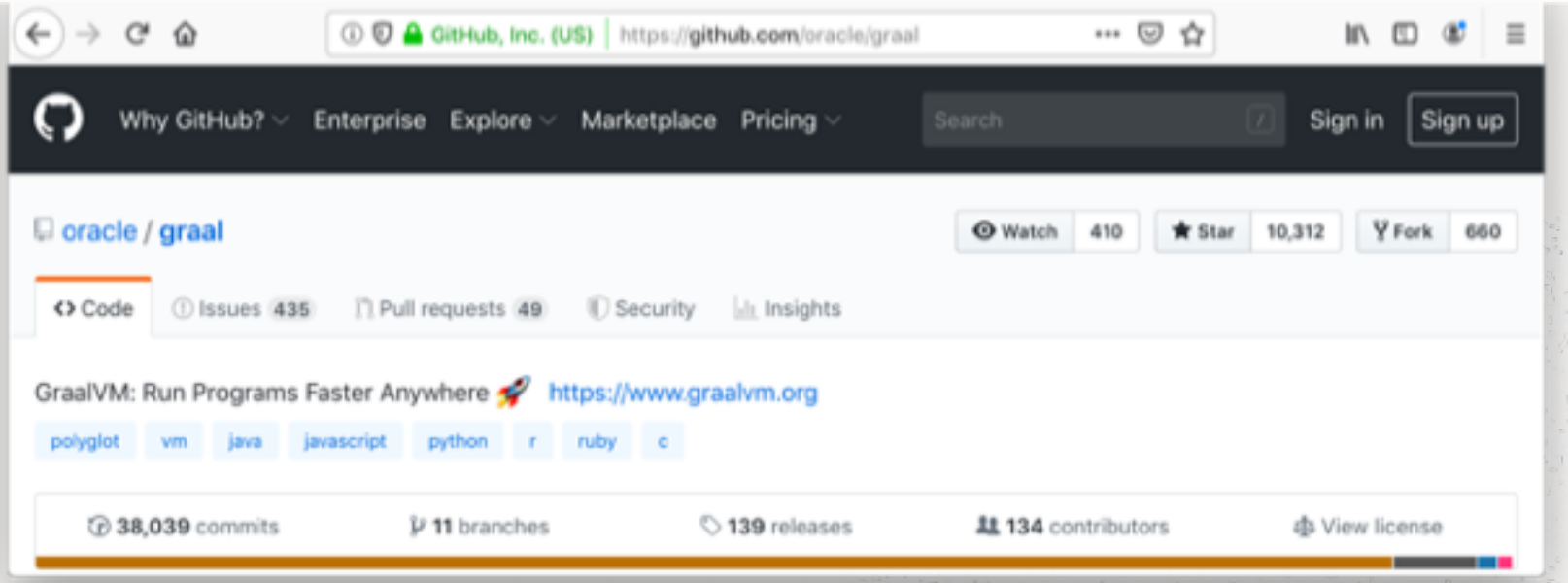
HotSpot JVM
Oracle RDBMS
Node.js
Standalone
Spark
…

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

© 2019 Oracle

Download and use it for your application!

- 100% production ready
- https://www.graalvm.org
- https://github.com/graalvm/
- @graalvm



© 2019 Oracle

Thank you

© 2019 Oracle