

ORACLE®

Maximizing Performance

with

GraalVM™

Thomas Wuerthinger (@thomaswue)

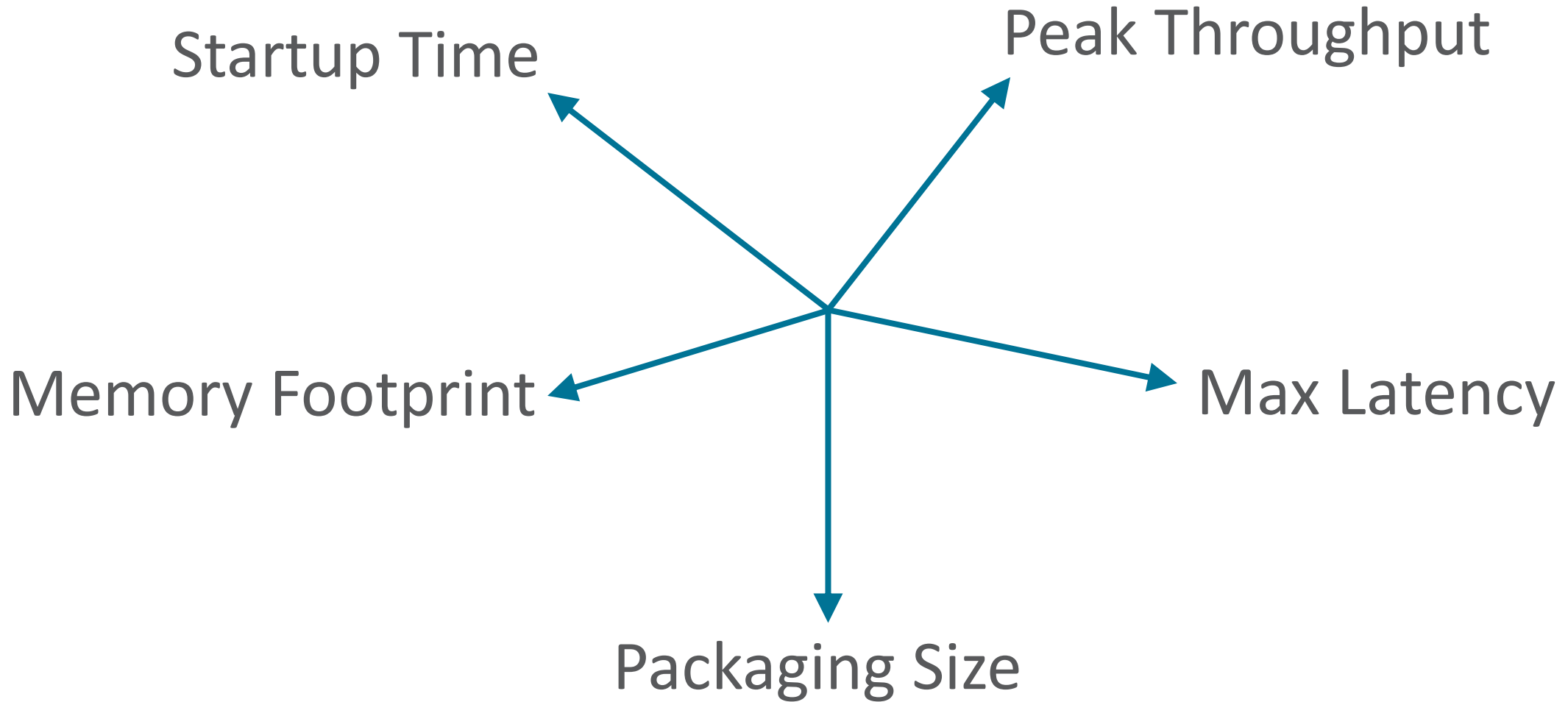
Senior Research Director

Oracle Labs

June 25, 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™



OpenJDK™



ORACLE®
DATABASE



standalone



Community Edition (CE)

GraalVM CE is available for free for development and production use. It is built from the GraalVM sources available on [GitHub](#). We provide pre-built binaries for GraalVM CE for Linux on x86 64-bit systems.

[DOWNLOAD FROM GITHUB](#)

Enterprise Edition (EE)

GraalVM EE provides additional performance, security, and scalability relevant for running critical applications in production. It is free for evaluation uses and available for download from the [Oracle Technology Network](#). We provide binaries for GraalVM EE for Linux or Mac OS X on x86 64-bit systems.

[DOWNLOAD FROM OTN](#)



GraalVM™

JIT

AOT

java MyMainClass

OpenJDK™

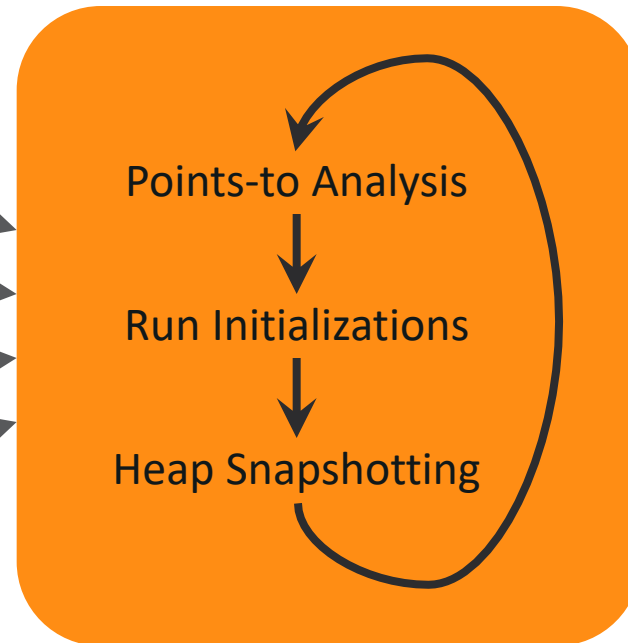
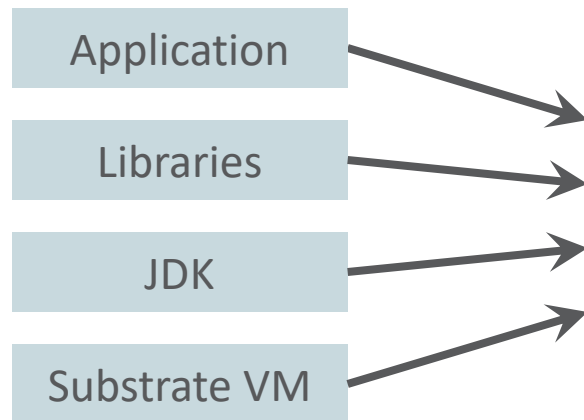
native-image MyMainClass

./mymainclass



GraalVM AOT for Native Images

Input:
All classes from application,
libraries, and VM



Iterative analysis until
fixed point is reached

Ahead-of-Time
Compilation



Output:
Native executable

Code in
Text Section

Image Heap in
Data Section

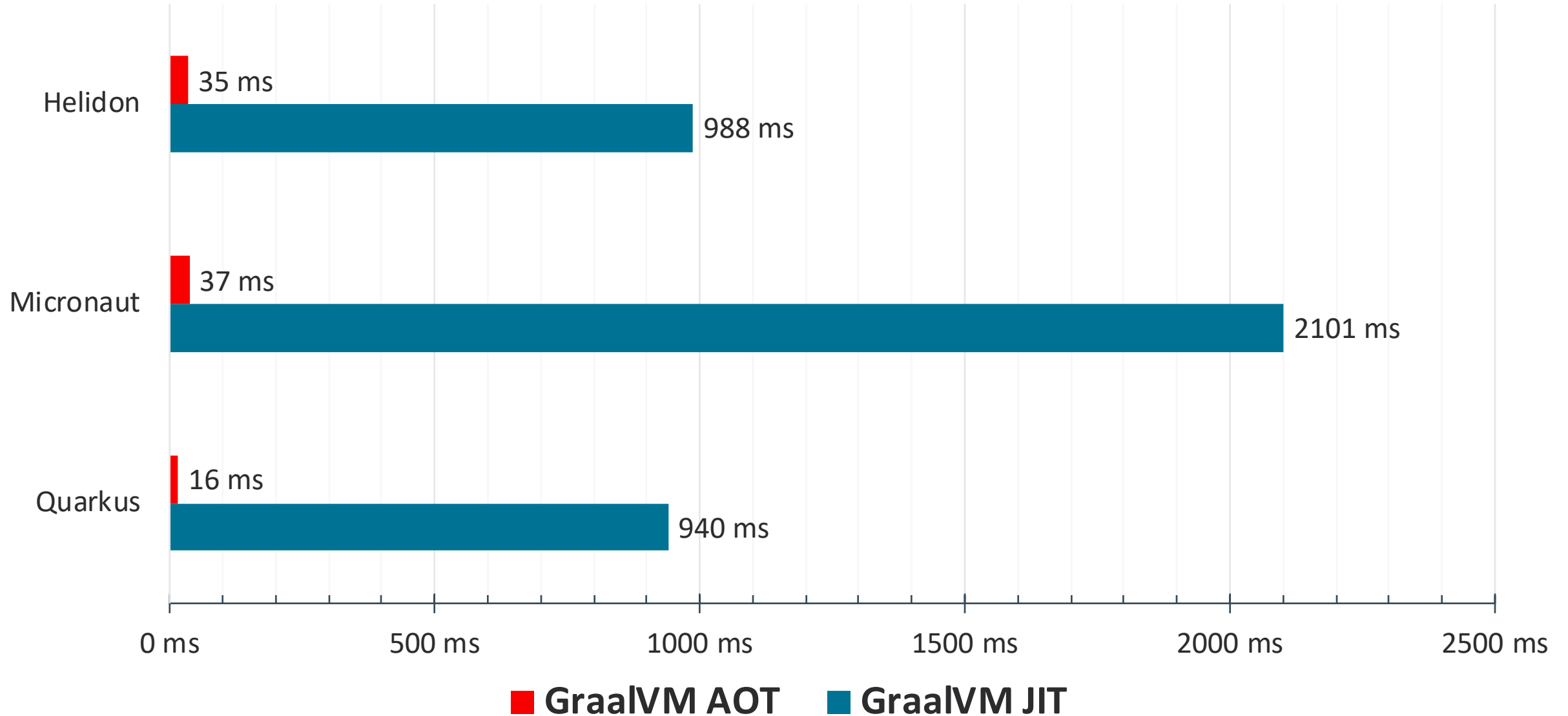
Image Heap
Writing



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

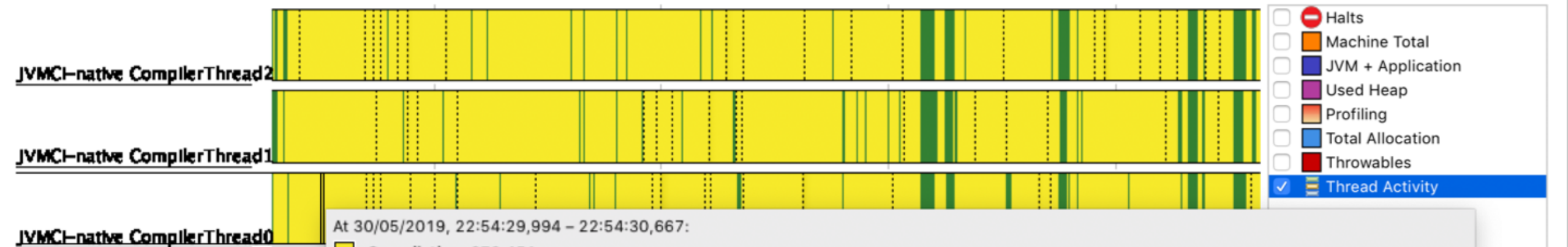
AOT vs JIT: Startup Time



Java Flight Recorder Compilation Information

<No Selection> Aspect: <No Selection>

Thread	Profiling Samples	Total I/O Time	Total Blocked Time	Class Loading Time	Total Allocated
Reference Handler	10				
JVMCI-native CompilerThread2					34,4 M
JVMCI-native CompilerThread1					34,8 M
JVMCI-native CompilerThread0					23,6 M
JFR request timer	2				10,3 M



At 30/05/2019, 22:54:29,994 – 22:54:30,667:

- Compilation: 673,451 ms
- Thread: JVMCI-native CompilerThread0
- Java Method: Object org.scalastyle.scalariform.VisitorHelper\$\$anonfun\$org\$scalastyle\$scalariform\$VisitorHelper\$\$myVisit\$1.apply(Object)
- Compilation ID: 9.633
- Compilation Level: 4
- Succeeded: true
- On Stack Replacement: false
- Compiled Code Size: 35,2 KiB
- Inlined Code Size: 11,3 KiB

At -∞ – ∞:

- Thread Lifespan of JVMCI-native CompilerThread0: N/A

Stack Trace

Pattern\$CharProperty java.util.regex.Pa

Pattern\$Node java.util.regex.Pattern.sec

Pattern\$Node java.util.regex.Pattern.exp

Pattern\$Node java.util.regex.Pattern.group0()

AOT vs JIT: Memory Footprint

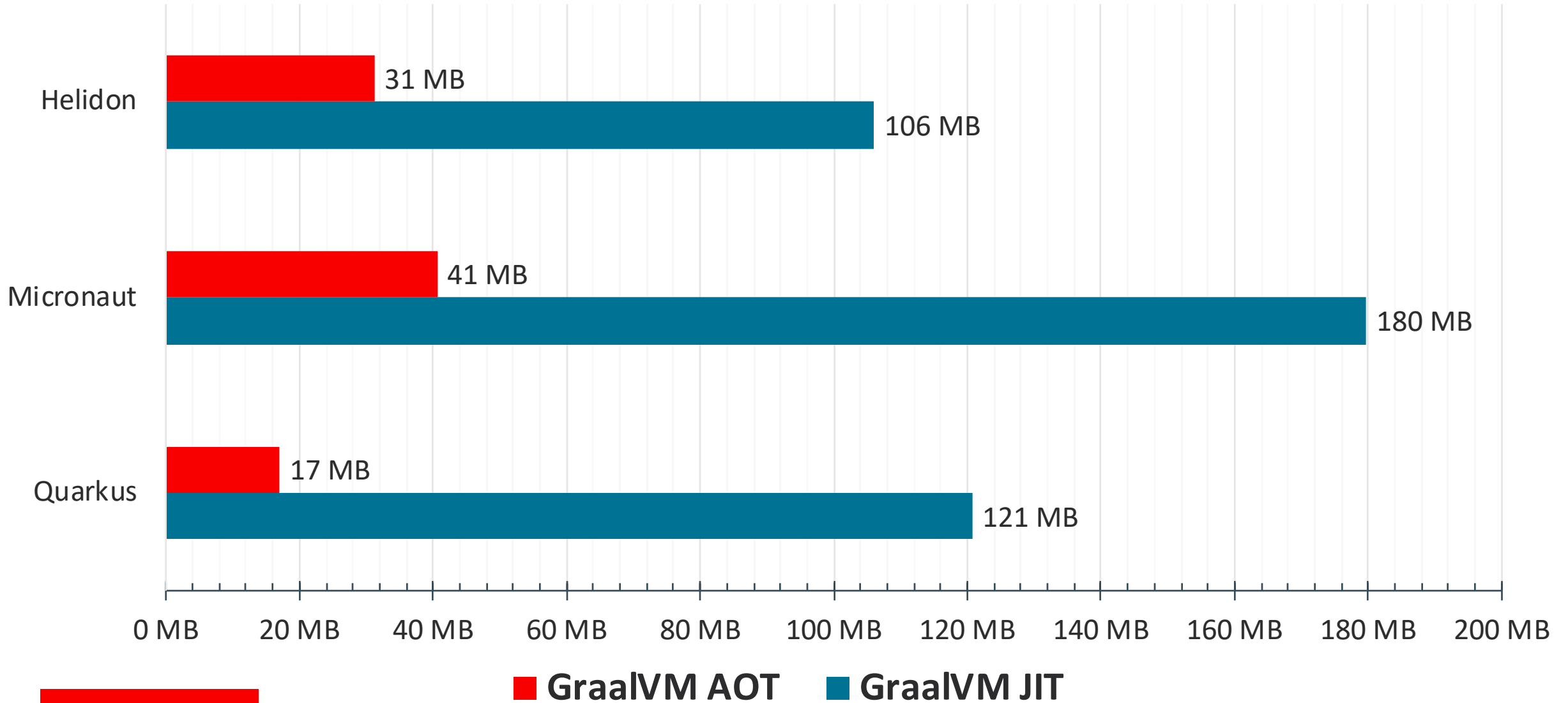
- JIT

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

- AOT

- Loaded application executable
- Application data

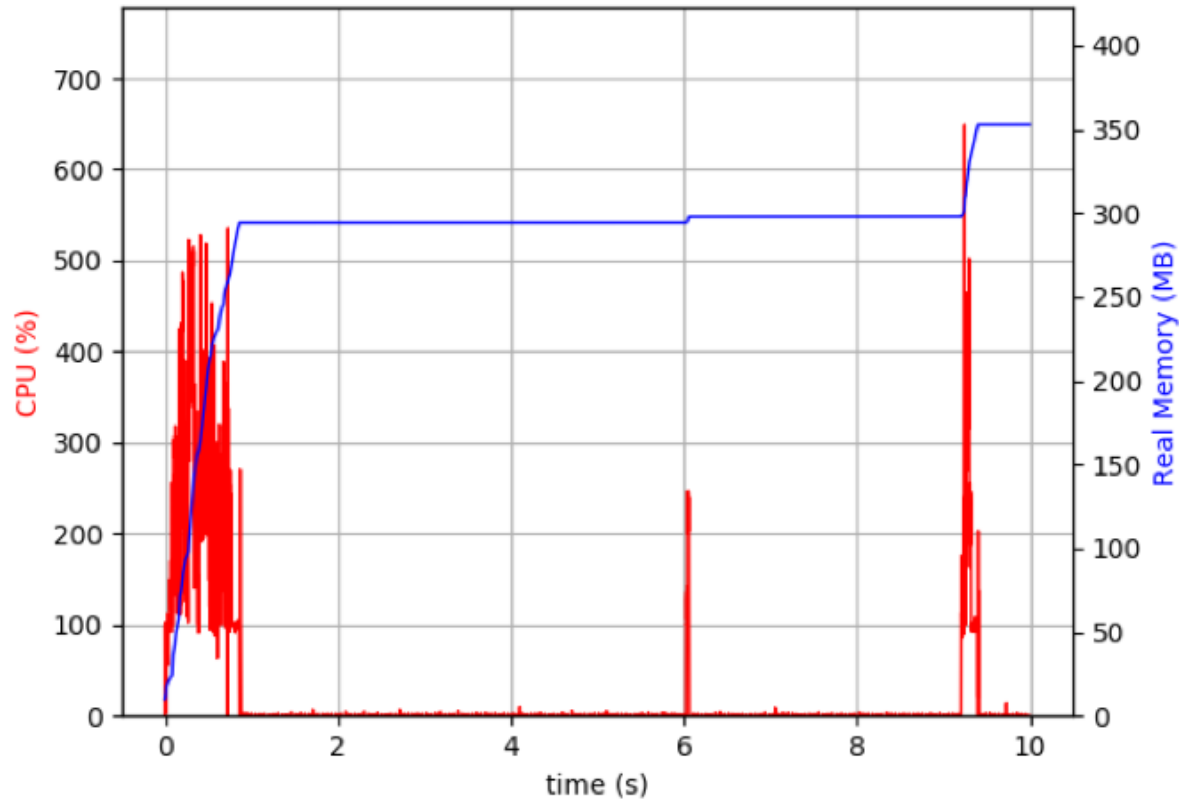
AOT vs JIT: Memory Footprint



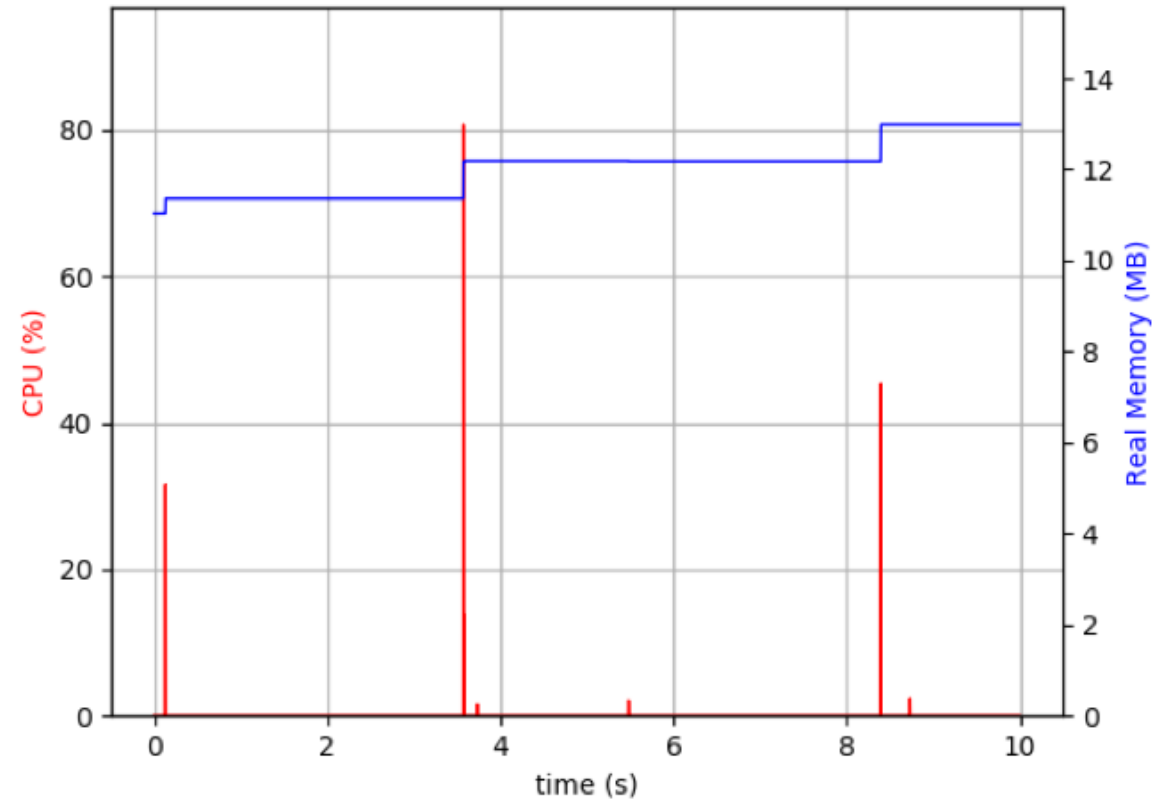
Web Server Startup and Memory Footprint

Starting up and serving 2 requests in the first 10s

- **JIT 800ms / 350Mb**



- **AOT 8ms / 13Mb**



Which is fastest?

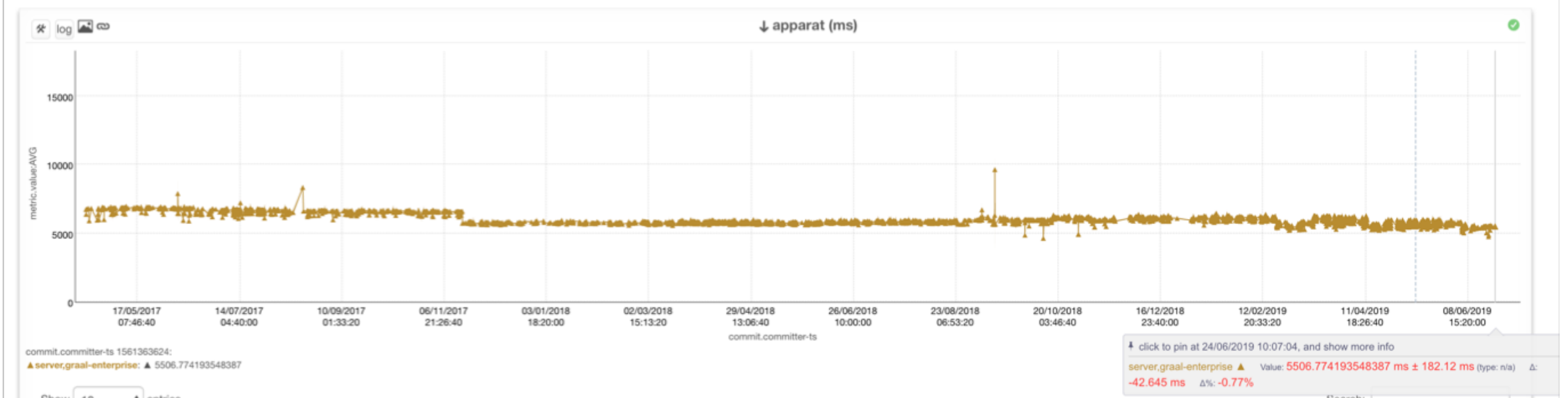
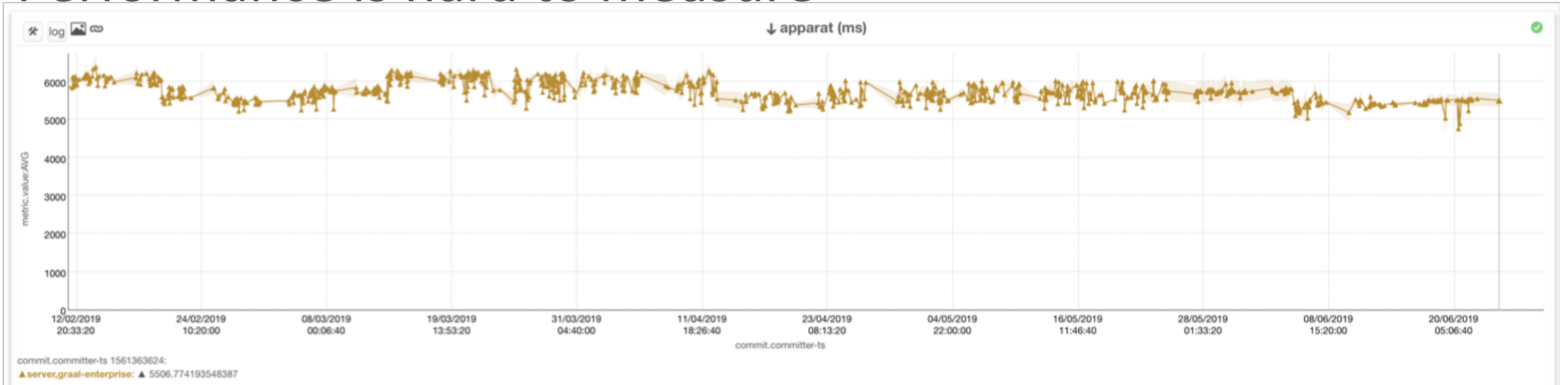
```
int negate1(int a) {  
    return -a;  
}
```

```
int negate2(int a) {  
    int b = a + 0;  
    return -b * 1;  
}
```

```
int negate3(int a) {  
    Object[] array = new Object[] {Integer.valueOf(a)};  
    return -(Integer)array[0];  
}
```

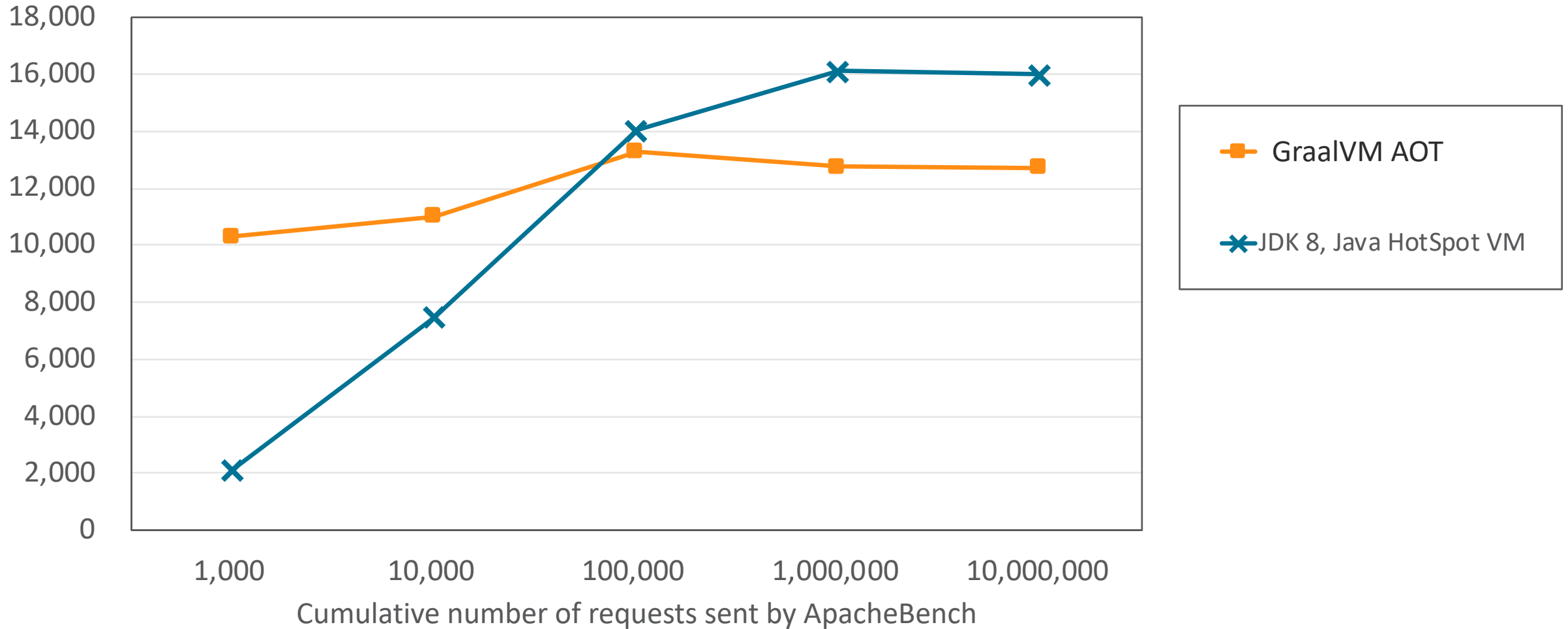
```
static Object[] cachedArray = new Object[1];  
int negate4(int a) {  
    cachedArray[0] = Integer.valueOf(a);  
    return -(Integer)cachedArray[0];  
}
```

Performance is hard to measure

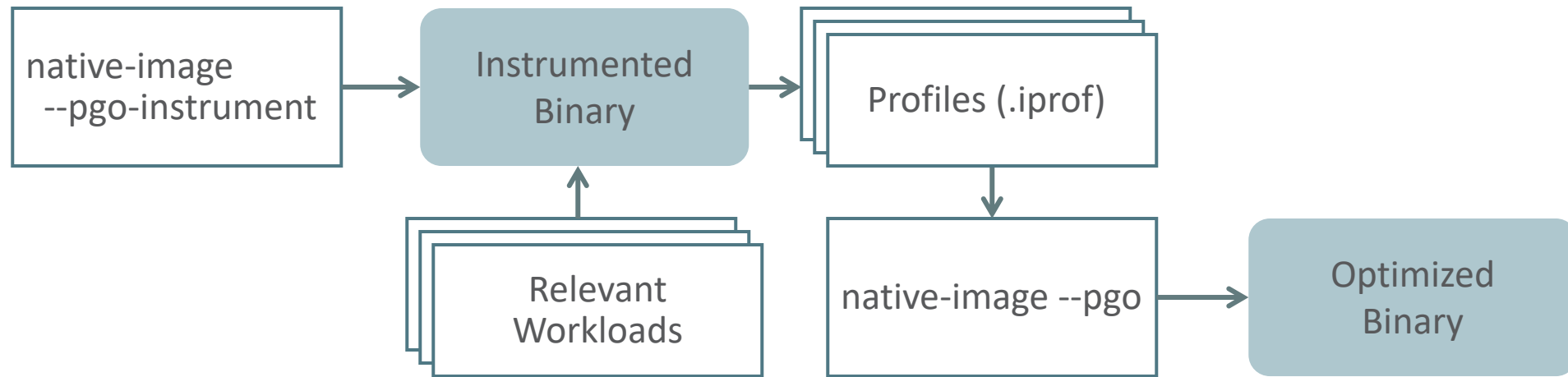


AOT vs JIT: Throughput

Handled requests per second

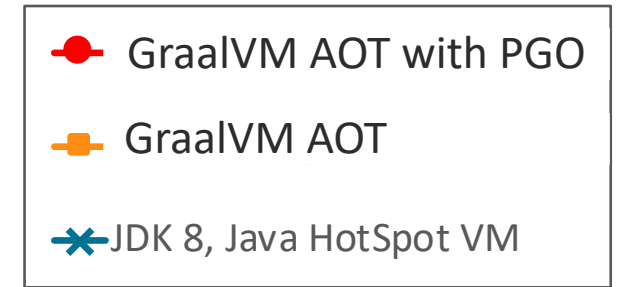
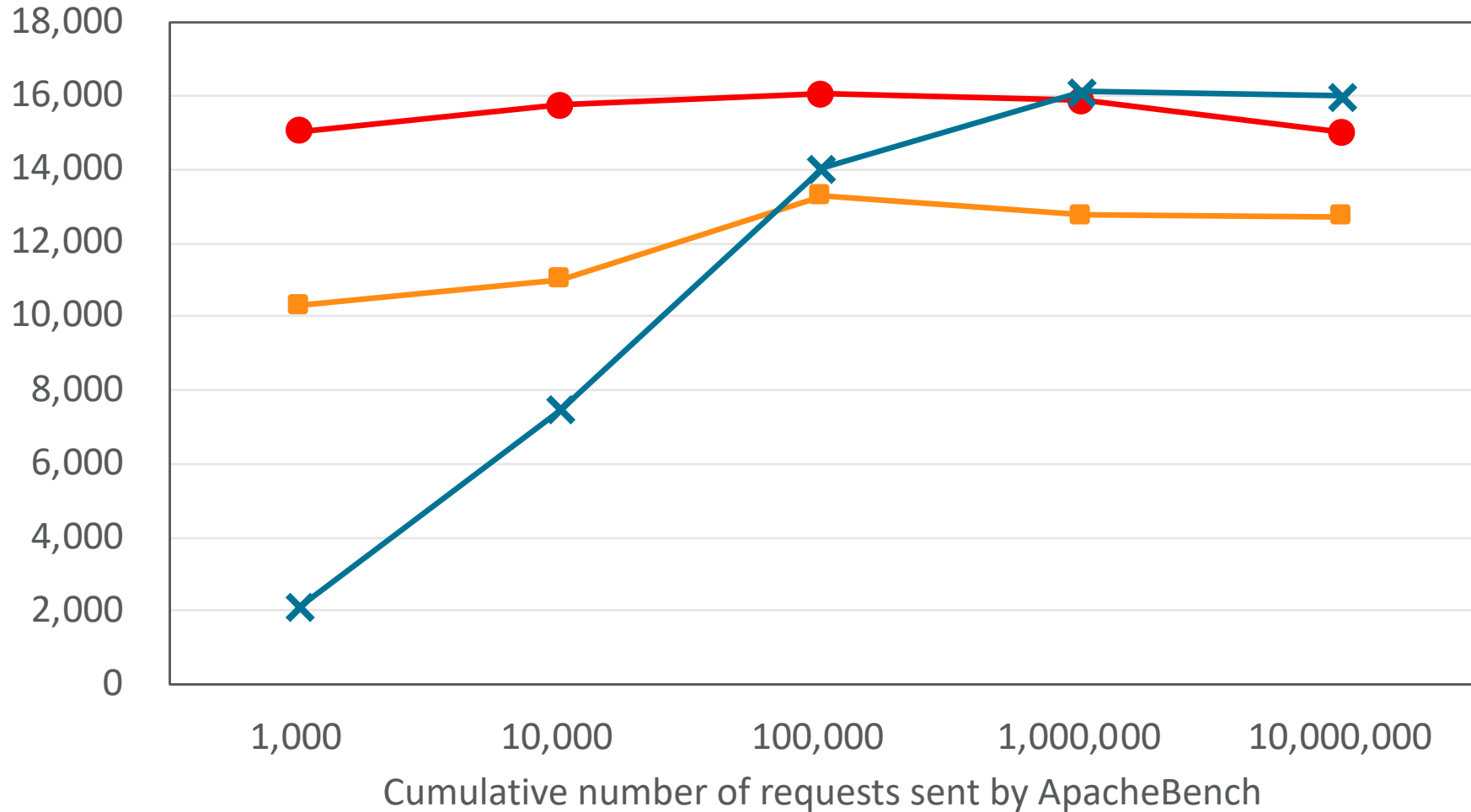


Profile-Guided Optimizations (PGO)



AOT vs JIT: Throughput

Handled requests per second



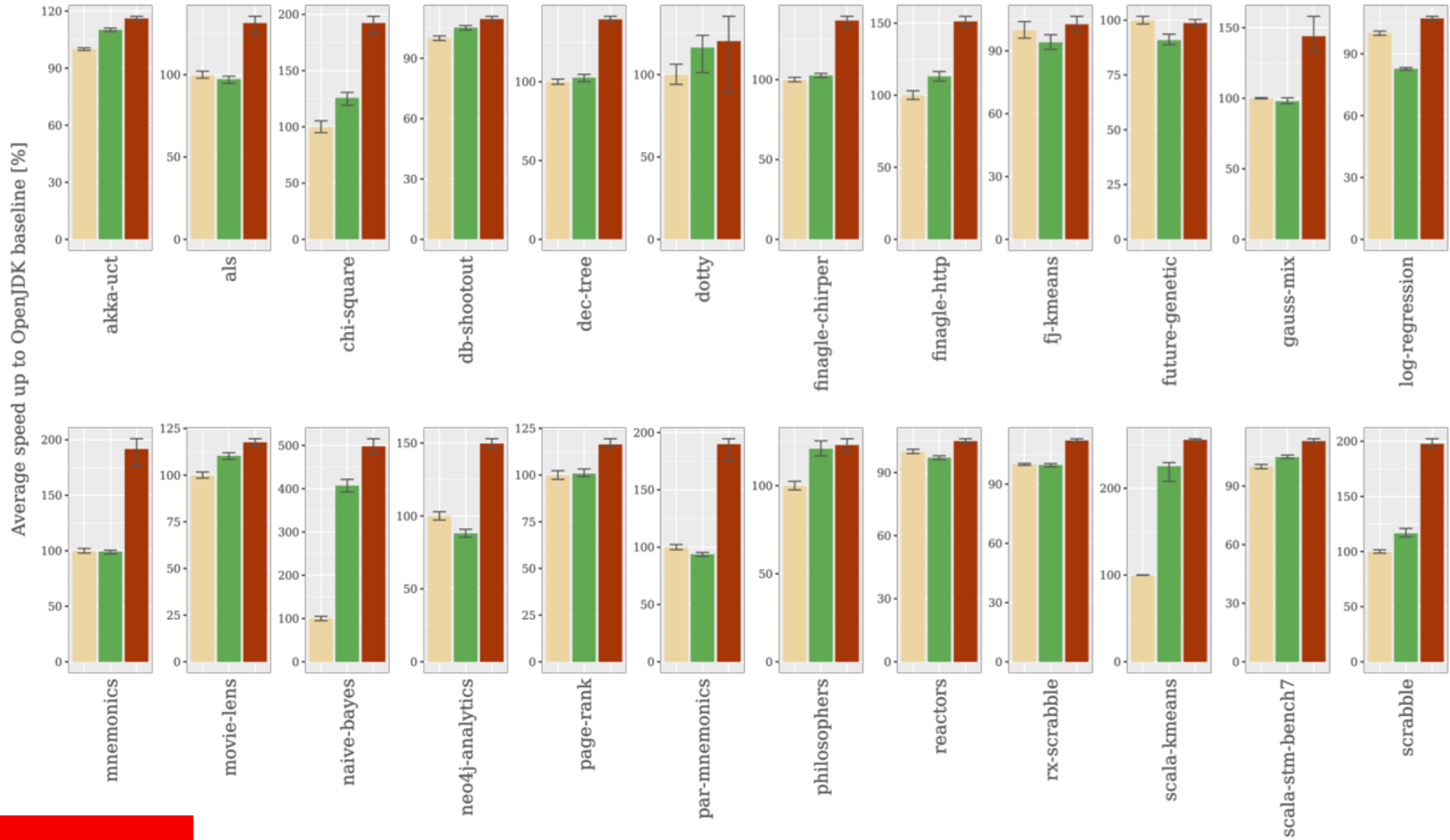
AOT vs JIT: Peak Performance

- JIT
 - Profiling at startup enabled better optimizations
 - Can make optimistic assumptions about the profile and deoptimize
- AOT
 - Needs to handle all cases in machine code
 - Profile-guided optimizations help
 - Predictable performance

More Benchmarks...

- Optimizing a compiler for too few benchmarks results in typical overfitting problems
- Therefore we started together with academic collaborators
<https://renaissance.dev>
- All benchmark data can be interesting; careful with conclusions though.

Renaissance.dev



AOT vs JIT: Max Latency

- JIT

- Many low latency GC options available
 - G1
 - CMS
 - ZGC
 - Shenandoah

- AOT

- Only regular stop© collector
- Assumes small heap configuration
- Can quickly restart; could use load balancer instead of GC

- Peak vs max latency trade-offs:

- Loop safepoints
- Parallel stop-the-world GC

AOT vs JIT: Packaging Size

- JIT
 - Use jlink for smaller package
 - Lightweight docker image (e.g., alpine linux)
- AOT
 - Everything in single binary
 - Can run on bare metal docker
 - Substantially smaller constant overhead
- Peak vs packaging trade-offs:
 - Inlining
 - Code duplication

GraalVM JIT

Peak Throughput

Max Latency

No Configuration

GraalVM AOT

Startup Time

Memory Footprint

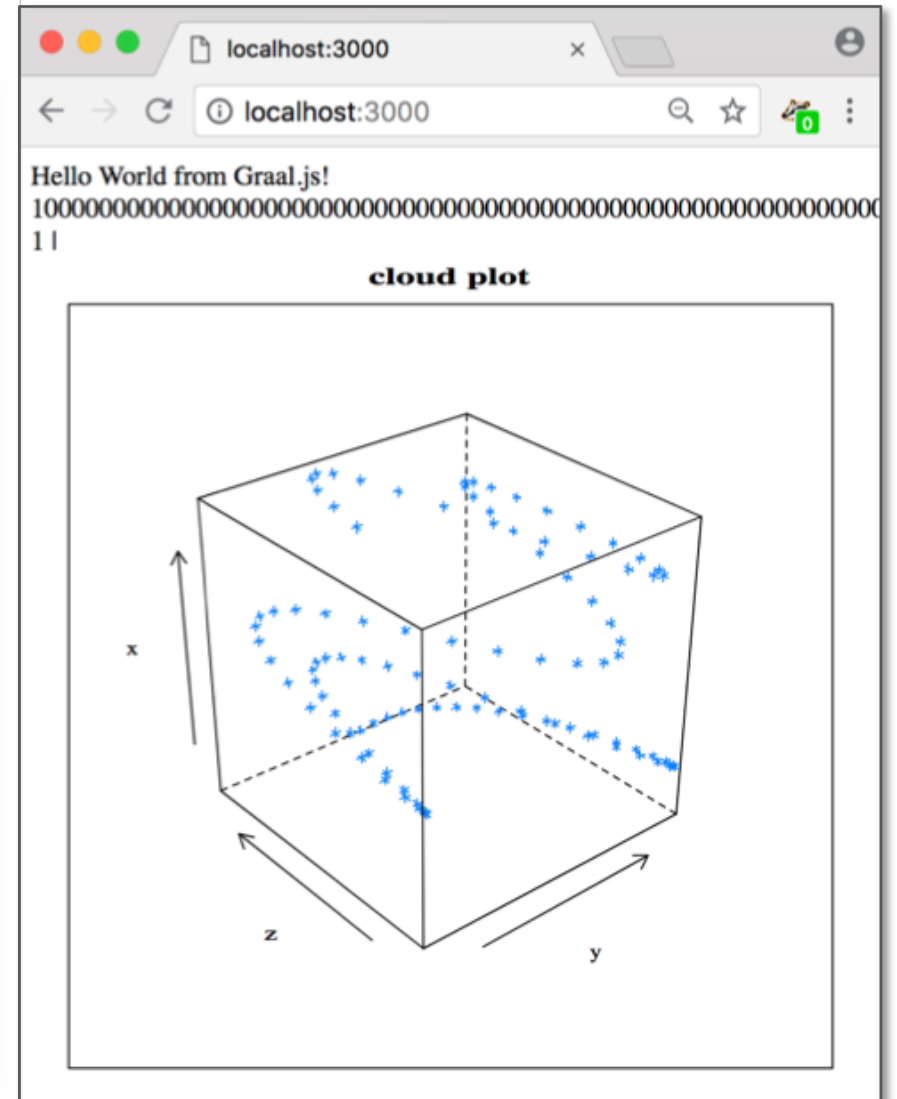
Packaging Size

Can AOT get better?

- Collecting profiles up-front
- Low-latency GC option
- Tracing agent for configuration

GraalVM can to much more...

```
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);
})
```



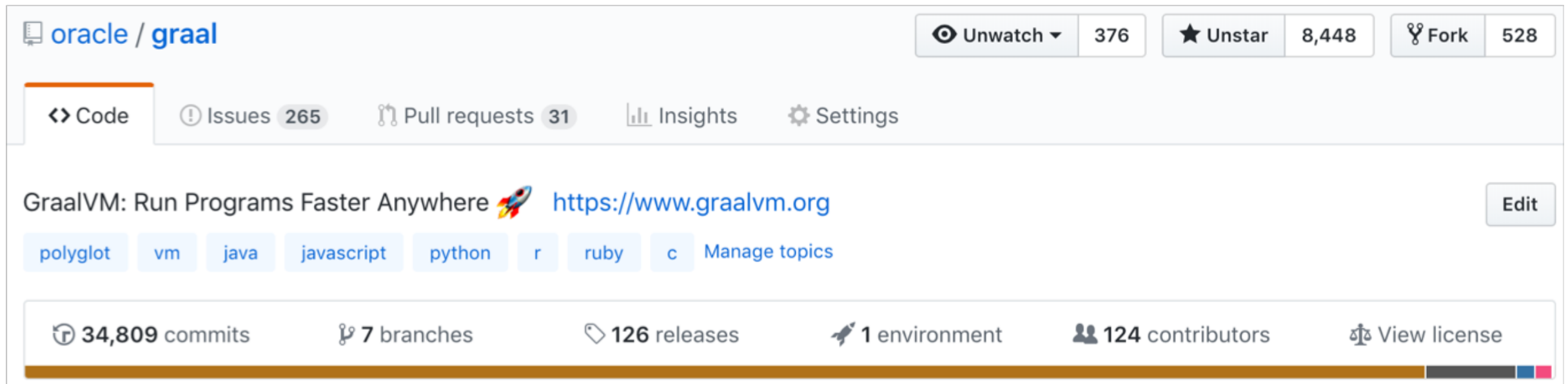
Multiplicative Value-Add of GraalVM Ecosystem



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

GraalVM Community

- <https://www.graalvm.org>
- Open source on GitHub at <https://github.com/oracle/graal>



The screenshot shows the GitHub repository page for 'oracle / graal'. At the top, the repository name is displayed with a folder icon. To the right, there are buttons for 'Unwatch' (376), 'Unstar' (8,448), and 'Fork' (528). Below this, a navigation bar includes 'Code', 'Issues' (265), 'Pull requests' (31), 'Insights', and 'Settings'. The main content area features the repository description: 'GraalVM: Run Programs Faster Anywhere' with a rocket icon and a link to 'https://www.graalvm.org'. Below the description are tags for 'polyglot', 'vm', 'java', 'javascript', 'python', 'r', 'ruby', and 'c', along with a 'Manage topics' link. At the bottom, a statistics bar shows: 34,809 commits, 7 branches, 126 releases, 1 environment, 124 contributors, and a 'View license' link.

Q/A

@graalvm

@thomaswue



The screenshot shows the Twitter profile for GraalVM. The header features the GraalVM logo in a white circle on a dark teal background. Below the header, the profile name 'GraalVM' and handle '@graalvm' are displayed, along with the text 'Follows you'. The bio reads: 'Universal VM for a polyglot world. Our mission: Make development more productive and run programs faster anywhere.' To the right, statistics are shown: 105 Tweets, 40 Following, and 6,121 Followers. The 'Tweets' tab is selected, showing a pinned tweet from April 17, 2018, announcing GraalVM with a link to a blog post and the hashtag #GraalVM.

GraalVM

GraalVM
@graalvm Follows you

Universal VM for a polyglot world. Our mission: Make development more productive and run programs faster anywhere.

Tweets 105 Following 40 Followers 6,121 LI 1

Tweets Tweets & replies

📌 Pinned Tweet

GraalVM @graalvm · 17 Apr 2018
Announcing GraalVM: Run Programs Faster
blogs.oracle.com/developers/announcing-graalvm
#GraalVM

Integrated Cloud

Applications & Platform Services

ORACLE®