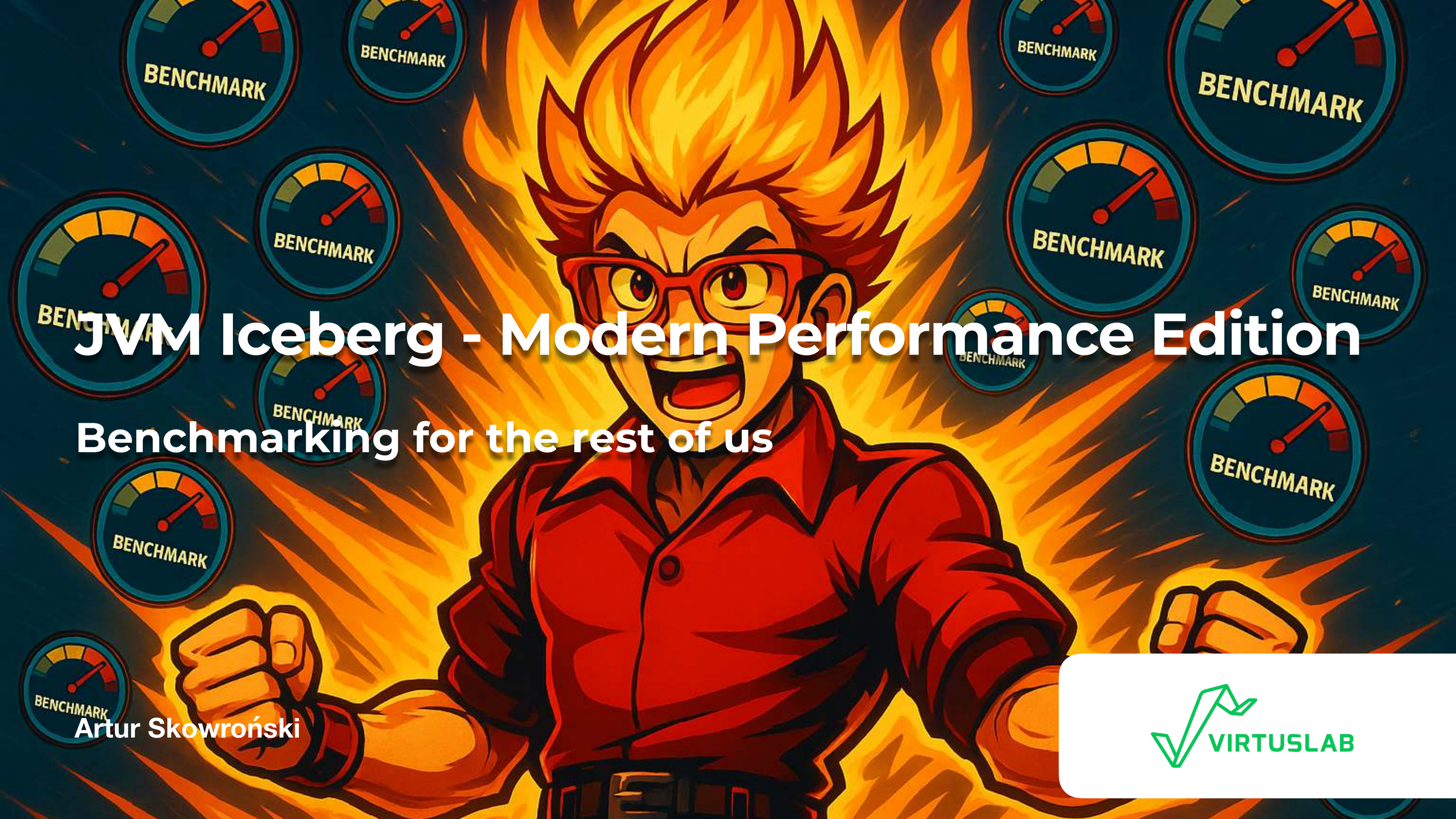



# JVM Iceberg - Modern Performance Edition

## Benchmarking for the rest of us

Artur Skowroński

VIRTUSLAB

Lies, bigger lies and benchmark...

**Artur Skowronski**

Head of Java / Kotlin Development

VIRTUSLAB

# JVM Weekly

Home | Notes | Foojay.io Community Newsletter | Archive | About

## "Optimizing the GC when Migrating Cloud Workloads to Arm" with Kieran Hejma...

Today, we have a guest post from Kieran Hejmadi, an Software and Academic Ecosystem Manager in Arm.

MAY 1 · ARTUR SKOWRONSKI AND KIERAN HEJMADI

Latest | Top | Discussions

### April - The Rest of the Story - JVM Weekly vol. 127

I know there is a lot of April left, not my fault, calendar is working against my publishing schedule this month...

APR 24 · ARTUR SKOWRONSKI

### Java in AWS with Wojtek Gawroński - JVM Weekly vol. 126

I'm experimental mood recently, so today, we have the interview!

APR 17 · ARTUR SKOWRONSKI AND WOJTEK GAWROŃSKI

### Make Final Mean Final... and the other JEPs - JVM Weekly vol. 125

It's been a while since we've seen JEPs! But you know, not the fully baked ones, not the ones that have advanced far i...

**JVM Weekly**

From the latest updates in JVM languages like Java, Kotlin, and Scala to emerging technologies like GraalVM and Quarkus, this newsletter covers a wide range of topics that are of interest to developers and tech enthusiasts.

✓ Subscribed

**Recommendations** MANAGE

- **Tales from the jar side** — Ken Kousen
- **Architecture Weekly** — Oskar Dudycz
- **Engineering Primer** — Bartek Antoniak
- **Tech Talks Weekly**

# SCALA 3.7.0 RELEASED!

Wednesday 7 May 2025

Wojciech Mazur, VirtusLab

*RELEASE*

# Model Context Protocol

Inside your favourite LSP server

*Tomasz Godzik VirtusLab*

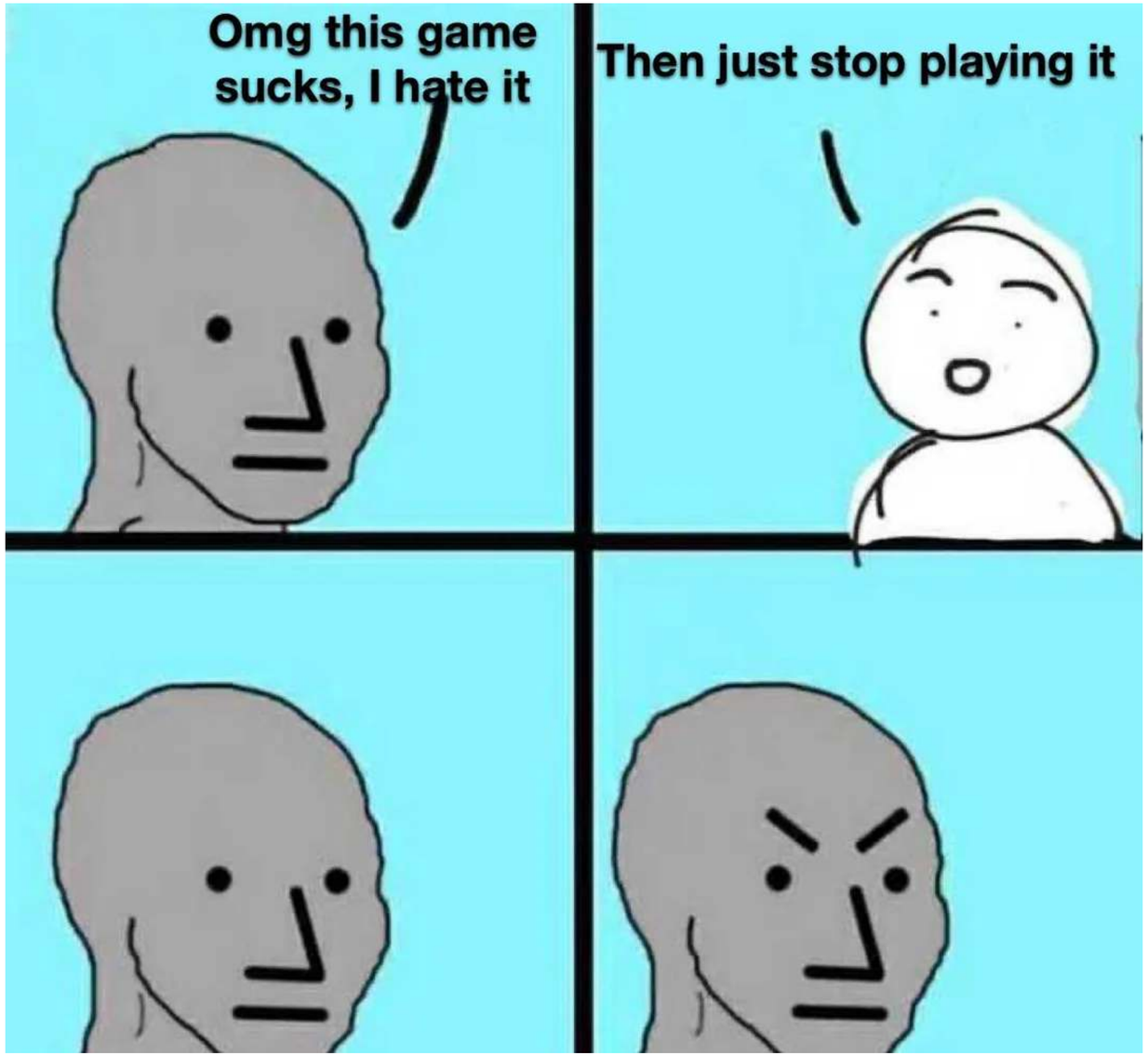
VIRTUSLAB

Bazel
STEWARD

## Bazel Steward

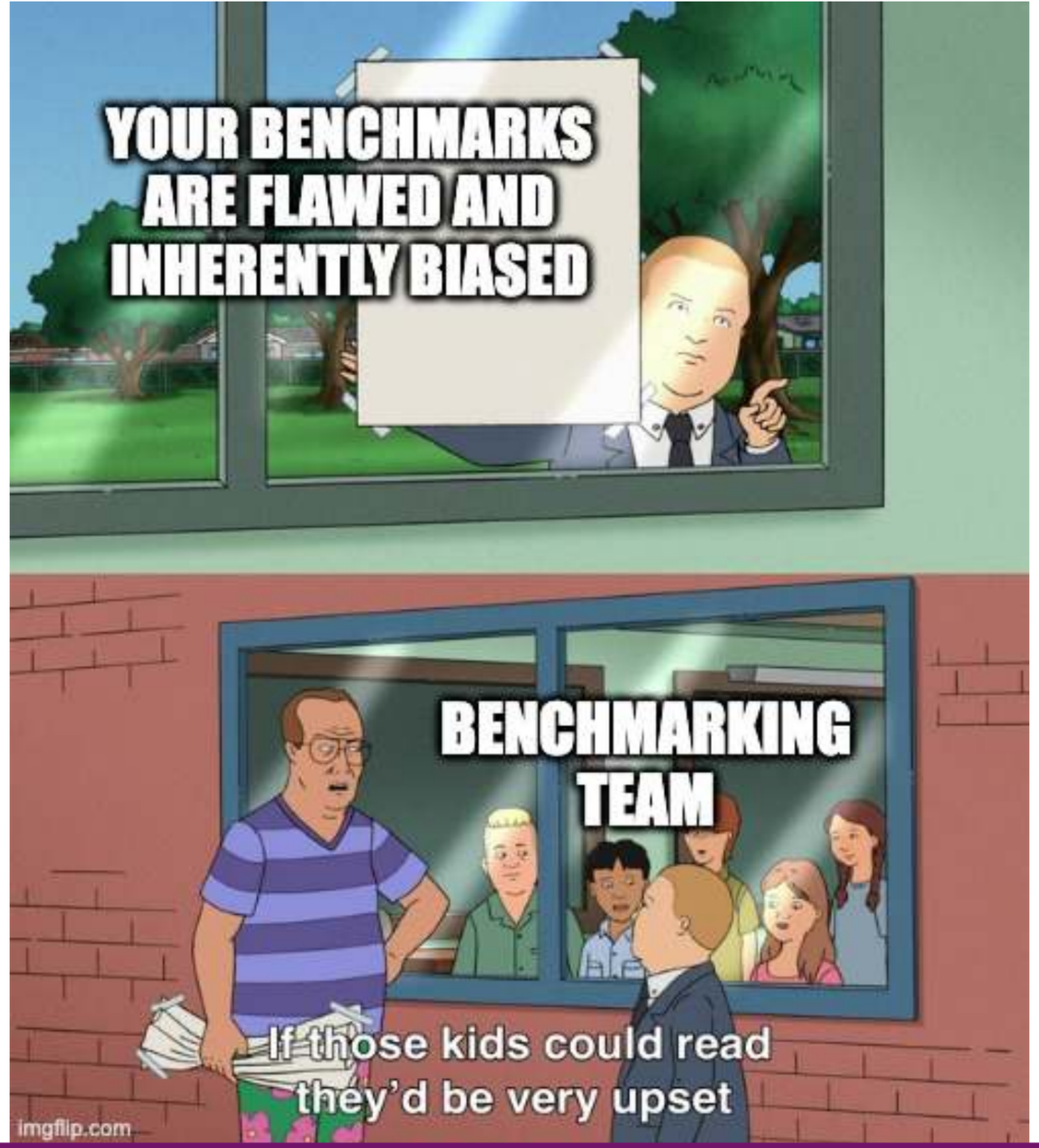
Bazel Steward helping

Bazel Steward is a bot that helps you keep your library dependencies, Bazel and Bazel rules up-to-date.




I hate and love performance at the same time

From one hand, that one of the most interesting topics




# Discussing benchmark is freaking hard

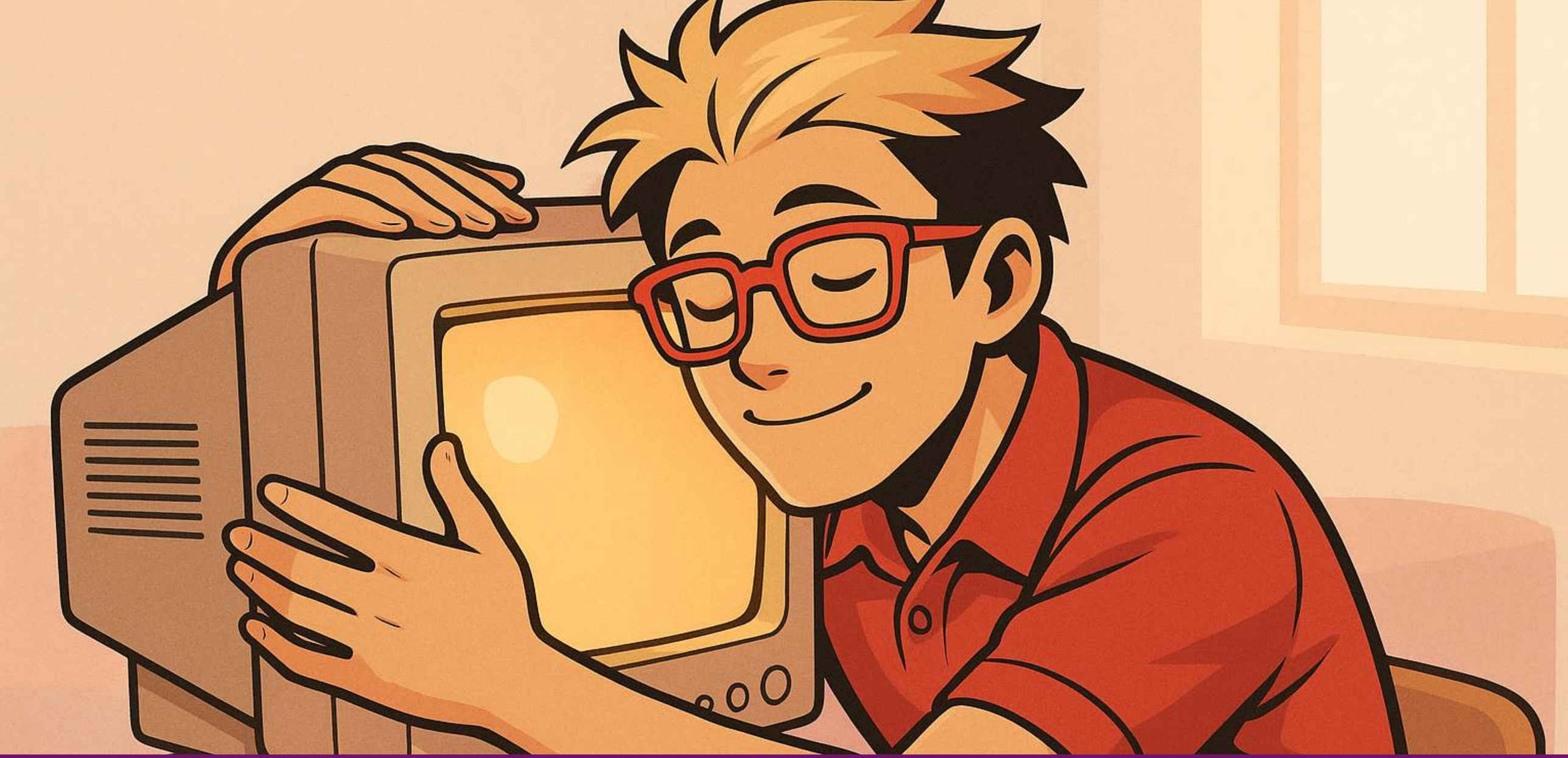
It is very hard to compare apps

# JVM arguments master sheet

Posted by Pankaj Kumar on April 2nd, 2020 at 0 Comments

| | Description |
|---|---|
| -help | Prints the help message. To see more options, use -X. |
| -server | Runs the JVM in server mode. Takes longer to start but offers better performance. |
| -client | Runs the JVM in client mode. Default in 32-bit JVM. Takes less time to start but offers less performance. |
| -version | Prints the version of the JVM. |
| -showversion | Prints the version of the JVM and continues. |
| -cp | Specifies the classpath. |
| -classpath | Specifies the classpath. |
| -D | Sets a system property. |
| -verbose:class | Prints information about classes loaded. |
| -verbose:gc | Prints information about garbage collection. |
| -verbose:jni | Prints information about native method calls. |
| -verbose:sizes | Prints information about the sizes of the JVM. |
| -verbose:stack | Prints information about the stack. |
| -verbose:threads | Prints information about threads. |
| -verbose:module | Prints information about modules. |
| -verbose:gc,class | Prints information about garbage collection and classes loaded. |
| -verbose:gc,jni | Prints information about garbage collection and native method calls. |
| -verbose:gc,sizes | Prints information about garbage collection and the sizes of the JVM. |
| -verbose:gc,stack | Prints information about garbage collection and the stack. |
| -verbose:gc,threads | Prints information about garbage collection and threads. |
| -verbose:gc,module | Prints information about garbage collection and modules. |
| -verbose:class,jni | Prints information about classes loaded and native method calls. |
| -verbose:class,sizes | Prints information about classes loaded and the sizes of the JVM. |
| -verbose:class,stack | Prints information about classes loaded and the stack. |
| -verbose:class,threads | Prints information about classes loaded and threads. |
| -verbose:class,module | Prints information about classes loaded and modules. |
| -verbose:jni,sizes | Prints information about native method calls and the sizes of the JVM. |
| -verbose:jni,stack | Prints information about native method calls and the stack. |
| -verbose:jni,threads | Prints information about native method calls and threads. |
| -verbose:jni,module | Prints information about native method calls and modules. |
| -verbose:sizes,stack | Prints information about the sizes of the JVM and the stack. |
| -verbose:sizes,threads | Prints information about the sizes of the JVM and threads. |
| -verbose:sizes,module | Prints information about the sizes of the JVM and modules. |
| -verbose:stack,threads | Prints information about the stack and threads. |
| -verbose:stack,module | Prints information about the stack and modules. |
| -verbose:threads,module | Prints information about threads and modules. |
| -verbose:gc,class,jni | Prints information about garbage collection, classes loaded and native method calls. |
| -verbose:gc,class,sizes | Prints information about garbage collection, classes loaded and the sizes of the JVM. |
| -verbose:gc,class,stack | Prints information about garbage collection, classes loaded and the stack. |
| -verbose:gc,class,threads | Prints information about garbage collection, classes loaded and threads. |
| -verbose:gc,class,module | Prints information about garbage collection, classes loaded and modules. |
| -verbose:gc,jni,sizes | Prints information about garbage collection, native method calls and the sizes of the JVM. |
| -verbose:gc,jni,stack | Prints information about garbage collection, native method calls and the stack. |
| -verbose:gc,jni,threads | Prints information about garbage collection, native method calls and threads. |
| -verbose:gc,jni,module | Prints information about garbage collection, native method calls and modules. |
| -verbose:gc,sizes,stack | Prints information about garbage collection, the sizes of the JVM and the stack. |
| -verbose:gc,sizes,threads | Prints information about garbage collection, the sizes of the JVM and threads. |
| -verbose:gc,sizes,module | Prints information about garbage collection, the sizes of the JVM and modules. |
| -verbose:gc,stack,threads | Prints information about garbage collection, the stack and threads. |
| -verbose:gc,stack,module | Prints information about garbage collection, the stack and modules. |
| -verbose:gc,threads,module | Prints information about garbage collection, threads and modules. |
| -verbose:class,jni,sizes | Prints information about classes loaded, native method calls and the sizes of the JVM. |
| -verbose:class,jni,stack | Prints information about classes loaded, native method calls and the stack. |
| -verbose:class,jni,threads | Prints information about classes loaded, native method calls and threads. |
| -verbose:class,jni,module | Prints information about classes loaded, native method calls and modules. |
| -verbose:class,sizes,stack | Prints information about classes loaded, the sizes of the JVM and the stack. |
| -verbose:class,sizes,threads | Prints information about classes loaded, the sizes of the JVM and threads. |
| -verbose:class,sizes,module | Prints information about classes loaded, the sizes of the JVM and modules. |
| -verbose:class,stack,threads | Prints information about classes loaded, the stack and threads. |
| -verbose:class,stack,module | Prints information about classes loaded, the stack and modules. |
| -verbose:class,threads,module | Prints information about classes loaded, threads and modules. |
| -verbose:jni,sizes,stack | Prints information about native method calls, the sizes of the JVM and the stack. |
| -verbose:jni,sizes,threads | Prints information about native method calls, the sizes of the JVM and threads. |
| -verbose:jni,sizes,module | Prints information about native method calls, the sizes of the JVM and modules. |
| -verbose:jni,stack,threads | Prints information about native method calls, the stack and threads. |
| -verbose:jni,stack,module | Prints information about native method calls, the stack and modules. |
| -verbose:jni,threads,module | Prints information about native method calls, threads and modules. |
| -verbose:sizes,stack,threads | Prints information about the sizes of the JVM, the stack and threads. |
| -verbose:sizes,stack,module | Prints information about the sizes of the JVM, the stack and modules. |
| -verbose:sizes,threads,module | Prints information about the sizes of the JVM, threads and modules. |
| -verbose:stack,threads,module | Prints information about the stack, threads and modules. |
| -verbose:gc,class,jni,sizes | Prints information about garbage collection, classes loaded, native method calls and the sizes of the JVM. |
| -verbose:gc,class,jni,stack | Prints information about garbage collection, classes loaded, native method calls and the stack. |
| -verbose:gc,class,jni,threads | Prints information about garbage collection, classes loaded, native method calls and threads. |
| -verbose:gc,class,jni,module | Prints information about garbage collection, classes loaded, native method calls and modules. |
| -verbose:gc,class,sizes,stack | Prints information about garbage collection, classes loaded, the sizes of the JVM and the stack. |
| -verbose:gc,class,sizes,threads | Prints information about garbage collection, classes loaded, the sizes of the JVM and threads. |
| -verbose:gc,class,sizes,module | Prints information about garbage collection, classes loaded, the sizes of the JVM and modules. |
| -verbose:gc,class,stack,threads | Prints information about garbage collection, classes loaded, the stack and threads. |
| -verbose:gc,class,stack,module | Prints information about garbage collection, classes loaded, the stack and modules. |
| -verbose:gc,class,threads,module | Prints information about garbage collection, classes loaded, threads and modules. |
| -verbose:gc,jni,sizes,stack | Prints information about garbage collection, native method calls, the sizes of the JVM and the stack. |
| -verbose:gc,jni,sizes,threads | Prints information about garbage collection, native method calls, the sizes of the JVM and threads. |
| -verbose:gc,jni,sizes,module | Prints information about garbage collection, native method calls, the sizes of the JVM and modules. |
| -verbose:gc,jni,stack,threads | Prints information about garbage collection, native method calls, the stack and threads. |
| -verbose:gc,jni,stack,module | Prints information about garbage collection, native method calls, the stack and modules. |
| -verbose:gc,jni,threads,module | Prints information about garbage collection, native method calls, threads and modules. |
| -verbose:gc,sizes,stack,threads | Prints information about garbage collection, the sizes of the JVM, the stack and threads. |
| -verbose:gc,sizes,stack,module | Prints information about garbage collection, the sizes of the JVM, the stack and modules. |
| -verbose:gc,sizes,threads,module | Prints information about garbage collection, the sizes of the JVM, threads and modules. |
| -verbose:gc,stack,threads,module | Prints information about garbage collection, the stack, threads and modules. |
| -verbose:class,jni,sizes,stack | Prints information about classes loaded, native method calls, the sizes of the JVM and the stack. |
| -verbose:class,jni,sizes,threads | Prints information about classes loaded, native method calls, the sizes of the JVM and threads. |
| -verbose:class,jni,sizes,module | Prints information about classes loaded, native method calls, the sizes of the JVM and modules. |
| -verbose:class,jni,stack,threads | Prints information about classes loaded, native method calls, the stack and threads. |
| -verbose:class,jni,stack,module | Prints information about classes loaded, native method calls, the stack and modules. |
| -verbose:class,jni,threads,module | Prints information about classes loaded, native method calls, threads and modules. |
| -verbose:class,sizes,stack,threads | Prints information about classes loaded, the sizes of the JVM, the stack and threads. |
| -verbose:class,sizes,stack,module | Prints information about classes loaded, the sizes of the JVM, the stack and modules. |
| -verbose:class,sizes,threads,module | Prints information about classes loaded, the sizes of the JVM, threads and modules. |
| -verbose:class,stack,threads,module | Prints information about classes loaded, the stack, threads and modules. |
| -verbose:jni,sizes,stack,threads | Prints information about native method calls, the sizes of the JVM, the stack and threads. |
| -verbose:jni,sizes,stack,module | Prints information about native method calls, the sizes of the JVM, the stack and modules. |
| -verbose:jni,sizes,threads,module | Prints information about native method calls, the sizes of the JVM, threads and modules. |
| -verbose:jni,stack,threads,module | Prints information about native method calls, the stack, threads and modules. |
| -verbose:sizes,stack,threads,module | Prints information about the sizes of the JVM, the stack, threads and modules. |
| -verbose:gc,class,jni,sizes,stack | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM and the stack. |
| -verbose:gc,class,jni,sizes,threads | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM and threads. |
| -verbose:gc,class,jni,sizes,module | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM and modules. |
| -verbose:gc,class,jni,stack,threads | Prints information about garbage collection, classes loaded, native method calls, the stack and threads. |
| -verbose:gc,class,jni,stack,module | Prints information about garbage collection, classes loaded, native method calls, the stack and modules. |
| -verbose:gc,class,jni,threads,module | Prints information about garbage collection, classes loaded, native method calls, threads and modules. |
| -verbose:gc,class,sizes,stack,threads | Prints information about garbage collection, classes loaded, the sizes of the JVM, the stack and threads. |
| -verbose:gc,class,sizes,stack,module | Prints information about garbage collection, classes loaded, the sizes of the JVM, the stack and modules. |
| -verbose:gc,class,sizes,threads,module | Prints information about garbage collection, classes loaded, the sizes of the JVM, threads and modules. |
| -verbose:gc,class,stack,threads,module | Prints information about garbage collection, classes loaded, the stack, threads and modules. |
| -verbose:gc,jni,sizes,stack,threads | Prints information about garbage collection, native method calls, the sizes of the JVM, the stack and threads. |
| -verbose:gc,jni,sizes,stack,module | Prints information about garbage collection, native method calls, the sizes of the JVM, the stack and modules. |
| -verbose:gc,jni,sizes,threads,module | Prints information about garbage collection, native method calls, the sizes of the JVM, threads and modules. |
| -verbose:gc,jni,stack,threads,module | Prints information about garbage collection, native method calls, the stack, threads and modules. |
| -verbose:gc,sizes,stack,threads,module | Prints information about garbage collection, the sizes of the JVM, the stack, threads and modules. |
| -verbose:class,jni,sizes,stack,threads | Prints information about classes loaded, native method calls, the sizes of the JVM, the stack and threads. |
| -verbose:class,jni,sizes,stack,module | Prints information about classes loaded, native method calls, the sizes of the JVM, the stack and modules. |
| -verbose:class,jni,sizes,threads,module | Prints information about classes loaded, native method calls, the sizes of the JVM, threads and modules. |
| -verbose:class,jni,stack,threads,module | Prints information about classes loaded, native method calls, the stack, threads and modules. |
| -verbose:class,sizes,stack,threads,module | Prints information about classes loaded, the sizes of the JVM, the stack, threads and modules. |
| -verbose:jni,sizes,stack,threads,module | Prints information about native method calls, the sizes of the JVM, the stack, threads and modules. |
| -verbose:gc,class,jni,sizes,stack,threads | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM, the stack and threads. |
| -verbose:gc,class,jni,sizes,stack,module | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM, the stack and modules. |
| -verbose:gc,class,jni,sizes,threads,module | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM, threads and modules. |
| -verbose:gc,class,jni,stack,threads,module | Prints information about garbage collection, classes loaded, native method calls, the stack, threads and modules. |
| -verbose:gc,class,sizes,stack,threads,module | Prints information about garbage collection, classes loaded, the sizes of the JVM, the stack, threads and modules. |
| -verbose:gc,jni,sizes,stack,threads,module | Prints information about garbage collection, native method calls, the sizes of the JVM, the stack, threads and modules. |
| -verbose:class,jni,sizes,stack,threads,module | Prints information about classes loaded, native method calls, the sizes of the JVM, the stack, threads and modules. |
| -verbose:gc,class,jni,sizes,stack,threads,module | Prints information about garbage collection, classes loaded, native method calls, the sizes of the JVM, the stack, threads and modules. |
| -Xmx | Sets the maximum heap size. |
| -Xms | Sets the initial heap size. |
| -Xmx -Xms | Sets the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size and the maximum heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx | Sets the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size, the initial heap size, the maximum heap size and the initial heap size. |
| -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms -Xmx -Xms

This is a master sheet with most of the JVM arguments:



You can be inherently biased

Or your incentives can be screwed

## JDK 23

This release is the Reference Implementation of version 23 of the Java SE Platform, as specified by JSR 398 in the Java Community Process.

JDK 23 reached General Availability on 17 September 2024. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

### Features

455: Primitive Types in Patterns, instanceof, and switch (Preview)
466: Class-File API (Second Preview)
467: Markdown Documentation Comments
469: Vector API (Eighth Incubator)
473: Stream Gatherers (Second Preview)
471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal
474: ZGC: Generational Mode by Default
476: Module Import Declarations (Preview)
477: Implicitly Declared Classes and Instance Main Methods (Third Preview)
480: Structured Concurrency (Third Preview)
481: Scoped Values (Third Preview)
482: Flexible Constructor Bodies (Second Preview)

## JDK 24

This release is the Reference Implementation of version 24 of the Java SE Platform, as specified by JSR 399 in the Java Community Process.

JDK 24 reached General Availability on 18 March 2025. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

### Features

404: Generational Shenandoah (Experimental)
450: Compact Object Headers (Experimental)
472: Prepare to Restrict the Use of JNI
475: Late Barrier Expansion for G1
478: Key Derivation Function API (Preview)
479: Remove the Windows 32-bit x86 Port
483: Ahead-of-Time Class Loading & Linking
484: Class-File API
485: Stream Gatherers
486: Permanently Disable the Security Manager
487: Scoped Values (Fourth Preview)
488: Primitive Types in Patterns, instanceof, and switch (Second Preview)
489: Vector API (Ninth Incubator)
490: ZGC: Remove the Non-Generational Mode
491: Synchronize Virtual Threads without Pinning
492: Flexible Constructor Bodies (Third Preview)
493: Linking Run-Time Images without JMODs
494: Module Import Declarations (Second Preview)
495: Simple Source Files and Instance Main Methods (Fourth Preview)
496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism
497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm
498: Warn upon Use of Memory-Access Methods in sun.misc.Unsafe
499: Structured Concurrency (Fourth Preview)
501: Deprecate the 32-bit x86 Port for Removal

## JDK 25

This release will be the Reference Implementation of version 25 of the Java SE Platform, as specified by JSR 400 in the Java Community Process.

### Schedule

| | |
|---|---|
| 2025/06/05 | Rampdown Phase One (branch from main line) |
| 2025/07/17 | Rampdown Phase Two |
| 2025/08/07 | Initial Release Candidate |
| 2025/08/21 | Final Release Candidate |
| 2025/09/16 | General Availability |

### Status

JDK 25 is in Rampdown Phase One. The overall feature set is frozen. No further JEPs will be targeted to this release.

The stabilization branch, jdk25, is open for select bug fixes and, with approval, late enhancements per the JDK Release Process (JEP 3). Integrate most stabilization changes via backports from the main line.

- RDP 1 candidate bugs
- Bug-Deferral Process
- Late-Enhancement Request Process

### Features

470: PEM Encodings of Cryptographic Objects (Preview)
502: Stable Values (Preview)
503: Remove the 32-bit x86 Port
505: Structured Concurrency (Fifth Preview)
506: Scoped Values
507: Primitive Types in Patterns, instanceof, and switch (Third Preview)
508: Vector API (Tenth Incubator)
509: JFR CPU-Time Profiling (Experimental)
510: Key Derivation Function API
511: Module Import Declarations
512: Compact Source Files and Instance Main Methods
513: Flexible Constructor Bodies
514: Ahead-of-Time Command-Line Ergonomics
515: Ahead-of-Time Method Profiling
518: JFR Cooperative Sampling
519: Compact Object Headers
520: JFR Method Timing & Tracing
521: Generational Shenandoah

The most interesting things coming in JDK is performance

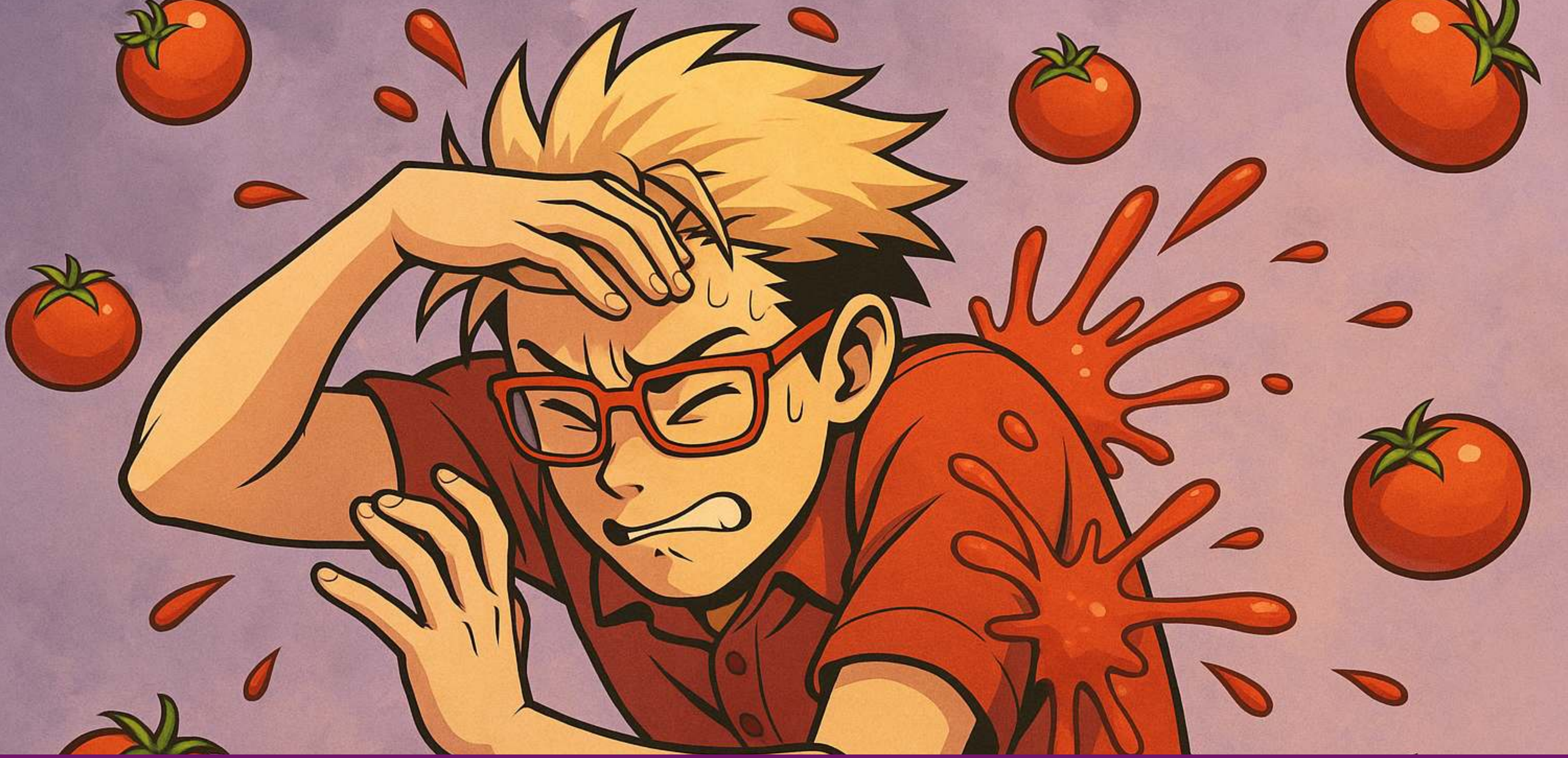
Startup Time

Memory Usage

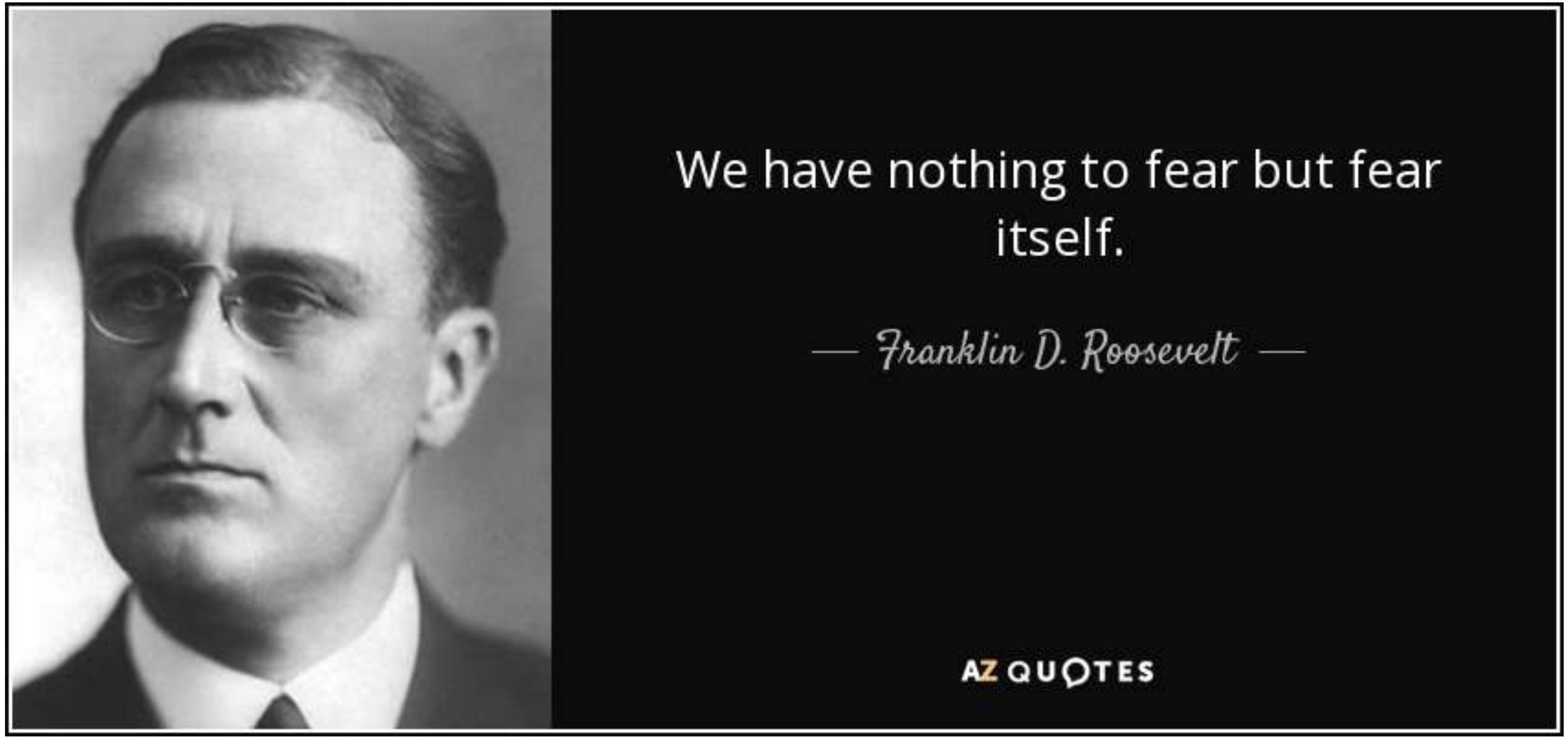
Throughput

CPU Consumption

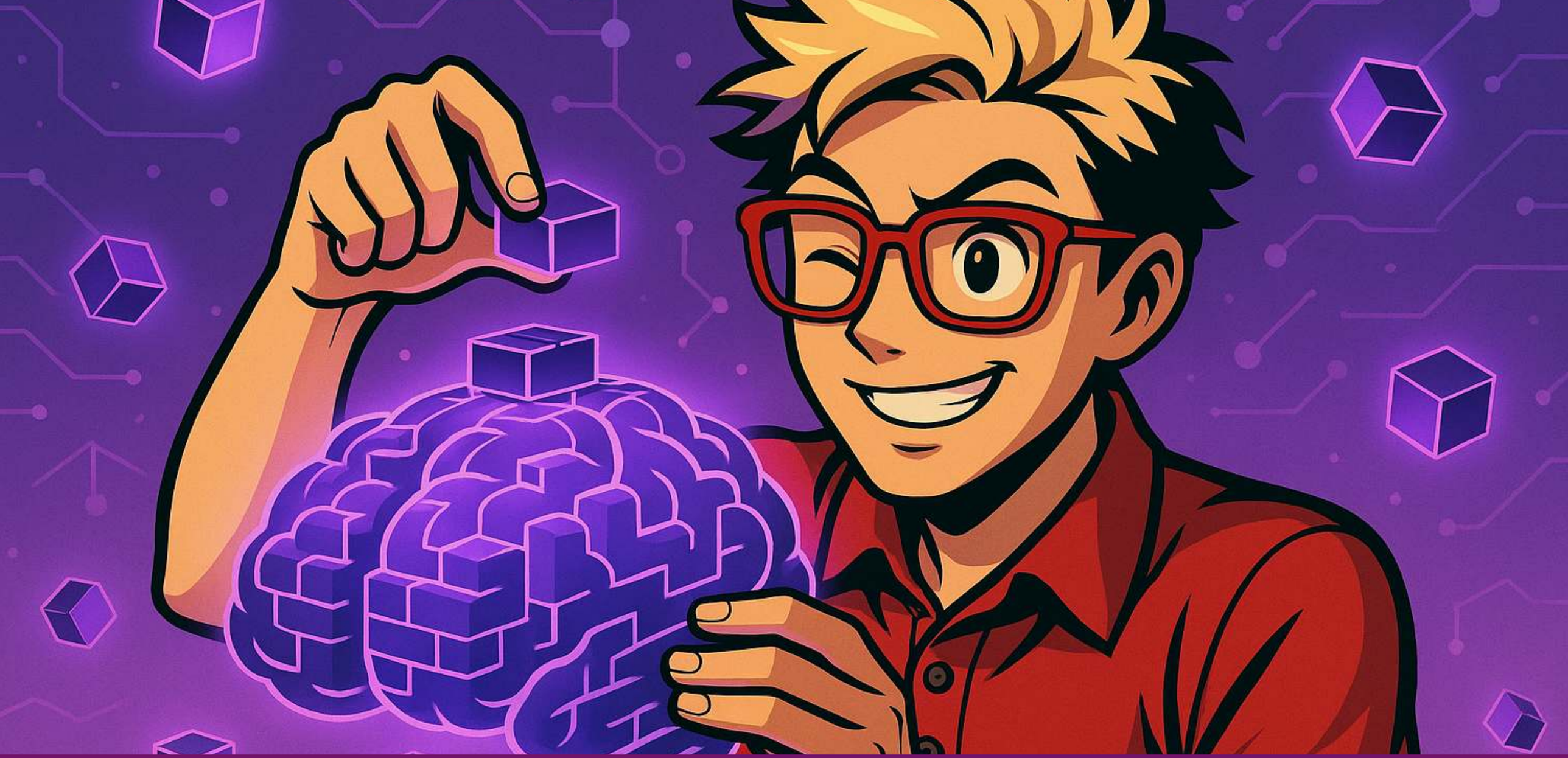
There are interesting metrics to be achieved



I’m a bit scared there will be some critics




If you are scared, you already lost 😉



I want you to help build mental model what is achievable

155 Slides / 50 Minutes

# I want to beat the record for WPM and SPS

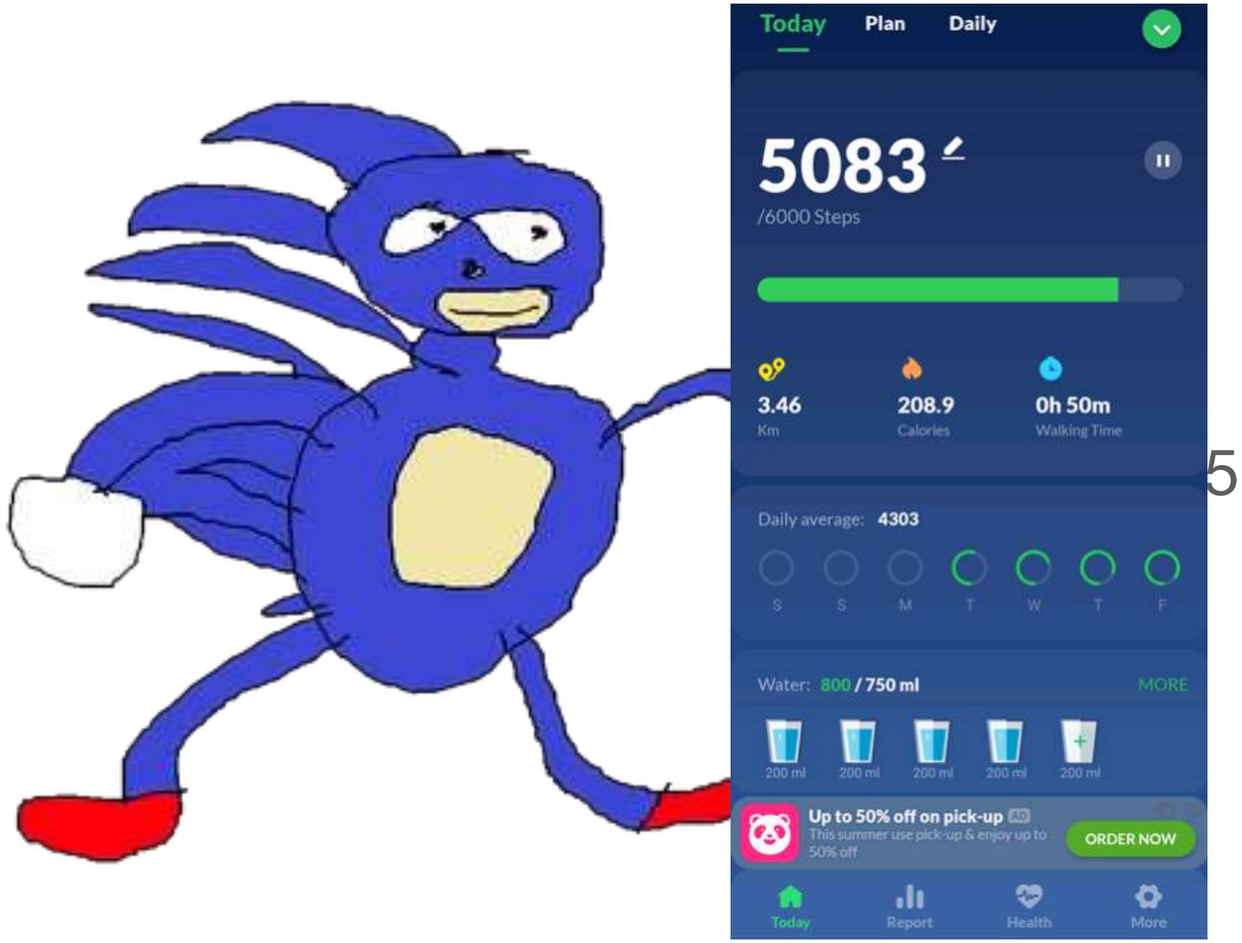
155 Slides / 50 Minutes

# I want to beat the record for WPM and SBPPS

155 Slides / 50 Minutes

Words-per-Minute and Steps-by-presenter-per-Slide

# I want to beat the record for WPM and SBPPS




5 Slides / 50 Minutes

Words-per-Minute and Steps-by-presenter-per-Slide

We still scratch the surface

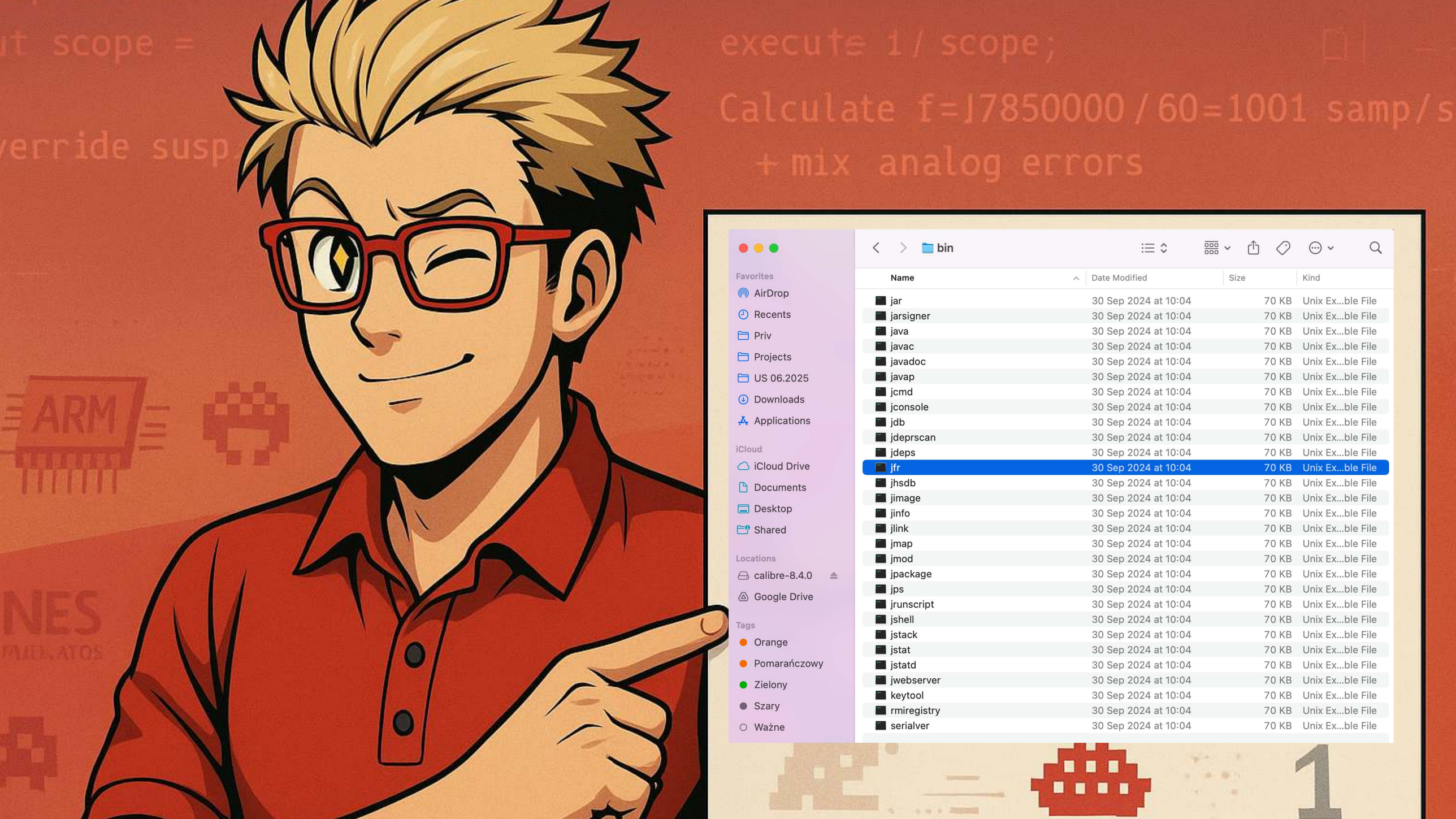
Tooling

JDK version impact

Features improving Cold Start, Throughput and memory

A bit about hardware

# Tooling


execute 1 / scope;
Calculate f=17850000 / 60=1001 samp/s
+ mix analog errors
ARM
NES
bin
Favorites
AirDrop
Recents
Priv
Projects
US 06.2025
Downloads
Applications
iCloud
iCloud Drive
Documents
Desktop
Shared
Locations
calibre-8.4.0
Google Drive
Tags
Orange
Pomarańczowy
Zielony
Szary
Ważne
Name
Date Modified
Size
Kind
jar 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jarsigner 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
java 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
javac 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
javadoc 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
javap 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jcmd 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jconsole 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jdb 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jdeprscan 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jdeps 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jfr 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jhsdb 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jimage 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jinfo 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jlink 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jmap 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jmod 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jpackage 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jps 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jrunscript 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jshell 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jstack 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jstat 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jstatd 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
jwebserver 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
keytool 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
rmiregistry 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File
serialver 30 Sep 2024 at 10:04 70 KB Unix Ex...ble File

**Java Flight Recorder (JFR)** is a built-in, low-overhead profiling and event-collection framework that continuously captures detailed runtime data from the JVM and your application - including CPU hotspots, memory allocations, thread states, garbage-collection pauses, and I/O activity.

## JFR Explorer

**Very early preview version! Idea for this tool by Johannes Bechberger. JFRExplorer on GitHub**

JDK11 JDK12 JDK13 JDK14 JDK15 JDK16 JDK17 JDK18 JDK19 JDK20

Search JFR Events:

| Name | Category | Label | Attributes | Fields |
|---|---|---|---|---|
| AllocationRequiringGC | JVM -> GC -> Detailed | Allocation Requiring GC | | |
| BooleanFlag | JVM -> Flag | Boolean Flag | | |
| BooleanFlagChanged | JVM -> Flag | Boolean Flag Changed | | |
| ClassDefine | JVM -> Class Loading | Class Define | | |

**AllocationRequiringGC — Attributes**

| Name | Value |
|---|---|
| thread | true |
| stackTrace | true |
| startTime | false |

**AllocationRequiringGC — Fields**

| Name | Type | ContentType | Label |
|---|---|---|---|
| gcId | uint | | Pending GC Identifier |
| size | ulong | bytes | Allocation Size |

**BooleanFlag — Attributes**

| Name | Value |
|---|---|
| period | endChunk |

**BooleanFlag — Fields**

| Name | Type | Label |
|---|---|---|
| name | string | Name |
| value | boolean | Value |
| origin | FlagValueOrigin | Origin |

**BooleanFlagChanged — Attributes**

| Name | Value |
|---|---|
| startTime | false |

**BooleanFlagChanged — Fields**

| Name | Type | Label |
|---|---|---|
| name | string | Name |
| oldValue | boolean | Old Value |
| newValue | boolean | New Value |
| origin | FlagValueOrigin | Origin |

**ClassDefine — Attributes**

| Name | Value |
|---|---|
| thread | true |
| stackTrace | true |
| startTime | false |

**ClassDefine — Fields**

| Name | Type | Label |
|---|---|---|
| definedClass | Class | Defined Class |
| definingClassLoader | ClassLoader | Defining Class Loader |

# Java Flight Recorder

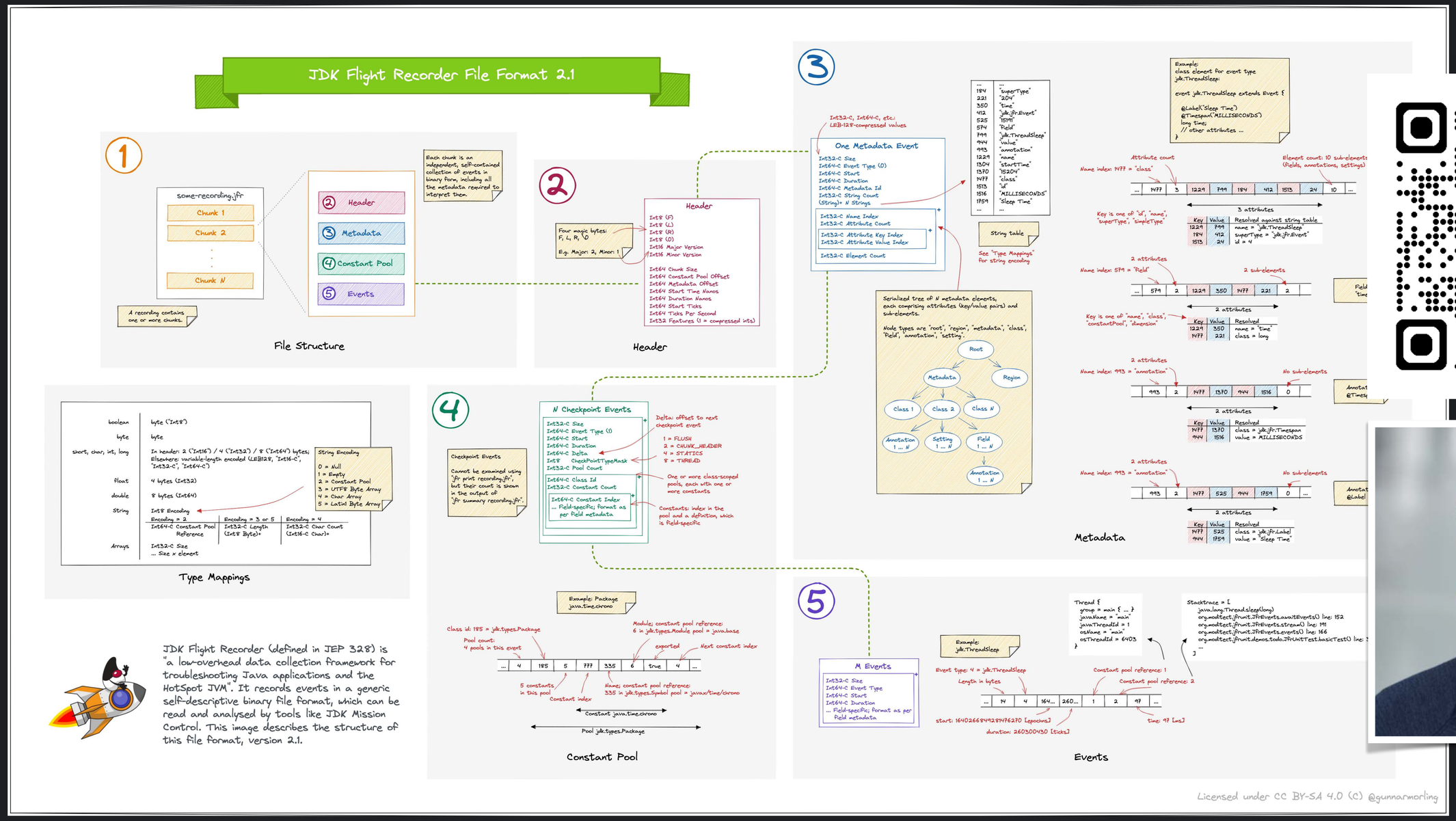
# Java Flight Recorder Events

Starting with OpenJDK-11 Java Flight Recorder Events are available to monitor your running JVM. A matrix of all events available in different OpenJDK Versions is available here.

Version Overview

| VERSION | EVENTS | TYPES |
|---|---|---|
| OpenJDK 11 | 136 | 52 |
| OpenJDK 12 | 136 | 52 |
| OpenJDK 13 | 143 | 53 |
| OpenJDK 14 | 145 | 58 |
| OpenJDK 15 | 157 | 59 |
| OpenJDK 16 | 159 | 59 |
| OpenJDK 17 | 167 | 59 |
| OpenJDK 18 | 165 | 59 |

© BestSolution.at



# Java Flight Recorder File Format

Introduction
Flight Recorder
JVM
JVM: Class Loading
JVM: Code Cache
JVM: Compiler
JVM: Diagnostics
JVM: Flag
JVM: GC: Collector
JVM: GC: Configuration
JVM: GC: Detailed
ZThreadDebug
G1MMU
EvacuationInformation
ObjectCountAfterGC
G1EvacuationYoungStatistics
G1EvacuationOldStatistics
G1BasicIHOP
G1AdaptiveIHOP
PromoteObjectInNewPLAB
PromoteObjectOutsidePLAB
PromotionFailed
EvacuationFailed
ConcurrentModeFailure
GCCPUTime
AllocationRequiringGC
TenuringDistribution
G1HeapRegionTypeChange
ObjectCount
G1HeapRegionInformation
ZAllocationStall
ZPageAllocation
ZRelocationSet
ZRelocationSetGroup
ZStatisticsCounter
ZStatisticsSampler
ZThreadPhase
ZUncommit
ShenandoahHeapRegionStateChange
ShenandoahHeapRegionInformation
ShenandoahEvacuationInformation
JVM: GC: Heap
JVM: GC: Metaspace
JVM: GC: Phases
JVM: GC: Reference
JVM: Heap

# GCCPUTime

default profiling startTime 17 21 25 26

Category: Java Virtual Machine / GC / Detailed

Appearing in: G1GC, ParallelGC, SerialGC
Missing in: ShenandoahGC, ZGC

GC CPU Time information. Supported: G1GC, ParallelGC and SerialGC

▶ Code Context

| Configuration | enabled |
|---|---|
| **default** | true |
| **profiling** | true |

| Field | Type | Description |
|---|---|---|
| **gcId** | uint | GC Identifier |
| **userTime** | ulong: nanos | User Time |
| **systemTime** | ulong: nanos | System Time |
| **realTime** | ulong: nanos | Real Time |

▶ Examples 3

# AllocationRequiringGC

startTime eventThread stackTrace 11 17 21 25 26 graal vm

Category: Java Virtual Machine / GC / Detailed

https://sap.github.io/SapMachine/jfrevents/25.html

OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC
Mastodon
Bluesky

Bylaws · Census
Legal

**Workshop**

**JEP Process**

**Source code**
GitHub
Mercurial

**Tools**
Git
jtreg harness

**Groups**
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters

# JEP 509: JFR CPU-Time Profiling (Experimental)

| | |
|---|---|
| *Authors* | Jaroslav Bachorík, Johannes Bechberger, & Ron Pressler |
| *Owner* | Johannes Bechberger |
| *Type* | Feature |
| *Scope* | JDK |
| *Status* | Completed |
| *Release* | 25 |
| *Component* | hotspot / jfr |
| *Discussion* | hotspot dash jfr dash dev at openjdk dot org |
| *Effort* | M |
| *Duration* | S |
| *Reviewed by* | Markus Grönlund |
| *Endorsed by* | Vladimir Kozlov |
| *Created* | 2024/08/04 10:34 |
| *Updated* | 2025/06/10 08:20 |
| *Issue* | 8337789 |

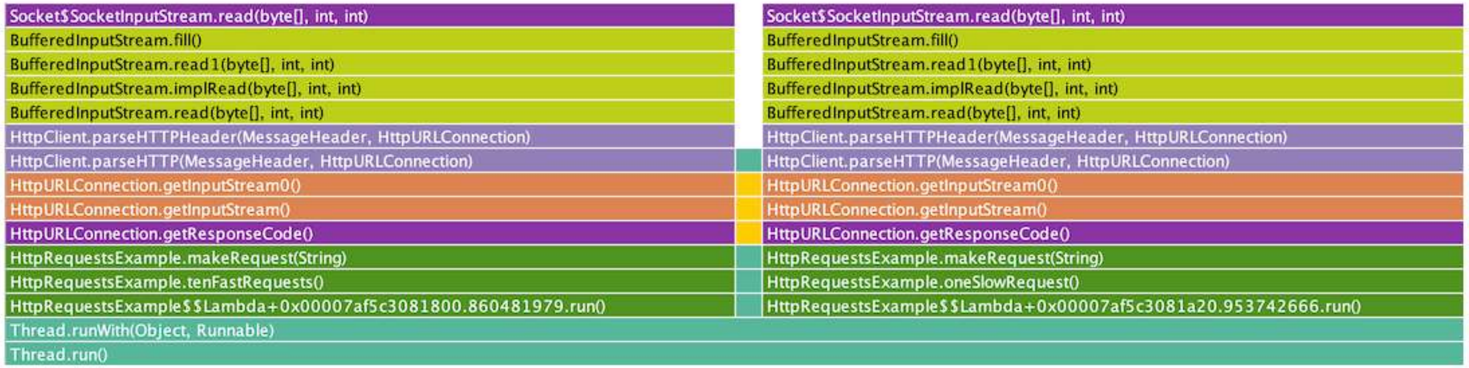
## Summary

Enhance the JDK Flight Recorder (JFR) to capture CPU-time profiling information on Linux. This is an experimental feature.

## Motivation

A running program consumes computational resources such a:
cycles, and elapsed time. To *profile* a program is to measure th
such resources by specific elements of the program. A profile r
e.g., one method consumes 20% of a resource, while another r
only 0.1%.

We can generate a graphical profile, in the form of a flame graph, by using the JDK Mission Control tool (JMC):




# JEP 509: JFR CPU-Time Profiling (Experimental)

OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC
Mastodon
Bluesky

Bylaws · Census
Legal

**Workshop**

**JEP Process**

**Source code**
GitHub
Mercurial

**Tools**
Git
jtreg harness

**Groups**
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters

## JEP 509: JFR CPU-Time Profiling (Experimental)

| | |
|---|---|
| *Authors* | Jaroslav Bachorík, Johannes Bechberger, & Ron Pressler |
| *Owner* | Johannes Bechberger |
| *Type* | Feature |
| *Scope* | JDK |
| *Status* | Completed |
| *Release* | 25 |
| *Component* | hotspot / jfr |
| *Discussion* | hotspot dash jfr dash dev at openjdk dot org |
| *Effort* | M |
| *Duration* | S |
| *Reviewed by* | Markus Grönlund |
| *Endorsed by* | Vladimir Kozlov |
| *Created* | 2024/08/04 10:34 |
| *Updated* | 2025/06/10 08:20 |
| *Issue* | 8337789 |

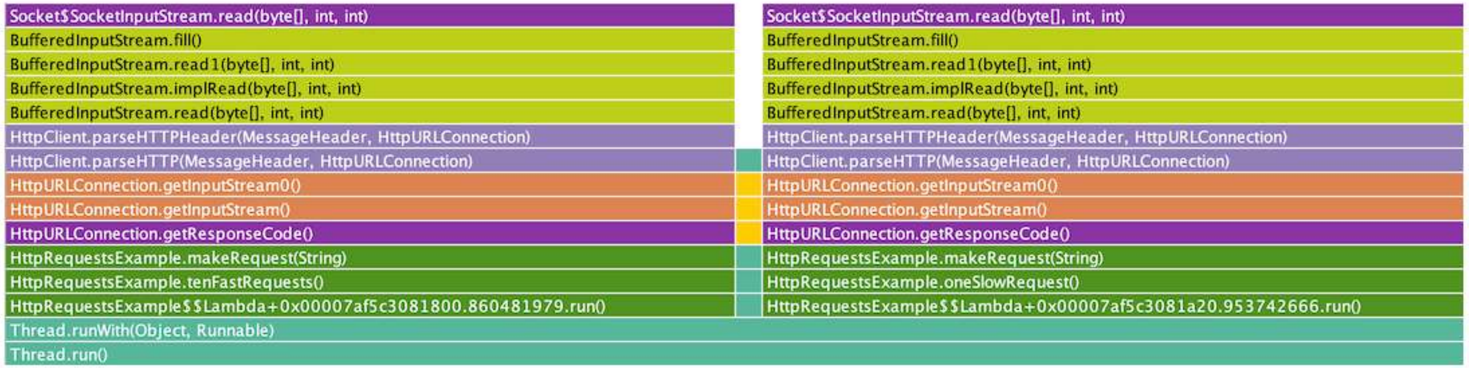
### Summary

Enhance the JDK Flight Recorder (JFR) to capture CPU-time profiling informatio
Linux. This is an experimental feature.

### Motivation

A running program consumes computational resources such a
cycles, and elapsed time. To *profile* a program is to measure th
such resources by specific elements of the program. A profile r
e.g., one method consumes 20% of a resource, while another r
only 0.1%.

We can gene te a graphical profile, in the form of a flame graph, by using the JDK Mission Control tool (JMC):



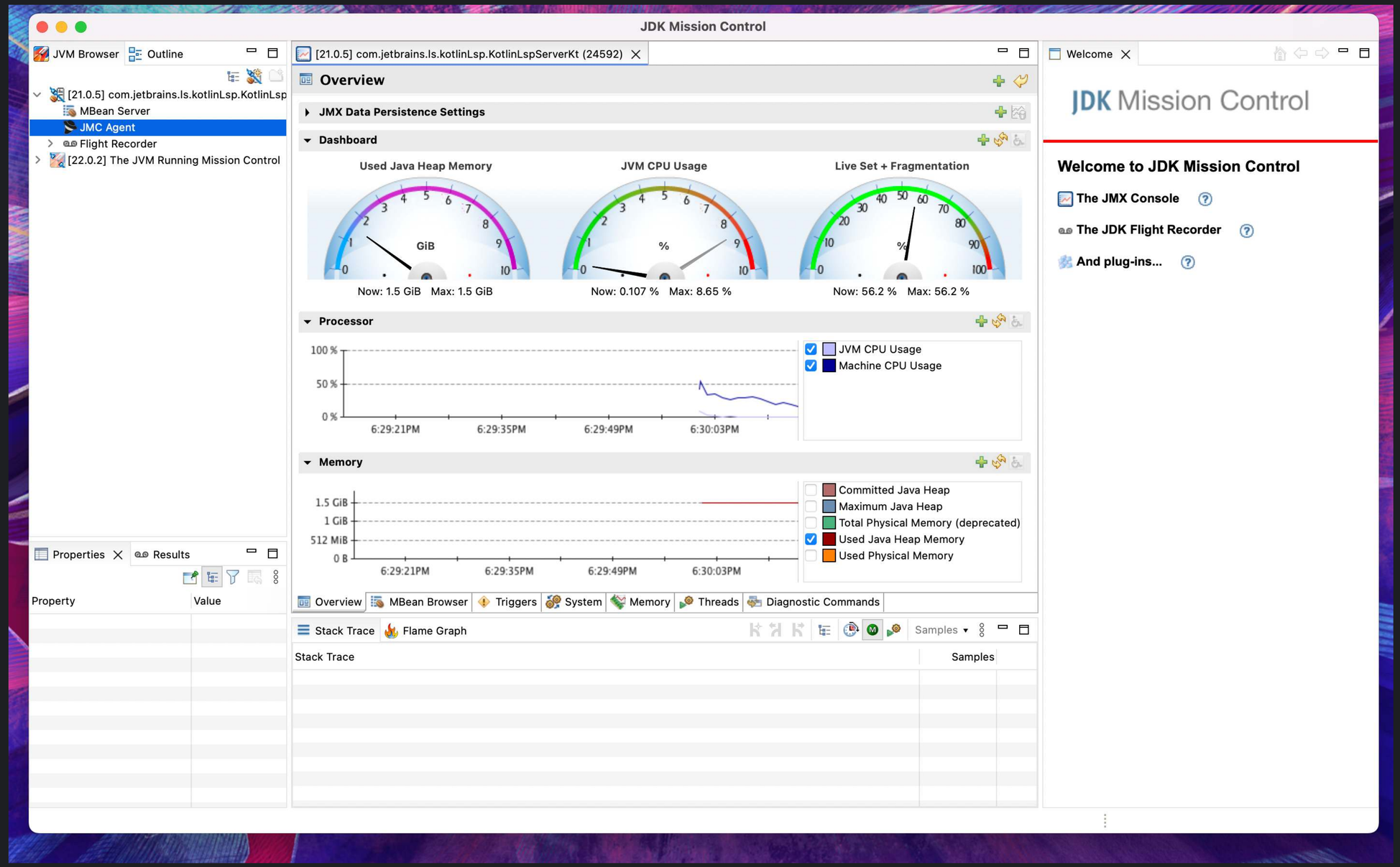

# JEP 509: JFR CPU-Time Profiling (Experimental)

# Java Mission Control

ORACLE

The resulting flight-record files can be inspected in tools such as **JDK Mission Control** for live or post-mortem analysis, enabling engineers to diagnose performance bottlenecks and operational anomalies with negligible impact on production workloads.

# Java Mission Control



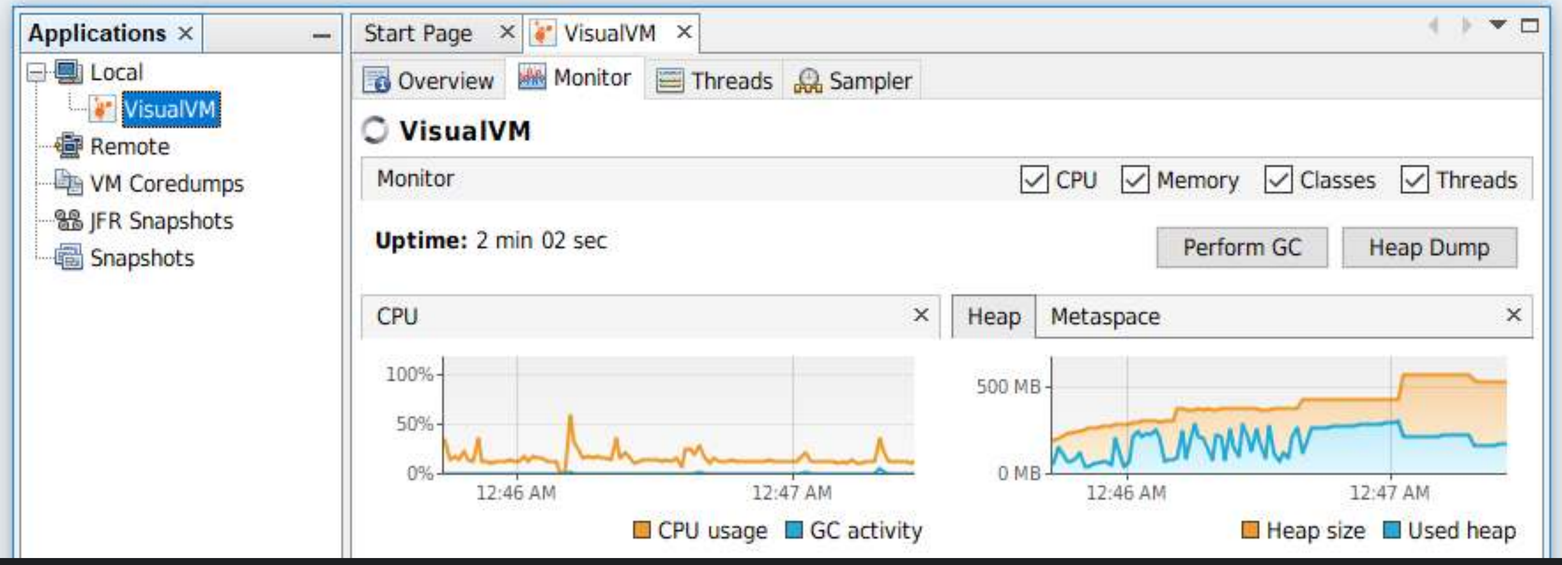

# Java Mission Control

# VisualVM

**All-in-One Java Troubleshooting Tool**

VisualVM is a visual tool integrating commandline JDK tools and lightweight profiling capabilities.
Designed for both development and production time use.

**Download VisualVM 2.2**



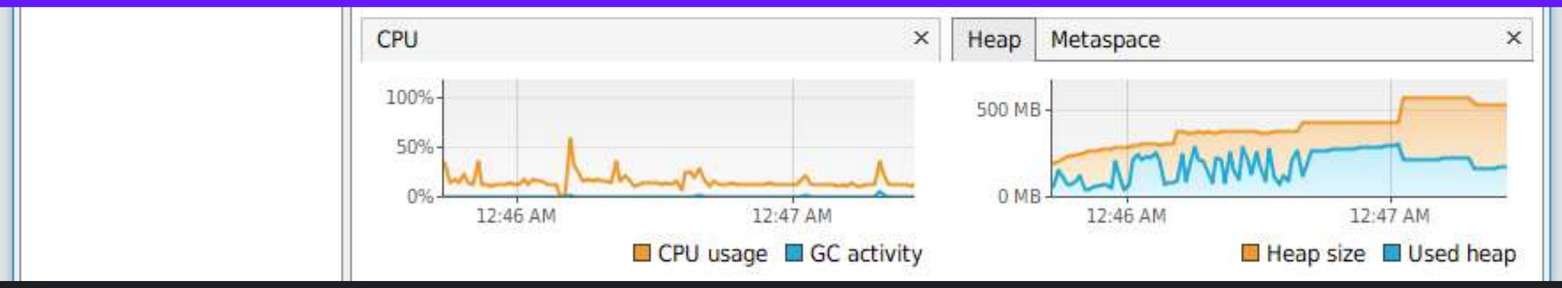

VisualVM

VisualVM

All-in-One Java Troubleshooting Tool

VisualVM is a visual tool integrating commandline JDK tools and lightweight profiling capabilities.

VisualVM is a visual tool integrating commandline JDK tools and lightweight profiling capabilities. Designed for both development and production time use.




# VisualVM

# Async-profiler

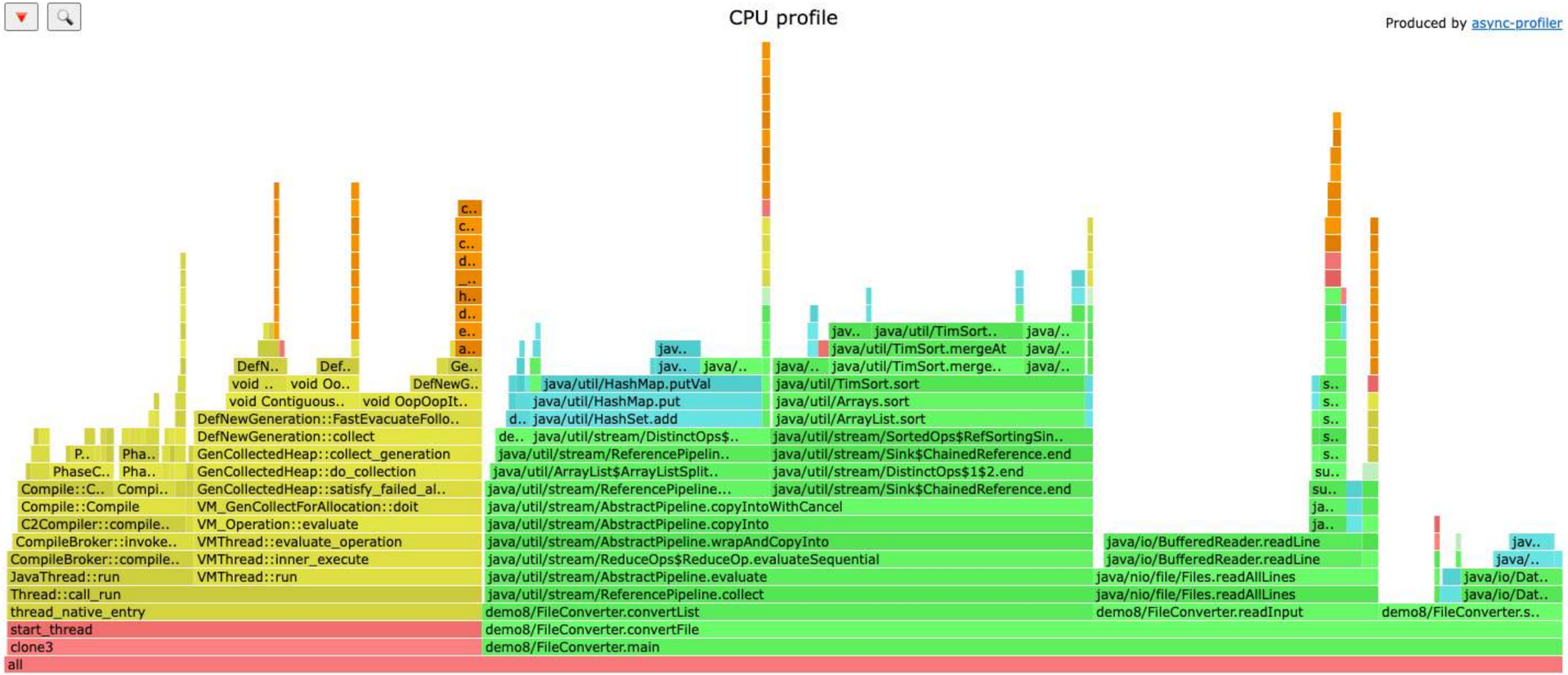
This project is a low overhead sampling profiler for Java that does not suffer from the Safepoint bias problem. It features HotSpot-specific API to collect stack traces and to track memory allocations. The profiler works with

High-resolution sampling profiler for CPU, allocations, locks, and native-Java mixed stacks.

... with native support for flame graphs

- Contended locks
- Hardware and software performance counters like cache misses, page faults, context switches
- and more.

Async-Profiler




# Async-profiler

# Java Microbenchmark Harness (JMH)

JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM.

## Usage

JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM.

build configuration. We strongly recommend new users make use of the archetype to setup the correct environment.

### Samples

In order to understand JMH tests and maybe write your own, it might be useful to work through the JMH Samples. See the test comments for run instructions.

Java Microbenchmark Harness

```java
import org.openjdk.jmh.annotations.*;

import java.util.concurrent.TimeUnit;

@BenchmarkMode(Mode.Throughput) // Measures how many ops/sec
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@State(Scope.Thread) // Each thread gets its own state instance
public class StringBenchmark {

    private String base = "Hello";
    private String suffix = "World";

    @Benchmark
    public String usePlusOperator() {
        return base + suffix;
    }

    @Benchmark
    public String useStringBuilder() {
        return new StringBuilder().append(base).append(suffix).toString();
    }
}
```

# JMH

## 🧪 JMH (Java Microbenchmark Harness)

- **Use when:** You need **accurate** benchmarking of small code snippets or algorithm comparisons.
- Designed for microbenchmarks, not whole-application profiling.
- Helps eliminate JIT warm-up noise, dead-code elimination, and benchmarking pitfalls.
- Use **during development** or **algorithm design**, **not on production apps**.

## 🔍 Java Flight Recorder (JFR)

- **Use when:** you want a safe, all-around profiler running in production, with minimal impact and rich event data.
- Lightweight event recorder for the JVM - gives a broad overview of system behavior over time.
- Sampling based — lower granularity for method profiling and no native stack support (can't see JNI/native frames).
- Works best when paired with Java Mission Control for visualization.

## 📊 Java Mission Control (JMC)

- **Use when:** You want a graphical UI to analyze JFR recordings.

  Helpful for post-mortem analysis or diving deep into performance bottlenecks over time.

  Best suited for profiling in production or staging environments using JFR data.

  Use it to correlate application metrics and JVM internals like **class loading** or **memory pressure**.

# Heuristics to use

## 🔥 async-profiler

- **Use when:** You want high-accuracy, low-overhead profiling with support for native + Java stacks.
- It capturesative and Java CPU stacks, allocations, lock contention, cache misses (perf events).
- Use async-profiler when you want to zoom in on a hot method, native bottleneck, or lock contention - especially for short bursts of profiling in staging or prod.

**JFR** is like a black box recorder in an airplane: records everything over time, reliably, safely.

**async-profiler** is like a heat camera: zooms in with sharp precision, but doesn't show you the full cabin.

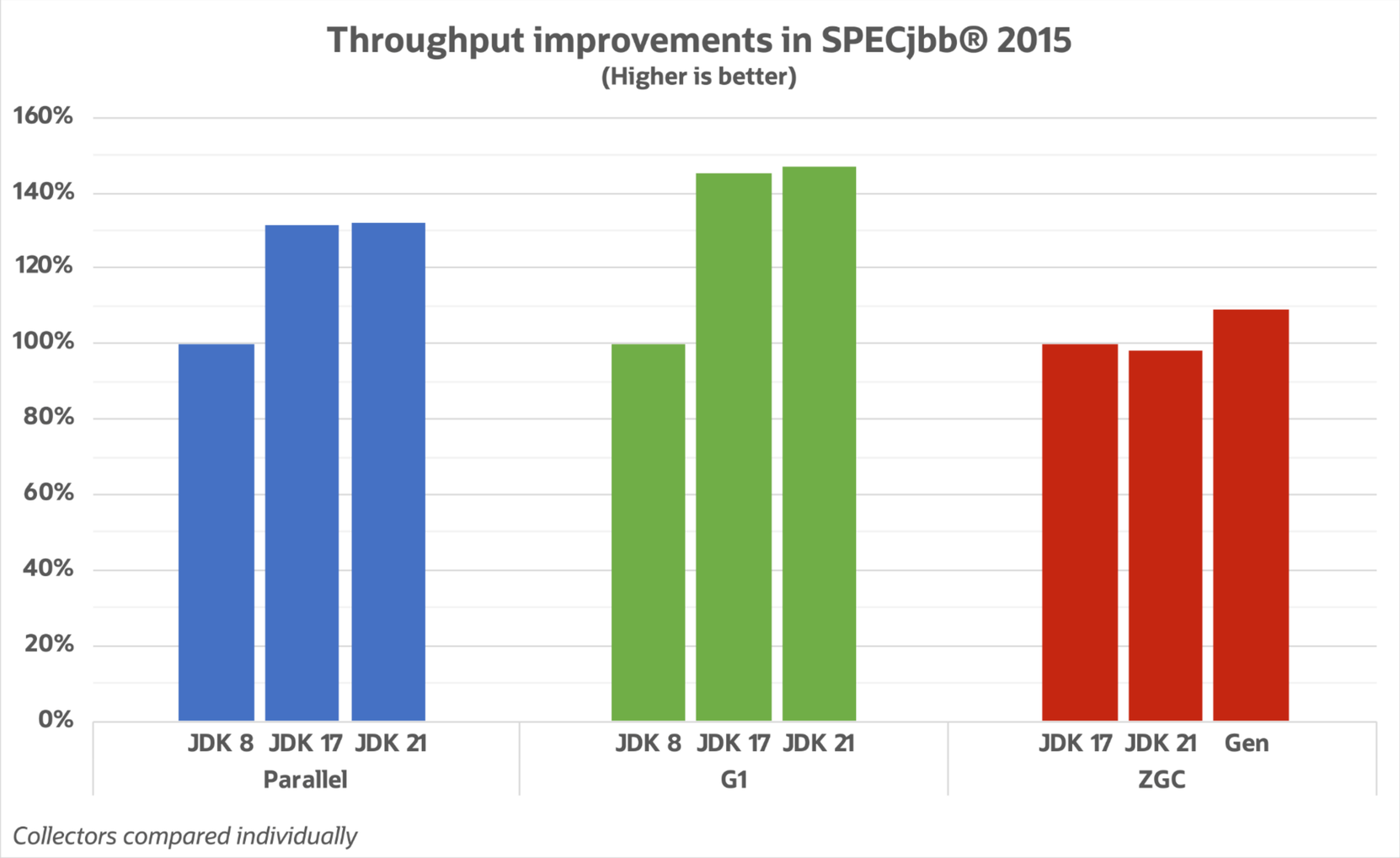
## 🧠 VisualVM

- **Use when:** You need a quick, all-in-one local profiler/debugger for a running JVM.
- Useful for on-the-fly heap dumps, thread inspection, CPU sampling, and basic GC analysis.
- Good for development and testing, but **not recommended** for production due to higher overhead.

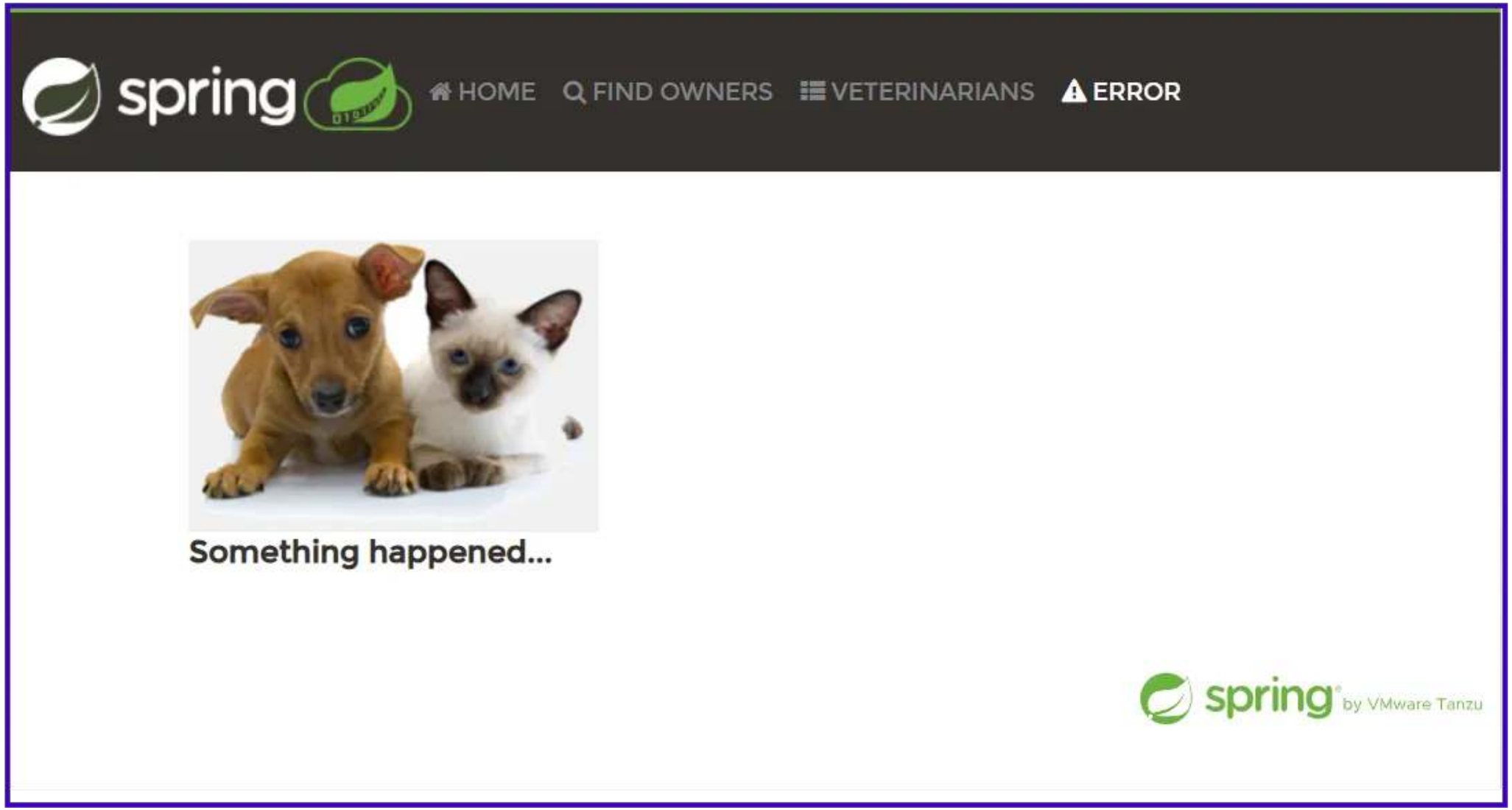
Which JDK to pick… ? It’s easy and not easy answer

| JDK | Feature | Typical effect |
|---|---|---|
| 9 | **Compact Strings** (JEP 254) store Latin-1 text as 1 byte/char | –40-50 % heap for string-heavy apps |
| 9 | **Indified String concat** (JEP 280) | Up to 2.9 × faster string building and far less garbage in tight loops |
| 11 | Improved Memory Management in Containers | Since Java 11, the JVM became container-aware (JEP 238, etc.), and further refinements have been made. By Java 17, the JVM more accurately sizes the heap and GC regions based on cgroup. |
| 16 | Elastic Metaspace (JEP 387) | Returns class metadata; avoids 100 %+ RSS spikes under heavy class-loading |

There is a lot of small features that bring




# Garbage Collectors: +30% from JDK 8 to JDK 21




Spring PetClinic

ArturSkowronski / jdk-performance-benchmarks-iceberg

Type / to search

Code | Issues | Pull requests 2 | Actions | Projects | Wiki | Security | Insights | Settings

jdk-performance-benchmarks-iceberg Public

Pin | Watch 0 | Fork 0 | Star 0

main | 4 Branches | 0 Tags | Go to file t | Add file | Code

ArturSkowronski Merge pull request #2 from ArturSkowronski/codex/create-ben... 58030cc · 19 minutes ago 3 Commits

| | | |
|---|---|---|
| scripts | Add script to run PetClinic with configurable JDK | 19 minutes ago |
| .gitignore | Initial commit | 30 minutes ago |
| README.md | Add script to run PetClinic with configurable JDK | 19 minutes ago |

README

# JDK Performance Benchmarks - Iceberg

This repository contains a helper script for running the Spring PetClinic application in Docker with different JDK versions. It can be used to measure the impact of JDK upgrades on the application.

## Prerequisites

- Docker installed on the host machine.
- Network access to download Docker images and Maven dependencies.

About

*No description, website, or topics provided.*

Readme
Activity
0 stars
0 watching

ArturSkowronski/jdk-performance-benchmarks-iceberg

**Some checks haven't completed yet**
5 in progress, 2 queued checks

2 pending checks

- PetClinic Build Matrix / build (graalvm) (pull_request) *Queued — Waiting to run this check...*
- PetClinic JMeter Tests / jmeter (pull_request) *Queued — Waiting to run this check...*

5 in progress checks

- PetClinic Build Matrix / build (8) (pull_request) *Started now — This check has started...*
- PetClinic Build Matrix / build (11) (pull_request) *Started now — This check has started...*
- PetClinic Build Matrix / build (23) (pull_request) *Started now — This check has started...*
- PetClinic Build Matrix / build (crac) (pull_request) *Started now — This check has started...*

**No conflicts with base branch**
Merging can be performed automatically.

Merge pull request

You can also merge this with the command line. View command line instructions.

**Add a comment**

ArturSkowronski/jdk-performance-benchmarks-iceberg

ArturSkowronski / jdk-performance-benchmarks-iceberg

Type / to search

Code | Issues | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

← Back to pull request #5

## Add VisualVM and JMC support #7

Cancel workflow

Summary

Jobs

jmeter

Run details

Usage

Workflow file

### jmeter

Started 1m 45s ago

Search logs

- Set up job 0s
- Run actions/checkout@v4 1s
- Start PetClinic container 7s
- Wait for PetClinic 1m 36s

```
1 ▶ Run for i in {1..30}; do
```

- Run JMeter test
- Stop PetClinic
- Upload JMeter Results
- Post Run actions/checkout@v4

# ArturSkowronski/jdk-performance-benchmarks-iceberg

```
Digest: sha256:088ac52b759a198a5afa5ae13d0a6306e9f2017d71ad140ff57427f6930406f7
Status: Downloaded newer image for justb4/jmeter:5.5
START Running Jmeter on Fri Jun  6 18:43:48 CEST 2025
JVM_ARGS=-Xmn2988m -Xms11952m -Xmx11952m
jmeter args=-n -t /tests/petclinic.jmx -l /tests/results.jtl
jmeter ALL ARGS=-Dlog4j2.formatMsgNoLookups=true -n -t /tests/petclinic.jmx -l /tests/results.jtl
Jun 06, 2025 6:43:49 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Creating summariser <summary>
Created the tree successfully using /tests/petclinic.jmx
Starting standalone test @ June 6, 2025 6:43:50 PM CEST (1749228230229)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary =      1003 in 00:05:23 =    6.1/s Avg:     Min:    21 Max:    21 Err:     5 (0.05%)
Tidying up ...    @ June 6, 2025 6:48:32 PM CEST (1749228230702)
... end of run
END Running Jmeter on Fri Jun  6 18:43:51 CEST 2025
```

PetClinic_JMeter_Results - JDK 11

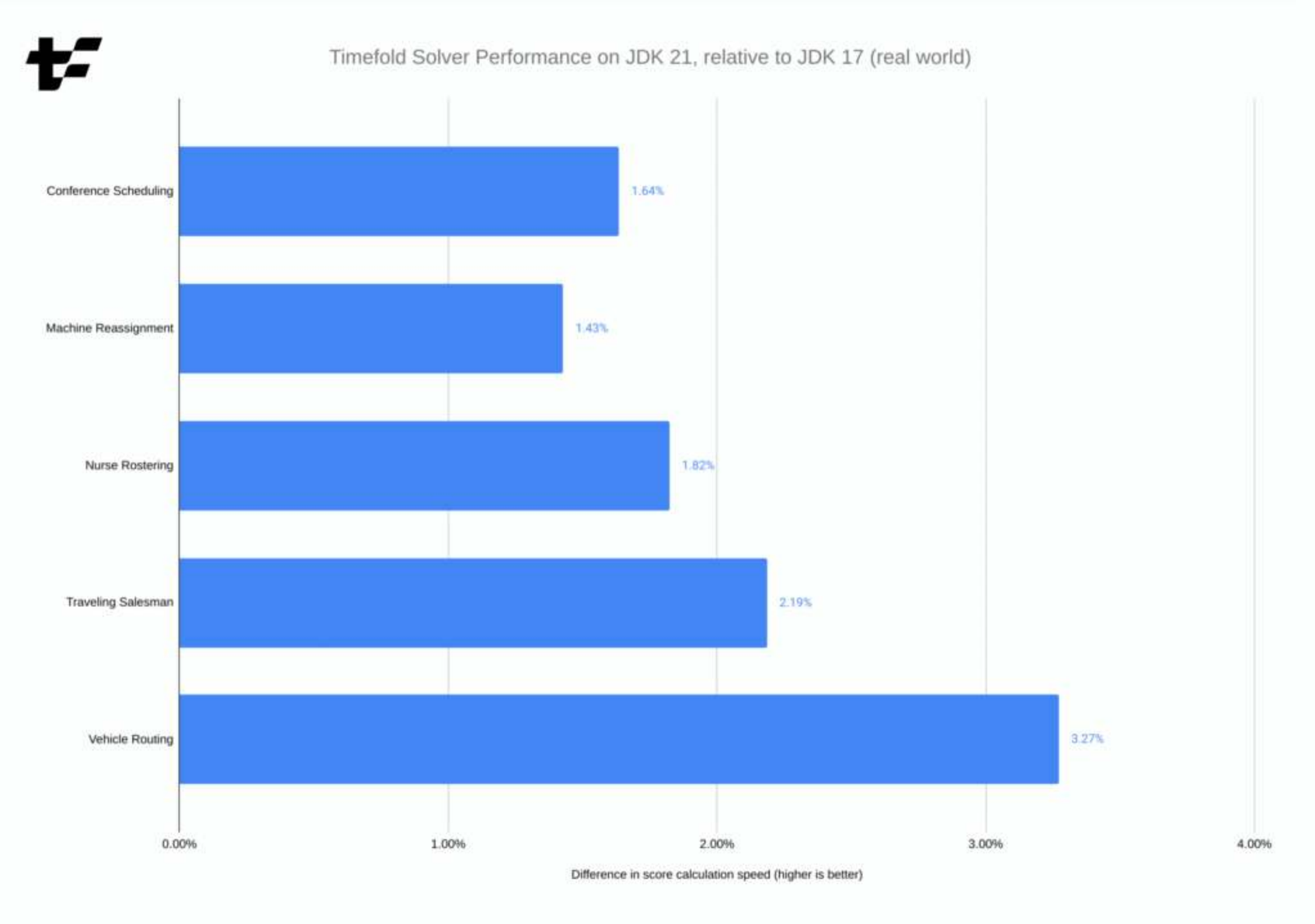
| Label | #Samples | KO | Error % | Average | Min | Max | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total | 155949 | 1920 | 1.2% | 1486.81 | 0 | 56751 | 3088.5 | 3910.2 | 5715.72 | 77.31 | 3352.08 | 0.0 |
| Test | 9681 | 1646 | 16.95% | 23262.45 | 6206 | 84815 | 31548.78 | 37471.29 | 49858.92 | 4.84 | 3236.64 | 0.0 |
| Find Owners | 19492 | 212 | 1.19% | 1105.24 | 0 | 54737 | 2517.37 | 3177.71 | 4731.66 | 9.6 | 60.37 | 0.0 |
| Edit Owner(POST) | 9761 | 186 | 1.92% | 1689.4 | 0 | 9450 | 3519.32 | 4341.77 | 6113.95 | 4.87 | 33.66 | 0.0 |
| Vests | 9796 | 158 | 1.69% | 1798.35 | 0 | 55417 | 3429.69 | 4093.12 | 5766.83 | 4.82 | 39.4 | 0.0 |
| Edit Owner Page | 9770 | 149 | 1.49% | 1087.17 | 0 | 8577 | 2565.25 | 3287.48 | 4797.34 | 4.86 | 39.12 | 0.0 |
| Add New Pet(POST) | 9743 | 147 | 1.42% | 1665.29 | 0 | 11451 | 3589.83 | 4320.97 | 6004.9 | 4.95 | 37.59 | 0.0 |
| Edit Pet(POST) | 9725 | 138 | 1.35% | 1677.87 | 0 | 9940 | 3520.24 | 4461.71 | 6090.66 | 4.73 | 42.03 | 0.0 |
| Add New Pet Page | 9753 | 128 | 1.26% | 1136.72 | 0 | 54739 | 2679.27 | 3347.99 | 4998.13 | 4.92 | 36.72 | 0.0 |
| Add New Owner(POST) | 9775 | 126 | 1.42% | 1692.81 | 0 | 55526 | 3410.21 | 4393.65 | 6030.79 | 4.91 | 33.33 | 0.0 |
| Find Owner By LastName | 9681 | 111 | 1.1% | 1668.22 | 0 | 12463 | 3472.03 | 4294.35 | 5771.14 | 4.81 | 39.23 | 0.0 |
| Add New Visit Page | 9718 | 103 | 1.1% | 1120.52 | 0 | 55546 | 2599.82 | 3245.04 | 4493.91 | 4.76 | 35.68 | 0.0 |
| Home Page | 9807 | 100 | 0.97% | 3400.82 | 0 | 13121 | 5487.95 | 6261.73 | 8062.1 | 4.89 | 2757.78 | 0.0 |
| Find Owner By ID | 9696 | 95 | 1.03% | 1121.85 | 0 | 10062 | 2606.04 | 3219.06 | 4576.59 | 4.84 | 37.86 | 0.0 |
| Add New Owner Page | 9784 | 93 | 0.97% | 1105.64 | 0 | 9122 | 2638.02 | 3267.02 | 4625.92 | 4.91 | 38.4 | 0.0 |
| Edit Pet Page | 9737 | 92 | 0.93% | 1106.44 | 1 | 9051 | 2632.15 | 3343.19 | 4958.86 | 4.89 | 36.79 | 0.0 |
| Add New Visit(POST) | 9711 | 82 | 0.84% | 1174.65 | 0 | 11351 | 2718.77 | 3380.2 | 5191.97 | 5.01 | 35.86 | 0.0 |

PetClinic_JMeter_Results - JDK 17

| Label | #Samples | KO | Error % | Average | Min | Max | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total | 155949 | 1920 | 1.23% | 1625.69 | 0 | 61737 | 3359.0 | 4237.0 | 6230.99 | 85.46 | 3698.19 | 0.0 |
| Test | 9681 | 1646 | 17.00% | 26008.99 | 6722 | 91500 | 35522.2 | 41307.5 | 55217.26 | 5.25 | 3617.78 | 0.0 |
| Find Owners | 19492 | 212 | 1.09% | 1199.71 | 0 | 60264 | 2781.0 | 3567.0 | 5198.56 | 10.71 | 66.15 | 0.0 |
| Edit Owner(POST) | 9761 | 186 | 1.91% | 1828.14 | 0 | 10530 | 3897.6 | 4862.0 | 6665.8 | 5.37 | 36.85 | 0.0 |
| Vests | 9796 | 158 | 1.61% | 2041.87 | 0 | 61737 | 3809.0 | 4567.15 | 6290.87 | 5.38 | 42.35 | 0.0 |
| Edit Owner Page | 9770 | 149 | 1.53% | 1220.9 | 0 | 9325 | 2831.8 | 3583.25 | 5214.61 | 5.37 | 42.4 | 0.0 |
| Add New Pet(POST) | 9743 | 147 | 1.51% | 1826.14 | 0 | 12431 | 3850.0 | 4737.2 | 6609.52 | 5.37 | 41.07 | 0.0 |
| Edit Pet(POST) | 9725 | 138 | 1.42% | 1853.48 | 0 | 10826 | 3924.4 | 4847.8 | 6740.06 | 5.35 | 45.1 | 0.0 |
| Add New Pet Page | 9753 | 128 | 1.31% | 1249.49 | 0 | 60820 | 2898.0 | 3705.2 | 5499.68 | 5.37 | 41.16 | 0.0 |
| Add New Owner(POST) | 9775 | 126 | 1.29% | 1886.76 | 0 | 60785 | 3854.4 | 4815.6 | 6816.84 | 5.38 | 37.02 | 0.0 |
| Find Owner By LastName | 9681 | 111 | 1.15% | 1807.01 | 0 | 13669 | 3836.6 | 4670.9 | 6496.88 | 5.34 | 43.78 | 0.0 |
| Add New Visit Page | 9718 | 103 | 1.06% | 1215.75 | 0 | 60820 | 2849.3 | 3584.25 | 5048.43 | 5.36 | 39.46 | 0.0 |
| Home Page | 9807 | 100 | 1.02% | 3733.1 | 0 | 14214 | 5973.0 | 6829.6 | 9007.52 | 5.37 | 3100.9 | 0.0 |
| Find Owner By ID | 9696 | 95 | 0.98% | 1210.31 | 0 | 11016 | 2830.6 | 3576.45 | 4996.3 | 5.36 | 41.17 | 0.0 |
| Add New Owner Page | 9784 | 93 | 0.95% | 1205.7 | 0 | 9913 | 2861.5 | 3611.75 | 5160.45 | 5.37 | 42.35 | 0.0 |
| Edit Pet Page | 9737 | 92 | 0.94% | 1233.34 | 1 | 9935 | 2909.2 | 3697.1 | 5483.48 | 5.37 | 41.43 | 0.0 |
| Add New Visit(POST) | 9711 | 82 | 0.84% | 1283.03 | 0 | 12414 | 3010.6 | 3789.2 | 5585.84 | 5.36 | 40.08 | 0.0 |

I was able to achieve 9% gains by just changing Java version

I will be using both my own results, as well as available numbers




“How fast is Java 21?” by timefold.ai

# How much faster is Java 17?

Geoffrey De Smet
OptaPlanner creator

Wed 15 September 2021

Java 17 (released yesterday) comes with many new features and enhancements. However, most of those require code changes to benefit from. *Except for performance.* Simply switch your JDK installation and you get a free performance boost. But how much? Is it worth it? Let's find out by comparing the benchmarks of JDK 17, JDK 16 and JDK 11.

## Executive summary

On average, for OptaPlanner use cases, these benchmarks indicate that:

- **Java 17 is 8.66% faster than Java 11** and 2.41% faster than Java 16 for G1GC (default).
- Java 17 is 6.54% faster than Java 11 and 0.37% faster than Java 16 for ParallelGC.
- The Parallel Garbage Collector is 16.39% faster than the G1 Garbage Collector.

No big surprises here: the latest JDK is faster and the high throughput garbage collector is faster than the low latency garbage collector.

"How much faster is Java 17?" by optaplanner.org

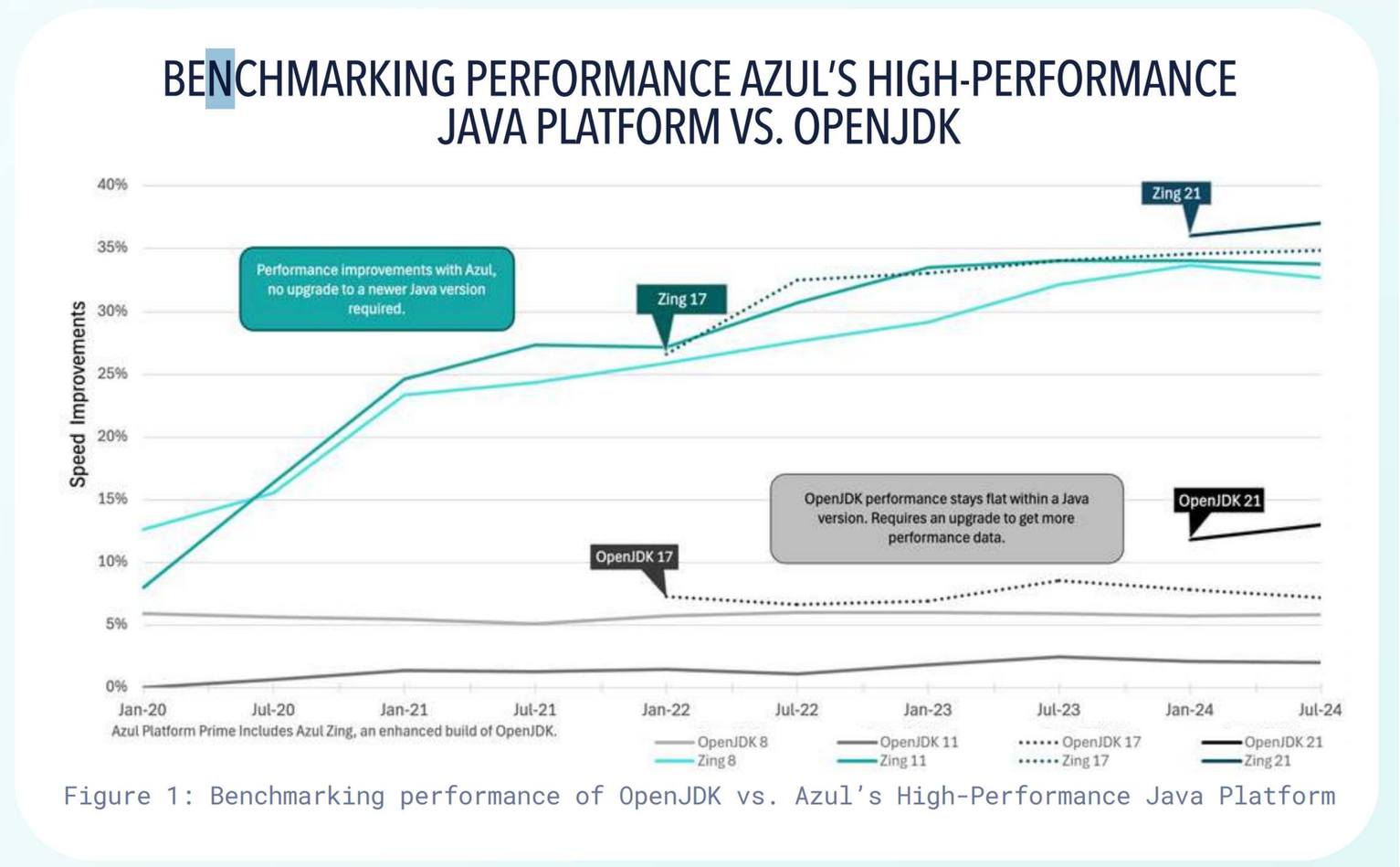



Figure 1: Benchmarking performance of OpenJDK vs. Azul's High-Performance Java Platform

"High-Performance Java Platform Benchmark Report " by Azul

Documentation > Amazon Q > User Guide

# Upgrading Java versions with Amazon Q Developer

PDF RSS Focus mode

Amazon Q Developer can upgrade your Java applications to newer language versions in the integrated development enviror (IDE). Changes Amazon Q can make to upgrade your code include updating deprecated code components and APIs as well a upgrading libraries, frameworks, and other dependencies in your code.

To transform your code, Amazon Q first builds your code in the source language version and verifies that it has the informat necessary to perform the transformation. After Amazon Q successfully transforms your code, you verify and accept the char your IDE. For more information about how Amazon Q transforms your code, see How Amazon Q Developer transforms code Java language upgrades.

**Topics**

Learn / Azure / Migrate / Ask Learn

# Azure Migrate application and code assessment for Java

02/26/2025

In this article

- Overview
- Download
- Get started
- Documentation

Show 6 more

**Note**

We've released a new version of *Azure Migrate application and code assessment for Java*, version 7.x under preview, based on a new analyzer engine. To download and experiment with the new engine, available for Windows, Linux, and macOS, see Azure Migrate application and code assessment for Java version 7

This guide describes how to use the Azure Migrate application and code assessment tool for Java to assess and replatform any type of Java application. The tool enables you to evaluate application readiness for replatforming and migration to Azure. This tool is offered as a CLI (command-line interface) and assesses Java application binaries

Rewriting is much easier

Documentation > Amazon Q > User Guide

# Upgrading Java versions with

PDF RSS Focus mode

Amazon Q Developer can upgrade your Java applications to newer language ve
(IDE). Changes Amazon Q can make to upgrade your code include updating dep
upgrading libraries, frameworks, and other dependencies in your code.

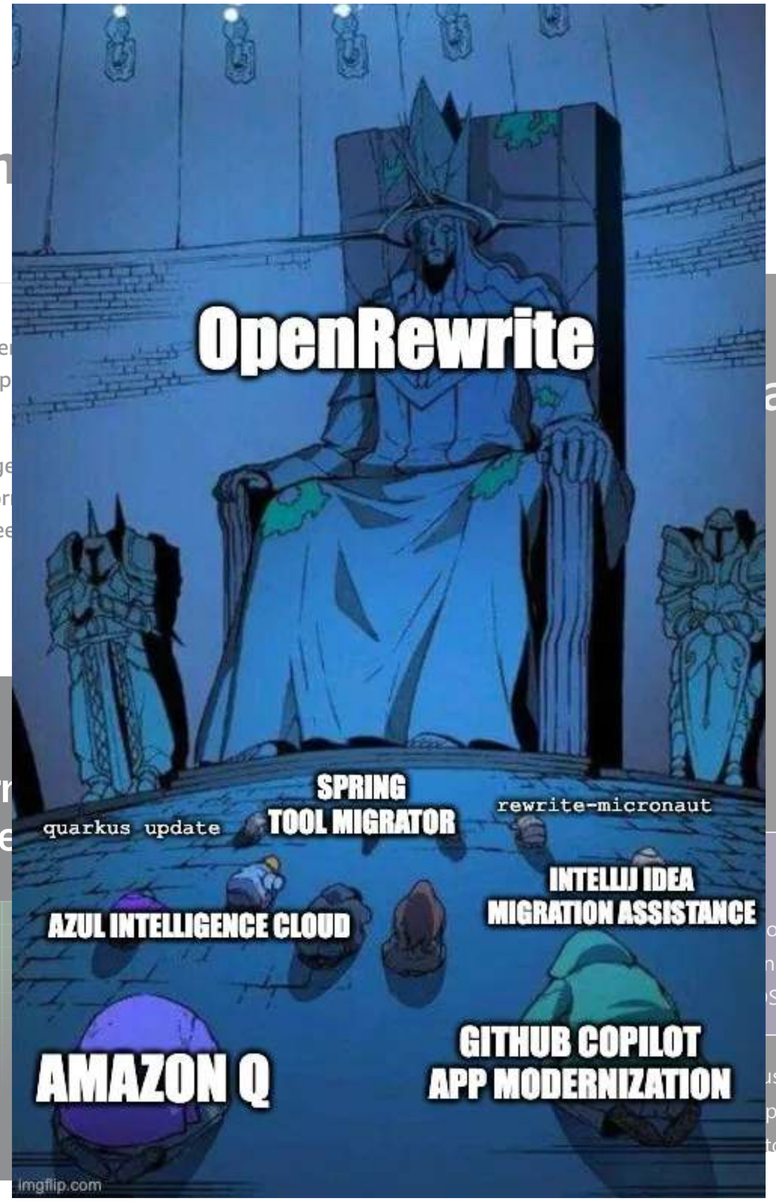
To transform your code, Amazon Q first builds your code in the source language
necessary to perform the transformation. After Amazon Q successfully transfor
your IDE. For more information about how Amazon Q transforms your code, see
Java language upgrades.

Topics

RELEASE

MAY 19, 2025 • 2 MINUTE READ

GitHub Copilot app moderr
for Java now in public pre




Ask Learn

ate application and code
for Java

on of *Azure Migrate application and code assessment for Java*, version 7.x under
nalyzer engine. To download and experiment with the new engine, available for
S, see Azure Migrate application and code assessment for Java version 7

use the Azure Migrate application and code assessment tool for Java to assess and
pplication. The tool enables you to evaluate application readiness for replatforming
tool is offered as a CLI (command-line interface) and assesses Java application binaries

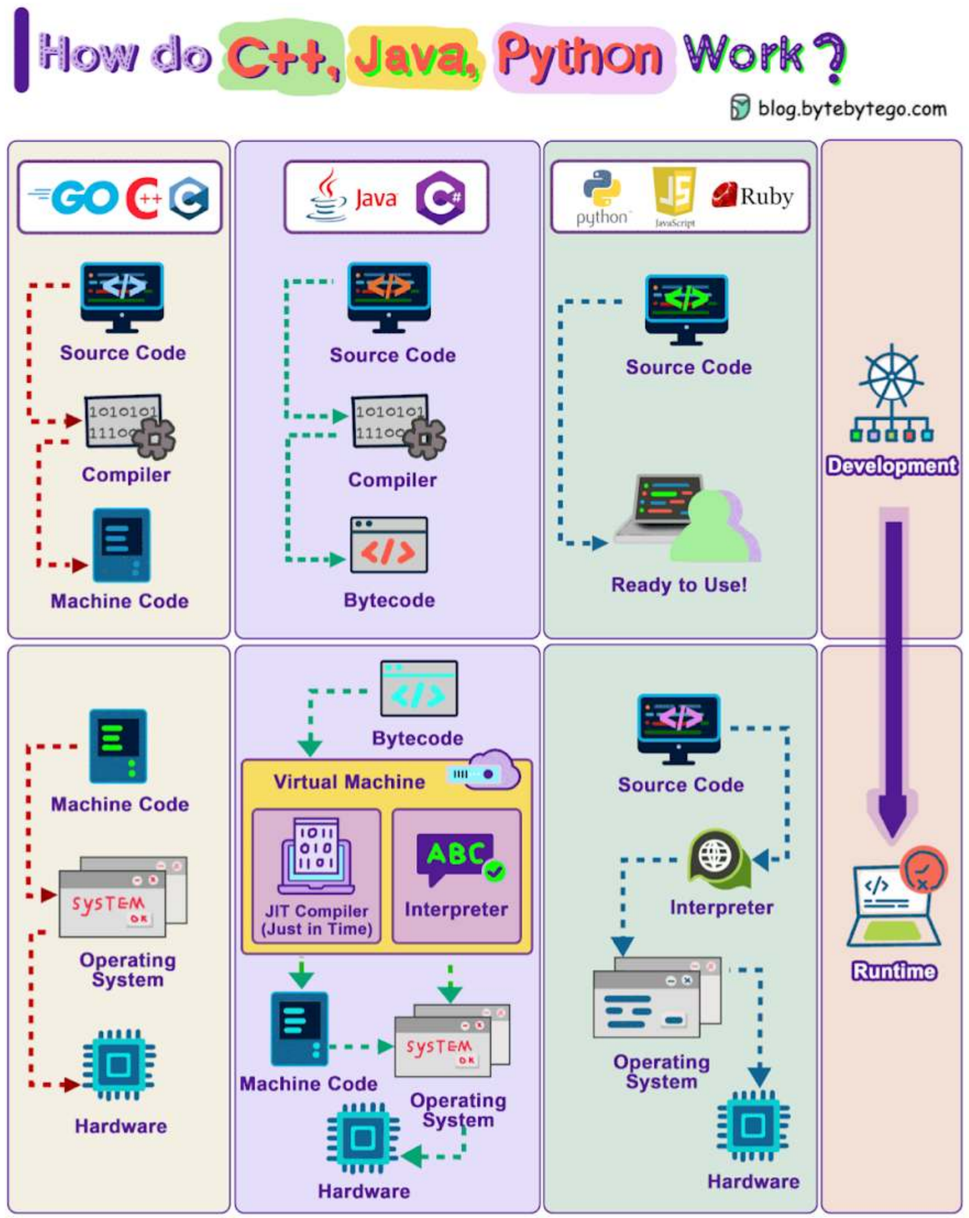
Rewriting is much easier... thanks to OpenRewrite

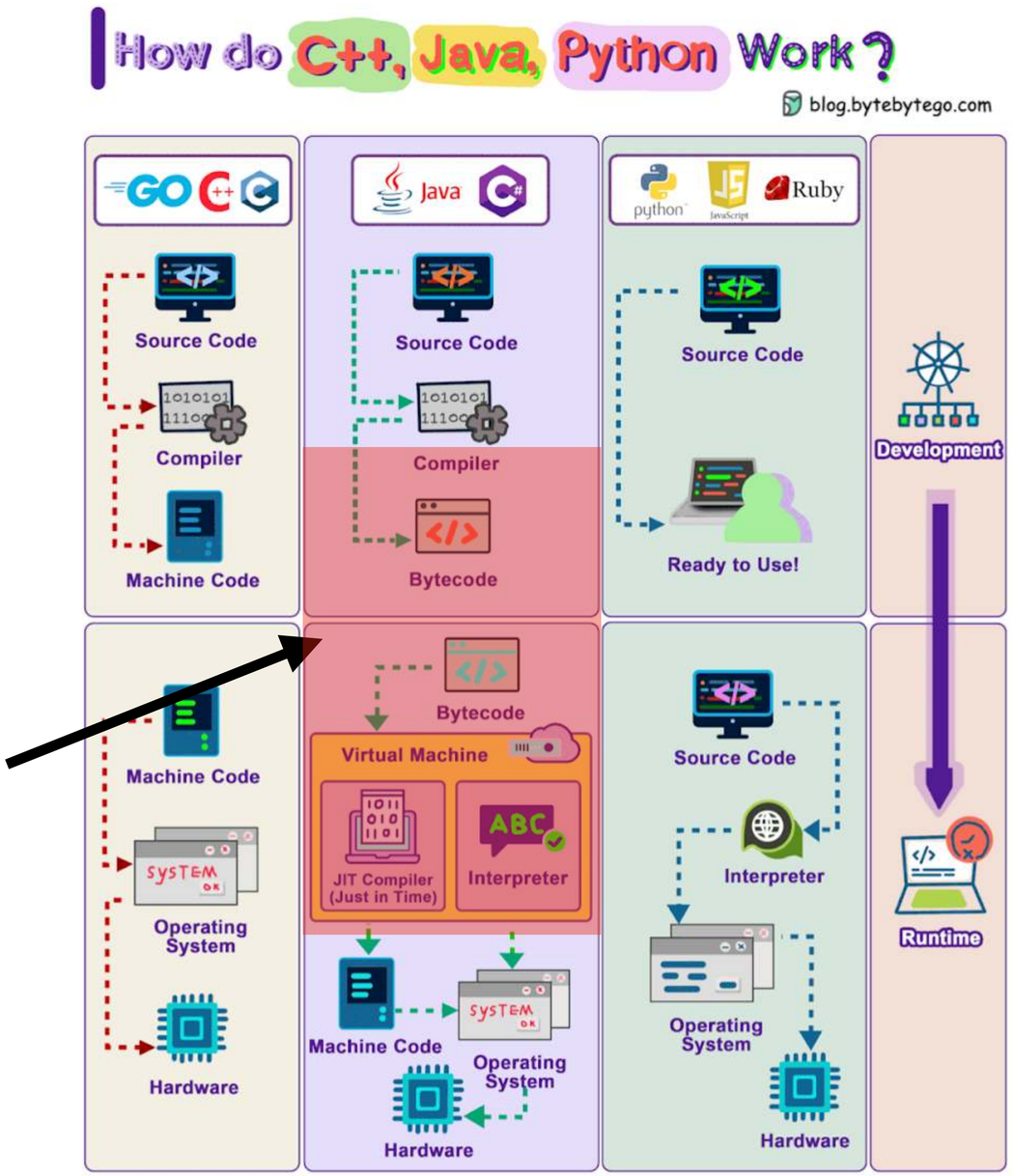
## It’s very simple:

- **Always** use the most modern JDK ! But especially upgrade to JDK 17 is a blast.
- However… be careful about deprecations, there is a lot of cleanup recently.
- The most volatile parts are Garbage Collectors. Mostly for good, but sometimes there are surprises
- Consider “special versions” like Azul Prime Platform, they can bring a lot of value if you have “niche” needs

# Heuristics to use

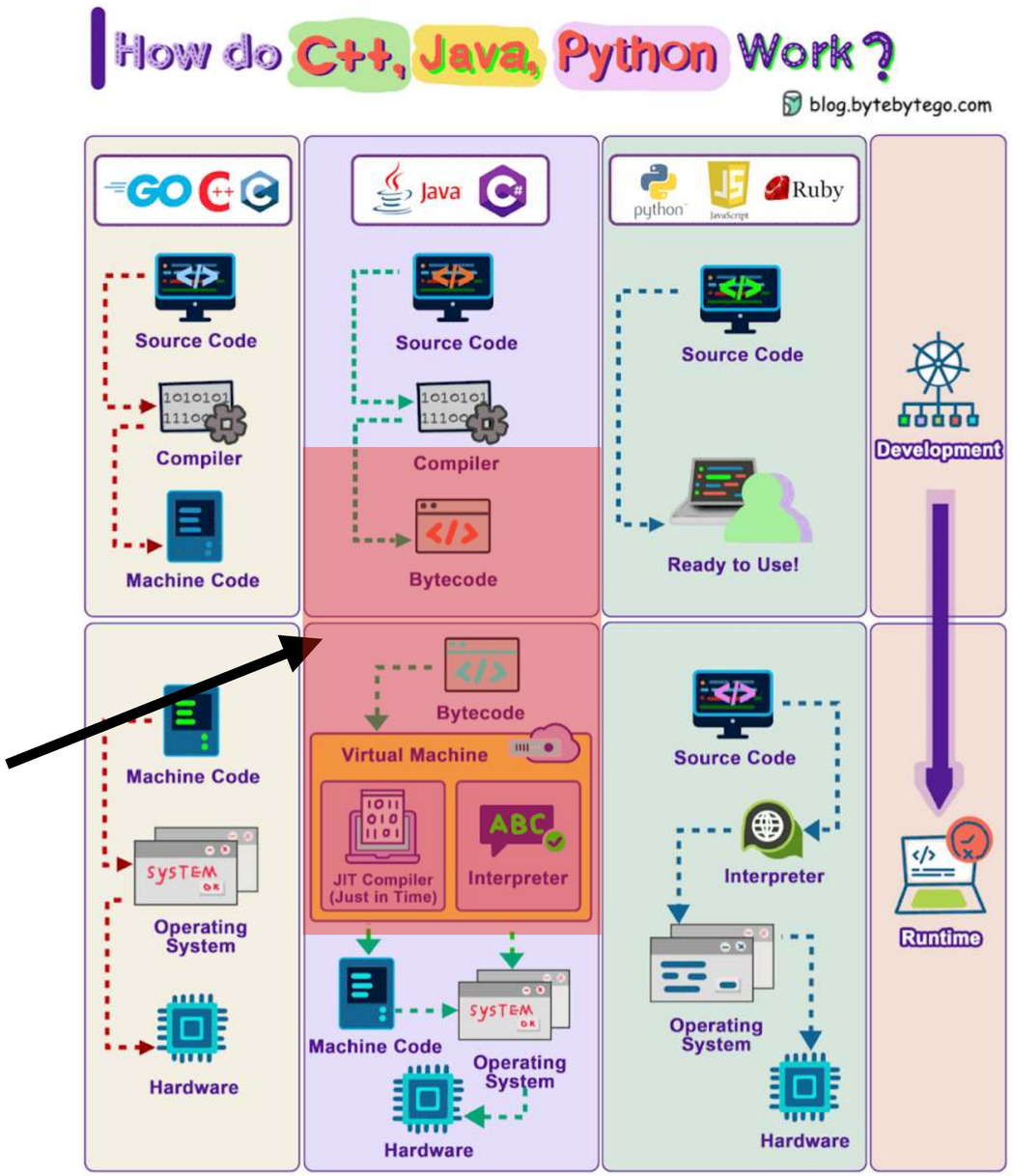
# Cold Start


How do C++, Java, Python Work?
blog.bytebytego.com
GO
C++
Source Code
Compiler
Machine Code
Java
Source Code
Compiler
Bytecode
python
JavaScript
Ruby
Source Code
Ready to Use!
Development
Machine Code
Operating System
Hardware
Bytecode
Virtual Machine
JIT Compiler (Just in Time)
Interpreter
Machine Code
Operating System
Hardware
Source Code
Interpreter
Operating System
Hardware
Runtime

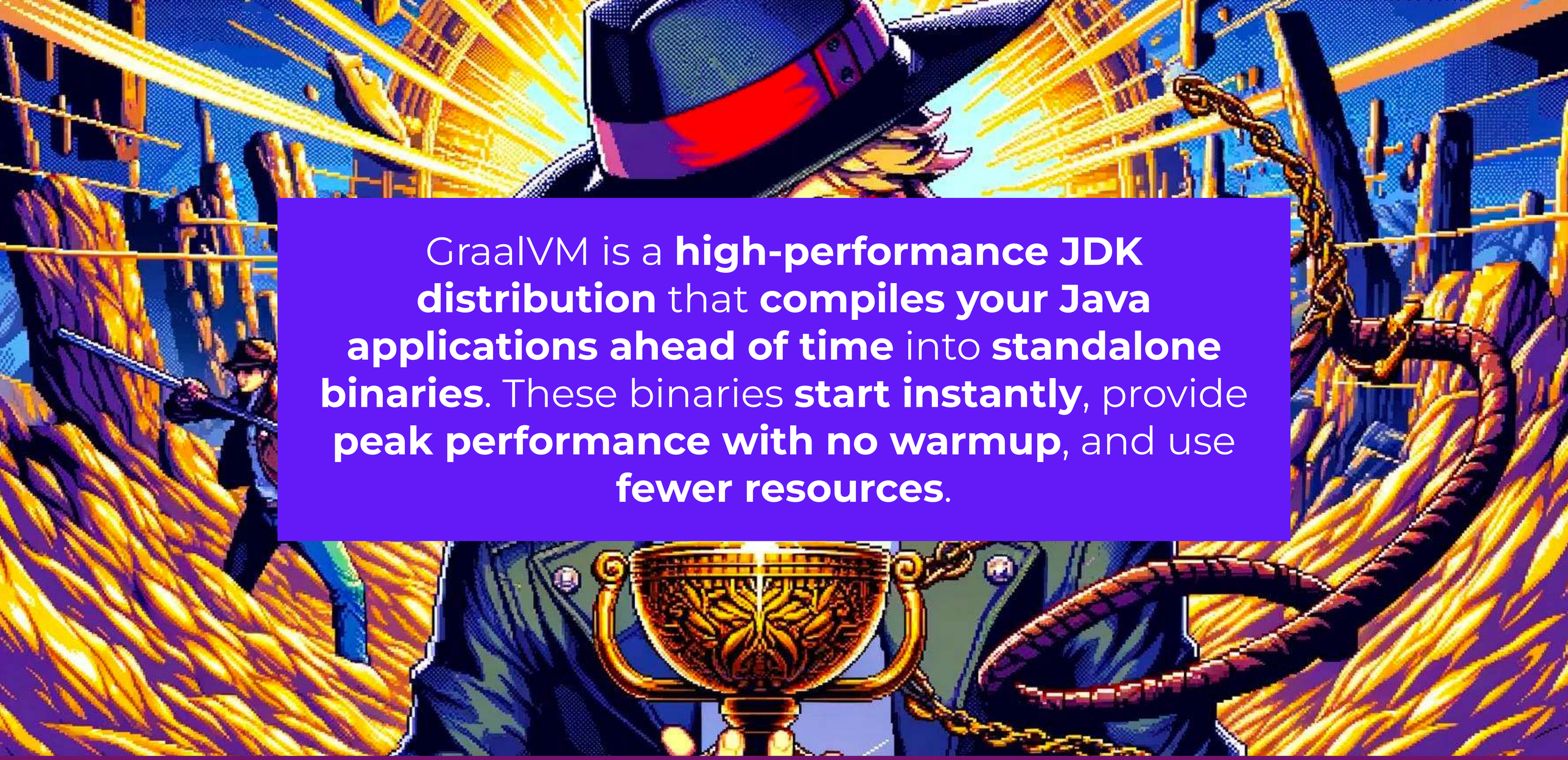

How do C++, Java, Python Work?
blog.bytebytego.com
GO
C++
Source Code
Compiler
Machine Code
Machine Code
Operating System
Hardware
Java
Source Code
Compiler
Bytecode
Bytecode
Virtual Machine
JIT Compiler (Just in Time)
Interpreter
Machine Code
Operating System
Hardware
python
JavaScript
Ruby
Source Code
Ready to Use!
Source Code
Interpreter
Operating System
Hardware
Development
Runtime

AND THEN WE TOLD THEM
WRITE ONCE RUN ANYWHERE
memegenerator.net

gifbin.com


How do C++, Java, Python Work?
blog.bytebytego.com
GO
C++
Source Code
Compiler
Machine Code
Machine Code
Operating System
Hardware
Java
Source Code
Compiler
Bytecode
Bytecode
Virtual Machine
JIT Compiler (Just in Time)
Interpreter
Machine Code
Operating System
Hardware
python
JavaScript
Ruby
Source Code
Ready to Use!
Source Code
Interpreter
Operating System
Hardware
Development
Runtime

GraalVM

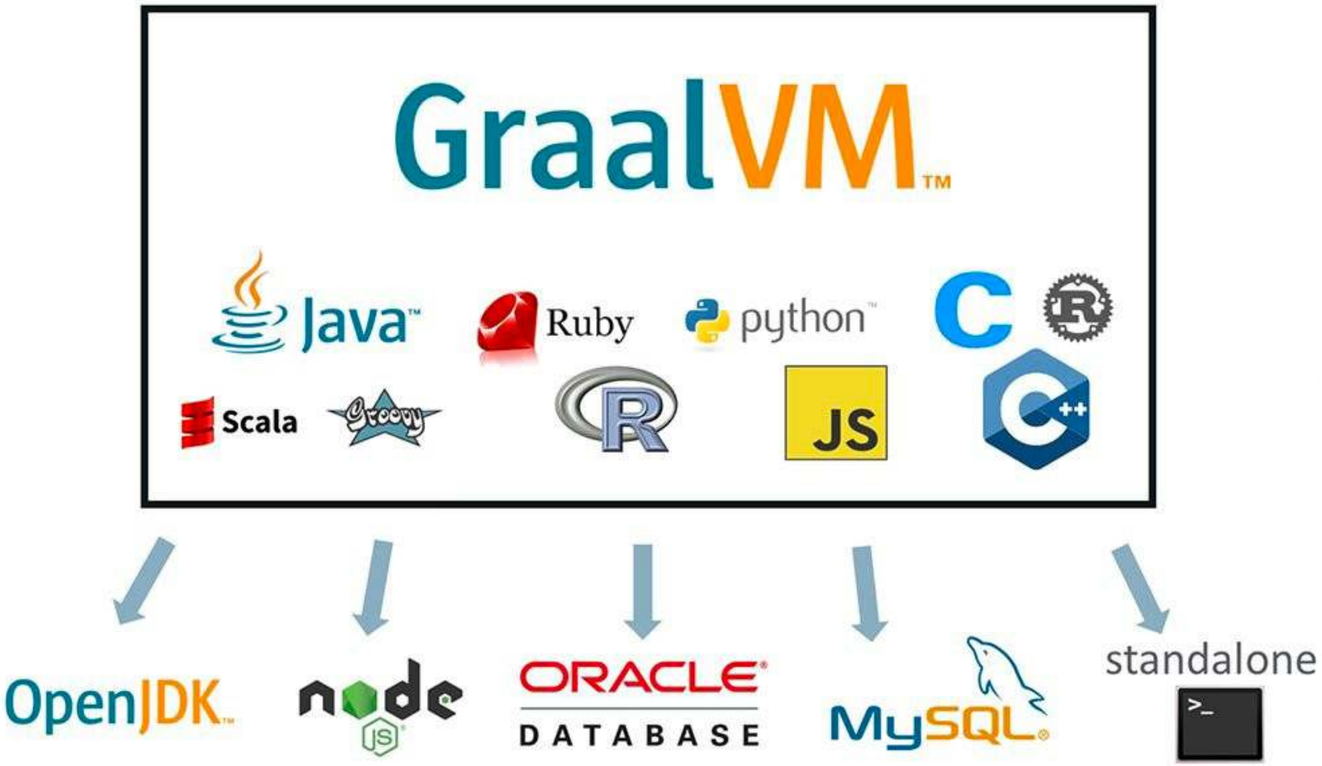


GraalVM is a **high-performance JDK distribution** that **compiles your Java applications ahead of time** into **standalone binaries**. These binaries **start instantly**, provide **peak performance with no warmup**, and use **fewer resources**.

GraalVM

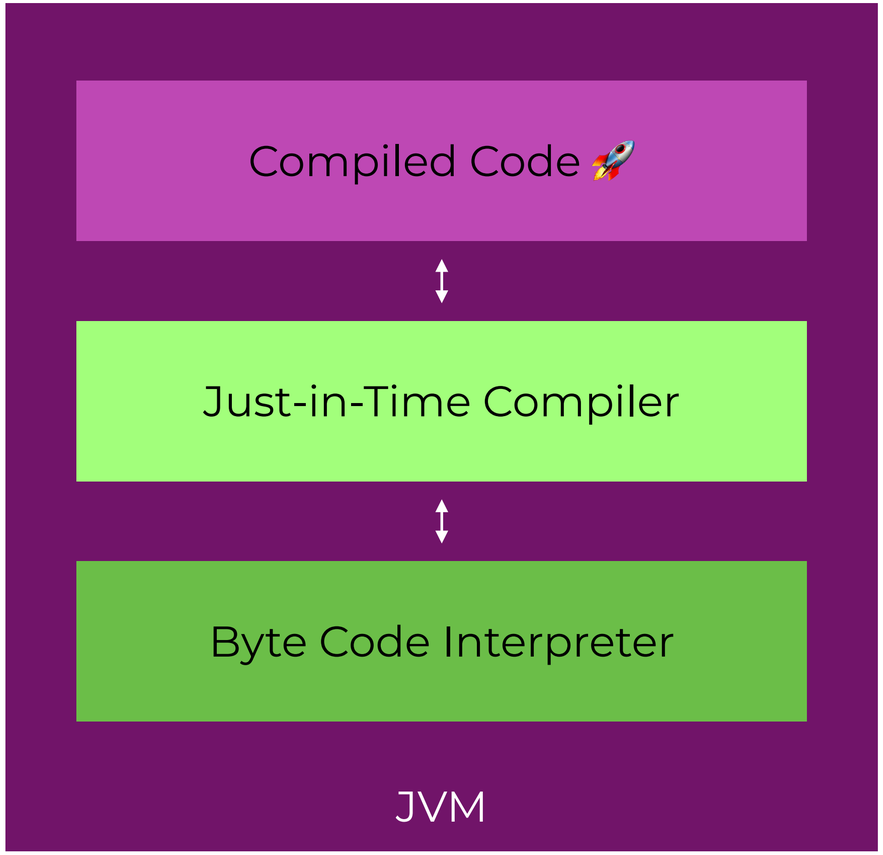
makeagif.com

GraalVM

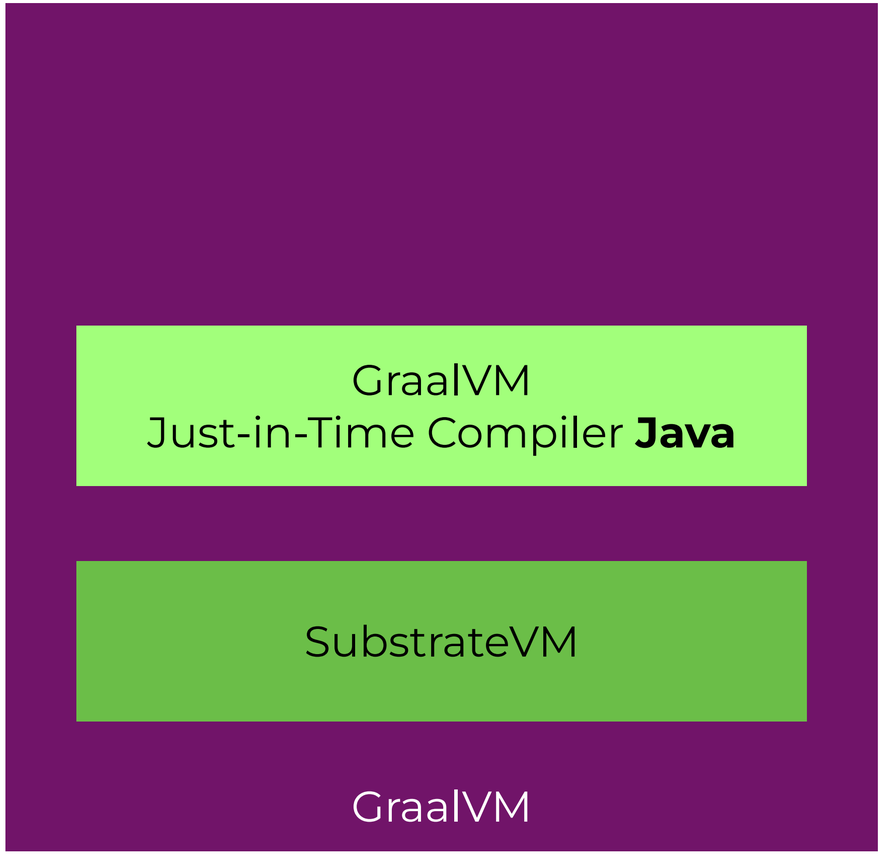

GraalVM
Java
Ruby
python
C
Scala
Groovy
R
JS
C++
OpenJDK
node
ORACLE
DATABASE
MySQL
standalone

LLVM

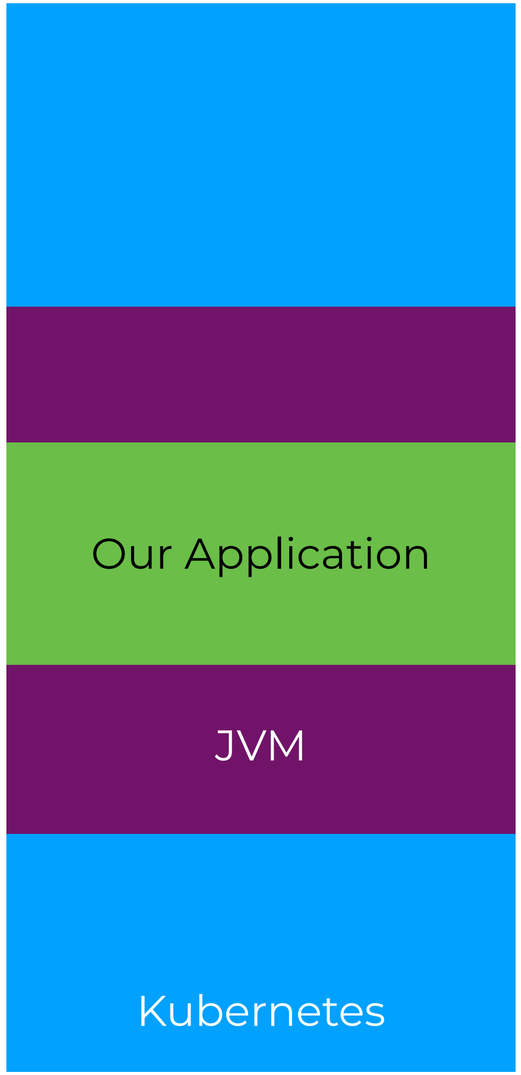
# GraalVM


Compiled Code 🚀
Just-in-Time Compiler
Byte Code Interpreter
JVM

# GraalVM




# GraalVM




GraalVM

GraalVM
Just-in-Time Compiler Java

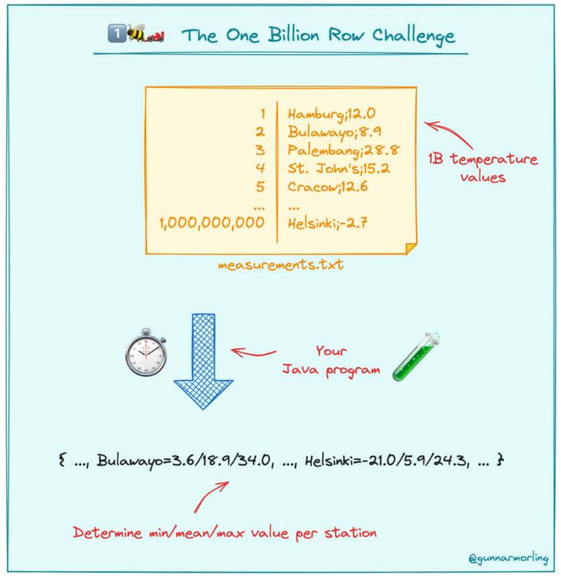
# GraalVM

# 1️⃣🐝🏎️ The One Billion Row Challenge

*Status Jan 12: As there has been such a large number of entries to this challenge so far (100+), and this is becoming hard to manage, please only create new submissions if you expect them to run in 10 seconds or less on the evaluation machine.*

*Status Jan 1: This challenge is open for submissions!*

The One Billion Row Challenge (1BRC) is a fun exploration of how far modern Java can be pushed for aggregating one billion rows from a text file. Grab all your (virtual) threads, reach out to SIMD, optimize your GC, or pull any other trick, and create the fastest implementation for solving this task!




## Results

These are the results from running all entries into the challenge on eight cores of a Hetzner AX161 dedicated server (32 core AMD EPYC™ 7502P (Zen2), 128 GB RAM).

| # | Result (m:s.ms) | Implementation | JDK | Submitter | Notes |
|---|---|---|---|---|---|
| 1 | 00:01.878 | link | 21.0.2-graal | Thomas Wuerthinger, Quan Anh Mai, Alfonso² Peterssen | GraalVM native binary, uses Unsafe |
| 2 | 00:01.926 | link | 21.0.2-graal | Artsiom Korzun | GraalVM native binary, uses Unsafe |
| 3 | 00:01.970 | link | 21.0.2-graal | Van Phu DO | GraalVM native binary, uses Unsafe |
| | 00:02.081 | link | 21.0.2-graal | Jaromir Hamala | GraalVM native binary, uses Unsafe |
| | 00:02.157 | link | 21.0.2-graal | Roy van Rijn | GraalVM native binary, uses Unsafe |
| | 00:02.188 | link | 21.0.1-open | Serkan ÖZAL | uses Unsafe |
| | 00:02.205 | link | 21.0.2-graal | tivrfoa | GraalVM native binary, uses Unsafe |
| | 00:02.319 | link | 21.0.2-graal | Yavuz Tas | GraalVM native binary, uses Unsafe |

Holy Grails are often not that Holy

# Closed World Assumption

# Closed World Assumption

# Introducing the Tracing Agent: Simplifying GraalVM Native Image Configuration

Christian Wimmer · Follow
Published in graalvm · 6 min read · Jun 5, 2019

385 3

*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

*GraalVM Native Image Generation* applies static analysis and ahead-of-time compilation to build a highly optimized native executable, called *Native Image*, for Java applications. This requires a closed-world assumption of reachable application classes: all classes need to be known at native image generation time so that the static analysis can process them.

The closed-world assumption conflicts with the open-world approach of Java reflection: Using functionality from the package `java.lang.reflect`, an application developer can look up classes, methods, and fields by name and access / invoke them. In typical cases, the names are loaded from

README.md

# GraalVM Reachability Metadata Repository

This repository enables users of GraalVM Native Image to share and reuse metadata for libraries and frameworks in the Java ecosystem. The repository is integrated with GraalVM Native Build Tools beginning with version `0.9.13`: you can enable automatic use of the metadata repository for Gradle projects or Maven projects.

This web page provides an overview of libraries and frameworks that are tested and thus ready for GraalVM Native Image. If you would like to see your library or framework in the list too, please open a pull request and extend this JSON file. Before submitting a pull request, please read this guide.

## Rationale

When you use Native Image to build native executables it only includes the elements reachable from your application entry point, its dependent libraries, and JDK classes discovered through static analysis. However, the reachability of some elements (such as classes, methods, or fields) may not be discoverable due to Java's dynamic features including reflection, resource access, dynamic proxies, and serialization. If an element is not reachable, it is not included in the generated executable at build time, which can lead to failures at run time. Native Image has built-in metadata for JDK classes but user code and dependencies may use dynamic features of Java that are undiscoverable by the Native Image analysis. For this reason, Native Image accepts additional reachability metadata in the form of JSON files. Since this metadata is specific to a specific code base, the JSON files providing the corresponding metadata can be shared for libraries and frameworks. This repository is a centralized place for sharing such files for libraries and frameworks that do not provide built-in metadata yet. It is also used to retrofit metadata for older versions of libraries and frameworks.

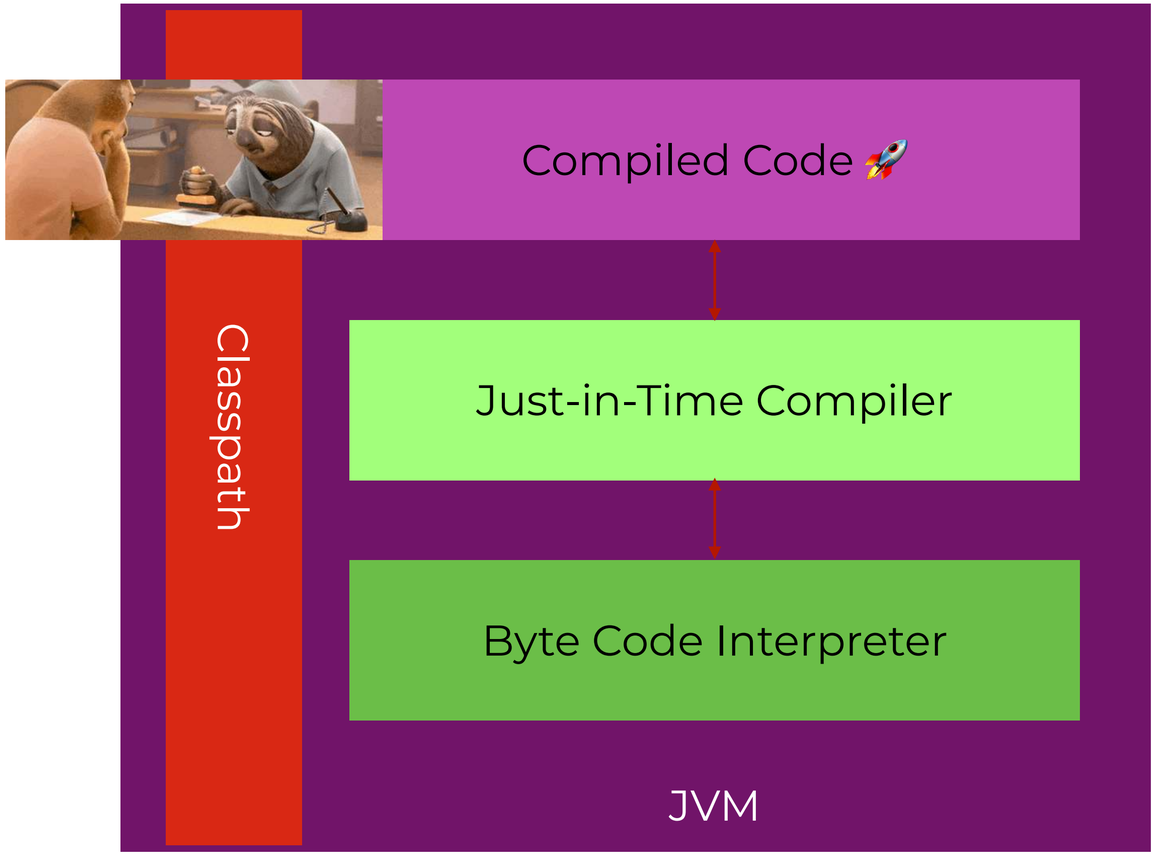
If you are a library or framework maintainer, the best way to make your code ready for GraalVM Native Image is to provide reachability metadata as part of your JARs. Please visit this web page for more information.

## Contributing

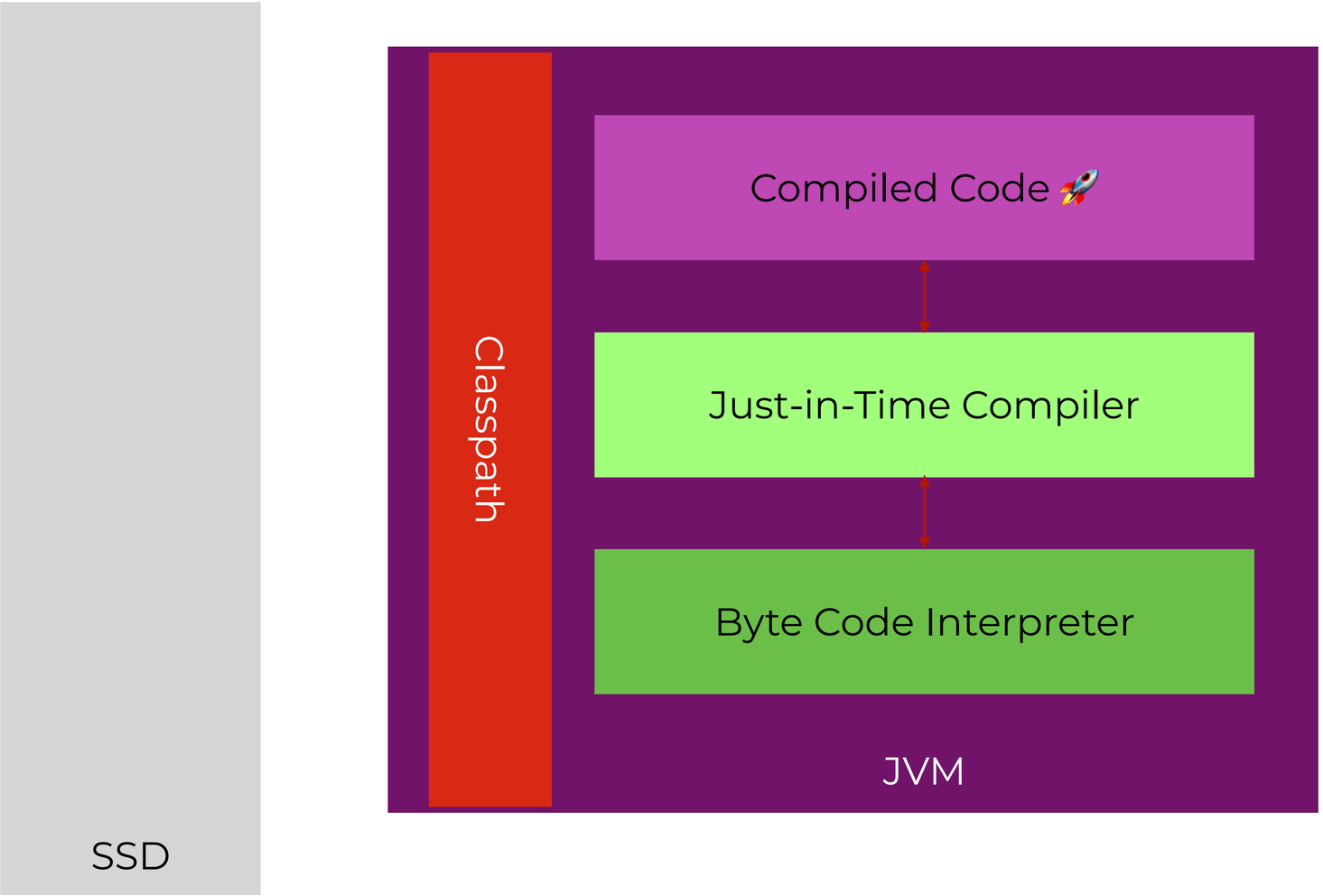
We welcome contributions from the community. Before submitting a pull request, please review our contribution guide.

# Closed World Assumption

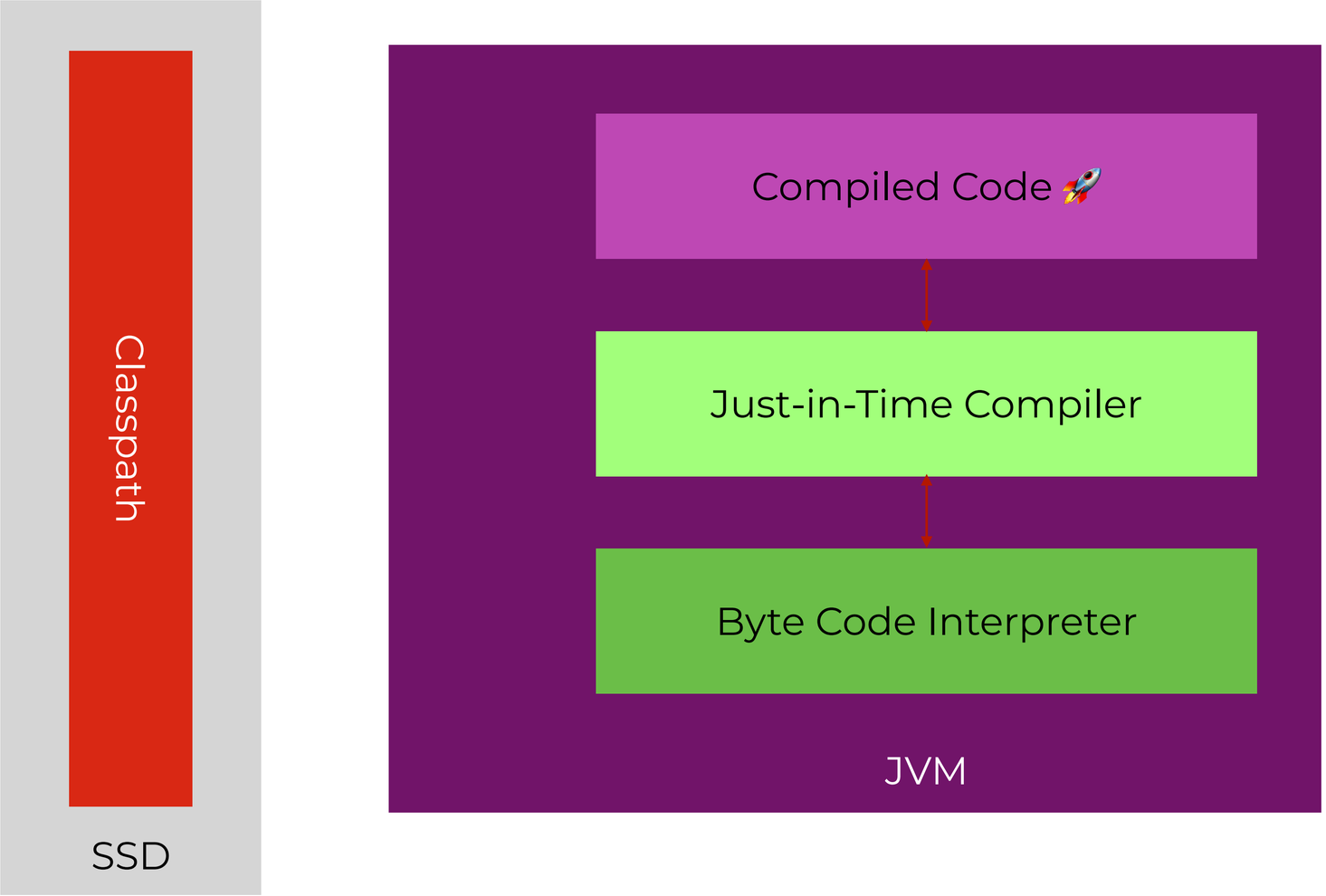
To improve startup and footprint, extend the existing **Class-Data Sharing ("CDS")** feature to allow application classes to be placed in the shared archive.



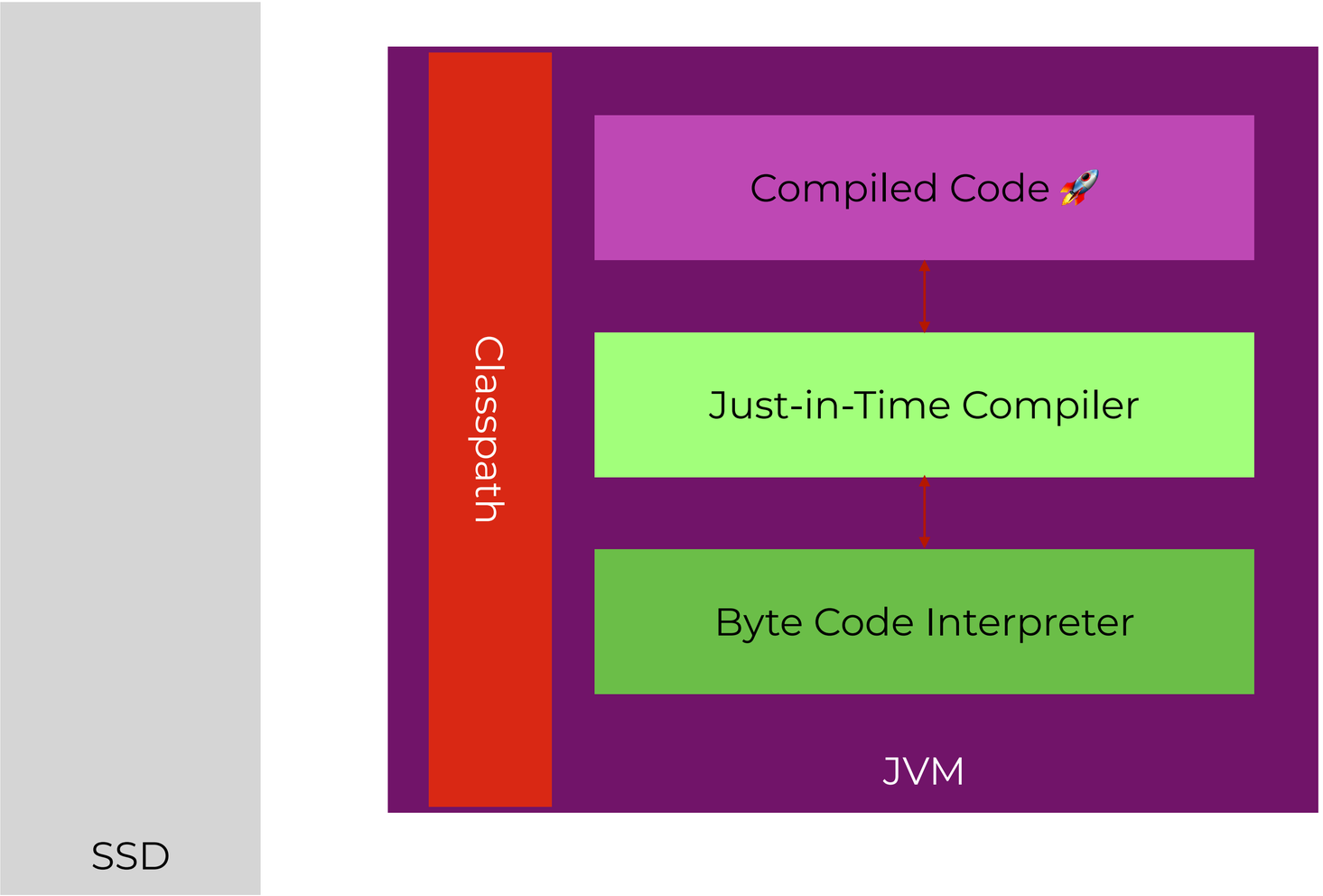

# Project Leyden




# Project Leyden


Classpath
SSD
Compiled Code 🚀
Just-in-Time Compiler
Byte Code Interpreter
JVM

# AppCDS




# Project Leyden

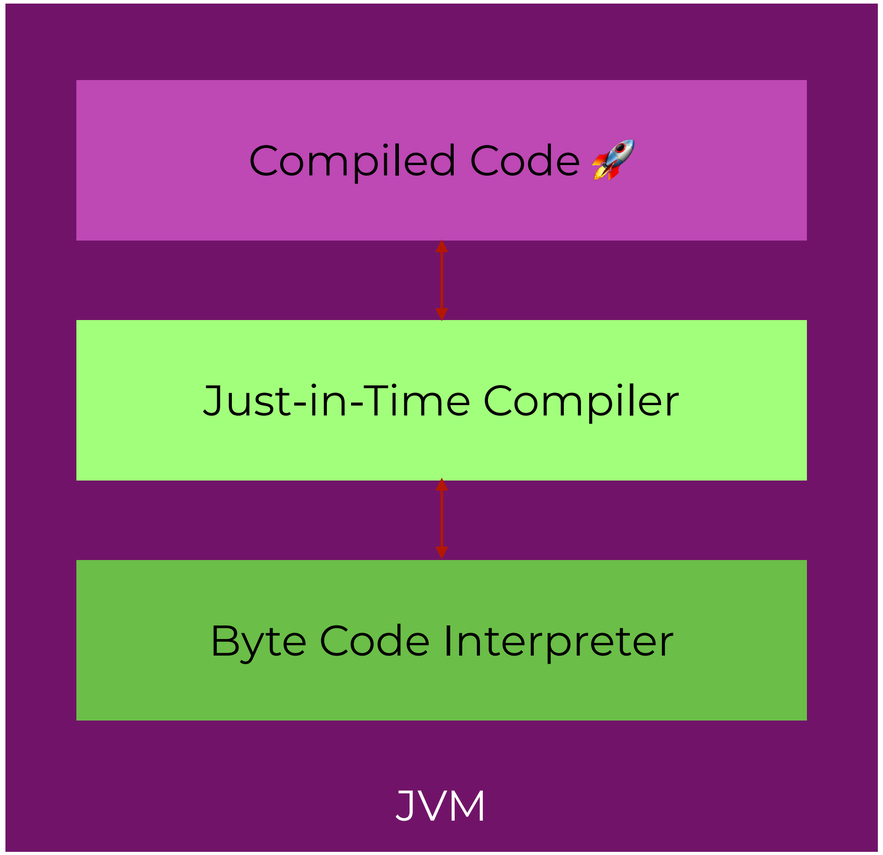
THIS IS
YOUR BRAIN
THIS IS YOUR
BRAIN ON DRUGS

Project Coordinated Restore at Checkpoint

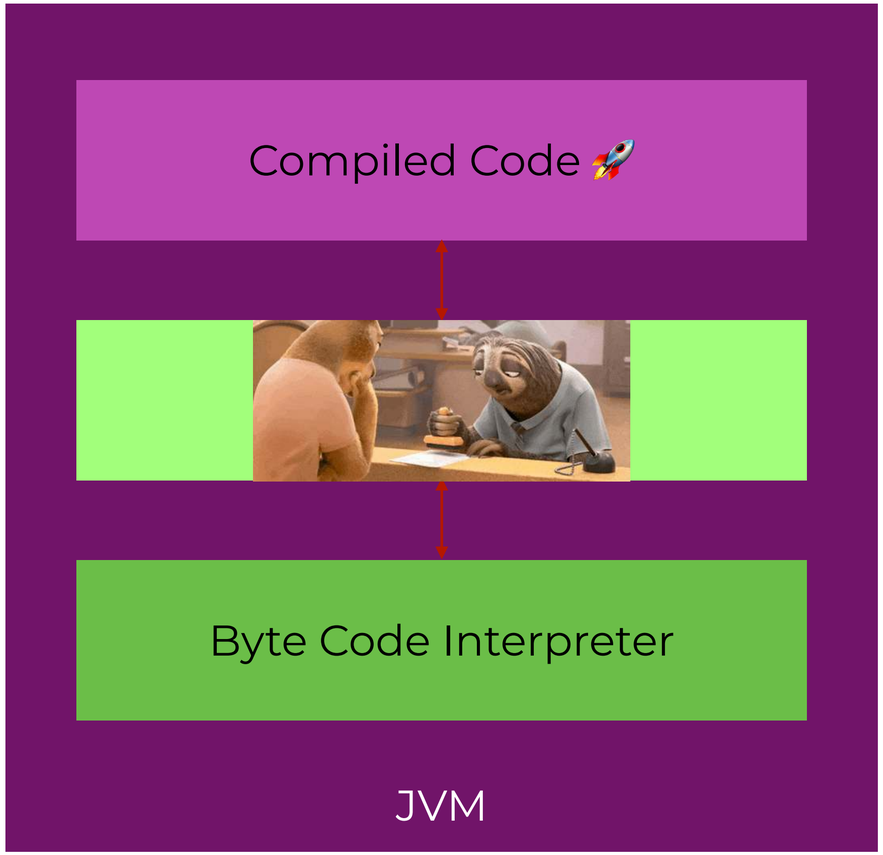
The **CRaC Project** researches **coordination of Java** programs with **mechanisms to checkpoint** a Java instance while it is executing.

Restoring from the image could be a solution to some of the problems with the **start-up** and **warm-up** times

Project Coordinated Restore at Checkpoint




# Project CRaC




# Project CRaC

AppCDS

NOOO... YOU CANNOT
SAVE STATE, YOU ARE
BREAKING ORIGINAL EXPERIENCE
imgflip.com

CRaC

SUPER
MARIO BROS.
HAHA, EMULATOR
GOES BRRRRRR

# CRaC

## Coordinated Restore at Checkpoint

org.crac Java library
Maven Central

JavaDoc for the org.crack
JavaDoc

## Other Useful Links

OpenJDK CRaC Project
Page

OpenJDK CRaC Project
Wiki Page

OpenJDK with CRaC
GitHub

org.crac Java library
GitHub

Examples of using CRaC
GitHub

Azul CRaC Documentation
Azul.com

## CRaC: Coordinated Restore at Checkpoint.

The CRaC (Coordinated Restore at Checkpoint) Project researches the coordination of Java programs with mechanisms to checkpoint (make an image of, snapshot) a Java instance while executing. Restoring from the image could solve some of the problems with the start-up and warm-up times. The primary aim of the Project is to develop a new standard mechanism-agnostic API to notify Java programs about the checkpoint and restore events. Other research activities will include, but will not be limited to, integration with existing checkpoint/restore mechanisms and development of new ones, changes to JVM and JDK to make images smaller and ensure they are correct.

To develop using CRaC API for the Java runtimes with no CRaC support (spericially if you are developing on macOS or Windows), use the org.crac Java library (JavaDoc).

## Projects with CRaC Support

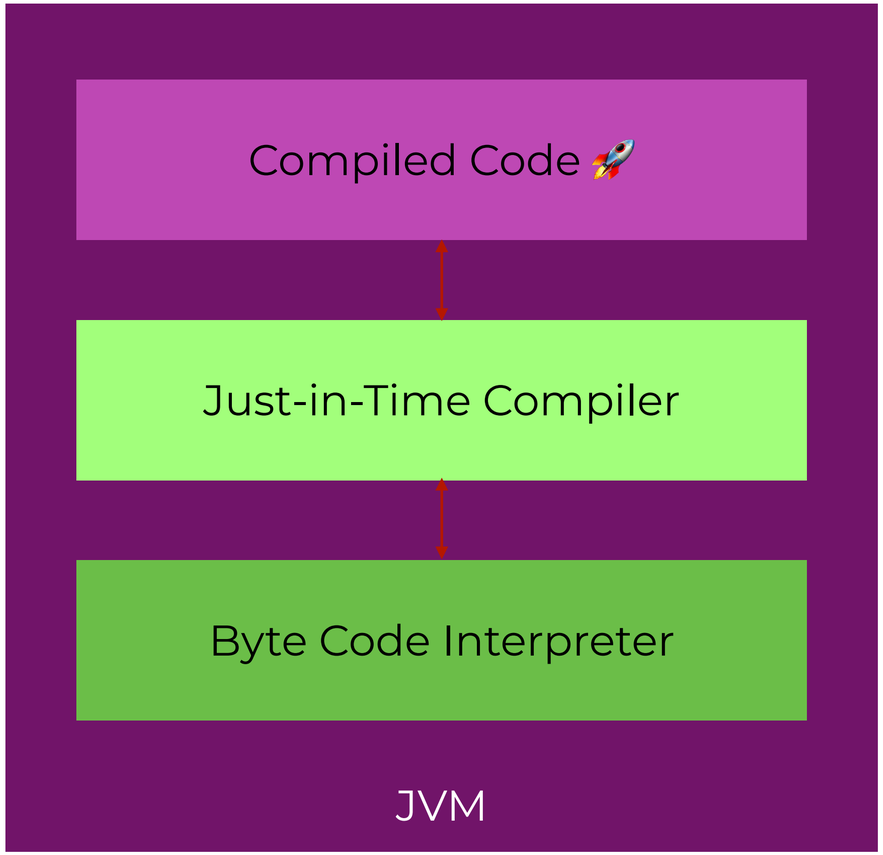
- Micronaut added support for CRaC to the Micronaut CRaC Module (Hikari DataSources and Redis coordination) and the Build System (a single command to generate a docker image with CRaC image included ). See example. To use CRaC with Micronaut, you can add the "crac" feature at Micronaut Launch.
- Basic CRaC support has been a part of Quarkus since 2.10.0. See example.

## Contact

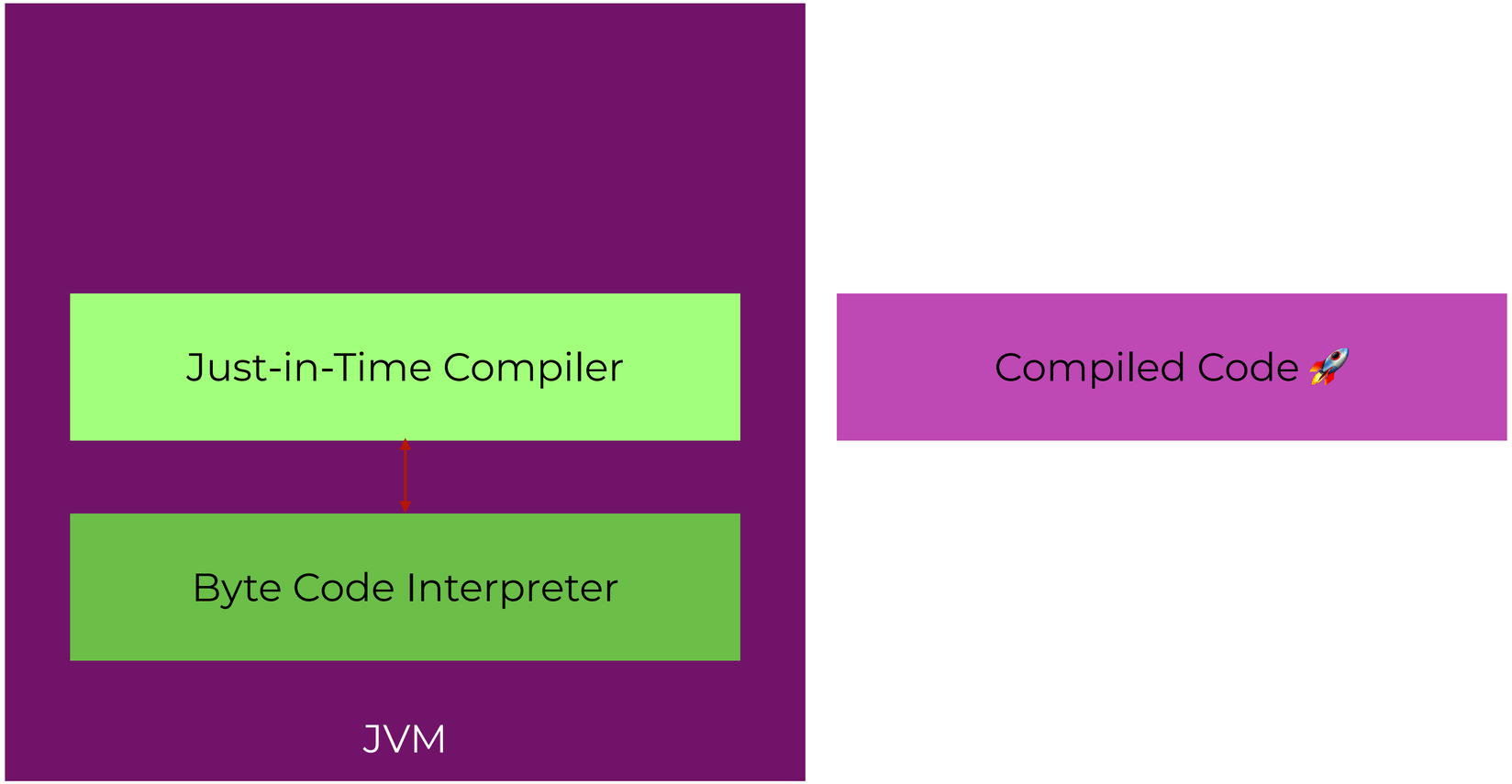
Please post your questions at Foojay.io.

Project CRaC

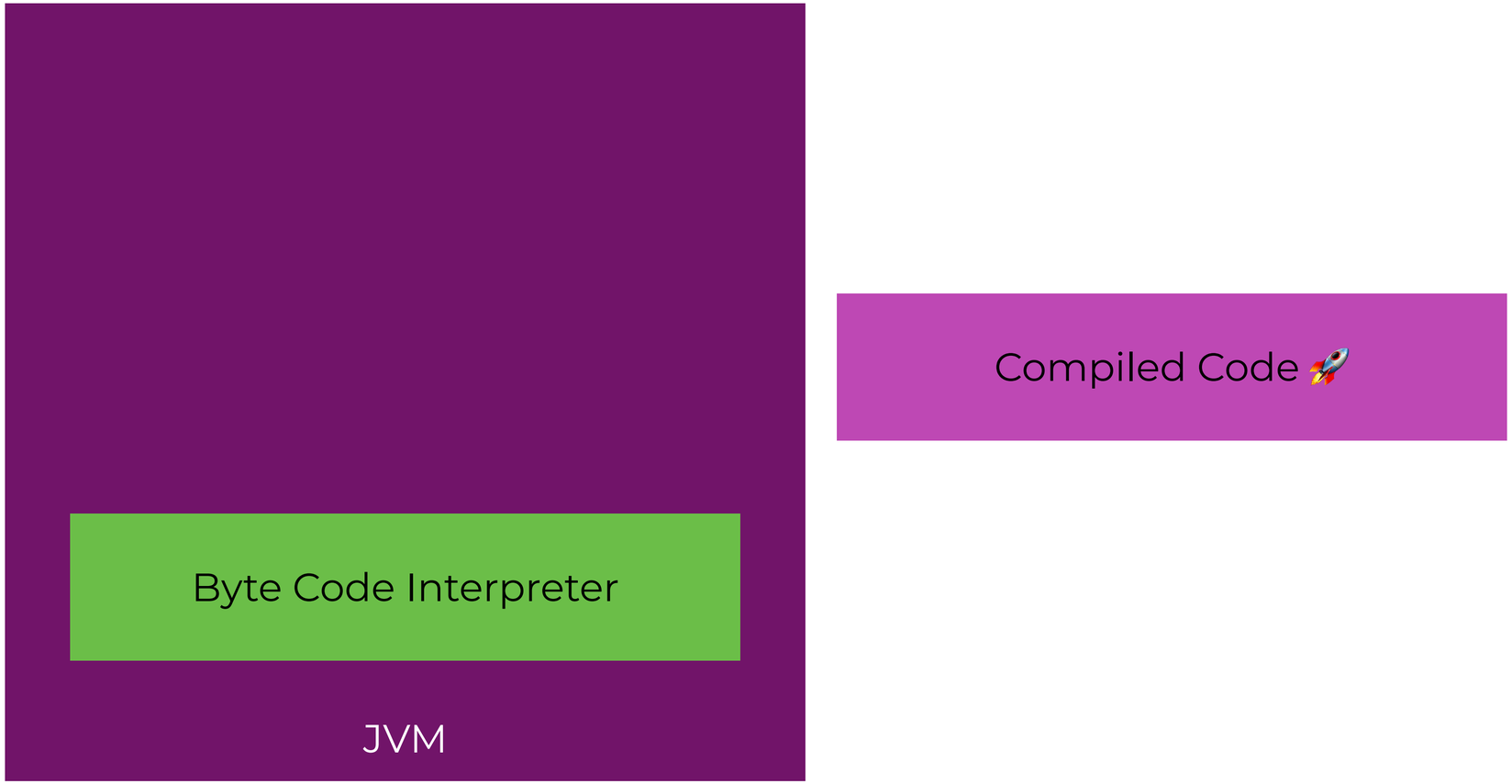
CRIU (stands for Checkpoint and Restore in Userspace)



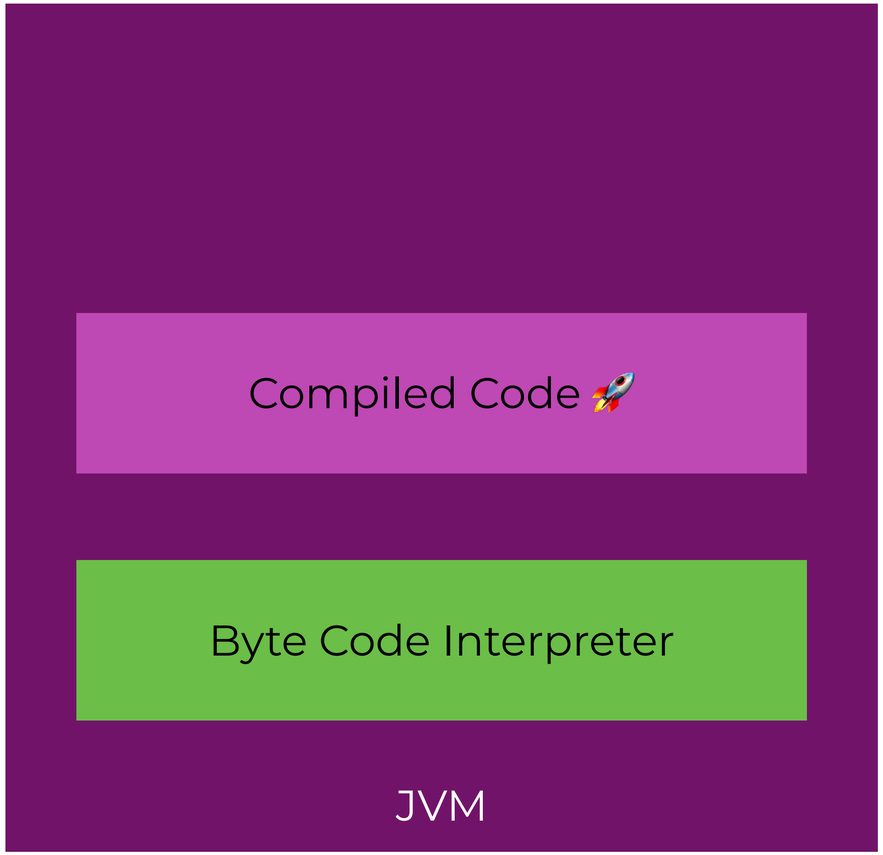

# Project CRaC




# Project CRaC

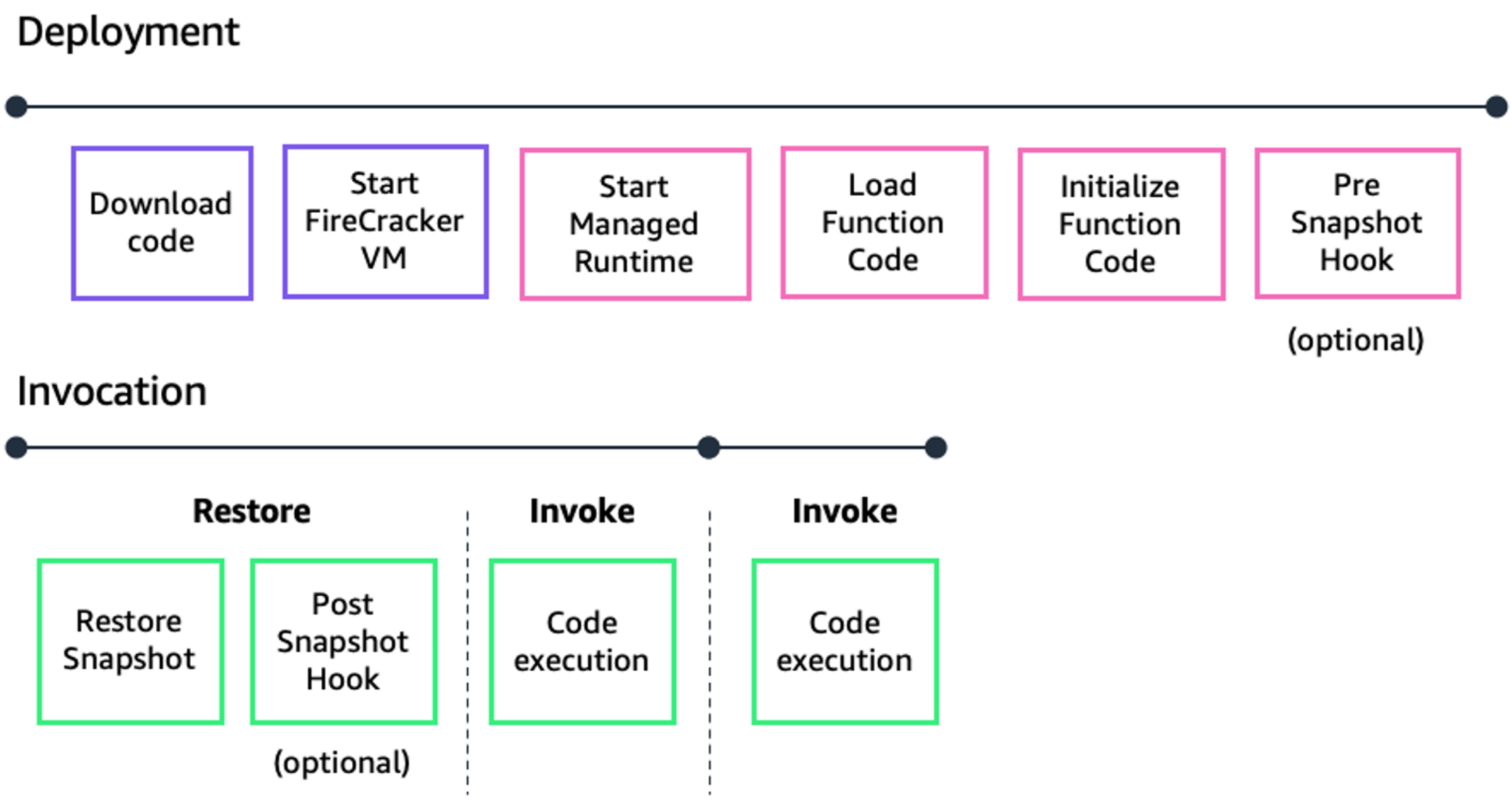



# Project CRaC

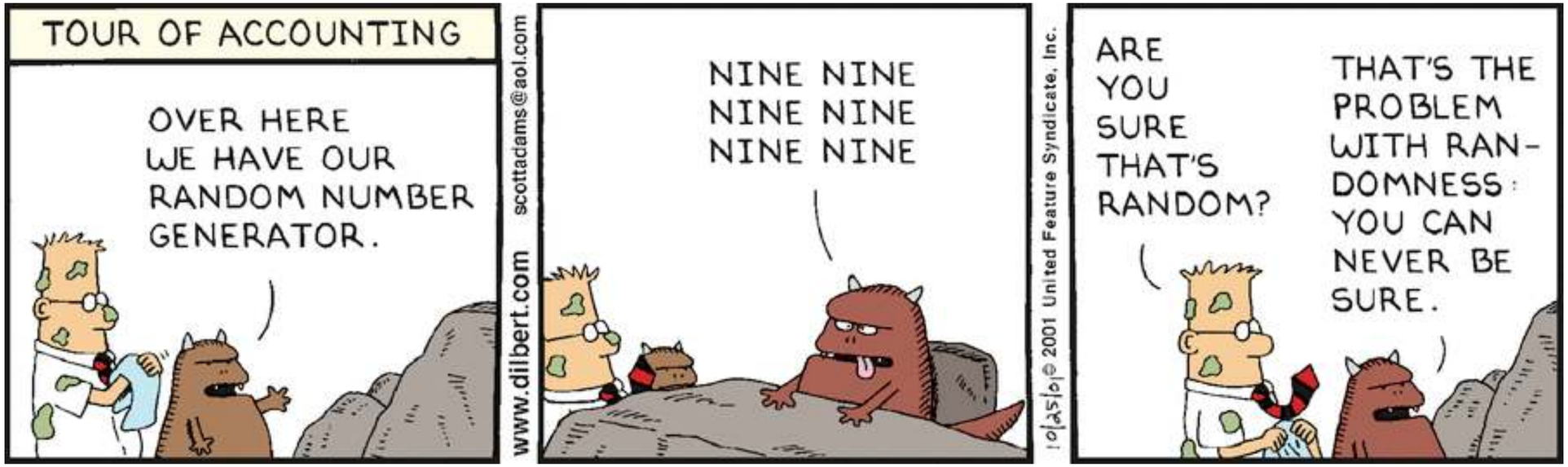



# Project CRaC

# Project CRaC




# Project CRaC




Project CRaC

# Faster startup for containerized applications with Open Liberty InstantOn

Open Liberty InstantOn provides fast startup times for MicroProfile and Jakarta EE applications. With InstantOn, your applications can start in milliseconds, without compromising on throughput, memory, development-production parity, or Java language features.

InstantOn uses the Checkpoint/Restore In Userspace (CRIU) feature of the Linux kernel to take a checkpoint of the JVM that can be restored later. With InstantOn, the application is developed as normal, and then InstantOn makes a checkpoint of the application process when the application container image is built. When the application is restored, it runs in the same JVM, which provides complete parity between development and production. Because the checkpoint process takes only seconds, your CI/CD process is barely affected.

When you build an application container image, InstantOn creates an extra image layer that contains a checkpoint of the Open Liberty application process. When the application image starts, the Open Liberty runtime recognizes the InstantOn layer and restores the application process to the checkpoint that was created during the application container image build.

Project CRaC

Kec blog

About

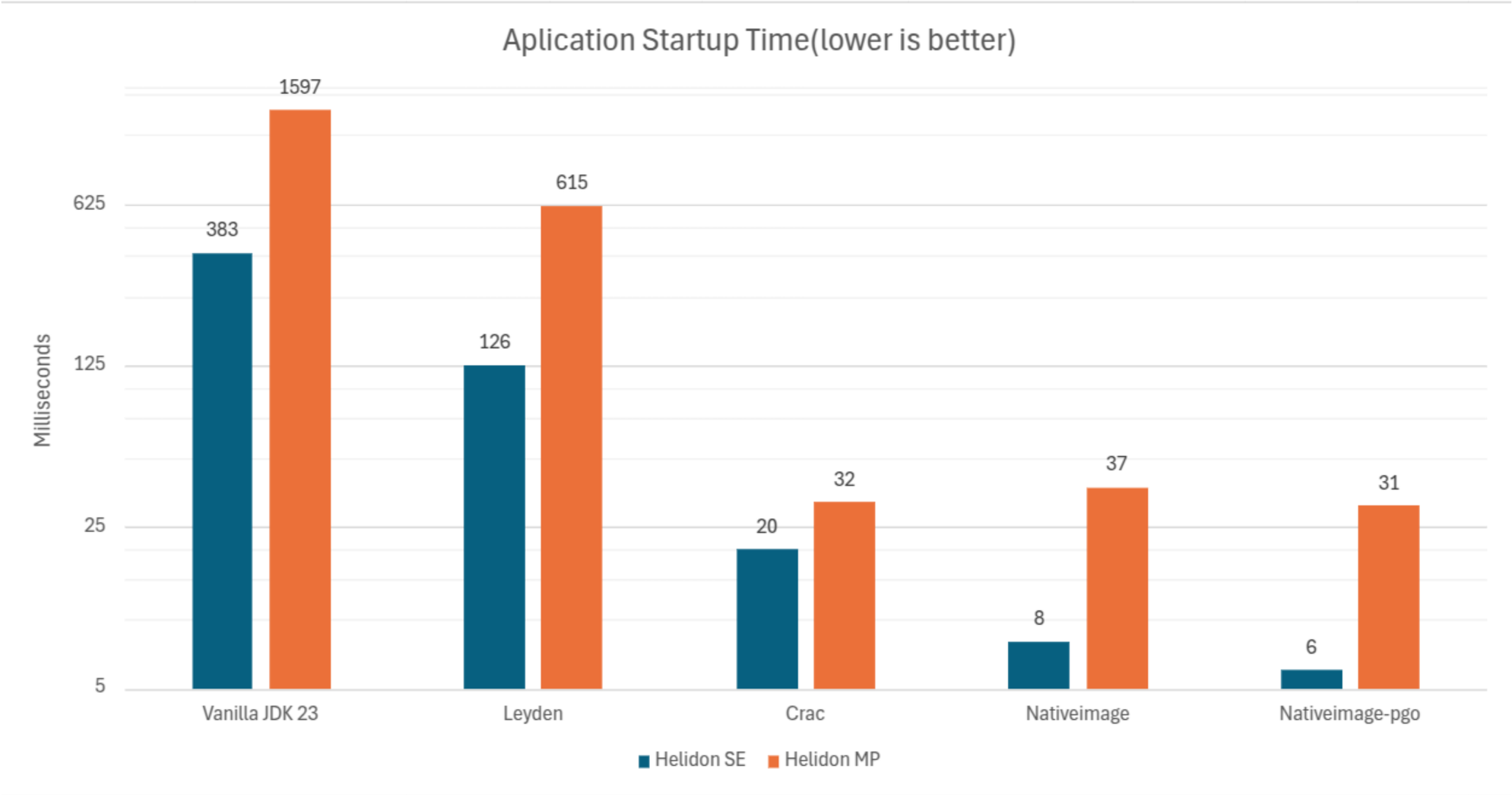
# Helidon and AOT startup optimization benchmark

Mar 7, 2025

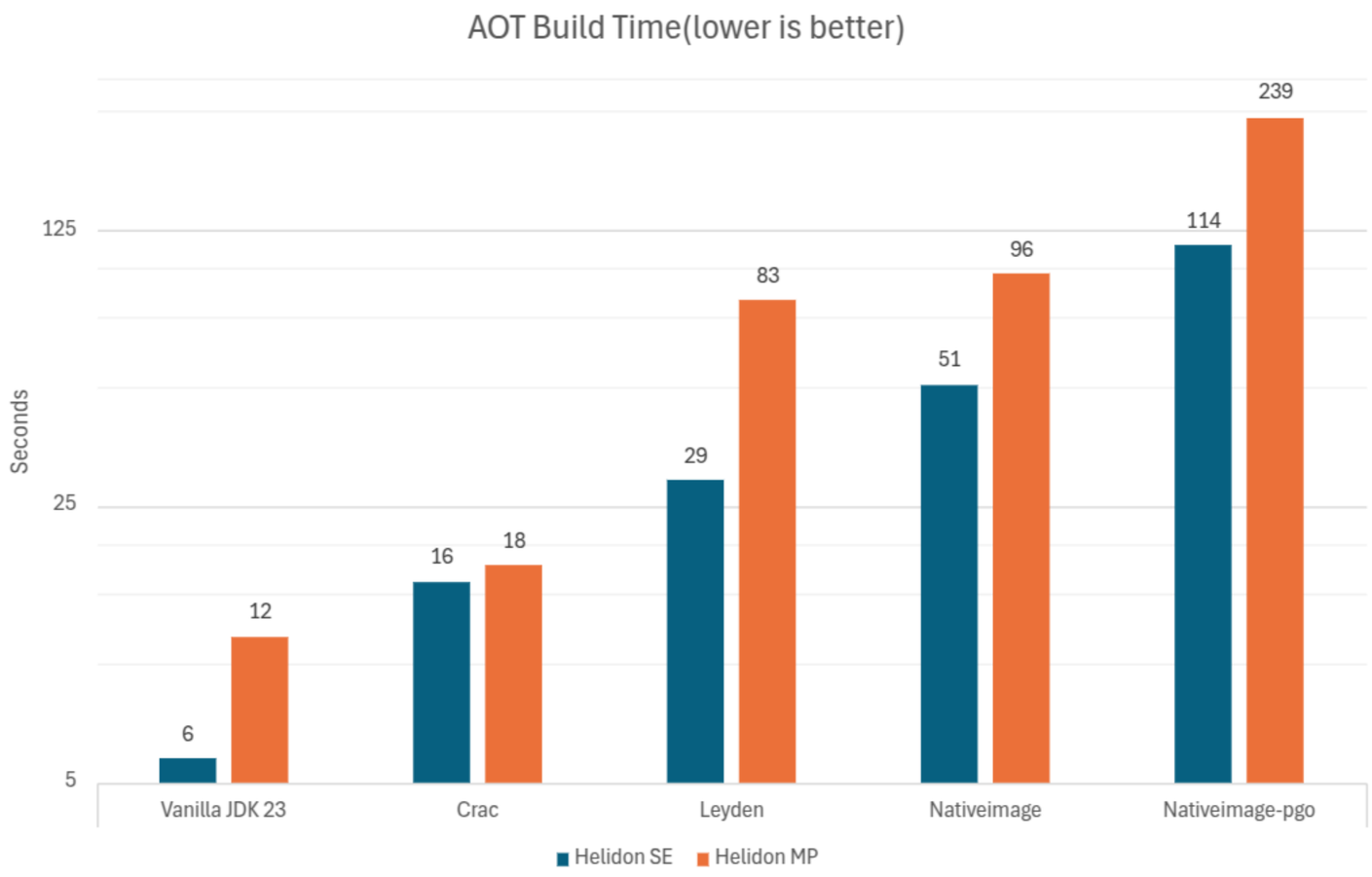
Helidon and AOT startup optimization benchmark

*Leyden vs. NativeImage vs. CRaC, which makes Helidon start faster?*

Helidon and AOT startup optimization benchmark by Daniel Kec




# Project CRaC




# Project CRaC

## GraalVM Native Image

- **Fastest cold start & tiniest image:** produces a self-contained binary (no JDK layer) that launches in single-digit ms - ideal for serverless, scale-to-zero, and extremely high churn micro-pods.
- **PGO variant beats all others** in both startup and immediate-throughput but demands two compilation passes and warm-up profiling.
- **Trade-offs:** longest build times, strict reflection/resource configuration, no JIT (throughput can fall behind vanilla JDK on longer runs).
- **Best fit:** latency-critical functions, memory-constrained environments, or where shipping a JDK is undesirable - and you can invest time in native-image tuning.

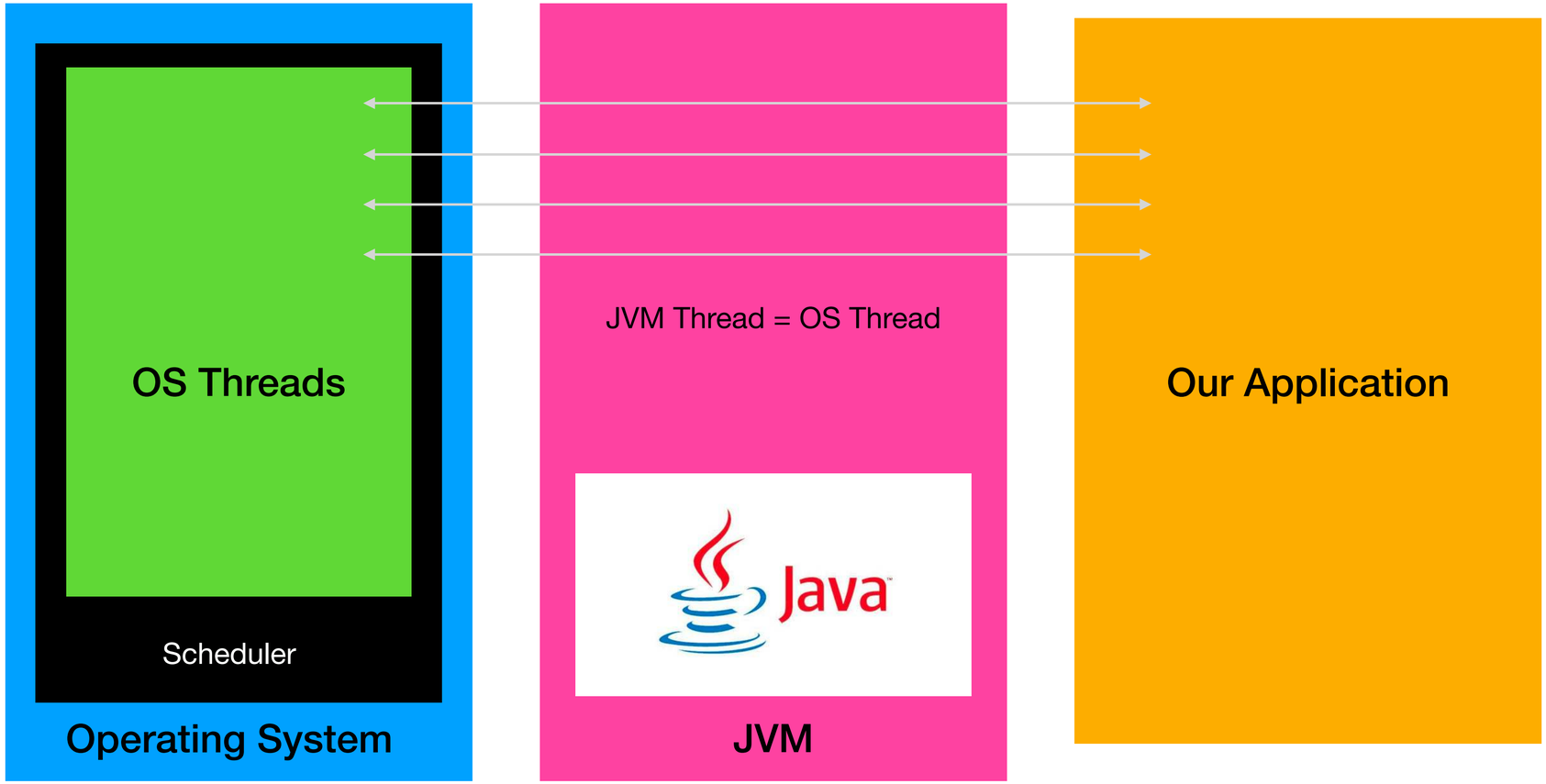
## Leyden (CDS & future JEP 483 AOT)

- **Plug-and-play:** no code or framework tweaks; works with plain app just by running on a Leyden-enabled JDK.
- **Good all-rounder:** roughly halves startup time and trims warm-up cost while preserving full JIT performance for long-running services.
- **Best fit:** teams that want faster scale-up without changing build pipelines or wrestling with reflection/resource configs, and are happy to carry a standard JDK in the container.

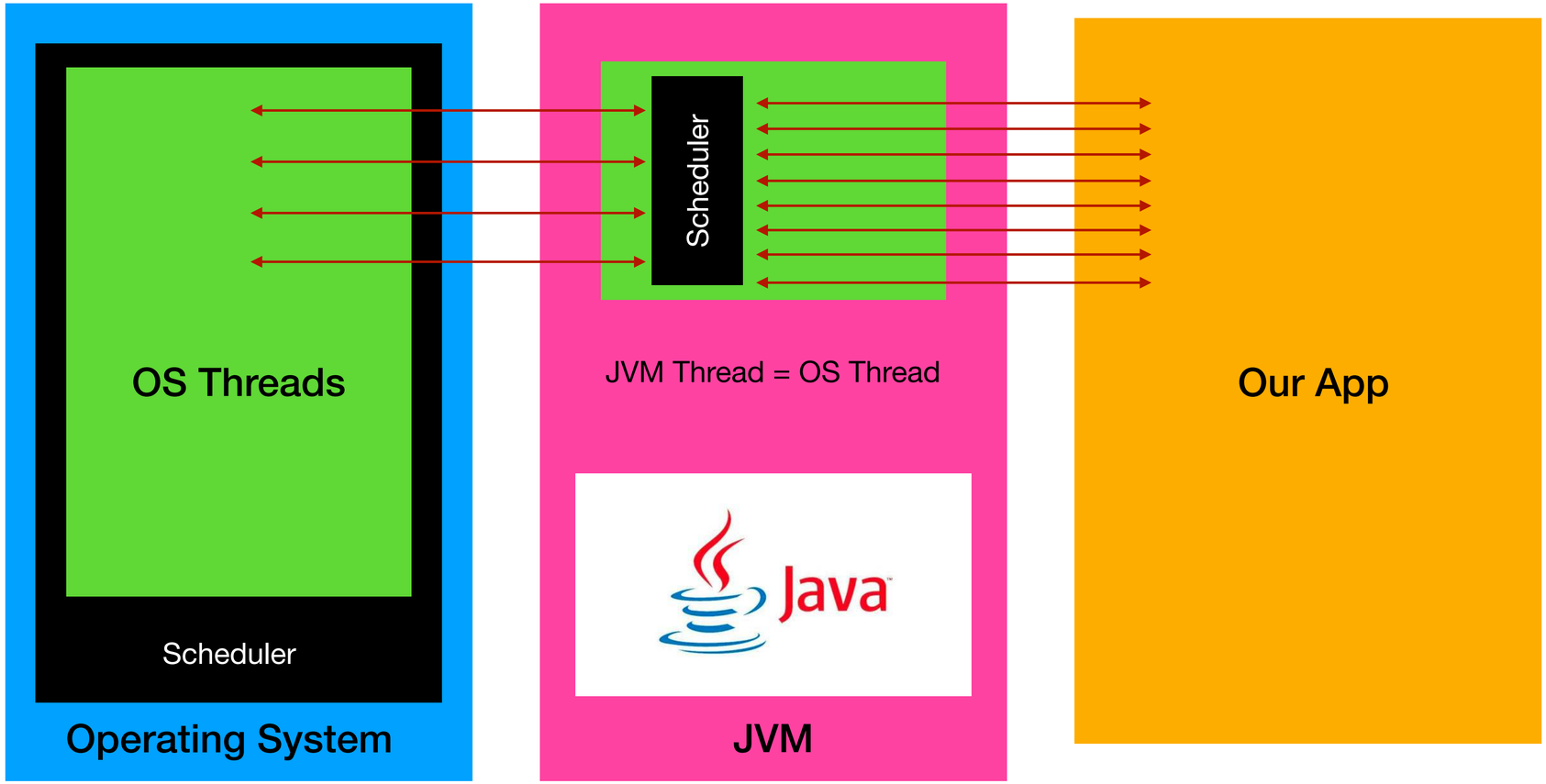
## CRaC (Coordinated Restore at Checkpoint)

- **Snapshot magic:** start a full JDK, warm it up, then checkpoint; restored image pops up in ~20 ms while retaining normal JIT throughput.
- **Fastest AOT build cycle:** snapshotting is much quicker than full native compilation;
- **Caveats:** app must close file handles/sockets before checkpoint; still needs a JDK in the runtime image.
- **Best fit:** services that benefit from both near-instant startup and long-run JIT speed - e.g., JVM-heavy apps that scale frequently under load spikes.

# Computation




# Virtual Threads


OS Threads
Scheduler
Operating System
Scheduler
JVM Thread = OS Thread
Java
JVM
Our App

# Results

### Support for Virtual Threads

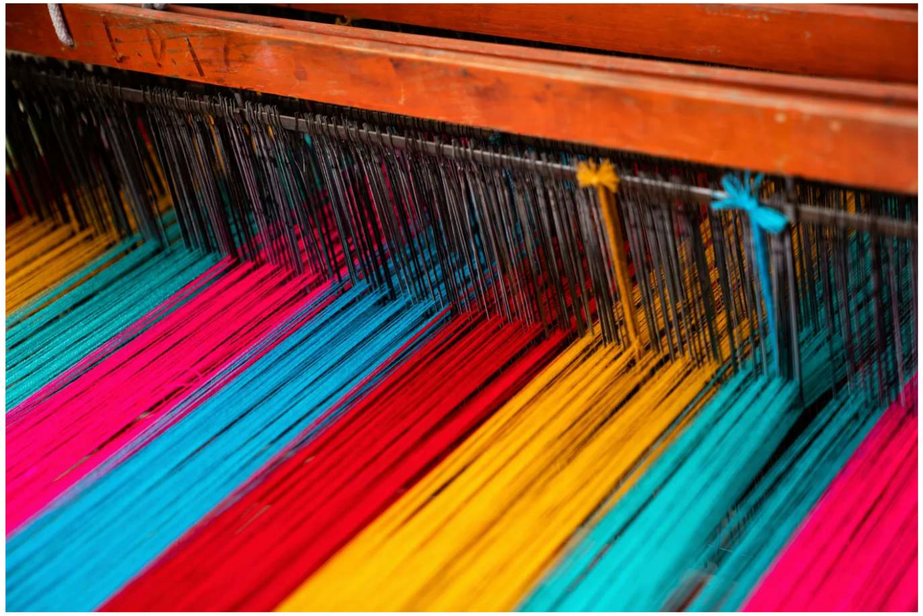
Spring Boot 3.2 ships support for virtual threads. To use virtual threads, you need to run on Java 21 and set the property `spring.threads.virtual.enabled` to `true`.

# Virtual Threads

# Java 21 Virtual threads — The Hype is real?

Ajinkya Vaze Follow 4 min read · Mar 14, 2024

135 3



# I/O-bound workloads

## Java 21 Virtual threads — The Hype is real?

The performance testing model is simple.

Run a spring boot application with embedded tomcat which has a **slow HTTP GET endpoint** and **takes 4 seconds to respond**.

Client (Gatling) hits the endpoint at a constant rate of **400 requests/second for 1 minute**.

The more requests application can process successfully, better the performance.

| | Total Number of requests | Success responses | Throughput (Based on success responses) | Max Number of threads (JVM) | Total Memory utilisation | Heap memory utilisation | Thread memory utilisation | Error % |
|---|---|---|---|---|---|---|---|---|
| Default thread pool (no virtual threads) | 24,000 | 3200 | 53.33 req/sec | 222 | 968 mb | 323 mb | 501 mb | 86.77% |
| With virtual threads | 24,000 | 24,000 | 400 req/sec | 58 | 1040 mb | 774 mb | 117 mb | 0% |
| **Change** | **0** | **+20800** | **+347 req/sec** | **-164** | **+72 mb** | **+451 mb** | **-384 mb** | **-86.77%** |

# Numbers

| | Total Number of requests | Success responses | Throughput (Based on success responses) | Max Number of threads (JVM) | Total Memory utilisation | Heap memory utilisation | Thread memory utilisation | Error % |
|---|---|---|---|---|---|---|---|---|
| Default thread pool (no virtual threads) | 24,000 | 3200 | 53.33 req/sec | 222 | 968 mb | 323 mb | 501 mb | 86.77% |
| With virtual threads | 24,000 | 24,000 | 400 req/sec | 58 | 1040 mb | 774 mb | 117 mb | 0% |
| **Change** | **0** | **+20800** | **+347 req/sec** | **-164** | **+72 mb** | **+451 mb** | **-384 mb** | **-86.77%** |

With the virtual threads, an application was able to process **347** more requests per second...

# Numbers

| | Total Number of requests | Success responses | Throughput (Based on success responses) | Max Number of threads (JVM) | Total Memory utilisation | Heap memory utilisation | Thread memory utilisation | Error % |
|---|---|---|---|---|---|---|---|---|
| Default thread pool (no virtual threads) | 24,000 | 3200 | 53.33 req/sec | 222 | 968 mb | 323 mb | 501 mb | 86.77% |
| With virtual threads | 24,000 | 24,000 | 400 req/sec | 58 | 1040 mb | 774 mb | 117 mb | 0% |
| **Change** | **0** | **+20800** | **+347 req/sec** | **-164** | **+72 mb** | **+451 mb** | **-384 mb** | **-86.77%** |

With the virtual threads, an application was able to process **347** more requests per second...

...and memory utilisation by threads is **384mb** less as compared to the application not using virtual threads.

# Numbers

| | Total Number of requests | Success responses | Throughput (Based on success responses) | Max Number of threads (JVM) | Total Memory utilisation | Heap memory utilisation | Thread memory utilisation | Error % |
|---|---|---|---|---|---|---|---|---|
| Default thread pool (no virtual threads) | 24,000 | 3200 | 53.33 req/sec | 222 | 968 mb | 323 mb | 501 mb | 86.77% |
| With virtual threads | 24,000 | 24,000 | 400 req/sec | 58 | 1040 mb | 774 mb | 117 mb | 0% |
| **Change** | **0** | **+20800** | **+347 req/sec** | **-164** | **+72 mb** | **+451 mb** | **-384 mb** | **-86.77%** |

The application was serving a significantly a greater number of requests (347 req/sec more) which means more objects on the heap to support these requests.

# Numbers

# Java Virtual Threads: a Case Study

JUL 11, 2024 • 19 MIN READ

LIKE

by

Gary DeVal FOLLOW

WebSphere/Java Performance Engineer

Vijay Sundaresan FOLLOW

Senior Technical Staff Member and Performance Architect

AND 3 MORE

**Write for InfoQ**

**Feed your curiosity.**

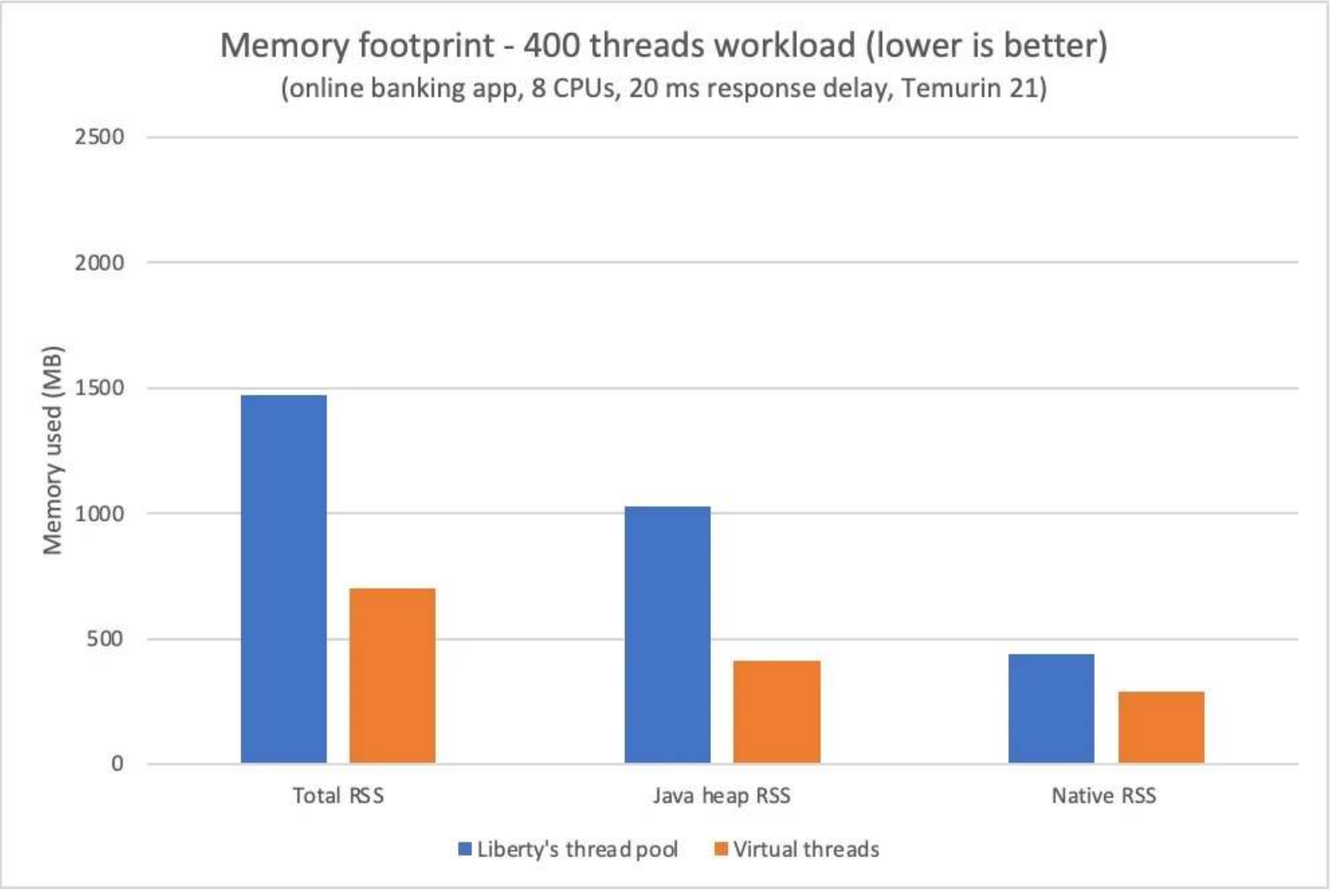
Help 550k+ global senior developers each month stay ahead.

Get in touch

## Key Takeaways

- Virtual threads are an important advancement in Java concurrent programming, but they do not offer a clear advantage over Open Liberty's existing autonomic thread pool for running typical cloud-native Java workloads.
- For CPU-intensive workloads, throughput is lower with virtual threads than with Open Liberty's thread pool for currently unclear reasons.
- Virtual thread ramp-up time from idle to maximum throughput is quicker than Open Liberty's thread pool due to its thread-per-request model.
- Memory footprint in Open Liberty deployments can vary greatly based on factors like application design, workload level, and garbage collection behavior, so the reduced footprint of virtual threads may not result in an overall reduction in memory used.
- Virtual threads showed some unexpected performance issues in some use cases that Java developers should be aware of. We are working with the OpenJDK Community to investigate the root cause and to try to resolve the issues.

CPU-bound workloads


Memory footprint - 400 threads workload (lower is better)
(online banking app, 8 CPUs, 20 ms response delay, Temurin 21)
Memory used (MB)
2500
2000
1500
1000
500
0
Total RSS
Java heap RSS
Native RSS
Liberty's thread pool
Virtual threads

# Numbers

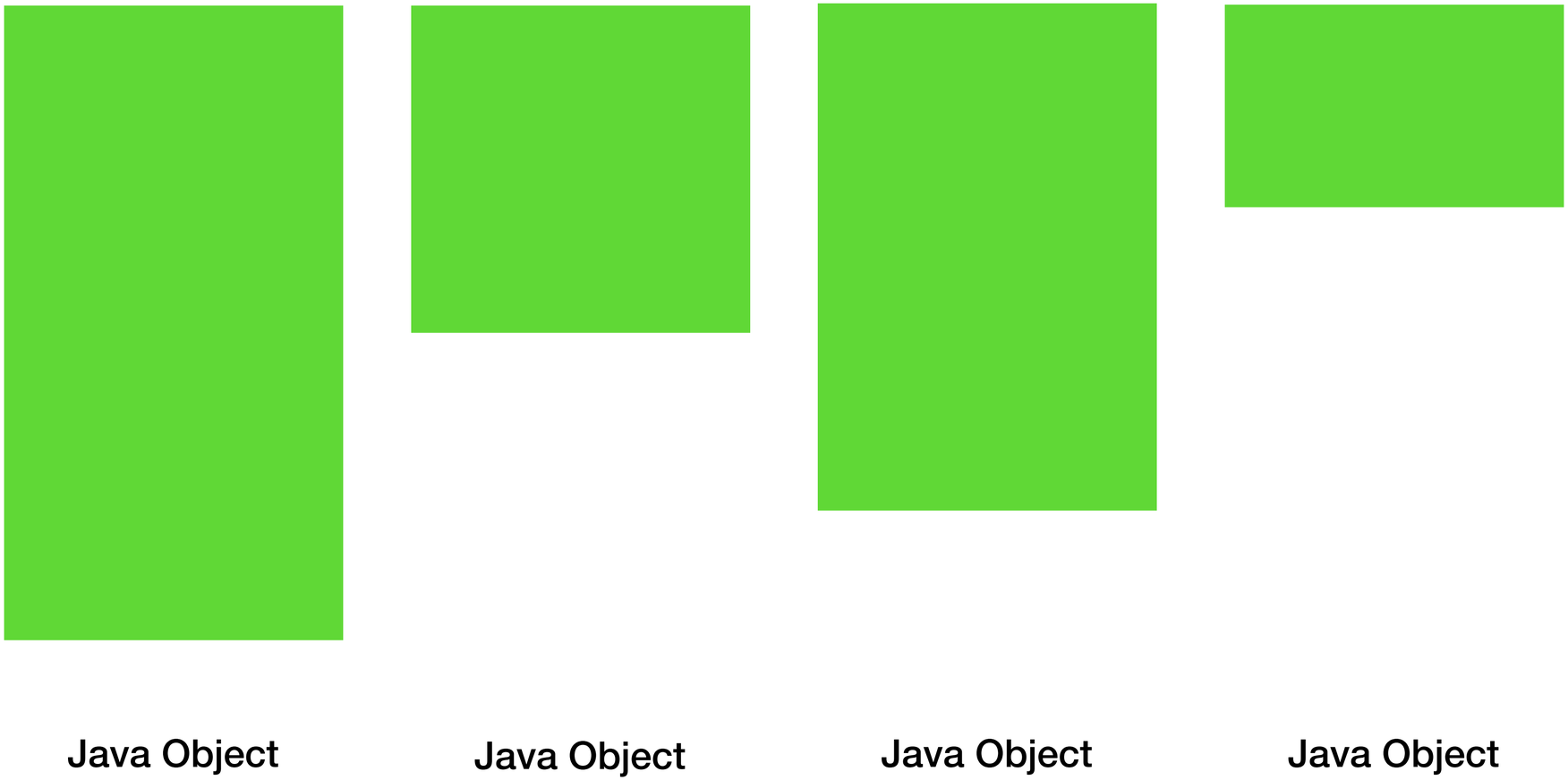
## VTs won’t speed up pure compute; they help when blocking

- Use VTs for high-latency, blocking I/O (JDBC, HTTP, gRPC)… especially when connection limits or thread explosions were the previous bottleneck.
- Test before using on CPU-heavy loops on bounded pools
- **Watch heap/G1 GC** – millions of VTs move stack frames into the heap

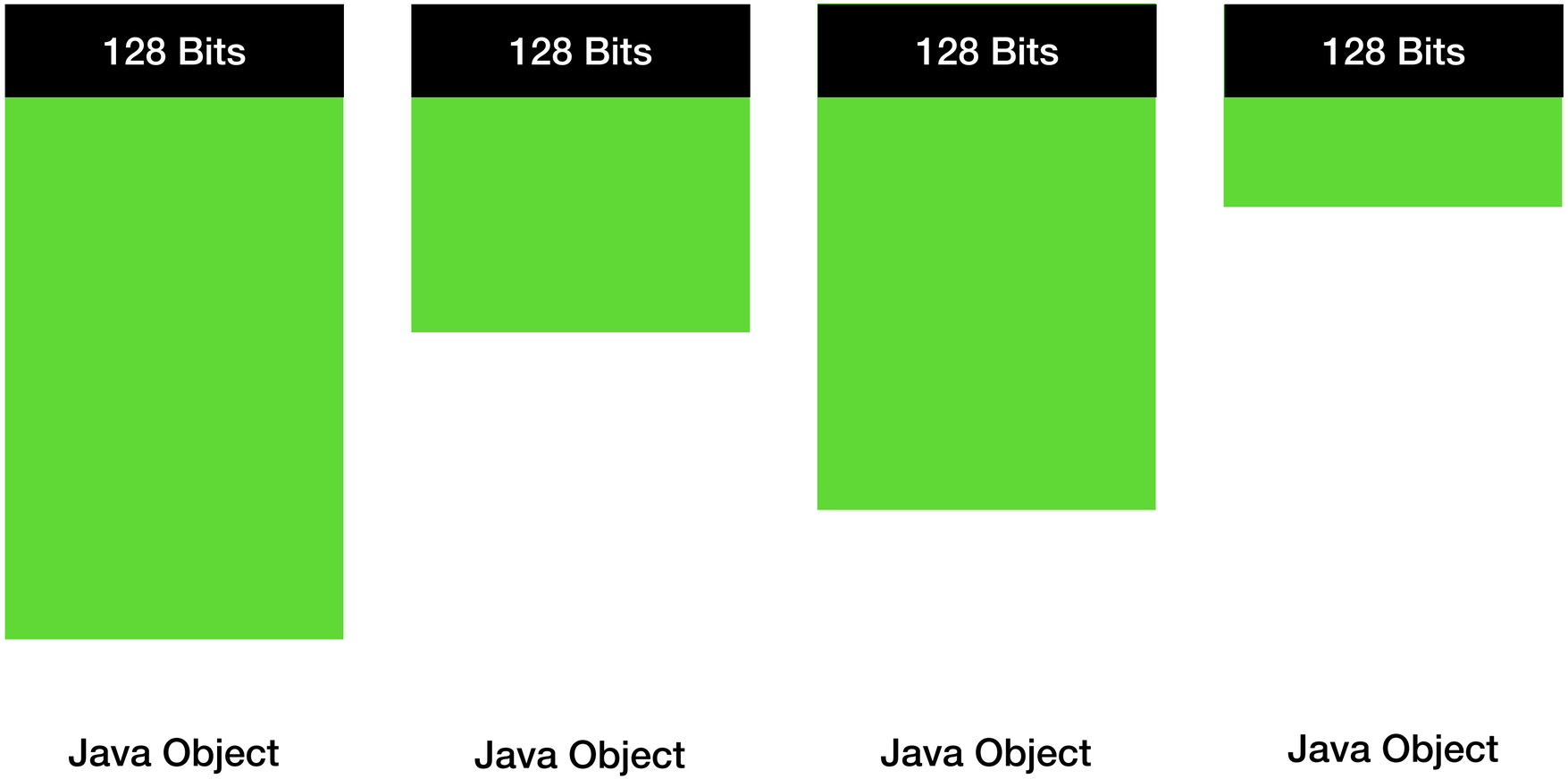
# Heuristics to use

# Memory Management

Garbage Collectors




# Project Liliput




# Project Liliput

64 bytes for
Garbage Collector / Locks / etc

64 bytes for
Class Pointer

Project Liliput

64 bytes for
Garbage Collector / Locks / etc
32 bytes for
Class Pointer

32 Bity dla
Wskaźnika Klasy

32 Bity dla
Garbage Collectora / Locka / Identity

# Nullness markers to enable flattening

**Dan Smith** daniel.smith at oracle.com
*Tue Feb 7 01:26:42 UTC 2023*

- Previous message (by thread): Valhalla super class/interfaces and bridges
- Next message (by thread): Nullness markers to enable flattening
- **Messages sorted by:** [ date ] [ thread ] [ subject ] [ author ]

---

A quick review:

The Value Objects feature (see https://openjdk.org/jeps/8277163) captures the Valhalla project's central idea: that objects don't have to have identity, and if programmers opt out of identity, JVMs can provide optimizations comparable to primitive performance.

However, one important implementation technique is not supported by that JEP: maximally flattened heap storage. ("Maximally flattened" as in "just the bits necessary to encode an instance".) This is because flattened fields and arrays store an object's field values directly, and so 1) need to be initialized "at birth" to a non-null class instance, 2) may not store null, and 3) may by updated non-atomically. These are semantics that need to be surfaced in the language model.

We've tackled (3) by allowing value classes to be declared non-atomic (syntax/limitations subject to bikeshedding), and then claiming by fiat that fields/arrays of such classes are tearing risks. Races are rare enough that this doesn't really call for a use-site opt-in, and we don't necessarily need any deeper explanation for how new objects derived from random combinations of old objects can be created by a read operation. That's just how it works. <shrug>

We also allow value classes to declare that they support an all-zeros default instance (again, subject to bikeshedding). You could imagine similarly claiming that fields/arrays of these classes are null-hostile, as a side effect of how their storage works. But this is an idiosyncrasy that is going to affect a lot more programmers, and "that's just how it works" is pretty unsatisfactory. Sometimes programs count on being able to use 'null' in their computation. We need something in the language model to let programs opt in/out of nulls at the use site, and thus opt out/in of maximally flattenable heap storage.

We've long discussed "reference type" vs. "value type" as the language concept that captures this distinction. But where we once had a long list of differences between references and values, most of those have gone away. Notably, it's *not* useful for performance intuitions to imagine that references are pointers and values are inline. Value objects get inlined when the JVM want to do so. Reference-ness is not relevant.

Really, for most programmers, nullness is all that distinguishes a "reference type" from a "value type".

Meanwhile, expressing nullness is not a problem unique to Valhalla. Whether a variable is meant to store nulls is probably the most important property of most programs that isn't expressible in the language. Workarounds include informal javadoc specifications, type annotations (as explored by JSpecify), lots of 'Objects.requireNonNull' calls, and blanket "if you pass in a null, you might get an NPE" policies.

In Amber, pattern matching has its own problems with nullness: there are a lot of ad hoc rules to distinguish between "is this a non-null instance of class Foo?" vs. "is this null *or* an instance of class Foo?", because there's no good way to express those two queries as explicitly different.

---

To address these problems, we've been exploring nullness markers as an alternative to '.val' and '.ref'. The goal is a general-purpose feature that lets programmers express intent about nulls, and that is preserved at runtime sufficiently for JVMs to observe that "not null" + "value class" + "non-atomic (or compact) class" --> "maximally flattenable storage". There are no "value types", and there is no direct control over flattenability.

(A lot of these ideas build on what JSpecify has done, so appreciation to them for the good work and useful documentation.)

Some key ideas:

Project Liliput

## JEP 519: Compact Object Headers

| | |
|---|---|
| *Owner* | Roman Kennke |
| *Type* | Feature |
| *Scope* | Implementation |
| *Status* | Integrated |
| *Release* | 25 |
| *Component* | hotspot / runtime |
| *Discussion* | hotspot dash dev at openjdk dot org |
| *Effort* | S |
| *Duration* | XS |
| *Relates to* | JEP 450: Compact Object Headers (Experimental) |
| *Reviewed by* | Coleen Phillimore, Stefan Karlsson, Vladimir Kozlov |
| *Endorsed by* | Vladimir Kozlov |
| *Created* | 2025/04/15 14:01 |
| *Updated* | 2025/05/23 09:38 |
| *Issue* | 8354672 |

### Summary

Change compact object headers from an experimental feature to a product feature.

### Non-Goals

It is not a goal to make compact object headers be the default object-header layout.

# Compact Object Headers in Dragonwell JDK: Reducing Costs and Increasing Efficiency for Java Applications

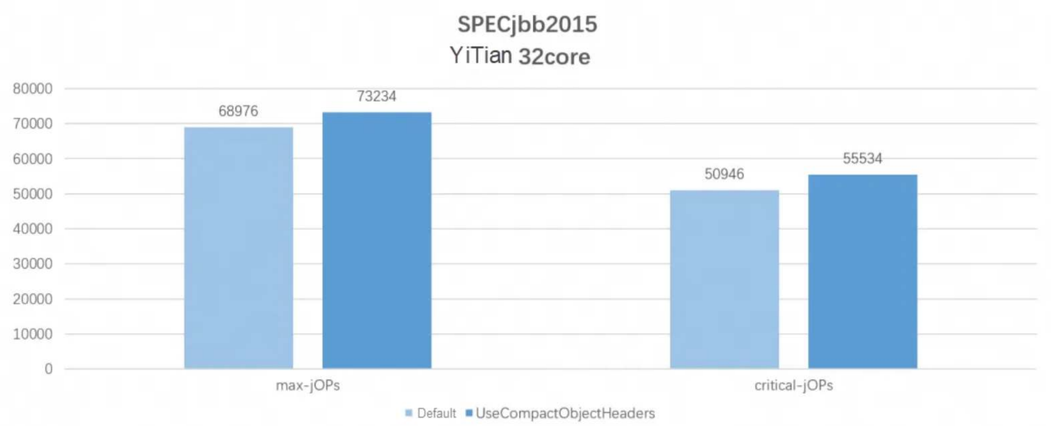
Alibaba Cloud Community April 22, 2024 2,329 0

This article introduces the latest technology of OpenJDK: compact object headers which is now provided by Dragonwell JDK and significantly optimizes the memory usage of Java applications.

*By Liang Mao*

## SPECjbb2015

In the SPECjbb2015 benchmark test on the YiTian platform, max(absolute throughput) and critical (low latency required throughput) are improved by 6.17% and 9.01% respectively.



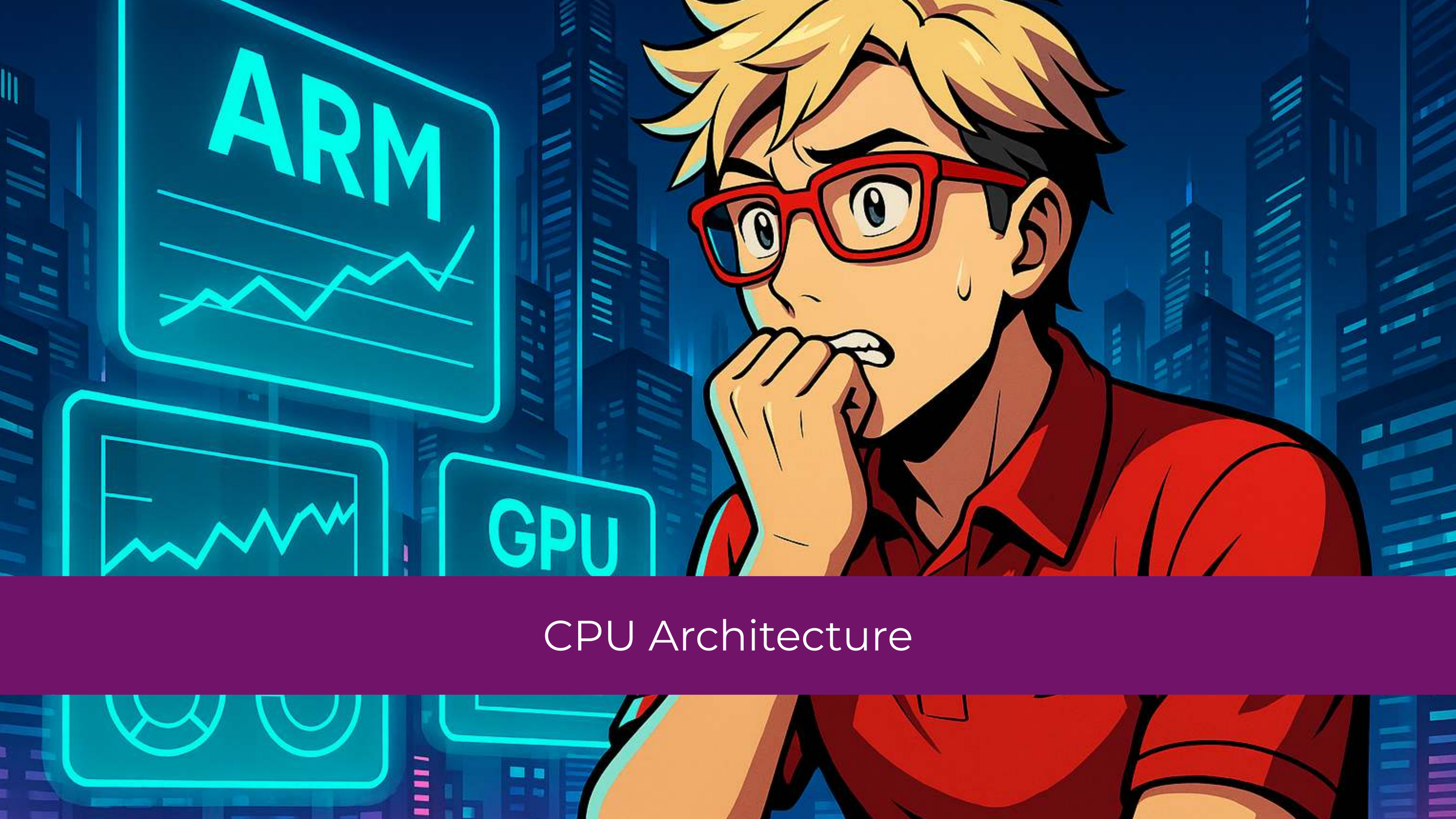

Project Liliput

**Typical heap-size reductions of 5–15 % in real-world apps heavy on small objects** (collections, JSON, reactive pipelines)

- **Memory-pinched deployments:** tight-RAM containers, FaaS, or cloud instances where shaving ~8 B off every object lets you run with a smaller heap or pack more pods per node.
- **Large-object-count services:** event streaming, caching layers, actor systems, high-fan-out microservices - anything allocating millions of tiny objects per second.
- **Skip it** if memory isn't your bottleneck or heap is dominated by large arrays/off-heap buffers

# Heuristics to use

Project Valhalla




# CPU Architecture

Trying to save on AWS? Click here to get a demo of Vantage from a FinOps expert.

Region: US East (N. Virginia) | Pricing Unit: Instance | Cost: Hourly | Reserved: 1-year - No Upfront | Visible: Columns | Compare | Clear Filters | Export | Search...

| Name | API Name | Instance Memory | vCPUs | Instance Storage | Network Performance | On Demand | Linux Reserved cos |
|---|---|---|---|---|---|---|---|
| Filter... | Filter... | Min Mem: 0 | Min vCPUs: 0 | Min Storage: 0 | Filter... | Filter... | Filter... |
| T4G Nano | t4g.nano | 0.5 GiB | 2 vCPUs for a 1h 12m burst | EBS only | Up to 5 Gigabit | $0.0042 hourly | $0.0026 ho |
| T3A Nano | t3a.nano | 0.5 GiB | 2 vCPUs for a 1h 12m burst | EBS only | Up to 5 Gigabit | $0.0047 hourly | $0.0029 ho |
| T3 Nano | t3.nano | 0.5 GiB | 2 vCPUs for a 1h 12m burst | EBS only | Up to 5 Gigabit | $0.0052 hourly | $0.0033 ho |
| T2 Nano | t2.nano | 0.5 GiB | 1 vCPUs for a 1h 12m burst | EBS only | Low to Moderate | $0.0058 hourly | $0.0036 ho |
| T4G Micro | t4g.micro | 1.0 GiB | 2 vCPUs for a 2h 24m burst | EBS only | Up to 5 Gigabit | $0.0084 hourly | $0.0053 ho |
| T3A Micro | t3a.micro | 1.0 GiB | 2 vCPUs for a 2h 24m burst | EBS only | Up to 5 Gigabit | $0.0094 hourly | $0.0059 ho |
| T3 Micro | t3.micro | 1.0 GiB | 2 vCPUs for a 2h 24m burst | EBS only | Up to 5 Gigabit | $0.0104 hourly | $0.0065 ho |
| T2 Micro | t2.micro | 1.0 GiB | 1 vCPUs for a 2h 24m burst | EBS only | Low to Moderate | $0.0116 hourly | $0.0072 ho |
| T4G Small | t4g.small | 2.0 GiB | 2 vCPUs for a 4h 48m burst | EBS only | Up to 5 Gigabit | $0.0168 hourly | $0.0105 ho |
| T3A Small | t3a.small | 2.0 GiB | 2 vCPUs for a 4h 48m burst | EBS only | Up to 5 Gigabit | $0.0188 hourly | $0.0118 ho |
| T1 Micro | t1.micro | 0.613 GiB | 1 vCPUs | EBS only | Very Low | $0.0200 hourly | $0.0140 ho |
| T3 Small | t3.small | 2.0 GiB | 2 vCPUs for a 4h 48m burst | EBS only | Up to 5 Gigabit | $0.0208 hourly | $0.0130 ho |
| T2 Small | t2.small | 2.0 GiB | 1 vCPUs for a 4h 48m burst | EBS only | Low to Moderate | $0.0230 hourly | $0.0144 ho |
| A1 Medium | a1.medium | 2.0 GiB | 1 vCPUs | EBS only | Up to 10 Gigabit | $0.0255 hourly | $0.0161 ho |
| T4G Medium | t4g.medium | 4.0 GiB | 2 vCPUs for a 4h 48m burst | EBS only | Up to 5 Gigabit | $0.0336 hourly | $0.0211 ho |
| C6G Medium | c6g.medium | 2.0 GiB | 1 vCPUs | EBS only | Up to 10 Gigabit | $0.0340 hourly | $0.0214 ho |
| C7G Medium | c7g.medium | 2.0 GiB | 1 vCPUs | EBS only | Up to 12.5 Gigabit | $0.0363 hourly | $0.0239 ho |
| T3A Medium | t3a.medium | 4.0 GiB | 2 vCPUs for a 4h 48m burst | EBS only | Up to 5 Gigabit | $0.0376 hourly | $0.0236 ho |

Trying to save on AWS? Click here to get a demo of Vantage from a FinOps expert.

Region: US East (N. Virginia) | Pricing Unit: Instance | Cost: Hourly | Reserved: 1-year - No Upfront | Visible: Columns | Compare | Clear Filters | Export | Search...

- 1-year - No Upfront
- 1-year - Partial Upfront
- 1-year - Full Upfront
- 3-year - No Upfront
- 3-year - Partial Upfront
- 3-year - Full Upfront
- 1-year convertible - No Upfront
- 1-year convertible - Partial Upfront
- 1-year convertible - Full Upfront
- 3-year convertible - No Upfront
- 3-year convertible - Partial Upfront
- 3-year convertible - Full Upfront

| Name | | | vCPUs | Instance Storage | Network Performance | On Demand | Linux Reserved cos |
|---|---|---|---|---|---|---|---|
| Filter... | | | Min vCPUs: 0 | Min Storage: 0 | Filter... | Filter... | Filter... |
| U-12TB1 Metal | | | 448 vCPUs | EBS only | 100 Gigabit | unavailable | unavail |
| HPC6ID 32xlarge | | | 64 vCPUs | EBS only | 200 Gigabit | unavailable | unavail |
| HPC7A 12xlarge | | | 24 vCPUs | EBS only | 300 Gigabit | unavailable | unavail |
| HPC6A 48xlarge | | | 96 vCPUs | EBS only | 100 Gigabit | unavailable | unavail |
| HPC7A 48xlarge | | | 96 vCPUs | EBS only | 300 Gigabit | unavailable | unavail |
| TRN2 48xlarge | | | 192 vCPUs | EBS only | 3200 Gigabit | unavailable | unavail |
| U-9TB1 Metal | | | 448 vCPUs | EBS only | 100 Gigabit | unavailable | unavail |
| U-6TB1 Metal | | | 448 vCPUs | EBS only | 100 Gigabit | unavailable | unavail |
| I2 Large | i2.large | 15.0 GiB | 2 vCPUs | EBS only | Moderate | unavailable | unavail |
| MAC1 Metal | mac1.metal | 32.0 GiB | 12 vCPUs | EBS only | 25 Gigabit | unavailable | unavail |
| MAC2-M2 Metal | mac2-m2.metal | 24.0 GiB | 8 vCPUs | EBS only | 10 Gigabit | unavailable | unavail |
| MAC2 Metal | mac2.metal | 16.0 GiB | 8 vCPUs | EBS only | 10 Gigabit | unavailable | unavail |
| MAC2-M2PRO Metal | mac2-m2pro.metal | 32.0 GiB | 12 vCPUs | EBS only | 10 Gigabit | unavailable | unavail |
| U-18TB1 Metal | u-18tb1.metal | 18432.0 GiB | 448 vCPUs | EBS only | 100 Gigabit | unavailable | unavail |
| MAC2-M1ULTRA Metal | mac2-m1ultra.metal | 128.0 GiB | 20 vCPUs | EBS only | 10 Gigabit | unavailable | unavail |
| U-24TB1 Metal | u-24tb1.metal | 24576.0 GiB | 448 vCPUs | EBS only | 100 Gigabit | unavailable | unavail |
| High Storage Eight Extra Large | hs1.8xlarge | 117.0 GiB | 16 vCPUs | EBS only | 10 Gigabit | unavailable | unavail |
| Cluster Compute Eight Extra Large | cc2.8xlarge | 60.5 GiB | 32 vCPUs | EBS only | 10 Gigabit | unavailable | unavail |

# Has x86 lost the data center battle? Arm claims victory as it declares close to 50 percent of compute shipped to top hyperscalers in 2025 will be Arm-based

News By Wayne Williams published April 5, 2025

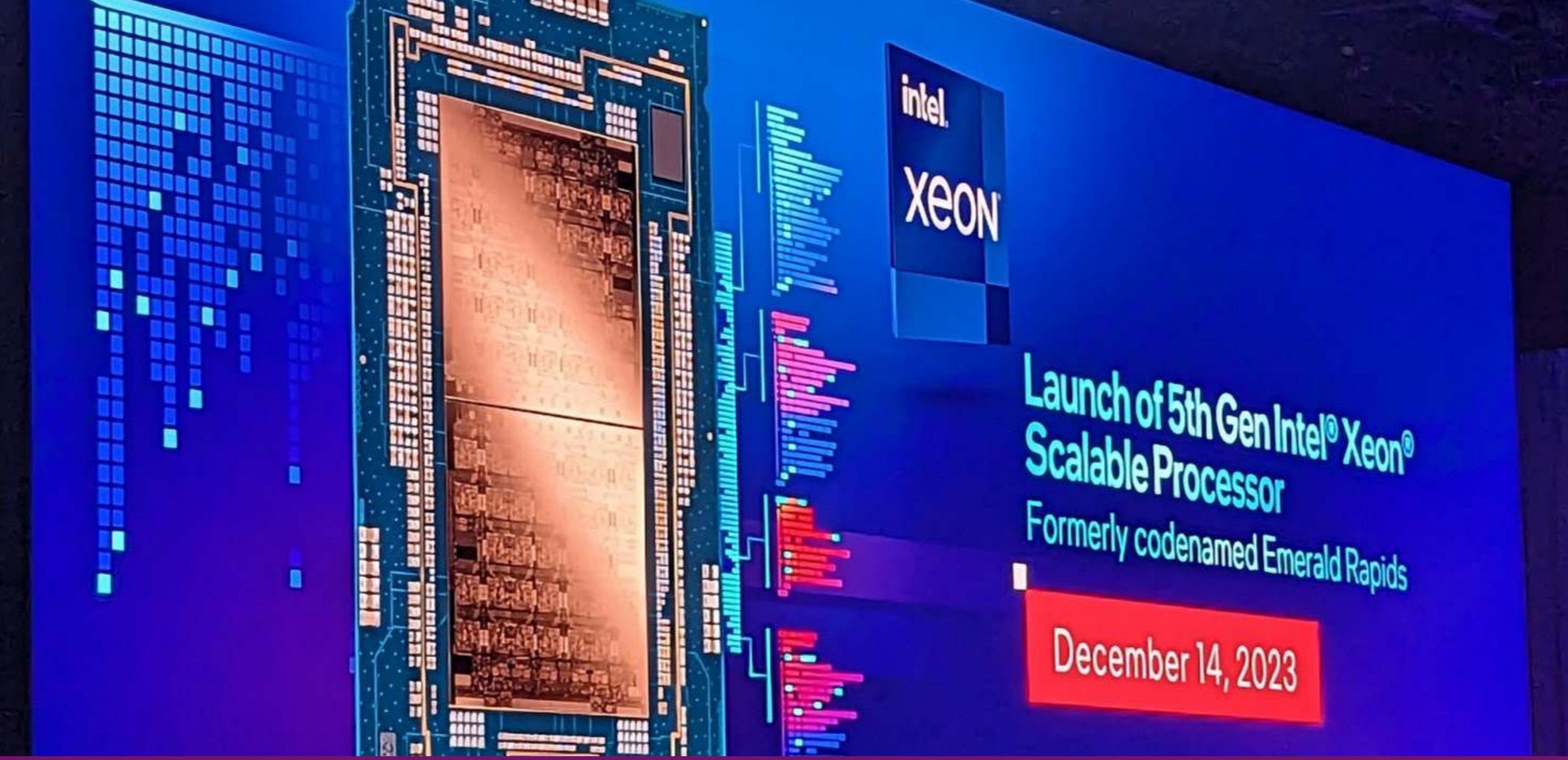
AI demand is behind the Neoverse architecture's meteoric rise

Comments (1)

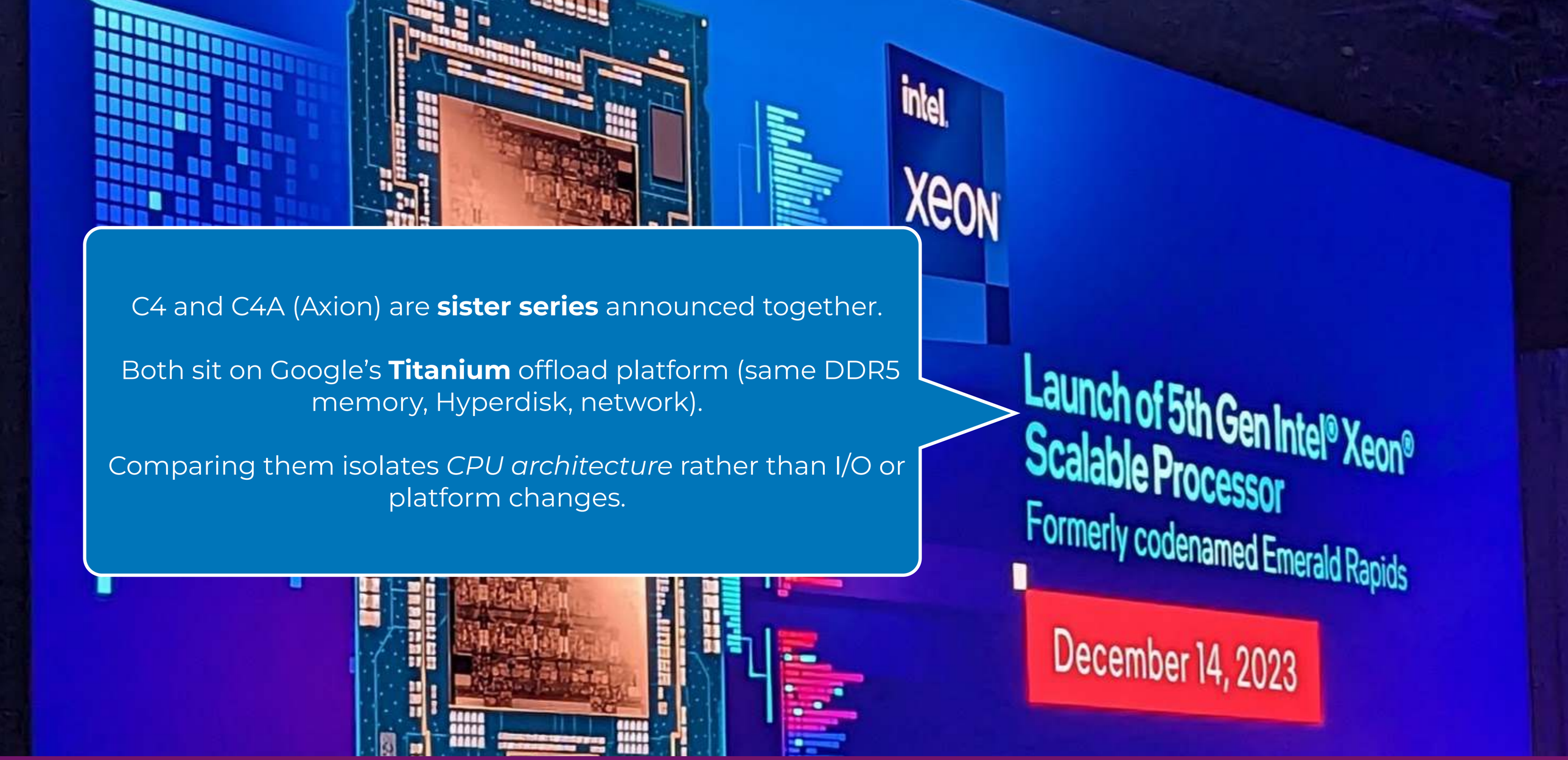
When you purchase through links on our site, we may earn an affiliate commission. Here's how it works.

(Image credit: Shutterstock / gguy)

- Arm says Neoverse powers nearly half of 2025 hyperscaler compute shipments
- AI growth makes power-efficiency essential for hyperscale data centers
- AWS, Google, Microsoft are leading shift from x86 to Arm infrastructure




# Axion vs Xeon vs T2A




C4 and C4A (Axion) are **sister series** announced together.

Both sit on Google's **Titanium** offload platform (same DDR5 memory, Hyperdisk, network).

Comparing them isolates *CPU architecture* rather than I/O or platform changes.

# Axion vs Xeon vs T2A

phoronix

ARTICLES & REVIEWS | NEWS ARCHIVE | FORUMS | PREMIUM | CONTACT | CATEGORIES

ENHANCED BY Google Search

# Latest Linux Hardware Reviews, Open-Source News & Benchmarks

## FEX 2505 Released With Many Fixes For Running x86_64 Binaries On Linux AArch64

7 Hours Ago - Free Software - FEX 2505 - 2 Comments

FEX 2505 is out today as the newest version of this user-space software that enables running x86_64 binaries on ARM64/AArch64 Linux hosts with great speed that a number of games and applications are enjoying much success with this emulator.

## Redox OS Seeing More Software Porting, Many Internal Improvements

10 Hours Ago - Operating Systems - Redox OS - 9 Comments

The open-source, Rust-based Redox OS operating system has published their April 2025 status report to outline interesting changes made over the past several weeks to this innovative project.

## Featured 21-Way Intel Core / AMD Ryzen Linux Laptop Comparison On Ubuntu 25.04

5 May 10:32 AM EDT - Computers - 10 Comments

As part of fresh re-testing of existing laptops on-hand given the recent release of Ubuntu 25.04 and then also recent Linux reviews of some interesting models like the Framework Laptop 13 with AMD Strix Point and the Lenovo ThinkPad X1 Carbon Gen 13 Aura Edition, I have been running a lot of Linux laptop benchmarks the past few weeks. I ended up taking things a bit further after those reviews and have now extended it to a 21-way laptop comparison of AMD Ryzen and Intel Core SoCs from the past few generations in looking at their performance on Ubuntu 25.04 across more than 200 benchmarks.

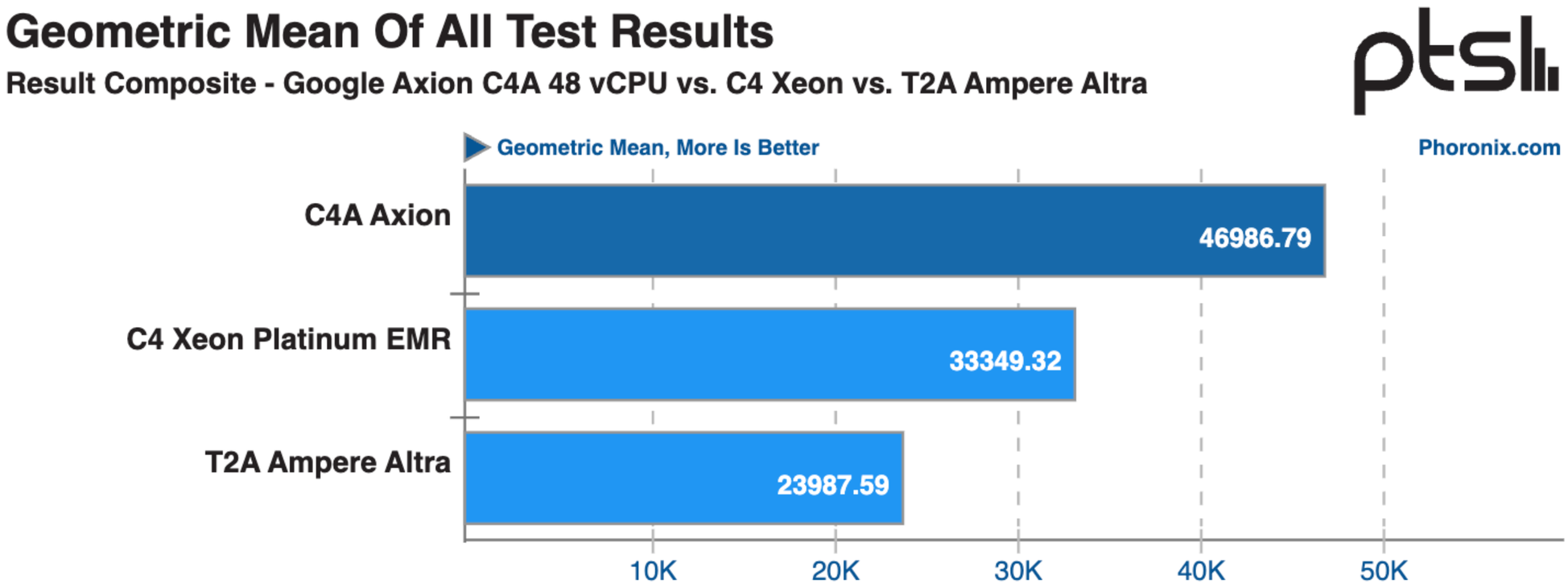
### Most Popular News This Week

- Bytedance Proposes Faster Linux Inter-Process Communication With "Run Process As Library"
- Valve's Proton 10.0 Beta Released With More Windows Games Now Playable On Linux
- Valve Releases Steam Survey Results For April 2025
- KDE Plasma 6.4 Tackles An 18 Year Old Feature Request, More Wayland Protocols Added
- Wine 10.7 Brings An Exciting Performance Optimization
- Plasma LTS Releases Being Discontinued, Better KDE Telemetry Like Valve's Steam Survey
- Linux 6.16 Will Begin Reporting The Cause Of Your AMD Zen System Being Reset/Rebooted
- Redis 8.0 Released: Now Tri-Licensed With AGPLv3
- Kcompressd Proposed For Accelerated Memory Compression On Linux
- Intel's Vulkan Linux Driver Adds Memory Pool Support For Some Massive Performance Gains

### Popular In The Forums

- NVIDIA Encouraging CUDA Users To Upgrade From Maxwell / Pascal / Volta
  49 Comments
- Full Disk Encryption Performance With Ubuntu 25.04 + Framework Laptop 13 Strix Point
  44 Comments

# Phoronix




# Axion vs Xeon vs T2A

| VM type | uArch / ISA | Hourly price (us-central1) | CoreMark (4 vCPU) | CoreMark per $ (higher = better) | $/perf index* |
|---|---|---|---|---|---|
| **N2-standard-4** | Intel Ice Lake (x86) | **$0.1942** | **66 884** | 344 k | **1.0 ×** |
| **T2A-standard-4** | Ampere Altra (Arm) | $0.1540 | 94 096 | 612 k | 1.78 × |
| **C4-standard-4** | Intel Emerald Rapids (x86) | $0.1977 | ≈131 000 **(~1.95 × N2)** † | 663 k | 1.92 × |
| **C4A-standard-4** | Google Axion V2 (Arm) | $0.1796 | ≈183 000 **(~2.7 × N2)** † | **1 021 k** | **2.96 ×** |

# N2 vs Axion vs Xeon vs T2A

## ARM are one of the best cost/performance gain you can have

- If your services are I/O-heavy, highly parallel, or licensed per vCPU (physical cores mean less contention and better economics) - that is very good choice.
- More modern ARM CPU - better results (Neoverse makes a real difference)
- Always use a modern OpenJDK (Java 11.09 or newer). Newer JDKs include Arm-specific optimizations for the latest Arm platforms, such as AWS Graviton 4.

# Wrap Up

# Upgrade JDK

- **Pick your cold-start super-power:**
  GraalVM Native Image hands you single-digit-millisecond launches and the tiniest containers at the cost of long native-image builds and no JIT, whereas Project CRaC snapshots a fully-warmed JVM and comes back in ~20 ms with full JIT speed, giving the best blend of startup and sustained throughput.
- Virtual Threads **shine for I/O-heavy traffic**
- Compact Object Headers (Project Lilliput) are real – early SPECjbb runs show **5-15 % smaller heaps** and ~8 % less CPU.
- ARM instances are the cost-performance secret weapon – Google's C4A Axion and AWS Graviton class machines deliver up to **~3× better perf-per-dollar**

# Heuristics to use

# JVM Weekly

Home | Notes | Foojay.io Community Newsletter | Archive | About

## "Optimizing the GC when Migrating Cloud Workloads to Arm" with Kieran Hejma...

Today, we have a guest post from Kieran Hejmadi, an Software and Academic Ecosystem Manager in Arm.

MAY 1 · ARTUR SKOWRONSKI AND KIERAN HEJMADI

Latest | Top | Discussions

### April - The Rest of the Story - JVM Weekly vol. 127

I know there is a lot of April left, not my fault, calendar is working against my publishing schedule this month...

APR 24 · ARTUR SKOWRONSKI

### Java in AWS with Wojtek Gawroński - JVM Weekly vol. 126

I'm experimental mood recently, so today, we have the interview!

APR 17 · ARTUR SKOWRONSKI AND WOJTEK GAWROŃSKI

### Make Final Mean Final... and the other JEPs - JVM Weekly vol. 125

It's been a while since we've seen JEPs! But you know, not the fully baked ones, not the ones that have advanced far i...

**JVM Weekly**

From the latest updates in JVM languages like Java, Kotlin, and Scala to emerging technologies like GraalVM and Quarkus, this newsletter covers a wide range of topics that are of interest to developers and tech enthusiasts.

✓ Subscribed

**Recommendations** MANAGE

- **Tales from the jar side** — Ken Kousen
- **Architecture Weekly** — Oskar Dudycz
- **Engineering Primer** — Bartek Antoniak
- **Tech Talks Weekly**

VIRTUSLAB
Competition results
Wednesday 18:00
Thursday 18:00
Friday 15:00
Build what others rely on
See our open positions
rebel
VIRTUSLAB
VIRTUS LAB
Ever been anywhere that Koda wasn't?

# Thank you 🙇

@ArturSkowronski

VIRTUSLAB

jvm weekly