



Supercharge your JVM performance with Project Leyden and Spring Boot

Moritz Halbritter

Spring Engineering Team

2026-02-10 @ Voxxed Days CERN

About me



git log

- GraalVM native image support
- Observability
- Docker Compose & Testcontainers support
- Virtual threads
- SSL hot reload
- CDS, SBOM
- Structured Logging
- JSpecify Nullability
- ...

Goals



Understand how Project Leyden boosts Java startup and performance.



Demonstrate how to use Leyden-related optimizations available in JDK 25 with Spring Boot.

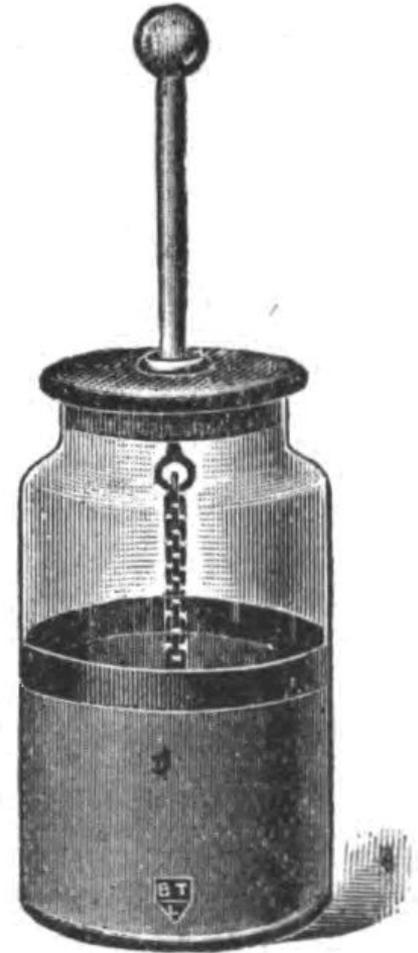


Learn techniques you can apply today along with insights into what's ahead.

**Performance is a journey, not
a finish line.**

Project Leyden

Improve the startup time and warmup time of Java applications by shifting computation earlier or later in time.



Static Analysis

Dynamic Observation

Training Runs

Purpose

- Exercise the startup and warmup code paths, under observation.
- Discover ahead-of-time what you'd otherwise find early at runtime.

What Constitutes a Training?

- **Best training run is observing the application in production.**
- Usually needs writing a small driver program (like an integration test).
- Runs at build time (similar to an integration test).

Why Do They Work?

- Effective for the same reason dynamic compilation is.
- Analysis is driven by what the program actually does.

JDK 24's Three-Phase Workflow

Training Run

```
> java -XX:AOTMode=record -XX:AOTConfiguration=app.aotconf \  
      -cp app.jar com.example.App ...
```

Assembly Phase

```
> java -XX:AOTMode=create -XX:AOTConfiguration=app.aotconf \  
      -XX:AOTCache=app.aot -cp app.jar
```

Deployment Run

```
> java -XX:AOTCache=app.aot -cp app.jar com.example.App ...
```

JEP 483: Ahead-of-Time Class Loading & Linking

Authors Ioi Lam, Dan Heidinga, & John Rose

Owner Ioi Lam

Type Feature

Scope JDK

Status Closed / Delivered

Release 24

Component hotspot / runtime

Discussion leyden dash dev at openjdk dot org

Relates to JEP 515: Ahead-of-Time Method Profiling

JEP 514: Ahead-of-Time Command-Line Ergonomics

JEP 515: Ahead-of-Time Method Profiling

Author Igor Veresov & John Rose

Owner John Rose

Type Feature

Scope Implementation

Status Closed / Delivered

Release 25

Component hotspot / compiler

Discussion leyden dash dev at openjdk dot org

Effort M

Duration M

JEP 514: Ahead-of-Time Command-Line Ergonomics

Owner John Rose

Type Feature

Scope JDK

Status Closed / Delivered

Release 25

Component hotspot / runtime

Discussion leyden dash dev at openjdk dot org

Effort M

Duration S

Relates to JEP 483: Ahead-of-Time Class Loading & Linking

JDK 18

JDK 19

JDK 20

JDK 21

JDK 22

JDK 23

JDK 24

JDK 25

JDK 26

JDK 27

JDK 25's Two-Phase Workflow

```
# Training Run + Assembly Phase
```

```
> java -XX:AOTCacheOutput=app.aot -cp app.jar com.example.App ...
```

```
# Deployment Run
```

```
> java -XX:AOTCache=app.aot -cp app.jar com.example.App ...
```

Leyden is fully compatible

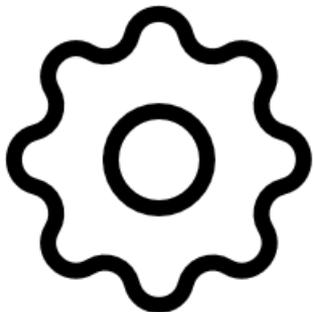
AOT Cache with Spring Boot



```
➤ j █
```

```
java -jar spring-petclinic-4.0.0-SNAPSHOT.jar
```

AOT Cache Requirements



JVM

The same JVM must be used.

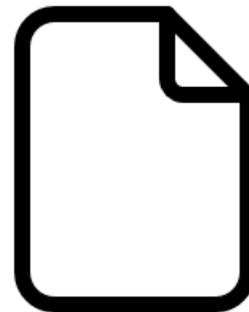


Classpath

Must be specified as a list of jars.

No directories, no wildcards, no nested jar.

Deployment classpath must be a superset of the training one.



Files

The timestamp of the jars must be preserved.

Extract Uber Jar

```
> java -Djarmode=tools -jar application.jar extract --destination extracted
```

```
> tree extracted
```

```
extracted
```

```
├─ application.jar
```

```
└─ lib
```

```
    ├─ commons-logging-1.3.5.jar
```

```
    ├─ jackson-annotations-2.20.jar
```

```
    ├─ jackson-core-3.0.0-rc9.jar
```

```
    ├─ jackson-databind-3.0.0-rc9.jar
```

```
    ├─ jakarta.annotation-api-3.0.0.jar
```

```
    ├─ jspecify-1.0.0.jar
```

```
    ├─ jul-to-slf4j-2.0.17.jar
```

```
    └─ ...
```



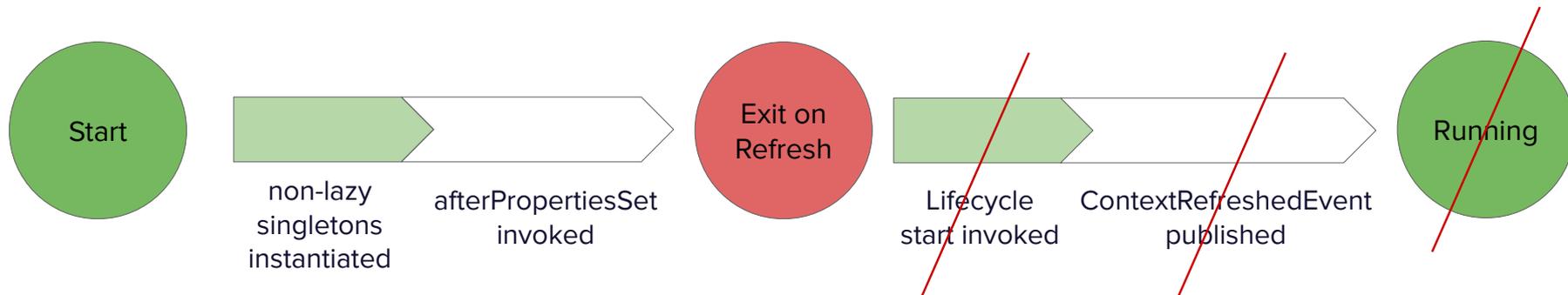
```
j █
```

```
java -jar extracted/spring-petclinic-4.0.0-SNAPSHOT.jar
```

Exit the Application Before Context Refresh

Poor man's training run

```
> java -Dspring.context.exit=onRefresh -jar extracted/application.jar
```

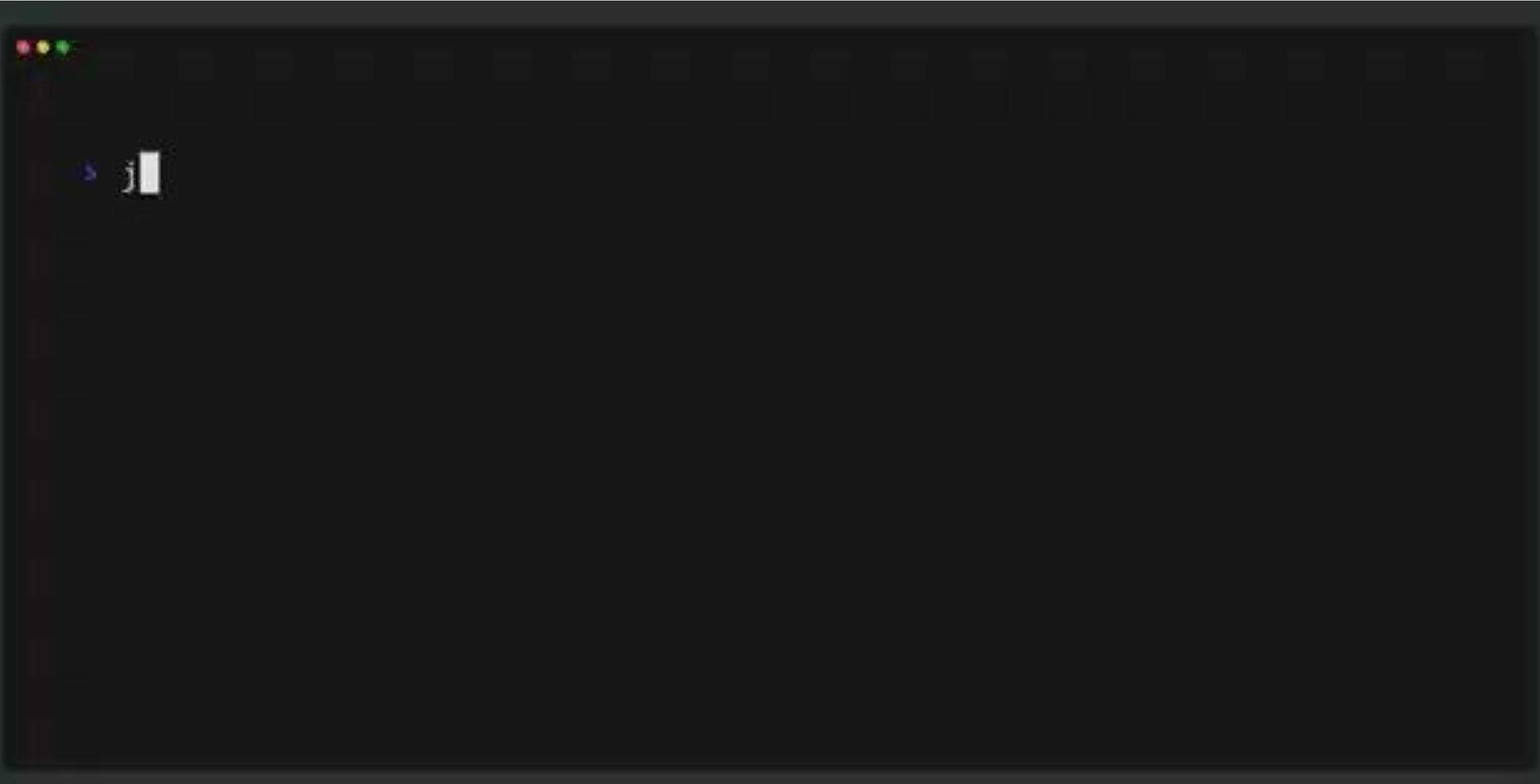


Create and use the AOT Cache

```
# Create the AOT cache
> java -XX:AOTCacheOutput=app.aot -Dspring.context.exit=onRefresh \
    -jar extracted/application.jar
```

```
# Let's see how big it is
> ll app.aot
Permissions Size Name
.rw-r--r--@ 53M app.aot
```

```
# Run the application with the AOT cache
> java -XX:AOTCache=app.aot -jar extracted/application.jar
```

A terminal window with a dark background. In the top-left corner, there are three small colored circles (red, yellow, green) representing window control buttons. Below them, a blue prompt character is followed by a white cursor block. The rest of the terminal is empty.

```
java -XX:AOTCache=extracted/app.aot -jar extracted/spring-petclinic-4.0.0-SNAPSHOT.jar
```

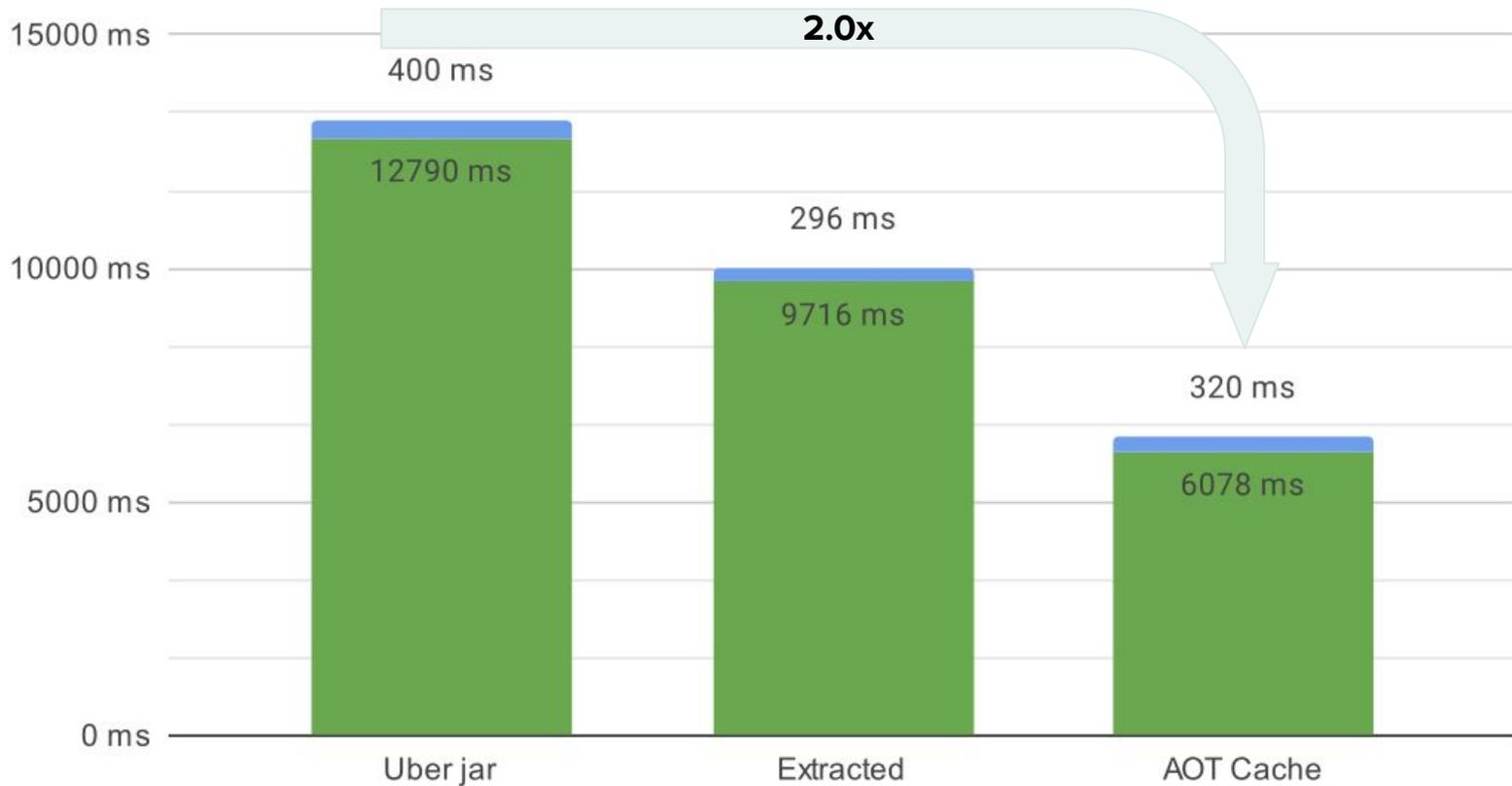
Time to first request (Spring Boot 4.0.2, Java 25)



RSS (Spring Boot 4.0.2, Java 25)



User time / Kernel time (Spring Boot 4.0.2, Java 25)



AOT Cache and Spring AOT

AOT Cache

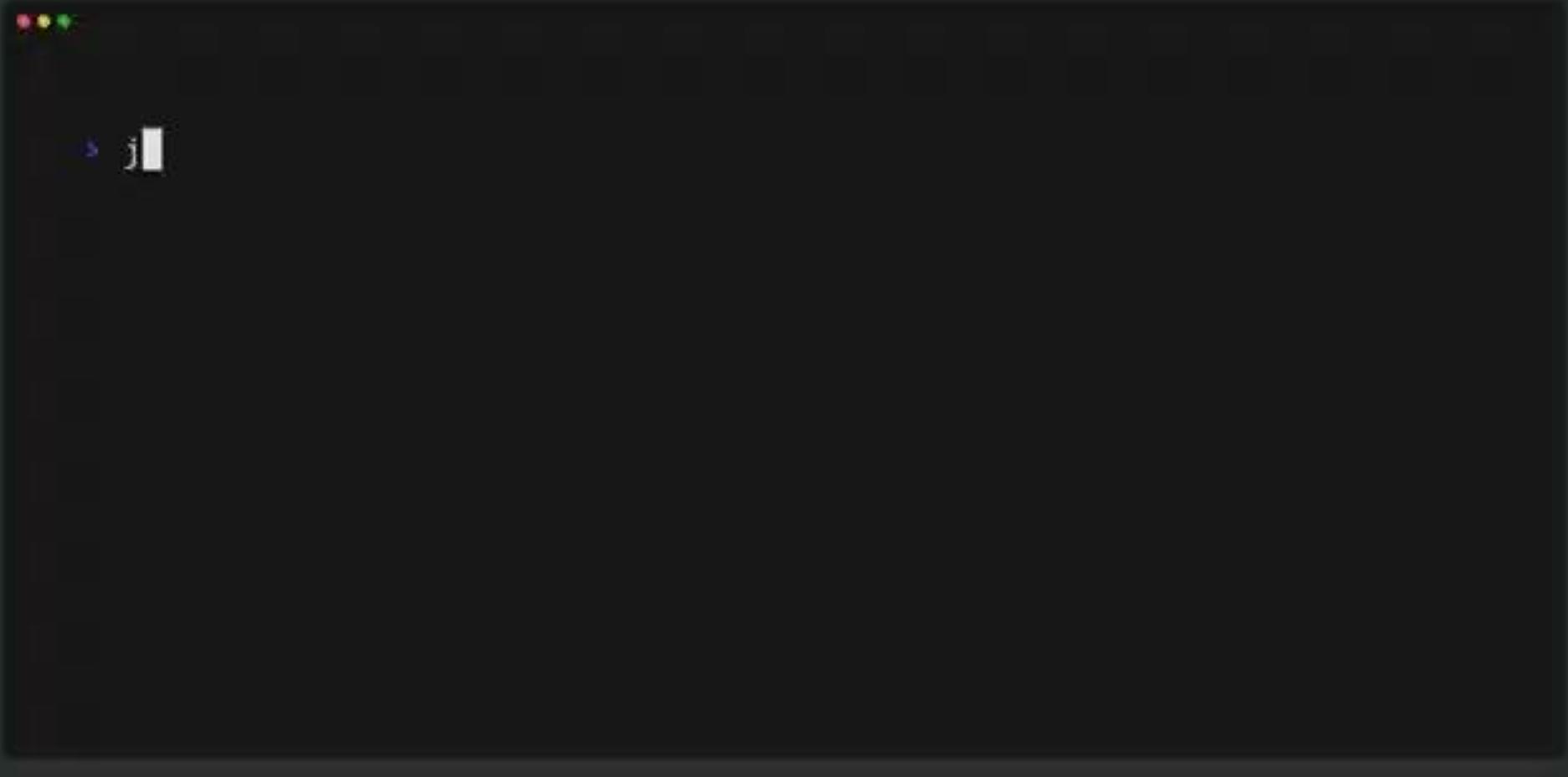
JVM feature developed via Project Leyden to improve the efficiency of the JVM. It supersedes CDS.

Spring AOT

Spring feature mandatory for GraalVM native images Support.

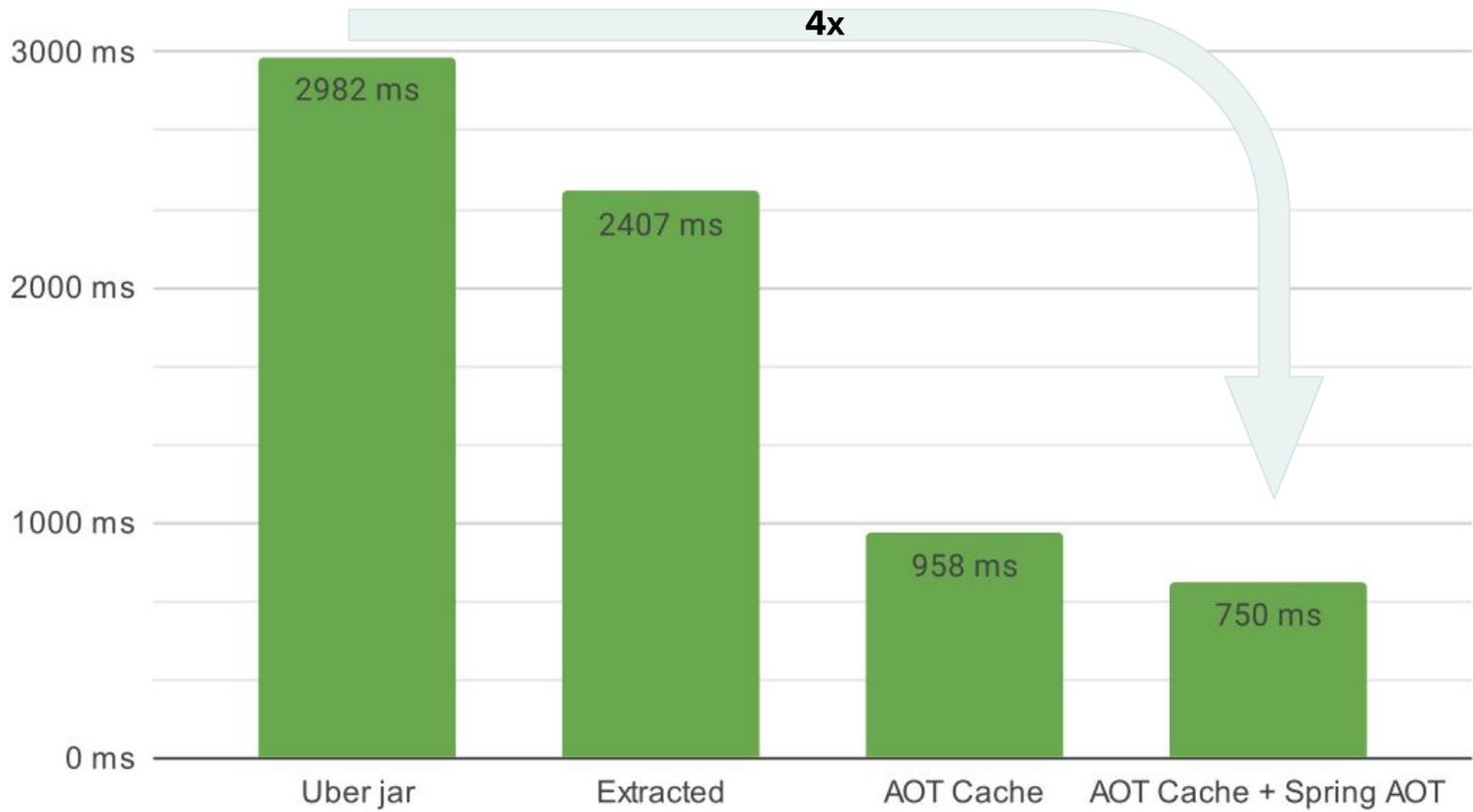
Can also be used on the JVM to speed up the startup process and lower the memory consumption.

Generates code ahead of time for the bean arrangement and other features, e.g. Spring Data repositories

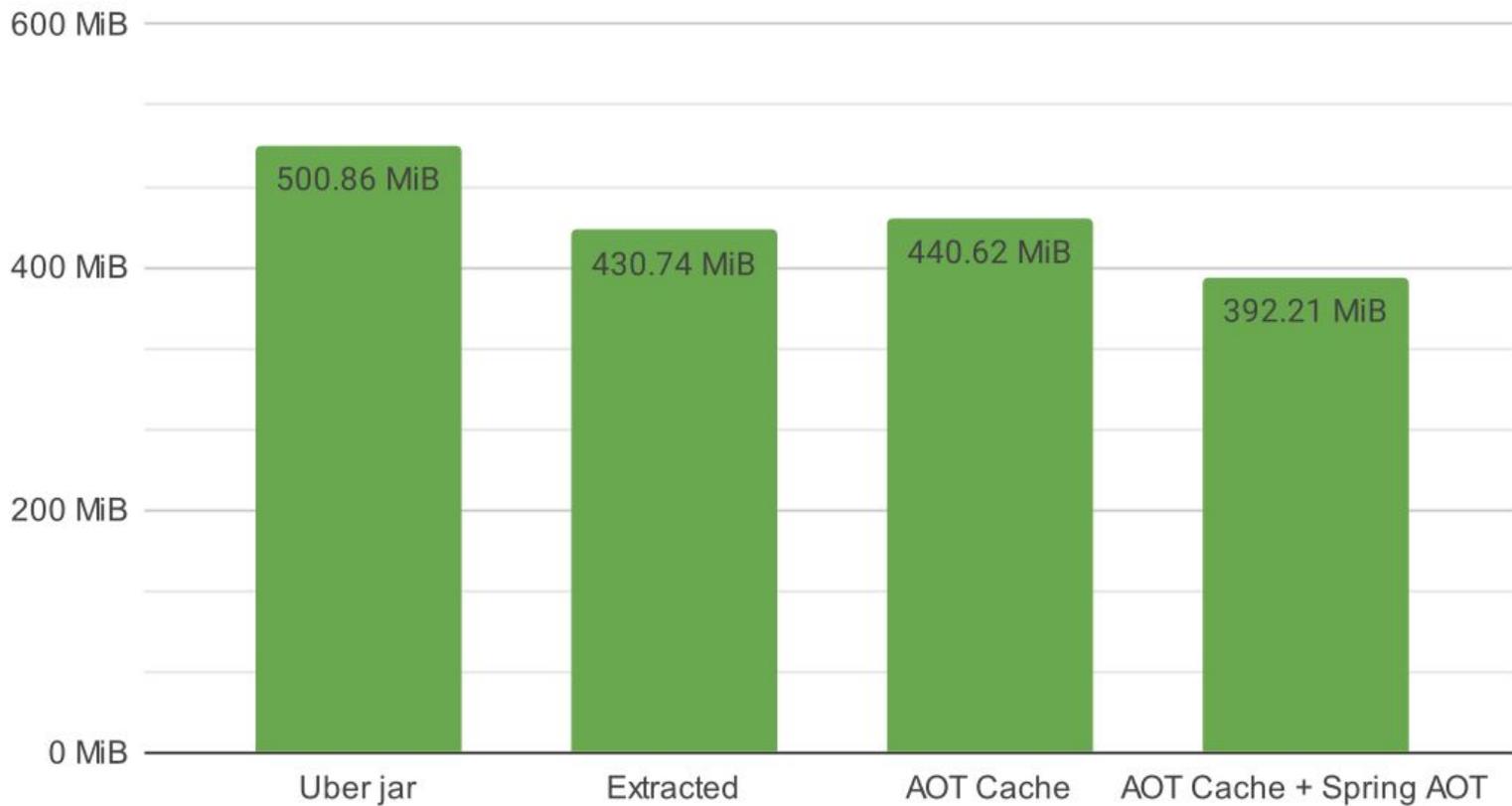
A terminal window with a dark background. In the top-left corner, there are three small colored circles (red, yellow, green) representing window control buttons. On the left side, there is a blue prompt character followed by a white cursor block.

```
java -XX:AOTCache=extracted-aot/app.aot -Dspring.aot.enabled=true -jar extracted-aot/spring-petclinic-4.0.0-SNAPSHOT-aot.jar
```

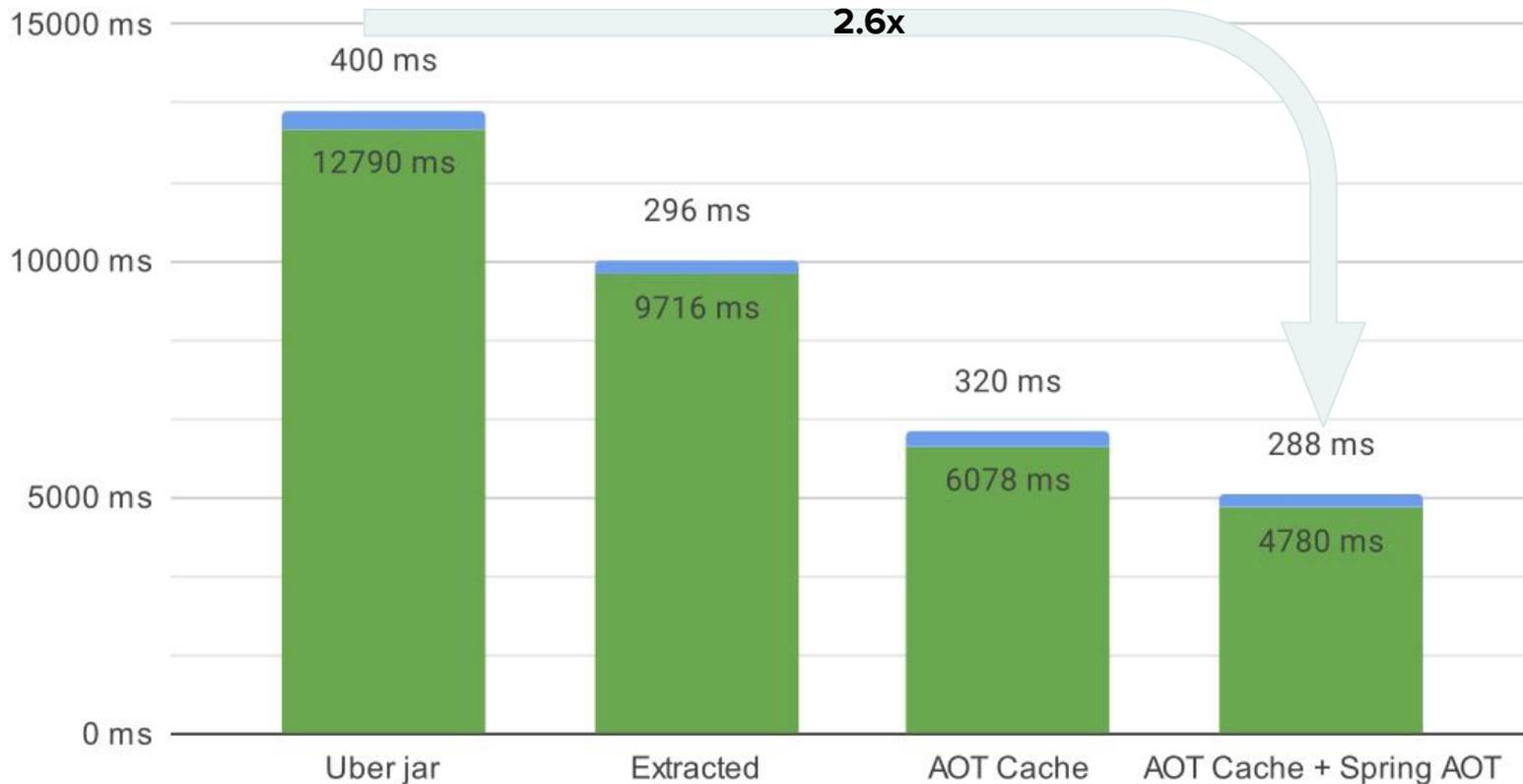
Time to first request (Spring Boot 4.0.2, Java 25)



RSS (Spring Boot 4.0.2, Java 25)



User time / Kernel time (Spring Boot 4.0.2, Java 25)



JEP 515: Ahead-of-Time Method Profiling

Author Igor Veresov & John Rose
Owner John Rose
Type Feature
Scope Implementation
Status Closed / Delivered
Release 25
Component hotspot / compiler
Discussion leyden dash dev at openjdk dot org
Effort M
Duration M
Relates to [JEP 483: Ahead-of-Time Class Loading & Linking](#)
Reviewed by Alex Buckley, Dan Heidinga, Vladimir Kozlov
Endorsed by Vladimir Kozlov
Created 2024/02/01 20:40
Updated 2025/07/17 19:01
Issue 8325147

Summary

Improve warmup time by making method-execution profiles from a previous run of an application instantly available, when the HotSpot Java Virtual Machine starts. This will enable the JIT compiler to generate native code immediately upon application startup, rather than having to wait for profiles to be collected.

Ways to Do Training Runs

Stop the application before context refresh

- Easy setup
- Can be done while building the image
- Improves start-up time by caching initial class loading

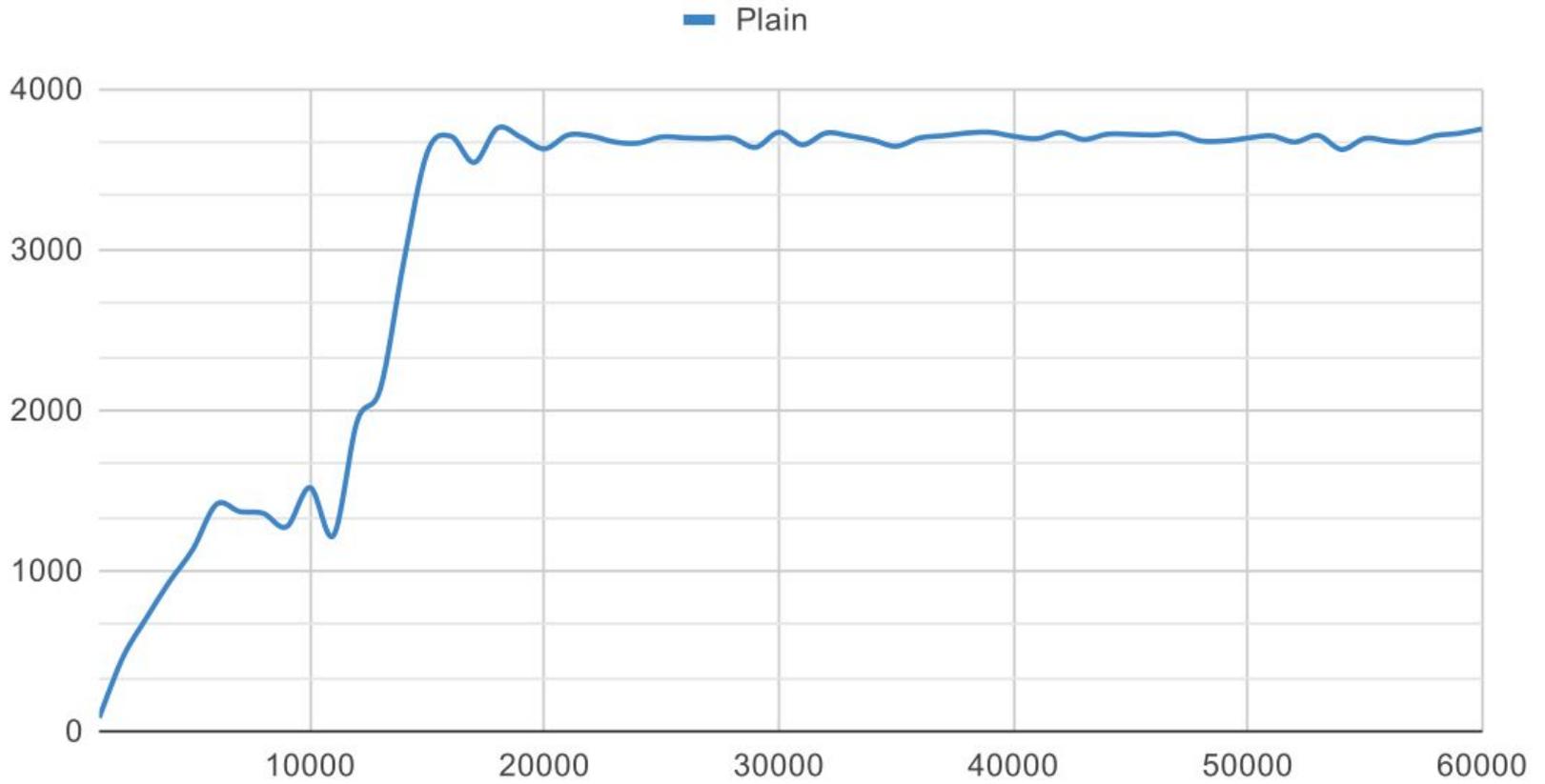
- Doesn't improve warm-up time because it doesn't collect profiling information

Run integration tests / mirror production traffic

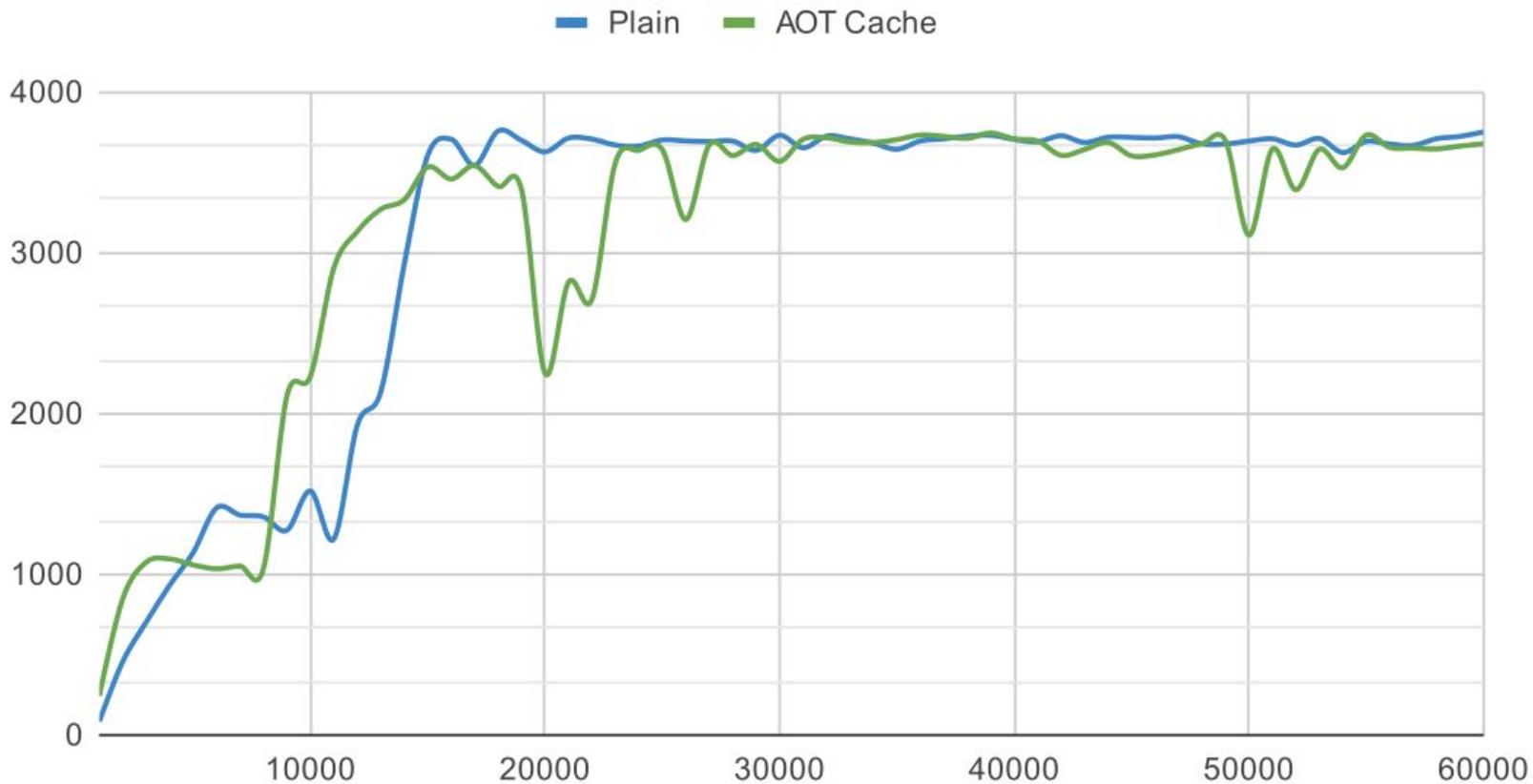
- Exercises many/all hot code paths used in production
- Improves start-up time
- Improves warm-up time by collecting profiling information

- Can't easily be done while building the image
- More involved setup

Requests / second (1 CPU)



Requests / second (1 CPU)



Warm-up run: Load generator against the vets.html page

**All of this is already possible
today using Java 25!**

What's next?

JEP 516: Ahead-of-Time Object Caching with Any GC

Owner Erik Österlund
Type Feature
Scope JDK
Status Completed
Release 26
Component hotspot/gc
Discussion hotspot dash dev at openjdk dot org
Effort M
Duration M
Reviewed by Alex Buckley, loi Lam, Stefan Karlsson, Vladimir Kozlov
Endorsed by Vladimir Kozlov
Created 2024/02/16 09:49
Updated 2025/11/21 13:11
Issue 8326035

JEP draft: Ahead-of-Time Code Compilation

Owner John Rose
Type Feature
Scope Implementation
Status Draft
Component hotspot/compiler
Created 2024/06/30 04:47
Updated 2025/10/09 16:21
Issue 8335368

Summary

Improve startup and warmup time by making native code from a previous run of an application instantly available, when the HotSpot Java Virtual Machine starts. This will greatly reduce the initial load on the JIT compiler, reducing its interference with the application during startup, particularly in configurations with fewer cores. The JIT is then free to delay the generation of native code, unless and until the previously generated code proves insufficiently performant.

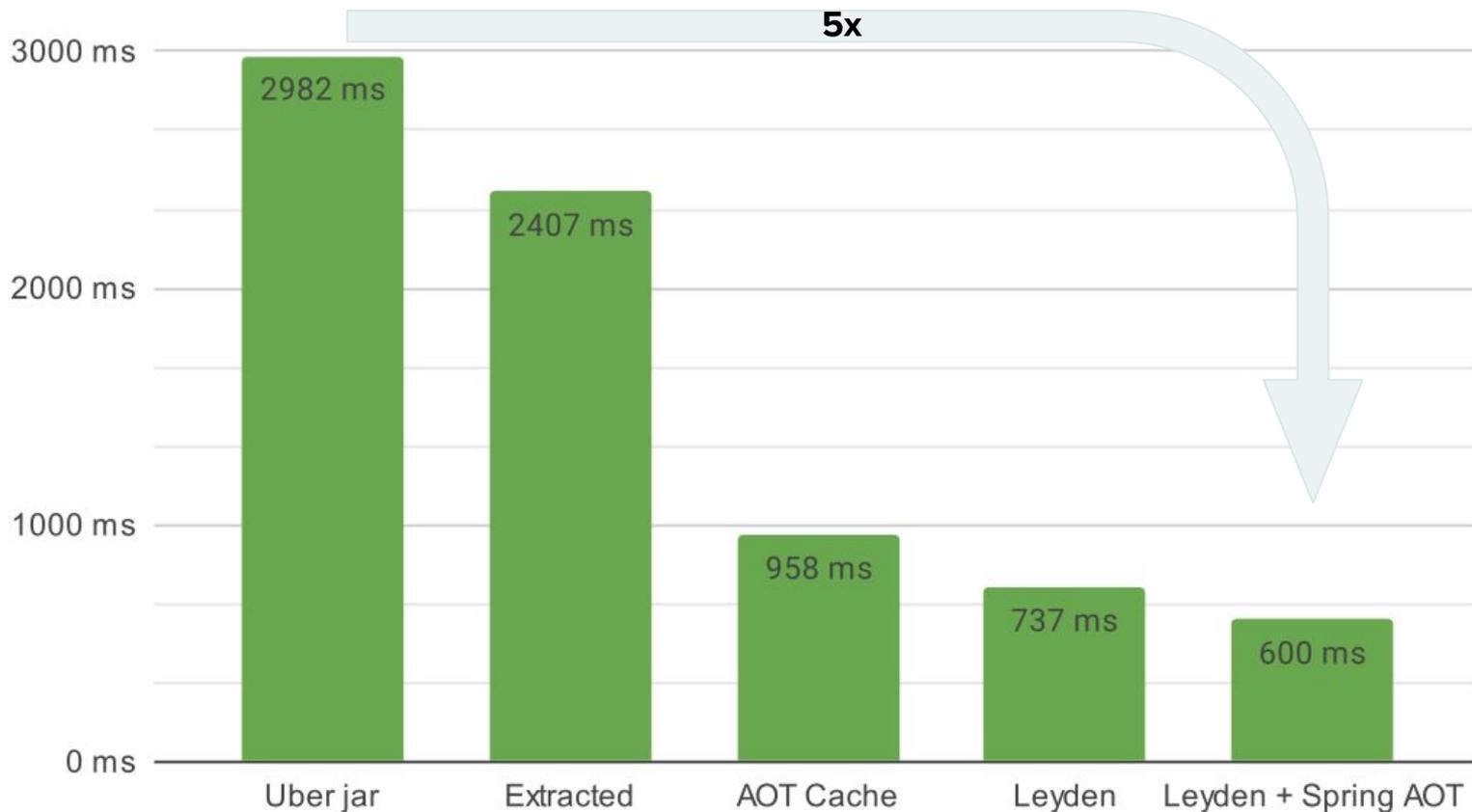
JDK 18
JDK 19
JDK 20
JDK 21
JDK 22
JDK 23
JDK 24
JDK 25
JDK 26
JDK 27
???



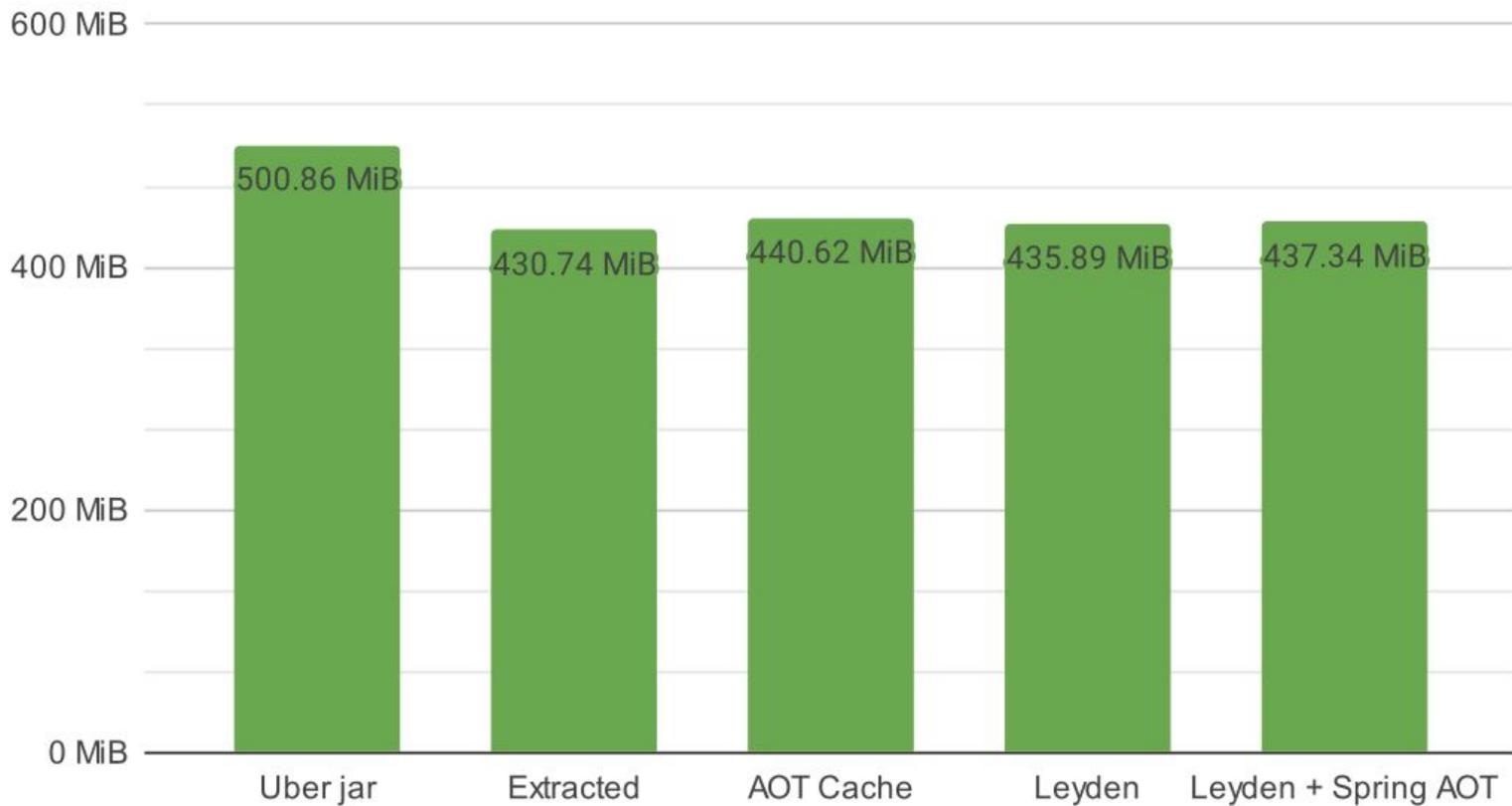
```
> j |
```

```
java -XX:AOTCache=extracted-aot/app-leyden.aot -Dspring.aot.enabled=true -jar extracted-aot/spring-petclinic-4.0.0-SNAPSHOT-aot.jar
```

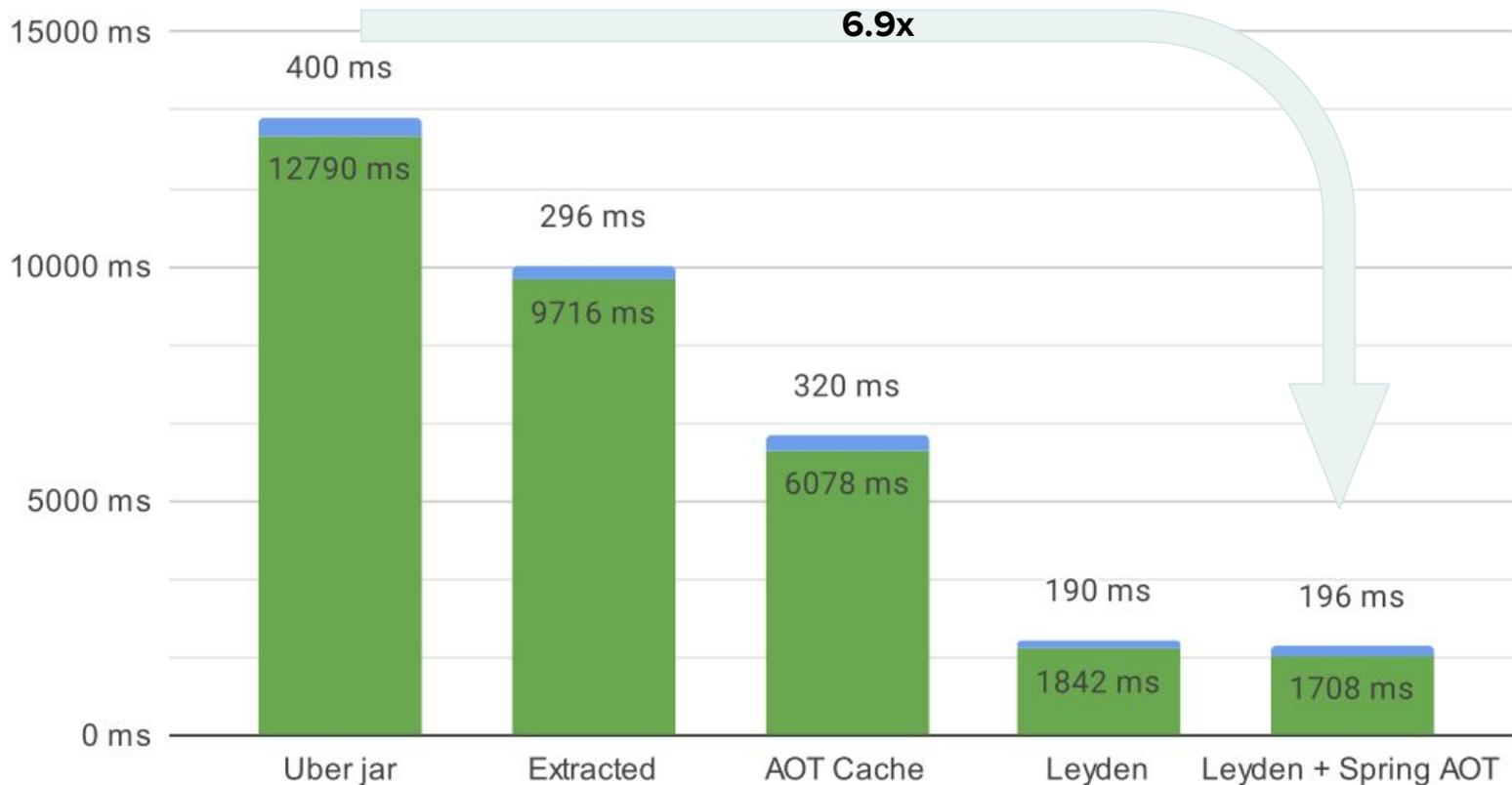
Time to first request (Spring Boot 4.0.2, Java Leyden)



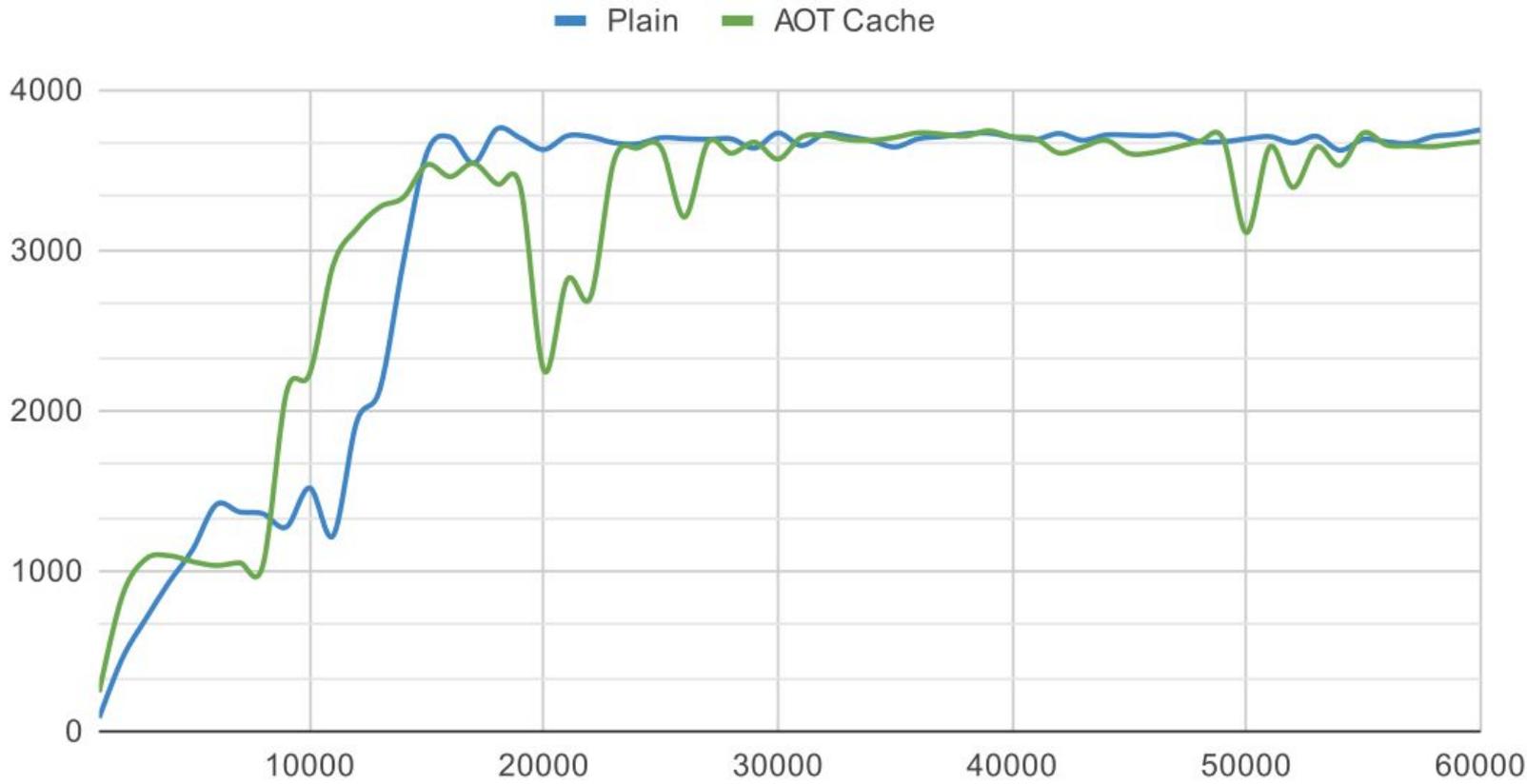
RSS (Spring Boot 4.0.2, Java Leyden)



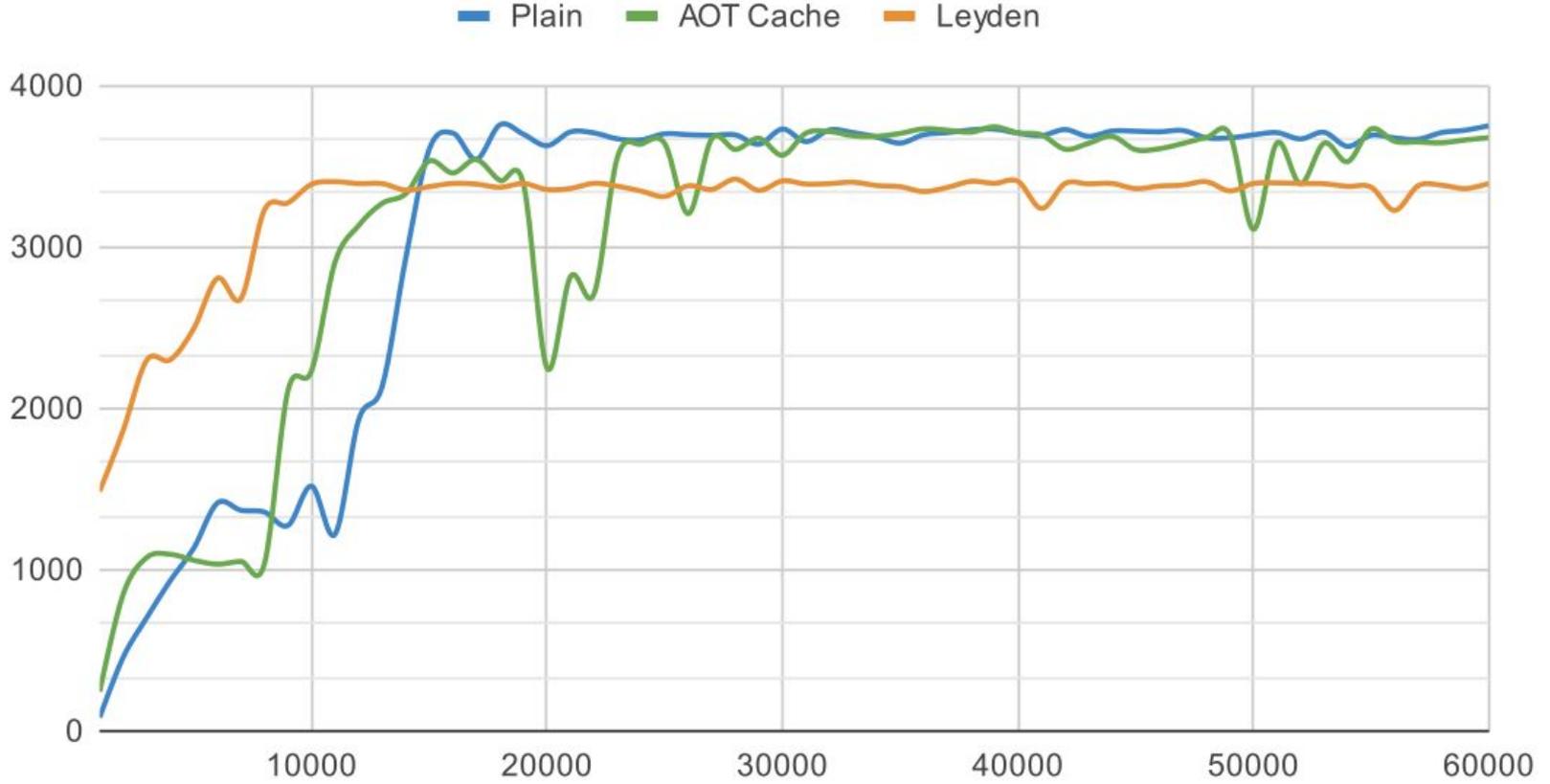
User time / Kernel time (Spring Boot 4.0.2, Java Leyden)



Requests / second (1 CPU)



Requests / second (1 CPU)



Balance of Performance and Portability in Leyden

Portability is a priority

- Should AOT code use the best possible instructions or compile for a “typical” deployment target?
- Also applies to other parts of the system as well (heap sizes, garbage collectors, etc)

Extend AOT capabilities to user-defined classloaders

- Training runs are currently focused on the 3 built-in classloaders (System, Extension, and Boot loaders) and the classes loaded by them.
- User classloaders would benefit from the same concept but there a lot of challenges to get there.

AOT improvements planned in Spring projects

- [Introduce modular Spring AOT](#)
 - Pick and choose which parts of Spring AOT are used
 - e.g. you may want Spring Data repositories and predefined classes, but you don't want a frozen bean arrangement
- [Precompute classpath scanning](#)
- [Customized condition evaluation](#)
- [Add hook points for bean optimization](#)
- ...

Current prototype:

```
processAot {  
    beanRegistration = false  
    predefinedClasses = true  
    classpathIndexes = true  
    reachabilityMetadata = false  
}
```

Invest in training your application today.

Keep up with the JDK and Spring Boot releases to unlock available optimizations.

Get better performance each release, with Leyden fully Java Platform-compatible.

Actionable right now

Using Java 25, AOT Cache and Spring AOT:

- No prototype JDKs, no preview flags, just java 25 GA
- Improve startup time by **4x**
- CPU consumption during startup and warmup reduced by **2.6x**
- And this will only improve with future JDKs!

Thank you

Contact me at moritz.halbritter@broadcom.com

mhalbritter.github.io

