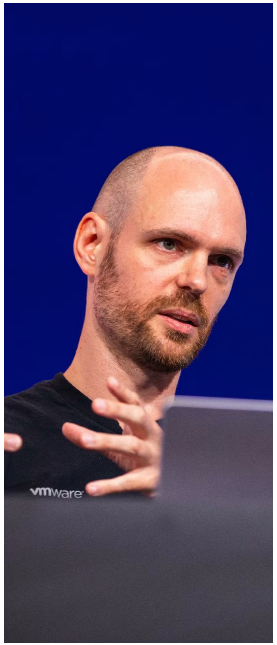
Supercharge your JVM performance with Project Leyden and Spring Boot

Moritz Halbritter
Spring Engineering Team
2026-03-24 @ Voxxed Days Zurich

About me



git log

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

spring

What's the problem?

```
Started Application in 10.492 seconds (process running for 10.677)
```

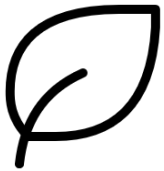
- CPU time spent on startup costs money
- Long startup time limits scalability

Why do we want scalability?

- Handle traffic spikes
- Ensure responsiveness
- Rapid recovery
- Saves money and energy
    - Think about scale to zero

Goals

Understand how Project Leyden boosts Java startup and performance.

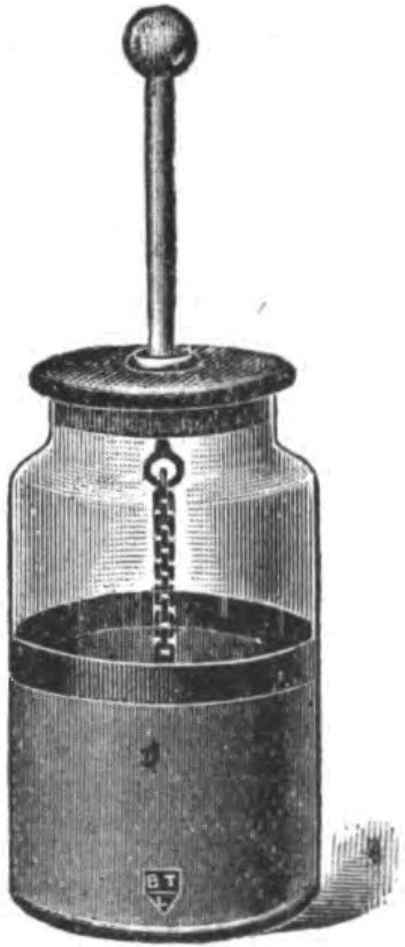


Demonstrate how to use Leyden-related optimizations available in JDK 25 with Spring Boot.

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

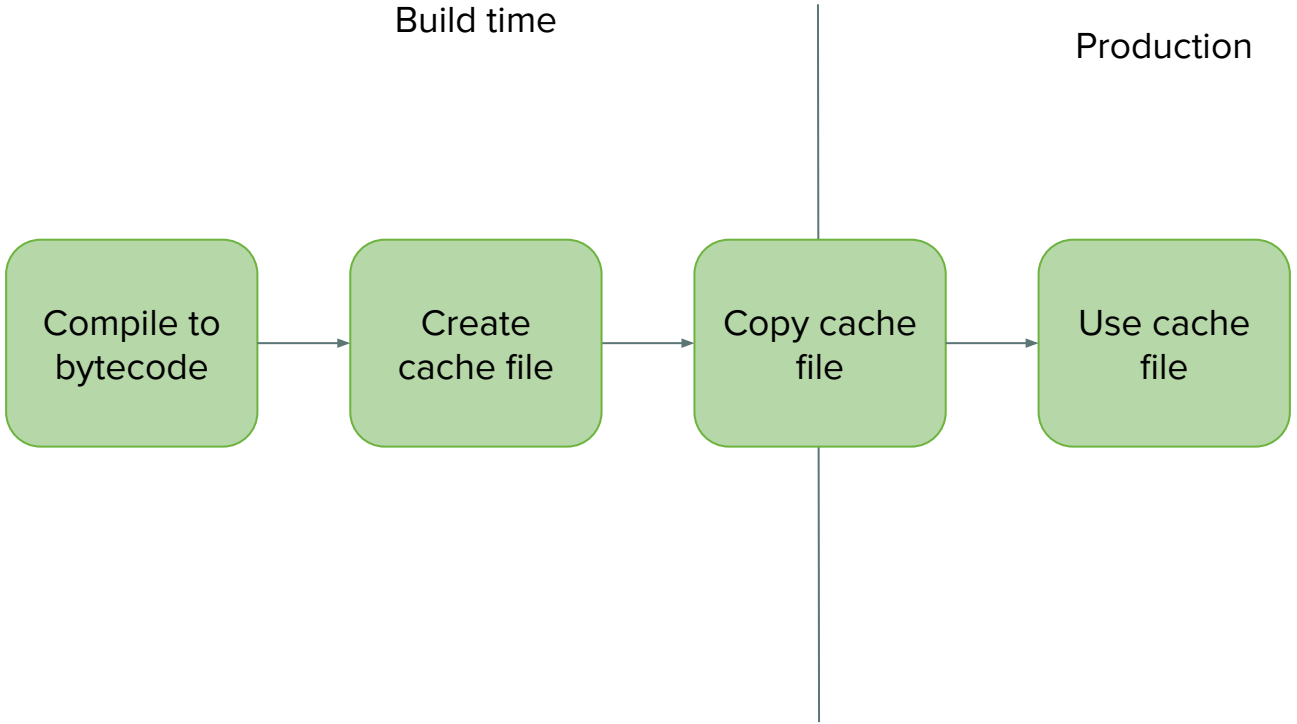
Project Leyden

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



spring

General idea



spring

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.

JEP 310: Application Class-Data Sharing

Owner	Ioi Lam
Type	Feature
Scope	Implementation
Status	Closed / Delivered
Release	10
Component	hotspot / runtime
Discussion	hotspot dash dev at openjdk dot java dot net
Relates to	JEP 350: Dynamic CDS Archives
Reviewed by	Karen Kinnear, Mikael Vidstedt, Vladimir Kozlov
Endorsed by	Mikael Vidstedt, Vladimir Kozlov
Created	2017/08/08 22:02
Updated	2023/08/24 19:13
Issue	8185996

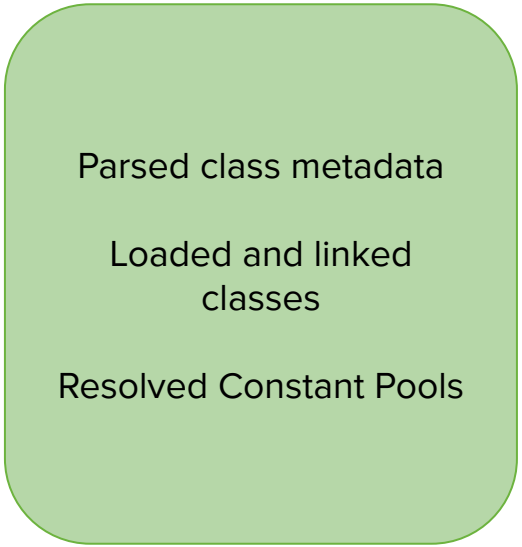
JEP 483: Ahead-of-Time Class Loading & Linking

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

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 ...
```

JDK 24: What's in the cache?



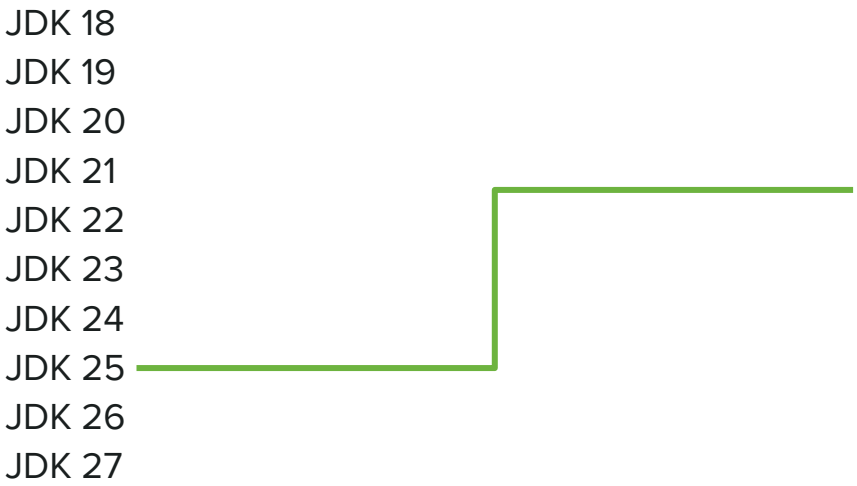
JDK 18
JDK 19
JDK 20
JDK 21
JDK 22
JDK 23
JDK 24
JDK 25
JDK 26
JDK 27

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 25's Two-Phase Workflow

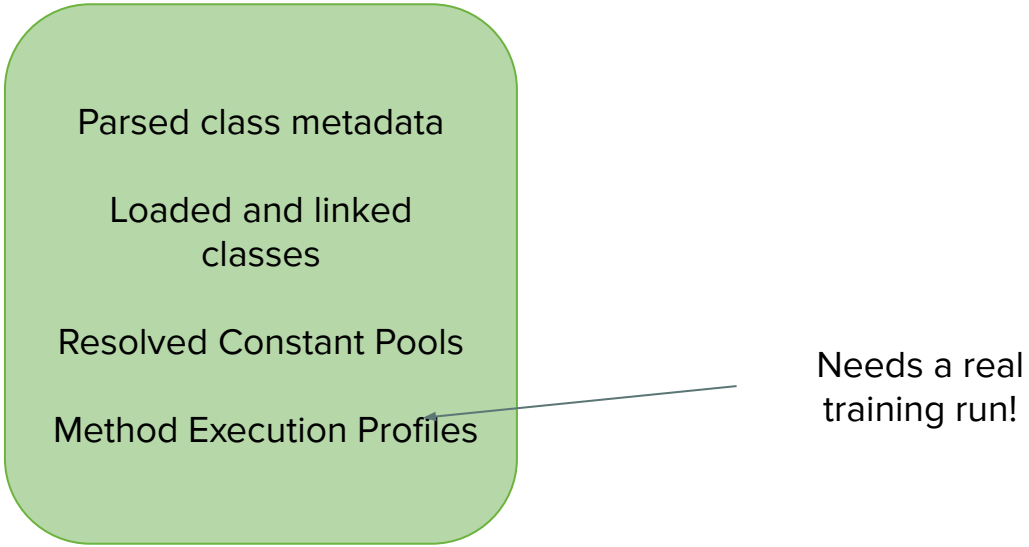
```
# Training Run + Assembly Phase
> java -XX:AOTCacheOutput=app.aot -cp app.jar com.example.App ...


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



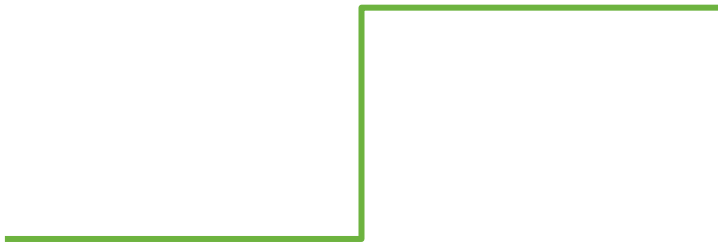
JEP 515: Ahead-of-Time Method Profiling

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

JDK 25: What's in the cache?



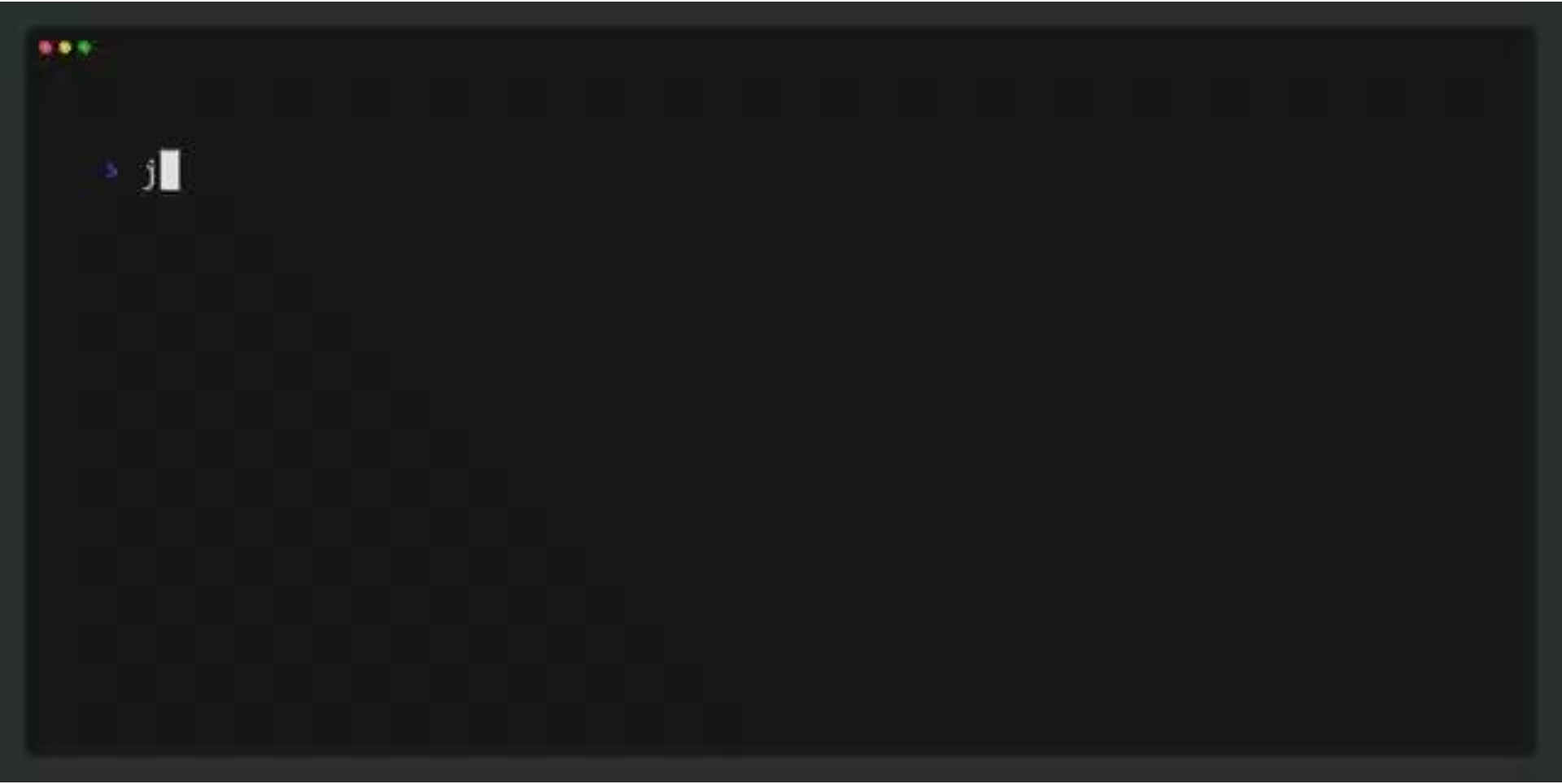
JDK 18
JDK 19
JDK 20
JDK 21
JDK 22
JDK 23
JDK 24
JDK 25
JDK 26
JDK 27



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

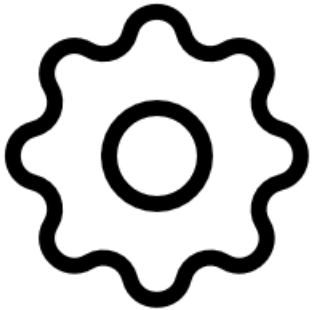
Owner	Erik Österlund
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	26
Component	hotspot / gc
Discussion	hotspot dash dev at openjdk dot org
Effort	M
Duration	M
Reviewed by	Alex Buckley, Ioi Lam, Stefan Karlsson, Vladimir Kozlov
Endorsed by	Vladimir Kozlov
Created	2024/02/16 09:49
Updated	2025/12/17 15:09
Issue	8326035

AOT Cache with Spring Boot




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

AOT Cache Requirements



JVM

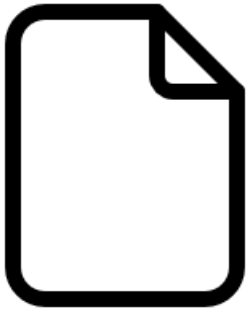
The same JVM must be used.

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.

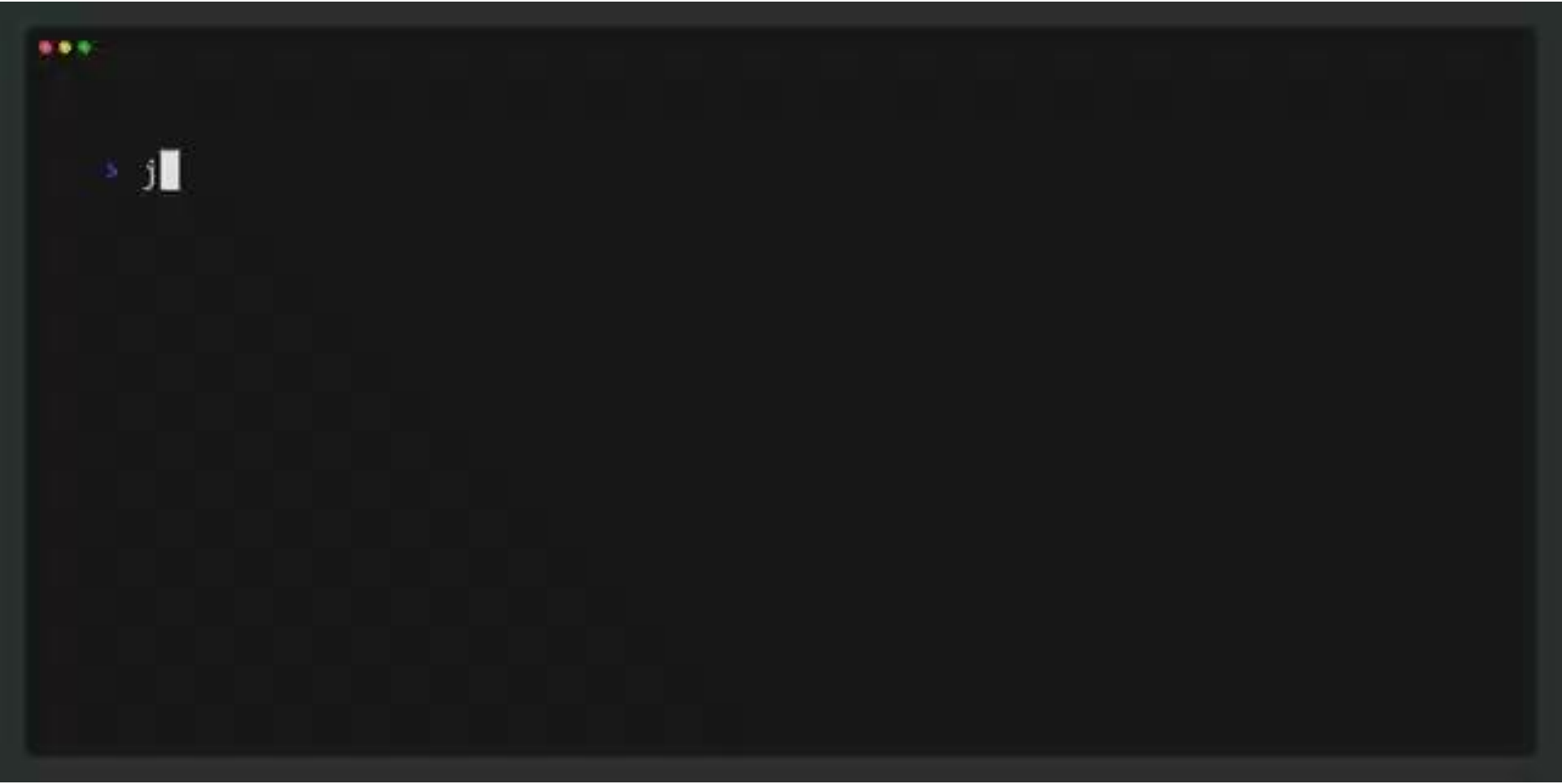


Files

The timestamp of the jars must be preserved.

Extract Uber Jar

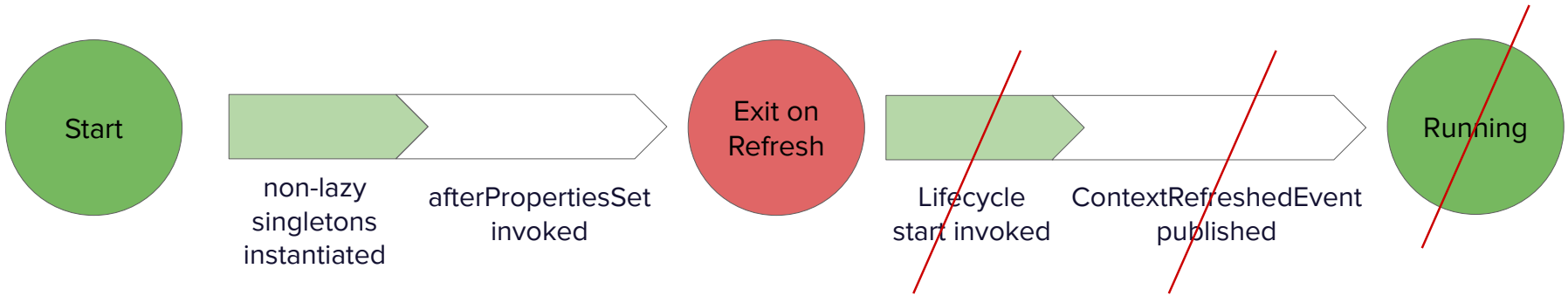
```
> java -Djarmode=tools -jar application.jar extract --destination extracted


> 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
    ├── ...
```



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
```

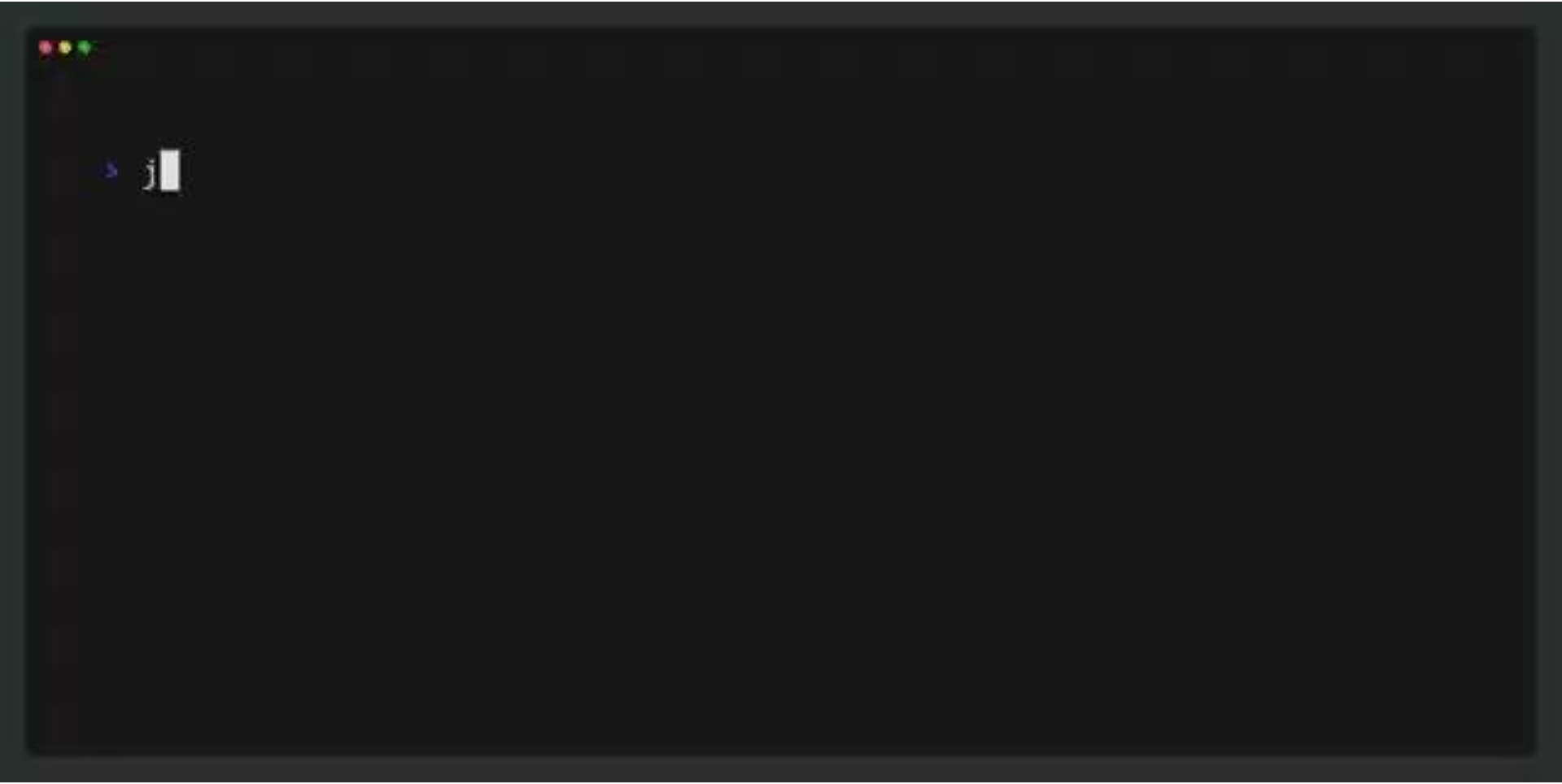


Idea: Capture class loading of most important classes

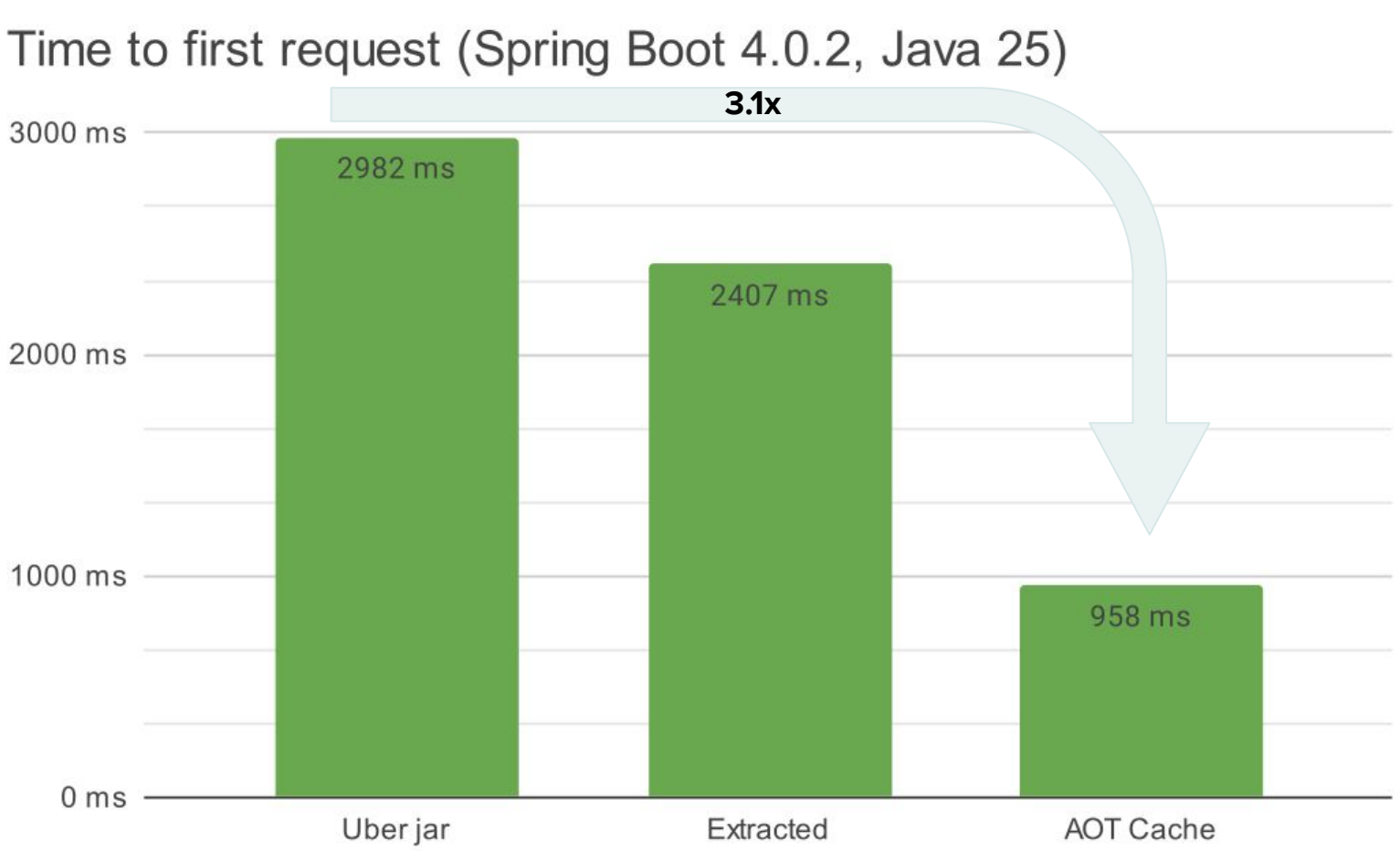
spring

Create and use the AOT Cache

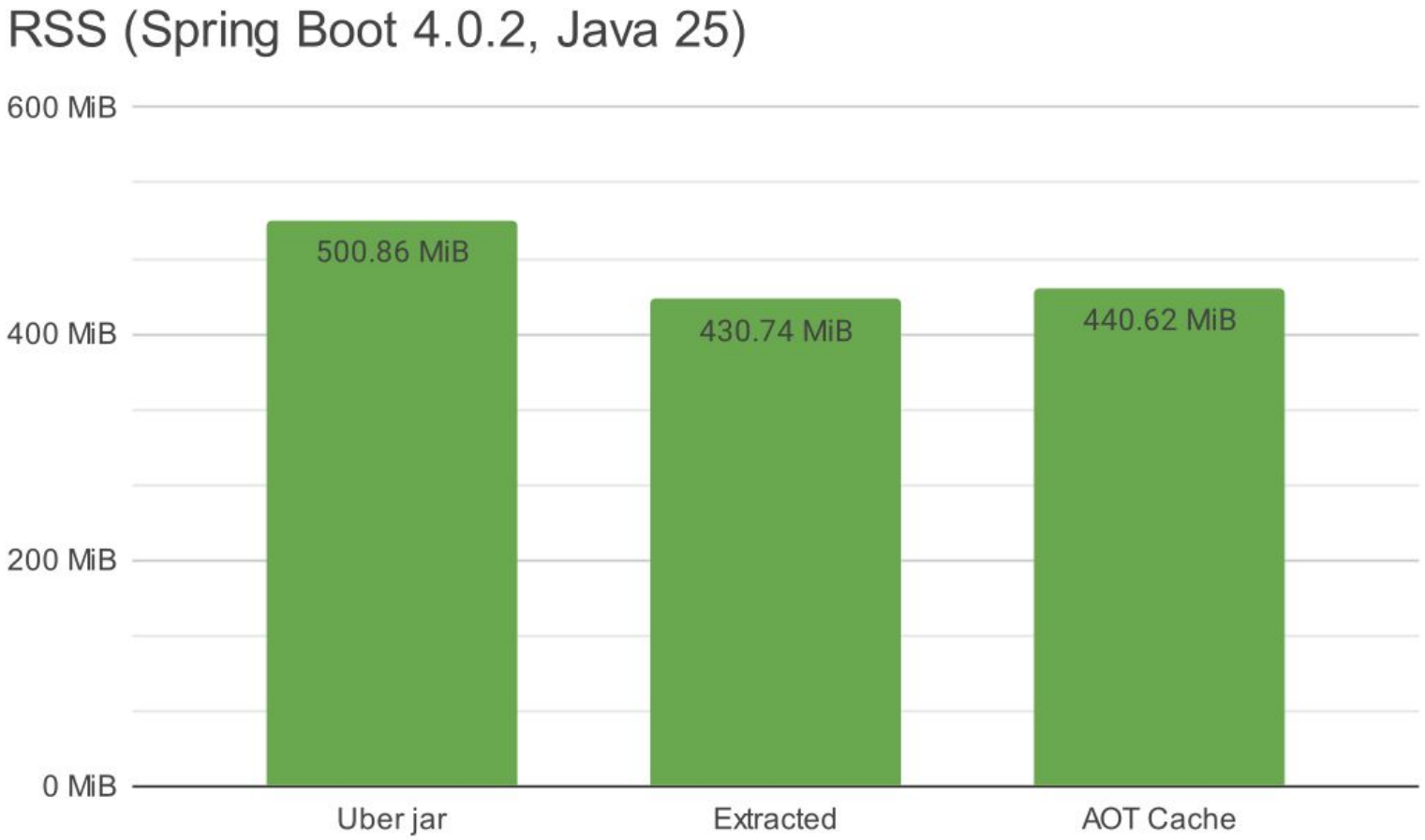
```
# Create the AOT cache
> java -XX:AOTCacheOutput=app.aot -Dspring.context.exit=onRefresh \
       -jar extracted/application.jar


# 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
```



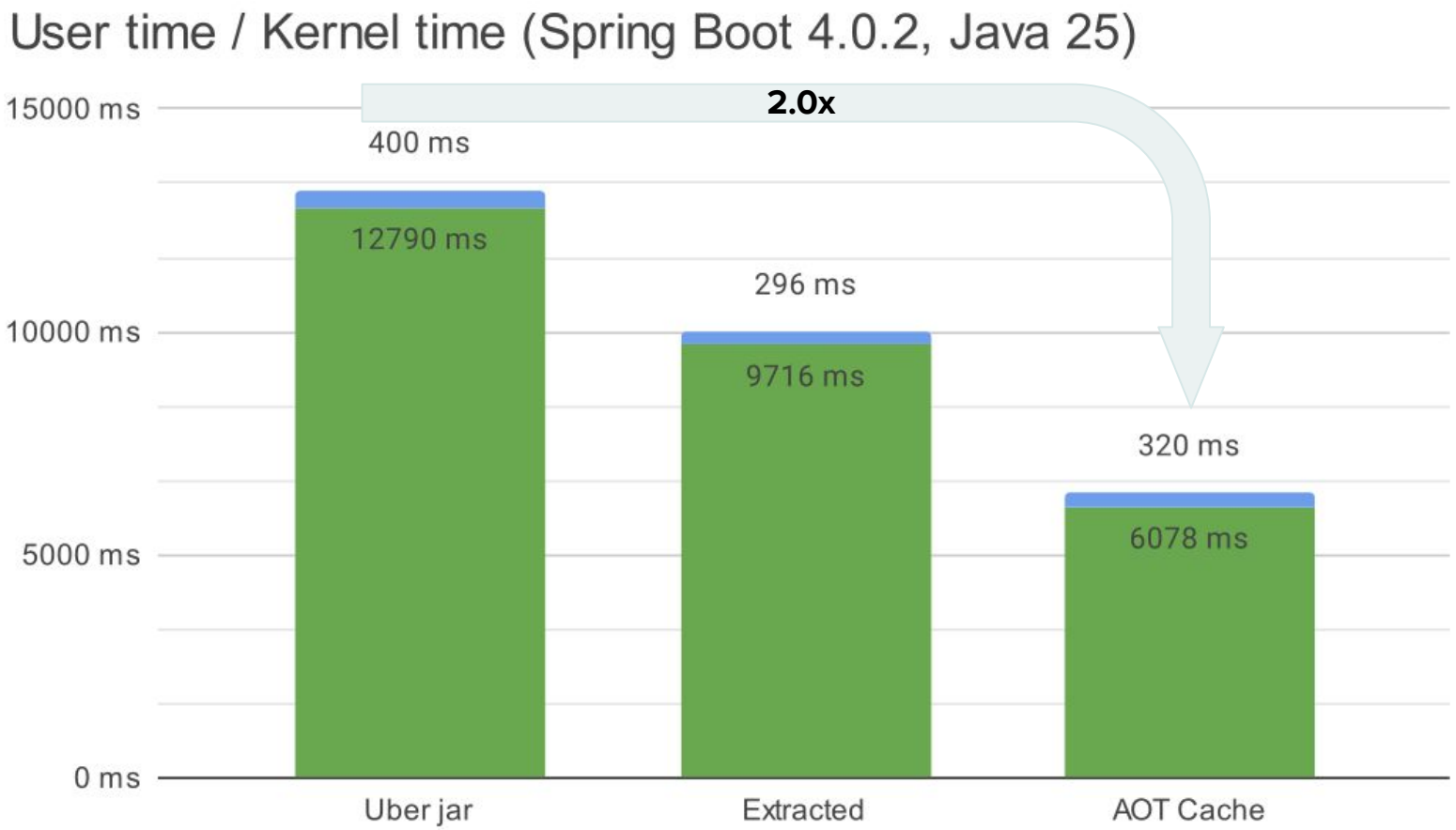
```
java -XX:AOTCache=extracted/app.aot -jar extracted/spring-petclinic-4.0.0-SNAPSHOT.jar
```


Time to first request (Spring Boot 4.0.2, Java 25)
3.1x
3000 ms
2000 ms
1000 ms
0 ms
2982 ms
2407 ms
958 ms
Uber jar
Extracted
AOT Cache

spring


RSS (Spring Boot 4.0.2, Java 25)
600 MiB
400 MiB
200 MiB
0 MiB
500.86 MiB
430.74 MiB
440.62 MiB
Uber jar
Extracted
AOT Cache

spring


User time / Kernel time (Spring Boot 4.0.2, Java 25)
15000 ms
10000 ms
5000 ms
0 ms
2.0x
400 ms
12790 ms
296 ms
9716 ms
320 ms
6078 ms
Uber jar
Extracted
AOT Cache

AOT Cache and Spring AOT

AOT Cache

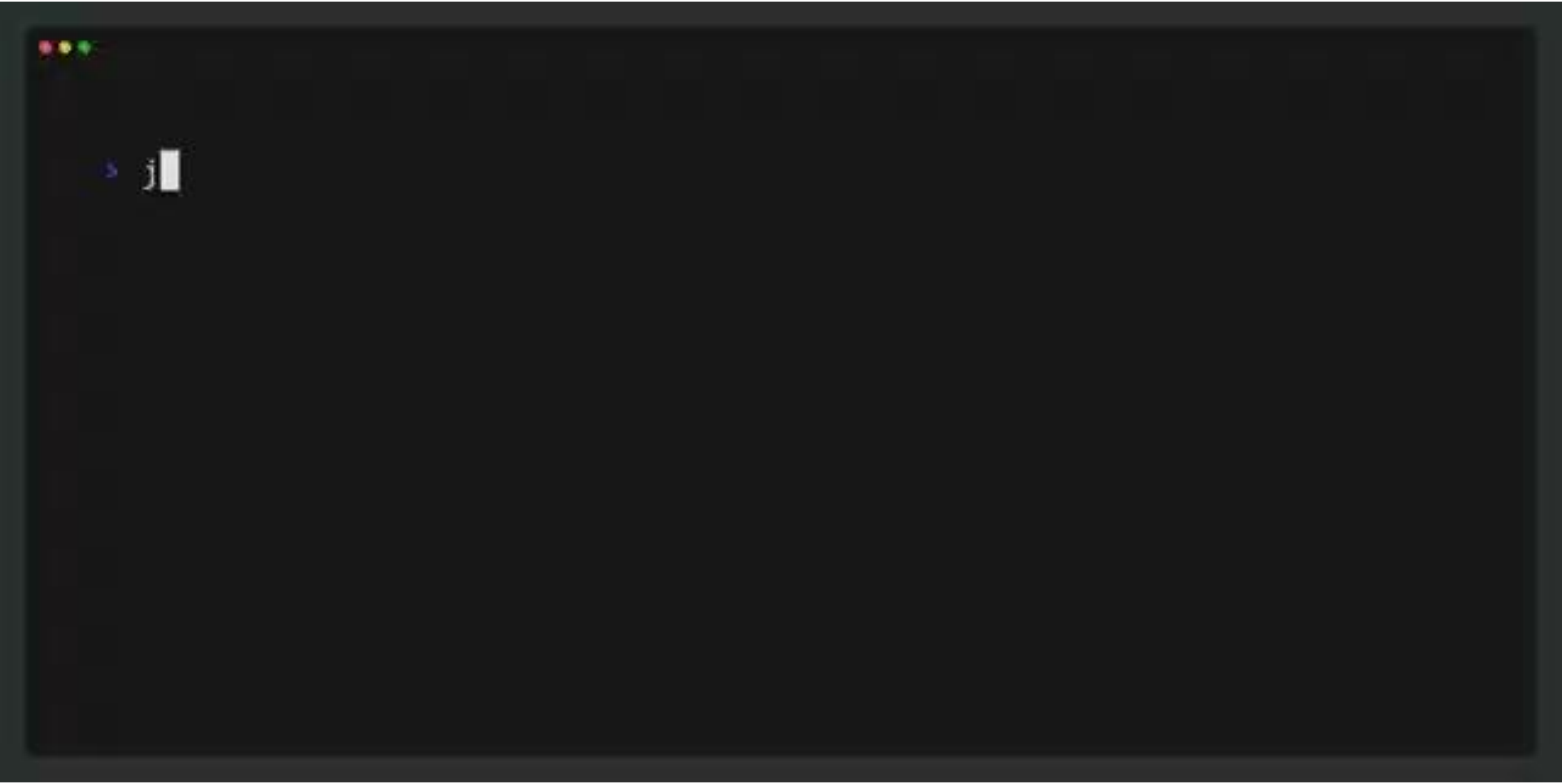
JVM feature developed via Project Leyden to improve the efficiency of the JVM. It supersedes CDS.

Spring AOT

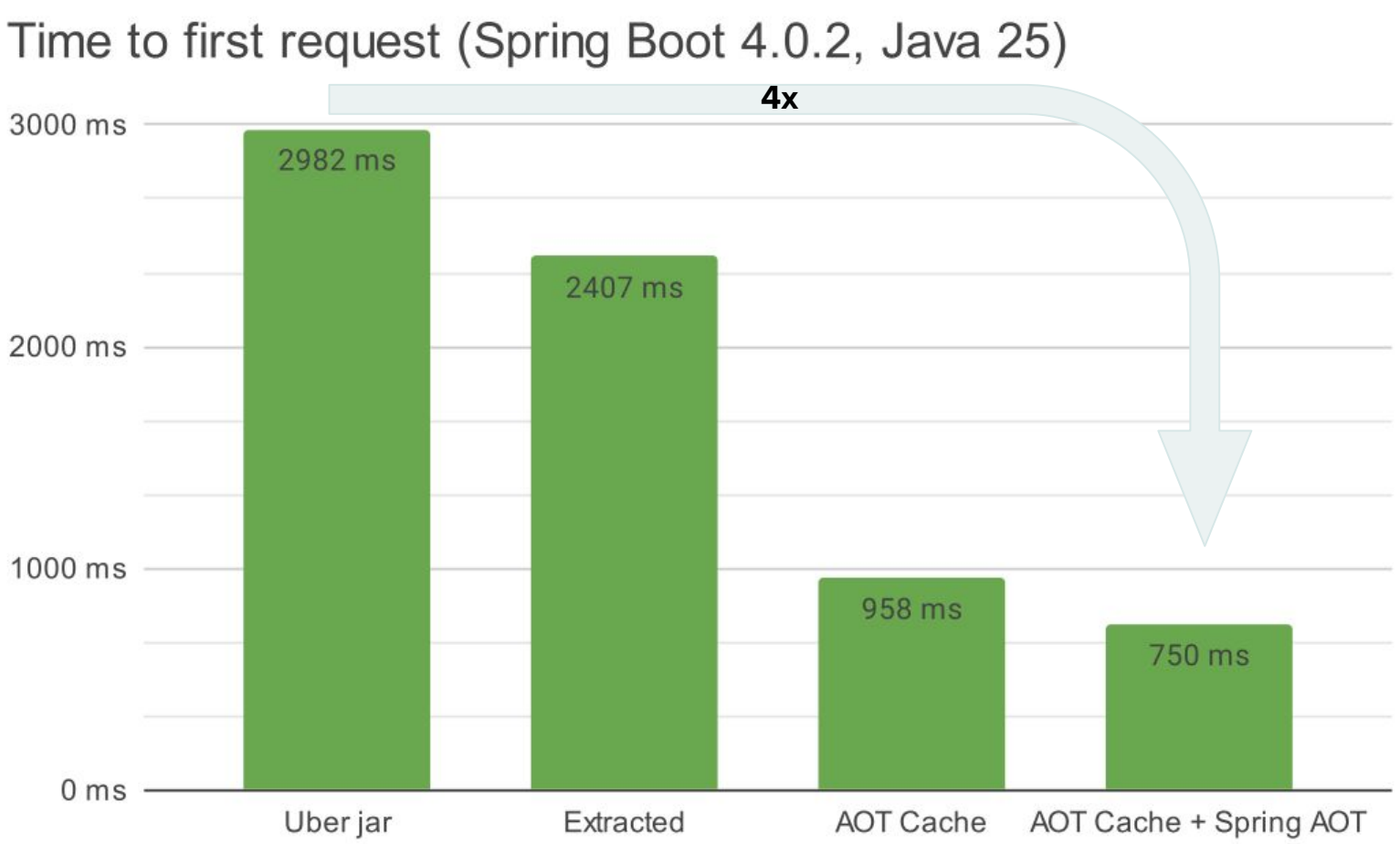
Spring feature mandatory for GraalVM native images Support.

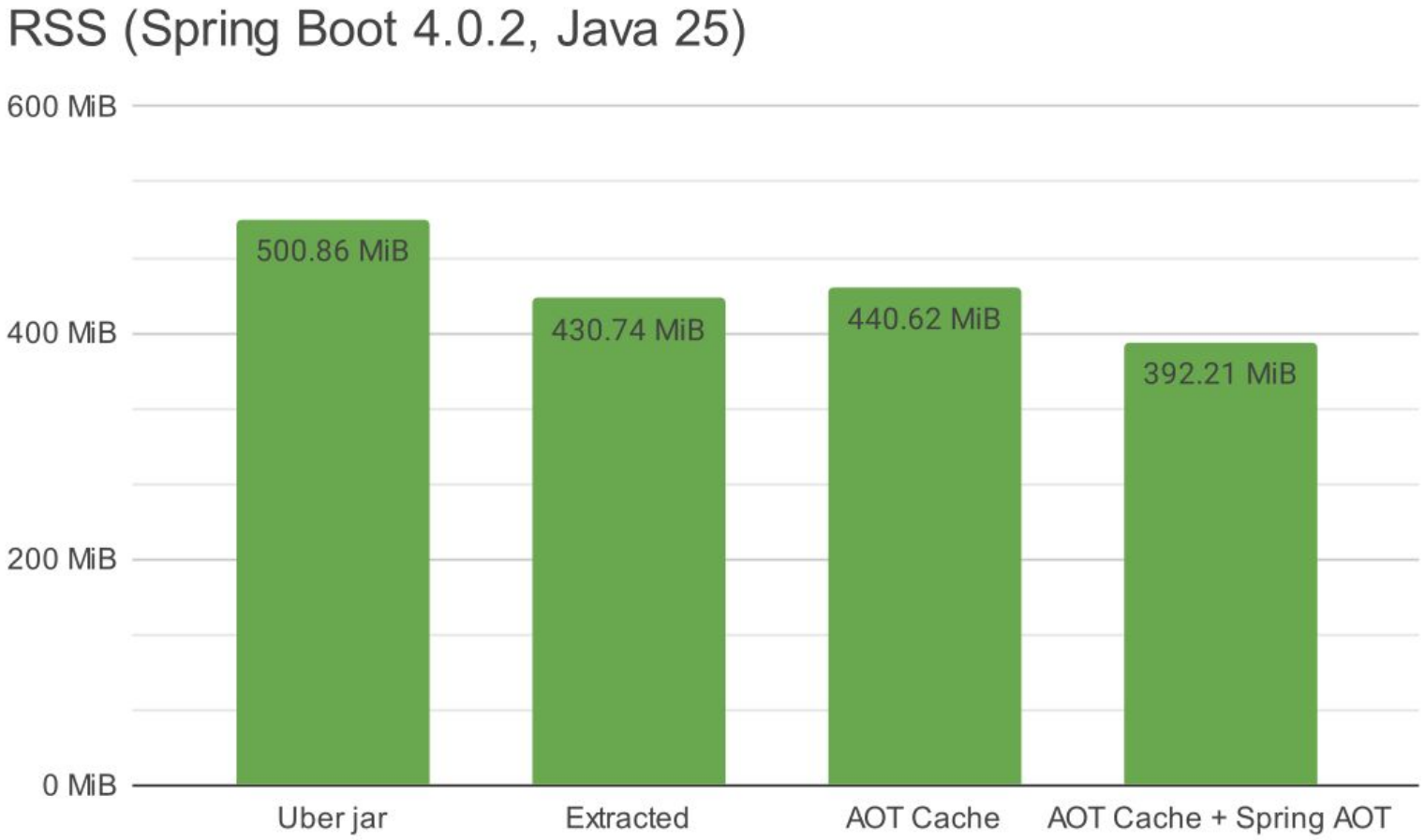
Can also be used on the JVM to speed up the startup process and lower the memory consumption.

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

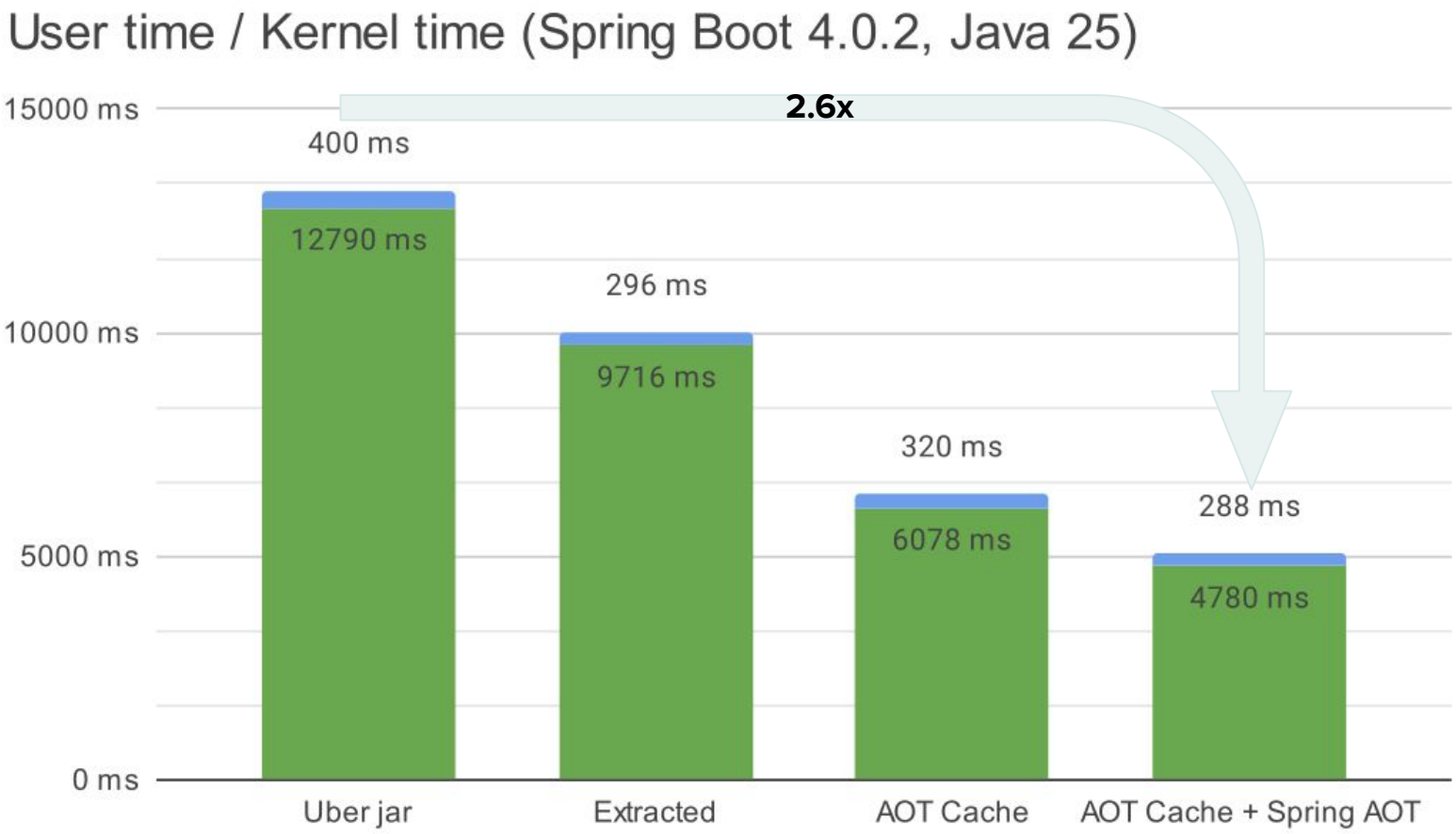


```
java -XX:AOTCache=extracted-aot/app.aot -Dspring.aot.enabled=true -jar extracted-aot/spring-petclinic-4.0.0-SNAPSHOT-aot.jar
```


Time to first request (Spring Boot 4.0.2, Java 25)
4x
3000 ms
2000 ms
1000 ms
0 ms
2982 ms
2407 ms
958 ms
750 ms
Uber jar
Extracted
AOT Cache
AOT Cache + Spring AOT


RSS (Spring Boot 4.0.2, Java 25)
600 MiB
400 MiB
200 MiB
0 MiB
500.86 MiB
430.74 MiB
440.62 MiB
392.21 MiB
Uber jar
Extracted
AOT Cache
AOT Cache + Spring AOT

spring


User time / Kernel time (Spring Boot 4.0.2, Java 25)
15000 ms
10000 ms
5000 ms
0 ms
2.6x
400 ms
12790 ms
296 ms
9716 ms
320 ms
6078 ms
288 ms
4780 ms
Uber jar
Extracted
AOT Cache
AOT Cache + Spring AOT

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
“Poor man’s training run”

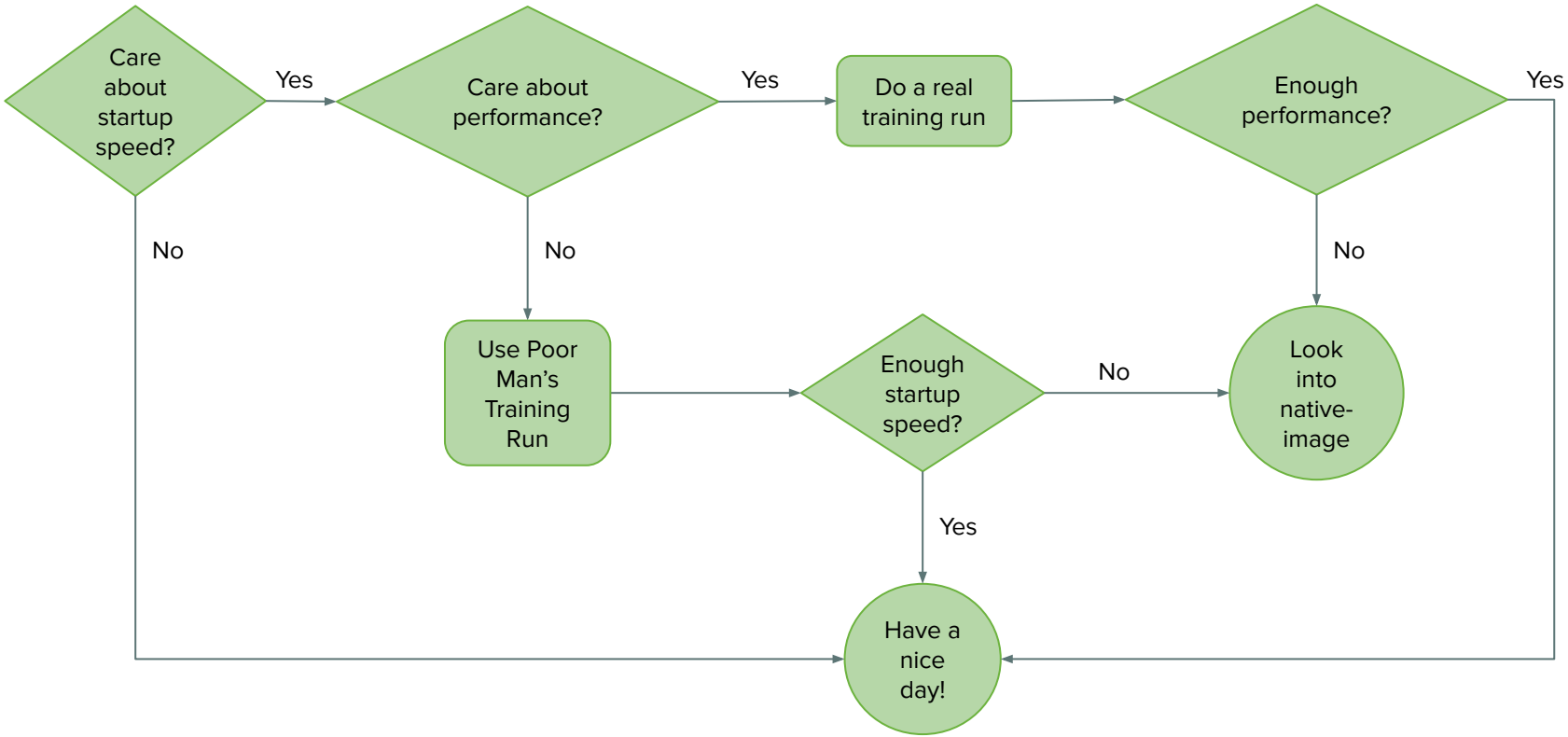
- Easy setup
- Can be done while building the image
- Improves start-up time by caching initial class loading

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

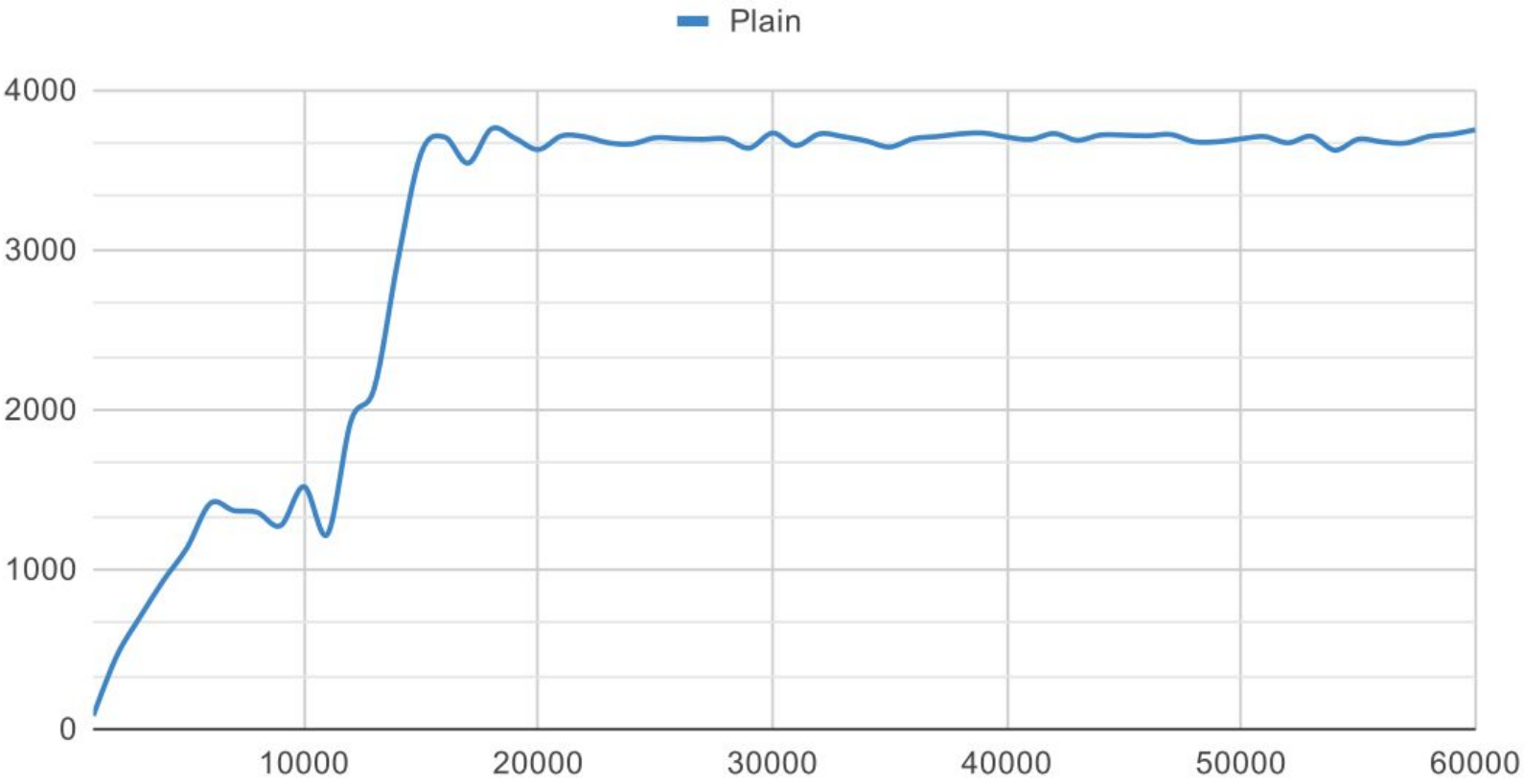
Run integration tests / mirror production traffic

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

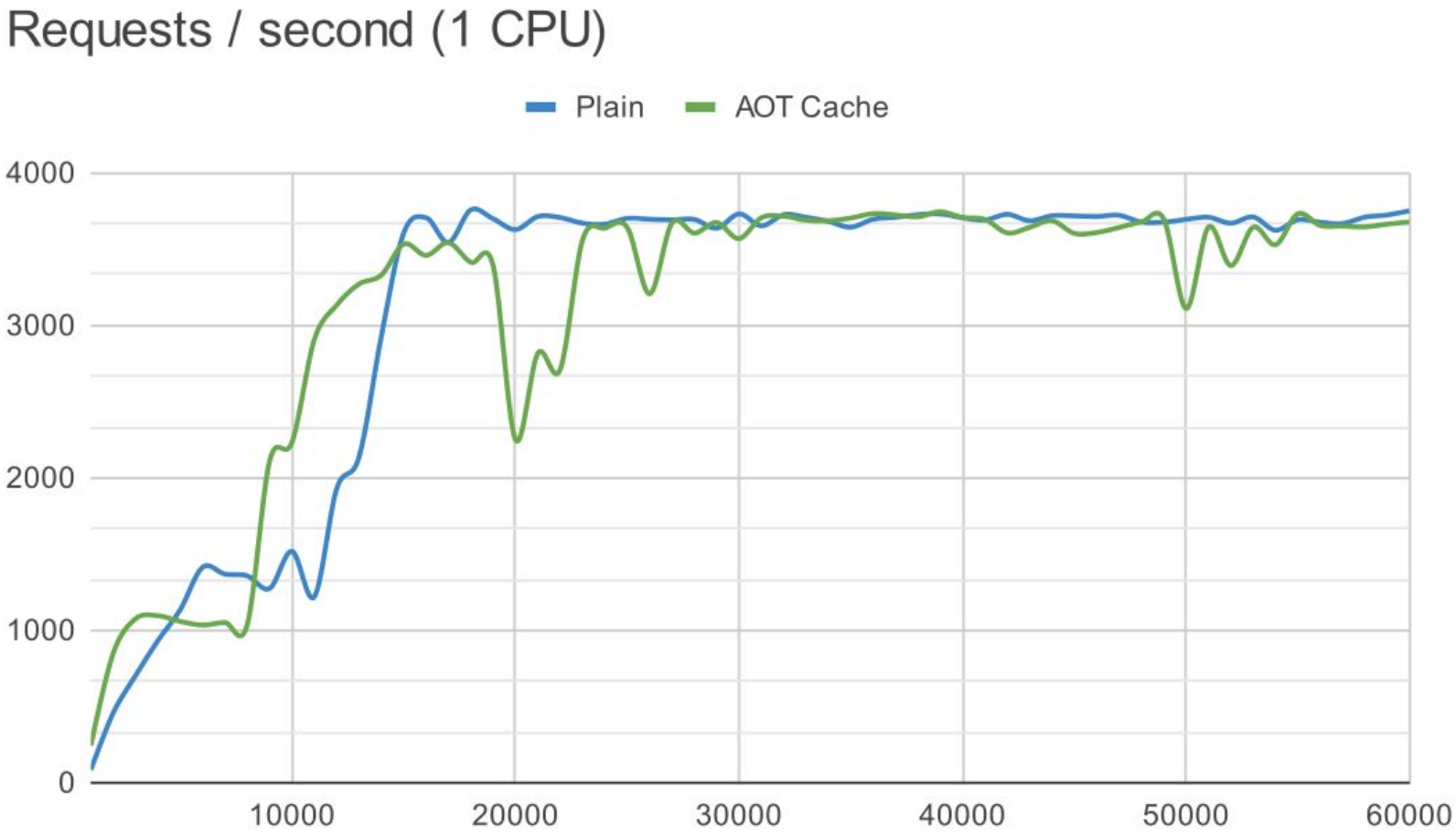
- Can’t easily be done while building the image
- More involved setup


Care about startup speed?
Yes
Care about performance?
Yes
Do a real training run
Enough performance?
Yes
No
No
No
Use Poor Man's Training Run
Enough startup speed?
No
Look into native-image
Yes
Have a nice day!

Requests / second (1 CPU)



spring


Requests / second (1 CPU)
Plain
AOT Cache
4000
3000
2000
1000
0
10000
20000
30000
40000
50000
60000

All of this is already possible today using Java 25!

What’s next?

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

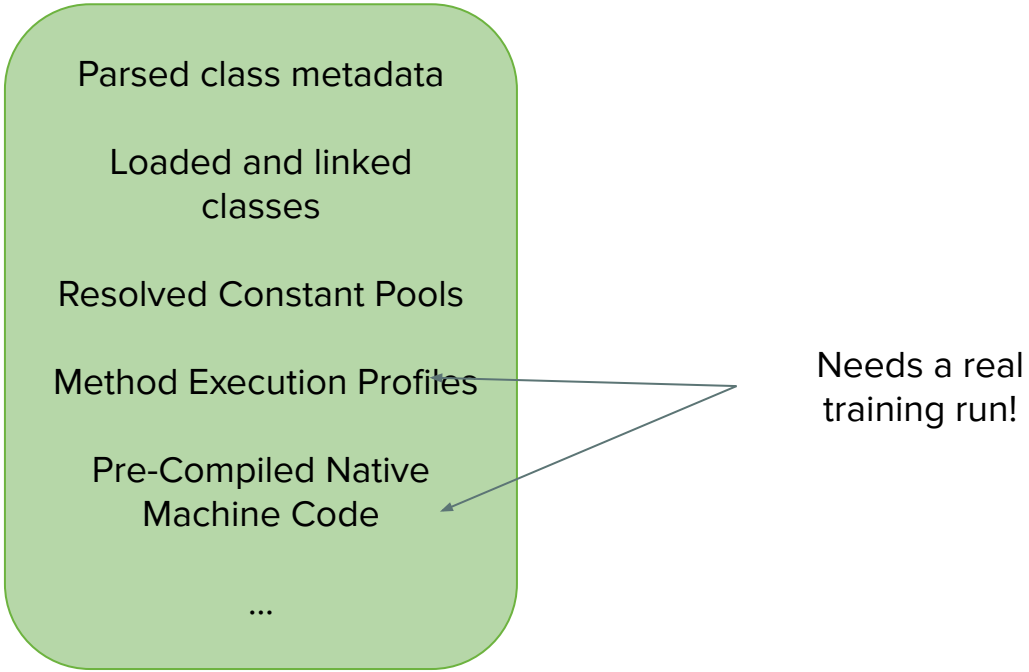
JEP draft: Ahead-of-Time Code Compilation

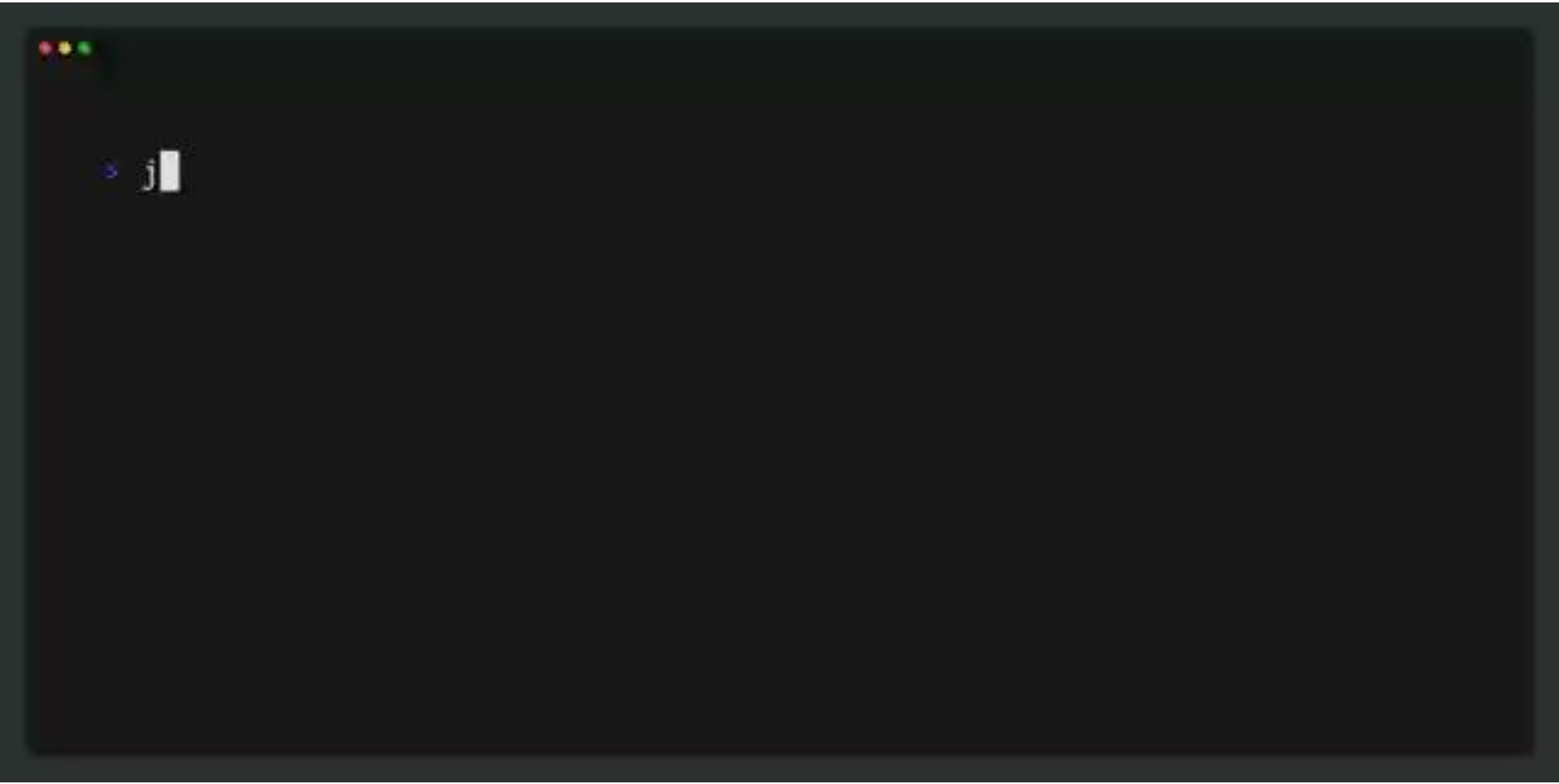
Owner	John Rose
Type	Feature
Scope	Implementation
Status	Draft
Component	hotspot / compiler
Created	2024/06/30 04:47
Updated	2025/10/09 16:21
Issue	8335368

Summary

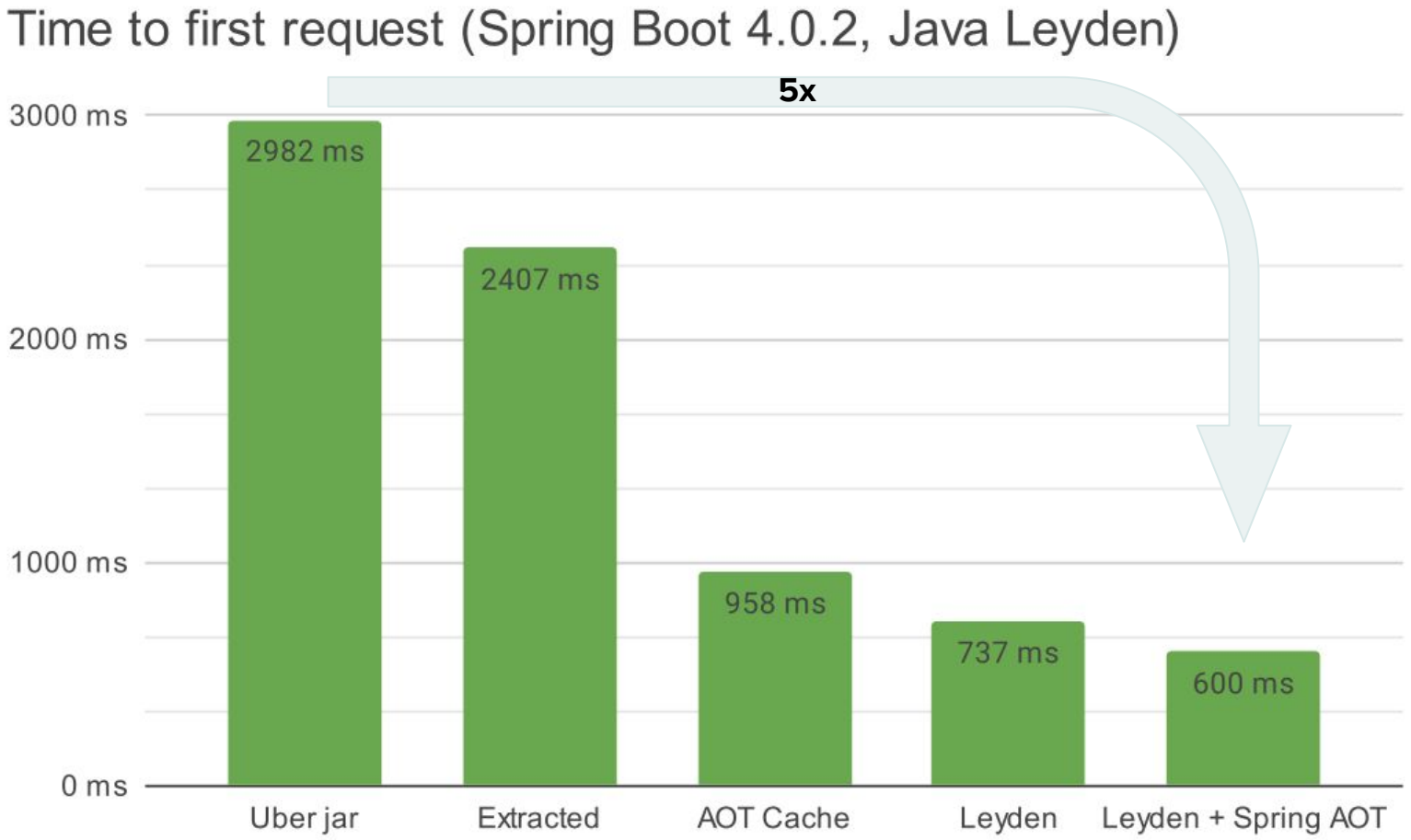
Improve startup and warmup time by making native code from a previous run of an application instantly available, when the HotSpot Java Virtual Machine starts. This will greatly reduce the initial load on the JIT compiler, reducing its interference with the application during startup, particularly in 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.

Project Leyden: What's in the cache?

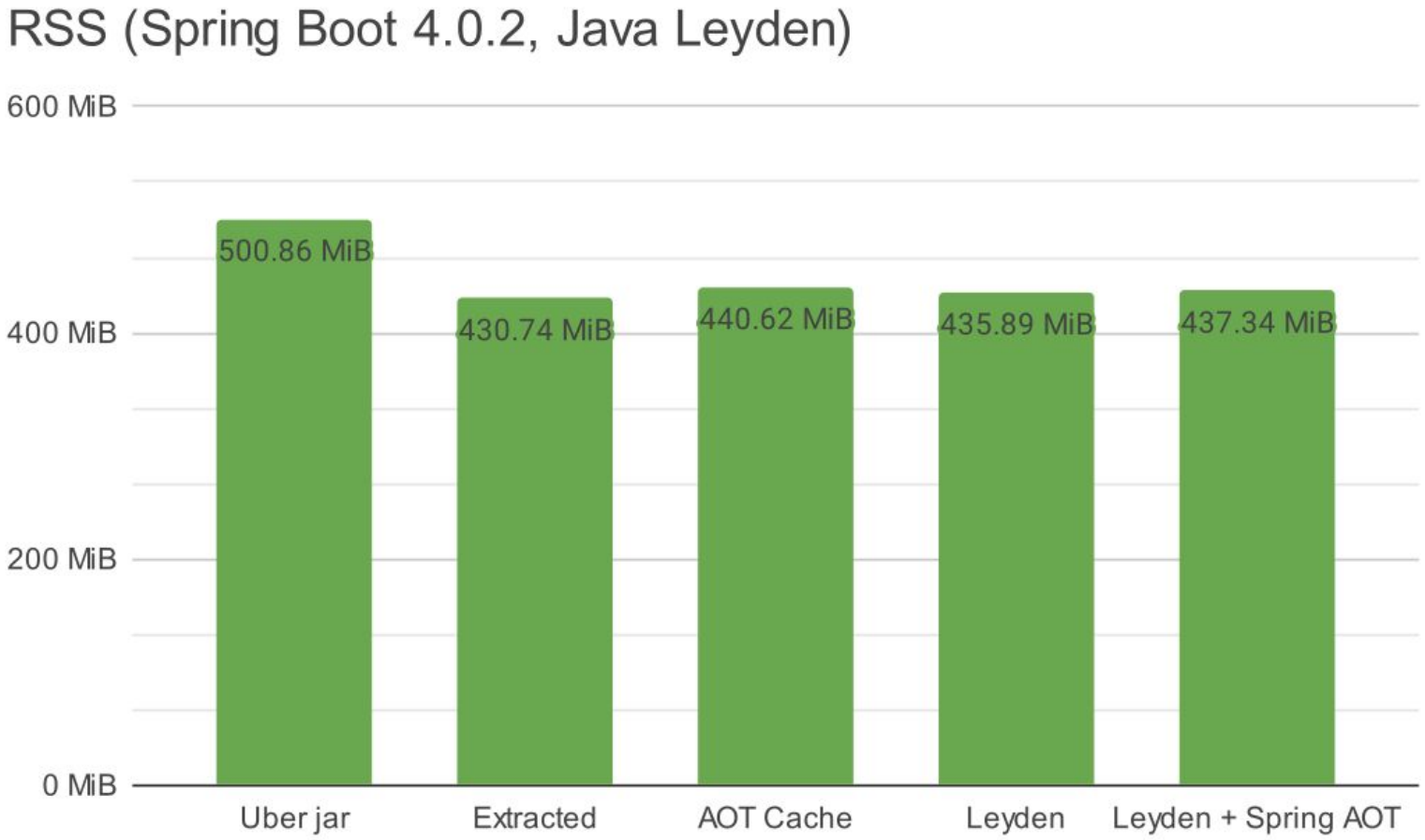




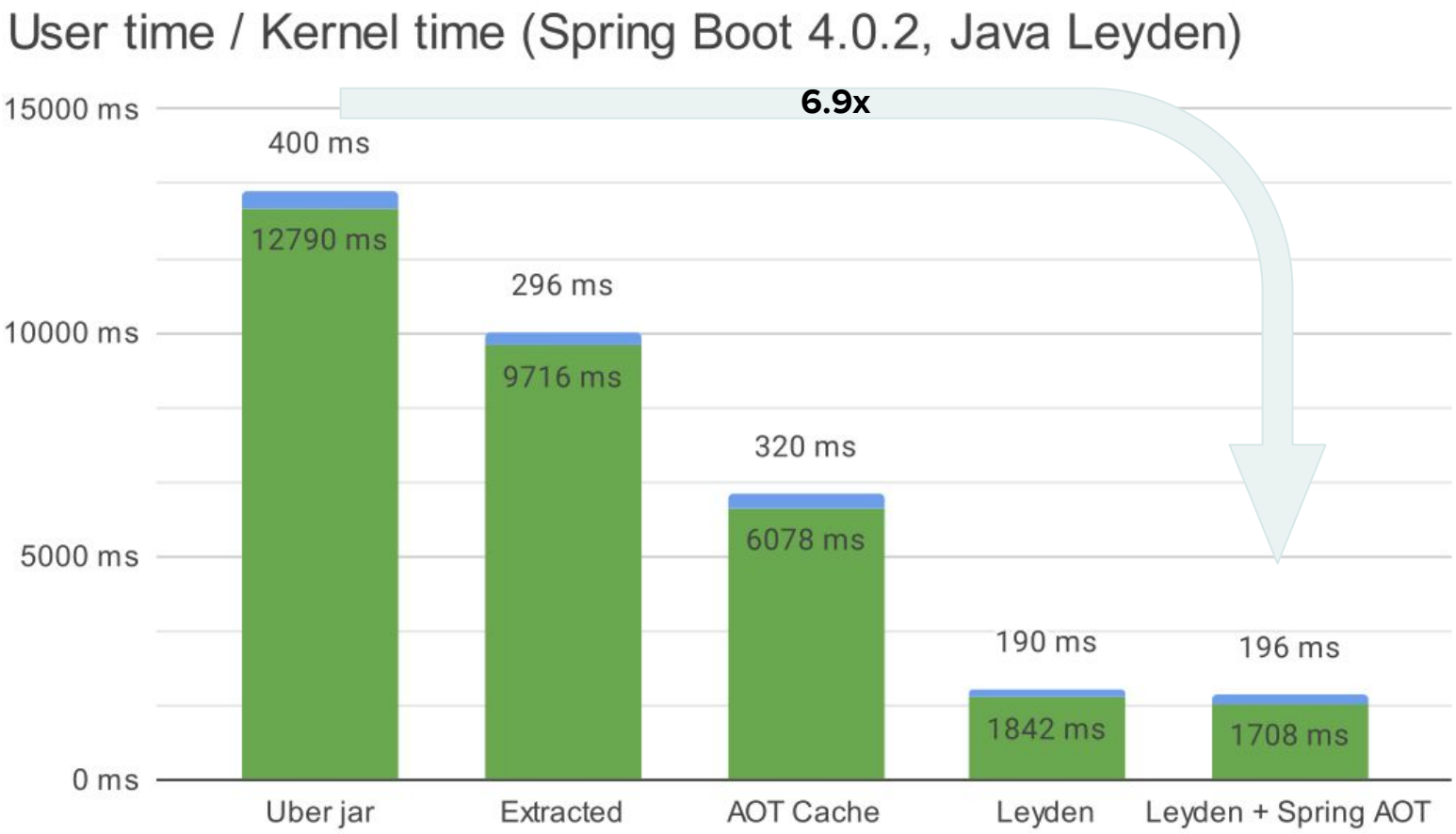

```
java -XX:AOTCache=extracted-aot/app-leyden.aot -Dspring.aot.enabled=true -jar extracted-aot/spring-petclinic-4.0.0-SNAPSHOT-aot.jar
```



spring Leyden built from git commit 9f2ebf85cbef2011ed7080a3ee445fc63594b85b

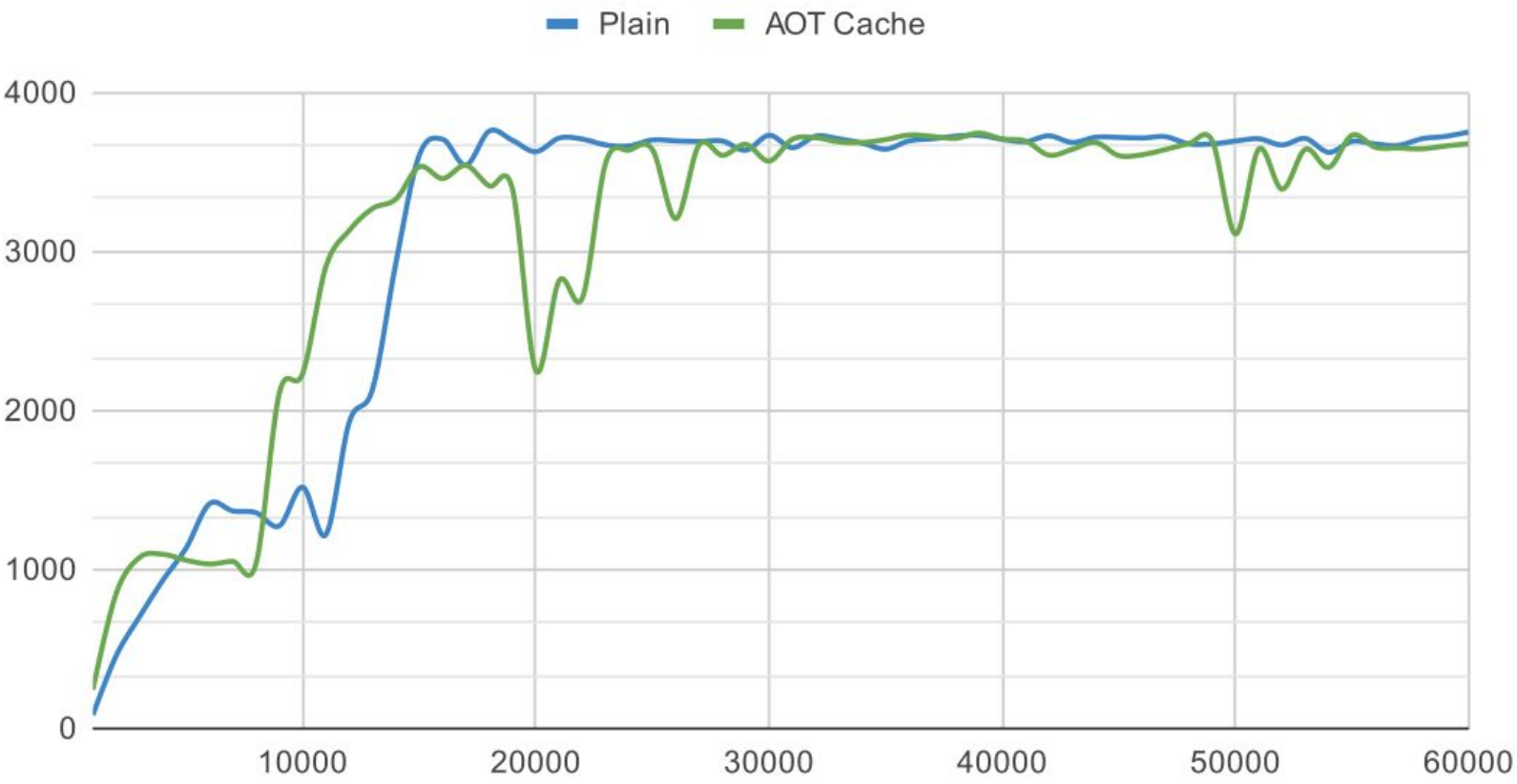

RSS (Spring Boot 4.0.2, Java Leyden)
600 MiB
400 MiB
200 MiB
0 MiB
500.86 MiB
430.74 MiB
440.62 MiB
435.89 MiB
437.34 MiB
Uber jar
Extracted
AOT Cache
Leyden
Leyden + Spring AOT

spring


User time / Kernel time (Spring Boot 4.0.2, Java Leyden)
6.9x
15000 ms
10000 ms
5000 ms
0 ms
400 ms
12790 ms
296 ms
9716 ms
320 ms
6078 ms
190 ms
1842 ms
196 ms
1708 ms
Uber jar
Extracted
AOT Cache
Leyden
Leyden + Spring AOT

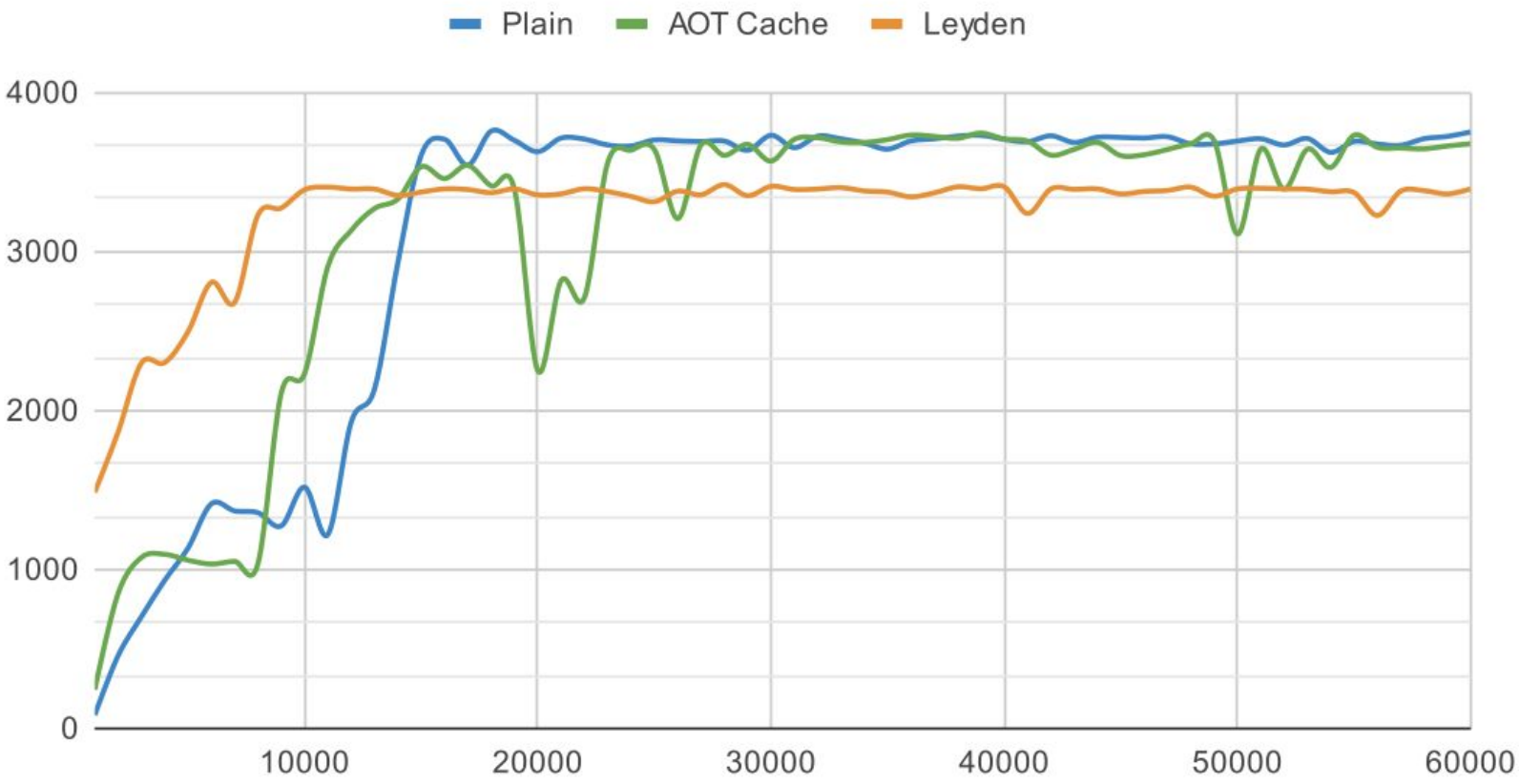
spring

Requests / second (1 CPU)



spring

Requests / second (1 CPU)



spring

https://bugs.openjdk.org/browse/JDK-8369735

https://bugs.openjdk.org/browse/JDK-8371086

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
}
```

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