QUARKUS

Quarkus and Leyden, the bumpy road ahead

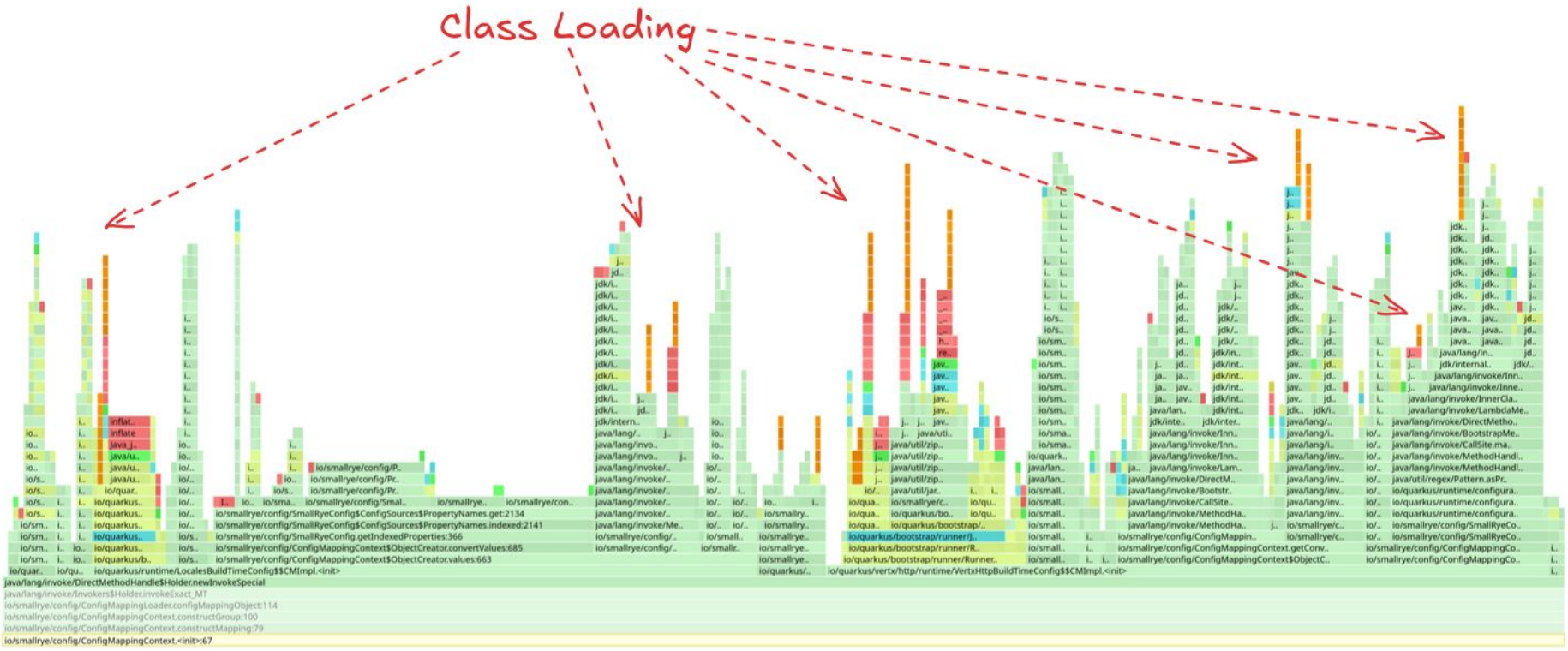
Guillaume Smet

gsmet@ibm.com

Hey Quarkus, can you start any faster?

Yes!

How it started?



How can we optimize Quarkus startup further?

We have no visibility!

Isn't Project Leyden supposed to help with class loading?

Could we profile application startup with Leyden enabled?

Let's stop there for a minute

AOT?

- AOT = Ahead of Time - vs JIT = Just in Time
- Shift computation from runtime to build time
- Do not necessarily mean compilation to native
- Do not necessarily mean closed world assumption
- Often causes some trade-offs as not everything is known at build time

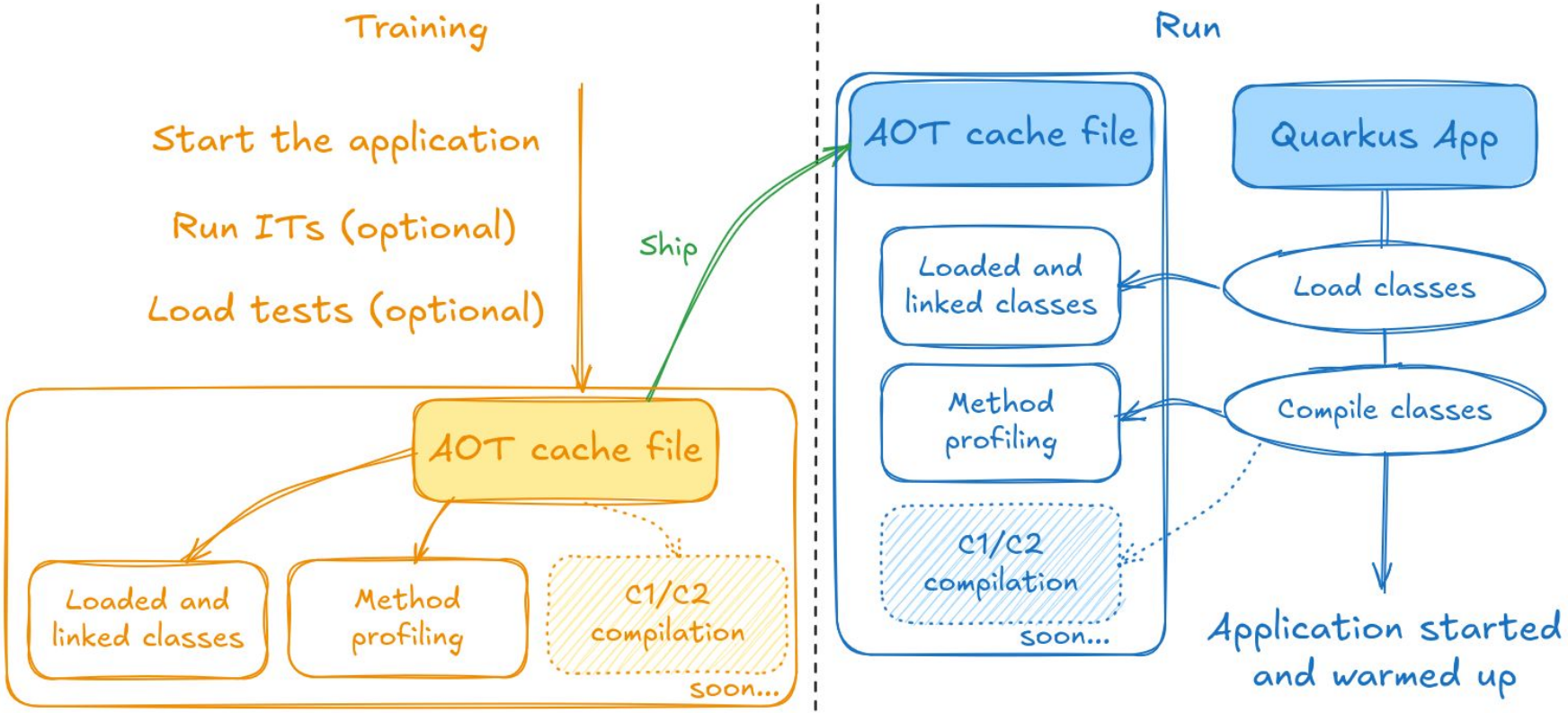
The (not so) brief history of Java AOT

- 1998 - GCJ: let's compile Java to native using GCC
- 2004 - CDS in 2004 in JDK 5 - for bootstrap classes, not real AOT
- 2007 - AOT compilation in IBM J9
- 2014 - AppCDS in JDK - opensourced in 2018
- 2018 - First public release of GraalVM (after years of research)
- 2018 - Quarkus development starts, first public release in 2019
- 2020 - Project Leyden approved
- 2025 - Project Leyden becomes a reality: JEP 483 in JDK 24

What is Project Leyden?

- Ahead of Time (AOT) for OpenJDK 24+
- AOT, yes, but still Java (i.e. a dynamic language)
  - With GraalVM, if a class is not known AOT, error
  - With Leyden, if a class it not known AOT, we try to load it
- AOT, sure, but you keep all the goodness from OpenJDK:
  - True Java, no weirdness
  - Amazing JIT
  - All GCs

What is Project Leyden?



What does Leyden bring?

- Faster startup - classes are loaded and linked in the AOT cache file
- Faster warmup - Work in progress
- RSS improvements - Future goal

- More clarity: no more profiles plagued with class loading

Leyden constraints

- Easy to train! But...
  - Exact same JDK version between build and run
  - Classes have to be loaded from a JDK class loader (no custom class loader for you!) - WIP
  - Future: when using compile cache, CPU capabilities need to be exactly the same - WIP

Our first experiments

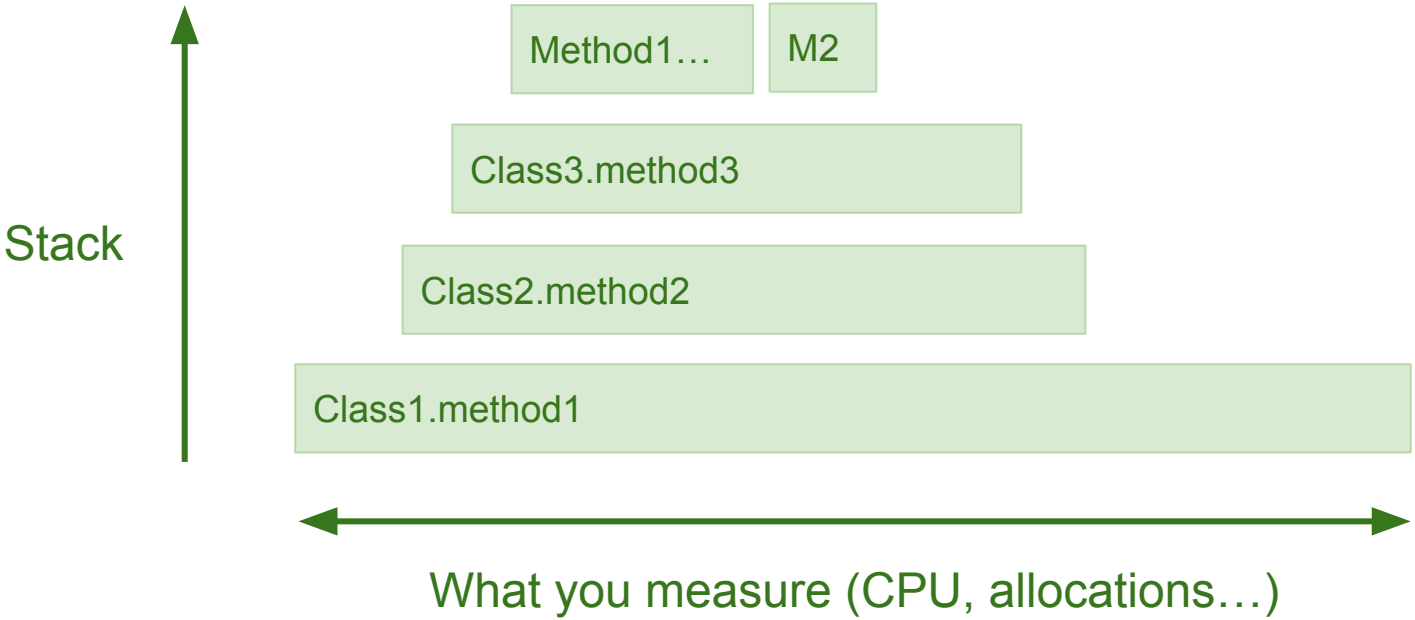
- Remember: our goal was only to get more clarity in profiling data
- REST App startup time: ~ 400 ms
- With Project Leyden: ~ 200 ms - custom class loader cough cough
- With Project Leyden + legacy jar: ~ 130 ms

That's when we understood we could get more from this initiative.

Our road to faster startup

Profiling, more profiling (and flaaaaamegraaaaphs!)

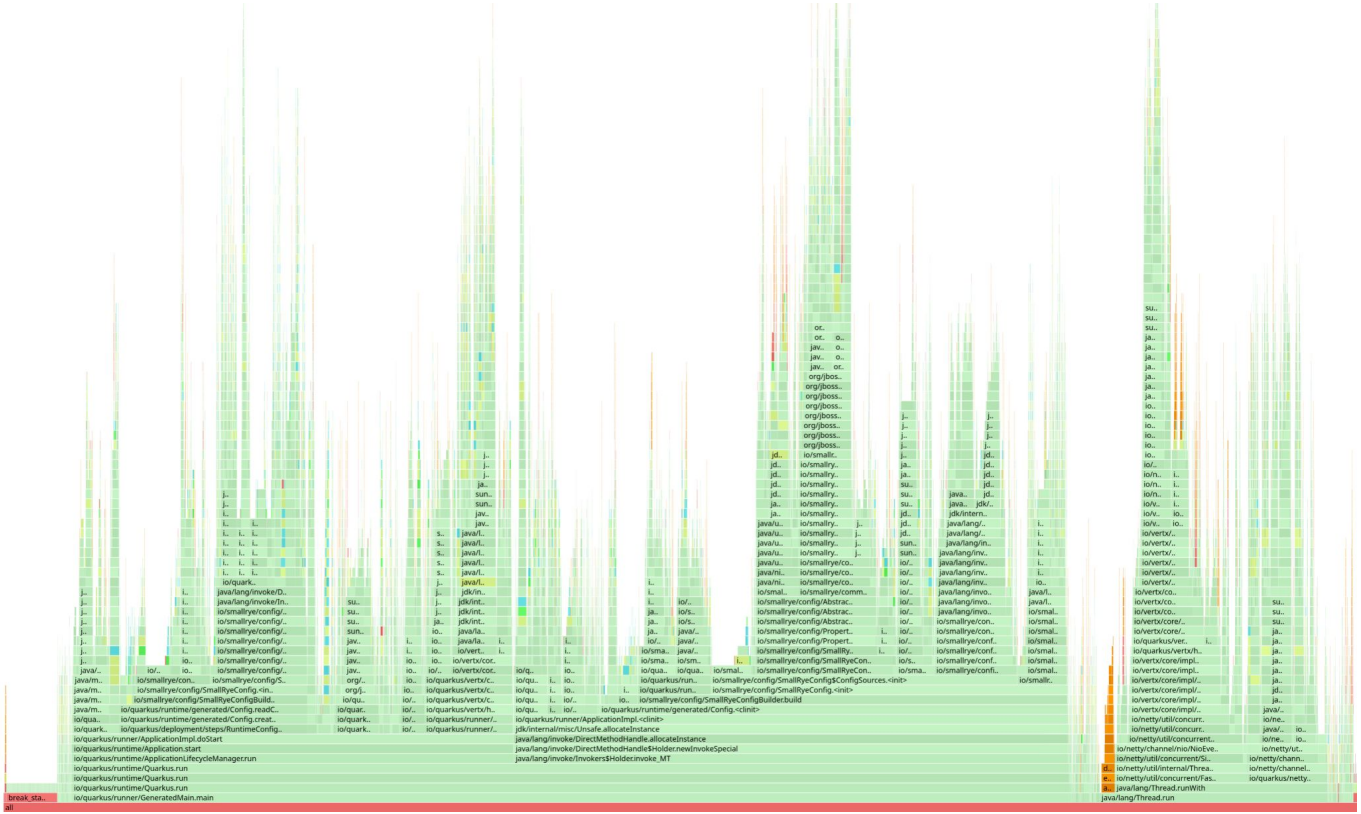
Flame... what?


Method1…
M2
Class3.method3
Class2.method2
Class1.method1
Stack
What you measure (CPU, allocations…)

We are in < 100 ms territory
Every milliseconds matters

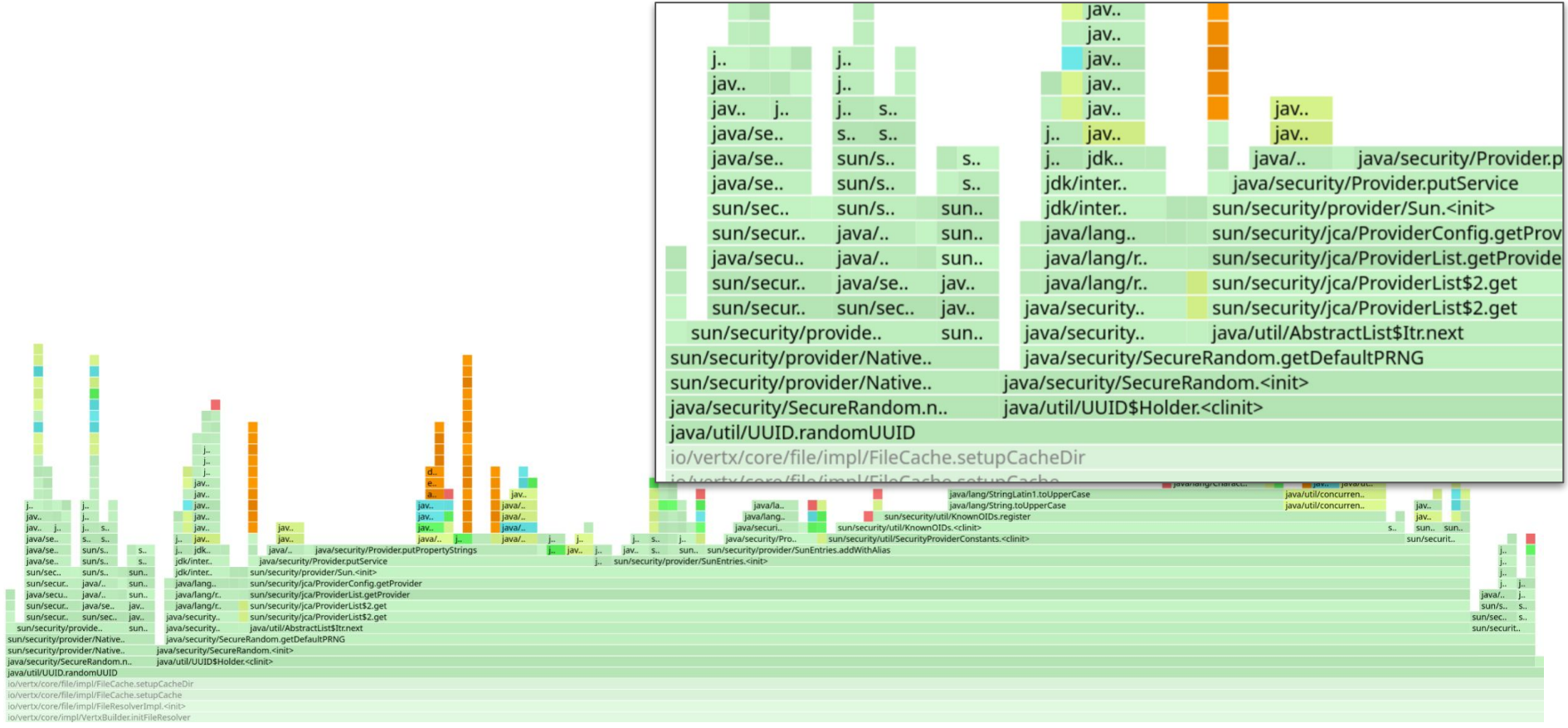
Let's start with a simple REST app

More clarity



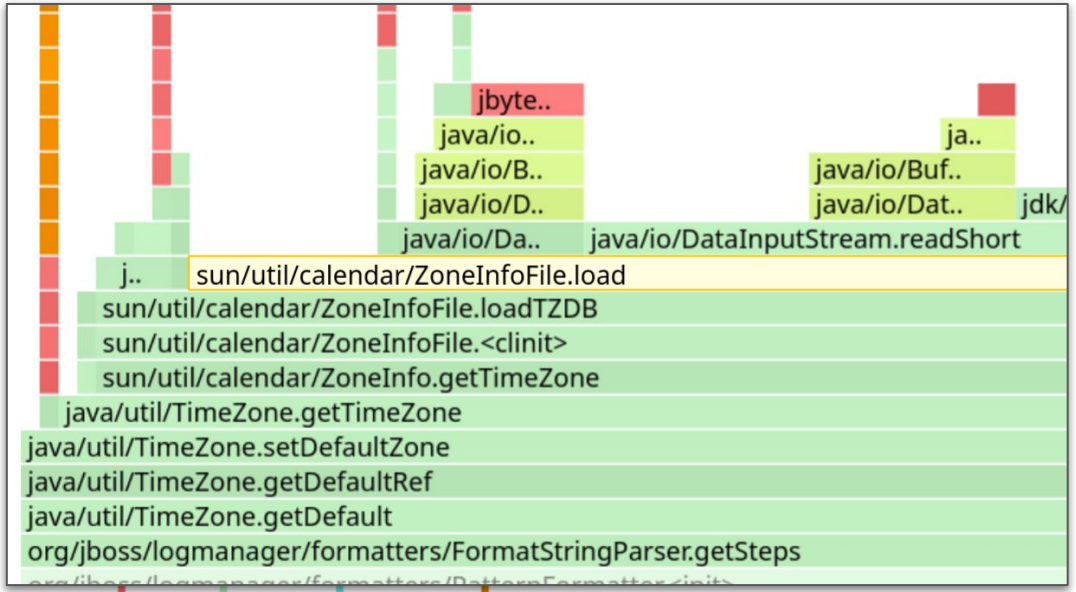
BigDecimal <clinit>

UUID generation



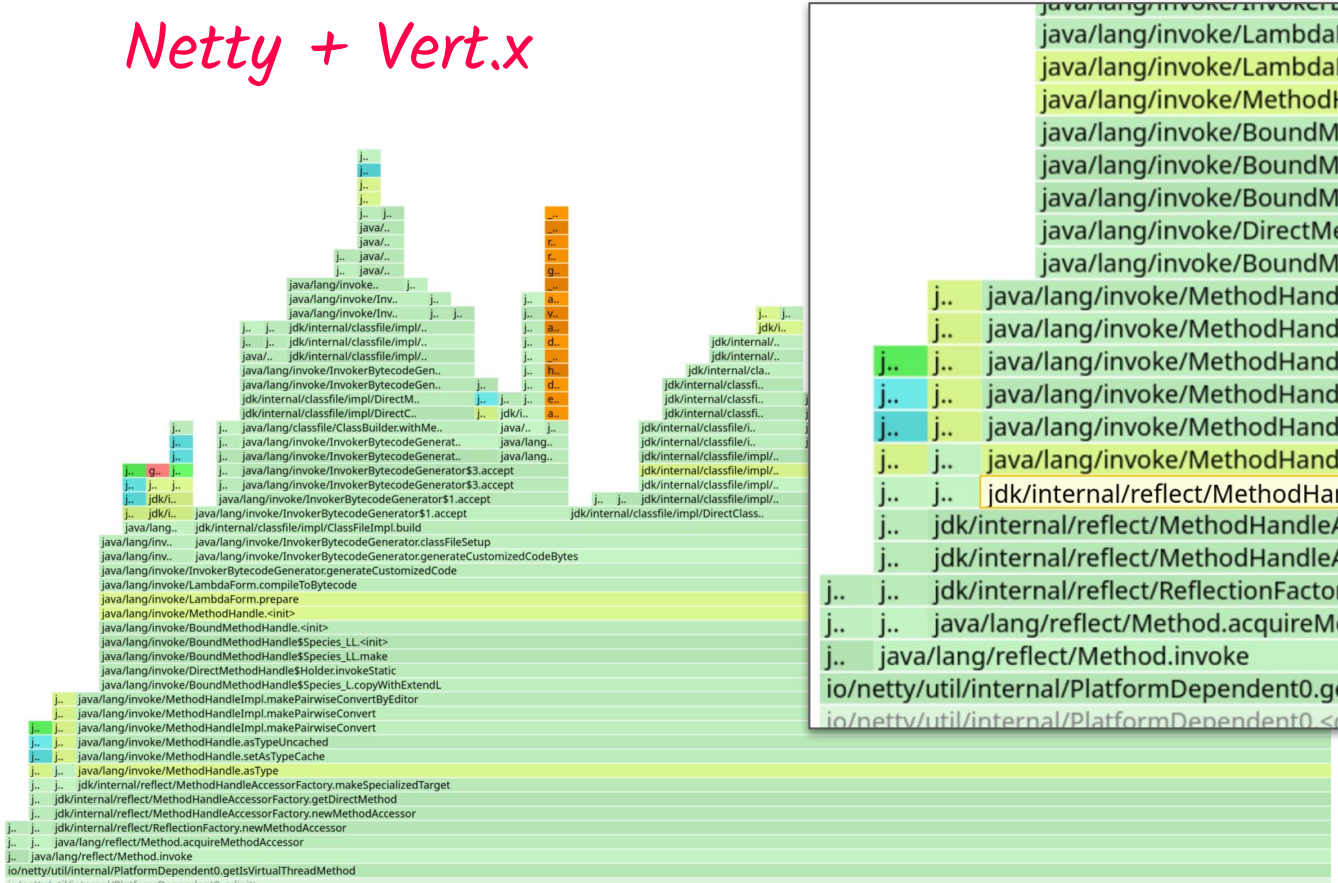
Loading time zone database

+ Zone rules database

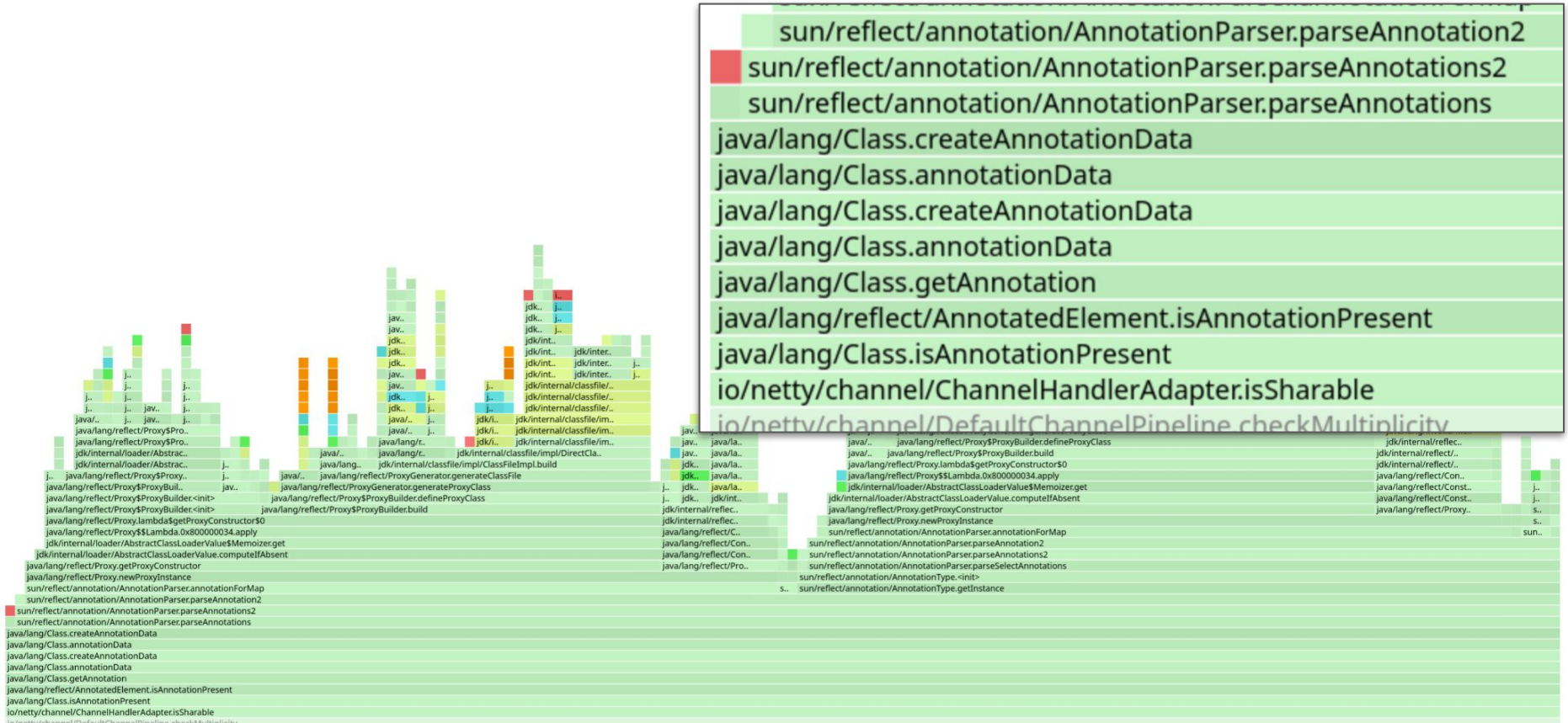


Compatibility layers via reflection

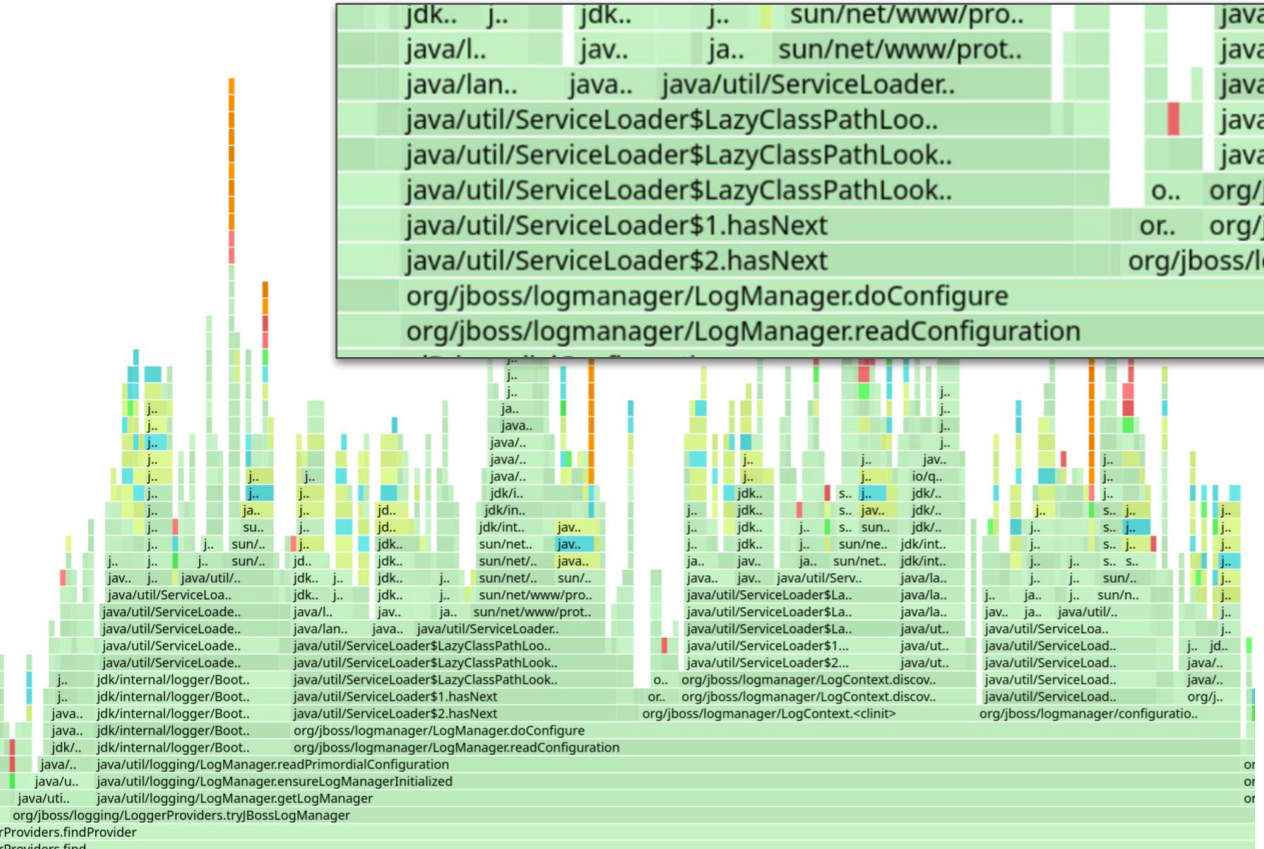
Netty + Vert.x



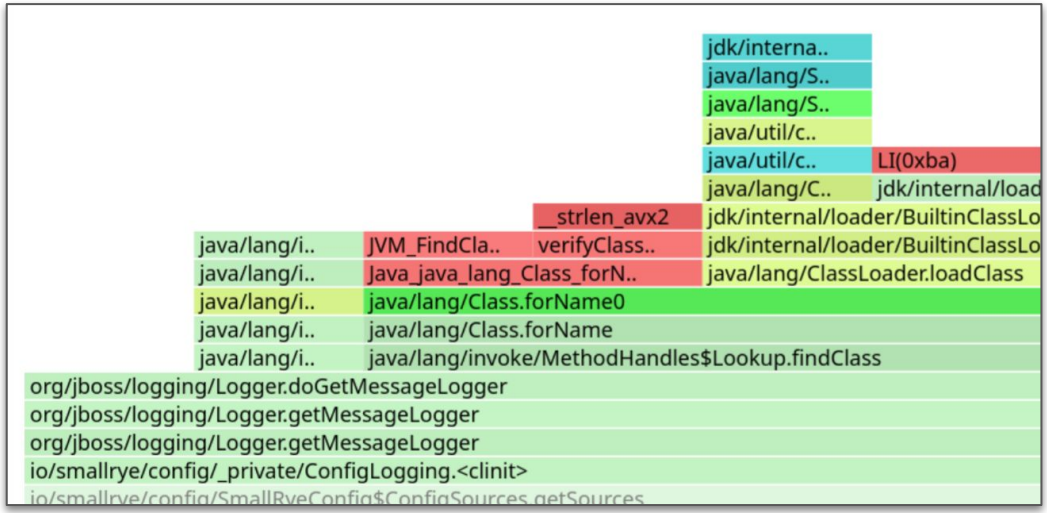
Marker annotations



Service loader



Loading non-existent classes



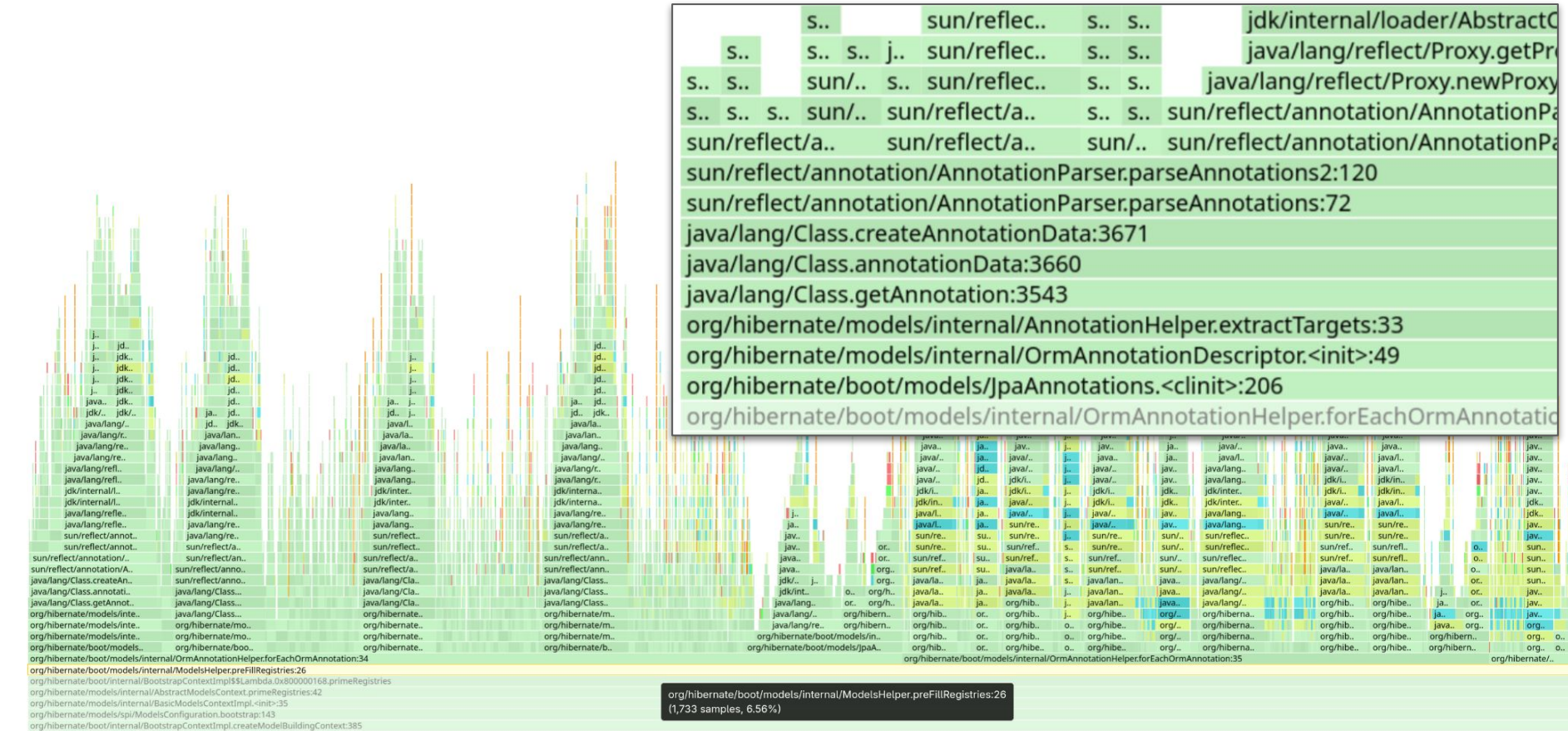
Negative lookups

- Remember, Project Leyden is not a closed world
- Negative lookups are not cached = we need to look in the classpath
- If the class is not there, we go through the whole classpath to figure out it's not there

It has a cost.
But it is important as training is not comprehensive.

What about Hibernate ORM?

Reading annotations



Looking for package-info

Same approach for all key components and extensions

SmallRye Config, Quarkus Messaging...

Remember?
We knew it was great.

Polished Leyden integration

- A specific Jar packaging tailored to Leyden constraints (classes loaded from the Platform class loader + various optimizations)
- Full integration of the training run within our IT infra
- Inclusion of the cache file in the container images

It's one system property away.

Current status

Some numbers * - REST sample

Version	Startup time	Image size
Main - fast-jar	370 ms	456 MB
Main - aot-jar + Leyden	80 ms	495 MB
Mandrel native	17 ms	155 MB

Default application obtained with quarkus create app

Note: expect more improvements in the future

* *Numbers obtained unscientifically on a laptop, do not quote*

Some numbers - Large CRUD app

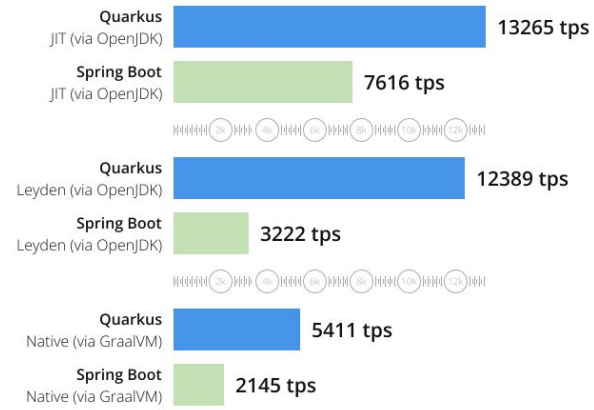
Version	Startup time	Image size
Main - fast-jar	3 189 ms	517 MB
Main - aot-jar + Leyden	924 ms	715 MB
Mandrel native	242 ms	244 MB

Very large CRUD app with 9000 .java files

* Numbers obtained unscientifically on a laptop, do not quote

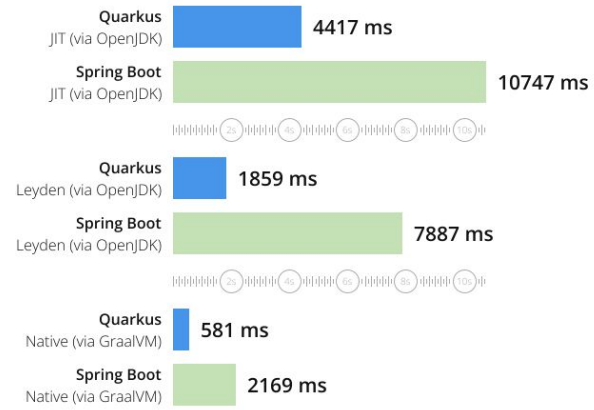
Throughput

(Higher is better)



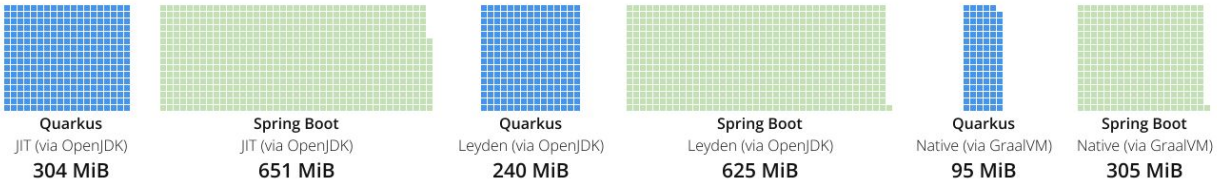
Boot + First Response Time

(Lower is better)



Memory (RSS after first request)

(Smaller is better)



Quarkus: 3.34.3
Spring Boot: 4.0.5
JVM: 25.0.2-tem
GraalVM: 25.0.2-graalce

Memory: -Xmx512m -Xms512m
JVM args: -XX:+UseNUMA
CPUS: 4

Source: quarkusio/spring-quarkus-perf-comparison
Scenario: tuned
Commit: b9a5812
Execution date: 2026-04-17

Fresh from the oven, take them with a grain of salt

Image size

- Has always been an issue and is amplificated by the cache file
- Tree shaking on build (included in Quarkus 3.35+)
- Jlink image packaging (included in Quarkus 3.37+)
  - Comes with ClassLoader challenges
  - Might not work very well with Leyden

Throughput

- Throughput with Leyden slightly lower
- Our performance team + IBM OpenJDK team on it

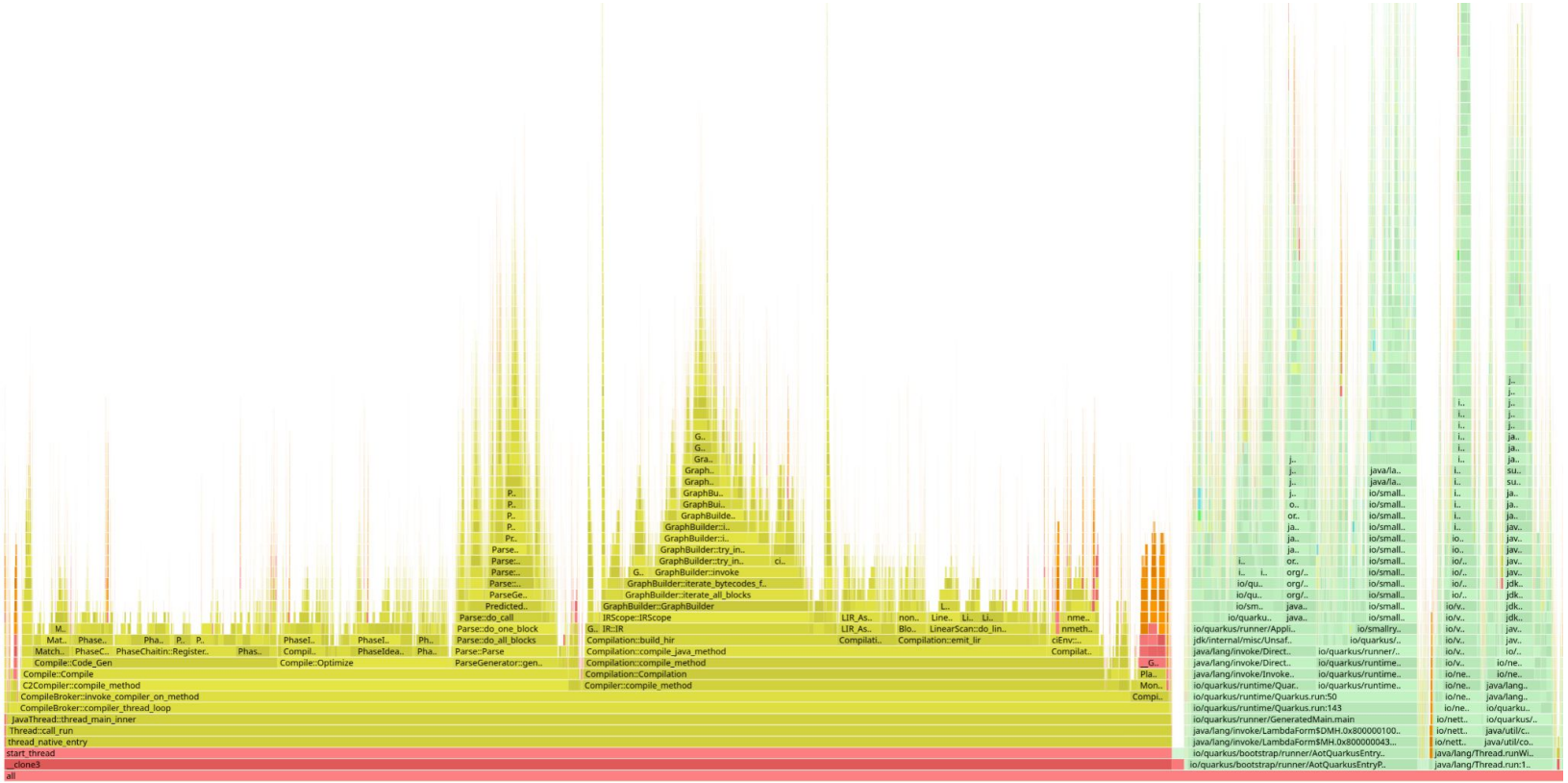
- https://bugs.openjdk.org/browse/JDK-8386852

Conclusion

Conclusion

- We now have clearly identified the cost of class loading and linking
- We have improved Quarkus startup time for all runtimes (and other components!)
- Still improving by extending this work to new extensions and components
- Project Leyden still under heavy work

The next frontier?



Thanks!


I'M LEAVING.
WHY?
BECAUSE PEOPLE KEEP ASKING ME QUESTIONS.
POORLY DRAWN LINES