bell//soft

HOTSPOT VS J9

WHICH ONE TO CHOOSE TO CUT YOUR CLOUD BILLS

Dmitry Chuyko

www.bell-sw.com | 2023

bellsoft

WHO WE ARE

Dmitry Chuyko

Performance architect with 20+ years of experience in IT

Active OpenJDK contributor

Areas of expertise:

- OpenJDK optimization for x86 and ARM
- Development of microcontainers for Java
- Enhancement of Java HotSpot

www.bell-sw.com

@dchuyko

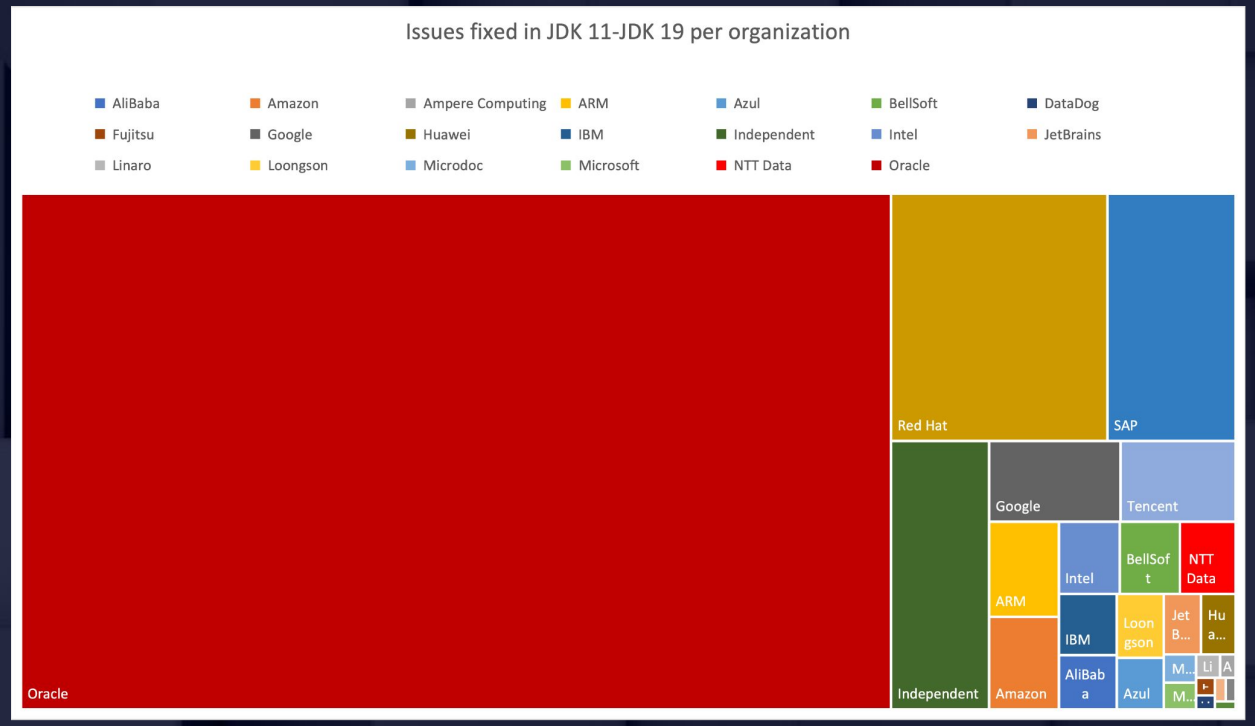
About BellSoft

BellSoft was founded in 2017 by Java and Linux experts with 10+ years of experience working in Sun/Oracle.
Headquarters in San Jose, California.

Members of:

- JCP executive committee
- OpenJDK Vulnerability Group
- GraalVM advisory board
- Linux Foundation
- Cloud Native Computing Foundation

bellsoft



BellSoft's contributions to OpenJDK

- Leading OpenJDK contributor
- Developed and integrated JEP 315 (aarch64 optimization) and JEP 386 (Alpine Linux port)
- Maintain the upstream Arm port
- Brought musl libc support in GraalVM

Our products

bellsoft

Java

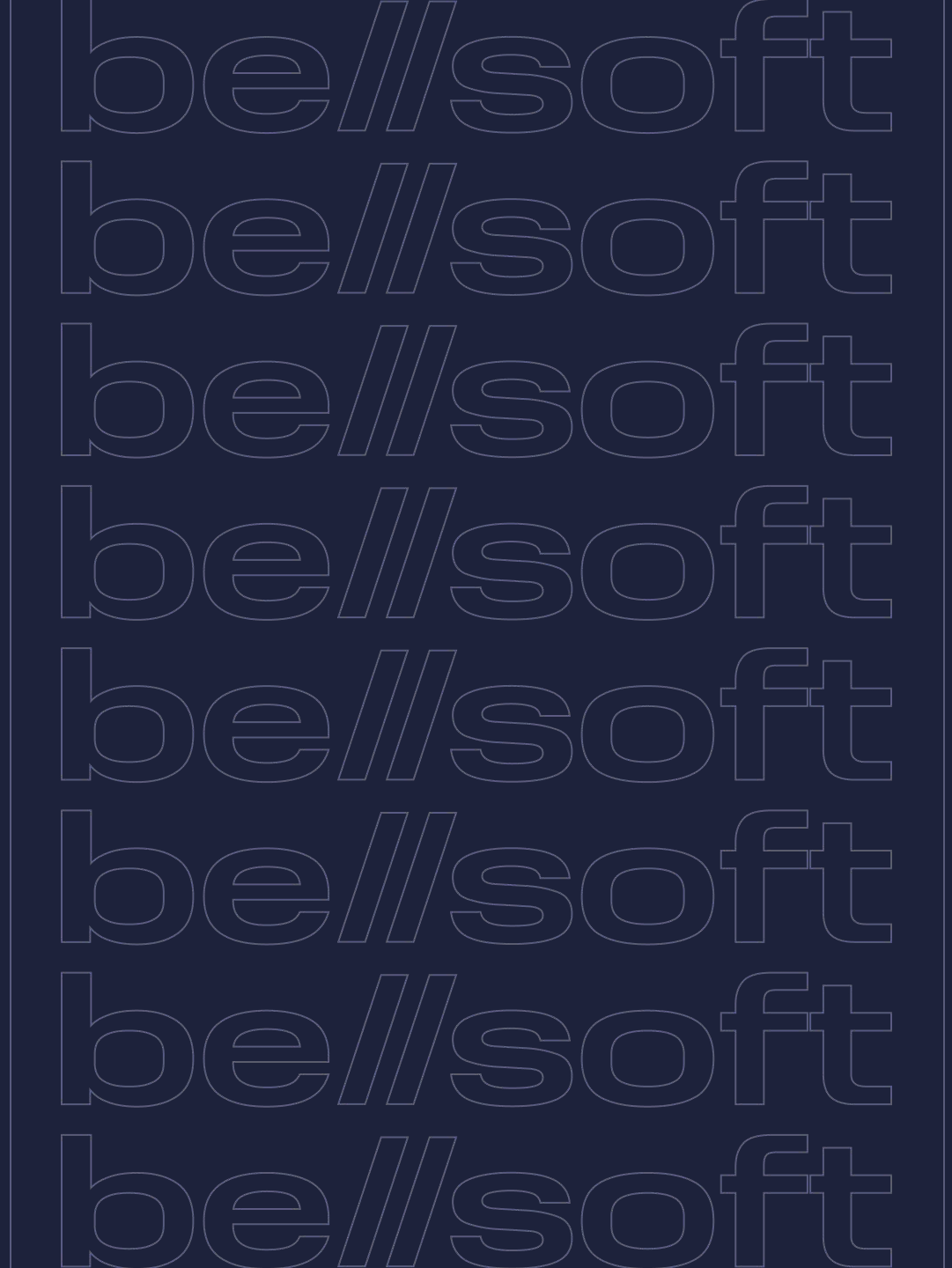
Liberica JDK is a 100% open-source vanilla Java 8, 11, and 17 implementation.

Liberica Native Image Kit JDK is a GraalVM based tool for creating performant native images.

Release schedule for all the products conforms to the LTS roadmap.
All products are available for a large number of platforms.

Linux

Alpaquita Linux is 100% Alpine compatible, secure, and optimized for Java.



be//soft

Some of Java features that made it great

Garbage collection

Class library

Specifications

- JLS
- JVMS
- TCK

Backward compatibility

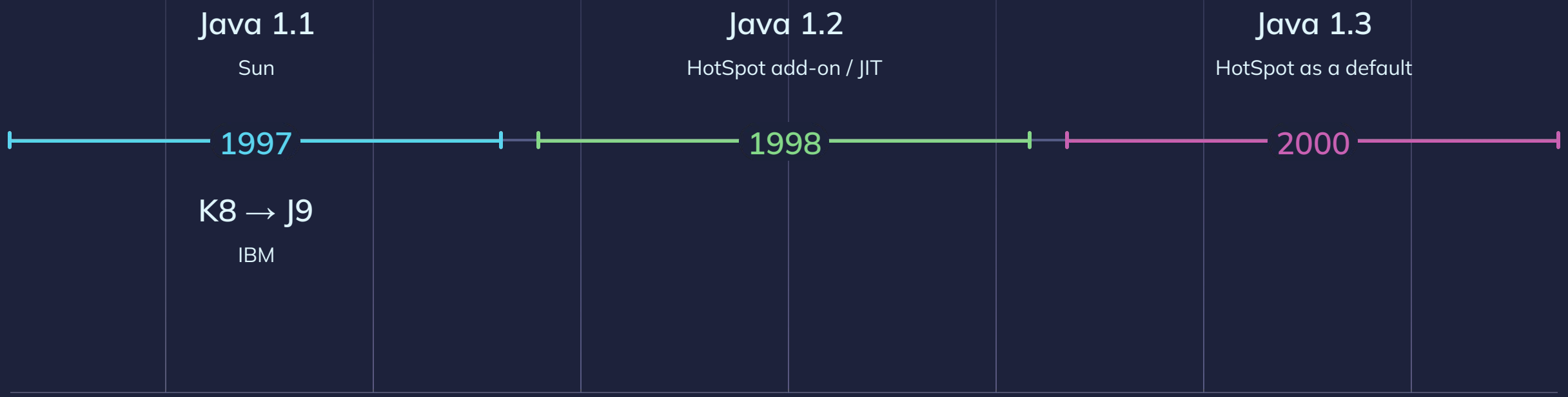
JVM

bellsoft

> The Java Virtual Machine is the cornerstone of the Java platform. It is the component of the technology responsible for its hardware- and operating system independence, the small size of its compiled code, and its ability to protect users from malicious programs.

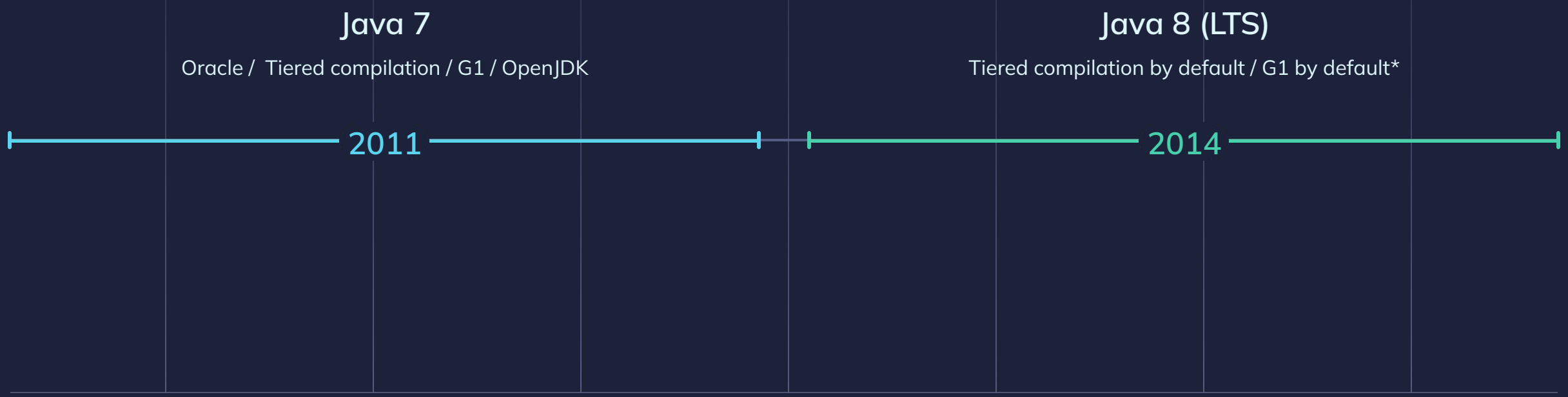
Dark ages

bellsoft



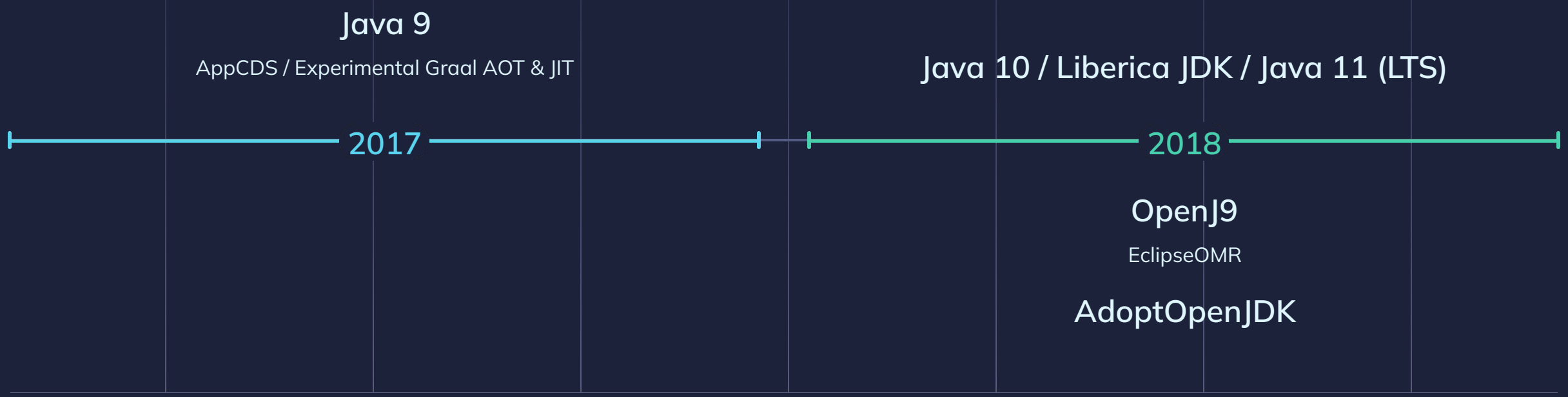
Gallant age

bellsoft



Steam age

bellsoft



Modern age

bellsoft

Modern age

bellsoft

bellsoft

DayTrader7 study claims

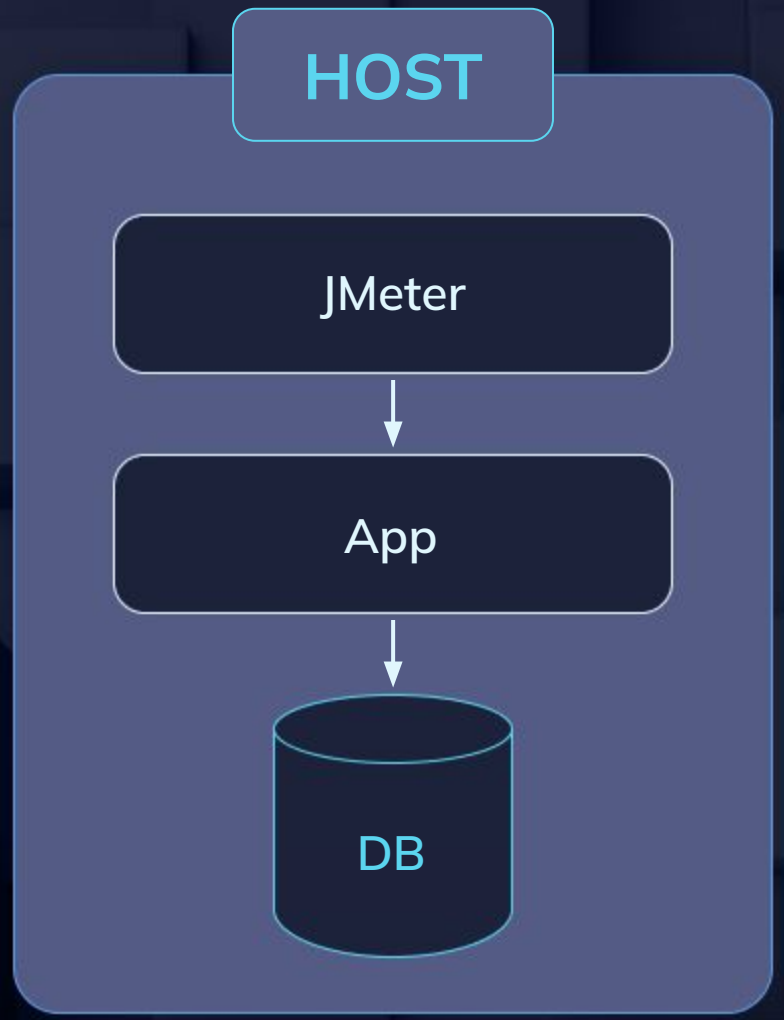
Setup

- Java 11. AdoptOpenJDK with HotSpot/OpenJ9
- Monolith app on OpenLIberty
- Desktop, 4 CPUs, 8 GB RAM (Xmx256m, Xmx1g)
- Linux
- 3 OpenJ9 configurations

  Single (default) HotSpot configuration

Summary

- **51 - 55%** faster startup time
- **45 - 57%** smaller startup footprint
- **33%** smaller footprint during application ramp-up
- Comparable throughput



bellsoft

Our setup

Desktop

- AMD Ryzen 5 3600X @ 3.79GHz, 1x6x1
- 8 GB RAM
- Windows 10 / Ubuntu 20.04.3

Server

- Intel Xeon Platinum 8268, 1x24x2
- 128 GB RAM
- CentOS 7

Java

- AdoptOpenJDK 11.0.11 with OpenJ9
- Liberica JDK 11.0.12 LTS
- Xmx256m, Xmx1g

Methodology

Main VM configuration types

- OpenJ9 (1) (heap)
- OpenJ9 (2) (heap, class cache)
- OpenJ9 (3) (heap, class cache, fast compilation)
- HotSpot (heap)

  HotSpot (heap, AppCDS, C1)

Startup

- RSS
- Application reports it’s started

No load

- RSS

Under load

- RSS
- Throughput (TPS): JMeter, simple and mixed
- Latency (ms) – WRK2, simple and mixed

Repeated experiments

Desktop startup and no load. Configurations

OpenJ9 (1) (heap 256 MB):

- “-Xmx256m“

OpenJ9 (2) (heap 256 MB, class cache):

- “-Xmx256m -Xshareclasses:name=mvn”

OpenJ9 (3) (heap 256 MB, class cache, fast compilation):

- “-Xmx256m -Xshareclasses:name=mvn -Xtune:virtualized -Xscmx200m”

HotSpot (heap 256 MB) aka default:

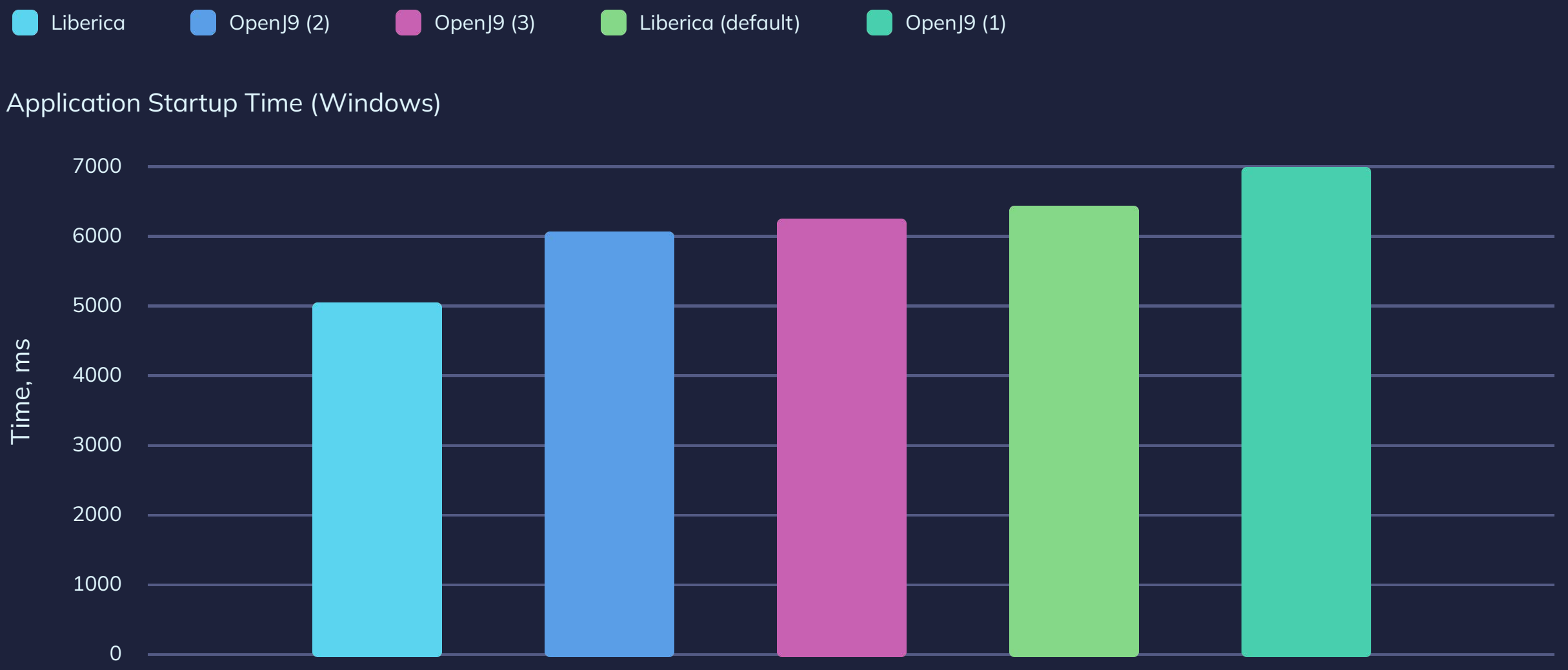
- “-Xmx256m”

HotSpot (heap 256 MB, AppCDS, C1):

- “-Xmx256m -XX:SharedArchiveFile=app-cds.jsa -XX:TieredStopAtLevel=1”

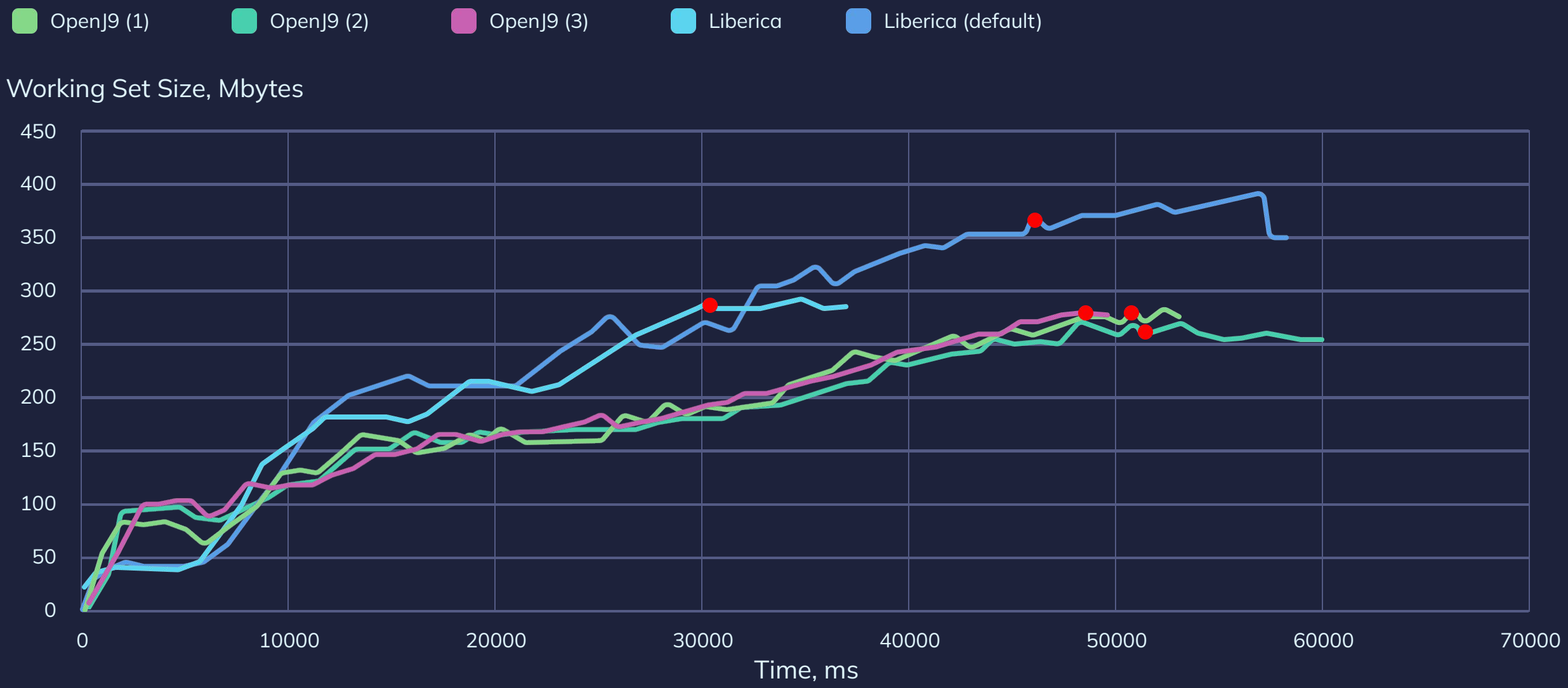
Startup time on desktop (lower is better)

bellsoft



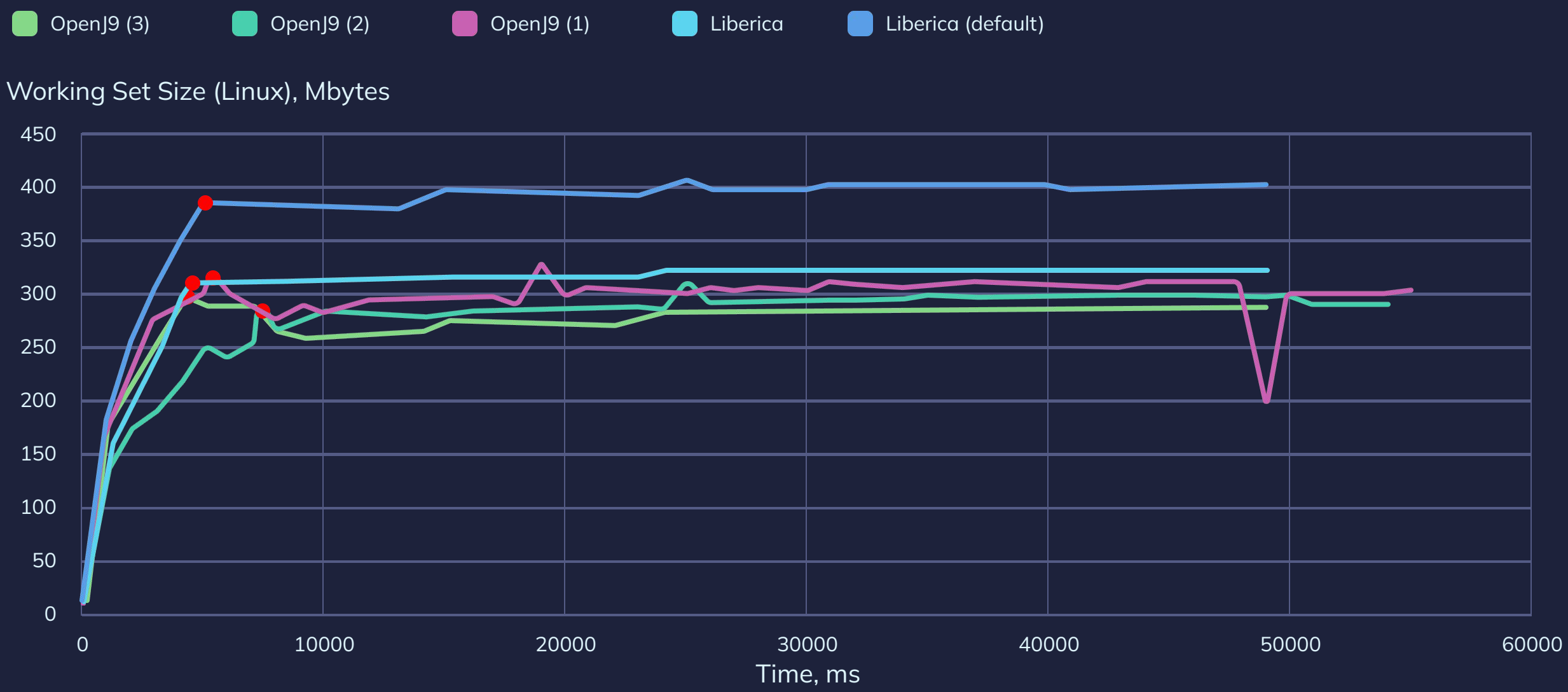
No load footprint on desktop (lower is better)

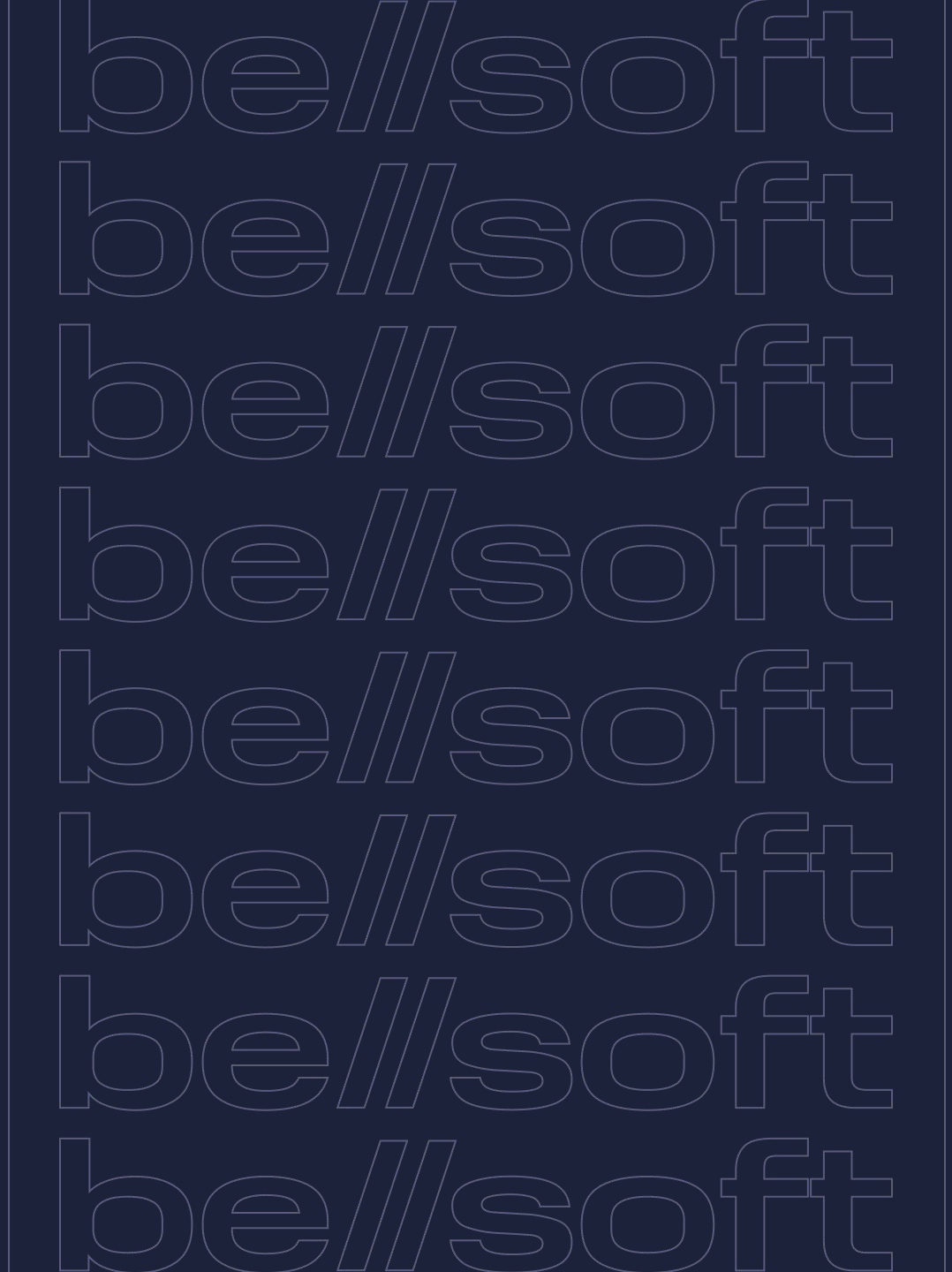
bellsoft



No load footprint on desktop (lower is better)

bellsoft





Server under load. Configurations

OpenJ9 (1) (heap 1G):

- “-Xmx1G“

OpenJ9 (2) (heap 1G, class cache):

- “-Xmx1G -Xshareclasses:name=mvn”

OpenJ9 (3) (heap 1G, class cache, fast compilation):

- “-Xmx1G-Xshareclasses:name=mvn -Xtune:virtualized -Xscmx200m”

HotSpot (max heap 1G, initial heap 80 MB):

- “-Xmx1G -Xms80m”

HotSpot (max heap 1G, initial heap 80 MB, AppCDS, C1):

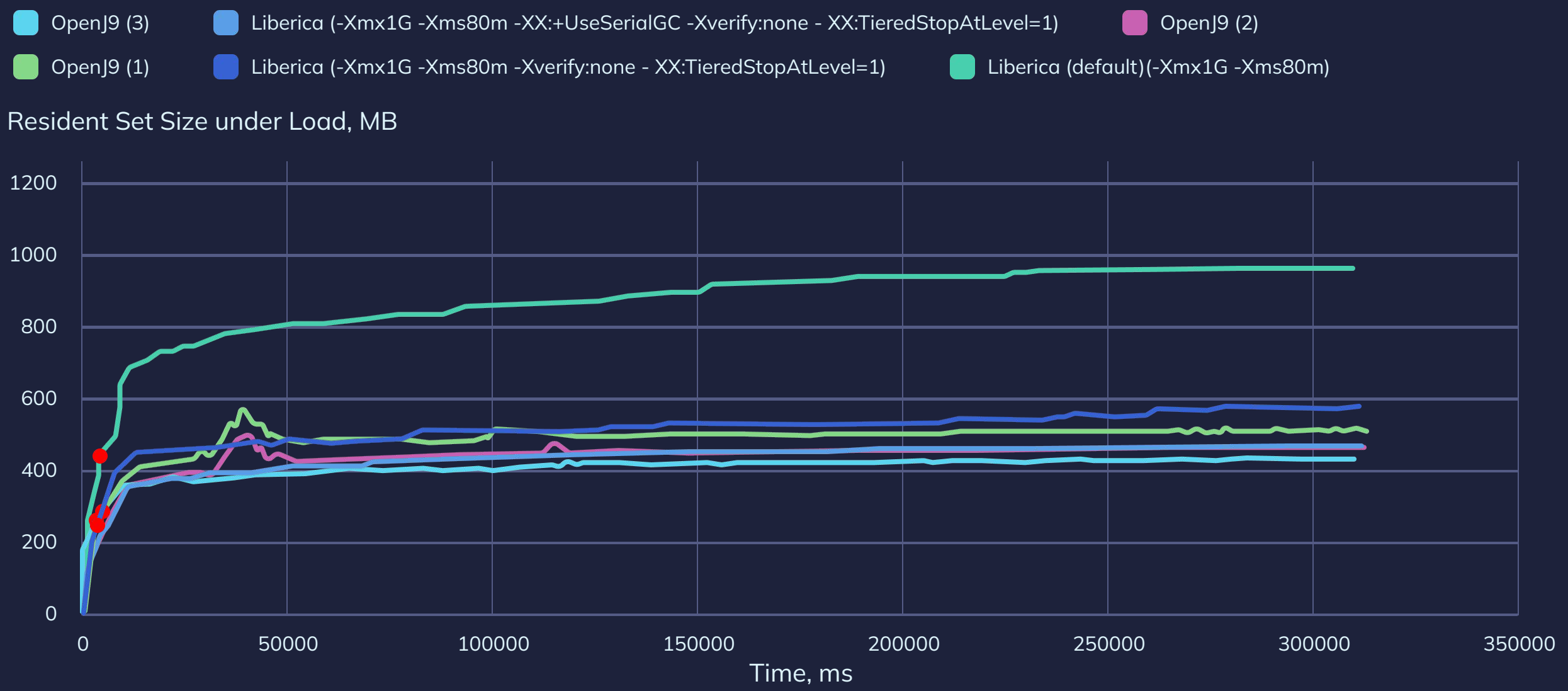
- “-Xmx1G -Xms80m-XX:SharedArchiveFile=app-cds.jsa -XX:TieredStopAtLevel=1”

HotSpot (max heap 1G, initial heap 80 MB, SerialGC):

- “-Xmx1G -Xms80m-XX:SharedArchiveFile=app-cds.jsa -XX:TieredStopAtLevel=1 -XX:+UseSerialGC“

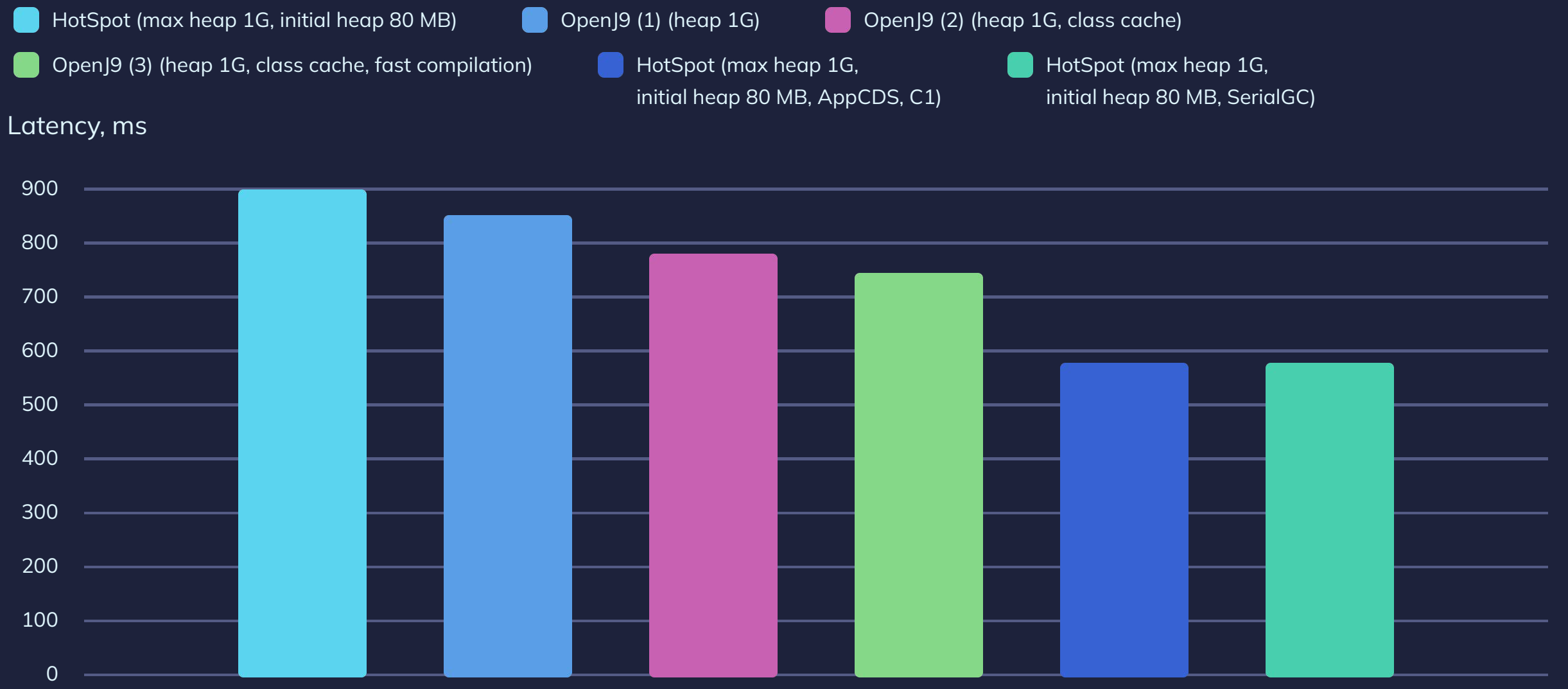
Footprint on server under load (lower is better)

bellsoft



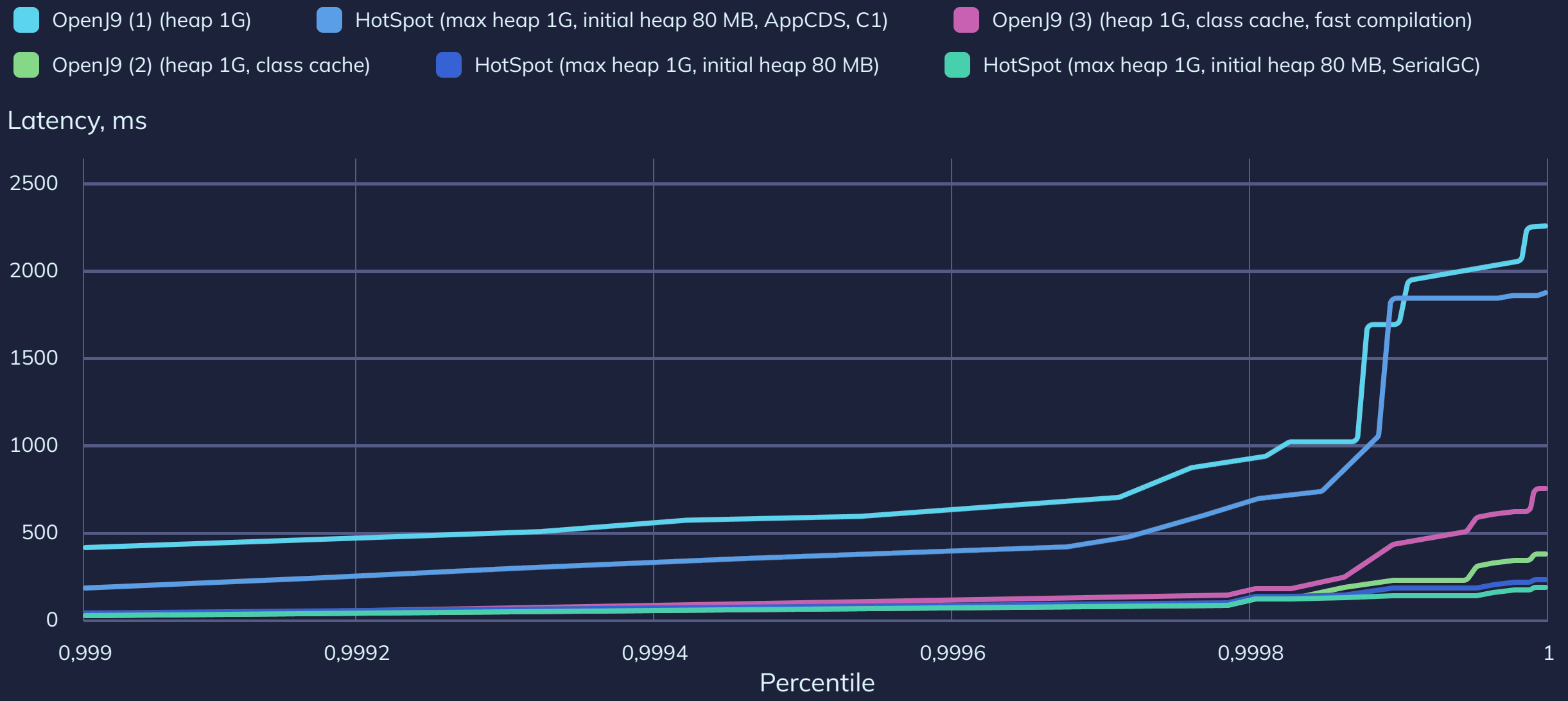
Throughput on server after warmup

bellsoft



Latency on server after warmup

bellsoft



Conclusions

bellsoft

Takeaways

Every benchmark lies differently

- Check your setup
- Check your app

HotSpot and OpenJ9 show great results

- Years of optimizations, many similarities
- Some configurations may differ significantly, any side may “win”

Tuning gives you a lot

- Defaults are usually good
- Study your app and JVM

Thank you for your attention!

https://bell-sw.com

@dchuyko

dmitry.chuyko@bell-sw.com