Open Liberty

OpenJ9

OpenJDK™

Simple Tweaks To Get the Most Out Of Your JVM

Jamie Lee Coleman
Software Developer/Advocate @ IBM
Twitter: @JamieLeeC

Java™

GraalVM™

What we will be looking at…

- What is a JVM and why is it important?
- How the cloud has changed the JVM
- HotSpot, OpenJ9 & GraalVM Overview
- Picking the right runtime to run on your JVM
- Tweaking your JVM
- OpenJ9 Class Cache + Demo
- OpenJ9 Idle Tuning
- OpenJ9 JIT Server
- CRIU
- Recap

Why is the JVM important?

Well firstly without it we would have no Java!

What does the JVM do

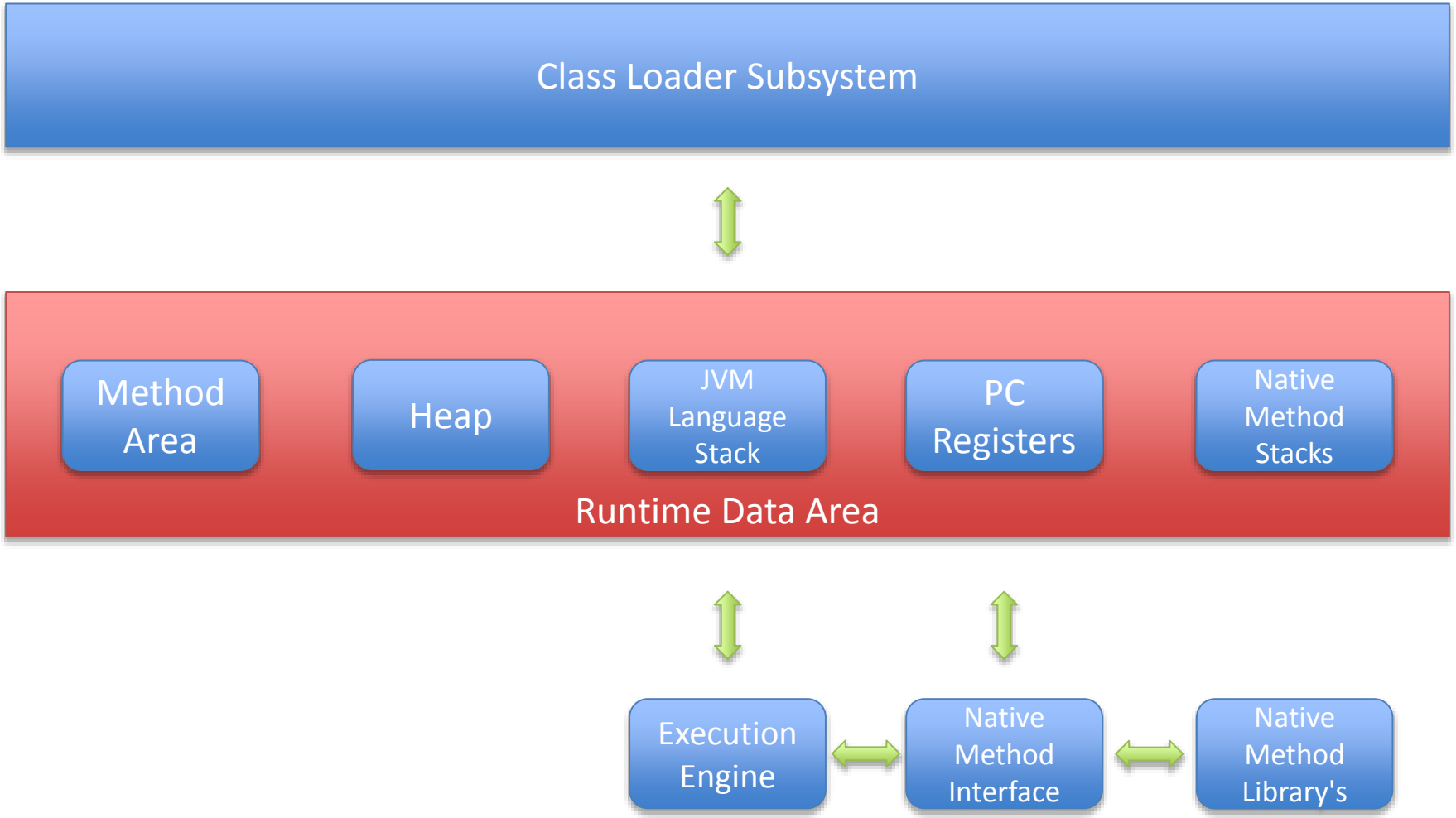
A Java Virtual Machine (JVM) is responsible for taking your application bite code and converting it to a machines language for execution.

Most other languages compile code for a specific system but in Java, the compiler converts the code for a Java Virtual Machine that can be used on any system hence the phrase “Write once, run anywhere”.

Why do people use JVM's?

- Cross Platform
- Large Library Ecosystem
- Great Garbage collector
- Powerful Monitoring tools
- Proven and Robust!

JVM Architecture



ClassLoader Subsystem

Loading

- Loads classes using 3 different class loaders named the **Bootstrap**, **Extension** and **Application ClassLoaders**

Linking

- **Verify** the generated bite code, **prepare** all static variables and assign default values and **resolves** all symbolic memory references

Initialization

- Assigns static variables with their original values and executes the **static** block

Runtime Data Area

Method Area:

- Stores **metadata**, **constant runtime pool** and **code for methods**.

Heap:

- Common memory shared between multiple threads that stores **Object, Instance Variables** and **Arrays**.

JVM Language Stacks:

- Created when a thread is created and stores **local variables**

PC Registers:

- Every thread has its own Register, and it stores the address of the **Java virtual machine instruction** which is currently executing.

Native Method Stack:

- Holds **Native method information**. For every thread that is created so is a native method stack.

Execution Engine

Execution Engine:

- Reads bytecode and executes it using the **Interpreter** and **JIT Compiler**

Native method interface (JNI):

- Interacts with the Native Method Libraries to provide the required libraries for the execution engine.

Native Method Library’s

- A collection of Native Libraries.

How the cloud has changed the JVM

$ $ $ $ $

Cloud computing
energy = money

Money changes everything

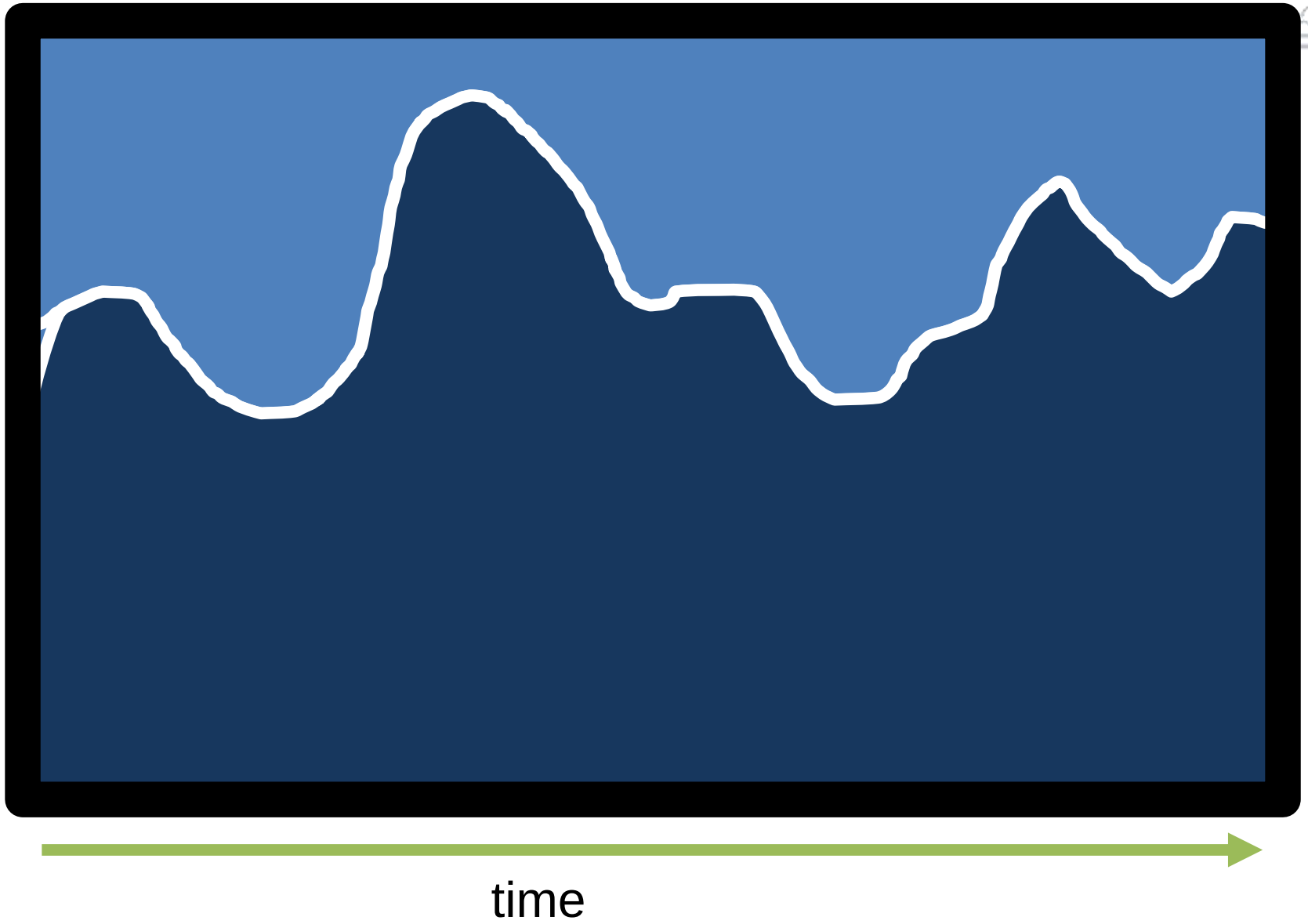
With a **measurable** and direct relationship between $£€¥₽ and CPU/RAM, disk etc the financial success or failure of a project is even easier to see

And that means…

Even more focus on value for money and as a result focus on energy.

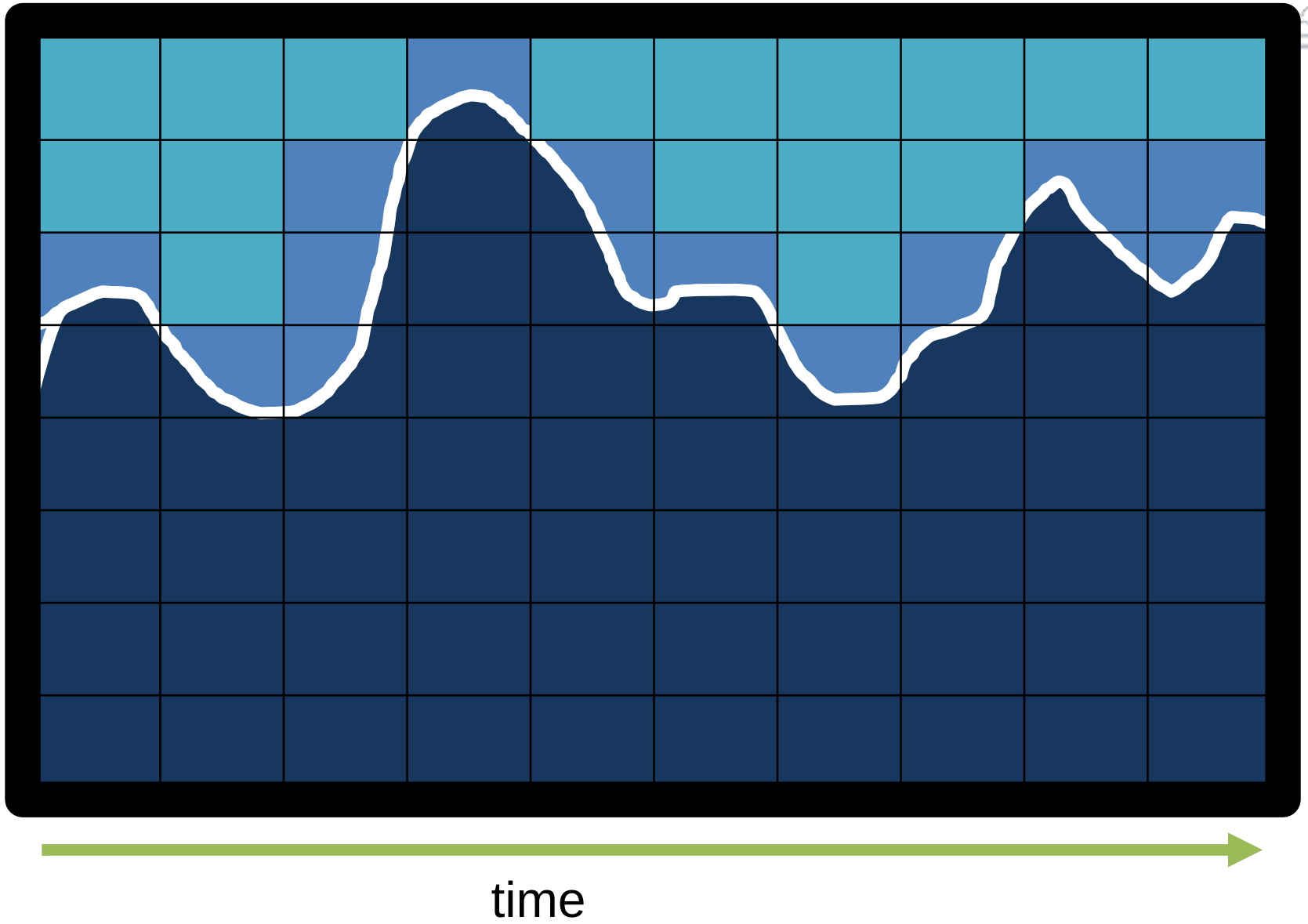
What the Cloud offers

Open Liberty



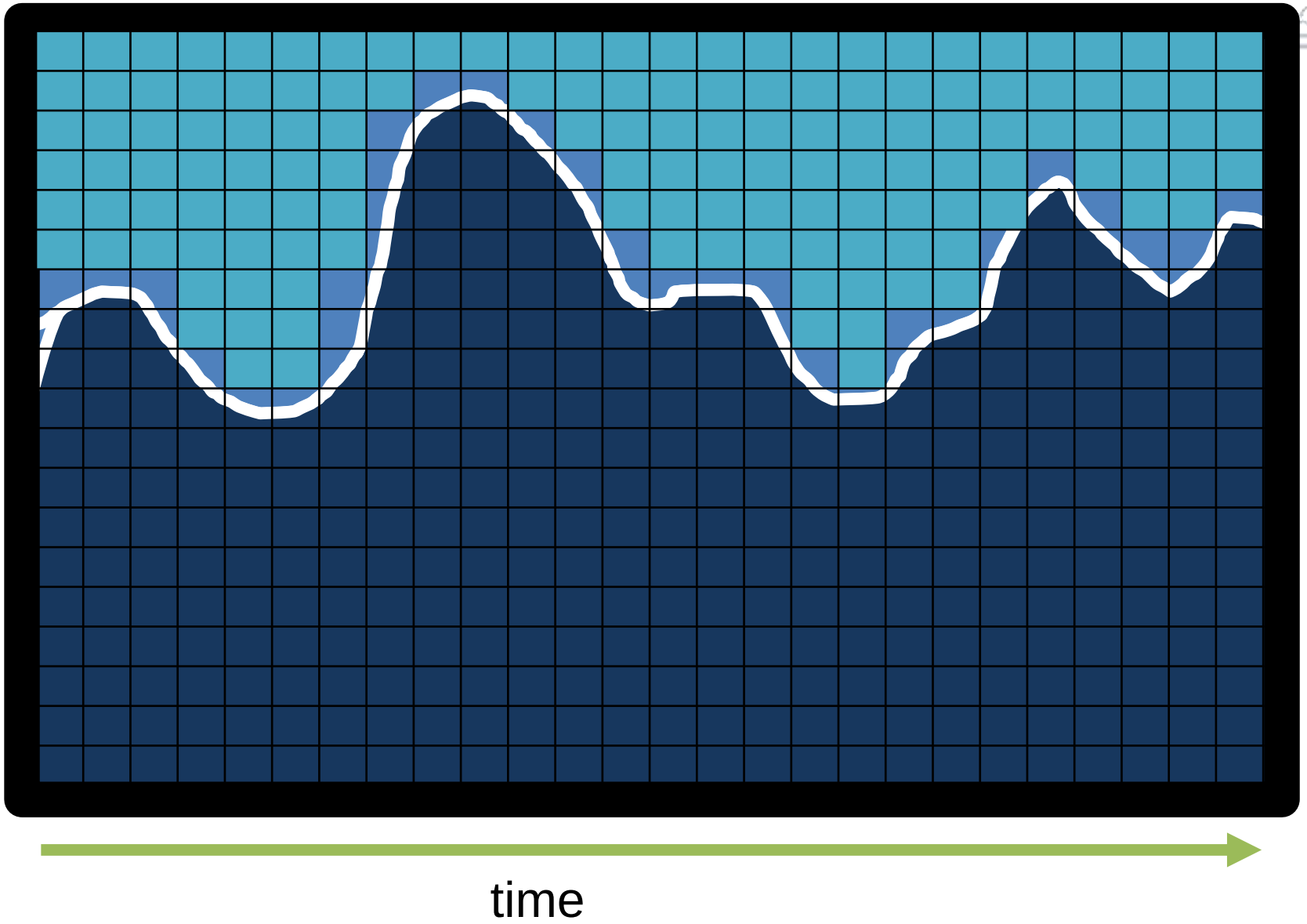
What the Cloud offers

Open Liberty



What the Cloud offers

Open Liberty



Microservices and the JVM

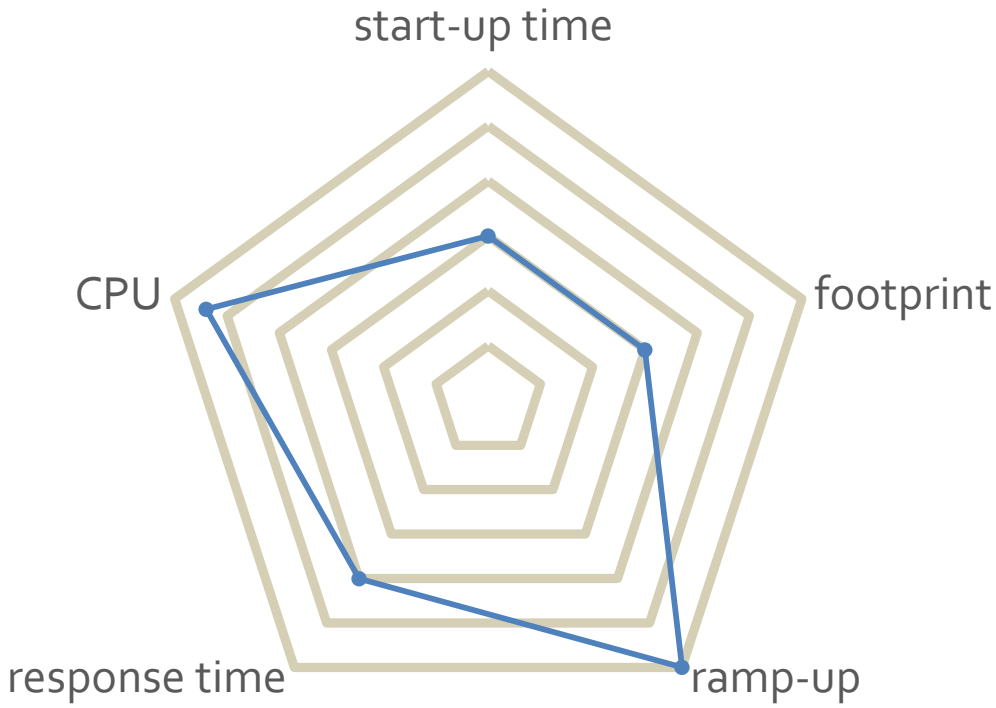
Open Liberty

Many metrics must be balanced for Java Performance

Open Liberty

- **Wide variety of use cases means many metrics must be balanced**
- Different goals → different design decisions
- **No single "right" answer**
- Must keep a balance → make sensible trade-offs

- **Key performance metrics tracked**
  - start-up time
  - ramp-up time
  - memory footprint
  - response time
  - CPU/Throughput



Optimizing for cloud requires a different balance across these performance metrics

Java ME Inside!

Java ME

JAVA ME REQUIREMENTS	
Small footprint - On disk and runtime. - Very limited RAM, usually more ROM	
Fast startup - Everybody wants their games to start quickly	
Quick / immediate rampup - Game shouldn’t play better the longer you play	

Java ME vs the JVM in the Cloud

JAVA ME REQUIREMENTS	JVM IN THE CLOUD REQUIREMENTS
Small footprint - On disk and runtime. - Very limited RAM, usually more ROM	Small footprint - Improves density for providers - Improves cost for applications
Fast startup - Everybody wants their games to start quickly	Fast startup - Faster scaling for increased demand
Quick / immediate rampup - Game shouldn't play better the longer you play	Quick / immediate rampup - GB/hr is key, if you run for less time you pay less money

Why should developers care about optimisation of our Applications?

It does not save that much money!

Open Liberty

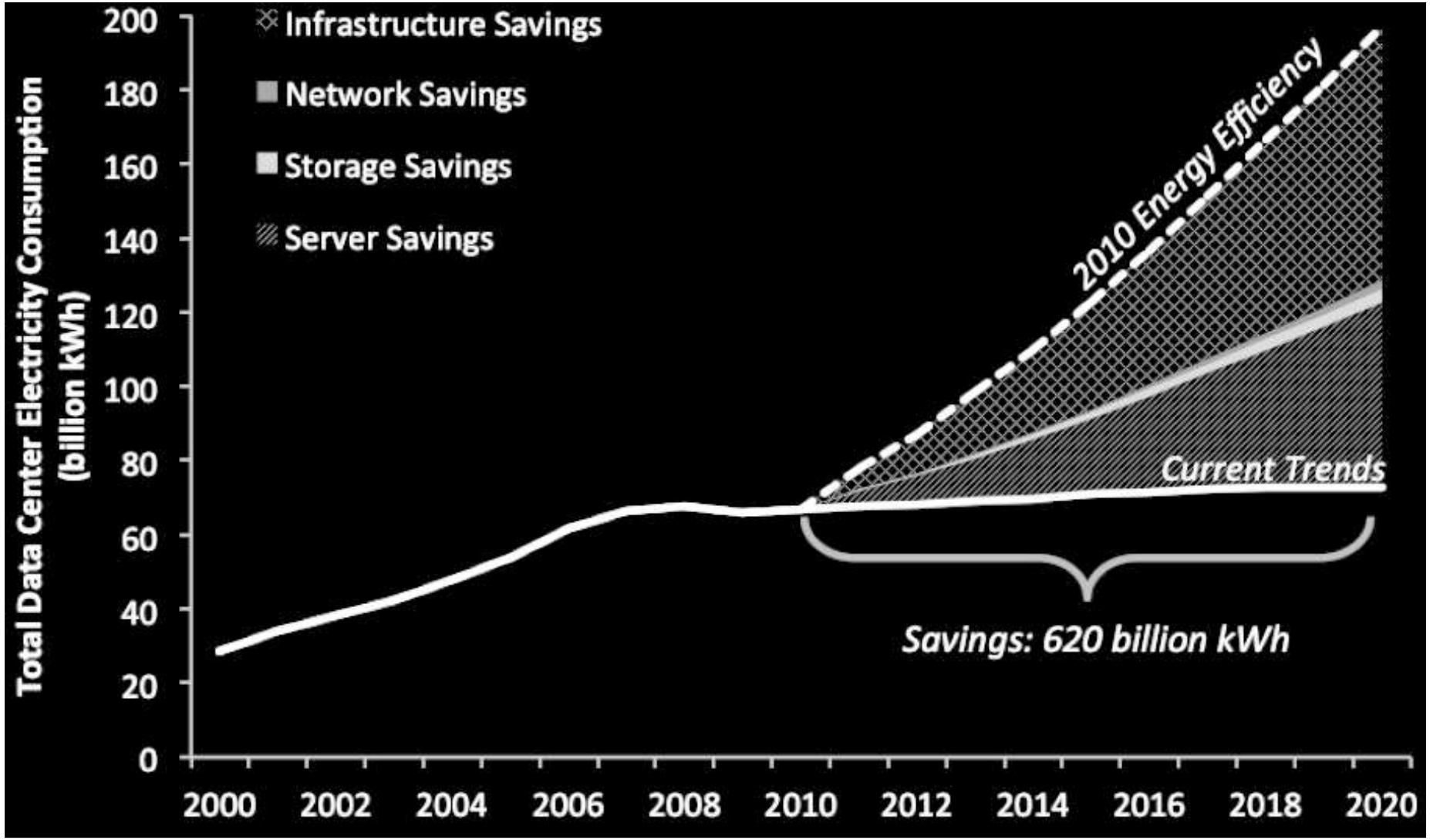
Why should developers care?

There is over 500,000 data centres worldwide

The area of land they consume is around the same as 6,000 football pitches

UK energy consumption x 1.5 == global data centre energy consumption (2019).

Luckily Hardware efficiency is helping



The Different JVMs Available

There is a JVM for everything

Azul

Codename One (Mobile)

Eclipse OpenJ9

GraalVM

HotSpot

JamVM (IOT Devices)

JikesRVM (Research)

leJOS (Robotics for Lego)

Maxine

Apache Harmony

JOP

Juice

Jupiter

Kaffe

Mika VM (Embedded Devices)

NanoVN (Asuro Robot)

SableVM

SquawkVM

SuperWaba

TakaTuka

TinVM

WonkaVM

Most popular JVM's

- Oracle JDK
- OpenJDK (via Adopt OpenJDK)
- OpenJDK (via Oracle)
- OpenJDK (via Amazon Coretto)
- Azul
- GraalVM
- Eclipse OpenJ9 (via Adopt OpenJDK)

HotSpot, OpenJ9 and GraalVM

Overview

Overview of HotSpot

Formally known as “Java HotSpot Performance Engine”

Original release: 27th April 1999

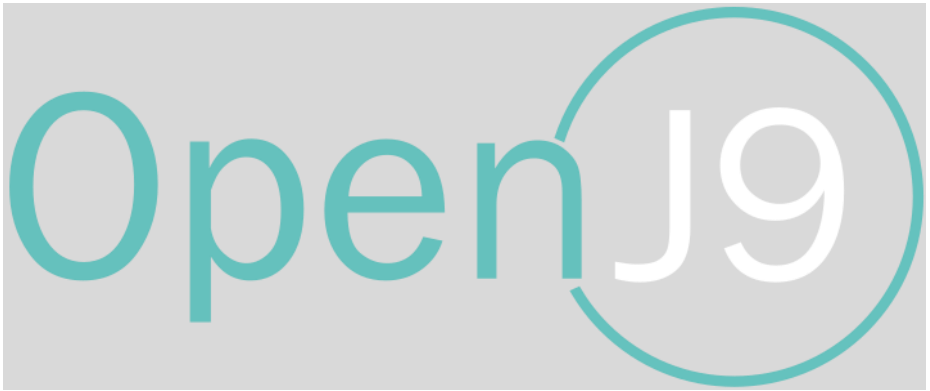
HotSpot became the default Sun (Now Oracal) JVM in Java 1.3

OpenJDK™

On 13 November 2006, the HotSpot JVM and the Java Development Kit (JDK) became Open Source paving the way for OpenJDK and many other JVM implementations.

“HotSpot continually analyzes the program's performance for *hot spots* which are executed often or repeatedly. These are then targeted for optimizing, leading to high-performance execution with a minimum of overhead for less performance-critical code” - Wikipedia

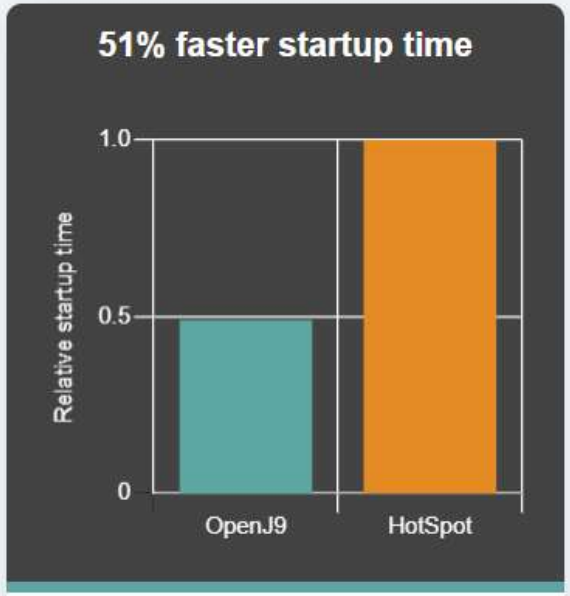
Overview of OpenJ9



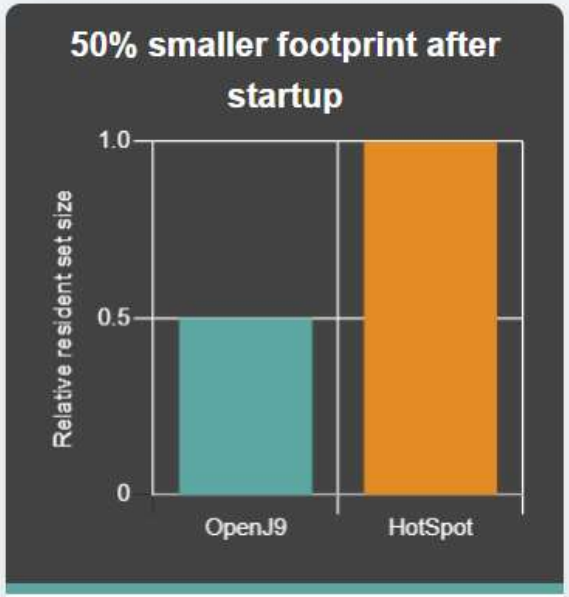
- Designed from the start to span all the operating systems needed by IBM products
- This JVM can go from small to large
- Can handle constrained environments or memory rich ones
- Is used by the largest enterprises on the planet

Overview of OpenJ9

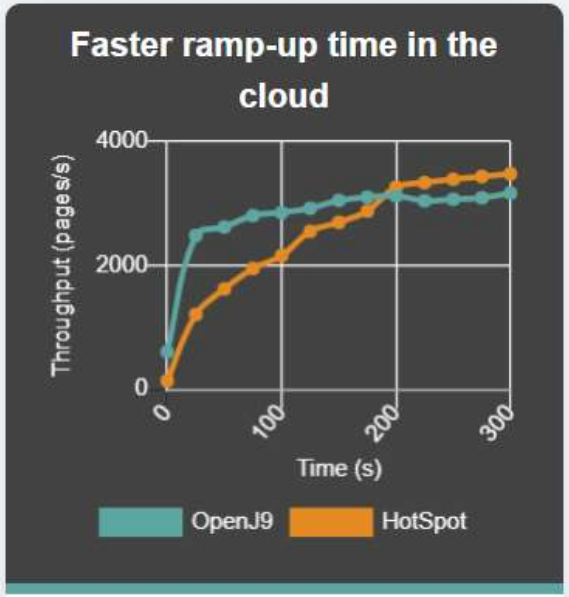
Open Liberty



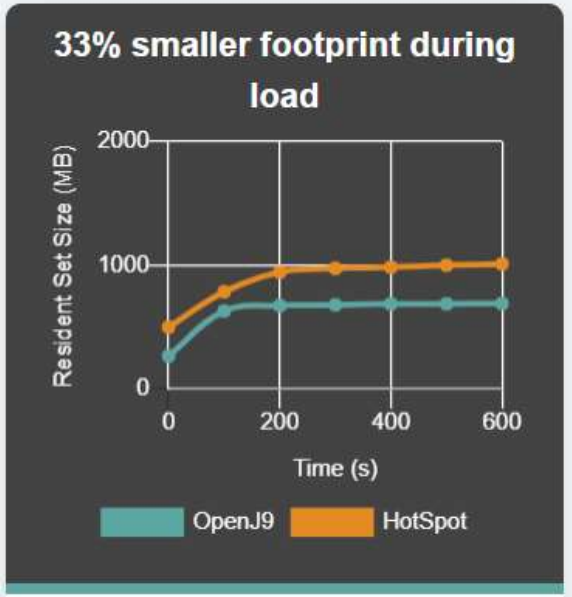
By using shared classes cache and AOT technology, OpenJ9 starts in roughly half the time it takes HotSpot.



After startup, the OpenJ9 footprint is half the size of HotSpot, which makes it ideal for cloud workloads.



OpenJ9 reaches peak throughput much faster than HotSpot making it especially suitable for running short-lived applications.



Consistent with the footprint results after startup, the OpenJ9 footprint remains much smaller than HotSpot when load is applied.

Overview of GraalVM

Original Release: May 2019 v19.0

Based on HotSpot OpenJDK with routes from the Maxine Virtual machine developed by Sun (Oracal) and the University of Manchester UK.

Why is it different to other JVM's?

- It has a new JIT compiler for Java, GraalVM Compiler
- It allows the ahead-of-time compilation of Java applications with the GraalVM Native Image
- Truffle Language Implementation framework and the GraalVM SDK to enable additional programming language runtimes
- LLVM and JavaScript Runtime

GraalVM™

GraalVM Project Goals:

- To improve the performance of Java virtual machine-based languages to match the performance of native languages.
- To reduce the startup time of JVM-based applications by compiling them ahead-of-time with GraalVM Native Image technology.
- To enable GraalVM integration into the Oracle Database, OpenJDK, Node.js, Android/iOS, and to support similar custom embeddings.
- To allow freeform mixing of code from any programming language in a single program, billed as "polyglot applications".
- To include an easily extended set of "polyglot programming tools".

GraalVM™

Picking the right Runtime

Open Source Runtimes from IBM & RedHat

Open Liberty™

An IBM Open Source Project

A lightweight open framework for building fast and efficient cloud-native Java microservices.

Build cloud-native apps and microservices while running only what you need. Open Liberty™ is the most flexible server runtime available to Java™ developers in this solar system.

Get Open Liberty

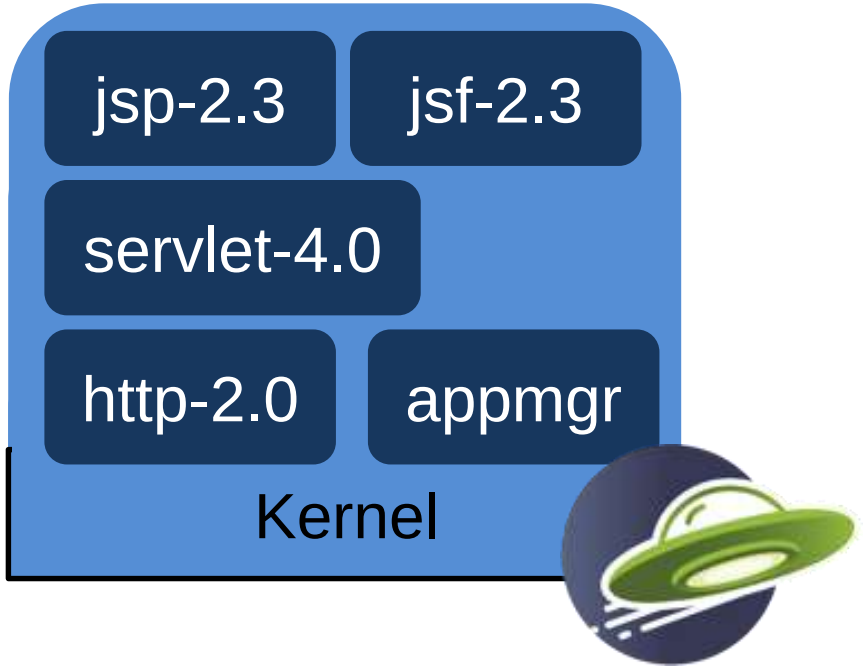
Liberty

- Developer friendly
  - Just enough application server
  - Fast inner loop with dev mode
  - Support for industry standard dev tools
  - Jakarta EE, Java EE, MicroProfile, Spring APIs
  - Zero Migration
- Cloud Ready
  - Container optimized
  - Designed for dev/ops
  - Small disk footprint
  - Efficient memory usage
  - Fast startup
  - High throughput
  - Self-Tuned Thread Pool

Just Enough Application Server

- You control which features are loaded into each server instance


```
<feature>jsf-2.3</feature>
```



API Support

- First shipped in WAS 8.5 in 2012
  - Servlet + JSP + JPA
- Web Profile 6 in 2014
- Java EE 7 in 2016 – first commercial product to certify
- Java EE 8 in 2018 – first to certify
- Jakarta EE 8 in 2019 – first to certify
- Eclipse MicroProfile – first to deliver 1.0-1.4, 2.0-2.1, 3.0

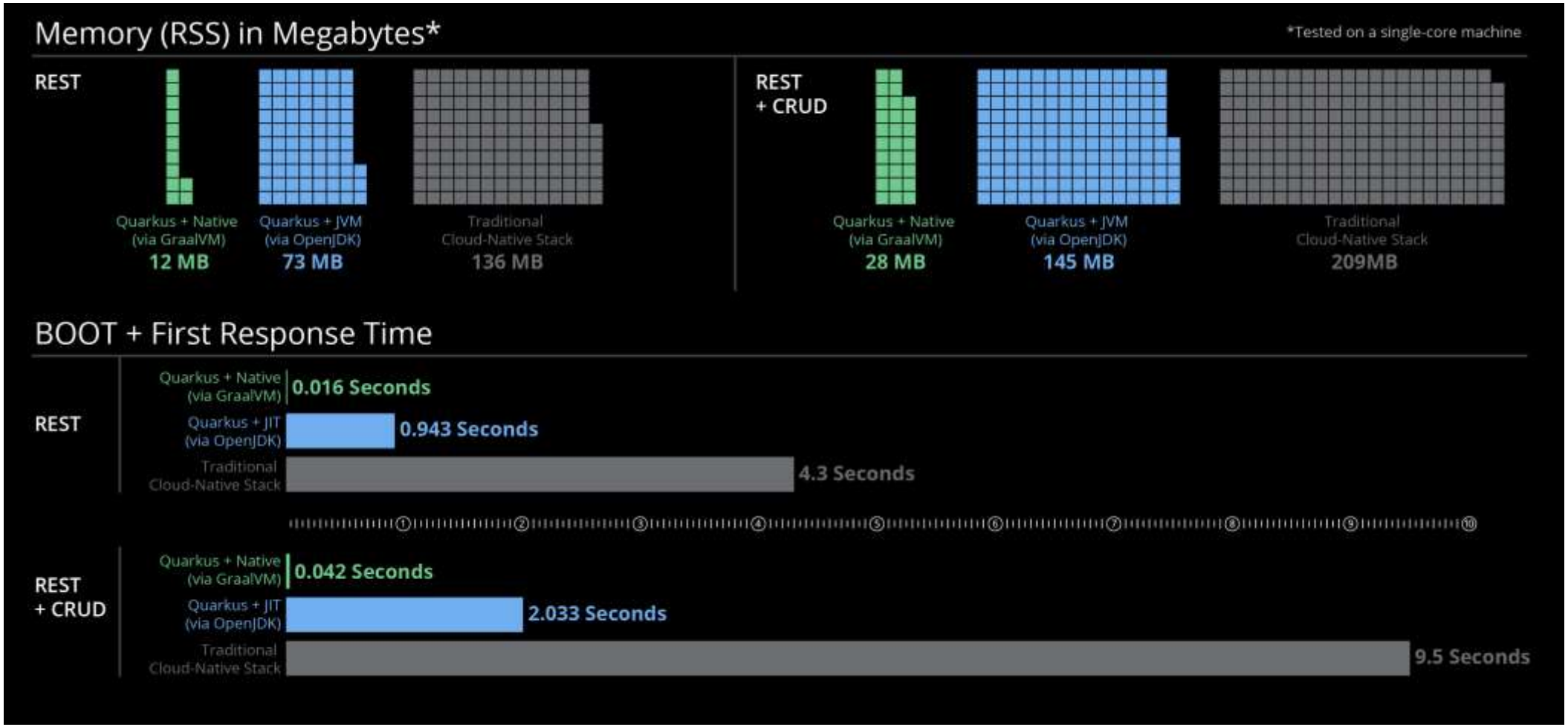

QUARKUS
Supersonic Subatomic Java
A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards.

Quarkus

- Unified Configuration
- Zero config, live reload
- Streamlined code for the 80% common usages
- Native Execution
- Support for many libraries such as MicroProfile

Quarkus

Open Liberty



Other Runtimes

Open Liberty

Tweaking the JVM

Making Your Java Code Perform

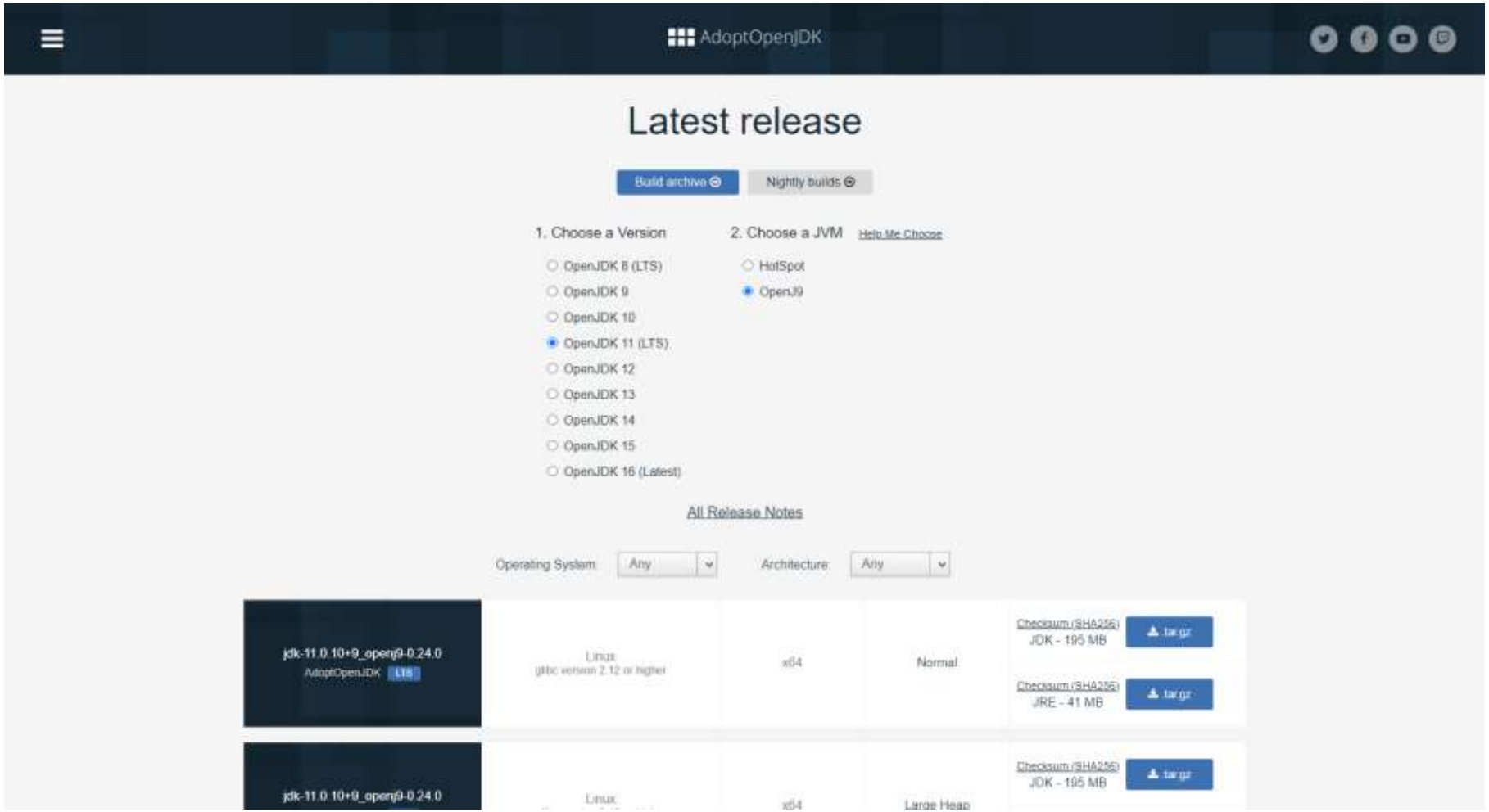
Making the most out of any JVM

1. Writing efficient code
2. Use the right JVM for your needs
3. Pick a runtime that is right for your application
4. Use Profiling to get the most out of your JVM
5. Tweak your JVM depending on your needs in development and production

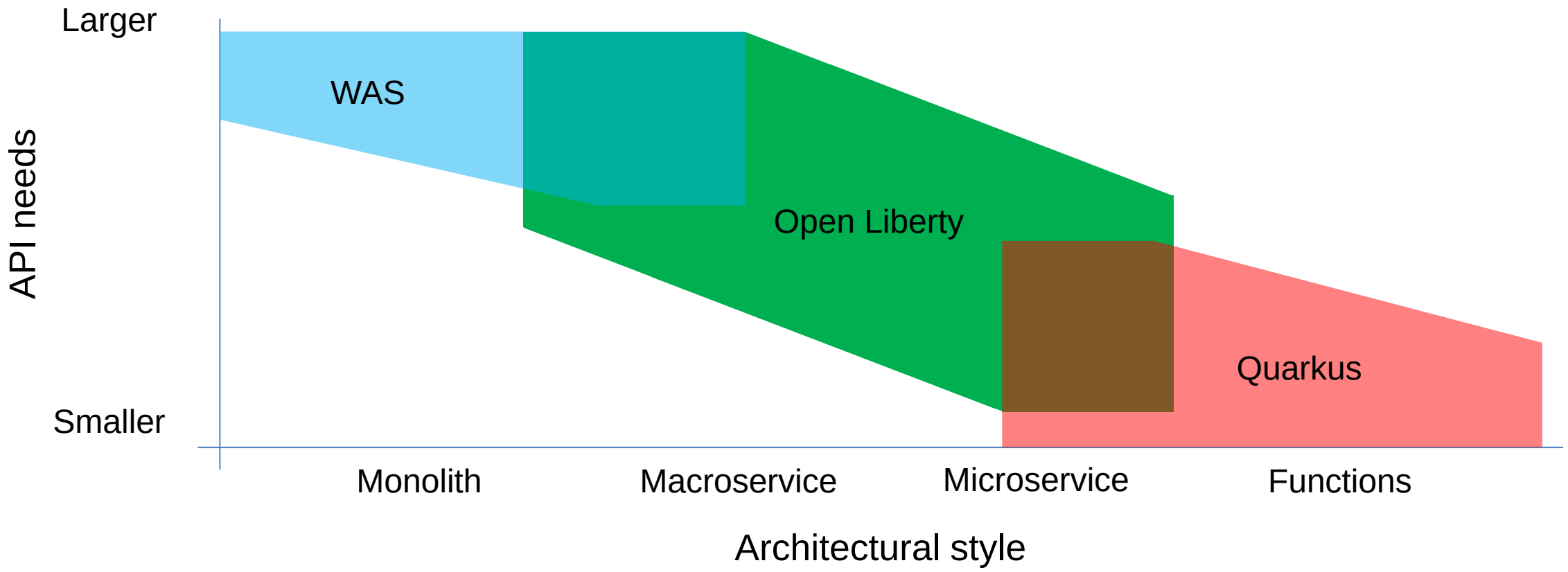
1. Writing efficient code

- Use primitives where possible
- Use a StingBuilder rather than an StringBuffer
- Avoid using an Iterator
- Avoid using BigInteger and BigDecimal

2. Use the right JVM for your needs



3. Pick a runtime that is right for your application



4. Profiling

Java Profilers are really useful tools that enable you to monitor Java bytecode constructs and operations inside the JVM. This includes things such as:

- Garbage Collections
- Object Creation
- Method Executions
- Iterative Executions
- Thread Executions

4. Profiling Cont

Most Popular Java Profilers:

- JProfiler
- YourKit
- Java VisualVM
- NetBeans Profiler
- Oracle Java Mission Control (Included in Oracle JDK)

5. Tweak your JVM

- Tune your Garbage Collector (GC)
- Use the AOT Compiler
- Enable Class Data Sharing (HotSpot & OpenJ9)
- Enable Idle Tuning
- Use a JIT Server
- Start with min Heap Size

OpenJ9 Class Data Sharing

For Super Fast Start-up

What is class sharing?

- A *shared classes cache* is an area of shared memory of a fixed size that persists beyond the lifetime of the JVM or a system reboot unless a non-persistent shared cache is used. Any number of shared caches can exist on a system, and all are subject to operating system settings and restrictions.

- A shared cache cannot grow in size. When it becomes full, JVMs can still load classes from it but can no longer store any data into it. You can create a large, shared classes cache up front, while setting a soft maximum limit on how much shared cache space can be used. You can increase this limit when you want to store more data into the shared cache without shutting down the JVMs that are connected to it.

How does class sharing work?

When a JVM loads a class, it first looks in the class loader cache to see if the class it needs is already present. If yes, it returns the class from the class loader cache. Otherwise, it loads the class from the filesystem and writes it into the cache as part of the defineClass() call. Therefore, a non-shared JVM has the following class loader lookup order:

- Classloader cache
- Parent
- Filesystem

In contrast, a JVM running with the class sharing feature uses the following order:

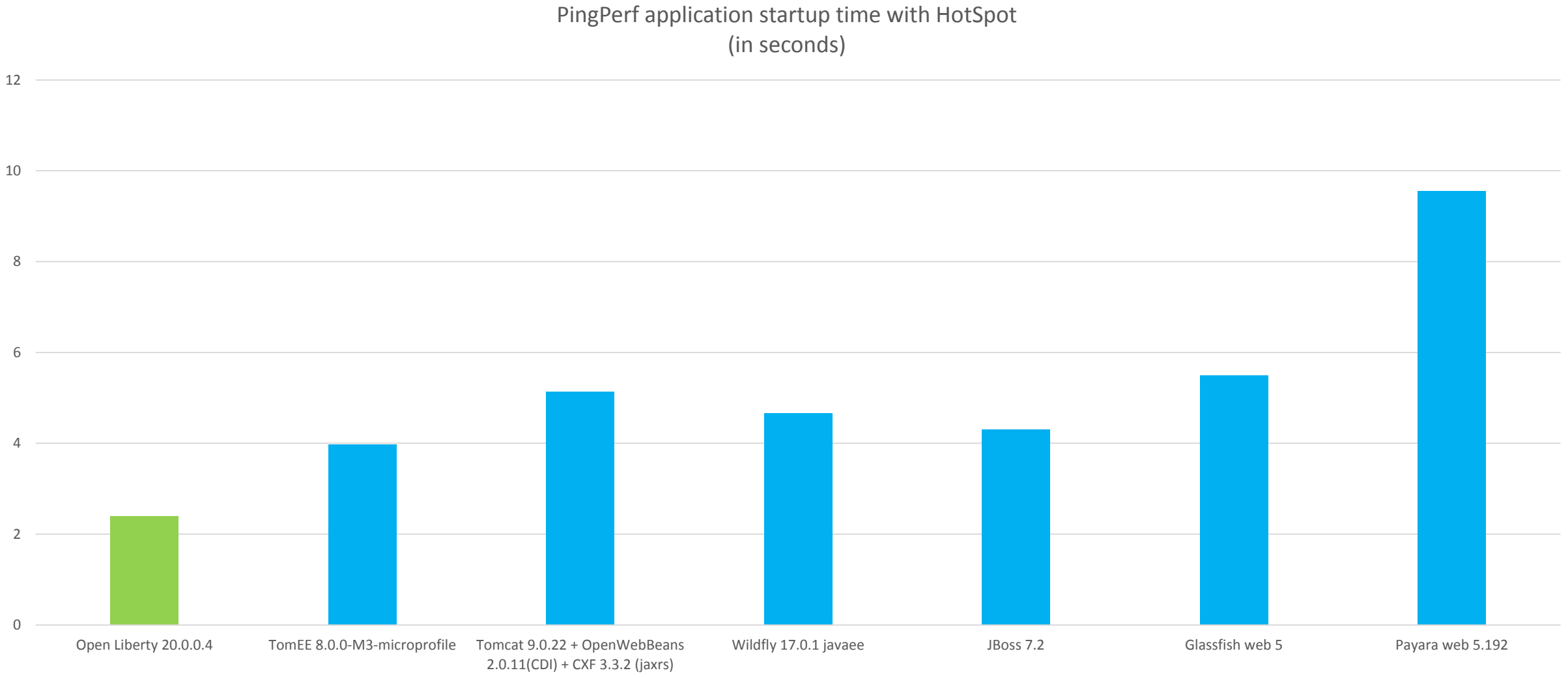
- Classloader cache
- Parent
- Shared classes cache
- Filesystem

How to Enabling class sharing

To enable class sharing, add -Xshareclasses[:name=<cachename>] to an existing Java command line.

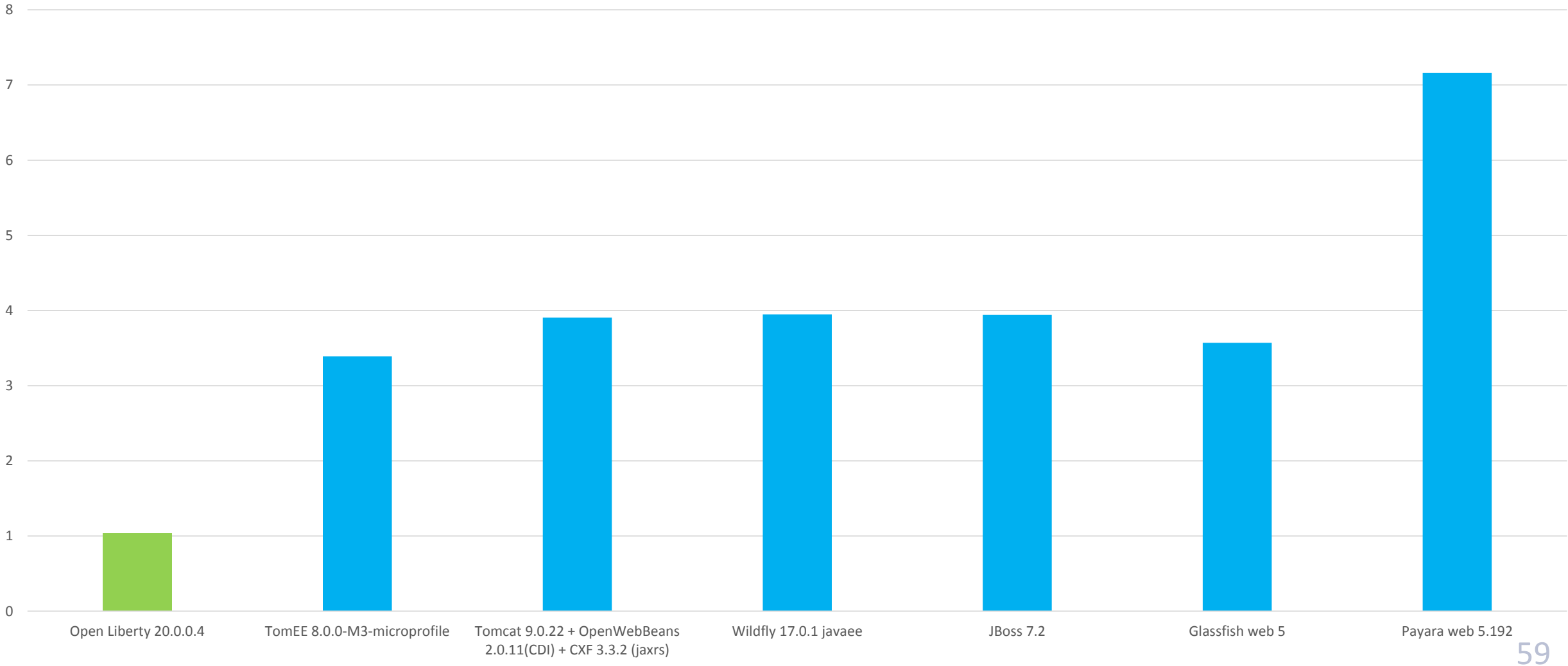
You can specify the shared cache size using the parameter -Xscmx<size>[k|m|g. This parameter only applies if a new shared cache is created.

Startup time comparison (using HotSpot JVM)

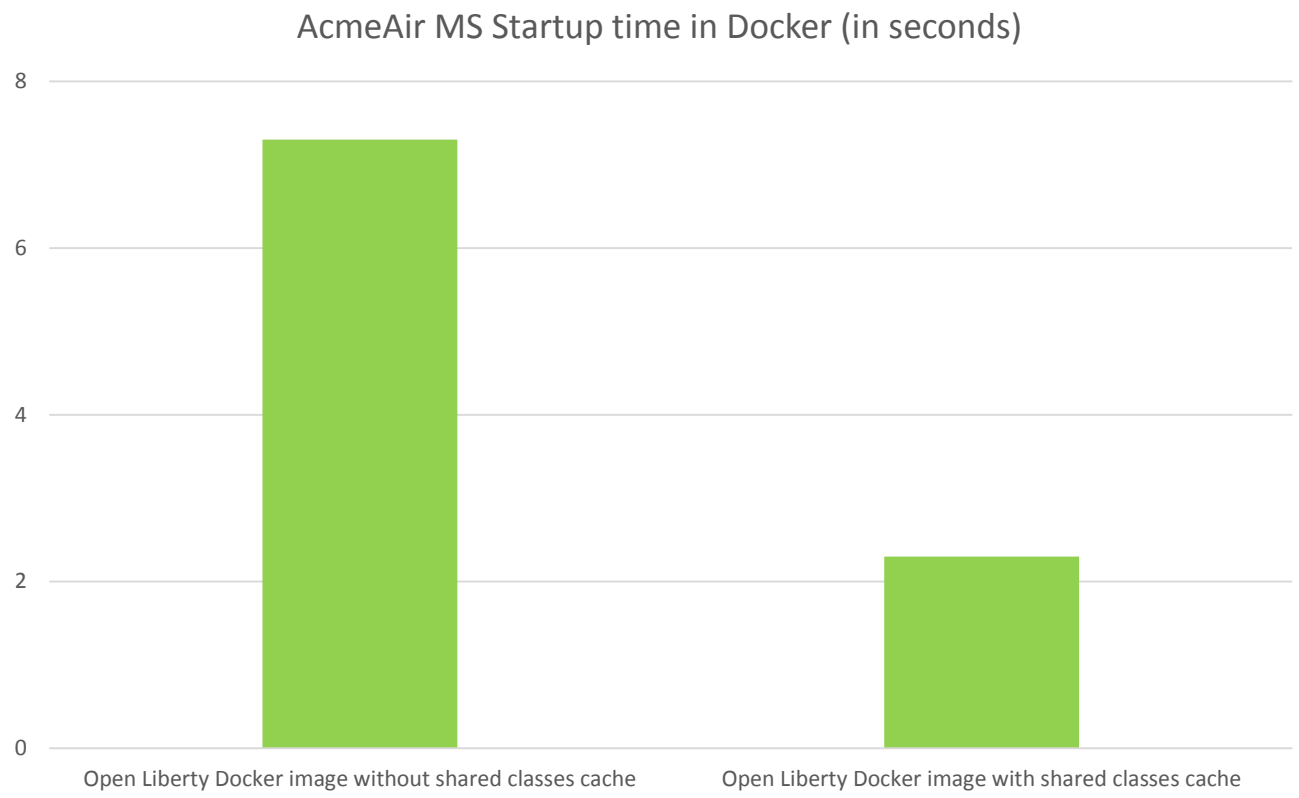


Startup time comparison (using OpenJ9 JVM)

PingPerf application startup time with OpenJ9 Shared Classes Cache (in seconds)

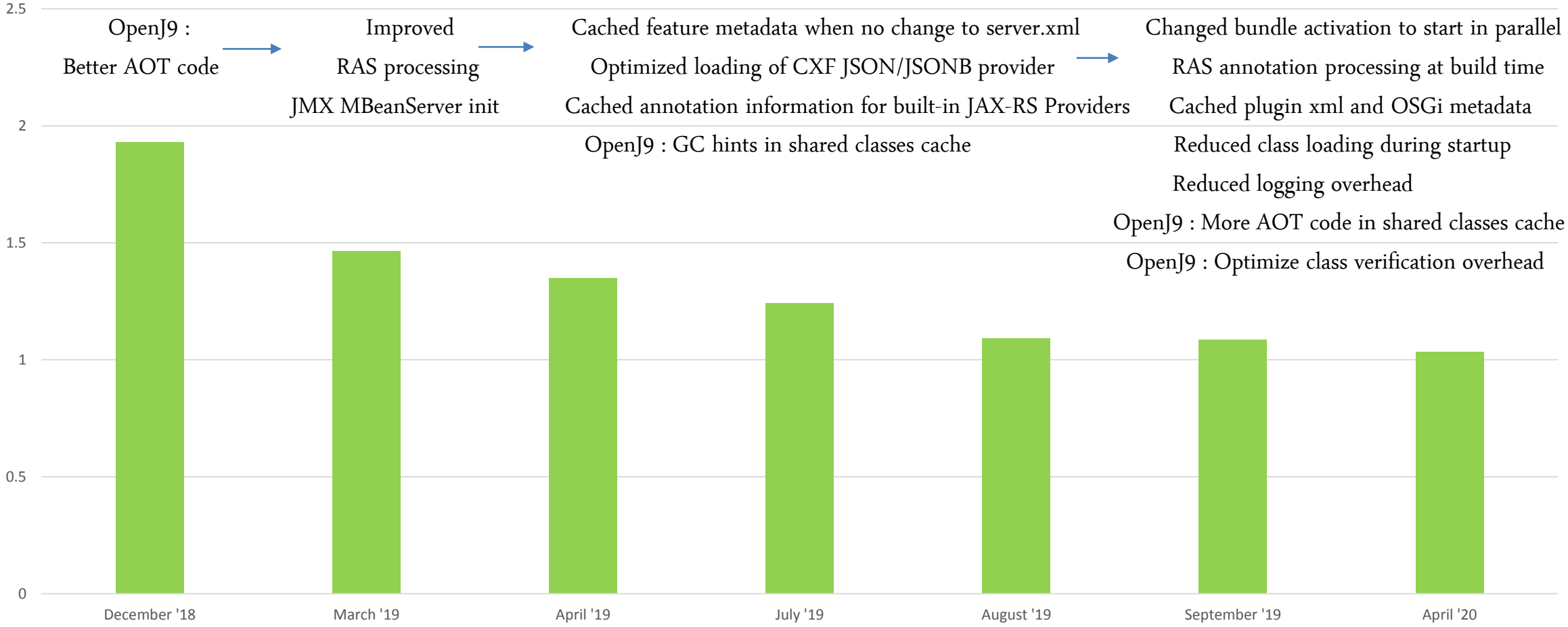


Open Liberty and OpenJ9 shared classes cache in Docker



Open Liberty with OpenJ9: the road to one second startup time

2018-2020 Progression of OpenLiberty+OpenJ9 startup time (seconds)



OpenJ9 Class Cache

Demo

Interactive Demo

If you want to run through the demo's at the same time please go to:

https://openliberty.io/guides/getting-started.html

OpenJ9 Idle Tuning

Open Liberty

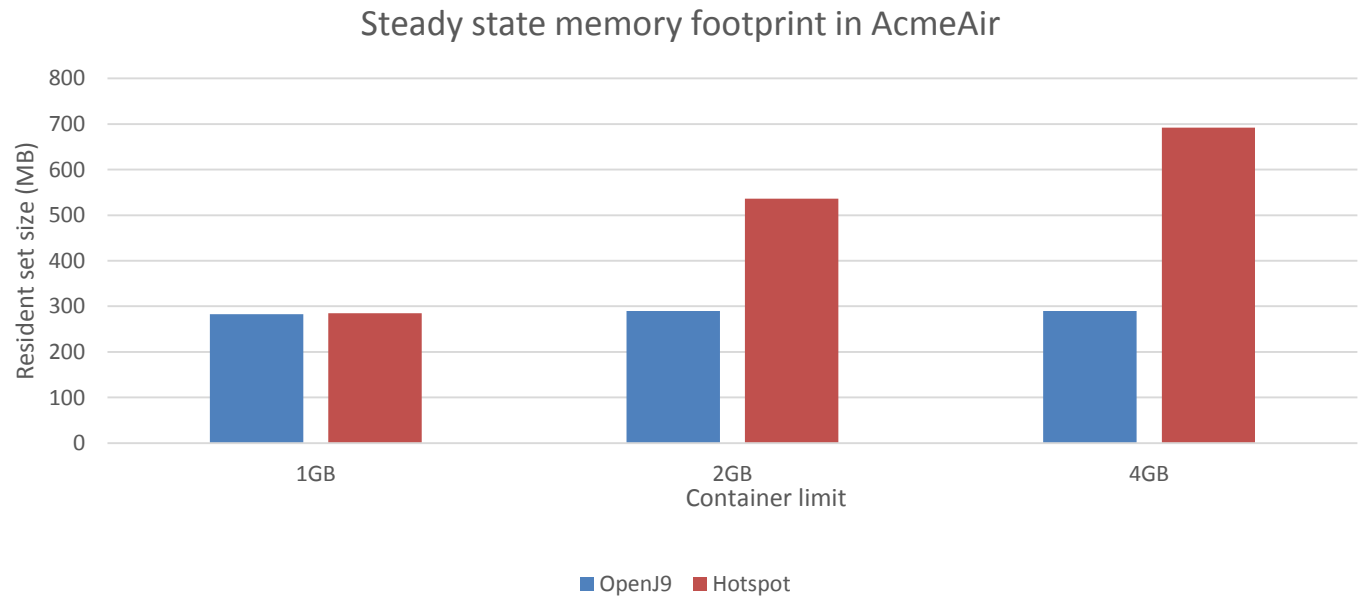
OpenJ9 JVM judiciously uses computing resources

Open Liberty

Computing resources ==

Cloud it's about sharing; do not be greedy in using all resources

- OpenJ9 is conservative with heap growth
- OpenJ9 frees memory used transiently during JIT compilation



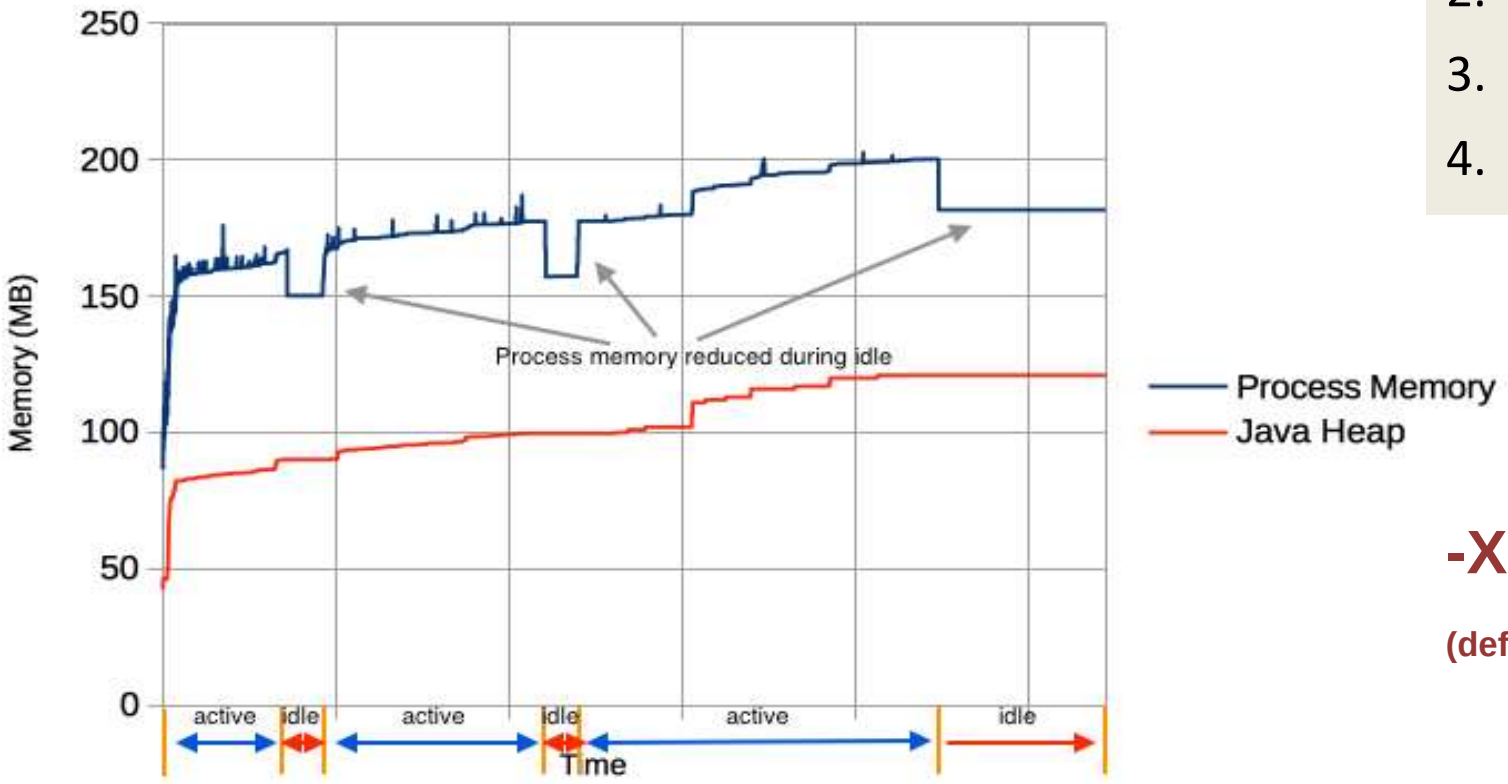
For large containers OpenJ9 uses less than half the memory

Free Resources when Applications are Idling

OpenJ9 frees resources when applications are idling

anthesisgroup.com: “Some 30 percent of VMs are zombies”

https://anthesisgroup.com/wp-content/uploads/2017/03/Comatsoe-Servers-Redux-2017.pdf



1. Detect idle state
2. Trigger GC
3. Compact Java heap
4. Send OS reclamation hint

-XX:+IdleTuningGcOnIdle

(default setting in container)

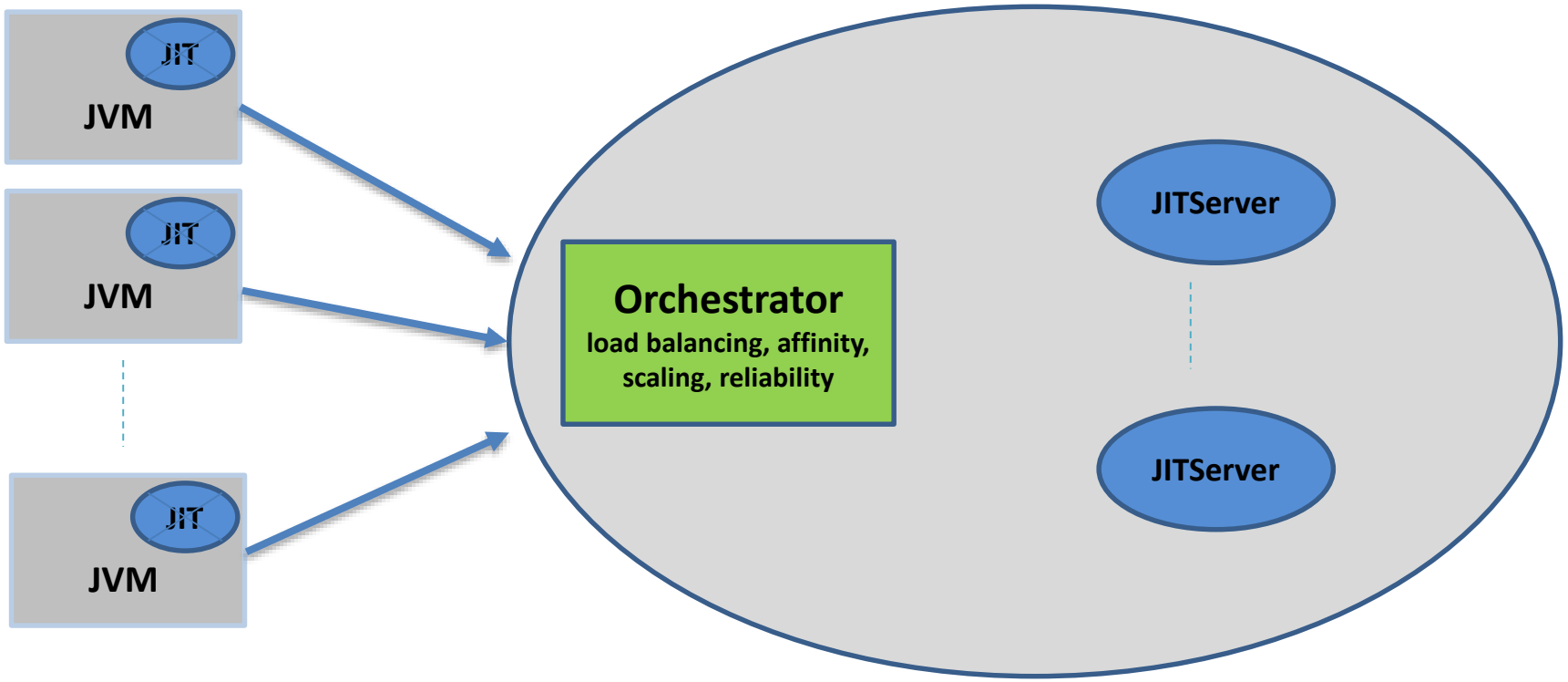
Benchmark: https://github.com/blueperf/acmeair

More details: https://developer.ibm.com/javasdk/2017/09/25/still-paying-unused-memory-java-app-idle

OpenJ9 JIT Server

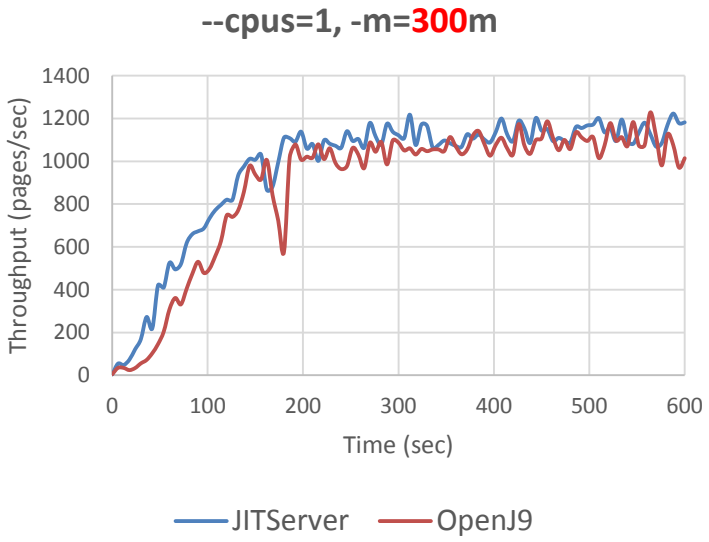
Open Liberty

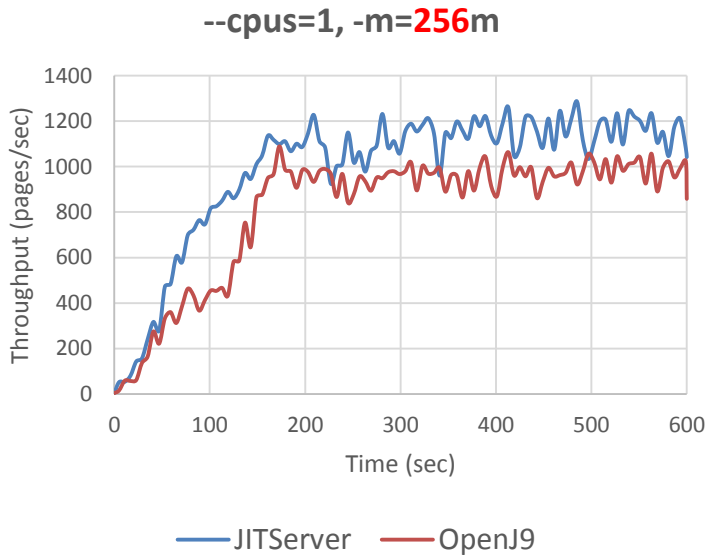
JIT server

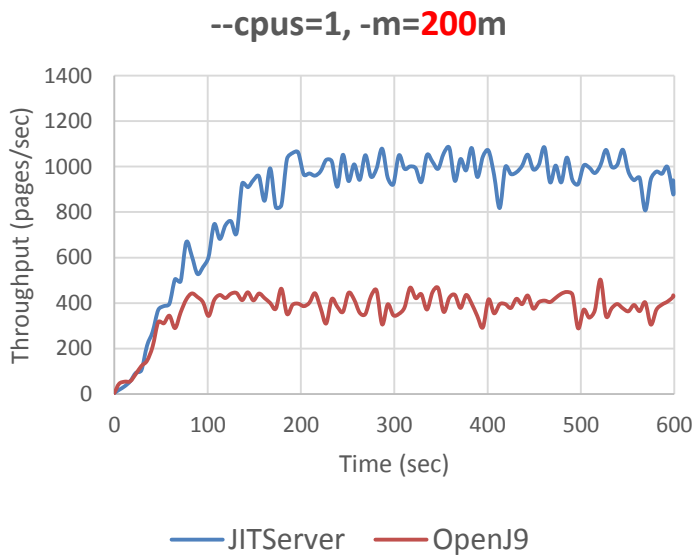

JIT
JVM
JIT
JVM
JIT
JVM
Orchestrator
load balancing, affinity, scaling, reliability
JITServer
JITServer

JITServer Performance

JIT server performance : Open Liberty Daytrader7 throughput







Smaller memory limit

Throughput benefits grow in constrained environments

How to enable a JIT Server

Launch OpenJ9 in client mode so that the VM sends requests to the JITServer using the following command:

“-XX:+IseJITServer”

Then run the following command to start a JITServer process that listens for incoming compilation requests:

“jitserver”

Future tech to the rescue?

Faster start-up for Java applications with CRIU snapshots

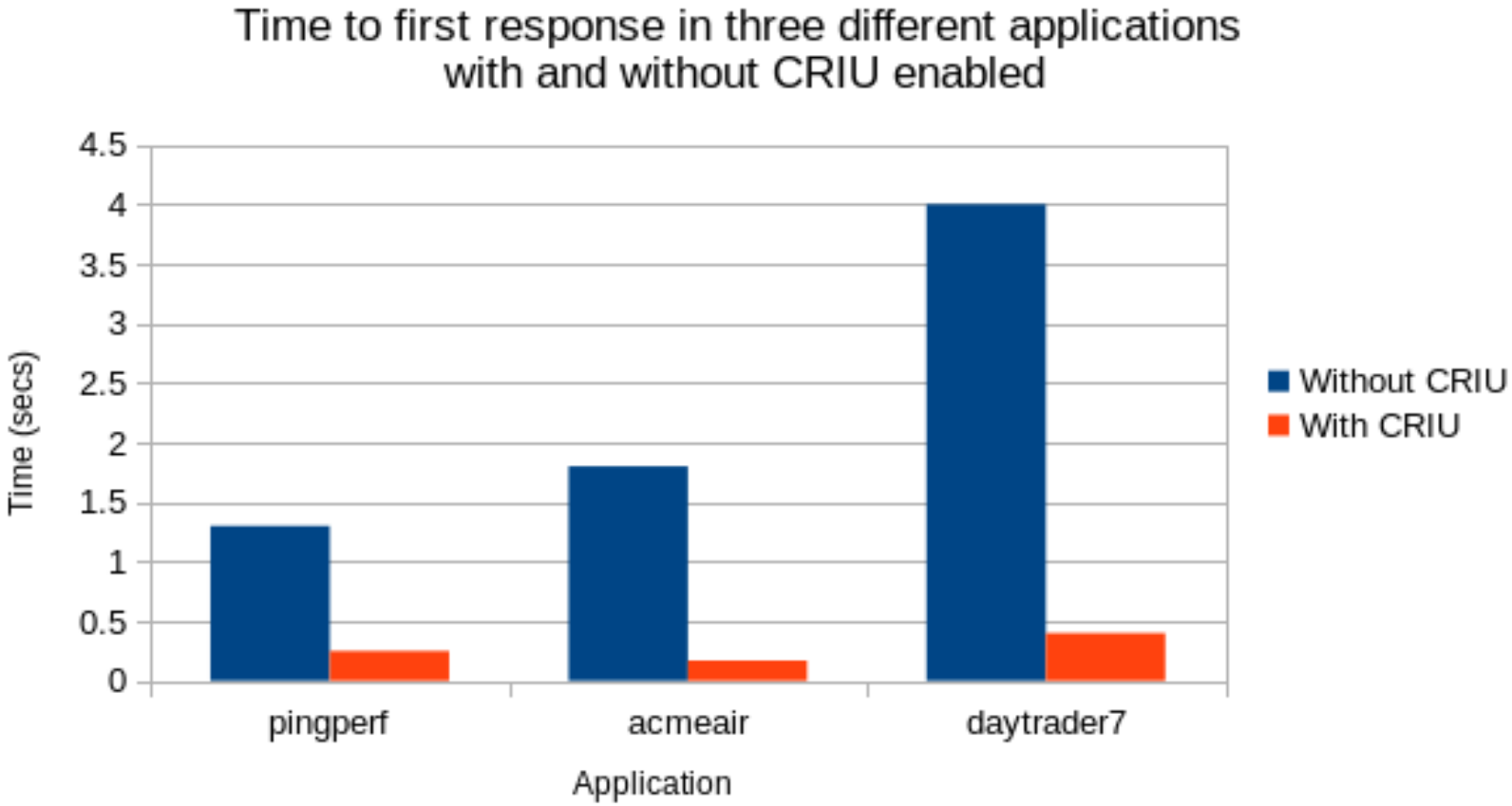
CRIU

Checkpoint/Restore in Userspace (CRIU) is a potential solution for reducing start-up time in Java applications.

CRIU is a feature on the Linux operating system that enables a snapshot (checkpoint) to be taken of a running application

The application instance can then be restarted from the point at which the snapshot was taken.

CRIU



5-10 times improvement in start-up time, with the greater benefits of CRIU coming from more complex, slower to start, applications.

Challenges with CRIU

- Lack of encryption on CRIU snapshots
- No Address Space Layout Randomization (ASLR)
- Predictable random number generators
- Running as non-root user
- No API to specify when the snapshot should be taken

Recap

What if anything have you learned today?

How do you get the most out of your JVM?

Write efficient code

Use the right JVM for your needs

Pick a runtime that is right for your application

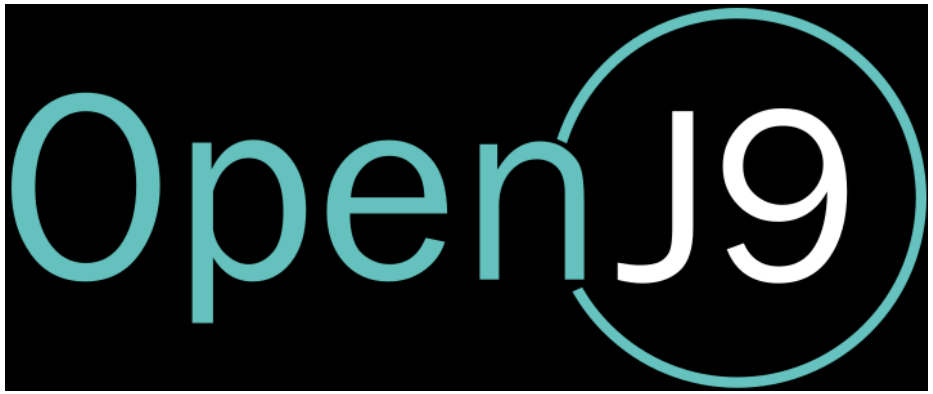
Use Profiling

&

Tweaking your JVM

=

Less Money & Energy!



Links and Materials

JVM's

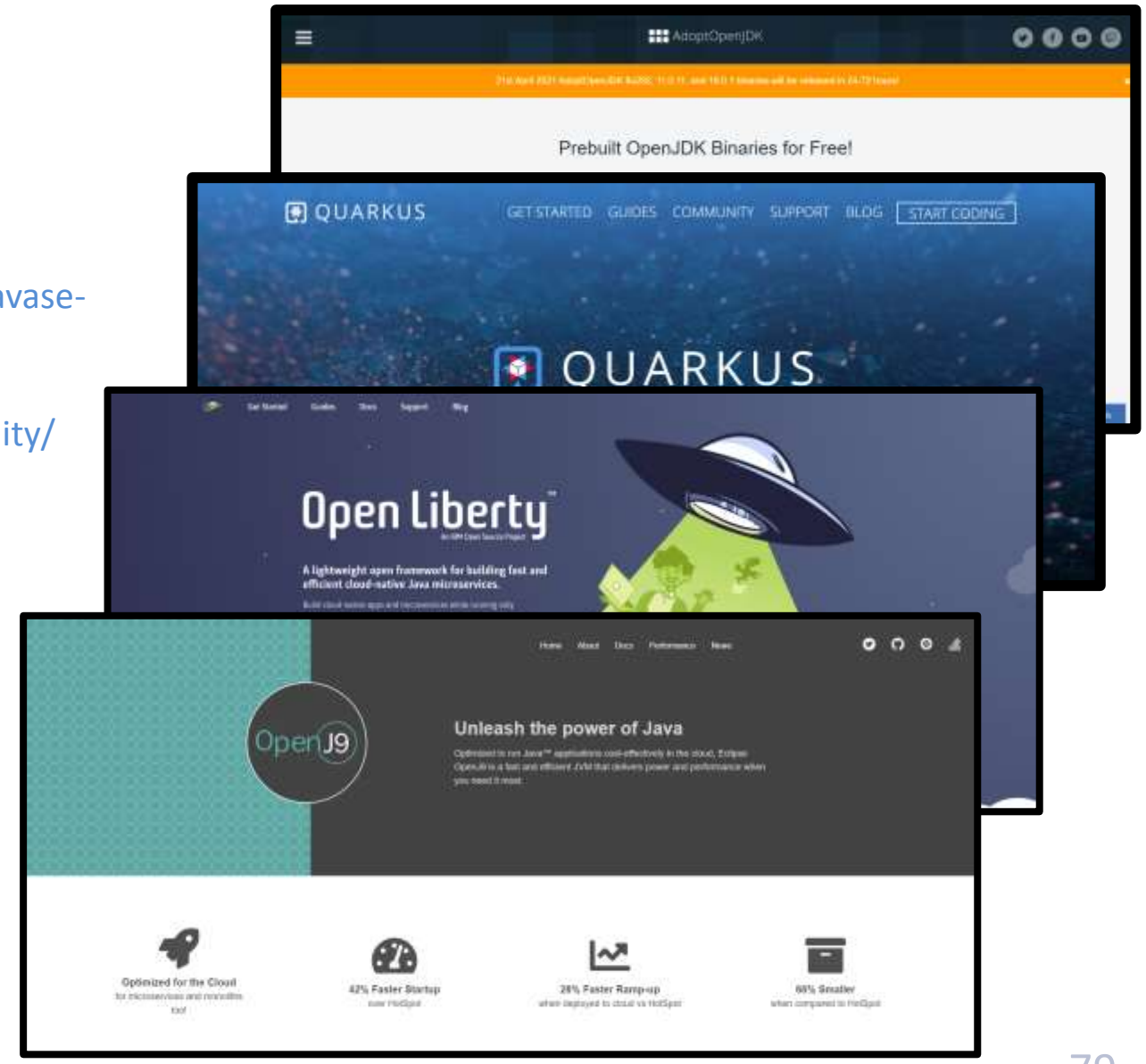
- AdoptOpenJDK: https://adoptopenjdk.net/
- OpenJ9: https://www.eclipse.org/openj9/
- HotSpot JDK: https://openjdk.java.net/groups/hotspot/
- Oracle JDK: https://www.oracle.com/uk/java/technologies/javase-downloads.html
- GraalVM: https://www.graalvm.org/
- Azul (Zulu): https://www.azul.com/downloads/zulu-community/
- Amazon JDK: https://aws.amazon.com/corretto/

Runtimes

- Open Liberty Site: https://openliberty.io/
- Open Liberty Guides: https://openliberty.io/guides
- Quarkus Site: https://quarkus.io/

Other Useful Links

- Jakarta EE Homepage: https://jakarta.ee/
- OpenJ9 JIT Server Doc: https://www.eclipse.org/openj9/docs/jitserver/
- OpenJ9 Class Data Sharing Doc: https://www.eclipse.org/openj9/docs/shrc/
- OpenJ9 Idle Tuning Doc: https://www.eclipse.org/openj9/docs/xxidletuninggconidle/



References to research for this talk

- https://snyk.io/blog/36-of-developers-switched-from-oracle-jdk-to-an-alternate-openjdk-distribution-over-the-last-year/
- https://en.wikipedia.org/wiki/List_of_Java_virtual_machines
- https://www.guru99.com/java-virtual-machine-jvm.html#:~:text=Java%20Virtual%20Machine%20(JVM)%20is,code%20for%20a%20particular%20system.
- https://dzone.com/articles/jvm-architecture-explained
- https://www.quora.com/Whats-the-attraction-of-the-JVM-platform
- https://stackify.com/java-performance-tuning/
- https://en.wikipedia.org/wiki/Comparison_of_Java_virtual_machines
- https://www.baeldung.com/java-profilers

Thank You

Jamie Lee Coleman
Software Developer/Advocate @ IBM
jlcoleman@uk.ibm.com
@Jamie_Lee_C