Stephen Hellberg

Support Architect, IBM

OpenJDK

OpenJ9

Compelling Java for Cloud Workloads

Stephen Hellberg

Long-standing member of IBM support team for:

IBM Java,IBM SDK for ...[Node.js, Apache Spark]

IBM Runtimes

Supporting Java Since Version 1.2.2

Open Source, Security, occasional Speaker!

Please note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Outline

- This talk is about a JVM but its also about evolution, new environments, new economics and new opportunities

Java is ubiquitous

#1 programming language

TOBIE 2019 index ranks Java as the most popular programming language, again.

#1 developer platform in the cloud

Java is the language of choice in the Cloud.
Developers are productive in Java.

12 million developers

With millions of skilled Java developers invested in the platform, the future is bright!

Used in 80% of worldwide enterprises

From construction to finance, and retail to communications, Java powers the world's economy.

Over 38 billion active JVMs

That's more than four for every person on the planet!
Java is everywhere.

Vendor competition *and* collaboration delivered

- The **fastest** runtime environments
- The **most scalable** runtime environments
- The **best** garbage collectors
- The **greatest** dynamically re-optimizing compilers

The best environment for long running Java only applications

Let’s think about what drives us

Economics

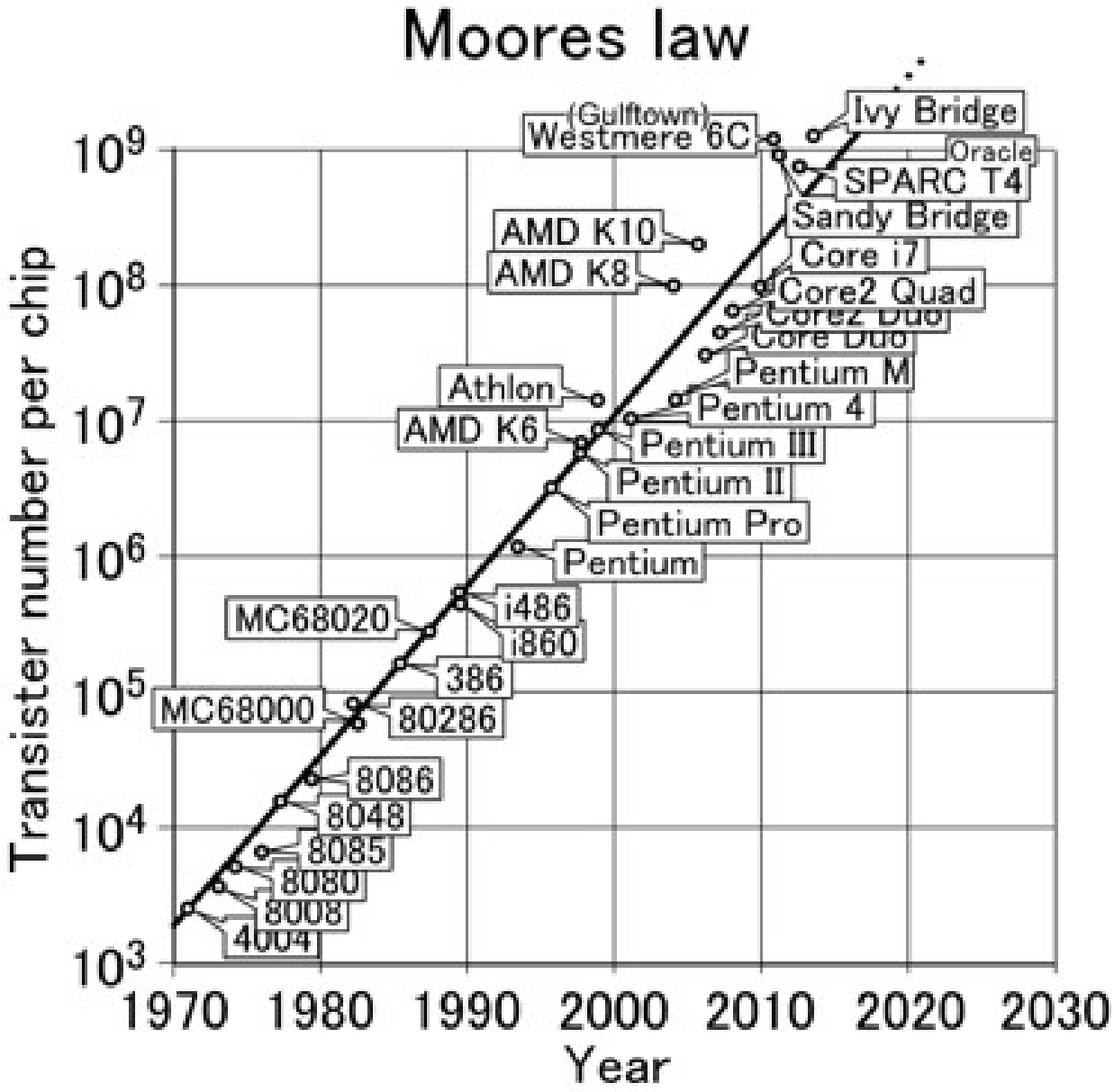
Faster, Cheaper, Better

For many years Java innovation focused on performance

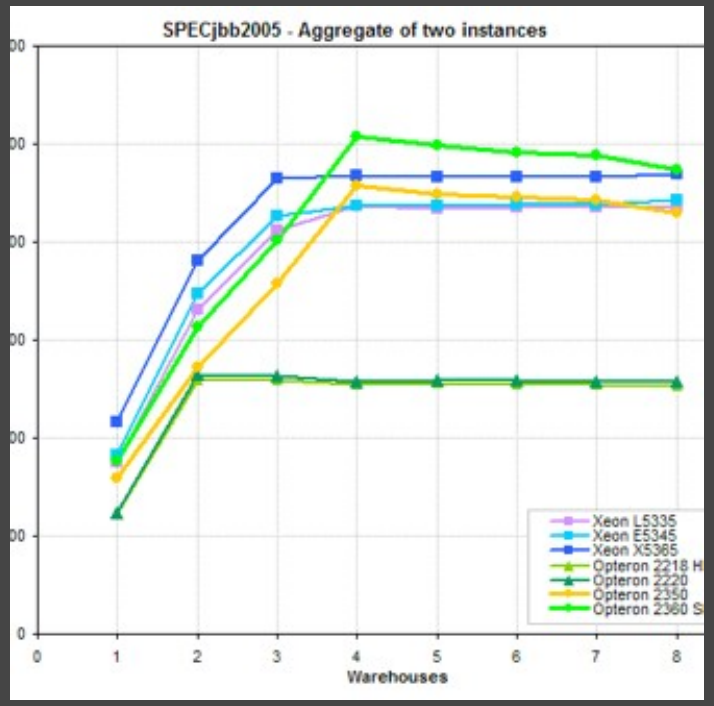
For a very long time Java and the JVM have had one evolutionary pressure

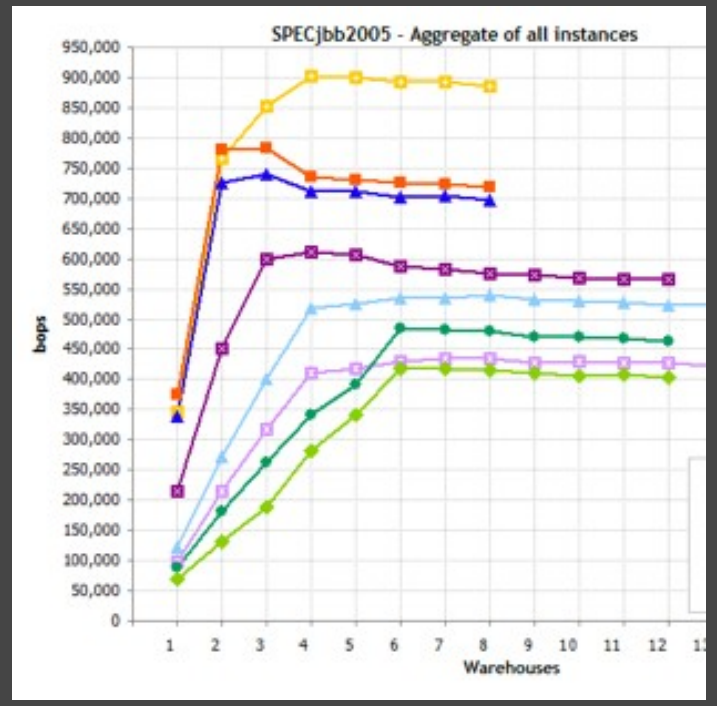
Maximize the opportunities offered by Moore's Law

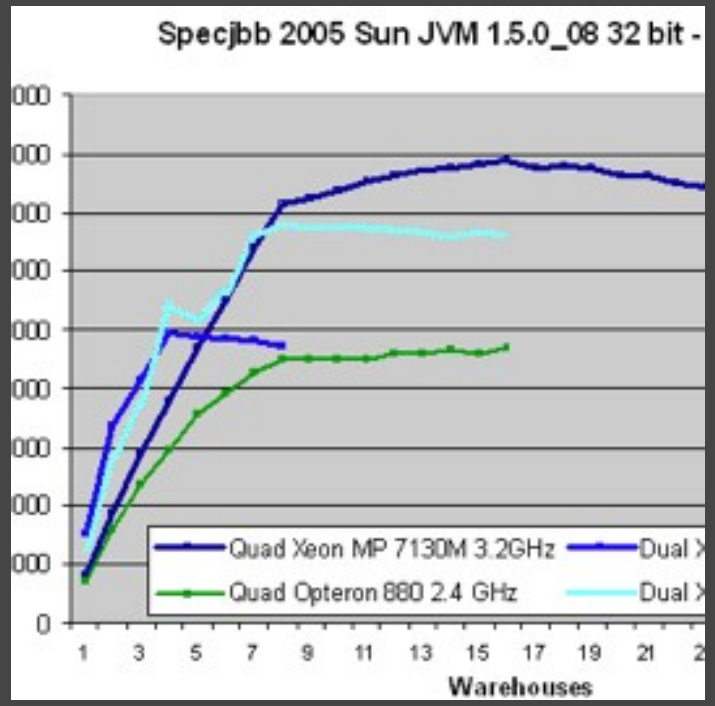
Forever...



https://commons.wikimedia.org/wiki/File:Moores_law_(1970-2011).PNG





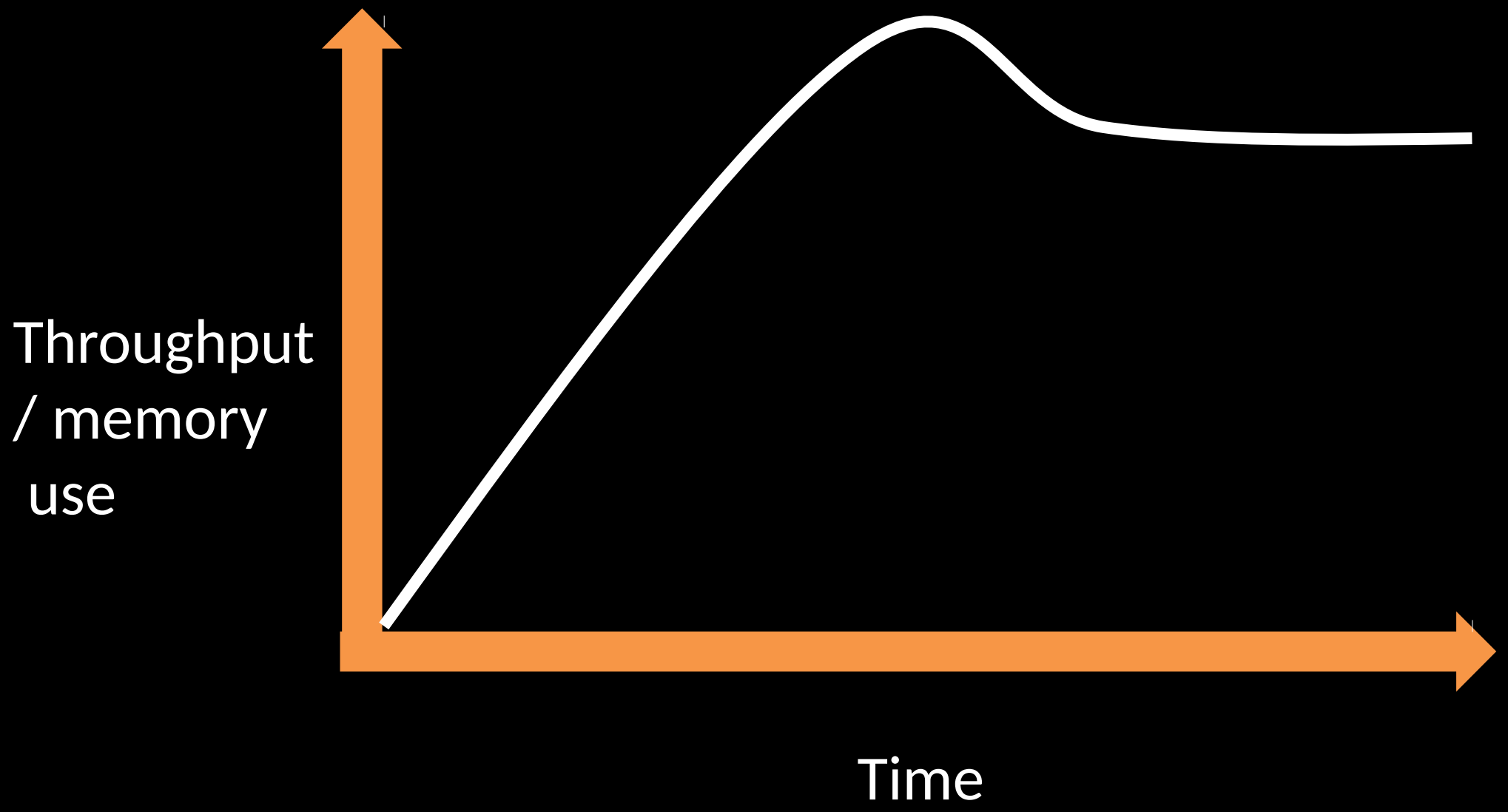


Java has tuned itself to this pressure

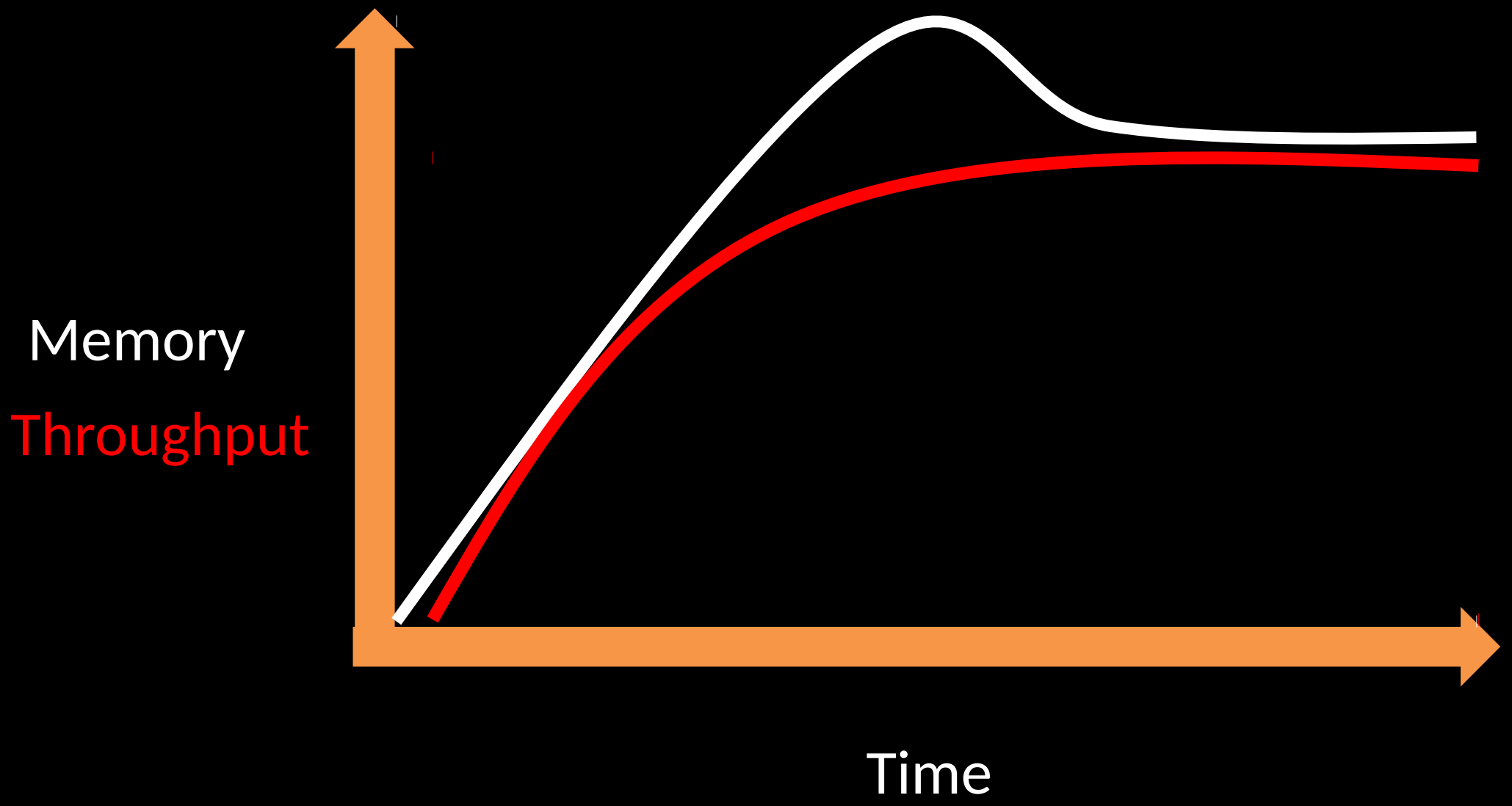
We built a walled garden and made it the best place to run enterprise applications of a certain kind

https://www.flickr.com/photos/40139809@N00/

Traditional profile



Traditional profile



But, the world doesn’t stand still!

Even when Moore’s law broke

- The JVM didn’t have to do much – it already had a good multi-cpu story.
- But Java needed to change - and we added streams and lambdas..
- Not as quick to deliver as we’d like
- Just about made it.
- Our Java survived

So far our Java releases have given us great performance

For long running, multi-core applications

Exactly what we needed

Exactly what the economics required.

So far our Java releases have given us great performance

From the smallest devices to the largest

As well as real-time and soft-real-time

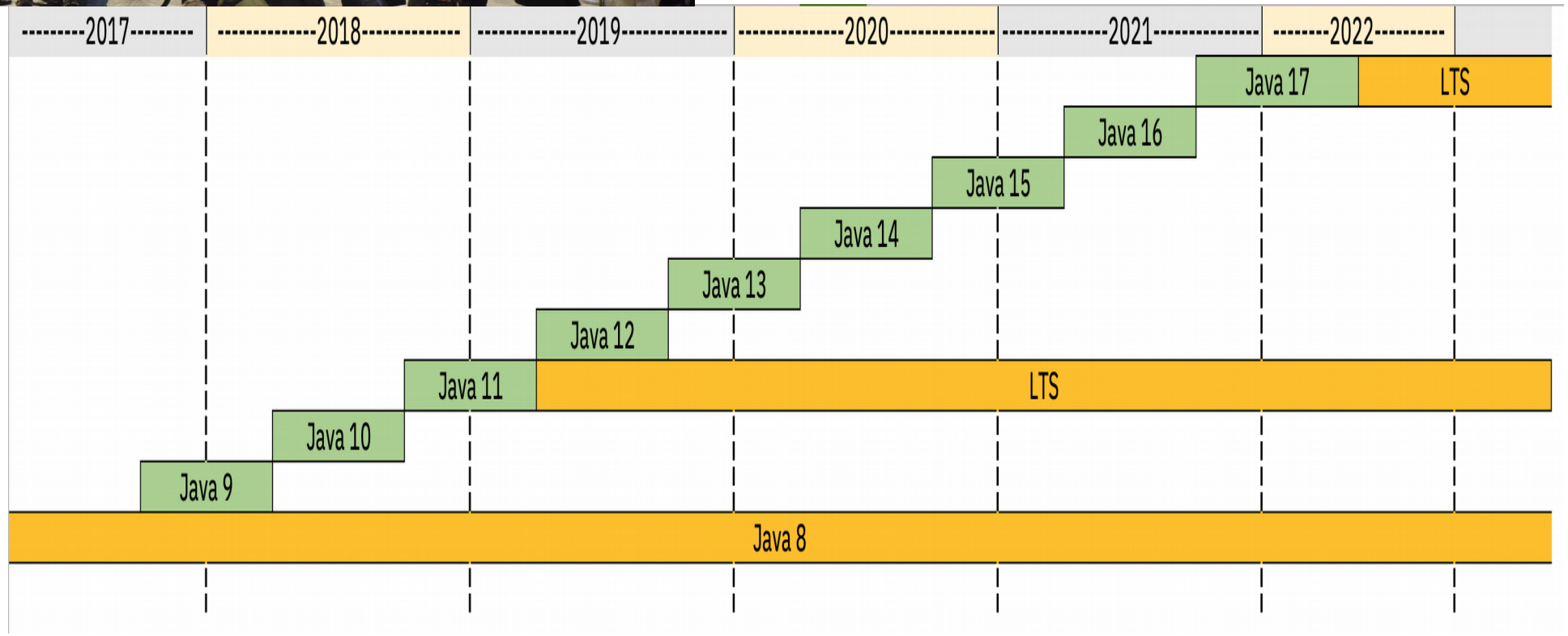
An ecosystem of JVMs

But things have changed.

A new race is here

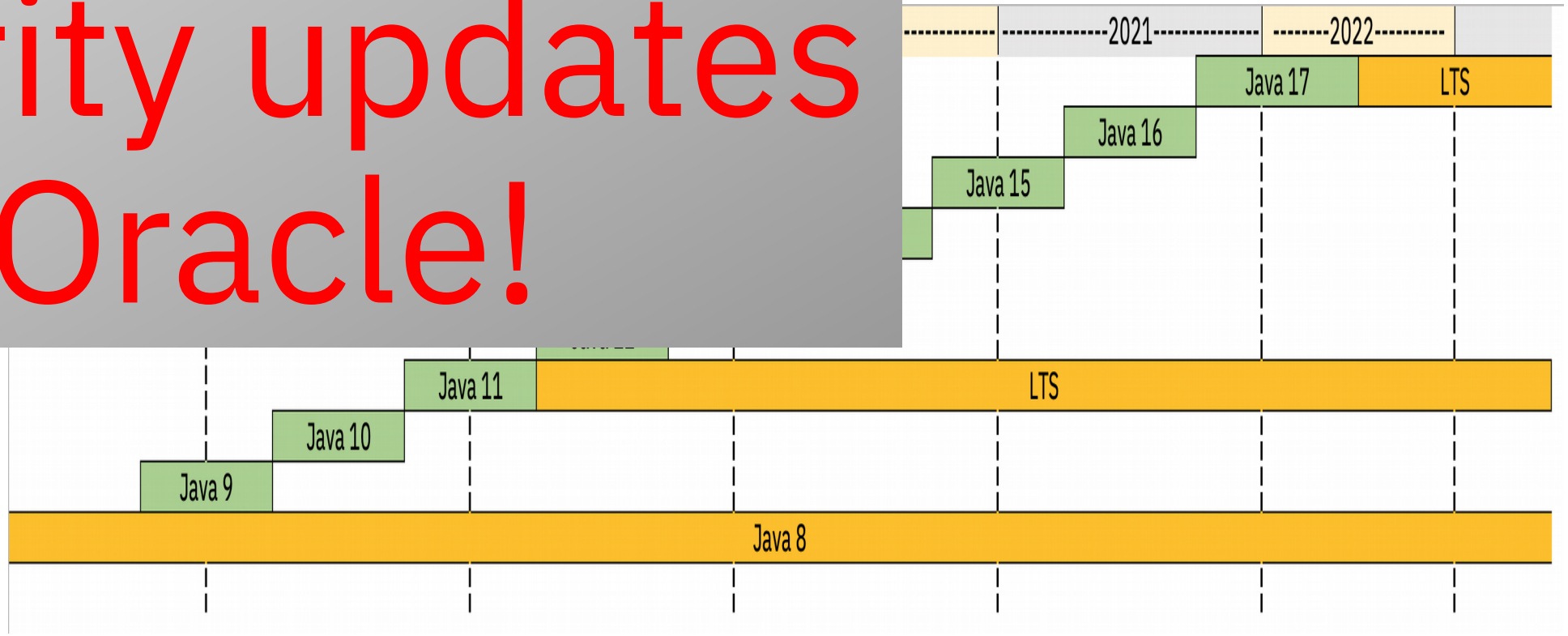
new economics
new environments

All Change!


2017
2018
2019
2020
2021
2022
Java 17
LTS
Java 16
Java 15
Java 14
Java 13
Java 12
Java 11
LTS
Java 10
Java 9
Java 8

All Change!

No more free security updates from Oracle!



Runtime Language

Type Safe

Bytecode: JIT Compiled

Garbage Collected

Concurrent Threaded

All Platforms

The JVM's design characteristics allow us to imagine taking it to new places

No other runtime environment comes close

Java & the JVM is an enabler for the future

The JVM enables you to move your application to new environments.

JVM

Your model

Multiple real applications


Does ‘Java’ have a future?

But the fact is that Java’s innovation is mostly driven by factors outside our control

Aka “Cloud”

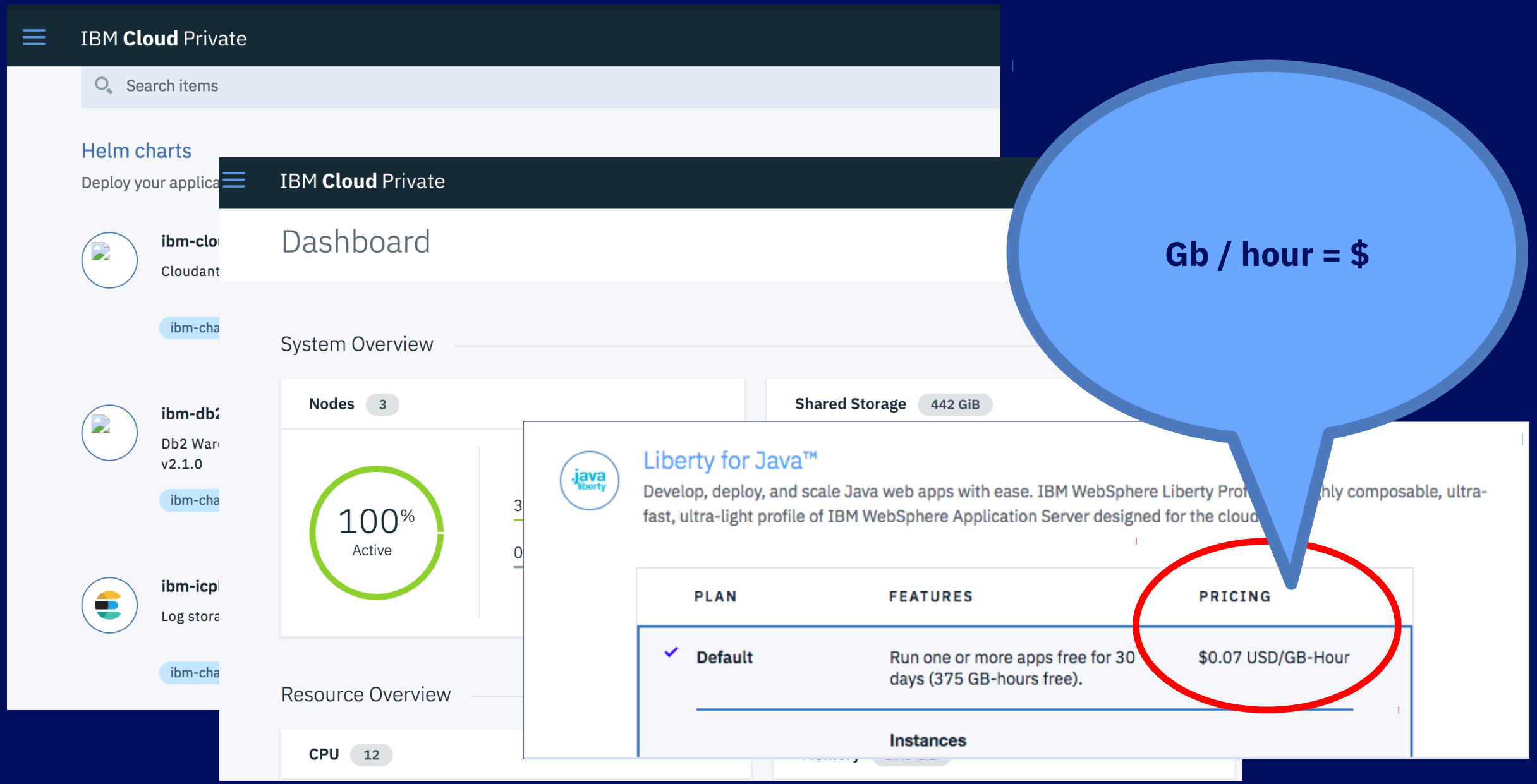
Faster, cheaper, easier, better …

Economics still rules

Faster, Cheaper, Better

Cloud economics and new programming models have changed the game…

Dynamic compute instances, pay for what you use....



Cloud isn’t going away:
in fact its coming to you

What ‘Cloud’ promises

a *virtual*, *dynamic* environment which maximizes use, is *infinitely scalable*, *always available* and needs *minimal upfront investment* or commitment

Take your code – host it on someone else's machine and pay **only** for the resource you use for the time you use it
AND be able to do that very quickly and repeatedly in parallel

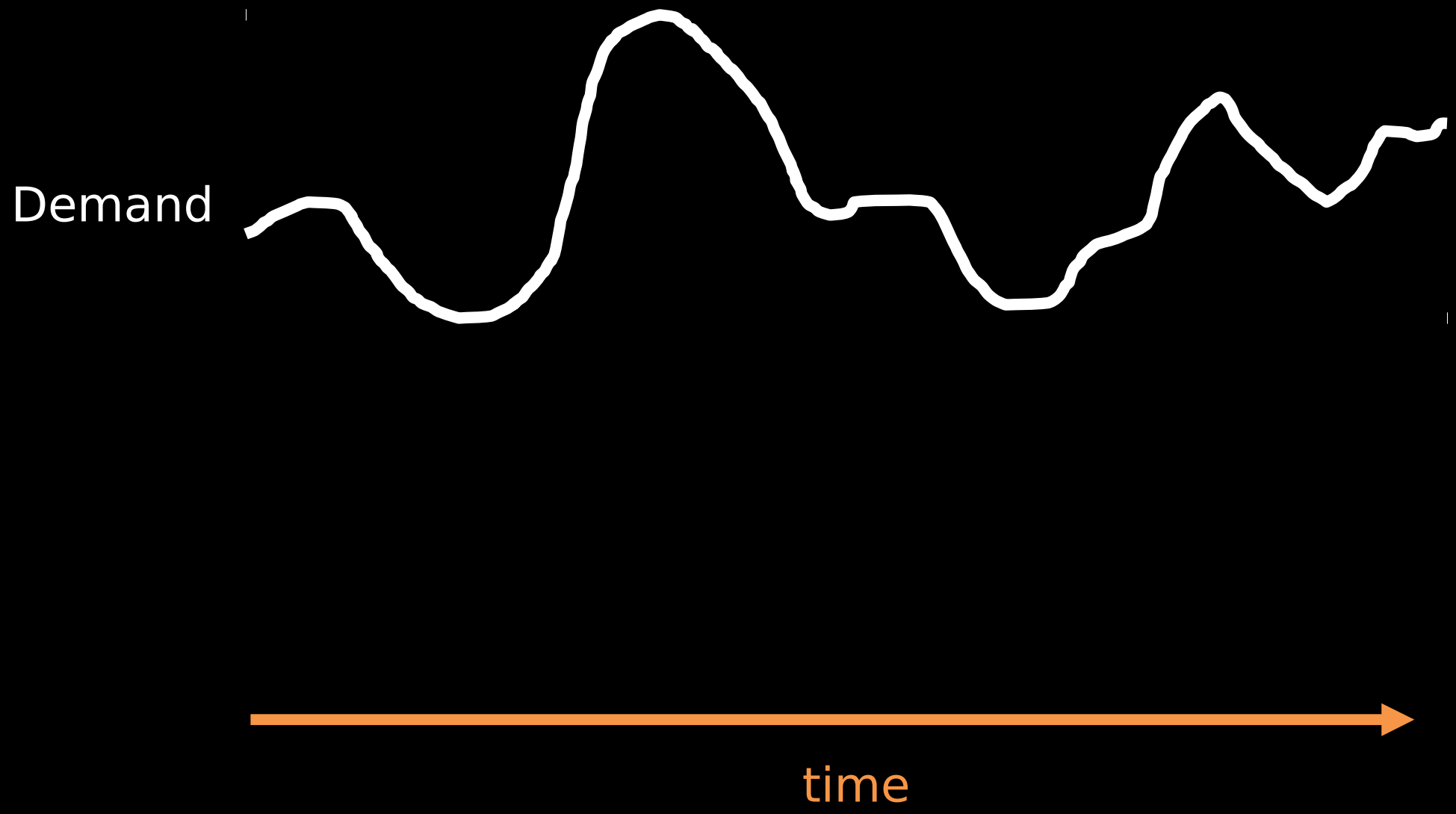
“Compute on demand” – it’s what we’ve always wanted

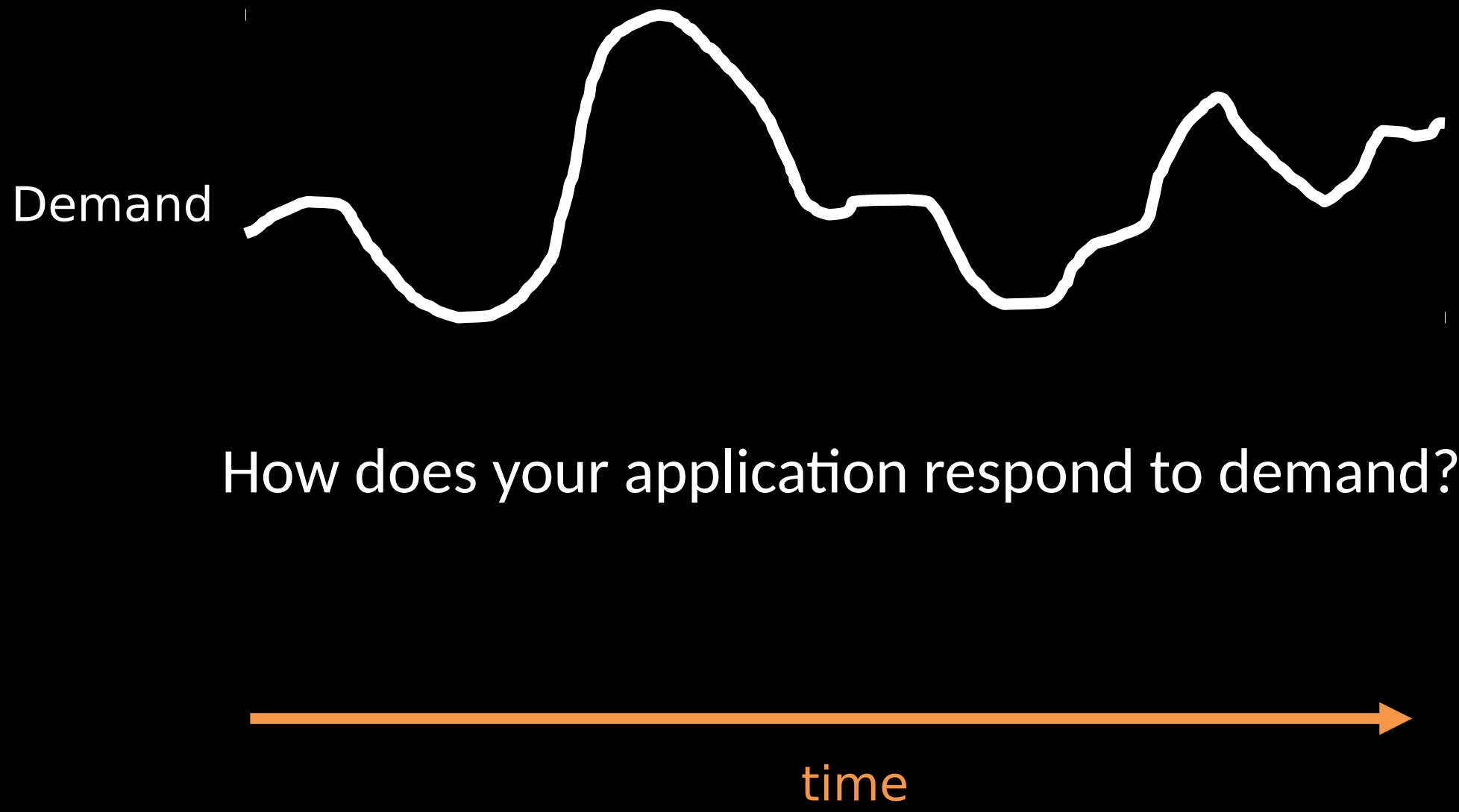
https://www.flickr.com/photos/vuhung/

Cloud Economics

We really are getting closer all the time to ‘Compute on Tap’

https://www.flickr.com/photos/leunix/

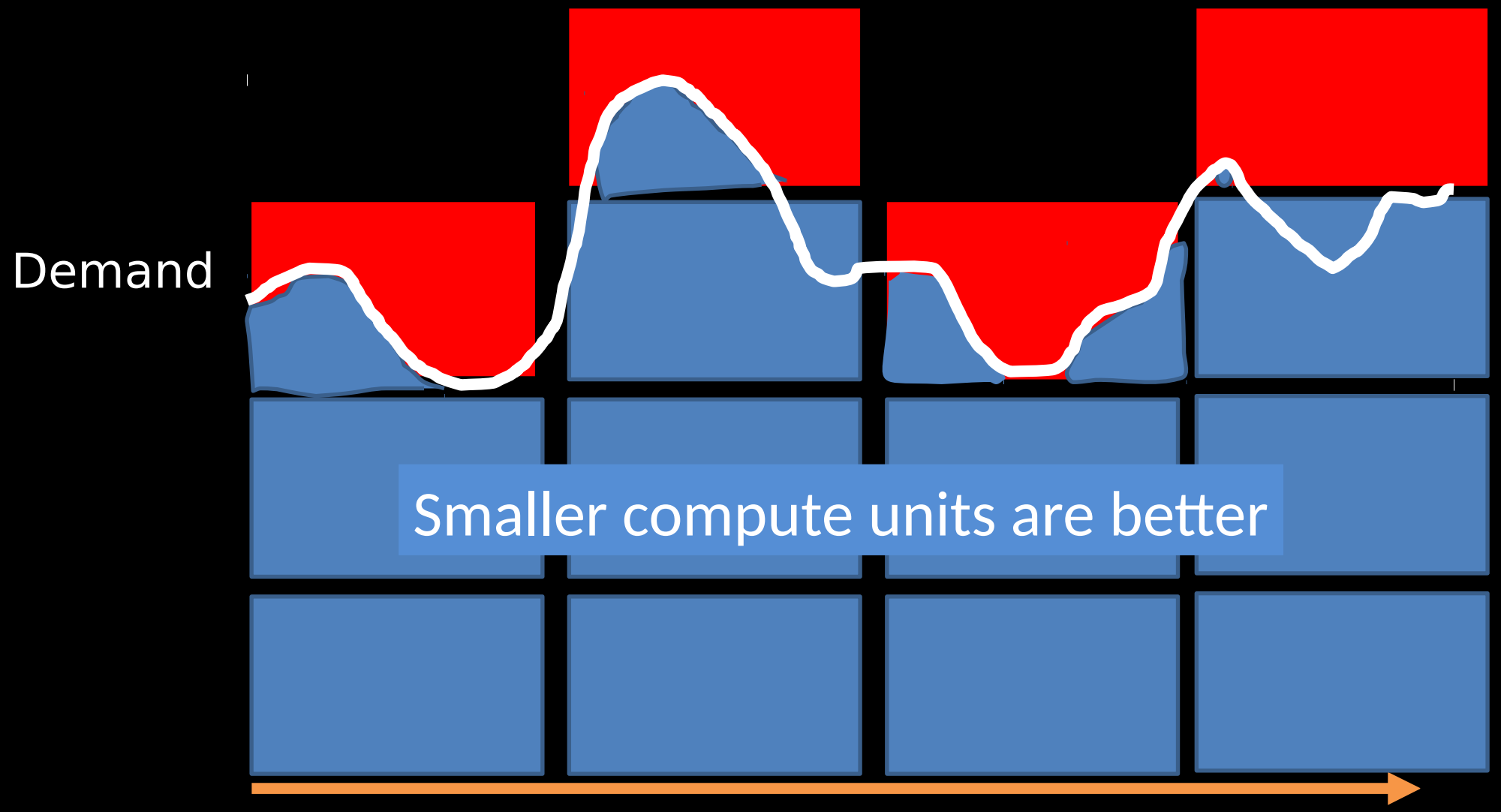

Demand
time

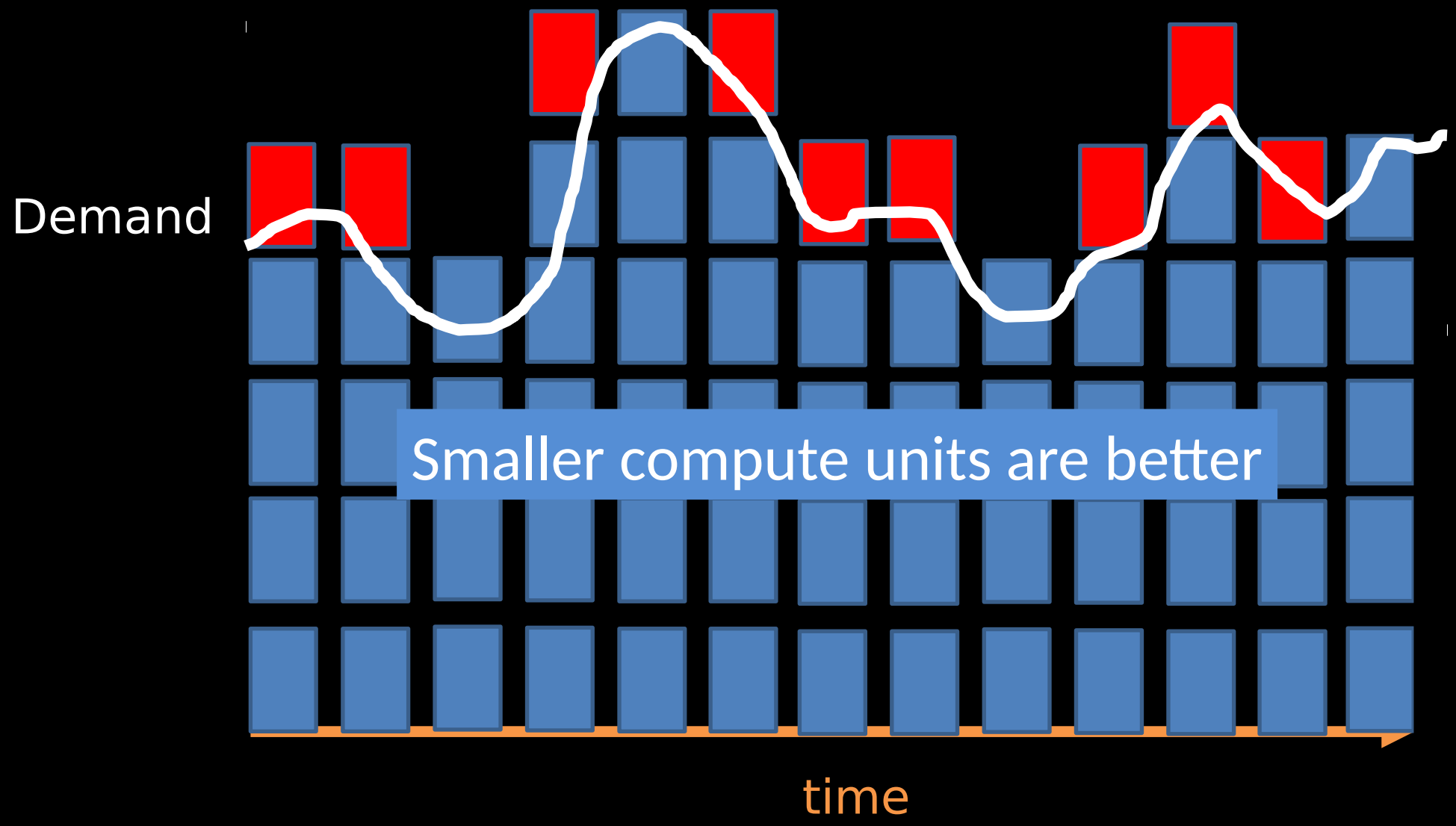


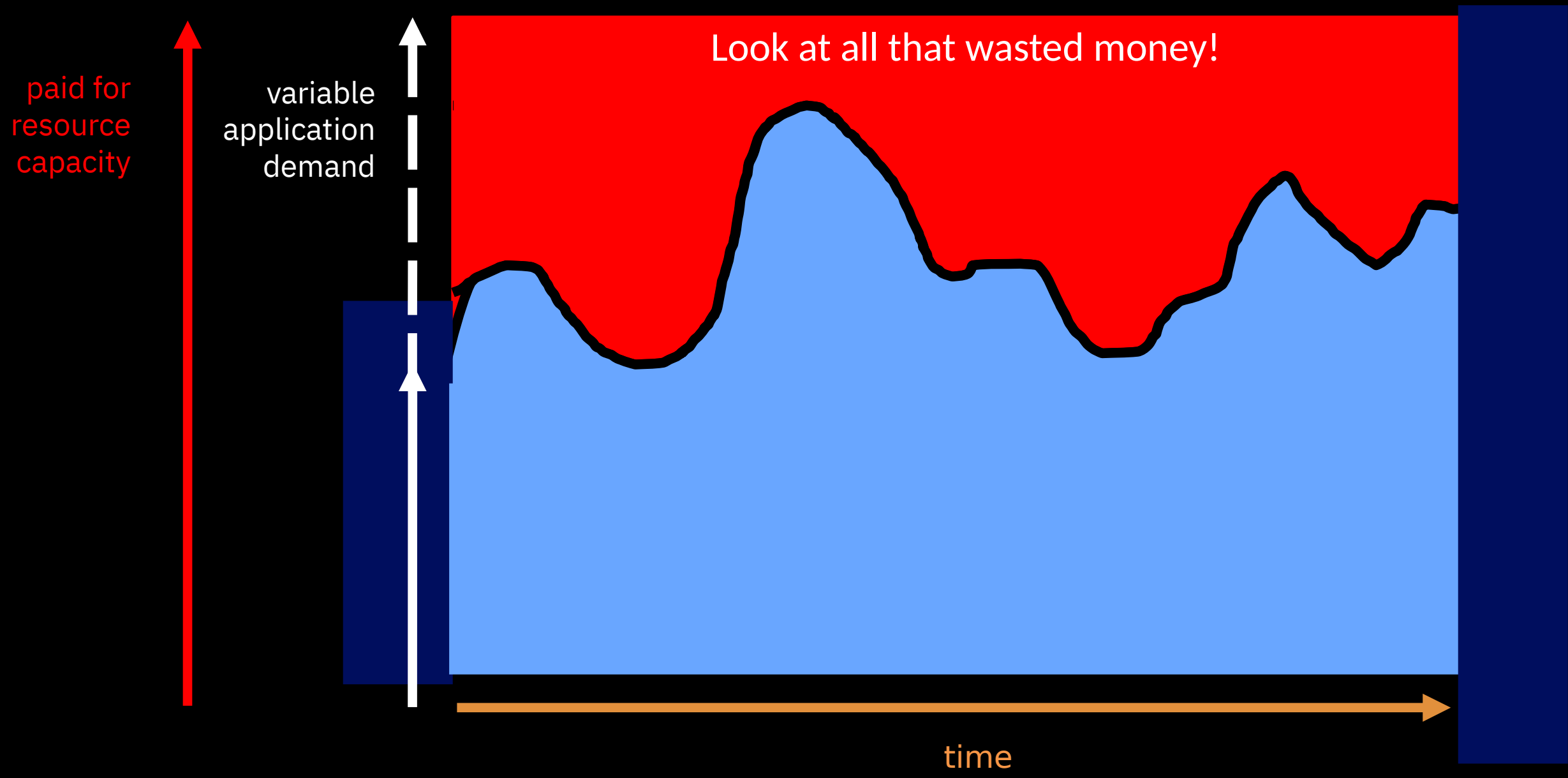
How does your application respond to demand?


Demand
One big server running all the time?
time


Demand
Look at all that wasted money!
time

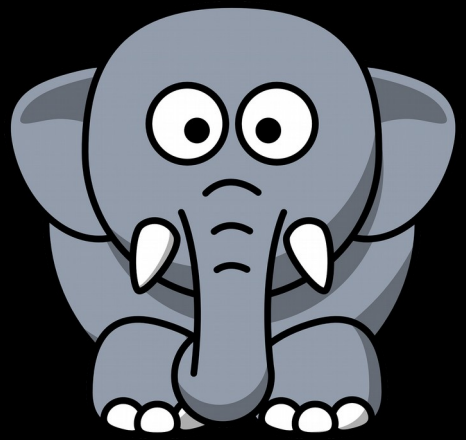

Demand
Smaller compute units are better

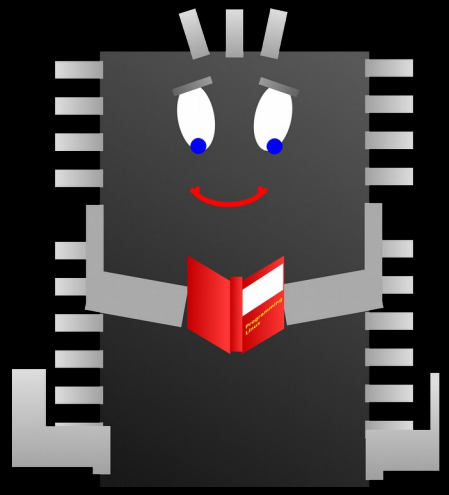

Demand
Smaller compute units are better
time



Cloud demands:

- Small runtime memory footprint
- Small deployment sizes
- Fast starting applications
- No resource usage when idle





Cloud computing: compute == money

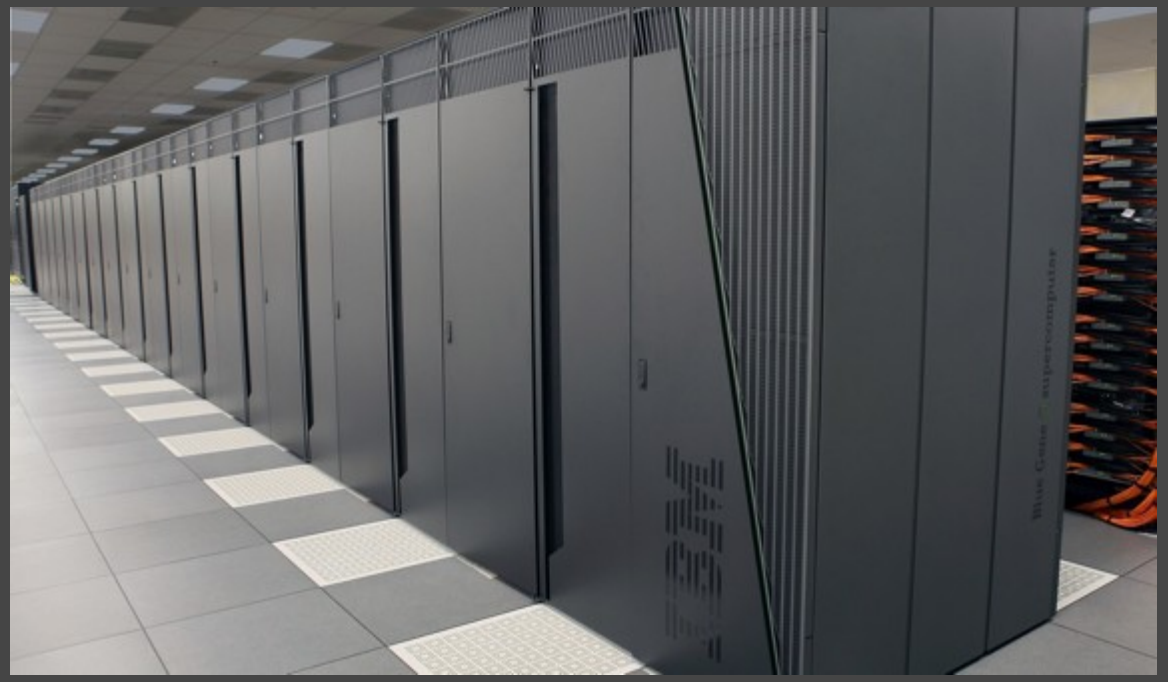
Money changes everything

With a **measureable** 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.

How does OpenJ9 Help?



Designed for small environments and large.
From megabytes to terabytes
For the widest range of CPUs , architectures and operating systems.

Eclipse Open J9

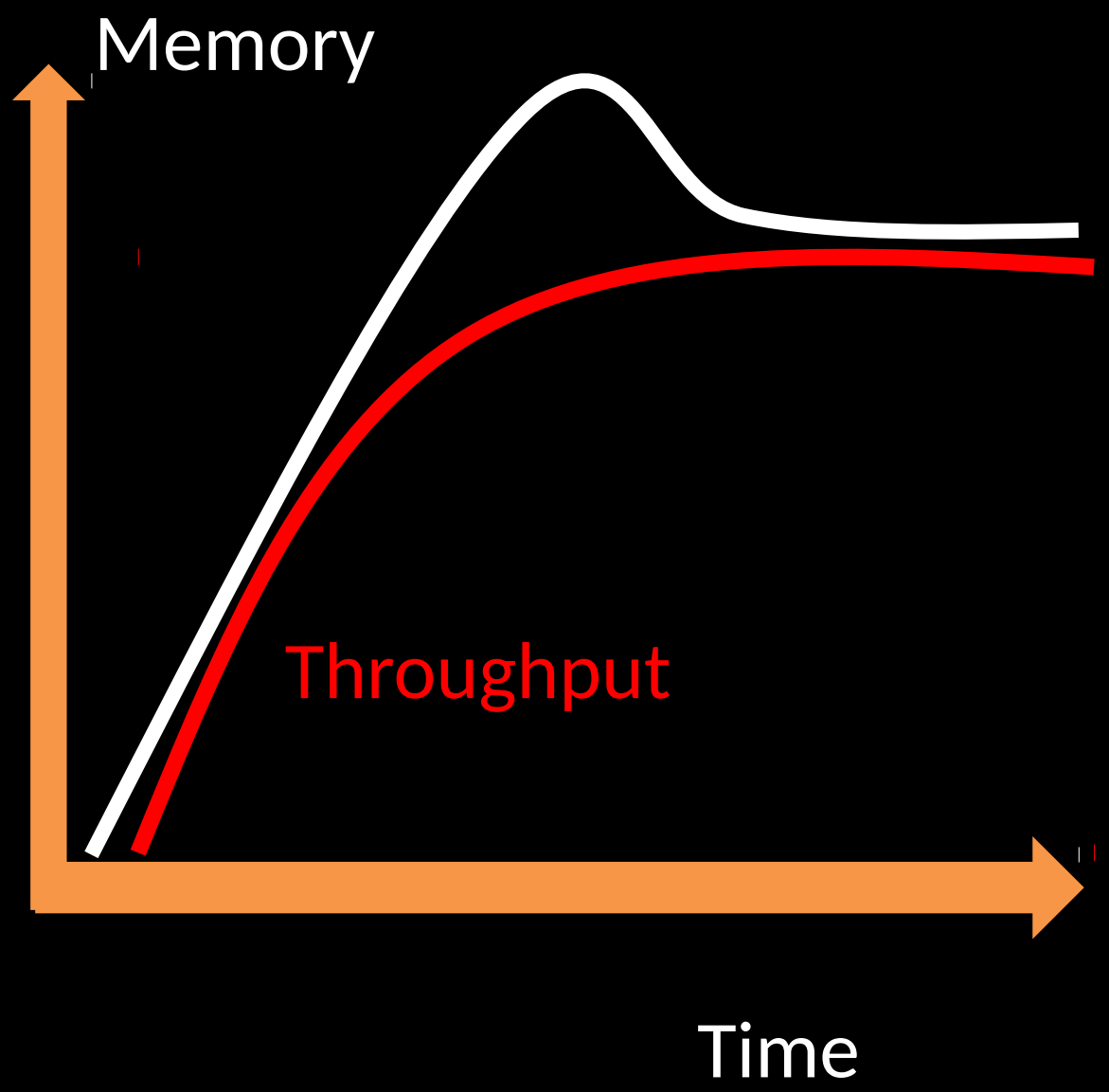
Designed from the start to span all the operating systems needed by IBM products

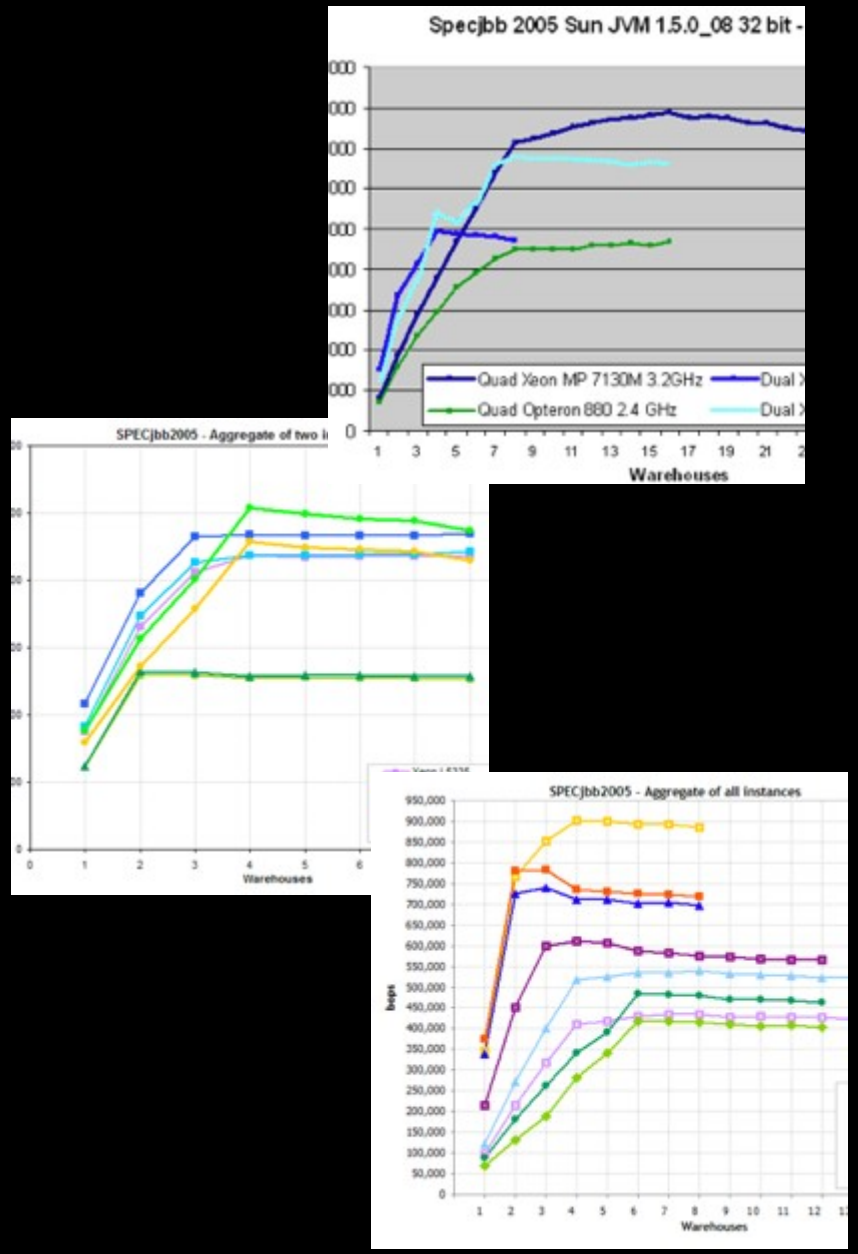
This JVM can go from small to large

Can handle constrained environments or memory rich ones

Is used by the largest enterprises on the planet

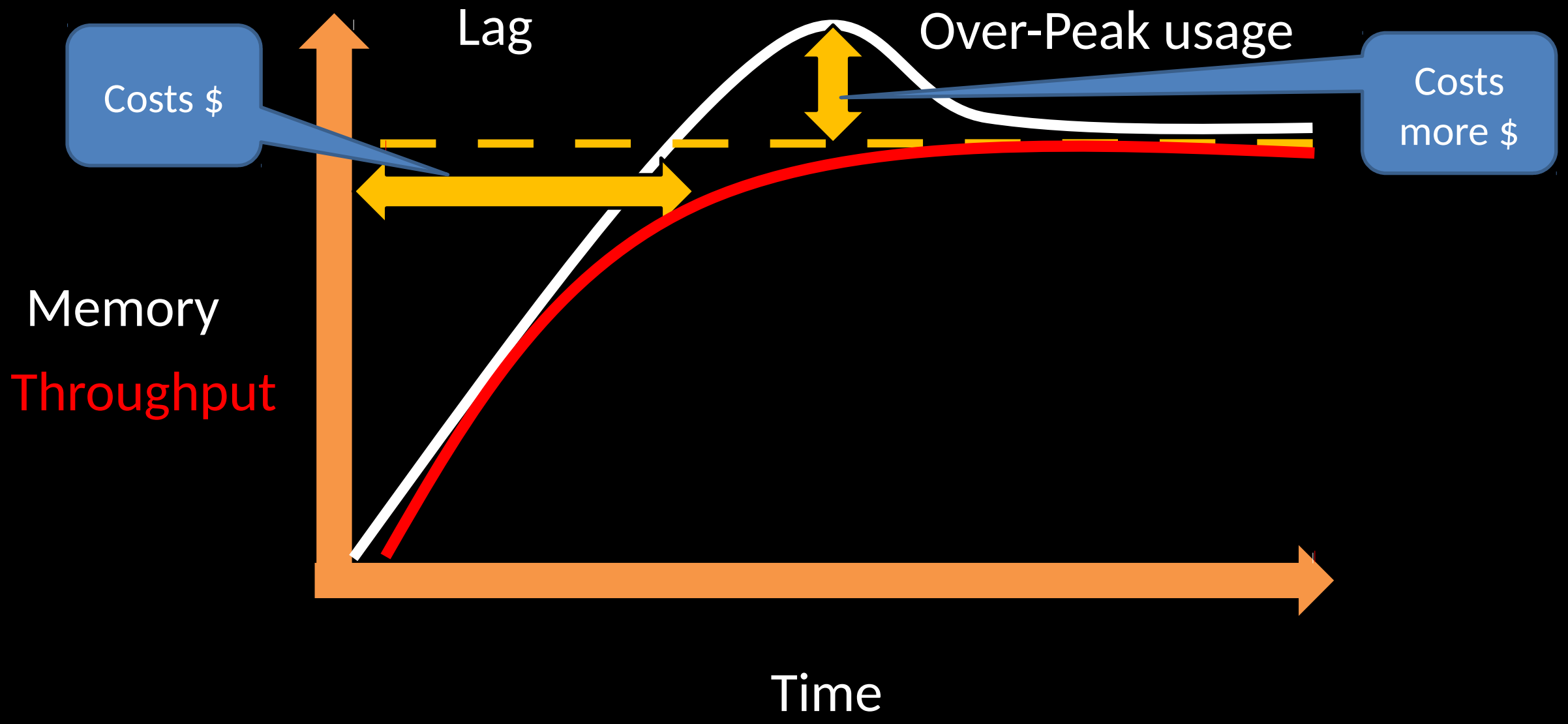
If any JVM can be said to be at the heart of the enterprise – its this one.





Standard Profiles look like this

But this shape does not work so well for the cloud!



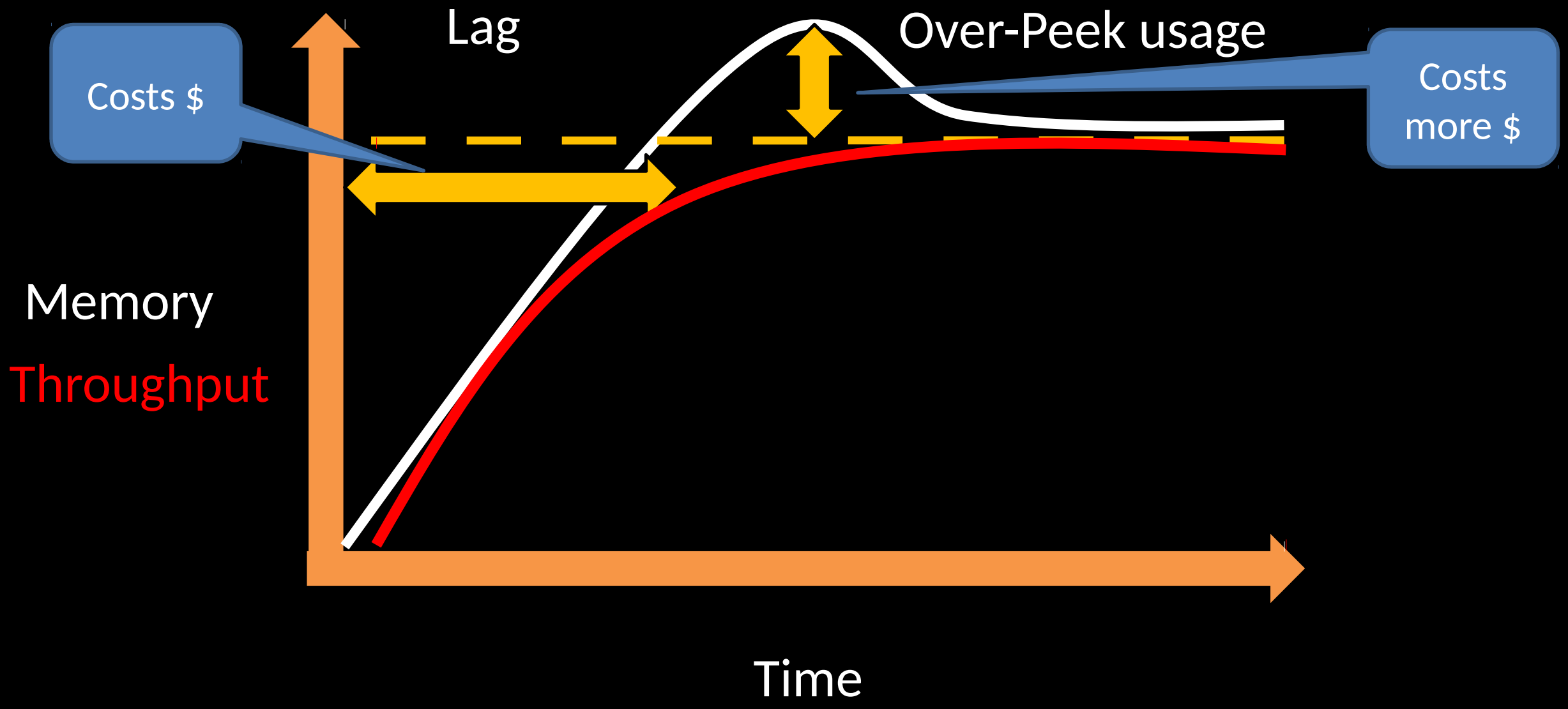
compute == money

$ == GB/hr

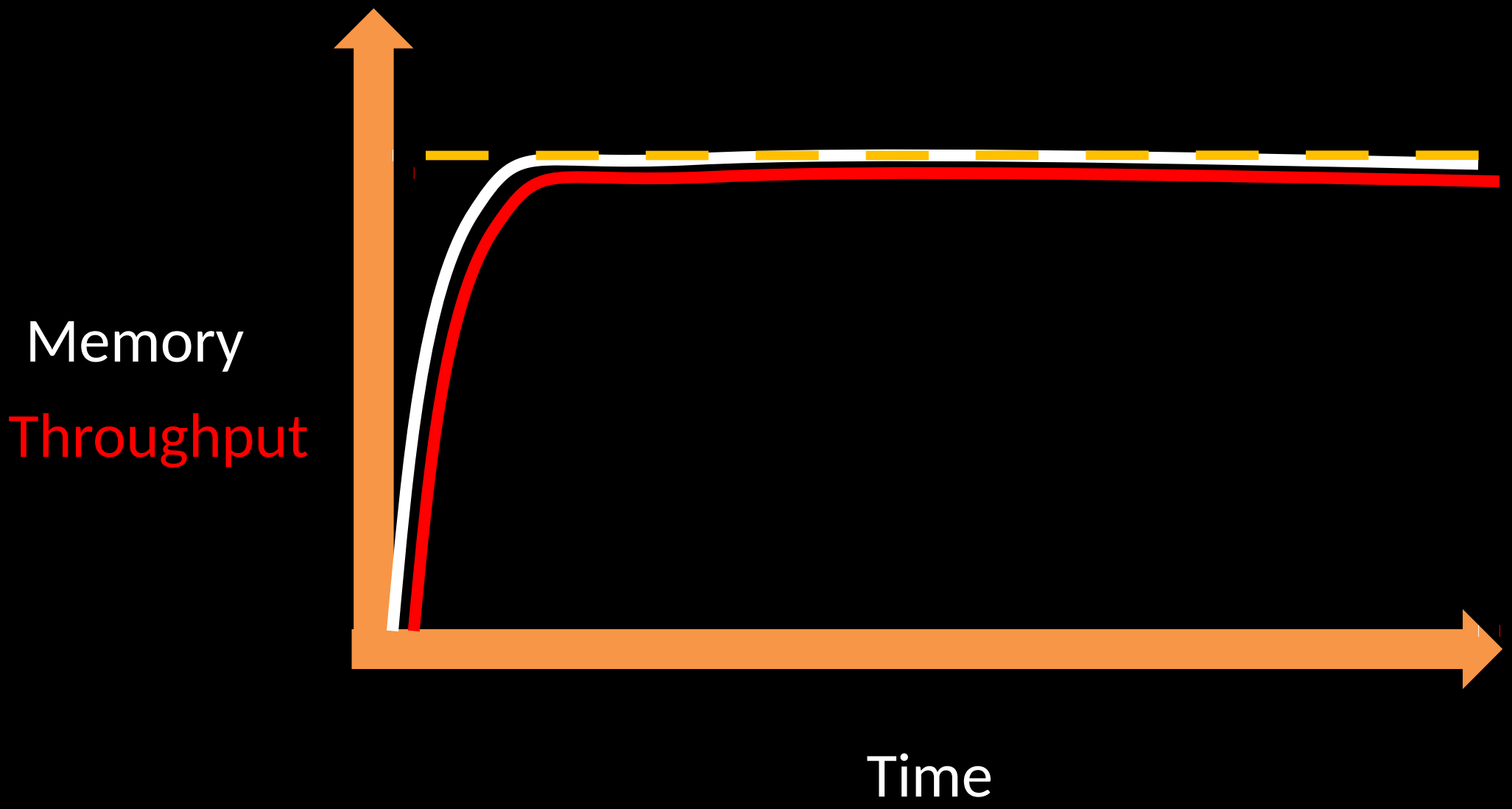
-Xmx: $100

IBM

Doesn't fit new model



More like this please



Java ME Inside!

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
  - Your game should not play better the longer you play

Java Cloud requirements

- Small footprint
  - Improves density for providers
  - Improves cost for applications
- Fast startup
  - Faster scaling for increased demand
- Quick / immediate rampup
  - GB/hr is key, if you run for less time you pay less money

OpenJ9 may may have its roots in small devices..

IBM z14 at a glance

System, Processor, Memory
Five hardware models: M01, M02, M03, M04, M05
10 core 5.2GHz 14nm PU SCM
1 - 170 PUs configurable as CPs, zIIPs, IFLs, ICFs
Increased Uniprocessor capacity
Up to 33 sub capacity CPs at capacity settings 4, 5, or 6
CPC Drawers and backplane Oscillator
Enhanced SMT and new instructions for SIMD
Enhanced processor/cache design with 1.5x more on-chip cache sizes
Up to 32 TB DRAM, protected by Redundant Array of Independent Memory (RAIM)
Virtual Flash Memory (VFM)
192 GB HSA
Improved pipeline design and cache management

Announce: July 17, 2017

I/O Subsystem, Parallel Sysplex, STP, Security
PCIe Gen3 I/O fanouts with 16 GBps Buses
6 CSS, 4 Subchannel sets per CSS
0 – 5 PCIe I/O Drawer Gen3 (no I/O Drawer)
Next generation FICON Express16S+
10 GbE RoCE Express2
Integrated Coupling Adapter (ICA SR) and Coupling express LR for coupling links
Support for up to 256 coupling CHPIDs per CPC
CFCC Level 22
Crypto Express6S and CMPSC compression and Huffman Coding compression
STP configuration and usability enhancements (GUI)
IBM zHyperLink Express
OSA-Express6S
Secure Service Container

RAS, simplification and others	
L3 Cache Symbol ECC	Acoustic and thin covers (space saving)
Enhanced Dynamic Memory Relocation for EDA and CDR	Virtual Flash Memory (replaces IBM zFlash Express)

But it runs just as well on the largest

OpenJDK OpenJ9

66%
smaller footprint

42%
faster start-up

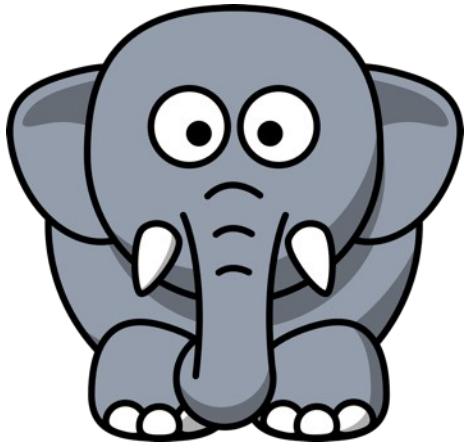
3x
faster to peak performance
in constrained environments

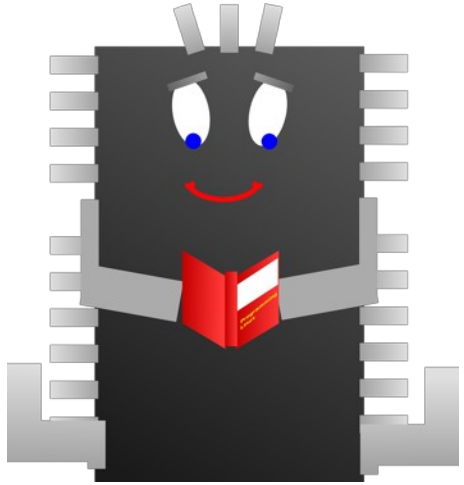
100%
throughput performance

https://www.eclipse.org/openj9/oj9_performance.html

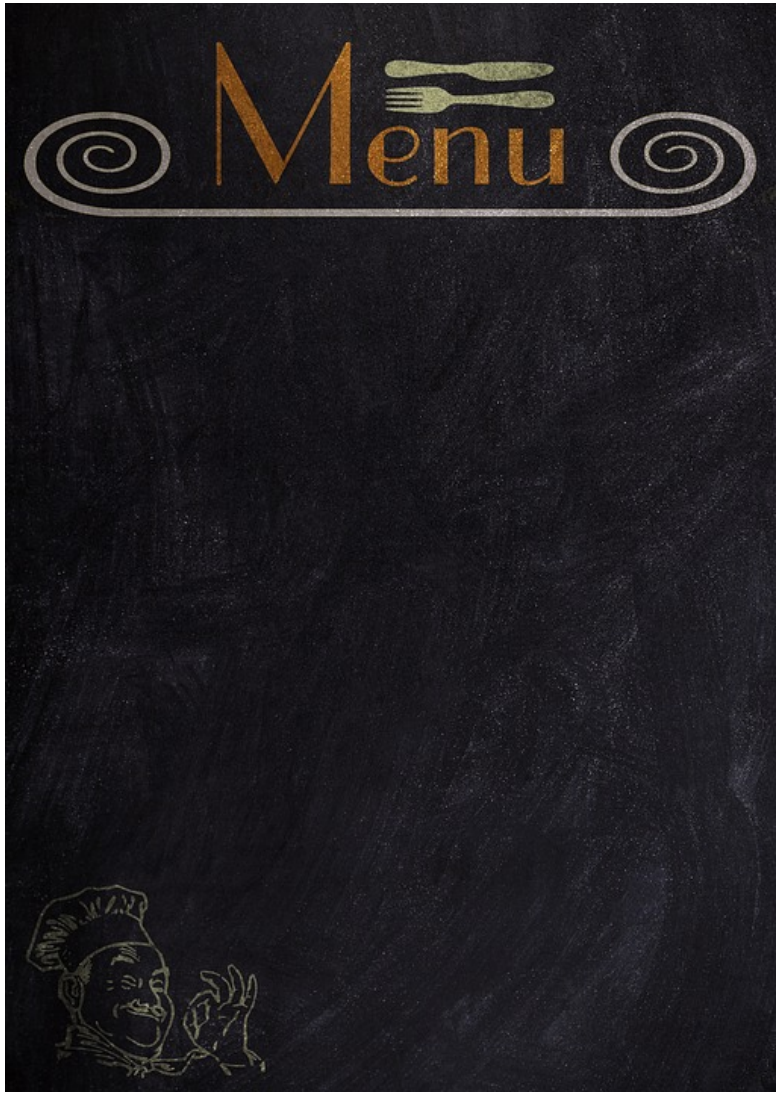
Key elements

- Designed for scaling from the smallest to the largest
- Comes with several custom garbage collectors (even a soft-real time one)
- Has a class sharing approach that allows sharing of state and constant data in





OpenJ9's Garbage Collection 'Policies'
(aka HotSpot's GC Modes)



Set using `-Xgcpolicy:`*<policy>*

`gencon` – Generational GC (Default)

`balanced`
- Large 64-bit heaps, logical (like 'gencon')/physical separation
- Similar conceptually to HotSpot's G1 GC

`optthruput` – Optimised for 'batch'; most efficient GC (in GC terms!)

`optavgpause` – Optimised for 'responsiveness' / interactive appls.

`metronome` - For demanding (soft) real-time appls.
(Reference-counting GC; Designed to deliver hard real-time!)

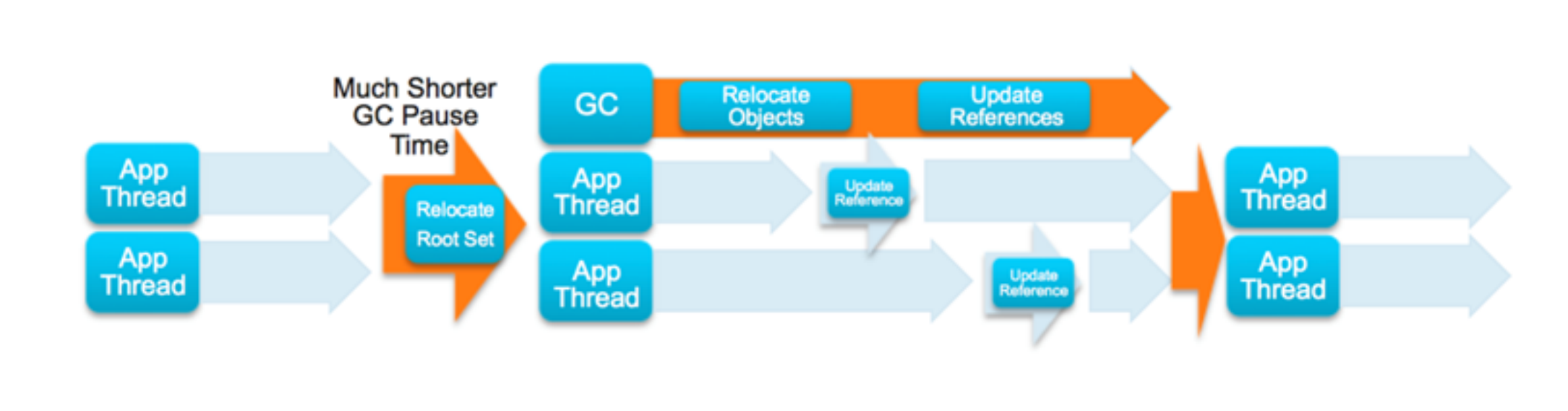
Plus, concurrent Scavenge (pauseless GC) without the need for hardware transactional memory support (PPC and mainframe)

NEW AND IMPROVED

New Pause-less GC vs Traditional GC

Traditional GC Cycle

Pause-Less GC Cycle



X86: Pause-less Garbage Collection
Java Store Inventory and Point of Sale Application

Java GC-tuning made easier

High scavenge pause times made this application a candidate for Pause-less GC

Up to **40%** better throughput for **response-time** constrained Service Level Agreements (SLAs) at an 8% loss to peak throughput (no SLAs)

Up to **22x** better average GC pause-times

Enable Pause-less GC with:

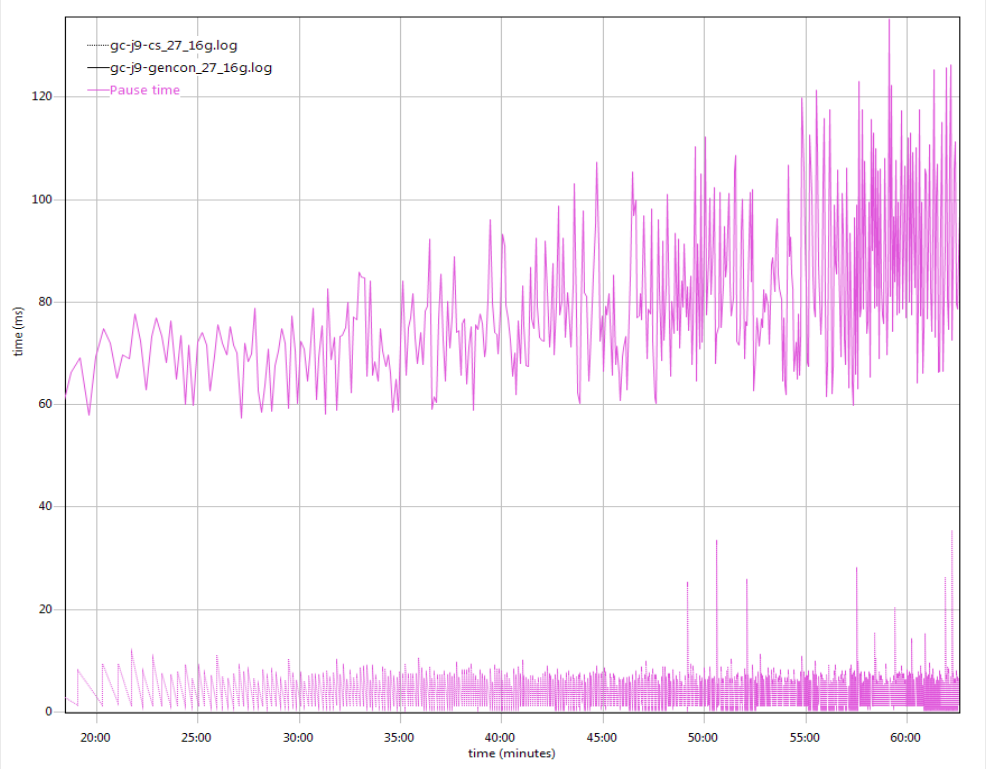
- IBM JDK SR5 FP27 or newer on 64-bit X86
- Available on both Windows and Linux

Pause-less GC implemented via software read barriers (no special hardware support)
JVM option: **-Xgc:concurrentScavenge**

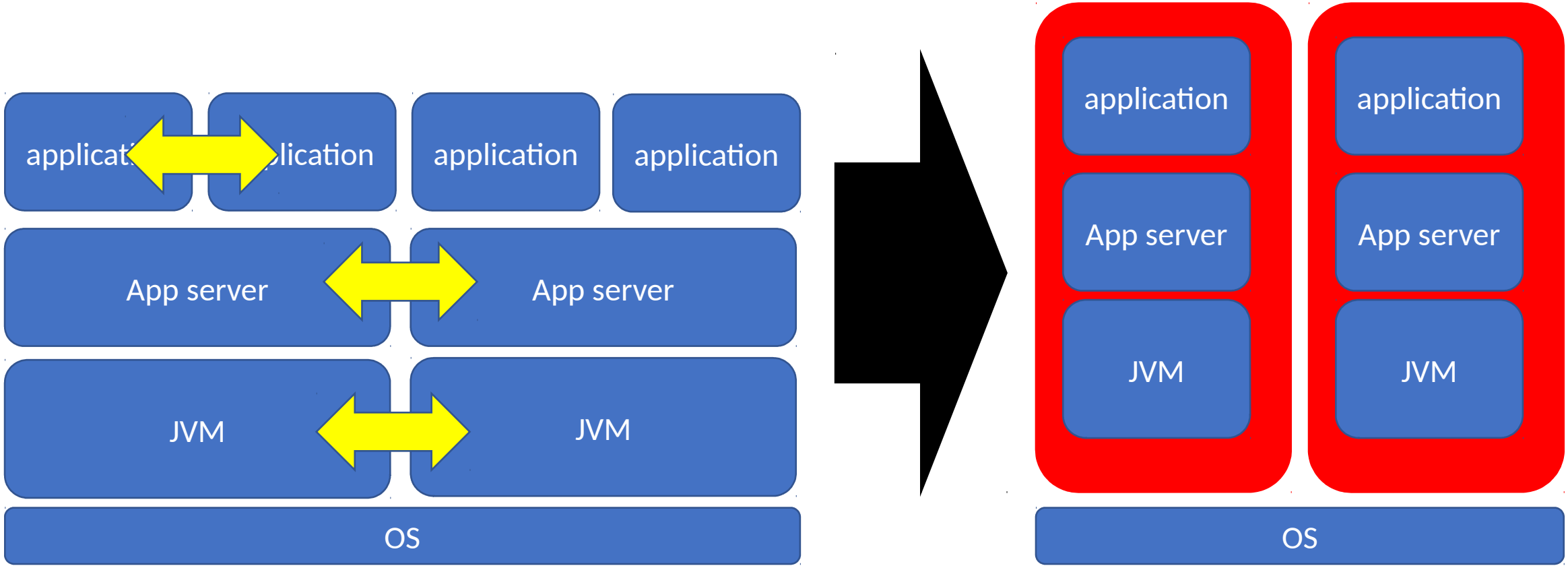
(Controlled measurement environment, results may vary)

Pause time

Variant	Mean time (ms)	Minimum time (ms)	Maximum time (ms)	Total time (ms)
gc-j9-cs_27_16g.log	6.3	0.67	143	35089
gc-j9-gencon_27_16g.log	142	57.3	468	405737



OpenJ9 Shared Classes can work at all levels



Shared Classes cache

-Xshareclasses

  -enables the share classes cache

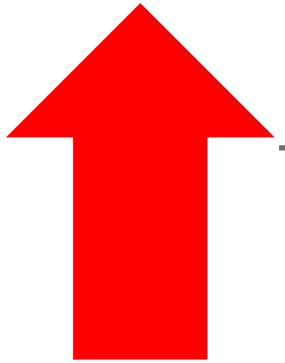
-Xscmx50M

  - sets size of the cache

Available for **application** as well as standard Java class library!

ShareClasses cache

ShareClasses cache



This isn’t the greatest format for running, compiling or even caching

ShareClasses cache

So when loading J9 splits it into two parts

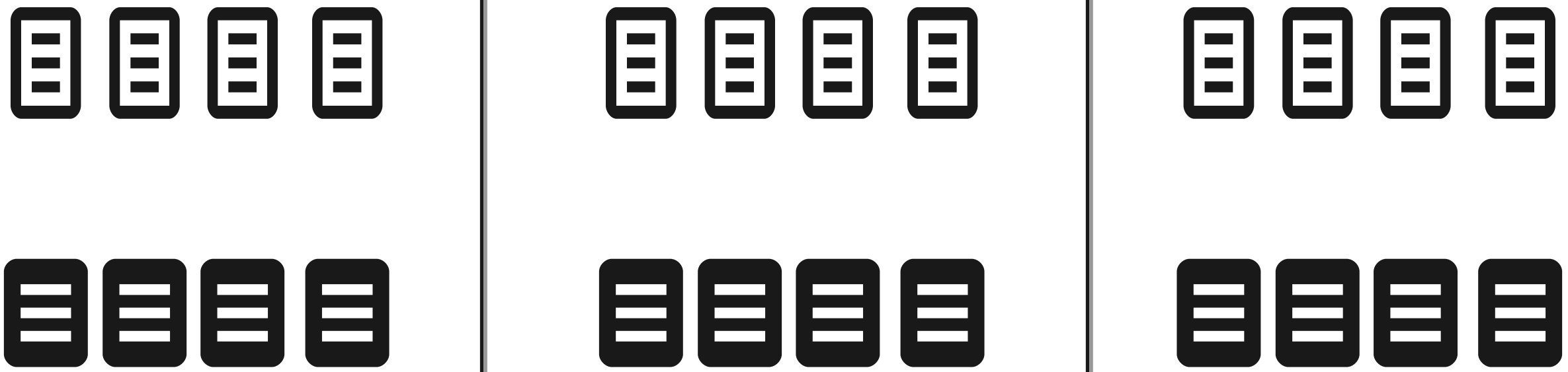
The read only part
Position independent

The stateful part

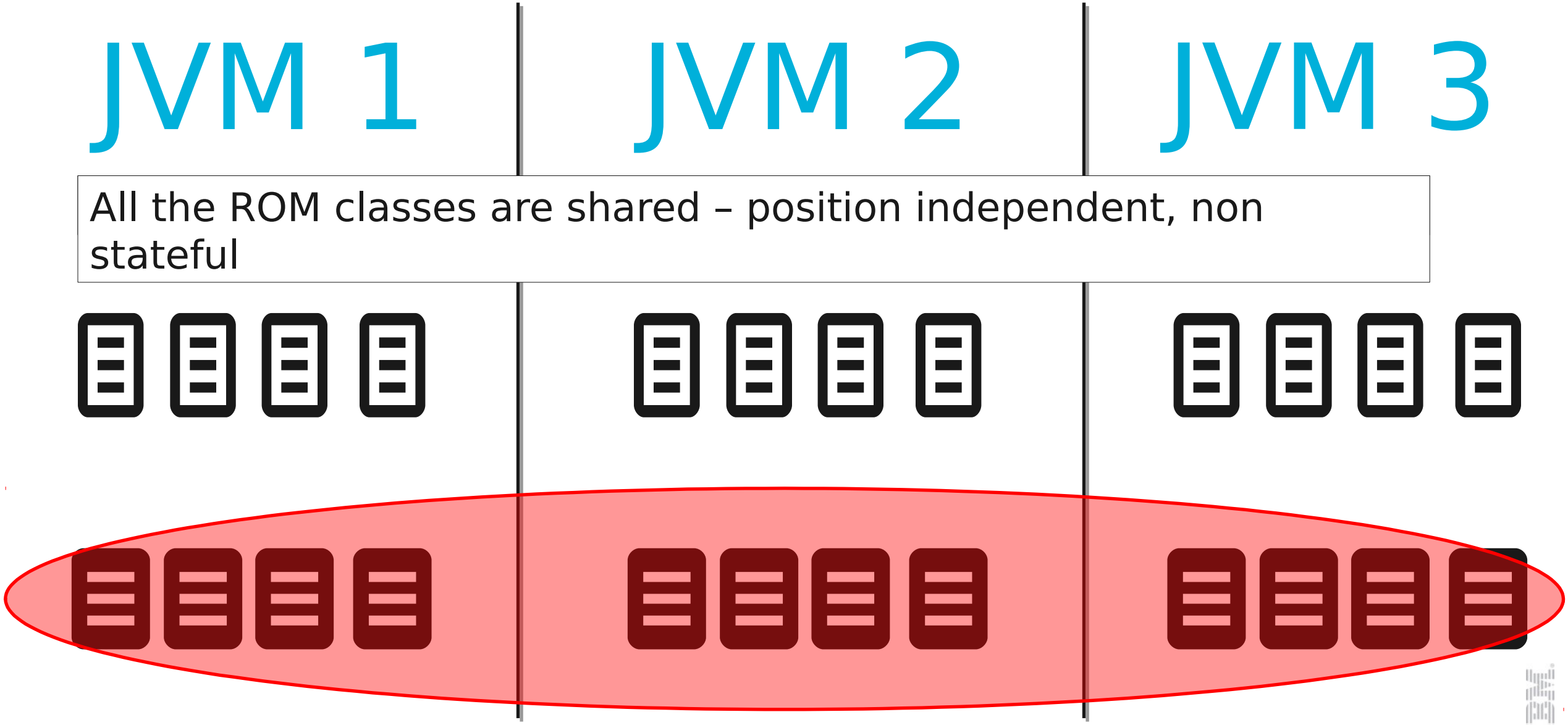
ShareClasses: ROM pays off

JVM 1	JVM 2	JVM 3

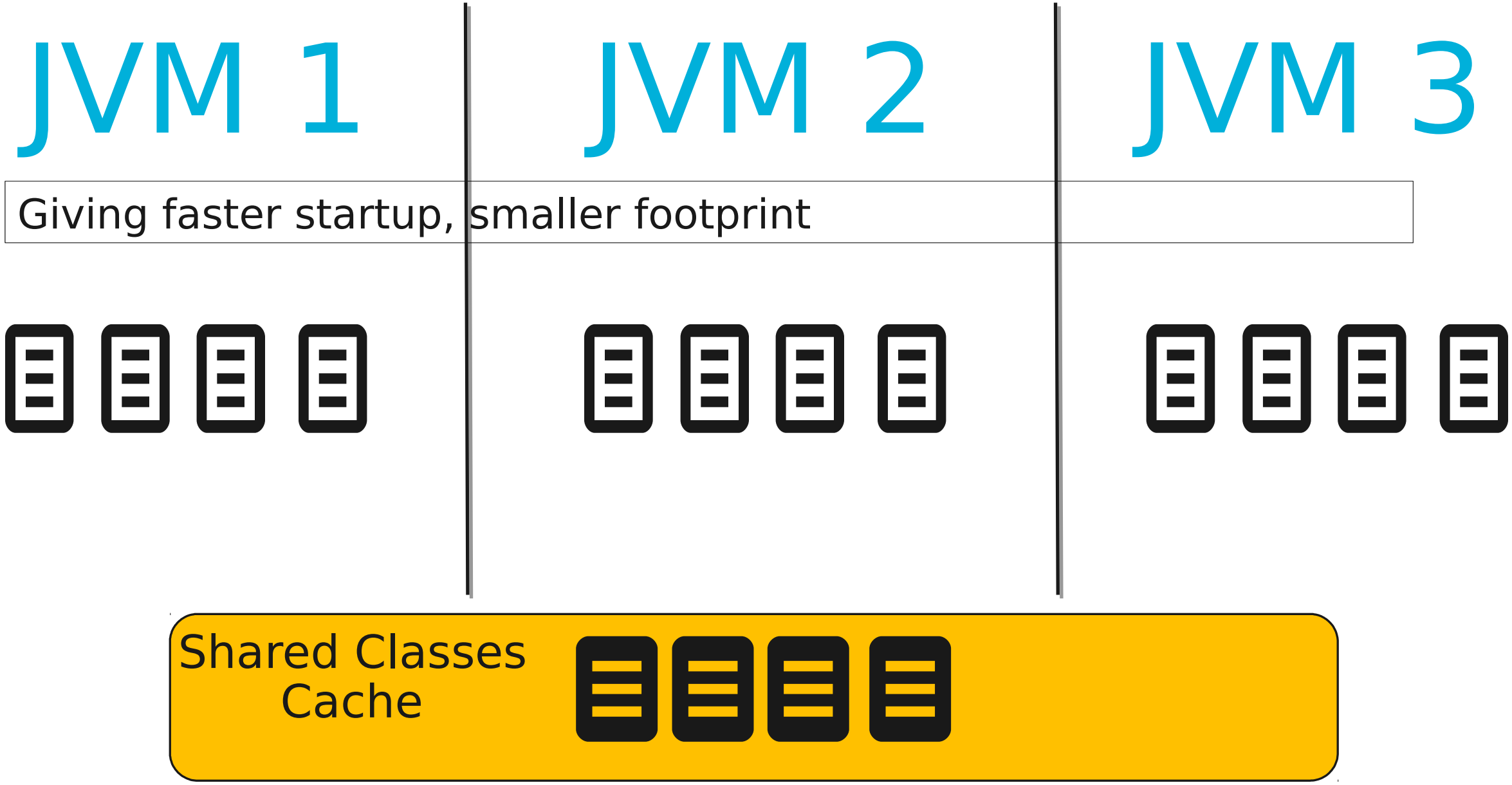
Three JVMs running the same code – on the same machine



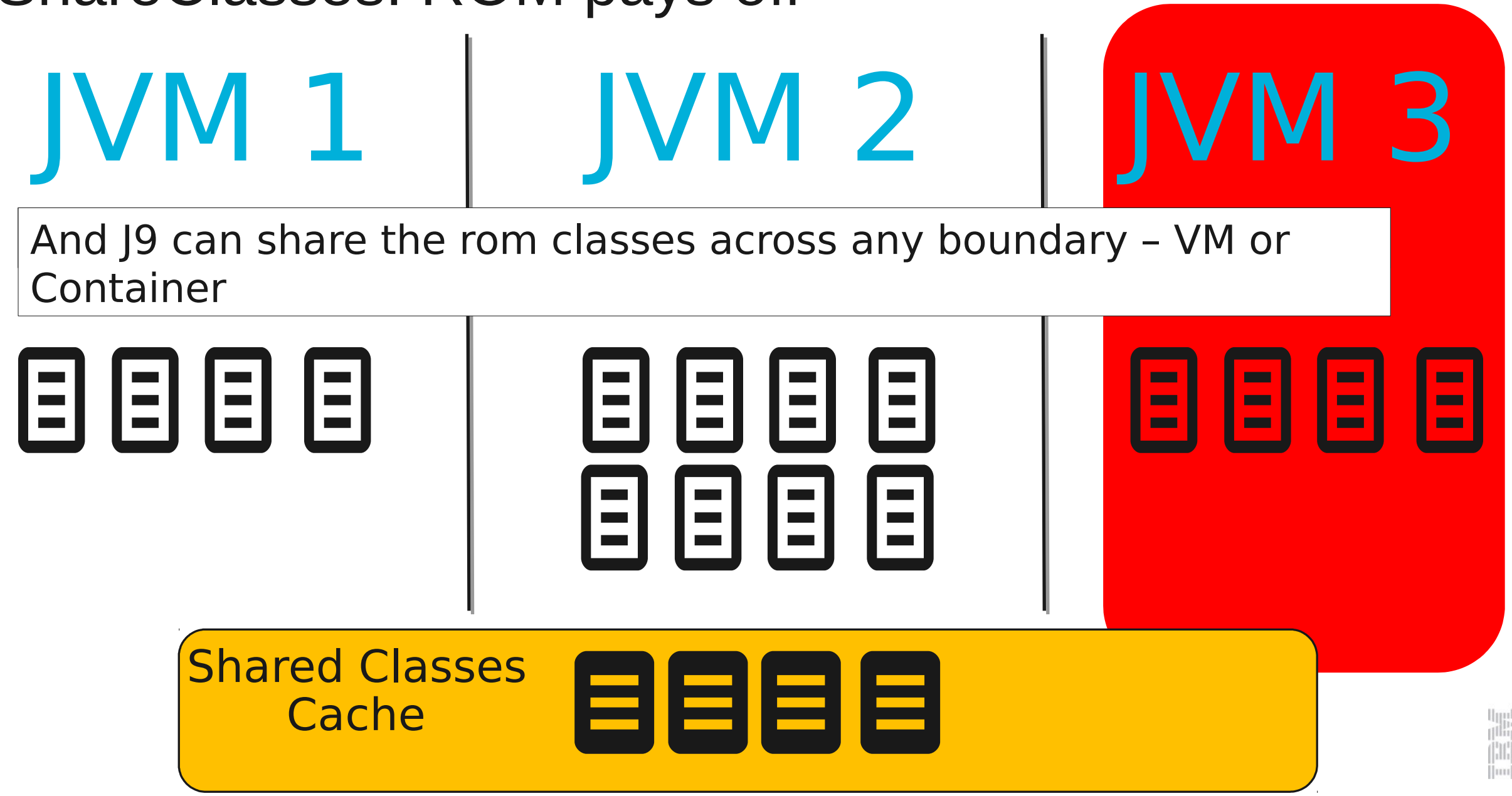
ShareClasses: ROM pays off



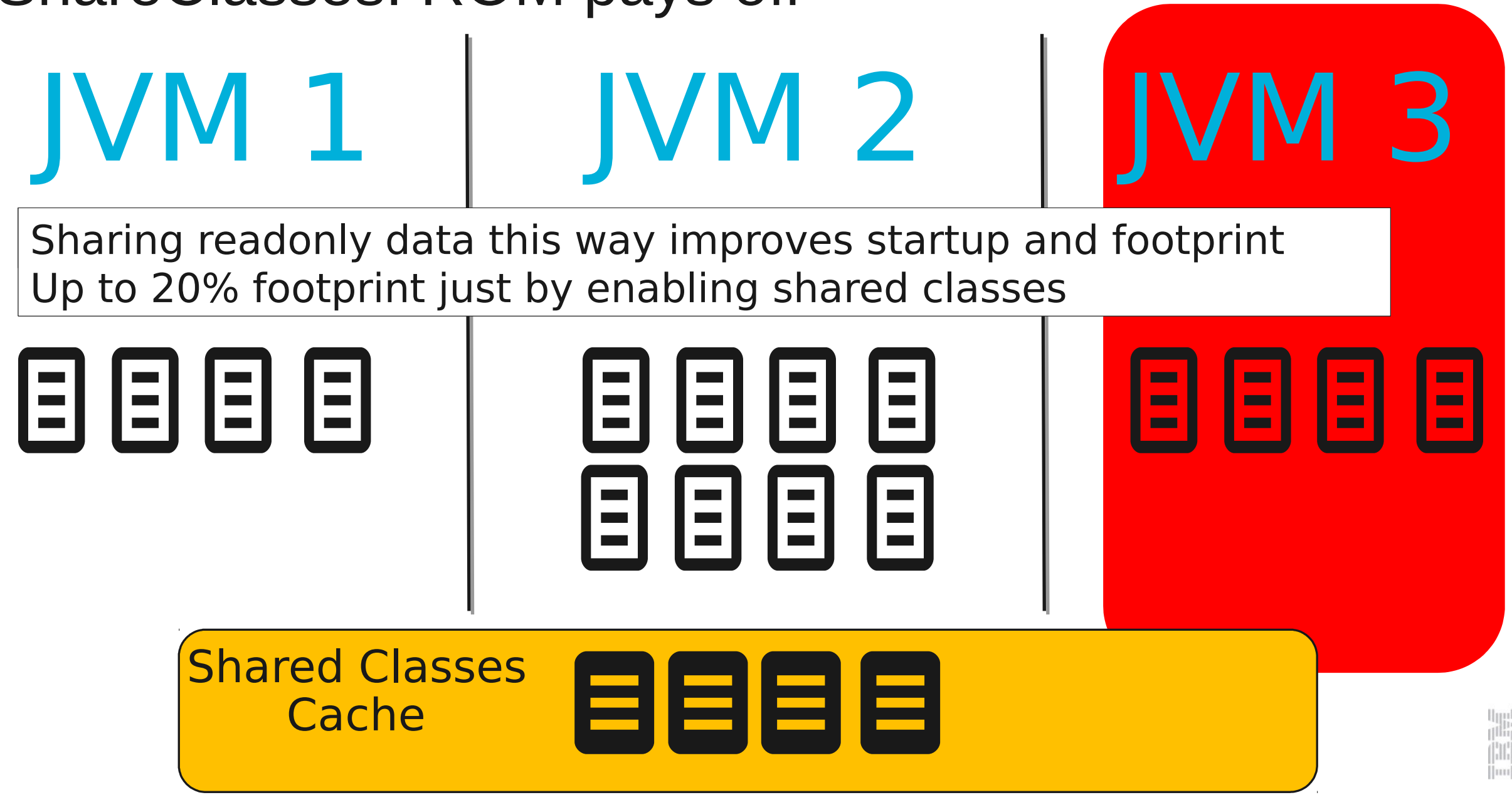
ShareClasses: ROM pays off



ShareClasses: ROM pays off



ShareClasses: ROM pays off



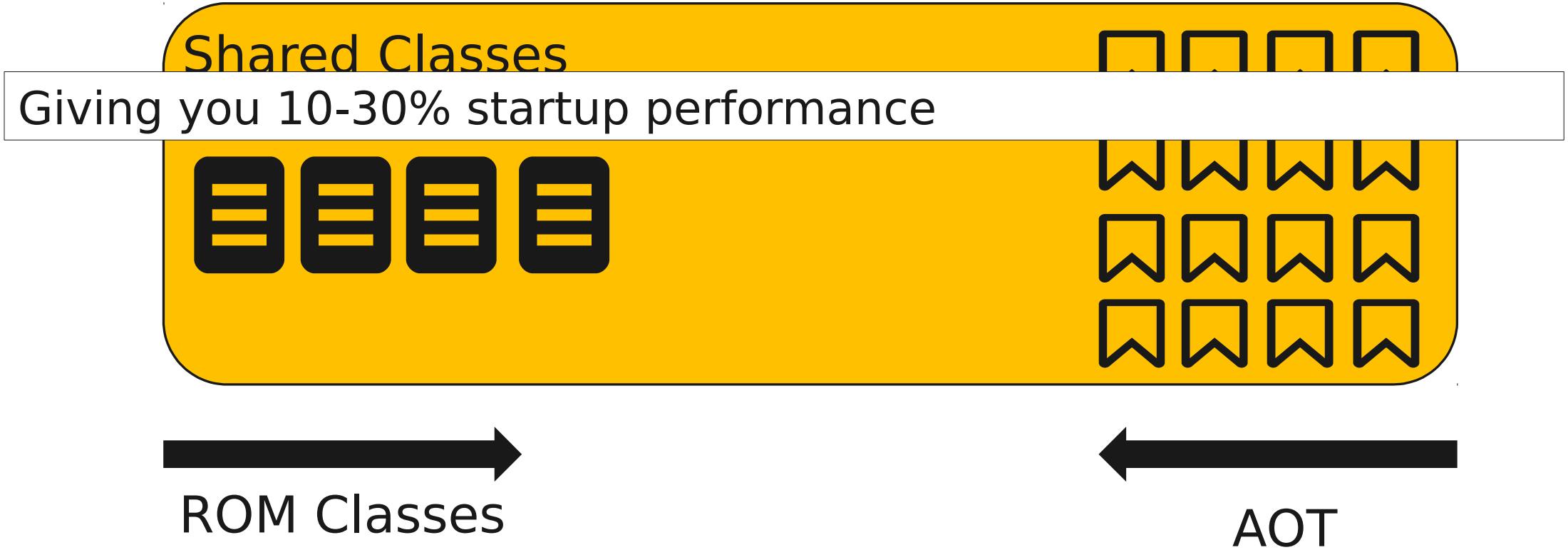
“And J9 can share JITed code too

“Dynamic” AOT through ShareClasses


```
$ java –Xshareclasses ...
```

“And J9 can share JITed code too

“Dynamic” AOT through ShareClasses




```
$ java –Xshareclasses ...
```

ShareClasses and AOT

- Distinction between ‘cold’ and ‘warm’ runs
- Dynamic AOT compilation
  - Relocatable format
  - AOT loads are ~100 times faster than JIT compilations
  - More generic code slightly less optimized
    - Generate AOT code only during start-up
    - Recompilation helps bridge the gap

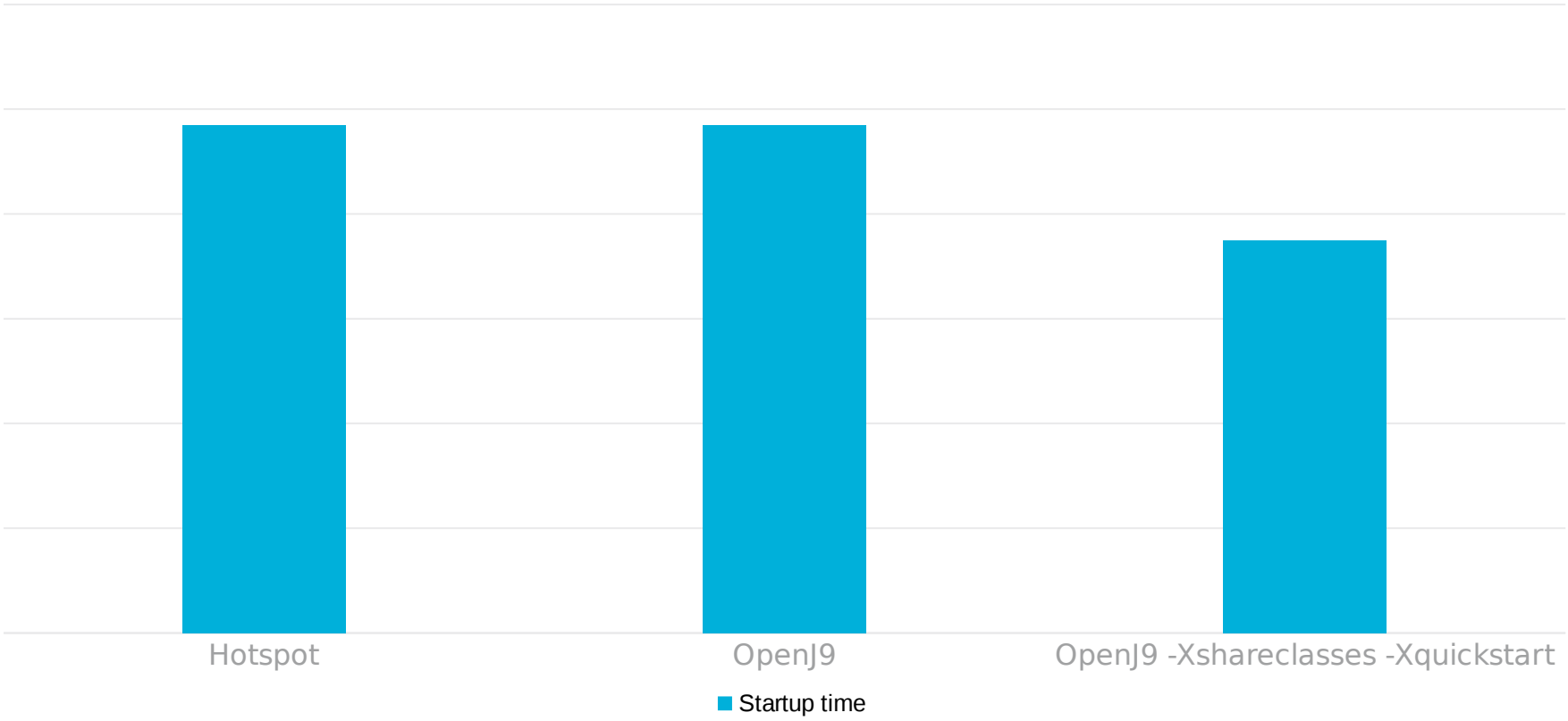
More tuning options

- -Xquickstart
  - Designed for the fastest start-up
  - Ideal for short-lived tasks
  - May limit peak throughput
- -Xtune:virtualized
  - Tuning for containers
  - Enables VM idle management
  - Improves start-up and ramp-up. Trade-off of small throughput loss

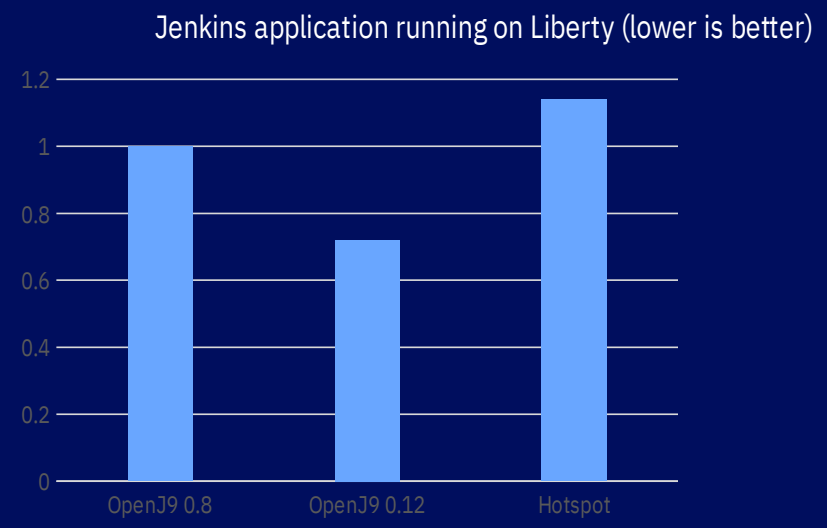

Want to see
what that all
means for Java?

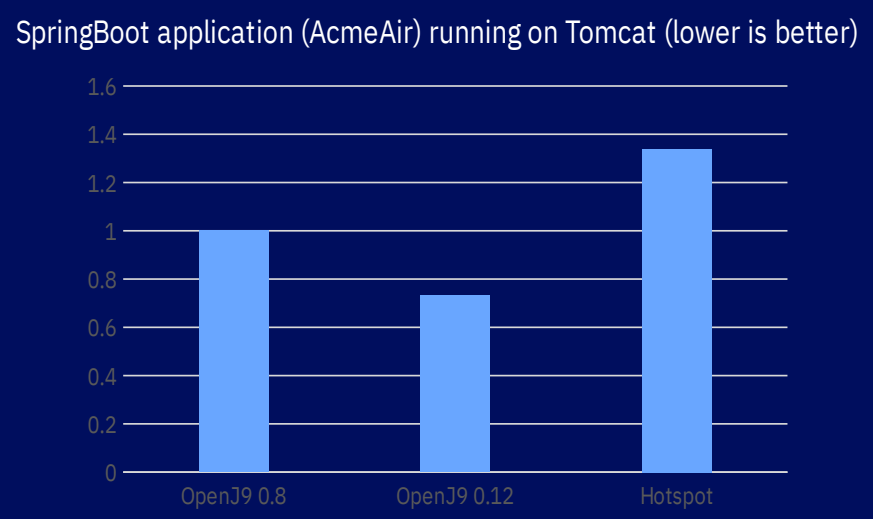
Results

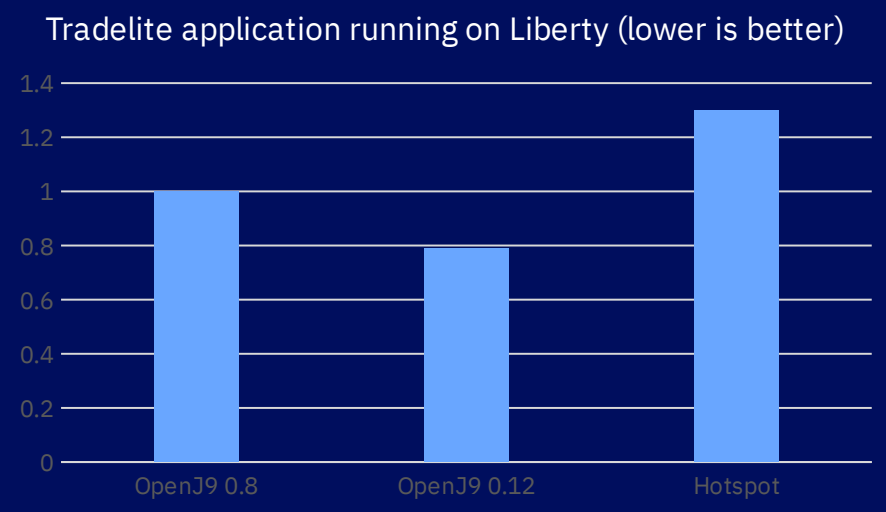
Startup time is ~30% faster with
OpenJ9 –Xshareclasses -Xquickstart

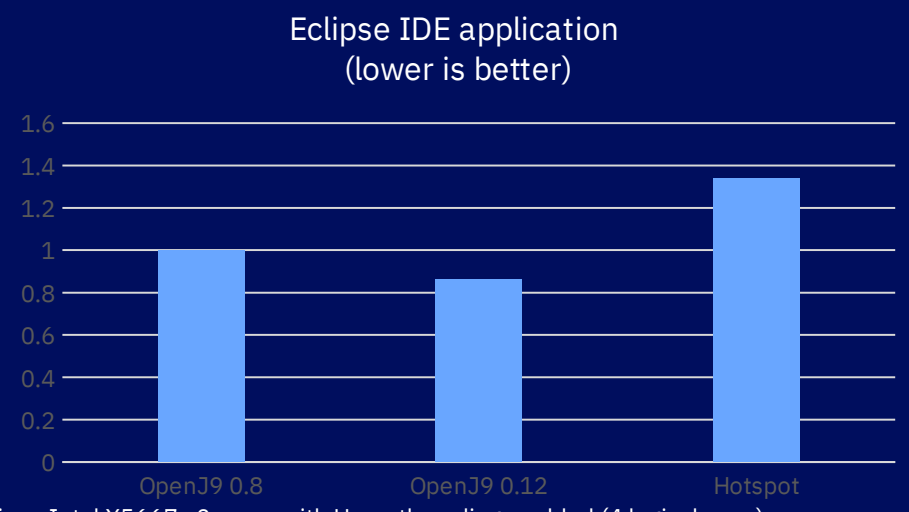


Java8 startup time comparison





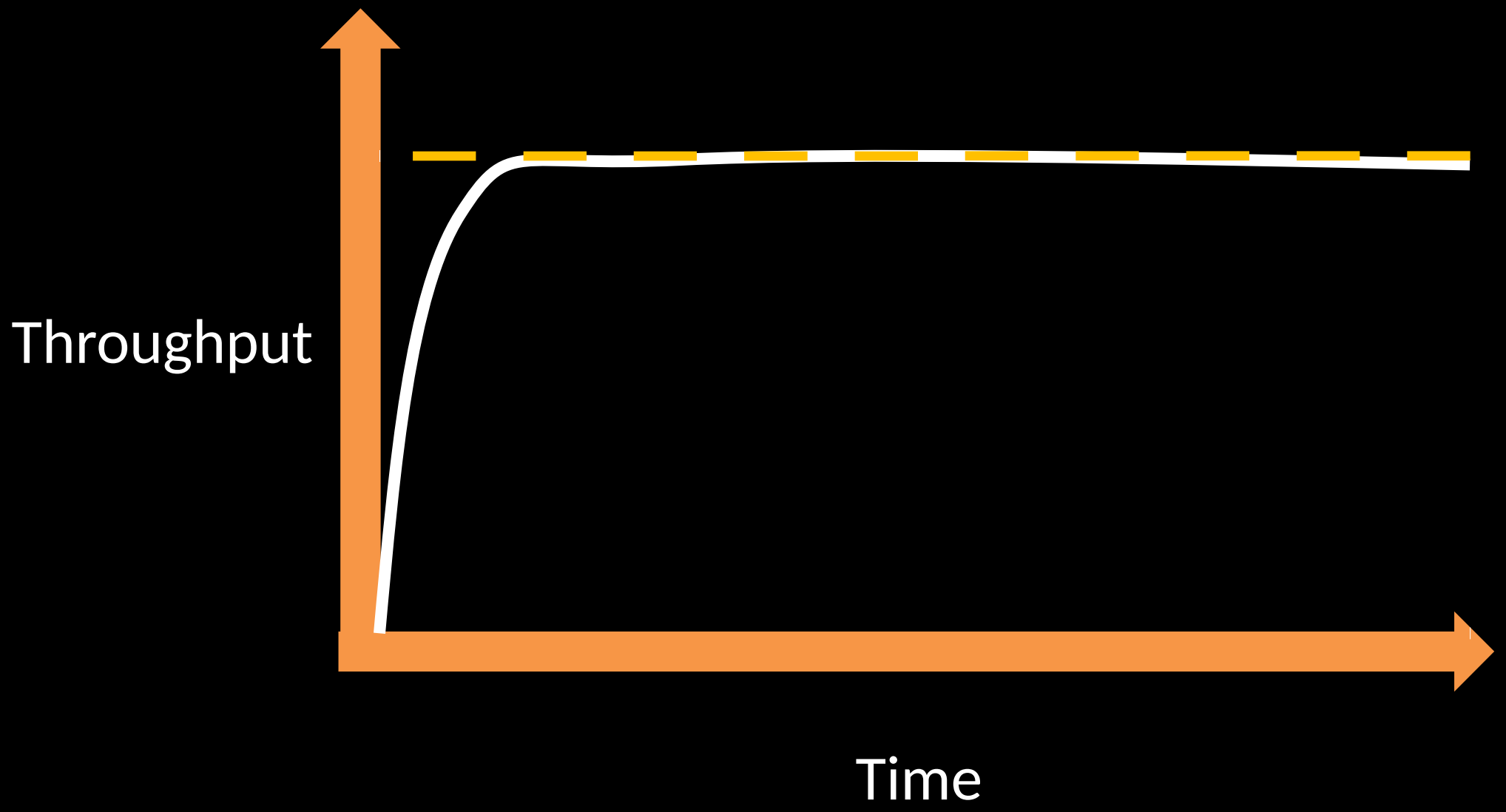




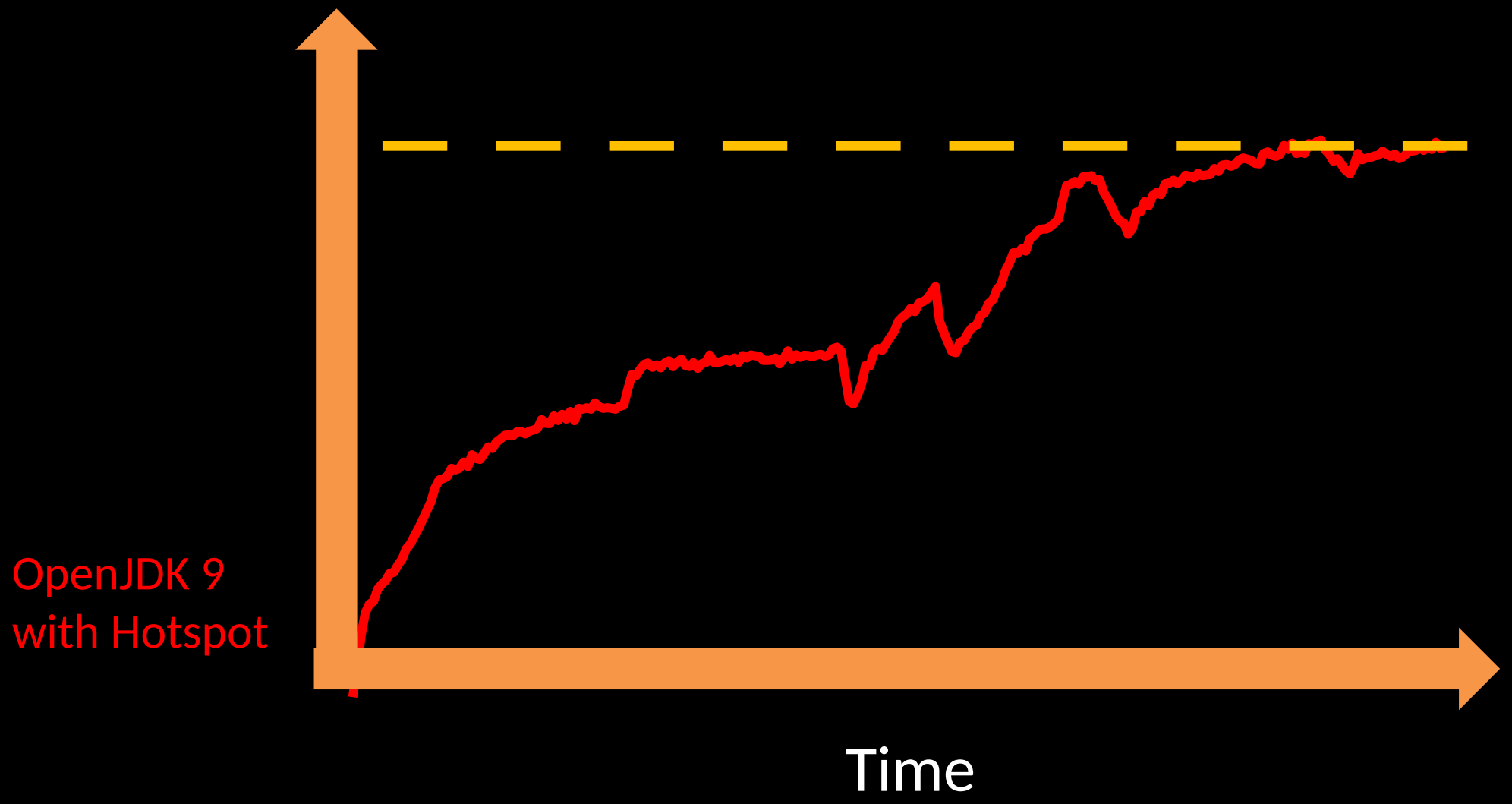
System info : Linux Intel X5667 : 2 cores with Hyperthreading enabled (4 logical cpus)

OpenJ9 improved application startup time on average by 15% in the past year

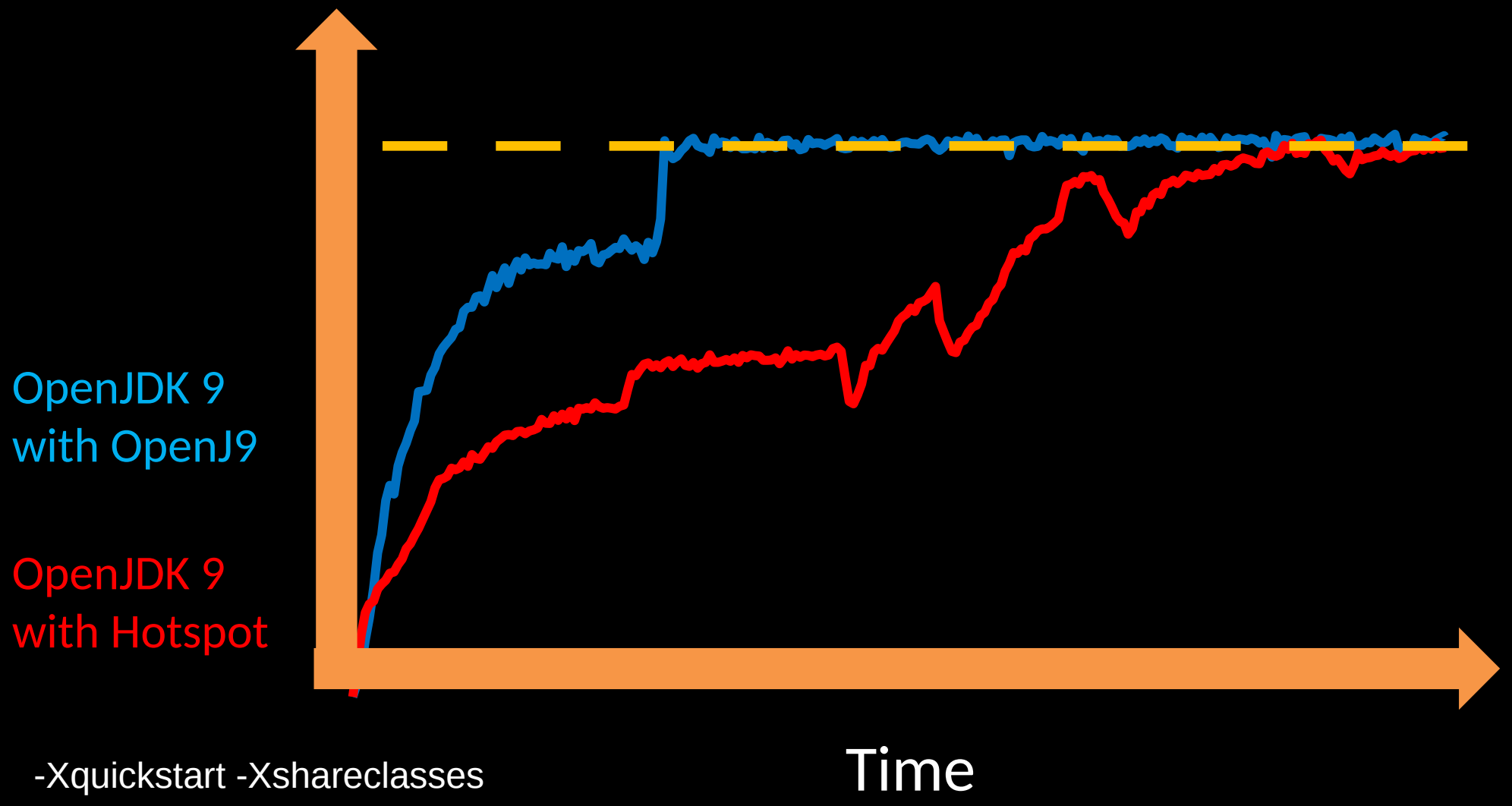
More like this please



Real data

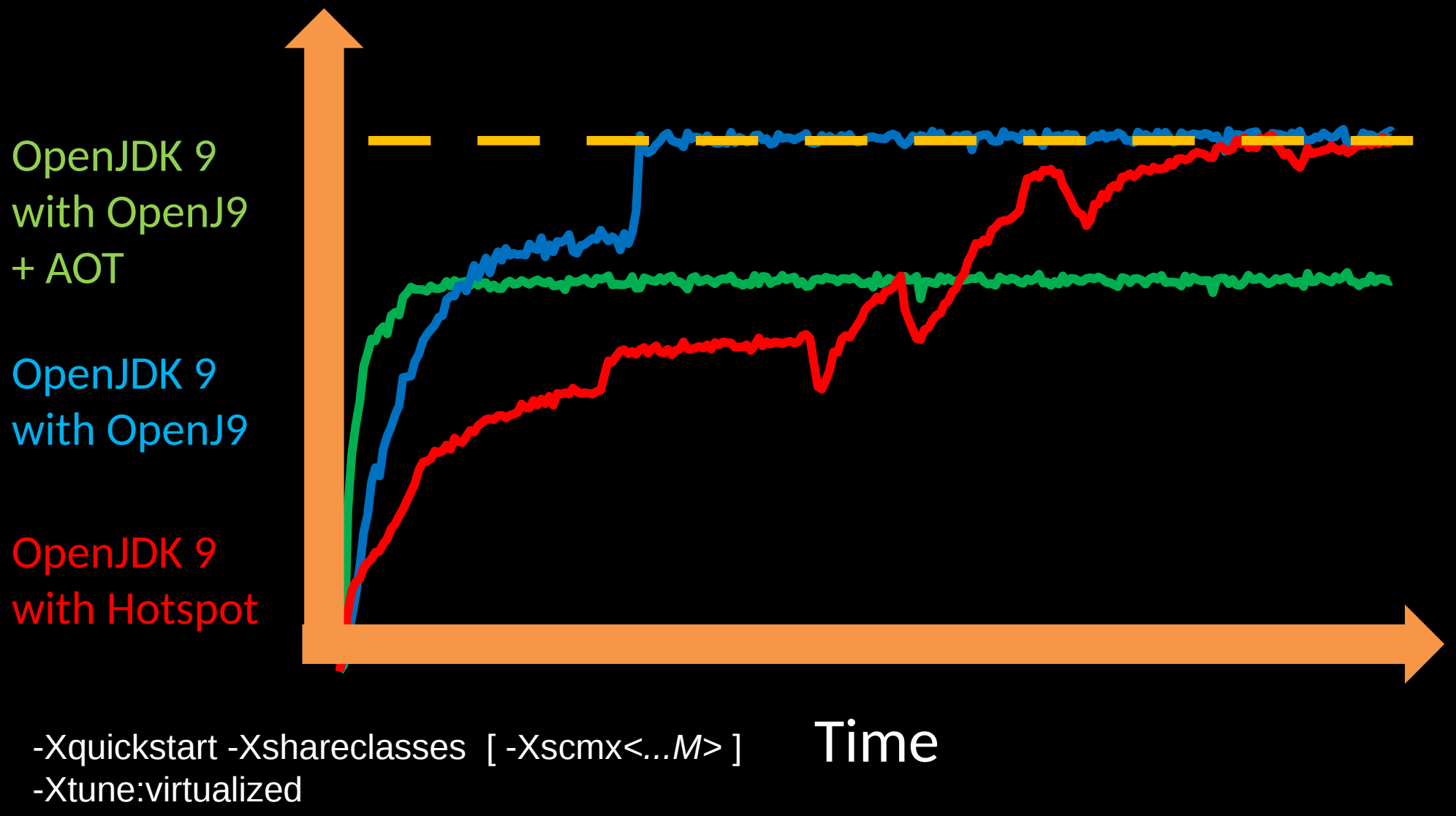


Real data

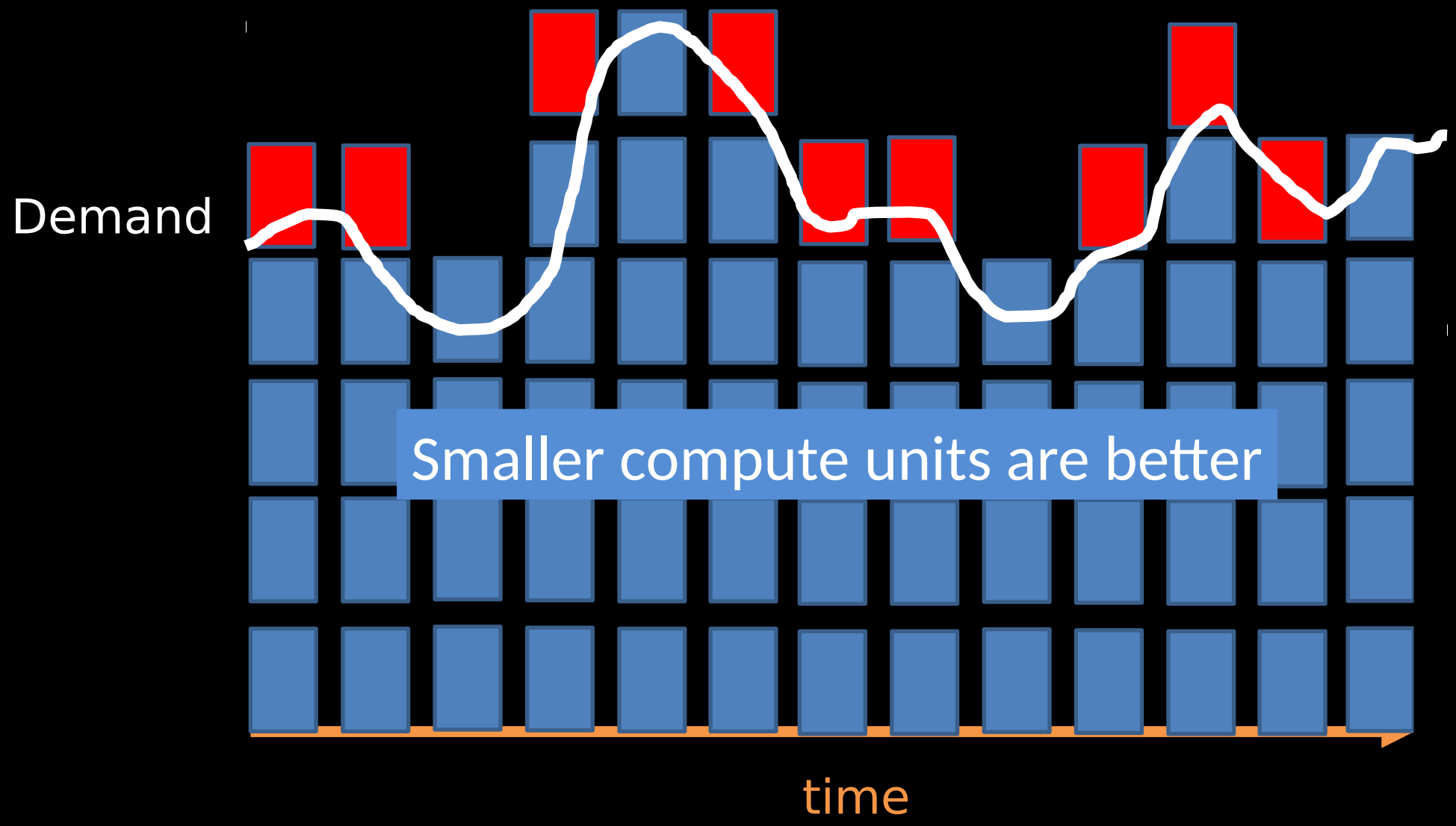


-Xquickstart -Xshareclasses

Real data



-Xquickstart -Xshareclasses [-Xscmx<*...M*>]
-Xtune:virtualized


Demand
Smaller compute units are better
time

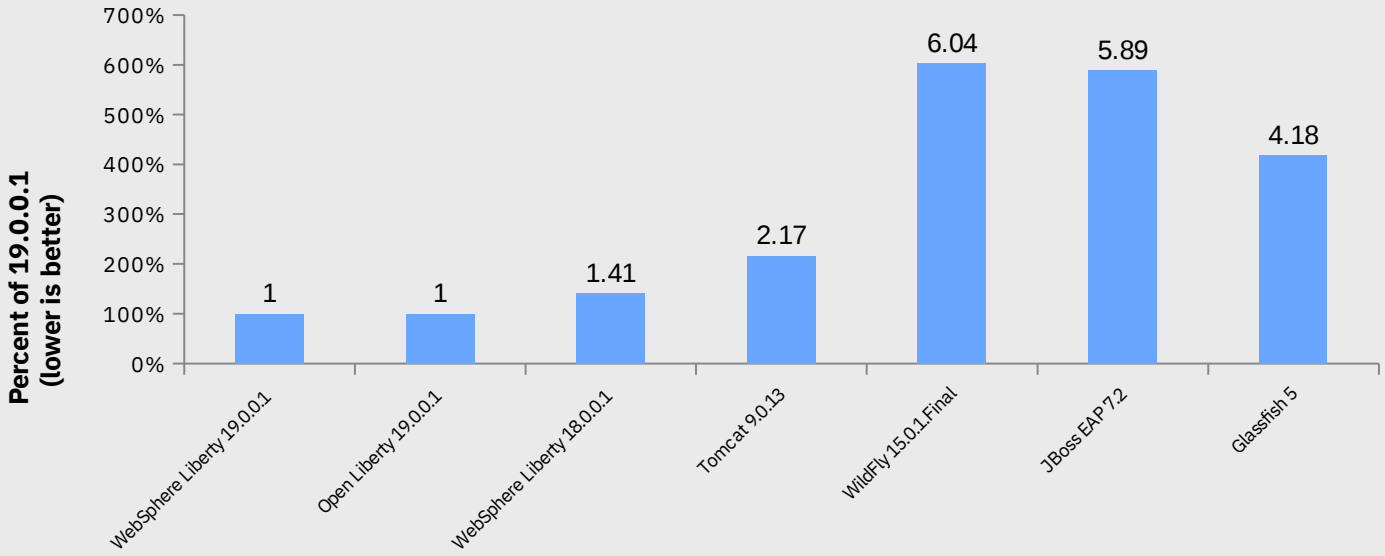
Startup and Footprint (Jenkins)

- WAS 19.0.0.1 Liberty Server startup is much faster than most other Lightweight App Servers
  - 19.0.0.1 is 117% faster to start up than any other lightweight app server
- WAS 19.0.0.1 Liberty Server Memory Footprint is much smaller than other Lightweight App Servers

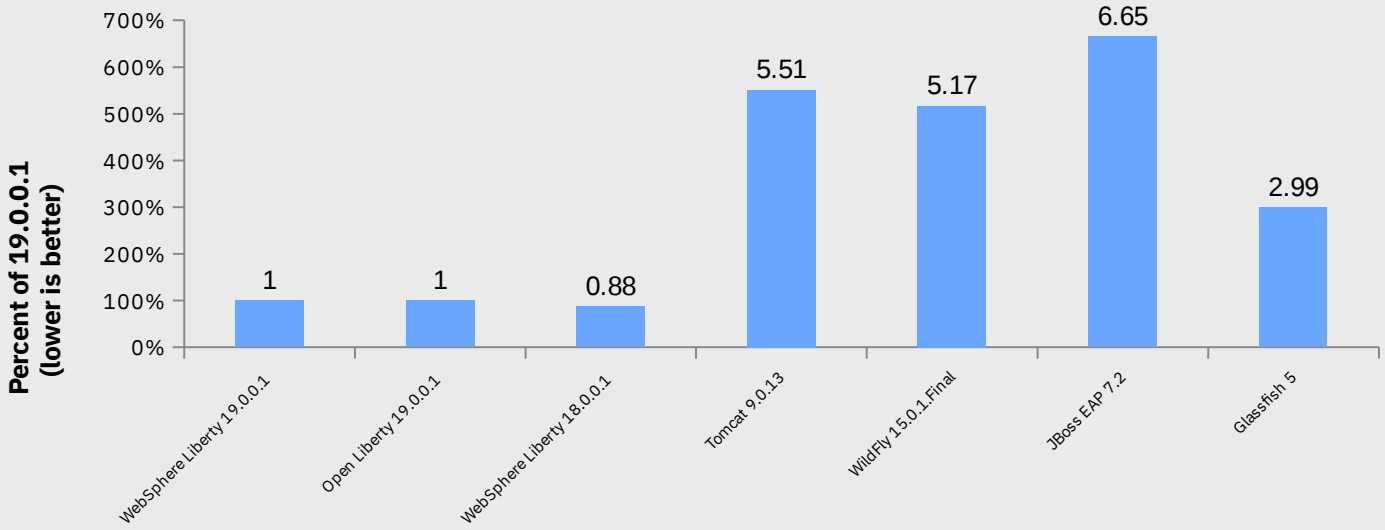
System Configuration:

SUT: LinTel – SLES 11.4, Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz, 4 physical cores, 64GB RAM.
Oracle JDK 8 u201 is used for non-IBM app servers & IBM JDK 8 SR5 FP30 is used for WebSphere Liberty 19.0.0.1, and IBM JDK 8 SR5 FP10 for 18.0.0.1

Startup Time Comparison of lightweight Servers - Jenkins app (lower is better)



End Footprint (RSS) Comparison of lightweight Servers - Jenkins app (lower is better)



Don’t just take my word for it

OpenJ9

Follow

Mike Milinkovich
@mmilinkov

Eclipse @openj9 is “mindblowlingly good” good for Docker and microservice use cases with Java. Check it out!

Java and Docker, the limitations
Mismatch in virtualizationThe combination of Java and Docker isn’t a match made in heaven, initially it was far from it. For

Roy van Rijn
@royvanrijn

Follow

Oh wow, switching from OpenJDK 9 to #Eclipse OpenJ9 is impressive, everything worked out of the box (just two changes in my Dockerfile) and memory usage went from 300MiB to 130MiB, the Spring Boot service is now almost 'micro' 😍👍🏼

Ariya Hidayat @AriyaHidayat

Whoa, seems that @OpenJ9 is quite promising: using much less memory than the official Oracle HotSpot JVM! twitter.com/royvanrijn/sta…

Roy van Rijn @royvanrijn
As promised: a write-up about @…
with a simple CP…

Parameswaran Selvam @ParamSelvam

JEP to release unused memory back to OS openjdk.java.net/jeps/8204089 in draft which is currently available with @openj9 developer.ibm.com/javasdk/2017/0… …org/openj9/oj9_res…

Nick Ebbitt @nickebbitt · 20 Aug 2018
Played around with some of the @adoptopenjdk artifacts this evening, specifically the Dockerfiles they produce for @OpenJDK Hotspot and @openj9 .github.com/nickebbitt/jav…

nickebbitt/java-in-docker
Comparing the options for running Java in docker. Contribute to nickebbitt/java-in-docker development by creating an
github.com

Nick Ebbitt
@nickebbitt

Really interesting to observe that with a very basic @springboot web app the memory footprint of @openj9 is over 100 MB less. Definitely plan to explore this further.
pic.twitter.com/UAjiRgQABK
22:25 - 20 Aug 2018

Mike Milinkovich @mmilinkov
Follow

Eclipse @openj9 is "mindblowlingly good" good for Docker and microservice use cases with Java. Check it out!

Java and Docker, the limitations
Mismatch in virtualizationThe combination of Java and Docker isn't a match made in heaven, initially it was far from it. For starters, th...
royvanrijn.com

Roy va…
@royvanri…
Follow

Oh wow, switching from OpenJDK 9 to #Eclipse OpenJ9 is impressive, everything worked out of the box (just two changes in my Dockerfile) and memory usage went from 300MiB to 130MiB, the Spring Boot service is now almost 'micro' 😍👍🏼

OpenJDK with OpenJ9 Performance Advantages over OpenJDK (Hotspot)

IBM Runtimes for Business includes support for the high performant OpenJ9 Runtime technology

Superior Runtime Characteristics

OpenJDK OpenJ9

66% smaller footprint

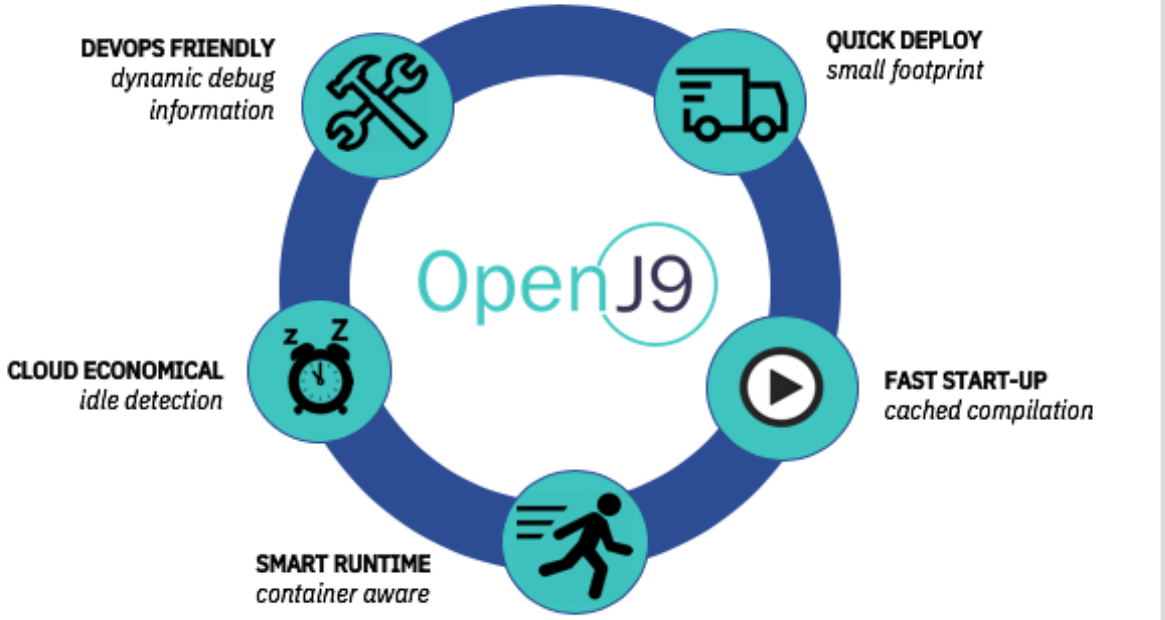
42% faster start-up

3x faster to peak performance in constrained environments

100% throughput performance

https://www.eclipse.org/openj9/oj9_performance.html

Whole Life-Cycle Optimizations



Broad Platform Support

Linux x64 | Linux x64 Large Heap | Windows x32

Windows x64 | Linux s390x | Linux ppc64le | AIX ppc64 | macOS x64

Jan 2019

AdoptOpenJDK and IBM Runtimes for Business: Support for open source Java

Open, multiplatform, production-ready distribution of OpenJDK with Eclipse OpenJ9. Pay only for Java runtime environments where you need enterprise-grade support.

Contact an IBM representative

View pricing and buy now

Get a supported Java runtime for your desktop
Or just run it for free

adoptopenjdk.net

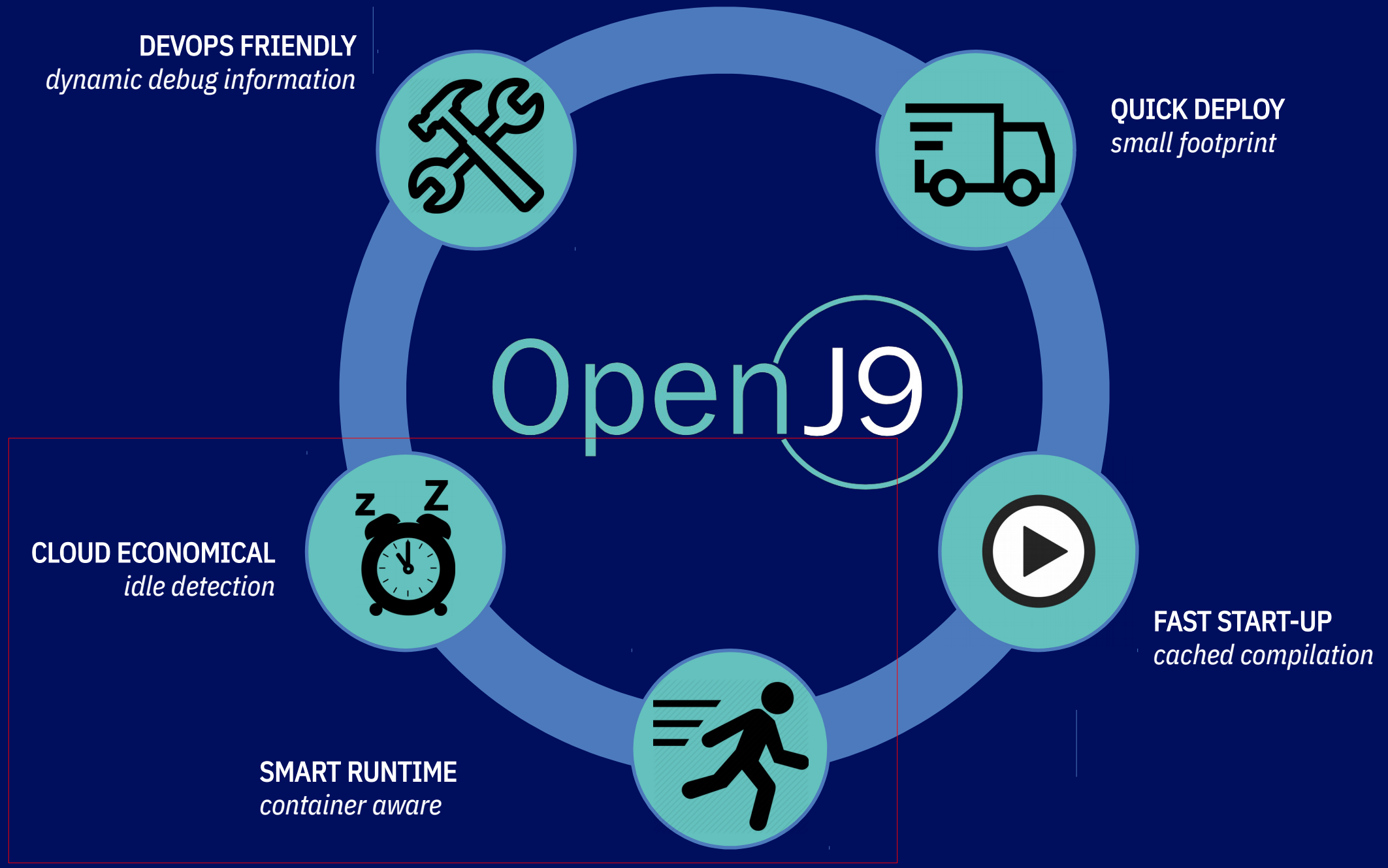
OpenJ9-OpenSSL JCE acceleration on different ciphers

Open source JCE solution uses OpenSSL native code to accelerate JCE ciphers

OpenJ9-OpenSSL JCE improves crypto performance significantly on Linux X86

Primitive (1024 bytes payload)	Speedup : OpenJ9-OpenSSL vs OpenJ9
aes-128-cbc-encrypt	4.5X
aes-128-gcm-encrypt	15X
aes-128-cbc-decrypt	15X
aes-128-gcm-decrypt	13.5X
sha256	2X
rsa	3X

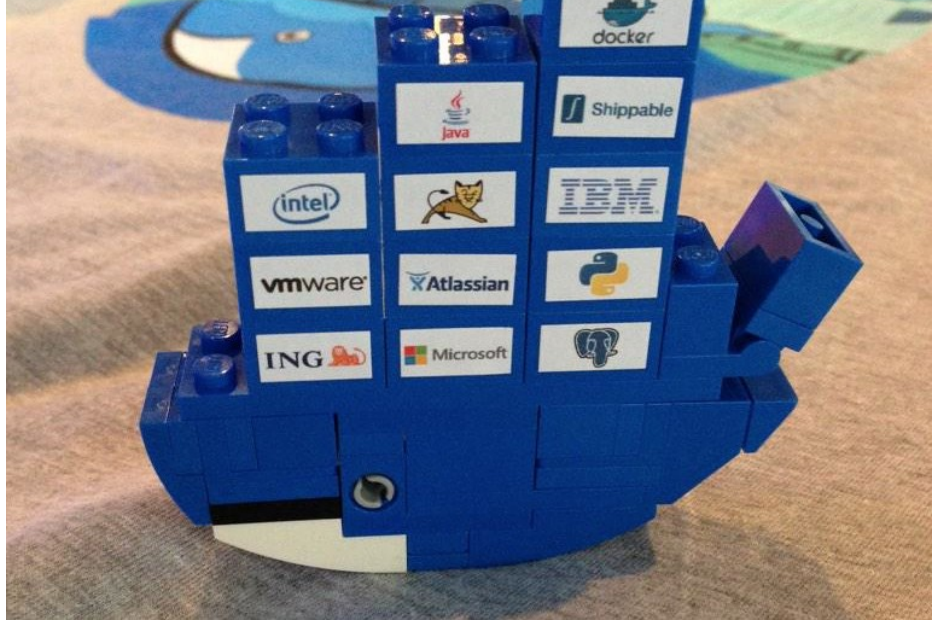
OpenJ9-Ope


DEVOPS FRIENDLY
dynamic debug information
QUICK DEPLOY
small footprint
OpenJ9
CLOUD ECONOMICAL
idle detection
FAST START-UP
cached compilation
SMART RUNTIME
container aware

We’re not alone running on a ‘cloud’ infrastructure…

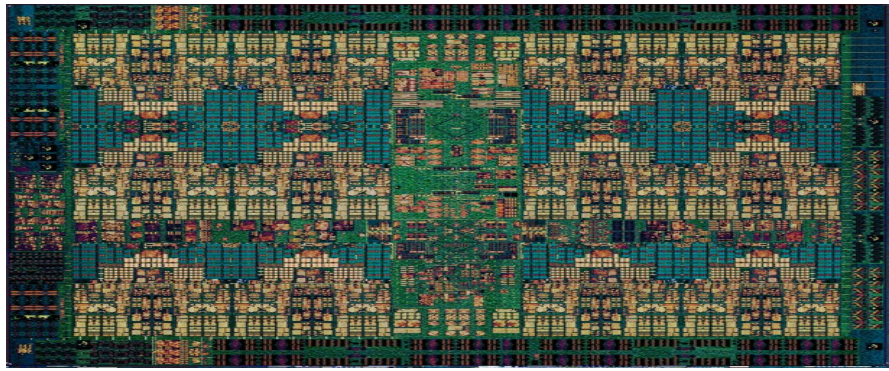
Consuming resources, if not productive, costs us (and others)

https://



`-XX:+UseContainerSupport`

Need to let the JVM know its not in ‘walled garden’ mode!

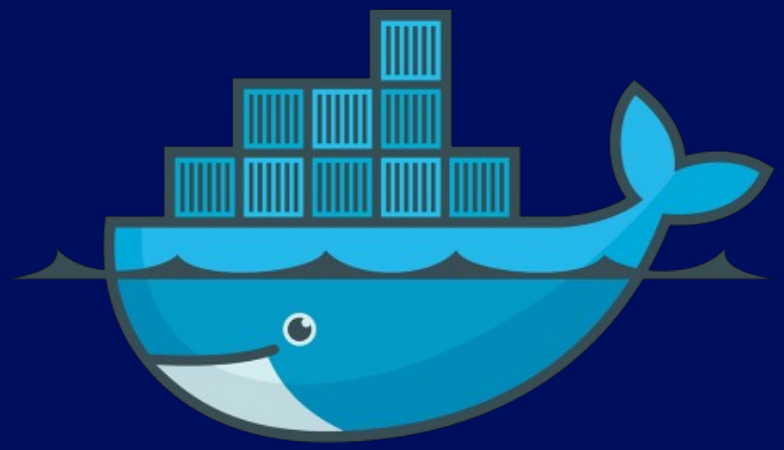


Attentive to dynamic number of physical cores
`Runtime.availableProcessors()` based on cgroup limits

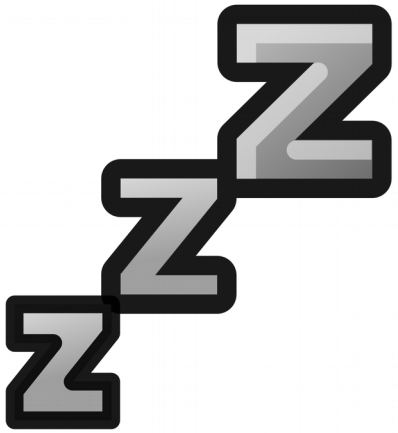
`-XX:InitialRAMPercentage` / `-XX:MaxRAMPercentage`

(Instead of -Xms / -Xmx)

OpenJ9 scales based on container limits



- OpenJ9 is container-aware wrt CPUs as well as memory limits
- JVM tailors resource usage as per constraints imposed by orchestrators, e.g. Kubernetes
- Default parameters for GC and JIT are now tuned if OpenJ9 is running in a container
- OpenJ9 autotunes itself even as Kubernetes dynamically modifies resource constraints



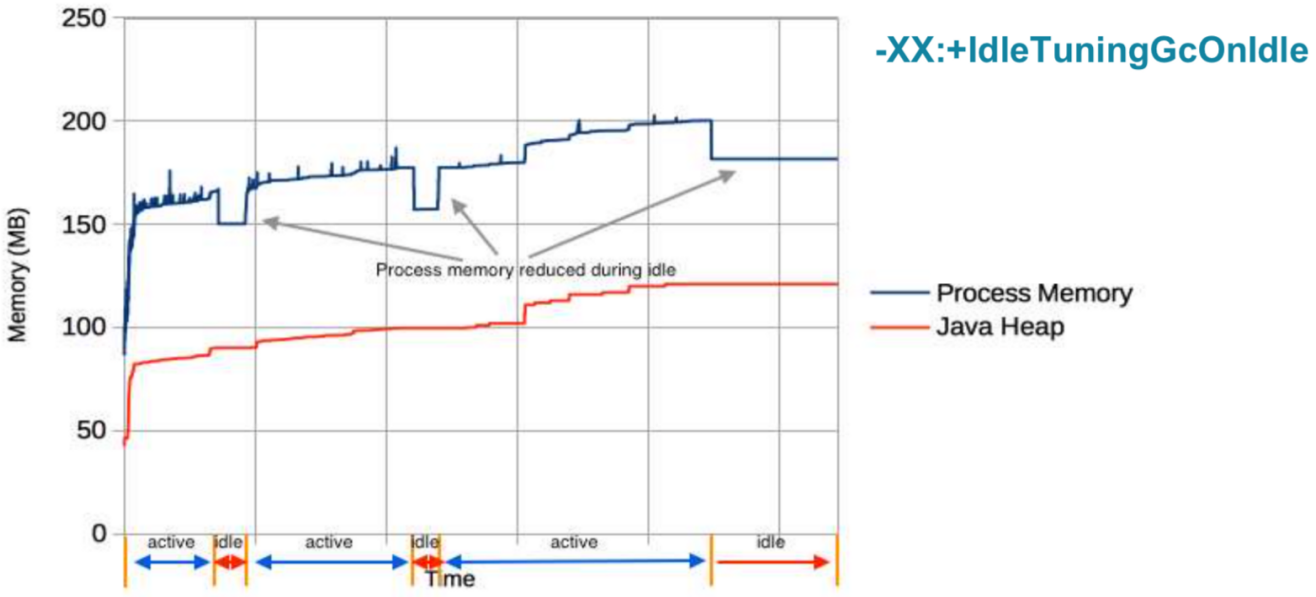
MicroServices & Serverless/FaaS

Don’t use resources unnecessarily

Good citizen

Be prepared!

`-XX:+IdleTuningGcOnIdle`



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

`-XX:+IdleTuningCompactOnIdle`

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

If any JVM can be said to be at the heart of the enterprise – its this one.

IBM donated J9 to Eclipse because we believe it’s the best way to move Java forward

- It offers a new place to start
- As the future emerges we can see that Java needs to handle new technologies, new hardware.
- Whether GPUs or Neuromorphic Processors or even ultimate prize of Quantum computers: Java must adapt.

- We can’t do it on our own. We have to do it together

OpenJ9

Eclipse OpenJ9
Created Sept 2017

http://www.eclipse.org/openj9
https://github.com/eclipse/openj9

Dual License:
Eclipse Public License v2.0
Apache 2.0

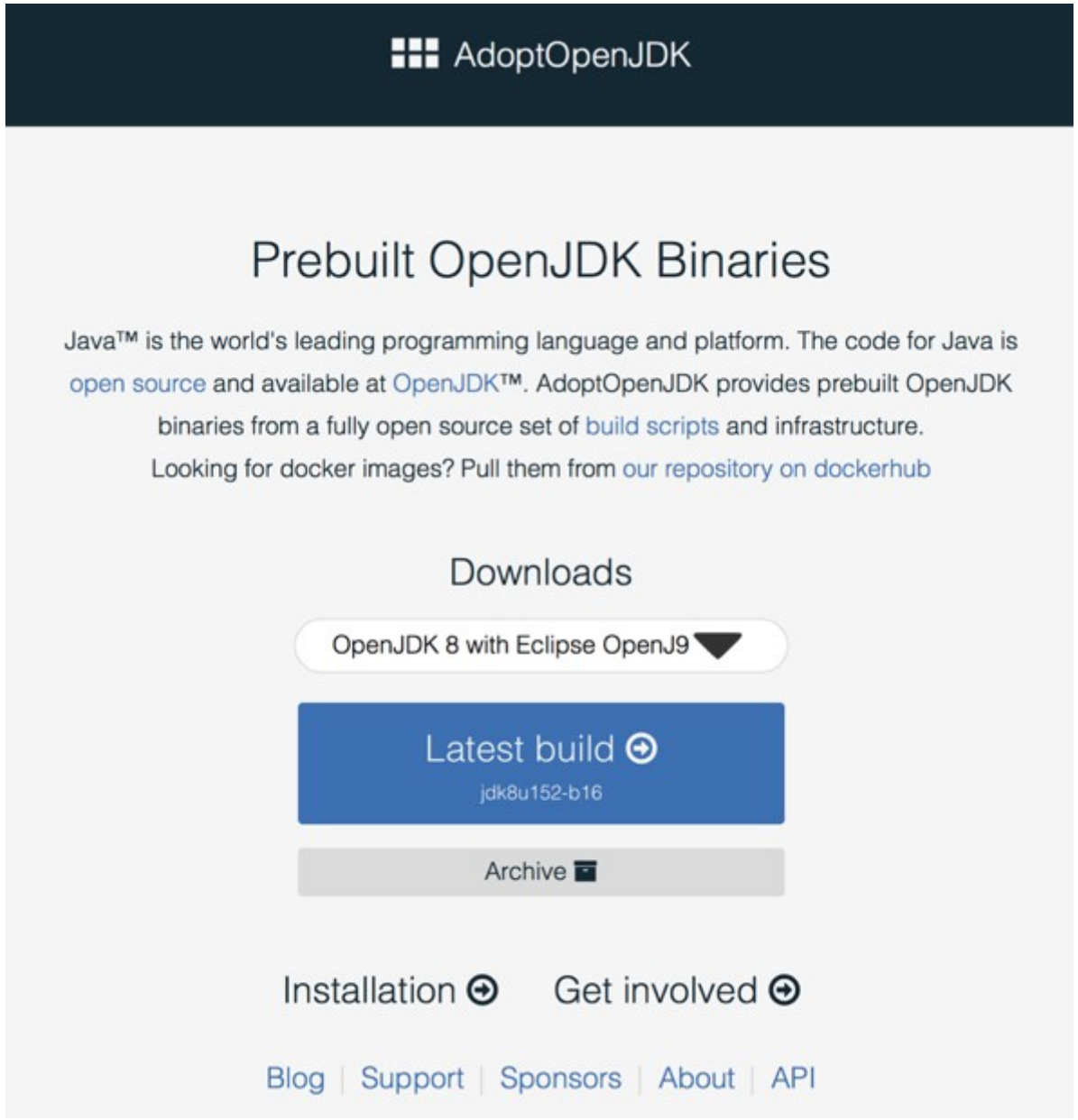
Users and contributors very welcome

https://github.com/eclipse/openj9/blob/master/CONTRIBUTING.md

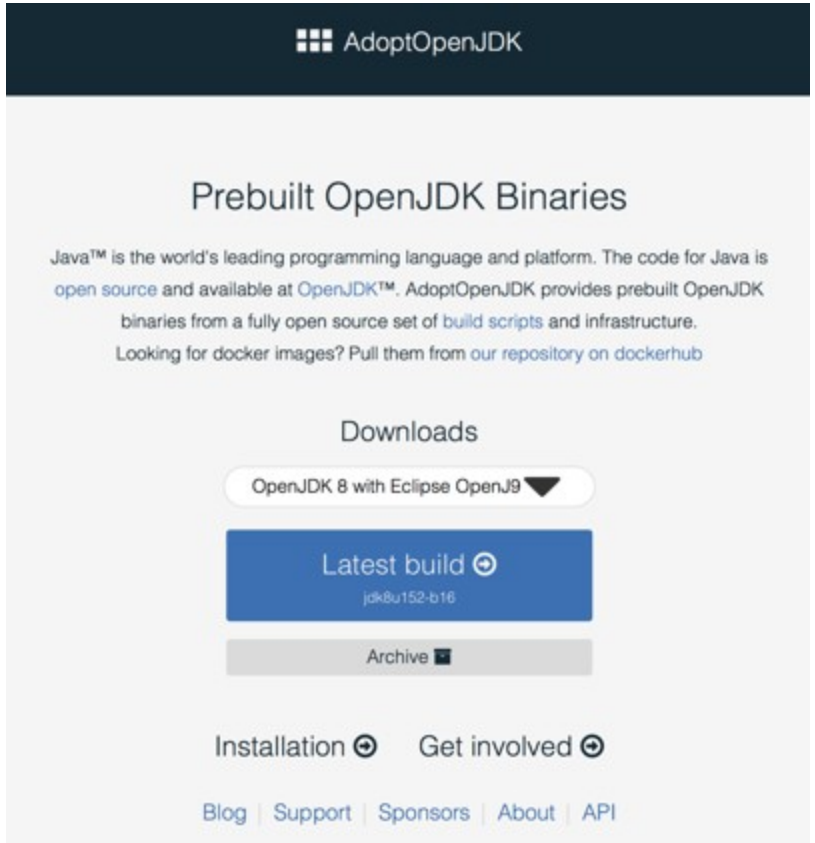
It’s surprisingly frugal
It’s surprising fast

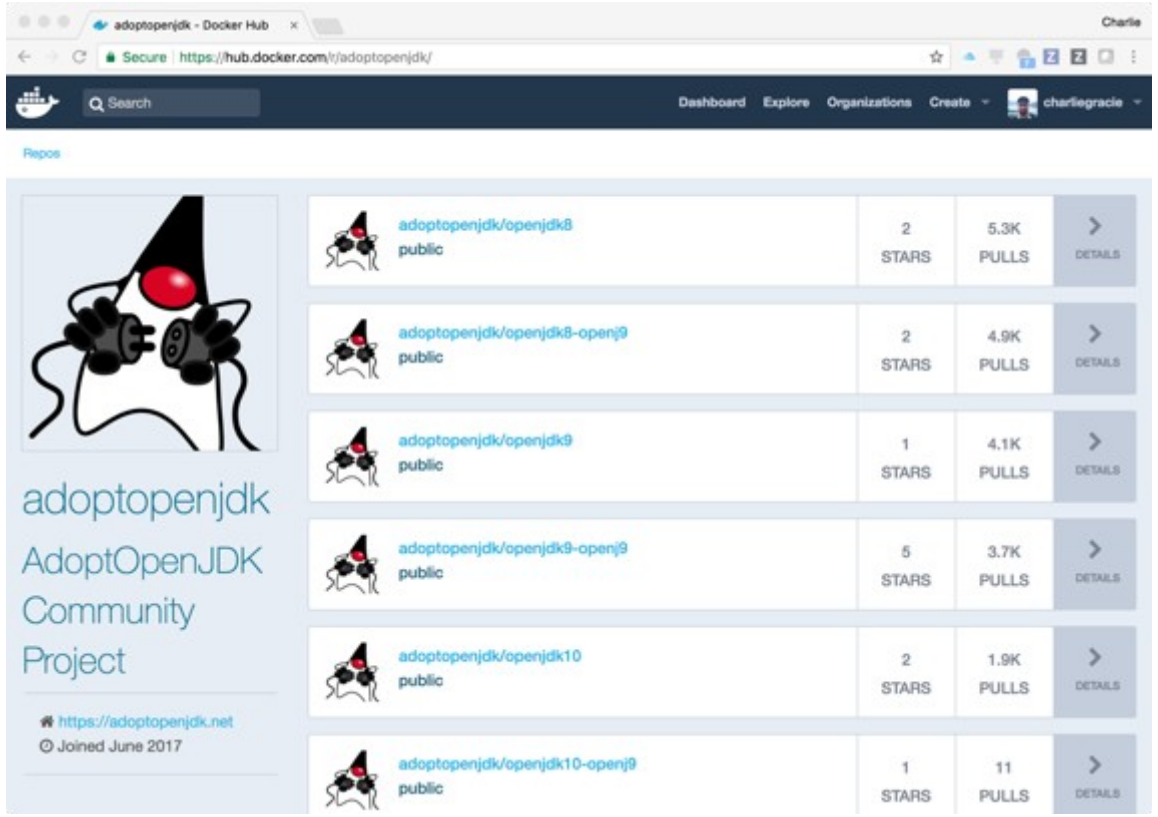
And its available today

OpenJ9



adoptopenjdk.net





Fresh Java - how you like it.

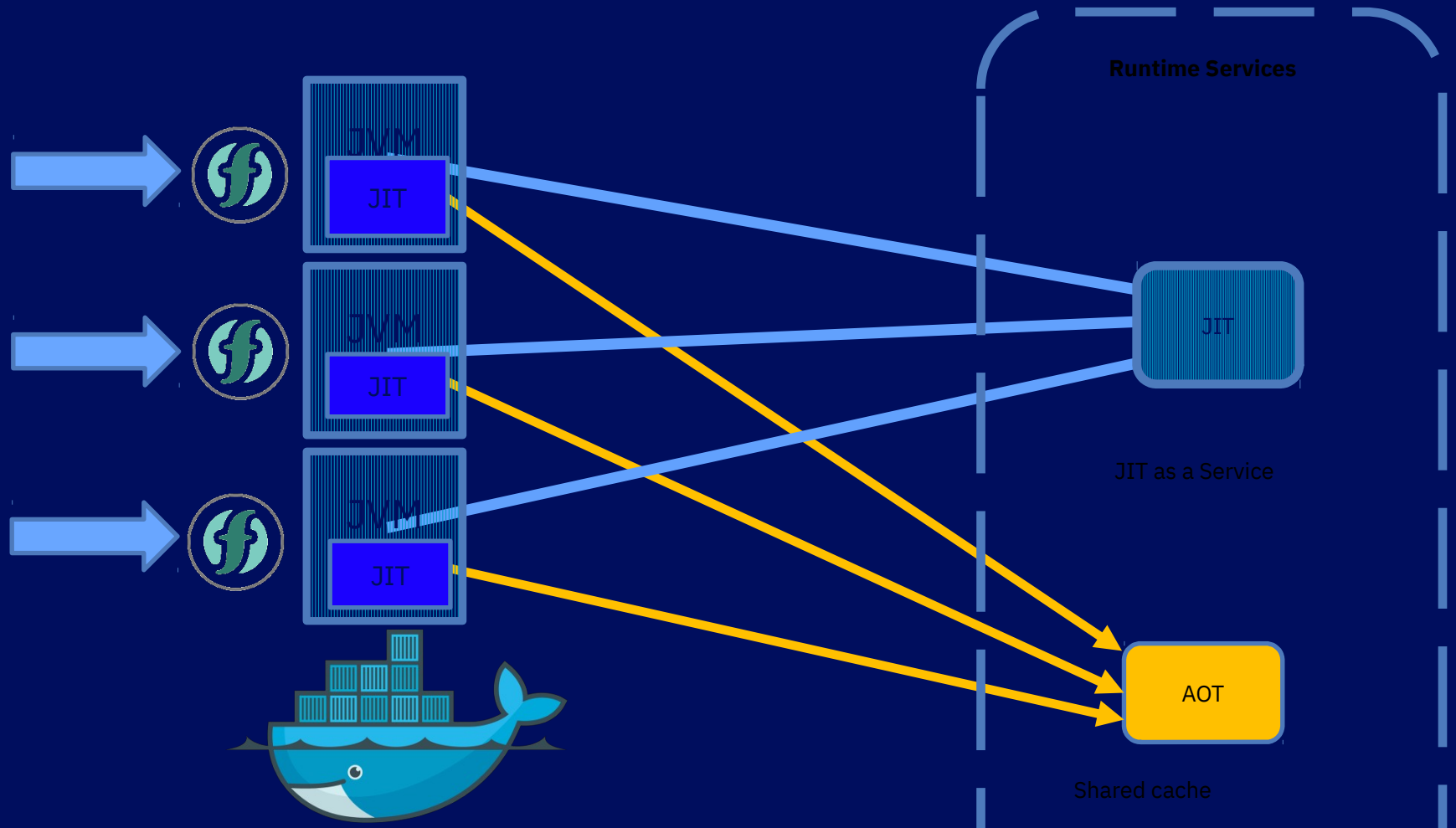
Java 8, 9, 10, 11, ..

IBM contributed J9 to Eclipse because modern Java problems can’t be solved by the few.

We all need to work together to take Java in new directions

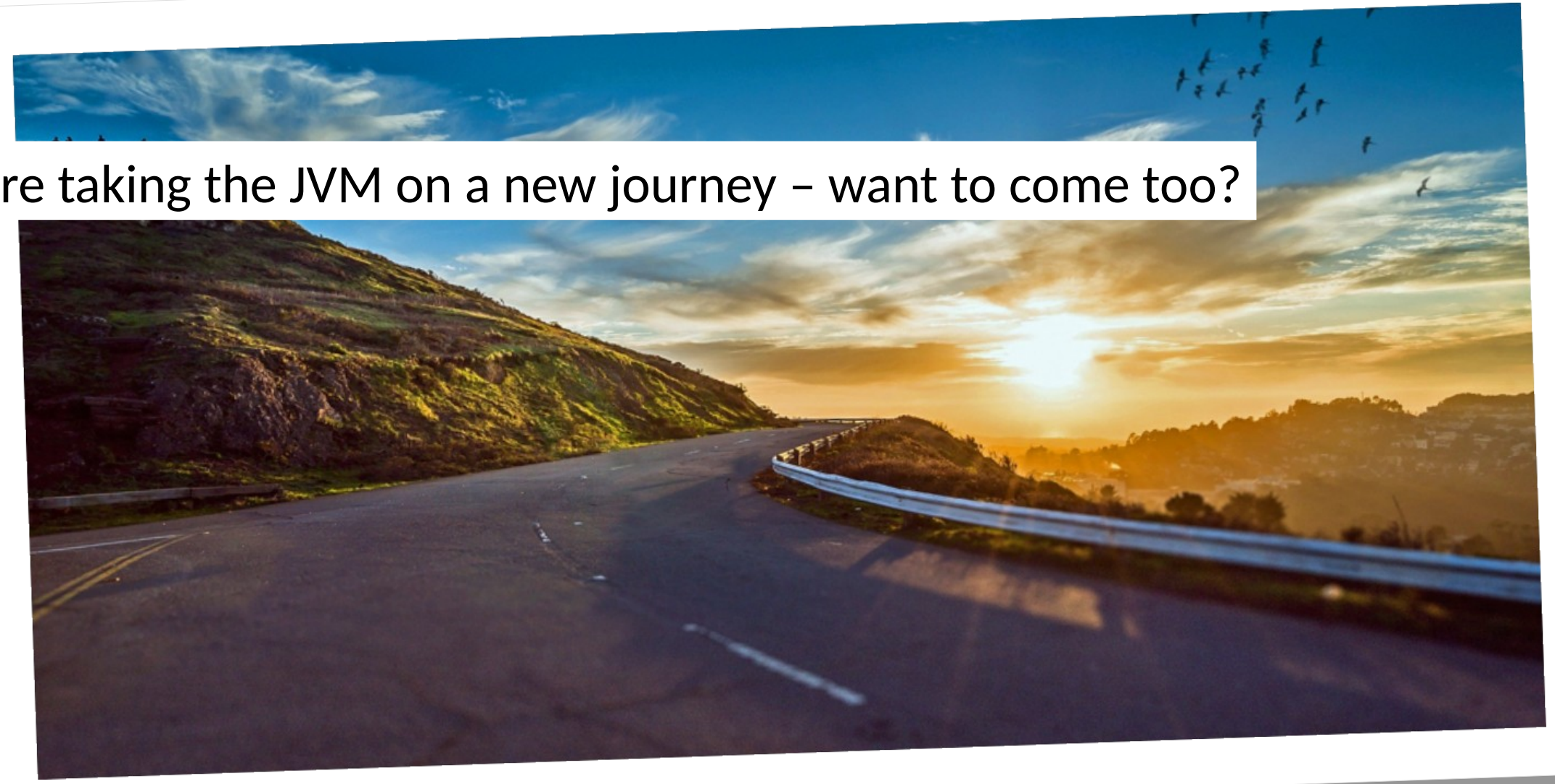
Future

Ahead of Time and Just in Time Compilation



AOT: Best available cached compilation
JIT : Best runtime performance

We’re taking the JVM on a new journey – want to come too?



What’s in store for Java?

GPU’s?

Quantum Computers?

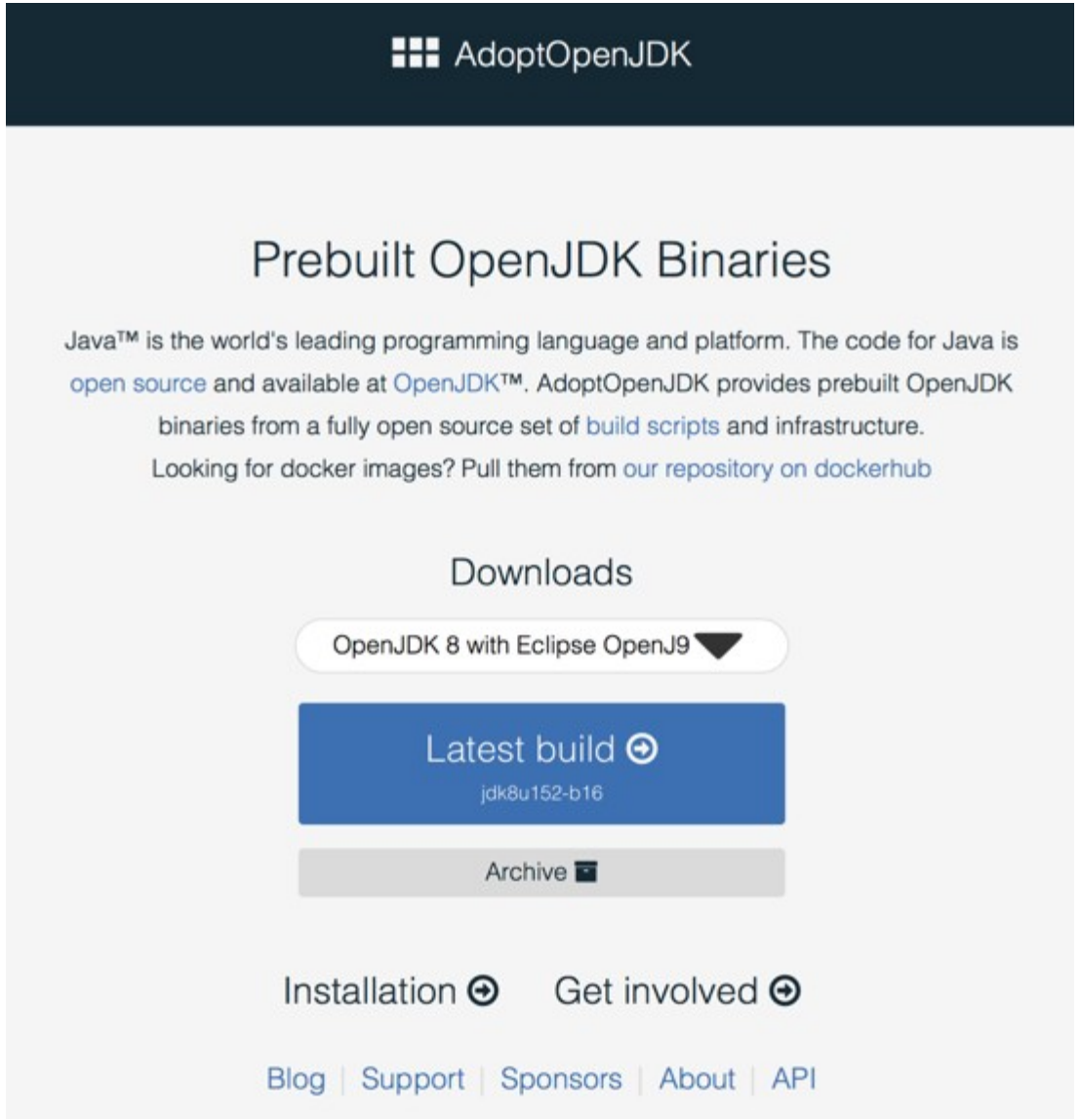
FPGAs?

<your goal here>?

Thank you

ECLIPSE FOUNDATION

OpenJ9



adoptopenjdk.net