OpenJ9 a Lean, Mean Java Virtual Machine

Billy Korando
Developer Advocate - IBM
@BillyKorando
william.korando@ibm.com

https://billykorando.com/category/openj9/
https://github.com/wkorando/openj9-batch-processor

@BillyKorando

Subscribe to the Java Newsletter

https://developer.ibm.com/newsletters/java/

For your Spring & JakartaEE needs

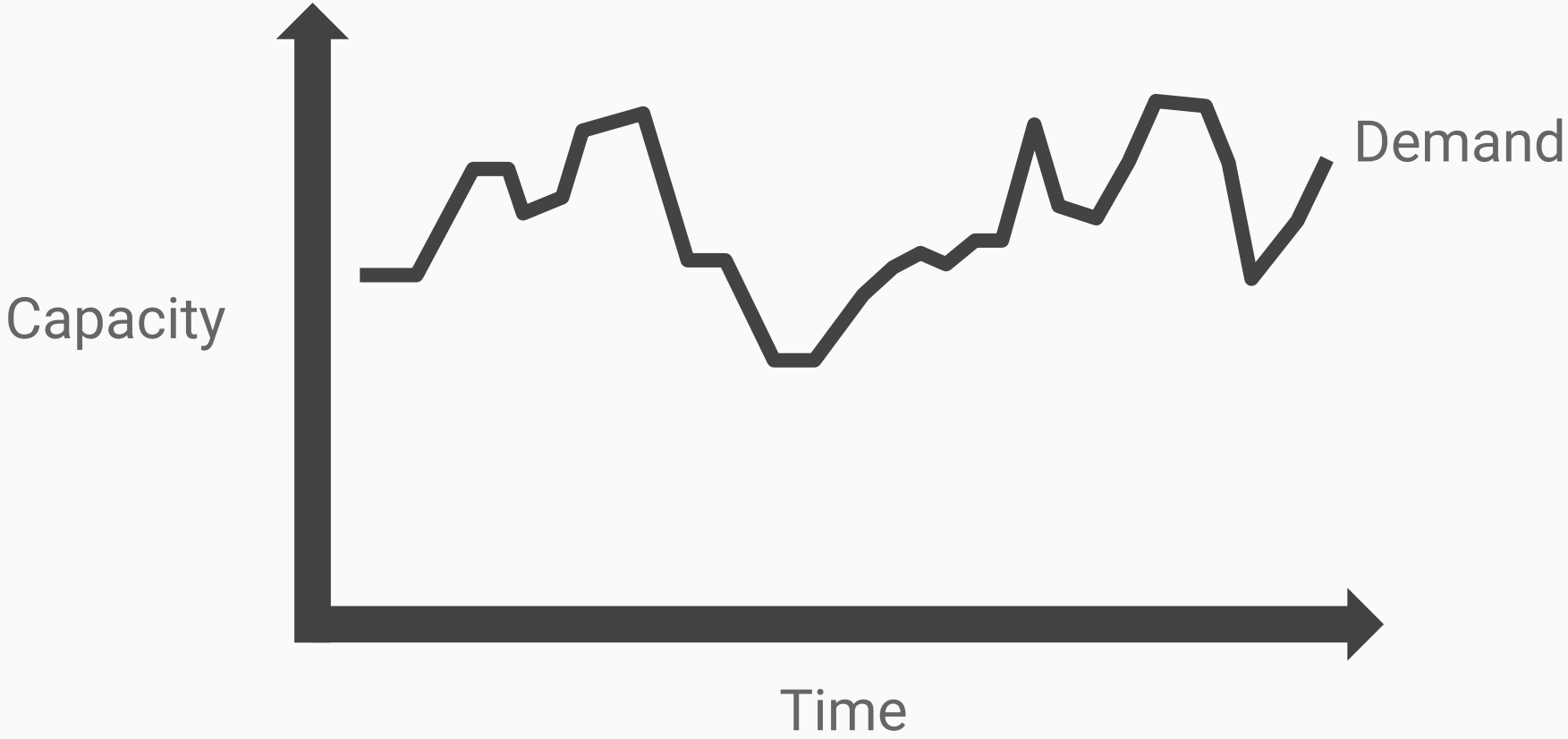
https://cloud.ibm.com/docs/java

@BillyKorando

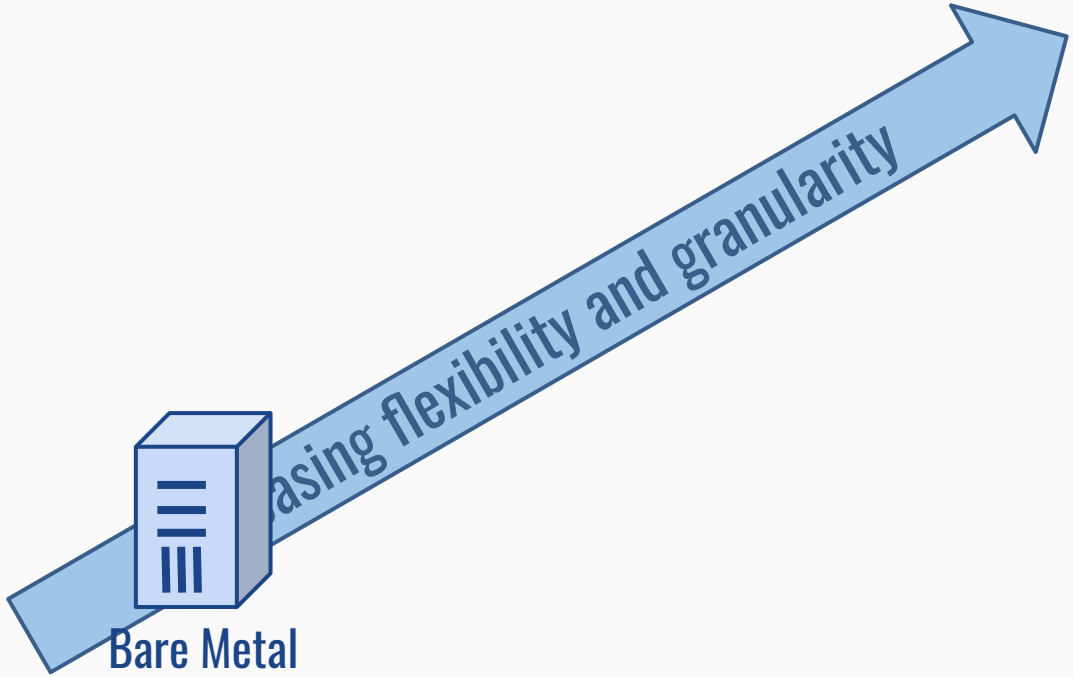
Ideal Demand

@BillyKorando

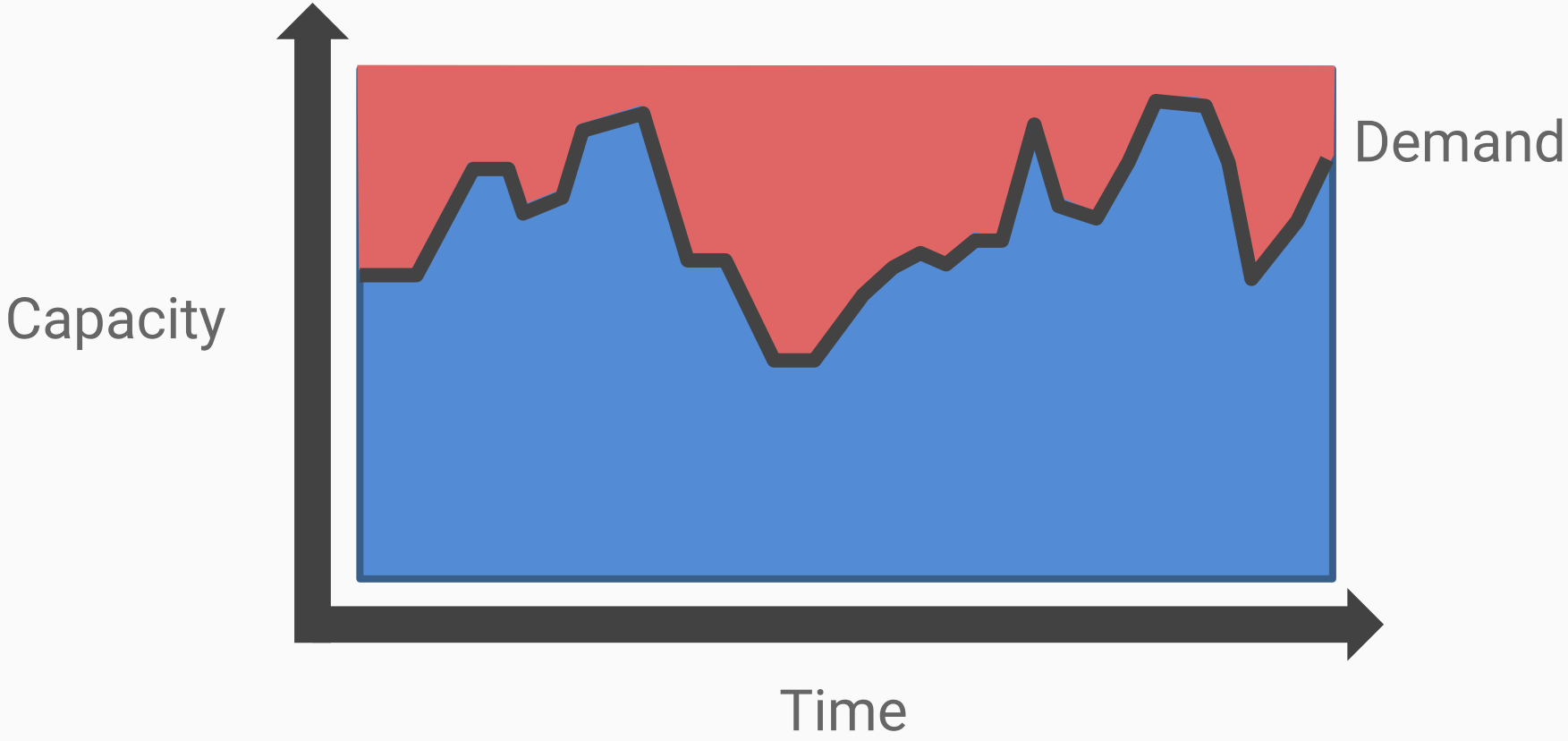
Real World Demand



Evolution of Running Applications

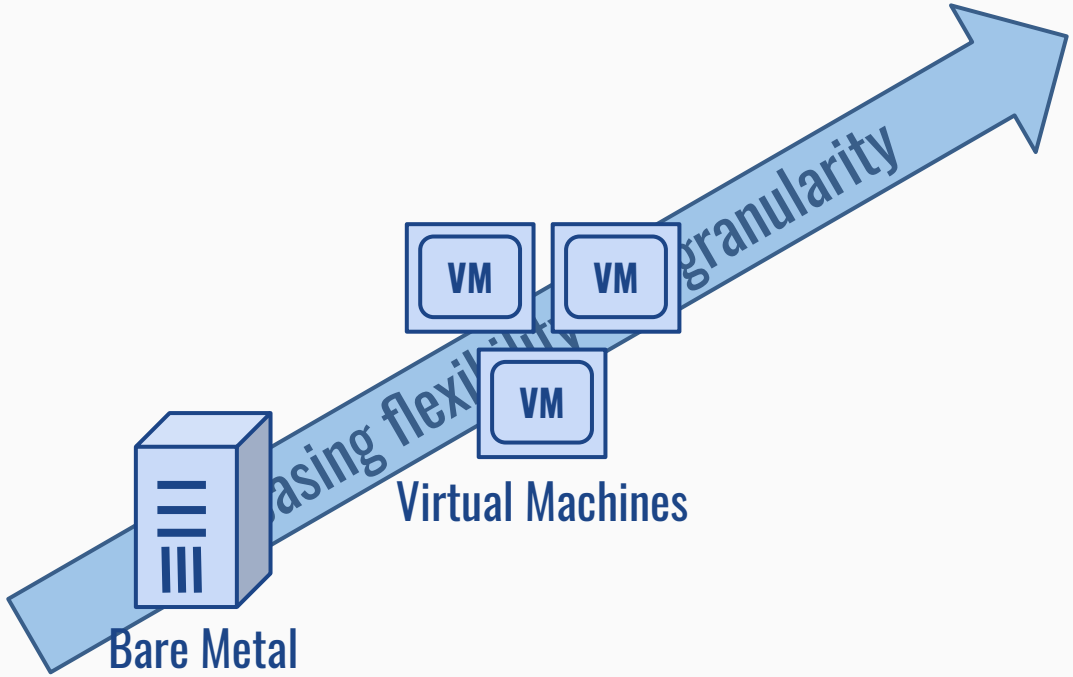


Meeting Demand - Bare Metal

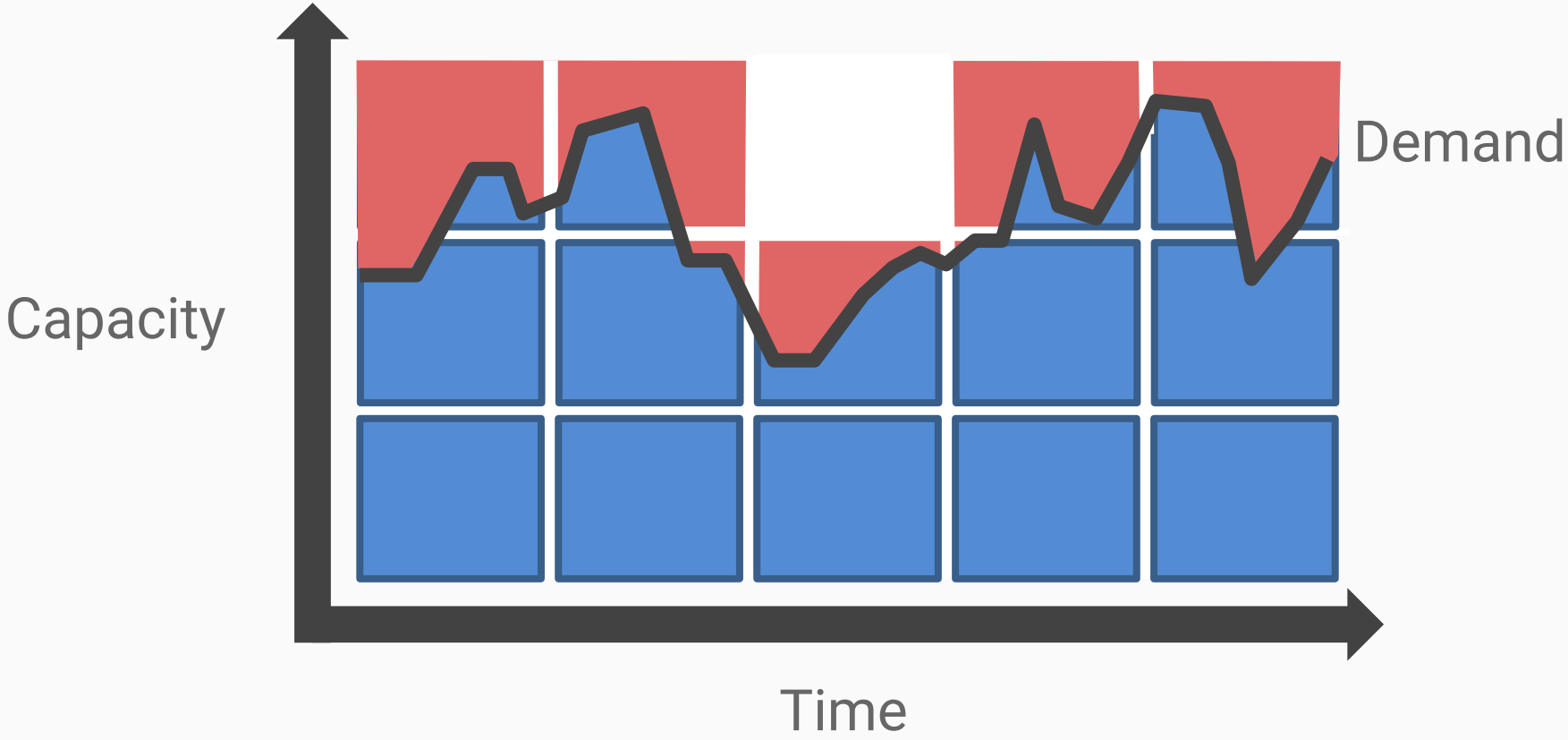


@BillyKorando

Evolution of Running Applications

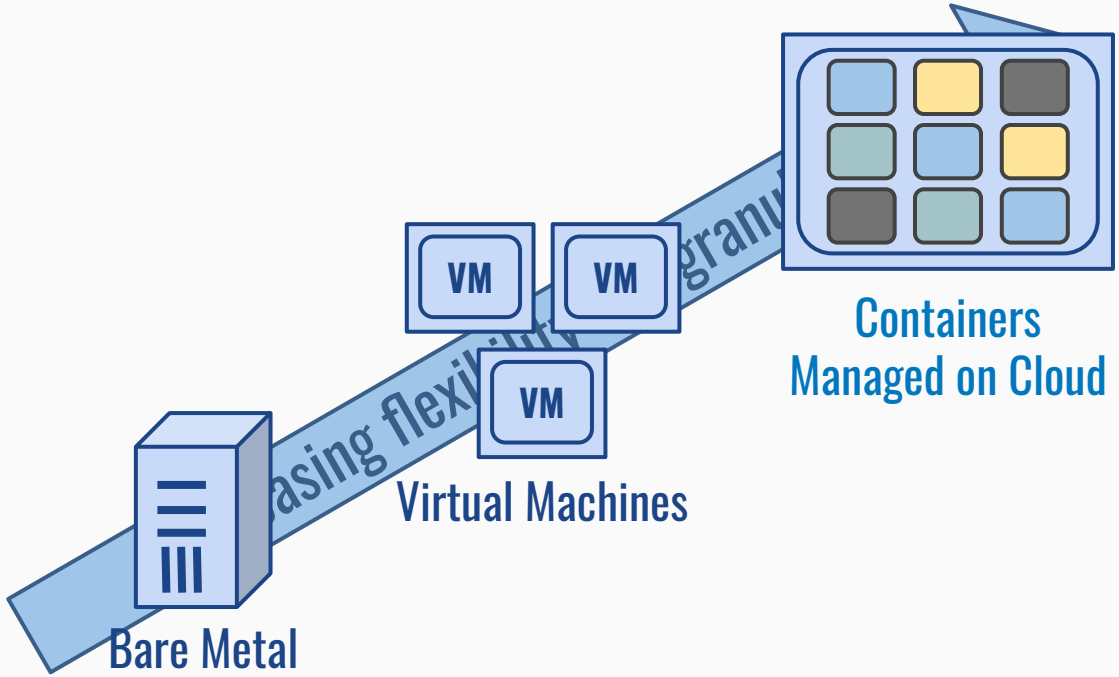


Meeting Demand - VMs

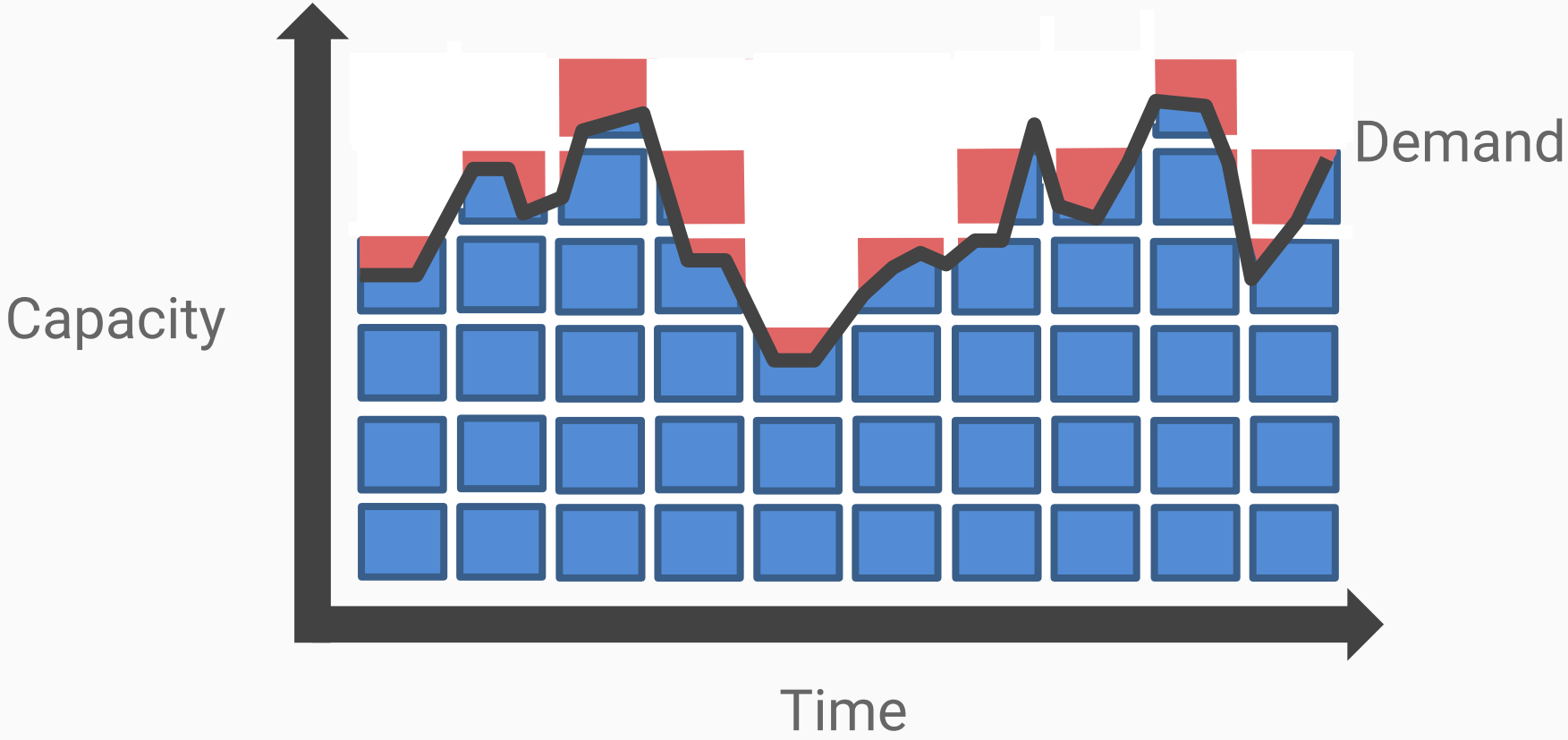


@BillyKorando

Evolution of Running Applications



Meeting Demand - Containers/Cloud



@BillyKorando

Benefits of Running in the Cloud

- Rapidly expand/reduce capacity in response to demand
- Reduced capital costs
- Reduced operations costs

The Cloud Gives Us Compute on Demand

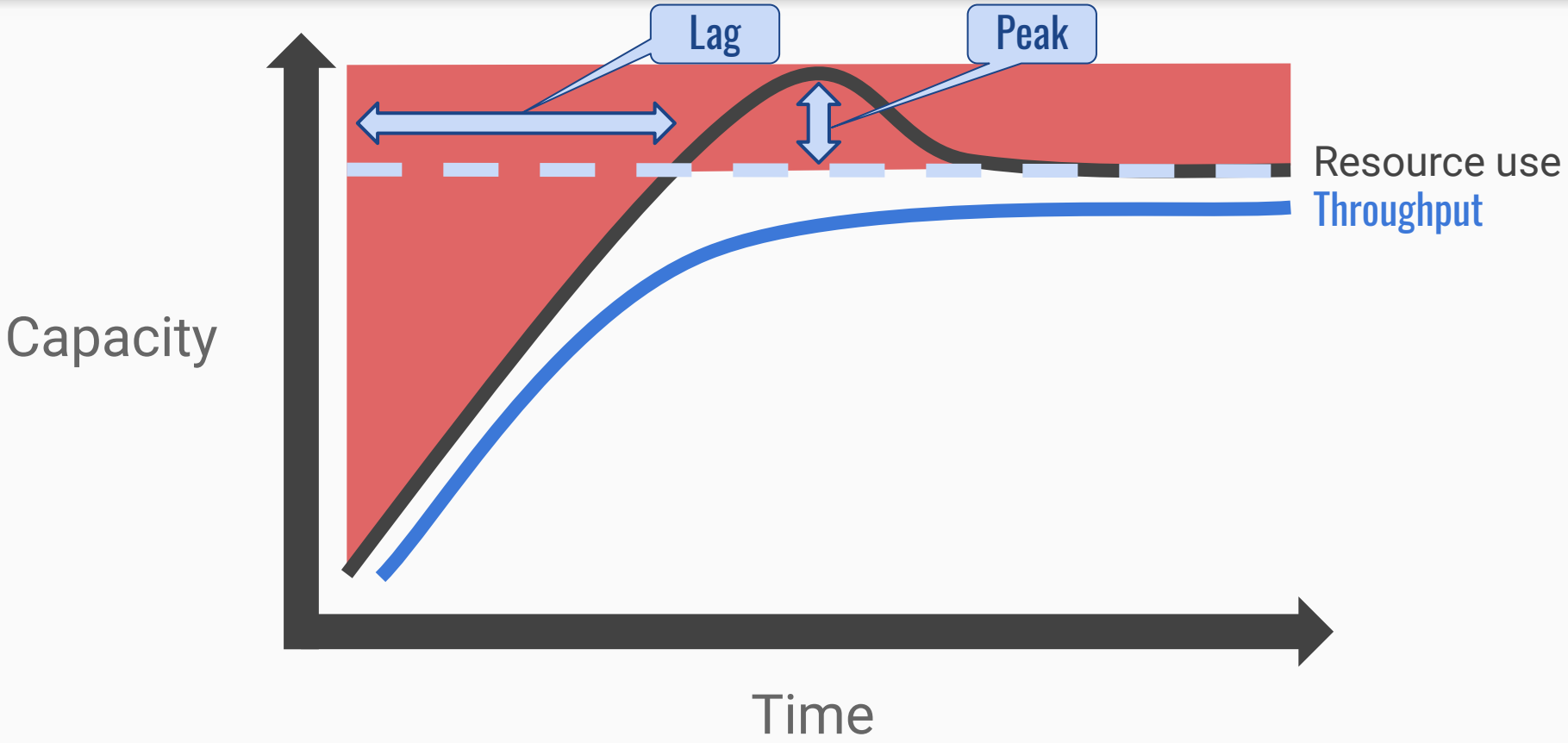
@BillyKorando

The Demand Is Metered

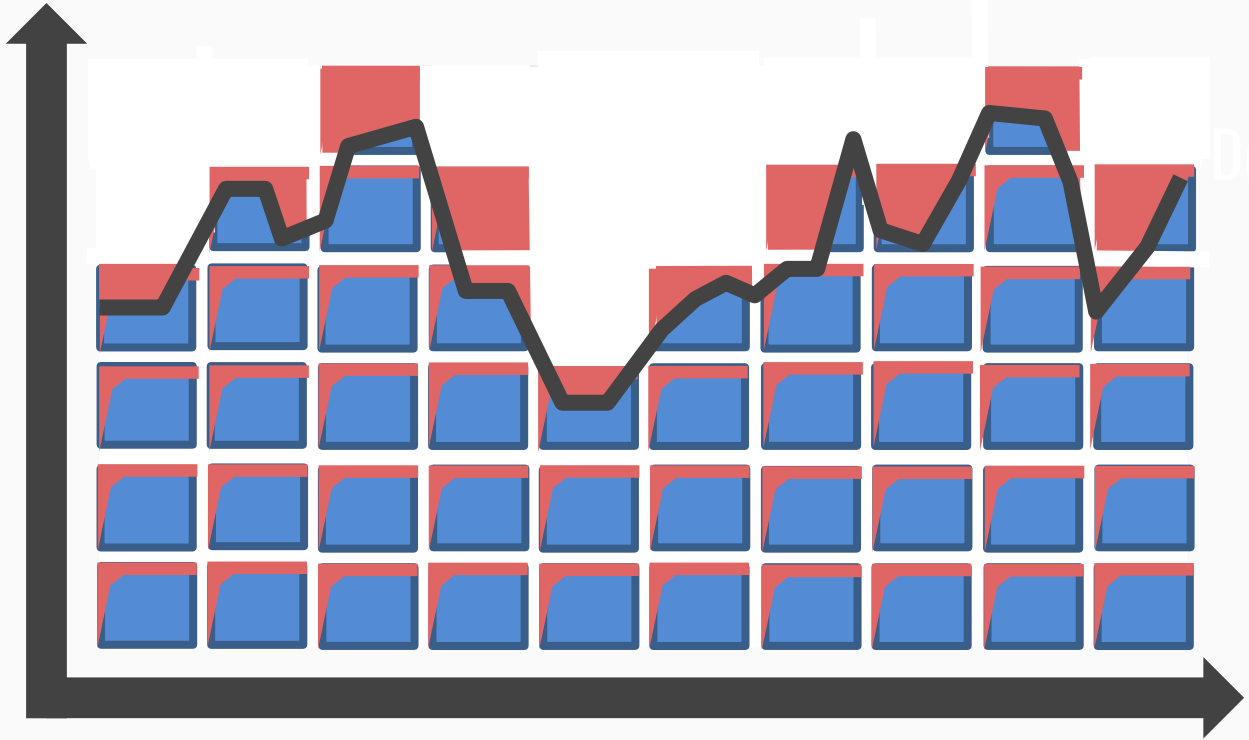
-Xmx =

@BillyKorando

Traditional Profile



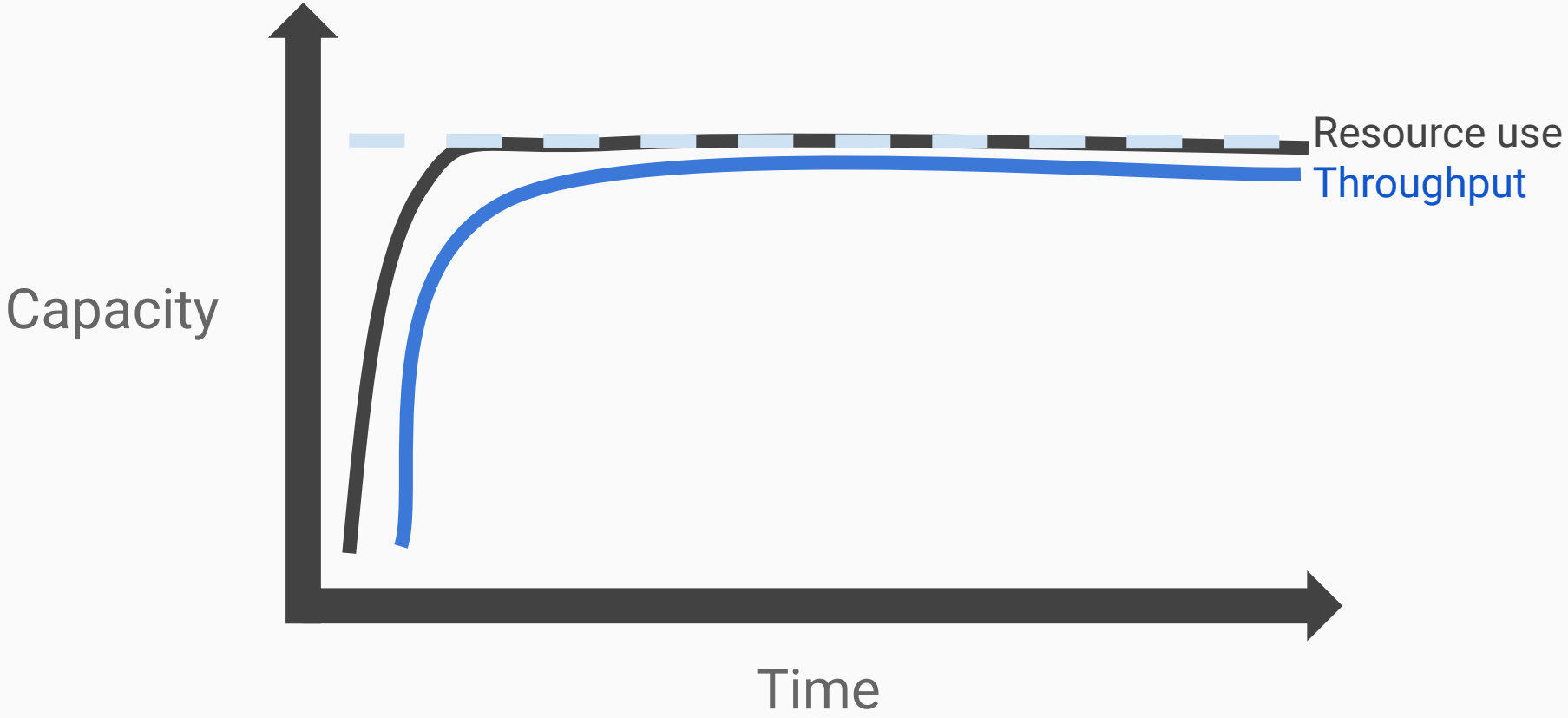
Meeting Demand - Containers/Cloud



Demands of Running in the Cloud

- Fast startup
- Minimal memory footprint
- Minimal CPU usage
- Small deployment size
- No resource usage when idle

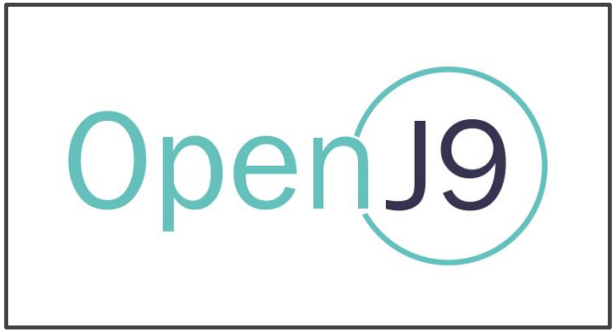
Ideal Profile



@BillyKorando

Introducing OpenJ9

@BillyKorando



Eclipse OpenJ9

Created Sept 2017

http://www.eclipse.org/openj9

https://github.com/eclipse/openj9

Dual License:

Eclipse Public License 2.0

Apache 2.0

Users and contributors welcome

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

IBM Offers Support for OpenJ9

https://www.ibm.com/us-en/marketplace/support-for-runtimes

@BillyKorando

A Little History on OpenJ9

@BillyKorando

OpenJ9 != Java 9

Built off a Smalltalk VM called K8. Developers thought Java a step back, but the new VM a step forward, thus J9

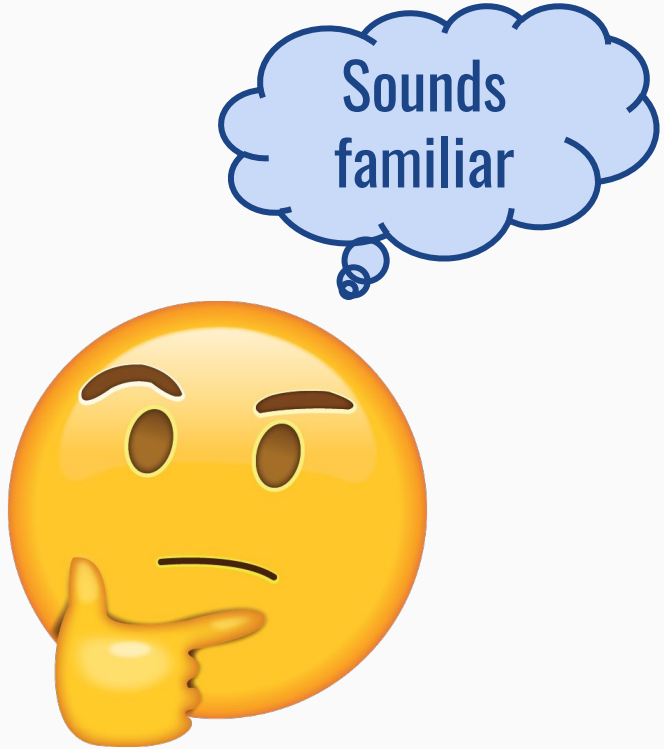
Source: https://medium.com/@rservant/how-did-the-j9-in-openj9-get-its-name-95a6416b4cb9

Background in Mobile Devices Running Java ME

@BillyKorando

Requirements for Running on Mobile

- Fast startup
- Minimal memory footprint
- Minimal CPU usage
- Small storage footprint
- No resource usage when idle



@BillyKorando

The Demo

Comparing Docker baselines (i.e. no JVM tuning) of:

Hotspot, Corretto, GraalVM

Vs

OpenJ9

The Demo

Spring Batch application overview:

1. Reading from CSV (~20K records)
2. Performing data transform
3. Writing to in-memory database (H2) using JPA
4. Performing SHA hashes on records
5. Performing checks on records
    a. Simple logic and regexes
6. Printing to console
7. Transform and print object as JSON to console

Tuning OpenJ9

@BillyKorando

Performance Isn’t Just Throughput!

Other considerations…

- Resource usage
- Startup/rampup time
- Footprint…

Let’s look at how to tune OpenJ9...

Migrating to OpenJ9

OpenJ9 supports a lot of the same basic command-line arguments Hotspot uses...

-Xmx, -Xms, -XX:InitialRamPercentage, -Xbootclasspath, and more!

https://www.eclipse.org/openj9/docs/cmdline_migration/

Cloud Native Arguments for a Cloud Native JVM

`-Xtune:virtualized`

- Tuning for containers
- Enabled VM idle management
- Improve start-up and ramp-up
- Small loss in throughput

Sharing Optimizations

`-Xquickstart`

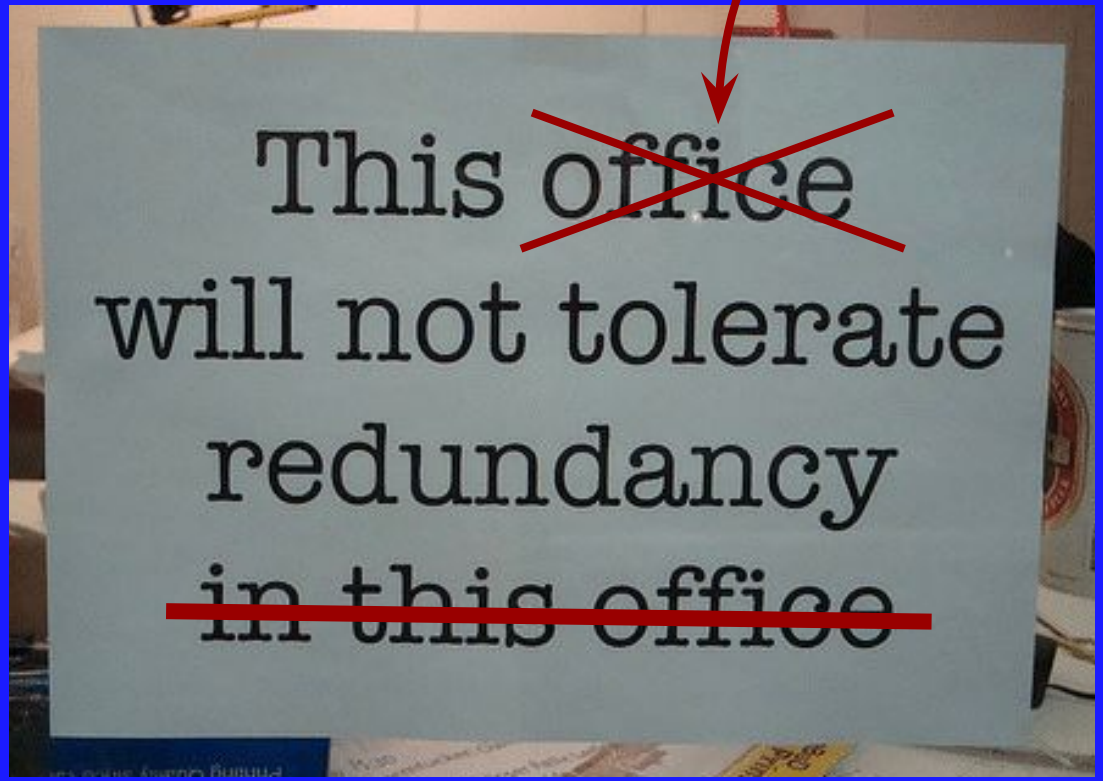
- Faster startup
- Tradeoff is some throughput loss
- Great for short-lived tasks (i.e. functions)

Additional Tuning...

- -XX:+UseContainerSupport (enabled by default)
  - Allow the JVM to use more available container memory
    - 50% < 1GB
    - 512 MB 1-2GB
    - 75% >2GB
    - Can be modified
- -XX:+IdleTuningCompactOnIdle
  - Compacts the Java heap down when idle (think defraging)
- -XX:+IdleTuningGcOnIdle
  - Releases heap memory back to system when idle (more effective with the above enabled)

JVMs do a lot of Repetitive Work

- Rebuilding the same class files (again)
- Performing the same optimizations (again)
- Carry around their own JIT to do the above

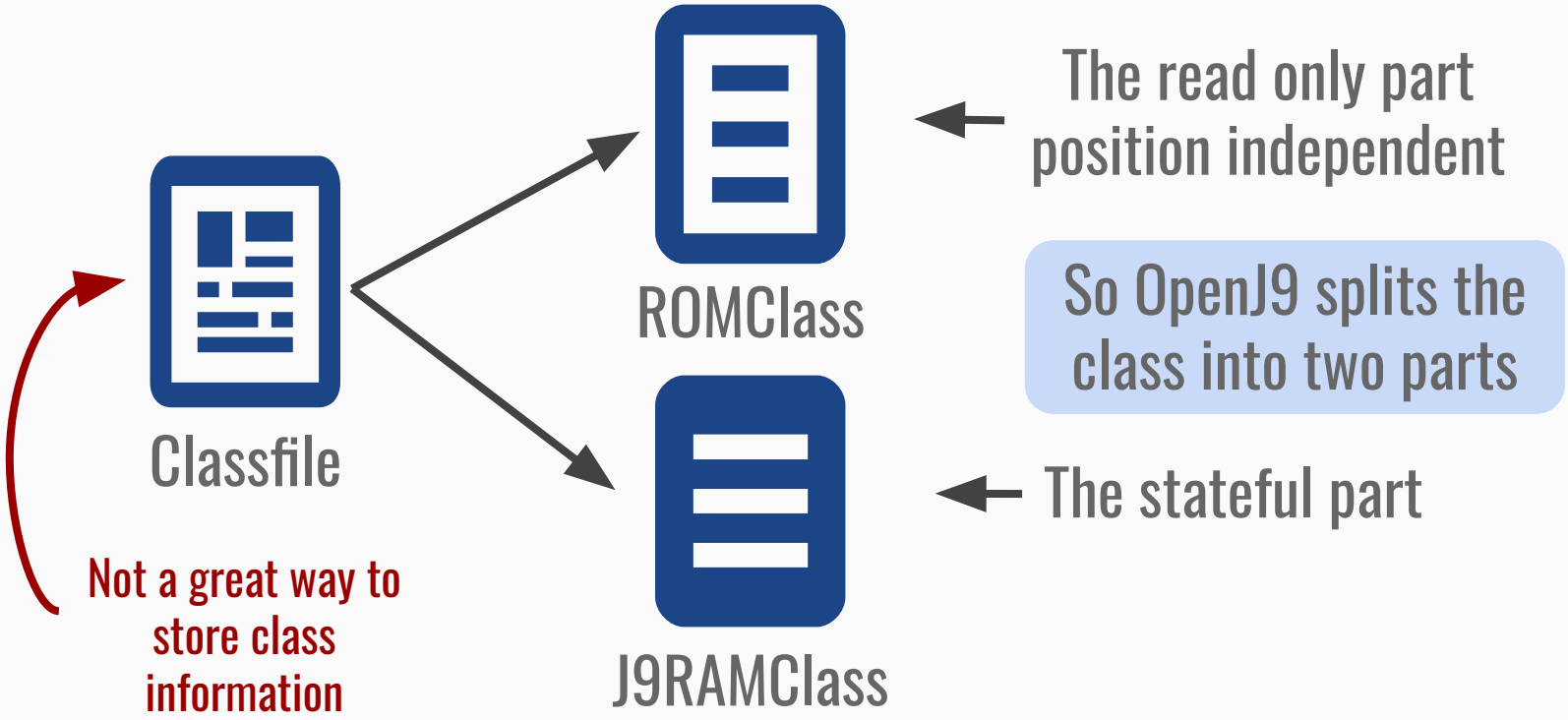


@BillyKorando

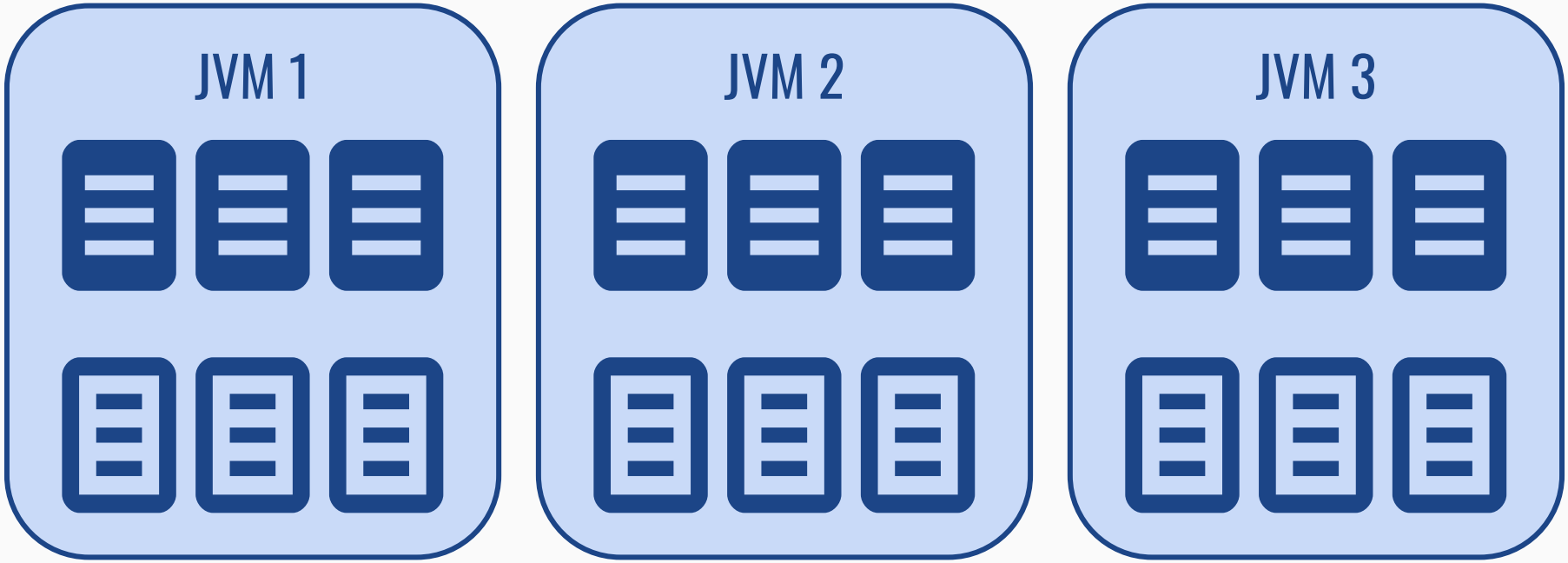
With OpenJ9 JVMs Can Work Together

@BillyKorando

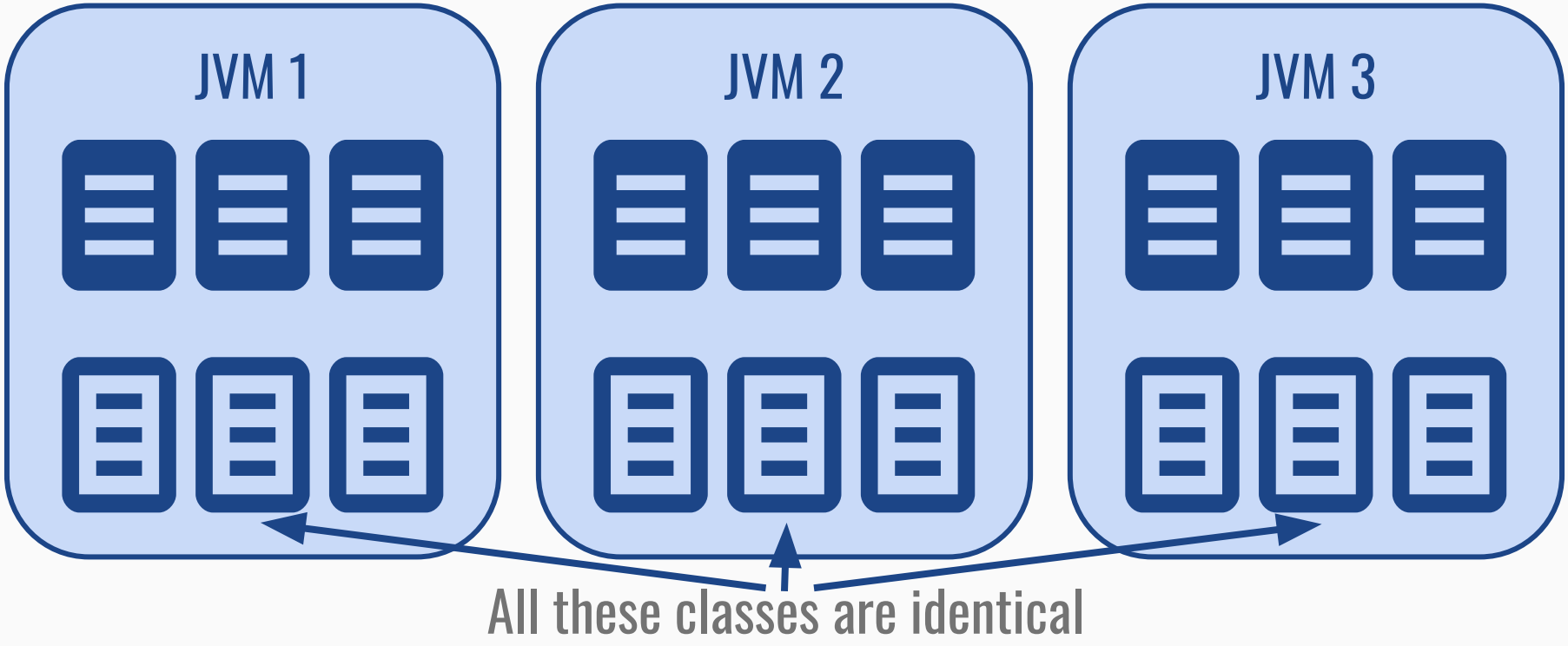
Sharing Class Content



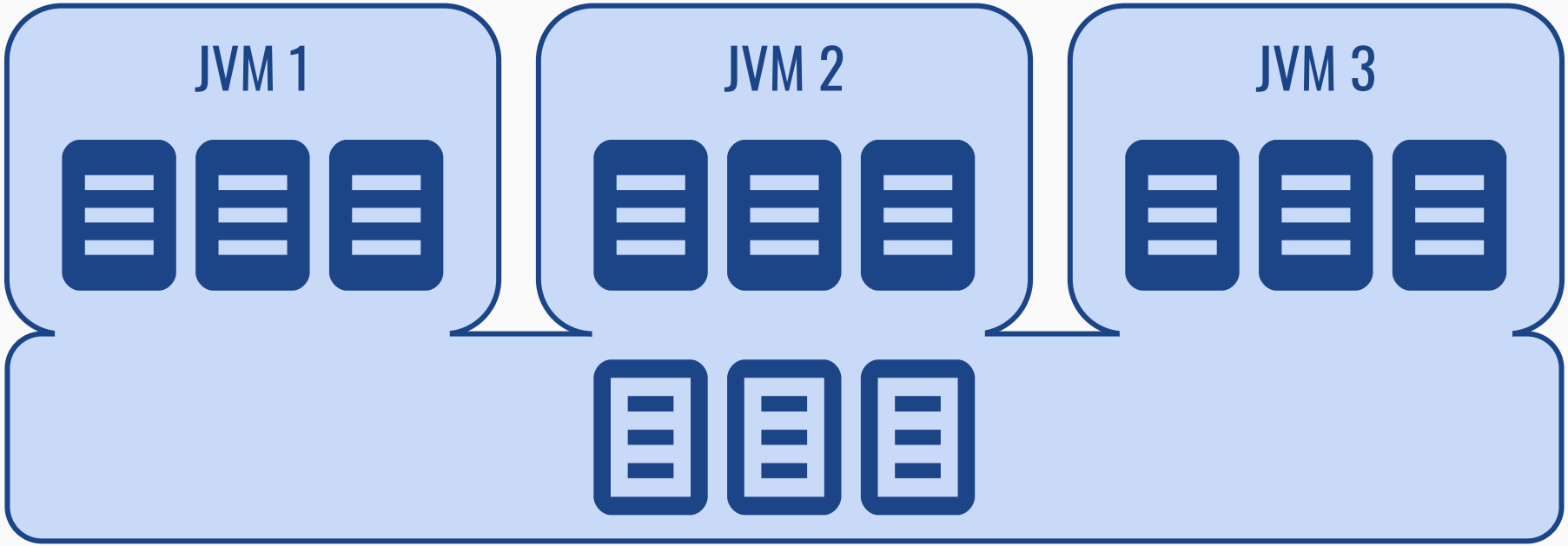
Shared Static Classes in Practice



Sharing Classes in Practice



Shared Static Classes in Practice



So let's share them!

@BillyKorando

Sharing Class Content

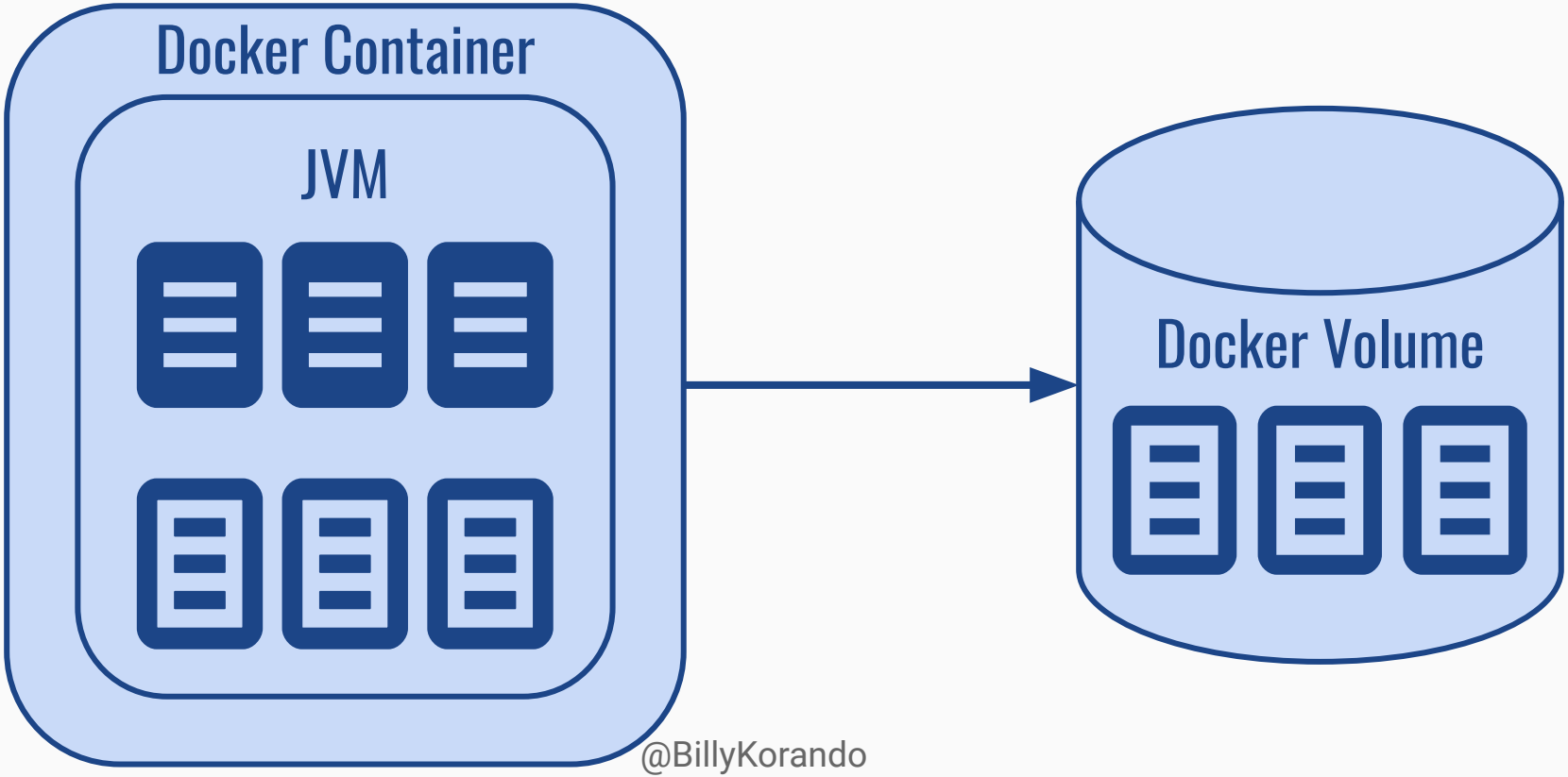
`-Xshareclasses`
Enables class sharing

`-Xscmx=50M`
Sets size of the cache (in this case 50 MB)
If not set, default is 300MB

Sharing Class Content with Containers

Reduces startup and footprint by ~20%

Shared Static Classes in Containers with Volumes



Sharing Class Content with Containers

1. Create a docker volume

```
docker volume create java-shared-classes
```

2. Have the docker image mount the volume

```
--mount
source=java-shared-classes,target=/cache
```

3. Tell the VM to store class info there

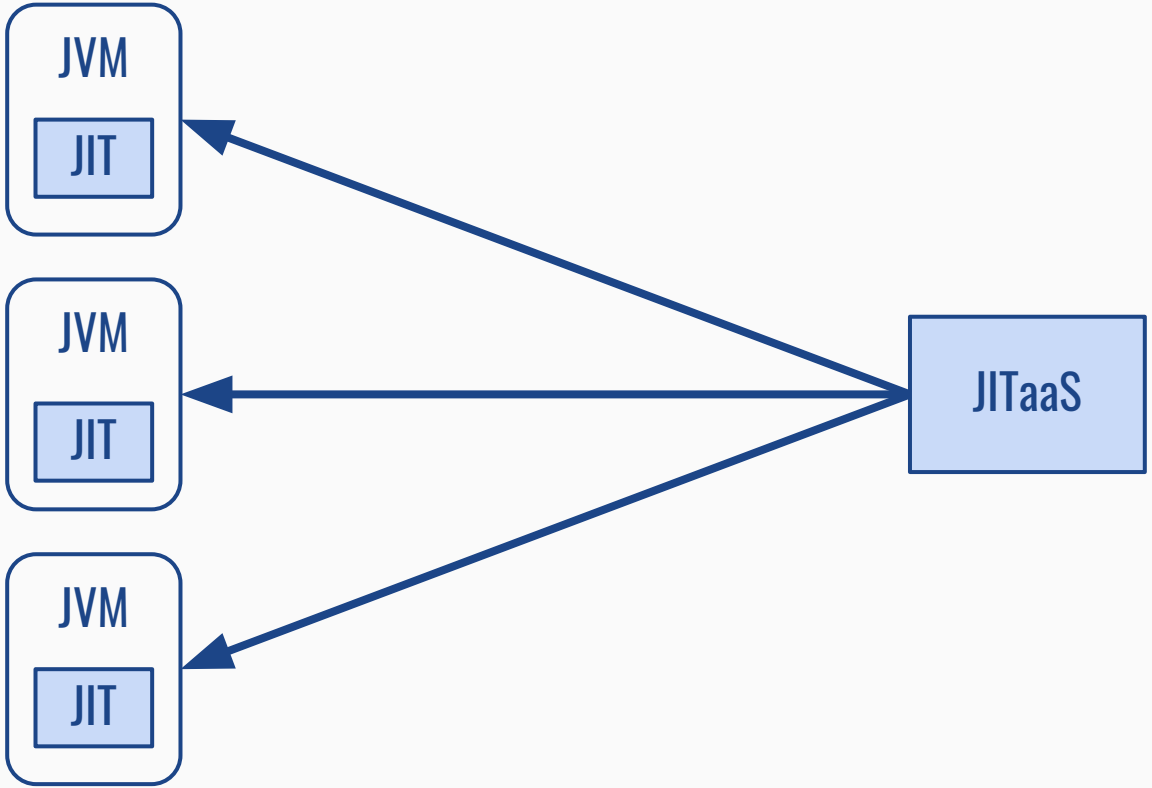
```
-Xshareclasses:cacheDir=/cache
```

(The above are just sample values)

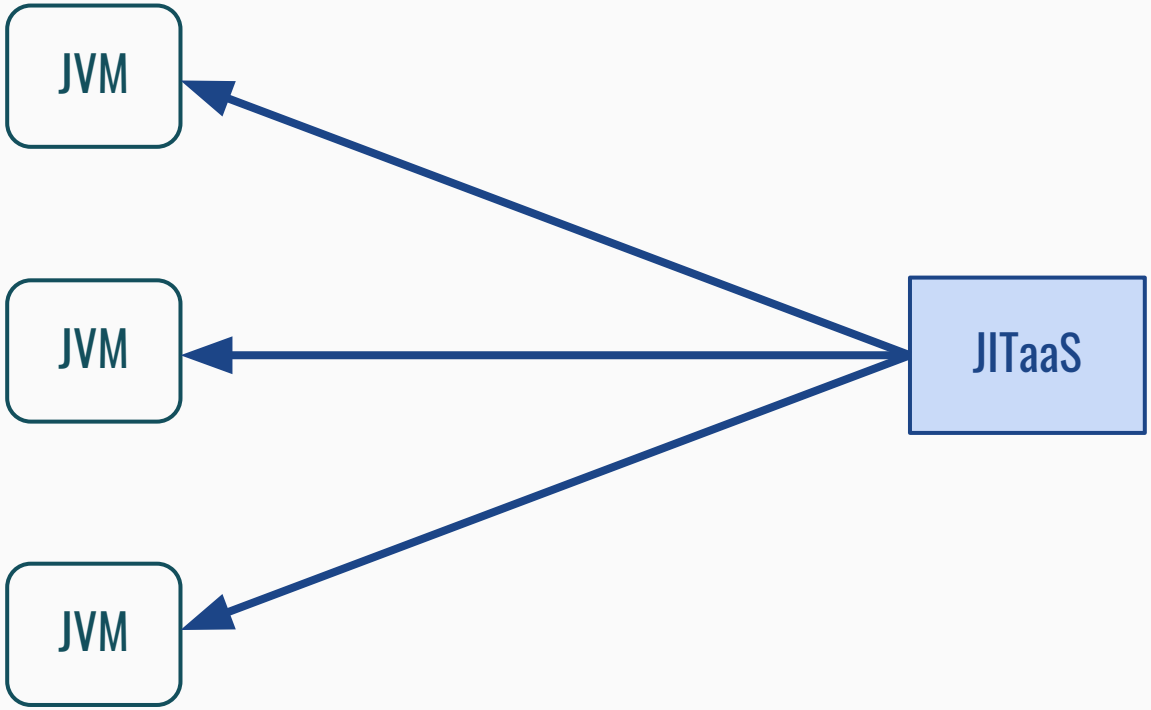
@BillyKorando

Speaking of JITs...

JIT as a Service (JITaas)



JIT as a Service (JITaas)



JITaaS Advantages

- ~40% less CPU usage
- No memory spikes from JIT optimizations
- You only need resources for *your* application's needs

Enabling JITaaS

`-XX:JITaaSServer`

Enables a standalone JVM to run a server from processing JIT

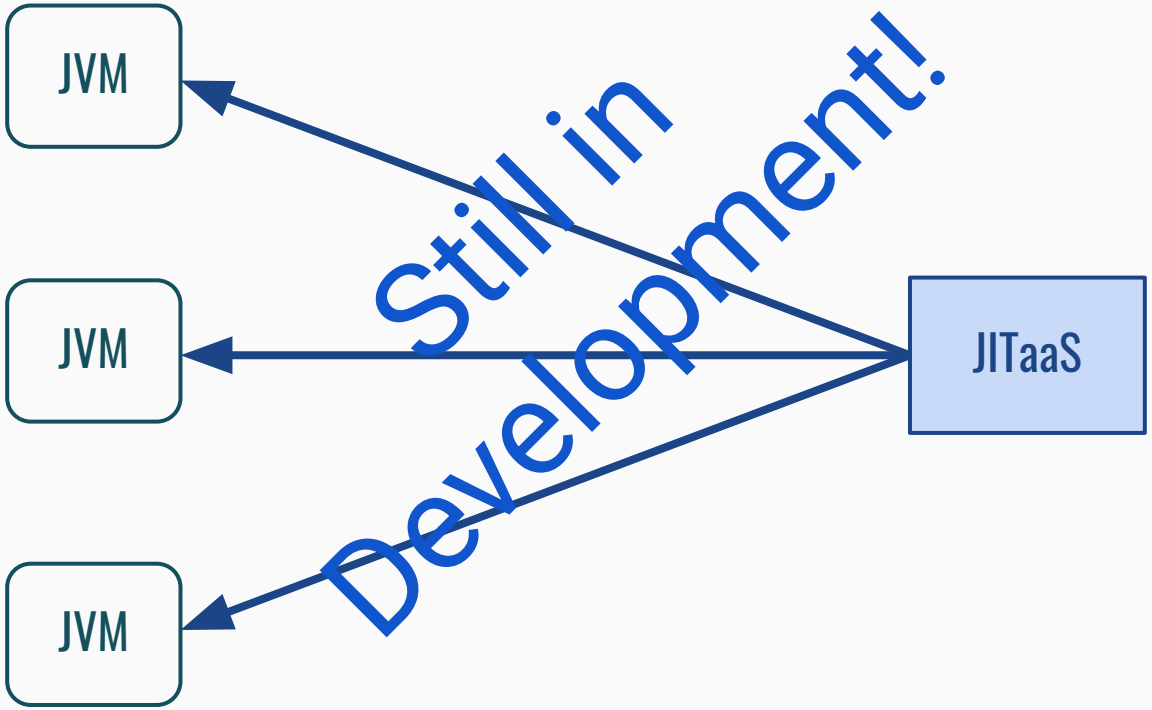
`-XX:JITaaSClient:server=<host>`

Tell client where JITaaS is located

`-Xnojit`

Not necessary, but then you won't be getting much benefit :)

JIT as a Service (JITaas)



Additional Reading

OpenJ9 User Docs: https://www.eclipse.org/openj9/docs/

JVM comparisons: https://chriswhocodes.com/

OpenJ9 Class Sharing: https://developer.ibm.com/tutorials/j-class-sharing-openj9/

OpenJ9 Class Sharing in Docker: http://ibm.biz/openj9-class-sharing-docker

Q&A

@BillyKorando

Download OpenJ9: adoptopenjdk.net
Slides: ibm.biz/OpenJ9-lean-mean-JVM
Code: https://github.com/wkorando/openj9-batch-processor
IBM Cloud sign-up: https://ibm.biz/BdzzsQ