Java™

ORACLE®

Java for the containers

Vaibhav Choudhary (@vaibhav_c)
Java Platforms Team
https://blogs.oracle.com/vaibhav

Java
Your
Next
(Cloud)

Safe Harbor Statement

Agenda of the day …

1. A brief on Container
2. Java - as first class citizen in Container
3. Respect Container Boundary
4. Leverage features inside container
5. Final thoughts

Respective the container Boundary

Processor Boundary (--cpu-shares)

```
public class CoreCheck {
   public static void main(String[] args) {
     System.out.println(Runtime.getRuntime().availableProcessors());
 }}
```

JDK8 :

```
docker run --rm --cpu-shares 2048 -v /root/:/mnt/mydata -it ubuntu bash
root@7599aae2613d:/mnt/mydata/jdk-8/bin# ./java CoreCheck

8
```

```
FROM openjdk:10
ADD CoreCheck.java .
WORKDIR .
RUN javac CoreCheck.java
CMD ["java", "CoreCheck" ]
```

```
docker build -t hello-world .
docker run --cpu-shares 2048 java-helloworld

2
```

Processor Boundary (--cpuset-cpus)

```
public class CoreCheck {
   public static void main(String[] args) {
     System.out.println(Runtime.getRuntime().availableProcessors());
 }}
```

JDK8 :

```
docker run --rm --cpuset-cpus 3,4,6 -v /root/:/mnt/mydata -it ubuntu bash
root@7599aae2613d:/mnt/mydata/jdk-8/bin# ./java CoreCheck

8 (ideally it should say 3)
```

```
FROM openjdk:10
ADD CoreCheck.java .
WORKDIR .
RUN javac CoreCheck.java
CMD ["java", "CoreCheck" ]
```

```
docker build -t hello-world .
docker run --cpuset-cpus 3,4,6 java-
helloworld

3 (Processor 3rd, 4th and 6th)
```

Here are the results (Respecting Logs)

```
public class HelloWorld {
   public static void main(String[] args) {
     System.out.println(Runtime.getRuntime().availableProcessors());
 }}
```

JDK10 :

```
root@4795a54e037e:/mnt/mydata/jdk-10b38/bin# ./java -Xlog:os+container=trace HelloWorld

[0.001s][trace][os,container] OSContainer::init: Initializing Container Support
[0.001s][trace][os,container] Path to /memory.limit_in_bytes is /sys/fs/cgroup/memory/
memory.limit_in_bytes
```

JDK9 :

```
root@4795a54e037e:/mnt/mydata/jdk-9/bin# ./java -Xlog:os+container=trace HelloWorld
[0.001s][error][logging] Invalid tag 'container' in what-expression.
Invalid -Xlog option '-Xlog:os+container=trace'
```

Here are the results (ActiveProcessorCount)

```
public class HelloWorld {
   public static void main(String[] args) {
     System.out.println(Runtime.getRuntime().availableProcessors());
 }}
```

JDK10 :

```
root@4795a54e037e:/mnt/mydata/jdk-10b38/bin# ./java -
   XX:ActiveProcessorCount=4 HelloWorld
4
```

JDK8 :

```
root@4795a54e037e:/mnt/mydata/jdk1.8.0_152/bin# ./java -
   XX:ActiveProcessorCount=4 HelloWorld
Unrecognized VM option 'ActiveProcessorCount=4'
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

Cont.. (Respecting Memory Boundaries)

```
public class MemoryCheck {
   public static void main(String[] args) {
     System.out.println(Runtime.getRuntime().maxMemory());
 }}
```

JDK8 :

```
docker run --rm --memory 100m -v /root/:/mnt/mydata -it ubuntu bash
root@7599aae2613d:/mnt/mydata/jdk-8/bin# ./java CoreCheck
```

It will print 1/4 of the Physical Memory (It should understand docker memory limit)

```
FROM openjdk:10
ADD MemoryCheck.java .
WORKDIR .
RUN javac MemoryCheck.java
CMD ["java", "MemoryCheck" ]
```

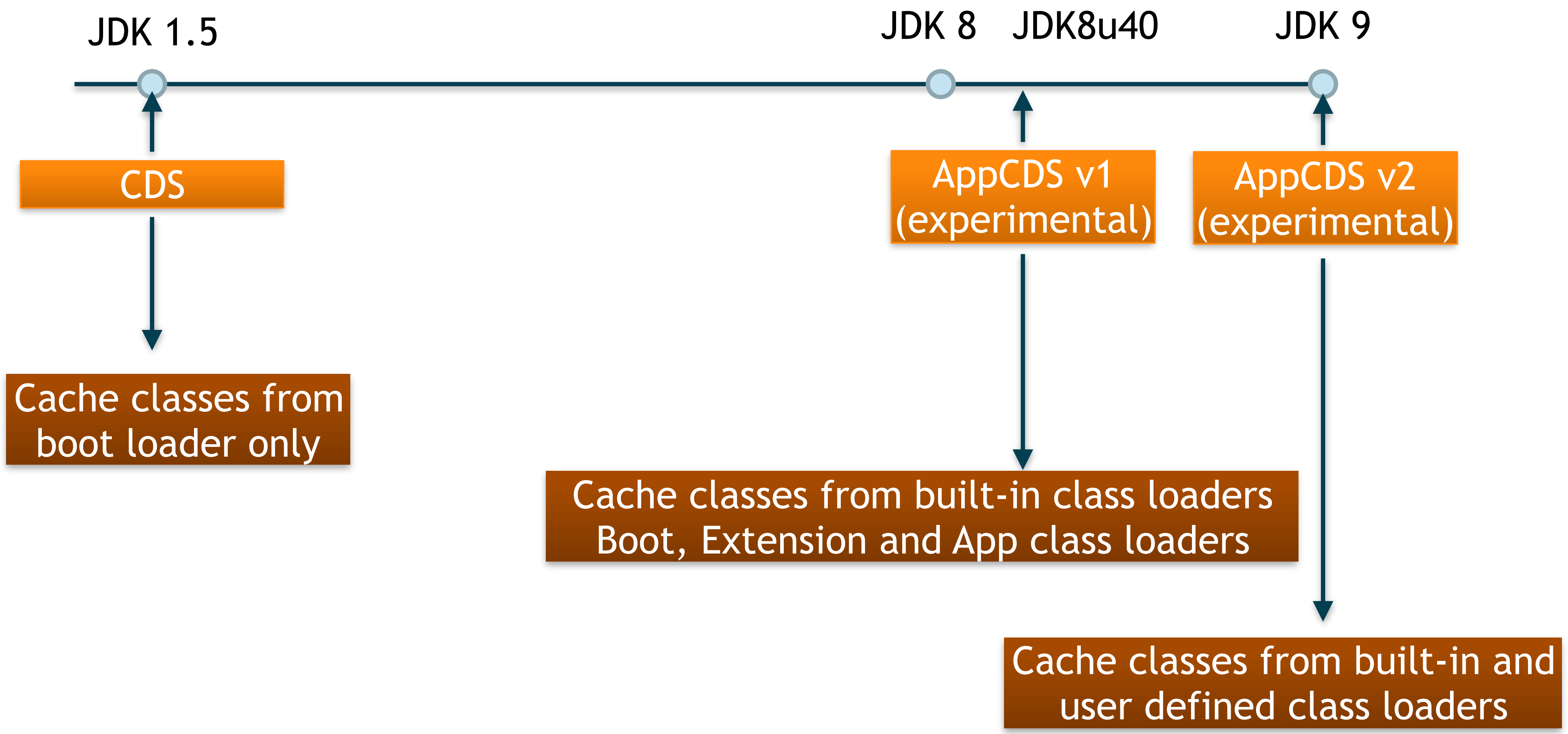
```
docker build -t hello-world .
docker run --memory 100m java-helloworld

Half of the docker limit (50m)
```

Solution 1 : Application Class Data Sharing (AppCDS)

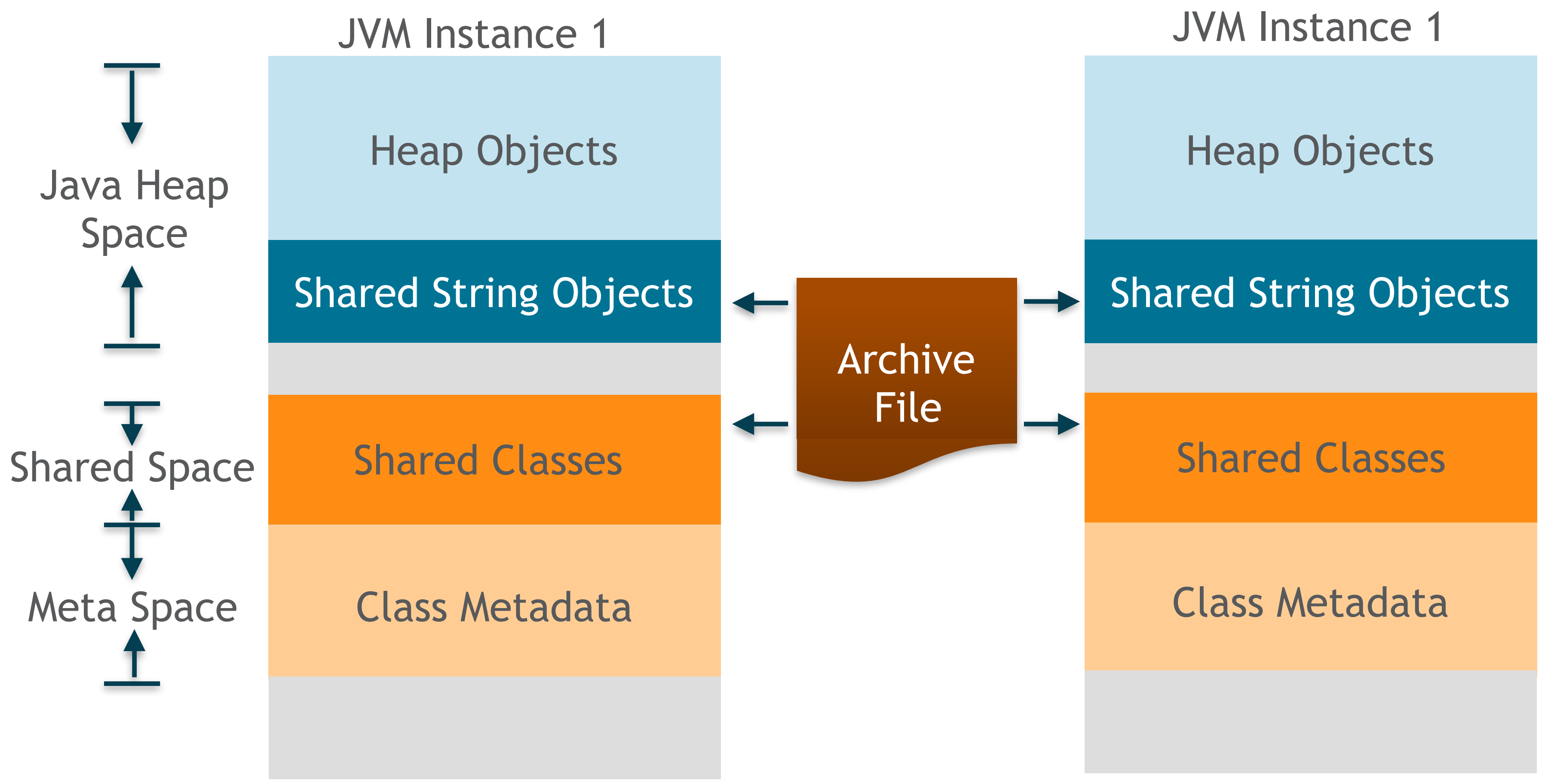
How Application Class Data Sharing (AppCDS) Help?

- CDS only supports archiving classes on the bootstrap class path, which provides limited (or no) benefit for large applications in today's cloud environment as majority of loaded classes are application classes.
- AppCDS extends the archiving support beyond core library classes
  - Enables archiving classes loaded by all class loaders
    - PlatformClassLoader (the ExtensionClassLoader in JDK 8 and earlier versions)
    - AppClassLoader
    - User defined class loaders
- AppCDS allows archiving and sharing read-only java heap objects with G1 GC
- AppCDS provides further reduction in storage for archived class metadata

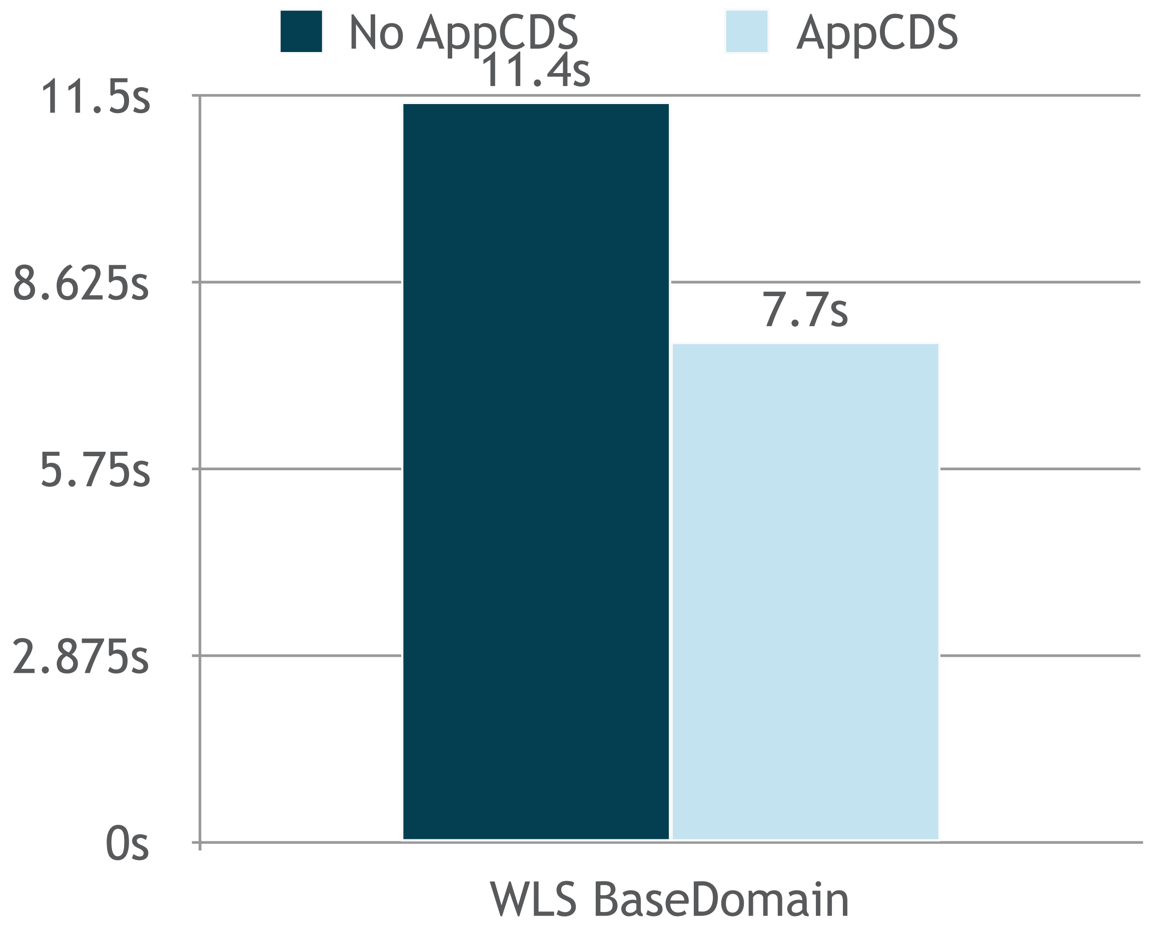
A Quick Look at CDS/AppCDS Evolution


JDK 1.5
JDK 8
JDK8u40
JDK 9
CDS
AppCDS v1 (experimental)
AppCDS v2 (experimental)
Cache classes from boot loader only
Cache classes from built-in class loaders Boot, Extension and App class loaders
Cache classes from built-in and user defined class loaders

AppCDS Architectural View



Startup Time Improvements for Oracle Web Logic Server



- About 30% startup time improvement observed with AppCDS for Oracle Web Logic Server (WLS) base domain

AppCDS inside Container - How to use

- Simple CDS use in Docker

DockerFile

```
FROM openjdk:10
ADD HelloWorld.java .
WORKDIR .
RUN javac HelloWorld.java
RUN ["java", "-Xshare:dump" ]
CMD ["java", "-showversion" , "-Xshare:on", "HelloWorld" ]
```

Cont…

- Determine the class to archive

```
java -Xshare:off -XX:+UseAppCDS -XX:DumpLoadedClassList=hello.lst -cp hello.jar
HelloWorld
```

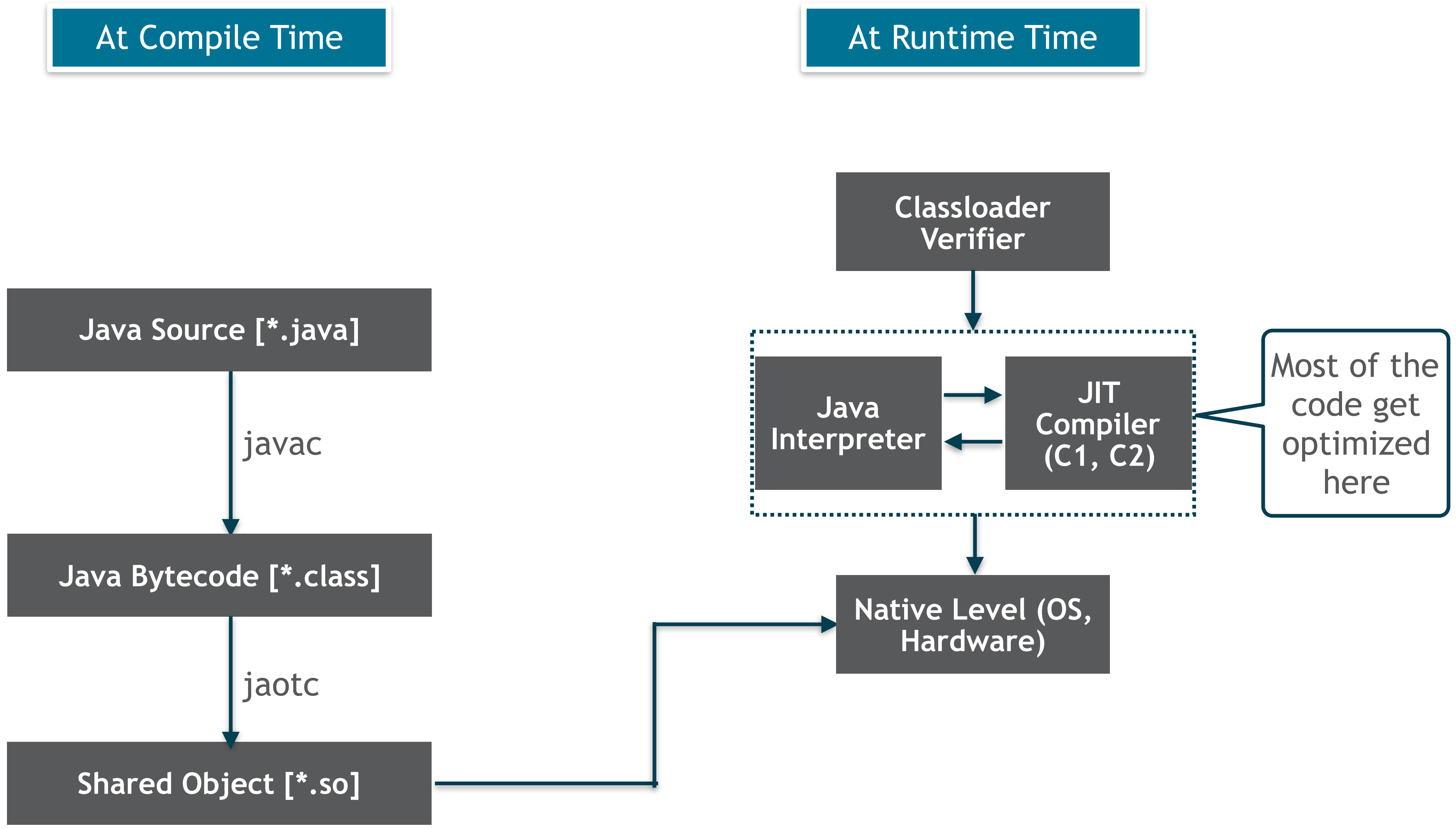
- Creating the AppCDS archive

```
java -Xshare:dump -XX:+UseAppCDS -XX:SharedClassListFile=hello.lst
-XX:SharedArchiveFile=hello.jsa -cp hello.jar
```

- Using the AppCDS archive

```
java -Xshare:on -XX:+UseAppCDS -XX:SharedArchiveFile=hello.jsa
-cp hello.jar HelloWorld
```

Solution 2: Ahead of Time Compilation (AOT)


At Compile Time
At Runtime Time
Java Source [*.java]
javac
Java Bytecode [*.class]
jaotc
Shared Object [*.so]
Classloader Verifier
Java Interpreter
JIT Compiler (C1, C2)
Most of the code get optimized here
Native Level (OS, Hardware)

- jaotc --output HelloWorld.so HelloWorld.class
- java -XX:AOTLibrary=./HelloWorld.so HelloWorld

```
-bash-4.1$ ./java -XX:+PrintAOT -XX:AOTLibrary=./HelloWorld.so
HelloWorld
     15    1     loaded    ./HelloWorld.so  aot library
    153    1     aot[ 1]   HelloWorld.<init>()V
    153    2     aot[ 1]   HelloWorld.main([Ljava/lang/String;)V
Hello AOT
```

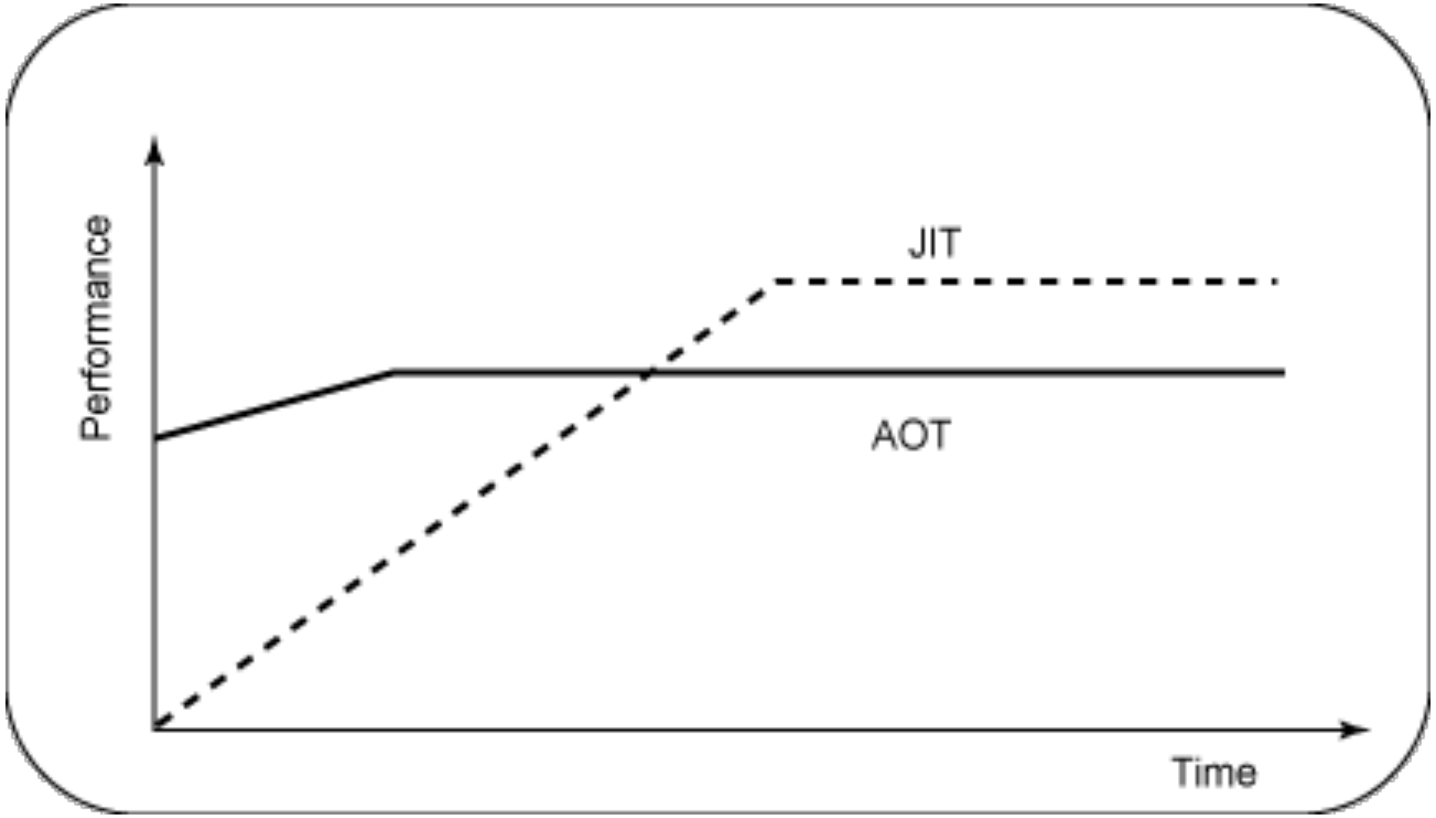
- jaotc --output base.so --module java.base

AOT run (jaotc)

```
-bash-4.1$ ./jaotc --info --output HelloWorld.so HelloWorld.class
Compiling HelloWorld...
1 classes found (62 ms)
2 methods total, 2 methods to compile (6 ms)
Compiling with 2 threads
.
2 methods compiled, 0 methods failed (923 ms)
Parsing compiled code (2 ms)
Processing metadata (21 ms)
Preparing stubs binary (1 ms)
Preparing compiled binary (0 ms)
Creating binary: HelloWorld.o (17 ms)
Creating shared library: HelloWorld.so (26 ms)
Total time: 1886 ms
```

	Dynamic (JIT)	Static (AOT)
Start-up performance	Tunable, but not so good	Best
Steady-state performance	Best	Good
Interactive performance	Not so good	Good
Deterministic performance	Tunable, but not best	Best

* https://www.ibm.com/developerworks/library/j-rtj2/index.html



* https://www.ibm.com/developerworks/library/j-rtj2/index.html

Solution 3: Modularity

DockerFile should look like

```
FROM openjdk:10 as small
ADD jdk-10.tar.gz /jdk
ENV PATH=$PATH:/jdk/jdk-10/bin
RUN [“jlink”, “—compress=2”, “—module-path”, “/jdk/jdk-10/jmods”,
“—add-modules”, “java.base”, “—output”,  “/linked” ]

FROM openjdk:10
COPY —from=small /linked /jdk
ADD HelloWorld.class
CMD [“java”, “-showversion”, “HelloWorld”]
```

Links

- For any query, you can drop me mail at vaibhav.x.choudhary@oracle.com

- Good links :-
  - Bangalore JUG resource - http://bangalorejug.org/downloads/
  - Container Aware Java - http://openjdk.java.net/jeps/8182070
  - JDK-8146115 - https://bugs.openjdk.java.net/browse/JDK-8146115
  - Better containerization by JDK10 : https://mjg123.github.io/2018/01/10/Java-in-containers-jdk10.html
  - VJUG Talk :- Java in Container world