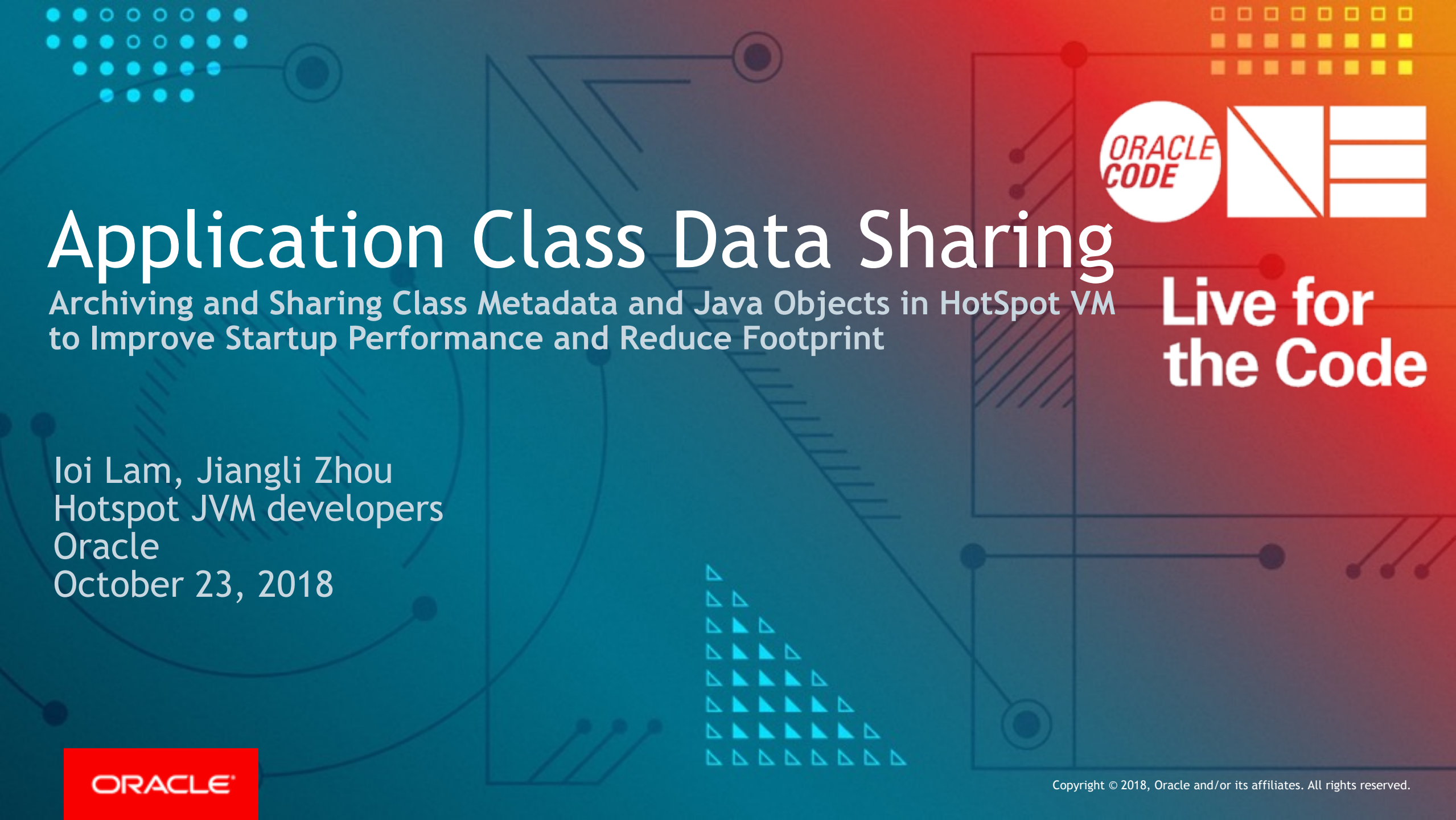


Application Class Data Sharing

Archiving and Sharing Class Metadata and Java Objects in HotSpot VM to Improve Startup Performance and Reduce Footprint

Live for the Code

Ioi Lam, Jiangli Zhou
Hotspot JVM developers
Oracle
October 23, 2018

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Agenda

1. What is Class Data Sharing?
2. What's New Since JDK 9
3. Module Support
4. Archive Java Heap Objects
5. Startup Time and Footprint Results
6. Features In Incubation
7. Conclusion

What is Class Data Sharing?

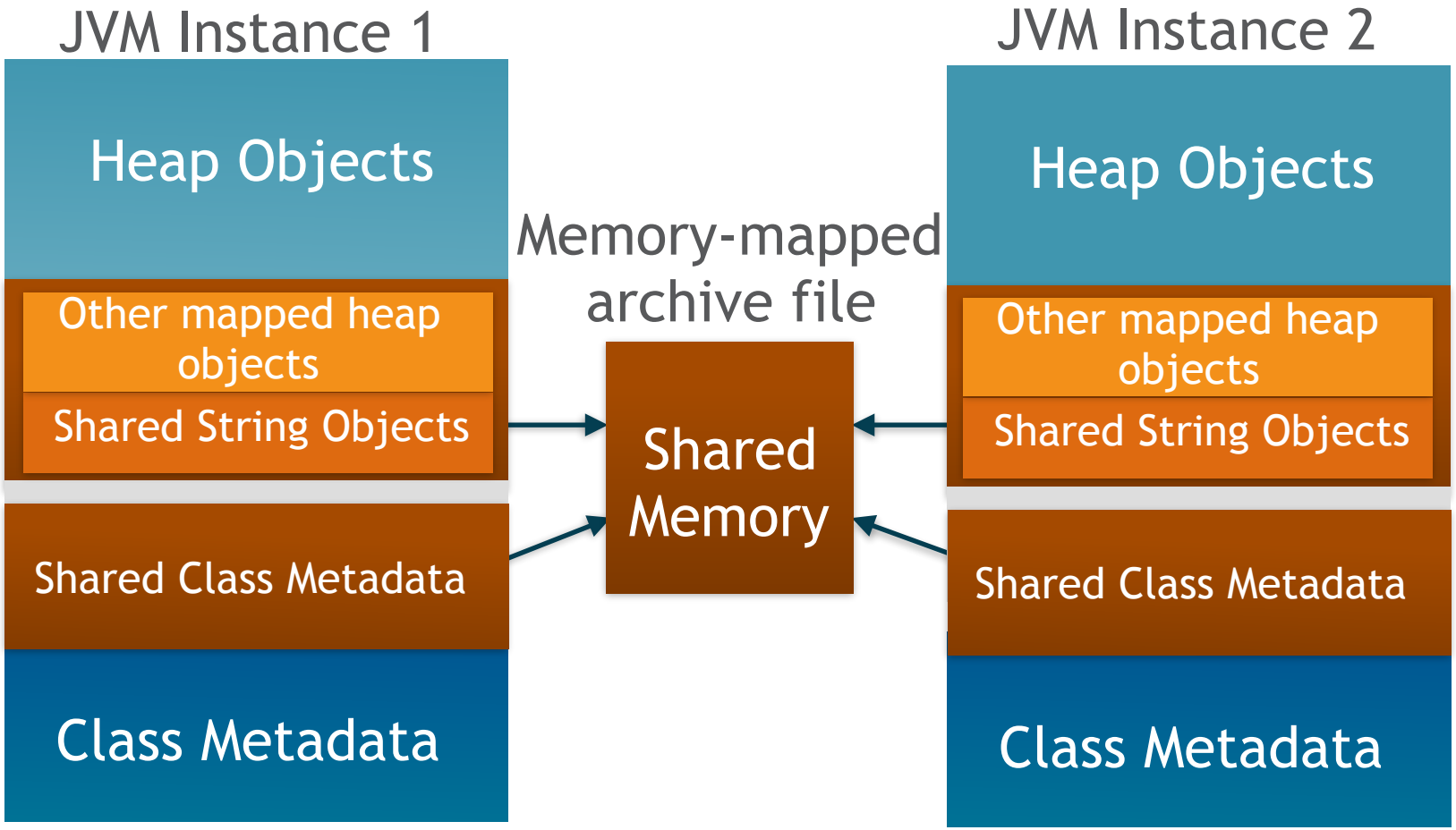
Archiving and Sharing Class Metadata and Java Objects in HotSpot JVM

Why Class Data Sharing(CDS)?

- The problem — loading Java classes
  - They take a long time to load — slow startup
  - They take up memory space

- The solution
  - Load the classes once and save into an “archive”
    - The archive is quickly memory mapped for all future app launches
  - Part of the memory map is shared by multiple JVM processes
    - Reduces footprint and improves startup time

CDS Explained



Archiving Class Metadata and Java Objects

- CDS dump time process
  - Preload a set of classes
  - Class files are parsed into HotSpot JVM internal representations (class metadata)
  - Class metadata is split into different parts (read-only versus read-write)
    - Copied into separate archive spaces
  - Selected Java objects are copied into the archive heap regions
  - All archived class metadata and Java heap data are written into the shared archive

CDS Archive Spaces

- Read-only spaces
  - RO: read-only metadata
  - OD: optional data
- GC read-only space
  - ST: closed archive heap region
    - Shared java.lang.String objects

- Read-write spaces
  - RW: read-write metadata
  - MC: misc code
  - MD: misc data
  - OA: open archive heap region

```
mc  space:     21712 [  0.1% of total] out of     24576 bytes [ 88.3% used] at 0x0000000800000000
rw  space:   4287008 [ 22.6% of total] out of   4288512 bytes [100.0% used] at 0x0000000800006000
ro  space:   7379720 [ 39.0% of total] out of   7380992 bytes [100.0% used] at 0x000000080041d000
md  space:      2752 [  0.0% of total] out of      4096 bytes [ 67.2% used] at 0x0000000800b27000
od  space:   6508464 [ 34.4% of total] out of   6508544 bytes [100.0% used] at 0x0000000800b28000
st0 space:    438272 [  2.3% of total] out of    438272 bytes [100.0% used] at 0x00000007ffc00000
oa0 space:    286720 [  1.5% of total] out of    286720 bytes [100.0% used] at 0x00000007ff800000
total    : 18924648 [100.0% of total] out of  18931712 bytes [100.0% used]
```

CDS Runtime Process in Subsequent Executions

Sharing Archived Class Metadata and Java Objects

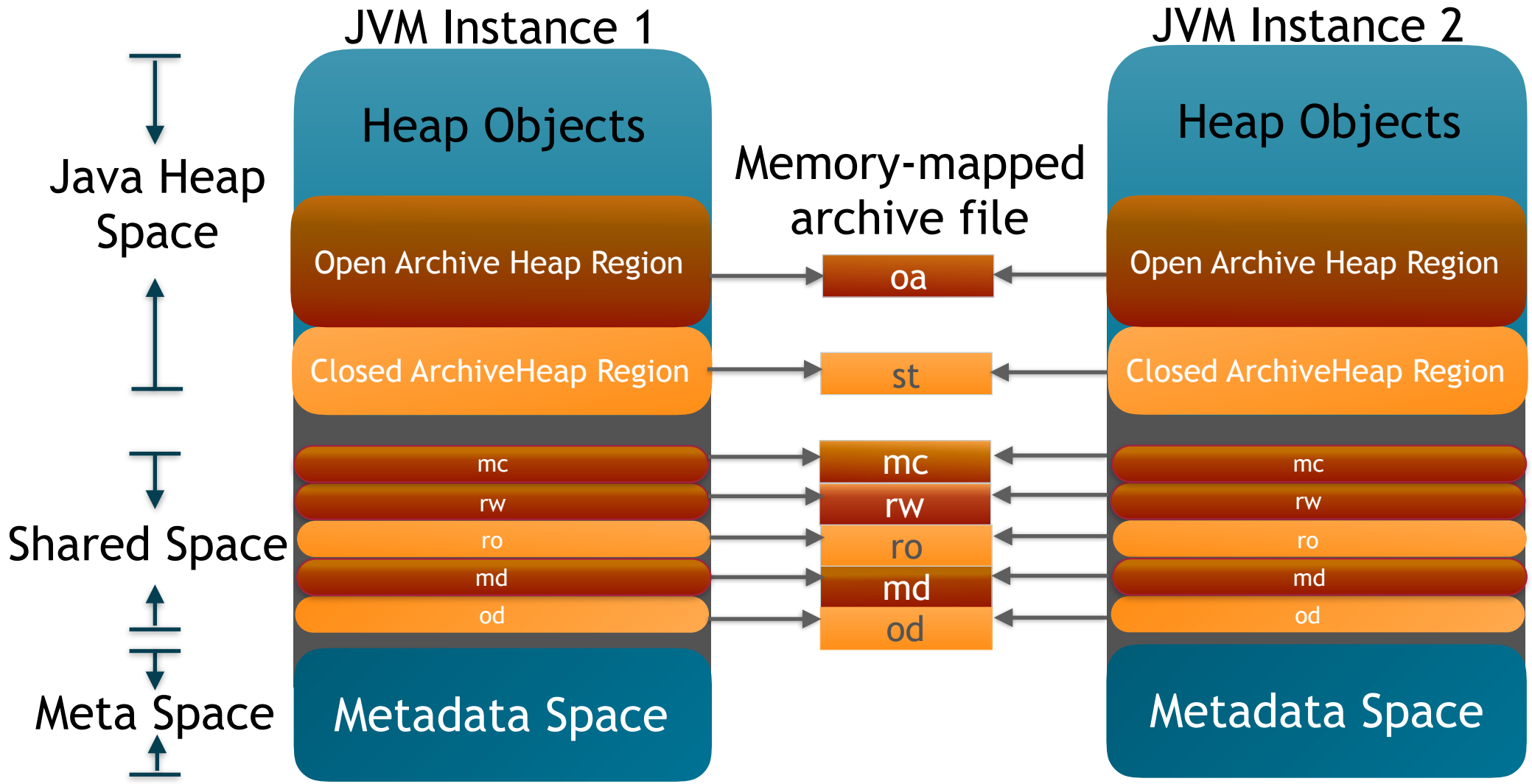
- Archived class metadata is memory mapped
- Archived Java heap data is mapped into runtime Java heap
- The mapped RO pages are shared among multiple HotSpot JVM processes
  - RO, OD
- The mapped RW pages are shared copy-on-write
  - ST: GC read-only
  - MC, RW, MD, OA

CDS Runtime Process (Continued)

Sharing Archived Class Metadata and Java Objects

- The JVM can lookup classes from the mapped data without searching/reading/parsing the class files from JAR files
  - Mapped class metadata can be used by the Hotspot JVM directly with minimal processing
- Mapped Java objects are used directly

Class Data Sharing Architecture



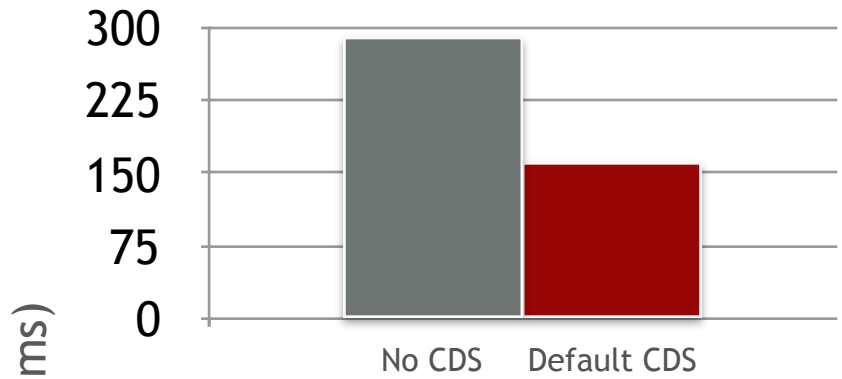
Default CDS Archive - JEP 341

- EA access available in JDK 12 now!
- Packaged with downloadable OpenJDK binaries
- Improve out-of-box Java startup time
  - No dumping step needed to benefit from CDS
  - No tuning required
- Optimized for cloud environment
  - Example, function as service
    - Small Java heap

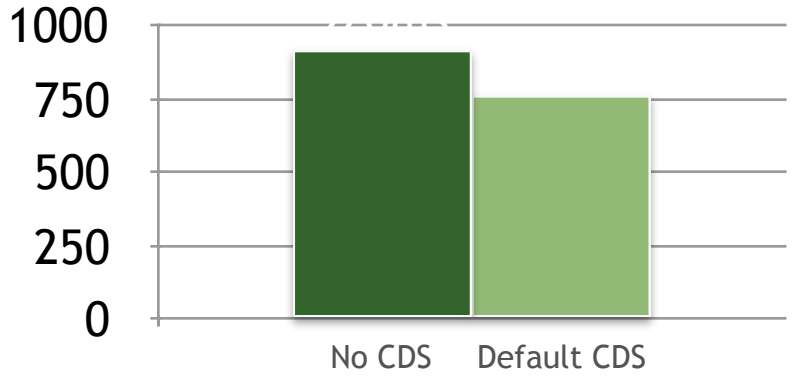
- Generated at JDK build time
  - Binary compatible with each JDK release
  - Created using G1 with 128M Java heap
- Works with different GCs and Java heap sizes at runtime
  - G1 supports archived Java heap objects
- https://openjdk.java.net/jeps/341

JDK 12 Startup Time With Default CDS Archives

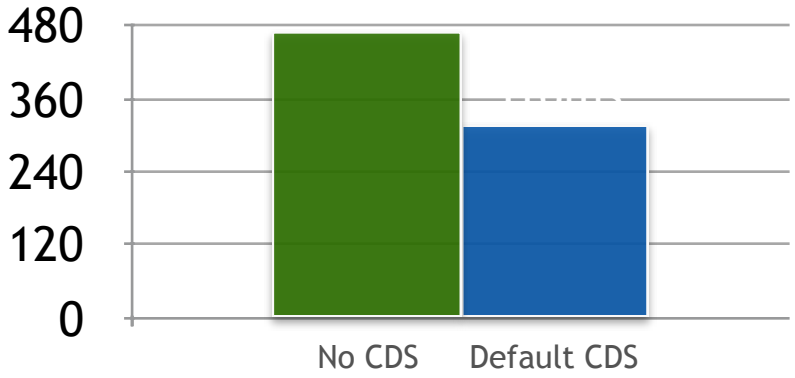
Noop: 45% improvement



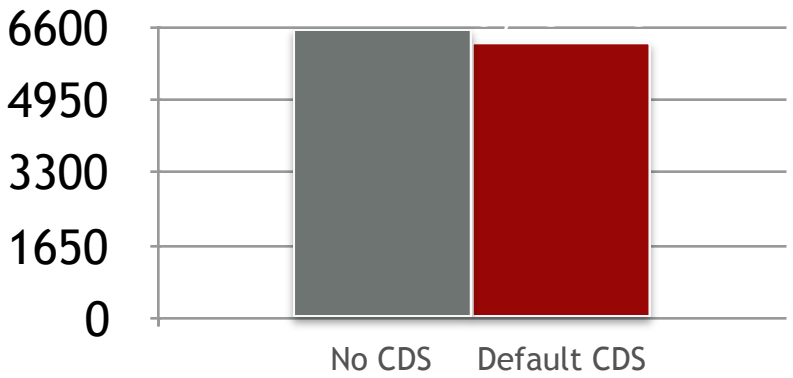
Jetty: 17% improvement



Fn Hello: 32.52% improvement



SpringBoot: 5.2% improvement



-Xmx128m
Intel(R) Xeon(R)
CPU E5-2690 0 @ 2.90GHz
Oracle Linux 6.4

Using CDS & Default CDS Archives

- CDS = archive system classes only
- Default CDS archive in JDK 12
  - No special command line option required, **just run Java**
- In JDK 11 and older releases
  - Step 1: dump - *java -Xshare:dump*
  - Step 2: run - *java -Xshare:auto [options …]*
  - -Xshare:auto is enabled by default in JDK 11 and can be omitted

Using AppCDS

- AppCDS = archive system classes + application classes
- JDK 11 & 12
  - Step 1: trial run - *java -XX:DumpLoadedClassList=<classlist> [options …]*
  - Step 2: dump - *java -Xshare:dump -XX:SharedArchiveFile=<archive> [options …]*
  - Step 3: run - *java -Xshare:auto -XX:SharedArchiveFile=<archive> [options]*
  - -Xshare:auto is enabled by default and can be omitted
  - Recommend using the same Java heap size for both dump time and runtime with G1 GC
    - Better startup time performance and more memory sharing

What's New Since JDK 9

Our Focus Areas

- Ease of use
- Improve Java startup time
- Reduce runtime footprint

JDK 9

- Generate the default classlist at JDK build-time
- Startup time improvement
  - Closed archive heap region in G1
    - Archive shareable Java objects: Shared Strings
  - Experimental support for customized class loader
- Footprint
  - Shared Strings
  - Compact symbol table layout in shared archive
  - Support archiving extra shared symbols specified in the configuration file

JDK 10

- Open source AppCDS
- Decouple archive dumping process from the boot loader
- Startup time improvement
  - Open archive heap region in G1
    - Archive general Java heap objects
      - Constant pool resolved_references arrays for archived classes
  - Pre-resolve all CONSTANT_String_info to java.lang.String objects
    - Archive all resolved strings
- Footprint
  - Make constant pool read-only and move to RO space

JDK 11

- Streamline and simplify CDS and AppCDS usage
  - Make UseAppCDS option obsolete
    - No -XX:+UseAppCDS is needed to use AppCDS feature
  - CDS/AppCDS behavior is determined by the classlist and archive content
    - Automatic and transparent process
  - Make -Xshare:auto the default for -server VM
- Support archiving classes from —module-path
- Startup time improvement
  - Support archiving java.lang.Class objects (mirrors)

JDK 12

- Startup time improvement
  - Default CDS archive with OpenJDK binary distributions
  - Support targeted Java object sub-graph archiving
    - Use open archive heap region in G1
    - Support core library
      - System module objects
      - Boot layer configuration
      - IntegerCache

Module Support

Module Support

- Module classes can be archived and shared at runtime
  - Boot/Platform/Application module classes in the runtime image
  - Application module classes in --module-path
- Support modular JAR only
  - Exploded module is not supported
  - A non-empty directory in the module path causes a fatal error
- No strict module path check
  - Dump time and runtime module paths can be different
  - Removing JAR(s) from module path does not invalidate existing archive
  - Module classes are filtered at runtime
    - Shared class runtime visibility check

Module Path Supporting Matrix

Options	-Xshare:dump	-Xshare:{on,auto}
--module-path	Y	Y
--module	Y	Y
--add-module	Y	Y
--upgrade-module-path	N	CDS is disabled
--patch-module	N	CDS is disabled
--limit-modules	N	CDS is disabled

- Any valid combinations of -cp and --module-path are supported
- --upgrade-module-path, --patch-module, or --limit-modules cause runtime warning:

```
Java HotSpot(TM) 64-Bit Server VM
warning: CDS is disabled when the --limit-modules option is specified.
```

Command Line Examples

- Create a shared archive using —module-path
  - *$ java -Xshare:dump -XX:SharedClassListFile=<class list file> *
    *-XX:SharedArchiveFile=<shared archive file> *
    --module-path=<path to modular jar> -m <module name>
- Run with shared archive using —module-path
  - *$ java -XX:SharedArchiveFile=<shared archive file> *
    --module-path=<path to modular jar> -m <module name>

Archive Java Heap Objects

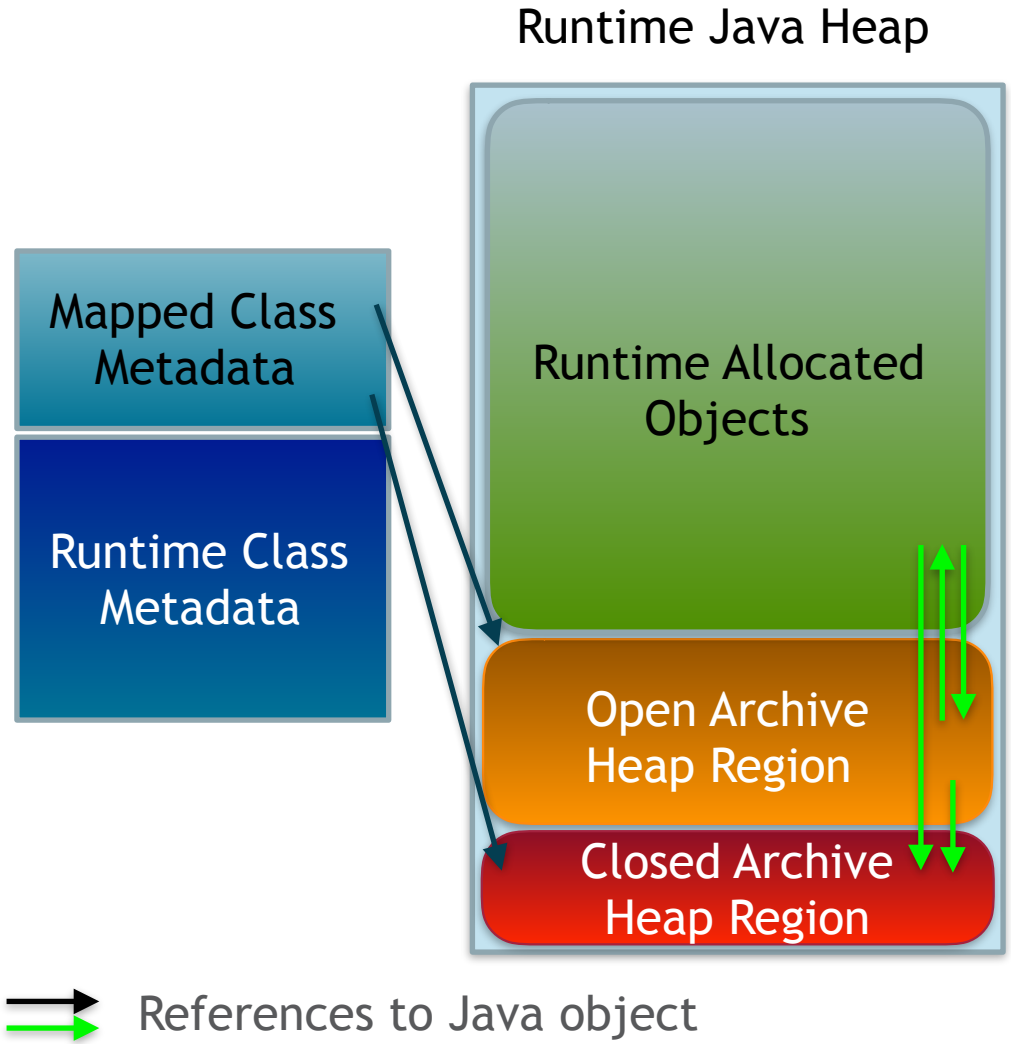
Supports G1 GC on 64-bit Non-Windows Platforms

Archive Java Heap Objects

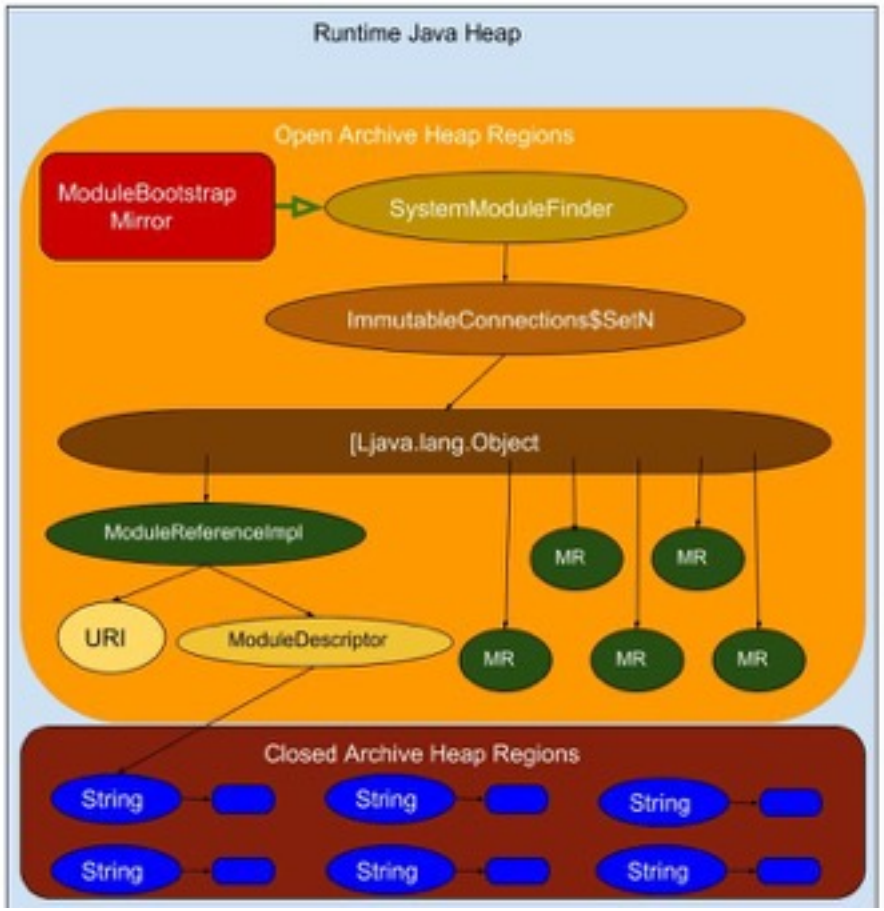
- Support G1 GC
- Selected Java objects are archived at dump time
- Archived objects are pinned
- Support different runtime Java heap sizes
  - Relocation is done automatically when necessary
    - Mapped Java heap regions are not within the runtime heap range
    - Compressed oop encoding mode is different
- Improve startup time and runtime memory footprint

Archive Heap Regions

- Closed archive heap region
  - GC does not mark objects and does not follow references
  - Archive special types of object only
    - String, primitive array, etc
  - Mapped memory can be shared between different JVMs at runtime
    - Relocation disables sharing
    - Synchronizing on archived objects can write into the memory page and disable sharing
- Open archive heap region
  - GC marks objects in the region and follows references
  - Archive general object



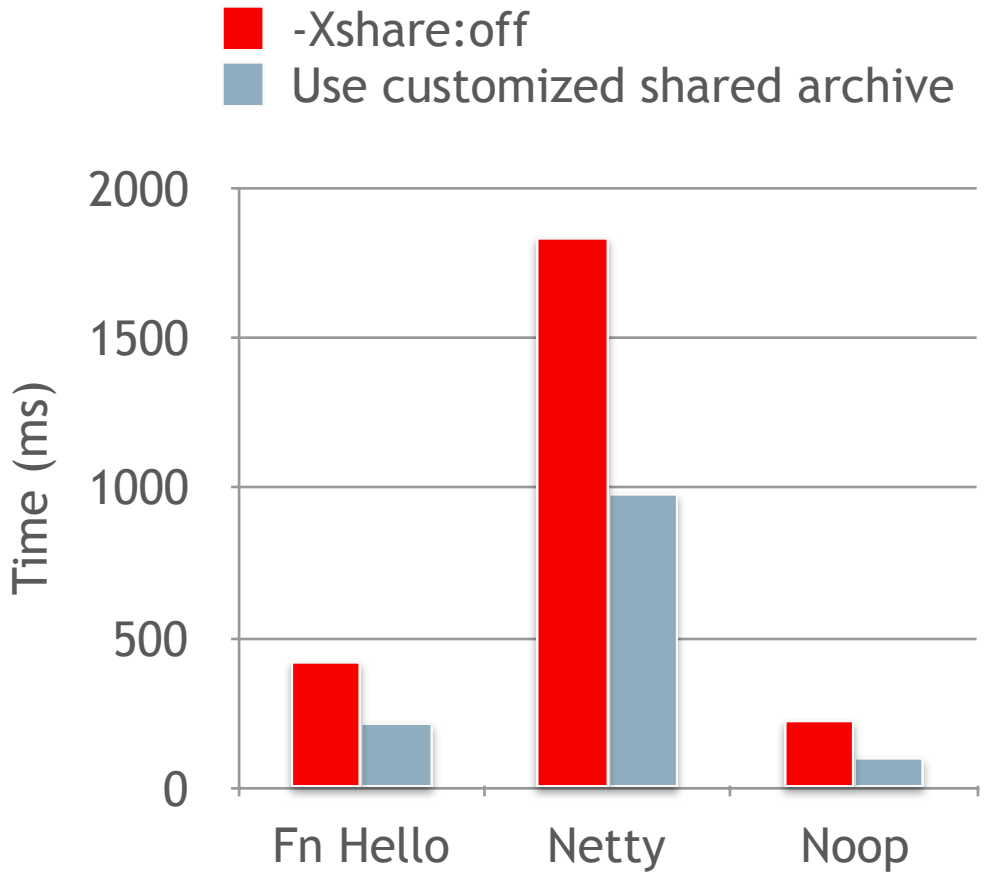
Archive Targeted Java Object Subgraphs



- Support core library only
- Restrictions
  - Objects are created at JVM initialization time
  - Immutable after initialization
  - No runtime context dependency
- Subgraph entry points
  - Static reference fields
  - Registered with CDS at dump time
  - Archive complete and closed subgraphs from entry points
- Reduce startup time by bypassing byte code execution for creating & initializing the objects at runtime

Startup Time and Footprint Results

JDK12 AppCDS Startup Time Results



G1 -Xmx128m
Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz
8 core 20480 KB cache
Oracle Linux 6.4

Application	Fn Hello	Netty	Noop
Startup Time Improvement %	49.04%	46.44%	54.04%

CDS/AppCDS & Containers

- CDS archive can be shared by containers on the same host
  - Put archive in the “parent” docker image, and create a “child” image for each containerized app
- Example: Two Java EE App Server containers on linux-x64
  - About 10,000 shared classes
  - No CDS: 1117.98 MiB
  - With CDS: 930.45 MB

Features In Incubation

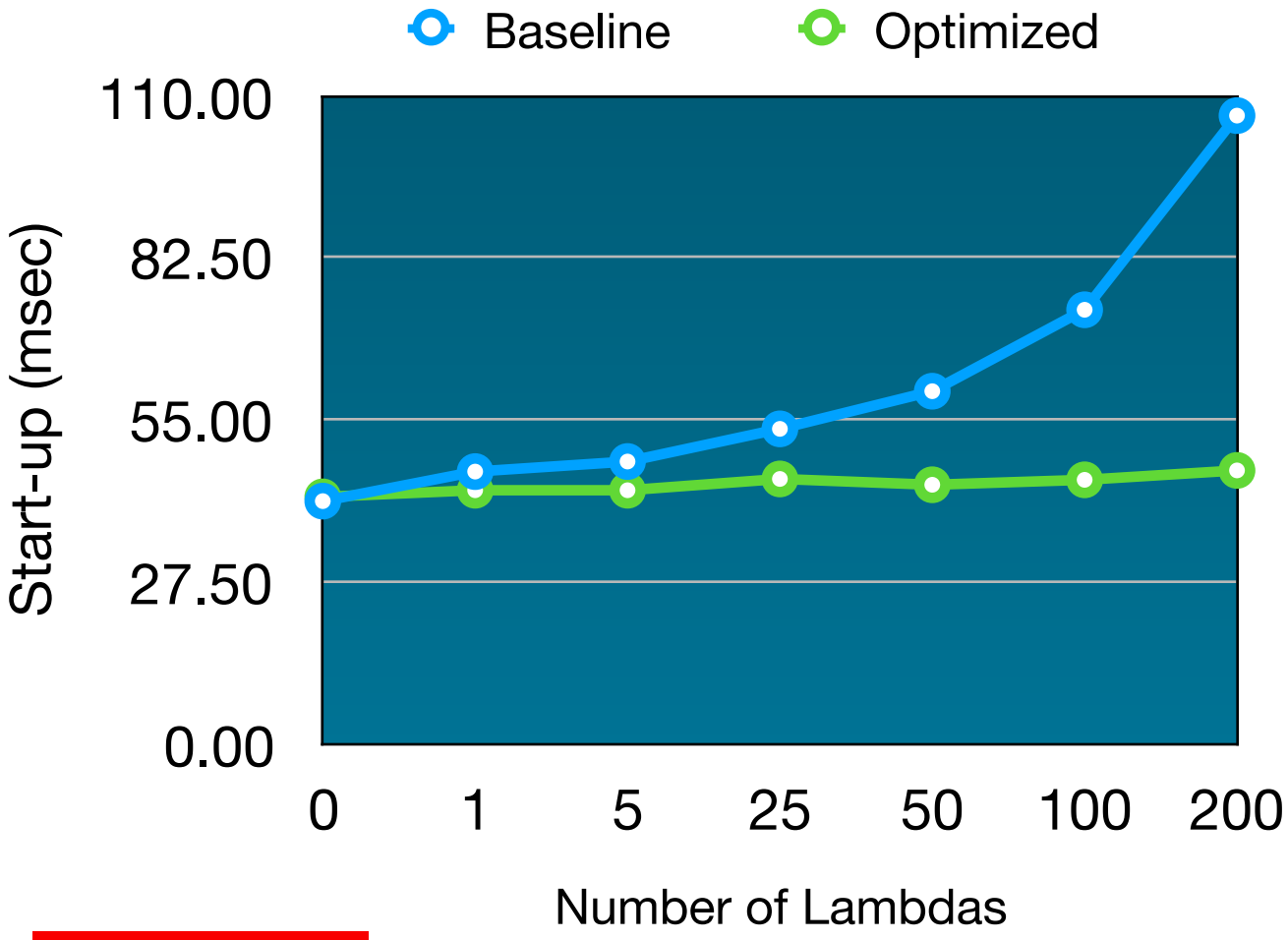
CDS & Lambda

- Lambda expressions are implemented with dynamically generated classes
- One or more generated classes for each -> in this example code
- This gives flexibility, at the expense of slow start-up time.

```
int result = values.stream()
   .filter(e -> e > 3)
   .filter(e -> e % 2 == 0)
   .map(e -> e * 2)
   .findFirst()
   .orElse(0);
```

- Solution: store the generated classes in CDS

CDS & Lambda (Continued)



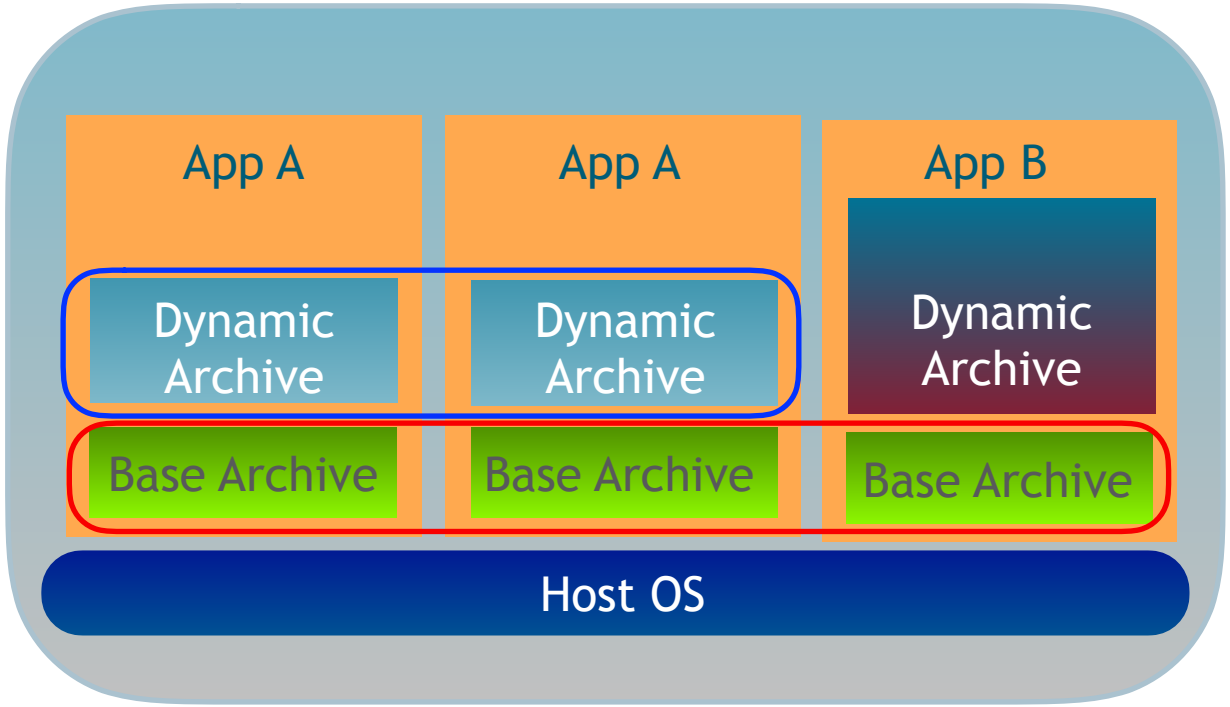
Cost per Lambda

Baseline: 0.329ms

Optimized: 0.024ms

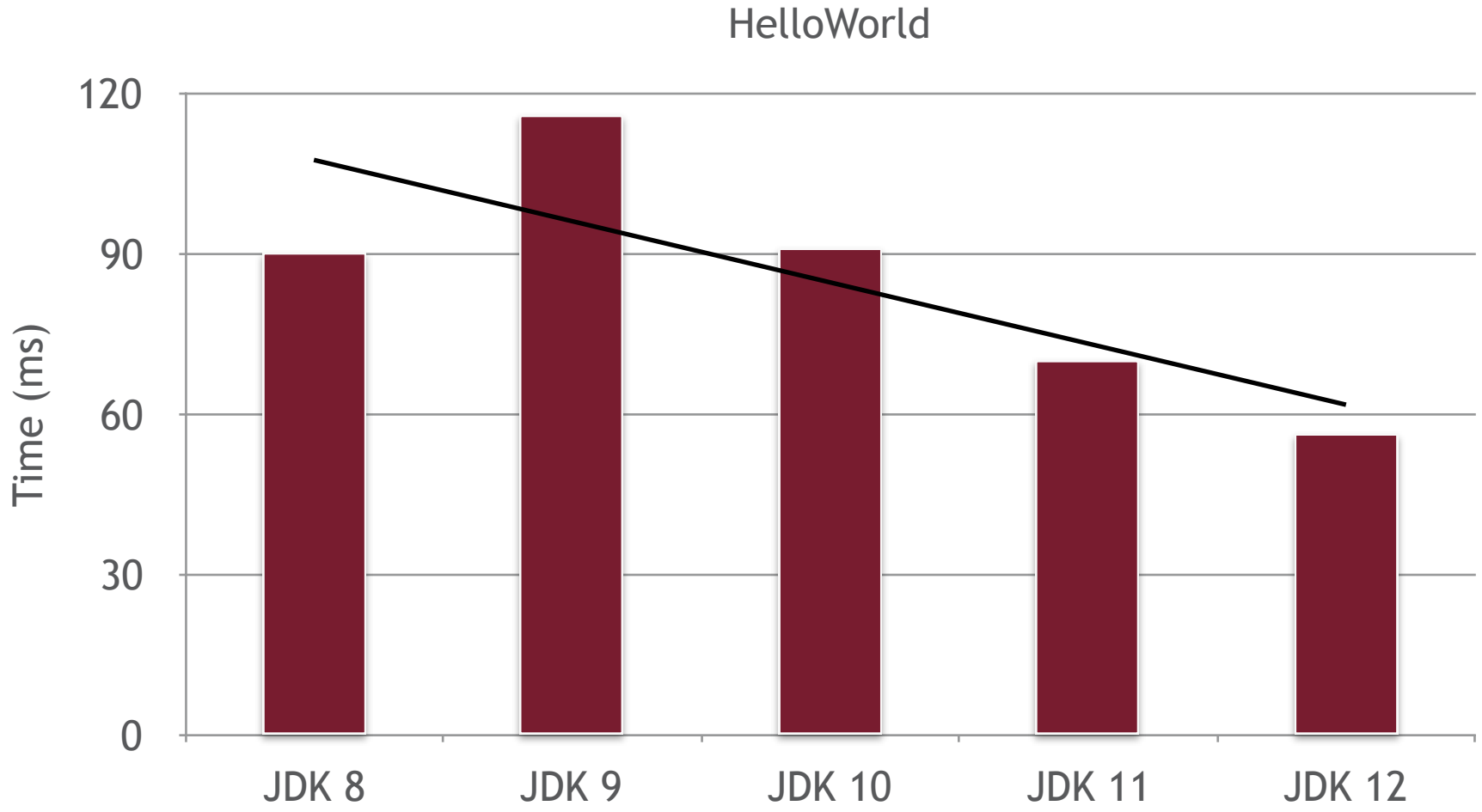
Hybrid Archiving With Layers

- Ease of use
  - Eliminate trial runs
- Static archiving at JDK build time
  - Base layer
  - Package with JDK, default CDS archive
  - Shared across different applications running in multiple JVM processes
- Dynamic archiving at application runtime
  - Classes used by application are archived dynamically
    - Application classes
    - Referenced system classes
  - On top of the base layer



Conclusion

History and Future: JDK Startup Time Trend



-Xshare:auto -Xmx32m
Intel(R) Xeon(R)
ES-2630 @ 2.40GHz
Ubuntu 16.04.3 LTS

Q & A