
ORACLE®

JavaOne™

ORACLE

Leveraging Application Class Data Sharing to Optimize Startup Time and Memory Footprint in Cloud Environments

Ioi Lam, Jiangli Zhou
Software Developer
Java Hotspot Virtual Machine, Oracle
Sept. 18, 2016

Java Your Next (Cloud)

JavaOne 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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

1. What is Class Data Sharing
2. Why Application Class Data Sharing?
3. Architecture Overview
4. Improved Startup Time and Reduced Memory Footprint
5. How to Use Application Class Data Sharing
6. Deep Dive

Class Data Sharing Overview

- Goal
  - Improve startup time
  - Reduce runtime memory footprint
- Class Data Sharing (CDS) approach
  - Dump time process
    - Preload a set of classes
    - Classes are parsed into their Java VM private internal representations (class metadata)
    - Metadata is split into read-only (RO) and read-write (RW) parts, and allocated in separate memory regions
    - All loaded class metadata is saved to a file (the shared archive)

Class Data Sharing Overview (continued)

- CDS runtime process
  - Shared archive is memory-mapped into the JVM runtime address space in subsequent JVM executions
  - The mapped RO pages are shared among multiple JVM processes, the mapped RW pages are shared copy-on-write
  - JVM can lookup classes from the mapped memory region without searching/reading/ parsing the class files from JAR files
  - Mapped class metadata can be used by JVM directly with minimal processing
- Benefits
  - Reduce the cost of class loading for archived classes at runtime and improves startup time
  - Save runtime memory usage by sharing among different JVM processes

Agenda

1. What is Class Data Sharing
2. **Why Application Class Data Sharing?**
3. Architecture Overview
4. Improved Startup Time and Reduced Memory Footprint
5. How to Use Application Class Data Sharing
6. Deep Dive

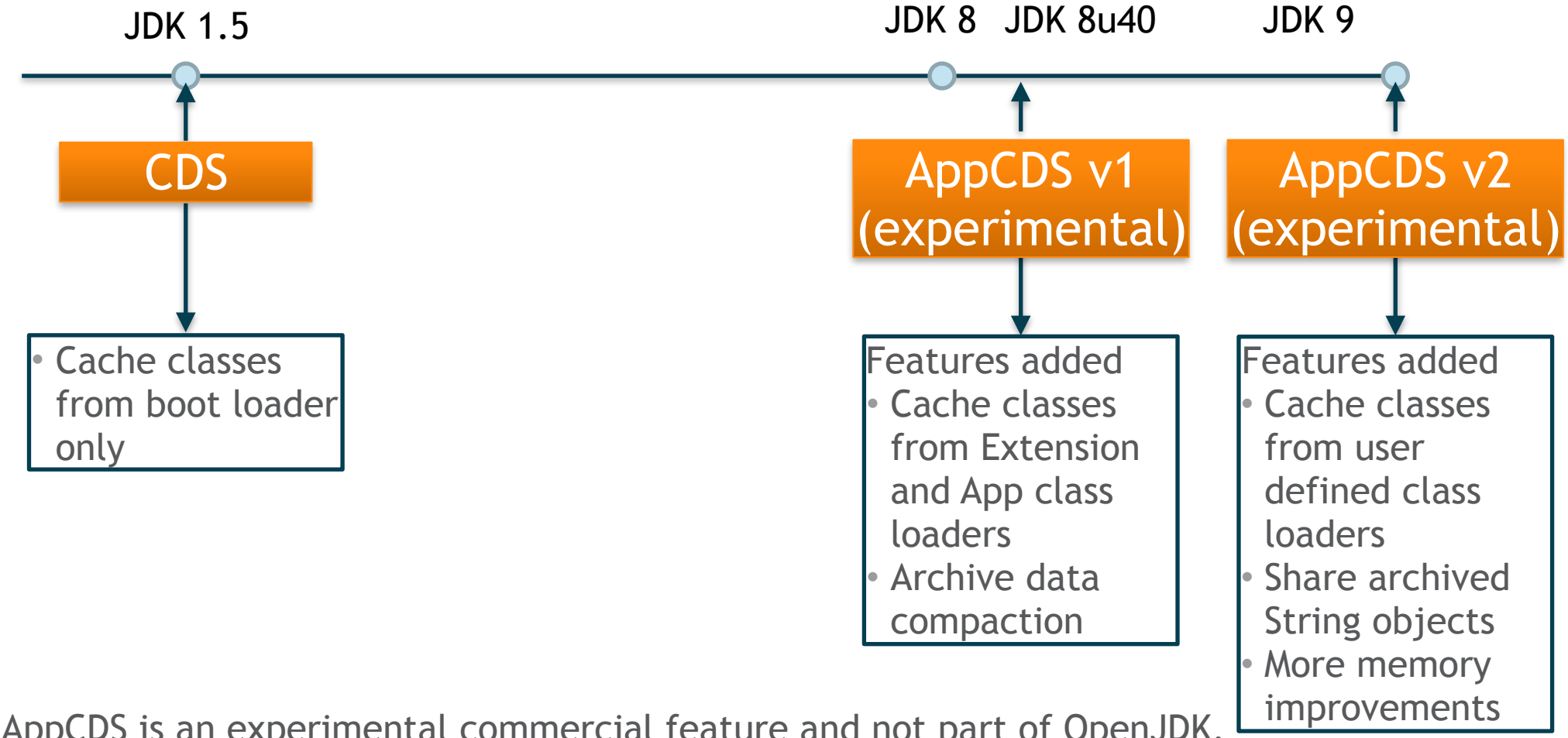
Application Class Data Sharing (AppCDS) Enhancements

- CDS has evolved since JDK 8u40
  - Before JDK 8u40
    - Only supported core library classes loaded by the bootstrap class loader
  - JDK 8u40 & JDK 9
    - Application classes are supported
    - Classes loaded by all class loaders can be archived, including
      - Bootstrap class loader (NULL class loader)
      - PlatformClassLoader (the ExtensionClassLoader in JDK 8 and earlier versions)
      - AppClassLoader (the default system class loader)
      - User defined class loaders

AppCDS Enhancements (continued)

- Other improvements in JDK 8u40 & JDK 9
  - Allow archiving and sharing interned java.lang.String objects
  - Provide further reduction in storage for archived class metadata

A Quick Look at CDS/AppCDS Evolution



NOTE: AppCDS is an experimental commercial feature and not part of OpenJDK.

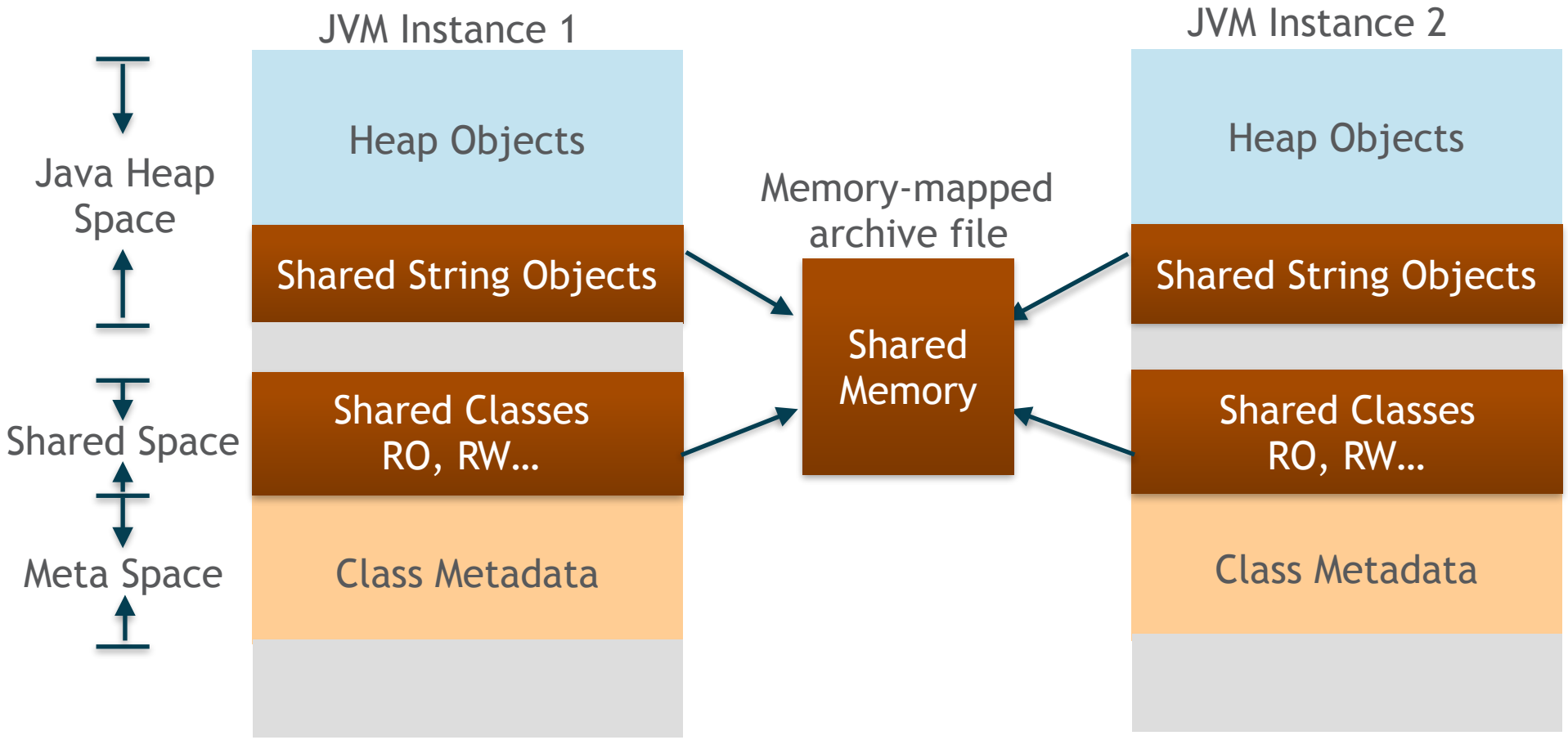
AppCDS Can Reduce The Cost of Your Cloud!

- The cloud computing shift
  - Cloud based platforms such as Oracle Java Cloud Service using Oracle WebLogic Server runs services on clusters of JVM instances.
  - The shift to microservices in software development allows rapid implementation of complex systems using micro-sized functional subsystems with isolation and interoperability.
- AppCDS can help!

Agenda

1. What is Class Data Sharing
2. Why Application Class Data Sharing?
3. **Architecture Overview**
4. Improved Startup Time and Memory Footprint
5. How to Use Application Class Data Sharing
6. Deep Dive

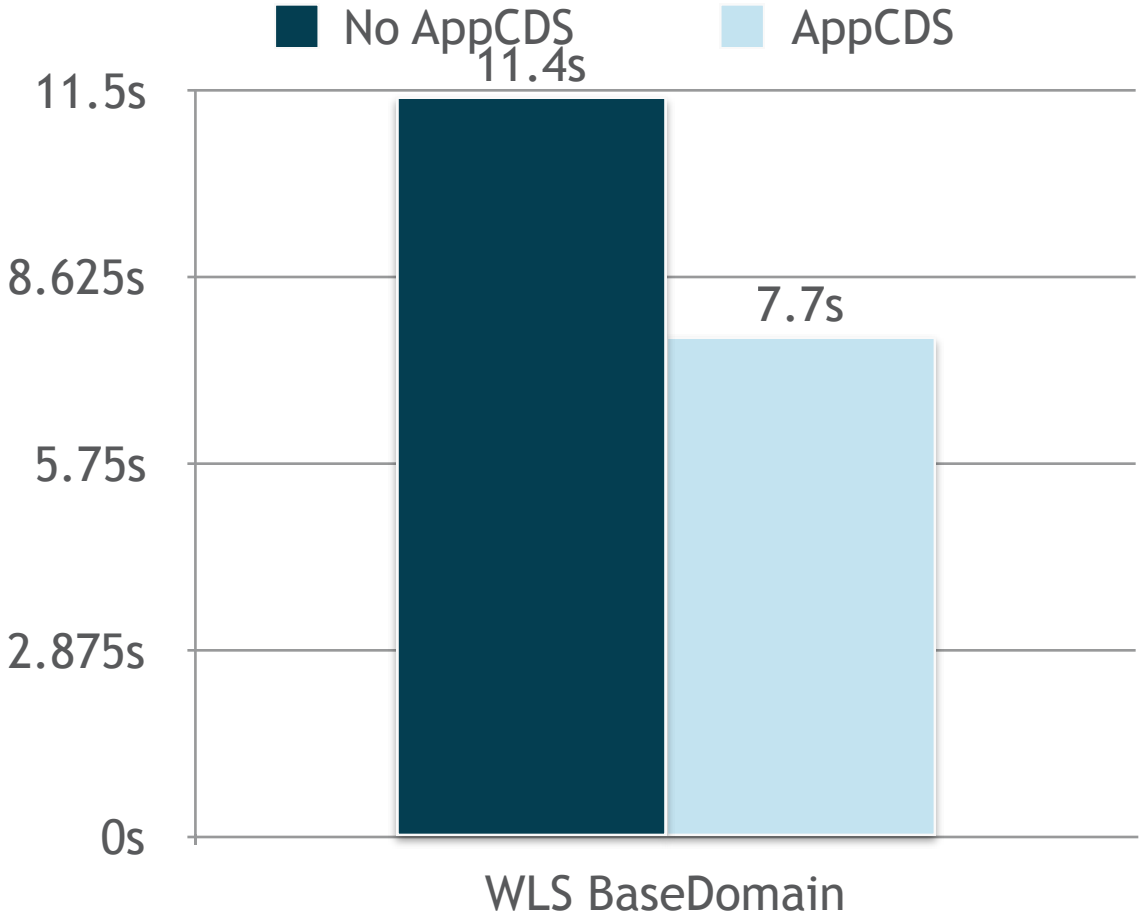
AppCDS Architectural View



Agenda

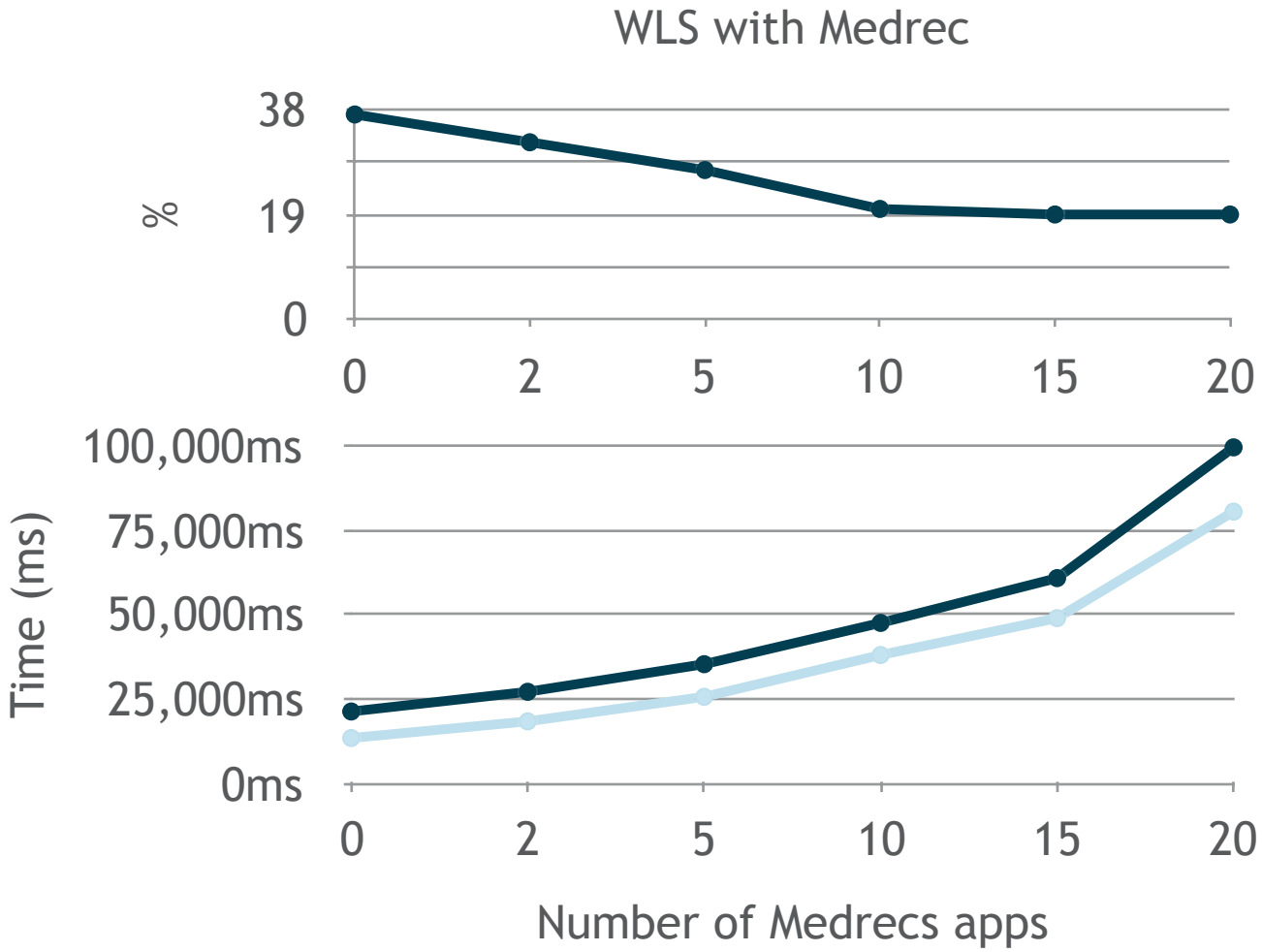
1. What is Class Data Sharing
2. Why Application Class Data Sharing?
3. Architecture Overview
4. Improved Startup Time and Reduced Memory Footprint
5. How to Use Application Class Data Sharing
6. Deep Dive

Startup Time Improvements for Sample Oracle WebLogic Server Environment



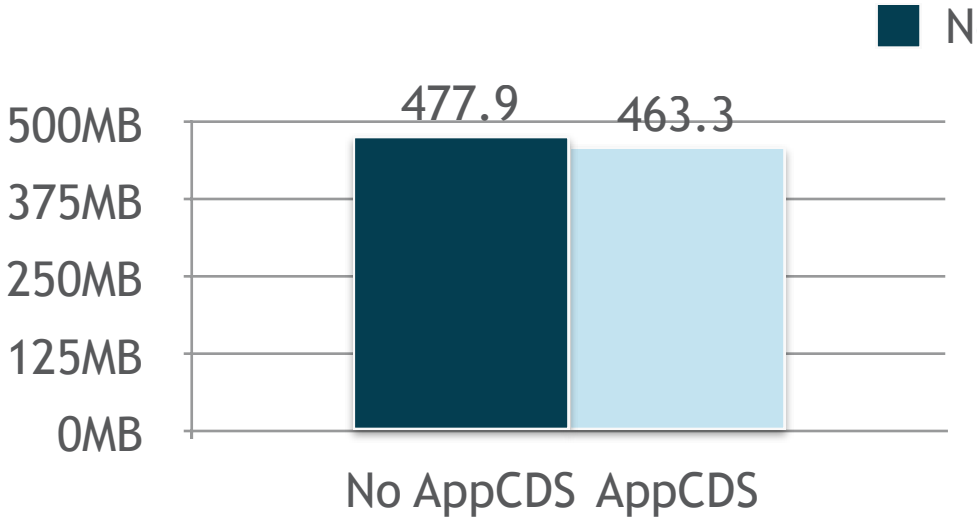
- About 30% startup time improvement observed with AppCDS for Oracle WebLogic Server (WLS) base domain

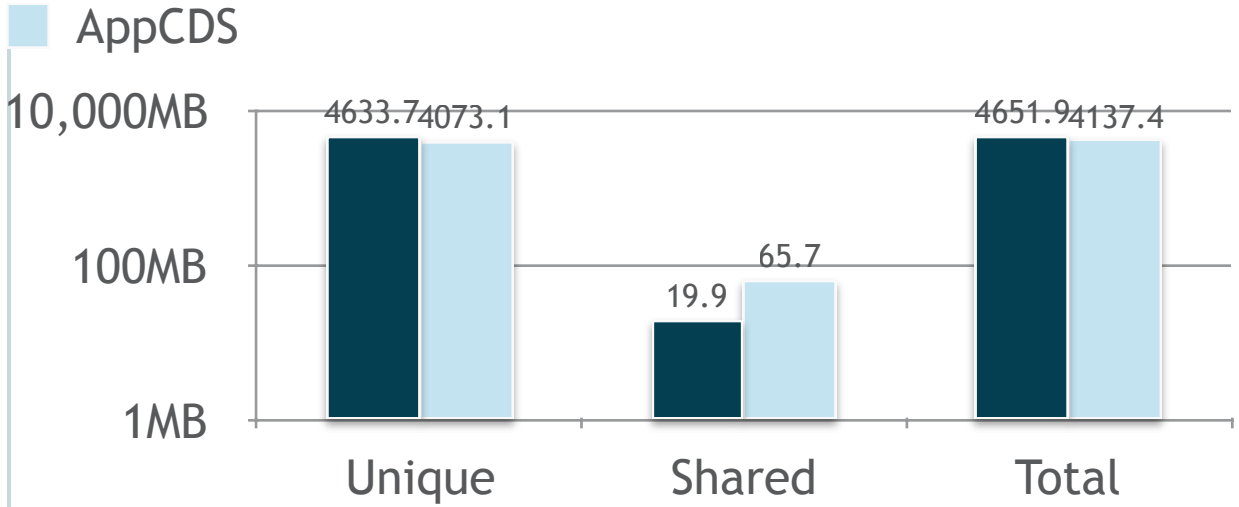
Startup Time Improvements for Sample Oracle WebLogic Server Environment (continued)



- Single JVM instance is used in the measurement
- 19% ~ 37% startup improvements with AppCDS running 20 to 0 Medrecs apps in WLS

Memory Footprint Saving for Sample Oracle WebLogic Server Environment





- With single JVM instance running WLS base domain, using AppCDS improves memory footprint
- Memory footprint improved by optimal layout and compaction of archived data

- With 10 JVM instances running WLS base domain simultaneously, shared memory increases
- There is 11.06% saving in total memory footprint with AppCDS

Memory Footprint Savings for Sample Oracle WebLogic Server Environment (continued)

	Common Domain		
	No AppCDS (kb)	AppCDS (kb)	%
JVM Process 1	1705243	1662706	2.49
JVM Process 2	9743548	9643919	1.02
JVM Process 3	2061671	1933873	6.20
JVM Process 4	2261076	2053933	9.16
JVM Process 5	2577195	2345621	8.99
JVM Process 6	2086027	1982791	4.95
JVM Process 7	3131469	2783527	11.11
JVM Process 8	1462726	1311244	10.36

	Procurement Domain		
	No AppCDS (kb)	AppCDS (kb)	%
JVM Process 1	1714699	1690194	1.43
JVM Process 2	1894248	1859755	1.82
JVM Process 3	2313387	2130547	7.90
JVM Process 4	3158520	3003468	4.91
JVM Process 5	1879472	1768085	5.93

- Average footprint savings: 5%

Startup Time and Memory Footprint Savings for Big Data Cluster Computing Systems

- Good startup time and memory footprint savings are also measurable with other real-world applications using AppCDS
  - Apache Spark big data framework with KMeans workload and 20 slaves (1 slave per process mode)
  - Average startup saving is 11% for each slave with AppCDS
  - Memory saving is 10% per slave on average with AppCDS

Agenda

1. What is Class Data Sharing
2. Why Application Class Data Sharing?
3. Architecture Overview
4. Improved Startup Time and Reduced Memory Footprint
5. How to Use Application Class Data Sharing
6. Deep Dive

How to Use AppCDS

- Create class list and archive config file with an application trial run
  - Run application with -Xshare:off -Xlog:classload=trace
  - Create class list from the classload log
  - Dump symbols and strings from the running application using jcmd
    - jcmd <pid> VM.symboltable -verbose
    - jcmd <pid> VM.stringtable -verbose
  - Create the archive config file using the dumped symbols and strings
- Create shared archive
  - -Xshare:dump -XX:+UnlockCommercialFeatures -XX:+UseAppCDS -XX:SharedArchiveFile=<jsa> -XX:SharedClassListFile=<classlist> -XX:SharedArchiveConfigFile=<config_file>

How to Use AppCDS (continued)

- Run application with the shared archive
  - -Xshare:on -XX:+UnlockCommercialFeatures -XX:+UseAppCDS -XX:SharedArchiveFile=<jsa>
- Link to JDK 8u40 Release Notes for AppCDS
  - http://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html#app_class_data_sharing
  - http://www.oracle.com/technetwork/java/javase/8u40-relnotes-2389089.html

Agenda

1. What is Class Data Sharing
2. Why Application Class Data Sharing?
3. Architecture Overview
4. Improved Startup Time and Reduced Memory Footprint
5. How to Use Application Class Data Sharing
6. Deep Dive

AppCDS for Built-in Class Loaders

- Feature added in JDK 8u40
- Classes for the boot loader, ExtensionClassLoader (PlatformClassLoader) and AppClassLoader can be archived/shared adhering to Java runtime semantics
  - Each archived class is tagged so it can be loaded by the correct class loader at runtime
    - ClassLoader delegation order is preserved
  - Inheritance and class hierarchy are not changed
    - Updates to JAR files invalidate the existing archive file
    - Runtime class paths are required to be compatible with dump time class paths
      - Class path orders are validated at runtime
      - Runtime class path can be only extended by appending additional path components to the end

AppCDS for Built-in Class Loaders (continued)

- Security
  - Pre-verification at dump-time
  - Runtime security checks
  - Fast runtime ProtectionDomain handling

AppCDS for User Defined Class Loaders

- Supported since JDK 9
- User defined class loader can work with AppCDS without changing its existing code
- Delegation and class file lookup are controlled by the user defined class loaders
  - No change to the user defined class loader behavior
- Validation via fingerprint match
  - Class bytes checksum(class fingerprint) is computed at dump time and saved with the archived class
  - Class fingerprint is checked against runtime recomputed value
    - Fingerprints must be valid for the class being loaded and all its super types, otherwise the class is not loaded from the archive

AppCDS for User Defined Class Loaders (continued)

- Runtime loading
  - JAR files on the class loader's path are searched
  - Class bytes are read from the JAR file
  - JVM looks up an archived class from mapped memory before defining the class
- New ClassList Format

```
java/lang/Object id: 0
Interface1 id: 1 super: 0 source: cust.jar
Interface2 id: 2 super: 0 source: cust.jar
ParentClass id: 3 super: 0 source: cust.jar
ChildClass id: 4 super: 3 interfaces: 1 2 source: cust.jar
```

- Benefit
  - Footprint saving due to sharing of archived class metadata (same as built-in class loaders)
  - Startup time saving due to pre-parsing, pre-verification and pre-layout of metadata

AppCDS for the Jigsaw Modular System

- Module system in JDK 9
  - Jigsaw provides modularity beyond the traditional JAR file and package concepts
  - Standard class-loading facilities using the class paths are extended to modules
    - Classes can be loaded from new locations, the modular runtime image, module path, etc
- AppCDS can archive classes from the modular runtime image in JDK 9
  - May support archiving classes in modules from other locations in the future
    - E.g. classes from the module path

AppCDS for the Jigsaw Modular System (continued)

- Class loader's visibility boundaries
  - Determined by the modules defined within the class loader
  - Decide whether a class may be loaded at runtime ("visibility rules")
- AppCDS respects the visibility boundaries when loading archived classes at runtime
  - Class loader type info for a module class from the runtime image is saved at dump time
  - The runtime visibility is checked when loading an archived class
    - Use the module information defined by the class loaders

Sharing String Objects in AppCDS

- Interned strings can be archived in JDK 9
  - Support G1 GC and compressed object & klass pointers on 64-bit platforms
  - java.lang.String objects referenced from the string table and the underlying value arrays are allocated in (or copied from original heap addresses) the ‘archive’ heap region at dump time
- Archived String objects are mapped into the Java heap at runtime
  - The string data is memory mapped at the same offset from the runtime java heap base
    - Not required to be mapped at the same memory address as the dump time location
  - Same compressed object & klass encodings are required for runtime and dump time

Sharing String Objects in AppCDS (continued)

- Mapped string data can be shared among multiple JVM instances at runtime
  - The objects in the mapped ‘archive’ heap region are ‘pinned’
    - GC does not write to the objects in the region
  - Hash values are precomputed and stored within the strings at dump time
    - Avoid writes to the objects at runtime for better sharing
  - Mapped as RW (synchronization on shared strings require writes to the objects)
- Extra strings can be added to the archived based on profiling results
  - -XX:SharedArchiveConfigFile=<config_file>
  - String section starts with @SECTION: String

Sharing Symbols (JVM Internal strings)

- Symbols are canonicalized strings including class name, method name and field name etc
- Symbols used by classes at dump time are archived
- Extra symbols can be added to the archive based on profiling results
  - -XX:SharedArchiveConfigFile=<config_file>
  - Symbol section starts with @SECTION: Symbol

Archive Config File in AppCDS

- Specify extra data for including in the shared archive
- Support symbols and strings currently
- Example of config file

```
VERSION: 1.0
@SECTION: String
0:
4: Java
10: StringLock
@SECTION: Symbol
0 -1:
19 -1: java/io/InputStream
27 -1: (C)Ljava/lang/StringBuffer;
```

Archive Data Compaction and Increased Sharability

- Shared symbol table and string table are compacted and read-only
  - For more info, please see https://bugs.openjdk.java.net/browse/JDK-8059510
- Use trampolines for interpreter and compiled invocation entries in archived methods
  - Avoids write to every archived method
  - For more info, please see https://bugs.openjdk.java.net/browse/JDK-8145221
- Special “nofast” opcodes in archived classes
  - The “nofast” opcode are not modified at runtime
  - For more info, please see https://bugs.openjdk.java.net/browse/JDK-8074345

Contact Information

- For licensing related questions, please contact:

  Aurelio Garcia-Ribeyro | Group Product Manager, Java Platform Group

  Phone +1 408.276.6529 | Mobile: +1 305.439.7223

  aurelio.garciaribeyro@Oracle.COM

- For technical related questions, please contact:

  Ioi Lam, ioi.lam@oracle.com

  Jiangli Zhou, jiangli.zhou@oracle.com

Q & A

Integrated Cloud

Applications & Platform Services

JavaOne™

ORACLE


ORACLE®