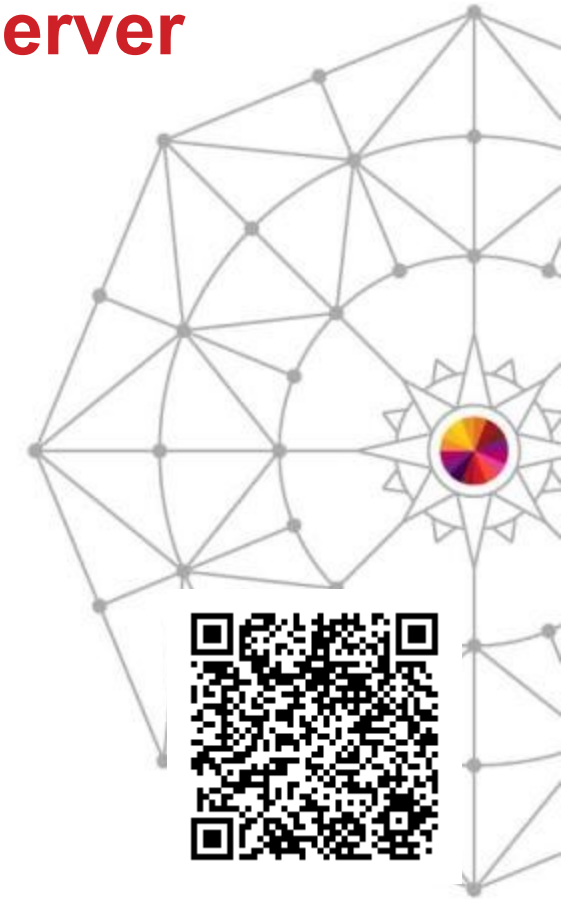
#SHAREorg

SHARE

CICS and Java: How the JVM Server Transforms Java in CICS

Ian J Mitchell,
IBM Distinguished Engineer,
AIM z CTO and CICS Portfolio Architect
IBM Hursley

Thursday 13th March
Session Number : 14922



SHARE in Anaheim

Abstract

CICS has for a long time provided a Java environment for application development. In recent releases of CICS the JVM Server has transformed CICS into a first-class hosting environment for Java. This session will provide a brief history of the development of the Java environment within CICS, followed by a detailed look at the capabilities offered by CICS version 4. In particular we will look at how the OSGi framework provides excellent lifecycle management of Java applications without having to restart the JVM Server, how Java application can be eligible for zAAP offload thereby reducing the cost of a transaction, and how the JVM Server supports multiple concurrent transactions, reducing the storage requirements and the need for multiple JVM instances in a single region.

Agenda

- JVM Options in CICS TS v4.2 and v5.1
  - JVM Pool
  - JVM Server
- 64 Bit JVM Support
- OSGi for application management
- WODM Rules Execution Engine

Overview of Java program support in CICS

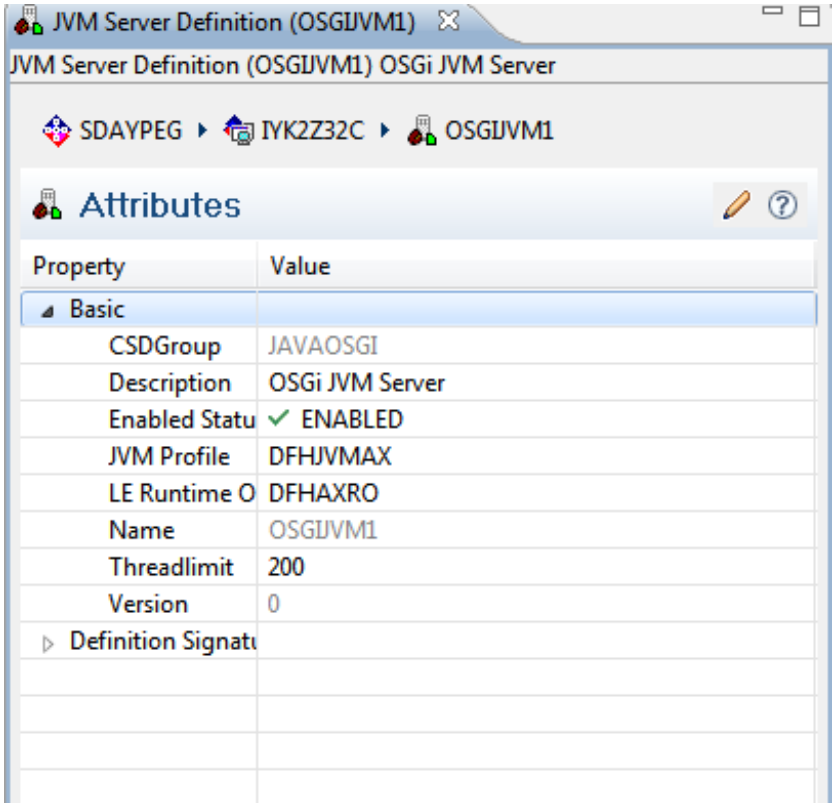
"Traditional" pooled JVMs

- Multiple JVMs in a CICS region
- Single-thread, program isolation
- J8 (CICS Key) or J9 (User key) TCBs
- MAXJVMTCBs in SIT
- No JVM definition except in JVM profile via PROGRAM
- EJB and CORBA support

"New" JVM servers

- Supports JCICS interfaces for CICS Java programs
- Can have multiple JVM Servers per region
- Multi-threaded, up to 256 parallel tasks
- Facilitates data-sharing between Java applications
- Industry-standard
- T8 TCBs
- JVMSERVER and PROGRAM definitions required
- Requires deployment as OSGi bundle within a CICS BUNDLE
- No EJB or CORBA support

Defining a JVM server



JVM Profile

- JVM profile in HFS in JVMPROFILEDR
- DFHJVMAX is default

LE Runtime Options

- LE storage options
- Defaults to DFHAXRO

Threadlimit

- Max number of T8 threads

Profile contents

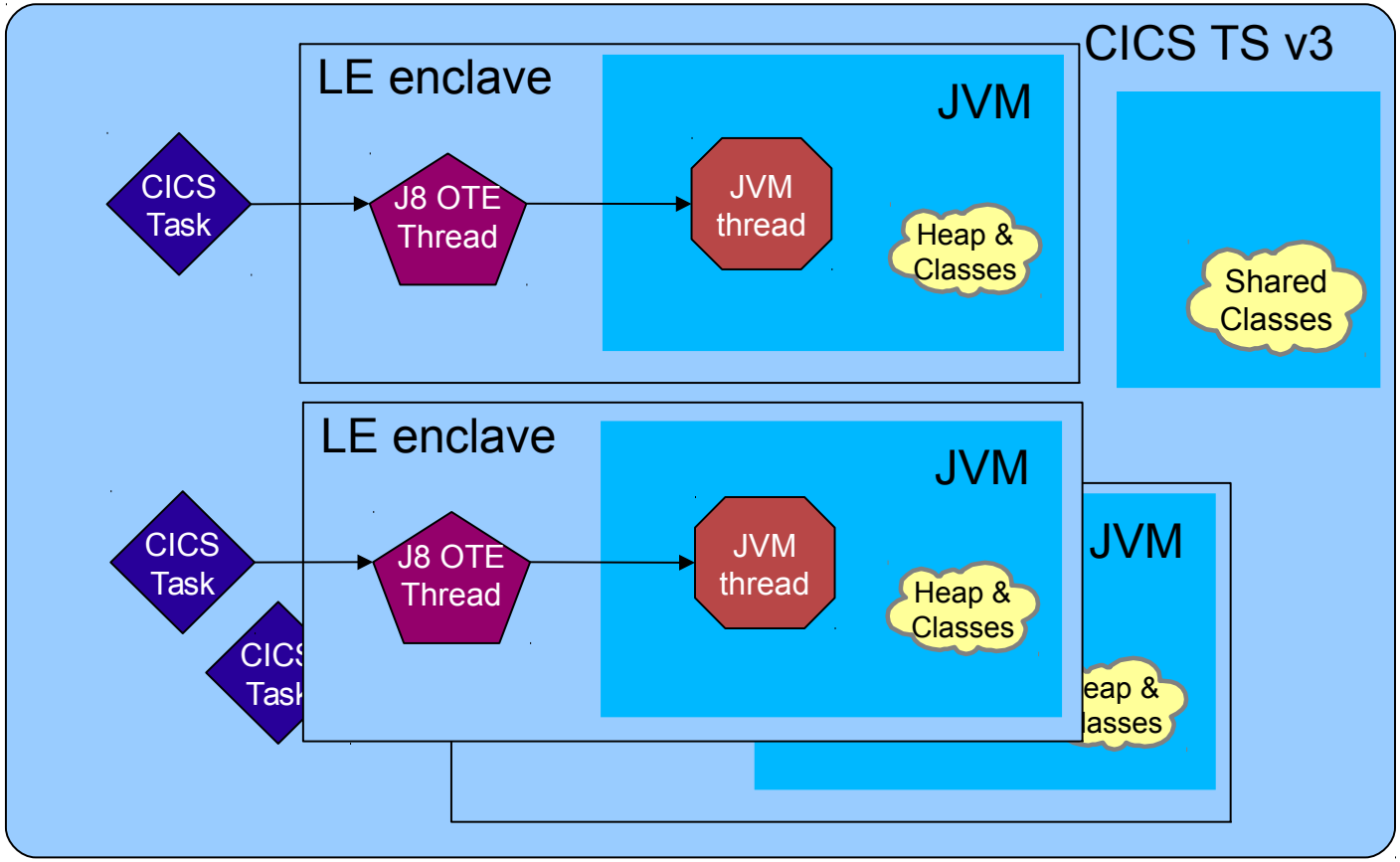
DFHWLP.jvmprofile | CICSB.jcl

```
################################################################################
# JVM profile: DFHWLP                                                          #
#                                                                              #
#   This sample CICS JVM profile is for a Liberty JVM server.                  #
#   The JVM server contains a configured instance of a web container           #
#   that is based on the Liberty profile technology.                           #
################################################################################
#
#                         Symbol Substitution
#                         -------------------
#
# The following substitutions are supported:
#   &USSHOME;     => The value of the USSHOME SIT parameter.
#   &CONFIGROOT; => the location of configuration files, such as the
#                     JVM profile.
#   &APPLID;      => The applid of the CICS region.
#   &JVMSERVER;   => The name of the JVMSERVER resource.
#   &DATE;        => Date the JVMSERVER is enabled.  Dyymmdd
#   &TIME;        => Time the JVMSERVER is enabled.  Thhmmss
#
# All variables must be delimited with & and ;
# Using substitutions means that you can use the same profile
# for multiple regions and still have unique working directories
# and output destinations for each region.
#
# With this substitution
#     ENV_VAR=myvar.&APPLID;.&JVMSERVER;.data
# becomes
#     ENV_VAR=myvar.ABCDEF.JSERVER1.data
# for a JVMSERVER resource with the name JSERVER1 in a CICS region
# with applid ABCDEF.
#
# Note: The continuation character for use with JVMProfiles is '\'.
#*******************************************************************************
#
#                         Required parameters
#                         -------------------
#
# JAVA_HOME specifies the location of the Java directory.
#
JAVA_HOME=/usr/lpp/java/J7.0_64/
#
# Set the current working directory. If this environment variable is
# set, a change to the specified directory is issued before the JVM
# is initialized, and the STDIN, STDOUT and STDERR streams are
# allocated to this directory
```

JVMPool Architecture - CICS TS v3 (and v2)

A single CICS task dispatched into a JVM in the pool at a time. So concurrent task count limited to the number of JVMs that can fit in the 31-bit address space.

Each JVM 'costs' ~20Mb plus the application heap value.

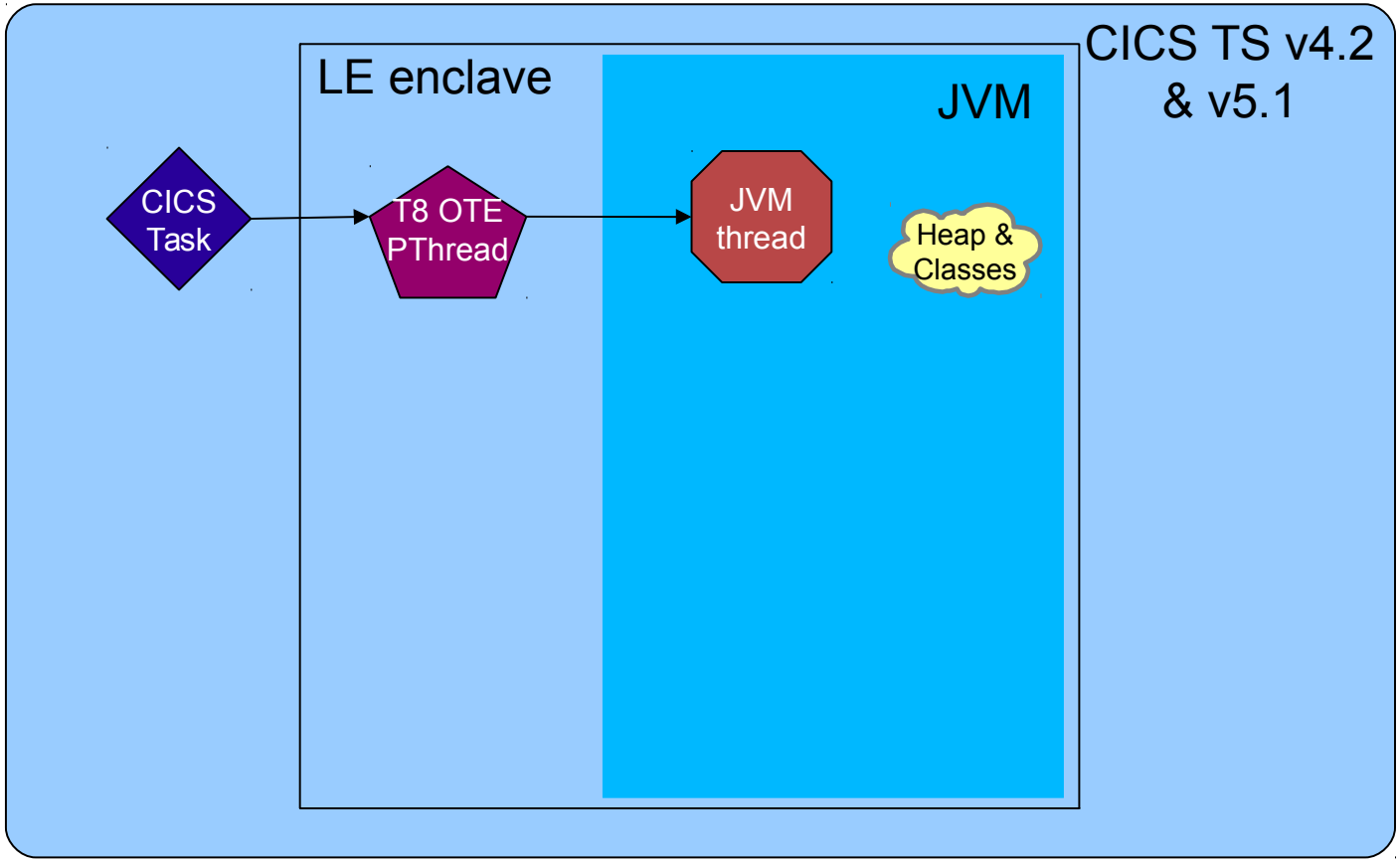


JVM Server Architecture

New CICS OTE TCB "mode".

Called "T8" - dubbed as both a CICS TCB and an LE "pthread".

JNI call to attach a pthread to an existing JVM.



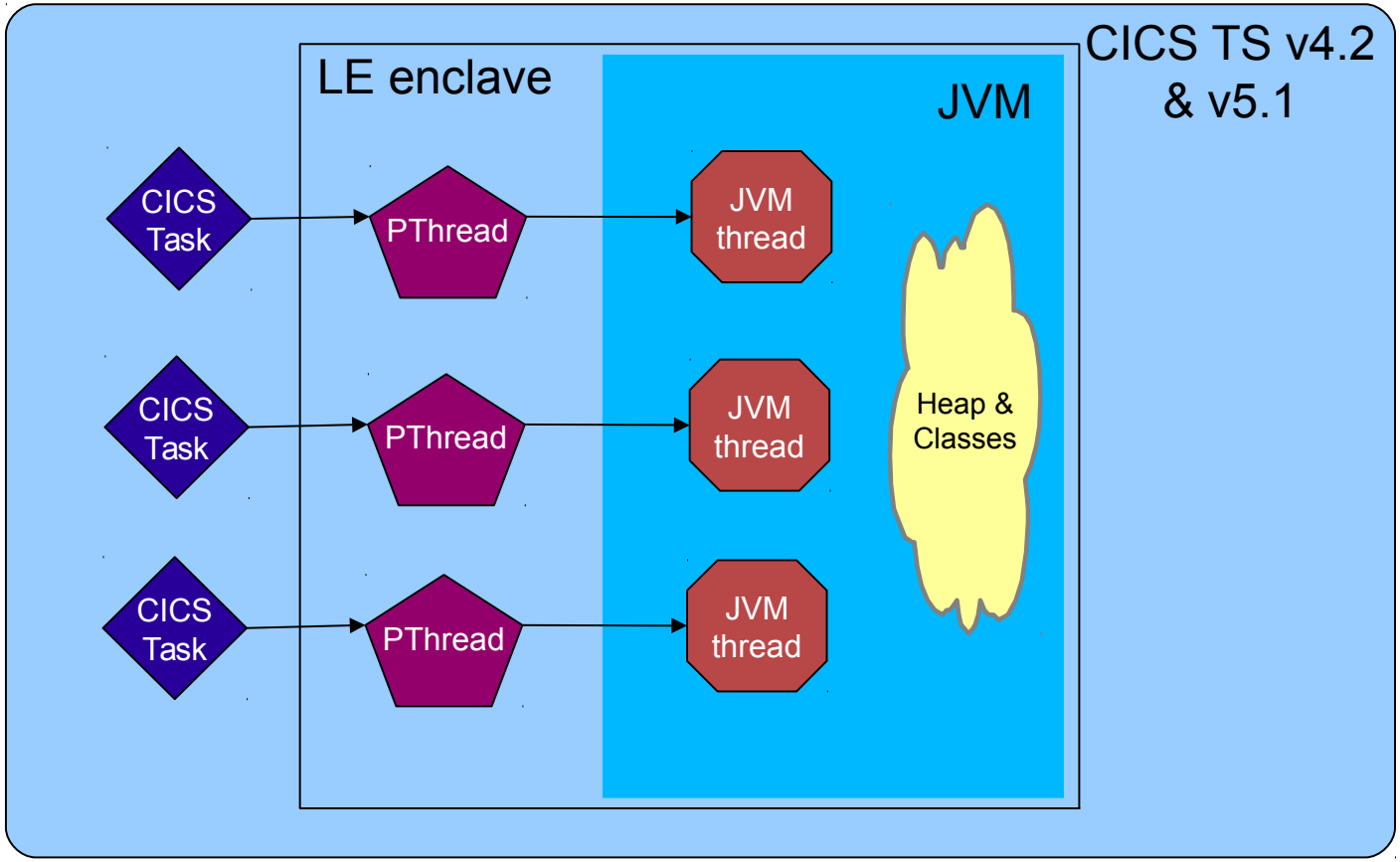
JVM Server Architecture

Can attach multiple pthread/T8/CICS tasks to the JVM at the same time.

Therefore serve **more requests** using a single JVM.

JVMServer thread "cost" is very small.

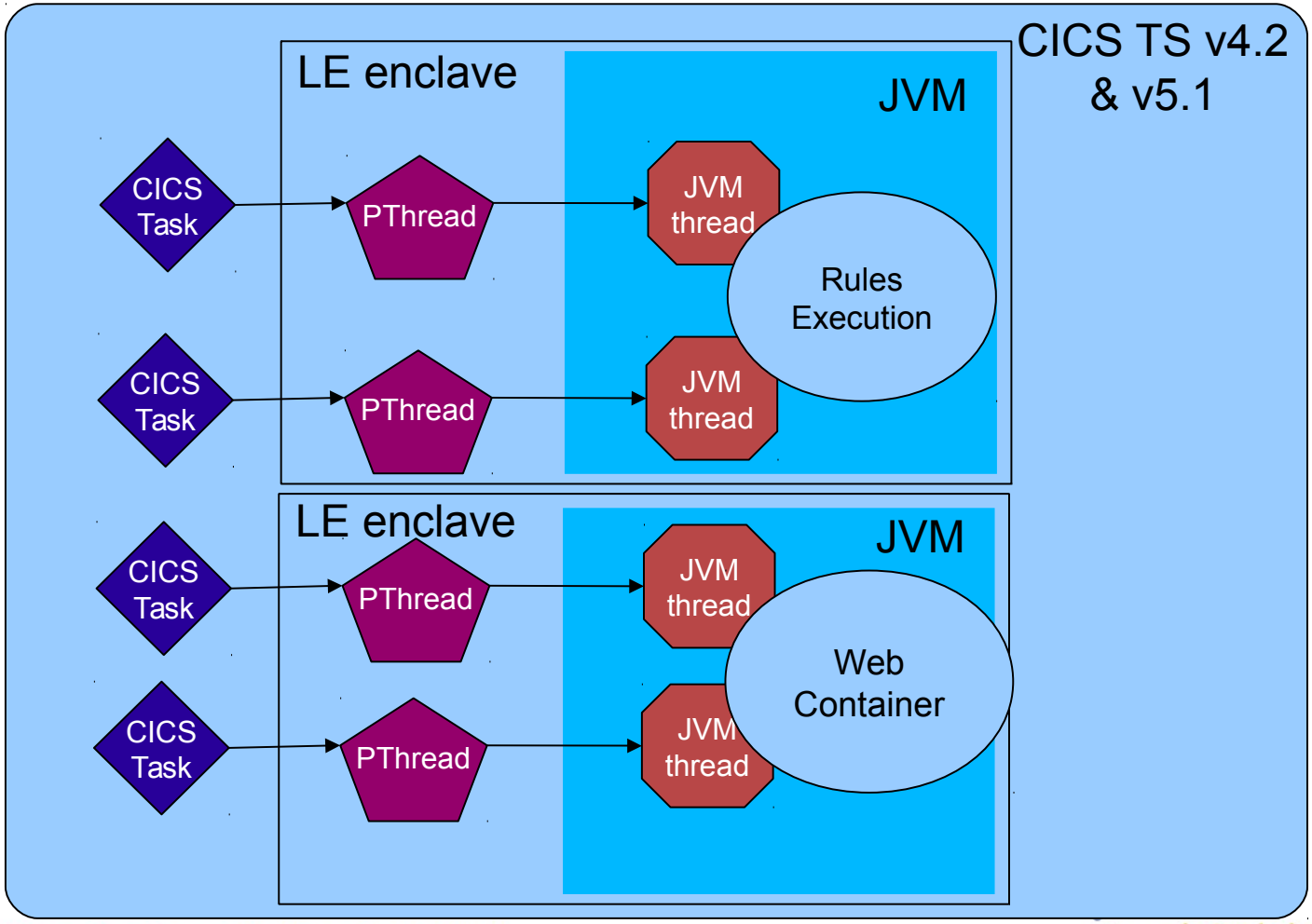
Result is **hundreds of tasks** concurrently per region.



JVM Server Architecture

Architected to allow multiple JVMServers in a single CICS.

Different types of work, or just a degree of isolation.



JVM Server: Thread-safe and OTE

Java and Thread-safety – **yes, it may be a concern!**

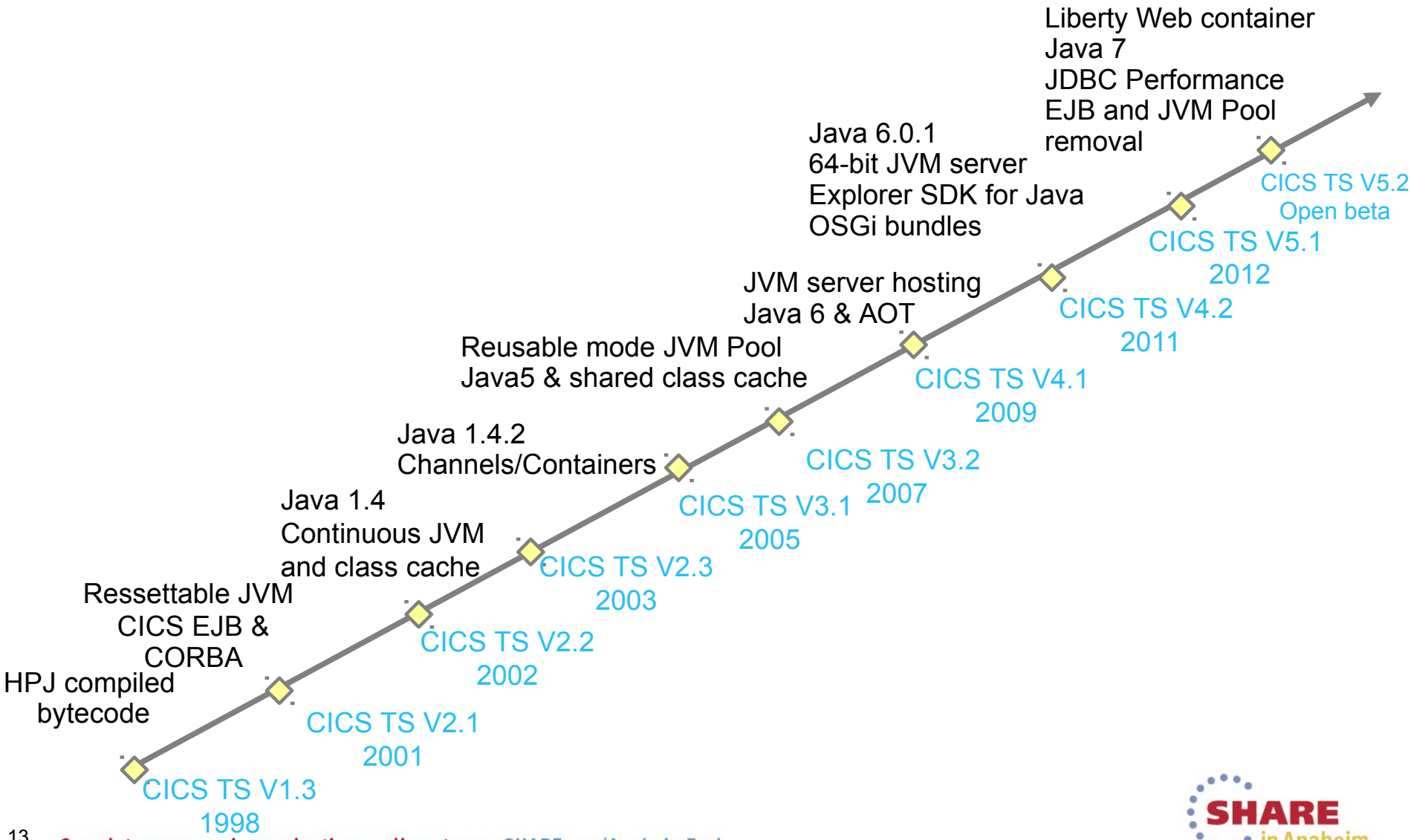
- In a pooled JVM, static objects are 'mine' – there's only one application thread
- In a JVM Server, static objects are shared (visible and accessible) with all the other threads/tasks/transactions in the same server
- Validate whether objects should be thread-local or static
- Ensure the concurrent versions of library classes are used

OTE – T8 and L8 threads

- In v4.2, T8 TCBs are for Java server threads, L8 TCBs are required for DB2 → TCB switch for every JDBC command
- **New in v5.1** T8 TCBs can be used for DB2 so the TCB switch is saved.

Support for Java 6/7 64-bit JVMs

CICS Java Support - Release history

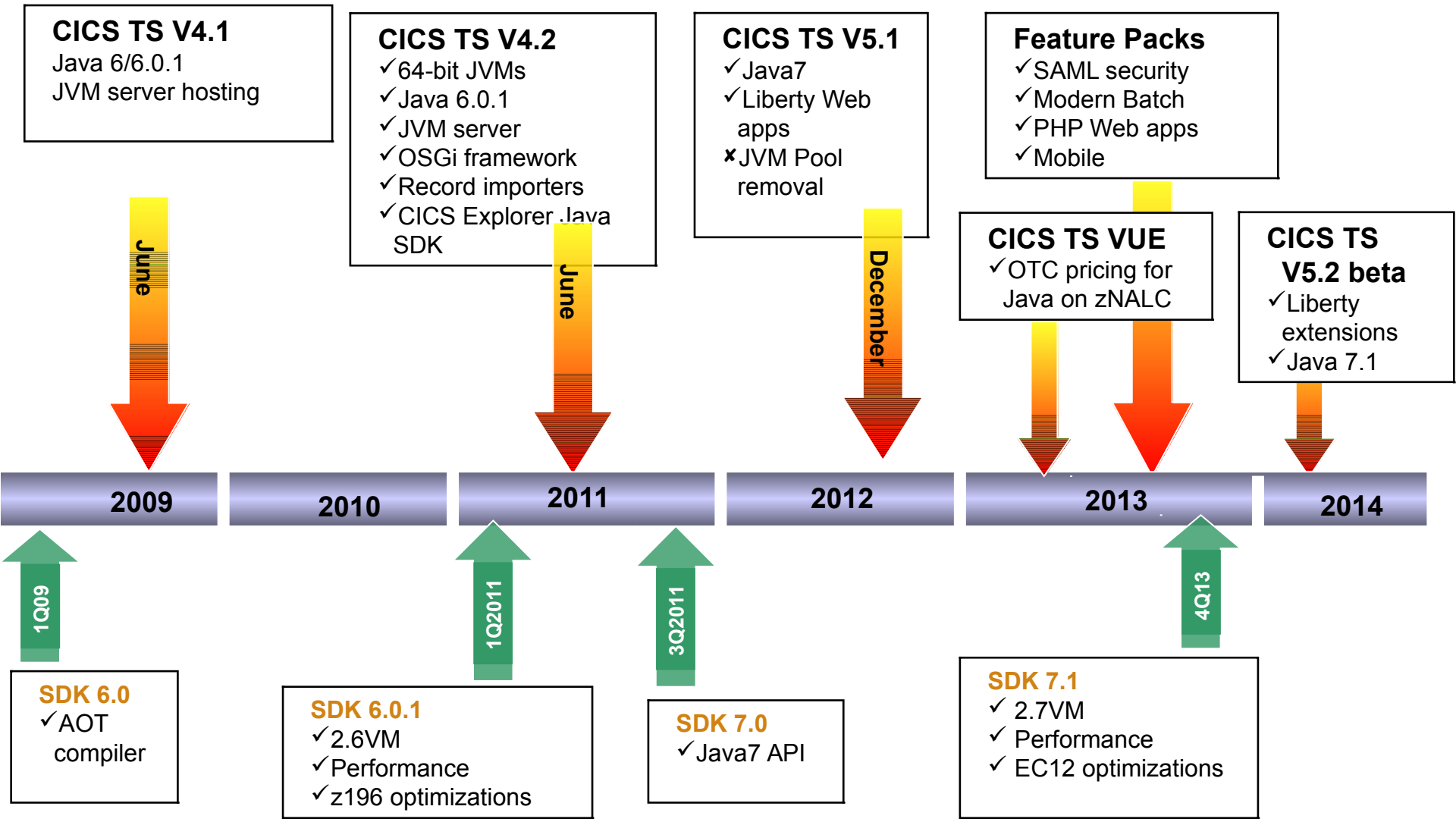


Support for Java 64-bit JVMs

CICS now supports 64-bit JVMs

- Both Pooled JVMs and JVMSERVERs
- Java 6.0.1 (CICS v4.2), Java 7 (CICS v5.1) or Java 7.1 (CICS TS 5.2 Open Beta)
  - If JAVA_HOME points to other than JVM then abend ASJJ
    - *DFHSJ0900 09/27/2010 11:00:07 IYK2ZIK1 Illegal Java version. CICS requires Java version 1.6.0 but has found Java version 1.5.0.*
- Java byte codes do not need recompilation (write once run anywhere)
- Support for 31-bit JVMs dropped
  - If JAVA_HOME points to a 31-bit installation, then abend ASJD
    - *DFHSJ0503 09/27/2010 10:50:21 IYK2ZIK1 DFHJVMPR Attempt to load DLL libjvm.so has failed. Runtime error message is EDC5253S An AMODE64 application is attempting to load an AMODE31 DLL load module. (errno2=0xC40B0013)*
- Java 6.0.1
  - IBM zEnterprise optimized version of Java 6 JVM
    - *Exploits new z196 instruction set*
    - *Improved GC*
    - *Improved JIT – (stores interpreter profiling information in class cache)*
    - *Significant performance improvements*
  - Download from z/OS Java website

CICS Java Roadmap - 2014



Java Execution Environments and Interoperability

Capitalize on pre-existing assets, artifacts, processes, core competencies, platform strengths

IBM Java Execution Offerings

- Transactional/Interactive
  - *WebSphere for z/OS (WAS z/OS)*
  - *WebSphere Process Server for z/OS (WPS)*
  - *JCICS*
  - *IMS Java*
  - *DB2 Stored Procedures*
- Batch oriented
  - *WebSphere Compute Grid (WAS-CG)*
    - *WAS/JEE runtime extensions*
- JZOS component of z/OS SDK
  - *JES/JSE-based environment*
  - *z/OS V1R13 Java/COBOL Batch Runtime Env.**
  - *JES/JSE-based, designed to inter-op with DB2 while maintaining transaction integrity*

Open Source or non-IBM vendor Application Server and Frameworks

- Tomcat, JBoss
- iBatis, Hibernate, Spring
- Ant

COBOL/Native Interoperability

- COBOL Invoke maps to JNI
- RDz and JZOS** have tooling to map COBOL copy books to Java classes
- JCICS
- IMS Java, JMP/JBP
- WAS CG, WOLA
- etc

IBM Java Runtime Environment

- IBM's implementation of Java 5/6/7 are built with **IBM J9 Virtual Machine** and **IBM Testarossa JIT Compiler** technology
  - *Independent clean-room JVM runtime & JIT compiler*

- Combines best-of breed from embedded, development and server environments… from a cell-phone to a mainframe!
  - *Lightweight flexible/scalable technology*
  - *World class garbage collection – gencon, balanced GC policies*
  - *Startup & Footprint - Shared classes, Ahead-of-time (AOT) compilation*
  - *64-bit performance - Compressed references & Large Pages*
  - *Deep System z exploitation – zEC12/z196/z10/z9/z990 exploitation*
  - *Cost-effective for z - zAAP Ready!*

- Millions of instances of J9/TR compiler

zEC12 – More Hardware for Java

- **Continued aggressive investment in Java on Z**
- **Significant set of new hardware features tailored and co-designed with Java**
  - ***Hardware Transaction Memory (HTM) (no zVM)***
    - Better concurrency for multi-threaded applications
    - eg. ~2X improvement to juc.ConcurrentLinkedQueue
  - ***Run-time instrumentation (RI)***
    - Innovation new h/w facility designed for managed runtimes
    - Enables new expanse of JRE optimizations
  - ***2GB page frames (no zVM)***
    - Improved performance targeting 64-bit heaps
  - ***Pageable 1MB large pages using flash (no zVM)***
    - Better versatility of managing memory
  - ***New software hints/directives***
    - Data usage intent improves cache management
    - Branch pre-load improves branch prediction
  - ***New trap instructions***
    - Reduce over-head of implicit bounds/null checks

New 5.5 GHz 6-Core Processor Chip
Large caches to optimize data serving
Second generation OOO design

*Up-to **60%** improvement in throughput amongst Java workloads measured with zEC12 and Java7SR3*

Engineered Together—IBM Java and zEC12 Boost Workload Performance
http://www.ibmsystemsmag.com/mainframe/trends/whatsnew/java_compiler/

Support for Java 6 64-bit JVMs

Pooled JVMs **(v4.2 only)**

- Support for many more JVMs per CICS region
  - 100+ can be possible
- Larger heap sizes
  - Reduces impact of Garbage Collection
- Profile changes
  - JAVA_HOME=/usr/lpp/java6_64/J6.0_64
  - USSHOME replaces CICS_HOME system initialization parameter

Support for Java 6 64-bit JVMs

JVM Server

- Messages now DFHSJxxxx instead of DFHLExxxx
- Much larger heaps possible
- Garbage Collection runs after an allocation failure
  - CJGC transaction is no longer used
  - Default GC policy uses more efficient gencon model
  - Heap dynamically sized by JVM
  - -Xcompressedrefs option uses 32-bit pointers to address 64-bit storage
  - Works for heaps up to 25GB
    - *Reduces CPU consumption but only recommended for use with single JVM server regions*

Support for Java 6 64-bit JVMs

MEMLIMIT

- Java stack and heap are now allocated in above the bar storage
- Above the bar requirement per Pooled JVM
  - –Xmx value in JVM profile
  - HEAP64 value in DFHJVMRO (default 8M)
  - LIBHEAP64 value in DFHJVMRO (default 1M)
  - STACK64 value in DFHJVMRO (default 1M) times 5 (application thread plus system threads)

Support for Java 6 64-bit JVMs

MEMLIMIT

- Above the bar requirement per JVM Server
  - –Xmx value in JVM profile (default 512M)
  - HEAP64 value in DFHAXRO (default 50M)
  - LIBHEAP64 value in DFHAXRO (default 1M)
  - STACK64 value in DFHAXRO (default 1M) times number of threads
    - *THREADLIMIT plus system threads*
    - *Number of GC helper threads depends on –Xgcthreads parameter*
      - » *Default is one less than the number of physical CPUs available*

Support for Java 6 64-bit JVMs

JDBC and SQLJ

- DB2 8.1 or 9.1 required to support 64-bit applications
- DB2 FP4 required for CICS TS 4.2 Java
- Make sure you have the latest DB2 JDBC (JCC) Fixpack

WMQ

- 64-bit driver required
- OSGi bundle required for JVM server

Middleware bundles (MQ and DB2)

- Need to be added to JVM servers using OSGI_BUNDLES and LIBPATH_SUFFIX settings in JVM profile

Native DLLs (JNI)

- All native DLLs must be recompiled with LP64 compiler option and bound as AMODE(64)
- LE will not allow an AMODE(31) DLL to be loaded by an AMODE(64) DLL

CICS OSGi Support

CICS OSGi Support Overview

OSGi

- OSGi development and packaging now required to deploy CICS applications to a JVM server 1
- Existing CICS Java applications using main() method linkage can run unchanged if wrapped in an OSGi bundle
- All JVM server applications must be thread-safe and can't use stabilised CICS EJB or CORBA functions
- Equinox used as OSGi implementation

CICS Explorer SDK

- Provides CICS Java development toolkit for use in any Eclipse 4.2.2 IDE (i.e RAD 8.0 or vanilla Eclipse SDK)
- Can be used to develop and deploy applications for any release of CICS (CICS TS 3.2 onwards)
- Java projects are developed as Plug-in Projects and then packaged in a CICS bundle and exported to zFS
- CICS TS V3.2/V4.1 Pooled JVM applications classes/JARs can be wrapped and deployed to OSGi JVM servers

[1] **Note** Axis2 CICS Web Services applications do not support OSGi packaging in v4.2

OSGI Bundle types in CICS

OSGi Bundles

- Just a jar with a few extra lines in the jar manifest file

Application Bundles

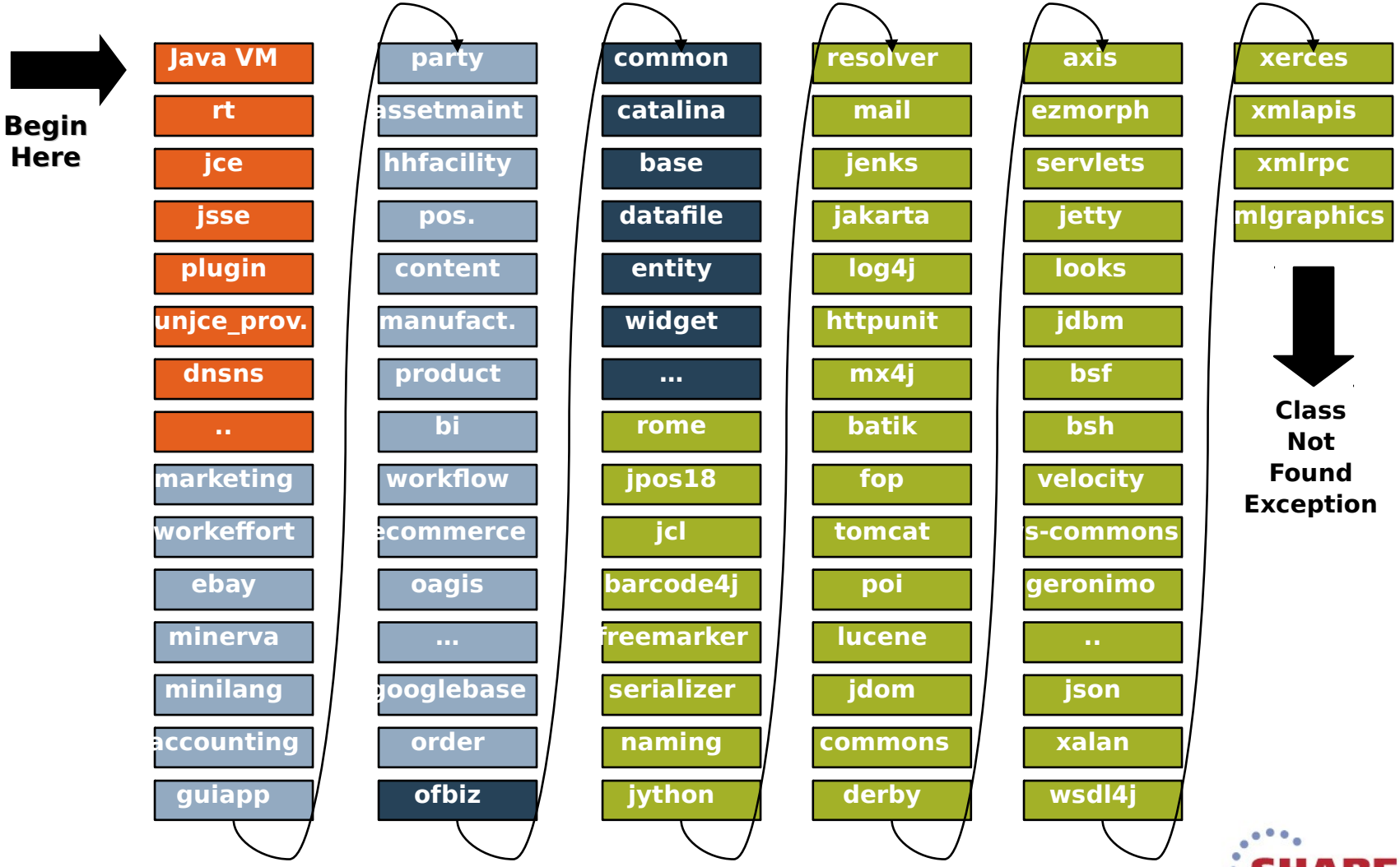
- Provide one or more entry points which can be LINKed too by CICS.
- This is done by using the CICS-MainClass directive
- Can import packages from other bundles, i.e. JCICS

Library Bundles

- Provide no entry points but simply export code to be used by other bundles
- Shared library services

```
Manifest.mf
Bundle-SymbolicName: com.ibm.cics.server.examples.hello
Bundle-Version: 1.0.0
...
CICS-MainClass: examples.hello.HelloCICSWorld
Export-Package: my.library.classes 1.0.0
```

The Global Classpath



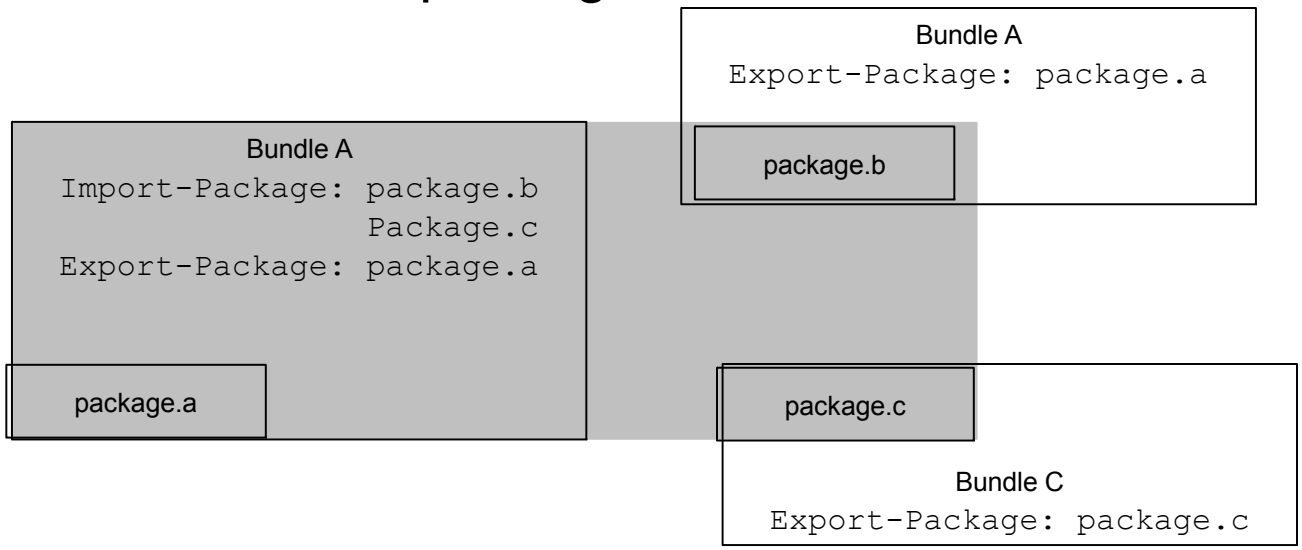
Class loading with OSGi

No more CLASSPATH

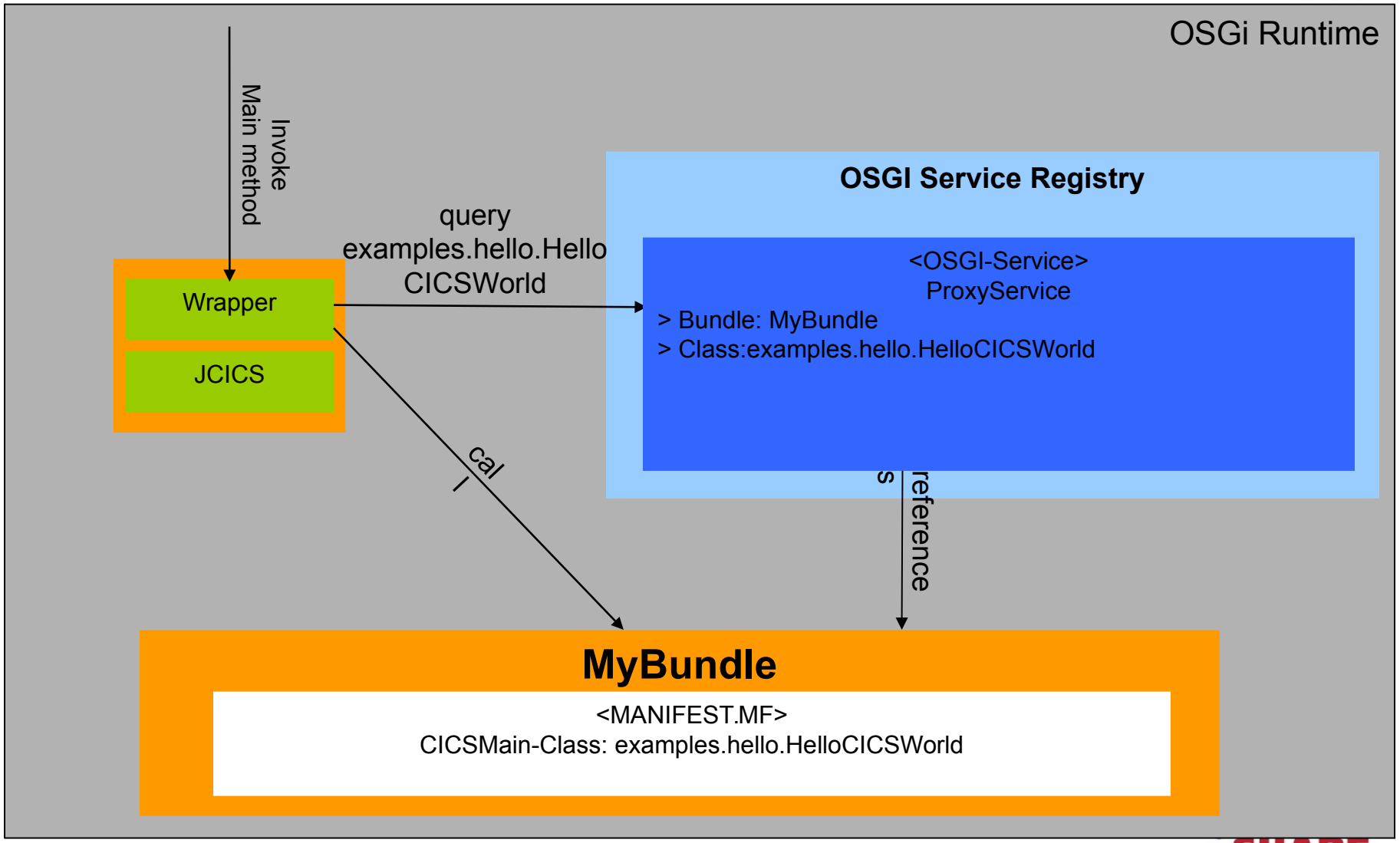
- Each bundle has its own class loader

Class space is the classes required for the bundle

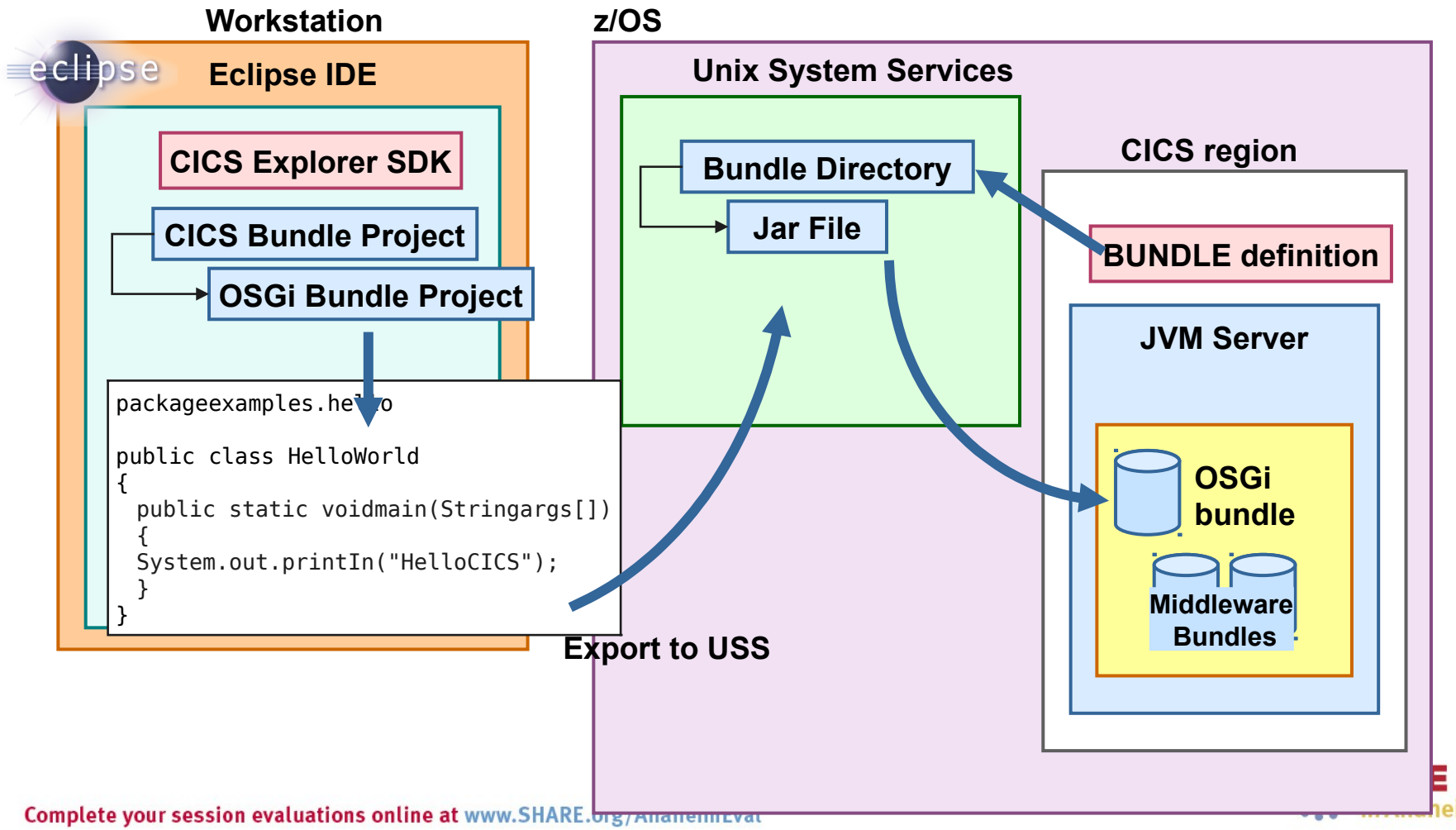
Smallest unit is a package



JVMSERVER OSGi Details

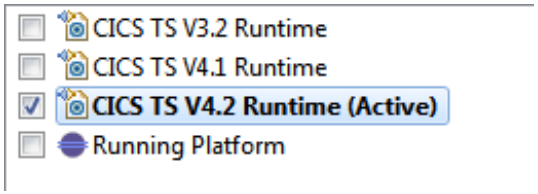


OSGi bundle deployment



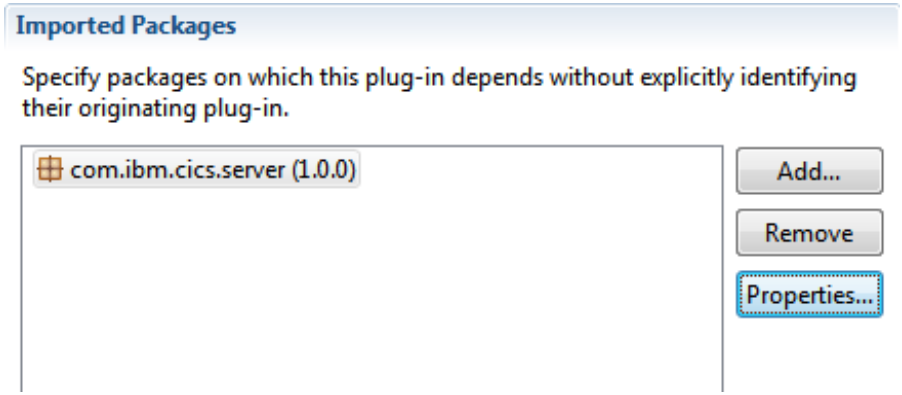
CICS Explorer SDK - Development

1. Install CICS Explorer SDK into Eclipse
2. Set Target Platform
   (sets JCICS and JVM levels)
   - Window → Preferences…→
     Target Platform → Add… → Template



3. Create New OSGi Project
   - New → Plug-in Project

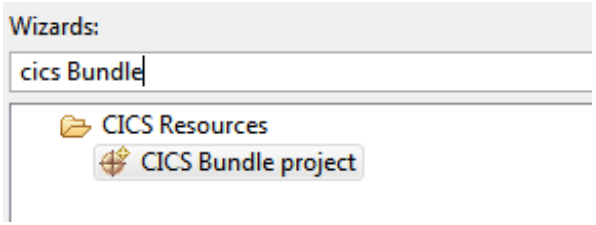
4. Provided access to JCICS package
   - MANIFEST.MF → Dependencies →
     Imported Packages →
     com.ibm.cics.server
   - Add other bundle imports if required



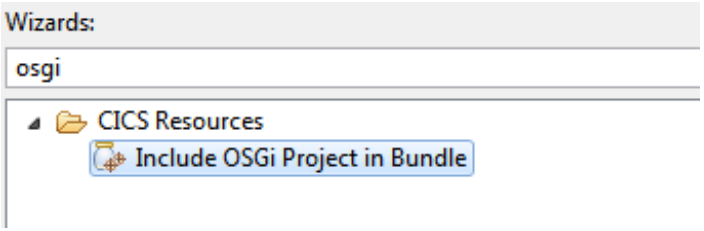
5. Import/Create your Java class

CICS Explorer SDK - Deployment

6. Create CICS Bundle
   - New→CICS Bundle Project



7. Add OSGi bundle meta-data file to CICS Bundle
   - New→Include OSGi Project in Bundle

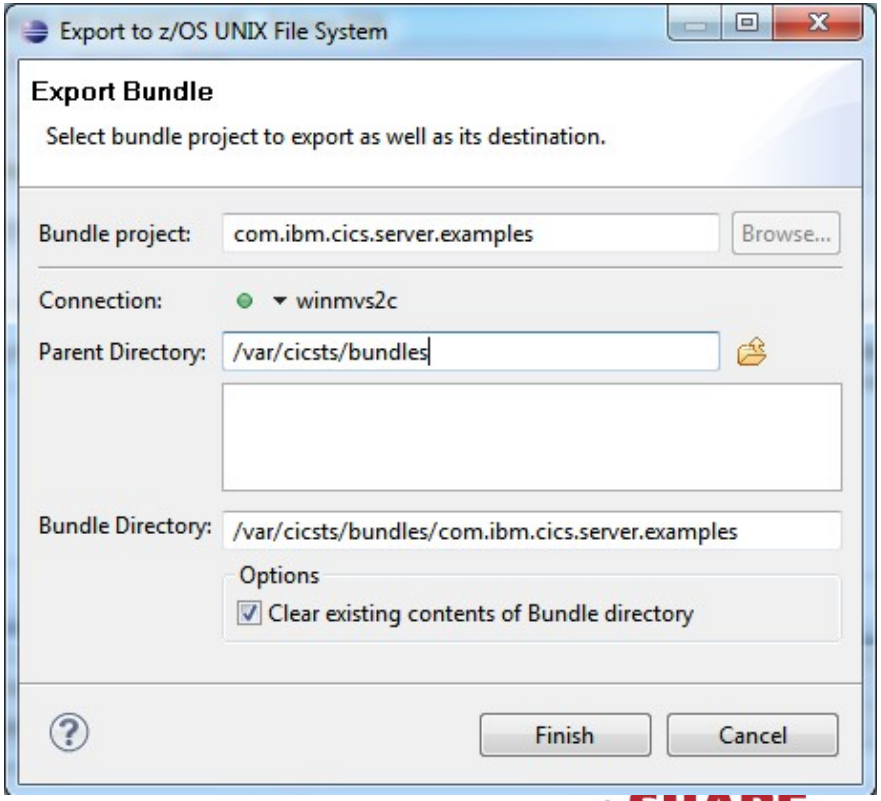


CICS Explorer SDK – Deployment 2

8. Provide CICS region userid read access to bundledir
   - mkdir /var/cicsts/bundles
   - chmod 750 /var/cicsts/bundles1[1]

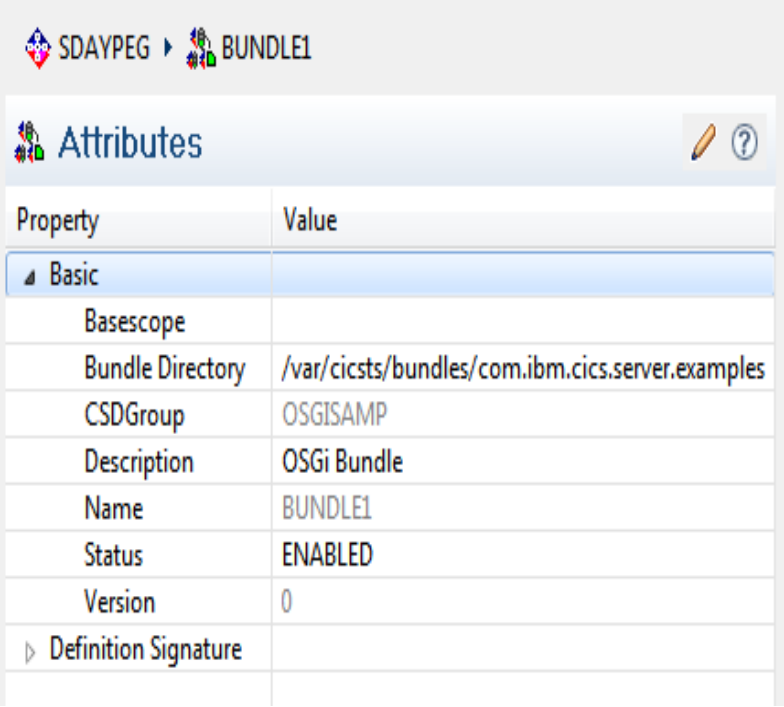
9. Connect CICS Explorer to USS FTP daemon
   - Windows → Open Perspective → z/OS

10. Export CICS Bundle to CICS
    - →CICS to z/OS UNIX File System



[1] **Note:** CICS region userid and FTP user must be in same USS group

Defining a CICS BUNDLE



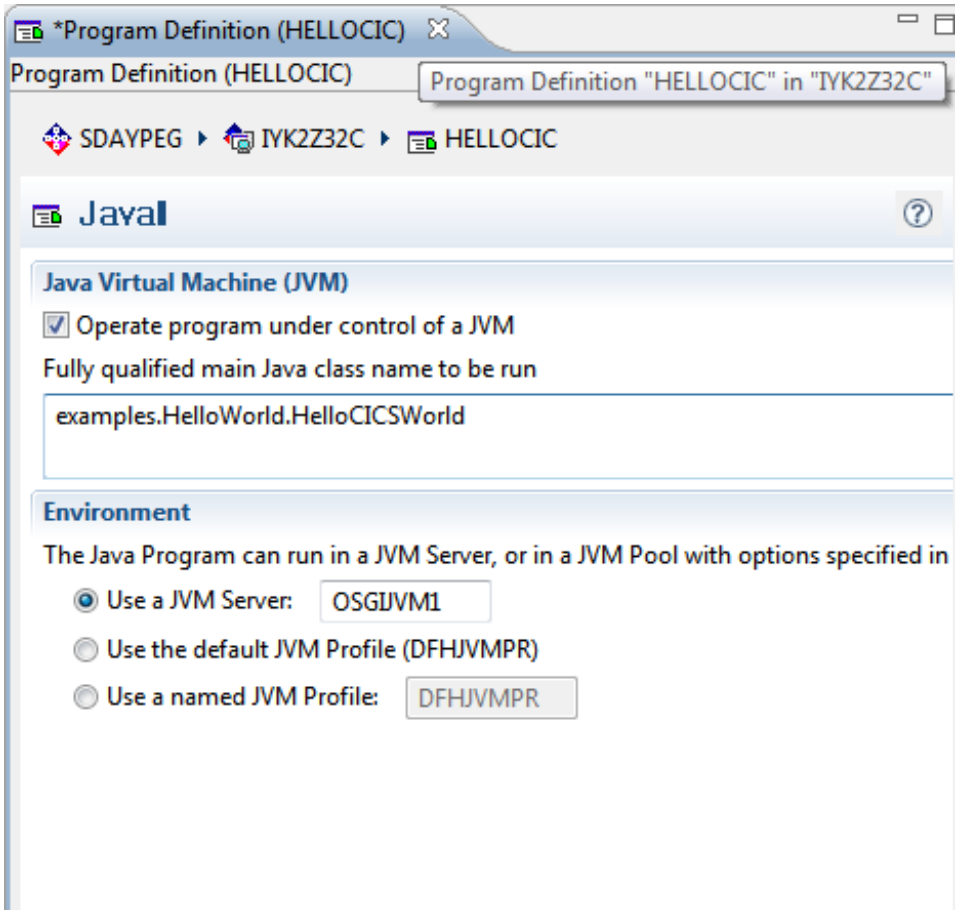
Bundle Directory

- Name of directory containing deployed JAR and bundle meta data files

Status

- ENABLED→Activate on install of resource

Defining a Program to run in JVMSERVER



JVMServer

- Name of JVM server resource

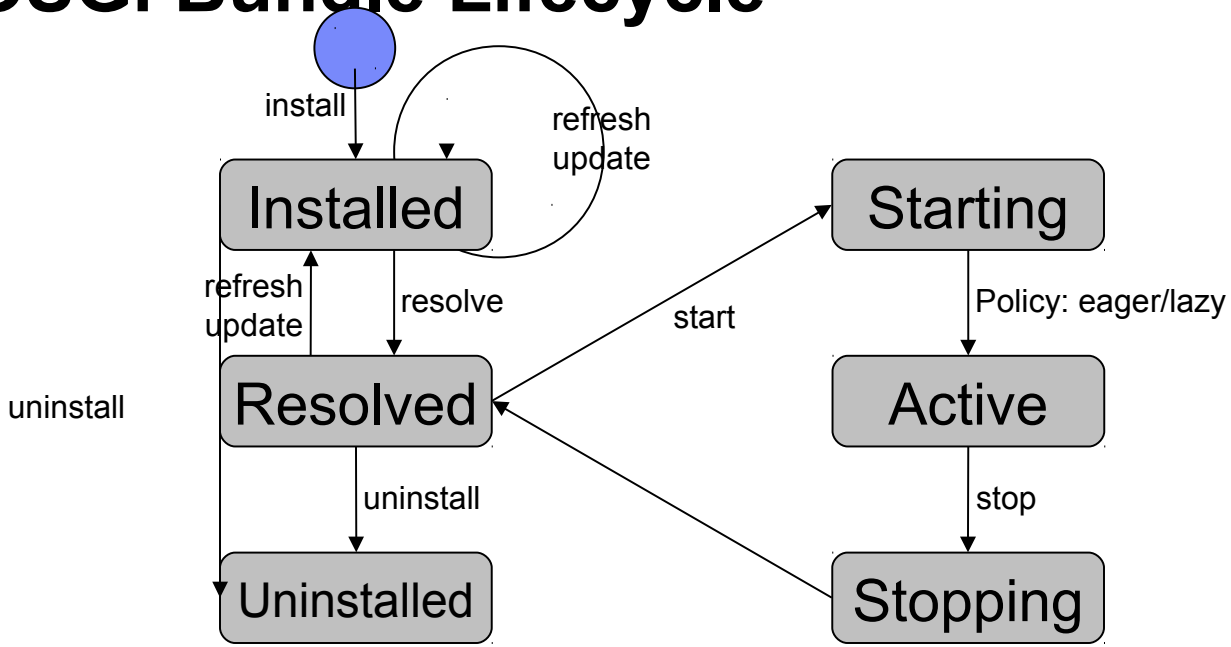
Main Java class

- OSGIService defined in the OSGi bundle manifest
- Either an alias or the full package.class name

Also required

- CONCURRENCY(THREADSAFE)
- EXECKEY(CICS)

OSGi Bundle Lifecycle



OSGi bundle state displayed in CICS Explorer OSGi bundle view

Tasks | Programs | JVM Serve | Bundles | Bundle Par | OSGi Bund | OSGi Servi | JVM Serve | Bundle De | Program D | Transactio

CNX0211I Context: IYK2Z32C. Resource: OSGIBUND. 4 records collected at 24-Mar-2011 09:41:29 — Symbolic Name:

Region	Symbolic Name	State	Bundle Part	Bundle	JVM Server	Install Time	Version	Bundle ID
IYK2Z32C	com.ibm.cics.server.examples.hello	✓ ACTIVE	hello	SAMPLES	OSGIJVM1	24-Mar-2011 09:41:11	1.0.0	13
IYK2Z32C	com.ibm.cics.server.examples.jcics	✓ ACTIVE	jcics	SAMPLES	OSGIJVM1	24-Mar-2011 09:41:11	1.0.0	14
IYK2Z32C	com.ibm.cics.server.examples.web	✓ ACTIVE	cicsweb	SAMPLES	OSGIJVM1	24-Mar-2011 09:41:11	1.0.0	15
IYK2Z32C	sleep	✓ ACTIVE	sleep	SLEEP	OSGIJVM1	23-Mar-2011 21:49:46	1.1.0	12

Java Pool and EJB Statement of Direction

CICS TS V4.2 announce letter

A future release of CICS TS intends to **discontinue support for session beans using Enterprise Java Beans (EJB), and the Java pool infrastructure**. Customers are encouraged to migrate Java applications to the new JVM server infrastructure, and to migrate EJB applications to Java SE components and make them available through web services or the JEE Connector Architecture (JCA). **CICS will continue to support Java as a first class application programming language for CICS applications**, including enhancements to the CICS interfaces, the deployment infrastructure, and Java runtime environment.

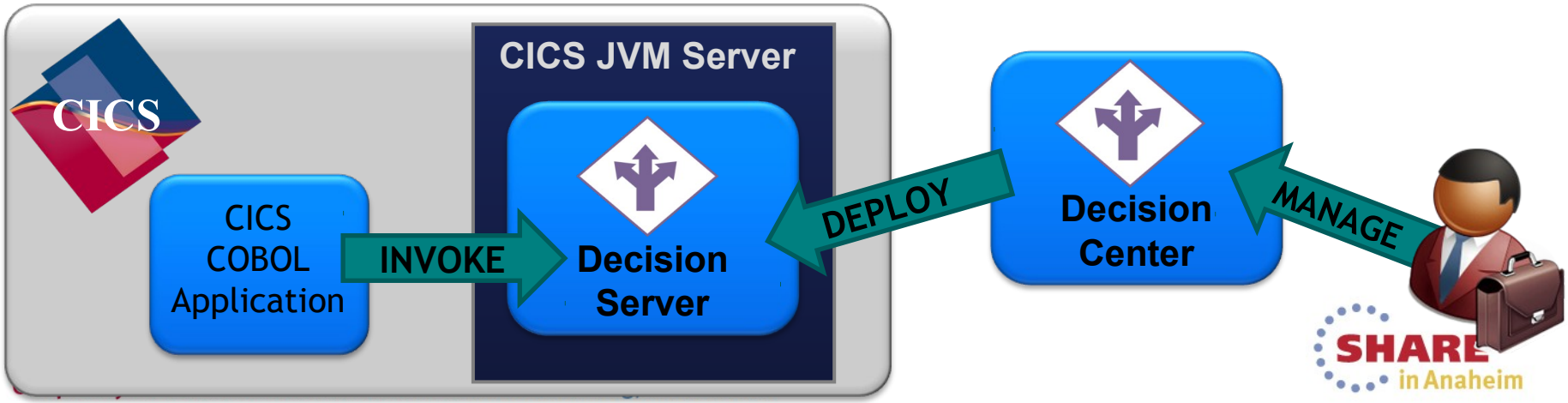
GONE IN V5.1!!!

ODM Rules Execution Engine in CICS JVM Server

Operational Decision Management & CICS

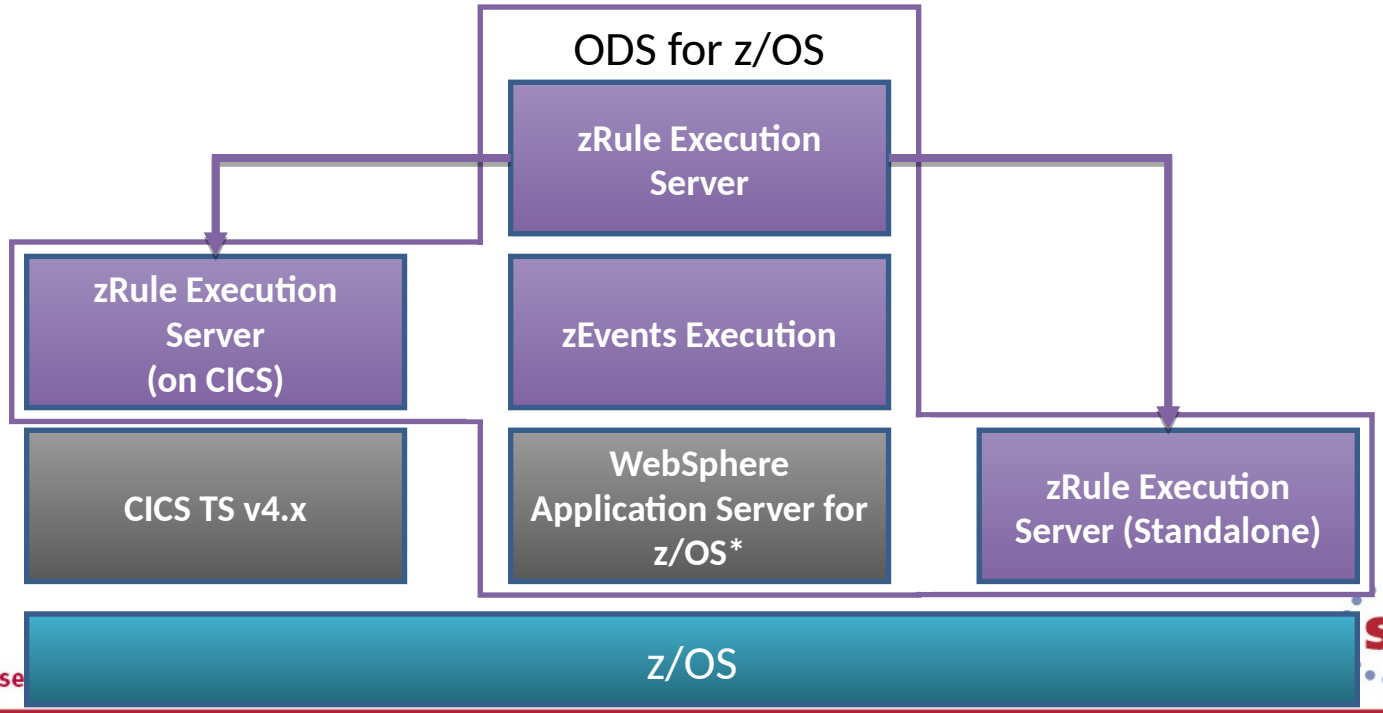
Externalize embedded business rule logic & execute within CICS

- *Gain business agility with existing and new CICS applications*
  - Manage decision logic on a separate lifecycle to application code
  - Ability to react to changes in a fast paced, competitive marketplace
- *Lower the cost of maintaining your business applications*
  - Improvement operational efficiency and total cost of ownership
- *Consistent Decision evaluation across the enterprise*
  - Author decision rules once and deploy to multiple systems on z/OS and distributed
- *Optimized decision execution*
  - Highly efficient rule execution engine
  - Local optimization of Decision Server within the CICS JVM Server environment



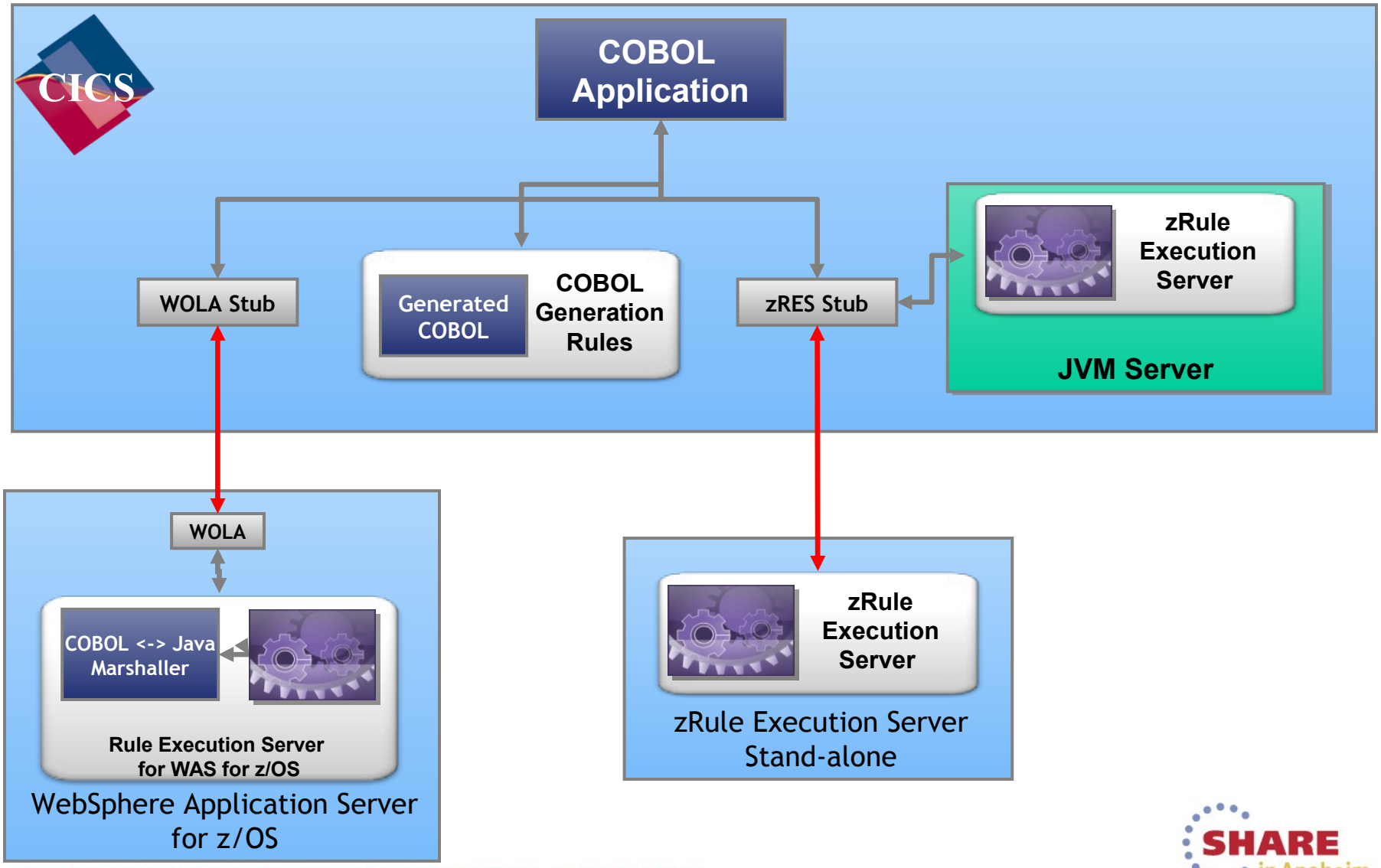
Decision Server for z/OS

- Decisions can be invoked from existing CICS and batch applications
- Runtime support for COBOL data types
- Flexible runtime deployment to fit any System z environment:
  - Deployed on WebSphere Application Server for z/OS
  - Deployed standalone to z/OS
  - Deployed in CICS TS 4.x JVMServer environment

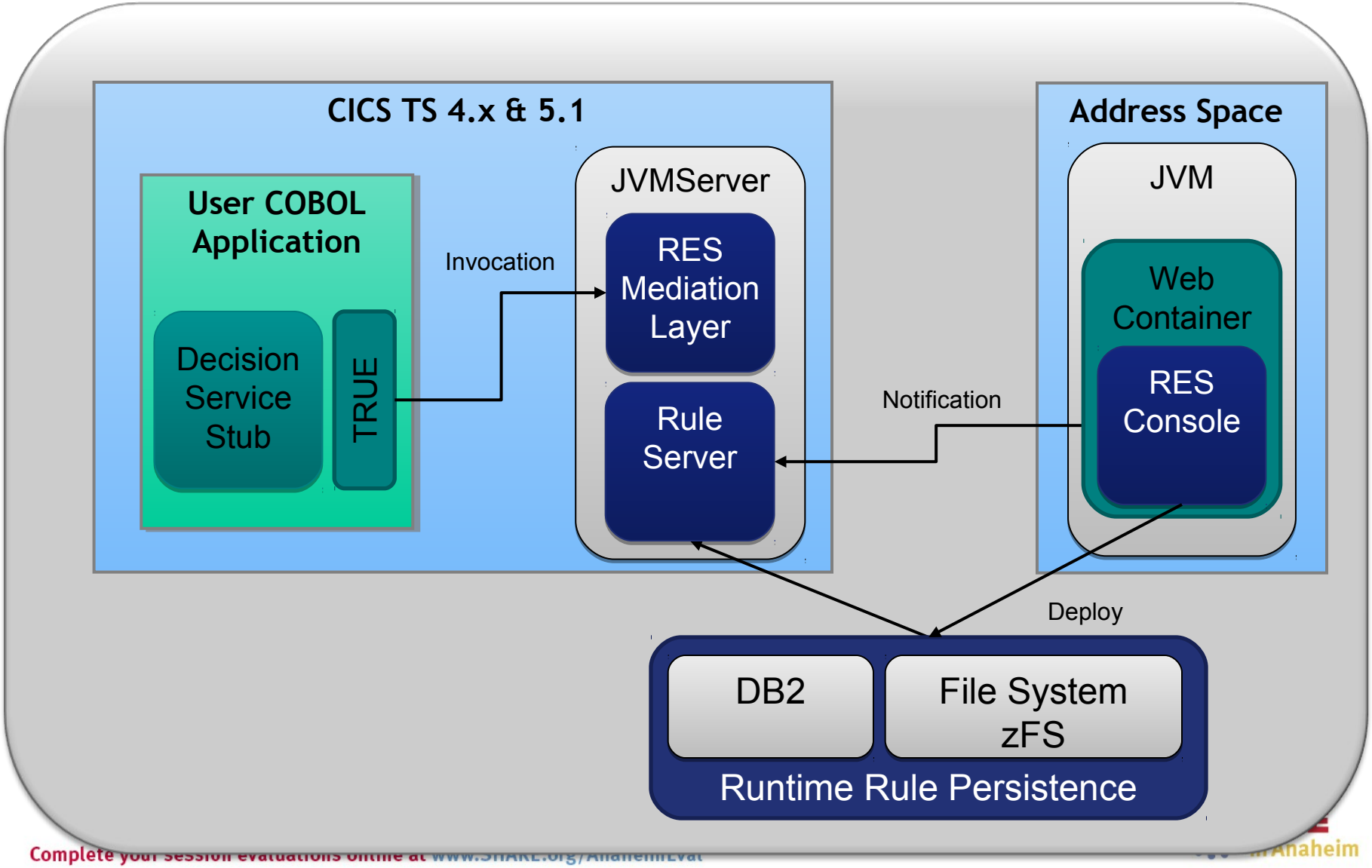


*OEM

Rule invocation options for CICS



zRule Execution Server for z/OS – CICS TS 4.x



Summary

JVM Options in CICS TS v4.2 and v5.1

- JVM Pool
- JVM Server

64 Bit JVM Support

OSGi for application management

WODM Rules Execution Engine

Questions?