Advanced Programming Specialist Group

Ivan D. Hargreaves

9th October 2014

Can Big Iron run New Era Web Applications?

?

Introductions

Ivan D. Hargreaves – IBM Software Engineer, Technical Leader and Lead Developer of Java technologies in CICS.

Ian J. Mitchell – IBM Distinguished Engineer, System z CTO & CICS Portfolio Architect

Abstract

Mainframe servers such as the IBM zSeries with the CICS transaction monitor are well known for running traditional enterprise online applications written in COBOL. But can they run the new style Java applications appropriate for today's web environment?

This talk will outline how the latest version, CICS Transaction Server Version V5.2, not only supports Java and other contemporary languages but also supports a new lightweight implementation of the IBM WebSphere Application Server, known as WebSphere Liberty Profile (WLP).

This provides a local environment for running Web Applications, and EJBs and is designed to be fast starting and dynamically deployable for both developer productivity and production workloads.

In this session, we introduce WLP, recap the integration of Java-based technologies into CICS, and describe the particular challenges in running the WebSphere Liberty Profile inside CICS on z/OS.

Customer **I**nformation **C**ontrol **S**ystem - CICS

- IBM® CICS® Transaction Server for z/OS® is **the** premier enterprise-grade, mixed-language application server
- It supports z/OS Assembler, PL/1, COBOL, Java, and even PHP
- First released in 1968 as PU-CICS (Public Utility), then in 1969 as CICS
- Development moved in 1974 from Illinois (USA), to Hursley (United Kingdom)

Hursley House

IBM CICS

"CICS is probably the most successful piece of software of all time . . . It is the mainstay of business computing throughout the world . . . Millions of users unknowingly activate CICS every day, and if it were to disappear the world economy would grind to a halt."

Phil Manchester, Personal Computer Magazine

- 9 out of the top 10 banks in the global Fortune 100 use CICS
- 7 out of the top 10 of the US Fortune 100 use CICS, including the top 2
- 100% of banks in the US Fortune 100 use CICS
- 75% of top US Fortune 100 Insurance companies use CICS
- 30 years and $1 trillion invested in CICS applications (IDC)
- Hundreds of CICS software packages from 100 ISVs
- 950,000 programmers earn their living from CICS
- CICS handles more than 30 billion transactions/day valued at over $1 trillion/week for 30 million end users of CICS Apps

In excess of 1 billion transactions a day being achieved by several customers

Although most people are blissfully unaware of CICS, they probably make use of it several times a week, for almost every commercial electronic transaction they make. In the whole scheme of things, CICS is much more important than Microsoft Windows."

Martin Campbell-Kelly, From Airline Reservations to Sonic the Hedgehog (A History of the Software Industry)

What can happen in a single second?

Every second there are approximately:

- **250** ITunes downloads
- **3,600** Instagram photos taken
- **7,000** Twitter tweats
- **30,000** Facebook likes
- **43,000** YouTube views
- **60,000** Google searches
- **1,157,407** CICS transactions

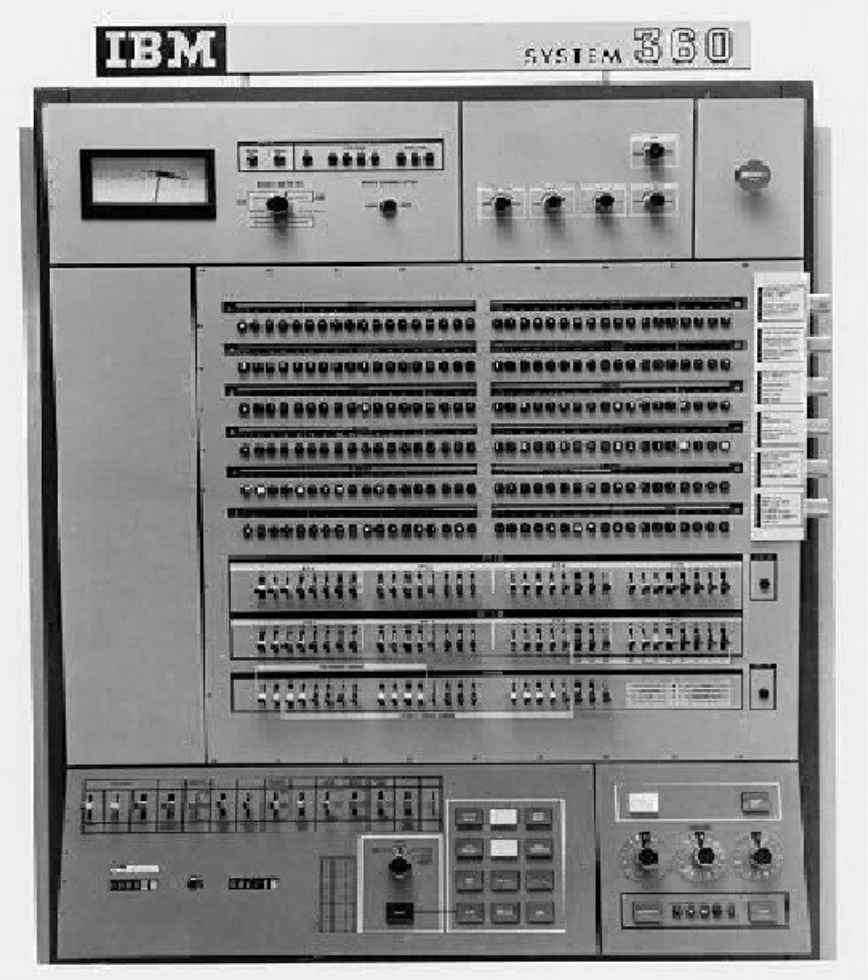
Workload comparison (transactions per second):

- **50** Google App engine
- **1500** Amazon Webservices / Windows Azure
- **5000** Distributed Systems typically

CICS? we don't really know, we haven't hit the limits...
...but some customers run in excess of **150,000** transactions per second.

Java in CICS

In the beginning....



Sometime later....

- 1996, Java 1 (JDK 1.0.2) was the first stable release of Java
- 1998 CICS TS v1.3 was released, with support for the Java Virtual Machine
  - Applications could be written in Java
  - CICS ran it's own JVM
  - JCICS classes exposed the CICS API to Java programs
- 2001 CICS TS v2.1 offered support for EJB 1.0 (session beans)
- 2002 CICS TS v2.2 was shipped with enhanced support for EJBs

Foibles of Java in CICS!

Inefficient

- Each Transaction used a whole JVM to itself
- Pools of JVMs were necessary to provide concurrency
- Customers were hitting 31-bit storage limits with 10+ JVMs

Non-standard

- Making Java work in a CICS-way, just didn't work
- JVM phaseout, classcache management, and trace integration were all CICS specific
- There were also CICS equivalents for standard JVM options

Restricted capability

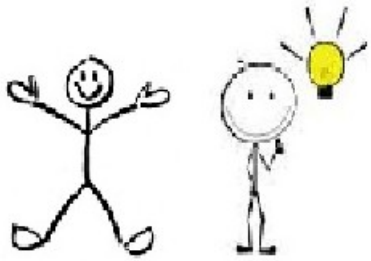
- JVMs were stateless, and you never knew which JVM your App would run in
- Standard 'server' type Java applications just couldn't run
- Debugging proved difficult, to which JVM do you attach the debugger?

...Java in CICS failed to gain mass favour amongst Mainframe programmers who thought it slow, inefficient and alien. While Java programmers thought it was too 'Mainframe like' and inaccessible.

Then one day a couple of smart colleagues payed me a visit...

...what IF?

"...runs just like it does elsewhere"

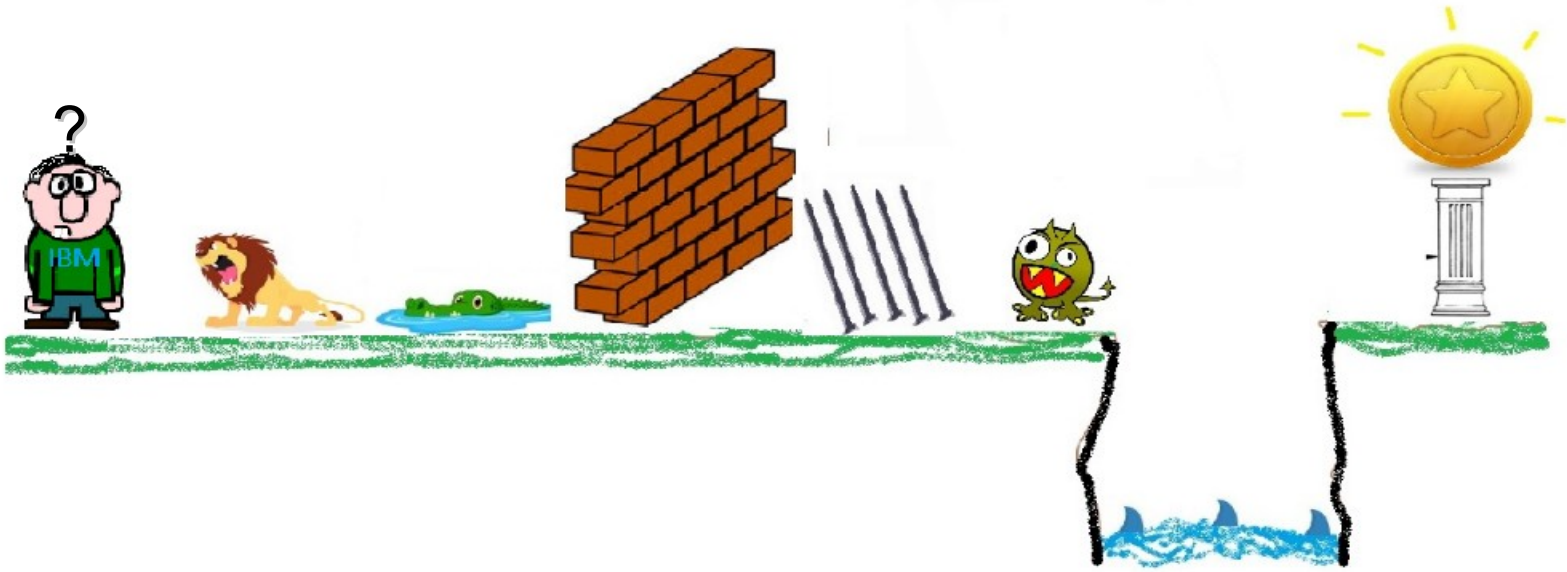


..."multi-threaded JVM, a task per thread!"

"Leverages Legacy programs and data"

"...by the way, it's your next assignment".

Challenges...



- How do we attach multiple CICS TCBs (Task Control Blocks) to the same JVM when LE (Language Environment) only has a single entry-point?
- How can a single JVM share the context of many CICS tasks at once?
- If we were to start multiple POSIX threads, how would CICS dispatcher maintain control of them, or handle their lifecycles and that of the JVM?
- How can we integrate two different execution stacks (CICS and LE) on the same TCB providing access to both environments and robust recovery?

Technical ingredients...

- Create JVM on Initial Process Thread (IPT) of our POSIX environment LE (Language Environment) within CICS's address space
- Once initialised, place the IPT into an MVS wait loop
- When CICS dispatcher needs a new Java enabled Task Control Block (TCB or thread), send an MVS ECB (event control block) to wake the IPT, and run a routine to spawn a new POSIX thread
- Bootstrap the thread up through the CICS kernel, dubbing it as CICS capable
- Use a pool of T8 TCBs to prevent repeated dubbing and destruction of threads (T8s now in MVS wait too)
- When a CICS transaction is required to run Java, push its task state onto a free Java enabled T8 TCB

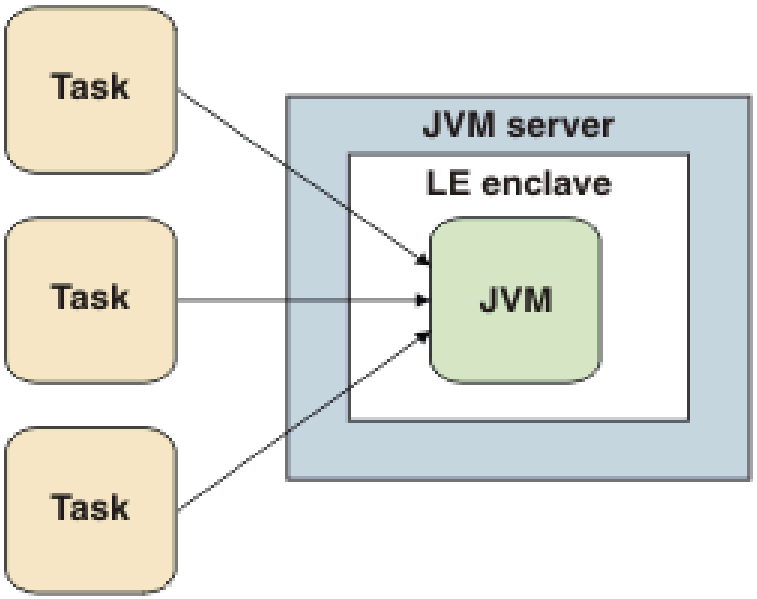
(continued)

Add some 'magic'...

- Use assembler PC and PR instructions, along with a special z/OS hardware stack, to provide a switching mechanism between execution in CICS and execution in LE
- Issue JNI (Java Native) code to attach the POSIX (T8) thread to the long-running JVM and execute Java code with the added CICS transaction/task context (thread local)
- On return, detach Java thread from POSIX TCB, switch stack back into CICS, and disassociate CICS Task environment from T8 (SJKEP control block to provide return)
- Place T8 TCB back into CICS dispatcher pool for re-use

CICS capable *Java* code in a multi-threaded **server** JVM!

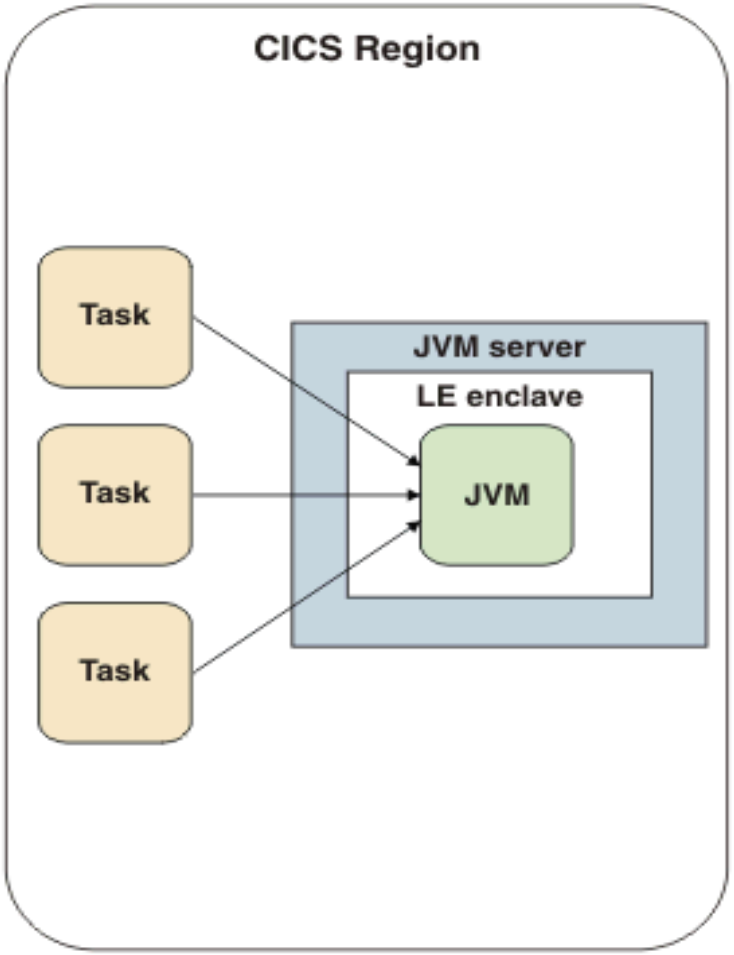
The JVM server



- New CICS resource
- Backed by a long running JVM that handles multiple CICS tasks concurrently
- Each CICS Task becomes a thread running in the JVM server
- Each Task can run Java **and** access CICS resources (JCICS API = EXEC CICS)
- JVM server uses a 64-bit LE (Language Environment) Enclave, and 64-bit JVM

JVM server details

- Up to 256 concurrent tasks per JVMSERVER
- Application data may be shared (Server/Engine)
- Multiple JVM servers per region
- **No** EJB or CORBA support
- Debugging - so much easier
- JVM configuration provided by JVM profile
- LE (POSIX) configuration provided by LE runtime options file
- Statistics produced for resource, and for JVM.



The rise of JVM server in CICS

- In 2009 the *Dynamic Scripting Feature Pack for CICS TS v4.1 was released*
- It was based on a new technology and CICS resource called the "**JVMSERVER**"
- In CICS TS v4.2, the JVM server was given full announcement and provided a new way in to run Java applications in CICS
- The 'pooled' Java model was announced as deprecated

Advantages

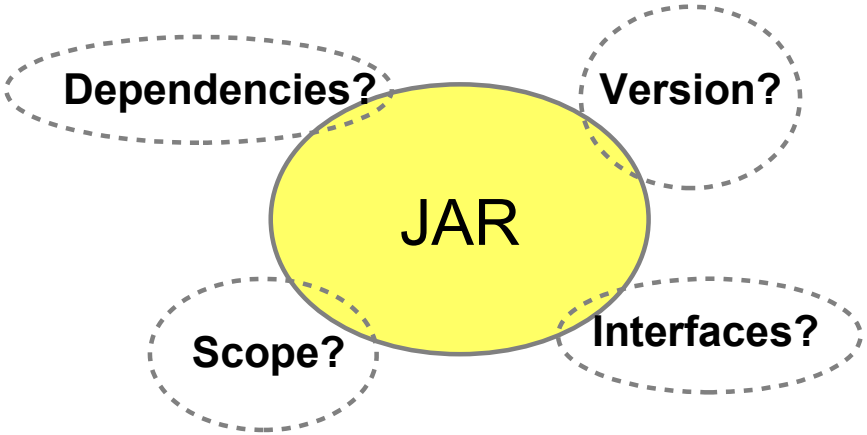
- Scalable, robust, centralised, **standardised**, Java
- Leveraging **legacy** data and programs
- Fully **integrated** with CICS
- Significant **storage reduction**
- Java Application Servers/Engines supported (sMASH for PHP)
- **64-bit** JVM supporting very large JVM heaps
- **Multiple** JVM servers possible for App separation or workload balancing

...but why run Java on the Mainframe?

- **Consolidation** of servers & infrastructure, with centralised admin & control
- **Robustness** & **scalability** of CICS/Mainframe
- **Security** of Mainframe
- "**Centre of gravity**" of data? (CICS data/DB2 data)
- **zIIP** advantage, potential to convert existing CICS applications to COBOL
- Some customers can **utilise** existing capacity, no need to commission a new server

Issues with Java

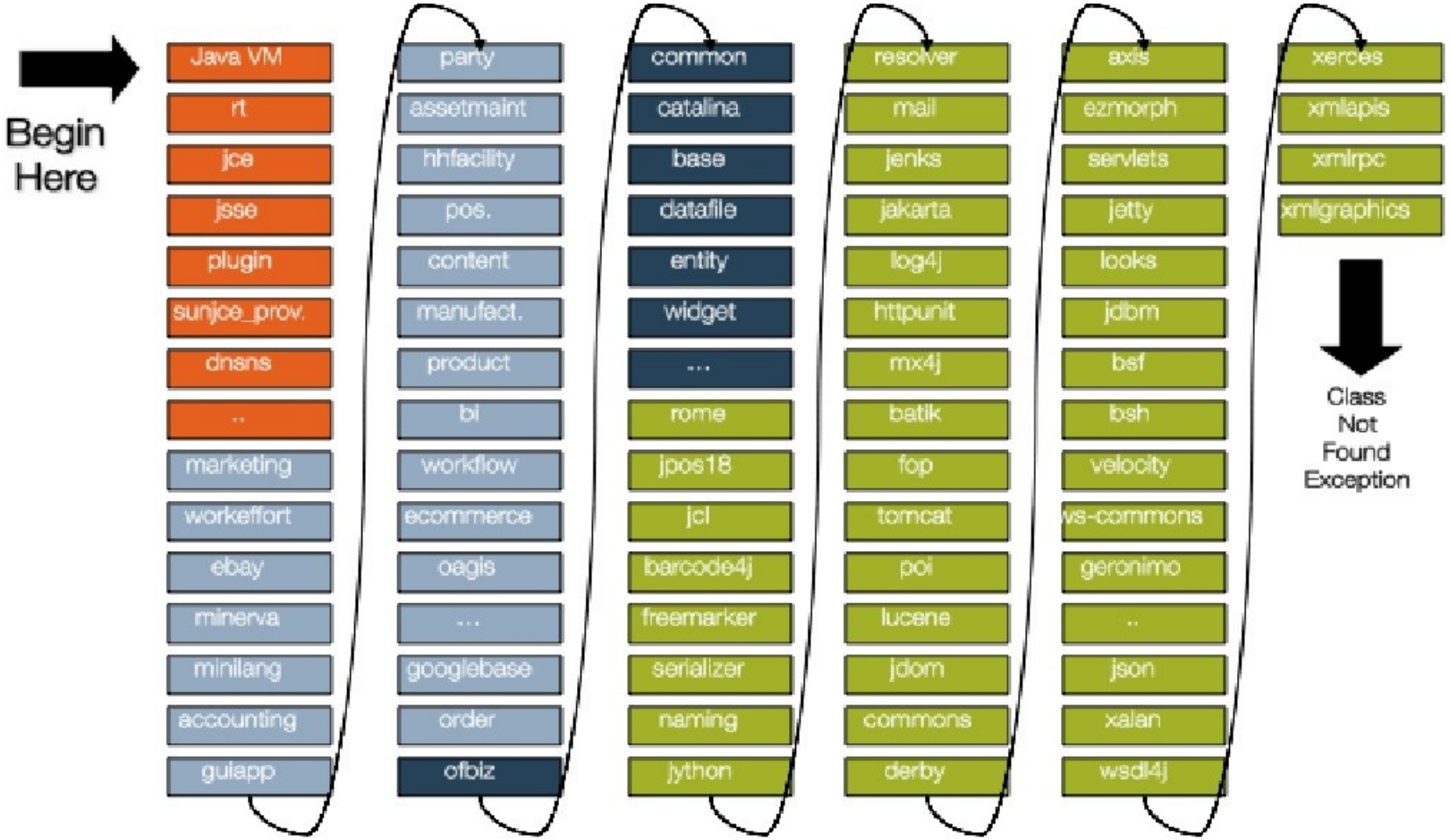
Problems with JAR files!



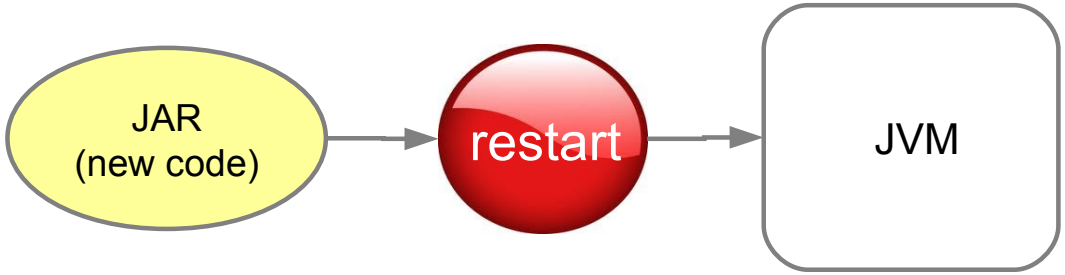
- No modularization characteristics:
  - No scoping
  - No way to declare dependencies
  - No versioning
  - No indication of interfaces provided (or implemented/consumed)

Classpath clutter

- Classpath clutter, JAR hell, ClassNotFoundException



JVM requires restart

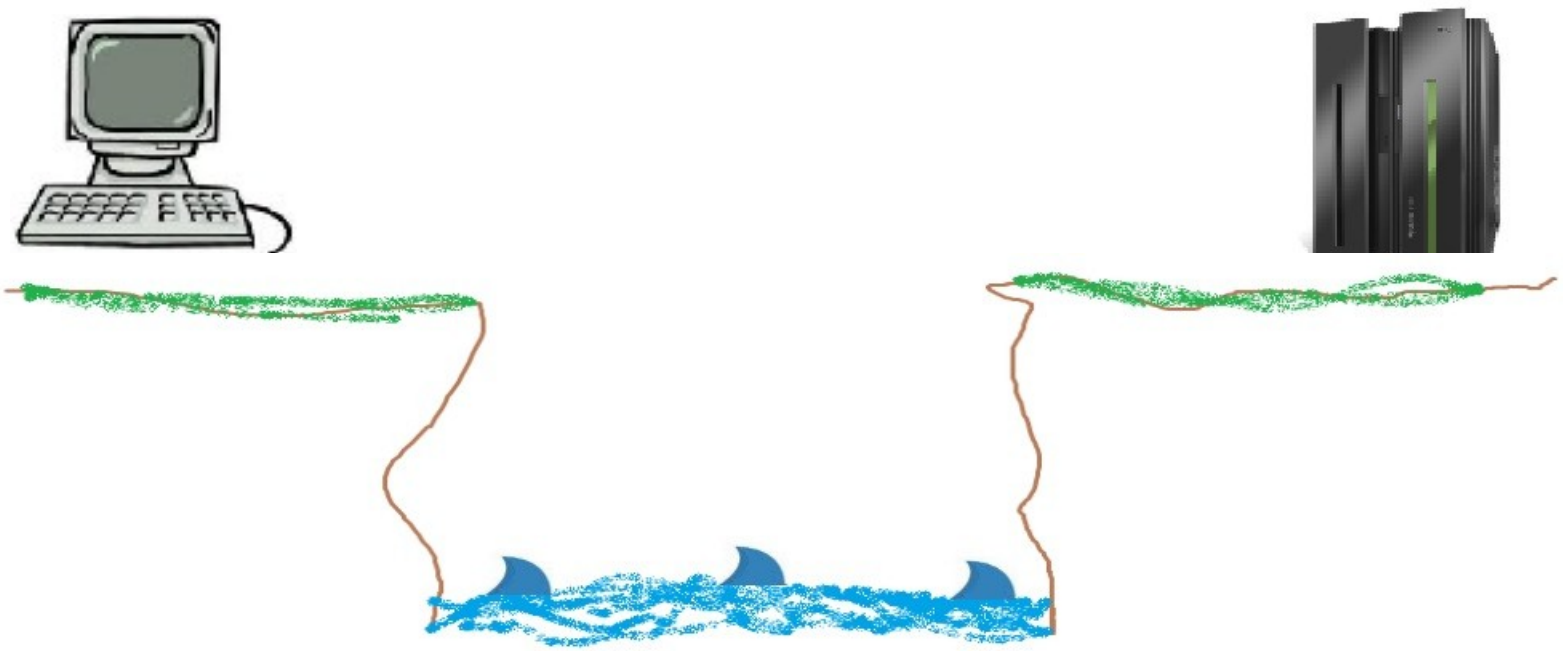


- Adding a new application, requires **JVM restart**

No Development/Deployment path!

PC Client Development

CICS TS

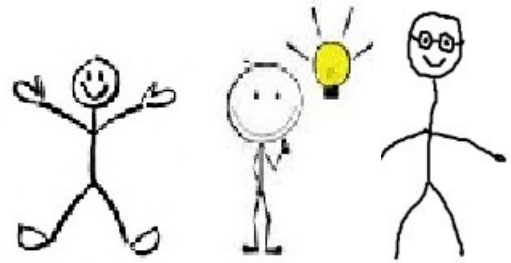


- No standard development or **deployment** tools for Java code in CICS!

OSGi and the JVM server

Open Services Gateway initiative (OSGi)

"Modular development"



"Dynamic update"

"versioning"

...and then one day a delegate of **three** colleagues knocked at the door...

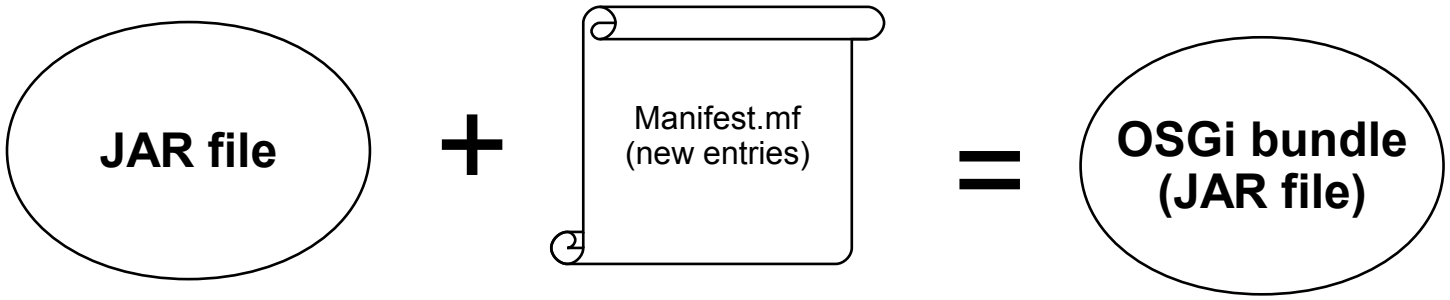
"Have you ever heard of OSGi?"

What is OSGi?

*"The **OSGi framework** is a module system and service platform for the Java programming language that implements a complete and dynamic component model, something that does not exist in standalone Java/VM environments. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot"*

Wikipedia

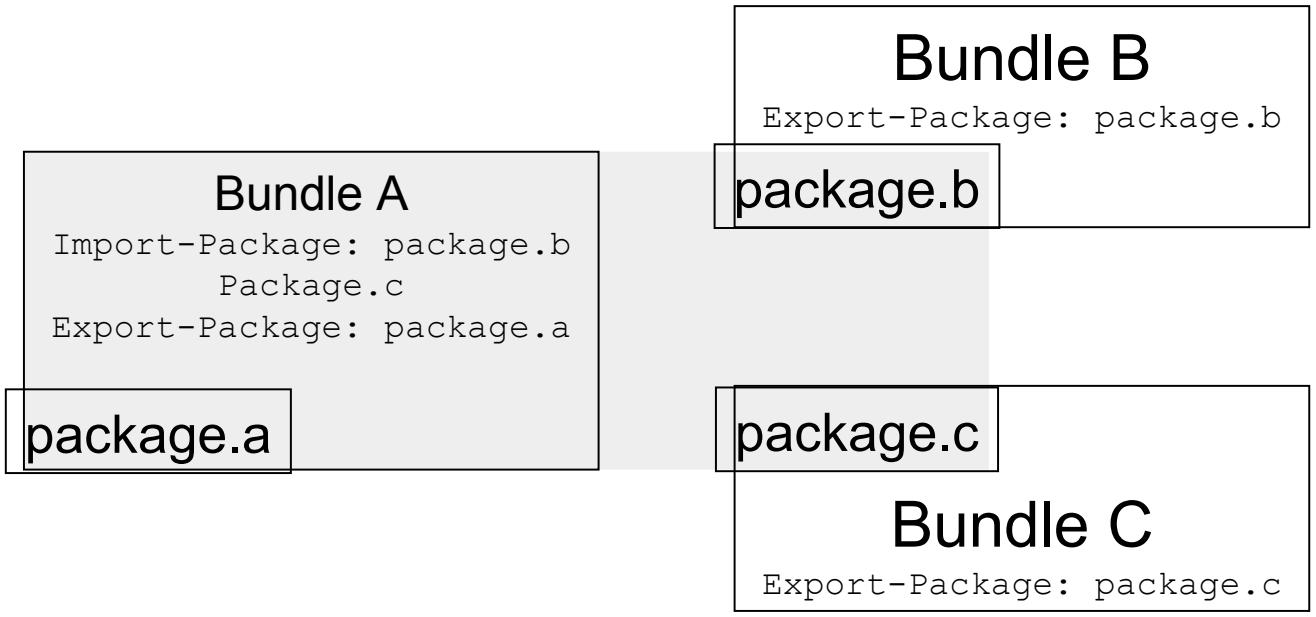
What is an OSGi bundle?



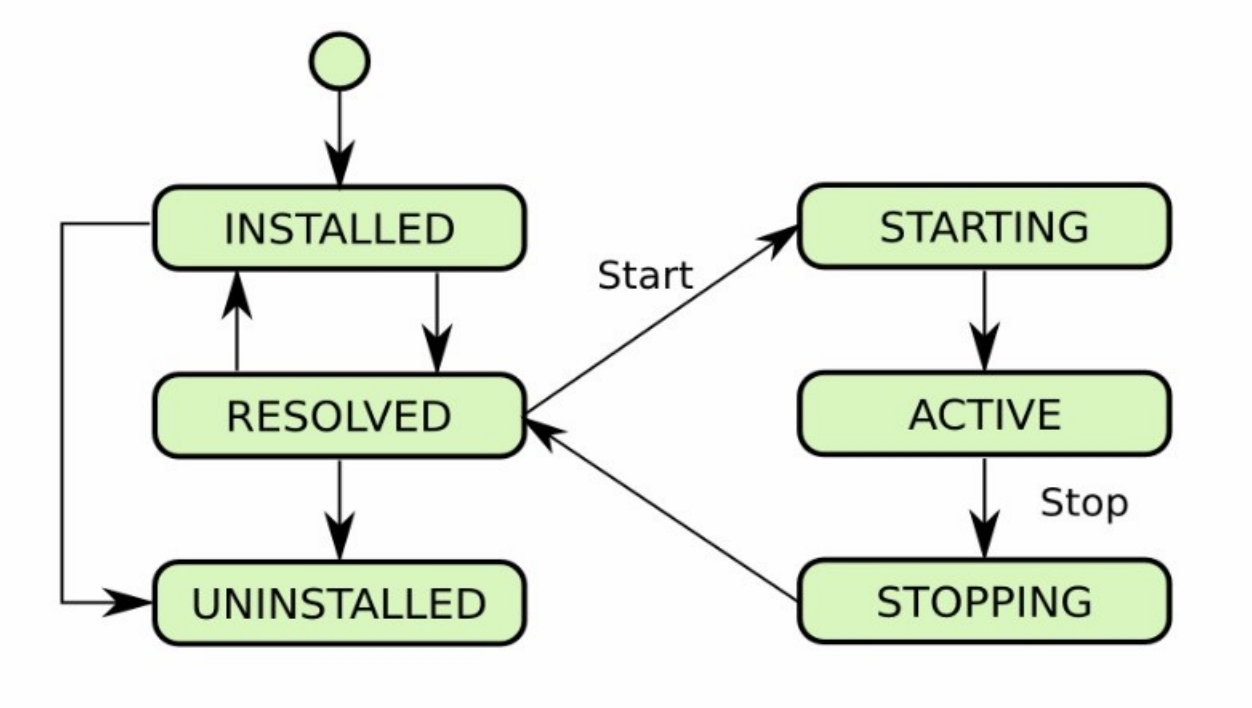

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: CICS TS OSGi Interfaces
Bundle-SymbolicName: com.ibm.cics.osgi;singleton:=true
Bundle-Version: 1.0.0
Bundle-Vendor: IBM Corporation
Export-Package: com.ibm.cics.osgi;version="1.0.0"
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Import-Package: com.ibm.cics.domains;version="[1.0.0,2.0.0)",
 org.osgi.framework;version="[1.3.0,2.0.0)"
```

Class loading with OSGi

- No more CLASSPATH
- Each bundle has its own class loader
- 'Class space' - the set of classes reachable by the bundle (not just its own class loader)
- Delegate to other bundle's class loaders for 'Imported' packages



OSGi Bundle Life-cycle



- Installation and resolution steps ensure that once an OSGi bundle is STARTED – all dependencies have been resolved – i.e. **No more ClassNotFoundException!**

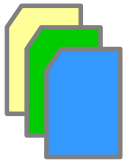
Goals of OSGi and JVM server

1. **Allow existing Java programs to run unchanged in JVM server:**
   - No recompiling, just repackage
   - Provide the same Program linkage and JCICS API as before
2. **Allow a program to be installed, updated and discarded without restarting JVM**
3. **Allow multiple versions of the same program or library to be installed concurrently in the same JVM**
4. **Improve the End-to-end experience for a Java application developer**

Repackage

Recycle

Version

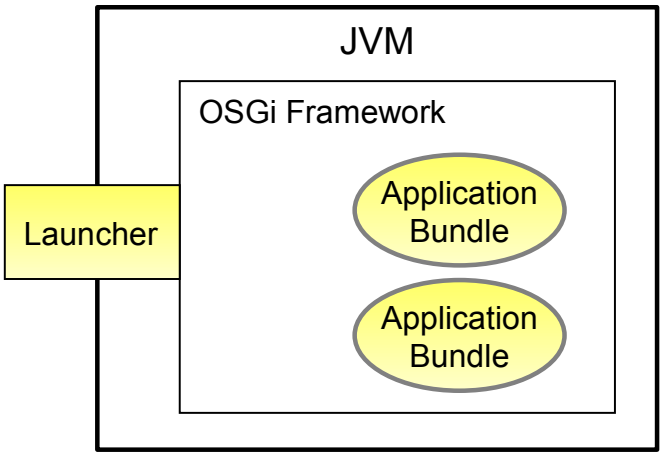


Develop/Deploy

Technical ingredients...

- Embed **Equinox** OSGi framework into the JVM server
- Extend CICS bundles to contain OSGi bundles as '**bundleparts**' and provide lifecycle
- CICS Java applications written and deployed as OSGi bundles
- CICS Explorer SDK (Eclipse Plug-in) gives full Java DEVELOPMENT environment
- CICS Explorer SDK controls deployment to z/FS (z/OS filing system for Unix System Services)

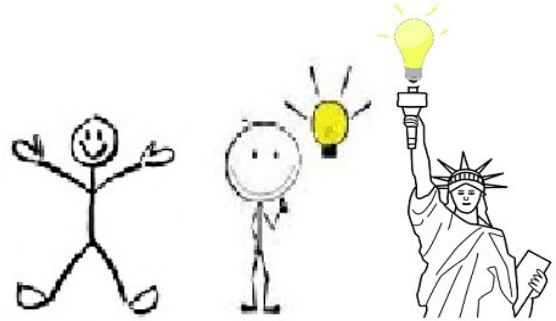
OSGi enabled JVM server



- Application update without JVM restart (Dynamic update)
- Multiple concurrent versions of an Application
- Dependency Resolution (reduced runtime failures - ClassNotFoundException)
- Defined Interfaces (Import/Export Package) and Modular development

WebSphere Application Server - Liberty Profile

WebSphere Liberty Profile (WLP)



"Java EE"

"Portability"

"Web-Apps"

"EJBs"

"Servlets/JSPs"

"Did you hear about the new lightweight profile of WebSphere Application Server? - it's written to run in an OSGi framework!"

Liberty is...

A LIGHTWEIGHT

COMPOSABLE

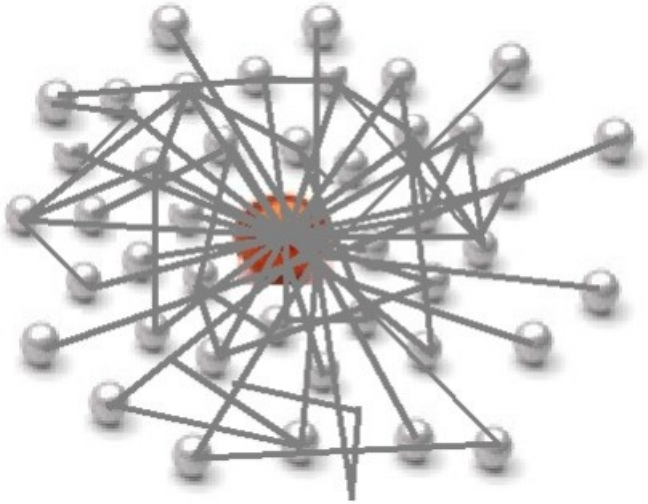
FAST

…'Profile' of WebSphere Application Server

(c.f. Application Servers such as GlassFish, Jboss, and Jetty)

Composable – you pick the Features

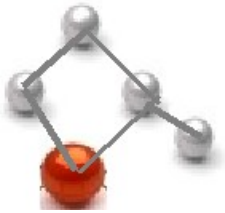
If this is Traditional WAS (tWAS)...



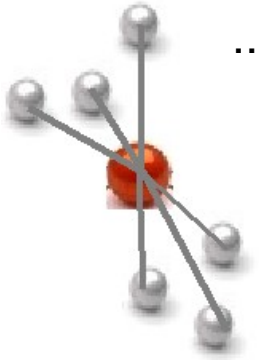
WebSphere®

...this is Liberty (WAS)

...so is this



...or even this!



Easy configuration

- Everything runs 'out of the box', you only configure, when you want to change something...***"Configuration by exception"***.
- This is the entire configuration needed to run Liberty as a Web-container with Servlet support.

```
<server description="new server">

   <featureManager>
      <feature>servlet-3.0</feature>
   </featureManager>

   <application id="BasicWeb" location="BasicWeb.war"
      name="BasicWeb" type="war"/>

</server>
```

The “Liberty” Profile

- A *sub-set* of WAS capabilities (JEE 6 Web profile container)
- Written entirely in Java, runs within an OSGi Framework
- Capabilities are offered through pluggable 'Features'
- Support for JEE APIs such as JNDI, JTA, JDBC, JPA, EJBlite, JMS
- Support for JSON, JAX-RS (RESTful API), JAX-WS (Web-services)

Liberty in CICS. Plain sailing?

- CICS has an industry standard Java environment AND supports OSGi
- Liberty Profile is written entirely in Java and runs in an OSGi framework
- We can just embed Liberty as an OSGi application into CICS - right?

Sadly not.

A few challenges...

- Liberty is not written entirely as OSGi bundles – it consists of a JVM launcher, OSGi bootstrap code, and pre-OSGi classloader logic
  - *Bypass JVM launcher logic, drive Liberty bootstrap within our own JVM*
  - *Repackaged our CICS OSGi controller logic, and deploy as a Liberty Product Extension, giving us a measure of control over the OSGi framework (The Trojan!)*

- TCP/IP listener for Web-requests gives a non-CICS entry point onto non-CICS enabled threads....i.e. no JCICS API, no CICS data, no CICS program link. **No point!**
  - *Java concurrency package and OSGi registry allow us to override the ExecutorService, replacing it with a CICS one*
  - *The CICS ExecutorService uses a 'reverse' ECB solution to trigger CICS to start a new task, drive it into the JVM as a thread, and run some dispatching logic to match up the Runnable (workload) with the CICS enabled thread*

- Fixpack roulette. The internal challenges of acquiring bug-fixes to another product within a large corporation like IBM!

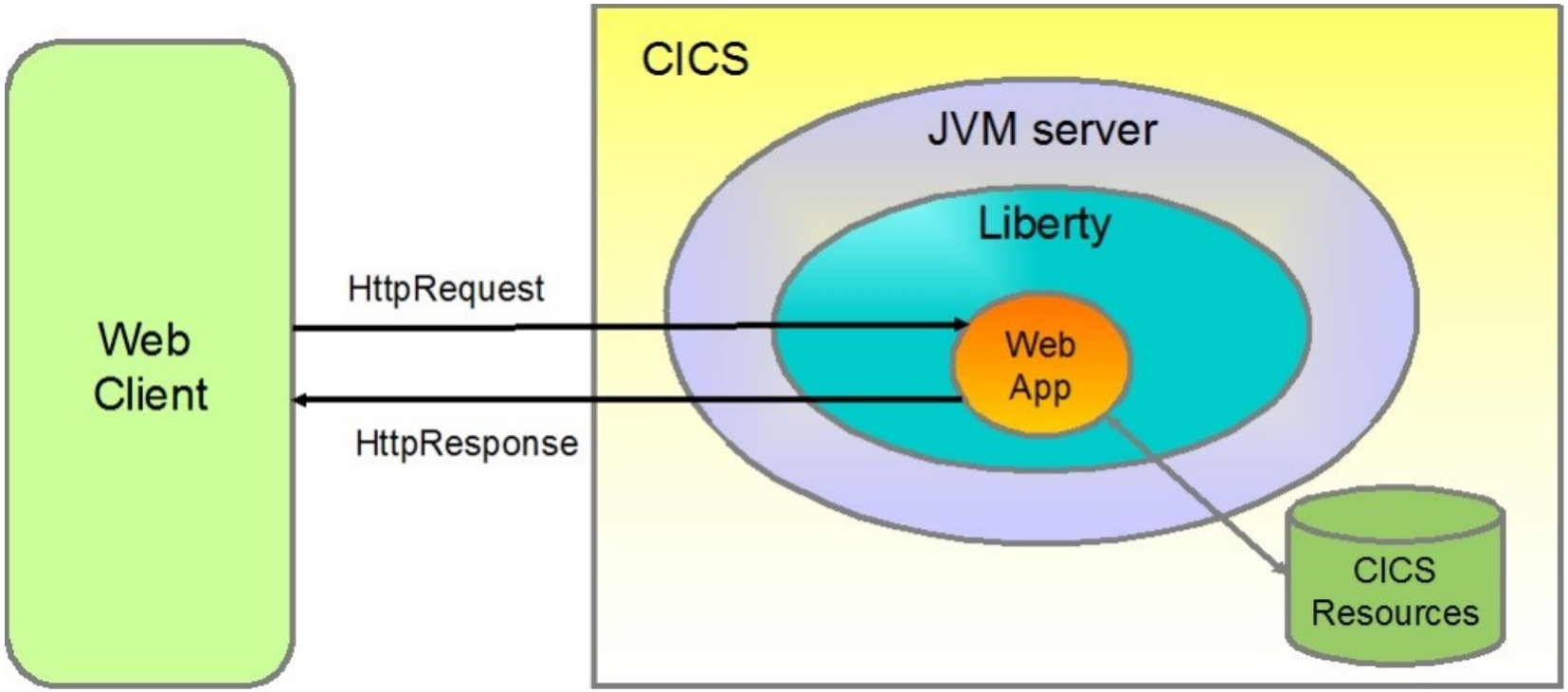
- Security Integration
  - *delegate to the angel process for SAF credentials*

- “ThreadIdentityService” - thank goodness for Architects and Extension points!

Technical ingredients...

- Embed a **WebSphere Liberty Profile Server** in our JVM
- Use Liberty 'Product Extension' to inject CICS code and get a measure of control
- Replace ExecutorService, causing Liberty applications to run as CICS tasks
- Use **CICS bundle resources** to control dynamic lifecycle of Web-applications
- Integrate with CICS transactions (UOW) and Security - for Enterprise class service
- Provide CICS Explorer SDK and WebSphere Developer Tooling (WDT) giving full Java and Web DEVELOPMENT environment

Fast. Lightweight. Local.



Ebejeebies!
(EJBs, EARs, WARs, WABs & EBAs)

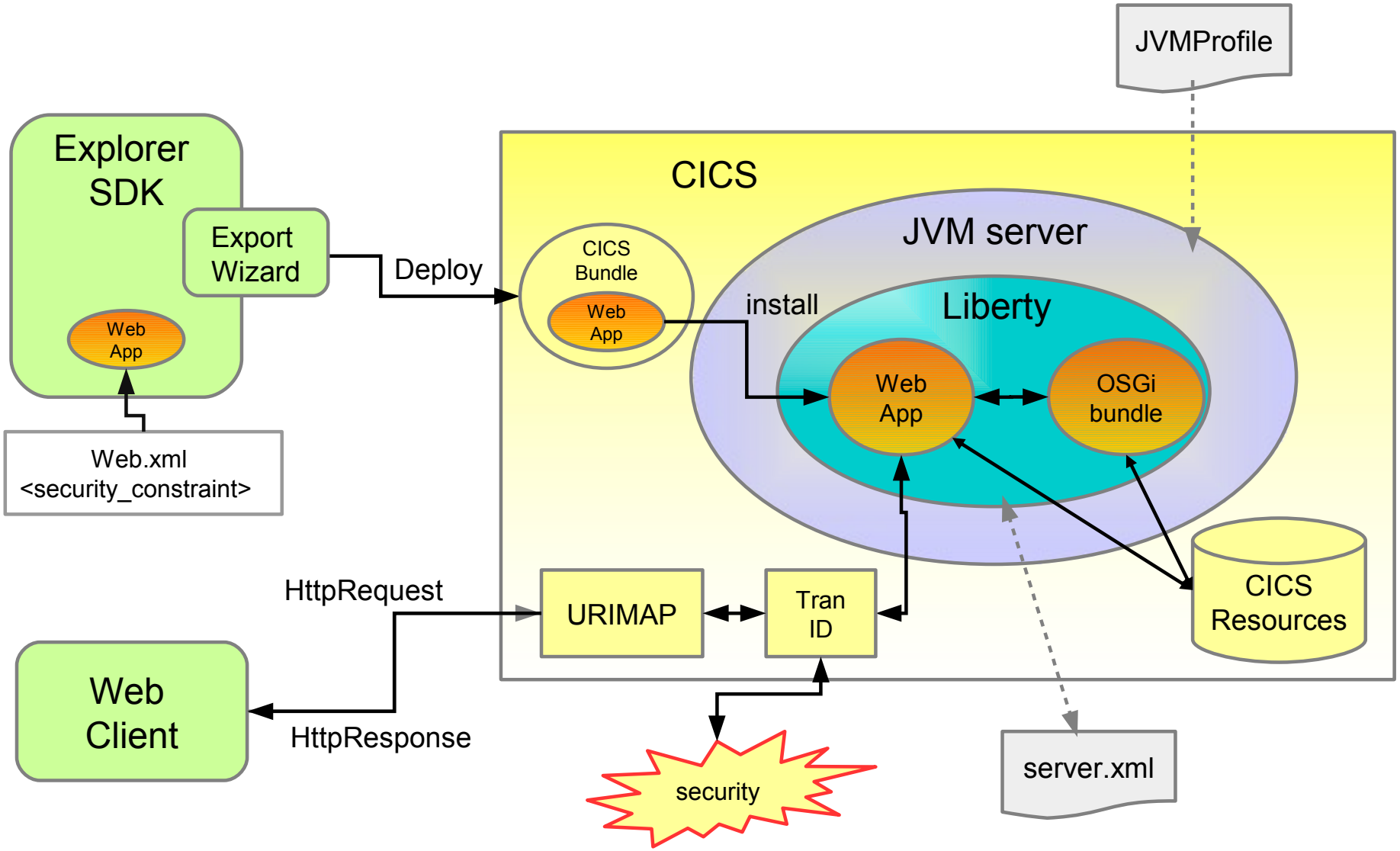
- Dynamic Web Applications are deployed as WARs
- Enterprise Applications and EJBs are deployed as EARs
- OSGi Applications are deployed as Enterprise Bundle Archives (EBAs)
- OSGi Applications can contain OSGi bundles and Web Application Bundles (WAB)
- EBAs provide isolation from each other, with access to 'common' bundles.
- Servlets can use the exported packages of OSGi bundles in the same EBA, allowing presentation logic to reside in the Servlet, and business logic in the companion OSGi bundle(s).

Benefits to CICS

- Embedded in CICS JVM server, provides “off the shelf” Web-server capabilities (JSP/JSF and Servlets)
- Java EE Applications have direct, local, access to CICS data and resources (no adapters or connectors)
- JAX-RS and JSON can be used for lightweight access to CICS data
- JAX-WS can be used to expose CICS services as Web-services
- Servlet/JSP/JSF can “front-end” existing CICS applications (and use JCICS program LINK) or link to back-end logic implemented in standard portable EJB* applications

*EJBlite

High Level Application overview



Summary of Key Benefits

Local. Lightweight. Fast. Web Applications run locally in CICS with direct access to CICS data and resources. No adapters, no converters, same address space.

Standard tools for developers. Familiar, industry standard tools with Eclipse and Dynamic Web Projects. CICS Explorer SDK enhances the deployment experience.

Portable. Presentation logic in Servlets, business logic in OSGi bundles. Servlets are portable across runtimes. Bundles provide componentization.

Modular design. Architected in a modular way using OSGi, the server only enables and starts the features required by the applications and configuration. If you're not using a feature, it won't start in your server runtime

Dynamic runtime. Features can be added to the server dynamically, using the OSGi framework, while the server is running, with zero downtime and server restarts. Similarly server and application config can be updated without the need to restart.

Eclipse based tools. The eclipse tools for the Liberty Profile are small and very well integrated with the Liberty Profile environment

Why run Web-Apps in CICS?

- **Robustness, scalability, security, & centralisation** of CICS/Mainframe
- "**Centre of gravity**" of data and business logic
- No network Latency, **no connectors**, runs in same address space
- **zIIP** advantage
- Portability of Applications
- Supports Mobile and RESTful Workloads
- ***"Because it's there" :-)***

Questions?