HOW DENSE IS THE CLOUD OF OSGI?

Tom Watson

IBM

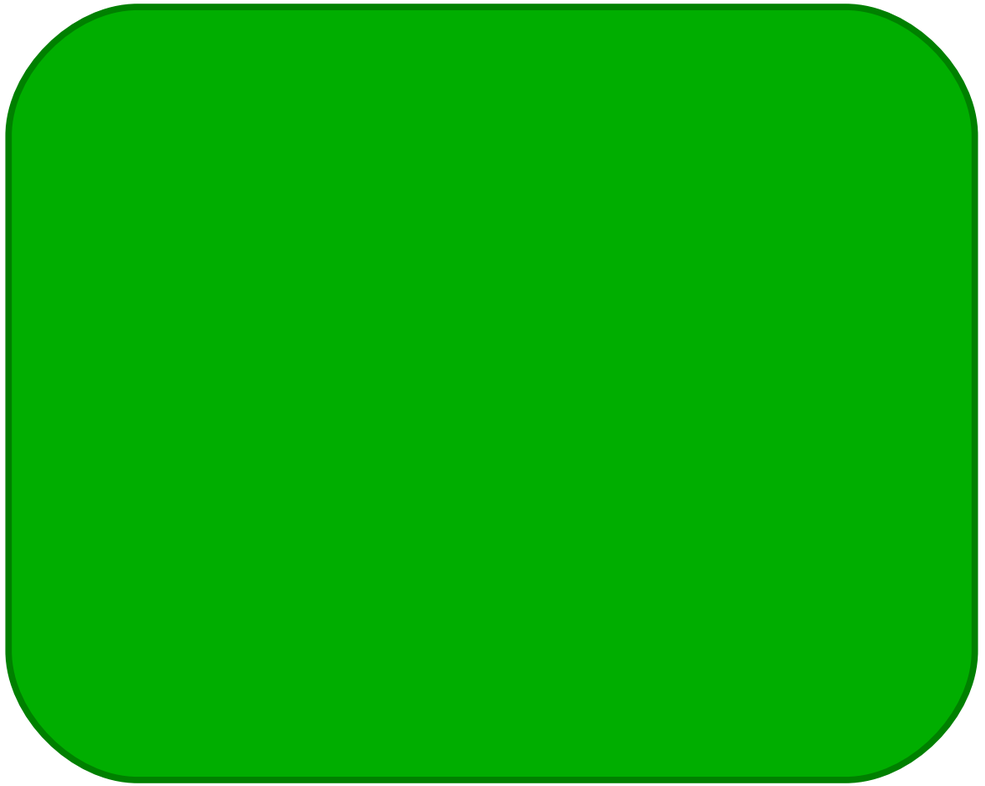
June 11th 2014

Agenda

- Applications and OSGi
- Cost of deploying an application
- Sharing resources, density and multi-tenant VMs
- Equinox enhancements to increase density
- Limitations

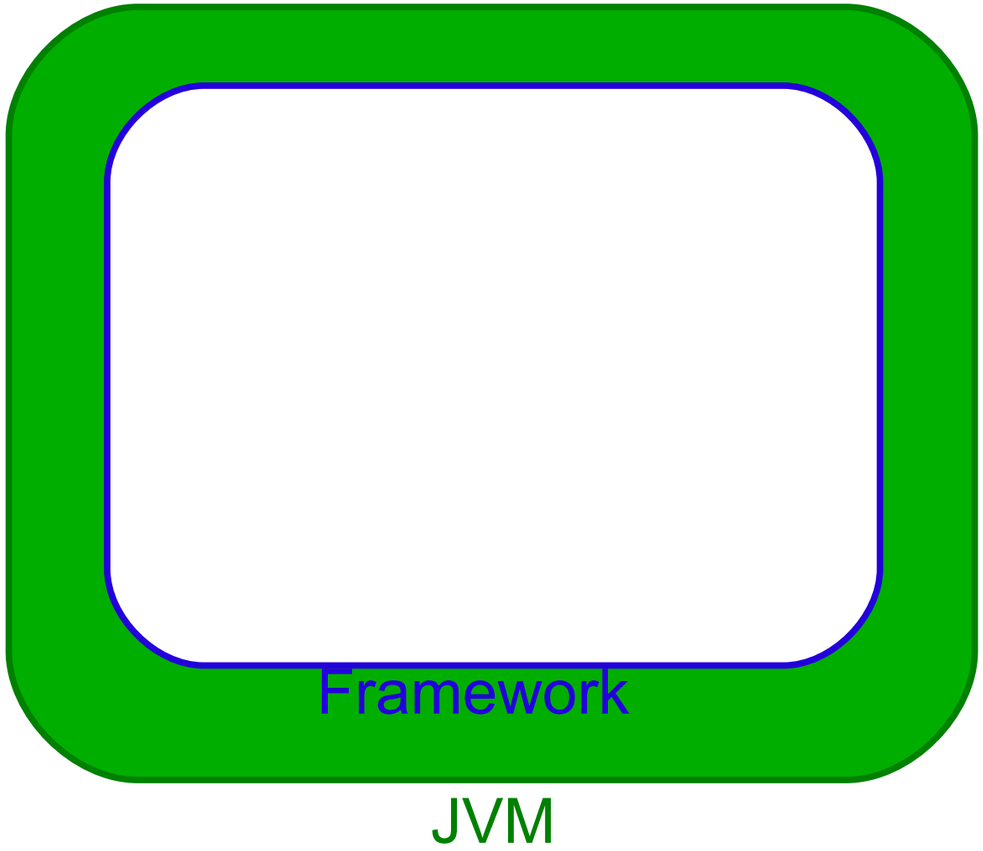
OSGi Applications

- What is Required to Run
  - JVM



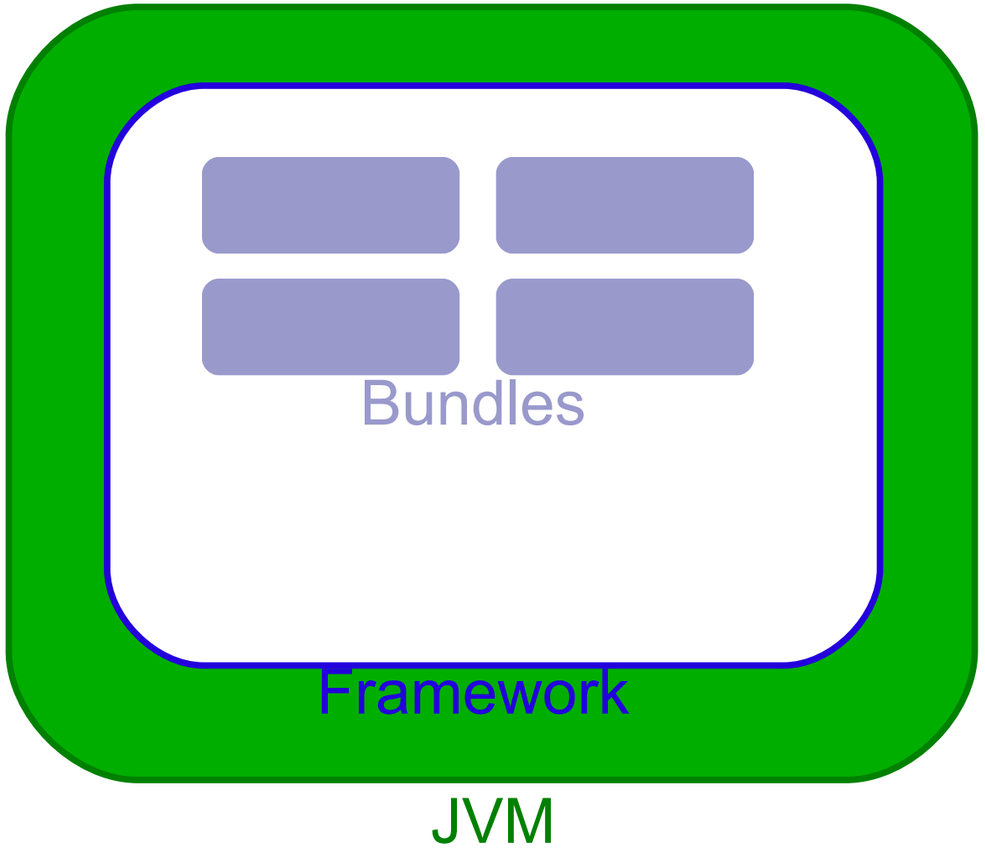
OSGi Applications

- What is Required to Run
  - JVM
  - OSGi Framework



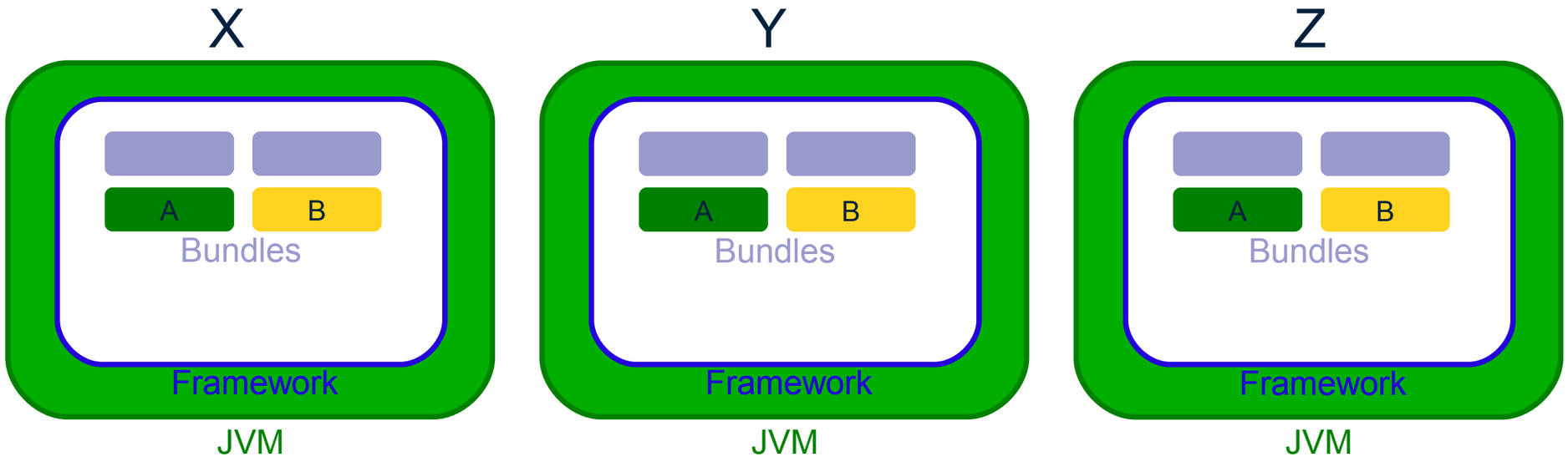
OSGi Applications

- What is Required to Run
  - JVM
  - OSGi Framework
  - A set of bundles



OSGi Applications – Cost

- Each OSGi application instance pays the overhead cost
  - JVM
  - Framework
  - A set of bundles

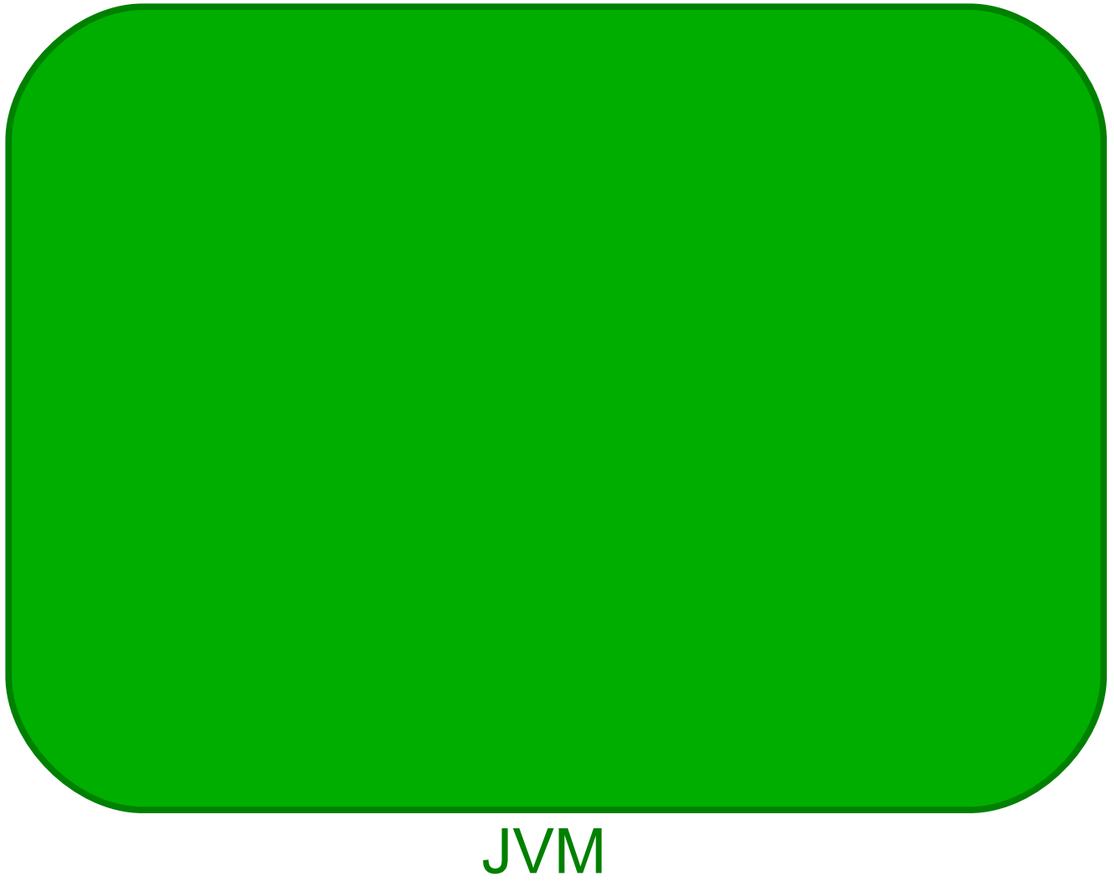


Multi-tenancy vs. Multi-instance

- Multitenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client organizations (tenants).
- Multitenancy is contrasted with a multi-instance architecture where separate software instances (or hardware systems) are set up for different client organizations.
- With a multitenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance.

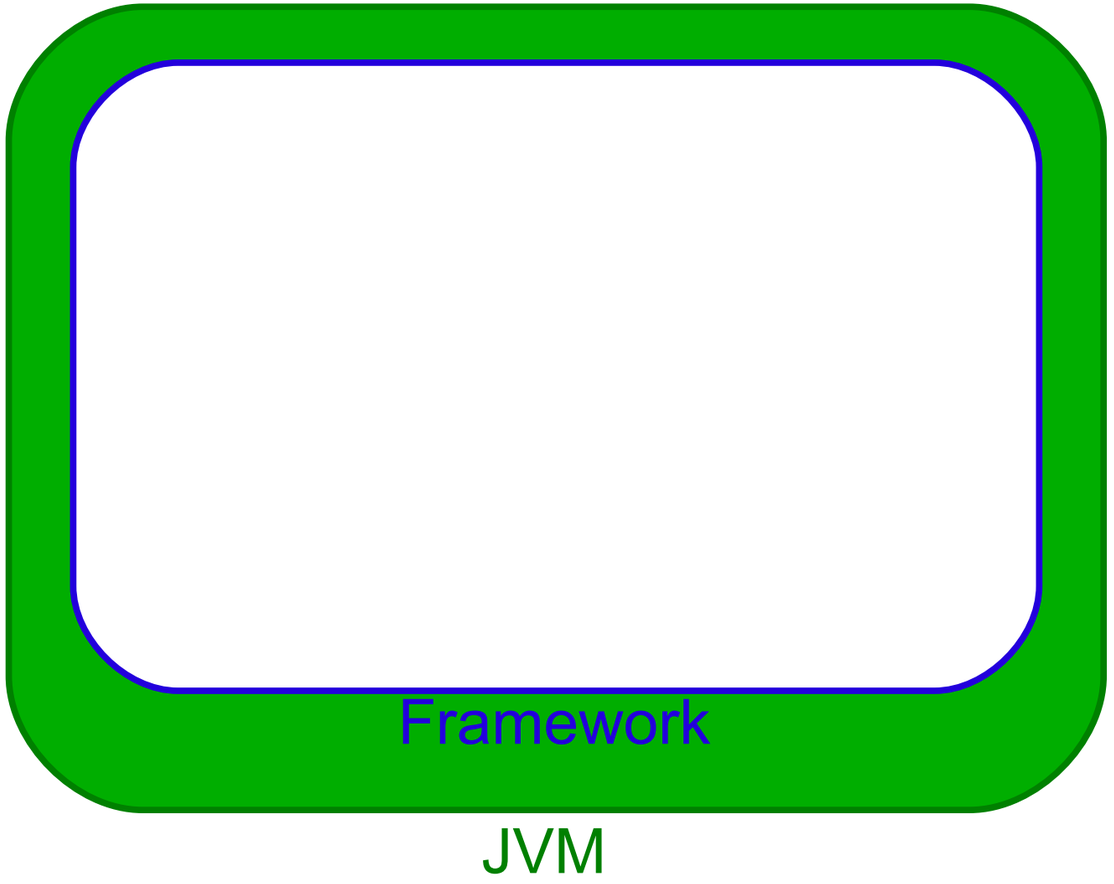
OSGi Applications - Subsystems

- OSGi Subsystems can help
  - Single JVM



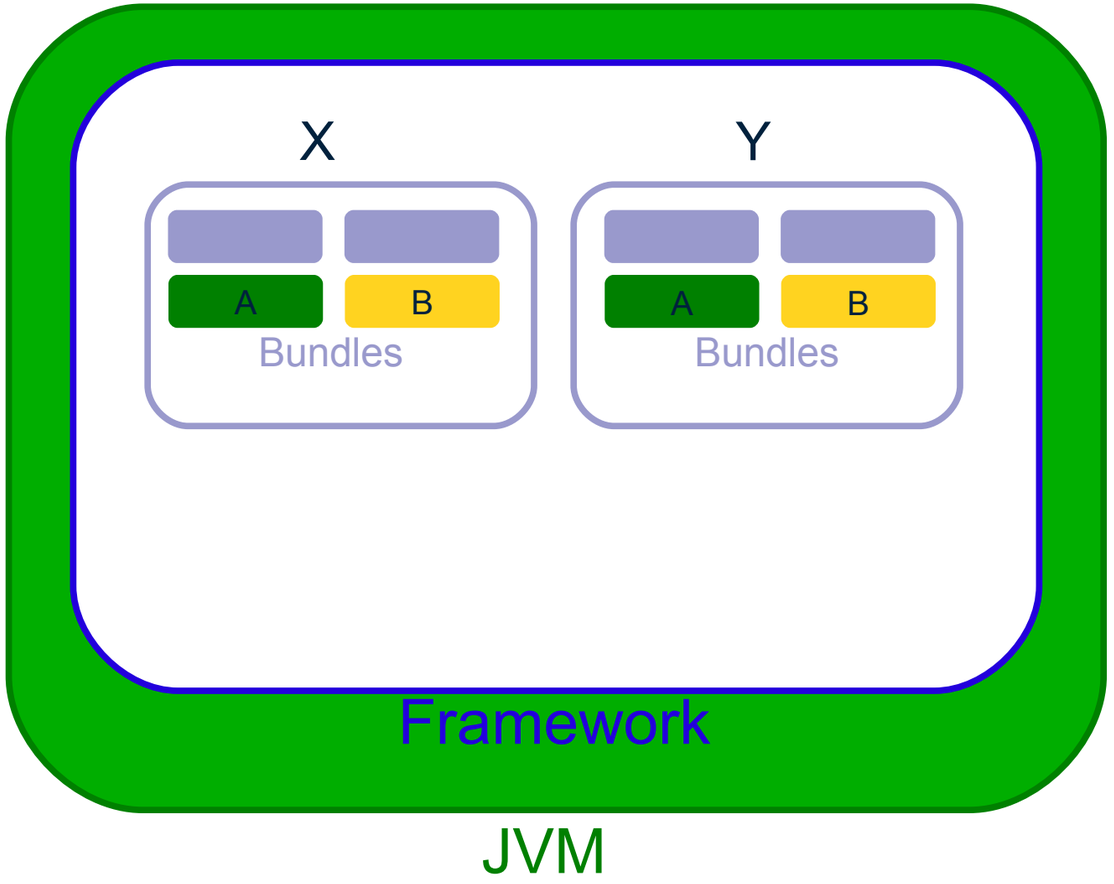
OSGi Applications - Subsystems

- OSGi Subsystems can help
  - Single JVM
  - Single OSGi Framework



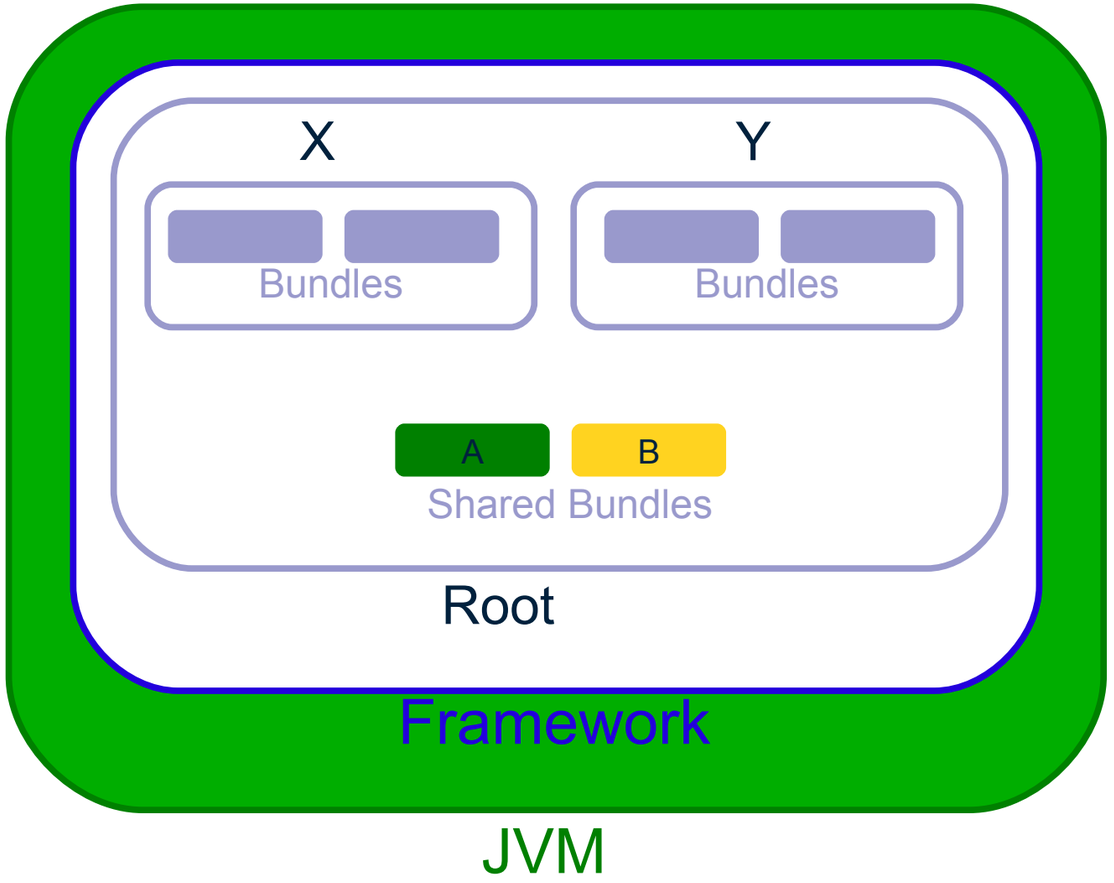
OSGi Applications - Subsystems

- OSGi Subsystems can help
  - Single JVM
  - Single OSGi Framework
  - Applications are isolated with a subsystem (tenant?)



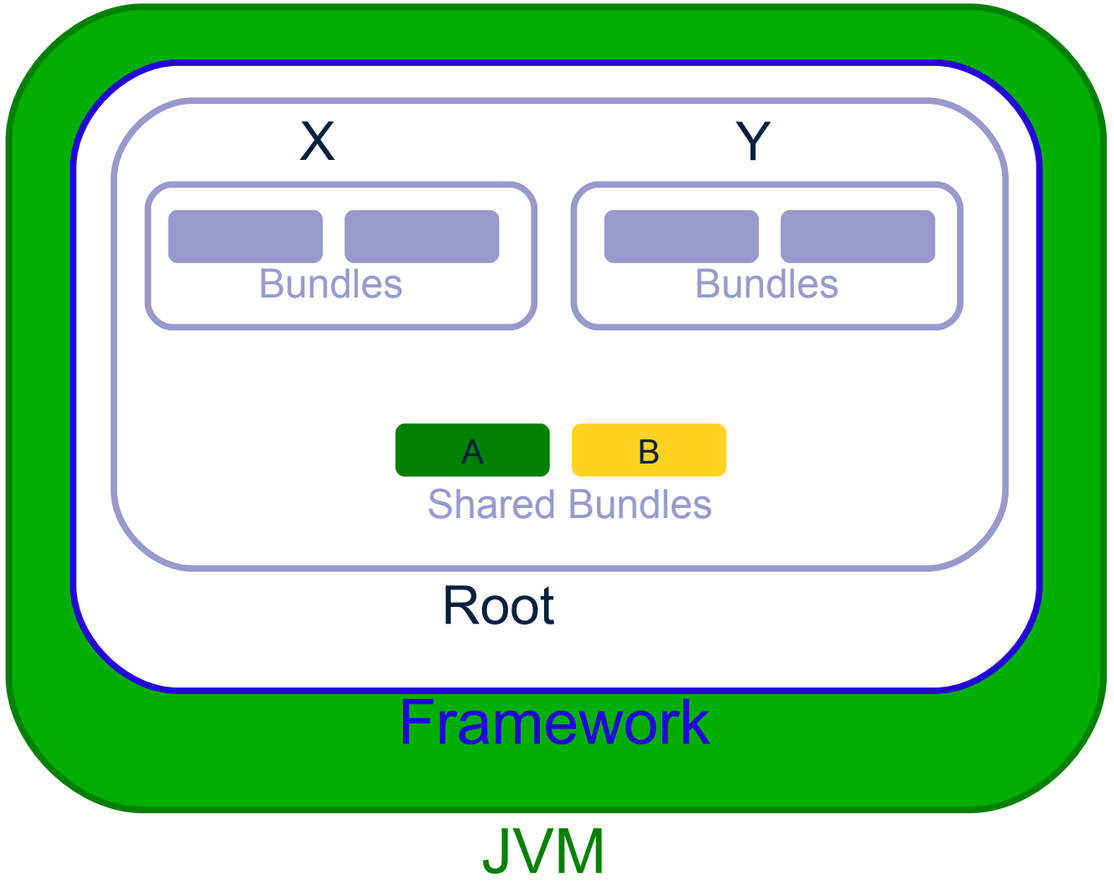
OSGi Applications - Subsystems

- OSGi Subsystems can help
  - Single JVM
  - Single OSGi Framework
  - Applications are isolated with a subsystem (tenant?)
  - Packages and services can be shared directly



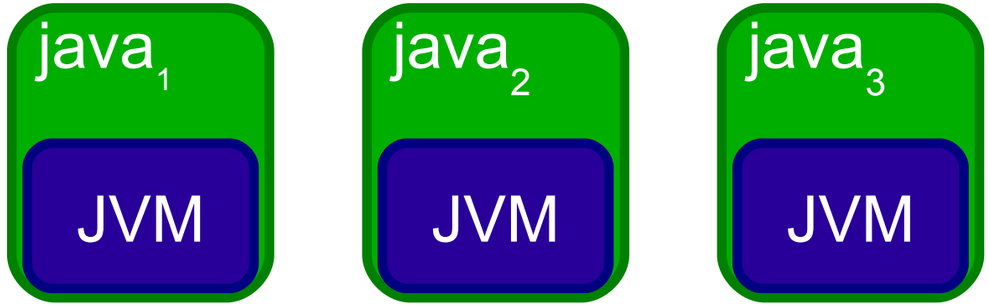
OSGi Applications - Subsystems

- Advantages
  - Maximize sharing
  - Directly share OSGi Services and other shared objects.
- Disadvantages
  - Applications must play nicely with shared resources
  - Unable to monitor resource consumption
  - Requires modifications to component models to understand subsystems
    - DS, Blueprint, Config ...



Multi-tenant JVM

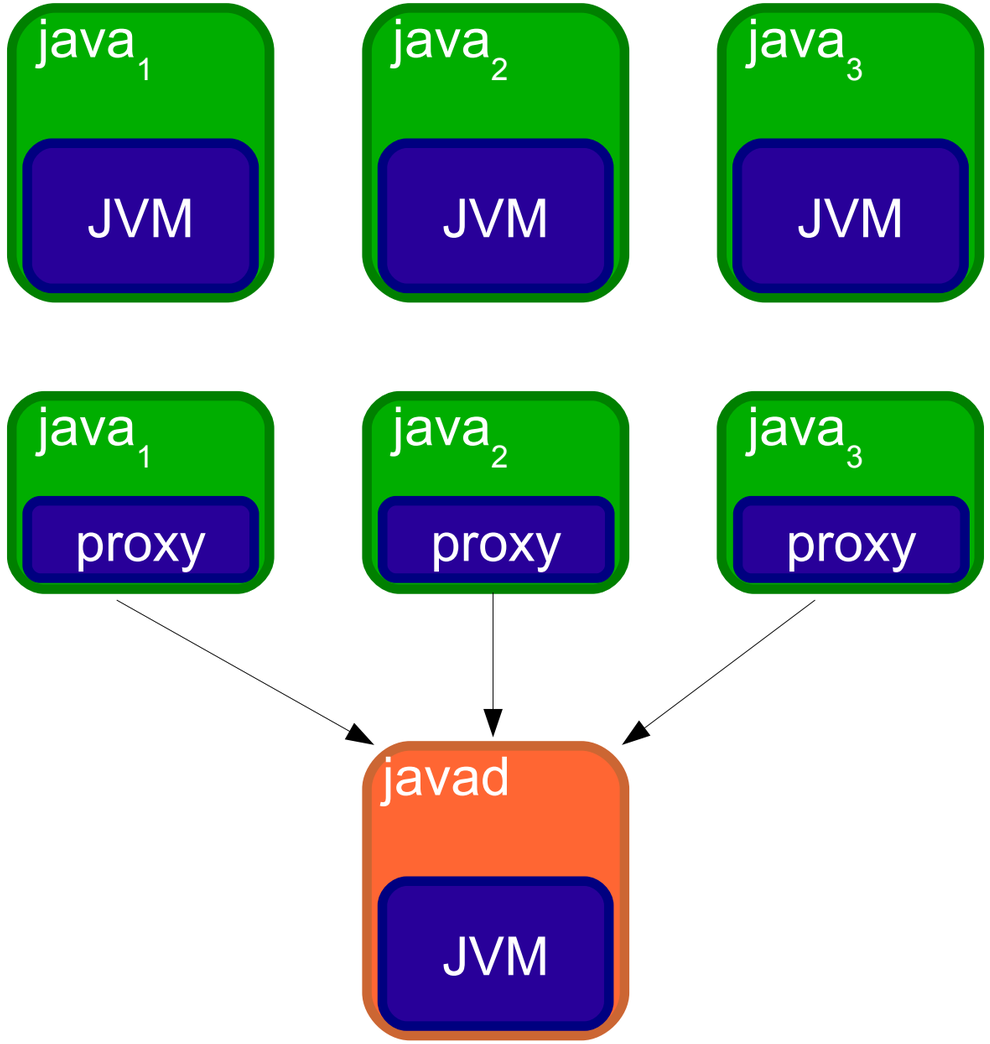
- A standard `java` invocation creates a dedicated (non-shared) JVM in each processes



Multi-tenant JVM

- A standard `java` invocation creates a dedicated (non-shared) JVM in each processes

- IBM's Multitenant JVM puts a lightweight 'proxy' JVM in each `java` invocation. The 'proxy' knows how to communicate with the shared JVM deamon called `javad`
  - `javad` is launched and shuts down automatically
  - No changes required to applications
  - `javad` process is where aggressive sharing of runtime artifacts happen

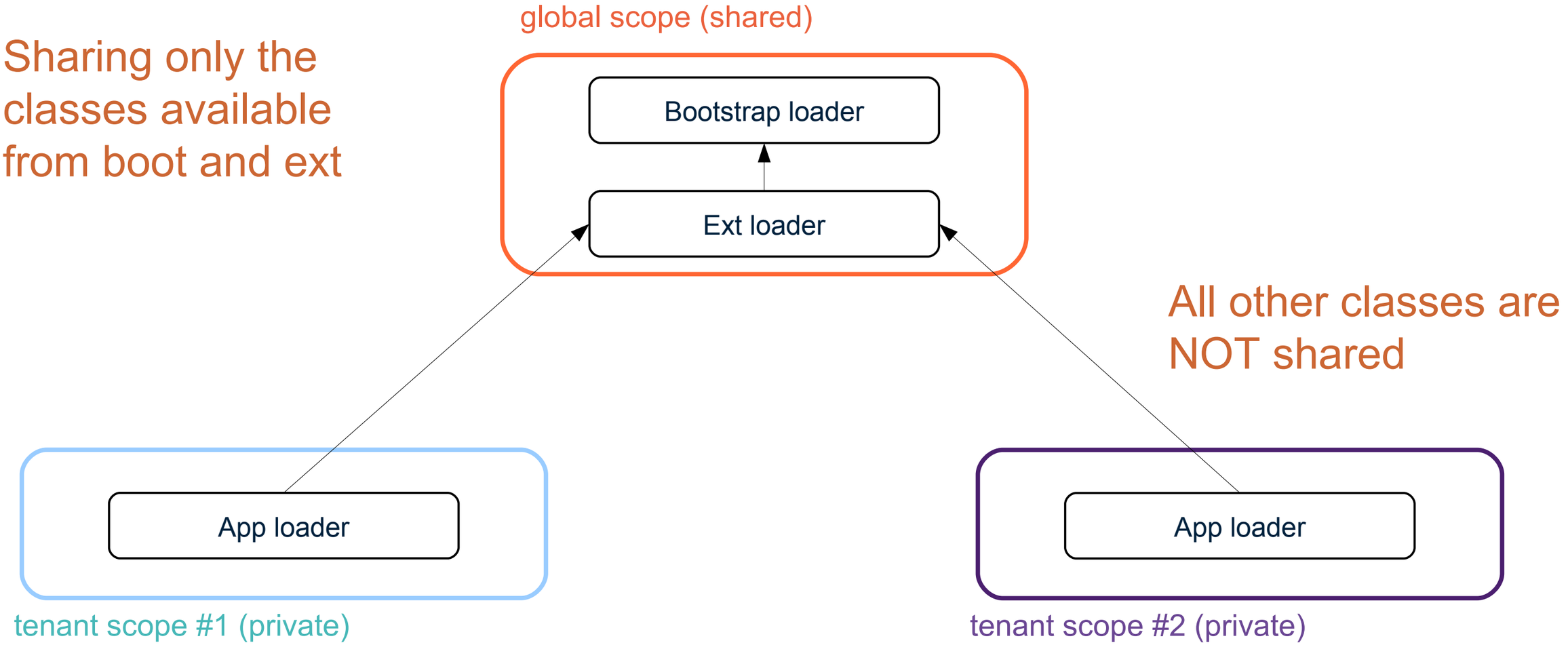


Multi-tenant JVM – Tenant Isolation

- Each tenant behaves like a dedicated JVM
  - Static APIs only effect the tenant they are being called from: `System.exit`, `System.getProperty`
  - No visibility to other tenant threads
- Sharing occurs at the class loader
  - Eliminates loading of duplicate classes that are shared across applications
  - Allows the sharing of compiled (JIT) code
  - A single GC instead of one for each JVM instance

Multi-tenant JVM – Class Loaders

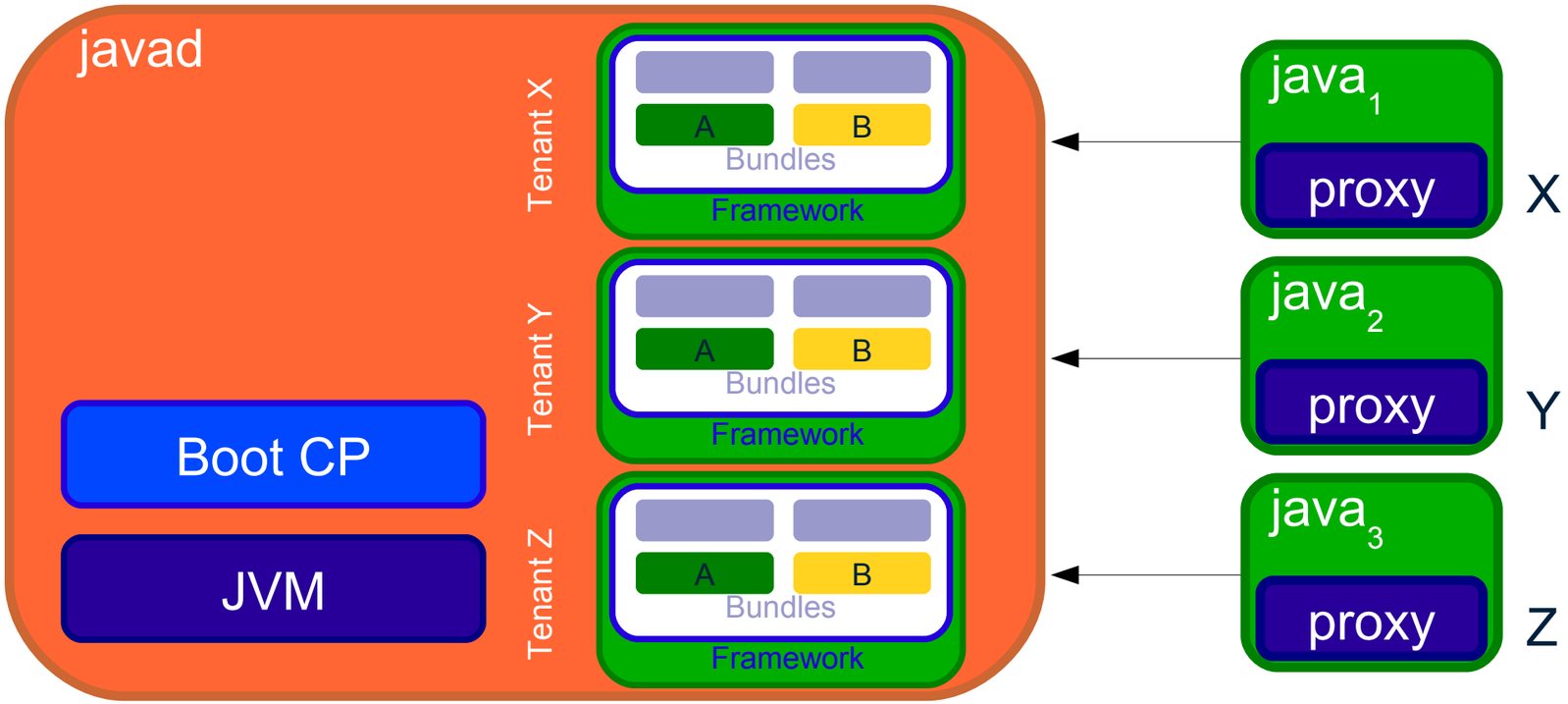
Sharing only the classes available from boot and ext



All other classes are NOT shared

Multi-tenant JVM – OSGi Applications

- Each OSGi application instance pays the overhead cost
  - ~~JVM~~ – now sharing the JVM, boot and ext class loaders
  - Framework – Loaded above the JVM, not shared
  - Bundles – Loaded above the framework, not shared



Increasing Density

- Multi-tenant JVM increases density by loading more classes with class loaders in the shared scope
- Boot and ext class loaders can be shared because they are consistent across tenants
- OSGi is dynamic!
  - Bundles are resolved at runtime and may get different wirings depending on what is installed
  - Dynamic imports allow the wiring for a particular bundle to change over time
  - OSGi weaving hooks may weave bytes differently across tenants

What Can We Share?

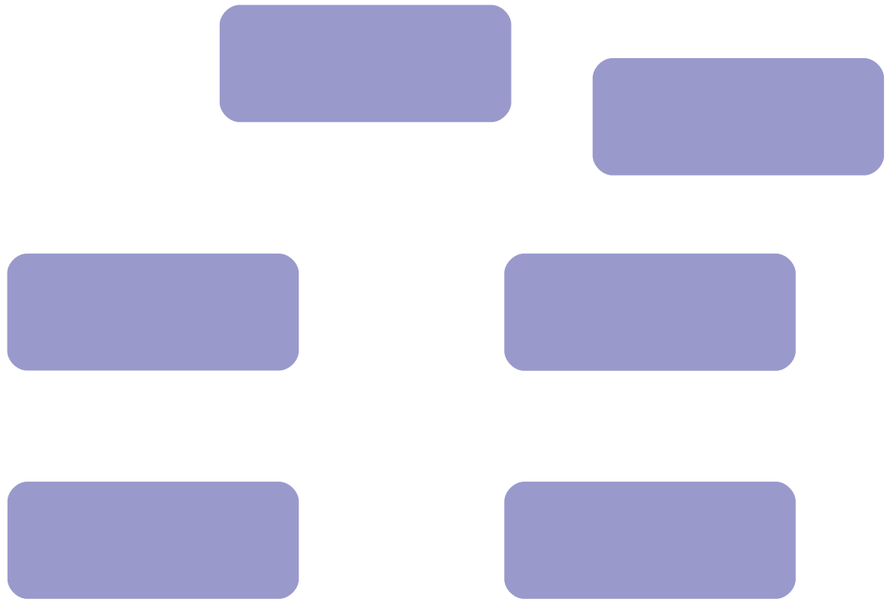
- The launcher and the framework implementation
  - Adding the launcher and framework implementation to the boot or ext loader allows the framework implementation classes to be shared
- But most OSGi applications (hopefully) have more code than the framework implementation has
  - What rules are required to allow OSGi application classes to be shared?
  - What limitations do these rules impose on the bundles that can be shared?

OSGi Loaders

- A subset of declared requirement types effect the OSGi class loader
  - The package, bundle and host requirements all have an effect on the overall OSGi class loader graph.
- Resolved bundles form a directed graph between the requirers and the providers
  - Connections are referred to as wires
  - A wiring for a bundle is the complete list of required wires
  - If the wiring for a bundle is identical between multiple tenants then it has potential for sharing

OSGi Loaders – Unique Wirings

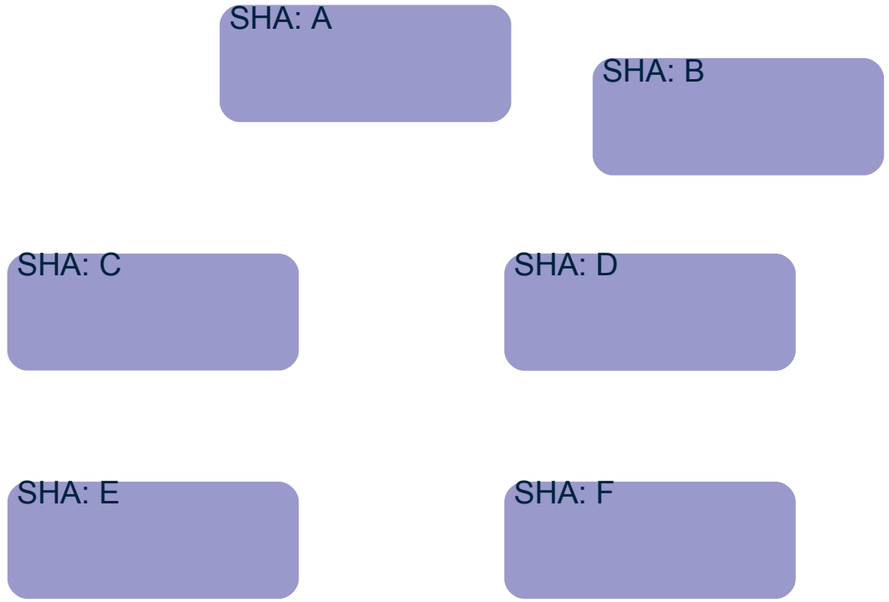
- Identify bundle resources
  - Calculate the SHA of the content



Bundles

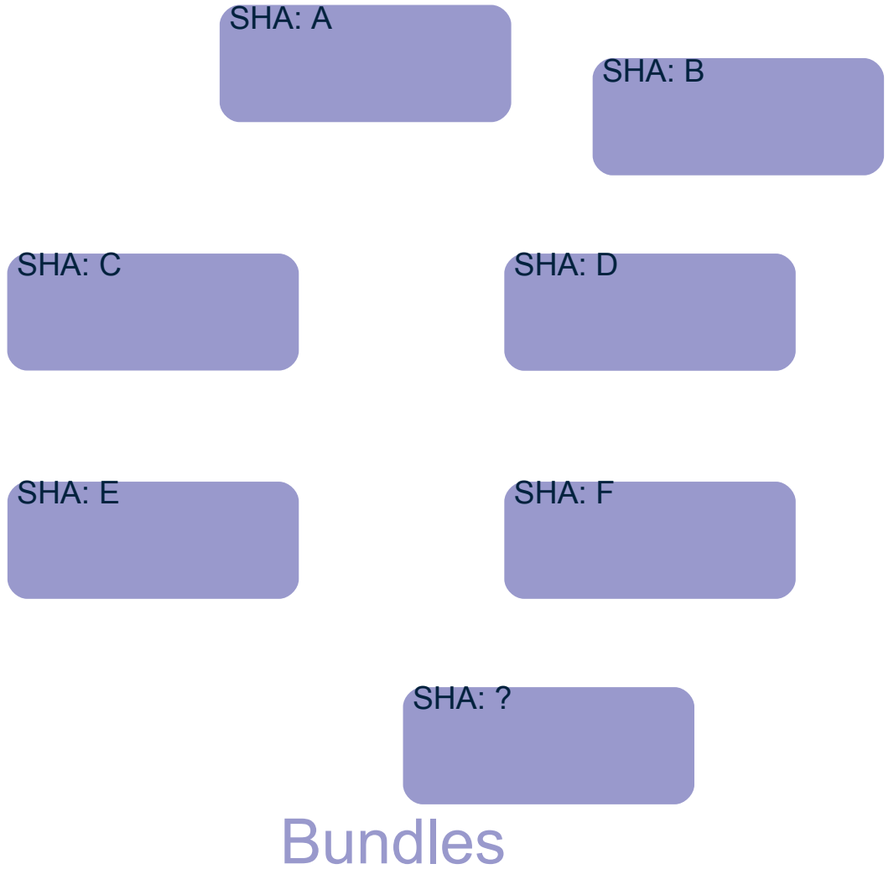
OSGi Loaders – Unique Wirings

- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA



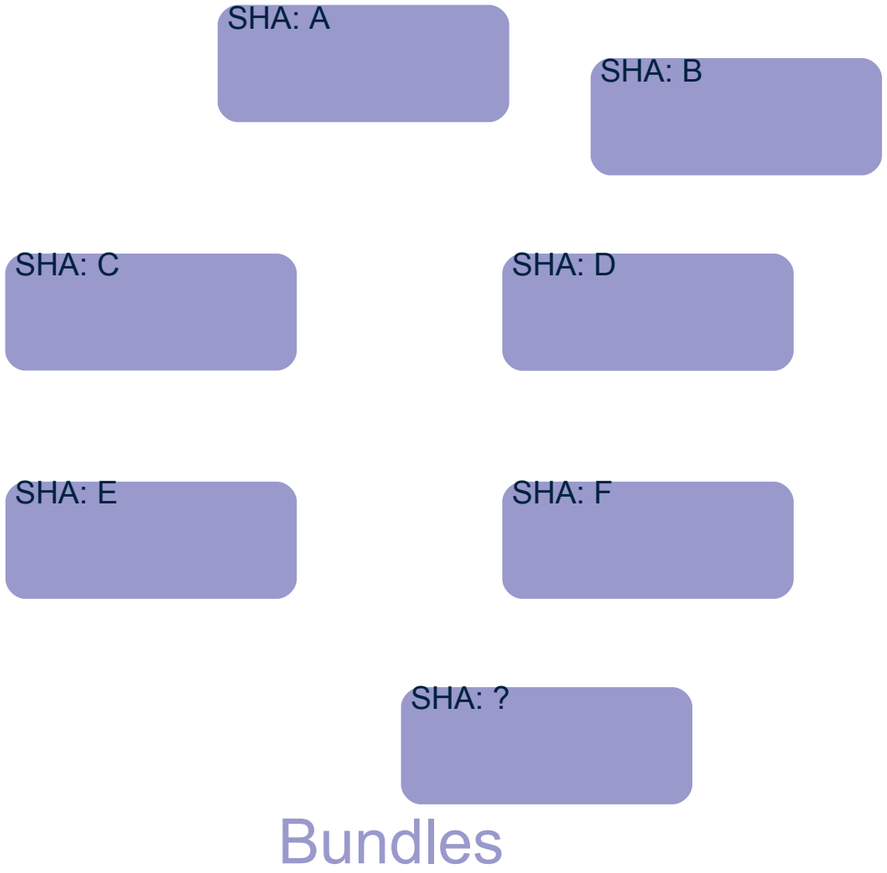
OSGi Loaders – Unique Wirings

- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA



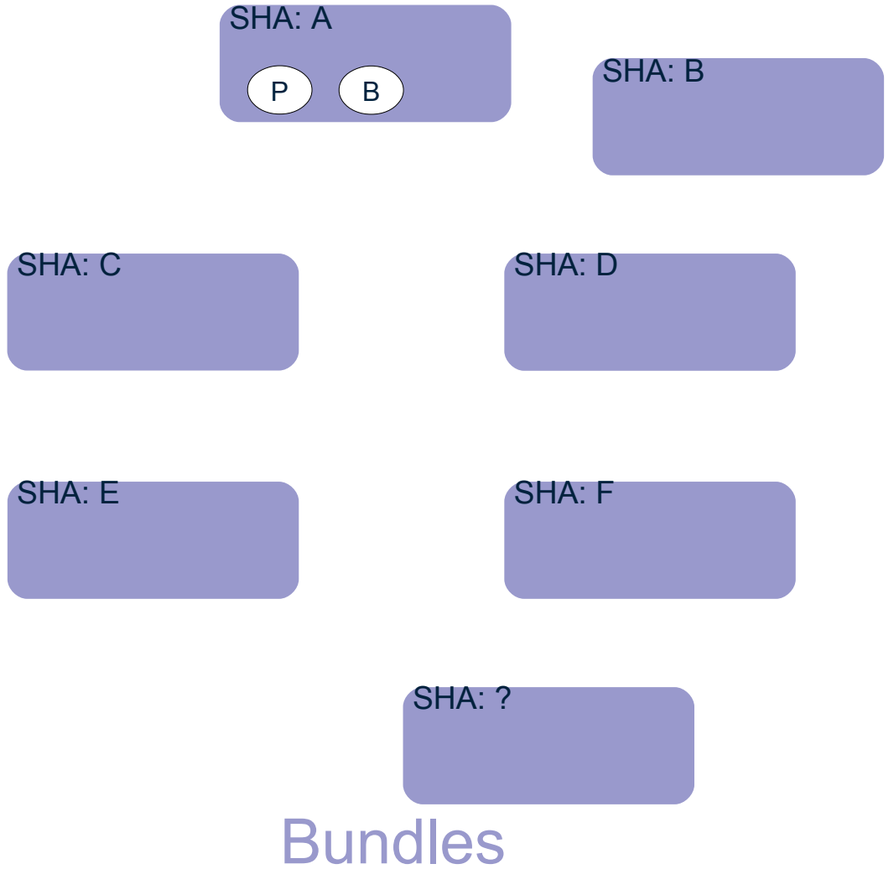
OSGi Loaders – Unique Wirings

- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values



OSGi Loaders – Unique Wirings

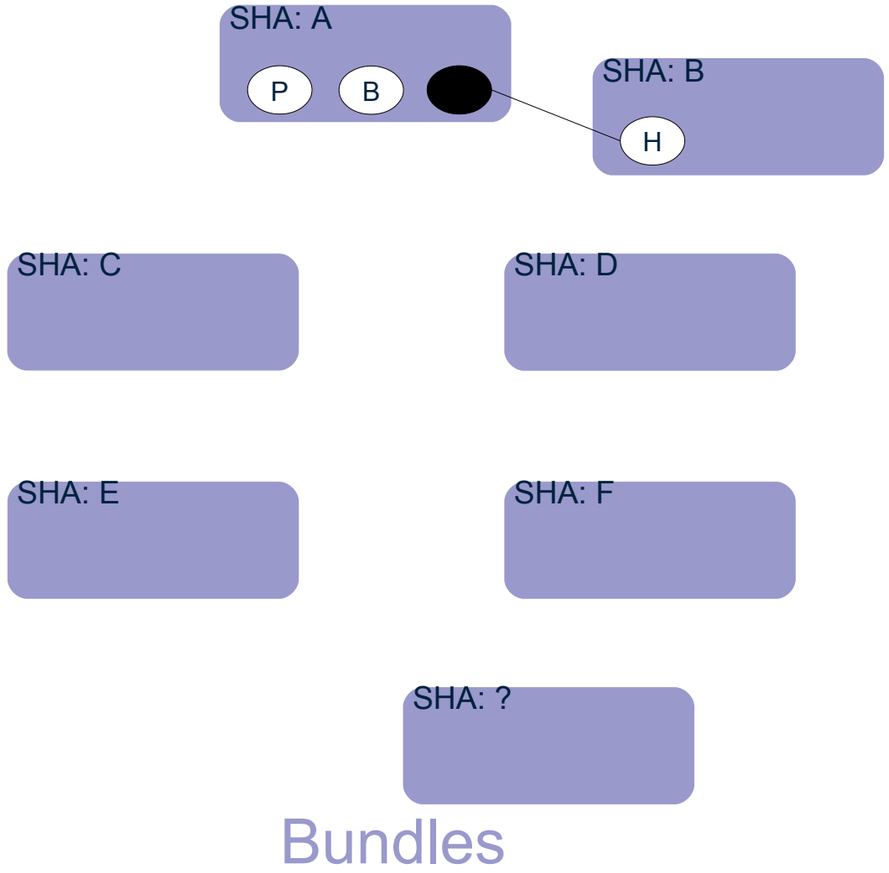
- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values
  - Seed with the SHA of the wiring's resource



A

OSGi Loaders – Unique Wirings

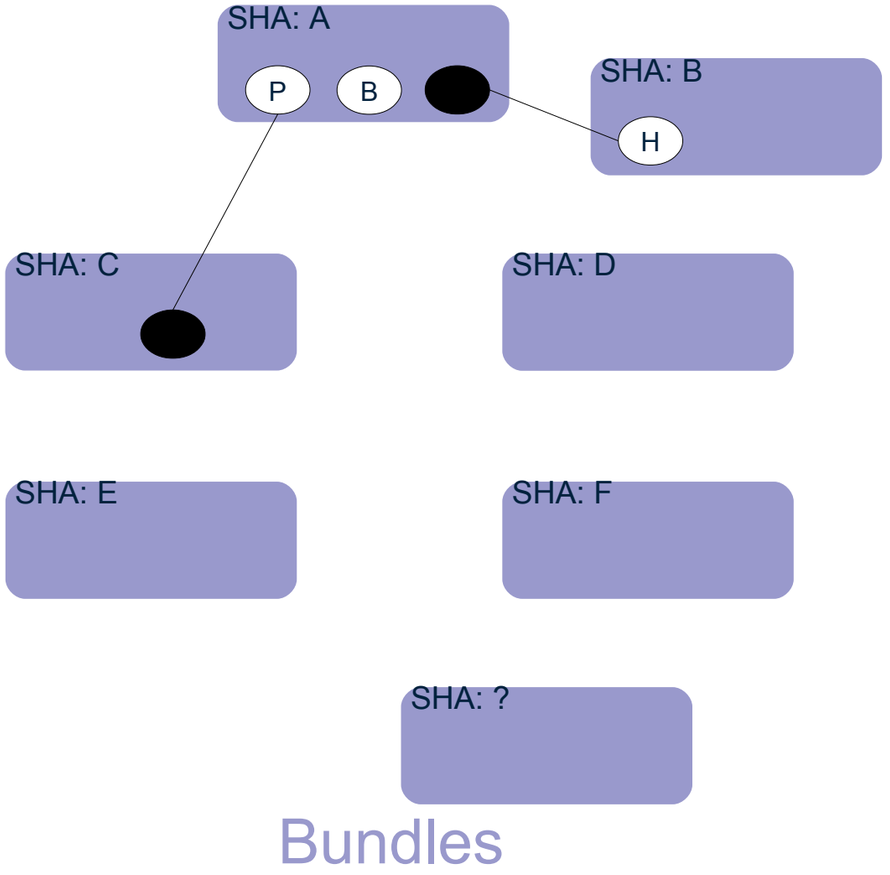
- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values
  - Seed with the SHA of the wiring's resource
  - Merge in SHA of attached fragments



A + B

OSGi Loaders – Unique Wirings

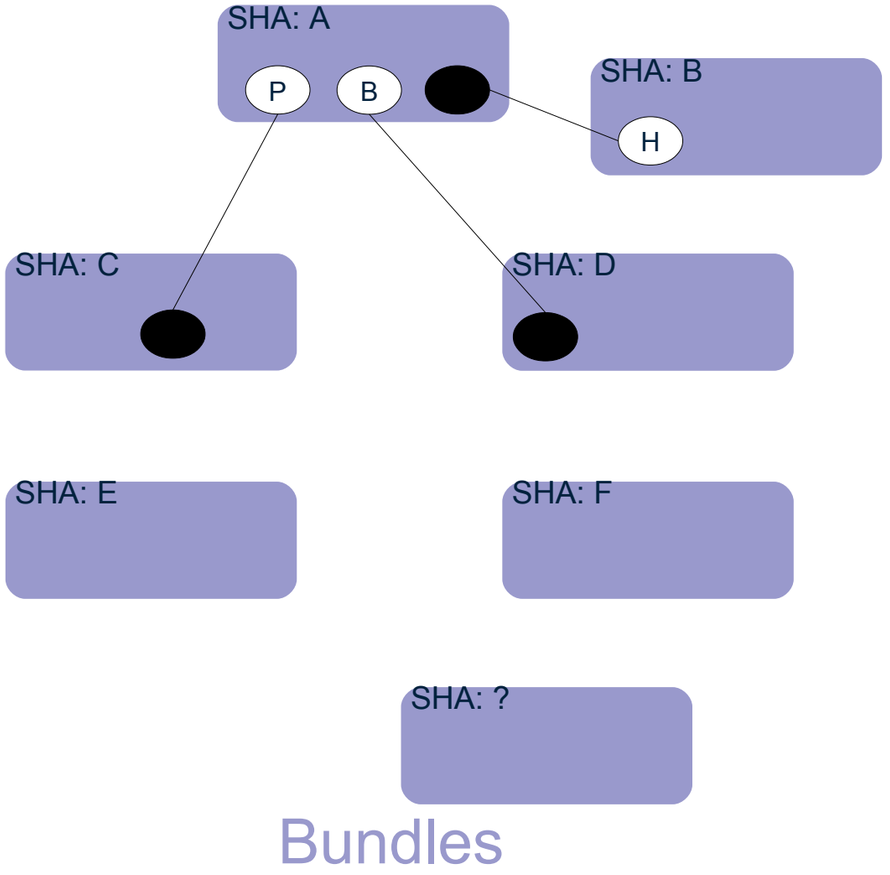
- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values
  - Seed with the SHA of the wiring's resource
  - Merge in SHA of attached fragments
  - Merge in SHA of resources providing package and bundle wires



A + B + C

OSGi Loaders – Unique Wirings

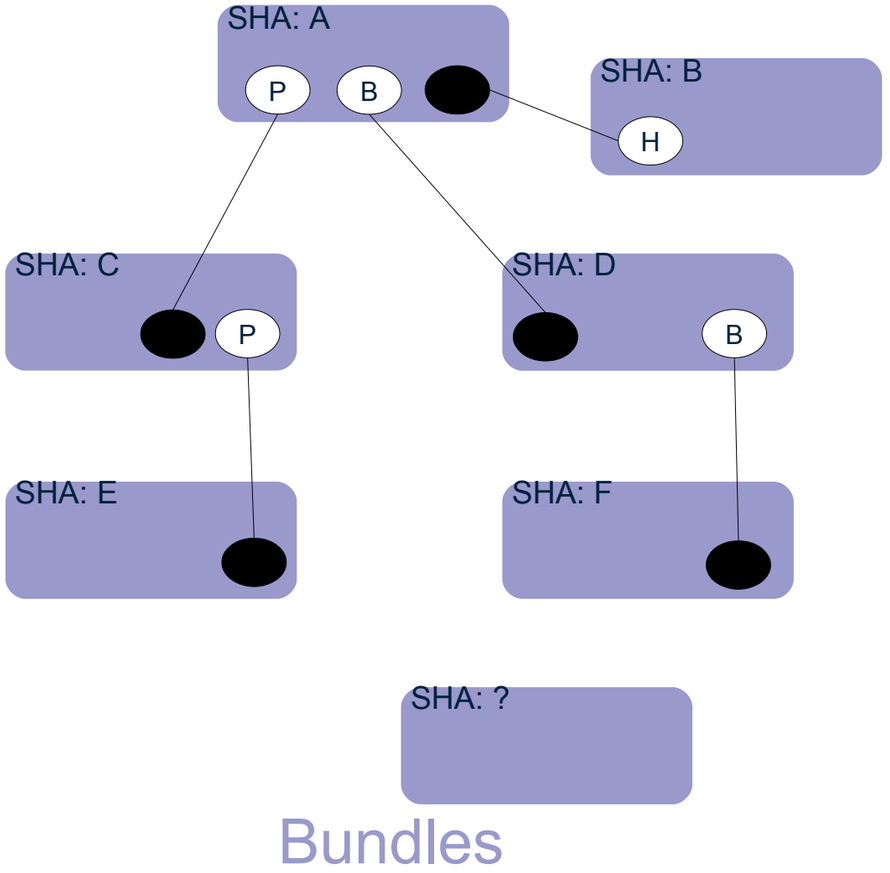
- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values
  - Seed with the SHA of the wiring's resource
  - Merge in SHA of attached fragments
  - Merge in SHA of resources providing package and bundle wires



A + B + C + D

OSGi Loaders – Unique Wirings

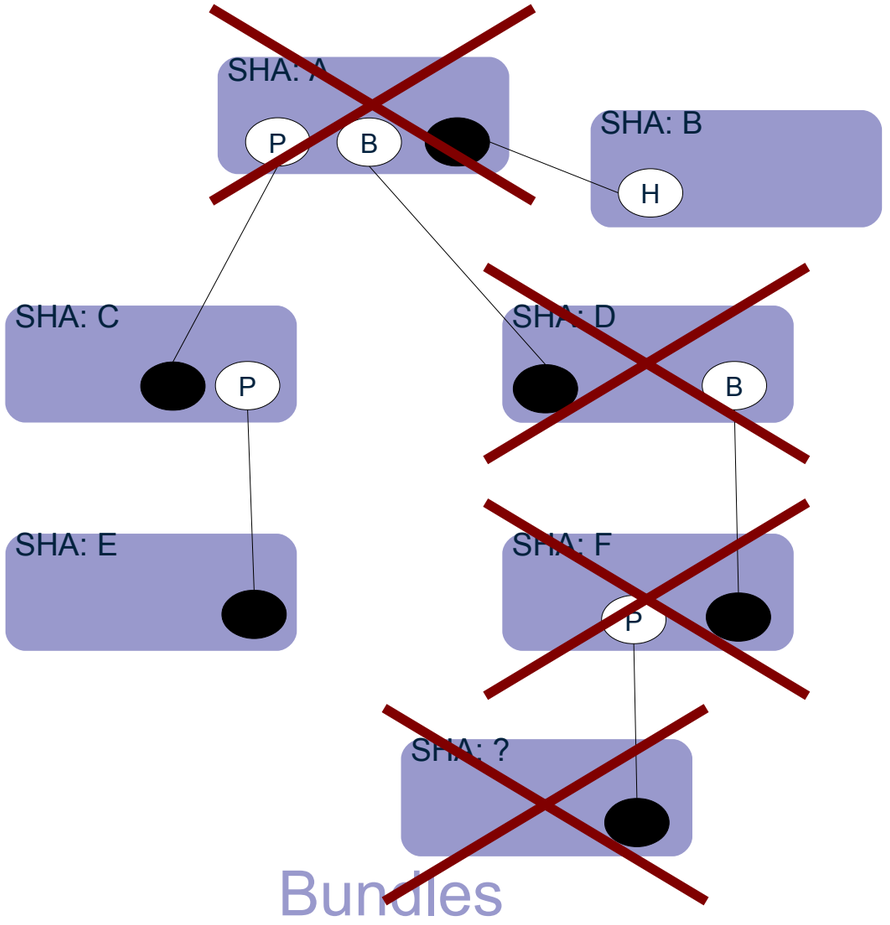
- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values
  - Seed with the SHA of the wiring's resource
  - Merge in SHA of attached fragments
  - Merge in SHA of resources providing package and bundle wires
  - Perform transitive closure for package and bundle requirements



A + B + C + D + E + F = Unique CL ID

OSGi Loaders – Unique Wirings

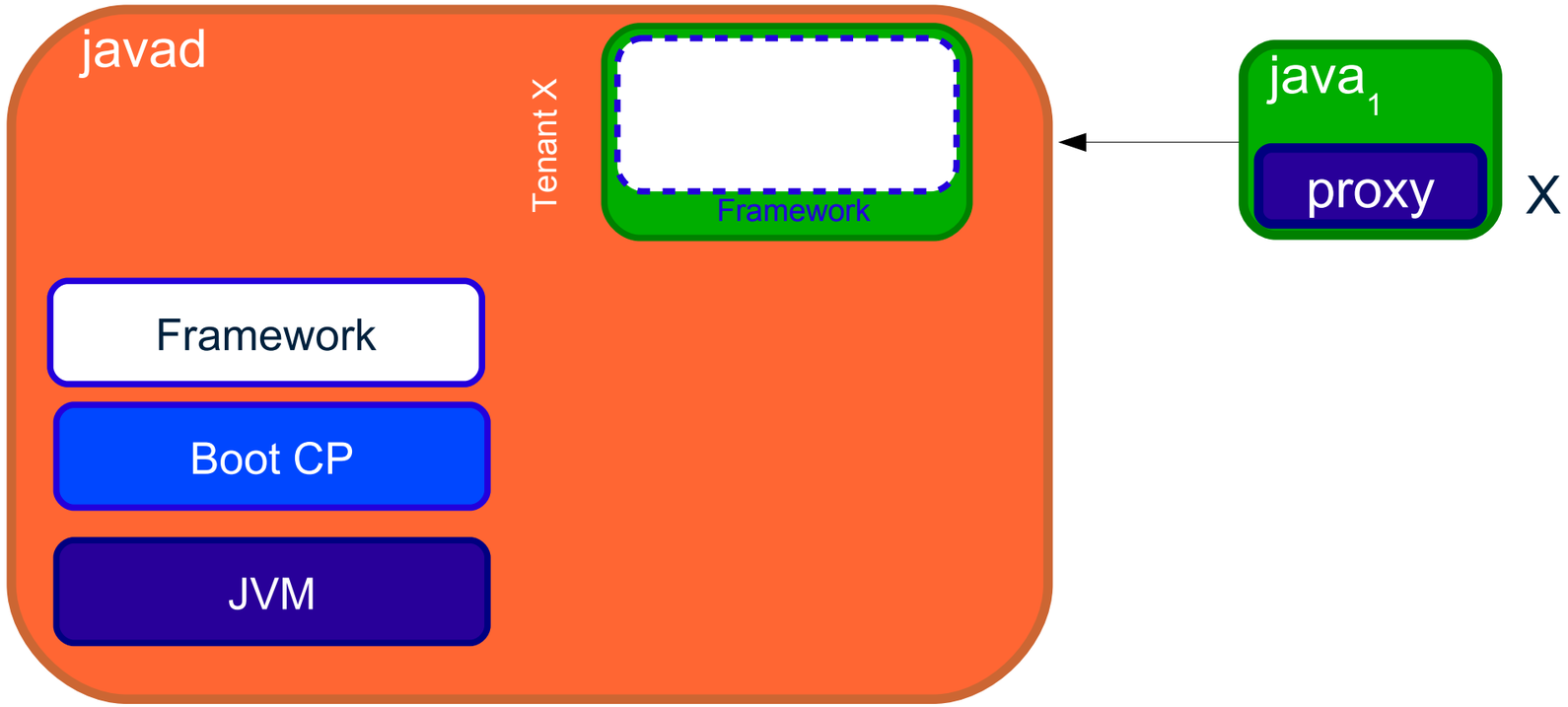
- Identify bundle resources
  - Calculate the SHA of the content
  - Only support jar'ed because it is easy to compute the SHA
  - Dynamic imports cause unknown SHA
- Identify bundle wiring
  - Build a SHA based on bundle content SHA values
  - Seed with the SHA of the wiring's resource
  - Merge in SHA of attached fragments
  - Merge in SHA of resources providing package and bundle wires
  - Perform transitive closure for package and bundle requirements
  - If an unknown content SHA is encountered then the wiring SHA is unknown (not shared)



A + B + C + D + E + F + ? = ?

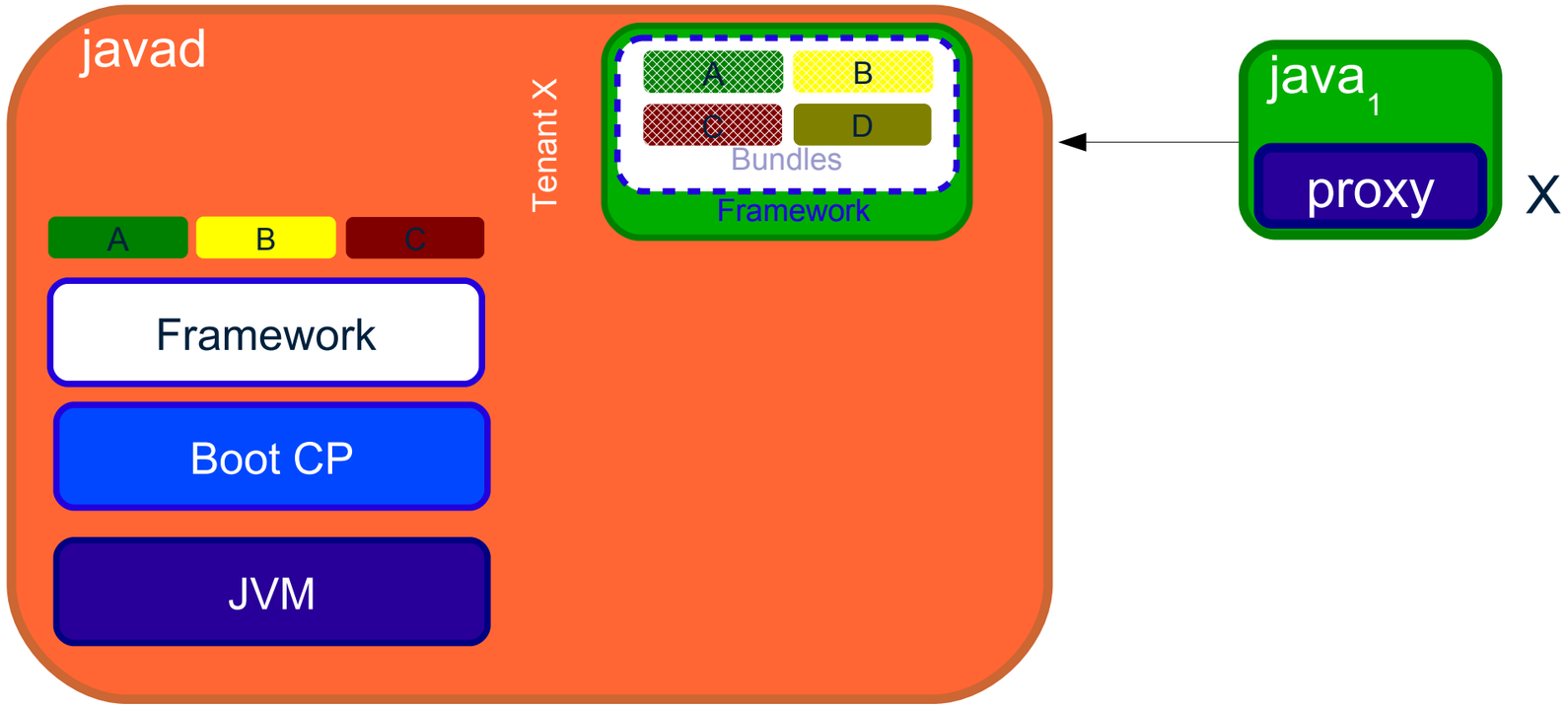
Discovering Shared OSGi Loaders

New `java` process X is started and a Framework **instance** is created in for tenant X in `javad`. Framework classes are **shared**.



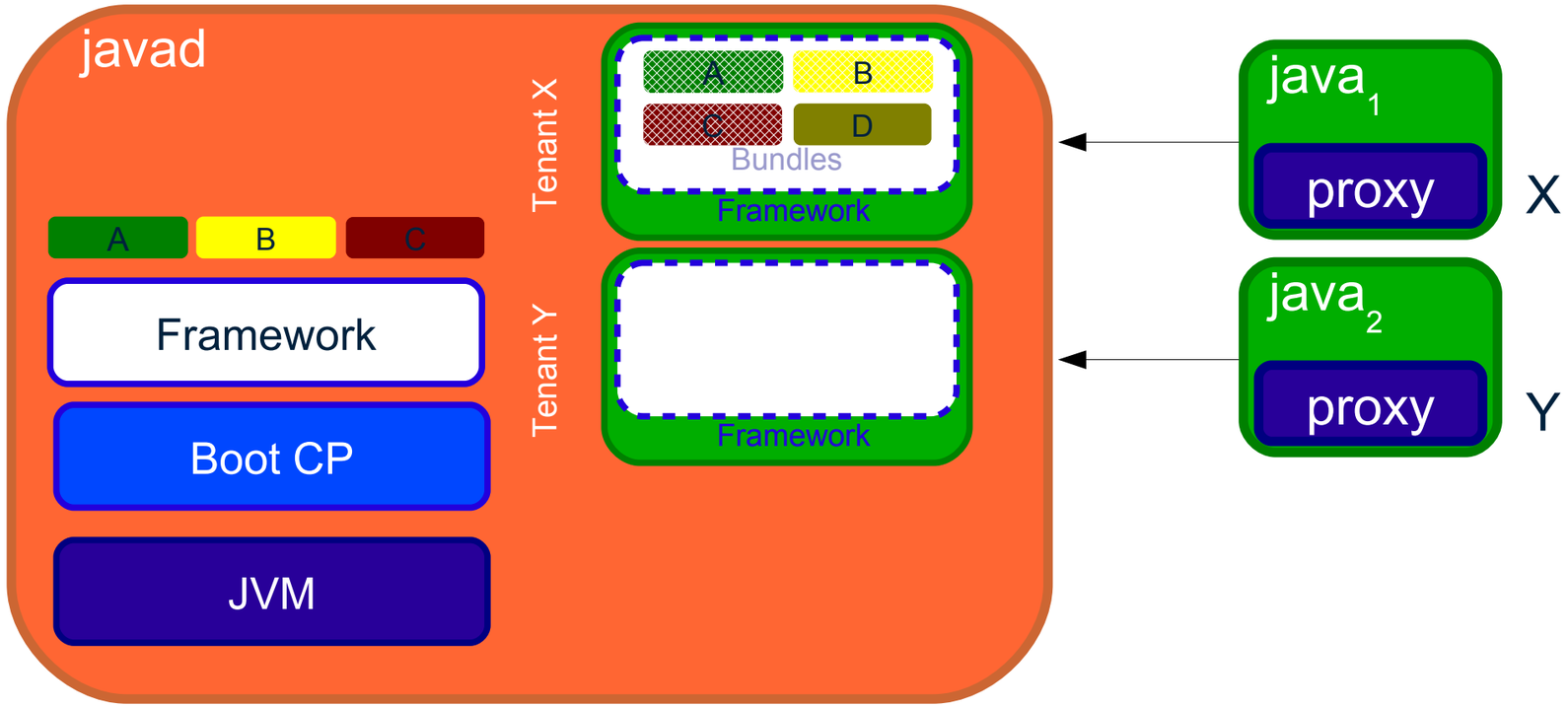
Discovering Shared OSGi Loaders

Bundles A, B, C & D are installed. Wiring unique ID (SHA) is **successfully** computed for A, B and C. **Shared** loaders **A**, **B** & **C** are **created**. A **tenant specific** loader **D** is **created**.



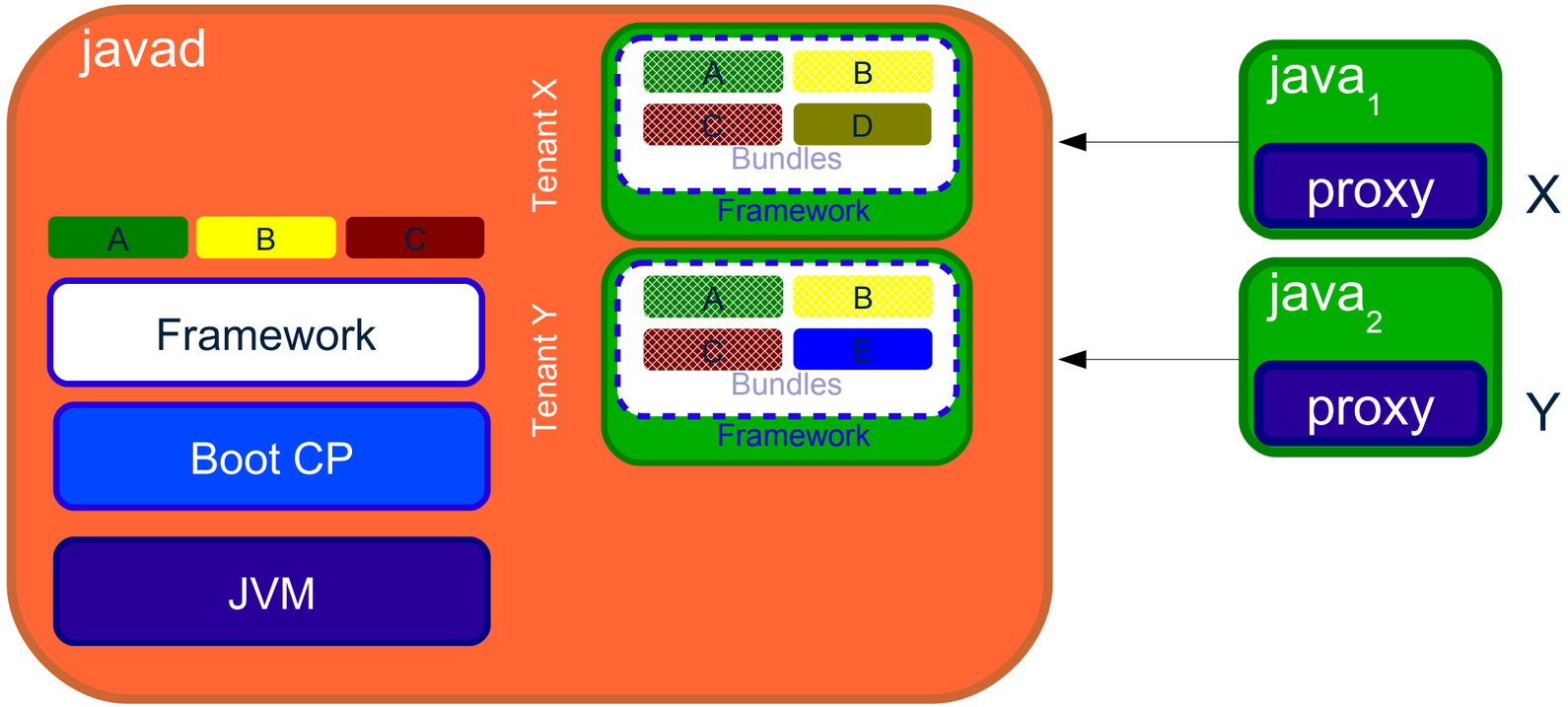
Discovering Shared OSGi Loaders

New `java` process Y is started and a Framework **instance** is created in for tenant Y in `javad`. Framework classes are **shared**.

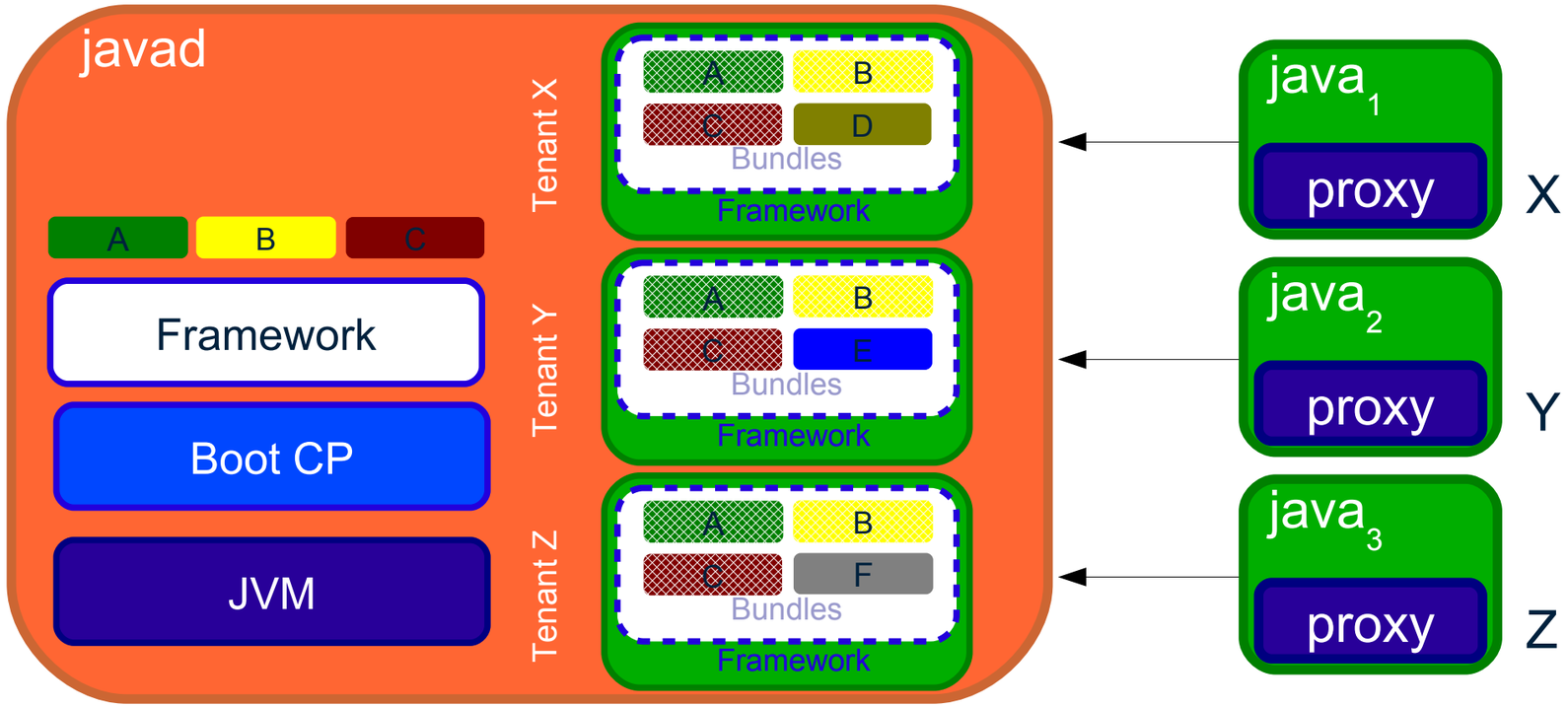


Discovering Shared OSGi Loaders

Bundles A, B, C & E are installed. Wiring unique ID (SHA) is **successfully** computed for A, B and C. **Shared** loaders **A**, **B** & **C** are **found**. A **tenant specific** loader **E** is **created**.



Discovering Shared OSGi Loaders

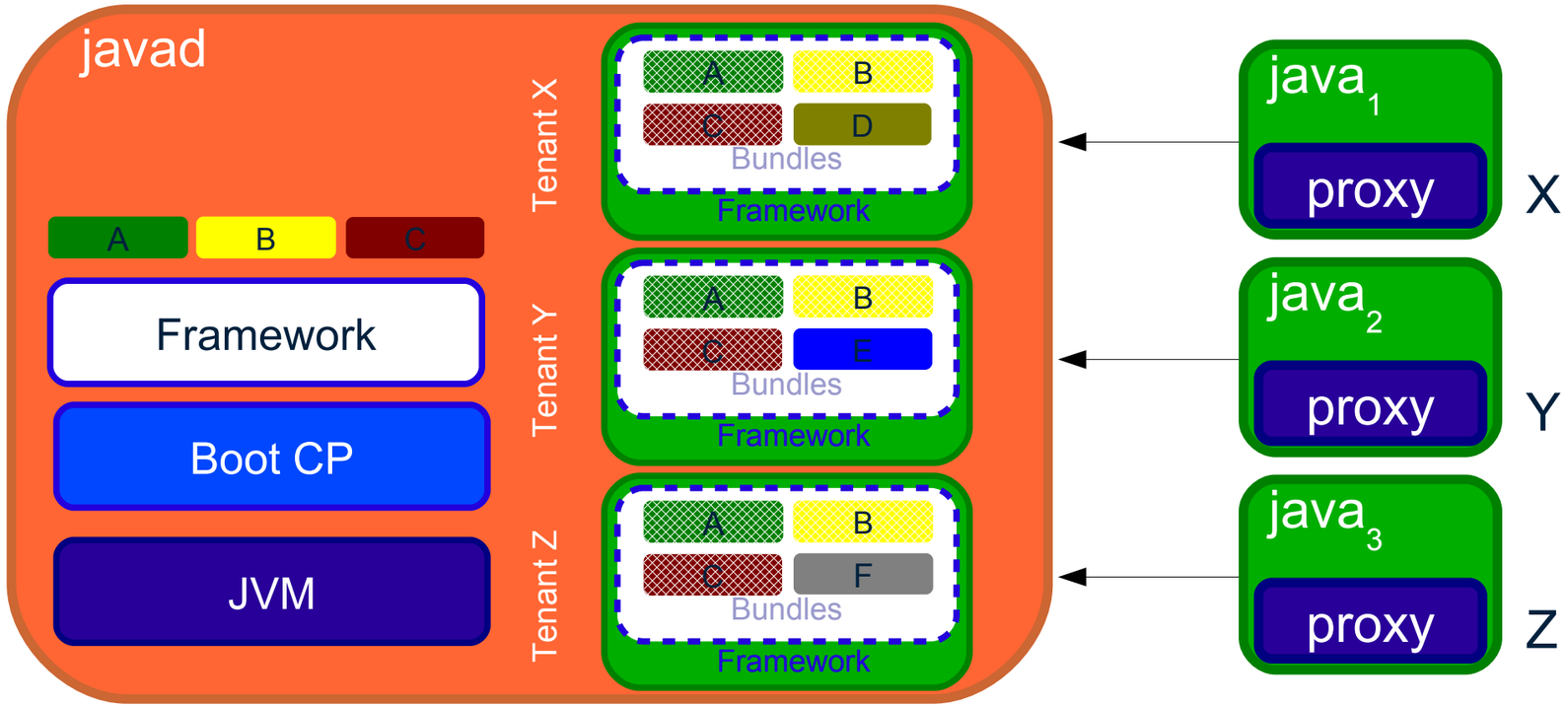


Even though each tenant does not have identical bundles installed they are able to share classes that are common

Discovering Shared OSGi Loaders

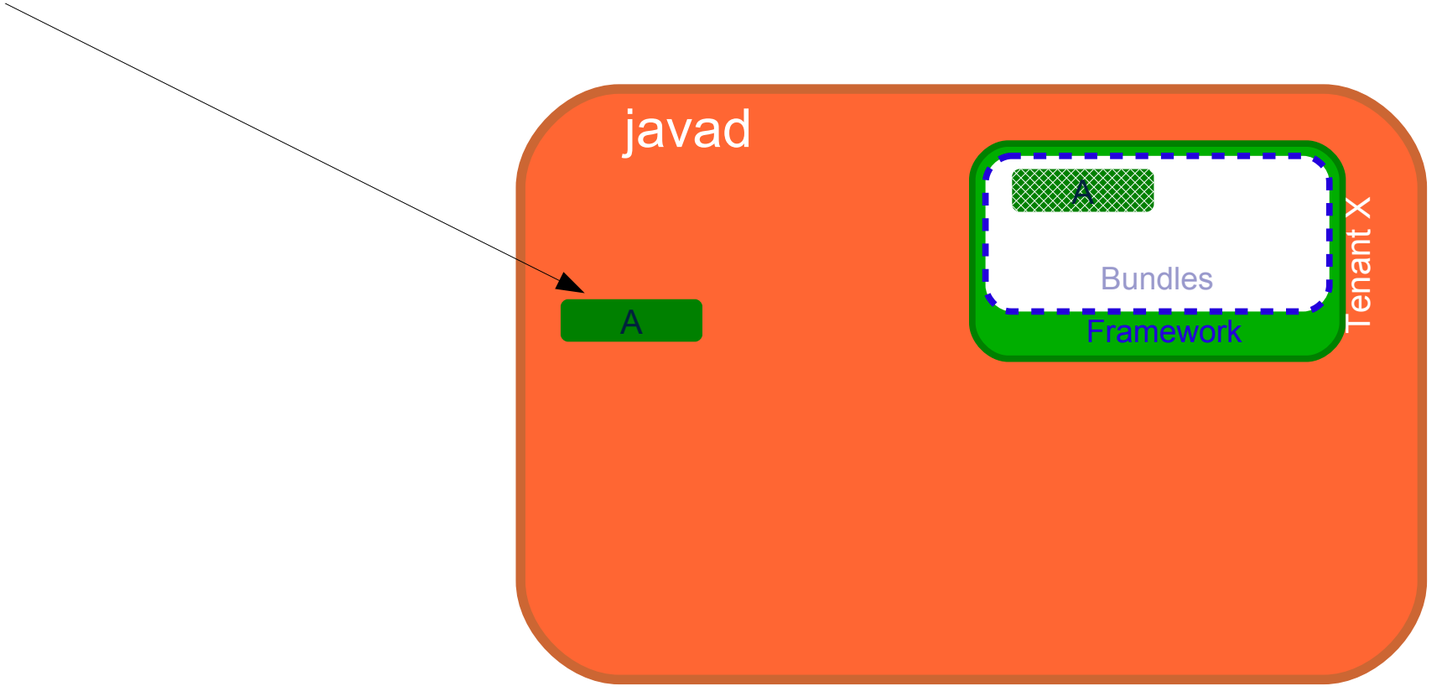
In order to provide proper isolation objects are not shared, only classes and loaders

This is not as dense as using subsystems, but it provides much better isolation



OSGi Loader - Multiplexing

- OSGi loaders must implement the BundleReference interface
- ProtectionDomain must provide the correct answers for each tenant
- Resource URLs must be tenant specific

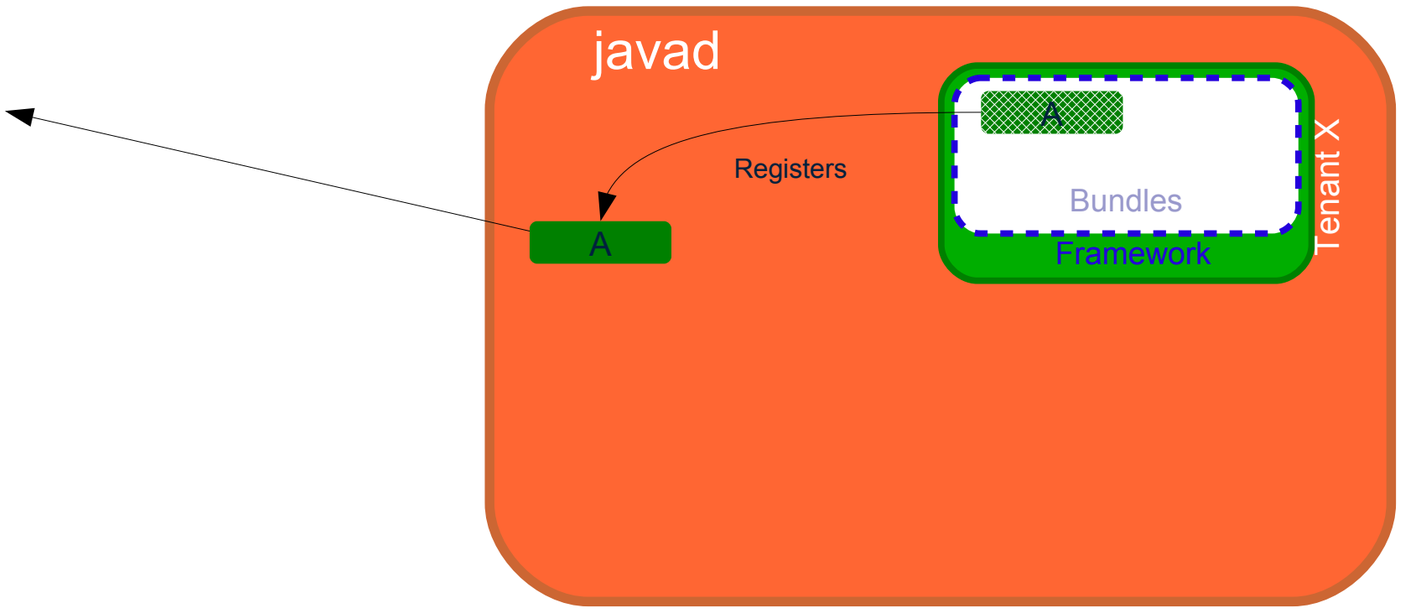


OSGi Loader - Multiplexing

- OSGi loaders must implement the BundleReference interface
- ProtectionDomain must provide the correct answers for each tenant
- Resource URLs must be tenant specific

```
tenantMap

{ tenantID_X → clTenant_X }
```

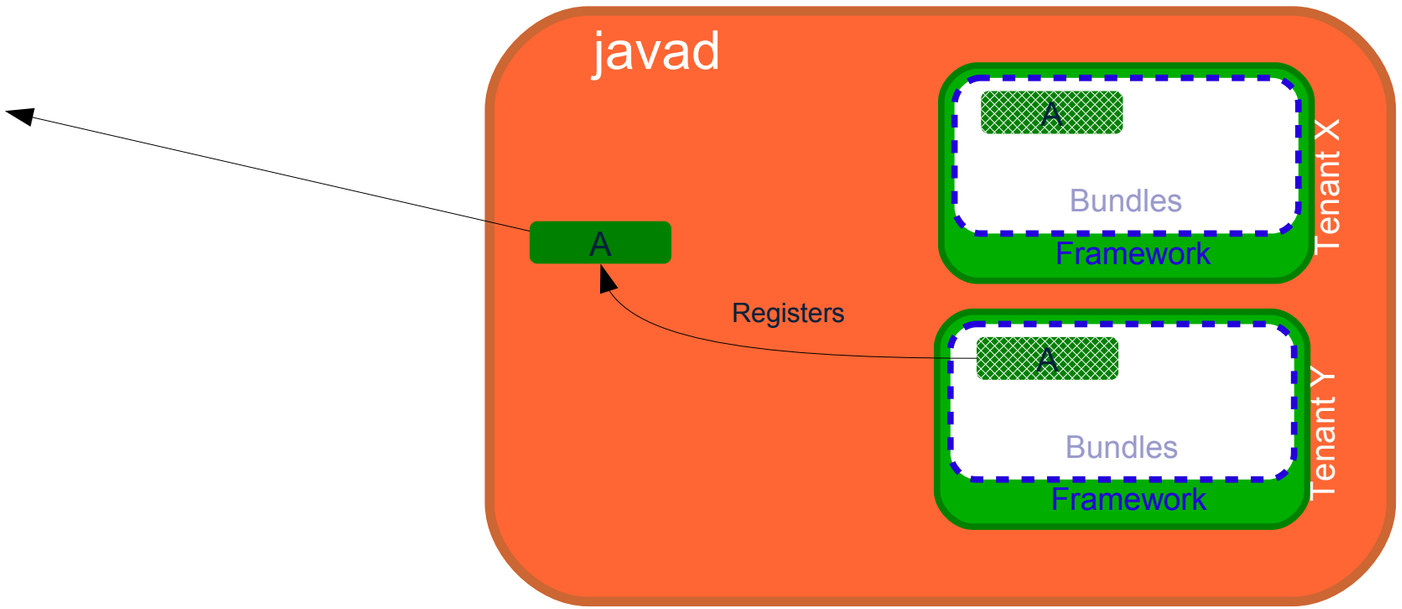


OSGi Loader - Multiplexing

- OSGi loaders must implement the BundleReference interface
- ProtectionDomain must provide the correct answers for each tenant
- Resource URLs must be tenant specific

```
tenantMap

tenantID_X → clTenant_X
tenantID_Y → clTenant_Y
```



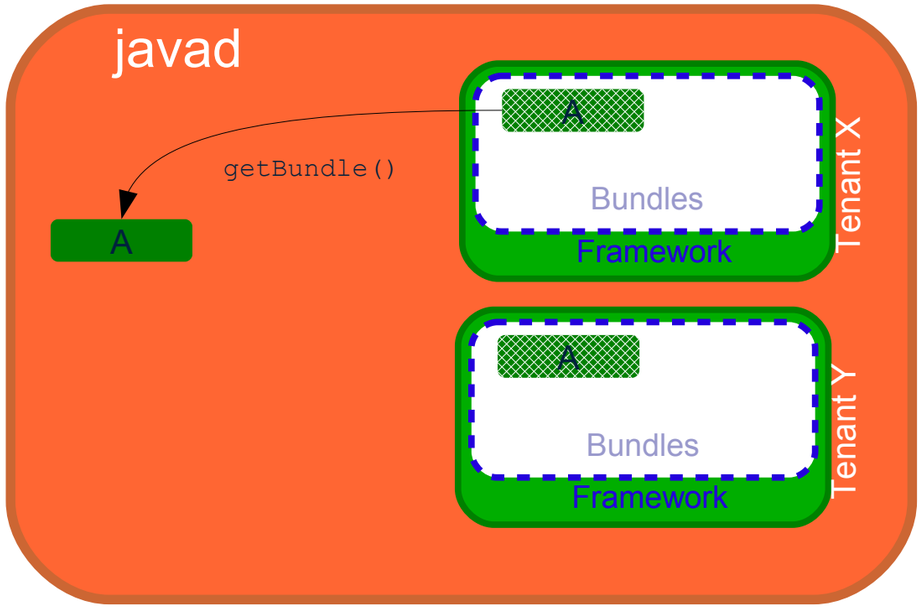
OSGi Loader - Multiplexing

- OSGi loaders must implement the BundleReference interface
- ProtectionDomain must provide the correct answers for each tenant
- Resource URLs must be tenant specific

```
tenantMap

tenantID_X → clTenant_X
tenantID_Y → clTenant_Y
```

```
((BundleReference) loaderA).getBundle();
```



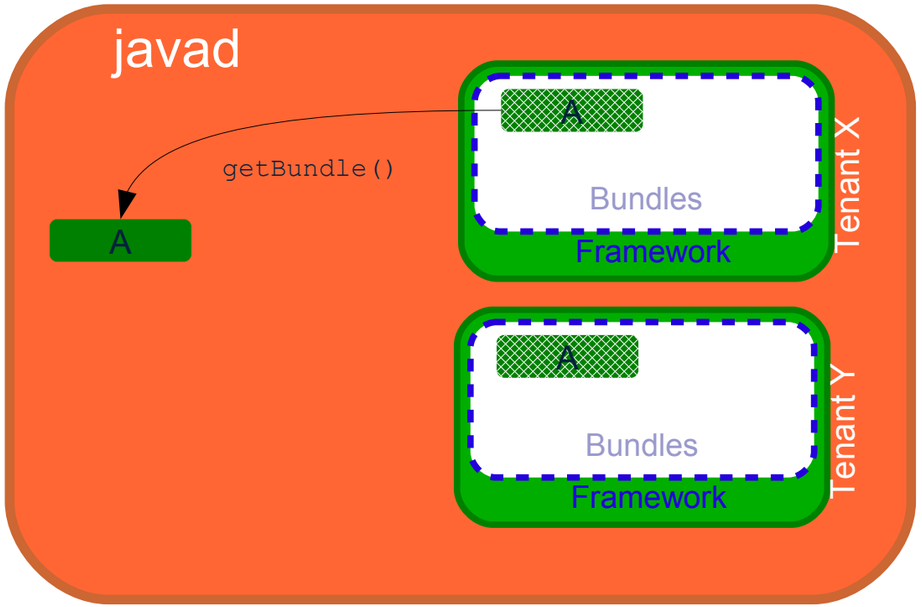
OSGi Loader - Multiplexing

- OSGi loaders must implement the BundleReference interface
- ProtectionDomain must provide the correct answers for each tenant
- Resource URLs must be tenant specific

```
{          tenantMap          }
  tenantID_X → clTenant_X
  tenantID_Y → clTenant_Y
```

```
((BundleReference) loaderA).getBundle();
```

```
tenantMap.get(tenantID_X).getBundle();
```

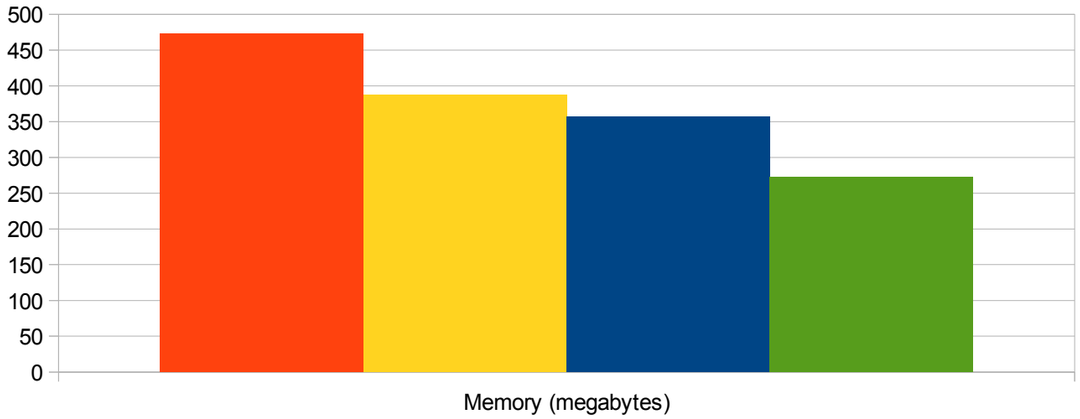


IBM JVM Multitenant - Notes

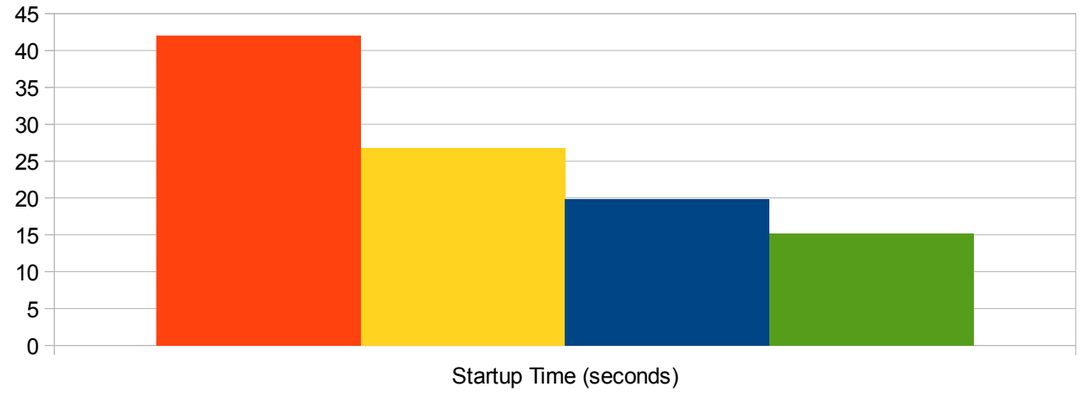
- Available in Java 7 (64-bit)
- Multitenant support is available as a tech preview

  https://www.ibm.com/developerworks/java/jdk/linux/download.html
- Additional patches to the VM are necessary
  - Create and discovery of shared class loaders with unique IDs
- Equinox enhancements are implemented as an equinox extension on top of Luna.

Results?



- Starting 10 Liberty Servers
  - Minimal server configuration
  - Servlet support only + OSGi console
  - About 75 bundles, most all shareable
- MT `javad` fully initialized
  - Shared class loaders already created
  - Classes loaded for startup
  - JIT cache populated



OSGi Loader - Limitations

- Dynamic imports kill ability to share
- Must not share class loaders for bundles that need weaving
- Native code may be able to be shared if no global state is stored in native code
- Lazy activated bundles are not supported

New OSGi Specifications

- Last week (week of June 2nd) the **OSGi Core Release 6 Specification** received its final approval and will be available this week to the public for downloading.
  - http://www.osgi.org/Specifications/HomePage
- Also happening last week, the OSGi Board of Directors approved the publication of an **Early Draft Specification of OSGi Enterprise Release 6** for downloading.
  - http://www.osgi.org/Specifications/Drafts

Trademarks

IBM and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.