JUDCon

JBoss Users & Developers Conference

Boston:2010

Zen of Class Loading

Jason T. Greene
EAP Architect, Red Hat
June 2010

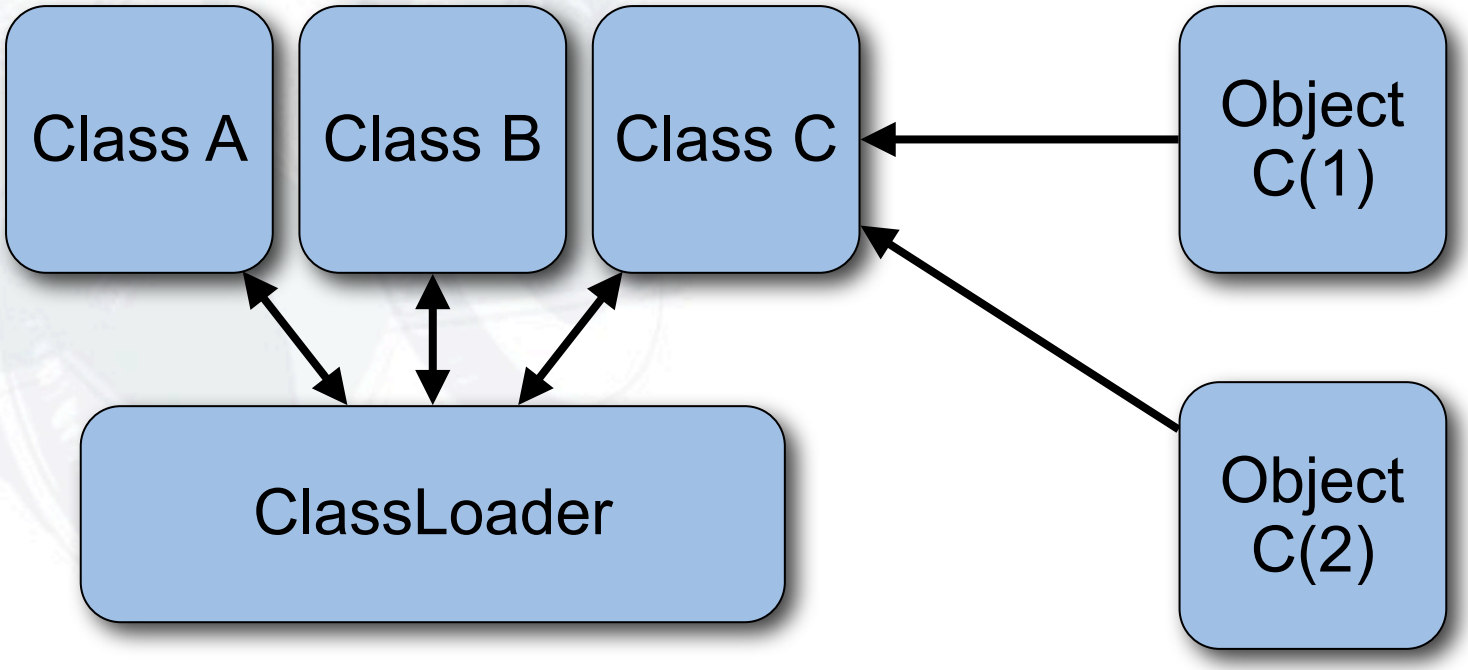
What is the Class class?

- Represents a class, enum, interface, annotation, or primitive marker in the JVM
- Allocates memory in both heap and permanent generation
- Is always referenced by every object instance of the Class
- *Unique* (by name) only to a ClassLoader
- Holds a reference to its ClassLoader

What is a ClassLoader?

- Responsible for loading classes
- Holds a strong reference to all classes it loads
- Has an optional parent for delegating
- Serves as an SPI for custom loaders
- Provides a byte buffer of class bytecode to the JVM

Class, Object, and ClassLoader reference pattern



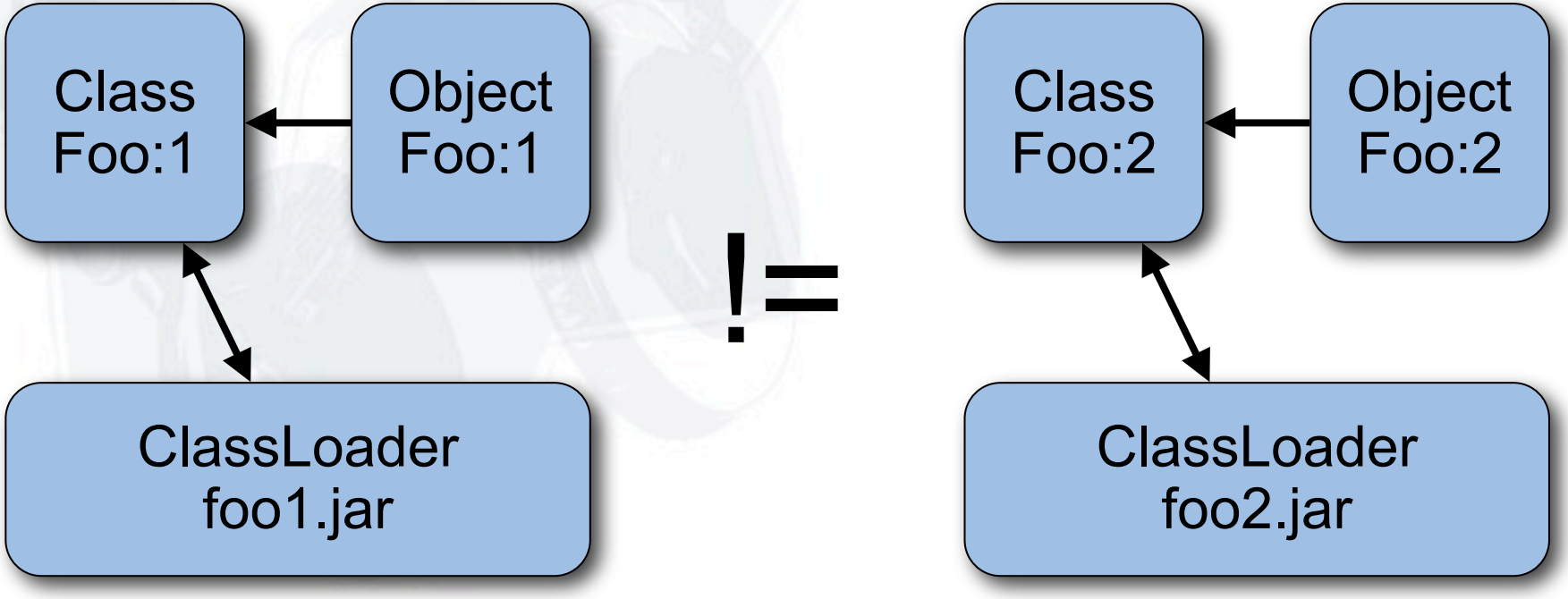
Understanding Uniqueness

```
URLClassLoader cl1 =
    new URLClassLoader(foo1Jar, null);
URLClassLoader cl2 =
    new URLClassLoader(foo2Jar, null);

Class fooClass1 = cl1.loadClass("Foo");
Class fooClass2 = cl2.loadClass("Foo");

fooClass1.equals(fooClass2) // FALSE!
```

Classloader Uniqueness (Isolation)



Loading by Reference vs ClassLoader

- Normal direct references by a class to another class ***reuses*** the referencing class' ClassLoader.

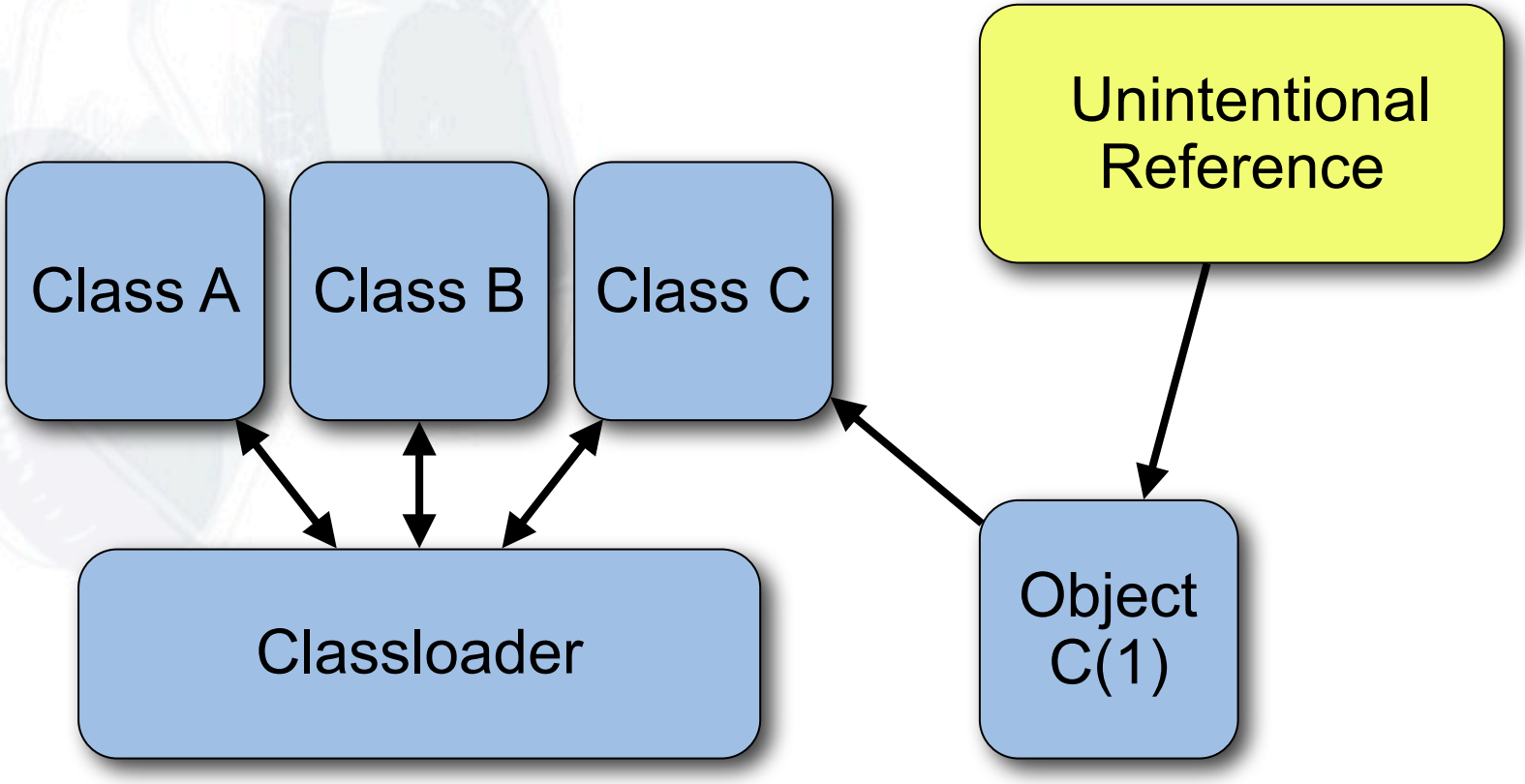
```
// Load using the cl1 loader
Class fooClass1 = cl1.loadClass("Bob");

// Reuses the loader for this class.Equiv to
// getClass().getClassLoader().loadClass()
Bob bob = new Bob();
Class bar = Bar.class;
```

Problem #1 - OutOfMemory:PermGen

- Too many Classes were loaded!
- Sizing could be wrong
  - Big application with many classes
- Possible Classloader leak
  - Hot Deployment creates new Classloaders on redeploy
  - Old loaders are intended to be GC'd
  - Anything with a direct or indirect reference to a loader will cause it to leak

ClassLoader Leak

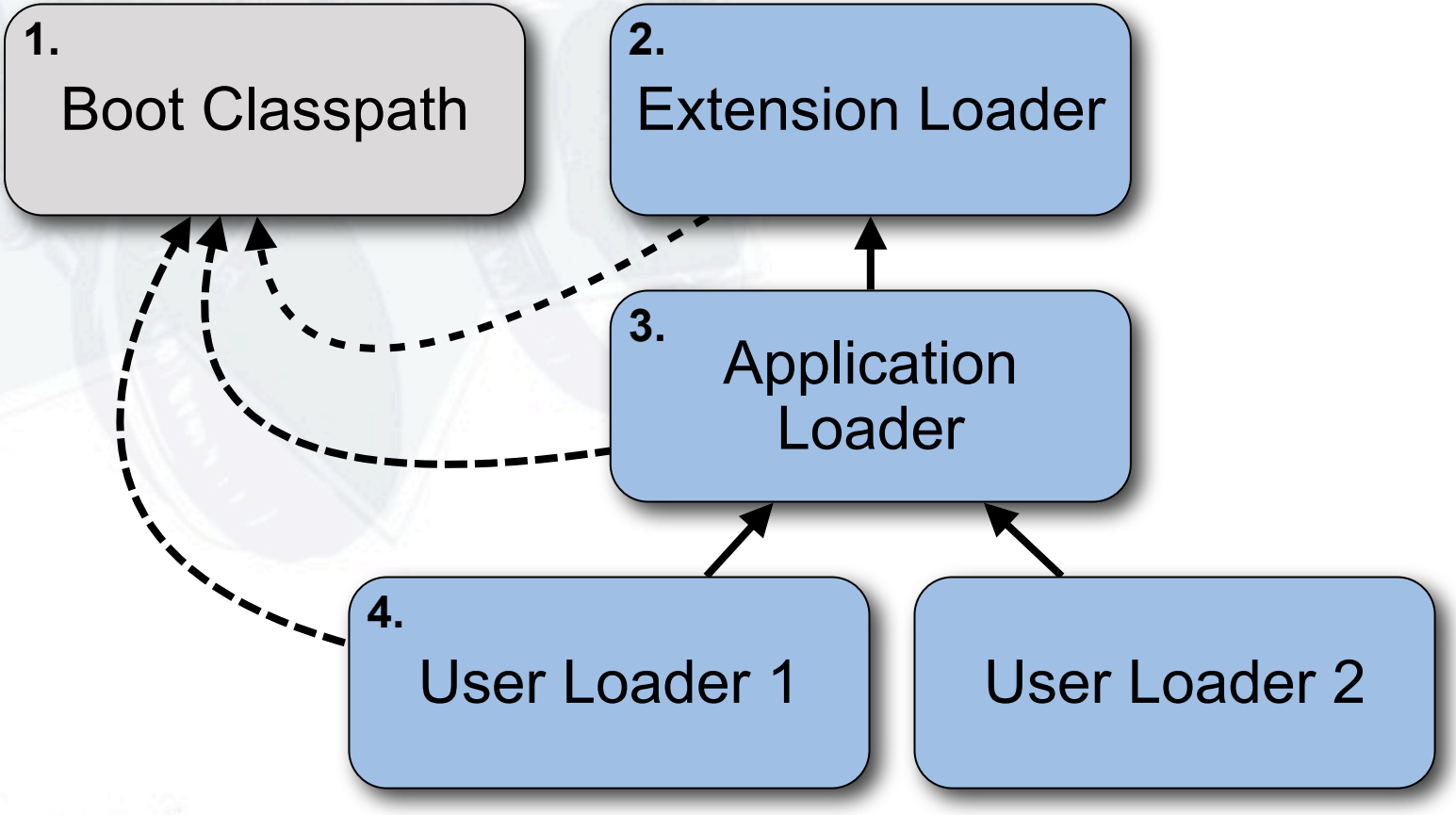


Common Leak Sources

- Static field on a class with a long lifecycle
- Caching frameworks
- Persistent Thread-locals
  - Automatic cleaning is ***extremely*** infrequent
- Framework bugs
  - Logging frameworks, third-party frameworks etc

JDK Class Loading Delegation (Parent First)

1-4 = User CL Search Order



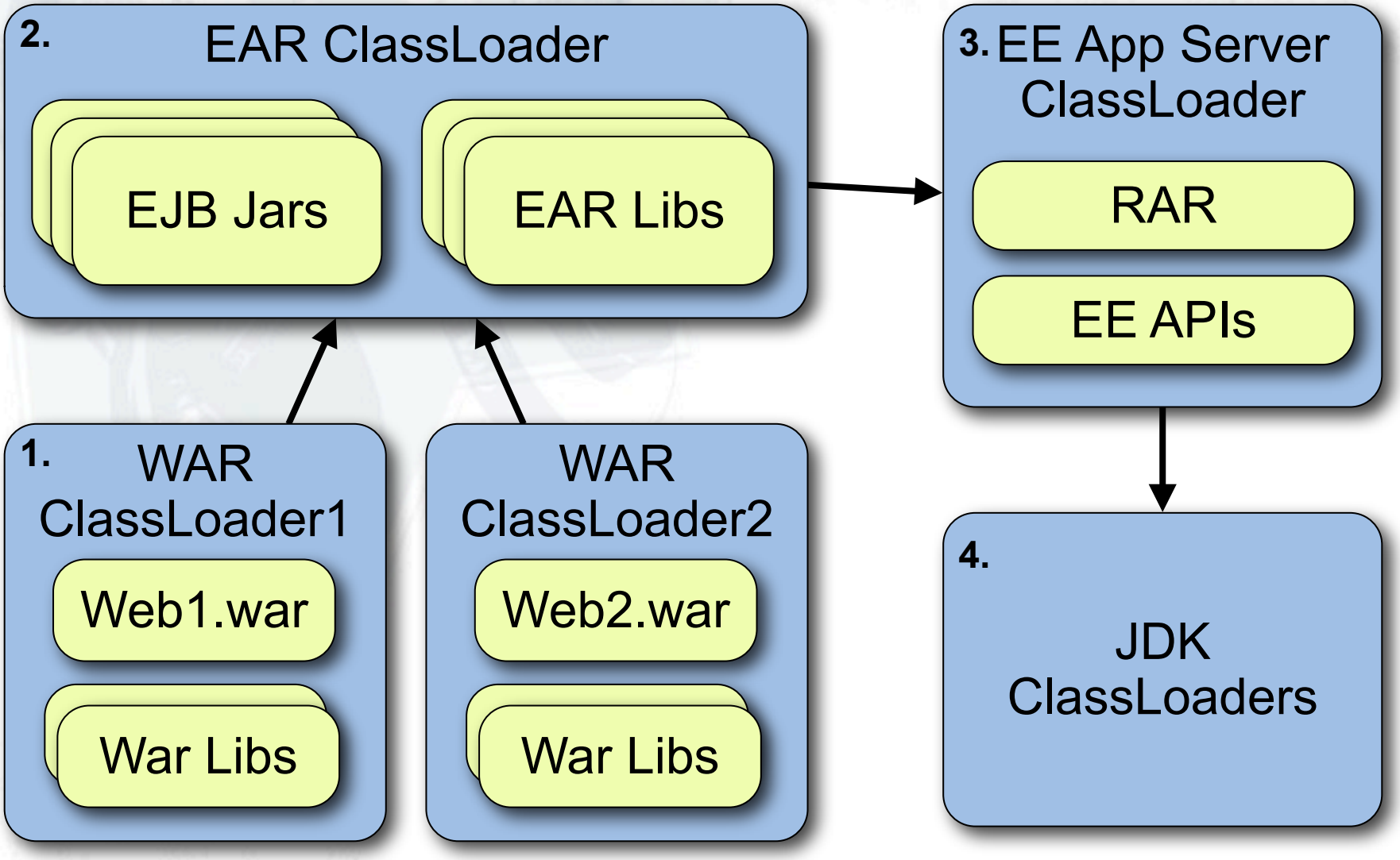
JDK Class Loading Delegation

- Parent-first prevents overriding a parent
- Custom classloaders can change this
  - and they do!
- Bootstrap is searched even if parent == null
- Bootstrap classes often return null for getClassLoader()

Problem #2 - ClassNotFoundException

- Classes are not visible to the loader
- Variety of causes
  - Unintentional reference/dependency
  - Reference from a parent to a child or a sibling to another sibling (only children can see parent's classes)
    - Ex. A jar in ear/lib accesses a class in a WAR
- Solutions
  - Audit cross jar references (e.g Tattletale)
  - Draw out how they should fit in the CL tree

EE Class Loading Model (Child First)



EE Class Loading Model

- Child-first allows deployments to override parent classes and jars
  - Common use-case is supporting bundling a war that needs a different version of a framework than the EAR
- EJB jars share classes within the EAR
- WARs do not share classes, but do share classes in the EAR
- RARs and Global EJB-JARs visible to everyone

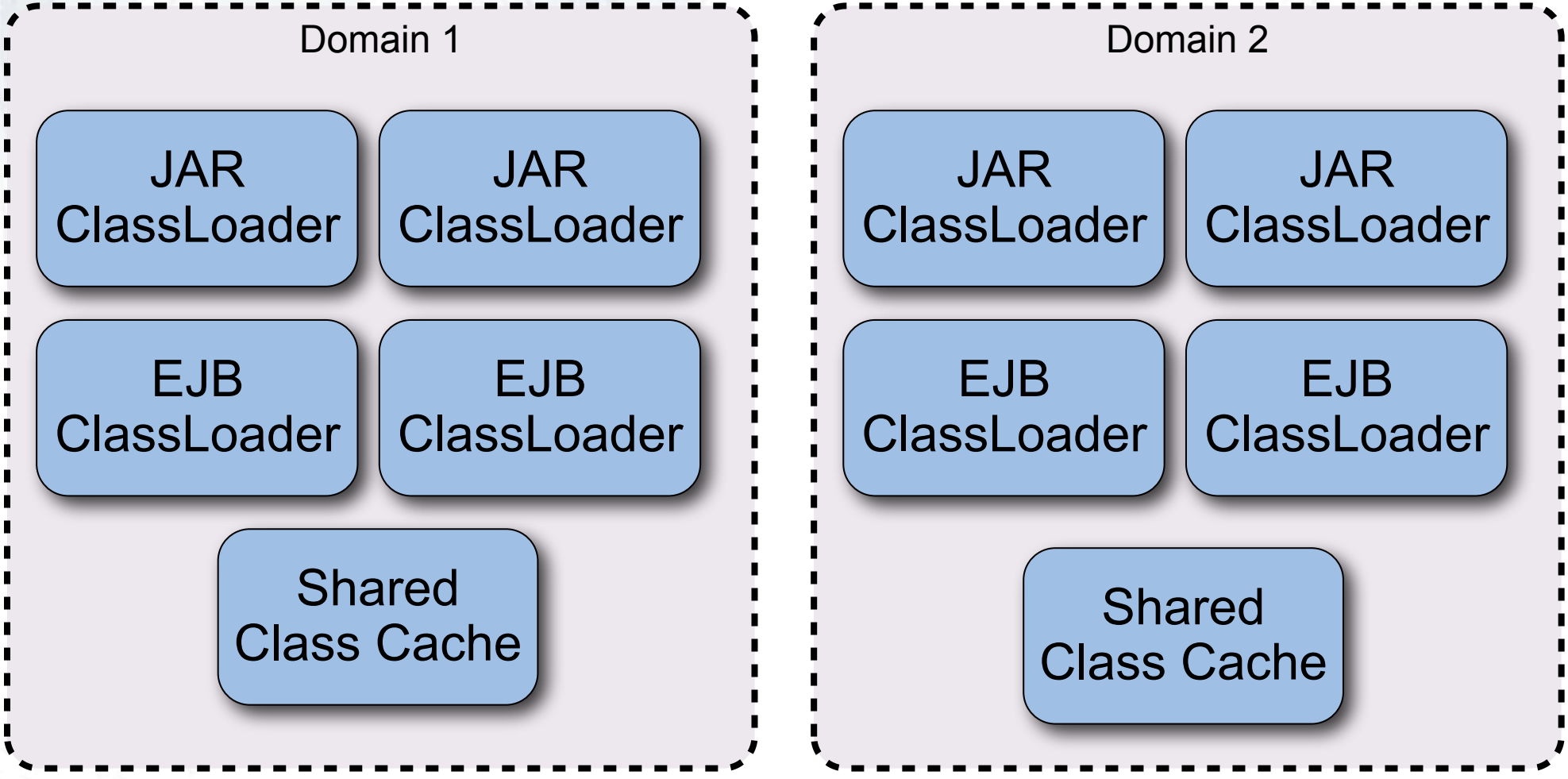
Problem #3 - ClassCastException on same name

- Duplicate classes in isolated loaders!
- Usually a packaging problem
- Common scenario - bundling ejb local interfaces in a WAR
  - The WAR ClassLoader's version is passed to the EJB, which has the EAR ClassLoader's version
  - The classes are not equal, so CCE
  - Also happens with containers that support pass-by-ref optimizations on Remote interfaces (JBoss does)

Solutions to ClassCastException

- Look for duplicate jars in your EAR and nest WARs
- Remove extra copies in the WAR or disable call-by-ref optimization if remote interfaces are used
- Disable WAR isolation if the container supports it (not recommended)

Domain Based Models (Historic JBoss CL)



Domain Based Models

- Hot deploy requires a ClassLoader per deployment
- However normal class loader isolation disallows pass-by-reference (ClassCastException)
- Domains allow class sharing between class loaders
- Duplicate classes are resolved using first-come-first-serve
- Allows big ball-of-mud
- Domains can also be hierarchical

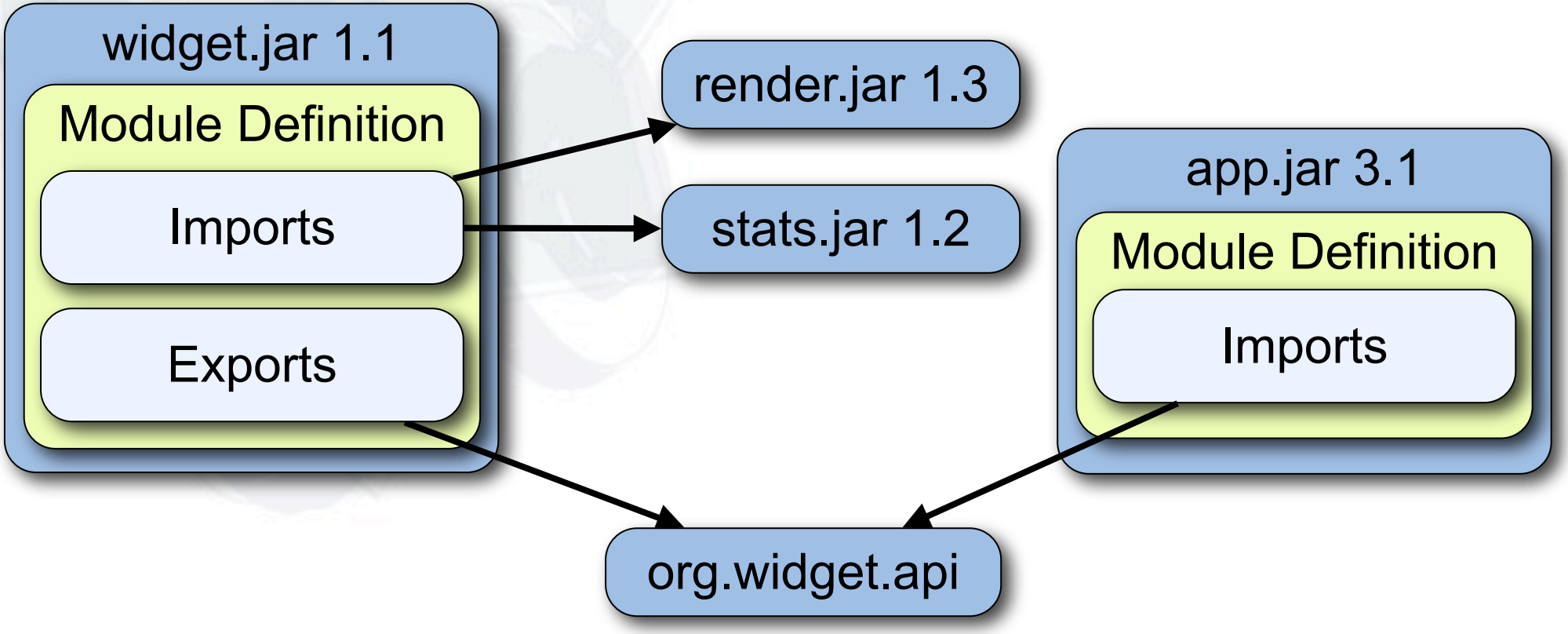
Evolution to Module Class Loaders

- Applications don't always fit hierarchical models
  - Allowing usage of different versions of library traditionally requires copying the jar to a local class loader
  - Cross jar references in a traditional model are not always clear
  - Changes to structure often have unintended side effects

Evolution to Module Class Loaders (cont)

- Modules move the reference behaviors to be per jar
  - Dependencies are expressed on a name and a version
  - Module system responsible for mapping everything

Module ClassLoader Models



OSGi Class Loading

- ClassLoader per bundle for hot deploy
- Supports a complex combination of bundle and package inclusion.

```
Bundle-Name: widget
Bundle-Version: 1.1
Require-Bundle: render;
                bundle-version="1.3"
Require-Bundle: stats;
                bundle-version="1.2"
Import-Package: org.extra;version=”1.4”
Export-Package: org.widget.api
```

JBoss AS5 ClassLoader

- Also supports package and module import/export

```
<classloading xmlns="urn:jboss:classloading:1.0">
   <capabilities>
      <module name="widget" version=”1.1”/>
      <package name=”org.widget.api”/>
   </capabilities>
   <requirements>
      <module name="render" version=”1.3”/>
      <module name="stats" version=”1.2”/>
   </requirements>
</classloading>
```

JBoss AS5 ClassLoader

- Module descriptor can be placed in any deployment
- Will override EE rules
- Also supports domain style loaders

```
<classloading xmlns="urn:jboss:classloading:1.0"
    domain="IsolatedDomain"
    parent-domain="DefaultDomain"
    parent-first="false">
</classloading>
```

Future of Class Loading

- JDK7 probably adopting modules
  - Project jigsaw - JSR294
  - JDK itself will become modularized
- EE7 will likely follow suit
  - Modularized EE Deployments
  - Modularized Application Servers