Project Jigsaw: Under The Hood

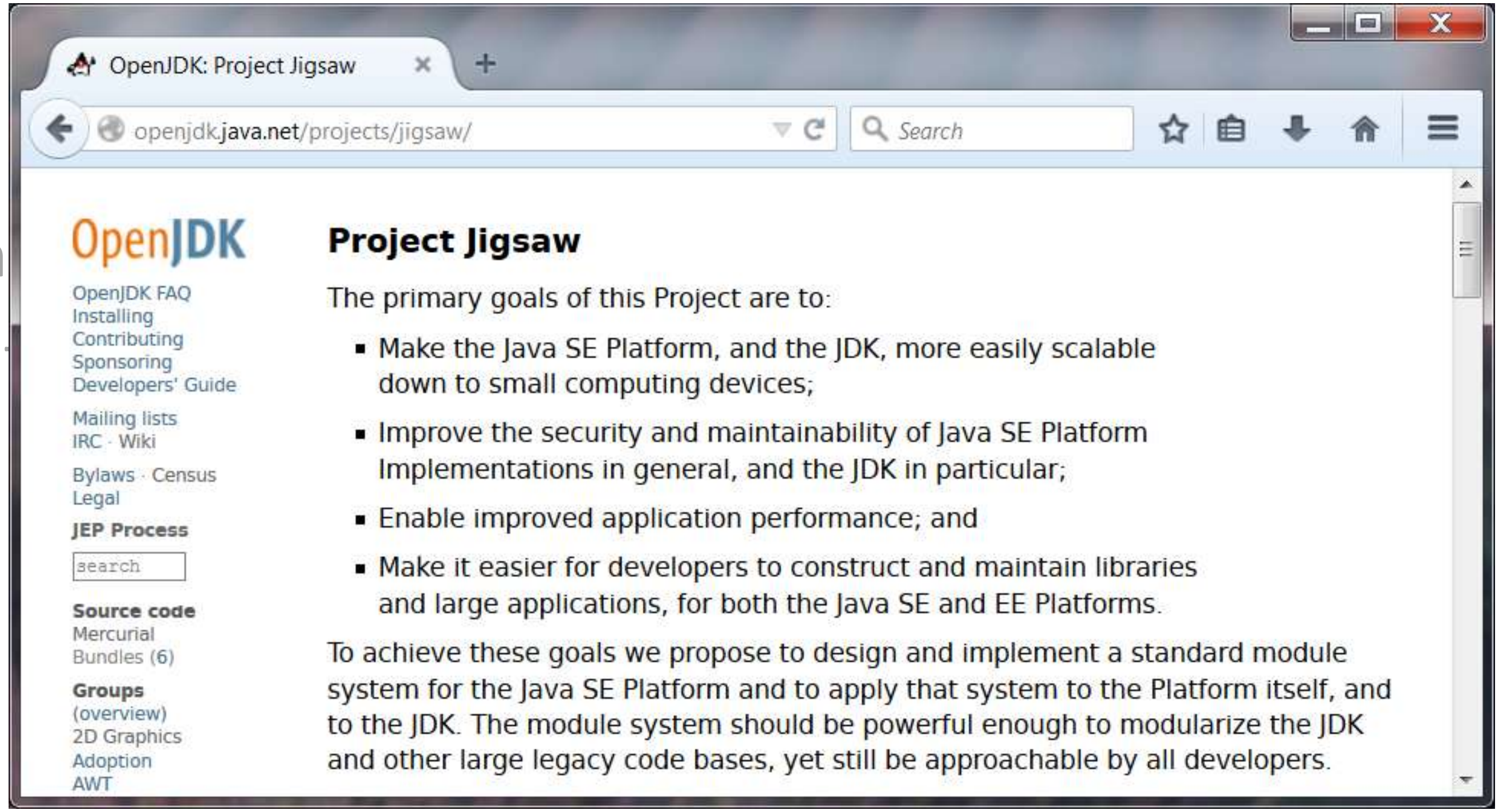
Alex Buckley
Java Platform Group, Oracle
September 2016

JDK 9 At A Glance

- Module System
- Modular JDK
- Language enhancements
- Library enhancements
- Tool enhancements

JDK 9 At A Glance

- Module System
- Modular JDK
- Language enhancem
- Library enhancemen
- Tool enhancements



Project Jigsaw

The primary goals of this Project are to:

- Make the Java SE Platform, and the JDK, more easily scalable down to small computing devices;
- Improve the security and maintainability of Java SE Platform Implementations in general, and the JDK in particular;
- Enable improved application performance; and
- Make it easier for developers to construct and maintain libraries and large applications, for both the Java SE and EE Platforms.

To achieve these goals we propose to design and implement a standard module system for the Java SE Platform and to apply that system to the Platform itself, and to the JDK. The module system should be powerful enough to modularize the JDK and other large legacy code bases, yet still be approachable by all developers.

Project Jigsaw: Under The Hood

Part I: Accessibility and Readability

Accessibility (JDK 1 – JDK 8)

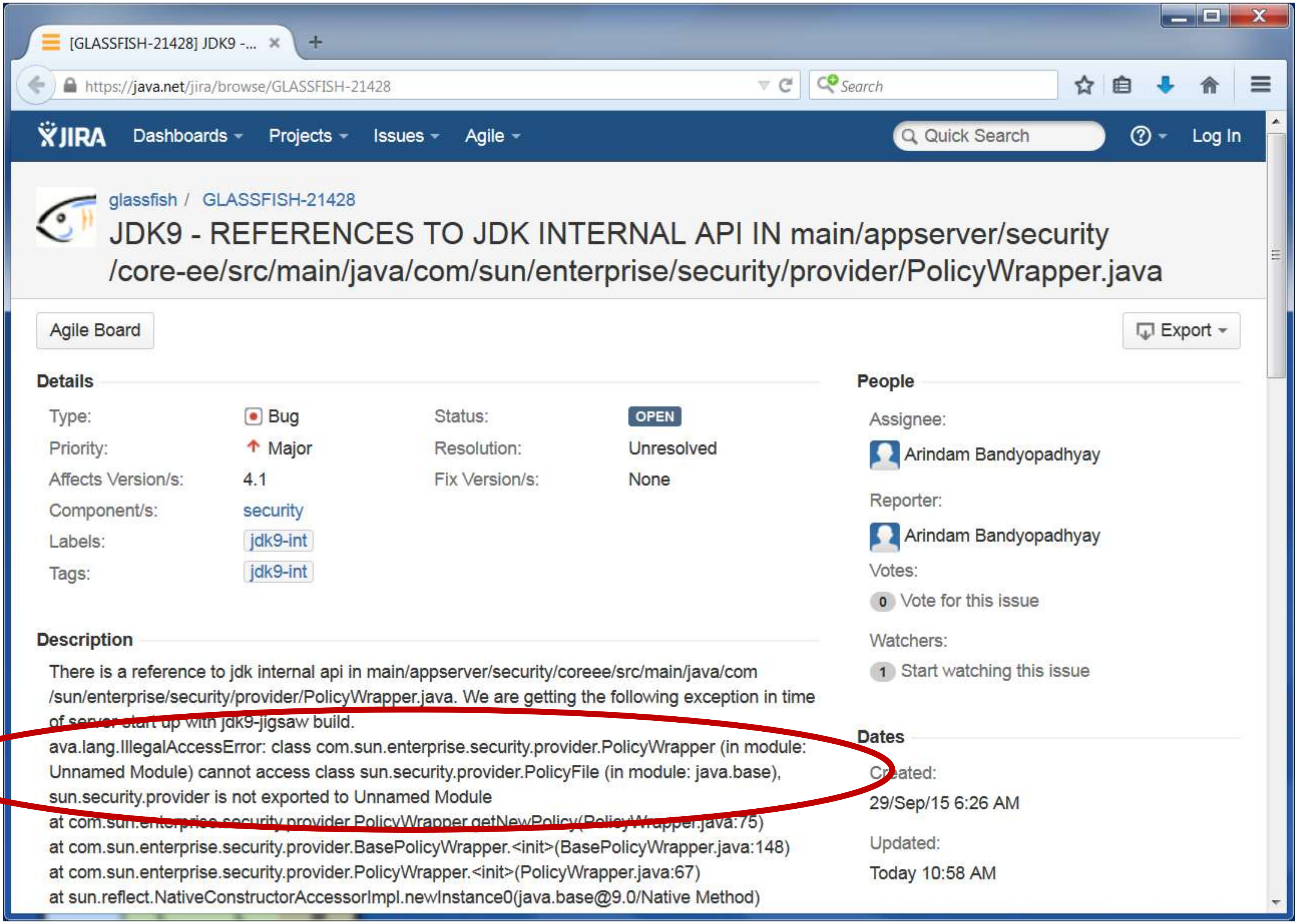
- public
- protected
- <package>
- private

Accessibility (JDK 9 –)

- *public to everyone*
- *public but only to specific modules*
- *public only within a module*
- protected
- <package>
- private

‘public’ no longer means “accessible”.

The result:



glassfish / GLASSFISH-21428

JDK9 - REFERENCES TO JDK INTERNAL API IN main/appserver/security/core-ee/src/main/java/com/sun/enterprise/security/provider/PolicyWrapper.java

Details

Type:	Bug	Status:	OPEN
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	4.1	Fix Version/s:	None
Component/s:	security		
Labels:	jdk9-int		
Tags:	jdk9-int		

People

Assignee: Arindam Bandyopadhyay

Reporter: Arindam Bandyopadhyay

Votes: 0 Vote for this issue

Watchers: 1 Start watching this issue

Dates

Created: 29/Sep/15 6:26 AM

Updated: Today 10:58 AM

Description

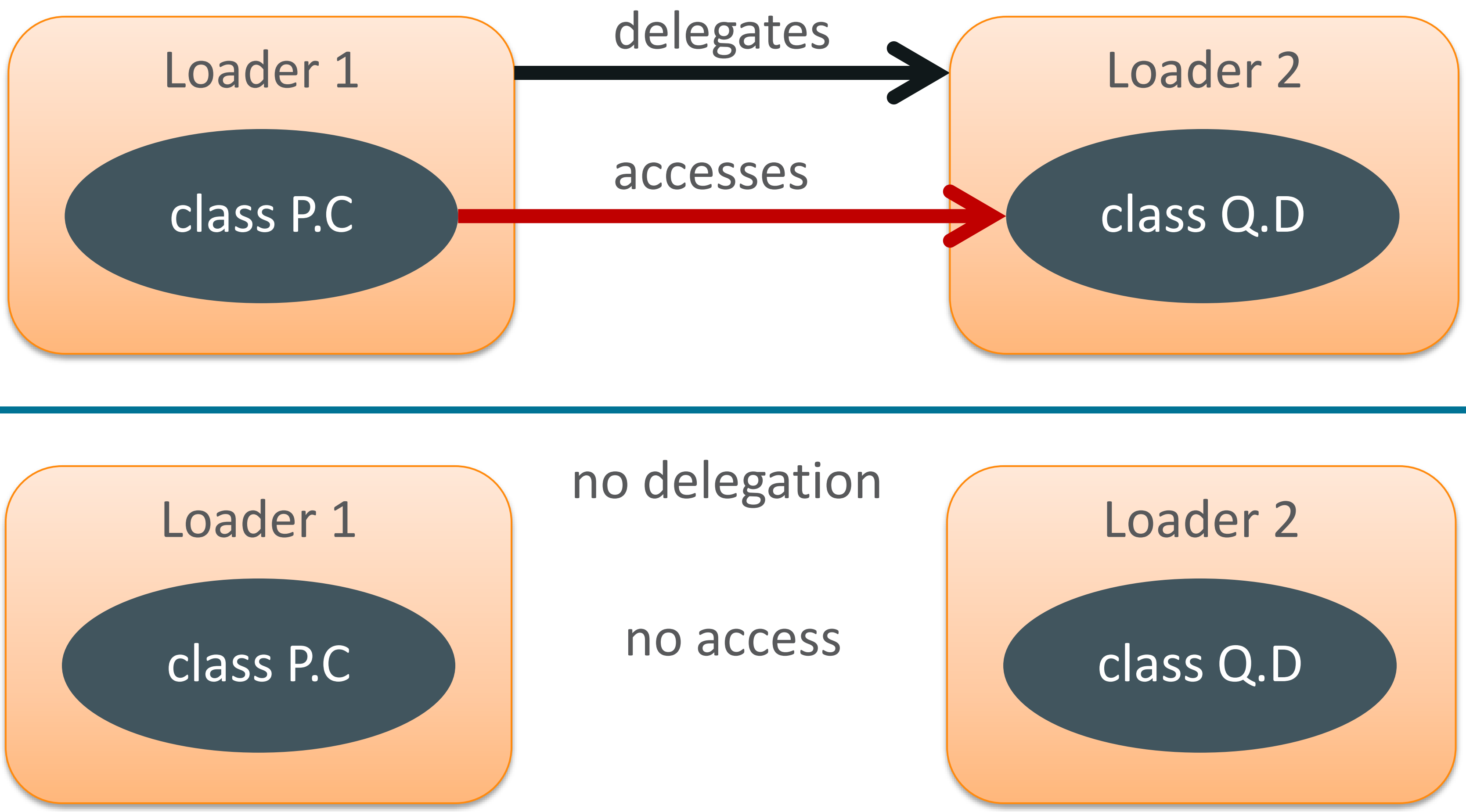
There is a reference to jdk internal api in main/appserver/security/coreee/src/main/java/com/sun/enterprise/security/provider/PolicyWrapper.java. We are getting the following exception in time of server start up with jdk9-jigsaw build.

```
ava.lang.IllegalAccessError: class com.sun.enterprise.security.provider.PolicyWrapper (in module: Unnamed Module) cannot access class sun.security.provider.PolicyFile (in module: java.base), sun.security.provider is not exported to Unnamed Module
at com.sun.enterprise.security.provider.PolicyWrapper.getNewPolicy(PolicyWrapper.java:75)
at com.sun.enterprise.security.provider.BasePolicyWrapper.<init>(BasePolicyWrapper.java:148)
at com.sun.enterprise.security.provider.PolicyWrapper.<init>(PolicyWrapper.java:67)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(java.base@9.0/Native Method)
```

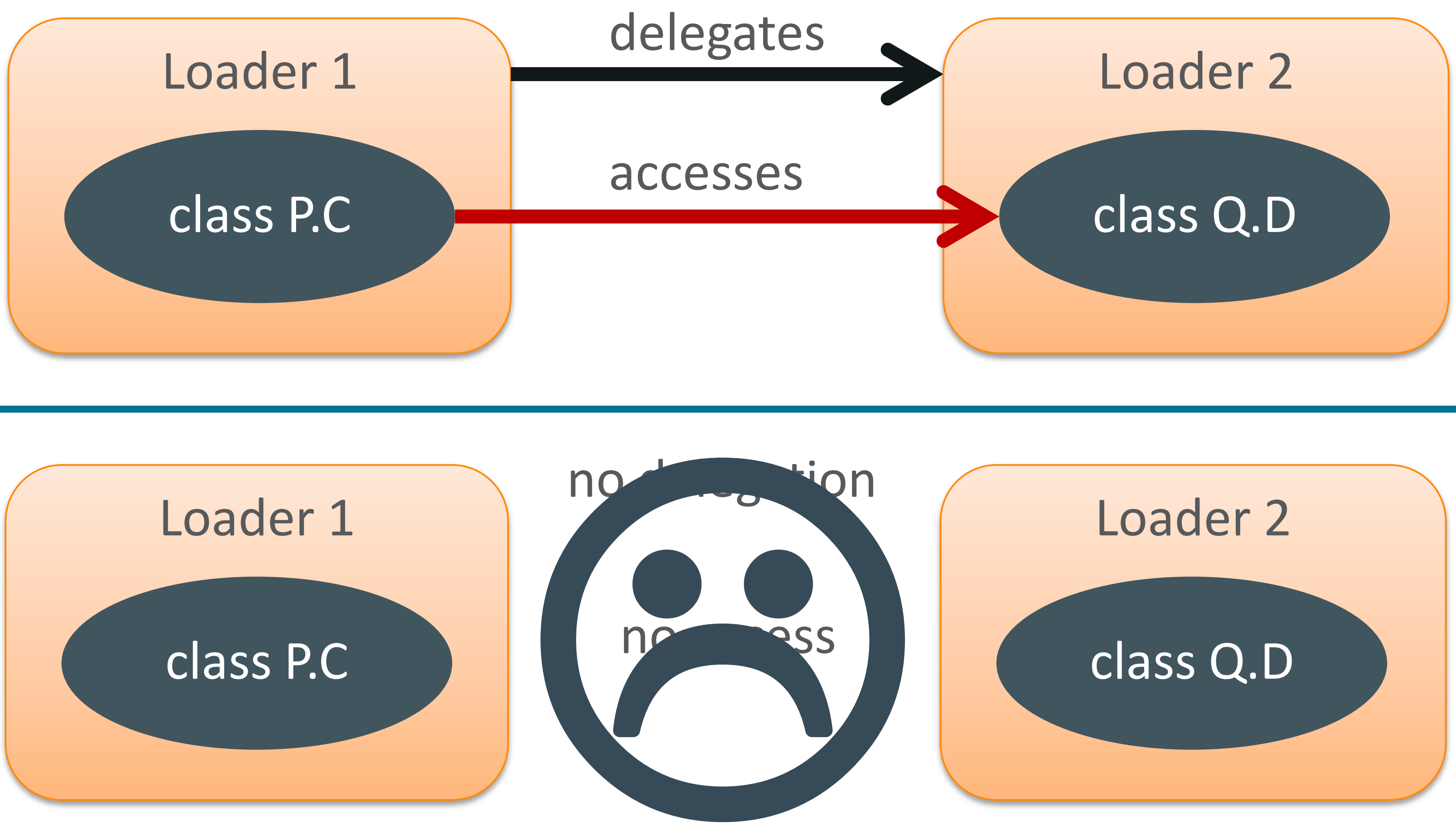
Accessibility and Module Declarations

```
// src/java.sql/module-info.java
module java.sql {
  exports java.sql;
  exports javax.sql;
  exports javax.transaction.xa;
}
```

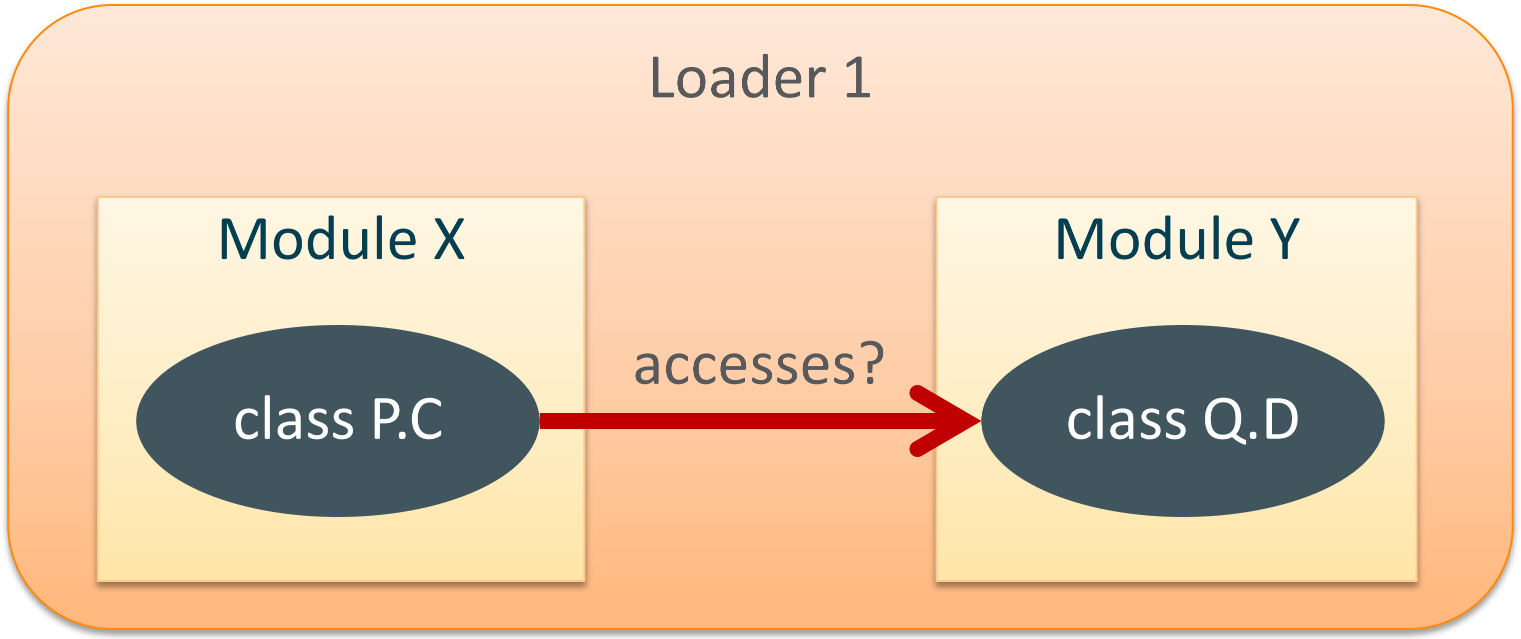
Accessibility and Class Loaders



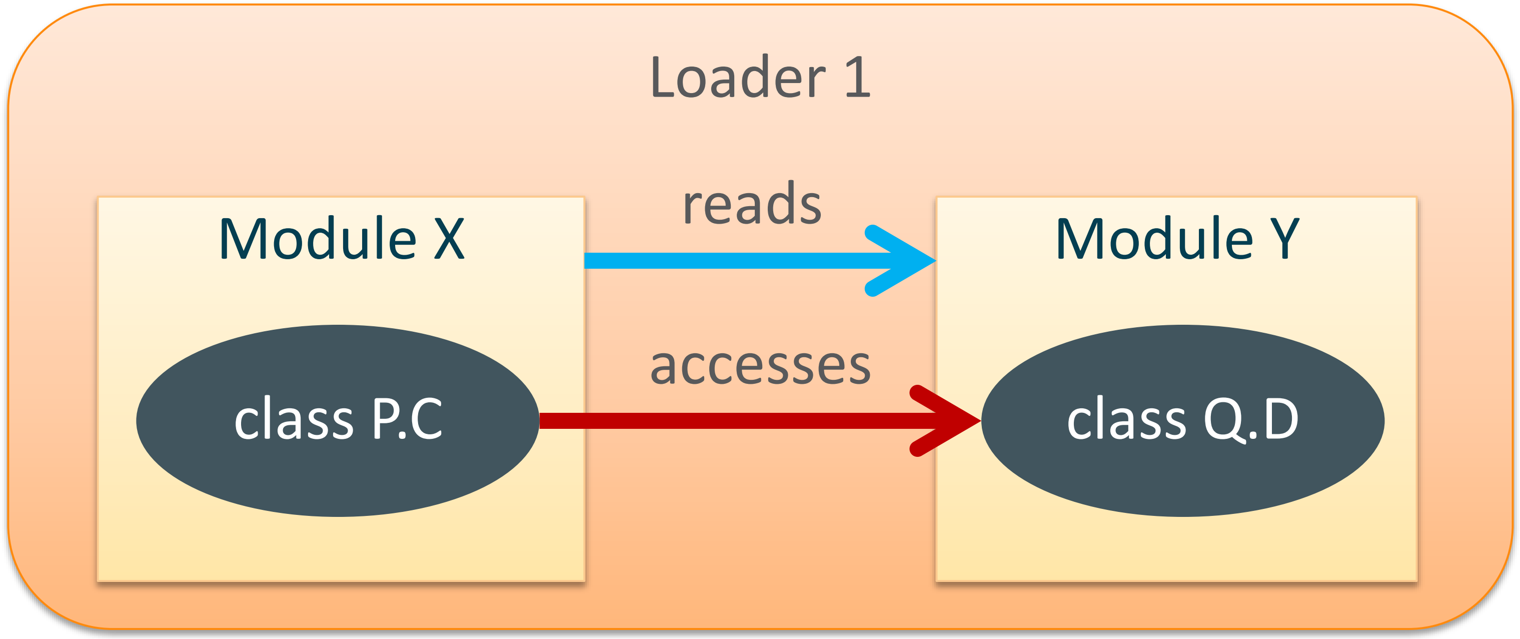
Accessibility and Class Loaders



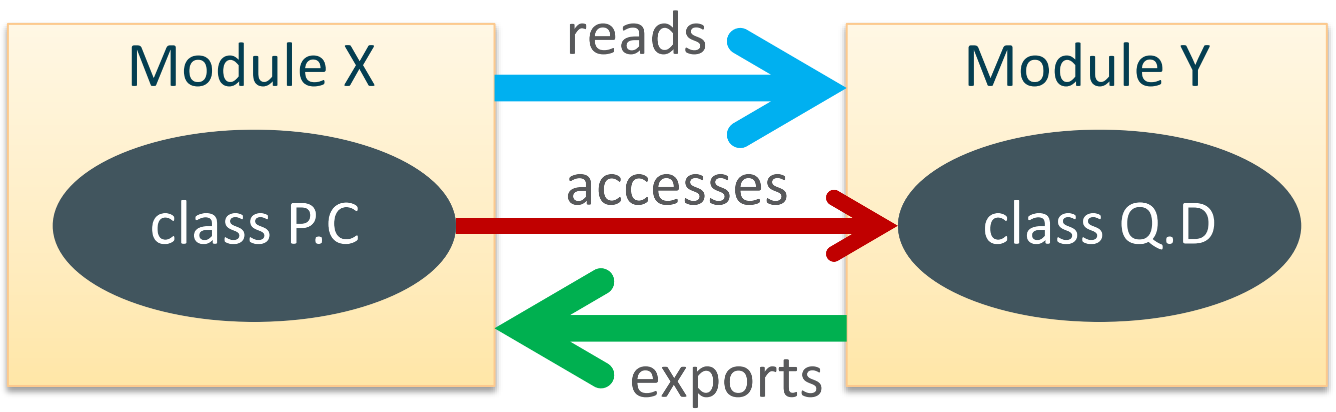
One Loader, Many Modules



The Role of Readability



The Role of Readability



```
module X {
  requires Y;
}
```

```
module Y {
  exports Q;
}
```

Readability in the JDK Module Graph

```
module java.sql {
  requires java.logging;
  exports java.sql;
}
```

```
module java.logging {
  exports java.util.logging;
}
```

```
package java.sql;
import java.util.logging.Logger;
public class DriverManager {
  new Logger() {..}
}
```

```
package java.util.logging;
public class Logger {
  ...
}
```

Readability in the JDK Module Graph

```
module java.sql {
  requires java.logging;
  exports java.sql;
}
```

```
module java.logging {
  exports java.util.logging;
}
```

```
package java.sql;
import java.util.logging.Logger;
public interface Driver {
   Logger getParentLogger();
}
```

```
package java.util.logging;
public class Logger {
  ...
}
```

Readability in the JDK Module Graph

```
module myapp {
  requires java.sql;
  requires java.logging; ☹
}
```

```
module java.sql {
  requires java.logging;
  exports java.sql;
}
```

```
module java.logging {
  exports java.util.logging;
}
```

Readability in the JDK Module Graph

```
module myapp {
  requires java.sql;
  ~~requires java.logging;~~ ☺
}
```

```
module java.sql {
  requires transitive java.logging;
  exports java.sql;
}
```

```
module java.logging {
  exports java.util.logging;
}
```

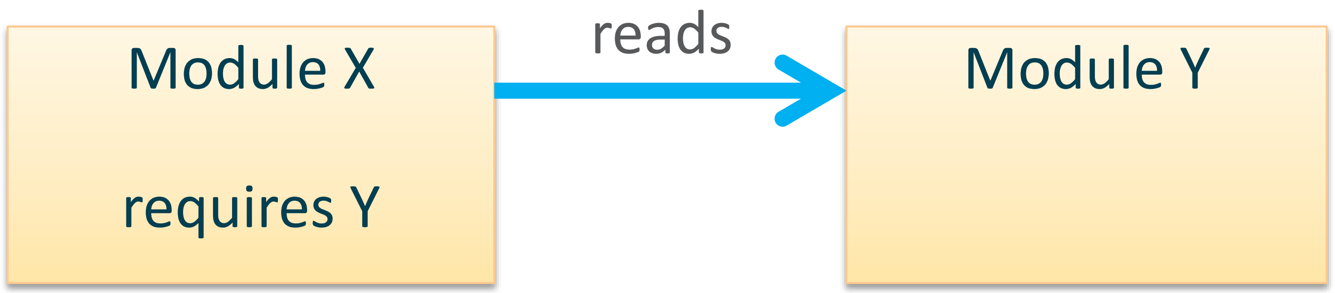
Readability in the JDK Module Graph

```
module myapp {
  requires java.sql;
  ~~requires java.logging;~~ ☺
}
```

```
module java.sql {                       module java.logging {
  requires transitive java.logging;       requires transitive logextras;
  exports java.sql;                       exports java.util.logging;
}                                       }
                                        module logextras { … }
```

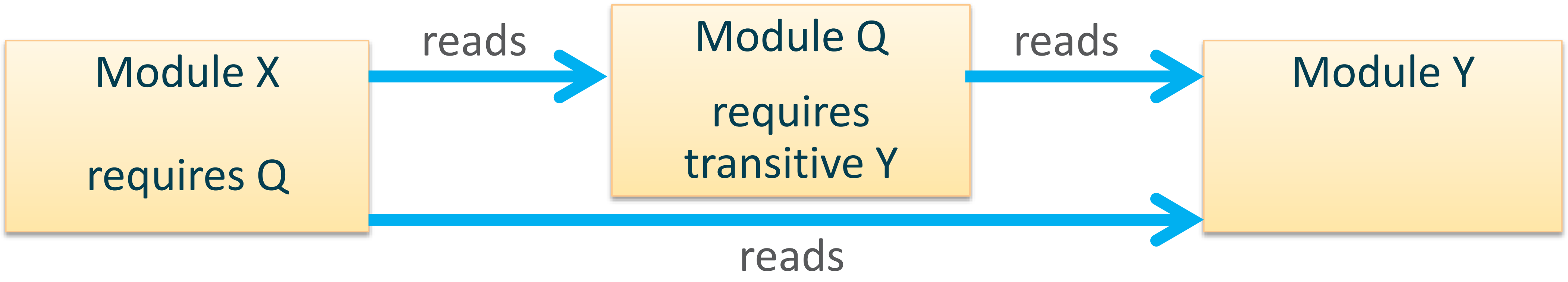
Direct and Implied Readability

- X reads Y if:
  - X requires Y



  or

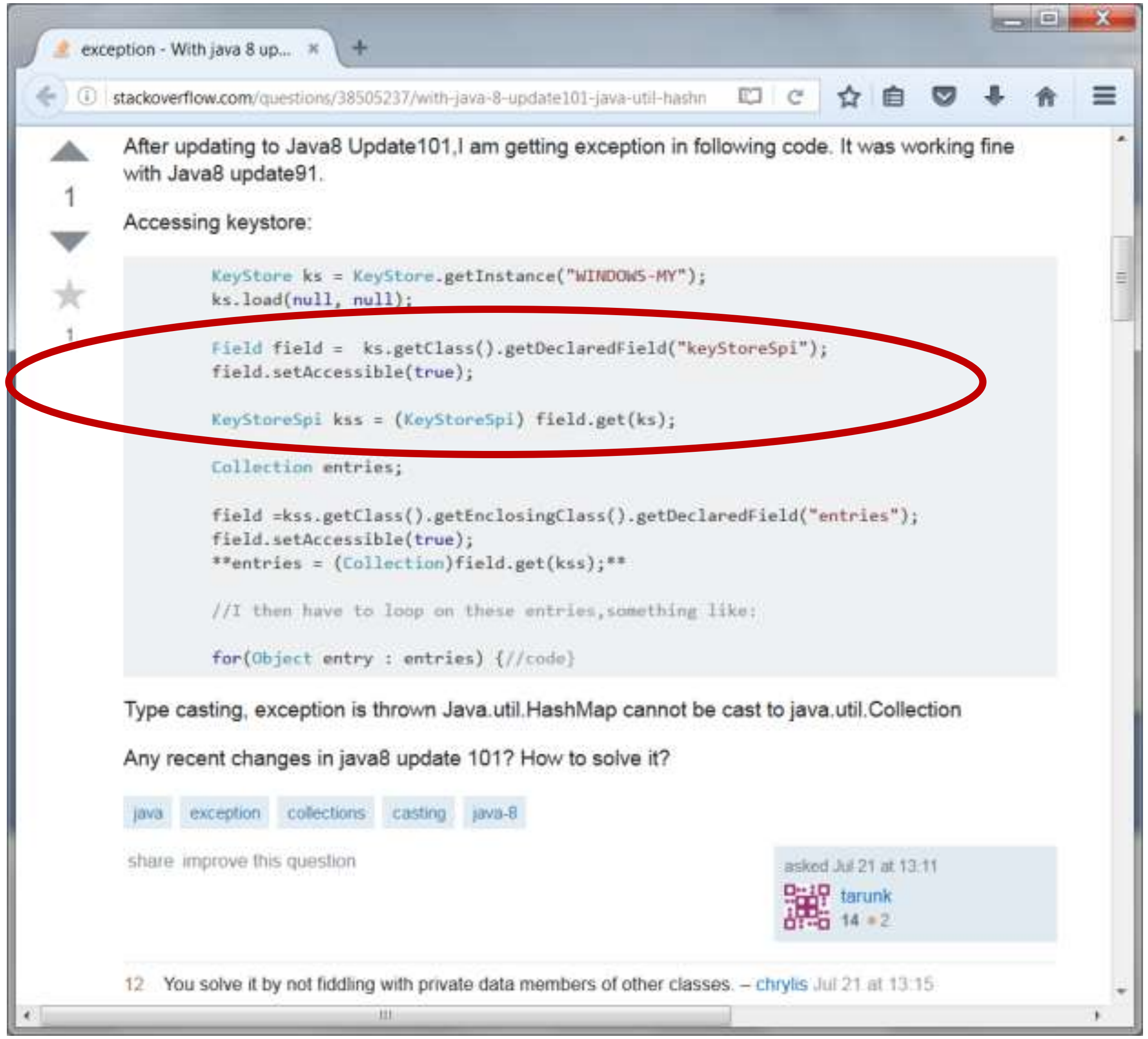
  - X reads Q, and Q requires transitive Y



Exports and Accessibility

```
module java.base {
  exports java.io;
  exports java.lang;
  exports java.lang.annotation;
  exports java.lang.invoke;
  exports java.lang.module;
  exports java.lang.ref;
  exports java.lang.reflect;
  exports java.math;
  exports java.net;
  exports java.nio;
  exports java.security;
  exports java.time;
  exports java.util;
  exports java.util.concurrent;
  exports java.util.function;
  exports java.util.stream;
  ....
```

http://stackoverflow.com/questions/38505237/with-java-8-update101-java-util-hashmap-cannot-be-cast-to-java-util-collection


exception - With java 8 up...
stackoverflow.com/questions/38505237/with-java-8-update101-java-util-hashn

After updating to Java8 Update101,I am getting exception in following code. It was working fine with Java8 update91.

Accessing keystore:

```
KeyStore ks = KeyStore.getInstance("WINDOWS-MY");
ks.load(null, null);

Field field =  ks.getClass().getDeclaredField("keyStoreSpi");
field.setAccessible(true);

KeyStoreSpi kss = (KeyStoreSpi) field.get(ks);

Collection entries;

field =kss.getClass().getEnclosingClass().getDeclaredField("entries");
field.setAccessible(true);
**entries = (Collection)field.get(kss);**

//I then have to loop on these entries,something like:

for(Object entry : entries) {//code}
```

Type casting, exception is thrown Java.util.HashMap cannot be cast to java.util.Collection

Any recent changes in java8 update 101? How to solve it?

java exception collections casting java-8

share improve this question

asked Jul 21 at 13:11
tarunk
14 • 2

12 You solve it by not fiddling with private data members of other classes. – chrylis Jul 21 at 13:15

Strong Encapsulation under Reflection

```
module java.base {
   exports java.security;
```

```
setAccessible(true)
```

```
module myapp {
   exports private com.myapp.lib;
```

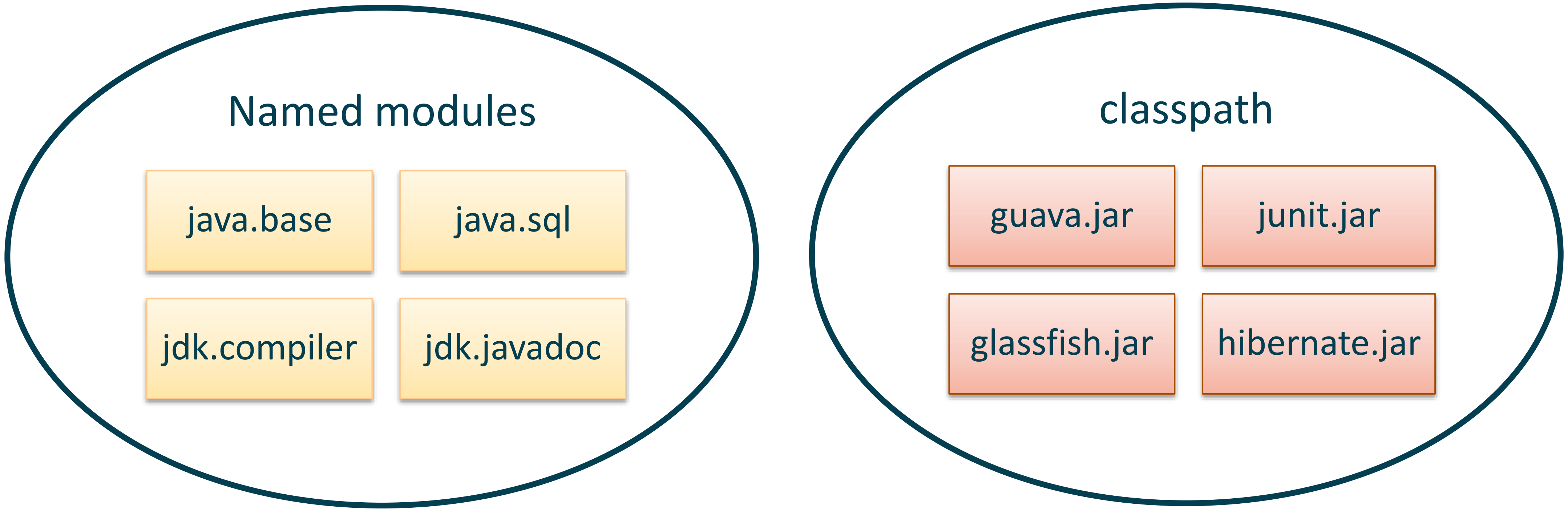
```
setAccessible(true)
```

Summary of Part I: Accessibility and Readability

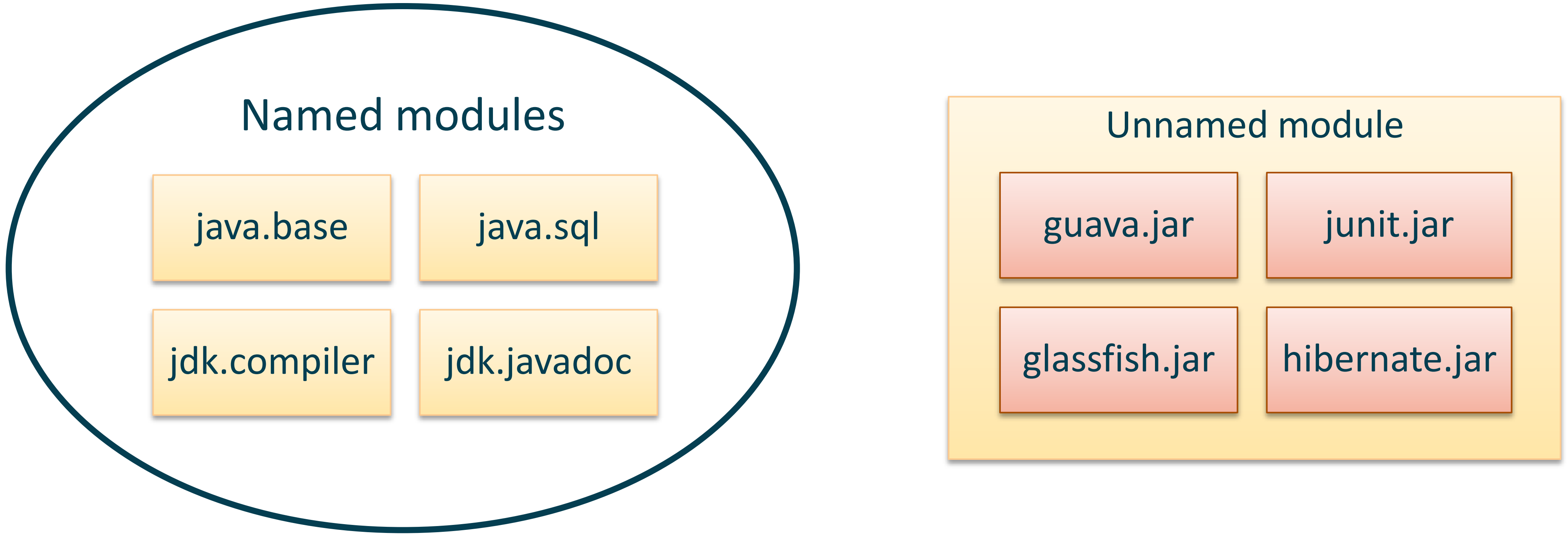
- Accessibility used to be a simple check for ‘public’ or “same package”.
- In JDK 9, accessibility strongly encapsulates module internals.
- Accessibility relies on readability, which can be direct or implied.
- **Accessibility is enforced by the compiler, VM, and Reflection.**

Part II: Different Kinds of Modules

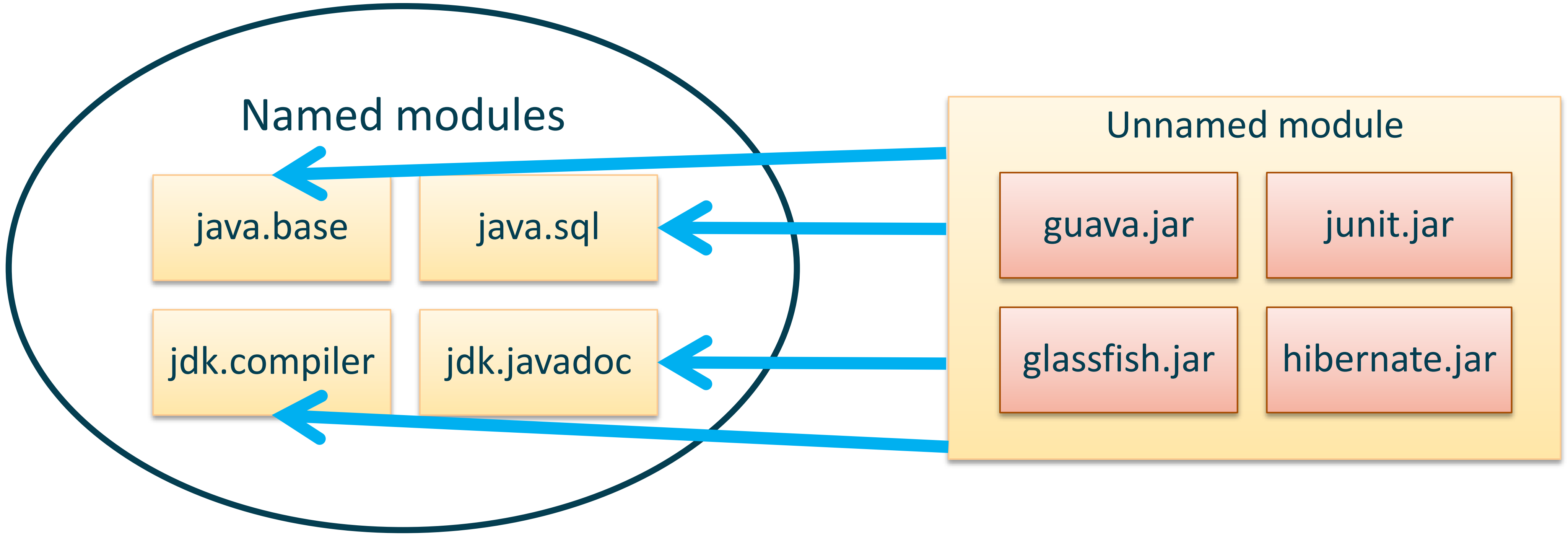
Named Modules



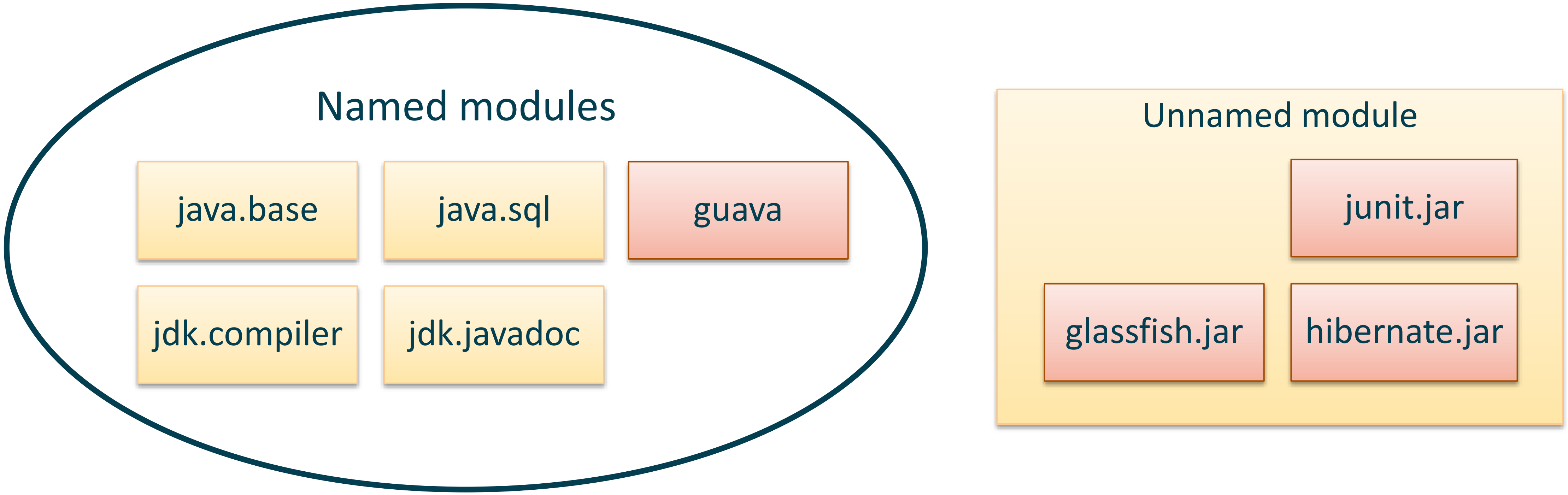
The Unnamed Module



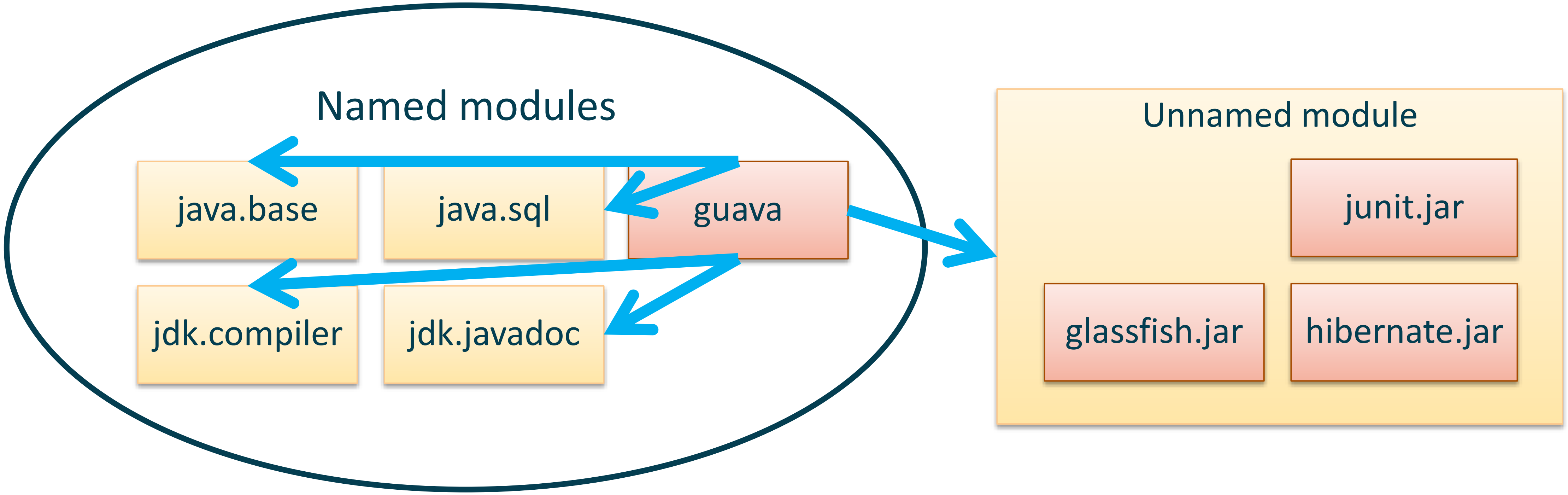
The Unnamed Module



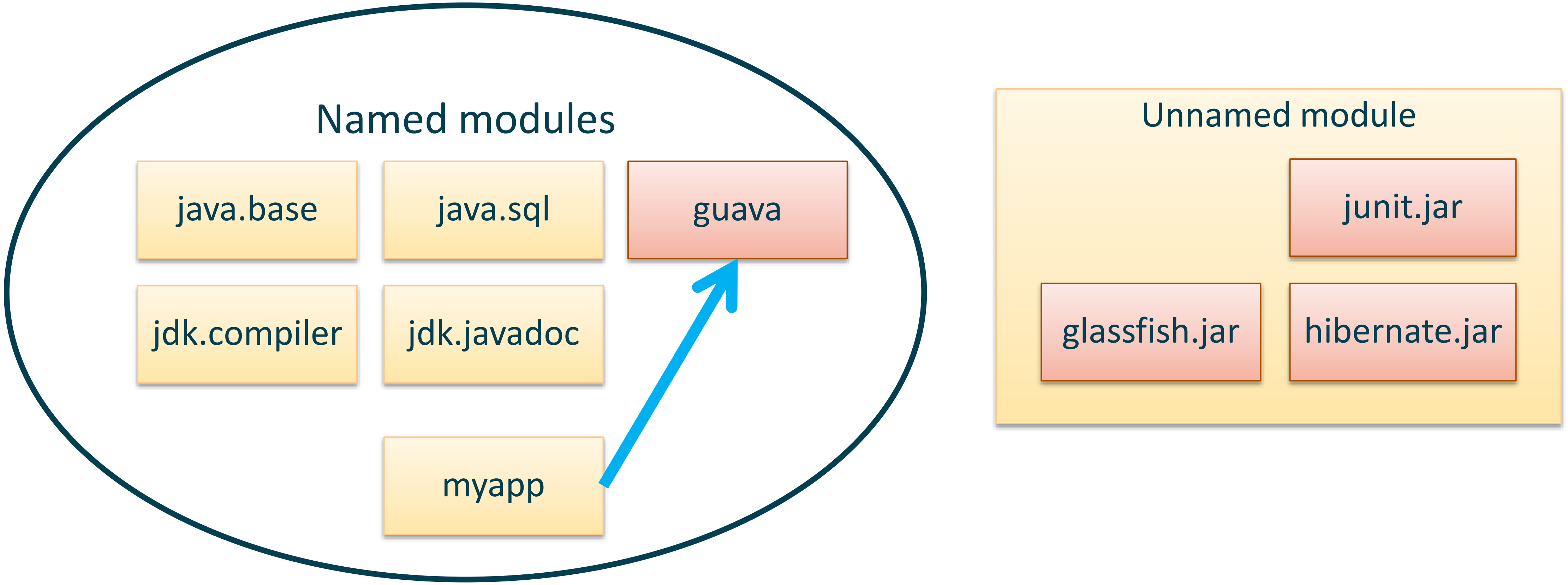
Automatic Modules



Automatic Modules



Automatic Modules



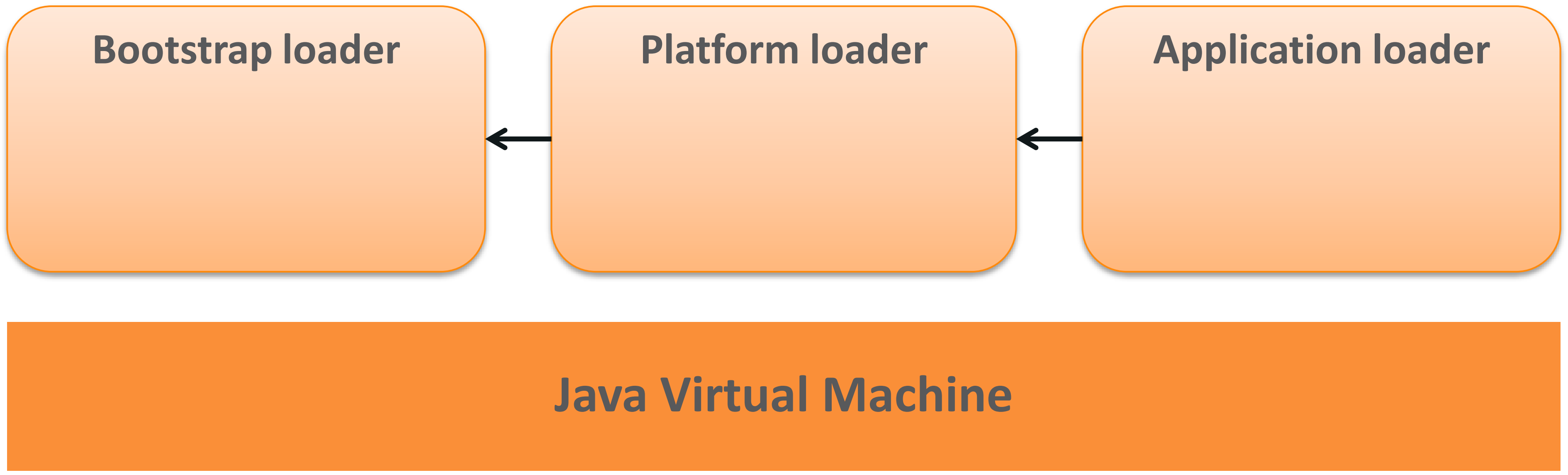
Summary of Part II: Different Kinds of Modules

- Explicit named modules (java.sql)
- Automatic named modules (guava)
- Unnamed module (a.k.a. classpath)
- **Lots of readability “for free” to help with migration.**

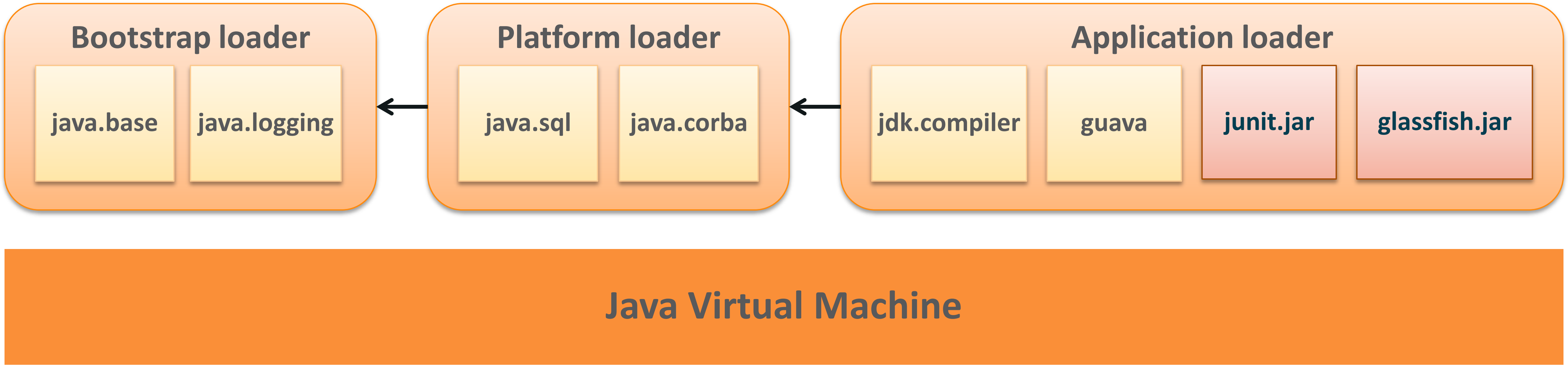
Part III: Loaders and Layers

Class loading doesn’t change.

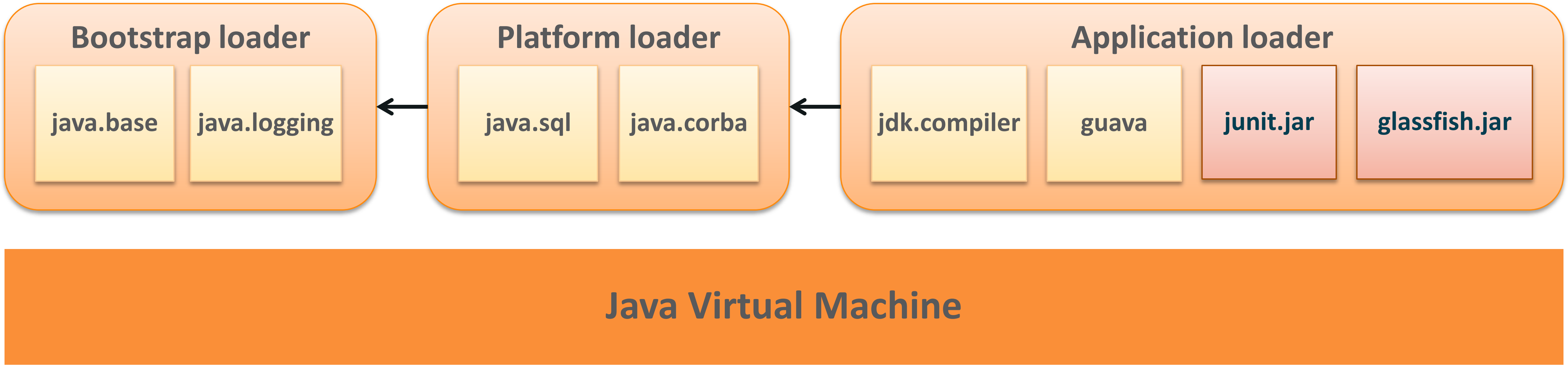
Class Loading in JDK 9



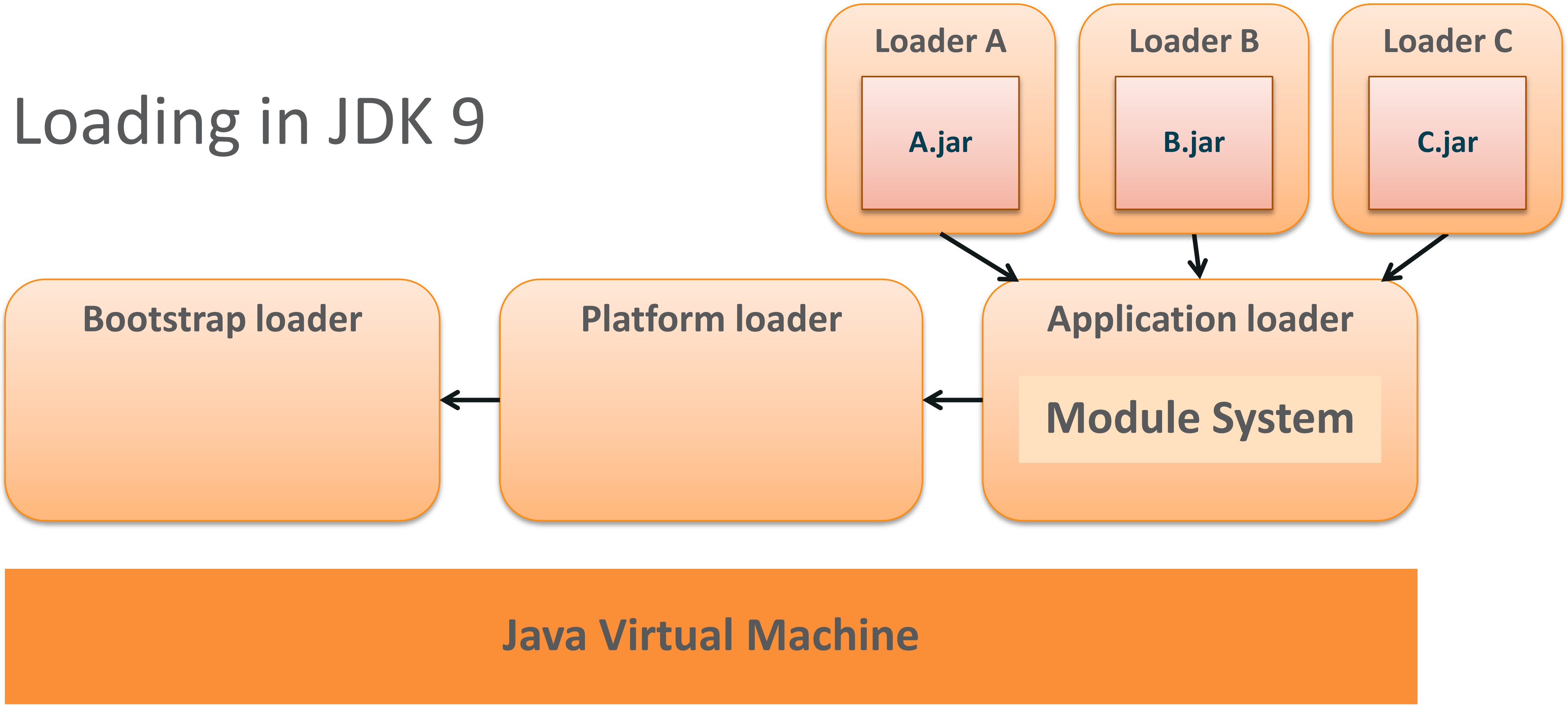
Class Loading in JDK 9



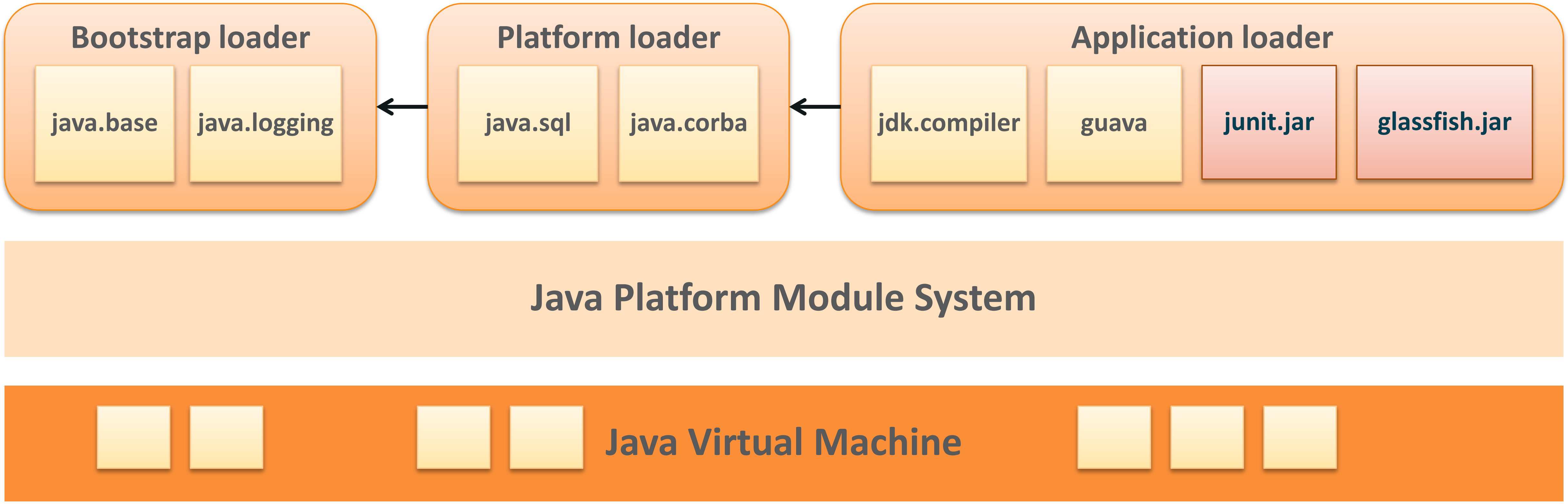
Class Loading in JDK 9



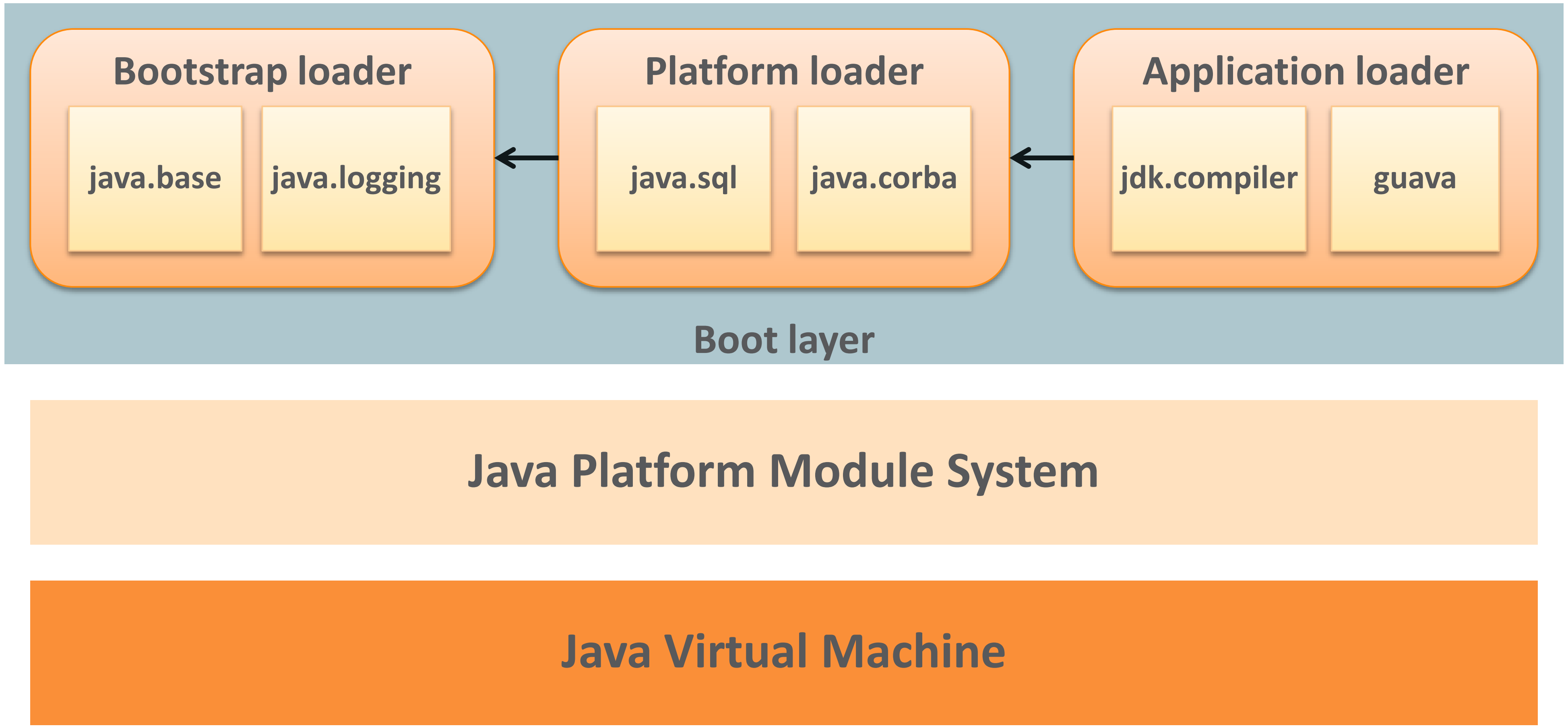
Class Loading in JDK 9



Modular Class Loading in JDK 9



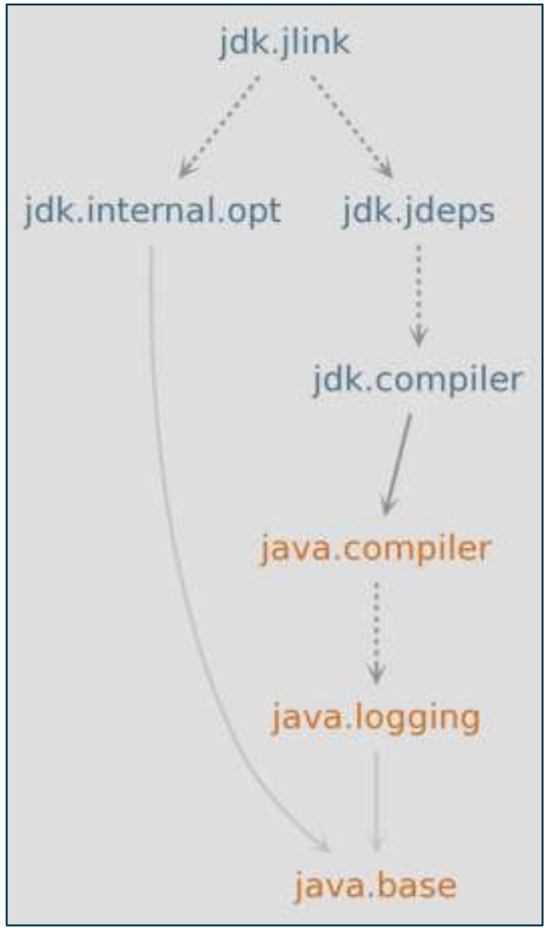
Layers



Bootstrap loader
java.base
java.logging
Platform loader
java.sql
java.corba
Application loader
jdk.compiler
guava
Boot layer
Java Platform Module System
Java Virtual Machine

Layer Creation

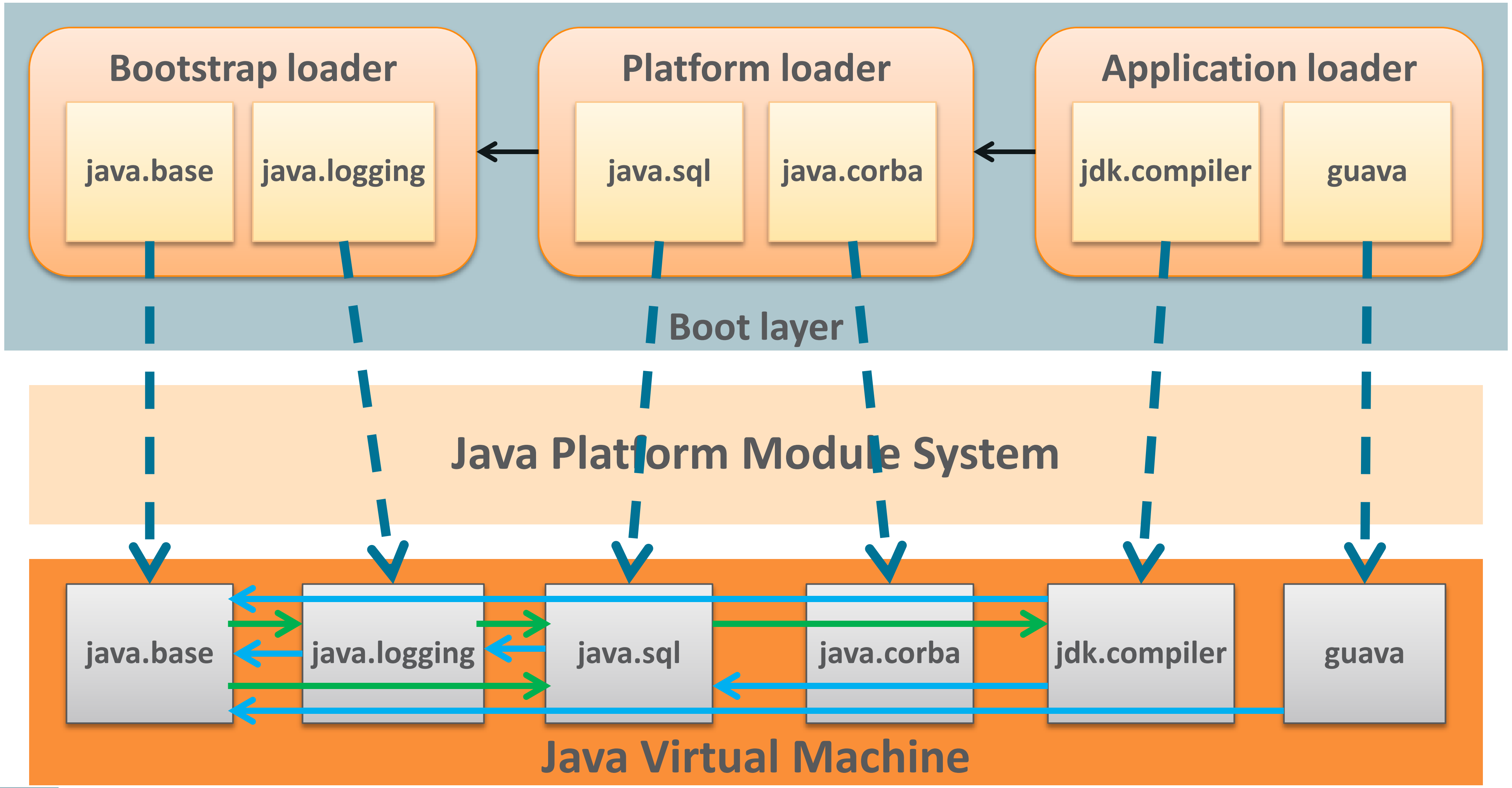
(1)



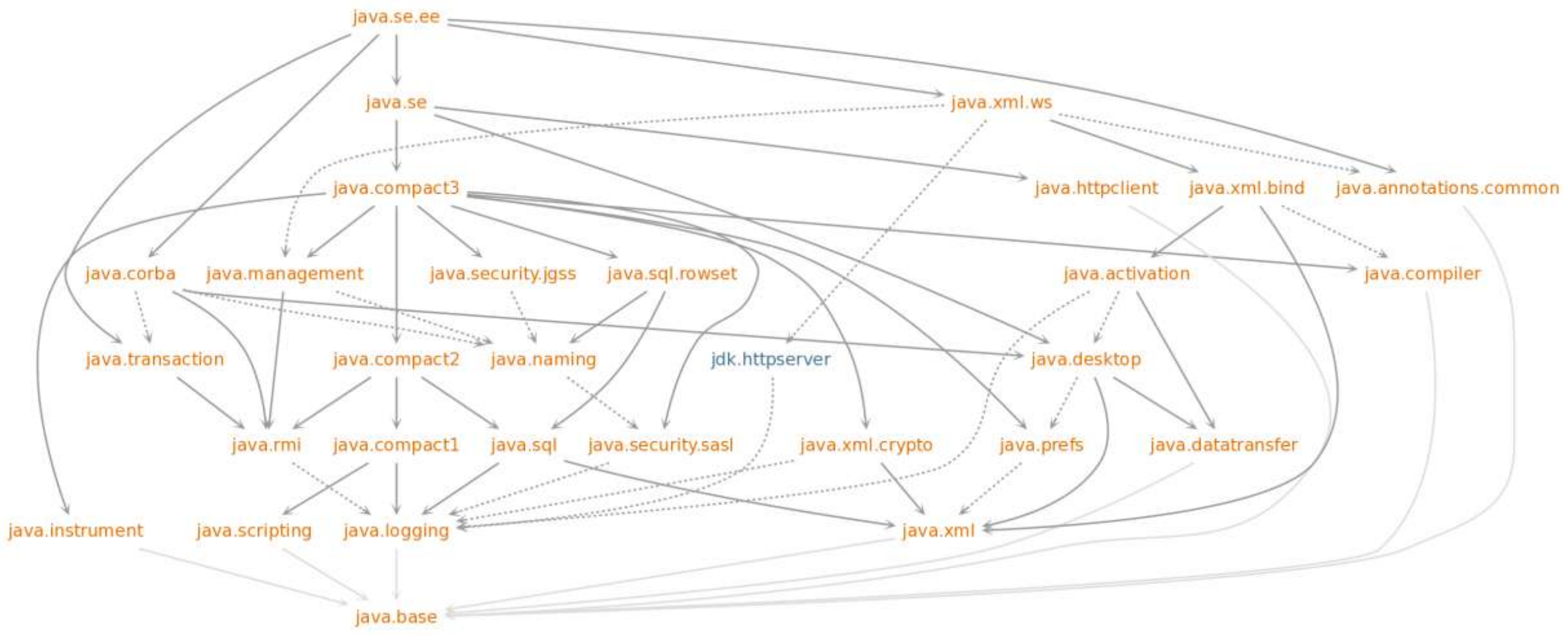
(2)

```
String moduleName -> {
    switch (moduleName) {
        case “java.base”:
        case “java.logging”:
            return BOOTSTRAP_LDR;
        default:
            return APP_LDR;
    }
}
```

Layers and the VM

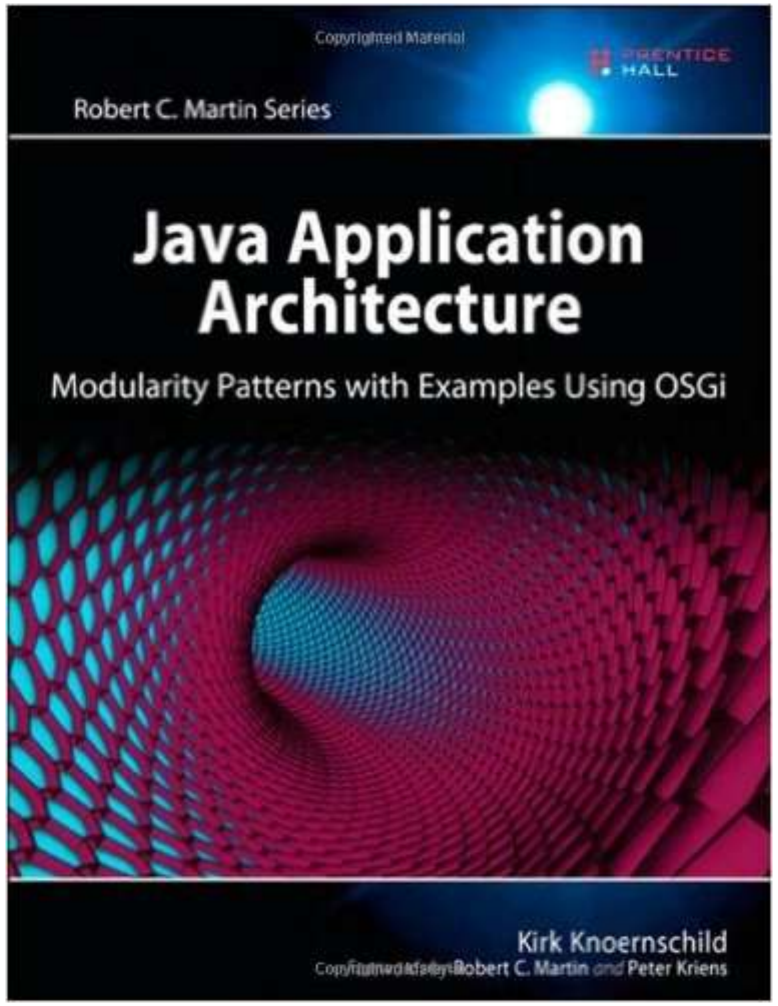


Well-Formed Graphs



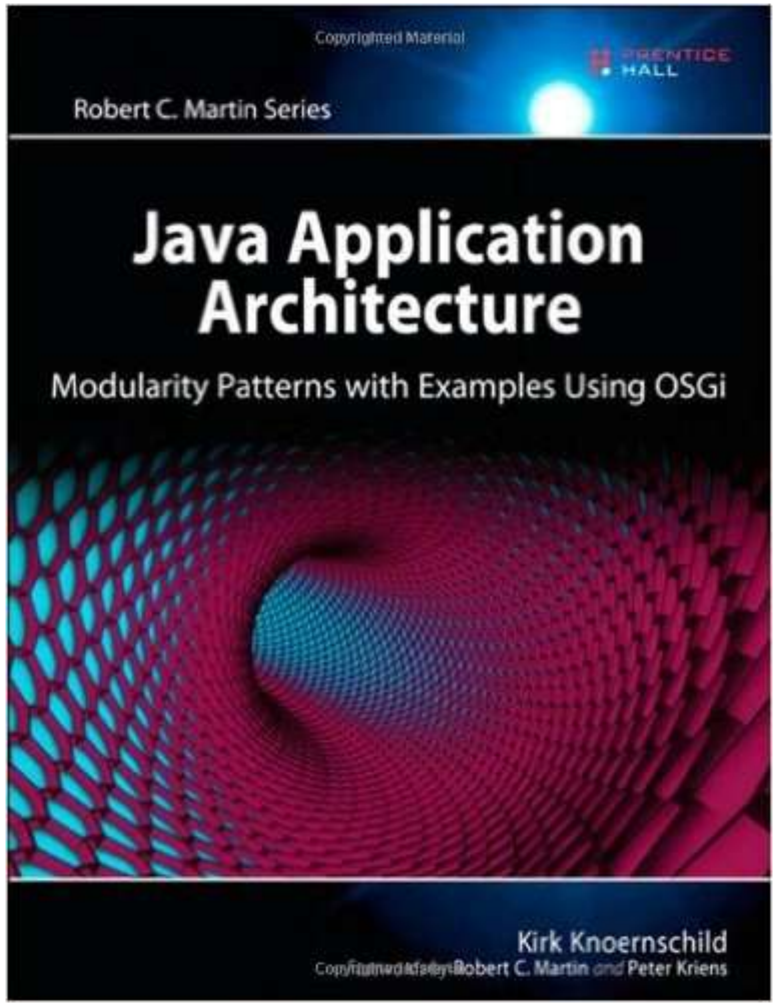
Well-Formed Graphs

"Excessive dependencies are bad. But, cyclic dependencies are especially bad."


Copyrighted Material
PRENTICE HALL
Robert C. Martin Series
Java Application Architecture
Modularity Patterns with Examples Using OSGi
Kirk Knoernschild
Forewords by Robert C. Martin and Peter Kriens

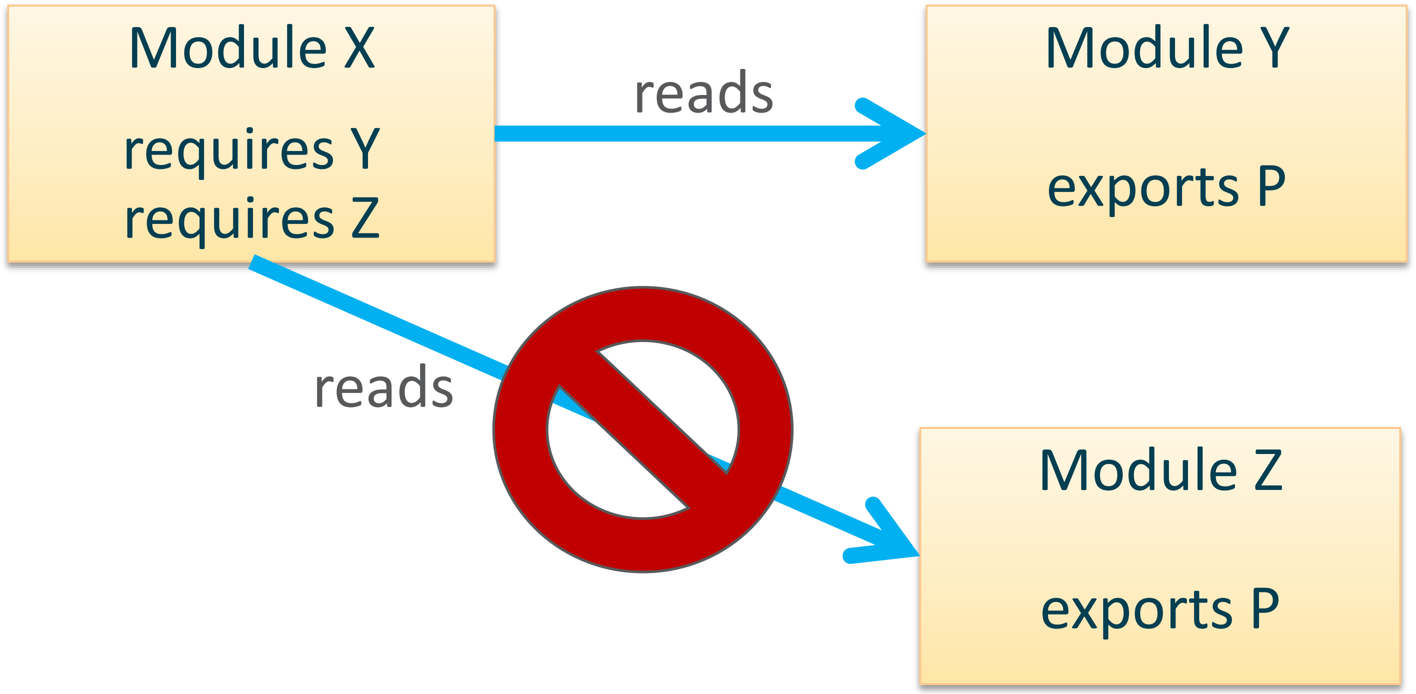
Well-Formed Graphs

“Generally speaking, cycles are always bad!
However, some cycles are worse than others.
Cycles among classes are tolerable, assuming they don’t cause cycles among the packages or modules containing them.
Cycles among packages may also be tolerable, assuming they don’t cause cycles among the modules containing them.
Module relationships must never be cyclic.”


Copyrighted Material
PRENTICE HALL
Robert C. Martin Series
Java Application Architecture
Modularity Patterns with Examples Using OSGi
Kirk Knoernschild
Forewords by Robert C. Martin and Peter Kriens

Well-Formed Graphs

- A module must read only one module that exports a package called P.



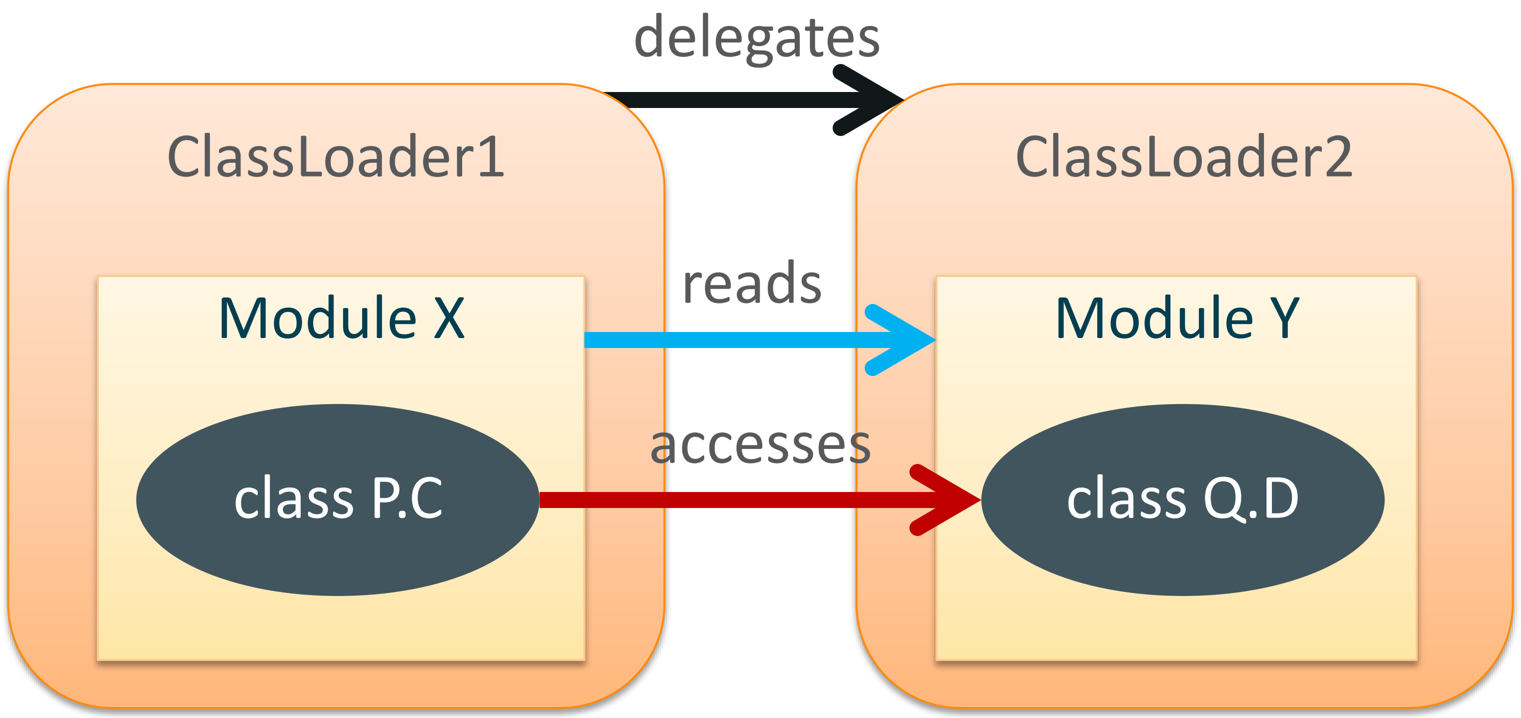
Well-Formed Maps

- Different modules with the same package must map to different loaders.

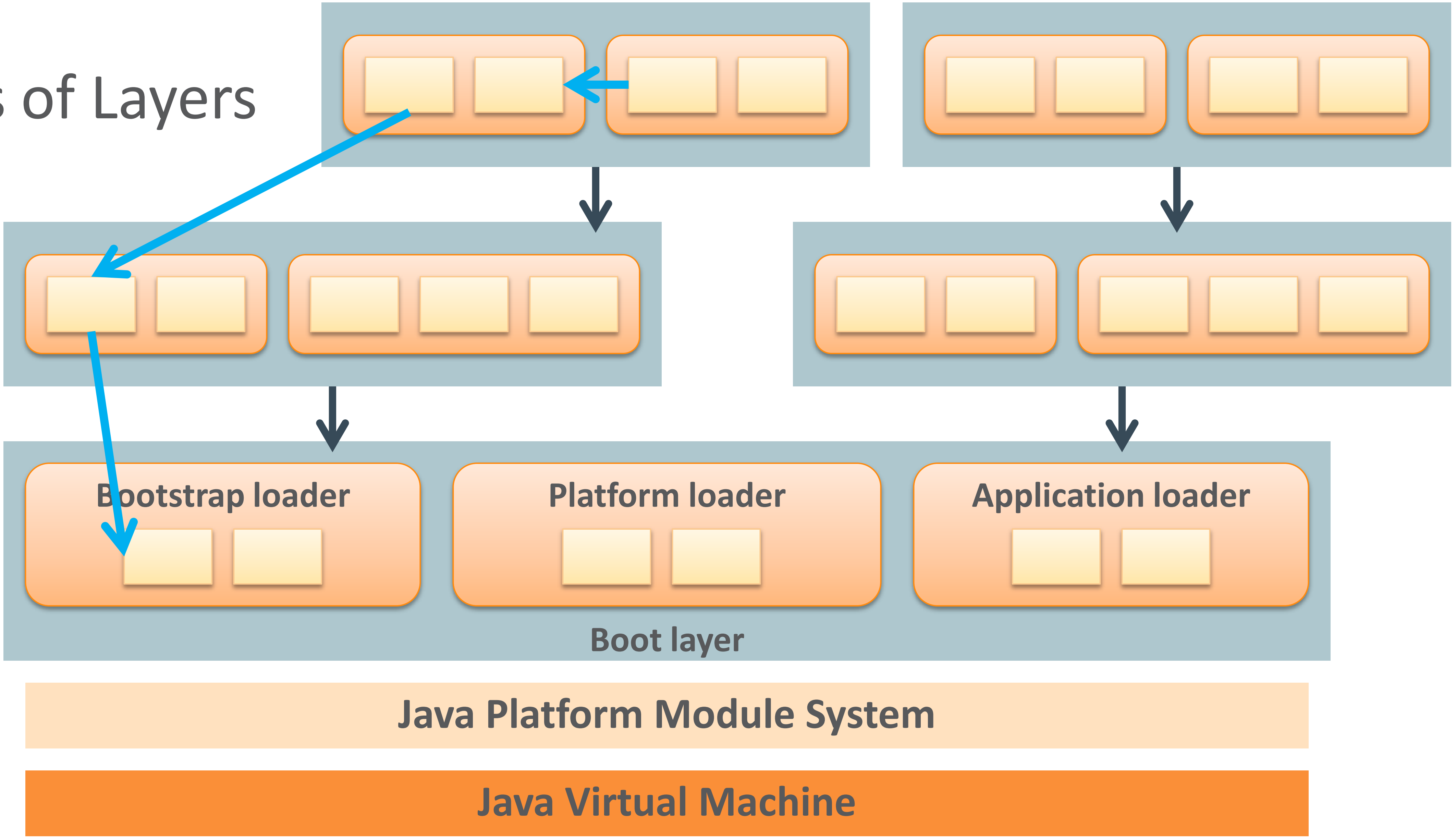
```
String moduleName -> {
   switch (moduleName) {
      case “java.base”:
      case “java.logging”:
         return BOOTSTRAP_LDR;
      default:
         return APP_LDR;
   }
}
```

Well-Formed Maps

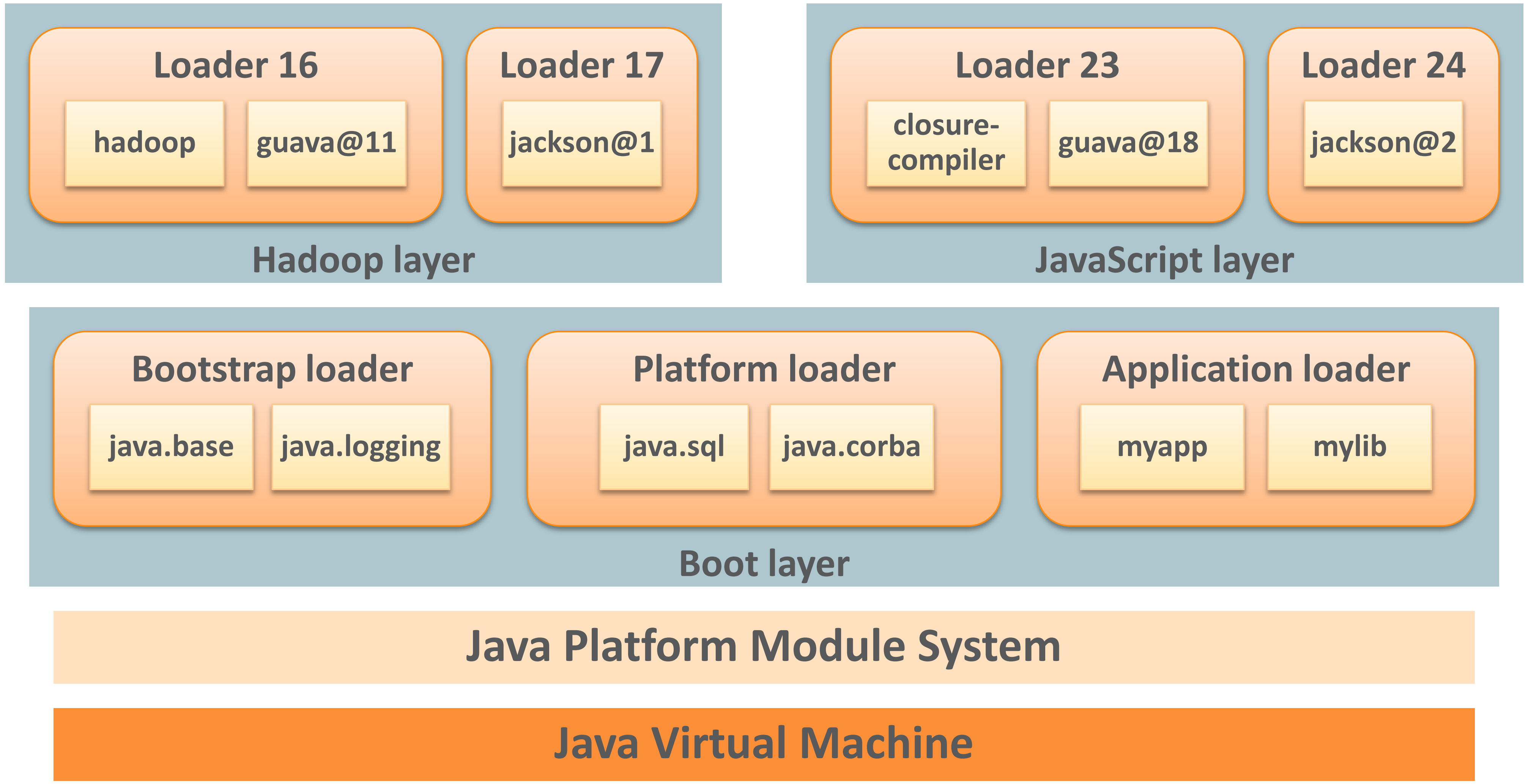
- Loader delegation must respect module readability.



Layers of Layers



Layers and Versions



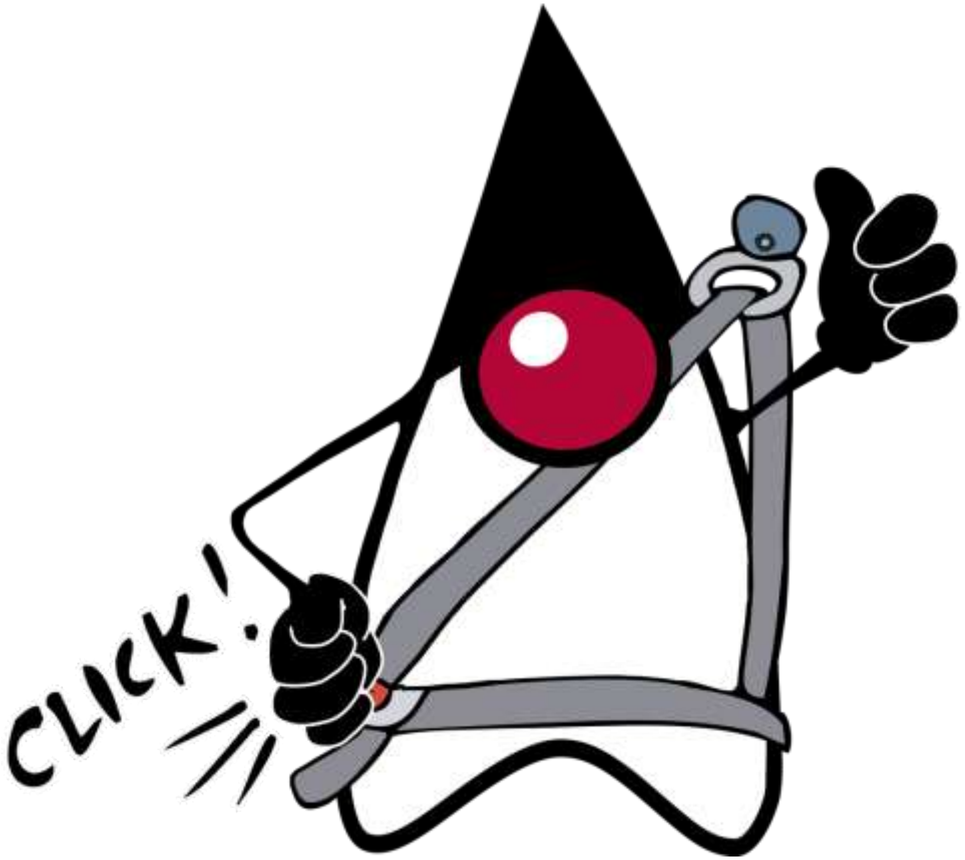
Summary of Part III: Loaders and Layers

- Modules do a better job of encapsulation than class loaders, but class loaders are still necessary.
- Layers control the relationship between modules and class loaders.
- **Assuming class loaders respect the module graph, the system is safe by construction – no cycles or split packages.**

Summary of Summaries

- **Strong encapsulation of modules by the compiler, VM, Core Reflection.**
- **Unnamed and automatic modules help with migration.**
- **The system is safe by construction – no cycles or split packages.**

The Module System: A Seat Belt, Not A Jetpack



Part IV: The Road Ahead

Incompatible Changes in JDK 9

- **java.util.{logging,jar}, java.awt[.dnd].peer**
- **org.omg.CORBA, javax.rmi, javax.xml.{bind,ws}, javax.annotation**
- **java[.vm][.specification].version**

- sun.misc
- sun.net.www, sun.security.x509, com.sun.org.apache.xerces.internal.jaxp
- rt.jar, tools.jar, -Xbootclasspath/p

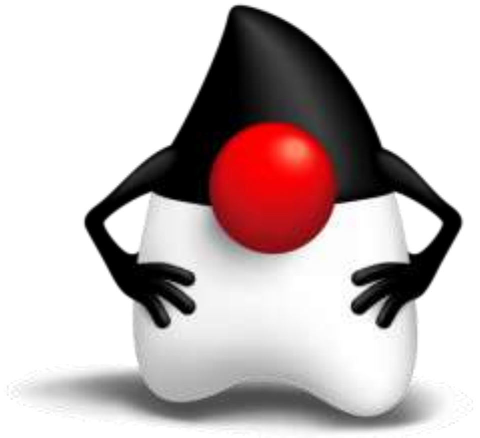
Incompatible Changes in JDK 9

- java.util.{logging,jar},   java.awt[.dnd].peer
- org.omg.CORBA,   javax.rmi,   javax.xml.{bind,ws},   javax.annotation
- java[.vm][.specification].version

- **sun.misc**
- **sun.net.www, sun.security.x509, com.sun.org.apache.xerces.internal.jaxp**
- **rt.jar, tools.jar, -Xbootclasspath/p**

Preparing for JDK 9

- JDK 8: Run *jdeps –jdkinternals MyApp.jar*
- JDK 9: Early Access binaries at http://jdk9.java.net/
- JEP 261: Module System
- JEP 260: Encapsulate Most Internal APIs
- JEP 223: New Version String Scheme
- JEP 220: Modular Run-Time Images
- JEP 200: The Modular JDK



Safe Harbor Statement

ORACLE®