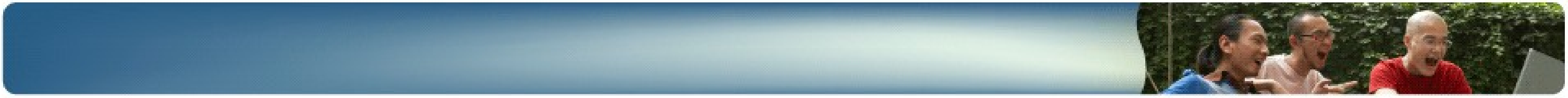
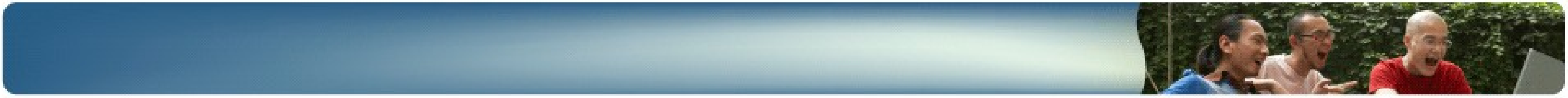
JDK 7 Overview

Sang Shin
Michèle Garoche
www.javapassion.com
“Learn with Passion!”



JDK 7 Features from Sun: Big

- Modularity
  - JSR 294 + Project Jigsaw
- Small language changes
  - Project Coin
- VM Support for dynamic languages
  - Da Vinci Machine project
- More new IO APIs
  - NIO.2
- Forward port Java SE6 u10 features
  - Java kernel (Improves download time), Quickstarter (Improves cold start-up time), New plug-in architecture, etc

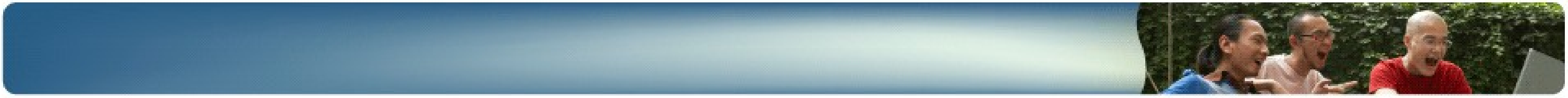


Features from Sun: Small

- SCTP (Stream Control Transmission Protocol)
- SDP (Socket Direct Proctocol)
  - Reliable, high-performance network streams over Infiniband connections on Solaris
- Unicode 5.1
- Swing updates
  - JXLayer (universal decorator for Swing components), DatePicker, possibly CSS styling

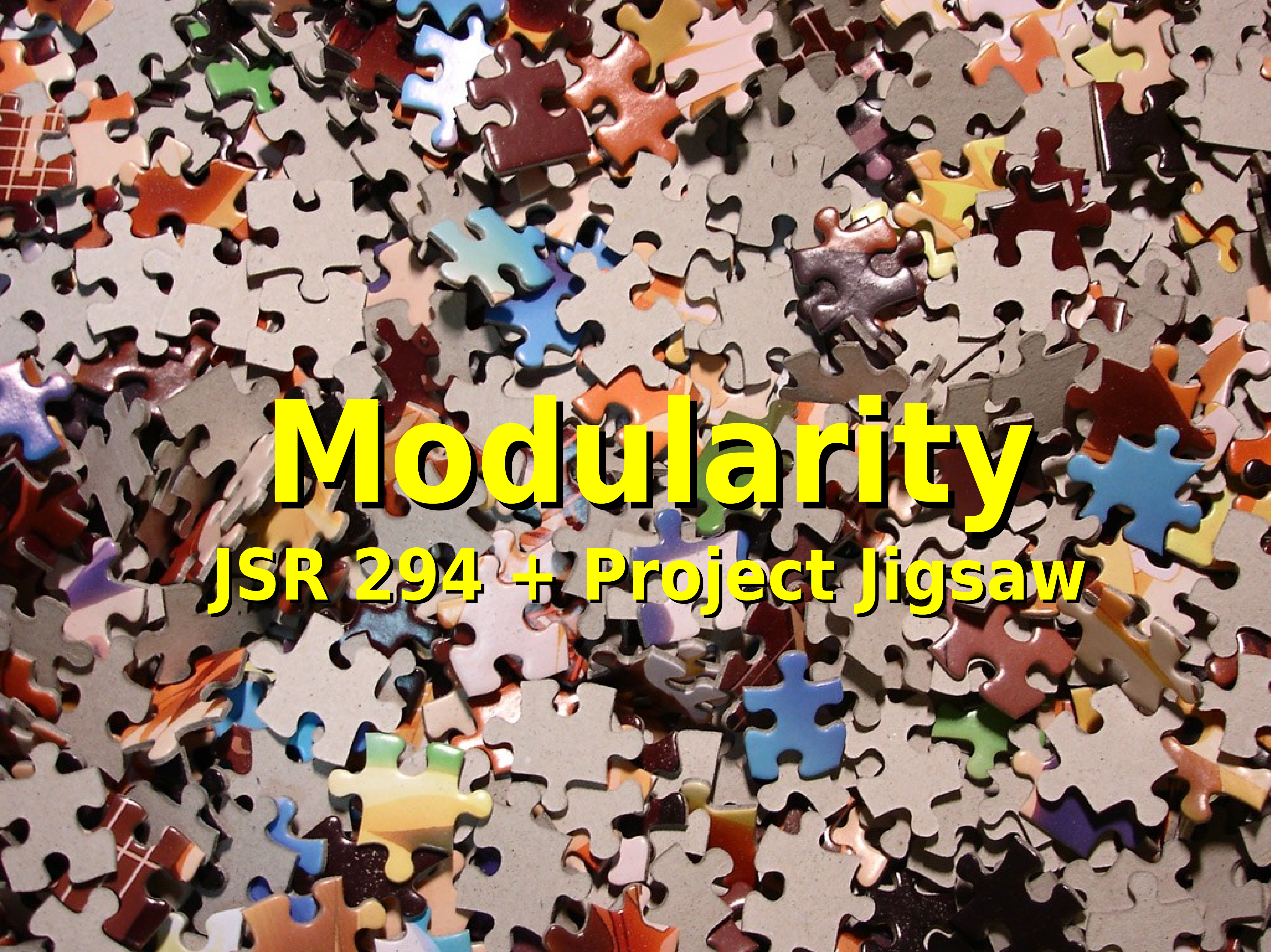
Features from Sun: Fast

- Garbage-First GC (G1)
  - Lower and predictable pause time
- Compressed 64-bit object pointers
  - A technique for compressing 64-bit pointers to fit into 32 bits
- Yet more HotSpot runtime compiler enhancements



Features from Others

- XRender pipeline for Java 2D
  - A new Java2D graphics pipeline based upon the X11 XRender extension
- Annotations on Java types (JSR 308)
  - Permit annotations on any occurrence of a type
- Concurrency and collections updates (JSR 166y)
  - Fork/join framework

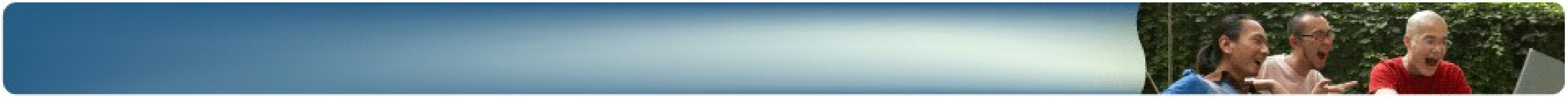


Modularity

JSR 294 + Project Jigsaw

Size matters

- JDK is big, really big
  - JDK 1.x – 7 top level packages, 200 classes
  - JDK 7 – too many top level packages, over 4000 classes
  - About 13MB today
- Historical baggage
  - Built as a monolithic software system – you cannot use a subset of it
  - Implementation code very interconnected

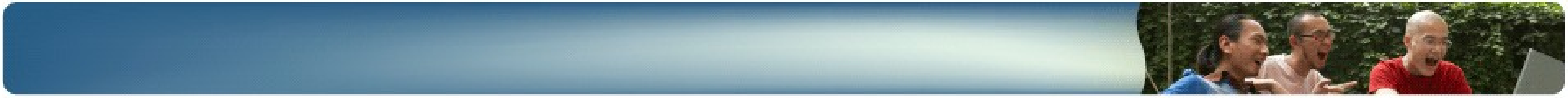


Problems of big, monolithic JDK

- Code modularity
  - JAR files don’t scale — “JAR/classpath hell”
    - No dependence
    - No versioning
    - No encapsulation of internal interfaces (cannot hide classes)
    - No well-defined relationship to native packaging systems
- Platform scalability
  - SE doesn’t scale down from the server/desktop
  - ME code doesn’t run on SE
- Performance
  - Download time, start-up time, memory footprint
    - Of the JRE itself, and of applications

Interim Solution – use JDK 6u10

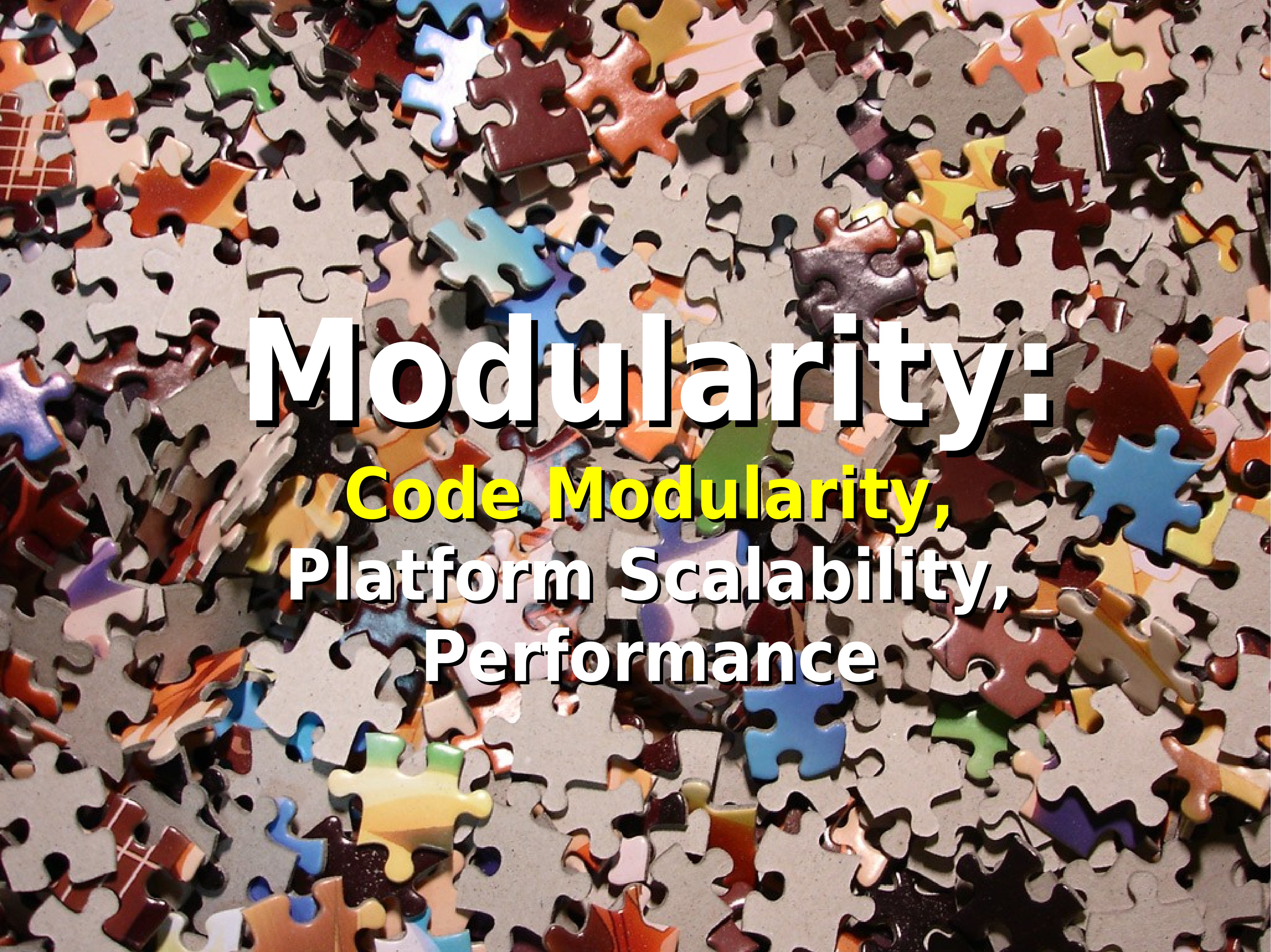
- Kernel installer, quick startup feature
- More like a painkiller, not the antidote



Solution: The Modular Java Platform

Modularize the platform — and applications too!

- Enables escape from “JAR/Classpath hell”
  - Eliminate the class path — once and for all!
  - Easily generate sensible rpm/deb/svr4/ips native packages
- Enables platform scalability
  - Well-specified SE subsets can fit into small devices
  - ME applications can run on SE (subsets)
- Enables significant performance improvements
  - Less work to start up
  - Incremental download of modules on demand
  - Pre-optimization of module content during install



Modularity:

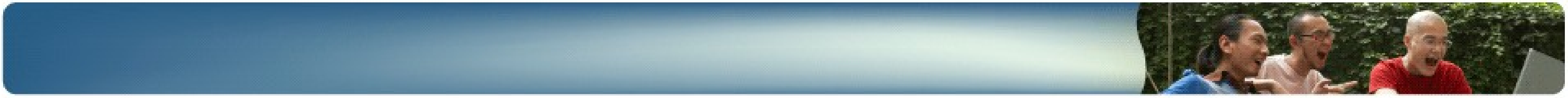
Code Modularity,
Platform Scalability,
Performance

Code Modularity Requirements

> Dependency
- Clear about what other code it depends on
- By explicitly listing its dependency on other code

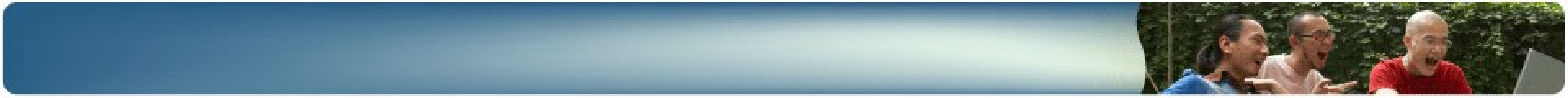
> Versioning
- Able to evolve without breaking clients
- By explicitly versioning itself and its dependencies

> Encapsulation
- Able to hide its internals from other modules
- By explicitly making those classes 'module-private'



JSR 294 and Project Jigsaw

- JSR 294
  - Language and VM changes to support module systems
- Jigsaw
  - A module system for JDK7
  - Module systems like Jigsaw & OSGi use language/VM features from 294
- OpenJDK Project Jigsaw
  - Hosts the reference implementation of JSR 294
  - Hosts the design and implementation of the Jigsaw module system



JSR 294 and Project Jigsaw (and OSGi)

> Why JDK 7 does not use OSGi as the module system of JDK 7?

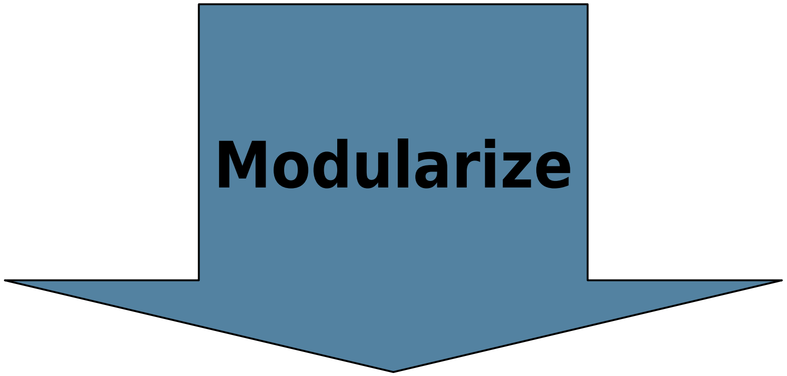
- The modularity for Java platform should be well integrated with the language and the VM
- OSGi (at least current version of it) is built on the top of current VM (pre-JDK7-JVM)
- Jigsaw has a focused goal of modulating JDK, thus has narrower goal than OSGi

> What is the relationship between JSR 294 and OSGi?

- JSR 294 defines modularity support in the language and VM
  - OSGi Alliance (Peter Kriens) is a member of JSR 294 expert group
- Future version of OSGi is highly likely built over JSR 294

Modularizing You Code

```
planetjdk/src/
                org/planetjdk/aggregator/Main.java
```



Modularize

```
planetjdk/src/
                org/planetjdk/aggregator/Main.java
                module-info.java
```

module-info.java

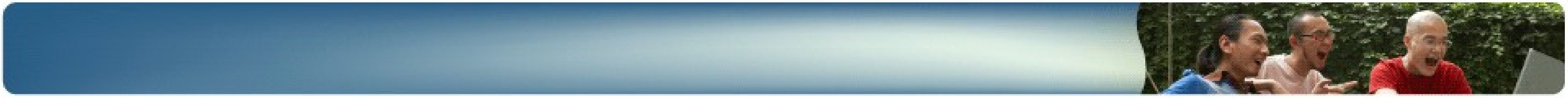
```
module org.planetjdk.aggregator {
    system jigsaw;
    requires module jdom;
    requires module tagsoup;
    requires module rome;
    requires module rome-fetcher;
    requires module joda-time;
    requires module java-xml;
    requires module java-base;
    class org.planetjdk.aggregator.Main;
}
```

Module name

Module system

Dependencies

Main class



Compiling Modules

```
javac -modulepath planetjdk/src:rssutils/rome
     org.planetjdk.aggregator/module-info.java
     org.planetjdk.aggregator/org/...
     com.sun.syndication/module-info.java
     com.sun.syndication/com/...
```

- Infers classes in org.planetjdk.aggregator/* belongs to module org.planetjdk.aggregator
  - From module-info.java

Running Modules

```
java -modulepath planetjdk/cls:rssutils/rome
   -m org.planetjdk.aggregator
```

> Launcher will look for org.planetjdk.aggregator from modulepath and executes it

- Assuming that module is a root module

“provides”

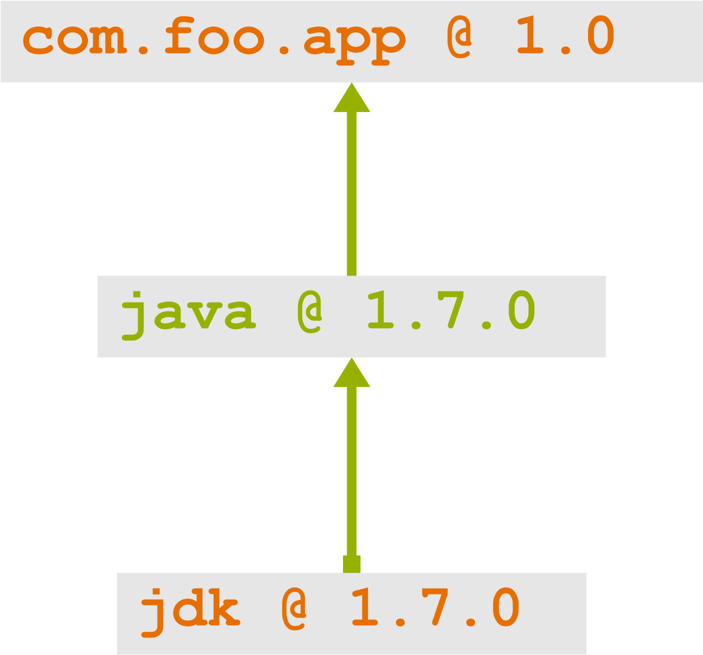
- In Jigsaw, a module can have aliases:

```
module jdk.core {
    system jigsaw;
    provides module java-base;
}
```

- This module satisfies `requires module java-base` in another module

Virtual module support with “provides”

EXPERIMENTAL

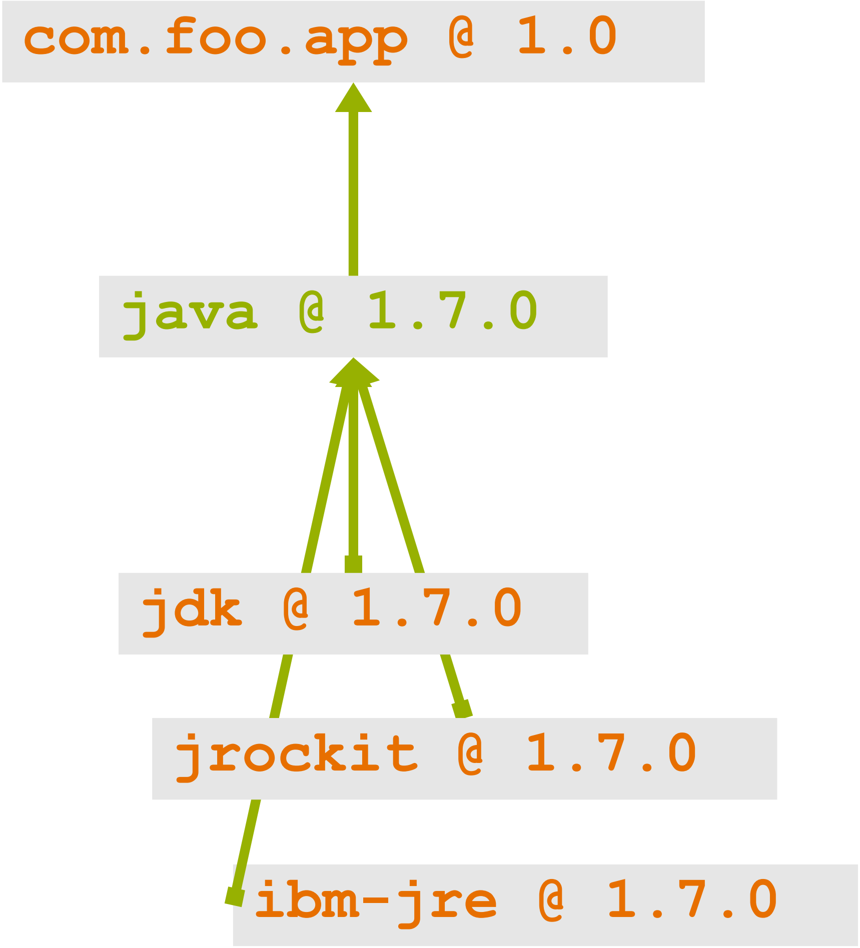


```
// app/module-info.java
module com.foo.app @ 1.0;
~~requires jdk @ 1.7.0;~~
requires java @ 1.7.0;
```

```
// jdk/module-info.java
module jdk @ 1.7.0;
provides java @ 1.7.0;
```

Virtual module support with “provides”

EXPERIMENTAL

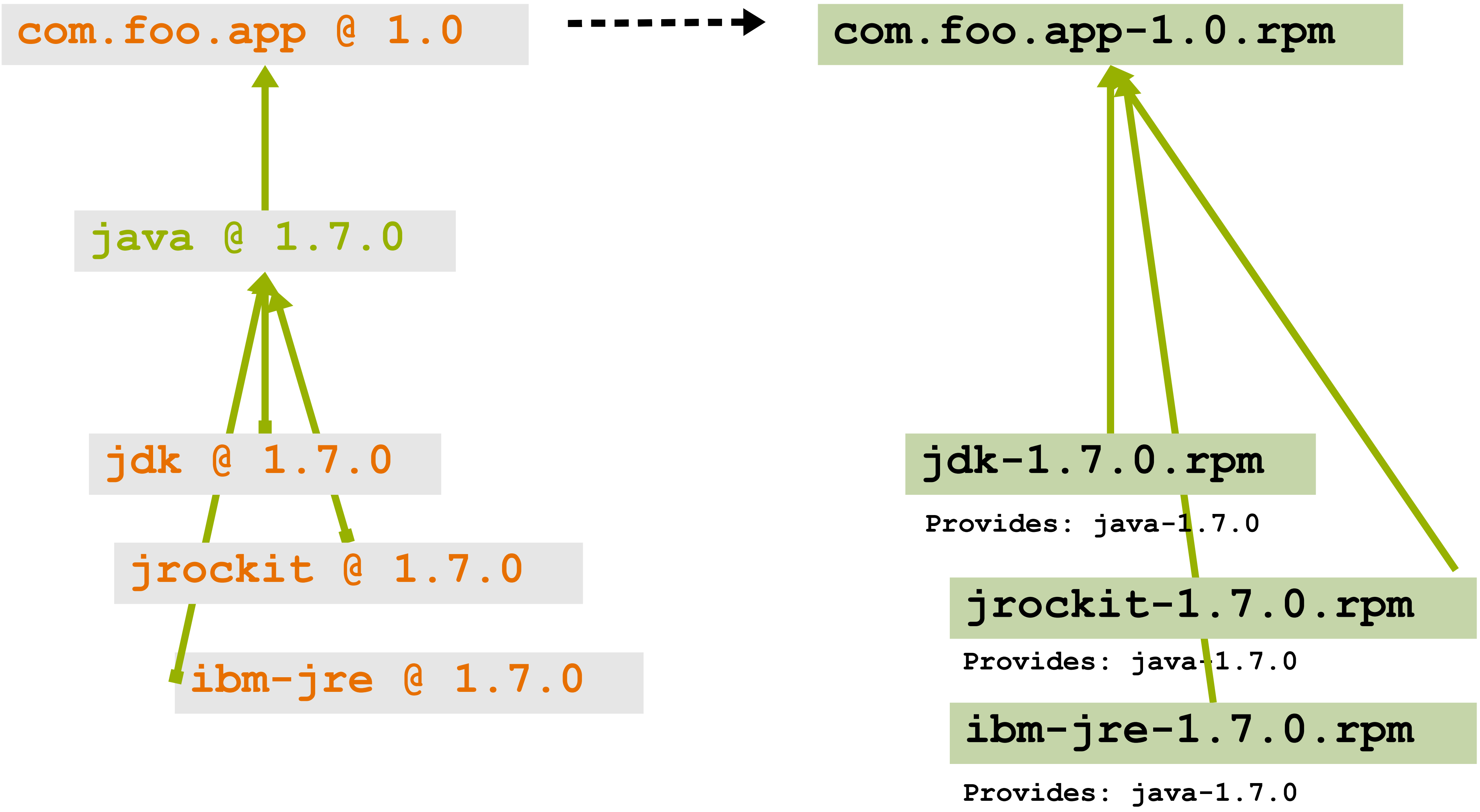


```
// app/module-info.java
module com.foo.app @ 1.0;
~~requires jdk @ 1.7.0;~~
requires java @ 1.7.0;
```

```
// jdk/module-info.java
module jdk @ 1.7.0;
provides java @ 1.7.0;
```

Virtual modules: Native packaging

EXPERIMENTAL



com.foo.app @ 1.0

java @ 1.7.0

jdk @ 1.7.0

jrockit @ 1.7.0

ibm-jre @ 1.7.0

com.foo.app-1.0.rpm

jdk-1.7.0.rpm

Provides: java-1.7.0

jrockit-1.7.0.rpm

Provides: java-1.7.0

ibm-jre-1.7.0.rpm

Provides: java-1.7.0

Module Private Accessibility

- Sharing types and members across packages but within a module

```
public class Foo {
  module String m() { ... }
}
```

- Class in the same module with Foo can access Foo.m
  - Meta data is stored in class files and is enforce by the VM

Module Versioning

```
module org.planetjdk.aggregator @ 1.0 {
  system jigsaw;
  requires module jdom @ 1.*;
  requires module tagsoup @ 1.2.*;
  requires module rome @ =1.0;
  requires module rome-fetcher @ =1.0;
  requires module joda-time @ [1.6,2.0);
  requires module java-xml @ 7.*;
  requires module java-base @ 7.*;
  class org.planetjdk.aggregator.Main;
}
```

Versioning in JSR 294

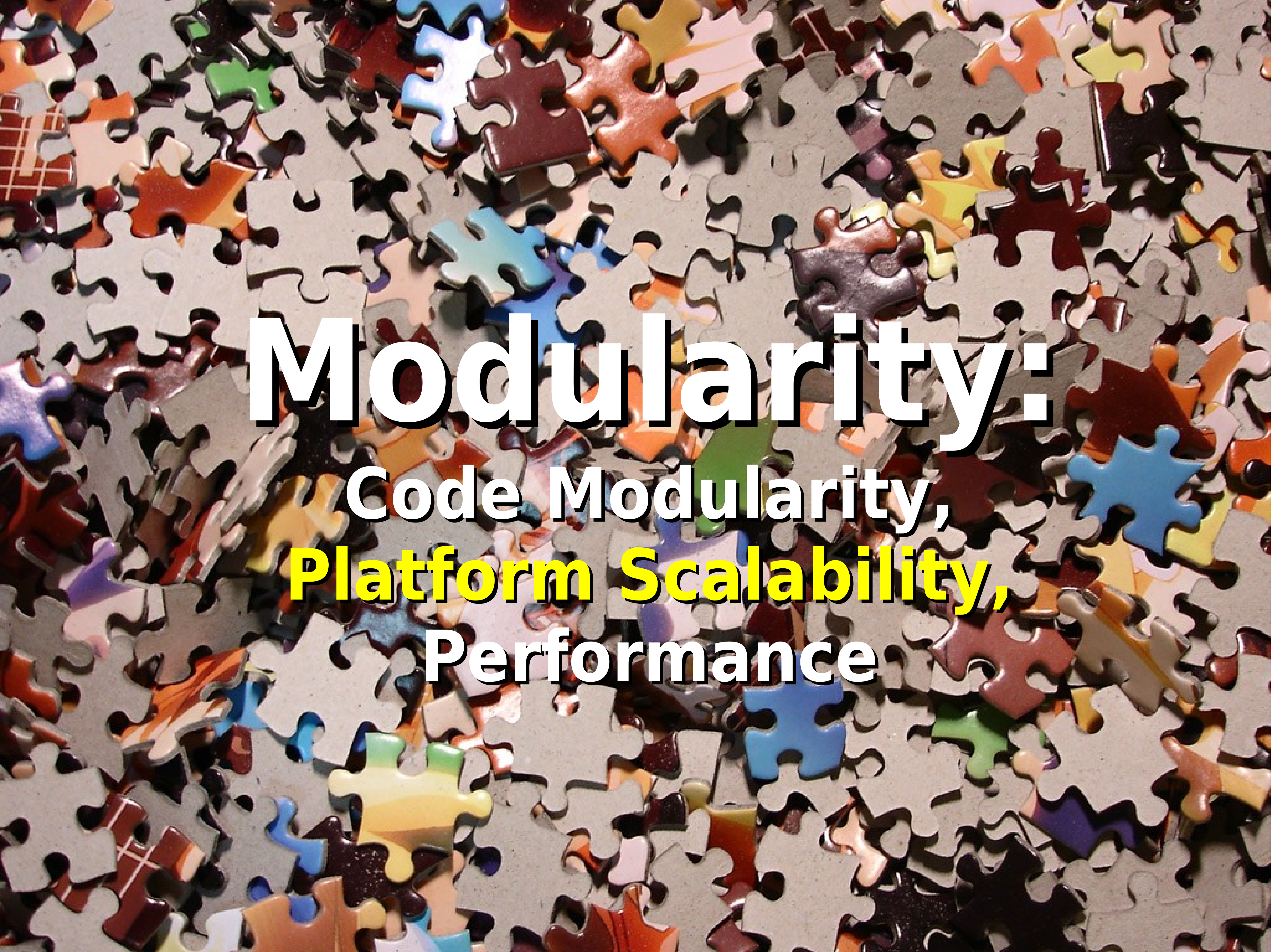
- Language + VM know that modules are versioned
- Version **structure**, **order**, and **ranges** are, however, implementation details of a particular module system (i.e., jigsaw module system has its versioning scheme)

Structure	1.2.3.4.5 I,II,III,IV,V..IX,X,L,C,D,M R11V16.4M3.2T200906031105		
Order	1.2.3.4 > 1.2.3 1.6:u13 < 1.7:b60 < 1.6:u14		
Range	[1.0, 2.0)	1.*	1.*\1.3
	1.0+	>=1.0	1.0

Native Packaging

- > Build native packaging from module information
- > New tool – jpkg

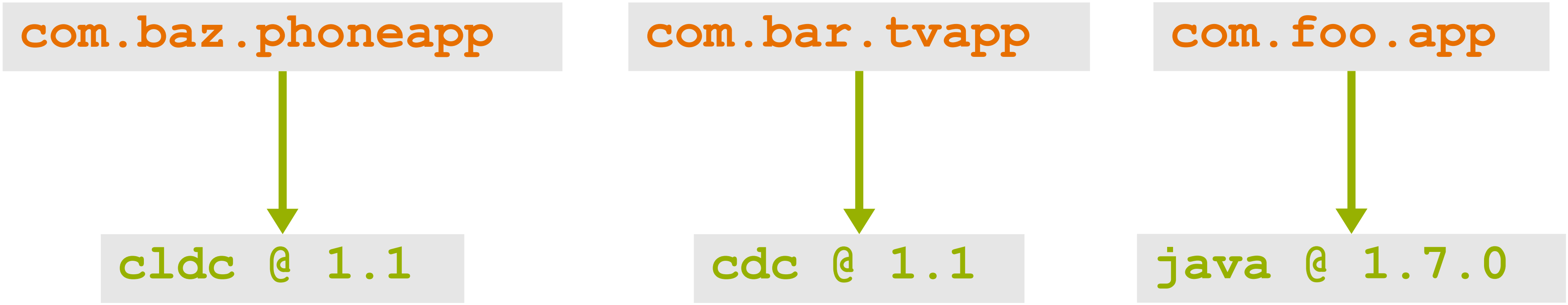
```
javac -modulepath planetjdk/src:rssutils/rome
   -d build/modules org.planetjdk.aggregator/module-info.java
   org.planetjdk.aggregator/org/...

jpkg -L planetjdk/cls:rssutils/rome -m build/modules
   deb -c aggregator

sudo dpkg -i org.planetjdk.aggregator_1.0_all.deb
/usr/bin/aggregator
```



Modularity:

Code Modularity,
Platform Scalability,
Performance

Three Java platforms



com.baz.phoneapp

cldc @ 1.1

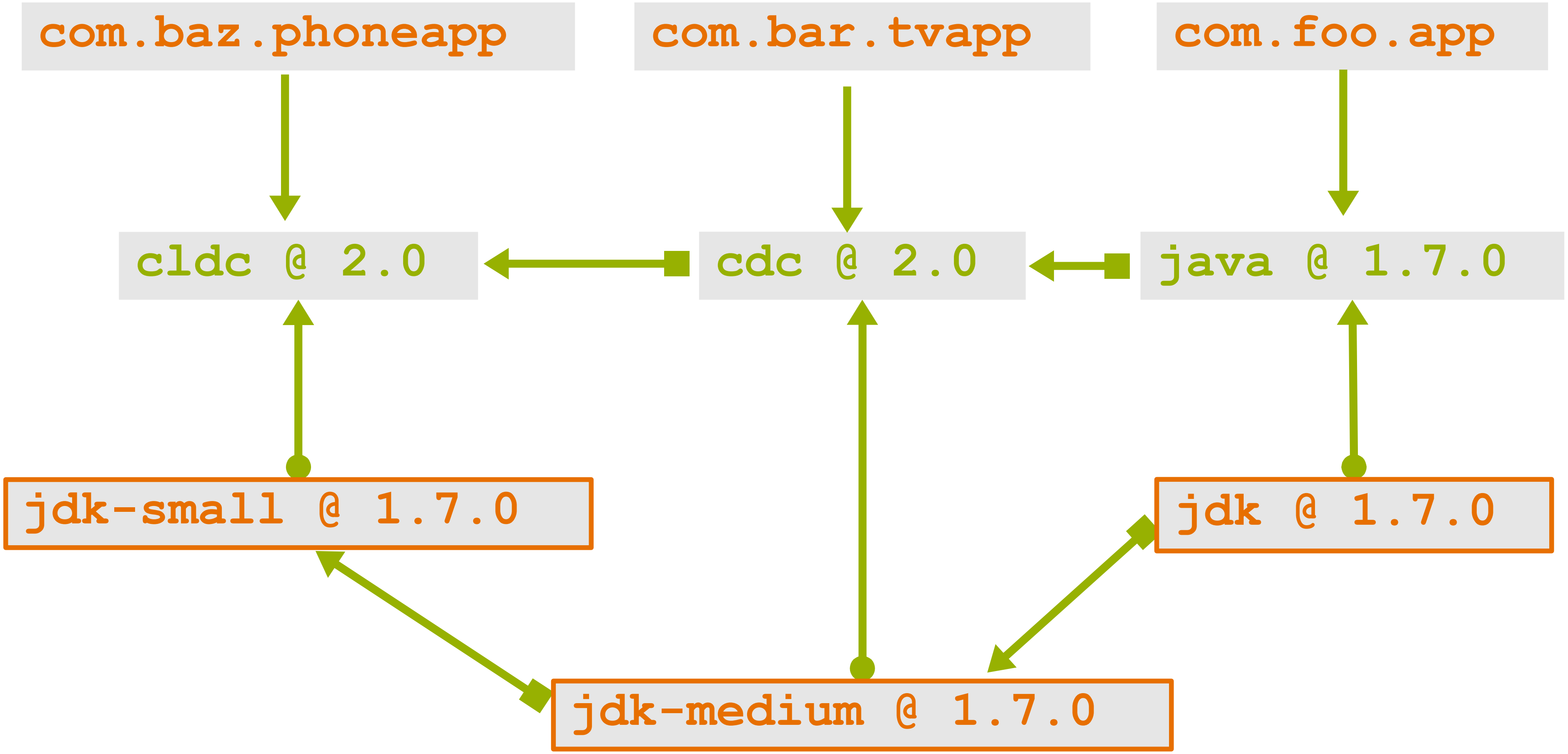
com.bar.tvapp

cdc @ 1.1

com.foo.app

java @ 1.7.0

One Java platform, one code base!



com.baz.phoneapp

com.bar.tvapp

com.foo.app

cldc @ 2.0

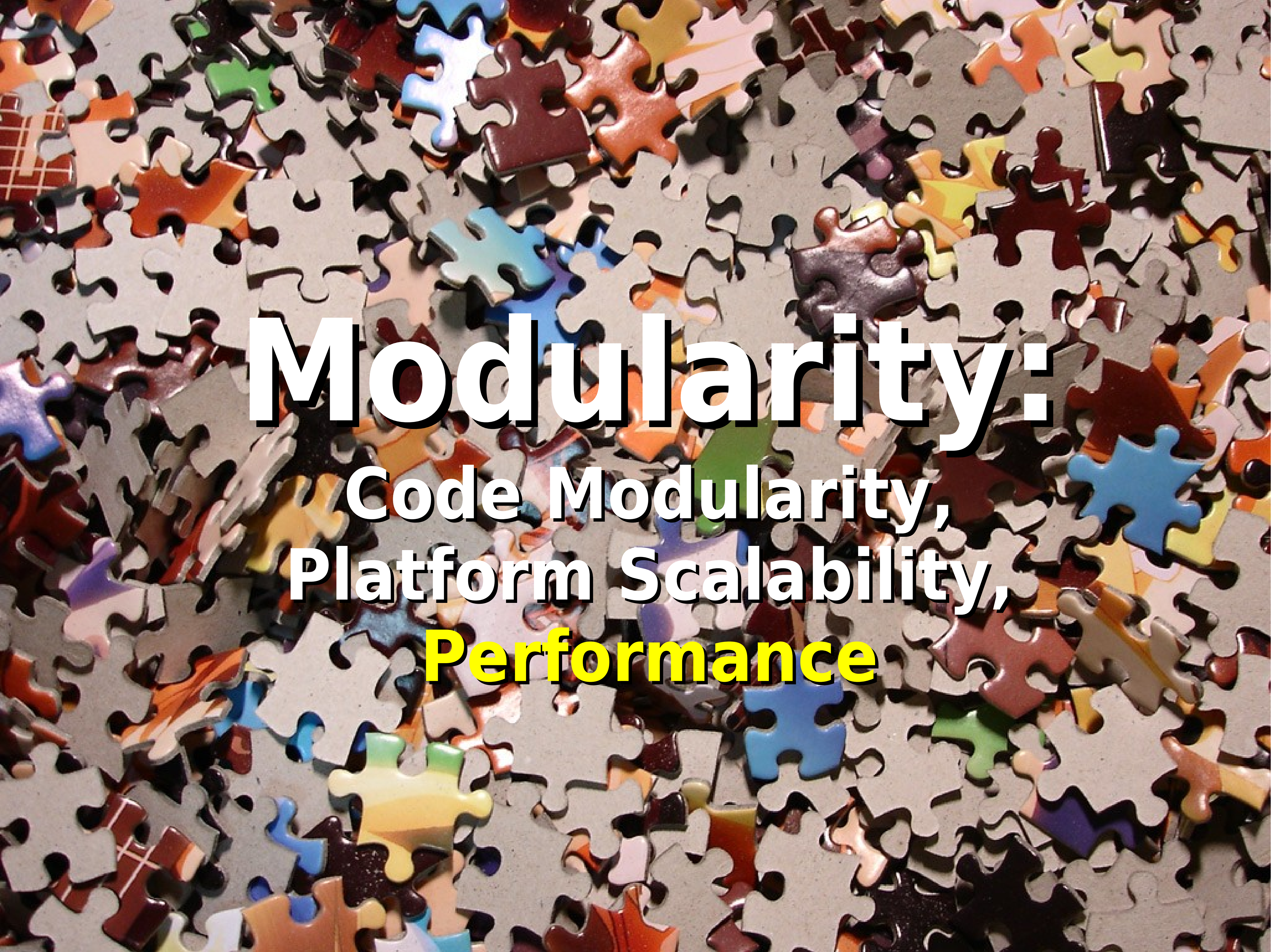
cdc @ 2.0

java @ 1.7.0

jdk-small @ 1.7.0

jdk @ 1.7.0

jdk-medium @ 1.7.0



Modularity:

Code Modularity, Platform Scalability, Performance

Performance

- Download time
  - No need to download an entire application all at once
    - Only need its initial modules to get started
  - No need to download the entire JRE all at once
    - Only need the modules initially required by the application
- Startup time
  - Module installation is an opportunity for pre-optimization
    - Pre-initialize static data, compile to native code, *etc.*
- Memory footprint
  - Modularizing the JDK will massively reduce coupling
    - Fewer classes to load, less state to initialize

Small (Language) Changes

Project Coin

Better Integer Literals

> Binary literals

```
int mask = 0b1010;
long num = 0b111110100000000L;
```

> With underscores

```
int bigMask = 0b1010_0101;
long big = 9_223_783_036_967_937L;
```

> Unsigned literals

```
byte b = 0xffu;
```

Better Type Inference

```
// Before JDK 7
Map<String, Integer> foo =
    new HashMap<String, Integer>();

// With JDK 7
Map<String, Integer> foo =
    new HashMap<>();
```

Strings in Switch

> Strings are constants too

```
String s = ...;
switch (s) {
  case "foo":
    return 1;

  case "bar":
    Return 2;

  default:
    return 0;
}
```

Resource Management: pre-JDK7

> Manually closing resources is tricky and tedious

```
public void copy(String src, String dest) throws IOException {
    InputStream in = new FileInputStream(src);
    try {
        OutputStream out = new FileOutputStream(dest);
        try {
            byte[] buf = new byte[8 * 1024];
            int n;
            while ((n = in.read(buf)) >= 0)
                out.write(buf, 0, n);
        } finally {
            out.close();
        }
    } finally {
        in.close();
}}
```

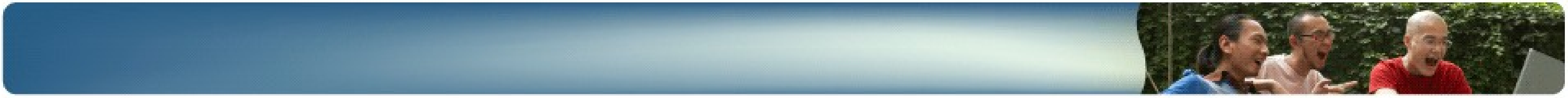
Automatic Resource Management

```
static void copy(String src, String dest) throws IOException {
    try (InputStream  in  = new FileInputStream(src);
         OutputStream out = new FileOutputStream(dest)) {
        byte[] buf = new byte[8192];
        int n;
        while ((n = in.read(buf)) >= 0)
            out.write(buf, 0, n);
    }
   //in and out closes
}
```

Closable Interface

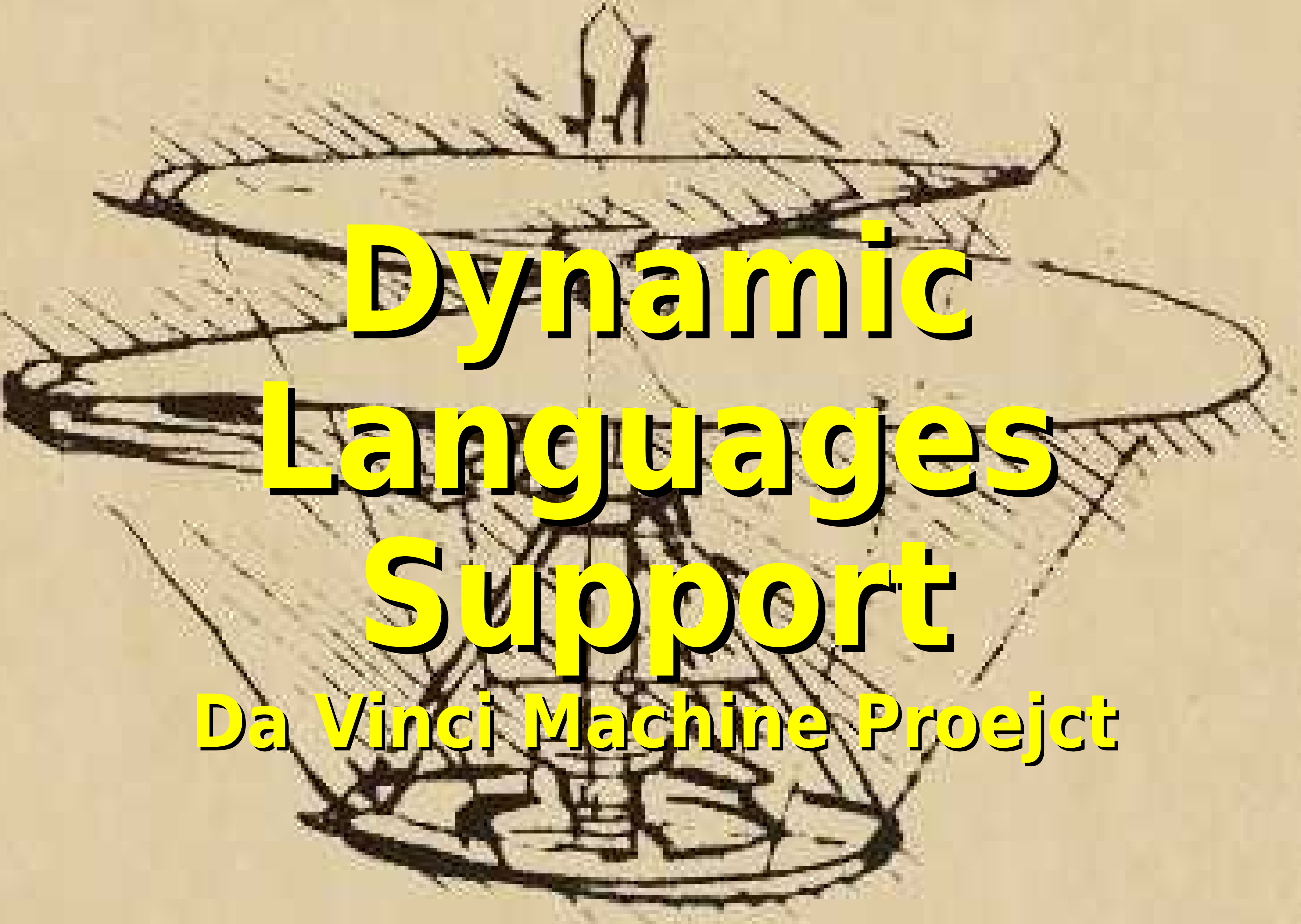
- Implemented by all auto close resources

```
package java.lang.auto;
/**
 * A resource that must be closed
 * when it is no longer needed.
 */
public interface AutoCloseable {
   void close() throws Exception;
}

package java.io;
public interface Closeable extends AutoCloseable {
   void close() throws IOException;
}
```



Index Syntax for Lists and Maps

```
List<String> list =
   Arrays.asList(new String[] {"a", "b", "c"});
String firstElement = list[0];      // JDK 7

Map<Integer, String> map = new HashMap<>();
map[Integer.valueOf(1)] = "One";    // JDK 7
```



Dynamic Languages Support

Da Vinci Machine Proejct

JVM Specification, First Edition (1997)

"The Java Virtual Machine knows nothing about the Java programming language, only of a particular binary format, the class file format."

"A class file contains Java Virtual Machine instructions (or bytecodes) and a symbol table, as well as other ancillary information."

"Any language with functionality that can be expressed in terms of a valid class file can be hosted by the Java virtual machine."

"Attracted by a generally available, machine-independent platform, implementors of other languages are turning to the Java Virtual Machine as a delivery vehicle for their languages."

"In the future, we will consider bounded extensions to the Java Virtual Machine to provide better support for other languages."

Languages on the Java Virtual Machine

Tea iScript

Zigzag JESS Jickle

Modula-2 CAL Lisp JavaScript

Correlate Nice

Simkin Drools Basic JudoScript

Groovy Eiffel Luck

Icon v-language

Prolog Mini Pascal

Tcl PLAN Hojo Scala

Rexx JavaFX Script

Funnel

Tiger Anvil Yassl FScript

Oberon

E Smalltalk

Logo JHCR JRuby

Tiger

Scheme

Ada G Clojure Sather

Phobos

Dawn

Processing WebL Sleep

TermWare

LLP Pnuts

Bex Script

BeanShell

Forth PHP

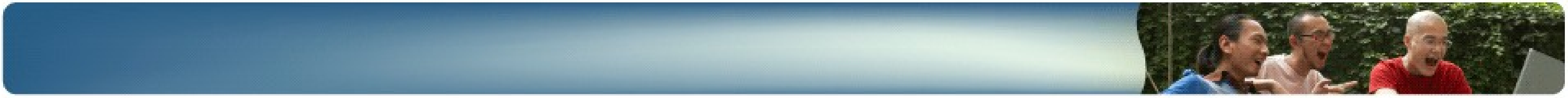
C#

SALSA ObjectScript

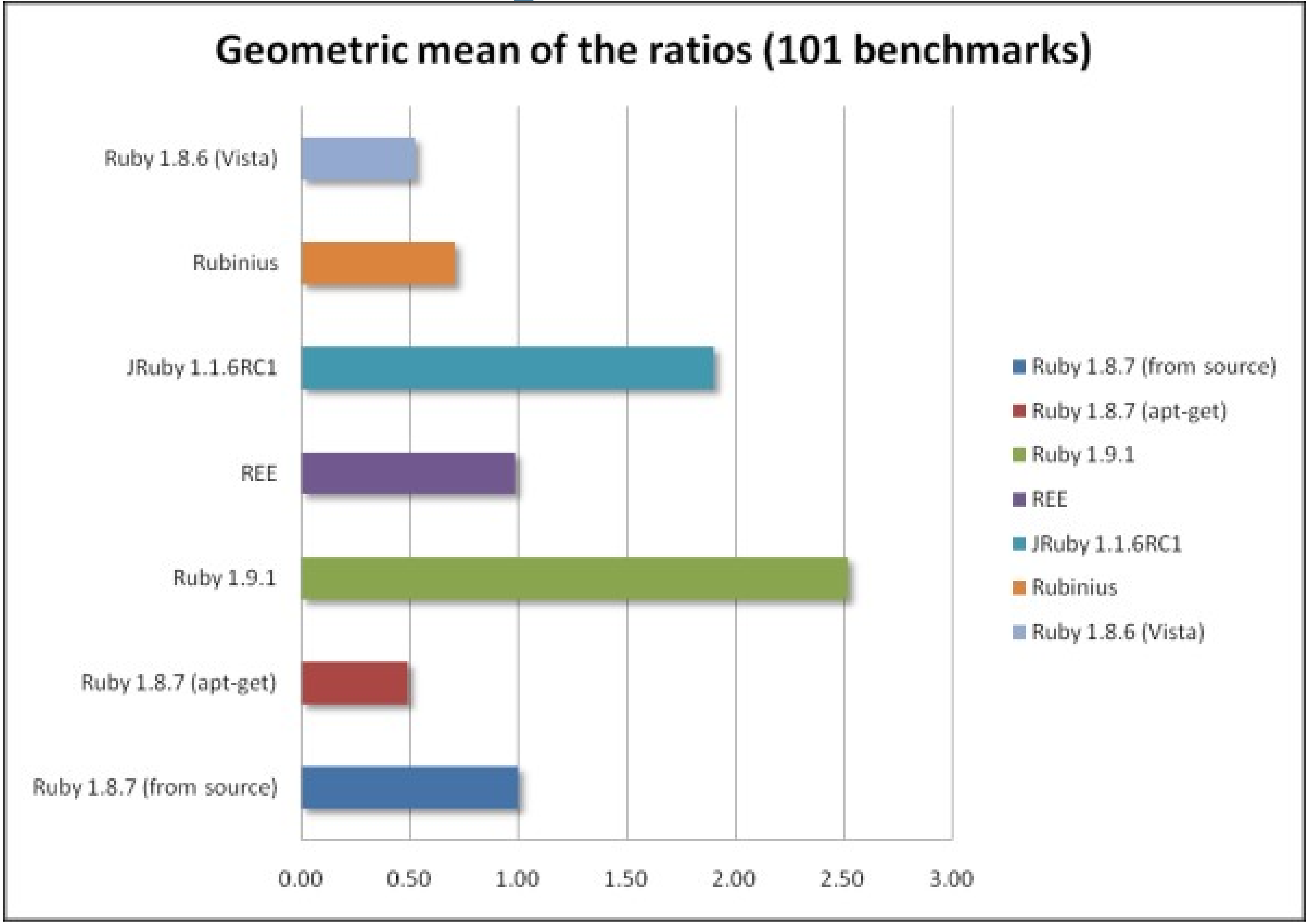
Yoix

Piccola

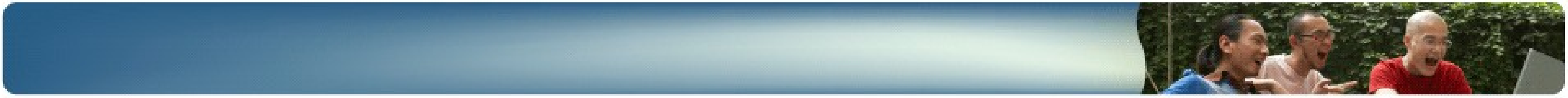
Jython



The Great Ruby Shootout 2008

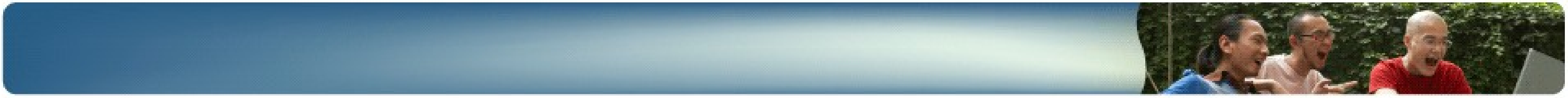


http://antoniocangiano.com/2008/12/09/the-great-ruby-shootout-december-2008/



Pain Points for Dynamic Languages

- The current JVM is designed for static typed language
- Biggest mismatch between current JVM and dynamic languages is method selection and invocation
  - Calling a method is cheap; selecting the right method is expensive
  - Static languages do most of their method selection at compile-time
  - Dynamic languages do almost none at compile-time (obviously) instead during runtime
- What is the one change in the JVM that would make life better for dynamic languages?
  - Flexible method calls!



JSR 292 (Support for Dynamically Typed Languages in the Java Virtual Machine)

> Introduced a new bytecode called “*invokedynamic*” and new linkage mechanism called Method Handle
  - Dynamic language compiler designers no longer have to do backdoor work

> Enables even higher performance of the applications written in dynamic language

> Da Vinci Machine Project, an OpenJDK community provides the implementation

NIO.2

java.io.File problems

> Many methods return just booleans
  - Don't know why something failed

> No copy / move support

> No symbolic link support

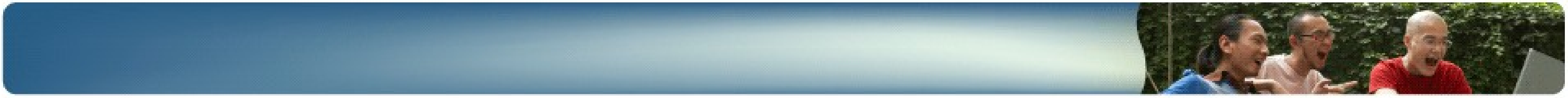
> No change notification support

> Limited support for file attributes

> Not extensible

NIO.2 Features

- Methods throw exceptions!
- Path operations (relative etc.)
- File operations (copy, check access, symlink)
- Directories (iterate)
- Recursive operations
- File change notification
- File attributes (NFSv4 ACL, Posix)
- Provider interface



NIO.2 Classes

> FileRef
  - Reference to a file

> Path
  - Locates a file using a system dependent path

> FileSystem
  - Provides interface to file system
  - Factory for objects to access files and other objects in the file system.
  - Default file system for local/platform file system

> FileStore
  - Underlying storage system

Using Path Class

```
import java.nio.file.*;

// FileSystems -> FileSystem -> Path
FileSystem fileSystem = FileSystems.getDefault();
Path homeDir = fileSystem.getPath("/Users/amiller”);

// Shortcut with Paths helper class
Path homeDir = Paths.get("/Users/amiller”);

// Resolve one path in terms of another
Path relativeTemp = Paths.get(“temp”);
Path absoluteTemp = relativeTemp.resolve(homeDir);

// Get relative path from a base
Path absoluteProfile = Paths.get("/Users/amiller/.profile”);
Path relativeProfile = absoluteProfile.relativize(homeDir);
assert relativeProfile.isRelative();
assert relativeProfile.getNameCount() == 1;
```

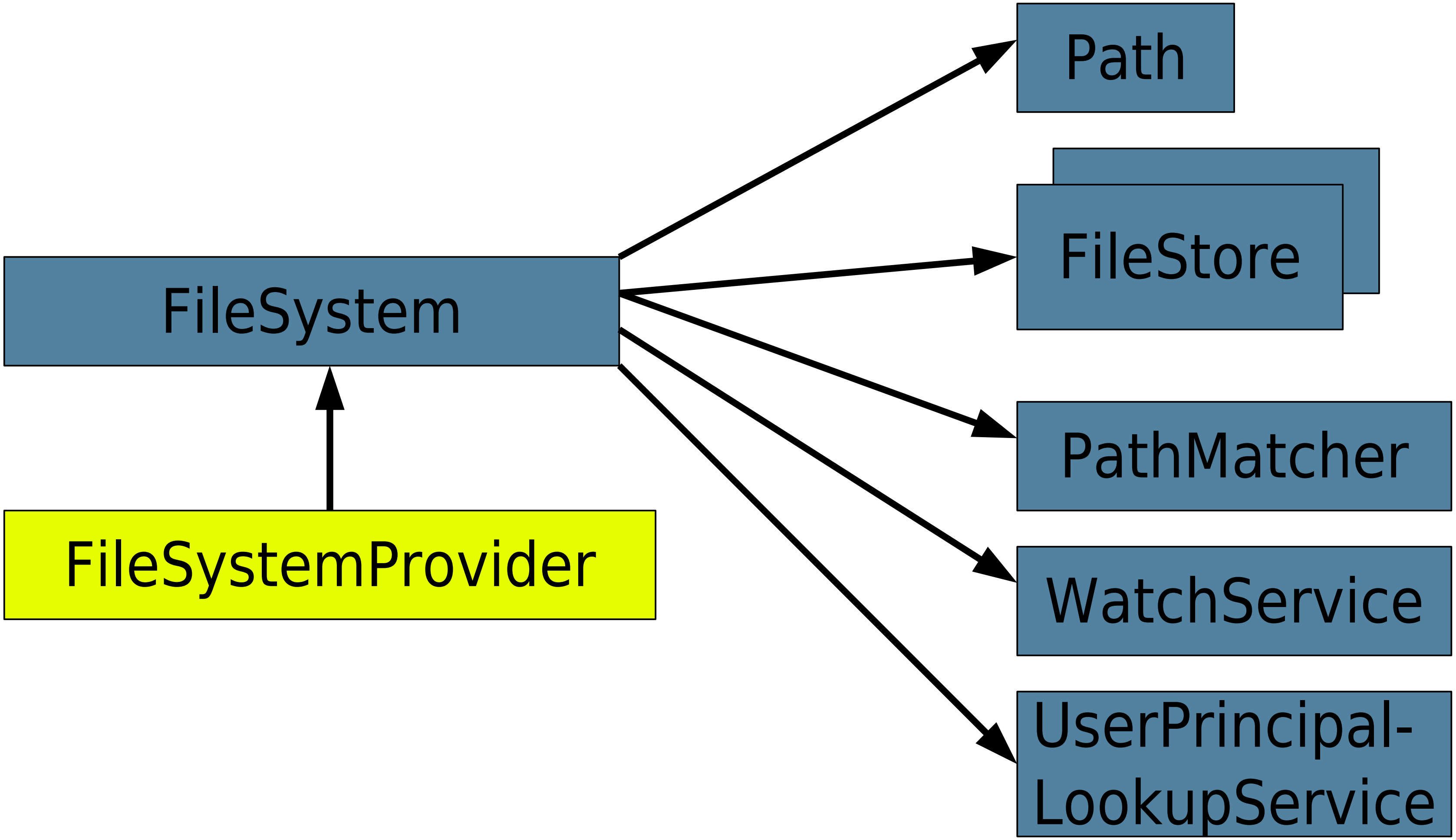
Appending to a file

```
import java.io.*;
import java.nio.file.*;
import static java.nio.file.StandardOpenOption.*;

Path journal = Paths.get("/some/path/journal.txt");

OutputStream stream =
    journal.newOutputStream(CREATE, APPEND);

try {
    writeEntry(stream); // normal stuff
} finally {
    stream.close();
}
```

Copying and Moving

```
import java.nio.file.*;

Path home = Paths.get("/Users/amiller");
Path secrets = home.resolve("secrets.txt");

// Steal secrets
secrets.copyTo(home.resolve("stolenSecrets.txt"));

// Hide secrets
secrets.moveTo(Paths.get("/Users/dvader/secrets.txt"));
```

FileSystem class as a factory



FileSystem

FileSystemProvider

Path

FileStore

PathMatcher

WatchService

UserPrincipal-LookupService

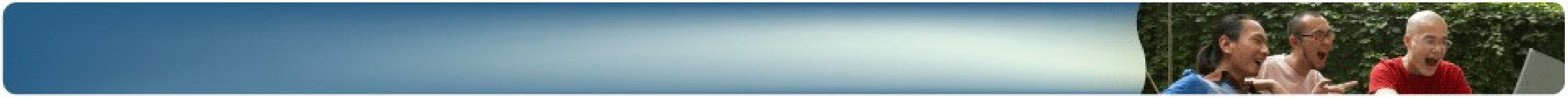
File Change Notification

- > Improve performance of applications that are forced to poll the file system today
- > *WatchService*
  - Watch registered objects for events and changes
  - Makes use of native event facility where available
  - Supports concurrent processing of events
- > *Path* implements *Watchable*
  - Register directory to get events when entries are created, deleted, or modified

Demo:

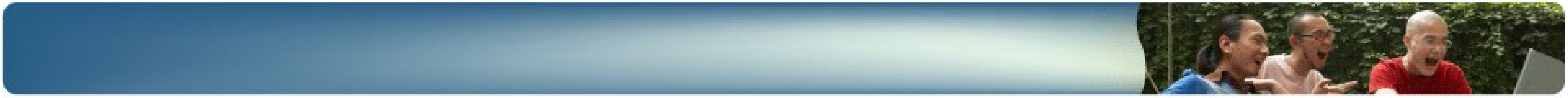
Building & Running NIO.2 Sample Apps from JDK 7 Build

Annotations on Java Types (JSR 308)



What are Type Annotations?

- The current (pre-JDK7) annotation syntax is useful but limited
  - Allowed only on class/method/field/variable declarations
- The Type Annotations syntax permits annotations to be written in more places
  - List<@NonNull Object>
- Programmers can use type annotations to write more informative types, and then tools such as type-checkers can detect and prevent more errors.

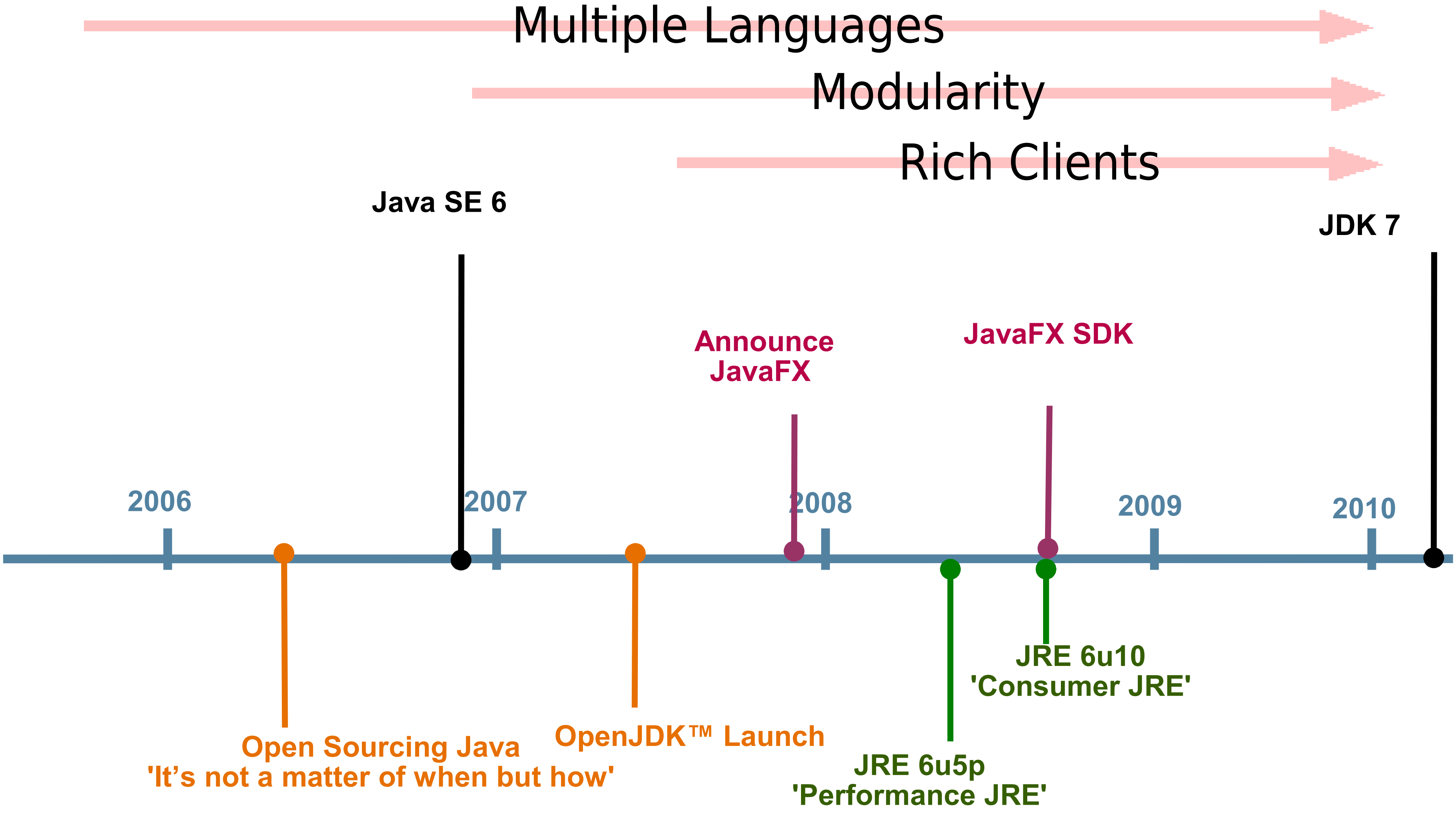


Examples of Type Annotations

- // generic type arguments:
  - Map<@NonNull String, @NonEmpty List<@Readonly Document>> files;
- // arrays:
  - Document[@Readonly] docs1;
- // type tests:
  - boolean isNonNull = myString instanceof @NonNull String;

Roadmap,
What will not be
in JDK 7

Java Platform Roadmap at a Glance



Multiple Languages

Modularity

Rich Clients

Java SE 6

JDK 7

Announce JavaFX

JavaFX SDK

Open Sourcing Java
'It’s not a matter of when but how'

OpenJDK™ Launch

JRE 6u5p
'Performance JRE'

JRE 6u10
'Consumer JRE'

What Will Not Be in JDK7

http://openjdk.java.net/projects/jdk7/features/

> Small Language changes that were proposed
  - Improved exception handling (safe rethrow, multi-catch)
  - Elvis and other null-safe operators
  - Large arrays

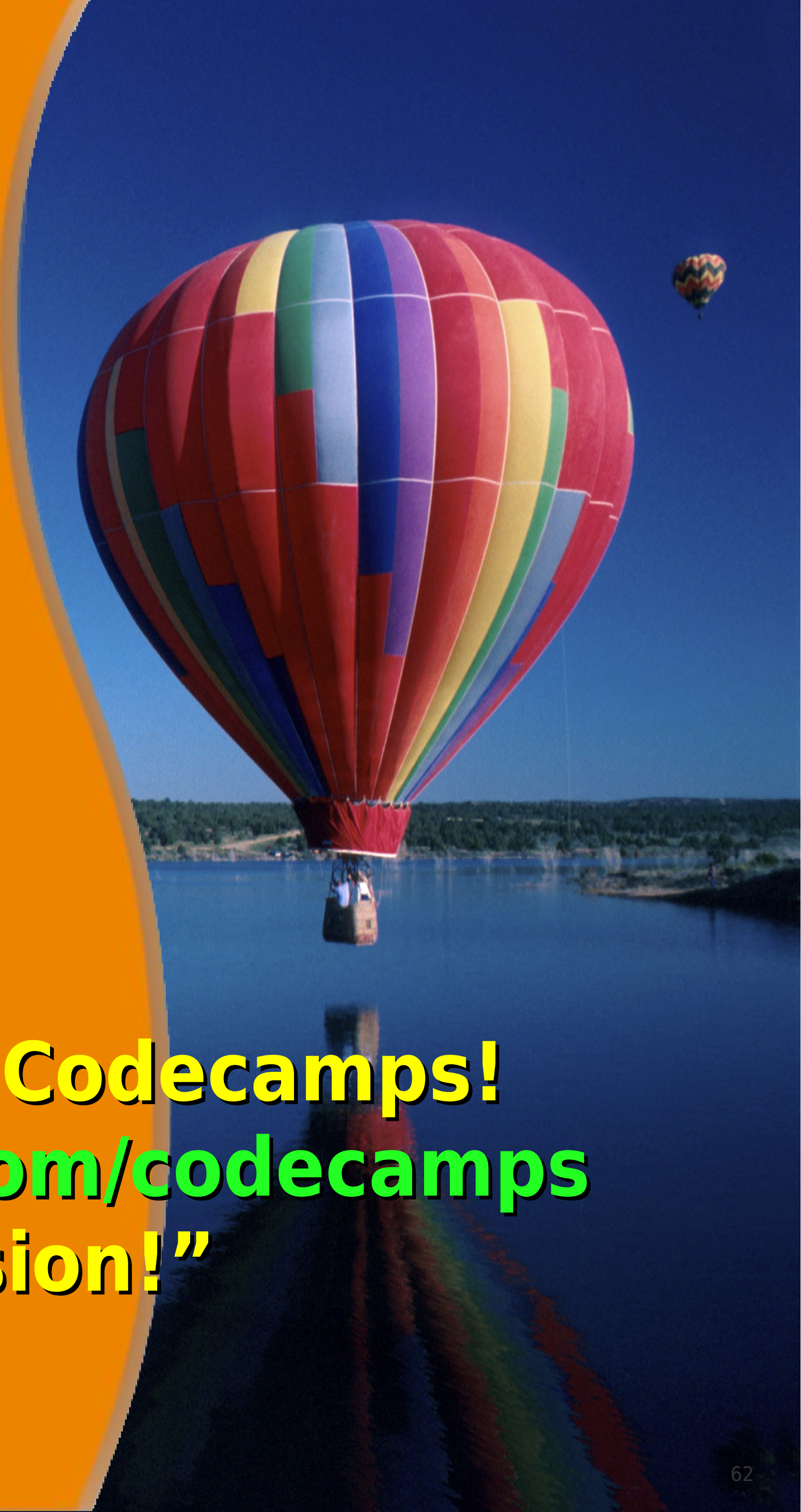
> Other language features
  - First class properties
  - Operator overloading

> JSR-295 Beans binding

> JSR-277 Java Module System

> JSR-296 Swing Application Framework

Thank you!



Check JavaPassion.com Codecamps!

http://www.javapassion.com/codecamps

“Learn with Passion!”