Apache Harmony's Approach to Implementing Java™ Platform, Standard Edition

Tim Ellison
Geir Magnusson Jr.

Apache Harmony Project

www.incubator.apache.org/harmony

TS-3752

Goals of the Talk

Within the Next 45 Minutes You Will…

Learn how Apache is building a compliant independent implementation of Java SE

Be inspired to participate in the Harmony community

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Apache Harmony

In the Beginning

- Accepted by the Apache Software Foundation (ASF) in May 2005
  - May 18th, in fact—Happy Birthday, Harmony!
- Project started in the Apache Incubator
  - Standard start for new projects at the ASF
  - A place for growing the community and the code

Apache Harmony

In the Beginning…

Three main project goals:

1. Compatible, independent implementation of Java SE 5.0
   - Available under the Apache License
2. Community-developed, modular architecture
   - Allow independent implementations to share runtime components
   - Allow independent innovation in runtime components
3. Protect IP rights of ecosystem

Apache Harmony

In the Beginning…

Motivations :

1. Create single open collaborative community
2. Enable continued adoption of Java via portable code under a liberal license
3. Provide an acceptably-licensed Java platform for Linux and other FOSS communities
4. Ensure Java can be a preferred platform for the FOSS-oriented developing economies

Apache Harmony

… and Then There Was Light! (History, 2005)

Broad community discussion

- Technical issues
  - Defined modules in Java and C code
  - Decided to maximize use of Java in class library code
  - VM: Java or C or C++ or mix or?
- Legal and IP issues
  - License issues with existing open source efforts
  - How to handle contributor exposure to Sun's class library
- Project governance issues
  - Acceptable prior access for contributors
  - Acceptable provenance for repurposed code contributions

Apache Harmony

… and Then There Was More Light! (History, 2005)

- Resolutions:
  - Prototype code for native modules, and agreement to split class library into logical modules
  - Reluctant recognition that reconciling GPLv2 and ALv2 was beyond project scope
  - A set of guidelines and eligibility criteria for contributors (people) and contributions (code)
- And then came the code!
  - ‘JC’ VM by Archie Cobbs (September 2005)
  - ‘bootstrap’ VM by Dan Lydick (September 2005)

Apache Harmony

… and Then There Was Code! (History 2005–Now)

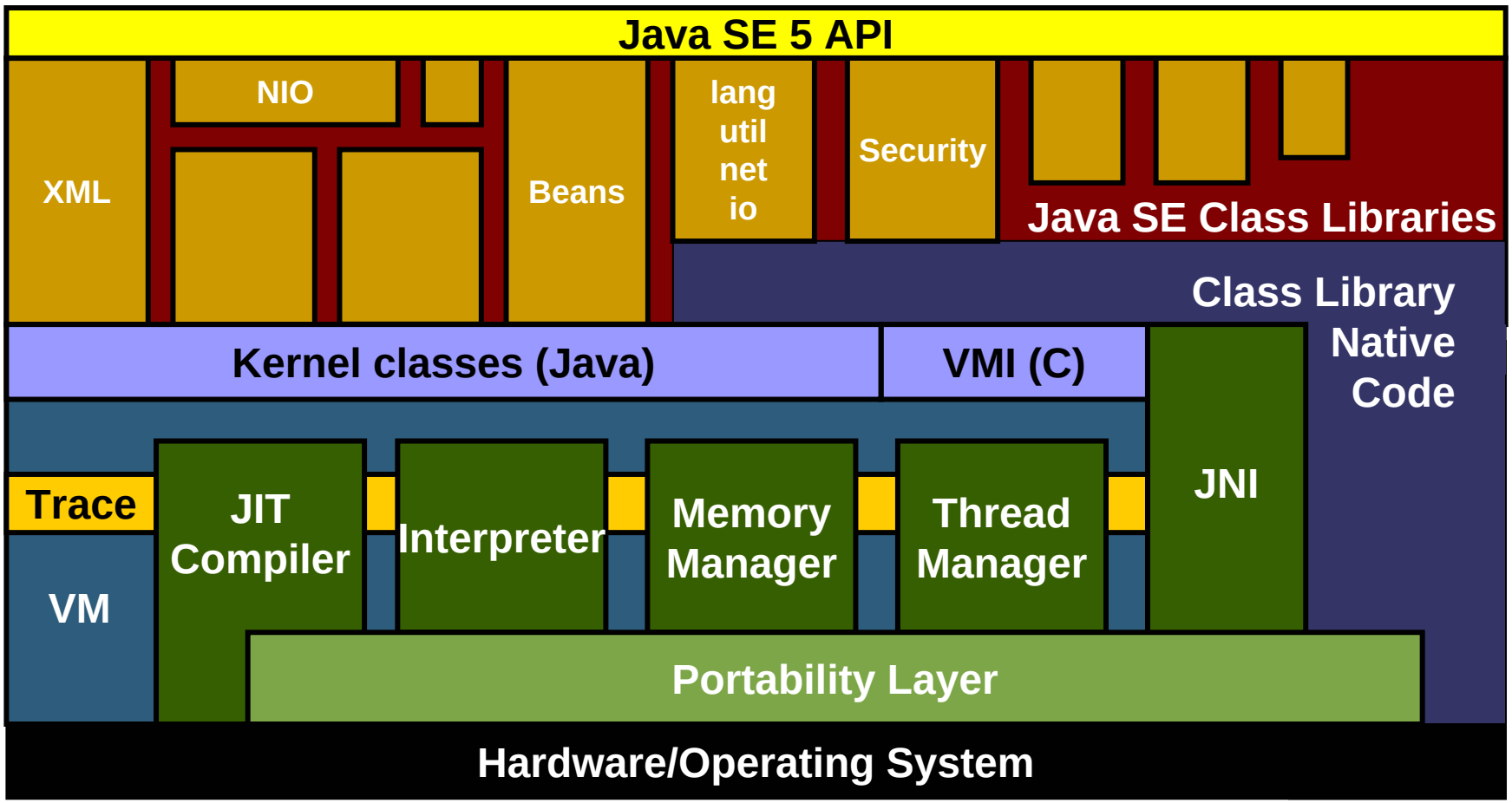
- Major class library code contributions
  - Core set of Java SE classes, Classlib/VM interface, test cases, eclipse plugin, et al (IBM Corp)
  - JSE security and cryptography, Beans, regex, math, RMI, et al (Intel Corp)
  - RMI and math (Cordoba Institute of Technology)
- Production-grade evaluation JVM
  - Available from IBM Developerworks site
    - Evaluation license (not FOSS)
- Another JVM of our own
  - Java Virtual Machine with JIT, Interpreter, GC etc (Intel Corp)

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Apache Harmony—Architecture

The 30,000 ft. View



Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Harmony's Class Library/VM Interface

Compatible VMs Are Required to Implement…

- VM-specific 'kernel' classes
  - 23 publicly defined Java SE types that the VM typically knows intimately, plus one helper
  - Harmony provides templates for many of these
- Standard Java Native Interface
  - To create objects etc. from class library natives
- Harmony defined VM Interface functions
  - 10 new C functions…

Harmony's Kernel Class List

VM-Specific Classes

```
java.lang.Object                    java.security.AccessControlContext
java.lang.Class                     java.security.AccessController
java.lang.ClassLoader               java.lang.reflect.AccessibleObject
java.lang.Compiler                  java.lang.reflect.Array
java.lang.Package                   java.lang.reflect.Constructor
java.lang.Runtime                   java.lang.reflect.Field
java.lang.StackTraceElement         java.lang.reflect.Method
java.lang.System                    java.lang.ref.PhantomReference
java.lang.Thread                    java.lang.ref.Reference
java.lang.ThreadGroup               java.lang.ref.WeakReference
java.lang.Throwable                 java.lang.ref.SoftReference
                                    org.apache.harmony.kernel.VM
```

- VM implementers provide concrete implementations
  - Either written from scratch, or derived from Harmony's stub implementations
  - Expect this set to grow modestly with new Java SE 5 types

Harmony's VMI Functions

Additional C-interface to the VM

- Access to structures and interfaces shared by the VM and class library
- The VMI provides:
  - Access to the operating system abstraction library (port library)
  - Access to per-VM storage functions (VMLS) which allows multiple VMs to exist in a single address space
  - Ability to get/set/iterate system properties
  - Major.minor version information to detect incompatible VMI shape changes
- The VMI does not :
  - Require any enhanced VM/class library linkage
  - Prescribe object layout, garbage collection, synchronization, and so on

Harmony's VMI Functions

Compatible VM's Are Required to Implement

```
/* Version check */
vmiError (JNICALL* CheckVersion) (VMInterface* vmi, vmiVersion* version);


/* Obtain VM structures & interfaces */
JavaVM*                (JNICALL* GetJavaVM)          (VMInterface* vmi);
JavaVMInitArgs*        (JNICALL* GetInitArgs)        (VMInterface* vmi);
HyPortLibrary*         (JNICALL* GetPortLibrary)     (VMInterface* vmi);
HyZipCachePool*        (JNICALL* GetZipCachePool)    (VMInterface* vmi);
HyVMLSFunctionTable*   (JNICALL* GetVMLSFunctions)   (VMInterface* vmi);


/* System properties */
vmiError (JNICALL* GetSystemProperty) (VMInterface* vmi, char* key, char** valuePtr);
vmiError (JNICALL* SetSystemProperty) (VMInterface* vmi, char* key, char* value);
vmiError (JNICALL* CountSystemProperties) (VMInterface* vmi, int* countPtr);
vmiError (JNICALL* IterateSystemProperties) (VMInterface* vmi,
                                  vmiSystemPropertyIterator iterator, void* userData);
```

VMs Expose the VMI Struct

Via Two Exported Functions

```
VMInterface* JNICALL VMI_GetVMIFromJavaVM (JavaVM* vm);

VMInterface* JNICALL VMI_GetVMIFromJNIEnv (JNIEnv* env);
```

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Harmony's Multi-Headed Launcher

Using the VMI to Launch Compatible VMs

- Multi-target launcher based upon JNI and VMI
- Ability to launch different VM implementations via command-line option
- Support for VM-specific options
  - Props files or command line

Harmony's Multi-Headed Launcher

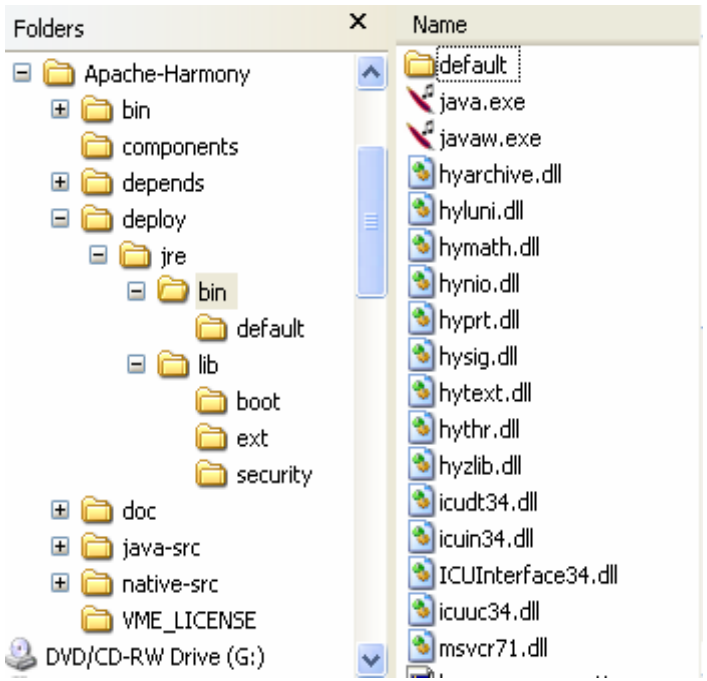
Using the VMI to Launch Compatible VMs

- VMs 'own' a subdirectory under jre/bin
  - VM implementation files and resources
  - VMI library and kernel classes
- Harmony VM launcher will:
  - Load the VM libraries
  - Prep VM-specific kernel classes
  - Invokes JNI_CreateJavaVM(…) , main(…), JarRunner…

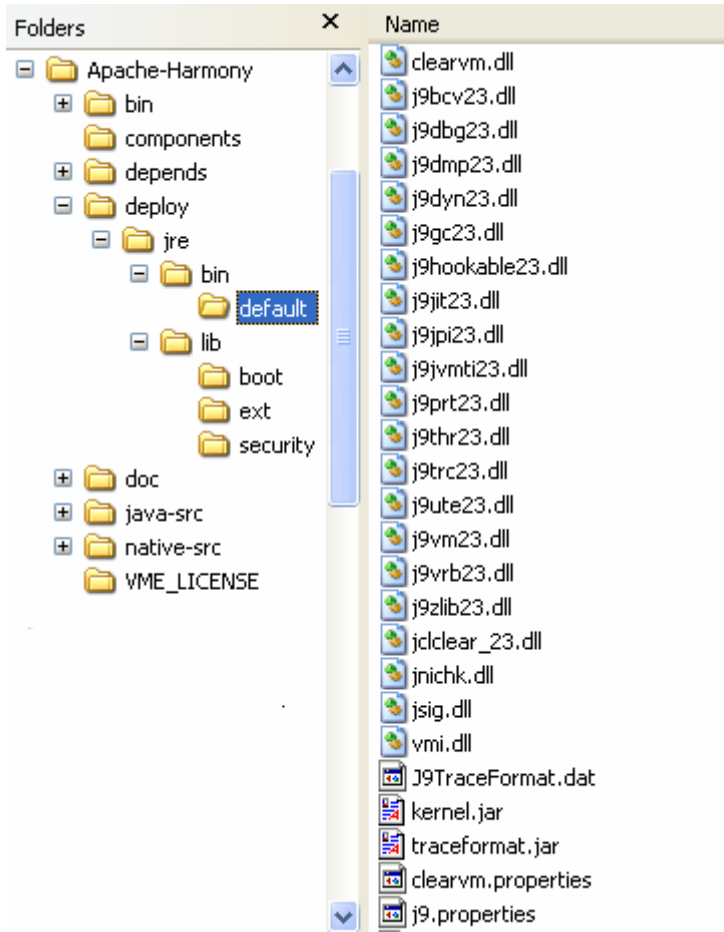
Harmony's Launcher

```
java -vmdir:<dir> -vm:<vmi_dll>
```

jre/bin



jre/bin/default

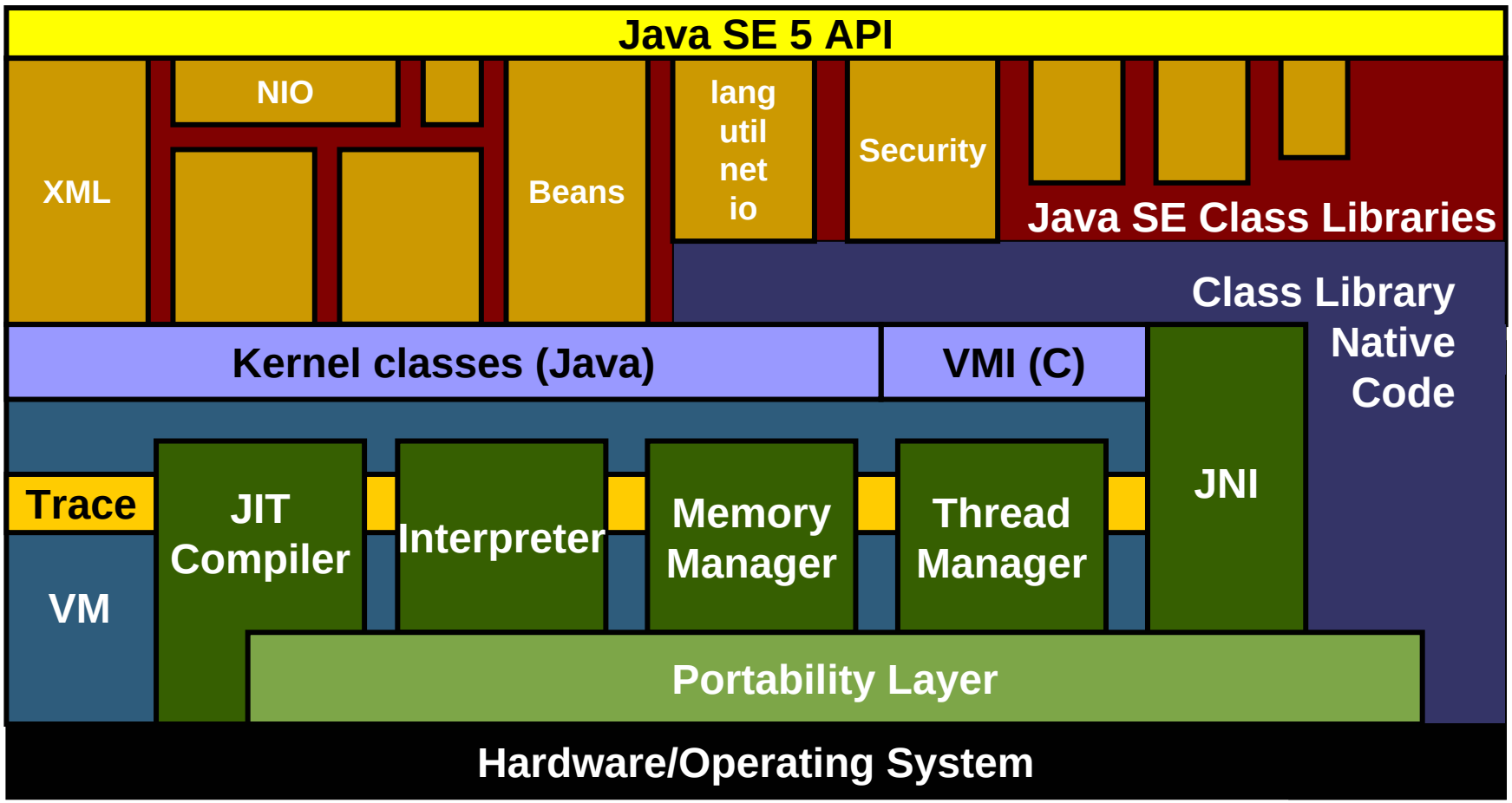


Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Apache Harmony—Architecture

The 30,000ft View



Class Library Modularity

Splitting Java SE into Manageable Chunks

- Makes the task of implementing Java SE manageable
  - Open source contributors can focus on specialities
- Makes the risk of prior exposure manageable
  - Can accommodate contributors in areas where they have not had prior access
- Factor-out third-party dependencies
- Facilitate assemblies of modules
  - Freedom of choice for module consumers
  - Unit of replacement for fixes and updates
- Facilitate contributions!

Class Library Modularity

Finding the Modules

- A module…
  - Is a set of related functionality
  - Is defined by a set of packages
  - May ‘export’ user-API and internal-API
  - May contain private internal implementation
- We discovered candidate modules by…
  - Looking at the package dependencies as expressed in the Java SE specification
  - Finding points of ‘weak implementation coupling’, i.e. minimizing use of internal APIs
  - Debating within the community which parts of the class library should be pluggable
  - Ensuring the resulting components are coherent chunks

Class Library Modularity

As they Are Defined Today…

Modules in progress...

Math	Logging	NIO
Prefs	Security	SQL
XML	JNDI™ API	IO
Auth	Beans	Crypto
Archive	JMX™ API	Text
RMI	Regex	Net
Lang	javax.net	Util

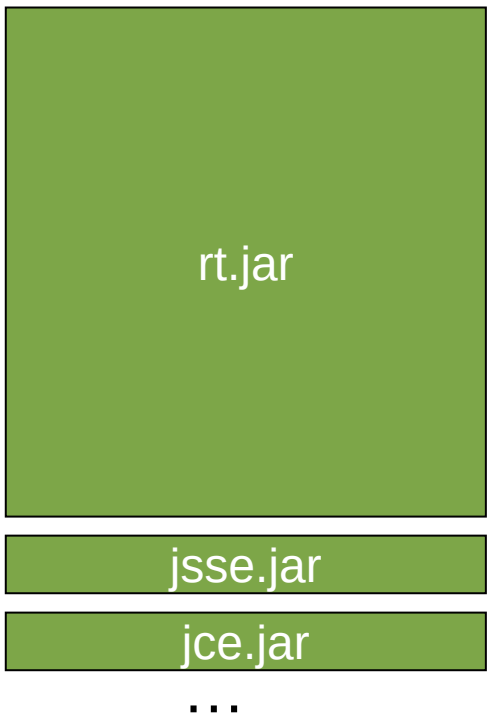
Modules not yet begun...

Lang-Management	Concurrent
ORB	Annotation
Image I/O	Instrument
Accessibility	Print
Sound	Swing
Applet	AWT

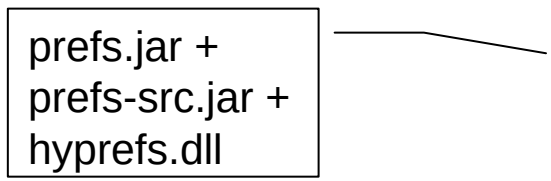
Class Library Modularity

Packaging Story

Conventional approach



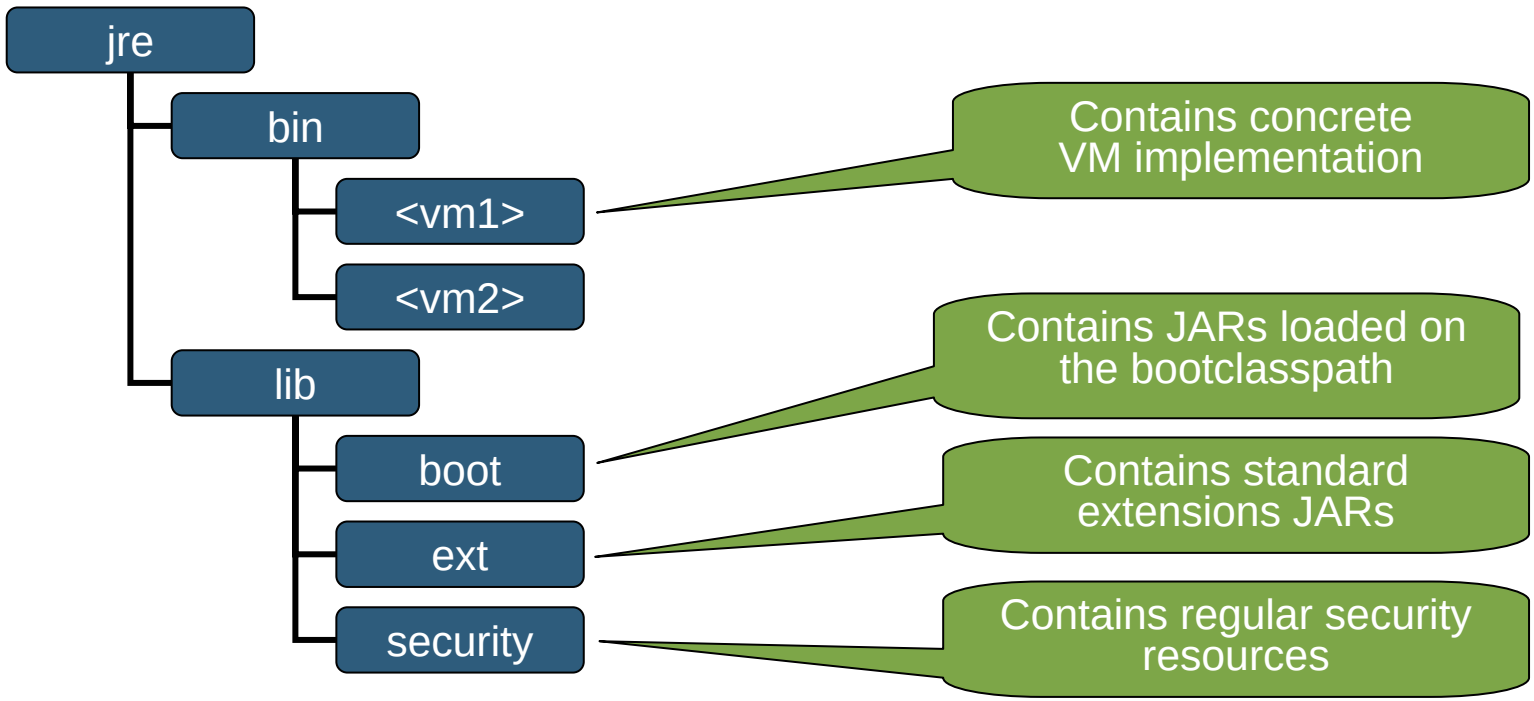
Harmony's approach



Class Library Modularity

Deployment Story

- Harmony Java runtime environments are laid out as follows



Class Library Modularity

Anatomy of a Module

- Modules are represented by separate source trees in the code repository
- Each module has OSGi metadata in its JAR manifest

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Harmony NIO
Bundle-SymbolicName: org.apache.harmony.nio
Bundle-Version: 1.0.0
Bundle-ClassPath: .
Import-Package: java.io, java.lang, java.lang.ref, java.net,
 java.nio.charset, java.security, java.util,…
Export-Package: java.nio, java.nio.channels,
java.nio.channels.spi, org.apache.harmony.nio
```

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

Harmony—Development Approach

Ensuring Intellectual Property (IP) Cleanliness

We must ensure a clean codebase and protect IP of others

- Contributors detail prior access via questionnaire
- Developers can contribute in functional areas where they have not studied closed-source implementations
- Existing code being contributed to the project must provide pedigree information

This is in addition to the standard Apache contribution processes

See: http://incubator.apache.org/harmony/auth_cont_quest.html

Harmony—Development Approach

Producing a Compatible Implementation

- We are following the Java SE specification as found in
  - Java SE 5 JavaDoc™
  - Java Language Specification
  - Java Virtual Machine Specification, etc.
- Testing! Testing! Testing!
  - Our own functional/API tests
  - Our own integration tests
  - Our own unit tests

  … all help to validate our implementation

We will soon apply for a Java SE 5 Java Compatibility Kit license

Harmony—Development Approach

Targeted Development Effort

Day to day life…

- Self-host—we support our own development process
  - Enough class library code to run Ant, Eclipse Java compiler
- Make it fun
  - Identify interesting target applications to run
  - Fill in the missing pieces
- Make it more fun (funner?)
  - Run the application's test suites

In the end, the community dictates where the effort is applied—
We're all volunteers… Scratch an itch!

Harmony—Development Approach

Measuring Progress

How do we track it all?

- At the coarsest level—“Has a module been started or not?”
- At a finer level—“How complete is it?”
- Use tools to determine binary compatibility with reference implementation (JAPI tools)
- Test coverage reports ensure that inherited behavior has been tested

Our goal is completeness in the modules we have

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

DEMO

Agenda

- A Brief History of the Project
- Architectural Overview
- A Tour of the Code
  - A Look at the Class Library/VM Interface
  - The Harmony Launcher
  - Class Library Modularity
- Development Approach
- Demo
- Future Directions

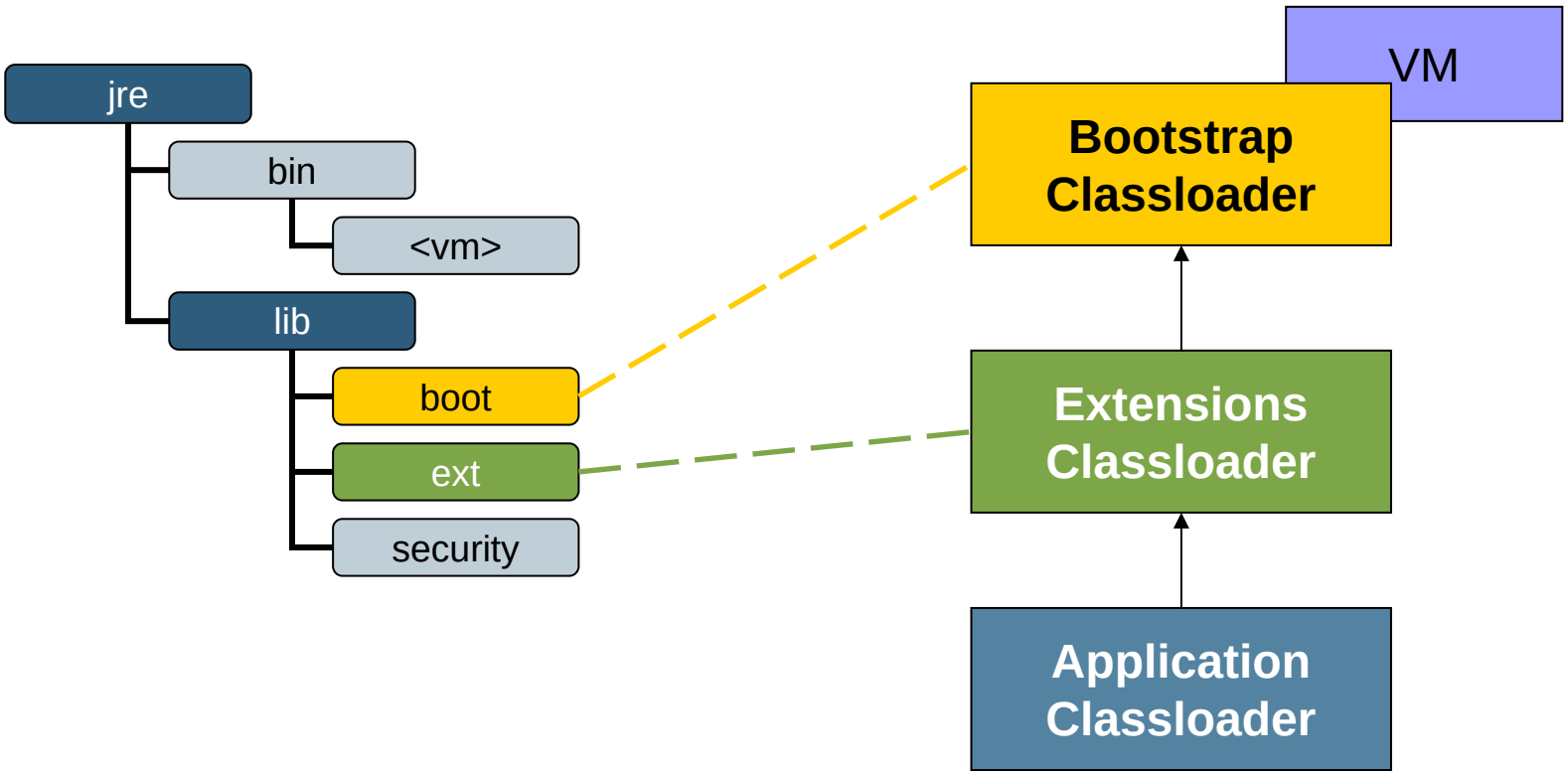
Apache Harmony—Future Directions

Runtime Support for Class Library Modularity

- OSGi runtime framework for Java based modules
  - Allows coincidental loading of versioned ‘bundles’
  - Enforce visibility rules defined by the module metadata
  - Lifecycle for deployment of Java and associated natives, start, stop, uninstall, etc.

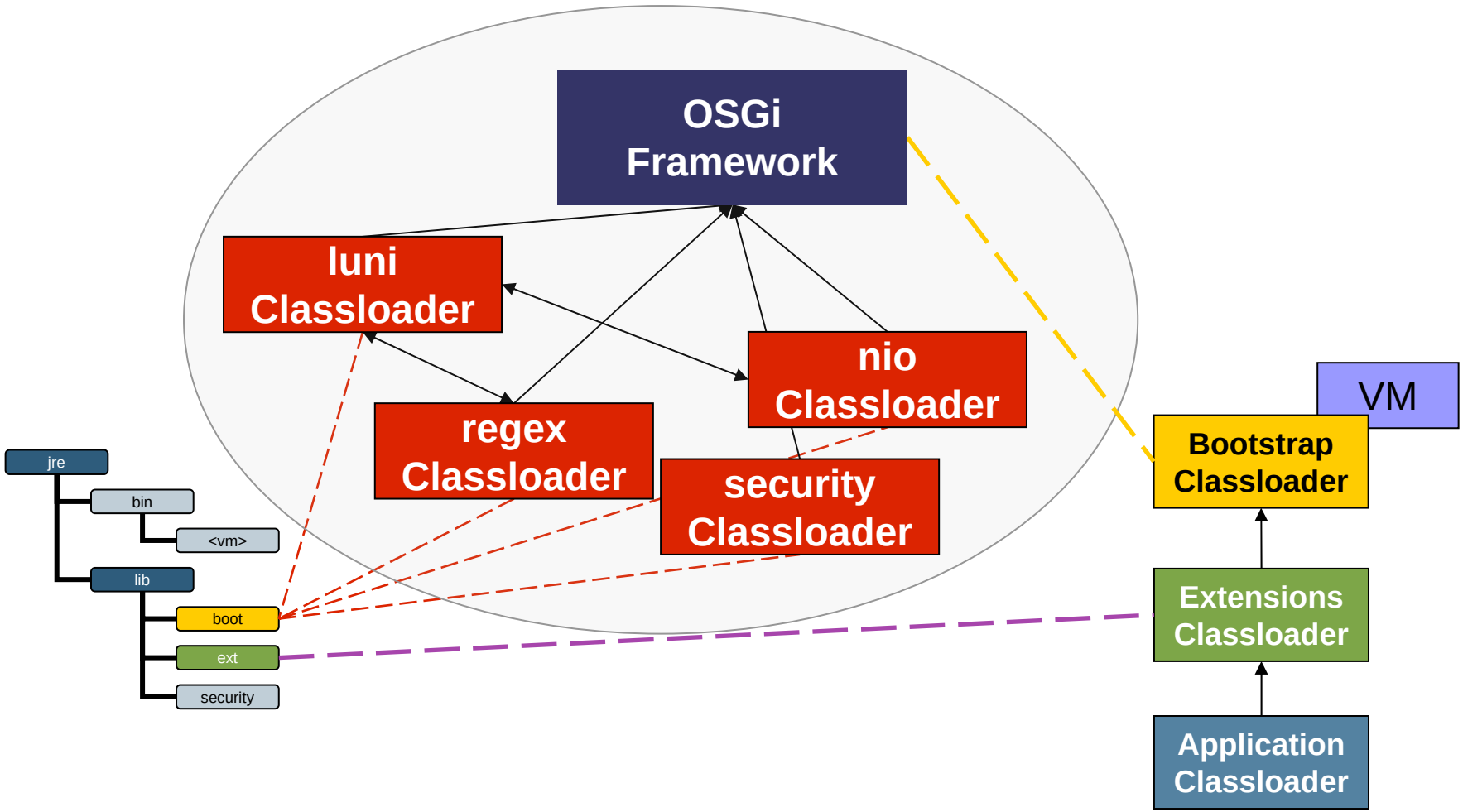
Apache Harmony—Future Directions

Runtime Support for Class Library Modularity



Apache Harmony—Future Directions

Runtime Support for Class Library Modularity



Summary

- Our goal is a compatible, open-source, Java SE implementation
- We invested time up-front getting the IP infrastructure right
- The community is focussed on building a first-class, modular runtime environment
- We have come a long way in one year!

Come and contribute to the fun in your area of interest!

For More Information

Here at JavaOne

- Talk to us! Buy us beer!
- BOF-0755, The Apache Harmony Project
  - Tonight—Thursday May 18, 6:30-7:30, Hall E 133

Out on the web

- Apache Harmony website and mailing list
  `http://incubator.apache.org/harmony`

Q&A

Tim Ellison
tellison@apache.org
Geir Magnusson Jr.
geirm@apache.org

Apache Harmony's Approach to Implementing Java™ Platform, Standard Edition

Tim Ellison
Geir Magnusson Jr.

Apache Harmony Project

www.incubator.apache.org/harmony

TS-3752