



the
POWER
of
JAVA™



Apache Harmony

<http://incubator.apache.org/harmony/>

JavaOne
Part of the Network of Software Solutions

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...



Apache Harmony

<http://incubator.apache.org/harmony/>

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

Be **inspired** to participate in the Harmony community

**Apache Harmony**<http://incubator.apache.org/harmony/>

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**<http://incubator.apache.org/harmony/>

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

Bits of History

- Proposal accepted by the Apache Software Foundation (ASF) in **May 2005**
- Project started in **the incubator**
 - Provides ASF oversight to Harmony's operating standards
 - A place for growing the community
 - "Business as usual" for new projects at the ASF
- Two main project goals:
 - Create a **compatible, independent implementation** of **Java SE** available under the **Apache Software License V2**
 - Create a **community-developed, modular** runtime (VM and class library) architecture to allow independent implementations to share runtime components, and allow independent innovation in runtime components

Apache Harmony

Bits of History—Summer 2005

- Discussion of broad technical approach
 - How to define modules in Java and C code
 - Pact to maximize the use of Java in class library code
 - Suitability of Java as a virtual machine (VM) implementation language
- Discussion of open source license compatibility
 - Desire to reuse existing open source efforts
 - Looking for opportunities to work with GNU Public Licensed (\pm exception) code
- Discussion of development rules
 - Acceptable prior access for contributors
 - Acceptable provenance for repurposed code contributions

Apache Harmony

Bits of History—Fall 2005

- Resolved discussion topics with:
 - Prototype code for native modules, and agreement to split class library into logical modules
 - Reluctant recognition that reconciling GPLv2 and ASLv2 was beyond project scope
 - A set of guidelines and eligibility criteria for contributors (people) and contributions (code)
- Received code contributions of
 - ‘JC’ VM by Archie Cobbs (September 2005)
 - ‘Bootstrap’ VM by Dan Lydick (September 2005)

Apache Harmony

Bits of History—Winter 2005/2006

- Received further code contributions
 - Core set of Java SE classes, and class library/VM interface from IBM Corp. (November 2005)
 - Much fuller implementation of Java SE security and cryptography code from Intel Corp. (November 2005)
- Evaluation JVMTM implementing proposed interface
 - Available from IBM Developerworks site
 - Not open source, but non-commercial, evaluation-style license
- ...and since then
 - More code contributions to the project, both from corporations, and individuals' direct participation

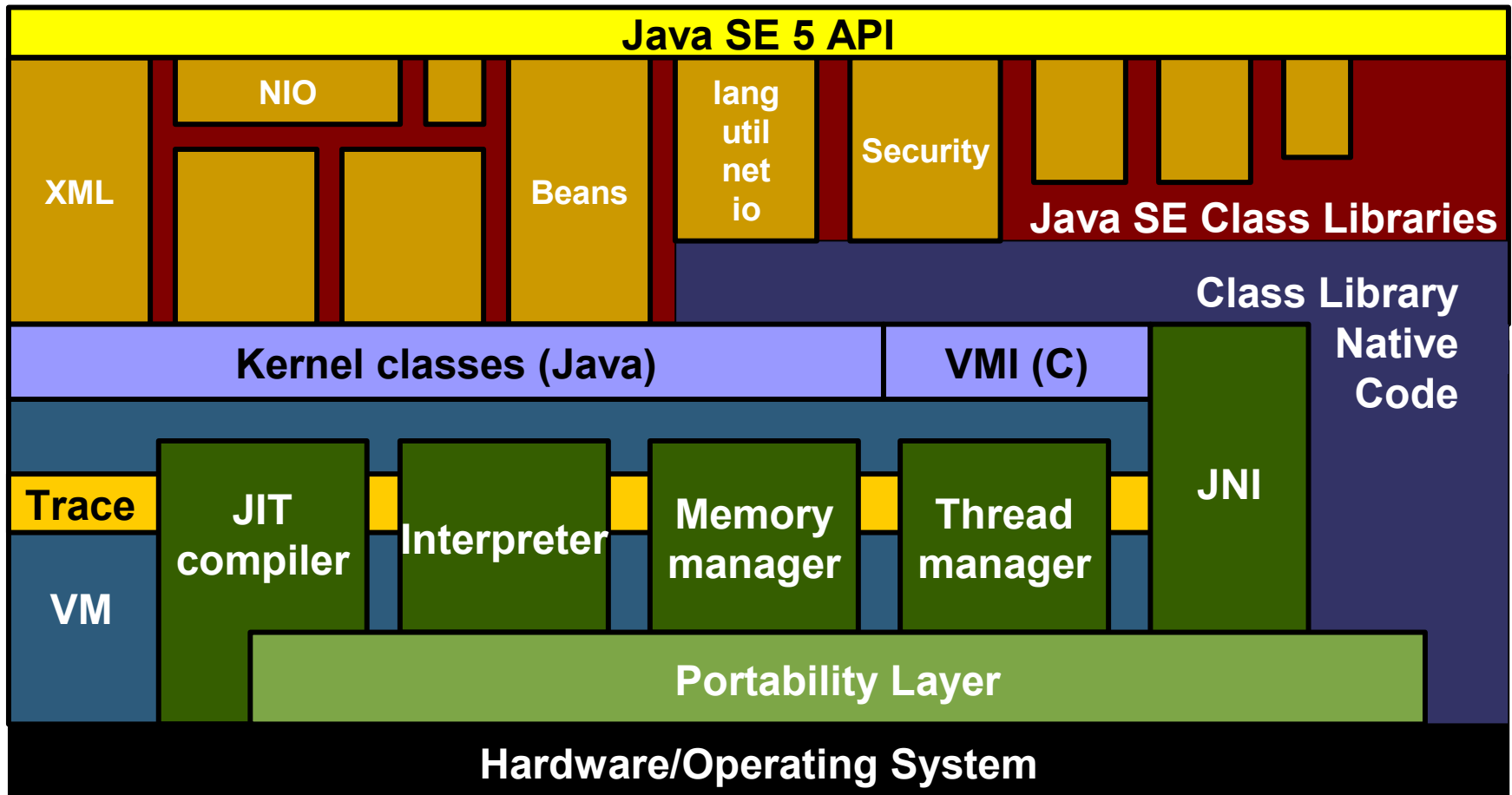
**Apache Harmony**<http://incubator.apache.org/harmony/>

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



**Apache Harmony**<http://incubator.apache.org/harmony/>

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.lang.Class
java.lang.ClassLoader
java.lang.Compiler
java.lang.Package
java.lang.Runtime
java.lang.StackTraceElement
java.lang.System
java.lang.Thread
java.lang.ThreadGroup
java.lang.Throwable
java.security.AccessControlContext
java.security.AccessController
java.lang.reflect.AccessibleObject
java.lang.reflect.Array
java.lang.reflect.Constructor
java.lang.reflect.Field
java.lang.reflect.Method
java.lang.ref.PhantomReference
java.lang.ref.Reference
java.lang.ref.WeakReference
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) ;
```

**Apache Harmony**<http://incubator.apache.org/harmony/>

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

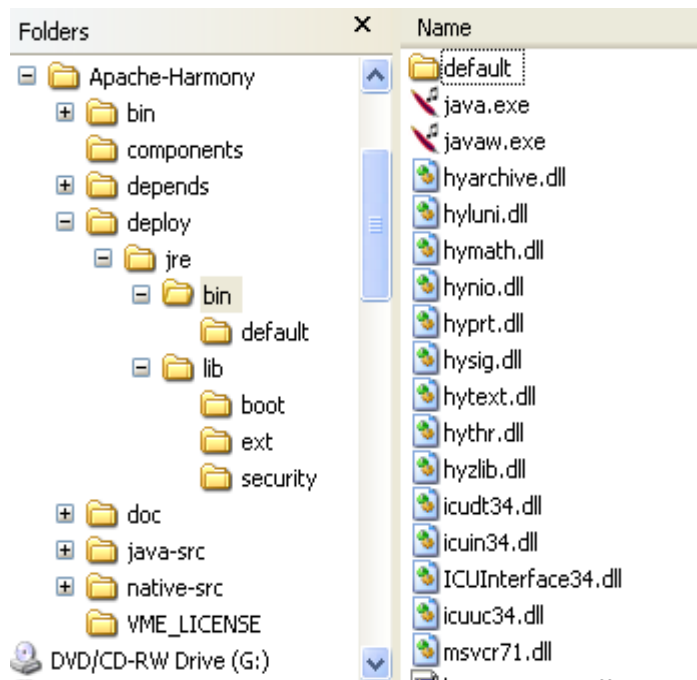
Using the VMI to Launch Compatible VMs

- 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...`

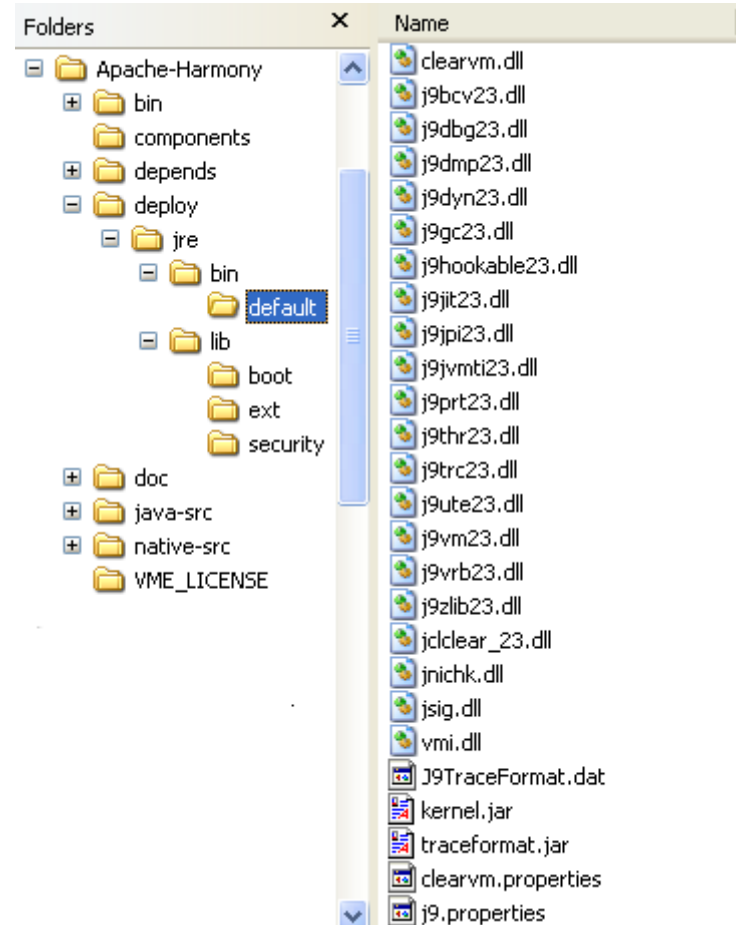
Harmony's Launcher

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

jre/bin



jre/bin/default



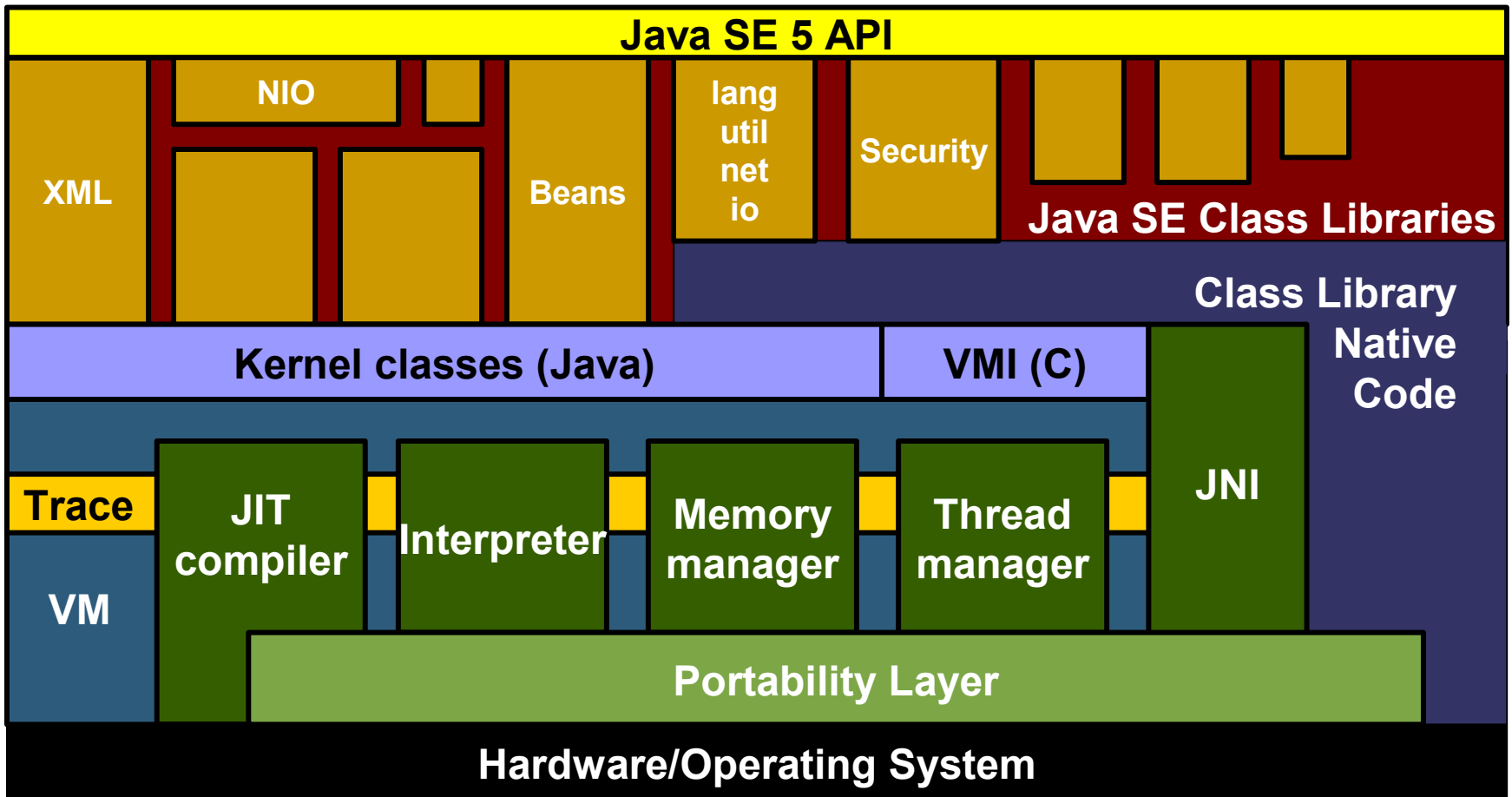
**Apache Harmony**<http://incubator.apache.org/harmony/>

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!

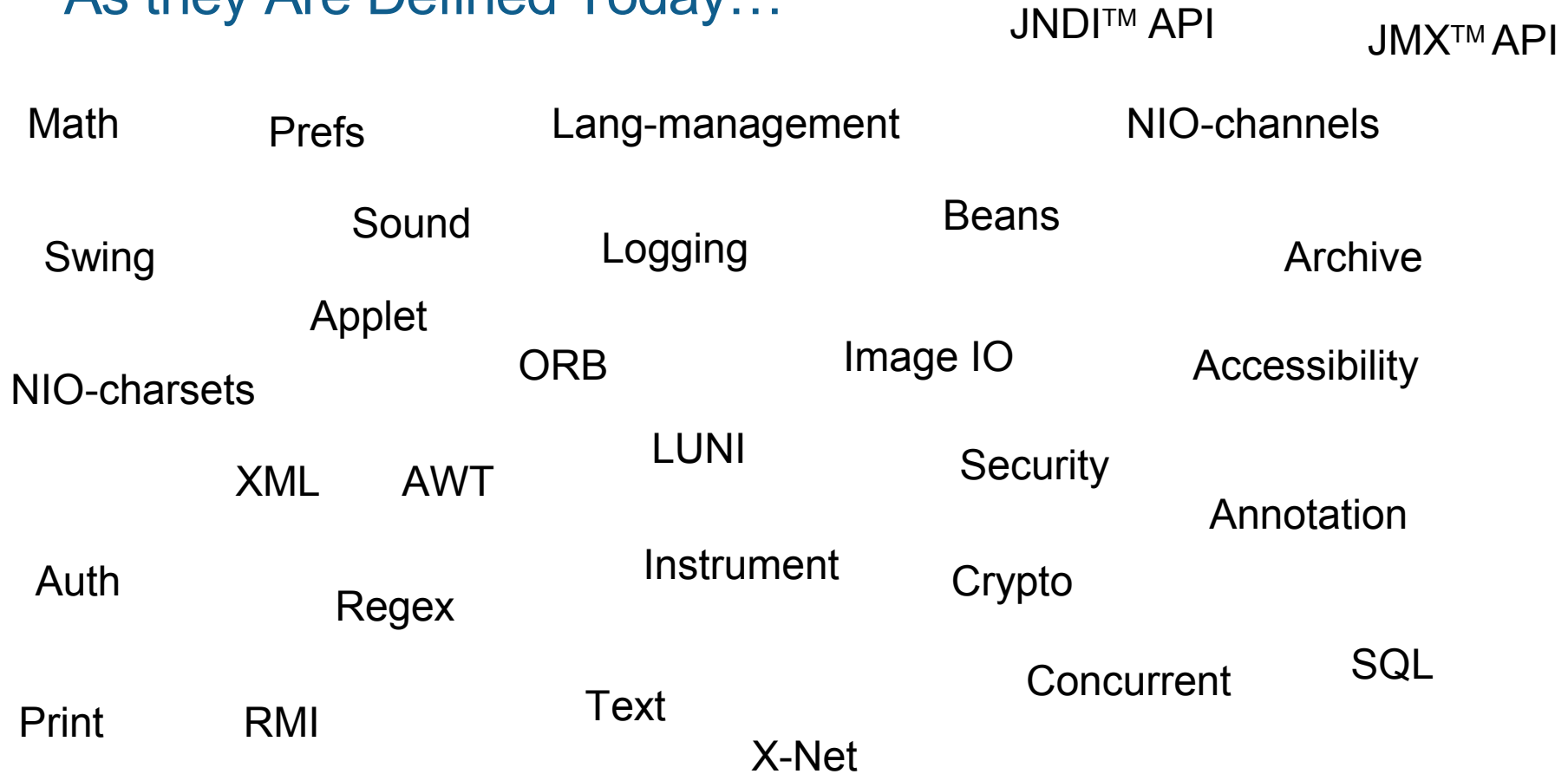
Class Library Modularity

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

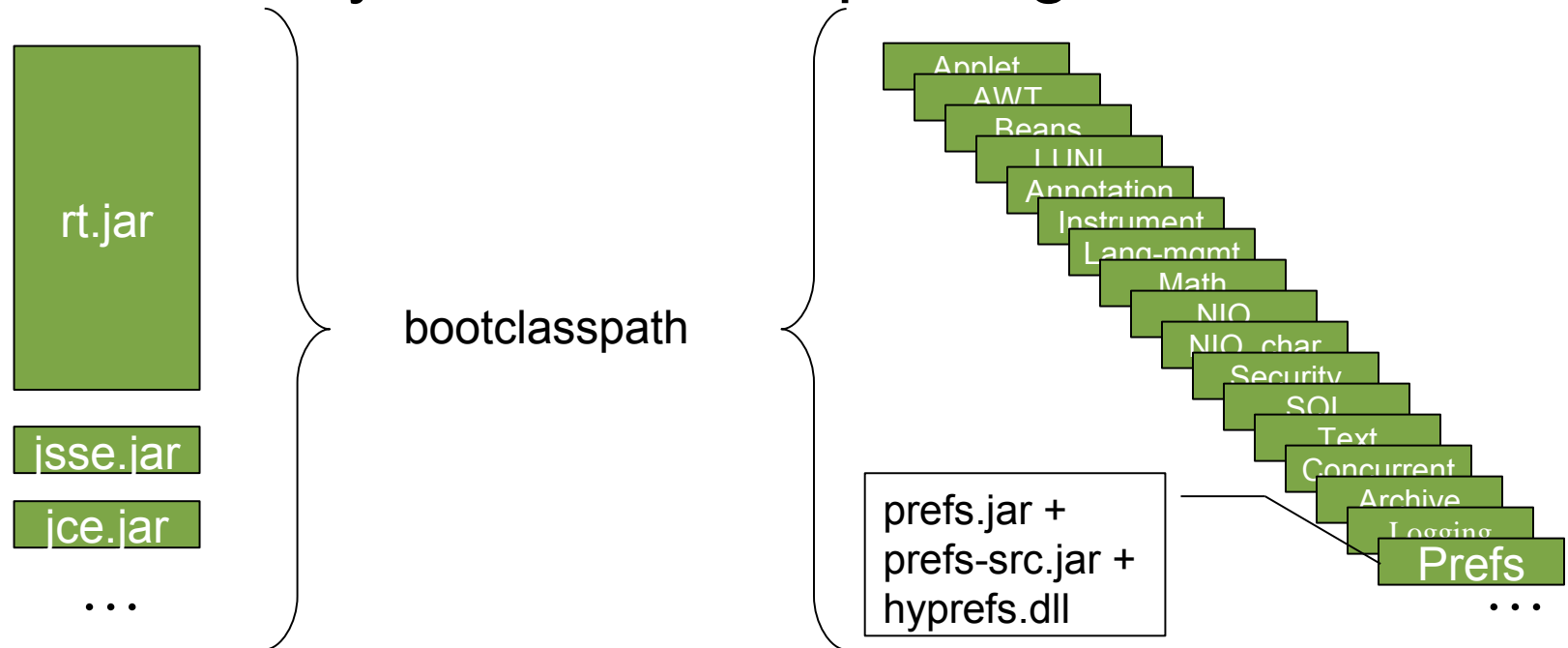
As they Are Defined Today...



Class Library Modularity

Packaging Story

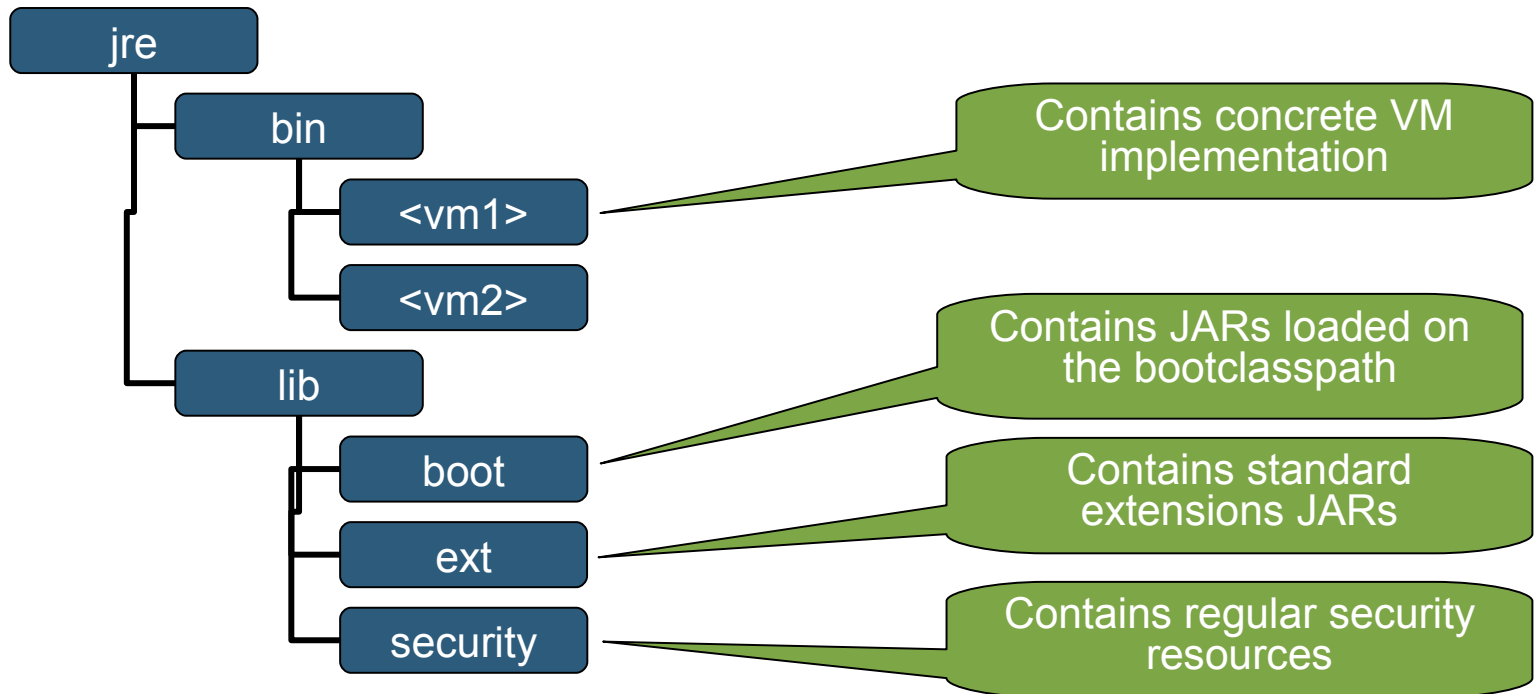
- Java SE runtimes typically comprise monolithic JARs and native libraries
- In Harmony each module packaged as a JAR



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: org.apache.harmony.nio;x-friends:="org.apache.
    harmony.luni",java.nio,java.nio.channels,java.nio.channels.spi
  
```

**Apache Harmony**<http://incubator.apache.org/harmony/>

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

- Contributors are asked to complete a questionnaire concerning prior access
- Developers can contribute in functional areas where they have not studied closed-source implementations (with certain exceptions)
- Existing code being contributed to the project must provide pedigree information
- Goal is to protect the Harmony code from inappropriate contributions and, as importantly, respect the property rights of other implementers

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

Harmony—Development Approach

Producing a Compatible Implementation

- Follow Java SE specification embodied in JavaDoc™, Java language specification, Java Virtual Machine specification, etc.
- Harmony's functional tests drive API to validate our implementation against reference implementation
- When we think we are ready, we will apply for a Java SE 5 Java Compatibility Kit technology compatibility kit scholarship

Harmony—Development Approach

Targeted Development Effort

- Identify interesting target applications to run. Fill in the missing pieces
- Focus on ability to support our own development process first
 - Enough code to run Ant, Eclipse Java compiler, ...
- Run the application's test suites to determine success
- Encourage application experts to report successes and problems to Harmony community
- In the end, the community dictates where the effort is applied

Harmony—Development Approach

Measuring Progress

- At the coarsest level—has a module been started or not?
- Use tools to determine binary compatibility with reference implementation (Java language tools)
- Test coverage reports ensure that inherited behavior has been tested
- Our ability to support increasingly sophisticated target applications
- Ultimately the Java Compatibility Kit will determine completeness

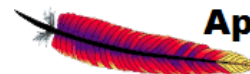
**Apache Harmony**<http://incubator.apache.org/harmony/>

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



**Apache Harmony**<http://incubator.apache.org/harmony/>

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**

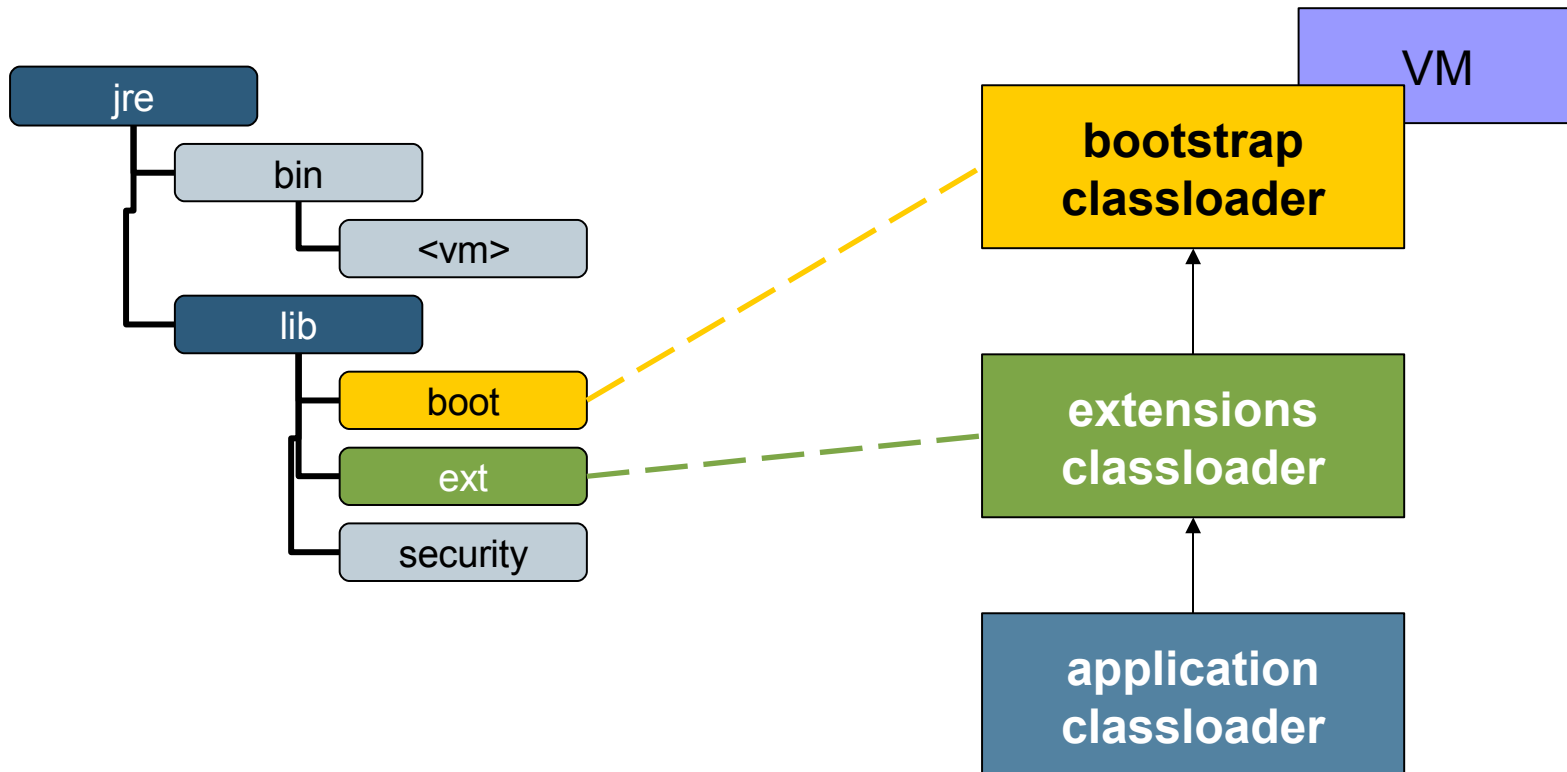
Apache Harmony—Future Directions

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 technology and associated natives, start, stop, uninstall, etc.

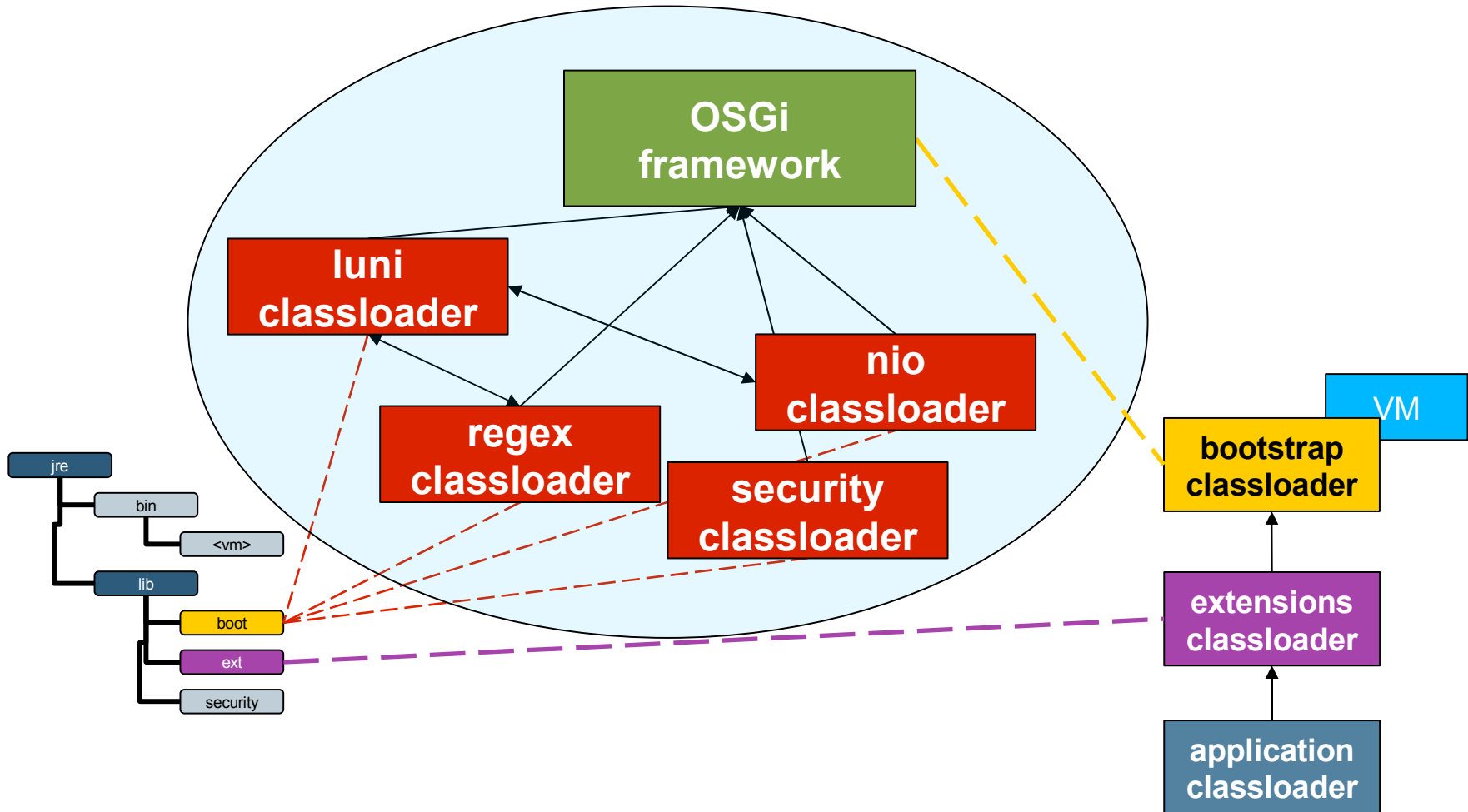
Apache Harmony—Future Directions

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!
- BOF-0755—The Apache Harmony Project

Out on the web

- Apache Harmony website and mailing lists
<http://incubator.apache.org/harmony>

Q&A

Tim Ellison
Geir Magnusson Jr.

tellison@apache.org
geirm@apache.org



the
POWER
of
JAVA™



Apache Harmony

<http://incubator.apache.org/harmony/>

JavaOne
Part of the Network of Software Solutions

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