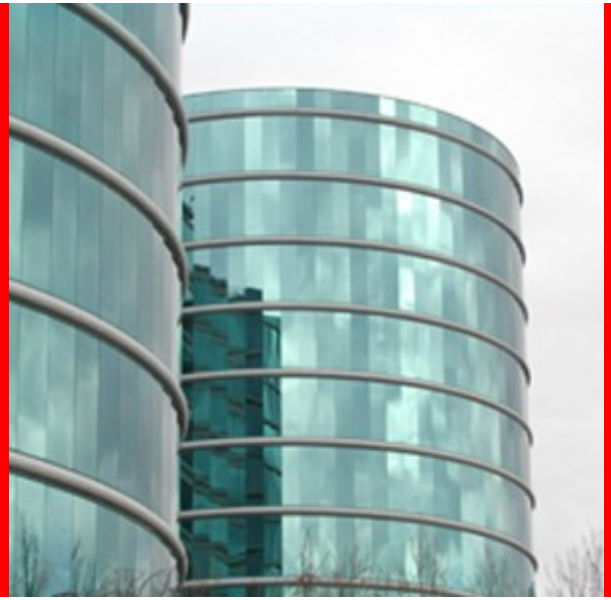
ORACLE®



ORACLE®

JDK7 The Future of The Java Platform

Simon Ritter
Technology Evangelist

JDK 7 Features

- Language
  - Annotations on Java types
  - Small language changes (Project Coin)
  - Modularity (JSR-294)
- Core
  - Modularisation (Project Jigsaw)
  - Concurrency and collections updates
  - More new IO APIs (NIO.2)
  - Additional network protocol support
  - Eliptic curve cryptography
  - Unicode 5.1

JDK 7 Features

- VM
  - Compressed 64-bit pointers
  - Garbage-First GC
  - Support for dynamic languages (Da Vinci Machine project)
- Client
  - Forward port Java SE6 u10 features
  - XRender pipeline for Java 2D

Modularity

Project Jigsaw

Size Matters

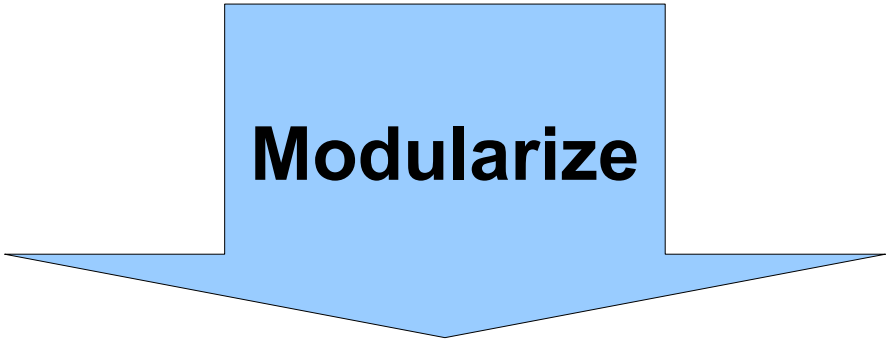
- JDK is big, really big
  - JDK 1.x – 7 top level packages, 200 classes
  - JDK 6 – 47 top level packages, over 4000 classes
  - About 12MB today
- Historical baggage
  - Build as a monolithic software system
- Problems
  - Download time, startup time, memory footprint, performance
  - Difficult to fit platform to newer mobile device
    - CPU capable, but constrained by memory
- Current solution – use JDK 6u10
  - Kernel installer, quick startup feature
  - More like a painkiller, not the antidote

New Module System for Java

- JSR-294 Improved Modularity Support in the Java Programming Language
- Requirements for JSR-294
  - Integrate with the VM
  - Integrate with the language
  - Integrate with (platform's) native packaging
  - Support multi-module packages
  - Support “friend” modules
- Open JDK's Project Jigsaw
  - Reference implementation for JSR-294
  - openjdk.java.net/projects/jigsaw
  - Can have other type of module systems based on OSGi, IPS, Debian, etc.

Modularizing You Code

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




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

module-info.java

Module name

```
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 system

Dependencies

Main class

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;
}
```

Small (Language) Changes

Project Coin

“Why don't you add X to Java?”

- Assumption is that adding features is always good
- Application
  - Application competes on the basis of completeness
  - User cannot do X until application supports it
  - Features rarely interacts “intimately” with each other
  - Conclusion: more features are better
- Language
  - Languages (most) are Turing complete
  - Can always do X; question is how elegantly
  - Features often interact with each other
  - Conclusion: fewer, more regular features are better

Adding X To Java

- Must be compatible with existing code
  - assert and enum keywords breaks old code
- Must respect Java's abstract model
  - Should we allowing padding/aligning object sizes?
- Must leave room for future expansion
  - Syntax/semantics of new feature should not conflict with syntax/semantics of existing and/or potential features
  - Allow consistent evolution

Changing Java Is Really Hard Work

- Update the Java Language Specification
- Implement it in the compiler
- Add essential library support
- Write test
- Update the VM specs
- Update the VM and class file tools
- Update JNI
- Update reflection APIs
- Update serialization support
- Update javadoc output

Better Integer Literals

- Binary literals

```
int mask = 0b1010;
```

- With underscores

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

- Unsigned literals

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

Better Type Inference

```
Map<String, Integer> foo =
          new HashMap<String,
Integer>();

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

- 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
}
```

Index Syntax for Lists and Maps

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

Map<Integer, String> map = new HashMap<>();
map[1] = "One";
```

Multi Catch (Maybe Back in JDK7)

```
try { ...
} catch (InvalidClassException e) { foo(); }
} catch (InvalidObjectException e) { foo(); }
} catch (FileNotFoundException e) { bar(); }
```

- Longstanding request to allow catching Ex1 and Ex2 together

```
try { ...
} catch (InvalidClassException,
                InvalidObjectException e1)
{ foo(); }
} catch (FileNotFoundException e2) { bar(); }
```

- Members of e1 and e2 have direct common superclass
  - `ObjectStreamException`

Dynamic Languages Support

Da Vinci Machine Proejct

Virtual Machines

- A software implementation of a specific computer architecture
  - Could be a real hardware (NES) or fictitious (z-machine)
- Popular in current deployments
  - VirtualBox, VMWare, VirtualPC, Parallels, etc
- “Every problem in computer science can be solved by adding a layer of indirection” – Butler Lampson
  - VM is the indirection layer
  - Abstraction from hardware
  - Provides portability

VM Have Mostly Won the Day

- Lots of languages today are targeting VM
  - Writing native compiler is a lot of work
- Why? Language need runtime support
  - Memory management, reflection, security, concurrency controls, tools, libraries, etc
- VM based systems provides these
  - Some features are part of the VM
- Lots of VM for lots of different languages
  - JVM, CLR, Dalvik, Smalltalk, Perl, Python, YARV, Rubinius, Tamarin, Valgrind (C++!), Lua, Postscript, Flash, p-code, Zend, etc

JVM Architecture

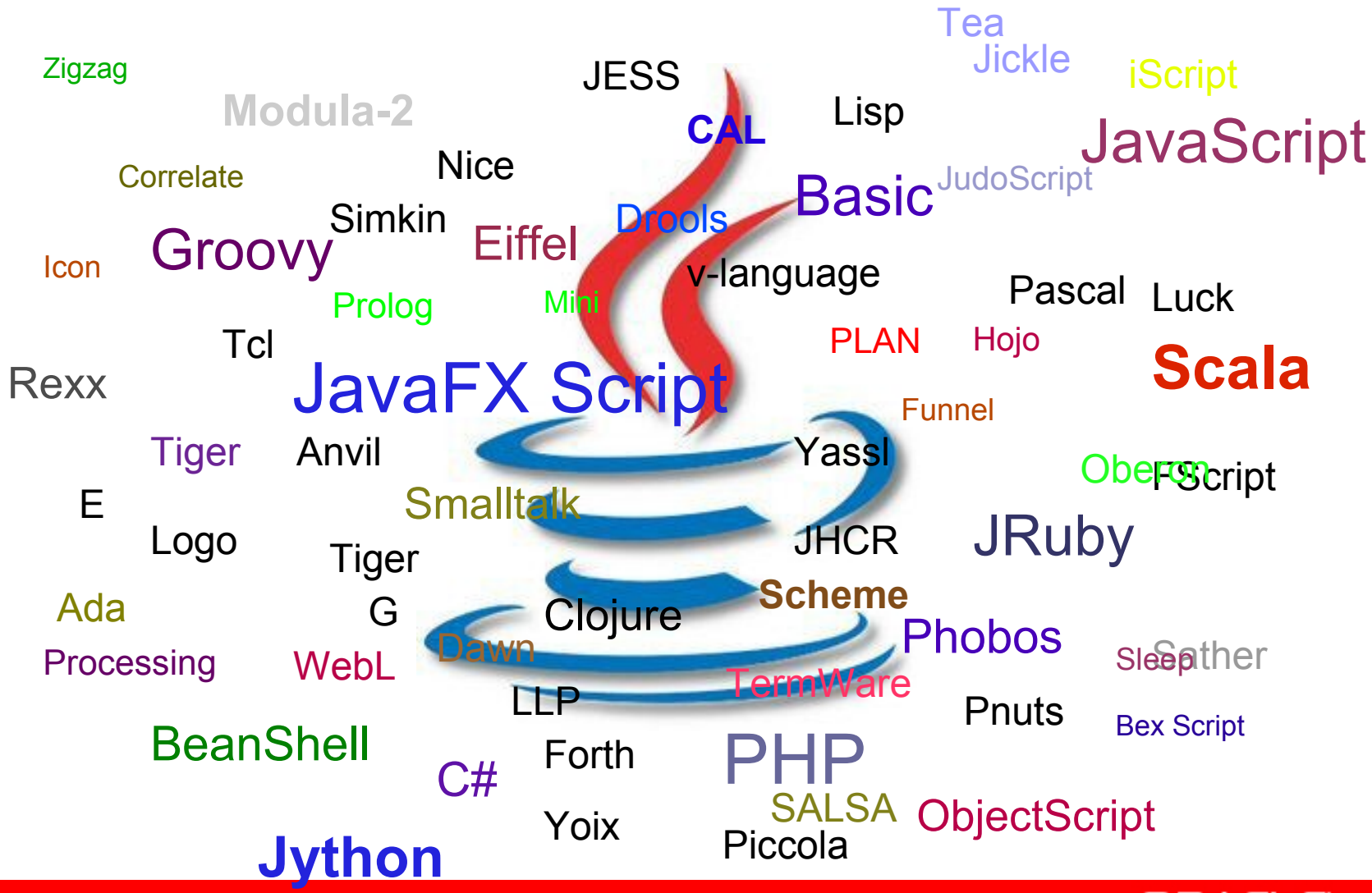
- Stack based
  - Push operand on to the stack
  - Instructions operates by popping data off the stack
  - Pushes result back on the stack
- Data types
  - Eight primitive types, objects and arrays
- Object model – single inheritance with interfaces
- Method resolution
  - Receiver and method name
  - Statically typed for parameters and return types
  - Dynamic linking + static type checking
    - Verified at runtime

JVM Specification

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

1.2 The Java Virtual Machine Spec.

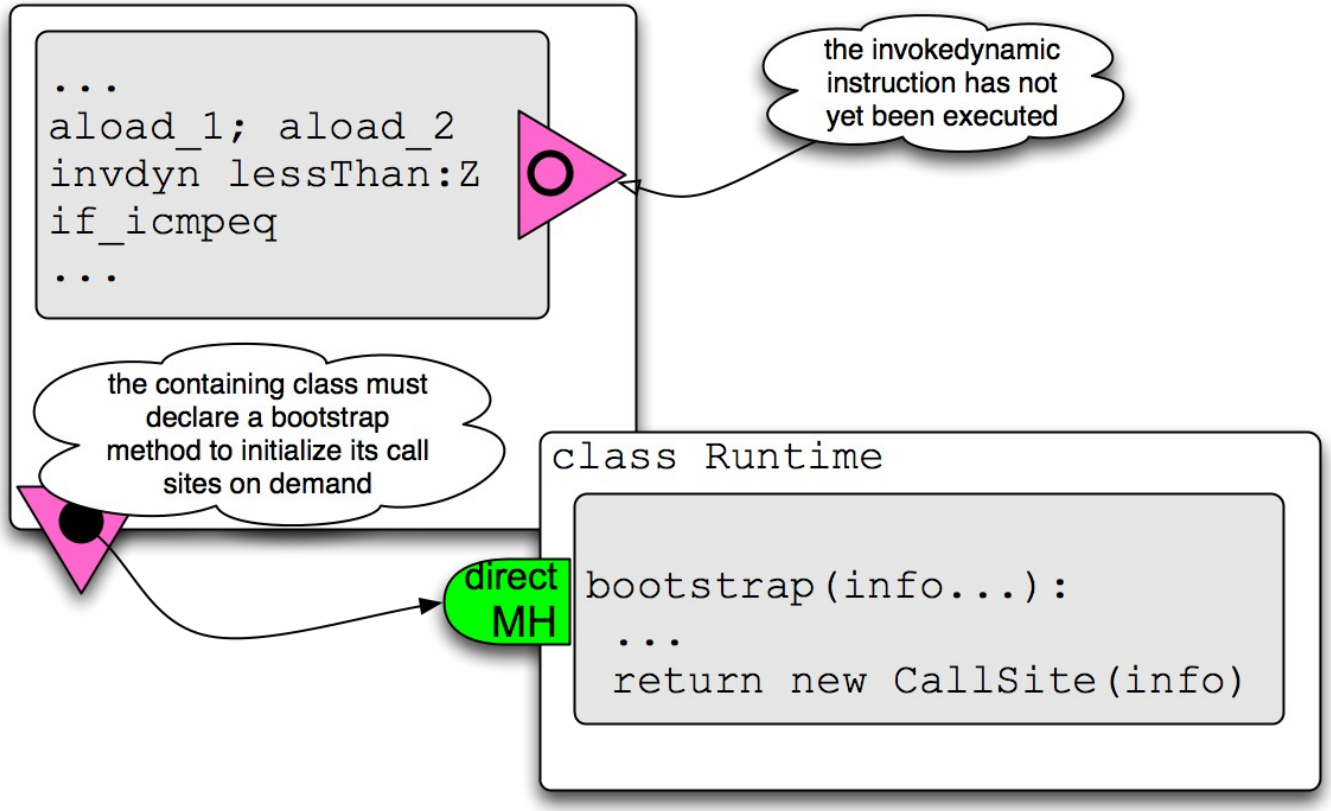
Languages Running on the JVM



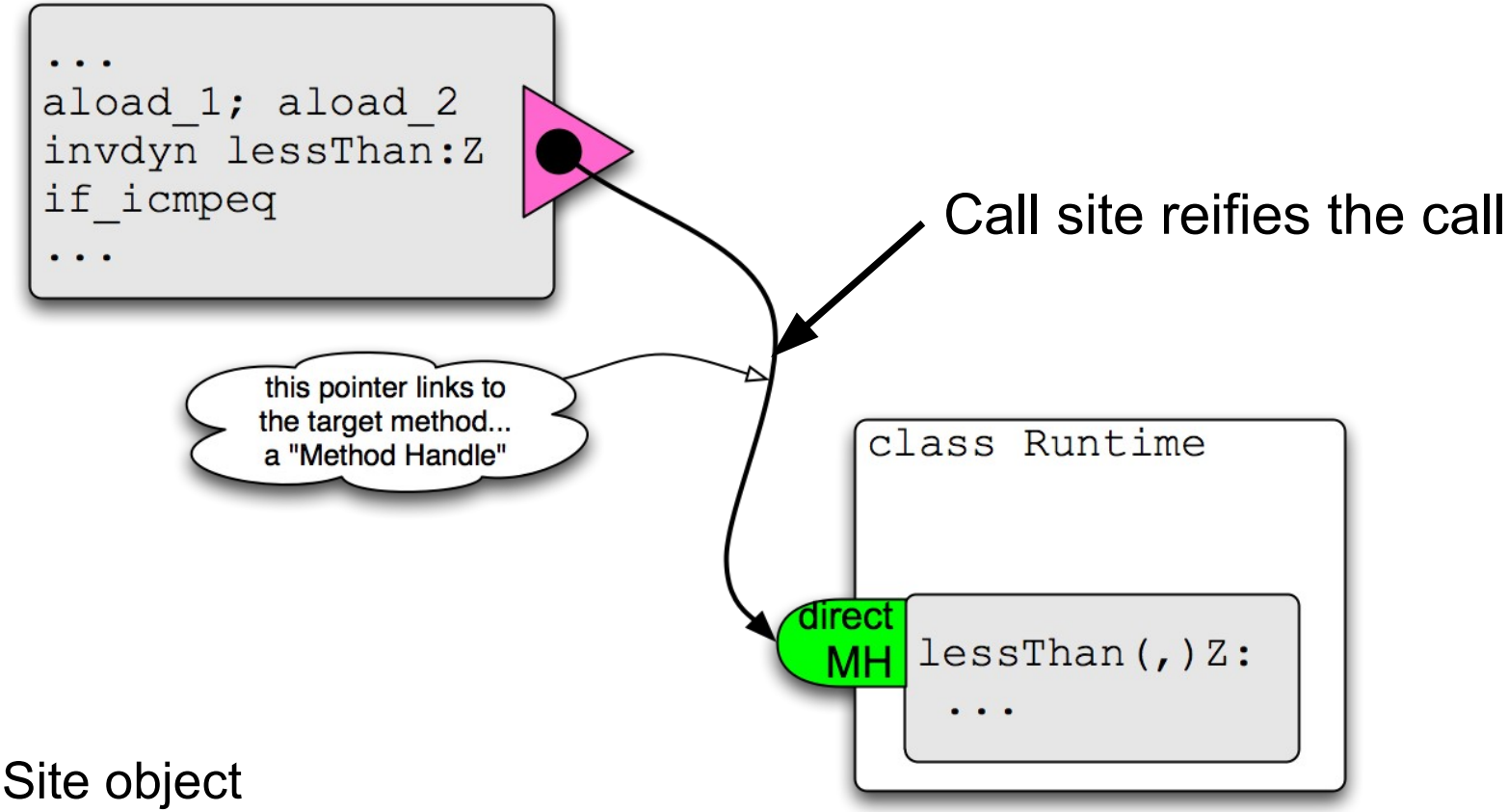
InvokeDynamic Bytecode

- JVM currently has four ways to invoke method
  - Invokevirtual, invokeinterface, invokestatic, invokespecial
- All require full method signature data
- InvokeDynamic will use method handle
  - Effectively an indirect pointer to the method
- When dynamic method is first called bootstrap code determines method and creates handle
- Subsequent calls simply reference defined handle
- Type changes force a re-compute of the method location and an update to the handle
  - Method call changes are invisible to calling code

Bootstrapping Method Handle



`invokedynamic` Byte Code



pink = CallSite object
green = MethodHandle object

Closures (Again)

- “I think it's time to add closures to Java”
  Mark Reinhold, Devoxx (November, 2009)
- What closures won't have
  - Control invocation statements
  - Non-local returns
  - Access to non-final variables unlikely

Closures (What We Know So Far)

- Two key features
  - Literal syntax for writing closures
  - Function types (closures are first class types)
- Platform requires two additional features
  - Closure conversion
  - Extension methods (for existing libraries like Collections)

JDK 7 Milestone Timeline

- M5 29 Oct 2009
  - Concurrency and collections updates (jsr166y)
  - Elliptic-curve cryptography (ECC)
  - JSR TBD: Small language enhancements (Project Coin)
  - Swing updates
  - Update the XML stack
- M6 18 Feb 2010
- M7 15 Apr 2010
- M8 3 Jun 2010
- M9 22 Jul 2010
- M10 9 Sep 2010

Summary

- Lots of new things coming
  - Some not covered here
  - Includes annotations for Java type, JXLayer, date picker, etc.
- Will make Java applications smaller, more concise, easier to read and understand
- Lots of nice libraries that will exploit the hardware
  - Xrender pipeline, filesystem G1 collector, forkjoin, compressed pointer 64-bit VM, etc
- Platform will be more robust and scalable
- Coming in 2010

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE®