# JVM for Dummies

(and for the rest of you, as well)

# Intro

- Charles Oliver Nutter
  - "JRuby Guy"
  - Sun Microsystems 2006-2009
  - Engine Yard 2009-
- Primarily responsible for compiler, perf
  - Lots of bytecode generation
  - Lots of JIT monitoring

# Today

- JVM Bytecode
  - Inspection
  - Generation
  - How it works
- JVM JIT
  - How it works
  - Monitoring
  - Assembly (don’t be scared!)

# Today

- JVM Bytecode
  - Inspection
  - Generation
  - How it works
- JVM JIT
  - How it works
  - Monitoring
  - Assembly

For Dummies

For people who want to feel like Dummies

# Part One: Bytecode

# Bytecode Definition

- "... instruction sets designed for efficient execution by a software interpreter ..."
- "... suitable for further compilation into machine code.

# Byte Code

- One-byte instructions
- 256 possible “opcodes”
- 200 in use on current JVMs
  - Room for more :-)
- Little variation since Java 1.0

# Microsoft's CLR

- Stack-based, but not interpreted
- Two-byte "Wordcodes"
- Similar operations to JVM

# Why Learn It

- Know your platform
    - Full understanding from top to bottom
- Bytecode generation is fun and easy
    - Build your own language?
- May need to read bytecode someday
    - Many libraries generate bytecode

# Hello World

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world");
    }
}
```

# javap

- Java class file disassembler
- Basic operation shows class structure
  - Methods, superclasses, interface, etc
- -c flag includes bytecode
- -public, -private, -protected
- -verbose for stack size, locals, args

# javap

```
~/projects/bytecode_for_dummies ➔ javap HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
    public HelloWorld();
    public static void main(java.lang.String[]);
}
```

# javap -c

```
~/projects/bytecode_for_dummies ➔ javap -c HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0:  aload_0
   1:  invokespecial #1; //Method java/lang/Object."<init>":()V
   4:  return

public static void main(java.lang.String[]);
  Code:
   0:  getstatic  #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:  ldc #3; //String Hello, world
   5:  invokevirtual #4; //Method java/io/PrintStream.println:
                                                     (Ljava/lang/String;)V
   8:  return

}
```

# javap -verbose

```
~/projects/bytecode_for_dummies ➔ javap -c -verbose HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object
  SourceFile: "HelloWorld.java"
  minor version: 0
  major version: 50
  Constant pool:
const #1 = Method   #6.#15; //  java/lang/Object."<init>":()V
const #2 = Field#16.#17;//  java/lang/System.out:Ljava/io/PrintStream;
const #3 = String   #18;//  Hello, world
const #4 = Method   #19.#20;//  java/io/PrintStream.println:(Ljava/lang/String;)V
const #5 = class#21;//  HelloWorld
...

{
```

# javap -verbose

```
...
public HelloWorld();
  Code:
   Stack=1, Locals=1, Args_size=1
   0: aload_0
   1: invokespecial  #1; //Method java/lang/Object."<init>":()V
   4: return
  LineNumberTable:
   line 1: 0
```

# javap -verbose

```
public static void main(java.lang.String[]);
  Code:
   Stack=2, Locals=1, Args_size=1
   0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:  ldc #3; //String Hello, world
   5:  invokevirtual #4; //Method java/io/PrintStream.println:
                                                 (Ljava/lang/String;)V
   8:  return
  LineNumberTable:
   line 3: 0
   line 4: 8


}
```

# Thank you!

# Thank you!

(Just Kidding)

Let’s try something a little easier...

# BiteScript

- (J)Ruby DSL for emitting JVM bytecode
  - Internal DSL
  - Primitive "macro" support
  - Reads like javap -c (but nicer)
- http://github.com/headius/bitescript

# Installation

- Download JRuby from http://jruby.org
- Unpack, optionally add bin/ to PATH
    - Ahead of PATH if you have Ruby already
- [bin/]jruby -S gem install bitescript
- `bite myfile.bs` to run myfile.bs file
- `bitec myfile.bs` to compile myfile.bs file

# BiteScript Users

- Mirah
  - Ruby-like language for writing Java code
  - BiteScript for JVM bytecode backend
- BrainF*ck implementation
- Other miscellaneous bytecode experiments

# javap -c

```
~/projects/bytecode_for_dummies ➔ javap -c HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0:  aload_0
   1:  invokespecial #1; //Method java/lang/Object."<init>":()V
   4:  return

public static void main(java.lang.String[]);
  Code:
   0:  getstatic  #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:  ldc #3; //String Hello, world
   5:  invokevirtual #4; //Method java/io/PrintStream.println:
                                                     (Ljava/lang/String;)V
   8:  return

}
```

# BiteScript

```
main do
  getstatic java.lang.System, "out",
            java.io.PrintStream
  ldc "Hello, world!"
  invokevirtual java.io.PrintStream, "println",
      [java.lang.Void::TYPE, java.lang.Object]
  returnvoid
end
```

# BiteScript

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println", [void, object]
  returnvoid
end
```

JRuby's "import" for Java classes

Shortcuts for void, int, string, object, etc

# BiteScript

```
main do
  ldc "Hello, world!"
  aprintln
  returnvoid
end
```

A BiteScript “macro”

# BiteScript

```
macro :aprintln do
  getstatic System, "out", PrintStream
  swap
  invokevirtual PrintStream, "println",
                [void, object]
end
```

# The Basics

- Stack machine
- Basic operations
- Flow control
- Class structures
- Exception handling

# Stack Machine

- The "operand stack" holds operands
- Operations push and/or pop stack values
  - Exceptions: nop, wide, goto, jsr/ret
- Stack must be consistent
  - Largest part of bytecode verifier
- Stack is explicitly sized per method

# The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                         [void, object]
  returnvoid
end
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                                [void, object]
  returnvoid
end
```

| Depth | Value |
|---|---|
| 0 | **out (a PS)** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                         [void, object]
  returnvoid
end
```

| Depth | Value |
|---|---|
| 0 | **“Hello, world!”** |
| 1 | out (a PS) |
| 2 | |
| 3 | |
| 4 | |

# The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                         [void, object]
  returnvoid
end
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                         [void, object]
  returnvoid
end
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Basic Operations

- Stack manipulation
- Local variables
- Math
- Boolean

# Stack Operations

| | | |
|---|---|---|
| 0x00 | nop | Do nothing. |
| 0x57 | pop | Discard top value from stack |
| 0x58 | pop2 | Discard top two values |
| 0x59 | dup | Duplicate and push top value again |
| 0x5A | dup_x1 | Dup and push top value below second value |
| 0x5B | dup_x2 | Dup and push top value below third value |
| 0x5C | dup2 | Dup top two values and push |
| 0x5D | dup2_x1 | ...below second value |
| 0x5E | dup2_x2 | ...below third value |
| 0x5F | swap | Swap top two values |

# Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

| Depth | Value |
|---|---|
| 0 | value_0 |
| 1 | value_1 |
| 2 | |
| 3 | |
| 4 | |

# Stack Juggling

- dup
- pop
- swap
- dup_x1
- dup2_x2

| Depth | Value |
|---|---|
| 0 | **value_0** |
| 1 | value_0 |
| 2 | value_1 |
| 3 | |
| 4 | |

# Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

| Depth | Value |
|---|---|
| 0 | value_0 |
| 1 | value_1 |
| 2 | |
| 3 | |
| 4 | |

# Stack Juggling

dup

pop

swap

dup_x1

dup2_x2

| Depth | Value |
|---|---|
| 0 | **value_1** |
| 1 | **value_0** |
| 2 | |
| 3 | |
| 4 | |

# Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

| Depth | Value |
|---|---|
| 0 | value_1 |
| 1 | value_0 |
| 2 | **value_1** |
| 3 | |
| 4 | |

# Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

| Depth | Value |
|---|---|
| 0 | value_1 |
| 1 | value_0 |
| 2 | value_1 |
| 3 | **value_1** |
| 4 | **value_0** |

# Typed Opcodes

`<type><operation>`

| | |
|---|---|
| b | byte |
| s | short |
| c | char |
| i | int |
| l | long |
| f | float |
| d | double |
| a | reference |

| |
|---|
| Constant values |
| Local vars (load, store) |
| Array operations (aload, astore) |
| Math ops (add, sub, mul, div) |
| Boolean and bitwise |
| Comparisons |
| Conversions |

# Where's boolean?

- Boolean is generally int 0 or 1
- Boolean operations push int 0 or 1
- Boolean branches expect 0 or nonzero
- To set a boolean...use int 0 or 1

# Constant Values

| | | |
|---|---|---|
| 0x01 | aconst_null | Push null on stack |
| 0x02-0x08 | iload_[m1-5] | Push integer [-1 to 5] on stack |
| 0x09-0x0A | lconst_[0,1] | Push long [0 or 1] on stack |
| 0x0B-0x0D | fconst_[0,1,2] | Push float [0.0, 1.0, 2.0] on stack |
| 0x0E-0x0F | dconst_[0,1] | Push double [0.0, 1.0] on stack |
| 0x10 | bipush | Push byte value to stack as integer |
| 0x11 | sipush | Push short value to stack as integer |
| 0x12 | ldc | Push 32-bit constant to stack (int, float, string) |
| 0x14 | ldc2_w | Push 64-bit constant to stack (long, double) |

# Why So Many?

- Reducing bytecode size
    - Special iconst_0 and friends take no args
    - bipush, sipush: only 8, 16 bits arguments
- Pre-optimizing JVM
    - Specialized instructions can be optimized
    - Doesn't matter at all now

# Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

| Depth | Value |
|---|---|
| 0 | **“hello”** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

| Depth | Value |
|---|---|
| 0 | **1.0d** |
| 1 | |
| 2 | “hello” |
| 3 | |
| 4 | |
| 5 | |

# Woah, Two Slots?

- JVM stack slots (and local vars) are 32-bit
- 64-bit values take up two slots
- “wide” before or “w” suffix
- 64-bit field updates not atomic!
  - Mind those concurrent longs/doubles!

# Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

| Depth | Value |
|---|---|
| 0 | **null** |
| 1 | 1.0d |
| 2 | |
| 3 | “hello” |
| 4 | |
| 5 | |

# Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

| Depth | Value |
|---|---|
| 0 | **4** |
| 1 | null |
| 2 | 1.0d |
| 3 | |
| 4 | “hello” |
| 5 | |

# Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

| Depth | Value |
|---|---|
| 0 | **2.0f** |
| 1 | 4 |
| 2 | null |
| 3 | 1.0 |
| 4 | |
| 5 | “hello” |

# Local Variable Table

- Local variables numbered from 0
  - Instance methods have “this” at 0
- Separate table maps numbers to names
- Explicitly sized in method definition

# Local Variables

| | | |
|---|---|---|
| 0x15 | iload | Load integer from local variable onto stack |
| 0x16 | lload | ...long... |
| 0x17 | fload | ...float... |
| 0x18 | dload | ...double... |
| 0x19 | aload | ...reference... |
| 0x1A-0x2D | Packed loads | iload_0, aload_3, etc |
| 0x36 | istore | Store integer from stack into local variable |
| 0x37 | lstore | ...long... |
| 0x38 | fstore | ...float... |
| 0x39 | dstore | ...double... |
| 0x3A | astore | ...reference... |
| 0x3B-0x4E | Packed stores | fstore_2, dstore_0, etc |
| 0x84 | iinc | Add given amount to int local variable |

# Local Variables

| Var | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | **“hello”** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | **4** |
| 1 | “hello” |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | |
| I | |
| 2 | |
| 3 | **4** |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | “hello” |
| I | |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | 4 |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | **0.0** |
| 1 | |
| 2 | “hello” |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | |
| 1 | **0.0** |
| 2 | |
| 3 | 4 |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | “hello” |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | **“hello”** |
| 1 | 0.0 |
| 2 | |
| 3 | 4 |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | “hello” |
| 1 | 0.0 |
| 2 | |
| 3 | 4 |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | **“hello”** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Local Variables

| Var | Value |
|---|---|
| 0 | “hello” |
| 1 | 0.0 |
| 2 | |
| 3 | **9** |
| 4 | |

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

| Depth | Value |
|---|---|
| 0 | “hello” |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

# Arrays

| | | |
|---|---|---|
| 0x2E-0x35 | [i,l,f,d,a,b,c,d]aload | Load [int, long, ...] from array (on stack) to stack |
| 0x4F-0x56 | [i,l,f,d,a,b,c,d]astore | Store [int, long, ...] from stack to array (on stack) |
| 0xBC | newarray | Construct new primitive array |
| 0xBD | anewarray | Construct new reference array |
| 0xBE | arraylength | Get array length |
| 0xC5 | multianewarray | Create multi-dimensional array |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | **2** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | **int[2] {0,0}** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | **int[2] {0,0}** |
| 1 | int[2] {0,0} |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | **0** |
| 1 | int[2] {0,0} |
| 2 | int[2] {0,0} |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | **-1** |
| 1 | 0 |
| 2 | int[2] {0,0} |
| 3 | int[2] {0,0} |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | int[2] {**-1**, 0} |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | **0** |
| 1 | int[2] {-1, 0} |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

| Depth | Value |
|---|---|
| 0 | -1 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Math Operations

| | add<br>+ | subtract<br>- | multiply<br>* | divide<br>/ | remainder<br>% | negate<br>-() |
|---|---|---|---|---|---|---|
| int | iadd | isub | imul | idiv | irem | ineg |
| long | ladd | lsub | lmul | ldiv | lrem | lneg |
| float | fadd | fsub | fmul | fdiv | frem | fneg |
| double | dadd | dsub | dmul | ddiv | drem | dneg |

# Boolean and Bitwise

| | shift left | shift right | unsigned shift right | and | or | xor |
|---|---|---|---|---|---|---|
| int | ishl | ishr | iushr | iand | ior | ixor |

# Conversions

To:

From:

| | int | long | float | double | byte | char | short |
|---|---|---|---|---|---|---|---|
| int | - | i2l | i2f | i2d | i2b | i2c | i2s |
| long | l2i | - | l2f | l2d | - | - | - |
| float | f2i | f2l | - | f2d | - | - | - |
| double | d2i | d2l | d2f | - | - | - | - |

# Comparisons

| | | |
|---|---|---|
| 0x94 | lcmp | Compare two longs, push int -1, 0, 1 |
| 0x95 | fcmpl | Compare two floats, push in -1, 0, 1 (-1 for NaN) |
| 0x96 | fcmpg | Compare two floats, push in -1, 0, 1 (1 for NaN) |
| 0x97 | dcmpl | Compare two doubles, push in -1, 0, 1 (-1 for NaN) |
| 0x98 | dcmpg | Compare two doubles, push in -1, 0, 1 (1 for NaN) |

# Flow Control

- Inspect stack and branch
  - Or just branch, via goto
- Labels mark branch targets
- Wide variety of tests

# Flow Control

| | | |
|---|---|---|
| 0x99 | ifeq | If zero on stack, branch |
| 0x9A | ifne | If nonzero on stack, branch |
| 0x9B | iflt | If stack value is less than zero, branch |
| 0x9C | ifge | If stack value is greater than or equal to zero, branch |
| 0x9D | ifgt | If stack value is greater than zero, branch |
| 0x9E | ifle | If stack value is less than or equal to zero, branch |
| 0x9F | if icmpeq | If two integers on stack are eq, branch |
| 0xA0 | if icmpne | If two integers on stack are ne, branch |
| 0xA1 | if icmplt | If two integers on stack are lt, branch |
| 0xA2 | if icmpge | If two integers on stack are ge, branch |
| 0xA3 | if icmpgt | If two integers on stack are gt, branch |
| 0xA4 | if icmple | If two integers on stack are le, branch |
| 0xA5 | if acmpeq | If two references on stack are the same, branch |
| 0xA6 | if acmpne | If two references on stack are different, branch |
| 0xA7 | goto | GOTO! |

# Other Flow Control

| | | |
|---|---|---|
| 0xA8 | jsr | Jump to subroutine (deprecated) |
| 0xA9 | ret | Return from subroutine (deprecated) |
| 0xAA | tableswitch | Branch using an indexed table of jump offsets |
| 0xAB | lookupswitch | Branch using a lookup-based table of jump offsets |
| 0xAC-0xB0 | [i,l,f,d,a]return | Return (int, long, float, double, reference) value |
| 0xB1 | return | Void return (exit method, return nothing) |
| 0xC6 | ifnull | If reference on stack is null |
| 0xC7 | ifnonnull | If reference on stack is not null |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **String[]<br>{“branch”}** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | 0 |
| 1 | String[]{“branch”} |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **“branch”** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                      [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **“branch”** |
| 1 | “branch” |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                  [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

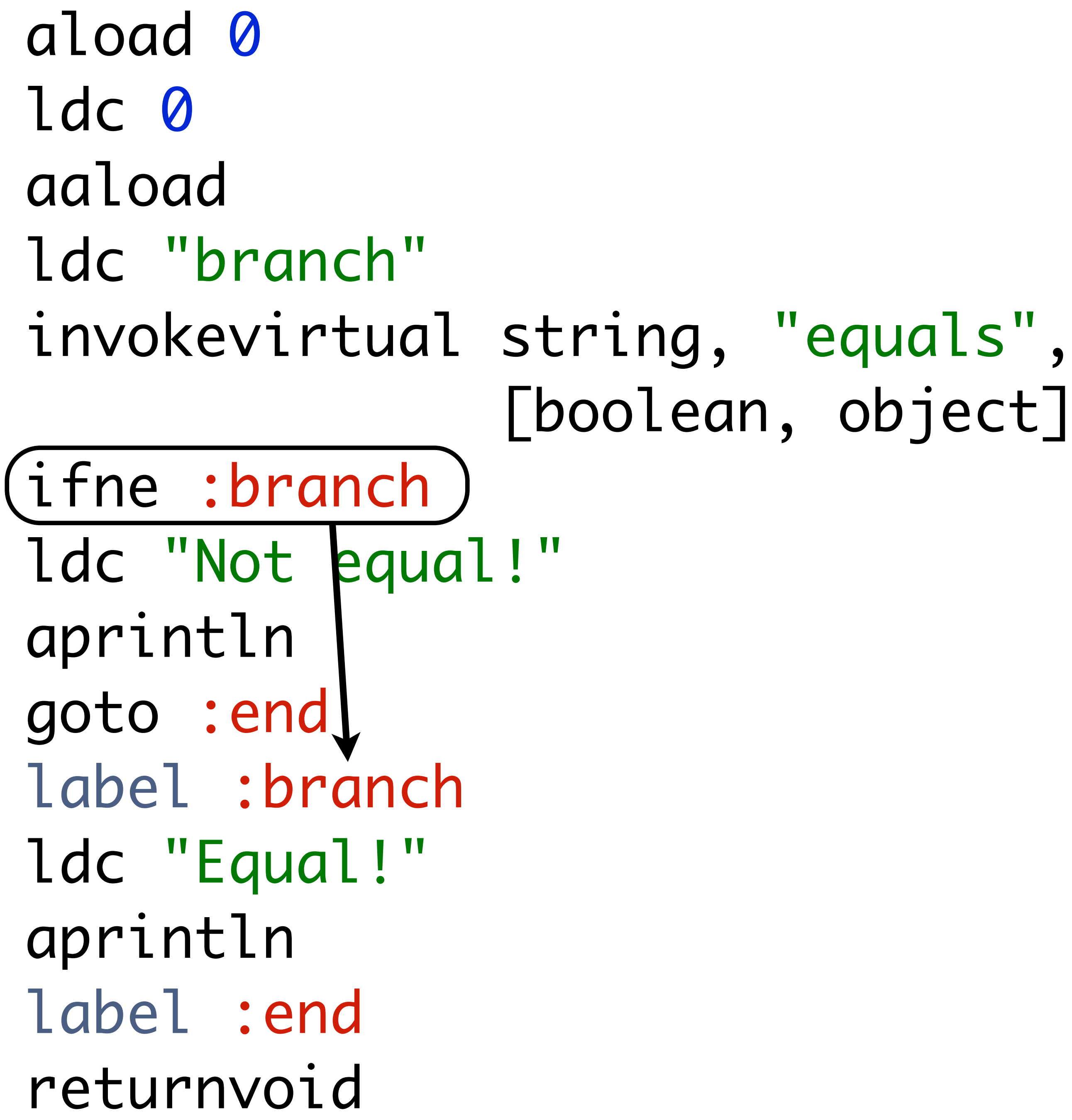
| Depth | Value |
|---|---|
| 0 | 1 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                     [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control



```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                      [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                     [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **“Equal!”** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Classes and Types

- Signatures!!!
  - Probably the most painful part
  - ...but not a big deal if you understand

# Using Classes

| | | |
|---|---|---|
| 0xB2 | getstatic | Fetch static field from class |
| 0xB3 | putstatic | Set static field in class |
| 0xB4 | getfield | Get instance field from object |
| 0xB5 | setfield | Set instance field in object |
| 0xB6 | invokevirtual | Invoke instance method on object |
| 0xB7 | invokespecial | Invoke constructor or "super" on object |
| 0xB8 | invokestatic | Invoke static method on class |
| 0xB9 | invokeinterface | Invoke interface method on object |
| 0xBA | invokedynamic | Invoke method dynamically on object (Java 7) |
| 0xBB | new | Construct new instance of object |
| 0xC0 | checkcast | Attempt to cast object to type |
| 0xC1 | instanceof | Push nonzero if object is instanceof specified type |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **an ArrayList (uninitialized)** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **an ArrayList (uninitialized)** |
| 1 | an ArrayList (uninitialized) |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | an ArrayList |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **a Collection** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **a Collection** |
| 1 | a Collection |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **“first element”** |
| 1 | a Collection |
| 2 | a Collection |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **1 (true)** |
| 1 | a Collection |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | a Collection |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **an ArrayList** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | 0 |
| 1 | an ArrayList |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                  [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | **“first element”** |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

| Depth | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

# Exceptions and Synchronization

| | | |
|---|---|---|
| - | trycatch | Table structure for a method indicating start/end of try/catch and logic to run on exception |
| 0xC2 | monitorenter | Enter synchronized block against object on stack |
| 0xC3 | monitorexit | Exit synchronized block (against same object) |

# More Examples

- A simple loop
- Fibonacci

# A Simple Loop

```
main do
  aload 0
  push_int 0
  aaload
  label :top
  dup
  aprintln
  goto :top
  returnvoid
end
```

# Fibonacci

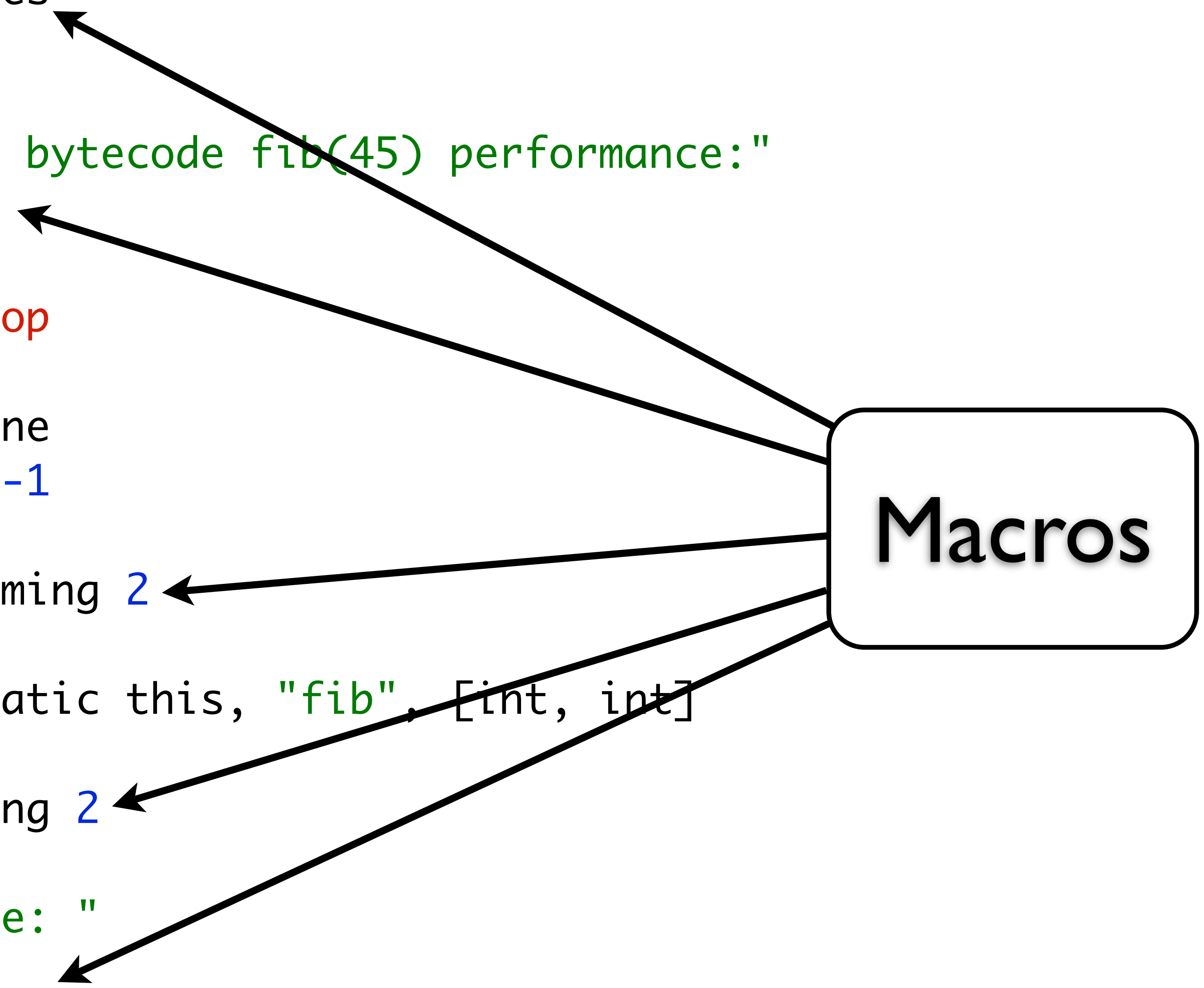
```
public_static_method "fib", [], int, int do
  iload 0
  ldc 2
  if_icmpge :recurse
  iload 0
  ireturn
  label :recurse
  iload 0
  ldc 1
  isub
  invokestatic this, "fib", [int, int]
  iload 0
  ldc 2
  isub
  invokestatic this, "fib", [int, int]
  iadd
  ireturn
end
```

# Fibonacci

```
main do
  load_times
  istore 1

  ldc "Raw bytecode fib(45) performance:"
  aprintln

  label :top
  iload 1
  ifeq :done
  iinc 1, -1

  start_timing 2
  ldc 45
  invokestatic this, "fib", [int, int]
  pop
  end_timing 2

  ldc "Time: "
  aprintln
  lprintln 2
  goto :top

  label :done
  returnvoid
end
```

# Fibonacci

```
main do
  load_times
  istore 1

  ldc "Raw bytecode fib(45) performance:"
  aprintln

  label :top
  iload 1
  ifeq :done
  iinc 1, -1

  start_timing 2
  ldc 45
  invokestatic this, "fib", [int, int]
  pop
  end_timing 2

  ldc "Time: "
  aprintln
  lprintln 2
  goto :top

  label :done
  returnvoid
end
```



Macros

# Fibonacci

```
macro :load_times do
  aload 0
  ldc 0
  aaload # number of times
  invokestatic JInteger, 'parseInt',
               [int, string]
end
```

# Fibonacci

```
macro :start_timing do |i|
  load_time
  lstore i
end
```

# Fibonacci

```
macro :load_time do
  invokestatic System, "currentTimeMillis", long
end
```

# Fibonacci

```
macro :end_timing do |i|
  load_time
  lload i
  lsub
  lstore i
end
```

# Fibonacci

```
macro :lprintln do |i|
  getstatic System, "out", PrintStream
  lload i
  invokevirtual PrintStream, "println",
                [void, long]
end
```

# ASM

- “All purpose bytecode manipulation and analysis framework.”
- De facto standard bytecode library
- http://asm.ow2.org

# Basic Process

- Construct a ClassWriter
- Visit structure
    - Annotations, methods, fields, inner classes
- Write out bytes

# Blah.java

```java
public class Blah implements Cloneable {
    private final String fieldName;

    public Blah() {
        fieldName = "hello";
    }

    public static Blah makeBlah() {
        return new Blah();
    }
}
```

# ClassWriter

```
ClassWriter cv = new ClassWriter(
        ClassWriter.COMPUTE_MAXS |
        ClassWriter.COMPUTE_FRAMES);
```

# COMPUTE...what?

- COMPUTE_MAXS
  - ASM will calculate max stack/local vars
- COMPUTE_FRAMES
  - ASM will calculate Java 6 stack map
  - Hints to verifier that we've pre-validated stack contents (sort of)

# Visit Class

```
cv.visit(
        Opcodes.V1_6,
        Opcodes.ACC_PUBLIC,
        "Blah",
        null,
        "java/lang/Object",
        new String[] {"java/lang/Cloneable"});
```

# Opcodes

- Interface full of constants
  - Bytecodes
  - Visibility modifiers
  - Java versions
  - Other stuff

# ACC_*

- Some you know
  - ACC_PUBLIC, ACC_ABSTRACT, etc
- Some you don't
  - ACC_BRIDGE, ACC_SYNTHETIC

# Java Version

- V1_1 through V1_7
    - Sorry 1.0!

# Class Names

```
"java/lang/Object"
```

```
packageClass.replaceAll('.', '/')
```

# Visit Source

```
cv.visitSource(
        "Blah.java",
        "JSR-45 source map here");
```

# Visit Annotation

```
AnnotationVisitor av = cv.visitAnnotation("some/Annotation", true);
av.visitArray("name1", ...);
av.visitEnum("name2", ...);
av.visitEnd();
```

# Blah.java

```java
private final String fieldName;
```

# Visit Field

```
FieldVisitor fv = cv.visitField(
        Opcodes.ACC_PRIVATE | Opcodes.ACC_FINAL,
        "fieldName",
        "Ljava.lang.String;",
        null);
fv.visitAnnotation(...);
fv.visitAttribute(...);
fv.visitEnd();
```

# Descriptor

```
"Ljava.lang.String;"
"(IF[JLjava.lang.Object;)V"
```

- Primitive types
  - B,C,S,I,J,F,D,Z,V
- Reference types
  - Lsome/Class;
- Array
  - Prefix with [

# Blah.java

```
public Blah() {
    ...
}

public static Blah makeBlah() {
    ...
}
```

# Visit Method

```
MethodVisitor construct = cv.visitMethod(
        Opcodes.ACC_PUBLIC,
        "<init>",
        "()V",
        null,
        null);
MethodVisitor makeBlah = cv.visitMethod(
        Opcodes.ACC_PUBLIC, ACC_STATIC,
        "makeBlah",
        "()LBlah;",
        null,
        null);
```

# Special Methods

- <init>
  - Constructor
- <clinit>
  - Static initializer

# MethodVisitor

- Visit annotation stuff
- Visit code
  - Bytecodes, frames, local vars, line nums
- Visit maxs
  - Pass bogus values if COMPUTE_MAXS

# Blah.java

```
public Blah() {
    fieldName = "hello";
}

public static Blah makeBlah() {
    return new Blah();
}
```

# Visit Method Body

```
construct.visitCode();
construct.visitVarInsn(ALOAD, 0);
construct.visitMethodInsn(INVOKESPECIAL,
        "java/lang/Object",
        "<init>",
        "()V");
construct.visitVarInsn(ALOAD, 0);
construct.visitLdcInsn("hello");
construct.visitFieldInsn(PUTFIELD,
        "Blah",
        "fieldName",
        "Ljava/lang/String;");
construct.visitInsn(RETURN);
construct.visitMaxs(2, 1);
construct.visitEnd();
```

# ASMifierClassVisitor

- Dump ASM visitor calls from .class file
- Very raw, but very useful

# Blah.java

```java
public class Blah implements Cloneable {
    private final String fieldName;

    public Blah() {
        fieldName = "hello";
    }

    public static Blah makeBlah() {
        return new Blah();
    }
}
```

```
~/oscon ➔ java -cp asm-3.3.1.jar:asm-util-3.3.1.jar \
                org.objectweb.asm.util.ASMifierClassVisitor \
                Blah.class
import java.util.*;
import org.objectweb.asm.*;
import org.objectweb.asm.attrs.*;
public class BlahDump implements Opcodes {

public static byte[] dump () throws Exception {

ClassWriter cw = new ClassWriter(0);
FieldVisitor fv;
MethodVisitor mv;
AnnotationVisitor av0;

cw.visit(V1_6, ACC_PUBLIC + ACC_SUPER, "Blah", null, "java/lang/Object", new String[] { "java/lang/Cloneable" });

{
fv = cw.visitField(ACC_PRIVATE + ACC_FINAL, "fieldName", "Ljava/lang/String;", null, null);
fv.visitEnd();
}
{
mv = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null);
mv.visitCode();
mv.visitVarInsn(ALOAD, 0);
mv.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V");
mv.visitVarInsn(ALOAD, 0);
mv.visitLdcInsn("hello");
mv.visitFieldInsn(PUTFIELD, "Blah", "fieldName", "Ljava/lang/String;");
mv.visitInsn(RETURN);
mv.visitMaxs(2, 1);
mv.visitEnd();
}
{
mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "makeBlah", "()LBlah;", null, null);
mv.visitCode();
mv.visitTypeInsn(NEW, "Blah");
mv.visitInsn(DUP);
mv.visitMethodInsn(INVOKESPECIAL, "Blah", "<init>", "()V");
mv.visitInsn(ARETURN);
mv.visitMaxs(2, 0);
mv.visitEnd();
}
cw.visitEnd();

return cw.toByteArray();
}
}
```

# Real-world Cases

- Reflection-free invocation
  - JRuby, Groovy, other languages
- Bytecoded data objects
  - Hibernate, other data layers
  - java.lang.reflect.Proxy and others
- Language compilers

# Part Two:
# JVM JIT

# JIT

- Just-In-Time compilation
- Compiled when needed
  - Maybe immediately before execution
  - ...or when we decide it's important
  - ...or never?

# Mixed-Mode

- Interpreted
  - Bytecode-walking
  - Artificial stack
- Compiled
  - Direct native operations
  - Native registers, memory, etc

# Profiling

- Gather data about code while interpreting
  - Invariants (types, constants, nulls)
  - Statistics (branches, calls)
- Use that information to optimize
  - Educated guess?

# Optimization

- Loop unrolling
- Lock coarsening
- Method inlining
- Dead code elimination
- Duplicate code elimination

# The Golden Rule of Optimization

Don’t do unnecessary work.

# Perf Sinks

- Memory accesses
  - By far the biggest expense
- Calls
  - Opaque memory ref + branch
- Locks, volatile writes
  - Kills multi-cpu perf

# Volatile?

- Each CPU maintains a memory cache
- Caches may be out of sync
  - If it doesn’t matter, no problem
  - If it does matter, threads disagree!
- Volatile forces synchronization of cache
  - Across cores and to main memory

# Inlining?

- Combine caller and callee into one unit
    - e.g. based on profile
    - Perhaps with a sanity check
- Optimize as a whole

# Inlining

```
int addAll(int max) {
    int accum = 0;
    for (int i = 0; i < max; i++) {
        accum = add(accum, i);
    }
    return accum;
}

int add(int a, int b) {
    return a + b;
}
```

# Inlining

```
int addAll(int max) {
    int accum = 0;
    for (int i = 0; i < max; i++) {
        accum = add(accum, i);
    }
    return accum;
}

int add(int a, int b) {
    return a + b;
}
```

Only one target is ever seen

# Inlining

```
int addAll(int max) {
    int accum = 0;
    for (int i = 0; i < max; i++) {
        accum = accum + i;
    }
    return accum;
}
```

Don’t bother making a call

# Call Site

- The place where you make a call
- Monomorphic (“one shape”)
    - Single target class
- Bimorphic (“two shapes”)
- Polymorphic (“many shapes”)
- Megamorphic (“you’re screwed”)

# Blah.java

```java
System.currentTimeMillis(); // static, monomorphic

List list1 = new ArrayList(); // constructor, monomorphic
List list2 = new LinkedList();

for (List list : new List[]{ list1, list2 }) {
    list.add("hello"); // bimorphic
}

for (Object obj : new Object[]{ 'foo', list1, new Object() }) {
    obj.toString(); // polymorphic
}
```

# Hotspot

- -client mode (C1) inlines, less aggressive
  - Fewer opportunities to optimize
- -server mode (C2) profiles, inlines
  - We'll focus on this
- Tiered mode combines them
  - -XX:+TieredCompilation

# Hotspot Inlining

- Profile to find “hot spots”
  - Largely focused around call sites
  - Profile until 10k calls
- Inline mono/bimorphic calls
- Other mechanisms for polymorphic calls

Now it gets fun!

# Monitoring the JIT

- Dozens of flags
  - PrintCompilation
  - PrintInlining
  - LogCompilation
  - PrintAssembly
- Some in product, some in debug...

```java
public class Accumulator {
    public static void main(String[] args) {
        int max = Integer.parseInt(args[0]);
        System.out.println(addAll(max));
    }

    static int addAll(int max) {
        int accum = 0;
        for (int i = 0; i < max; i++) {
            accum = add(accum, i);
        }
        return accum;
    }

    static int add(int a, int b) {
        return a + b;
    }
}
```

```
~/oscon ➔ java -version
openjdk version "1.7.0-internal"
OpenJDK Runtime Environment (build 1.7.0-internal-b00)
OpenJDK 64-Bit Server VM (build 21.0-b17, mixed mode)

~/oscon ➔ javac Accumulator.java

~/oscon ➔ java Accumulator 1000
499500
```

# PrintCompilation

- -XX:+PrintCompilation
- Print methods as they are jitted
  - Class + name + size

```
~/oscon ➔ java -XX:+PrintCompilation Accumulator 1000
  1       java.lang.String::hashCode (64 bytes)
  2       java.math.BigInteger::mulAdd (81 bytes)
  3       java.math.BigInteger::multiplyToLen (219 bytes)
  4       java.math.BigInteger::addOne (77 bytes)
  5       java.math.BigInteger::squareToLen (172 bytes)
  6       java.math.BigInteger::primitiveLeftShift (79 bytes)
  7       java.math.BigInteger::montReduce (99 bytes)
  8       sun.security.provider.SHA::implCompress (491 bytes)
  9       java.lang.String::charAt (33 bytes)
499500
```

```
~/oscon ➔ java -XX:+PrintCompilation Accumulator 1000
  1       java.lang.String::hashCode (64 bytes)
  2       java.math.BigInteger::mulAdd (81 bytes)
  3       java.math.BigInteger::multiplyToLen (219 bytes)
  4       java.math.BigInteger::addOne (77 bytes)
  5       java.math.BigInteger::squareToLen (172 bytes)
  6       java.math.BigInteger::primitiveLeftShift (79 bytes)
  7       java.math.BigInteger::montReduce (99 bytes)
  8       sun.security.provider.SHA::implCompress (491 bytes)
  9       java.lang.String::charAt (33 bytes)
499500
```

Where’s our methods?!

```
~/oscon ➔ java -XX:+PrintCompilation Accumulator 1000
  1       java.lang.String::hashCode (64 bytes)
  2       java.math.BigInteger::mulAdd (81 bytes)
  3       java.math.BigInteger::multiplyToLen (219 bytes)
  4       java.math.BigInteger::addOne (77 bytes)
  5       java.math.BigInteger::squareToLen (172 bytes)
  6       java.math.BigInteger::primitiveLeftShift (79 bytes)
  7       java.math.BigInteger::montReduce (99 bytes)
  8       sun.security.provider.SHA::implCompress (491 bytes)
  9       java.lang.String::charAt (33 bytes)
499500
```

Where's our methods?!

...remember...10k calls

10k loop, 10k calls to add

```
~/oscon ➔ java -XX:+PrintCompilation Accumulator 10000
  1       java.lang.String::hashCode (64 bytes)
  2       java.math.BigInteger::mulAdd (81 bytes)
  3       java.math.BigInteger::multiplyToLen (219 bytes)
  4       java.math.BigInteger::addOne (77 bytes)
  5       java.math.BigInteger::squareToLen (172 bytes)
  6       java.math.BigInteger::primitiveLeftShift (79 bytes)
  7       java.math.BigInteger::montReduce (99 bytes)
  8       sun.security.provider.SHA::implCompress (491 bytes)
  9       java.lang.String::charAt (33 bytes)
 10       Accumulator::add (4 bytes)
49995000
```

Hooray!

# What's this stuff?

```
1       java.lang.String::hashCode (64 bytes)
2       java.math.BigInteger::mulAdd (81 bytes)
3       java.math.BigInteger::multiplyToLen (219 bytes)
4       java.math.BigInteger::addOne (77 bytes)
5       java.math.BigInteger::squareToLen (172 bytes)
6       java.math.BigInteger::primitiveLeftShift (79 bytes)
7       java.math.BigInteger::montReduce (99 bytes)
8       sun.security.provider.SHA::implCompress (491 bytes)
9       java.lang.String::charAt (33 bytes)
```

# What’s this stuff?

```
1       java.lang.String::hashCode (64 bytes)
2       java.math.BigInteger::mulAdd (81 bytes)
3       java.math.BigInteger::multiplyToLen (219 bytes)
4       java.math.BigInteger::addOne (77 bytes)
5       java.math.BigInteger::squareToLen (172 bytes)
6       java.math.BigInteger::primitiveLeftShift (79 bytes)
7       java.math.BigInteger::montReduce (99 bytes)
8       sun.security.provider.SHA::implCompress (491 bytes)
9       java.lang.String::charAt (33 bytes)
```

Class loading, security, other boot logic.

# What if you see this...

```
~/oscon ➔ java -client -XX:+PrintCompilation Accumulator 1000
  1       java.lang.String::hashCode (64 bytes)
  2       java.math.BigInteger::mulAdd (81 bytes)
  3       java.math.BigInteger::multiplyToLen (219 bytes)
  4       java.math.BigInteger::addOne (77 bytes)
  5       java.math.BigInteger::squareToLen (172 bytes)
  5   made not entrant  java.math.BigInteger::squareToLen (172 bytes)
  7       java.math.BigInteger::montReduce (99 bytes)
  6       java.math.BigInteger::primitiveLeftShift (79 bytes)
  8       java.math.BigInteger::squareToLen (172 bytes)
  9       sun.security.provider.SHA::implCompress (491 bytes)
 10       java.lang.String::charAt (33 bytes)
499500
```

# Optimistic Compiler

- Assume profile is accurate
- Aggressively optimize based on profile
- Bail out if we’re wrong
    - And hope that we’re usually right

# Deoptimization

- Bail out of running code
- Monitoring flags describe process
  - “uncommon trap” - we were wrong
  - “not entrant” - don’t let anyone enter
  - “zombie” - on its way to deadness

# What if you see this...

```
 20       java.math.BigInteger::addOne (77 bytes)
 21       java.math.BigInteger::squareToLen (172 bytes)
 22       java.math.BigInteger::primitiveLeftShift (79 bytes)
---   n   java.lang.System::arraycopy (static)
 24       sun.security.provider.SHA::implCompress (491 bytes)
 23       java.math.BigInteger::montReduce (99 bytes)
 25       java.lang.String$CaseInsensitiveComparator::compare (115
bytes)
 26       java.lang.Character::toLowerCase (162 bytes)
```

# What if you see this...

```
 20       java.math.BigInteger::addOne (77 bytes)
 21       java.math.BigInteger::squareToLen (172 bytes)
 22       java.math.BigInteger::primitiveLeftShift (79 bytes)
---   n   java.lang.System::arraycopy (static)
 24       sun.security.provider.SHA::implCompress (491 bytes)
 23       java.math.BigInteger::montReduce (99 bytes)
 25       java.lang.String$CaseInsensitiveComparator::compare (115
bytes)
 26       java.lang.Character::toLowerCase (162 bytes)
```

Native calls don't compile, may be intrinsic. We'll come back to that.

# PrintInlining

- -XX:+UnlockDiagnosticVMOptions
  -XX:+PrintInlining
- Display hierarchy of inlined methods
- Include reasons for not inlining
- More, better output on OpenJDK 7

```
~/oscon ➔ java -XX:+UnlockDiagnosticVMOptions \
>              -XX:+PrintInlining \
>              Accumulator 10000
49995000
```

```
~/oscon ➔ java -XX:+UnlockDiagnosticVMOptions \
>              -XX:+PrintInlining \
>              Accumulator 10000
49995000
```

Um...I don’t see anything inlining...

```java
public class Accumulator {
    public static void main(String[] args) {
        int max = Integer.parseInt(args[0]);
        System.out.println(addAll(max));
    }

    static int addAll(int max) {
        int accum = 0;
        for (int i = 0; i < max; i++) {
            accum = add(accum, i);
        }
        return accum;
    }

    static int add(int a, int b) {
        return a + b;
    }
}
```

Called once

Called 10k times...

...but no calls to inline!

```java
public class Accumulator2 {
    public static void main(String[] args) {
        int max = Integer.parseInt(args[0]);
        System.out.println(
            new Accumulator2().addAllSqrts(max));
    }

    double addAllSqrts(int max) {
        double accum = 0;
        for (int i = 0; i < max; i++) {
            accum = addSqrt(accum, i);
        }
        return accum;
    }

    double addSqrt(double a, int b) {
        return a + sqrt(b);
    }

    double sqrt(int b) {
        return Math.sqrt(b);
    }
}
```

```
~/oscon ➔ java -XX:+UnlockDiagnosticVMOptions \
>              -XX:+PrintCompilation \
>              -XX:+PrintInlining \
>              Accumulator2 10000
...
     89    2             Accumulator2::addSqrt (8 bytes)
                            @ 3   Accumulator2::sqrt (6 bytes)   inline (hot)
                              @ 2   java.lang.Math::sqrt (5 bytes)   (intrinsic)
     89    3             Accumulator2::sqrt (6 bytes)
                            @ 2   java.lang.Math::sqrt (5 bytes)   (intrinsic)
666616.4591971082
```

A hot spot!

Calls treated specially by JIT

# Intrinsic?

- Known to the JIT
  - Don’t inline the bytecode
  - Do perform a specific native operation
    - e.g. kernel-level memory operation
    - e.g. optimized sqrt in machine code

# Did Someone Say MACHINE CODE?

# The Red Pill

- Knowing code compiles is good
- Knowing code inlines is better
- Seeing the actual assembly is best!

# Caveat

- I don’t really know assembly.
- But I fake it really well.

# PrintAssembly

- -XX:+PrintAssembly
- Google “hotspot printassembly”
- http://wikis.sun.com/display/HotSpotInternals/PrintAssembly
- Assembly dumping plugins

```
~/oscon ➔ java -XX:+UnlockDiagnosticVMOptions \
>              -XX:+PrintAssembly \
>              Accumulator 10000
OpenJDK 64-Bit Server VM warning: PrintAssembly is enabled;
turning on DebugNonSafepoints to gain additional output
Loaded disassembler from hsdis-amd64.dylib
...
```

```
Decoding compiled method 11343cbd0:
Code:
[Disassembling for mach='i386:x86-64']
[Entry Point]
[Verified Entry Point]
[Constants]
  # {method} 'add' '(II)I' in 'Accumulator'
  # parm0:    rsi       = int
  # parm1:    rdx       = int
  #           [sp+0x20]  (sp of caller)
  11343cd00: push   %rbp
  11343cd01: sub    $0x10,%rsp
  11343cd05: nop                                ;*synchronization entry
                                                ; - Accumulator::add@-1 (line 16)
  11343cd06: mov    %esi,%eax
  11343cd08: add    %edx,%eax                   ;*iadd
                                                ; - Accumulator::add@2 (line 16)
  11343cd0a: add    $0x10,%rsp
  11343cd0e: pop    %rbp
  11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                                ;   {poll_return}
  11343cd15: retq
```

Woah there, buddy.

# x86_64 Assembly 101

| | |
|---|---|
| add | Two's complement add |
| sub | ...subtract |
| mov* | Move data from a to b |
| jmp | goto |
| je, jne, jl, jge, ... | Jump if ==, !=, <, >=, ... |
| push, pop | Push/pop to/from call stack |
| call*, ret* | Call or return from subroutine |
| eax, ebx, esi, ... | Registers |
| rdx, rbx, rsi, ... | 64-bit registers |

# Register Machine

- Instead of stack moves, we have "slots"
- Move data into slots
- Call operations that work with slots
- Get new data out of slots
- JVM stack, locals end up as register ops

# Stack?

- Native code has a stack too
  - Maintains context from call to call
- Calling conventions
  - Caller preserves registers?
  - Callee preserves registers?
  - Many different styles

```
Decoding compiled method 11343cbd0:          <= address of new compiled code
Code:
[Disassembling for mach='i386:x86-64']       <= architecture
[Entry Point]
[Verified Entry Point]
[Constants]
  # {method} 'add' '(II)I' in 'Accumulator'  <= method, signature, class
  # parm0:    rsi       = int                <= first parm to method goes in rsi
  # parm1:    rdx       = int                <= second parm goes in rdx
  #           [sp+0x20]  (sp of caller)      <= caller’s pointer into native stack
```

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                ;*synchronization entry
                                              ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                   ;*iadd
                                              ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                              ;   {poll_return}
11343cd15: retq
```

rbp points at current stack frame, so we save it off.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                ;*synchronization entry
                                              ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                   ;*iadd
                                              ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                              ;   {poll_return}
11343cd15: retq
```

Two args, so we bump stack pointer by 0x10.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                  ;*synchronization entry
                                                ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                     ;*iadd
                                                ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)          # 1003fd000
                                                ;   {poll_return}
11343cd15: retq
```

Do nothing, e.g. to memory-align code.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                ;*synchronization entry
                                              ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                   ;*iadd
                                              ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                              ;   {poll_return}
11343cd15: retq
```

## At the "-1" instruction of our add() method... i.e. here we go!

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                  ;*synchronization entry
                                                ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                     ;*iadd
                                                ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)          # 1003fd000
                                                ;   {poll_return}
11343cd15: retq
```

Move parm1 into eax.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                  ;*synchronization entry
                                                ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                     ;*iadd
                                                ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)          # 1003fd000
                                                ;   {poll_return}
11343cd15: retq
```

Add parm0 and parm1, store result in eax.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                               ;*synchronization entry
                                             ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                  ;*iadd
                                             ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)       # 1003fd000
                                             ;   {poll_return}
11343cd15: retq
```

How nice, Hotspot shows us this is our “iadd” op!

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                               ;*synchronization entry
                                             ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                  ;*iadd
                                             ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)       # 1003fd000
                                             ;   {poll_return}
11343cd15: retq
```

Put stack pointer back where it was.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                ;*synchronization entry
                                              ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                   ;*iadd
                                              ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                              ;   {poll_return}
11343cd15: retq
```

Restore rbp from stack.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                ;*synchronization entry
                                              ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                   ;*iadd
                                              ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                              ;   {poll_return}
11343cd15: retq
```

Poll a “safepoint”...give JVM a chance to GC, etc.

```
11343cd00: push   %rbp
11343cd01: sub    $0x10,%rsp
11343cd05: nop                                ;*synchronization entry
                                              ; - Accumulator::add@-1 (line 16)
11343cd06: mov    %esi,%eax
11343cd08: add    %edx,%eax                   ;*iadd
                                              ; - Accumulator::add@2 (line 16)
11343cd0a: add    $0x10,%rsp
11343cd0e: pop    %rbp
11343cd0f: test   %eax,-0x1303fd15(%rip)        # 1003fd000
                                              ;   {poll_return}
11343cd15: retq
```

All done!

# Things to Watch For

- CALL operations
  - Indicate something failed to inline
- LOCK operations
  - Cache-busting, e.g. volatility

# CALL

```
1134858f5: xchg   %ax,%ax
1134858f7: callq  113414aa0     ; OopMap{off=316}
                                ;*invokespecial addAsBignum
                                ; - org.jruby.RubyFixnum::addFixnum@29 (line 348)
                                ;   {optimized virtual_call}
1134858fc: jmpq   11348586d
```

Ruby integer adds might overflow into Bignum, leading to addAsBignum call. In this case, it’s never called, so Hotspot emits callq assuming we won’t hit it.

# LOCK

## Code from a RubyBasicObject’s default constructor.

```
11345d823: mov    0x70(%r8),%r9d    ;*getstatic NULL_OBJECT_ARRAY
                                    ; - org.jruby.RubyBasicObject::<init>@5 (line 76)
                                    ; - org.jruby.RubyObject::<init>@2 (line 118)
                                    ; - org.jruby.RubyNumeric::<init>@2 (line 111)
                                    ; - org.jruby.RubyInteger::<init>@2 (line 95)
                                    ; - org.jruby.RubyFixnum::<init>@5 (line 112)
                                    ; - org.jruby.RubyFixnum::newFixnum@25 (line 173)
11345d827: mov    %r9d,0x14(%rax)
11345d82b: lock addl $0x0,(%rsp)    ;*putfield varTable
                                    ; - org.jruby.RubyBasicObject::<init>@8 (line 76)
                                    ; - org.jruby.RubyObject::<init>@2 (line 118)
                                    ; - org.jruby.RubyNumeric::<init>@2 (line 111)
                                    ; - org.jruby.RubyInteger::<init>@2 (line 95)
                                    ; - org.jruby.RubyFixnum::<init>@5 (line 112)
                                    ; - org.jruby.RubyFixnum::newFixnum@25 (line 173)
```

Why are we doing a volatile write in the constructor?

# LOCK

```
public class RubyBasicObject ... {
    private static final boolean DEBUG = false;
    private static final Object[] NULL_OBJECT_ARRAY = new Object[0];

    // The class of this object
    protected transient RubyClass metaClass;

    // zeroed by jvm
    protected int flags;

    // variable table, lazily allocated as needed (if needed)
    private volatile Object[] varTable = NULL_OBJECT_ARRAY;
```

Maybe it's not such a good idea to pre-init a volatile?

# LOCK

```
~/projects/jruby ➔ git log 2f935de1e40bfd8b29b3a74eaed699e519571046 -1 | cat
commit 2f935de1e40bfd8b29b3a74eaed699e519571046
Author: Charles Oliver Nutter <headius@headius.com>
Date:   Tue Jun 14 02:59:41 2011 -0500

    Do not eagerly initialize volatile varTable field in RubyBasicObject;
speeds object creation significantly.
```

# LEVEL UP!

# What have we learned?

- How to emit bytecode
- How to read bytecode
- How bytecode execution works
- How to monitor the Hotspot JIT
- How to find problems from asm code

You’re no dummy now.

;-)

# Thank you!

- headius@headius.com, @headius
- http://blog.headius.com
- http://github.com/headius/bitescript
- “java virtual machine specification”
- “jvm opcodes”