

Understanding HotSpot JVM Performance with JITWatch

Chris Newland, LJC (Docklands) 2015-11-10

License: Creative Commons-Attribution-ShareAlike 3.0

Bio

Chris Newland

ADVFN / Throgmorton Street Capital

Market data / FIX / trading systems

@chriswhocodes on Twitter

Play along at home!

```
git clone \
https://github.com/AdoptOpenJDK/jitwatch.git
```

```
mvn clean install exec:java
```



Caveat Emptor



Measure, don't guess

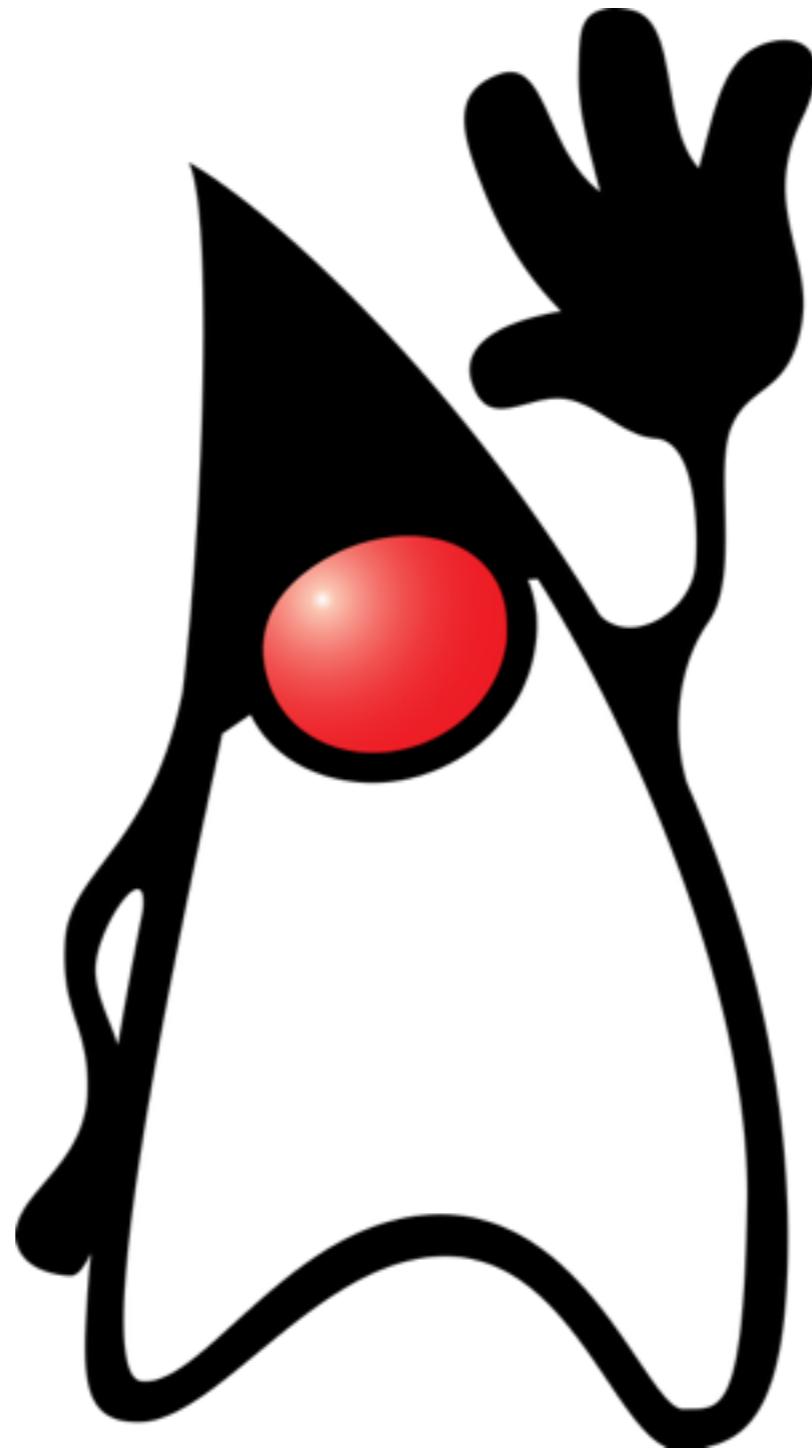
Stay SOLID

90%+ does not need optimising!

Premature optimization is the root of all evil

Donald Knuth

The Amazing JVM



50+ Languages

Java, Scala, Groovy, Clojure, JS, JRuby, Kotlin, ...

Object-Oriented and Functional

Strongly and Dynamically Typed



On Many Platforms

x86, ARM, SPARC, PPC, MIPS, Vega, ...

Linux, Windows, Solaris, OSX, AIX, ...

Single CPU to 1000s of cores

Tiny SoC to TeraBytes of RAM



How does the JVM support such diversity?

Abstraction!



*All problems in computer science can be solved by another level of indirection,
except of course for the problem of too many indirections.*

David Wheeler

High Level Language (Java)



Source Compiler (javac)



Bytecode



Bytecode Interpreter (JVM)



Platform (OS and Hardware)

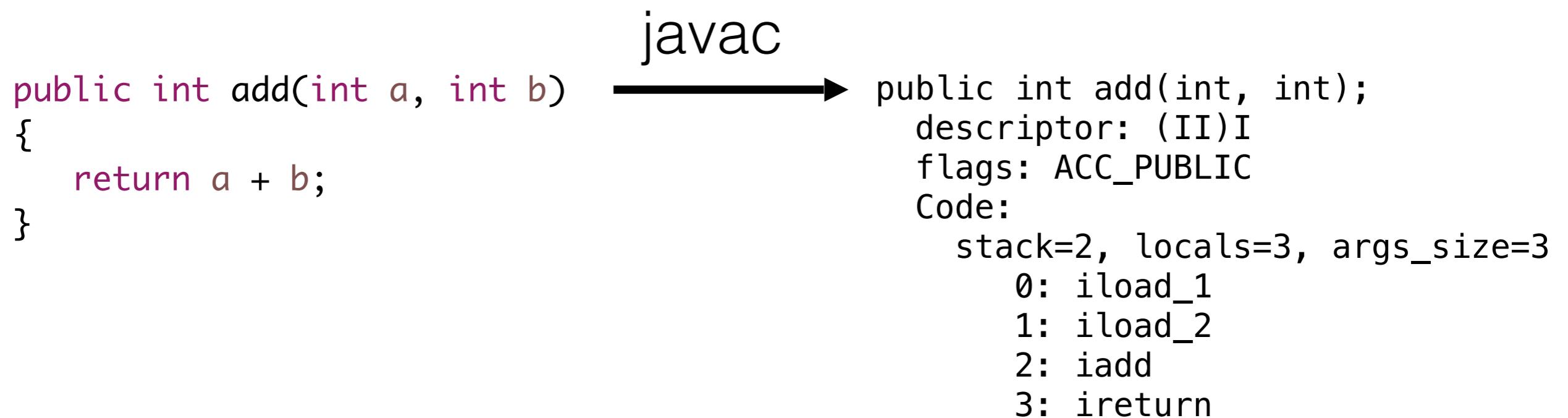
Bytecode

Portable instruction set

All opcodes represented in 1 byte

Executed on a **virtual stack machine**

javac compiler



Virtual Stack Machine?

```
while (running)
{
    opcode = getNextOpcode();

    switch(opcode)
    {
        case 00:
            // stuff
            break;
        case 01:
            // stuff
            break;

        ...
        case ff:
            // stuff
            break;
    }
}
```



Types of Compiler

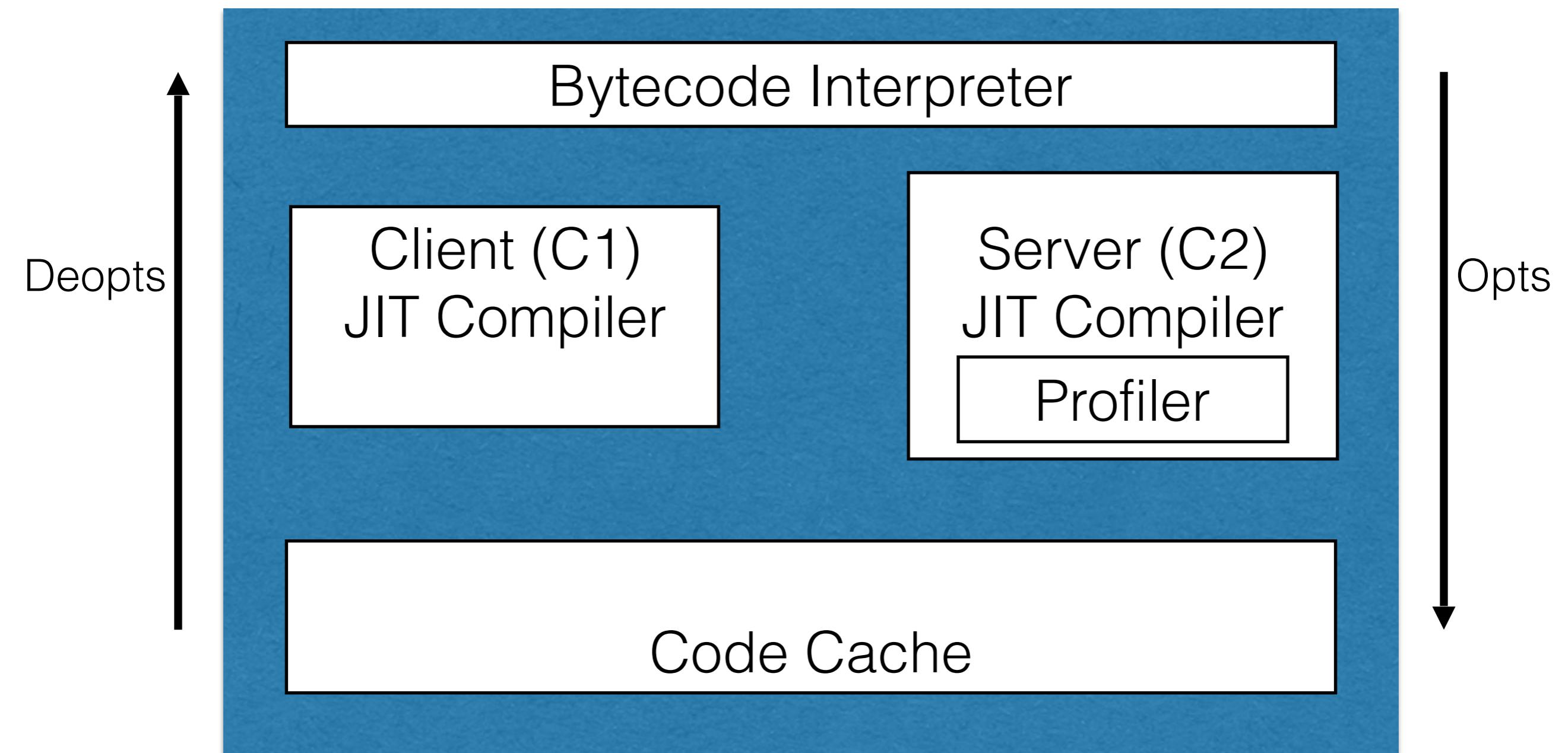
Ahead of Time (AOT)

- Produces native code
- Knowledge of target architecture
- Full performance from the start

Just In Time (JIT)

- Profiles running code
- Adaptive optimisations
- Takes time to build a profile

The HotSpot JVM



Talking JIT

Client compiler (C1)

- Starts quickly, simple optimisations

Server compiler (C2)

- Profile guided, waits for more information
- Advanced optimisations

Tiered Compilation (C1 + C2)

- Default in Java 8
- Enable in Java 7 with **-XX:+TieredCompilation**

Guess Again?

Most server (C2) optimisations are speculative

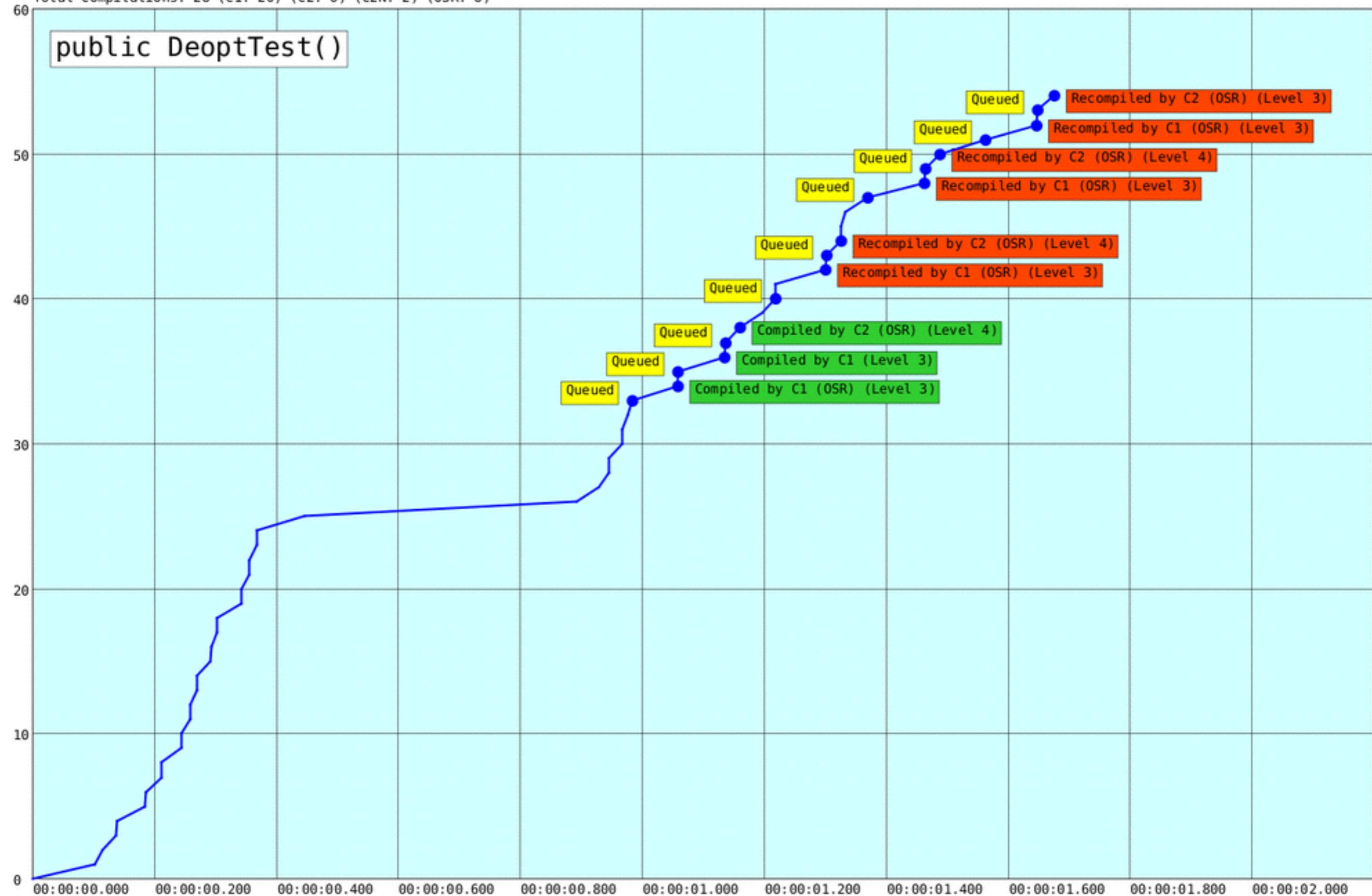
JVM needs a way back if decision was wrong

Uncommon traps verify if assumption holds

Wrong? Switch back to interpreted code

Compilations Timeline

Total Compilations: 28 (C1: 20) (C2: 6) (C2N: 2) (OSR: 8)



Repeated deopts can cause poor performance

The Code Cache

JVM region for JIT-compiled methods

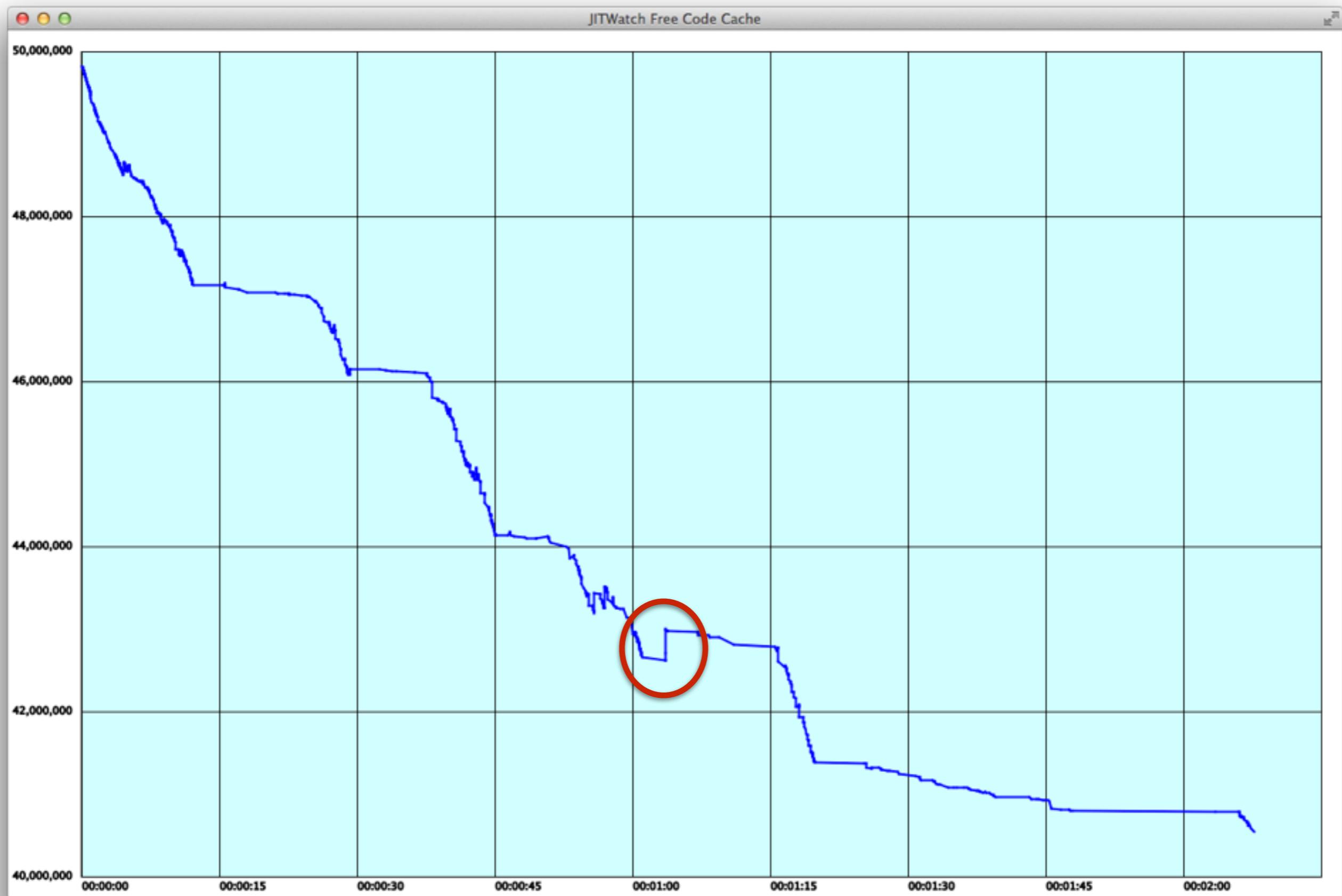
Can run out of space! (*TieredCompilation?*)

Can get fragmented!

Removals (deopts / class unloading)

-XX:ReservedCodeCacheSize=<size>m

Code Cache



I Know Kung-Fu!

HotSpot JIT compilers know ~100 optimisations

lock coarsening

strength reduction

loop unrolling

branch prediction

range check elimination

inlining

dead code elimination

CHA

compiler intrinsics

autobox elimination

copy removal

switch balancing

lock elision

null check elimination

instruction peepholing

devirtualisation

constant propagation

escape analysis

vectorisation

algebraic simplification

register allocation

subexpression elimination

Show Me

-XX:+UnlockDiagnosticVMOptions

-XX:+LogCompilation

-XX:+TraceClassLoading

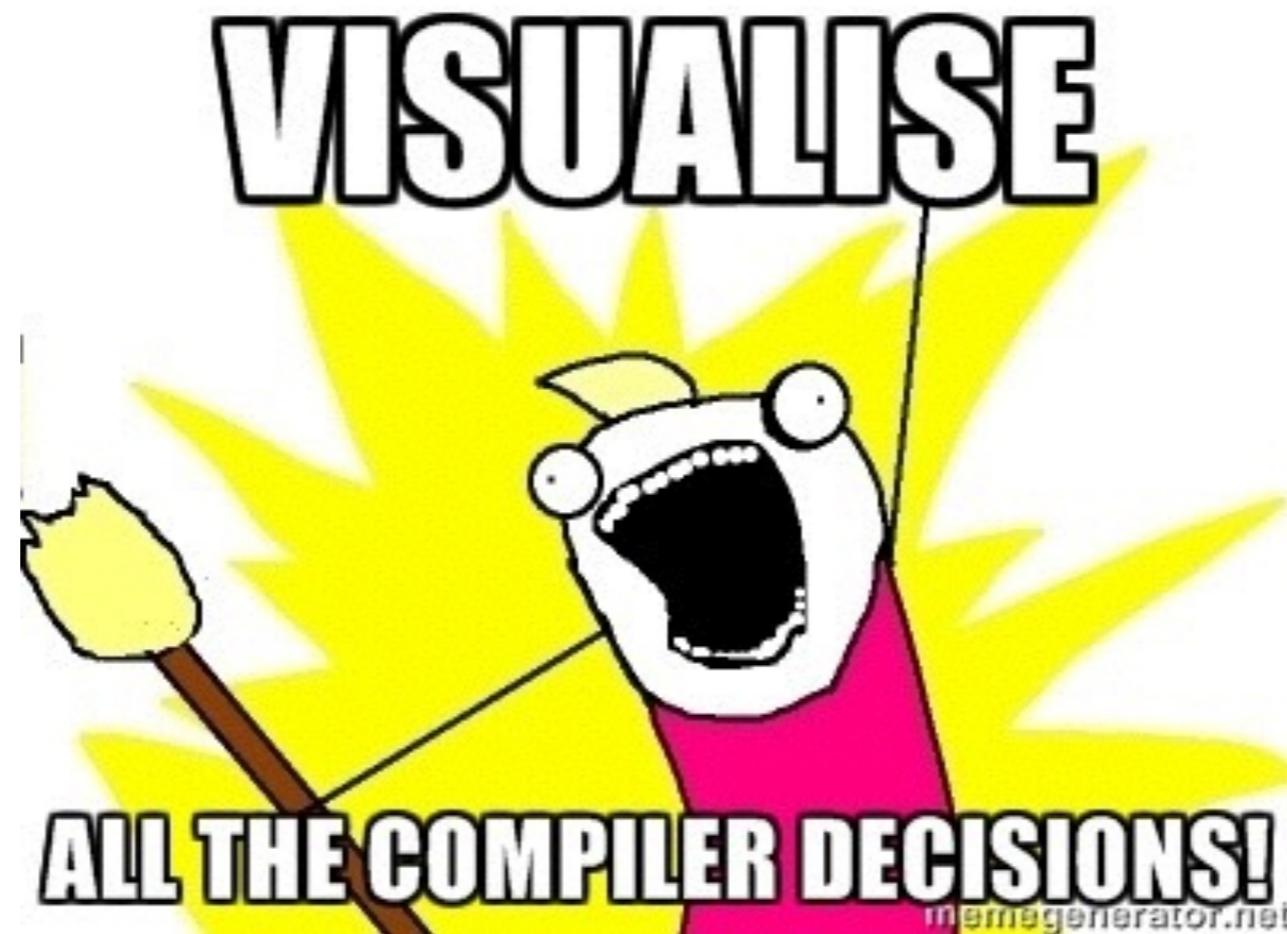
-XX:+PrintAssembly

hsdis binary in jre/lib/amd64/server

Significant performance overhead

I heard you like to grep?

Program	Size of HotSpot log
Microbenchmark	A few hundred KB
Desktop application	A few MB
Large application with assembly	50MB+



JITWatch

<https://github.com/AdoptOpenJDK/jitwatch/>

Compilations (when, how)

Deoptimisations (why)

Inlining successes and failures

Escape analysis (heap alloc, lock elision)

Branch probabilities

Intrinsics used

Getting Started

JITWatch - HotSpot Compilation Inspector

Sandbox Open Log Start Stop Config Chart Stats Histo TopList Code Cache TriView Suggestions (34) OVCs

Hide interfaces Hide uncompiled classes Hide non JIT-compiled class members

Packages

- java
- org
- org.adoptopenjdk
 - org.adoptopenjdk.jitwatch
 - org.adoptopenjdk.jitwatch.demo
- sun

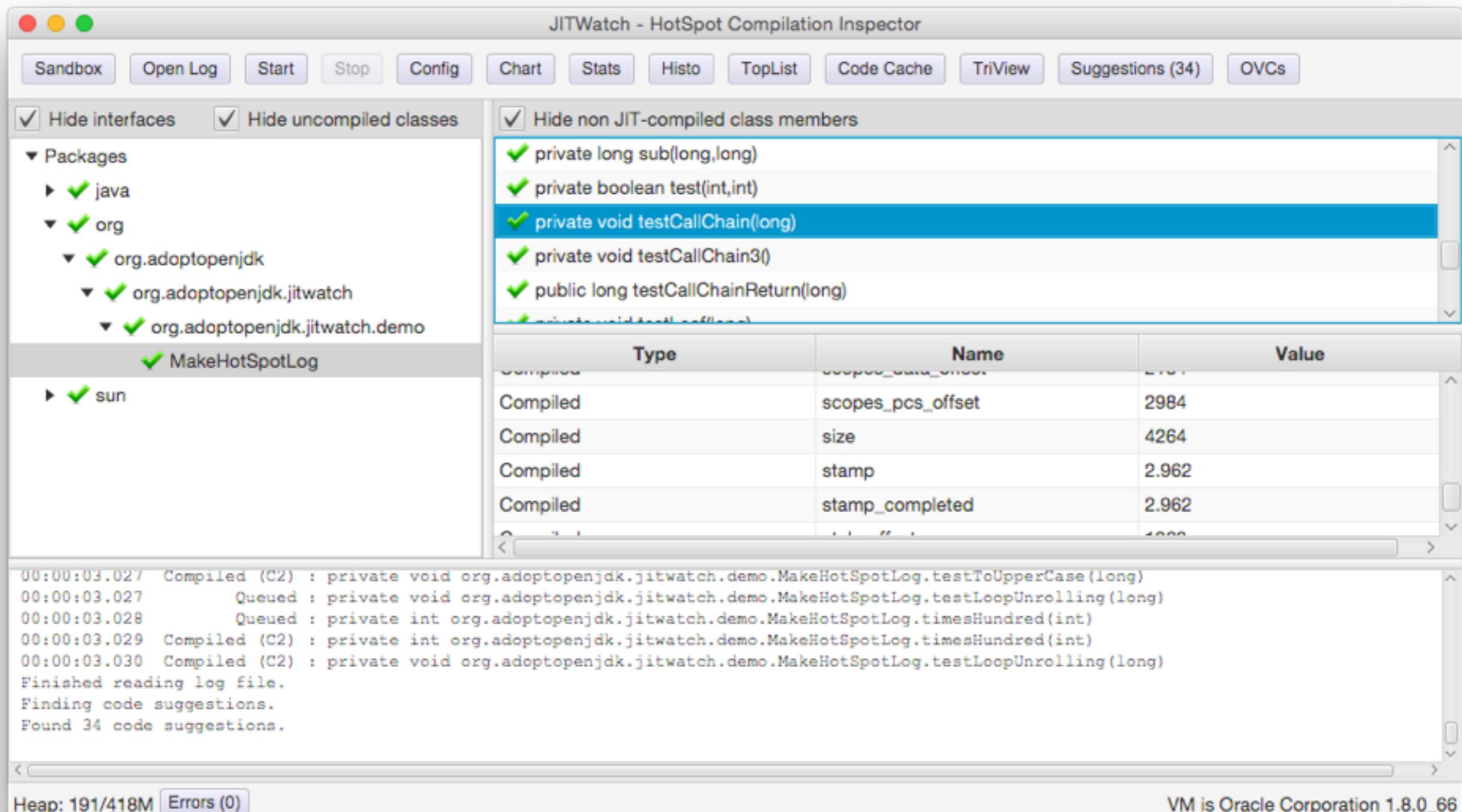
MakeHotSpotLog

private long sub(long,long)
private boolean test(int,int)
private void testCallChain(long)
private void testCallChain3()
public long testCallChainReturn(long)

Type	Name	Value
Compiled	scopes_pcs_offset	2984
Compiled	size	4264
Compiled	stamp	2.962
Compiled	stamp_completed	2.962

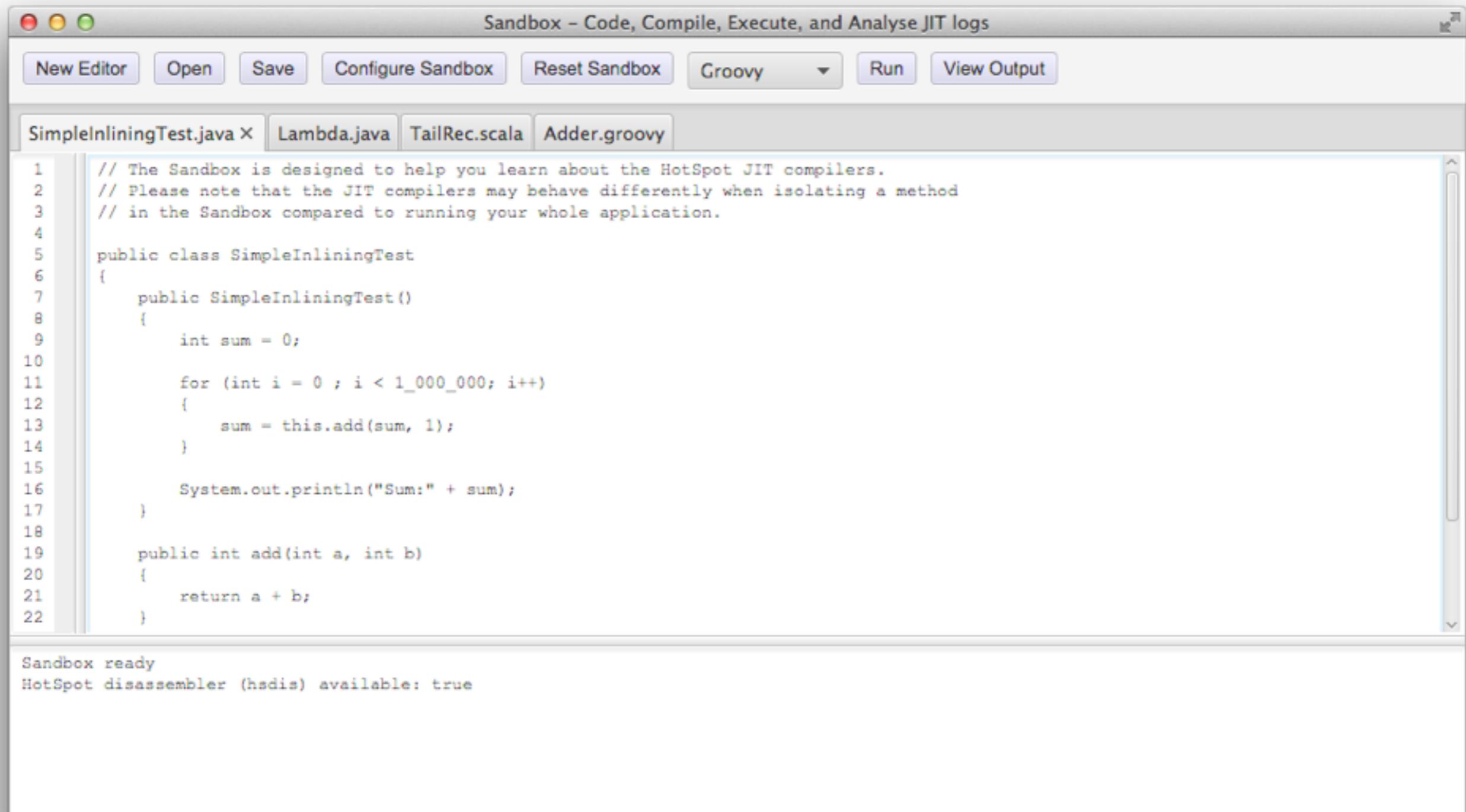
```
00:00:03.027 Compiled (C2) : private void org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.testToUpperCase(long)
00:00:03.027 Queued : private void org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.testLoopUnrolling(long)
00:00:03.028 Queued : private int org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:03.029 Compiled (C2) : private int org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:03.030 Compiled (C2) : private void org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.testLoopUnrolling(long)
Finished reading log file.
Finding code suggestions.
Found 34 code suggestions.
```

Heap: 191/418M Errors (0) VM is Oracle Corporation 1.8.0_66



Build with maven, ant, or gradle

Sandbox Mode



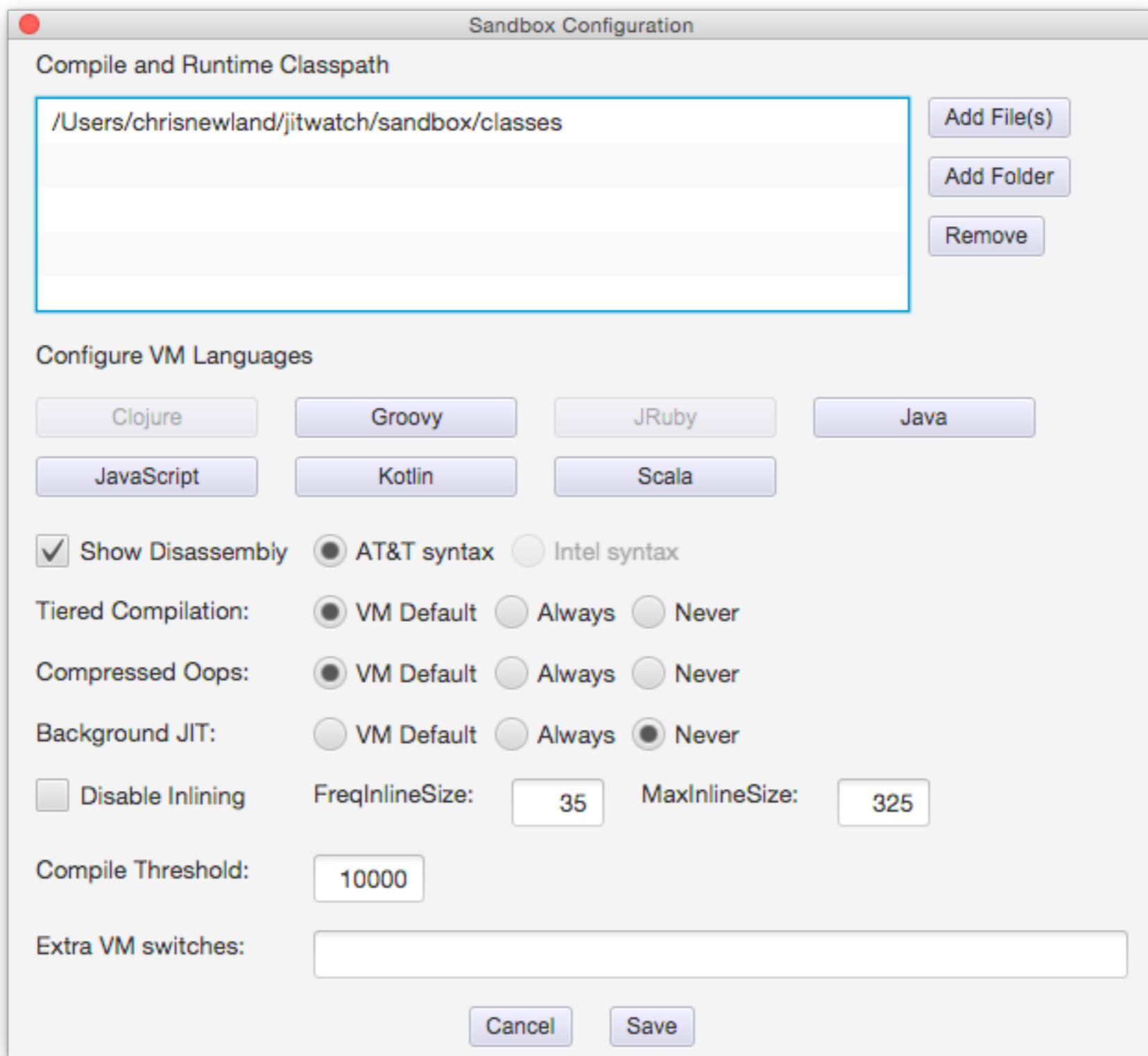
The screenshot shows the 'Sandbox - Code, Compile, Execute, and Analyse JIT logs' application window. The title bar includes standard OS X window controls and the application name. Below the title bar is a toolbar with buttons for 'New Editor', 'Open', 'Save', 'Configure Sandbox', 'Reset Sandbox', a dropdown menu set to 'Groovy', 'Run', and 'View Output'. A tab bar below the toolbar shows four tabs: 'SimpleInliningTest.java X' (selected), 'Lambda.java', 'TailRec.scala', and 'Adder.groovy'. The main area is a code editor containing Java code for a 'SimpleInliningTest' class. The code initializes a sum variable, loops 1,000,000 times adding 1 to sum, prints the result, and defines an add method. Lines 1 through 22 are numbered on the left. Below the code editor is a status message: 'Sandbox ready' followed by 'HotSpot disassembler (hsdis) available: true'.

```
// The Sandbox is designed to help you learn about the HotSpot JIT compilers.  
// Please note that the JIT compilers may behave differently when isolating a method  
// in the Sandbox compared to running your whole application.  
  
public class SimpleInliningTest  
{  
    public SimpleInliningTest()  
    {  
        int sum = 0;  
        for (int i = 0 ; i < 1_000_000; i++)  
        {  
            sum = this.add(sum, 1);  
        }  
        System.out.println("Sum:" + sum);  
    }  
    public int add(int a, int b)  
    {  
        return a + b;  
    }  
}
```

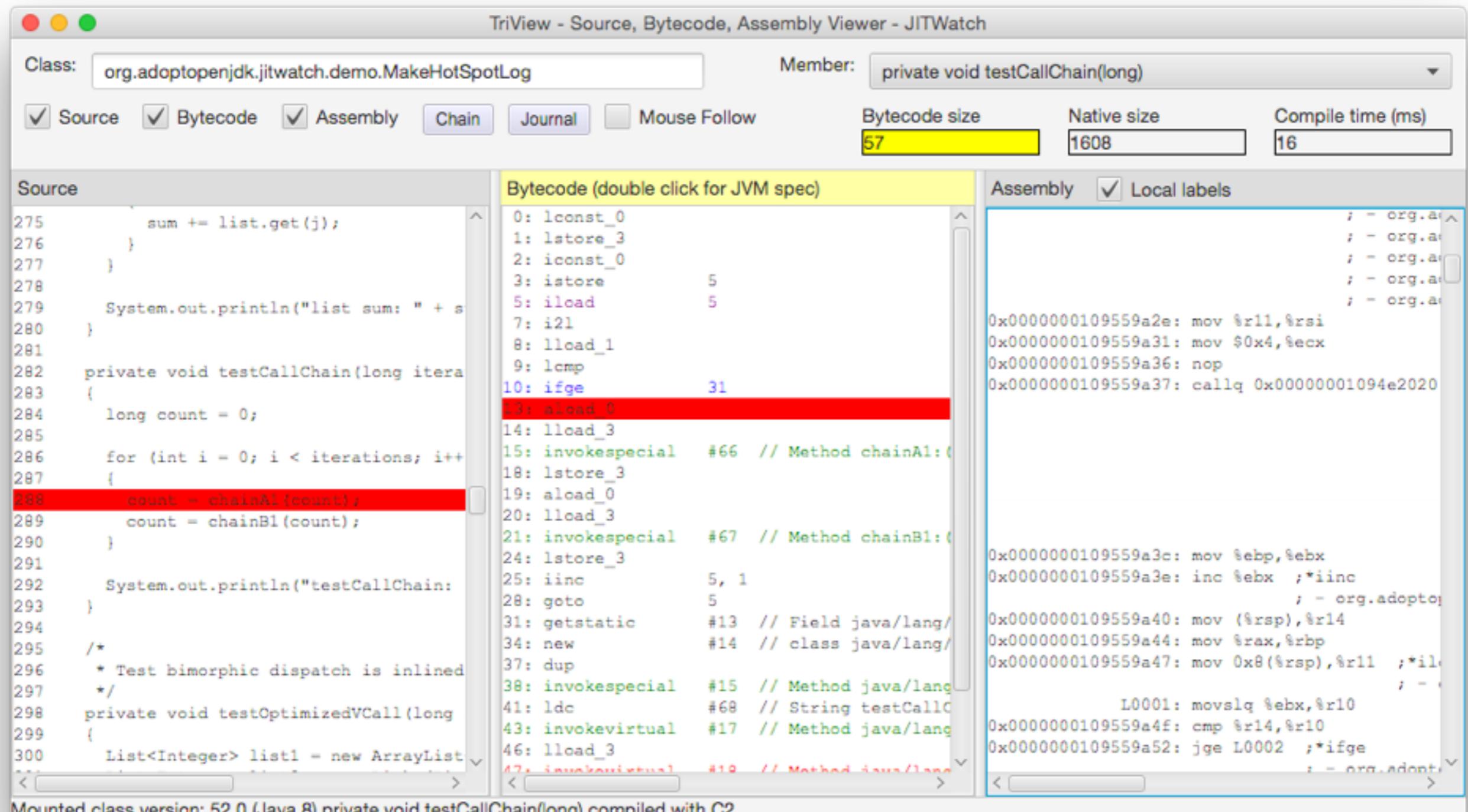
Sandbox ready
HotSpot disassembler (hsdis) available: true

Examples that exercise JIT behaviours

Sandbox Config



TriView Screen

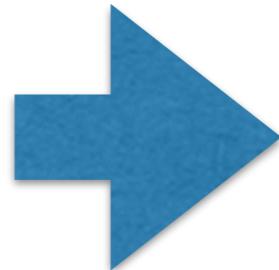


Inlining, traps, branches, escape analysis

Inlining

```
public int add(int x, int y) {  
    return x + y;  
}
```

```
int result = add(a, b);
```



```
int result = a + b;
```

Copy the body of the callee method into the call site

Eliminates the cost of method dispatch

Opens the door to other optimisations

Inlining Limits

Increases size of compiled code

< 35 bytes (**-XX:MaxInlineSize=n**)

< 325 bytes and “hot” (**-XX:FreqInlineSize=n**)

Inlining Failure Modes

JITWatch TopLists

Inlining Failure Reasons

Count	Reason
1647	too big
473	executed < MinInliningThreshold times
433	already compiled into a medium method
357	already compiled into a big method
271	call site not reached
257	native method
216	never executed
166	hot method too big
71	size > DesiredMethodLimit
33	recursive inlining is too deep
18	unloaded signature classes
11	exception method
8	inlining too deep
1	NodeCountInliningCutoff

BAD!

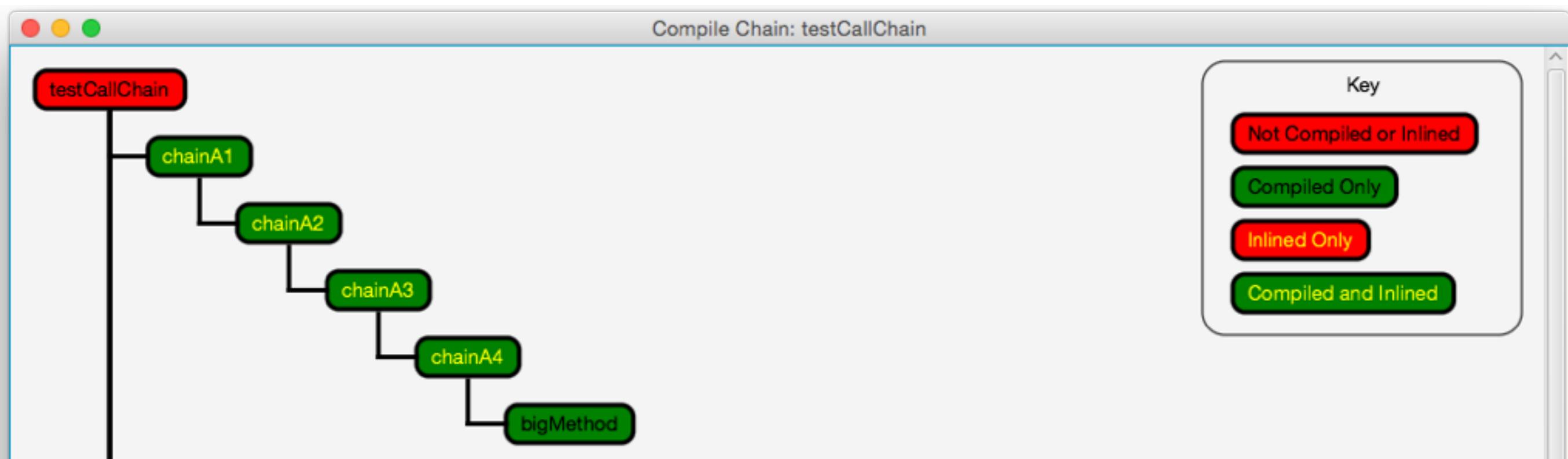
Inlining Suggestions

The screenshot shows a window titled "JITWatch Code Suggestions". A search bar at the top contains the text "org.adoptopenjdk". The main area is a table with three columns: "Score", "Type", and "Caller". There is one row in the table. The "Score" column has the value "20000". The "Type" column has the value "Inlining". The "Caller" column contains the following information:
org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog
private long chainA4(long)
View

The "Suggestion" column provides details about the inlining decision:
The call at bytecode 3 to
Class: org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog
Member: private long bigMethod(long,int)
was not inlined for reason: 'hot method too big'
The callee method is 'hot' but is too big to be inlined into the caller.
You may want to consider refactoring the callee into smaller methods.
Invocations: 20000
Size of callee bytecode: 350

Filter by package to exclude 3rd party and core libs

Call Chain



Look out for inlining failures in the call chain

Watch for overly deep call chains in hot code

JarScan Tool

- Static analysis tool, maven plugin
- Identifies methods > inlining threshold
- 3,613 above-threshold methods in 8u66 rt.jar
 - String.split
 - **String.toUpperCase / toLowerCase**
 - Parts of java.util.ComparableTimSort

```

public String toUpperCase(Locale locale) {
    if (locale == null) {
        throw new NullPointerException();
    }

    int firstLower;
    final int len = value.length;

    /* Now check if there are any characters that need to be changed. */
    scan: {
        for (firstLower = 0 ; firstLower < len; ) {
            int c = (int)value[firstLower];
            int srcCount;
            if ((c >= Character.MIN_HIGH_SURROGATE)
                && (c <= Character.MAX_HIGH_SURROGATE)) {
                c = codePointAt(firstLower);
                srcCount = Character.charCount(c);
            } else {
                srcCount = 1;
            }
            int upperCaseChar = Character.toUpperCaseEx(c);
            if ((upperCaseChar == Character.ERROR)
                || (c != upperCaseChar)) {
                break scan;
            }
            firstLower += srcCount;
        }
        return this;
    }

    /* result may grow, so i+resultOffset is the write location in result */
    int resultOffset = 0;
    char[] result = new char[len]; /* may grow */

    /* Just copy the first few upperCase characters. */
    System.arraycopy(value, 0, result, 0, firstLower);

    String lang = locale.getLanguage();
    boolean localeDependent =
        (lang == "tr" || lang == "az" || lang == "lt");
    char[] upperCharArray;
    int upperChar;
    int srcChar;
    int srcCount;
    for (int i = firstLower; i < len; i += srcCount) {
        srcChar = (int)value[i];
        if (((char)srcChar >= Character.MIN_HIGH_SURROGATE &&
            (char)srcChar <= Character.MAX_HIGH_SURROGATE) {
            srcChar = codePointAt(i);
            srcCount = Character.charCount(srcChar);
        } else {
            srcCount = 1;
        }
        if (localeDependent) {
            upperChar = ConditionalSpecialCasing.toUpperCaseEx(this, i, locale);
        } else {
            upperChar = Character.toUpperCaseEx(srcChar);
        }
        if ((upperChar == Character.ERROR)
            || (upperChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
            if (upperChar == Character.ERROR) {
                if (localeDependent) {
                    upperCharArray =
                        ConditionalSpecialCasing.toUpperCaseCharArray(this, i, locale);
                } else {
                    ...
                }
            }
        }
    }
}

```

String.toUpperCase()

439 bytes of bytecode

char[] can change size

Too big for inlining

ASCII Optimised

```
public String toUpperCaseASCII(String source)
{
    int len = source.length();

    char[] result = new char[len];

    for (int i = 0; i < len; i++)
    {
        char c = source.charAt(i);

        if (c >= 'a' && c <= 'z')
        {
            c -= 32;
        }

        result[i] = c;
    }

    return new String(result);
}
```

68 bytes of bytecode

Can be inlined

JMH Comparison

```
@State(Scope.Thread)
@BenchmarkMode(Mode.Throughput)
@OutputTimeUnit(TimeUnit.SECONDS)
public class Uppercase
{

    @Benchmark
    public String testStringToUpperCase()
    {
        return SOURCE.toUpperCase();
    }

    @Benchmark
    public String testCustomToUpperCase()
    {
        return toUpperCaseASCII(SOURCE);
    }

}
```

Benchmark	Mode	Cnt	Score	Error	Units
UpperCase.testCustomToUpperCase	thrpt	200	1792970.024	± 8598.436	ops/s
UpperCase.testStringToUpperCase	thrpt	200	820741.756	± 4346.516	ops/s

Custom version is more than twice as fast

Assertions

Disabled by default (enable with **-ea**)

Core-lib assertion code baked into bytecode

Counted in inlining budget

Can push a method over the inlining limit!

j.u.ComparableTimSort

Method	Size with assertions	Size without assertions	Saving
gallopLeft	327	244	25.4%
gallopRight	327	244	25.4%
mergeLo	652	517	18.6%
mergeHi	716	583	20.7%

Possible to create an rt.jar without assertions using OpenJDK

Modify javac to suppress assertion bytecode generation!

Callsite Polymorphism

HotSpot tracks observed implementations at each callsite.

Too many implementations can prevent inlining.

Implementations	Classification	Inlinable?
1	Monomorphic	Yes
2	Bimorphic	Yes
3+	Megamorphic	No*

-XX:TypeProfileMajorReceiverPercent=90

```

public class PolymorphismTest
{
    public interface Coin{ void deposit(); }

    public static int moneyBox = 0;

    public class Nickel implements Coin { public void deposit() { moneyBox += 5; } }
    public class Dime   implements Coin { public void deposit() { moneyBox += 10; } }
    public class Quarter implements Coin { public void deposit() { moneyBox += 25; } }

    public PolymorphismTest() {

        Coin nickel = new Nickel(); Coin dime = new Dime(); Coin quarter = new Quarter();
        Coin coin = null;

        // 1 = monomorphic dispatch - the method call will be inlined
        // 2 = bimorphic dispatch - the method call will be inlined
        // 3 = megamorphic dispatch - the method call will not be inlined
        final int maxImplementations = 3;

        for (int i = 0; i < 100000; i++) {
            switch(i % maxImplementations) {
                case 0: coin = nickel; break;
                case 1: coin = dime;   break;
                case 2: coin = quarter; break;
            }

            coin.deposit();
        }

        System.out.println("moneyBox:" + moneyBox);
    }
}

```

Implementation calls are balanced so megamorphic callsite will not be inlined.

Bimorphic

<input checked="" type="checkbox"/> Source	<input checked="" type="checkbox"/> Bytecode	<input checked="" type="checkbox"/> Assembly	Compile Chain	JIT Journal	Bytecode size 132	Native size 792	Compile time (ms) 8
<pre> 3 public interface Coin { void deposit(); } 4 5 public static int moneyBox = 0; 6 7 public class Nickel implements Coin { public void deposit() { moneyBox += 5; } } 8 9 public class Dime implements Coin { public void deposit() { moneyBox += 10; } } 10 11 public class Quarter implements Coin { public void deposit() { moneyBox += 25; } } 12 13 public PolymorphismTest() { 14 Coin nickel = new Nickel(); 15 Coin dime = new Dime(); 16 Coin quarter = new Quarter(); 17 18 Coin coin = null; 19 20 // change the variable maxImplementations to control the inlining behaviour 21 // 2 = bimorphic dispatch - the method call will be inlined 22 // 3 = megamorphic dispatch - the method call will not be inlined 23 24 final int maxImplementations = 2; 25 26 for (int i = 0; i < 100000; i++) { 27 switch(i % maxImplementations) { 28 case 0: coin = nickel; break; 29 case 1: coin = dime; break; 30 case 2: coin = quarter; break; 31 } 32 33 coin.deposit(); 34 } 35 36 System.out.println("moneyBox:" + moneyBox); 37 } 38 39 public static void main(String[] args) { 40 new PolymorphismTest(); 41 } 42 }</pre>	<p>Bytecode (double click for JVM spec)</p> <pre> 32: astore 4 34: iconst_2 35: istore 5 37: iconst_0 38: istore 6 40: iload 6 42: ldc #8 // int 100 44: if_icmpge 104 47: iload 6 49: iconst_2 50: irem 51: tableswitch { // 0 to 2 0:76 1:82 2:88 default:91 } 76: aload_1 77: astore 4 79: goto 91 82: aload_2 83: astore 4 85: goto 91 88: aload_3 89: astore 4 91: aload 4 93: invokeinterface #9, 1 // Interface 98: astore 6, 1 101: Inlined: Yes, inline (hot) Count: 11264 Iicount: 7282 Bytes: 9 Prof factor: 1 111: invokespecial #12 // Method Ctrl-click to inspect this method 114: Backspace to return 116: invokevirtual #4 // Method 119: getstatic #15 // Field m 122: invokevirtual #16 // Method 125: invokevirtual #17 // Method 128: invokevirtual #18 // Method 131: return</pre>	<p>Assembly</p> <pre> 0x000000010249c67e: add \$0x5,%r11d 0x000000010249c682: mov %r11d,0x98(%r11) 0x000000010249c689: jmpq 0x000000010249c690(%rip) 0x000000010249c68e: cmp %rsi,%r10 0x000000010249c691: jne 0x000000010249c693 0x000000010249c693: add \$0x5,%r11d 0x000000010249c697: mov %r11d,0x98(%r11) 0x000000010249c69e: jmp 0x000000010249c6a0 0x000000010249c6a0: mov \$0xffffffff 0x000000010249c6a5: xchg %ax,%ax 0x000000010249c6a7: callq 0x000000010249c6ac 0x000000010249c6ac: callq 0x000000010249c6b1 0x000000010249c6b1: mov %r13d,%r10d 0x000000010249c6b4: mov %r10d,%r13d 0x000000010249c6b7: mov \$0xffffffff 0x000000010249c6bc: mov %rcx,%rbp 0x000000010249c6bf: mov %rdx,(%rsp) 0x000000010249c6c3: mov %r13d,0x10(%r13) 0x000000010249c6c8: mov %rbx,0x18(%r13) 0x000000010249c6cd: mov %r14,0x20(%r14) 0x000000010249c6d2: nop 0x000000010249c6d3: callq 0x000000010249c6d8 0x000000010249c6dd: add \$0xa,%r11d</pre>					

Megamorphic

Source Bytecode Assembly Compile Chain JIT Journal

Bytecode size	Native size	Compile time (ms)
132	504	4

Source

```

3  public interface Coin { void deposit() ^ 
4
5  public static int moneyBox = 0;
6
7  public class Nickel implements Coin {
8
9  public class Dime implements Coin { pu
10 public class Quarter implements Coin {
11
12 public PolymorphismTest() {
13     Coin nickel = new Nickel();
14     Coin dime = new Dime();
15     Coin quarter = new Quarter();
16
17     Coin coin = null;
18
19     // change the variable maxImplemen
20     // 2 - bimorphic dispatch - the me
21     // 3 - megamorphic dispatch - the me
22
23     final int maxImplementations = 3;
24
25     for (int i = 0; i < 100000; i++) {
26         switch(i % maxImplementations) {
27             case 0: coin = nickel; break;
28             case 1: coin = dime; break;
29             case 2: coin = quarter; break;
30         }
31
32         coin.deposit();
33     }
34
35     System.out.println("moneyBox:" + mon
36

```

Bytecode (double click for JVM spec)

```

    /**
     * invokespecial   */ // method polymorp^
30: astore_3
31: aconst_null
32: astore        4
33: iconst_3
34: istore        5
35: iconst_0
36: istore        6
37: iload          6
38: ldc           #8  // int 100000
39: if_icmpge    104
40: iload          6
41: iconst_3
42: irem
43: tableswitch { // 0 to 2
44:     0:76
45:     1:82
46:     2:88
47: default:91
48: }
49: aload_1
50: astore        4
51: goto          91
52: astore        4
53: goto          91
54: astore        4
55: goto          91
56: astore        4
57: goto          91
58: invokeinterface #9, 1// InterfaceMethod
59: ldc           6, 1
60: Ctrl-click to inspect this method
61: Backspace to return 10 // Field java/lang
62: new           #11 // class java/lang
63: dup

```

Assembly

```

0x0000000109aecd2f: test %r9,%r9
0x0000000109aecd32: je 0x0000000109aece23
0x0000000109aecd38: mov 0x8(%r9),%r10
0x0000000109aecd3c: movabs $0x2592c5db8,%rdi ; {metadata('PolymorphismTest')
0x0000000109aecd46: cmp %rdi,%r10
0x0000000109aecd49: jne 0x0000000109aece35 ;*iload
0x0000000109aecd4f: jmp 0x0000000109aecd93 ; - PolymorphismTest::<init>@40 (line
0x0000000109aecd51: mov %rcx,0x18(%rsp) ;*aload
0x0000000109aecd56: mov %rcx,0x10(%rsp)
0x0000000109aecd5b: mov %r11,0x8(%rsp)
0x0000000109aecd60: mov %r9,(%rsp)
0x0000000109aecd64: mov %rbx,%ebp ;*tableswitch
0x0000000109aecd66: mov 0x18(%rsp),%rsi
0x0000000109aecd6b: xchg %ax,%ax
0x0000000109aecd6d: movabs $0xfffffffffffffff,%rax
0x0000000109aecd77: callq 0x0000000109abaf60 ; OopMap{[0]=Oop [8]=Oop [16]=Oo
0x0000000109aecd7c: inc %ebp ;*iinc
0x0000000109aecd7e: mov 0x18(%rsp),%r13
0x0000000109aecd83: mov %ebp,%rbx
0x0000000109aecd85: mov (%rsp),%r9
0x0000000109aecd89: mov 0x8(%rsp),%r11
0x0000000109aecd8e: mov 0x10(%rsp),%rcx ;*iload
0x0000000109aecd93: cmp $0x186a0,%rbx
0x0000000109aecd99: jge 0x0000000109aece00 ;*if_icmpge
0x0000000109aecd9b: movslq %rbx,%r8

```

Escape Analysis



Scope analysis optimisations for eliminating
heap allocations and object locks

NoEscape

Object does not escape method scope.

Can avoid memory allocation on the JVM heap.

Reduce overhead of high-churn objects.

ArgEscape

Object escapes method scope (as arg?)

Does not escape current thread.

Cannot eliminate heap allocation.

Locks on this object may be elided.

GlobalEscape

Accessible by other methods and threads.

Not available for heap elimination or lock elision.

Garbage collected at end of life.

Allocation Elimination

Object explosion

Fields are treated as locals

Register allocator decides where they are stored:

Prefer registers

Spill to stack if necessary

```
public class EscapeTest
{
    private final int value;

    public EscapeTest(final int value)
    {
        this.value = value;
    }

    public boolean equals(EscapeTest other)
    {
        return this.value == other.value;
    }

    public static int run()
    {
        int matches = 0;

        java.util.Random random = new java.util.Random();

        for (int i = 0; i < 100_000_000; i++)
        {
            int v1 = random.nextBoolean() ? 1 : 0;
            int v2 = random.nextBoolean() ? 1 : 0;

            final EscapeTest e1 = new EscapeTest(v1);
            final EscapeTest e2 = new EscapeTest(v2);

            if (e1.equals(e2))
            {
                matches++;
            }
        }

        return matches;
    }

    public static void main(final String[] args)
    {
        System.out.println(run());
    }
}
```

Inlining of equals()
prevents ArgEscape

Hot Loop Allocations

With Escape Analysis

```
java -Xms1G -Xmx1G -XX:+PrintGCDetails -verbose:gc EscapeTest
50001193
Heap
PSYoungGen      total 305664K, used 20972K [0x00000007aab00000, 0x00000007c0000000, 0x00000007c0000000)
eden space 262144K, 8% used [0x00000007aab00000,0x00000007abf7b038,0x00000007bab00000)
from space 43520K, 0% used [0x00000007bd580000,0x00000007bd580000,0x00000007c000000)
to   space 43520K, 0% used [0x00000007bab00000,0x00000007bab00000,0x00000007bd580000)
ParOldGen      total 699392K, used 0K [0x0000000780000000, 0x00000007aab00000, 0x00000007aab00000)
object space 699392K, 0% used [0x0000000780000000,0x0000000780000000,0x00000007aab00000)
Metaspace       used 2626K, capacity 4486K, committed 4864K, reserved 1056768K
  class space  used 285K, capacity 386K, committed 512K, reserved 1048576K
```

Without Escape Analysis

```
java -Xms1G -Xmx1G -XX:+PrintGCDetails -verbose:gc -XX:-DoEscapeAnalysis EscapeTest
[GC (Allocation Failure) [PSYoungGen: 262144K->368K(305664K)] 262144K->376K(1005056K), 0.0006532 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 262512K->432K(305664K)] 262520K->440K(1005056K), 0.0006805 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 262576K->416K(305664K)] 262584K->424K(1005056K), 0.0005623 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 262560K->352K(305664K)] 262568K->360K(1005056K), 0.0006364 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 262496K->400K(305664K)] 262504K->408K(1005056K), 0.0005717 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 262544K->384K(348672K)] 262552K->392K(1048064K), 0.0007290 secs] [Times: user=0.00 sys=0.01, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 348544K->32K(348672K)] 348552K->352K(1048064K), 0.0006297 secs] [Times: user=0.00 sys=0.01, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 348192K->32K(347648K)] 348512K->352K(1047040K), 0.0004195 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 347168K->0K(348160K)] 347488K->320K(1047552K), 0.0004126 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 347136K->0K(348160K)] 347456K->320K(1047552K), 0.0004189 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
50001608
Heap
PSYoungGen      total 348160K, used 180445K [0x00000007aab00000, 0x00000007c0000000, 0x00000007c0000000)
eden space 347136K, 51% used [0x00000007aab00000,0x00000007b5b37438,0x00000007bfe00000)
from space 1024K, 0% used [0x00000007bff00000,0x00000007bff00000,0x00000007c0000000)
to   space 1024K, 0% used [0x00000007bfe00000,0x00000007bfe00000,0x00000007bff00000)
ParOldGen      total 699392K, used 320K [0x0000000780000000, 0x00000007aab00000, 0x00000007aab00000)
object space 699392K, 0% used [0x0000000780000000,0x0000000780050050,0x00000007aab00000)
Metaspace       used 2626K, capacity 4486K, committed 4864K, reserved 1056768K
  class space  used 285K, capacity 386K, committed 512K, reserved 1048576K
```

Class: EscapeTest

Member: public static int run()

 Source Bytecode Assembly Chain Journal Mouse Follow

Bytecode size	Native size	Compile time (ms)
87	3080	8

Source

```

1  public class EscapeTest
2  {
3      private final int value;
4
5      public EscapeTest(final int value)
6      {
7          this.value = value;
8      }
9
10     public boolean equals(EscapeTest other)
11     {
12         return this.value == other.value;
13     }
14
15     public static int run()
16     {
17         int matches = 0;
18
19         java.util.Random random = new java.util.Random();
20
21         for (int i = 0; i < 100_000_000; i++)
22         {
23             int v1 = random.nextBoolean() ? 1 : 0;
24             int v2 = random.nextBoolean() ? 1 : 0;
25
26             final EscapeTest e1 = new EscapeTest(v1);
27             final EscapeTest e2 = new EscapeTest(v2);
28
29             if (e1.equals(e2))
30             {
31                 matches++;
32             }
33         }
34
35         return matches;
36     }
37
38     public static void main(final String[] args)
39     {
40         System.out.println(run());
41     }
42 }
```

Bytecode (double click for JVM spec)

```

0: iconst_0
1: istore_0
2: new #3 // class java/util/Random
5: dup
6: invokespecial #4 // Method java/util/Random."<init>": ()V
9: astore_1
10: iconst_0
11: istore_2
12: iload_2
13: ldc #5 // int 100000000
15: if_icmpge 85
18: aload_1
19: invokevirtual #6 // Method java/util/Random.nextInt:()I
22: ifeq 29
25: iconst_1
26: goto 30
29: iconst_0
30: istore_3
31: aload_1
32: invokevirtual #6 // Method java/util/Random.nextInt:()I
35: ifeq 42
38: iconst_1
39: goto 43
42: iconst_0
43: istore 4
46+new #7 // class EscapeTest
48: dup
49: iload_3
50: invokespecial #8 // Method "<init>": (I)V
53: astore 5
56+new #7 // class EscapeTest
58: dup
59: iload 4
61: invokespecial #8 // Method "<init>": (I)V
64: astore 6
66: aload 5
68: aload 6
70: invokevirtual #9 // Method equals: (LEscapeTest;)Z
73: ifeq 79
76: iinc 0, 1
79: iinc 2, 1
82: goto 12
85: iload_0
86: ireturn

```

Mounted class version: 52.0 (Java 8) public static int run() compiled with C2

JITWatch shows both heap allocations were eliminated

Branch Prediction

```
public class BranchPrediction
{
    public BranchPrediction()
    {
        int thingOne = 0;
        int thingTwo = 0;

        java.util.Random random = new java.util.Random();

        for (int i = 0; i < 1_000_000; i++)
        {
            if (random.nextBoolean())
            {
                thingOne++;
            }
            else
            {
                thingTwo++;
            }
        }

        System.out.println(thingOne + "/" + thingTwo);
    }

    public static void main(String[] args)
    {
        new BranchPrediction();
    }
}
```

HotSpot measures branches

Speculative execution

Code elimination

Puts in a trap

Branch Prediction

JITWatch Code Suggestions				
Score	Type	Caller	Suggestion	
5632	Branch	BranchPrediction public void BranchPrediction() View	Method contains an unpredictable branch at bytecode 30 that was observed 11264 times and is taken with probability 0.503374. It may be possible to modify the branch (for example by sorting a Collection before iterating) to make it more predictable.	
3351	Branch	java.util.Random public boolean nextBoolean() View	Method contains an unpredictable branch at bytecode 5 that was observed 6701 times and is taken with probability 0.501716. It may be possible to modify the branch (for example by sorting a Collection before iterating) to make it more predictable.	
3350	Branch	java.util.Random public boolean nextBoolean() View	Method contains an unpredictable branch at bytecode 5 that was observed 6700 times and is taken with probability 0.501791. It may be possible to modify the branch (for example by sorting a Collection before iterating) to make it more predictable.	

JITWatch highlights unpredictable branches

Branch Prediction

TriView - Source, Bytecode, Assembly Viewer - JITWatch

Class: BranchPrediction Member: public void BranchPrediction()

Source Bytecode Assembly Chain Journal Mouse Follow

Bytecode size: 78 Native size: 376 Compile time (ms): 5

Source	Bytecode (double click for JVM spec)	Assembly
1 public class BranchPrediction 2 { 3 public BranchPrediction() 4 { 5 int thingOne = 0; 6 int thingTwo = 0; 7 8 java.util.Random random = new ja 9 10 for (int i = 0; i < 1_000_000; i 11 { 12 if (random.nextBoolean()) 13 { 14 thingOne++; 15 } 16 else 17 { 18 thingTwo++; 19 } 20 } 21 22 System.out.println(thingOne + "/" 23 } 24 25 public static void main(String[] args 26 { 27 new BranchPrediction(). 28 }	Bytecode (double click for JVM spec) 6: iconst_0 7: istore_2 8: new #2 // class java/util/ 11: dup 12: invokespecial #3 // Method java/util/ 15: astore_3 16: iconst_0 17: istore 4 19: iload 4 21: ldc #4 // int 1000000 23: if_icmpge 48 26: aload_3 27: invokevirtual #5 // Method java/util/ 30: ifeq 39 33: iinc 36: goto 39 39: iinc 42: iinc 45: goto 19 48: getstatic #6 // Field java/lang/ 51: new #7 // class java/lang/ 54: dup 55: invokespecial #8 // Method java/lang/ 58: iload_1 59: invokevirtual #9 // Method java/lang/ 62: ldc #10 // String /	Assembly 0x000000010b54f894: test %r10d,%r10d 0x000000010b54f897: je L0003 ;*ifeq ; - java.util. ; - BranchPred. 0x000000010b54f899: shr \$0x2f,%rdi 0x000000010b54f89d: and \$0x1,%rdi 0x000000010b54f8a1: mov %edi,%r11d 0x000000010b54f8a4: test %r11d,%r11d 0x000000010b54f8a7: je L0000 ;*ifeq ; - BranchPred. 0x000000010b54f8a9: inc %ebx ;*iinc ; - BranchPred. 0x000000010b54f8ab: jmp L0001 L0003: mov \$0xffffffff65,%esi 0x000000010b54f8b2: mov %ebx,(%rsp) 0x000000010b54f8b5: mov %r14d,0x4(%rsp) 0x000000010b54f8ba: mov %r13,0x8(%rsp) 0x000000010b54f8bf: mov %r8,0x10(%rsp) 0x000000010b54f8c4: mov %r11,0x18(%rsp) 0x000000010b54f8c9: mov %r10d,0x20(%rsp) 0x000000010b54f8ce: nop 0x000000010b54f8cf: callq 0x000000010b4d7ee0

Count: 11264
Branch taken: 5697
Branch not taken: 5567
Taken Probability: 0.505771

Mounted class version: 52.0 (Java 8) public void BranchPrediction() compiled with C2

Intrinsics



Highly optimised native implementations

Use features of target CPU

Intrinsic optimised code found in

**Math, Unsafe, System, Class, Arrays, String,
StringBuilder, AESCrypt, ...**

Full list in

hotspot/src/share/vm/classfile/vmSymbols.hpp

Intrinsics

Don't assume core-libs code is unoptimised

Math.log10(double) is 2 CPU instructions

```
instruct log10D_reg(regD dst) %{
    // The source and result Double operands in XMM registers
    match(Set dst (Log10D dst));
    // fldlg2          ; push log_10(2) on the FPU stack; full 80-bit number
    // fyl2x          ; compute log_10(2) * log_2(x)
    format %{
        "fldlg2\t\t\t#Log10\n\t"
        "fyl2x\t\t\t# Q=Log10*Log_2(x)\n\t"
    %}
    ins_encode(Opcode(0xD9), Opcode(0xEC),    // fldlg2
              Push_SrcXD(dst),
              Opcode(0xD9), Opcode(0xF1),    // fyl2x
              Push_ResultXD(dst));

    ins_pipe( pipe_slow );
%}
```

(**hotspot/src/cpu/x86/vm/x86_64.ad**)

Intrinsics

TriView - Source, Bytecode, Assembly Viewer - JITWatch

Class: org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog Member: private void intrinsicTestMin(int)

Source Bytecode Assembly Chain Journal Mouse Follow

Bytecode size	Native size	Compile time (ms)
54	184	2

Source

```
137     {  
138         // x86 has intrinsic for System.arraycopy  
139         System.arraycopy(src, 0, dst, 0, src.length);  
140  
141         for (int dstVal : dst)  
142         {  
143             dstSum += add(dstSum, dstVal);  
144         }  
145     }  
146  
147     System.out.println("intrinsicTest: " + dstSum);  
148 }  
149  
150 // http://openjdk.5641.n7.nabble.com/Intrinsics-for-Math  
151 private void intrinsicTestMin(int iterations)  
152 {  
153     long sum = 0;  
154  
155     for (int i = 0; i < iterations; i++)  
156     {  
157         // x86 has intrinsic for Math.min  
158         sum = Math.min(i, i + 1);  
159     }  
160  
161     System.out.println("intrinsicTest: " + sum);  
162 }  
163  
164     private long add(long a, long b)  
165     {  
166         return a + b;  
167     }  
168  
169     private long sub(long a, long b)  
170     {  
171         return a - b;  
172     }  
173  
174     public void tooBigToInline(int iterations)  
175     {  
176         long count = 0;  
177  
178         for (int i = 0; i < iterations; i++)  
179         {
```

Bytecode (double click for JVM spec)

```
0: lconst_0  
1: lstore_2  
2: iconst_0  
3: istore        4  
5: iload          4  
7: iload_1  
8: if_icmpge     28  
11: iload          4  
13: iload          4  
15: iconst_1  
16: iadd  
17: invokestatic   #39  // Method java/lang/Math:min(I)I  
20: i2l  
21: lstore_2  
22: iinc        4, 1  
25: goto          5  
28: getstatic     #13  // Field java/lang/System.out:Ljava/io/PrintStream;  
31: new           #14  // class java/lang/StringBuilder  
34: dup  
35: invokespecial #15  // Method java/lang/StringBuilder.<init>()  
38: ldc            #38  // String intrinsicTest:  
40: invokevirtual #17  // Method java/lang/StringBuilder.append(I)V  
43: lload_2  
44: invokevirtual #18  // Method java/lang/StringBuilder.append(C)V  
47: invokevirtual #19  // Method java/lang/StringBuilder.toString()Ljava/lang/String;  
50: invokevirtual #20  // Method java/io/PrintStream.println:(Ljava/lang/String;)V  
53: return
```

Assembly

```
# (method) {0x0000000258cc54b0} 'intrinsicTestMin' '(I)V' in 'o^'  
0x000000010952cfa0: callq 0x000000010887e33c ; (runtime_call  
0x000000010952cfa5: data16 data16 nopl 0x0(%rax,%rax,1)  
0x000000010952cfb0: mov %eax,-0x1400(%rsp)  
0x000000010952cfb7: push %rbp  
0x000000010952cfb8: sub $0x30,%rsp  
0x000000010952efbc: mov 0x8(%rsi),%rbx  
0x000000010952fc0: mov (%rsi),%r14d  
0x000000010952fc3: mov 0x18(%rsi),%rbp  
0x000000010952efc6: mov %rsi,%rdi  
0x000000010952efc9: movabs $0x1088dc056,%r10  
0x000000010952cfcd3: callq ; - org.adoptopenjdk.jitwatch.demo.M  
0x000000010952cfcd6: cmp %rbp,%r14d  
0x000000010952cfcd9: jge L0003 ;*goto  
0x000000010952cfdb: mov %r14d,%r11d  
0x000000010952cfde: inc %r11d ;*iadd  
0x000000010952cfel: cmp %r11d,%r14d  
0x000000010952cfe4: mov %r11d,%r10d  
0x000000010952cfe7: cmovl %r14d,%r10d ;*invokestatic min  
0x000000010952cfed: jmp L0001  
0x000000010952cfed: data16 xchq %ax,%ax  
L0000: mov %r11d,%r9d  
0x000000010952cff3: inc %r9d ;*iadd  
0x000000010952cff6: cmp %r9d,%r11d  
0x000000010952cff9: mov %r9d,%r10d  
0x000000010952effc: cmovl %r11d,%r10d ;*invokestatic min  
0x000000010952d000: mov %r9d,%r11d ;*iload  
L0001: movslq %r10d,%rbx ; CopMap(eff=102)  
; - org.adoptopenjdk.jit  
; *goto  
0x000000010952d006: test %eax,-0xa400c(%rip) # 0x0000000109489  
; - org.adoptopen  
; *poll  
0x000000010952d00c: cmp %rbp,%r11d  
0x000000010952d00f: jl L0000 ;*if_icmpge  
; - org.adoptopenjdk.jit  
L0002: mov $0xffffffff65,%esi
```

Local labels

Ctrl-click to inspect this method
Backspace to return

Mounted class version: 52.0 (Java 8) private void intrinsicTestMin(int) compiled with C2

JITWatch highlights use of intrinsics

TL;DR

- JIT logs should be part of perf toolkit
- Keep methods small for inlining (Head Test)
- Use inlineable core lib methods
- Look out for unpredictable branches
- Use appropriate method visibility (CHA)
- Count interface implementations
- Check allocations in hot code are EA'd

Epilogue

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.

Donald Knuth, Computer Programming as an Art

Resources

- JITWatch on GitHub
 - <http://www.github.com/AdoptOpenJDK/jitwatch>
 - AdoptOpenJDK project
 - Pull requests are welcome!
- Mailing list
 - <groups.google.com/jitwatch>
- Twitter
 - [@chriswhocodes](#)