

Understanding HotSpot JVM Performance with JITWatch

Chris Newland, JavaZone 2016-09-08

Slides license: Creative Commons-Attribution-ShareAlike 3.0

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

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

Bio

Chris Newland

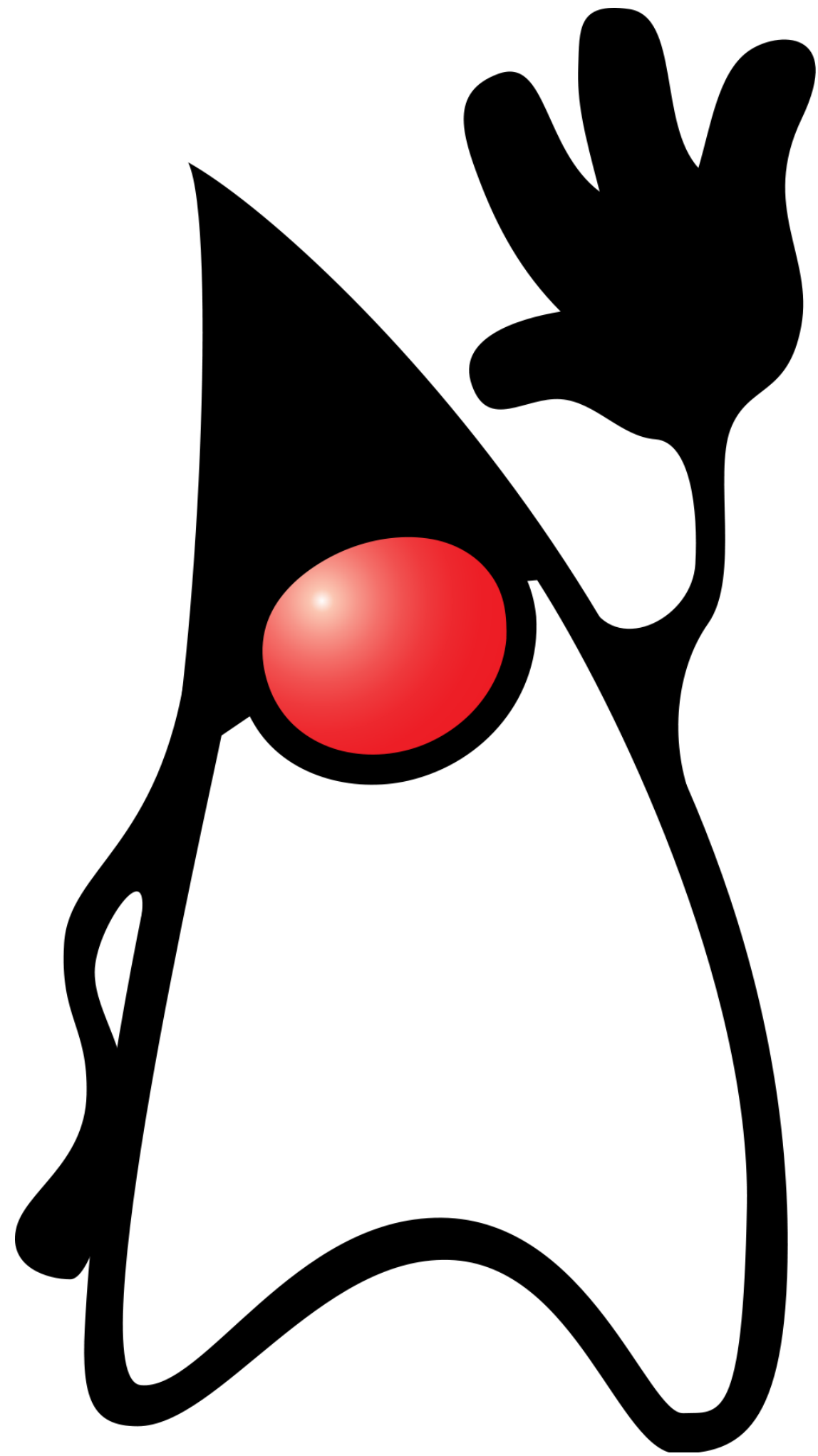
Market data guy at  **ADVFN**
www.advfn.com

@chriswhocodes on Twitter

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

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

The amazing JVM



An aerial photograph of a city, likely Prague, showing a dense cluster of buildings and a large, prominent cathedral with a tall spire in the center. The image is slightly blurred and serves as a background for the text.

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

Object-oriented and functional!

Strongly and dynamically typed!

Memory management and concurrency!

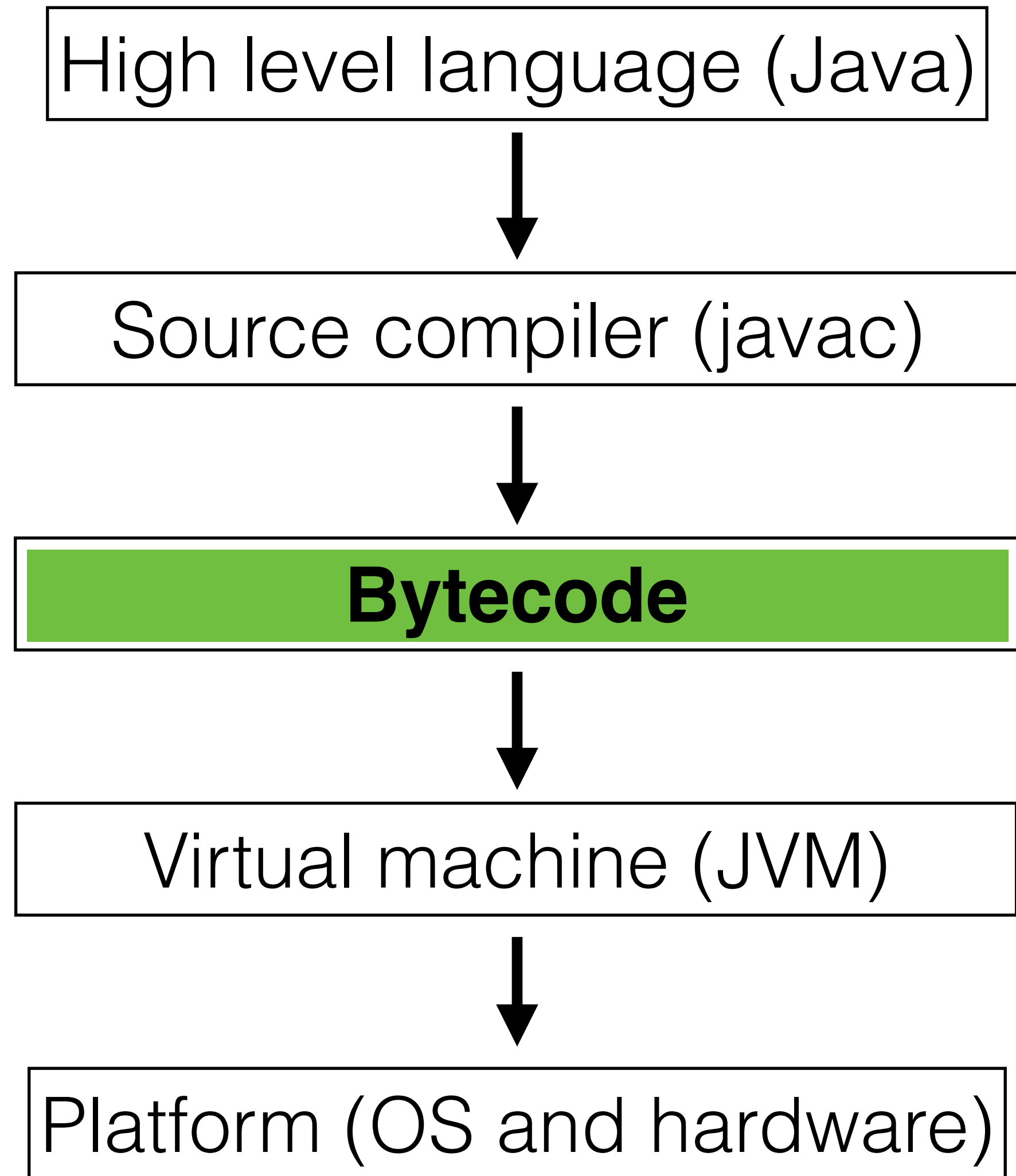
Abstraction!



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

A common language



Bytecode

(Portable instruction set, 256 possible instructions)

```
public int add(int a, int b) javac → public int add(int, int);  
{  
    return a + b;  
}  
descriptor: (II)I  
flags: ACC_PUBLIC  
Code:  
    stack=2, locals=3, args_size=3  
    0: iload_1  
    1: iload_2  
    2: iadd  
    3: ireturn
```

Interpreted on a virtual stack machine

A simple interpreter

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

    switch(opcode)
    {
    case 00:
        // handle
        break;
    case 01:
        // handle
        break;
    ...
    case ff:
        // handle
        break;
    }
}
```

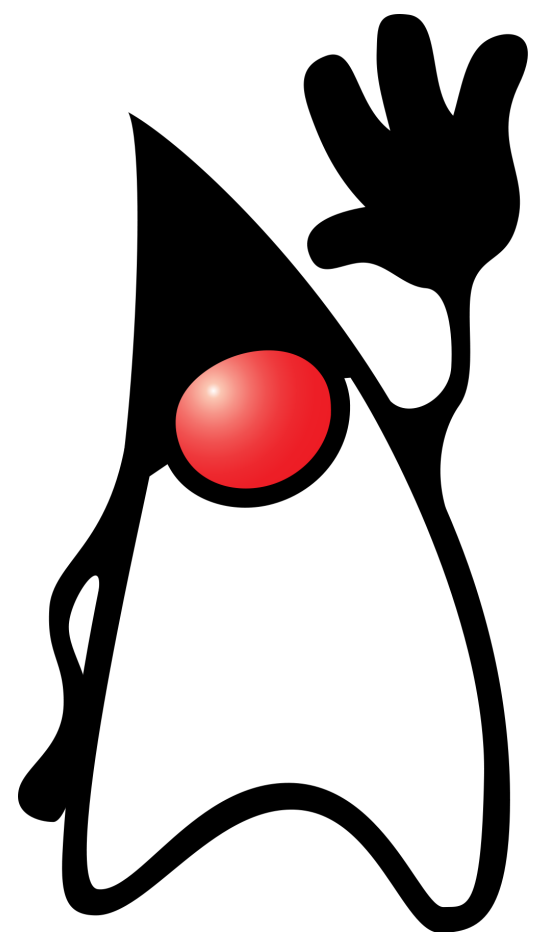


Running faster



Ahead of Time (AOT)

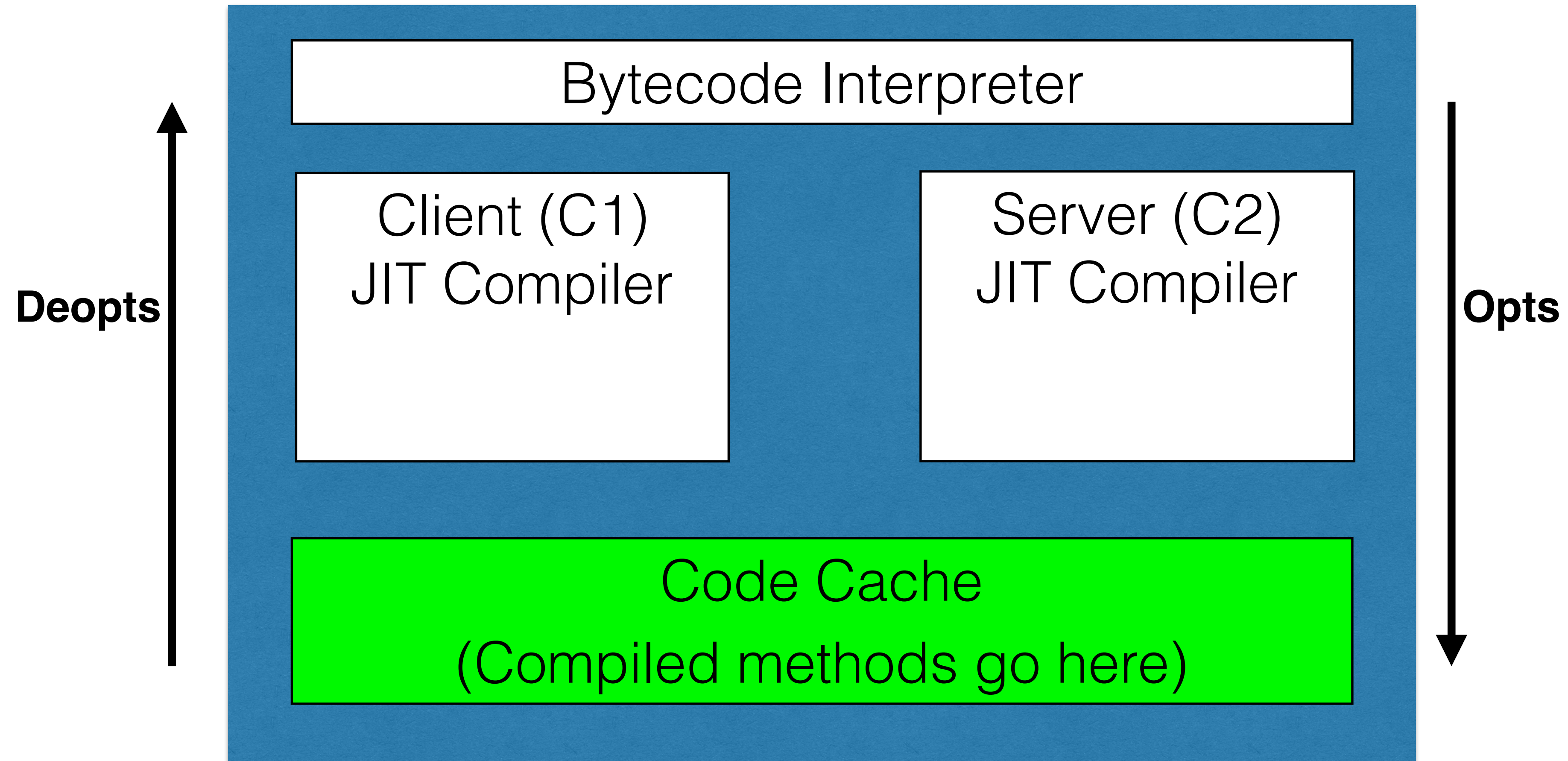
Produces native executable
Knowledge of target architecture
Full performance from the start



Just In Time (JIT)

Profiles running code
Adaptive optimisations
Takes time to build a profile

The HotSpot JVM



***Very tuneable. Such `-XX:+PrintFlagsFinal`. Wow!**

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal | \
egrep -i "compile|tier|cache|inline"
```

```

bool AlwaysCompileLoopMethods      = false      {product}
intx AutoBoxCacheMax                = 128        {C2 product}
bool C1ProfileInlinedCalls          = true        {C1 product}

intx CICompilerCount              := 3          {product}

bool CICompilerCountPerCPU          = true        {product}
uintx CodeCacheExpansionSize        = 65536       {pd product}
uintx CodeCacheMinimumFreeSpace     = 512000      {product}
ccstrlist CompileCommand            =              {product}
ccstr CompileCommandFile            =              {product}
ccstrlist CompileOnly               =              {product}

intx CompileThreshold            = 10000       {pd product}

bool CompilerThreadHintNoPreempt    = true        {product}
intx CompilerThreadPriority          = -1           {product}
intx CompilerThreadStackSize        = 0            {pd product}
bool DebugInlinedCalls              = true        {C2 diagnostic}
bool DontCompileHugeMethods         = true        {product}
bool EnableResourceManagementTLABCache = true      {product}
bool EnableSharedLookupCache        = true        {product}

intx FreqInlineSize              = 325         {pd product}

uintx G1ConcRSLogCacheSize          = 10          {product}
uintx IncreaseFirstTierCompileThresholdAt = 50      {product}
bool IncrementalInline              = true        {C2 product}
bool Inline                          = true        {product}
ccstr InlineDataFile                =              {product}
intx InlineSmallCode                = 2000        {pd product}
bool InlineSynchronizedMethods      = true        {C1 product}
intx MaxInlineLevel                 = 9           {product}

intx MaxInlineSize               = 35          {product}

intx MaxRecursiveInlineLevel        = 1           {product}
bool PrintCodeCache                 = false       {product}
bool PrintCodeCacheOnCompilation    = false       {product}
bool PrintTieredEvents              = false       {product}

uintx ReservedCodeCacheSize      = 251658240  {pd product}

intx Tier0BackedgeNotifyFreqLog     = 10         {product}
intx Tier0InvokeNotifyFreqLog       = 7           {product}
intx Tier0ProfilingStartPercentage  = 200        {product}
intx Tier23InlineNotifyFreqLog      = 20         {product}
intx Tier2BackEdgeThreshold         = 0           {product}
intx Tier2BackedgeNotifyFreqLog     = 14         {product}
intx Tier2CompileThreshold          = 0           {product}
intx Tier2InvokeNotifyFreqLog       = 11         {product}
intx Tier3BackEdgeThreshold         = 60000       {product}
intx Tier3BackedgeNotifyFreqLog     = 13         {product}
intx Tier3CompileThreshold          = 2000        {product}
intx Tier3DelayOff                  = 2           {product}
intx Tier3DelayOn                   = 5           {product}
intx Tier3InvocationThreshold       = 200        {product}
intx Tier3InvokeNotifyFreqLog       = 10         {product}
intx Tier3LoadFeedback              = 5           {product}
intx Tier3MinInvocationThreshold    = 100        {product}
intx Tier4BackEdgeThreshold         = 40000       {product}
intx Tier4CompileThreshold          = 15000      {product}
intx Tier4InvocationThreshold       = 5000       {product}
intx Tier4LoadFeedback              = 3           {product}
intx Tier4MinInvocationThreshold    = 60         {product}

bool TieredCompilation          = true        {pd product}

intx TieredCompileTaskTimeout        = 50         {product}
intx TieredRateUpdateMaxTime        = 25         {product}

```


HotSpot optimisations

lock coarsening

strength reduction

loop unrolling

branch prediction

range check elimination

inlining

CHA

dead code elimination

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

Compilation levels

Level	Description
0	Interpreter (does profiling)
1	C1
2	C1 + counters
3	C1 + counters + profiling
4	C2

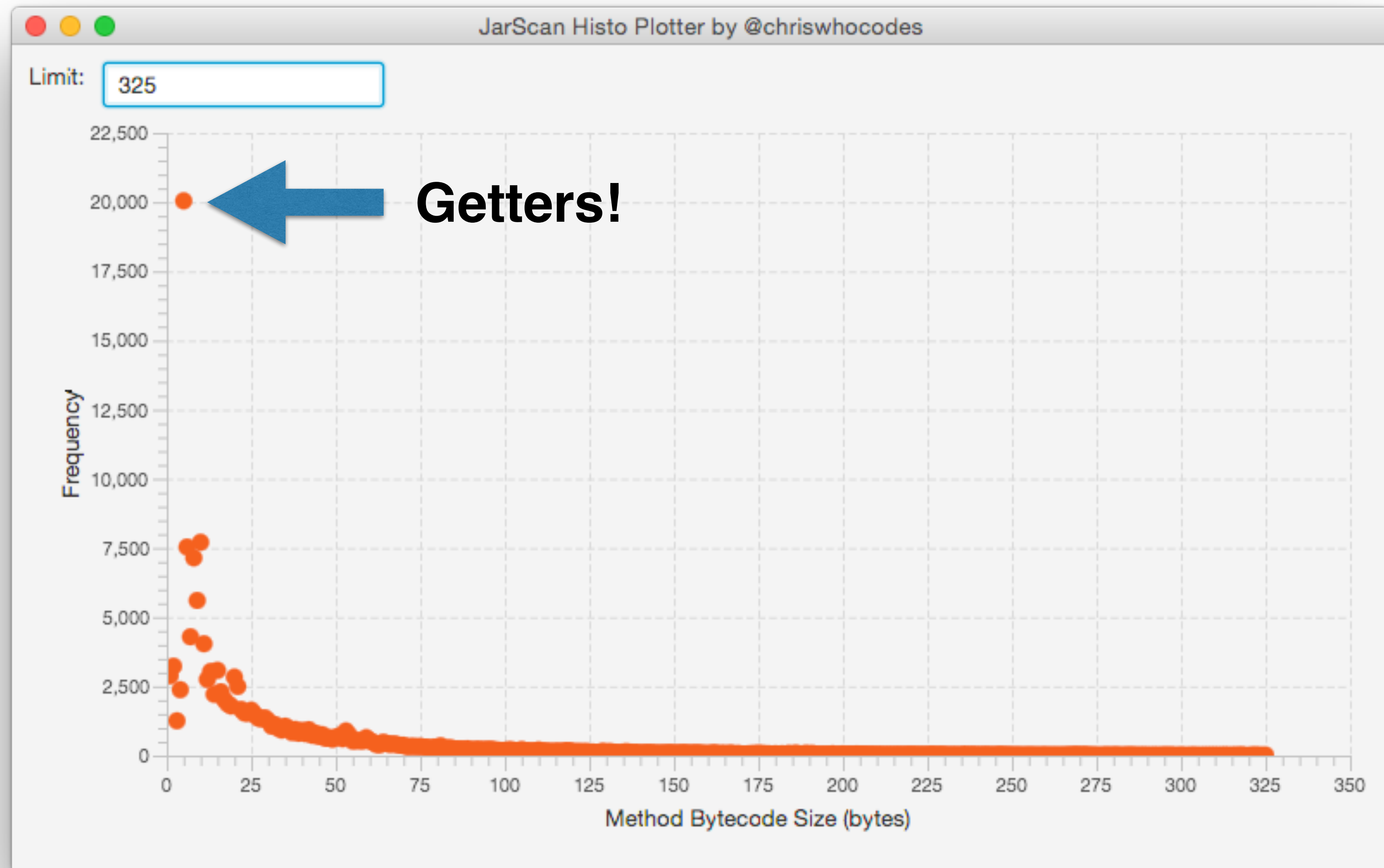
More info: <http://www.slideshare.net/maddocig/tiered>

Compilation patterns

Sequence	Explanation
0-3-4	Tiered Compilation
0-2-3-4	C2 queue busy?
0-3-1	Trivial method, profiling not needed
0-1	Getters?
0-4	No Tiered Compilation

Configure compiler threads with `-XX:CICompilerCount`

Trivial methods in the JDK



Code cache

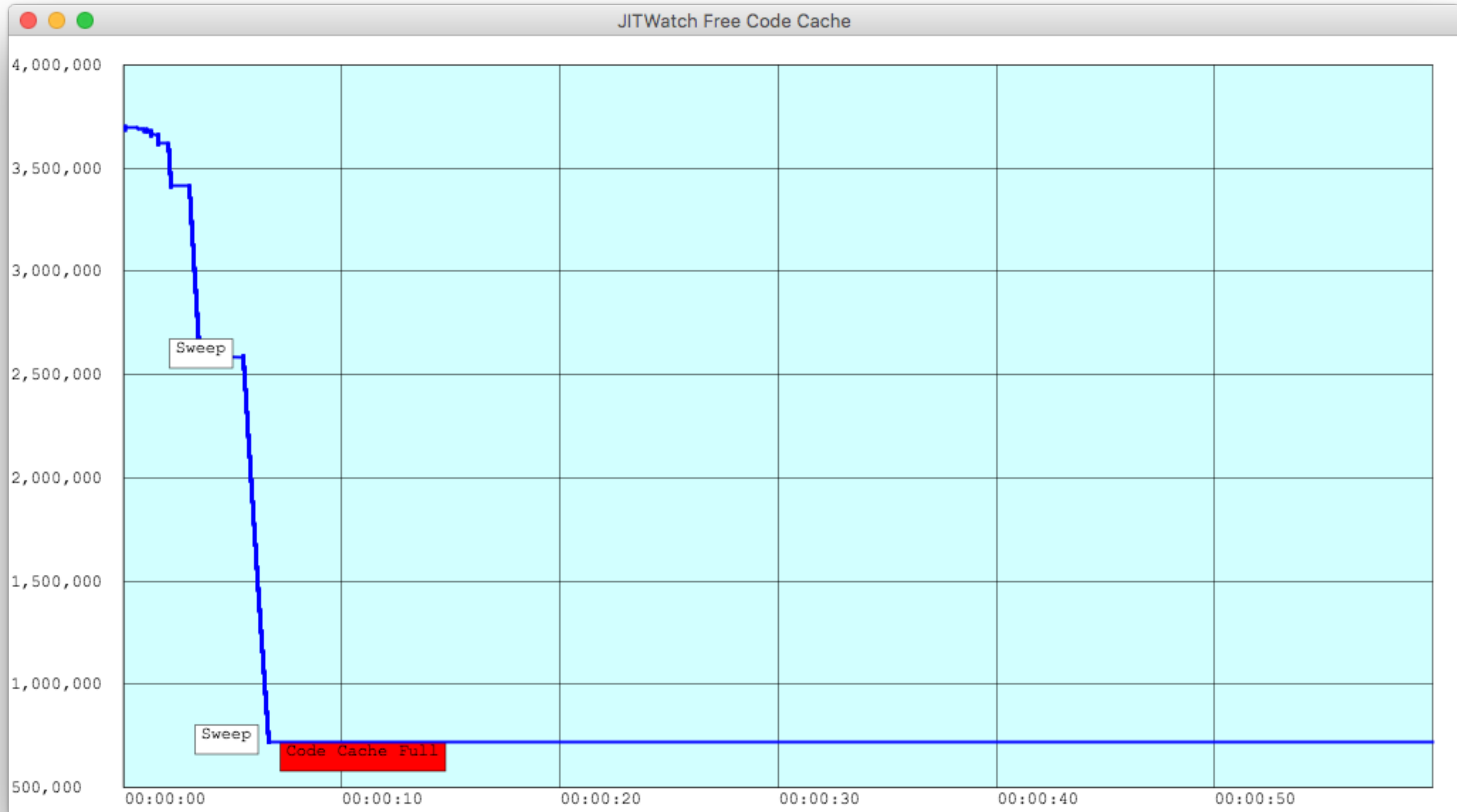
JVM region for JIT-compiled methods

Can run out of space

Can become fragmented

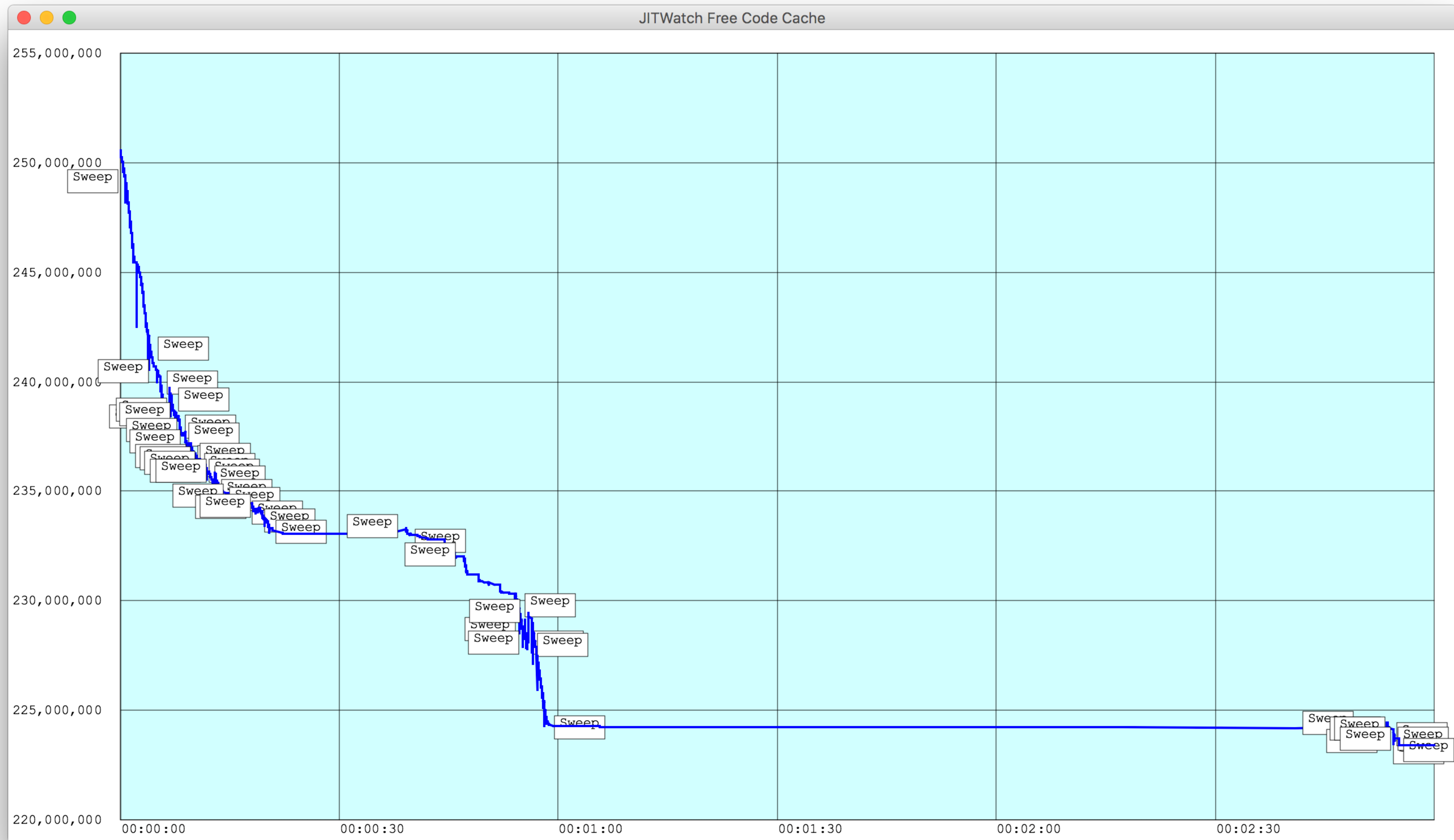
-XX:ReservedCodeCacheSize=<size>m

Code cache exhaustion



-XX:ReservedCodeCacheSize=4m

Sweeper activity



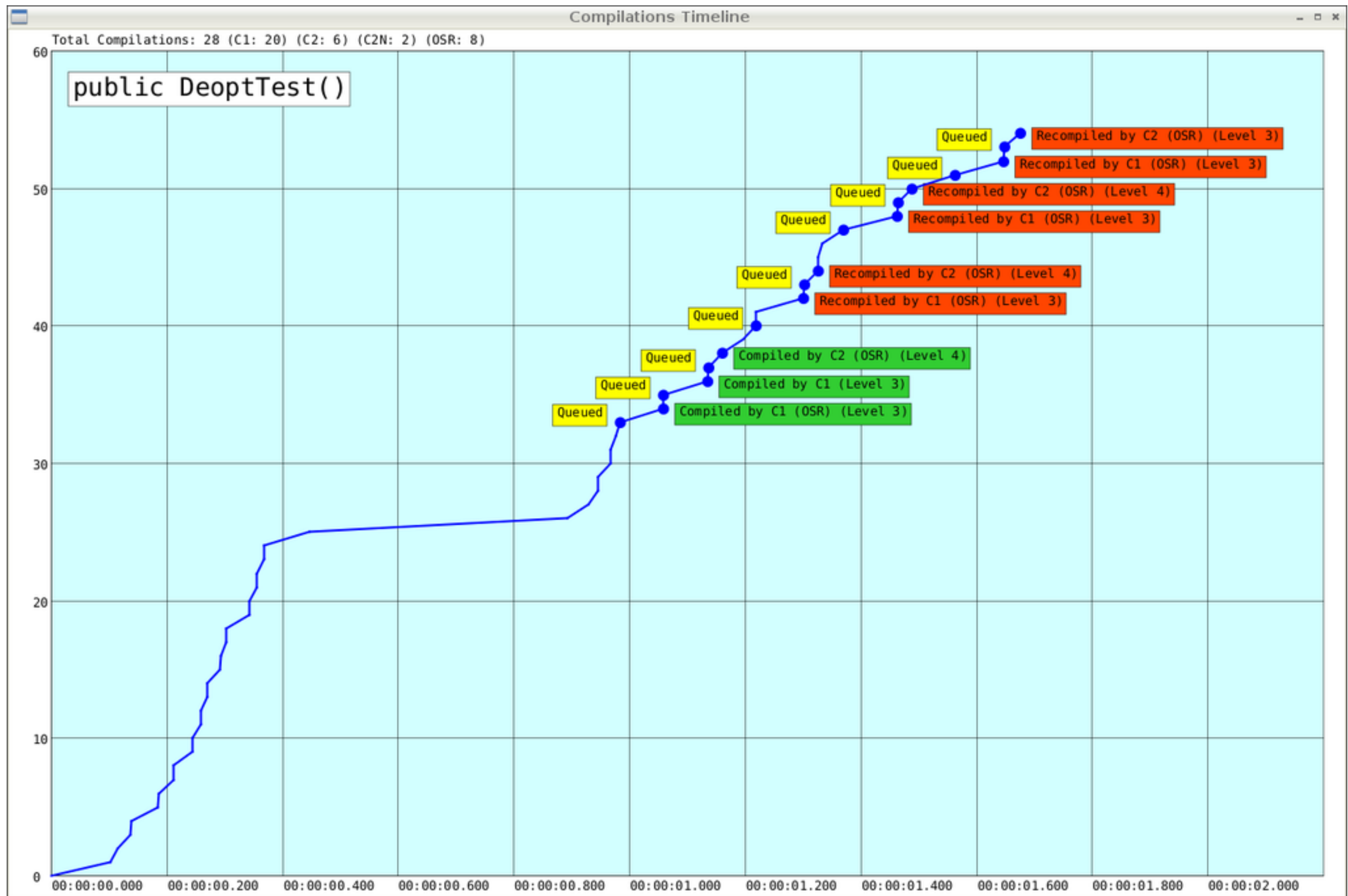
Guess again?

Many (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



Repeated deopts can cause poor performance

Logging the JIT

-XX:+UnlockDiagnosticVMOptions

-XX:+LogCompilation

-XX:+TraceClassLoading

-XX:+PrintAssembly

hsdis binary in jre/lib/amd64/server

I heard you like to grep?

```
<call method='739' count='9304' prof_factor='0.878029' inline='1' />
<inline_success reason='inline (hot)' />
<parse method='739' uses='8169' stamp='0.247'>
<parse_done nodes='253' live='246' memory='60712' stamp='0.247' />
</parse>
<bc code='50' bci='36' />
<uncommon_trap bci='36' reason='range_check' action='make_not_entrant' comment='range_check' />
<bc code='198' bci='39' />
<branch target_bci='84' taken='0' not_taken='9304' cnt='9304' prob='never' />
<uncommon_trap bci='39' reason='unreached' action='reinterpret' comment='taken never' />
<uncommon_trap bci='42' reason='predicate' action='maybe_recompile' />
<uncommon_trap bci='42' reason='loop_limit_check' action='maybe_recompile' />
<bc code='180' bci='43' />
<uncommon_trap bci='43' reason='null_check' action='maybe_recompile' />
<bc code='160' bci='47' />
<branch target_bci='76' taken='0' not_taken='9304' cnt='9304' prob='never' />
<bc code='165' bci='58' />
<branch target_bci='74' taken='9304' not_taken='0' cnt='9304' prob='always' />
<uncommon_trap bci='58' reason='unreached' action='reinterpret' comment='taken always' />
<bc code='198' bci='39' />
<branch target_bci='84' taken='0' not_taken='9304' cnt='9304' prob='never' />
<uncommon_trap bci='39' reason='unreached' action='reinterpret' comment='taken never' />
<parse_done nodes='384' live='369' memory='92008' stamp='0.247' />
</parse>
<bc code='192' bci='5' />
<class id='732' name='java/util/LinkedHashMap$Entry' flags='10' />
<dependency type='leaf_type' ctxk='732' />
<uncommon_trap bci='5' reason='null_check' action='make_not_entrant' />
<uncommon_trap bci='5' reason='class_check' action='maybe_recompile' />
<bc code='199' bci='10' />
<branch target_bci='15' taken='11050' not_taken='0' cnt='11050' prob='always' />
<bc code='182' bci='17' />
<type id='636' name='void' />
<method id='736' holder='732' name='readAccess' return='636' arguments='733' flags='0' bytes='25' compile_id='13' compiler='C2' icount='12027' />
<dependency type='unique_concrete_method' ctxk='736' />
<call method='736' count='11050' prof_factor='1' inline='1' />
<inline_success reason='inline (hot)' />
<parse method='736' uses='11050' stamp='0.247'>
<bc code='184' bci='6' />
<type id='628' name='boolean' />
<method id='751' holder='729' name='access$000' return='628' arguments='729' flags='4104' bytes='5' compile_id='14' compiler='C2' icount='12027' />
<call method='751' count='8728' prof_factor='0.918766' inline='1' />
<inline_success reason='accessor' />
<parse method='751' uses='8019' stamp='0.247'>
<parse_done nodes='449' live='431' memory='11119' stamp='0.247' />
</parse>
<bc code='153' bci='9' />
```

VISUALISE



ALL THE COMPILER DECISIONS!

JITWatch

Compilations (when, how)

Deoptimisations (why)

Inlining successes and failures

Escape analysis

Branch probabilities

Intrinsics used

Hot throws, stale tasks, and more!

Getting Started

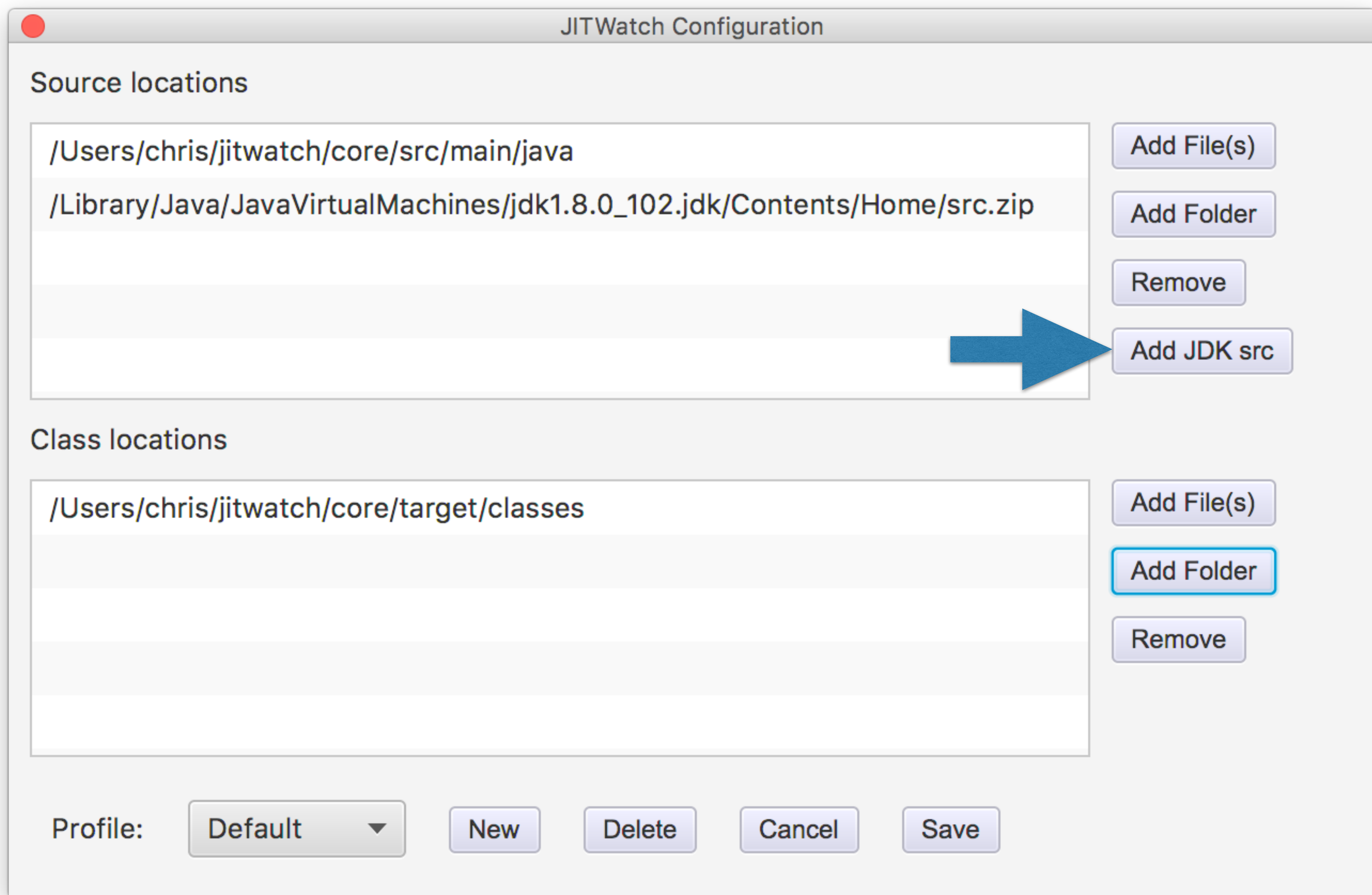
```
mvn clean compile
```

```
./makeDemoLogFile.sh
```

```
java version "1.8.0_102"  
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)  
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)  
VM Switches -XX:+UnlockDiagnosticVMOptions -XX:+TraceClassLoading -XX:  
+LogCompilation -XX:+PrintAssembly -XX:-UseCompressedOops  
Building example HotSpot log  
Java HotSpot(TM) 64-Bit Server VM warning: PrintAssembly is enabled;  
turning on DebugNonSafepoints to gain additional output  
Done
```

```
ls -lh hotspot_pid7127.log  
-rw-r--r--  1 chris  staff  12M  6 Sep 11:42 hotspot_pid7127.log
```

```
mvn exec:java
```



Mount source and class locations



JITWatch - HotSpot Compilation Inspector

Sandbox Open Log Start Stop Config Chart Stats Histo TopList Code Cache TriView Suggestions (196) OVCs

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

- ▼ Packages
 - ▶ ch
 - ▶ com
 - ▶ java
 - ▼ org
 - ▼ org.adoptopenjdk
 - ▼ org.adoptopenjdk.jitwatch
 - ▼ org.adoptopenjdk.jitwatch.demo
 - MakeHotSpotLog
 - ▶ sun

Queued	Compiled	Native Size	Compiler	Level
00:00:04.842	00:00:04.844	744	C1	Level 3
00:00:04.846	00:00:04.848	136	C2	Level 4

```
00:00:04.942 Compiled (C2) : private int org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:04.942 Queued : private int org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:04.943 Compiled (C2) : private int org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:05.449 Queued : public final int java.nio.Buffer.limit()
00:00:05.450 Compiled (C1) : public final int java.nio.Buffer.limit()
Finished reading log file.
Finding code suggestions.
Found 196 code suggestions.
```

Heap: 138/715M Errors (0) VM is Oracle Corporation 1.8.0_102

Class: Member:
 Source
 Bytecode
 Assembly
 Chain
 Journal
 LNT
 Mouseover

Bytecode size	Native size	Compile time
8	136	2ms

Source

```

308 {
309     ref = list1;
310 }
311 else
312 {
313     ref = list2;
314 }
315
316     ref.add(i);
317     // list2.add(i);
318 }
319
320     System.out.println("list sizes: ")
321 }
322
323
324 private long chainA1(long count)
325 {
326     return 1 + chainA2(count);
327 }
328
329 private long chainA2(long count)
330 {
331     return 2 + chainA3(count);
332 }
333
334 private long chainA3(long count)
335 {
336     return 3 + chainA4(count);
337 }
338
339 private long chainA4(long count)
340 {
341     // last link will not be inlined
342     return bigMethod(count, 4);
343 }
344

```

Bytecode (double click for JVM spec)

```

0: lconst_1
1: aload_0
2: lload_1
3: invokespecial #74 // Method chainA2:(J)J
6: ladd
7: lreturn

```

Class: org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog
 Method: chainA2
 JIT Compiled: Yes
 Inlined: Yes, inline (hot)
 Count: 5341
 iicount: 5823
 Bytes: 10
 Prof factor: 1

Ctrl-click to inspect this method
 Backspace to return

TriView screen

Assembly Labels #2 (C2 / Level 4)

```

# {method} {0x00000001980add30} 'chainA1' '(J)J' in 'org/adoptopenjdk/jitwatch/'
# this:      rsi:rsi   = 'org/adoptopenjdk/jitwatch/demo/MakeHotSpotLog'
# parm0:     rdx:rdx   = long
#           [sp+0x20] (sp of caller)
0x000000010401ed20: cmp 0x8(%rsi),%rax
0x000000010401ed24: jne 0x0000000103e24e20 ; {runtime_call}
0x000000010401ed2a: xchg %ax,%ax
0x000000010401ed2c: nopl 0x0(%rax)
[Verified Entry Point]
0x000000010401ed30: mov %eax,-0x14000(%rsp)
0x000000010401ed37: push %rbp
0x000000010401ed38: sub $0x10,%rsp ;*synchronization entry
; - org.adoptopenjdk.jitwatch.demo.MakeHotS
0x000000010401ed3c: mov $0x4,%ecx
0x000000010401ed41: xchg %ax,%ax
0x000000010401ed43: callq 0x0000000103e25020 ; OopMap{off=40}
; *invokespecial bigMethod
; - org.adoptopenjdk.jitwatch.dem
; - org.adoptopenjdk.jitwatch.dem
; - org.adoptopenjdk.jitwatch.dem
; - org.adoptopenjdk.jitwatch.dem
; {optimized virtual_call}
0x000000010401ed4c: add $0x6,%rax ;*ladd
; - org.adoptopenjdk.jitwatch.demo.MakeHotSp
0x000000010401ed51: test %eax,-0x1a2fd57(%rip) # 0x00000001025ef000
; {poll_return} *** SAFEPOINT
0x000000010401ed57: retq ;*invokespecial bigMethod
; - org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog::ch
; - org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog::ch
; - org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog::ch
; - org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog::ch
0x000000010401ed58: mov %rax,%rsi
0x000000010401ed5b: add $0x10,%rsp
0x000000010401ed5f: pop %rbp
0x000000010401ed60: jmpq 0x0000000103ee0920 ; {runtime_call}

```


invokespecial

Operation

Invoke instance method; special handling for superclass, private, and instance initialization method invocations

Format

```
invokespecial  
indexbyte1  
indexbyte2
```

Forms

invokespecial = 183 (0xb7)

Operand Stack

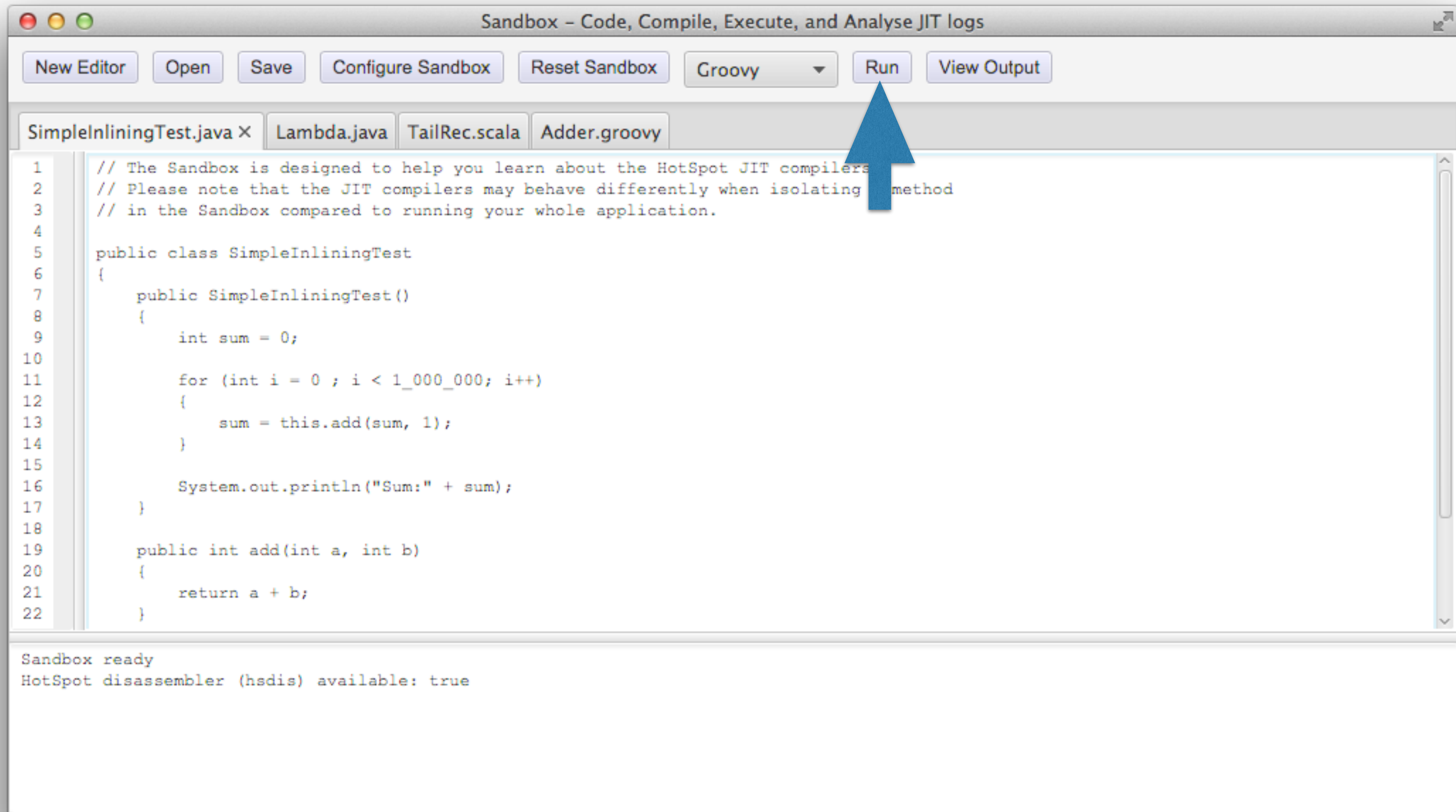
..., *objectref*, [*arg1*, [*arg2* ...]] →

...

Description

The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class ([§2.6](#)), where the value of the index is $(\text{indexbyte1} \ll 8) \mid \text{indexbyte2}$. The run-time constant pool item at that index must be a symbolic reference to a method ([§5.1](#)), which gives the name and descriptor ([§4.3.3](#)) of the method as well as a symbolic reference to the class in which the method is to be found. The named method is resolved ([§5.4.3.3](#)). Finally, if the resolved method is `protected` ([§4.6](#)), and it is a member of a superclass of the current class, and the method is not declared in the same run-time package ([§5.3](#)) as the current class, then the class of *objectref* must be either the current class or a subclass of the current class.

Sandbox Mode



The screenshot displays the 'Sandbox - Code, Compile, Execute, and Analyse JIT logs' application window. The interface includes a menu bar with options: 'New Editor', 'Open', 'Save', 'Configure Sandbox', 'Reset Sandbox', 'Groovy' (with a dropdown arrow), 'Run', and 'View Output'. Below the menu bar, there are tabs for 'SimpleInliningTest.java ×', 'Lambda.java', 'TailRec.scala', and 'Adder.groovy'. The main editor area shows the following Java code:

```
1 // The Sandbox is designed to help you learn about the HotSpot JIT compilers
2 // Please note that the JIT compilers may behave differently when isolating method
3 // in the Sandbox compared to running your whole application.
4
5 public class SimpleInliningTest
6 {
7     public SimpleInliningTest()
8     {
9         int sum = 0;
10
11         for (int i = 0 ; i < 1_000_000; i++)
12         {
13             sum = this.add(sum, 1);
14         }
15
16         System.out.println("Sum:" + sum);
17     }
18
19     public int add(int a, int b)
20     {
21         return a + b;
22     }
23 }
```

A blue arrow points to the 'Run' button in the top right of the editor area. At the bottom of the window, a console area displays the output: 'Sandbox ready' and 'HotSpot disassembler (hsdis) available: true'.

Sandbox Config

Sandbox Configuration

Compile and Runtime Classpath

Configure VM Languages

Show Disassembly AT&T syntax Intel syntax

Tiered Compilation: VM Default Always Never

Compressed Oops: VM Default Always Never

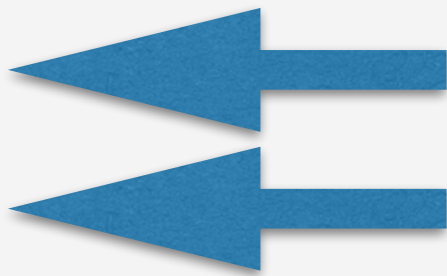
Background JIT: VM Default Always Never

On Stack Replacement: VM Default Always Never

Disable Inlining FreqInlineSize: MaxInlineSize:

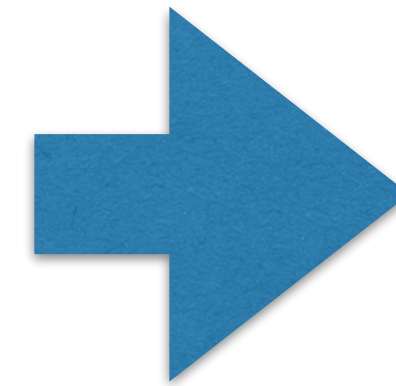
Compile Threshold:

Extra VM switches:



Inlining

```
int result = add(a, b);  
  
public int add(int x, int y) {  
    return x + y;  
}
```



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

- Copy the body of the callee method into the call site
- Eliminates the cost of method dispatch
- The “Gateway Optimisation”

Inlining Limits

Increases size of compiled code

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

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

Inlining Failure Modes

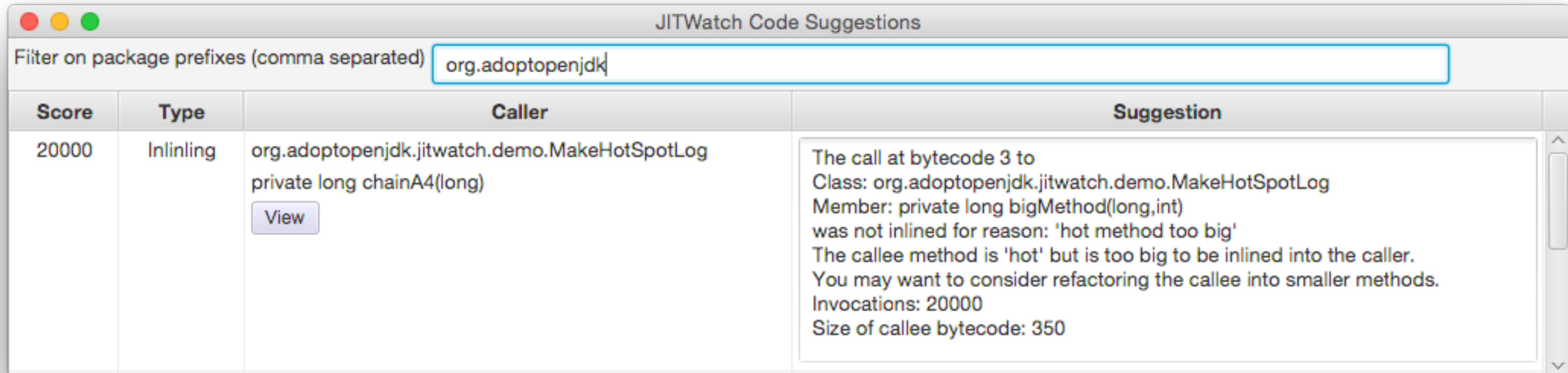
JITWatch TopLists

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!

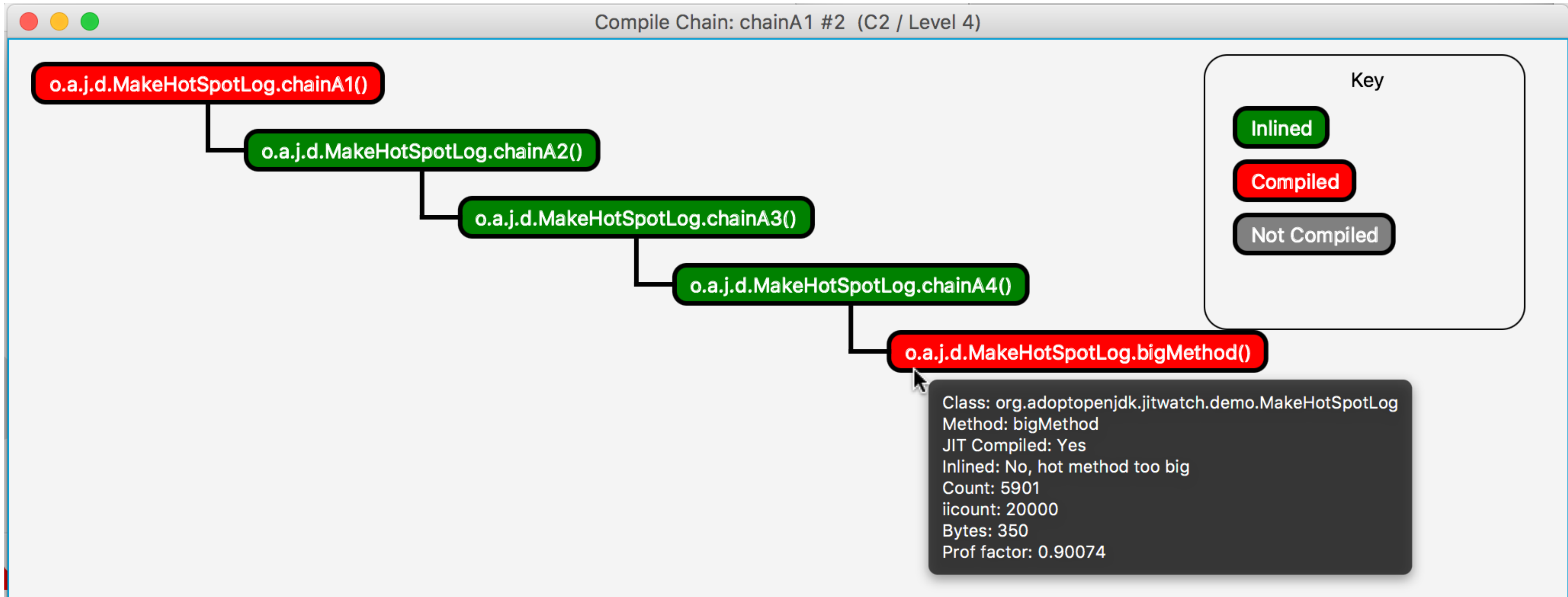
Inlining Suggestions



The screenshot shows a window titled "JITWatch Code Suggestions". At the top, there is a filter input field with the text "Filter on package prefixes (comma separated)" and the value "org.adoptopenjdk". Below the filter is a table with four columns: "Score", "Type", "Caller", and "Suggestion". The table contains one row with the following data:

Score	Type	Caller	Suggestion
20000	Inlining	org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog private long chainA4(long) View	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

Compile Chain



Look out for inlining failures or deep chains in hot code

JarScan Tool

Static bytecode analysis

Identifies methods above inlining threshold

>3500 above-threshold methods in JDK 8

- String.split
- **String.toUpperCase / toLowerCase**
- Parts of java.util.ComparableTimSort

Large JDK methods

```
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 {

```

java.lang.String.toUpperCase()

439 bytes of bytecode

char[] can change size

Too big for inlining

Specialised for ASCII

```
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);
}
```

69 bytes of bytecode

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 the ops/second

UpperCase.convertString()

String.toUpperCase()

j.u.Locale.getDefault()

j.l.String.toUpperCase()

Class: java.lang.String
Method: toUpperCase
JIT Compiled: Yes
Inlined: No, hot method too big
Count: 40960
iicount: 723
Bytes: 439
Prof factor: 1

Key

- Inlined
- Compiled
- Virtual Call
- Not Compiled

UpperCase.convertCustom()

toUpperCaseASCII()

UpperCase.toUpperCaseASCII()

j.l.String.length()

j.l.String.charAt()

j.l.String()

j.l.Object()

j.u.Arrays.copyOf()

Key

- Inlined
- Compiled
- Virtual Call
- Not Compiled

Assertions

Not enabled by default (requires **-ea**)

Core-lib assertion code baked into bytecode

Counted in inlining budget

Can push a method over the inlining limit!

j.u.ComparableTimSort

Used in Arrays.sort()

Method	Bytecode size with assertions	Bytecode 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 Morphism

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;

        final int maxImplementations = 2; // 2 OK, 3 Not inlined

        for (int i = 0; i < 100_000; i++) {
            switch(i % maxImplementations) {
                case 0: coin = nickel; break;
                case 1: coin = dime; break;
                case 2: coin = quarter; break;
            }

            coin.deposit(); // callsite in question
        }

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


Bimorphic

TriView - Source, Bytecode, Assembly Viewer - JITWatch

Class: PolymorphismTest Member: public void PolymorphismTest()

Source Bytecode Assembly Chain Journal LNT Mouseover

Bytecode size: 132 Native size: 824 Compile time: 12ms

Source	Bytecode (double click for JVM spec)	Assembly
31 } 32 } 33 34 public PolymorphismTest() 35 { 36 Coin nickel = new Nickel(); 37 Coin dime = new Dime(); 38 Coin quarter = new Quarter(); 39 40 Coin coin = null; 41 42 // change the variable 43 // 2 = bimorphic dispatch 44 // 3 = megamorphic dispatch 45 46 final int maxImplementationCount = 100000; 47 48 for (int i = 0; i < 100000; i++) 49 { 50 switch(i % maxImplementationCount) 51 { 52 case 0: coin = nickel; 53 case 1: coin = dime; 54 case 2: coin = quarter; 55 } 56 57 coin.deposit(); 58 } 59 60 System.out.println("mor"); 61 } 62 63 public static void main(String[] args) 64 { 65 new PolymorphismTest(); 66 } 67 }	37: iconst_0 38: istore 6 40: iload 6 42: ldc #8 // int 100000 44: if_icmpge #8, #9, L0011 47: iload 49: iconst_2 50: irem 51: tableswitch { 0:76 1:82 2:88 default:91 } 76: aload_1 77: astore 79: goto L0011 82: aload_2 83: astore 85: goto L0011 88: aload_3 89: astore 91: aload 93: invokeinterface #9, 1 // InterfaceMethod PolymorphismTest\$Coin.deposit:()V 98: iinc 6, 1 101: goto L0011 104: getstatic #10 // Field java/lang/System.out:Ljava/io/PrintStream; 107: new #11 // class java/lang/StringBuilder 110: dup 111: invokespecial #12 // Method java/lang/StringBuilder."<init>":()V 114: ldc #13 // String moneyBox: 116: invokevirtual #14 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder; 119: getstatic #15 // Field moneyBox:I 122: invokevirtual #16 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder; 125: invokevirtual #17 // Method java/lang/StringBuilder.toString:()Ljava/lang/String; 128: invokevirtual #18 // Method java/io/PrintStream.println:(Ljava/lang/String;)V 131: return	; - PolymorphismTest::<init> 0x0000000000000000: add \$0x5,%r10d 0x000000000000000e: mov %r10d,0xa0(%rbp) ;*putstatic money ; - PolymorphismTest ; - PolymorphismTest 0x00000000109e1b015: jmp L0011 L0017: mov \$0xffffffff6,%esi 0x00000000109e1b01c: data32 xchg %ax,%ax 0x00000000109e1b01f: callq 0x00000000109da61a0 ; OopMap{off=...} ; - PolymorphismTest 0x00000000109e1b024: callq 0x00000000109285538 ; {runtime_...} L0018: mov %r11d,%r13d L0019: mov %r13d,%r11d L001a: mov \$0xffffffffc6,%esi 0x00000000109e1b034: mov %r8,%rbp 0x00000000109e1b037: mov %rcx,(%rsp) 0x00000000109e1b03b: mov %r11d,0x10(%rsp) 0x00000000109e1b040: mov %rbx,0x18(%rsp) 0x00000000109e1b045: mov %r14,0x20(%rsp) 0x00000000109e1b04a: nop 0x00000000109e1b04b: callq 0x00000000109da61a0 ; OopMap{rbp=...} ; - PolymorphismTest ; {runtime_...} 0x00000000109e1b050: callq 0x00000000109285538 ; {runtime_...} L001b: cmp \$0x186a0,%r11d 0x00000000109e1b05c: jge L0023 0x00000000109e1b05e: jmp L001e ;*invokeinterface deposit ; - PolymorphismTest add \$0xa,%r10d 0x00000000109e1b064: mov %r10d,0xa0(%rbp) ;*synchronization ; - PolymorphismTest L001d: mov %r11d,%r13d 0x00000000109e1b06e: inc %r13d ;*iinc ; - PolymorphismTest::<init>

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

Bimorphic

```
#8 // int 100000
Class: PolymorphismTest$Dime
Method: deposit
JIT Compiled: No
Inlined: Yes, inline (hot)
Count: 11264
iicount: 7281
Bytes: 10
Prof factor: 1

Class: PolymorphismTest$Nickel
Method: deposit
JIT Compiled: No
Inlined: Yes, inline (hot)
Count: 11264
iicount: 7282
Bytes: 9
Prof factor: 1

Uncommon trap (reason:null_check, action:maybe_recompile)
Uncommon trap (reason:bimorphic, action:maybe_recompile)

Ctrl-click to inspect this method
Backspace to return
```



Class: PolymorphismTest

Member: public void PolymorphismTest()

Source Bytecode Assembly Mouseover

Bytecode size: 132 Native size: 536 Compile time: 4ms

```

31     }
32   }
33
34   public PolymorphismTest()
35   {
36     Coin nickel = new Coin();
37     Coin dime = new Dime();
38     Coin quarter = new Quarter();
39
40     Coin coin = null;
41
42     // change the variable maxImplementations
43     // 2 = bimorphic dispatch
44     // 3 = megamorphic dispatch
45
46     final int maxImplementations = 3;
47
48     for (int i = 0; i < 100000; i++)
49     {
50       switch(i % maxImplementations)
51       {
52         case 0: coin = nickel; break;
53         case 1: coin = dime; break;
54         case 2: coin = quarter; break;
55       }
56
57       coin.deposit();
58     }
59
60     System.out.println("moneyBox:" + moneyBox);
61   }
62
63   public static void main(String[] args)
64   {
65     new PolymorphismTest();
66   }
67 }
    
```

Megamorphic



```

35: istore      5
37: iconst_0
38: istore      6
40: iload       6
42: ldc         #8 // int 100000
44: if_icmpge   104
47: iload       6
49: iconst_3
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
93: invokeinterface #9, 1 // InterfaceMethod PolymorphismTest$Coin.deposit()
98: iinc        6, 1
101: goto
104: getstatic   #10 // Field java/lang/System.out:Ljava/io/PrintStream;
107: new         #11 // class java/lang/StringBuilder
110: dup
111: invokespecial #12 // Method java/lang/StringBuilder.<init>:()V
114: ldc         #13 // String moneyBox:
116: invokevirtual #14 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
119: getstatic   #15 // Field moneyBox:I
122: invokevirtual #16 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
125: invokevirtual #17 // Method java/lang/StringBuilder.toString():Ljava/lang/String;
128: invokevirtual #18 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
    
```

Virtual call, not inlined
 Ctrl-click to inspect this method
 Backspace to return

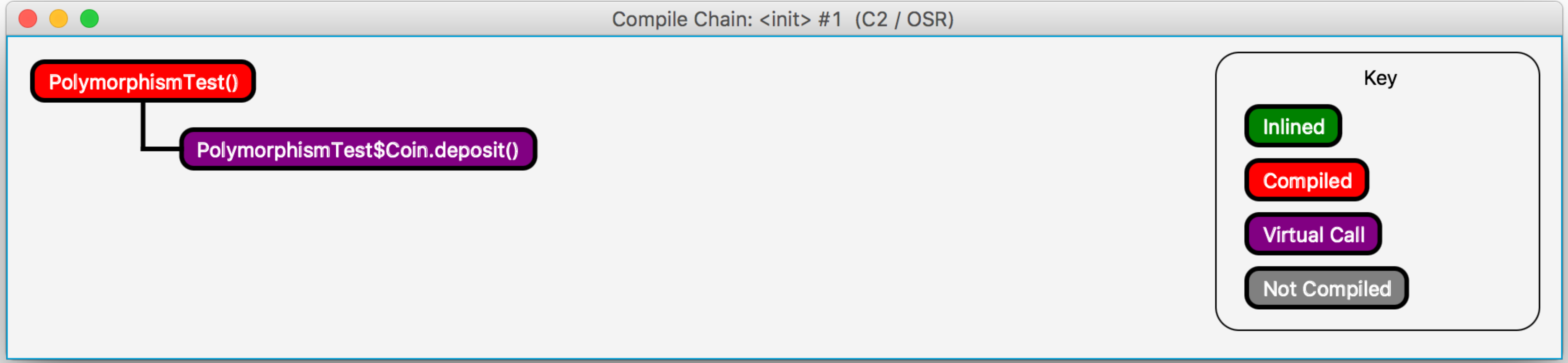
```

0x0000000111a7a330: test %r8,%r8
0x0000000111a7a333: je L000b
0x0000000111a7a339: mov 0x8(%r8),%r10
0x0000000111a7a33d: movabs $0x199aadd0,%r11 ; {metadata
0x0000000111a7a347: cmp %r11,%r10
0x0000000111a7a34a: jne L000e ;*iload
                                ; - PolymorphismTest::<init>
0x0000000111a7a350: jmp L0004
                                L0002: mov %rdi,0x18(%rsp) ;*aload
                                ; - PolymorphismTest::<init>
                                L0003: mov %rdi,0x10(%rsp)
0x0000000111a7a35c: mov %r9,0x8(%rsp)
0x0000000111a7a361: mov %r8,(%rsp)
0x0000000111a7a365: mov %r13d,%ebp ;*tableswitch
                                ; - PolymorphismTest::<init>
0x0000000111a7a368: mov 0x18(%rsp),%rsi
0x0000000111a7a36d: movabs $0xffffffffffffffff,%rax
0x0000000111a7a377: callq 0x0000000111a46220 ; OopMap{[0
                                ;*invokeint
                                ; - PolymorphismTest::<init>
                                ; {virtual
0x0000000111a7a37c: inc %ebp ;*iinc
                                ; - PolymorphismTest::<init>
0x0000000111a7a37e: mov 0x18(%rsp),%rbx
0x0000000111a7a383: mov %ebp,%r13d
0x0000000111a7a386: mov (%rsp),%r8
0x0000000111a7a38a: mov 0x8(%rsp),%r9
0x0000000111a7a38f: mov 0x10(%rsp),%rdi ;*iload
                                ; - PolymorphismTest::<init>
                                L0004: cmp $0x186a0,%r13d
0x0000000111a7a39b: jge L0008 ;*if_icmpge
                                ; - PolymorphismTest::<init>
0x0000000111a7a39d: movslq %r13d,%r10
0x0000000111a7a3a0: mov %r13d,%r11d
0x0000000111a7a3a3: sar $0x1f,%r11d
0x0000000111a7a3a7: imul $0x55555556,%r10,%r10
0x0000000111a7a3ae: sar $0x20,%r10
    
```


Megamorphic

```
91: aload          4
93: invokeinterface #9, 1// InterfaceMethod PolymorphismTest$Coin.de
98: iinc           6, 1
101: goto
104: getstatic      #10 // Field java/lang/System.out:Ljava/io/Print
107: new
110: dup
```

Virtual call, not inlined
Ctrl-click to inspect this method
Backspace to return



Escape Analysis



Scope-based optimisations

Eliminate heap allocations

Lock elision

NoEscape

```
public long noEscape ()
{
    long sum = 0;

    for (int i=0; i<BIG; i++)
    {
        MyObj foo = new MyObj(i);

        sum += foo.bar();
    }

    return sum;
}
```

Object foo doesn't escape the loop scope.

ArgEscape

```
public long argEscape ()
{
    long sum = 0;

    for (int i=0; i<BIG; i++)
    {
        MyObj foo = new MyObj(i);

        sum += extBar(foo);
    }

    return sum;
}
```

Object foo escapes loop scope by passing as arg to extBar().

Avoid heap allocations

NoEscape objects are “exploded”

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 val;

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

    public boolean equals(EscapeTest et) { return this.val == et.val; }

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

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

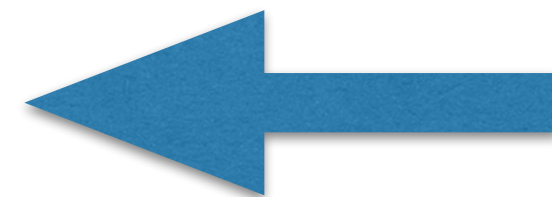
        for (int i = 0; i < 100_000_000; i++) {
            int v1 = random.nextBoolean() ? 1 : 0;
            int v2 = random.nextBoolean() ? 1 : 0;

            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 prevents ArgEscape of e2

Hot loop allocations

With Escape Analysis

100m loops. No GCs in 2.4s

```
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 [0x00000007bd580000,0x00000007bd580000,0x00000007c0000000)
  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

100m loops. 10 minor GCs in 2.4s

```
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:

Member:

Source Bytecode Assembly Mouseover

Bytecode size: Native size: Compile time:

Source

```

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

Bytecode (double click for JVM spec)

```

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.nextBoolean:()Z
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.nextBoolean:()Z
35: ifeq 42
38: iconst_1
39: goto 43
42: iconst_0
43: istore 4
45: new #7 // class EscapeTest
48: dup
49: iload_3
50: invokespecial #8 // Method "<init>":(I)V
53: astore 5
55: 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
    
```



```

Class: EscapeTest
Method: equals
JIT Compiled: Yes
Inlined: Yes, inline (hot)
Count: 40960
iicount: 46336
Bytes: 17
Prof factor: 1

Ctrl-click to inspect this method
Backspace to return
    
```


Branch prediction

```
public class BranchPrediction {
    public BranchPrediction() {
        int a = 0, b = 0;

        Random random = new Random();

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

        System.out.println(a + "/" + b);
    }

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


Filter on package prefixes (comma separated)

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: Member:

Source Bytecode Assembly Chain Journal LNT Mouseover

Bytecode size: **78** Native size: 376 Compile time: 5ms

Assembly Labels #1 (C2 / OSR)

Source

```
1 import java.util.Random;
2 public class BranchPrediction {
3     public BranchPrediction() {
4         int a = 0, b = 0;
5
6         Random random = new Random();
7
8         for (int i = 0; i < 1_000_000; i++) {
9             if (random.nextBoolean())
10                a++;
11            else
12                b++;
13        }
14
15        System.out.println(a + "/" + b);
16    }
17
18    public static void main(String[] args) {
19        new BranchPrediction();
20    }
21 }
```

Bytecode (double click for JVM spec)

```
0: aload_0
1: invokespecial #1 // Method java/lang/Object."<init>":()V
4: iconst_0
5: istore_1
6: iconst_0
7: istore_2
8: new #2 // class java/util/Random
11: dup
12: invokespecial #3 // Method java/util/Random."<init>":()V
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/Random.nextBoolean:()Z
30: ifeq 39
33: iinc
36: goto
39: iinc
42: iinc
45: goto 19
```

Assembly

```
0x000000010c210d05: lock cmpxchg %r8,0x10(%r11)
0x000000010c210d0b: sete %r10b
0x000000010c210d0f: movzbl %r10b,%r10d ;*invokevirtual c
; - java.util.co
; - BranchPredict
0x000000010c210d13: test %r10d,%r10d
0x000000010c210d16: je L0003 ;*ifeq
; - java.util.Random::next(
; - java.util.Random::nextI
; - BranchPrediction::<init
0x000000010c210d18: shr $0x2f,%rdi
0x000000010c210d1c: and $0x1,%rdi
0x000000010c210d20: mov %edi,%r11d
0x000000010c210d23: test %r11d,%r11d
0x000000010c210d26: jne L0000 ;*ifeq
; - BranchPrediction::<in
0x000000010c210d28: inc %r14d ;*iinc
; - BranchPrediction::<in
0x000000010c210d2b: jmp L0001
L0003: mov $0xffffffff,%r10d
```

Count: 11264
Branch taken: 5574
Branch not taken: 5690
Taken Probability: 0.494851

Intrinsics



Highly optimised native implementations

Use features of target CPU

Intrinsics exist for methods in
Math, Unsafe, System, Class, Arrays, String,
StringBuilder, AESCrypt, ...

Full list in

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

Intrinsics

Math.log10 (double) is 2 instructions on x86_64

from: hotspot/src/cpu/x86/vm/x86_64.ad

```
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));

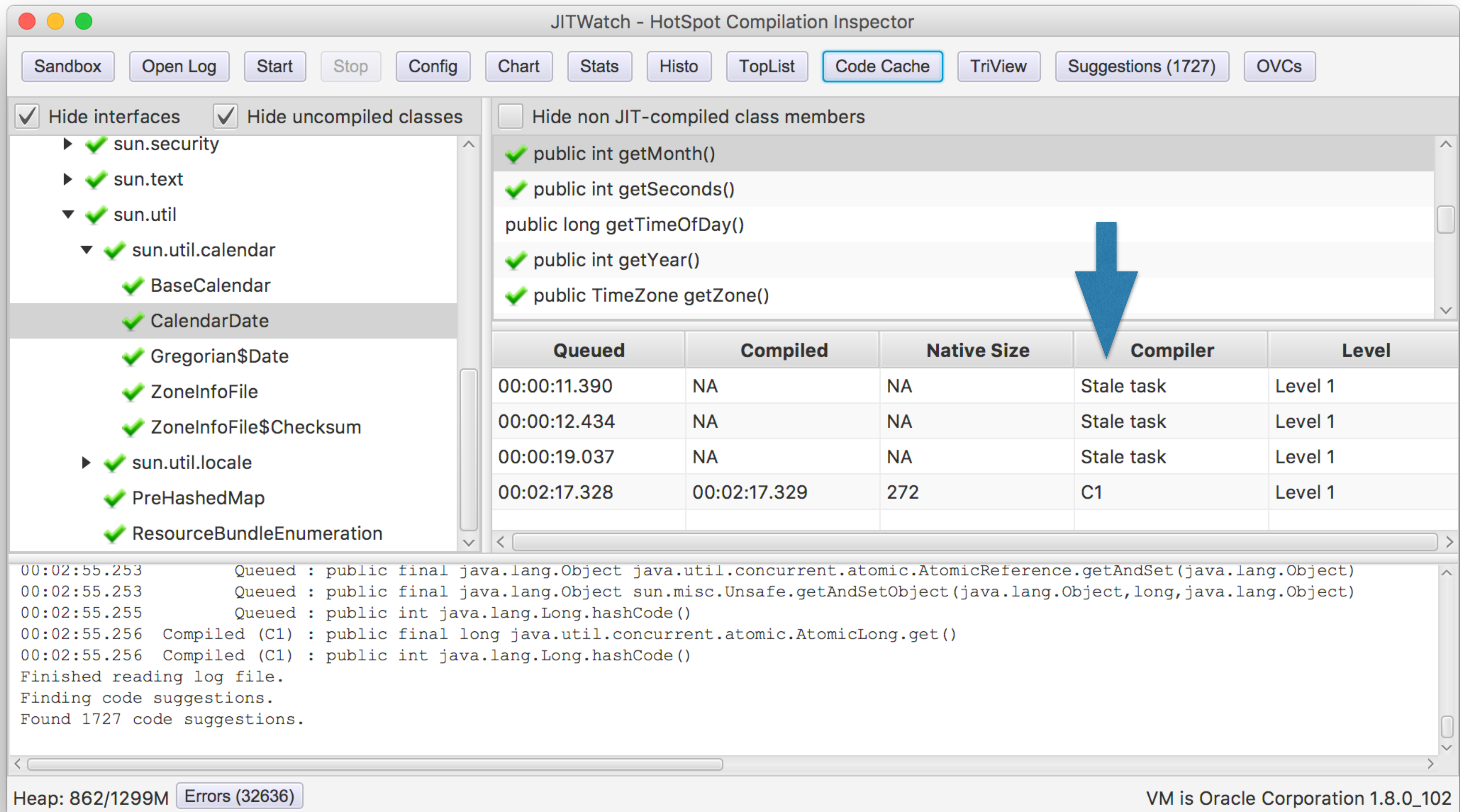
  ins_pipe( pipe_slow );
%}
```

JITWatch TopLists

Most-used Intrinsic

Count	Intrinsic
64	java.lang.System.arraycopy => _arraycopy
45	java.lang.Math.min => _min
25	java.lang.Object.hashCode => _hashCode
23	java.lang.Object.getClass => _getClass
22	java.lang.ref.Reference.get => _Reference_get
18	java.lang.String.equals => _equals
10	java.util.Arrays.copyOf => _copyOf
9	sun.misc.Unsafe.compareAndSwapObject => _compareAndSwapObject
8	java.lang.Object.clone => _clone
6	java.lang.Thread.currentThread => _currentThread
5	java.lang.Math.max => _max
3	sun.reflect.Reflection.getClassAccessFlags => _getClassAccessFlags
3	java.lang.Class.getComponentType => _getComponentType
3	sun.misc.Unsafe.compareAndSwapInt => _compareAndSwapInt
3	java.util.zip.CRC32.updateBytes => _updateBytesCRC32
3	java.lang.String.compareTo => _compareTo
3	sun.misc.Unsafe.getObjectVolatile => _getObjectVolatile
3	java.lang.reflect.Array.newInstance => _newInstance
2	java.lang.System.identityHashCode => _identityHashCode
2	sun.misc.Unsafe.putObject => _putObject
2	java.lang.System.nanoTime => _nanoTime

Stale tasks



JITWatch - HotSpot Compilation Inspector

Sandbox Open Log Start Stop Config Chart Stats Histo TopList Code Cache TriView Suggestions (1727) OVCs

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

- ▶ sun.security
- ▶ sun.text
- ▼ sun.util
 - ▼ sun.util.calendar
 - BaseCalendar
 - CalendarDate
 - Gregorian\$Date
 - ZoneInfoFile
 - ZoneInfoFile\$Checksum
 - ▶ sun.util.locale
 - PreHashedMap
 - ResourceBundleEnumeration

Queued	Compiled	Native Size	Compiler	Level
00:00:11.390	NA	NA	Stale task	Level 1
00:00:12.434	NA	NA	Stale task	Level 1
00:00:19.037	NA	NA	Stale task	Level 1
00:02:17.328	00:02:17.329	272	C1	Level 1

```
00:02:55.253 Queued : public final java.lang.Object java.util.concurrent.atomic.AtomicReference.getAndSet (java.lang.Object)
00:02:55.253 Queued : public final java.lang.Object sun.misc.Unsafe.getAndSetObject (java.lang.Object, long, java.lang.Object)
00:02:55.255 Queued : public int java.lang.Long.hashCode ()
00:02:55.256 Compiled (C1) : public final long java.util.concurrent.atomic.AtomicLong.get ()
00:02:55.256 Compiled (C1) : public int java.lang.Long.hashCode ()
Finished reading log file.
Finding code suggestions.
Found 1727 code suggestions.
```

Heap: 862/1299M Errors (32636) VM is Oracle Corporation 1.8.0_102

In the compile queue for >50ms without further invocations / back edges

Hot throws

TriView - Source, Bytecode, Assembly Viewer - JITWatch

Class: Member:

Source Bytecode Assembly Mouseover

Bytecode size	Native size	Compile time
8	152	2ms

Source

```
3 public class HotThrow
4 {
5     private Random random = new Random();
6
7     public HotThrow()
8     {
9         StringBuilder builder = new StringBuilder();
10
11        String string = "The quick brown fox jumps over the lazy dog";
12
13        char[] chars = string.toCharArray();
14
15        for (int i = 0 ; i < 1_000_000; i++)
16        {
17            int index = random.nextInt(100);
18
19            char c = getChar(chars, index);
20
21            builder.append(c);
22        }
23
24        System.out.println(builder.toString());
25    }
26
27    public char getChar(char[] chars, int index)
28    {
29        try
30        {
31            return chars[index];
32        }
33        catch (ArrayIndexOutOfBoundsException e)
34        {
35            return '*';
36        }
37    }
38
```

Bytecode (double click for JVM spec)

```
0: aload_1
1: iload_2
2: caload
3: ireturn
4: astore_3
5: bipush
6: 42
7: ireturn
```

java.lang.ArrayIndexOutOfBoundsException thrown by this operation has been pre-allocated.

```
switch (reason) {
    case Deoptimization::Reason_null_check:
        ex_obj = env()->NullPointerException_instance();
        break;
    case Deoptimization::Reason_div0_check:
        ex_obj = env()->ArithmeticException_instance();
        break;
    case Deoptimization::Reason_range_check:
        ex_obj = env()->ArrayIndexOutOfBoundsException_instance();
        break;
    case Deoptimization::Reason_class_check:
        if (java_bc() == Bytecodes::_astore) {
            ex_obj = env()->ArrayStoreException_instance();
        } else {
            ex_obj = env()->ClassCastException_instance();
        }
        break;
}
```

share/vm/opto/graphKit.cpp

TL;DR

JIT logs can reveal optimisation issues

Keep methods small for inlining (Head Test)

Check inline-ability of JDK methods used

Check for unpredictable branches

Use appropriate method visibility (CHA)

Count interface implementations

Are allocations in hot code 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

Thanks for listening!

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

