Understanding HotSpot JVM Performance with JITWatch

Chris Newland, JavaZone 2016-09-08


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

mvn clean install exec:java
```

Bio

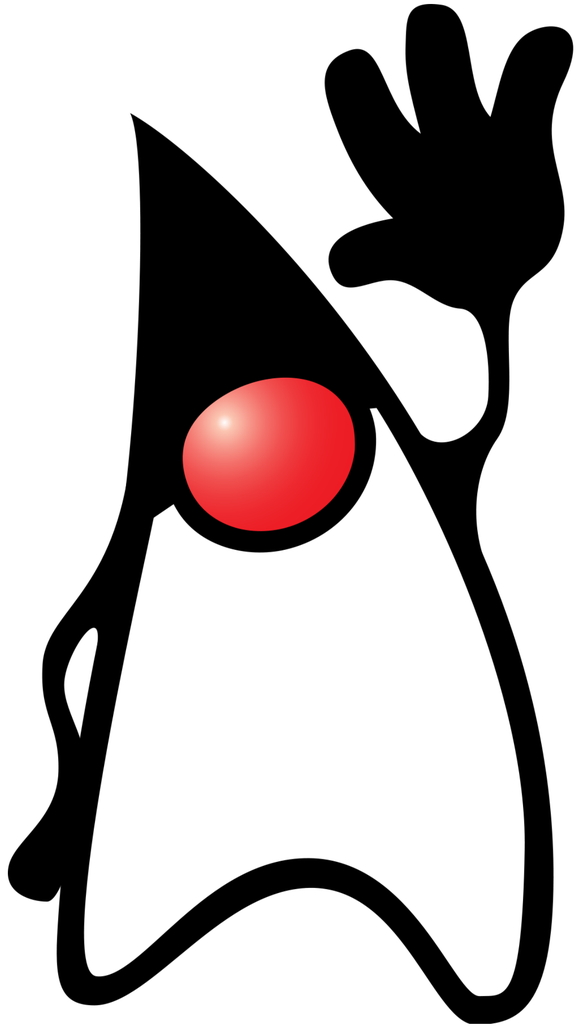
Chris Newland

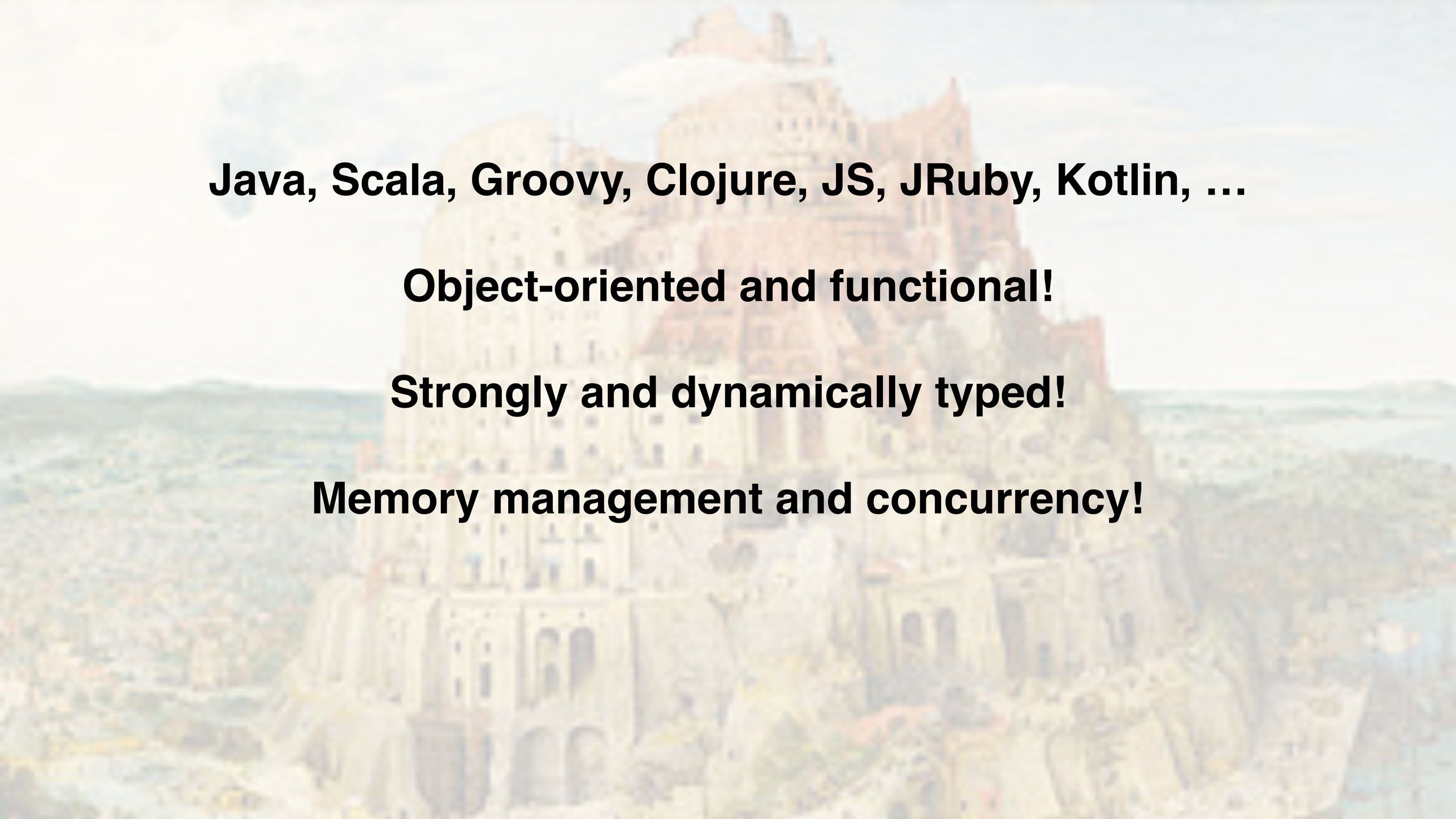
Market data guy at 

@chriswhocodes on Twitter

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

mvn clean install exec:java
```

The amazing JVM





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

Object-oriented and functional!

Strongly and dynamically typed!

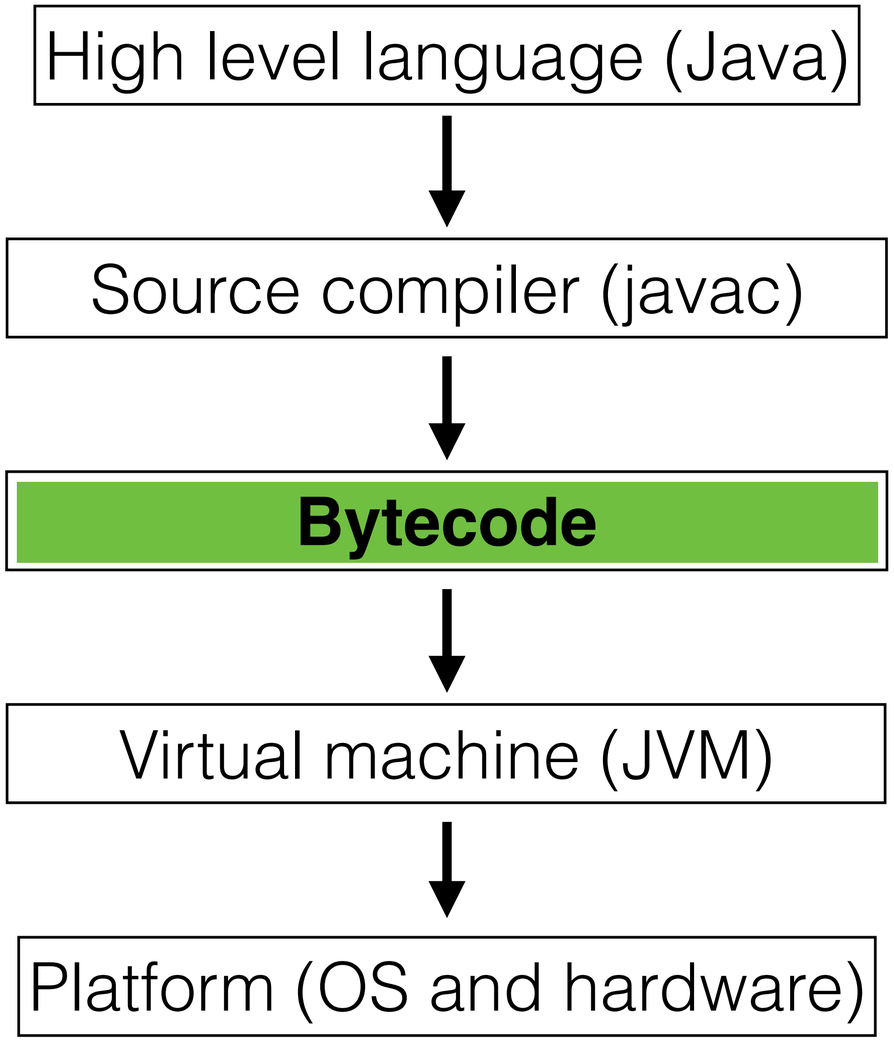
Memory management and concurrency!

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)
{
   return a + b;
}
```


```
public int add(int, int);
  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

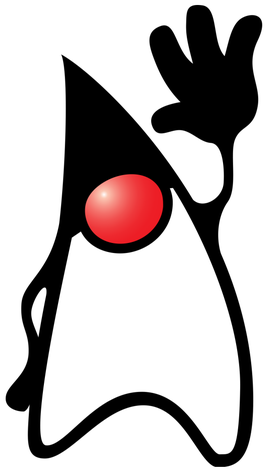
A simple interpreter

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

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

   ...

   case ff:
      // handle
      break;
   }
}
```

http://docklandsljc.uk/2016/06/hotspot-hood-microbenchmarking-java.html

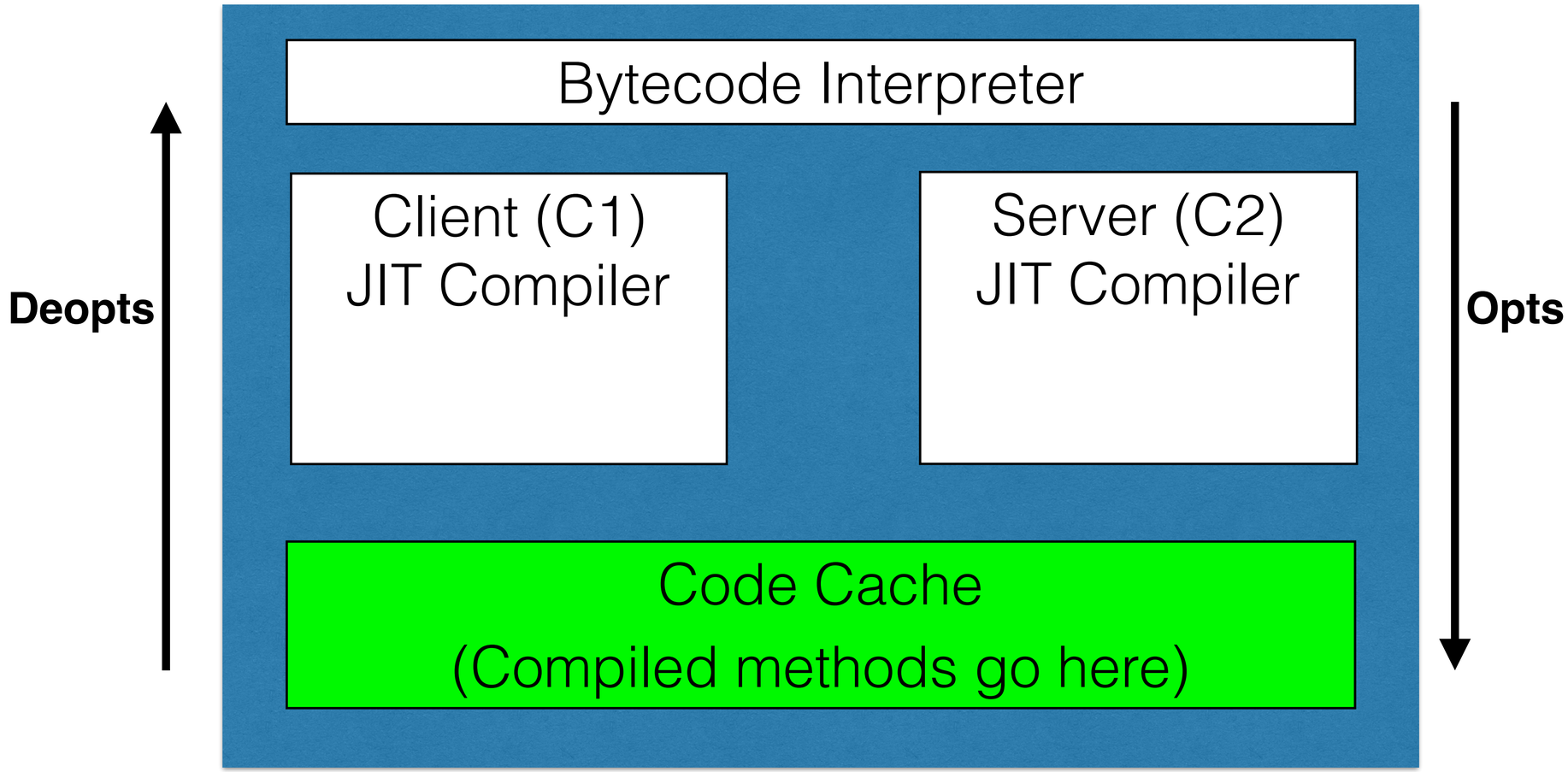
Running faster

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 Tier23InlineeNotifyFreqLog               = 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              = 600                                 {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

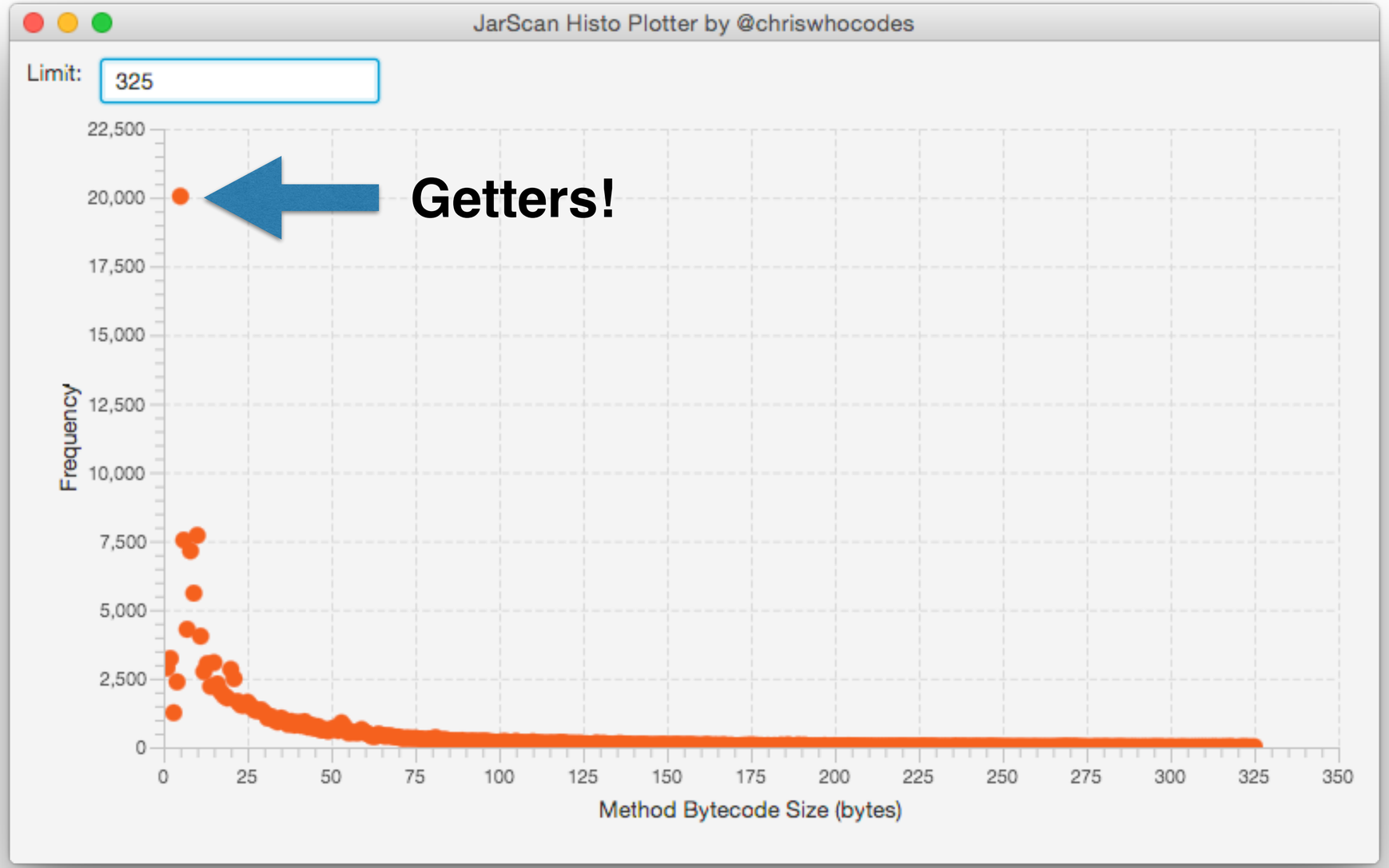
More info: http://www.slideshare.net/maddocig/tiered

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



https://www.chrisnewland.com/more-bytecode-geekery-with-jarscan-404

Code cache

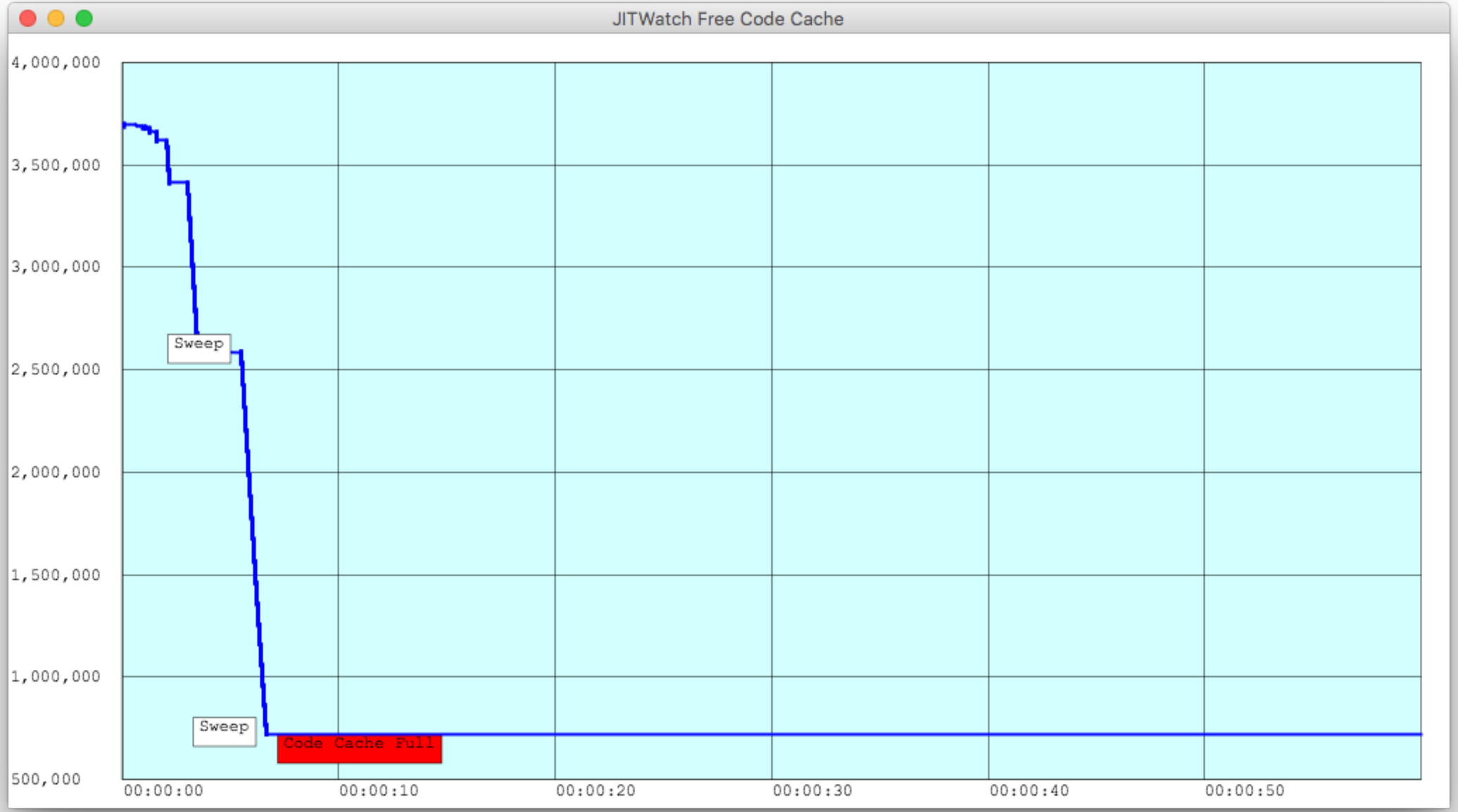
JVM region for JIT-compiled methods

Can run out of space

Can become fragmented

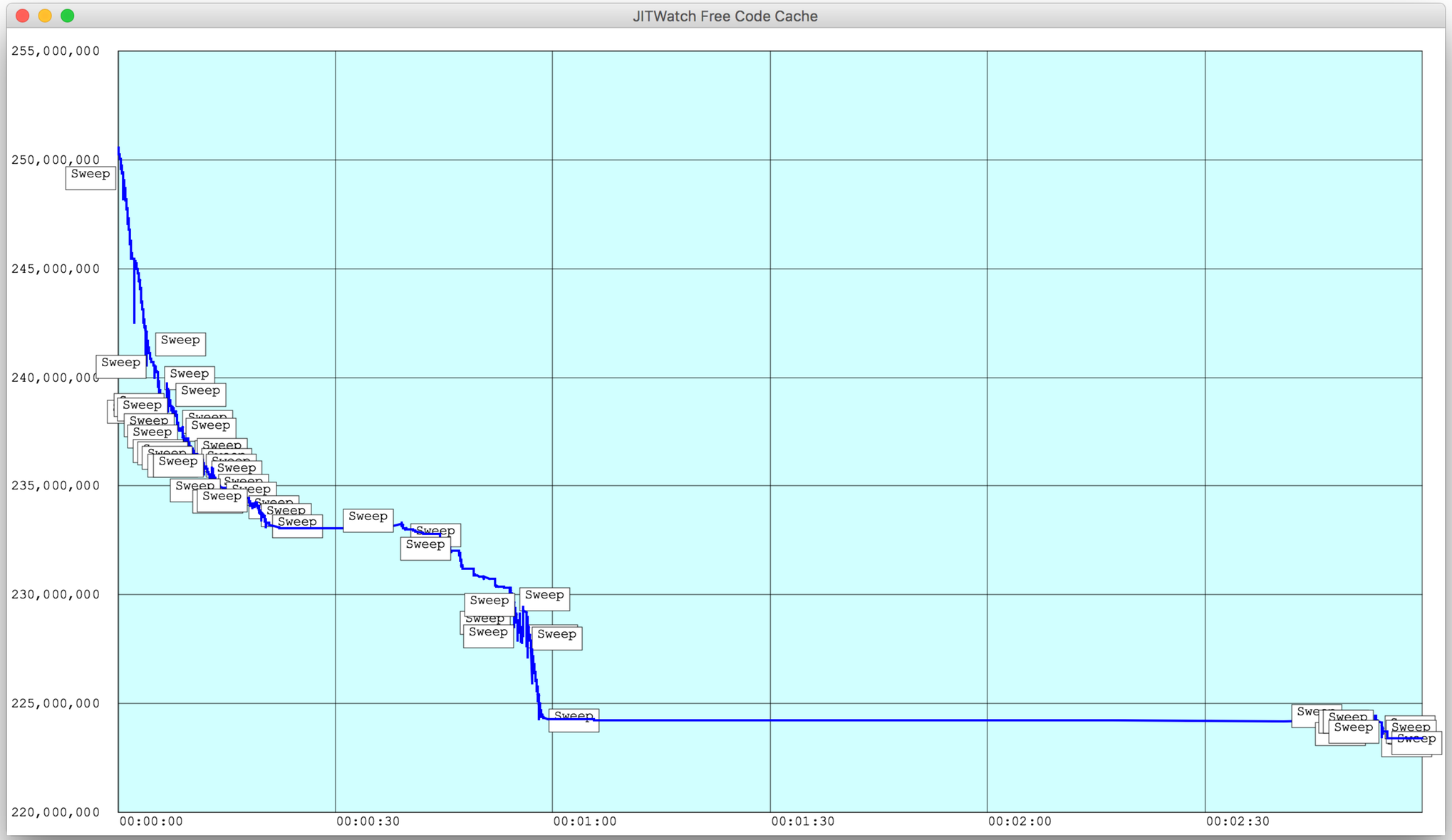
`-XX:ReservedCodeCacheSize=<size>m`

Code cache exhaustion



-XX:ReservedCodeCacheSize=4m

Sweeper activity



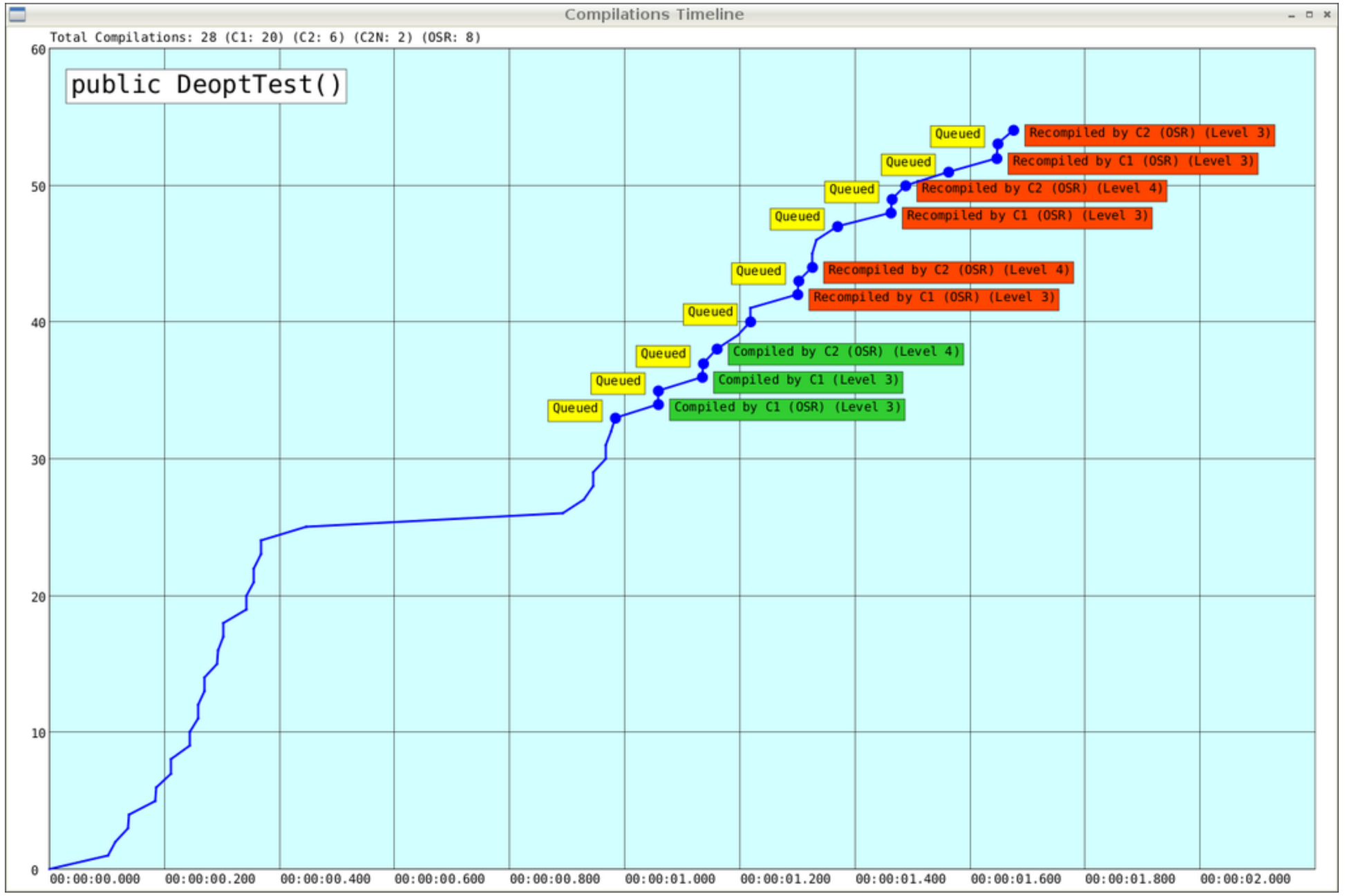
Guess again?

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

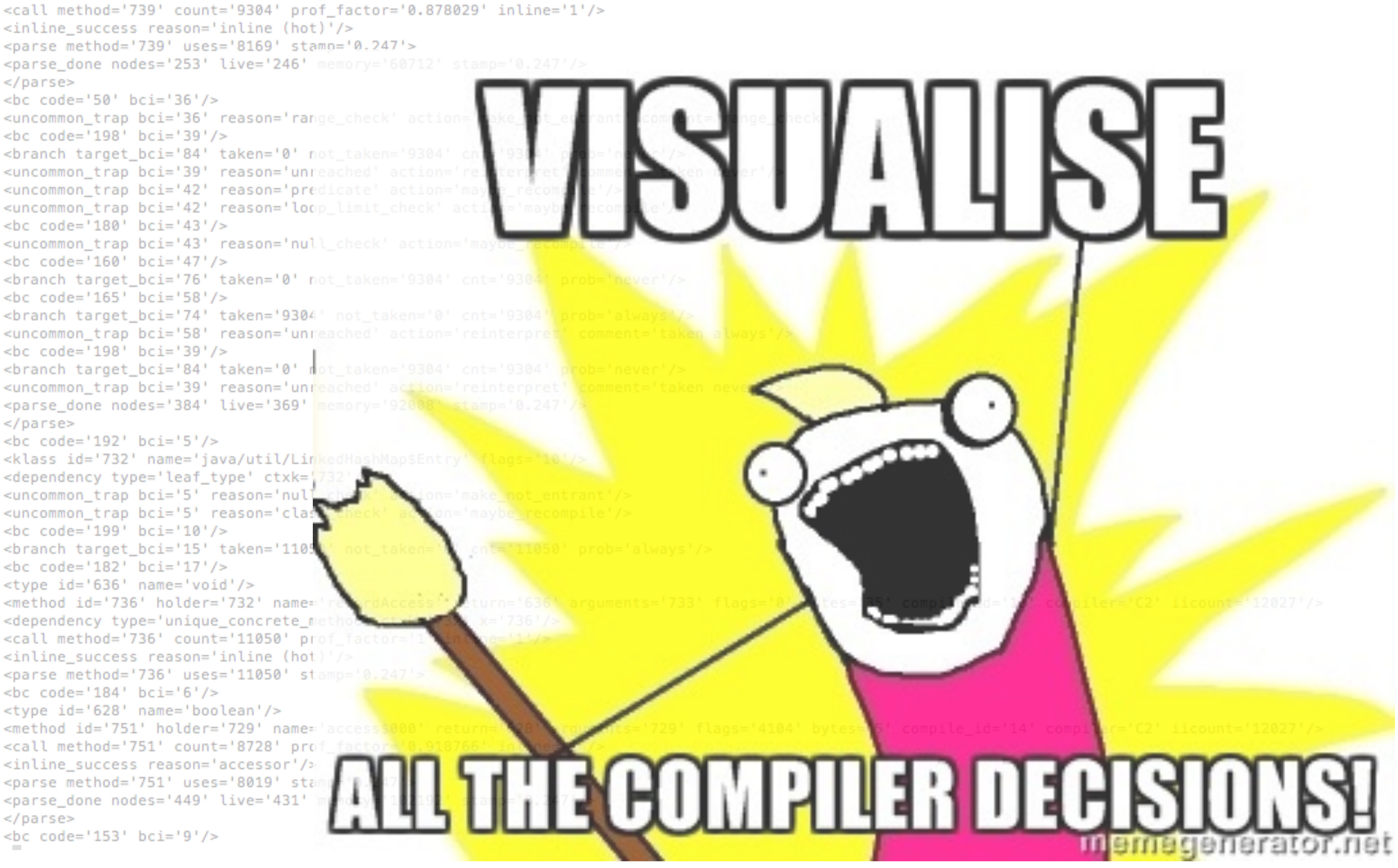
-XX:+LogCompilation

-XX:+TraceClassLoading

-XX:+PrintAssembly

hsdis binary in jre/lib/amd64/server

I heard you like to grep?



JITWatch

Compilations (when, how)

Deoptimisations (why)

Inlining successes and failures

Escape analysis

Branch probabilities

Intrinsics used

Hot throws, stale tasks, and more!

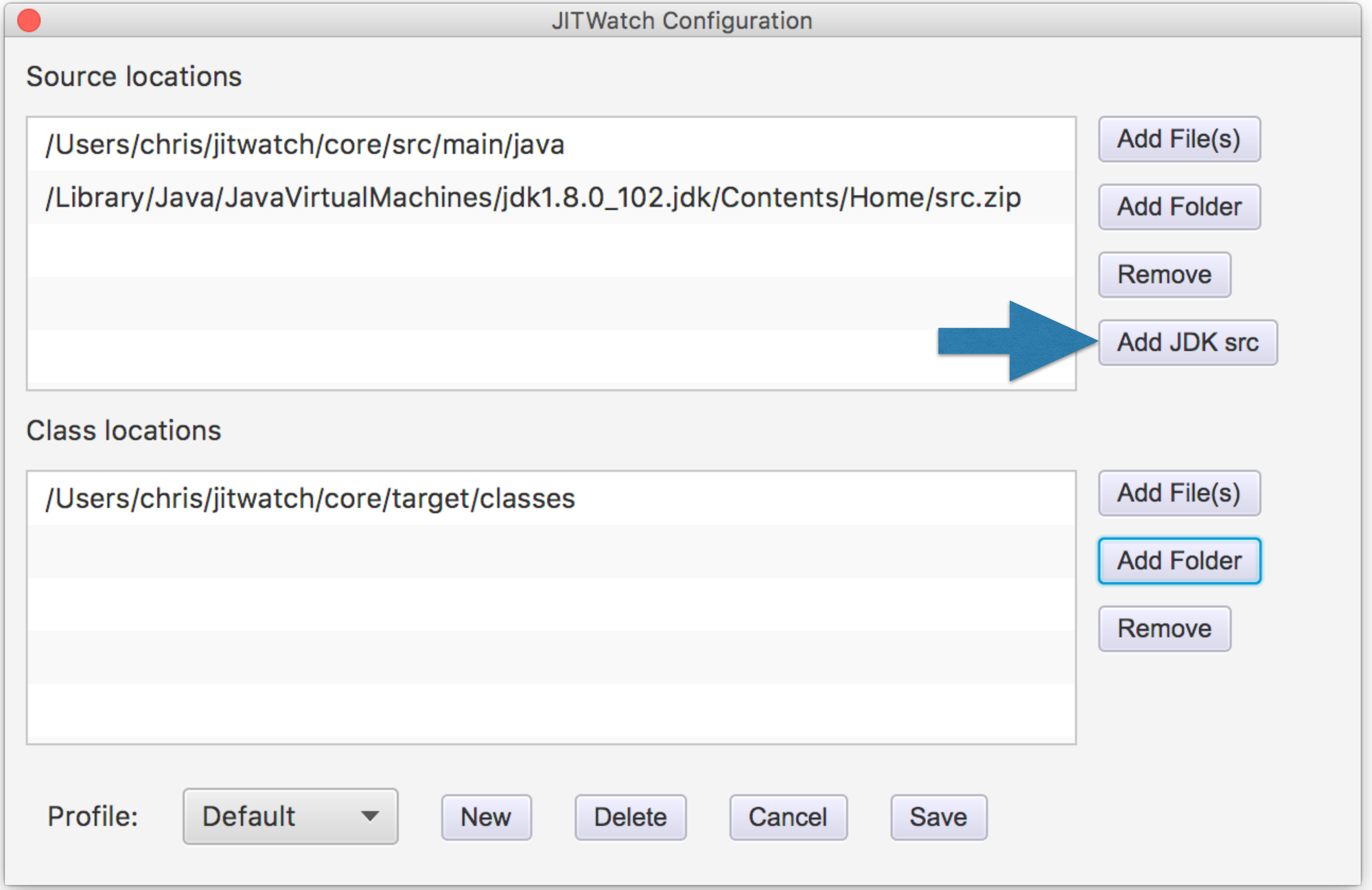
Getting Started

mvn clean compile

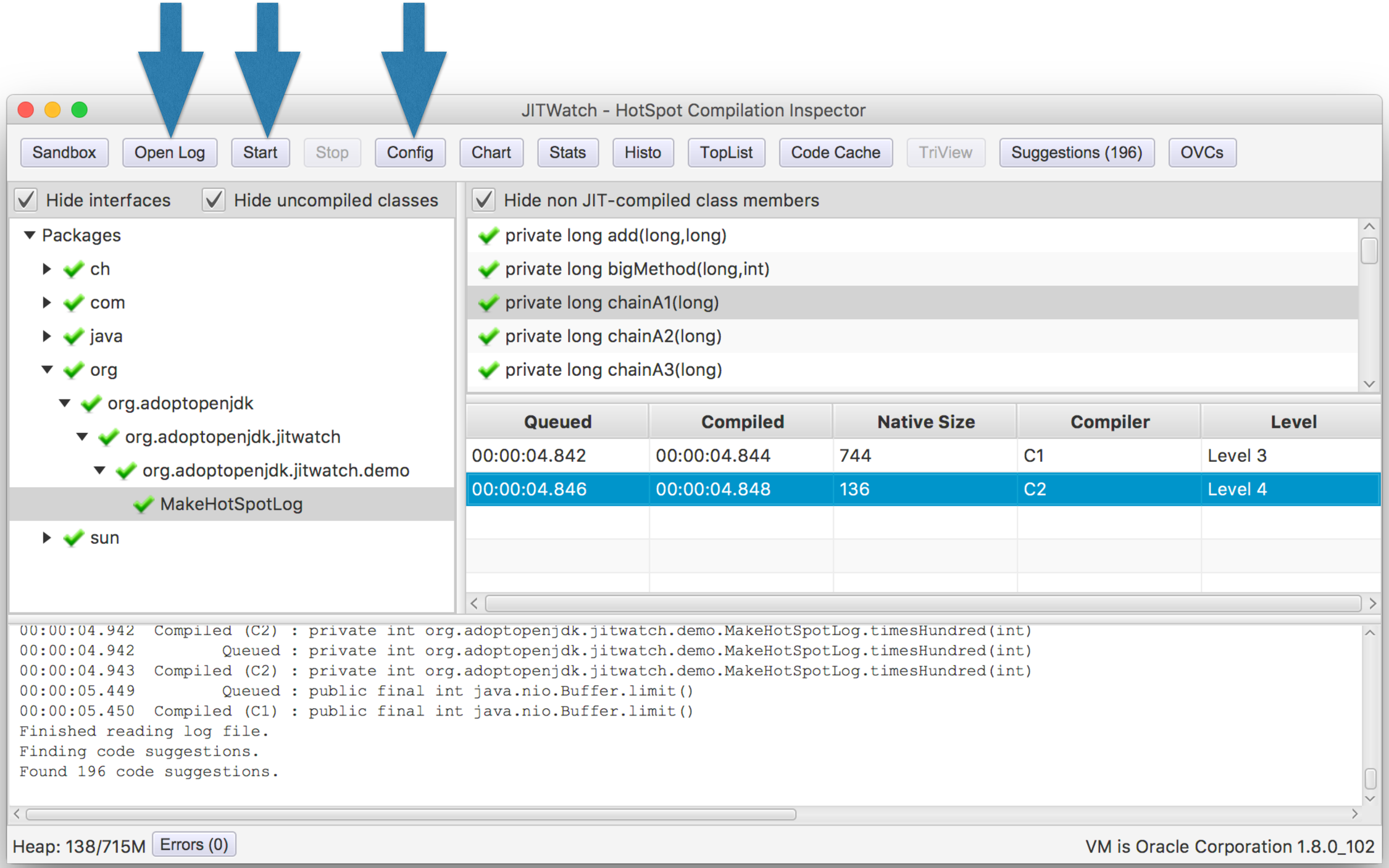
./makeDemoLogFile.sh

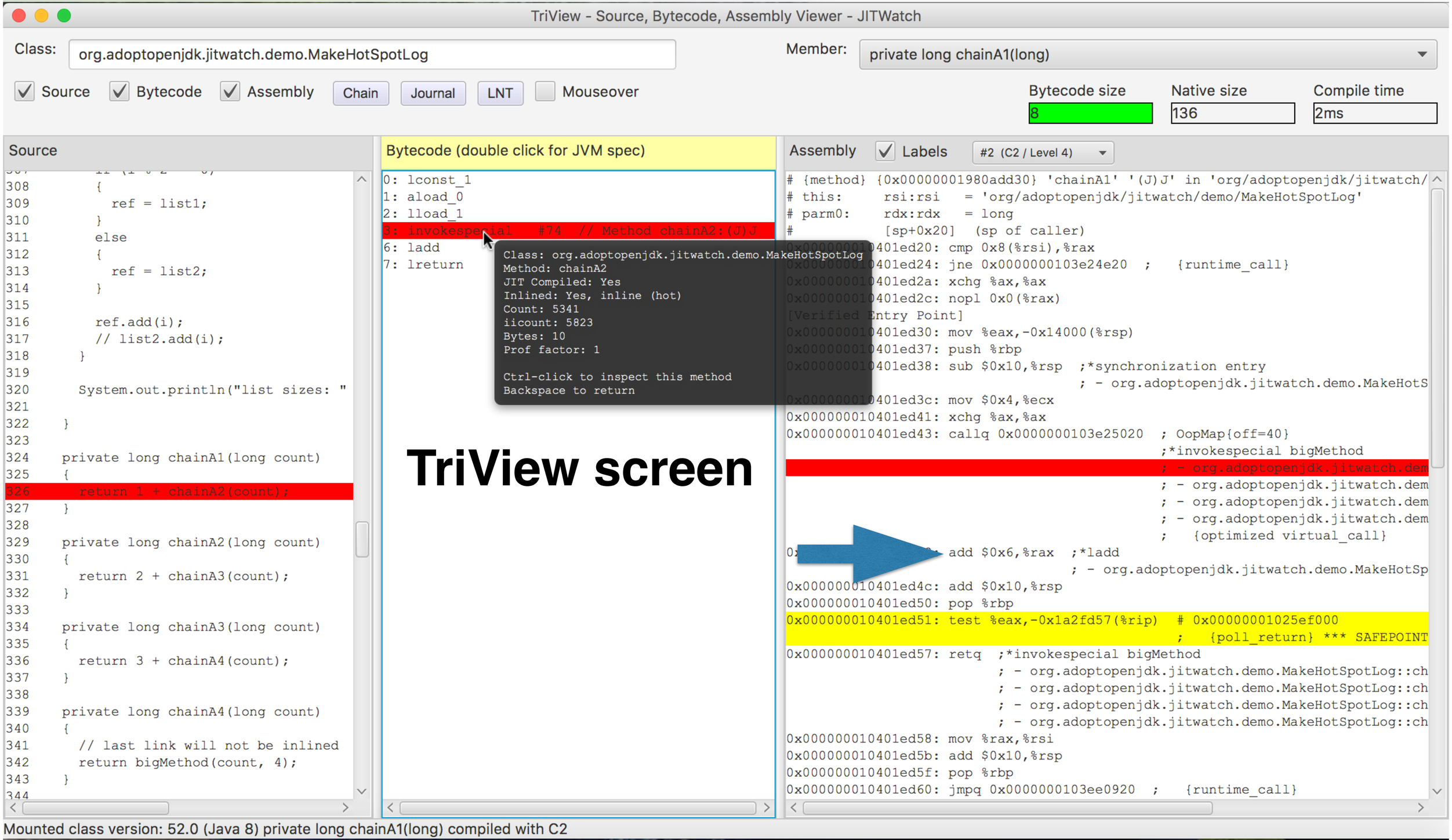
```
java version "1.8.0_102"
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
VM Switches -XX:+UnlockDiagnosticVMOptions -XX:+TraceClassLoading -XX:
+LogCompilation -XX:+PrintAssembly -XX:-UseCompressedOops
Building example HotSpot 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
private long add(long,long)
private long bigMethod(long,int)
private long chainA1(long)
private long chainA2(long)
private long chainA3(long)
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


TriView - Source, Bytecode, Assembly Viewer - JITWatch
Class:
org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog
Member:
private long chainA1(long)
Source
Bytecode
Assembly
Chain
Journal
LNT
Mouseover
Bytecode size
8
Native size
136
Compile time
2ms
Source
Bytecode (double click for JVM spec)
Assembly
Labels
#2 (C2 / Level 4)
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
private long chainA1(long count)
return 1 + chainA2(count);
private long chainA2(long count)
return 2 + chainA3(count);
private long chainA3(long count)
return 3 + chainA4(count);
private long chainA4(long count)
// last link will not be inlined
return bigMethod(count, 4);
add $0x6,%rax ;*ladd
Mounted class version: 52.0 (Java 8) private long chainA1(long) compiled with C2

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

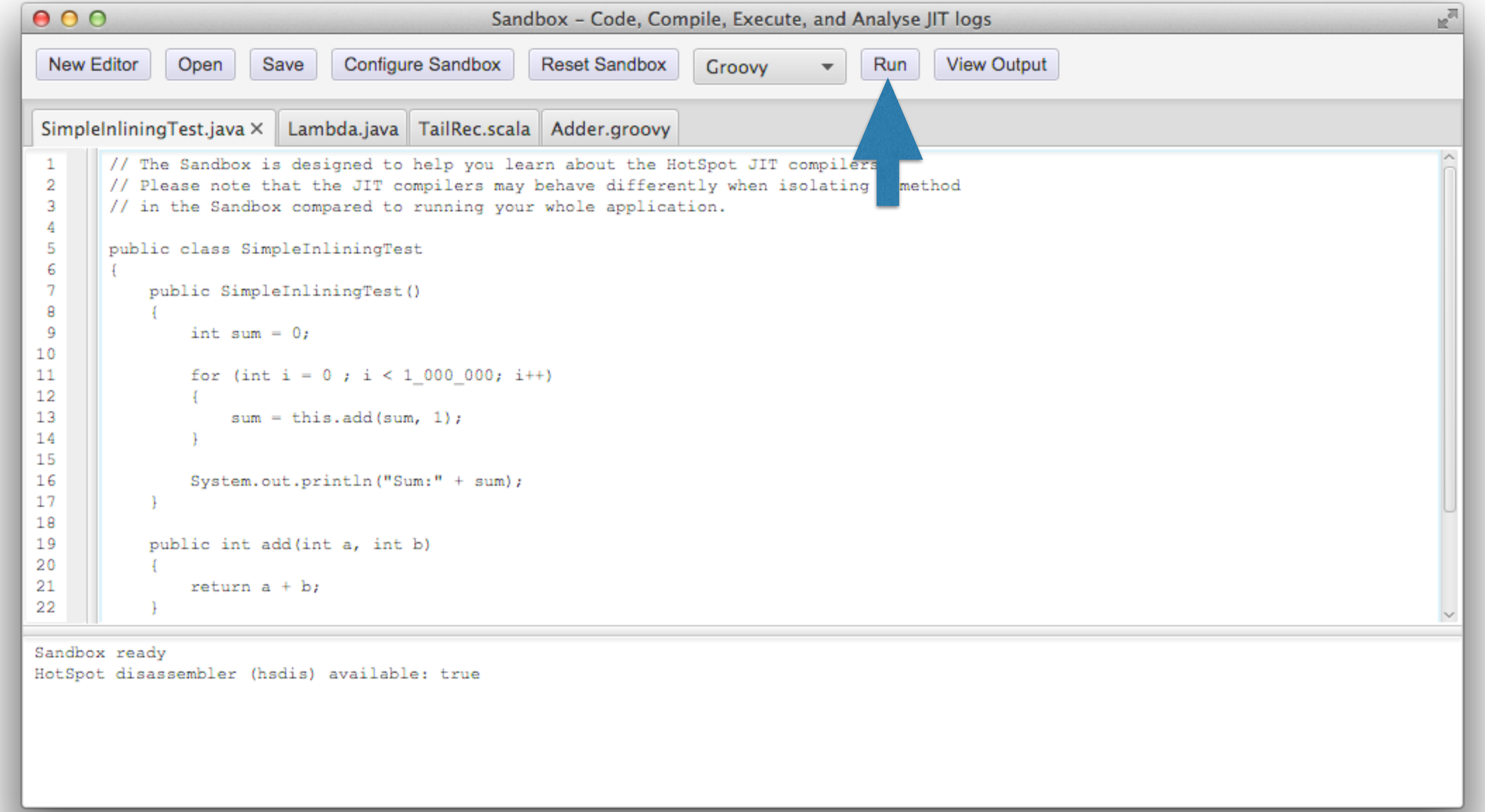
..., *objectref*, [*arg1*, [*arg2* ...]] →

...

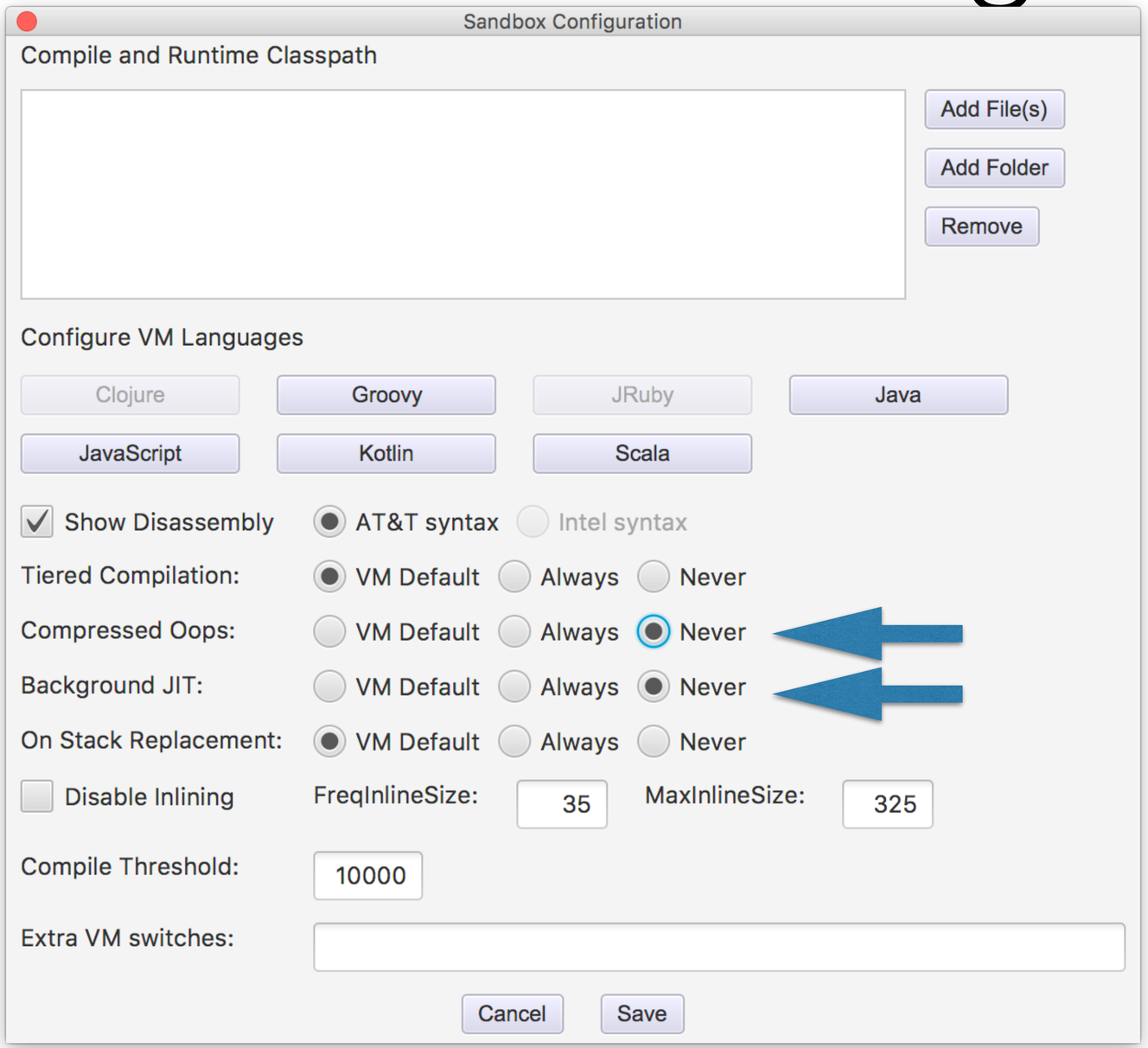
Description

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 (*indexbyte1* << 8) | *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

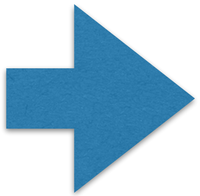


Sandbox Config



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”

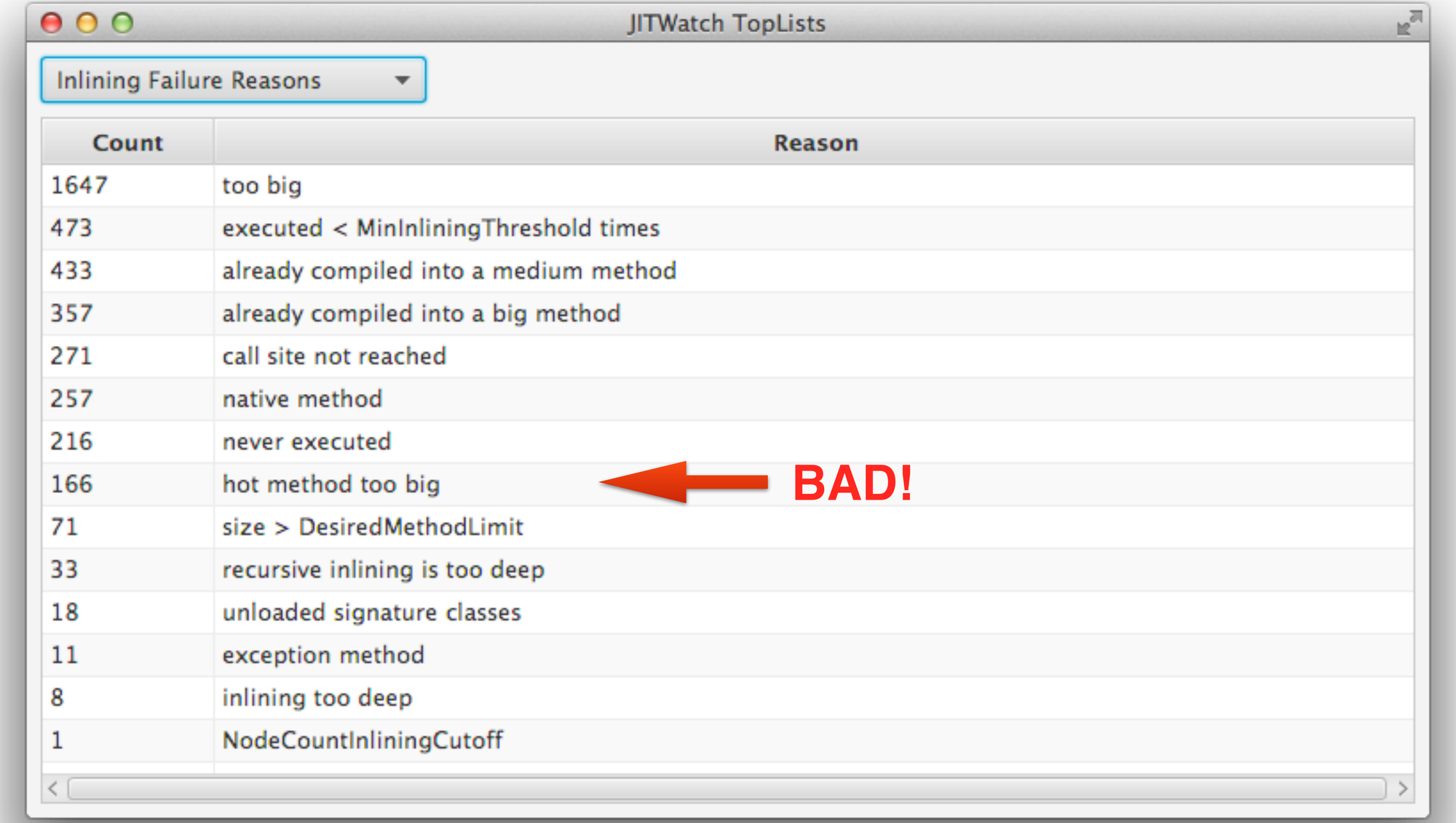
Inlining Limits

Increases size of compiled code

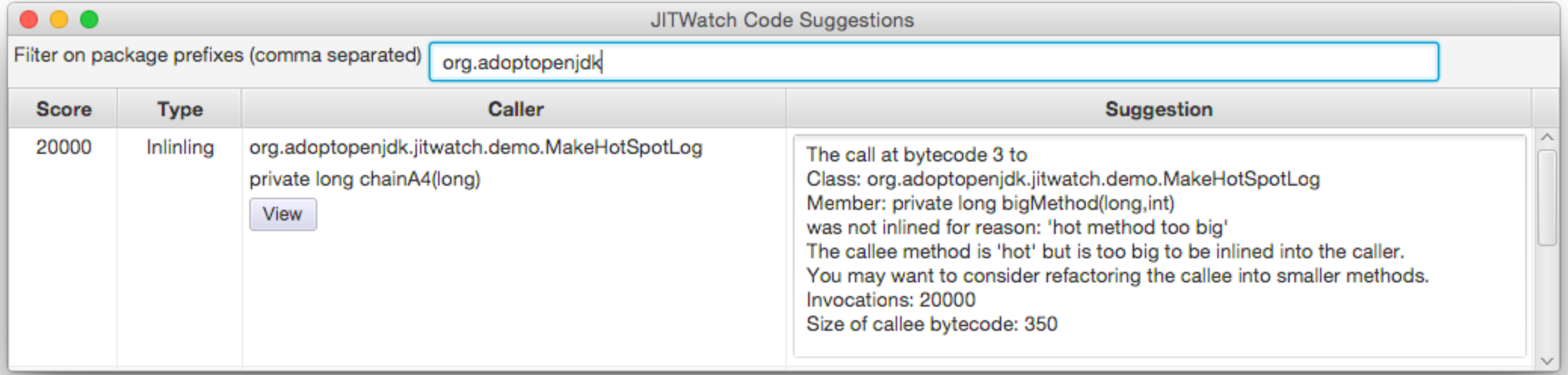
< 35 bytes (**`-XX:MaxInlineSize=n`**)

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

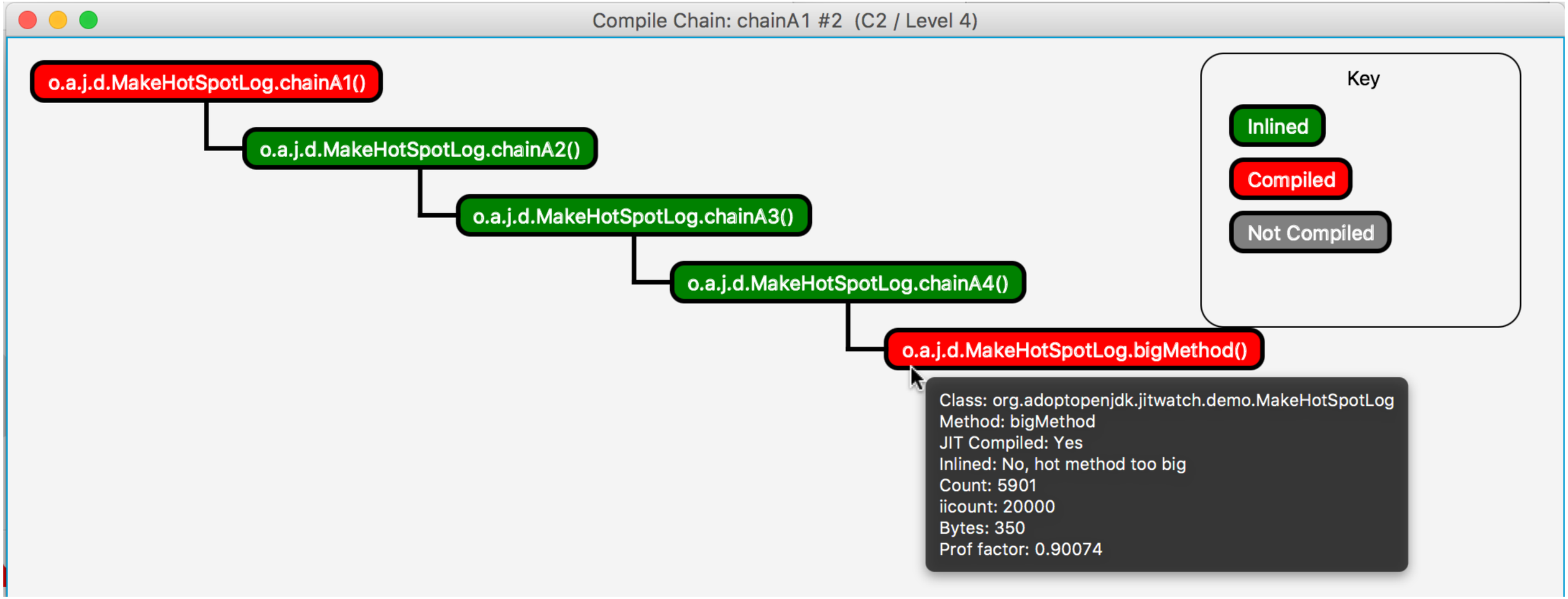
Inlining Failure Modes



Inlining Suggestions



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);
}
```

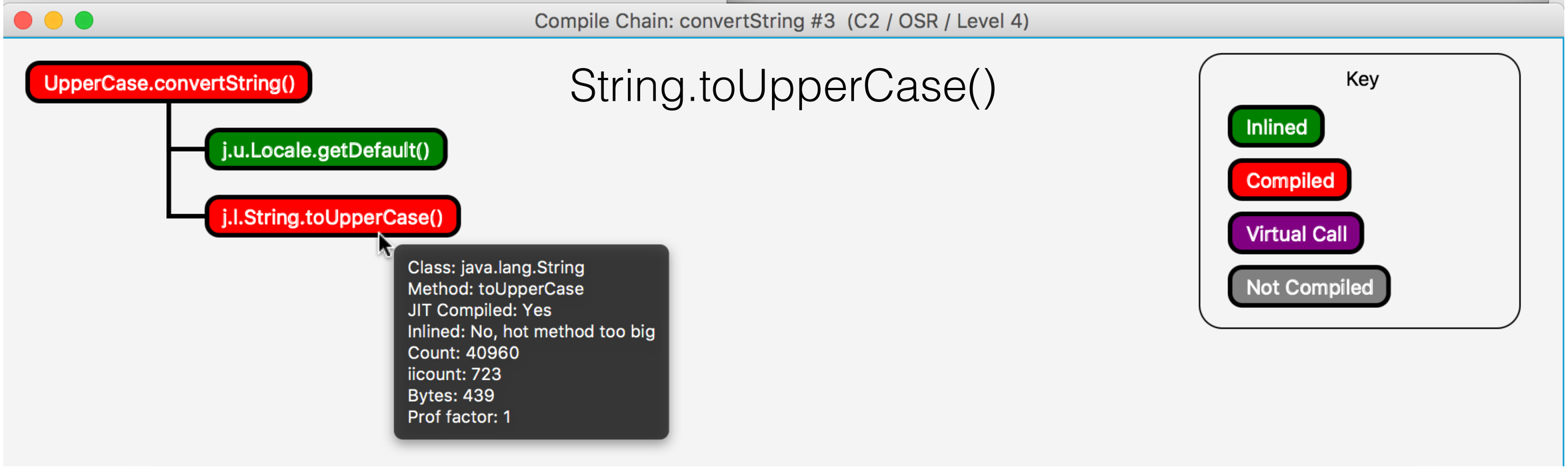
69 bytes of bytecode

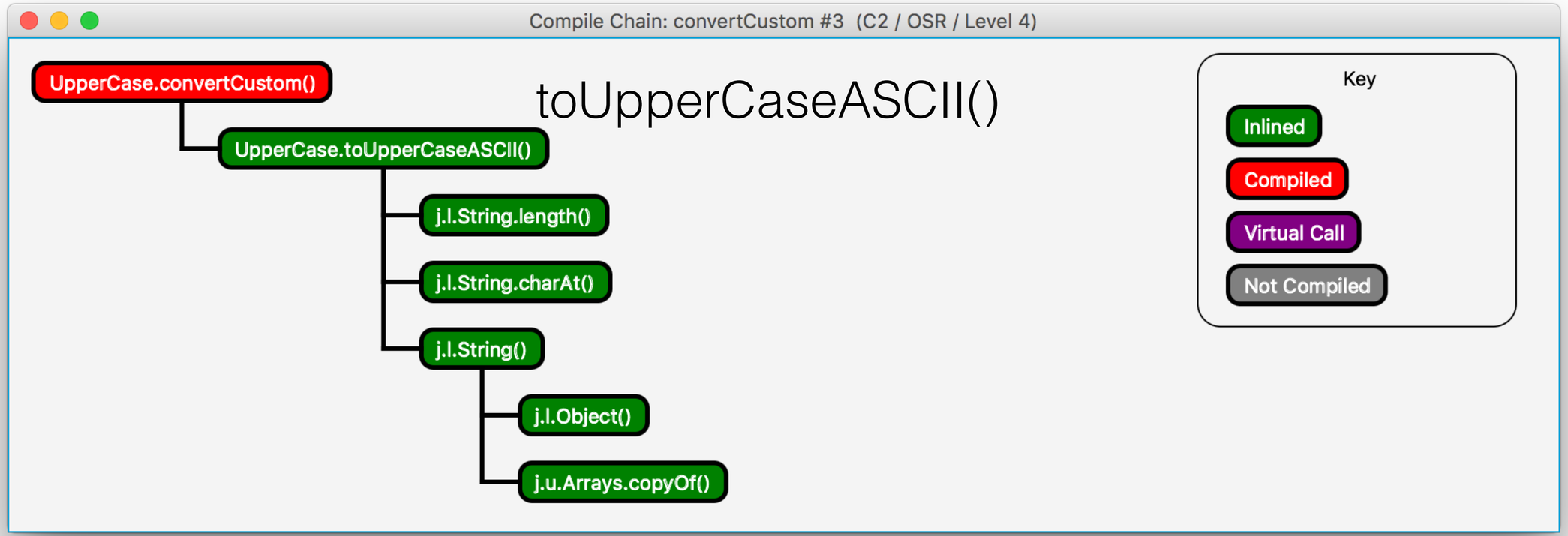
JMH Comparison

```
@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


Compile Chain: convertString #3 (C2 / OSR / Level 4)
UpperCase.convertString()
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
String.toUpperCase()
Key
Inlined
Compiled
Virtual Call
Not Compiled


Compile Chain: convertCustom #3 (C2 / OSR / Level 4)
UpperCase.convertCustom()
UpperCase.toUpperCaseASCII()
j.l.String.length()
j.l.String.charAt()
j.l.String()
j.l.Object()
j.u.Arrays.copyOf()
toUpperCaseASCII()
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.

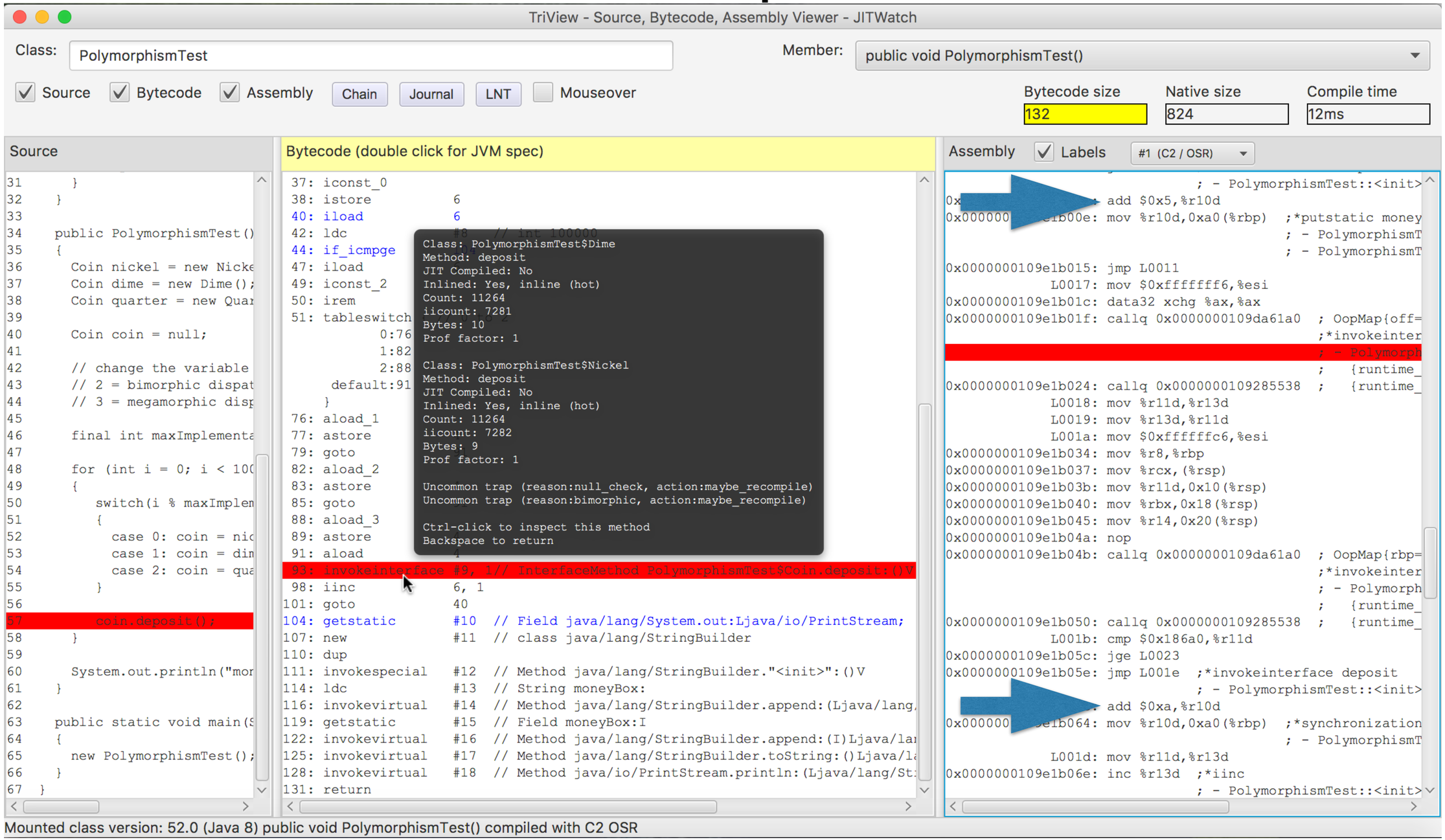
Too many implementations can prevent inlining.

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

`-XX:TypeProfileMajorReceiverPercent=90`

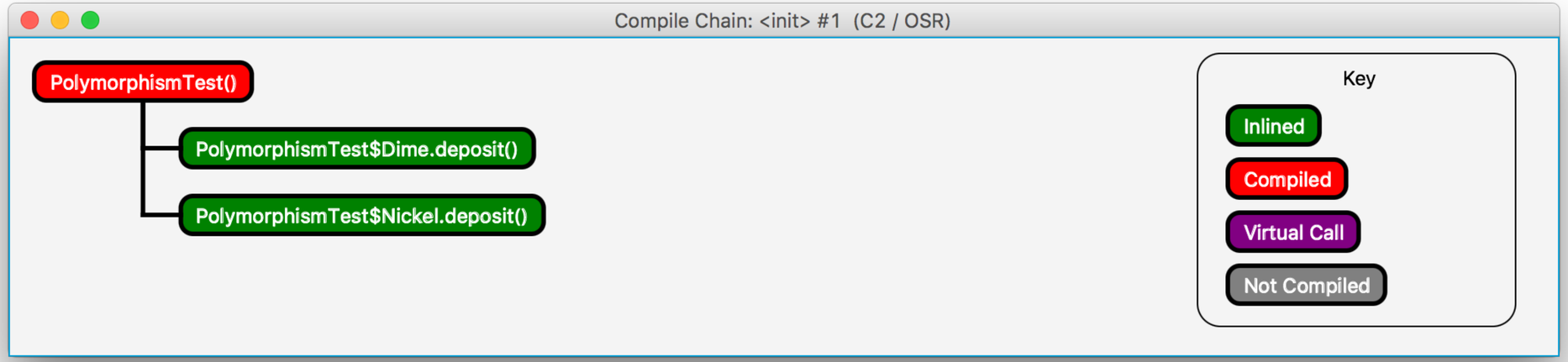
```
public class PolymorphismTest
{
   public interface Coin   { void deposit(); }

   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);
    }
}
```

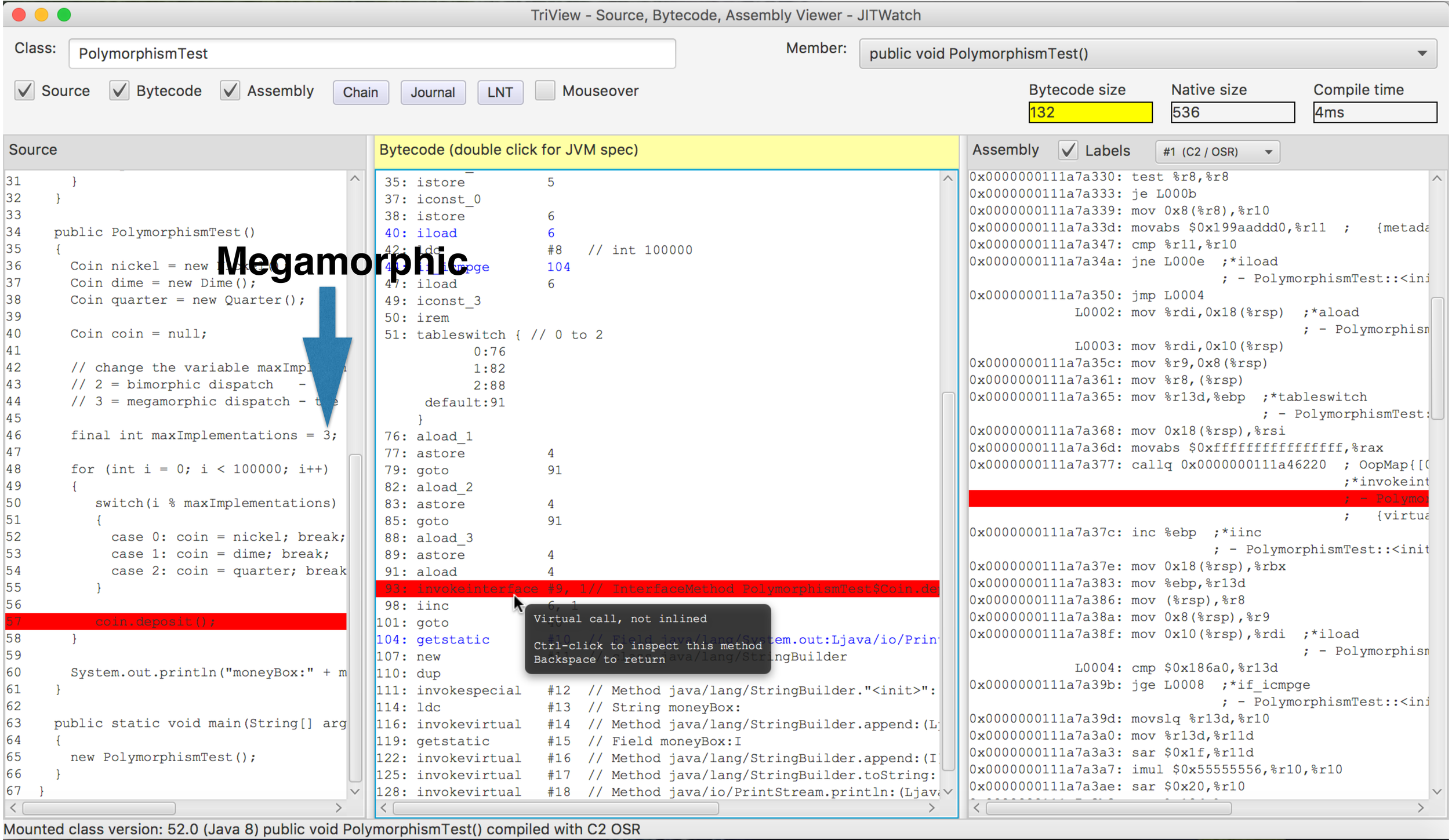
Bimorphic


TriView - Source, Bytecode, Assembly Viewer - JITWatch
Class: PolymorphismTest
Member: public void PolymorphismTest()
Source
Bytecode
Assembly
Chain
Journal
LNT
Mouseover
Bytecode size
Native size
Compile time
132
824
12ms
Source
Bytecode (double click for JVM spec)
Assembly
Labels
#1 (C2 / OSR)
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
93: invokeinterface #9, 1// InterfaceMethod PolymorphismTest$Coin.deposit:()V
Mounted class version: 52.0 (Java 8) public void PolymorphismTest() compiled with C2 OSR

Bimorphic

```
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
```



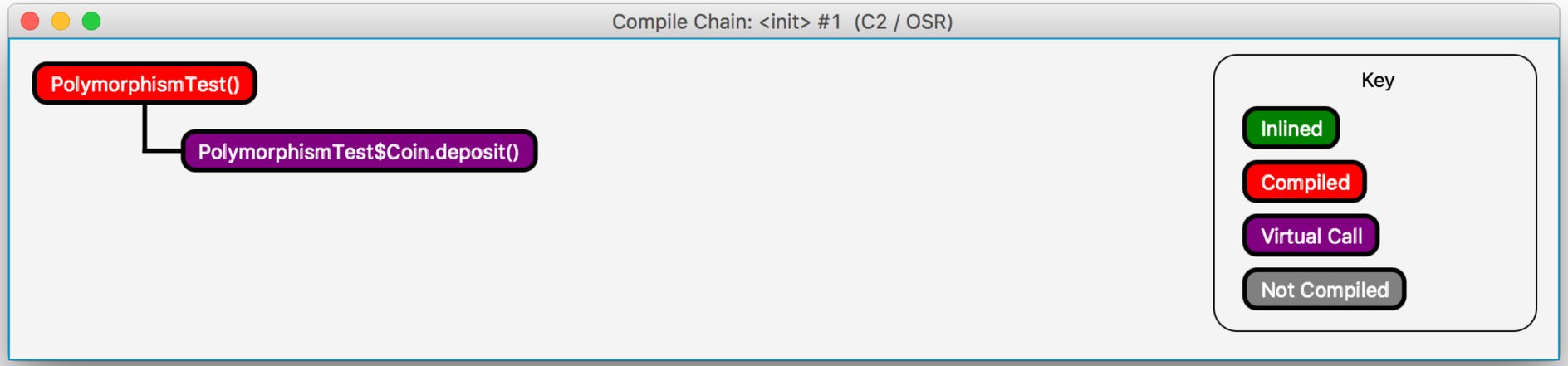

TriView - Source, Bytecode, Assembly Viewer - JITWatch
Class: PolymorphismTest
Member: public void PolymorphismTest()
Source
Bytecode
Assembly
Chain
Journal
LNT
Mouseover
Bytecode size
132
Native size
536
Compile time
4ms
Source
Bytecode (double click for JVM spec)
Assembly
Labels
#1 (C2 / OSR)
Megamorphic
Virtual call, not inlined
Ctrl-click to inspect this method
Backspace to return
Mounted class version: 52.0 (Java 8) public void PolymorphismTest() compiled with C2 OSR

Megamorphic

```
 91: aload           4
 93: invokeinterface #9, 1// InterfaceMethod PolymorphismTest$Coin.de
 98: iinc            6, 1
101: goto
104: getstatic                                    tem.out:Ljava/io/Prin
107: new                                          ringBuilder
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	ArgEscape

`public long noEscape()`
`{`
`  long sum = 0;`
 	
`  for (int i=0; i<BIG; i++)`
`  {`
`  MyObj foo = new MyObj(i);`
 	
`  sum += foo.bar();`
`  }`
 	
`  return sum;`
`}`
	`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 doesn't escape the loop scope.	Object foo escapes loop scope by passing as arg to extBar().

Avoid heap allocations

NoEscape objects are “exploded”

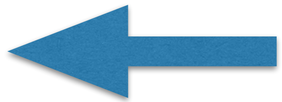
Fields are treated as locals

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

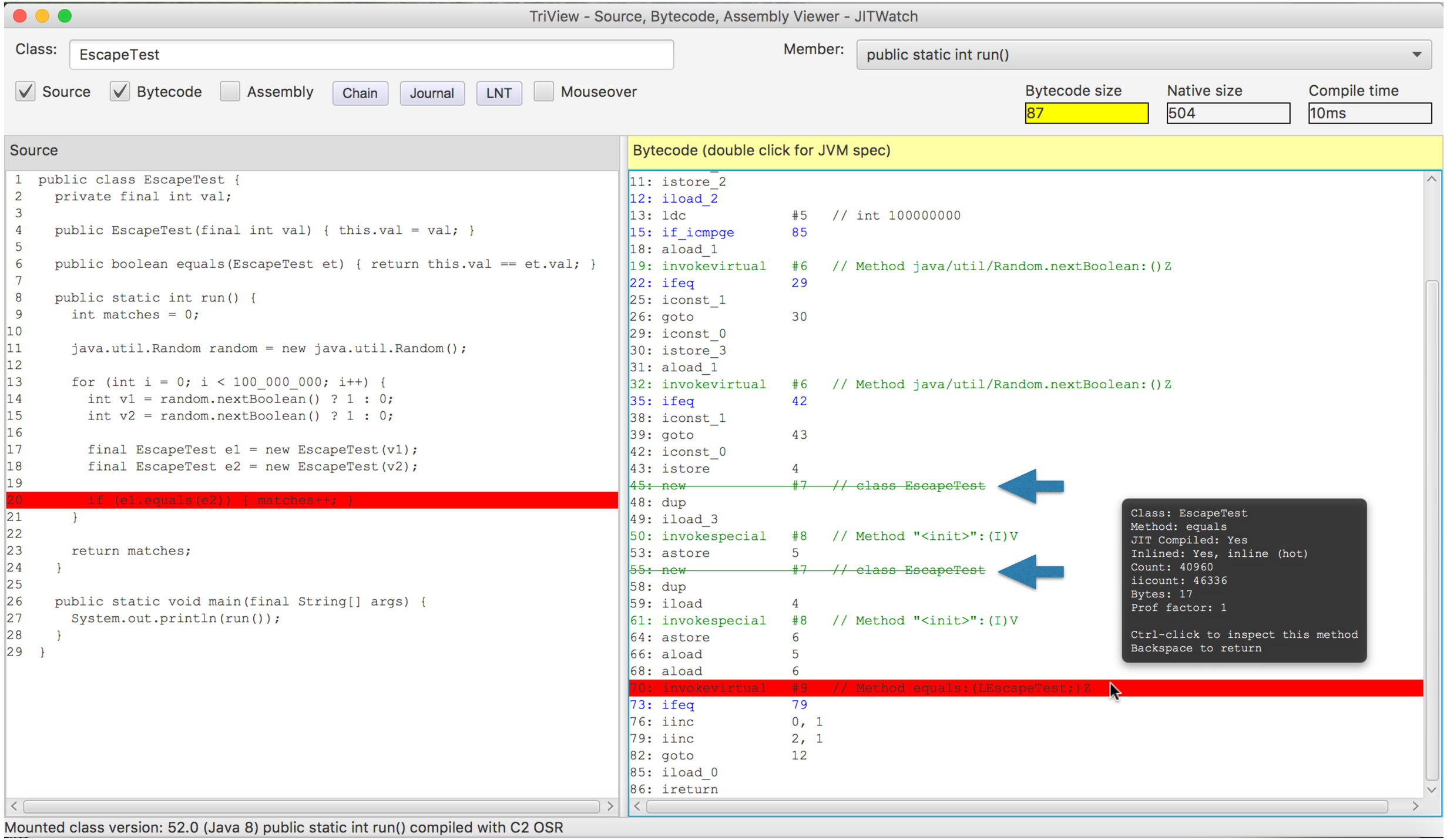
100m loops. No GCs in 2.4s

```
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

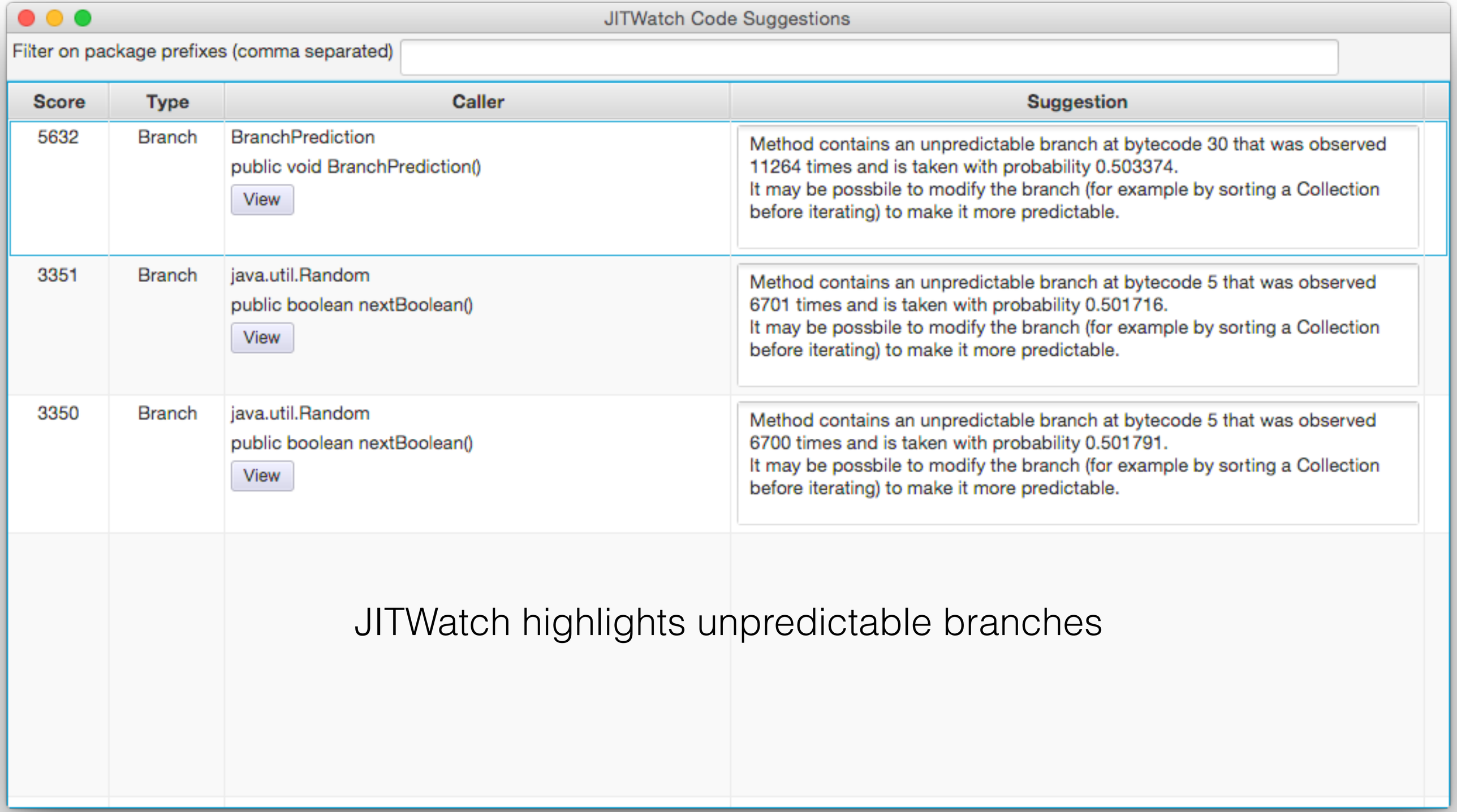
100m loops. 10 minor GCs in 2.4s

```
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
```

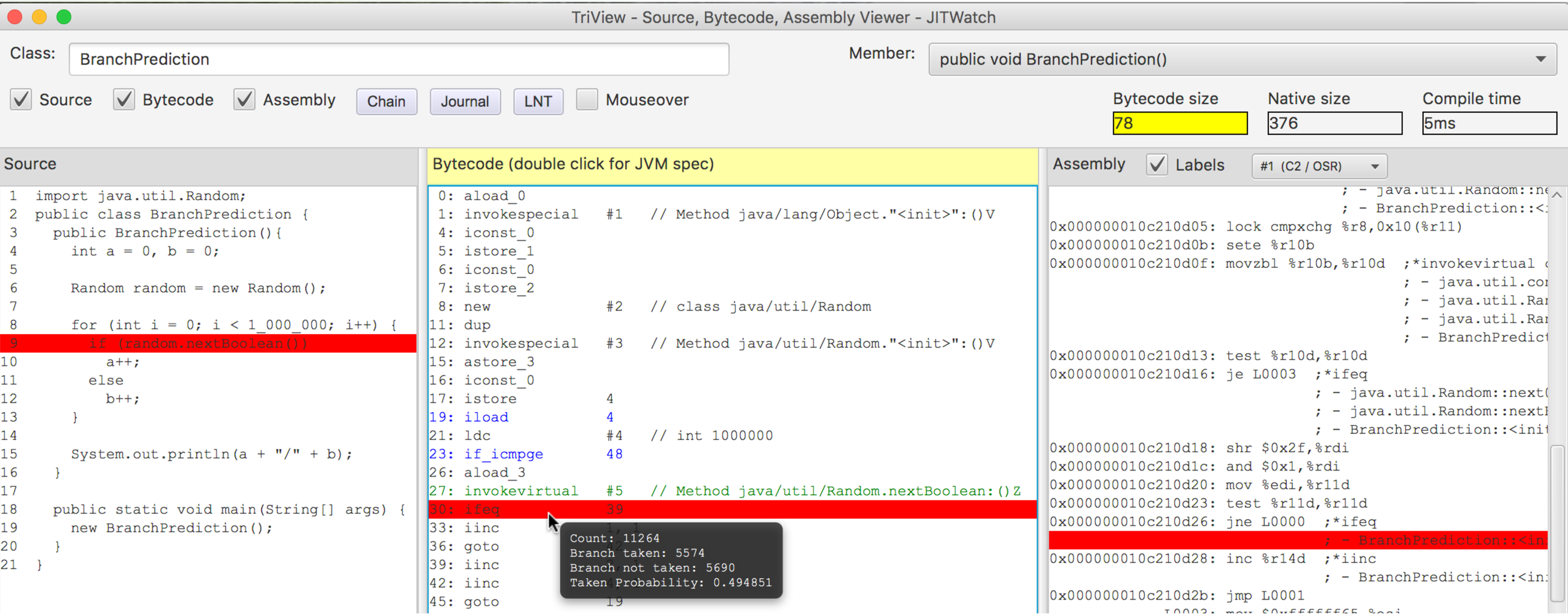

TriView - Source, Bytecode, Assembly Viewer - JITWatch
Class:
EscapeTest
Member:
public static int run()
Source
Bytecode
Assembly
Chain
Journal
LNT
Mouseover
Bytecode size
87
Native size
504
Compile time
10ms
Source
1 public class EscapeTest {
2 private final int val;
3
4 public EscapeTest(final int 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
23 return matches;
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
Mounted class version: 52.0 (Java 8) public static int run() compiled with C2 OSR

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();
  }
}
```


JITWatch Code Suggestions
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 possbile 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 possbile 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 possbile to modify the branch (for example by sorting a Collection before iterating) to make it more predictable.
JITWatch highlights unpredictable branches

Branch prediction



Intrinsics

Highly optimised native implementations

Use features of target CPU

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

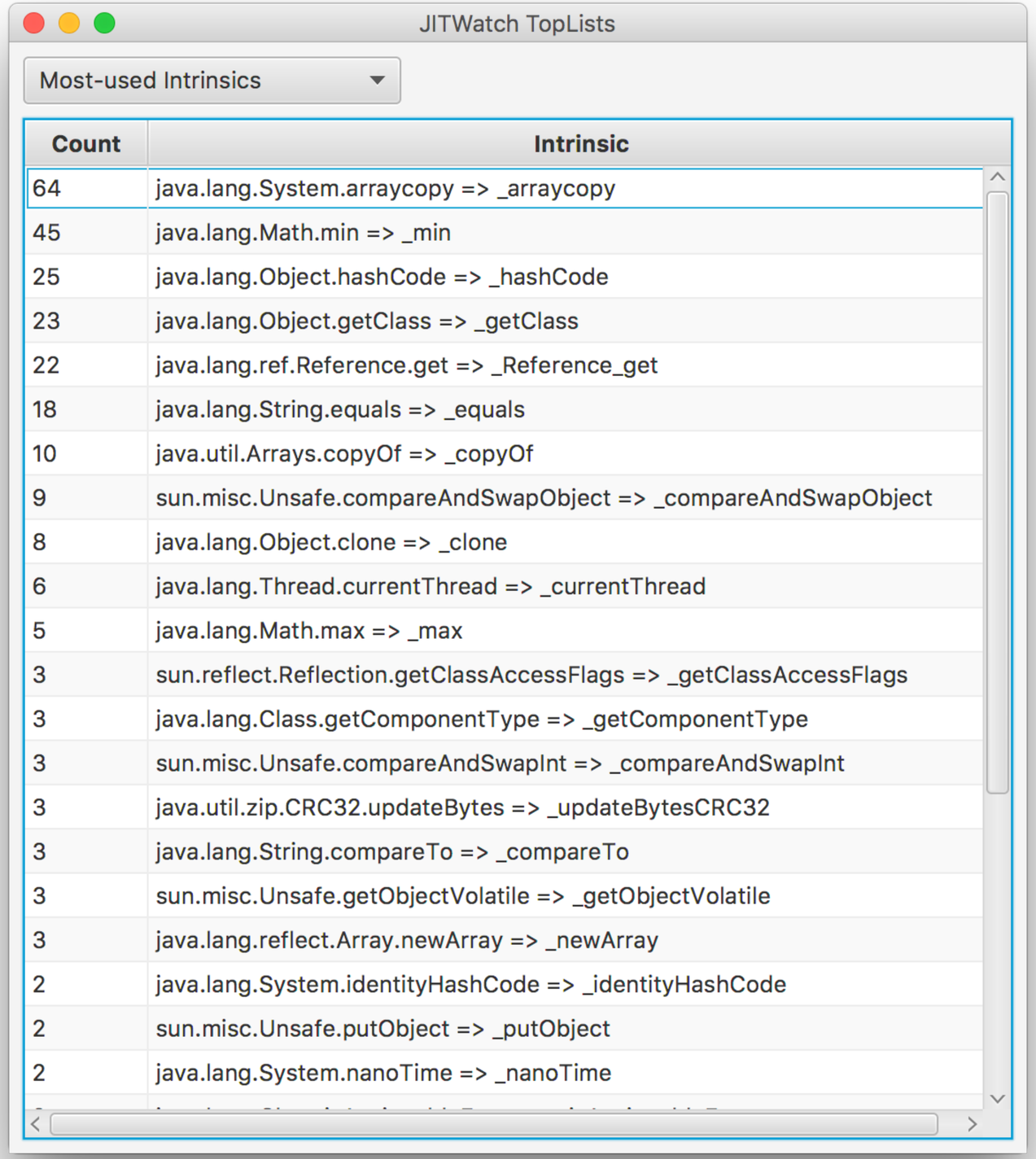
Full list in
`hotspot/src/share/vm/classfile/vmSymbols.hpp`

Intrinsics

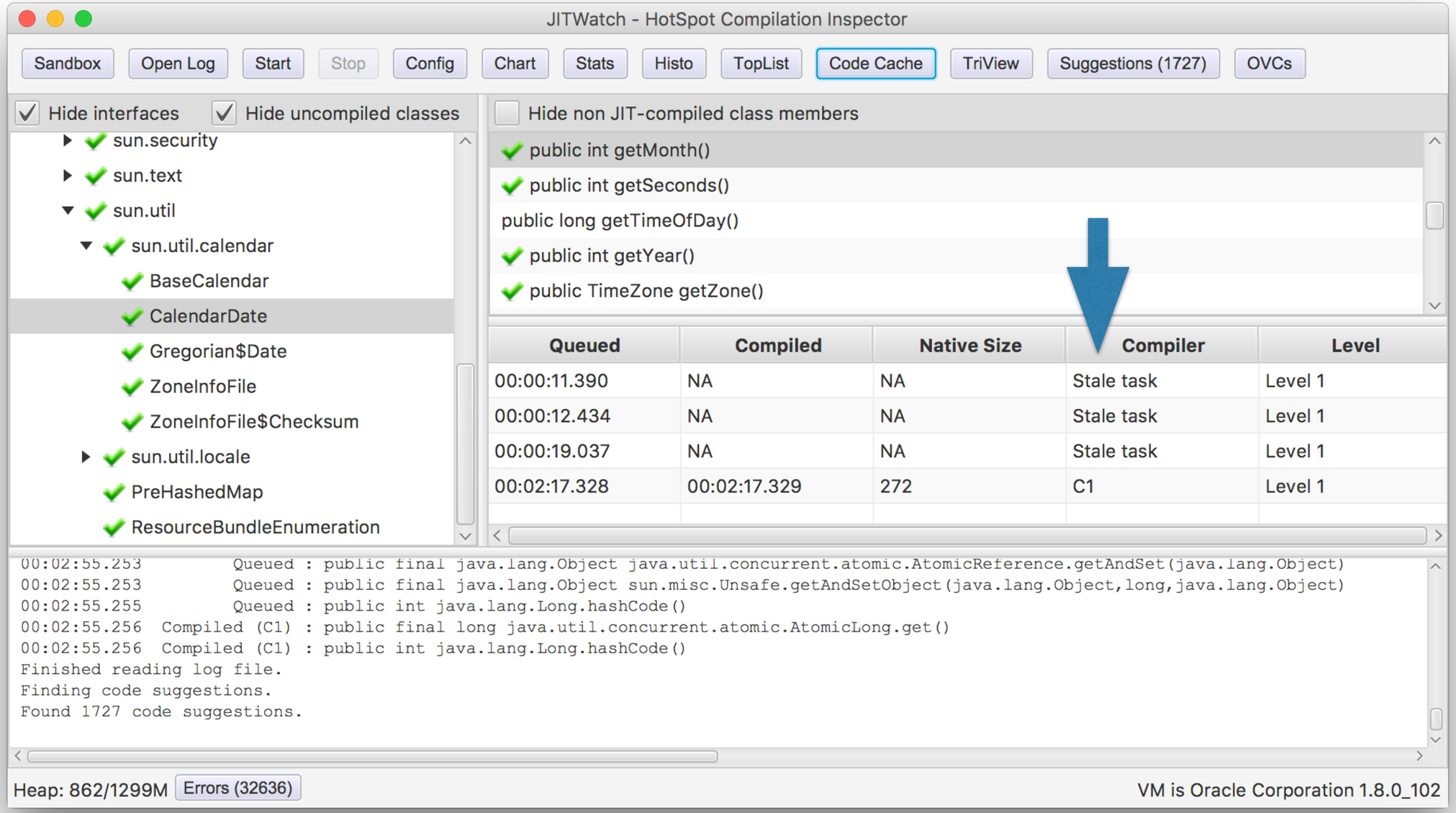
`Math.log10(double)` is 2 instructions on x86_64

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 Intrinsics
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.newArray => _newArray
2
java.lang.System.identityHashCode => _identityHashCode
2
sun.misc.Unsafe.putObject => _putObject
2
java.lang.System.nanoTime => _nanoTime

Stale tasks



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

Hot throws

TriView - Source, Bytecode, Assembly Viewer - JITWatch

Class: HotThrow

Member: public char getChar(char[],int)

Source ☑ Bytecode ☑ Assembly ☐ Chain Journal LNT ☐ Mouseover

Bytecode size: 8 | Native size: 152 | Compile time: 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
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::_aastore) {
       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


JavaZone