JVM MECHANICS

DOUGLAS Q. HAWKINS

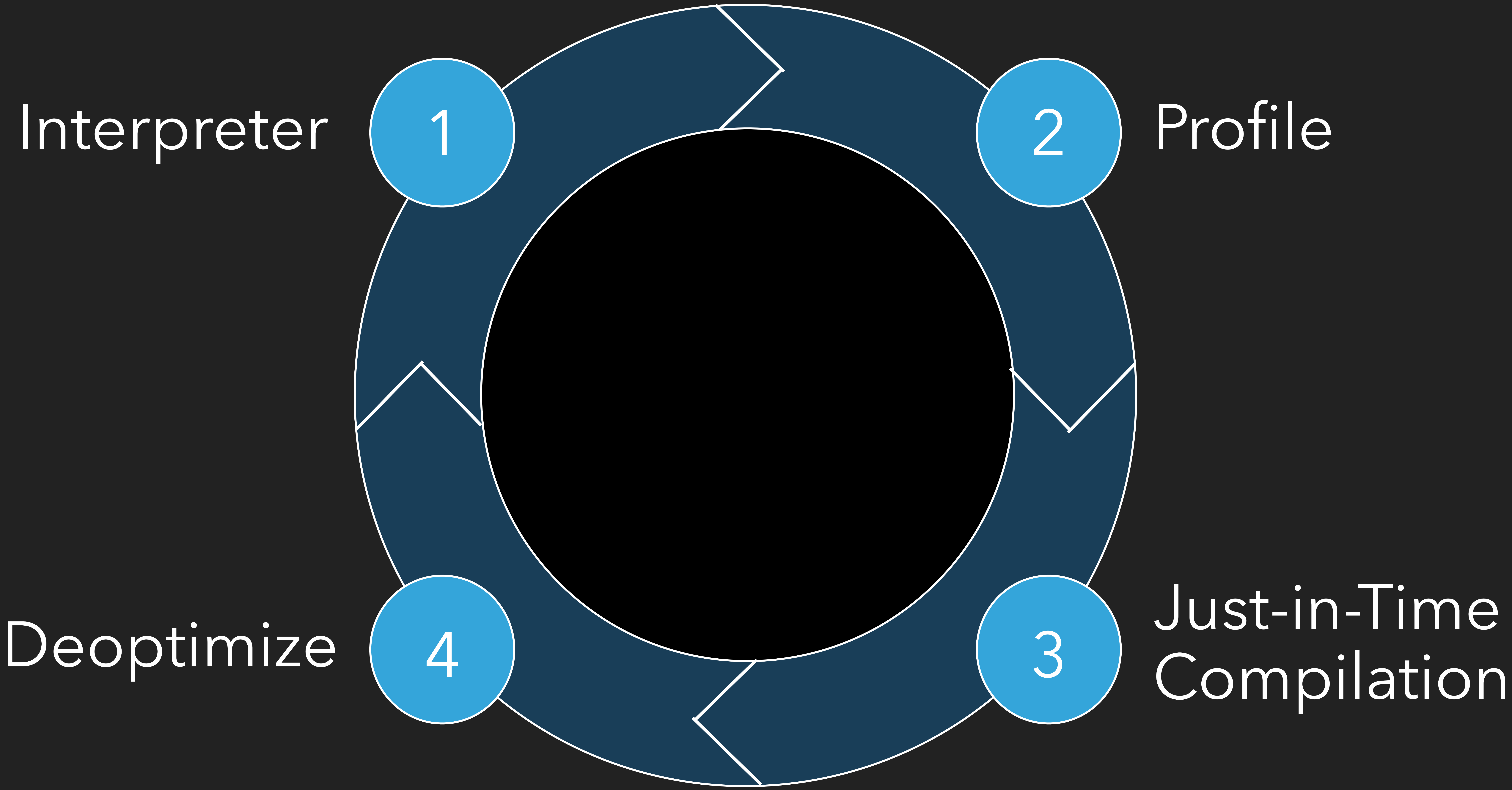
LEAD JIT DEVELOPER

AZUL SYSTEMS

@dougqh

dougqh@gmail.com

(SIMPLIFIED) CODE LIFECYCLE



TOPICS

WHAT TRIGGERS THE JUST-IN-TIME COMPILER?

WHY NOT AHEAD-OF-TIME?

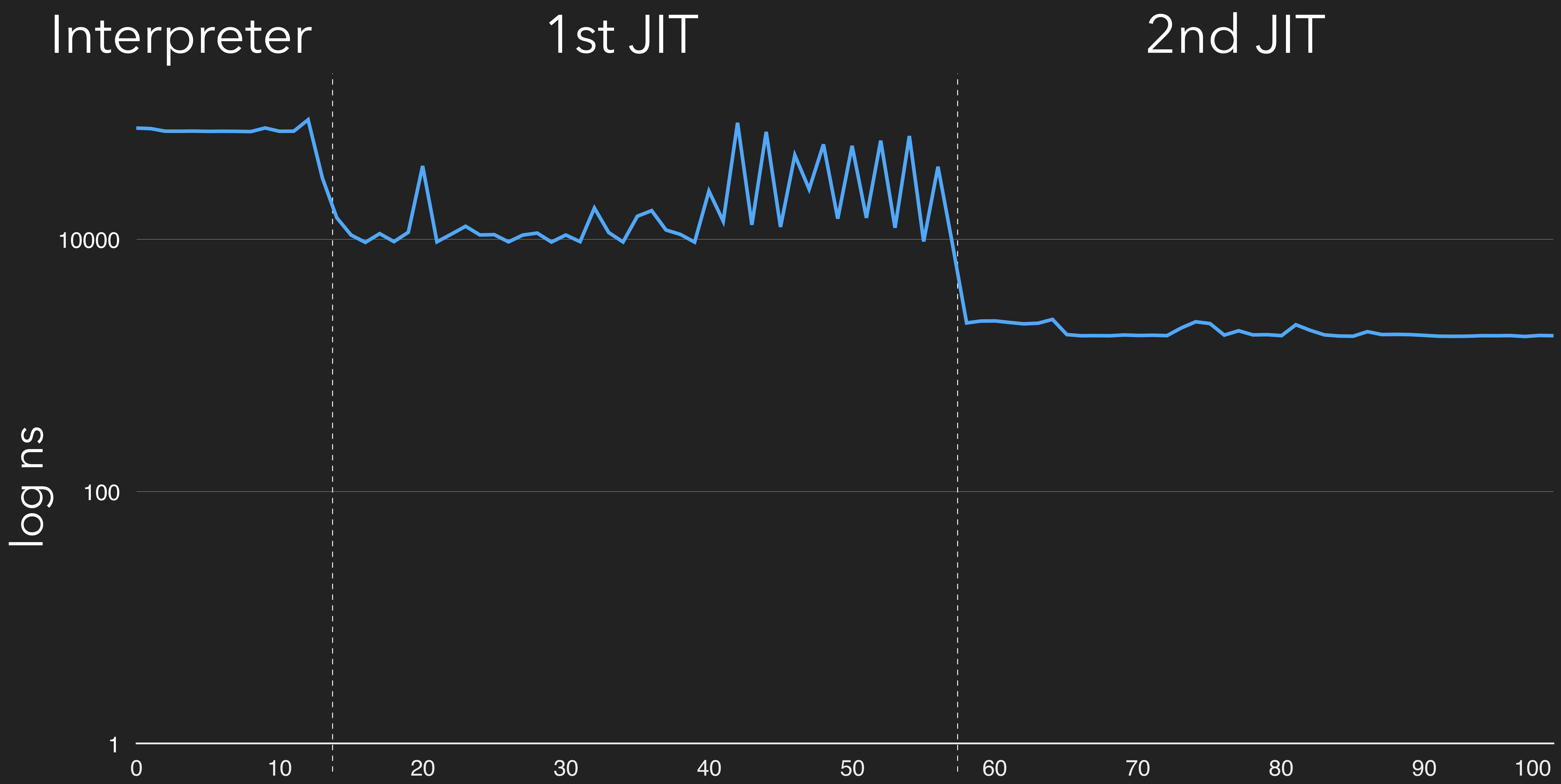
WHAT TRIGGERS THE UN-JIT?

IMPACT ON YOU

WHAT TRIGGERS THE JIT?

```java
public class SimpleProgram {
  static int blackhole;

  public static void main(String[] args) {
   int[] nums = randomInts(5_000);

    for ( int i = 0; i < 100; ++i ) {
      long startTime = System.nanoTime();

      blackhole = sum(nums);

      long endTime = System.nanoTime();
      System.out.printf("%d\t%d%n", i, endTime - startTime);
    }
  }
  …
}
```

```
 0  76484          45  12567          85  1709
 1  75764          46  46570          86  1858
 2  72254          47  25096          87  1762
 3  72189          48  56740          88  1767
 4  72364          49  14620          89  1760
 5  72014          50  55286          90  1735
 6  72177          51  14844          91  1710
 7  72040          52  60827          92  1706
 8  71721          53  12384          93  1710
 9  76635          54  66209          94  1726
10  72120          55  9638           95  1724
11  72241          56  37767          96  1729
12  89045          57  9350           97  1701
13  30692          58  2178           98  1733
14  14899          59  2255           99  1725
     …                …
```

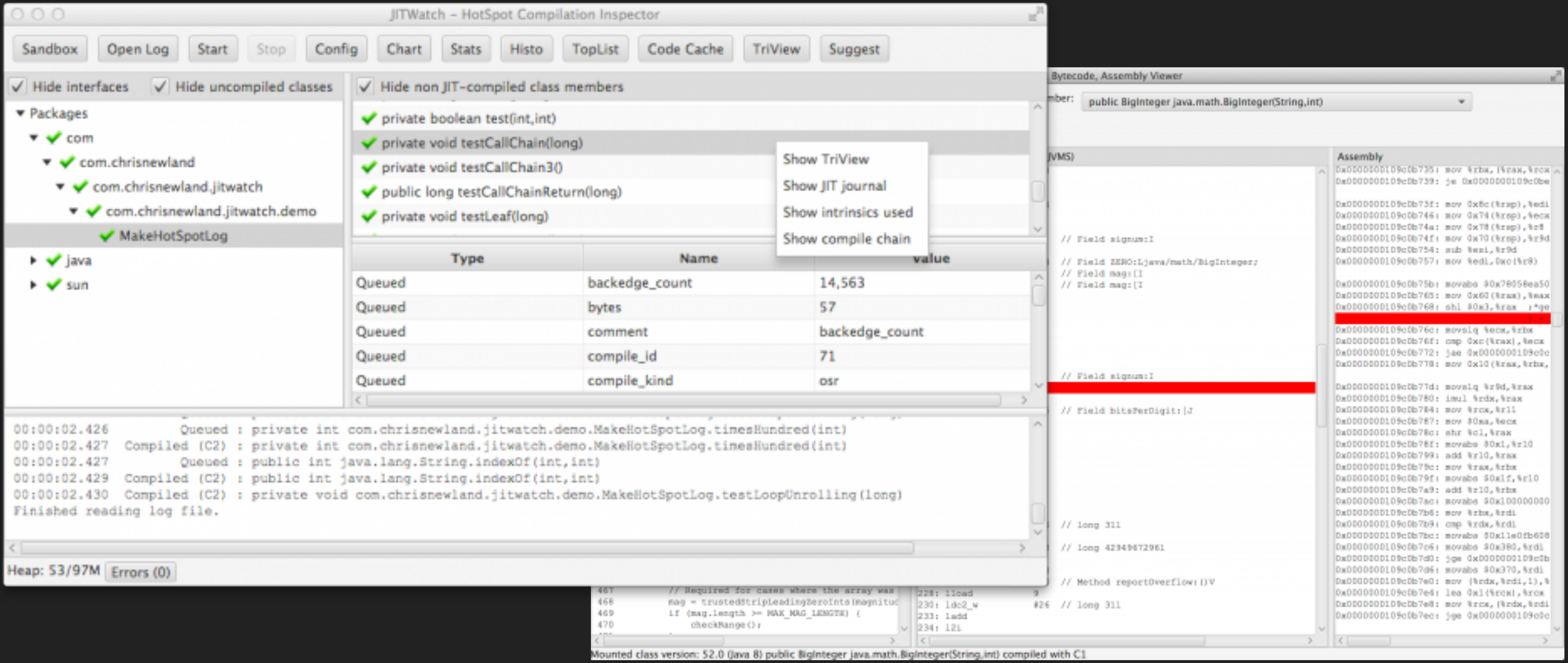

Interpreter
1st JIT
2nd JIT
log ns
10000
100
1
0
10
20
30
40
50
60
70
80
90
100

PRINT COMPILATION

```
java -XX:+PrintCompilation
```

```
61     1      3    java.lang.String::hashCode (55 bytes)
64     2      3    java.lang.String::charAt (29 bytes)
64     4      3    java.lang.String::indexOf (70 bytes)
64     3      3    java.lang.String::length (6 bytes)
65     5      3    java.lang.Object::<init> (1 bytes)
…
68    12      3    java.lang.String::getChars (62 bytes)
69    13      1    java.lang.ref.Reference::get (5 bytes)
78    14      3    java.lang.String::indexOf (7 bytes)
78    15      1    java.lang.Object::<init> (1 bytes)
81    16 %    3    example01.SimpleProgram::main @ 15 (48 bytes)
81    17      3    example01.SimpleProgram::main (48 bytes)
81    18 %    4    example01.SimpleProgram::main @ 15 (48 bytes)
```

JITWATCH



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

Type	Name	Value
Queued	backedge_count	14,563
Queued	bytes	57
Queued	comment	backedge_count
Queued	compile_id	71
Queued	compile_kind	osr

```
00:00:02.426         Queued : private int com.chrisnewland.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:02.427  Compiled (C2) : private int com.chrisnewland.jitwatch.demo.MakeHotSpotLog.timesHundred(int)
00:00:02.427         Queued : public int java.lang.String.indexOf(int,int)
00:00:02.429  Compiled (C2) : public int java.lang.String.indexOf(int,int)
00:00:02.430  Compiled (C2) : private void com.chrisnewland.jitwatch.demo.MakeHotSpotLog.testLoopUnrolling(long)
Finished reading log file.
```

Member: public BigInteger java.math.BigInteger(String,int)

```
// Required for cases where the array was
mag = trustedStripLeadingZeroInts(magnitud
if (mag.length >= MAX_MAG_LENGTH) {
    checkRange();
```

```
// Field signum:I
// Field ZERO:Ljava/math/BigInteger;
// Field mag:[I
// Field mag:[I
// Field signum:I
// Field bitsPerDigit:[J
// long 31l
// long 4294967296l
// Method reportOverflow:()V
228: lload          9
230: ldc2_w         #26  // long 31l
233: ladd
234: l2i
```

```
0x0000000109c0b735: mov %rbx,(%rax,%rcx
0x0000000109c0b739: je 0x0000000109c0be

0x0000000109c0b73f: mov 0x8c(%rsp),%edi
0x0000000109c0b746: mov 0x74(%rsp),%ecx
0x0000000109c0b74a: mov 0x78(%rsp),%r8
0x0000000109c0b74f: mov 0x70(%rsp),%r9d
0x0000000109c0b754: sub %esi,%r9d
0x0000000109c0b757: mov %edi,0xc(%r8)

0x0000000109c0b75b: movabs $0x78058ea50
0x0000000109c0b765: mov 0x60(%rax),%eax
0x0000000109c0b768: shl $0x3,%rax   ;*ge

0x0000000109c0b76c: movslq %ecx,%rbx
0x0000000109c0b76f: cmp 0xc(%rax),%ecx
0x0000000109c0b772: jae 0x0000000109c0c
0x0000000109c0b778: mov 0x10(%rax,%rbx,

0x0000000109c0b77d: movslq %r9d,%rax
0x0000000109c0b780: imul %rdx,%rax
0x0000000109c0b784: mov %rcx,%r11
0x0000000109c0b787: mov $0xa,%ecx
0x0000000109c0b78c: shr %cl,%rax
0x0000000109c0b78f: movabs $0x1,%r10
0x0000000109c0b799: add %r10,%rax
0x0000000109c0b79c: mov %rax,%rbx
0x0000000109c0b79f: movabs $0x1f,%r10
0x0000000109c0b7a9: add %r10,%rbx
0x0000000109c0b7ac: movabs $0x100000000
0x0000000109c0b7b6: mov %rbx,%rdi
0x0000000109c0b7b9: cmp %rdx,%rdi
0x0000000109c0b7bc: movabs $0x11e0fb608
0x0000000109c0b7c6: movabs $0x380,%rdi
0x0000000109c0b7d0: jge 0x0000000109c0b
0x0000000109c0b7d6: movabs $0x370,%rdi
0x0000000109c0b7e0: mov (%rdx,%rdi,1),%
0x0000000109c0b7e4: lea 0x1(%rcx),%rcx
0x0000000109c0b7e8: mov %rcx,(%rdx,%rdi
0x0000000109c0b7ec: jge 0x0000000109c0c
```

Mounted class version: 52.0 (Java 8) public BigInteger java.math.BigInteger(String,int) compiled with C1

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

LOG COMPILATION XML

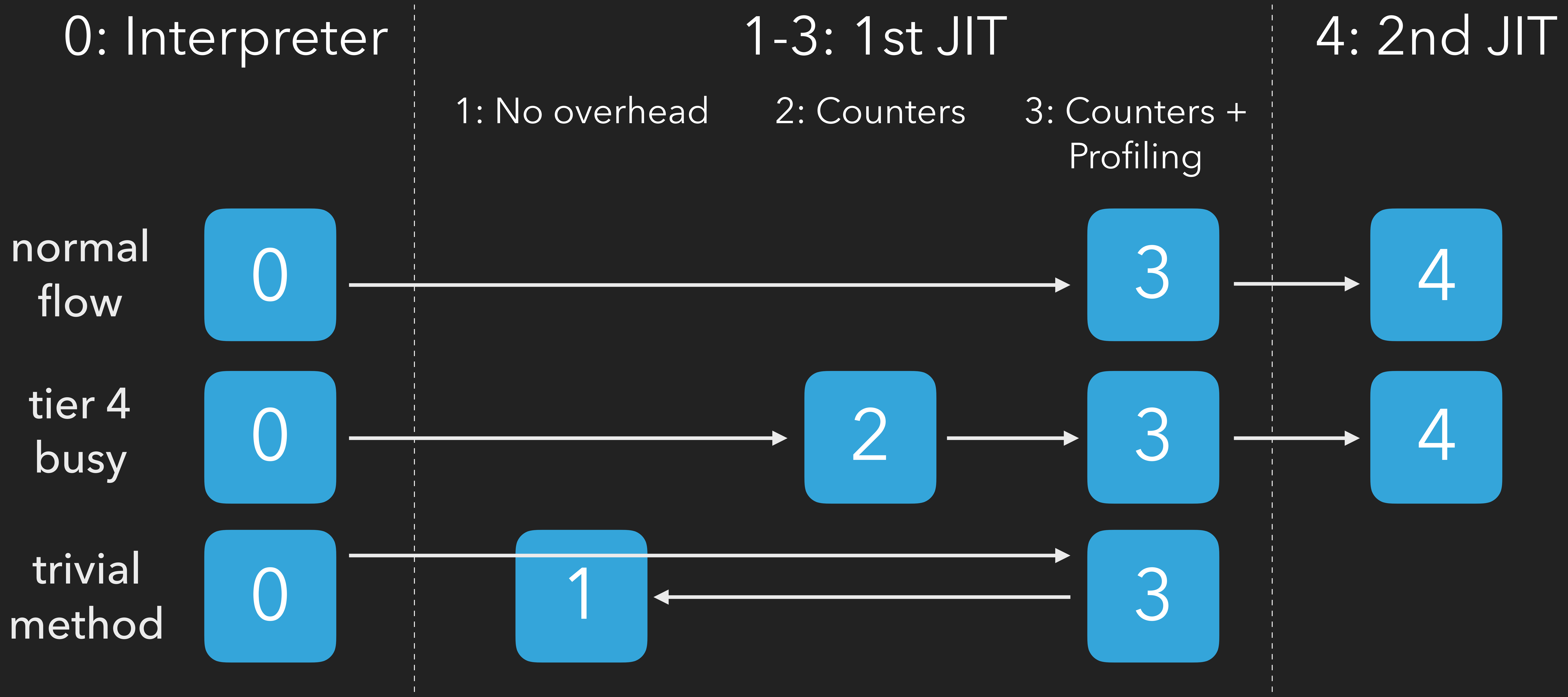
```
java -XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation
```

```
<task_queued compile_id=‘5’ level=‘3' stamp='0.130'
  method='java/lang/Object &lt;init&gt; ()V' bytes='1'
  count='1536' iicount='1536'
  comment='tiered' hot_count='1536'/>

<task compile_id='2' level='3' stamp=‘0.126’
  method='java/lang/String charAt (I)C' bytes='29'
  count='1955' iicount='1955' >
  …
  <task_done success='1' stamp=‘0.127' nmsize='616' count='2269'/>
</task>
```

TIERED COMPILATION

	C1: Client	C2: Server
Compilation Speed	Fast	Slow (4X)
Execution Speed	Slow	Fast (2X)

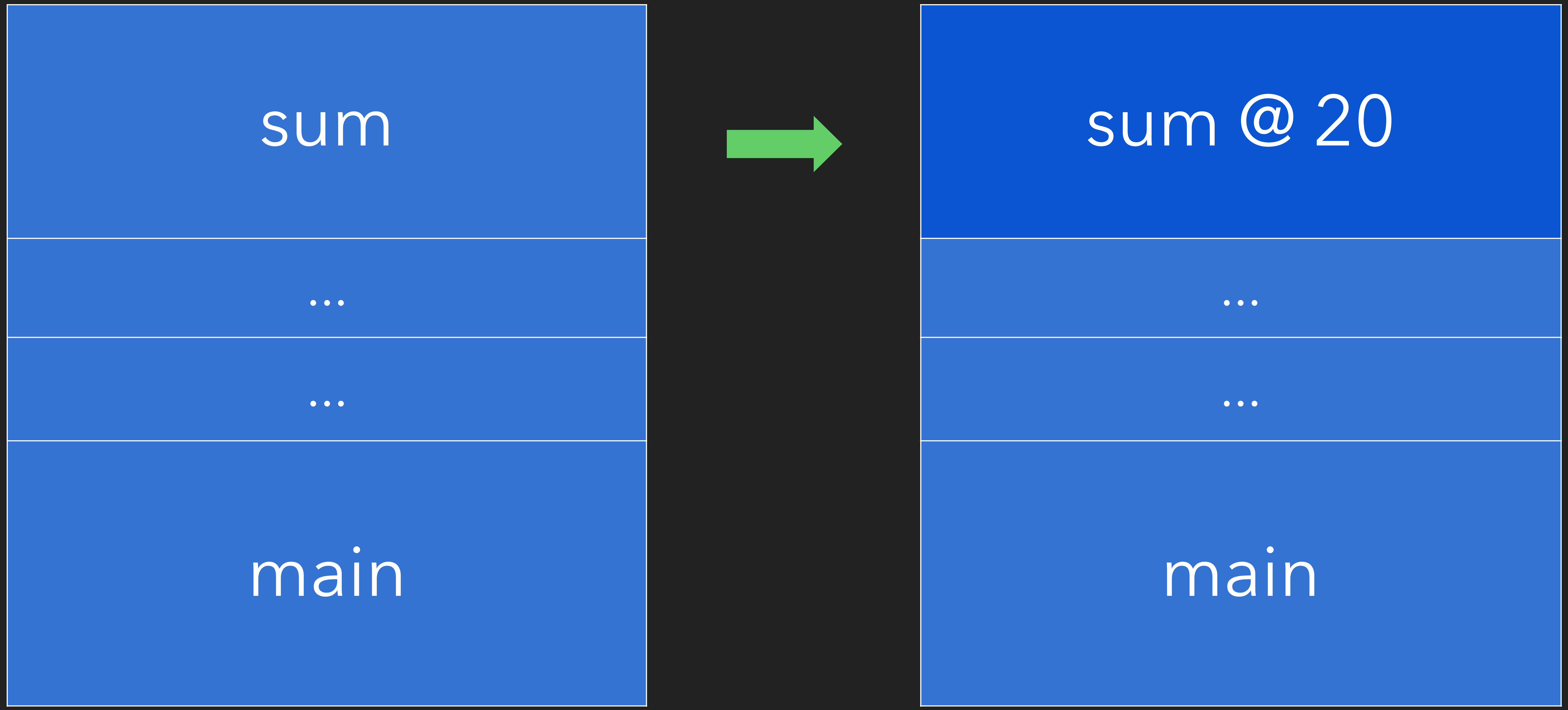
TIERED COMPILATION



METHOD JIT
VS.
TRACE JIT

LOOP COMPILATIONS

ON STACK REPLACEMENTS (OSR)



🟦 interpreter frame
🟦 compiled frame

OSR IS IMPORTANT TO OUR EXAMPLE

```
96    54 %   3   …SimpleProgram::main @ 15 (78 bytes)
12  38362
13  35214
14  37139
15  36598
15  36838
16  36429
17  18046

101   80 %  4   …SimpleProgram::main @ 15 (78 bytes)
62  2831
63  2371
64  2288
```

BUT WHEN?

INVOCATION COUNTER

Counter > Invocation Threshold
Hot Methods

BACKEDGE (LOOP) COUNTER

Counter > Backedge Threshold
Hot Loops

Invocation + Backedge Counters > Compile Threshold
Medium Hot Methods with Medium Hot Loops

BUT WHEN?

```
java -XX:+PrintFlagsFinal
```

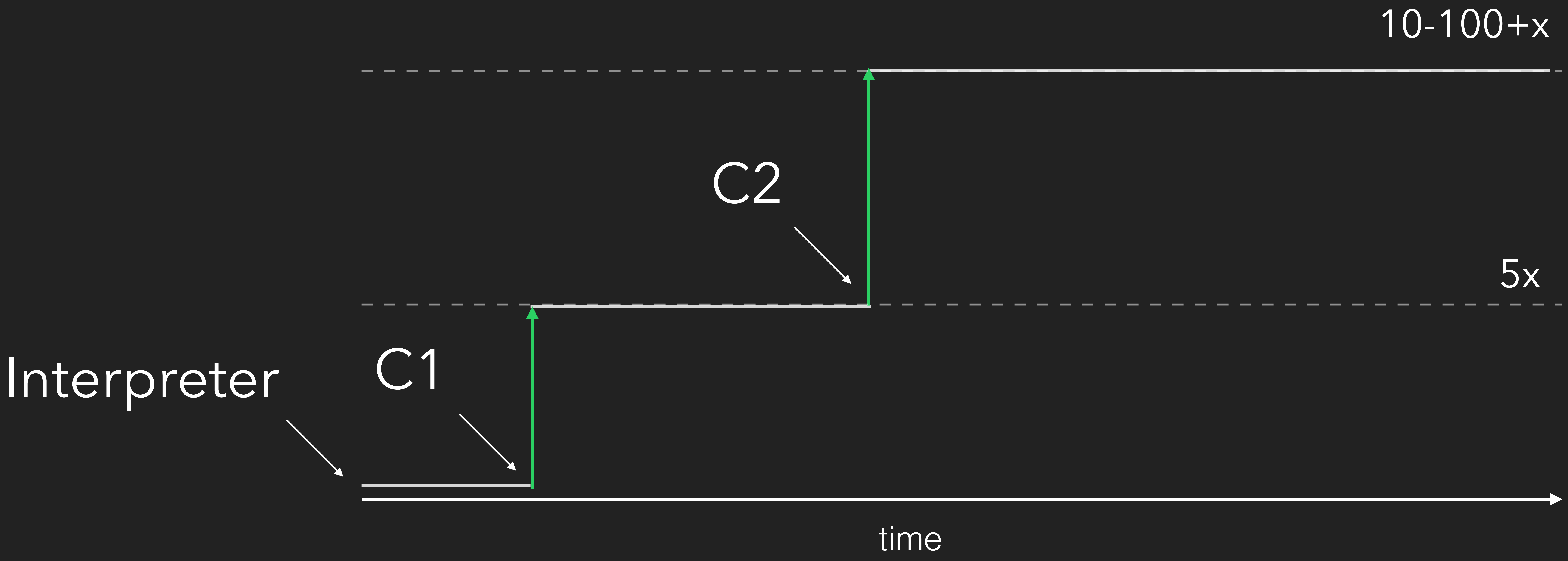
JAVA 8 (TIERED) THRESHOLDS

```
intx Tier2BackEdgeThreshold         = 0
intx Tier2CompileThreshold          = 0
intx Tier3BackEdgeThreshold         = 60000
intx Tier3CompileThreshold          = 2000
intx Tier3InvocationThreshold       = 200
intx Tier3MinInvocationThreshold    = 100
intx Tier4BackEdgeThreshold         = 40000
intx Tier4CompileThreshold          = 15000
intx Tier4InvocationThreshold       = 5000
intx Tier4MinInvocationThreshold    = 600
```

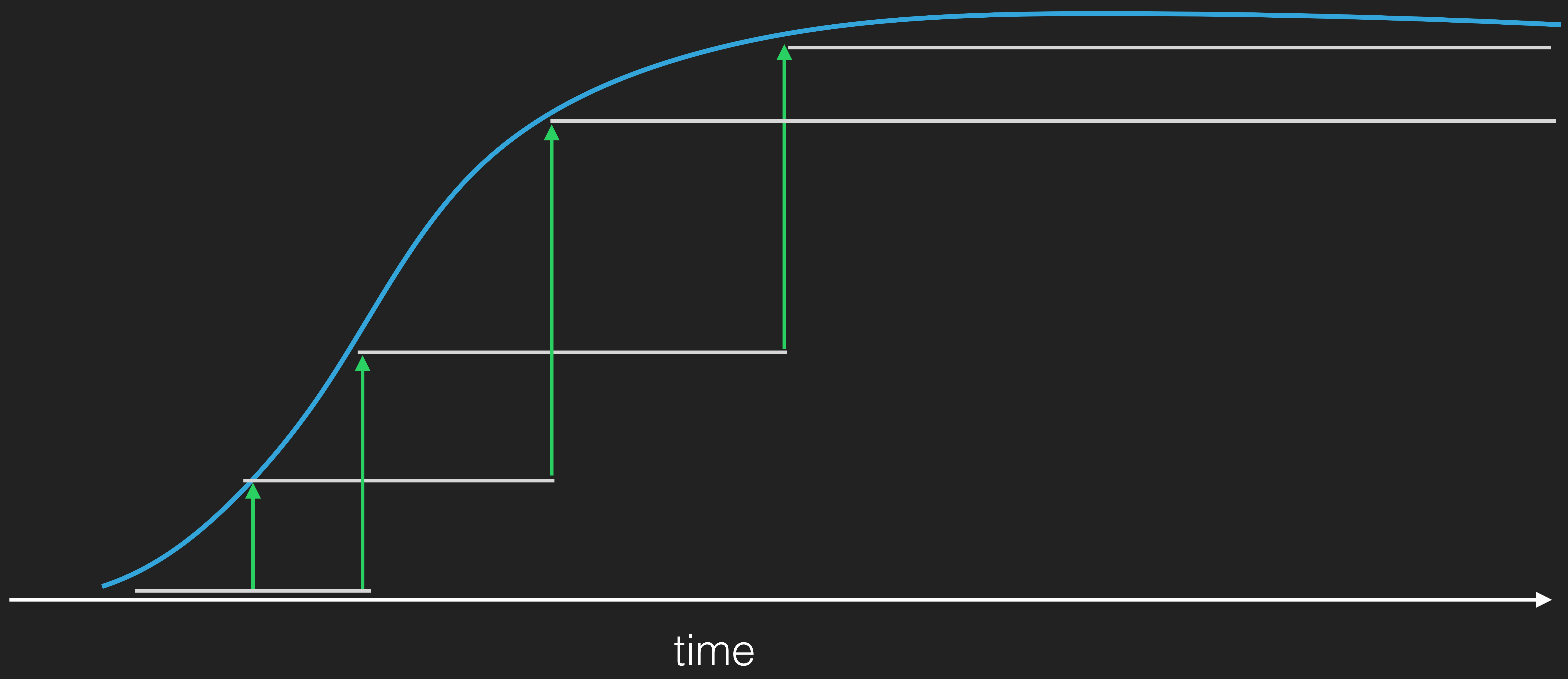
JAVA 7 (NON-TIERED) THRESHOLDS

```
intx BackEdgeThreshold              = 100000
intx CompileThreshold               = 10000
```

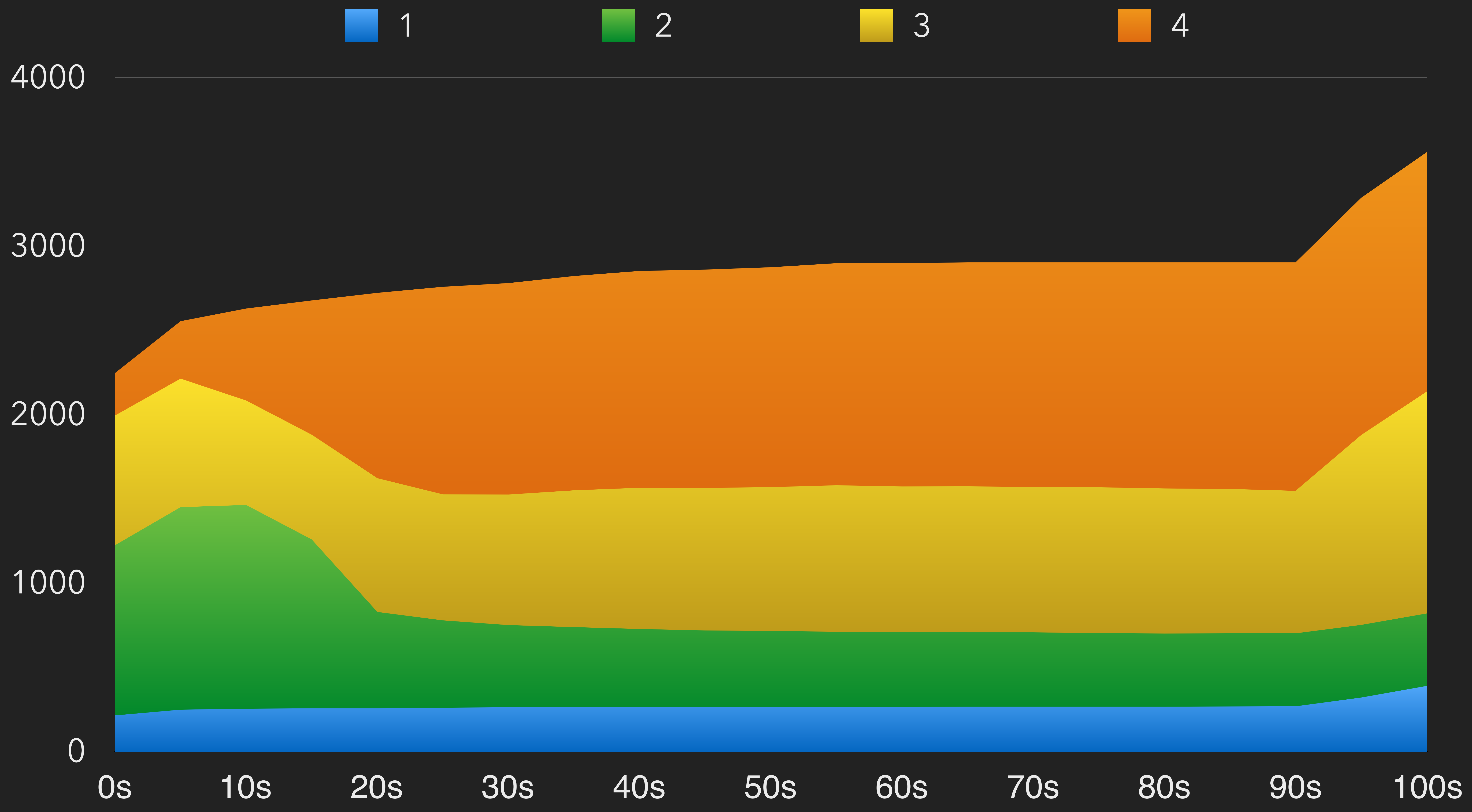
MENTAL MODEL



IN AGGREGATE



ACTIVE COMPILATIONS



WHY NOT AHEAD-OF-TIME?

WARM-UP &
RUN ONCE CODE

FROM MY PERSPECTIVE
JAVA IS A DYNAMIC LANGUAGE

DYNAMICALLY LOADED

LAZY INITIALIZED

RUNTIME CHECKED

DYNAMICALLY DISPATCHED

IT CANNOT GET MUCH MORE DYNAMIC.

DYNAMICALLY & LAZY LOADED

WHAT DOES JIT DO WITH AN UNLOADED CLASS?

```
if ( cond ) {
  return Class1.getStatic();
} else {
  return Class2.getStatic();
}
```

UNLOADED

JIT DOESN'T KNOW…

Fields

Methods

Parent Class

Interfaces

⋮

Anything

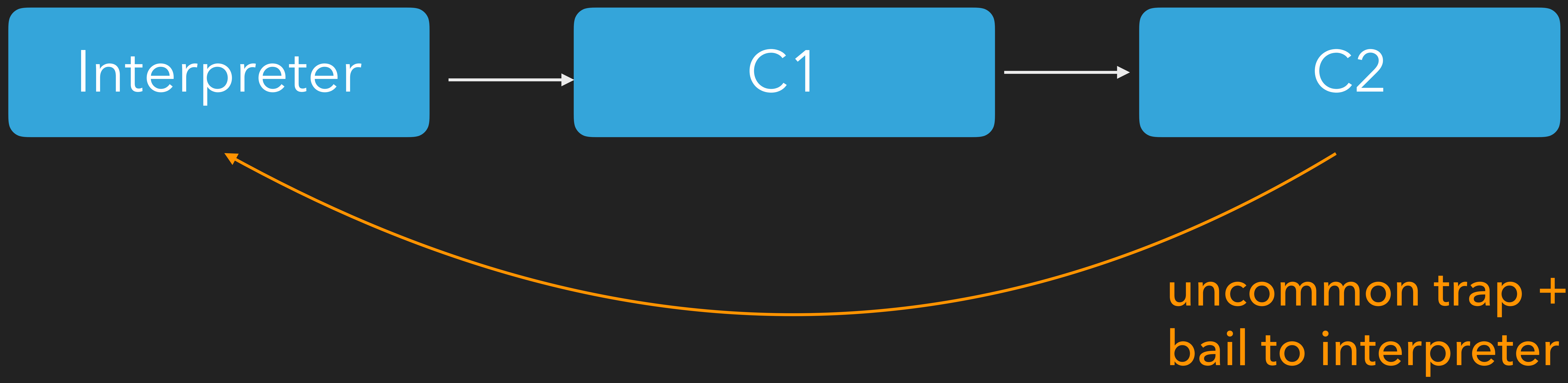
```
if ( cond ) {
  return Class1.getStatic();
} else {
  return Class2.getStatic();
}
```

$\downarrow$

```
if ( cond ) {
  return Class1.getStatic();
} else {
  uncommon_trap(:unloaded);
}
```

Give Up!

BAIL TO INTERPRETER



LAZY INITIALIZED

INITIALIZE CLASS AT FIRST...

Static Field Access

Static Method Call

New

Initialization of Child

INITIALIZATION CHECKING

```
MyClass.getStatic()
```

↓

```
if ( !vm.is_init(MyClass)) {
  vm.init(MyClass);
}
MyClass.getStatic()
```

Bloats size by 20%

Slows down by 5-10%

http://joeduffyblog.com/2015/12/19/safe-native-code/

JIT RUNS LATE

MOST CLASSES ARE ALREADY INITIALIZED

$$\begin{tikzcd}
 & \texttt{MyClass.getStatic()} \arrow[d, "\text{At compile time...}"] & \\
 & \texttt{vm.is\_init(MyClass)} \arrow[dl, "\text{initialized}"'] \arrow[dr, "\text{uninitialized}"] & \\
\texttt{MyClass.getStatic()} & & \texttt{uncommon\_trap(:uninitialized)}
\end{tikzcd}$$

WHAT IF A CLASS FAILS TO INIT?

UNINITIALIZED FOREVER

```
public class UninitializedForever {
 public static void main(String[] args) {
    for ( int i = 0; i < 10_000_000; ++i ) {
      try {
        new Uninitialized();
      } catch ( Throwable t ) {
        // ignore
      }
    }
  }
}
```

```
static class Uninitialized {
  static {
    if ( true ) {
      throw new RuntimeException();
    }
  }
}
```

THAT'S UNFORTUNATE

```
 612   24 % !   3       …UninitializedForever::main @ 5 (25 bytes)
 621   25   !   3       …UninitializedForever::main (25 bytes)
 952   26 % !   4       …UninitializedForever::main @ 5 (25 bytes)
 953   24 % !   3       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
1024   26 % !   4       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
1033   27 % !   3       …UninitializedForever::main @ 5 (25 bytes)
1405   28 % !   4       …UninitializedForever::main @ 5 (25 bytes)
1406   27 % !   3       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
1481   28 % !   4       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
1489   29 % !   3       …UninitializedForever::main @ 5 (25 bytes)
1855   30 % !   4       …UninitializedForever::main @ 5 (25 bytes)
 …
7339   55 % !   3       …UninitializedForever::main @ 5 (25 bytes)
7690   56 % !   4       …UninitializedForever::main @ 5 (25 bytes)
7690   55 % !   3       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
7761   56 % !   4       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
7769   57 % !   3       …UninitializedForever::main @ 5 (25 bytes)
```

LOG COMPILATION XML

```
java -XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation
```

TRAP INSTALLATION

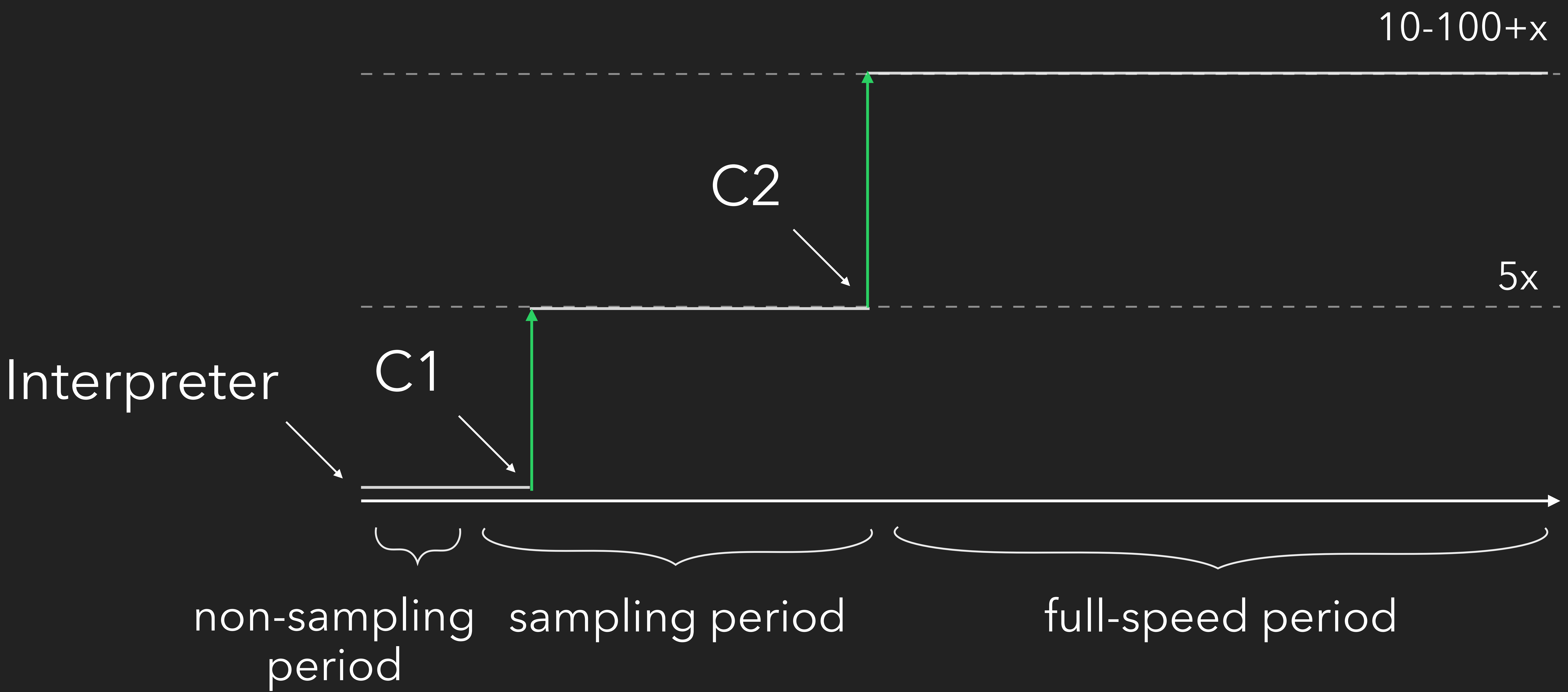
```
<task compile_id='26' stamp='1.403'
 compile_kind='osr' osr_bci='5'
 method='…UninitializedForever main ([Ljava/lang/String;)V'
 decompiles='1' uninitialized_traps='1'  …>
 …
 <bc code='187' bci='5'/>
 <klass id='835' name='…Uninitialized' flags='8'/>
 <uncommon_trap
   bci='5' reason='uninitialized'
   action='reinterpret' klass='835'/>
…
</task>
```

TRIGGERING TRAP

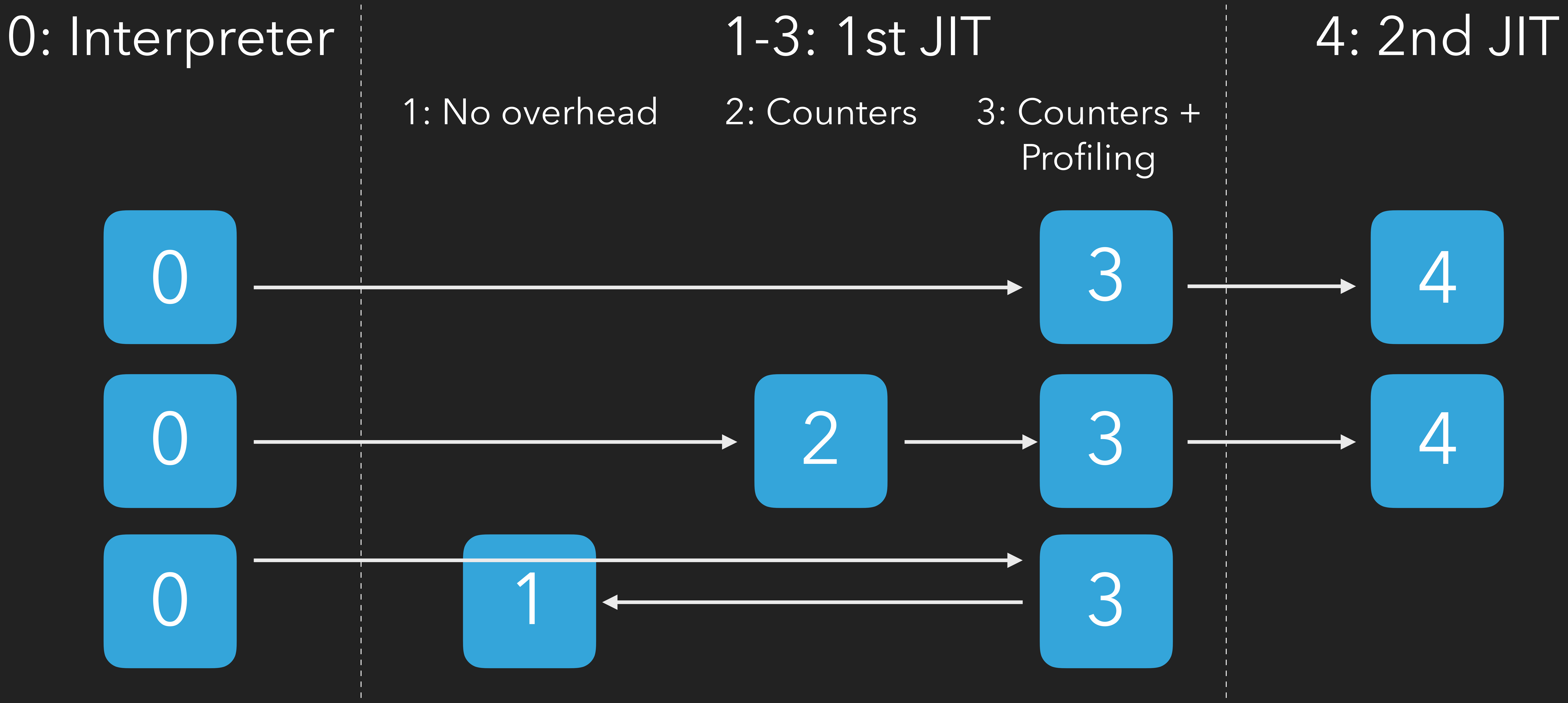
```
<uncommon_trap thread='7171' stamp='1.038'
 compile_id='24' compile_kind='osr'
 compiler='C2' level='4'
 reason='uninitialized' action='reinterpret'>
 …
</uncommon_trap>

<make_not_entrant thread='7171' stamp='1.038'
 compile_id='24' compile_kind='osr'
 compiler='C2' level='4' />
```

COMPILING LATER PROVIDES MORE INFO TO SPECULATE.

REFINED MENTAL MODEL



TIERED COMPILATION



WHAT ELSE TRIGGERS UN-JIT?

(DEOPTIMIZATION)

NULL CHECKING

```
public static void main(String[] args) {
  // warm-up hotMethod – enough to JIT
  for ( int i = 0; i < 20_000; ++i ) {
    hotMethod("hello");
  }

  for ( int i = 0; i < 10; ++i ) {
    System.out.printf("tempting fate %d%n", i);
    try {
      hotMethod(null);
    } catch ( NullPointerException e ) { }
  }
}
```

```
static final void hotMethod(
  Object value)
{
  value.hashCode();
}
```

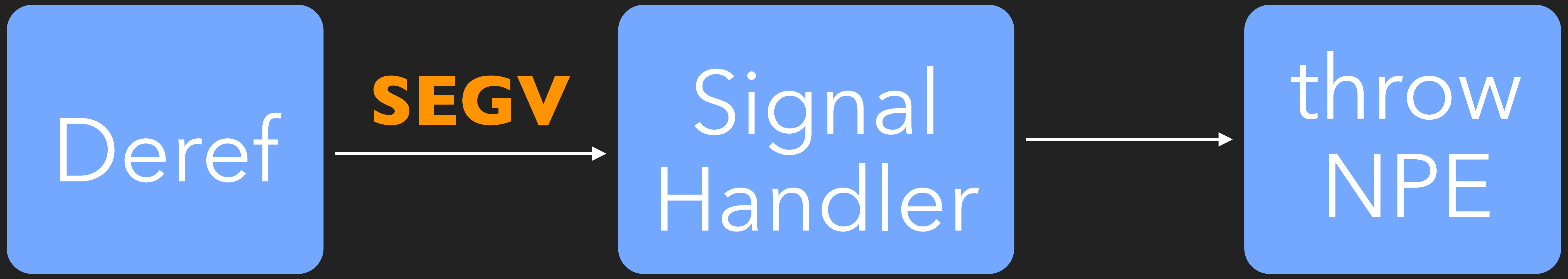
```
     81     1       java.lang.String::hashCode (55 bytes)
     84     2       …NullCheck::hotMethod (6 bytes)
     85     3 % !   …NullCheck::main @ 5 (69 bytes)
tempting fate 0
tempting fate 1
tempting fate 2
   5089     2       …NullCheck::hotMethod (6 bytes)   made not entrant
tempting fate 3
tempting fate 4
tempting fate 5
tempting fate 6
tempting fate 7
tempting fate 8
tempting fate 9
```

IMPLICIT NULL CHECK

```
if ( value == null ) {
  throw new NullPointerException();
}
value.hashCode();
```

$\longleftarrow$ Possible,
but improbable

```
0x10795f9cc: mov    0x8(%rsi),%r10d
     ; implicit exception: dispatches to 0x10795fe1d
```


$$\begin{tikzcd}\text{Deref} \arrow[r, "\text{SEGV}"] & \text{Signal Handler} \arrow[r] & \text{throw NPE}\end{tikzcd}$$

SLOW WHEN IT HAPPENS,
FAST WHEN IT DOESN’T.

UNREACHED / UNSTABLE IF

```
public static volatile Object thing = null;

public static void main(final String[] args) {
  for ( int i = 0; i < 20_000; ++i ) {
    hotMethod();
  }
  Thread.sleep(5_000); // wait for JIT
  thing = new Object();

  for ( int i = 0; i < 20_000; ++i ) {
    hotMethod();
  }
  Thread.sleep(5_000); // wait for JIT again
}
```

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("");
  else
    System.out.print("");
}
```

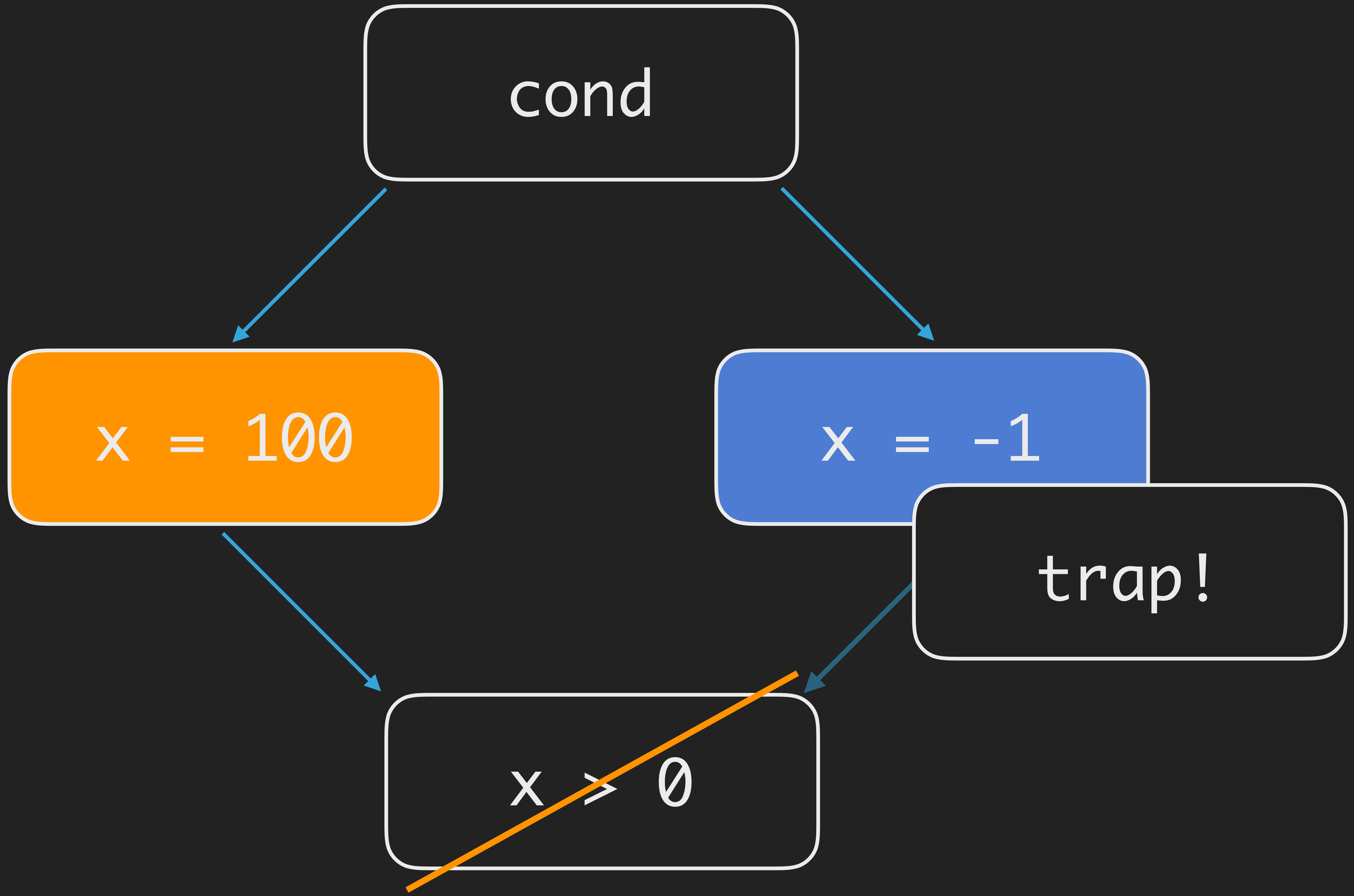
```
  79   18    3    …Unreached::hotMethod (26 bytes)
  83   33    4    …Unreached::hotMethod (26 bytes)
5089   33    4    …Unreached::hotMethod (26 bytes)   made not entrant
5089   36    3    …Unreached::hotMethod (26 bytes)
5090   38    4    …Unreached::hotMethod (26 bytes)
```

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("");
  else
    System.out.print("");
}
```

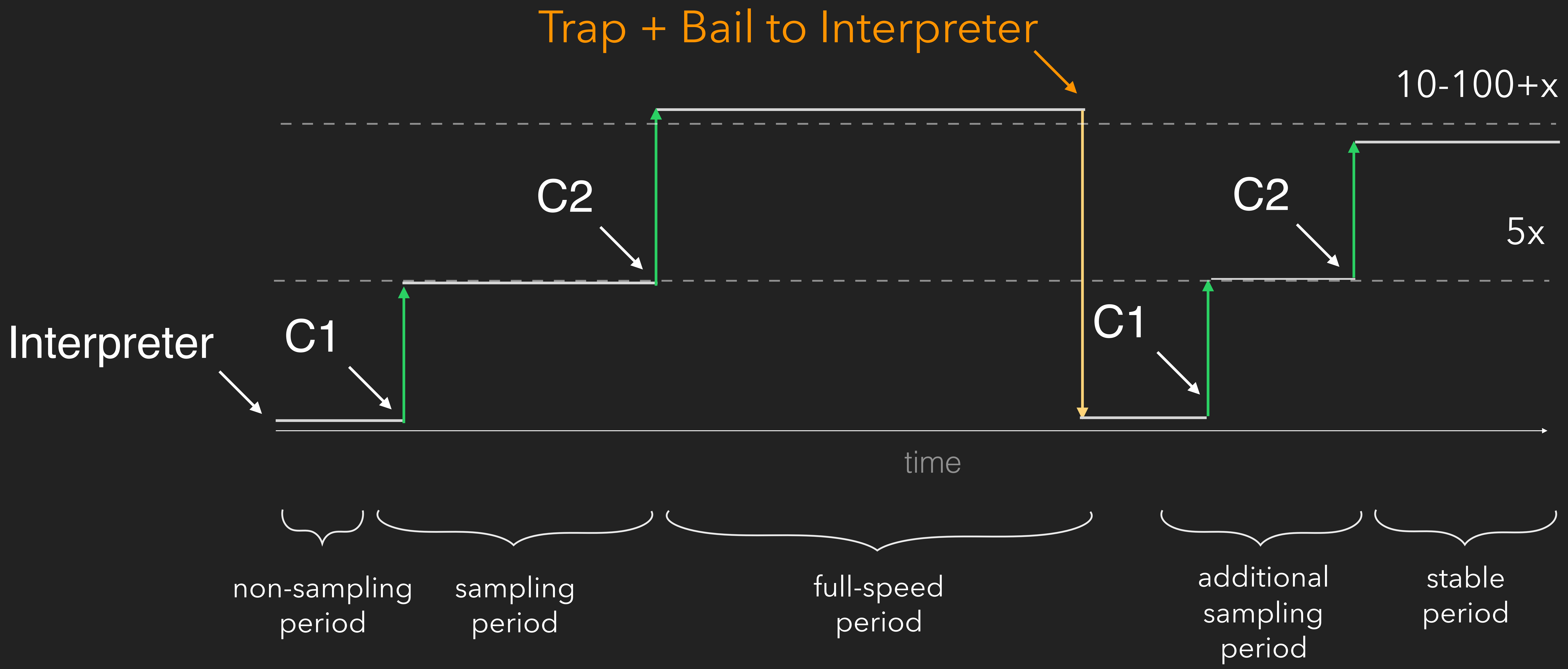
```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("");
  else
    uncommon_trap(:unreached);
}
```

```
<bc code='199' bci='3'/>
<branch target_bci='17'
  taken='0' not_taken='5800'
  cnt='5800' prob='never'/>
<uncommon_trap
  bci='3' reason='unstable_if'
  action='reinterpret'
  comment='taken never'/>
```

```
<uncommon_trap thread='7171' stamp='5.104'
  compile_id='29' compiler='C2' level='4'
  reason='unstable_if' action='reinterpret' >
  <jvms
    method='…Unreached hotMethod ()V'
    bci='3'…/>
</uncommon_trap>
```


cond
x = 100
x = -1
trap!
x > 0

REVISED MENTAL MODEL



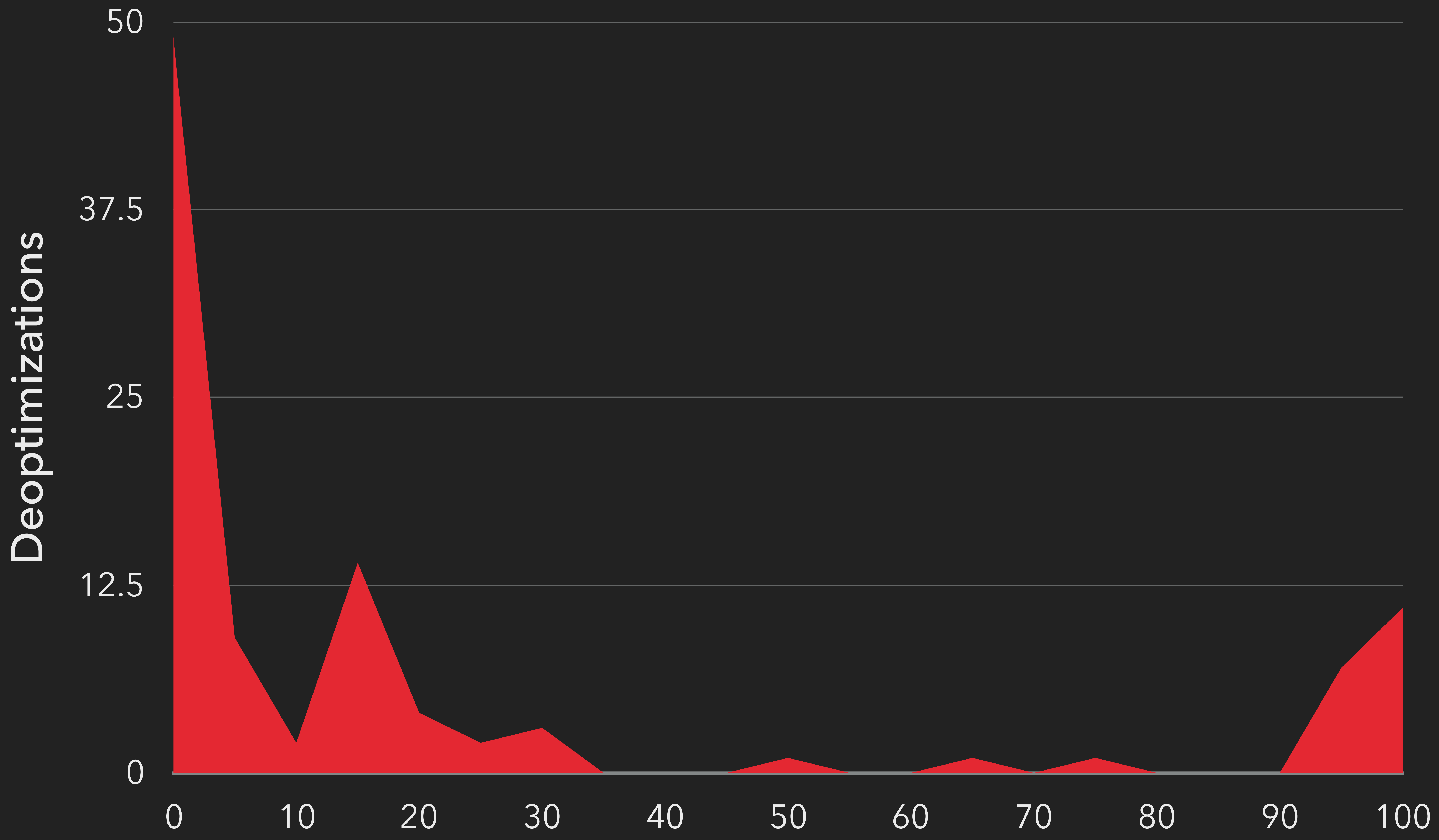
```
func.apply(x);
```

$\downarrow$

```
if ( func.getClass() == Square.class ) {
  x * x;
} else if ( func.getClass() == Sqrt.class ) {
  Math.sqrt(x)
} else {
  …
}
```

```
<klass id='780' name='Square' flags='1'/>
<klass id='781' name='Sqrt' flags='1'/>
<call method='783' count='23161'
  prof_factor='1' virtual='1' inline='1'
  receiver='780' receiver_count=‘19901'
  receiver2='781' receiver2_count='3260'/>
<predicted_call bci='3' klass=‘780'/>
<predicted_call bci='3' klass=‘781'/>
<uncommon_trap bci='3'
  reason=‘bimorphic'
  action='maybe_recompile'/>
```

DEOPTIMIZATIONS



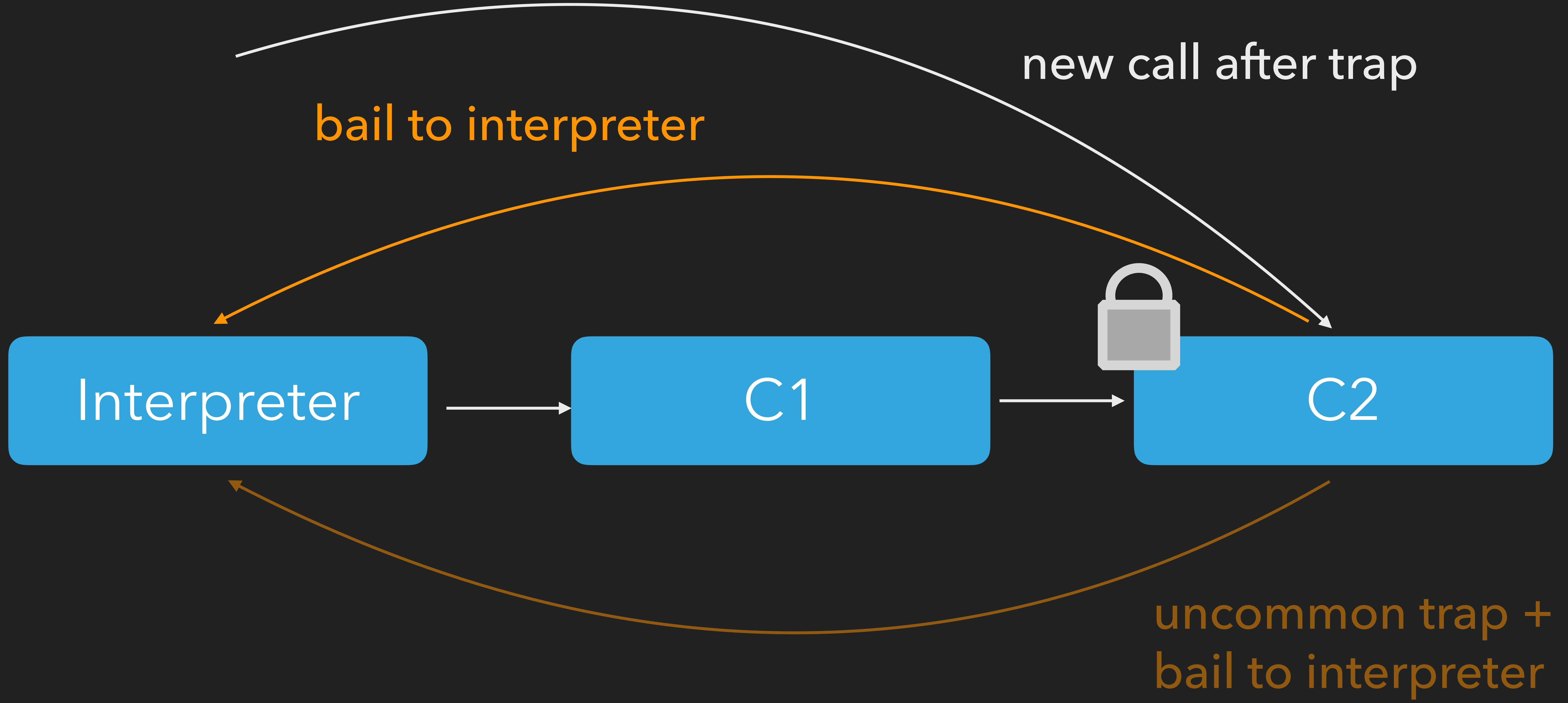
DOES THIS MATTER?

SPECULATIVE OPTIMIZATIONS

25+%

EVEN MORE INSIDE A LOOP

BAIL TO INTERPRETER + LOCK



HARD TO IDENTIFY DISRUPTION

25–50ms GOING SLOW — NOT STOPPED

AND SOME DO STOP THE WORLD

WHAT'S YOUR GOAL?

STARTUP TIME

PEAK PERFORMANCE

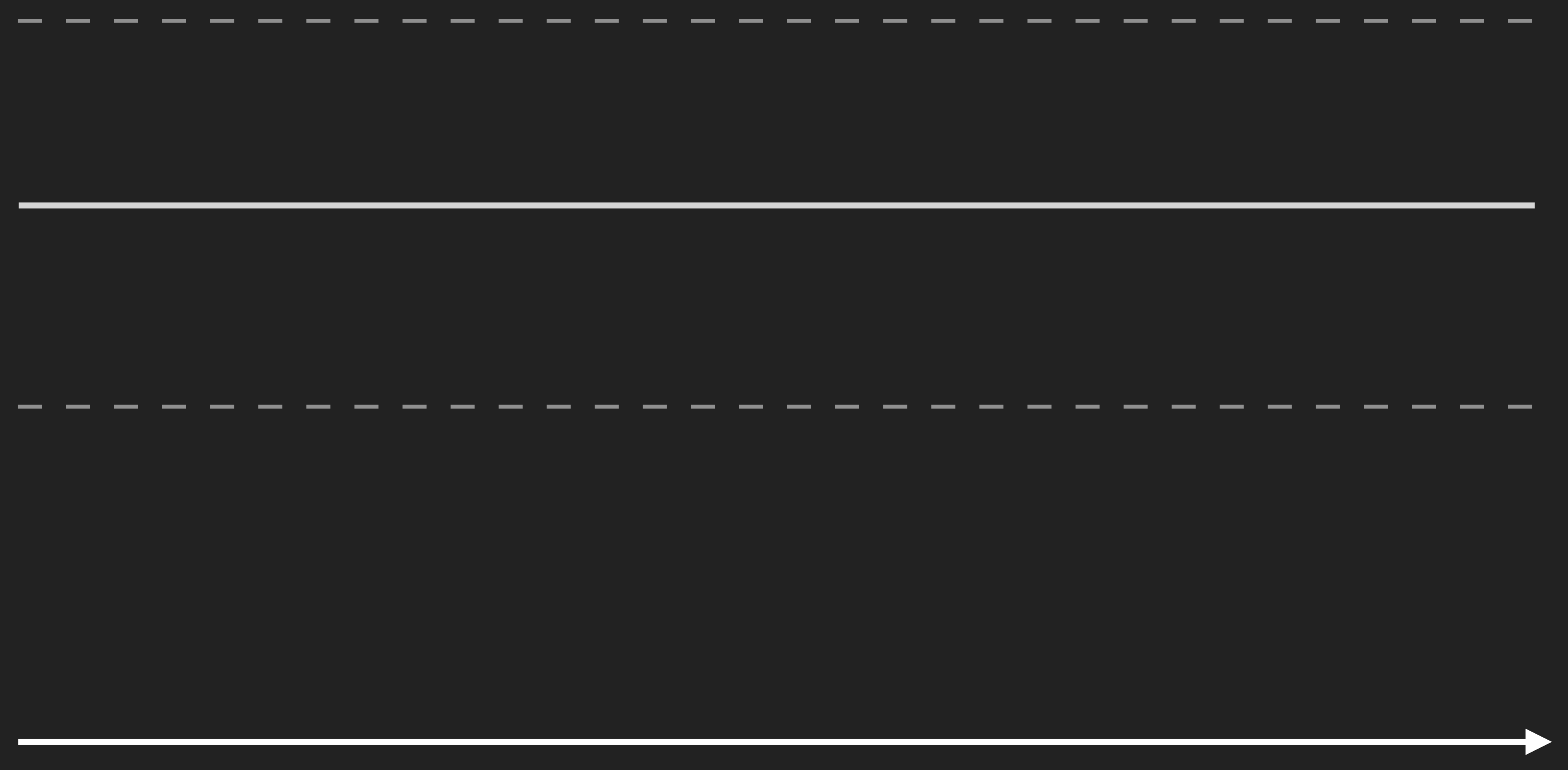
FIRST RESPONSE PERFORMANCE

PREDICTABLE PERFORMANCE

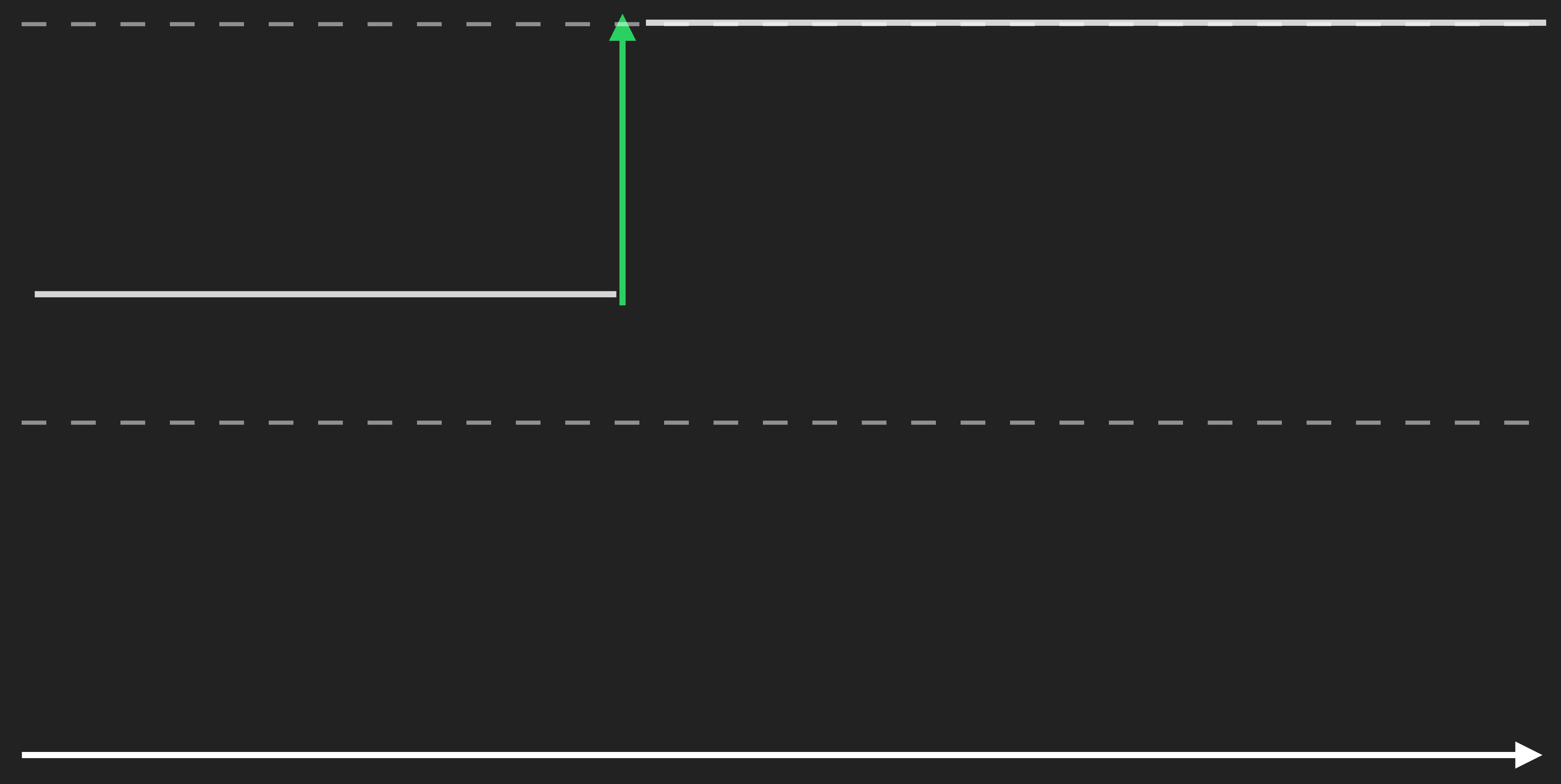
WHAT DOES THAT MEAN FOR A JIT OR AOT?

PICK YOUR PERFORMANCE GOAL

STABLE PERFORMANCE

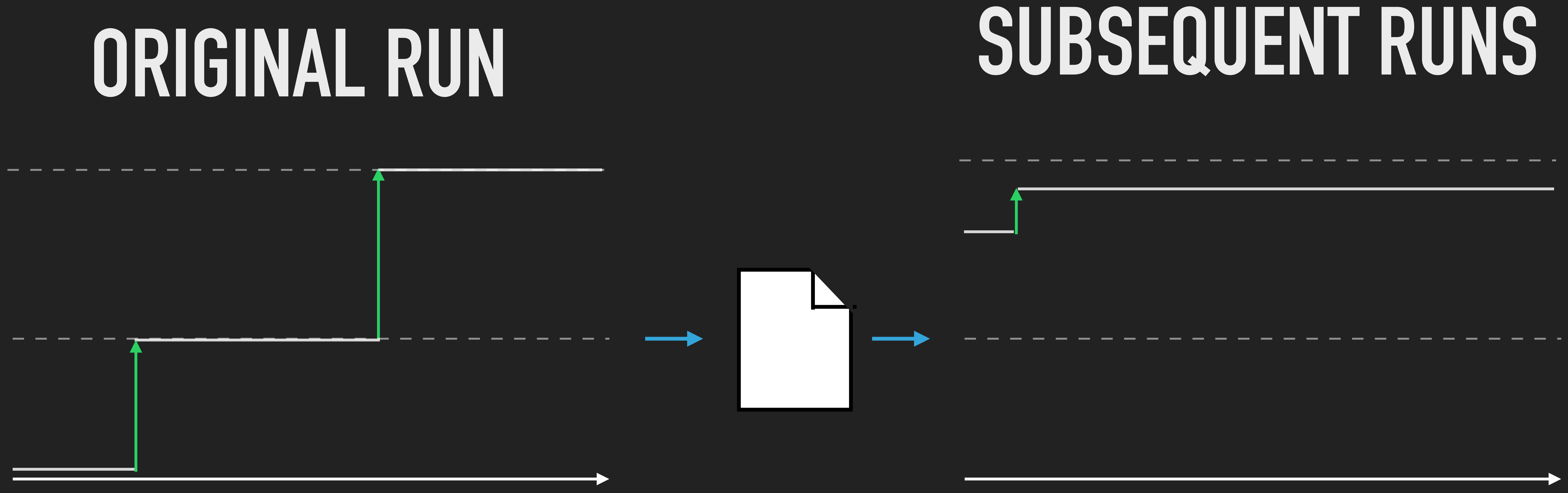


PEAK PERFOMANCE



https://www.youtube.com/watch?v=Xybzyv8qbOc

AZUL READYNOW



EACH APPROACH HAS PROS & CONS

STARTUP TIME

PEAK PERFORMANCE

FIRST RESPONSE PERFORMANCE

PREDICTABLE PERFORMANCE

IT CAN GET MORE DYNAMIC.

CLASS GENERATION ~~EVAL~~ == EVIL

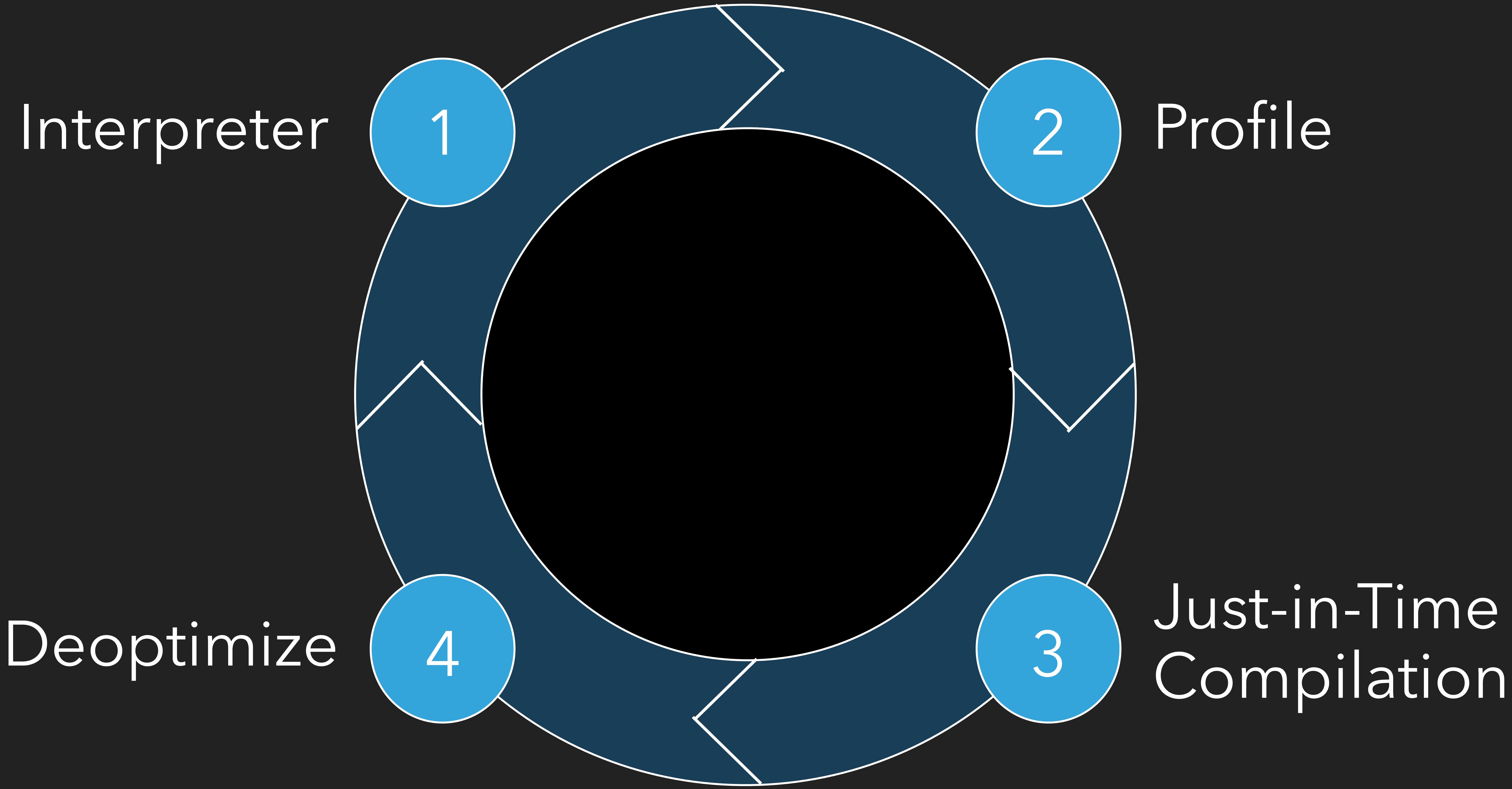
AOP   Rules Engines   JSON Libraries   Spring

xml.transform   ORMs: Hibernate   Reflection

Derby   Java 8 Lambdas   Dynamic Proxies

THAT'S A CHALLENGE FOR AN AOT.

(SIMPLIFIED) CODE LIFECYCLE



REFERENCES

ALEKSEY SHIPILЁV

http://shipilev.net/

JAVA SPECIALIST NEWSLETTER

http://www.javaspecialists.eu/

PSYCHOMATIC LOBOTOMY SAW

http://psy-lob-saw.blogspot.com/