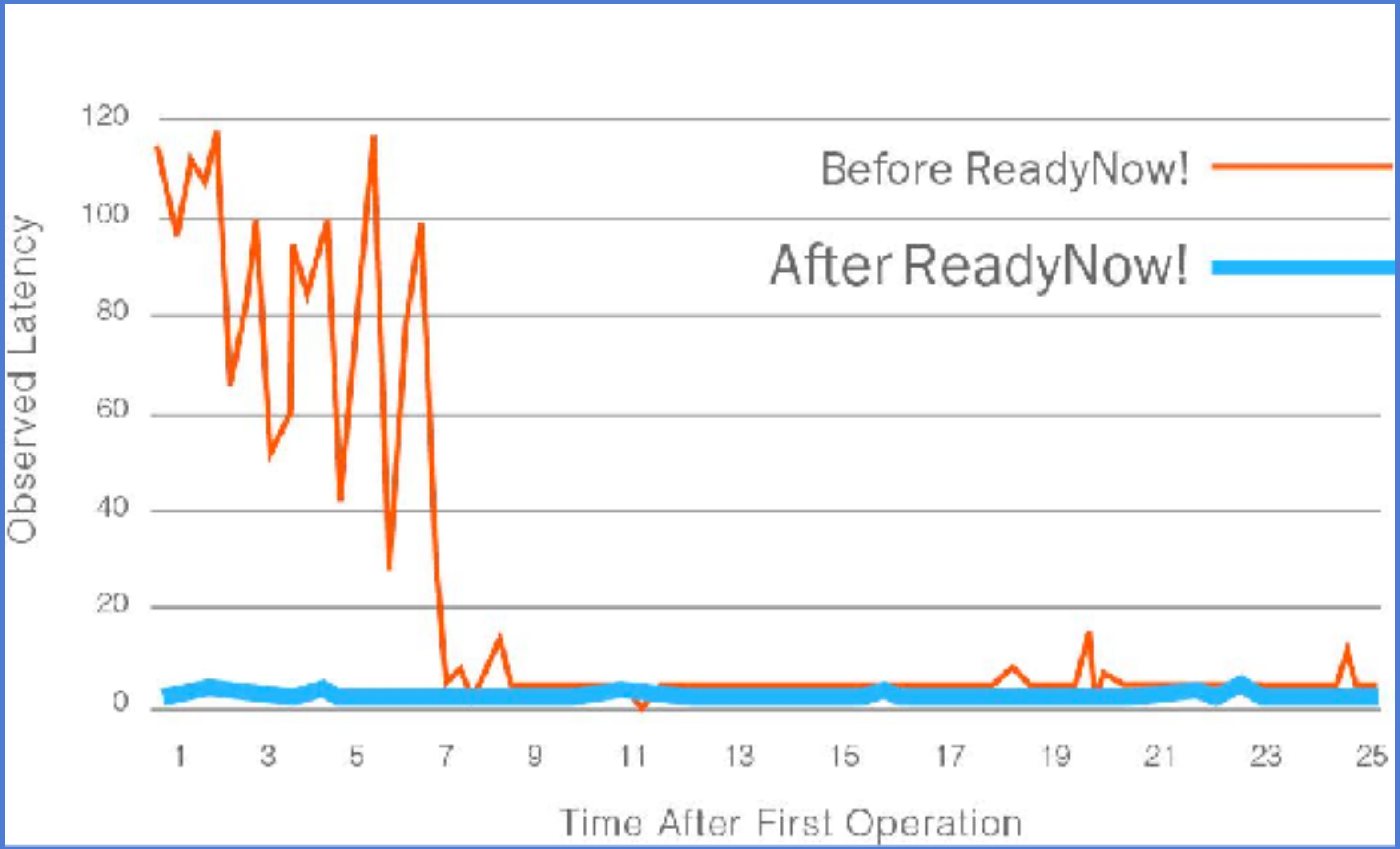
Understanding the JIT's Tricks

Douglas Q. Hawkins
@dougqh
VM Engineer
AZUL SYSTEMS®

Zing® Highly Scalable VM

Continuously Concurrent Compacting Collector

ReadyNow! for Low Latency Applications



http://www.azulsystems.com/solutions/zing/readynow

Zulu® Multi-Platform OpenJDK

Cloud Support including Docker and Azure

Embedded Support

http://zulu.org/

GOAL: Understand…

```
ArrayList<E>.forEach(Consumer<? super E> action) {
  for ( int i = 0; i < this.size; i++ ) {
    action.accept(this.elementData[i]);
  }
}
```

5 Lines?

Actual Implementation

```
ArrayList<E>.forEach(Consumer<? super E> action) {
  Objects.requireNonNull(action);
  final int expectedModCount = modCount;
  final E[] elementData = (E[]) this.elementData;
  final int size = this.size;

  for (int i=0; modCount == expectedModCount && i < size; i++) {
    action.accept(elementData[i]);
  }
  if (modCount != expectedModCount) throw new CME();
}
```

The Just-in-Time Compiler is a *Compiler*, but It’s a…

Profile Guided Speculatively Optimizing Compiler

So We Need to Understand…

Static Optimizations

Speculative Optimizations

Inter-procedural Analysis

Deoptimization

How They Fit Together

REAL GOAL:

Understand What You Can Rely on OpenJDK's JIT to do

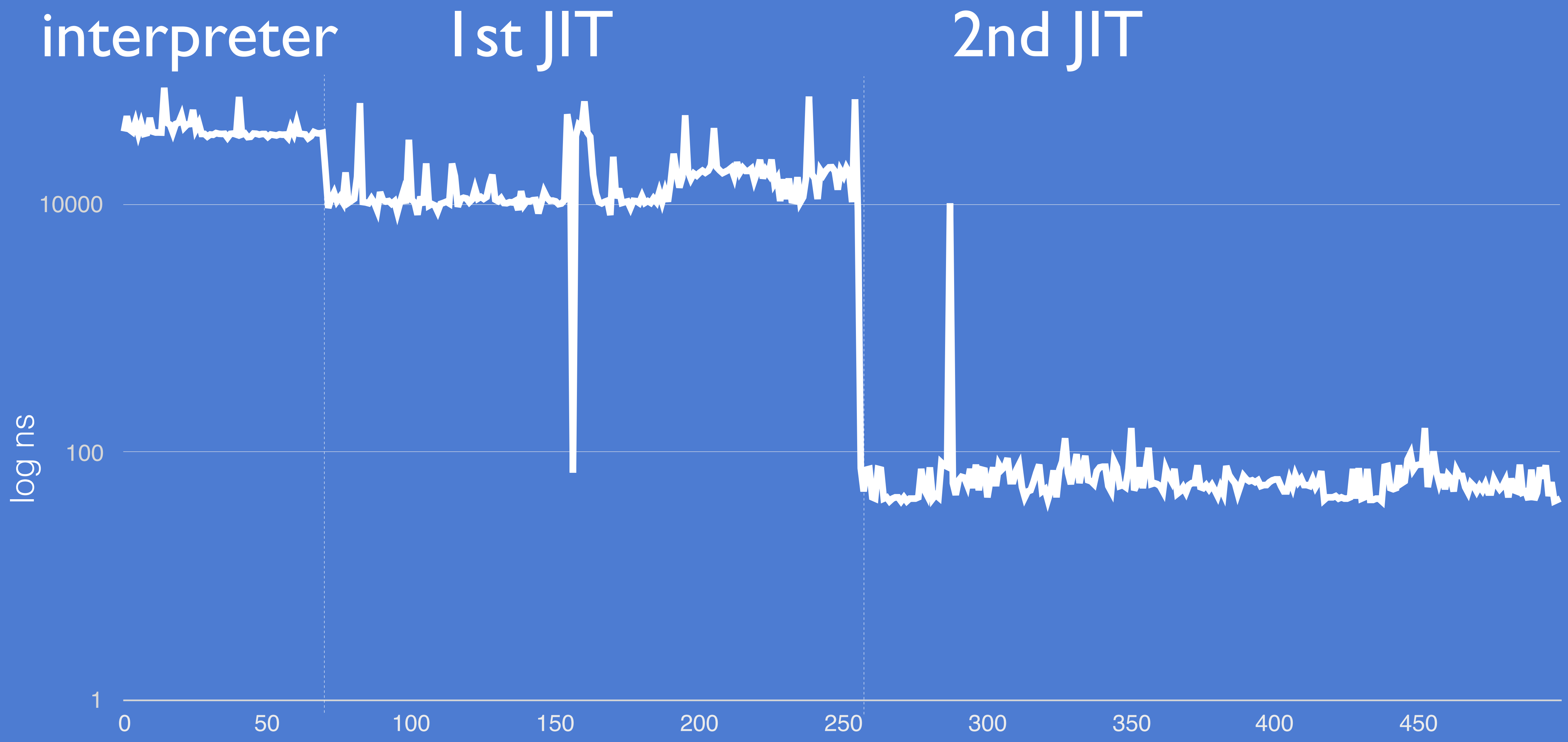
But First…

When Do We JIT?

http://openjdk.java.net/groups/hotspot/docs/RuntimeOverview.html

```
public class Allocation {
  static final int CHUNK_SIZE = 1_000;

  public static void main(String[] args) {
    for ( int i = 0; i < 500; ++i ) {
      long startTime = System.nanoTime();

      for ( int j = 0; j < CHUNK_SIZE; ++j ) {
        new Object();
      }

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

Warm-Up



interpreter
1st JIT
2nd JIT
log ns
10000
100
1
0
50
100
150
200
250
300
350
400
450

-XX:+PrintCompilation

```
 80   29       3     java.util.HashMap::newNode (13 bytes)
 81   30       3     java.util.HashMap::afterNodeInsertion
 81   31       3     java.lang.String::indexOf (7 bytes)

 96   73 %     3     example01a.Allocation::main @ 15 (78 bytes)
101   99 %     4     example01a.Allocation::main @ 15 (78 bytes)
```

When *Exactly*?

Two Compilations
of One Method?

What about Tiers 1 & 2?

Thresholds

-XX:+PrintFlagsFinal

Java 8 (Tiered) Thresholds

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

Java 7 (Non-tiered) Thresholds

```
intx BackEdgeThreshold                   = 100000         {pd product}
intx CompileThreshold                    = 10000          {pd product}
```

Counters

Invocation Counter
Backedge (Loop) Counter

Invocation Counter > Invocation Threshold
Hot Methods

Backedge Counter > Backedge Threshold
Hot Loops

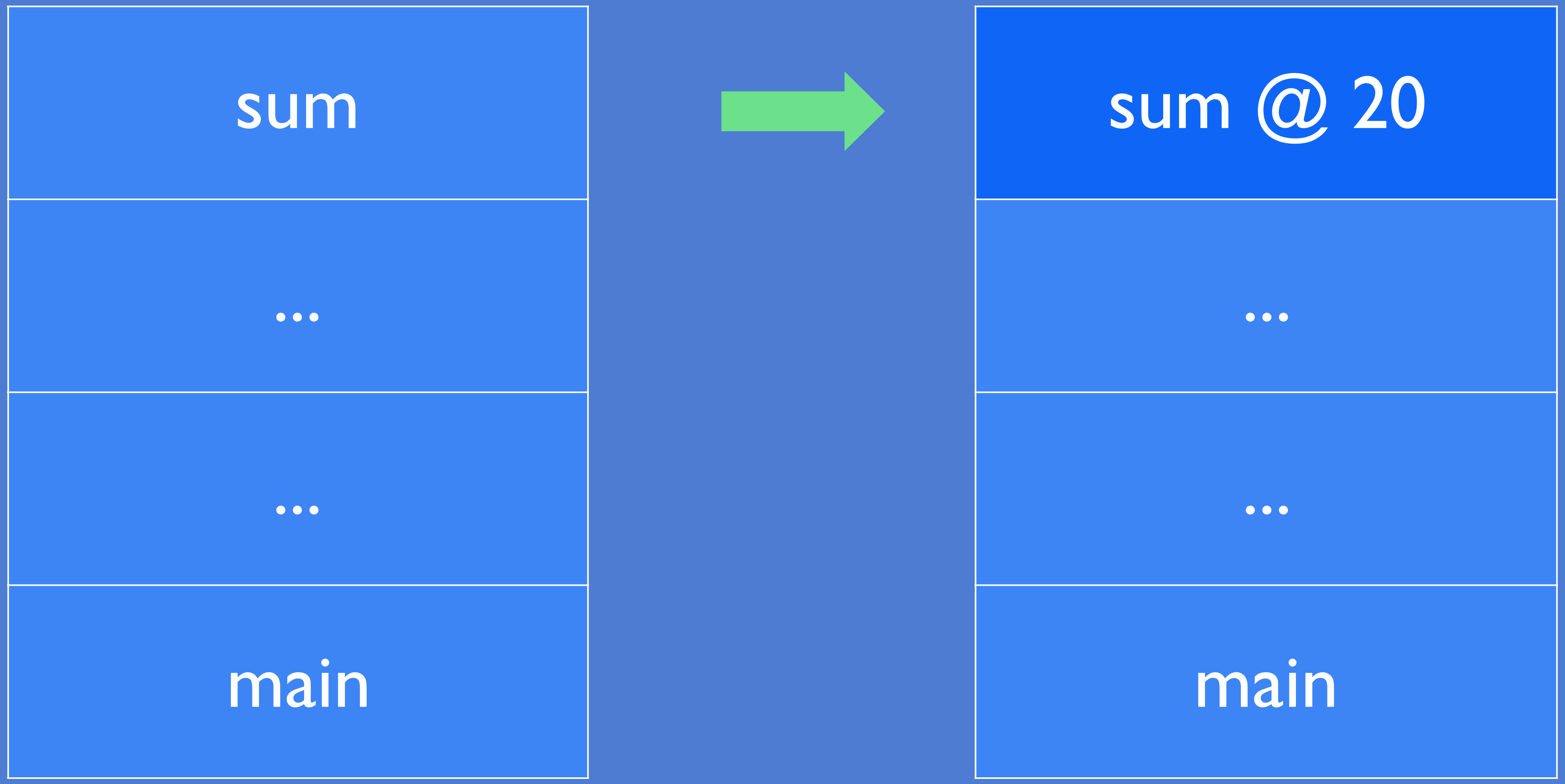
Invocation + Backedge Counter > Compile Threshold
Medium Hot Methods with Medium Hot Loops

Method or *Loop* JIT

If Loop Count (Backedges) > Threshold, Compile Loop

On-Stack Replacement

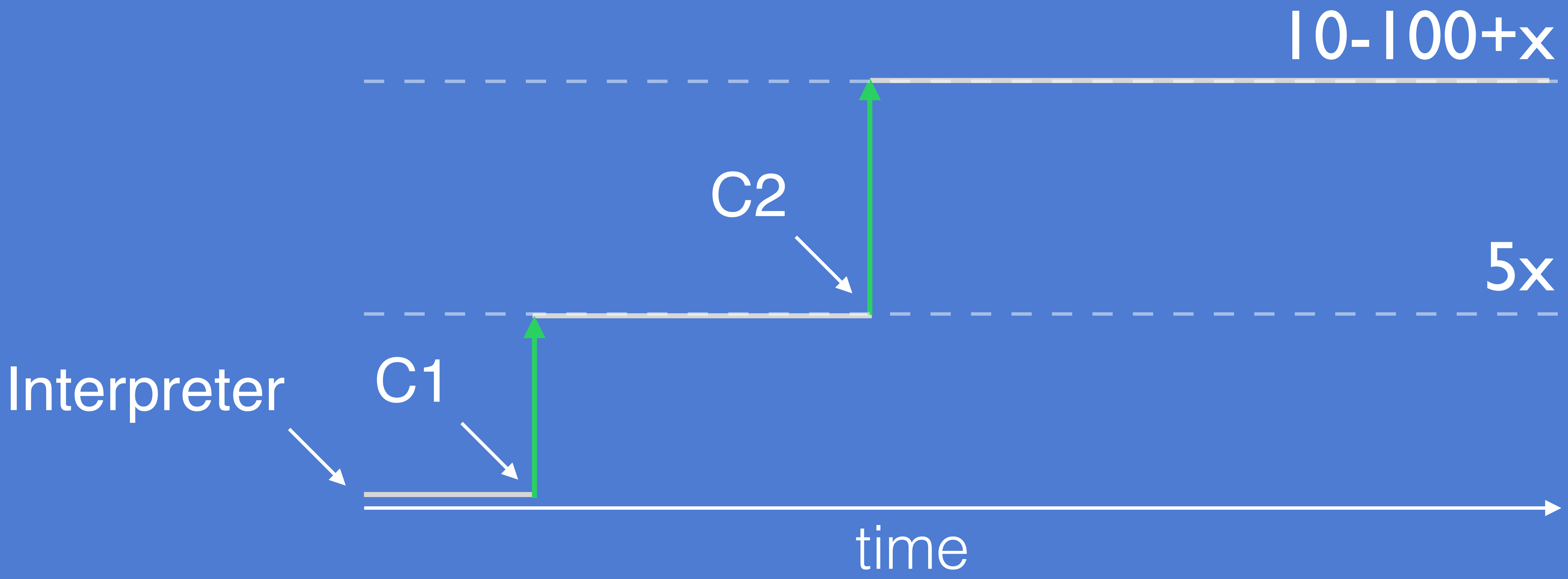
On-Stack Replacement



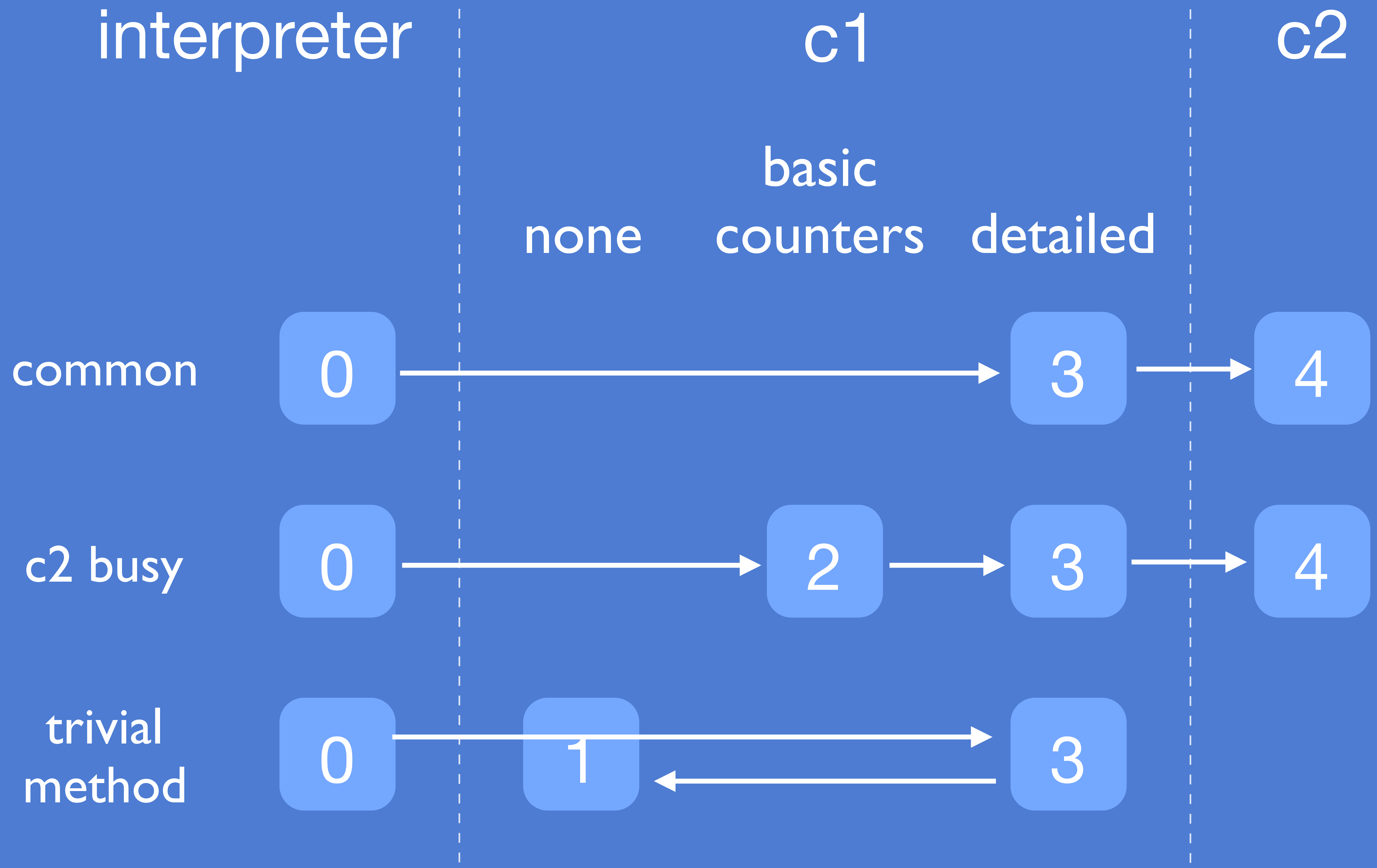
🟦 interpreter frame

🟦 compiled frame

Tiered Compilation

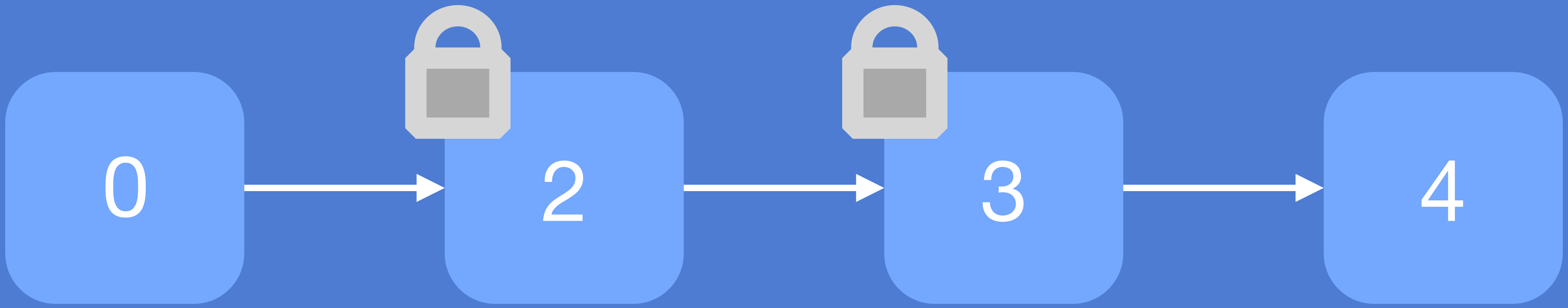


Tiered Compilation

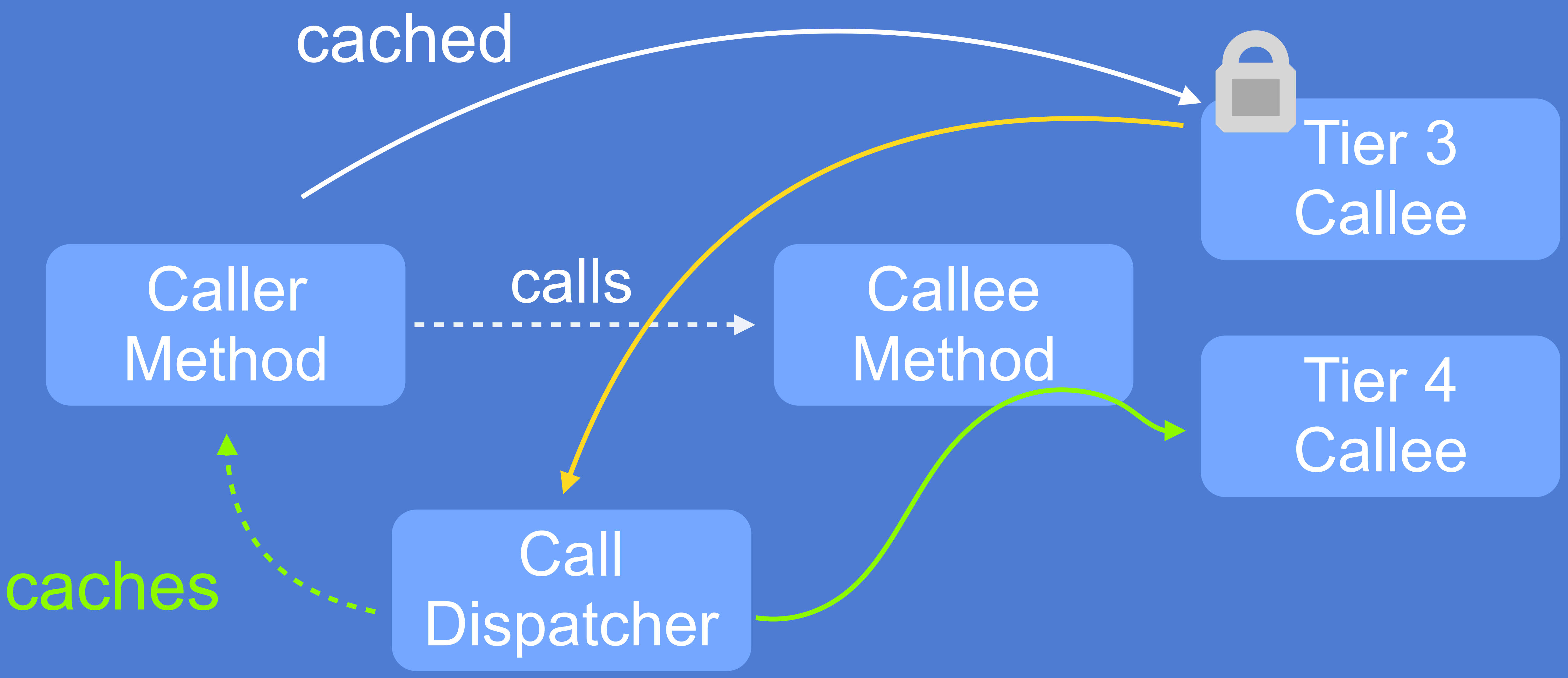


Made Not Entrant

```
12394   73 % 3 …Allocation::main @ -2 (78 bytes)   made not entrant
```



Lock Free Cache Invalidation



HotSpot's Job is to Find Hot Spots

Rules for Triggering the JIT Keep Changing

HotSpot JITs…

Hot Methods

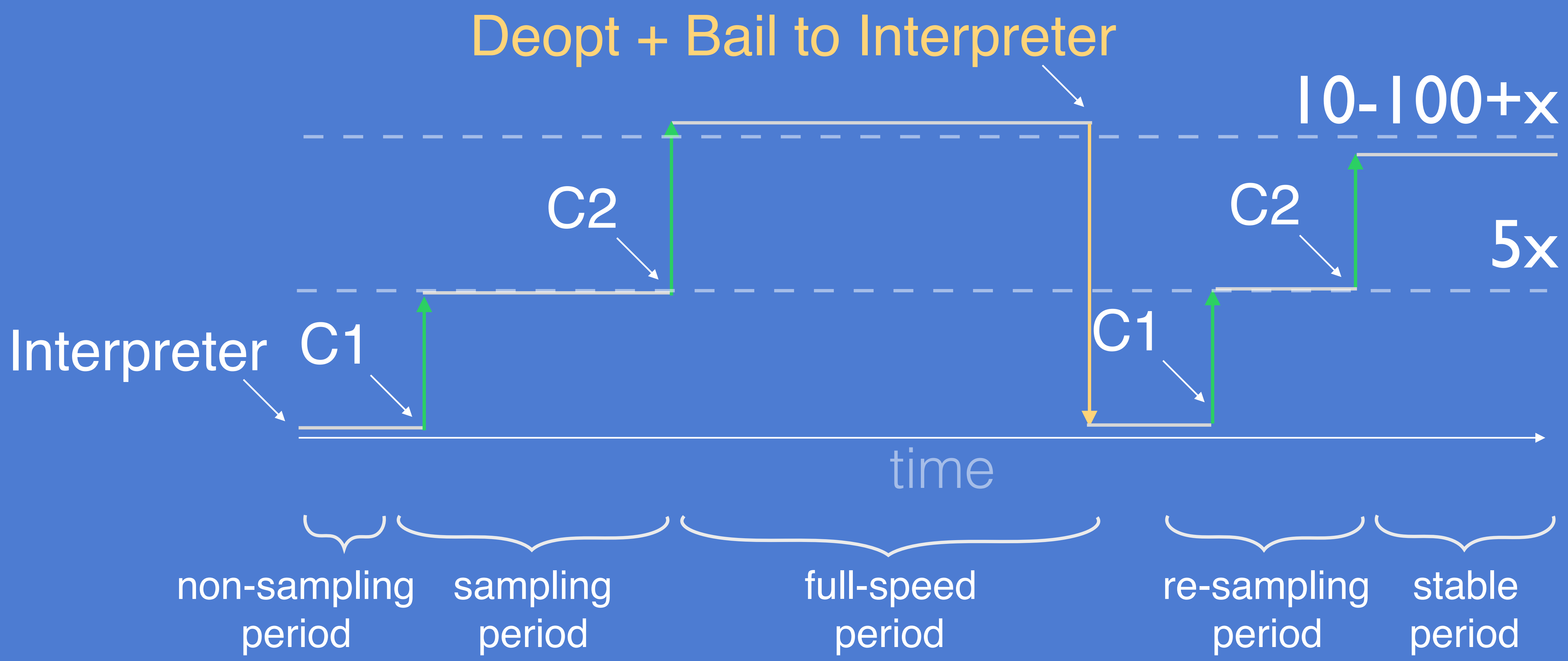
Hot Loops

Warm Methods with Warm Loops

Focus on Final Tier

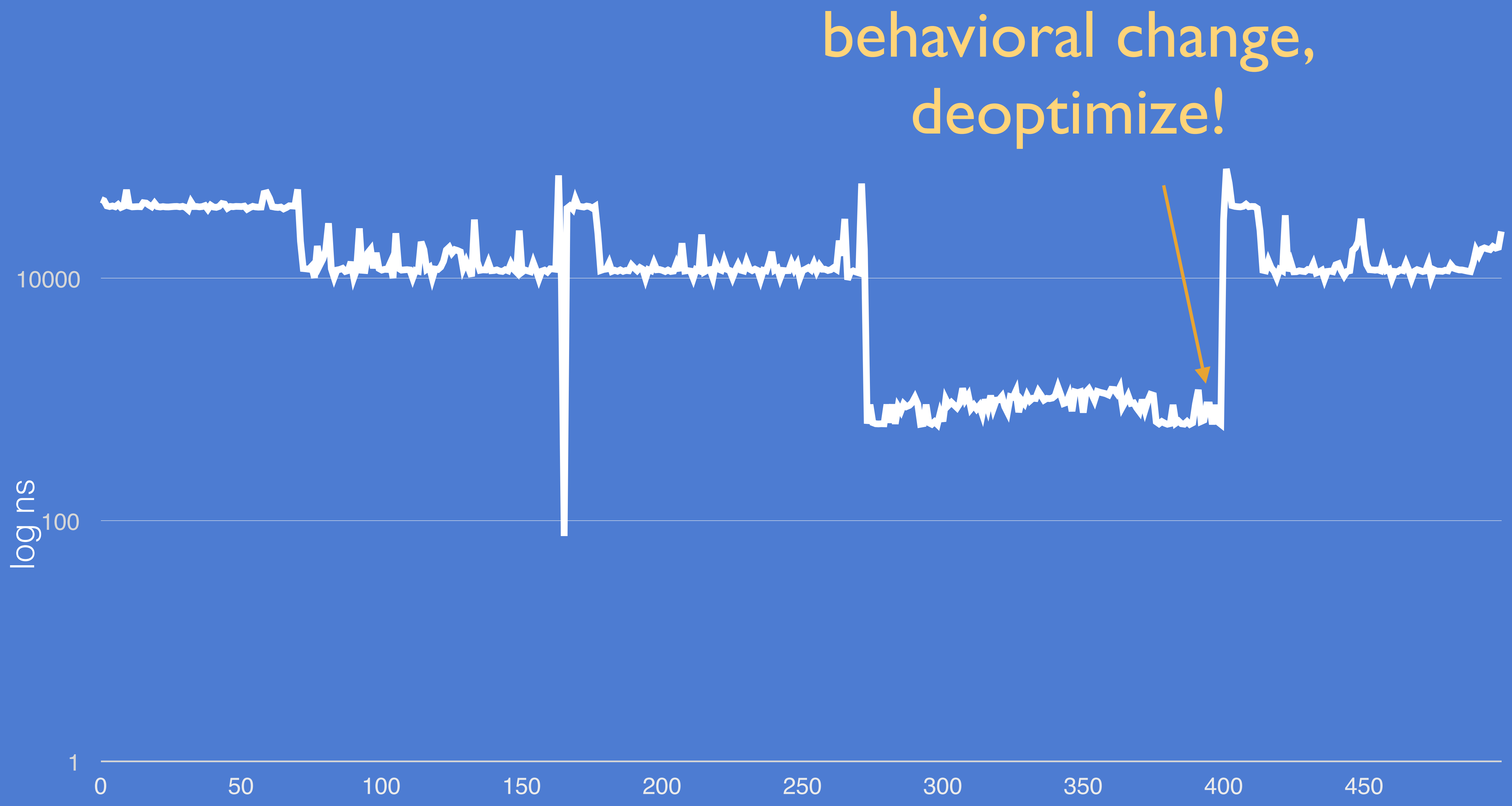
What Does ~~the JIT~~ C2 Do?

Profiling + Deoptimization



```
public class AllocationTrap {
  static final int CHUNK_SIZE = 1_000;

  public static void main(String[] args) {
    Object trap = null;

    for ( int i = 0; i < 500; ++i ) {
      long startTime = System.nanoTime();

      for ( int j = 0; j < CHUNK_SIZE; ++j ) {
        new Object();

        if ( trap != null ) {
          System.out.println("trap!");
          trap = null;
        }
      }
      if ( i == 400 ) trap = new Object();

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

Deoptimization



Returning to forEach

```
ArrayList<E>.forEach(Consumer<? super E> action) {
  for ( int i = 0; i < this.size; i++ ) {
    action.accept(this.elementData[i]);
  }
}
```

Something "Simpler"

"Simpler" Bound

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
 for ( int i = 0; i < this.elementData.length; i++ ) {
   action.accept(this.elementData[i]);
 }
}
```

Implied Safety Code

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
 if ( this == null ) throw new NPE();
 if ( this.elementData == null ) throw new NPE();

 for ( int i = 0; i < this.elementData.length; i++ ) {
   if ( this == null ) throw new NPE();
   if ( this.elementData == null ) throw new NPE();
   if ( i < 0 ) throw new AIOBE();
   if ( i >= this.elementData.length ) throw new AIOBE();

   if ( action == null ) throw new NPE();
   action.accept(this.elementData[i]);
 }
}
```

Null Check Elimination

this != null

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
 ~~if ( this == null ) throw new NPE();~~
 if ( this.elementData == null ) throw new NPE();

 for ( int i = 0; i < this.elementData.length; i++ ) {
   ~~if ( this == null ) throw new NPE();~~
   if ( this.elementData == null ) throw new NPE();
   if ( i < 0 ) throw new AIOBE();
   if ( i >= this.elementData.length ) throw new AIOBE();

   if ( action == null ) throw new NPE();
   action.accept(this.elementData[i]);
 }
}
```

Lower Bound Check Elimination

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();
    ~~if ( i < 0 ) throw new AIOBE();~~
    if ( i >= this.elementData.length ) throw new AIOBE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

i >= 0
i <= INT_MAX

Canonicalize Upper Bound

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();
    ~~if ( i >= this.elementData.length ) throw new AIOBE();~~
    if ( !(i < this.elementData.length) ) throw new AIOBE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

Canonicalize

Upper Bound Check Elimination

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();
    ~~if ( !(i < this.elementData.length) ) throw new AIOBE();~~

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

Common Sub-Expression*

* Really Global Value Numbering

Cached Length isn't Better!

Confuses the JIT

```
public class ArrayStream<E> interface Stream<E> {
  private final E[] elementData;
  private final int size;

  public ArrayStream(E… elements) {
    this.elementData = elements;
    this.size = elements.length;
  }

  forEach(Consumer<? super E> action) {
    if ( this.elementData == null ) throw new NPE();

    for ( int i = 0; i < this.size; i++ ) {
      if ( this.elementData == null ) throw new NPE();
      if ( i >= this.elementData.length ) throw new AIOBE();

      if ( action == null ) throw new NPE();
      action.accept(this.elementData[i]);
    }
  }
}
```

mismatch!

What Else?

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

What Else Can Be Done?

Null Pointer Checks

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

Required - cannot optimize further?

Assume non-null from above?

Required - cannot optimize further?

Implicit Null Check

Possible, but improbable

```
if ( this.elementData == null ) throw new NPE();
this.elementData.length;
```

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

$$\begin{tikzcd}
\text{deref value} \arrow[r, "\textbf{SEGV}"] & \text{signal handler} \arrow[r] & \text{throw NPE}
\end{tikzcd}$$

Three Nulls, You Deopt!

-XX:+PrintCompilation

```
    121     1       java.lang.String::hashCode (55 bytes)
    135     2       …NullCheck::hotMethod (6 bytes)
    136     3 % !   …NullCheck::main @ 5 (69 bytes)
tempting fate 0
tempting fate 1
tempting fate 2
   5144     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
```

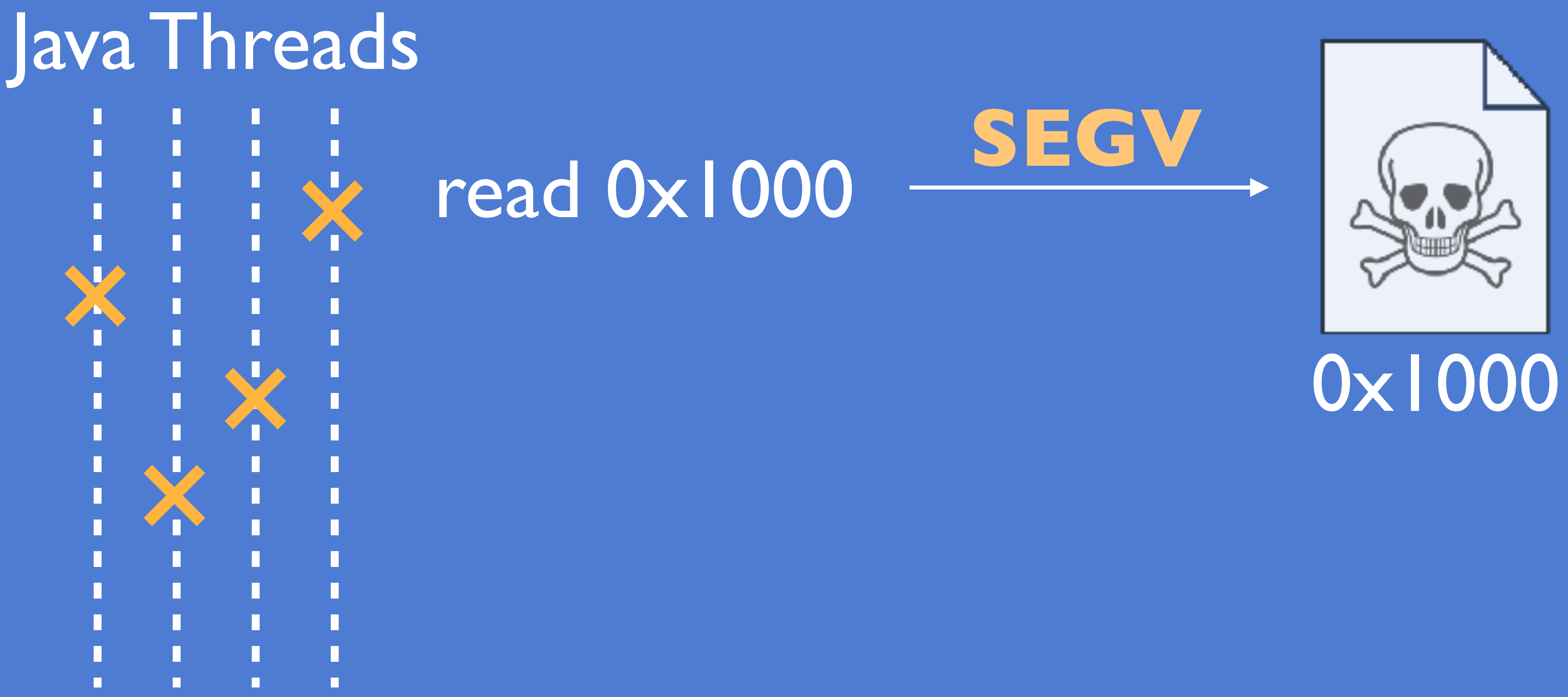
Stop the World

Need to Stop All Java Threads to...

Deoptimize

Lock Inflation / Deflation

Garbage Collect



Hot Exception Optimization

```
int caughtCount = 0;
Set<NullPointerException> nullPointerExceptions =
  new HashSet<>();

for ( Object object : objects ) {
  try {
    object.toString();
  } catch ( NullPointerException e ) {
    nullPointerExceptions.add( e );
    caughtCount += 1;
  }
}
```

```
Null Proportion: 0.100000 Caught:  10057  Unique:   2015
Null Proportion: 0.500000 Caught:  50096  Unique:   7191
Null Proportion: 0.900000 Caught:  89929  Unique:  11030
```

Hot Exceptions

```
int caughtCount = 0;
HashSet<NullPointerException> nullPointerExceptions =
  new HashSet<>();

for ( Object object : objects ) {
  try {
    object.toString();
  } catch ( NullPointerException e ) {
    boolean added = nullPointerExceptions.add(e);
    if ( !added ) e.printStackTrace();
    caughtCount += 1;
  }
}
```

java.lang.NullPointerException

No StackTrace???

Total Elimination is Better!

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

Assume non-null from above?

Common Sub-Expression

```
if ( this.elementData == null ) throw new NPE();

for ( … ) {
  if ( this.elementData == null ) throw new NPE();
}
```

Requires knowing that…

null doesn't change ←
this doesn't change ← easy

this.elementData doesn't change ← hard

The Other "Easy" One

Local Variable: action

CANNOT be changed by another thread or method.

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  for ( int i = 0; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[i]);
  }
}
```

Loop Peeling

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  // i=0 iteration
  if ( 0 < this.elementData.length ) {
    if ( this.elementData == null ) throw NPE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[0]);
  }

  // rest of the iterations
  for ( int i = 1; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();

    ~~if ( action == null ) throw new NPE();~~
    action.accept(this.elementData[i]);
  }
}
```

Back to this.elementData

CAN be changed by another thread.

Compiler Doesn’t Care!

Really?

No Synchronization Action -
Single Threaded Semantics

Loop Invariant Hoisting?

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  E[] elementData = this.elementData;
  int len = elementData.length;

  if ( elementData == null ) throw new NPE();

  // i=0 iteration
  if ( 0 < len ) {
    ~~if ( elementData == null ) throw NPE();~~

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[0]);
  }

  // rest of the iterations
  for ( int i = 1; i < elementData.length; i++ ) {
    ~~if ( elementData == null ) throw new NPE();~~

    action.accept(elementData[i]);
  }
}
```

Terrifying!

Producer Thread

```
sharedData = …;
sharedDone = true;
```

Consumer Thread

```
while ( !sharedDone );
print(sharedData);
```

$\downarrow$ Assume sharedData is loop invariant!

```
localDone = sharedDone;
while ( !localDone );
print(sharedData);
```

Not So Fast

Single Threaded Side Effects

```
ArrayStream<E>.forEach(Consumer<? super E> action) {
  if ( this.elementData == null ) throw new NPE();

  // i=0 iteration
  if ( 0 < this.elementData.length ) {
    if ( this.elementData == null ) throw NPE();

    if ( action == null ) throw new NPE();
    action.accept(this.elementData[0]);
  }

  // rest of the iterations
  for ( int i = 1; i < this.elementData.length; i++ ) {
    if ( this.elementData == null ) throw new NPE();

    action.accept(this.elementData[i]);
  }
}
```

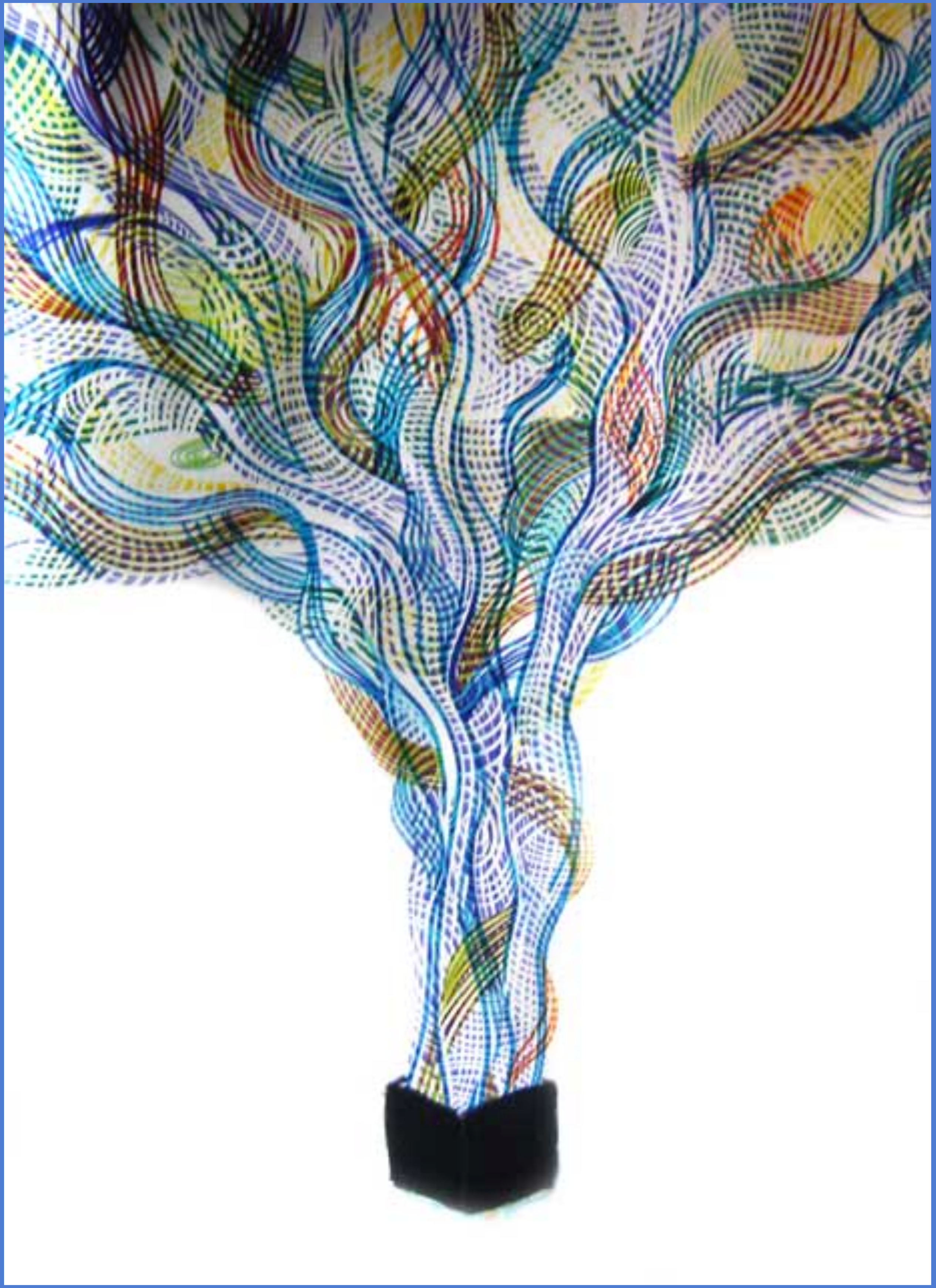
Can this.elementData change? YES

ArrayStream NOT ArrayList

this.elementData is final!

Doesn’t matter - reflection!

A Call is a “Black Box” which may contain “Evils”!



Inter-procedural Analysis

Prove No
Side Effects / Evils

Inter-procedural Analysis
=
Inlining

Inlining
Copy & Paste Callee Into Caller

$$\begin{tikzcd} \texttt{System.out.println(square(9));} \arrow[d, "\text{inline}"] \\ \texttt{System.out.println(9 * 9);} \arrow[d, "\text{constant folding}"] \\ \texttt{System.out.println(81);} \end{tikzcd}$$

Start with a Static Call

JMH: Java Measurement Harness

```
@Benchmark
public void cstyle() {
  for ( int i = 0; i < this.elementData.length; ++i ) {
    consume(this.elementData[i]);
  }
}

static int sum = 0;

@CompilerControl(CompilerControl.Mode.INLINE | DONT_INLINE)
static consume(int x) {
  sum += x;
}
```

Inlined

```
Benchmark                  Mode  Cnt    Score   Error  Units
LoopInvariant.cstyle       avgt   10  296.759 ± 3.619  ns/op
LoopInvariant.enhanced     avgt   10  294.379 ± 7.461  ns/op
LoopInvariant.hoisted      avgt   10  292.491 ± 7.623  ns/op
```

Not Inlined

```
Benchmark                 Mode  Cnt     Score    Error  Units
LoopInvariant.cstyle      avgt   10  2922.285 ± 48.199  ns/op
LoopInvariant.enhanced    avgt   10  2301.793 ± 37.154  ns/op
LoopInvariant.hoisted     avgt   10  2325.981 ± 39.935  ns/op
```

benchmarks.LoopInvariant

Not Just "Sugar"

```
for ( int x: this.elementData ) {
  consume(x);
}
```

$\Big\downarrow$ javac

```
E[] elementData = this.elementData;
for ( int i = 0; i < elementData.length; ++i ) {
  int x = elementData[i];
  consume(x);
}
```

More Terrifying!

Producer Thread

```
sharedData = …;
sharedDone = true;
```

Consumer Thread

```
while ( !sharedDone ) {
  fn();
}
print(sharedData);
```

$\downarrow$ Assume sharedData is loop invariant!

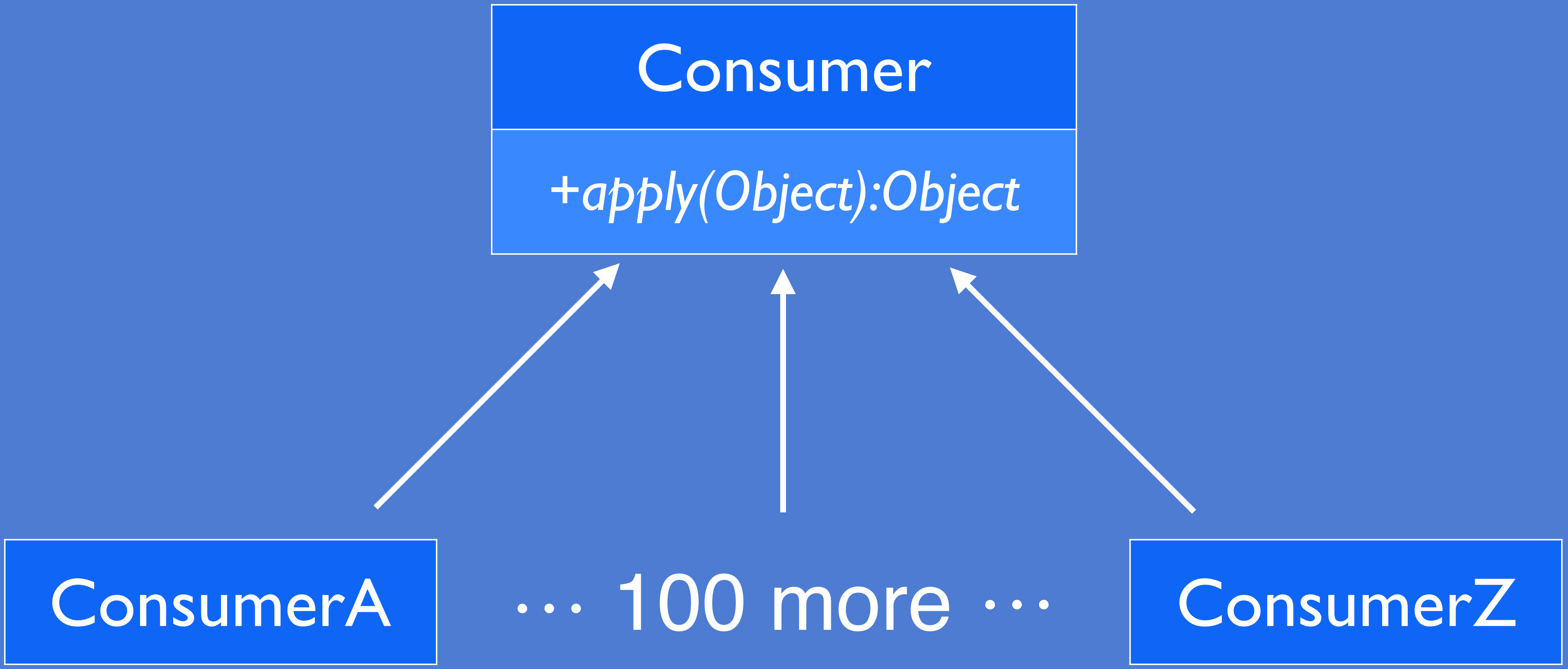
```
localDone = sharedDone;
while ( !localDone ) {
  // inlined fn() …
}
print(sharedData);
```

Inlining Numbers to Remember...

-XX:+PrintFlagsFinal

MaxTrivialSize	6
MaxInlineSize	35
FreqInlineSize	325
MaxInlineLevel	9
MaxRecursiveInlineLevel	1
MinInliningThreshold	250
Tier1MaxInlineSize	8
Tier1FreqInlineSize	35

What About Dynamic Calls?



http://shipilev.net/blog/2015/black-magic-method-dispatch/

Java is a *Dynamic* Language!

Dynamically Loaded

Dynamically Linked

Lazy Initialized

Typically Dynamically Dispatched

Even Dynamic Code Gen in JDK

Unloaded Class?!?

Don't Know…

Fields in Class

Methods in Class

Parent Class

Interfaces

Anything?

Give Up!

```
uncommon_trap(:unloaded)
```

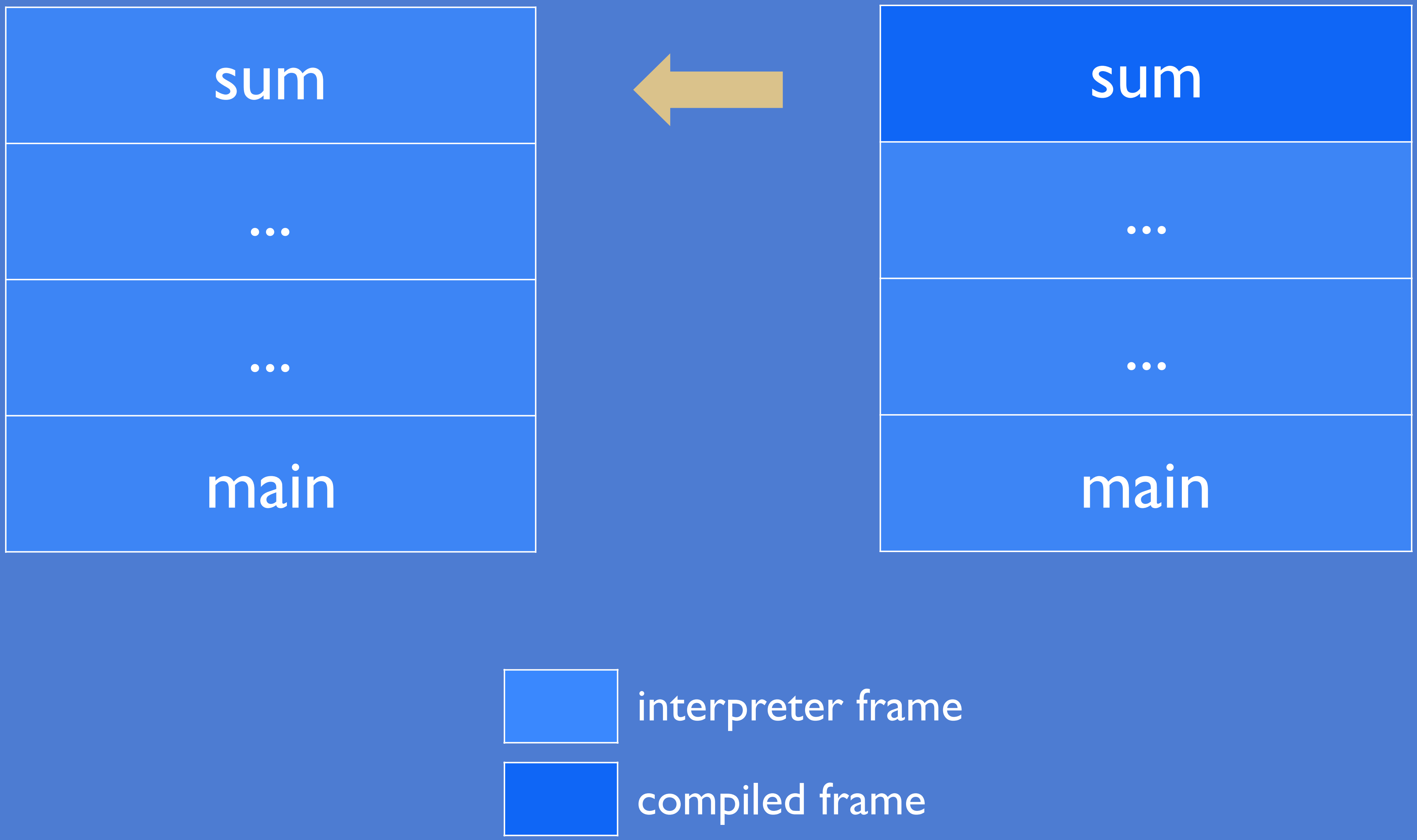
What? I give up!

Compile + Deopt Storm

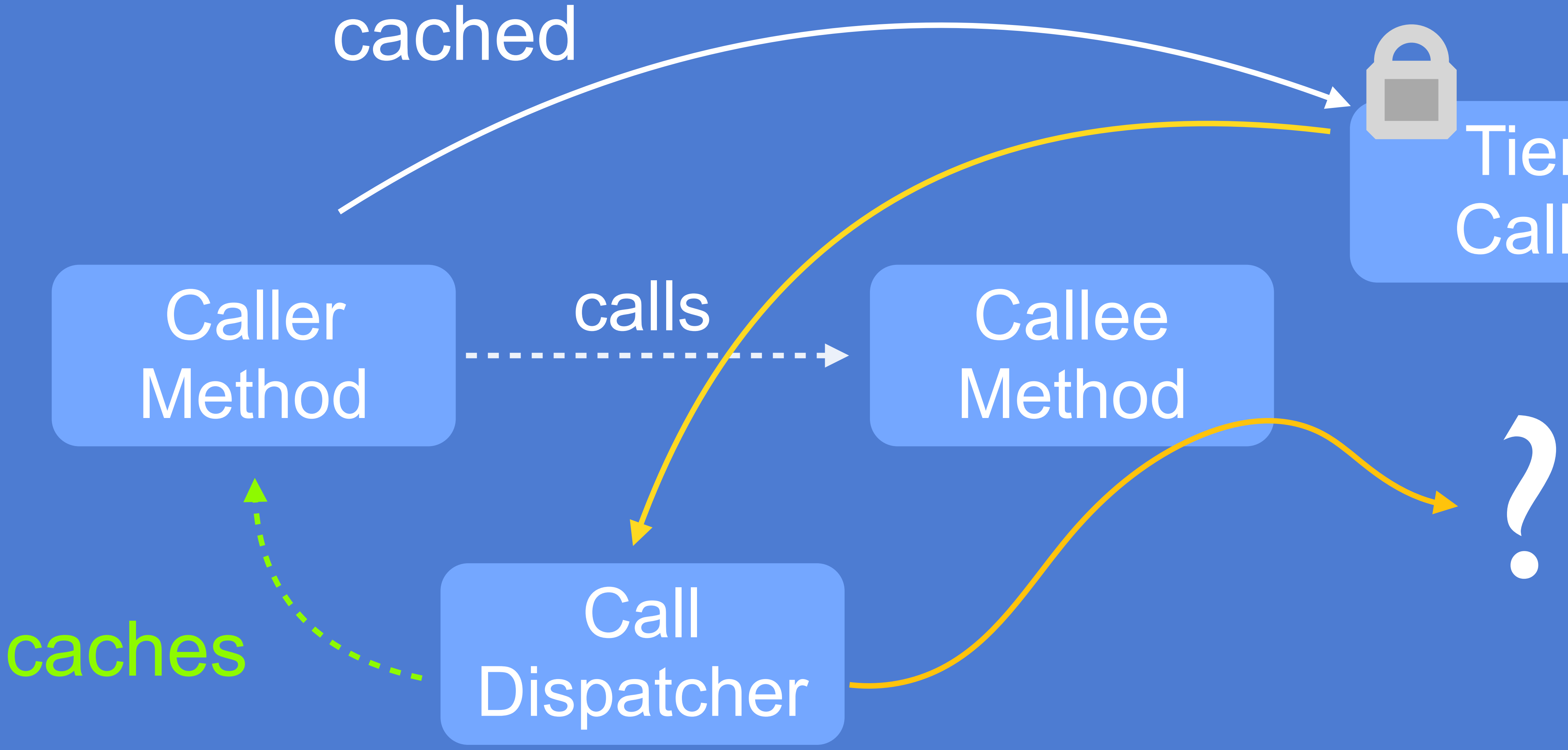
```java
public class UnloadedForever {
  public static void main(String[] args) {
    for ( int i = 0; i < 100_000; ++i ) {
      try {
        factory();
      } catch ( Throwable t ) {
        // ignore
      }
    }
  }

  static DoesNotExist factory() {
    return new DoesNotExist();
  }
}
```

Compile + Deopt Storm

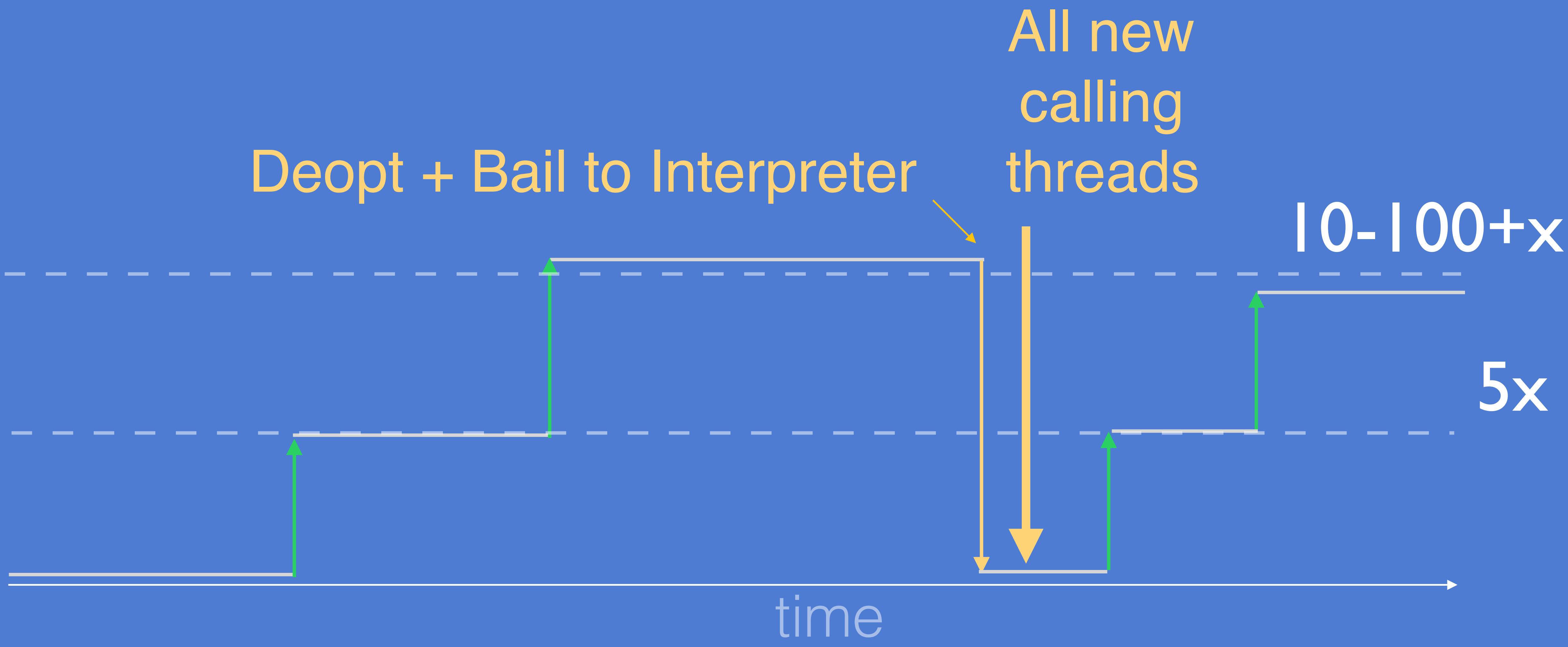
On-Stack Replacement in Reverse



Lock Free — but Hurts



Lock Free — but *Really* Hurts



Back to Dynamic/Virtual Calls
4 Strategies to “Devirtualize”

Static Analysis

Class Hierarchy Analysis

TypeProfile

Unique Concrete Method

Monomorphic

```
public class Monomorphic {
  public static void main(String[] args)
    throws InterruptedException
  {
    Func func = new Square();

    for ( int i = 0; i < 20_000; ++i ) {
      apply(func, i);
    }

    Thread.sleep(5_000);
  }

  static double apply(Func func, int x) {
    return func.apply(x);
  }
}
```

Static Analysis

```
public class Monomorphic {
  public static void main(String[] args)
    throws InterruptedException
  {
    Func func = new Square();

    for ( int i = 0; i < 20_000; ++i ) {
      apply(func, i);
    }

    Thread.sleep(5_000);
  }

  static double apply(Func func, int x) {
    return func.apply(x);
  }
}
```

Monomorphic

-XX:+PrintCompilation -XX:-BackgroundCompilation
-XX:+UnlockDiagnosticVMOptions -XX:+PrintInlining

```
217    1     java.lang.String::hashCode (55 bytes)
234    3     example03.support.Square::apply (4 bytes)
234    4 %   example03a.Monomorphic::main @ 13 (30 bytes)
  @ 15   example03a.Monomorphic::apply (7 bytes)   inline (hot)
    @ 3   example03.support.Square::apply (4 bytes)   inline (hot)
234    2     example03a.Monomorphic::apply (7 bytes)
 @ 3   example03.support.Square::apply (4 bytes)   inline (hot)
```

Potential for Deopt Storm

```
public class ChaStorm {
  public static void main(String[] args) throws… {
    Func func = new Square();

    for ( int i = 0; i < 10_000; ++i ) {
      apply1(func, i);
      …
      apply8(func, i);
    }

    System.out.println("Waiting for compiler...");
    Thread.sleep(5_000);

    System.out.println("Deoptimize...");
    System.out.println(Sqrt.class);

    Thread.sleep(5_000);
  }
}
```

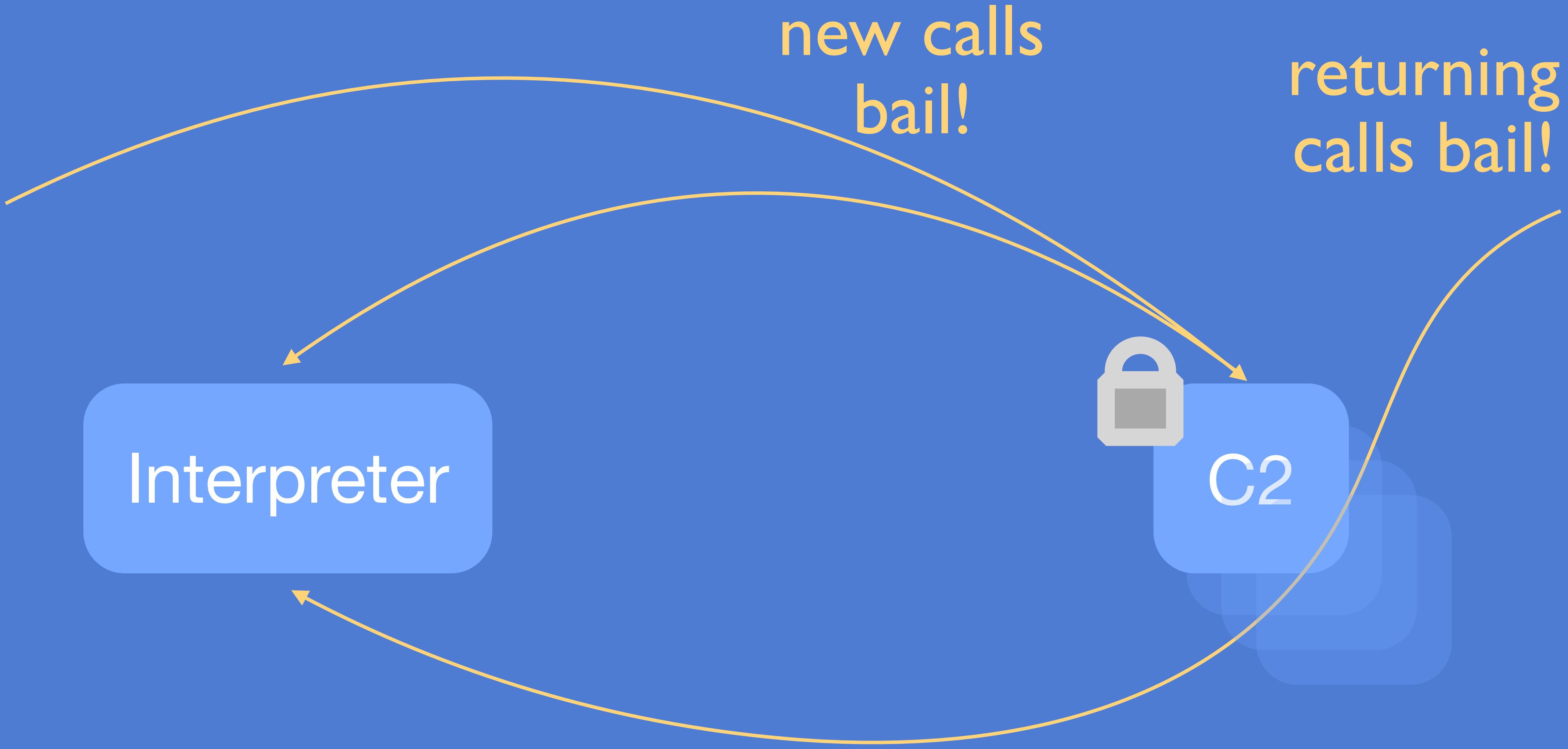
Potential for Deopt Storm

-XX:+PrintCompilation
-XX+PrintSafepointStatistics
-XX:PrintSafepointStatisticsCount=1

```
    152    1    java.lang.String::hashCode (55 bytes)
    166    2    example04.support.Square::apply (4 bytes)
    173    3    example04b.ChaStorm::apply1 (7 bytes)
    173    4    example04b.ChaStorm::apply2 (7 bytes)
Waiting for compiler...
    174    5    example04b.ChaStorm::apply3 (7 bytes)    …
    174    9    example04b.ChaStorm::apply7 (7 bytes)
    174   10    example04b.ChaStorm::apply8 (7 bytes)
Deoptimize...
   5176    9    example04b.ChaStorm::apply7 (7 bytes)   made not entrant
   5176    8    example04b.ChaStorm::apply6 (7 bytes)   made not entrant
   …
   5176    4    example04b.ChaStorm::apply2 (7 bytes)   made not entrant
   5176    3    example04b.ChaStorm::apply1 (7 bytes)   made not entrant
   5176   10    example04b.ChaStorm::apply8 (7 bytes)   made not entrant
class example04.support.Sqrt
 vmop                        [threads: total initially_running wait_to_block]    …
5.096: Deoptimize                   [      7         0              0     ]    …
```

http://blog.ragozin.info/2012/10/safepoints-in-hotspot-jvm.html

NOT Lock Free — and *Really, Really* Hurts



# No IHA

TypeProfile

Interpreter & C1 Gather Data

Track Types used at Each Call Site

-XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation

```
<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'/>
```

Bimorphic

```
Func func = …
double result = func.apply(20);
```

```
if ( func.getClass().equals(Square.class) ) {
  …
} else {
  uncommon_trap(class_check);
}
```

```
if ( func.getClass().equals(Square.class) ) {
  …
} else if ( func.getClass().equals(AlsoSquare.class) ) {
  …
} else {
  uncommon_trap(bimorphic);
}
```

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

Very Effective

Call-site specific

Works for 90-95% of call sites

Very few call sites are “megamorphic”

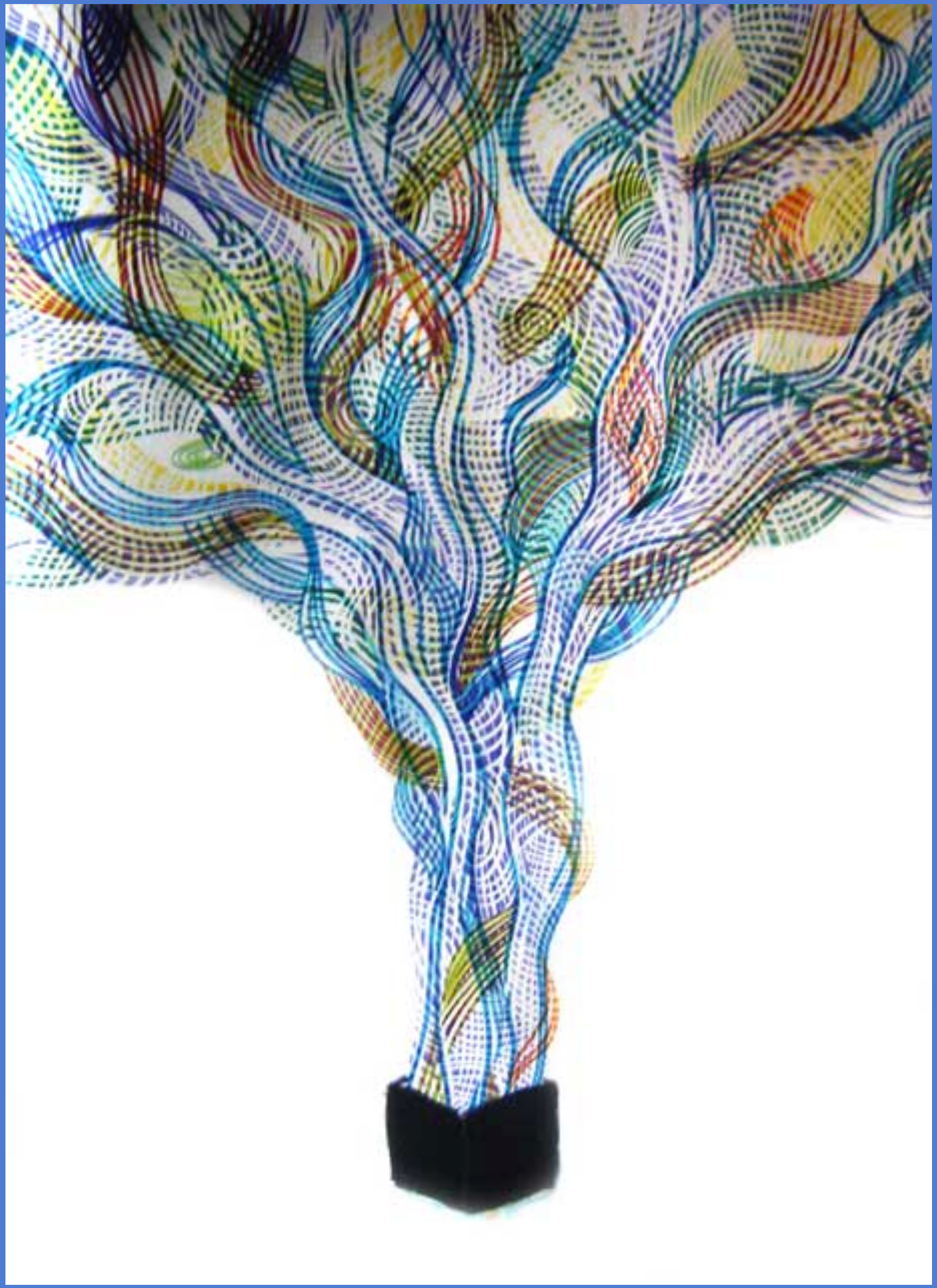
Slightly more overhead than
no check (3-5ns)

Why Trap?!?

```
if ( func.getClass().equals(Square.class) ) {
  …
} else {
  uncommon_trap(class_check);
}
```

↑

Why not func.apply?



Solved? NO

size could change!

```
ArrayList<E>.forEach(Consumer<? super E> action) {
 for ( int i = 0; i < this.size; i++ ) {
    action.accept(this.elementData[i]);
  }
}
```

megamorphic,
no (incomplete)
inlining.

elementData
could change!

Unless, Done Manually…

```
ArrayList<E>.forEach(Consumer<? super E> action) {
  Objects.requireNonNull(action);
  final int expectedModCount = modCount;
  final E[] elementData = (E[]) this.elementData;
  final int size = this.size;

  for (int i=0;
    modCount == expectedModCount && i < size;
    i++)
  {
    action.accept(elementData[i]);
  }
  if (modCount != expectedModCount) throw new CME();
}
```

Goal Accomplished? NO

More loop optimizations…

Loop Unrolling
Loop Unswitching
Vectorization

Interactions with Garbage Collector

Handling this.size with an uncommon trap

JIT (and All Compilers) Are Just Complex Pattern Matchers.

Like	Don’t Like
“Normal” Code	“Weird” Code
Small Methods	Big Methods
Immutability	Mutability
Local Variables	Native Methods*

* except for intrinsics: arraycopy, tan, …

VM Developer Blogs

PSYCHOSOMATIC, LOBOTOMY, SAW

Nitsan Wakart

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

ORACLE®

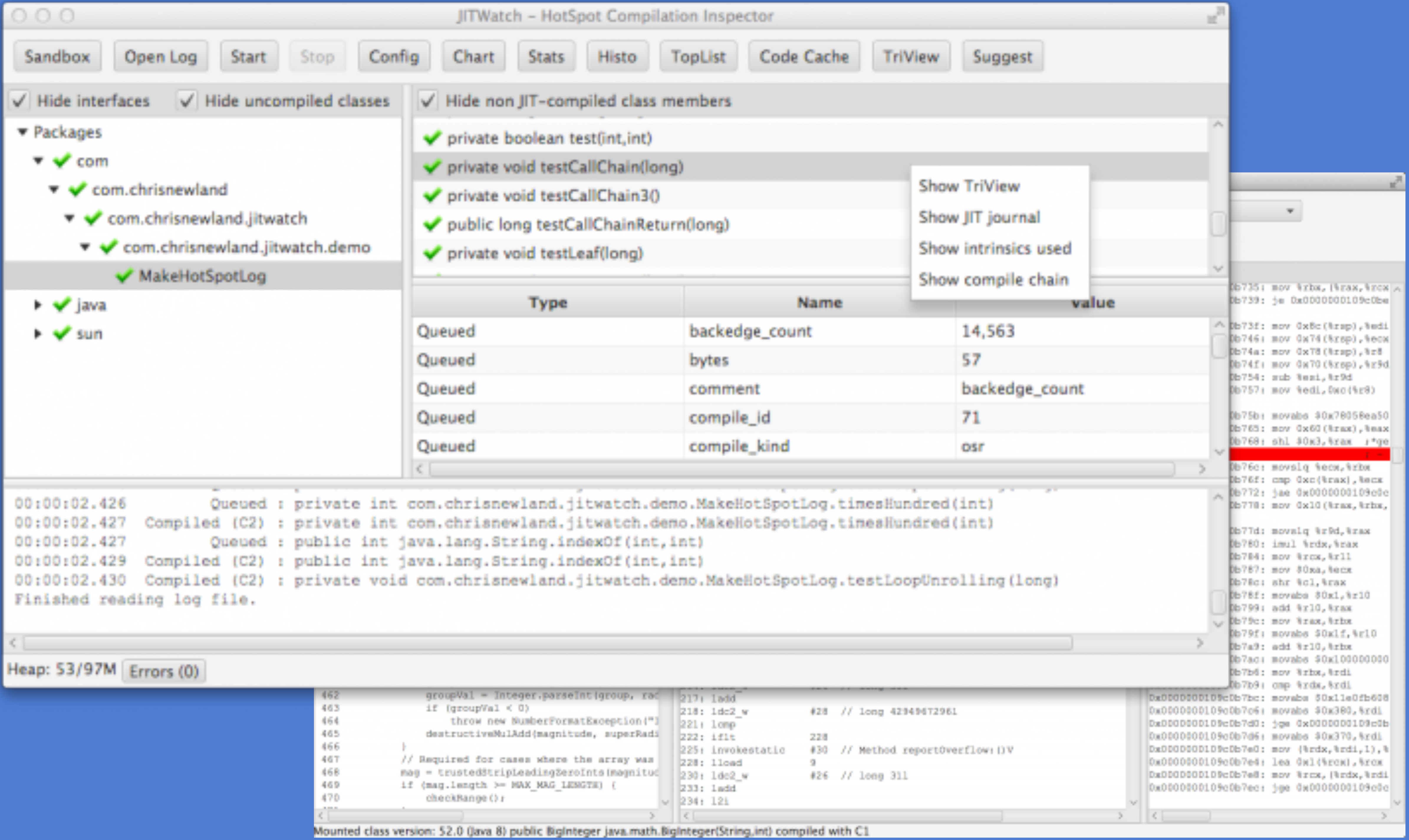
Aleksey Shipilëv

http://shipilev.net/

Igor Veresov

https://twitter.com/maddocig

JITWatch


JITWatch - HotSpot Compilation Inspector
Sandbox
Open Log
Start
Stop
Config
Chart
Stats
Histo
TopList
Code Cache
TriView
Suggest
Hide interfaces
Hide uncompiled classes
Hide non JIT-compiled class members
Packages
com
com.chrisnewland
com.chrisnewland.jitwatch
com.chrisnewland.jitwatch.demo
MakeHotSpotLog
java
sun
private boolean test(int,int)
private void testCallChain(long)
private void testCallChain3()
public long testCallChainReturn(long)
private void testLeaf(long)
Show TriView
Show JIT journal
Show intrinsics used
Show compile chain
Heap: 53/97M
Errors (0)

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.
```

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

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

Shameless Self-Promotion

O'REILLY®

Optimizing Java

Douglas Q. Hawkins

http://shop.oreilly.com/product/0636920043560.do

Questions?

Douglas Q. Hawkins
@dougqh
VM Engineer

AZUL SYSTEMS®