IS YOUR **PROFILER** SPEAKING THE SAME LANGUAGE AS YOU?

SIMON MAPLE
@SJMAPLE

ZEROTURNAROUND

JavaOne™

ABOUT **ME**

SIMON MAPLE

@SJMAPLE

ZEROTURNAROUND

ABOUT **ME**

SIMON MAPLE

@SJMAPLE

ZEROTURNAROUND

VIRTUAL JUG FOUNDER

LONDON JUG CO-ORGANISER

JAVA CHAMPION

JAVAONE ROCKSTAR SPEAKER

REBELLABS AUTHOR

MONITORING TOOLS

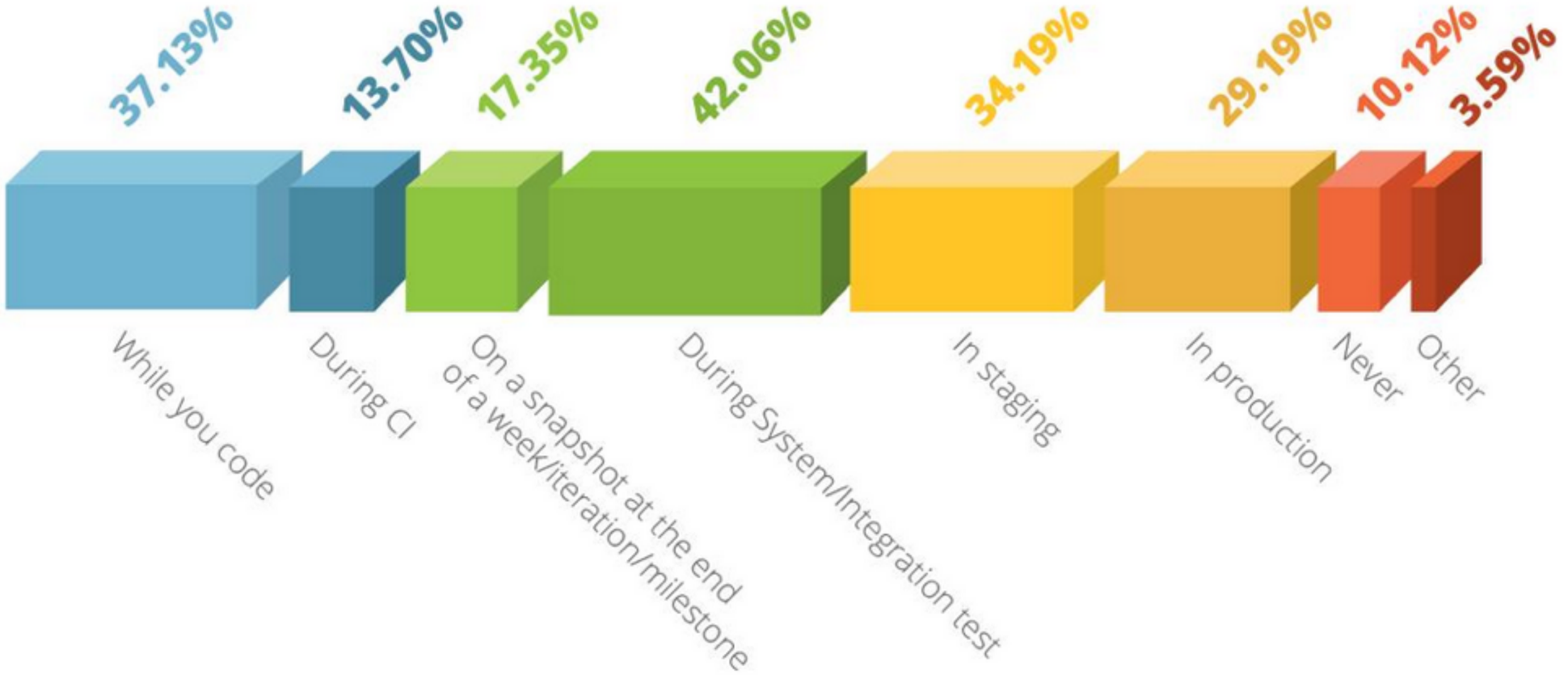
PROFILERS

TESTING TOOLS


THE DEVELOPER'S GUIDE TO UNDERSTANDING PERFORMANCE PROBLEMS.
DISCOVERING APPDYNAMICS, NEWRELIC, JAVA MISSION CONTROL, YOURKIT, JPROFILER, XREBEL, JMETER
And many other marvelous tools
dynatrace


REBELLABS
by ZEROTURNAROUND
RebelLabs' Java Performance Survey 2015
Hey, you made it! We're really happy you can spend ~3 minutes (verified by devs) to tell us about your profiler and performance tools and technologies you use, enjoy and get results from.
For each completed survey, we're donating 50 cents to a charity that provide support dogs for children with disabilities! See? Now you HAVE TO complete it and share with friends! ;-)
Get started!
press ENTER

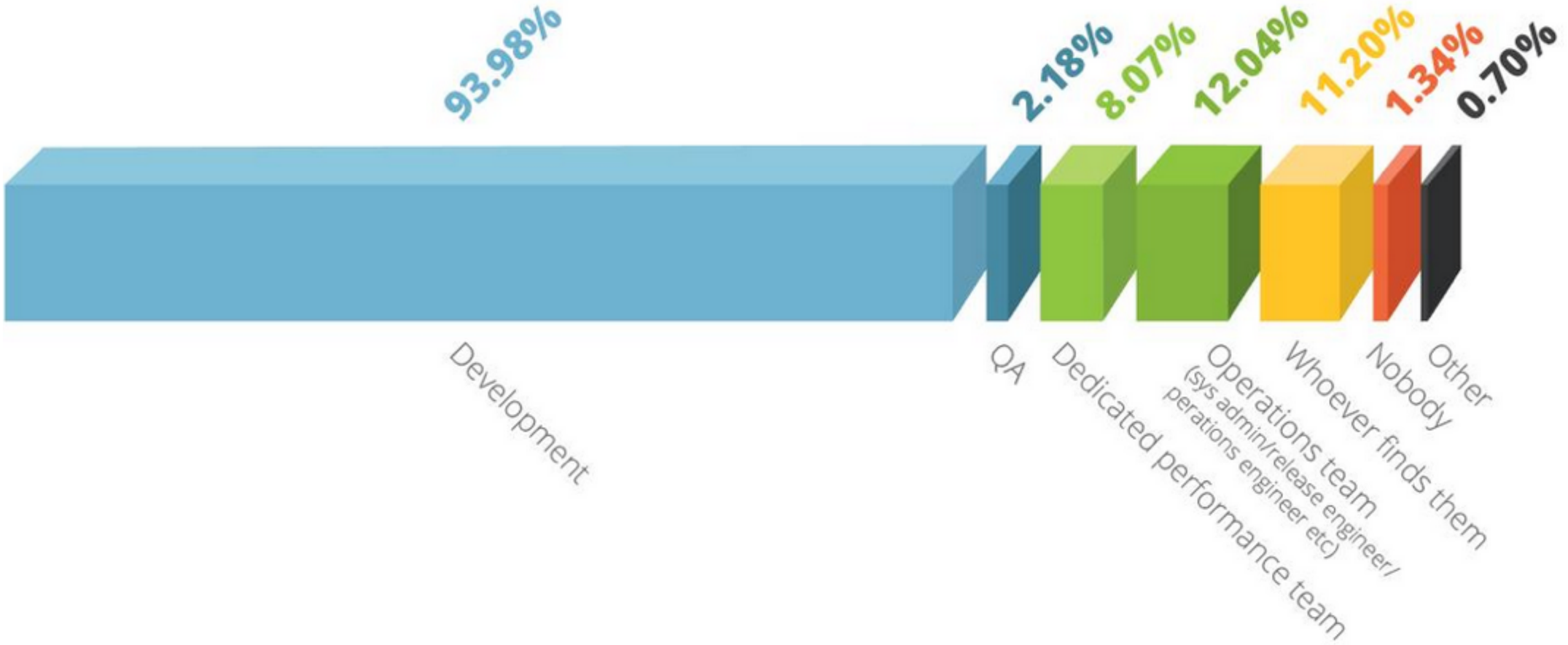
At what stage do you perform profiling and performance tuning on your applications?



REBEL LABS by ZERO TURNAROUND

source: http://zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/

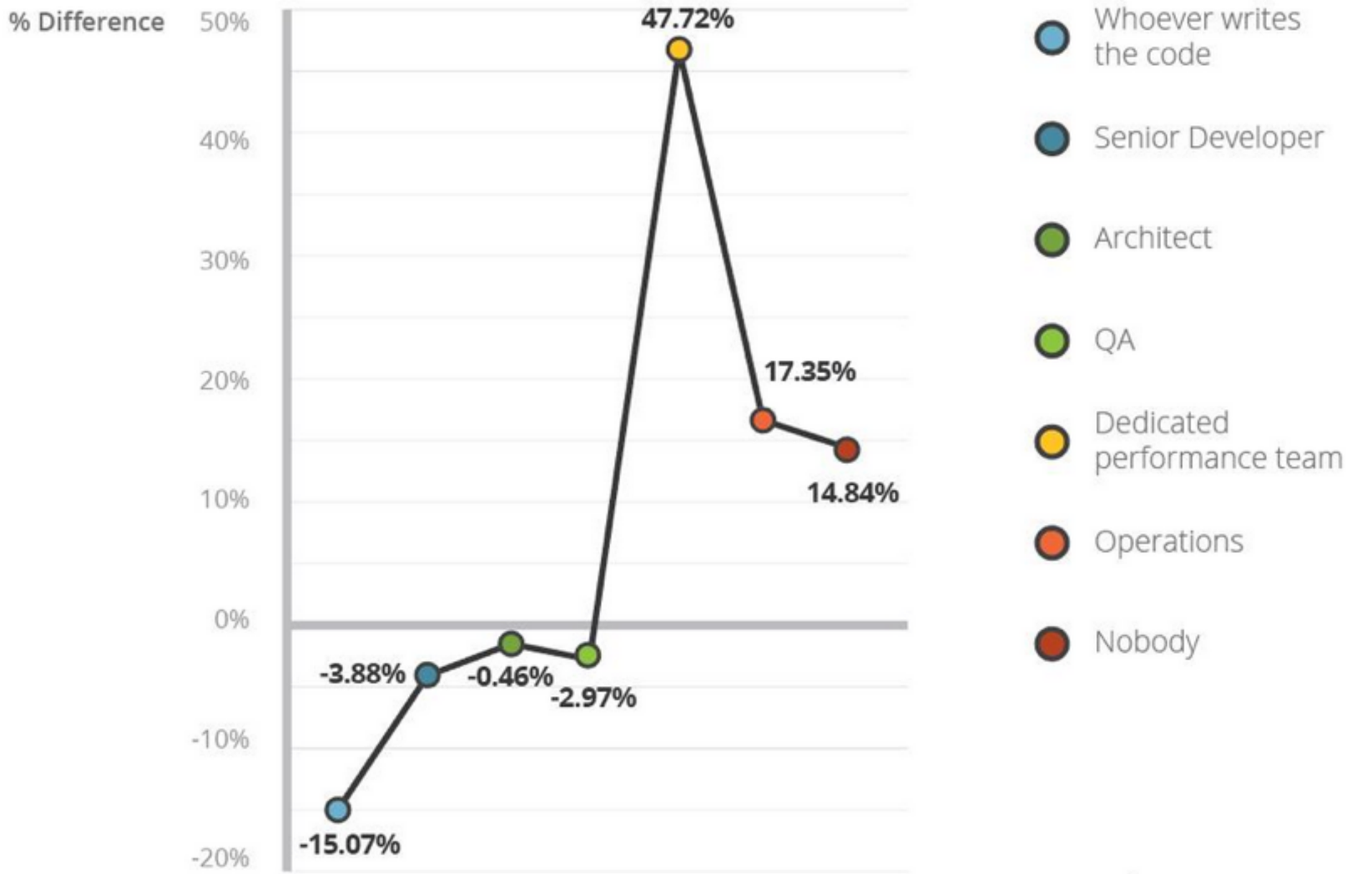
Who fixes the performance problems?



REBEL LABS by ZEROTURNAROUND

source: http://zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/

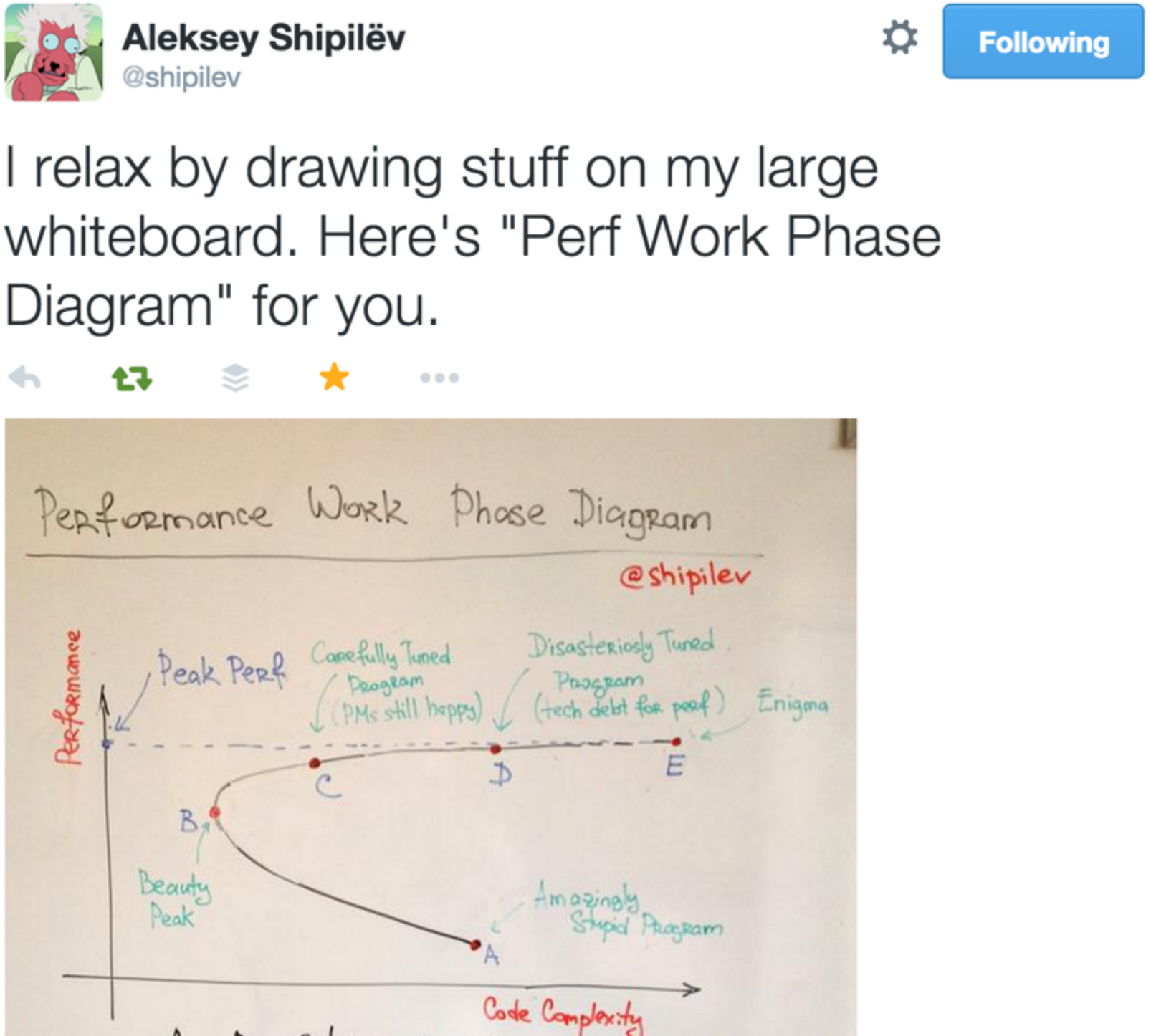
When you find an issue, how long does it take,
in days, to diagnose, fix and test on average?



REBELLABS by ZEROTURNAROUND

source: http://zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/

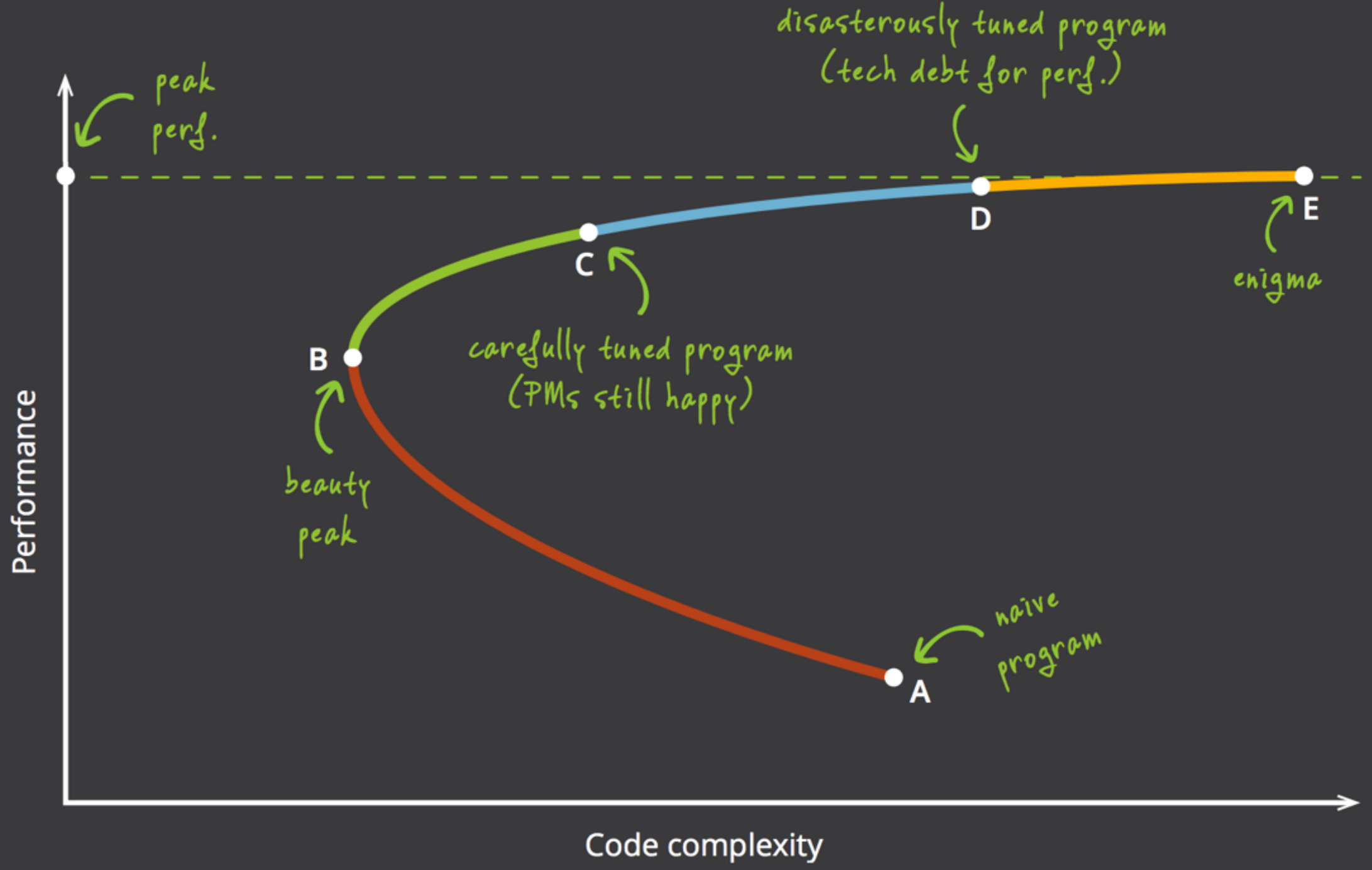
APPLICATION PERFORMANCE



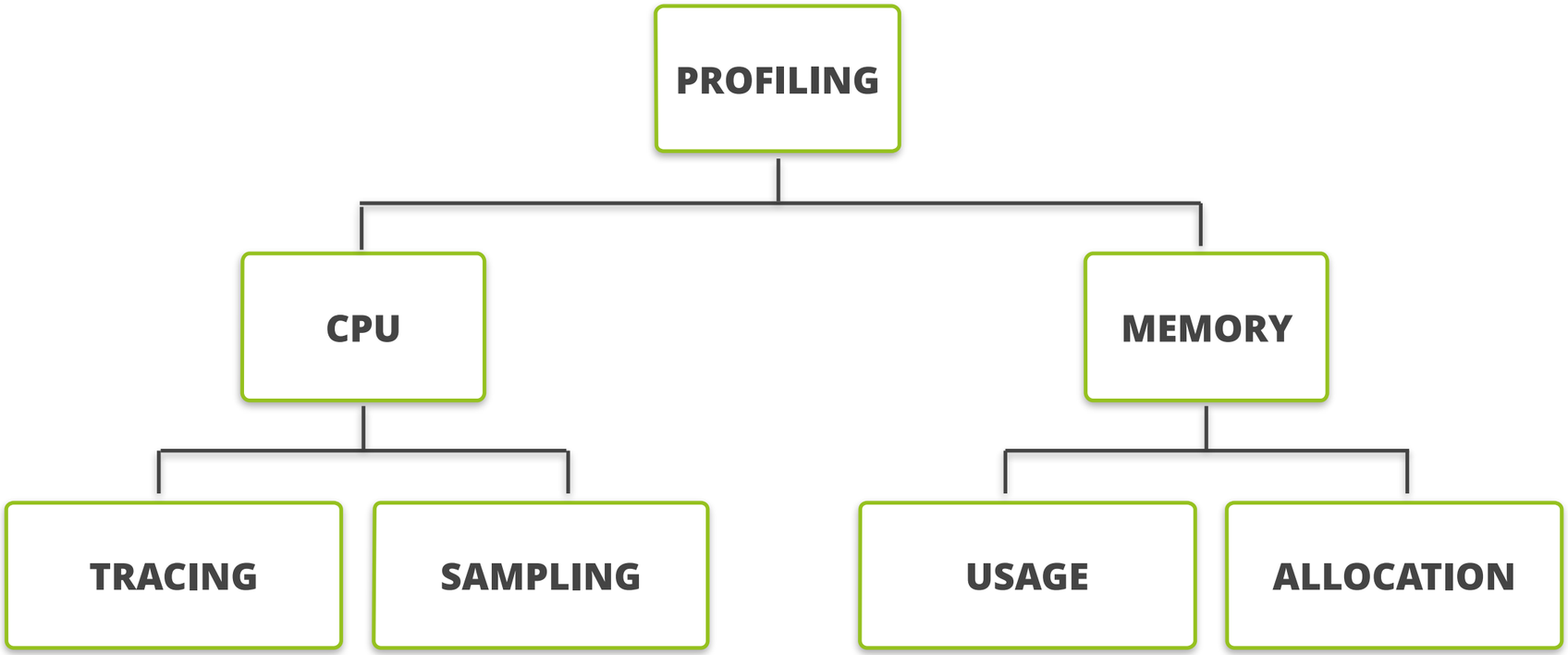
Aleksey Shipilëv
@shipilev

Following

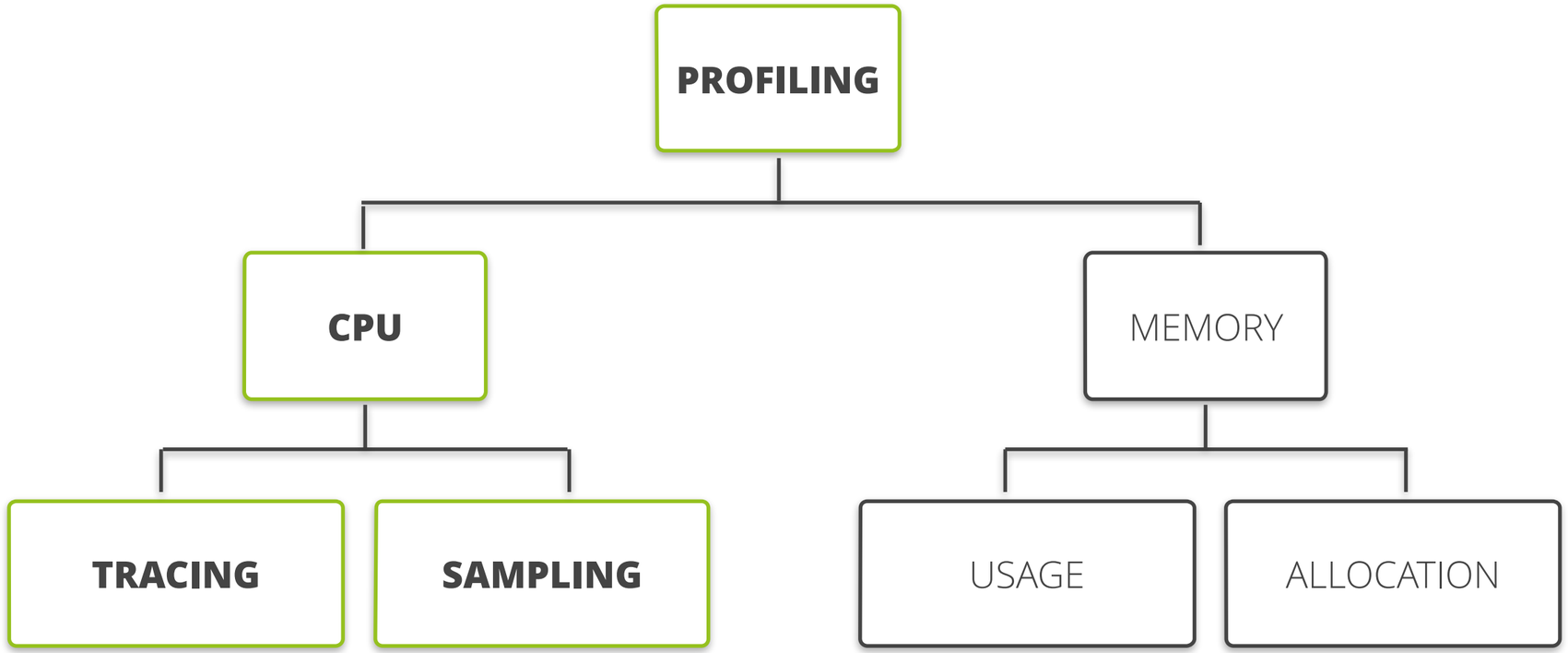
I relax by drawing stuff on my large whiteboard. Here's "Perf Work Phase Diagram" for you.


disasterously tuned program
(tech debt for perf.)
peak
perf.
D
E
C
enigma
carefully tuned program
(PMs still happy)
B
beauty
peak
naive
program
A
Performance
Code complexity

WHAT DOES PROFILING COVER?



WHAT DOES **PROFILING** COVER?



WHAT IS A **SAMPLE**?

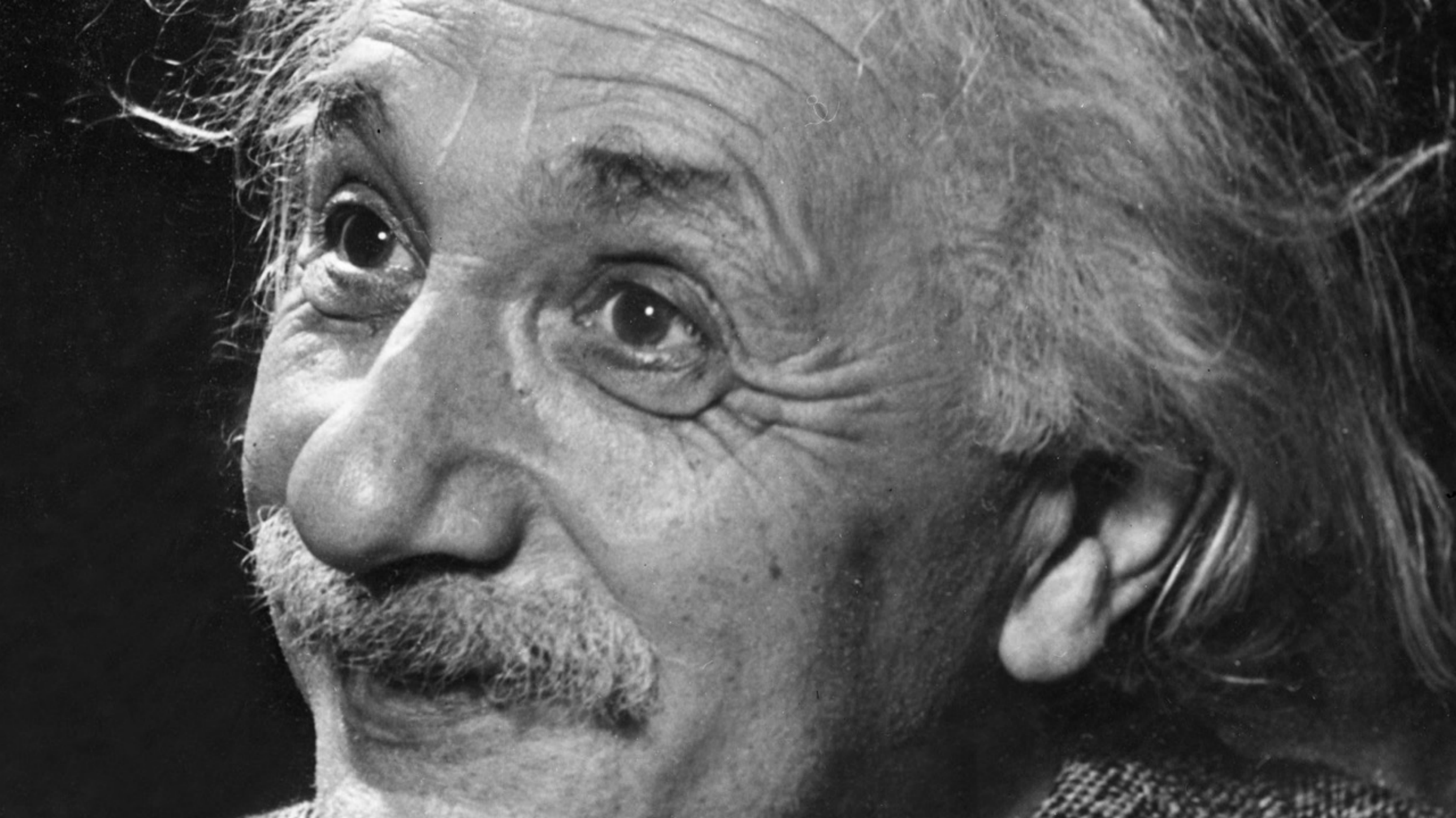
WHAT IS A **SAMPLE**?

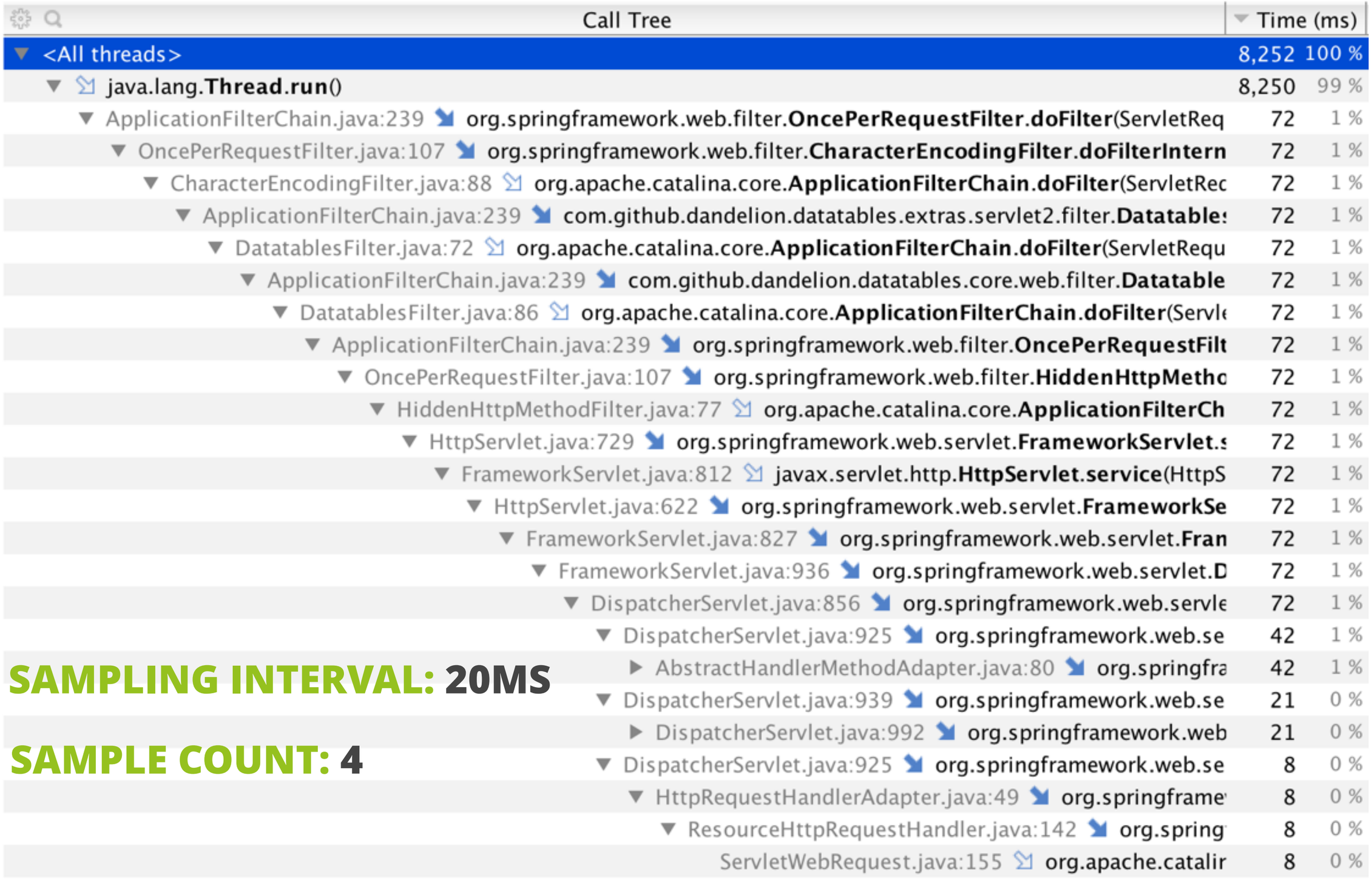
THREAD DUMP:

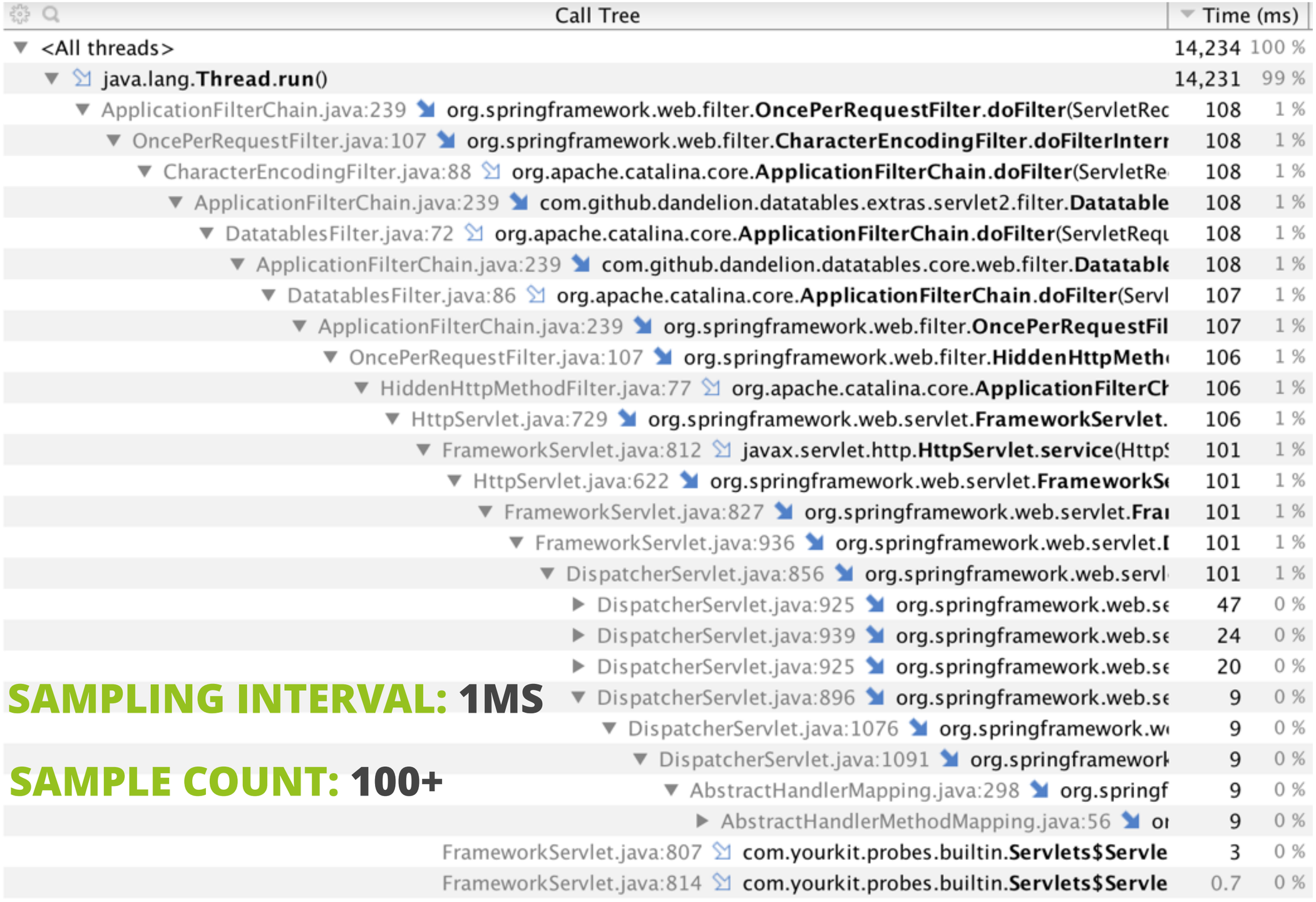
CTRL+BREAK ON WINDOWS

KILL -3 PID ON *NIX

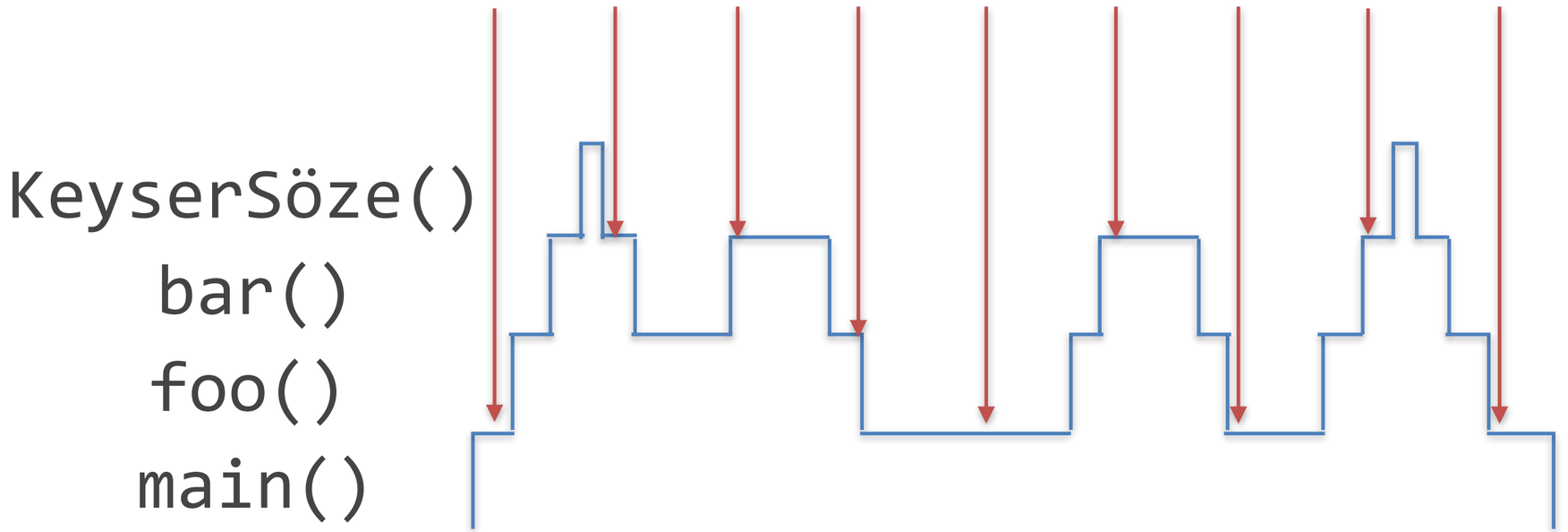
```
"http-nio-8080-exec-1" #40 daemon prio=5 os_prio=31 tid=0x00007fd7057ed800 nid=0x7313 runnable
   java.lang.Thread.State: RUNNABLE
      at o.s.s.p.w.OwnerController.processFindForm(OwnerController.java:89)
      at s.r.NativeMethodAccessorImpl.invoke0(Native Method)
      at s.r.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
      at s.r.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
      at j.l.r.Method.invoke(Method.java:497)
```




Call Tree
Time (ms)
<All threads> 8,252 100 %
java.lang.Thread.run() 8,250 99 %
ApplicationFilterChain.java:239 org.springframework.web.filter.OncePerRequestFilter.doFilter(ServletReq 72 1 %
OncePerRequestFilter.java:107 org.springframework.web.filter.CharacterEncodingFilter.doFilterIntern 72 1 %
CharacterEncodingFilter.java:88 org.apache.catalina.core.ApplicationFilterChain.doFilter(ServletRec 72 1 %
ApplicationFilterChain.java:239 com.github.dandelion.datatables.extras.servlet2.filter.Datatable 72 1 %
DatatablesFilter.java:72 org.apache.catalina.core.ApplicationFilterChain.doFilter(ServletRequ 72 1 %
ApplicationFilterChain.java:239 com.github.dandelion.datatables.core.web.filter.Datatable 72 1 %
DatatablesFilter.java:86 org.apache.catalina.core.ApplicationFilterChain.doFilter(Servl 72 1 %
ApplicationFilterChain.java:239 org.springframework.web.filter.OncePerRequestFilt 72 1 %
OncePerRequestFilter.java:107 org.springframework.web.filter.HiddenHttpMethc 72 1 %
HiddenHttpMethodFilter.java:77 org.apache.catalina.core.ApplicationFilterCh 72 1 %
HttpServlet.java:729 org.springframework.web.servlet.FrameworkServlet.s 72 1 %
FrameworkServlet.java:812 javax.servlet.http.HttpServlet.service(HttpS 72 1 %
HttpServlet.java:622 org.springframework.web.servlet.FrameworkSe 72 1 %
FrameworkServlet.java:827 org.springframework.web.servlet.Fran 72 1 %
FrameworkServlet.java:936 org.springframework.web.servlet.D 72 1 %
DispatcherServlet.java:856 org.springframework.web.servle 72 1 %
DispatcherServlet.java:925 org.springframework.web.se 42 1 %
AbstractHandlerMethodAdapter.java:80 org.springfra 42 1 %
DispatcherServlet.java:939 org.springframework.web.se 21 0 %
DispatcherServlet.java:992 org.springframework.web 21 0 %
DispatcherServlet.java:925 org.springframework.web.se 8 0 %
HttpRequestHandlerAdapter.java:49 org.springframe 8 0 %
ResourceHttpRequestHandler.java:142 org.spring 8 0 %
ServletWebRequest.java:155 org.apache.catalir 8 0 %
SAMPLING INTERVAL: 20MS
SAMPLE COUNT: 4

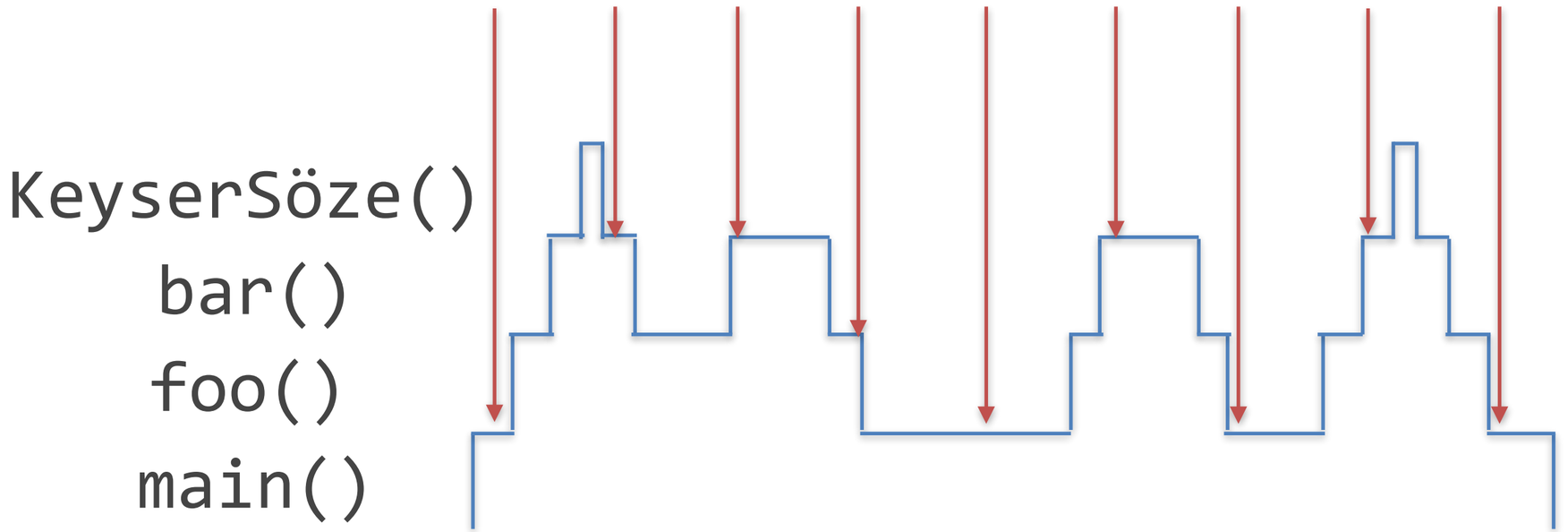

Call Tree
Time (ms)
<All threads> 14,234 100 %
java.lang.Thread.run() 14,231 99 %
ApplicationFilterChain.java:239 org.springframework.web.filter.OncePerRequestFilter.doFilter(ServletRec 108 1 %
OncePerRequestFilter.java:107 org.springframework.web.filter.CharacterEncodingFilter.doFilterInterr 108 1 %
CharacterEncodingFilter.java:88 org.apache.catalina.core.ApplicationFilterChain.doFilter(ServletRe 108 1 %
ApplicationFilterChain.java:239 com.github.dandelion.datatables.extras.servlet2.filter.Datatable 108 1 %
DatatablesFilter.java:72 org.apache.catalina.core.ApplicationFilterChain.doFilter(ServletRequ 108 1 %
ApplicationFilterChain.java:239 com.github.dandelion.datatables.core.web.filter.Datatable 108 1 %
DatatablesFilter.java:86 org.apache.catalina.core.ApplicationFilterChain.doFilter(Servl 107 1 %
ApplicationFilterChain.java:239 org.springframework.web.filter.OncePerRequestFil 107 1 %
OncePerRequestFilter.java:107 org.springframework.web.filter.HiddenHttpMeth 106 1 %
HiddenHttpMethodFilter.java:77 org.apache.catalina.core.ApplicationFilterCh 106 1 %
HttpServlet.java:729 org.springframework.web.servlet.FrameworkServlet. 106 1 %
FrameworkServlet.java:812 javax.servlet.http.HttpServlet.service(Https 101 1 %
HttpServlet.java:622 org.springframework.web.servlet.FrameworkSe 101 1 %
FrameworkServlet.java:827 org.springframework.web.servlet.Fra 101 1 %
FrameworkServlet.java:936 org.springframework.web.servlet.I 101 1 %
DispatcherServlet.java:856 org.springframework.web.servl 101 1 %
DispatcherServlet.java:925 org.springframework.web.se 47 0 %
DispatcherServlet.java:939 org.springframework.web.se 24 0 %
DispatcherServlet.java:925 org.springframework.web.se 20 0 %
DispatcherServlet.java:896 org.springframework.web.se 9 0 %
DispatcherServlet.java:1076 org.springframework.w 9 0 %
DispatcherServlet.java:1091 org.springframeworl 9 0 %
AbstractHandlerMapping.java:298 org.springf 9 0 %
AbstractHandlerMethodMapping.java:56 or 9 0 %
FrameworkServlet.java:807 com.yourkit.probes.builtin.Servlets$Servle 3 0 %
FrameworkServlet.java:814 com.yourkit.probes.builtin.Servlets$Servle 0.7 0 %
SAMPLING INTERVAL: 1MS
SAMPLE COUNT: 100+

SAMPLING

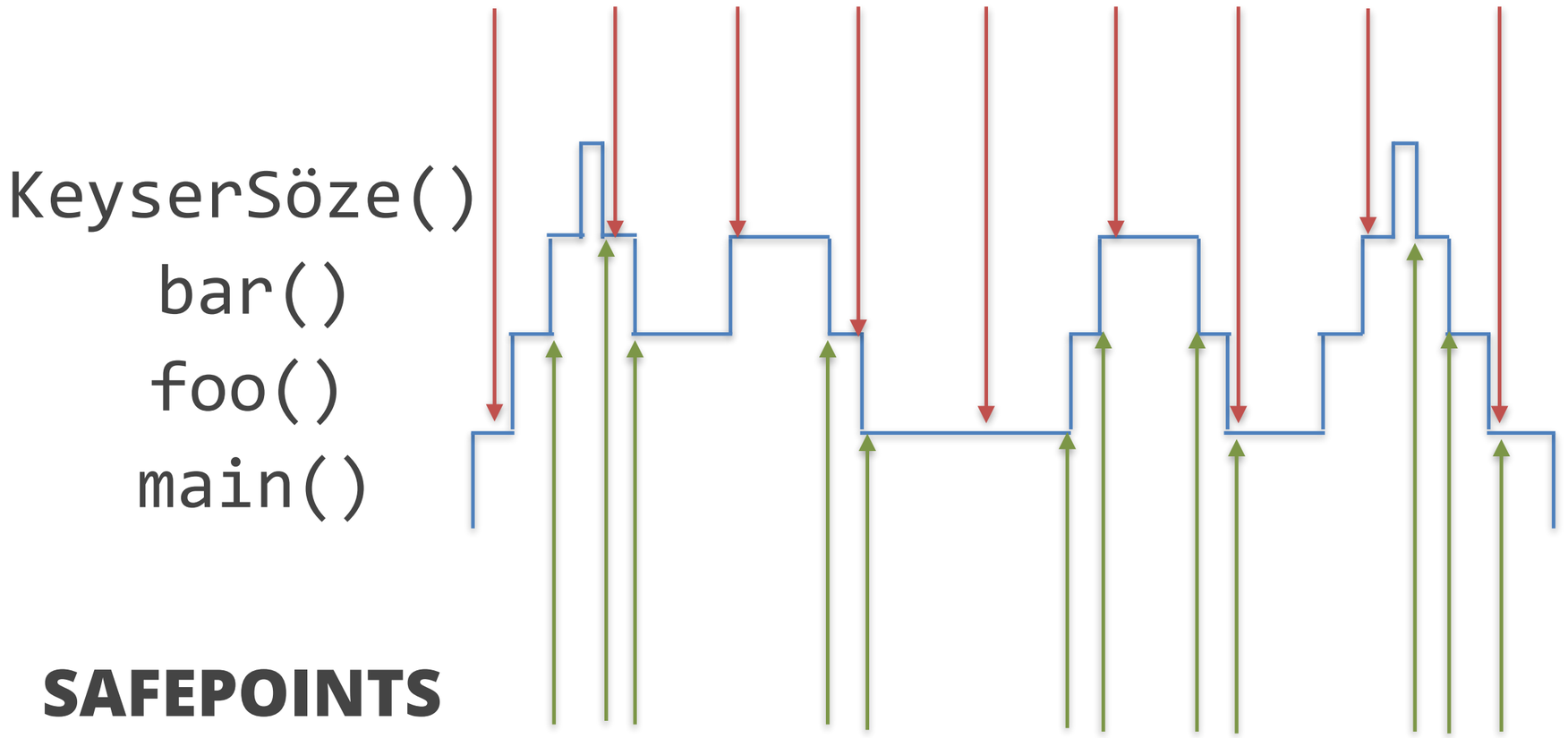



SAFEPOINTS

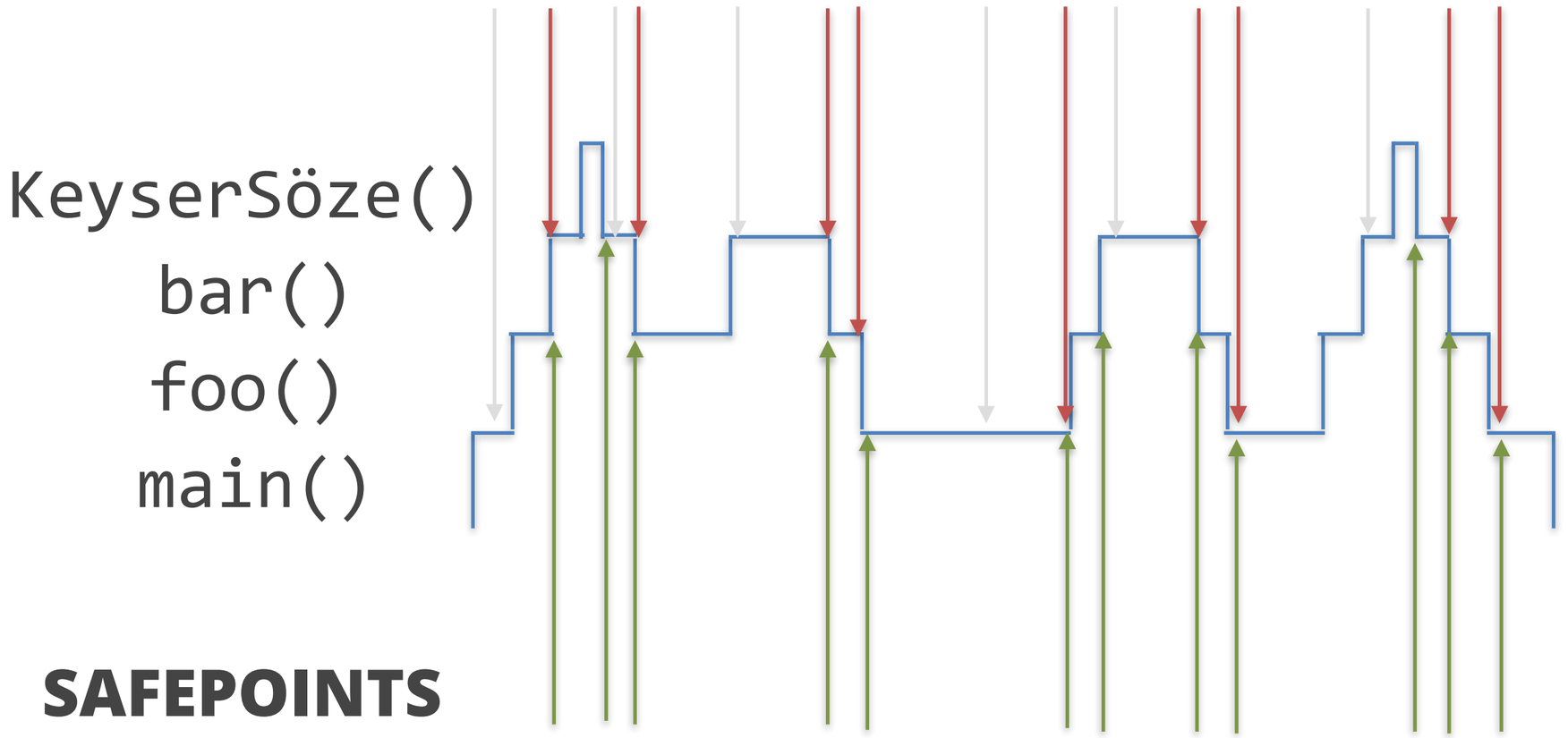
SAMPLING



SAFEPOINT BIAS



SAFEPOINT BIAS




CAN IT BE
TAMED?

TAMING SAFEPOINT BIAS

Java Mission Control

Proprietary protocol

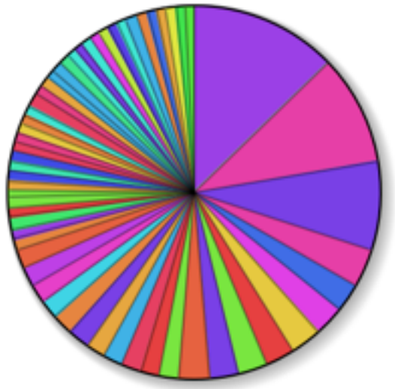
Java 7u40

Honest Profiler

JVMTI

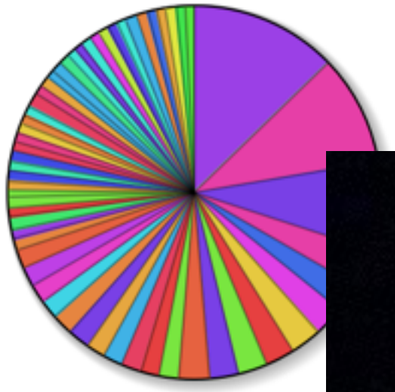
AsyncGetCallTrace

https://github.com/RichardWarburton/honest-profiler

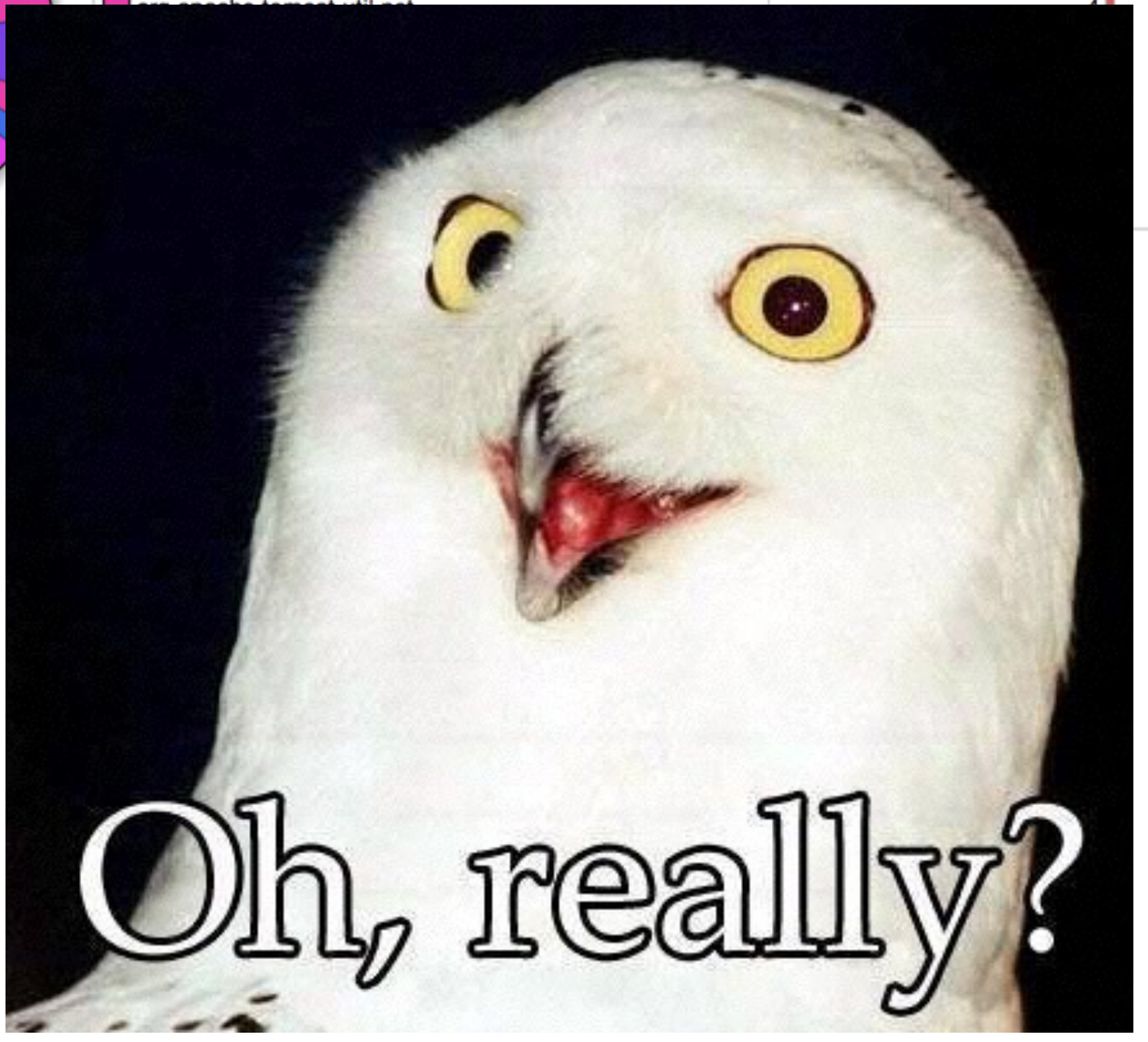


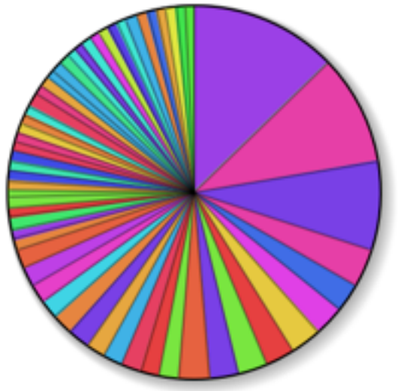
Package	Sample Count	Percentage
java.util	15	12.82%
org.hsqldb	11	9.40%
org.eclipse.jdt.internal.compiler.lookup	9	7.69%
org.apache.tomcat.util.net	4	3.42%
org.apache.jasper.compiler	3	2.56%
java.lang	3	2.56%
sun.nio.ch	3	2.56%
org.apache.coyote.http11	3	2.56%
org.apache.catalina.connector	3	2.56%
javax.el	3	2.56%

JAVA.UTIL IS HOT?



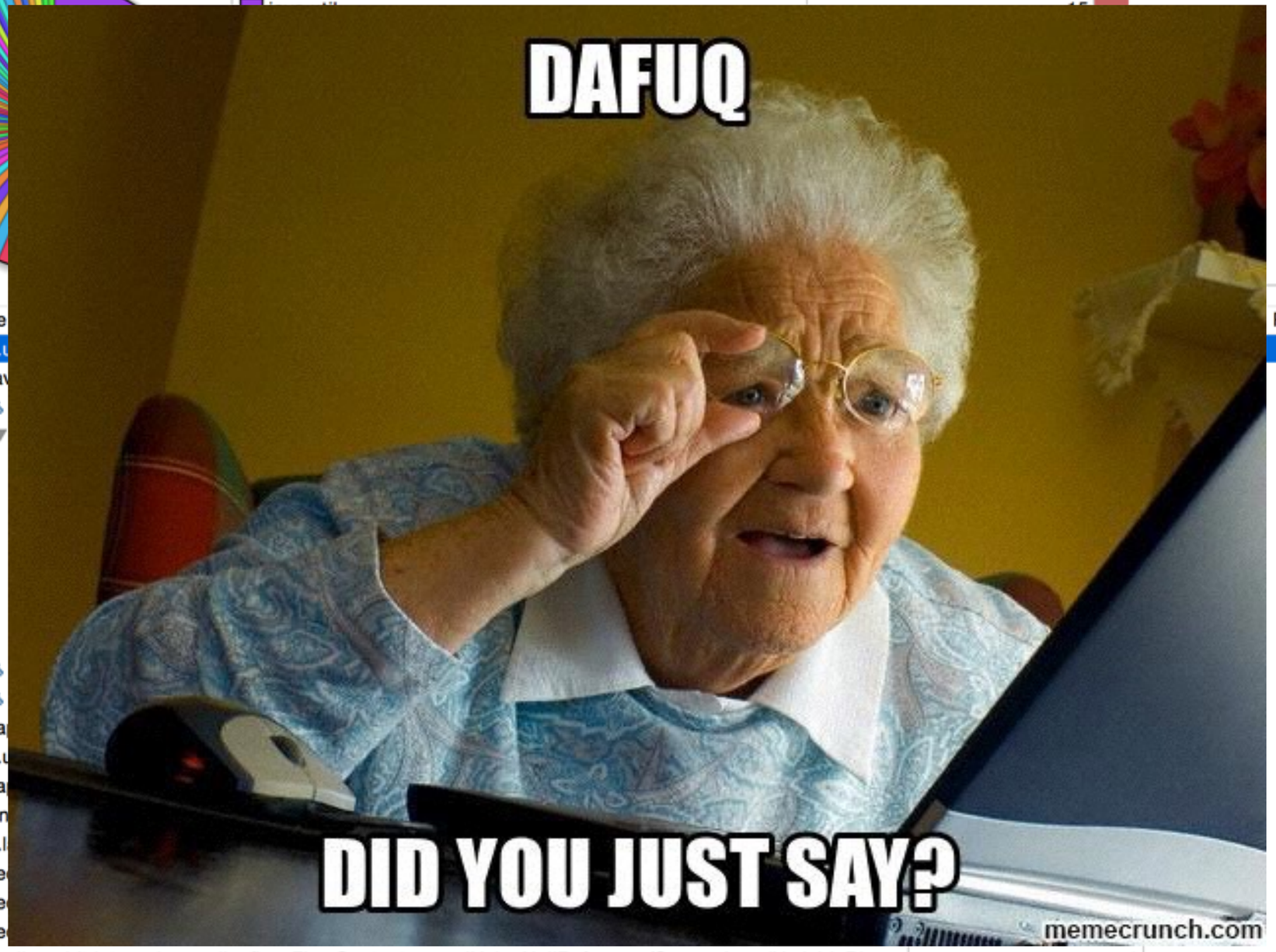
Package	Sample Count	Percentage
java.util	15	12.82%
org.hsqldb	11	9.40%
org.eclipse.jdt.internal.compiler.lookup	9	7.69%
[illegible]	4	3.42%
[illegible]	3	2.56%
[illegible]	3	2.56%
[illegible]	3	2.56%
[illegible]	3	2.56%
[illegible]	3	2.56%
[illegible]	3	2.56%





Package	Sample Count	Percentage
java.util	15	12.82%
org.hsqldb	11	9.40%
org.eclipse.jdt.internal.compiler.lookup	9	7.69%
org.apache.tomcat.util.net	4	3.42%
org.apache.jasper.compiler	3	2.56%
java.lang	3	2.56%
sun.nio.ch	3	2.56%
org.apache.coyote.http11	3	2.56%
org.apache.catalina.connector	3	2.56%
javax.el	3	2.56%

Stack Trace	Sample Count	Percentage
java.util.HashMap.getNode(int, Object)	5	4.27%
java.util.HashMap.get(Object)	5	4.27%
org.apache.catalina.webresources.AbstractArchiveResourceSet.getResource(String)	3	2.56%
org.apache.catalina.webresources.StandardRoot.getResourceInternal(String, boolean)	3	2.56%
org.apache.catalina.webresources.Cache.getResource(String, boolean)	3	2.56%
org.apache.catalina.webresources.StandardRoot.getResource(String, boolean, boolean)	3	2.56%
org.apache.catalina.webresources.StandardRoot.getClassLoaderResource(String)	3	2.56%
org.apache.catalina.loader.WebappClassLoaderBase.findResourceInternal(String…	3	2.56%
org.apache.catalina.loader.WebappClassLoaderBase.findResource(String)	2	1.71%
org.apache.catalina.loader.WebappClassLoaderBase.getResourceAsStrea…	2	1.71%
org.apache.catalina.loader.WebappClassLoaderBase.findClassInternal(String)	1	0.85%
org.apache.catalina.connector.Request.getAttribute(String)	1	0.85%
java.util.Collections$UnmodifiableMap.get(Object)	1	0.85%
org.apache.catalina.webresources.StandardRoot.getResourceInternal(String, boolean)	2	1.71%
java.util.jar.JarFile.getEntry(String)	2	1.71%
org.apache.catalina.connector.Request.parseLocales()	2	1.71%
sun.nio.ch.Util$BufferCache.get(int)	1	0.85%
java.lang.AbstractStringBuilder.ensureCapacityInternal(int)	1	0.85%
org.eclipse.jdt.internal.compiler.parser.Scanner.getCurrentIdentifierSource()	1	0.85%
org.eclipse.jdt.internal.compiler.lookup.MethodBinding.getTypeVariable(char[])	1	0.85%
org.eclipse.jdt.internal.compiler.lookup.Scope.getBinding(char[], int, InvocationSite, boolean)	1	0.85%
org.eclipse.jdt.internal.compiler.flow.FlowContext.recordUsingNullReference(Scope, LocalVariableBin…	1	0.85%
org.eclipse.jdt.internal.compiler.codegen.CodeStream.isDefinitelyAssigned(Scope, int, LocalVariableBinding)	1	0.85%



Package	Sample Count	Percentage
		12.82%
		9.40%
		7.69%
		3.42%
		2.56%
		2.56%
		2.56%
		2.56%
		2.56%

Stack Trace	Sample Count	Percentage
java.u		4.27%
jav		4.27%
		2.56%
		2.56%
		2.56%
		2.56%
		2.56%
		2.56%
		2.56%
		1.71%
		1.71%
		0.85%
		0.85%
		0.85%
org.a		1.71%
java.u		1.71%
org.a		1.71%
sun.n		0.85%
java.l		0.85%
org.e		0.85%
org.e		0.85%
org.e		0.85%
org.eclipse.jdt.internal.compiler.flow.FlowContext.recordUsingNullReference(Scope, LocalVariableBin...	1	0.85%
org.eclipse.jdt.internal.compiler.codegen.CodeStream.isDefinitelyAssigned(Scope, int, LocalVariableBinding)	1	0.85%

TRACING

(INSTRUMENTATION)

TRACING

```
public void businessMethod() {
  long start = System.currentTimeMillis();

  work();

  Profiler.log(System.currentTimeMillis()-start);
}
```

TRACING

```
public void businessMethod() {
  long start = System.nanoTime();

  work();

  Profiler.log(System.nanoTime()-start);
}
```

TRACING

```
public void businessMethod() {
  Profiler.start(“businessMethod”);
  try {
    work();
  } finally {
    Profiler.log("businessMethod");
  }
}
```

TIMING

SYSTEM.**NANOTIME**

SYSTEM.**CURRENTTIMEMILLIS**

PARALLEL THREAD

- SLEEP-WAKEUP-UPDATE
- YIELD-WAKEUP-UPDATE
- BUSY-LOOP-UPDATE

```
class Profiler {

  Loop loop;

  public static void start(String method) {
    long now = loop.getTime();
    …
  }
```

```
public class Loop implements Runnable {
    private volatile long time;
    public void run() {
        while (running) {
            time = System.nanoTime();
            sleep();
        }
    }

    public final long getTime() {
        return time;
    }
```

NANO SECONDS

Reading memory is **not** free.

It takes cycles = nanoseconds

Each (software) layer is **not** free.

JVM, JNI, OS, HW

Nanotrusting the NanoTime

http://shipilev.net/blog/2014/nanotrusting-nanotime/

NANO SECONDS PUT INTO PERSPECTIVE

GRANULARITY_NANOTIME: 26.300 +- 0.205 NS
LATENCY_NANOTIME: 25.542 +- 0.024 NS

LINUX

GRANULARITY_NANOTIME: 29.322 +- 1.293 NS
LATENCY_NANOTIME: 29.910 +- 1.626 NS

SOLARIS

GRANULARITY_NANOTIME: 371.419 +- 1.541 NS
LATENCY_NANOTIME: 14.415 +- 0.389 NS

WINDOWS

Nanotrusting the NanoTime

http://shipilev.net/blog/2014/nanotrusting-nanotime/

SLEEP TIMER: TIME-SLICING & SCHEDULING

MINIMUM SLEEP TIMES:

Win8: 1.7 ms (1.0 ms using JNI + socket poll)

Linux: 0.1 ms

OS X: 0.05 ms

VirtualBox + Linux: don’t ask :)

Still uses 25-50% of CPU core

YIELD?

Windows scheduler **skips yielded threads** in case of CPU starvation

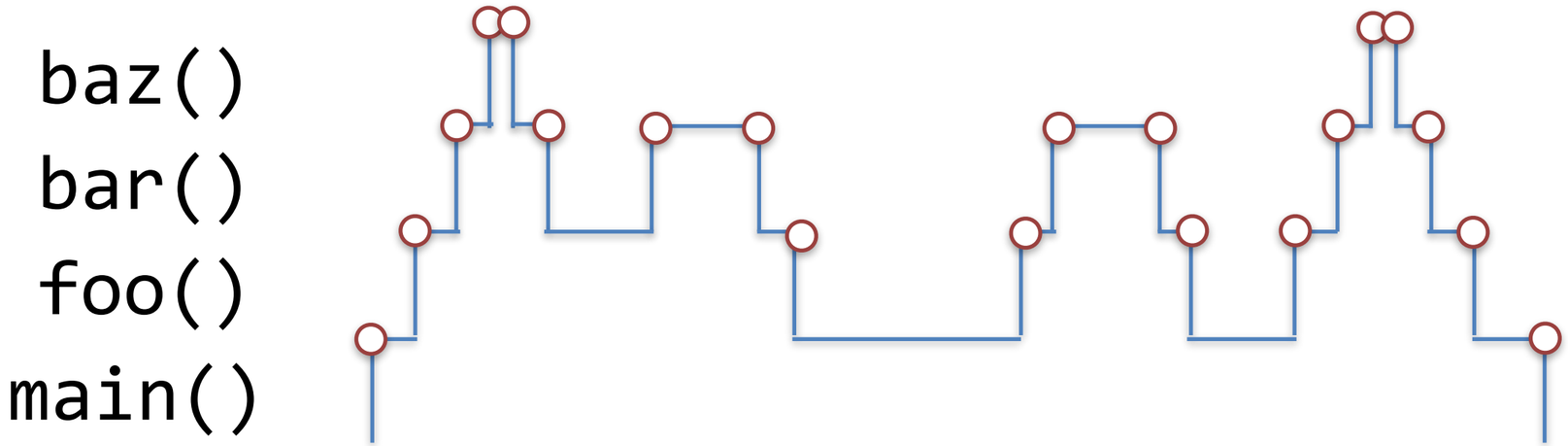
```
public class Loop implements Runnable {
    private volatile long time;
    public void run() {
        while (running) {
            time = System.nanoTime();
            sleep();
        }
    }
    private void sleep() {
      if (!MiscUtil.isWindows()) {
         Thread.yield();
      }
    }
```

```
public class Loop implements Runnable {
    private volatile long time;
    public void run() {
        while (running) {
            time = System.nanoTime();
            sleep();
        }
    }
    private void sleep() {
      if (!MiscUtil.isWindows()) {
         Thread.yield();
      }
    }
```

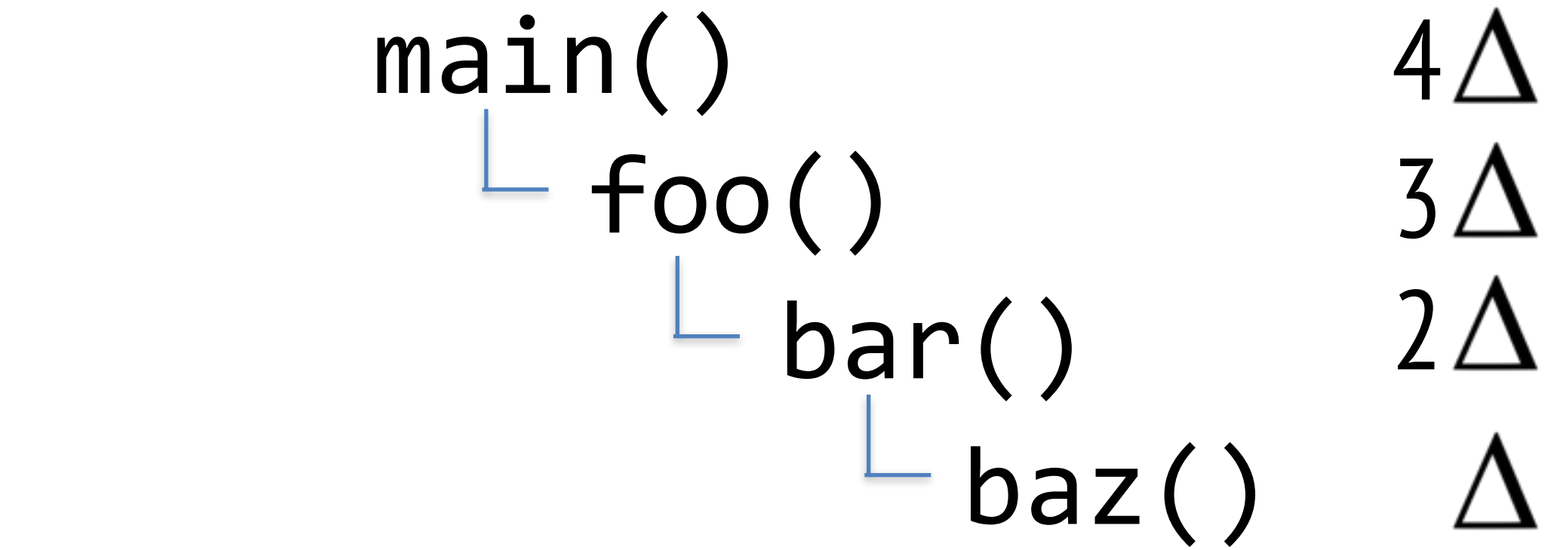
```
public class Loop implements Runnable {
    private volatile long time;
    public void run() {
        while (running) {
            time = System.nanoTime();
            sleep();
        }
    }
    private void sleep() {
      if (!MiscUtil.isWindows()) {
         Thread.yield();
      }
    }
```

```
public class Loop implements Runnable {
    private volatile long time;
    public void run() {
        while (running) {
            time = System.nanoTime();
            sleep();
        }
    }
    private void sleep() {
      if (!MiscUtil.isWindows()) {
         Thread.yield();
      }
    }
```

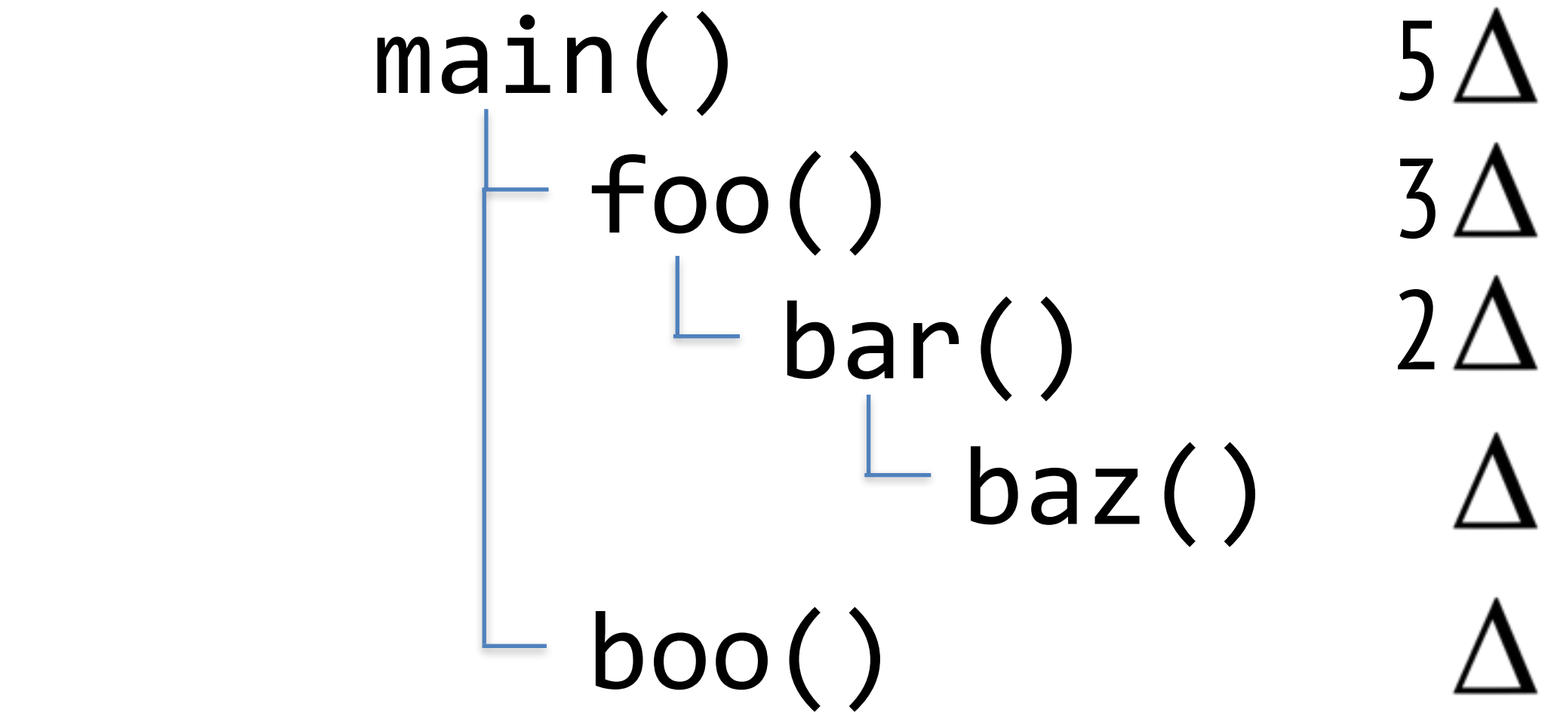
TRACING

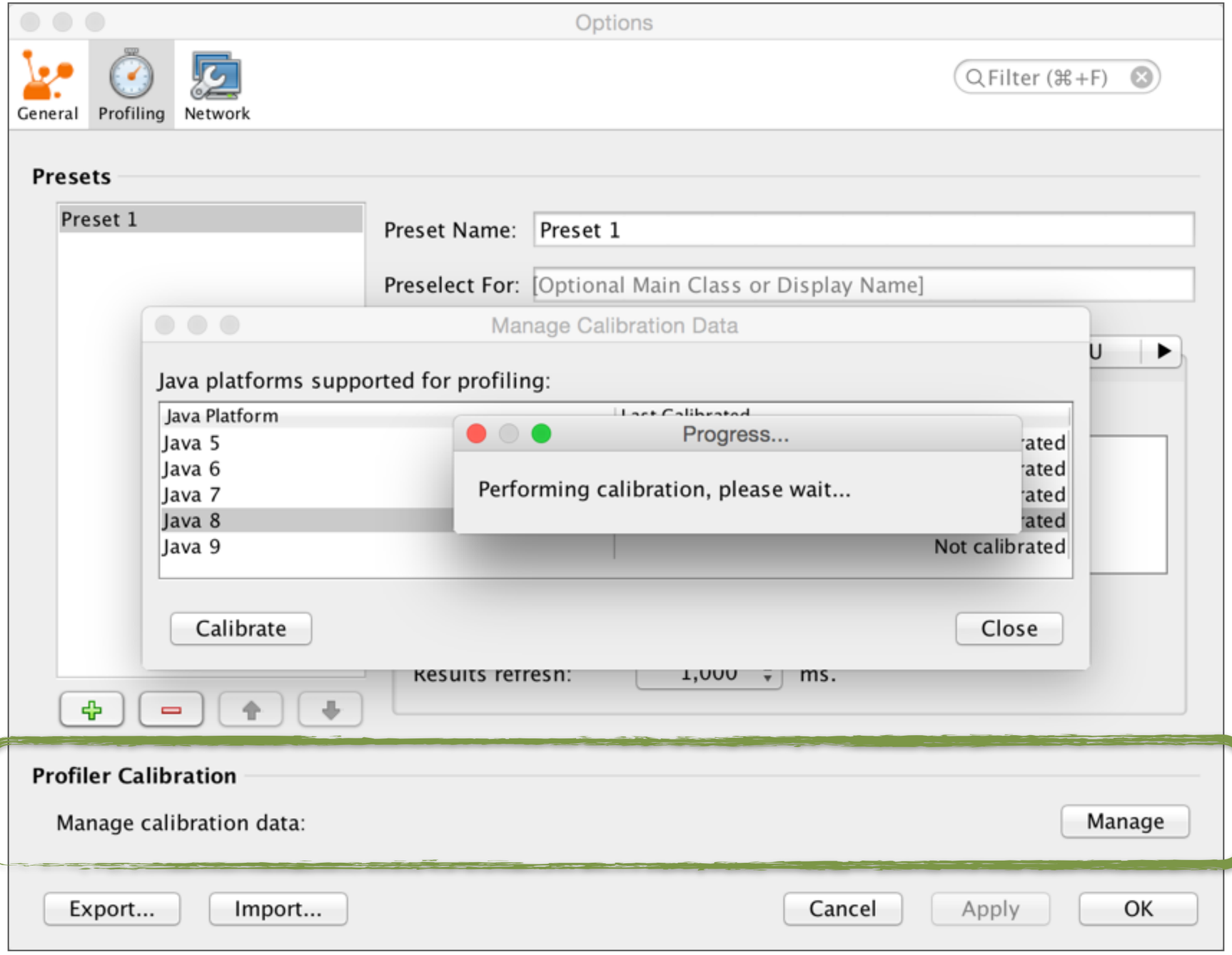


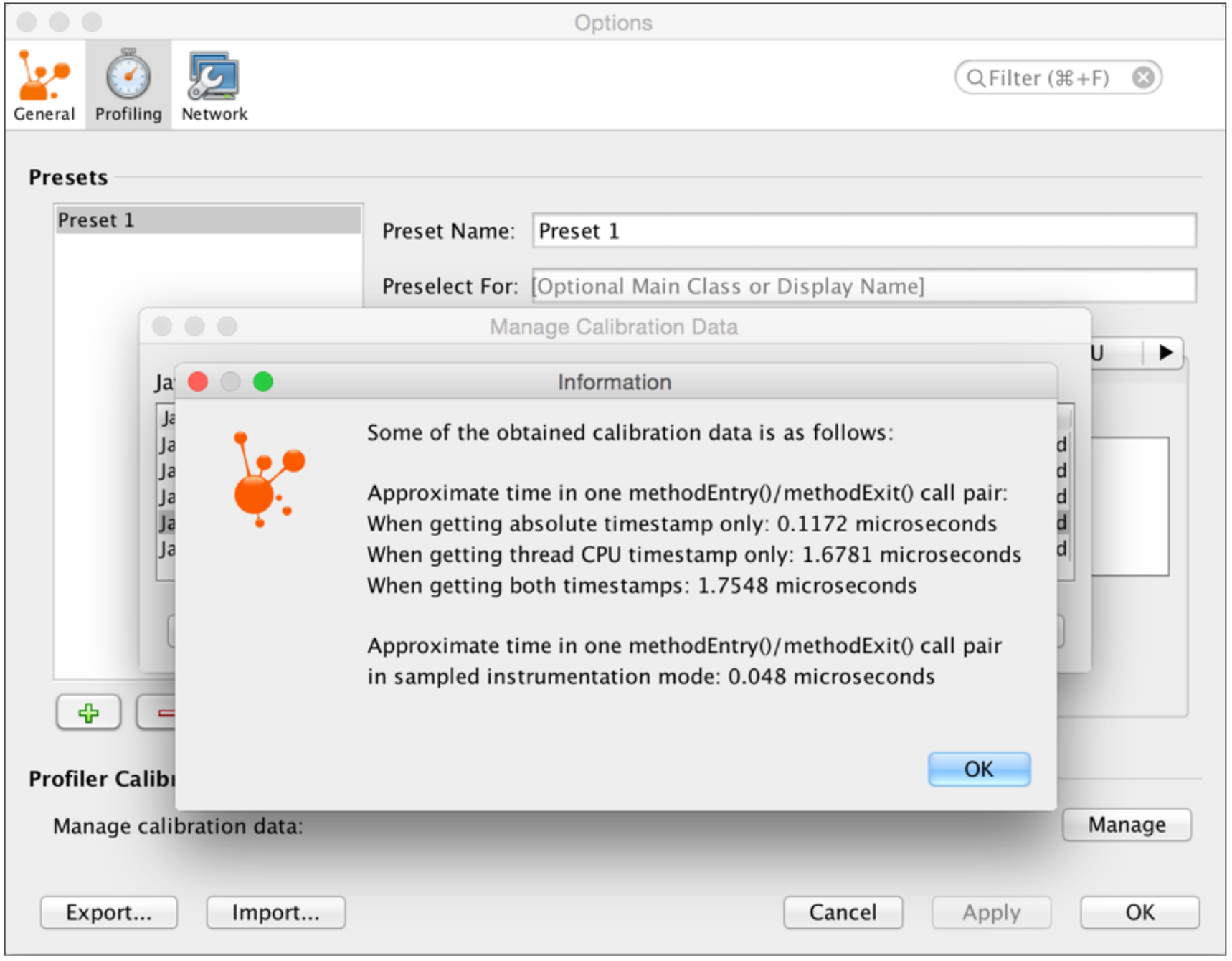
TRACING



TRACING

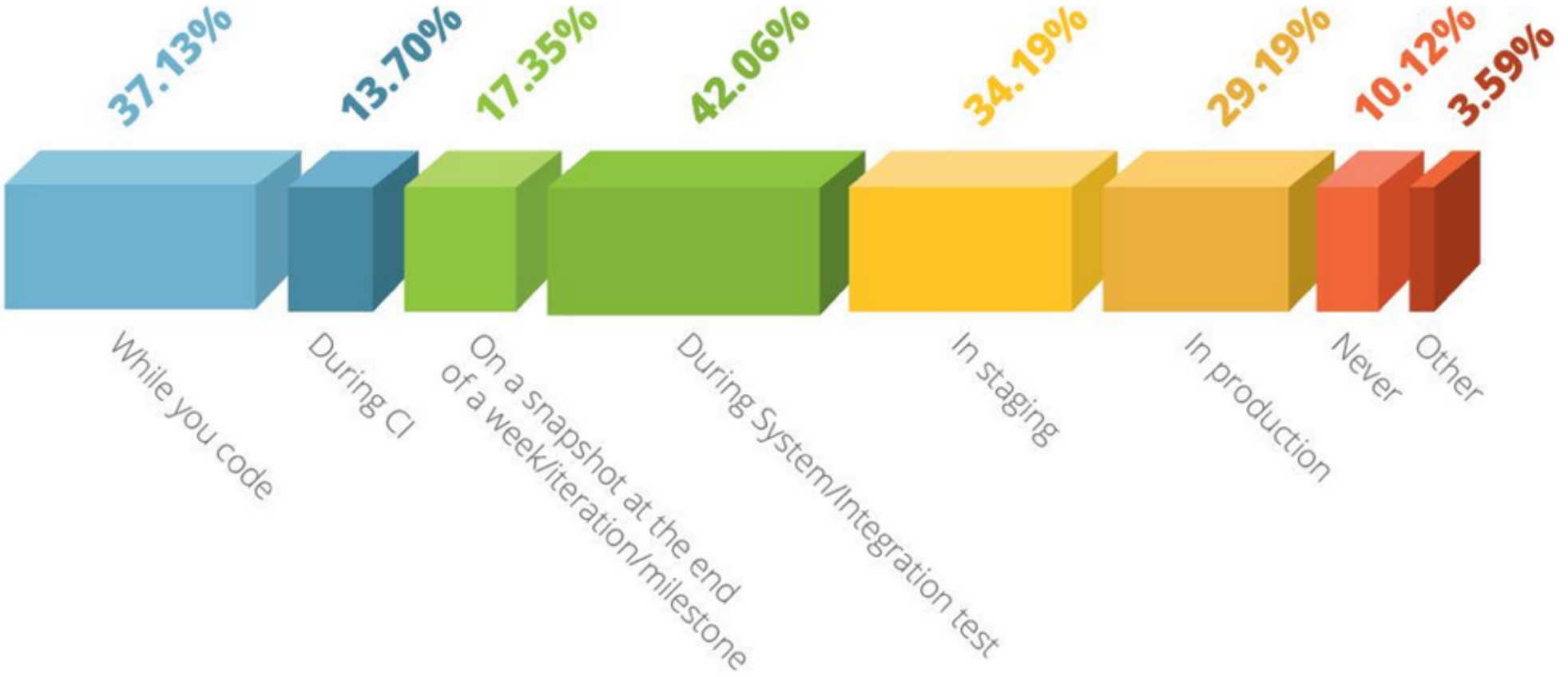



Options
General
Profiling
Network
Filter (⌘+F)
Presets
Preset 1
Preset Name: Preset 1
Preselect For: [Optional Main Class or Display Name]
Manage Calibration Data
Java platforms supported for profiling:
Java Platform
Java 5
Java 6
Java 7
Java 8
Java 9
Not calibrated
Progress...
Performing calibration, please wait...
Calibrate
Close
Results refresh:
1,000
ms.
Profiler Calibration
Manage calibration data:
Manage
Export...
Import...
Cancel
Apply
OK


Options
General
Profiling
Network
Filter (⌘+F)
Presets
Preset 1
Preset Name:
Preset 1
Preselect For:
[Optional Main Class or Display Name]
Manage Calibration Data
Information
Some of the obtained calibration data is as follows:
Approximate time in one methodEntry()/methodExit() call pair:
When getting absolute timestamp only: 0.1172 microseconds
When getting thread CPU timestamp only: 1.6781 microseconds
When getting both timestamps: 1.7548 microseconds
Approximate time in one methodEntry()/methodExit() call pair
in sampled instrumentation mode: 0.048 microseconds
OK
Profiler Calibr
Manage calibration data:
Manage
Export...
Import...
Cancel
Apply
OK

PERFORMANCE **PIPELINE**

At what stage do you perform profiling and performance tuning on your applications?



source: http://zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/

THE **SOFTWARE** **MANTRA**

1. Functionality
2. Quality
3. Performance
4. Security

THE **SOFTWARE** **MANTRA**

1. Functionality
2. Quality
3. Performance
4. Security

THE **SOFTWARE** **MANTRA**

1. Functionality - MAKE IT **WORK**
2. Quality
3. Performance
4. Security

THE **SOFTWARE** **MANTRA**

1. Functionality - MAKE IT **WORK**
2. Quality - MAKE IT WORK **WELL**
3. Performance
4. Security

THE **SOFTWARE** **MANTRA**

1. Functionality - MAKE IT **WORK**
2. Quality - MAKE IT WORK **WELL**
3. Performance - MAKE IT WORK **FAST**
4. Security

THE **SOFTWARE** **MANTRA**

1. Functionality - MAKE IT **WORK**
2. Quality - MAKE IT WORK **WELL**
3. Performance - MAKE IT WORK **FAST**
4. Security - ? ? ? ? ? ? ? ? ?

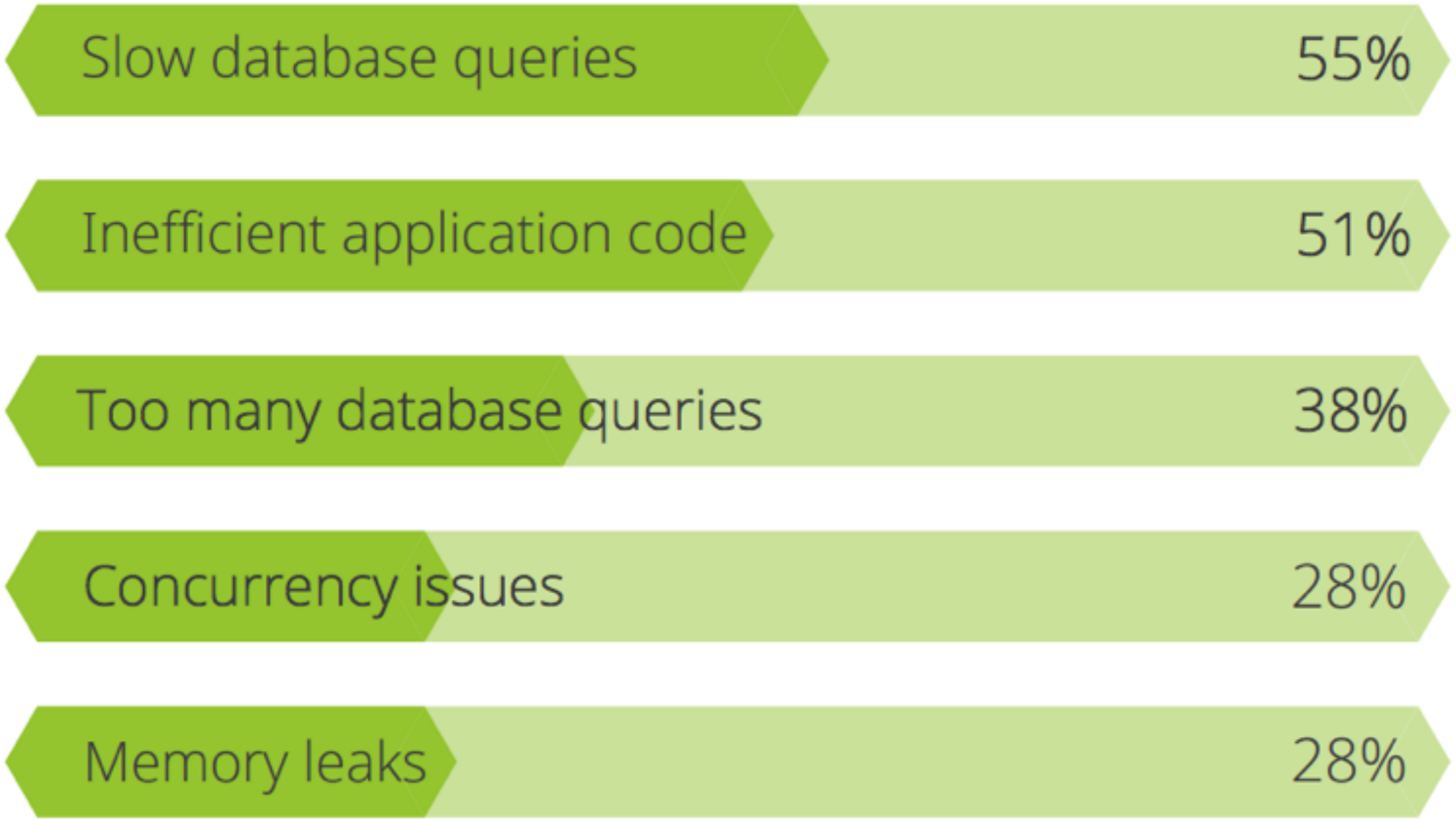
THE **SOFTWARE** **MANTRA**

1. Functionality - MAKE IT **WORK**
2. Quality - MAKE IT WORK **WELL**
3. Performance - MAKE IT WORK **FAST**
4. Security - MAKE IT WORK **SECURELY**

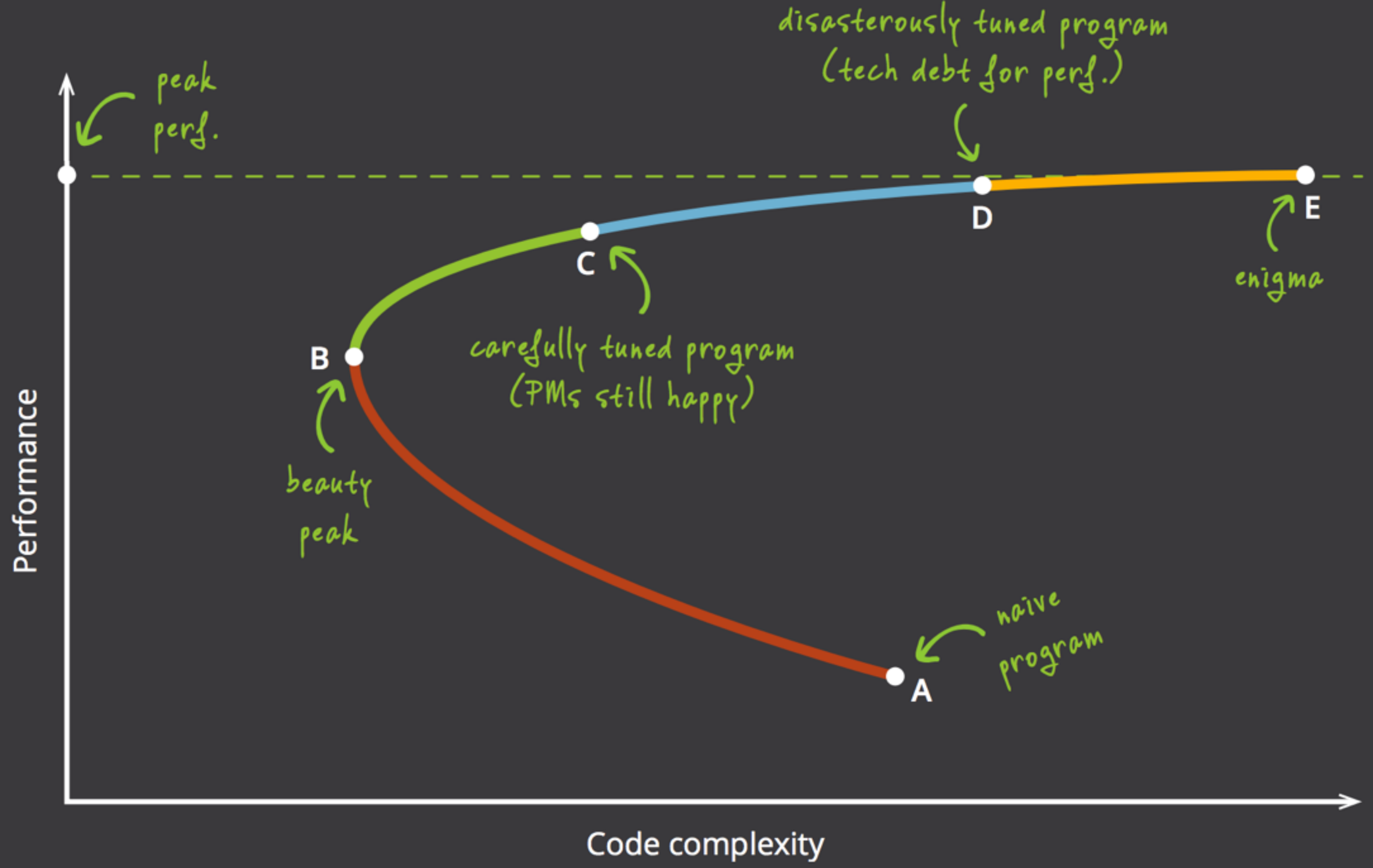
THE **SOFTWARE** **MANTRA**

1. Functionality - MAKE IT WORK
2. Quality - MAKE IT WORK WELL
3. **Performance - MAKE IT WORK FAST**
4. Security - MAKE IT WORK SECURELY

TOP 5 **PERFORMANCE** ISSUES



source: http://zeroturnaround.com/rebellabs/developer-productivity-report-2015-java-performance-survey-results/


disasterously tuned program
(tech debt for perf.)
peak
perf.
D
E
C
enigma
carefully tuned program
(PMs still happy)
B
beauty
peak
naive
program
A
Performance
Code complexity

REACTIVE OR **PROACTIVE**?

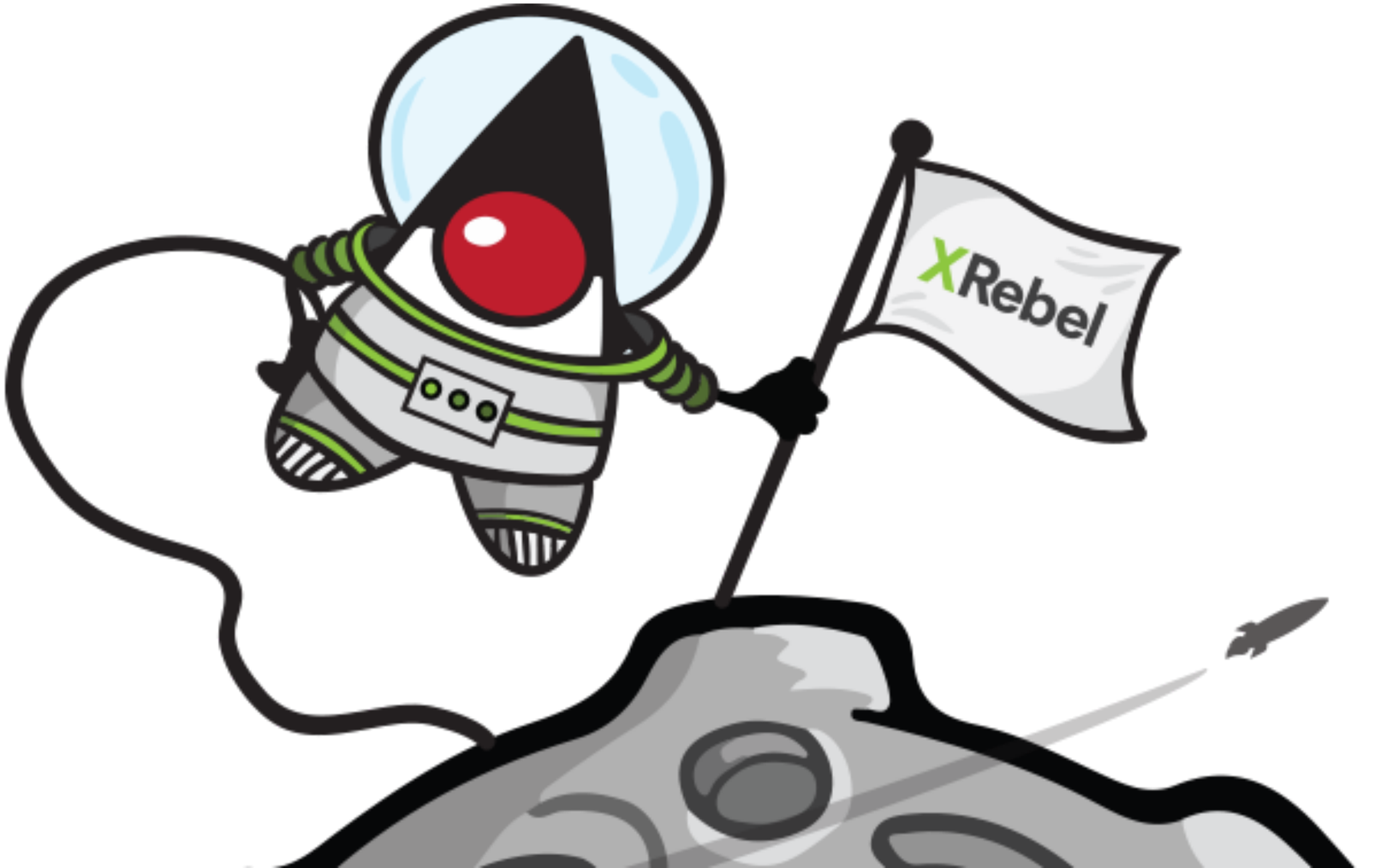
Reactive - Wait until an issue occurs, then resolve it as quickly as possible.

Proactive - prevent POTENTIAL issues from EVER occurring.

THE **PIPELINE**

	Lightweight Profiler	Classic Profiler	Performance tests	APM
Stage	Development	n/a	QA	Production
Type	Proactive	Research	Proactive	Reactive
Cost	$	$$	$$$$	$$$
Common issues	✔	✔	✔	✔
Other issues	✘	✔	✔	✔

INTRODUCING...



FREE STUFF!

JRebel

0t.ee/javaone-jr

XRebel

0t.ee/javaone-xr


YES!!!
I KILLED THAT PRESENTATION! ANY QUESTIONS?
memegenerator.net