Do you trust profilers? I once did too

Johannes Bechberger

Nightmares of a profiler developer

Johannes Bechberger's gruesome tales on Java profiling APIs

Image by Yatheesh Gowda from Pixabay

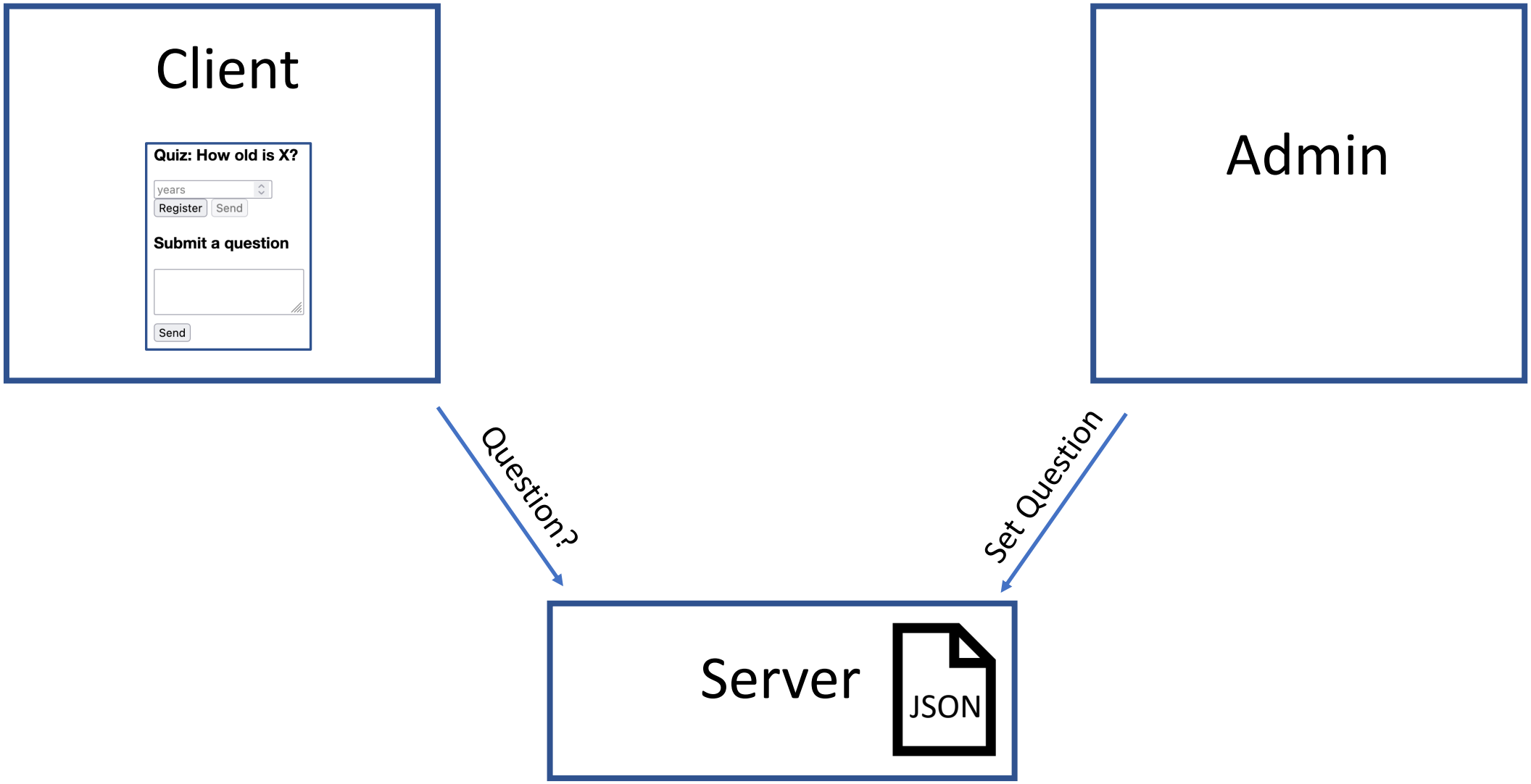
Quiz: How old is X?

years

Register Send

Submit a question

Send


Client
Quiz: How old is X?
years
Register
Send
Submit a question
Send
Admin
Question?
Set Question
Server
JSON

Premature Optimization

“premature optimization is the root of all evil.

— Donald Knuth

Source: Jacob Appelbaum, Wikipedia

So the event came...

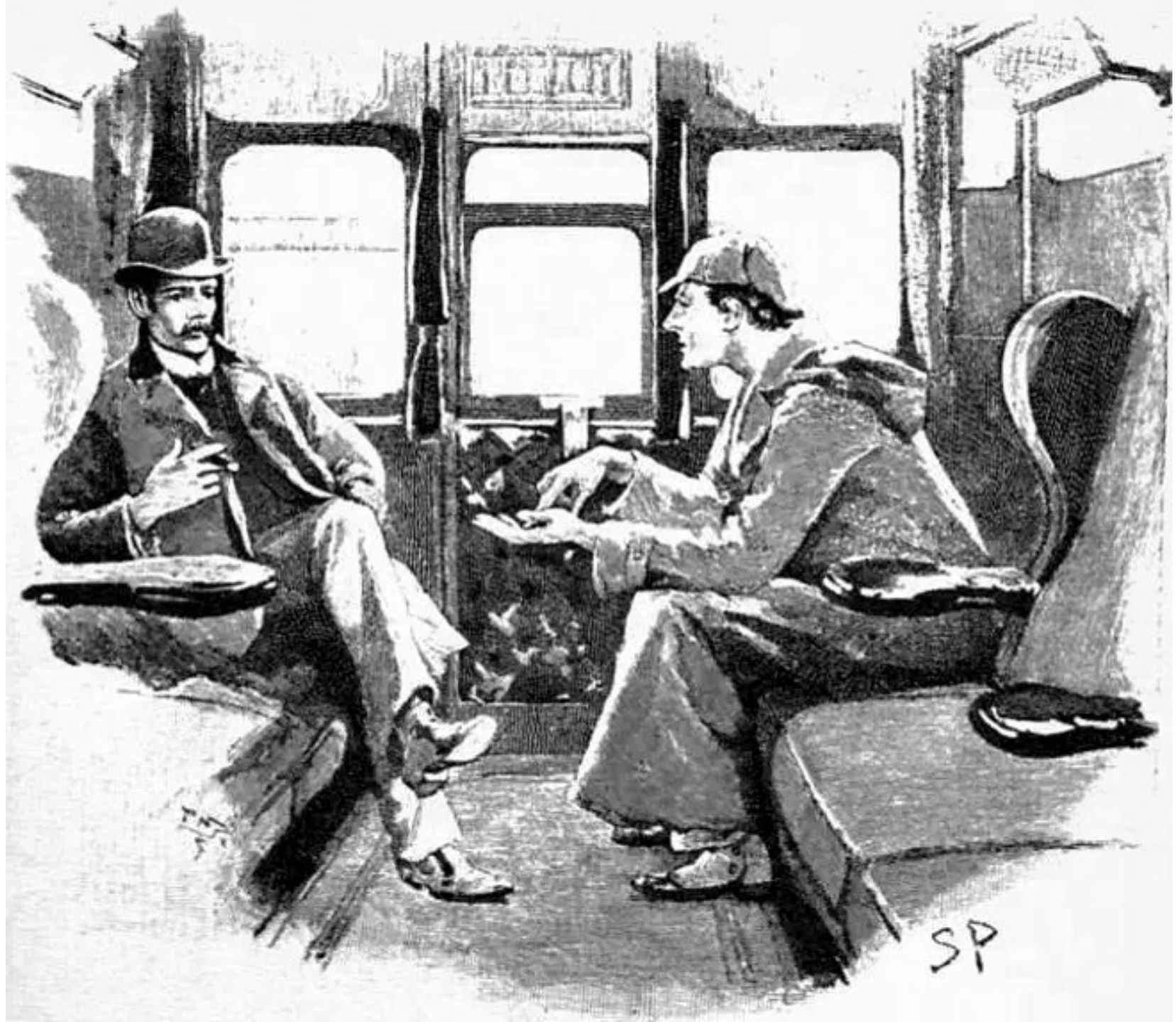
... and the server was overwhelmed

Premature Optimization

> 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%.
>
> A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; **but only after that code has been identified**.
>
> — Donald Knuth

Source: Jacob Appelbaum, Wikipedia

Profilers to the rescue

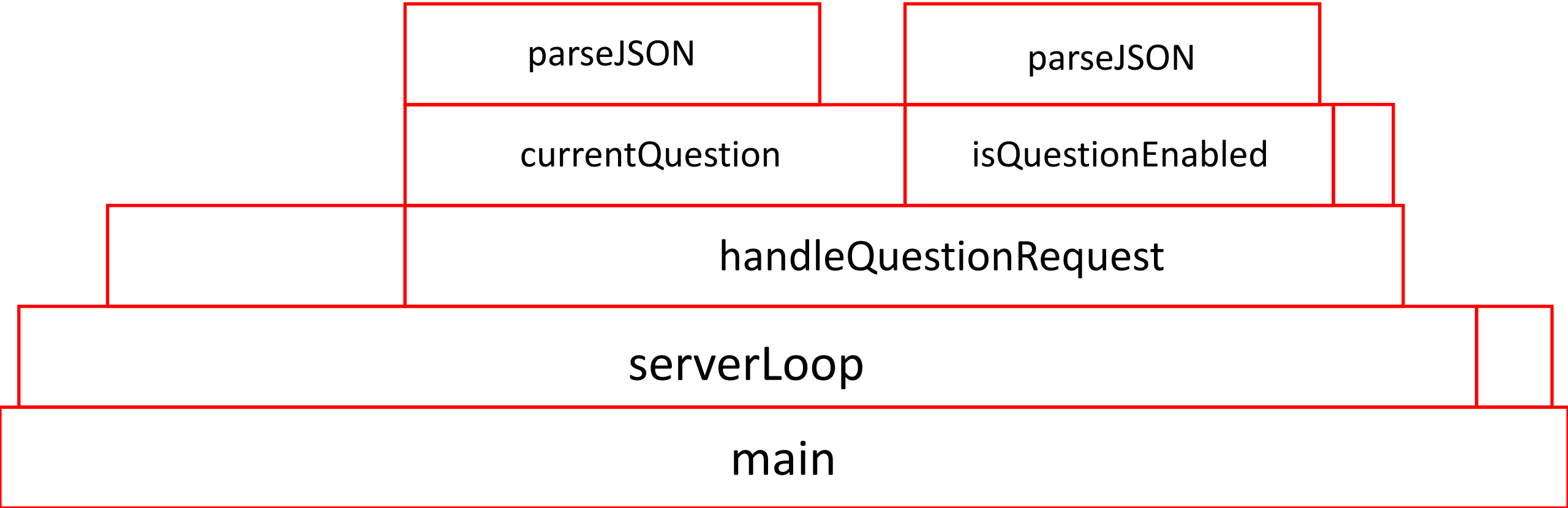

SP

Source: Abgedreht Karlsruhe

```
fun main() {
  ...
  serverLoop()
}
```

```
fun serverLoop() {
  while (true) {
    req = ...
    if (req.isQuestionRequest)
      handleQuestionRequest(req)
    ...
  }
}
```

```
fun handleQuestionRequest(req) {
  if (isQuestionEnabled()) {
    emit(currentQuestion().json)
  } else {
    emit({})
  }
}
```



Mario Fusco 🇪🇺🇺🇦 **@mariofusco@jvm.social**

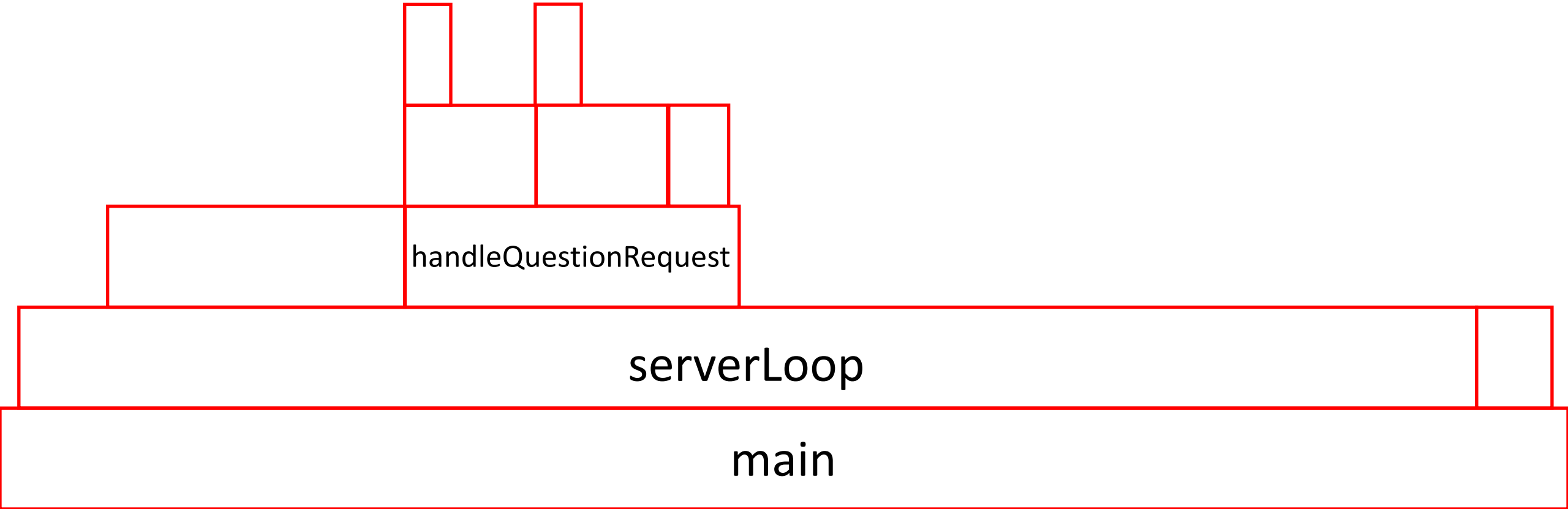
@mariofusco

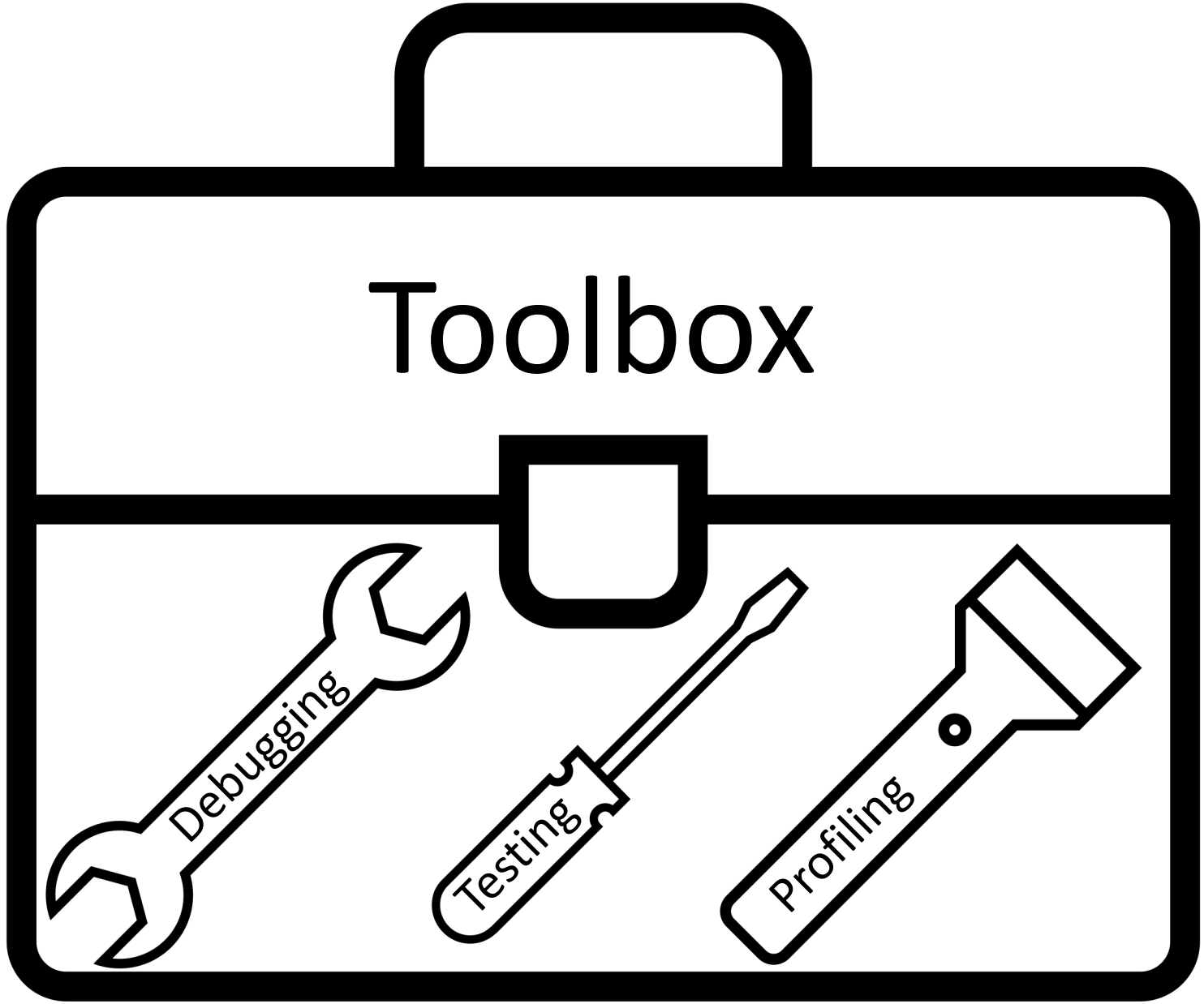
I love flamegraphs. When you do something stupid, it punches in your face and it's impossible not see it.

```
fun main() {
  ...
  serverLoop()
}
```

```
fun serverLoop() {
  while (true) {
    req = ...
    if (req.isQuestionRequest)
      handleQuestionRequest(req)
    ...
  }
}
```

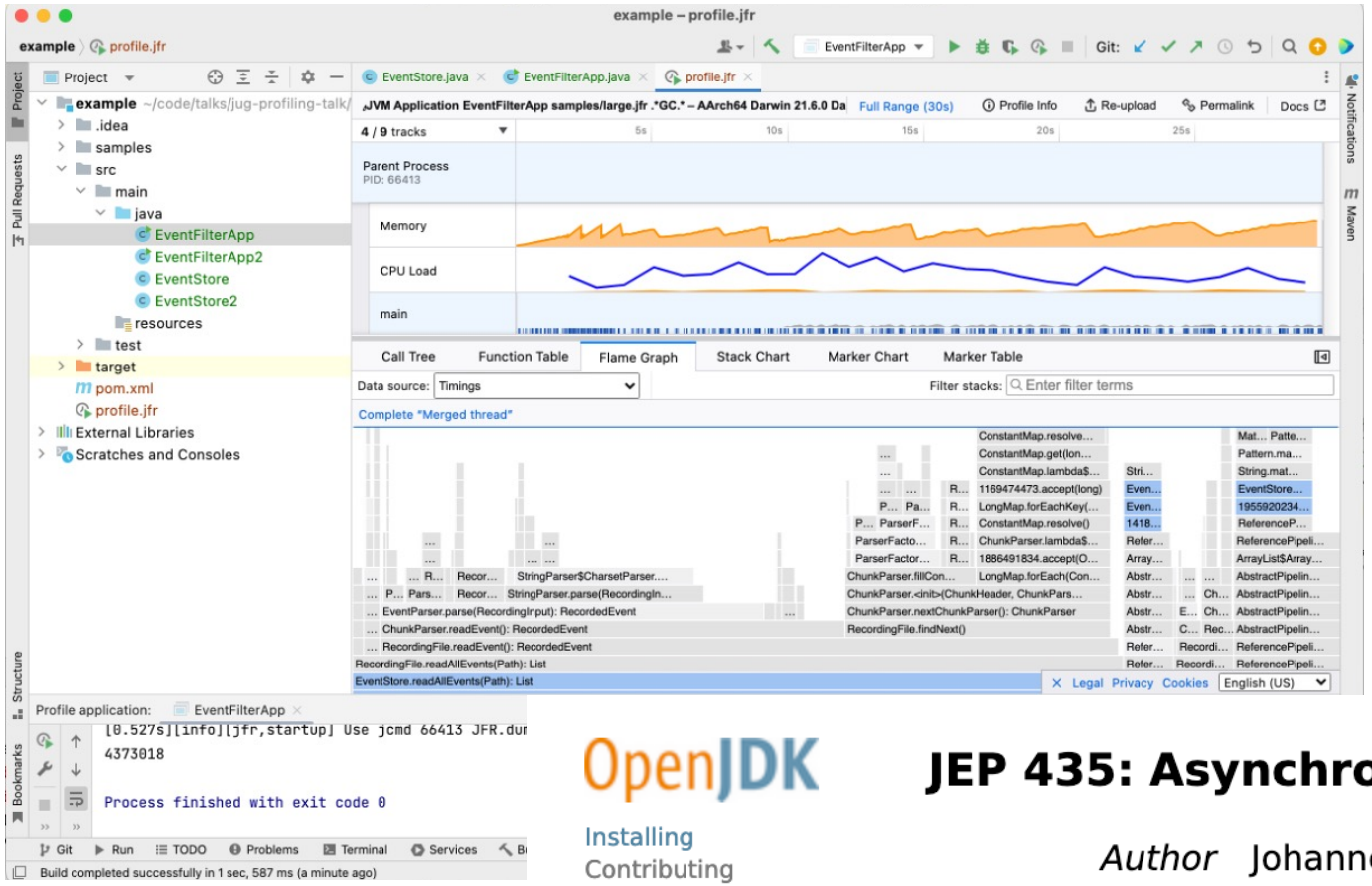
```
fun handleQuestionRequest(req) {
  if (isQuestionEnabled()) {
    emit(currentQuestion().json)
  } else {
    emit({})
  }
}
```




Toolbox
Debugging
Testing
Profiling


jar profiler


Sap
Machine



async-profiler

This project is a low overhead sampling profiler for Java that does not suffe
features HotSpot-specific APIs to collect stack traces and to track memory
with OpenJDK, Oracle JDK and other Java runtimes based on the HotSpot J

async-profiler can trace the following kinds of events:

- CPU cycles
- Hardware and Software performance counters like cache misses, branc

OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC

Bylaws · Census
Legal

JEP Process

Source code
Mercurial
GitHub

Tools
Git
jtreg harness

JEP 435: Asynchronous Stack Trace VM API

Author	Johannes Bechberger
Owner	Christoph Langer
Type	Feature
Scope	JDK
Status	Candidate
Component	hotspot / svc
Discussion	serviceability dash dev at openjdk dot org
Effort	S
Duration	S
Reviewed by	Andrei Pangin, Christoph Langer, Jaroslav Bachorík
Created	2022/04/04 11:02
Updated	2023/01/19 13:08

Who of you used a profiler?

THE NEW HACKER'S DICTIONARY

second edition

foreword and cartoons by Guy L. Steele Jr.

compiled by

ERIC S. RAYMOND

What is profiling?

> A report on the amounts of time spent in each routine of a program, used to find and tune away the hot spots in it.
>
> — The Jargon File

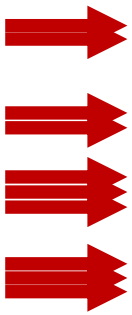
Different Profilers

Instrumenting Profilers *Inserting instructions into the code automatically*

```
fun serverLoop() {
  logEntry(“serverLoop”)
  while (true) {
    req = ...
    if (req.isQuestionRequest)
      handleQuestionRequest(req)
    ...
  }
  logExit(“serverLoop”)
}
```

But JITs?

Sampling Profilers

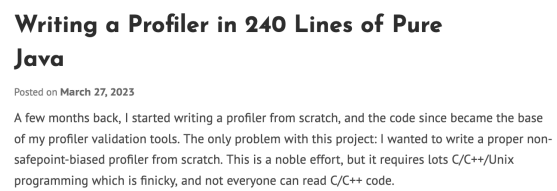


```
fun serverLoop() {

  while (true) {
    req = ...
    if (req.isQuestionRequest)
      handleQuestionRequest(req)
    ...
  }

}
```

Profilers aren’t rocket science

So let’s write our own*

*ignoring safepoint bias

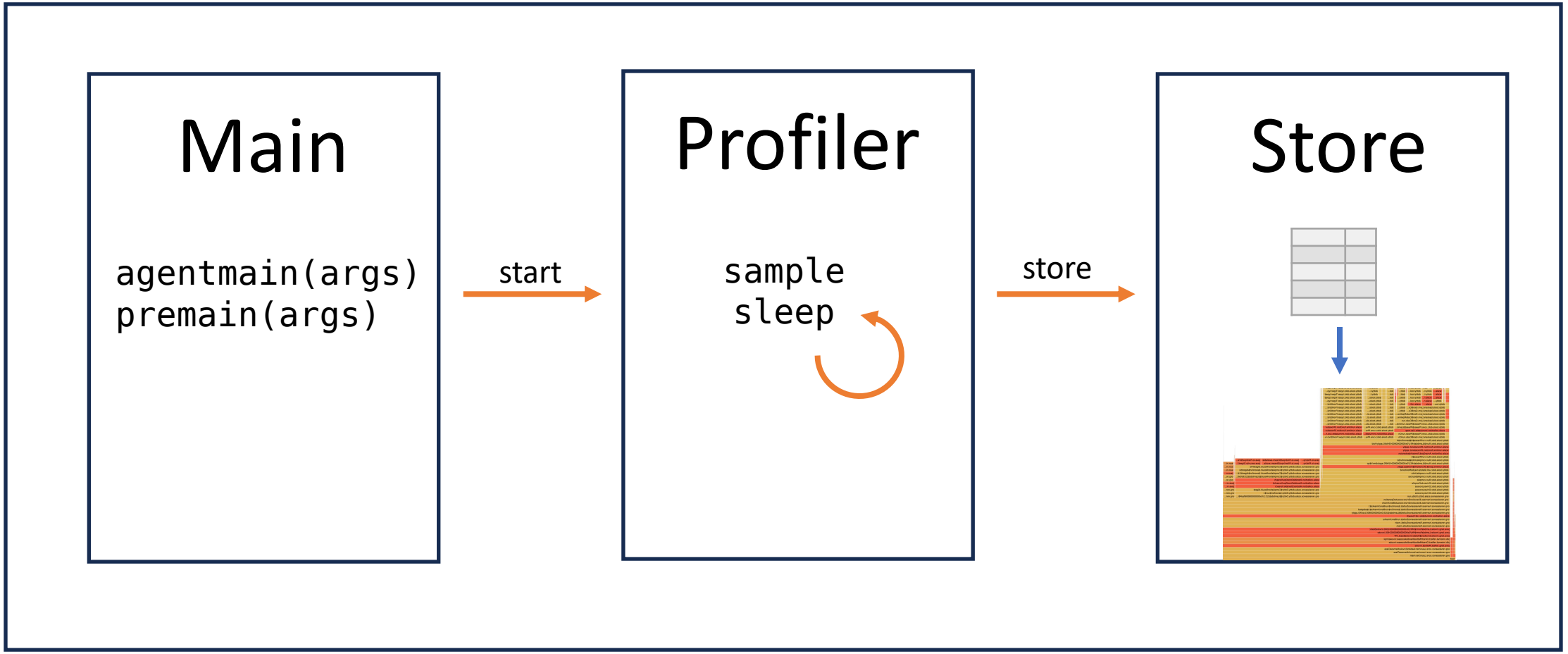


Writing a Profiler in 240 Lines of Pure Java

Posted on **March 27, 2023**

A few months back, I started writing a profiler from scratch, and the code since became the base of my profiler validation tools. The only problem with this project: I wanted to write a proper non-safepoint-biased profiler from scratch. This is a noble effort, but it requires lots C/C++/Unix programming which is finicky, and not everyone can read C/C++ code.

https://mostlynerdless.de


Main
agentmain(args)
premain(args)
start
Profiler
sample
sleep
store
Store


Main
agentmain(args)
premain(args)

Main Class

```
java -javaagent:./target/tiny_profiler.jar=agentArgs ...
```

Normally

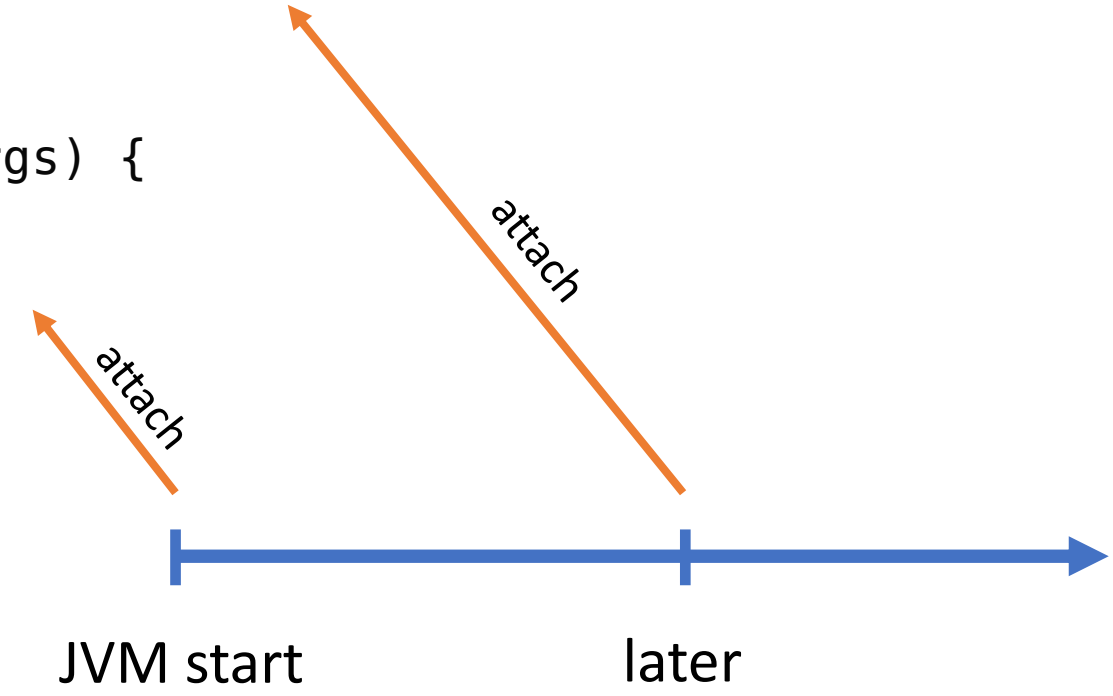
```
public static void main(String[] args) {
    ...
}

public static void main(String args[]) {
    ...
}

void main() {
    ...
}
```

Main Class

```
java -javaagent:./target/tiny_profiler.jar=agentArgs ...
```

```
public static void agentmain(String agentArgs) {
    premain(agentArgs);
}
```

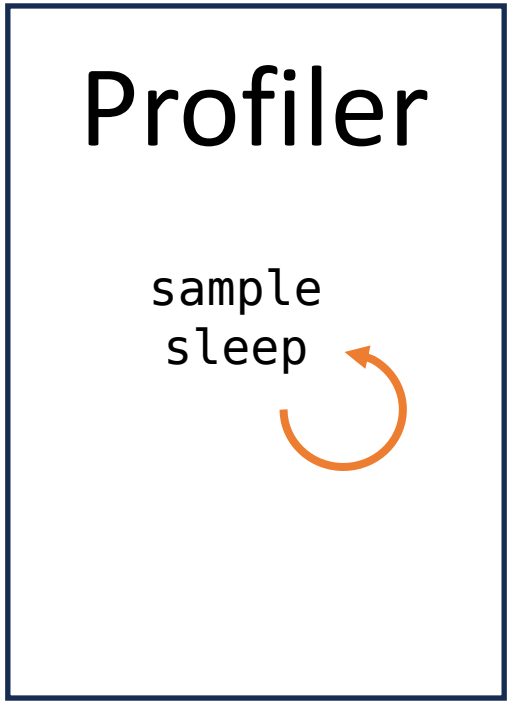
```
public static void premain(String agentArgs) {
    Main main = new Main();
    main.run(new Options(agentArgs));
}
```



Main class

```
private void run(Options options) {

    Thread t = new Thread(
        Profiler.newInstance(options));

    t.setDaemon(true);
    t.setName("Profiler");

    t.start();

}
```

```
public class Options {

    Duration interval;

    Optional<Path> flamePath;

    boolean printMethodTable;

}
```


Profiler
sample
sleep

Profiler

```
public static Profiler newInstance(Options options) {

    Profiler profiler = new Profiler(options);

    Runtime.getRuntime()
           .addShutdownHook(
                  new Thread(profiler::onEnd));

    return profiler;
}
```

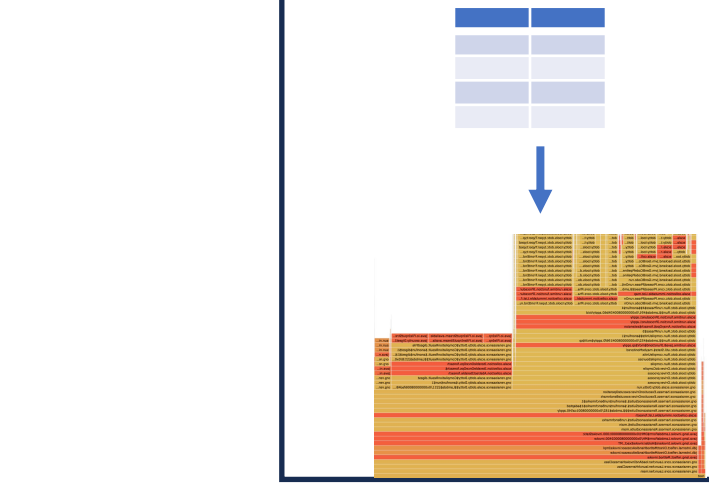
Profiler

```
private void sample() {
    Thread.getAllStackTraces()

      .forEach(
         (thread, st) -> {

            if (thread.isDaemon()) {
               return;
            }

            store.addSample(st);
         }

    });
}
```

StackTraceElement

```
class StackTraceElement {

    Class<?> declaringClassObject;

    String classLoaderName;
    String moduleName;
    String moduleVersion;

    String declaringClass;

    String methodName;
    String fileName;
    int    lineNumber;

}
```

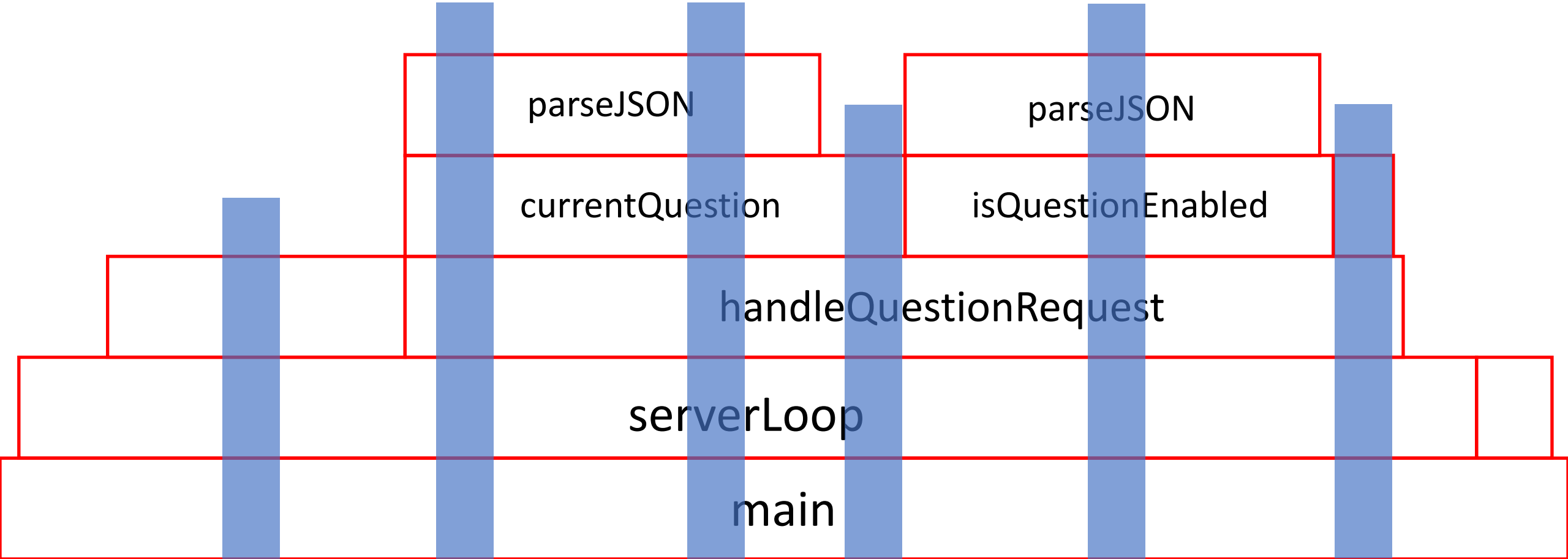
Store



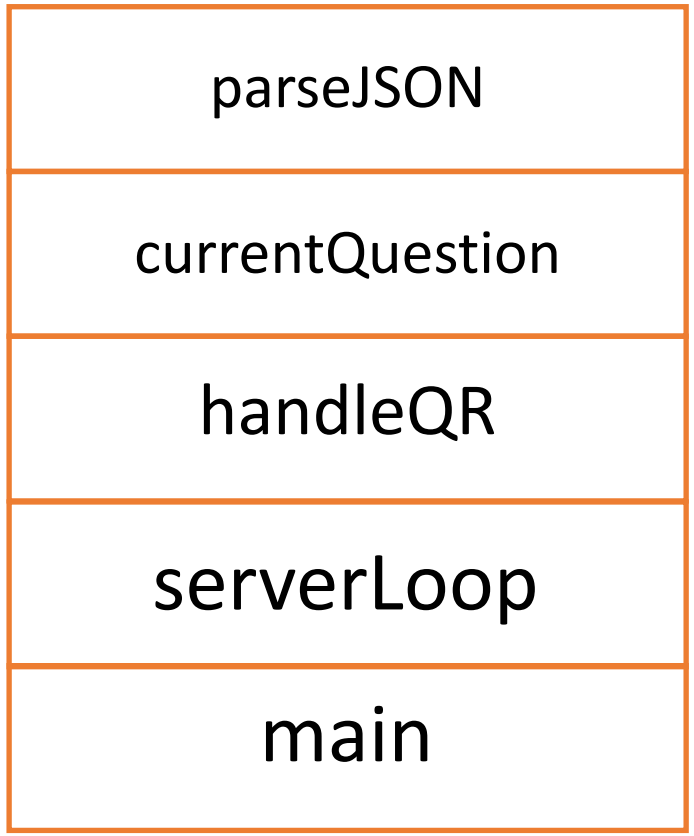
Store

```
class Store {

    Optional<Path> flamePath;

    Map<String, Long> methodOnTopSampleCount;
    Map<String, Long> methodSampleCount;
    long totalSampleCount = 0;

    Node rootNode = new Node("root");

    void addSample(StackTraceElement[] st)

    void printMethodTable()

    void storeFlameGraphIfNeeded()

}
```

Flamegraphs would be great

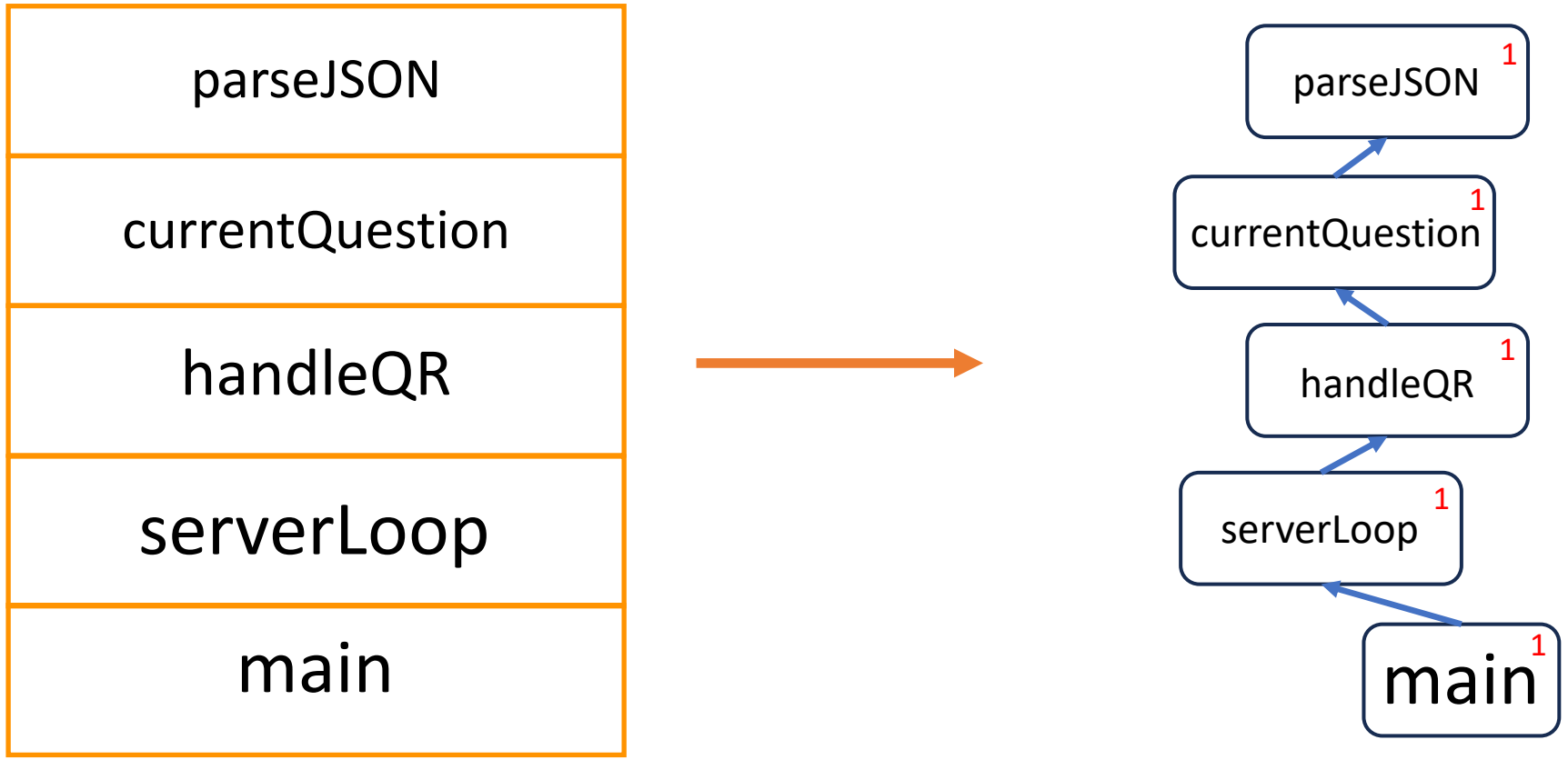


Turning traces into flamegraphs

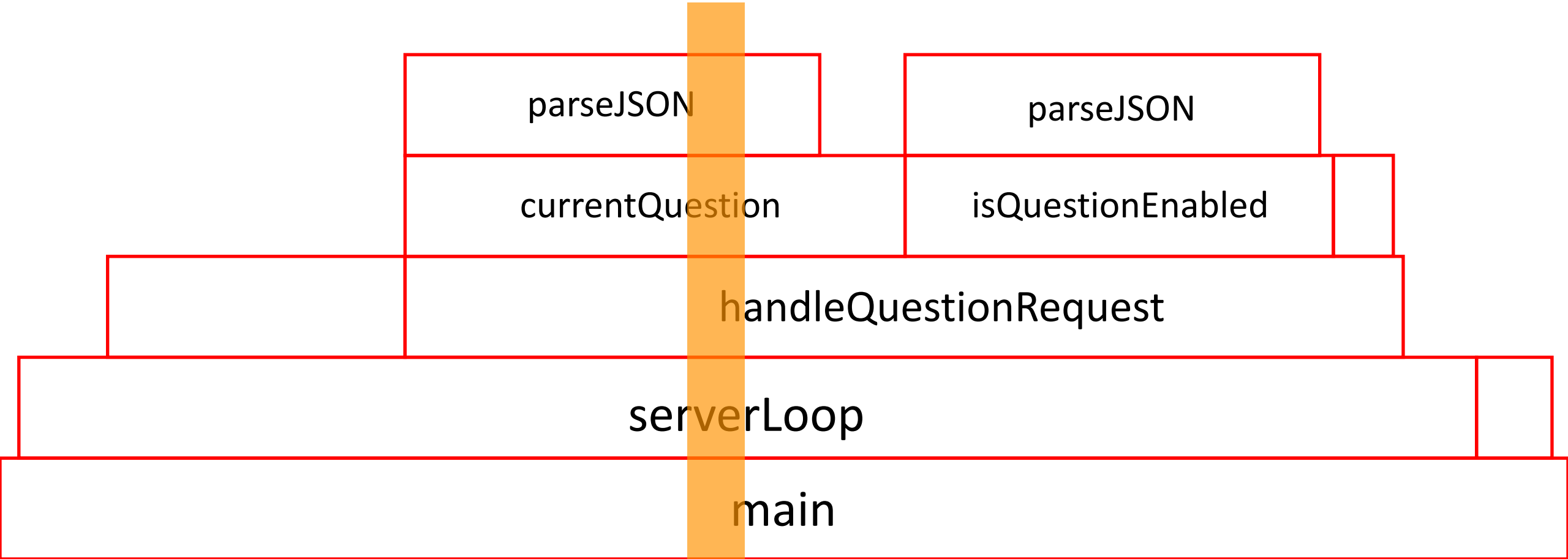



```
private static class Node {

    final String method;

    final Map<String, Node> children;
    long samples = 0;

    void addTrace(List<String> trace)

}
```

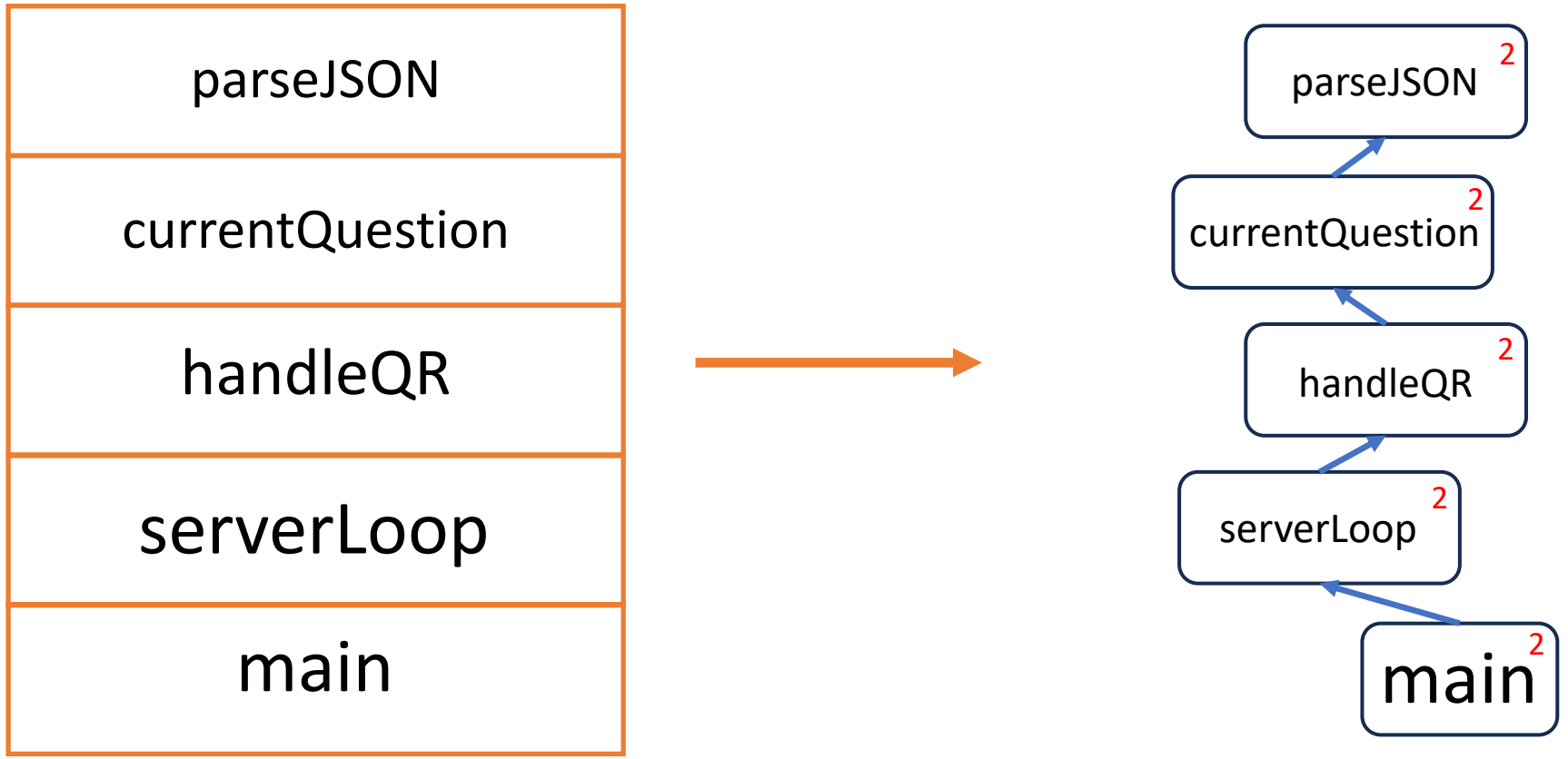
Turning traces into flamegraphs



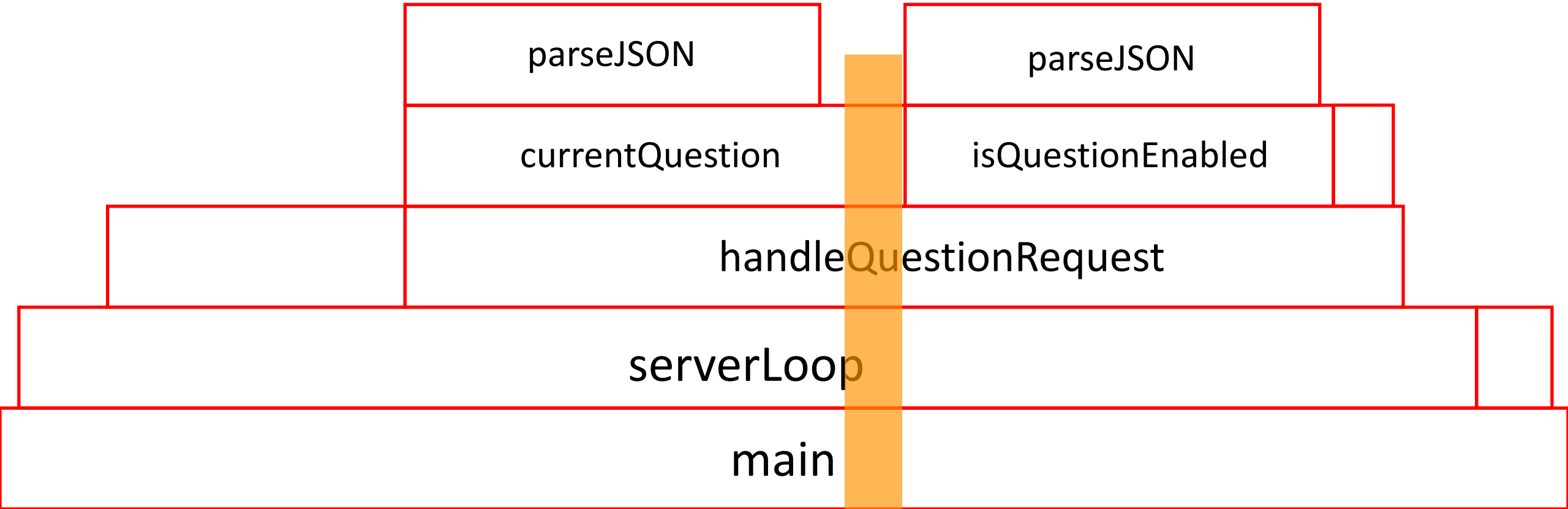
Flamegraphs would be great



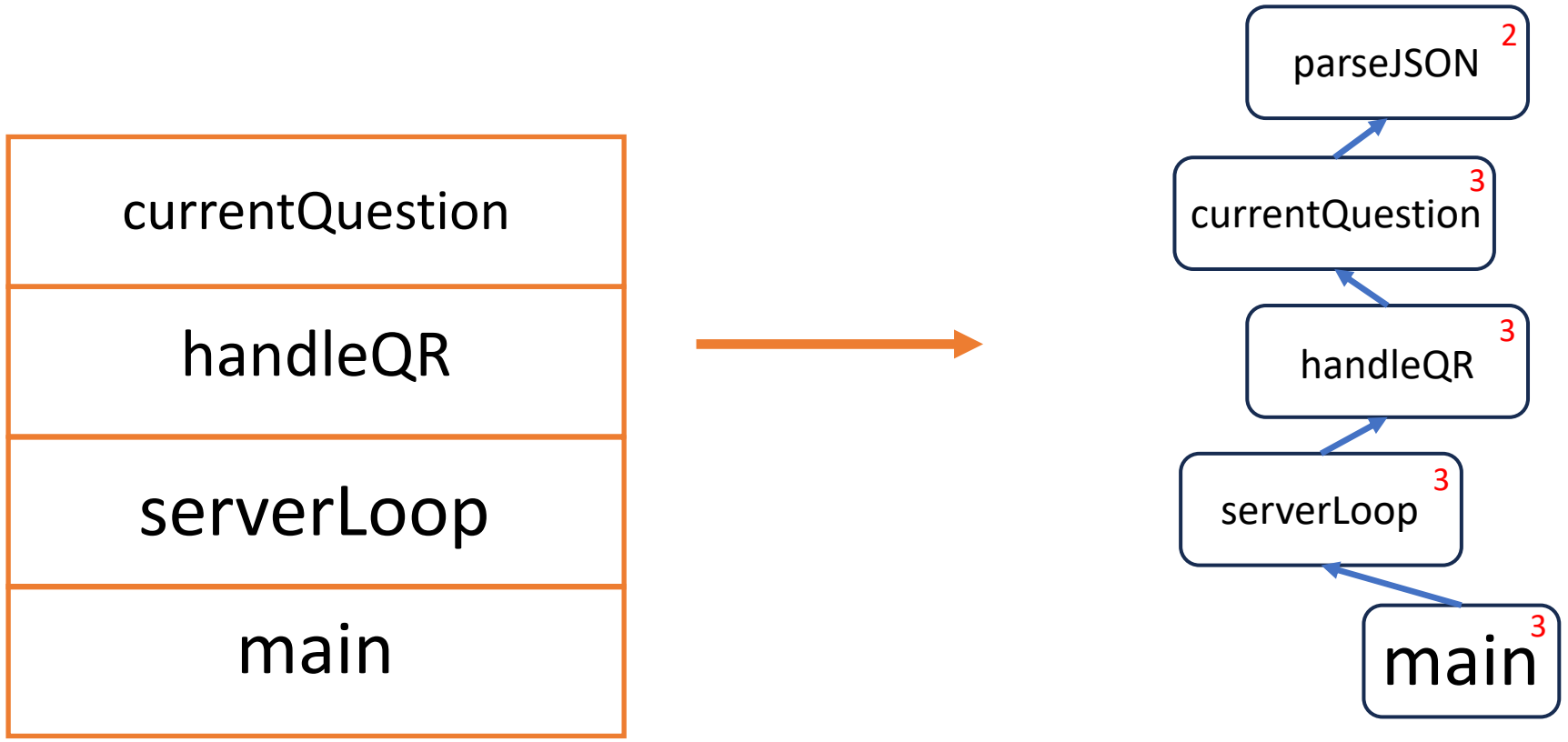
Turning traces into flamegraphs



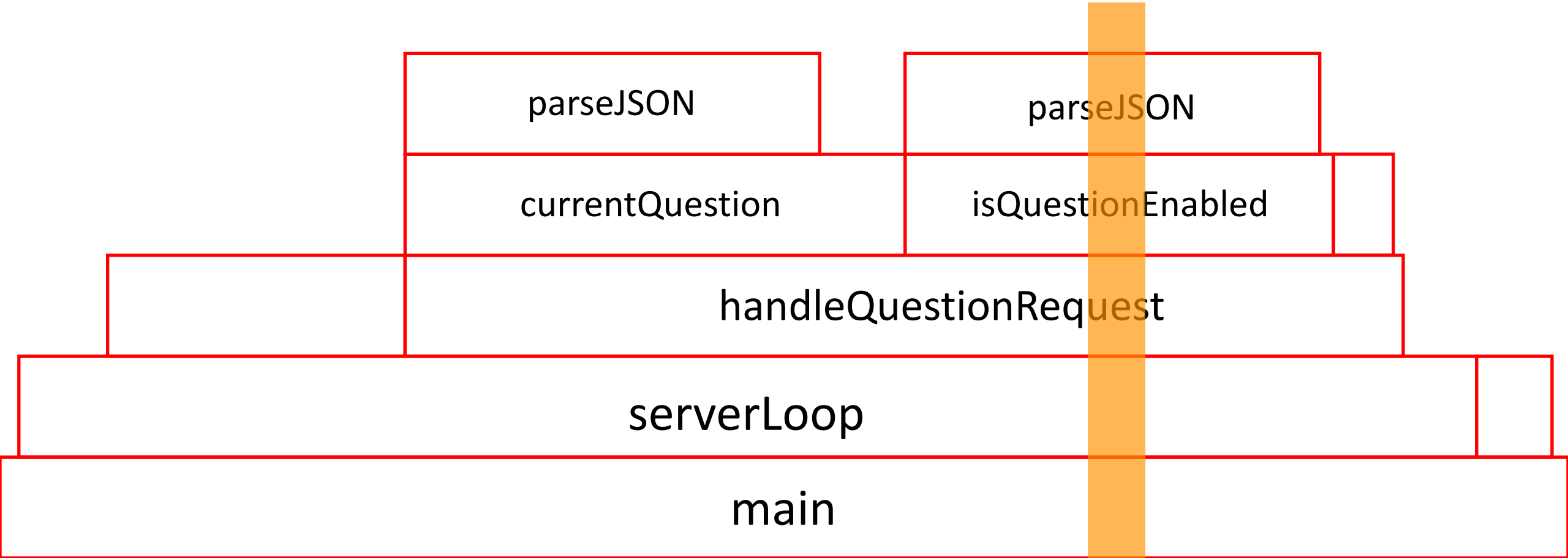
Flamegraphs would be great



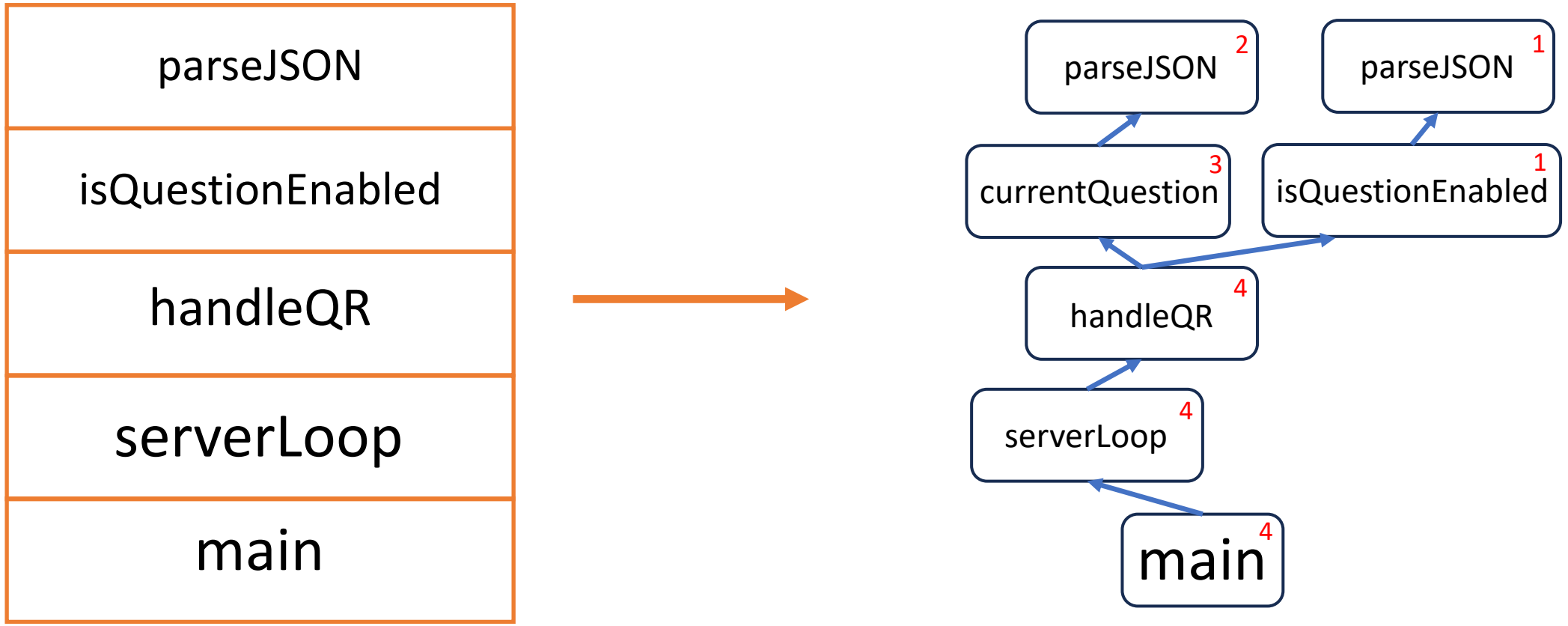
Turning traces into flamegraphs



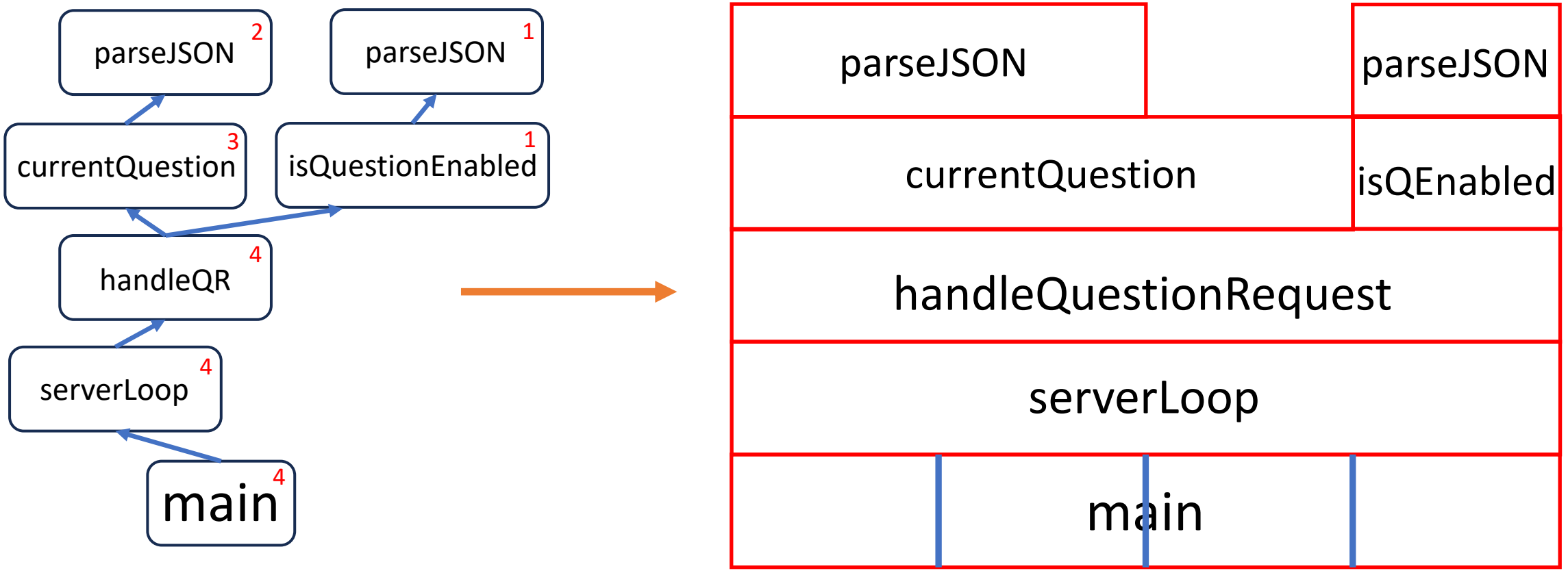
Flamegraphs would be great



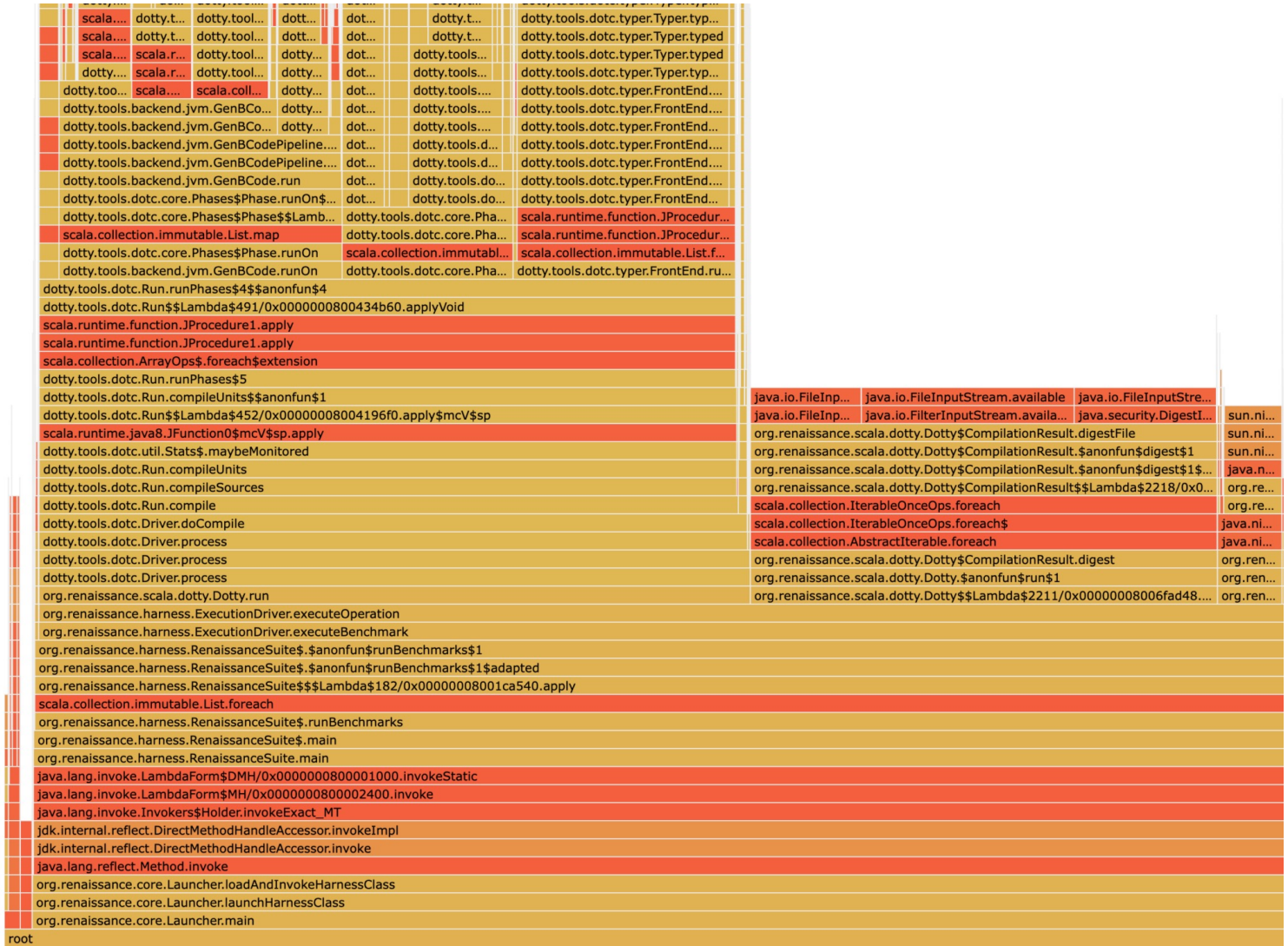
Turning traces into flamegraphs



Turning traces into flamegraphs

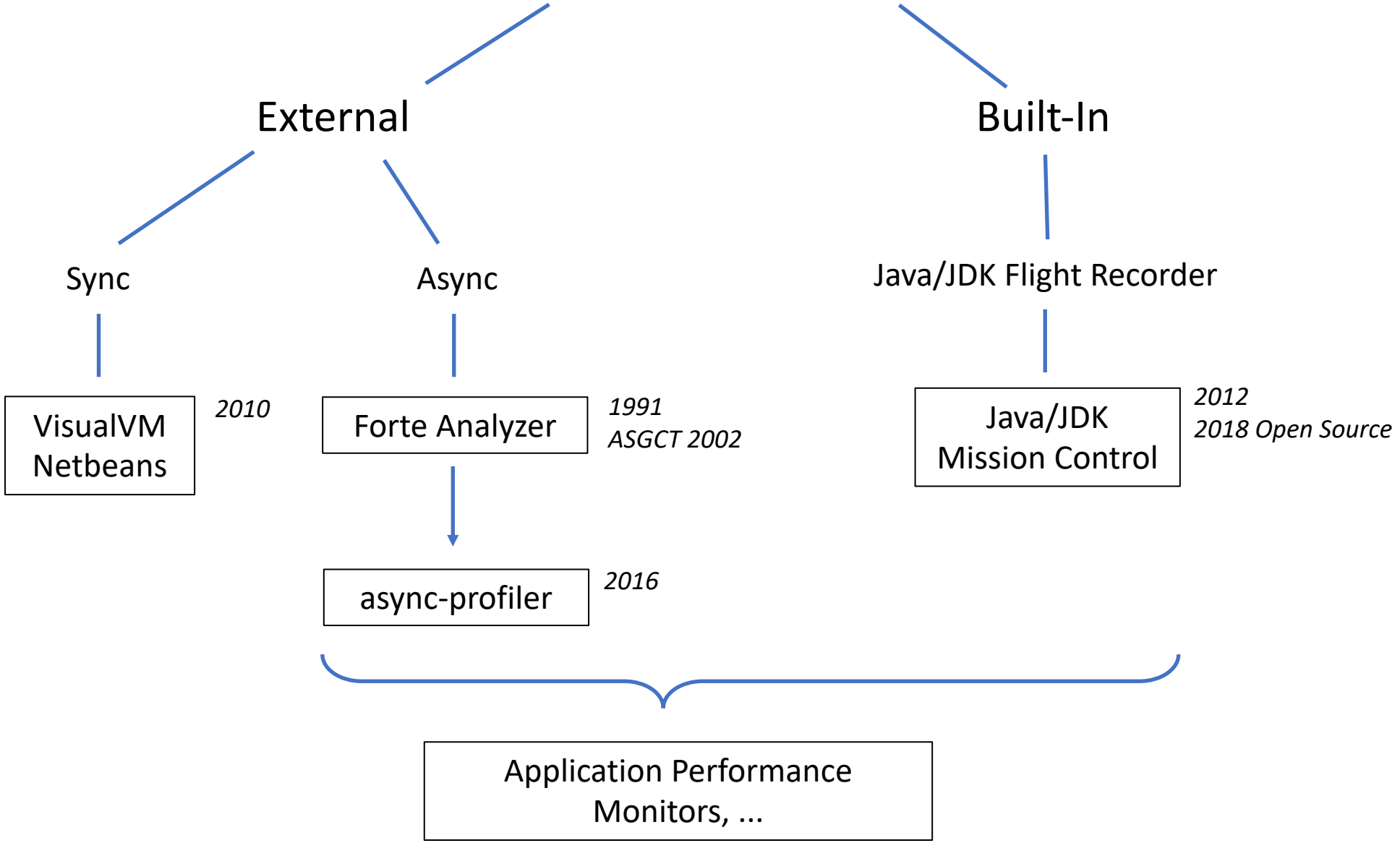


With the power of d3-flame-graph



Reality

Sampling Profiler



Obtaining a profile

Async Profiler

```
java \
    -agentpath:libasyncProfiler.so=start,\
     event=cpu,\
     file=flame.html,flamegraph \
    arguments
```

Download from GitHub

JDK Flight Recorder (JFR)

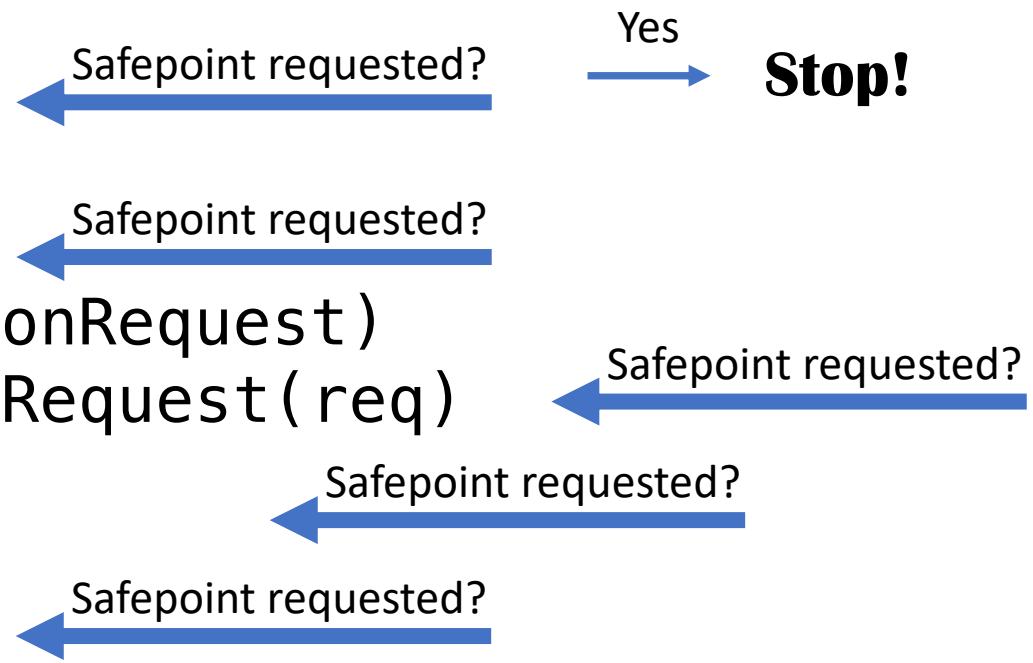
```
java \
  -XX:+UnlockDiagnosticVMOptions \
  -XX:+DebugNonSafepoints \
  -XX:+FlightRecorder \
  -XX:StartFlightRecording=filename=file.jfr \
  arguments
```

Already included in your JDK 8+

Don’t trust them

Safepoint

```
fun serverLoop() {

  while (true) {
    req = ...
    if (req.isQuestionRequest)
      handleQuestionRequest(req)
    ...
  }

}
```



Safepoint

Safepoint bias

using Safepoints

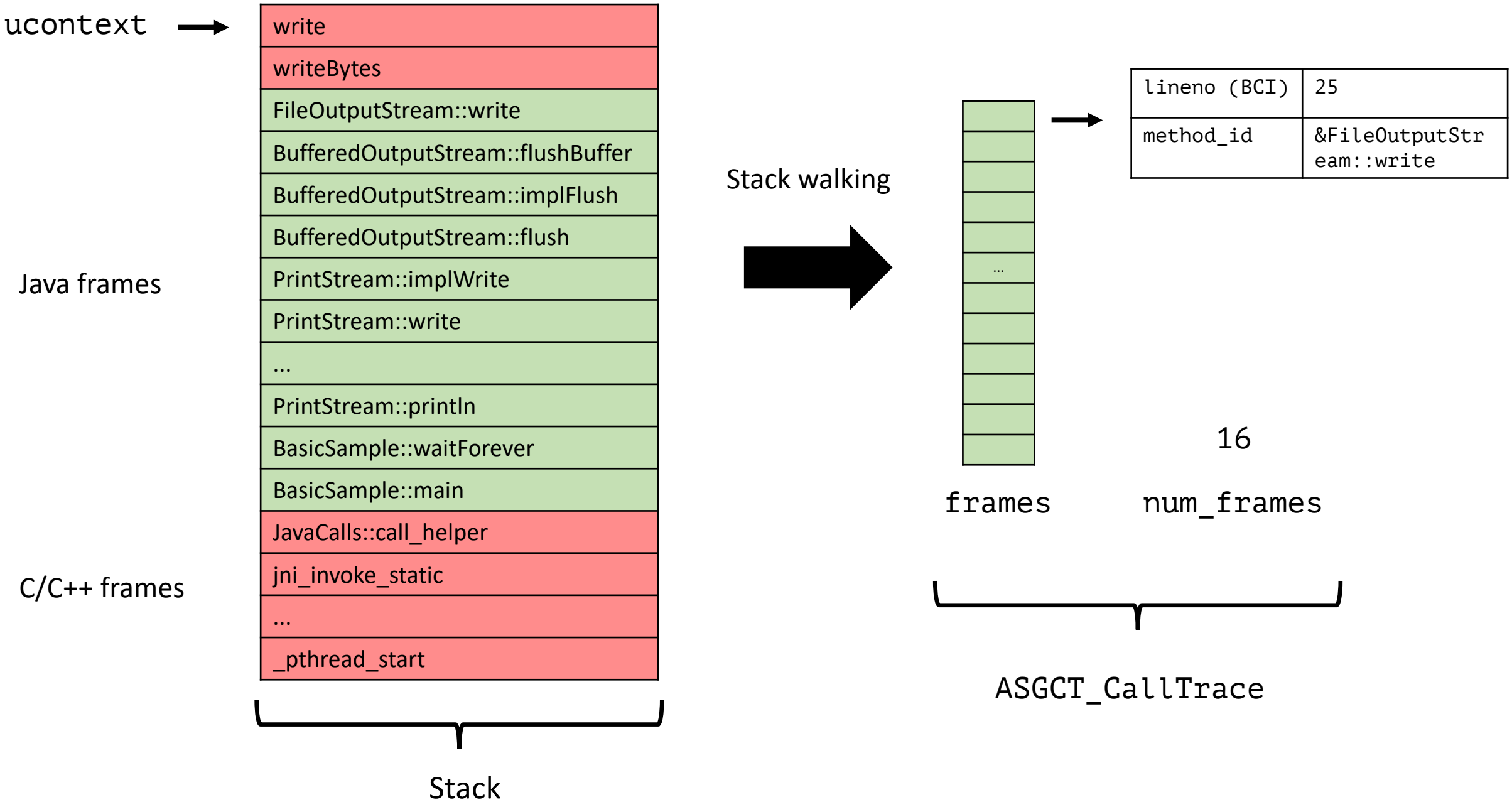
fully asynchronous

Safepoint bias with multiple threads

using Safepoints

fully asynchronous

AsyncGetCallTrace


ucontext
write
writeBytes
FileOutputStream::write
BufferedOutputStream::flushBuffer
BufferedOutputStream::implFlush
BufferedOutputStream::flush
PrintStream::implWrite
PrintStream::write
...
PrintStream::println
BasicSample::waitForever
BasicSample::main
JavaCalls::call_helper
jni_invoke_static
...
_pthread_start
Java frames
C/C++ frames
Stack
Stack walking
...
lineno (BCI)
25
method_id
&FileOutputStream::write
16
frames
num_frames
ASGCT_CallTrace

Tests in the OpenJDK?

1

But its a good one?

```
public class ASGCTBaseTest {
    // ...
    public static void main(String[] args) {
        if (!check()) {
            throw ...;
        }
    }

    private static native boolean check();
}
```

checked just this method

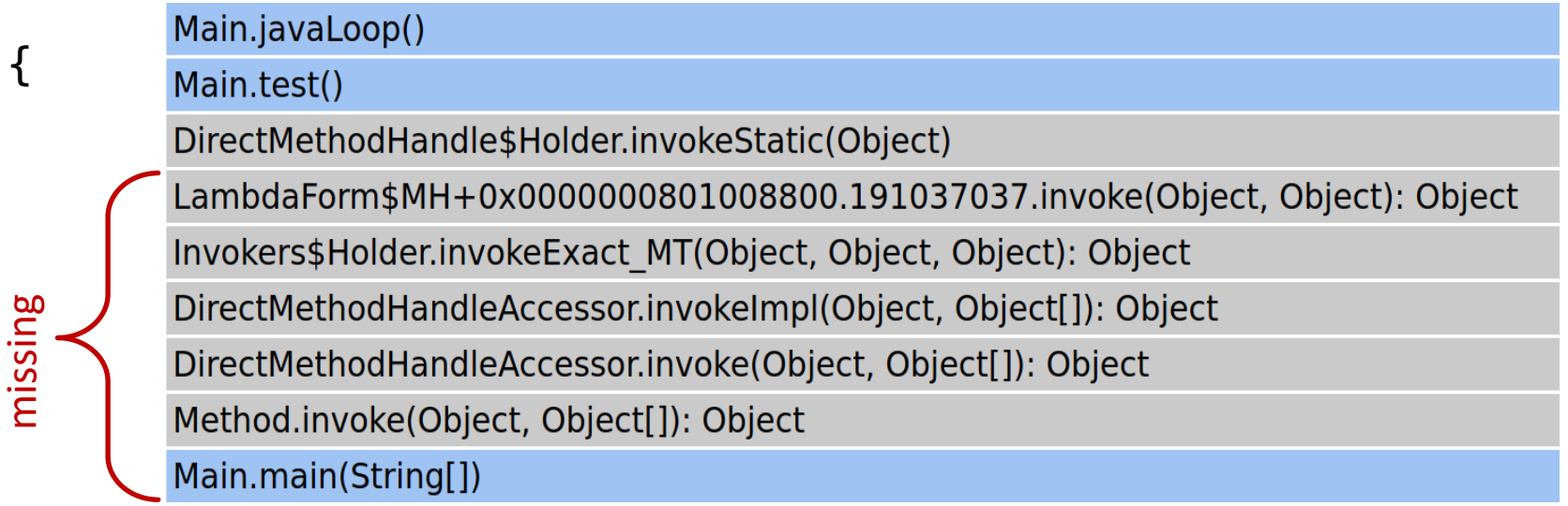
https://github.com/openjdk/jdk/blob/7d25bf7722f6fbe3633dc718adf6f755e354adb9/test/hotspot/jtreg/serviceability/AsyncGetCallTrace/MyPackage/ASGCTBaseTest.java

Take profiles with a grain of salt

Profilers have bugs, e.g. JDK-8302320

```
public class Main {
  public static void main(String[] args) ... {
    Class<?> klass = Main.class;
    Method mainMethod = klass.getMethod("test");
    mainMethod.invoke(null);
  }
  static void test() {
    javaLoop();
  }
  static void javaLoop() {
     /* endless loop */
  }
}
```

Flamegraph



But surely OpenJ9 is better?

0

Tests for AsyncGetCallTrace

Or GetStackTrace?

2

Safepoint biased is bad

```java
@Benchmark
public void blameSetResult() {
    byte b = 0;
    for (int i = 0; i < size; i++) {
        b += buffer[i];
    }
    setResult(b); /* LINE 38 */
}

private void setResult(byte b) {
    setResult(b == 1); /* LINE 90 */
}

@CompilerControl(CompilerControl.Mode.DONT_INLINE)
private void setResult(boolean b) {
    result = b;
}
```

http://psy-lob-saw.blogspot.com/2016/02/why-most-sampling-java-profilers-are.html

Profilers are just software

Tests could better

> 1

Performance and accuracy could be better

Could be safer*

*crashes are rare in practice

Not designed with safety in mind

8284828: Use `os::ThreadCrashProtection` to protect AsyncGetCallTrace from crashing #8225

Closed parttimenerd wants to merge 9 commits into `openjdk:master` from `parttimenerd:parttimenerd_8284828`

> I should add that the CrashProtection mechanism was mainly put in place as a result of having to deliver JFR from JRockit into Hotspot under a deadline, upholding feature-parity. The stack walking code was in really bad shape back then.
>
> – Markus Grönlund

> AGCT is a legacy mechanism that was created for one single unsupported purpose
>
> – David Holmes

TDD

Test Driven Development


TDD

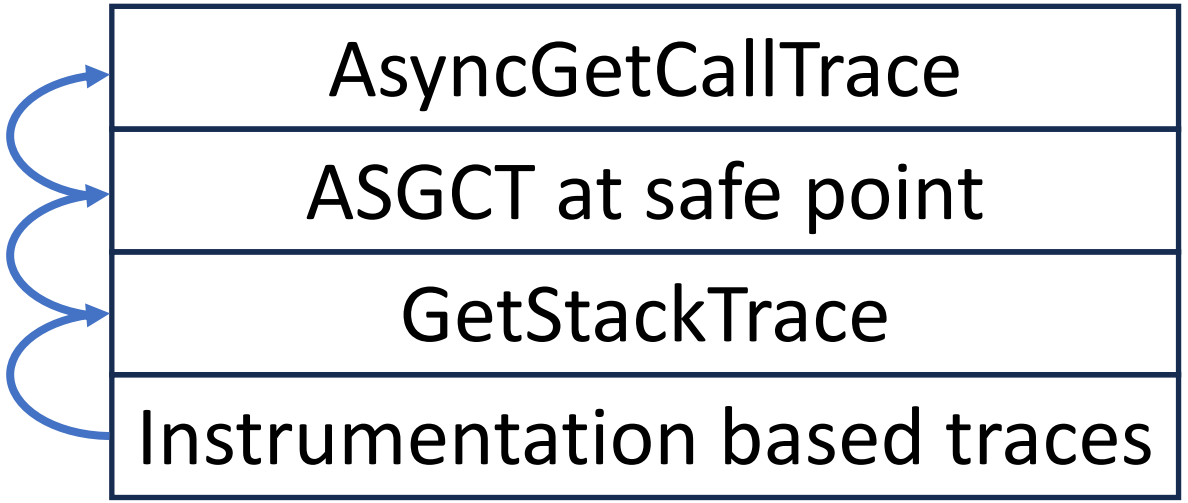
RDD

Rage Driven Development


More Tests

Testing Approaches

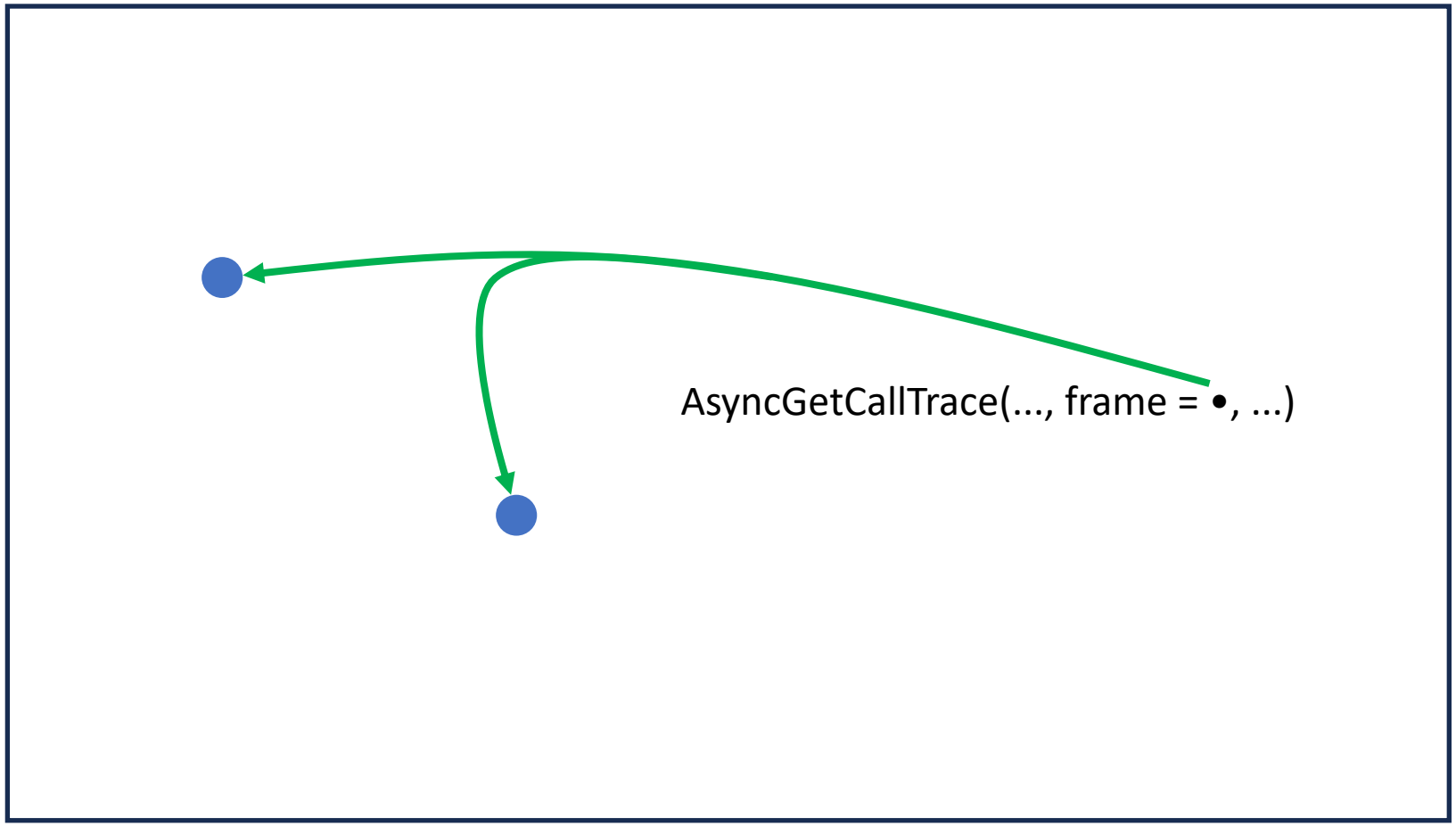
Layered Oracle



checks

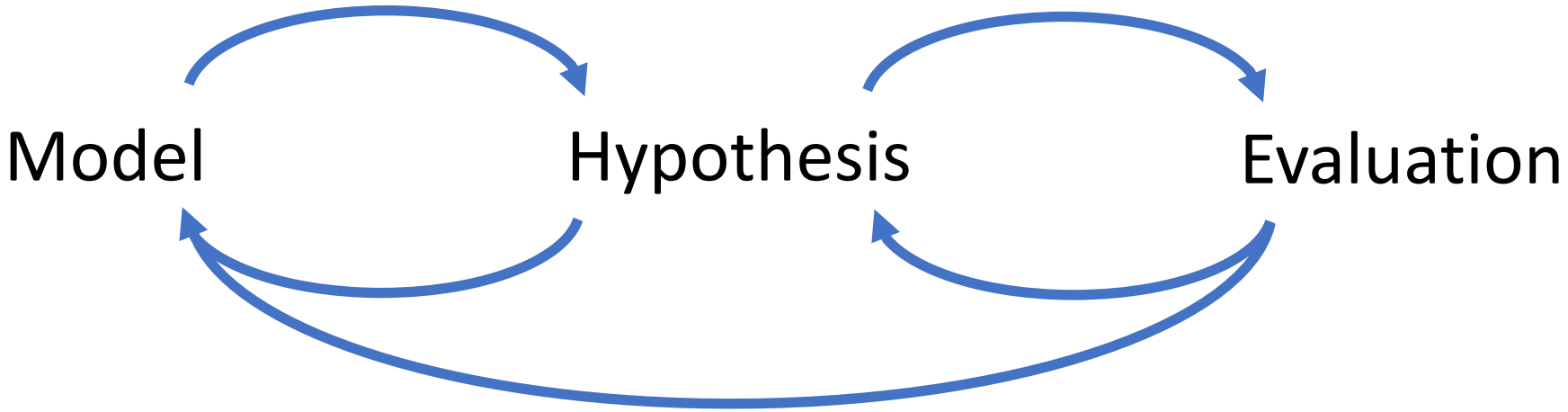
```
fun serverLoop() {
  logEntry(“serverLoop”)
  while (true) {
    req = ...
    if (req.isQuestionRequest)
      handleQuestionRequest(req)
    ...
  }
  logExit(“serverLoop”)
}
```

Fuzzing for safety



Experimentation technique

Profiling Loop



Profilers are great

But don’t trust them too much

@mostlynerdless.de on BlueSky
parttimenerd on GitHub
mostlynerdless.de

@SweetSapMachine on Twitter
sapmachine.io