POWERED BY Skywalking APACHE

2023 SUMMIT | APACHE • SkyWalking
CHINA · SHANGHAI

2023 · 上海

SkyWalking Summit

纵目 tetrate

演讲主题

陆家靖

收钱吧框架工具团队负责人、 SkyWalking PMC Member

“基于SkyWalking Agent的持续性能剖析与交互式诊断 ”

目录

CONTENTS

01. 持续性能剖析
Continuous Profiling

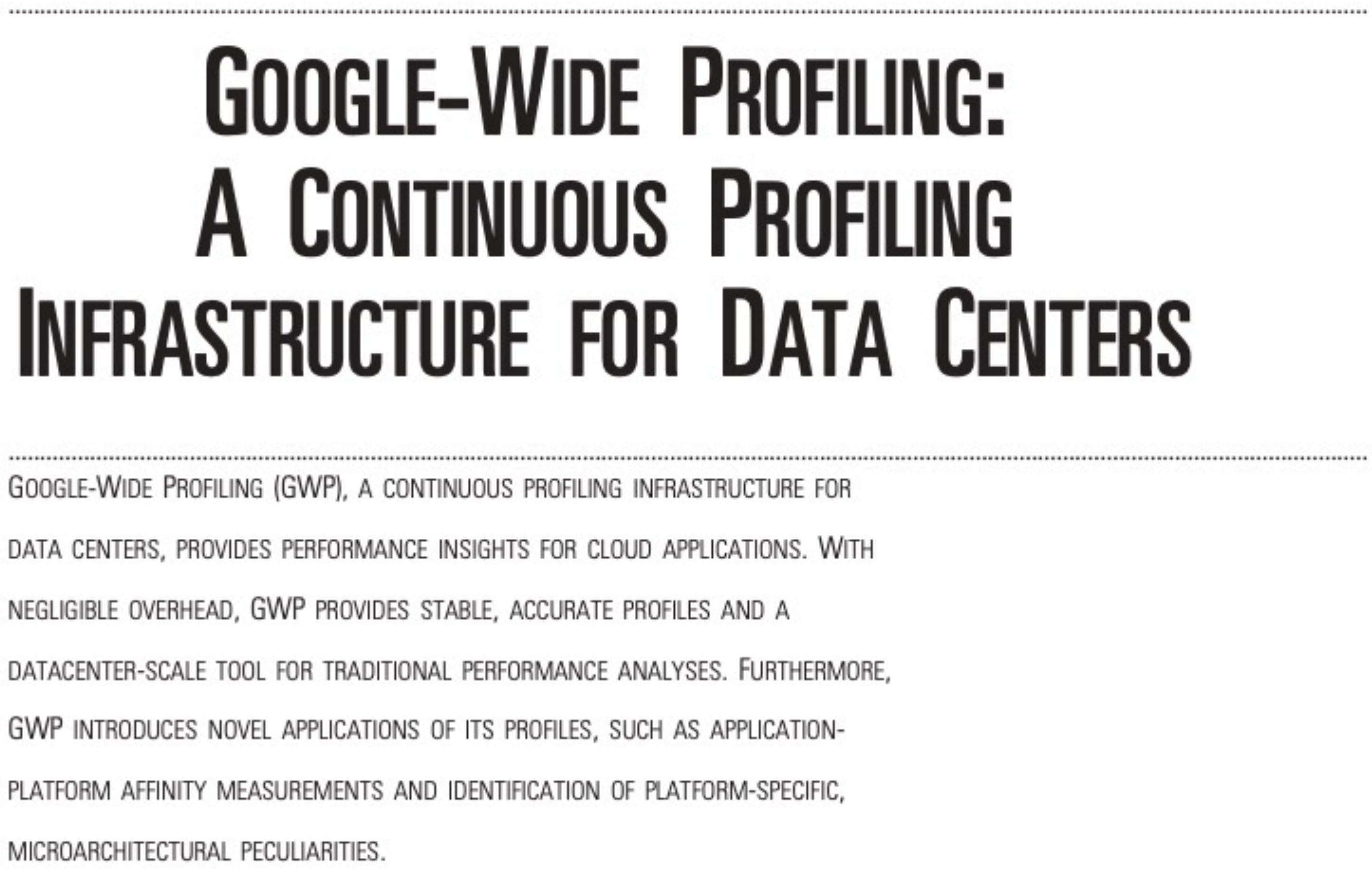
02. 交互式诊断
Interactive Diagnosis

01

持续性能剖析

Continuous Profiling

Continuous Profiling的发展史

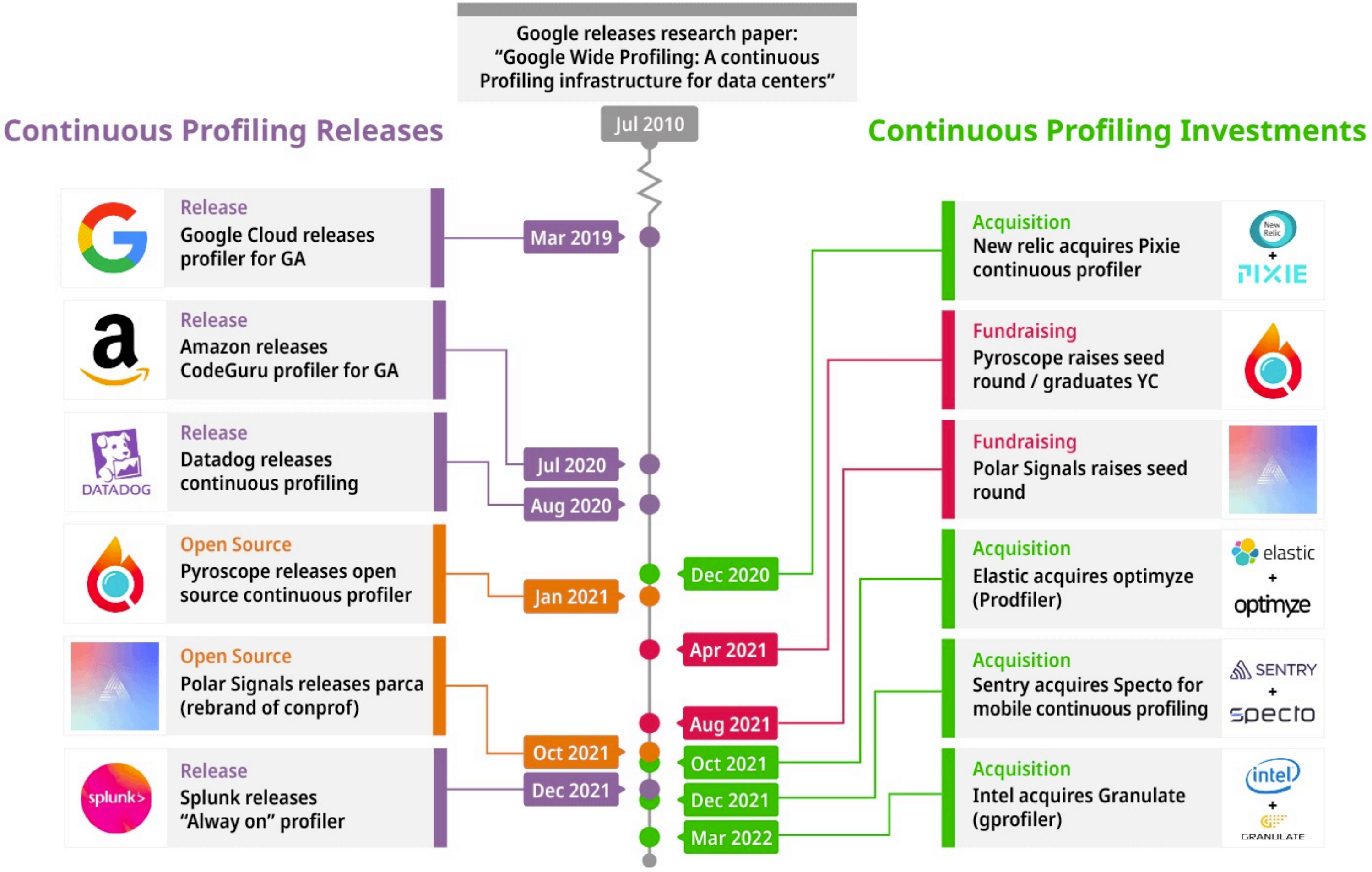


Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers

Google-Wide Profiling (GWP), a continuous profiling infrastructure for data centers, provides performance insights for cloud applications. With negligible overhead, GWP provides stable, accurate profiles and a datacenter-scale tool for traditional performance analyses. Furthermore, GWP introduces novel applications of its profiles, such as application-platform affinity measurements and identification of platform-specific, microarchitectural peculiarities.

FIG. GWP published by Google in 2010：low overhead, stable, accurate, scalable

Continuous Profiling的发展史



Google releases research paper: “Google Wide Profiling: A continuous Profiling infrastructure for data centers”

Jul 2010

Continuous Profiling Releases

- Mar 2019 — Release: Google Cloud releases profiler for GA
- Jul 2020 — Release: Amazon releases CodeGuru profiler for GA
- Aug 2020 — Release: Datadog releases continuous profiling
- Jan 2021 — Open Source: Pyroscope releases open source continuous profiler
- Oct 2021 — Open Source: Polar Signals releases parca (rebrand of conprof)
- Dec 2021 — Release: Splunk releases “Alway on” profiler

Continuous Profiling Investments

- Dec 2020 — Acquisition: New relic acquires Pixie continuous profiler
- Apr 2021 — Fundraising: Pyroscope raises seed round / graduates YC
- Aug 2021 — Fundraising: Polar Signals raises seed round
- Oct 2021 — Acquisition: Elastic acquires optimyze (Prodfiler)
- Dec 2021 — Acquisition: Sentry acquires Specto for mobile continuous profiling
- Mar 2022 — Acquisition: Intel acquires Granulate (gprofiler)

FIG. Since GWP， many major vendors have joined “Continuous Profiling” ：
Pyroscope is an open-source solution， acquired by Grafana Lab on 2023-03-15

Grafana Pyroscope

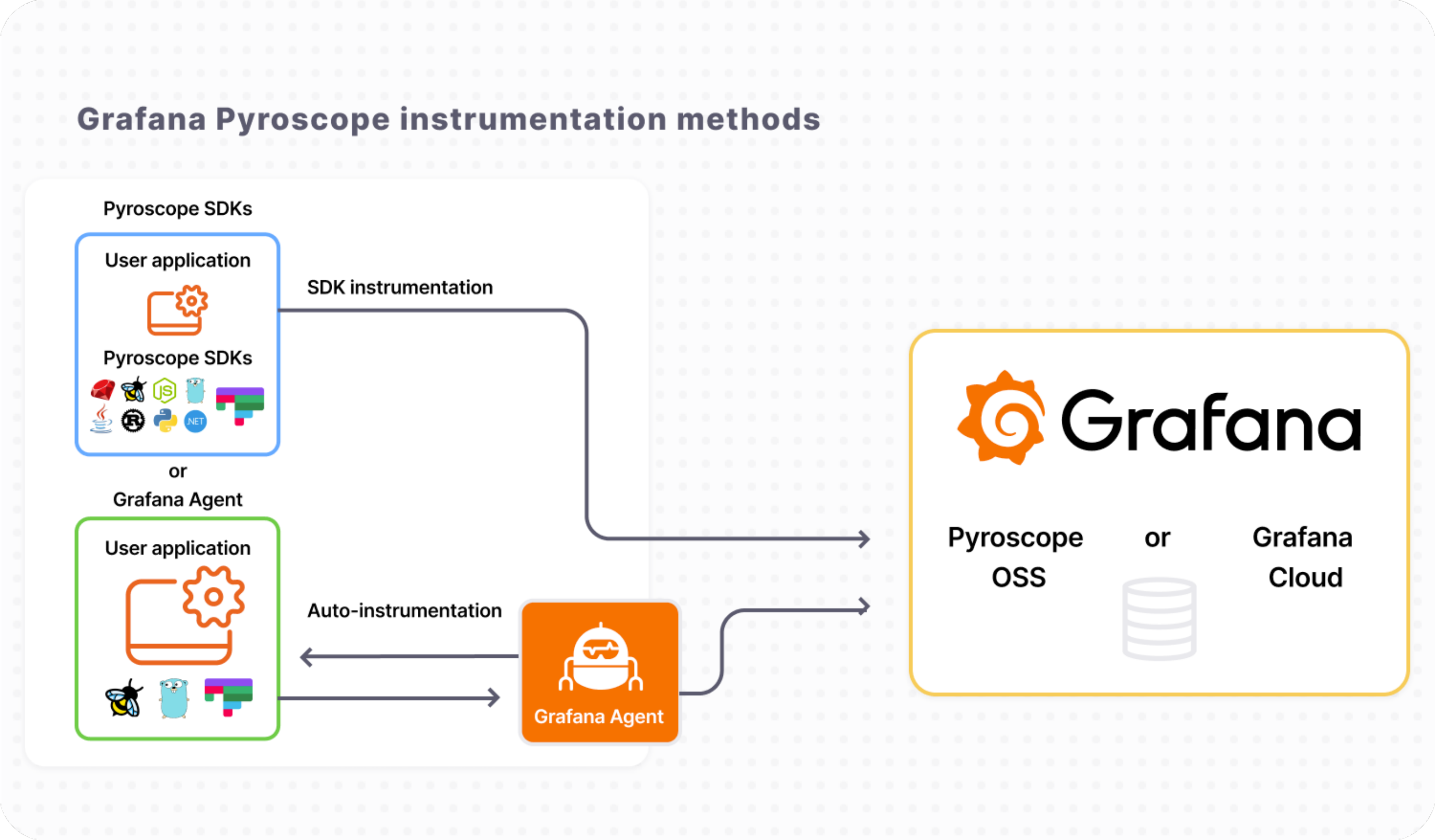
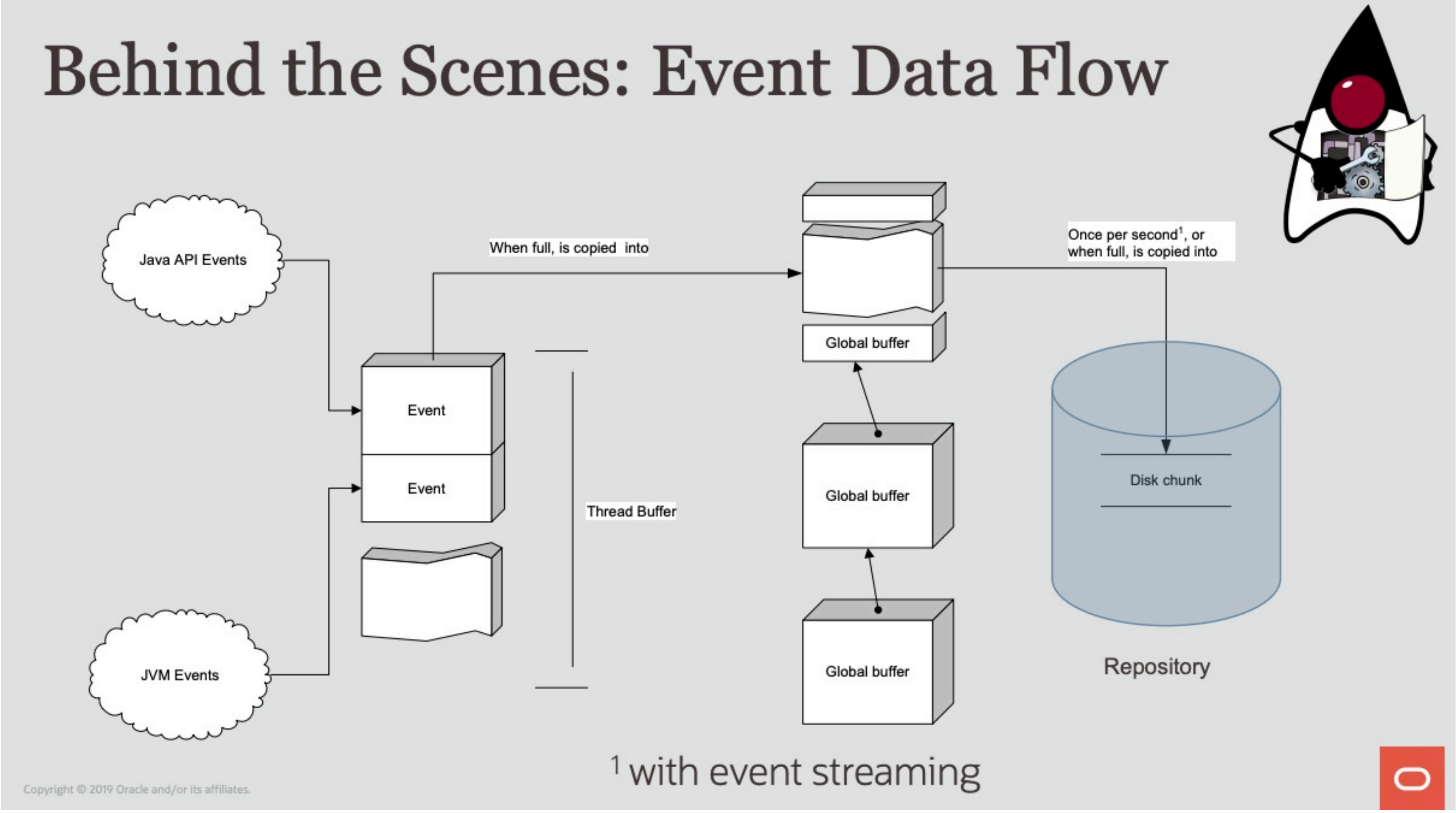


FIG. Architecture of the Grafana Pyroscope

Java: How to collect? Java Flight Recorder

- Capture both JVM and application data
  - GC
  - Synchronization
  - Compiler
  - CPU usage
  - Exceptions
  - I/O
- Sampling-based profiler
  - Very low overhead: 2-3%
- Buffers
  - Thread Buffer
  - Global Buffer
  - Repository (Disk chunk)

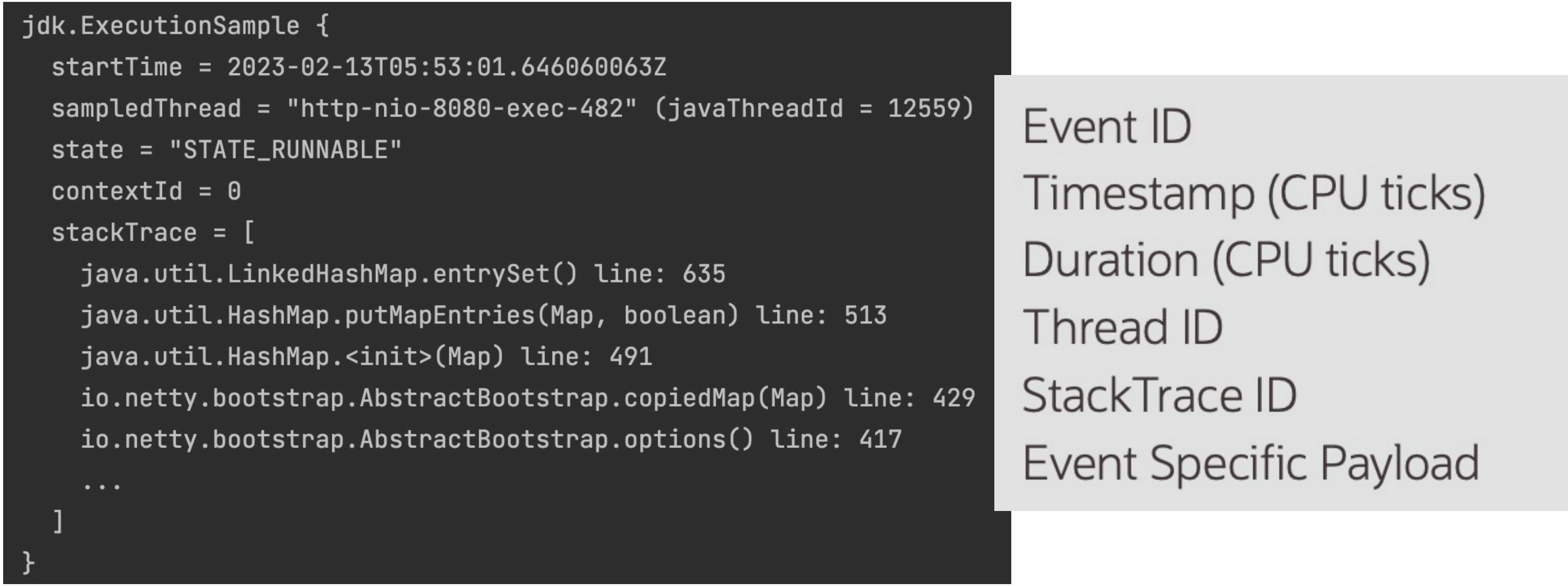


Behind the Scenes: Event Data Flow

[1] with event streaming

FIG. How JFR works in the background: API events and JVM events as sources. https://www.infoq.com/presentations/monitoring-jdk-jfr

Java: How to collect? Java Flight Recorder

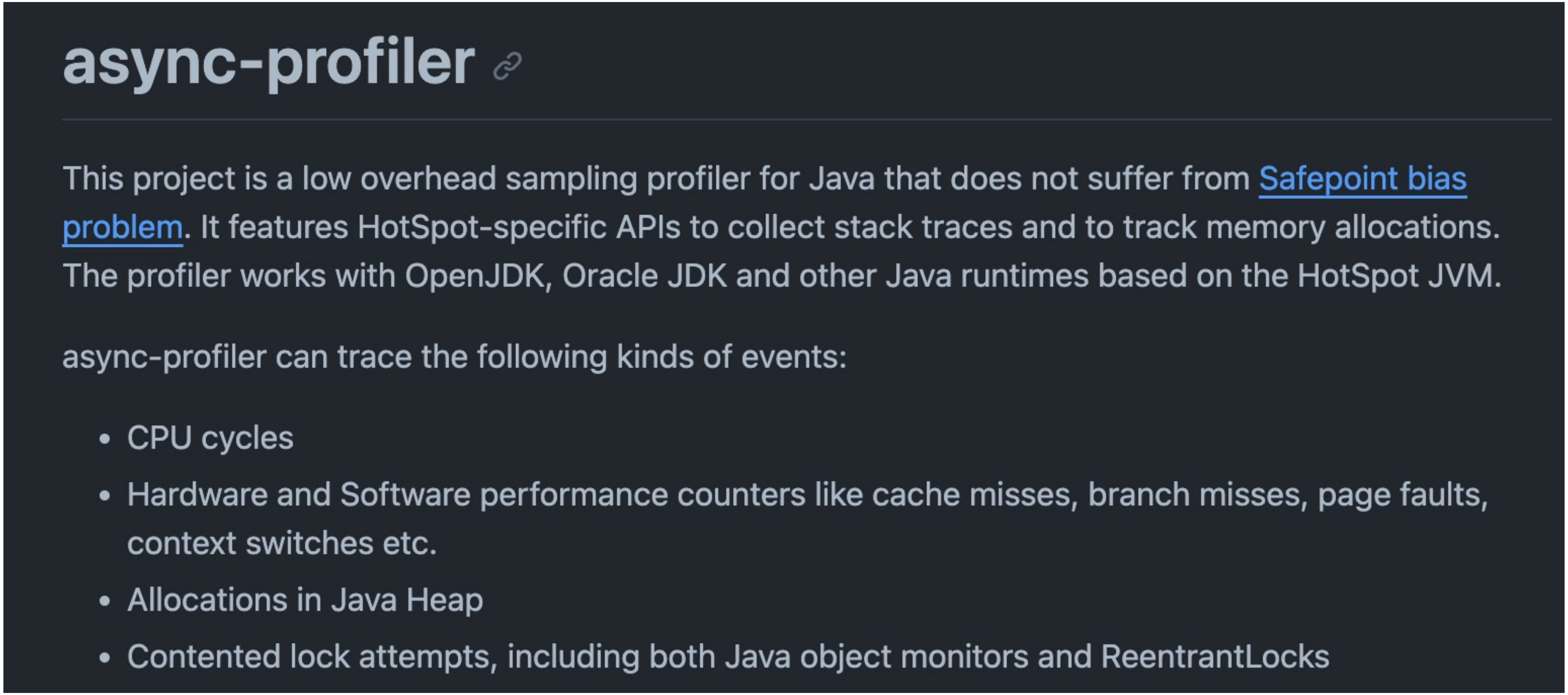


```
jdk.ExecutionSample {
  startTime = 2023-02-13T05:53:01.646060063Z
  sampledThread = "http-nio-8080-exec-482" (javaThreadId = 12559)
  state = "STATE_RUNNABLE"
  contextId = 0
  stackTrace = [
    java.util.LinkedHashMap.entrySet() line: 635
    java.util.HashMap.putMapEntries(Map, boolean) line: 513
    java.util.HashMap.<init>(Map) line: 491
    io.netty.bootstrap.AbstractBootstrap.copiedMap(Map) line: 429
    io.netty.bootstrap.AbstractBootstrap.options() line: 417
    ...
  ]
}
```

Event ID
Timestamp (CPU ticks)
Duration (CPU ticks)
Thread ID
StackTrace ID
Event Specific Payload

FIG. The anatomy of a JFR event and a typical example

Java: How to collect? Async Profiler



async-profiler

This project is a low overhead sampling profiler for Java that does not suffer from Safepoint bias problem. It features HotSpot-specific APIs to collect stack traces and to track memory allocations. The profiler works with OpenJDK, Oracle JDK and other Java runtimes based on the HotSpot JVM.

async-profiler can trace the following kinds of events:

- CPU cycles
- Hardware and Software performance counters like cache misses, branch misses, page faults, context switches etc.
- Allocations in Java Heap
- Contented lock attempts, including both Java object monitors and ReentrantLocks

Java: How to analyze? FlameGraph

https://github.com/brendangregg/FlameGraph

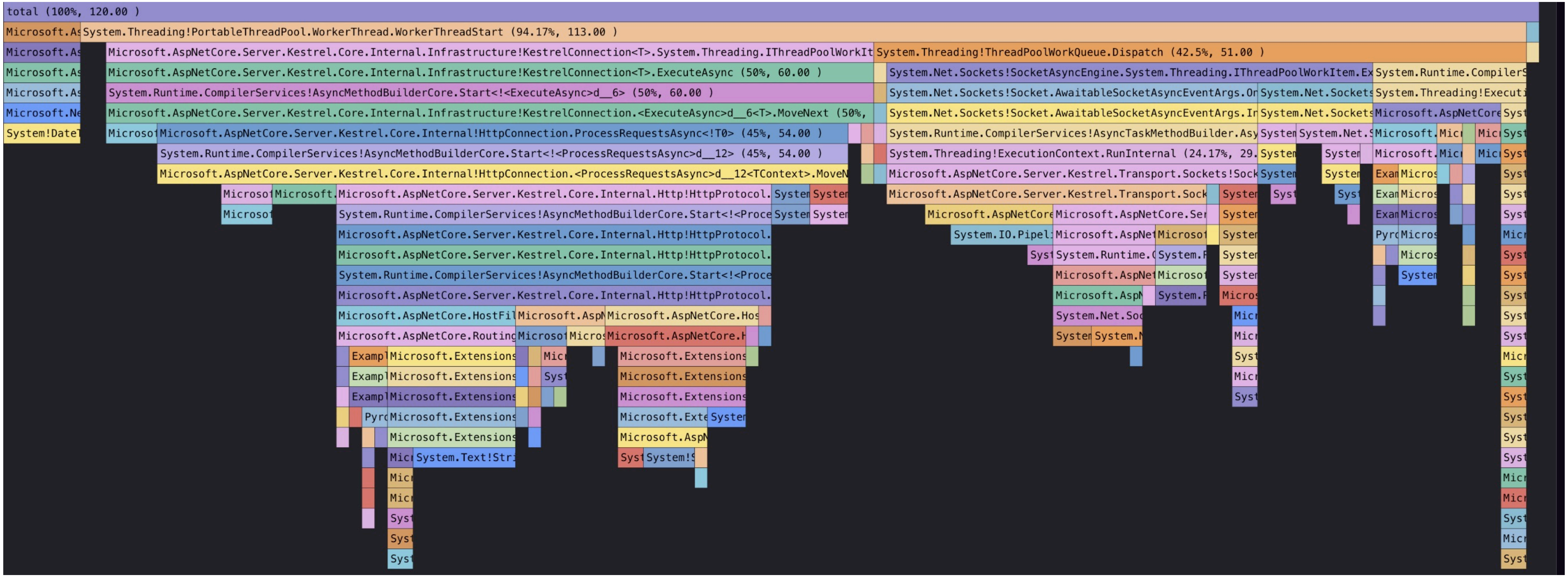
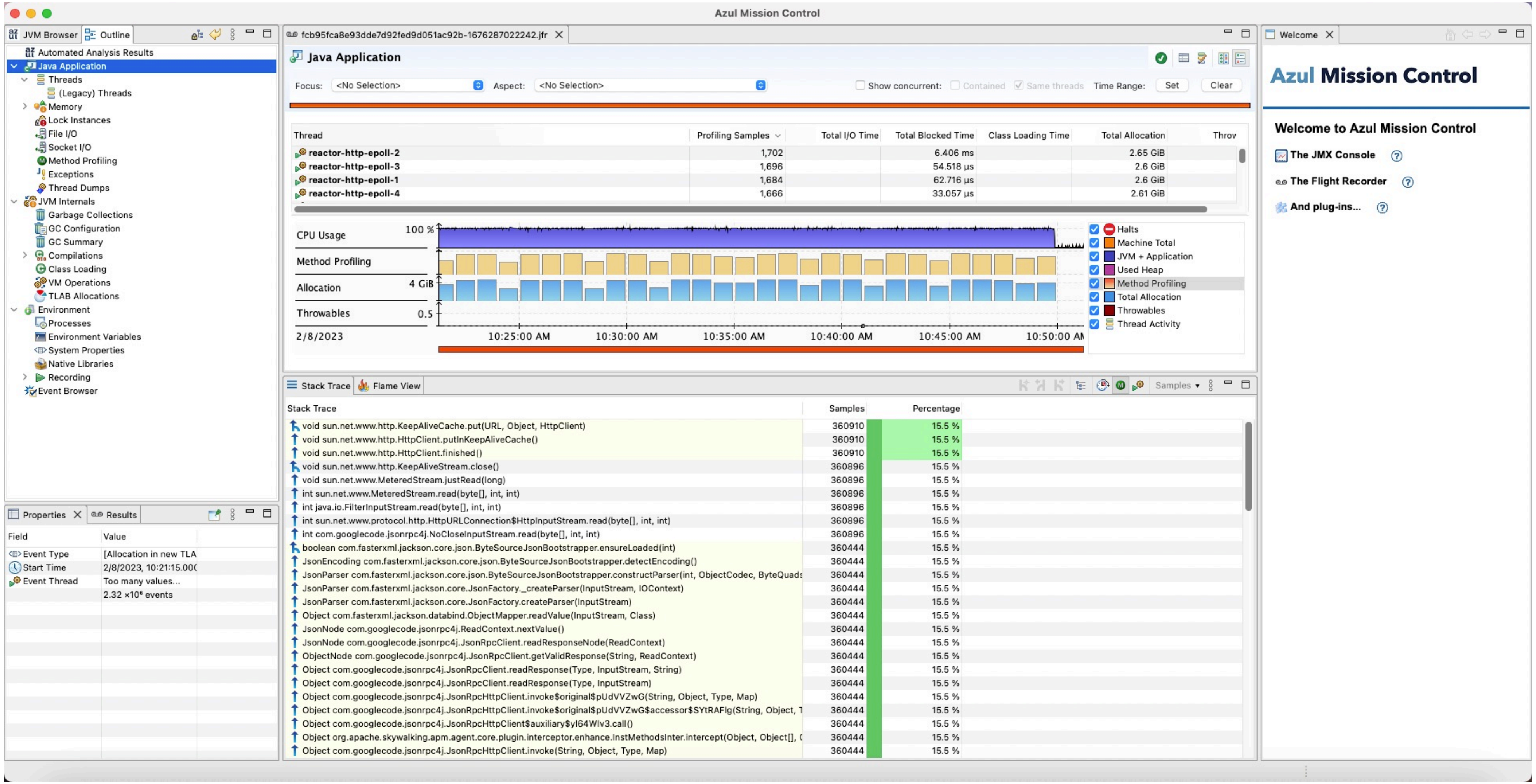


FIG. A typical flamegraph

Java: How to analyze? JDK Mission Control

https://www.azul.com/products/components/azul-mission-control/



Thread	Profiling Samples	Total I/O Time	Total Blocked Time	Class Loading Time	Total Allocation	Throv
reactor-http-epoll-2	1,702		6.406 ms		2.65 GiB	
reactor-http-epoll-3	1,696		54.518 µs		2.6 GiB	
reactor-http-epoll-1	1,684		62.716 µs		2.6 GiB	
reactor-http-epoll-4	1,666		33.057 µs		2.61 GiB	

Stack Trace	Samples	Percentage
void sun.net.www.http.KeepAliveCache.put(URL, Object, HttpClient)	360910	15.5 %
void sun.net.www.http.HttpClient.putInKeepAliveCache()	360910	15.5 %
void sun.net.www.http.HttpClient.finished()	360910	15.5 %
void sun.net.www.http.KeepAliveStream.close()	360896	15.5 %
void sun.net.www.MeteredStream.justRead(long)	360896	15.5 %
int sun.net.www.MeteredStream.read(byte[], int, int)	360896	15.5 %
int java.io.FilterInputStream.read(byte[], int, int)	360896	15.5 %
int sun.net.www.protocol.http.HttpURLConnection$HttpInputStream.read(byte[], int, int)	360896	15.5 %
int com.googlecode.jsonrpc4j.NoCloseInputStream.read(byte[], int, int)	360896	15.5 %
boolean com.fasterxml.jackson.core.json.ByteSourceJsonBootstrapper.ensureLoaded(int)	360444	15.5 %
JsonEncoding com.fasterxml.jackson.core.json.ByteSourceJsonBootstrapper.detectEncoding()	360444	15.5 %
JsonParser com.fasterxml.jackson.core.json.ByteSourceJsonBootstrapper.constructParser(int, ObjectCodec, ByteQuads	360444	15.5 %
JsonParser com.fasterxml.jackson.core.JsonFactory._createParser(InputStream, IOContext)	360444	15.5 %
JsonParser com.fasterxml.jackson.core.JsonFactory.createParser(InputStream)	360444	15.5 %
Object com.fasterxml.jackson.databind.ObjectMapper.readValue(InputStream, Class)	360444	15.5 %
JsonNode com.googlecode.jsonrpc4j.ReadContext.nextValue()	360444	15.5 %
JsonNode com.googlecode.jsonrpc4j.JsonRpcClient.readResponseNode(ReadContext)	360444	15.5 %
ObjectNode com.googlecode.jsonrpc4j.JsonRpcClient.getValidResponse(String, ReadContext)	360444	15.5 %
Object com.googlecode.jsonrpc4j.JsonRpcClient.readResponse(Type, InputStream, String)	360444	15.5 %
Object com.googlecode.jsonrpc4j.JsonRpcClient.readResponse(Type, InputStream)	360444	15.5 %
Object com.googlecode.jsonrpc4j.JsonRpcHttpClient.invoke$original$pUdVVZwG(String, Object, Type, Map)	360444	15.5 %
Object com.googlecode.jsonrpc4j.JsonRpcHttpClient.invoke$original$pUdVVZwG$accessor$SYtRAFlg(String, Object, T	360444	15.5 %
Object com.googlecode.jsonrpc4j.JsonRpcHttpClient$auxiliary$yl64Wlv3.call()	360444	15.5 %
Object org.apache.skywalking.apm.agent.core.plugin.interceptor.enhance.InstMethodsInter.intercept(Object, Object[], (	360444	15.5 %
Object com.googlecode.jsonrpc4j.JsonRpcHttpClient.invoke(String, Object, Type, Map)	360444	15.5 %

Field	Value
Event Type	[Allocation in new TLA
Start Time	2/8/2023, 10:21:15.00(
Event Thread	Too many values...
	2.32×10^6 events

Arch Overview

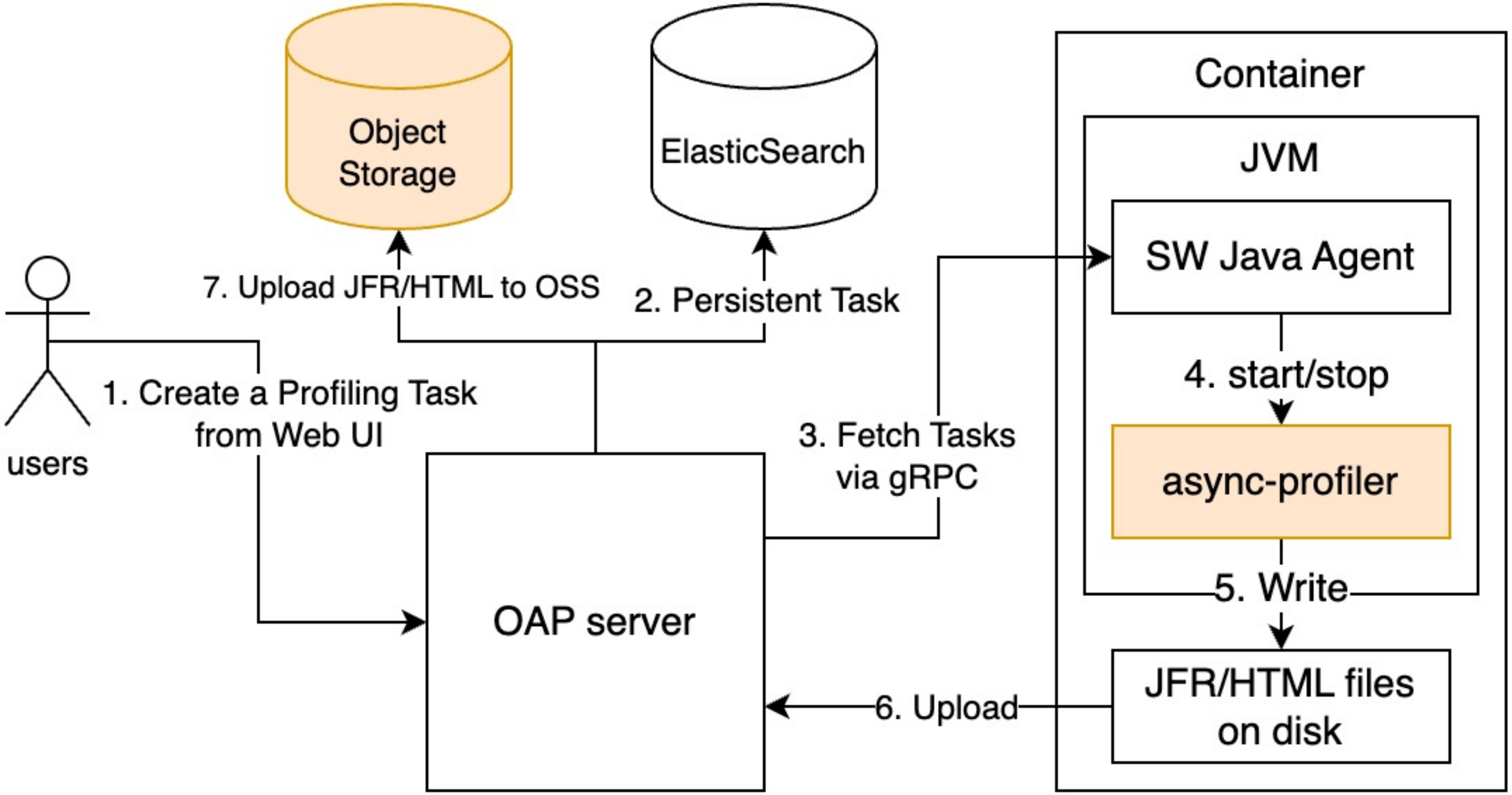
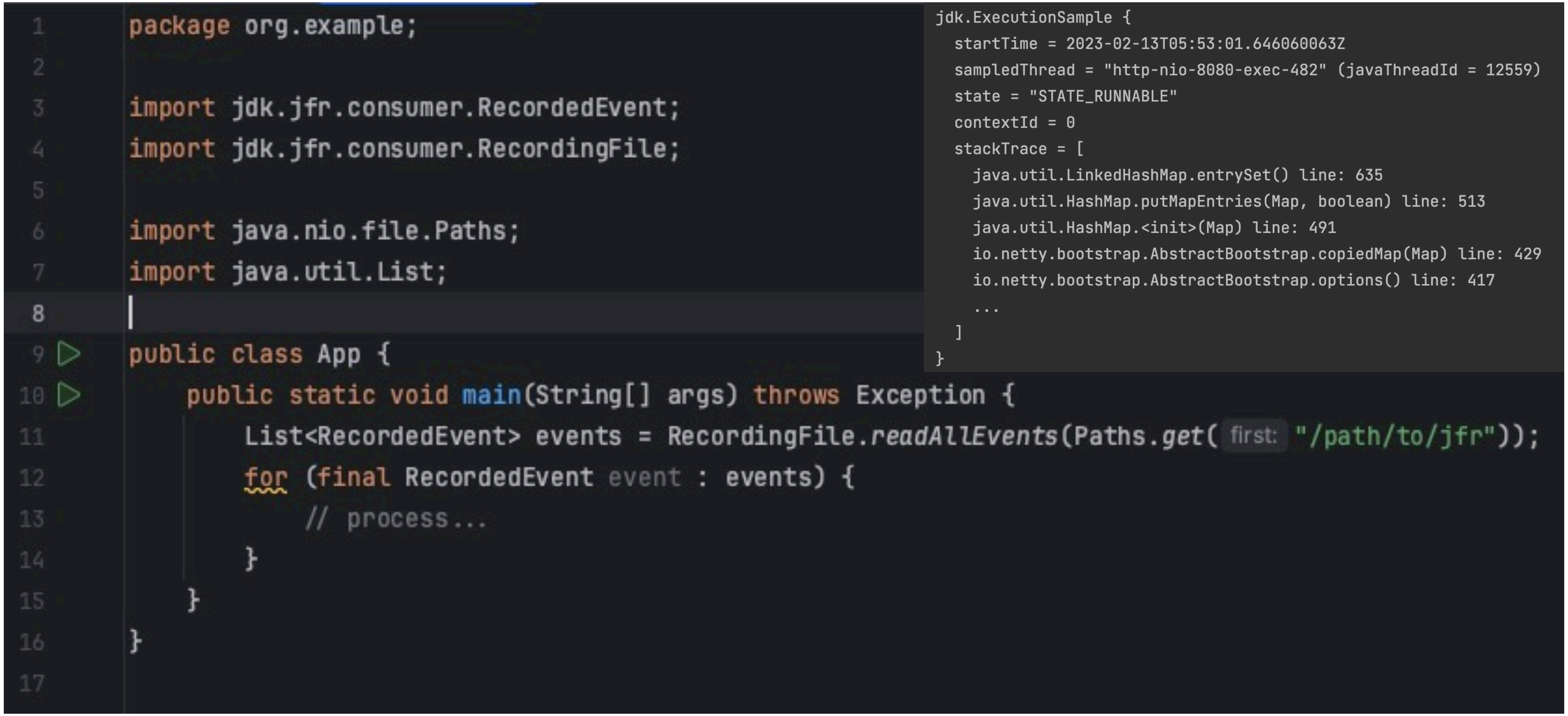


FIG. Overview of the system design

JFR Reader: read events w/ jfr mod



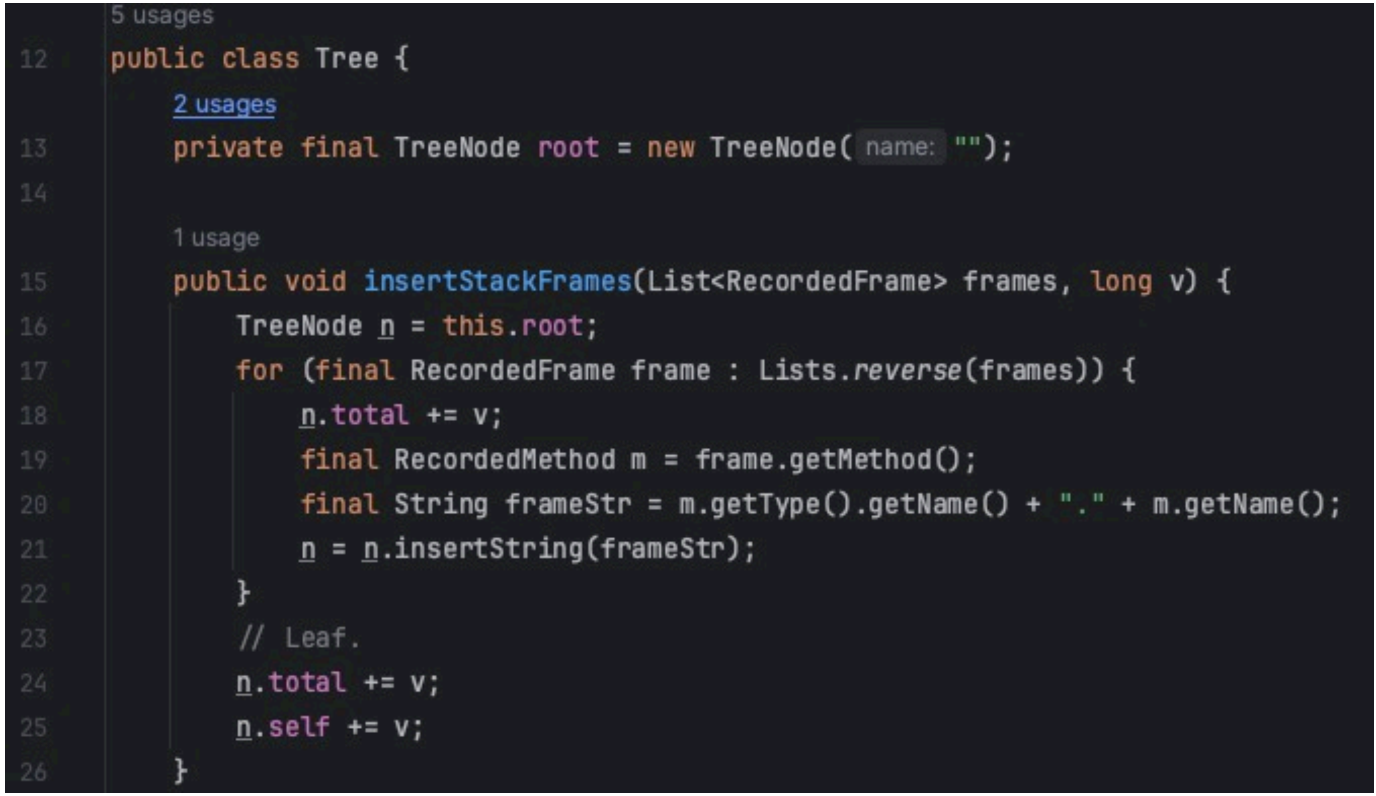
```
package org.example;

import jdk.jfr.consumer.RecordedEvent;
import jdk.jfr.consumer.RecordingFile;

import java.nio.file.Paths;
import java.util.List;

public class App {
    public static void main(String[] args) throws Exception {
        List<RecordedEvent> events = RecordingFile.readAllEvents(Paths.get( first: "/path/to/jfr"));
        for (final RecordedEvent event : events) {
            // process...
        }
    }
}
```

```
jdk.ExecutionSample {
  startTime = 2023-02-13T05:53:01.646060063Z
  sampledThread = "http-nio-8080-exec-482" (javaThreadId = 12559)
  state = "STATE_RUNNABLE"
  contextId = 0
  stackTrace = [
    java.util.LinkedHashMap.entrySet() line: 635
    java.util.HashMap.putMapEntries(Map, boolean) line: 513
    java.util.HashMap.<init>(Map) line: 491
    io.netty.bootstrap.AbstractBootstrap.copiedMap(Map) line: 429
    io.netty.bootstrap.AbstractBootstrap.options() line: 417
    ...
  ]
}
```

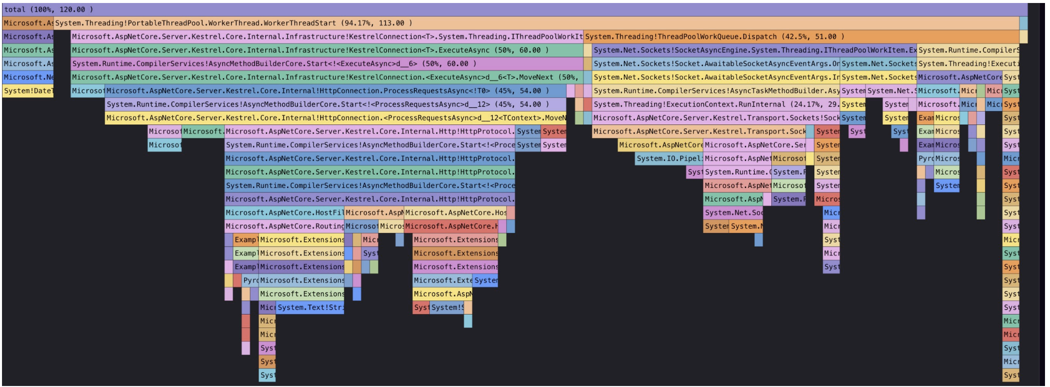
FIG. Read all events and then decode (JDK 8u262+)

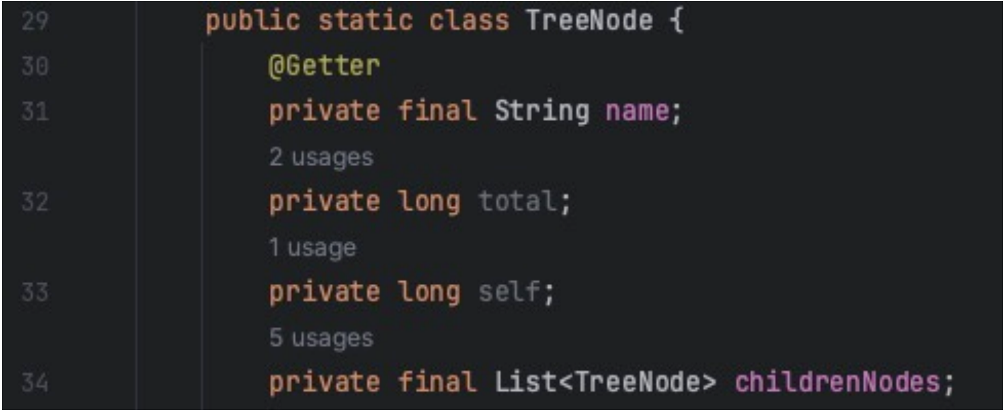
JFR Reader: build call stack

2023 SUMMIT | APACHE • SkyWalking CHINA · SHANGHAI



```
5 usages
public class Tree {
    2 usages
    private final TreeNode root = new TreeNode( name: "");

    1 usage
    public void insertStackFrames(List<RecordedFrame> frames, long v) {
        TreeNode n = this.root;
        for (final RecordedFrame frame : Lists.reverse(frames)) {
            n.total += v;
            final RecordedMethod m = frame.getMethod();
            final String frameStr = m.getType().getName() + "." + m.getName();
            n = n.insertString(frameStr);
        }
        // Leaf.
        n.total += v;
        n.self += v;
    }
```

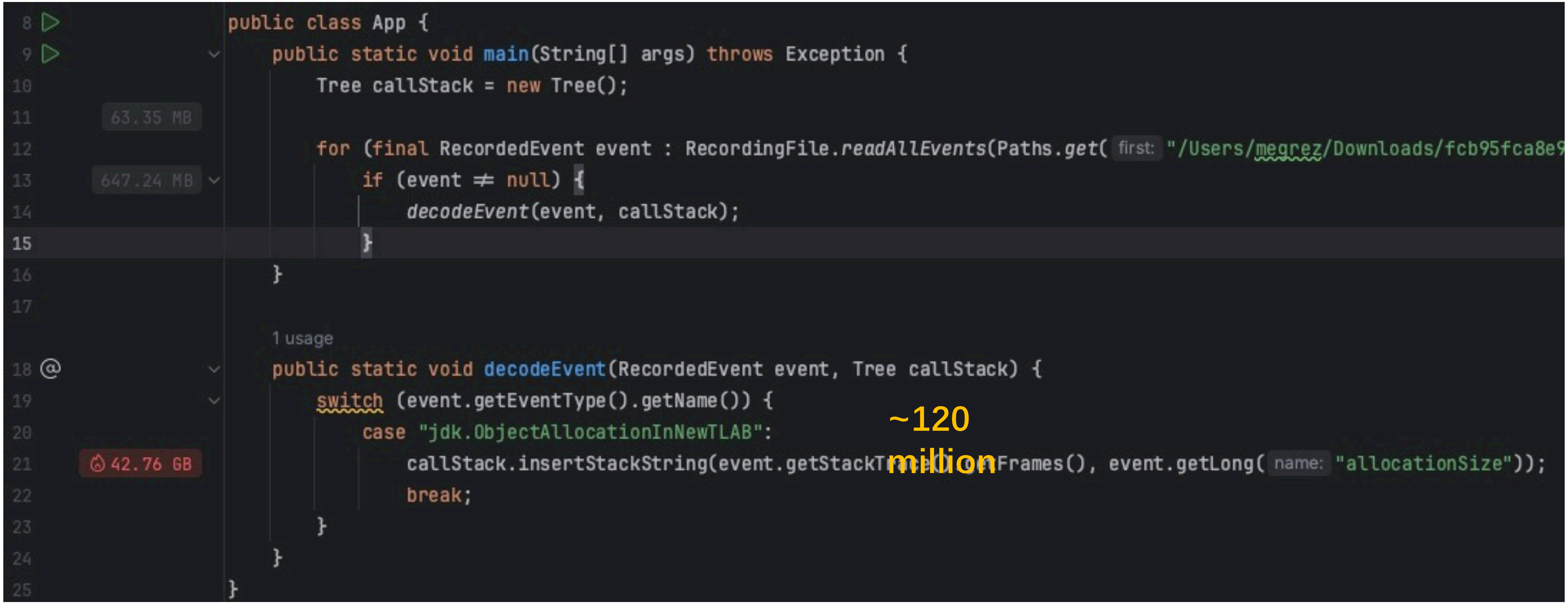




```
public static class TreeNode {
    @Getter
    private final String name;
    2 usages
    private long total;
    1 usage
    private long self;
    5 usages
    private final List<TreeNode> childrenNodes;
```

FIG. Build the call stack (Tree with treeNode as children)

JFR Reader: build call stack (~80M)

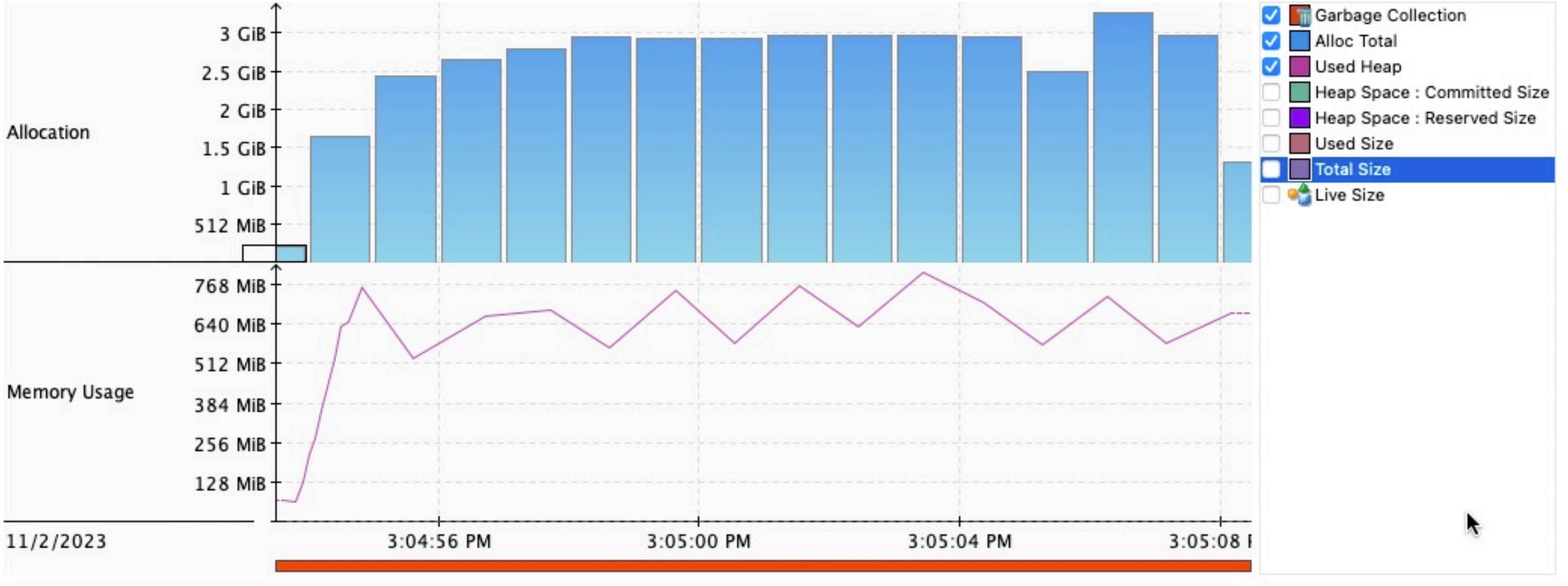


```java
public class App {
    public static void main(String[] args) throws Exception {
        Tree callStack = new Tree();

        for (final RecordedEvent event : RecordingFile.readAllEvents(Paths.get( first: "/Users/megrez/Downloads/fcb95fca8e9
            if (event ≠ null) {
                decodeEvent(event, callStack);
            }
    }

    public static void decodeEvent(RecordedEvent event, Tree callStack) {
        switch (event.getEventType().getName()) {
            case "jdk.ObjectAllocationInNewTLAB":
                callStack.insertStackString(event.getStackTrace().getFrames(), event.getLong( name: "allocationSize"));
                break;
        }
    }
}
```

~120 million

FIG. Memory issue: large heap size while building the call stack with millions of (allocation) events

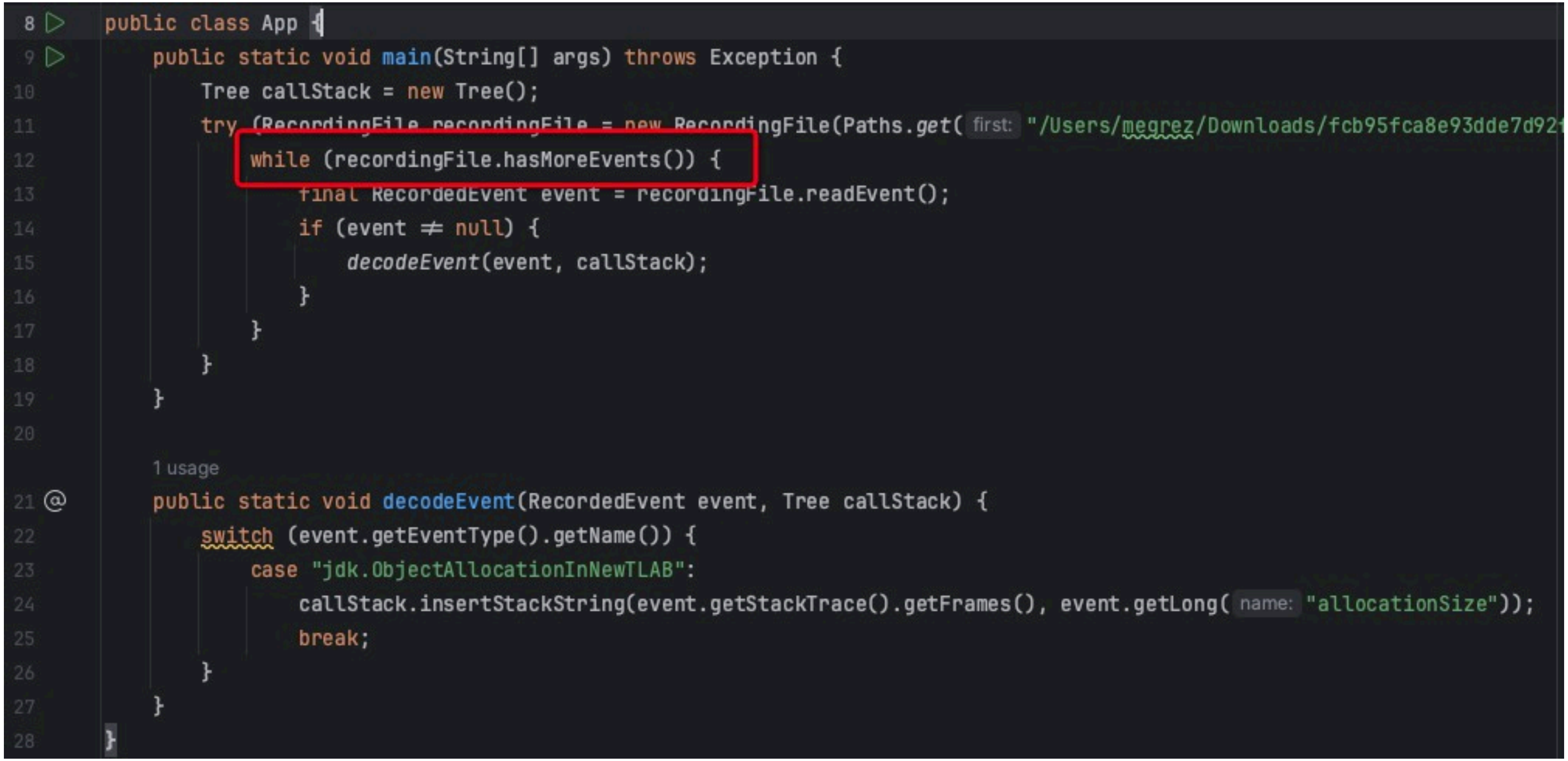
JFR Reader: build call stack (~80M)



☑ Garbage Collection
☑ Alloc Total
☑ Used Heap
☐ Heap Space : Committed Size
☐ Heap Space : Reserved Size
☐ Used Size
☐ Total Size
☐ Live Size

FIG. Memory issue: large heap size while building the call stack with millions of (allocation) events

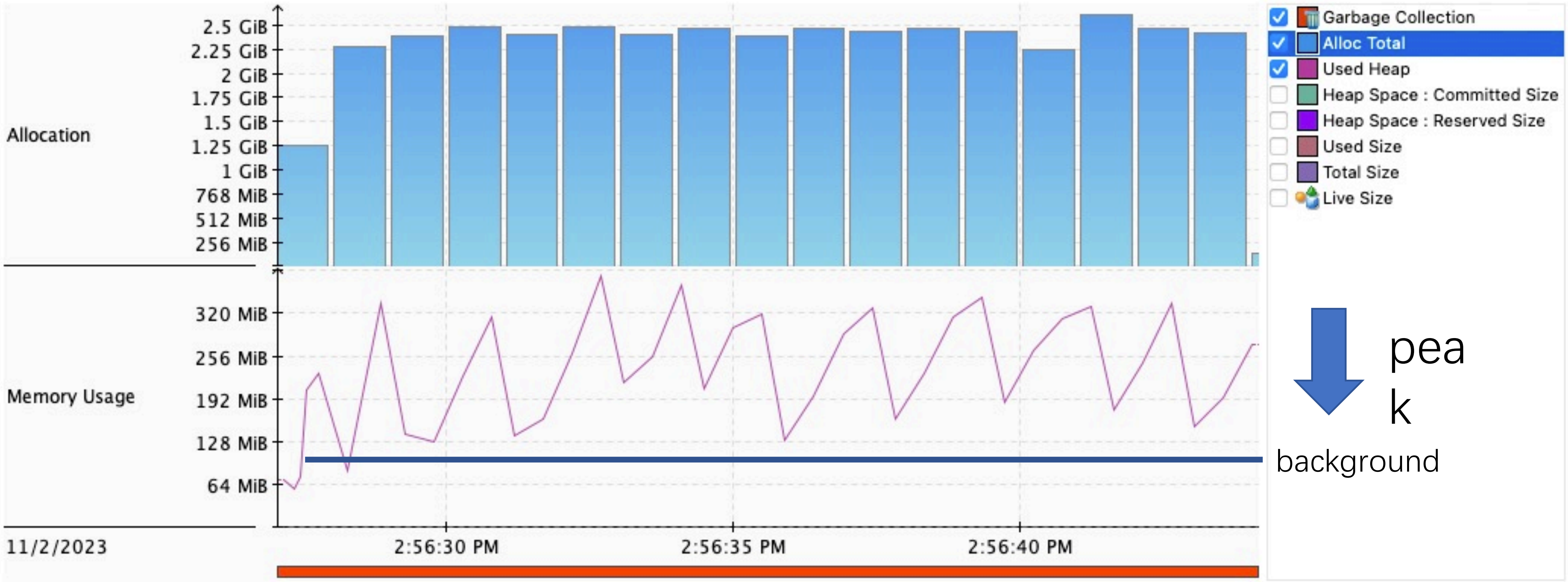
JFR Reader: Iterator pattern



```java
public class App {
    public static void main(String[] args) throws Exception {
        Tree callStack = new Tree();
        try (RecordingFile recordingFile = new RecordingFile(Paths.get( first: "/Users/megrez/Downloads/fcb95fca8e93dde7d92
            while (recordingFile.hasMoreEvents()) {
                final RecordedEvent event = recordingFile.readEvent();
                if (event ≠ null) {
                    decodeEvent(event, callStack);
                }
            }
        }
    }

    1 usage
    public static void decodeEvent(RecordedEvent event, Tree callStack) {
        switch (event.getEventType().getName()) {
            case "jdk.ObjectAllocationInNewTLAB":
                callStack.insertStackString(event.getStackTrace().getFrames(), event.getLong( name: "allocationSize"));
                break;
        }
    }
}
```

FIG. Process RecordEvent one by one

JFR Reader: Iterator pattern



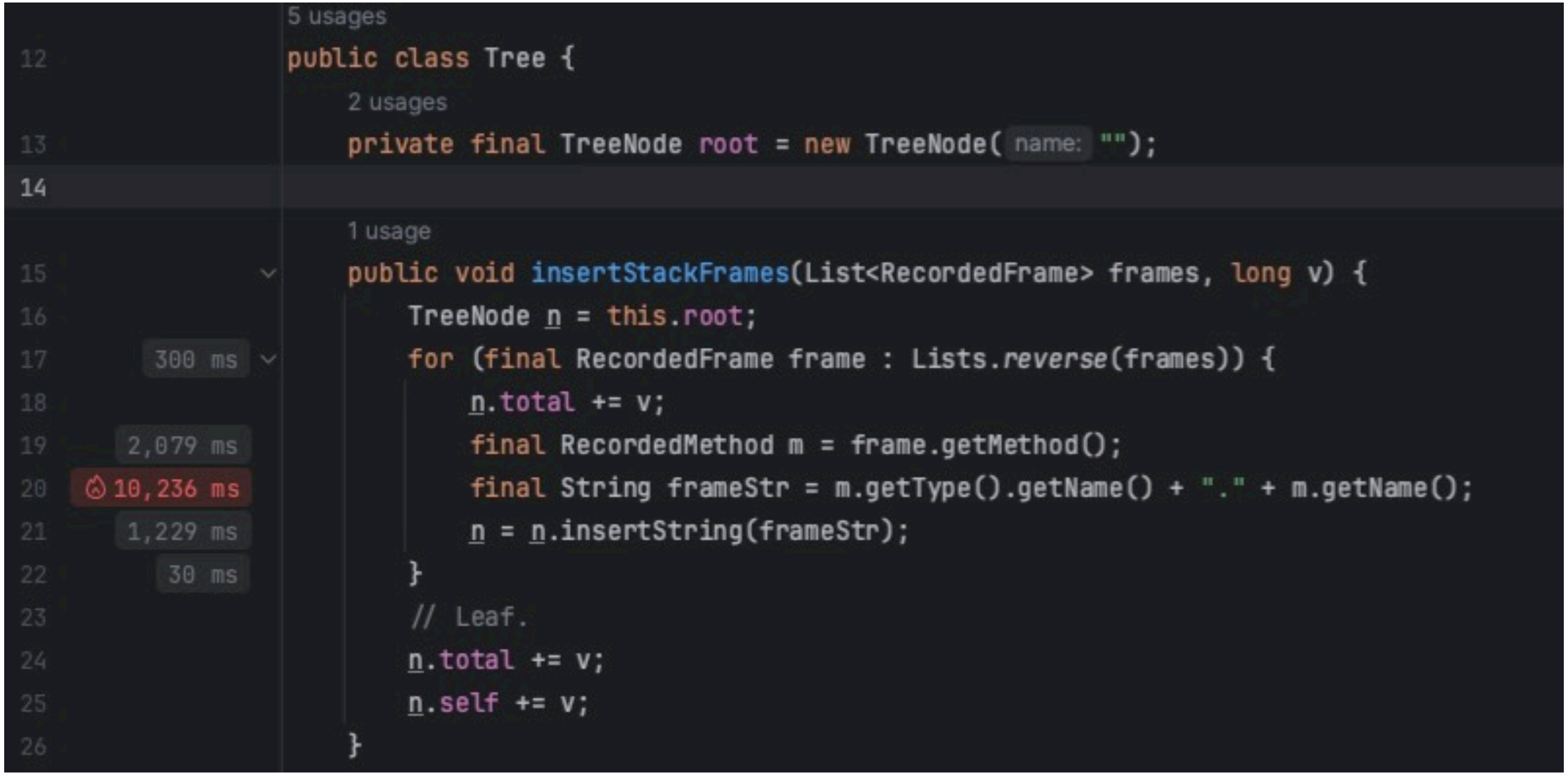
☑ Garbage Collection
☑ Alloc Total
☑ Used Heap
□ Heap Space : Committed Size
□ Heap Space : Reserved Size
□ Used Size
□ Total Size
□ Live Size

peak

background

FIG. Memory issue: large heap size while building the call stack with millions of (allocation) events

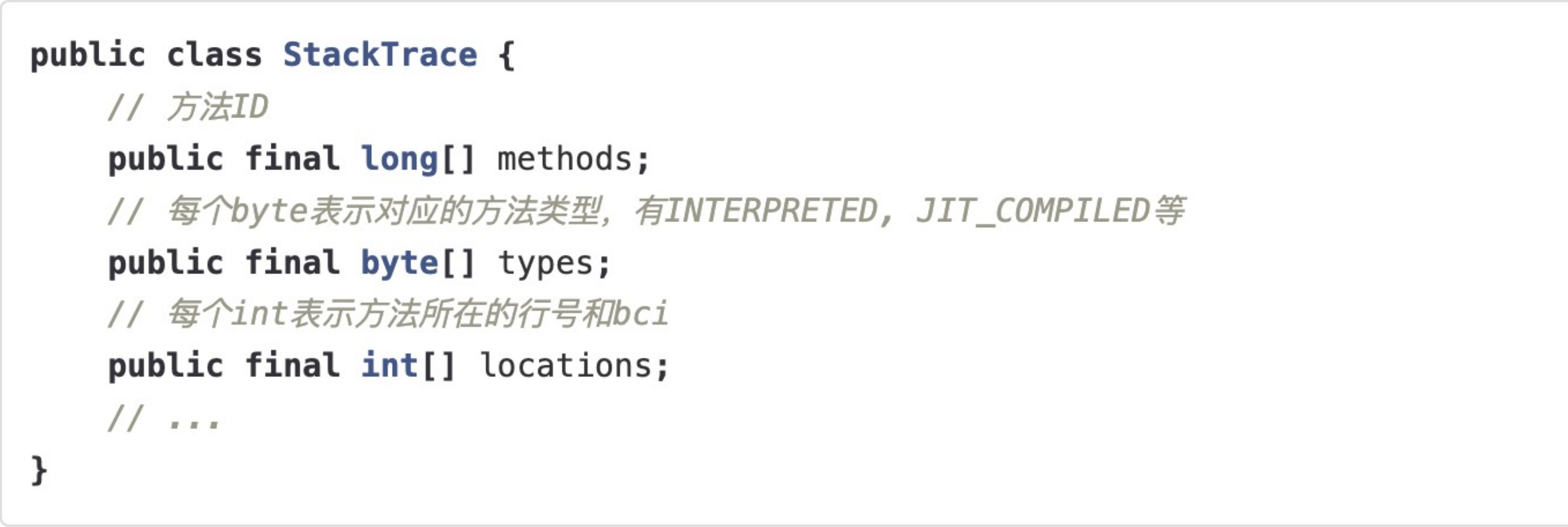
JFR Reader: Slow!



```
5 usages
public class Tree {
    2 usages
    private final TreeNode root = new TreeNode( name: "");

    1 usage
    public void insertStackFrames(List<RecordedFrame> frames, long v) {
        TreeNode n = this.root;
        for (final RecordedFrame frame : Lists.reverse(frames)) {
            n.total += v;
            final RecordedMethod m = frame.getMethod();
            final String frameStr = m.getType().getName() + "." + m.getName();
            n = n.insertString(frameStr);
        }
        // Leaf.
        n.total += v;
        n.self += v;
    }
```

FIG. Performance issue: most time spent on building frame names

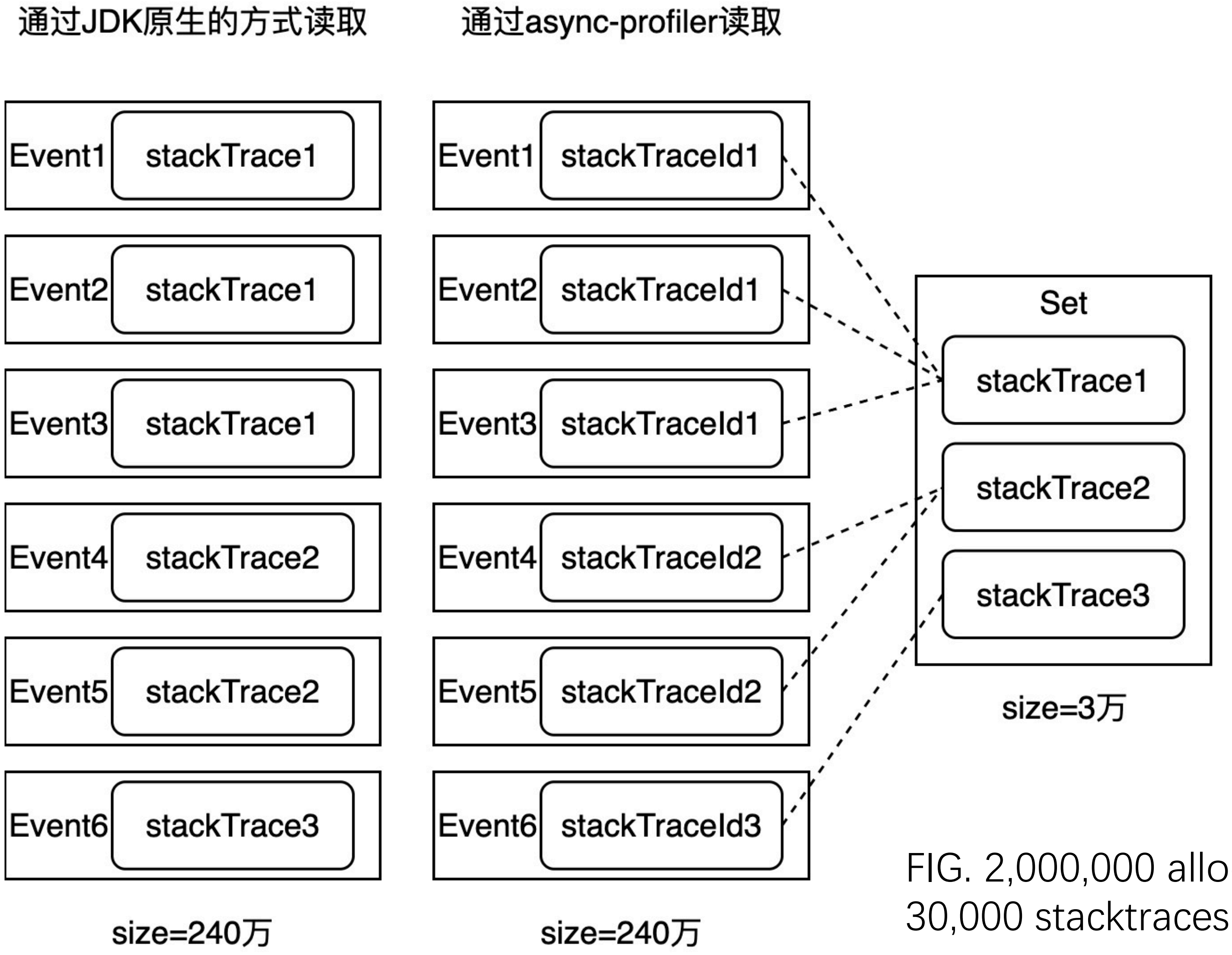
JFR Reader: use raw references



```java
public class StackTrace {
    // 方法ID
    public final long[] methods;
    // 每个byte表示对应的方法类型，有INTERPRETED，JIT_COMPILED等
    public final byte[] types;
    // 每个int表示方法所在的行号和bci
    public final int[] locations;
    // ...
}
```

FIG. use references instead of materialized stack trace

JFR Reader: use raw references



通过JDK原生的方式读取

Event1 stackTrace1

Event2 stackTrace1

Event3 stackTrace1

Event4 stackTrace2

Event5 stackTrace2

Event6 stackTrace3

size=240万

通过async-profiler读取

Event1 stackTraceId1

Event2 stackTraceId1

Event3 stackTraceId1

Event4 stackTraceId2

Event5 stackTraceId2

Event6 stackTraceId3

size=240万

Set

stackTrace1

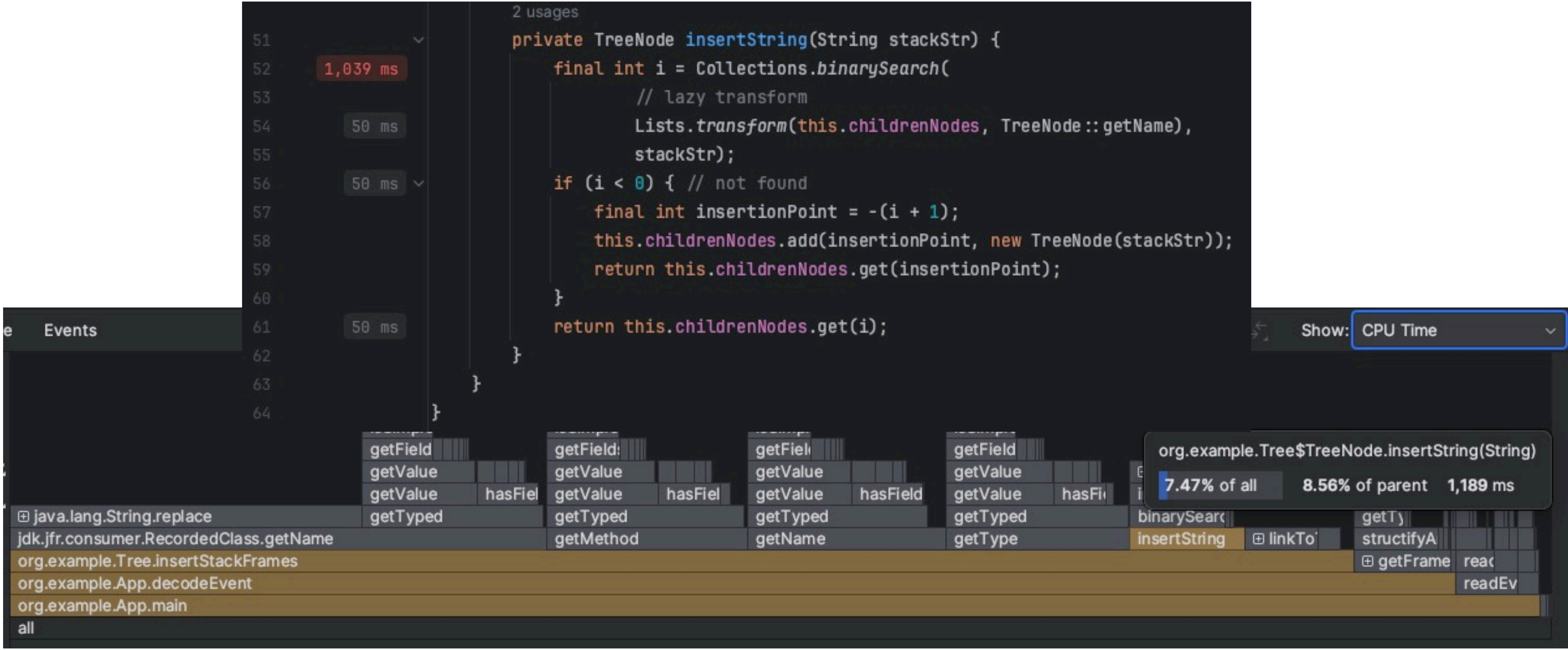
stackTrace2

stackTrace3

size=3万

FIG. 2,000,000 alloc events share 30,000 stacktraces

JFR Reader: binary search O(logN)?



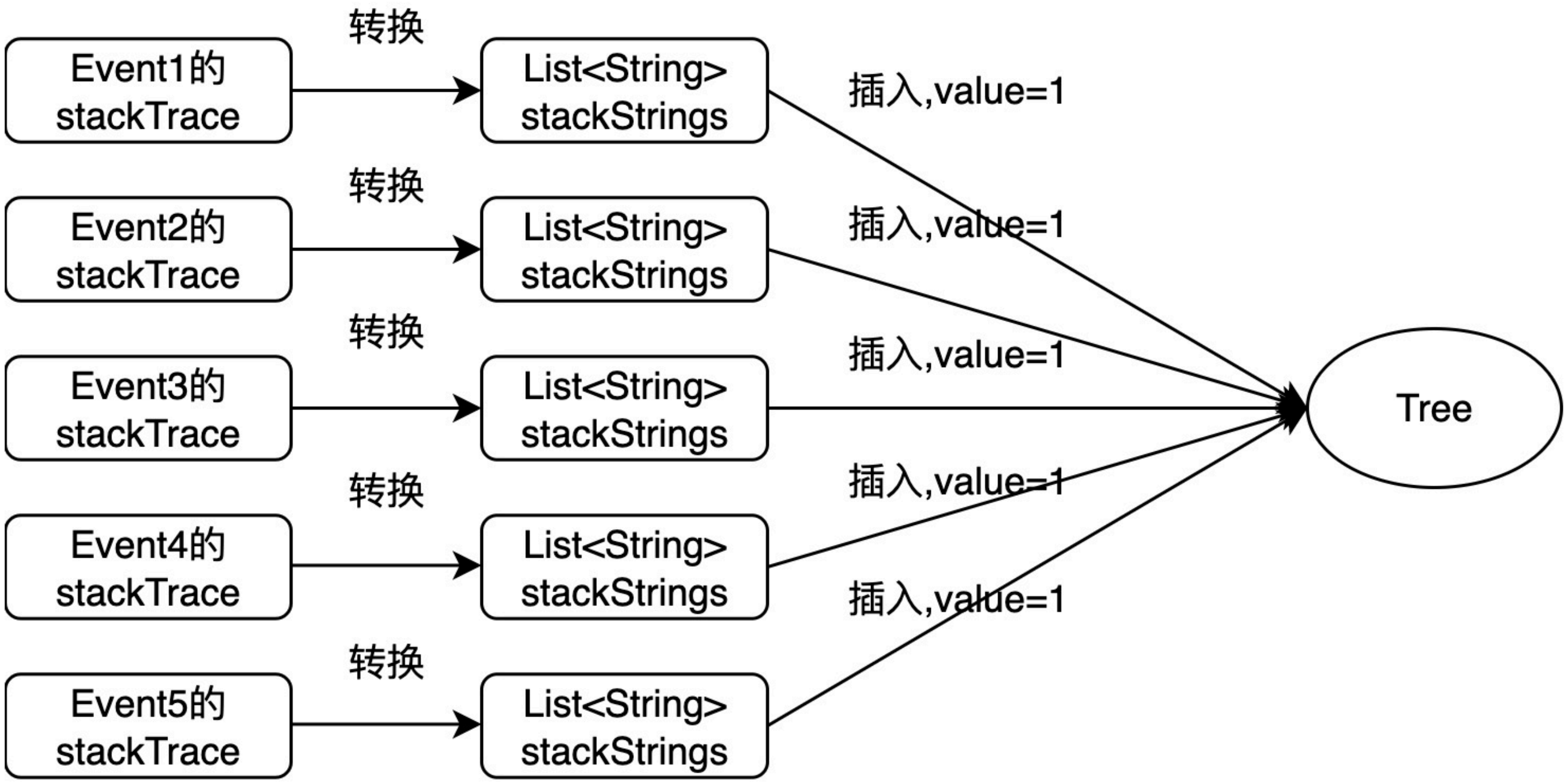
```
    2 usages
    private TreeNode insertString(String stackStr) {
        final int i = Collections.binarySearch(
                // lazy transform
                Lists.transform(this.childrenNodes, TreeNode::getName),
                stackStr);
        if (i < 0) { // not found
            final int insertionPoint = -(i + 1);
            this.childrenNodes.add(insertionPoint, new TreeNode(stackStr));
            return this.childrenNodes.get(insertionPoint);
        }
        return this.childrenNodes.get(i);
    }
  }
}
```

org.example.Tree$TreeNode.insertString(String)
7.47% of all 8.56% of parent 1,189 ms

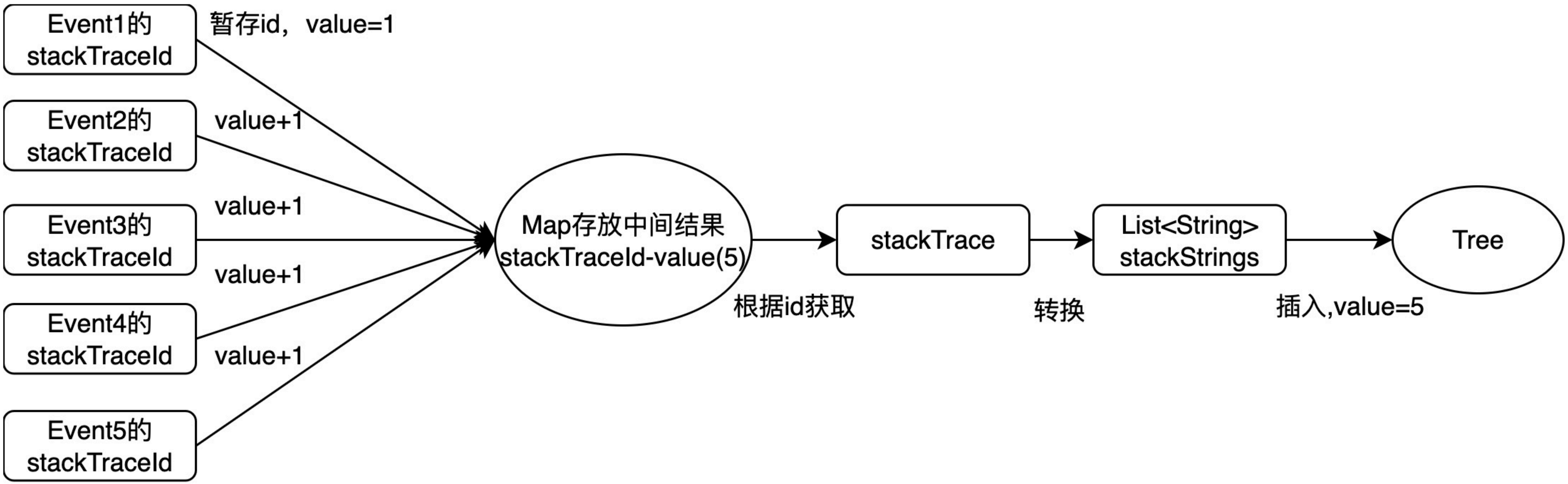
FIG. Another performance issue: too many binary searches during insertion even if binary search has O(logN) complexity

JFR Reader: insert first

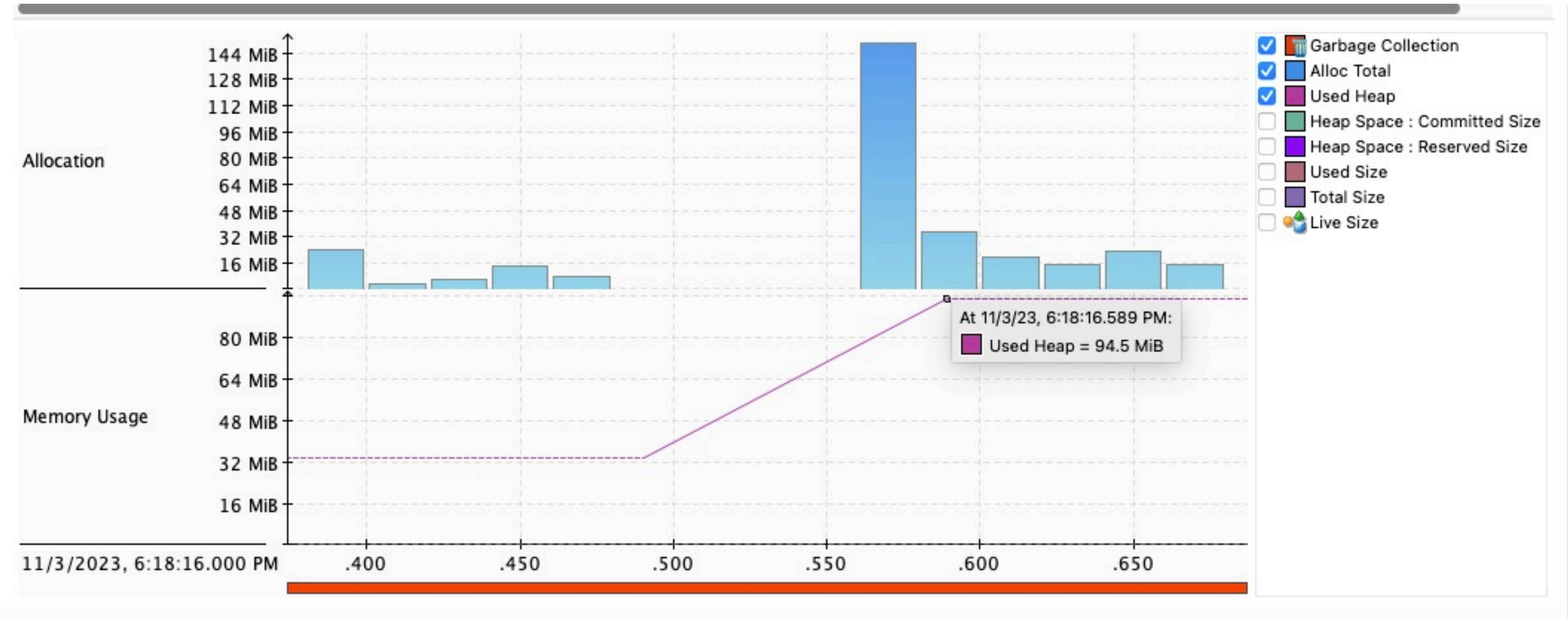
pyroscope



JFR Reader: aggregate first



JFR Reader: final round

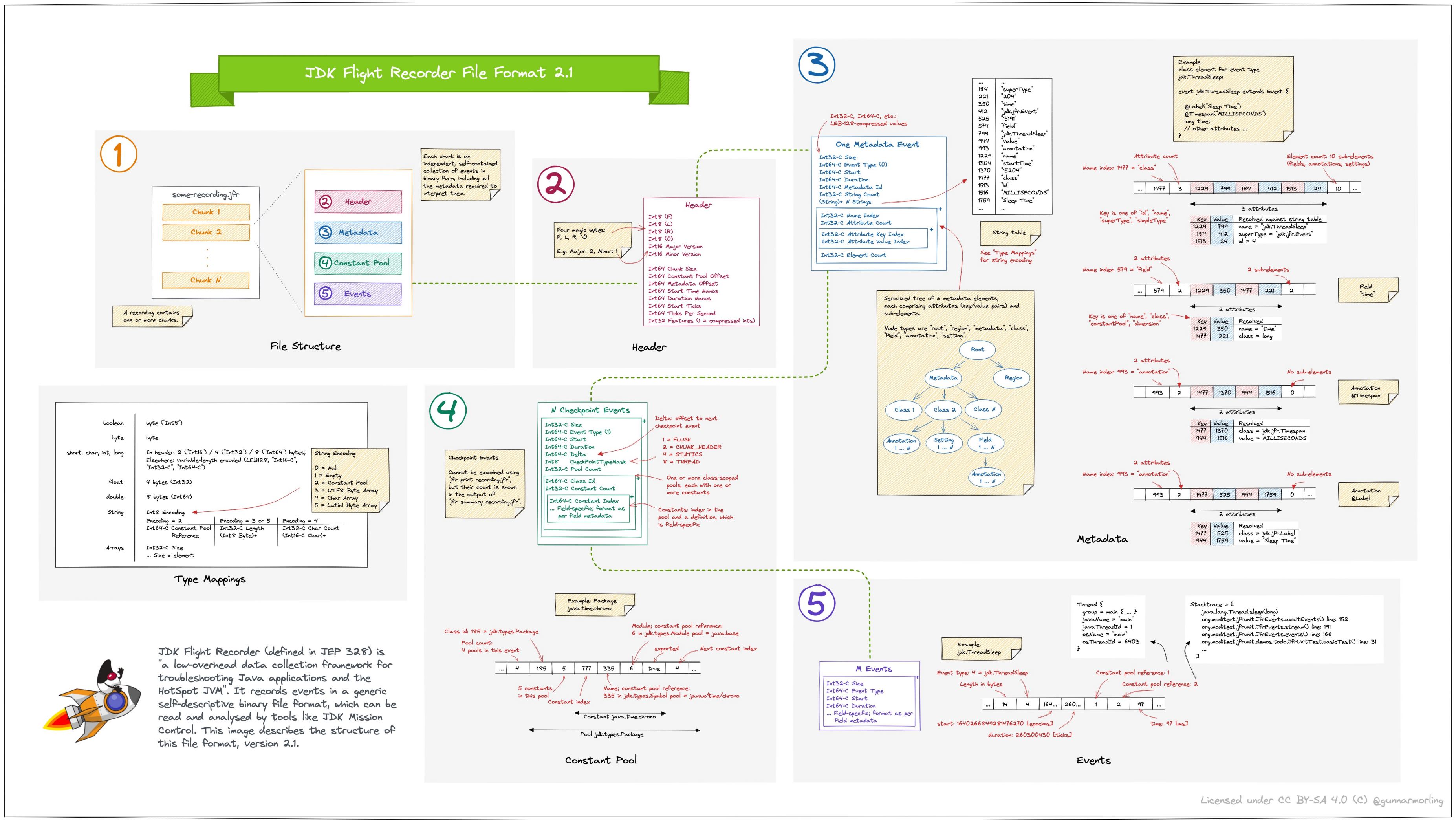


☑ Garbage Collection
☑ Alloc Total
☑ Used Heap
□ Heap Space : Committed Size
□ Heap Space : Reserved Size
□ Used Size
□ Total Size
□ Live Size

At 11/3/23, 6:18:16.589 PM:
Used Heap = 94.5 MiB

FIG. Final result: use **<100M** heap, and finish parsing **<1 second**

JFR Reader: What about large JFR file?



A recording contains one or more chunks.

Each chunk is an independent, self-contained collection of events in binary form, including all the metadata required to interpret them.

Four magic bytes: F, L, R, \0

E.g. Major: 2, Minor: 1

Header

```
Int8 (F)
Int8 (L)
Int8 (R)
Int8 (0)
Int16 Major Version
Int16 Minor Version

Int64 Chunk Size
Int64 Constant Pool Offset
Int64 Metadata Offset
Int64 Start Time Nanos
Int64 Duration Nanos
Int64 Start Ticks
Int64 Ticks Per Second
Int32 Features (1 = compressed ints)
```

Int32-C, Int64-C, etc.: LEB-128-compressed values

One Metadata Event

```
Int32-C Size
Int64-C Event Type (0)
Int64-C Start
Int64-C Duration
Int64-C Metadata Id
Int32-C String Count
(String)+ N Strings
  Int32-C Name Index
  Int32-C Attribute Count
    Int32-C Attribute Key Index
    Int32-C Attribute Value Index
  Int32-C Element Count
```

…	…
184	"superType"
221	"204"
350	"time"
412	"jdk.jfr.Event"
525	"15191"
574	"field"
799	"jdk.ThreadSleep"
944	"value"
993	"annotation"
1229	"name"
1304	"startTime"
1370	"15204"
1477	"class"
1513	"id"
1516	"MILLISECONDS"
1759	"Sleep Time"
…	…

See "Type Mappings" for string encoding

Serialized tree of N metadata elements, each comprising attributes (key/value pairs) and sub-elements.

Node types are "root", "region", "metadata", "class", "field", "annotation", "setting".

Example: class element for event type jdk.ThreadSleep:

```
event jdk.ThreadSleep extends Event {

  @Label("Sleep Time")
  @Timespan("MILLISECONDS")
  long time;
  // other attributes ...
}
```

Name index: 1477 = "class"; Attribute count; Element count: 10 sub-elements (fields, annotations, settings)

…	1477	3	1229	799	184	412	1513	24	10	…

3 attributes

Key is one of "id", "name", "superType", "simpleType"

Key	Value	Resolved against string table
1229	799	name = "jdk.ThreadSleep"
184	412	superType = "jdk.jfr.Event"
1513	24	id = 4

Name index: 579 = "field"; 2 attributes; 2 sub-elements

…	579	2	1229	350	1477	221	2	

2 attributes

Key is one of "name", "class", "constantPool", "dimension"

Key	Value	Resolved
1229	350	name = "time"
1477	221	class = long

Name index: 993 = "annotation"; 2 attributes; No sub-elements

	993	2	1477	1370	944	1516	0	

2 attributes

Key	Value	Resolved
1477	1370	class = jdk.jfr.Timespan
944	1516	value = MILLISECONDS

Name index: 993 = "annotation"; 2 attributes; No sub-elements

	993	2	1477	525	944	1759	0	…

2 attributes

Key	Value	Resolved
1477	525	class = jdk.jfr.Label
944	1759	value = "Sleep Time"

Type Mappings

boolean	byte ("Int8")
byte	byte
short, char, int, long	In header: 2 ("Int16") / 4 ("Int32") / 8 ("Int64") bytes; Elsewhere: variable-length encoded (LEB128, "Int16-C", "Int32-C", "Int64-C")
float	4 bytes (Int32)
double	8 bytes (Int64)

String	Int8 Encoding
	Encoding = 2: Int64-C Constant Pool Reference
	Encoding = 3 or 5: Int32-C Length (Int8 Byte)+
	Encoding = 4: Int32-C Char Count (Int16-C Char)+

Arrays	Int32-C Size … Size x element

String Encoding

0 = Null
1 = Empty
2 = Constant Pool
3 = UTF8 Byte Array
4 = Char Array
5 = Latin1 Byte Array

Checkpoint Events

Cannot be examined using "jfr print recording.jfr", but their count is shown in the output of "jfr summary recording.jfr".

N Checkpoint Events

```
Int32-C Size
Int64-C Event Type (1)
Int64-C Start
Int64-C Duration
Int64-C Delta
Int8    CheckPointTypeMask
Int32-C Pool Count
  Int64-C Class Id
  Int32-C Constant Count
    Int64-C Constant Index
    ... Field-specific; format as
      per field metadata
```

Delta: offset to next checkpoint event

1 = FLUSH
2 = CHUNK_HEADER
4 = STATICS
8 = THREAD

One or more class-scoped pools, each with one or more constants

Constants: index in the pool and a definition, which is field-specific

Example: Package java.time.chrono

Class id: 185 = jdk.types.Package; Pool count: 4 pools in this event; Module; constant pool reference: 6 in jdk.types.Module pool = java.base; exported; Next constant index

…	4	185	5	777	335	6	true	4	…

5 constants in this pool; Constant index; Name; constant pool reference: 335 in jdk.types.Symbol pool = javax/time/chrono

Constant java.time.chrono

Pool jdk.types.Package

M Events

```
Int32-C Size
Int64-C Event Type
Int64-C Start
Int64-C Duration
... Field-specific; format as per
  field metadata
```

Example: jdk.ThreadSleep

Event type: 4 = jdk.ThreadSleep; Length in bytes; Constant pool reference: 1; Constant pool reference: 2

…	14	4	164…	260…	1	2	97	…

start: 1640266849281476270 [epochns]; duration: 260300430 [ticks]; time: 97 [ms]

```
Thread {
  group = main { … }
  javaName = "main"
  javaThreadId = 1
  osName = "main"
  osThreadId = 6403
}
```

```
Stacktrace = [
    java.lang.Thread.sleep(long)
    org.moditect.jfrunit.JfrEvents.awaitEvents() line: 152
    org.moditect.jfrunit.JfrEvents.stream() line: 191
    org.moditect.jfrunit.JfrEvents.events() line: 166
    org.moditect.jfrunit.demos.todo.JfrUnitTest.basicTest() line: 31
    ...
]
```

JDK Flight Recorder (defined in JEP 328) is "a low-overhead data collection framework for troubleshooting Java applications and the HotSpot JVM". It records events in a generic self-descriptive binary file format, which can be read and analysed by tools like JDK Mission Control. This image describes the structure of this file format, version 2.1.

JFR Reader: What about large JFR file?

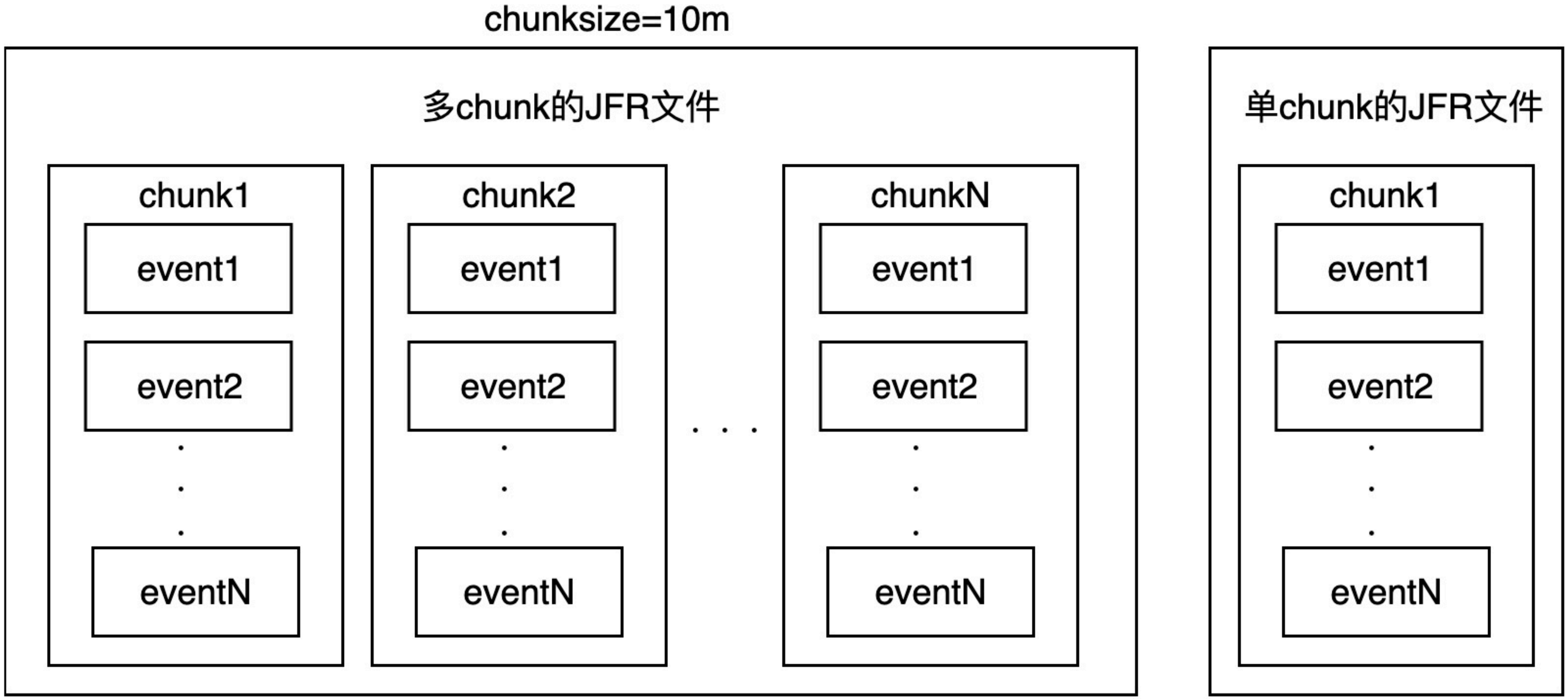
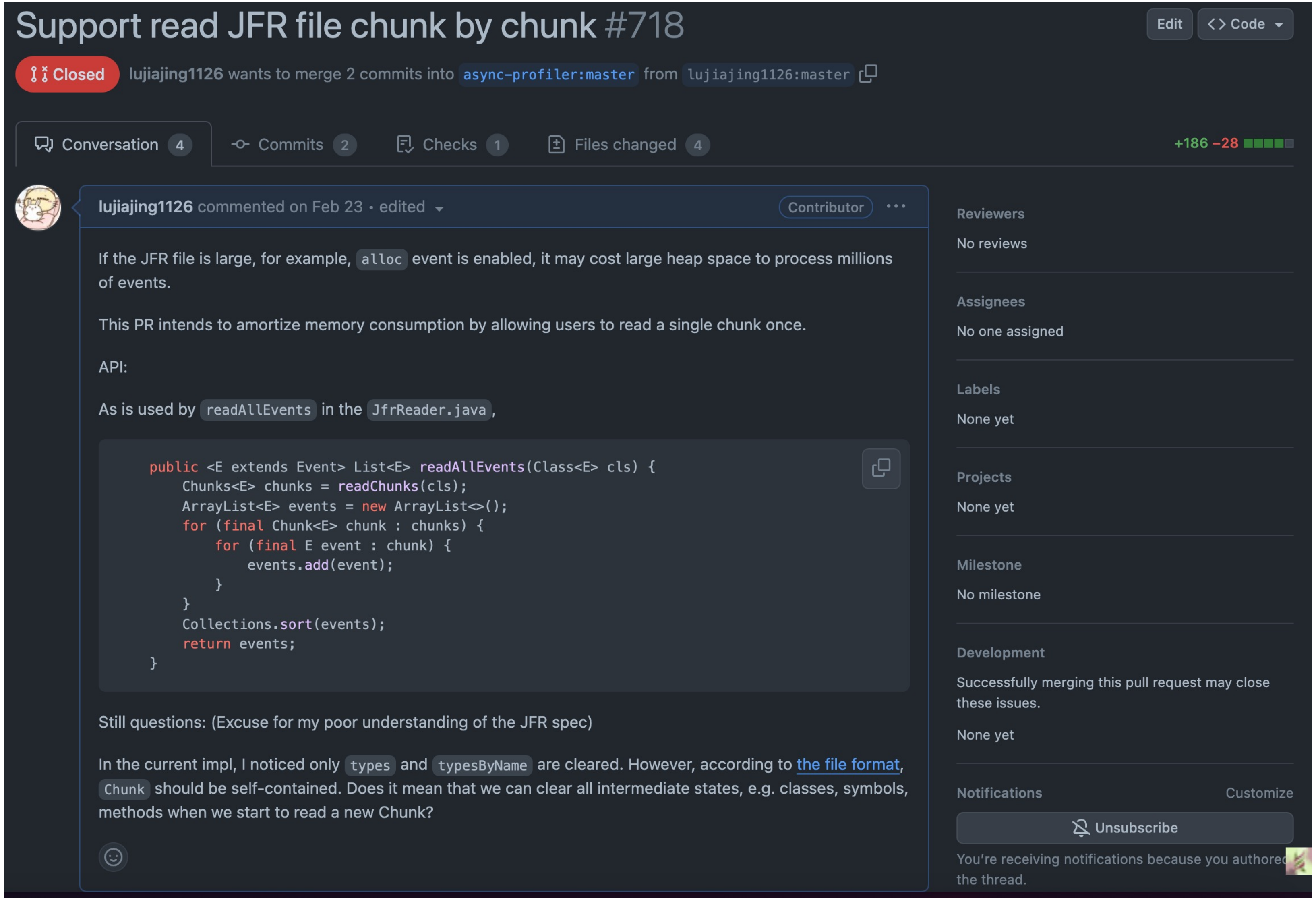


FIG. ChunkSize can be controlled by parameter

JFR Reader: What about large JFR file?

2023 SUMMIT | APACHE • SkyWalking CHINA · SHANGHAI



Support read JFR file chunk by chunk #718

Closed lujiajing1126 wants to merge 2 commits into `async-profiler:master` from `lujiajing1126:master`

lujiajing1126 commented on Feb 23 • edited

If the JFR file is large, for example, `alloc` event is enabled, it may cost large heap space to process millions of events.

This PR intends to amortize memory consumption by allowing users to read a single chunk once.

API:

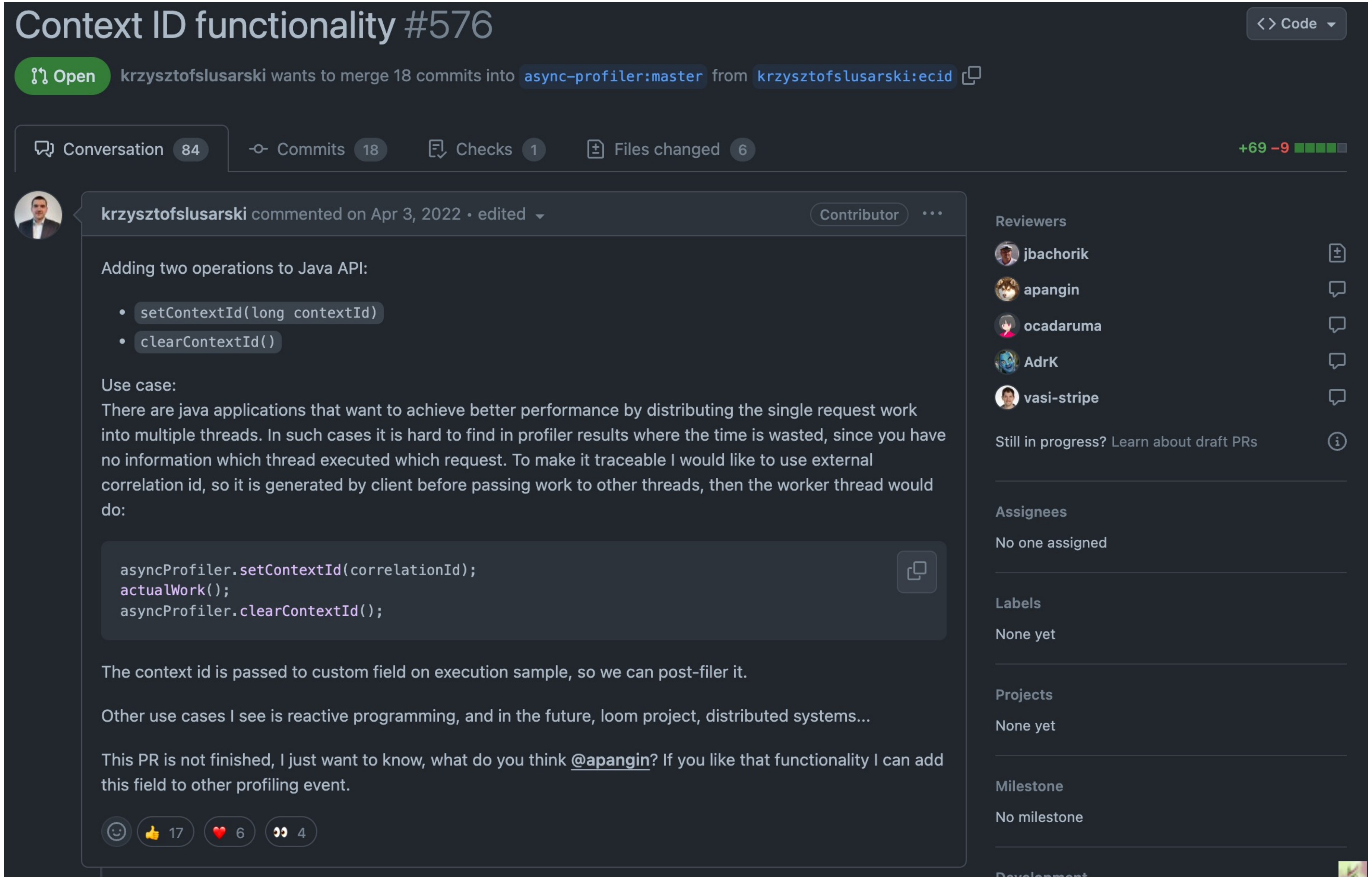
As is used by `readAllEvents` in the `JfrReader.java`,

```
    public <E extends Event> List<E> readAllEvents(Class<E> cls) {
        Chunks<E> chunks = readChunks(cls);
        ArrayList<E> events = new ArrayList<>();
        for (final Chunk<E> chunk : chunks) {
            for (final E event : chunk) {
                events.add(event);
            }
        }
        Collections.sort(events);
        return events;
    }
```

Still questions: (Excuse for my poor understanding of the JFR spec)

In the current impl, I noticed only `types` and `typesByName` are cleared. However, according to the file format, `Chunk` should be self-contained. Does it mean that we can clear all intermediate states, e.g. classes, symbols, methods when we start to read a new Chunk?

One more thing: correlation



Context ID functionality #576

Open krzysztofslusarski wants to merge 18 commits into async-profiler:master from krzysztofslusarski:ecid

krzysztofslusarski commented on Apr 3, 2022 • edited

Adding two operations to Java API:

- `setContextId(long contextId)`
- `clearContextId()`

Use case:
There are java applications that want to achieve better performance by distributing the single request work into multiple threads. In such cases it is hard to find in profiler results where the time is wasted, since you have no information which thread executed which request. To make it traceable I would like to use external correlation id, so it is generated by client before passing work to other threads, then the worker thread would do:

```
asyncProfiler.setContextId(correlationId);
actualWork();
asyncProfiler.clearContextId();
```

The context id is passed to custom field on execution sample, so we can post-filer it.

Other use cases I see is reactive programming, and in the future, loom project, distributed systems...

This PR is not finished, I just want to know, what do you think @apangin? If you like that functionality I can add this field to other profiling event.

Reviewers

- jbachorik
- apangin
- ocadaruma
- AdrK
- vasi-stripe

Still in progress? Learn about draft PRs

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

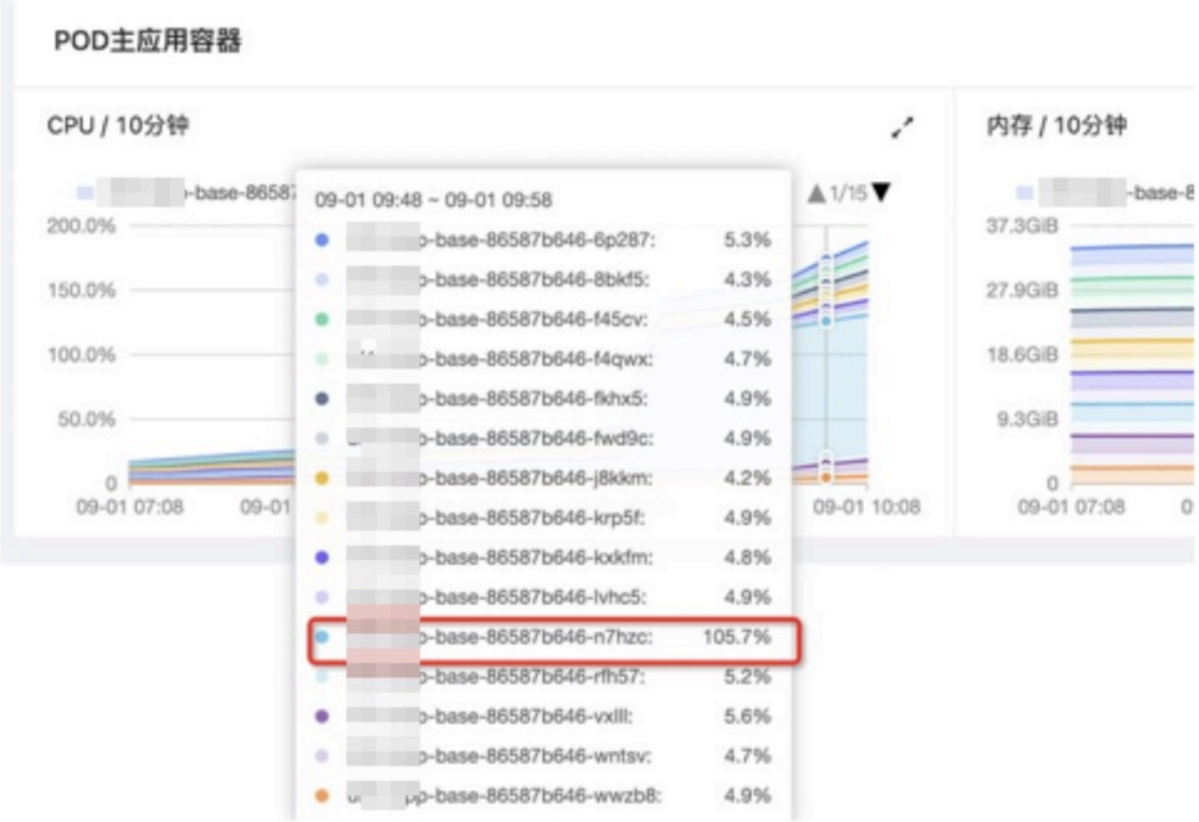
No milestone

02

交互式诊断

Interactive Diag.

How to diag. a CPU spike



POD主应用容器

CPU / 10分钟

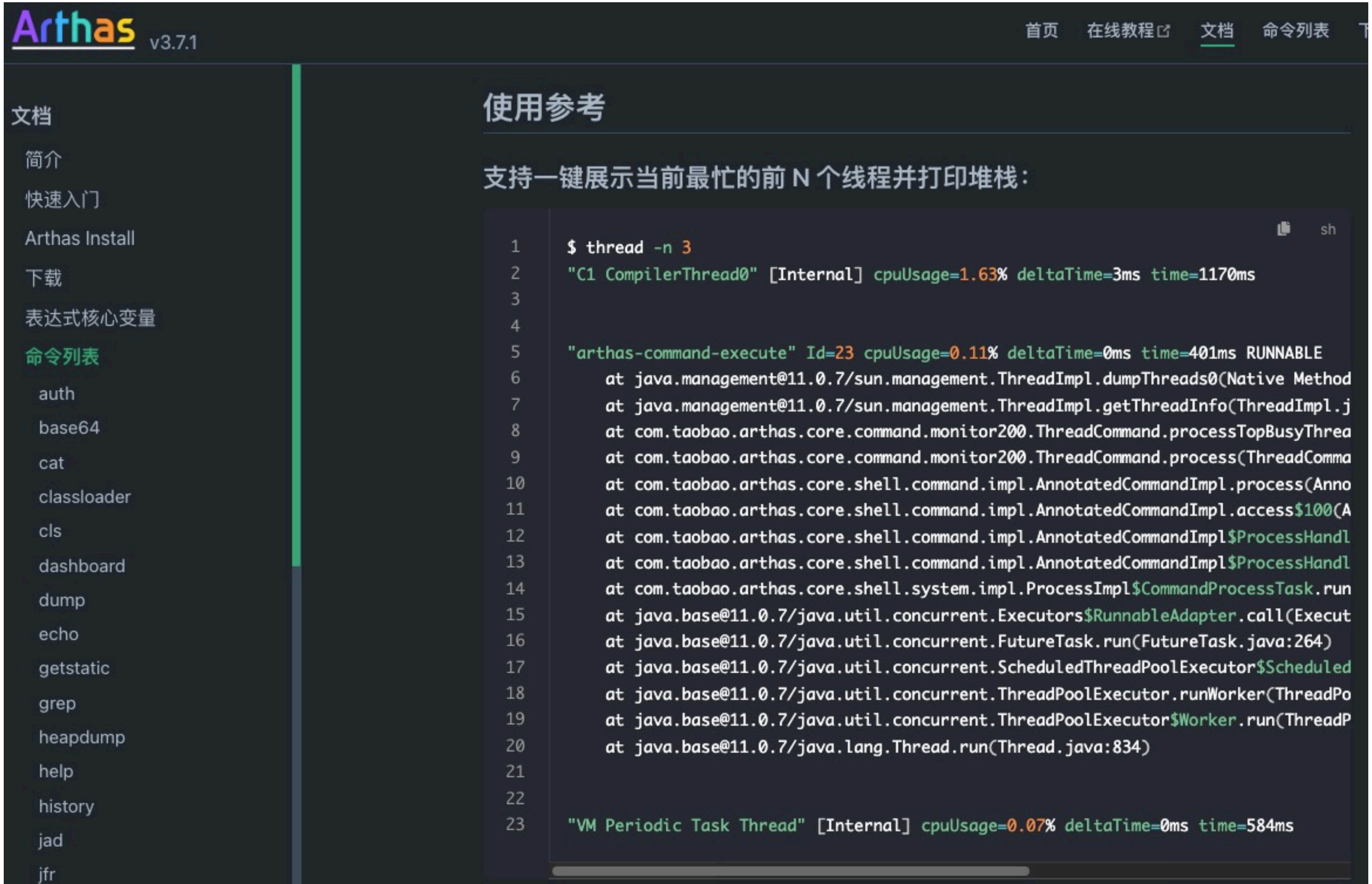
09-01 09:48 ~ 09-01 09:58

● [redacted]-base-86587b646-6p287:	5.3%
● [redacted]-base-86587b646-8bkf5:	4.3%
● [redacted]-base-86587b646-f45cv:	4.5%
● [redacted]-base-86587b646-f4qwx:	4.7%
● [redacted]-base-86587b646-fkhx5:	4.9%
● [redacted]-base-86587b646-fwd9c:	4.9%
● [redacted]-base-86587b646-j8kkm:	4.2%
● [redacted]-base-86587b646-krp5f:	4.9%
● [redacted]-base-86587b646-kxkfm:	4.8%
● [redacted]-base-86587b646-lvhc5:	4.9%
● [redacted]-base-86587b646-n7hzc:	105.7%
● [redacted]-base-86587b646-rfh57:	5.2%
● [redacted]-base-86587b646-vxlll:	5.6%
● [redacted]-base-86587b646-wntsv:	4.7%
● [redacted]pp-base-86587b646-wwzb8:	4.9%

内存 / 10分钟

[redacted]有一个节点 cpu 和 young gc 次数 遥遥领先，看起来很奇怪

How to diag. a CPU spike: Arthas



使用参考

支持一键展示当前最忙的前 N 个线程并打印堆栈：

```sh
$ thread -n 3
"C1 CompilerThread0" [Internal] cpuUsage=1.63% deltaTime=3ms time=1170ms


"arthas-command-execute" Id=23 cpuUsage=0.11% deltaTime=0ms time=401ms RUNNABLE
    at java.management@11.0.7/sun.management.ThreadImpl.dumpThreads0(Native Method
    at java.management@11.0.7/sun.management.ThreadImpl.getThreadInfo(ThreadImpl.j
    at com.taobao.arthas.core.command.monitor200.ThreadCommand.processTopBusyThrea
    at com.taobao.arthas.core.command.monitor200.ThreadCommand.process(ThreadComma
    at com.taobao.arthas.core.shell.command.impl.AnnotatedCommandImpl.process(Anno
    at com.taobao.arthas.core.shell.command.impl.AnnotatedCommandImpl.access$100(A
    at com.taobao.arthas.core.shell.command.impl.AnnotatedCommandImpl$ProcessHandl
    at com.taobao.arthas.core.shell.command.impl.AnnotatedCommandImpl$ProcessHandl
    at com.taobao.arthas.core.shell.system.impl.ProcessImpl$CommandProcessTask.run
    at java.base@11.0.7/java.util.concurrent.Executors$RunnableAdapter.call(Execut
    at java.base@11.0.7/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base@11.0.7/java.util.concurrent.ScheduledThreadPoolExecutor$Scheduled
    at java.base@11.0.7/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPo
    at java.base@11.0.7/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadP
    at java.base@11.0.7/java.lang.Thread.run(Thread.java:834)


"VM Periodic Task Thread" [Internal] cpuUsage=0.07% deltaTime=0ms time=584ms
```

How to integrate SkyWalking with Arthas

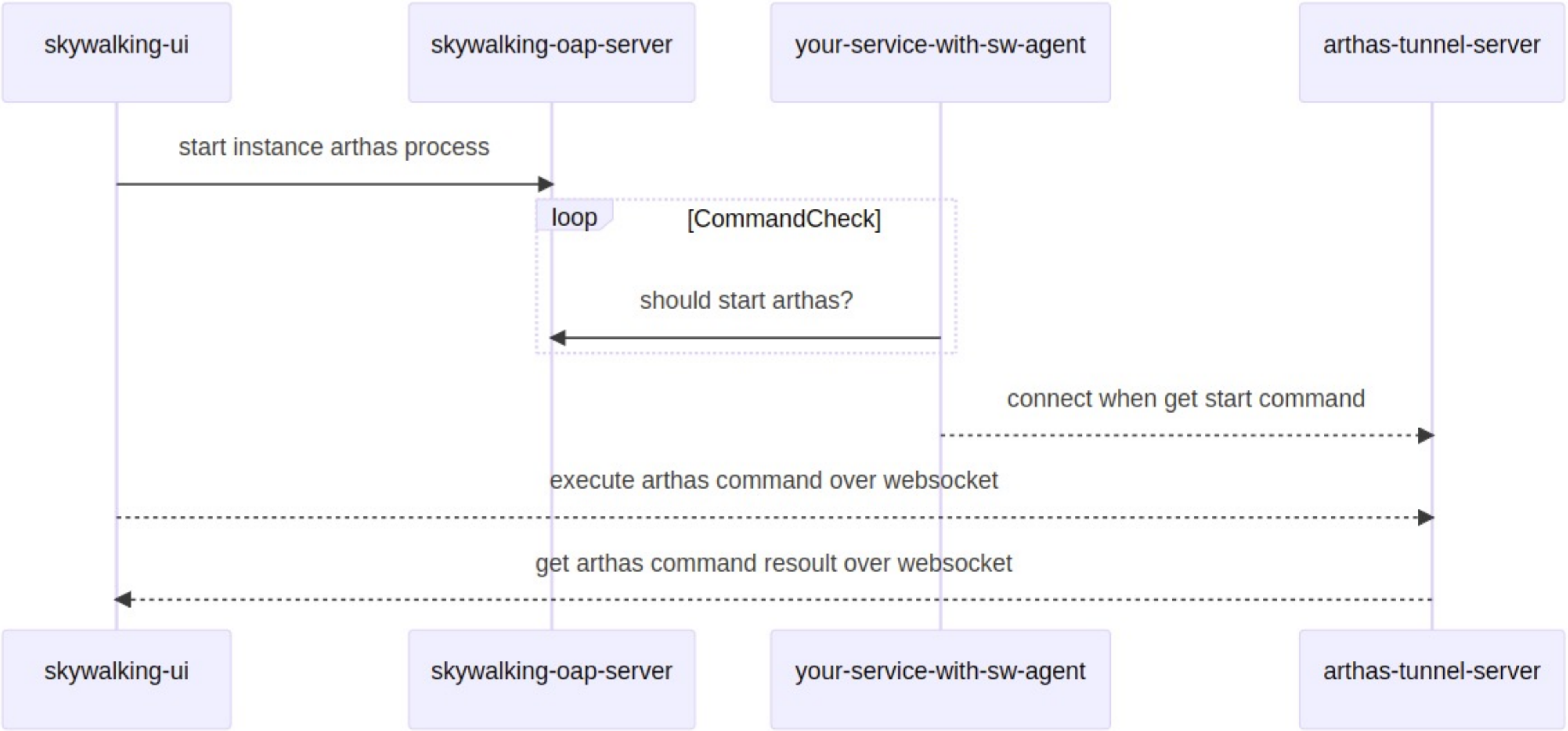
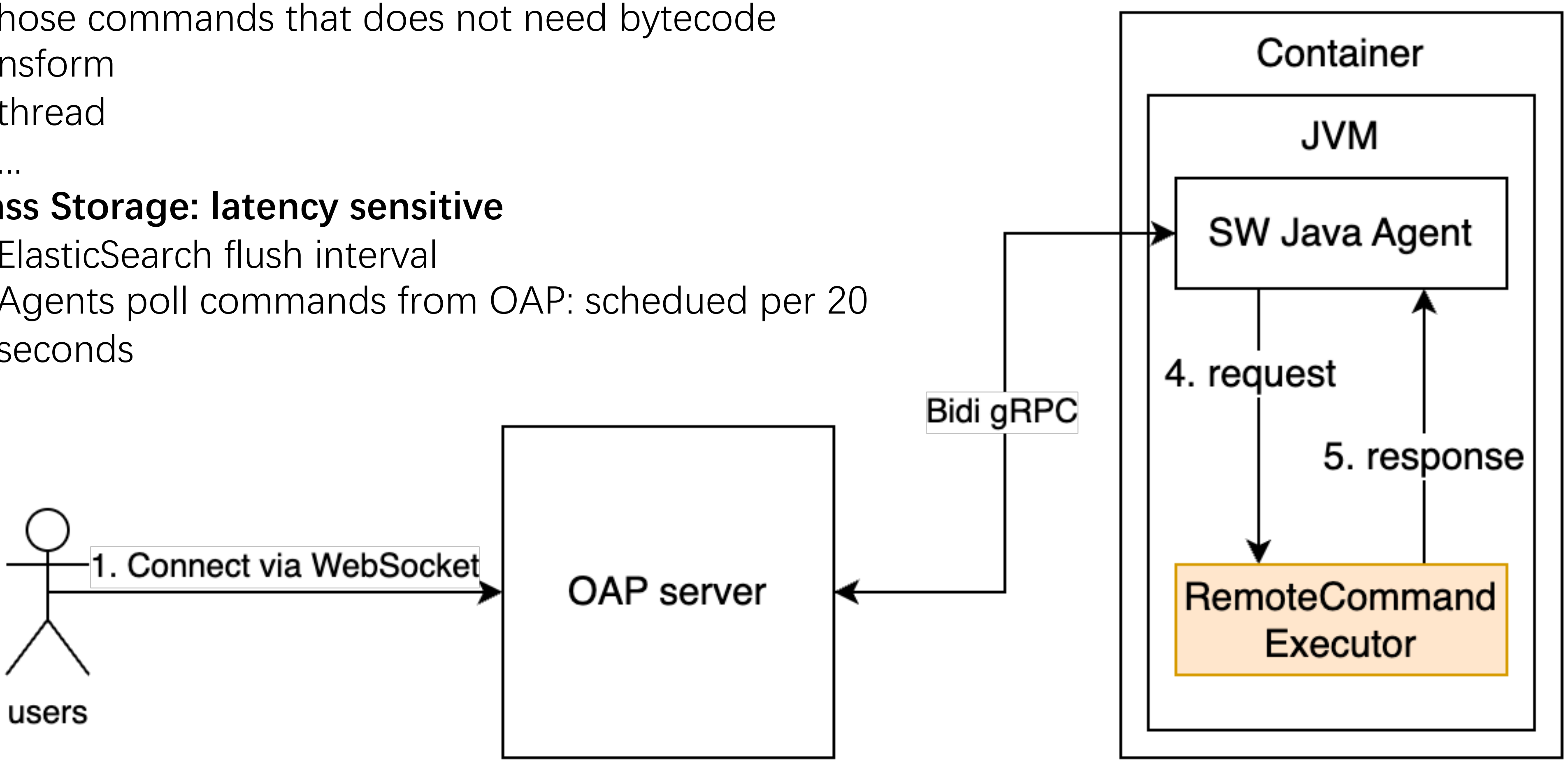

skywalking-ui
skywalking-oap-server
your-service-with-sw-agent
arthas-tunnel-server
start instance arthas process
loop
[CommandCheck]
should start arthas?
connect when get start command
execute arthas command over websocket
get arthas command resoult over websocket
skywalking-ui
skywalking-oap-server
your-service-with-sw-agent
arthas-tunnel-server

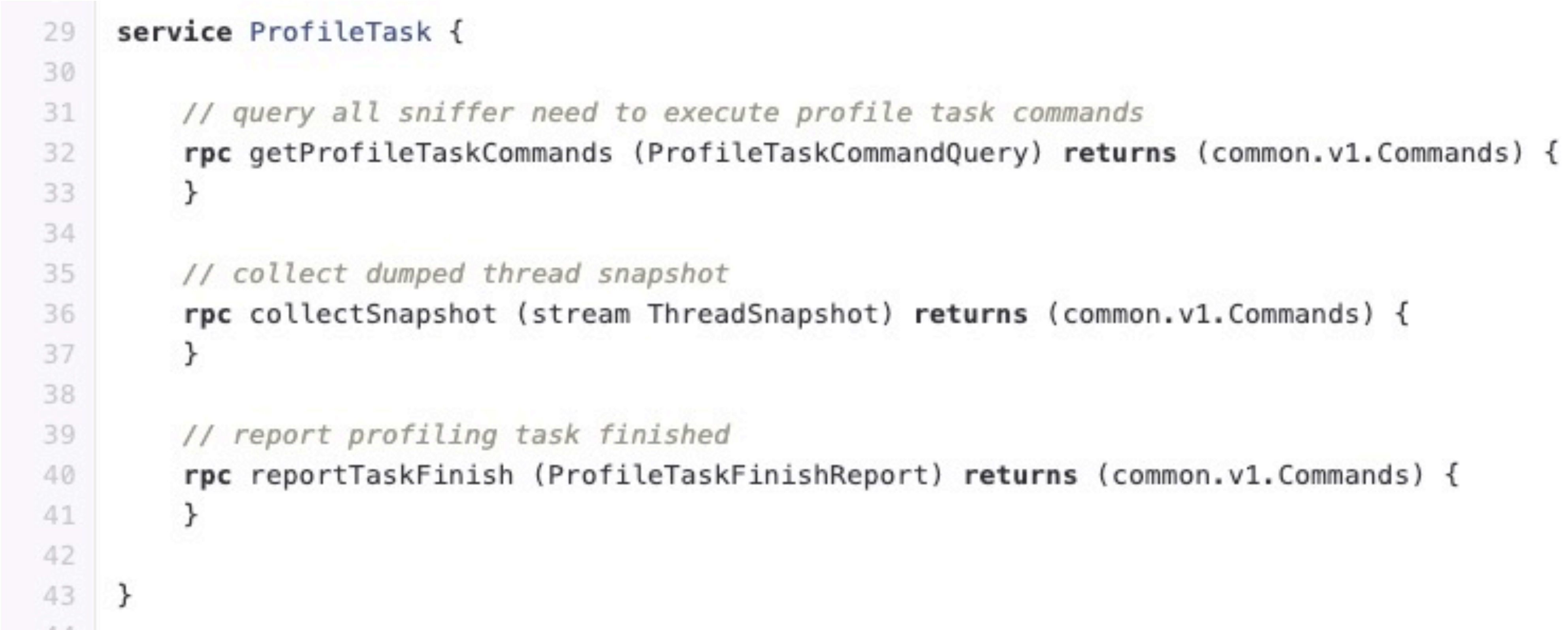
FIG. 将 Apache SkyWalking 与 Arthas 集成 By 魏翔
https://skywalking.apache.org/zh/2023-09-17-integrating-skywalking-with-arthas/

How to integrate SkyWalking with Arthas

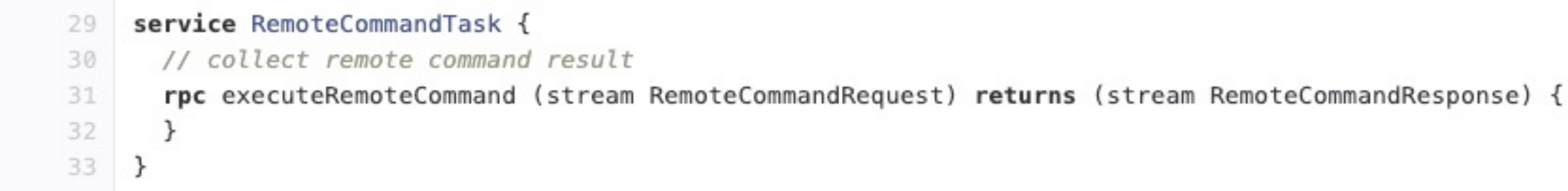
- For those commands that does not need bytecode retransform
  - thread
  - …
- **Bypass Storage: latency sensitive**
  - ElasticSearch flush interval
  - Agents poll commands from OAP: schedued per 20 seconds



Protocol Design: bidi over unary

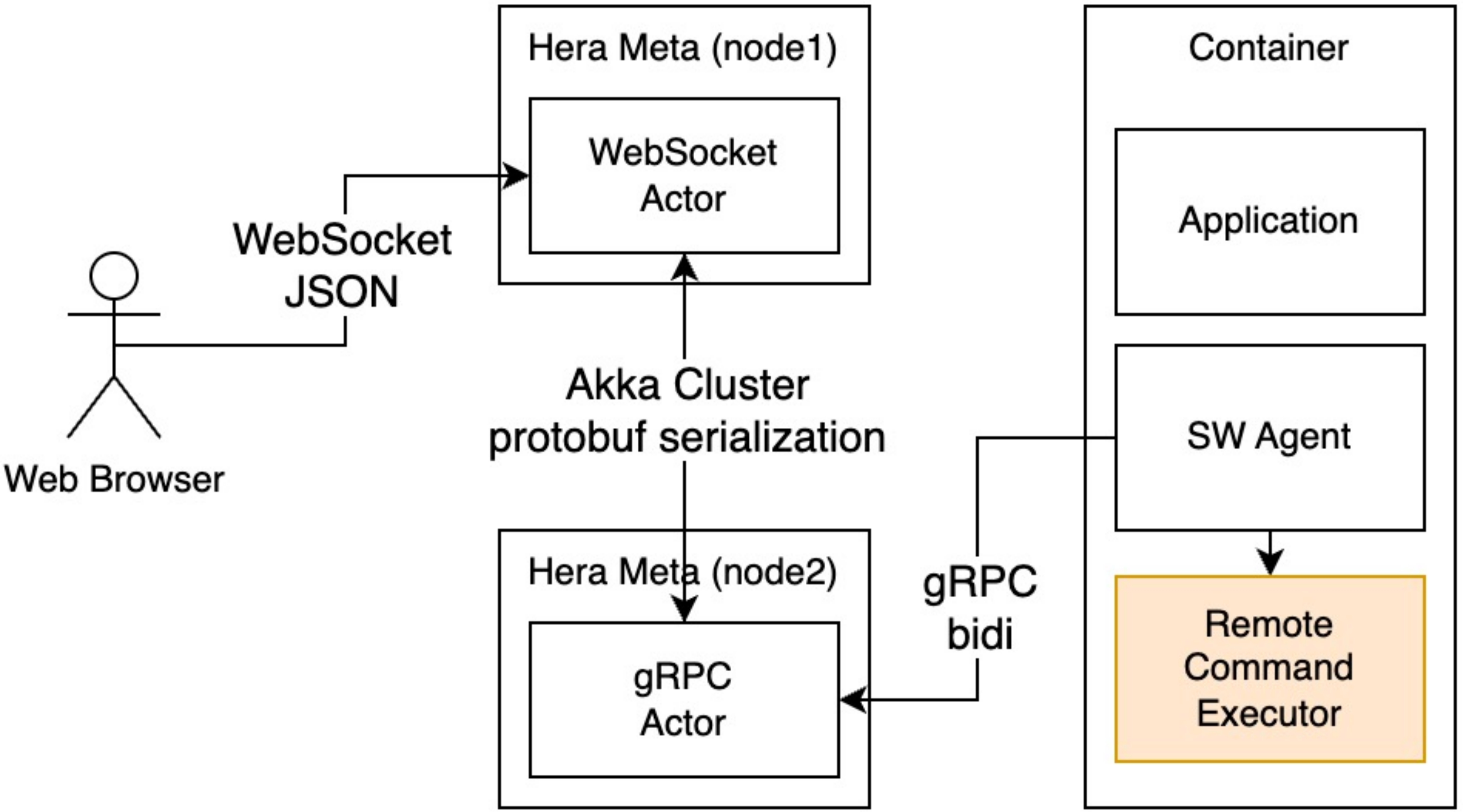


```
service ProfileTask {

    // query all sniffer need to execute profile task commands
    rpc getProfileTaskCommands (ProfileTaskCommandQuery) returns (common.v1.Commands) {
    }

    // collect dumped thread snapshot
    rpc collectSnapshot (stream ThreadSnapshot) returns (common.v1.Commands) {
    }

    // report profiling task finished
    rpc reportTaskFinish (ProfileTaskFinishReport) returns (common.v1.Commands) {
    }

}
```



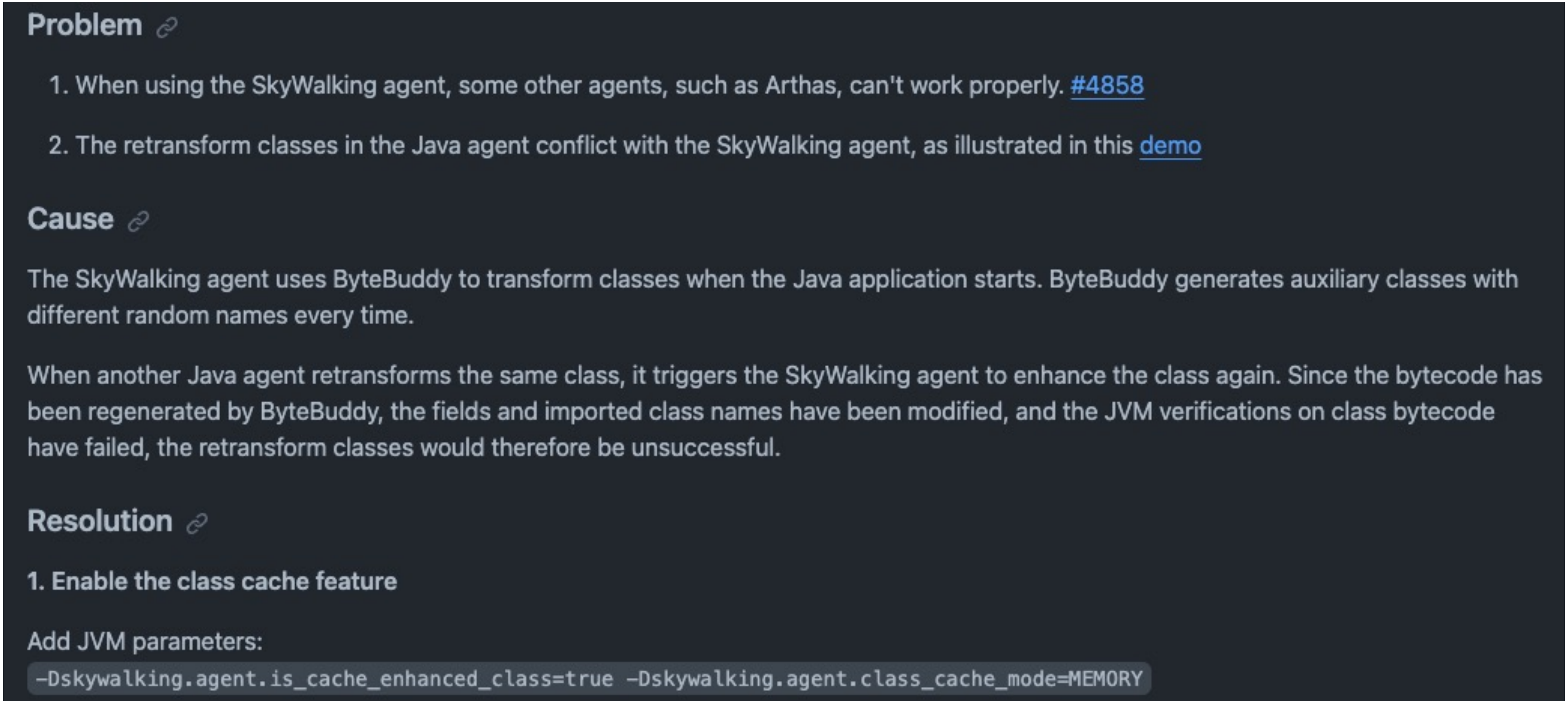
```
service RemoteCommandTask {
  // collect remote command result
  rpc executeRemoteCommand (stream RemoteCommandRequest) returns (stream RemoteCommandResponse) {
  }
}
```

What about distributed OAP?



What about retransform?

https://github.com/apache/skywalking/blob/master/docs/en/FAQ/Compatible-with-other-javaagent-bytecode-processing.md#compatibility-with-other-java-agent-bytecode-processes



Problem

1. When using the SkyWalking agent, some other agents, such as Arthas, can't work properly. #4858
2. The retransform classes in the Java agent conflict with the SkyWalking agent, as illustrated in this demo

Cause

The SkyWalking agent uses ByteBuddy to transform classes when the Java application starts. ByteBuddy generates auxiliary classes with different random names every time.

When another Java agent retransforms the same class, it triggers the SkyWalking agent to enhance the class again. Since the bytecode has been regenerated by ByteBuddy, the fields and imported class names have been modified, and the JVM verifications on class bytecode have failed, the retransform classes would therefore be unsuccessful.

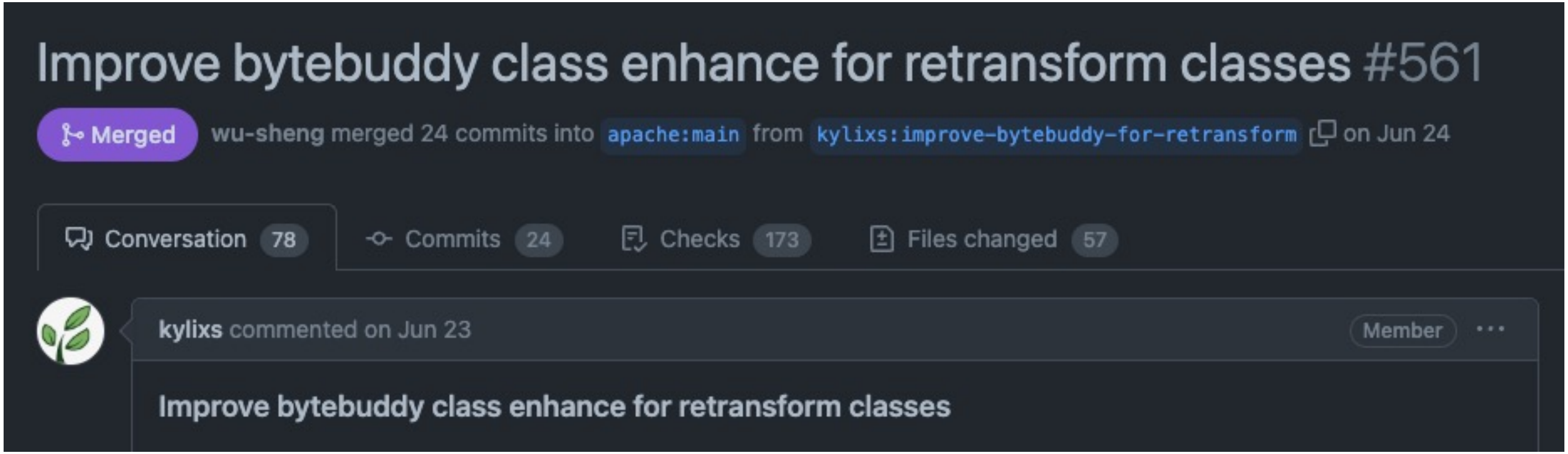
Resolution

1. Enable the class cache feature

Add JVM parameters:

```
-Dskywalking.agent.is_cache_enhanced_class=true -Dskywalking.agent.class_cache_mode=MEMORY
```

What about retransform? Changes in 9.0



Improve bytebuddy class enhance for retransform classes #561

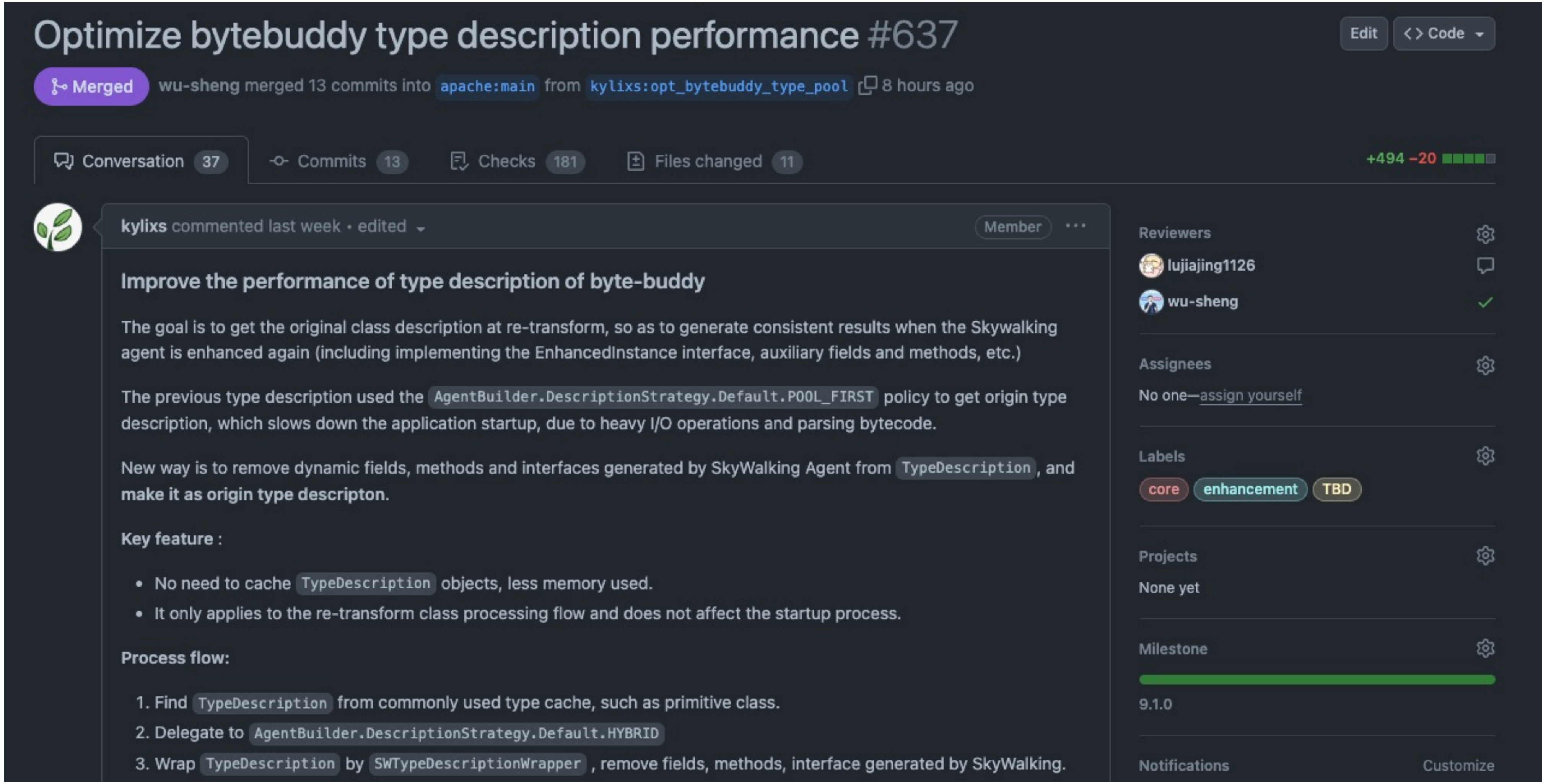
Merged wu-sheng merged 24 commits into `apache:main` from `kylixs:improve-bytebuddy-for-retransform` on Jun 24

kylixs commented on Jun 23 Member

Improve bytebuddy class enhance for retransform classes

- For those commands that does need bytecode retransform,
  - watch: observe method exec (parameter, result, exception…)
  - trace: trace method exec path
  - monitor: stat method exec (not real time)
- Main idea
  - For TypeDescription: always perfer bytecode from TypePool to reflection API
  - For aux. fields/methods: use stable prefix/suffix instead of random ones

Changes in 9.0: perf issue (resolved)



Optimize bytebuddy type description performance #637

Merged wu-sheng merged 13 commits into `apache:main` from `kylixs:opt_bytebuddy_type_pool` 8 hours ago

kylixs commented last week • edited

Improve the performance of type description of byte-buddy

The goal is to get the original class description at re-transform, so as to generate consistent results when the Skywalking agent is enhanced again (including implementing the EnhancedInstance interface, auxiliary fields and methods, etc.)

The previous type description used the `AgentBuilder.DescriptionStrategy.Default.POOL_FIRST` policy to get origin type description, which slows down the application startup, due to heavy I/O operations and parsing bytecode.

New way is to remove dynamic fields, methods and interfaces generated by SkyWalking Agent from `TypeDescription`, and **make it as origin type descripton.**

Key feature :

- No need to cache `TypeDescription` objects, less memory used.
- It only applies to the re-transform class processing flow and does not affect the startup process.

Process flow:

1. Find `TypeDescription` from commonly used type cache, such as primitive class.
2. Delegate to `AgentBuilder.DescriptionStrategy.Default.HYBRID`
3. Wrap `TypeDescription` by `SWTypeDescriptionWrapper` , remove fields, methods, interface generated by SkyWalking.

Reviewers
lujiajing1126
wu-sheng ✓

Assignees
No one—assign yourself

Labels
core enhancement TBD

Projects
None yet

Milestone
9.1.0

Notifications Customize

FIG. Using POOL_FIRST TypeDescription strategy in SW Java 9.0 caused almost double application launch time and larger heap size. Resolved in PR #637.

Q&A

欢迎提问交流

（仅限2位提问）

POWERED BY Skywalking APACHE

2023 SUMMIT | APACHE • SkyWalking
CHINA · SHANGHAI

2023 • SkyWalking Summit

感谢您的观看

纵目 tetrate