JMC/JFR: Kotlin spezial

Profiling/Monitoring with joy

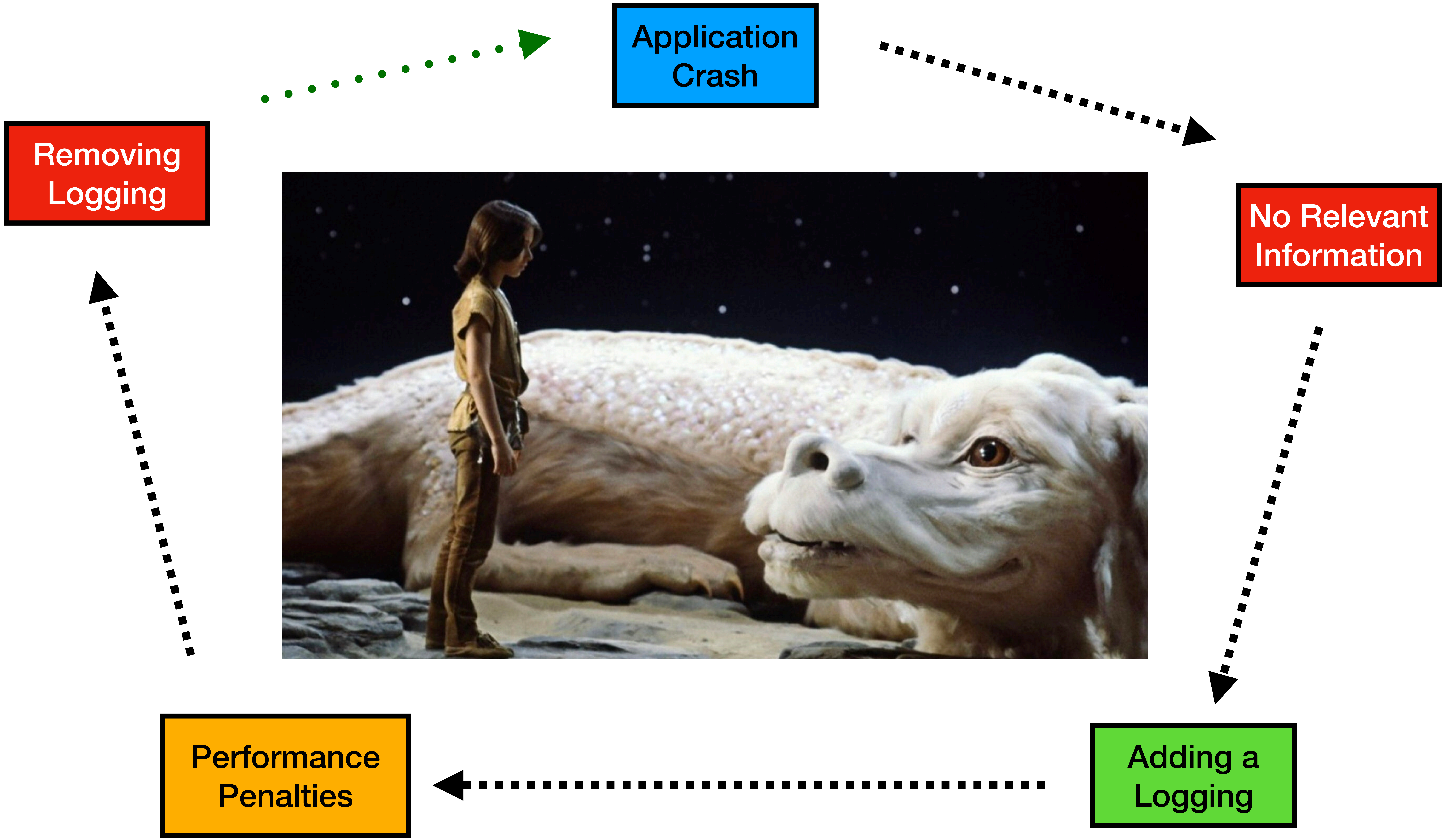
Miroslav Wengner

Safe Harbour Statement

All what you will hear can be different, this presentation is for motivational purposes …

Miroslav Wengner

- Husband, Father, Software Engineer, Technology Enthusiast
- OpenJDK Committer , Java Mission Control Project
- Co-Author of Robo4J Project (Duke Award)
- Contributor to other open-source projects
- Java Champion, JavaOne RockStar



Application Crash

No Relevant Information

Adding a Logging

Performance Penalties

Removing Logging

Agenda

- Brief history
- JFR in bullet points
- JFR fundamentals / Under the hood
- Performance (why overhead 1%)
- **DEMOS** : Java **“vs.”** Kotlin
  - *HotMethods, GC, Latencies and more…*
- Q/A

Brief history : back in time

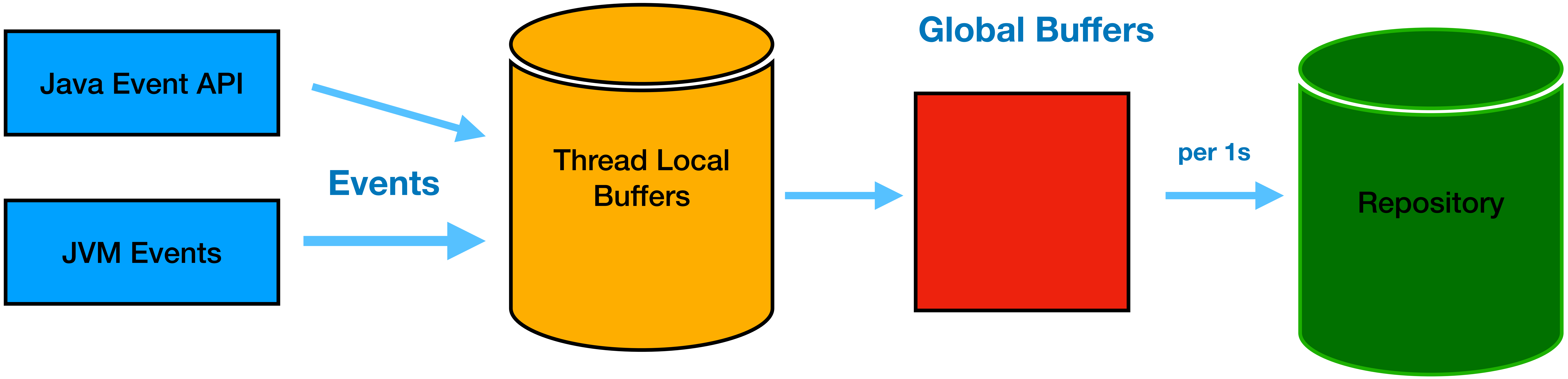
- 1998 Appeal Virtual Machines (AVM) - JRockit JVM
- 2002 AVM acquired by BEA
- 2008 acquired by Oracle
- 2012 JDK 7u4 update: Oracle integrated JFR into the HotSpot
- 2017 JDK 9 : Public APIs for creating and consuming data
- 2018 JDK 11: JMC/JFR announced to be fully Open-Sourced

JFR in bullets

- Java Flight Recorder is an **event based tracing framework**
- **Build** directly **into** the Java Virtual Machine
- Provides **access to** all **internal** events
- **Allows** to create **custom** events
- **Tries** to achieve a goal **1%** overhead

JFR Under the Hood

Event life cycle



JFR: Event - fundamental element

- Import “**jdk.jfr.Event**”
- Basic Element that caries valuable information

EventID:
Timestamp : when event was taken

Duration: not always

Thread ID: Thread where event has occurred

StackTrace ID: It’s optional referst to the StackTrace, default depth 64

Payload: custom information

```
Import jdk.jfr.Event;

public class SampleEvent extends Event {
   //internal logic
   String message;
}
```

```
…
void someAdvanceLogic() {

      SampleEvent e = new SampleEvent();

      e.message = “Important Information”;
      e.begin();

      // advanced logic

      e.end();
      e.commit();
}
…
```

Event: Tuning up

```
import jdk.jfr.Event
import jdk.jfr.Label
import jdk.jdf.Name

@Name(“com.openvalue.events.SampleEvent”)
@Label(“Sample Event”)
class SampleEvent extends Event {

    @Label(“Message”)
    String name;

    @Label(“Value”)
    int value;
}
```

Annotations: https://docs.oracle.com/en/java/javase/16/docs/api/jdk.jfr/jdk/jfr/class-use/MetadataDefinition.html

Event: Tuning up Kotlin way

```
...
import jdk.jfr.Category
import jdk.jfr.Description
import jdk.jfr.Event
import jdk.jfr.Label

….
@Label("Latency-Worker-SampleKotlinEvent")
@Category("Latency_Example")
@Description("something is done")
class SampleKotlinEvent() : Event()

...
```

JFR: Performance

- Usage of Thread Local Buffers
- Java Platform Optimization
- Methods: *INLINING*, CODE *ELIMINATION*, *SCALARIZATION*

- What happens when event is **enabled** / **disabled**

```
void someAdvanceLogic() {
    SampleEvent e = new SampleEvent();
    e.message = “Important Information”;

    e.begin();

    // advanced logic

    e.commit();
}
```

```
void commit() {
    // IF it’s not enabled -> NOT INTERESTING
    if(isEnabled()){
        //now() reads CPU clock register, cheap check
        long duration = now() - startTime;
        if(duration > THRESHOLD) {
            if (shouldCommit()) {
                // Cheap - Thread local writes
                actuallyCommit();
            }
        }
    }
}
```

Mikeal Vidstedt presented quite neat pseudo-code that helps to understand to the commit()

JFR: Enabled

```
void someAdvanceLogic(){
    // allocating event
    SampleEvent e = new SampleEvent();

    e.begin();   -> INLINING => e.startTime = now(); ->  e.startTime = <JVM intrinsic>

    // advanced logic

    // timestamp, likewise INLINING, implicit end()
    e.commit();

    // JFR ENABLED STATE
    if(e.isEnabled()){
        // perform additional checks and maybe actuallyCommit()
    }
}
```

JFR: Disabled - part 1

```
void someAdvanceLogic() {
    SampleEvent e = new SampleEvent();

    // INLINING from the previous slide
    e.startTime = <JVM intrinsic>;

    // advanced logic

    //INLINING
    if(false) {          // result e.isEnabled()
        //perform additional checks
        //CODE ELIMINATION -> will be removed
    }
}
```

JFR: Disabled - part 2

```
void someAdvanceLogic() {

   SampleEvent e = new SampleEvent();  //  SCALARIZATION  -> REMOVAL


   e.begin()
    1. initial state:      e.begin();
    2. INLINING =>  e.startTime = <JVM intrinsic>;
    3. INLINING =>  long startTime = <JVM intrisince>;
    4. CODE ELIMINATION => long startTime = <JVM intrisince>;  REMOVAL

   //business logic

}
```

JFR: Disabled - part 3

```
void someAdvanceLogic() {
        //business logic
}
```

JFR: Data Visualistion

- Command line tool avaliable from JDK 11 => **jfr**

```
$jfr summary <JFR_file>
$jfr print —json <JFR_FILE>
```

- JFR GUI. (**DEMO**)
  - Automated analysis
  - Java Application => Thread, Memory, etc.
  - Event Browser

Java Mission Control Project

Current release 8.1

Release	Milestone	Date
==========	===========	==========
8.1.0	GA	2021-08-02
8.2.0	RDS	2021-11-24
8.2.0	RDS 2	2021-12-22
8.2.0	GA	2022-01-19

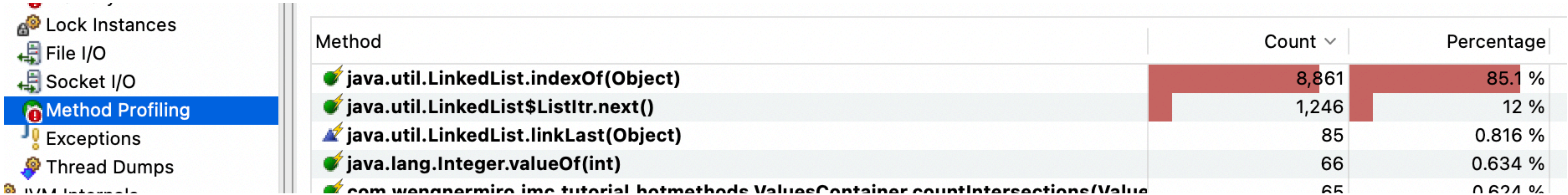
- New Allocation Events for JDK 16
- JMC Agent, JMC Agent Plugin
- Performance Improvements : Perser, Rules
- Many others

DEMOS: Agenda

- Profiling equivalent solution in Java “vs.” Kotlin
- examples:
  - **Hot-Methods (SHOW TIME)**
  - Garbage Collection (SHOW TIME)
  - **Latency (SHOW TIME)**
    - JMC Agent + JMC Agent Plugin

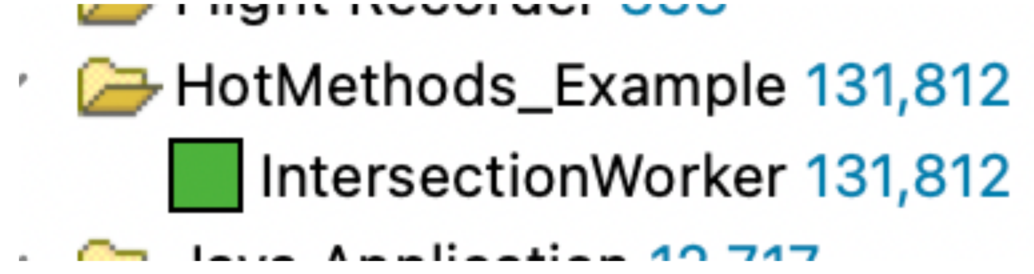
https://developers.redhat.com/blog/2020/10/29/collect-jdk-flight-recorder-events-at-runtime-with-jmc-agent#introduction_to_the_jmc_agent_plugin

DEMO: Hot-Methods



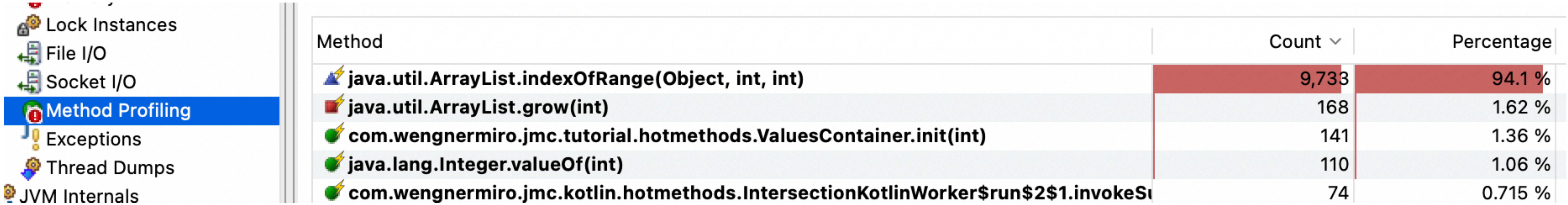
- Lock Instances
- File I/O
- Socket I/O
- Method Profiling
- Exceptions
- Thread Dumps

Method	Count	Percentage
java.util.LinkedList.indexOf(Object)	8,861	85.1 %
java.util.LinkedList$ListItr.next()	1,246	12 %
java.util.LinkedList.linkLast(Object)	85	0.816 %
java.lang.Integer.valueOf(int)	66	0.634 %
com.wengnermiro.jmc.tutorial.hotmethods.ValuesContainer.countIntersections(Value	65	0.624 %



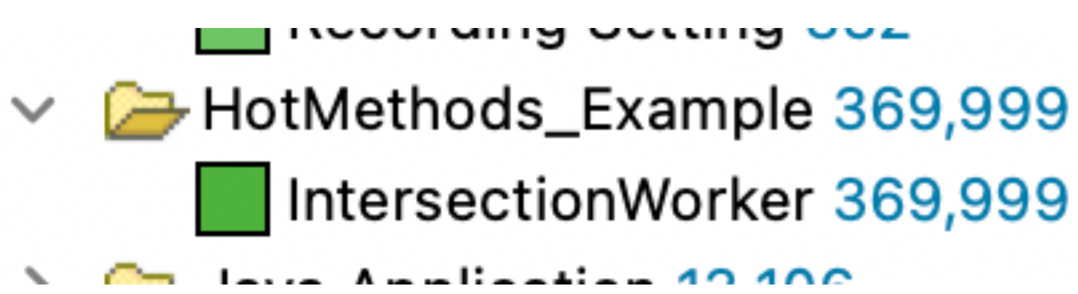
HotMethods_Example 131,812

🟩 IntersectionWorker 131,812



- Lock Instances
- File I/O
- Socket I/O
- Method Profiling
- Exceptions
- Thread Dumps
- JVM Internals

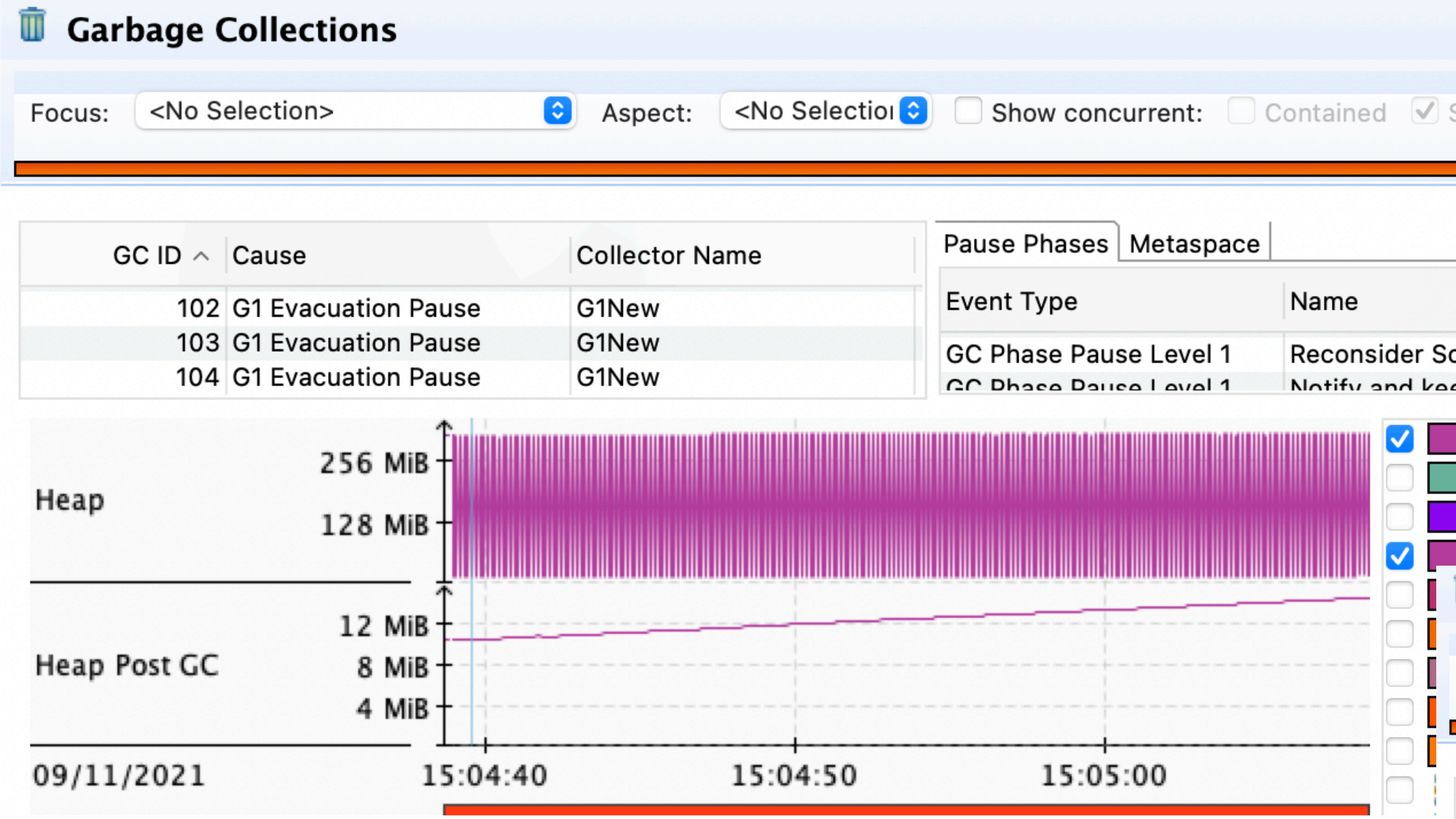
Method	Count	Percentage
java.util.ArrayList.indexOfRange(Object, int, int)	9,733	94.1 %
java.util.ArrayList.grow(int)	168	1.62 %
com.wengnermiro.jmc.tutorial.hotmethods.ValuesContainer.init(int)	141	1.36 %
java.lang.Integer.valueOf(int)	110	1.06 %
com.wengnermiro.jmc.kotlin.hotmethods.IntersectionKotlinWorker$run$2$1.invokeSı	74	0.715 %



HotMethods_Example 369,999

🟩 IntersectionWorker 369,999

DEMO: Garbage-Collection



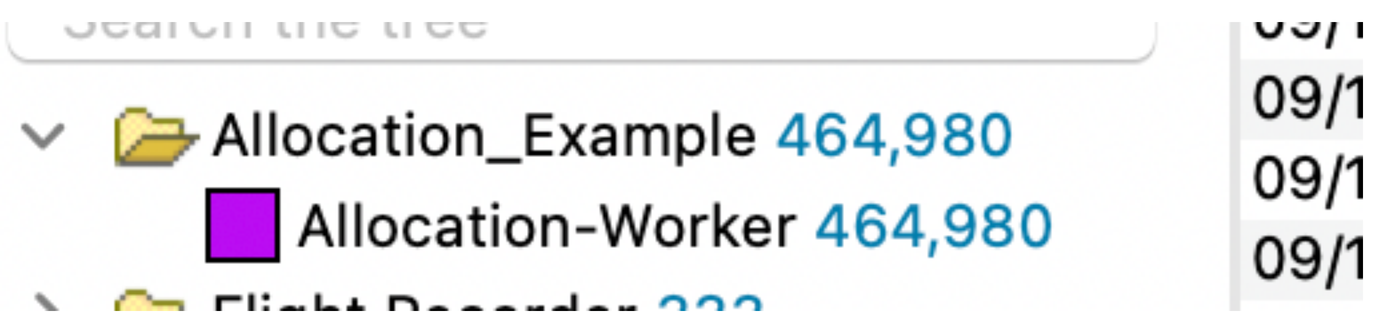
Garbage Collections

Focus: <No Selection> Aspect: <No Selectioi ☐ Show concurrent: ☐ Contained ☑ S

GC ID	Cause	Collector Name
102	G1 Evacuation Pause	G1New
103	G1 Evacuation Pause	G1New
104	G1 Evacuation Pause	G1New

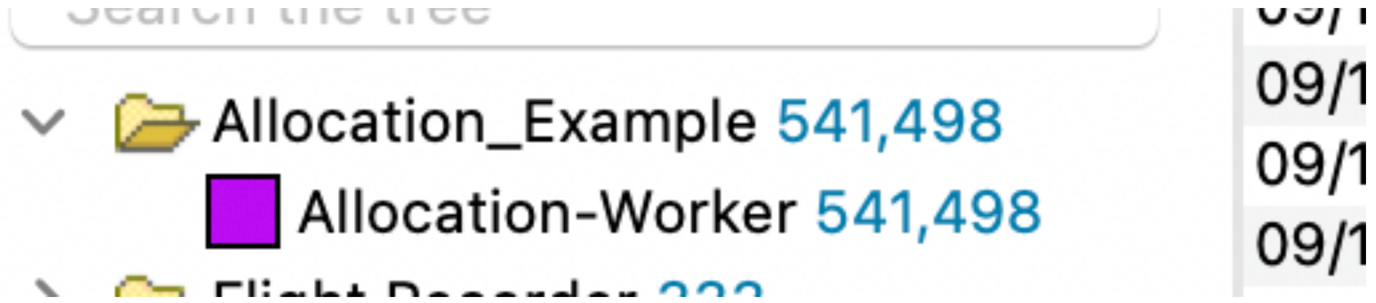
Pause Phases | Metaspace

Event Type	Name
GC Phase Pause Level 1	Reconsider Sc
GC Phase Pause Level 1	Notify and kee



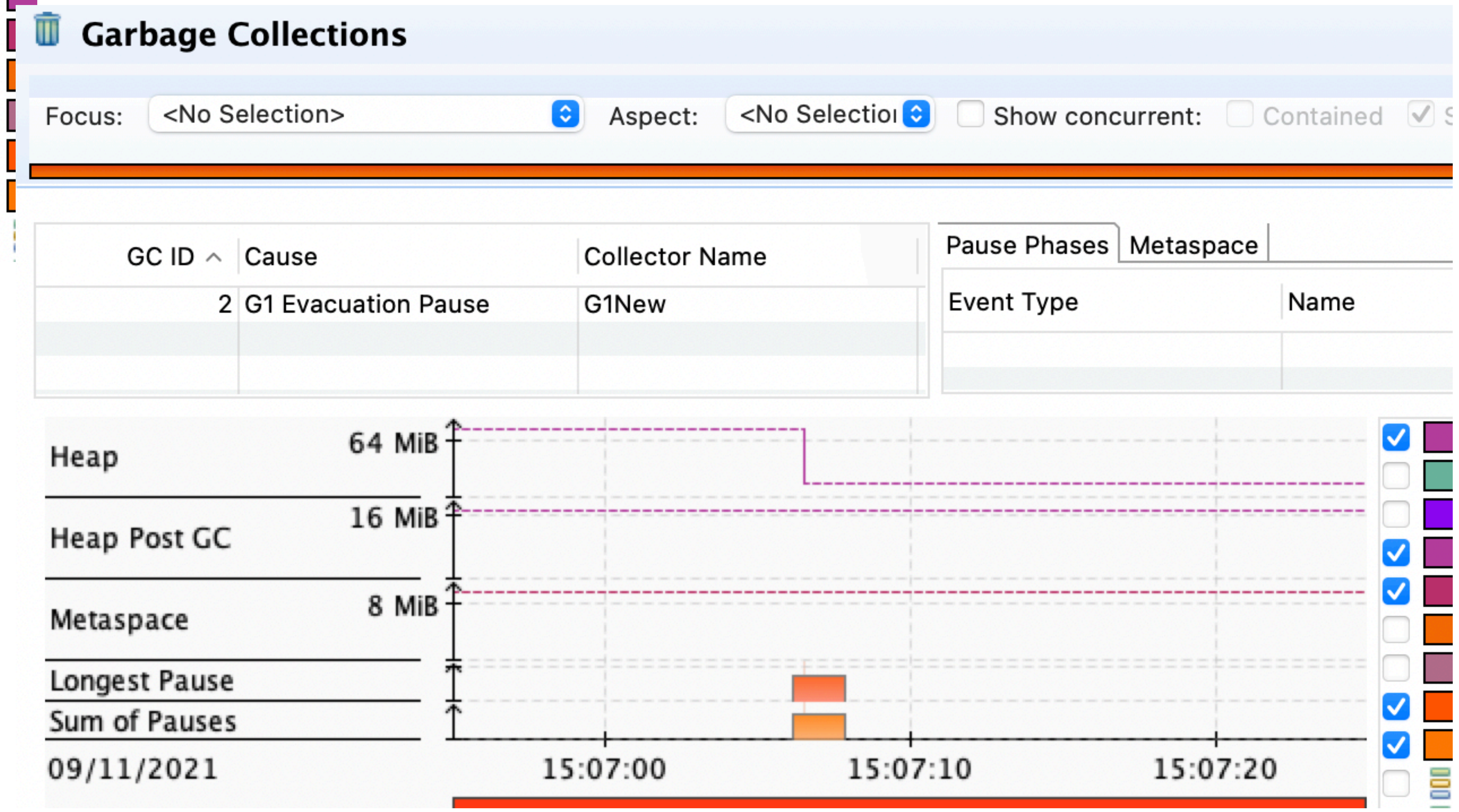
Search the tree

- Allocation_Example 464,980
  - Allocation-Worker 464,980
- Flight Recorder 222



Search the tree

- Allocation_Example 541,498
  - Allocation-Worker 541,498
- Flight Recorder 222



Garbage Collections

Focus: <No Selection> Aspect: <No Selectioi ☐ Show concurrent: ☐ Contained ☑ S

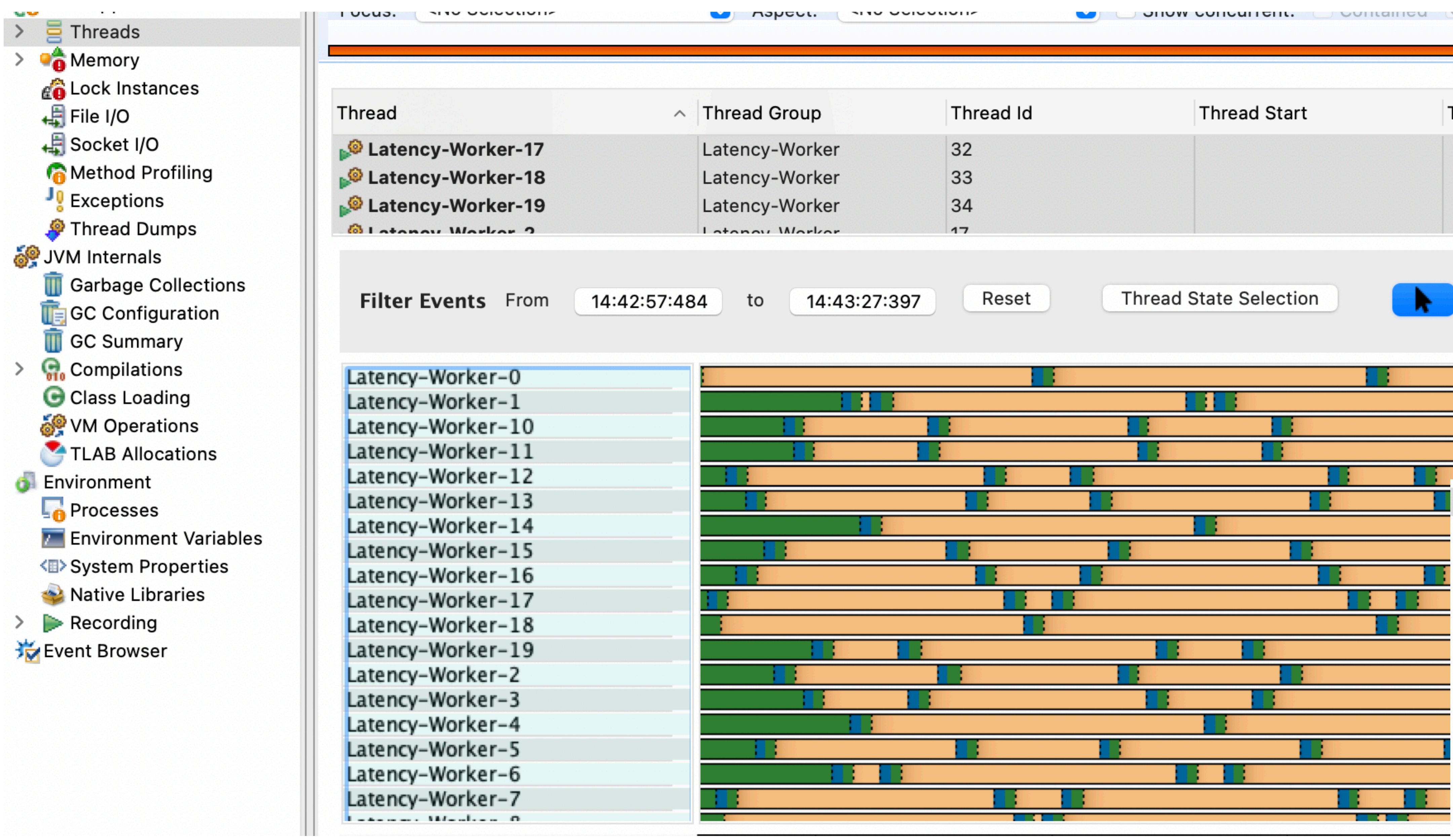
GC ID	Cause	Collector Name
2	G1 Evacuation Pause	G1New

Pause Phases | Metaspace

Event Type	Name

DEMO: Latency

OPEN VALUE



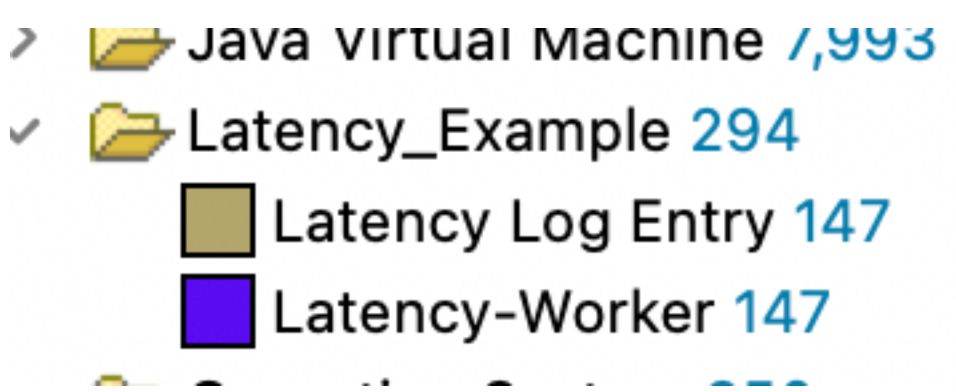
Focus: <No Selection> Aspect: <No Selection> Show concurrent: Contained Same threads Time Rang

- Threads
- Memory
- Lock Instances
- File I/O
- Socket I/O
- Method Profiling
- Exceptions
- Thread Dumps
- JVM Internals
- Garbage Collections
- GC Configuration
- GC Summary
- Compilations
- Class Loading
- VM Operations
- TLAB Allocations
- Environment
- Processes
- Environment Variables
- System Properties
- Native Libraries
- Recording
- Event Browser

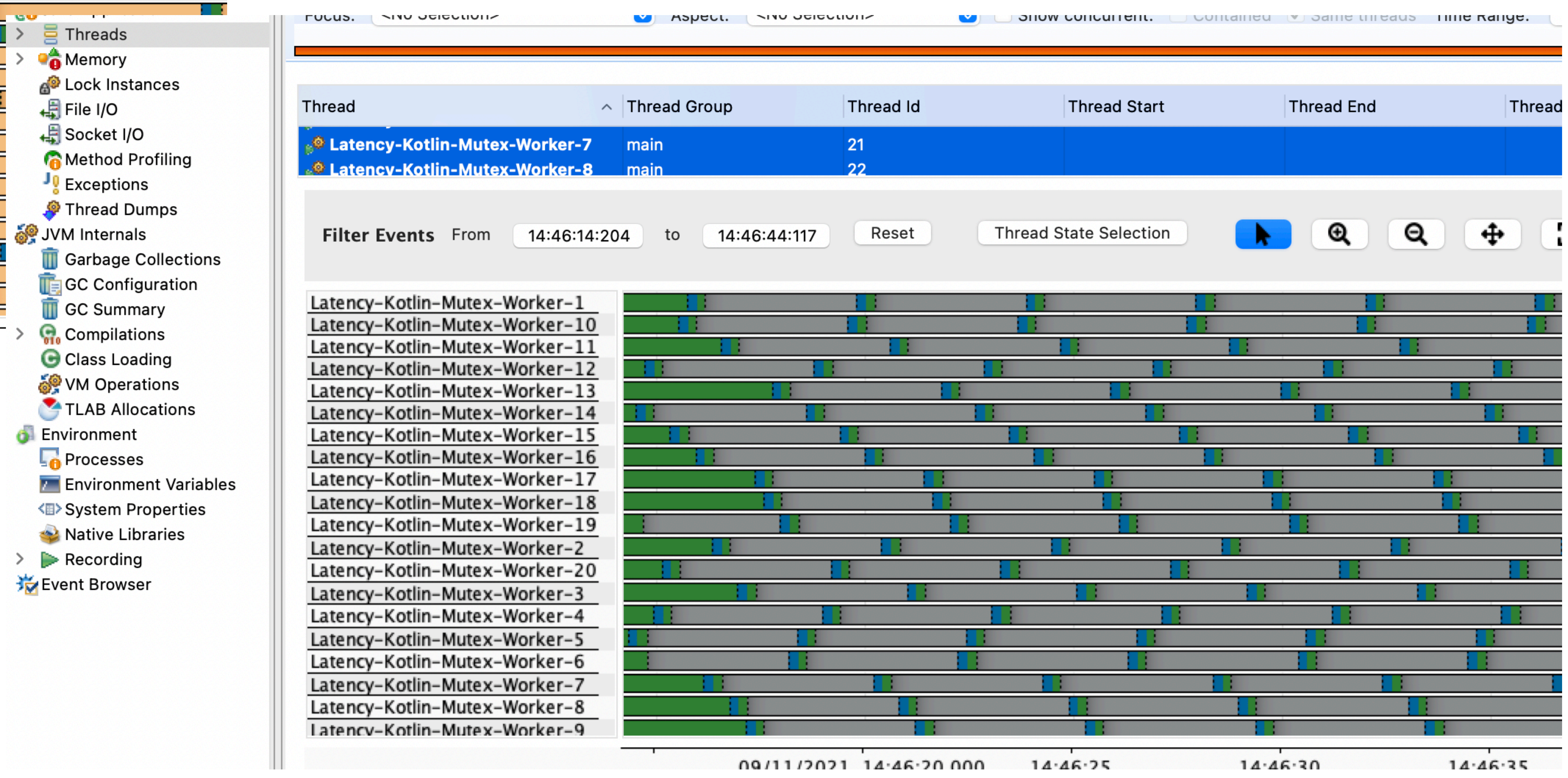
Thread	Thread Group	Thread Id	Thread Start	Thread End
Latency-Worker-17	Latency-Worker	32		
Latency-Worker-18	Latency-Worker	33		
Latency-Worker-19	Latency-Worker	34		
Latency-Worker-2	Latency-Worker	17		

Filter Events From 14:42:57:484 to 14:43:27:397 Reset Thread State Selection

Latency-Worker-0
Latency-Worker-1
Latency-Worker-10
Latency-Worker-11
Latency-Worker-12
Latency-Worker-13
Latency-Worker-14
Latency-Worker-15
Latency-Worker-16
Latency-Worker-17
Latency-Worker-18
Latency-Worker-19
Latency-Worker-2
Latency-Worker-3
Latency-Worker-4
Latency-Worker-5
Latency-Worker-6
Latency-Worker-7
Latency-Worker-8



- Java Virtual Machine 7,993
- ✓ Latency_Example 294
  - Latency Log Entry 147
  - Latency-Worker 147



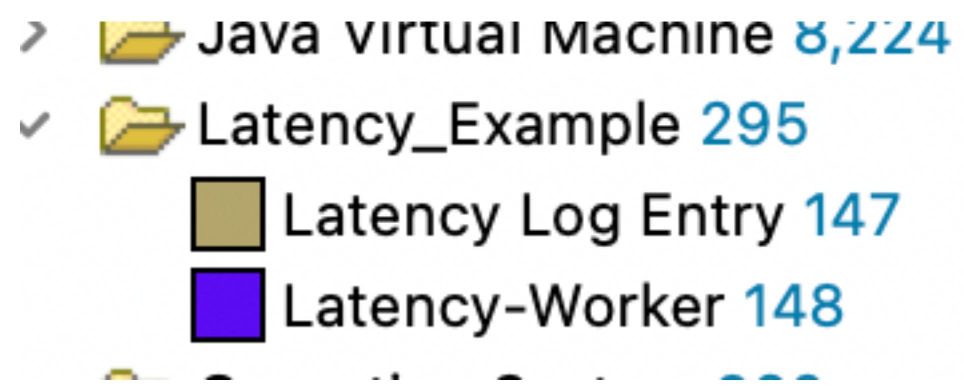
Focus: <No Selection> Aspect: <No Selection> Show concurrent: Contained Same threads Time Range:

- Threads
- Memory
- Lock Instances
- File I/O
- Socket I/O
- Method Profiling
- Exceptions
- Thread Dumps
- JVM Internals
- Garbage Collections
- GC Configuration
- GC Summary
- Compilations
- Class Loading
- VM Operations
- TLAB Allocations
- Environment
- Processes
- Environment Variables
- System Properties
- Native Libraries
- Recording
- Event Browser

Thread	Thread Group	Thread Id	Thread Start	Thread End	Thread
Latency-Kotlin-Mutex-Worker-7	main	21			
Latency-Kotlin-Mutex-Worker-8	main	22			

Filter Events From 14:46:14:204 to 14:46:44:117 Reset Thread State Selection

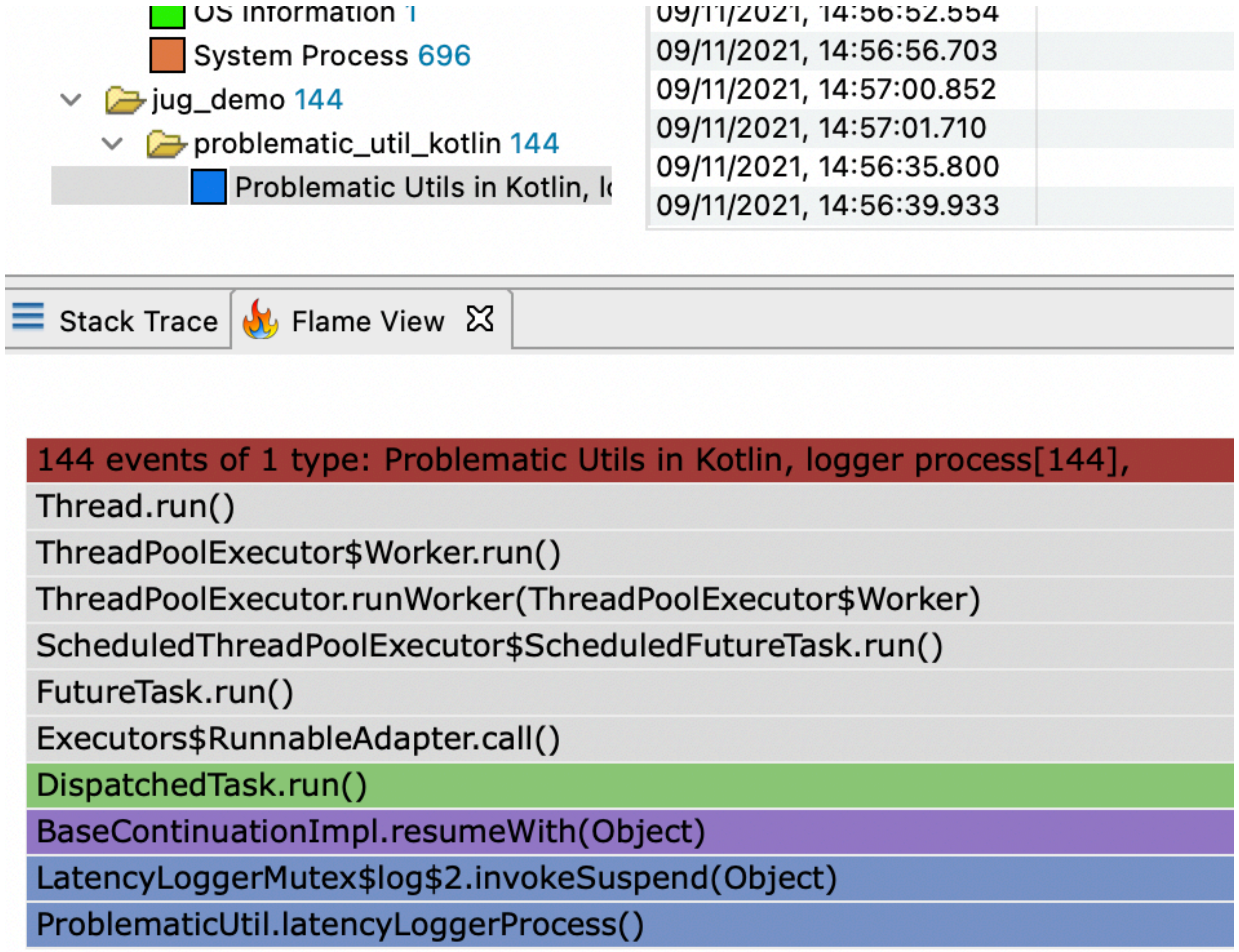
Latency-Kotlin-Mutex-Worker-1
Latency-Kotlin-Mutex-Worker-10
Latency-Kotlin-Mutex-Worker-11
Latency-Kotlin-Mutex-Worker-12
Latency-Kotlin-Mutex-Worker-13
Latency-Kotlin-Mutex-Worker-14
Latency-Kotlin-Mutex-Worker-15
Latency-Kotlin-Mutex-Worker-16
Latency-Kotlin-Mutex-Worker-17
Latency-Kotlin-Mutex-Worker-18
Latency-Kotlin-Mutex-Worker-19
Latency-Kotlin-Mutex-Worker-2
Latency-Kotlin-Mutex-Worker-20
Latency-Kotlin-Mutex-Worker-3
Latency-Kotlin-Mutex-Worker-4
Latency-Kotlin-Mutex-Worker-5
Latency-Kotlin-Mutex-Worker-6
Latency-Kotlin-Mutex-Worker-7
Latency-Kotlin-Mutex-Worker-8
Latency-Kotlin-Mutex-Worker-9



- Java Virtual Machine 8,224
- ✓ Latency_Example 295
  - Latency Log Entry 147
  - Latency-Worker 148

DEMO: Latency - Agent

- JMC-Agent v.1.0.0 for Java 11



🟩 OS Information 1 — 09/11/2021, 14:56:52.554
🟧 System Process 696 — 09/11/2021, 14:56:56.703
jug_demo 144 — 09/11/2021, 14:57:00.852
problematic_util_kotlin 144 — 09/11/2021, 14:57:01.710
🟦 Problematic Utils in Kotlin, lo — 09/11/2021, 14:56:35.800
09/11/2021, 14:56:39.933

```
144 events of 1 type: Problematic Utils in Kotlin, logger process[144],
Thread.run()
ThreadPoolExecutor$Worker.run()
ThreadPoolExecutor.runWorker(ThreadPoolExecutor$Worker)
ScheduledThreadPoolExecutor$ScheduledFutureTask.run()
FutureTask.run()
Executors$RunnableAdapter.call()
DispatchedTask.run()
BaseContinuationImpl.resumeWith(Object)
LatencyLoggerMutex$log$2.invokeSuspend(Object)
ProblematicUtil.latencyLoggerProcess()
```

JFR: How to Get

- **Clone**: https://github.com/openjdk/jmc. => script **build.sh**
- **AdoptOpenJDK:** http://adoptopenjdk.net/jmc
- **Azul**: https://www.azul.com/products/zulu-mission-control
- **RedHat**: distributes as RPMs in Fedora and RHEL
- **Oracle**: https://www.oracle.com/java/technologies/jdk-mission-control.html

JMC-JVM-LANG Tutorial: https://github.com/mirage22/jmc-jvm-lang-tutorial

JFR-Tutorial: https://github.com/thegreystone/jmc-tutorial

OPEN VALUE

Q / A

Thank YOU !

twitter: @miragemiko
gitlab:@mirage22

DEMOS: https://github.com/mirage22/jmc-jvm-lang-tutorial