Welcome to the session...

**The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.**

Basically, don't trust anything I say...

ORACLE

# Java Flight Recorder - Next generation diagnostics and profiling

Rickard Bäckman
HotSpot JVM Compiler Team, Java Platform

```
public class Example {
  public void saveTransaction(Transaction tn) {
    synchronized (someLock) {
      // save to some shared data structure
    }
  }
}
```

```
public class Example {
  public void saveTransaction(Transaction tn) {
    System.err.println(threadId + “Waiting for lock”);
    synchronized (someLock) {
      System.err.println(threadId + “Got lock”);
      // save to some shared data structure
    }
    System.err.println(threadId + “Lock released”);
  }
}
```

... few million lines of output
[T1] Waiting for lock
[T3] Waiting for lock
[T2] Waiting for lock
[T2] Got lock
[T1] Got lock
[T1] Lock released
[T3] Got lock
[T3] Lock released
[T2] Lock released
... another million lines of output

# Java Flight Recorder - Overview

## Released in 7u40

# Java Flight Recorder - Overview

# Java Flight Recorder - Overview

Event Recorder & Profiler

# Java Flight Recorder - Overview

Implemented in the JVM

# Java Flight Recorder - Overview

No restart required

# Java Flight Recorder - Overview

After-the-facts analysis

-

What went wrong?

# Event Recorder

JVM tracks lots of information

# Event Recorder

JVMTI?

# Event Recorder

Instrumented JVM & JDK

-

Make that information visible

ORACLE

# Profiler

What is my CPU doing?!?

# Inside the JVM

Core in the JVM
High-level things in Java

# Inside the JVM

Access to
JVM internals & subsystems

# Inside the JVM

Class Unloading
&
Safepoints

# Non-intrusive

Designed to run all the time.
In your production environment

# Non-intrusive

Low overhead
Captures information that is already there

# Non-intrusive

Enable & disable at runtime

# After-the-facts

Problem in the environment

# After-the-facts

What happened before?

# After-the-facts

Keeps history of information

# After-the-facts

Java Mission Control

-

Dump the data on an SLA breach

# Things JFR captures

Compilation statistics

Socket read/write

VM Operations

Context switches

Object Count

Exceptions

Class Unload

Garbage Collection

CPU Load

Method Compilation

Class Load

Object allocation

Threads

Locks

Metaspace Statistics

CodeCache

Profiling

Thread.sleep

File read/write

Garbage Collection statistics

ORACLE

# Sampling Profiler

How to implement a profiler?

# Sampling Profiler

JVMTI

# Sampling Profiler

Byte code instrumentation

# Sampling Profiler

Stop the thread,
walk the stack,
collect the methods.

# Demo

# Enable Java Flight Recorder

-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder

ORACLE

# Start recording

-XX:StartFlightRecording=filename=<...>,duration=5m

# Start on demand

Java Mission Control

or

jcmd <pid> JFR.start <options>

# Some interesting options

stackdepth=<nr>

dumponexit=<true/false>

# Events

Abstraction of captured information

# Events

Captures something interesting
Latency? Performance?

# Instant Events

Happens “instantly”

-

Thread start, Thread stop, Class unload

# Duration Events

Happens over a period of time

-

Compilation, Garbage Collection,
Wait for a lock

# Requestable Events

Something that should be done repeatedly

-

CPU Load, Profiling Events
Context switches

ORACLE

# Settings & Filtering

Enable & disable

# Settings & Filtering

Threshold

# Settings & Filtering

Frequency / Period

# Settings & Filtering

Stack traces

# Settings & Filtering

Control signal / noise

# Settings & Filtering

Filter early!

# Recording

"Start a recording"

# Recording

Generates a stream of events

# Recording

Applies a filter and captures the remaining

# Recording

One stream of Events

# Recording

Multiple recordings

# Recording

The Union of Events

# Profiles

A preconfigured settings for a recording

# Profiles

default & profile

# Profiles

Create your own

# Java Flight Recorder

Implementation

# Java Flight Recorder

Challenges

Assembly!

Implementation

File format

Events

C++

Low level

Memory buffers

# Challenges

Overhead must be low

# Challenges

Memory footprint

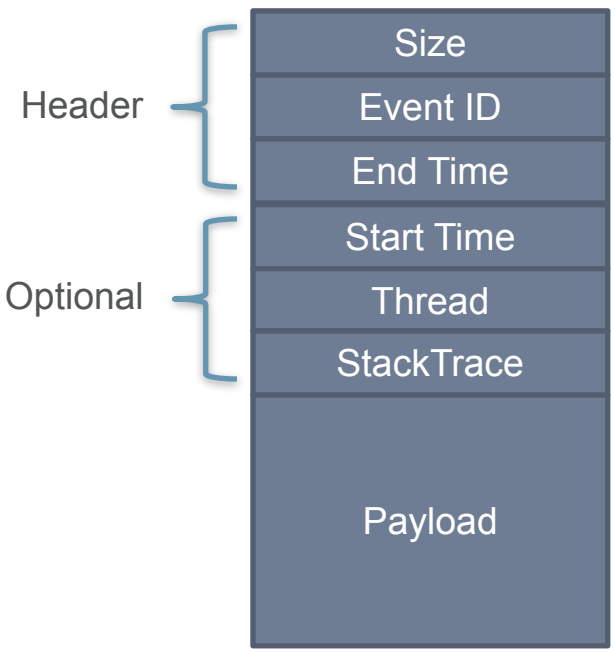
# Challenges

The JVM environment is changing

# Challenges

Scalability

# Events in detail



# Events in detail

Self-describing Events

# Event Metadata

Name, Path, Description

# Event Payload Metadata

Name, Type, Description
&
Content Type

# Content Type

## Semantics of a Value

| Content Type | Displayed as |
|---|---|
| Bytes | 42 MB |
| Percentage | 42% |
| Address | 0xDEADBEEF |
| Millis | 242 ms |

# Content Type

Can be multiple Values

| Content Type | Displayed |
| --- | --- |
| Class | Class Loader |
| | Name |
| | Modifiers (public,final) |

# Event definition in HotSpot

```
<event id="ThreadSleep"
  path="java/thread_sleep"
  label="Java Thread Sleep"
  has_thread="true"
  has_stacktrace="true"
  is_instant="false">
    <value type="MILLIS" field="time" label="Sleep Time"/>
</event>
```

What does it actually mean?
path is a identifier for the UI, label is a description of the Event
has_thread means each event is tied to a Thread
has_stacktrace means get a stack trace for the Event
is_instant decides whether it is instant event or not
Finally the payload with a type name and a description.

# Event in C++

```
EventThreadSleep event;
...
...
if (event.should_commit()) {
  event.set_time(millis);
  event.commit();
}
```

An example of an event in the HotSpot source code.
This event repots the time a thread slept.
Thread.sleep()

So the event is created and when the thread returns from sleep we check if the event was enabled and should be committed. If so we save the time and commit.

# Or in assembler

```
281: cmp  BYTE [rbp-0x60],0x0 # should commit
285: je      300              # jump if zero
287: mov  QWORD rbp-0x58],rbx  # save time
291: lea    rdi,[rbp-0x70] # get pointer
295: call   0x101b17dc4 # commit
```

This is basically the assembler code generated for the same code (for amd64). Reserve some memory on the stack, initialize it. Check if the event was enabled, otherwise jump away. Save the time, get the pointer and call commit.

The overhead of an event that is disabled is very small. Compare a value to zero and jump away.

# Buffers

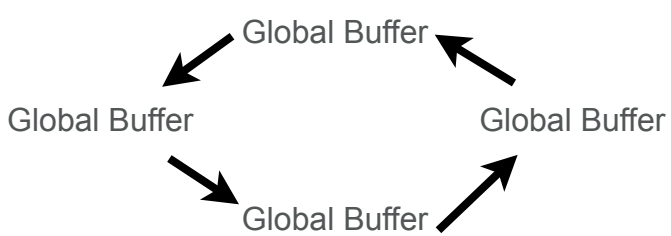
Thread-local buffers

# Buffers

Promoted to Global buffers

-

Global buffers are circular



# File Format

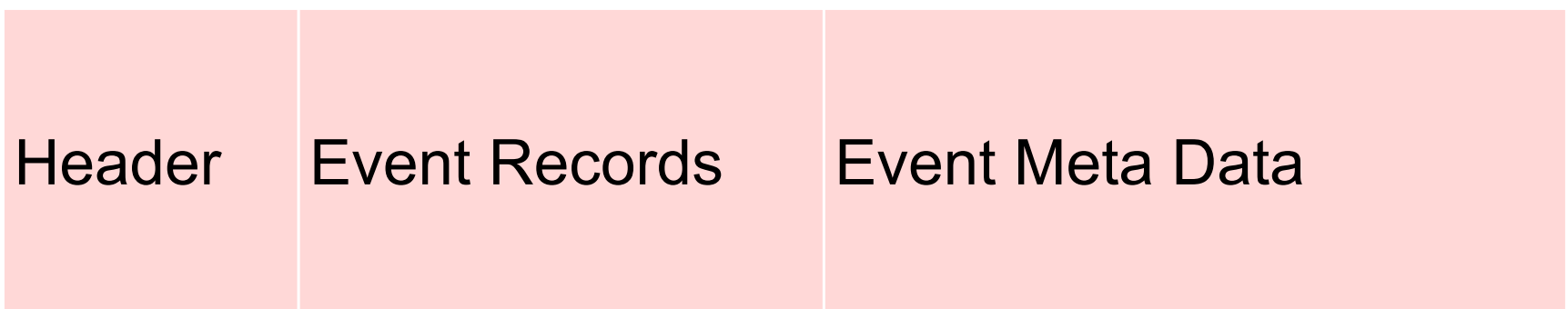
Binary proprietary

# File Format

Fast writing

# File Format

Self Contained



# Low-level

Enough High-Level
Problems & Solutions

# Starting Problem

Classes are referenced by Events

# Starting Problem

java.lang.String

# Starting Problem

org.springframework.security.ui.preauth.PreAuthenticatedGrantedAuthoritiesWebAuthenticationDetails

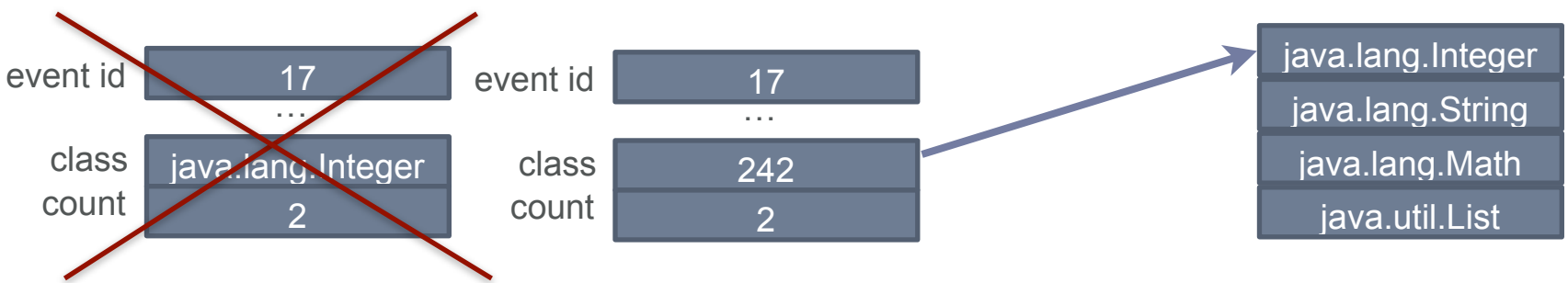
# Starting Problem

99 characters!

# Starting Problem

```
class Klass {
    …
    u8 _trace_id;
    …
}
```

## Introduce a unique ID



# New problem

Mapping

# Class List

Must be synchronized

# Class List

Class Load?

# Class List

Class Load?

-

Startup performance!

# Class List

End of a Recording?

# Class List

End of a Recording?

-

Classes can unload!

# Class List

Piggyback Class Unloading!

# Class List

The Class List can grow Big

# Class List

Lots of classes
+ Long class names
=
Lots of wasted memory

# Tagged Classes

Reserve a bit in the ID

# Tagged Classes

Tag referenced classes!

```
#define CLASS_USED 1

void use_class_id(Klass* const klass) {
  klass->_trace_id |= CLASS_USED;
}
```

# Class List

Classes come & go.
Don’t waste memory

# Checkpoints

Special Event

# Checkpoints

Writes the Class List,
Resets the tags &
clears the Class List

```
class_pool.lookup(242)
  → java.lang.Integer

method_pool.lookup(314)
  →
java.lang.Math:pow()
```

# Constant Pool

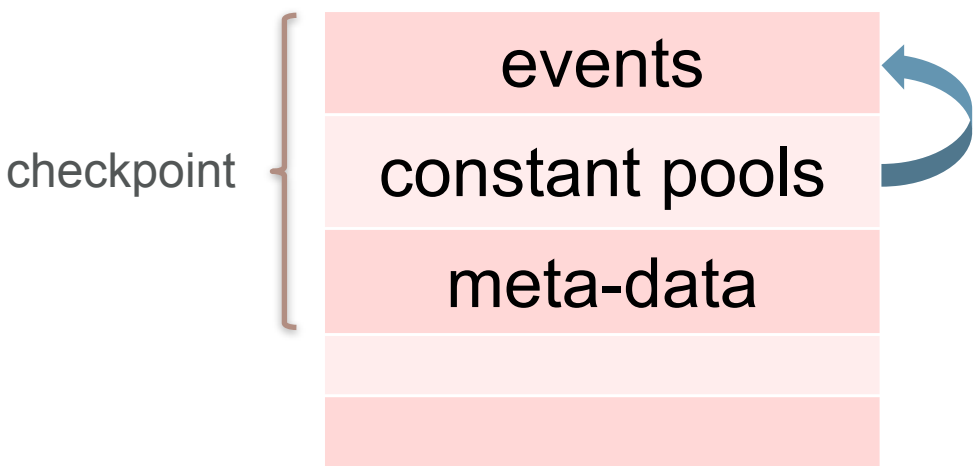
Classes, Methods, Stack Traces, Threads, Thread Groups, Strings

ORACLE

# Checkpoints - revisit

Events +
Constant Pools +
Event Metadata
=
Checkpoint

# Checkpoints - revisit



# Checkpoints - revisit

Contain everything required
to parse the events prior
to the checkpoint

# More information

- Whitepaper

  http://www.oracle.com/missioncontrol

- User Guide

  http://docs.oracle.com/javase/7/docs/technotes/guides/jfr/index.html

- Forum

  http://forums.oracle.com/community/developer/english/java/java_hotspot_virtual_machine/java_mission_control

ORACLE

# Q&A

# Thank you