ORACLE®

Java™
ORACLE®

The Conference for Java
& Software Innovation
Oct 10 – 12, 2016 | London

Ola Westin | Java Platform Group, Oracle

Production Time Profiling Using Java Flight Recorder

Safe Harbor Statement

Program Agenda

What do people say about Java Mission Control?

“I managed to do in one day what I've failed to do in 2+ weeks using <profiling tool> and <another profiling tool>.”

– *Maurizio Cimadamore, Oracle/Java LangTools*

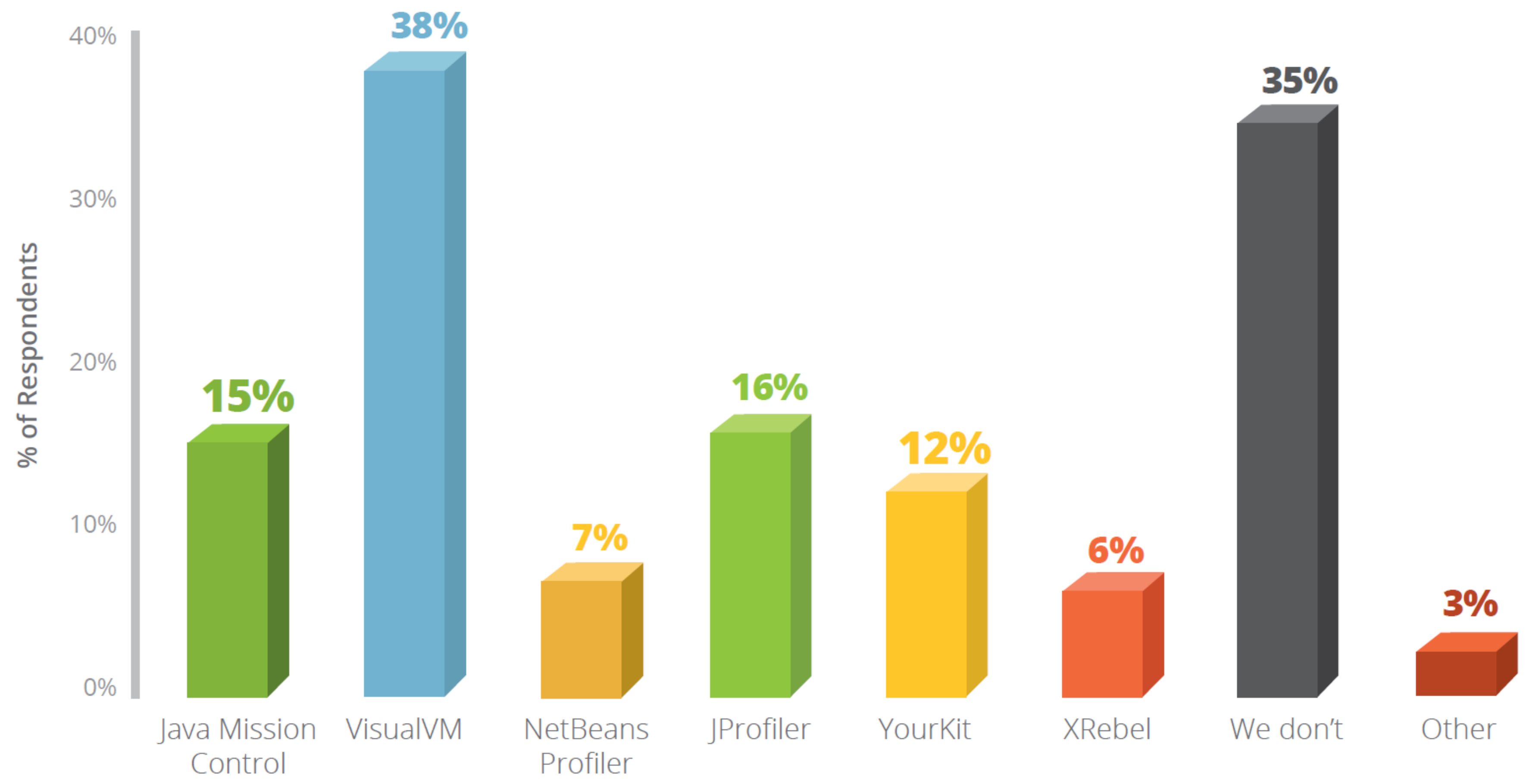
“JMC is my main tool for getting insight into the rhythm of a JVM and the running applications. … I have used recordings to resolve critical production issues caused by latency, memory-leaks or threading.”

– *Allan Thrane Andersen, Tryg*

“looking all over for Java memory profiling tools. Realize Mission Control was sitting there in java/bin the whole time!”

– *@PerlinMandleBro*

RebelLabs Java Tools and Technologies Landscape 2016, Java Profiler Usage



”Java Mission Control profiling tool”

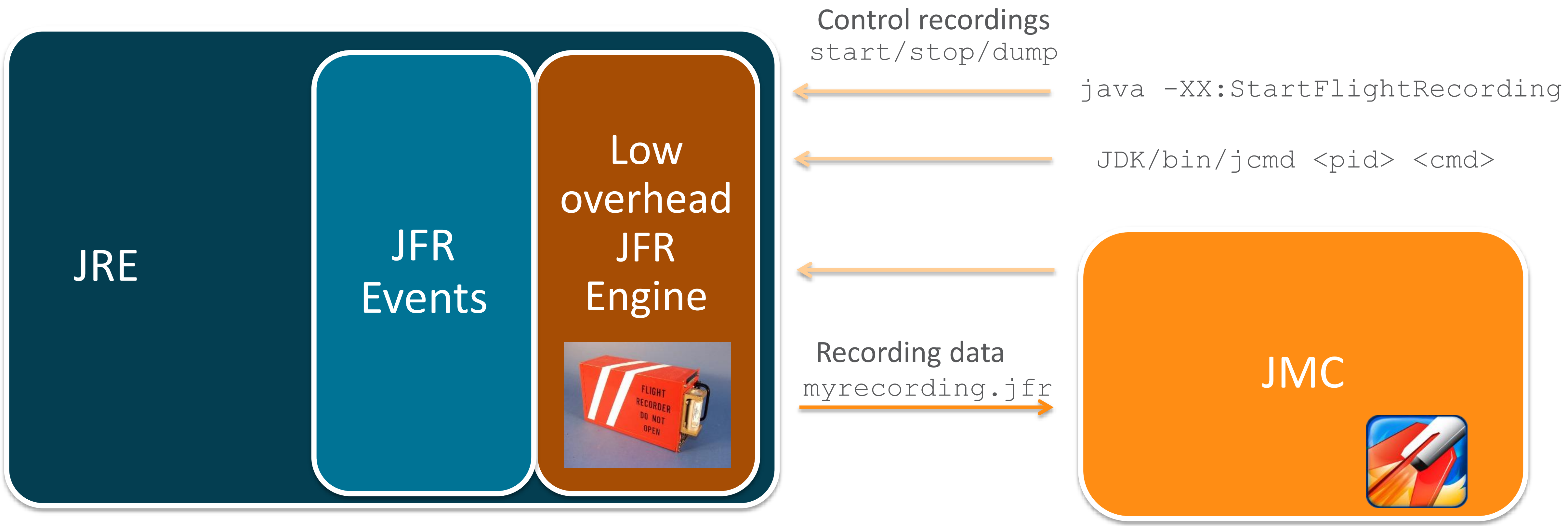
What do they mean?

Probably:

- Data from Java Flight Recorder
- Visualized in Java Mission Control

Java Flight Recorder (JFR) & Java Mission Control (JMC)

A brief overview



```
start/stop/dump
java -XX:StartFlightRecording
JDK/bin/jcmd <pid> <cmd>
myrecording.jfr
```

```
java -XX:+FlightRecorder
```

```
JDK/bin/jmc
```
or Eclipse plug-ins

Overview of Java Mission Control

The graphical client

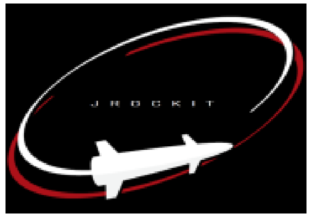
Java Mission Control

- A tool suite for monitoring JVM behavior
  - JMX console
    - Real time monitoring
  - Flight recorder
    - Production time profiling and diagnostics

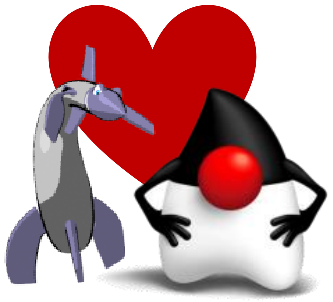
- Free for development and evaluation
  - Tool usage is free
  - Data creation in production requires a commercial license
    - tiny.cc/javalicense

History of Java Mission Control

- JRockit
  - JRockit Mission Control
  - JRockit Flight Recorder



- Appeal (JRockit) → BEA Systems → **Oracle** ← Sun (HotSpot)



- Best JRockit features → HotSpot JVM

- JFR and JMC released with Java 7u40 in 2013

JMC Installation and Startup

- Bundled with JDK

  Windows/Linux: `<JDK>/bin/jmc`

  Mac: `(/usr/bin/) jmc`

  For additional logging, add options:

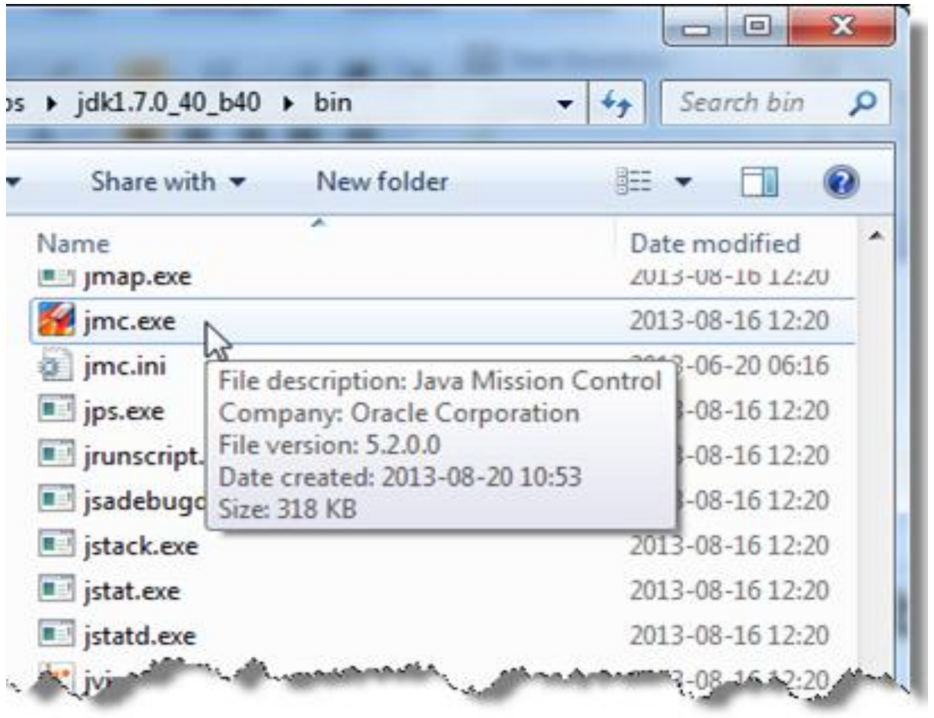
  `-consoleLog -debug`

- Eclipse plugins
  - Install from update site

    http://oracle.com/missioncontrol, Eclipse Update Site

- Experimental plugins
  - Install from within the JMC application or from

    http://oracle.com/missioncontrol, Eclipse Experimental Update Site



jdk1.7.0_40_b40 ▸ bin

Search bin

Share with ▾ New folder

Name	Date modified
jmap.exe	2013-08-16 12:20
jmc.exe	2013-08-16 12:20
jmc.ini	-06-20 06:16
jps.exe	-08-16 12:20
jrunscript.	-08-16 12:20
jsadebugd	-08-16 12:20
jstack.exe	2013-08-16 12:20
jstat.exe	2013-08-16 12:20
jstatd.exe	2013-08-16 12:20

File description: Java Mission Control
Company: Oracle Corporation
File version: 5.2.0.0
Date created: 2013-08-20 10:53
Size: 318 KB

Experimental Plugins

Downloadable from within Mission Control

- Dtrace
  - JFR style visualization of data produced by DTrace
- JOverflow
  - Memory anti-pattern analysis from hprof dumps
- JMX Console plugins
- Java Flight Recorder plugins
  - WebLogic
  - JavaFX

Overview of Java Flight Recorder

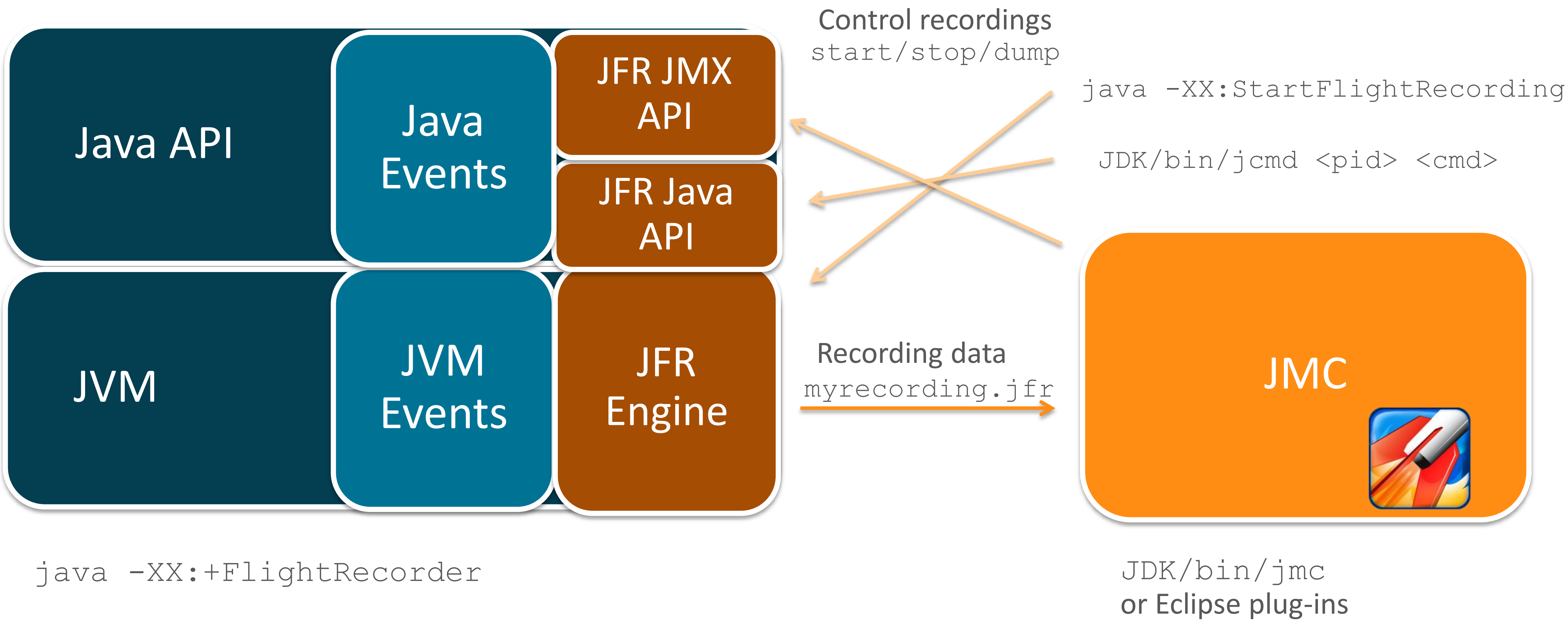
Low overhead profiling

Java Flight Recorder

- High performance Event Recorder
- Build into the JVM
  - Already available runtime information
  - Measuring the real behavior, doesn’t disable JVM optimizations
- Very detailed information
- Extremely low overhead (about 1-2%)
  - Can keep it always on and dump when necessary

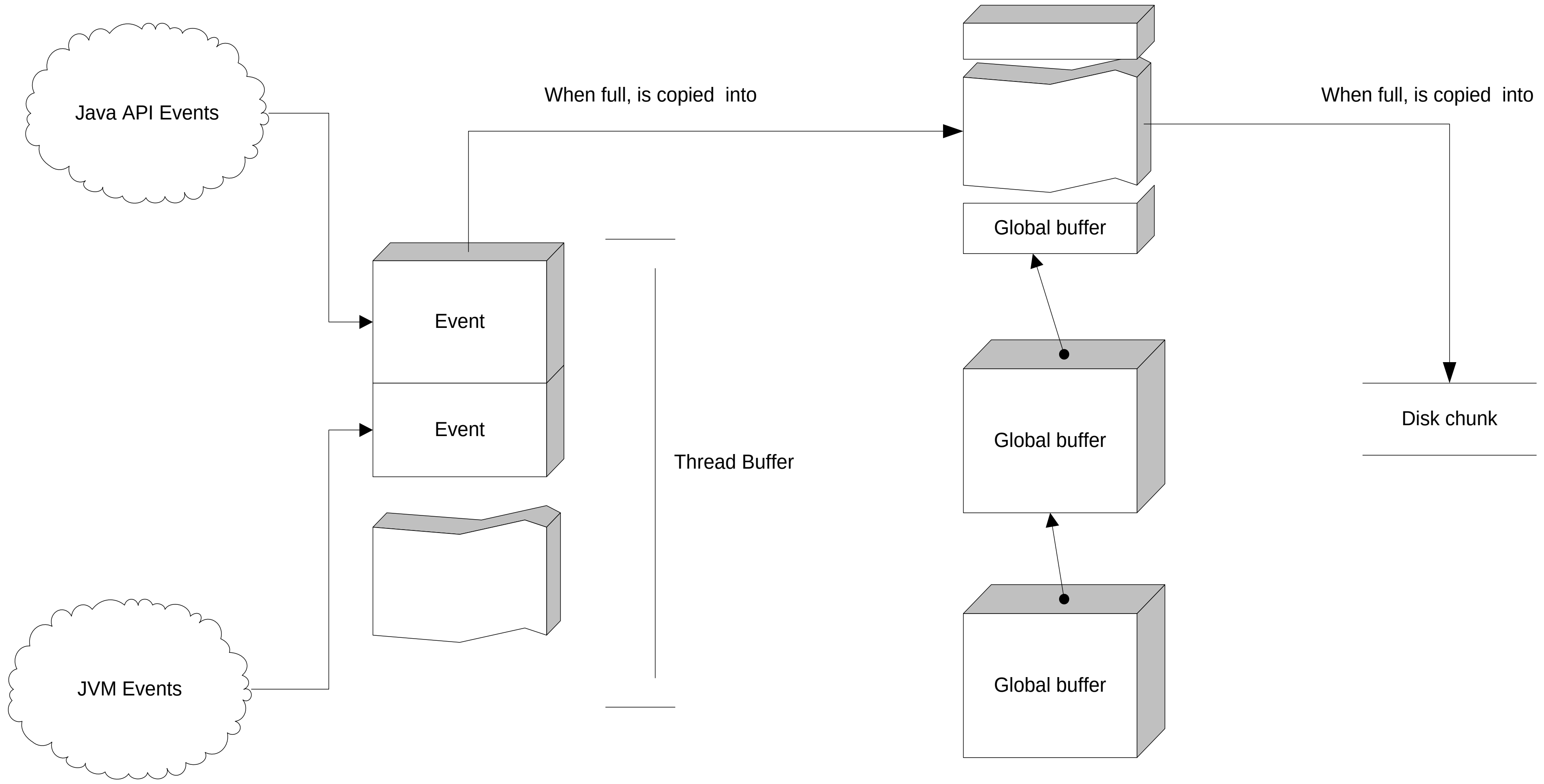
Java Flight Recorder (JFR) & Java Mission Control (JMC)

A slightly more detailed overview



Flight Recorder Inner Workings

Performance, performance, performance



Data collected by JFR

- Java application behavior
  - Threads & locks
  - I/O
  - Exceptions
  - etc.
- JVM behavior
  - Allocation & garbage collection
  - JIT Compiler
  - etc.
- Events implemented by the different subsystem teams

Different kinds of events

- Instant event – e.g. Thrown exception, Thread start
  - Something that happens at a single point in time
- Duration event – e.g. Garbage collection, Thread sleep
  - Timing for something
  - Configurable *threshold*
- Requestable event – e.g. Method profiling sample
  - Polled from a separate thread with a specified frequency
  - Configurable *period*
- Period and threshold settings impact the performance overhead

Method sampling

- Sampling profiler
  - Not logging every single call to your method
    - Part of how we get the low overhead
    - Detects hot methods
  - Does not require threads to be at safepoints
    - Startup flags can be used to give more accurate traces for inlined methods, but we recommend avoiding them to keep performance up

      ```
      -XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints
      ```
  - Not sampling threads calling native code

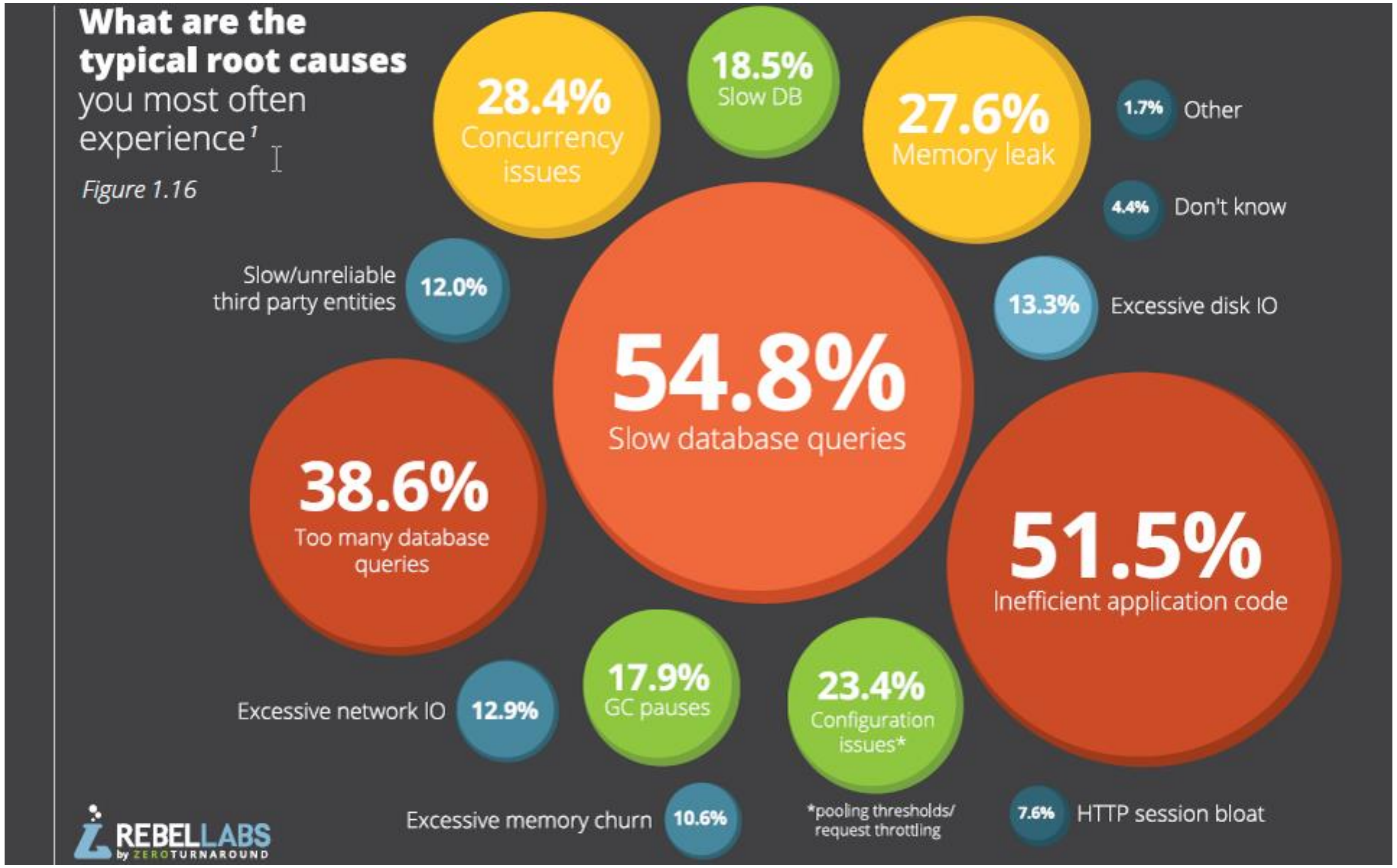
Event Configuration Templates

- Predefined templates
  - Default – designed to get as much data as possible while staying below 1% overhead
  - Profile – even more information, about 2% overhead
- Can be overridden when starting from Mission Control
- Stored as XML files in `<JRE>/lib/jfr/*.jfc`
  - Design your own from the Mission Control GUI

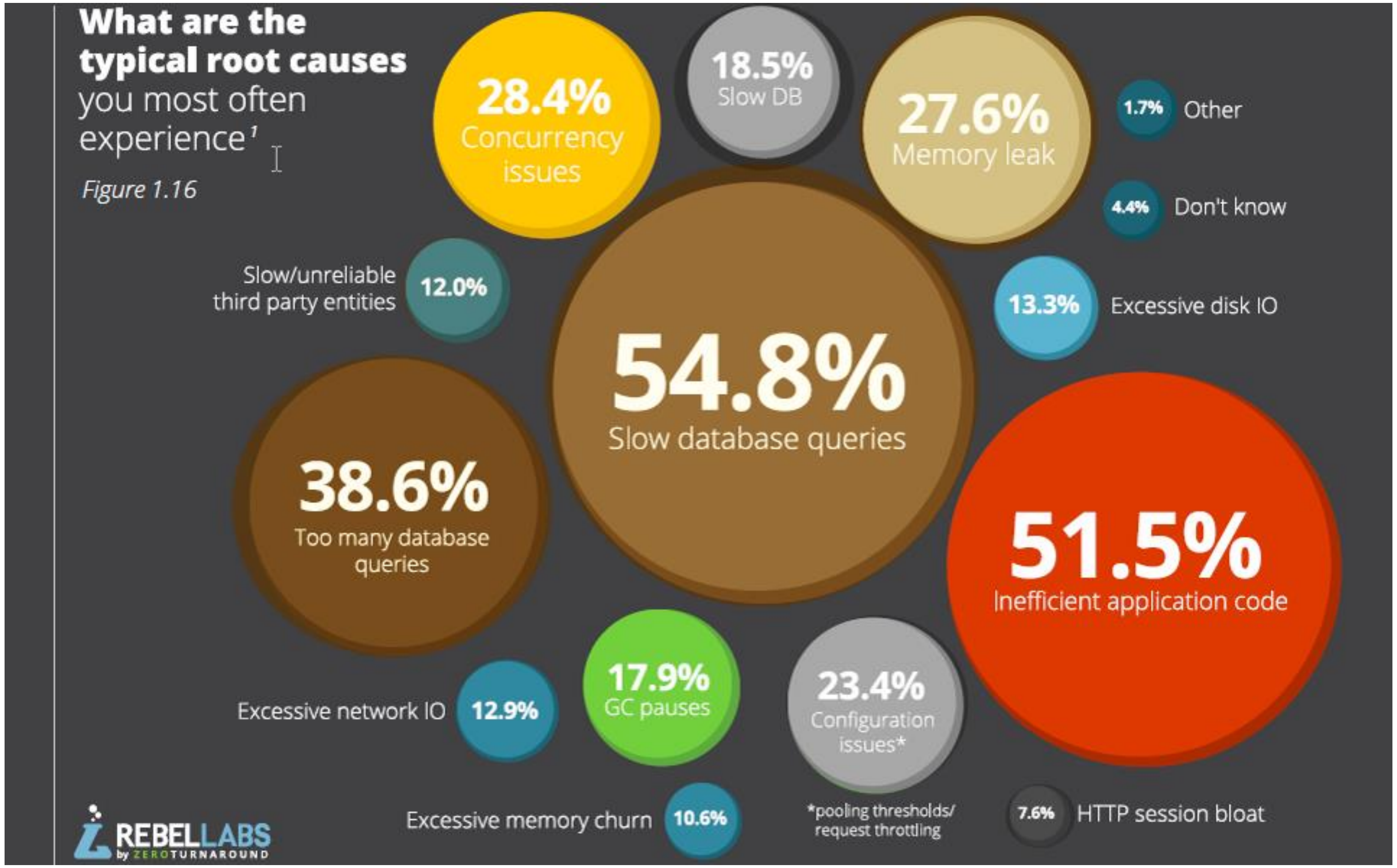
Different kinds of recordings

- Time fixed recordings (profiling recordings)
  - Runs for a specified time
  - Dumped to file when done, or opened automatically in the JMC GUI
  - Example use case: doing a 1 minute recording to test performance under load
- Continuous recordings
  - Runs until the JVM stops, or until explicitly stopped
  - Dumped to file when requested or when the JVM stops (if enabled)
  - Example use case: enable at startup and dump the last N minutes when something has gone wrong

RebelLabs Developer Productivity Report 2015, Java Performance Survey



RebelLabs Developer Productivity Report 2015, Java Performance Survey



Creating Recordings

More than one way

Preparations

- Start the JVM that you want to record from

```
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
```

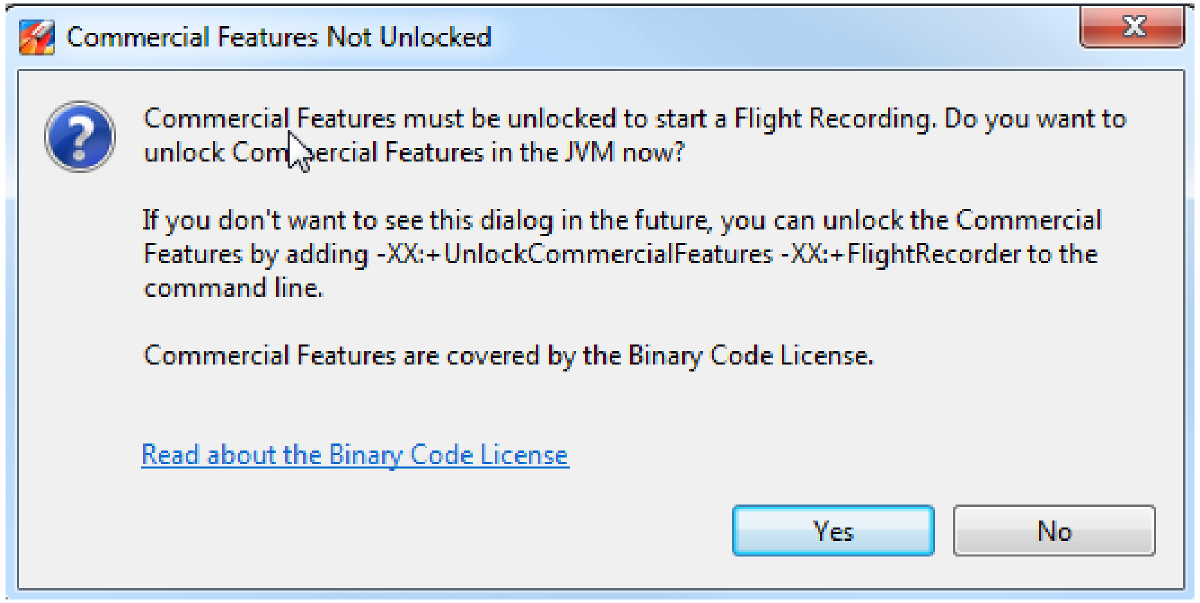
- In Java 8u40 and later it is possible to enable at runtime if needed
  - Using Java Mission Control
  - Using jcmd

Creating Recordings Using Mission Control

Easy and intuitive

1. Find a JVM to record from in the JVM Browser
2. Double click the Flight Recorder node under the JVM
3. Follow the wizard

On Java 8u40 and later, automatic enablement from JMC if startup flags are missing.



Commercial Features must be unlocked to start a Flight Recording. Do you want to unlock Commercial Features in the JVM now?

If you don't want to see this dialog in the future, you can unlock the Commercial Features by adding -XX:+UnlockCommercialFeatures -XX:+FlightRecorder to the command line.

Commercial Features are covered by the Binary Code License.

Read about the Binary Code License

Creating Recordings Using Startup Flags

- Time fixed

```
-XX:StartFlightRecording=delay=20s,duration=60s,
filename=C:\tmp\myrec.jfr,settings=profile,name=MyRecording
```

- Continuous with dump only on demand

```
-XX:StartFlightRecording=settings=default
```

- Continuous with dump on JVM exit

```
-XX:StartFlightRecording=settings=default
-XX:FlightRecorderOptions=dumponexit=true,
dumponexitpath=C:\tmp
```

- See documentation for Java options (google "java options")

https://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html

Creating Recordings Using JCMD

Useful for controlling JFR from the command line

Usage: `jcmd <pid> <command>`

- Starting a recording

```
jcmd 7060 JFR.start name=MyRecording settings=profile
delay=20s duration=2m filename=c:\tmp\myrecording.jfr
```

- Dumping a recording

```
jcmd 7060 JFR.dump name=MyRecording filename=C:\tmp\dump.jfr
```

- Unlocking commercial features (if not done with JVM startup flags)

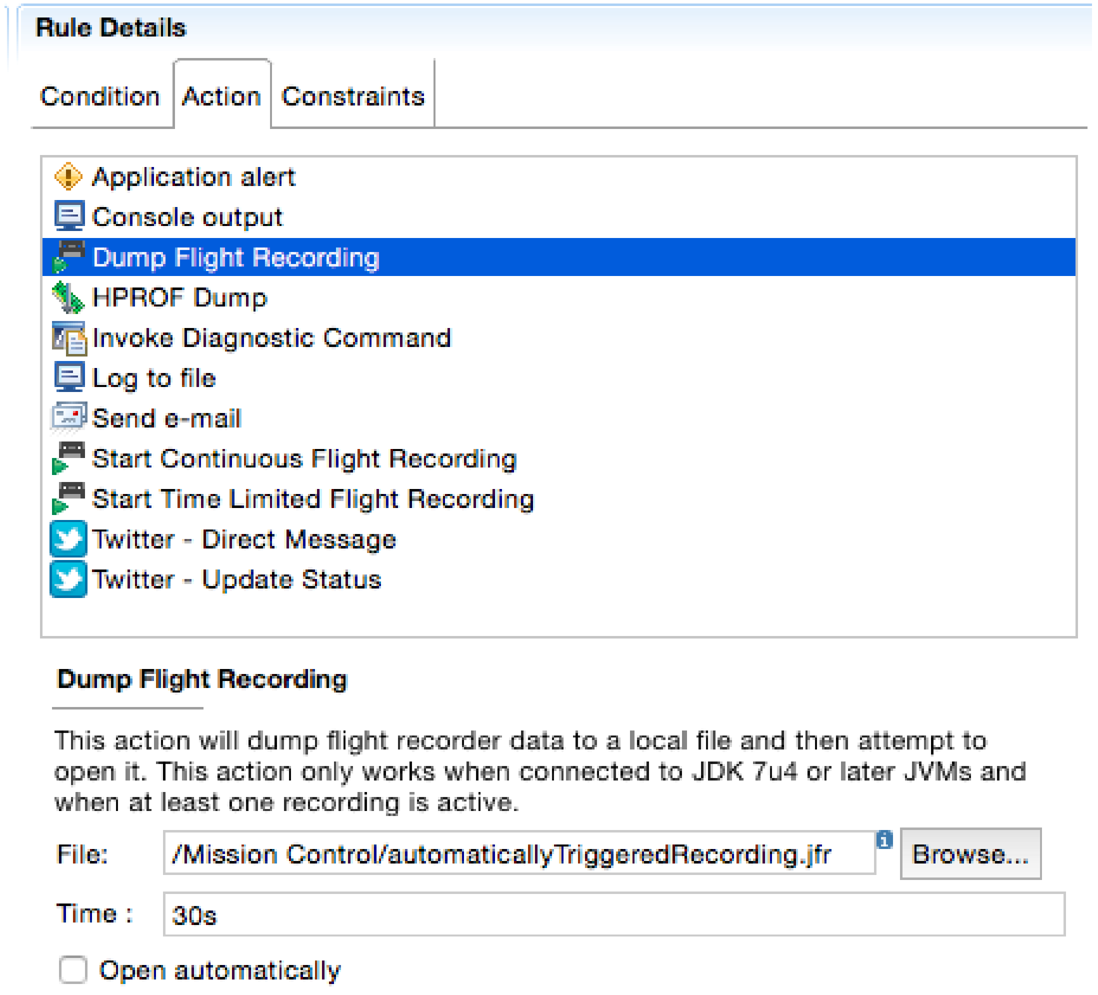
```
jcmd 7060 VM.unlock_commercial_features
```

Use `jcmd <pid> help` to see other things you can do with jcmd

Creating Recordings Using JMX Console Triggers

When real-time monitoring your application

- Start JMC
- Connect a JMX Console to your application
- Configure and enable rules on the triggers tab
- Recording will be started or dumped when the trigger occurs



Dump Flight Recording

This action will dump flight recorder data to a local file and then attempt to open it. This action only works when connected to JDK 7u4 or later JVMs and when at least one recording is active.

File: /Mission Control/automaticallyTriggeredRecording.jfr

Time : 30s

☐ Open automatically

Recordings On Remote Production Systems

- Remote access
  - Enable with `-Dcom.sun.management.jmxremote...` and connect with JMC
    - Consider security!
  - Run `jcmd` locally on the server and transfer the resulting JFR file to your workstation
- Choose when to enable JFR
  - At startup

    Allocates a small amount of memory for buffers.
  - Dynamically
    - Zero overhead at startup. Initialization overhead occurs when enabling. Might cause some classes to be deoptimized.
    - Good if you want to avoid restarting the JVM.

Analyzing Recordings

Using the graphical client

What the data can tell you

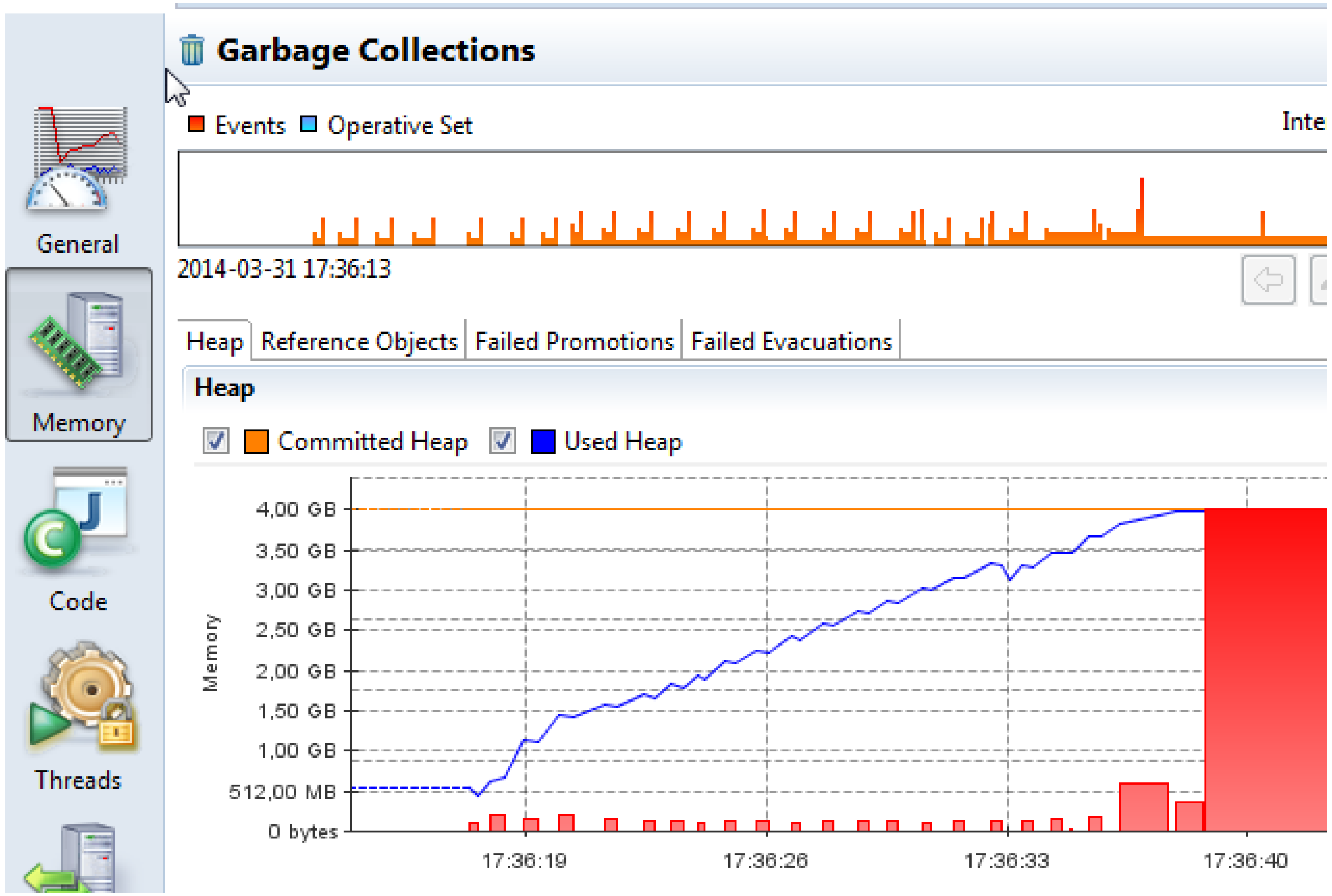
What questions do you want answered?

- Only you know what your application is supposed to be doing
  - Batch job or real time trading
  - Should the CPU usage be high or low?
  - If you have a theory about what is wrong, then you can start your investigation from there

- Not trivial to compare performance of different recordings
  - You may want to add custom data for tracking e.g. transaction times

Analyzing Flight Recordings in JMC

- Preconfigured tabs
- Highlights various areas of common interest
  - Code
  - Memory
  - Threads
  - etc.



Demo

Flight recorder startup and analysis

Customization, Future, Links

Adding Your Own Events (unsupported)

```java
import com.oracle.jrockit.jfr.*;

public class Example {
    private final static String PRODUCER_URI = "http://www.example.com/demo/";
    private Producer myProducer;
    private EventToken myToken;

    public Example() throws URISyntaxException, InvalidEventDefinitionException, InvalidValueException {
        myProducer = new Producer("Demo Producer", "A demo event producer.", PRODUCER_URI);
        myToken = myProducer.addEvent(MyEvent.class);
    }

    @EventDefinition(path="demo/myevent", name = "My Event",
       description="An event triggered by doStuff.", stacktrace=true, thread=true)
    private class MyEvent extends TimedEvent {
        @ValueDefinition(name="Message", description="The logged important stuff.")
        private String text;
        public MyEvent(EventToken eventToken) { super(eventToken); }
        public void setText(String text) { this.text = text; }
    }

    public void doStuff() {
        MyEvent event = new MyEvent(myToken);
        event.begin();
        String importantResultInStuff = "";
        // Generate the string, then set it...
        event.setText(importantResultInStuff);
        event.end();
        event.commit();
    }
}
```

Parsing Recordings (unsupported)

Reusing the event instance (not thread safe)

```
...
private MyEvent event = new MyEvent(myToken);

public void doStuffReuse() {
   event.reset();
   event.begin();
   String importantResultInStuff = "";
   // Generate the string, then set it...
   event.setText(importantResultInStuff);
   event.end();
   event.commit();
}
...
```

If JFR is not enabled, then the custom events give 0% overhead

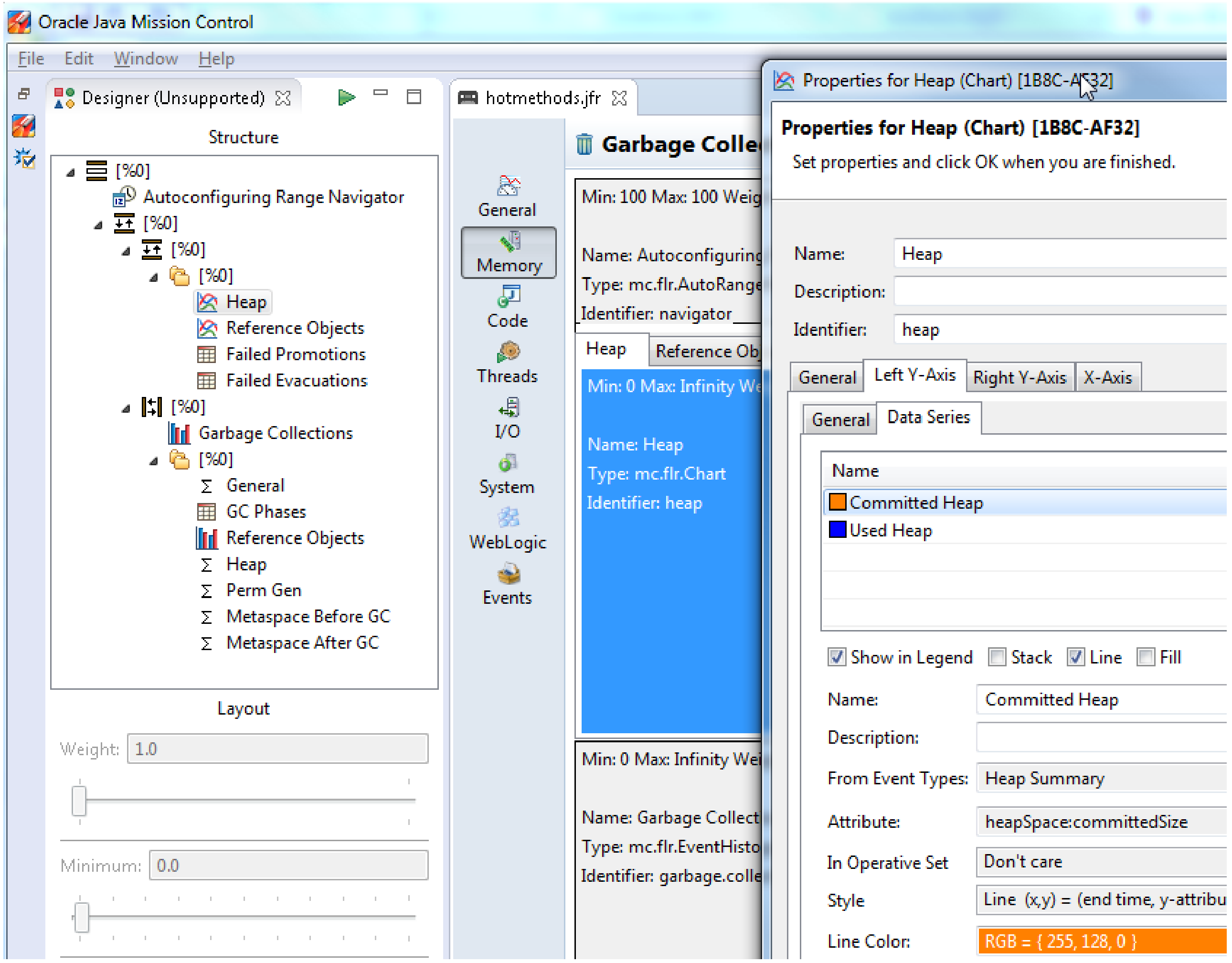
Parsing Recordings (unsupported)

Two unsupported options

- The JDK parser
  - `import oracle.jrockit.jfr.parser.*;`
  - SAX style parser
- The JMC parser
  - `import com.jrockit.mc.flightrecorder.FlightRecording;`
  - DOM style parser
- http://hirt.se/blog/?p=446

Built In GUI Editor (unsupported)

- Show view -> Designer
- Customize the existing GUI or produce entirely new GUIs for events
- Export the created GUI to share it with others



Structure:

```
[%0]
  Autoconfiguring Range Navigator
  [%0]
    [%0]
      [%0]
        Heap
        Reference Objects
        Failed Promotions
        Failed Evacuations
    [%0]
      Garbage Collections
      [%0]
        General
        GC Phases
        Reference Objects
        Heap
        Perm Gen
        Metaspace Before GC
        Metaspace After GC
```

Layout

Weight: 1.0

Minimum: 0.0

Min: 100 Max: 100 Weig

Name: Autoconfiguring
Type: mc.flr.AutoRange
Identifier: navigator

Min: 0 Max: Infinity We

Name: Heap
Type: mc.flr.Chart
Identifier: heap

Min: 0 Max: Infinity Wei

Name: Garbage Collect
Type: mc.flr.EventHisto
Identifier: garbage.colle

Properties for Heap (Chart) [1B8C-AF32]

Properties for Heap (Chart) [1B8C-AF32]

Set properties and click OK when you are finished.

Name:	Heap
Description:	
Identifier:	heap

Name
🟧 Committed Heap
🟦 Used Heap

[✓] Show in Legend [ ] Stack [✓] Line [ ] Fill

Name:	Committed Heap
Description:	
From Event Types:	Heap Summary
Attribute:	heapSpace:committedSize
In Operative Set	Don't care
Style	Line (x,y) = (end time, y-attribu
Line Color:	RGB = { 255, 128, 0 }

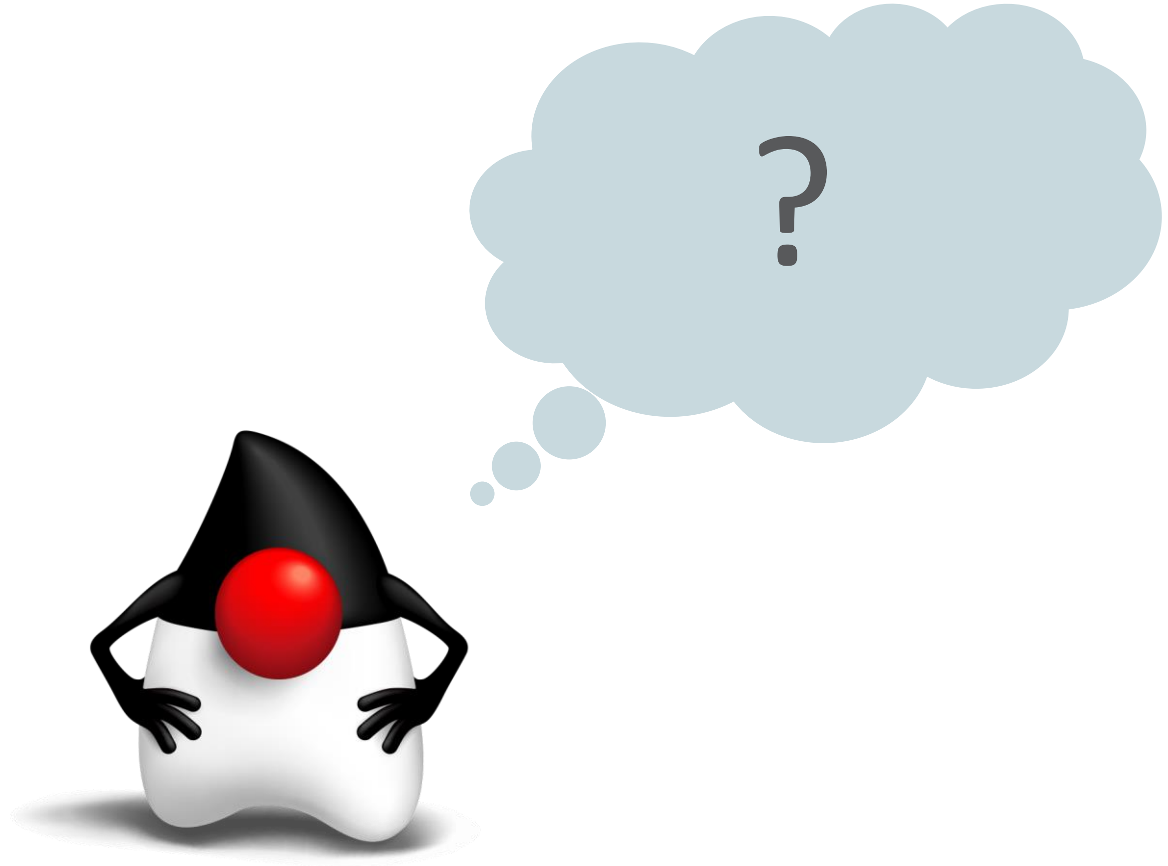
Future

- Flight recorder
  - Supported API for creating events
  - Performance enhancements
  - Can write data to disk even in bad situations
  - More events
- Java Mission Control
  - Automated analysis of flight recordings
  - Reworked JFR GUI
- See slides from JavaOne 2016 presentation by Marcus Hirt
  - Java Mission Control and JFR in JDK 9: A Sneak Peek [CON1509]

Resources

- Home page
  - http://oracle.com/missioncontrol (Click Discussion to find the forum)
  - http://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/index.html
- Twitter
  - @javamissionctrl
  - @hirt
- Blog
  - http://hirt.se/blog/
- Facebook
  - https://www.facebook.com/javamissionctrl

Q&A



ORACLE®