Robo4J mit NetBeans und Java Flight Recorder

Java Mission Control und Java Flight Recorder

Miro Wengner eXaring
Sen. Developer

Wolfgang Weigend
Sen. Leitender Systemberater
Java Technology and Architecture

Safe Harbor Statement

Faster and Easier Use and Redistribution of Java SE

- Oracle is proposing to increase the release cadence of Java SE to every six months
- Oracle will simplify how developers, customers, and consumers use Java SE
  - **Starting with JDK 9 GA Oracle plans to ship OpenJDK builds under the GPL**
  - Oracle has proposed a time-driven release model for Java SE instead of the historical feature-driven model
  - **Oracle JDK will contribute previously commercial features such as Java Flight Recorder to OpenJDK**
  - Oracle will work with other OpenJDK contributors to make the community infrastructure complete, modern and accessible
- The Oracle JDK will continue as a commercial long term support offering
  - The Oracle JDK will primarily be for commercial and support customers once OpenJDK binaries are interchangeable with the Oracle JDK (target late 2018)
  - Oracle will continue to enhance the packaging and distributing of complete ready-to-run applications

Java Mission Control

- A tools suite for ***production*** use (fine in development too)
  - Basic monitoring
  - 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

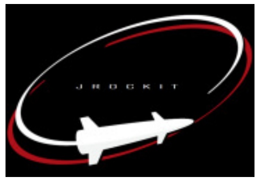
- **Starting with JDK 9 GA Oracle plans to ship OpenJDK builds under the GPL**
- **Oracle JDK will contribute previously commercial features such as Java Flight Recorder to OpenJDK**

”Java Mission Control profiling tool”

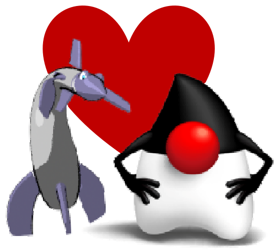
- Data from Java Flight Recorder
- Visualized in Java Mission Control

Java Mission Control - History

- JRockit Flight Recorder



- Appeal (JRockit) acquired by BEA Systems
  - Acquired by Oracle and acquired Sun Microsystems



- Best JRockit features converged with HotSpot JVM

- JFR and JMC released with JDK 7u40

Java Mission Control Main Tools

Two main tools:

- JMX Console
  - Online monitoring
- Flight Recorder
  - Offline low overhead profiler

- JRockit Mission Control also had the **Mem**ory **Leak** Analyzer

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 plug-ins
- Java Flight Recorder plug-ins
  - WLS
  - JavaFX

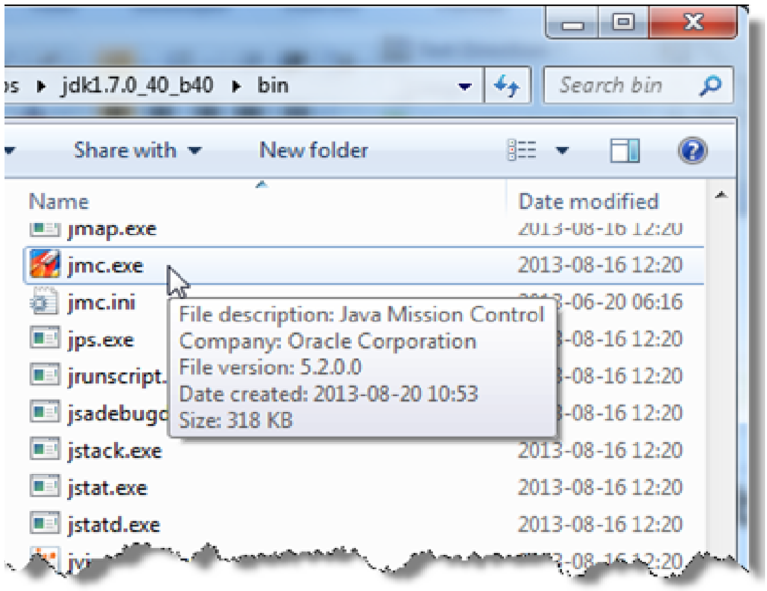
JMC installation/startup

`<JDK>/bin/jmc`

– Mac: `(/usr/bin/) jmc`

  Add if needed:

```
-consoleLog -debug     ( | more 2>&1 )
```



s ▸ 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	3-06-20 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
jvi	3-08-16 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

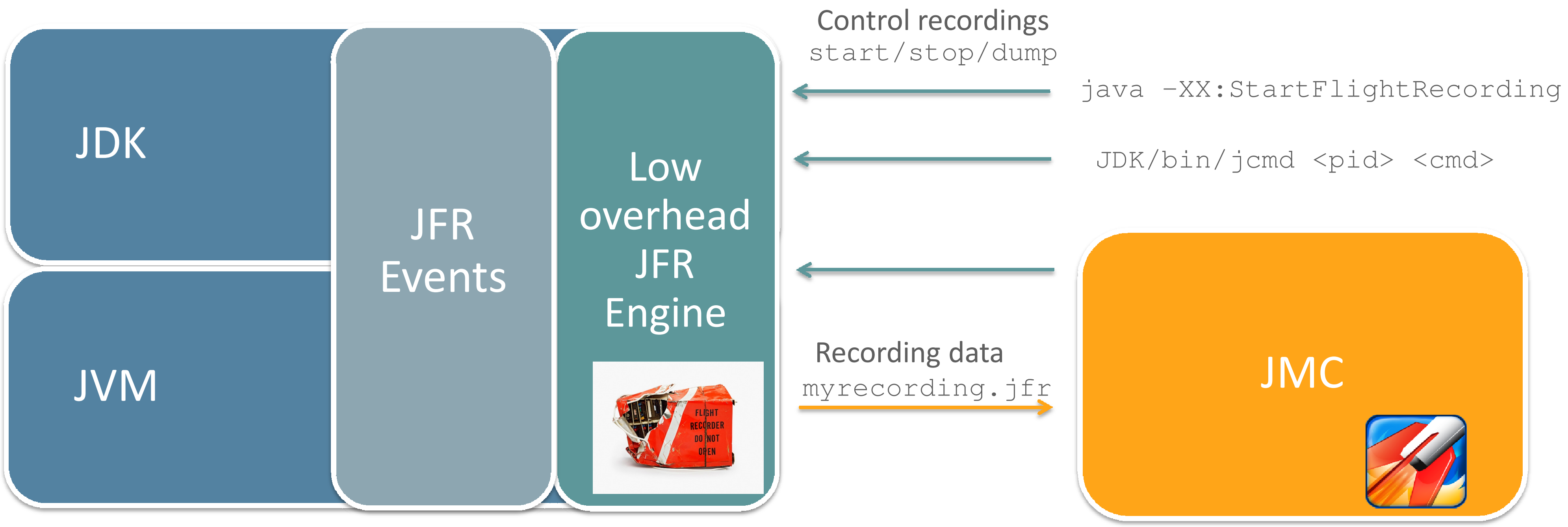
- Eclipse plug-ins
  – Install from update site on OTN: http://oracle.com/missioncontrol, Eclipse Update Site

- Experimental plug-ins: Install from within the JMC app, or from
  https://oracle.com/missioncontrol, Eclipse Experimental Update Site

Java Flight Recorder

- High Performance Event Recorder
- Built into the JVM
  - Already available runtime information
  - Measuring the real behavior, doesn’t disable JVM optimizations
- Binary recordings
  - Self contained self describing chunks
- Very detailed information
- Extremely low overhead (~ 2..3%)
  - Can keep it always on, dump when necessary

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



```
start/stop/dump
```

```
java -XX:StartFlightRecording
```

```
JDK/bin/jcmd <pid> <cmd>
```

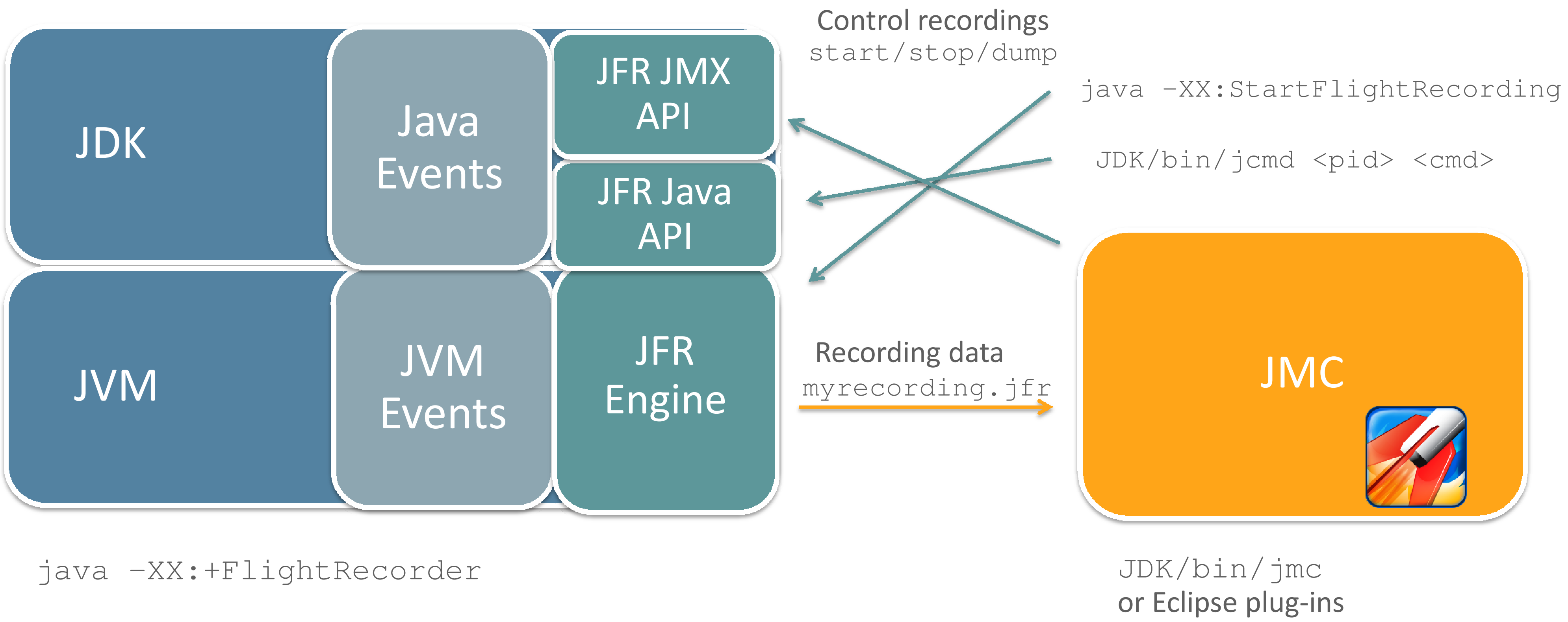
```
myrecording.jfr
```

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

```
JDK/bin/jmc
```

or Eclipse plug-ins

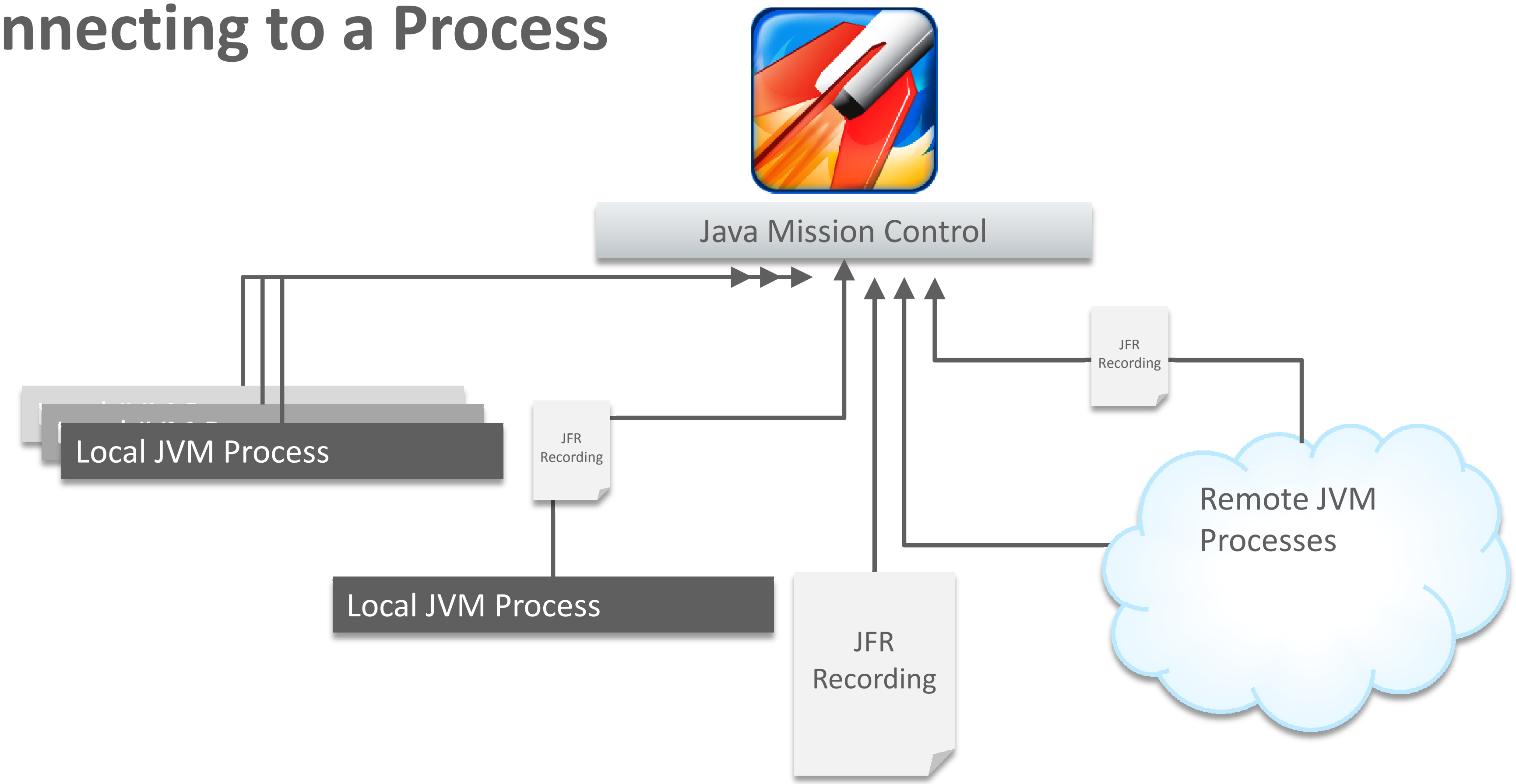
Java Flight Recorder (JFR) and Java Mission Control (JMC)



Connecting to a Process

- The entry point to running Java processes is the JVM Browser
- By default, it will list all discovered locally running processes
- The JVM Browser shows, by default, a flat list of all discovered and defined connectors. The list can be split into a tree to separate locally running processes, JDP (Java Discovery Protocol) discovered ones, and custom defined connectors

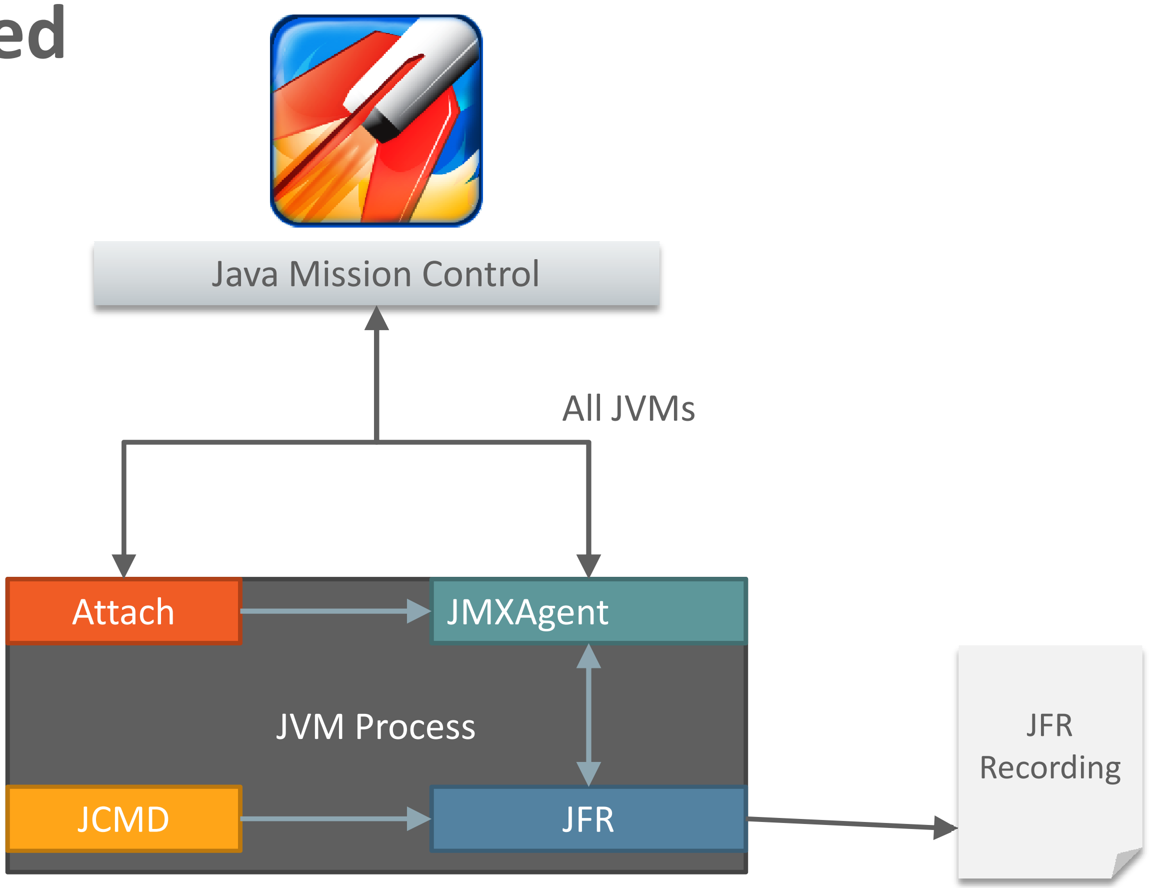
Connecting to a Process



Protocols involved

- Java Attach (only locally running Java processes)
  - Used for local discovery of JVMs
  - Used for starting the local JMX management agent, should a tool requiring JMX want to connect
  - Used directly by some tools, such as the tool for starting the external JMX agent
- JMX (normally JMXRMI, but can be configured)
  - Used by most tools for communication and transfer of data
  - Note that JFR can be used fully without ever using JMX through jcmd and/or command line options

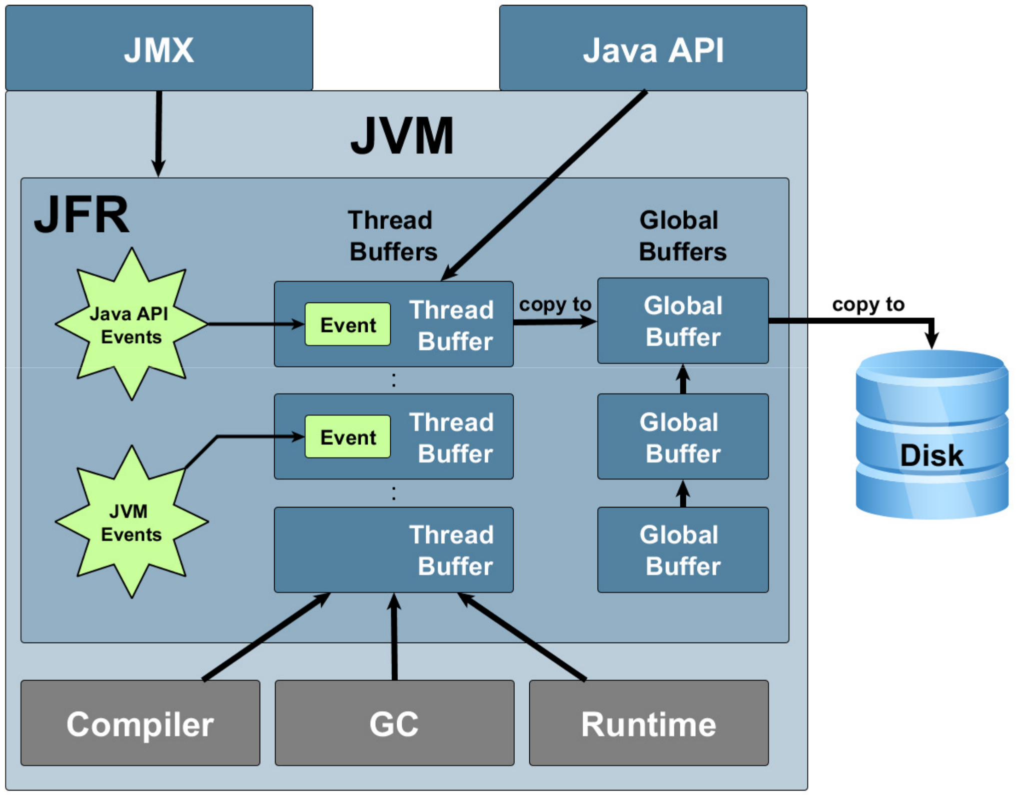
Protocols involved



Java Flight Recorder – Architecture

- Information gathering
  - Instrumentation calls all over the JVM
  - Application information via Java API
- Collected in Thread Local buffers

  ⇢ Global Buffers ⇢Disk
- Binary, proprietary file format
- Managed via JMX
- Java Flight Recorder
  - Start from JMC 5.5 or CLI
- Activate Flight Recorder
  - -XX: +UnlockCommercialFeatures
  - -XX: +FlightRecorder

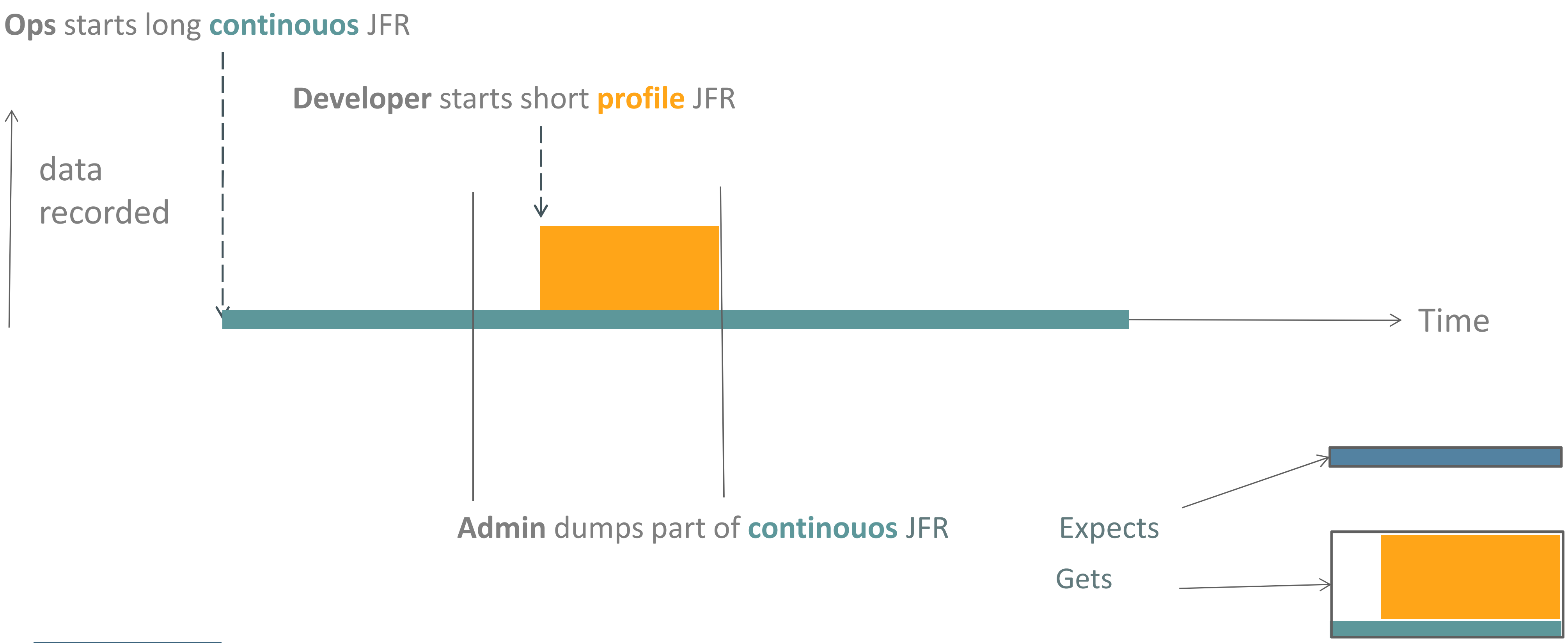


Different Kinds of Recordings

- Continuous Recordings
  - Have no end time
  - Must be explicitly dumped
  - Example use case: Enable at startup, dump the last X minutes when something goes wrong
- Time Fixed Recordings (‘profiling recordings’)
  - Have a fixed time
  - If started from Java Mission Control, opened automatically in the GUI
  - Example use case: Performance testing under load, do a 1 minute recording

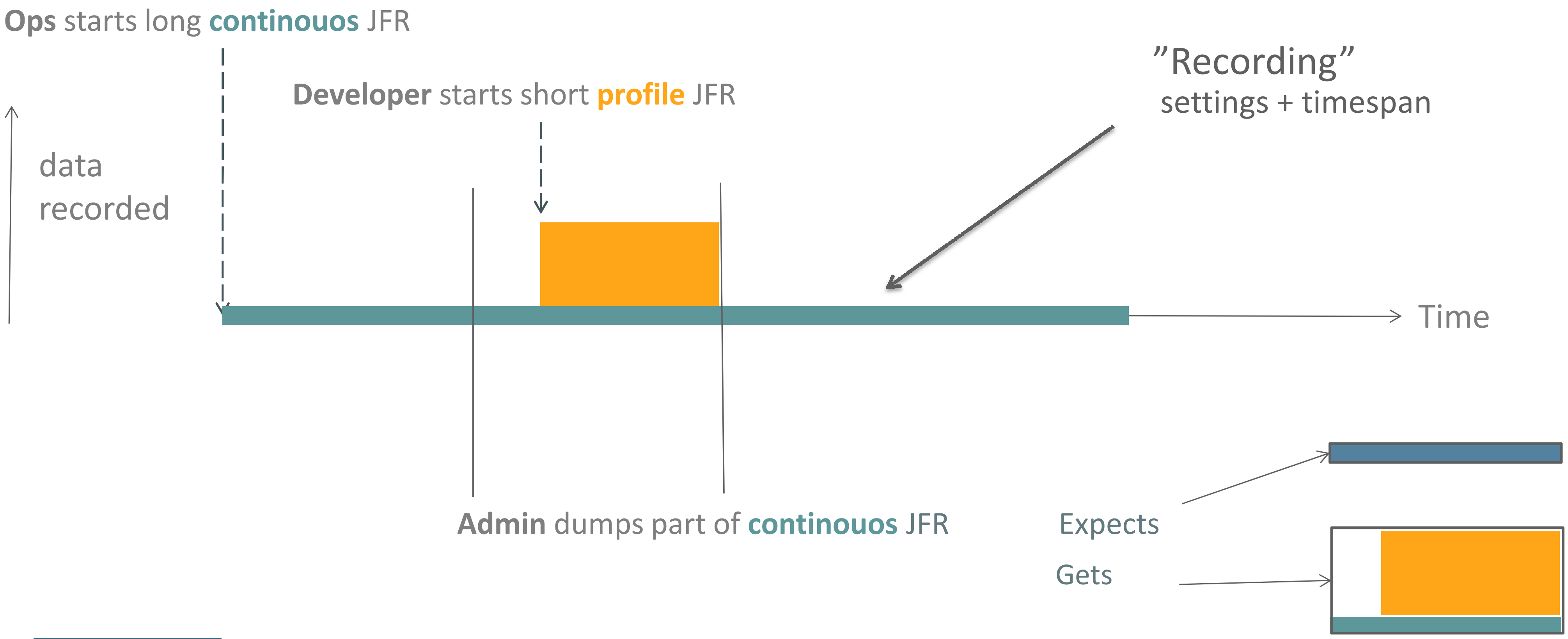
How to Think About Recordings

What data does the **Admin** get when he dumps a recording?


Ops starts long **continouos** JFR
Developer starts short **profile** JFR
data recorded
Time
Admin dumps part of **continouos** JFR
Expects
Gets

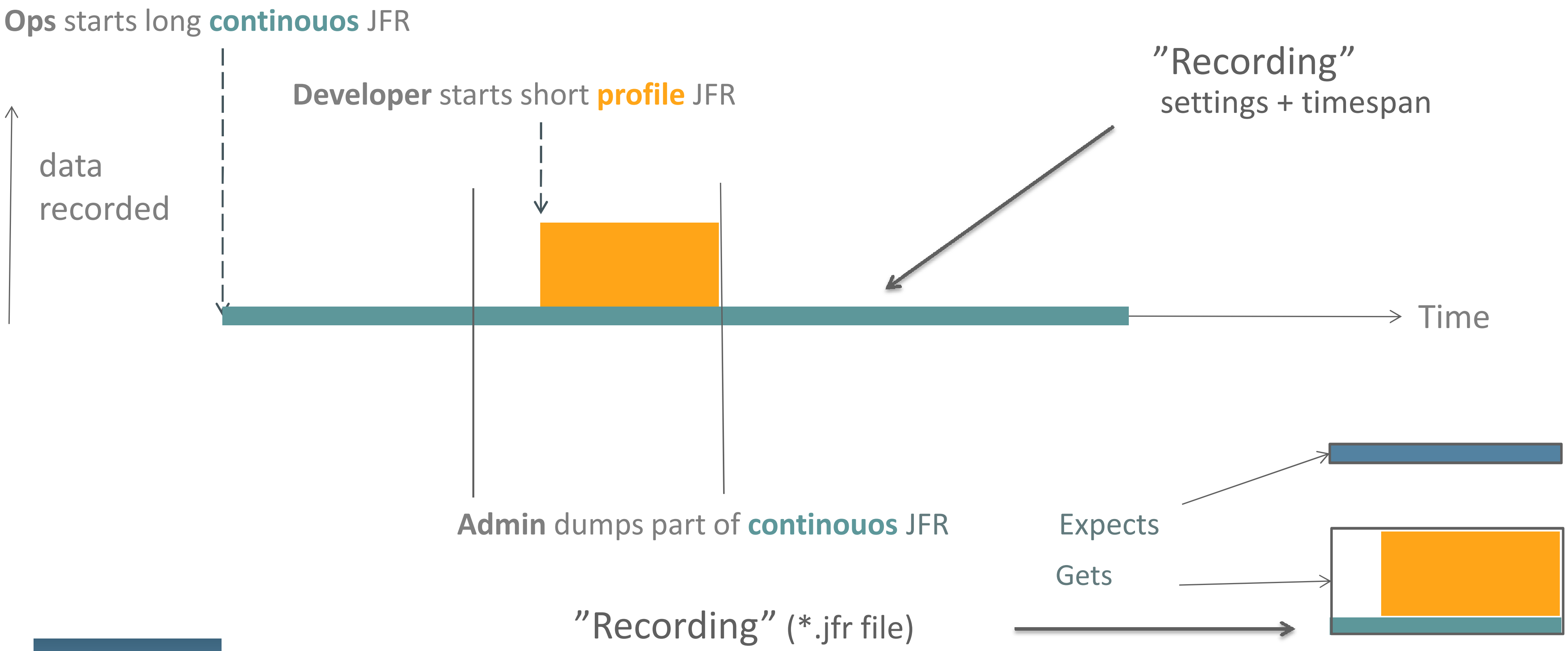
How to Think About Recordings

What data does the **Admin** get when he dumps a recording?


Ops starts long **continouos** JFR
Developer starts short **profile** JFR
data recorded
”Recording”
settings + timespan
Time
Admin dumps part of **continouos** JFR
Expects
Gets

How to Think About Recordings

What data does the **Admin** get when he dumps a recording?


Ops starts long **continouos** JFR
Developer starts short **profile** JFR
data recorded
Time
”Recording” settings + timespan
Admin dumps part of **continouos** JFR
Expects
Gets
”Recording” (*.jfr file)

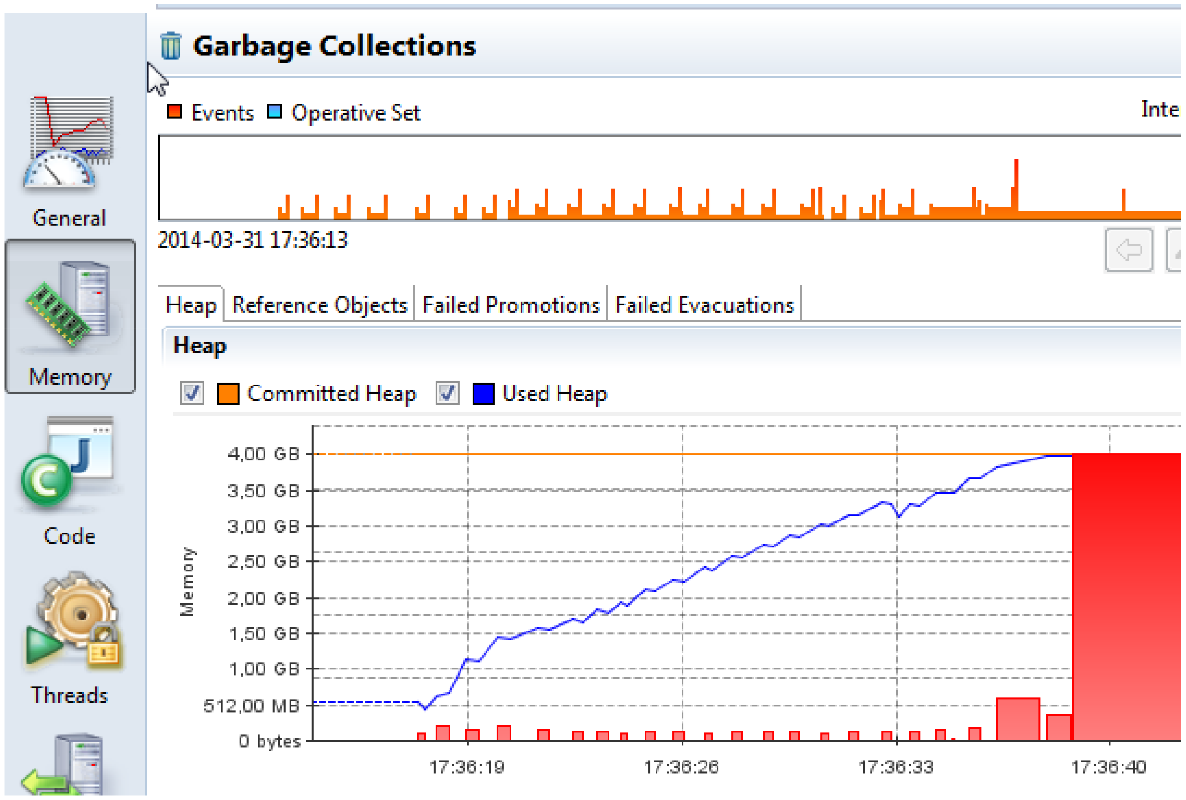
When analyzing Flight Recordings

- Only you know what your application is supposed to be doing
  - Batch job, or real time trading?
  - Do you want the CPU usage to be high or low?
  - If you have a theory about what is wrong, you can find out why

- Not trivial to see which recording has the best performance
  - Possibly to add custom data for tracking for example transaction times

Analyzing Flight Recordings in JMC

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



Roadmap

- JMC 6.0.0 with JDK 9
  - Automatic analysis of Flight Recordings
  - Greatly revised user interface (more modern, cleaner)
- Continually
  - New event types (improved I/O events, GC events, loaded libraries)

Improvements (1)

- New **Supported API's**
  - Easier to use
  - Moved namespace from oracle.jrockit.* to jdk.jfr.*
- Not compatible with old unsupported APIs
  - Modularized
- Performance enhancements
  - Compressed Integers
  - Smarter Event Classes
- Event reference does not escape into the generated code
- No event object reuse required

Improvements (2)

- Can emit data to disk even in bad situations
  - Useful in fatal situation, e.g. out-of-memory or crash
- New Events
  - More detailed safe point information
  - More detailed code cache information
  - New compiler events for detailed inlining information
  - New G1 specific information for better visualization of region states
  - Module events (loaded module, full transitive closure)
  - NativeLibrary (load, periodic event, by default each chunk)

Roadmap - JDK 9 JFR Features

- Easy to use supported APIs for all things Flight Recorder
  - Allows for custom events
  - Programmatic access for reading Flight Recordings
  - Programmatic access for controlling the Flight Recorder
  - Modularized, works on smaller profiles
- Improved command line ergonomics
- Can dump on crashes and out-of-memory

Summary

- Java Flight Recorder provides a common view to the JVM and the Java application
  - JVM Events and Java API Events
- Extremely low overhead (<= 2..3%)
  - Can keep it always on, dump when necessary
- Tooling for analysing recordings built into the Oracle JDK via Java Mission Control
- Java APIs available for recording custom information into the Flight Recorder in the Oracle JDK and with JDK 9 GA the OpenJDK get Java Flight Recorder
- Third party integration giving holistic view of the detailed information recorded by the Flight Recorder (WebLogic Server, JavaFX)

Thanks!

Wolfgang.Weigend@oracle.com

Twitter: @wolflook

Miro Wengner eXaring