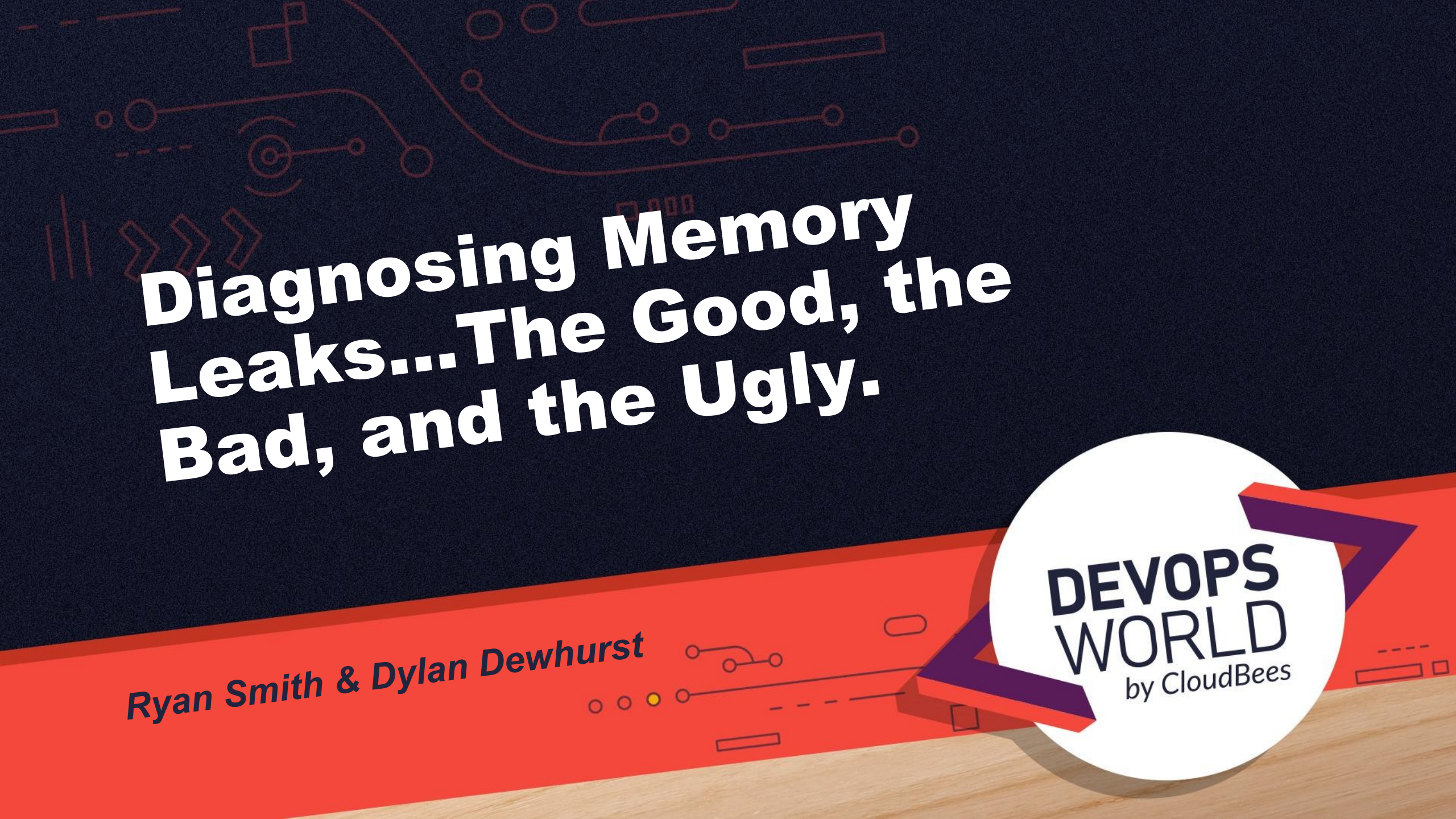
Diagnosing Memory Leaks...The Good, the Bad, and the Ugly.

Ryan Smith & Dylan Dewhurst



Speakers

cloudbees®

- #1 contributor to The Jenkins Project
- Home of the largest group of Jenkins-Certified Engineers
- Offers Enterprise CI/CD products and services

Ryan Smith

- Global Escalation Manager at CloudBees
- Jenkins Performance and Stability Team Lead
- 10+yr Enterprise Java Evangelist
- JVM Whisperer

Dylan Dewhurst

- Senior Developer Support Engineer (DSE)
- Jenkins Performance and Stability Team member
- Been at CloudBees for ~2 years

What We'll Cover

- How to Identify a Memory Leak
- How to capture and analyze Heap Dumps
- How to diagnose Heap Memory Leaks
- How to diagnose Metaspace Memory Leaks
- How to diagnose Java Native Memory Leaks
- Monitoring Best Practices

Diagnosing Memory Leaks...The Good, the Bad, and the Ugly

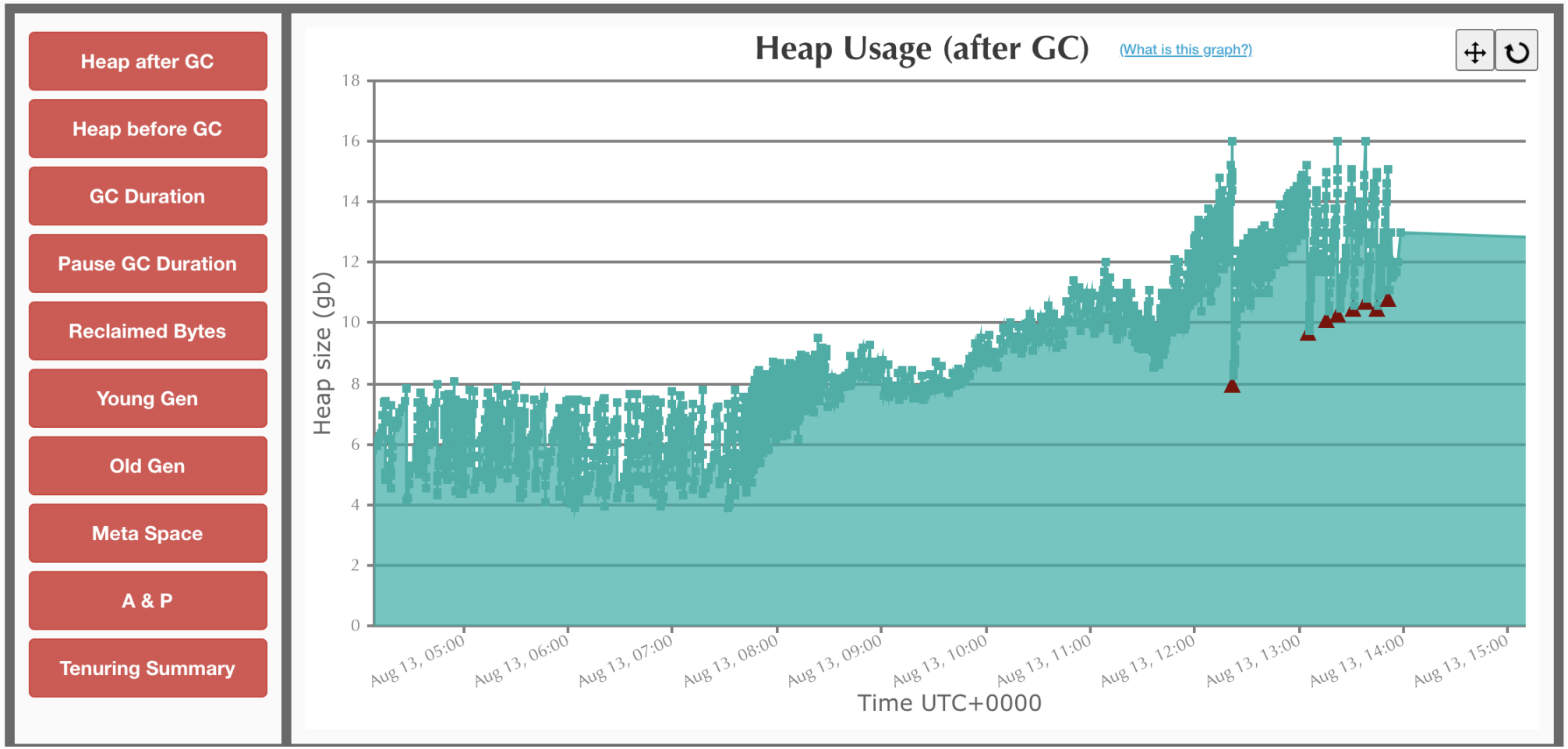
How to Identify A Memory Leak

DEVOPS WORLD
by CloudBees

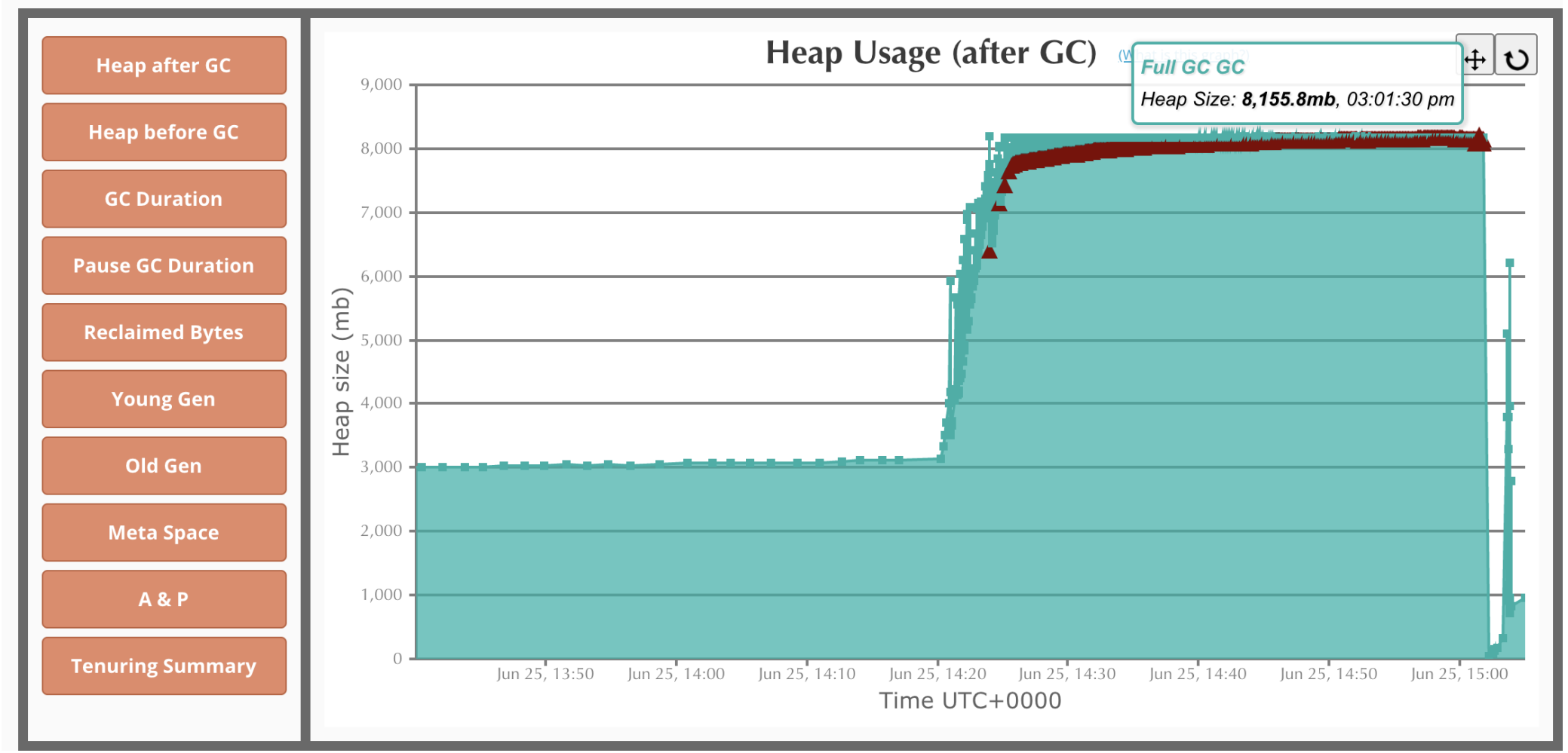
GC log analysis talk

- How to Get the Most out of Jenkins
- Avoiding Pitfalls with Jenkins

GC logs displaying leak graphed by GCeasy



GC logs displaying leak graphed by GCeasy



OutOfMemory Errors

- **java.lang.OutOfMemoryError: Java heap space**
  - JVM is unable to allocate space for an object in heap
- **java.lang.OutOfMemoryError: GC Overhead limit exceeded**
  - Garbage collector is running all the time and Java program is making very slow progress
- **java.lang.OutOfMemoryError: Requested array size exceeds VM limit**
  - The application attempted to allocate an array that is larger than the heap size
- **java.lang.OutOfMemoryError: unable to create new native thread**
  - The JVM has created too many threads and is not able to create any more

oom-killer errors

- These will be found in the **`dmesg`** logs

```
[Wed Aug  5 18:09:22 2020] TCP agent conne invoked oom-killer: gfp_mask=0x14000c0(GFP_KERNEL), nodemask=(null), order=0, oom_score_adj=-998
[Wed Aug  5 18:09:22 2020] TCP agent conne cpuset=87776ea7182b314de2a222052de860013e11f1e3579d439bd05af19b06df70ac mems_allowed=0-1
[Wed Aug  5 18:09:22 2020] CPU: 18 PID: 36112 Comm: TCP agent conne Tainted: P           OE    4.15.0-91-generic #92~16.04.1-Ubuntu
...
[Wed Aug  5 18:09:22 2020] [ pid ]   uid  tgid total_vm      rss pgtables_bytes swapents oom_score_adj name
[Wed Aug  5 18:09:22 2020] [67956]  1000 67956     1089      196    57344        0          -998 tini
[Wed Aug  5 18:09:22 2020] [67982]  1000 67982 13195171  5961748 52785152        0          -998 java
[Wed Aug  5 18:09:22 2020] [70437]  1000 70437     4806      780    77824        0          -998 sh
[Wed Aug  5 18:09:22 2020] [70453]  1000 70453     4806      506    69632        0          -998 sh
[Wed Aug  5 18:09:22 2020] [70454]  1000 70454     5517      546    77824        0          -998 script
[Wed Aug  5 18:09:22 2020] [70456]  1000 70456     4839      876    73728        0          -998 bash
[Wed Aug  5 18:09:22 2020] Memory cgroup out of memory: Kill process 67956 (tini) score 0 or sacrifice child
[Wed Aug  5 18:09:22 2020] Killed process 67982 (java) total-vm:52780684kB, anon-rss:23833172kB, file-rss:13820kB, shmem-rss:0kB
```

Diagnosing Memory Leaks...The Good, the Bad, and the Ugly

How to Capture and Analyze Heap Dumps

DEVOPS WORLD
by CloudBees

Obtain the Process ID

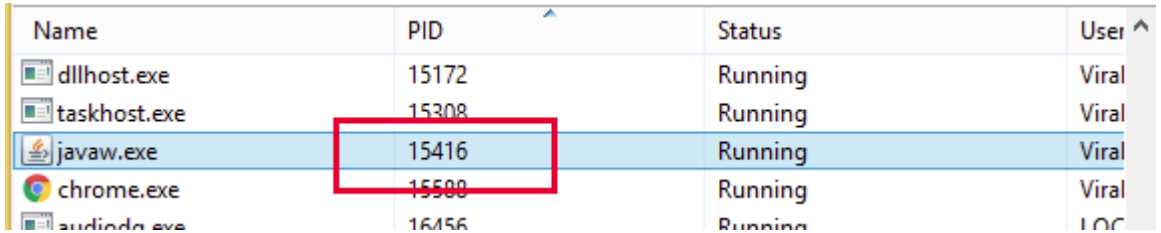
jps - From Unix Terminal or Windows Cmd with the JDK installed into the OS. Lists the instrumented Java Virtual Machines (JVMs) on the target system.

`ps -ef | grep java` - From *Nix Terminals only. It is used to obtain the process id of all java processes.

Process Explorer - There are task manager UI applications (System Tools) which can be used to obtain the process id. Find the process and view its PID in the corresponding column.

On Windows and Linux (Ubuntu) named **Task Manager**

On Mac OS named **Activity Monitor**



Essential JDK Tools for JVM Gunslingers

JCMD

- Used to send diagnostic command requests to the JVM
- Must be used on the same machine where the JVM is running
- Must run as the same user and group id's as the JVM

JMAP

- Tool to print statistics about memory in a running JVM
- Can be used for local or remote processes

Capture a Heap Dump

- `jcmd $PID GC.heap_dump $FILENAME` - From Unix Terminal or Windows Powershell

OR

- `jmap -dump:format=b,file=$FILENAME.bin $PID` - From Unix Terminal or Windows Powershell

NOTE: The JVM will need to perform a full garbage collection cycle before it can generate a heap dump.

-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=${PATH}

Move the Heap Dump

- Heap Dumps can be quite **LARGE** in size
- Use the **`split`** utility to split the generated heapdump file into 2GB segments for ease of movement

```
split -b 2gb $FILENAME
```

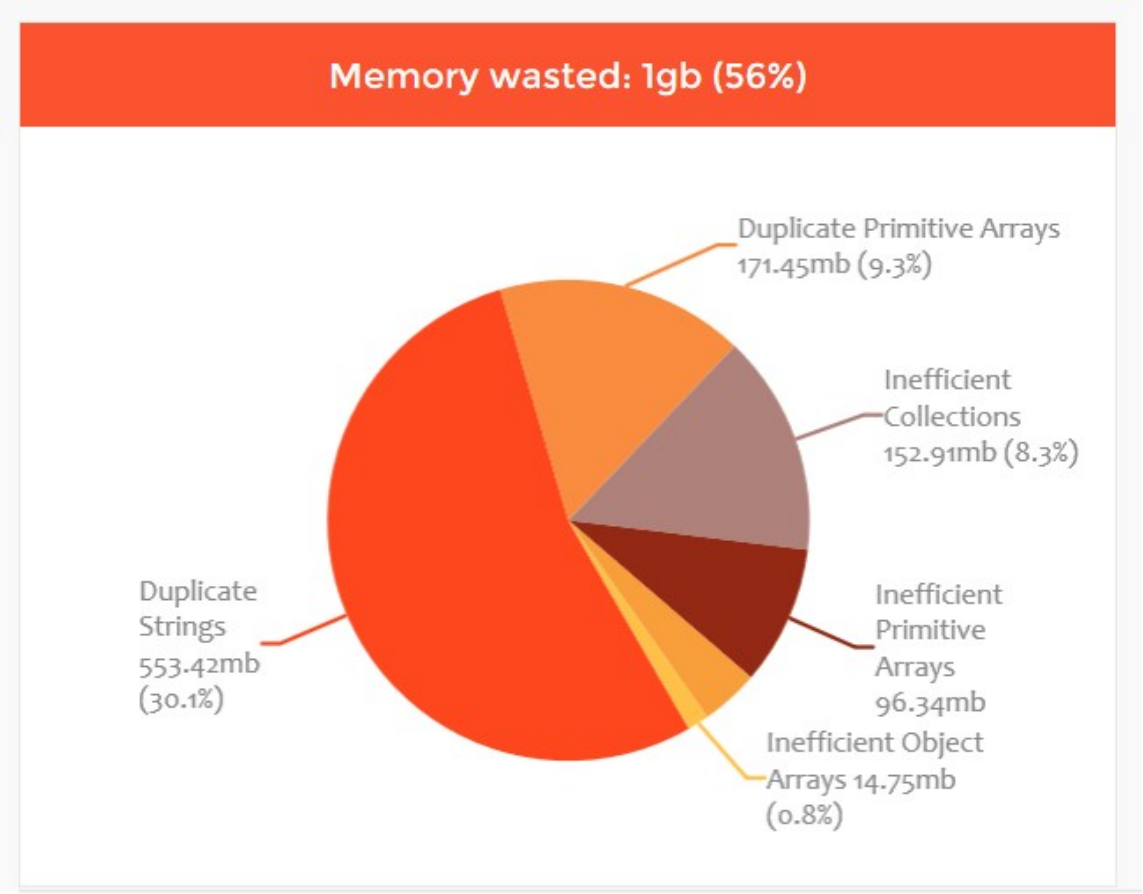
To concatenate files after splitting:

`cat ${file1} {file2} {file3} > {output_filename}` -From Unix Terminal or Windows Powershell

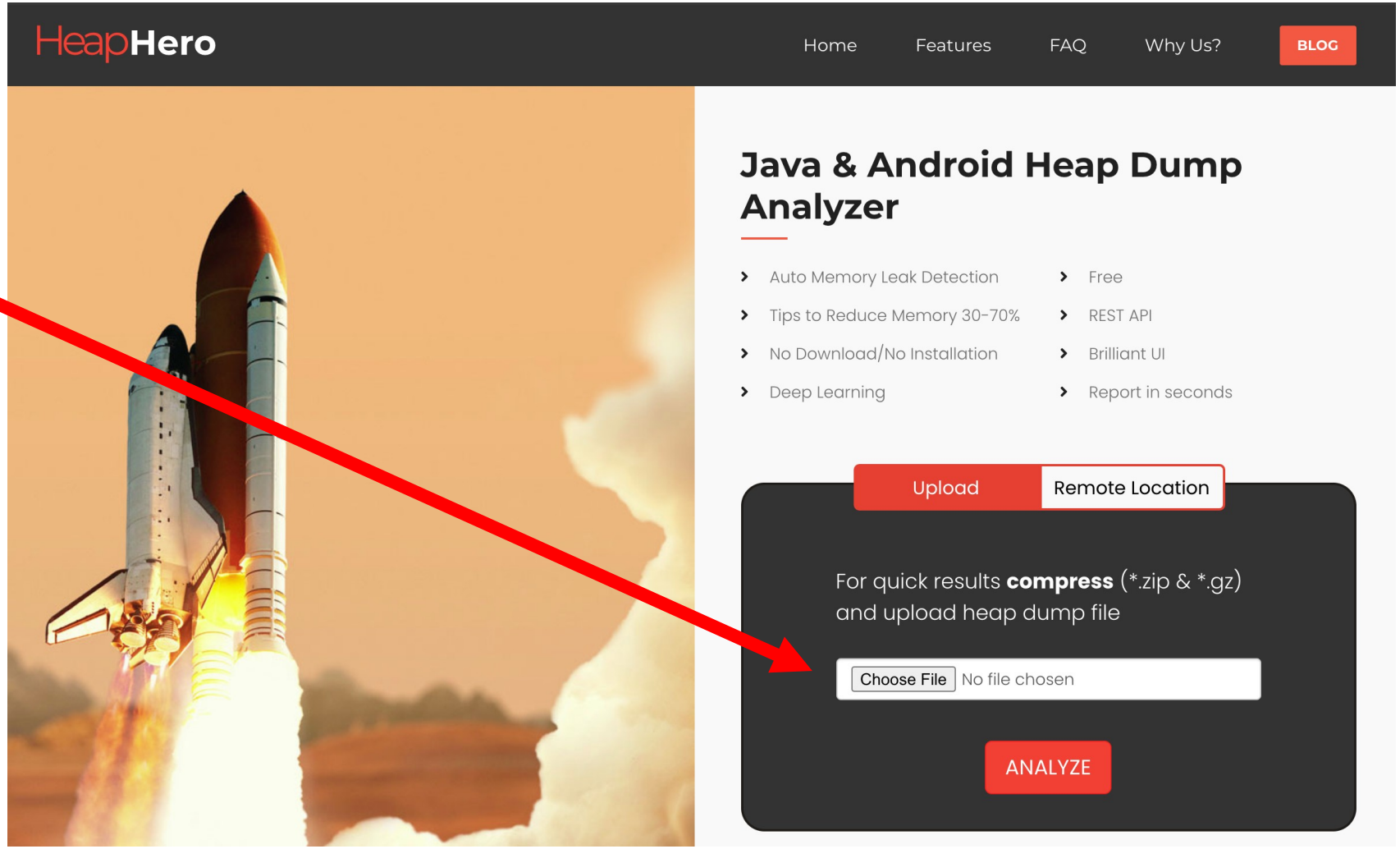
`copy /B {file1} + {file2} + {file3} {output_filename}` -From Windows CMD

Analyze Heap Dumps

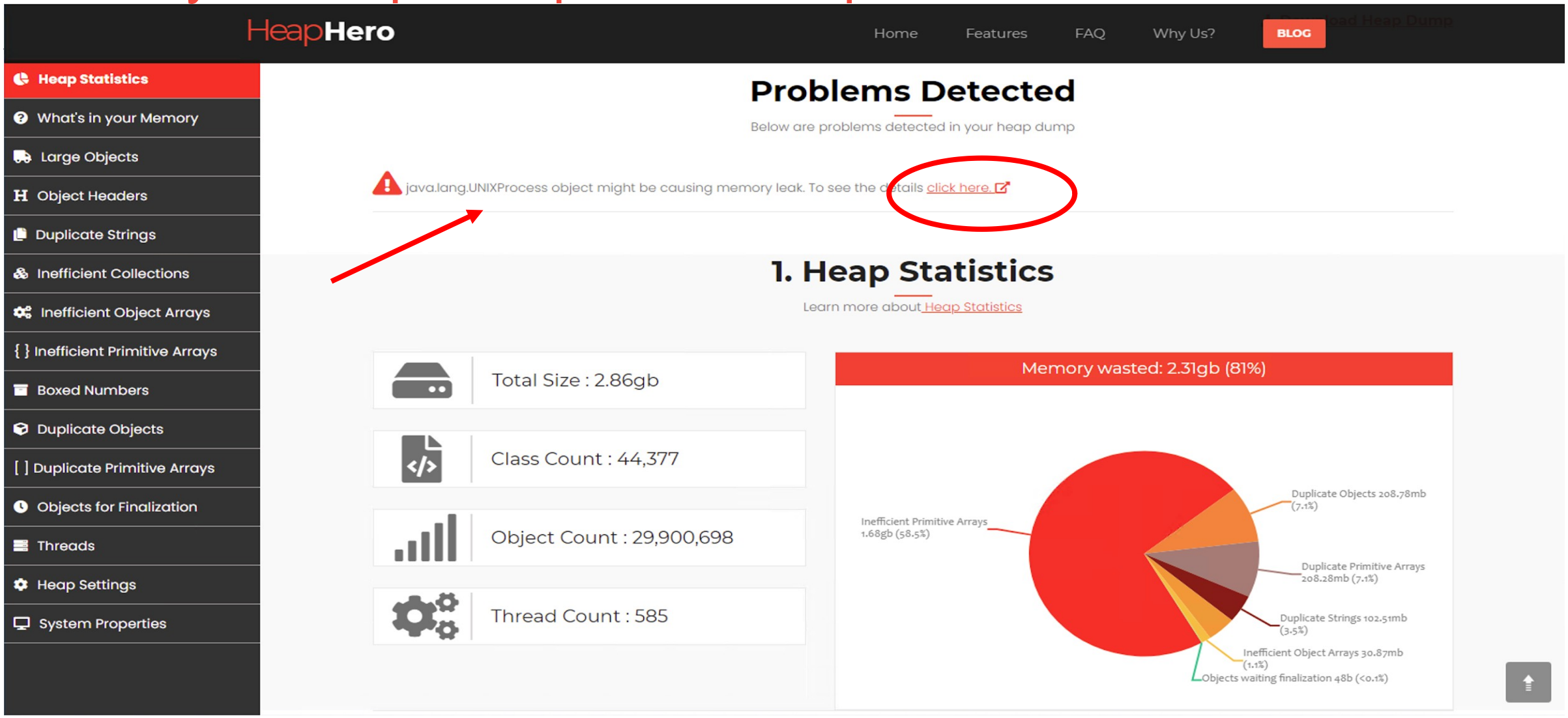
- HeapHero
  - Enterprise edition can be deployed on-prem
  - Free version available online
  - Award winning deep learning algorithms
- Eclipse MAT
  - OSS Project
  - Free
  - Steep learning curve
- IBM Memory Analyzer
- YourKit
  - Not Free
- JavaVisualVM



Analyze Heap Dumps with HeapHero



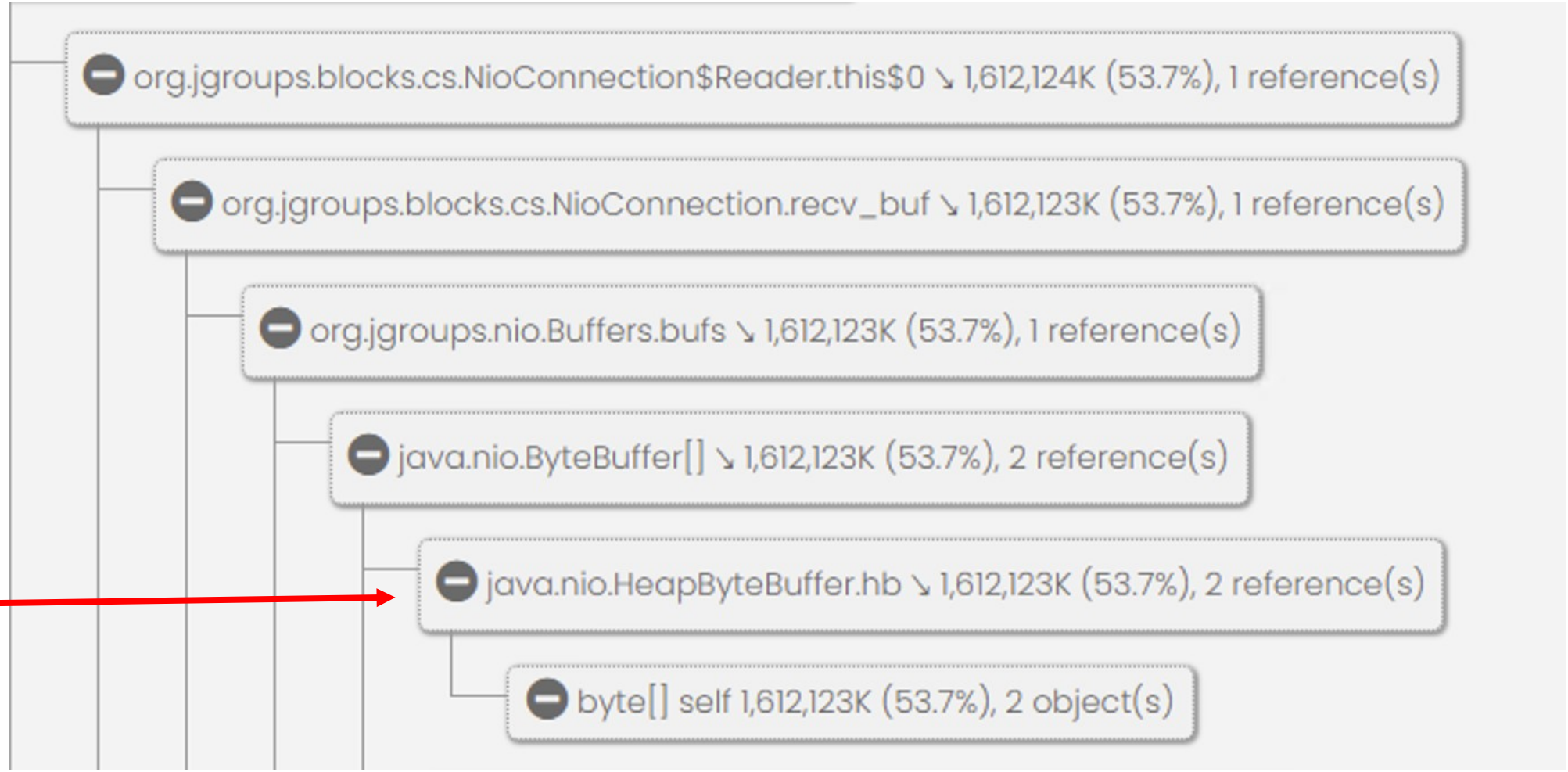
Analyze Heap Dumps with HeapHero



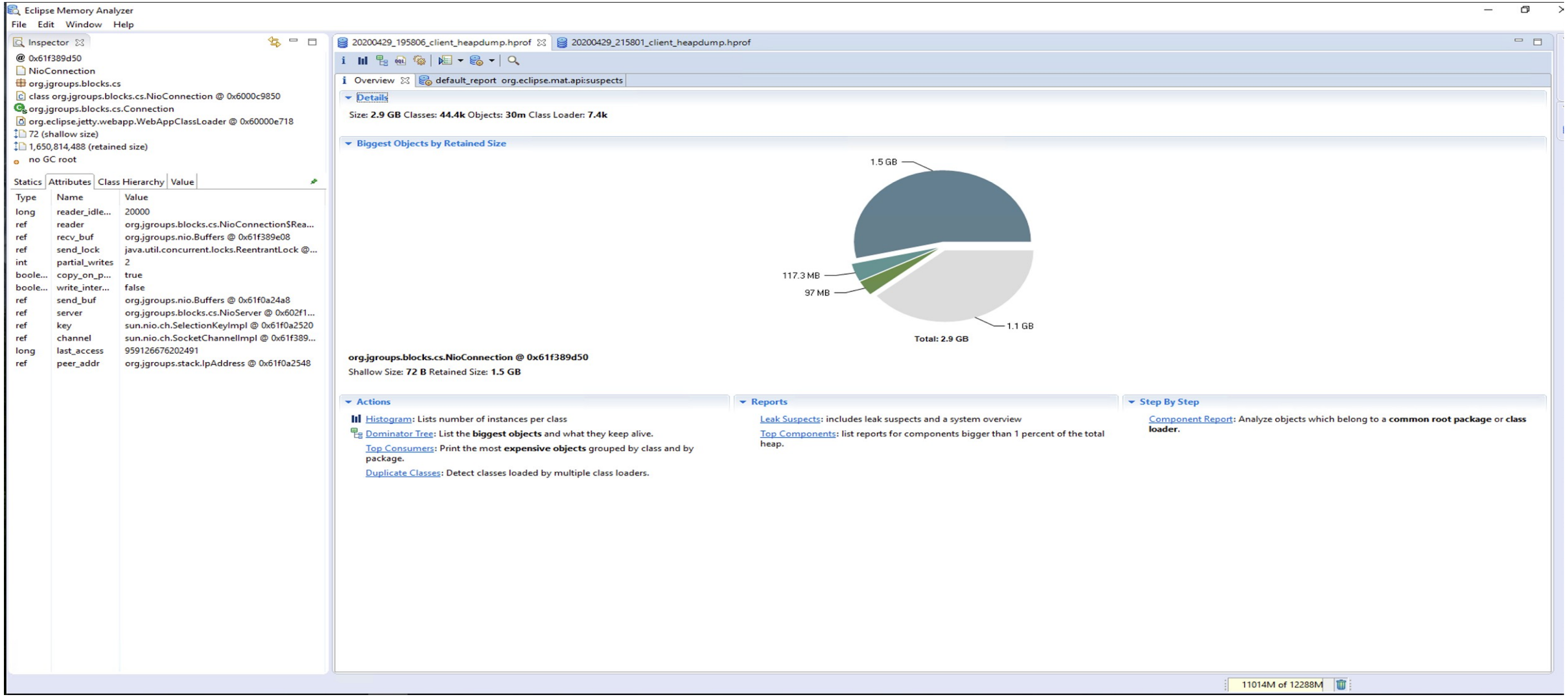
Analyze Heap Dumps with HeapHero

Analyze Heap Dumps with HeapHero

Explore the Object Tree:



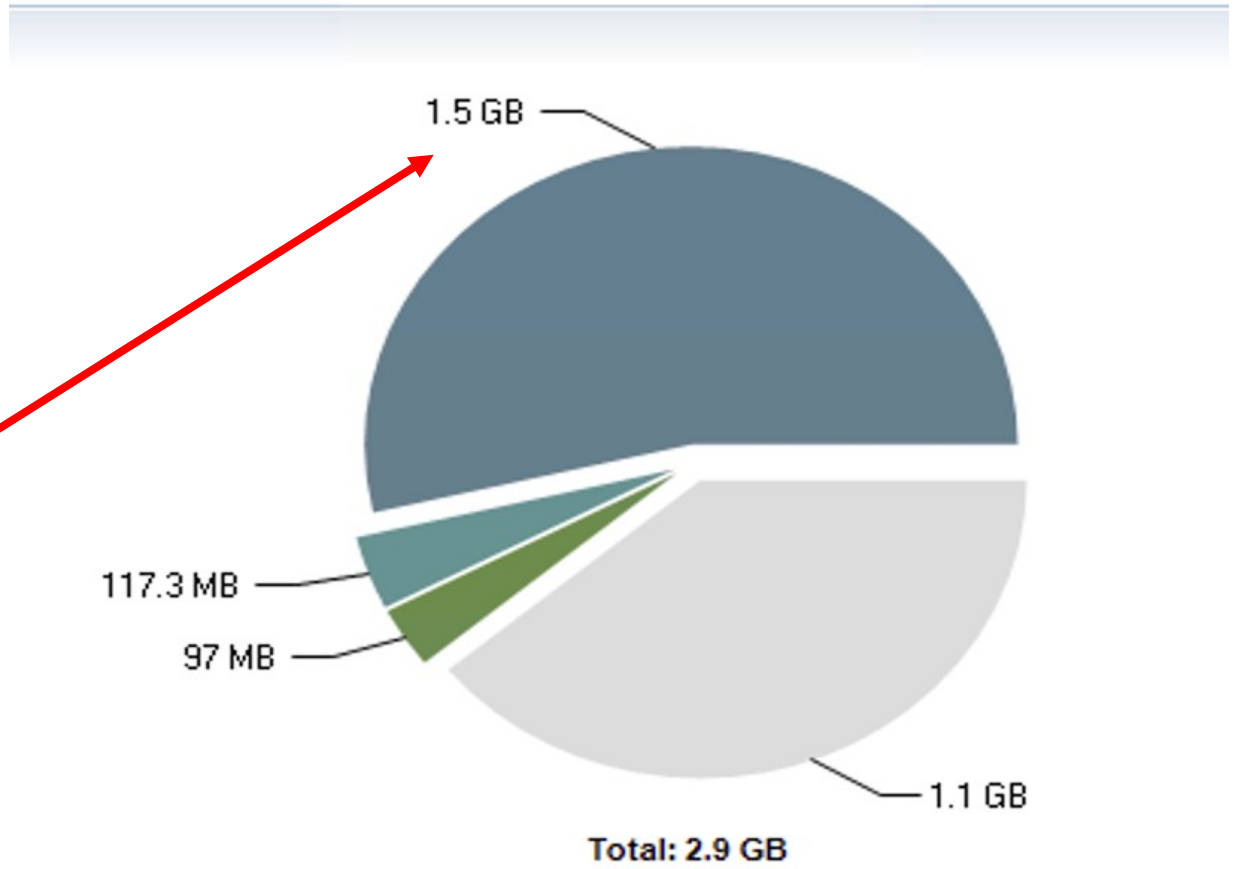
Analyze Heap Dumps with Eclipse MAT



Analyze Heap Dumps with Eclipse MAT

Graph showing biggest objects by their retained size:

We can see 1 object holding onto ~1.5GB



Analyze Heap Dumps with Eclipse MAT

Within the Leak Suspects Report we can see that `jgroups NioConnections` are holding onto ~1.5GB

Leak Suspects

Leak Suspects

System Overview

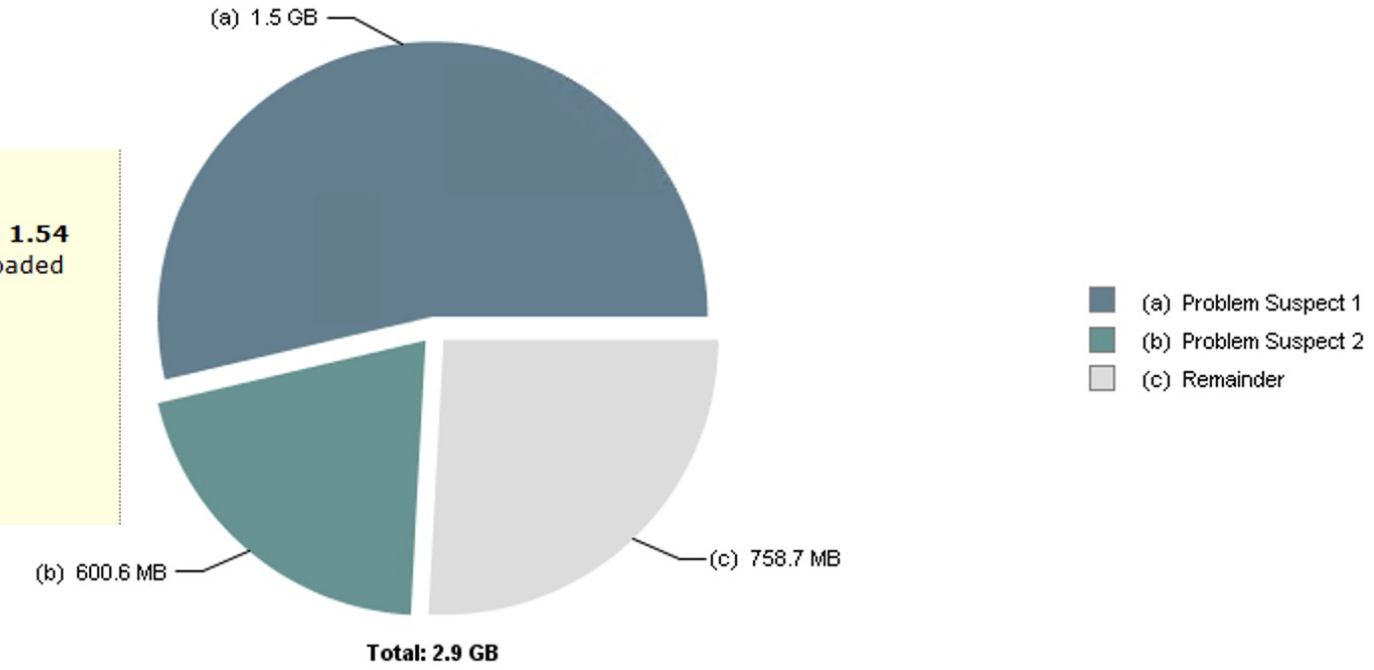
Leaks

Overview

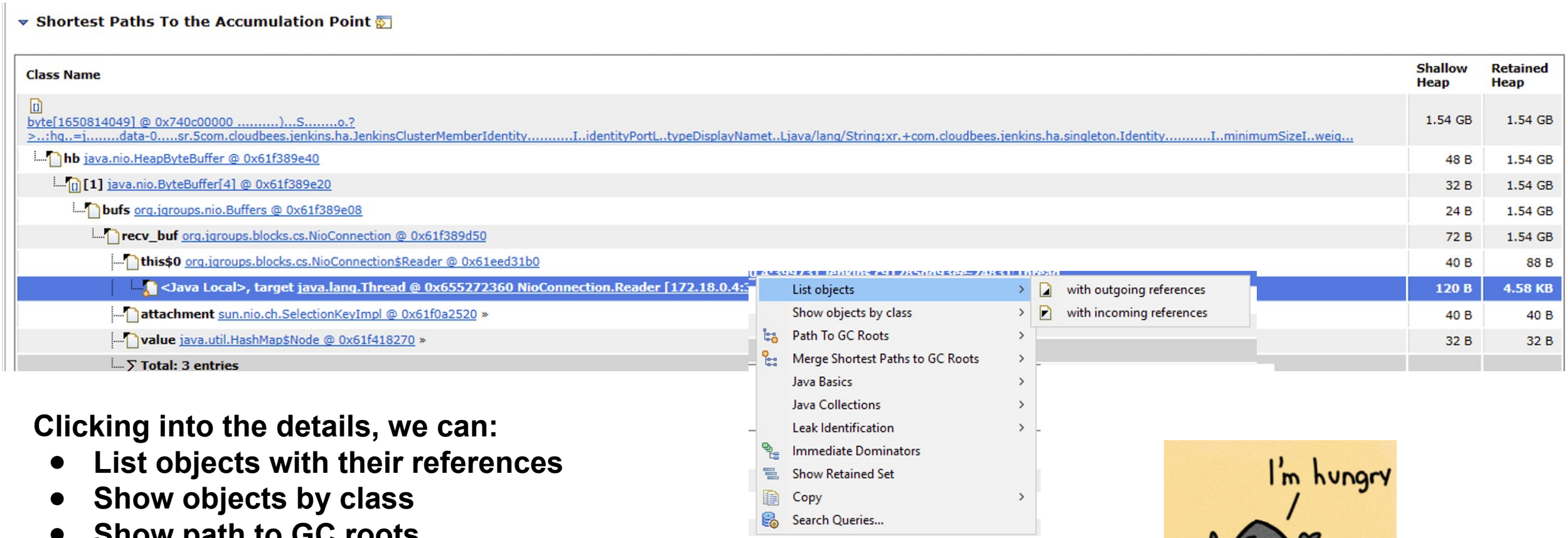
One instance of **"org.jgroups.blocks.cs.NioConnection"** loaded by **"org.eclipse.jetty.webapp.WebAppClassLoader @ 0x60000e718"** occupies **1.54 GB (53.67%)** bytes. The memory is accumulated in one instance of **"byte[]"** loaded by **"<system class loader>"**.

Keywords
byte[]
org.eclipse.jetty.webapp.WebAppClassLoader @ 0x60000e718
org.jgroups.blocks.cs.NioConnection

Details »



Analyze Heap Dumps with Eclipse MAT



Clicking into the details, we can:

- **List objects with their references**
- **Show objects by class**
- **Show path to GC roots**
- **Perform a search query**

Links for additional training on HeapHero / Eclipse MAT

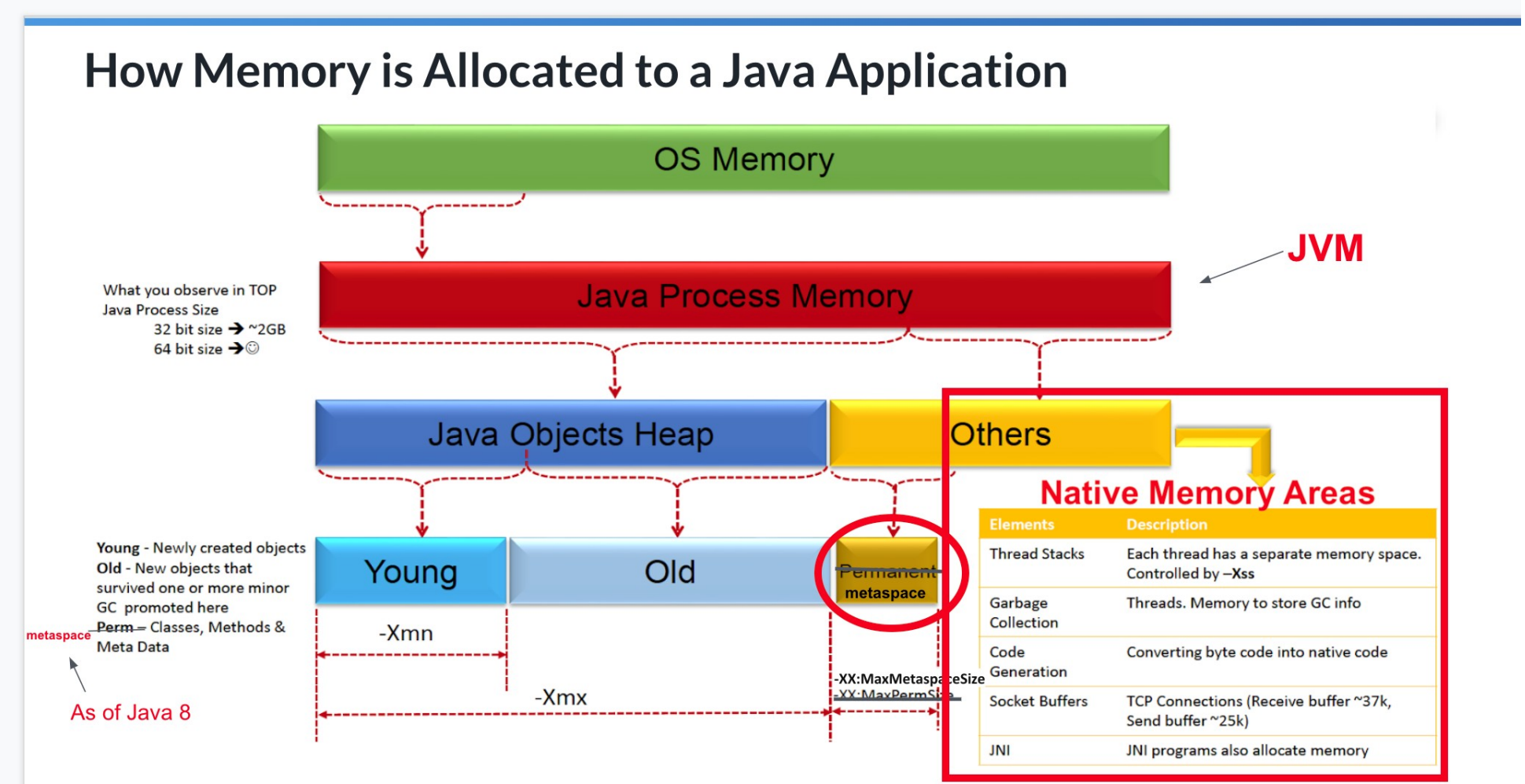
- Chris Grindstaff - Java Memory Analysis
- Kevin Grigorenko - Eclipse MAT Deep Dive
- Eclipse Documentation
- HeapHero User Manual
- HeapHero Blog - Tier1App Team

Diagnosing Memory Leaks...The Good, the Bad, and the Ugly

Java Heap Memory Leaks

DEVOPS WORLD
by CloudBees

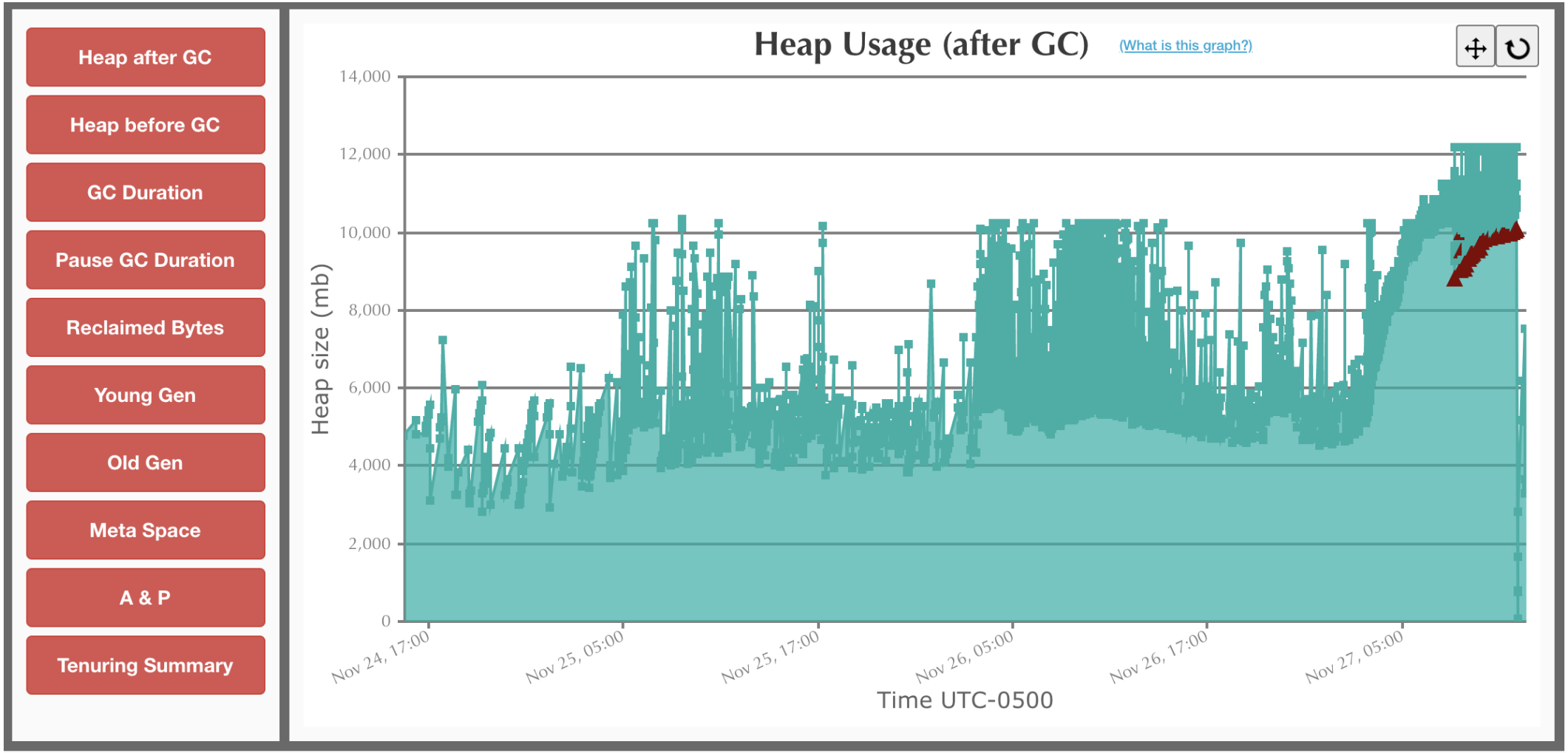
JVM (Java Virtual Machine) memory allocation



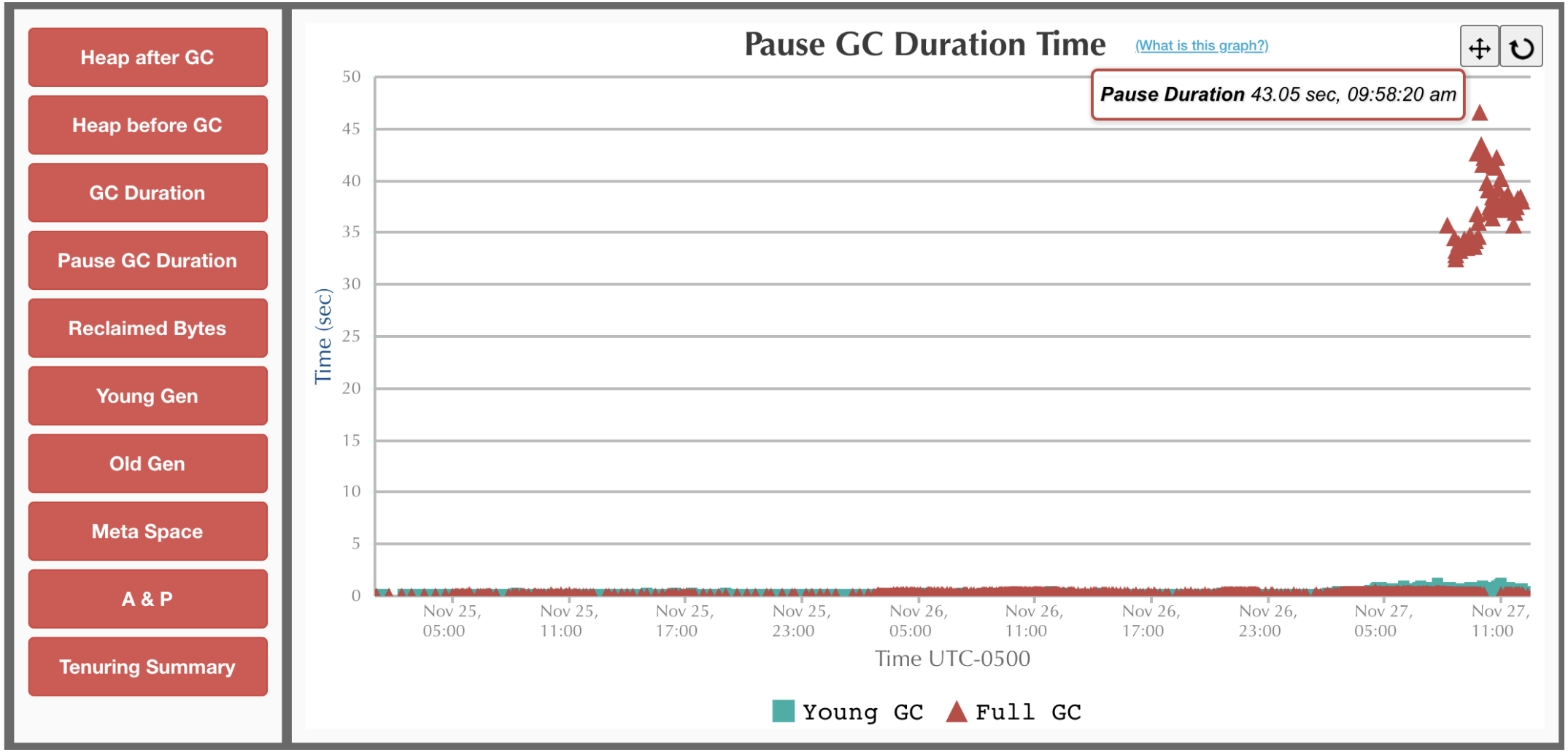
Heap related JVM arguments

- -Xms
  - Sets the initial heap size. Can be in KB, MB, or GB.
- -Xmx
  - Sets the maximum heap size. Can be in KB, MB, or GB.
- -XX:+UseContainerSupport
  - Tells the JVM to use the CPU and RAM allocated to the docker container rather than what is on the host server/VM for certain calculations such as default CPU and heap settings. Should be on by default for newer JDK versions.
- -XX:InitialRAMPercentage
  - Sets initial heap size as a percentage of total memory
- -XX:MaxRAMPercentage
  - Sets maximum heap size as a percentage of total memory

Heap memory leak GC graph



Heap memory leak GC pauses



Heap memory leak observed symptoms

top

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12354 jenkins   20   0   22.2g  16.6g   8336 S 605.9 53.2  11289:24 /usr/bin/java
```

top -H

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
21188 jenkins   20   0   22.2g  16.6g   8336 R  84.0 53.2 163:41.28 java
21189 jenkins   20   0   22.2g  16.6g   8336 R  84.0 53.2 163:48.31 java
21186 jenkins   20   0   22.2g  16.6g   8336 R  80.0 53.2 163:45.20 java
21192 jenkins   20   0   22.2g  16.6g   8336 R  80.0 53.2 163:53.75 java
21185 jenkins   20   0   22.2g  16.6g   8336 R  76.0 53.2 163:41.64 java
21187 jenkins   20   0   22.2g  16.6g   8336 R  76.0 53.2 163:38.22 java
21190 jenkins   20   0   22.2g  16.6g   8336 R  72.0 53.2 163:33.27 java
21191 jenkins   20   0   22.2g  16.6g   8336 R  68.0 53.2 163:40.76 java
```

504 Gateway Time-out

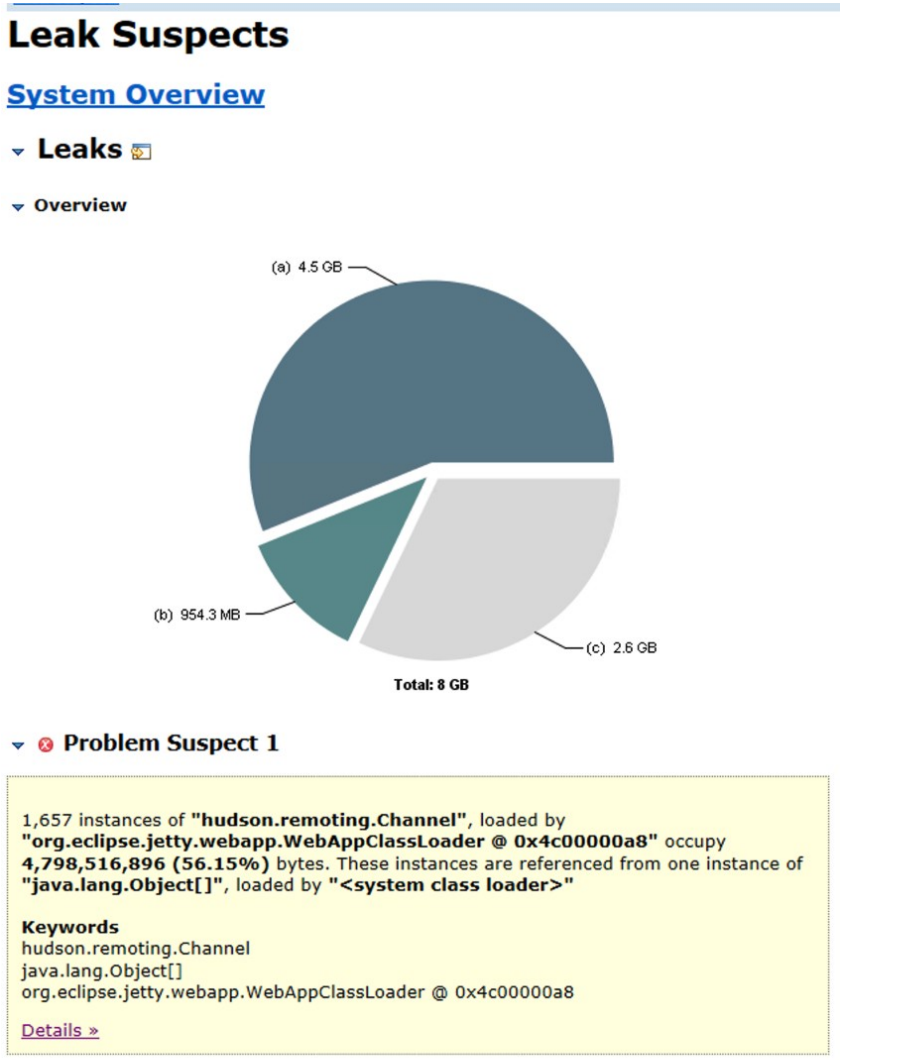
Thread Dump #1 RUNNABLE

Gang worker#6 (Parallel GC Threads)

threadId:0x00007f7be806b000 - nativeId:**0x52c7** - nativeId (decimal):21191

What to do after discovering the heap memory leak?

- Collect a heap dump while the heap is full
  - `-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath`
- Analyze it using your preferred tool: HeapHero, Eclipse MAT, YourKit, etc..
- We typically use MAT and run the Leak Suspects Report



▼ **Common Path To the Accumulation Point**

Class Name	Ref. Objects	Shallow Heap	Ref. Shallow Heap	Retained Heap
hudson.UDPBroadcastThread @ 0x531b1f340 Jenkins UDP 33848 monitoring thread **Thread**	1,394	136	256,496	744
jenkins hudson.model.Hudson @ 0x4c0150200	1,394	320	256,496	330,781,024
extensionLists java.util.concurrent.ConcurrentHashMap @ 0x4c2799070	1,394	64	256,496	119,344
table java.util.concurrent.ConcurrentHashMap$Node[512] @ 0x54eff0a70	1,394	2,064	256,496	119,280
[53] java.util.concurrent.ConcurrentHashMap$Node @ 0x4c27a6568	1,394	32	256,496	792
val hudson.ExtensionList @ 0x4c0ca2500	1,394	40	256,496	760
extensions java.util.ArrayList @ 0x4c0a535f8	1,394	24	256,496	632
elementData java.lang.Object[21] @ 0x4c0a53610	1,394	104	256,496	608
[20] hudson.ExtensionComponent @ 0x4c0a53678	1,394	24	256,496	24
instance org.jvnet.hudson.test.ChannelShutdownListener @ 0x4c0a53690	1,394	16	256,496	7,464
channels java.util.ArrayList @ 0x4c0a536a0	1,394	24	256,496	7,448
elementData java.lang.Object[1851] @ 0x628c12c88	**1,394**	**7,424**	**256,496**	**7,424**
[1654] hudson.remoting.Channel @ 0x69cc14e60	1	184	184	1,693,816
[366] hudson.remoting.Channel @ 0x4fa29dea0	1	184	184	1,348,776
[1368] hudson.remoting.Channel @ 0x5b0c34508	1	184	184	3,012,320
[1421] hudson.remoting.Channel @ 0x57a981588	1	184	184	4,340,856
[835] hudson.remoting.Channel @ 0x604b639a0	1	184	184	5,740,992
[1190] hudson.remoting.Channel @ 0x589523f78	1	184	184	1,637,712
[776] hudson.remoting.Channel @ 0x51881bfd0	1	184	184	4,699,896
[730] hudson.remoting.Channel @ 0x5ab2c4578	1	184	184	1,233,224
[308] hudson.remoting.Channel @ 0x54efa4560	1	184	184	4,249,552
[231] hudson.remoting.Channel @ 0x4dd204408	1	184	184	4,722,896
[1382] hudson.remoting.Channel @ 0x53ea52dc8	1	184	184	2,310,552
[1140] hudson.remoting.Channel @ 0x6250400f8	1	184	184	5,759,464
[1129] hudson.remoting.Channel @ 0x5b5d79210	1	184	184	1,253,720
[180] hudson.remoting.Channel @ 0x4d5012230	1	184	184	2,020,928
[1424] hudson.remoting.Channel @ 0x570dcbfe8	1	184	184	5,255,400
[1423] hudson.remoting.Channel @ 0x5bcf41ab0	1	184	184	1,131,280
[1321] hudson.remoting.Channel @ 0x5f62431b8	1	184	184	1,526,616
[1147] hudson.remoting.Channel @ 0x562bedad0	1	184	184	1,105,256
[739] hudson.remoting.Channel @ 0x54e598b50	1	184	184	1,230,960
[1282] hudson.remoting.Channel @ 0x5ee5f3398	1	184	184	1,666,000
[741] hudson.remoting.Channel @ 0x524dc8498	1	184	184	1,581,216
[202] hudson.remoting.Channel @ 0x5249ac8b8	1	184	184	118,360
[358] hudson.remoting.Channel @ 0x51e488110	1	184	184	2,916,808
[879] hudson.remoting.Channel @ 0x4fb51bac0	1	184	184	732,944
[1109] hudson.remoting.Channel @ 0x5a59f7c60	1	184	184	1,252,320
Total: 25 of 1,394 entries; 1,369 more	**1,394**	**256,496**	**256,496**	

Next steps for Jenkins after heap analysis

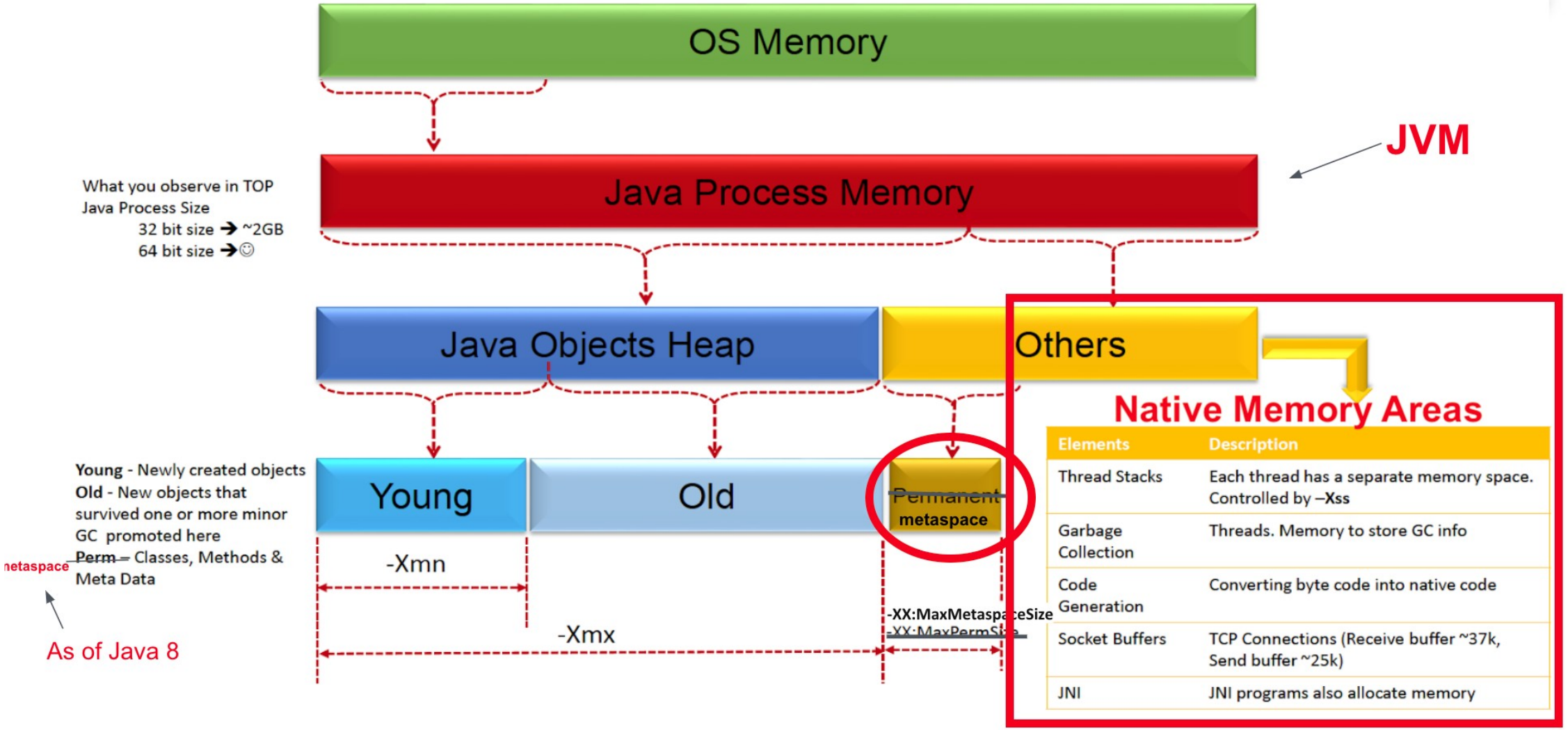
- If leak is from a plugin
  - Update plugin
  - Verify the plugins configuration
  - Check the Jenkins Jira site for open bugs related to leaks
  - If no bug is reported, file one
  - In the case of a CloudBees plugin, submit a support ticket
- If leak is from a pipeline job, check for:
  - Large objects being loaded
  - Loops loading many objects

Diagnosing Memory Leaks...The Good, the Bad, and the Ugly

Metaspace Memory Leaks

DEVOPS WORLD
by CloudBees

How Memory is Allocated to a Java Application



Essential JDK Tools for Measuring Metaspace

JSTAT

- Utility tool in the JDK
- Provides JVM performance-related statistics like garbage collection, compilation activities.

JCMD

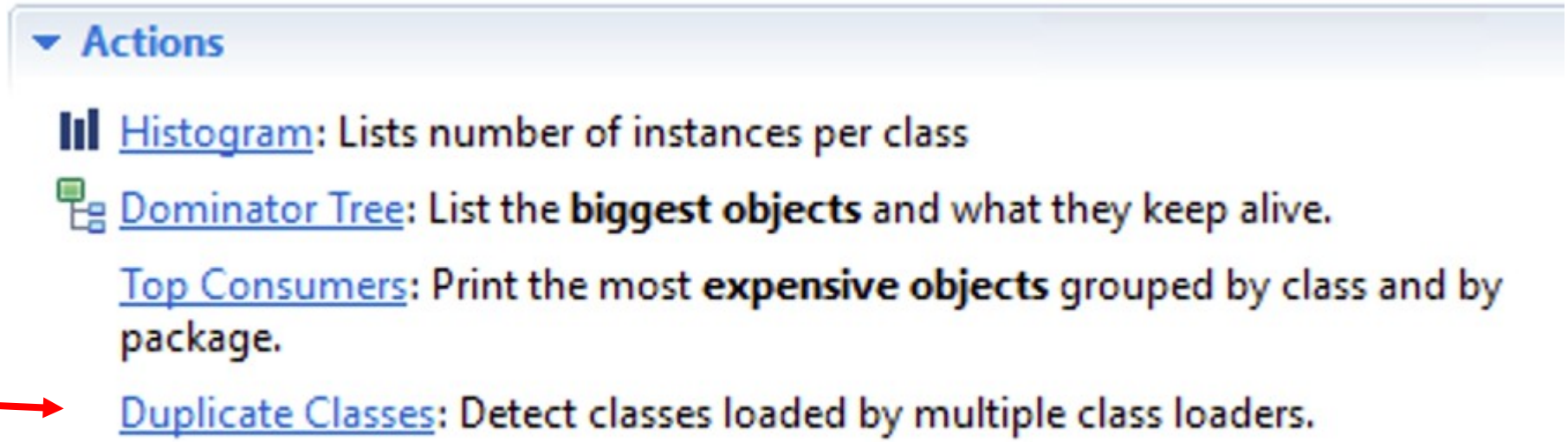
- Used to send diagnostic command requests to the JVM
- Must be used on the same machine where the JVM is running
- Must run as the same user and group id's as the JVM

Essential JDK Tools for Measuring Metaspace

```
jstat -gc ${PID}
```

```
jcmd ${PID} GC.class_stats > GC_class_output.log
```

Heap Dump and analyze Duplicate Classes with MAT



Essential JDK Tools for Measuring Metaspace

```
jstat -gc ${PID}
```

```
jenkins jstat -gc 32215
 S0C     S1C       S0U     S1U
 0.0    75776.0   0.0    75776.0
```

. . .

```
MC                 MU
1944268.0 1055786.4
```

Metaspace Capacity (Kb)
Metaspace Usage (Kb)

-XX:MetaspaceSize=
-XX:MaxMetaspaceSize=

Essential JDK Tools for Measuring Metaspace

```
jcmd ${PID} GC.class_stats > GC_class_output.log
```

```
jenkins jcmd 32215 GC.class_stats
32215:
Index Super  InstBytes KlassBytes
    1    -1  311568552         464
    2    -1  305263360         464
    3    -1  117440760         464
    4    10  116563296         608
    5    10   59643200         552
    6    38   51787584         784
```

```
Total ClassName
  592 [C
  592 [B
  592 [Ljava.lang.Object;
74232 java.lang.String
 4392 java.util.concurrent.ConcurrentHashMap$Node
 4952 com.google.common.cache.LocalCache$StrongWriteEntry
 4616 java.util.HashMap$Node
  600 [Ljava.util.HashMap$Node;
 41256 java.util.ArrayList
 6864 java.lang.Object
 29040 java.lang.reflect.Method
118360 org.codehaus.groovy.ast.ClassNode
 4408 com.google.common.cache.LocalCache$StrongValueReference
 11600 java.util.concurrent.atomic.AtomicLong
```

Essential JDK Tools for Measuring Metaspace

```
jcmd ${PID} GC.class_stats > GC_class_output.log
```

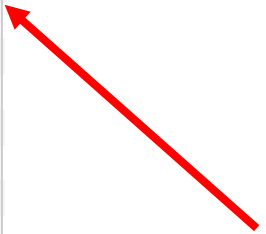
lots of duplicate classes loaded here....

```
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
18992 groovyx.net.http.HTTPBuilder
```

Essential JDK Tools for Measuring Metaspace

Analysis of Heap Dump in Eclipse MAT Duplicate Classes feature:

Class Name	Count
<Regex>	<Numeric>
groovyx.net.http.AuthConfig	16,362
groovyx.net.http.ContentEncodingRegistry	16,362
groovyx.net.http.EncoderRegistry	16,362
groovyx.net.http.HTTPBuilder	16,362
groovyx.net.http.HTTPBuilder$RequestConfigDelegate	16,362
groovyx.net.http.HttpResponseDecorator	16,362
groovyx.net.http.HttpResponseException	16,362
groovyx.net.http.Method	16,362
groovyx.net.http.ParserRegistry	16,362
groovyx.net.http.RESTClient	16,362
groovyx.net.http.ResponseParseException	16,362
groovyx.net.http.URIBuilder	16,362
java.lang.invoke.LambdaForm$BMH	583
java.lang.invoke.LambdaForm$DMH	146
WorkflowScript	104
org.jenkinsci.plugins.pipeline.modeldefinition.ModelInterpreter	102
org.jenkinsci.plugins.pipeline.modeldefinition.ModelInterpreter$_createWithCredentialsP	102
org.jenkinsci.plugins.pipeline.modeldefinition.agent.CheckoutScript	102
org.jenkinsci.plugins.pipeline.modeldefinition.agent.impl.LabelScript	102
org.jenkinsci.plugins.docker.workflow.Docker	97
org.jenkinsci.plugins.docker.workflow.Docker$Container	97
org.jenkinsci.plugins.docker.workflow.Docker$Image	97
org.jenkinsci.plugins.pipeline.modeldefinition.agent.impl.AbstractDockerPipelineScript	96
org.jenkinsci.plugins.pipeline.modeldefinition.agent.CheckoutScript$doCheckout	95
Total: 24 of 1,233 entries; 1,209 more	**201,390**



Essential JDK Tools for Measuring Metaspace

Led to multiple PR's for Script Security:

https://github.com/jenkinsci/script-security-plugin/pull/252

https://github.com/jenkinsci/script-security-plugin/pull/253

Full description of the issue here:

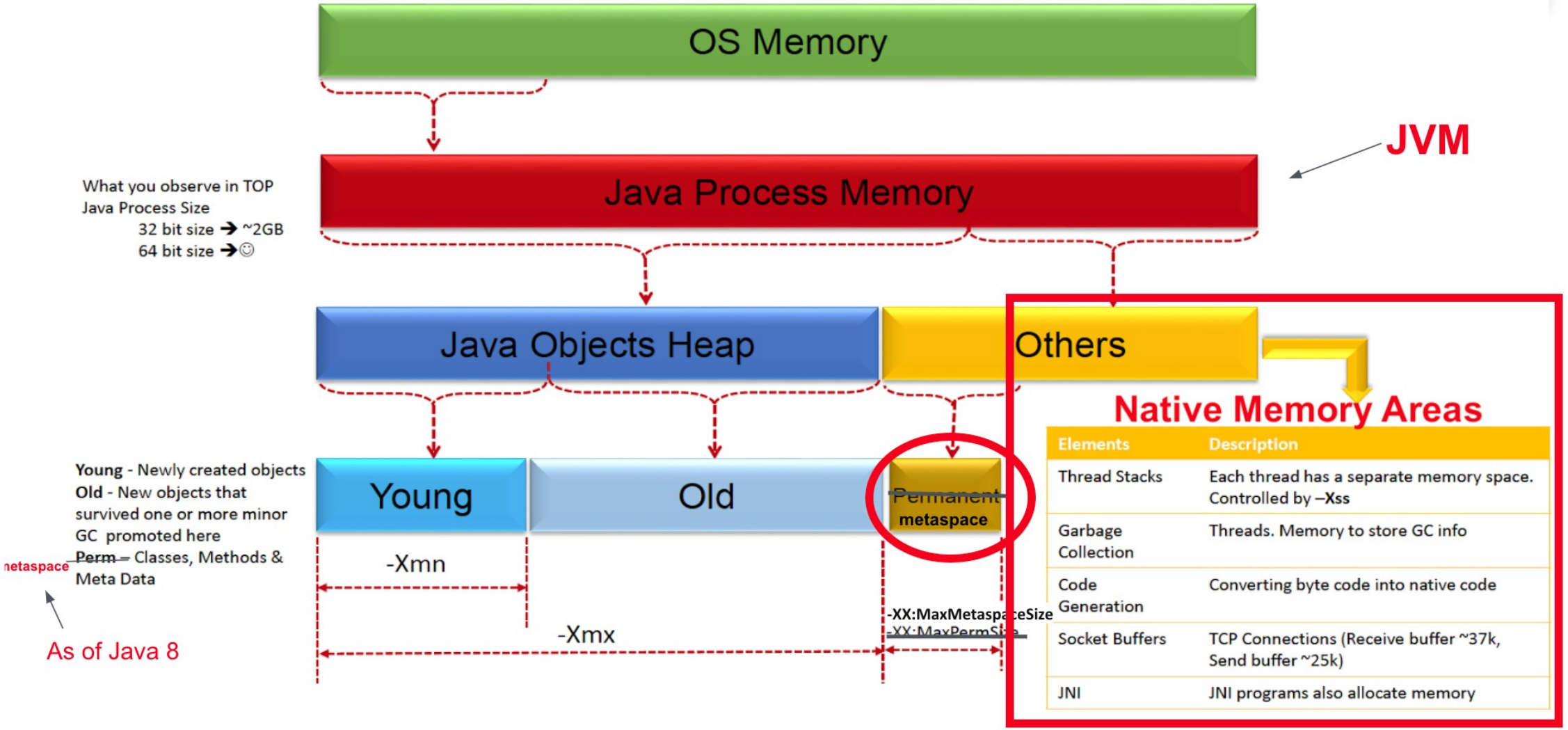
https://support.cloudbees.com/hc/en-us/articles/360029574172-Metaspace-Leak-Due-to-classes-not-being-cleaned-up

Diagnosing Memory Leaks...The Good, the Bad, and the Ugly

Java Native Memory Tracking

DEVOPS WORLD
by CloudBees

How Memory is Allocated to a Java Application



Elements	Description
Thread Stacks	Each thread has a separate memory space. Controlled by –Xss
Garbage Collection	Threads. Memory to store GC info
Code Generation	Converting byte code into native code
Socket Buffers	TCP Connections (Receive buffer ~37k, Send buffer ~25k)
JNI	JNI programs also allocate memory

Configure NMT Tracking

Oracle Documentation on NMT

-XX:NativeMemoryTracking=summary

To start baseline run the following **jcmd** command:

```
jcmd <pid> VM.native_memory baseline
```

To gather a summary, run the following **jcmd** command:

```
jcmd $PID VM.native_memory summary
```

NMT Summary

```
Native Memory Tracking:

Total: reserved=9769936KB +130080KB, committed=8644184KB +212676KB

-                 Java Heap (reserved=7340032KB, committed=7340032KB)
                            (mmap: reserved=7340032KB, committed=7340032KB)

-                     Class (reserved=1279073KB +47364KB, committed=262493KB +52484KB)
                            (classes #36038 +5114)
                            (malloc=9313KB +2308KB #183403 +54470)
                            (mmap: reserved=1269760KB +45056KB, committed=253180KB +50176KB)

-                    Thread (reserved=267357KB -41298KB, committed=267357KB -41298KB)
                            (thread #267 -40)
                            (stack: reserved=266176KB -41120KB, committed=266176KB -41120KB)
                            (malloc=870KB -132KB #1338 -200)
                            (arena=312KB -47 #529 -80)
```

NMT Summary Continued...

```
-                     Code (reserved=278652KB +13258KB, committed=183820KB +90738KB)
                           (malloc=29052KB +13258KB #30409 +10291)
                           (mmap: reserved=249600KB, committed=154768KB +77480KB)

-                       GC (reserved=371875KB +18969KB, committed=371875KB +18969KB)
                           (malloc=66723KB +18969KB #364357 +127871)
                           (mmap: reserved=305152KB, committed=305152KB)

-                 Compiler (reserved=768KB -205KB, committed=768KB -205KB)
                           (malloc=637KB -205KB #3163 +746)
                           (arena=131KB #6)

-                 Internal (reserved=164392KB +103183KB, committed=164388KB +103179KB)
                           (malloc=164356KB +103179KB #69335 +8065)
                           (mmap: reserved=36KB +4KB, committed=32KB)
```

NMT Summary Explained

- NMT takes a baseline then measures **against** baseline
- 11 different areas of memory consumption from the JVM.
- Most notably, “Java Heap” is the amount of heap space allocated to the JVM.
- “Class” can be traced to metaspace, as this is where class metadata is stored.
- Other 9 areas should hover around 10-250MB respectively. When we see areas of native memory above 1GB it is considered abnormal for Jenkins.

Script it!

- Run Hourly to diagnose an NMT memory leak
- 10% Overhead to the JVM
- Add to a crontab:
- 0 * * * * JENKINS_HOME='/path/to/jenkins-home' $JENKINS_HOME/support


```
#!/bin/bash

TSTAMP="$(date +'%Y%m%d_%H%M%S')"

jenkinsPid="$(pgrep -o java)"

nmtLog="$JENKINS_HOME/support/nmt.log"

echo $TSTAMP $JENKINS_CLUSTER_ID >> $nmtLog


jcmd $jenkinsPid VM.native_memory summary >> $nmtLog
```

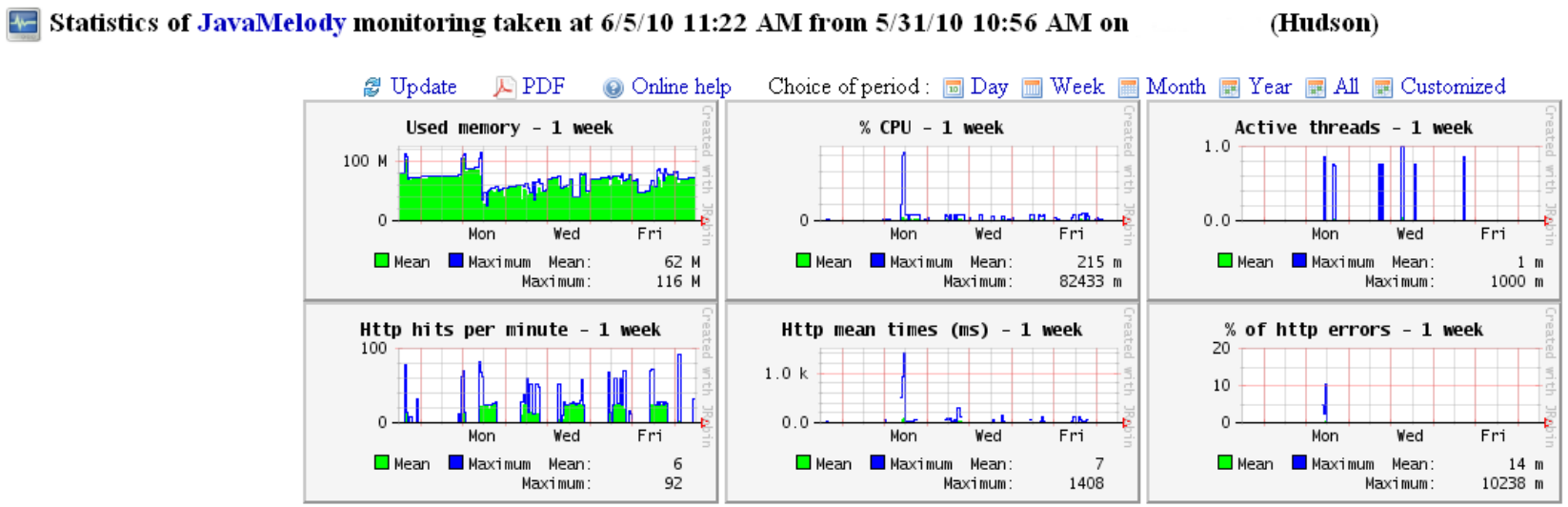
Diagnosing Memory Leaks...The Good, the Bad, and the Ugly

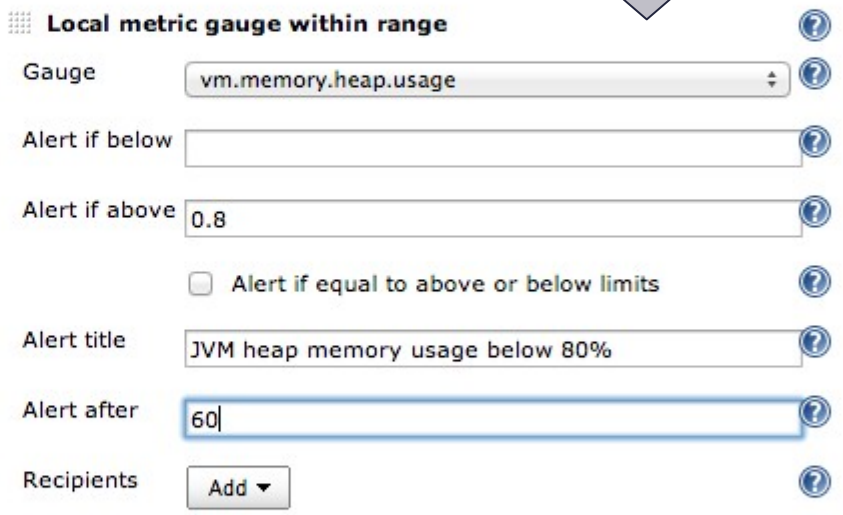
Monitoring Best Practices

DEVOPS WORLD
by CloudBees

Jenkins Plugins

- One of the most popular is the Monitoring plugin
  - Provides stats and graphs on CPU, memory, threads, etc
- CloudBees Monitoring plugin for CB customers
  - Can configure alerts for when something strays from the norm
  - Example alert: JVM heap memory usage is over 80% for more than a minute
- The main problem with plugins is they can't be used if Jenkins is down

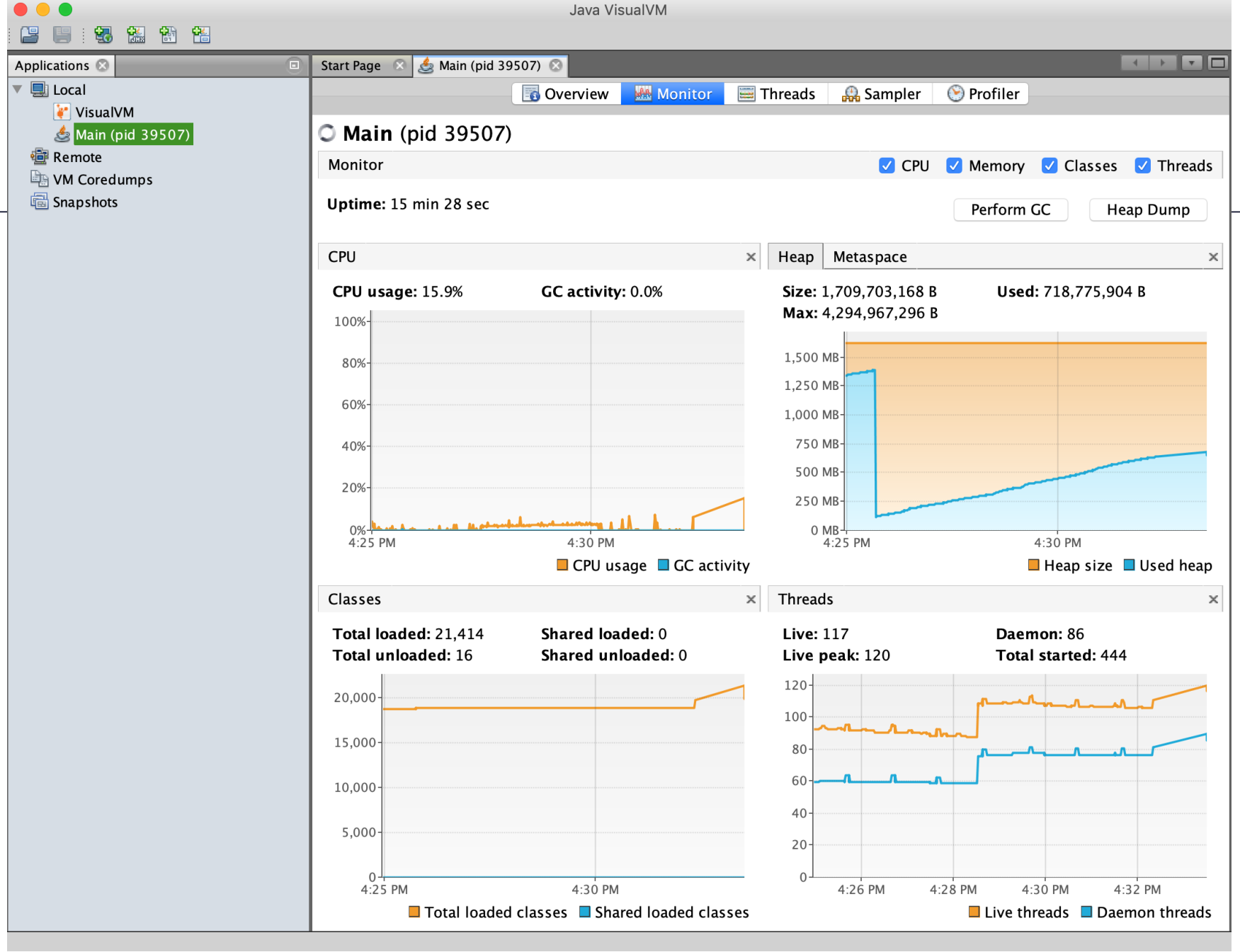




DEVOPSWORLD by CloudBees

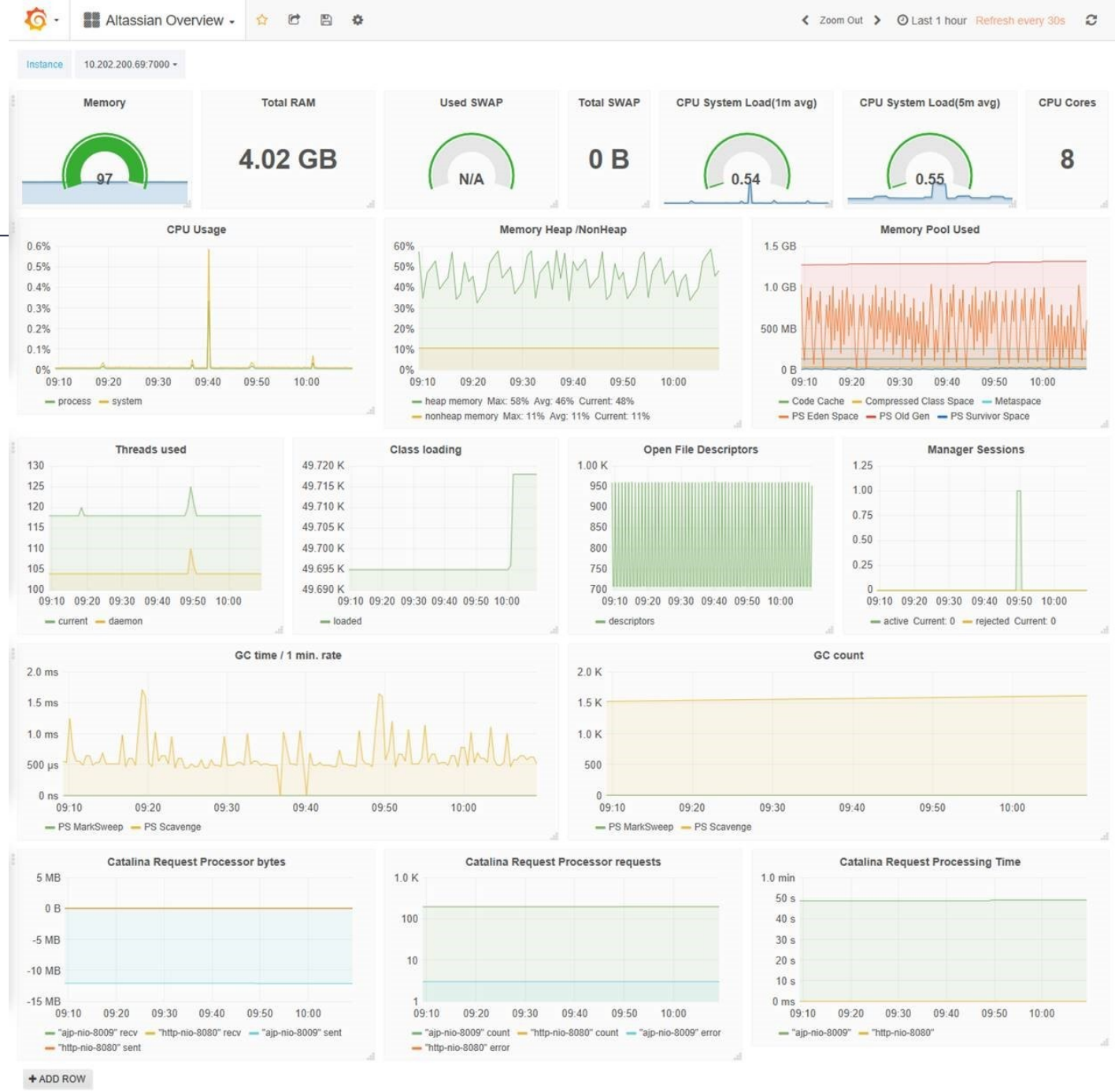
JMX monitoring

- Built into the JVM, but needs to be enabled if using remote monitoring with: `com.sun.management.jmxremote.port=portNum`
- Java VisualVM is bundled with JDK 8+ and connects via JMX
  - Can live monitor CPU, memory, classes, threads. Can also take heap and thread dumps
  - VisualVM does not have alerting.



Java Agents

- Java Agents are jar files that inject code into your JVM to run
- Added using java arg: `-javaagent: /path/to/agent.jar`
- Can provide better metrics, but have overhead
- Datadog
  - Very robust and customizable UI
  - Monitors many metrics and has alerts
  - Paid
- Prometheus
  - Text only by default. Can be hooked into visualization tools like Grafana
  - Many different pieces make it more customizable, but more complicated
  - Free and open source
- Pick a tool with alerting capabilities, get a baseline of memory usage, set an alert to trigger when memory is above the threshold for an extended time



Thanks!

Q&A