ADDO

ALL DAY DEVOPS

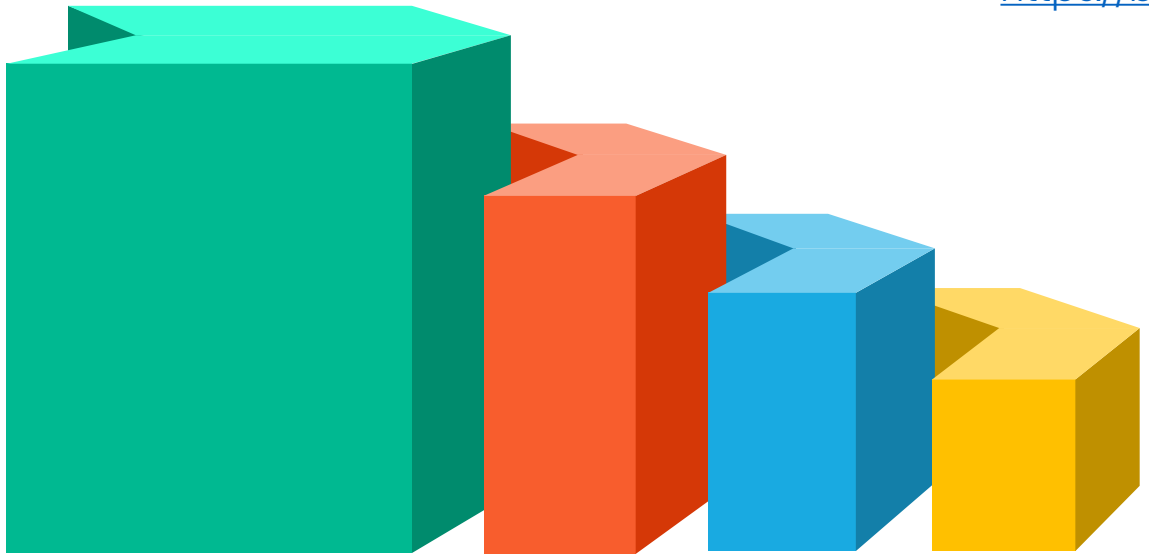
NOVEMBER 6, 2019

Troubleshooting Real Production Problems

Ram Lakshmanan
Architect: GCeasy.io, fastThread.io, HeapHero.io

Troubleshooting CPU spike

https://blog.fastthread.io/2018/12/13/how-to-troubleshoot-cpu-problems/



Step 1: Confirm

Don't trust anyone

```
top - 23:13:26 up 102 days, 21:09,  2 users,  load average: 2.91, 2.99, 2.55
Tasks:  99 total,   1 running,  98 sleeping,   0 stopped,   0 zombie
Cpu(s):100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8178640k total,  3821576k used,  4357064k free,   158700k buffers
Swap:        0k total,        0k used,        0k free,   646524k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
31294 ec2-user  20   0 4415m  22m  14m S 199.1  0.3   0:28.53 java
 3108 newrelic  20   0  239m 7708 4784 S  0.7  0.1  59:09.51 nrsysmond
15153 tomcat    20   0 9132m 2.6g  17m S  0.3 33.7 134:34.52 java
    1 root      20   0 19640 2676 2344 S  0.0  0.0   0:07.55 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:10.95 ksoftirqd/0
    5 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0     0    0    0 S  0.0  0.0   1:27.11 kworker/u30:0
    7 root      20   0     0    0    0 S  0.0  0.0   3:22.96 rcu_sched
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      RT   0     0    0    0 S  0.0  0.0   0:01.96 migration/0
   10 root      RT   0     0    0    0 S  0.0  0.0   0:01.44 migration/1
   11 root      20   0     0    0    0 S  0.0  0.0   0:14.63 ksoftirqd/1
   13 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/1:0H
   14 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
   15 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   18 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 perf
   22 root      20   0     0    0    0 S  0.0  0.0   0:00.00 xenwatch
   23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 xenbus
  134 root      20   0     0    0    0 S  0.0  0.0   0:03.71 khungtaskd
  135 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 writeback
```

'top' tool is your good friend

Step 2: Identify Threads

top –H –p {pid}

ec2-user@ip-172-31-15-129:/usr/share/tomcat8/logs

```
top - 23:14:36 up 102 days, 21:10,  2 users,  load average: 2.97, 3.00, 2.58
Tasks:  15 total,   3 running,  12 sleeping,   0 stopped,   0 zombie
Cpu(s):100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8178640k total,  3822428k used,  4356212k free,   158700k buffers
Swap:        0k total,        0k used,        0k free,   646536k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
31306 ec2-user  20   0 4415m  22m  14m R 69.3  0.3   0:56.53 java
31307 ec2-user  20   0 4415m  22m  14m R 65.6  0.3   0:56.83 java
31308 ec2-user  20   0 4415m  22m  14m R 64.0  0.3   0:55.48 java
31294 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31295 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.05 java
31296 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31297 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31298 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31299 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31300 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31301 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31302 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31303 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31304 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.00 java
31305 ec2-user  20   0 4415m  22m  14m S  0.0  0.3   0:00.02 java
```

Example:
top -H -p 31294

Step 3: Capture thread dumps

https://blog.fastthread.io/2016/06/06/how-to-take-thread-dumps-7-options/

01
jstack (since Java 5)

jstack -l <pid> > /tmp/threadDump.txt

02
kill -3

Kill -3 <pid>
Useful when only JRE is installed

03
jVisualVM

JDK tool. Now Open source.
GUI based option.

04
JMC

JDK tool. Now Open source.
GUI based option.

05
Windows (Ctrl + Break)

Helpful during development phase

06
ThreadMXBean

Programmatic way to capture thread dumps

07
APM Tools

Few APM Tools does provide this support

08 ⭐
Jcmd (since Java 7)

jcmd <pid> Thread.print > /tmp/threadDump.txt

Anatomy of thread dump

1 2019-02-26 17:13:23

2 Full thread dump Java HotSpot(TM) 64-Bit Server VM (23.7-b01 mixed mode):

3

```
"InvoiceThread-A996" prio=10 tid=0x00002b7cfc6fb000 nid=0x4479 runnable [0x00002b7d17ab8000]
   java.lang.Thread.State: RUNNABLE
        at com.buggycompany.rt.util.ItinerarySegmentProcessor.setConnectingFlight(ItinerarySegmentProcessor.java:380)
        at com.buggycompany.rt.util.ItinerarySegmentProcessor.processTripType0(ItinerarySegmentProcessor.java:366)
        at com.buggycompany.rt.util.ItinerarySegmentProcessor.processItineraryByTripType(ItinerarySegmentProcessor.java:254)
        at com.buggycompany.rt.util.ItinerarySegmentProcessor.templateMethod(ItinerarySegmentProcessor.java:399)
        at com.buggycompany.qc.gds.InvoiceGeneratedFacade.readTicketImage(InvoiceGeneratedFacade.java:252)
        at com.buggycompany.qc.gds.InvoiceGeneratedFacade.doOrchestrate(InvoiceGeneratedFacade.java:151)
        at com.buggycompany.framework.gdstask.BaseGDSFacade.orchestrate(BaseGDSFacade.java:32)
        at com.buggycompany.framework.gdstask.BaseGDSFacade.doWork(BaseGDSFacade.java:22)
        at com.buggycompany.framework.concurrent.BuggycompanyCallable.call(buggycompanyCallable.java:80)
        at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
        at java.util.concurrent.FutureTask.run(FutureTask.java:166)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
        at java.lang.Thread.run(Thread.java:722)
```

```
"Reconnection-1" prio=10 tid=0x00007f0442e10800 nid=0x112a waiting on condition [0x00007f042f719000]
   java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x007b3953a98> (a java.util.concurrent.locks.AbstractQueuedSynchr)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:186)
        at java.lang.Thread.run(Thread.java:722)
   :
   :
```

1. Timestamp at which thread dump was triggered
2. JVM Version info
3. Thread Details - <<details in following slides>>

1 2 3 4 5

"InvoiceThread-A996" prio=10 tid=0x00002b7cfc6fb000 nid=0x4479 runnable [0x00002b7d17ab8000]

6 java.lang.Thread.State: RUNNABLE

7
```
at com.buggycompany.rt.util.ItinerarySegmentProcessor.setConnectingFlight(ItinerarySegmentProcessor.java:380)
at com.buggycompany.rt.util.ItinerarySegmentProcessor.processTripType0(ItinerarySegmentProcessor.java:366)
at com.buggycompany.rt.util.ItinerarySegmentProcessor.processItineraryByTripType(ItinerarySegmentProcessor.java:254)
at com.buggycompany.rt.util.ItinerarySegmentProcessor.templateMethod(ItinerarySegmentProcessor.java:399)
at com.buggycompany.qc.gds.InvoiceGeneratedFacade.readTicketImage(InvoiceGeneratedFacade.java:252)
at com.buggycompany.qc.gds.InvoiceGeneratedFacade.doOrchestrate(InvoiceGeneratedFacade.java:151)
at com.buggycompany.framework.gdstask.BaseGDSFacade.orchestrate(BaseGDSFacade.java:32)
at com.buggycompany.framework.gdstask.BaseGDSFacade.doWork(BaseGDSFacade.java:22)
at com.buggycompany.framework.concurrent.BuggycompanyCallable.call(buggycompanyCallable.java:80)
at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
at java.util.concurrent.FutureTask.run(FutureTask.java:166)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)
```

1 **Thread Name** - *InvoiceThread-A996*

2 **Priority** - Can have values from 1 to *10*

3 **Thread Id** - 0x00002b7cfc6fb000 – Unique ID assigned by JVM. It's returned by calling the Thread.getId() method.

4 **Native Id** - 0x4479 - This ID is highly platform dependent. On Linux, it's the pid of the thread. On Windows, it's simply the OS-level thread I
a process. On Mac OS X, it is said to be the native pthread_t value.

5 **Address space** - 0x00002b7d17ab8000 -

6 **Thread State** - RUNNABLE

7 **Stack trace** -

6 thread states

01 NEW

02 TERMINATED

03 RUNNABLE

04 BLOCKED

Thread 2: BLOCKED

```
public void synchronized getData() {
    makeDBCall();
}
```

Thread 1: Runnable

05 WAITING

```
wait();
```

06 TIMED_WAITING

```
Thread.sleep(10);
```

Step 4: Identify lines of code causing CPU spike

Thread Ids: 31306, 31307, 31308

High CPU consuming Threads Ids reported in ‘top –H’.

Let’s look up these thread Ids in Thread dump

HexaDecimal equivalent:

- 31306 → 7a4a
- 31307 → 7a4b
- 31308 → 7a4c

Source code

```
1: package com.buggyapp.cpuspike;
2:
3: /**
4: *
5: * @author Test User
6: */
7: public class Object1 {
8:
9:          public static void execute() {
10:
11:                 while (true) {
12:
13:                         doSomething();
14:                 }
15:         }
16:
17:         public static void doSomething() {
18:
19:         }
20: }
```

‘Free’ Thread dump analysis tools

Freely available Thread dump analysis tools

01

FastThread

http://fastThread.io/

02

Samurai

http://samuraism.jp/samurai/en/index.html

03

IBM Thread & Monitor analyzer

https://developer.ibm.com/javasdk/tools/

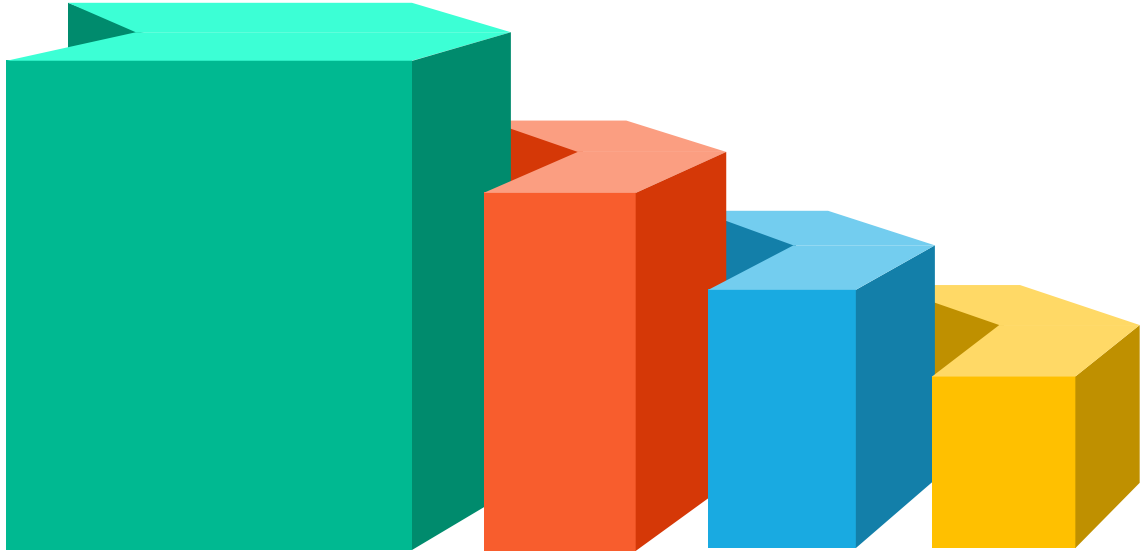
04

Visual VM

https://visualvm.github.io/

CPU spike in a major trading application

Troubleshooting unresponsive app

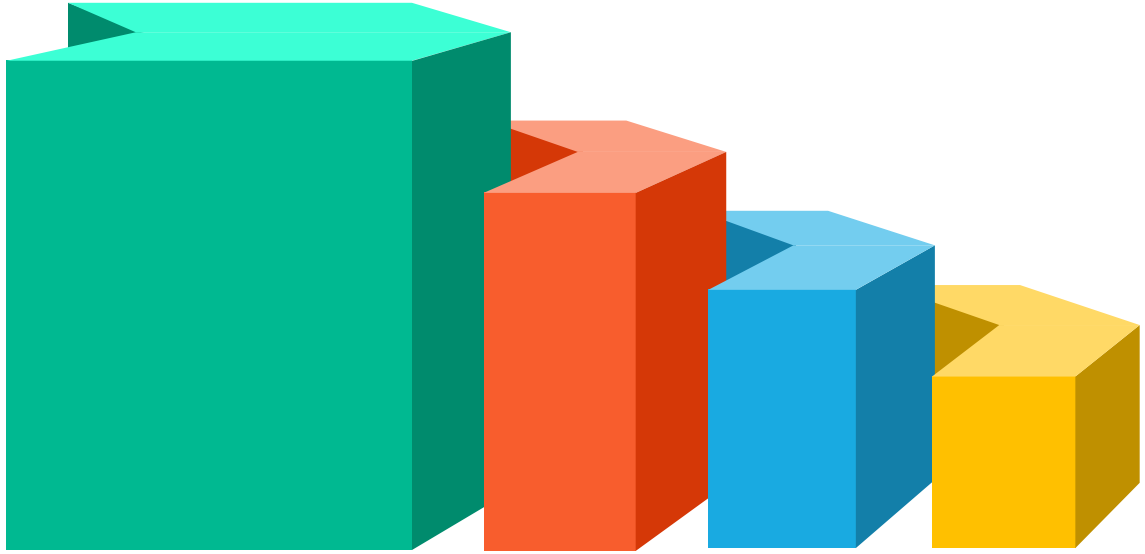


Unresponsiveness in a B2B Travel application

Process 70% of N. America oversease Leisure travel ticketing

Troubleshooting OutOfMemoryError

Unable to create new native thread

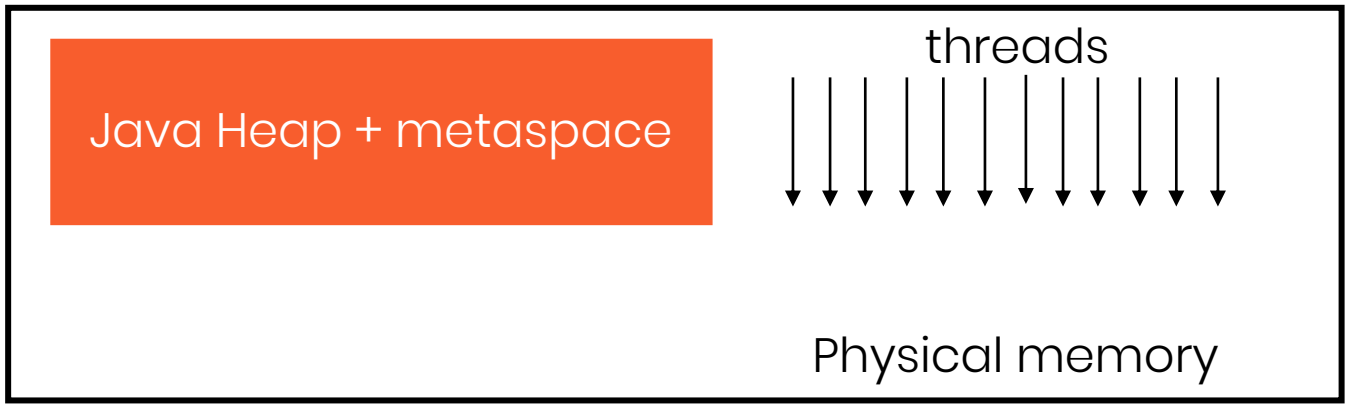


Major financial institution in N. America

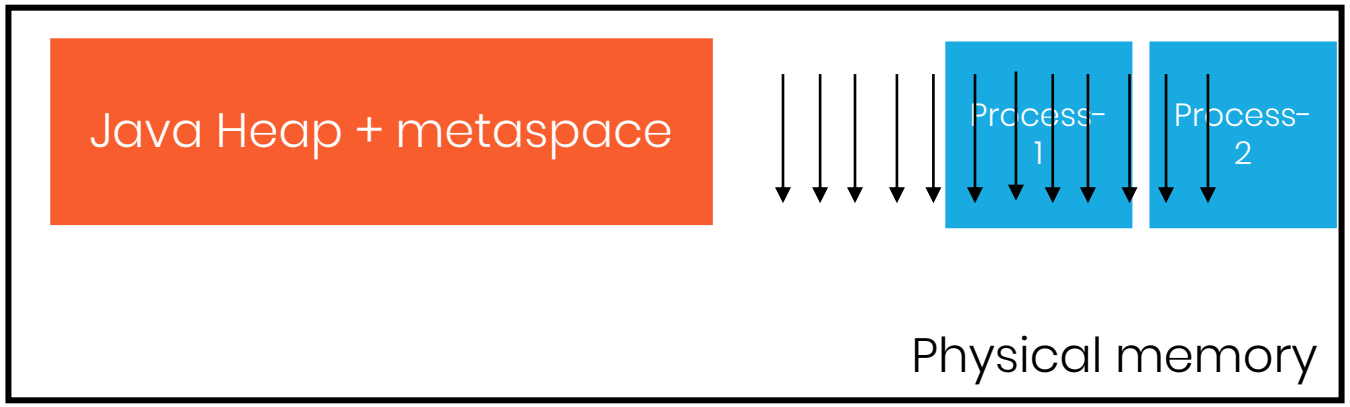
- Thread dump troubleshooting pattern: RSI

- https://map.tinyurl.com/yxho6lan

OOM: Unable to create new native thread



Key: Threads are created **outside heap, metspace**



Solution:

1. Fix thread leak
2. Increase the Thread Limits Set at Operating System(ulimit –u)
3. Reduce Java Heap Size
4. Kills other processes
5. Increase physical memory size
6. Reduce thread stack size (-Xss).

Note: can cause StackOverflowError

8 types - OutOfMemoryError

https://blog.gceasy.io/2015/09/25/outofmemoryerror-beautiful-1-page-document/

```
java.lang.OutOfMemoryError: <type>
```

01 Java heap space

02 GC overhead limit exceeded

03 Requested array size exceed VM limit

04 Permgen space

05 Metaspace

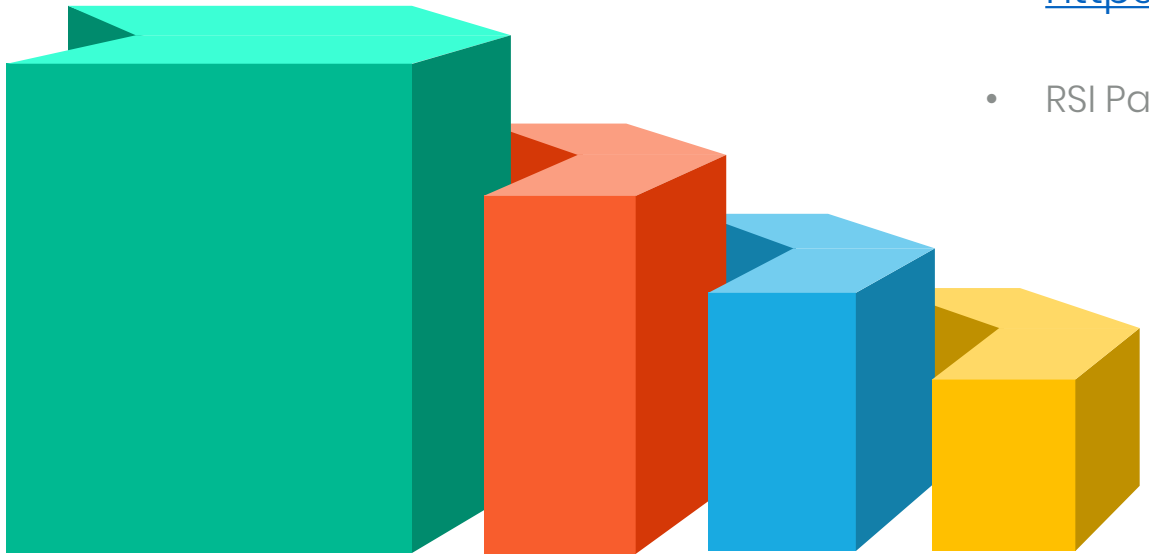
06 Unable to create new native thread

07 Kill process or sacrifice child

08 reason stack_trace_with_native method

Troubleshooting unresponsive app

- https://tinyurl.com/yywdmvyy
- RSI Pattern – Same pattern, different problem.

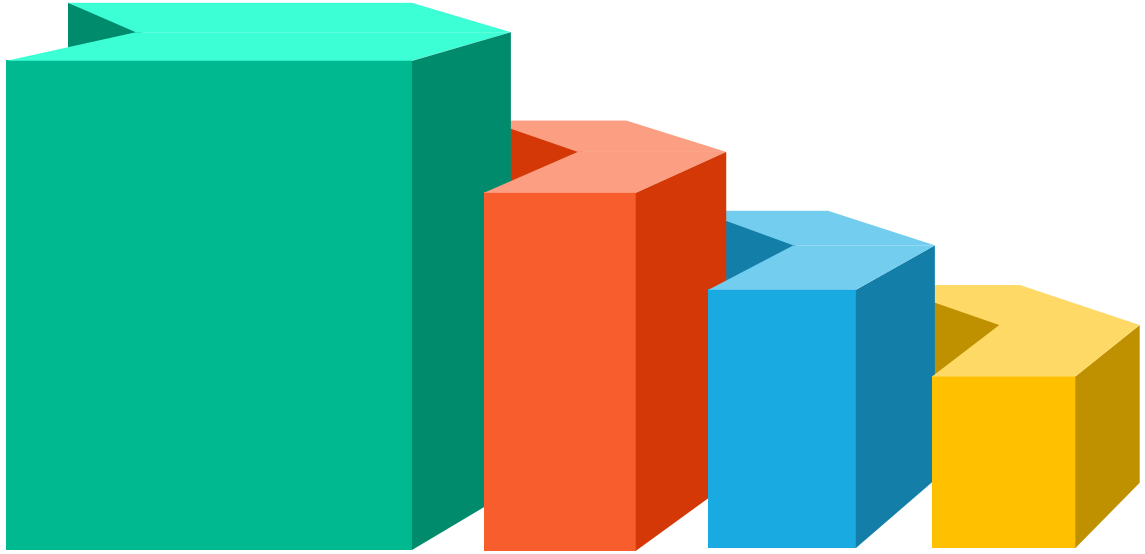


Thread dump analysis Patterns

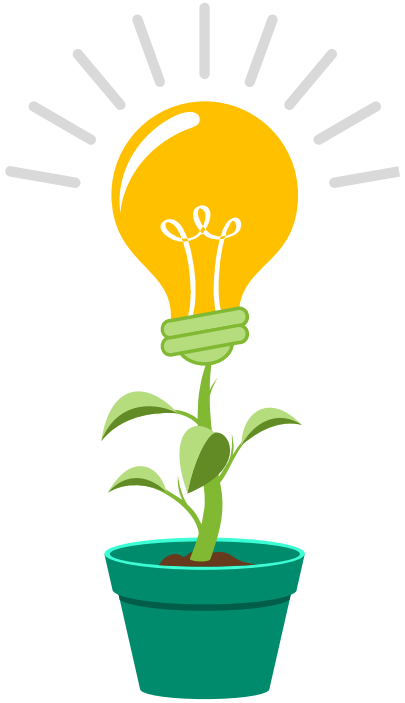
https://blog.fastthread.io/category/thread-dump-patterns/

- RSI Pattern
- Treadmill Pattern
- Leprechaun Pattern
- All Roads leads to Rome Pattern
- Traffic Jam Pattern
- Athlete Pattern
- Atherosclerosis Pattern
- Several Scavengers Pattern
- few more ...

Troubleshooting Memory problems



Enable GC Logs (always)



Till Java 8:

-XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:<file-path>

From Java 9:

-Xlog:gc*:file=<file-path>

‘Free’ GC Log analysis tools

Freely available Garbage collection log analysis tools

01

GCeasy

http://gceasy.io/

02

GC Viewer

https://github.com/chewiebug/GCViewer

03

IBM GC & Memory visualizer

https://developer.ibm.com/javasdk/tools/

04

HP Jmeter

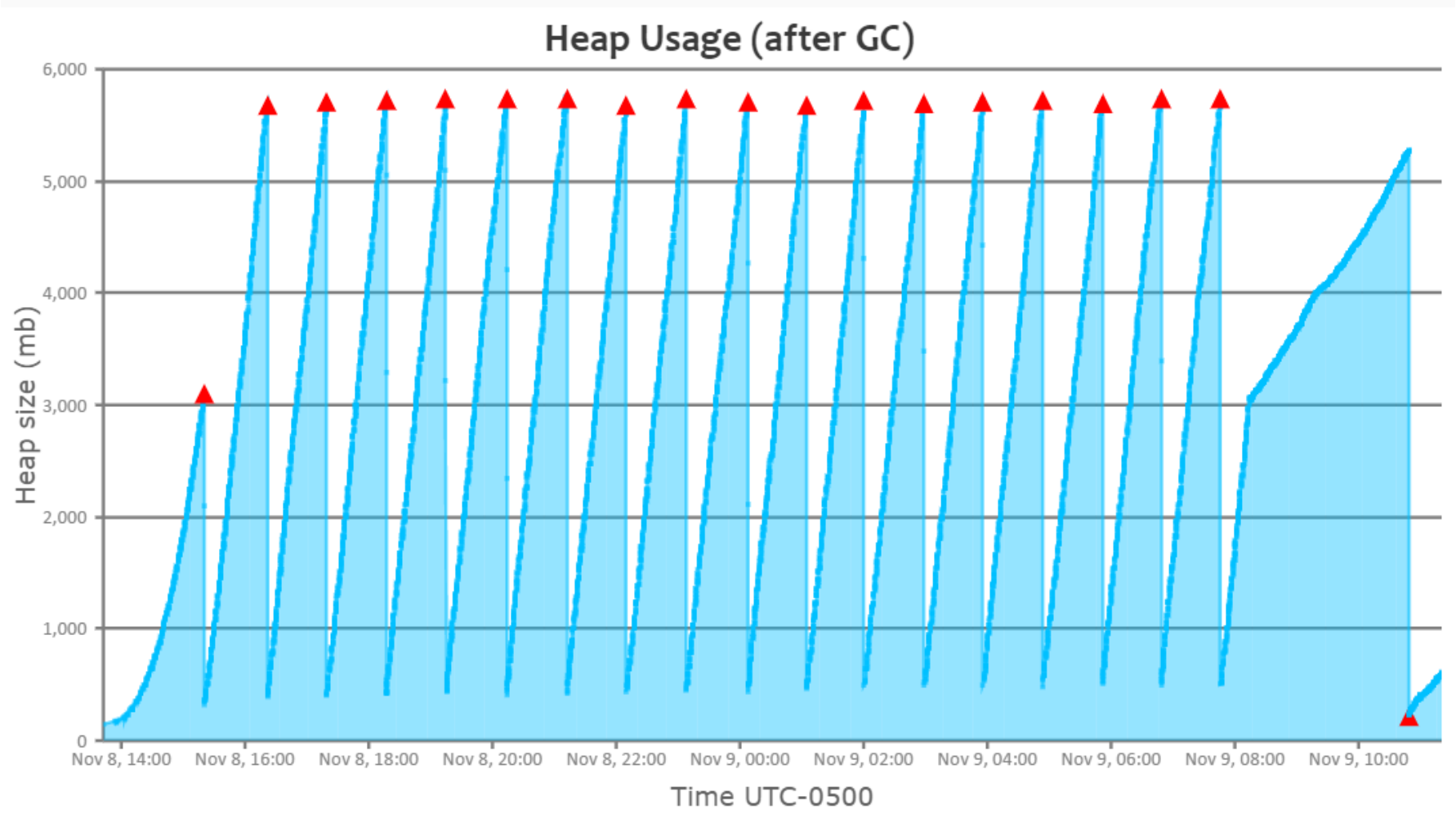
https://h20392.www2.hpe.com/portal/swdepot/displayProductInfo.do?productNumber=HPJMETER

05

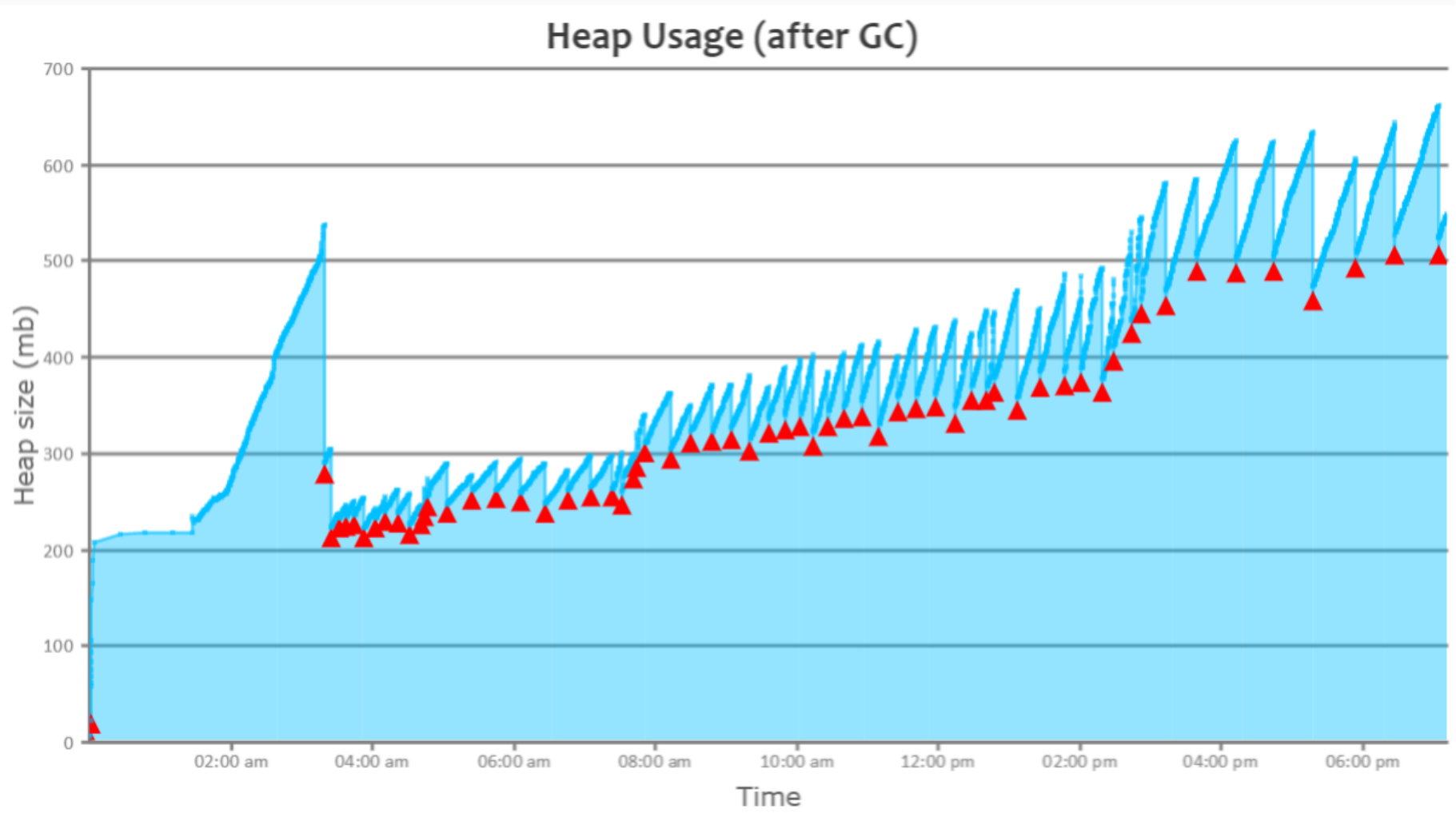
Google Garbage cat (cms)

https://code.google.com/archive/a/eclipselabs.org/p/garbagecat

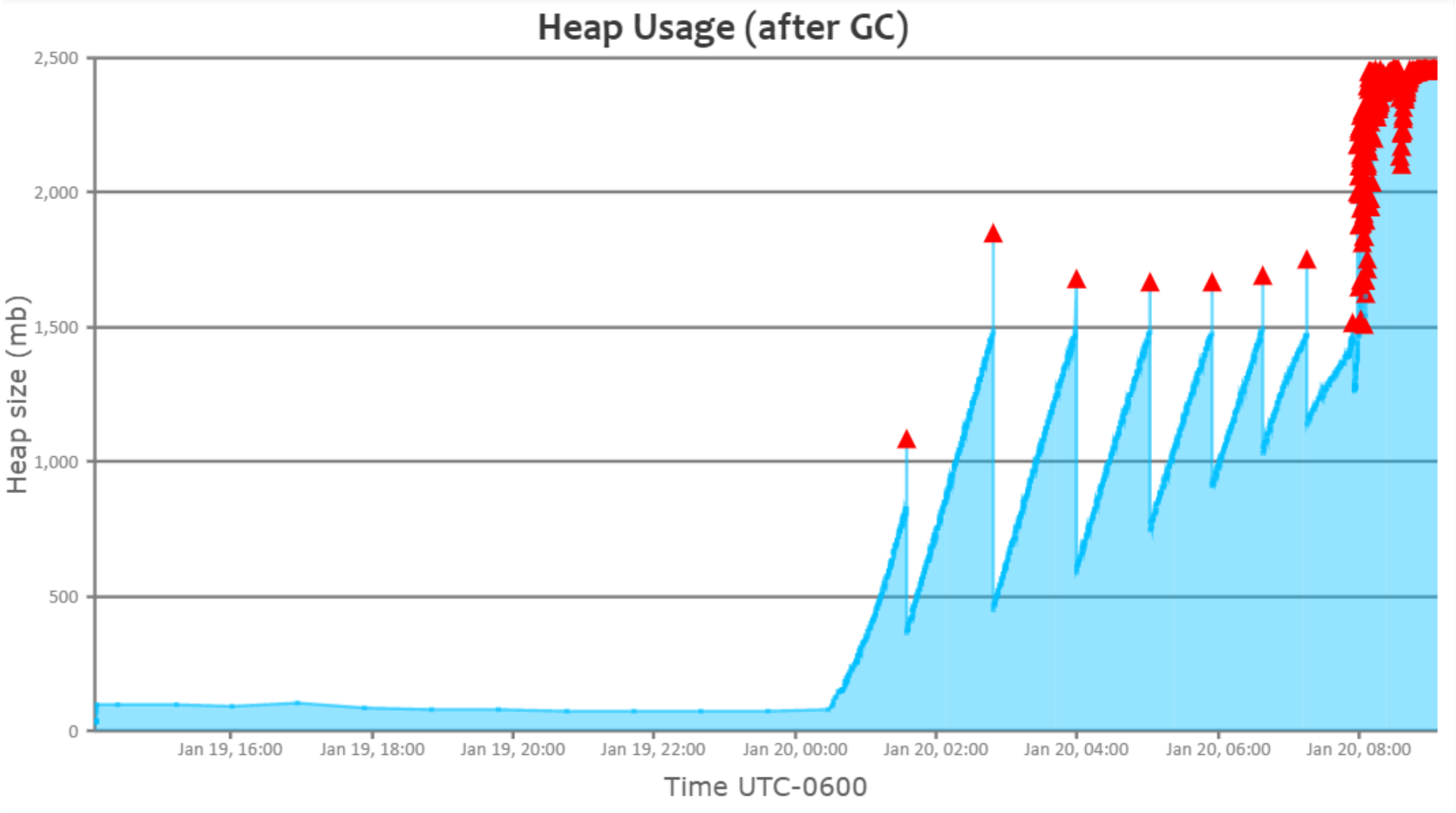
Heap usage graph



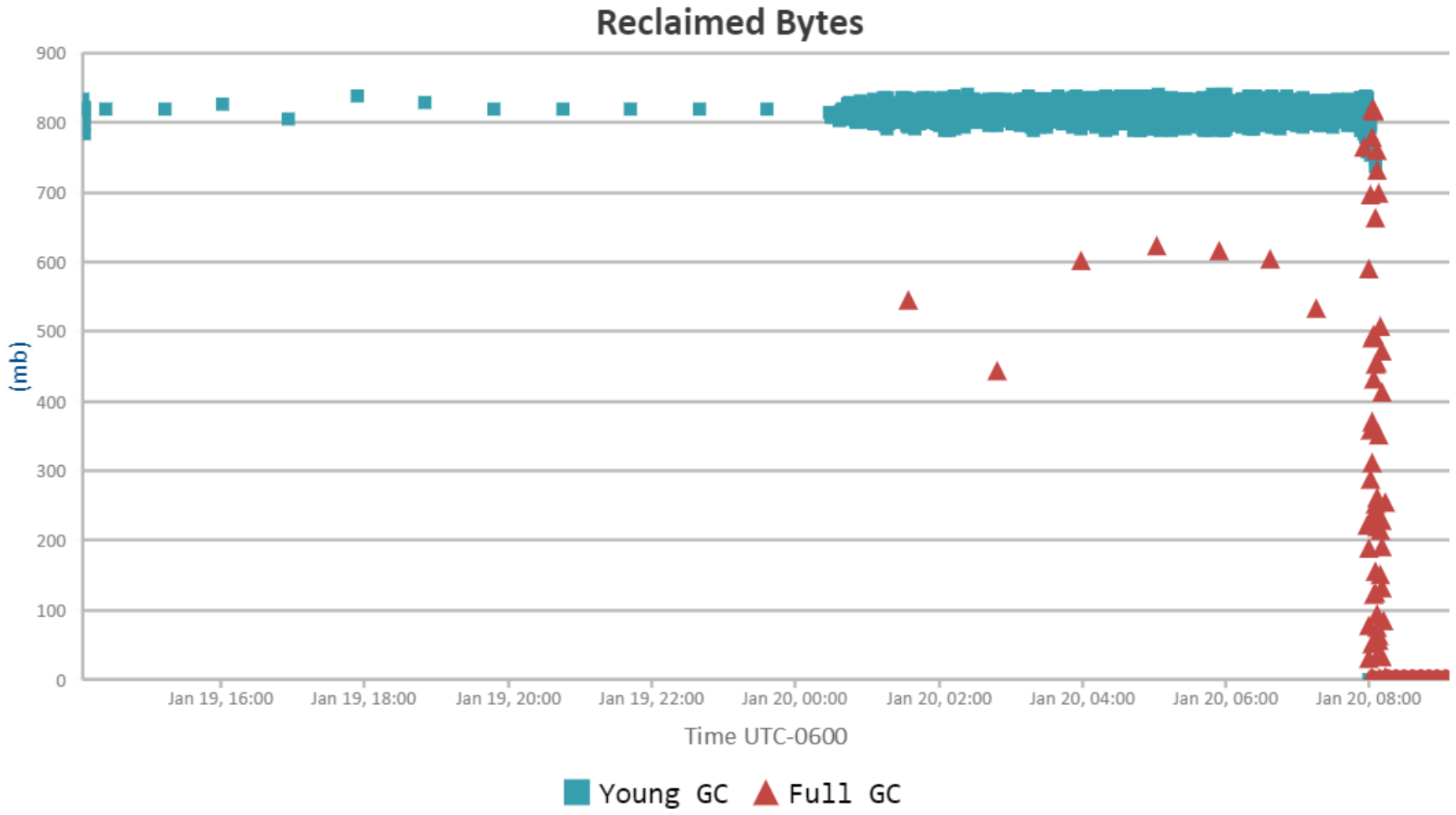
What is your observation?



Memory Problem



Corresponding – Reclaimed bytes chart



How to diagnose memory leak?

Capture heap dumps

jmap -dump:live,file=<file-path> <pid>
Example: jmap -dump:live,file=/opt/tmp/AddressBook-heapdump.bin 37320

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/logs/heapdump

Eclipse MAT, HeapHero

Two good tools to analyze memory leaks

Capture heap dumps

https://blog.fastthread.io/2016/06/06/how-to-take-thread-dumps-7-options/

01

jmap (since Java 5)

jmap -dump:live,file=<file-path> <pid>

02 ⭐

HeapDumpOnOutOfMemoryError

-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=<file-path>

03

jVisualVM

JDK tool. Now Open source. GUI based option.

04

IBM administrative console

If you are using WAS, this option can be used

05

ThreadMXBean

Programmatic way to capture thread dumps

06

APM Tools

Few APM Tools does provide this support

07 ⭐

Jcmd (since Java 7)

jcmd <pid> GC.heap_dump <file-path>

Micro-metrics

https://blog.gceasy.io/2019/03/13/micrometrics-to-forecast-application-performance/

Macro-Metrics

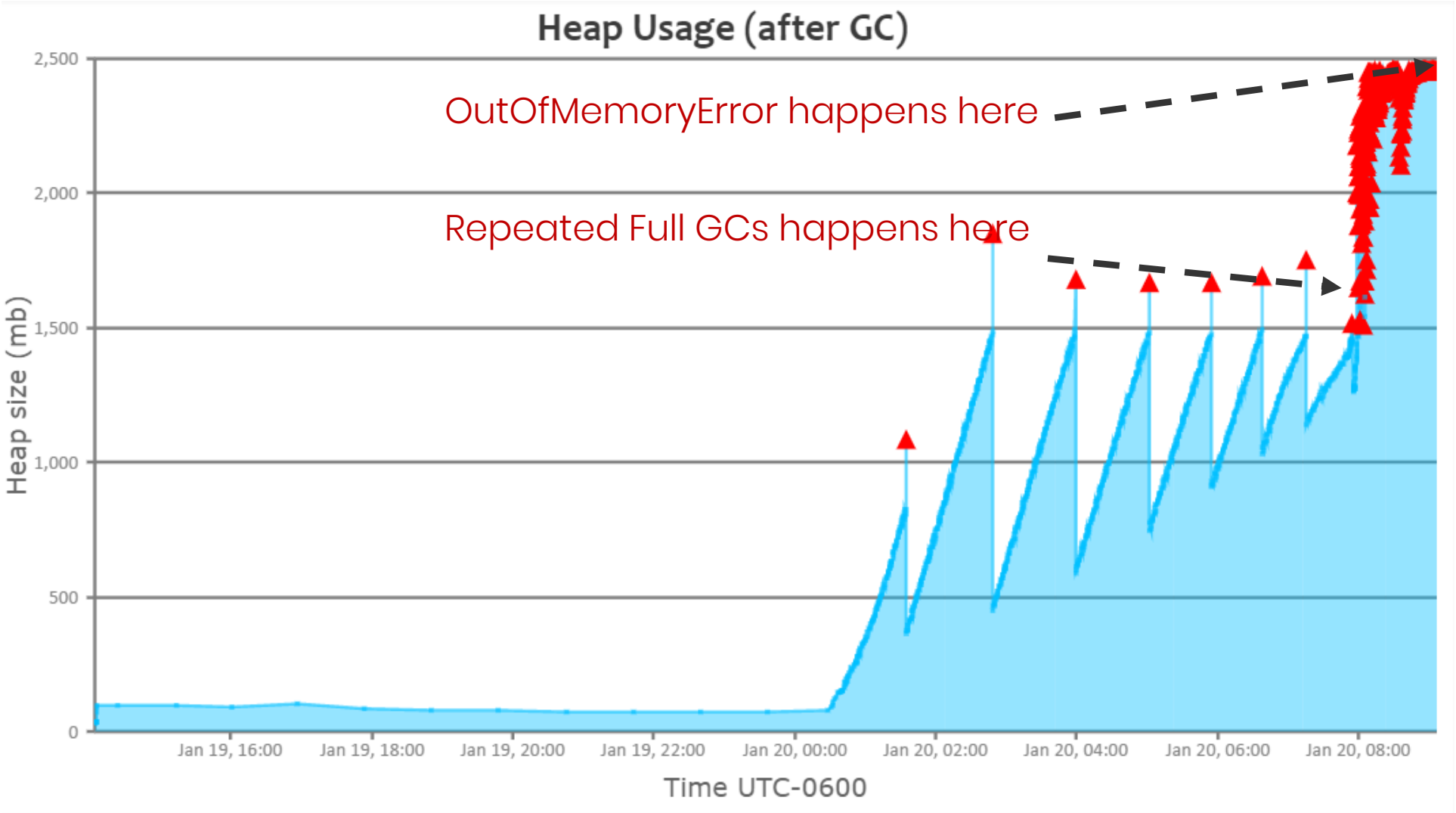
Can't forecast scalability, availability, performance problems

CPU

MEMORY

RESPONSE TIME

Micro-metrics: Early Indicators



What are Micrometrics?

https://blog.gceasy.io/2019/03/13/micrometrics-to-forecast-application-performance/

GC Throughput

Amount time application spends in processing customer transactions vs amount of time application spend in doing GC

GC Latency

If pause time starts to increase, then it's an indication that app is suffering from memory problems

Object Reclamation rate

If number of objects created in unit time

File Descriptors

File descriptor is a handle to access: File, Pipe, Network Connections. If count grows it's a lead indicator that application isn't closing resources properly.

Thread States

If BLOCKED thread state count grows, it's an early indication that your application has potential to become unresponsive

Few more...

TCP/IP States, Hosts count, IOPS, ..

right data @ right time

What data to capture?

GC Log

Thread Dumps

Heap Dumps

top -H

top

netstat

vmstat

dmesg

ps

Disk Usage

IBM Script: https://map.tinyurl.com/y4gz6o7q
Captures all of the above artifacts

Thank you my friends!

Ram Lakshmanan

ram@tier1app.com

@tier1app

https://www.linkedin.com/company/gceasy