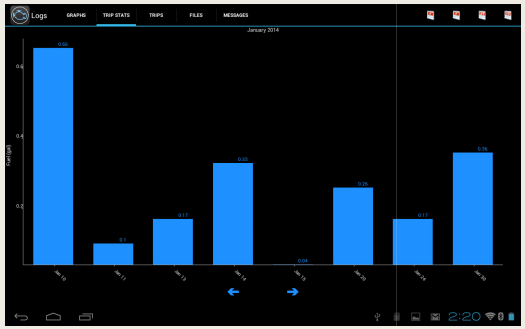
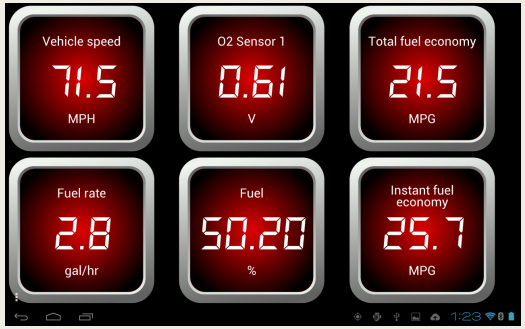
avaCro'16

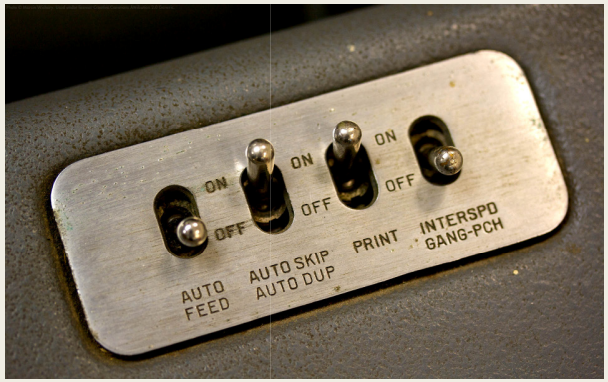




JVM PROBLEM DIAGNOSTICS

Danijel Mitar, King ICT

Aleksander Radovan, King ICT



KING

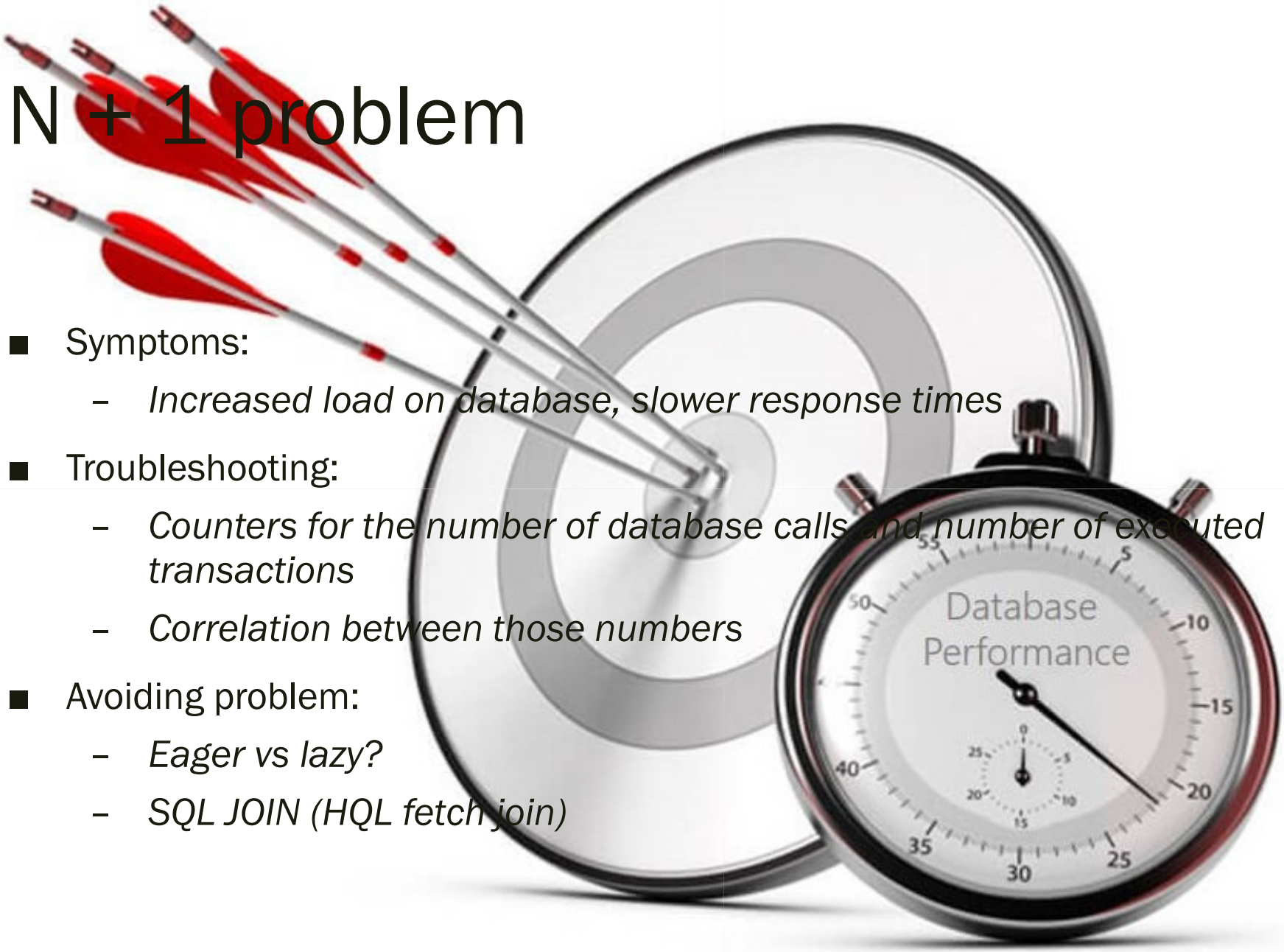
Contents

- Top performance problems
  - *Database*
  - *Memory*
- JVM monitoring
- Demo: GC algorithms & heap dump analysis
- JVM tuning flags
- Performance testing tools

Top performance problems

- Database
  - *N + 1 problem*
  - *Caching*
  - *Connection pools*
- Memory
  - *STW (Stop-The-World) garbage collections*
  - *Memory leak*

N + 1 problem

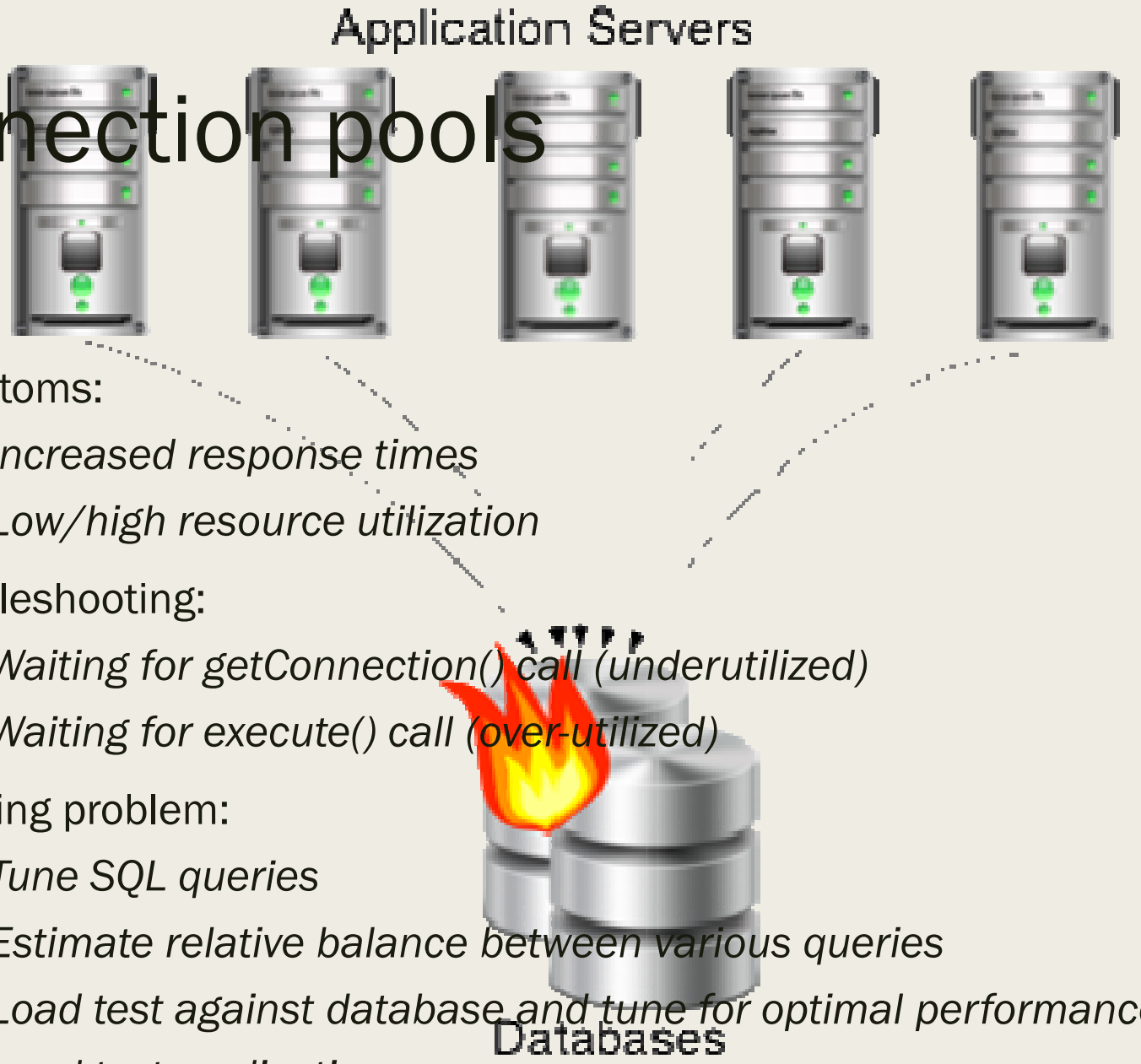


- Symptoms:
  - *Increased load on database, slower response times*
- Troubleshooting:
  - *Counters for the number of database calls and number of executed transactions*
  - *Correlation between those numbers*
- Avoiding problem:
  - *Eager vs lazy?*
  - *SQL JOIN (HQL fetch join)*

Caching

- Symptoms:
  - *Increased CPU overhead and disk I/O rate*
- Troubleshooting:
  - *Memory monitoring*
  - *Hit ratio vs miss ratio*
- Avoiding problem:
  - *Thorough planning*
  - *Proper cache configuration*
  - *Eventual consistency*

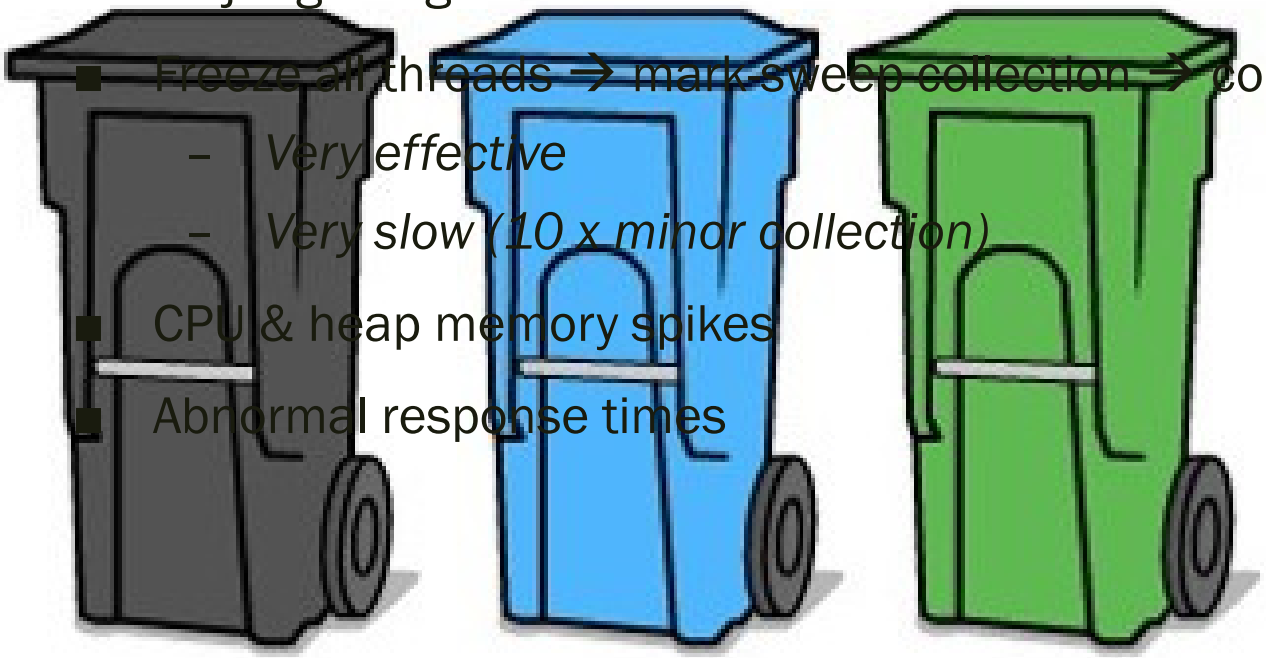
Connection pools

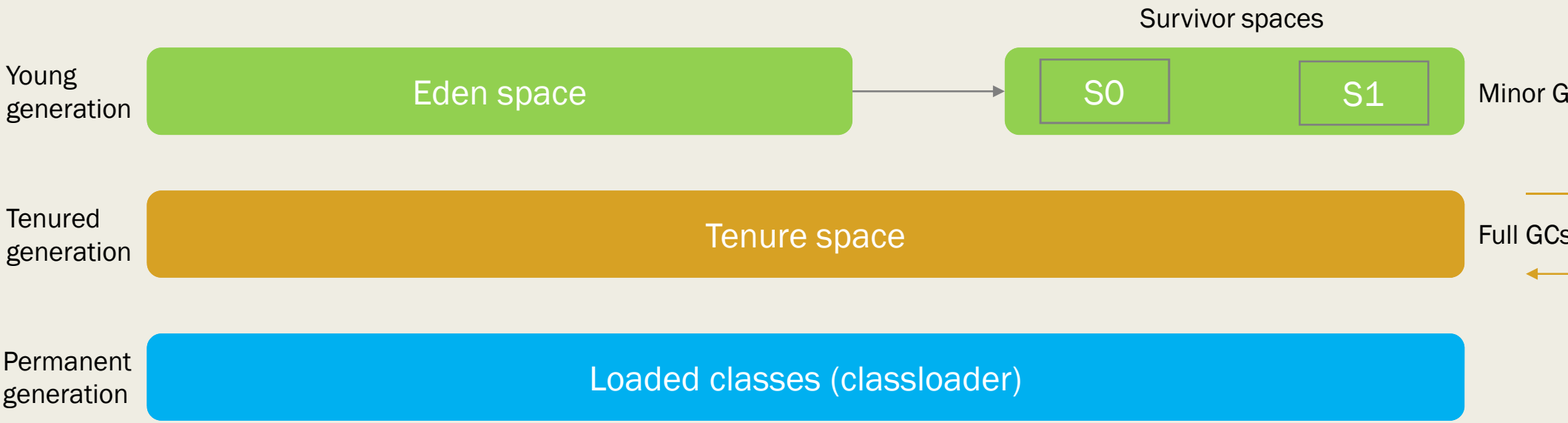


- Symptoms:
  - *Increased response times*
  - *Low/high resource utilization*
- Troubleshooting:
  - *Waiting for getConnection() call (underutilized)*
  - *Waiting for execute() call (over-utilized)*
- Avoiding problem:
  - *Tune SQL queries*
  - *Estimate relative balance between various queries*
  - *Load test against database and tune for optimal performance*
  - *Load test application*

Stop-The-World garbage collections

- Major garbage collection
- Freeze all threads → mark-sweep collection → compaction
  - *Very effective*
  - *Very slow (10 x minor collection)*
- CPU & heap memory spikes
- Abnormal response times



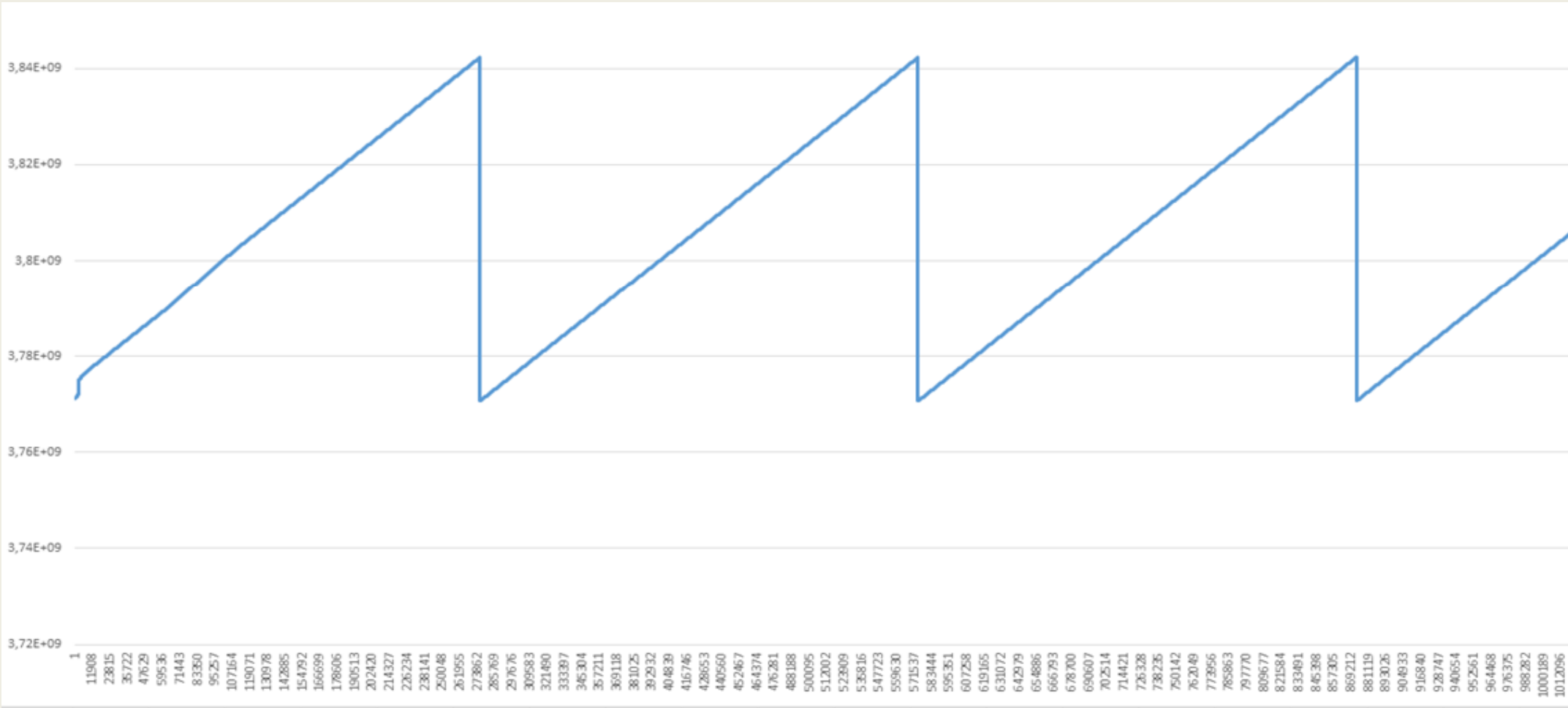

Survivor spaces
Young generation
Eden space
S0
S1
Minor G
Tenured generation
Tenure space
Full GCs
Permanent generation
Loaded classes (classloader)


Uptime: 26 hrs 28 min 41 sec
Perform GC
Heap Dump
CPU
CPU usage: 70,7%
GC activity: 24,6%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
12:50
13:00
13:10
13:20
13:30
13:40
13:50
14:00
CPU usage
GC activity
Heap
PermGen
Size: 5.849.612.288 B
Used: 5.160.146.096 B
Max: 6.442.450.944 B
5 GB
4 GB
3 GB
2 GB
1 GB
0 GB
Heap size
Used heap
Classes
Total loaded: 1.301.936
Shared loaded: 0
Total unloaded: 1.053
Shared unloaded: 0
1.200.000
1.000.000
800.000
600.000
400.000
200.000
0
Total loaded classes
Shared loaded classes
Threads
Live: 128
Daemon: 110
Live peak: 197
Total started: 154.964
140
120
100
80
60
40
20
0
Live threads
Daemon threads

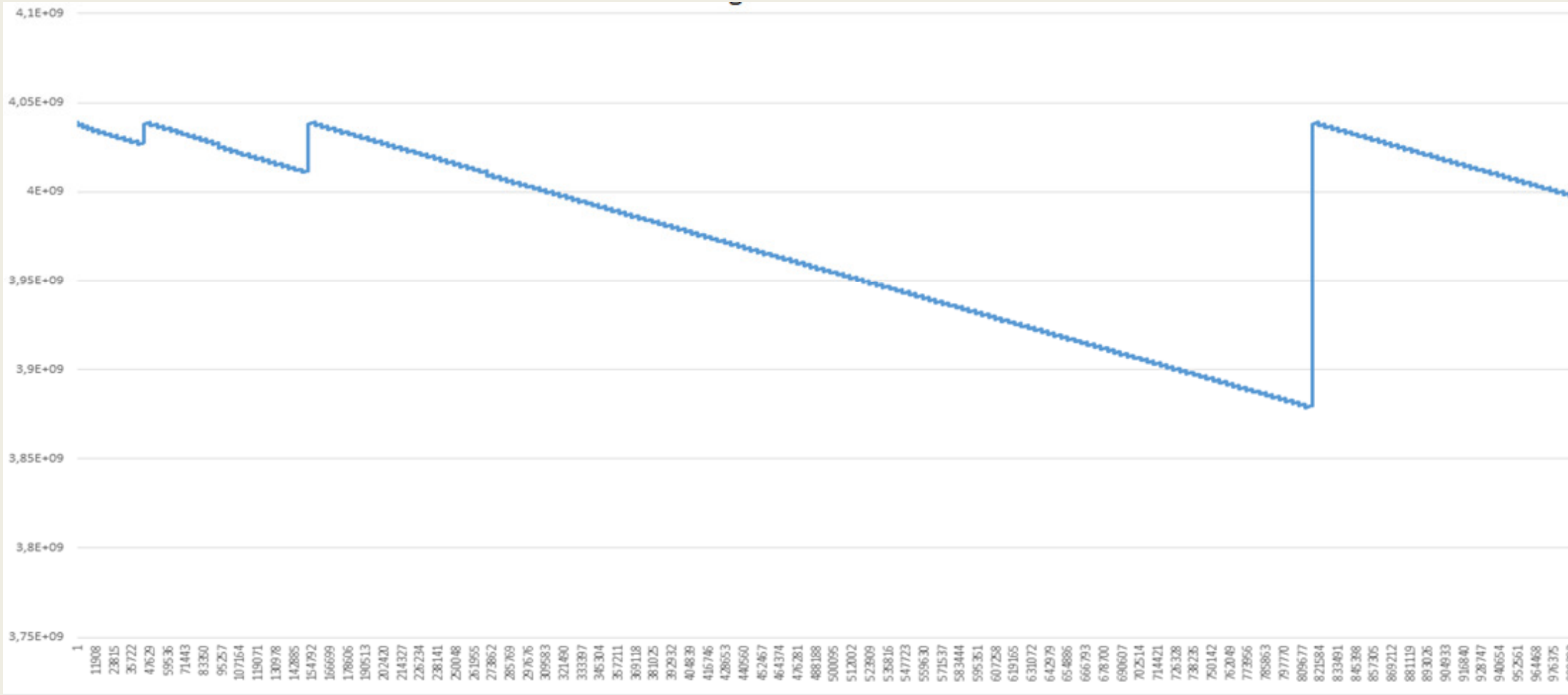
Die young or live forever.

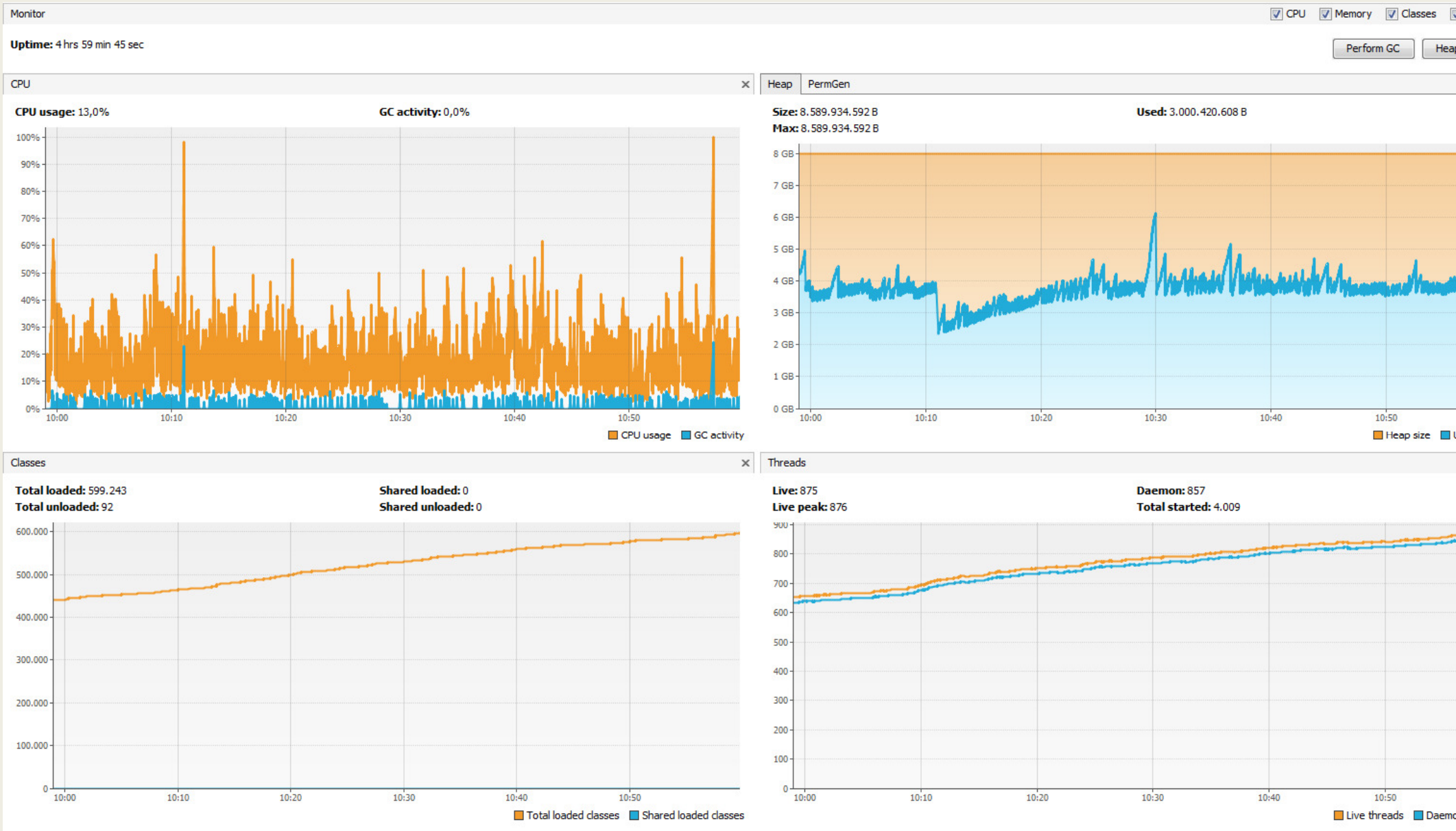
Turtle theory, Young generation theory of garbage collection

GC Algorithms – Serial, Parallel & CMS

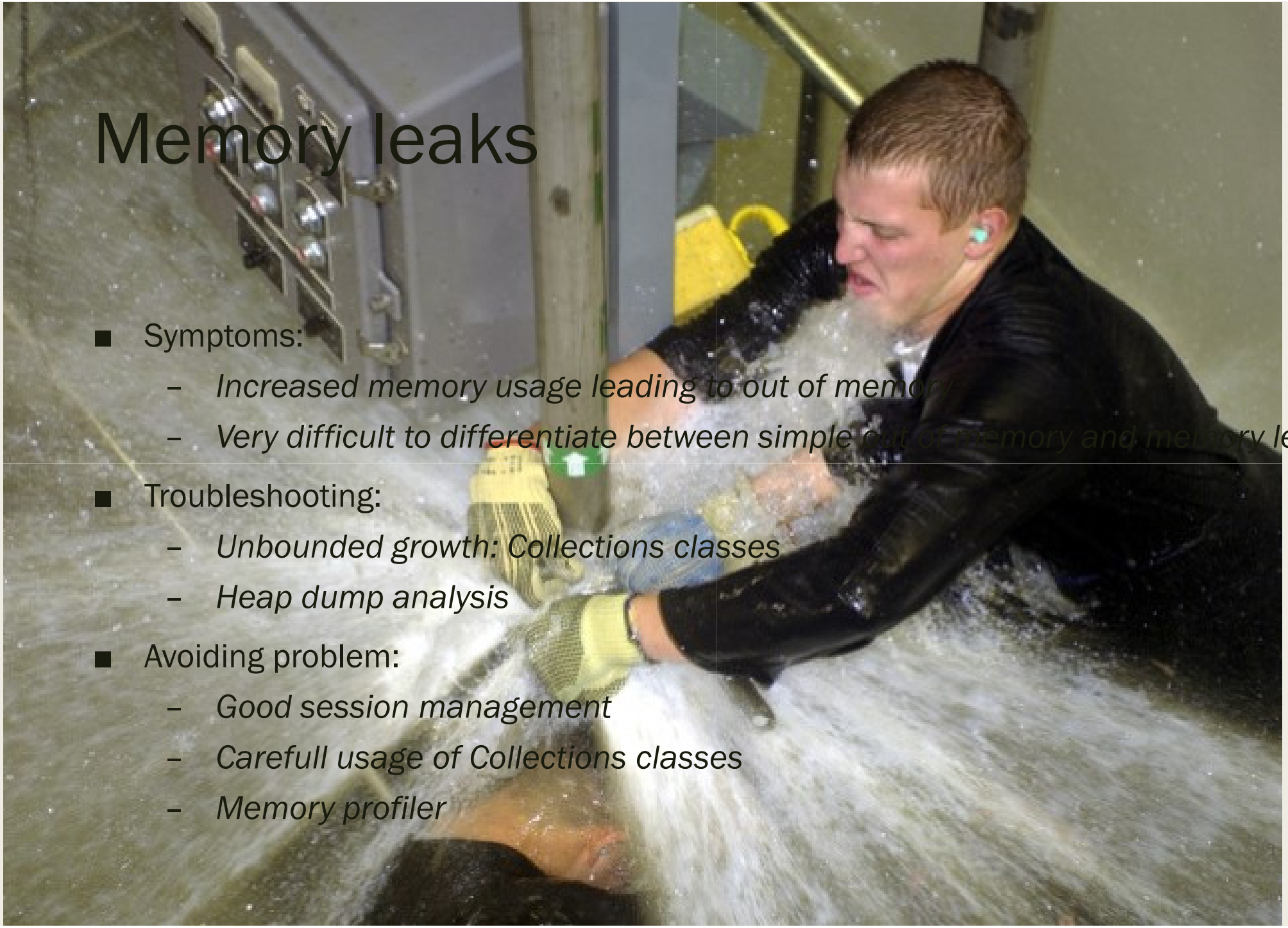


GC Algorithms – Garbage First



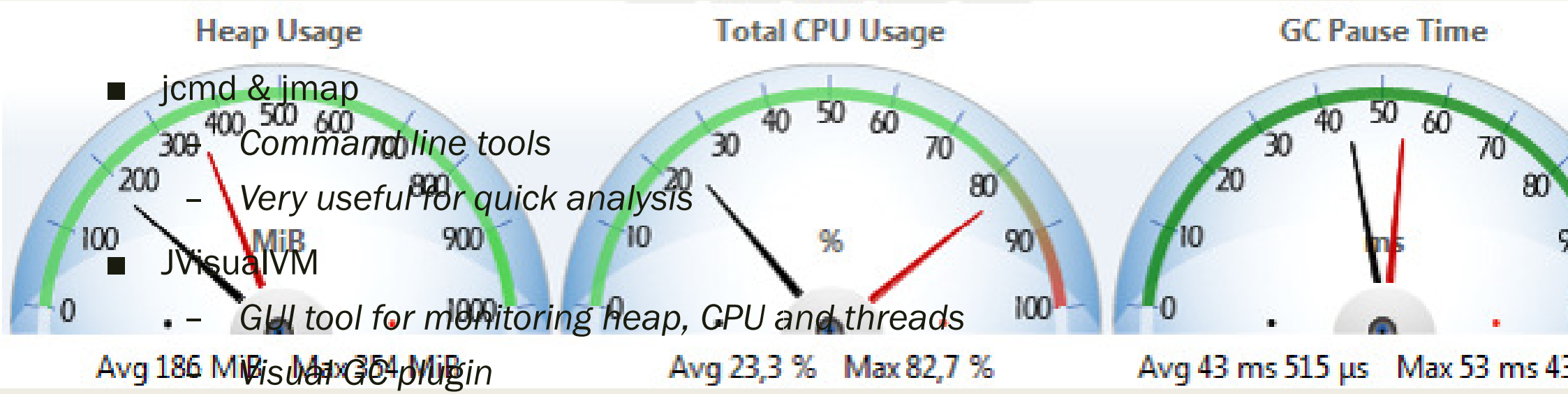

Monitor
CPU
Memory
Classes
Uptime: 4 hrs 59 min 45 sec
Perform GC
CPU
CPU usage: 13,0%
GC activity: 0,0%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
10:00
10:10
10:20
10:30
10:40
10:50
CPU usage
GC activity
Heap
PermGen
Size: 8.589.934.592 B
Max: 8.589.934.592 B
Used: 3.000.420.608 B
8 GB
7 GB
6 GB
5 GB
4 GB
3 GB
2 GB
1 GB
0 GB
Heap size
Classes
Total loaded: 599.243
Total unloaded: 92
Shared loaded: 0
Shared unloaded: 0
600.000
500.000
400.000
300.000
200.000
100.000
0
Total loaded classes
Shared loaded classes
Threads
Live: 875
Live peak: 876
Daemon: 857
Total started: 4.009
900
800
700
600
500
400
300
200
100
0
Live threads

Memory leaks



- Symptoms:
  - *Increased memory usage leading to out of memory*
  - *Very difficult to differentiate between simple out of memory and memory leak*
- Troubleshooting:
  - *Unbounded growth: Collections classes*
  - *Heap dump analysis*
- Avoiding problem:
  - *Good session management*
  - *Carefull usage of Collections classes*
  - *Memory profiler*

JVM monitoring



- jcmd & jmap
  - *Command line tools*
  - *Very useful for quick analysis*
- JVisualVM
  - *GUI tool for monitoring heap, CPU and threads*
  - *Visual GC plugin*
- Heap dump analysis
  - *jhat*
  - *MAT - Memory Analyzer Tool (Eclipse)*
  - *VisualVM launcher (IntelliJ IDEA)*


DEMO
6
MÖNCHENGLADBACH

JVM tuning flags

Flag	What it does	When to use it
-server	Chooses the server compiler.	For long-running applications that need fast performance.
-client	Chooses the client compiler.	For applications where startup is the most important factor.
-XX:+TieredCompilation	Uses tiered compilation (both client and server).	For applications where you want the best possible performance and have enough available native memory for the extra compiled code.

JVM tuning flags

Flag	What it does	When to use it
-XX:+UseSerialGC	Uses a simple, single-threaded GC algorithm.	For small (100 MB) heaps.
-XX:+UseParallelOldGC	Uses multiple threads to collect the old generation while application threads are stopped.	When your application can tolerate occasional long pauses and you want to maximize throughput while minimizing CPU usage.
-XX:+UseParallelGC	Uses multiple threads to collect the young generation while application threads are stopped.	Use in conjunction with UseParallelOldGC.

JVM tuning flags

Flag	What it does	When to use it
-XX:+UseConcMarkSweepGC	Uses background thread(s) to remove garbage from the old generation with minimal pauses.	When you have available CPU the background thread, you do want long GC pauses, and you have a relatively small heap.
-XX:+UseParNewGC	Uses multiple threads to collect the young generation while application threads are stopped.	Use in conjunction with Use ConcMarkSweepGC.
-XX:+UseG1GC	Uses multiple threads to collect the young generation while application threads are stopped, and background thread(s) to remove garbage from the old generation with minimal pauses.	When you have available CPU the background thread, you do want long GC pauses, and you not have a small heap.
In development for Java 9: Shenandoah GC		

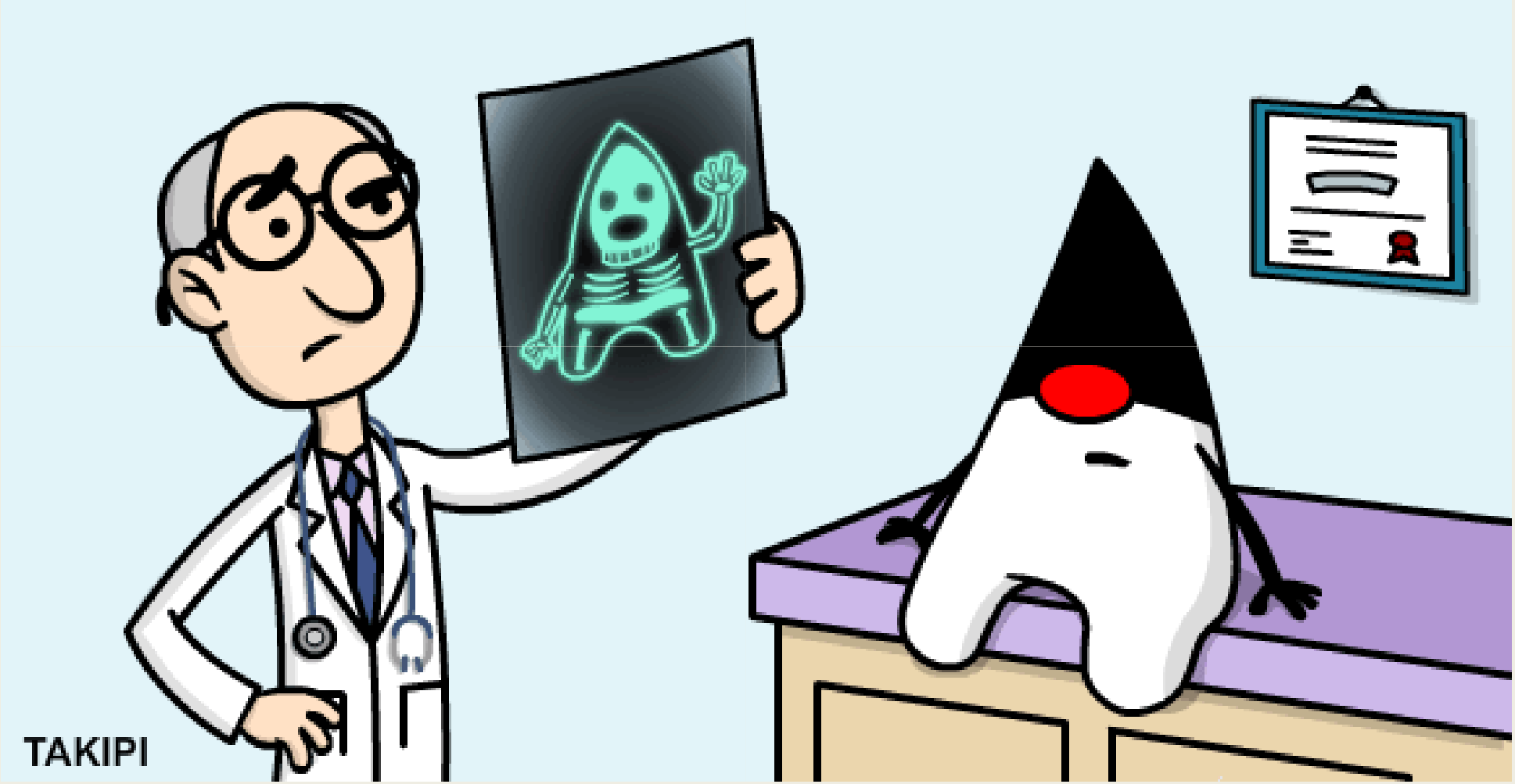
Performance testing tools

- JMeter
- Gatling
- The Grinder
- Faban

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Donald Knuth

QUESTIONS?



Resources

Oaks, S. (2014). Java Performance: The Definitive Guide. Sebastopol, Kalifornija: O'Reilly Media, Inc.

Oransa, O. (2014). Java EE 7 Performance Tuning and Optimization, Birmingham: Packt Publishing Ltd.

http://help.eclipse.org/mars/index.jsp?topic=/org.eclipse.mat.ui.help/welcome.html

http://www.vogella.com/tutorials/EclipseMemoryAnalyzer/article.html

http://infoq.com/articles/Diagnosing-Common-Java-Database-Performance-Hotspots

https://www.pluralsight.com/courses/understanding-java-vm-memory-management

http://www.zabbix.com/documentation.php

http://blog.king-ict.hr/dmitar