Typical Issues with Middleware

HrOUG 2016

Timur Akhmadeev

October 2016

Pythian

love your data®

About Me

Dev Perf DBA

Database Consultant at Pythian
10+ years with Database and Java
Systems Performance and Architecture
OakTable member
3rd conference as a speaker
timurakhmadeev.wordpress.com
pythian.com/blog/author/akhmadeev
twitter.com/tmmdv
timur.akhmadeev@gmail.com

About Pythian

10K+

Systems

400+

People in 200 cities in 35 countries

Founded in

1997

Global Leader In IT Transformation And Operational Excellence

Unparalleled Expertise

- Top 5% in Databases, Applications, Infrastructure, Big Data, Cloud, Data Science, and DevOps

Unmatched Certifications

- 9 Oracle ACEs, 4 Oracle ACE Directors, 1 Oracle ACE Associate
- 6 Microsoft MVPs, 1 Microsoft Certified Master
- 5 Google Platform Qualified Developers
- 1 Cloudera Champion of Big Data
- 1 Mongo DB Certified DBA Associate Level
- 1 DataStax Certified Partner, 1 MVP

Broad Technical Experience

- Oracle, Microsoft, MySQL, Oracle EBS, Hadoop, Cassandra, MongoDB, virtualization, configuration management, monitoring, trending, and more

Agenda

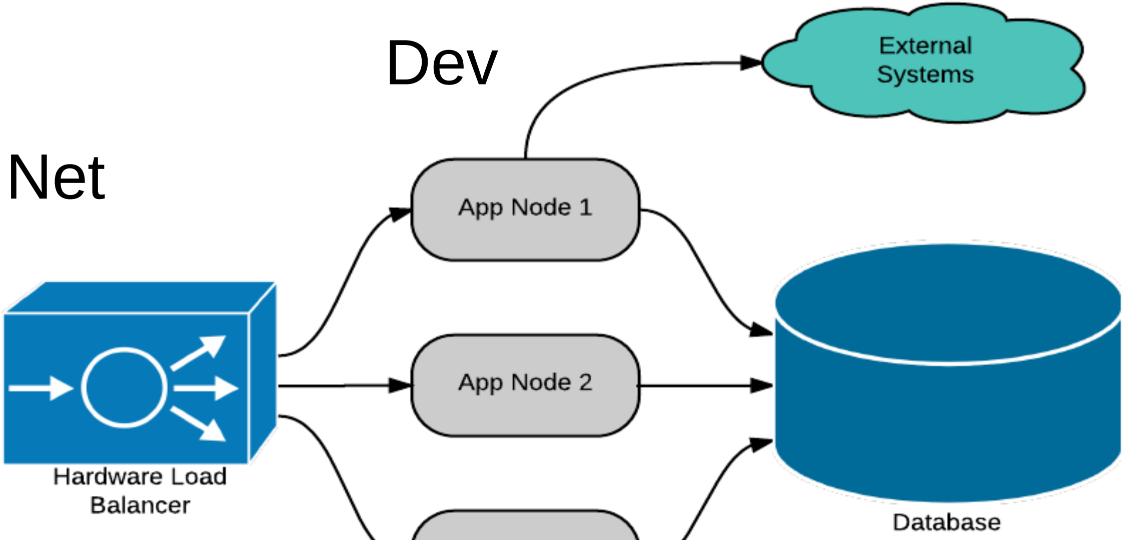
- Background
- Architecture
- Typical Issues
- Approach to Troubleshooting

Architecture

NOC

Net

Dev



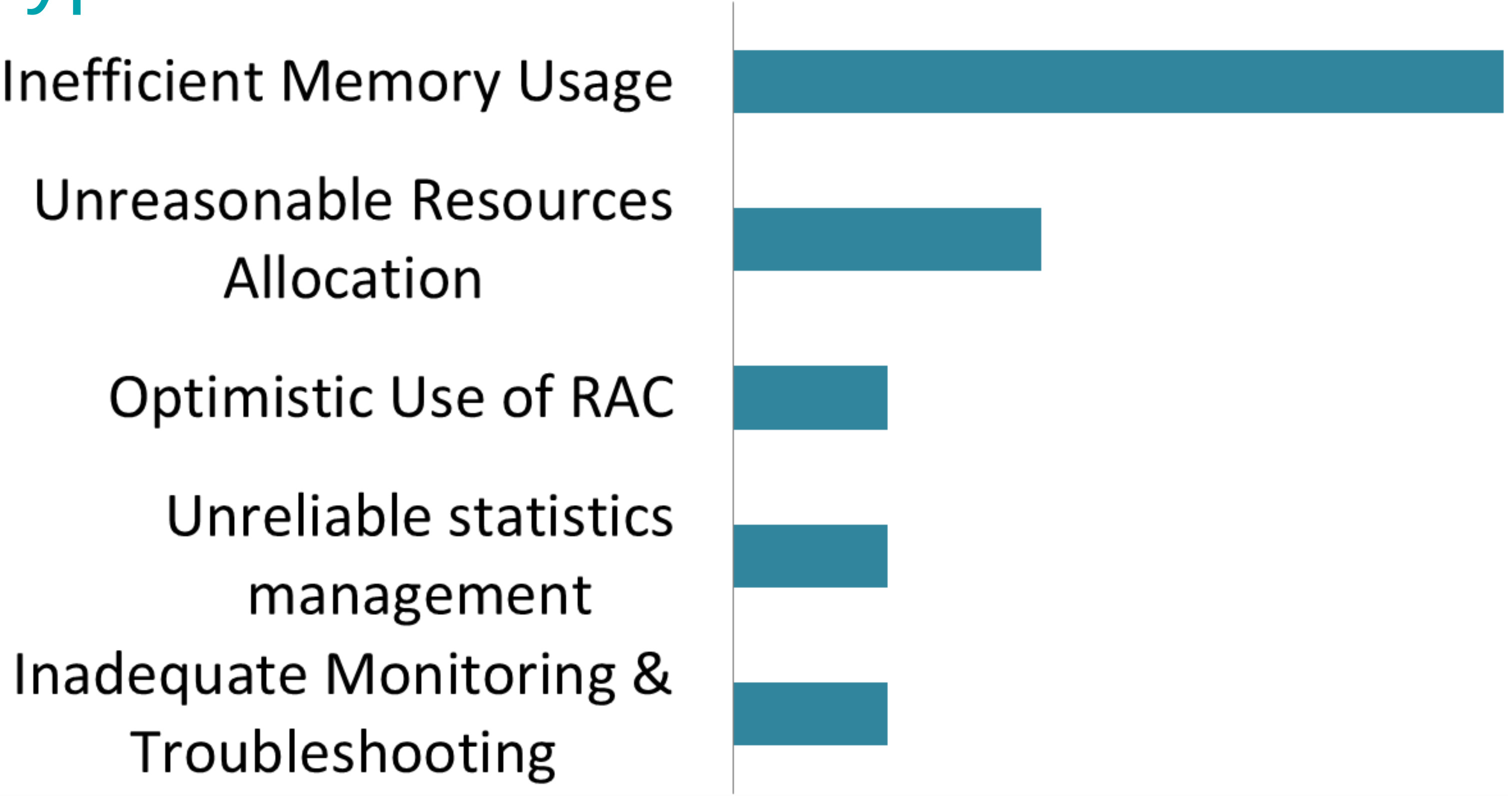
DBA

Storage

Typical MW Issues

- Failures
  - Out of Memory, Crashes
- Stability
  - Hangs, changes in response times

Typical Reasons of Issues



Inefficient Memory Usage

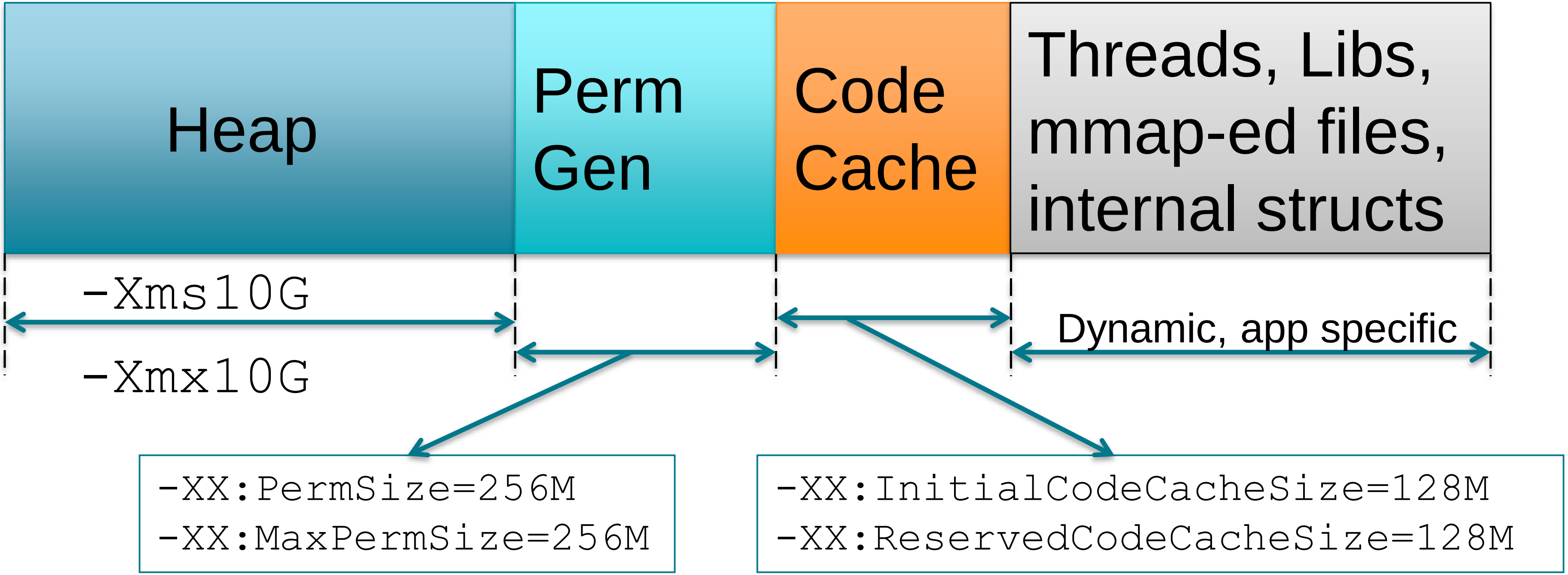
OS Memory Usage – Database

- Still very common to miss HugePages
- HugePages are a must
  - Lock SGA in memory
  - Reduce OS page tables footprint
  - Reduce sys% CPU time
- THP have to be disabled

OS Memory Usage – Middleware

- Possible to use HugePages with Java
- Recommended by Oracle
  - Oracle Commerce MAA Configuration Best Practices, July 2015
- Recommended by VMware
  - Large Pages Performance – case study
- Recommended to disable THP

Pre-Java 8 Memory Layout



Heap

Perm Gen

Code Cache

Threads, Libs, mmap-ed files, internal structs

-Xms10G

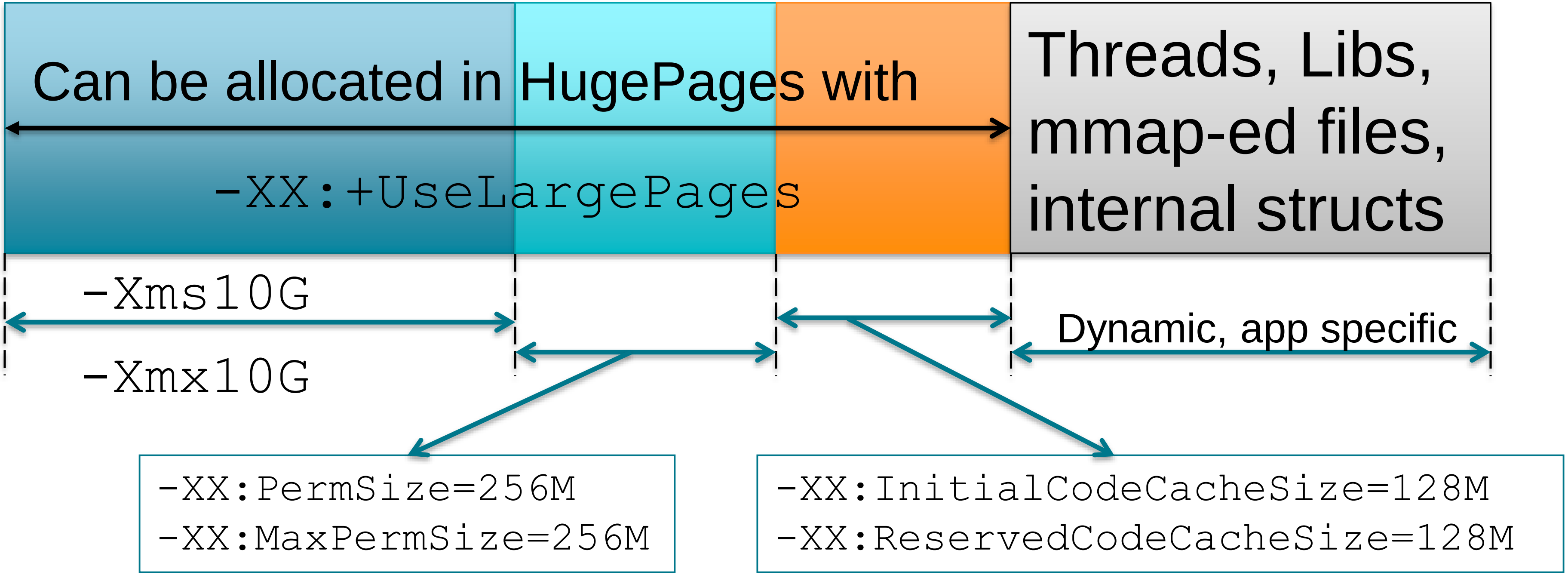
-Xmx10G

Dynamic, app specific

```
-XX:PermSize=256M
-XX:MaxPermSize=256M
```

```
-XX:InitialCodeCacheSize=128M
-XX:ReservedCodeCacheSize=128M
```

Pre-Java 8 Memory Layout



Can be allocated in HugePages with

-XX:+UseLargePages

Threads, Libs, mmap-ed files, internal structs

-Xms10G

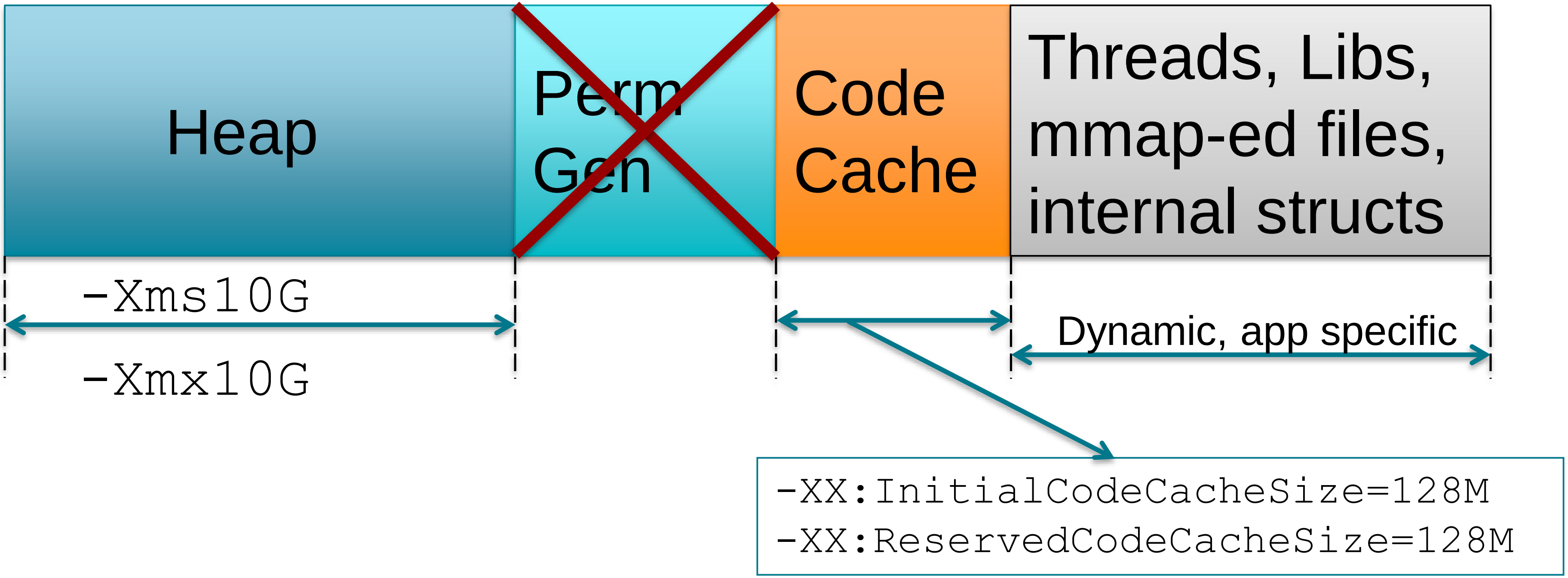
-Xmx10G

Dynamic, app specific

```
-XX:PermSize=256M
-XX:MaxPermSize=256M
```

```
-XX:InitialCodeCacheSize=128M
-XX:ReservedCodeCacheSize=128M
```

Java 8 Memory Layout



Heap

~~Perm Gen~~

Code Cache

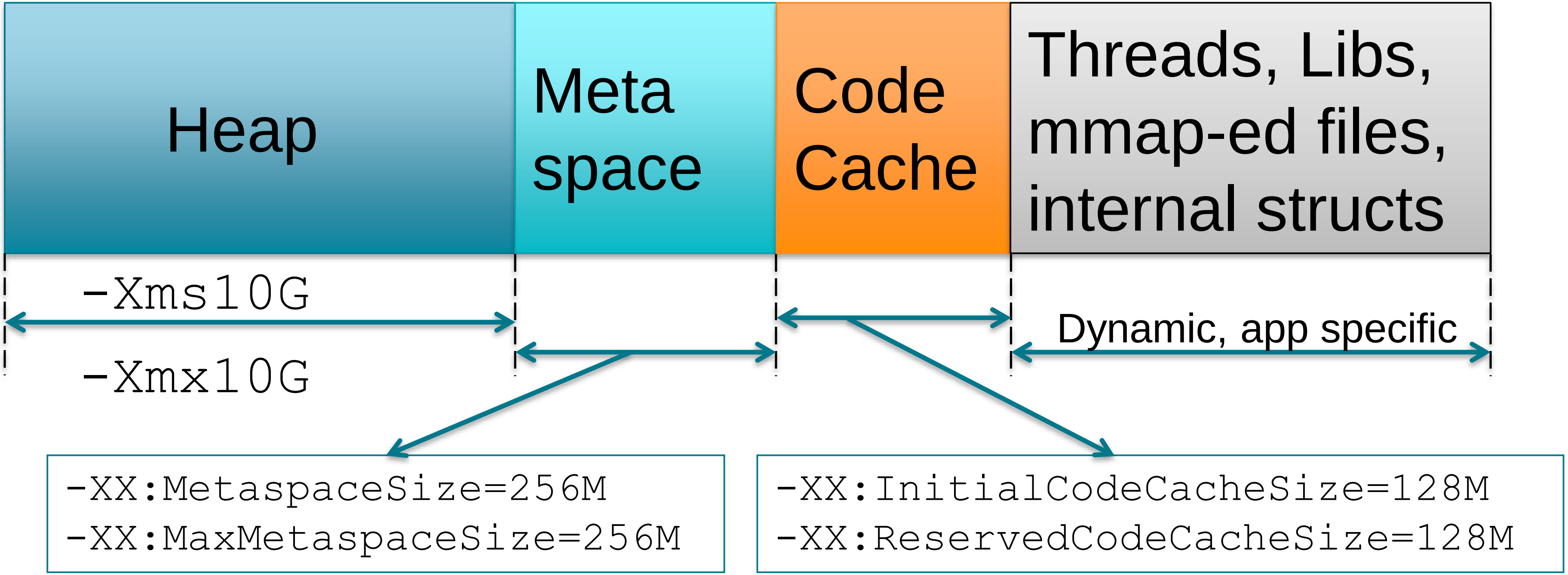
Threads, Libs, mmap-ed files, internal structs

```
-Xms10G
-Xmx10G
```

Dynamic, app specific

```
-XX:InitialCodeCacheSize=128M
-XX:ReservedCodeCacheSize=128M
```

Java 8 Memory Layout



Heap

Meta space

Code Cache

Threads, Libs, mmap-ed files, internal structs

-Xms10G

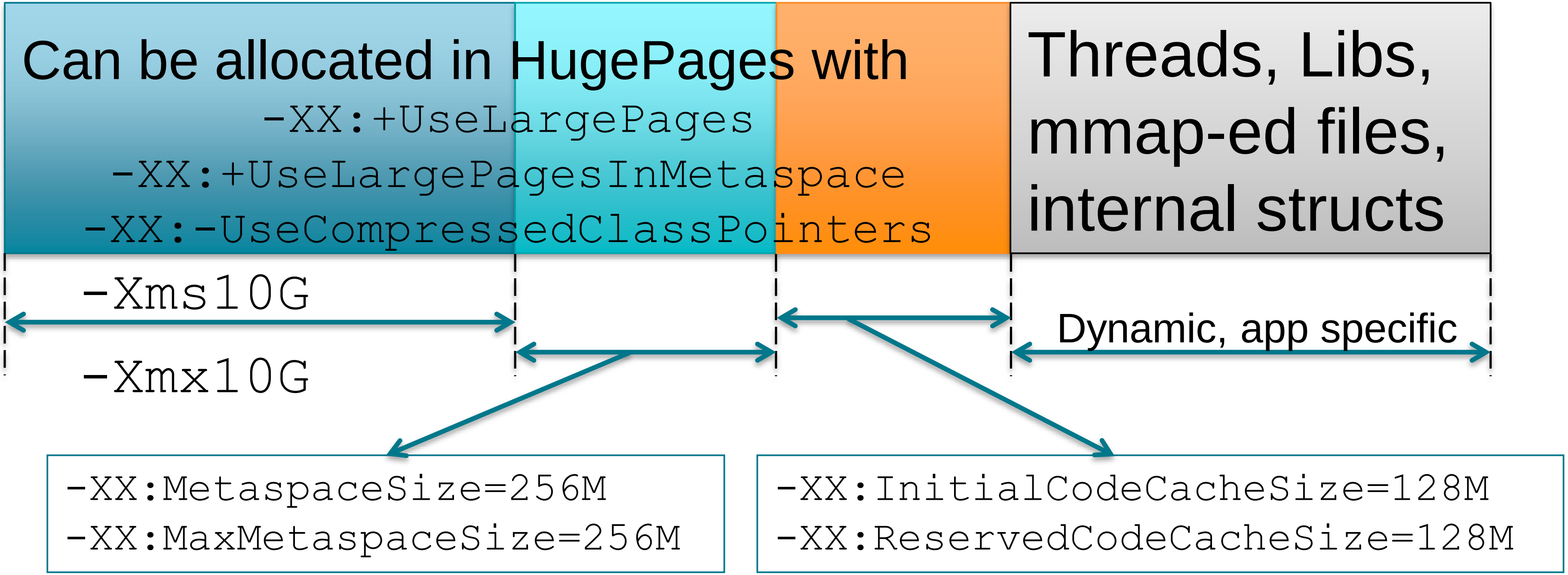
-Xmx10G

Dynamic, app specific

```
-XX:MetaspaceSize=256M
-XX:MaxMetaspaceSize=256M
```

```
-XX:InitialCodeCacheSize=128M
-XX:ReservedCodeCacheSize=128M
```

Java 8 Memory Layout



Can be allocated in HugePages with

```
-XX:+UseLargePages
-XX:+UseLargePagesInMetaspace
-XX:-UseCompressedClassPointers
```

Threads, Libs, mmap-ed files, internal structs

```
-Xms10G
-Xmx10G
```

Dynamic, app specific

```
-XX:MetaspaceSize=256M
-XX:MaxMetaspaceSize=256M
```

```
-XX:InitialCodeCacheSize=128M
-XX:ReservedCodeCacheSize=128M
```

Java 8 Memory and HugePages

```
-XX:+UseLargePages
```

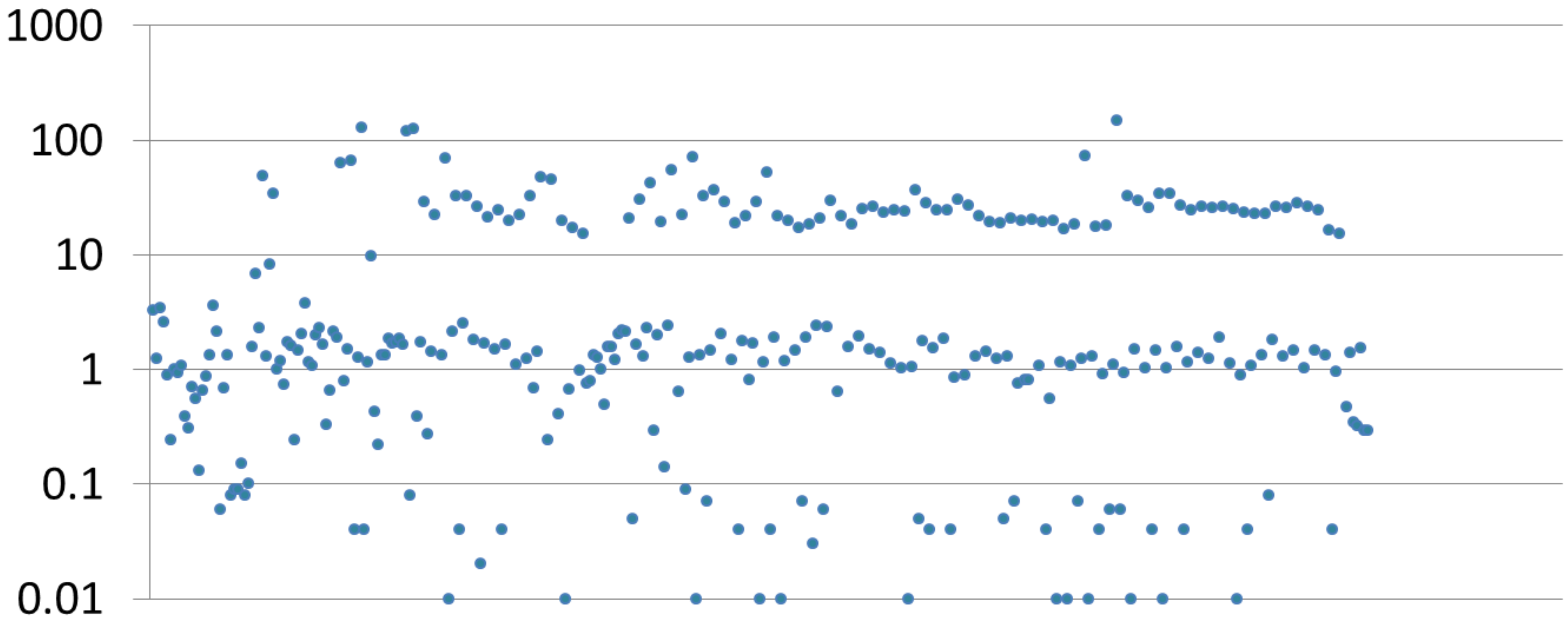
- If not enough pages, default pages are used
- For Metaspace in HugePages as well:

```
-XX:+UseLargePagesInMetaspace
-XX:-UseCompressedClassPointers
```

Memory Usage – Java Heap

- Application creates objects in heap
- Heap is cleaned up automatically
- Cleaning is called Garbage Collection
- Major cause of pause time with Java based apps

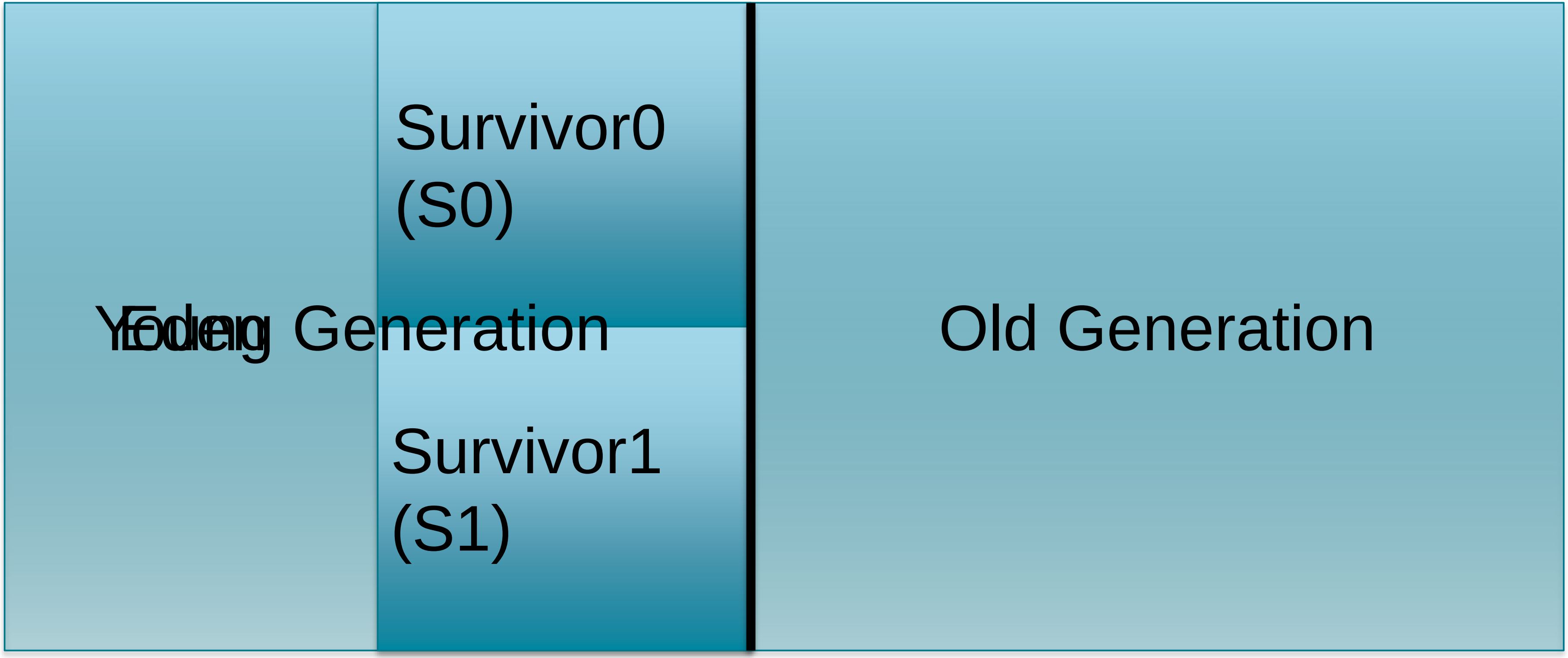
GC Times, sec



Memory Usage – Java Heap

- Long GC pause is a result of
  - poor sizing and configuration
  - insufficient heap
  - memory leak
  - application deficiencies
- May lead to java.lang.OutOfMemoryError
- More OOM https://plumbr.eu/outofmemoryerror

Memory Usage – Java Heap



Young Generation

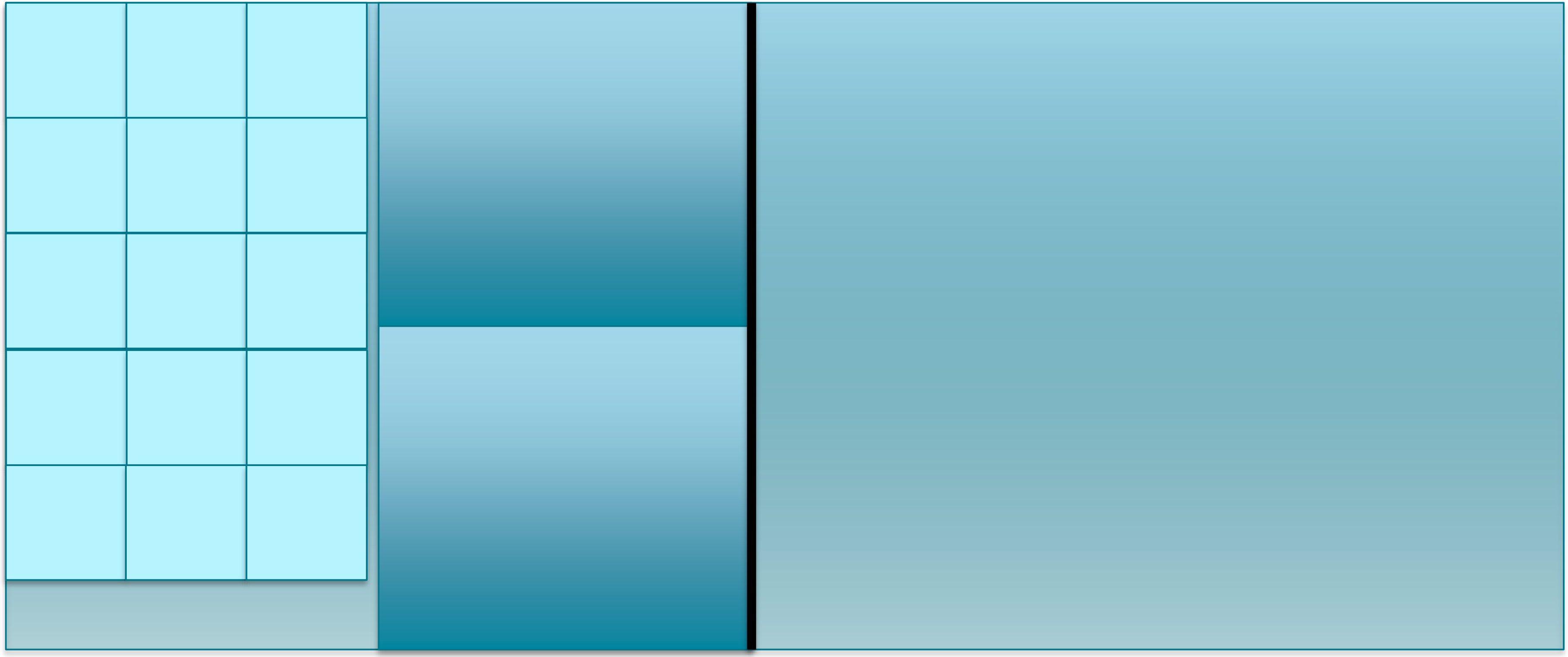
Eden

Survivor0 (S0)

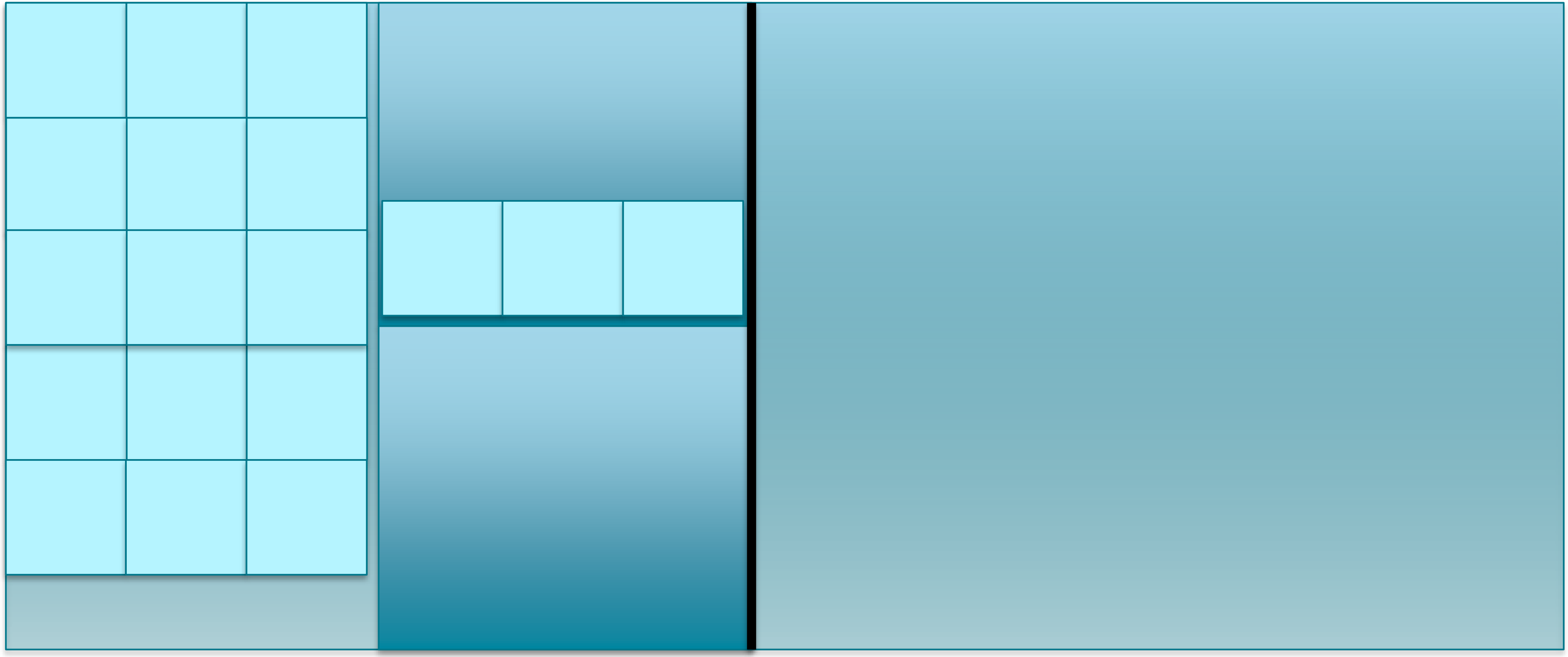
Survivor1 (S1)

Old Generation

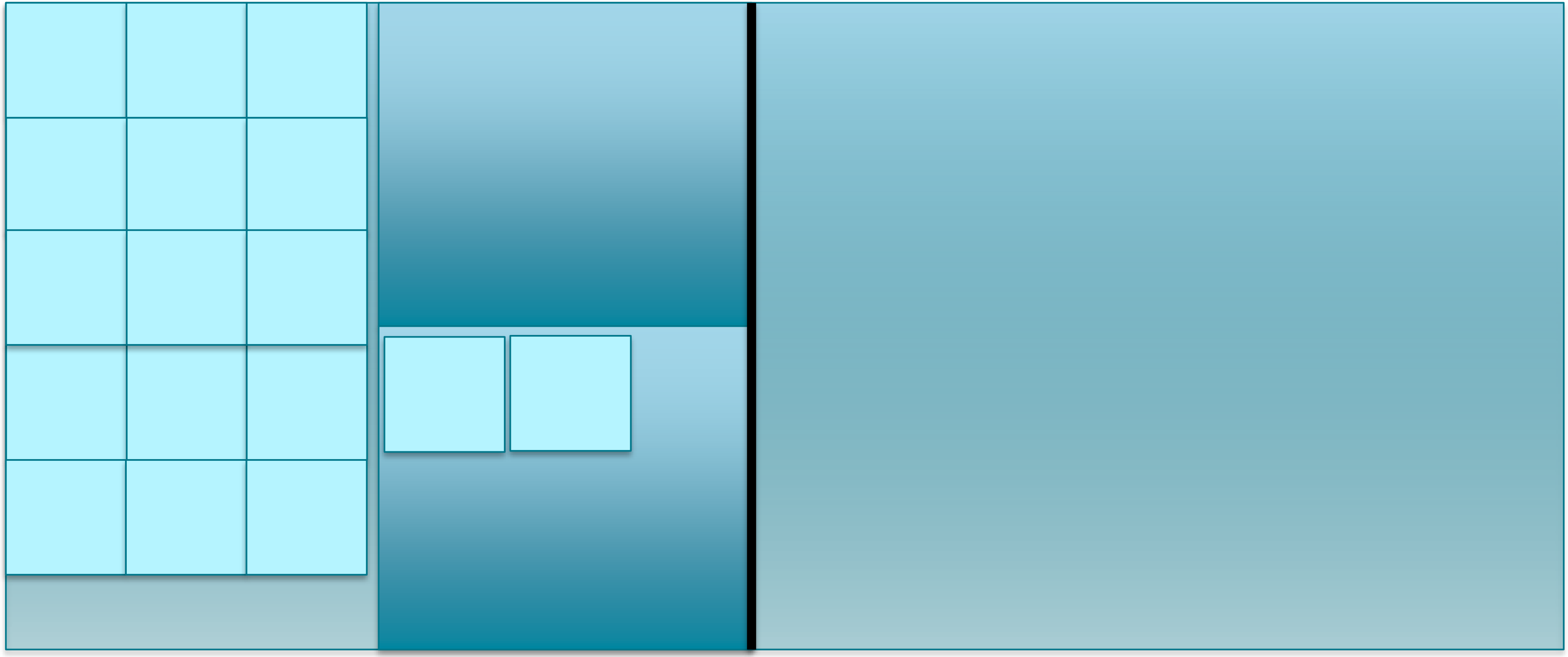
Memory Usage – Java Heap



Memory Usage – Java Heap



Memory Usage – Java Heap



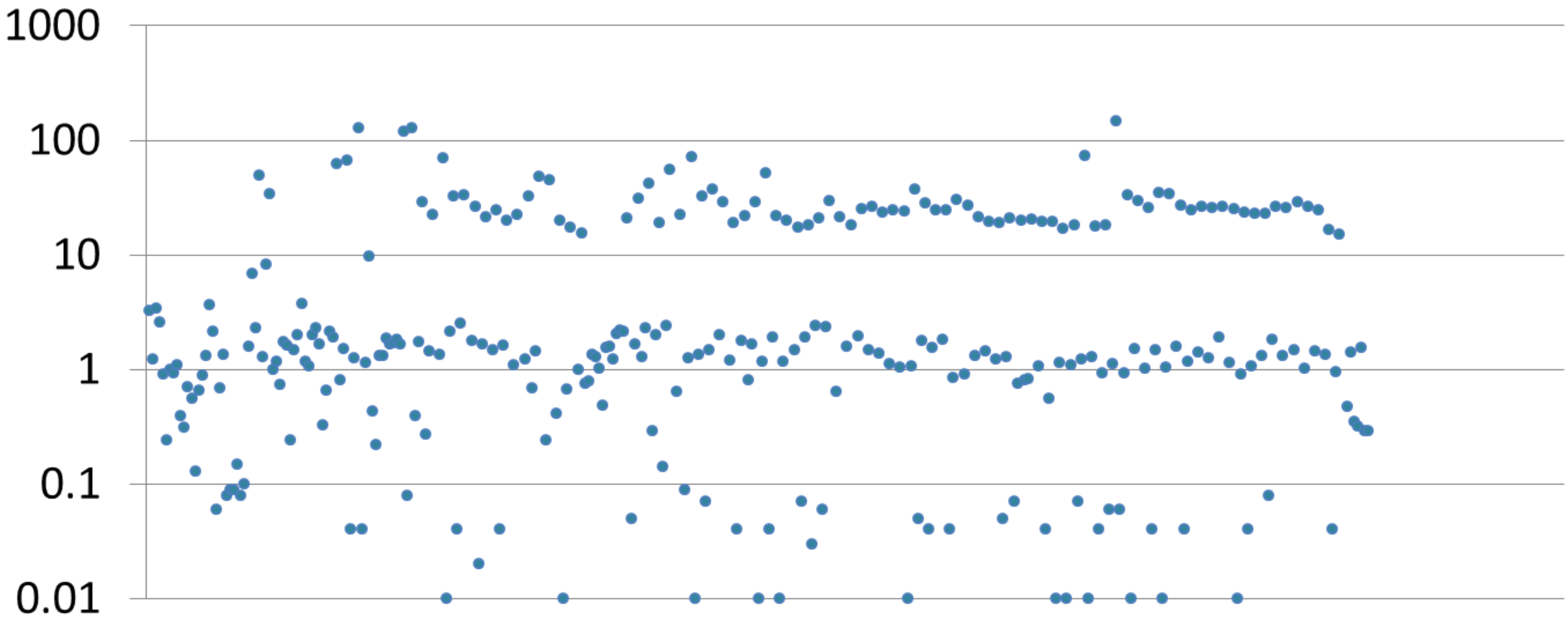
Memory Usage – Java Heap

- High allocation rates in general is not an issue
  - As long as objects become garbage quick enough
- Short requests are usually easier to handle
- Long running requests are challenging for GC
  - Those that keep large active data set
- Large live set is an issue
  - GC takes proportionally more time

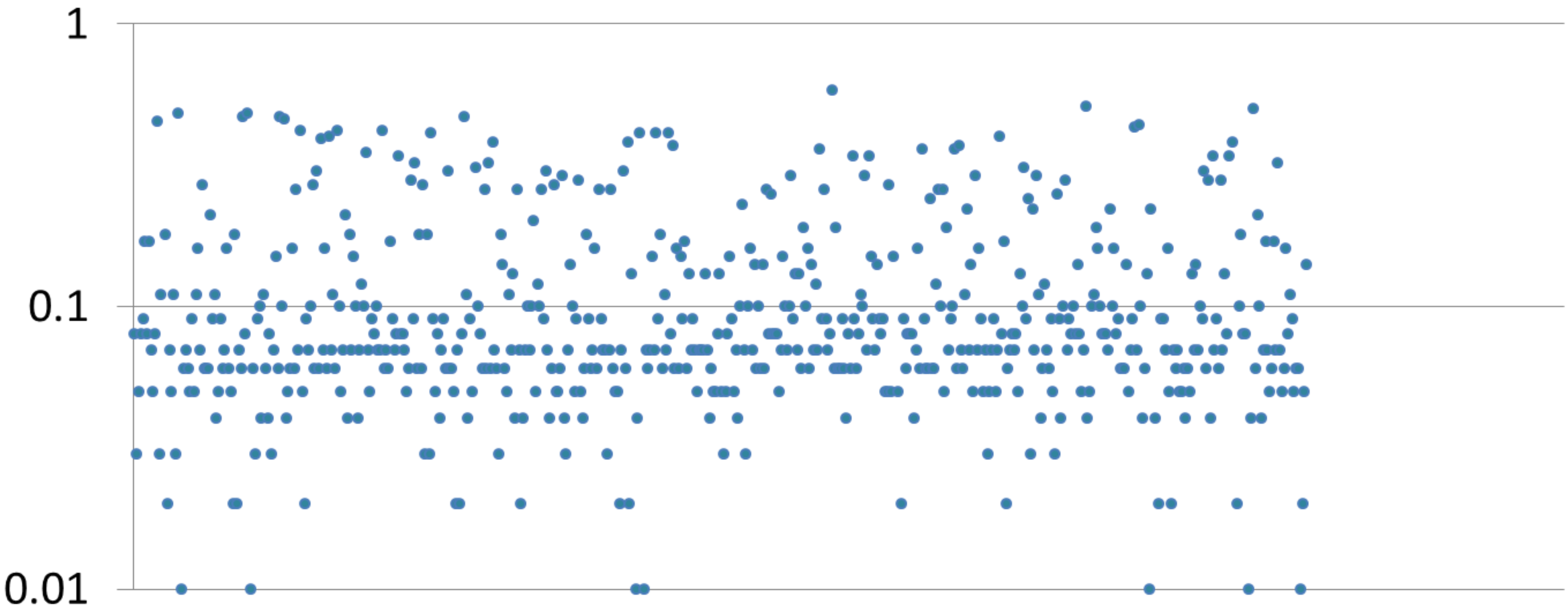
High Level Comparison of Collectors

Feature	ParallelOld	mCMS	G1
Live Data Set	Small to Medium	Medium to Large	Medium to XXL
Major GC pauses	Up to few secs	50..500ms+	Up to few secs
Memory Usage	Minimal	Medium	Large
Target	Throughput	Latency	Throughput or Latency
Downsides	High pause times with large live sets	* Fragmentation * Serial Full GC if promotion failure	* Complicated * Often it is slower than CMS (yet)

GC Times Before, sec



GC Times After, sec



Tools for GC Monitoring

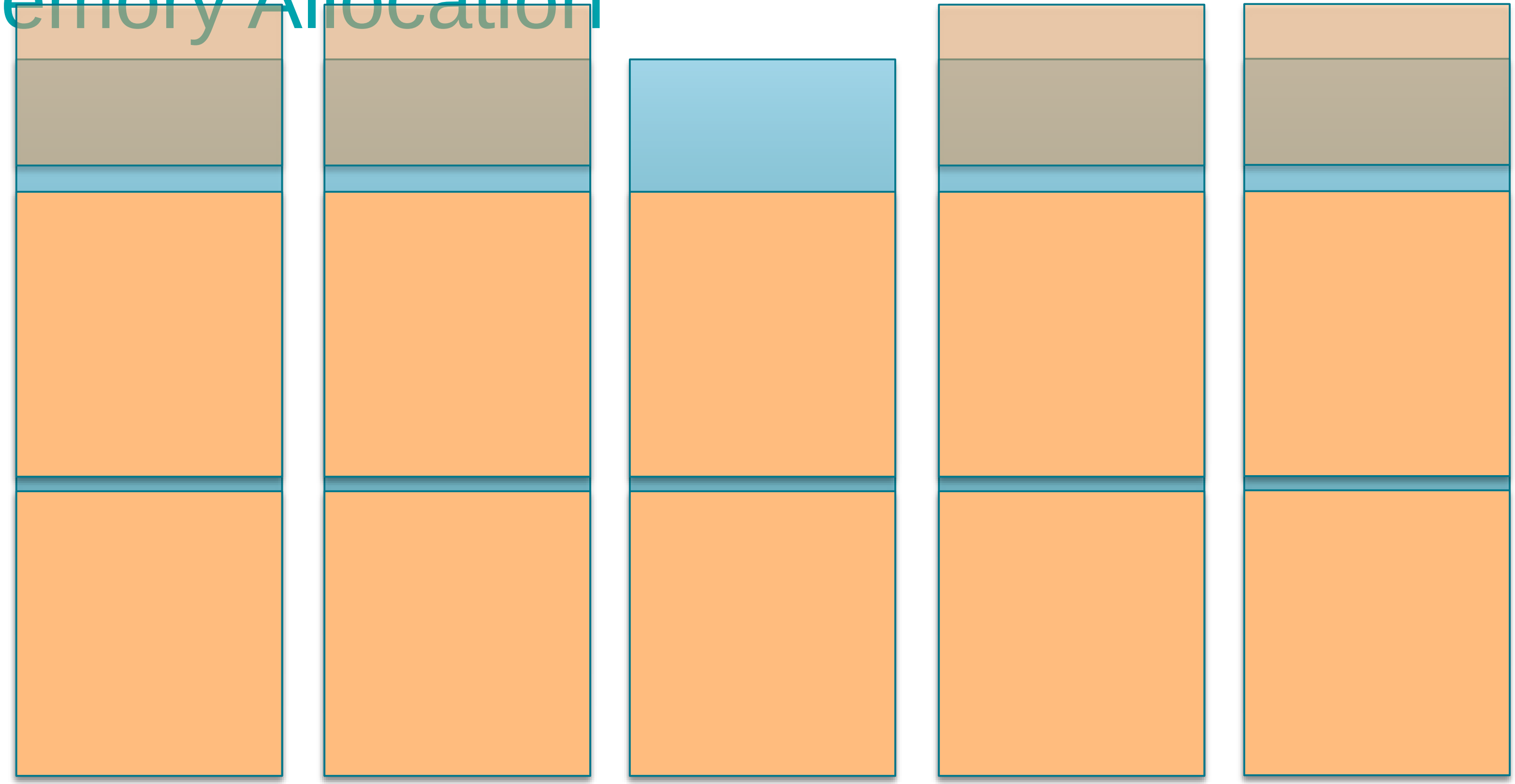
- GC log + GCViewer / http://gceasy.io
- jstat: command line, tabular output

```
jstat -gcutil PID 5s 10
```

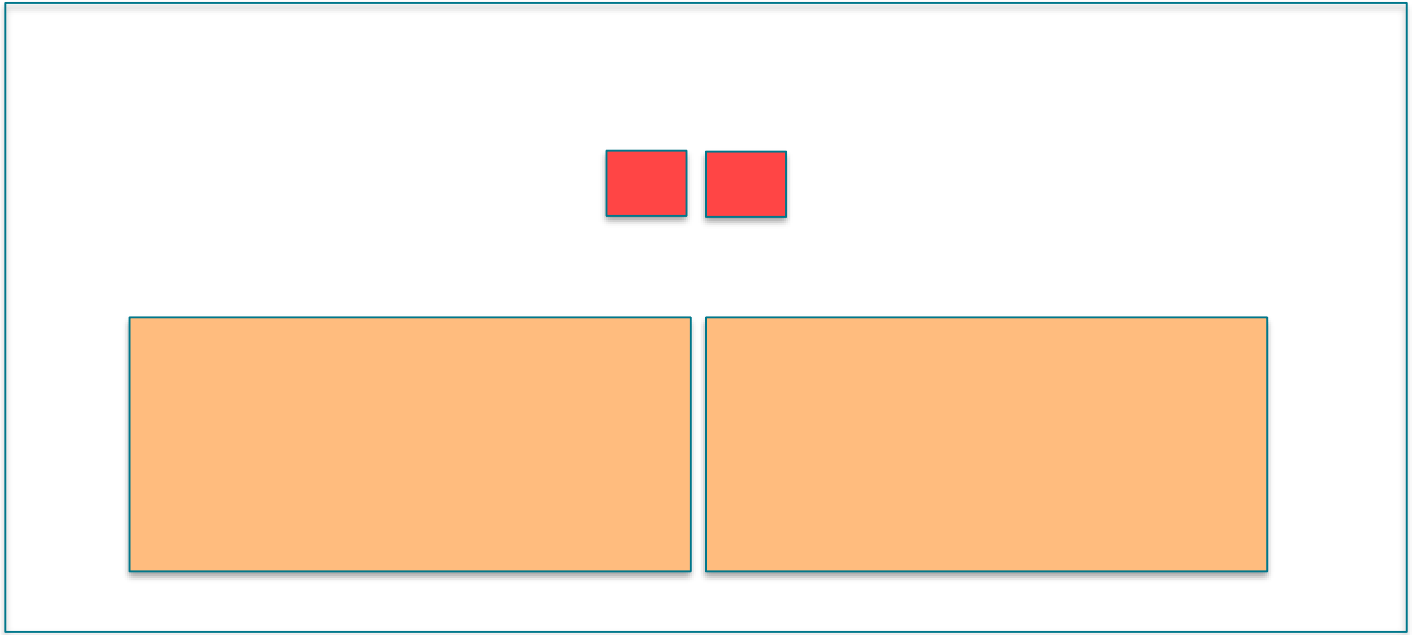
- jconsole/jvisualvm/jmc

Resources Allocation

Memory Allocation



CPU Allocation



Resources Allocation Advice

- Follow generic sizing rules
- Do not allocate less than 4 vCPUs per JVM
- Run 1 App Server per VM

Resources Allocation Advice

- Do not allocate more than 2 node MW cluster
  - If you don’t know how many you really need
- Allocate dedicated instances for critical services
- Split short & batch tasks between nodes

Resources Allocation Advice

Know the limits of an App Server instance

- concurrent users
- requests/second
- traffic/second and /request
- queries/second and /request
- garbage/second
- how big live set could be
- bottlenecks

Resources Allocation – DB Connections

- Large dynamic connection pools do not work

  http://www.youtube.com/watch?v=Oo-tBpVewP4

  http://www.youtube.com/watch?v=XzN8Rp6glEo
- The problem is easy to appear with
  - large MW clusters
  - multiple connection pools to same DB

DB Connections Advice

`DBCPUs*10/N`

Optimistic Use of Oracle RAC

Optimistic Use of RAC

- Clients want RAC because “HA & scalable”
  - Especially those clients that never had it in-house
- Expectations are
  - all apps scale well in RAC
  - RAC provides protection from node failures
- Often licenses are acquired in advance

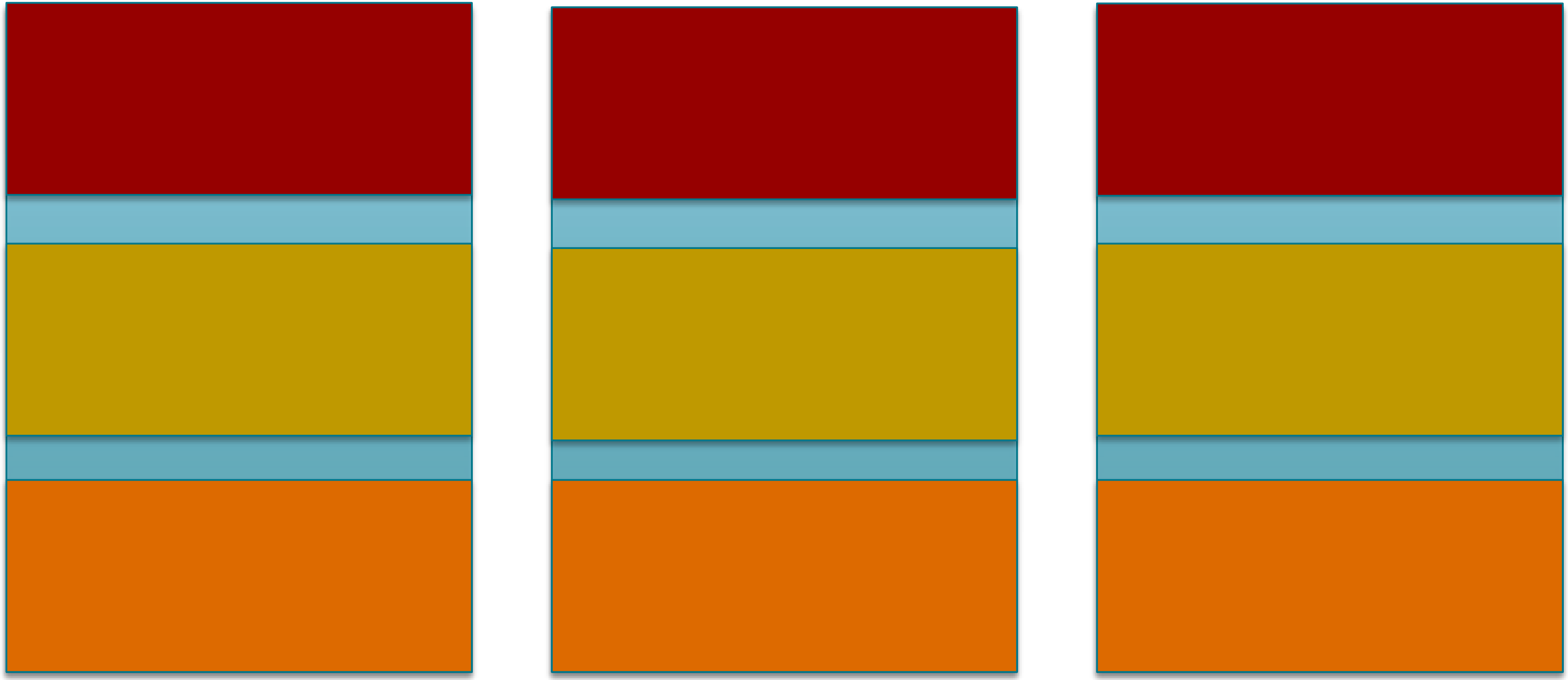
Optimistic Use of RAC

- SQL spending time in gc waits
- App behaves worse than with single instance DB
- Sometimes clients think it’s not enough HW and try to add more nodes to RAC

RAC Advice

- Treat it as a consolidation platform
- Use services. Even without RAC!
- Service Affinity to single node

Optimistic Use of RAC



RAC Advice

- Using 5y+ HW makes no sense
- Follow OraCHK recommendations (carefully)
- Active GridLink with WebLogic
  - https://docs.oracle.com/middleware/1212/wls/JDBCA/gridlink_datasources.htm

Unreliable Statistics Management

Typical Statistics Management

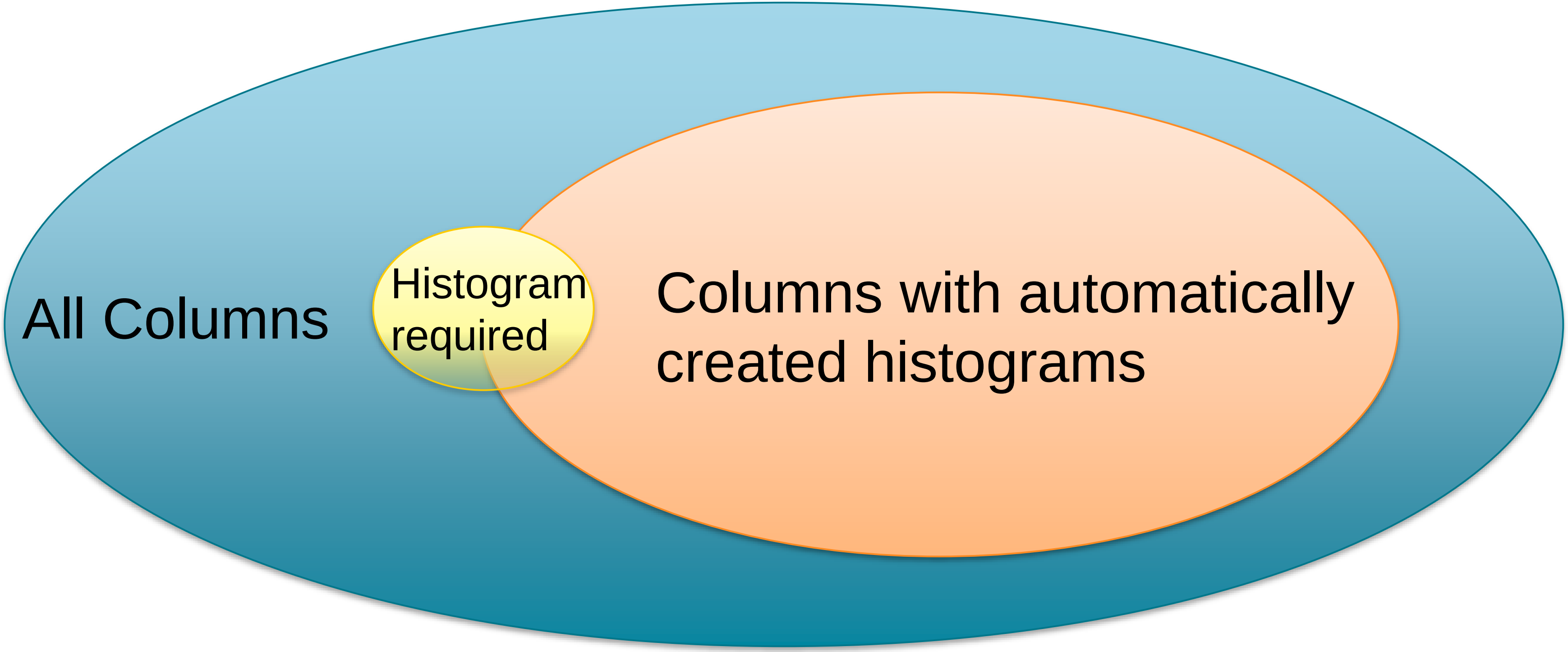
- Default task makes changes in production
  - silently with no change control
  - same effect as testing code in production right away
- Usually runs way more often than needed
  - Some clients run it manually even more often
- Histograms by default METHOD_OPT

Histograms

- By default Oracle creates a histogram when
  - Column is used in SQL condition
  - Skew in the column data distribution

Histograms

- When App really needs a histogram
  - Column is used in SQL condition
  - Skew in the column data distribution
  - App uses literals in SQL condition
  - Histogram helps SQL to run optimally



All Columns

Histogram required

Columns with automatically created histograms

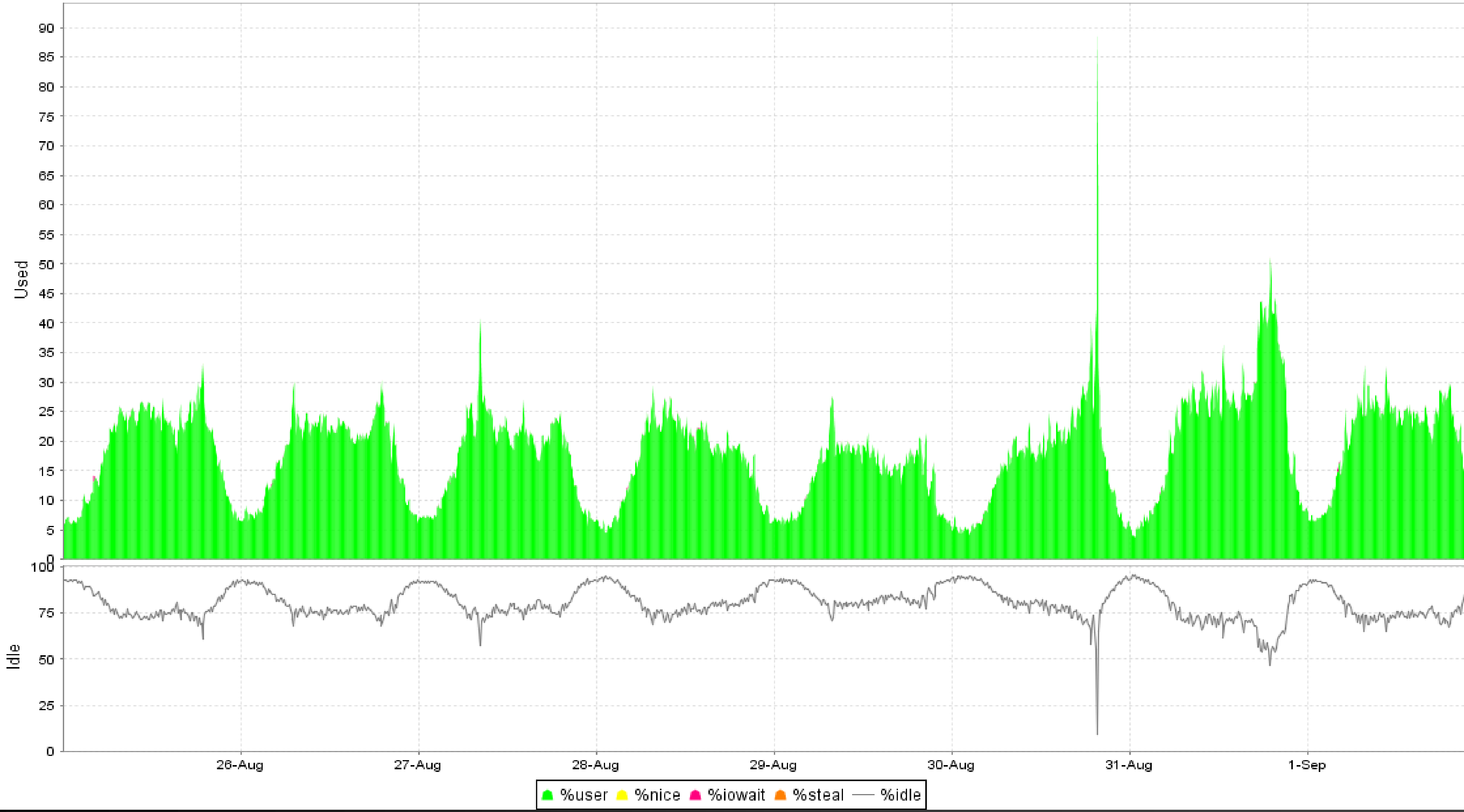
Consequences of Histograms

- As a result of unnecessary histograms
  - Increased number of plans in memory
  - Unnecessary CPU, memory and disk overheads
  - Unexpected plan changes with bind peeking
    - Adaptive features suppose to help sometimes
- People “fix” it with different plan stability options
  - Without even trying to analyze the cause

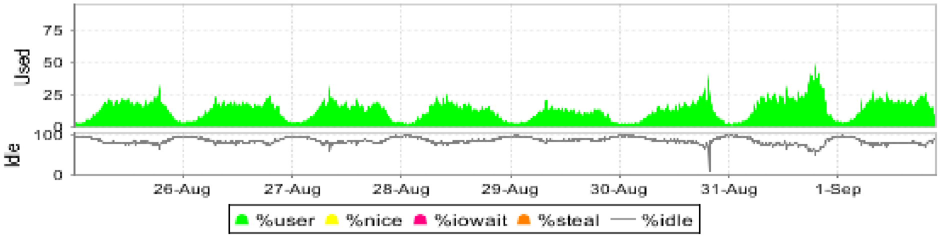
Statistics Management Advice

- Statistics as a code
  - Do not let Oracle to change your code at random
  - Set statistics as part of the code delivery
  - Fix Min/Max, bad histograms, partition stats, temp tables, new tables, etc.
- Create histograms manually

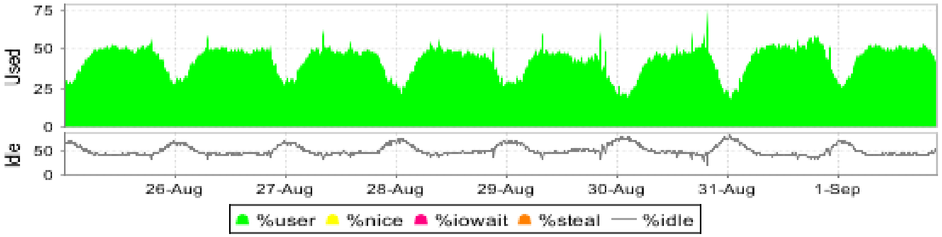
Inadequate Monitoring and Troubleshooting


Used
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
Idle
100
75
50
25
0
26-Aug
27-Aug
28-Aug
29-Aug
30-Aug
31-Aug
1-Sep
%user
%nice
%iowait
%steal
%idle

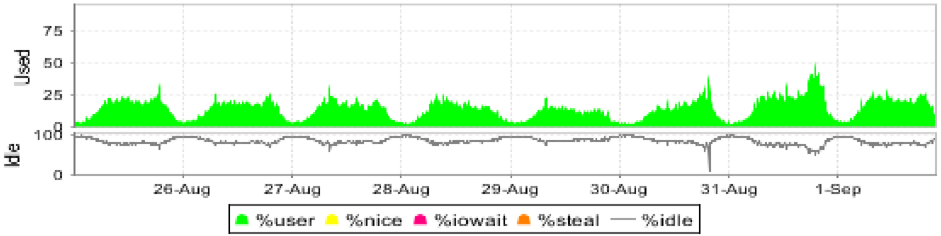
CPU 0



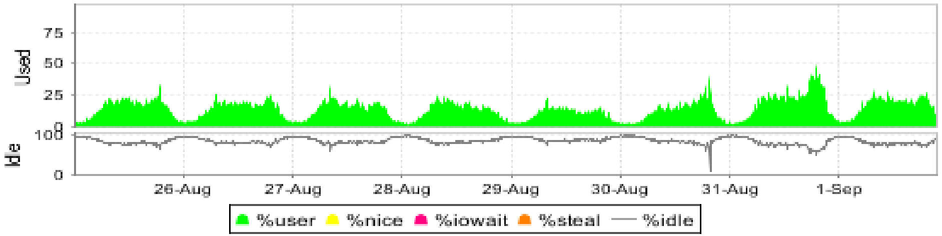
CPU 1



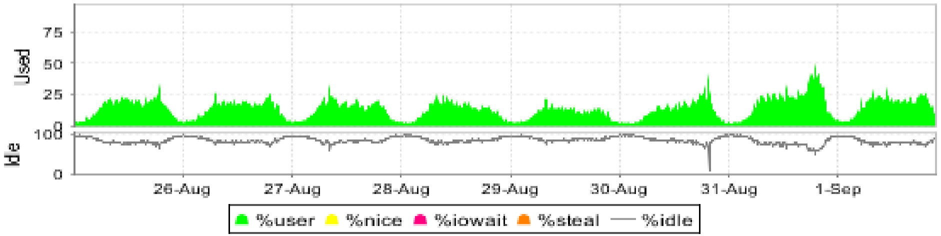
CPU 2



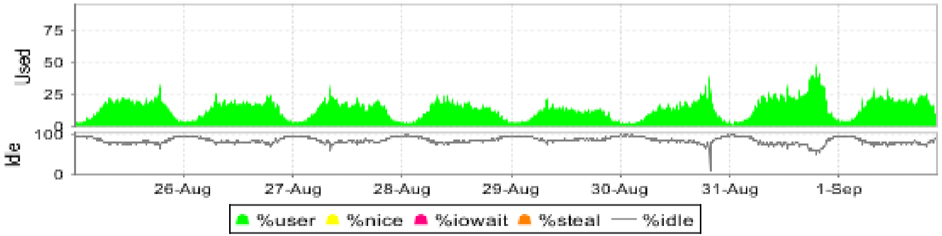
CPU 3



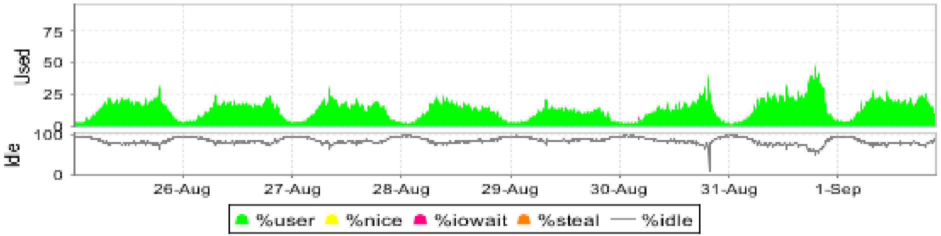
CPU 4



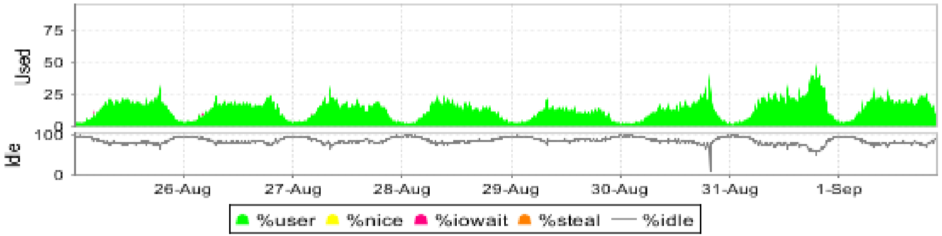
CPU 5

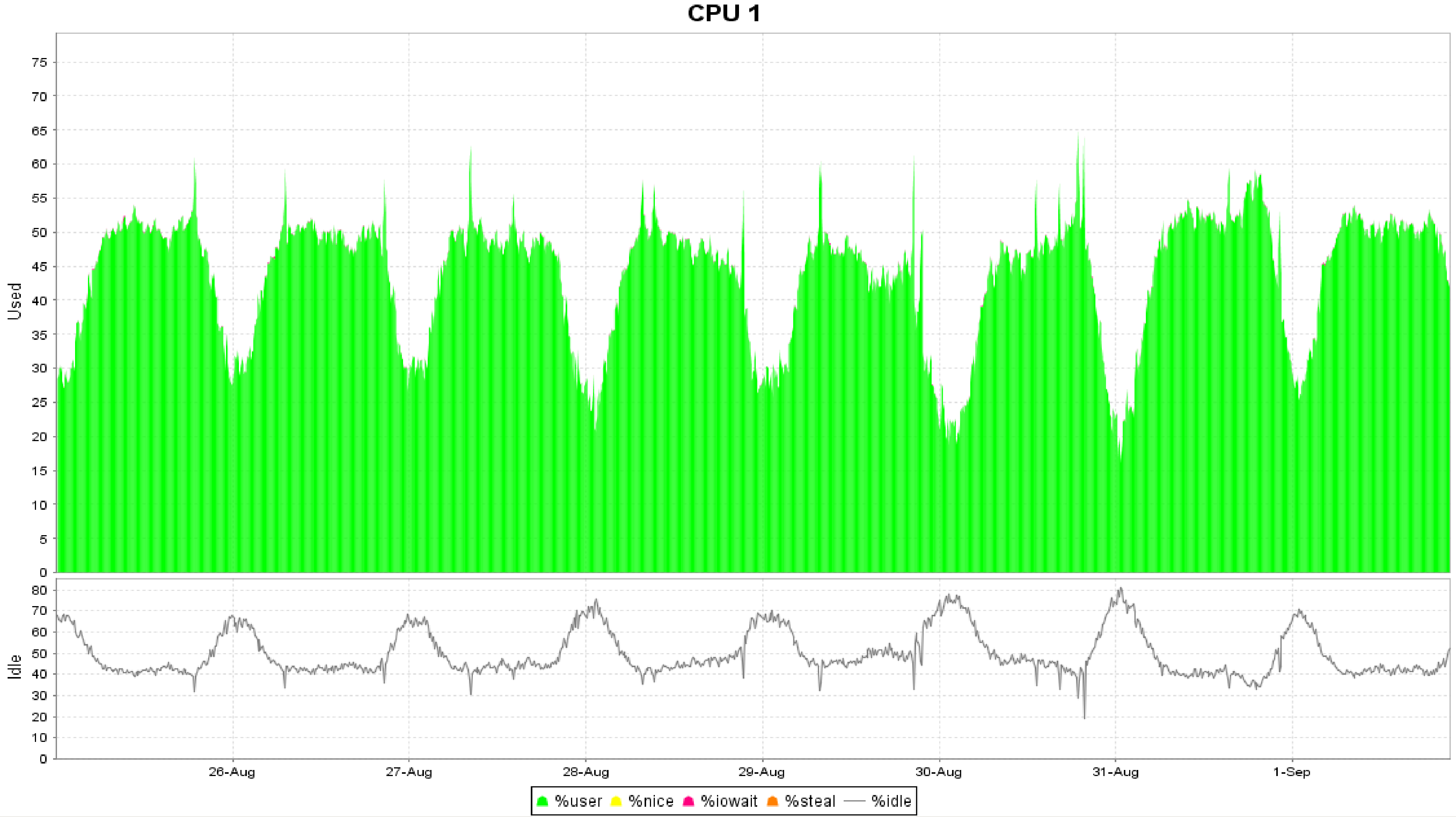


CPU 6



CPU 7




CPU 1
Used
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
Idle
80
70
60
50
40
30
20
10
0
26-Aug
27-Aug
28-Aug
29-Aug
30-Aug
31-Aug
1-Sep
🟩 %user
🟨 %nice
🟥 %iowait
🟧 %steal
— %idle

Troubleshooting

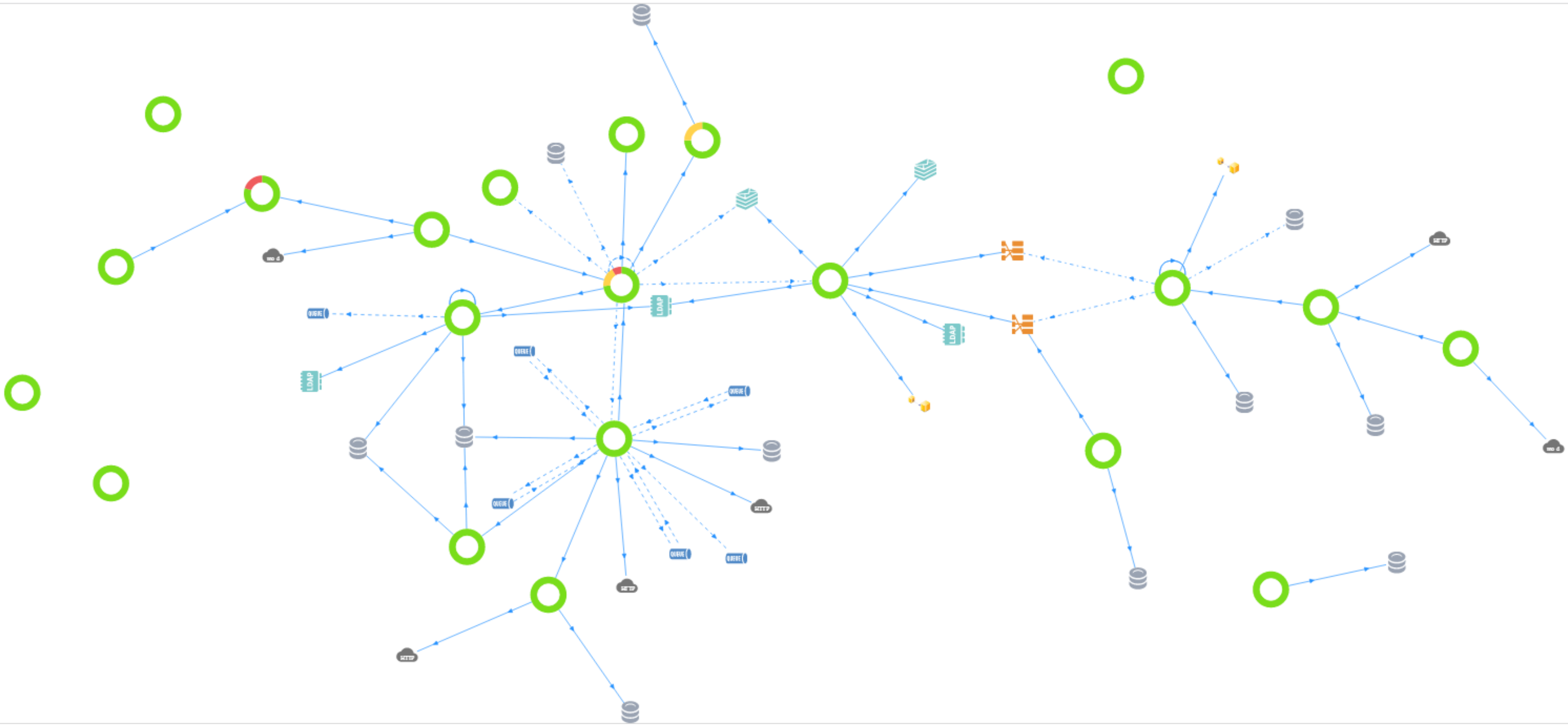
Minimal Diagnostics

- OS level metrics
- GC activity: log and/or jstat
- Thread Dumps: top + poor man’s profiler
  - http://www.pythian.com/blog/a-simple-way-to-monitor-java-in-linux/
  - jvmtop https://github.com/patric-r/jvmtop
  - SJK https://github.com/aragozin/jvm-tools
- Heap Dump + Memory Analyzer

Troubleshooting

- Recommended things to have
  - Application Performance Management software
    - AppDynamics
    - NewRelic
  - Java Flight Recorder
  - JVisualVM

APM


QUEUE
QUEUE
QUEUE
QUEUE
QUEUE
QUEUE
LDAP
LDAP
LDAP
HTTP
HTTP
HTTP
HTTP
HTTP

Summary

- Memory efficiency
- Start small. Scale up first
- RAC as a consolidation platform
- Think and plan stats management
- Don’t just restart. Gather diagnostics.

Thank You!

Q & A