ORACLE®

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remain at the sole discretion of Oracle.


Extreme (WLS/Java) Performance Workshop

JVM Performance Tuning

Zhao Yi

Consulting Solution Architect – OFM A Team

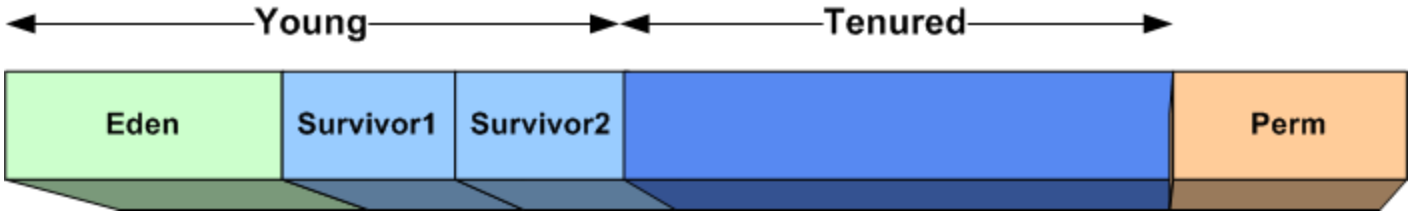
ORACLE

Agenda

- JVM Fundamentals
- JVM Performance Tuning
- GC Fundamentals
- Hotspot Internals
- Hotspot Tuning
- Diagnosing GC Issues

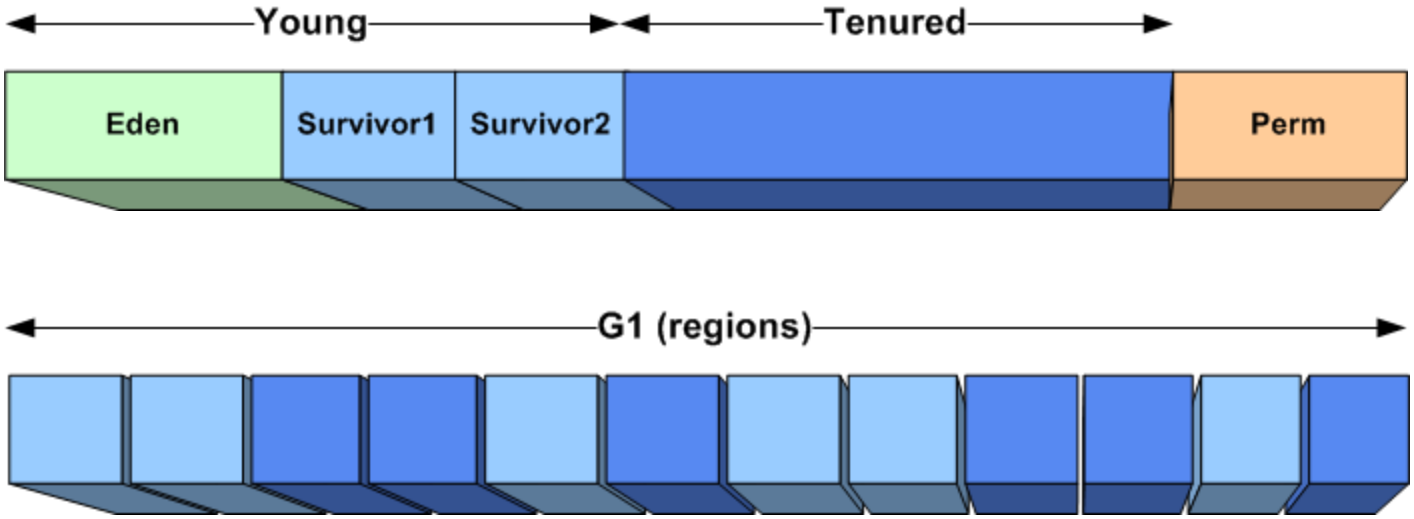
JVM Fundamentals

Hotspot Heap – JDK 6



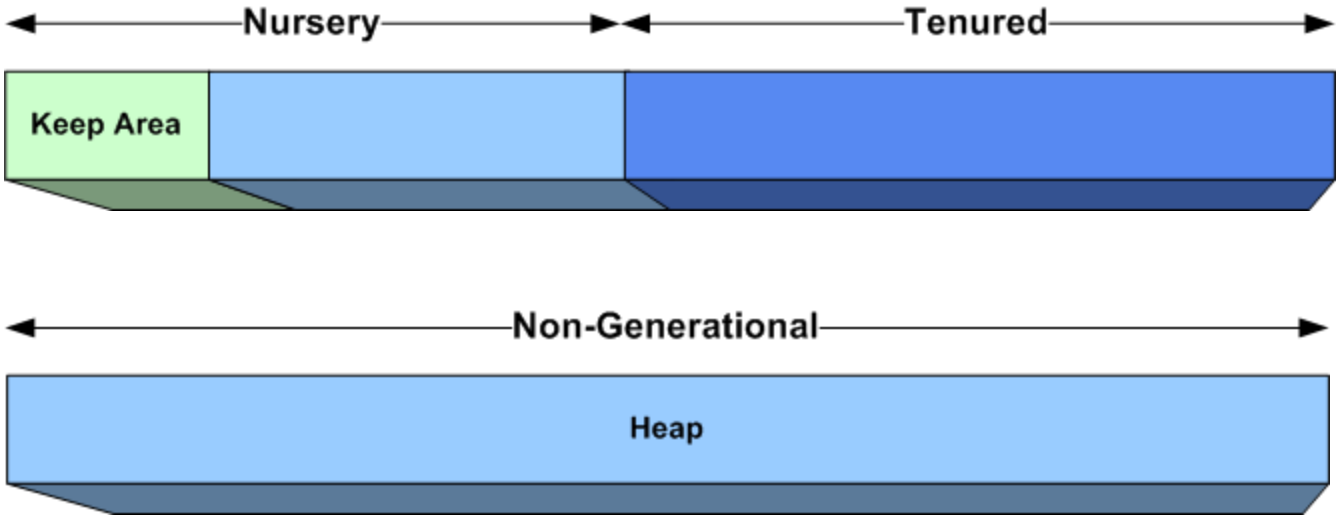
JVM Fundamentals

Hotspot Heap – JDK 7



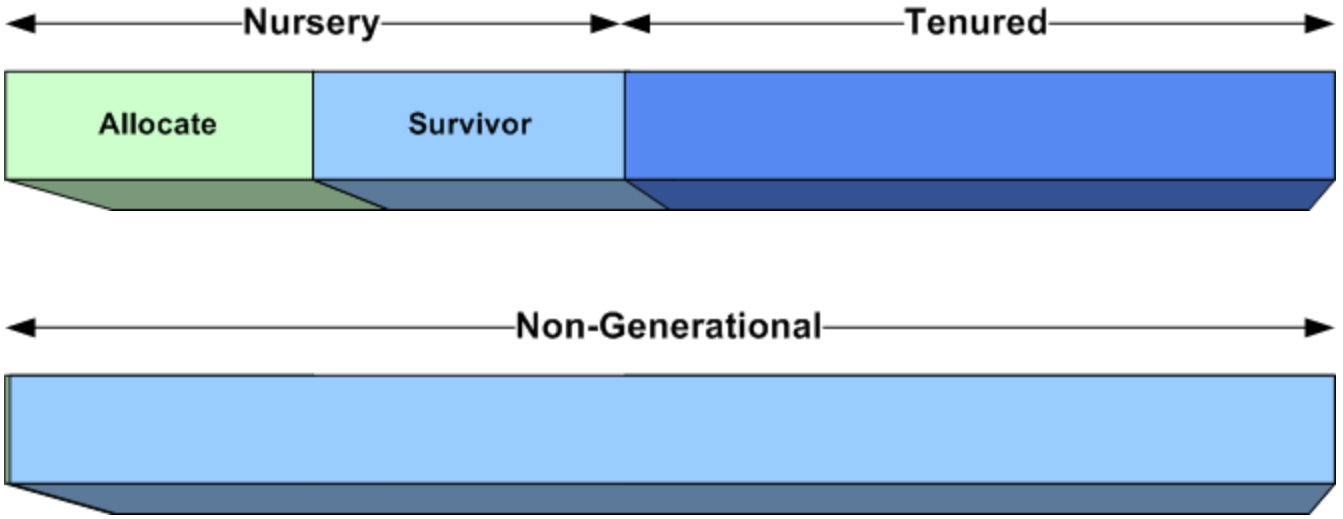
JVM Fundamentals

JRockit Heap



JVM Fundamentals

IBM JVM



JVM Fundamentals

Java Compilation

- Interpreted Mode
  - Byte code (.class) is interpreted initially*
- Just in Time (JIT) Compile
  - Byte code is compiled to Native code when marked as a ‘hotspot’
    - E.g. Method count crosses ‘-XX:CompileThreshold’ in Hotspot JVM
    - *JRockit only runs in compiled mode
- Optimization
  - JVM collects heuristics and optimizes compiled code

JVM Fundamentals

Adaptive Memory Management

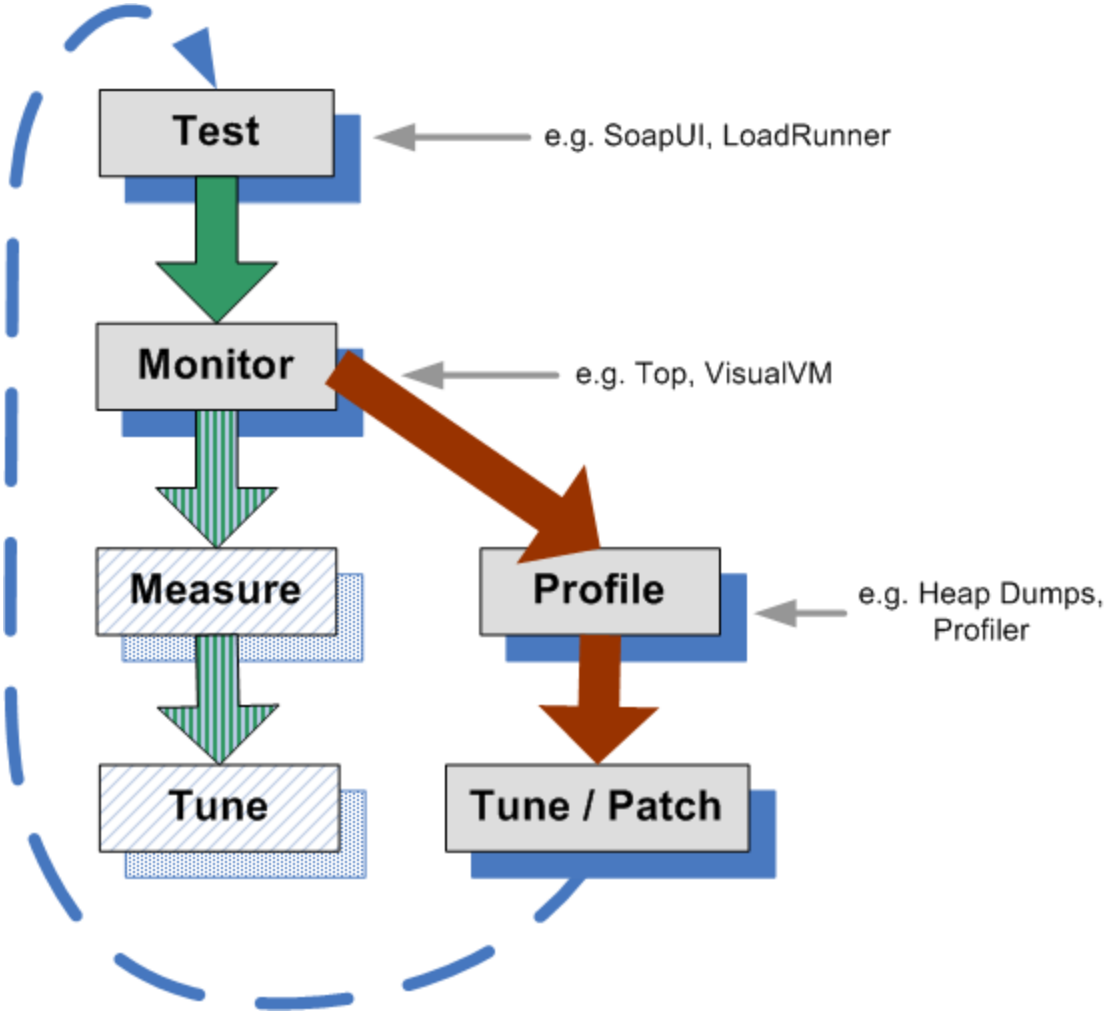
- Hotspot calls it ‘Ergonomics’
- Collects heap usage and GC statistics
- Some values selected at startup based on server type
  - For instance, generation sizes, heap size, gc threads
- Some values adjusted dynamically based on statistics
  - For instance, survivor space
  - JRockit switches GC algorithm dynamically
- Good out of the box performance
- *Manually well tuned JVM yields better performance*

Agenda

- JVM Fundamentals
- JVM Performance Tuning
- GC Fundamentals
- Hotspot Internals
- Hotspot Tuning
- Diagnosing GC Issues

JVM Performance Tuning

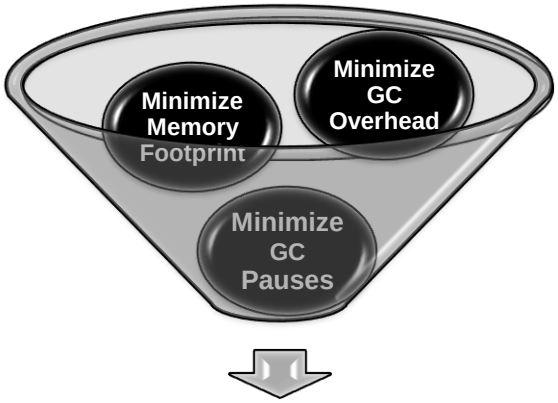
Performance Tuning Process



JVM Performance Tuning

Performance metrics for Tuning

- **Memory Footprint**
  - Less important on 64-bit
  - Prevent swapping
- **Startup Time**
  - Somewhat important (e.g. production)
- **Throughput**
  - Transactions processed over time (e.g. TPS)
  - Important
- **Responsiveness**
  - Latency, round trip time, user wait time, etc
  - Very Important
  - NOTE: Common mistake to use single message latency as indication of machine performance



You can only pick two!!

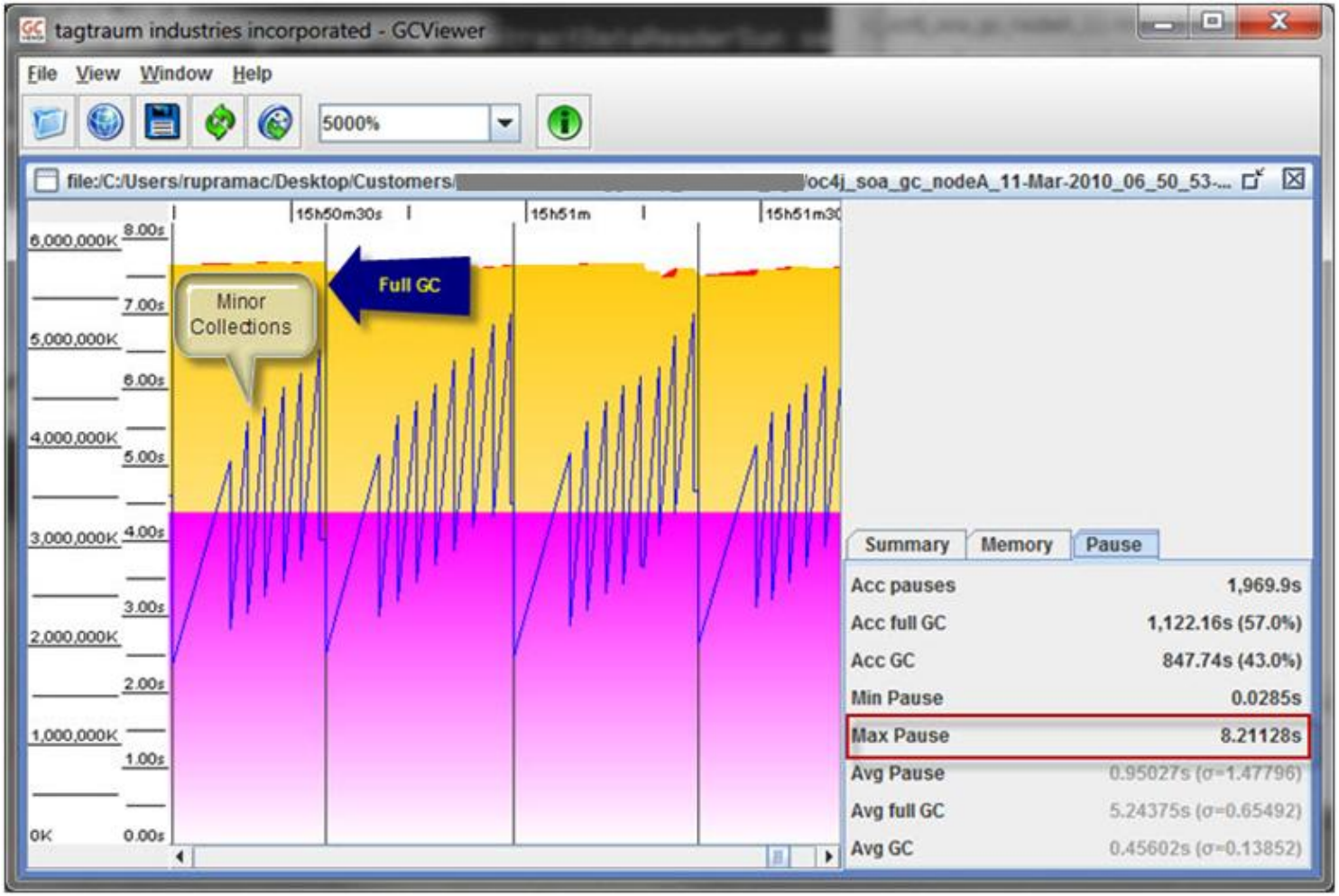
JVM Performance Tuning

JVM Monitoring and Profiling

- JVM Monitoring tools
  - Online: JRMC (JRockit), VisualVM (Hotspot)
  - Offline: GC logs, GCViewer (all), JFR
- JVM Profiling
  - Memory, CPU, Lock/Monitor profiling
  - Profilers
    - Most IDE's (JVMTI)
    - MAT (Heap dumps)
    - JRMC (JRockit)
    - VisualVM (Hotspot)
    - 3rd Party profilers (JProfiler, YourKit)

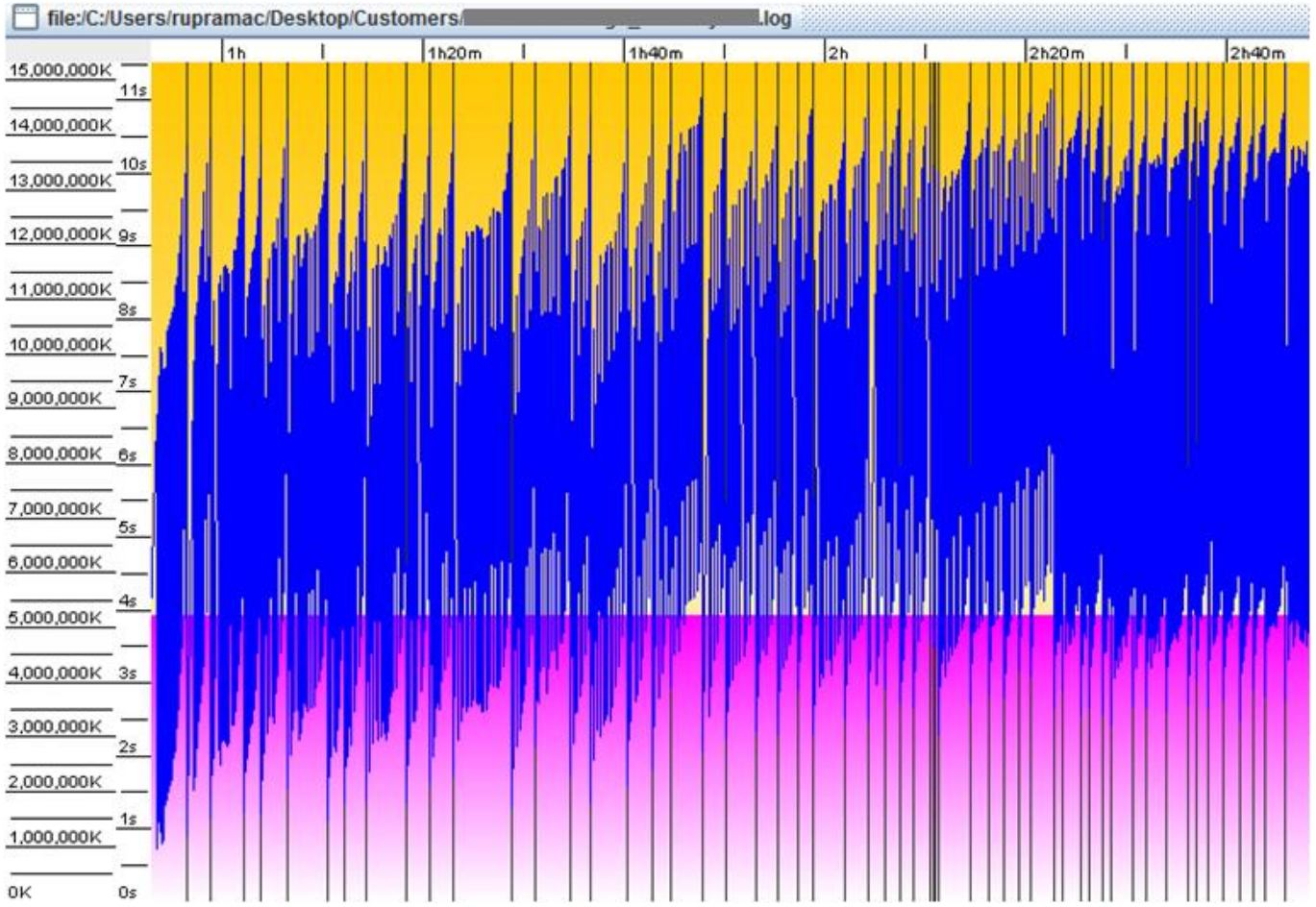
JVM Performance Tuning

GCViewer – Offline analysis of GC logs



JVM Performance Tuning

GCViewer – Memory Leak Pattern

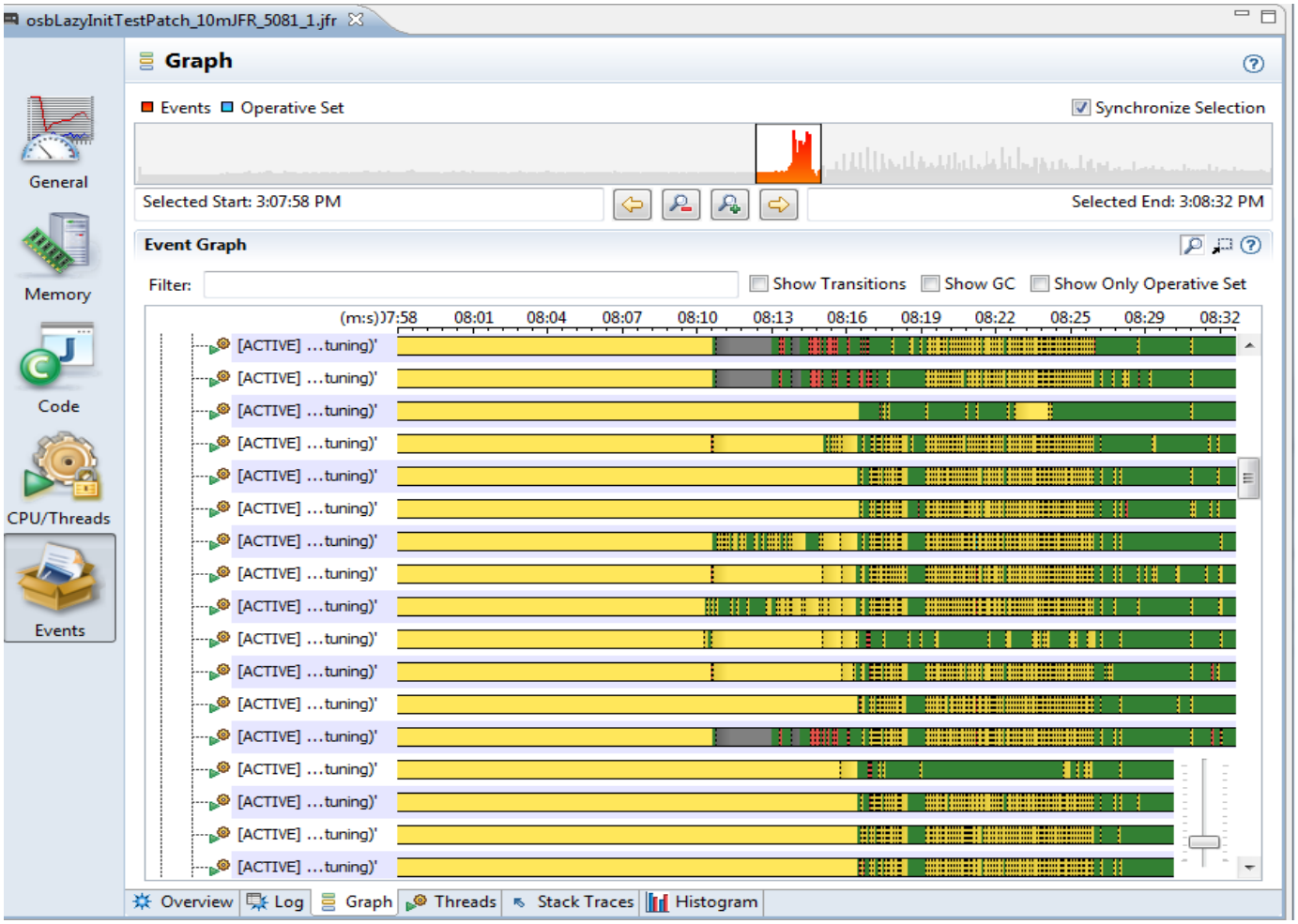


JVM Performance Tuning

JRockit Mission Control - Console

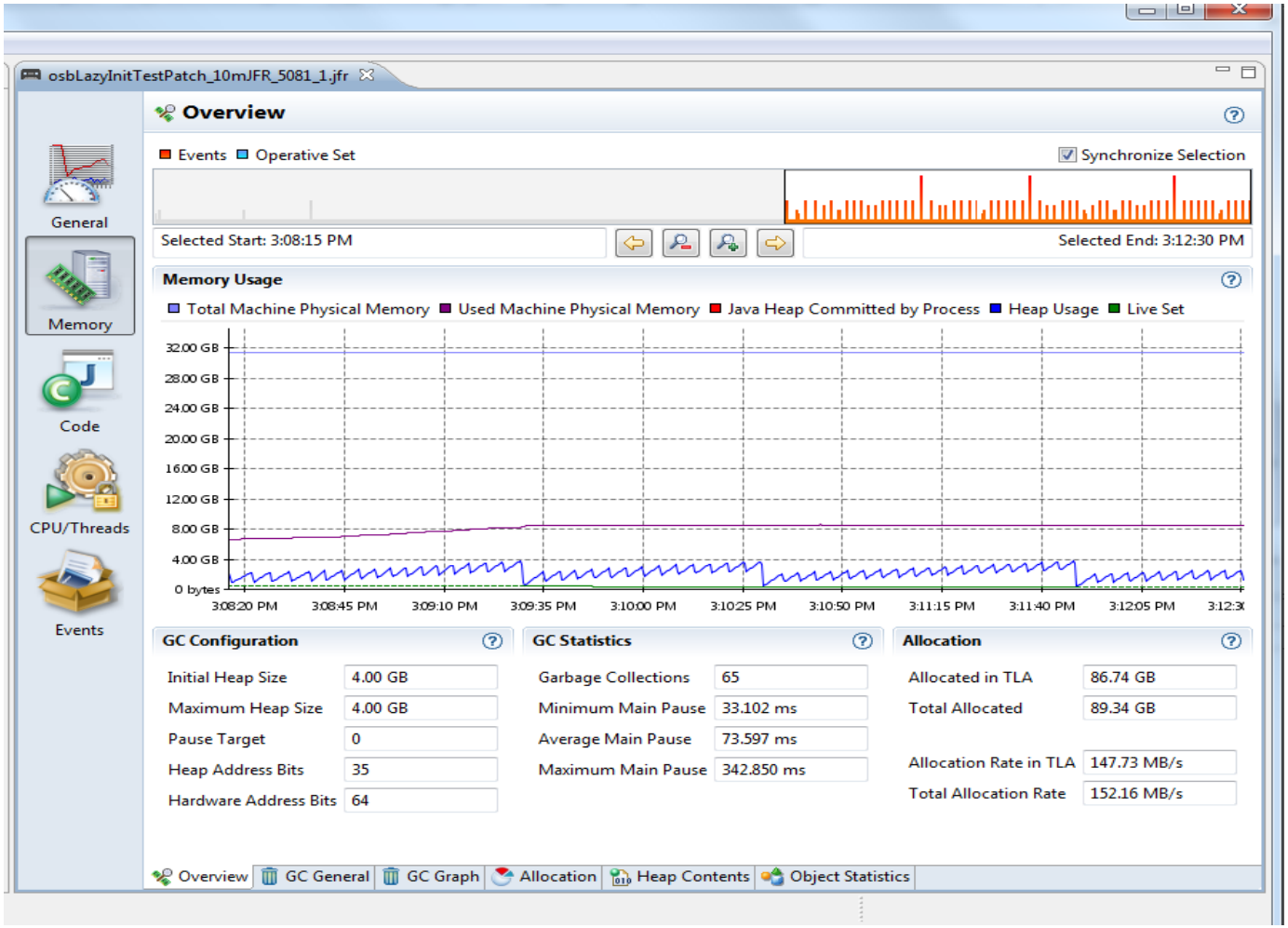
JVM Performance Tuning

JRockit Mission Control - JFR



JVM Performance Tuning

JRockit Mission Control - JFR



JVM Performance Tuning

GC Tuning - Generic

- We will cover minimum parameters yielding maximum performance
- General Tuning Advice
  - Keep it simple
  - Provide the basic parameters (-X parameters)
    - -Xms, -Xmx, -Xmn
  - Select a GC/performance priority
    - Throughput vs Pause Time
  - Keep most defaults for the rest
    - Let *ergonomics* compute the right values
    - Fine tune only if defaults don’t work

ORACLE

JVM Performance Tuning

GC Tuning Parameters - Generic

- Most commonly used GC parameters
  - Garbage collection policy (e.g. non-generational, concurrent, parallel GC, deterministic, etc)
  - GC parallelism (serial vs parallel vs concurrent)
  - Generation Sizes
- Most commonly used GC diagnostics parameters
  - GC logging (e.g. *-verbose:gc*)
  - Log Verbosity level. E.g. *-XX:+PrintGCDetails (*Hotspot*),* *–Xverbose:gcpause* (JRockit)
  - GC log file. E.g. *-Xloggc:<file> (*Hotspot*), -Xverboselog: (*JRockit)
  - ***GC logging is safe to leave on in production***

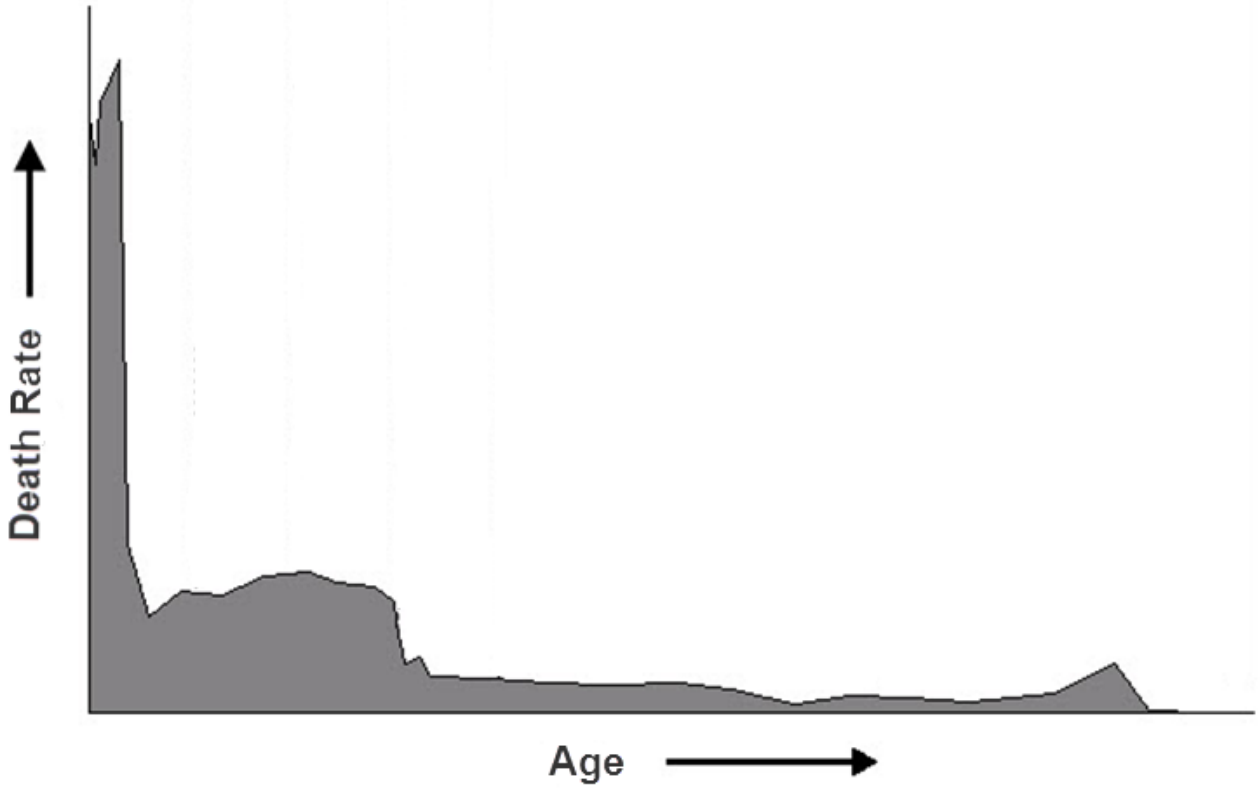
ORACLE

Agenda

- JVM Fundamentals
- JVM Performance Tuning
- GC Fundamentals
- Hotspot Internals
- Hotspot Tuning
- Diagnosing GC Issues

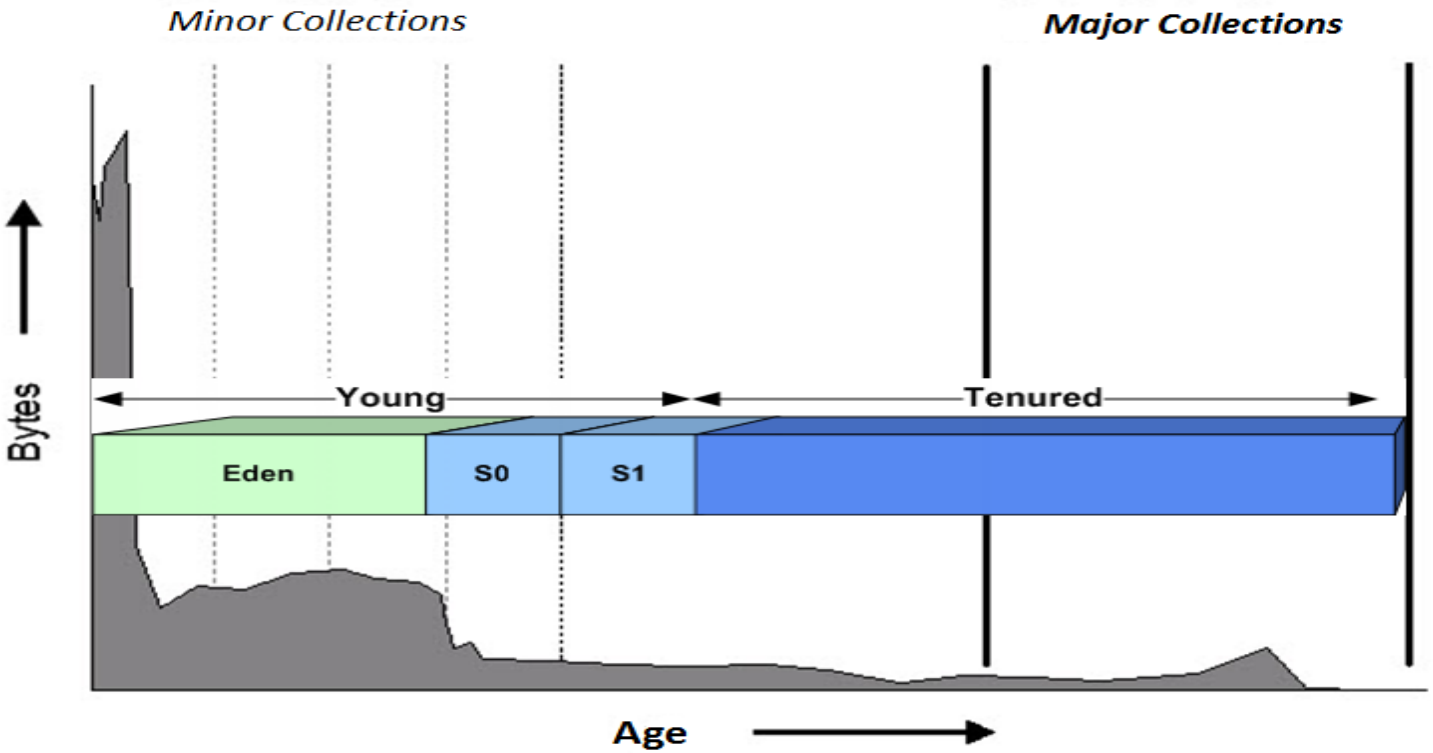
GC Fundamentals

Garbage Distribution – Objects die young



GC Fundamentals

Garbage Distribution - Generational Collection



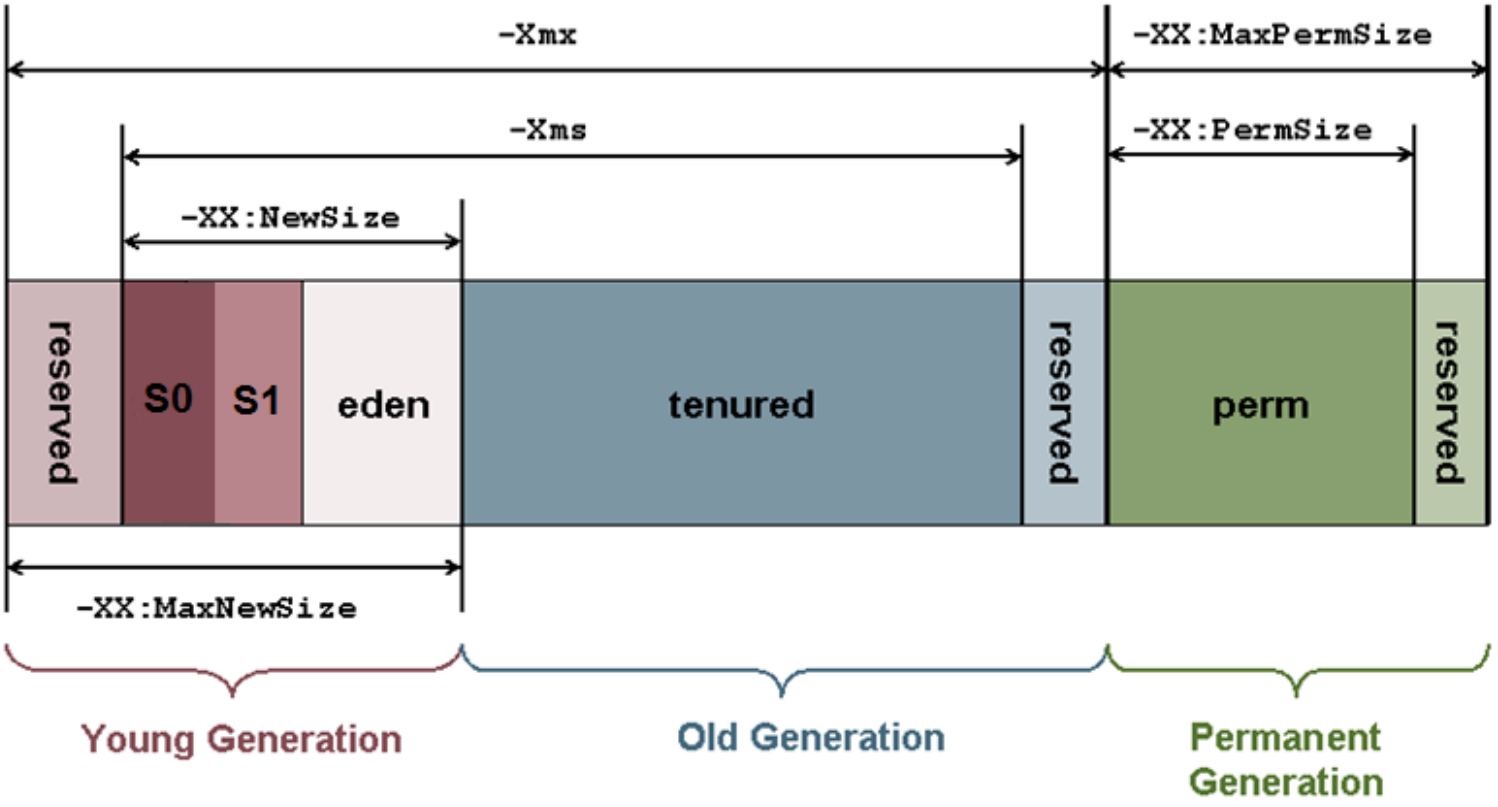
- Buckets ideally correspond to distribution curve
- Collect each bucket with most efficient algorithm
- Size buckets for best GC performance

Agenda

- JVM Fundamentals
- JVM Performance Tuning
- GC Fundamentals
- Hotspot Internals
- Hotspot Tuning
- Diagnosing GC Issues

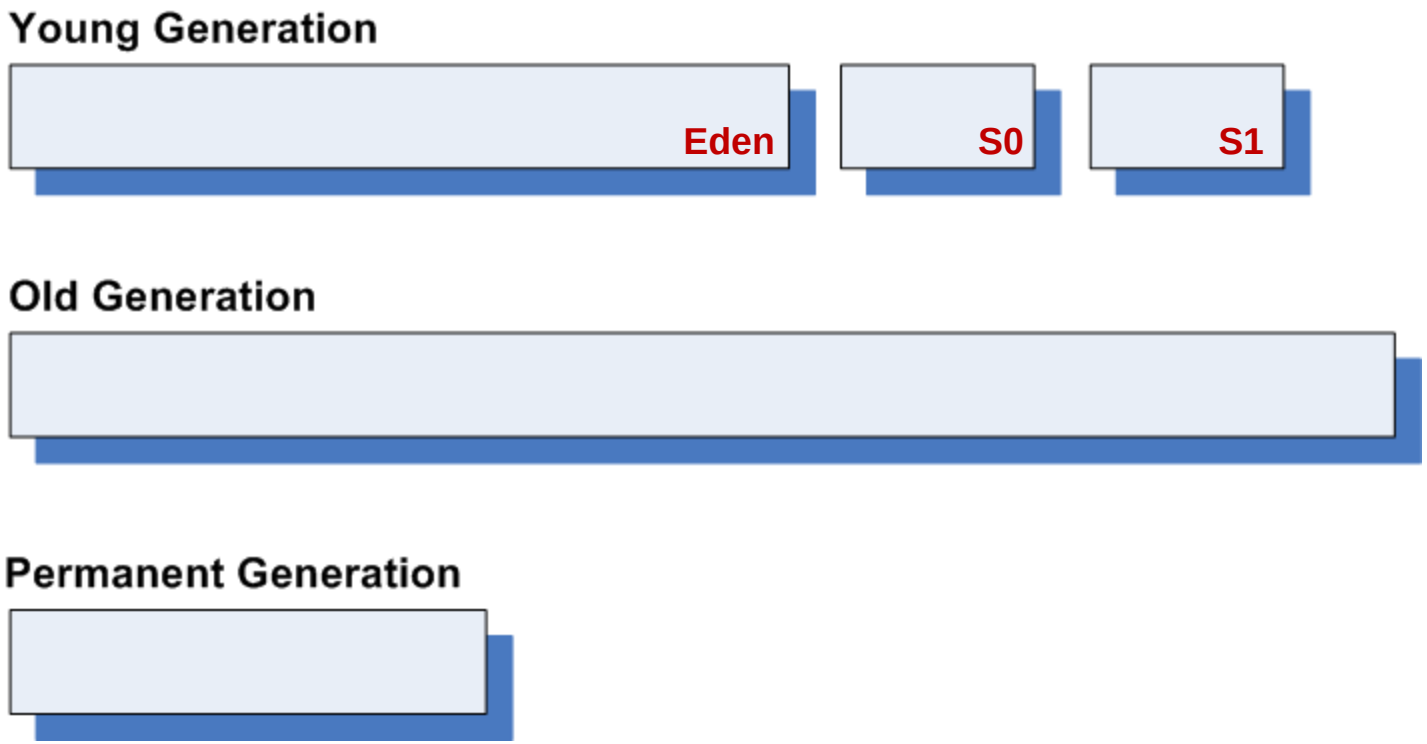
Hotspot Internals

Hotspot Heap



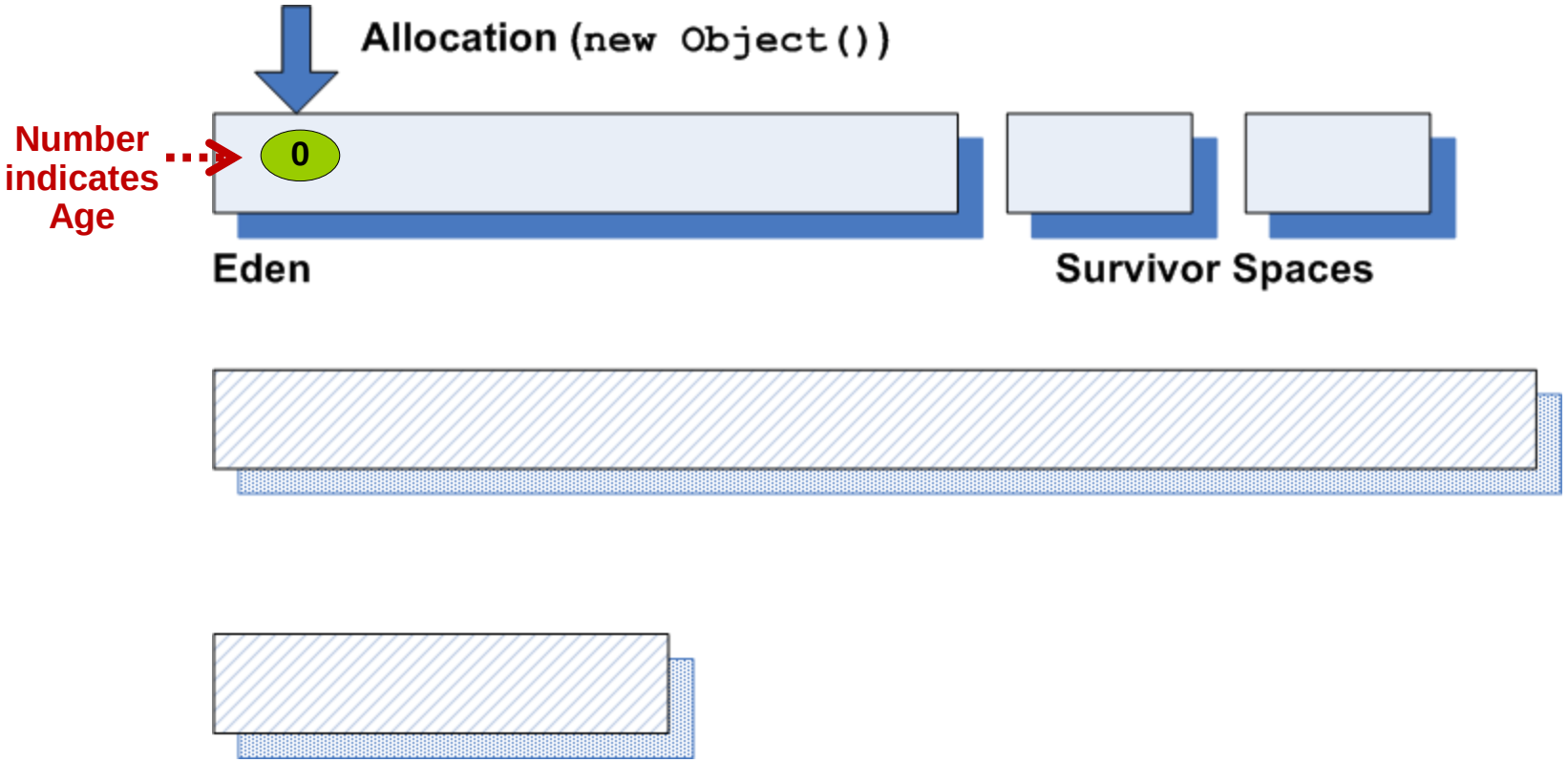
Hotspot Internals

Generations & Object Lifecycle



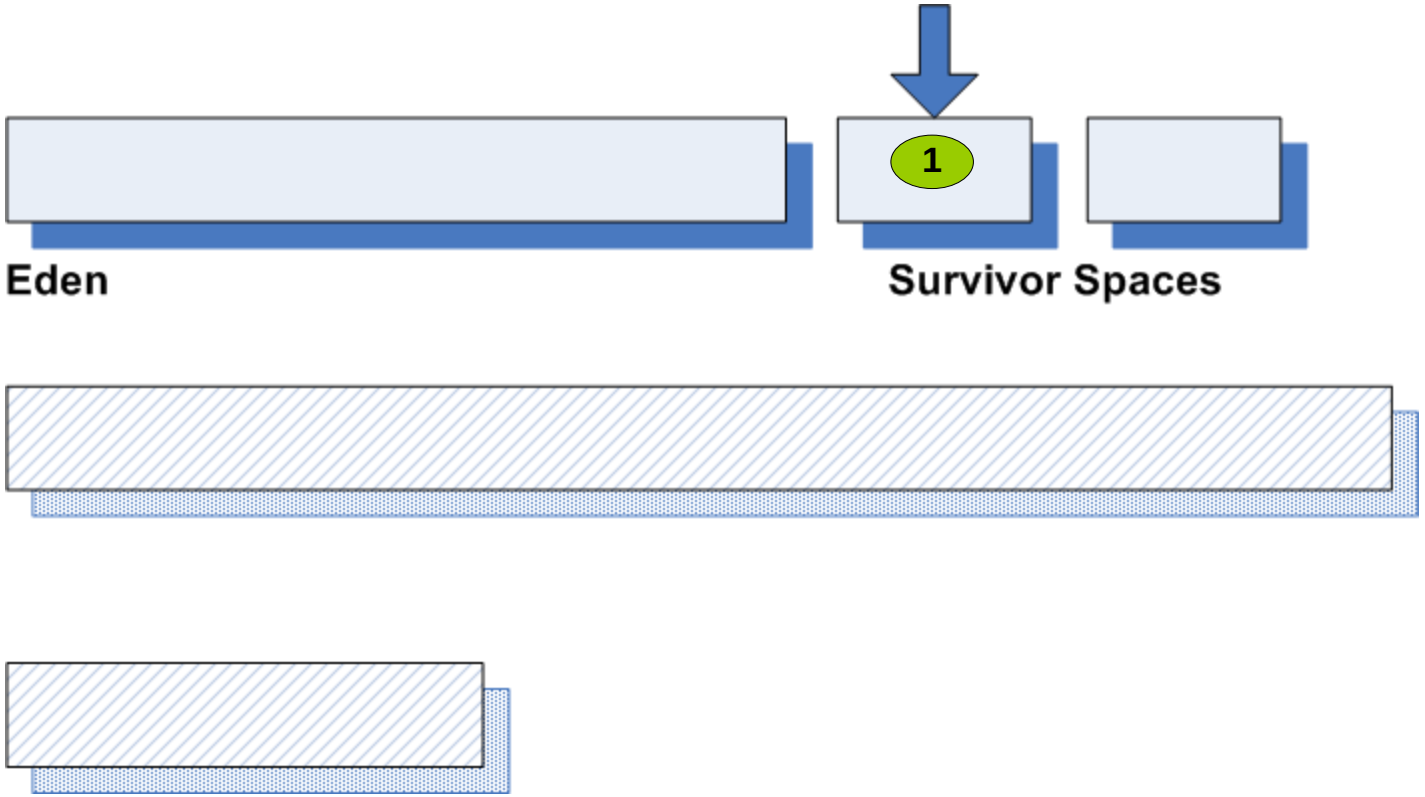
Hotspot Internals

Object lifecycle



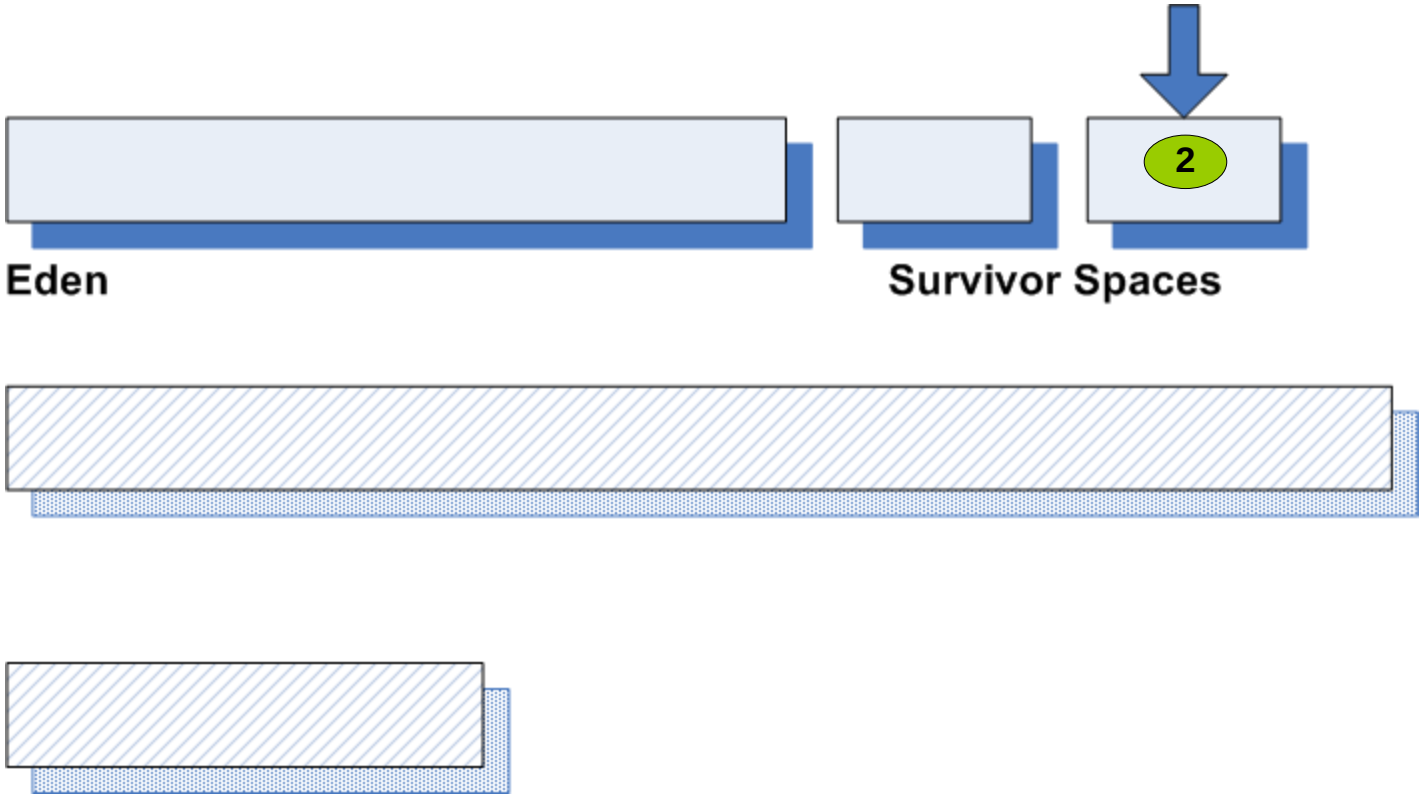
Hotspot Internals

Object lifecycle



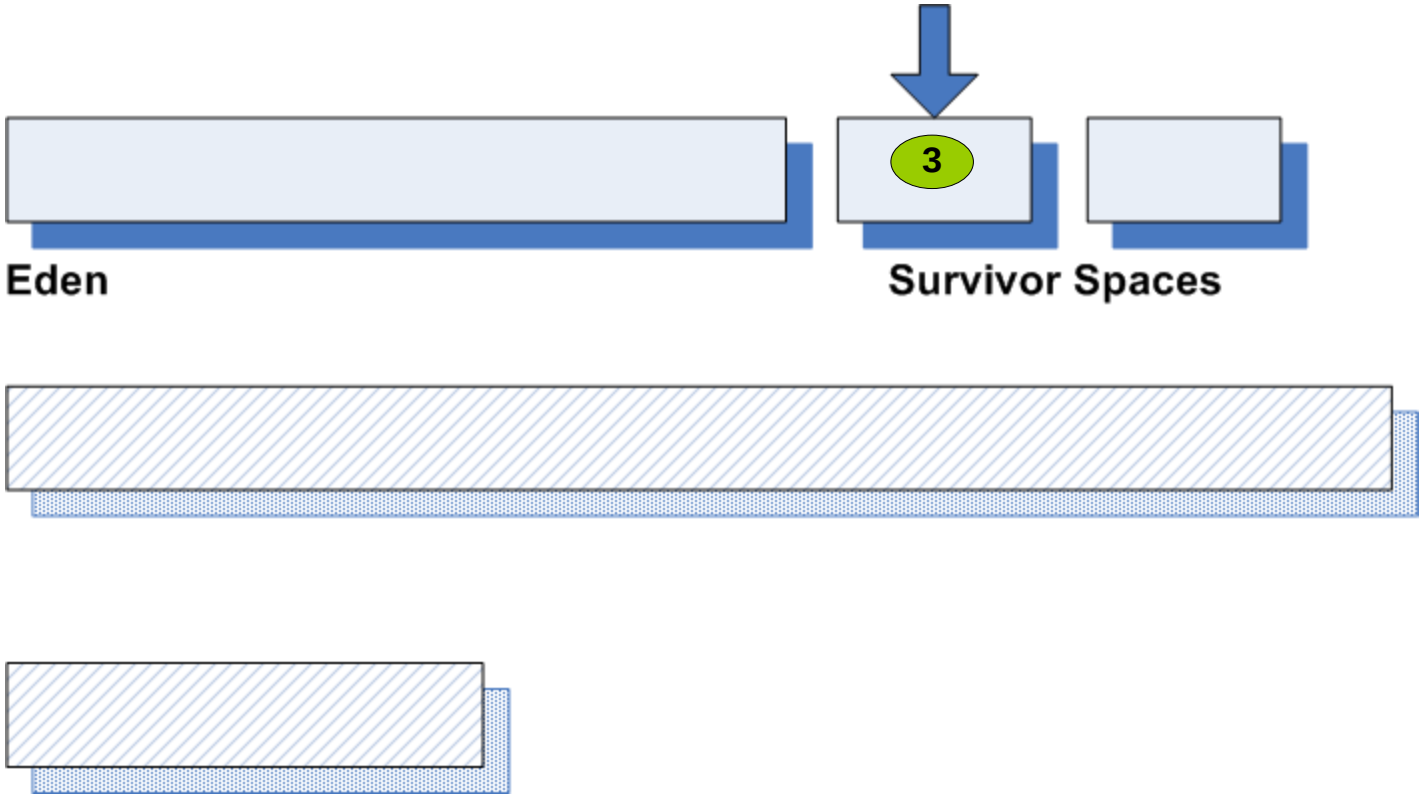
Hotspot Internals

Object lifecycle



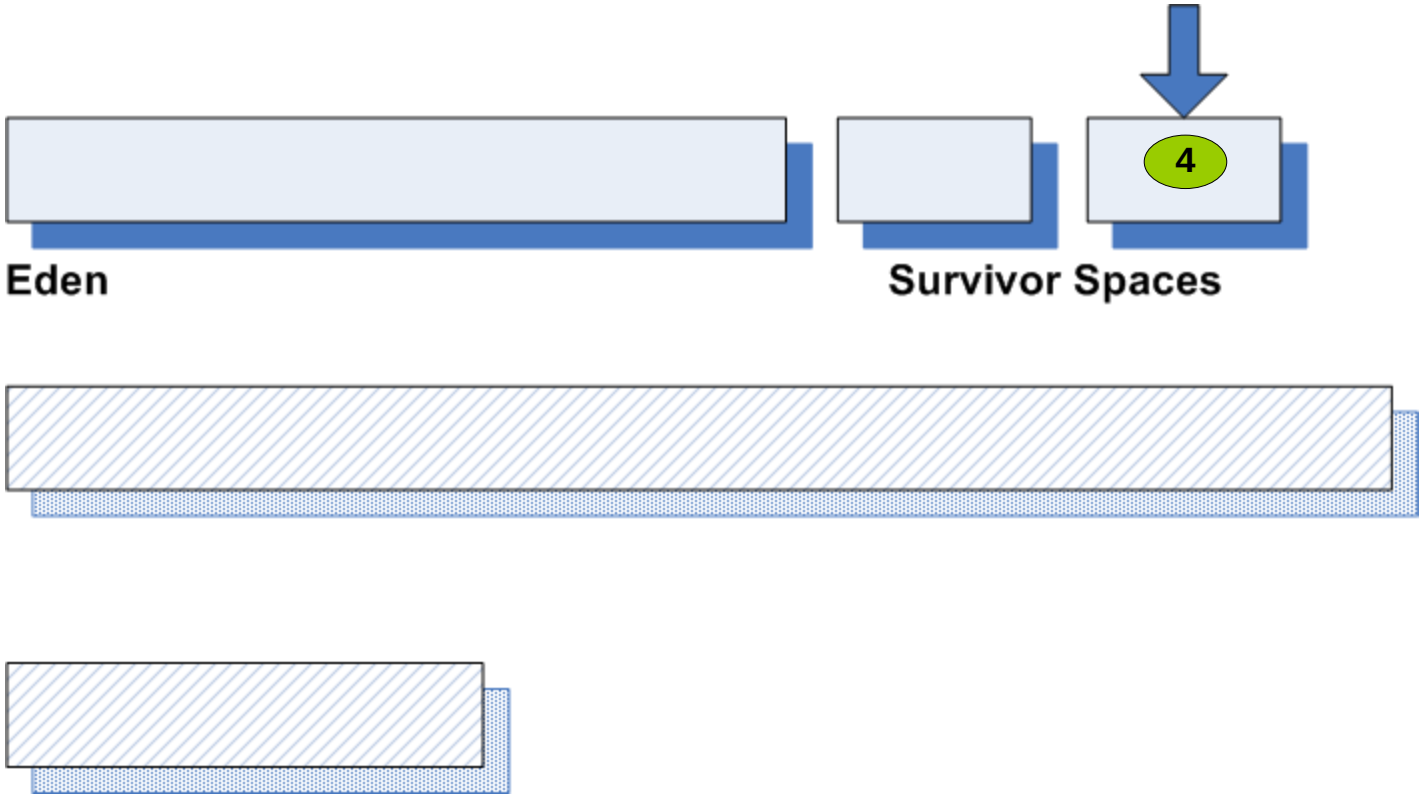
Hotspot Internals

Object lifecycle



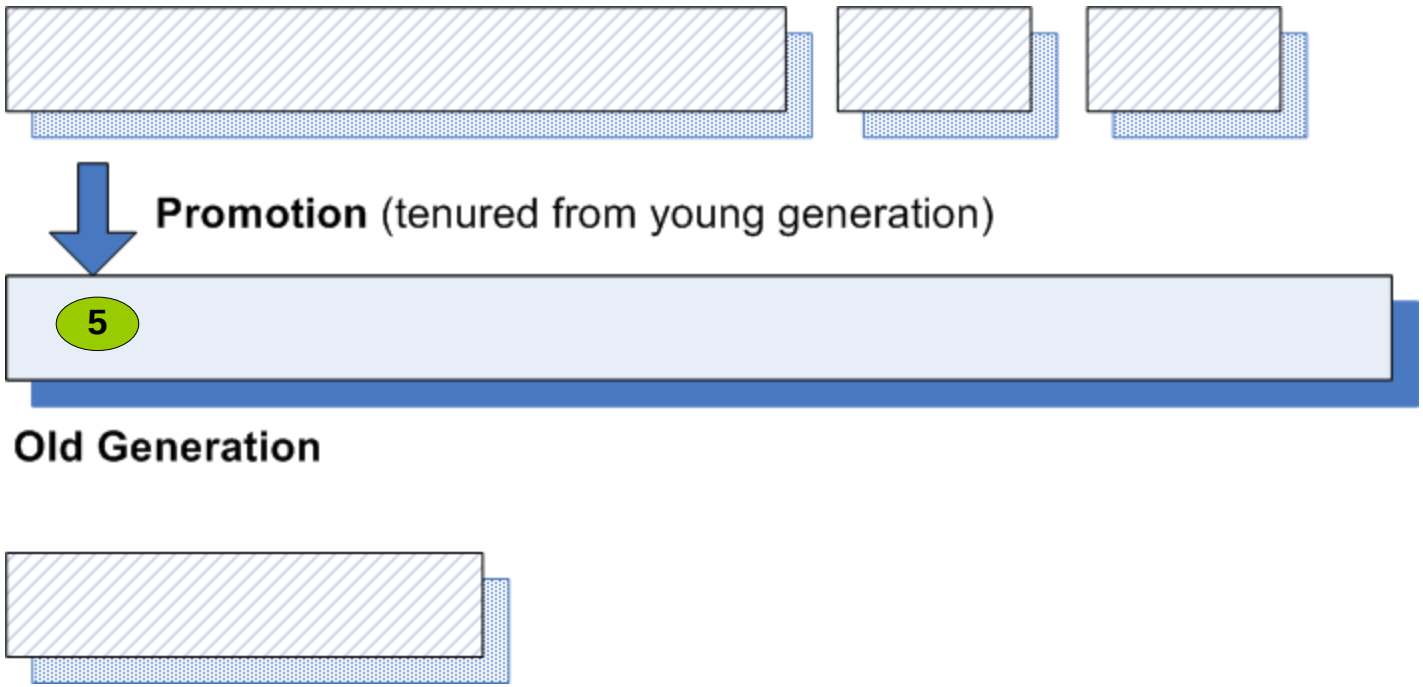
Hotspot Internals

Object lifecycle



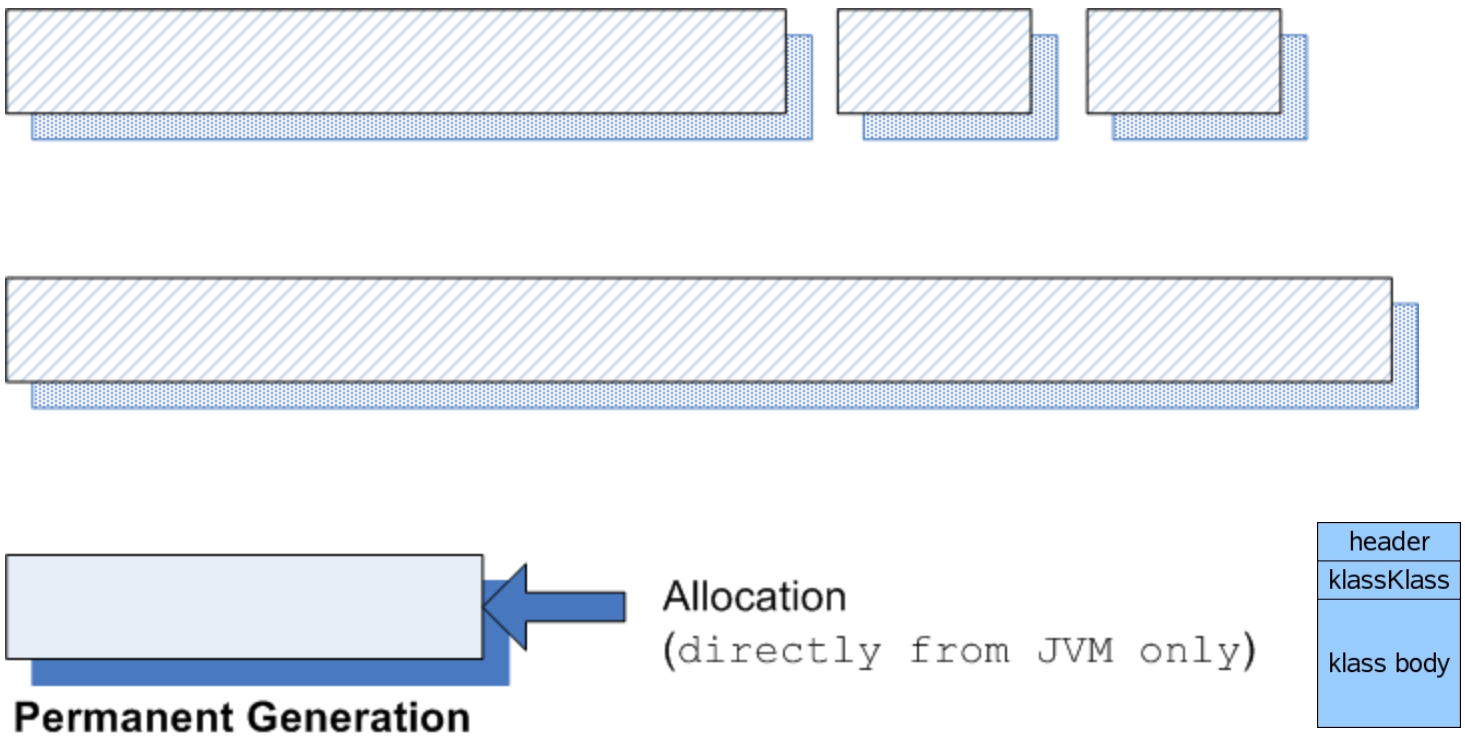
Hotspot Internals

Object lifecycle



Hotspot Internals

Object lifecycle

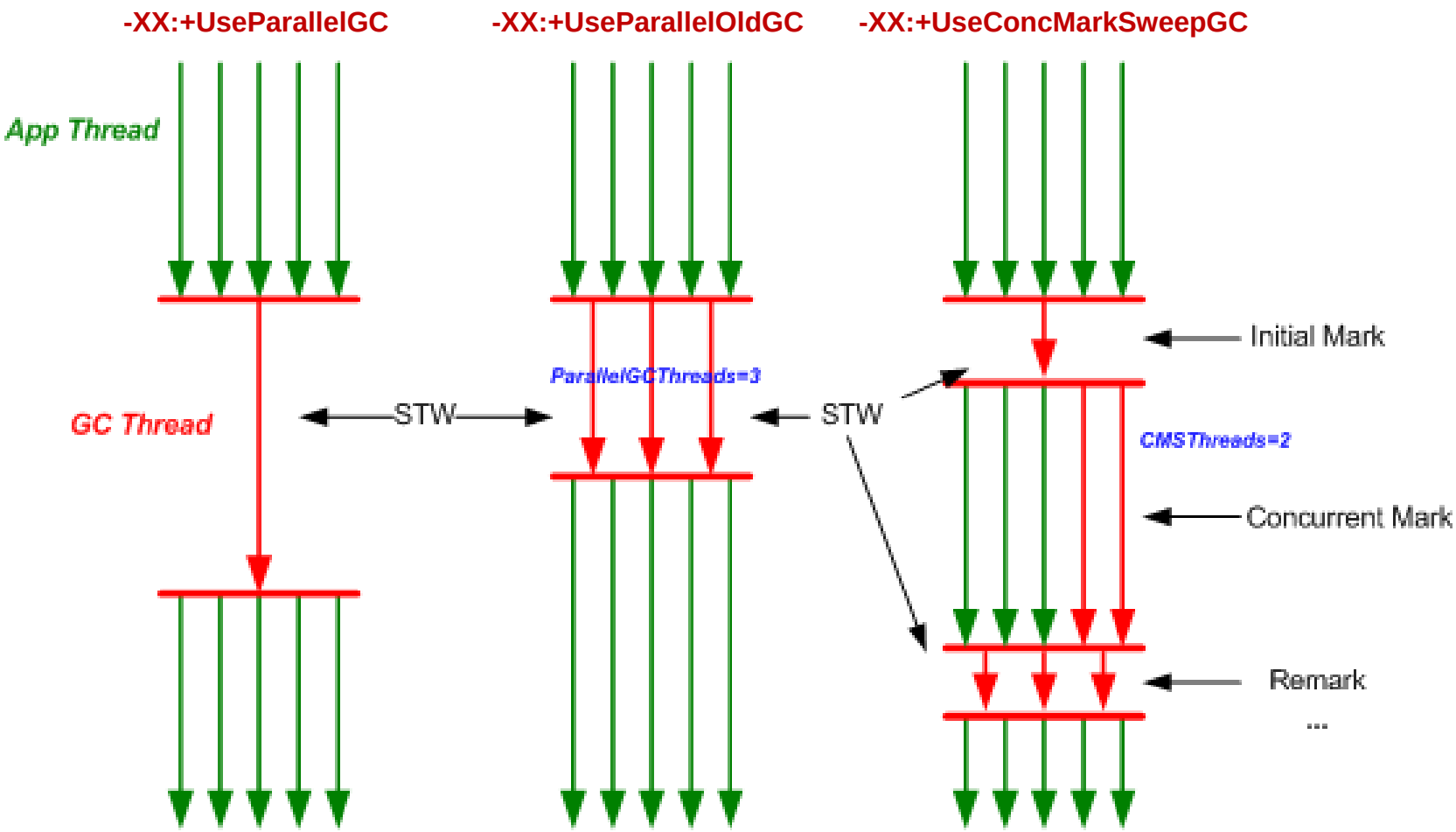


Agenda

- JVM Fundamentals
- JVM Performance Tuning
- GC Fundamentals
- Hotspot Internals
- Hotspot Tuning
- Diagnosing GC Issues

Hotspot Tuning

Garbage Collectors - Old Collection



Hotspot Tuning

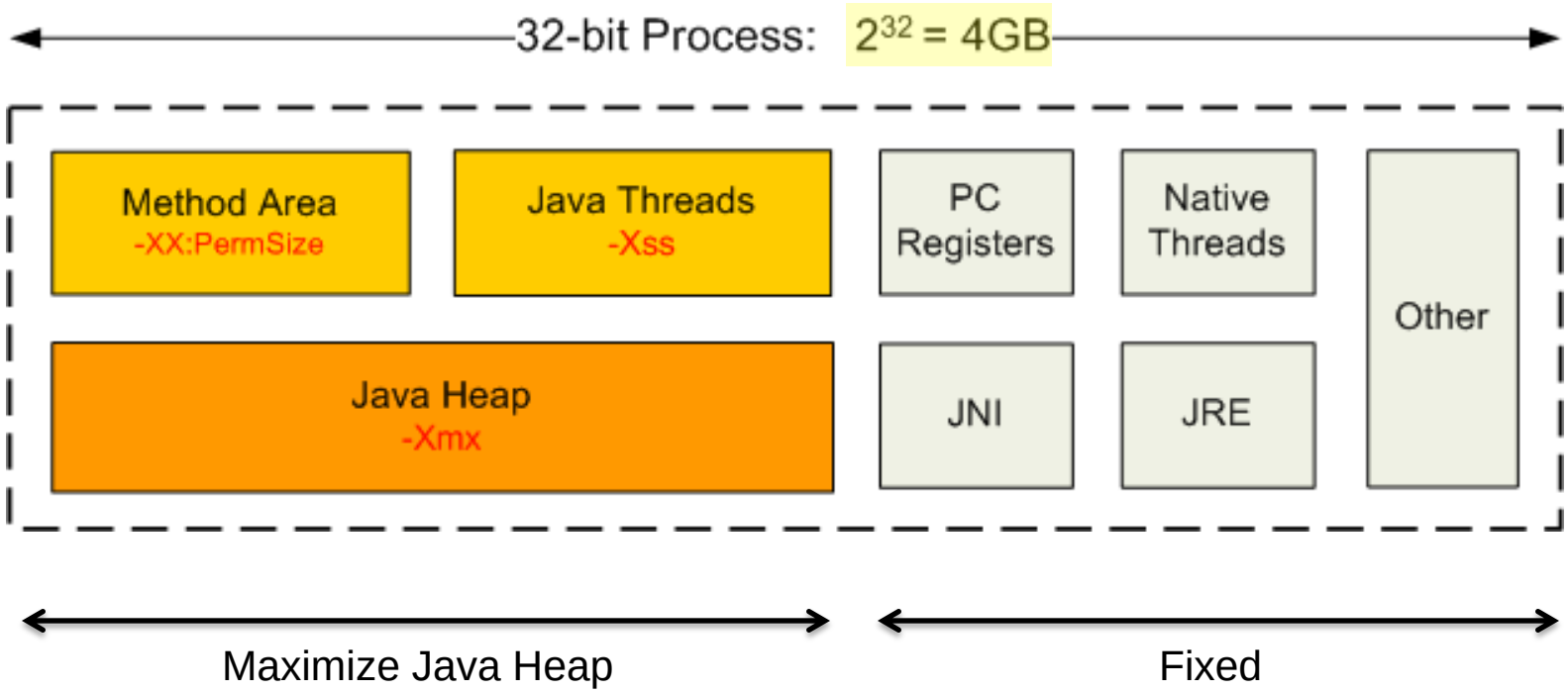
Garbage Collectors

Hotspot generational collectors

- Young Gen collection aka ‘minor’ collection
  - Uses copy collection (most efficient type)
- Old Gen collection aka ‘major’ collection
  - Requires ‘Full GC’ to compact fragmented heap space
  - Old collection can be
    - serial old (-XX:+UseParallelGC)
    - parallel old (-XX:+UseParallelOldGC)
    - concurrent (-XX:+UseConcMarkSweepGC)

Hotspot Tuning

Hotspot Native Process Space



Hotspot Tuning

32-bit vs. 64-bit

- 32-bit
  - For heap sizes up to 2.5G/3G or so
  - Reduce stack to maximize heap (-xss:128k)
- 64-bit with/without compressed references
  - -XX:+/-**UseCompressedOops** (default JDK6_18+)
  - Compressed references: 32GB Max (26GB best)
  - -Xmx: 26G (compressed) / unlimited (regular)
- 32-bit → 64-bit migration
  - Higher heap size requirements (around 20%)
  - Slight throughput impact (without compressed refs)
- 64-bit preferred for today's servers
  - Only option starting with Fusion Middleware 12c

ORACLE

Hotspot Tuning

Sizing Heap

- Young Generation size determines
  - ***Frequency*** of minor GC
  - ***Number of objects*** reclaimed in minor GC
- Old Generation Size
  - Should hold application's ***steady-state*** live data size
  - Try to ***minimize frequency*** of major GC's
- JVM footprint should not exceed physical memory
  - Max of 80-90% RAM (leave room for OS)
- **Thumb Rule:** Try to maximize objects reclaimed in young gen. Minimize Full GC frequency

Hotspot Tuning

Sizing Heap (cont'd)

- Resize of any generation requires Full GC
- Set –Xmx = –Xms
  - Prevents resizing (Full GC) to grow from Xms to Xmx
  - Better performance
  - Not always best for production availability (swapping preferred to OOME)
- Perm Size
  - *-XX:PermSize = -XX:MaxPermSize*
  - Perm Gen occupancy is hard to predict
  - Set high enough to prevent PermGen OOME
- Set -XX:NewSize = -XX:MaxNewSize
  - Using –Xmn instead preferred

Hotspot Tuning

Parallel GC Threads

- Number of parallel GC threads controlled by
  - *-XX:ParallelGCThreads=<num>*
  - Default value assumes 1 JVM
- Adjust *ParallelGCThreads* value for
  - Number of JVMs deployed on the system/virtual-machine
  - CPU chip architecture and cores e.g., Sun CMT, Intel Hyperthreads
- Example:
  - Exalogic compute node has 2 x 6-core Intel CPU's that have hyperthreading (2 threads per core), for 24 virtual CPU's.
  - If each node runs 4 WLS Instances
    - 24/4 = 6
    - Set *–XX:ParallelGCThreads* <= 6 as starting point per WLS JVM

Hotspot Tuning

CMS Collector Tuning

- Concurrent Mark Sweep (low pause) collector
  - *-XX:+UseConcMarkSweepGC*
- Pros:
  - Better worst-case latencies than Throughput Collector
- Cons:
  - Lower application throughput than Throughput Collector
  - Fragmentation - lengthier (albeit concurrent) GC cycles
- Increase old gen size by at least 20% to 30%
- Tune the young generation as described so far
- Need to be even more careful about avoiding premature promotion
  - Promotion in CMS is expensive
  - Causes more fragmentation
  - ***Full GC inevitable***

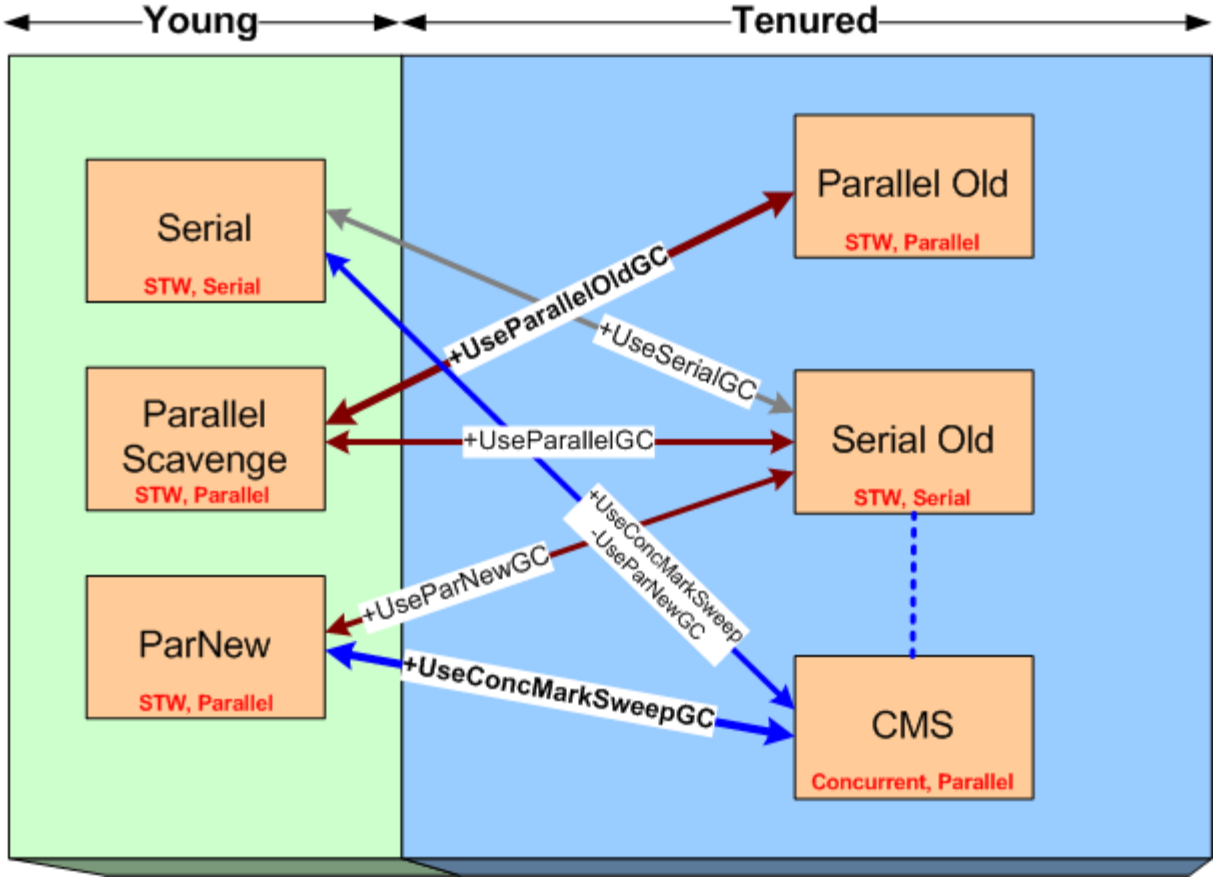
Hotspot Tuning

CMS Initiating Threshold

- Starting a CMS cycle too early
  - Frequent CMS cycles
- Starting a CMS cycle too late
  - Chance of an evacuation failure / Full GC
  - Safer to do earlier than later
- Default CMS initiating threshold
  - Computed dynamically
  - Almost always starts too late (CMS miss)
- To override default
  - **-XX:CMSInitiatingOccupancyFraction**=<percent> (e.g. 50)
    - Set low enough to prevent '*Concurrent Mode Failure*'
    - Always use with ***-XX:+UseCMSInitiatingOccupancyOnly***

Hotspot Tuning

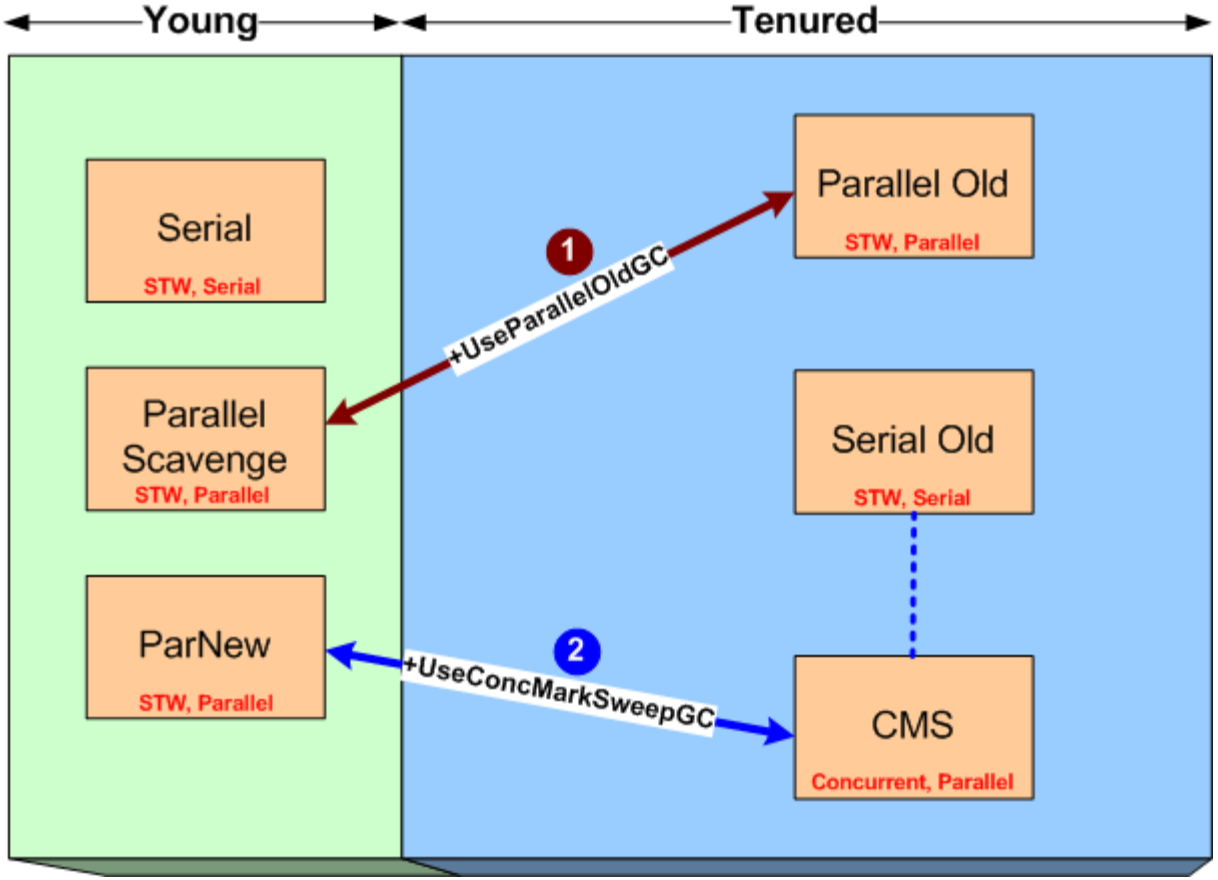
Valid GC combinations



Source: Jon Masamitsu's blog

Hotspot Tuning

GC Recommendations



Source: Jon Masamitsu's blog

Hotspot Tuning

Default GC Values – JDK6

Defaults	Recommended
-XX:+UseParallelGC	-XX:+UseParallel**Old**GC
ParallelGCThreads=CPU	ParallelGCThreads=CPU's**/JVM's**
SurvivorRatio=32	SurvivorRatio=**8**
PermSize=64M	PermSize=**MaxPermSize**
No GC logging	**Verbose** GC logging

Hotspot Tuning

Large Pages

- Use Large Pages for OS VM pages
  - Memory intensive applications
  - Applications using large objects
- Using Large Pages on Hotspot
  - **-XX:+UseLargePages**
    - On Solaris, ‘on’ by default
  - **-XX:LargePageSizeInBytes**=4m
    - Valid values differ by OS and chip architecture
    - E.g. max supported on x86 usually 2m

ORACLE

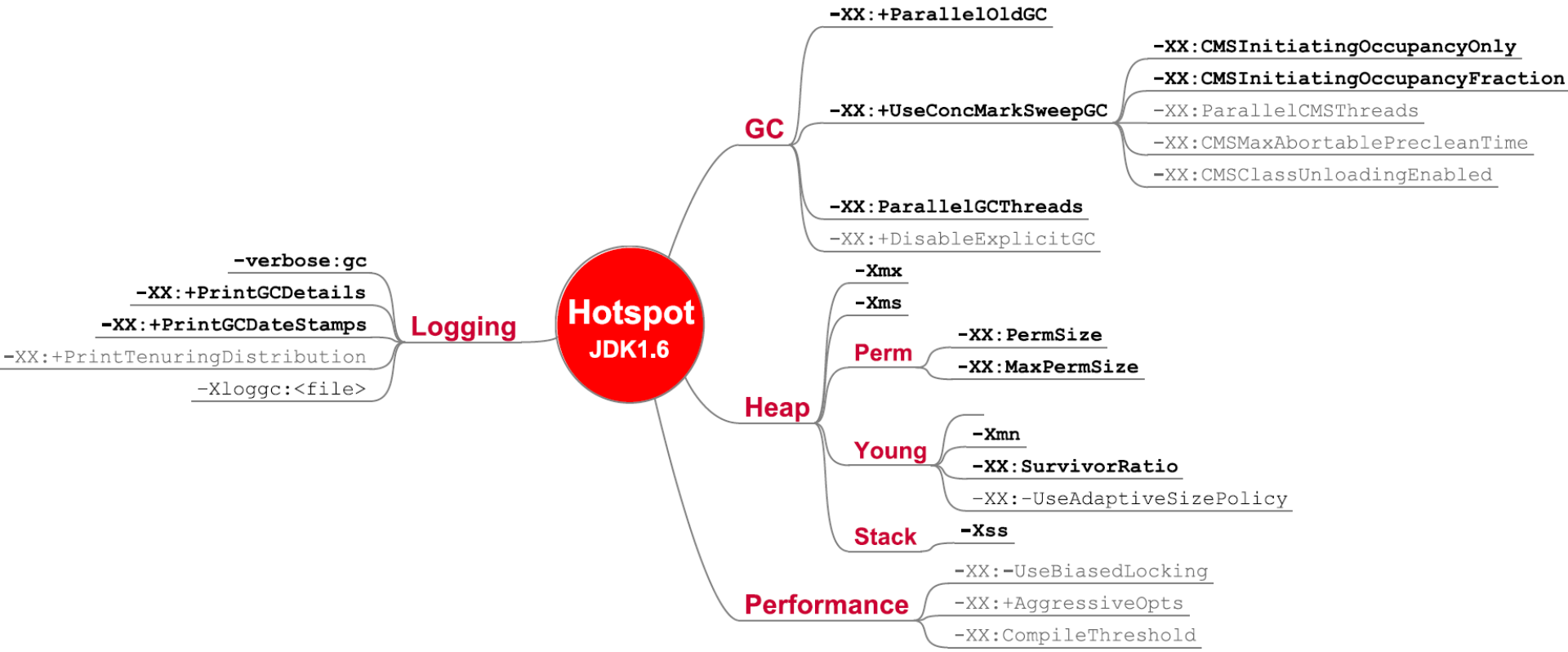
Hotspot Tuning

Large Pages Example

- Exalogic and Linux Huge Pages
  - Intel chip supports large pages (up to 2m)
  - Regular Page size = 2KB
  - Out-of-the-box Huge Page configuration on Exalogic
    - Huge Page size = 2MB
    - Number of Huge Pages = 10,000
    - Total 20GB reserved for Huge Pages
      - Regular pages get (96GB – 20GB)
    - Change '*vm.nr_hugepages*' to rebalance
- Use ***-XX:+UseLargePagesForHeap*** on JRockit
    - JVM heap now allocated on 20GB Huge Page Block
- Up to 15% performance improvement

Hotspot Tuning

Hotspot Tuning Cheat-sheet



Agenda

- JVM Fundamentals
- JVM Performance Tuning
- GC Fundamentals
- Hotspot Internals
- Hotspot Tuning
- Diagnosing GC Issues

Diagnosing GC Issues

Reading GC Logs - Minor GC

- GC logs are ***required*** for GC diagnosis

```
[GC[PSYoungGen: 99952K->14688K(109312K)]
  422212K->341136K(764672K), 0.0631991 secs]
  [Times: user=0.06 sys=0.00, real=0.06 secs]
```

- 'PSYoungGen' indicates this is a minor collection
  - Throughput collector used (e.g. UseParallelGC)
- Numbers mean *SIZE_BEFORE_GC->SIZE_AFTER_GC(MAX_SIZE)*
  - STW pause = 0.06 secs ('real' time)
- Similar minor collection log for CMS would show 'ParNew'

```
[GC[ParNew: 99952K->14688K(109312K)]
  422212K->341136K(764672K), 0.0631991 secs]
  [Times: user=0.06 sys=0.00, real=0.06 secs]
```

Diagnosing GC Issues

Reading GC Logs - Full GC

- Throughput collector ‘Full GC’

```
[Full GC[PSYoungGen: 11456K->0K(110400K)]
   [ParOldGen: 651536K->58466K(655360K)]
    662992K->58466K(765760K)
   [PSPermGen: 10191K->10191K(22528K)], 1.1178951
   secs[Times: user=1.01 sys=0.00, real=1.12 secs]
```

- CMS collector ‘Full GC’

```
[Full GC 59.550: [CMS59.608: [CMS-concurrent-sweep:
0.189/0.191 secs] [Times: user=0.37 sys=0.00, real=0.19
secs]
(concurrent mode failure): 1048575K->1048575K(1048576K),
3.4256231 secs] 2936061K->2206984K(2936064K), [CMS Perm
: 2621K->2621K(524288K)], 3.4263668 secs]
[Times: user=3.35 sys=0.00, real=3.42 secs]
```

Diagnosing GC Issues

GC log analysis

- Full GC usual cause for long pauses
- Search for ‘Full GC’ in verbose GC logs

Full GC in log	Action
Heap dump taken?	Ignore this Full GC (triggered by heap dump)
Shows string “Full GC (system)”	Set *-XX:+DisableExplicitGC*
Shows Perm full	Increase *–XX:MaxPermSize*
Shows heap resized	Set *–Xmx=-Xms, -XX:NewSIze=-XX:MaxNewSIze*
Shows ‘concurrent mode failure’	*Reduce -XX:CMSInitiatingOccupancyFraction* *OR* *Forgot to use –XX:+CMSInitiatingOccupancyOnly*

ORACLE

Full GC

```
20416.613: [CMS-concurrent-sweep-start]
20420.628: [CMS-concurrent-sweep: 4.004/4.015 secs]
20420.628: [CMS-concurrent-reset-start]
20420.892: [CMS-concurrent-reset: 0.264/0.264 secs]
20422.176: [Full GC 20422.177: [CMS (concurrent mode failure): 1815018K-
>912719K(1835008K), 18.2639275 secs] 1442583K->912719K(2523136K), [CMS
Perm : 202143K->142668K(262144K)], 18.2649505 secs]
```

- Heap exhausted prior to CMS completion
- Dynamically adjusted CMS Initiation Occupancy incorrect
- Manually specify a more conservative initiation occupancy
  - -XX:+UseCMSInitiatingOccupancyOnly
  - -XX:CMSInitiatingPermOccupancyFraction=<percent>

Full GC

```
429417.135: [GC 429417.135: [ParNew: 1500203K->100069K(1747648K),
0.3973722 secs] 3335352K->1935669K(3844800K), 0.3980262 secs] [Times:
user=0.85 sys=0.00, real=0.40 secs]
430832.180: [GC 430832.181: [ParNew: 1498213K->103052K(1747648K),
0.3895718 secs] 3333813K->1939101K(3844800K), 0.3902314 secs] [Times:
user=0.83 sys=0.01, real=0.39 secs]
431370.238: [Full GC 431370.238: [CMS: 1836048K->1808511K(2097152K),
43.4328330 secs] 2481043K->1808511K(3844800K), [CMS Perm : 524287K-
>475625K(524288K)], 43.4336938 secs] [Times: user=40.13 sys=0.73,
real=43.43 secs]
```

- Full GC caused by PermGen exhaustion
  - Old gen is not close to full, but Full GC triggered
  - Perm was full before Full GC
- Resolution:
  - Increase *–XX:MaxPermSize*
  - Use *–XX:+CMSClassUnloadingEnabled*

Full GC

```
39195.195: [Full GC (System) 39195.195: [CMS: 641844K-
>617525K(1318912K), 27.0243921 secs] 751876K->617525K(1698624K), [CMS
Perm : 205856K->205495K(421888K)], 27.0250058 secs] [Times: user=69.89
sys=0.05, real=27.03 secs]
39222.431: [Full GC (System) 39222.431: [CMS: 617525K-
>612104K(1318912K), 25.8235298 secs] 639071K->612104K(1698624K), [CMS
Perm : 205498K->205495K(421888K)], 25.8240855 secs] [Times: user=51.70
sys=0.02, real=25.82 secs]
```

- Back to back Full GC
- Full GC has '(System)' in log
- Resolution:
  - Have customer remove `system.gc()` in code
  - Add '*-XX:+DisableExplicitGC*' flag

Full GC

```
296.544: [GC [PSYoungGen: 736K->64K(832K)] 96847K->96191K(1023808K), 0.0013899
    secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
296.546: [GC [PSYoungGen: 703K->64K(832K)] 96831K->96207K(1023808K), 0.0007021
    secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
296.547: [GC [PSYoungGen: 101K->32K(832K)] 96244K->96199K(1023808K), 0.0005676
    secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
296.548: [Full GC [PSYoungGen: 32K->0K(832K)] [PSOldGen: 96167K-
    >56751K(1022976K)] 96199K->56751K(1023808K) [PSPermGen: 8115K->8115K(51200K)],
    0.0189618 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]
```

- Normal Full GC log with no apparent trigger (lot of heap left on each generation)
  - User could have triggered a heap dump
  - Heap dump causes a full GC prior to dumping file

Full GC Pause - External Factors

```
[Full GC 957910K->747933K(1004928K), 0.0077580 secs]
[Full GC 959079K->747525K(1004928K), 0.0069880 secs]
[Full GC 959014K->748193K(1004928K), 4.8153540 secs]
[Full GC 916083K->697827K(1004928K), 14.8503310 secs]
[Full GC 831689K->657890K(1008320K), 14.5647330 secs]
[Full GC 893862K->688939K(1004928K), 7.4950890 secs]
[Full GC 385884K->240312K(1004928K), 91.2939710 secs]
```

- Heap size is only 1G
- Full GC time of 91s extremely high for this size heap
- Inspect OS for thrashing
  - Memory: Excessive Paging
  - CPU: ~100%

Questions?