淺談 Java GC 原理、調教和新發展

Leon Chen
陳逸嘉

Java Architect
文曄科技

Java™

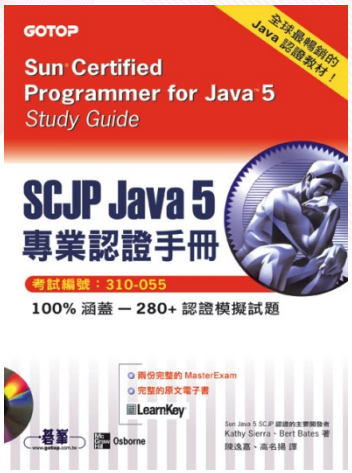
ORACLE APPROVED EDUCATION CENTER

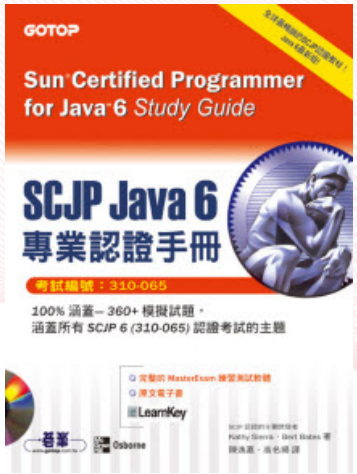
陳逸嘉 (Leon Chen)

- Master of NTHU CS
- More than 15 years in Java/JEE, as programmer, architect, developer leader and consultant, in finance/telecom domain.
  - 工研院
  - 得捷
  - 新光人壽
  - Ericsson
  - Oracle
  - 文曄科技 (WT Microelectronics)

著作

- TW Patent 182927, 一種文章切割方法, 2003, 陳逸嘉, 林一中
- TW Patent 206819, 自動服務組合方法及系統, 2004, 陳逸嘉, 許維德, 洪鵬翔
- US Patent 7,617,174, Method and system for automatic service composition, 2009, Leon Chen, Wei-Tek Hsu, Peng-Hsiang Hung

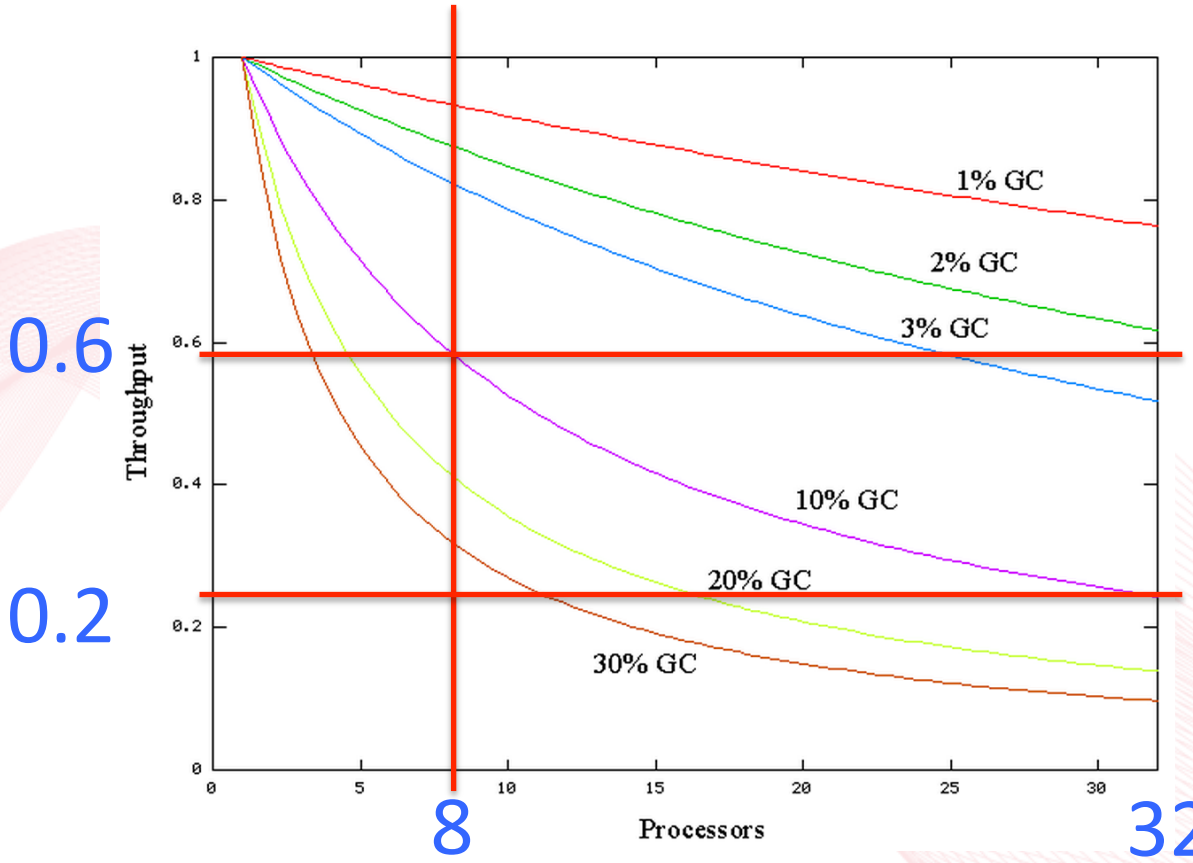




Garbage Collection

我真的需要花時間去 Tune GC 嗎?

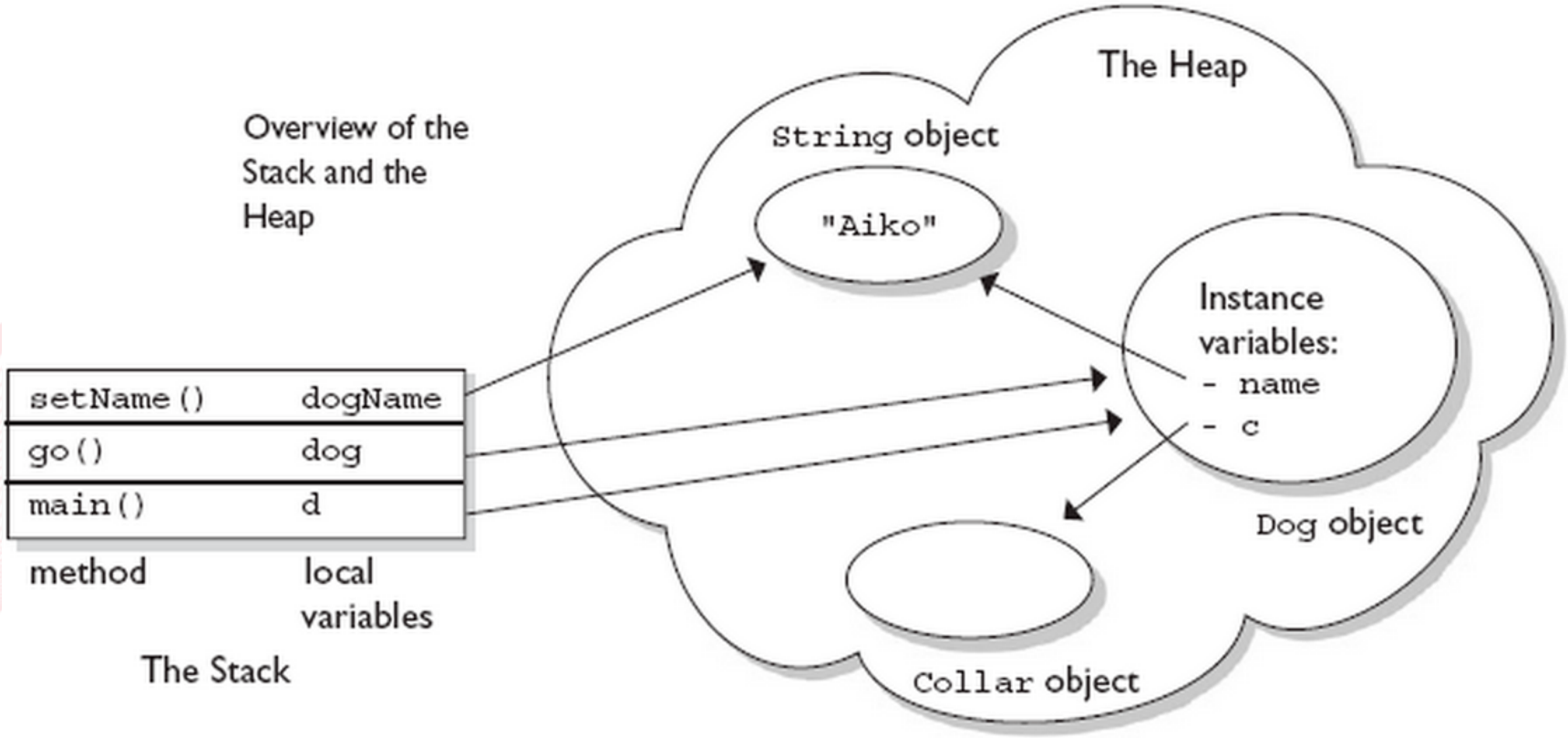
Performance Impact of GC



Java SE 6 HotSpot[tm] Virtual Machine Garbage Collection Tuning
http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html

有資格被回收的物件?

Java Memory Management

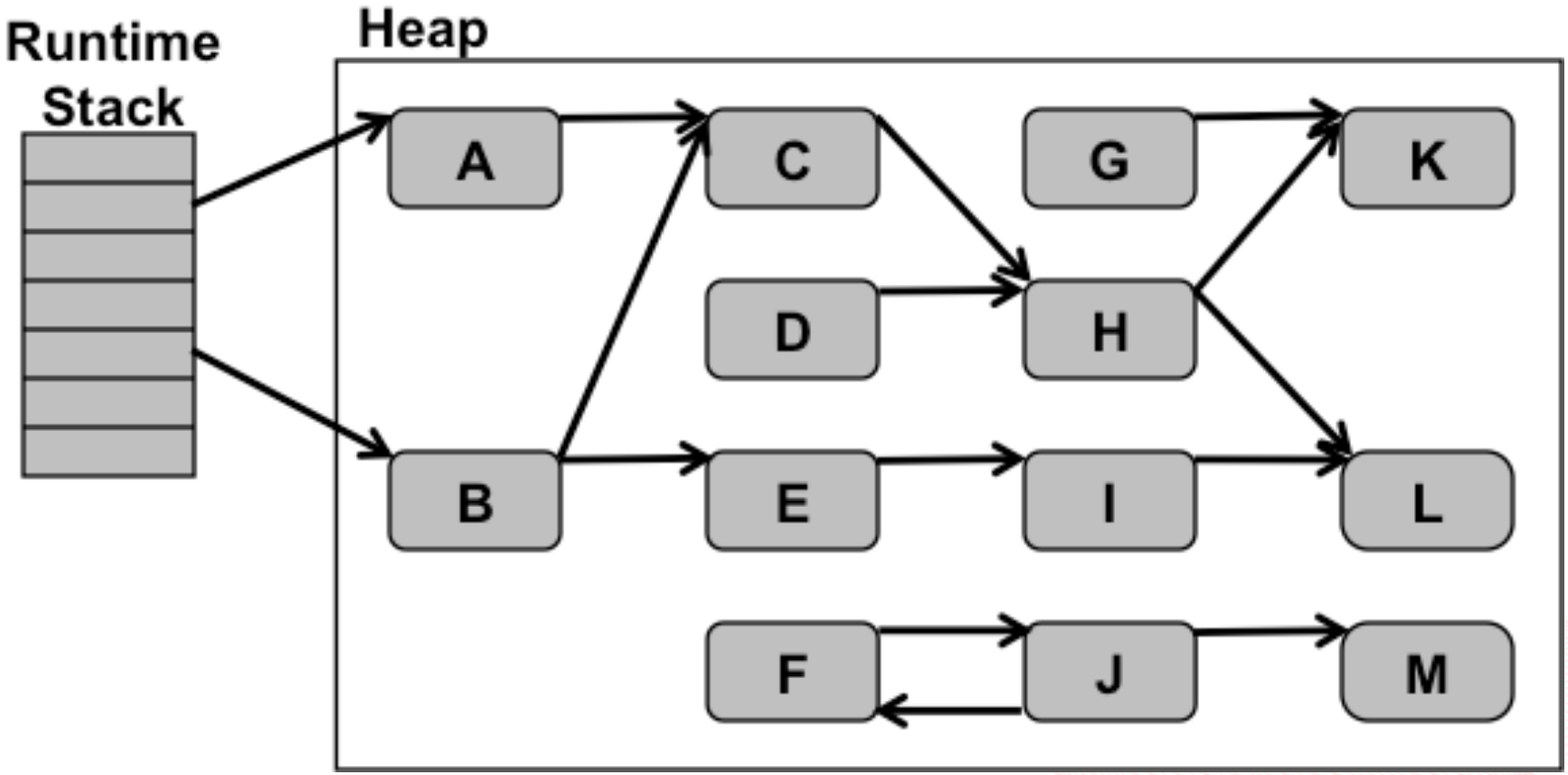


Java Memory Management

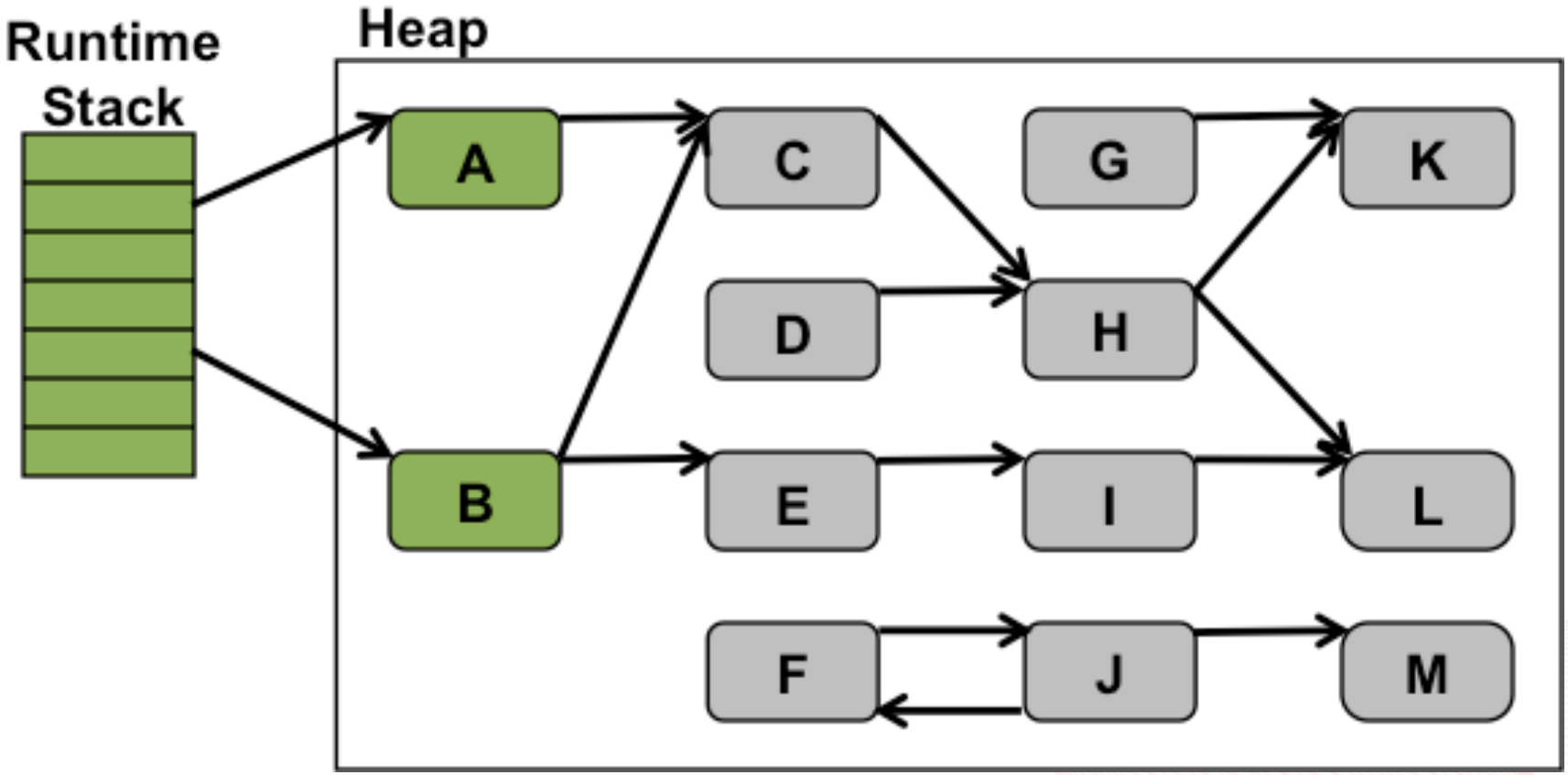
GURU99.COM

https://www.youtube.com/watch?v=450ma1zSIvA

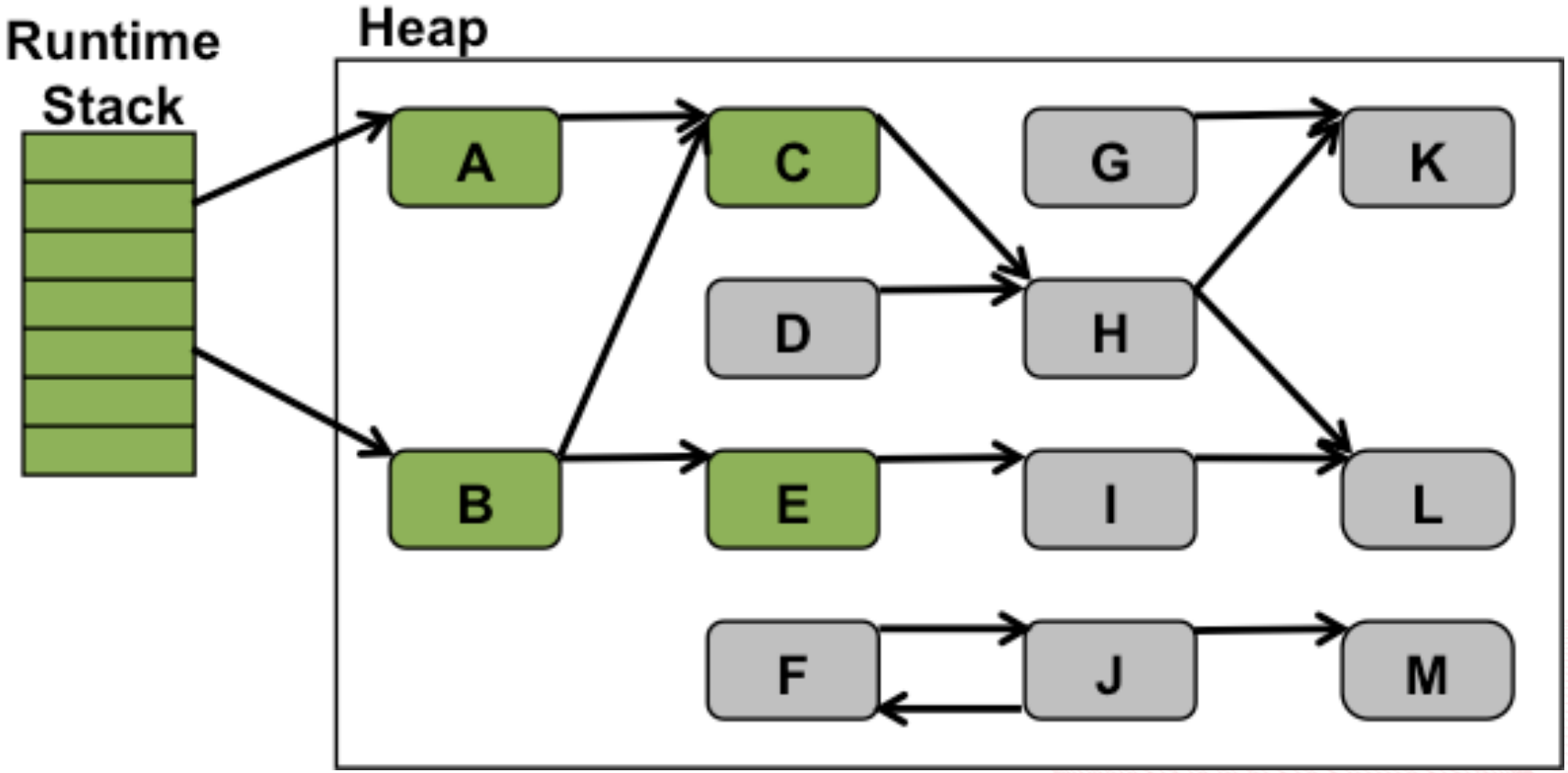
Tracing GC Example



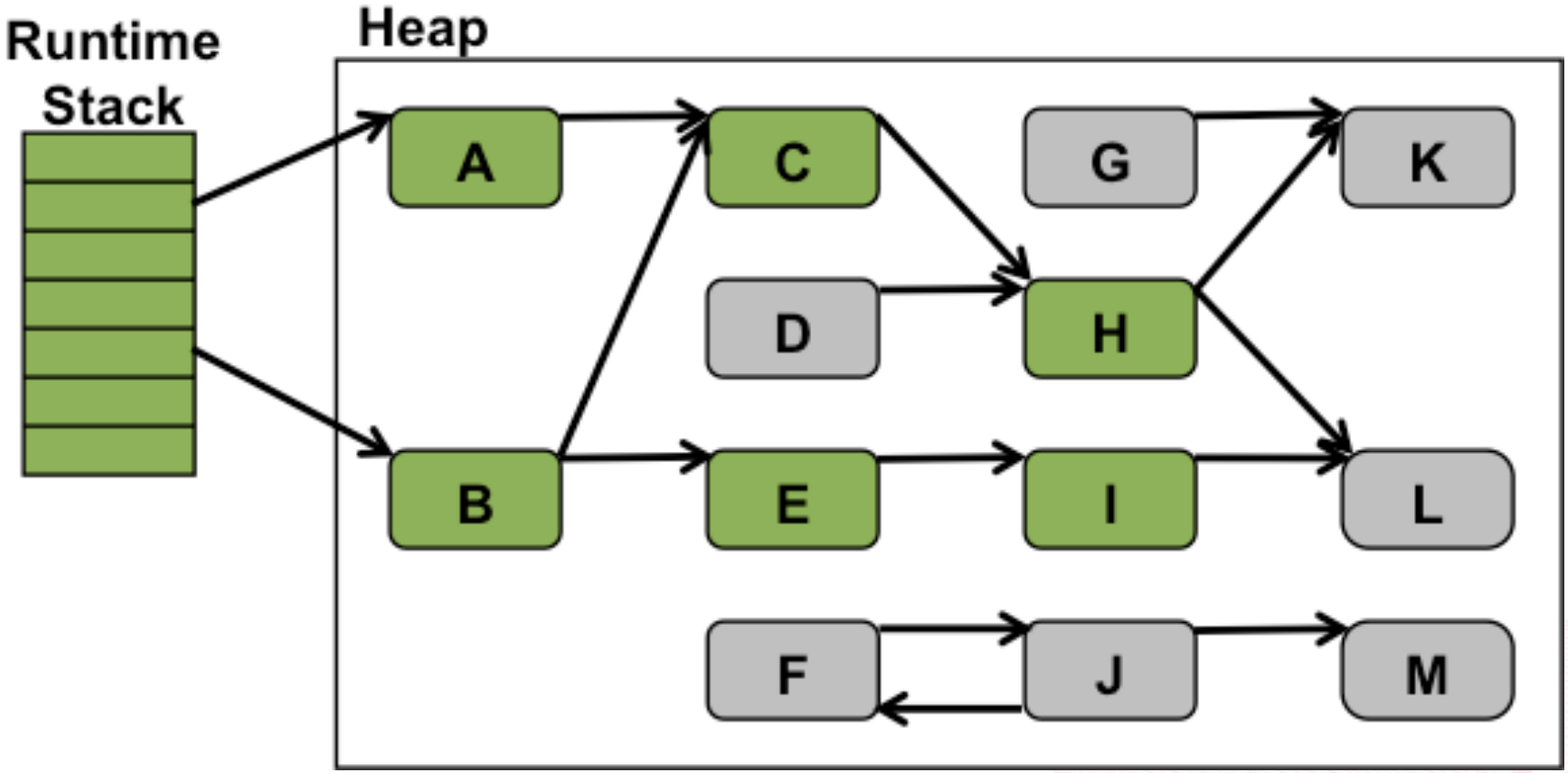
Tracing GC Example



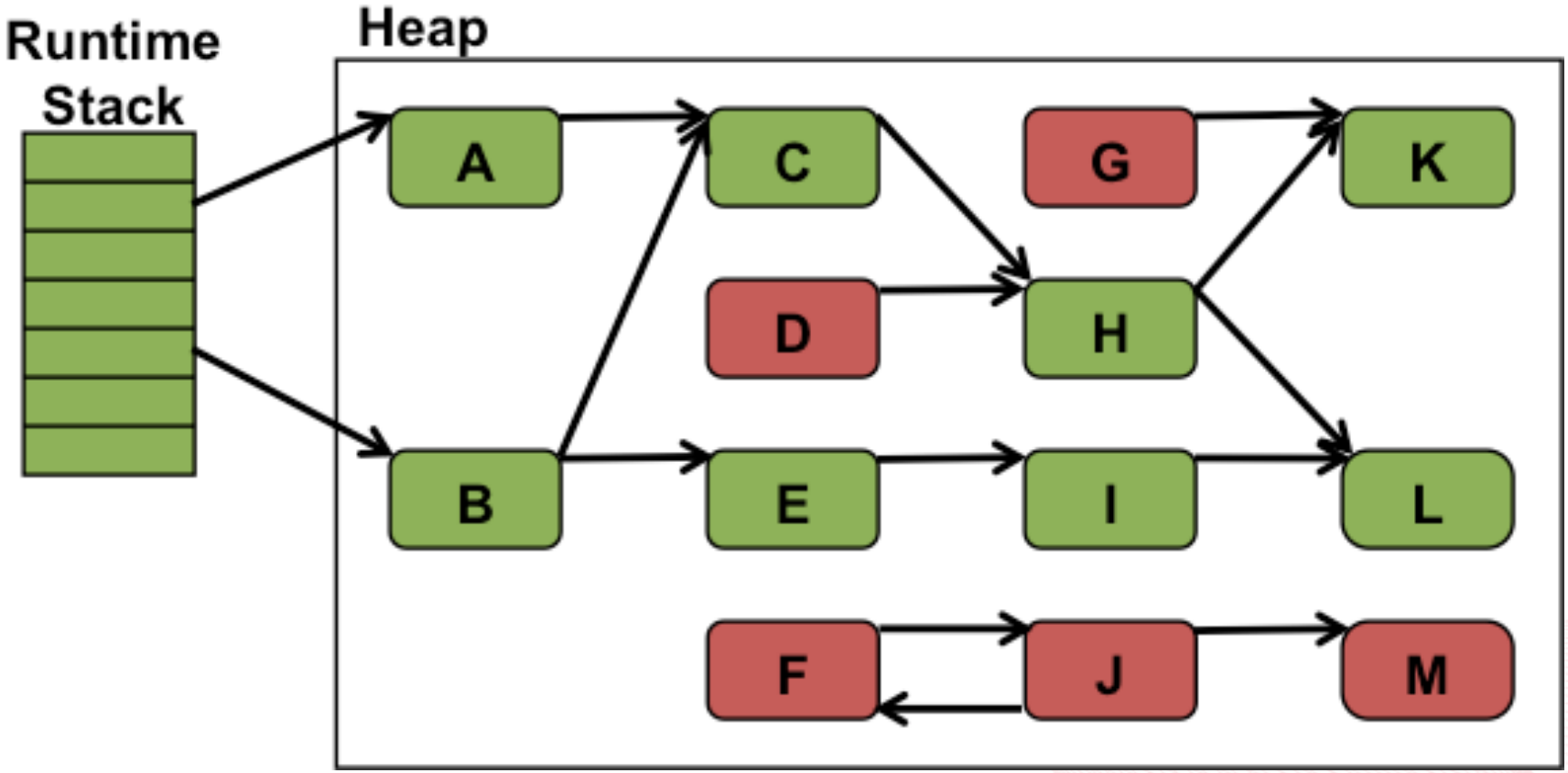
Tracing GC Example



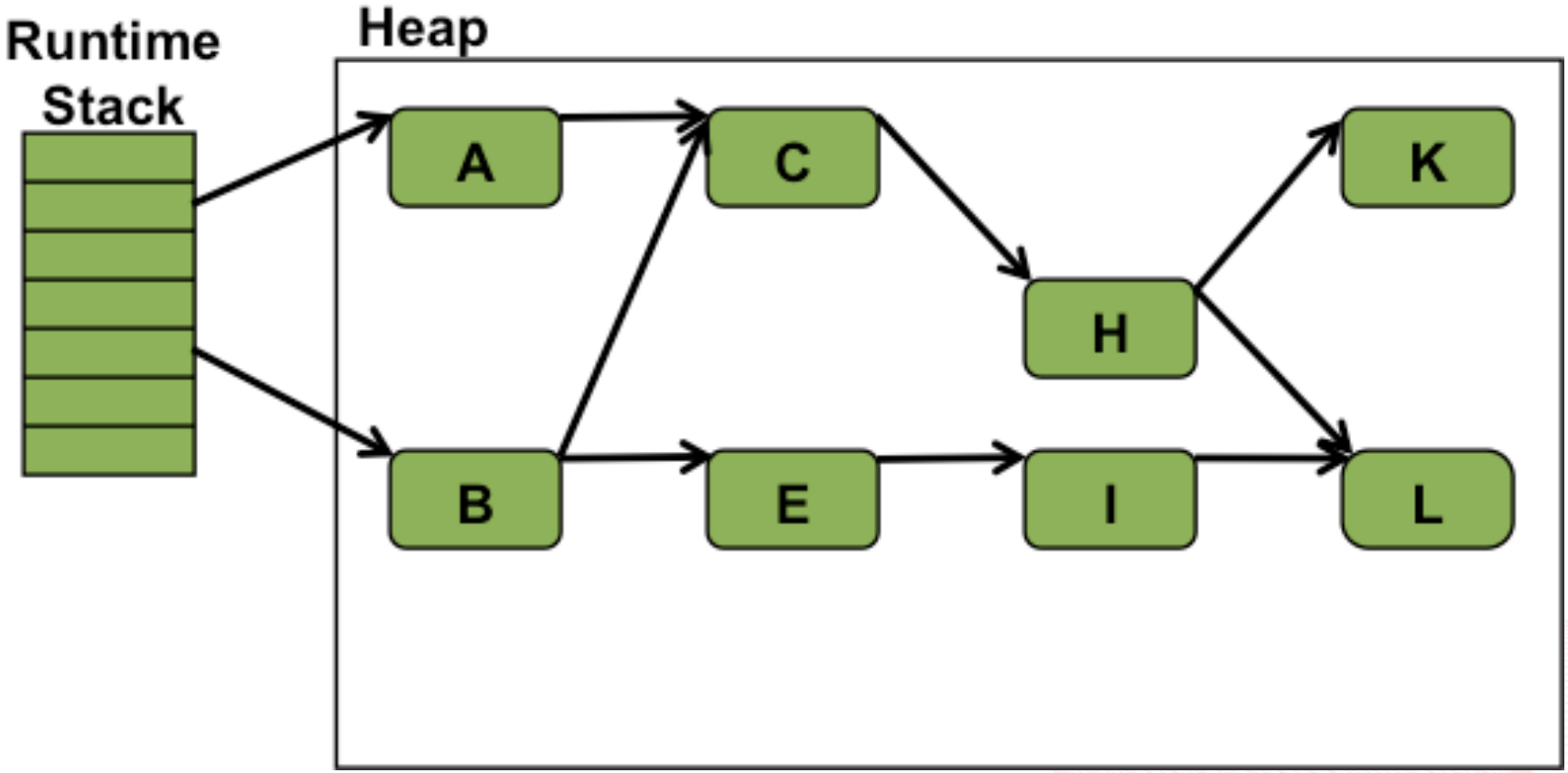
Tracing GC Example



Tracing GC Example

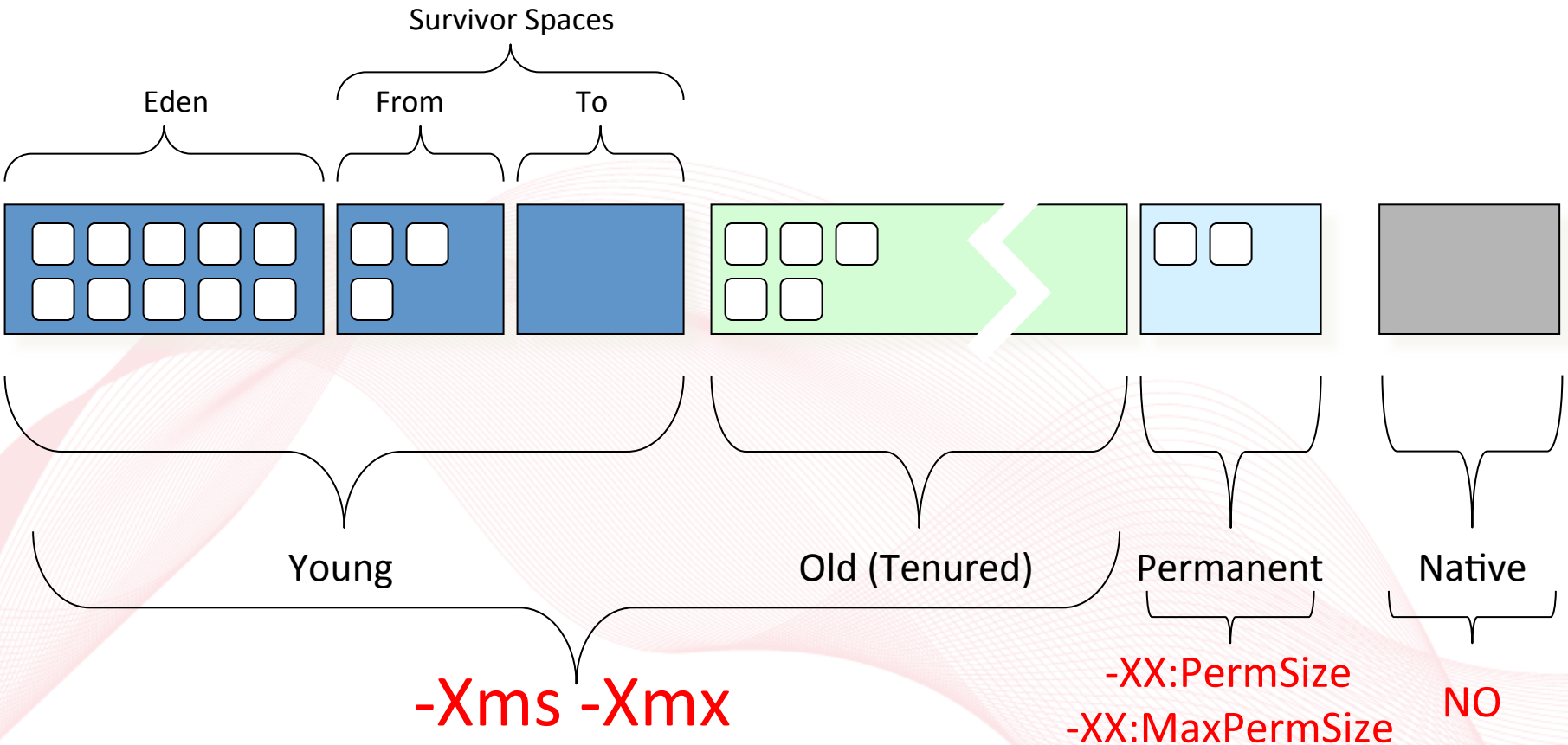


Tracing GC Example



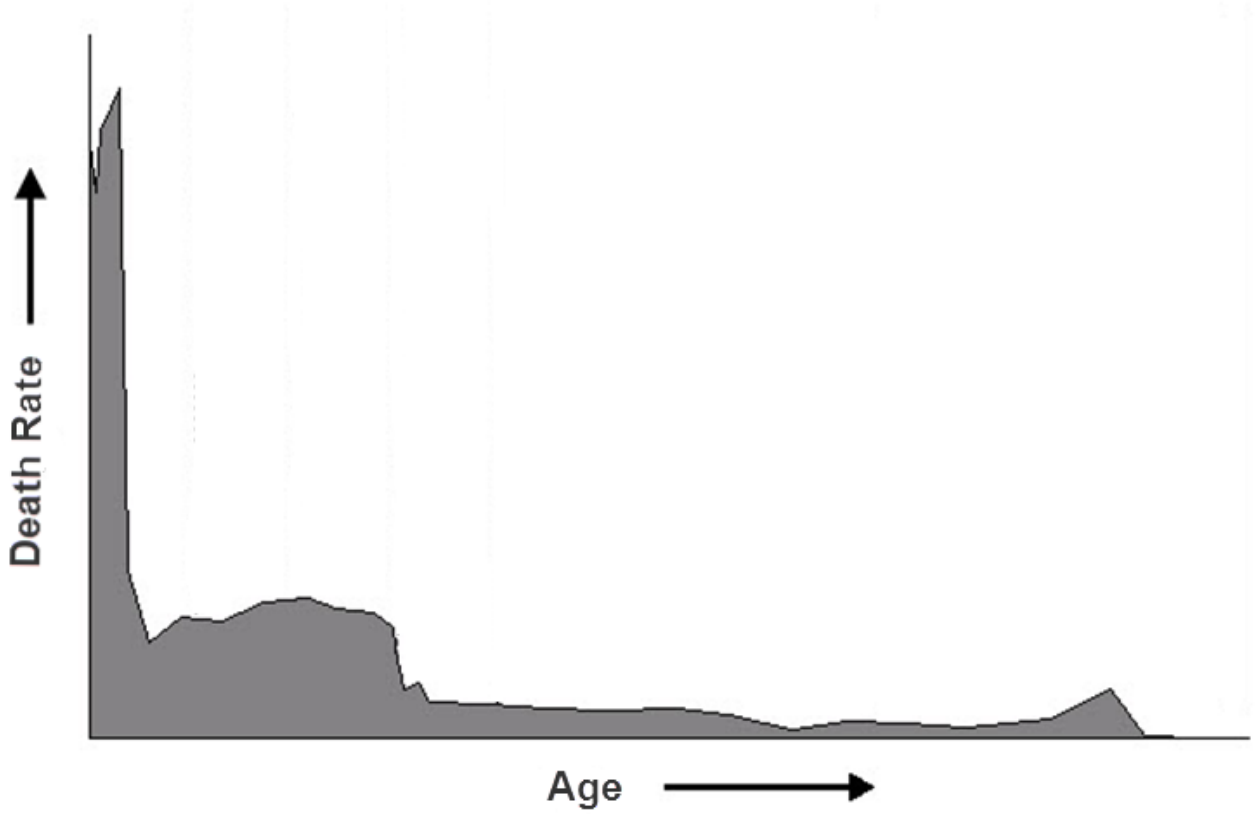
JAVA HEAP STRUCTURE AND GC

Hotspot JVM Heap Layout



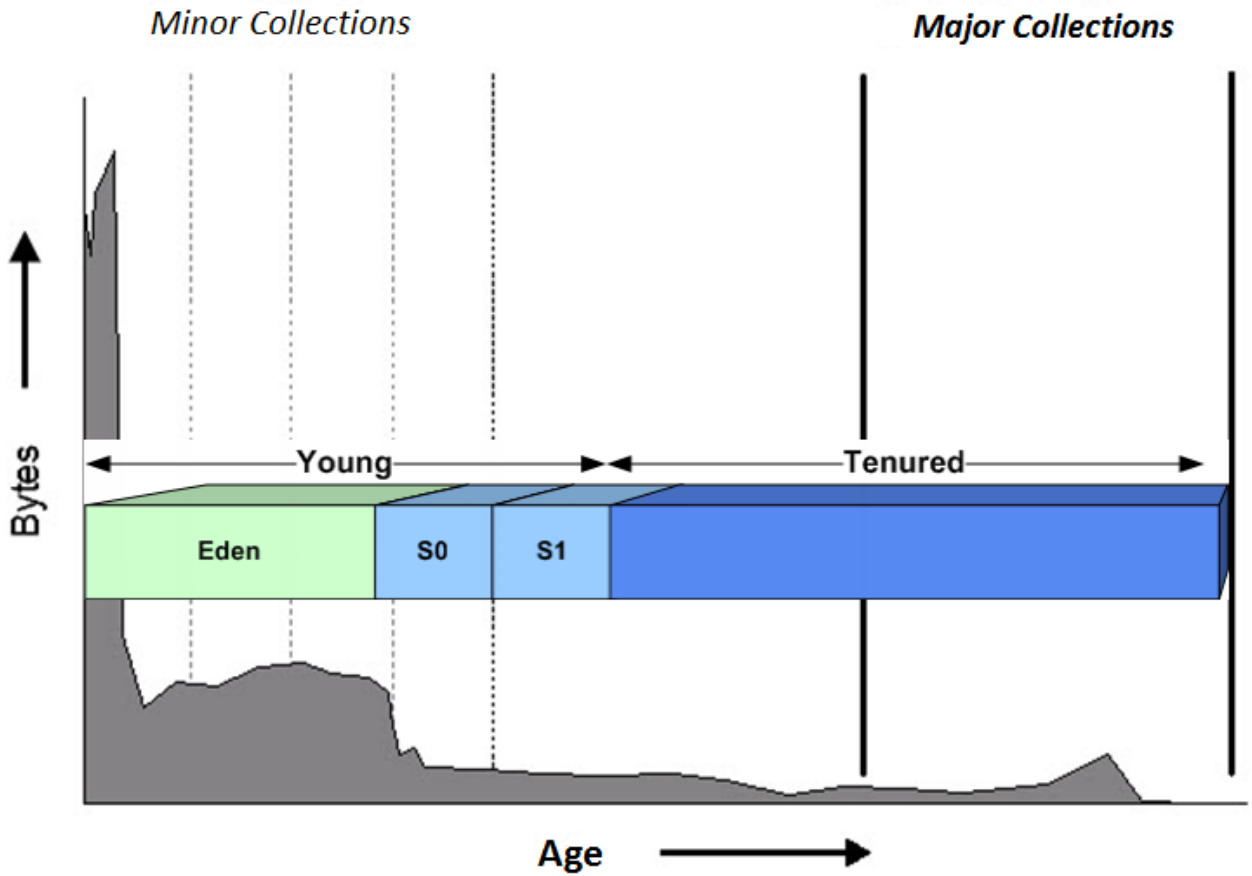
GC Fundamentals

Garbage Distribution – Objects die young

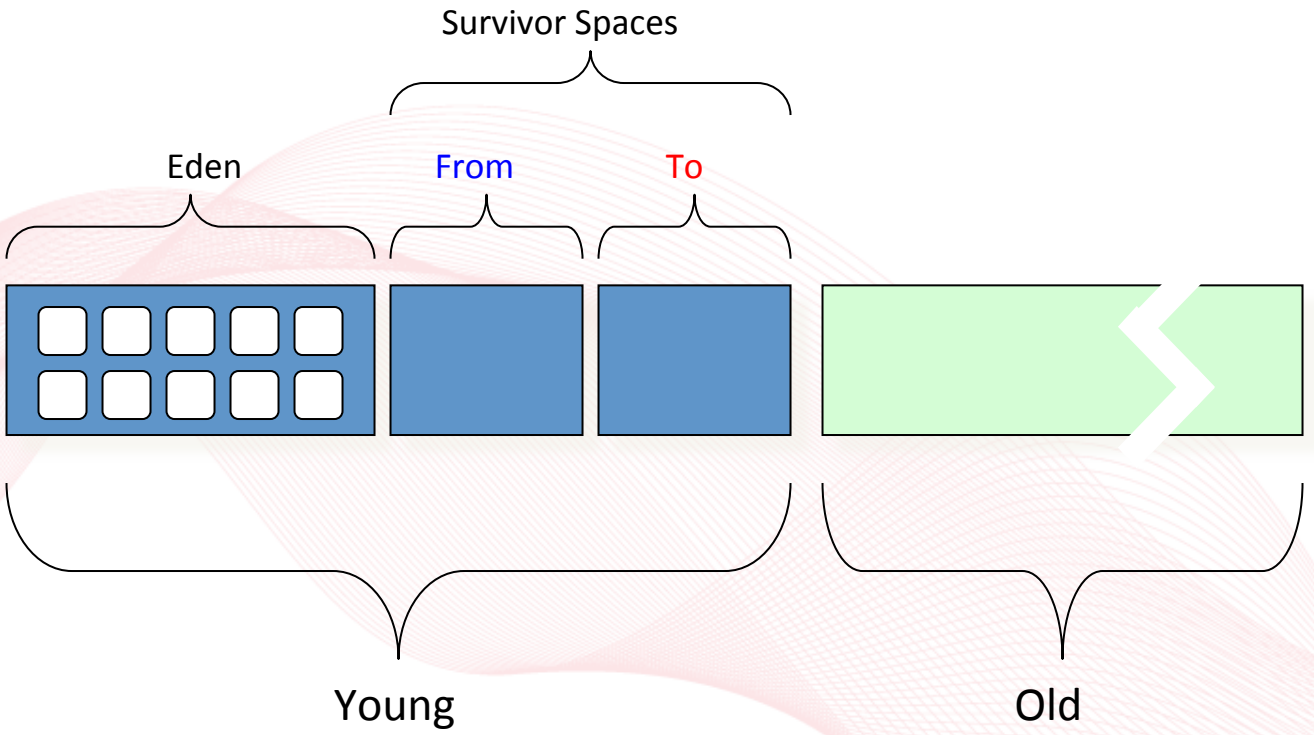


GC Fundamentals

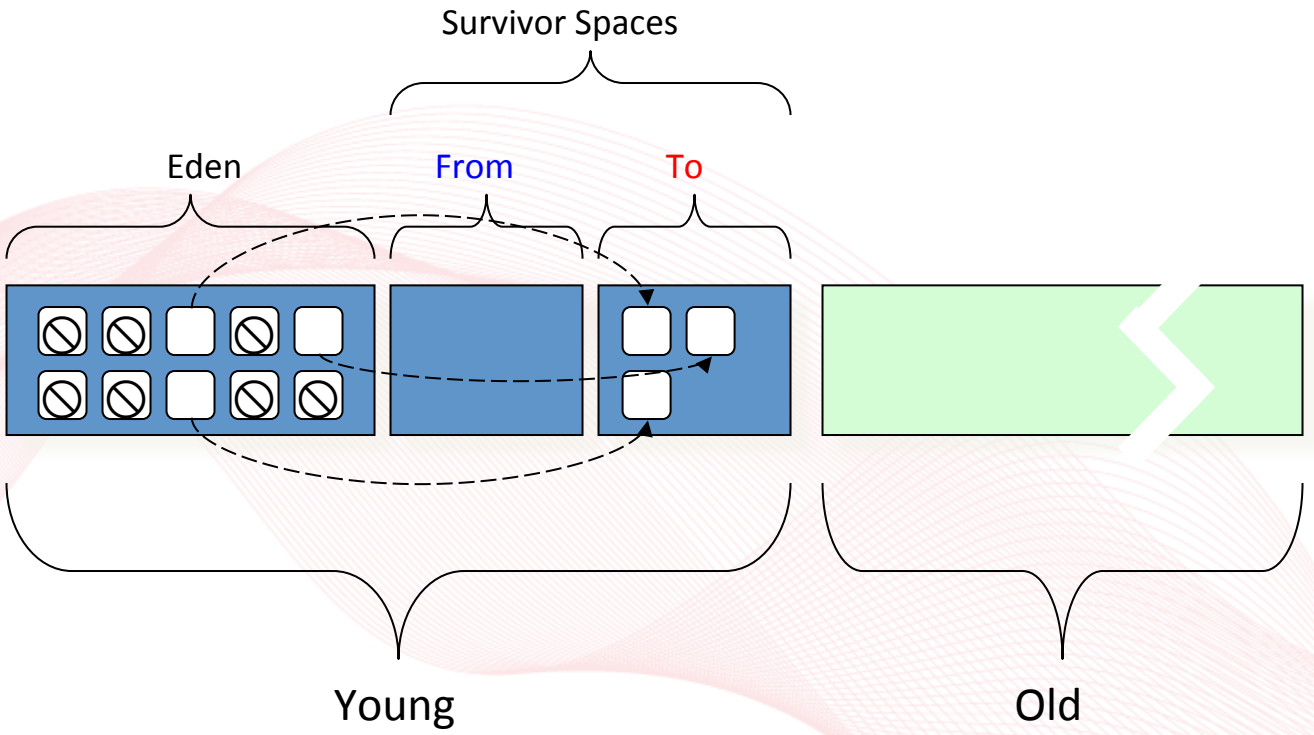
Garbage Distribution – Objects die young



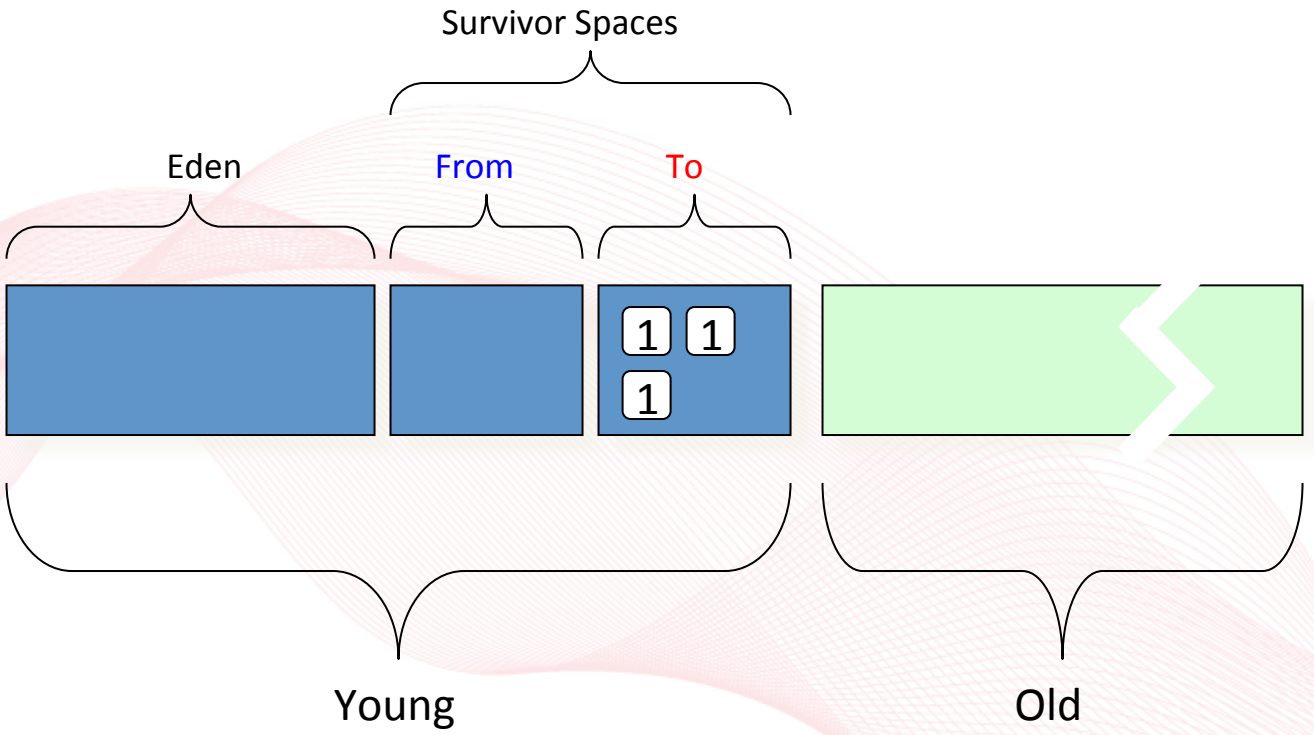
Minor GC (Young Generation GC)



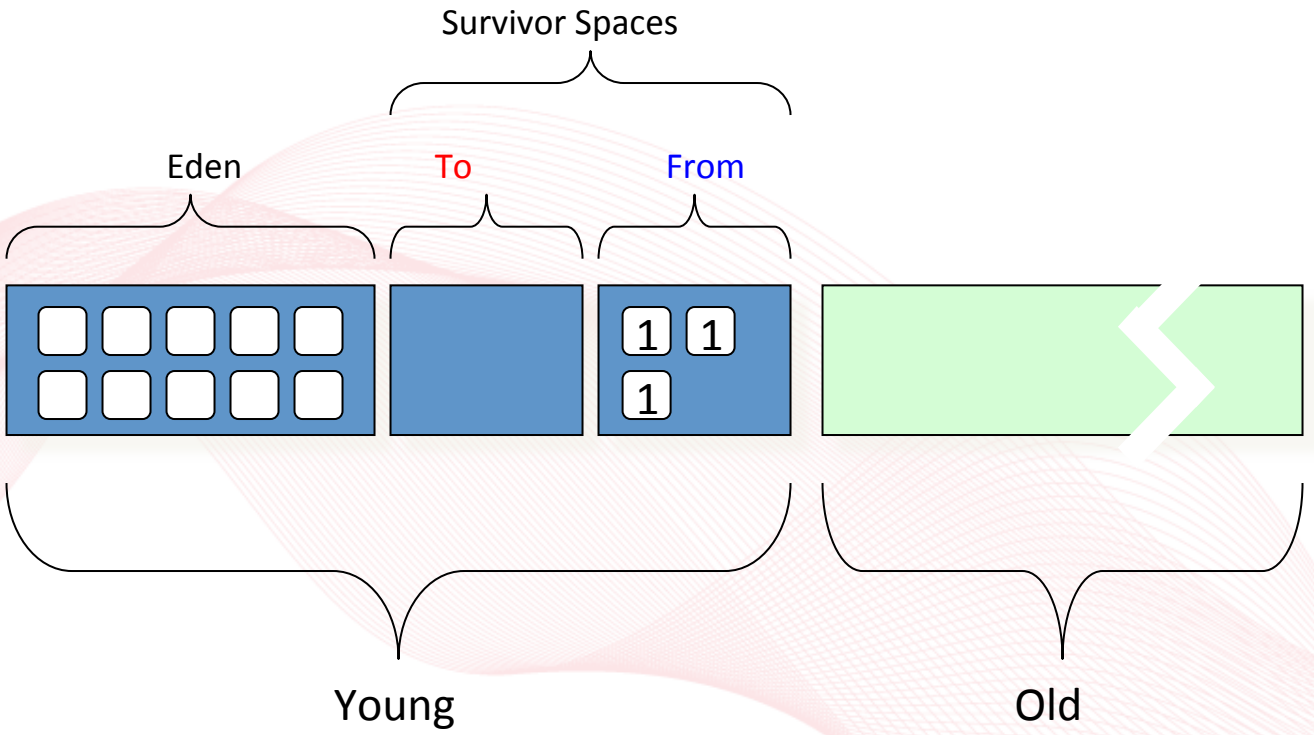
Minor GC (Young Generation GC)



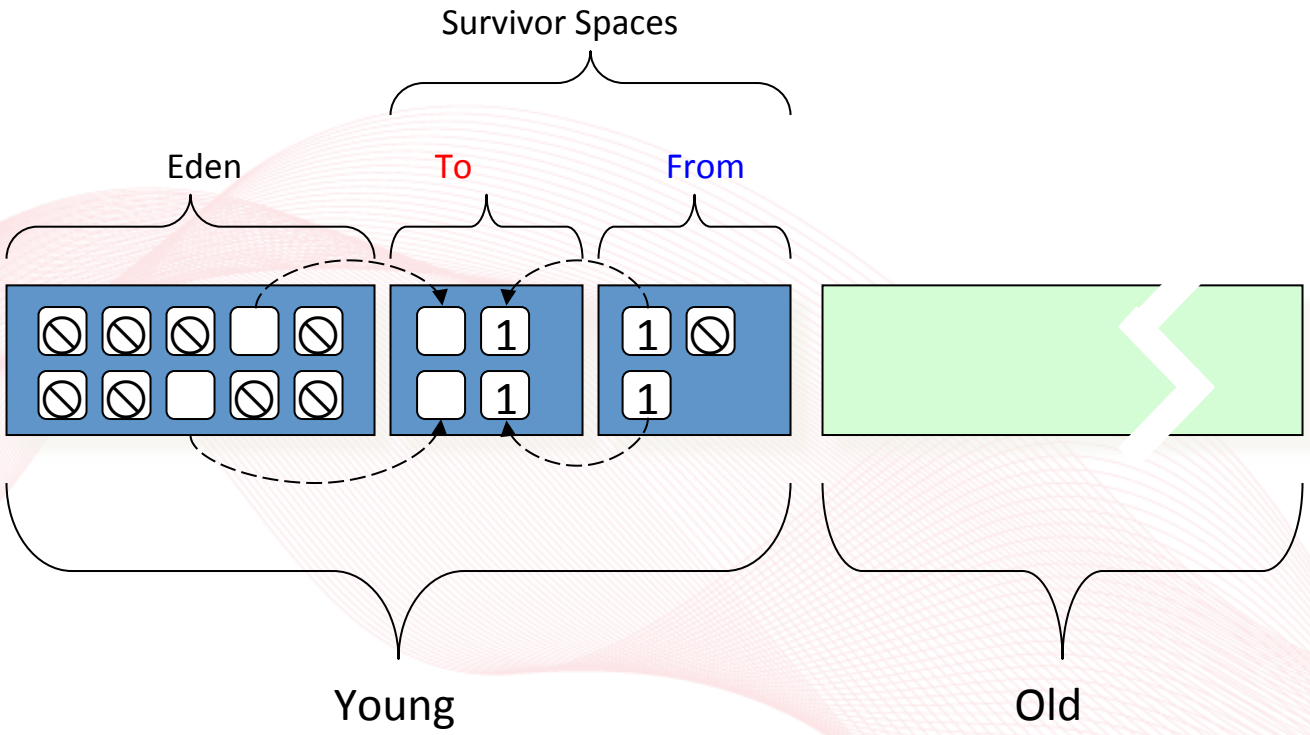
Minor GC (Young Generation GC)



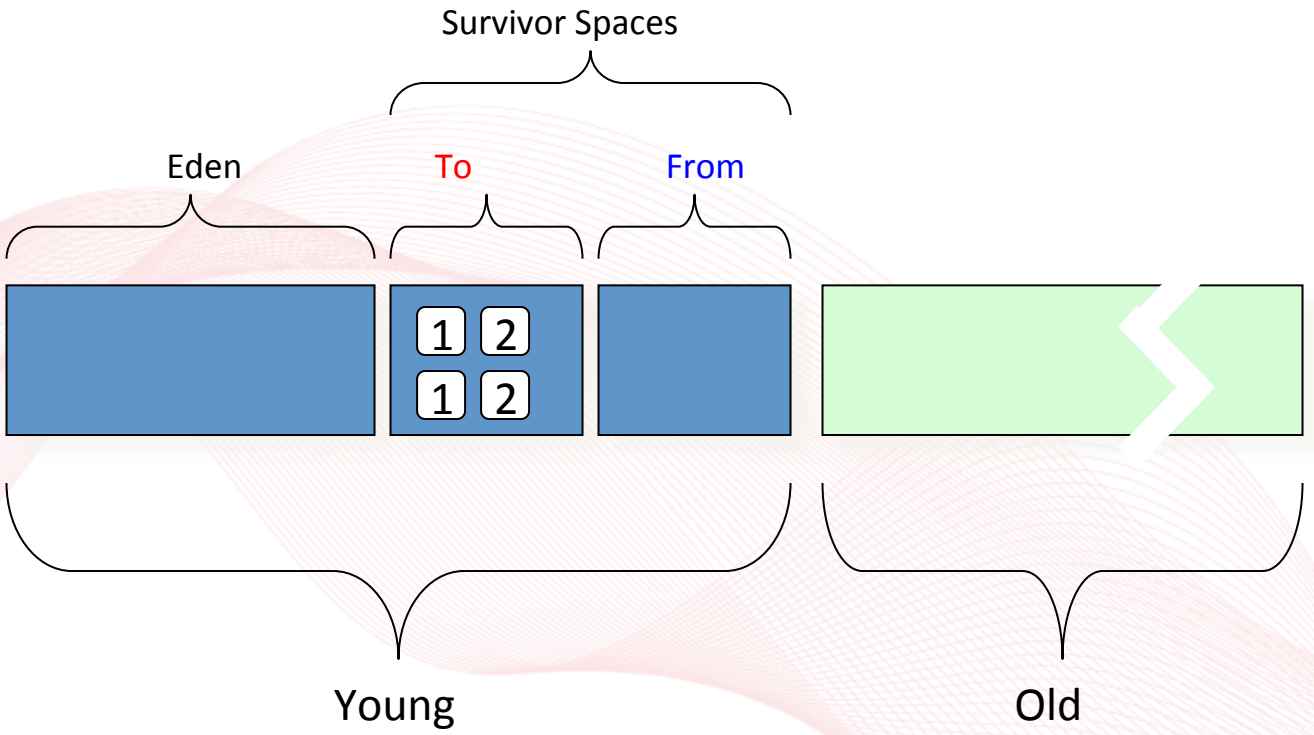
Minor GC (Young Generation GC)



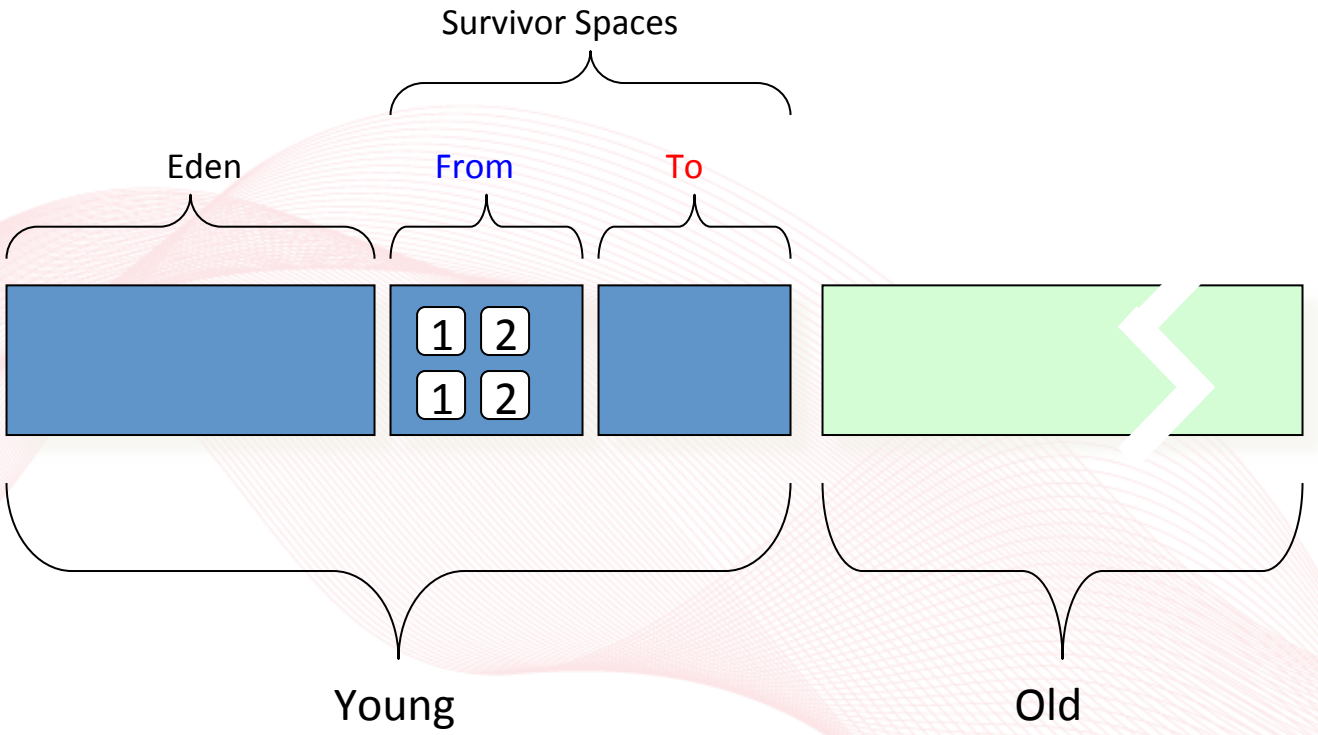
Minor GC (Young Generation GC)



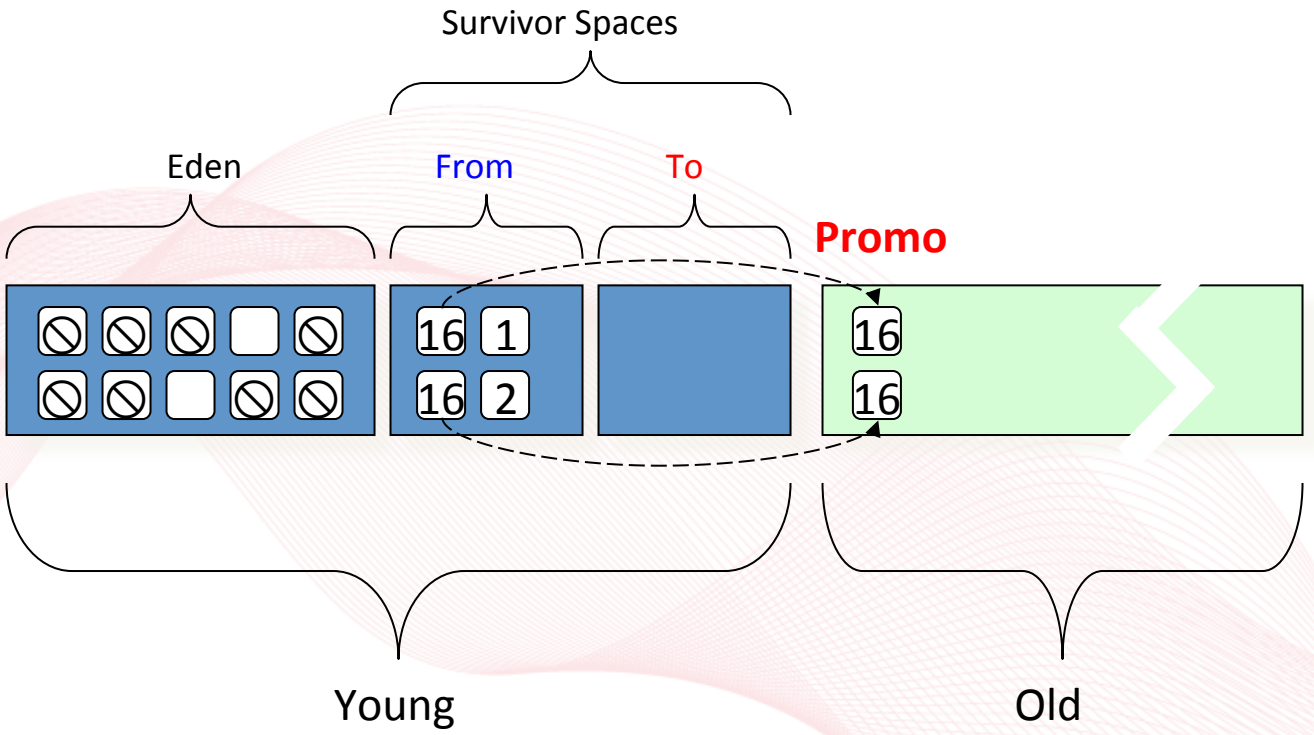
Minor GC (Young Generation GC)



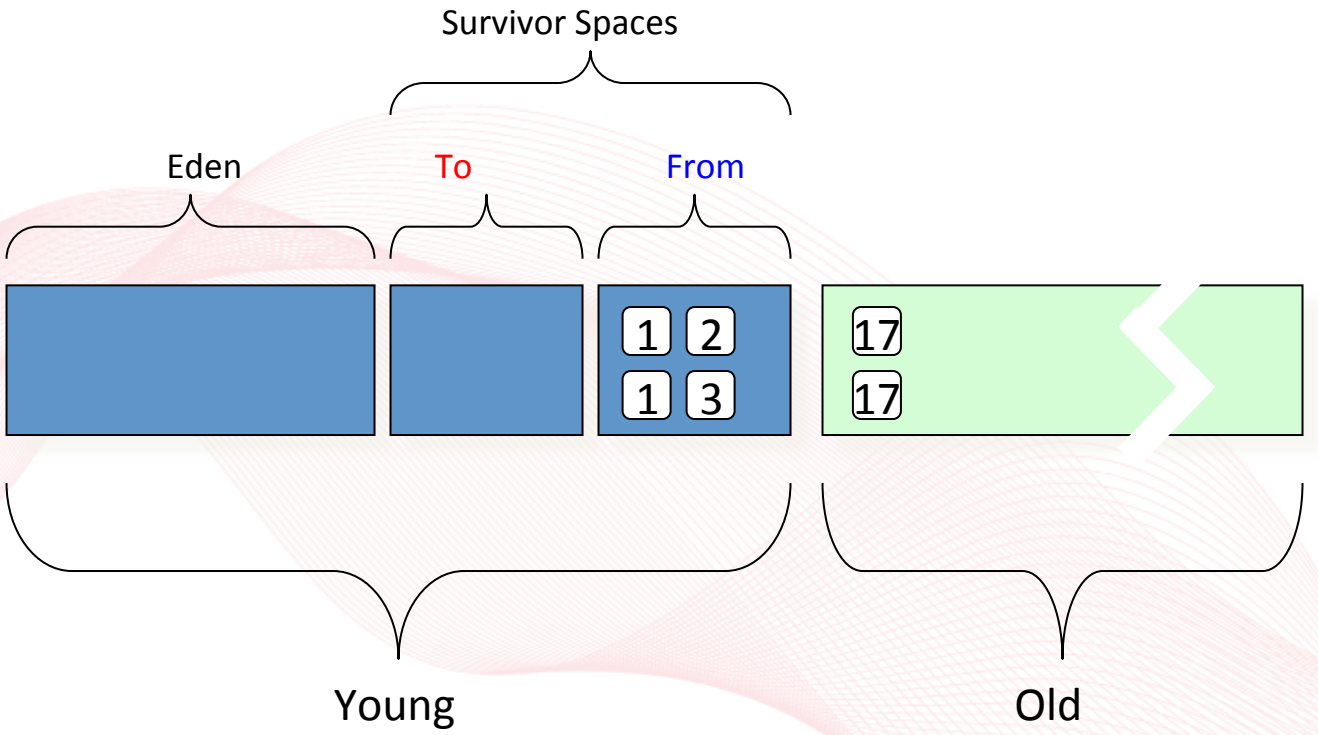
Minor GC (Young Generation GC)



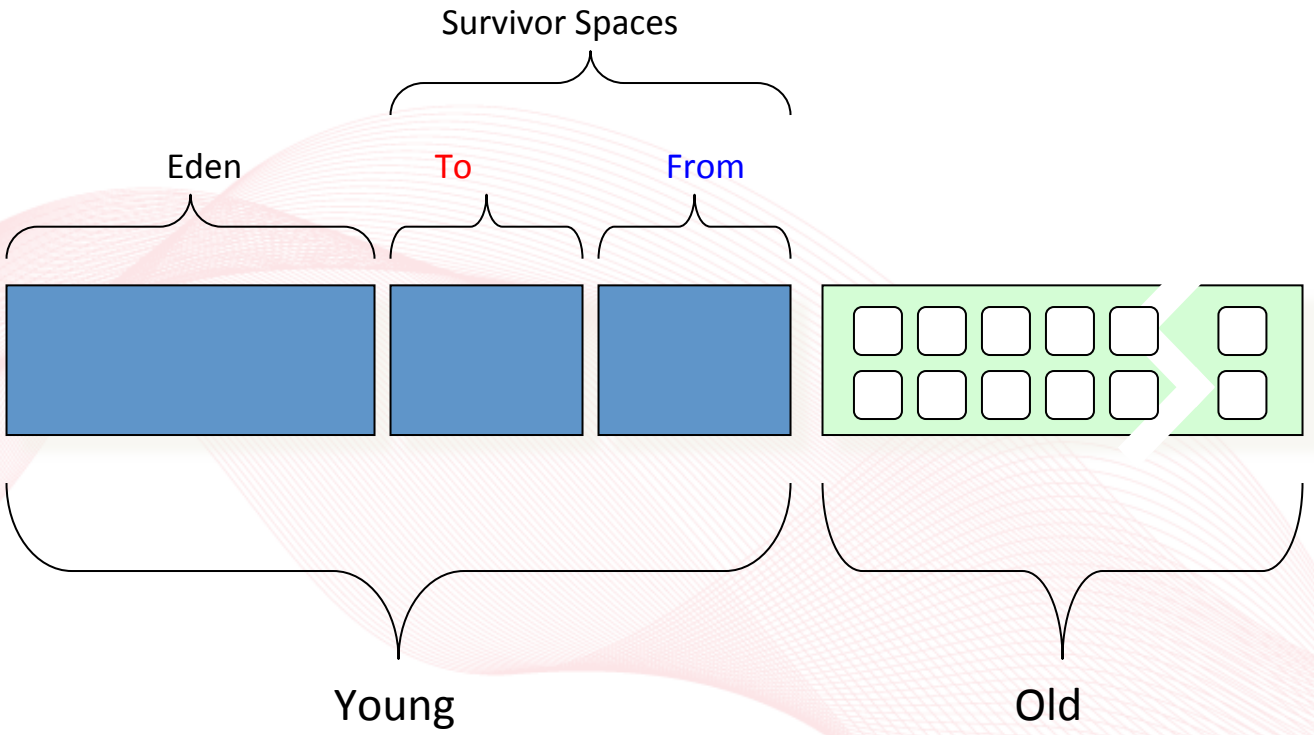
Minor GC (Young Generation GC)



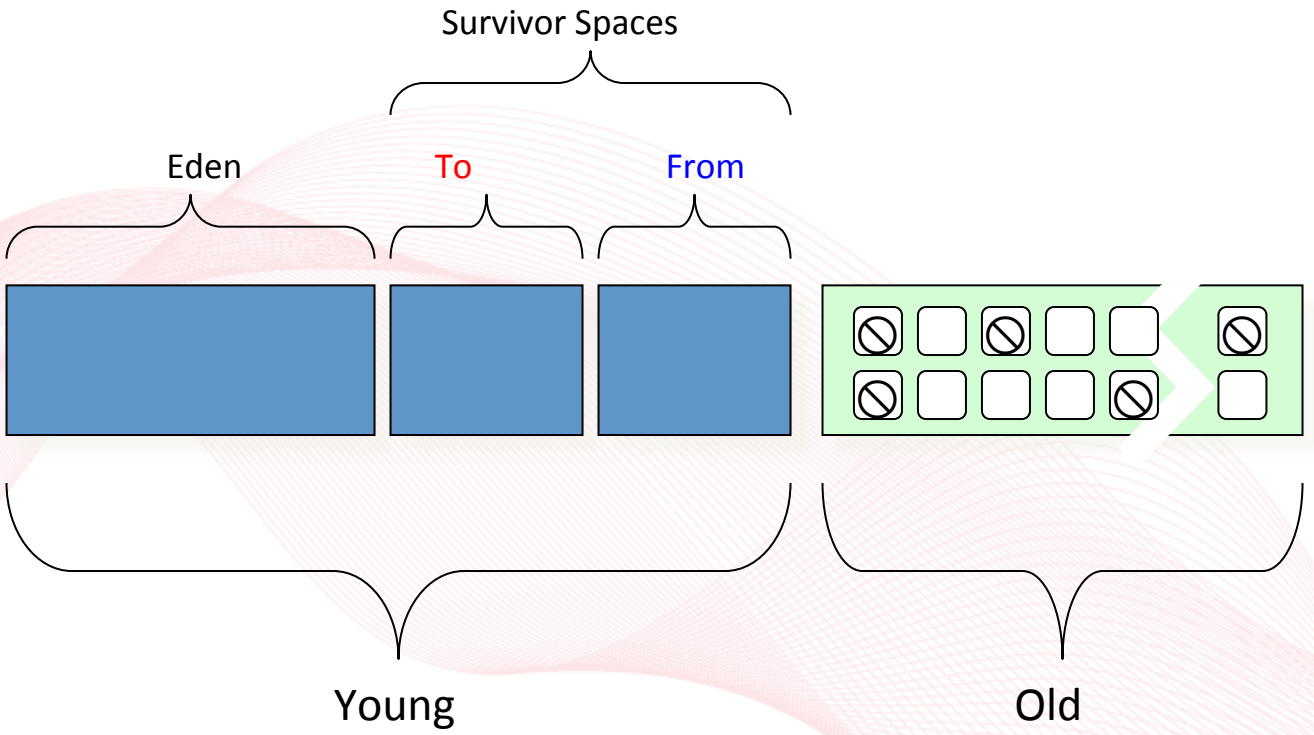
Minor GC (Young Generation GC)



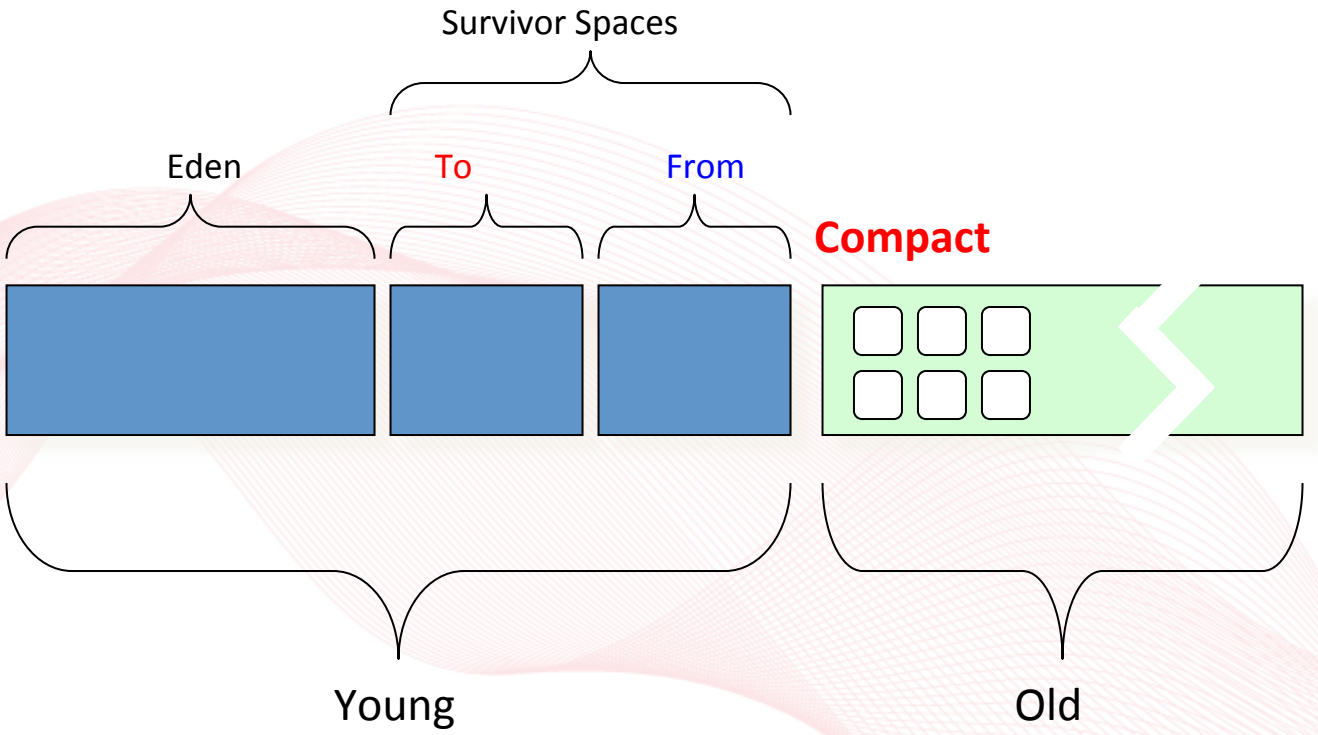
Major GC (Old Generation GC, Full GC)



Major GC (Old Generation GC, Full GC)



Major GC (Old Generation GC, Full GC)



Tuning Guideline

- 讓 heap 的 size 足以容納程式的一般運作
- Major GC 通常會比 Minor GC 花更多的時間
- 盡可能減少 Major GC 的次數

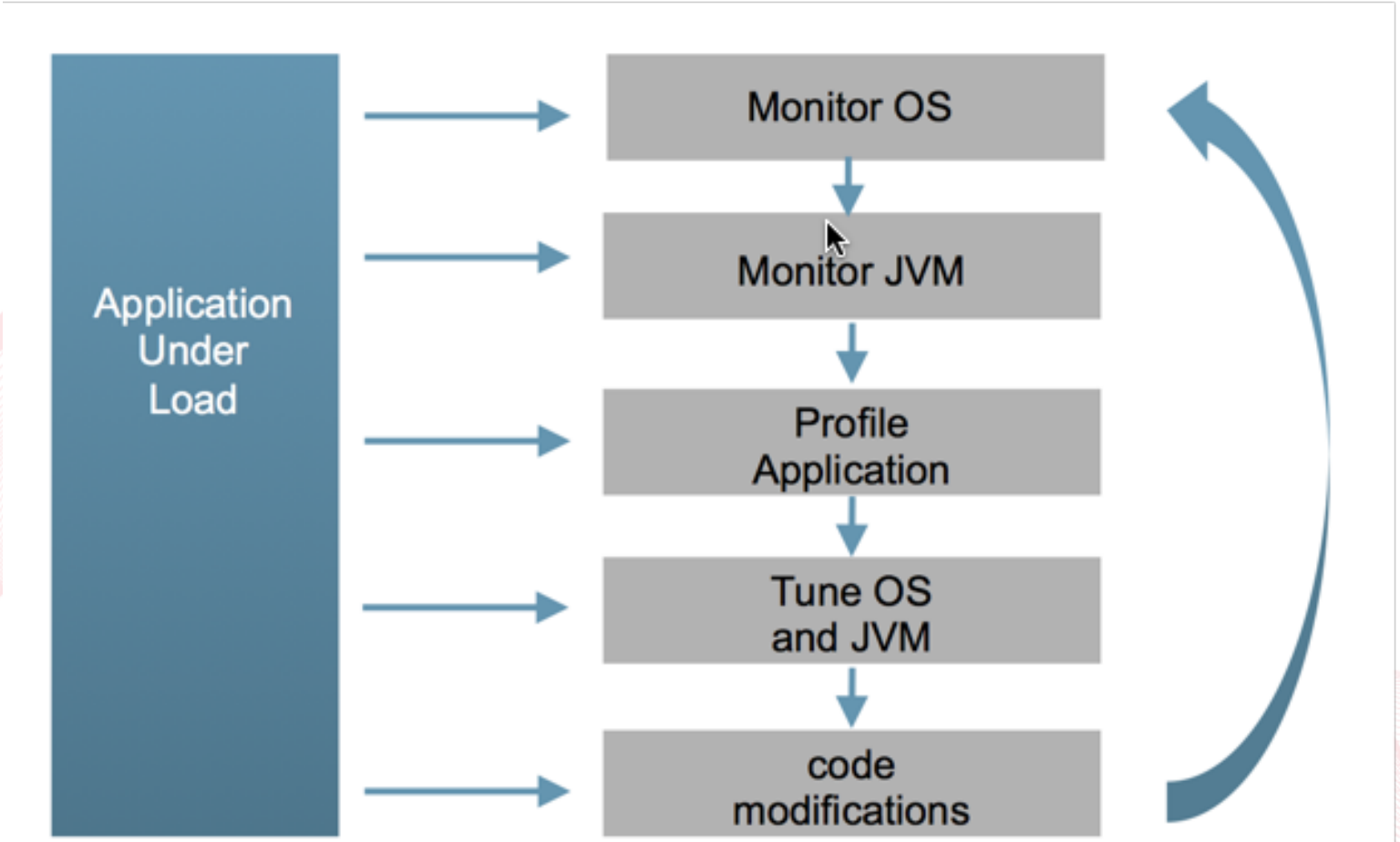
HOTSPOT GC TUNING

花太多時間做 GC → Performance變慢

Performance變慢→花太多時間做 GC

不一定!!!

Performance Tuning Methodology



Advanced JVM Tuning; Java One 2011; David Keenan, Monica Beckwith

GC Logging in Production

- Don't be afraid to enable GC logging in production
  - Very helpful when diagnosing production issues
- Extremely low / non-existent overhead
  - Maybe some large files in your file system.
  - We are surprised that customers are still afraid to enable it

If Someone doesn't enable GC logging in production

I shoot them!

Most Important GC Logging Parameters

- You need at least:
  - -XX:+PrintGCDetails
  - -XX:+PrintGCTimeStamps
    - Add -XX:+PrintGCDateStamps if you must know the time
- Also useful
  - -Xloggc:<file>
  - Rotate GC Log, After Java 1.6_34 (or 1.7_2):

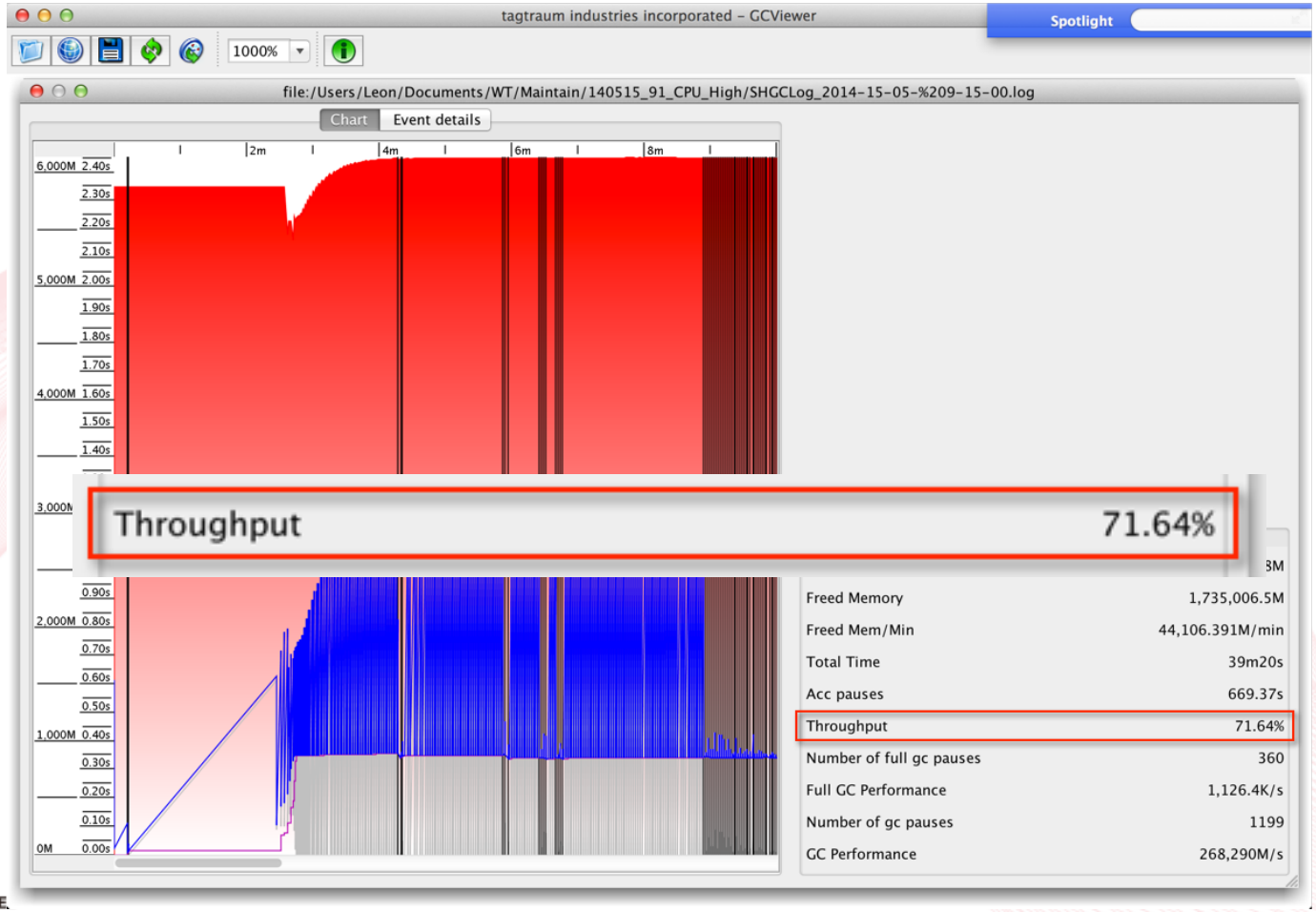
    -XX:+UseGCLogFileRotation

    -XX:NumberOfGCLogFiles=5

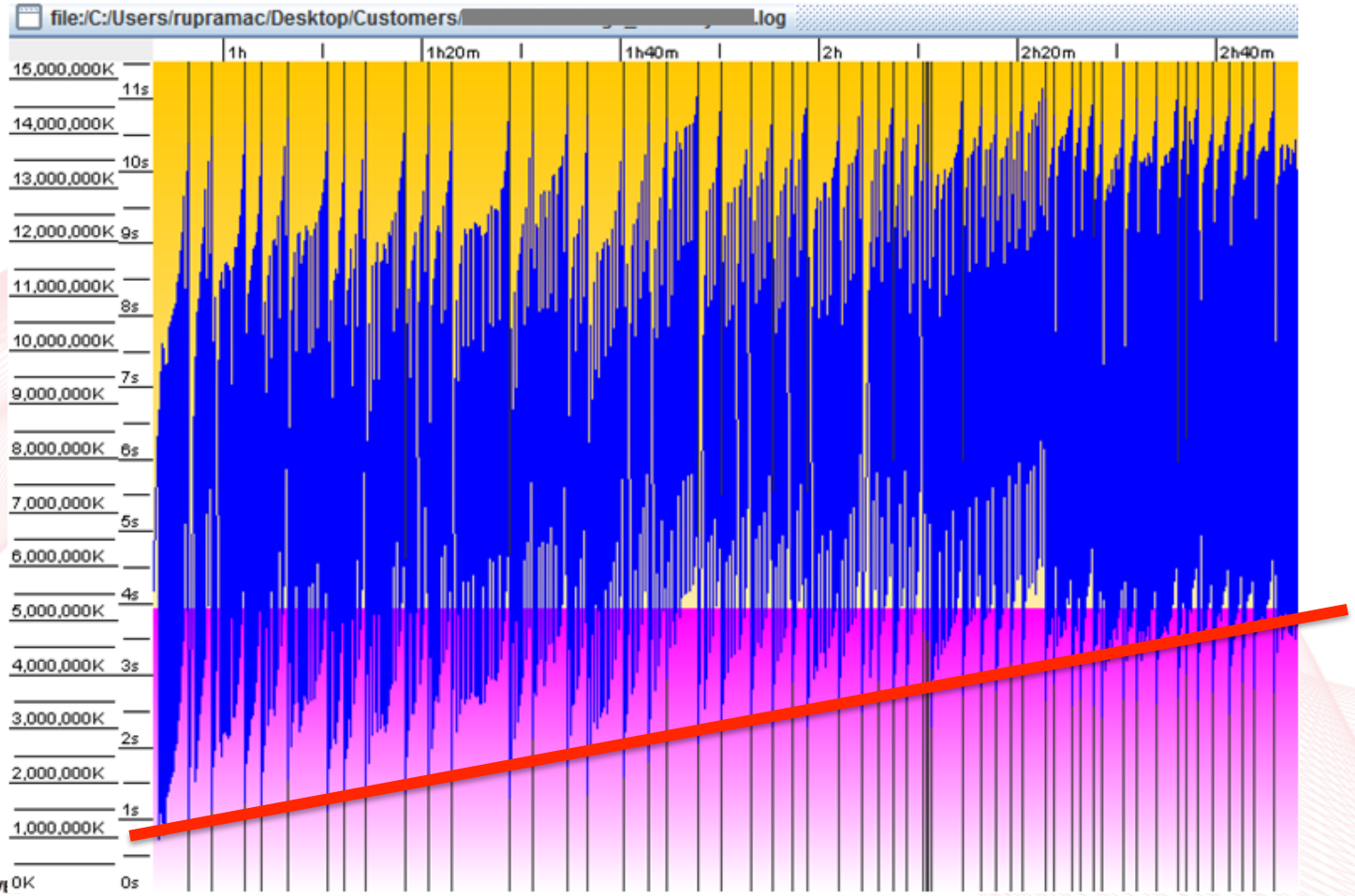
    -XX:GCLogFileSize=10M

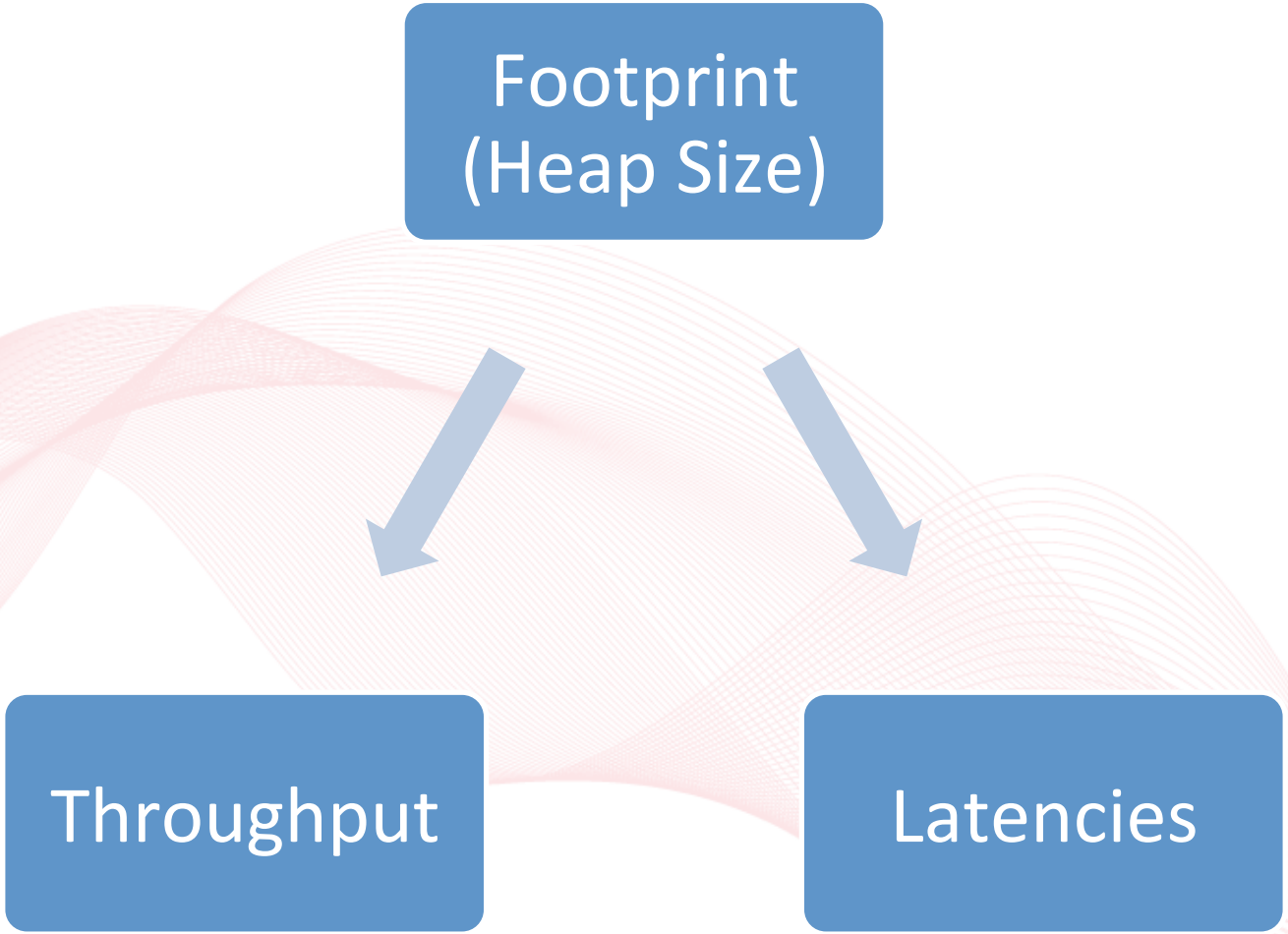
GCViewer – Offline analysis of GC logs

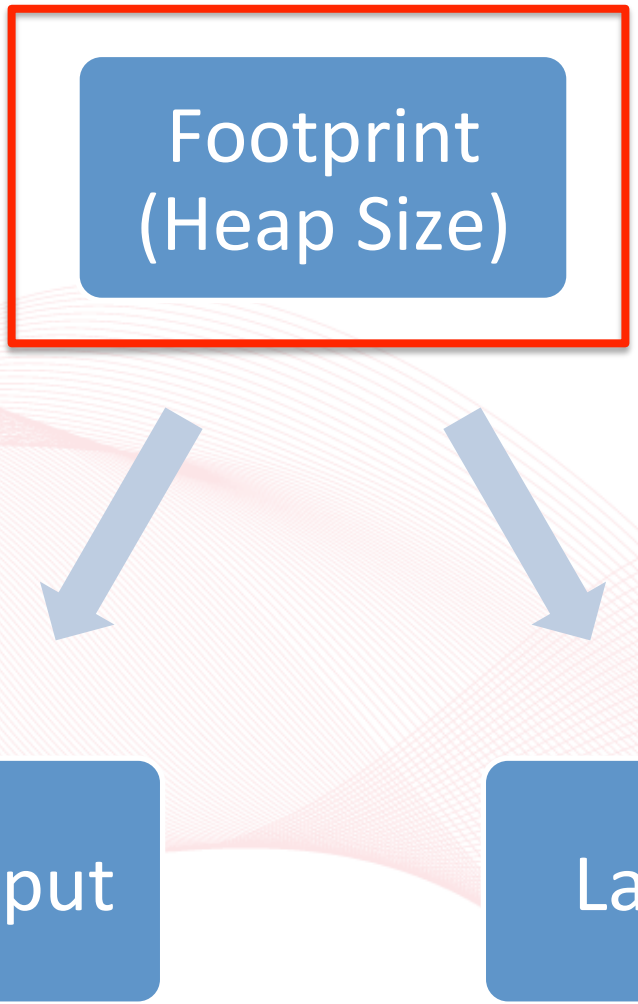
https://github.com/chewiebug/GCViewer



GCViewer – Memory Leak Pattern




Footprint
(Heap Size)
Throughput
Latencies


Footprint
(Heap Size)
Throughput
Latencies

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

Calculate Live Data Size (LDS)

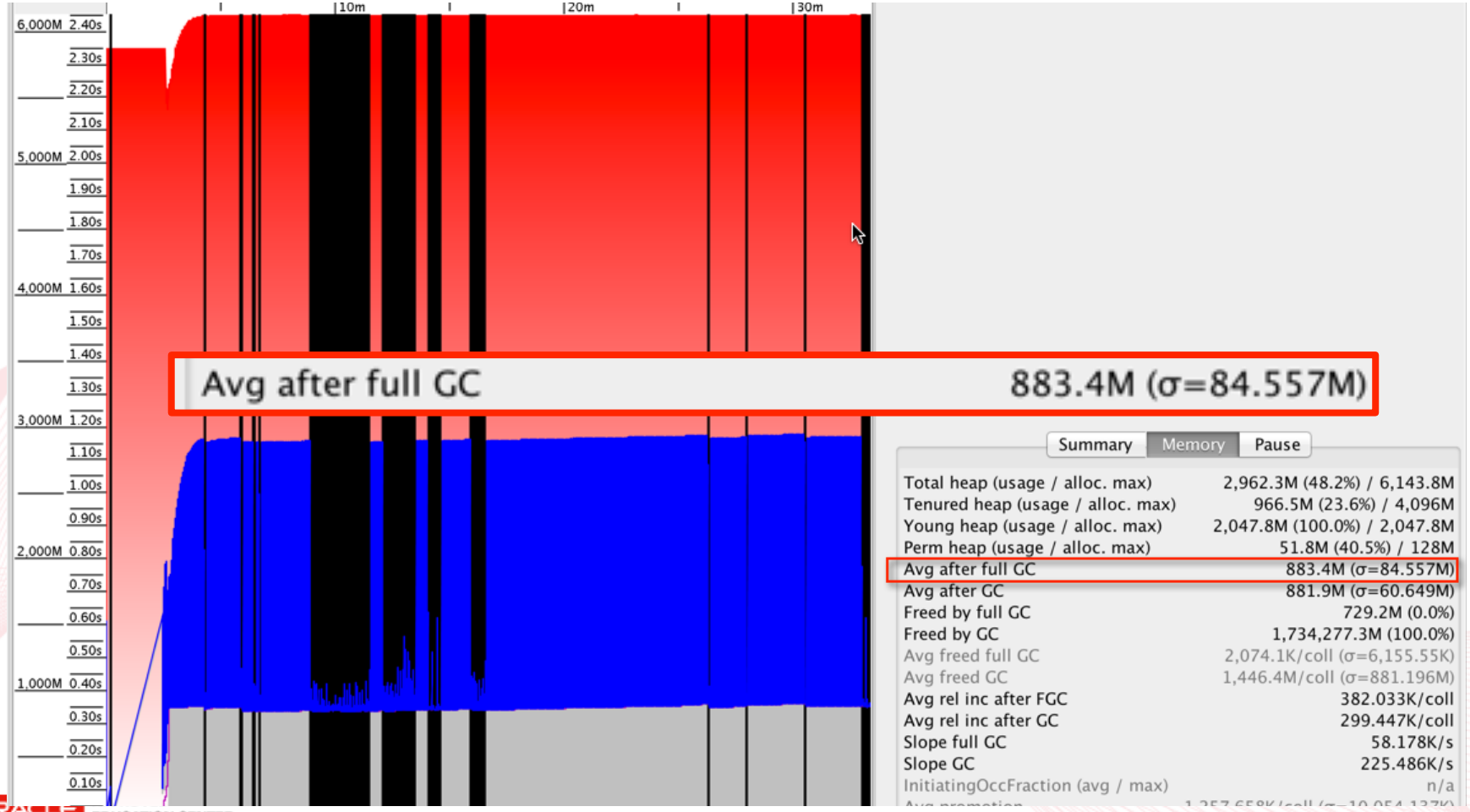
- 經過 full GC 後, 所剩下的 heap size
- Make sure you get full GCs during steady-state
- Induce a couple if necessary
  - JConsole / VisualVM
  - jmap -histo:live <pid>
  - System.gc
    - Remove it before deployment. :-)

Calculate Live Data Size

- GC log example:

```
170.517: [Full GC
          [PSYoungGen: 10128K->0K(163392K)]
          [ParOldGen: 348898K->161378K(350208K)]
                      359026K->161378K(513600K)
          [PSPermGen:   5799K->5798K(16384K)],
                                 0.3224960 secs]
```

Calculate Live Data Size by GCViewer

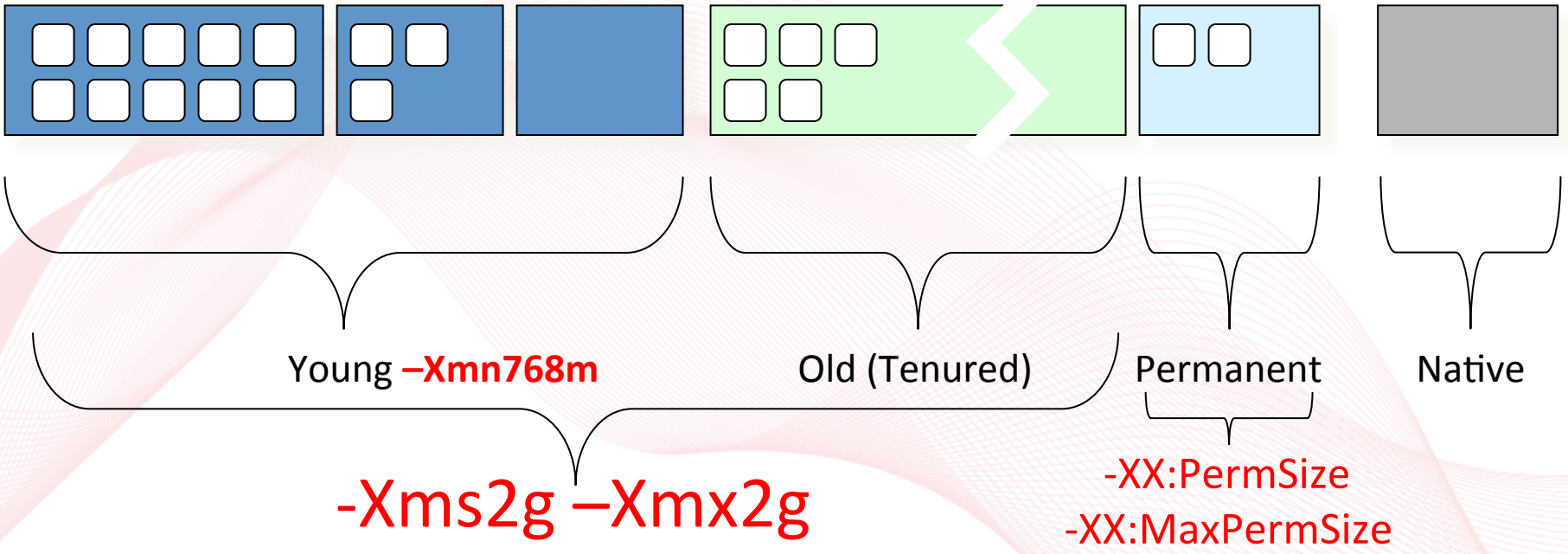


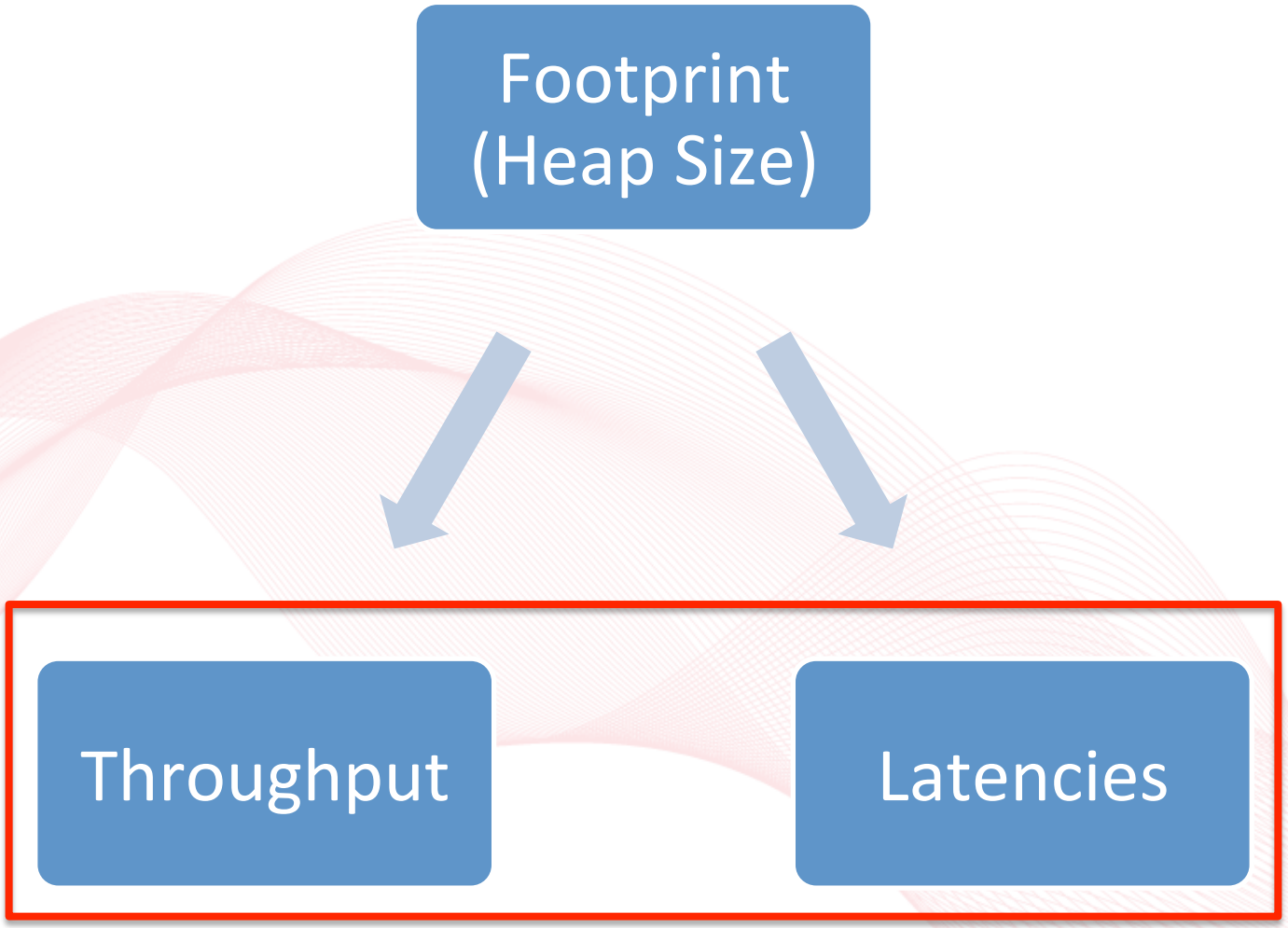
Initial Heap Configuration

- Rule of thumb
  - Set -Xms and -Xmx to 3x to 4x LDS
  - Set both -XX:PermSize and -XX:MaxPermSize to around 1.2x to 1.5x the max perm gen size
  - Set the generation sizes accordingly
    - Young gen should be around 1x to 1.5x LDS
    - Old gen should be around 2x to 3x LDS
    - e.g., young gen should be around 1/3-1/4 of the heap size

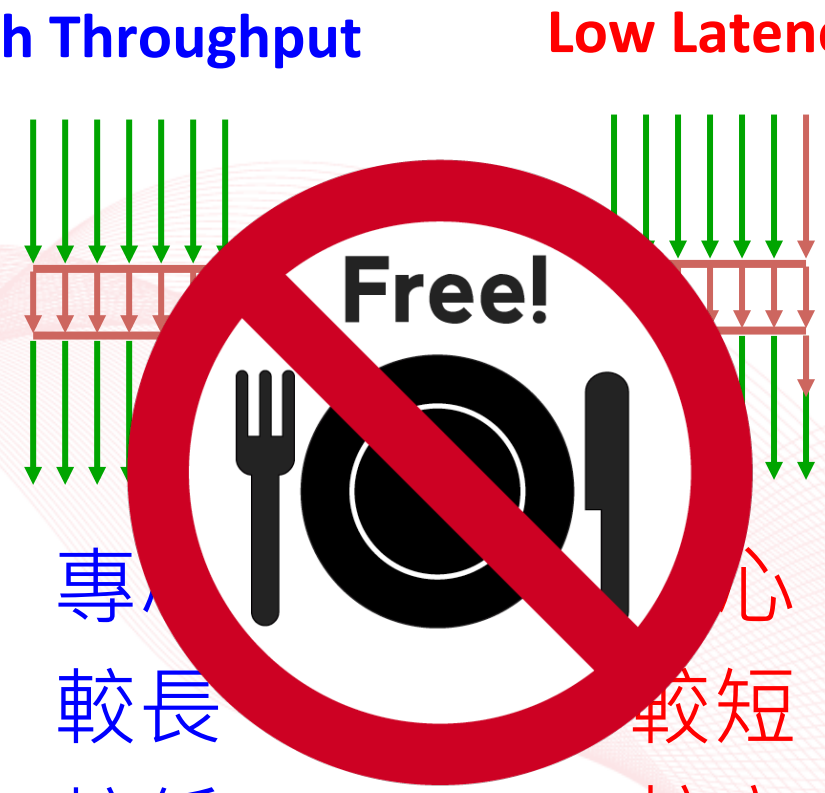
Hotspot JVM Heap Layout

For LDS of 512m : -Xmn768m -Xms2g -Xmx2g




Footprint
(Heap Size)
Throughput
Latencies

High Throughput vs. Low Latency



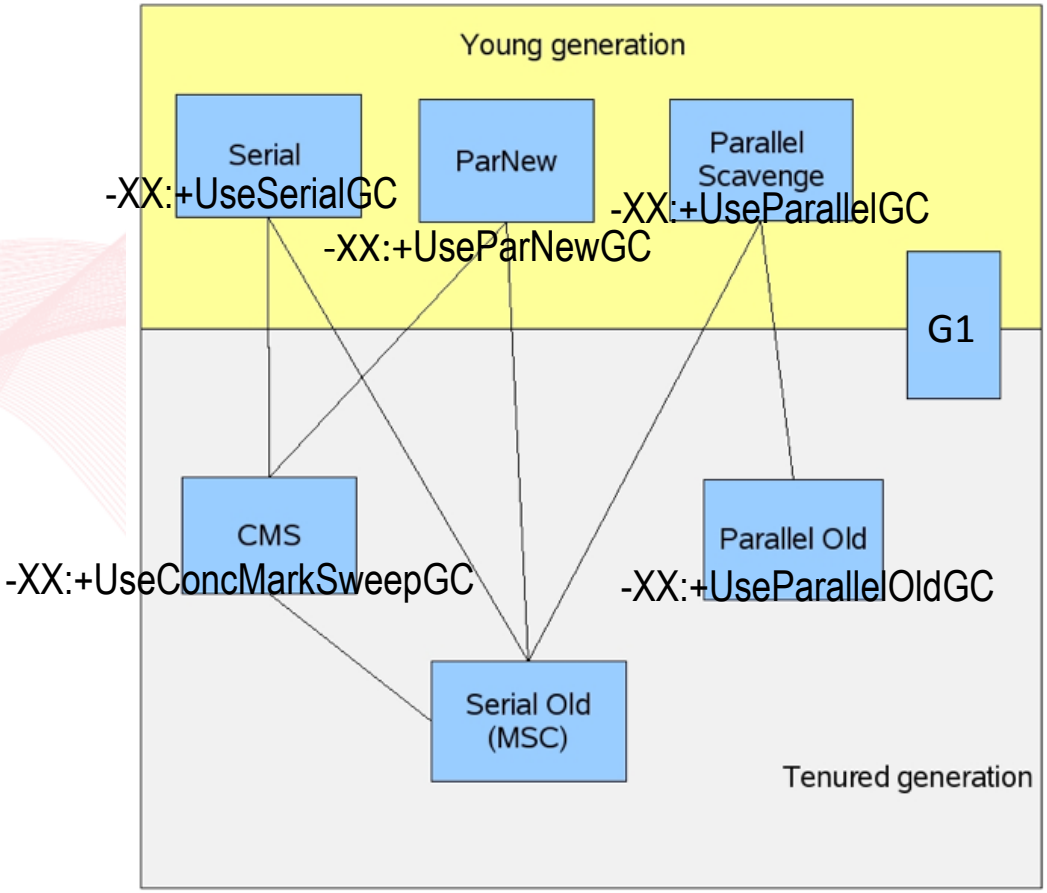
High Throughput vs. Low Latency

- For most applications, GC overhead is small
  - 2% – 5%
- High Throughput GCs
  - Move most work to GC pauses
  - Application threads do as little as possible
  - Least overall GC overhead
- Low Latency (Pause) GCs
  - Move work out of GC pauses
  - Application threads do more work
    - Bookkeeping for GC more expensive
  - More overall GC overhead

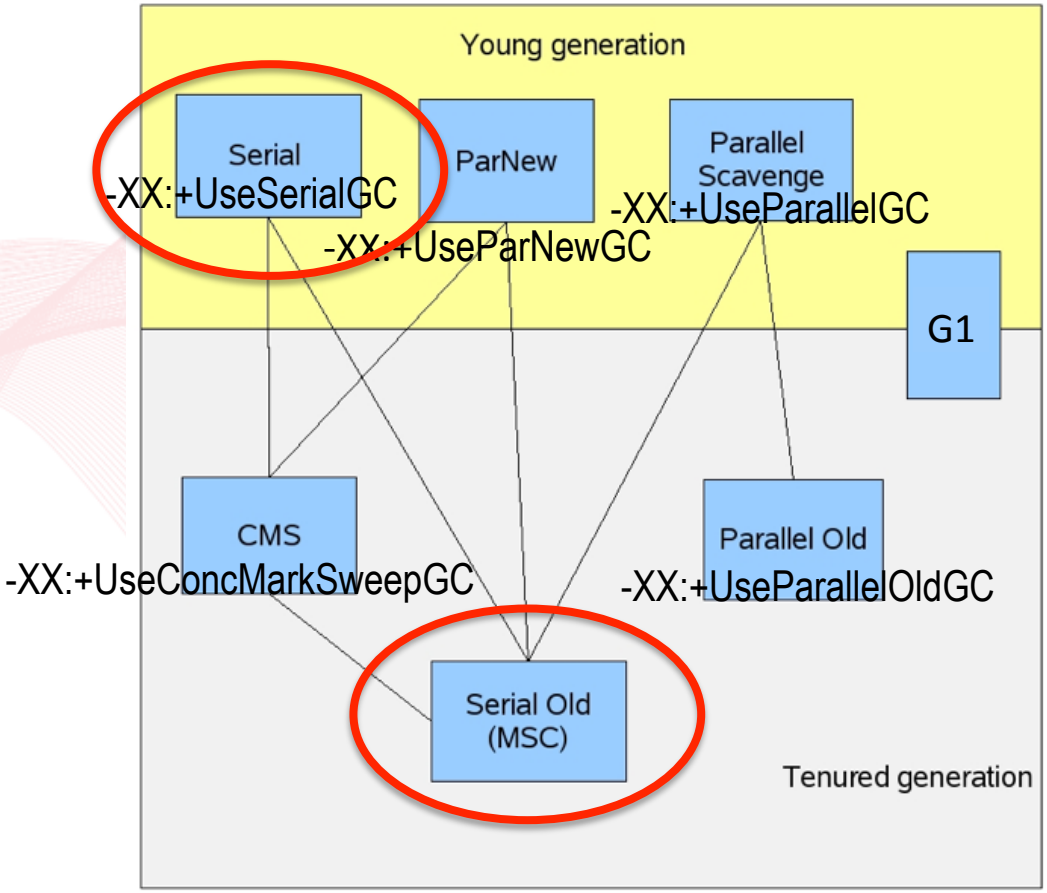
Application Requirement

- Different applications have different requirements
  - Higher Throughput:
    - Batch processing
    - Web application: pauses during garbage collection may be tolerable, or simply obscured by network latencies
  - Lower Latencies:
    - Interactive graphics application

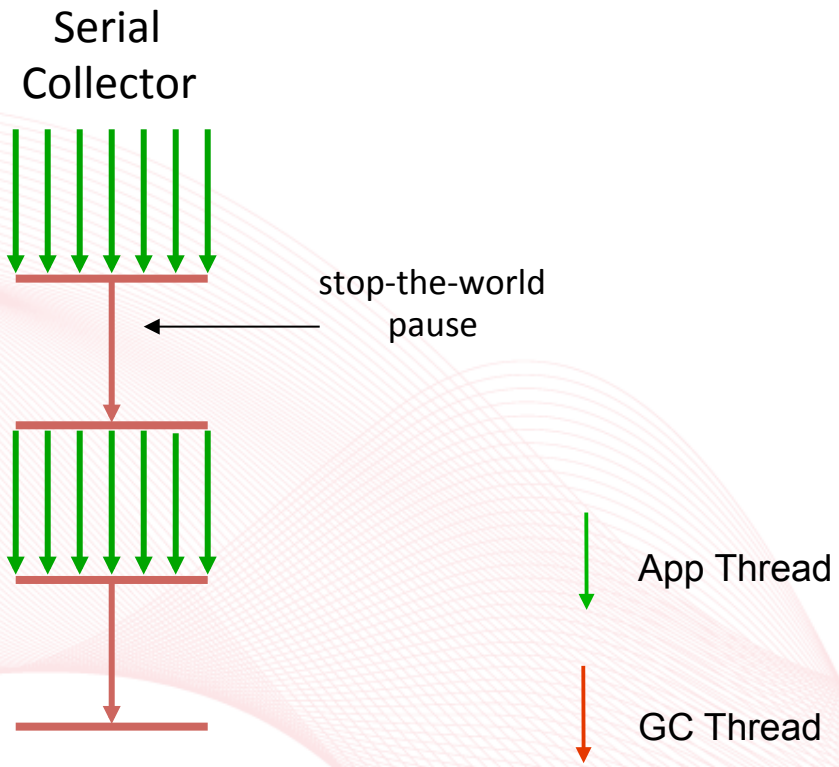
HotSpot Garbage Collectors in Java SE 6



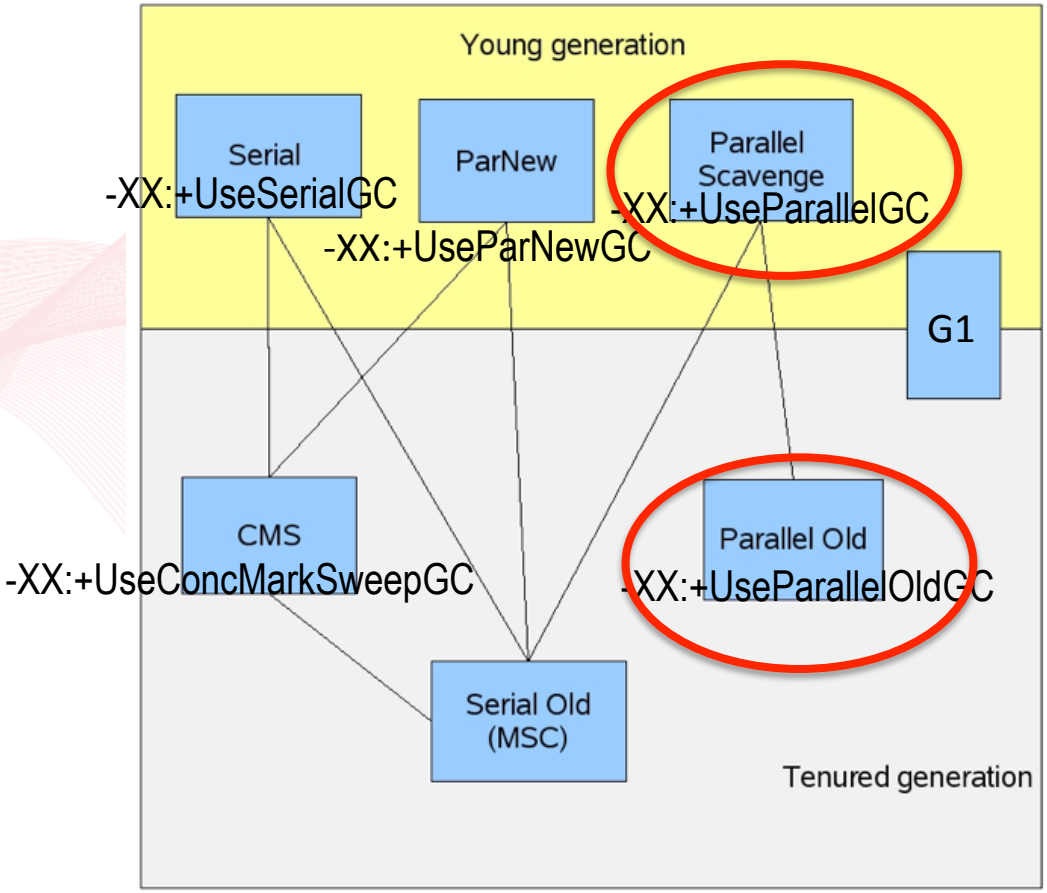
HotSpot Garbage Collectors in Java SE 6



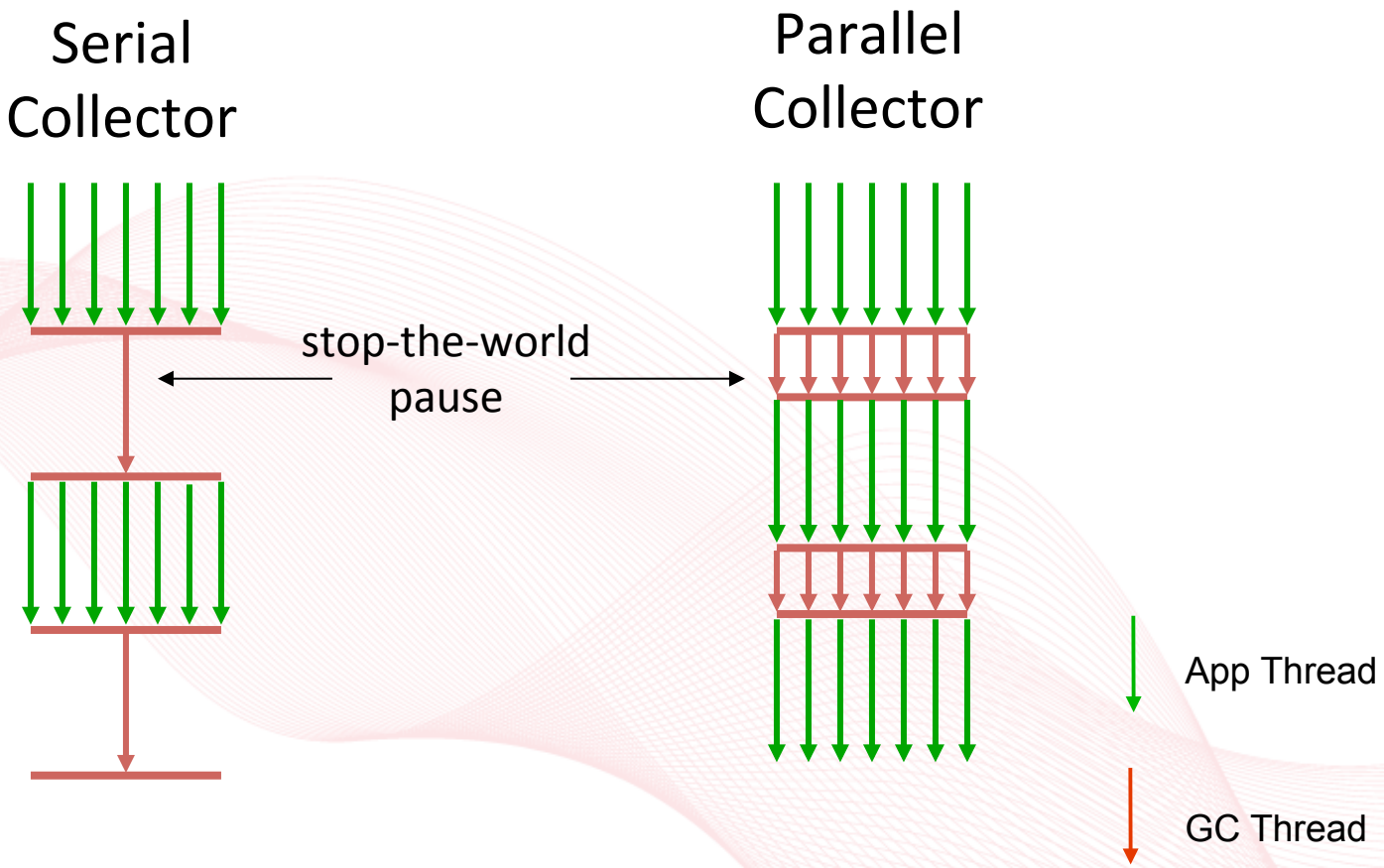
Serial Collector



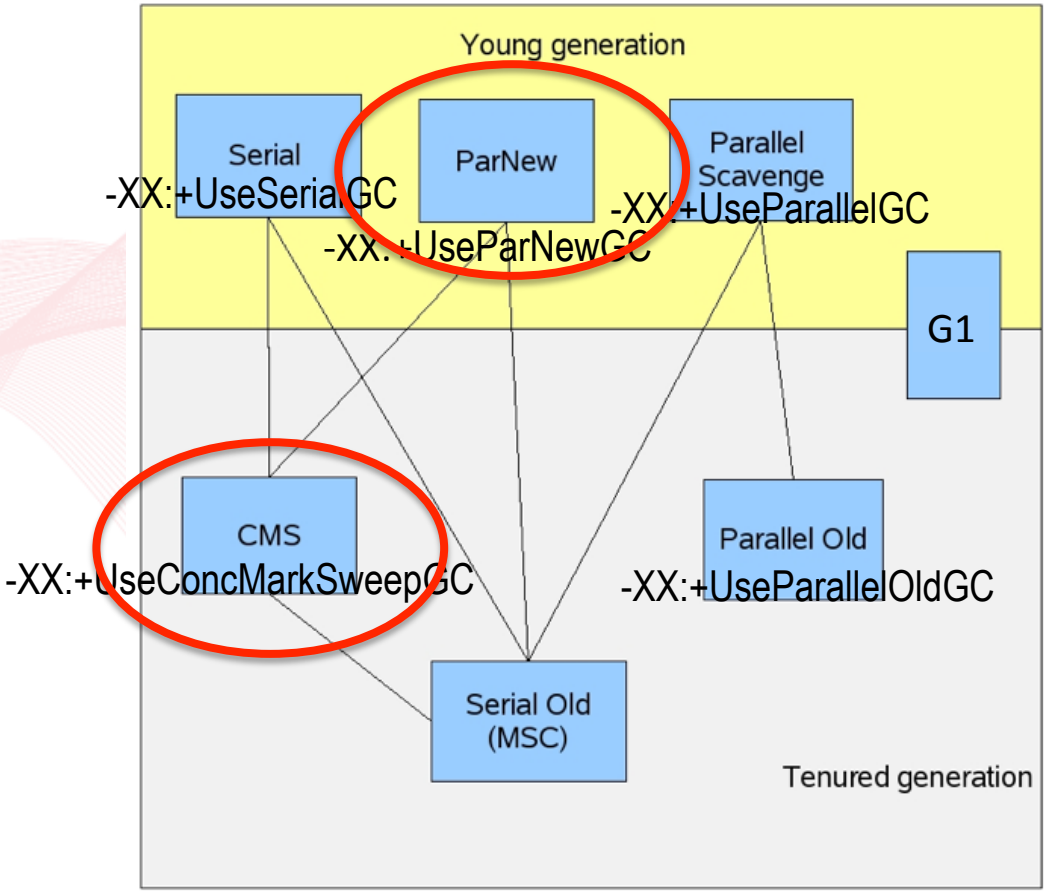
HotSpot Garbage Collectors in Java SE 6



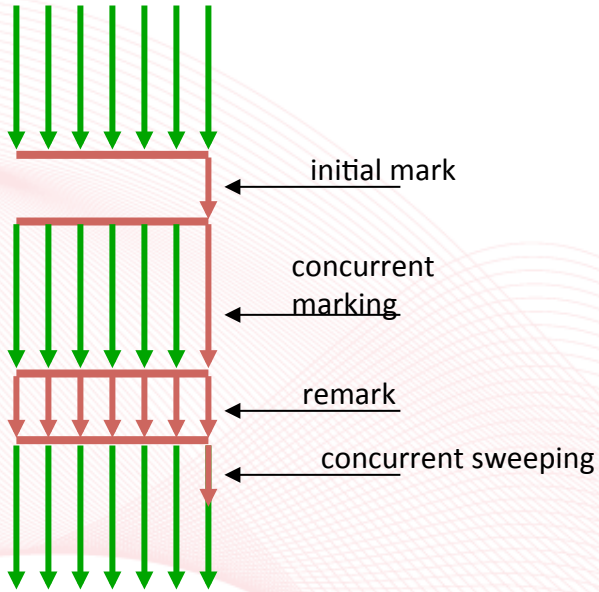
Serial VS Parallel Collector



HotSpot Garbage Collectors in Java SE 6



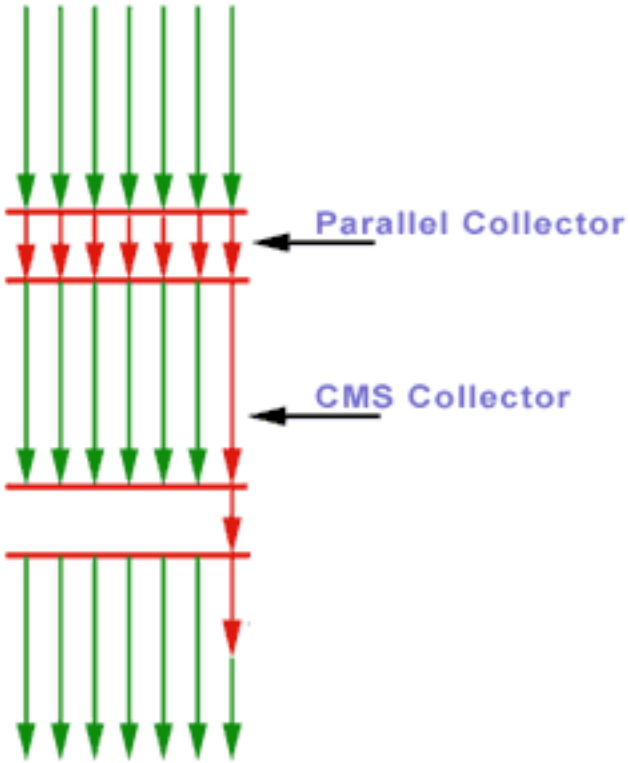
CMS Collector



App Thread

GC Thread

CMS Collector with ParNewGC



App Thread

GC Thread

GC Suggestions

Generation	Low Latency Collectors	Throughput Collectors
	2+ CPUs	2+ CPUs
Young	`-XX:+UseParNewGC`	`-XX:+UseParallelGC`
Old	`-XX:+UseConcMarkSweepGC`	`-XX:+UseParallelOldGC`

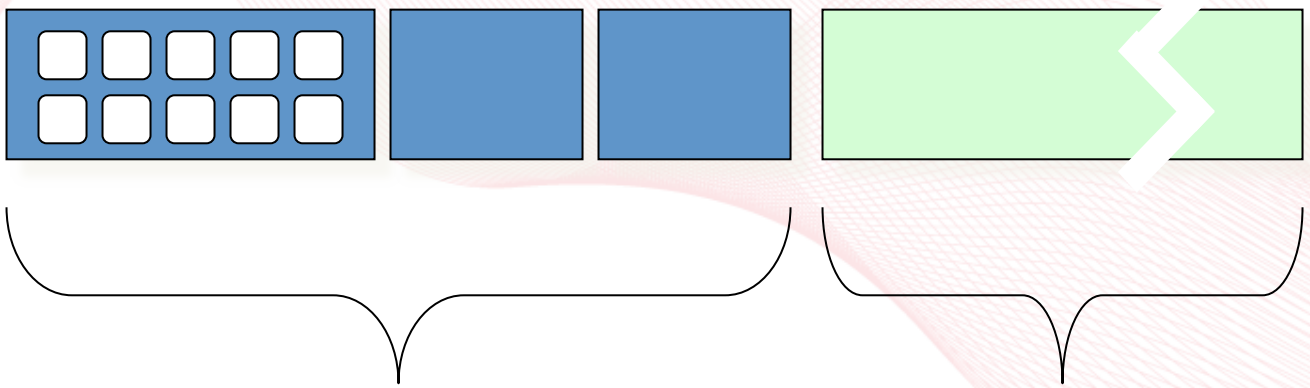
New G1 Collector as of WLS10.3.1/JDK 6 Update 14

NEXT GENERATION GC

Problems of Current GC

- Memory is cheaper and cheaper
- 64 bits JVM
- We can use very large heap!

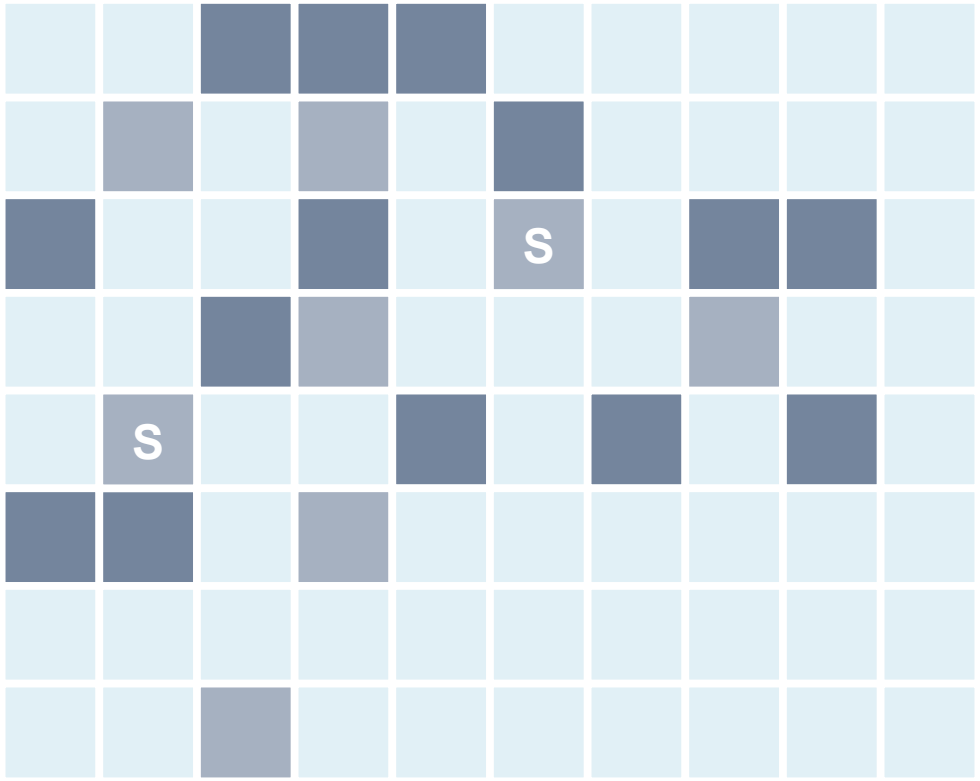
How to handle large heap size?



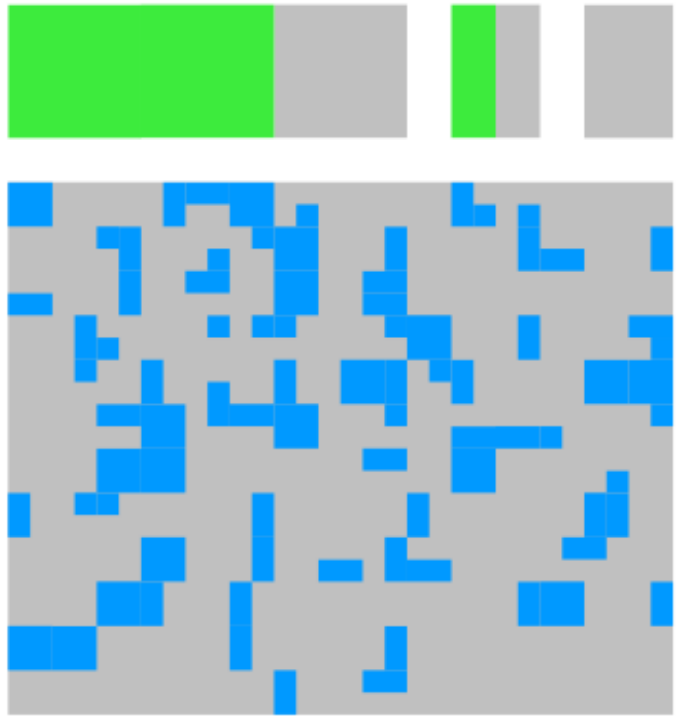
G1 Collector

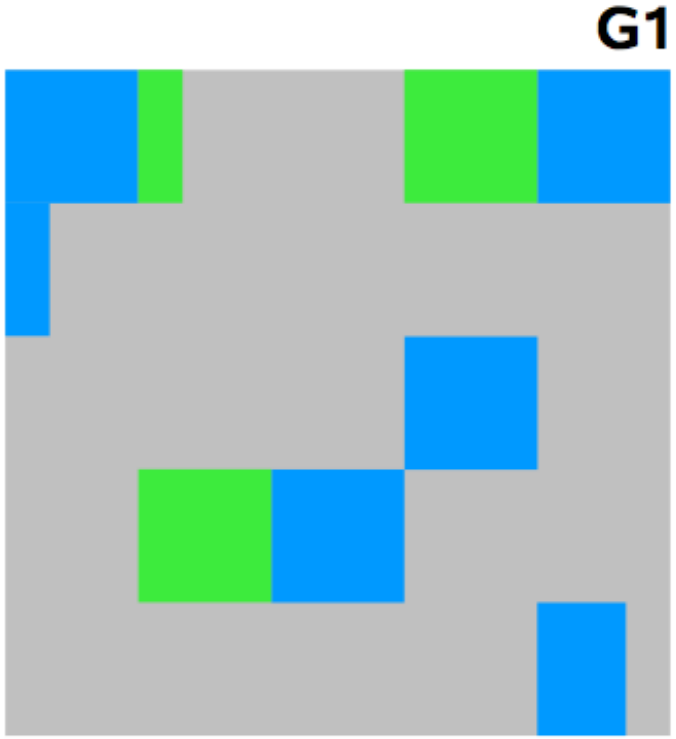
- The Garbage-First Collector
  - Since JDK 6u14, officially supported as of JDK 7u4
  - -XX:+UseG1GC
  - Future CMS Replacement
  - Server “Style” low latency collector
    - Parallel
    - Concurrent
  - Generational
  - Good Throughput
  - Compacting
  - Improved ease-of-use
  - Predictable (Soft Real-Time)

G1 – Heap Layout



CMS vs G1 Collectors





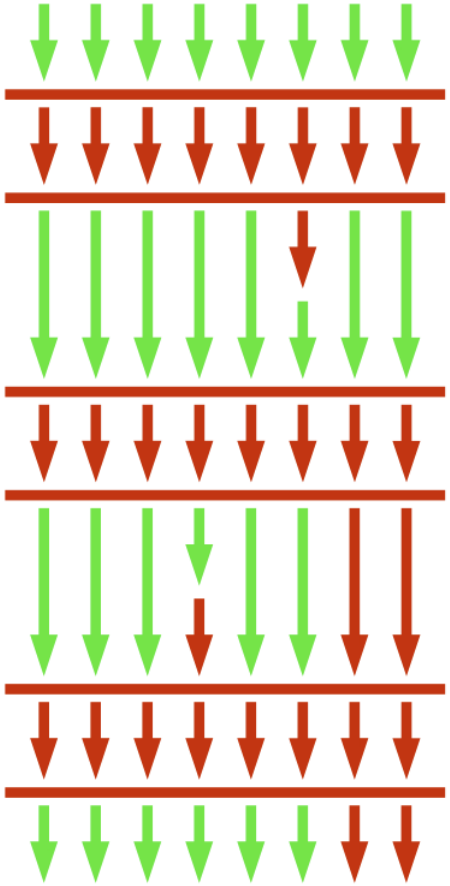
Non-Allocated Space

Young Generation

Old Generation

- Heap split into regions
- Young generation (A set of regions)
- Old generation (A set of regions)

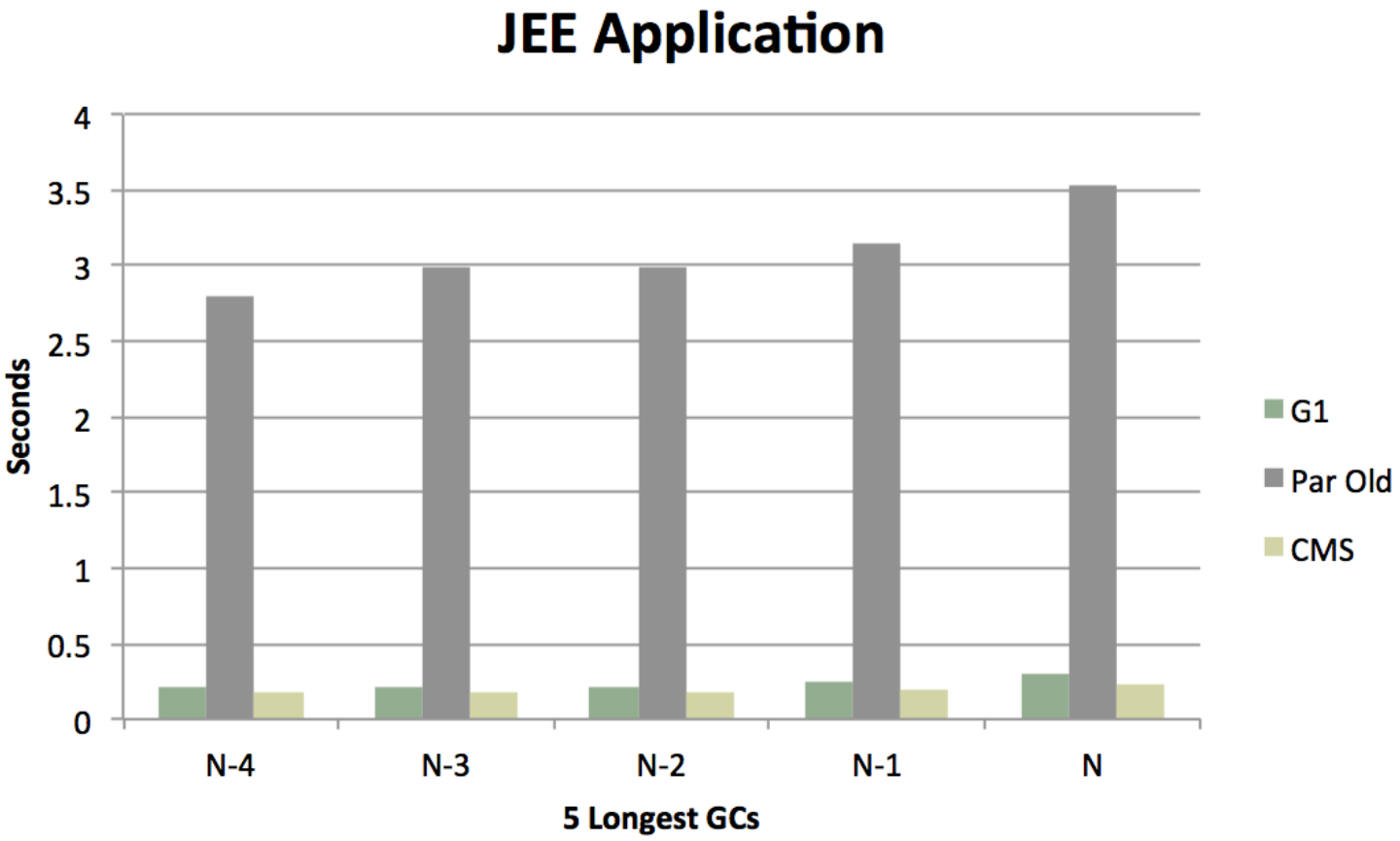
G1 Collector: Parallelism & Concurrency



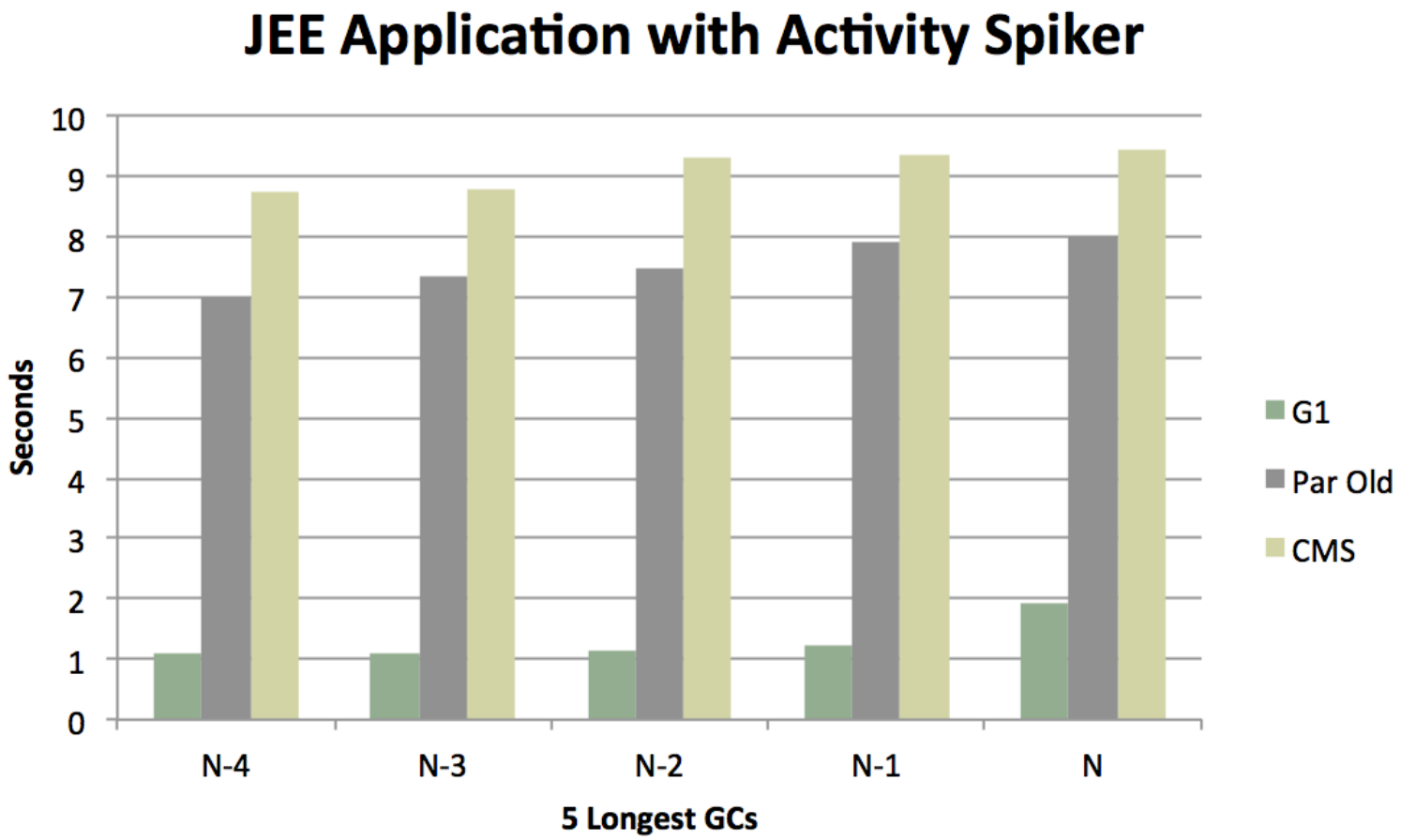
App Thread

GC Thread

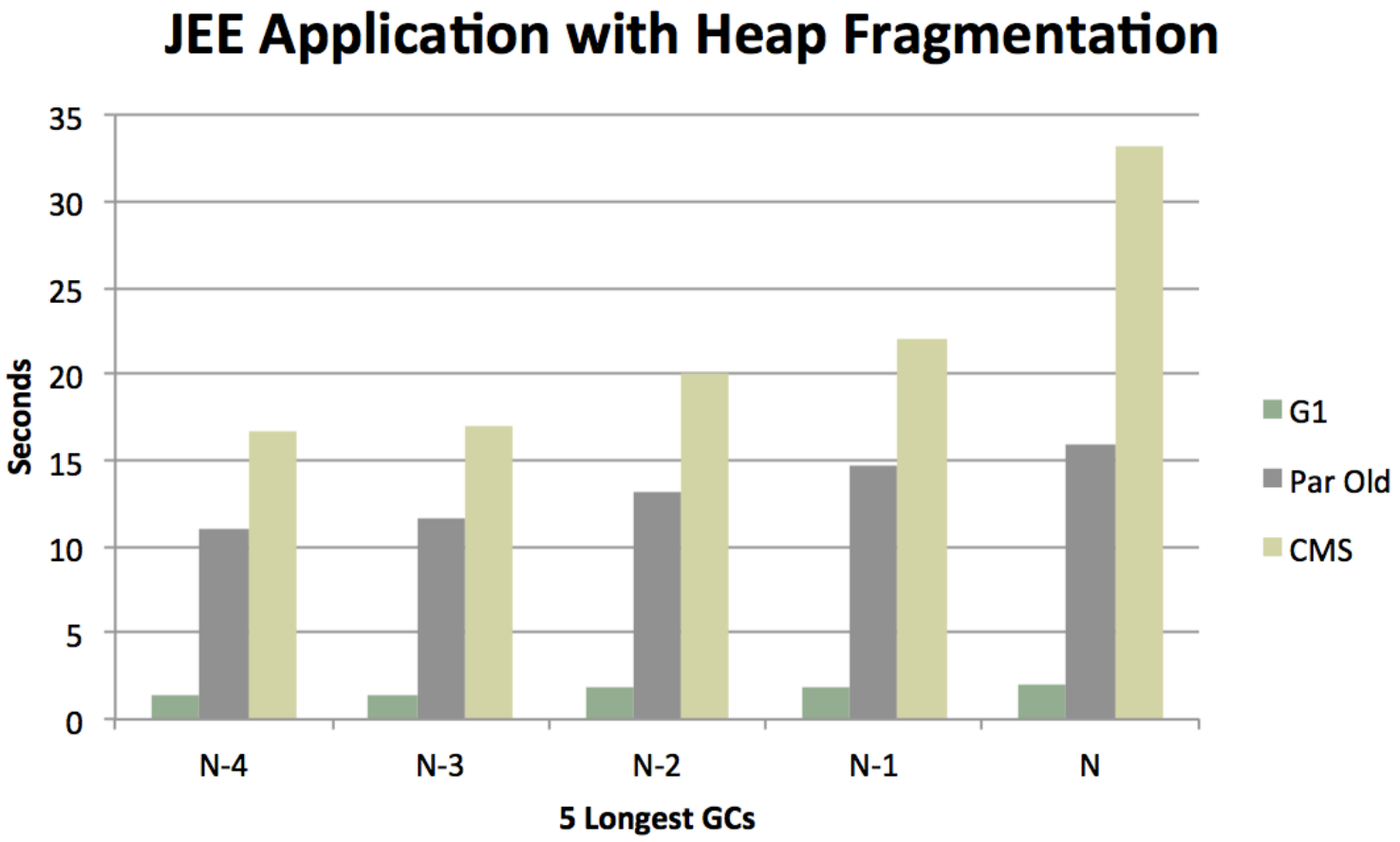
GC Comparison



GC Comparison



GC Comparison



GC Comparison

Collector	2. Garbage creation in test [MB/s]	3. Average New Gen GC Pause [ms]	4. Sigma of New Gen Pauses [ms]	5. Average Old Gen GC Pause [s]	6. Max Old Gen GC Pause [s]	7. Acc. GC Pause time [%]	8. Acc. Concurrent GC time [%]
ParallelGC	30.2	57	46	7.7	9.1	0.59	-
ParNewGC	36	62	45	6.9	7.4	0.73	-
CMS	30,9	78	69	0.65	0.97	0.81	0.06
	51.6	96	35	0.21	0.73	1.6	0.05
G1	50.1	272	41	0.28	0.67	7.0	0.78

http://blog.mgm-tp.com/2013/03/garbage-collection-tuning/

什麼時候該使用G1?

- Large heaps
  - Typically ~6GB or larger heaps
- Applications that needs low pauses

什麼時候不該使用 G1?

- G1需要更多的computing resources和 memory 來運行
- No large heaps
  - Typically smaller than 6GB
- Applications that don't need low pauses

SUMMARIZE

Summarize

- Professional developer 必須非常了解 variables, objects 的 life cycle, objects 何時有資格被會收
- GC will hurt system performance
- Must monitor GC activities
- Must enable GC logging

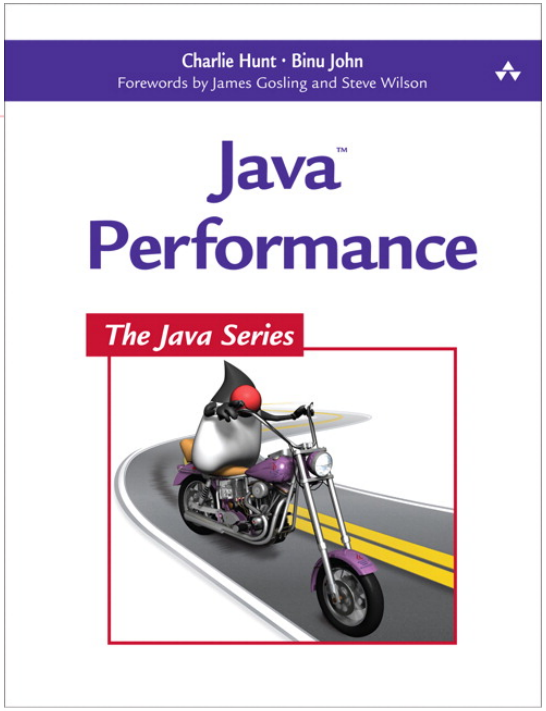
Tuning Procedures

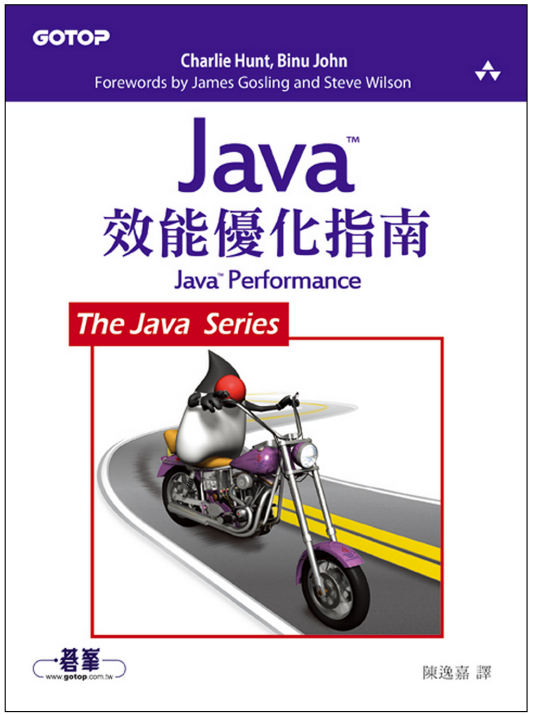
- Observer LDS to decide initial size setting
- Use parallel GC first
- If the GC pause time can't be accepted, then trying to use CMS
- If heap is over 6G, and look for lower down pause time, you can consider G1
- Monitoring -> Analyzing -> Tuning -> Monitoring -> …

REFERENCES

References

- Step-by-Step: Garbage Collection Tuning in the Java HotSpot™ Virtual Machine – JavaOne 2010
- The Garbage Collection Mythbusters – JavaOne 2010
- GC Tuning for the HotSpot JVM – JavaOne 2009
- uPortal Performance Optimization. Faizan Ahmed. Architect and Engineering Group. Enterprise Systems & Services. RUTGERS. faizan@rutgers.edu.
- http://java.sun.com/javase/technologies/hotspot/gc/gc_tuning_6.html

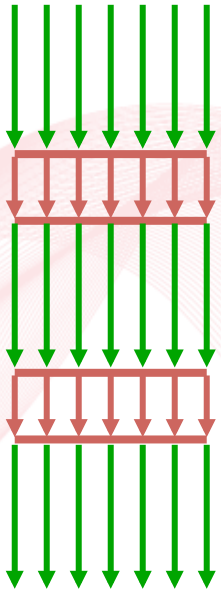




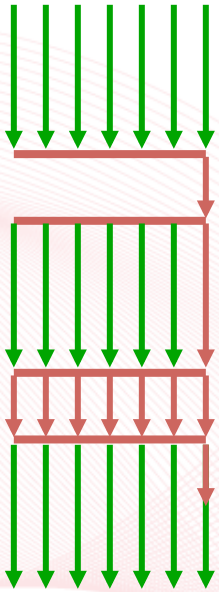
- **Java Performance –**
  Charlie Hunt, Binu John
- **HotSpot GC Tuning Guide**
  for Java SE 6
  http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
- **Memory Management in the Java HotSpot VM**
  http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf
- **Java 6 Performance Whitepaper**
  http://java.sun.com/performance/reference/whitepapers/6_performance.html

Q&A

Parallel



Concurrent (CMS)



G1