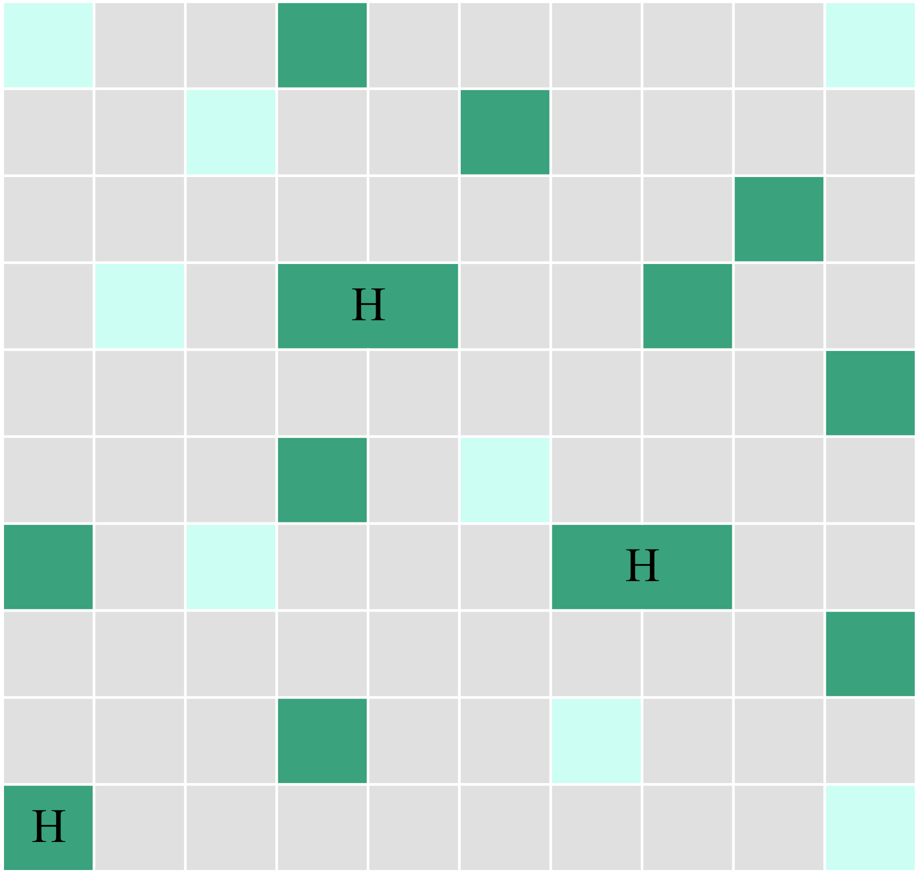
HOL: Garbage-First Collector Tuning

Monica Beckwith

G1 GC Regions

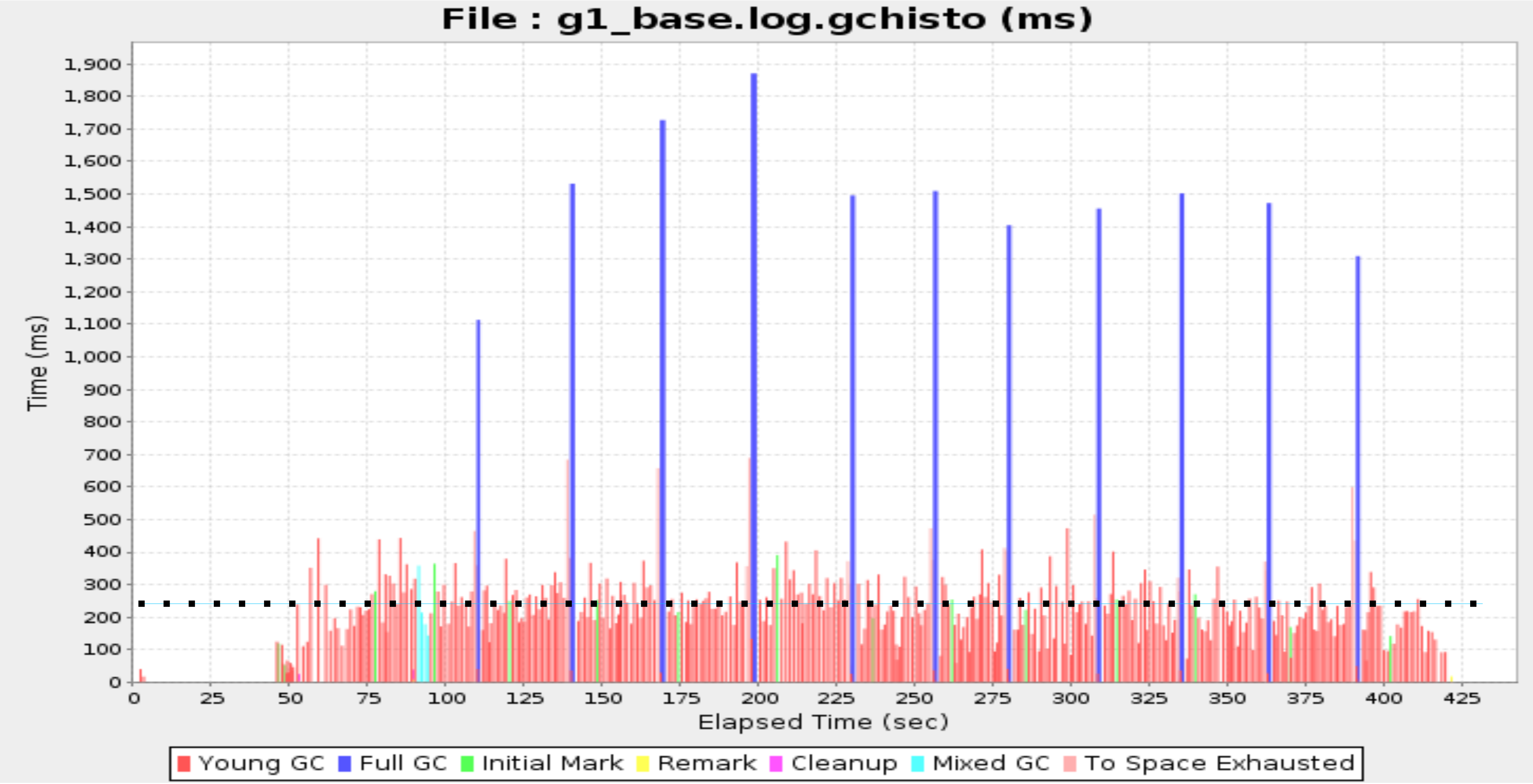


- H Humongous Regions
- 🟩 Old Regions
- □ Young Regions

Young Collection – Ergonomics and Adaptability

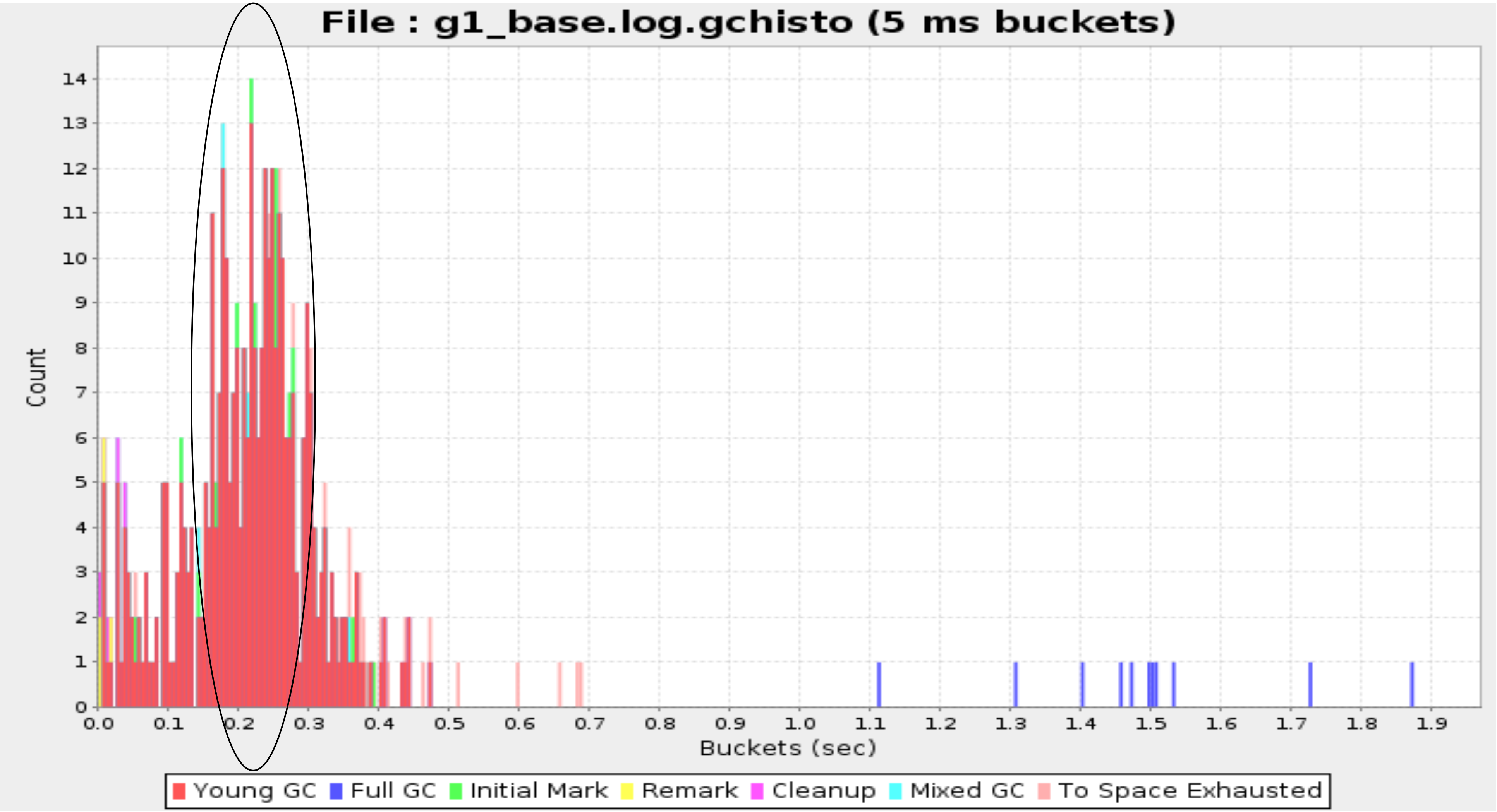
- Young generation size is based on your pause time target and internally set min and max bounds
  - -XX:MaxGCPauseMillis = 200 (default)
  - Default min nursery size = 5% of your Java heap
  - Default max nursery size = 60% of your Java heap
- Prediction logic
  - Determines how much time it will take to collect 1 region
  - (Re-)Sizes the young generation accordingly after each collection

Sample GCHisto Timeline Plot

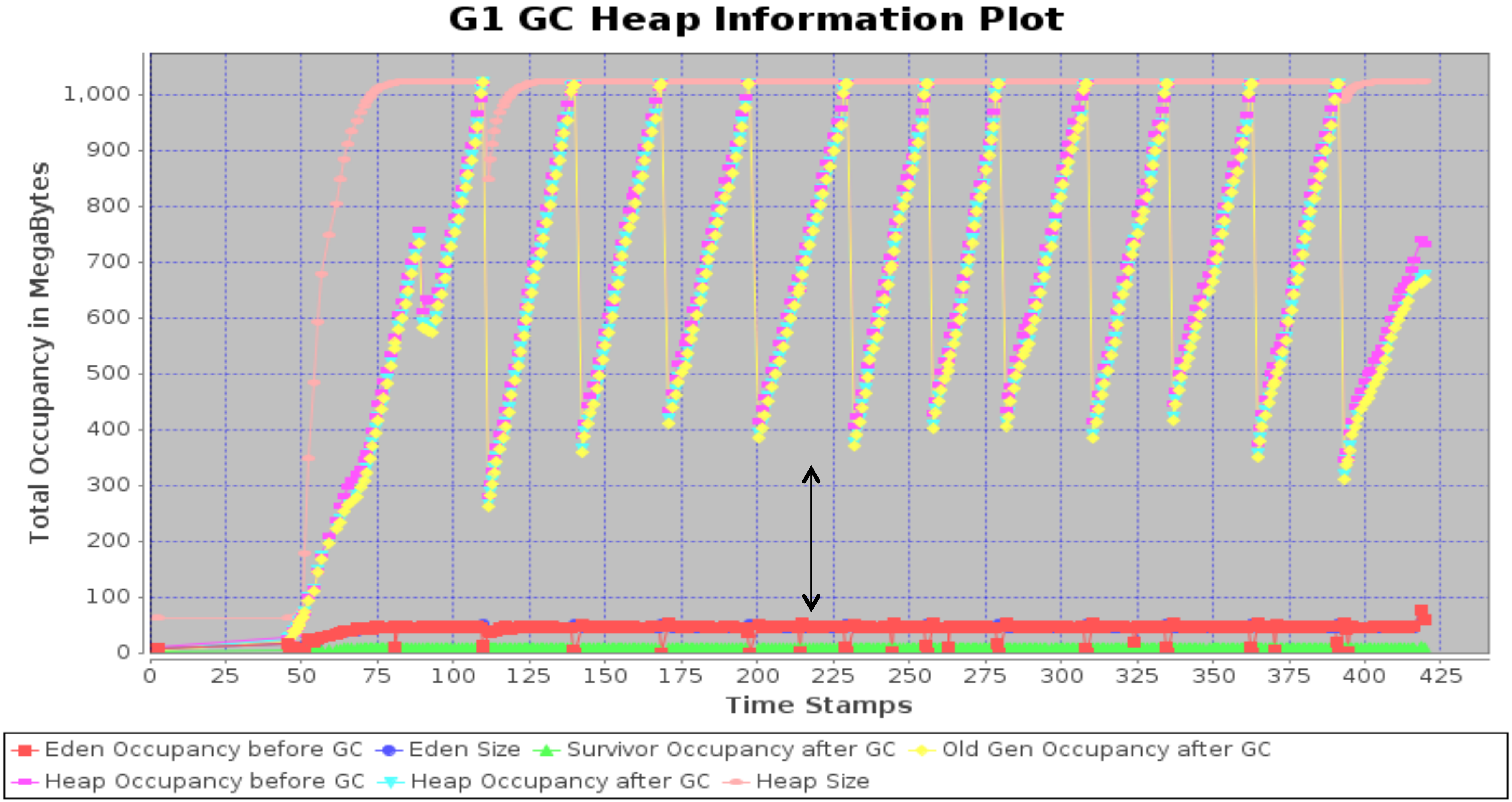


Servergy
Save Energy. Work Smart.

Sample GCHisto Pause Distribution Plot



Sample Heap Information Plot



How to Increase Max Limit on Nursery?

-XX:MaxNewSize=800m

Marking Threshold and Concurrent Cycle

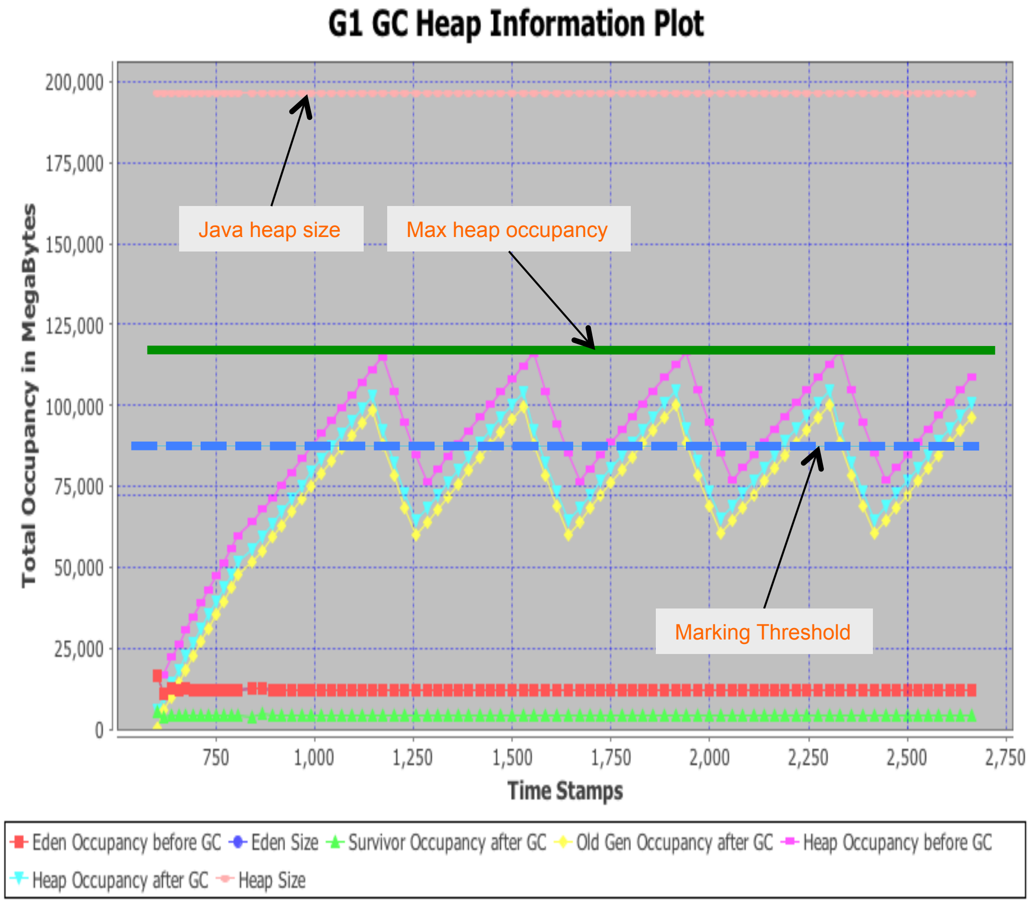
- Threshold default: 45% of your Java heap
  - -XX:InitiatingHeapOccupancyPercent=<value>
- When threshold's crossed, G1 starts a concurrent cycle
  - Some phases are concurrent and some are stop-the world
  - Multi-phased concurrent marking cycle finds the "best" regions to be collected
  - Live-ness accounting

Marking Threshold and Concurrent Cycle

- After the marking phase is complete, G1 has information on which old regions to collect
  - Regions are ordered based on "collection efficiency"
    - Expensive regions would be regions with lots of live data and large RSets
  - Completely free regions are collected during cleanup phase

```
6530.615: [GC cleanup 13G->12G(18G),
0.0388540 secs]
```

Marking Threshold – Example 1 (non-lab)

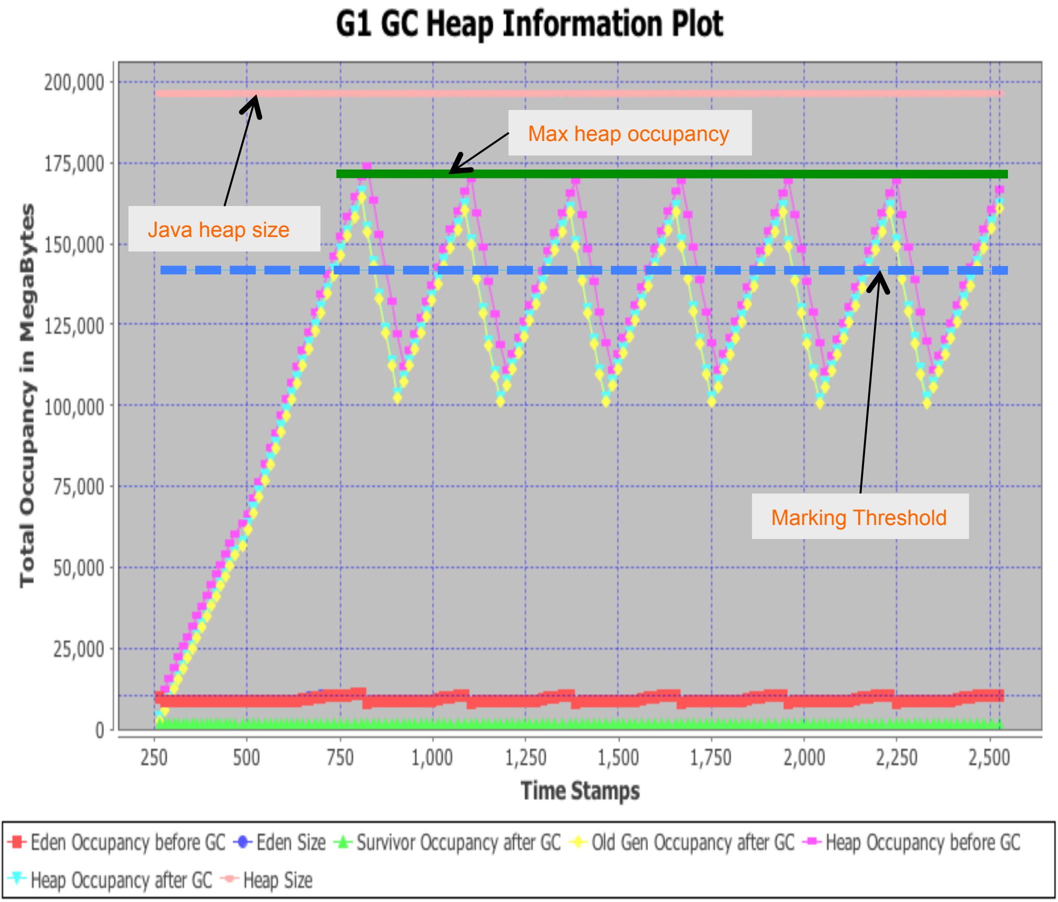


Java heap size

Max heap occupancy

Marking Threshold

Default IHOP



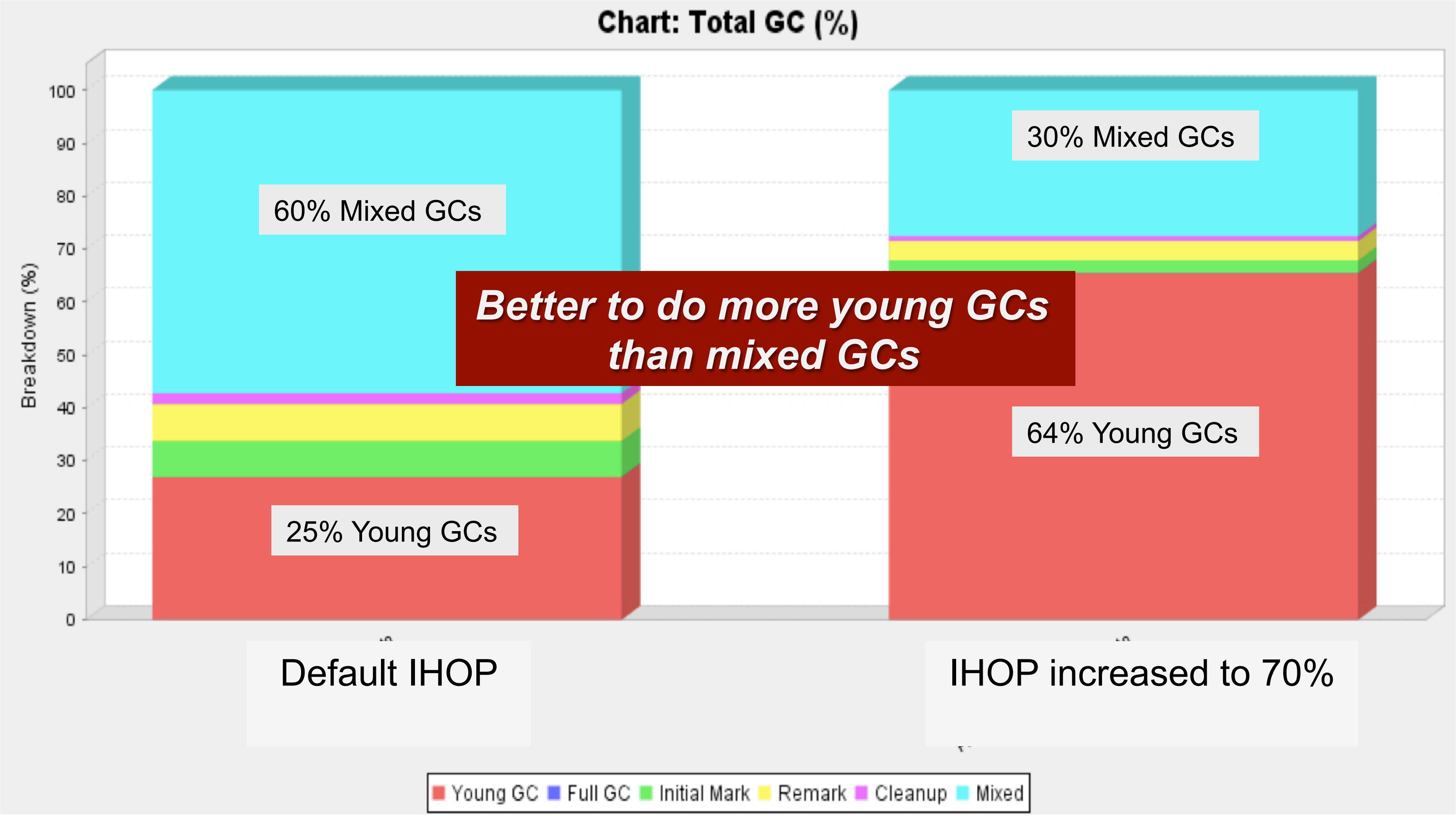
Max heap occupancy

Java heap size

Marking Threshold

IHOP increased to 75%

Marking Threshold – Example 2 (non-lab)



60% Mixed GCs

25% Young GCs

30% Mixed GCs

64% Young GCs

Better to do more young GCs than mixed GCs

Default IHOP

IHOP increased to 70%

How to Increase the Marking Threshold?

```
-XX:InitiatingHeapOccupancyPercent=55
```

Taming Mixed GCs

- Adjust -XX:G1HeapWastePercent
  - Defaults to 10% of your Java heap
  - Lower value means you are willing to collect expensive regions during your mixed collection.
  - Higher value means that you are willing to "waste" that much heap.

Evacuation Failures

- Evacuation failures indicate that G1 ran out of heap regions either –
  - while copying to survivor regions or
  - while promoting or copying live objects in-to the old generation
- Prior to Java 7u40 evacuation failures shown as a “to-space overflow” in the GC logs
- Java 7u40 onwards shows “to-space exhausted” in the GC logs

Evacuation Failures – How to Avoid Them?

- Get a baseline with bare minimum options:
  - -Xmx, -Xms and -XX:MaxGCPauseMillis=<value>
  - Over-tuning is NOT for G1

Evacuation Failures

- Plot the heap utilization stats from the log
  - Marking threshold too high?
    - Can’t keep up with promotions
  - Marking threshold too low?
    - Not reclaiming much space from marking cycle
- Concurrent cycles taking a long time to complete?
  - Increase the thread count: ConcGCThreads

Evacuation Failures

- Sometimes survivor space gets exhausted
  - Increase the G1ReservePercent
    - It’s a false ceiling
    - Defaults to 10
    - G1 will cap it off at 50%

So, Let's Get It Started!

* Remember to increase the thread count to 750.

Lab 4 Tuning

Tuning parameters recommendation for Lab 4:

-Xms == -Xmx

-XX:MaxNewSize=800m

kirk@kodewerk.com

monica@beckwithclan.com