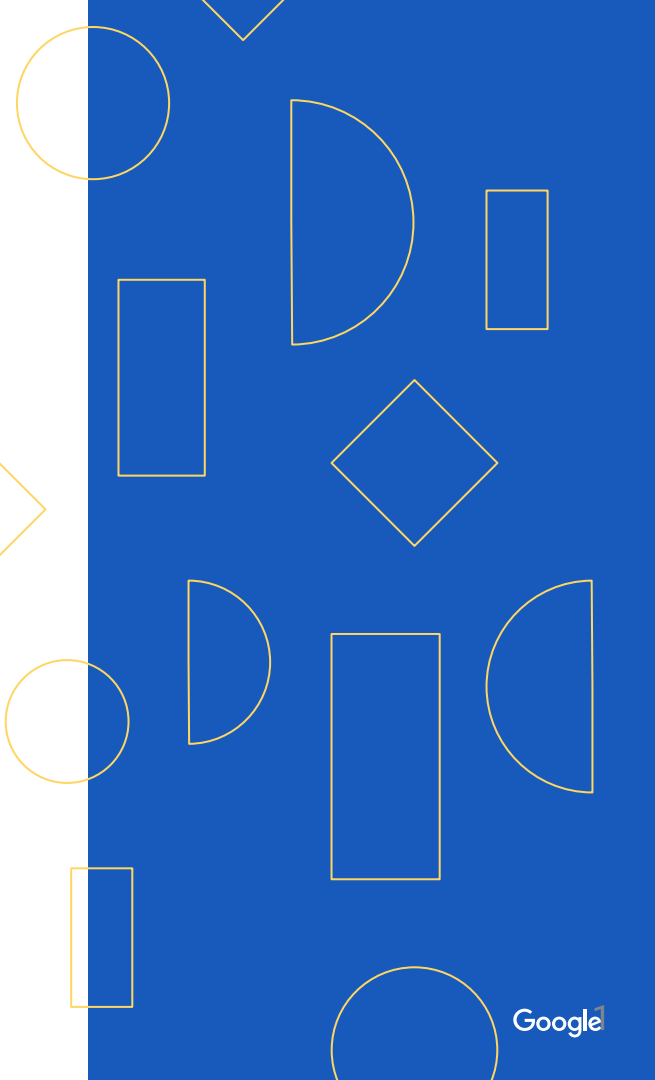
No More Xmx!

Adaptable Heap Sizing (AHS) for Containerized Java Applications

Jonathan Joo



About me!

- Senior SWE @ Java Platform Team at Google (US)
- Prior: Distributed Filesystems @ Rubrik

✉ jonathanjoo@google.com

in/jonathan-joo

Java Platform Team

- Making sure Java "just works"
- Custom Google fork of the OpenJDK
- Handle upgrades and improvements
  - ex. 8 -> 11
  - Local updates to our JDK

Agenda

01 Background

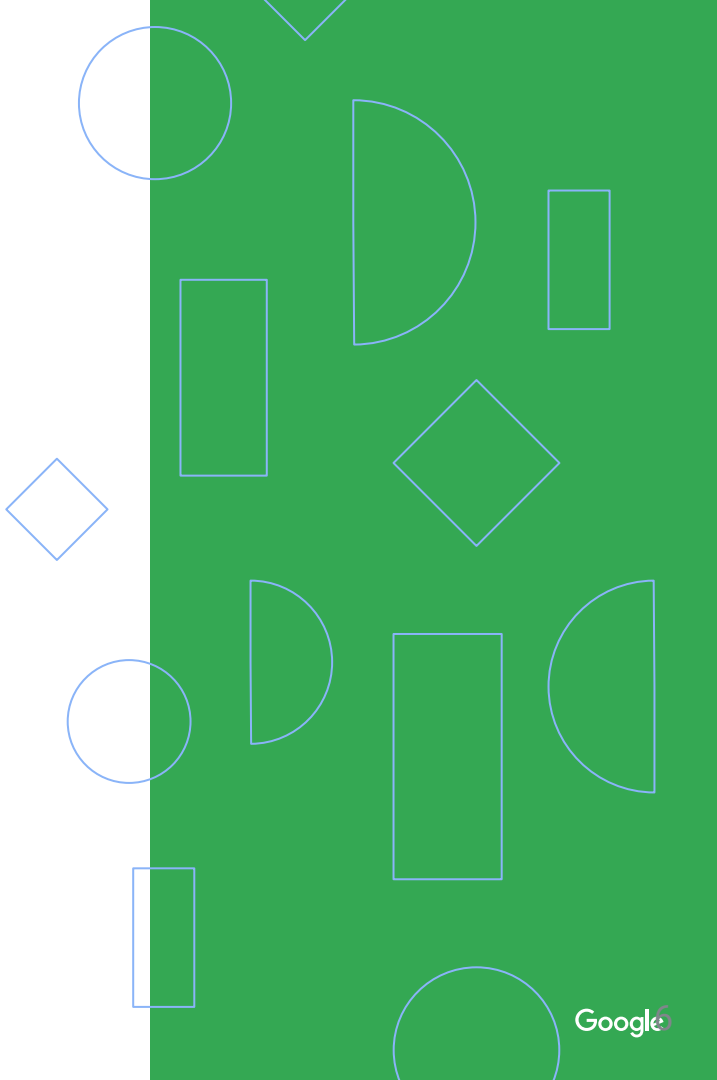
02 Implementation of AHS

03 Benefits of AHS

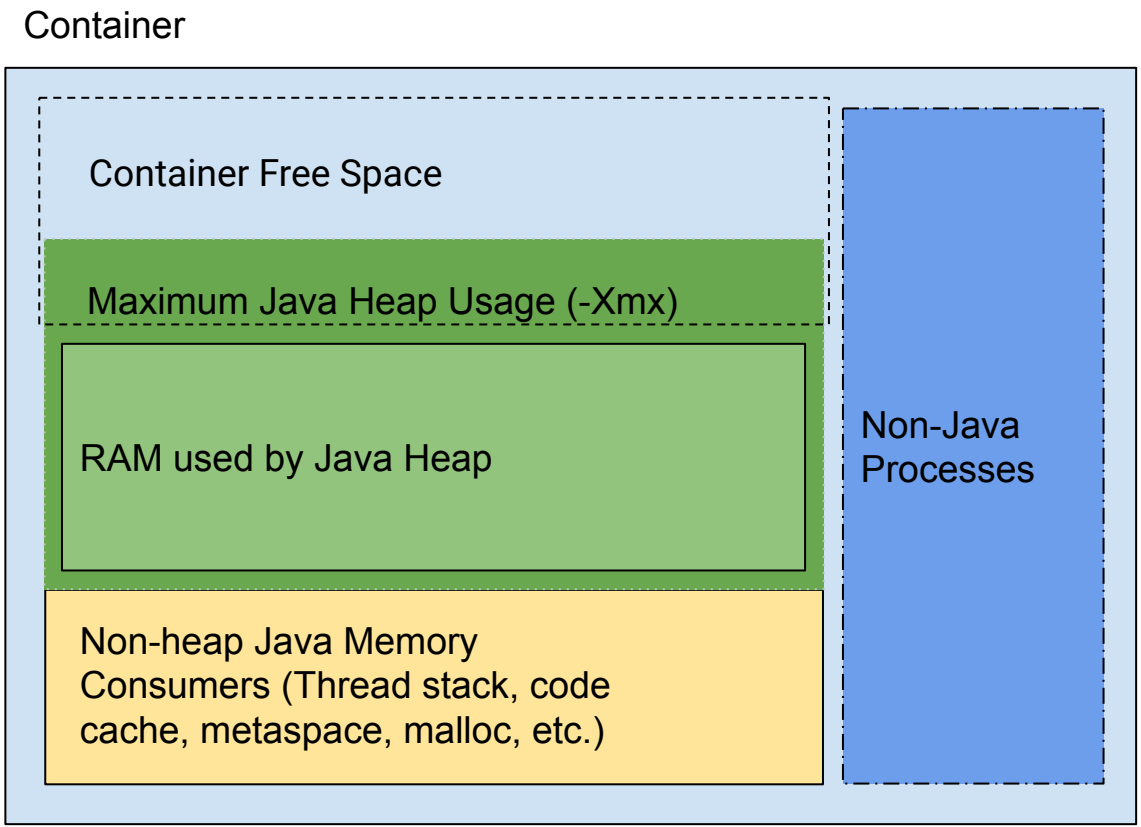
04 Looking forward

Note: AHS = Adaptable Heap Sizing

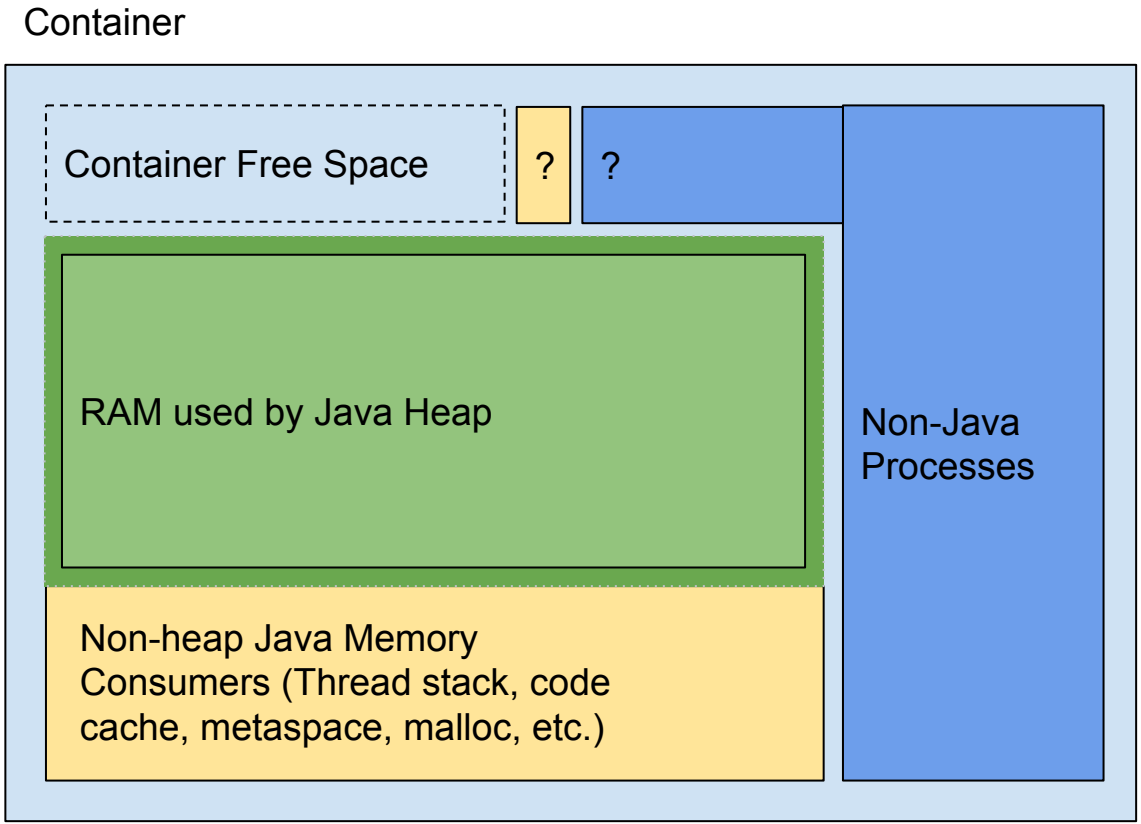
Background



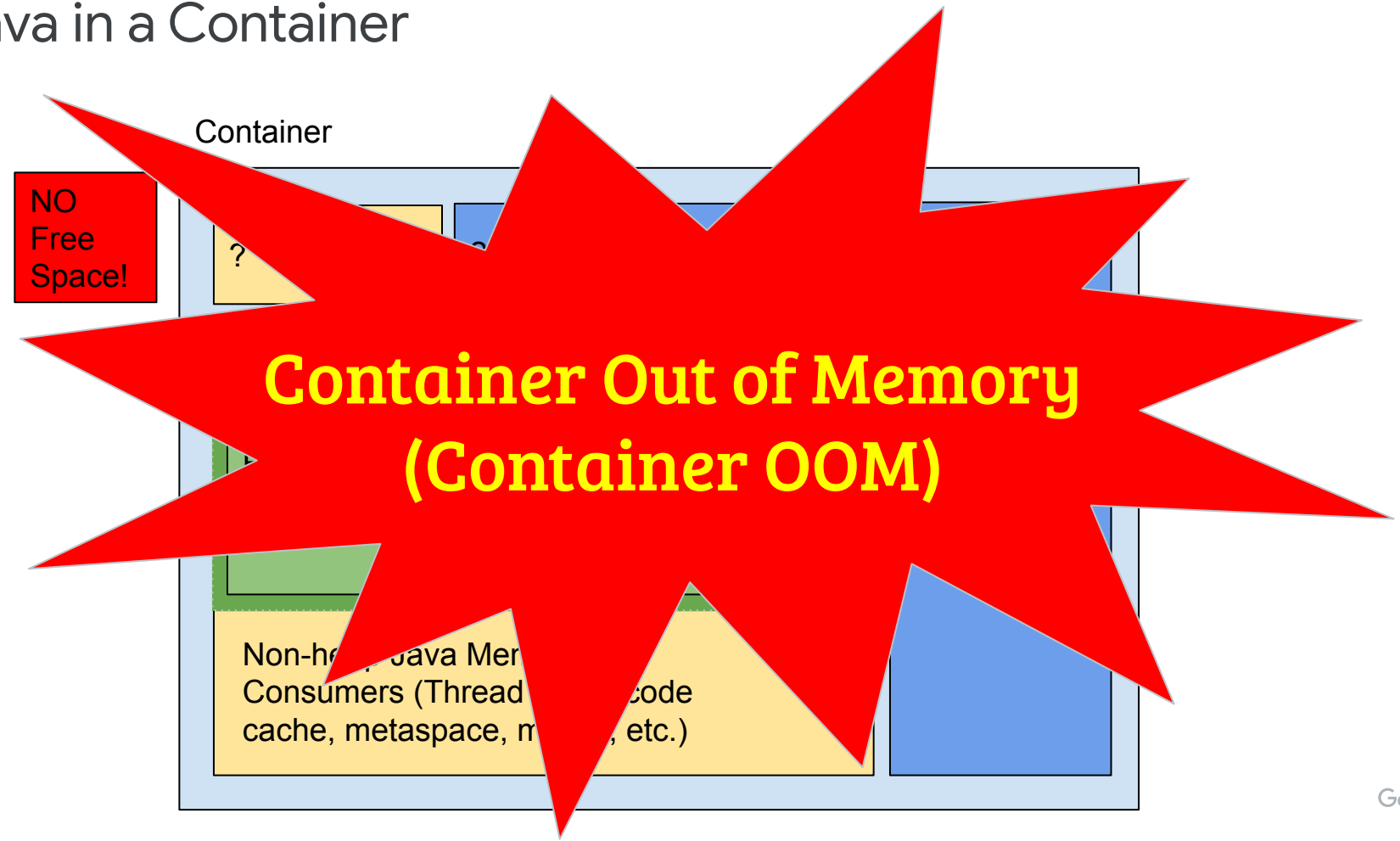
Java in a Container



Java in a Container



Java in a Container



Why is this such a common issue?

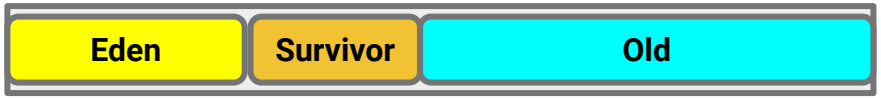
Java Configuration @ Google

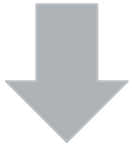
https://pixabay.com/photos/greece-parthenon-temple-ruins-1594689/

Java Configuration @ Google

JDK8

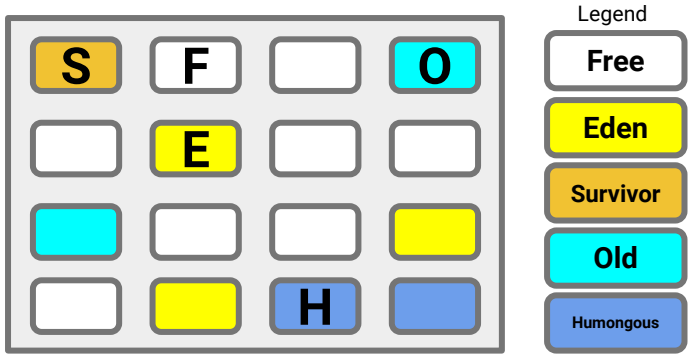
Concurrent Mark Sweep (CMS)





JDK11

Garbage First (G1)



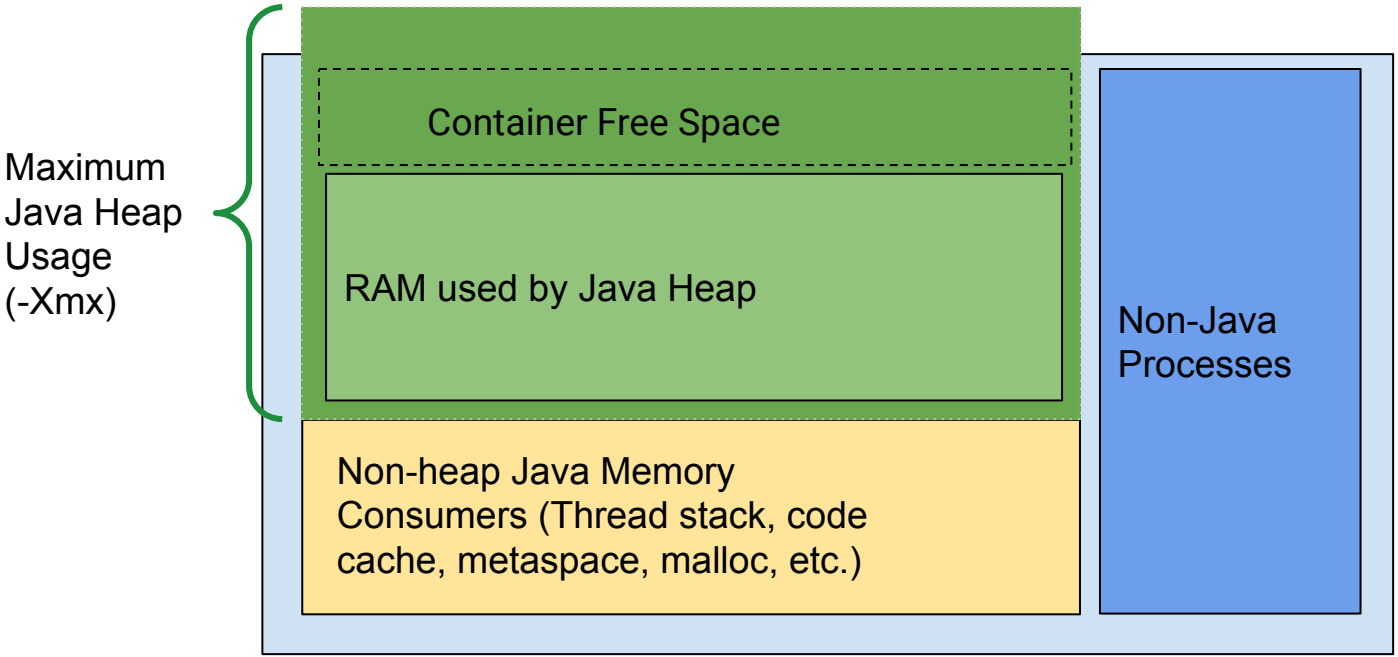
Java Configuration @ Google

- G1 differences:
  - Heap regions non-contiguous
  - Performs more compaction
  - Generally higher throughput
  - Uses more non-heap memory
  - **Uses heap more greedily**

A problem:

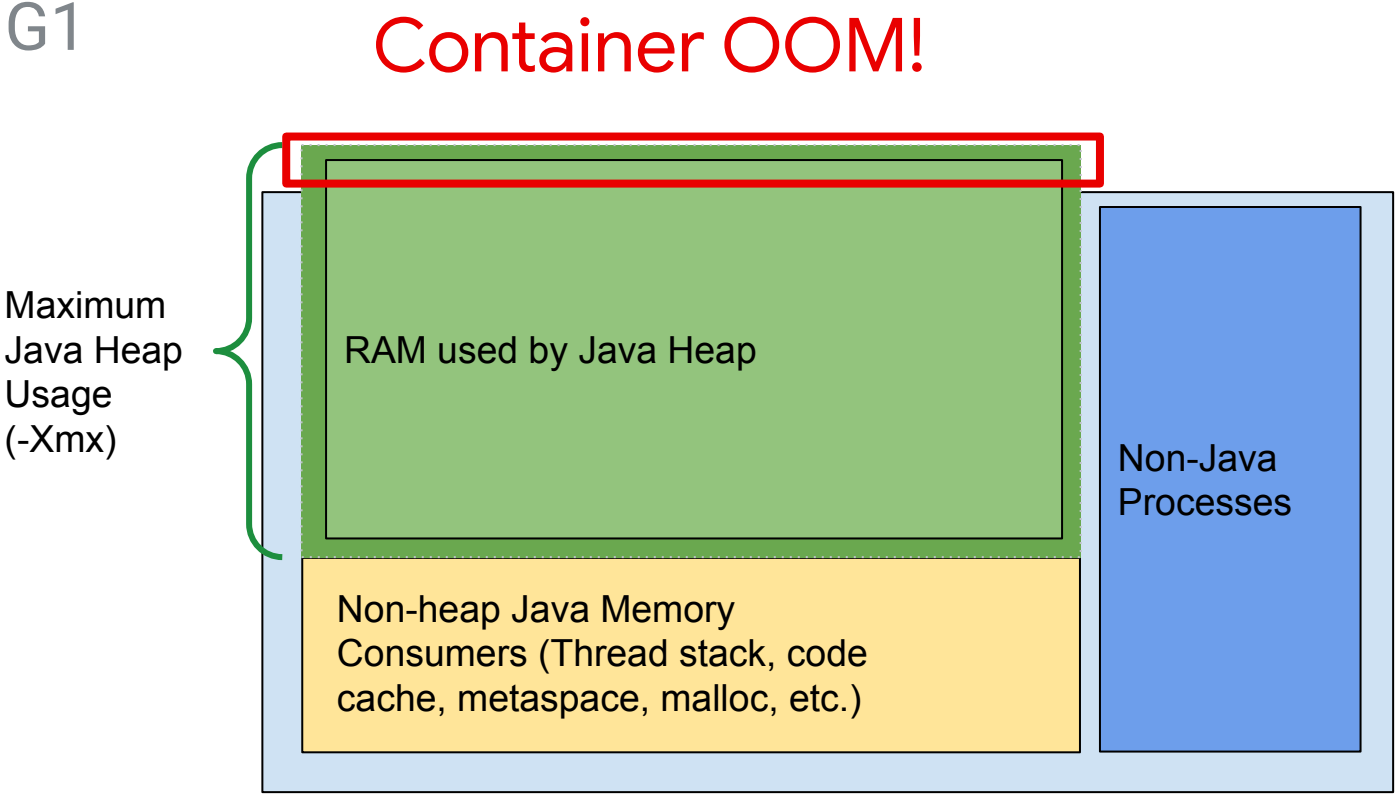
Java in a Container

CMS



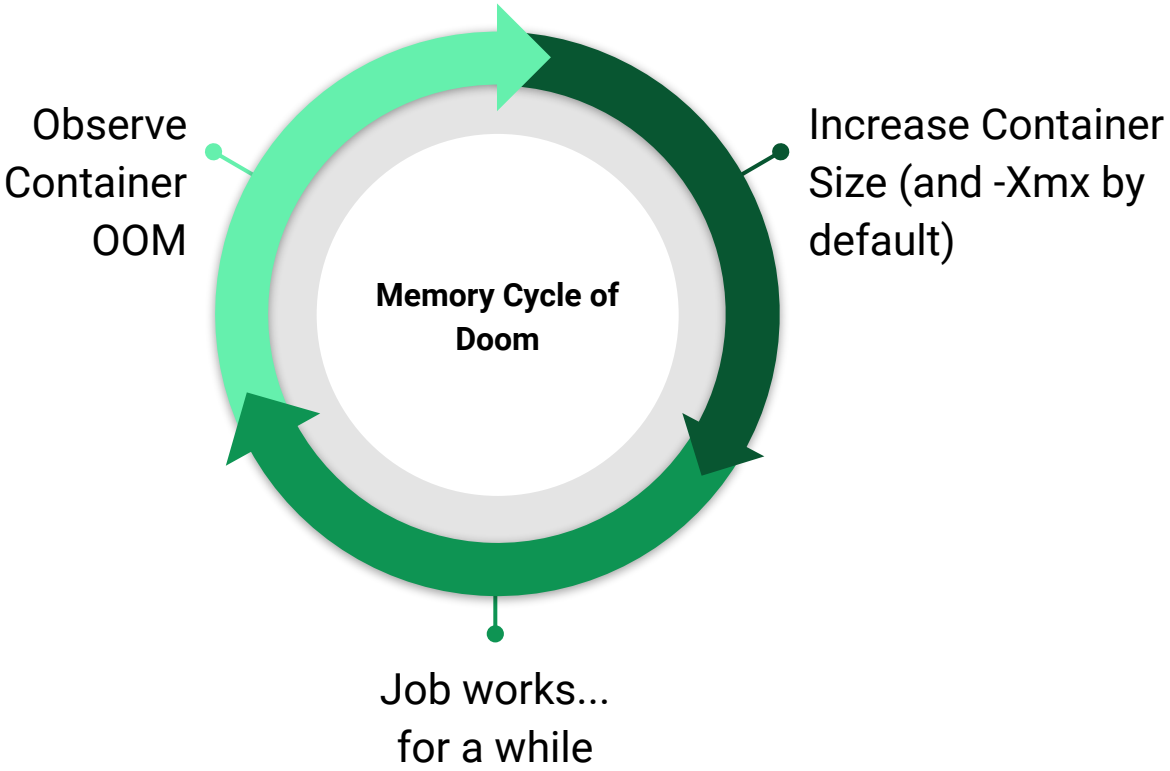
Java in a Container

G1



Another common problem:

Antipattern



Solution:

Solution:

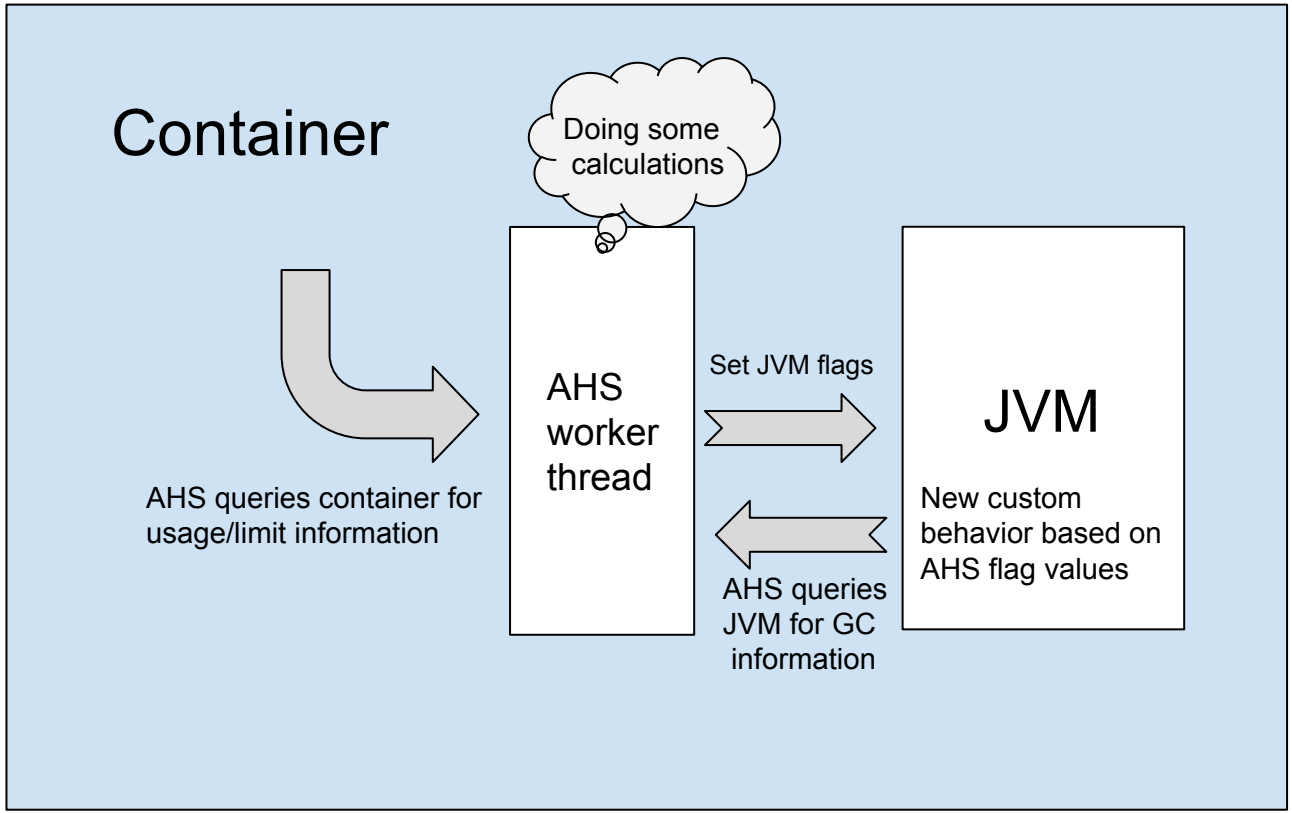
https://pixabay.com/vectors/superman-logo-superman-superherc

AHS Goals

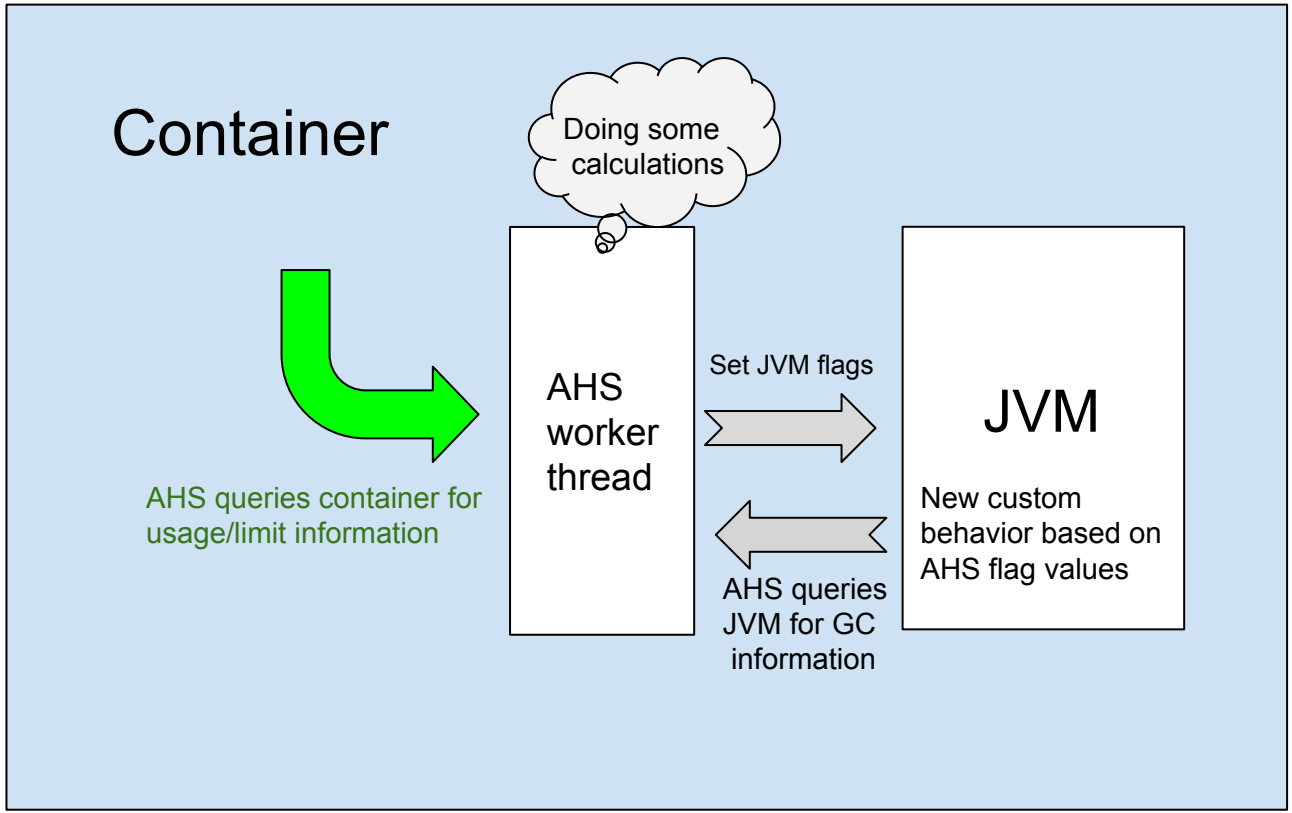
- Don't have to set a "correct" Xmx.
- Prevent container OOM if possible.
- Prevent excessive heap usage.

Implementation of Adaptable Heap Sizing

High Level Overview

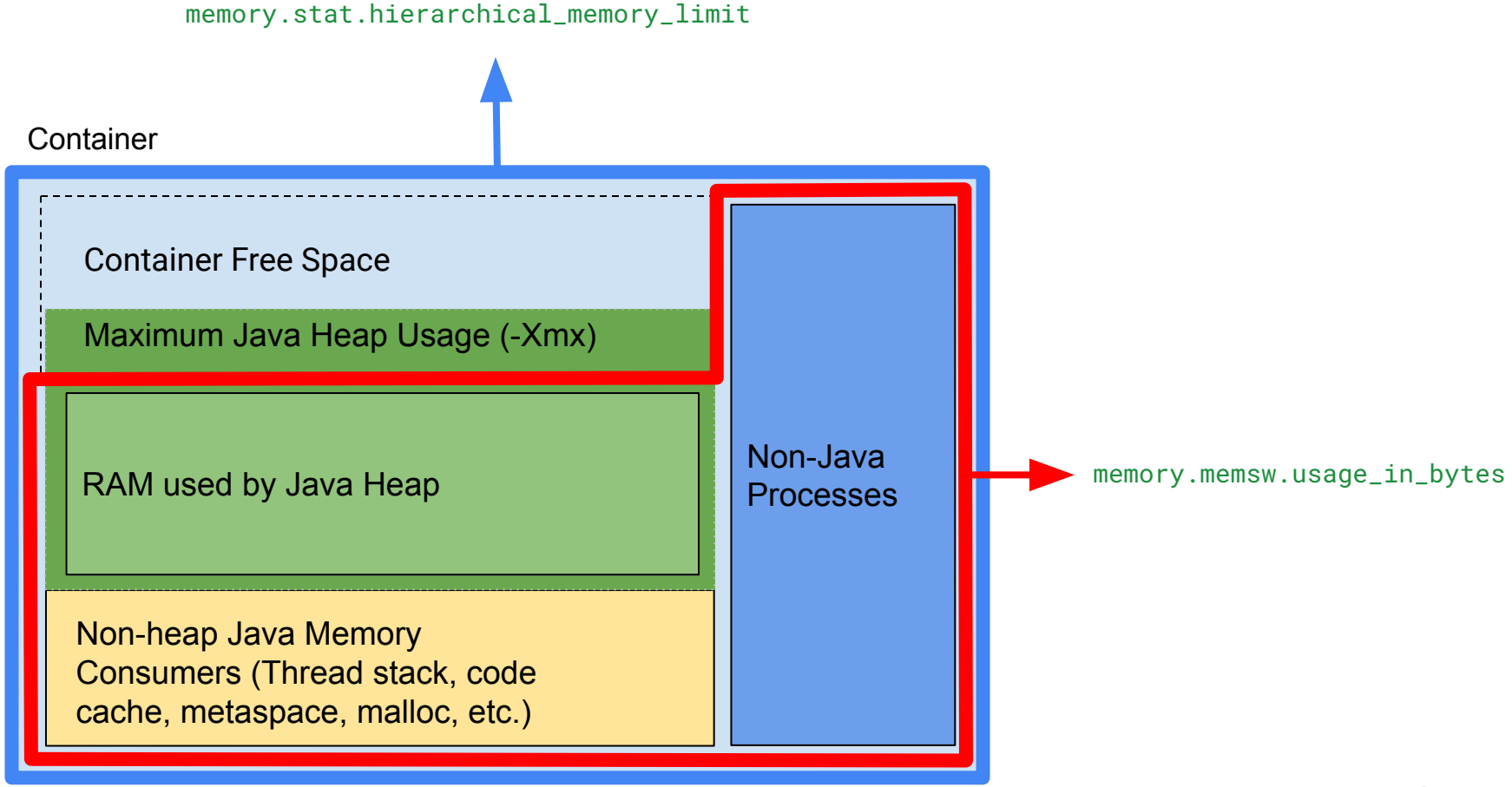


Step 1: Obtaining Container Information

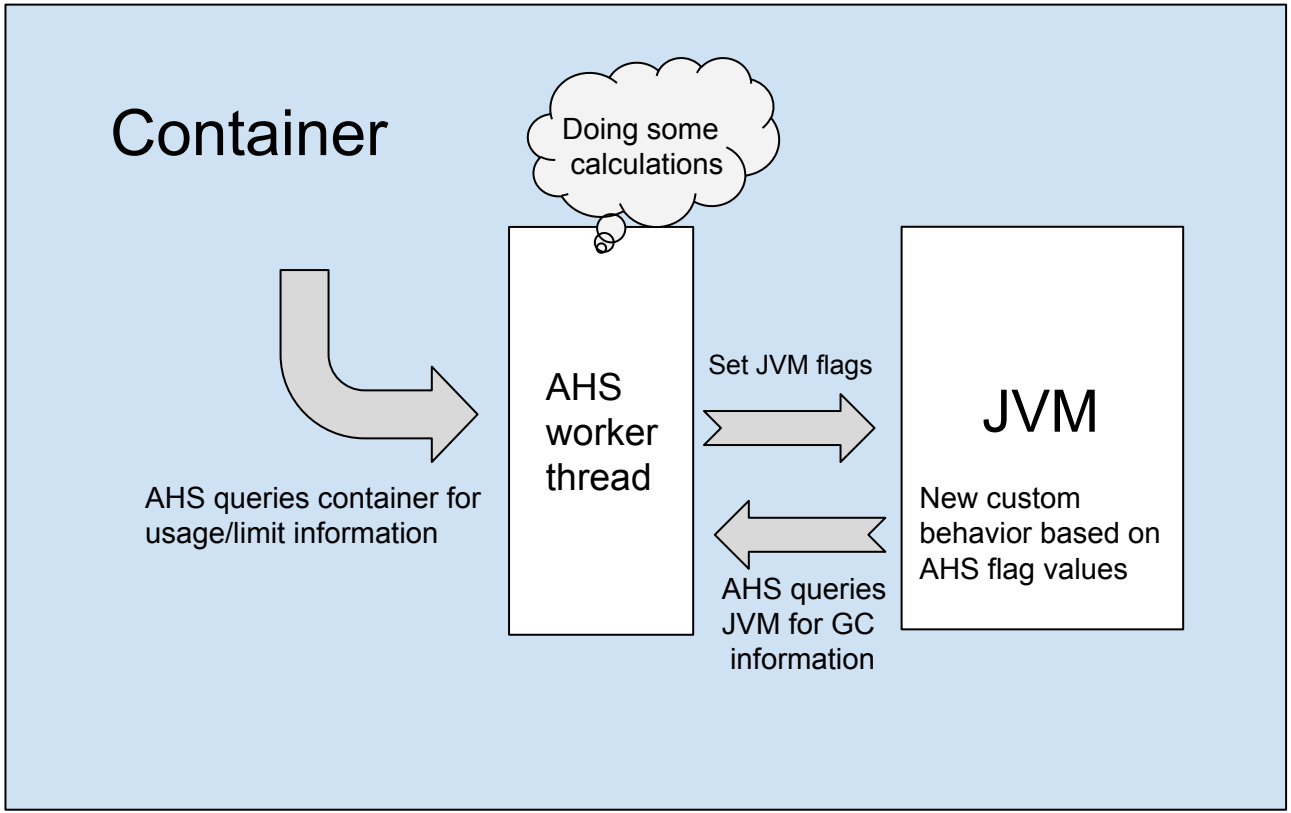


Step 1: Obtaining Container Information

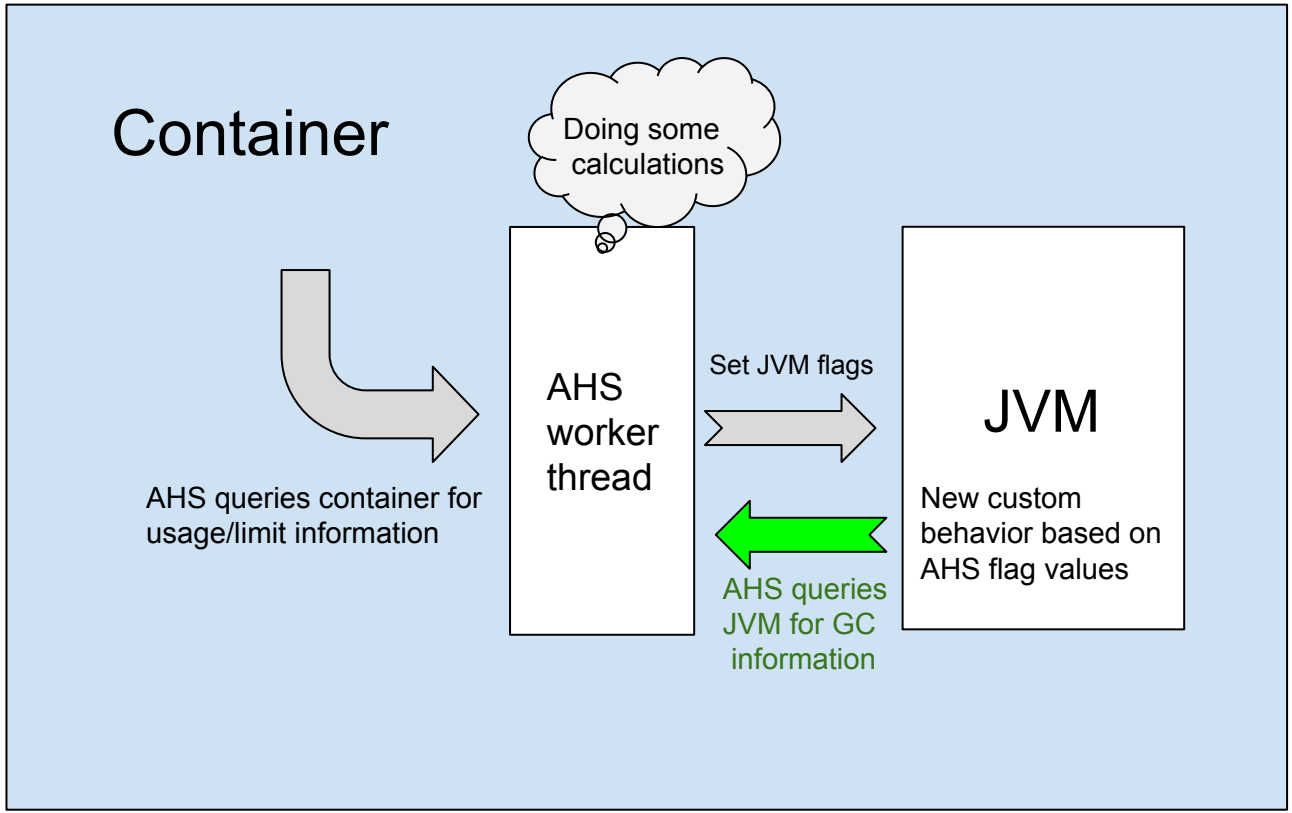
- Container information is obtained by querying `cgroup` information
  - Container Limit
  - Container Size
- This gives us a snapshot of what the container looks like at a given point in time.


memory.stat.hierarchical_memory_limit
Container
Container Free Space
Maximum Java Heap Usage (-Xmx)
RAM used by Java Heap
Non-heap Java Memory Consumers (Thread stack, code cache, metaspace, malloc, etc.)
Non-Java Processes
memory.memsw.usage_in_bytes

Step 2: Obtaining GC Information



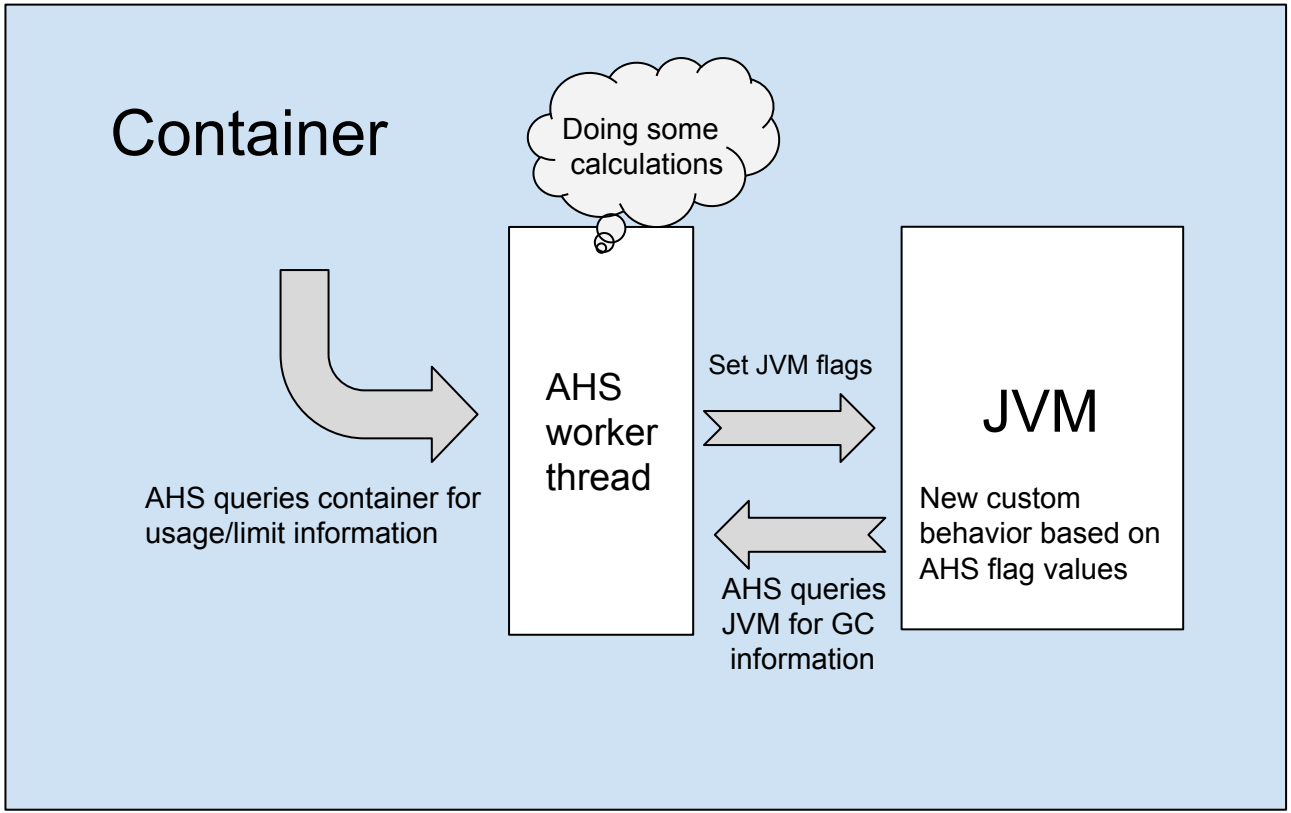
Step 2: Obtaining GC Information



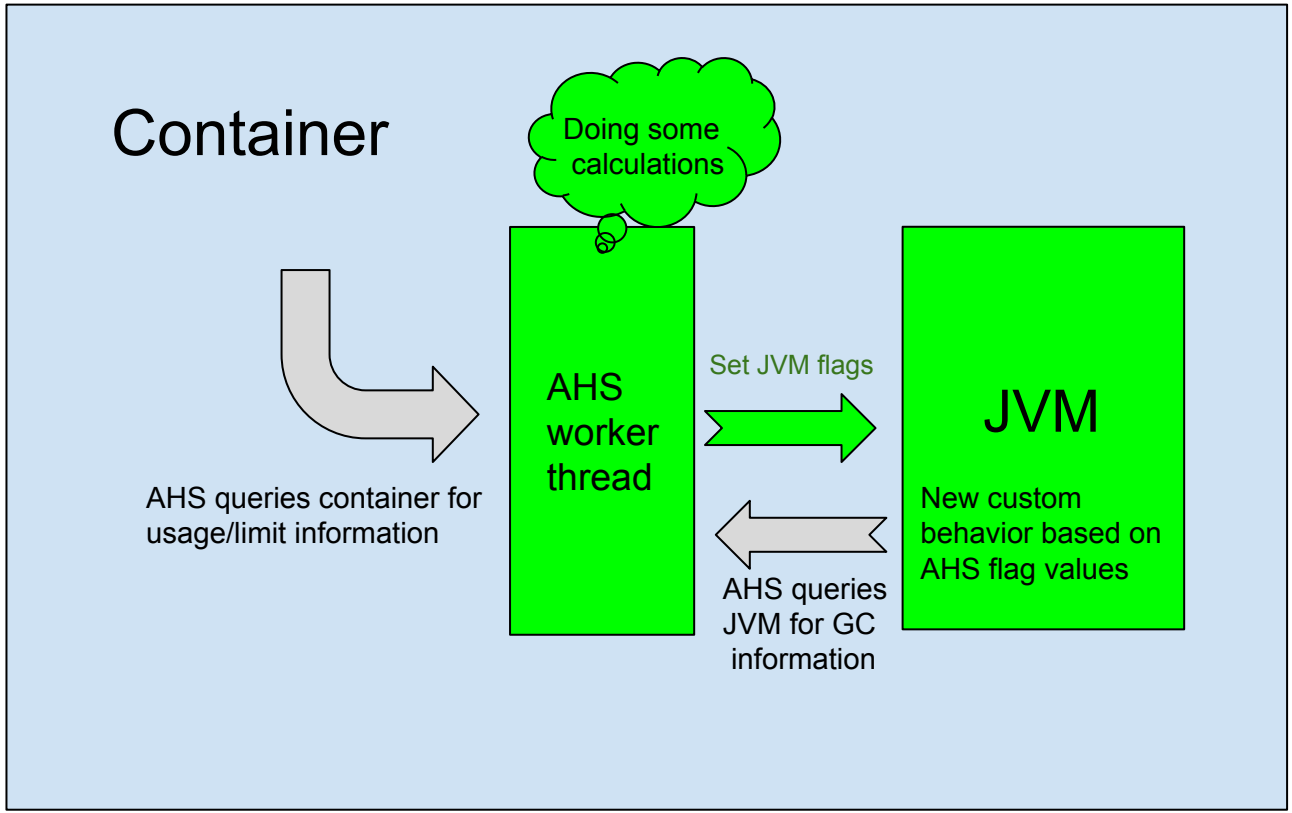
Step 2: Obtaining GC information

- GC information is obtained by reading from the `HSPerfData` file.
- HsPerf file is a custom-formatted binary file written by the JVM.
  - Used for JVM monitoring and performance testing.
- We track GC time information as the JDK runs, as part of a Google-local patch.
- This GC information is then written to the HSPerf file.
- The GC information is then read by the AHS worker thread.

Step 3: AHS Worker Thread



Step 3: AHS Worker Thread



Step 3: AHS Worker Thread

- Lightweight Native thread (C++).
- External-to-JVM thread enabled at runtime.
- Periodically queries:
  - Container RAM information.
  - JVM GC information.
- Responsible for informing JVM about how to react.

Step 3: AHS Worker Thread

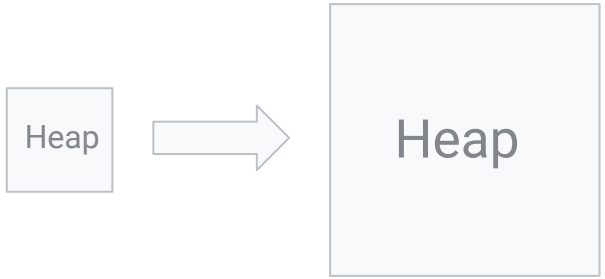
- Introduce two new `manageable` JVM flags:
  - `ProposedHeapSize`
  - `CurrentMaxExpansionSize`
- What is a `manageable` flag?
  - Can be changed dynamically at runtime.

ProposedHeapSize

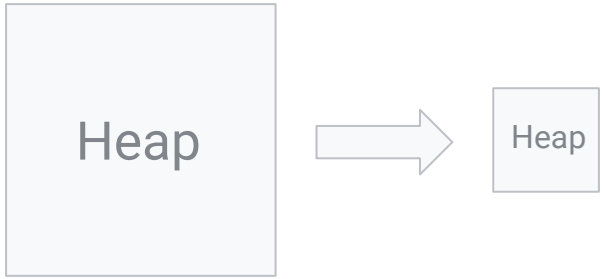
- AHS has a target GC CPU overhead percentage.
- ProposedHeapSize helps achieve target GC CPU overhead.
- High GC CPU overhead = Larger ProposedHeapSize
- Low GC CPU overhead = Smaller ProposedHeapSize
  - The bigger the heap, the less amount of time spent doing GC.
- Is a soft target.

ProposedHeapSize

GC 🔥 =



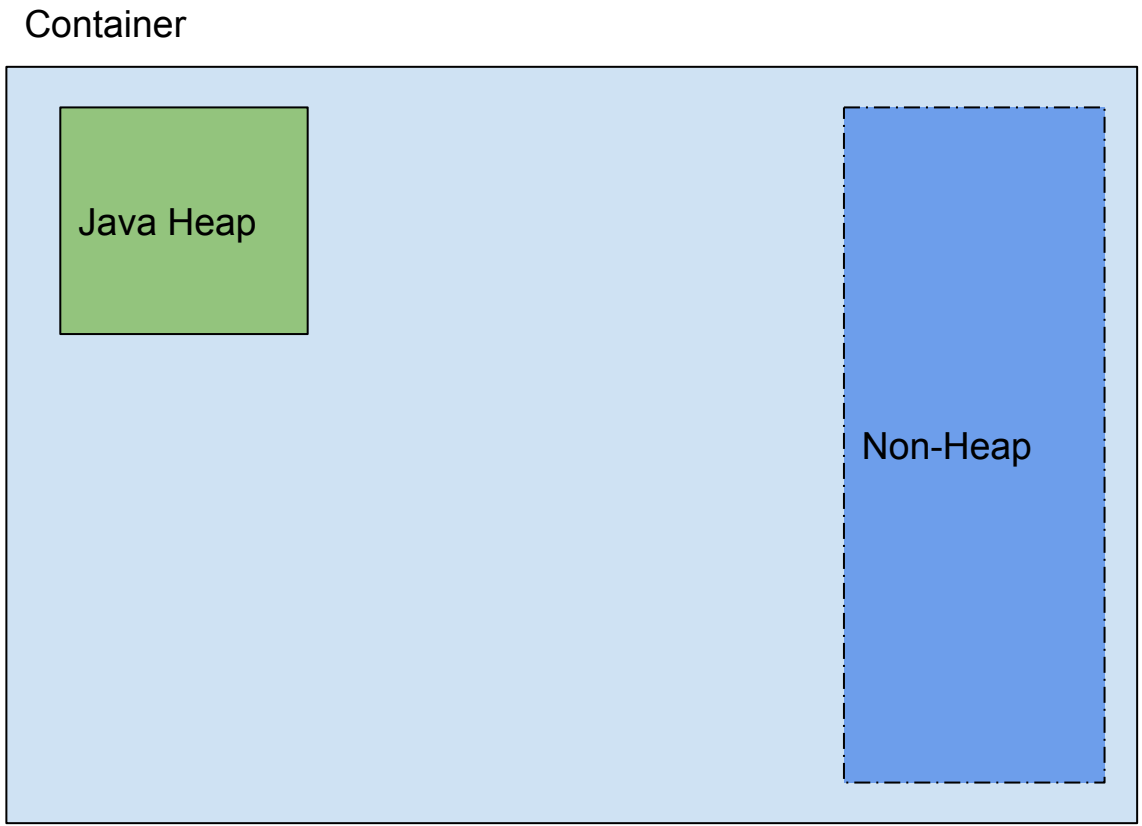
GC 😴 =



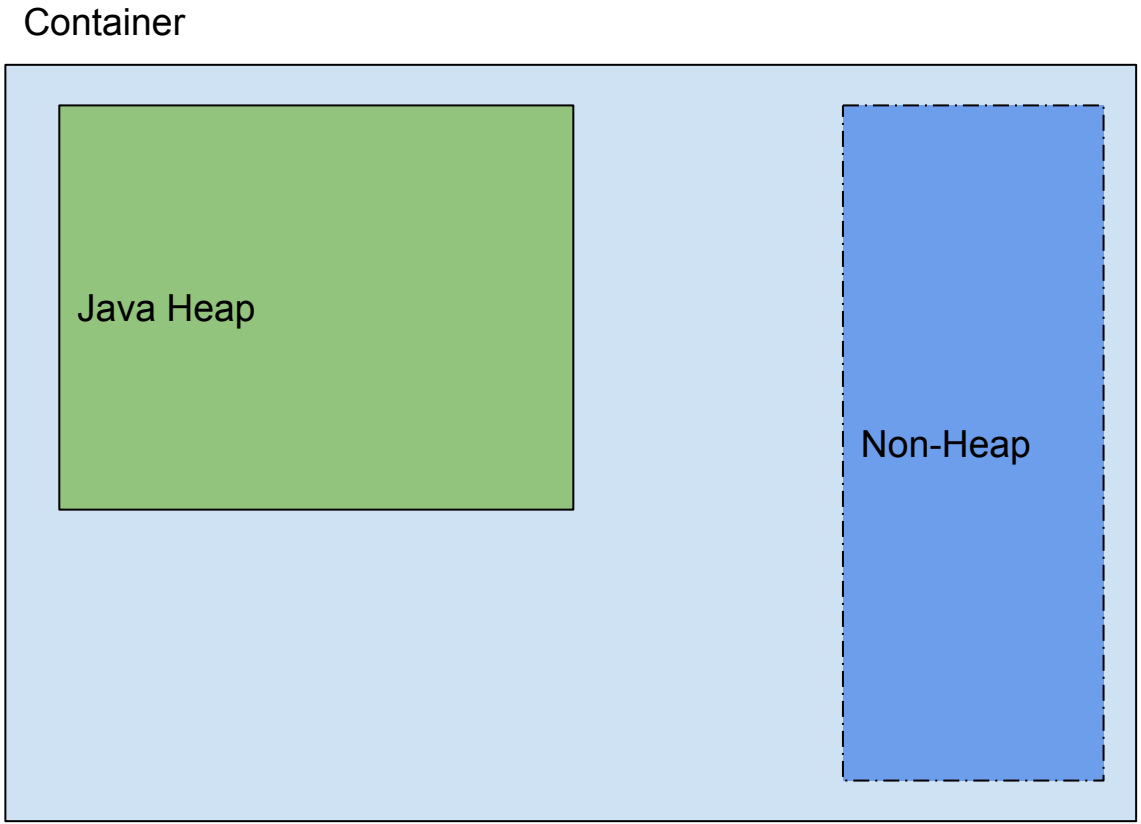
`CurrentMaxExpansionSize`

- `CurrentMaxExpansionSize` prevents heap from expanding too much.
- Safeguard against container OOMs.
- ≅ (*`Container Limit - Container Usage`*)
  - aka free space!
- Is a hard limit.

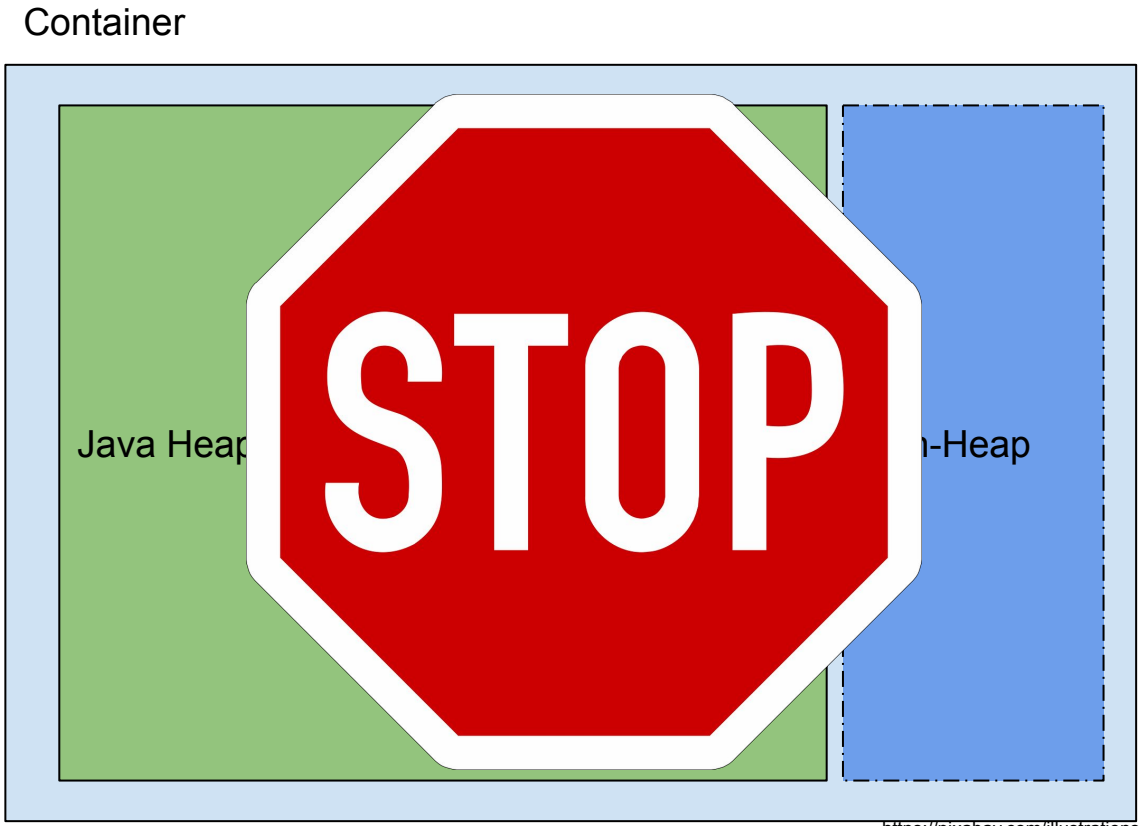
CurrentMaxExpansionSize



CurrentMaxExpansionSize



CurrentMaxExpansionSize

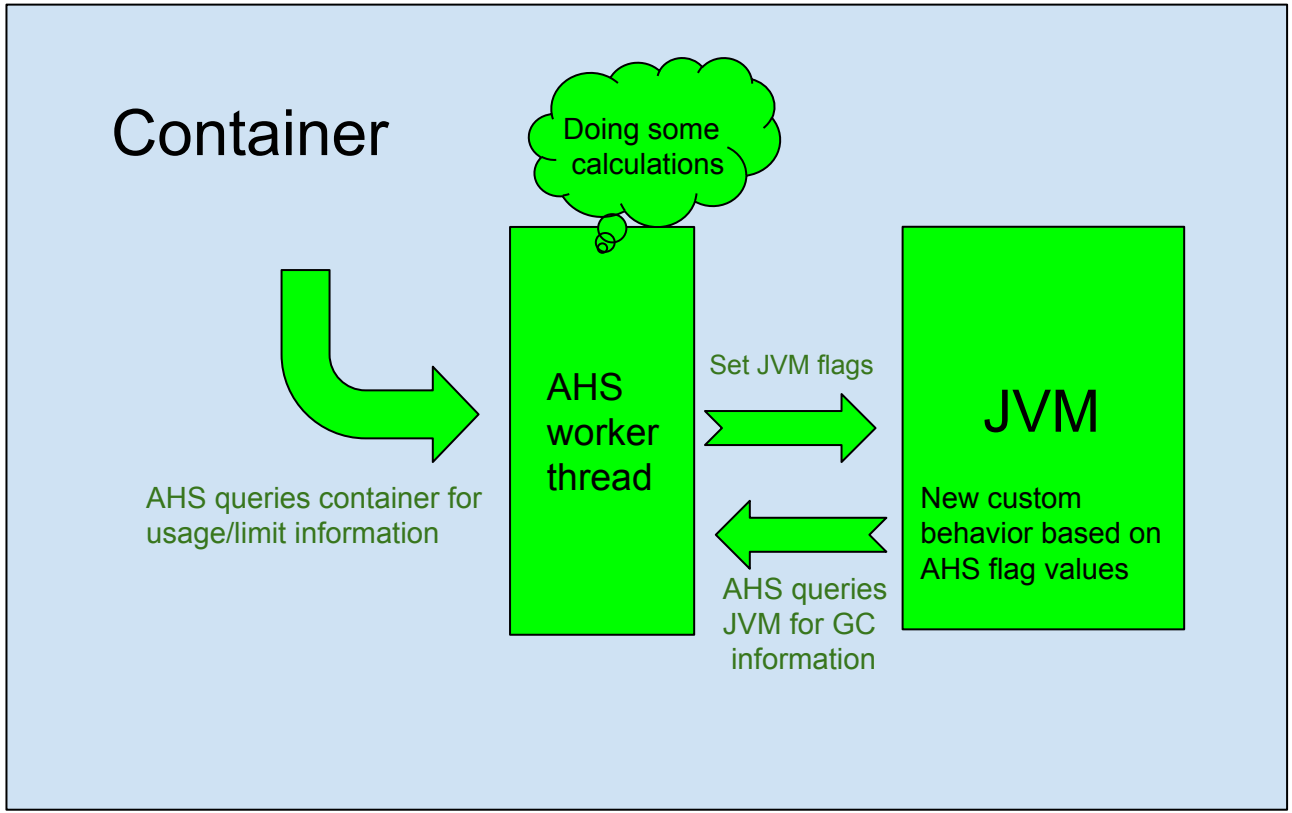


https://pixabay.com/illustrations/traffic-sign-traffic-signs-sign-6627/

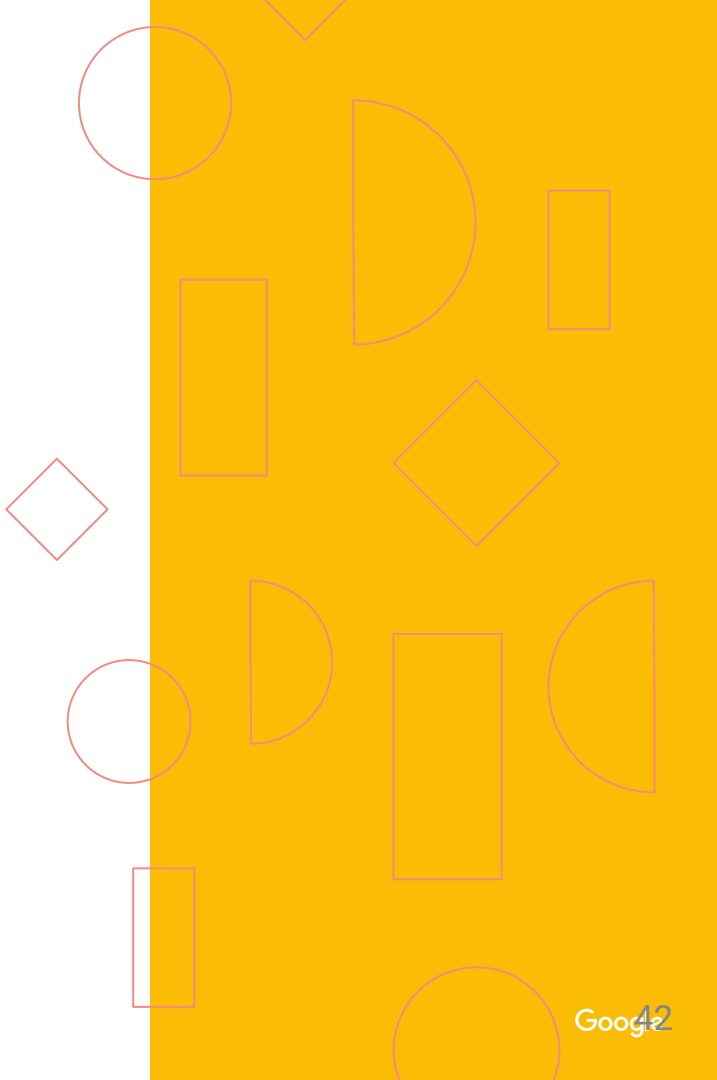
Step 3: AHS Worker Thread

- Recap:
  - AHS Worker Thread is responsible for taking container and GC information, and then setting the two manageable flags:
    - `ProposedHeapSize`
    - `CurrentMaxExpansionSize`
  - The JVM uses these flags to modify its behavior.

Implementation



Benefits of Adaptable Heap Sizing



Why AHS? - Benefits

- Simplified tuning.
- Decrease in container OOM errors.
- Memory savings for poorly-tuned jobs.
- Most servers have no perceivable difference in latency.

Why AHS? - Benefits

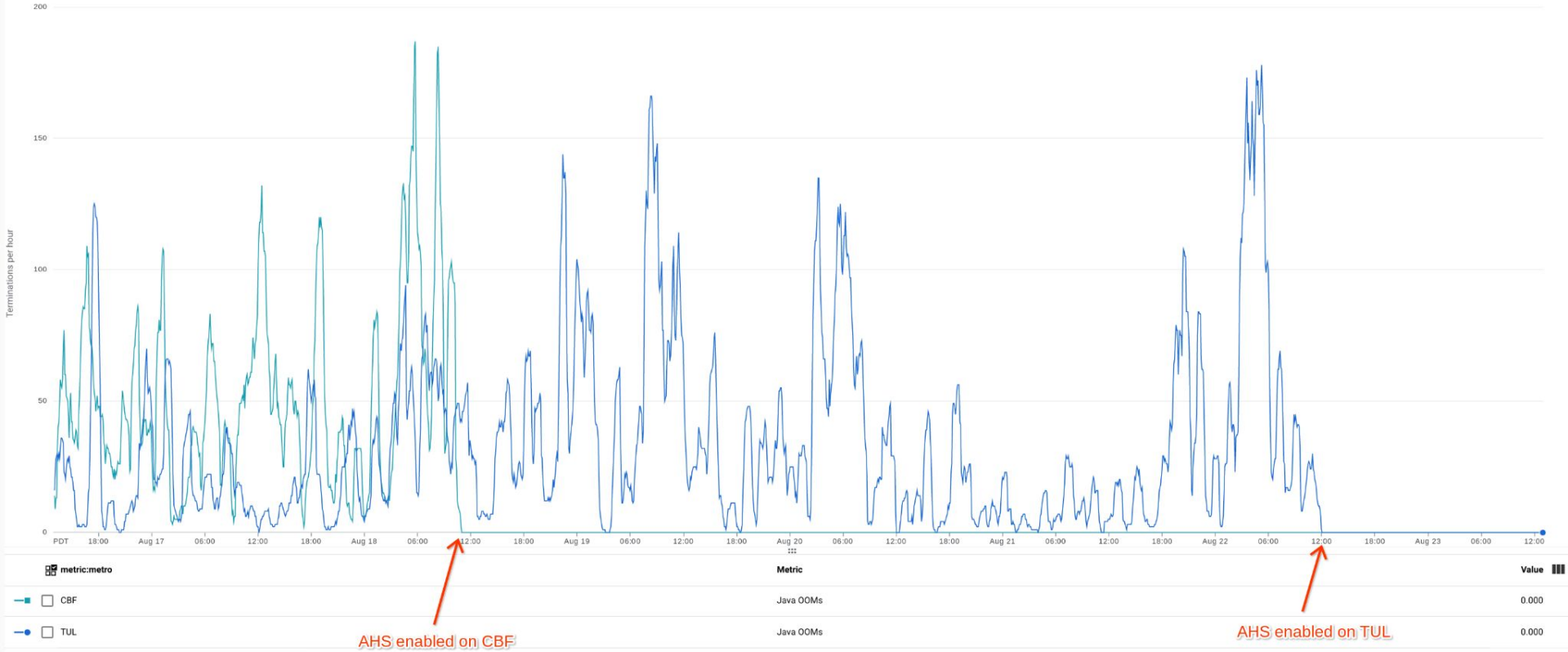
- Simplified tuning.
- Decrease in container OOM errors.
- Memory savings for poorly-tuned jobs.
- Most servers have no perceivable difference in latency.

Why AHS? - Benefits

- Simplified tuning.
- Decrease in container OOM errors.
- Memory savings for poorly-tuned jobs.
- Most servers have no perceivable difference in latency.

Case study - OOM Reductions

Reduction in Container OOM



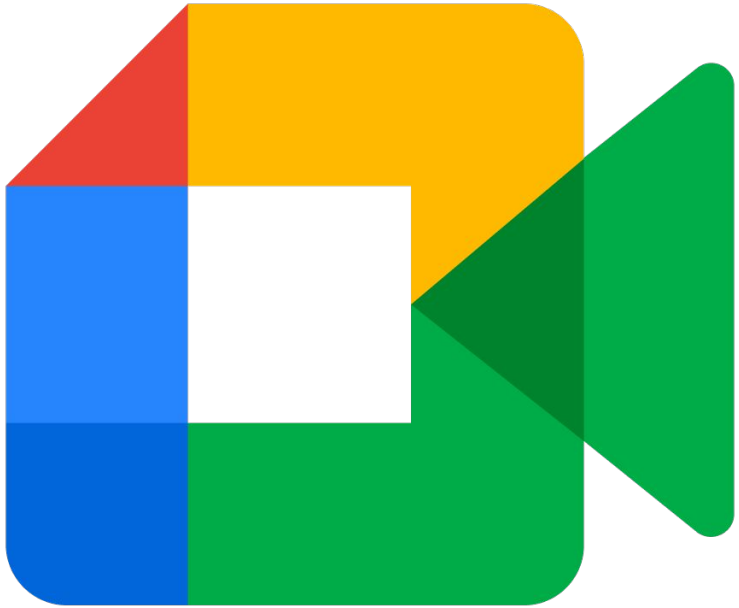
Other Metrics

- ~20k live tasks spread among ~600 live jobs at a given moment.
- ~90 TiB RAM used
- "Throughput, CPU use, etc. all appear unchanged".
- Running for months with no reported issues!

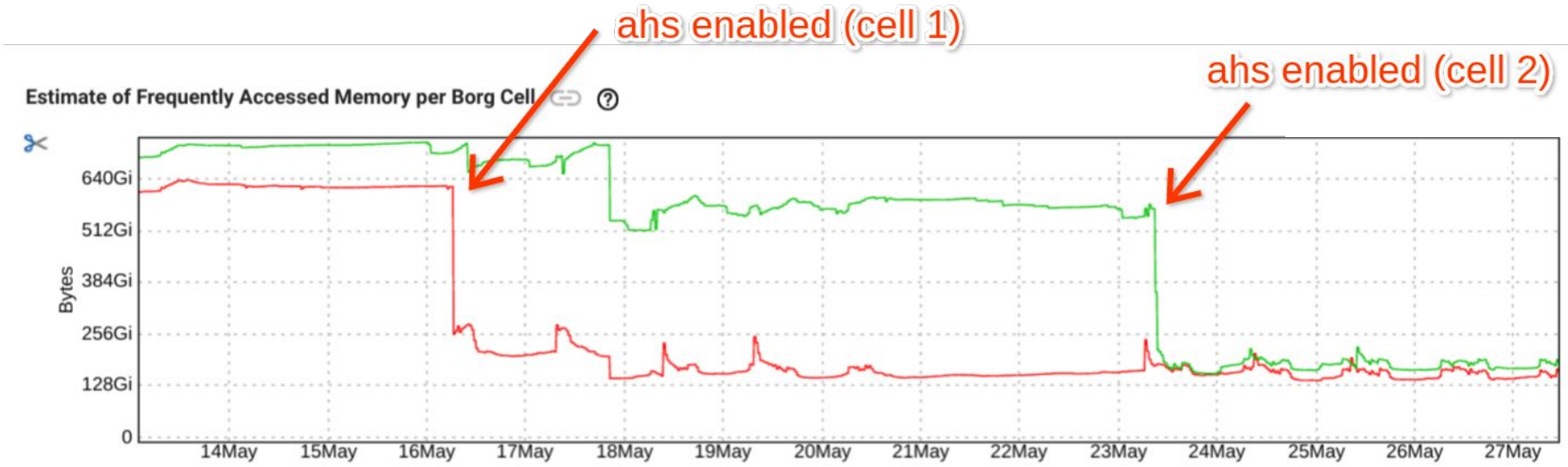
Why AHS? - Benefits

- Simplified tuning.
- Decrease in container OOM errors.
- Memory savings for poorly-tuned jobs.
- Most servers have no perceivable difference in latency.

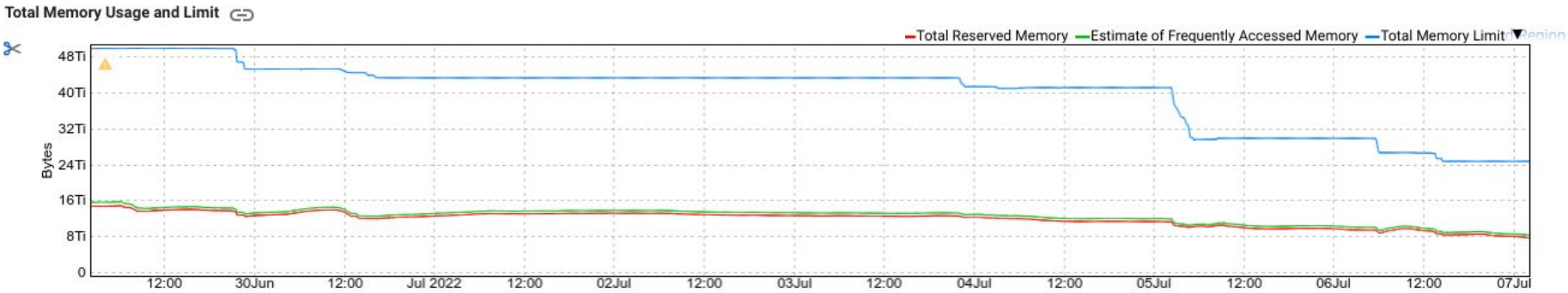
Case study - Memory savings + Latency



Reduction in Memory Usage

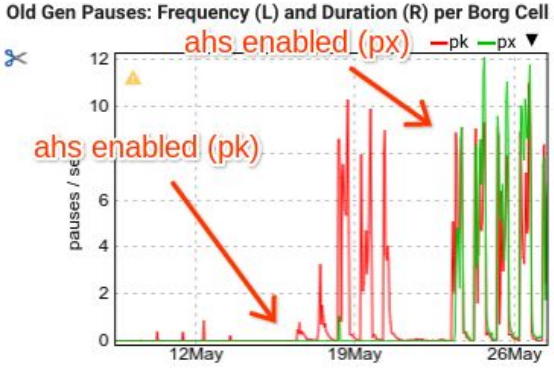


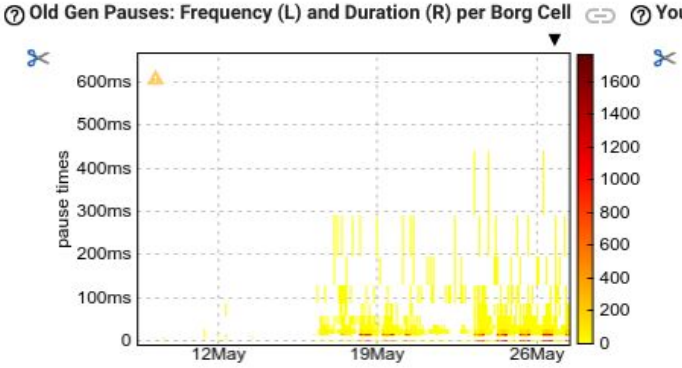
Reduction in Memory Usage

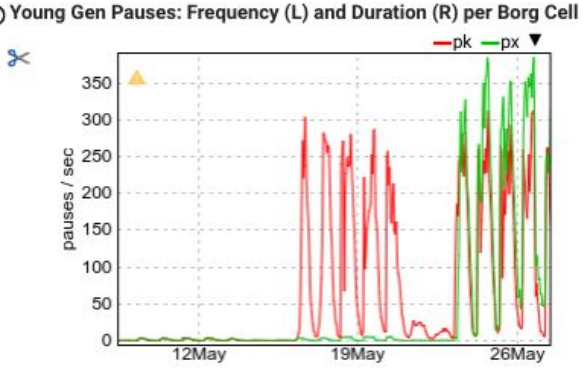


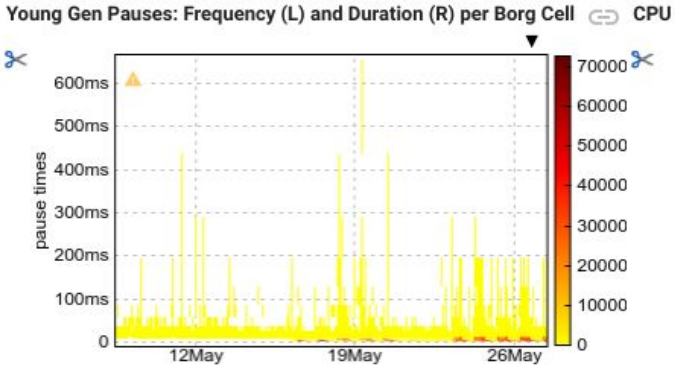
24+TB of
RAM savings!

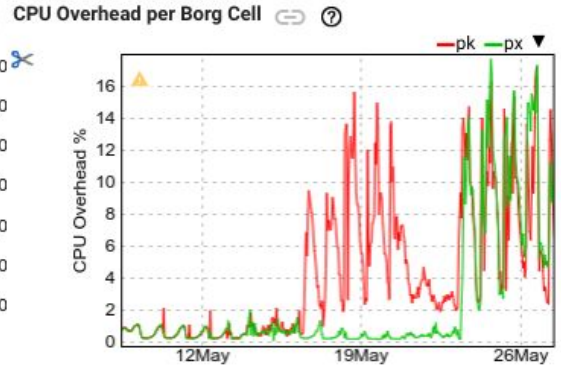
Garbage Collection Metrics

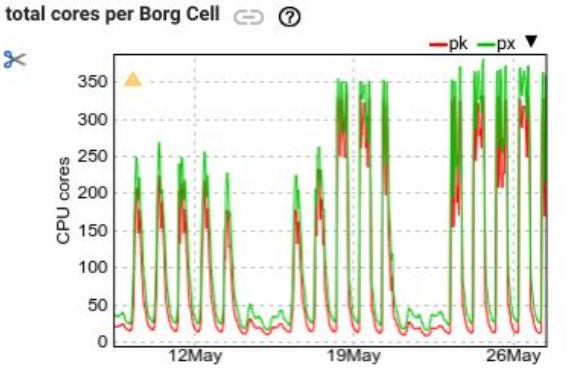




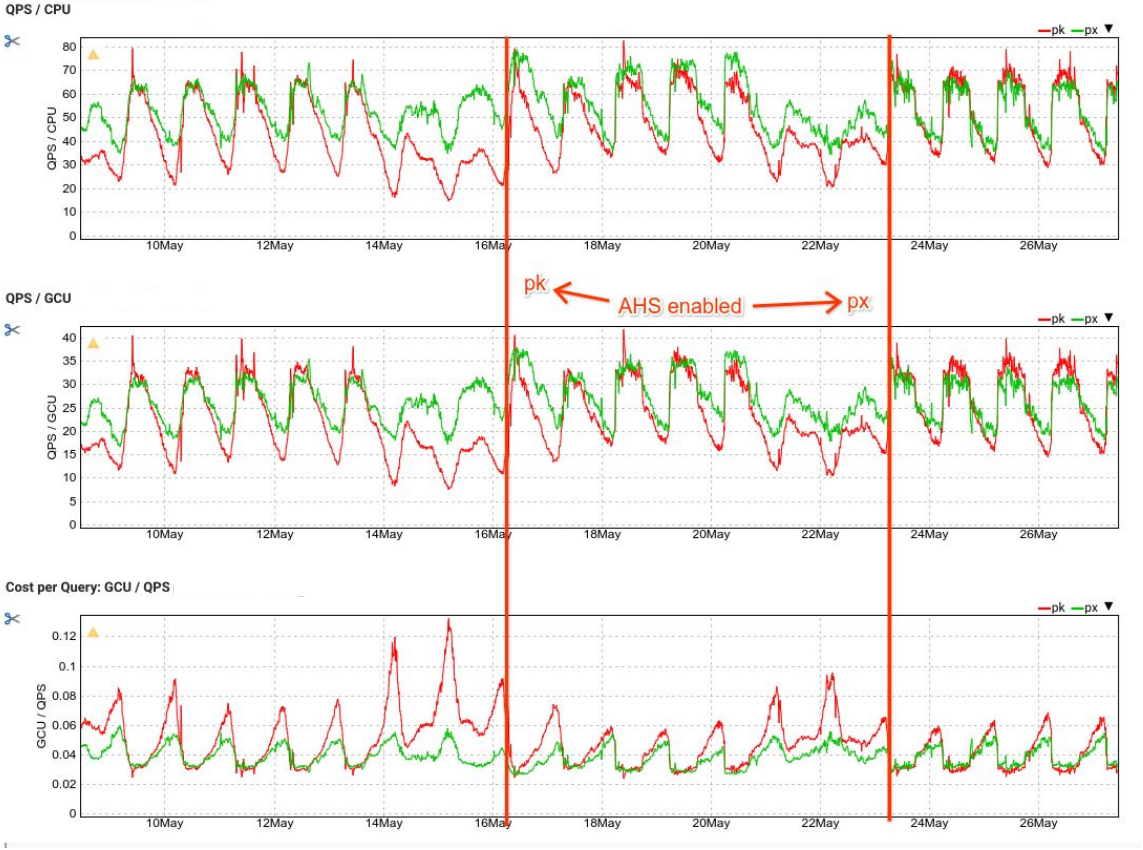




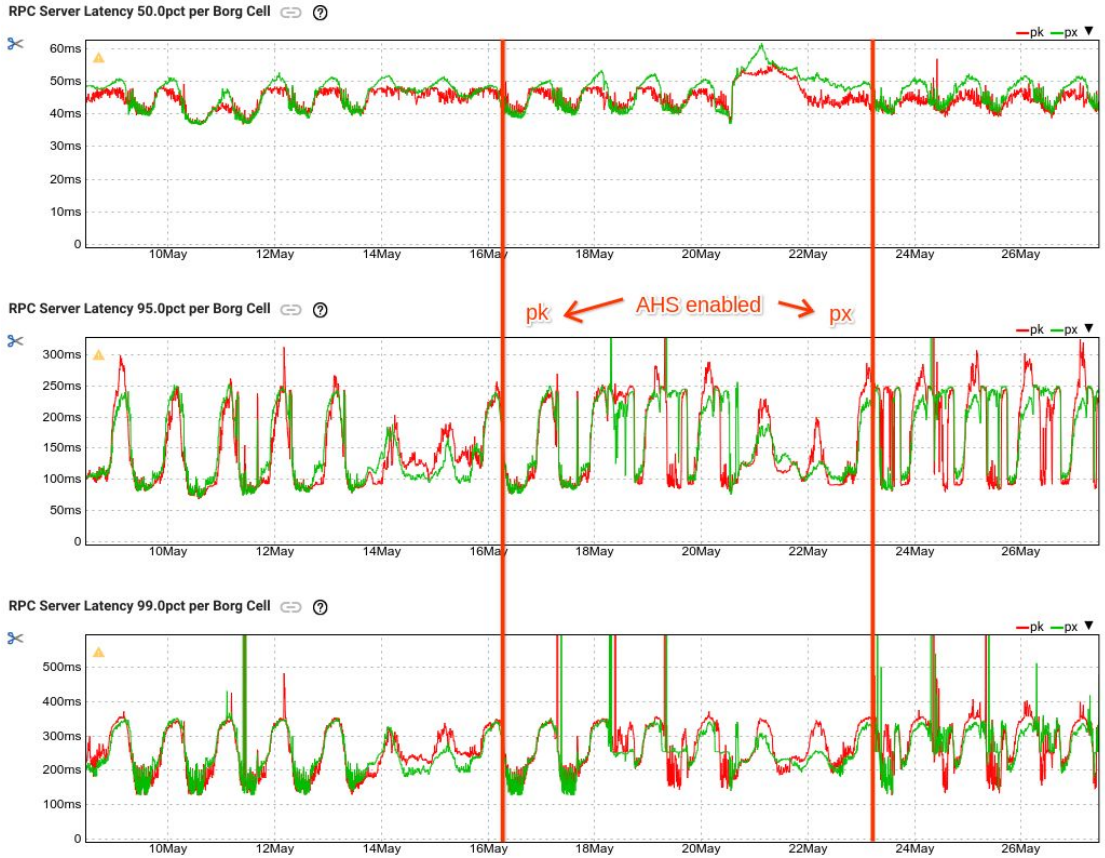




Throughput


QPS / CPU
pk
px
QPS / CPU
80
70
60
50
40
30
20
10
0
10May
12May
14May
16May
18May
20May
22May
24May
26May
pk
AHS enabled
px
QPS / GCU
pk
px
QPS / GCU
40
35
30
25
20
15
10
5
0
10May
12May
14May
16May
18May
20May
22May
24May
26May
Cost per Query: GCU / QPS
pk
px
GCU / QPS
0.12
0.1
0.08
0.06
0.04
0.02
0
10May
12May
14May
16May
18May
20May
22May
24May
26May

Latency


RPC Server Latency 50.0pct per Borg Cell
pk
px
60ms
50ms
40ms
30ms
20ms
10ms
0
10May
12May
14May
16May
18May
20May
22May
24May
26May
RPC Server Latency 95.0pct per Borg Cell
pk
AHS enabled
px
300ms
250ms
200ms
150ms
100ms
50ms
0
RPC Server Latency 99.0pct per Borg Cell
500ms
400ms
300ms
200ms
100ms

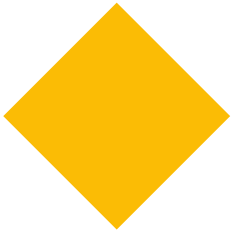
Looking forward

Upstreaming to OpenJDK

- AHS presented to the OpenJDK.
  - Similarities to existing RFEs (Request for Enhancement):
    - `ProposedHeapSize`: JDK-8236073 (Use SoftMaxHeapSize to guide GC heuristics)
    - `CurrentMaxExpansionSize`: JDK-8204088 (Dynamic Max Memory Limit)

This is where I could use your help!

- These principles could offer huge benefits industry-wide
- But this is a lot of work for one person
- 100% of my effort is internal (as of now)
- Would love to get more people involved in upstream work
  - JDK-8236073 (Use SoftMaxHeapSize to guide GC heuristics)
  - JDK-8204088 (Dynamic Max Memory Limit)



Thank you


jonathanjoo@google.com


in/jonathan-joo