Do Your GC Logs Speak To You

Visualizing G1GC, the Garbage First Garbage Collector

Kodewerk
Java™ Performance Services

About Me

Java Champions

- Consultant (www.kodewerk.com)
  - performance tuning and training
- Helped establish www.javaperformancetuning.com
- Member of Java Champion program
- Other stuff... (google is you care to)

Kodewerk
Java™ Performance Services

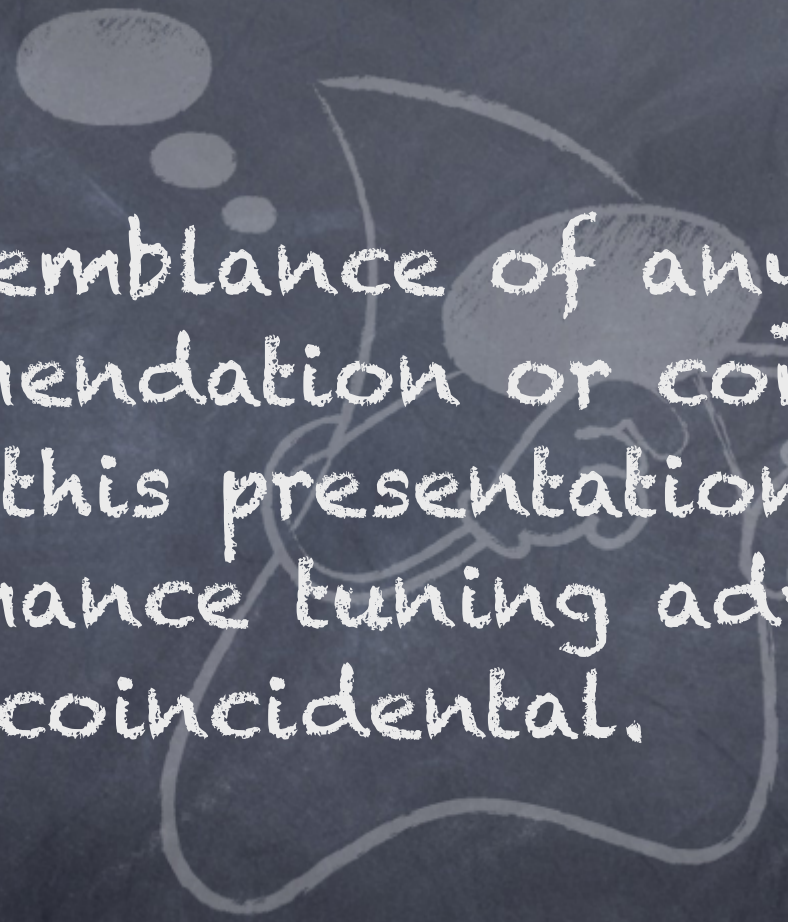
- Most recently founded a company

jClarity

- We are well on the road to delivering the next generation of advanced performance tooling

Kodewerk
Java™ Performance Services

Disclaimer



The resemblance of any opinion, recommendation or comment made during this presentation to performance tuning advice is merely coincidental.

Kodewerk

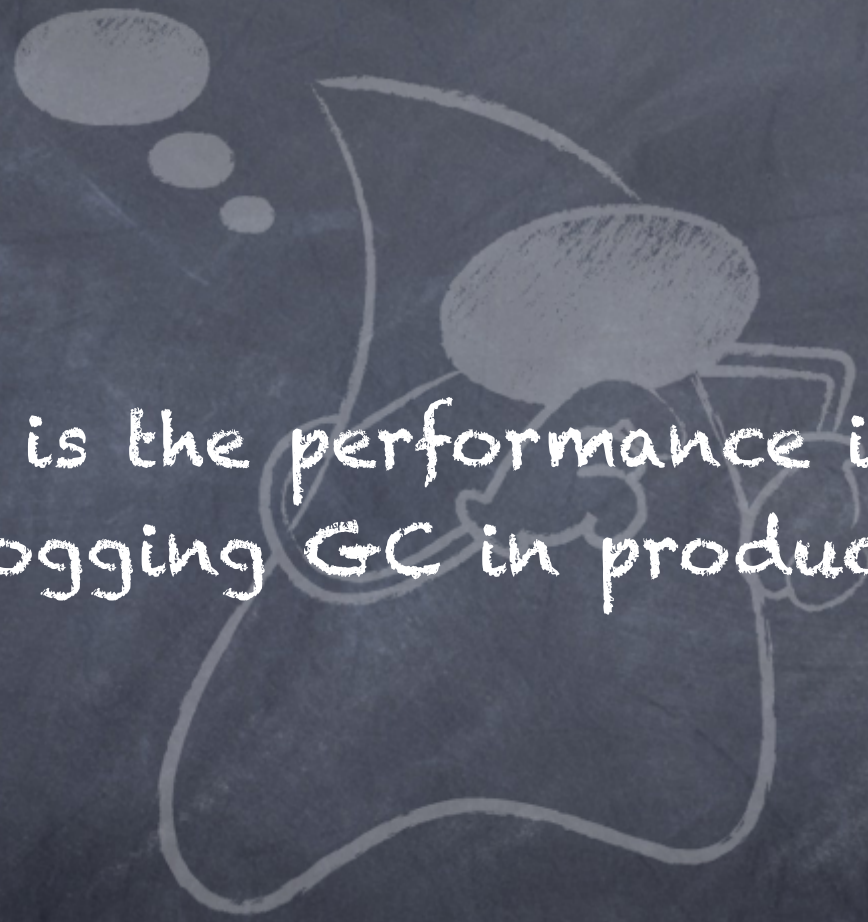
Java™ Performance Services

Why collect GC logs?

Kodewerk
Java™ Performance Services

GC logs contain the information you need to make informed choices about how to tune Java Memory managemet

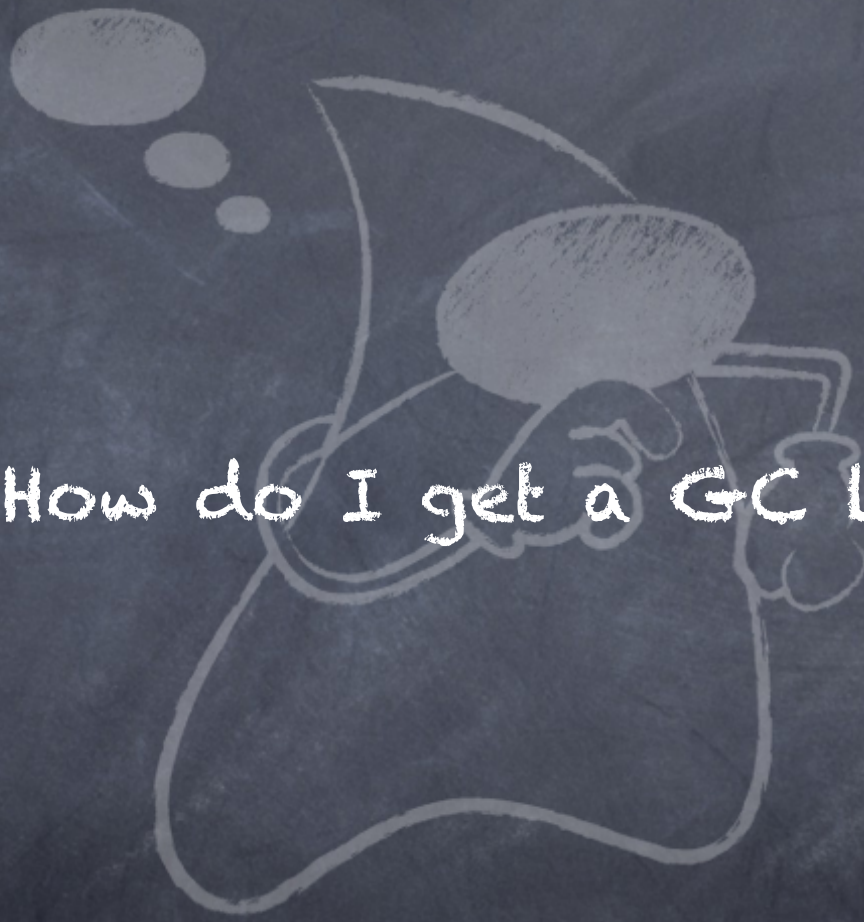
Kodewerk
Java™ Performance Services



What is the performance impact of logging GC in production?

Kodewerk

Java™ Performance Services



How do I get a GC log?

Kodewerk
Java™ Performance Services

- -verbose:gc (not recommended)
- -Xloggc:gc.log (recommended)
- -XX:+UseGCLogFileRotation (7.0 feature)
  - -XX:NumberOfGCLogFiles=10
  - -XX:GCLogFileSize=1g
- -XX:+PrintGCDetails
- -XX:+PrintTenuringDistribution
- -XX:+PrintHeapAtGC

Kodewerk
Java™ Performance Services

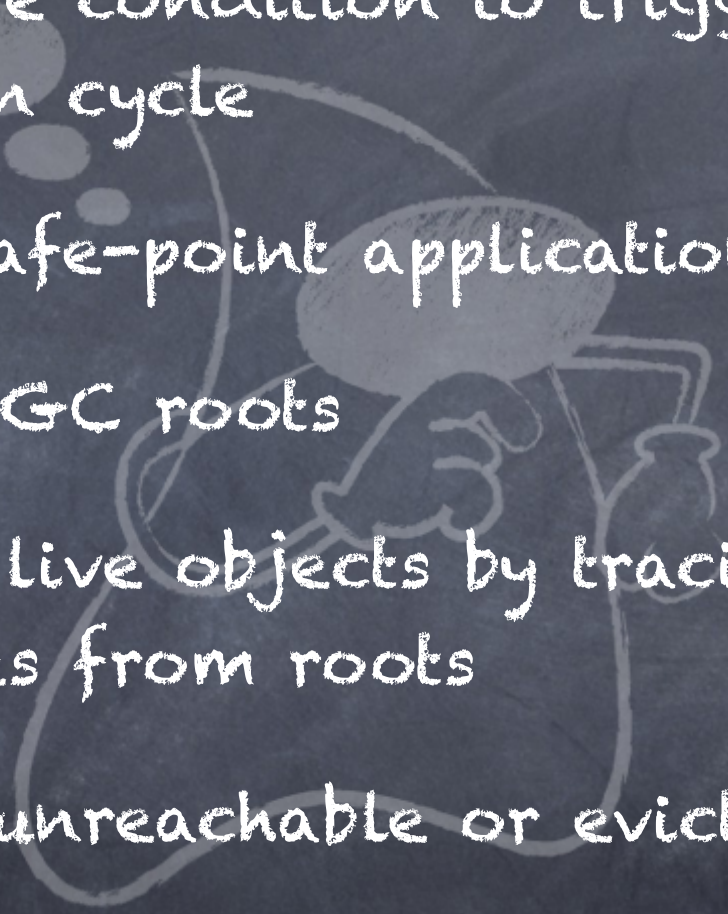
Generational GC

- Split memory into different memory pools
- Objects allocated in young and eventually moved to tenured
- Pools cleaned via mark/sweep collector

Kodewerk
Java™ Performance Services

Mark Sweep Review

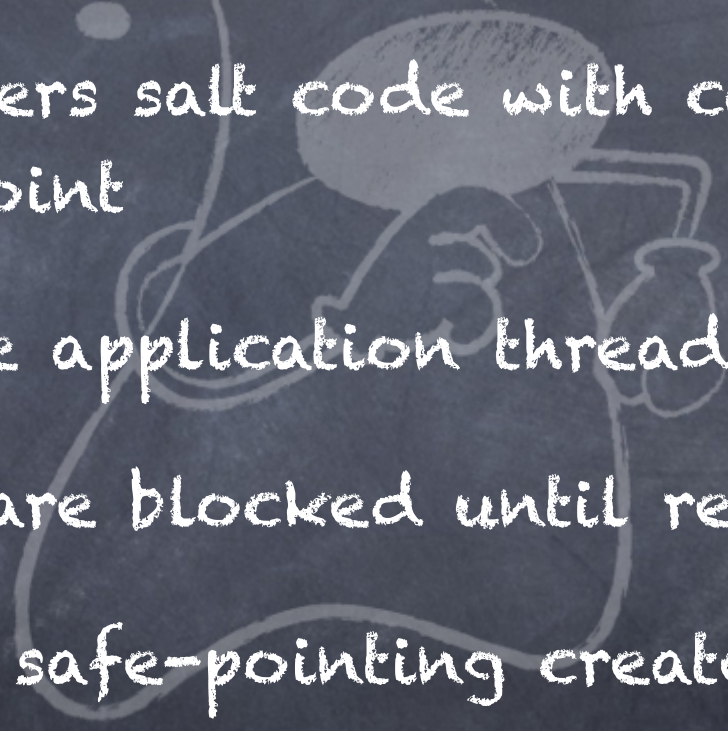
- Meet some condition to trigger a collection cycle
- Call to safe-point application threads
- Find all GC roots
- Mark all live objects by tracing references from roots
- Reclaim unreachable or evict survivors
  - fix all dangling pointers



Kodewerk
Java™ Performance Services

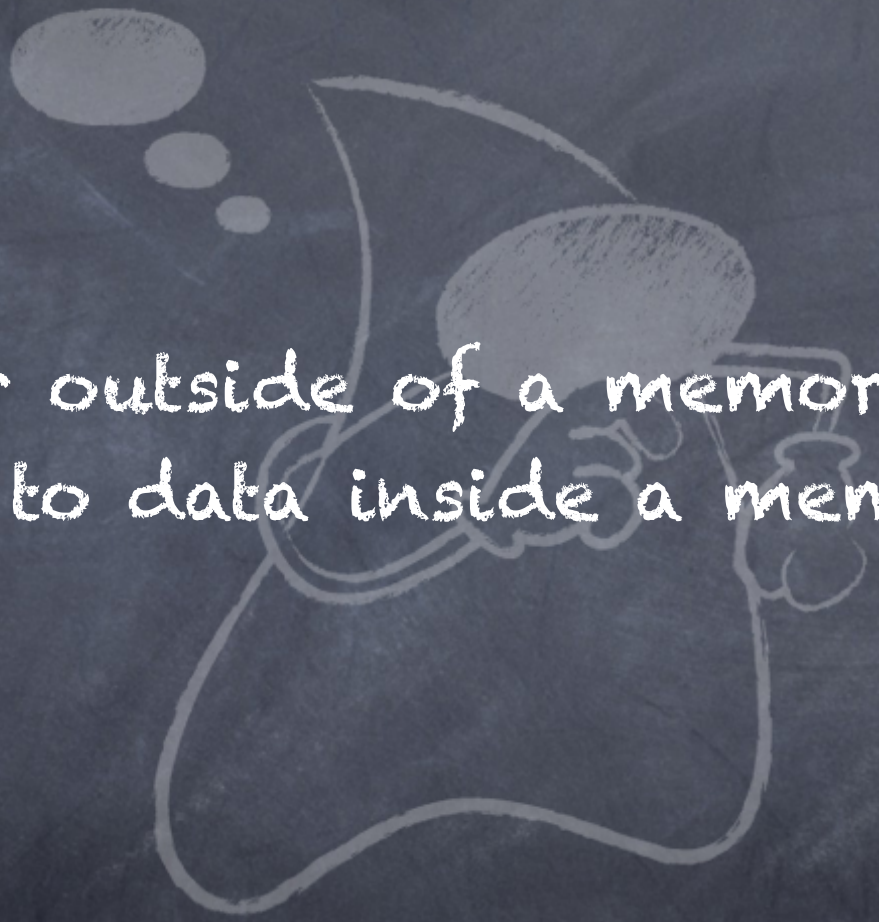
What is a Safe-Point?

- A point in a threads execution when it can safely be interrupted
  - compilers salt code with calls to safe-point
  - capture application threads
- Threads are blocked until released
- Frequent safe-pointing creates Scheduling pressure



Kodewerk
Java™ Performance Services

What is a GC Root?



a pointer outside of a memory pool pointing to data inside a memory pool

Kodewerk
Java™ Performance Services

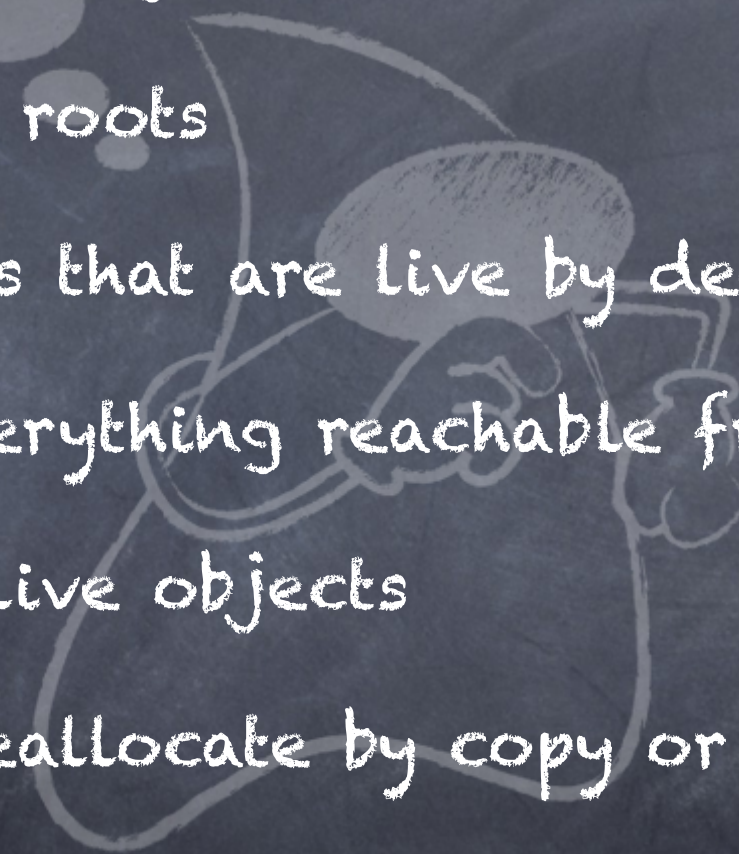
Where's Waldo?

- Aside from Perm gen, globals, registers, stack frames, locks, VM data structures roots for;
  - tenured are in young
  - young are in tenured

Kodewerk
Java™ Performance Services

What's in a Pause

- Safe-pointing (2x context switch)
- scan for roots
  - objects that are live by definition
- mark everything reachable from a root
  - trace live objects
- sweep (reallocate by copy or compaction)
  - finding and repoint dangling pointers

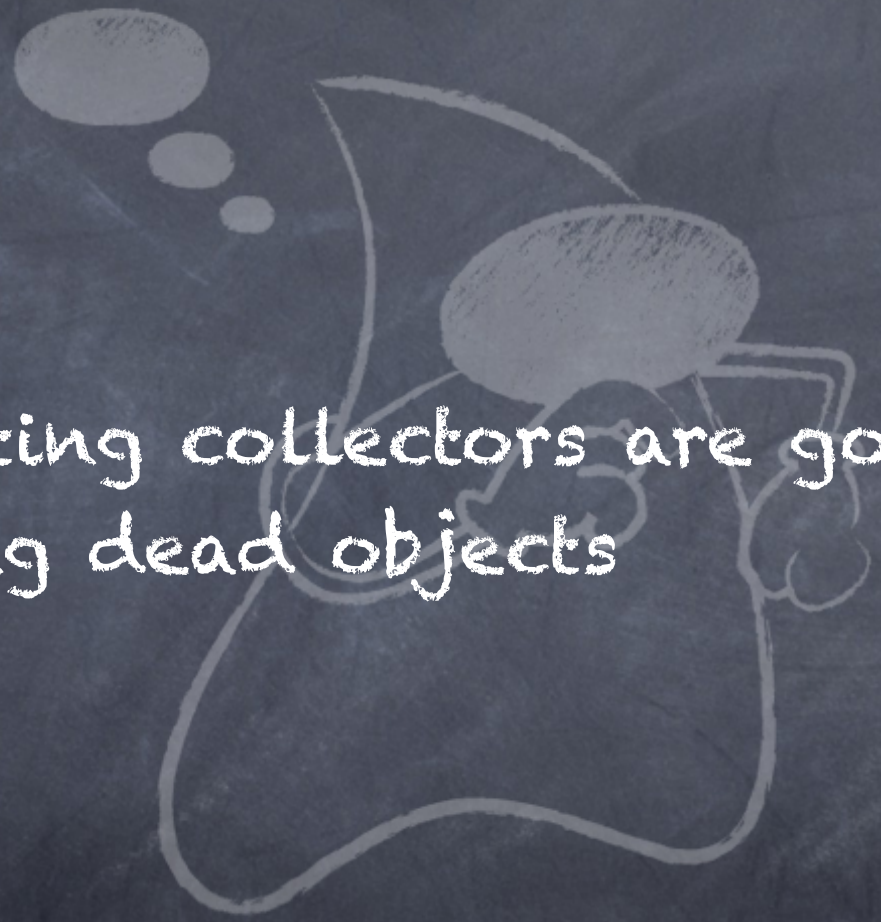


Kodewerk
Java™ Performance Services

Challenges

- Applications need to process more data than they ever had to
  - translates to larger working sets
- GC pause dominated by working set size
  - scan for roots cost dominated by heap size
- larger heaps and working set == longer pauses

Kodewerk
Java™ Performance Services

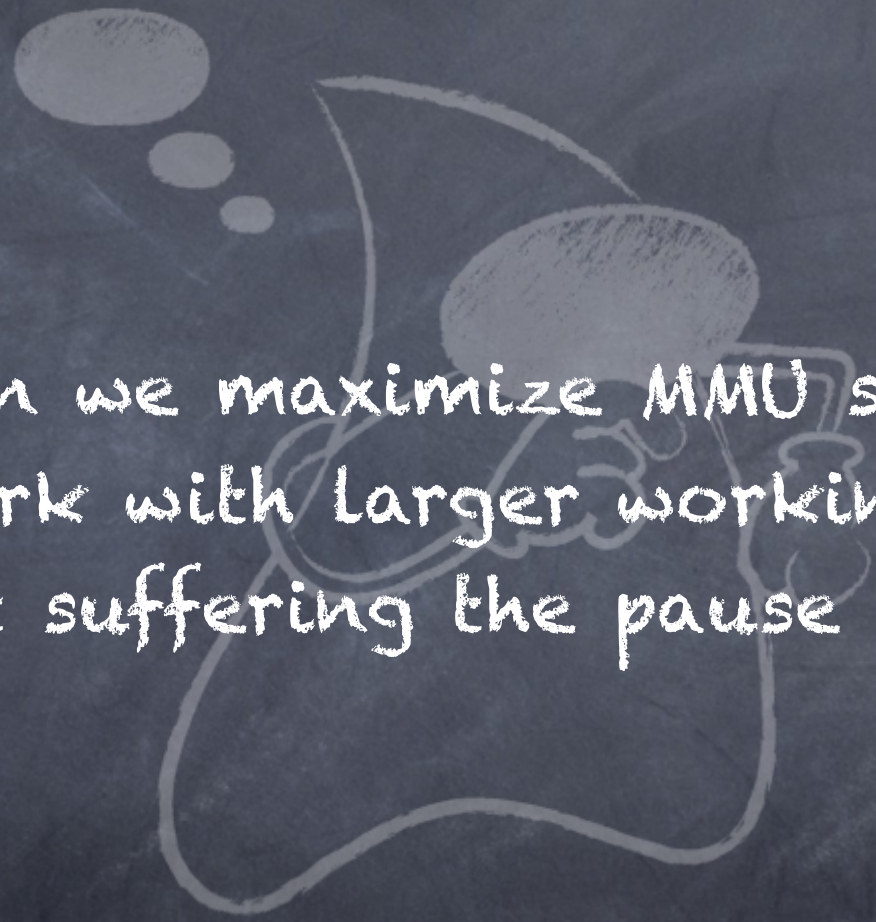


Evacuating collectors are good at ignoring dead objects

Kodewerk
Java™ Performance Services

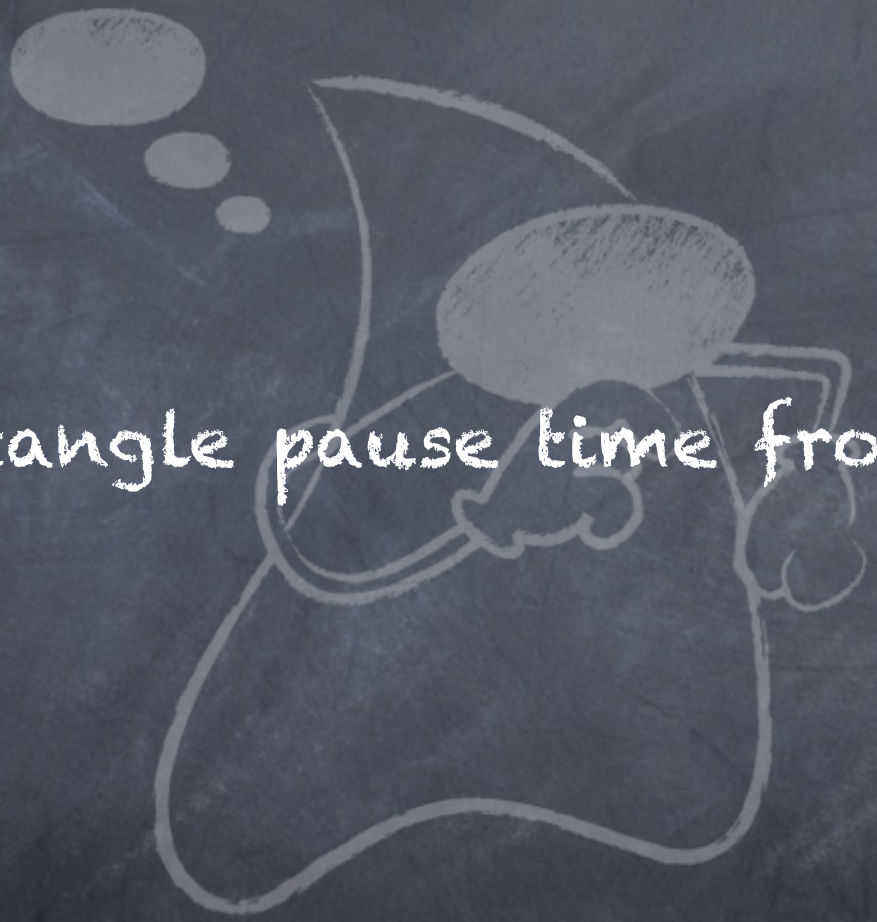
Not so good as heap size and corresponding live set size grow

Kodewerk
Java™ Performance Services



How can we maximize MMU so that we can work with larger working sets without suffering the pause

Kodewerk
Java™ Performance Services



Disentangle pause time from heap size?

Kodewerk

Java™ Performance Services

Ignore long lived objects?

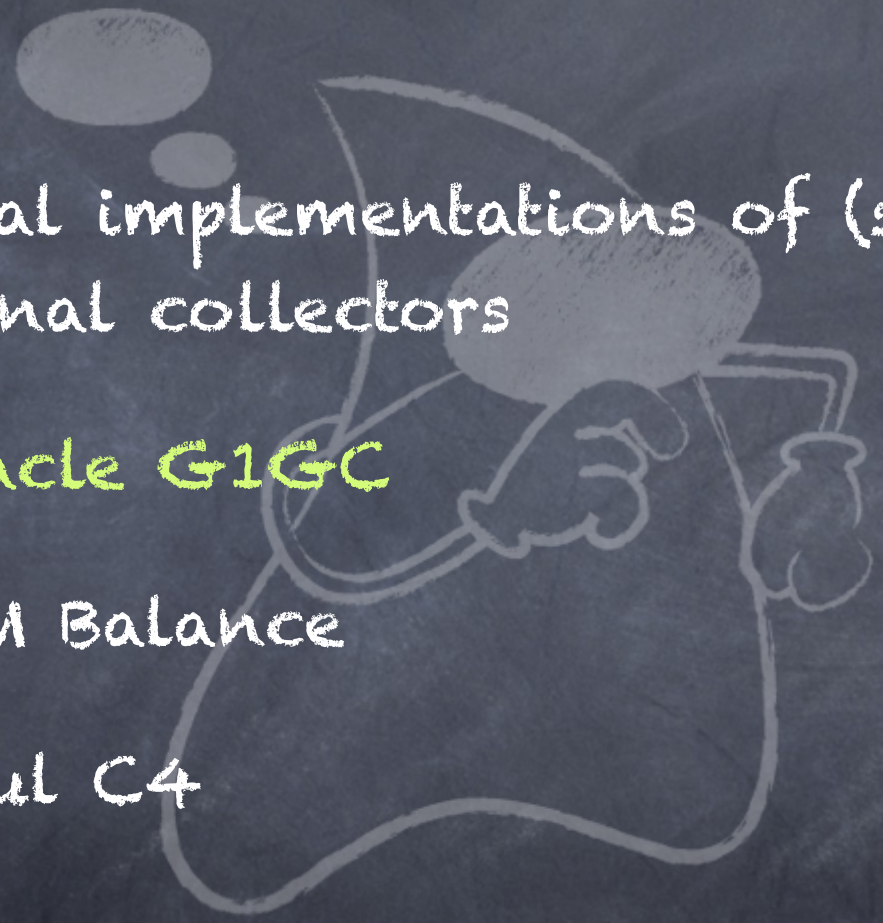
Kodewerk

Java™ Performance Services

Enter Regional Collectors

Kodewerk
Java™ Performance Services

Regional Collectors

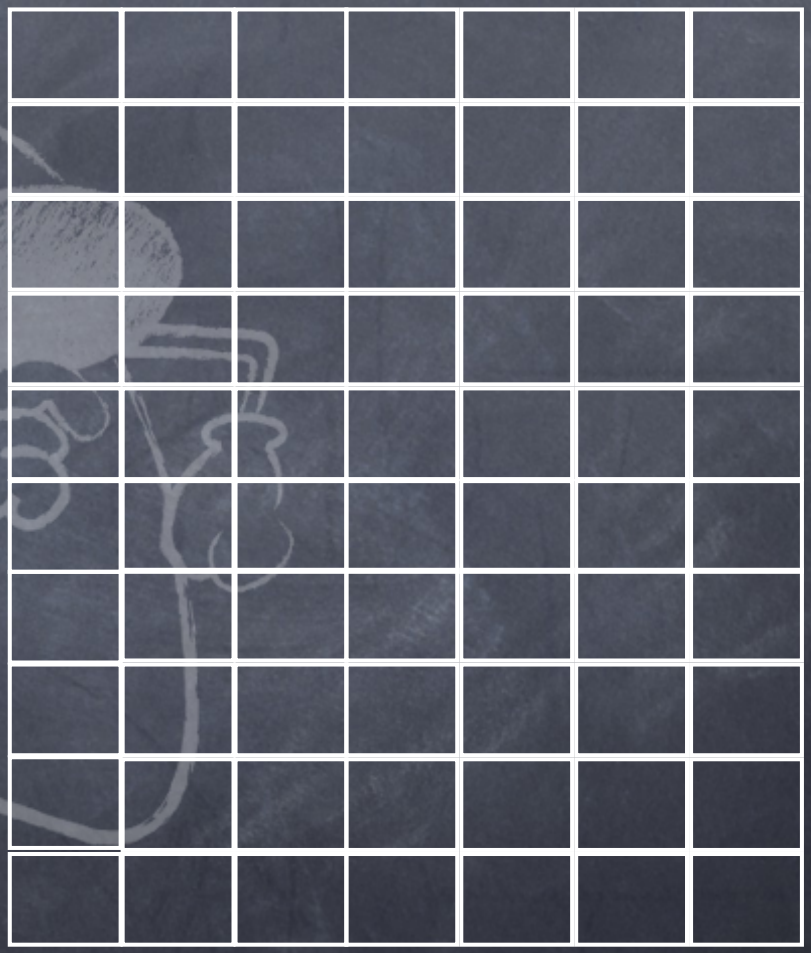


- Several implementations of (sort of) regional collectors
  - Oracle G1GC
  - IBM Balance
  - Azul C4

Kodewerk
Java™ Performance Services

G1 Heap Structure

- Heap divided into ~2000 uniformly sized regions
  - size ranges from 1m-32m
    - size determined ergonomically

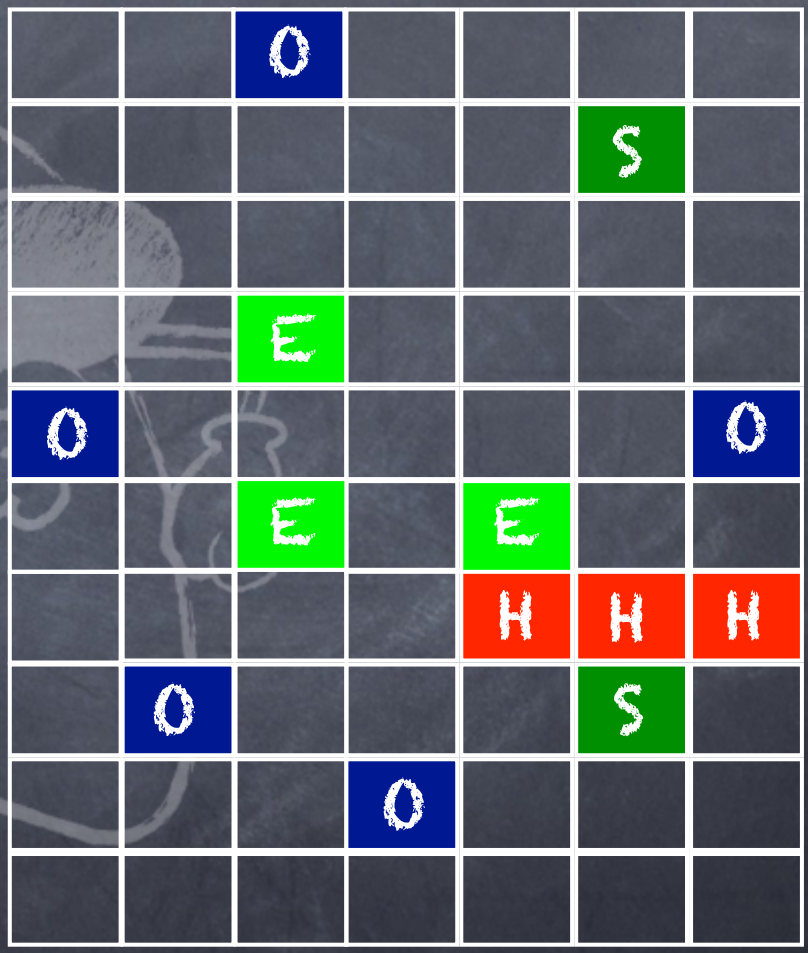


-XX:G1HeapRegionSize=<n>

Kodewerk
Java™ Performance Services

Region Sets

- At any time any region can belong to;
  - eden
  - survivor
  - old
  - humungous
    - really really big
  - un-used
- regions are taken from and returned to un-used



Kodewerk
Java™ Performance Services

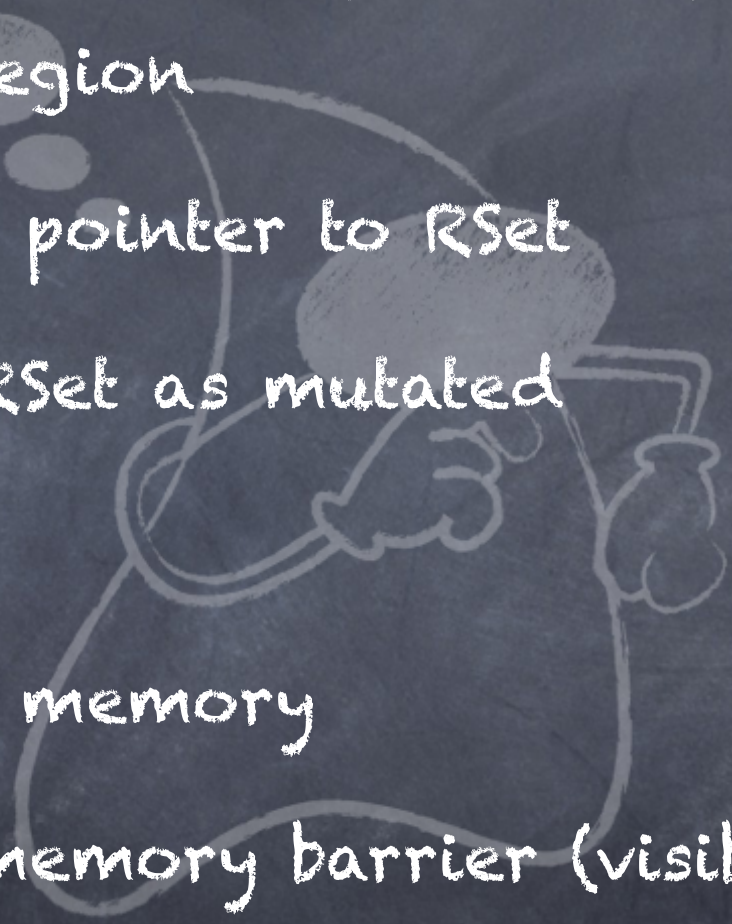
Region Sets

- Un-used is the list of free regions
- Objects are created in Eden
- Survivor and old serves the same purpose as they do in a generational heap
- Objects larger than 50% of a region are humungous
  - combine contiguous regions to create a larger space

Kodewerk
Java™ Performance Services

Remember Set (RSet)

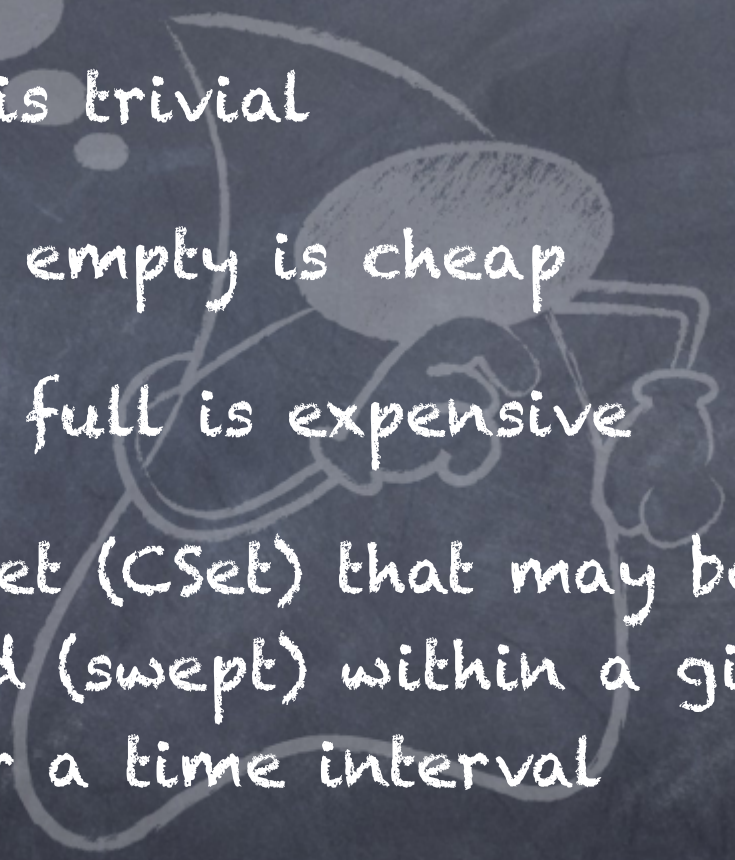
- Set of cards that track external pointers into its region
  - record pointer to RSet
  - mark RSet as mutated
- Cost
  - <5% of memory
  - write memory barrier (visibility)
  - indirection



Kodewerk
Java™ Performance Services

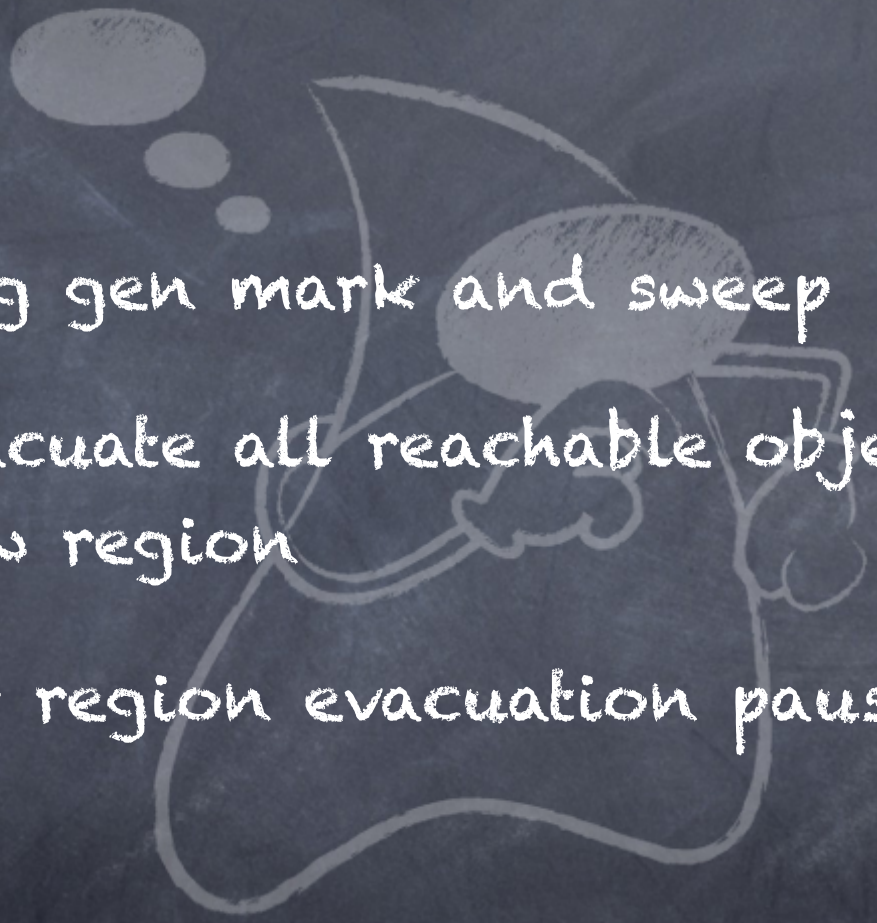
Building a CSet

- Find regions "ripe" for collection
  - empty is trivial
  - almost empty is cheap
  - almost full is expensive
- Build a set (CSet) that may be evacuated (swept) within a given pause time over a time interval
  - may not be able to comply



Kodewerk
Java™ Performance Services

G1 Phases

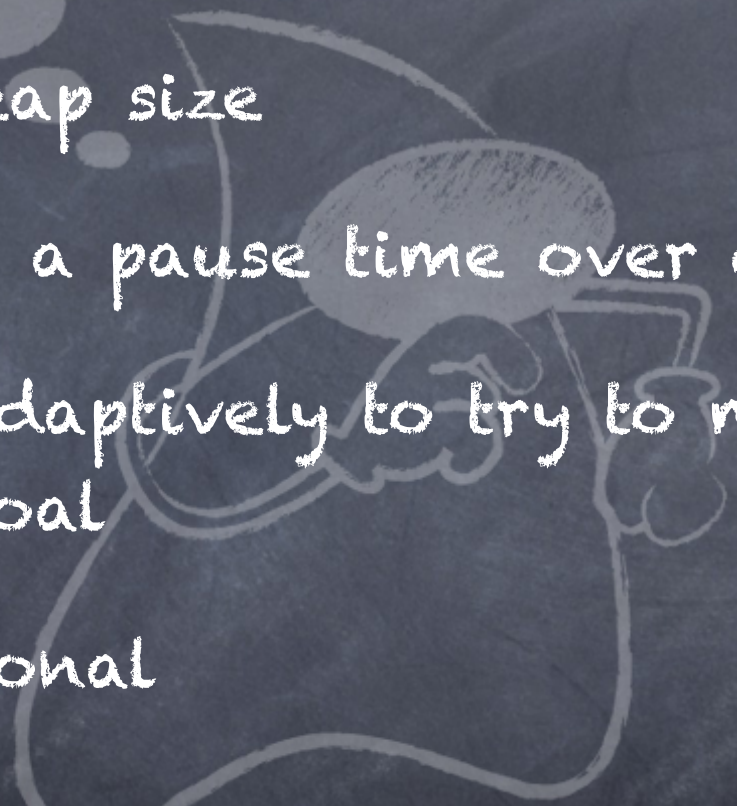


- Young gen mark and sweep
  - evacuate all reachable objects to a new region
  - per region evacuation pauses

Kodewerk
Java™ Performance Services

G1 Characteristics

- Mostly self tuning
  - max heap size
  - specify a pause time over an interval
  - sizes adaptively to try to meet pause time goal
- Generational
  - young gen
  - old gen mark



Kodewerk
Java™ Performance Services

GC Steps

- Mostly concurrent mark sweep
- young gen collector is mark and sweep
- old gen collector is mark only
  - old gen regions are swept by young gen collector
- Fully evacuating
  - no need for compaction

```
63.170: [GC pause (young)
Desired survivor size 524288 bytes, new threshold 15 (max 15)
- age   1:      82912 bytes,      82912 total
- age   2:     230888 bytes,     313800 total
, 0.00333500 secs]
   [Parallel Time:   2.8 ms]
      [GC Worker Start (ms):  63170.2  63170.2  63170.2  63170.3  63170.3  63172.7  63172.8  63172.8
       Avg: 63171.2, Min: 63170.2, Max: 63172.8, Diff:   2.5]
      [Ext Root Scanning (ms):  1.3  1.8  1.2  1.0  1.1  0.0  0.0  0.0
       Avg:   0.8, Min:   0.0, Max:   1.8, Diff:   1.8]
      [Update RS (ms):  0.0  0.0  0.1  0.2  0.1  0.0  0.0  0.0
       Avg:   0.0, Min:   0.0, Max:   0.2, Diff:   0.2]
         [Processed Buffers : 0 0 4 3 5 0 0 0
          Sum: 12, Avg: 1, Min: 0, Max: 5, Diff: 5]
      [Scan RS (ms):  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
       Avg:   0.0, Min:   0.0, Max:   0.0, Diff:   0.0]
      [Object Copy (ms):  1.0  0.6  1.1  1.1  1.1  0.0  0.0  0.0
       Avg:   0.6, Min:   0.0, Max:   1.1, Diff:   1.1]
      [Termination (ms):  0.2  0.2  0.3  0.2  0.2  0.1  0.0  0.0
       Avg:   0.2, Min:   0.0, Max:   0.3, Diff:   0.3]
         [Termination Attempts : 4 1 6 2 4 1 1 1
          Sum: 20, Avg: 2, Min: 1, Max: 6, Diff: 5]
      [GC Worker End (ms):  63172.8  63172.8  63172.8  63172.8  63172.8  63172.8  63172.8  63172.8
       Avg: 63172.8, Min: 63172.8, Max: 63172.8, Diff:   0.0]
      [GC Worker (ms):  2.6  2.6  2.6  2.5  2.5  0.1  0.0  0.0
       Avg:   1.6, Min:   0.0, Max:   2.6, Diff:   2.6]
      [GC Worker Other (ms):  0.2  0.2  0.2  0.2  0.3  2.6  2.7  2.8
       Avg:   1.2, Min:   0.2, Max:   2.8, Diff:   2.6]
   [Clear CT:   0.1 ms]
   [Other:   0.4 ms]
      [Choose CSet:   0.0 ms]
      [Ref Proc:   0.4 ms]
      [Ref Enq:   0.0 ms]
      [Free CSet:   0.0 ms]
   [Eden: 3072K(3072K)->0B(2048K) Survivors: 1024K->1024K Heap: 6999K(10M)->5288K(10M)]
 [Times: user=0.02 sys=0.00, real=0.00 secs]
```

Kodewerk
Java™ Performance Services

```
63.170: [GC pause (young)
Desired survivor size 524288 bytes, new threshold 15 (max 15)
- age   1:      82912 bytes,      82912 total
- age   2:     230888 bytes,     313800 total
, 0.00333500 secs]
```

- Pure evacuation pause of young gen regions lasting 0.00333500 seconds started 63.170 seconds after VM startup

Kodewerk
Java™ Performance Services

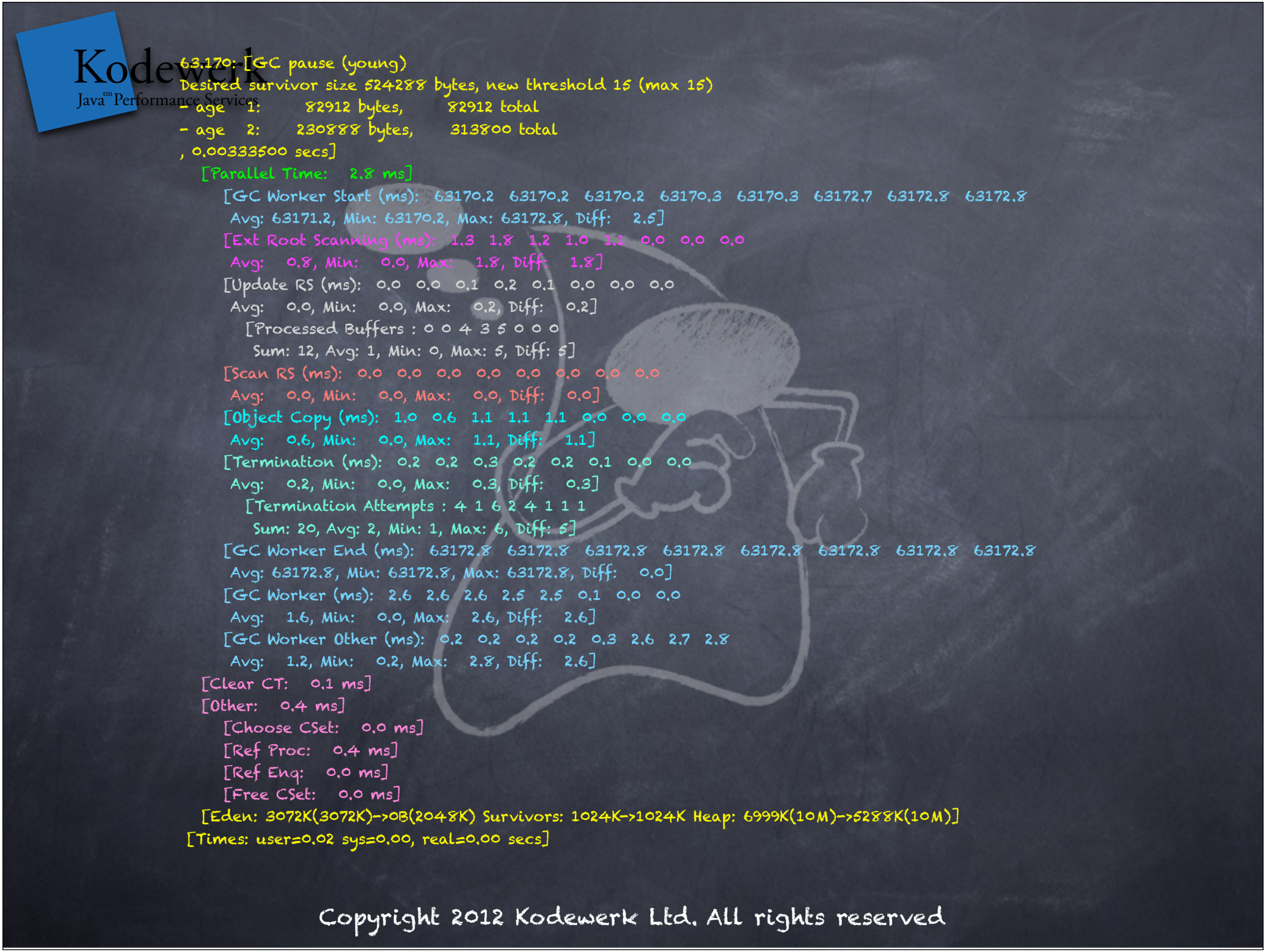
```
63.170: [GC pause (young)
Desired survivor size 524288 bytes, new threshold 15 (max 15)
- age   1:      82912 bytes,      82912 total
- age   2:     230888 bytes,     313800 total
, 0.00333500 secs]
```

- Desired survivor size is 524288 bytes
- Max tenuring threshold is 15
- Calculated threshold is 15
  - reflects bytes @ ag1 + age 2 < desired

Kodewerk
Java™ Performance Services

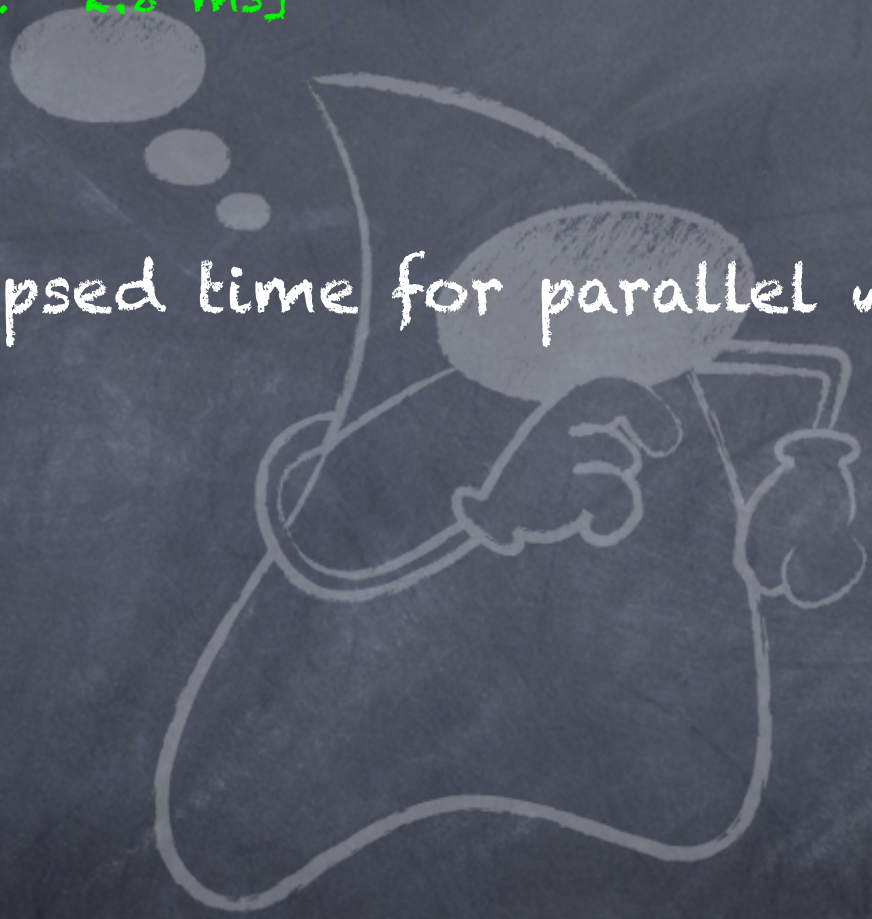
```
[Eden: 3072K(3072K)->0B(2048K) Survivors: 1024K->1024K
Heap: 6999K(10M)->5288K(10M)]
[Times: user=0.01 sys=0.00, real=0.00 secs]
```

- Evacuation changes memory consumption
  - reported on
- Report format for Eden, survivor, and total heap
  - before(size)->after(size)
  - old must be calculated



Kodewerk
Java™ Performance Services

[Parallel Time: 2.8 ms]



- Total elapsed time for parallel worker threads

```
[GC Worker Start (ms):  63170.2  63170.2  63170.2  63170.3  63170.3
63172.7  63172.8  63172.8
 Avg: 63171.2, Min: 63170.2, Max: 63172.8, Diff:   2.5]
.....
[GC Worker End (ms):  63172.8  63172.8  63172.8  63172.8
63172.8  63172.8  63172.8  63172.8
 Avg: 63172.8, Min: 63172.8, Max: 63172.8, Diff:   0.0]
```

- Time stamp for when each GC worker started and then ended
- Statistical summary of record

Kodewerk
Java™ Performance Services

```
[GC Worker (ms):  2.6  2.6  2.6  2.5  2.5  0.1  0.0  0.0
 Avg:   1.6, Min:   0.0, Max:   2.6, Diff:   2.6]
[GC Worker Other (ms):  0.2  0.2  0.2  0.2  0.3  2.6  2.7  2.8
 Avg:   1.2, Min:   0.2, Max:   2.8, Diff:   2.6]
```

- Total concurrent time from start and stop record
- Other is activity not accounted for in the summary records

Kodewerk
Java™ Performance Services

```
[Ext Root Scanning (ms):  1.3  1.8  1.2  1.0  1.1  0.0  0.0  0.0
 Avg:   0.8, Min:   0.0, Max:   1.8, Diff:   1.8]
```

- Per-thread time to scan for roots

Kodewerk
Java™ Performance Services

```
[Update RS (ms):  0.0  0.0  0.1  0.2  0.1  0.0  0.0  0.0
        Avg:  0.0, Min:  0.0, Max:  0.2, Diff:  0.2]
   [Processed Buffers : 0 0 4 3 5 0 0 0
    Sum: 12, Avg: 1, Min: 0, Max: 5, Diff: 5]
```

- Per-thread time to process update buffers
- Mutator threads are still working
  - updates to RSet maintained in a buffer
- Number of buffers processed by each thread
- Statistical summary for each record

Kodewerk
Java™ Performance Services

```
[Scan RS (ms):  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 Avg:   0.0, Min:   0.0, Max:   0.0, Diff:   0.0]
```

- Per-thread time to process RSets

```
[Object Copy (ms):  1.0  0.6  1.1  1.1  1.1  0.0  0.0  0.0
 Avg:   0.6, Min:   0.0, Max:   1.1, Diff:   1.1]
```

- Per-thread time spent copying objects in the CSet to other regions

Kodewerk
Java™ Performance Services

```
[Termination (ms):  0.2  0.2  0.3  0.2  0.2  0.1  0.0  0.0
 Avg:   0.2, Min:   0.0, Max:   0.3, Diff:   0.3]
   [Termination Attempts : 4 1 6 2 4 1 1 1
      Sum: 20, Avg: 2, Min: 1, Max: 6, Diff: 5]
```

- Per-thread time of offer to terminate
- Follow up is number of termination attempts
  - maybe offered work from other threads queue
    - work stealing

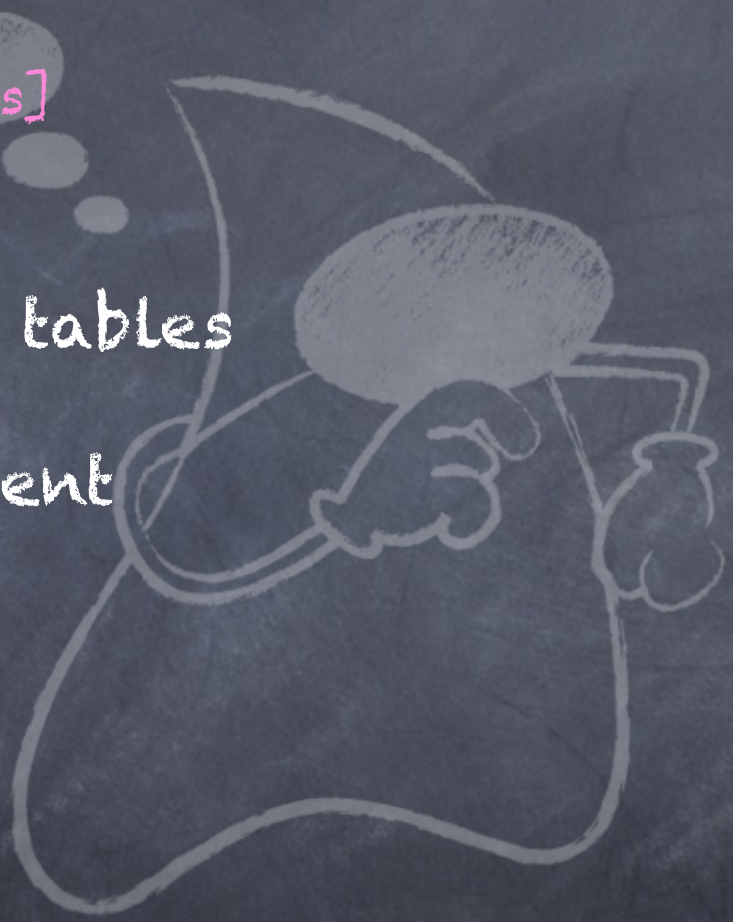
Kodewerk
Java™ Performance Services

```
[Termination (ms):  0.2  0.2  0.3  0.2  0.2  0.1  0.0  0.0
 Avg:   0.2, Min:   0.0, Max:   0.3, Diff:   0.3]
   [Termination Attempts : 4 1 6 2 4 1 1 1
      Sum: 20, Avg: 2, Min: 1, Max: 6, Diff: 5]
```

- Per-thread time of offer to terminate
- Follow up is number of termination attempts
  - maybe offered work from other threads queue
    - work stealing

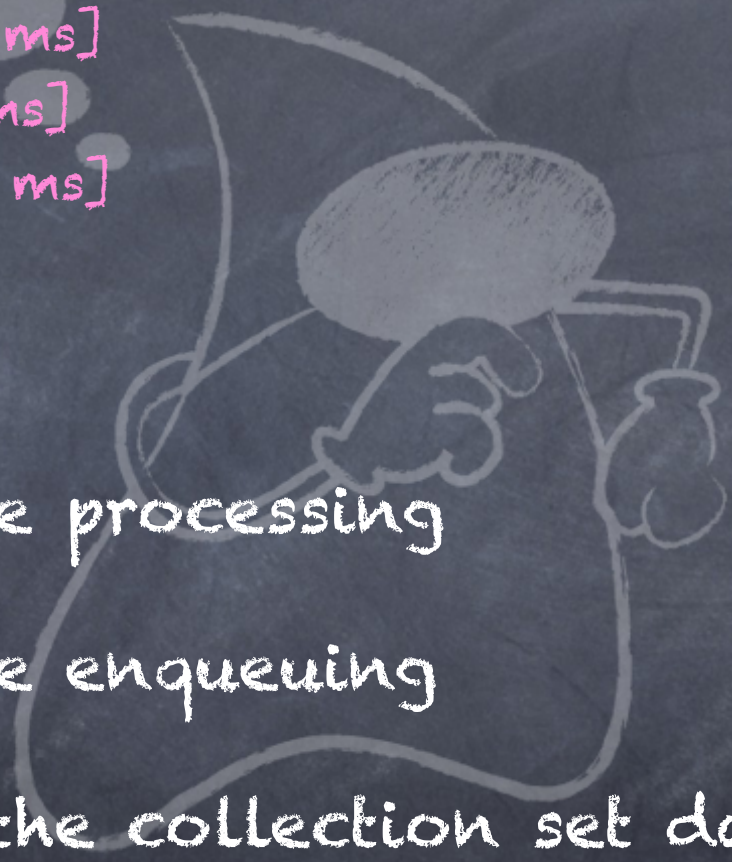
Kodewerk
Java™ Performance Services

[Clear CT: 0.1 ms]

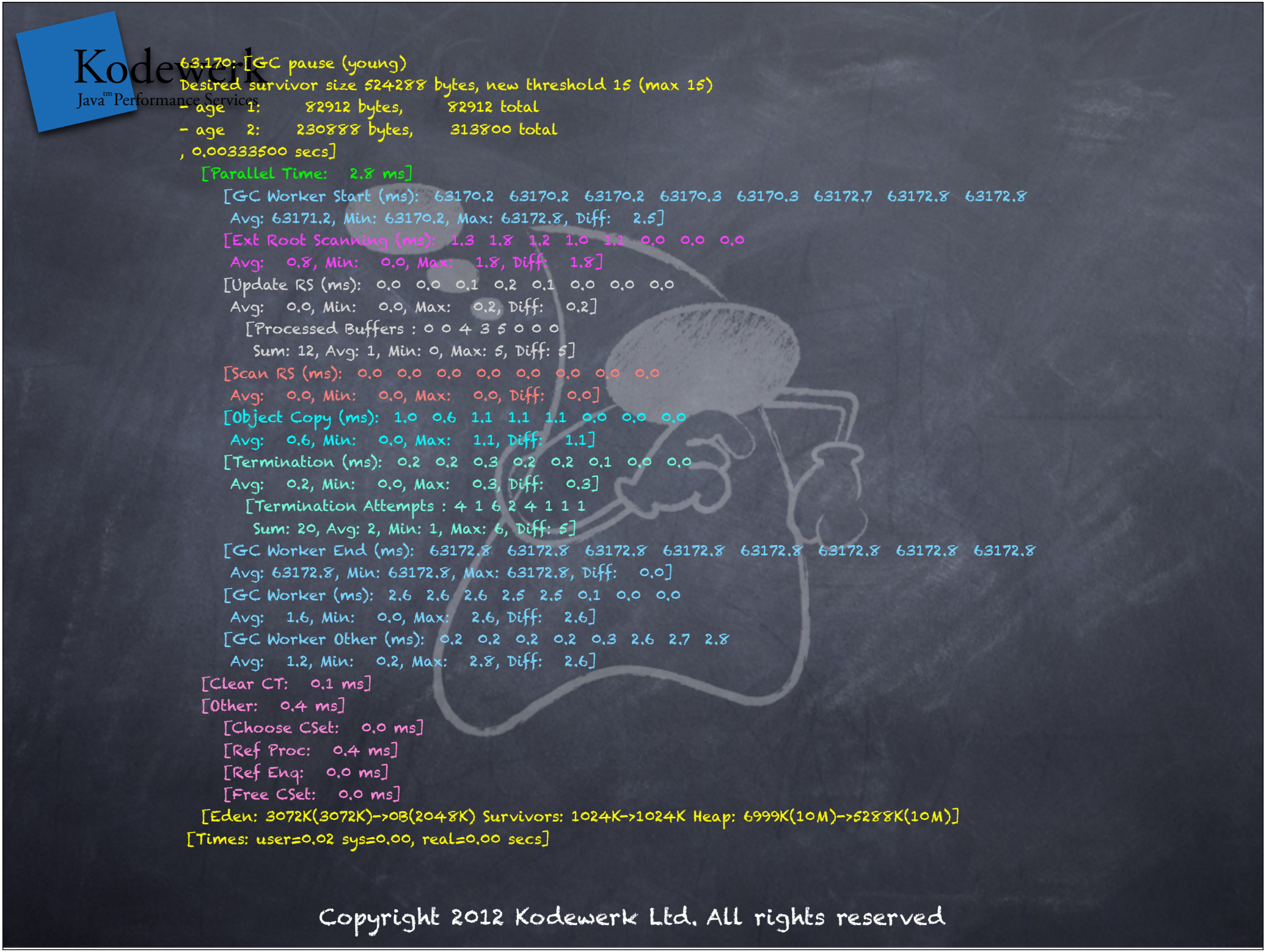


- Clear card tables
  - pause event

```
[Other:    0.4 ms]
   [Choose CSet:    0.0 ms]
   [Ref Proc:    0.4 ms]
   [Ref Enq:    0.0 ms]
   [Free CSet:    0.0 ms]
```



- Other tasks
  - reference processing
  - reference enqueuing
  - freeing the collection set data structure



Kodewerk
Java™ Performance Services

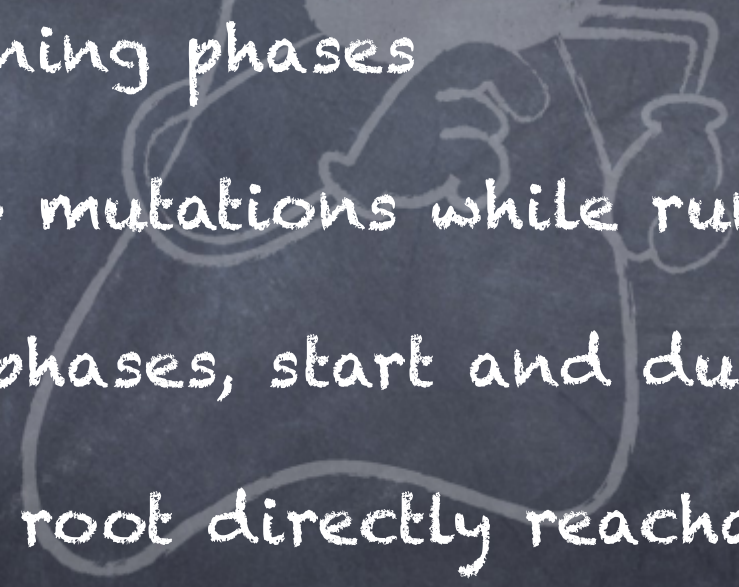
```
63.233: [GC pause (young)
Desired survivor size 524288 bytes, new threshold 1 (max 15)
- age   1:    1275728 bytes,    1275728 total
- age   2:      81624 bytes,    1357352 total
- age   3:     230888 bytes,    1588240 total
 (initial-mark), 0.00522500 secs]
```

- Old space initial-mark has been piggybacked onto the evacuation phase

Kodewerk
Java™ Performance Services

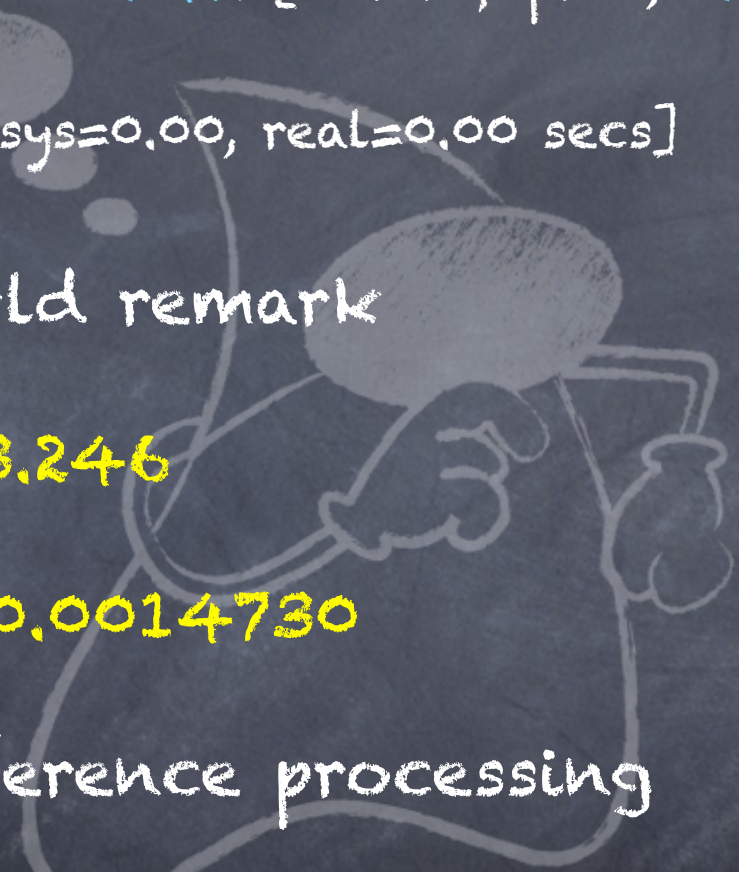
```
63.239: [GC concurrent-root-region-scan-start]
63.239: [GC concurrent-root-region-scan-end, 0.0006690]
63.239: [GC concurrent-mark-start]
63.246: [GC concurrent-mark-end, 0.0066900 sec]
```

- Several scanning phases
  - account fo mutations while running
- Concurrent phases, start and duration
  - scan from root directly reachable from survivor spaces



Kodewerk
Java™ Performance Services

63.246: [GC remark 63.247: [GC ref-proc, 0.0000480 secs], 0.0014730 secs]
[Times: user=0.01 sys=0.00, real=0.00 secs]



- Stop-the-world remark
  - starts @ 63.246
  - duration: 0.0014730
- Includes reference processing
  - starts @ 63.247
  - duration: 0.0000480

Kodewerk
Java™ Performance Services

80.197: [GC concurrent-cleanup-start]
80.197: [GC concurrent-cleanup-end, 0.0000740]

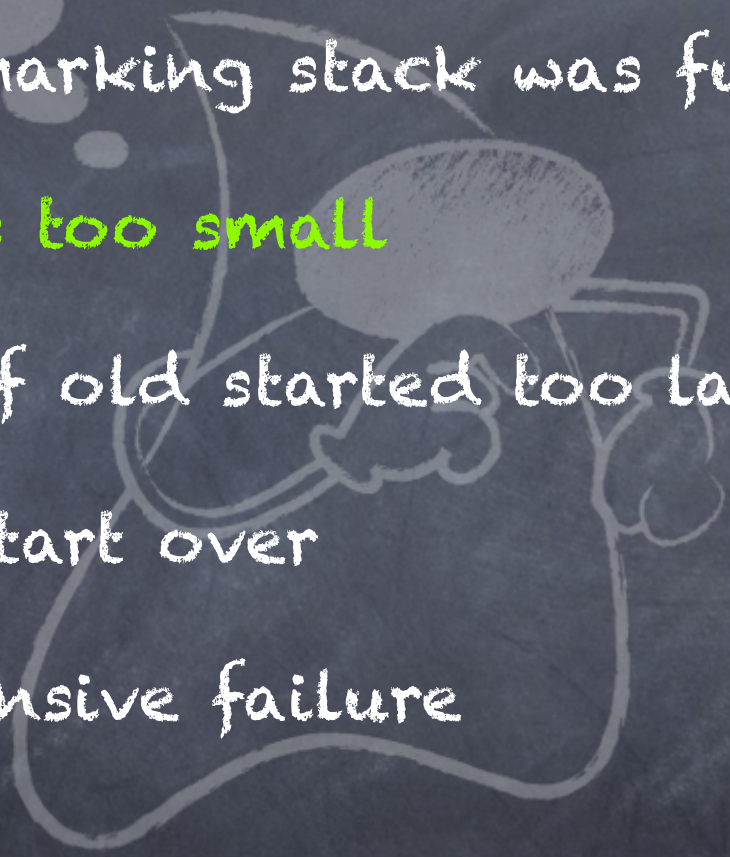
- Return empty regions back to unused
- Concurrent time of 0.0000749 seconds

Kodewerk
Java™ Performance Services

Other Records

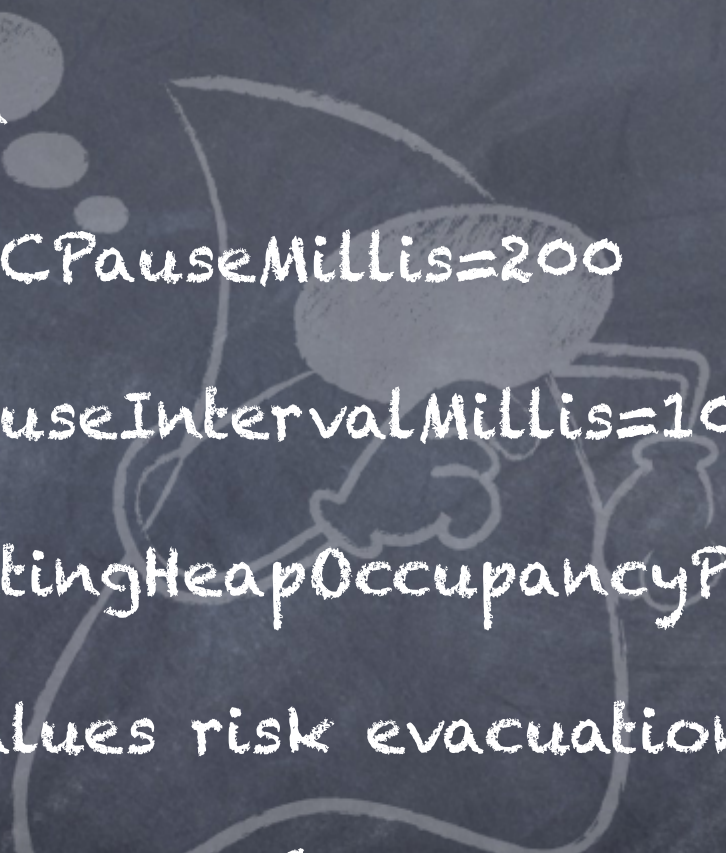
[GC concurrent-mark-reset-for-overflow]

- Global marking stack was full
  - heap is too small
  - scan of old started too late
  - must start over
    - expensive failure



Kodewerk
Java™ Performance Services

Flags and Stuff



- -mx, -mn
- -XX:MaxGCPauseMillis=200
- -XX:GCPauseIntervalMillis=1000
- -XX:InitiatingHeapOccupancyPercent=45
  - high values risk evacuation failures
- numerous other flags that get hairy

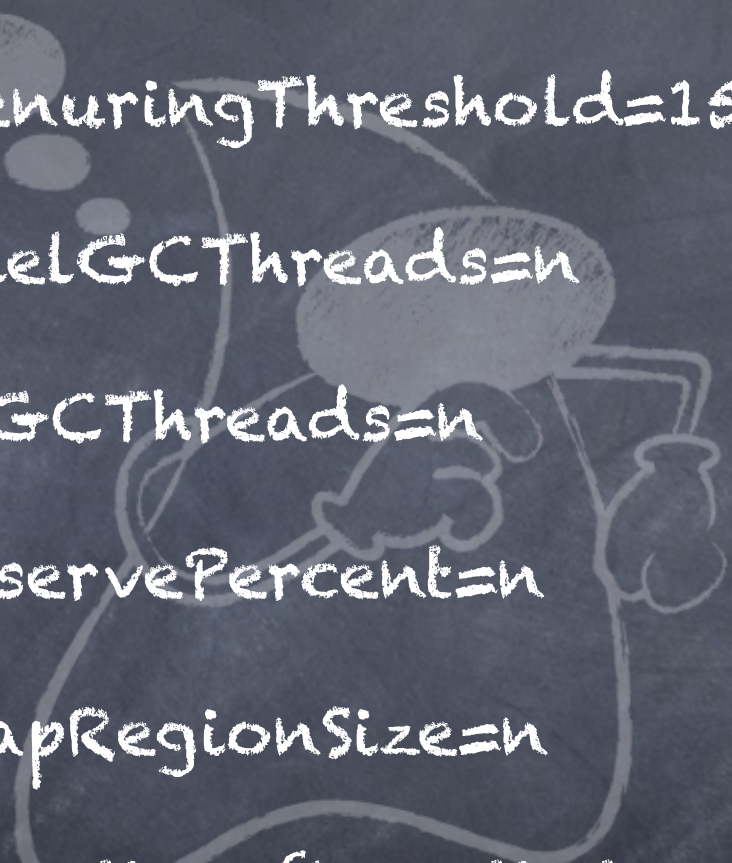
Kodewerk
Java™ Performance Services

Flags and Stuff

- -XX:+UseG1GC
- -mx, -mn
- -XX:MaxGCPauseMillis=200
- -XX:GCPauseIntervalMillis=1000
- -XX:InitiatingHeapOccupancyPercent=45
- -XX:NewRatio=2
- -XX:SurvivorRatio=8

Kodewerk
Java™ Performance Services

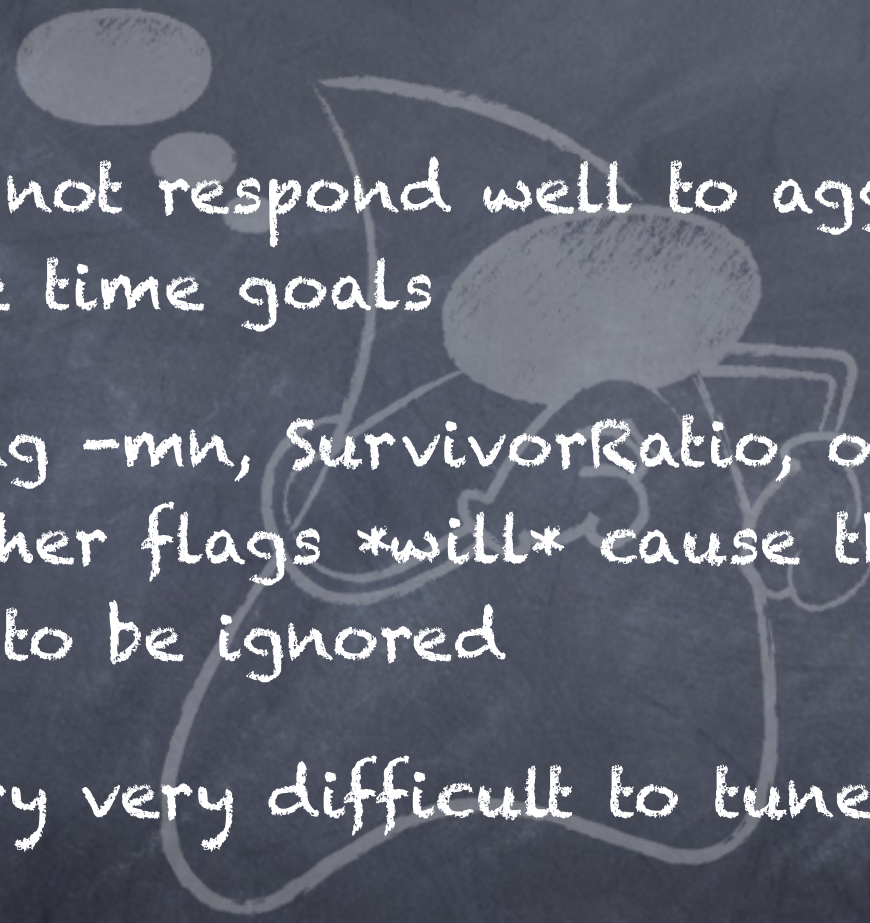
Flags and Stuff



- -XX:MaxTenuringThreshold=15
- -XX:ParallelGCThreads=n
- -XX:ConcGCThreads=n
- -XX:G1ReservePercent=n
- -XX:G1HeapRegionSize=n
- numerous other flags that get hairy

Kodewerk
Java™ Performance Services

Flags and Stuff

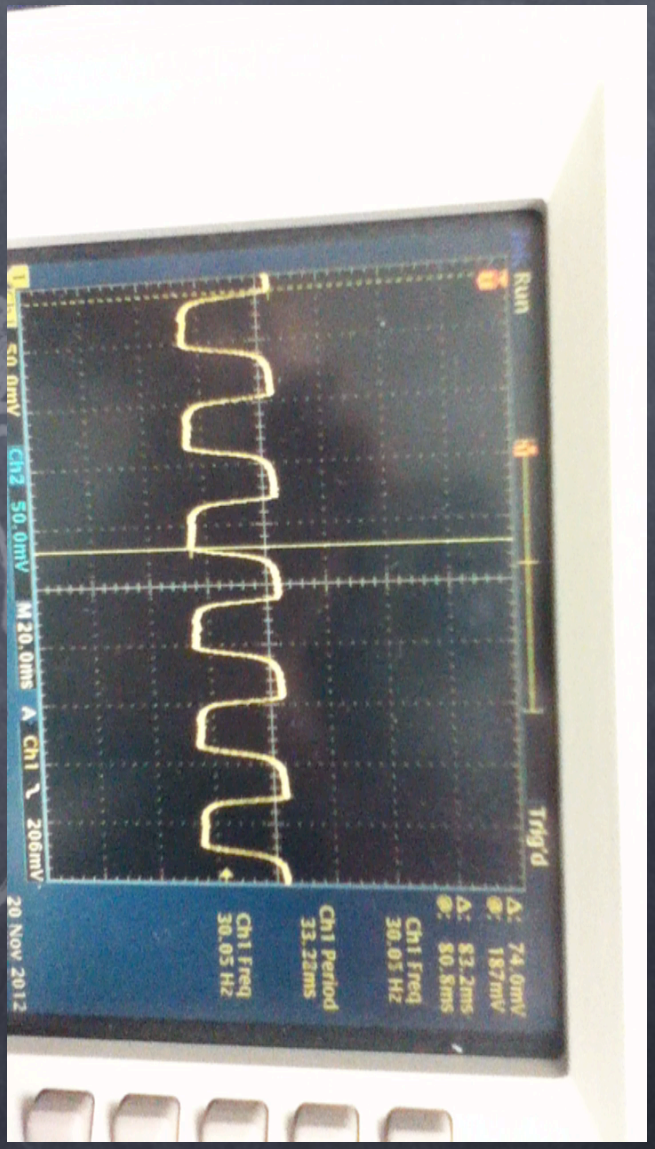


- Does not respond well to aggressive pause time goals
- Setting -mn, SurvivorRatio, or a number of other flags *will* cause the pause time to be ignored
  - very very difficult to tune

Kodewerk
Java™ Performance Services

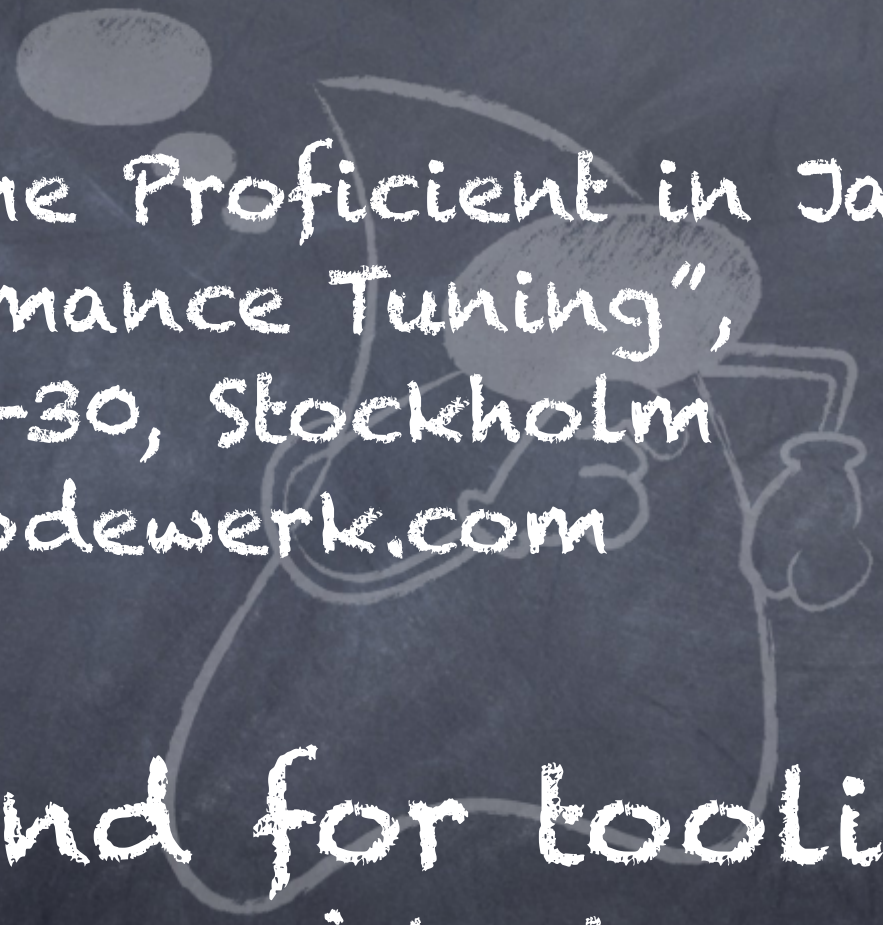
GC Jitter???

- FX application to flip screen between black and white on each vsync
  - 60hz signal==16ms update interval
- 10Gig heap, 10ms over 200ms pause time goal
  - never hit it!



Kodewerk
Java™ Performance Services

Wanna learn more?



"Become Proficient in Java Performance Tuning",
May 27-30, Stockholm
www.kodewerk.com

And for tooling

www.jclarity.com