The Squeak VM
Exploring Garbage Collection
A view from 10,000 meters to the bits

By John M McIntosh
Corporate Smalltalk Consulting Ltd.
http://www.smalltalkconsulting.com
johnmci@smalltalkconsulting.com

Maintainer of the Squeak Macintosh VM.
Blogs for OOPSLA, Camp Smalltalk, etc.
Team member **Sophie/TK4** *http://www.futureofthebook.org*

Squeak a Generational Garbage Collector

Target Platform: 16Mhz 68030 Mac SE/30 (1988)

- Copying compacting collector with two generations.
- YoungSpace GC in less than 10ms to avoid music playback problems.
- Avoid full GC if possible.
- Direct pointers, swizzle at load time, update when objects move.
- Simple allocation, move a pointer, check some things.
- IGC on allocation count, or memory needs.
- Full GC on memory needs.
- Can auto grow/shrink endOfMemory (3.0+ feature).
- Tenure objects if threshold exceeded.

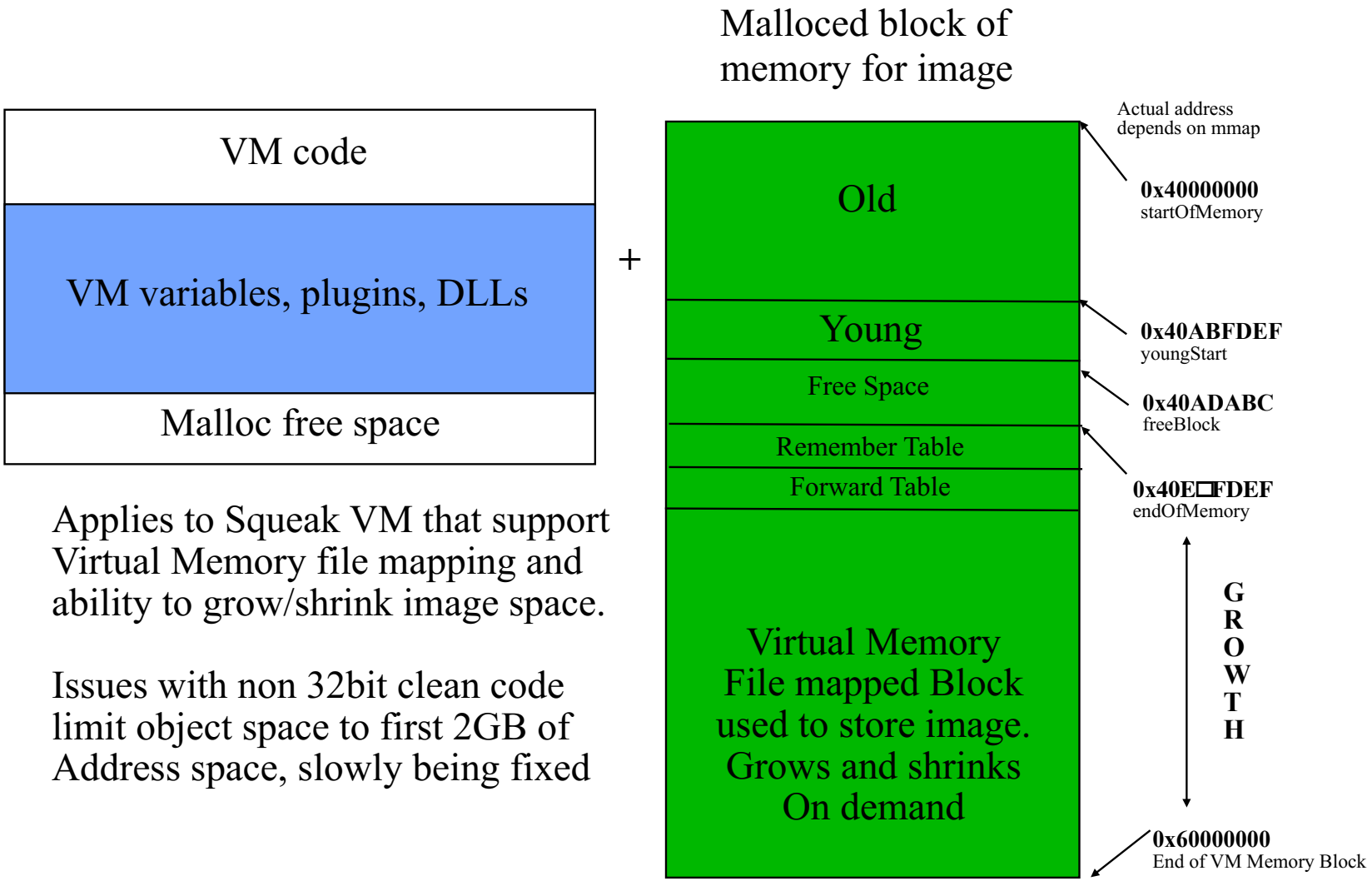
In 1976 Peter Deutsch (Smalltalk & GhostScript fame) noted:

> “Statistics show that a newly allocated datum is likely to be either 'nailed down' or abandoned within a relatively short time”.

David Ungar 1984 (now at Sun)

> “Most objects die young”.

Squeak Memory Layout



Applies to Squeak VM that support Virtual Memory file mapping and ability to grow/shrink image space.

Issues with non 32bit clean code limit object space to first 2GB of Address space, slowly being fixed

Image is read/written from/to disk via this block

Squeak Decisions

Allocating an object, means updating a pointer then nilling or zeroing the new object's words and filling in the header.

Exceed N allocations, *allocationsBetweenGCs*, invokes a IGC.

Allocate enough memory to cut in to *lowSpaceThreshold*, causes:

A IGC, and possible FullGC, and maybe signal lowspace semaphore.
For 3.x it may advance endOfMemory if possible.

Too many survivors (according to *tenuringThreshold*),

IGC will move youngStart pointer past survivors after IGC to tenure all of youngspace to oldspace.

On Full GC youngStart could get moved back towards memoryStart.

Remember Table and Forwarding Table altered on each full GC based on number of objects in image.

Remember Table full, invoke a full GC, and reallocate.

Squeak GC Flaws

Remember Table

Remembers objects, not slots in objects.

Issue: Allocate 1,000,000 collection, then placed in Oldspace, put new object reference in any slot. Each young space GC will scan all 1M slots looking for Old to Young reference and to fix oops via forward table logic.

Simple/Stupid, not yet fixed.

Forward Table Space

Re-calculated after full GC and after free space calculations, steals space from free space, then can suddenly have no free space and crash VM.

Pending fix to resize space to leave reasonable amount of free space.

Squeak GC Flaws

Weak Objects

a) Failure to finalize young weak objects when referenced by WeakArrays living in OldSpace, was only done by forcing a full GC and tenuring the young objects and realizing the need to finalize.

Fixed with April 2005 GC changes, other platforms Summer 05. Uncovered issues with (b)

b) Lots of weak objects, poor performance.

40K weak objects means scanning all of them in smalltalk to resolve corpses, results in poor performance side effects. Note Seaside impact and alternative solutions.

Simple/Stupid, not yet fixed.

Squeak GC Flaws

Smalltalk lowSpaceThreshold

200,000 for interpreted VM, trigger to cause GC activity.
Most likely to change in the near future.
Set higher 2MB?

Smalltalk lowSpaceWatcher

Logic is primitive. VM triggers semaphore if memory free drops under the lowSpaceThreshold, this causes a dialog to appear.

Much work in spring of 2005 to ensure dialog does appear, and problem process or all user processes are suspended to allow developer to debug issue.

Squeak VM tuning 2004/2005

Issue: Squeak does excessive IGC when we approach end of memory

(*lowSpaceThreshold),* if memory is used up slowly then, we will approach 1 IGC event per allocation before we reach the decision point about when to expand memory. A flaw in the decision algorithm.

Jan 2005, altered Mac VM to change growth bias, and to collect more statistics to enable a VisualWorks like memory policy. (Other platforms June 2005).

Therefore we should poke at the GC logic a bit?

a) Collect more data so we can see what the IGC/GC is doing

b) Grow to certain size before doing full GC.
This greatly reduces GC activity for certain applications.

Smalltalk setGCBiasToGrowGCLimit: 16*1024*1024.
Smalltalk setGCBiasToGrow: 1.

Squeak VM Data, array of values

1	end of old-space (0-based, read-only)
2	end of young-space (read-only)
3	end of memory (read-only)
4	allocationCount (read-only)
5	allocations between GCs (read-write)
6	survivor count tenuring threshold (read-write)
7	full GCs since startup (read-only)
8	total milliseconds in full GCs since startup (read-only)
9	incremental GCs since startup (read-only)
10	total milliseconds in incremental GCs since startup (read-only)
11	tenures of surviving objects since startup (read-only)
21	root table size (read-only)
22	root table overflows since startup (read-only)
23	bytes of extra memory to reserve for VM buffers, plugins, etc.
24	memory threshold above which shrinking object memory (rw)
25	memory headroom when growing object memory (rw)

Squeak VM Data, array of values

Additional data Summer of 2005

26 interruptChecksEveryNms - force an ioProcessEvents every N milliseconds,
in case the image is not calling getNextEvent often (rw)
27 number of times mark loop iterated for current IGC/FGC (read-only) includes ALL marking
28 number of times sweep loop iterated for current IGC/FGC (read-only)
29 number of times make forward loop iterated for current IGC/FGC (read-only)
30 number of times compact move loop iterated for current IGC/FGC (read-only)
31 number of grow memory requests (read-only)
32 number of shrink memory requests (read-only)
33 number of root table entries used for current IGC/FGC (read-only)
34 number of allocations done before current IGC/FGC (read-only)
35 number of survivor objects after current IGC/FGC (read-only)
36 millisecond clock when current IGC/FGC completed (read-only)
37 number of marked objects for Roots of the world, not including RT entries for IGC/FGC (r)
38 milliseconds taken by current IGC (read-only)
39 Number of finalization signals for Weak Objects pending when current IGC/FGC completed (r)
40 VM word size - 4 or 8 (read-only) 64 BIT SUPPORT

Squeak GC Monitoring

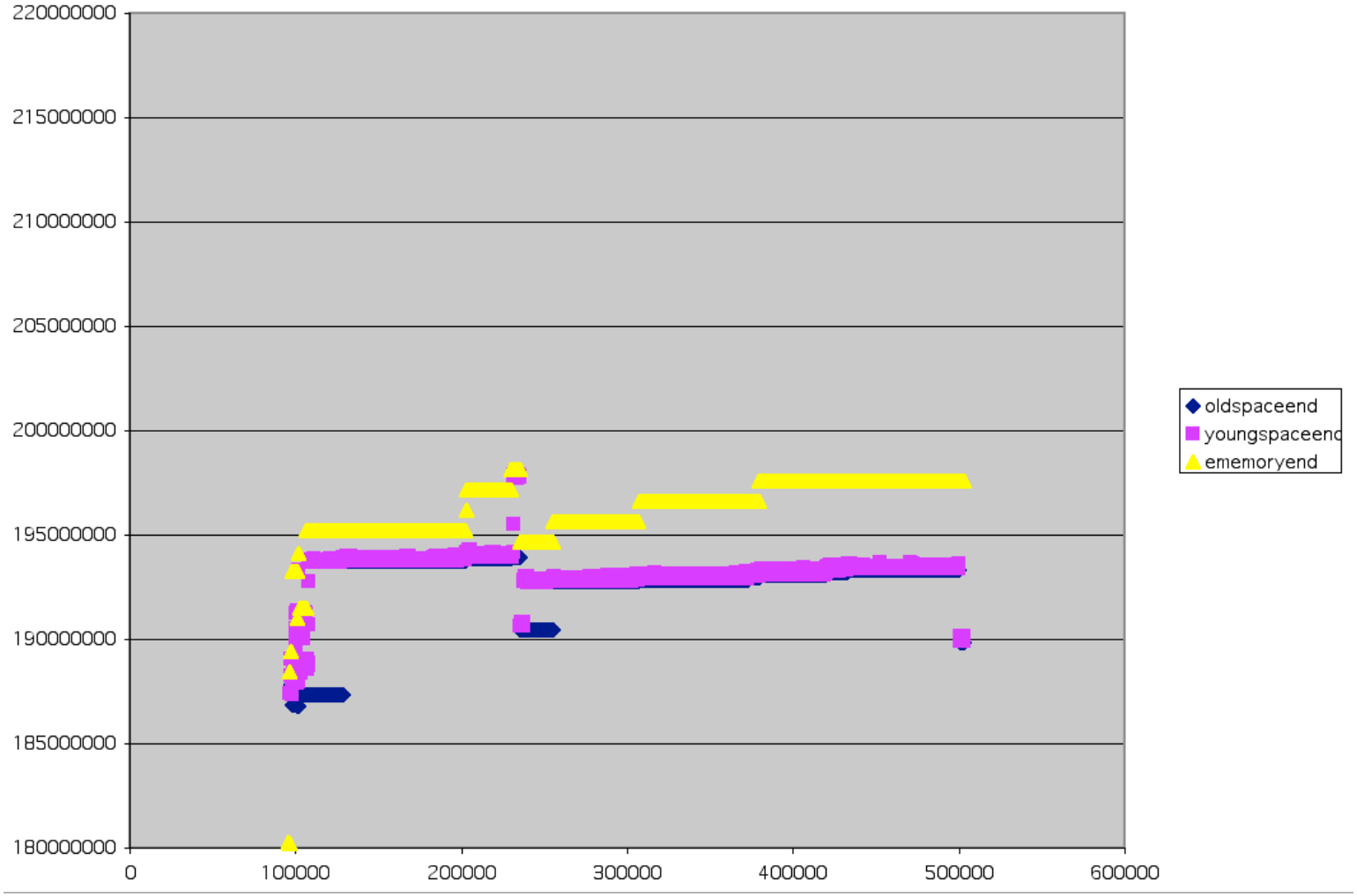
GCMonitor Class to do active/passive monitoring, & altering
Think of VisualWorks MemoryPolicy logic

```
GCMonitor>>calculateGoals
(statMarkCount ) > (statAllocationCount*2)
        ifTrue: [Smalltalk forceTenure].  "Tenure if we think too much root table marking is going on"

(statIGCDeltaTime < 1) ifTrue: “Less that 1 ms youngspace GC time, make it a bit longer”
        [target _ (Smalltalk vmParameterAt: 5)+21.
        Smalltalk vmParameterAt: 5 put: target.  "do an incremental GC after this many allocations"
        Smalltalk vmParameterAt: 6 put: target*3//4.  "tenure when more than this many objects survive the GC"].

(statIGCDeltaTime > 1) ifTrue: “Greater than 1 ms youngspace GC time, make it a bit shorter”
        [target _ ((Smalltalk vmParameterAt: 5)-27) max: 4000.
        Smalltalk vmParameterAt: 5 put: target.  "do an incremental GC after this many allocations"
        Smalltalk vmParameterAt: 6 put: target*3//4.  "tenure when more than this many objects survive the GC"].
```

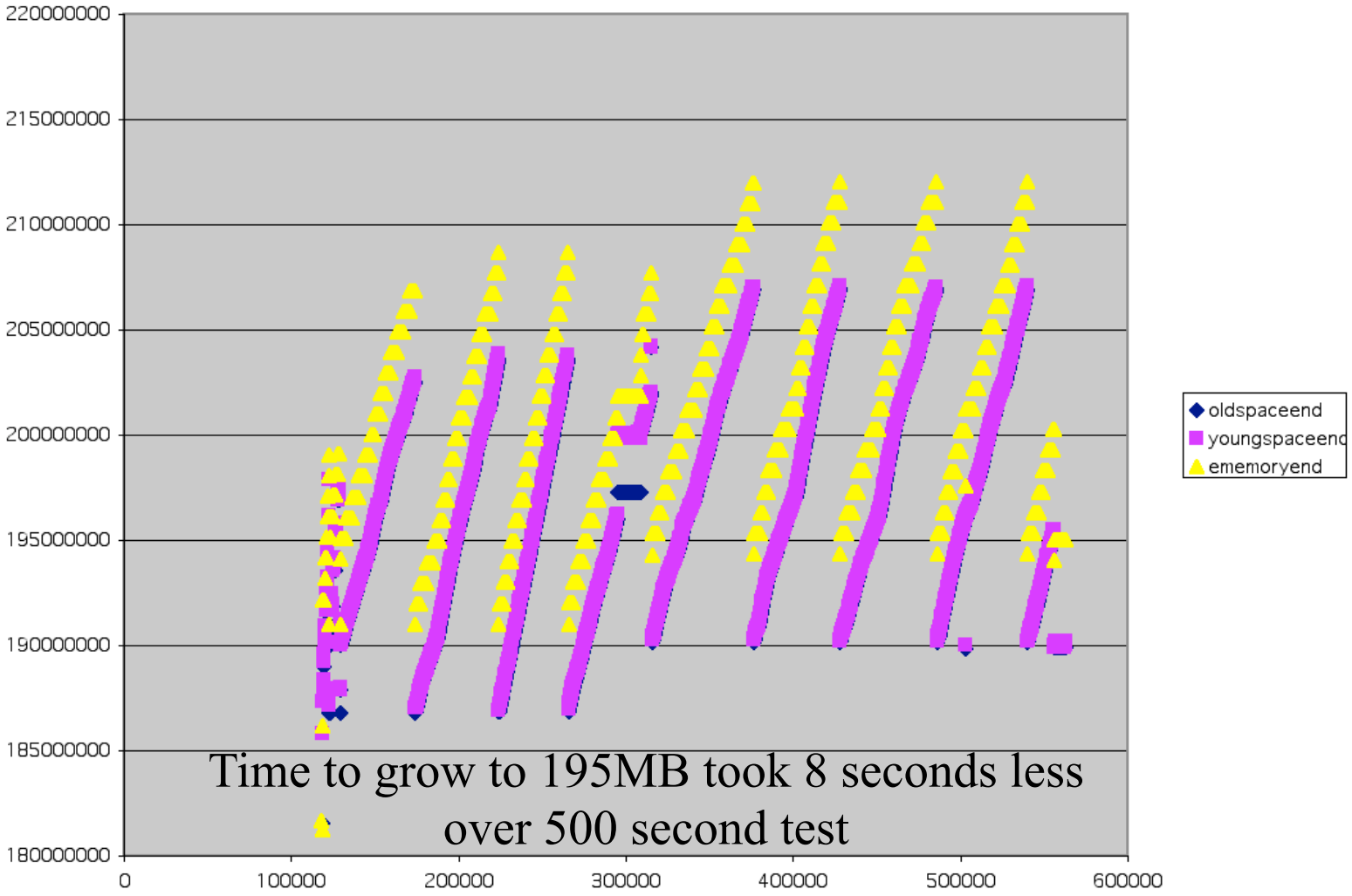
Lots of future experimentation is required.

- How many objects in young space, drives what decision?
- How many bytes in young space, drives what decision?
- Allocations, versus tenuring rate, drives what decision?
- Proper size of FreeSpace?
- Tenuring threshold should be?

Croquet memory footprint startup (original)

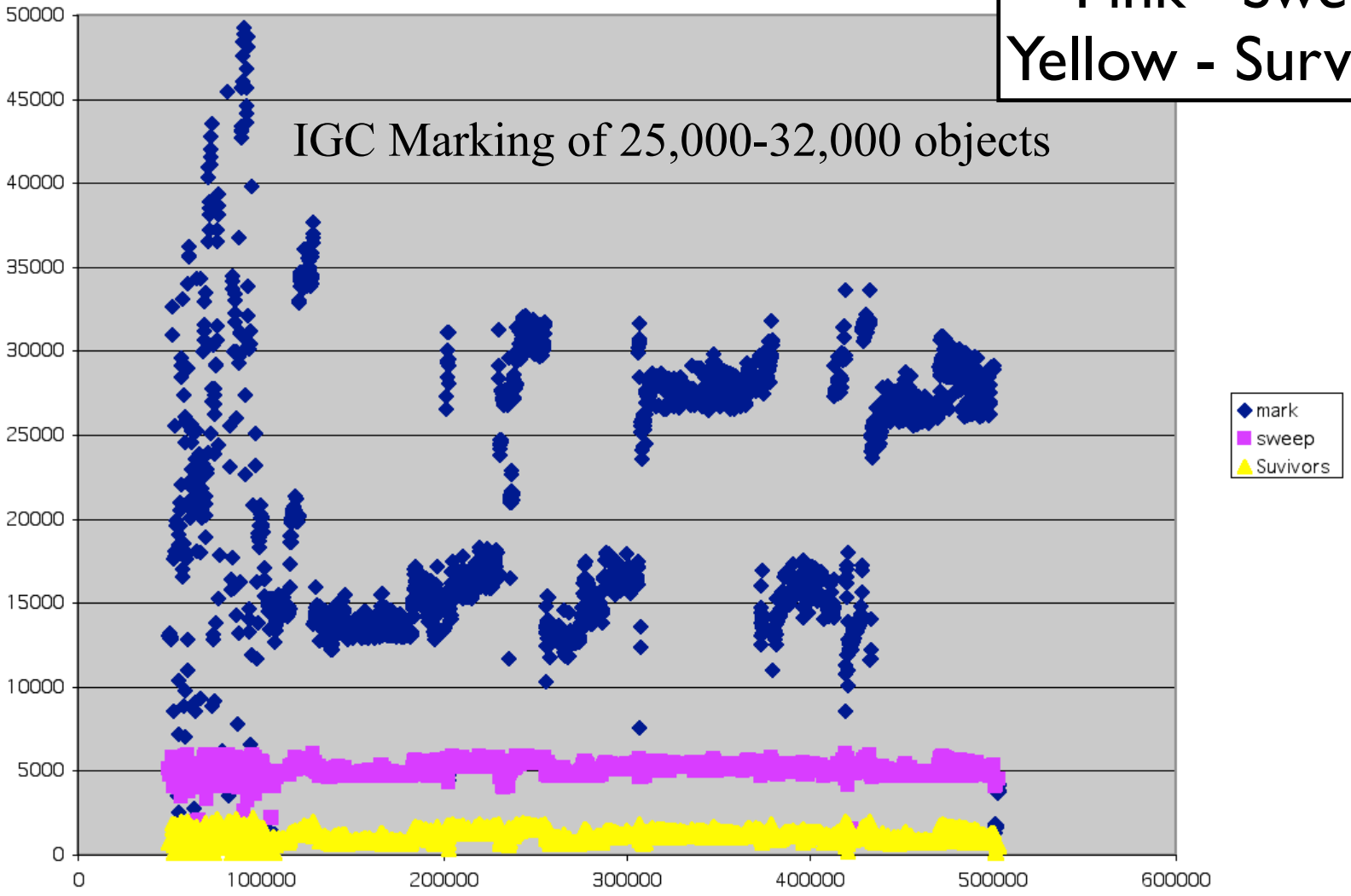


Croquet memory footprint Bias to grow

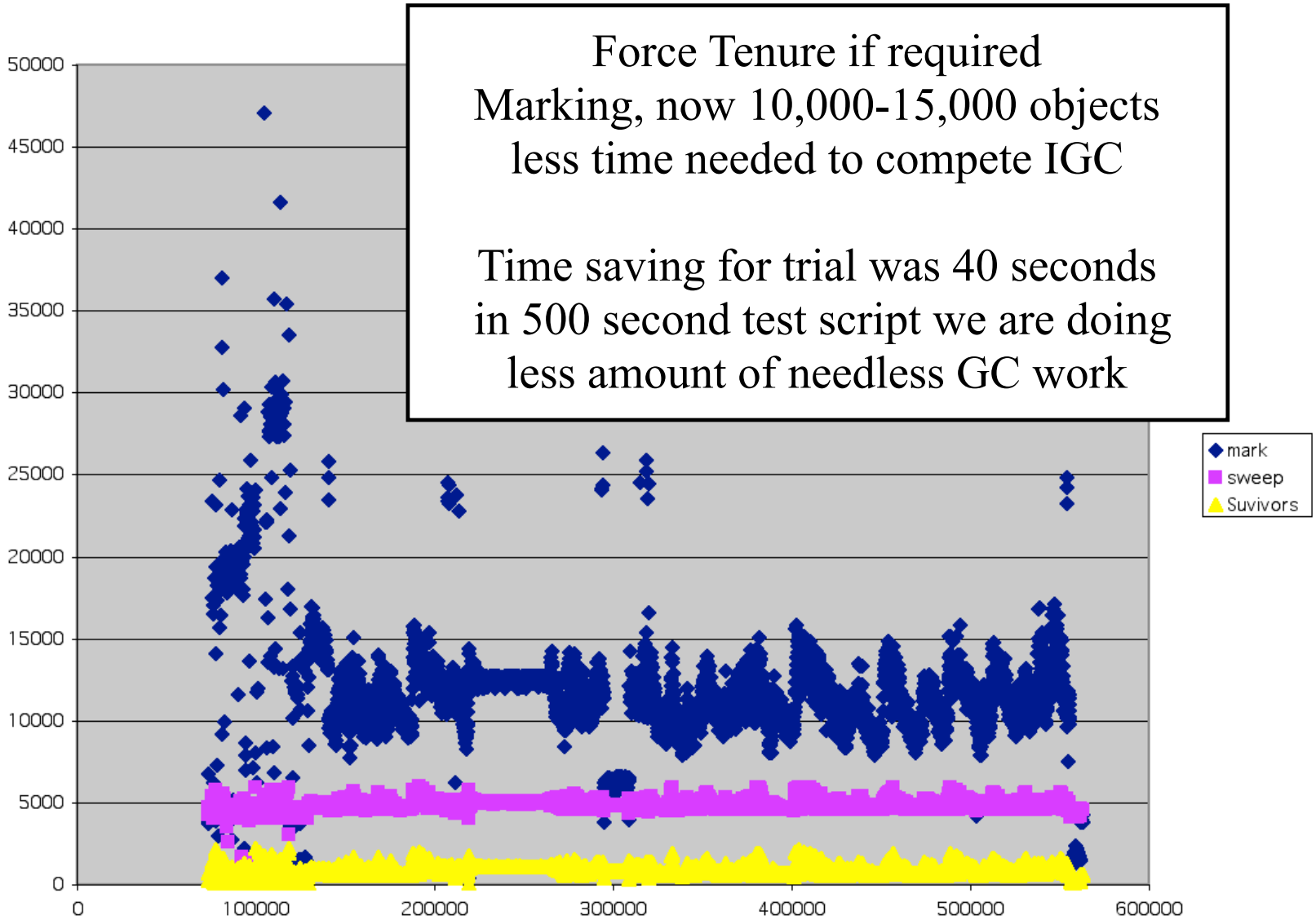


Croquet mark/sweep/survivor counts (Original)

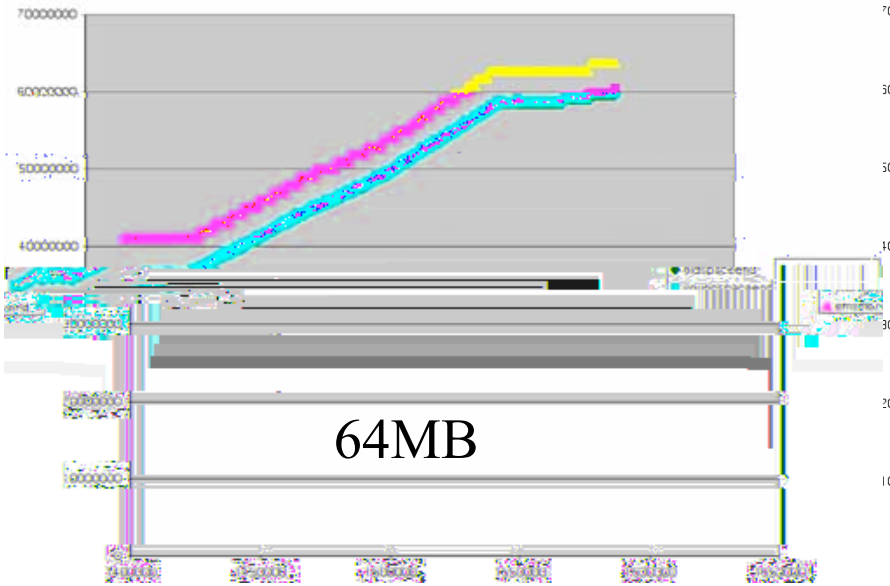
Blue - Mark
Pink - Sweep
Yellow - Survivors

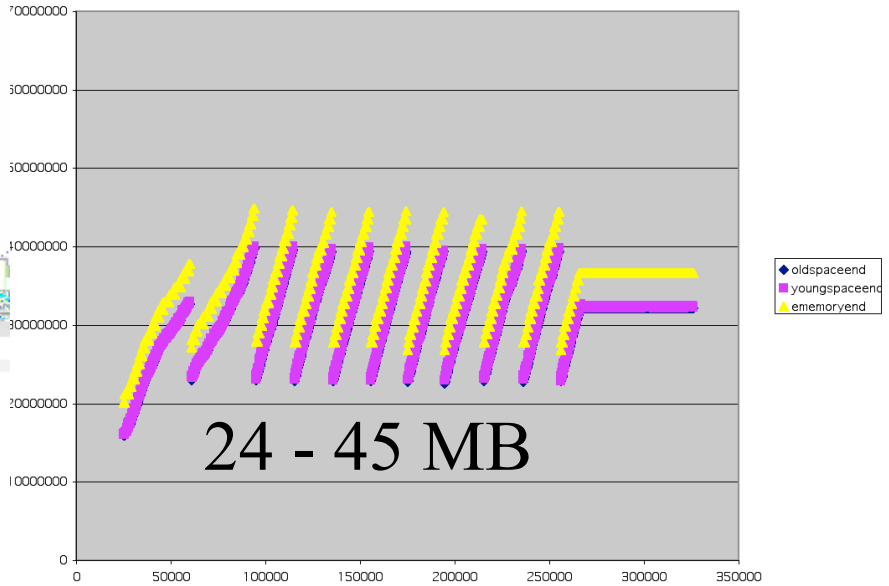


Croquet mark/sweep/survivor counts (Bias to Grow, tenure if Root Table too big)



Seaside before/after wget test





4 threads of: wget --recursive --no-parent --delete-after --non-verbose
http://localhost/seaside/alltests
until logs reach certain size.

190,342 ms of data
4 Full GC, 2,558 Incremental GC, 285 Tenures
3,069 ms Full GC time
14,079 ms Incremental GC time (17,148 ms GC total)

167,292 ms of data
5 Full GC, 2,551 Incremental GC, 592 Tenures
4,023 ms Full GC time
8,571 ms Incrmental GC time (12,594 ms GC total)

(23 seconds savings) for same amount of work 167, versus 190 seconds

Squeak Commands

Smalltalk garbageCollect — Full GC, returns bytes free.
Smalltalk garbageCollectMost — Incremental GC
Utilities vmStatisticsReportString — Report of GC numbers.
Smalltalk getVMParameters — Raw GC numbers.
Smalltalk extraVMMemory — Get/Set extra Heap Memory

Addresses a macintosh OS 9.1 or earlier os memory allocation issue

Smalltalk bytesLeftString — Get bytes free + expansions

New in summer of 2005:

Smalltalk setGCSemaphore: — Semi to signal on IGC

Smalltalk setGCBiasToGrow:/setGCBiasToGrowGCLimit:

Alter Squeak basic growth behaviour

Smalltalk isRoot:/isYoung:/rootTable/rootTableAt:

Which memory space is that object in, or is it a root?

Forwarding Logic:

Do GC (young space, or full).
Examine each Object in GC start-end range decide if it will move?
    On move compute new address stick in forward tables, mark as forward.

For roots of the world, (interpreter oops, special oops, remember table)
    Examine objects, alter old oops with new oops if marked as forwarded.
    * performance issue with root objects

For each object in GC start-end range:
    Alter slots & header, old oops with new oops.
    Move object in memory (compaction) to lower address if lower area is free.
    Update where the free pointer is and the size.
    Repeat until done, or resource problem...

