Intro to Garbage Collection in Smalltalk

* By John M McIntosh
  - Corporate Smalltalk Consulting Ltd.
  - johnmci@smalltalkconsulting.com
* What is this GC stuff:
  - Why does it run.
  - What does it do.
  - Why do I care.
  - How do I tune it?

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Garbage Collection - a worldly view

* On **comp.lang.smalltalk** in the last year:
  - foo makeEmpty, makes a collection empty, easier on the Garbage Collector!
  - It's wasting time collecting garbage, couldn't it do it concurrently?
  - Garbage Collector thrashing will get you. . .
  - I don't like Garbage Collectors (call me a luddite), but a real C programmer understands the complexity of his allocation/disposal memory logic.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

But Garbage is?

* Stuff/Objects you can’t get to via the roots of the World.
* If you can’t get to it, how do you know it’s Garbage?
* Well because you can’t get to it you *know* it’s Garbage!
* The VM knows what is Garbage, trust it.

* Yes, objects become sticky, or objects seem to disappear but these aren’t GC faults, just a lack of understanding on your part.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

GC Theory and Implementation

* If that object is going to be garbage collected:
  - When does it get GC?
  - How does it get GC ?
  - More importantly, how do we tune the garbage collector for best performance and least impact on our application? GC work *is* overhead after all.
* First we will discuss theory.
* Then we will explore implementations of theory in VisualWorks, VisualAge, and Squeak.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Evil Sticky Objects

* But I've got this object that doesn't get garbage collected!
  - UnGC objects are your problem, not the VM's!
* Discovering why an object isn't garbage collected is an art form, we could talk for a day on how to do that.
* But remember, multiple garbage collectors in most Smalltalks might mean a full global garbage collection is required to really GC an object. Sometimes that object *is* garbage it's just not garbage yet. (IE Foo allInstances)
* Using become: to zap a sticky object is a bad thing.
  - It doesn't fix the root of the problem

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Automatic Garbage Collection works

* The theories dates back to the 60s. Implementations we see today are decades old. I'm sure much of the actual code is decades old too.
* UnGCed objects are held by you, or the application framework, you will find the holder some day.
* If you can find a garbage collector bug, I'm sure you could become famous!
  - March 2001, Squeak GC issue found, VM failure on class reshape
  - F all 2000, Squeak finalization bug!

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

GC Overhead

* Critical issue for people fearing the GC.
* Pick a number between 2 and 40%.
* A conservative guess is 10% for good implementations of good algorithms.
* Some vendors will claim 3% (Perhaps).
* The trick is to hide the overhead where you won't notice it.
* Less mature implementations use slow algorithms or aren't tuned. Which fits most new language implementations since the GC is a boring part of any VM support/coding.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

GC Algorithms (Smalltalk Past and Present)

* Reference counting

* Mark/Sweep

* Copying

* Generational

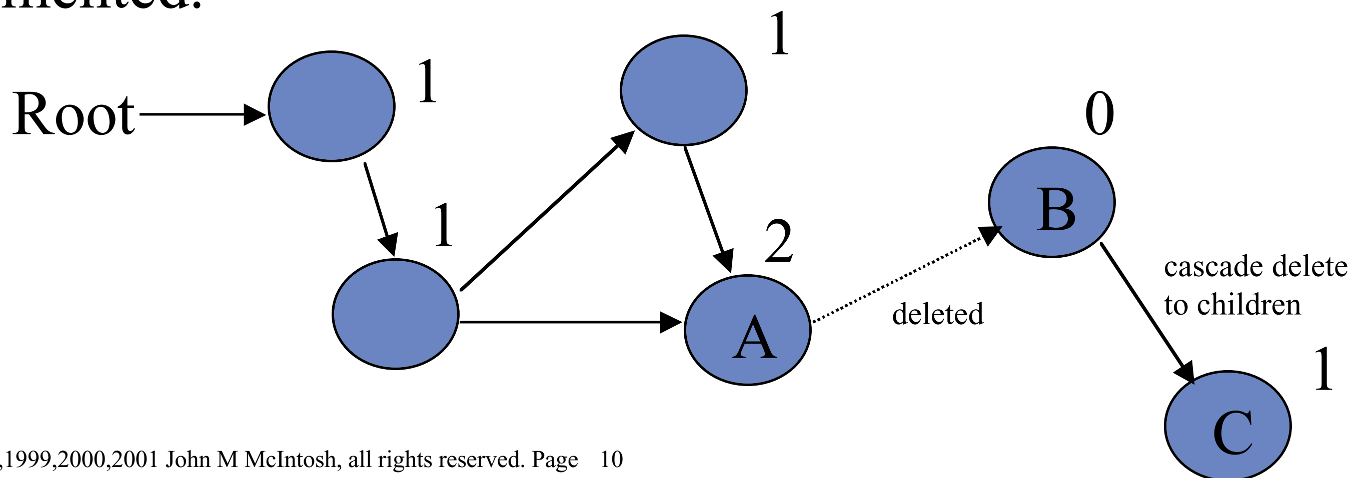
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Reference Counting

* Algorithms
  - Collins
  - Weizenbaum
  - Deutsch-Bobrow
* Original Smalltalk blue book choice.
  - currently used by Perl and other popular scripting languages.
* Easy to implement and debug.
* Cost is spread throughout computation, the GC engine is mostly hidden.
* *Simple, just count references to the object..*

Reference Counting

* Each object has a reference counter to track references.
* As references to an object are made/deleted, this counter is incremented/decremented.
* When a reference count goes to *zero*, the object is deleted and any referenced children counters get decremented.



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

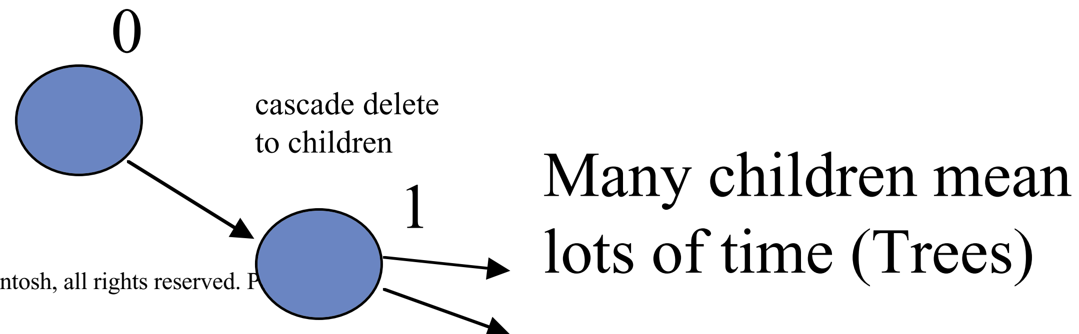
Reference Counting

* Impact on user is low, versus higher for copy algorithms.
  - Since reference counting has little impact on an interactive environment all early Smalltalk systems used it, since implementations using other algorithms had very noticeable side effects.
  - Paging cost is low. Studies showed that associated objects clump in groups, so it was very likely that objects referenced by the deleted object were on same memory page, avoiding a page-in event.
  - (Neither of these strengths/issues apply today).
* Oh, and Finalization (hang on) happens right away!
* Seems simple, but there are issues....

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Reference Counting - Issues

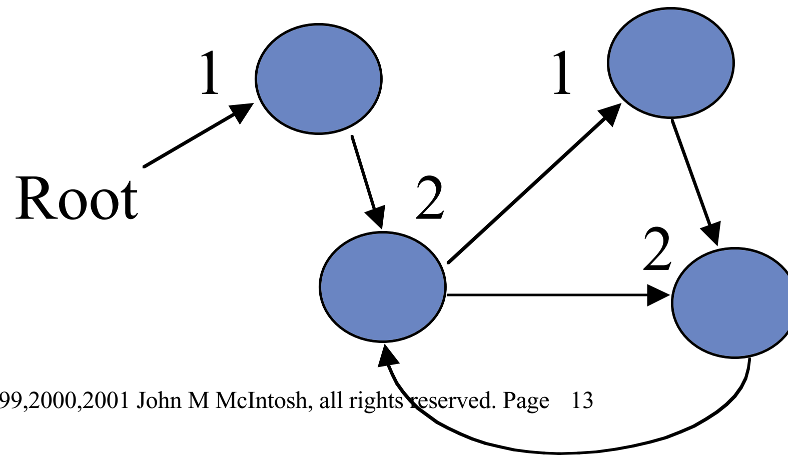
* A bug can lead to disaster. Memory leaks away.
  - An object could get de-referenced, but not GCed, a bad thing.
  - In early Smalltalk systems, a small extension to the VM called the Tracer was used to clone Smalltalk Images. It also fixed reference count problems for the developers. Many bits in your image today were born in the 70’s and fixed by the Tracer. (See Squeak)
* Do we need counter for each object, or just a bit?
* Cost! Deleting an object and it’s children can be expensive!



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved. P

Reference Counting - Issues

- **Overhead is a big factor**. Early Smalltalk systems showed reference counting cost upwards of 40% of run time if not implemented in hardware. (special CPU, not generic)
- Cyclic data structures and counter overflows are issues.
  - Solved by using another GC algorithm (Mark/Sweep), which will run on demand to fix these problems.



Cycles in reference links aren't handled correctly.

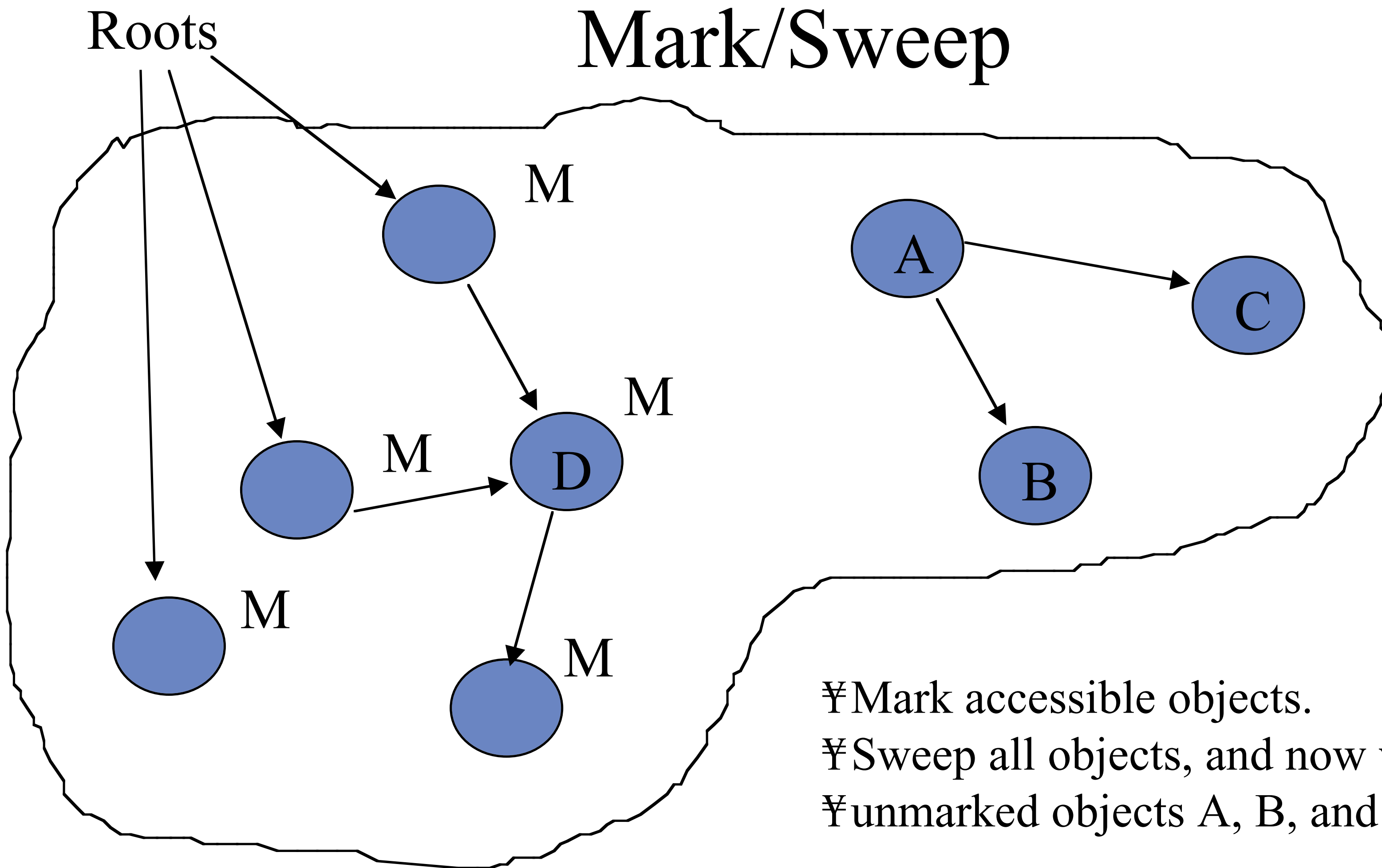
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Mark/Sweep

* Algorithms
  - McCarthy
  - Schorr-Waite
  - Boehm-Demers-Weiser
  - Deutsch-Schorr-Waite
* Start at the roots of the world, and examine each object.
* For each object, mark it, and trace its children.
* When done, sweep memory looking for unmarked objects, these objects are free.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Mark/Sweep



¥Mark accessible objects.
¥Sweep all objects, and now we realize,
¥unmarked objects A, B, and C are free

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Mark/Sweep

* No cost on pointer references.
  - Your application runs faster, since you defer the GC work until later, then you pay.
* Storage costs -> it might be free.
  - All we need is a wee bit in the object header...
  - Can be implemented as concurrent or incremental.
  - Many implementations of Java use a mark/sweep collector, running on a concurrent low priority thread. Pauses in the GUI allow the GC to run. VW uses the same solution (non-concurrent).
* Trick is to decide when to invoke a GC cycle.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Mark/Sweep - Issues

* Sweep touches each object and thrashes pageable memory.
  - Good algorithms that use mark bitmaps can reduce problem by not directly touching object which is easier on VM subsystem.
    * Mostly ignored today (I think)
* Sweep needs to check each object, perhaps Millions!
  - There are ways to tie sweeps to allocation, anyone do this?
* Recursive operation which could run out of memory.
  - This can be recovered from. (Slow, saw impact in Squeak, 2000).
* Cost is as good as copying algorithms.
  - Copy algorithms are better if you have lots of allocations and short- lived objects.
* Can be concurrent .... But means complexity!

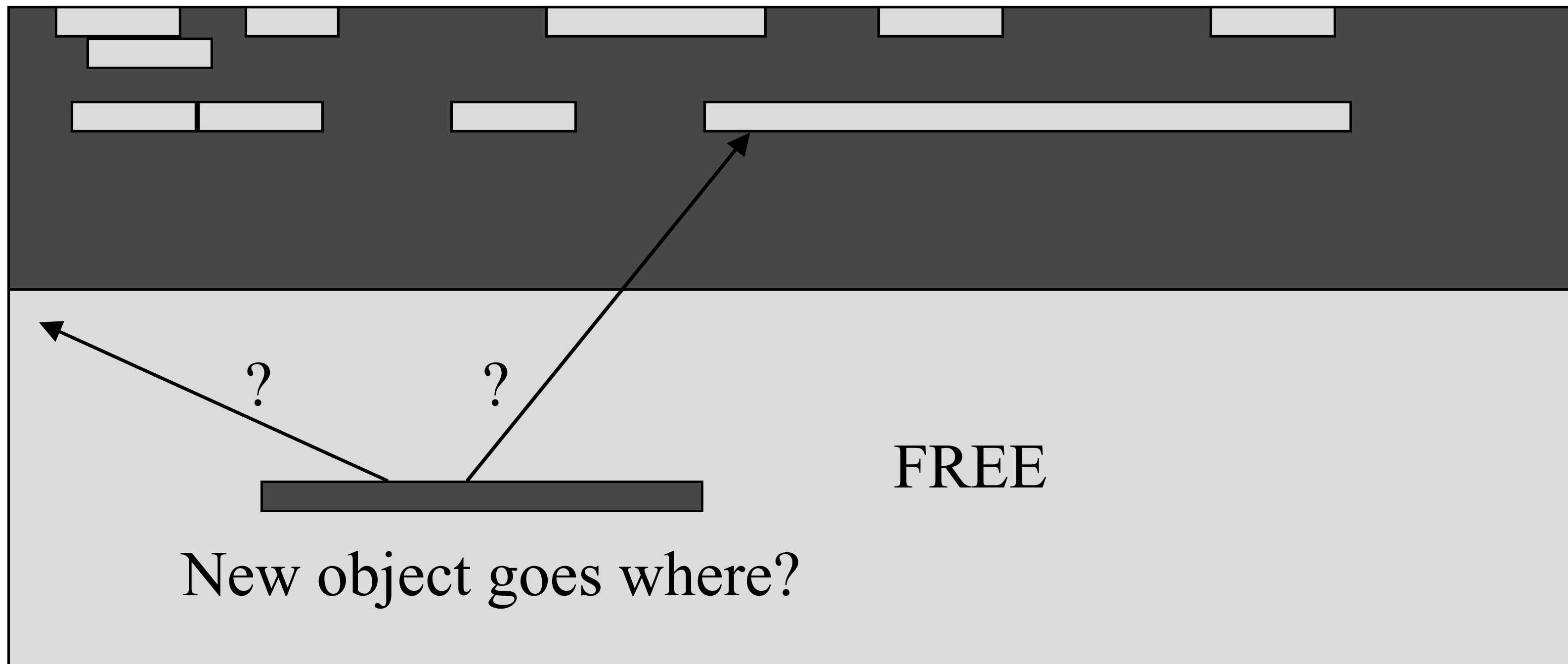
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Allocating swiss cheese

* Freed objects leave holes in memory!
* Both mark/sweep and reference counting schemes share this problem.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Swiss Cheese - Allocation?



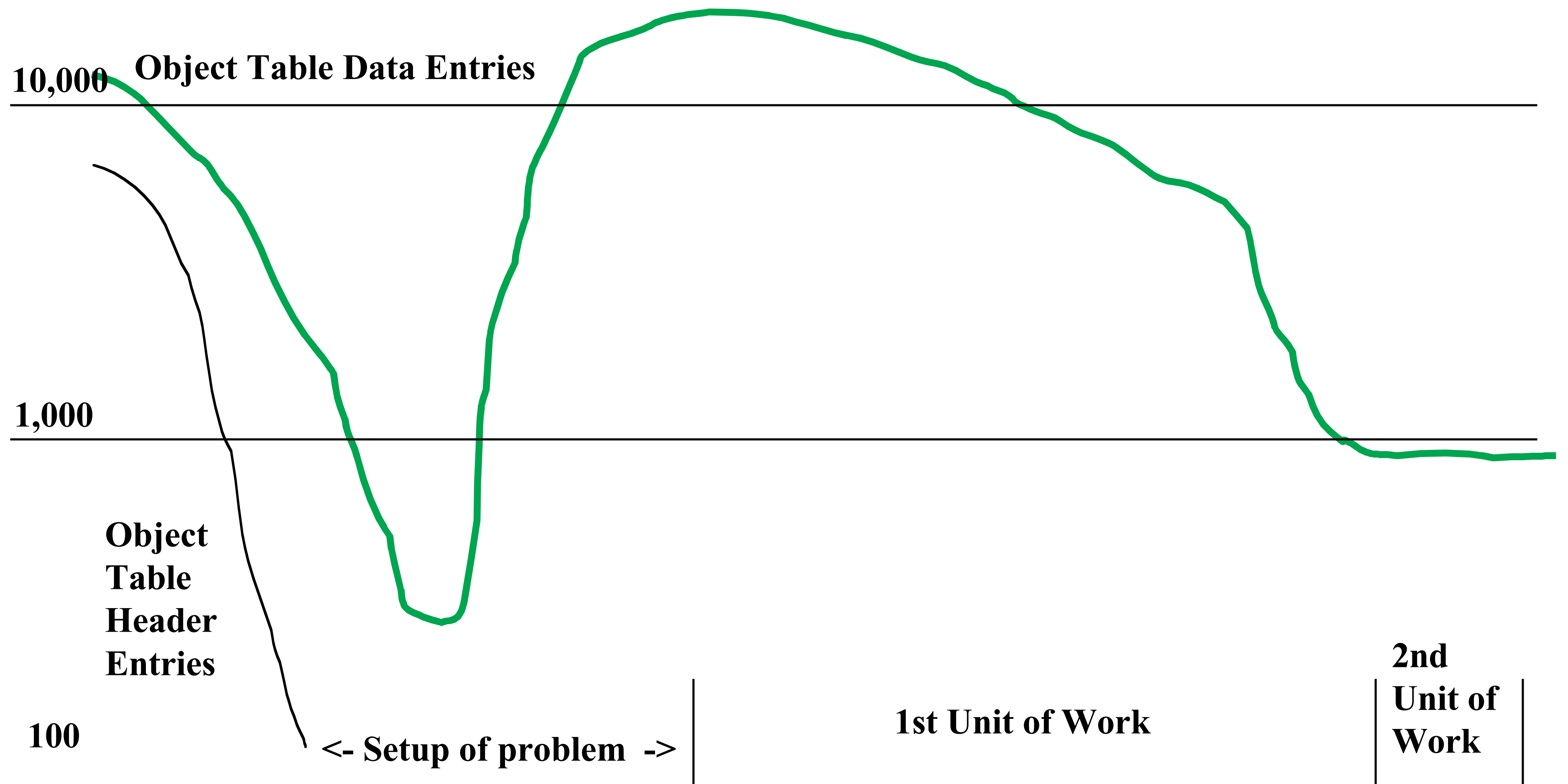
* Which hole should a new object be allocated from?
* How to allocate an object is a serious question, many theories exist.
* Which the Virtual machine uses is a real factor

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Explain the following VW problem, propose a solution

100,000 Logarithmic scale

First unit of work takes 10x longer than 2nd unit of work. Why?



Time Taken, (work units have same computation effort)

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved

Swiss Cheese - Free Space?

* Sum of total free space is large, but fragmentation means you can't allocate large objects, or allocation takes a *long* time. Like being surrounded by sea water, but none of it is drinkable!. . .

* Solution: Compaction is required!
  - Means moving memory and updating references.
  - This is **Expensive!**

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Mark/Sweep Compaction

* Algorithms
  - Two-Finger
  - Lisp-2
  - Table Based
  - Treaded
  - Jonkers
* Basically on Sweep phase clump all objects together, slide objects together, and the holes float up!
* However it touches all memory, moves all objects in the worst possible cases. Expensive, but can be avoided until required!

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Mark/Sweep Compaction

* Triggered if: (All decision points in VisualWorks)
  - Fragmentation has gotten too large. (VW) (some counters not all)
  - Largest free block is below some threshold. (Squeak, VW)
  - A large object cannot be allocated. (Squeak,VW)
* Object Tables (Good or bad?) (VisualWorks?)
  - This makes address changes easy. Remember, a reference isn't a memory address. It is a descriptor to an object. The VM must at some point map references to virtual memory addresses, and this is usually done via a lookup table known as the Object Table. An reference change mean a cheap table entry update.
* **In general, compaction events are expensive**.
  - Early Smalltalk systems expressed them in minutes.
    The same can apply today!

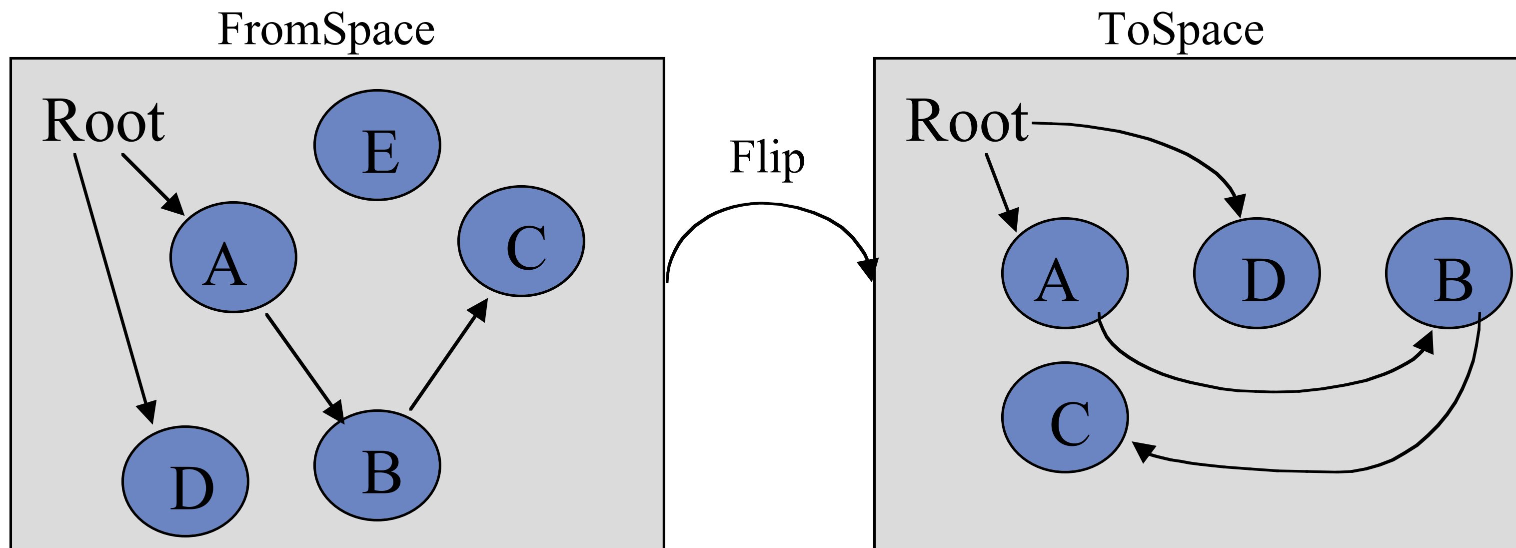
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Copying

* Algorithms
  - Cheney
  - Fenichel-Yochelson
* Two areas called semi-spaces are used. One is live and called the *FromSpace*. The other is empty and called the *ToSpace,* mmm actually the names aren't important.
* Allocate new objects in FromSpace, when FromSpace is full then copy survivor objects to the empty semi-space ToSpace.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Copying - The Flip or Scavenge



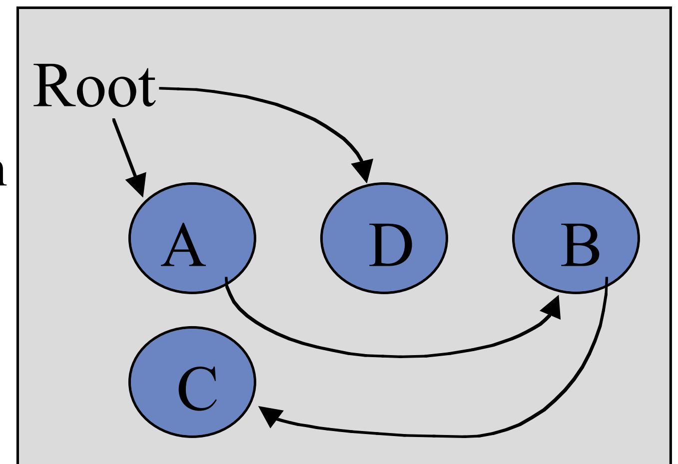
* When FromSpace is full we Flip to ToSpace:
  - Copy roots of the world to ToSpace.
  - Copy root accessible objects to ToSpace.
  - Copy objects accessible from root accessible objects to ToSpace.
  - Repeat above til done.
* Cost is related to number of accessible objects in FromSpace.

Notice change of Placement and E isn't copied.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

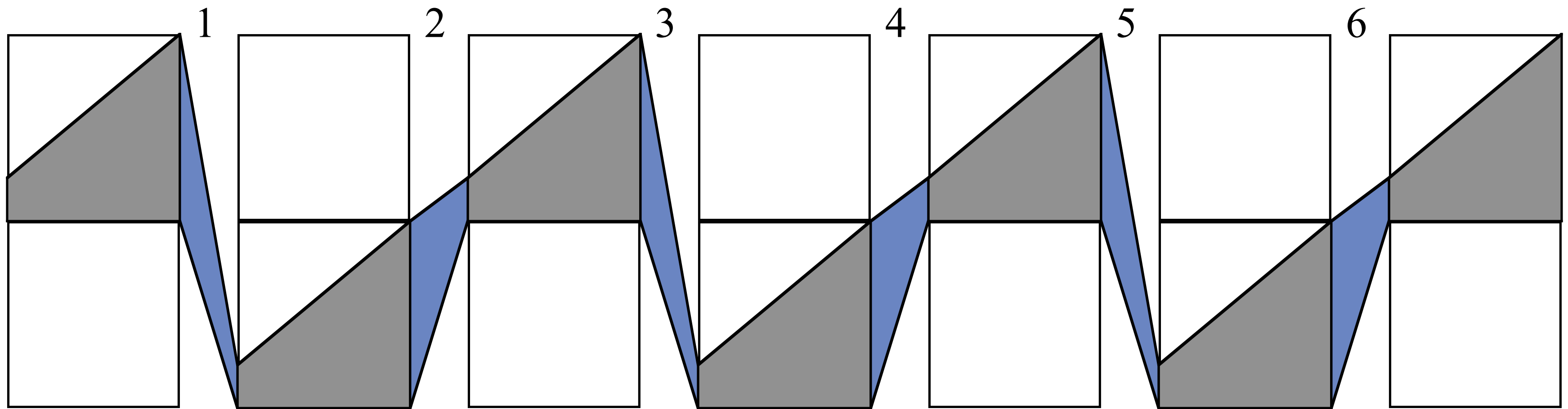
Copying - Nice Things

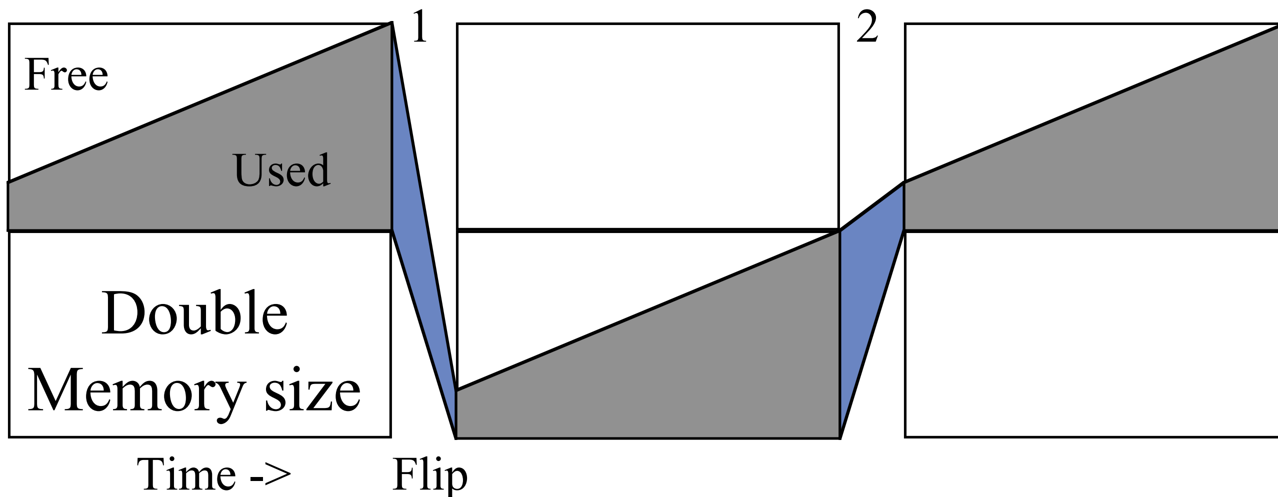
* Allocation is very cheap. Objects grow to the bottom of semi-space, so we have a short allocation path. A flip ensures memory is compacted, no fragmentation occurs.
* VM hardware can indicate boundaries of semi-spaces.
* Touches only live objects (garbage is never touched)
* Object locality?
  - Sophisticated algorithms can copy objects based on relationships, increasing the probability that an object's children live on the same page of memory.
    Couldn't say if anyone attempts this



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Copying - Adding Memory!





Copy cycle cost is the same, but two copy cycles versus six means 1/3 the memory moved. Less overhead, and application runs faster.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Copying - Beyond the Copy Bump

* Needs double the memory.
  - *"No such thing as a free lunch." (Maybe we don't care today)*
* Moves Large objects around.
  - Use LargeSpace for large objects, only manipulate headers.
  - Need FixedSpace to manage fixed objects
* Old objects get moved around forever...
  - Division of memory into read-only, fixed and large will improve performance. But rules to classify an object are?
* Paging? Does that happen anymore? Could be cheaper to add more memory. . .
* But lots of survivors can ruin your day.

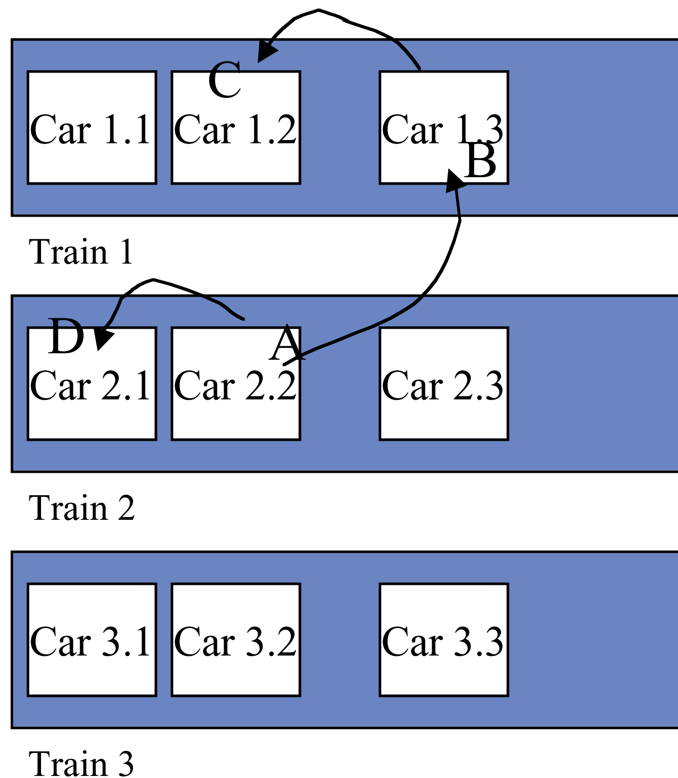
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

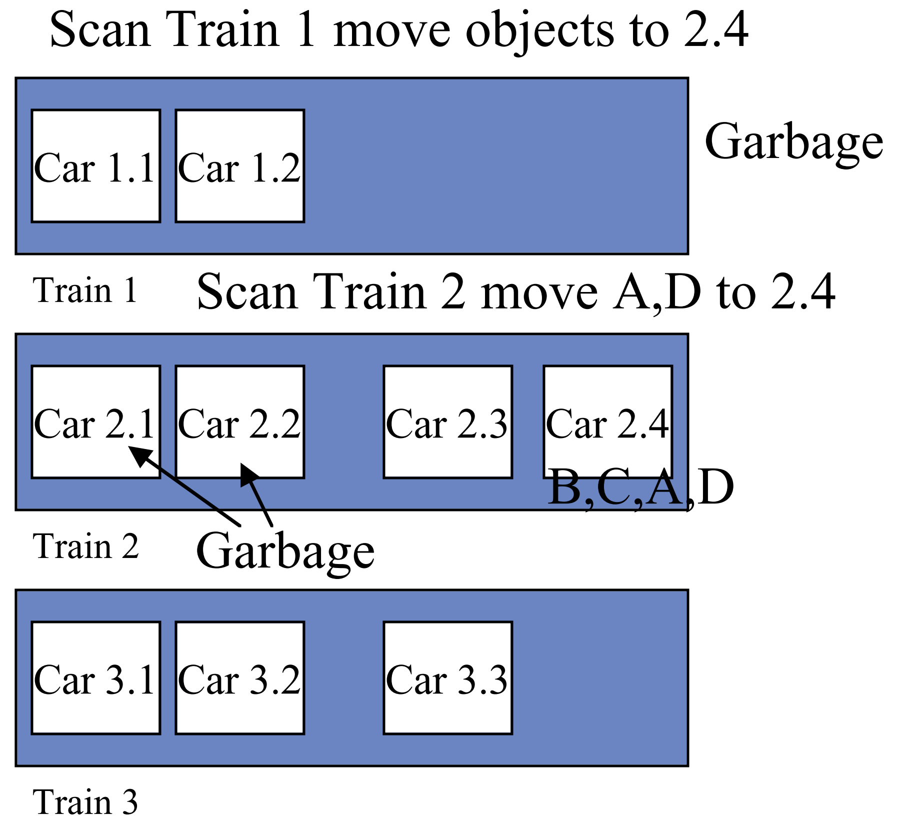
Copy or Mark/Sweep?

* Hardware dictates that Sweep is faster than Copy because linear memory access is faster than random access to memory locations. Tough to measure (nanoseconds)
* But, hey you don’t have a choice. . . You must live with what you have, only Java claimed to provide the feature of changing out the GC if you wanted (but do they?)
* In reality no GC algorithm is "best", but some are very good for Smalltalk , and not for other languages.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Train Algorithm (A Java sideShow)





PS Look at VA segments and wonder

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Generational Garbage Collector

* Algorithms
  - Lieberman-Hewitt
  - Moon
  - Ungar
  - Wilson

* Most agree this theory is the best for Smalltalk.
* Basically we only worry about active live objects.

Generational Garbage Collector

* In 1976 Peter Deutsch 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
  - 'Most objects die young'
* Now remember studies show:
  - Only 2% of Smalltalk objects survive infancy. Other languages might have different conclusions
  - 80 to 98 percent of Objects die before another MB of memory is allocated. (Hold that thought, is this still true?)
* So concentrate efforts on survivor objects.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Generational Garbage Collector

* Separate objects by age into two or more regions.
  - For example: Tektronix 4406 Smalltalk used seven regions, 3 bits
* Allocate objects in *new* space, when full copy accessible objects to *old* space. This is known as a *scavenge* event.
* Movement of objects to old space is called *tenuring*.
* Objects must have a high death rate and low old to young object references. (Eh?). . . Both very important issues, I'll explain in a few minutes.

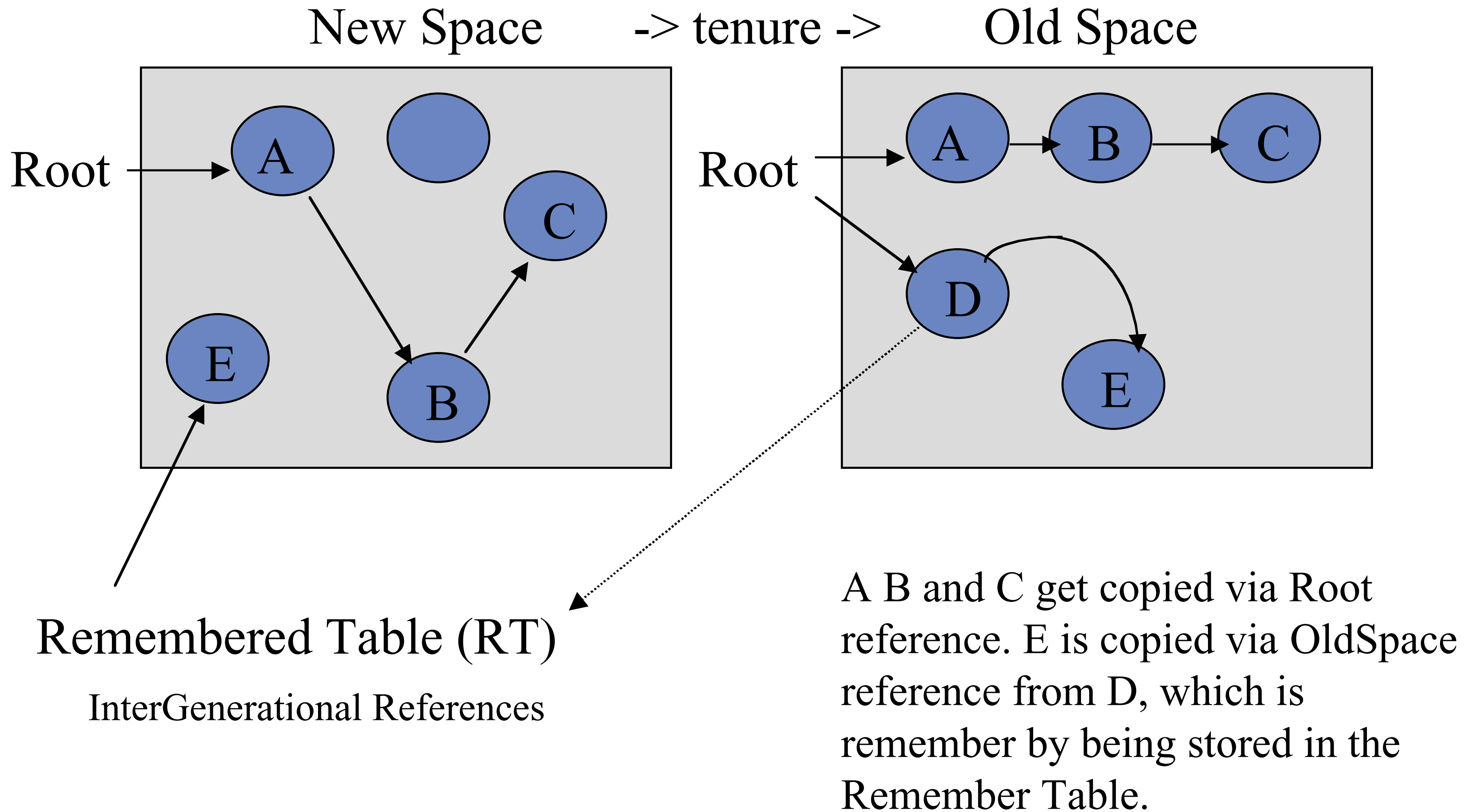
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

InterGenerational Pointers (Remember Tables)

* Must track references from old generation to new objects. Why? We don't want to search OldSpace for references into NewSpace, but how? Now remember:
  - (a) Few references from old to young are made: (< 4%).
  - (b) Most active objects are in NewSpace not OldSpace.
* Solution: Track these references as they happen!
  - In most Smalltalk systems this tracking storage is known as the Remembered Table (RT).
  - Tracking adds a minor bit of overhead. . .
  - But is there an issue?

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

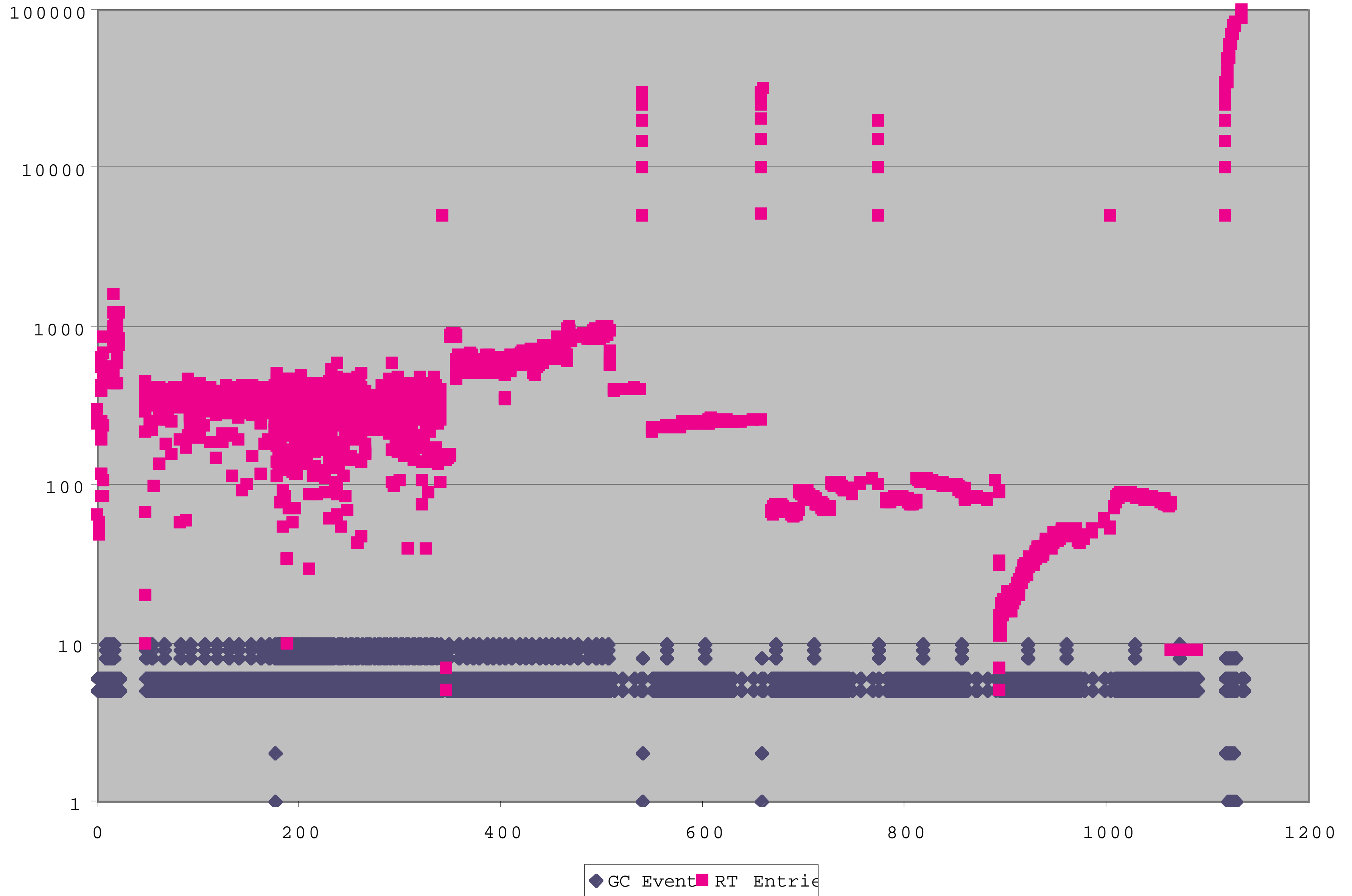
Generational Scavenge Event



A B and C get copied via Root reference. E is copied via OldSpace reference from D, which is remember by being stored in the Remember Table.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VM Failure occurs at end of chart, Why?



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Generations?

* Multiple generations are good, but only 2 or 3 are needed.
  - We could cascade objects down a set of OldSpace regions (early tek solution). The longer an object lives, the further down it goes, but effectiveness versus complexity gives diminishing returns.
  - Once a tenured object becomes garbage, we need another method to collect it, and a compacting Mark/Sweep collector is needed.
* Tuning a generational garbage collector is complex and time consuming. How many generations should we do? When and what should we tenure?
* David Ungar and Frank Jackson wrote many of the rules....

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualWorks Eden

* Ungar and Jackson Rules:
  (1) Only tenure when necessary.
  (2) Only tenure as many objects as necessary.
* These GC rules are fully exposed in VisualWorks creation space, or what we know as NewSpace:

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

A few items before the break:

* Stack Space
* Weak Objects
* Finalization

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Stack Space

* Each method sent needs an activation record, or context object. This exists for the life of the executing method.
* As much as 80% of objects created are context objects.
  - If we could avoid allocating, initializing, and garbage collecting these objects, we could make things run faster!
* Solutions:
  - Implement context allocation/deallocation as a stack. This is VW's StackSpace.
    * If StackSpace is full, older contexts are converted into objects.
  - Squeak has special MethodContext link-list to shortcut much of the allocation work on reuse of a context.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Weak Objects a GC Extension...

* The rule was:
  - If an object is accessible, then it isn't garbage.
* The weak reference concept changes that rule:
  - If the object is **only** accessible by a *weak* reference, then it can be garbage collected. If a *strong* (non-weak) reference exists, then the object cannot be GCed.
  - If a weak object is going to be or is garbage collected, then notify any interested parties. This is called *finalization*.
* Weak Objects are not Evil! They just have Weaknesses.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Weak Objects - VW Examples

* (1) Symbols are unique, but how?
  - The class Symbol has a class variable that contains an array of WeakArrays, defining all symbols in the image.
  - If you create a new symbol, it is hashed into one of the WeakArrays. If you refer to an existing symbol, the compiler finds the reference to the symbol in the WeakArray which is what the methodcontext points to.
  - If you delete the method containing the symbol, it might be the last strong reference to that symbol! If this is true, at some point the symbol is garbage collected, and finalization takes place. This puts a zero into the WeakArray slot, and that symbol disappears from your image!
* (2) The VW ObjectMemory used finalization to signal when a scavenge event has happened (in 2.5x, not 5.x).

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Weak Objects

* Each implementation has a different way of making object weak, indicating finalization, and doing the ’handshake’
  - VisualAge foo makeWeak
  - VW & Squeak see WeakArray
* Some implementations give pre-notification of finalization.
* All you need to do is read the docs.
* Remember Finalization can be slow and untimely.
  - Usually applications want instantaneous finalization, unable to achieve this results in nasty issues shortly after delivery.
    * VisualAge provides Process finalizeCycle to force finalization.
    * Ephemeron are? (fixing issues with finalization order).

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

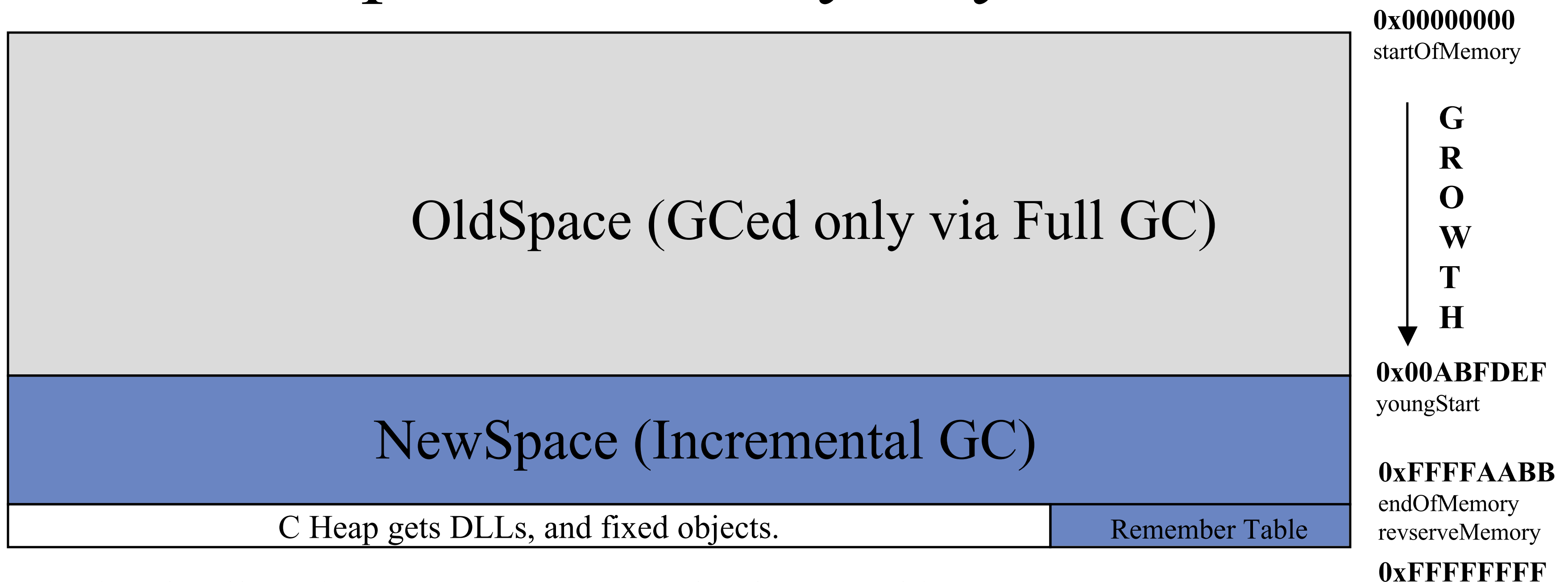
Thoughts

* GC theories are old, well understood algorithms.
* Each has trade-offs.
* Language differences will impact choice.
* Your mileage may vary.
* Nope, you can't turn it off!
  - Ah, maybe you can, but it will hurt.

* **Onward to some concrete implementations**

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Squeak Memory Layout



* 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 after IGC, moves youngStart down.
* Use LowSpace semaphore to signal memory problems

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Squeak Decisions

* Allocate an object, means updating a pointer then nilling/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 signal lowspace semaphore.
  - In 3.0 it may advance endOfMemory if possible.
* Too many survivors (according to *tenuringThreshold*),
  - IGC will move youngStart pointer past survivors after IGC.
* On Full GC youngStart could get moved back towards memoryStart.
* Remember Table is fixed size, if full this triggers fullGC.
* MethodContexts, allocated as objects, and remembered on free chain.
  - Cheaper initialization to reduce creation/reuse costs.
* Too small a forwarding table (set in VM) means multiple full GCs

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

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 headroom when growing object memory (rw)
25	memory threshold above which shrinking object memory (rw)

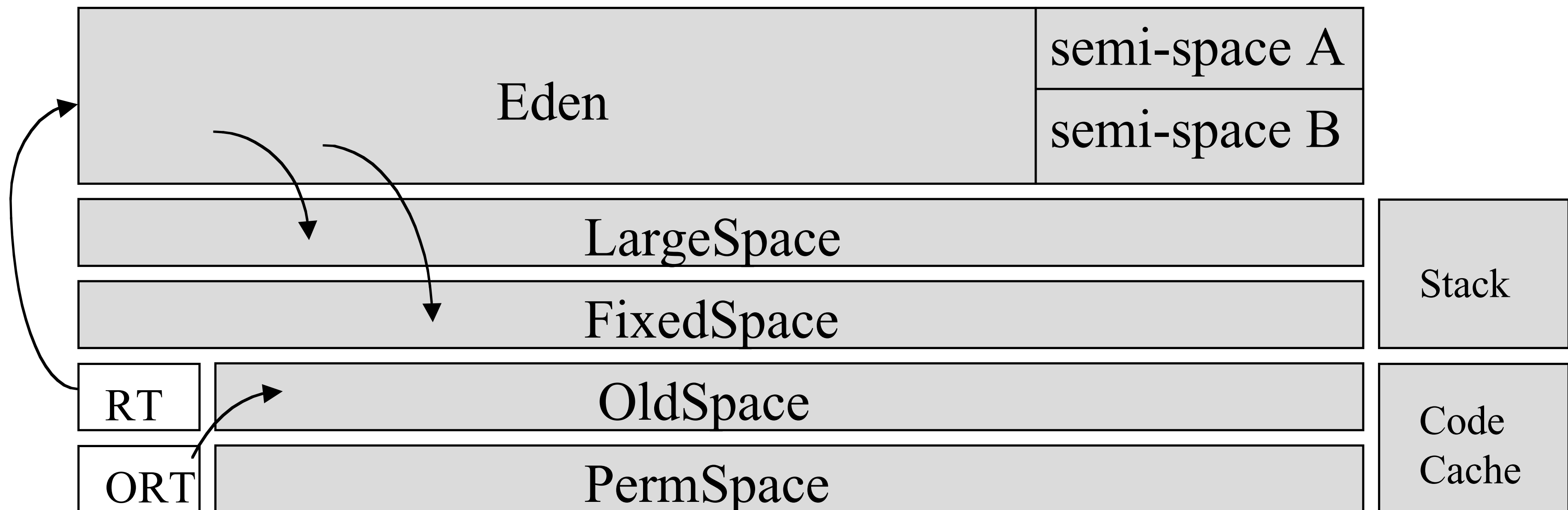
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Squeak Low Space

* Smalltalk lowSpaceThreshold
  - 200,000 for interpreted VM, 400,000 for JIT.
* Smalltalk lowSpaceWatcher
  - Logic is primitive. VM triggers semaphore if memory free drops under the lowSpaceThreshold, this causes a dialog to appear. Also includes logic for memoryHogs API but not used anywhere.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualWorks Memory Layout V5.i2



* Allocate objects or headers in Eden (bodies go in Eden, Large, or Fixed).
* Full? Copy Eden and active semi-space survivors to empty semi-space.
* When semi-space use exceeds threshold, tenure some objects to OldSpace.
* Once in OldSpace, use a Mark/Sweep GC to find garbage.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW Eden Tuning

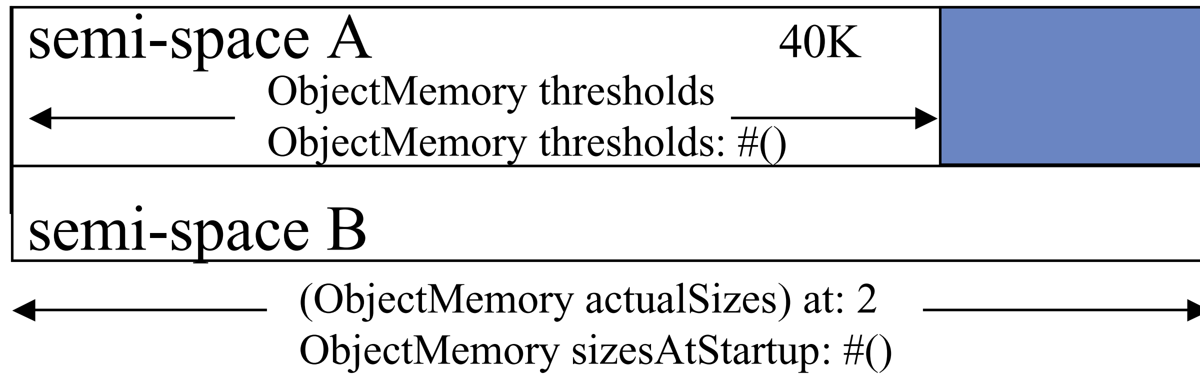
(ObjectMemory actualSizes) at: 1
ObjectMemory sizesAtStartup: #()

actualSizes	Returns 7 element array of byte sizes. First element is Eden’s current size in bytes (204800).
sizesAtStartup:	Needs 7 element array of multipliers. A multiplier 1.5 means allocate 1.5 times default size at startup. Eden is first slot value.
thresholds	Returns 3 element array of percentages. First element is Eden’s threshold when to start GC (96%).
thresholds:	Needs 3 element array of percentages. A value of 0.75 means start GC work at 75% full.

Is Eden sized OK? Perhaps...

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved

VW Survivor Spaces

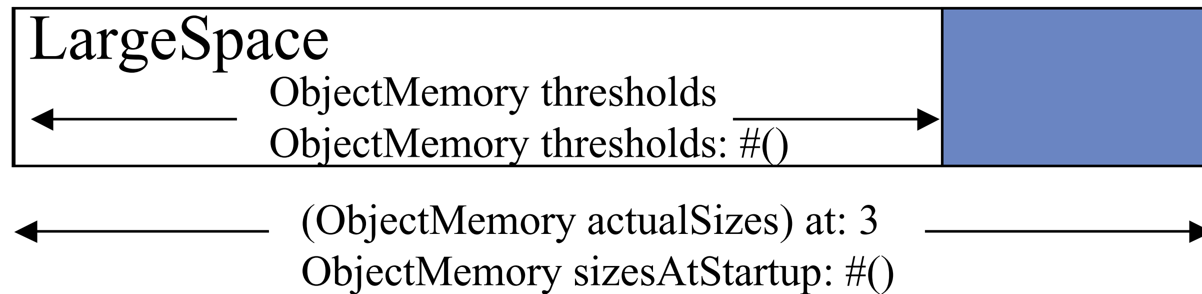


actualSizes	Second element is Survivor space's current size in bytes (40960) x 2.
sizesAtStartup:	Survivor space is second slot.
thresholds	Second element is Survivor space threshold of when to start tenure to OldSpace (62.5%). (Careful)
thresholds:	Set to desired values

Make bigger. Watch threshold!

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW LargeSpace - 1025 Magic



actualSizes	Third element is LargeSpace current size in bytes (204800).
sizesAtStartup:	Large space is third slot.
thresholds	Third element is LargeSpace threshold. Start tenure to OldSpace at (90.2344%).
thresholds:	Set to desired values

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

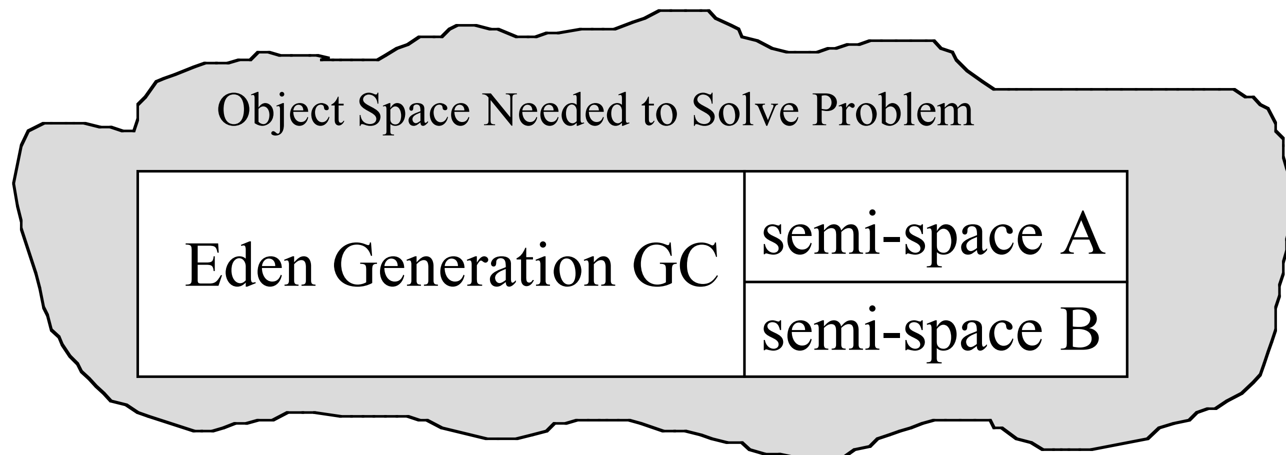
Make bigger!

Old, Fixed, Compiled (Size?)

* FixedSpace, OldSpace, and CompiledCodeCache could be made bigger. But application behavior will drive values for FixedSpace and OldSpace.
* CompiledCodeCache, try different values (small multiplier factors). Note PowerPC default value is too small. Might buy 1%
* StackSpace shouldn't need altering, depends on application, just check it to confirm.

Generational GC Issues:

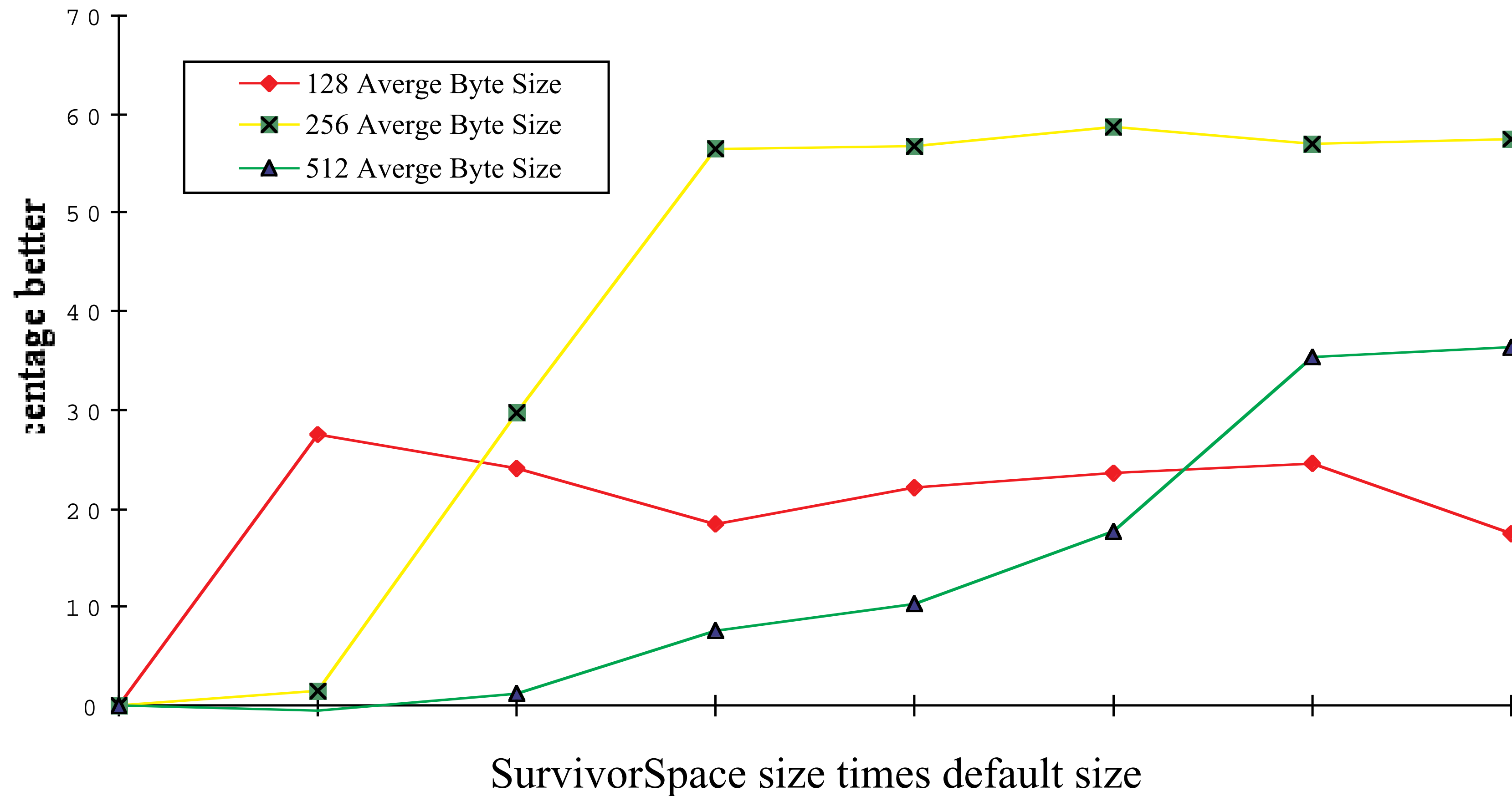
* Early Tenuring
  - Objects get tenured to OldSpace too early then they promptly die clogging OldSpace with corpses. See my article in Dec. 1996 Smalltalk Report (remember them?) (Also on my Web Site).
  - Issue: Problem Domain working set size exceeds NewSpace size.



¥ Solution - Make semi-spaces bigger.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Allocation Rate % better versus SuvivorSpace multiplier factor



55% better, because of less GC work

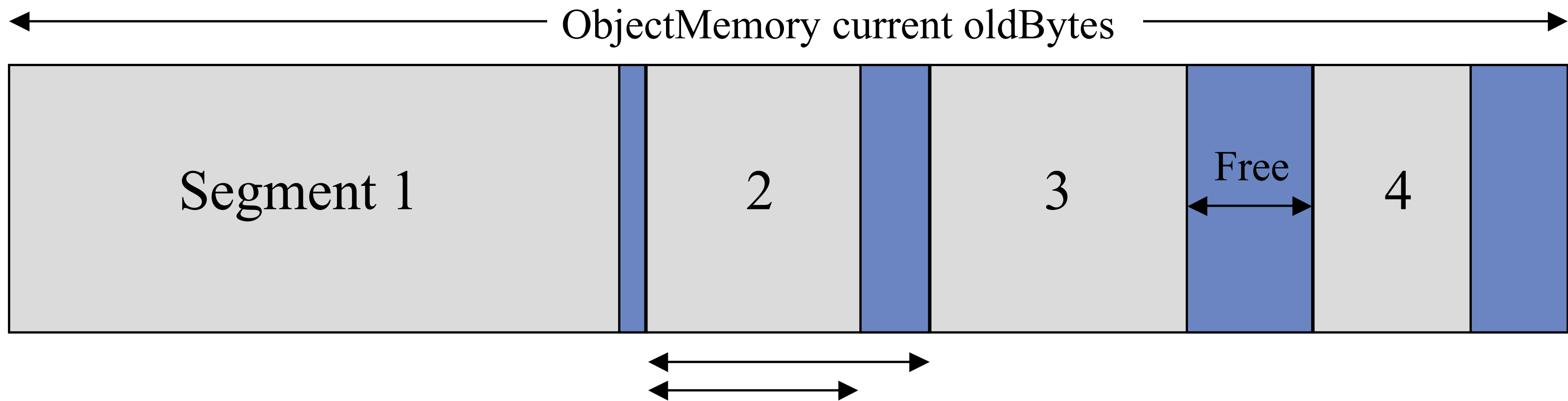
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

OldSpace Growth or GC?

* As Objects flow into OldSpace, what should it do?
  - Perform Incremental Mark/Sweep faster?
  - Stop and do a full GC?
  - Expand?
* Expansion is the easiest choice!
  - Much cheaper than GC work (VM viewpoint).
  - Paging is expensive (O/S viewpoint). Perhaps rare now
  - VisualWorks & VisualAge choice with modifications.
  - Squeak 3.x choice too.
  - But rules may allow/disallow

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW OldSpace - A Place to Die

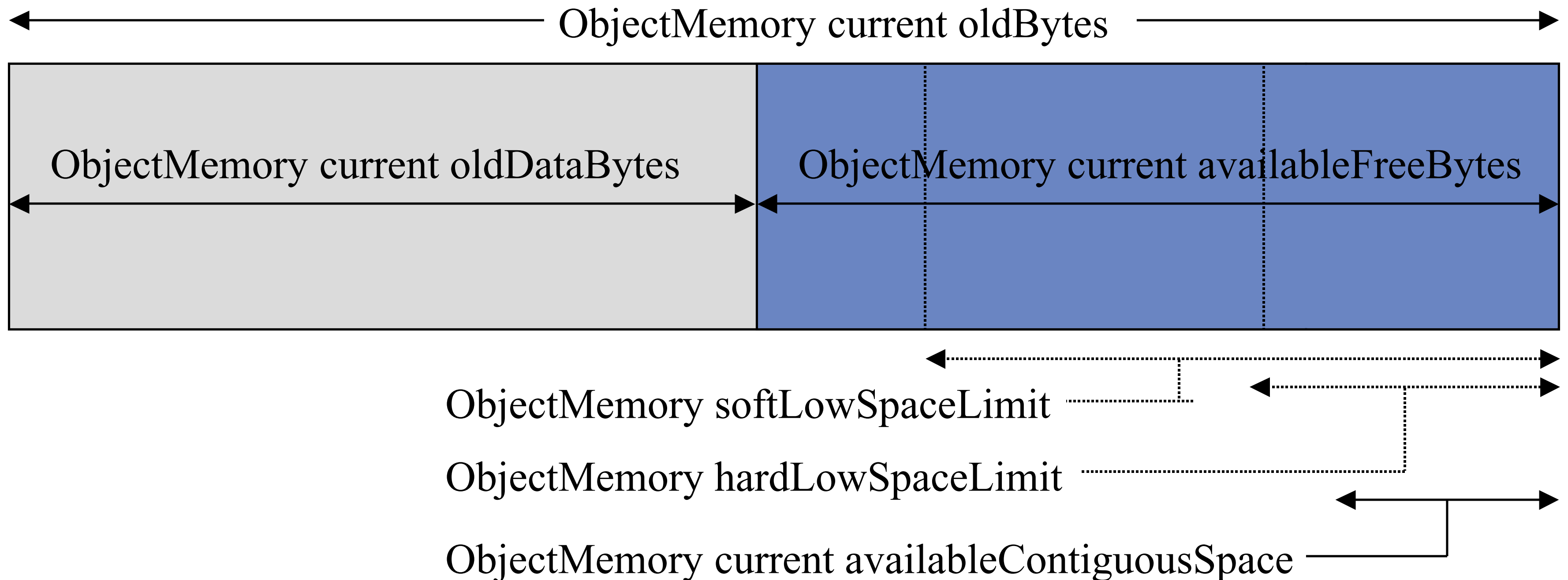


- Segments are mapped to indexed slots of instance of ObjectMemory.
- Allocated in blocks. Size set by needs, or by increment value.
  - Allocating a very large object will cause allocation of large block.
- Note new concept in VW 2.5.2: Shrink footprint.
  - ObjectMemory(class)>>shrinkMemoryBy:

Note FixedSpace is similar

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW OldSpace - Thresholds



* FreeSpace (total and contiguous) versus hard and soft low space limits affect behavior of IGC. Where should logic be placed?. . . (note some other structures lurk here)

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW OldSpace - OTEntries & OTData

* OldSpace segment has Object Table, Object Table Entries for Data bodies, and somewhere lurks the remember table (RT).
* Most of these tables grow/shrink based on dynamics/need. But only the Object bodies get compacted.
* Ability to move objects between segments means you can vacate, and free a block, thus shrinking the memory footprint of VW.
* Note PermSpace is similar. But only GCed on explicit request

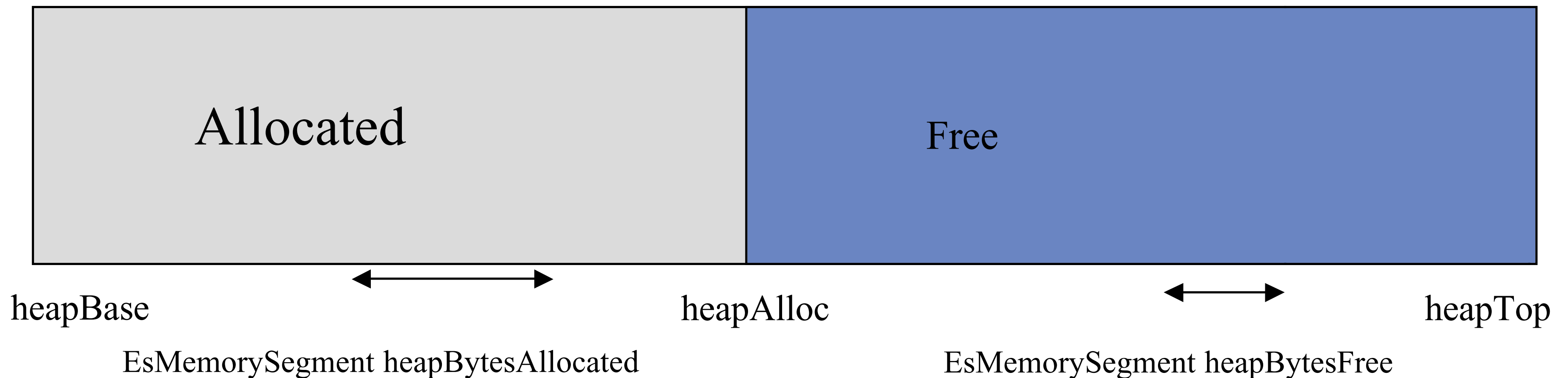
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

IBM VisualAge Memory Layout v5.5.1

NewSpace	Segments (lots!)		
semi-space - A 262,144	OldSpace (RAM/ROM)	Fixed Old Space	Code Cache
semi-space - B 252,144			

* Generational Copy Garbage Collector & Mark/Sweep.
    * Copy between semi-spaces until full
    * Then tenure some objects to current OldSpace segment
* Object loader/dumper can allocate segments (supports ROM)
* EsMemorySegment activeNewSpace
    *To see current NewSpace semi-spaces size. Default is 262,144 in size
    *abt -imyimage.im -mn###### (Start with ### byte in NewSpace)

VisualAge Segment



* Segments are mapped to instances of EsMemorySegment.
* OldSpace allocator seems simple, just allocate via pointer move
* Newer versions of VA will release memory based on -mlxxxx value

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualAge - Segments

EsMemorySegment activeNewSpace (scavenge target)

EsMemorySegment(

Type: NEW,RAM, (ID: 16r0)

Size: 262144 bytes [16r1AD966C -> 16r1B1966C] (69.4 % free)

Heap: [16r1AD966C -> 16r1B1966C]

Alloc: 16r1AED76C Scan: 16r1AD9F08

Free List: [16r0 -> 16r0]

Remembered Set: [16r0 -> 16r0])

* 4 of them in my image. But 2 are empty and about 2K in size. The other two are 262,144 bytes in size.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualAge - Segments

EsMemorySegment activeOldSpace (tenure target)

EsMemorySegment(

Type: OLD,RAM, (ID: 16r0)

Size: 2098152 bytes [16r1C30010 -> 16r1E303F8] (8.51 % free)

Heap: [16r1C30010 -> 16r1E2F6AC]

Alloc: 16r1E03EB8 Scan: 16r1E03EB8

Free List: [16r0 -> 16r0]

Remembered Set: [16r1E3039C -> 16r1E303F8])

* My Image has 231 segments range from 28 to 2,790,996 bytes

VisualAge has fine grained segments

EsMemoryTypeOld = Objects in the space are old.
EsMemoryTypeNew = Objects in the space are new.
EsMemoryTypeROM = Memory segment considered Read-only.
EsMemoryTypeRAM = Memory segment is Read-Write.
EsMemoryTypeFixed = Objects in this memory segment do not move.
EsMemoryTypeUser = User defined segment.
EsMemoryTypeCode = Segment contains translated code.
EsMemoryTypeExports = Segment containing IC export information.
EsMemoryTypeDynamicInfo = Contains Dynamic image information.
EsMemoryTypeAllocated= Was allocated by VM

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW MemoryPolicy-Logic

* Delegation of Responsibility for:
  - idleLoopAction
    * A low priority process that runs the Incremental Garbage Collector (IGC) based on heuristics after a scavenge event.
  - lowSpaceAction
    * A high priority process that runs when triggered by the VM if free space drops below the soft or hard thresholds. The soft threshold invokes the IGC to run in phases, and possibly does a compaction. The hard threshold triggers a full IGC and perhaps a full compacting GC if the situation is critical...

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualWorks Growth - Controls

MemoryPolicy>> measureDynamicFootprint ForGrowthRegime MemoryPolicy>> measureDynamicFootprint ForMemoryUpperBound	These two methods set up two blocks to: (1) Measure and control growth before aggressive IGC work. (2) Measure and control total growth of image. Default logic measures dynamic footprint.Alternate logic measures growth since VM start. Growth will occur until memory exceeds 16,000,000 (**growthRegimeUpperBound**). Maximum growth is CPU limited or (**memoryUpperBound**).
Instance variables idleLoopAllocationThreshold	(16,000,000) IGC Byte threshold- (Scavenges times Eden full threshold) needs to exceed this value -> ~80 scavenges with 200K Eden. When reached, idleLoopAction will trigger a full **interruptable** IGC cycle, collecting the maximum amount of garbage. If free space is fragmented, a compacting GC is done.
maxHardLowSpaceLimit	(250,000) Byte threshold-When reached, the lowSpaceAction process is triggered to do a full **non-interuptable** IGC cycle, and/or a possible compacting GC cycle, and/or grow OldSpace.
lowSpacePercent	(25%)- Ensures hard low space limit is minimum of 25% of free space or current maxHardLowSpaceLimit. As OldSpace is adjusted, the lowSpace limits are altered.
preferredGrowthIncrement	(1,000,000) Bytes-To grow if we grow.
growthRetryDecrement	(10,000)-If the preferred growth increment is too big for hosting O/S, decrement by this value and try again.
incrementalAllocationThreshold	(10,000)- Free space minus this, gives **SoftLowSpaceLimit. (**Trigger for **interruptable phased** IGC work).

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

When Does VW Growth Happen?

* Growth can happen when (but only up to **memoryUpperBound**):
  - (1) Allocation failure for new: or become: This means the new object we want has exceeded the maximum amount of continuous free space we have. Grow and/or garbage collect to meet need.
  - (2) Space is low (**HardLowSpaceLimit**), and growth is allowed. Easy choice-just grow by increment value, if we've not exceeded **growthRegimeUpperBound**. Otherwise, see next step.
  - (3) Space is low, and growth wasn't allowed ,and after we do a full GC, a incremental GC or compacting GC. Then grow by increment value, unless we've reach **memoryUpperBound.**
* Growth refused, then Notifier window.
  - Space warning bytes left: ####
  - Emergency: No Space Left

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW Incremental GC thoughts

* Free space is below soft limit. Run IGC in steps. For each step, ask the IGC to do a certain amount of work, or quota.
* Free space is below hard limit. Run full IGC cycle without interrupts.
* If image is 'idle'and we've scavenged **idleLoopAllocationsThreshold** bytes, then run IGC in microsteps. For each step, run to completion, but stop if interrupted.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Altering VW IGC Idle Behavior

MemoryPolicy Variables: incMarkingRate	(40) Factor used to calculate the number of Objects to mark in one IGC. pass. This is calculated from the minimum of 2,000 or: Objects * incGCAcclerationFactor * percentOfFreeSpaceAllocated incMarkingRate Objects here include ones in free chains.
incUnMarkingRate	(8) Factor used to calculate the number of Objects to unmark if an unmark step is called for. This is calculated from the minimum of 10,000 or: markedObject s* incGCAcclerationFactor *percentOfFreeSpaceAllocated incUnmarkingRate
incNillingRate	(2) Factor used to calculate the number of bytes of weak objects to examine when nilling. This is calculated from the minimum of 50,000 or: markedWeakBytes * incGCAcclerationFactor * percentOfFreeSpaceAllocated incNillingRate
incSweepingRate	(8) Factor used to calculate the number of Object to sweep if an sweep step is called for. This is calculated from the minimum of 10,000 or: Object s* incGCAcclerationFactor *percentOfFreeSpaceAllocated incSweepRate
incGCAcclerationFactor	(5) Modifier for IGC. Increasing value causes IGC to work harder for each step, up to the hard-coded limits.

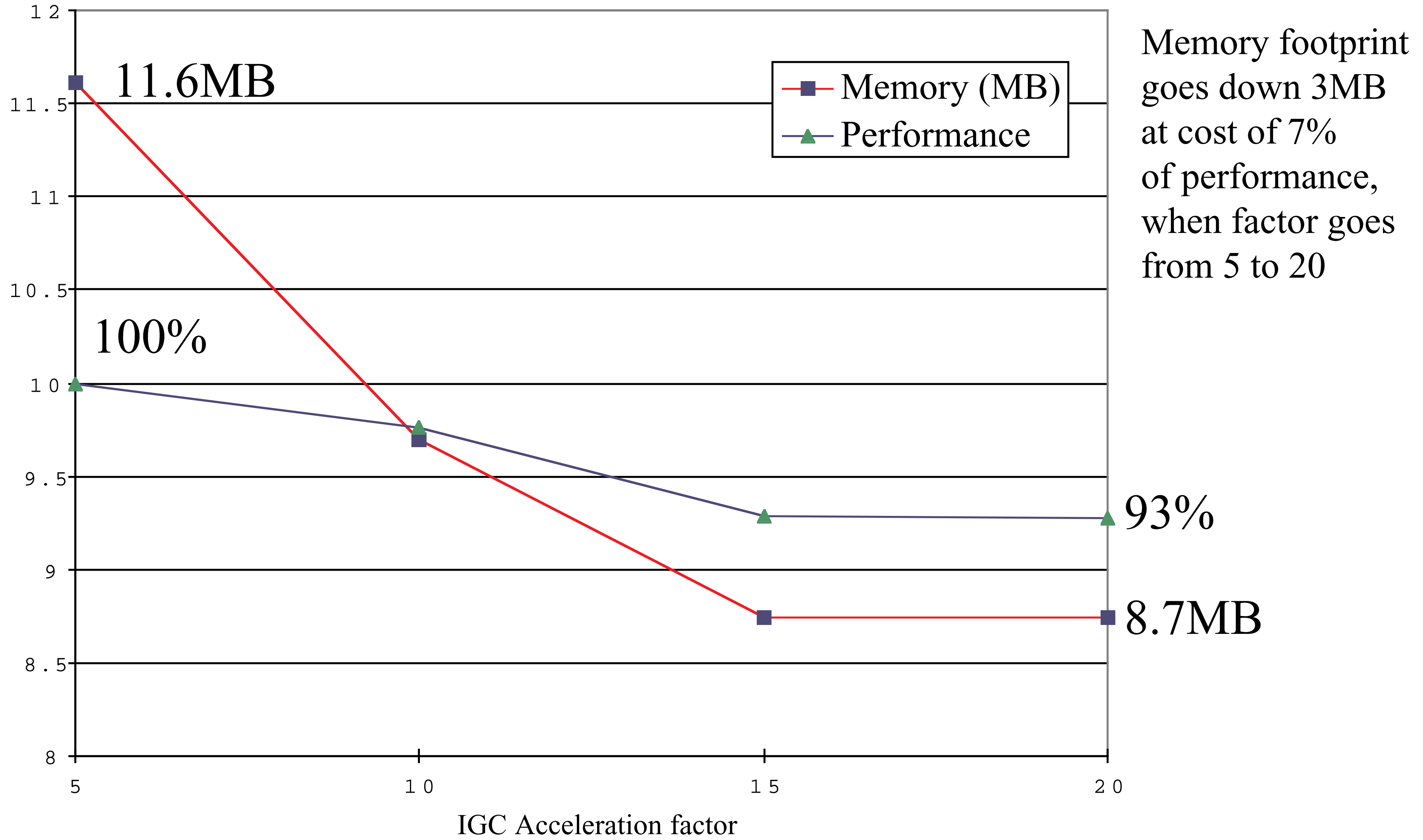
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW incGCAccelerationFactor

* Acceleration? Quotas? Limits?
  - Concept was to limit IGC idle work moderated by percentage of free space, to a given cut-off point. This allows us to increase effort as free memory decreases, but only to a point.
  - Executing a full quota means a work stoppage of N milliseconds. Quotas picked were based on CPUs of 1990. These default quotas are smaller than they could be. So consider increasing IGC work.
    * May slow image growth, but at cost of reduced performance....

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Memory and Performance Versus IGC Acceleration



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved. Page 71

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
*	Smalltalk bytesLeftString	Get bytes free + expansions

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VW - GC commands

ObjectMemory globalGarbageCollect ObjectMemory globalCompactingGC ObjectMemory compactingGC	Mark/Sweep PermSpace and OldSpace. Mark/Sweep all plus compact. Mark/sweep/compact Oldspace, not PermSpace.
ObjectMemory garbageCollect	Mark/Sweep and compact only OldSpace.
ObjectMemory quickGC	Incremental Mark/Sweep of OldSpace, no compaction.
ObjectMemory addToScavengeNotificationList: ObjectMemory removeFromScavengeNotificationList:	Add/Remove object to list of dependents to be notified, when a scavenge event has occurred.
ObjectMemory dynamicallyAllocatedFootprint	Report on VM memory usage.
ObjectMemory current availableFreeBytes	Report of free memory in OldSpace.
ObjectMemory current numScavenges	Report on number of Scavenges.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Many other queries exist

VisualWorks Thoughts

* Compaction trigger logic has holes.
  - Some fragmentation counters aren't considered.
  - Remember Table size isn't considered.
* NewSpace sizes is usually too small.
* Must look at growth limit and max limit. (settings)
* OldSpace growth increment is too small.
* Server applications can stress allocator/GC logic and cause VM failure. (Sad but true)

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualAge - Control

* Control is done via command line arguments:

-mo####	Initial OldSpace size. Set to image size (allow for initial growth).
-mn####	Size of NewSpace semispace... 256K Set higher?
-mf####	Size of FixedSpace. Change depending on application needs.
-mi####	OldSpace increment size. Defaults to 2048K.
-ml####	Minimum free memory threshold, 500K. Possible change
-mx####	Maximum growth limit defaults to unlimited. Pick a limit? -mx1 disables
-mc####	Code Cache size. Defaults to 2 million. -mcd to disable. Change size?

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualAge - GC commands

System globalGarbageCollect	Trigger a Mark/Sweep.
System scavenge	Trigger a NewSpace scavenge.
Process addGCEventHandler: Process removeGCEventHandler:	Add/Remove a handler from the queue of handlers that get notified when a scavenge or global GC occurs.
System totalAllocatedMemory	Amount of memory allocated by VM.
System availableMemory System availableFixedSpaceMemory System availableNewSpaceMemory System availableOldSpaceMemory	Amount of memory, various viewpoints.
EsbWorkshop new open	stat tool. Gives scavenger and global GC timings for code fragments, also other info.
[ ] reportAllocation:	Report of allocated classes for executed block.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

VisualAge Thoughts

* Increase NewSpace to avoid early tenuring.
* Increase OldSpace headroom.
  - Defer first Mark/Sweep
* Increase OldSpace Increment to reduce GC work.
  - Many grow requests add excessive overhead.
* Review Code Cache Size.
  - Trade performance for memory
* Watch freedom of expansion, or shrinkage.
  - Paging happens when?, watch -mlxxxx value

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Must Remember

* Always avoid paging.
* Tuning might solve growth issue and paging. . .
* Algorithms , algorithms, algorithms.
  - Solve memory problems in the code, not in the GC.
  - Don’t make garbage.
* Lots of time could be used by the GC
  - unless you look you don’t know

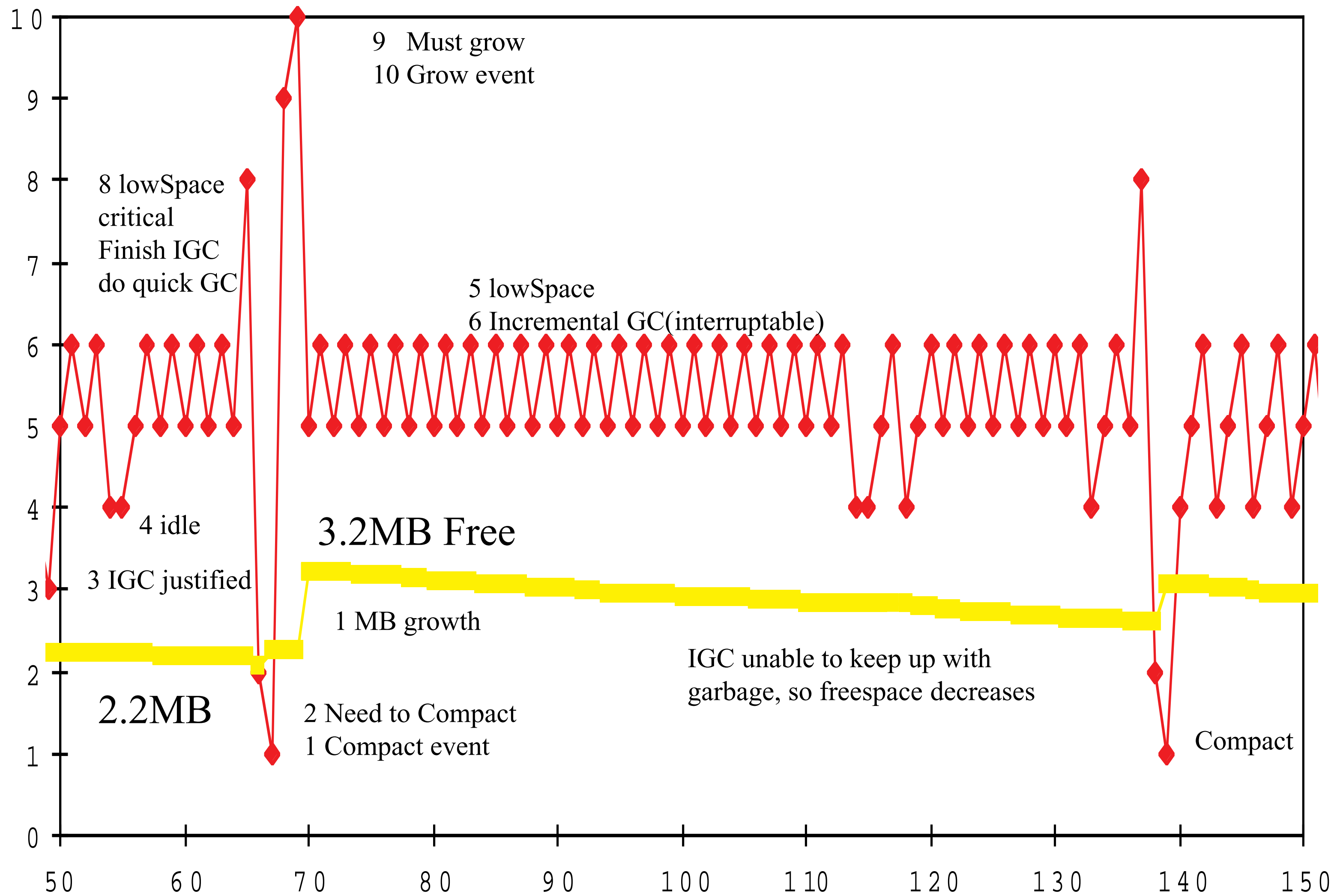
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Real world examples of tuning

Key to Events

1	Compact GC event: Full mark/sweep/compact OldSpace
2	Compacting decision has been made
3	IGC justified, interruptible, full cycle via idle loop call
4	Idle Loop Entered
5	Low Space Action Entered via VM trigger
6	*Incremental GC, (work quotas) attempt to cleanup OldSpace*
7	Request grow; Grow if allowed
8	LowSpace and we must grow, but first do aggressive GC work: Finish IGC, do OldSpace Mark/Sweep GC, if required followup with OldSpace Mark/Sweep/Compact
9	Grow Memory required
10	Grow Memory attempted, may fail, but usually is granted

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.



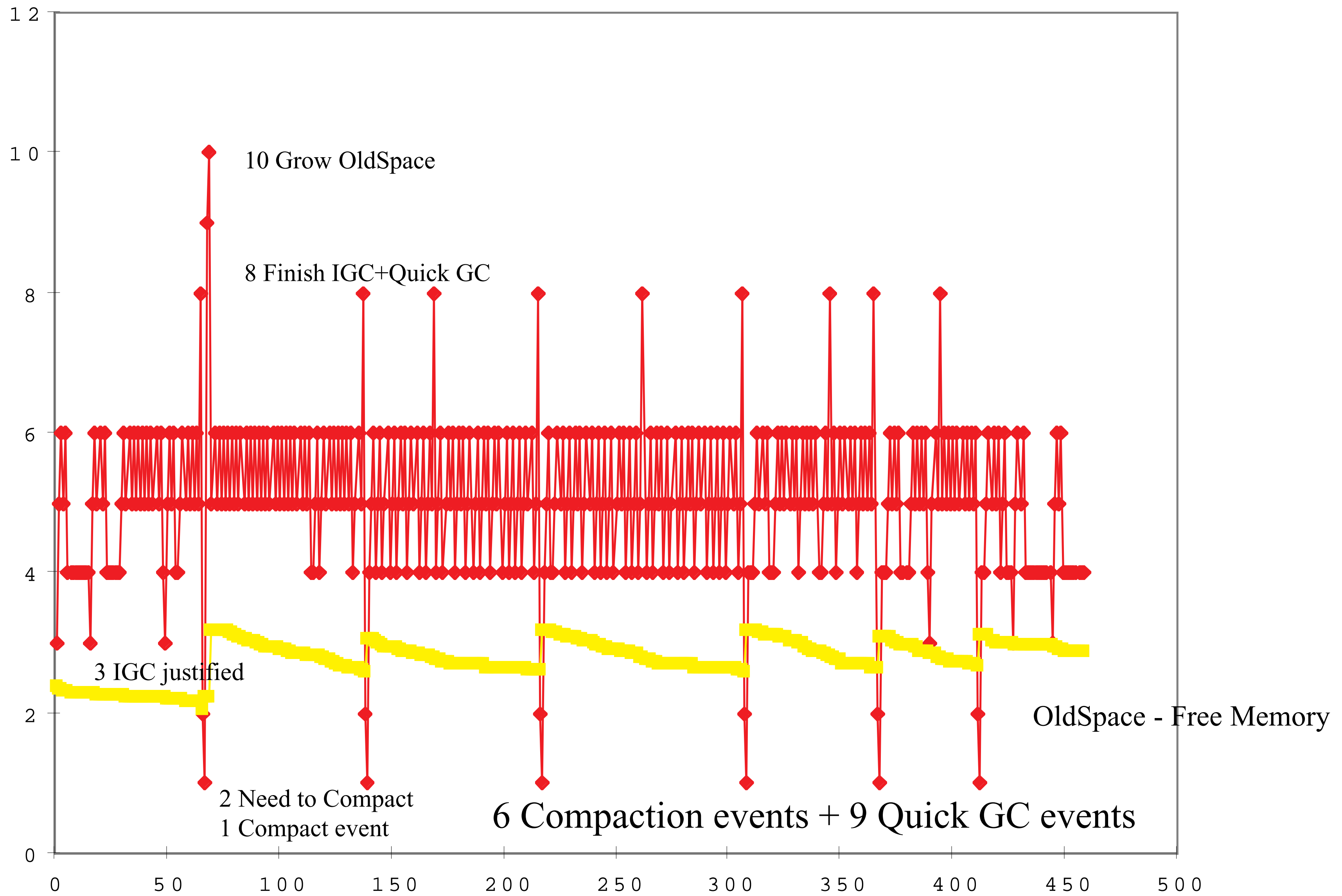
Notes: No Idle Loop GC work (3)

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

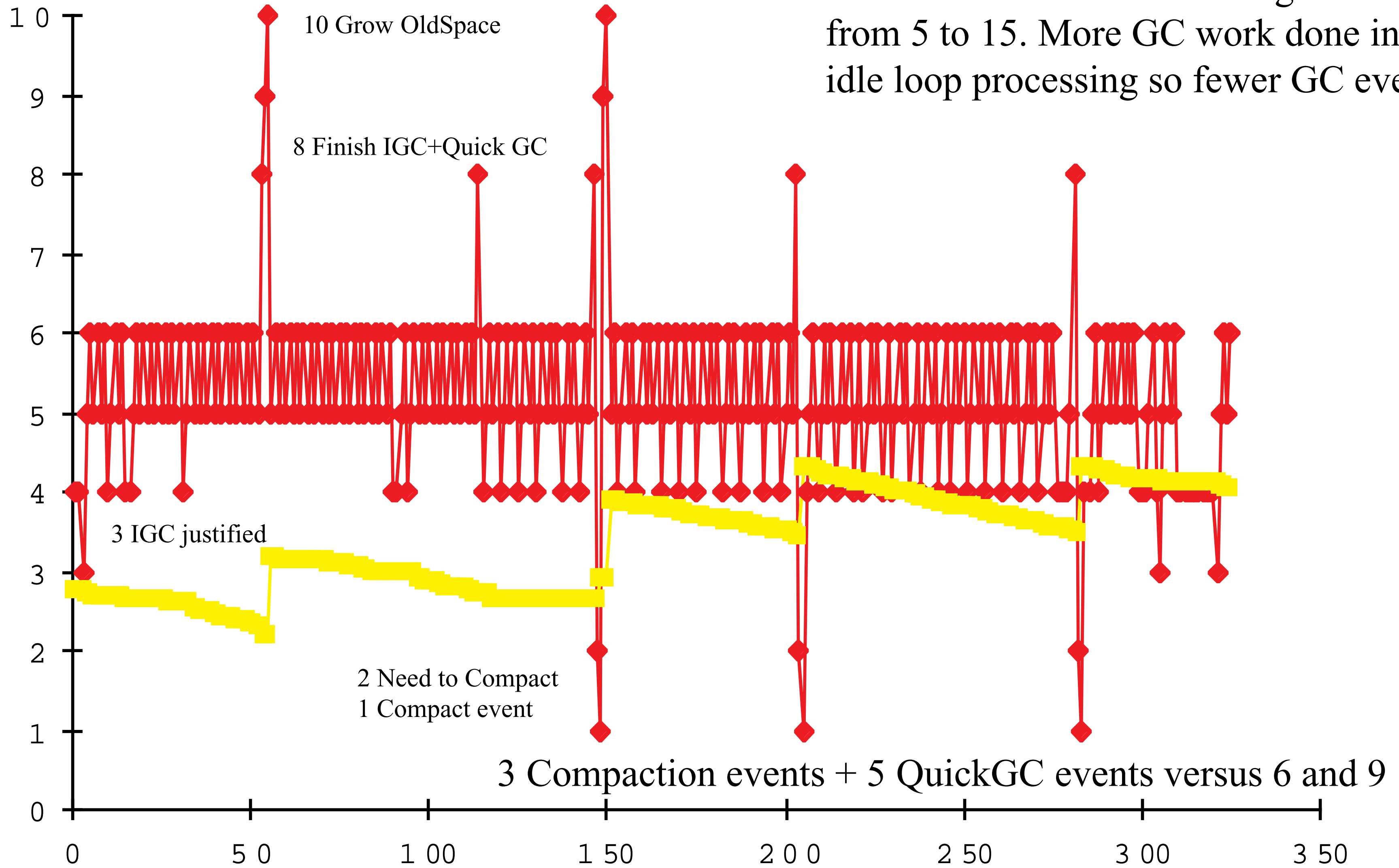
Test Case background

* Ad Hoc Viewer
  - Query returns 272 rows
  - Trace on scan lines, (7518 scan lines)
  - Scroll graphic from left to right, 33 screen updates, wait 1 second between scroll events
    * Automated test tool made test case repeatable. Delay was added to give the idle loop process the ability to run.
    * Think about SUnits to test expectations of response time.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

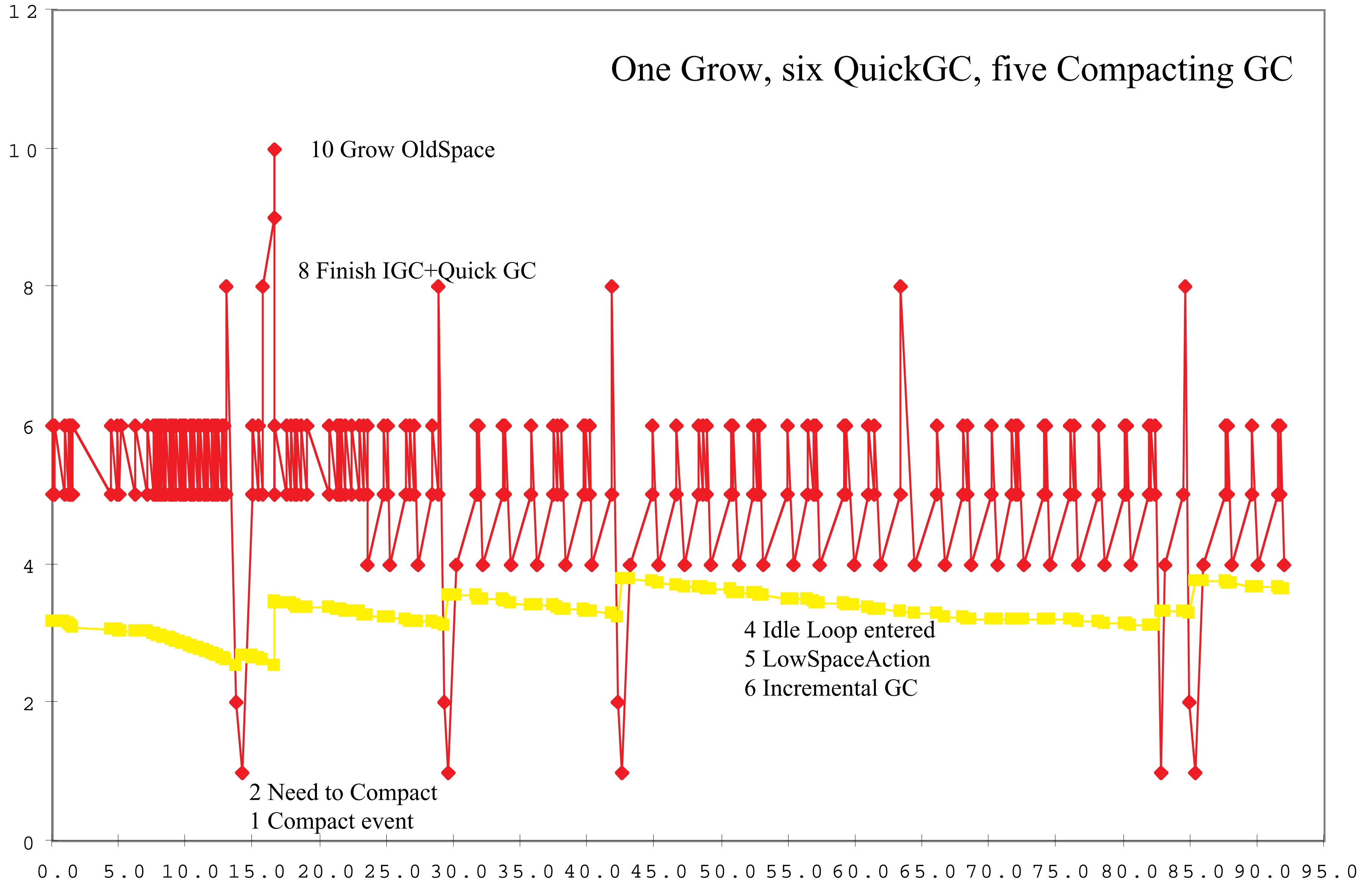


' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

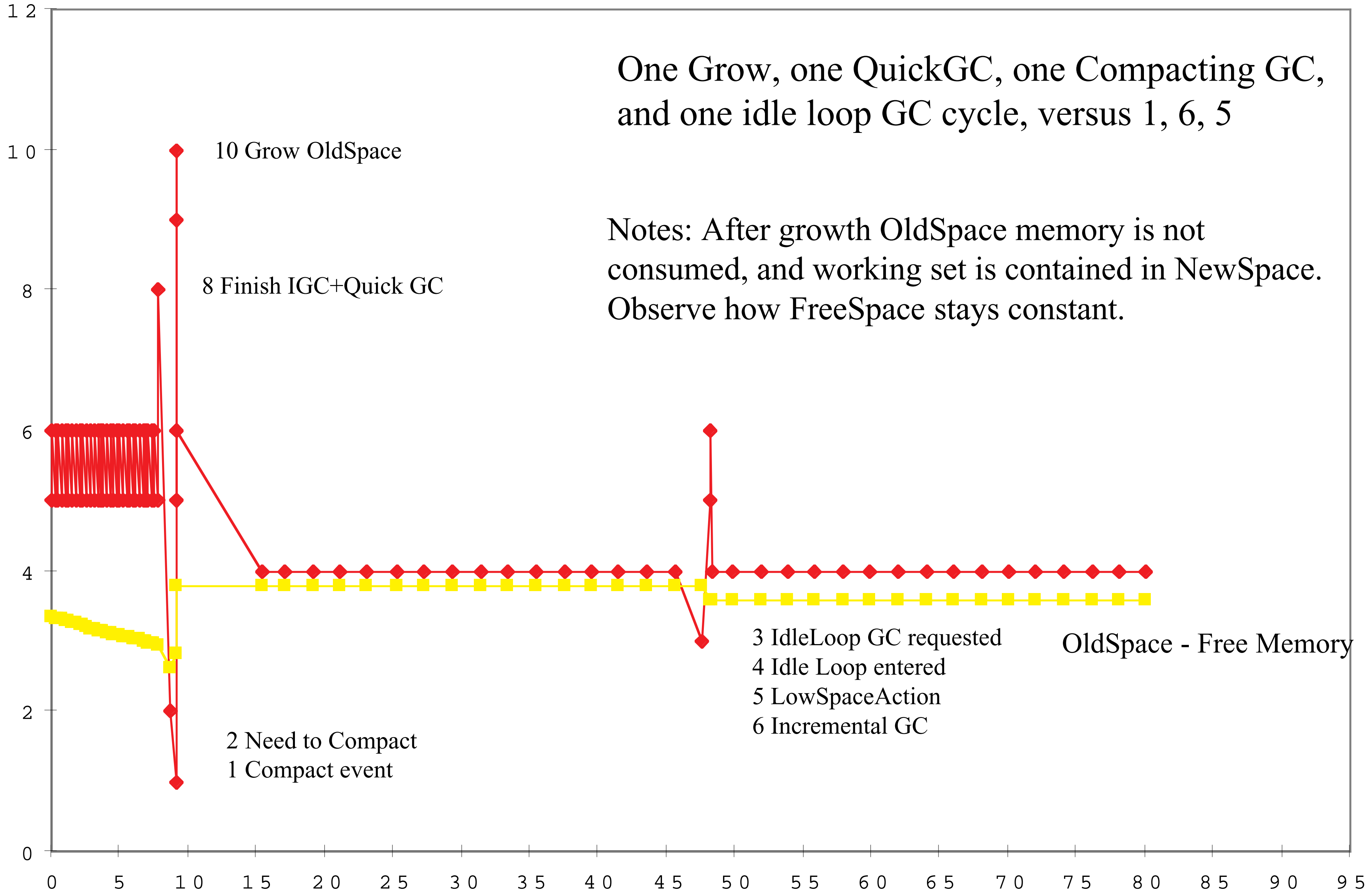


' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Old Case was ~460 seconds, we save 130 secs



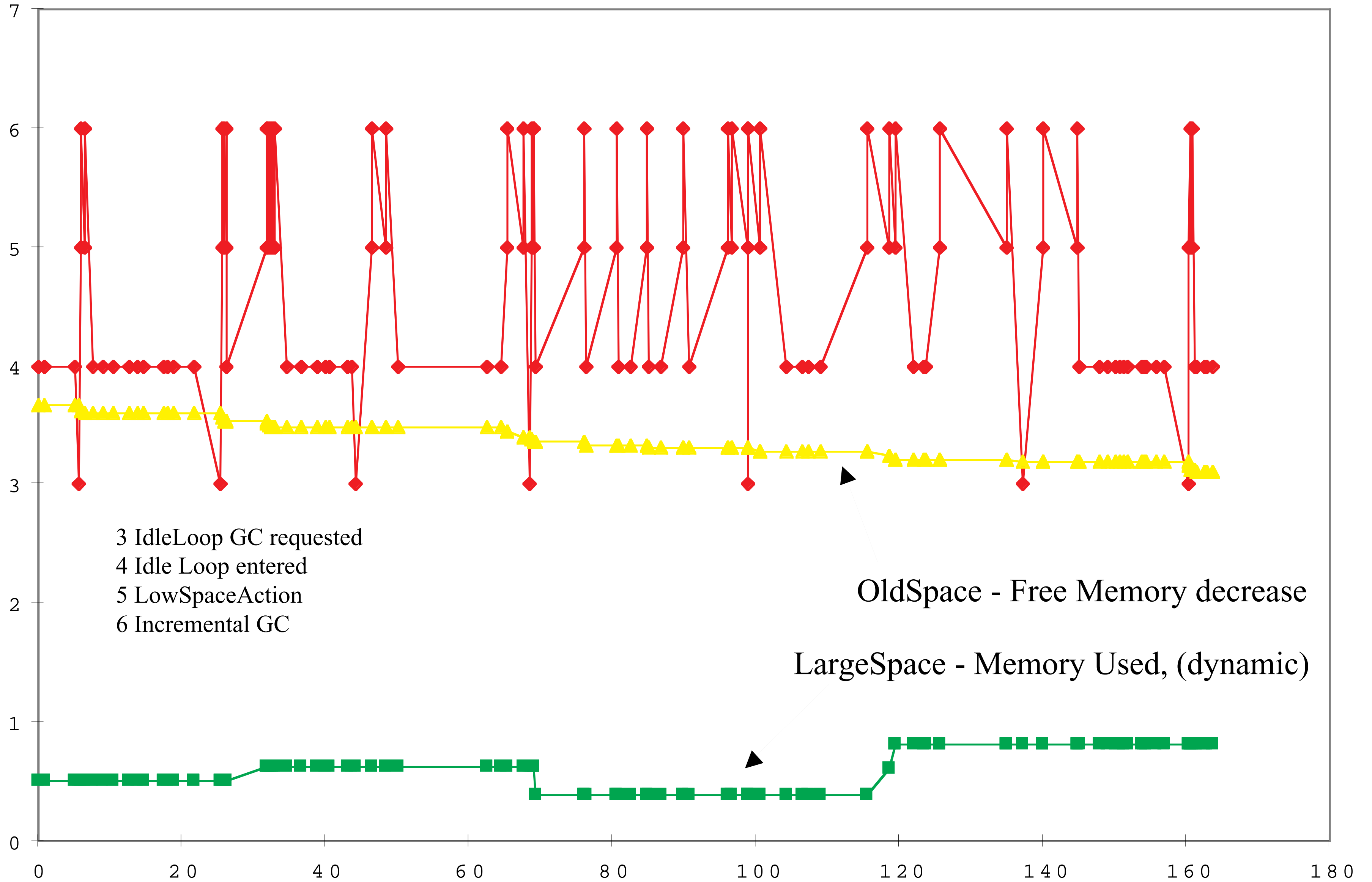
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.



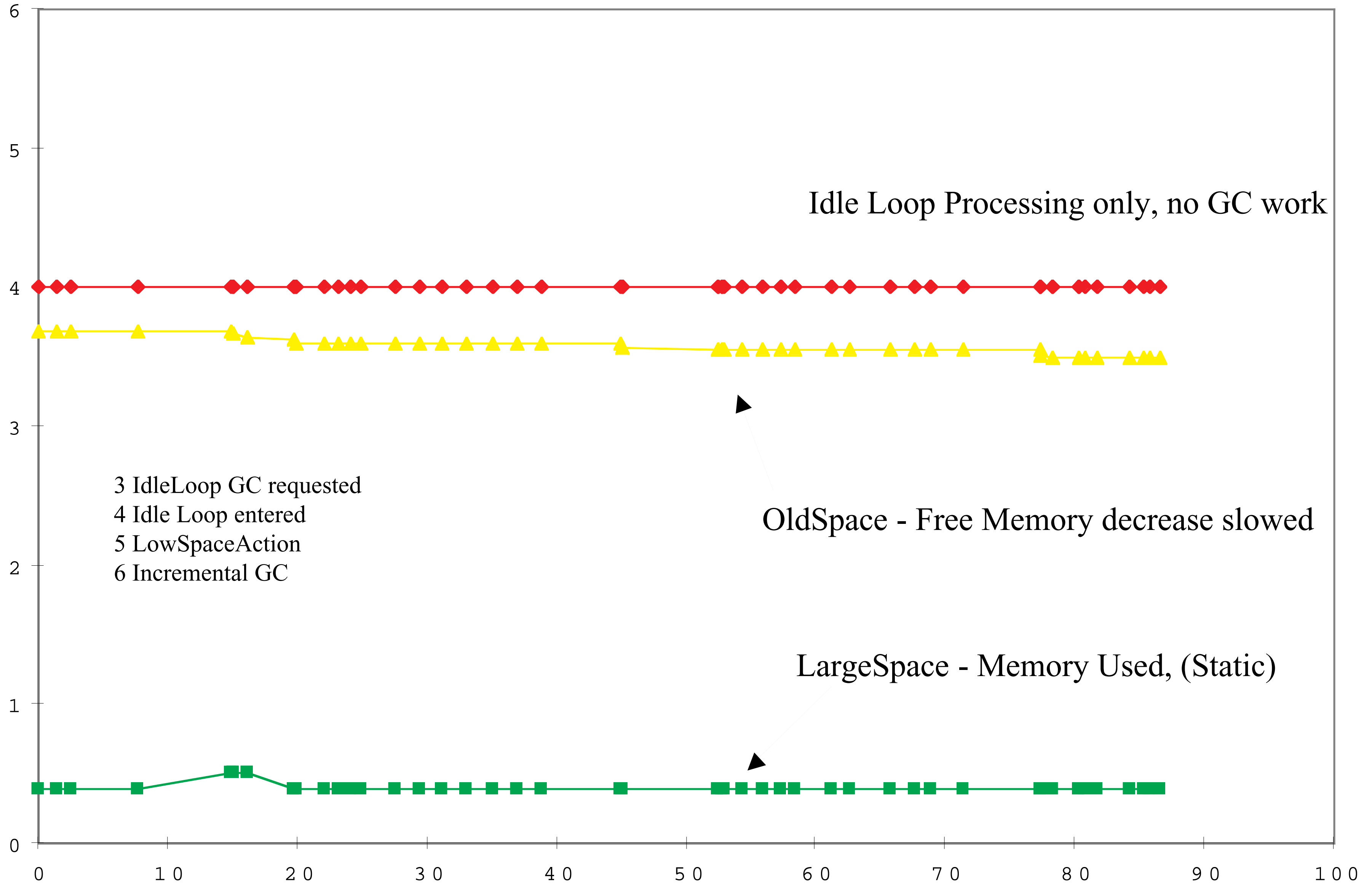
' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Seconds: 81 11 seconds better

Case 3: User interface interaction before changes



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.



' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.

Old case was ~164 seconds
We save ~70 seconds

Parting Thoughts

* Go slow, changing thresholds by 100x is bad. . .
* Server applications work best for tuning, but also have unique problems. (beware lack of idle work).
* Build SUnits for test cases to enable repeatability
* Vendors do supply tools (ie. VisualAge).
* Go look.
  You might be surprised where the time goes.

' copyright 1997,1998,1999,2000,2001 John M McIntosh, all rights reserved.