Control Theory for Heap Sizing

Jeremy Singer, David White
Richard Jones, David Matthews

University of Glasgow
University of Kent

A user bug report ...

- When I build a large session in Isabelle 2011-1 (to be precise, an extension of of linjaT in the Archive of Formal Proofs) w 1, this takes **1:51h**. While run vml requires **12GB** of memor /PID/ status). The final call to onData before writing the hea umes 17GB of memory.

Poly/ML mailing list, Nov 2011

- I now ran the sar on with the SVN version 1352 of Pol .35h, **16GB** for the session common data. What is happe er people experienced a similar s usage and runtime?

paging

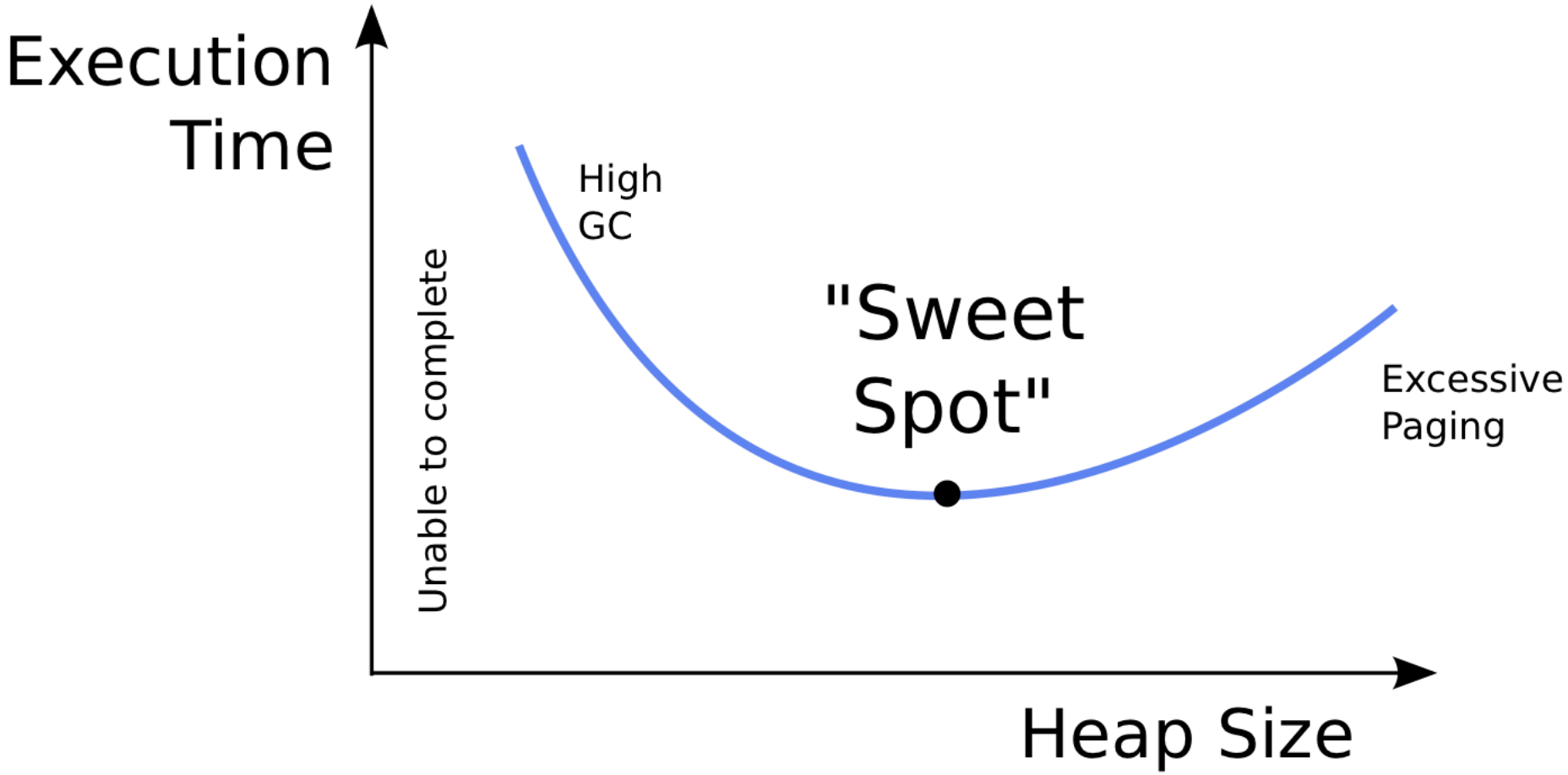
Another frustrated user …

- Every time I try to do ated in eclipse, like fire y to do a method name a e whole IDE hangs for sever no apparent reason.

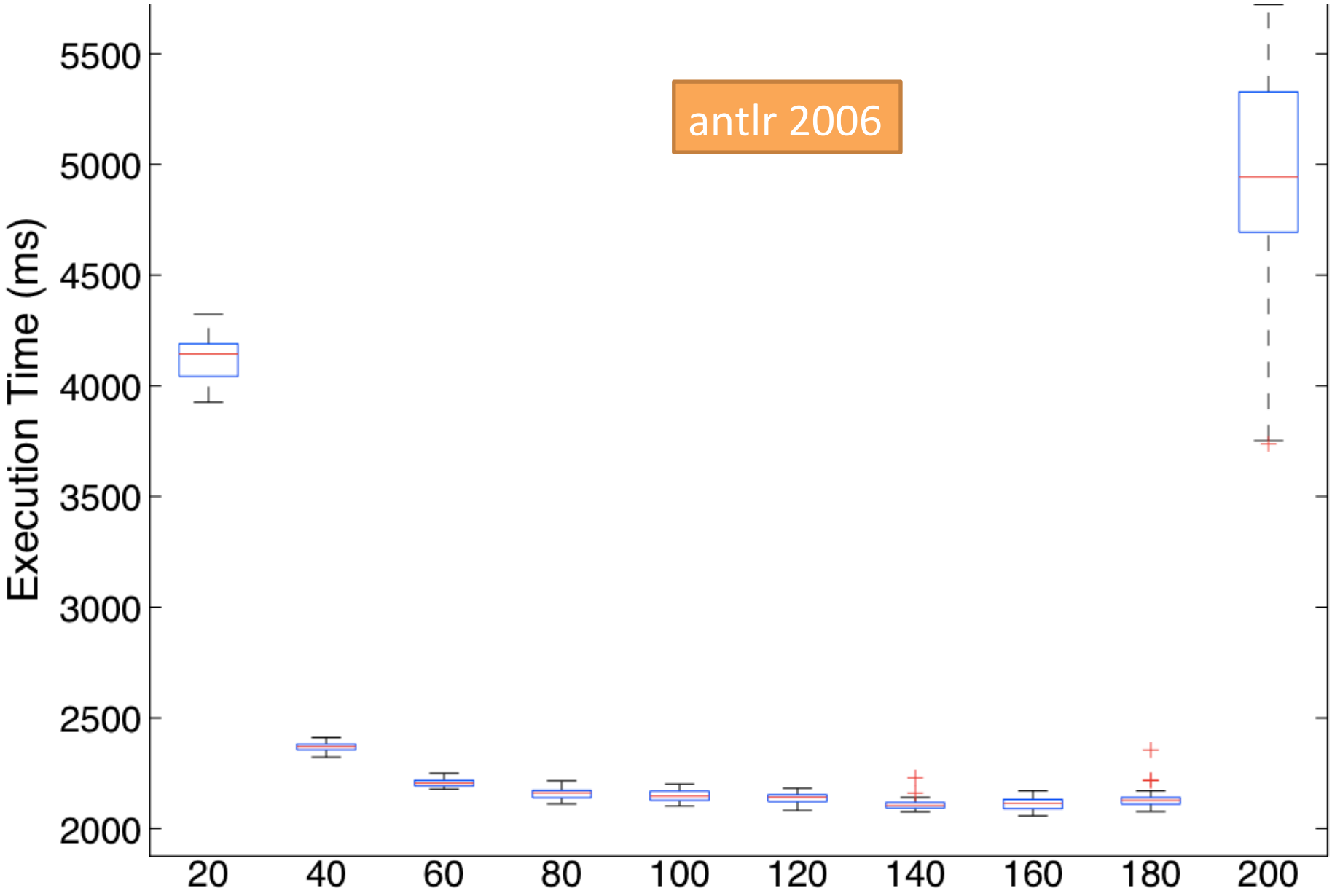
Glasgow CS students, Feb 2012

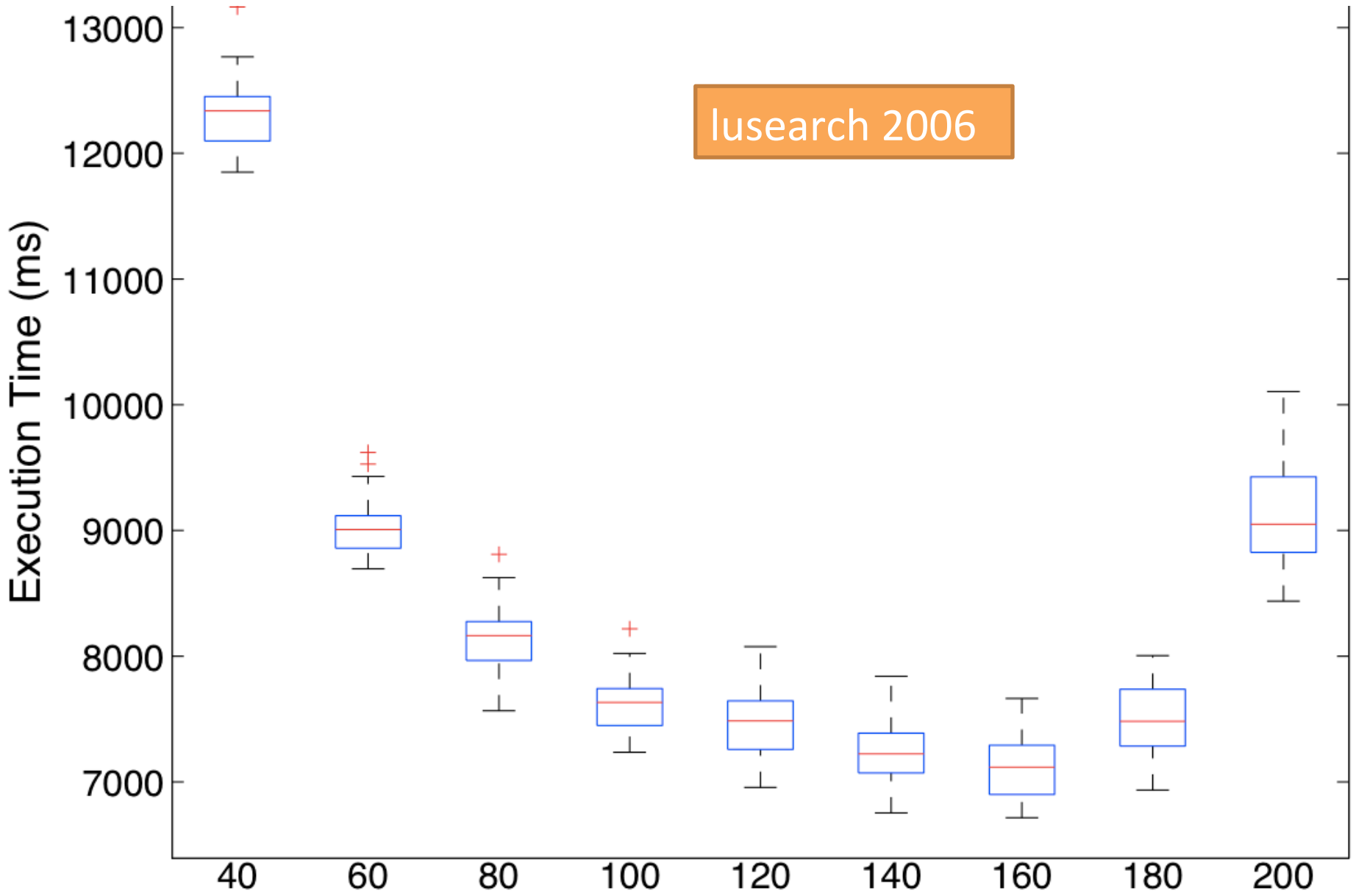
GC

What’s happening?

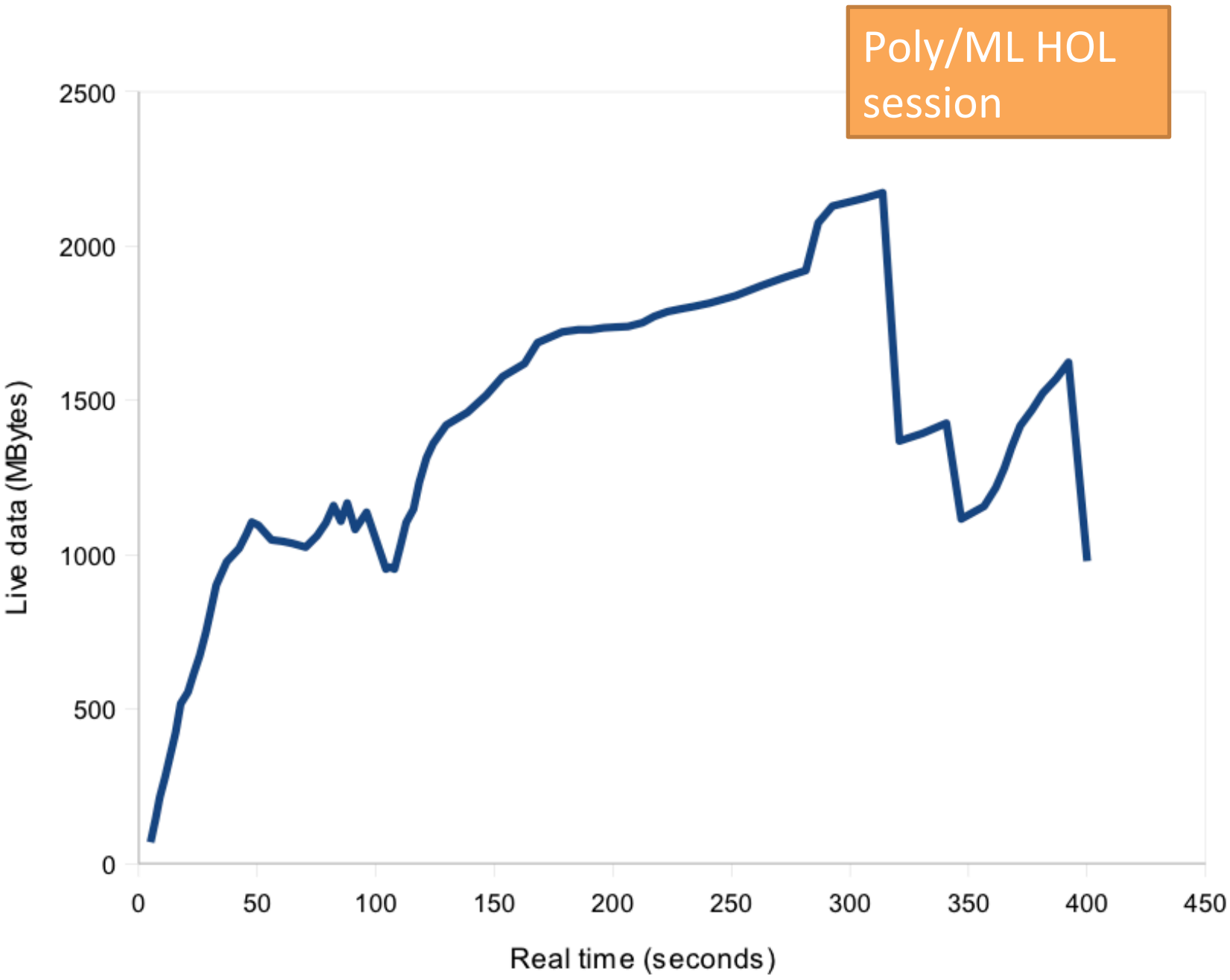

Execution Time
Unable to complete
High GC
"Sweet Spot"
Excessive Paging
Heap Size

Does this happen in real systems?


antlr 2006
Execution Time (ms)
5500
5000
4500
4000
3500
3000
2500
2000
20
40
60
80
100
120
140
160
180
200


lusearch 2006
Execution Time (ms)
13000
12000
11000
10000
9000
8000
7000
40
60
80
100
120
140
160
180
200

Is optimal heap size a static or dynamic property?


Poly/ML HOL session
Live data (MBytes)
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
350
400
450
Real time (seconds)

How do existing VMs handle heap growth?

Poly/ML

- **`adjustHeapSize()`** called after major GC
- if l is live data, heap size changed to $K+l$
- K is constant determined by initial parameters

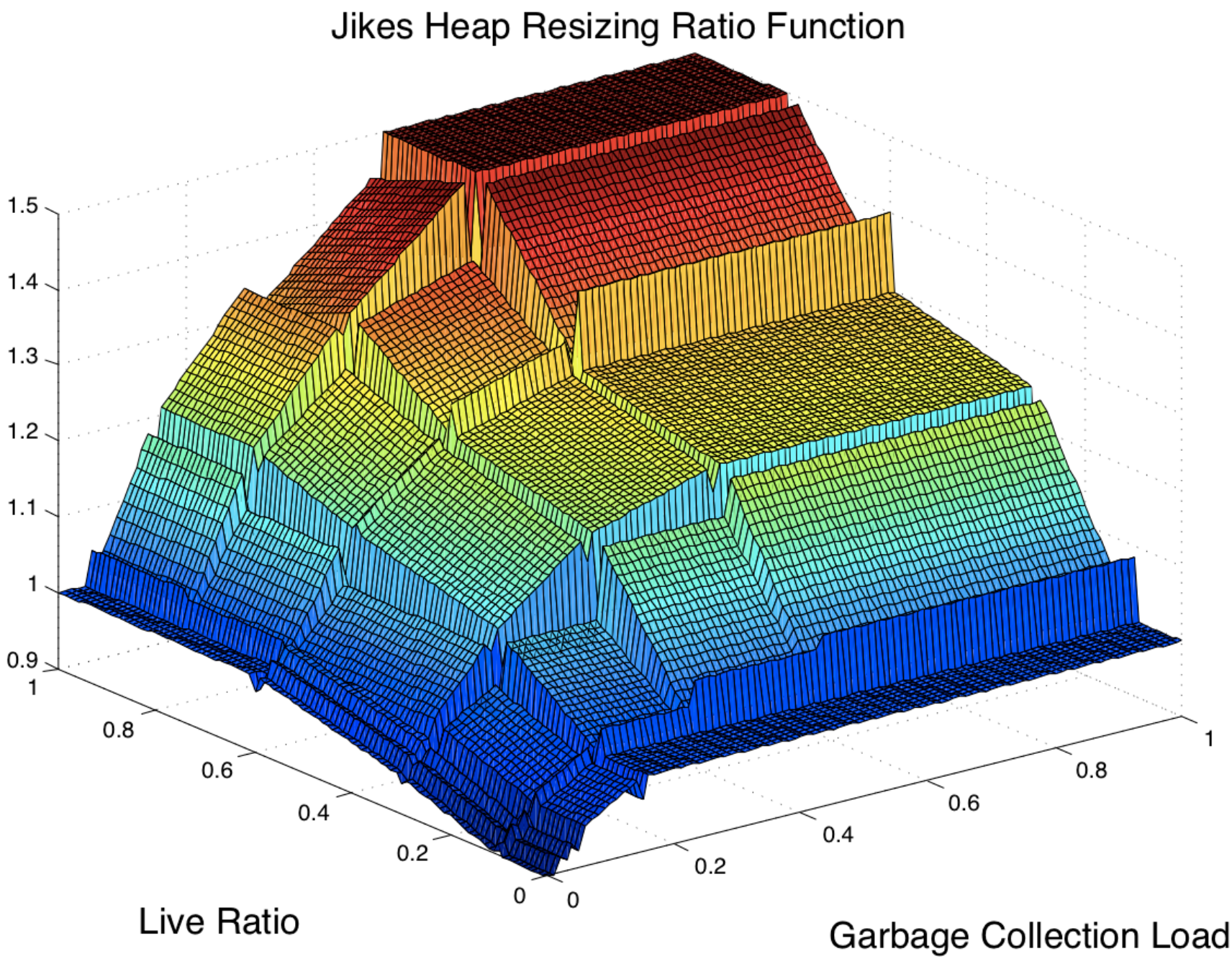
- source code comment: ‘somewhat naïve’

Jikes RVM

- **`HeapGrowthManager`** computes heap resize ratio after major GC
- lookup table of resize ratios, indexed by:

$$g = \frac{\text{Time taken for most recent collection}}{\text{Time since last collection}}$$

$$l = \frac{\text{Amount of live data on the heap}}{\text{Current heap size}}$$


Jikes Heap Resizing Ratio Function
1.5
1.4
1.3
1.2
1.1
1
0.9
1
0.8
0.6
0.4
0.2
0
Live Ratio
0
0.2
0.4
0.6
0.8
1
Garbage Collection Load

Bug fix

Dashboards | Projects | Issues

RVM / RVM-943

MMTk HeapGrowthManager heap growth ratio computation has discontinuities

Log In

Details

Type:	Bug	Status:	Resolved
Priority:	Minor	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	3.1.2
Component/s:	MMTk		
Labels:	None		
Environment:	affects all.		

- In HeapGrowthManager.computeHeapChangeRatio(), the current implementation determines the heap size change ratio by a lookup in the 2-dimensional function table (indexed by liveRatio and gcLoad). Given a current liveRatio X and gcLoad Y, the code finds the table rows and columns with nearest values above and below X and Y, then does interpolation from these table lookup values to determine the heap size change ratio.
- However, there is a bug in the interpolation. If X (or Y, respectively) is exactly equal to a row (or column, respectively) label value, then the interpolation still happens, between values in rows (cols) either side of row X (col Y). This leads to discontinuities in the heap sizing function - see attached graphs.
- The submitted patch suppresses interpolation (interpolation correction value becomes 0) in the case where X or Y falls on a label value exactly, so avoiding the discontinuity.

OpenJDK

- GC ergonomics system allows user to specify high-level goals for GC
  - desired max GC pause time
  - desired application throughput
  - minimum heap size

OpenJDK

- **`AdaptiveSizePolicy`** applies fixed rules to satisfy these targets:
  - if current pause time > pause time goal, then *decrease* heap size
  - else if application's throughput goal is not being met, then *increase* heap size
  - else *decrease* heap size to reduce memory footprint

OpenJDK

- David Vengerov [ISMM11]
- [The ergonomics system consists of] some heuristic rules that do not guarantee that the GC throughput will actually be maximised as a result

Dalvik

- Easy to grow heap, more difficult to shrink
  - non-moving objects in mature space

$$h' = \frac{\text{current size of live data}}{\text{target utilization rate}}$$

- Heap sizing policy entirely opaque to user

Dalvik heap resizing app

Find the best Android apps | Hot today | Hot this week | All-time popular | Top rated

All apps » Tools » VM Heap Tool (root only)

VM Heap Tool (root only)

by martino

Install

Free

50,000-250,000 downloads, **365** ratings (4.50 average), 51 kb, Permissions, Official Page, Contact

Add to list ▼ | Like | +1 0 | Tweet 2 | Email | QR / more ▼

Requires *FULL ROOT* (aka S-OFF or NAND unlock) and *BUSYBOX*. Please use Busybox version 1.17.x as 1.18.x compile is broken!

Allows to change the Dalvik VM heap

Heuristics have problems

Problems with heuristics

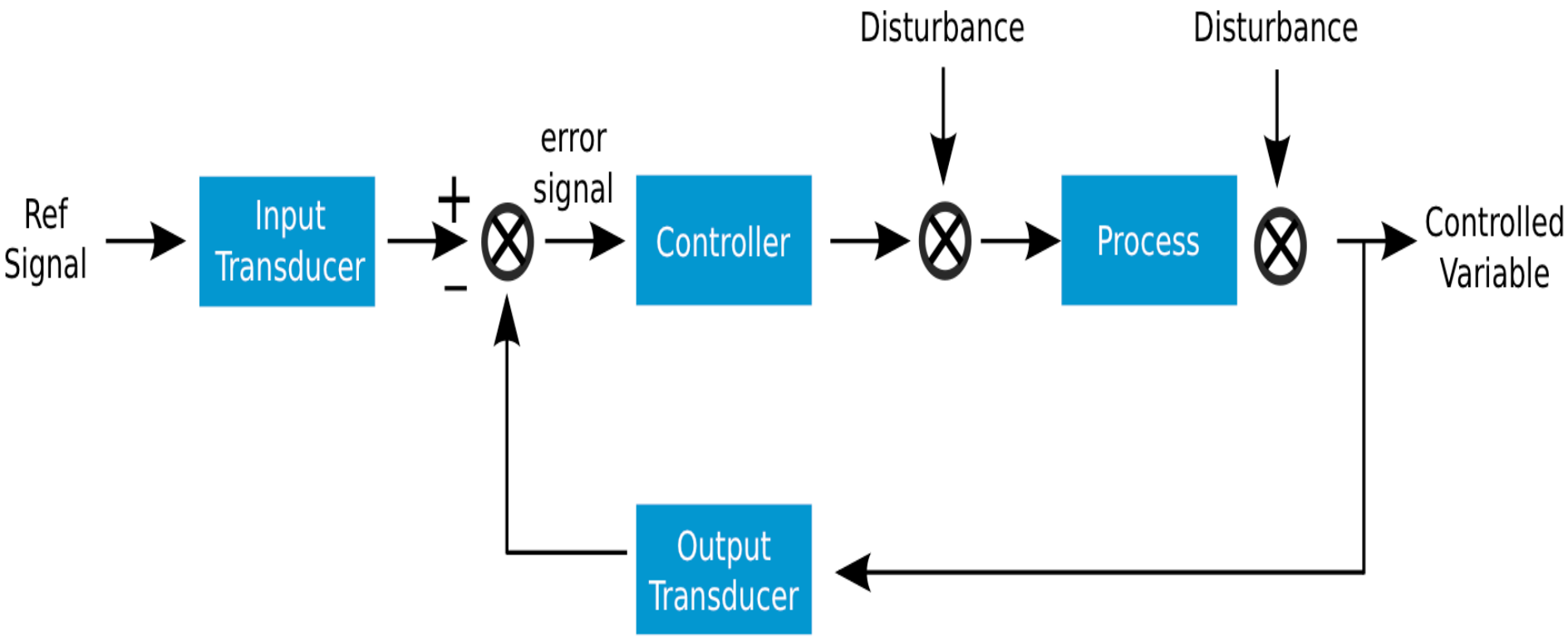
- require expert construction
- … and tuning
- opaque to end-users
- sometimes opaque to experts!
- No guarantee that they are doing the *right thing*

Make memory management rigorous!

Our approach

- Apply mathematical models from other disciplines
- Find appropriate analogy with GC, and apply
  - thermodynamics (Baker)
  - economics (Singer/Jones)
  - **control systems engineering (this project)**

What is a control system?


Disturbance
Disturbance
error
signal
Ref
Signal
Input
Transducer
+
−
Controller
Process
Controlled
Variable
Output
Transducer

How to apply controller to heap sizing?

- What will we control?
    - VM heap size
- What will we measure?
    - GC overhead

Use a standard black-box controller

- PID controller

$$u(t) = K_c \left(\epsilon(t) + \frac{1}{T_i} \int \epsilon(t)\ dt + T_d \frac{d\epsilon(t)}{dt} \right) + b$$

- $u(t)$ is heap resize ratio at time t
- $e(t)$ is difference between GC overhead at time t, and target GC overhead

Tune PID parameters

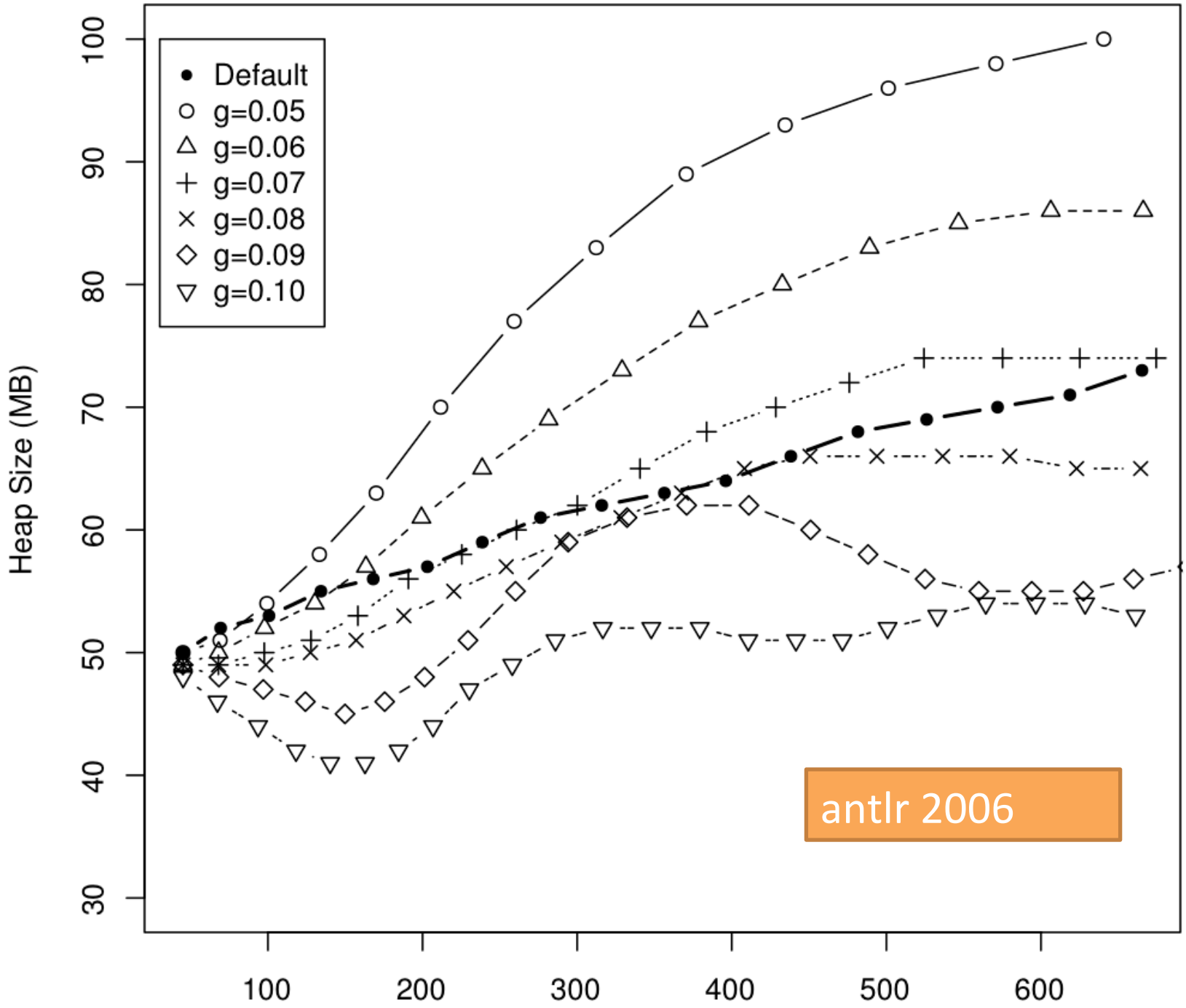
- Software system has advantages over mechanical system
- Feed in signal, look for frequency of oscillating response
- Apply standard Ziegler-Nichols equations to determine PID parameters

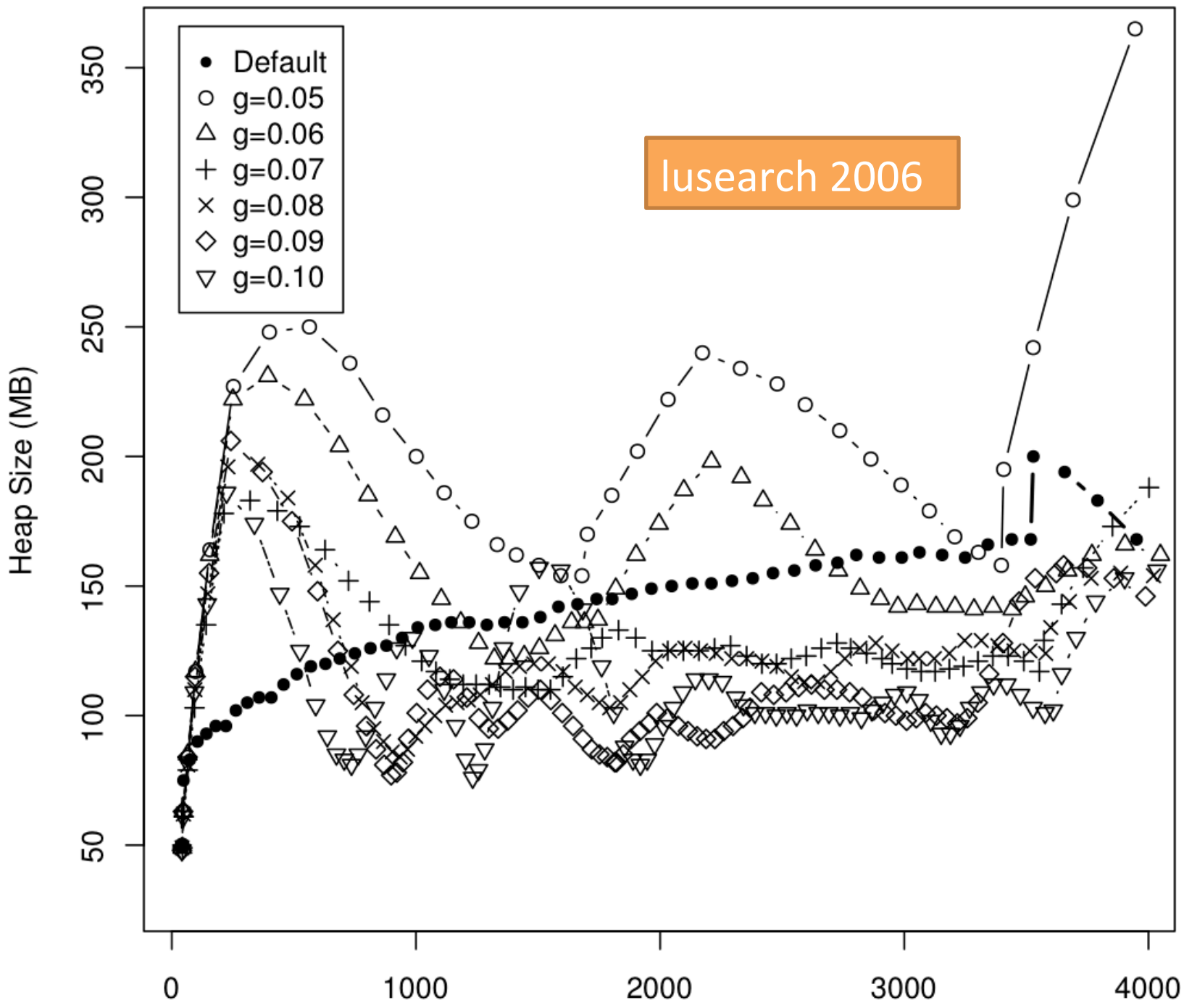
How often would we need to tune?

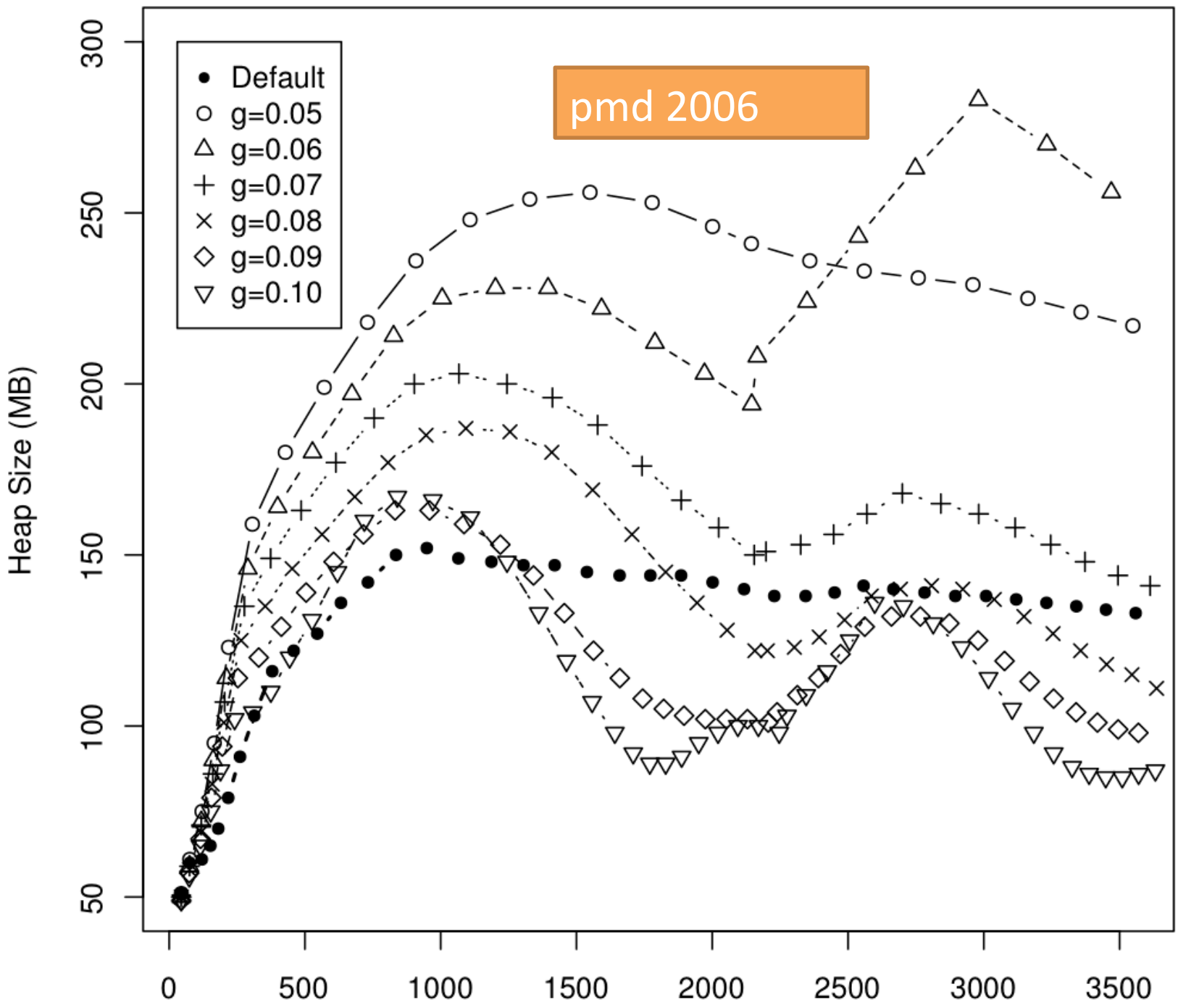
- at VM install time?
- per program?
- per program execution?
- when system updates?
- ???

How to execute…

- specify a desired GC overhead
- start system running…


Default
g=0.05
g=0.06
g=0.07
g=0.08
g=0.09
g=0.10
Heap Size (MB)
30
40
50
60
70
80
90
100
100
200
300
400
500
600
antlr 2006


Default
g=0.05
g=0.06
g=0.07
g=0.08
g=0.09
g=0.10
lusearch 2006
Heap Size (MB)
50
100
150
200
250
300
350
0
1000
2000
3000
4000


pmd 2006
Default
g=0.05
g=0.06
g=0.07
g=0.08
g=0.09
g=0.10
Heap Size (MB)
50
100
150
200
250
300
0
500
1000
1500
2000
2500
3000
3500

How to produce these graphs?

- Single runs, not repeated runs
- Give a range of GC overheads that bracket existing system behaviour
- x-axis is bytes allocated, not wallclock time
- what is a good result?

Are our new results better?

What does *better* mean?

- More responsive heap resizing
  - is this efficient?
- Reduced area under the heapsize/time curve
  - RAM rental – in Megabyte seconds (cloud)
- More flexible way to trade off time/space for computation?
- More intuitive for user?

Future work

- deal with paging (other side of curve)
  - requires another controller?
- deploy in more runtime systems
- demonstrate improvements
  - … make the world a better place

System of systems