University of Glasgow

Mathematical Memory Management

Jeremy.Singer@glasgow.ac.uk

@jsinger_compsci

- **dynamic memory allocation** requires a **runtime heap**
- use `malloc` and `free` to allocate and deallocate heap space

- Problems with *explicit* deallocation
  - forgotten `free()`
  - double `free()`

Automatic Memory Management

- a.k.a. *Garbage Collection (GC)*
- Automatically deallocate a block of memory when it is no longer reachable
- *Reachability* is conservative approximation for *liveness*

When are objects unreachable?

- use *reference counting*

- use *tracing*

GC varieties

- generational vs non-generational
- moving vs non-moving
- copying vs compacting
- stop-the-world vs concurrent

Live demo

Lots of possibilities

- How do you find the best settings for your system? … for your application?

1. domain expertise
2. exhaustive searching
3. machine learning

1. Domain Expertise

```
Java -Xmx12g -XX:MaxPermSize=64M -XX:PermSize=32M-XX:MaxNewSize=2g
     -XX:NewSize=1g -XX:SurvivorRatio=128 -XX:+UseParNewGC
     -XX:+UseConcMarkSweepGC -XX:MaxTenuringThreshold=0
     -XX:CMSInitiatingOccupancyFraction=60 -XX:+CMSParallelRemarkEnabled
     -XX:+UseCMSInitatingOccupancyOnly -XX:ParallelGCThreads=12
     -XX:LargePageSizeInBytes=256m ...
```

2. Exhaustive Search

The Taming of the Shrew: Increasing Performance by Automatic Parameter Tuning for Java Garbage Collectors

Philipp Lengauer
Christian Doppler Laboratory MEVSS
Johannes Kepler University Linz, Austria
philipp.lengauer@jku.at

Hanspeter Mössenböck
Institute for System Software
Johannes Kepler University Linz, Austria
hanspeter.moessenboeck@jku.at

ABSTRACT

Garbage collection, if not tuned properly, can considerably impact application performance. Unfortunately, configur-

However, while object allocations produce a direct and easy to understand performance impact, the costs of garbage collections are easily overlooked. Programmers are often unaware of the proportion their application spends on collect-

- around 300 GC parameters
- search parameter space for 4 hours
- select *optimal* configuration

3. Machine Learning

- if we can *characterise* application workloads in a general way, we can *correlate* these with appropriate GC configurations

- my ISMM 2007 paper “Intelligent Selection of Application-Specific Garbage Collectors”

[ISMM 2007]

Intelligent Selection of Application-Specific Garbage Collectors

Jeremy Singer Gavin Brown
Ian Watson
University of Manchester, UK
{jsinger,gbrown,iwatson}@cs.man.ac.uk

John Cavazos
University of Edinburgh, UK
jcavazos@inf.ed.ac.uk

Abstract

Java program execution times vary greatly with different garbage collection algorithms. Until now, it has not been possible to determine the best GC algorithm for a particular program without exhaustively profiling that program for

1. Introduction

1.1 Importance of GC

In managed runtime environments such as the Java Virtual Machine (JVM) and the Common Language Runtime

Feature vector

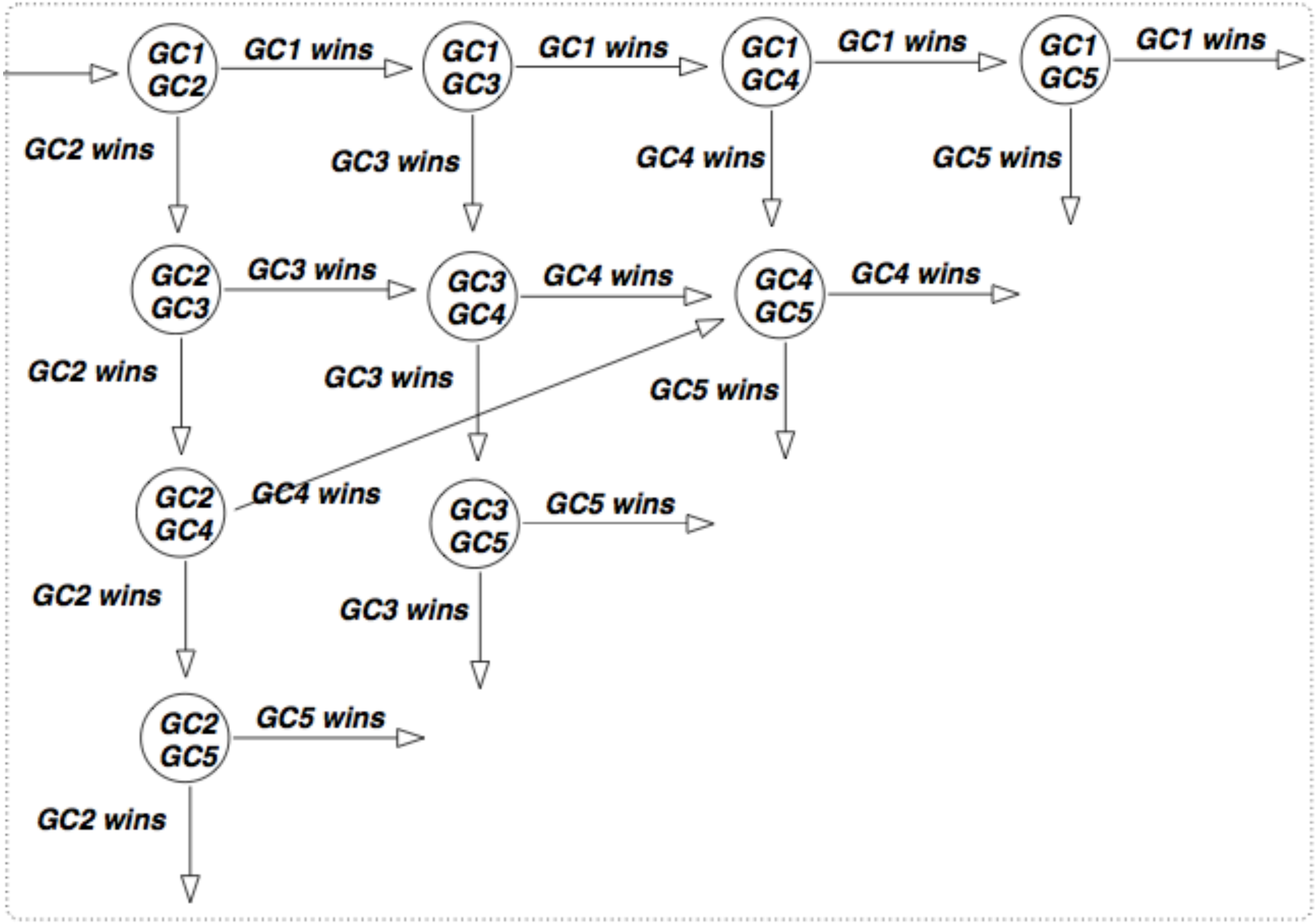
- characterizes a single Java application

- *static* (e.g. CK metrics, source code metrics)
- *dynamic* (e.g. object demographics)
- *VM* (e.g. #GCs in reference collector)

Training Phase

- Build a predictor based on performance of known benchmarks

- Tournament predictor, a forest of decision trees


GC1
GC2
GC1 wins
GC1
GC3
GC1 wins
GC1
GC4
GC1 wins
GC1
GC5
GC1 wins
GC2 wins
GC3 wins
GC4 wins
GC5 wins
GC2
GC3
GC3 wins
GC3
GC4
GC4 wins
GC4
GC5
GC4 wins
GC2 wins
GC3 wins
GC5 wins
GC2
GC4
GC4 wins
GC3
GC5
GC5 wins
GC2 wins
GC3 wins
GC2
GC5
GC5 wins
GC2 wins

Single Decision Tree

```
dynamic_num_bytes <= 91306040
|static_lack_of_cohesion_of_methods <= 5: Gen
|static_lack_of_cohesion_of_methods > 5
||dynamic_num_minor_gcs <= 6: NonGen
||dynamic_num_minor_gcs > 6: Gen
dynamic_num_bytes > 91306040
|static_lack_of_cohesion_of_methods <= 47371
||dynamic_arrays_size_u128B <= 0.11: Gen
||dynamic_arrays_size_u128B > 0.11
|||ratio_curr_to_min_heap <= 15.515152: Gen
|||ratio_curr_to_min_heap > 15.515152: NonGen
|static_lack_of_cohesion_of_methods > 47371: NonGen
```

Results

- Mean application speedup of 5% over set of 20 Java benchmarks.
- Oracle predictor suggested 17% speedup was possible.

We have ***characterized*** a GC/application interaction using statistics

– now –

Can we ***understand*** the interaction using an analogy?

[ISMM 2010]

The Economics of Garbage Collection

Jeremy Singer
University of Manchester
United Kingdom
jsinger@cs.man.ac.uk

Richard Jones
University of Kent
United Kingdom
R.E.Jones@kent.ac.uk

Gavin Brown Mikel Luján *
University of Manchester
United Kingdom

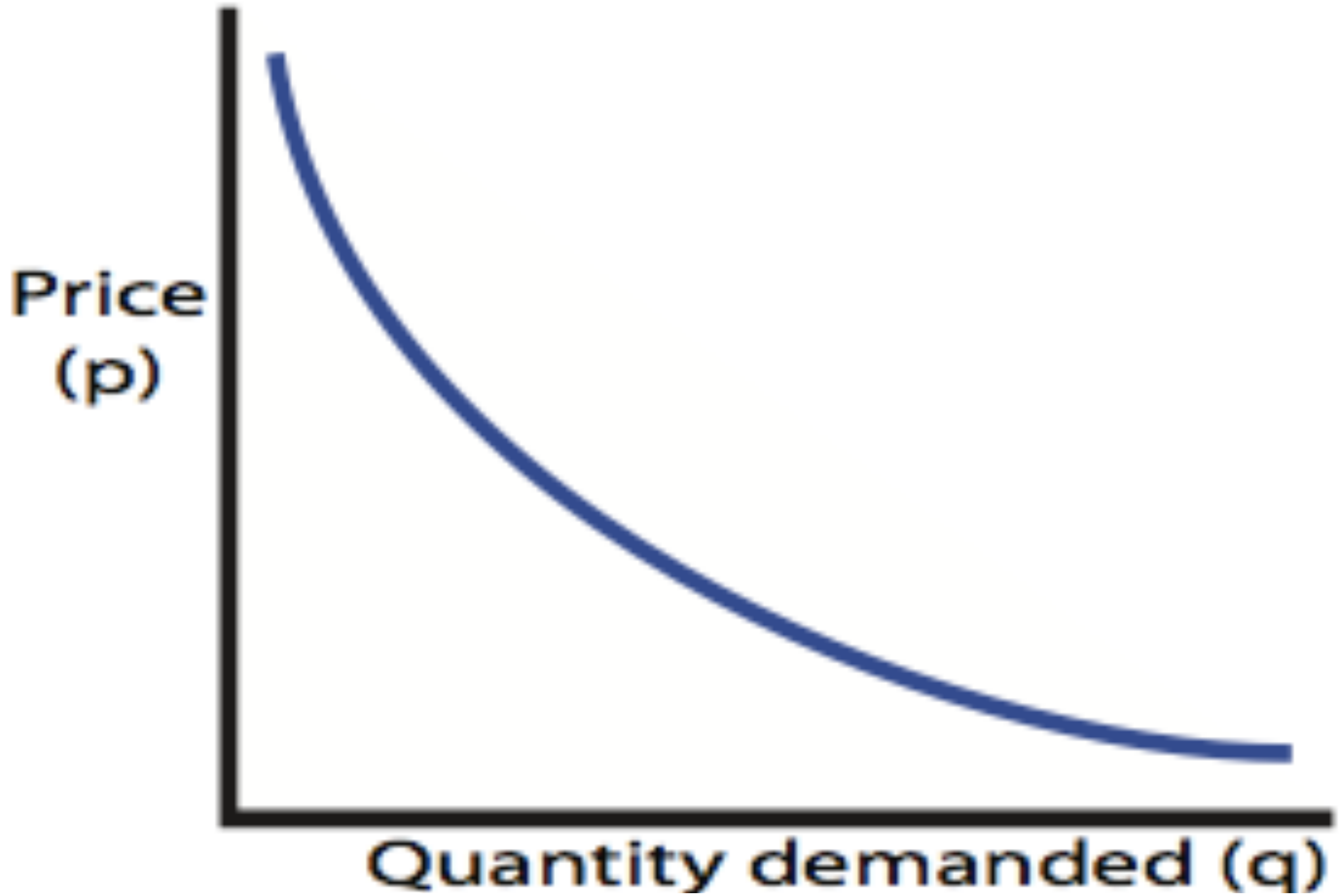
mailto:R.E.Jones@kent.ac.uk

Abstract

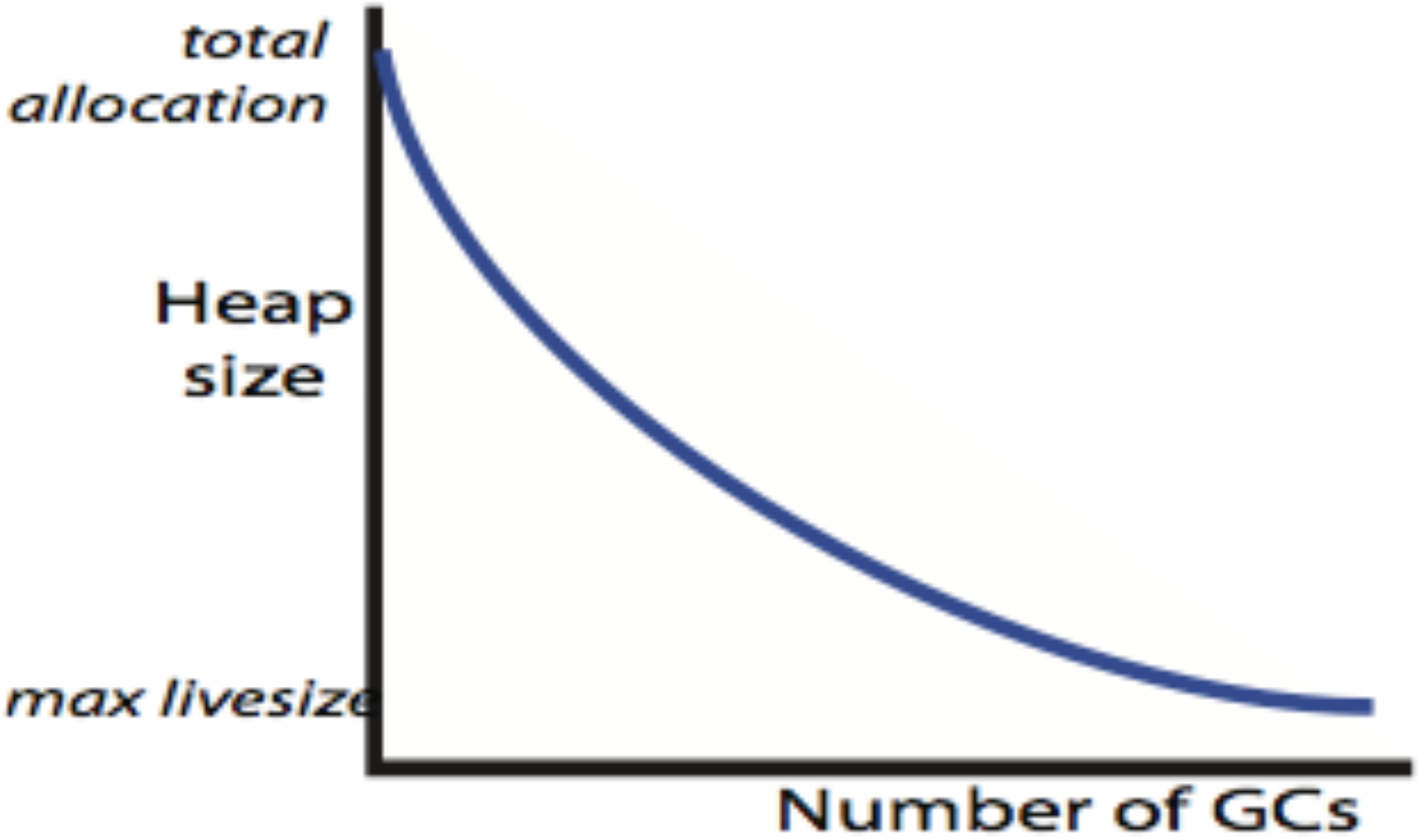
This paper argues that economic theory can improve our understanding of memory management. We introduce the *allocation curve*, as an analogue of the demand curve from microeconomics. [redacted]

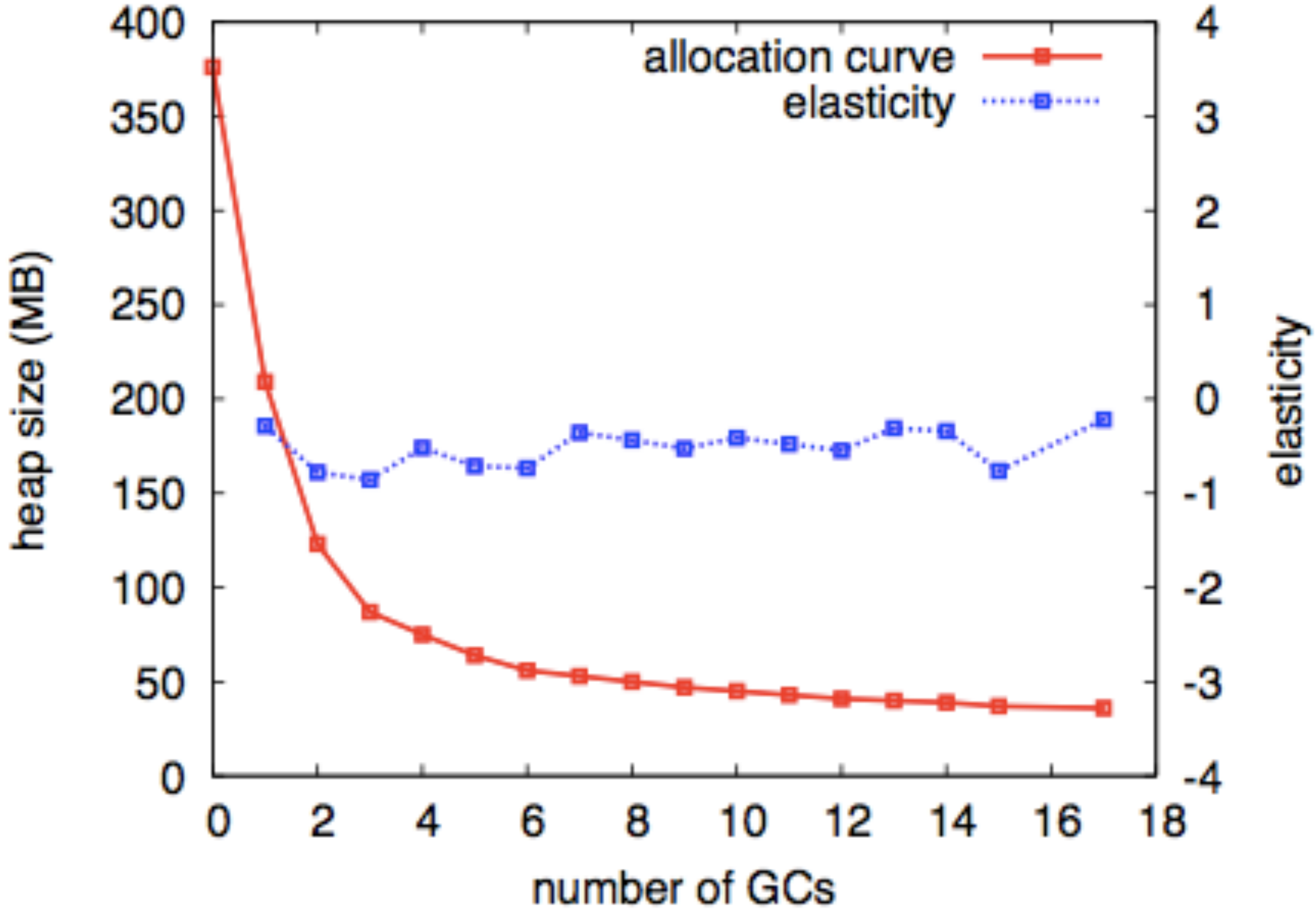
To the best of our knowledge, this is the first time that economic theory has been used in the context of automatic memory management. There are two main aims to our work. First, we intend to use economic theory to improve our understanding of memory management, by identifying parallels between concepts in each domain. [redacted]

economic demand curve



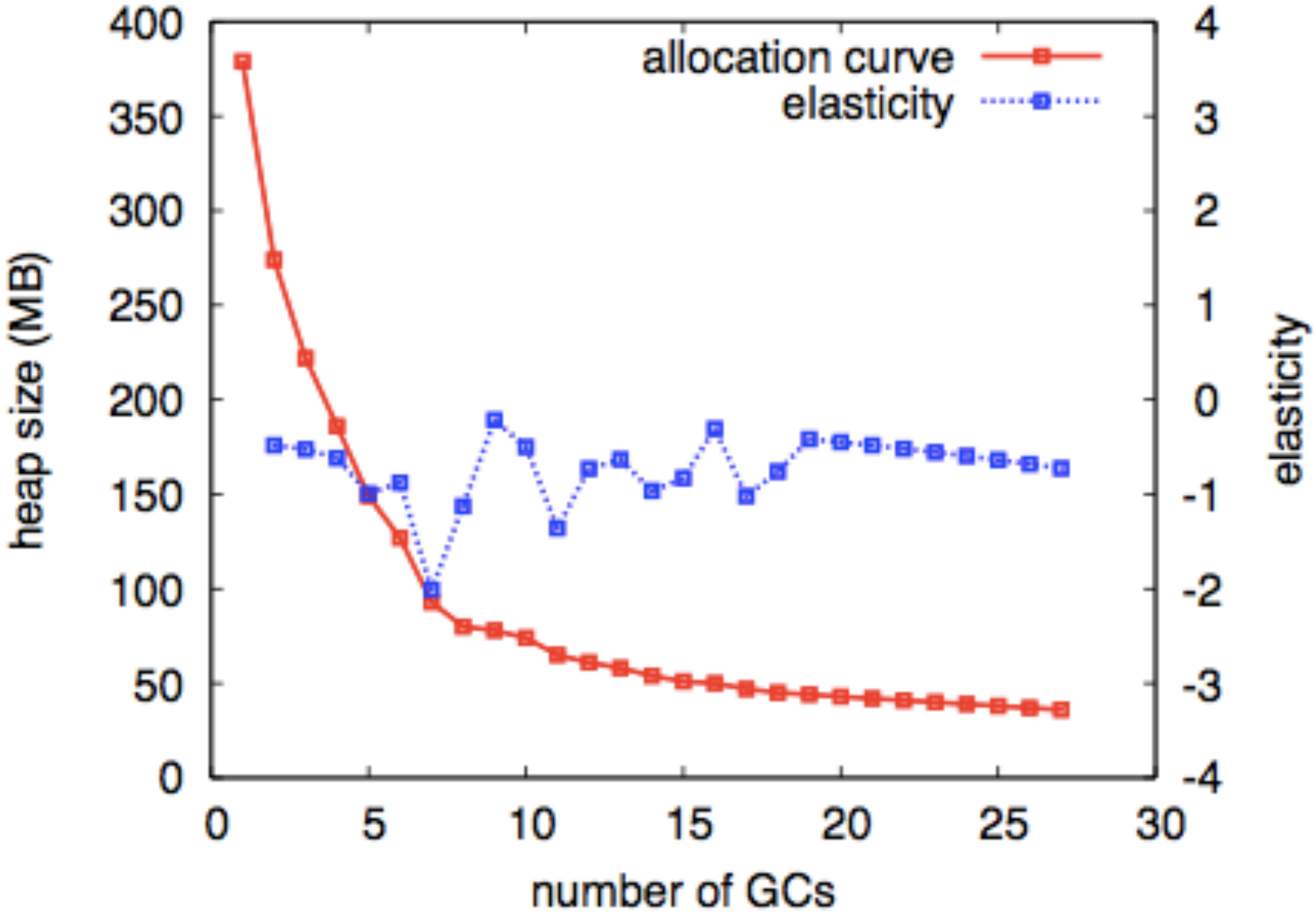
GC allocation curve





allocation curve
elasticity
heap size (MB)
elasticity
number of GCs
400
350
300
250
200
150
100
50
0
4
3
2
1
0
-1
-2
-3
-4
0
2
4
6
8
10
12
14
16
18

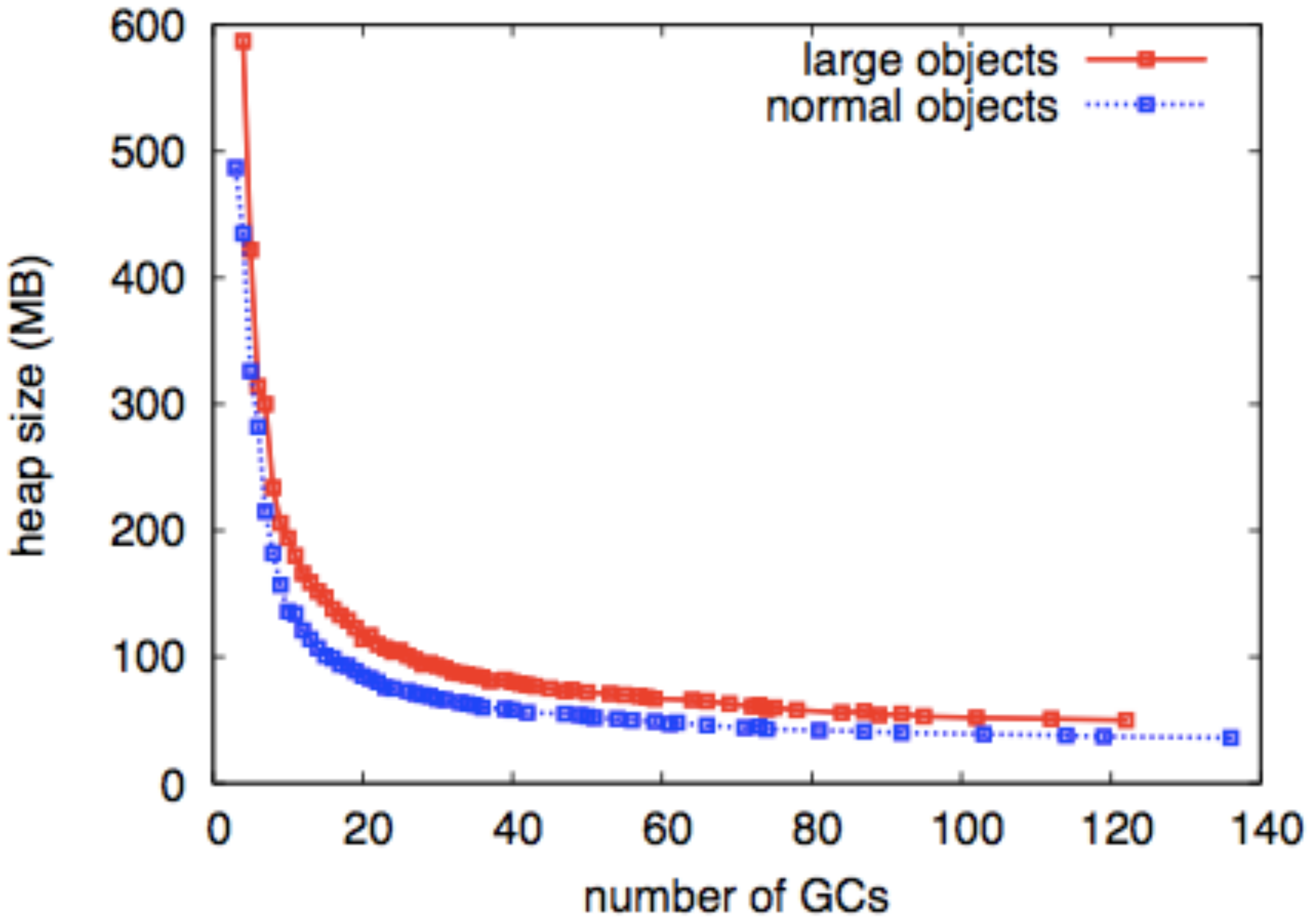
(a) antlr

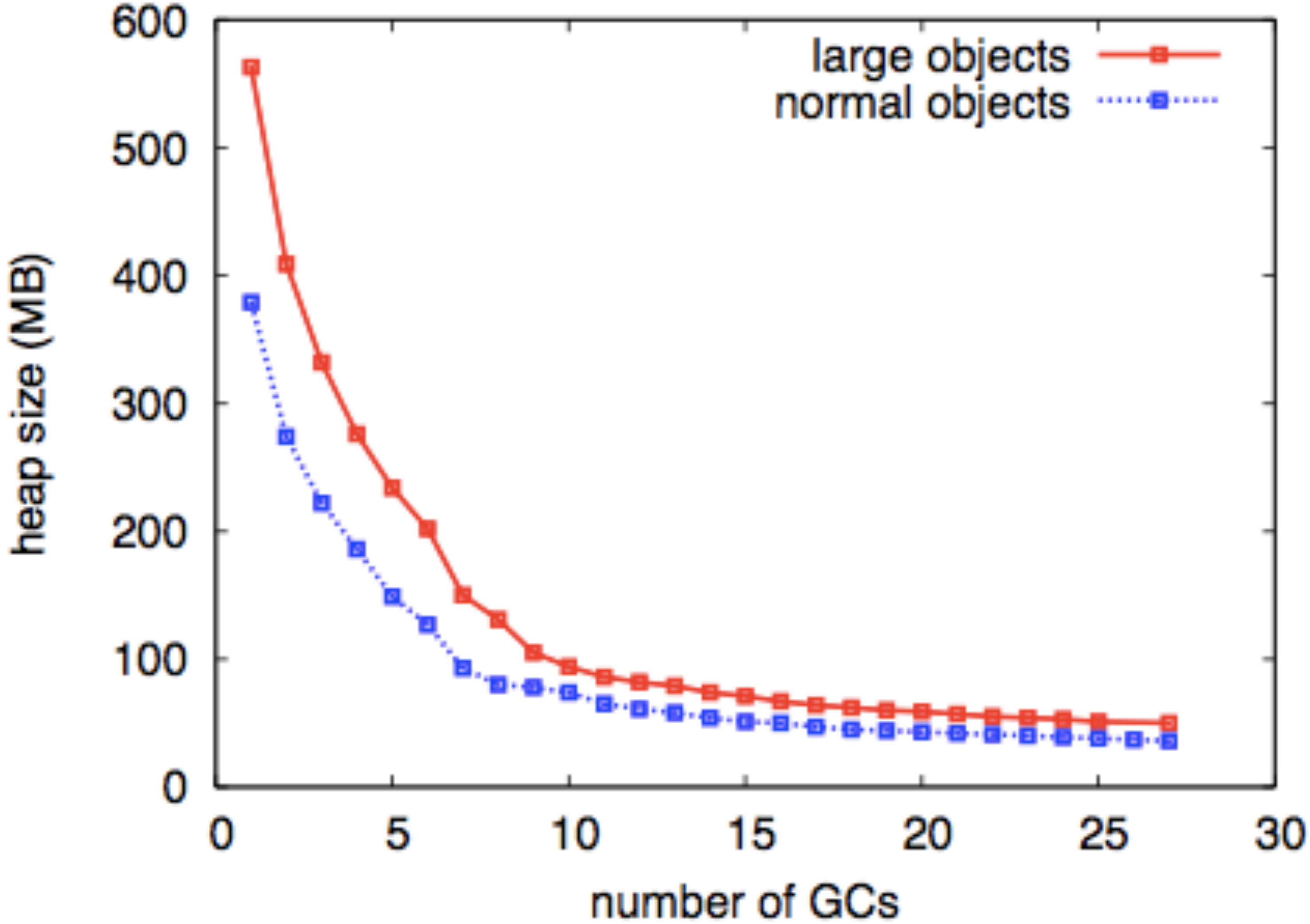

allocation curve
elasticity
heap size (MB)
elasticity
number of GCs
400
350
300
250
200
150
100
50
0
4
3
2
1
0
-1
-2
-3
-4
0
5
10
15
20
25
30

(e) luindex

Effect of *taxation*

- product tax *shifts* demand curve *up* price axis


large objects
normal objects
heap size (MB)
number of GCs
600
500
400
300
200
100
0
0
20
40
60
80
100
120
140
(b) bloat



large objects
normal objects
heap size (MB)
600
500
400
300
200
100
0
0
5
10
15
20
25
30
number of GCs

(e) luindex

Analogy

- price is like heap size
  - cost incurred
- consumer demand is like GC overhead
  - direct impact on actual consumer
- tax is like object header size
  - hidden overhead on every allocation

Why are analogies helpful?

- you help me!

We have *characterized* a GC/ application interaction using statistics

and *understood* the interaction using an analogy

– now –

Can we *control* the interaction using a mathematical model?

[ISMM 2013]

Control Theory for Principled Heap Sizing

David R. White Jeremy Singer
School of Computing Science
University of Glasgow
{david.r.white,jeremy.singer}@glasgow.ac.uk

Jonathan M. Aitken
Department of Computer Science
University of York
jonathan.aitken@york.ac.uk

Richard E. Jones
School of Computing
University of Kent
r.e.jones@kent.ac.uk

Abstract

We propose a new, principled approach to adaptive heap sizing based on control theory. We review current state-of-the-art heap sizing mechanisms, as deployed in Jikes RVM and HotSpot. We

paging [36]. Setting a large static heap size is an inefficient use of memory; this should be avoided.

This paper proposes the use of *control theory* [24] to adjust heap sizes dynamically. In contrast to existing, heuristic-based tech-

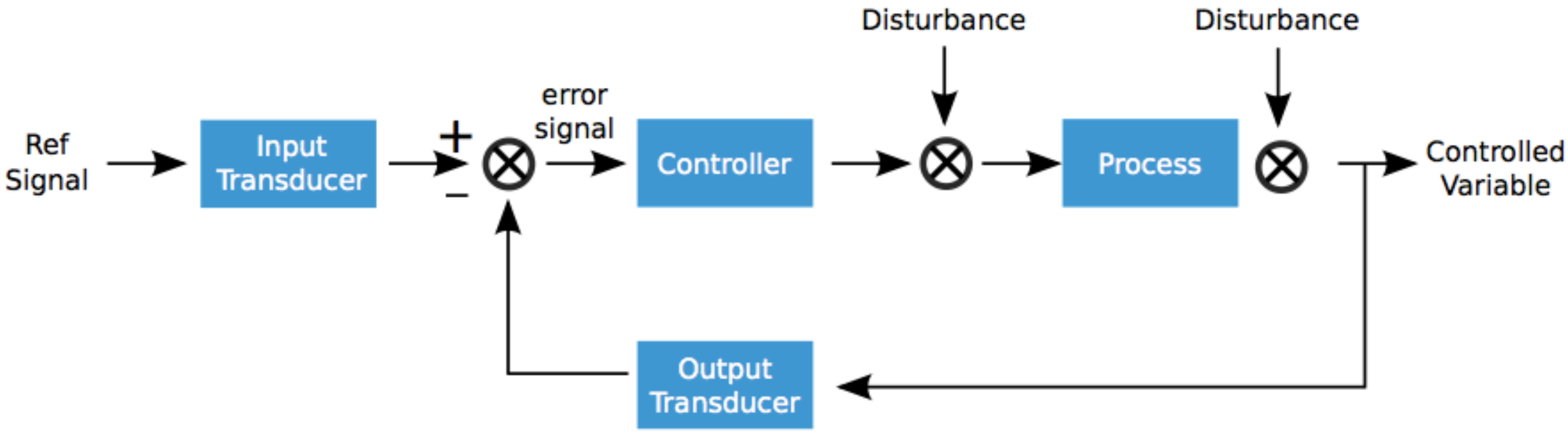


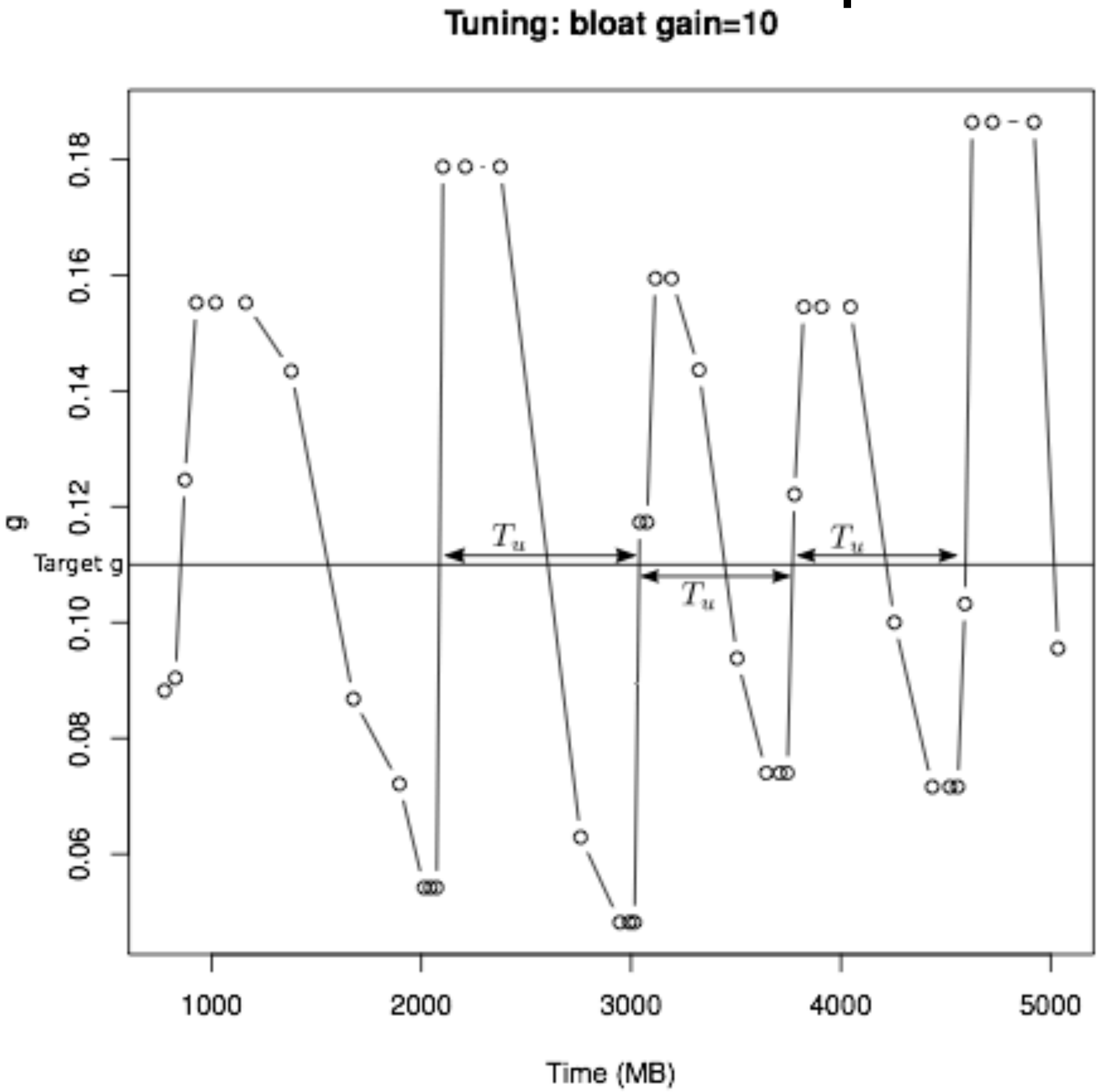
Figure 3: A closed-loop control system

- process: application running in JVM
- controlled variable: GC overhead [0,1]
- reference: target GC overhead
  - *set by user / sysadmin*
- error: difference between observed overhead and target overhead
- control: heap size
  - increase heap size => reduce GC overhead

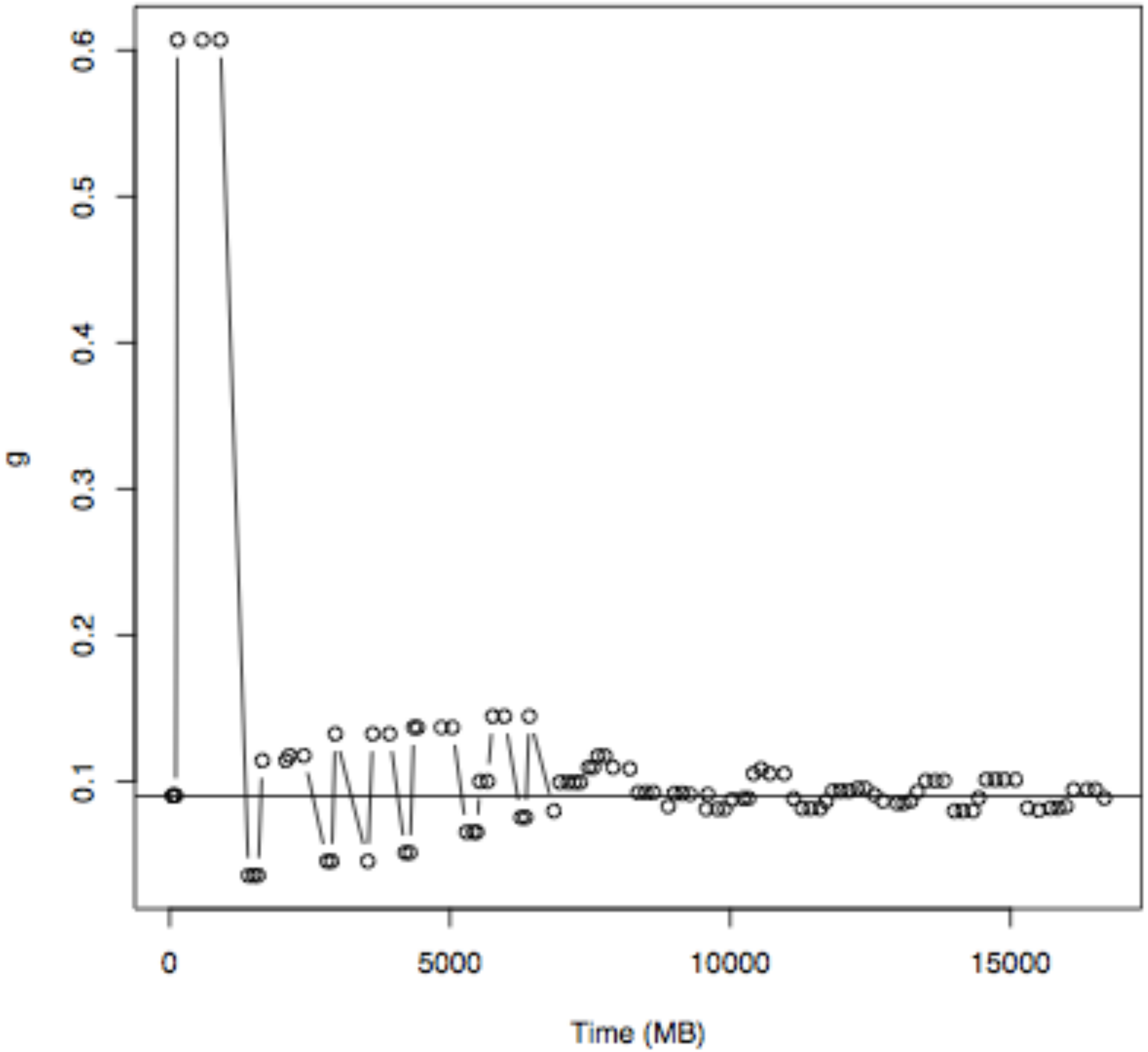
Mathematical Model: PID

$$u(t) = K_c \left(\epsilon(t) + \frac{1}{T_i} \int_0^t \epsilon(t)\, dt + T_d \frac{d\epsilon(t)}{dt} \right) + b$$

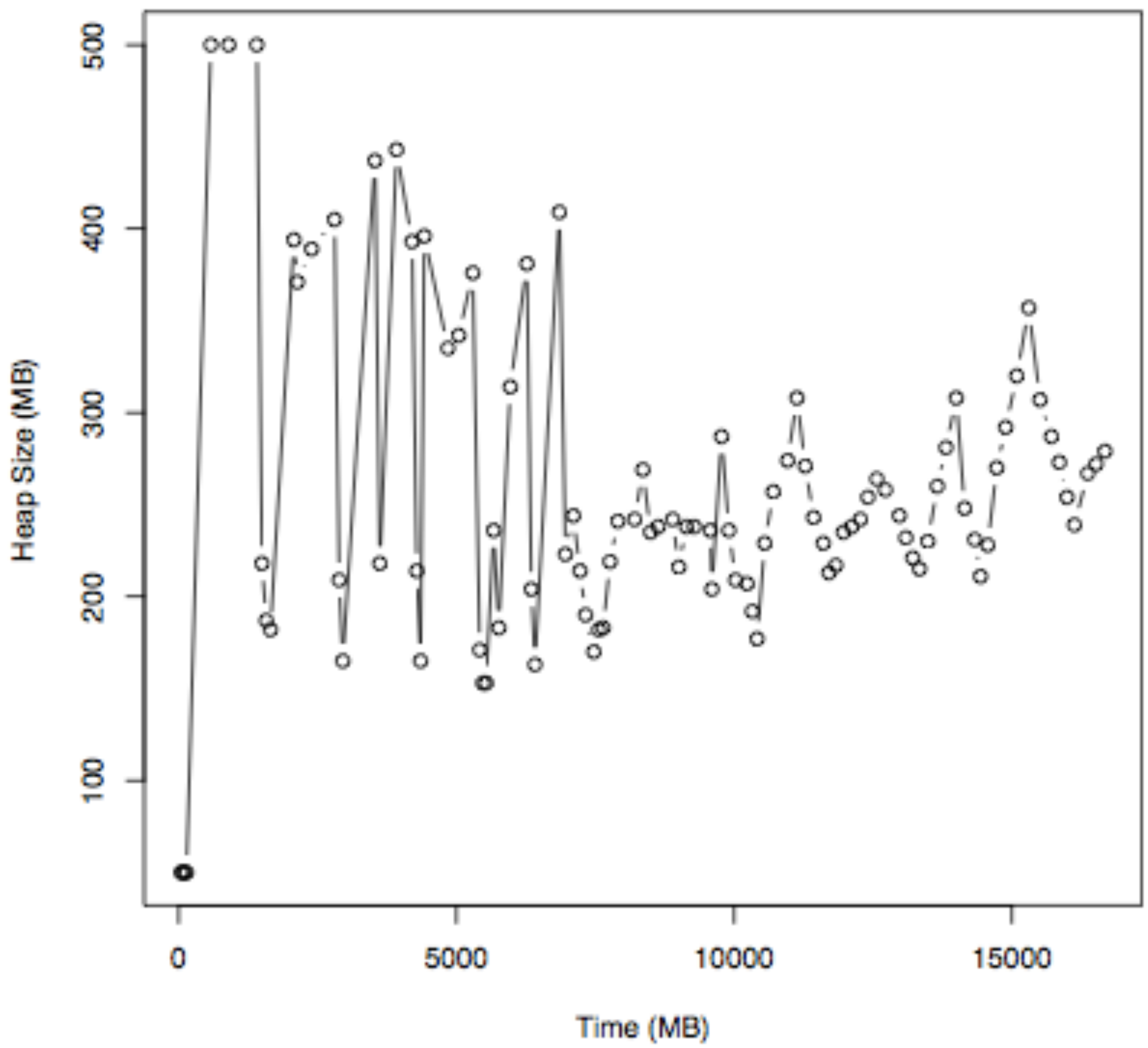
Tune to determine parameters



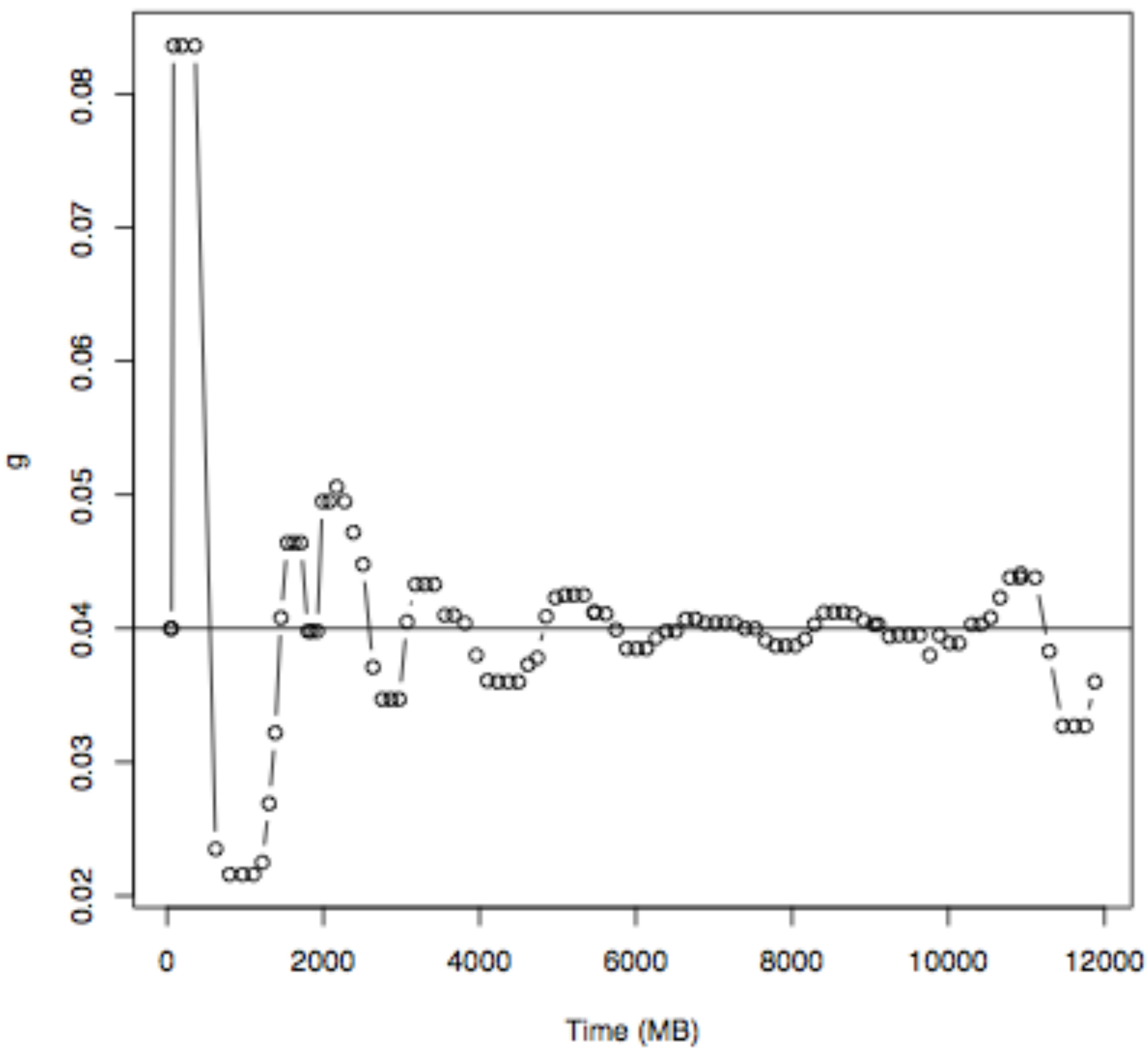
Examples of controlled systems



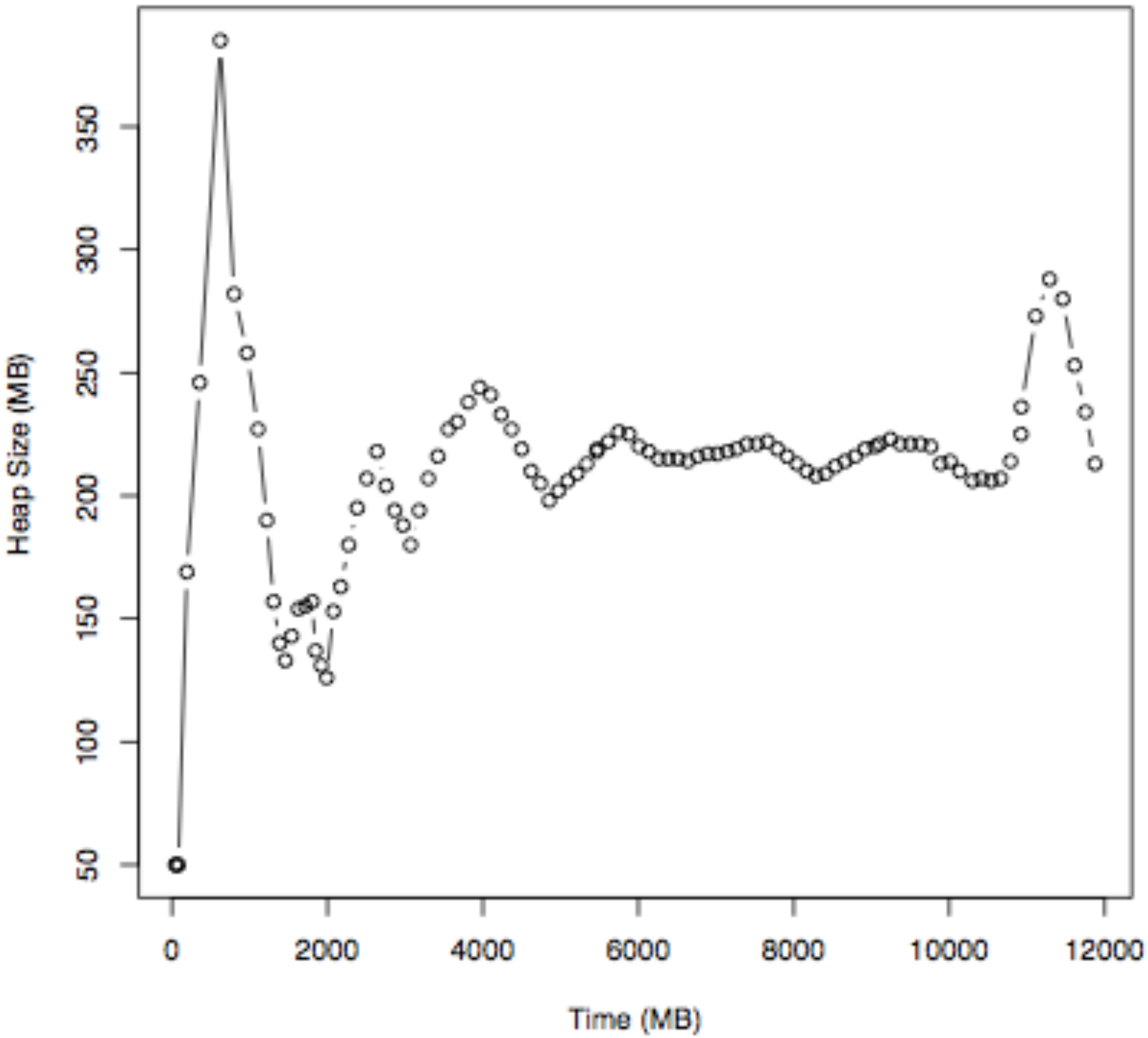
(a) GC Overhead for DaCapo 2009 pmd



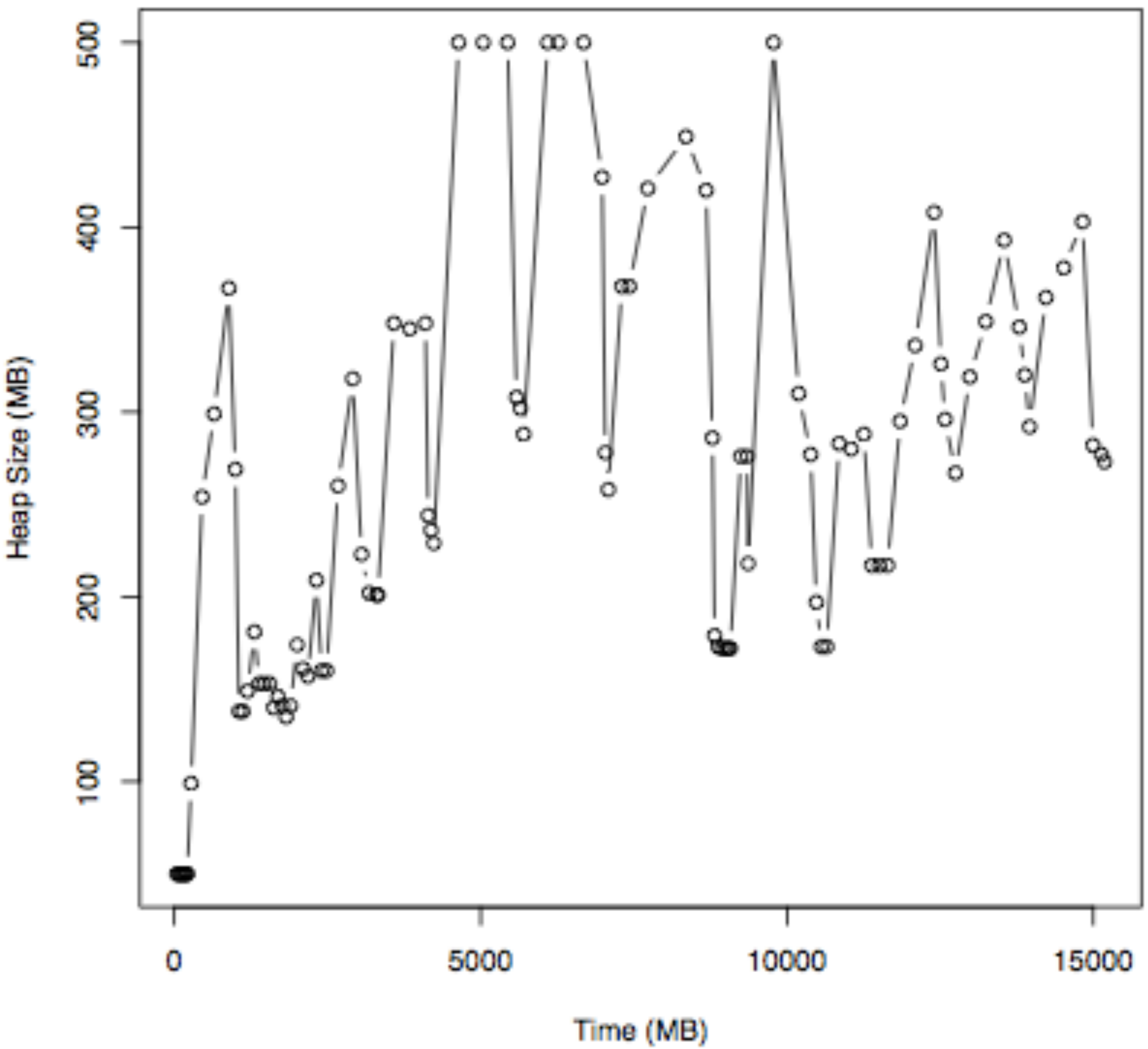
(d) Heap Size for DaCapo 2009 pmd



(c) GC Overhead for DaCapo 2009 xalan



(f) Heap Size for DaCapo 2009 xalan



(k) Heap Size for DaCapo 2006 eclipse

Conclusions

Garbage Collectors are Complex Software Systems

- Possible to *characterize* them and *control* them, using standard techniques
- statistical (machine learning, ISMM 2007)
- mathematical analogy (economics, ISMM 2010)
- differential equations (control theory, ISMM 2013)

Concluding Challenge

- I have looked at *Garbage Collection*
- For the complex software systems you study, which mathematical abstractions would be appropriate for *characterization* and *control*?