



An Adaptable Heap Visualisation Framework

Tony Printezis

tony@dcs.gla.ac.uk

Dept of Computing Science

University of Glasgow

Richard Jones

r.e.jones@ukc.ac.uk

Computing Laboratory

University of Kent at Canterbury

Overview

- ▶ Introduction ◀
- ❑ *Demo 1: Basics*
- ❑ Architecture
- ❑ *Demo 2: More Facilities*
- ❑ Conclusions / Future Directions

Who Can Use It?

- ❑ Three groups of people:

Who Can Use It?

- ❑ Three groups of people:
 - **Memory System Implementers**
 - Debug aspects of memory management mechanism
 - Discover performance bottlenecks

Who Can Use It?

□ Three groups of people:

⇒ **Memory System Implementers**

- Debug aspects of memory management mechanism
- Discover performance bottlenecks

⇒ **Educators**

- Illustrate algorithms
- Students might actually pay attention!

Who Can Use It?

□ Three groups of people:

⇒ **Memory System Implementers**

- Debug aspects of memory management mechanism
- Discover performance bottlenecks

⇒ **Educators**

- Illustrate algorithms
- Students might actually pay attention!

⇒ **Sophisticated Application Developers**

- Observe application's memory behaviour
- Possibly optimise allocation / memory access pattern

Visualisation Granularity Trade-Offs

❑ **Fine Granularity (say object-level)**

- ✓ high detail
- ✓ can perform advanced processing over accurate data
- ✗ slow to collect / send / store data
- ✗ large data volumes
- ✗ scalability problems
- ✗ screen size limitations

❑ **Coarse Granularity (say block-level)**

- ✗ lower detail
- ✗ information loss
- ✓ faster to collect / send / store data
- ✓ smaller data volumes
- ✓ can adjust detail to deal with scalability and screen limitation problems

Visualisation Granularity Trade-Offs

❑ **Fine Granularity (say object-level)**

- ✓ high detail
- ✓ can perform advanced processing over accurate data
- ✗ slow to collect / send / store data
- ✗ large data volumes
- ✗ scalability problems
- ✗ screen size limitations

▶ **Coarse Granularity (say block-level)** ◀

- ✗ lower detail
- ✗ information loss
- ✓ faster to collect / send / store data
- ✓ smaller data volumes
- ✓ can adjust detail to deal with scalability and screen limitation problems

Terminology

- ❑ The heap is split into fixed-size *Blocks*
- ❑ Each block is represented on the screen by a rectangular *Tile*
- ❑ The way the tile is painted represents some attribute(s) of the block
 - ⇒ percentage of used space
 - ⇒ number of objects
 - ⇒ etc.
- ❑ A *Space* is a collection of consecutive tiles on the screen
- ❑ Each space represents a *Component* of a system
 - ⇒ for example, generations, free lists, etc.
 - ⇒ component divided into logical blocks
- ❑ Several distinct spaces might be required

Overview

- ❑ Introduction
- ▶ *Demo 1: Basics* ◀
- ❑ Architecture
- ❑ *Demo 2: More Facilities*
- ❑ Conclusions / Future Directions

Overview

- ❑ Introduction
- ❑ *Demo 1: Basics*
- ▶ **Architecture** ◀
- ❑ *Demo 2: More Facilities*
- ❑ Conclusions / Future Directions

Client-Server Model

- ❑ Client-Server
 - ⇒ the VM is the server
 - ⇒ the visualiser is the client
- ❑ Flexible
 - ⇒ visualiser is running inside a separate process
 - minimal interference to the VM process
 - ⇒ can run client on a different machine if necessary
 - ⇒ can connect / disconnect the visualiser as / when necessary
 - ⇒ the visualiser can be written in a different language to the VM
- ❑ Communication is performed over simple TCP sockets
 - ⇒ portable
 - ⇒ as architecture, OS, and VM independent as possible

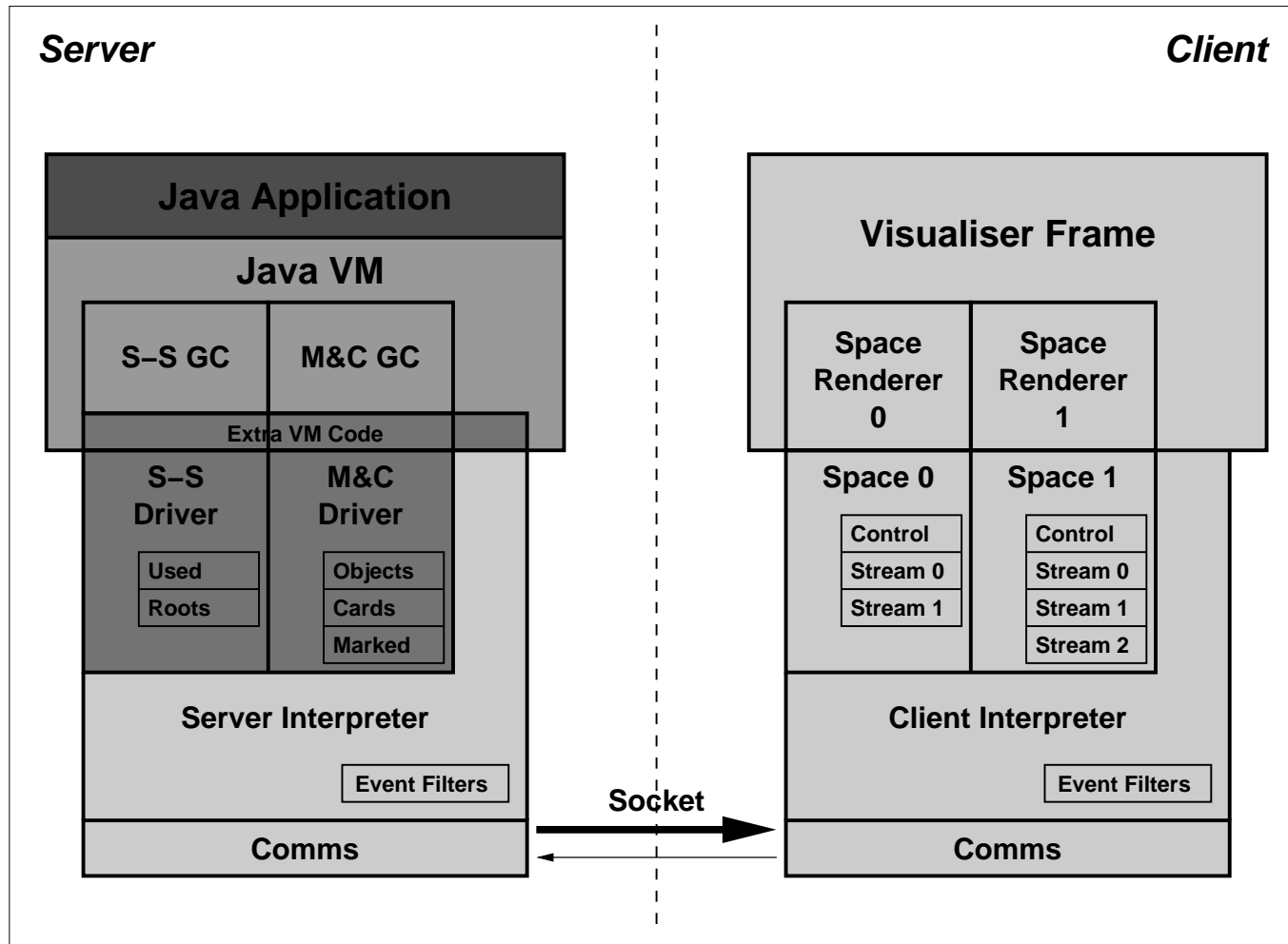
Customisation

- ❑ All aspects of **GCspy** are totally customisable
 - ⇒ number of spaces used
 - ⇒ number of attributes per space
 - ⇒ names of attributes / spaces / etc.
 - ⇒ etc.

- ❑ *None of the above is hardcoded in the visualiser*
 - ⇒ instead, bootstrap information is sent at connection-time

- ❑ Abstractions
 - ⇒ a space has a number of associated *Streams*
 - ⇒ each stream contains a value for each tile in the space
 - ⇒ a stream represents an attribute of the space
 - ⇒ presentation information per stream / space is necessary

Architecture



Data Collection in ResearchVM

❑ Piggy-Backing on GC Operations

- ✓ small performance penalty
- ✗ cannot easily turn data collection on and off
- ✗ affecting large parts of the GC code

❑ Complete Heap Sweeps

- ✗ sweeps are slow
- ✓ can efficiently turn data collection on and off
- ✓ code localised

❑ Concurrently

- ✓ doesn't increase pause times
- ✗ information might be inconsistent
- ✗ might interfere with thread scheduling

Data Collection in ResearchVM

❑ Piggy-Backing on GC Operations

- ✓ small performance penalty
- ✗ cannot easily turn data collection on and off
- ✗ affecting large parts of the GC code

▶ Complete Heap Sweeps ◀

- ✗ sweeps are slow
- ✓ can efficiently turn data collection on and off
- ✓ code localised

❑ Concurrently

- ✓ doesn't increase pause times
- ✗ information might be inconsistent
- ✗ might interfere with thread scheduling

❑ GCspy does *not* impose a data collection method

Events

- ❑ A set of events is defined in the VM
 - ⇒ start of young GC
 - ⇒ end of young GC
 - ⇒ start of old GC
 - ⇒ etc.
- ❑ Transmissions to the client are only sent at event points
- ❑ Pausing also only happens at event points
- ❑ Events are (of course!) customisable
 - ⇒ sent to the visualiser at connection-time

Event Filters

- ❑ Four operations are associated with each event
 - ⇒ **Enable / Disable** event
 - ⇒ **Delay** after event
 - ⇒ **Pause** after event
 - ⇒ **Period** (skip n events)
- ❑ These are sent to and implemented by the server, not the client
 - ⇒ when an event is disabled or skipped, the VM operates at full speed
- ❑ Event filters allow the user to
 - ⇒ easily find the events they are more interested in
 - ⇒ disable / skip events to compensate for the slow data collection times

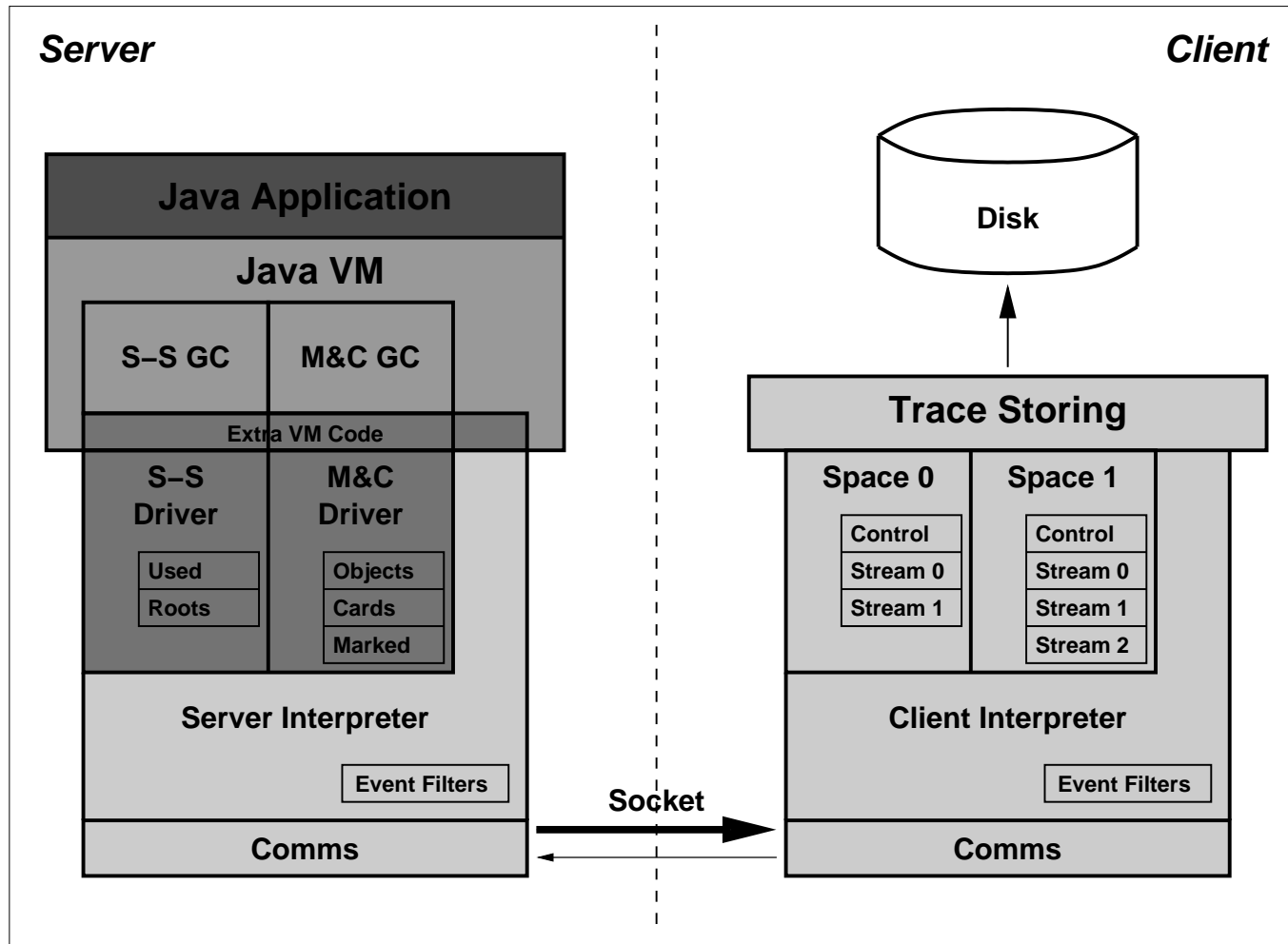
Trace Storing / Replaying Facilities

- ❑ Goal: store transmissions from the VM and replay them at a later time
 - ⇒ compare behaviour of old and improved system
 - ⇒ distribute them to other people
 - ⇒ decrease probability of “demoitis!”

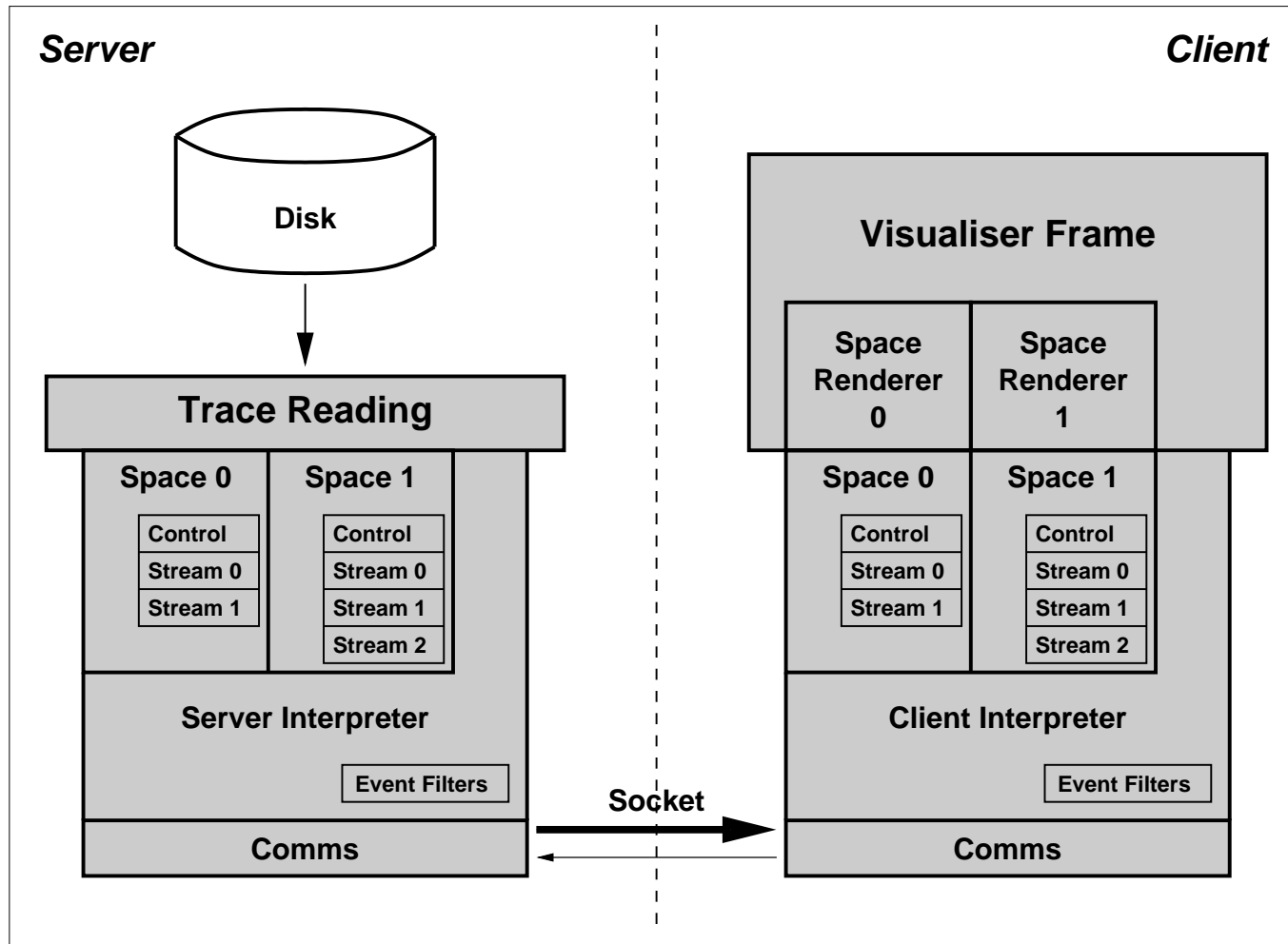
- ✓ Trace-replaying tool behaves *exactly* like the VM
- ✓ Compact size (especially with compression)
 - ⇒ complete trace of a large Javac job: 2.6MB
- ✓ More “realistic” trace replaying
 - ⇒ can compensate for data-collection times

- ✗ Coarse granularity does *not* allow detailed statistics gathering

Trace Storing



Trace Replaying



Overview

- ❑ Introduction
- ❑ *Demo 1: Basics*
- ❑ Architecture
- ▶ *Demo 2: More Facilities* ◀
- ❑ Conclusions / Future Directions

Overview

- ❑ Introduction
- ❑ *Demo 1: Basics*
- ❑ Architecture
- ❑ *Demo 2: More Facilities*
- ▶ **Conclusions / Future Directions** ◀

Portability / Ease Of Use

- ❑ The abstractions introduced allow **GCspy** to be *highly portable*
 - ⇒ initial development was done using ResearchVM
 - S-S, M&C, M&S, CM&S, and F-L drivers
 - Train driver on the way!
 - ⇒ plans to include it in other systems are under way
 - HotSpot
 - Sphere (persistent store)
 - Jikes RVM (server in Java!)
 - Dylan (BDW garbage collector)
 - C allocators?

- ❑ Ease of use
 - ⇒ about 30 mins to add a new stream (includes data collecting code)
 - ⇒ about 2 hours to deduce the CM&S driver from the M&S driver
 - ⇒ about 2 hours to write and incorporate the F-L driver

Future Directions

- ❑ Better, more flexible visualisations
 - ⇒ show several streams at once
- ❑ Other visualisation styles?
 - ⇒ e.g. scatter plots, histograms
- ❑ 2D organisation for some spaces
- ❑ Rewind facilities
 - ⇒ looping?
- ❑ Introduce tile re-ordering
- ❑ Breakpoints

- ❑ Can we provide memory behaviour profiling facilities to developers?

Concluding Remarks

- ❑ Introduced **GCspy**, a heap visualisation framework
 - ⇒ coarse-grain heap visualisation
 - ⇒ client-server architecture
 - ⇒ easily incorporated in most systems
 - ⇒ highly customised
 - ⇒ trace storing / replaying facilities
 - ⇒ history traces

“I must say that perhaps these are the neatest graphical summaries of a complete heap history I’ve ever seen!”

— M. Wolczko, SunLabs West

For more information...

For screenshots, documents, talks, etc.:

<http://www.dcs.gla.ac.uk/~tony/gcspy.www>

