

Visualising Dynamic Memory Allocators

A.M. Cheadle, A.J. Field, J.W. Ayres, N. Dunn,
R.A. Hayden, J. Nyström-Persson

Department of Computing, Imperial College London

Motivation...

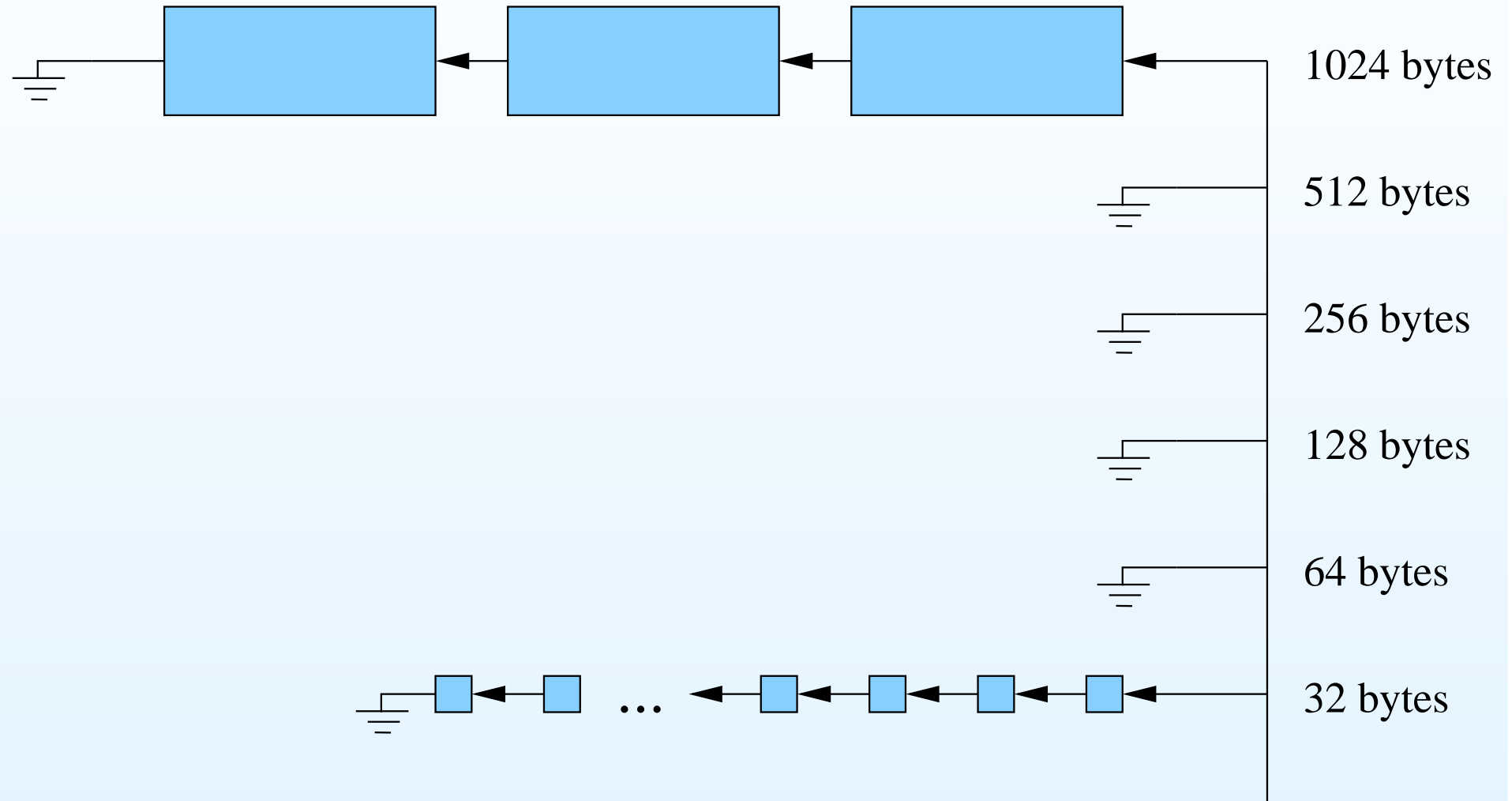
Visualisation of:

- 1 dmalloc
- 2 GHC (incremental collector and block allocators)
- 3 ECLiPSe Constraint Logic Programming System
- 4 Shared Memory Heap Layers (Telco in-memory database)

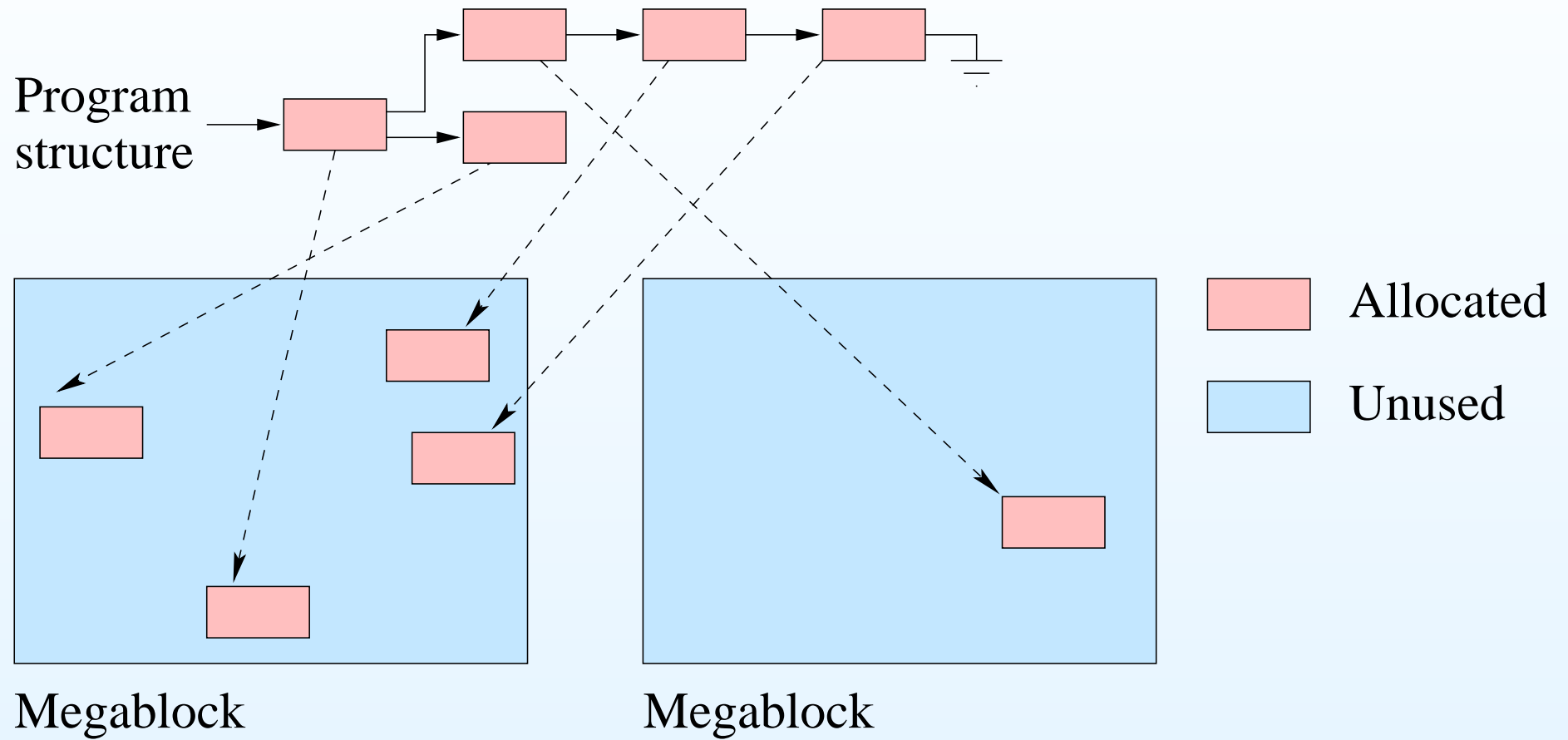
The Problem...

- You build a “memory manager” (custom, general-purpose...)
 - Is it buggy?
 - Is it performing well?
 - Can we optimise it?
- Standard debuggers may not help much
- Sometimes a picture paints a thousand words...

Poor segregated free list sizing?



Why is the memory footprint so big?

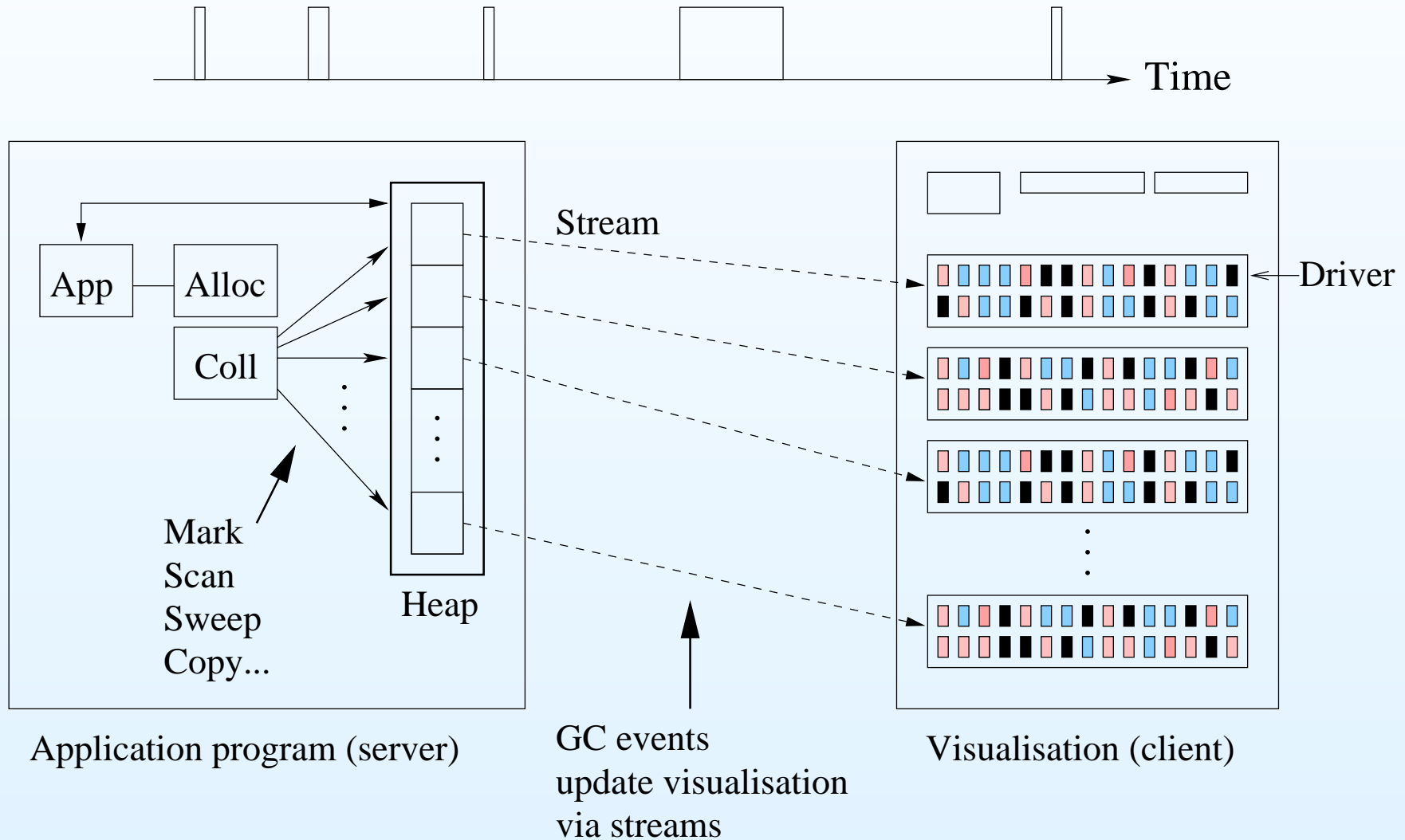


GCspy

- A visualisation framework typically used for rendering heap state
- Client (visualiser) / Server (application) architecture
- Tailored to GC:
 - Low-frequency events (e.g. minor/major collection)
 - Typically “flat” region-based heap layout

Current GCspy Model

GC events (minor/major collections), $O(0.1 - 10)?$ per second



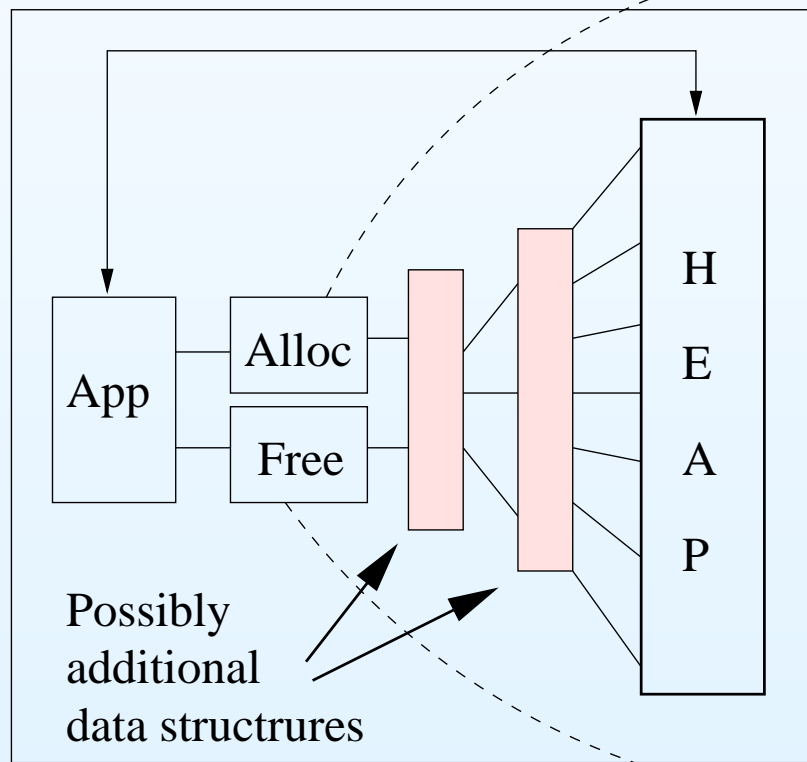
Visualising a Generational Collector

The screenshot displays the GCspy Windows interface, which provides a visual overview of the garbage collection process. The interface is divided into several sections:

- Current Event:** Shows the current event as "End Young GC".
- Tile Info:** Provides details for the selected block (Block 77):
 - Block 77 [44314000-44318000)
 - Block Size: 16K
 - Used Space: 16,384 bytes (100.0%)
 - Objects: 793
 - Reference Fields: 2,173
 - Refs To Old: 796
 - Refs To Permanent: 885
 - Internal Refs: 933
- View Chooser:** A dropdown menu currently set to "Objects".
- Magnification:** A zoom control with a slider and a color legend.
- Young Generation [S-S GC]:** A visualization showing the state of the young generation. It includes a "View: Objects" link, a progress bar, and buttons for "Activate", "Summary", "History", "Legend", "Colors", and "Clear Markers".
- Old Generation [M&C GC]:** A visualization showing the state of the old generation. It includes a "View: Objects" link, a large grid of colored blocks, and buttons for "Activate", "Summary", "History", "Legend", "Colors", and "Clear Markers".
- Permanent Space [M&C GC]:** A visualization showing the state of the permanent space. It includes a "View: Objects" link, a progress bar, and buttons for "Activate", "Summary", "History", "Legend", "Colors", and "Clear Markers".
- Control Panel:** Located at the bottom, it includes a play/pause button, a "Connected to arrow:3000 [Sun HotSpot] (Paused)" status indicator, and "Connect" and "Disconnect" buttons.

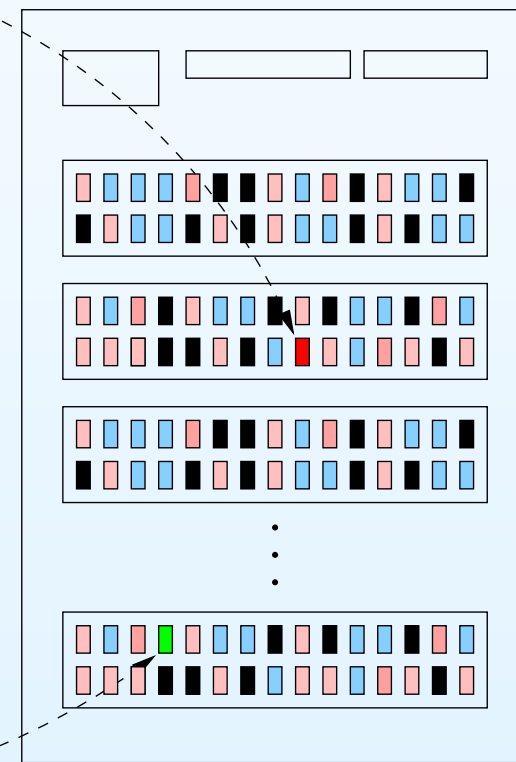
What about Dynamic Allocators?

Alloc/dealloc events $O(10^4 - 10^6)$? per second



Application program (server)

Individual
allocs/frees
(high volume)



Visualisation (client)

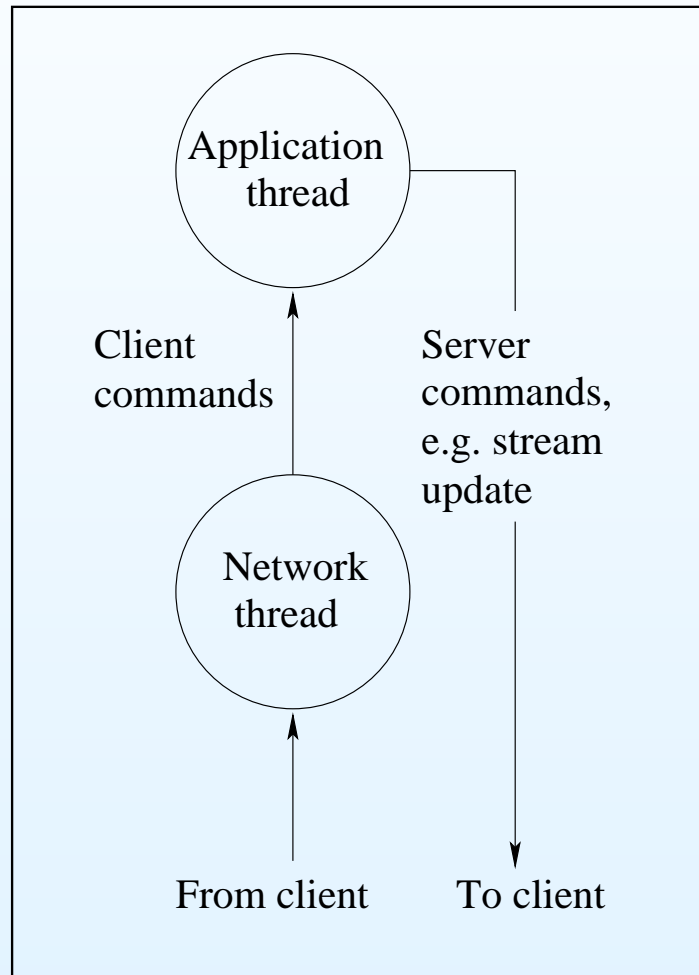
Goals

Extend GCspy to provide built-in support for:

- 1 High volume, fine-grain event handling
- 2 Targeted hierarchical visualisation
- 3 Additional performance debugging capabilities

1. High-volume Fine-grain Event Handling

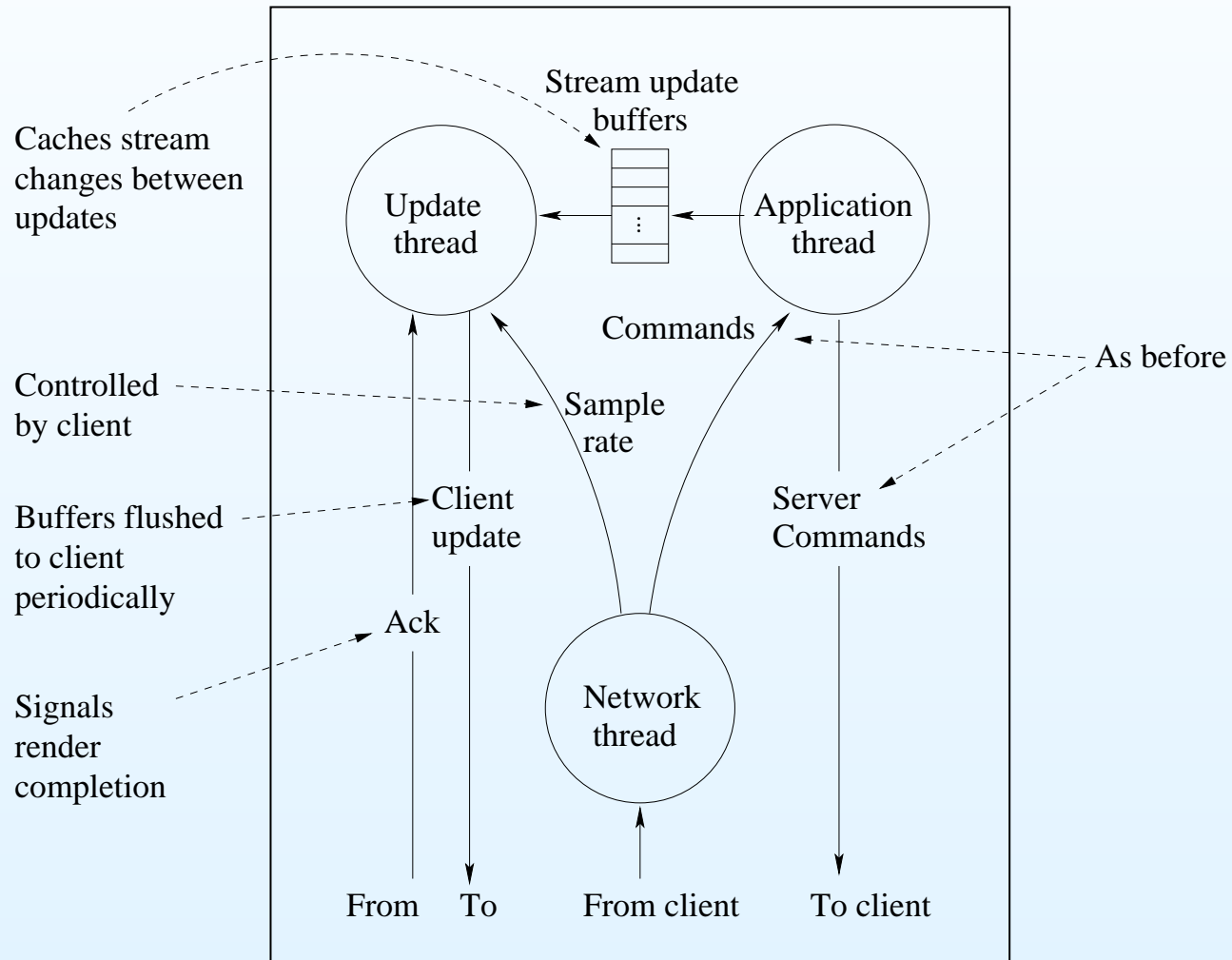
- Current Server Architecture:



Server thread architecture

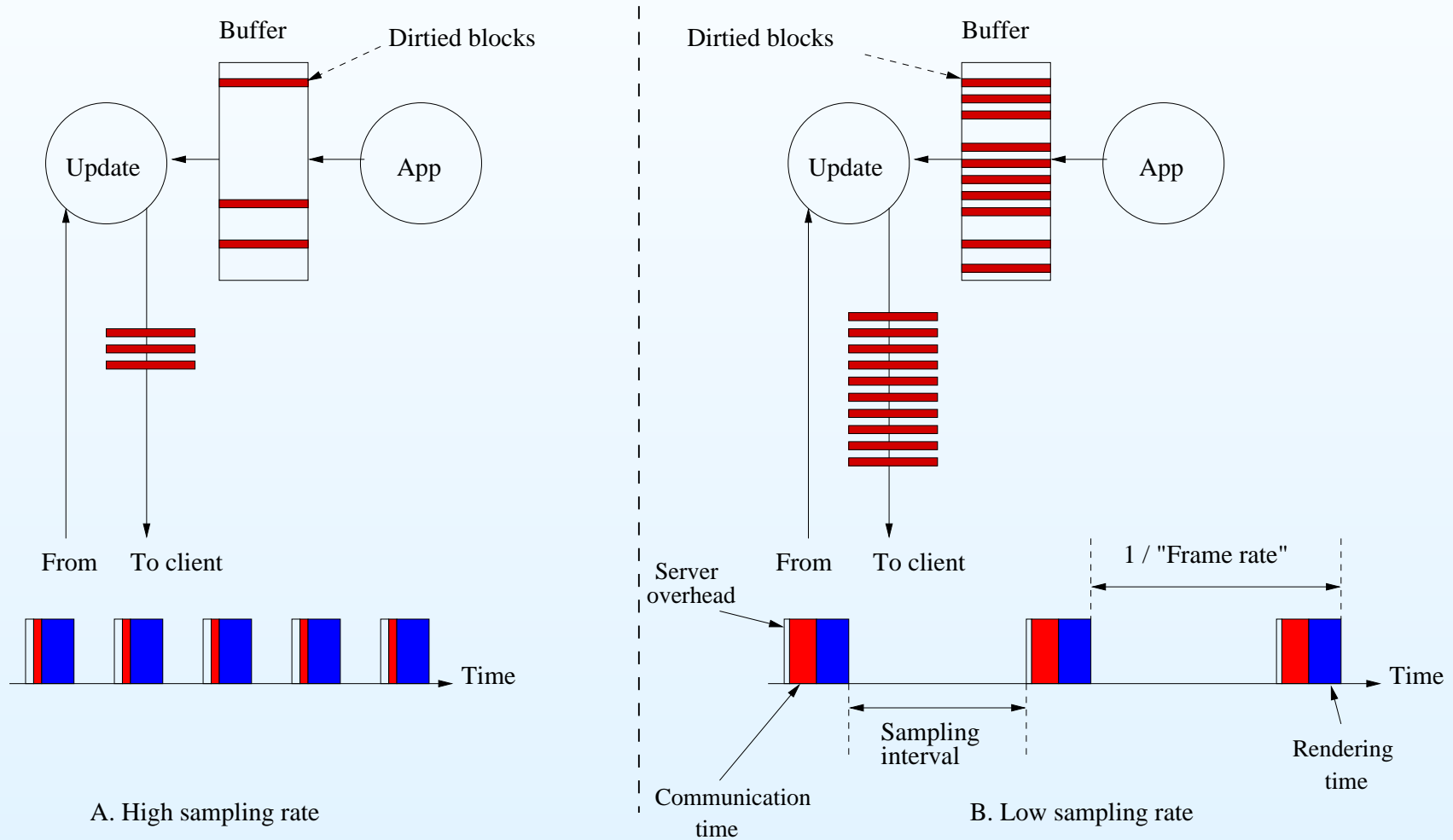
1. High-volume Fine-grain Event Handling

- Enhanced Server Architecture:



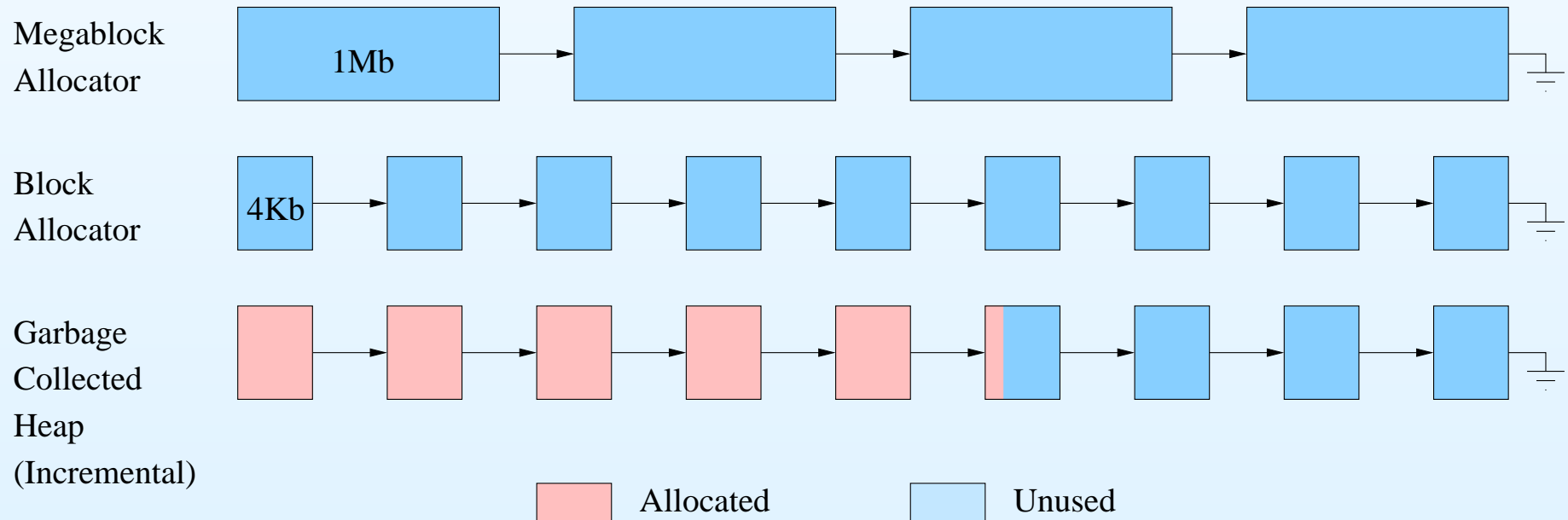
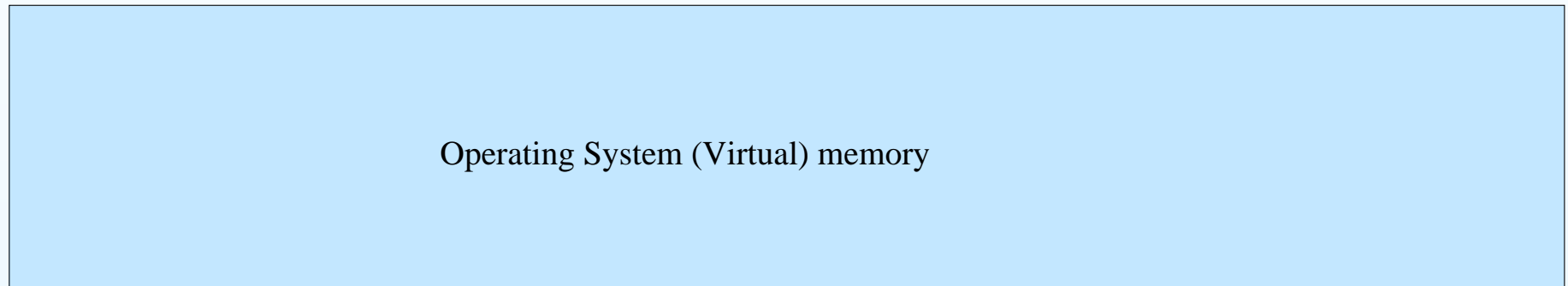
Sampling interval

- The sampling interval controls several factors...



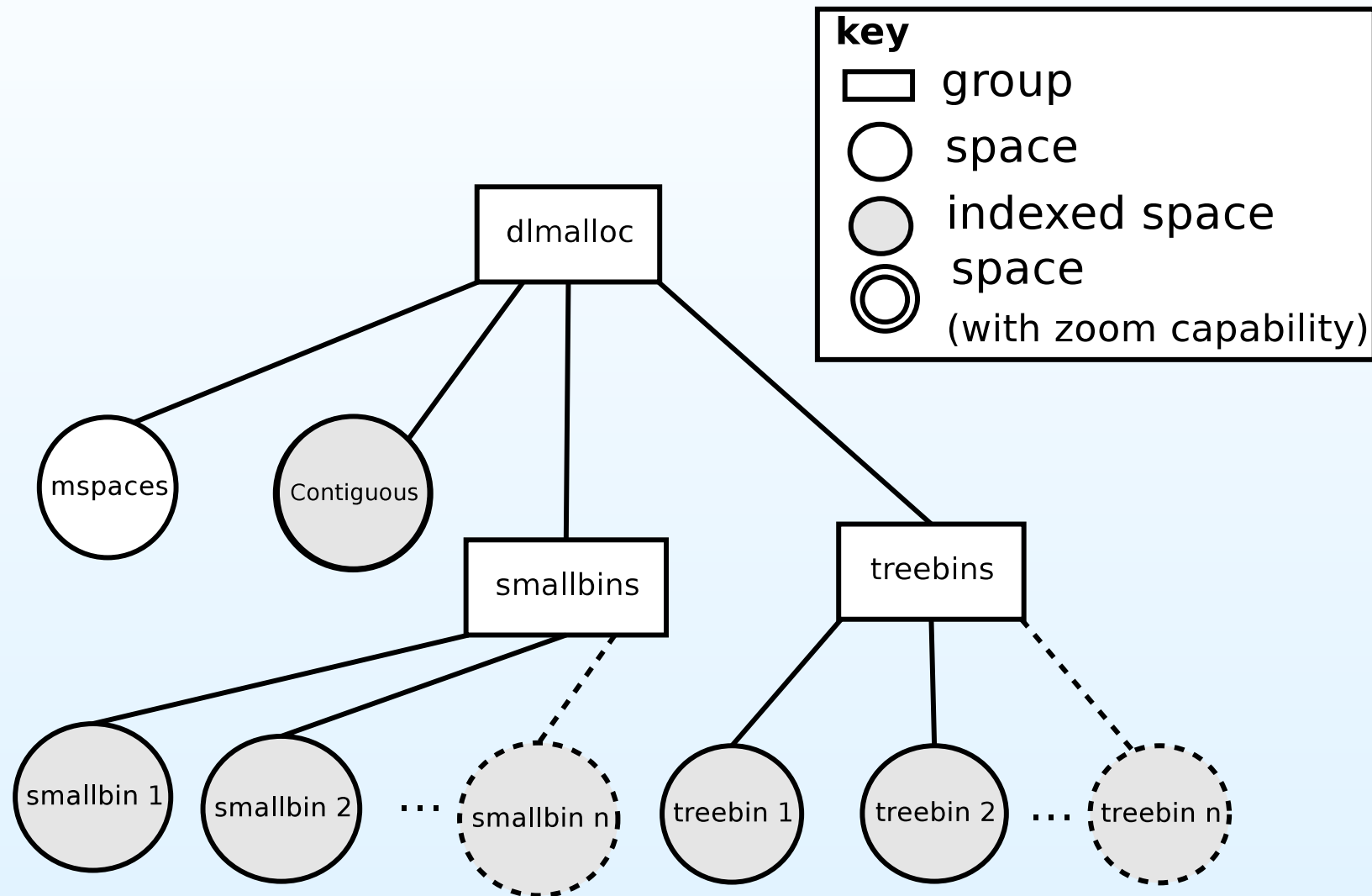
2. Data Structure Visualisation

Example: GHC (Haskell)

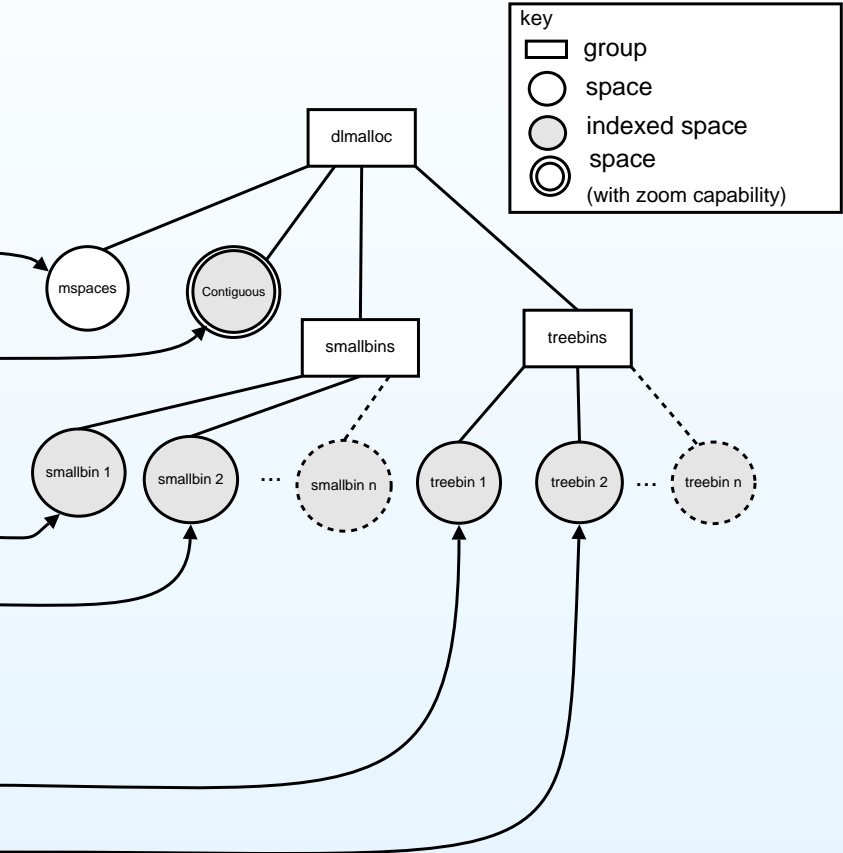
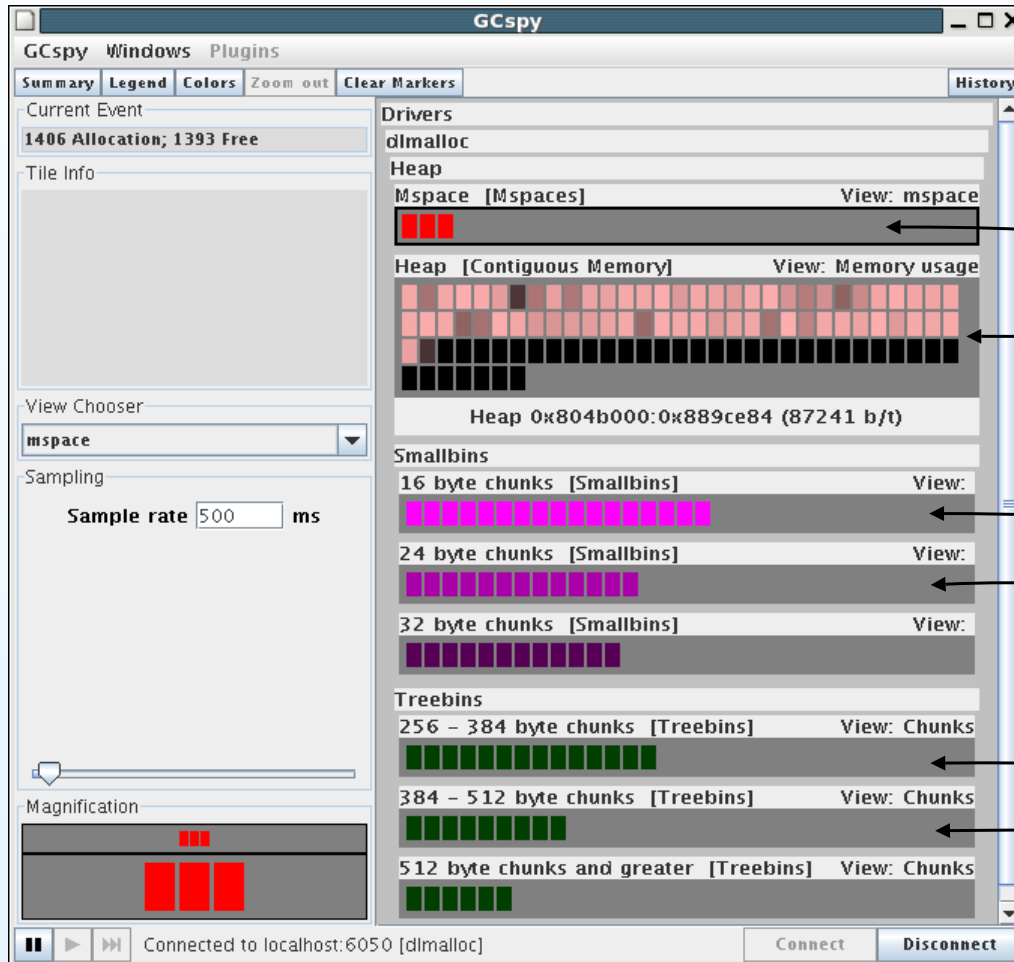


2. Data Structure Visualisation

Example: `dlmalloc`



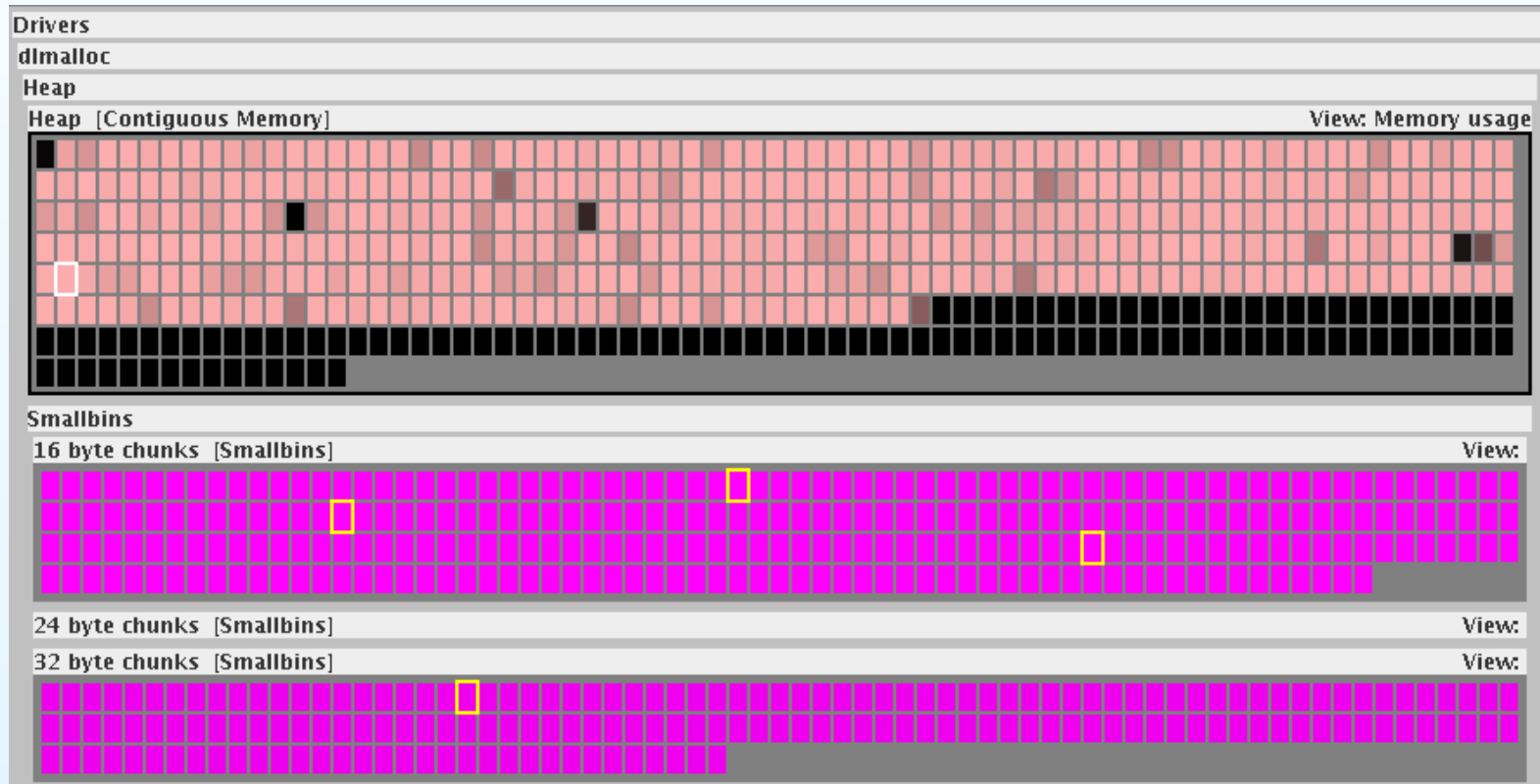
Example: dlmalloc mapping to (new) GCspy



- Note driver hierarchy

“Collapsing” Drivers

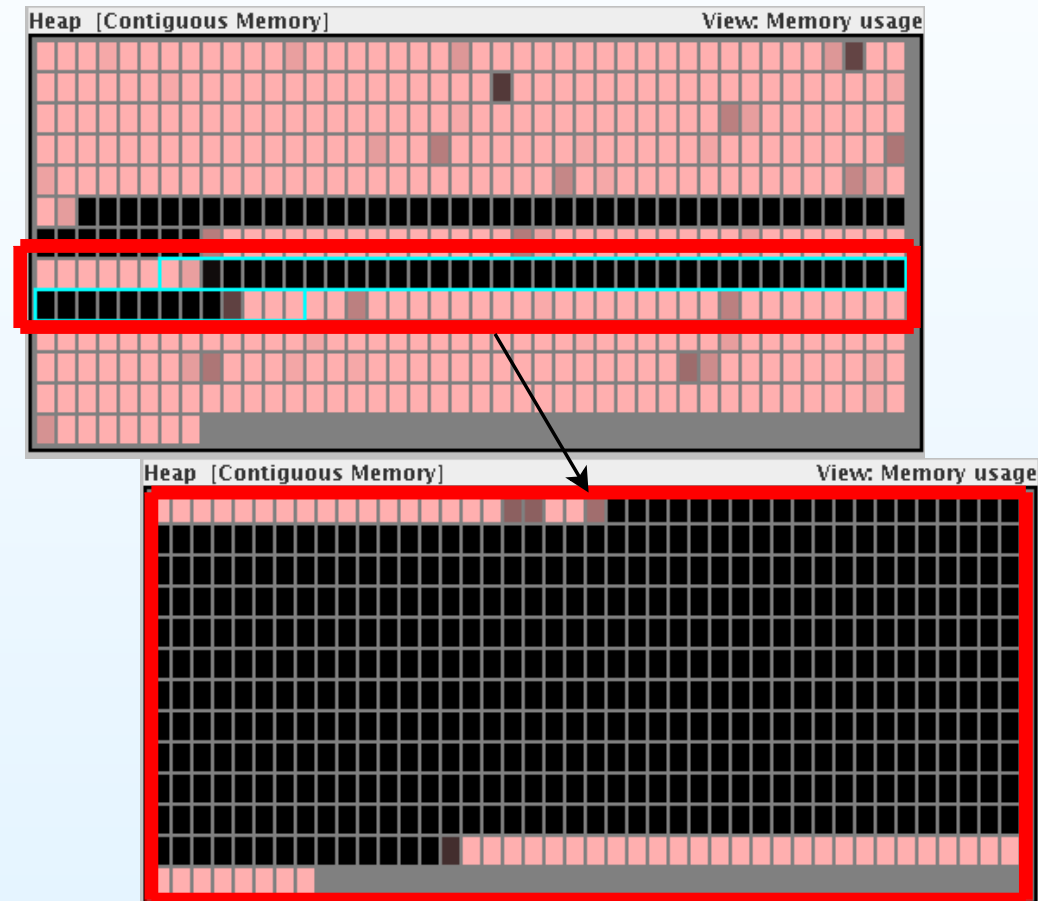
- To avoid visual information overload we can close part of visualisation:



- Note – also reduces communication

Zooming

- To focus visualisation effort, we can zoom in:



- Zooming reconfigures drivers automatically

3. Performance Debugging

- **Goal:** focus visualisation effort in response to particular phenomena, e.g.
 - Unusually large (small) allocations/frees
 - Allocations in specific memory regions
 - Signs of memory fragmentation
- We combine two mechanisms to achieve this:
 - Triggers (new) – conditions defined in terms of event *attributes*
 - Plugins – process and display specified event attributes issued after a trigger is *fired*

Triggers

- Events augmented with attributes (e.g. “allocation size” or “allocation location”);
- Trigger conditions specified in terms of these attributes;
- Compared on the server to ensure conditions are detected at the exact point at which they occur.

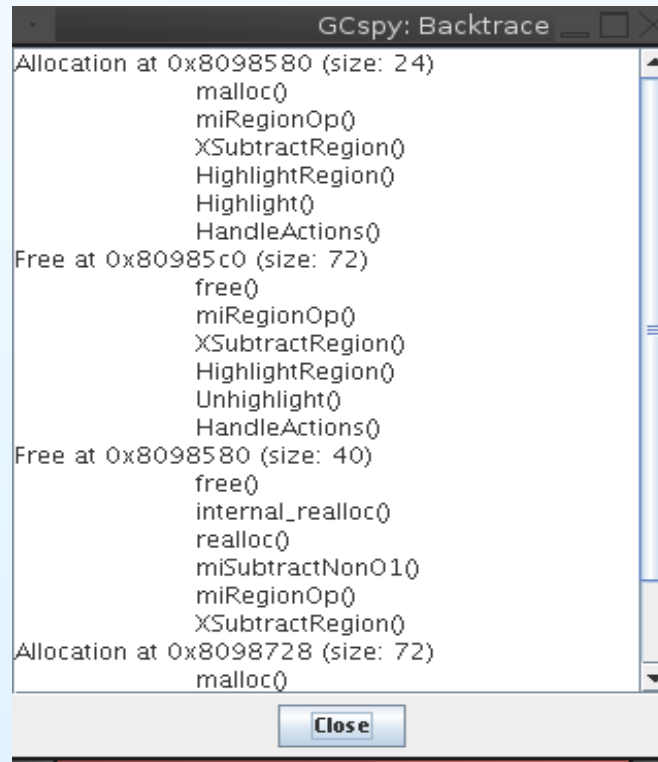
No trigger has fired

Event	Attribute	Comparison	Threshold	Plugin	Enabled
Allocation	Size	<	128	Backtrace	<input checked="" type="checkbox"/>
Allocation	Size	>	65536	Backtrace	<input checked="" type="checkbox"/>
Allocation	Location	>	1585446912	Backtrace	<input checked="" type="checkbox"/>
Allocation	Fragmentation (512, 524288)	>	2000	Backtrace	<input checked="" type="checkbox"/>

Add trigger **Close**

Plugins

- Triggers can launch specific *plugins* when they fire (e.g. backtrace, memory display...)
- When a trigger fires, enter *single-step* mode automatically
- Plugins typically open a client-side window, e.g.



```
GCspy: Backtrace
Allocation at 0x8098580 (size: 24)
  malloc()
  miRegionOp()
  XSubtractRegion()
  HighlightRegion()
  Highlight()
  HandleActions()
Free at 0x80985c0 (size: 72)
  free()
  miRegionOp()
  XSubtractRegion()
  HighlightRegion()
  Unhighlight()
  HandleActions()
Free at 0x8098580 (size: 40)
  free()
  internal_realloc()
  realloc()
  miSubtractNonO1()
  miRegionOp()
  XSubtractRegion()
Allocation at 0x8098728 (size: 72)
  malloc()
```

Close

Enhanced Framework Performance

- GCspy performance unchanged when used for GC visualisation (DaCapo benchmark suite)
- To evaluate enhanced framework, use contrived benchmark, varying:
 - Application's event generation rate
 - Sampling rate
 - Client-side visualisation (no. of tiles)

Enhanced Framework Performance

Sampling interval (ms)	Measurement	2000 tiles			8000 tiles		
		Mean inter-event time (μs)			Mean inter-event time (μs)		
		0	50	100	0	50	100
100	No. dirty tiles	1347	566	354	4932	1601	978
	Comm. time (ms)	14.62	5.04	3.33	151.09	39.87	24.01
	Effective frame rate	2.59	2.65	2.63	0.43	0.49	0.52
200	No. dirty tiles	1585	938	629	6953	2897	1836
	Comm. time (ms)	22.62	9.14	5.89	200.81	78.03	44.42
	Effective frame rate	2.07	2.08	2.10	0.40	0.42	0.47
500	No. dirty tiles	1893	1643	1229	7784	5770	3926
	Comm. time (ms)	38.17	17.74	10.81	292.20	153.45	102.30
	Effective frame rate	1.32	1.28	1.29	0.35	0.42	0.39

Summary and Conclusions

Key contributions

- Built-in facility for periodic incremental client updates
- Targeted hierarchical visualisation of data structures
- Additional performance debugging aids (triggers, backtraces...)
- A complete integration with `dmalloc`

Conclusions

- Additional features come “for free” for existing GCspy users
- Client-side visualisation is the bottleneck
- Smooth visualisations are possible, even for event rates of $O(10^5)/s$

Future work

- Improve visualisation of *dmalloc*'s treebins
- Enhance the trigger heuristics engine
 - Trigger specification
 - More plugins
- Incremental rendering
- Integrate with GHC and ECLiPSe