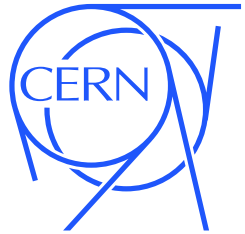


Memory saving techniques

Jakob Blomer

many thanks to Axel Naumann, Peter Hristov,
Nathalie Rauschmayr, Eva Sicking, Danilo Piparo

October 10, 2012

Compressed Memory

Assumption Plain old event parallelism / single threaded software
Memory consumption has to decrease in order to make use of all the cores in a machine

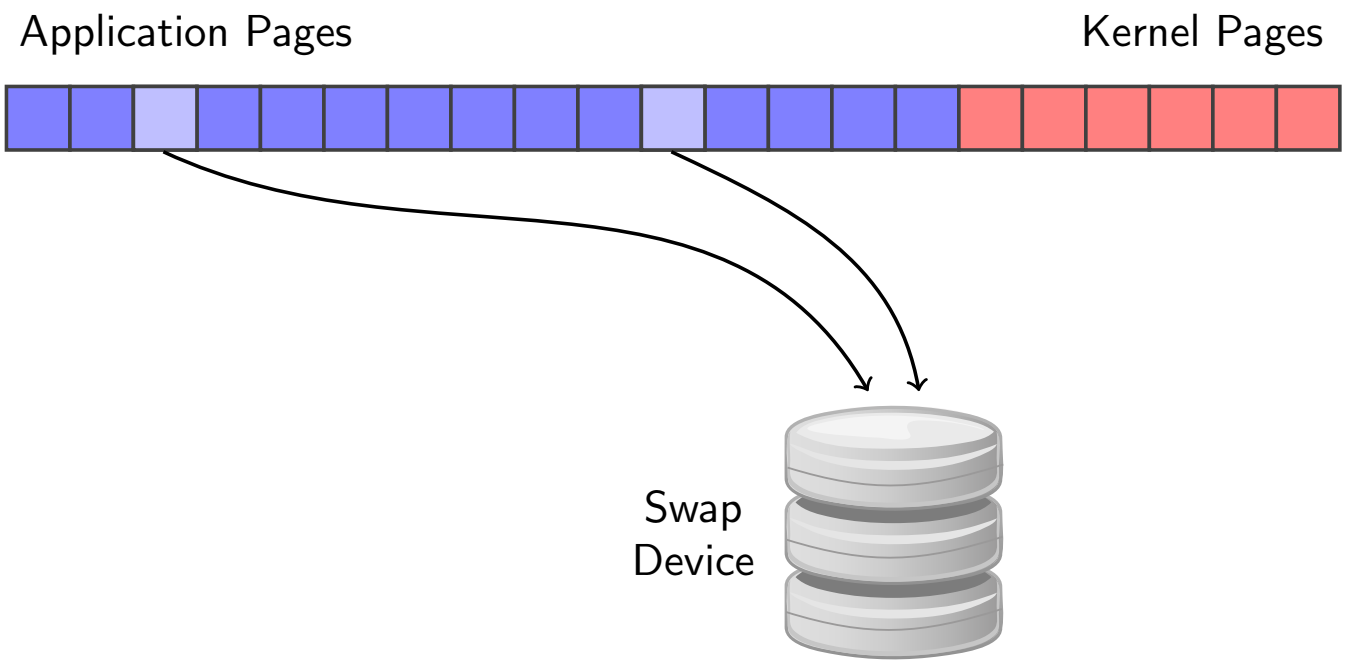
Idea Compress (part of the) memory
Trade CPU time for memory consumption
Drain performance by overcommitting

Example Enable Hyper-Threading if memory consumption falls below 1 GB

Side effect Improve existing software

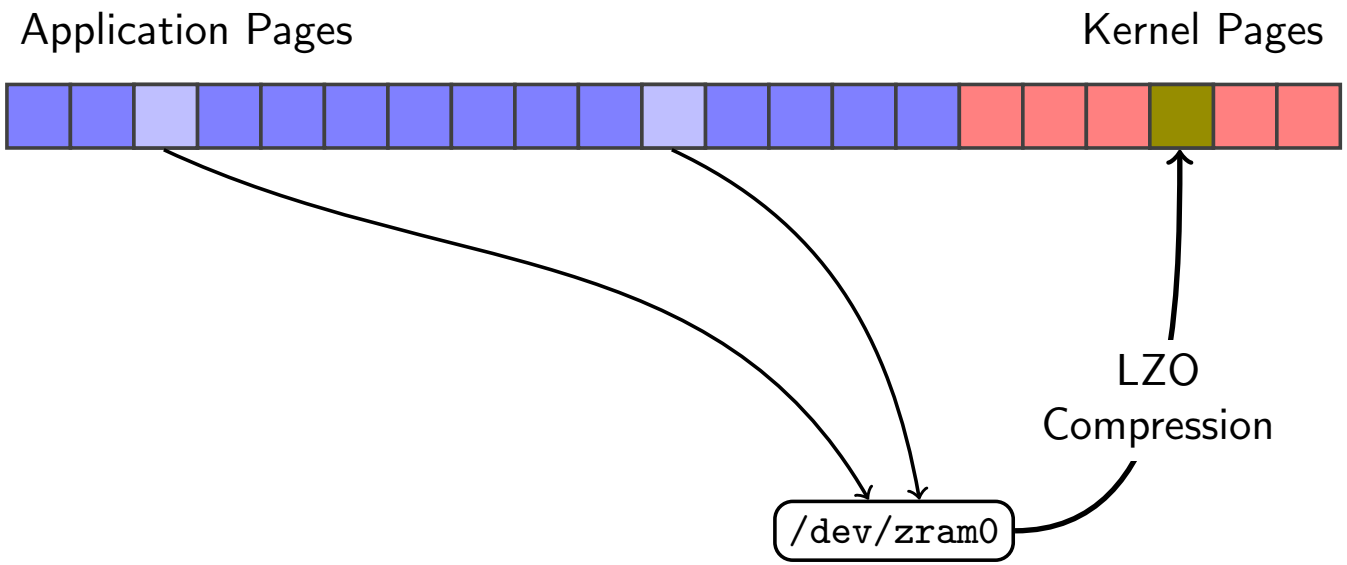
Kernel-compressed Memory – Principle

- Kernel module *compcache* / *zram* provides a virtual block device for swapping
- Originally developed for "small" devices (Netbooks, phones, ...)
- Part of Kernel >= 2.6.34, can be compiled for SLC6 (with drawbacks)
- Can be easily set up inside a virtual machine



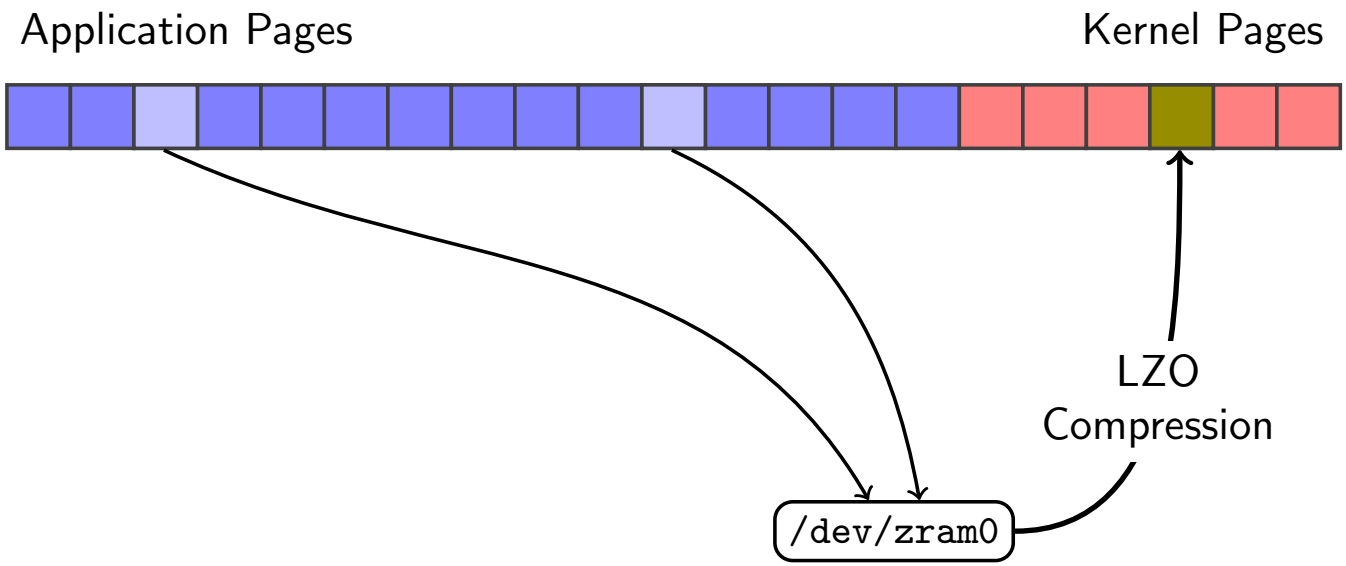
Kernel-compressed Memory – Principle

- Kernel module *compcache* / *zram* provides a virtual block device for swapping
- Originally developed for "small" devices (Netbooks, phones, ...)
- Part of Kernel >= 2.6.34, can be compiled for SLC6 (with drawbacks)
- Can be easily set up inside a virtual machine



Kernel-compressed Memory – Principle

- Kernel module *compcache* / *zram* provides a virtual block device for swapping
- Originally developed for "small" devices (Netbooks, phones, ...)
- Part of Kernel >= 2.6.34, can be compiled for SLC6 (with drawbacks)
- Can be easily set up inside a virtual machine



Change in strategy: not swap at all $\mapsto$ swap whenever possible (`/proc/sys/vm/swappiness`)

Kernel-compressed Memory and cgroups

The system memory pressure and the swappiness are not fine-grained enough handles for measurements

Linux cgroups allow to put the application into a limited memory container:

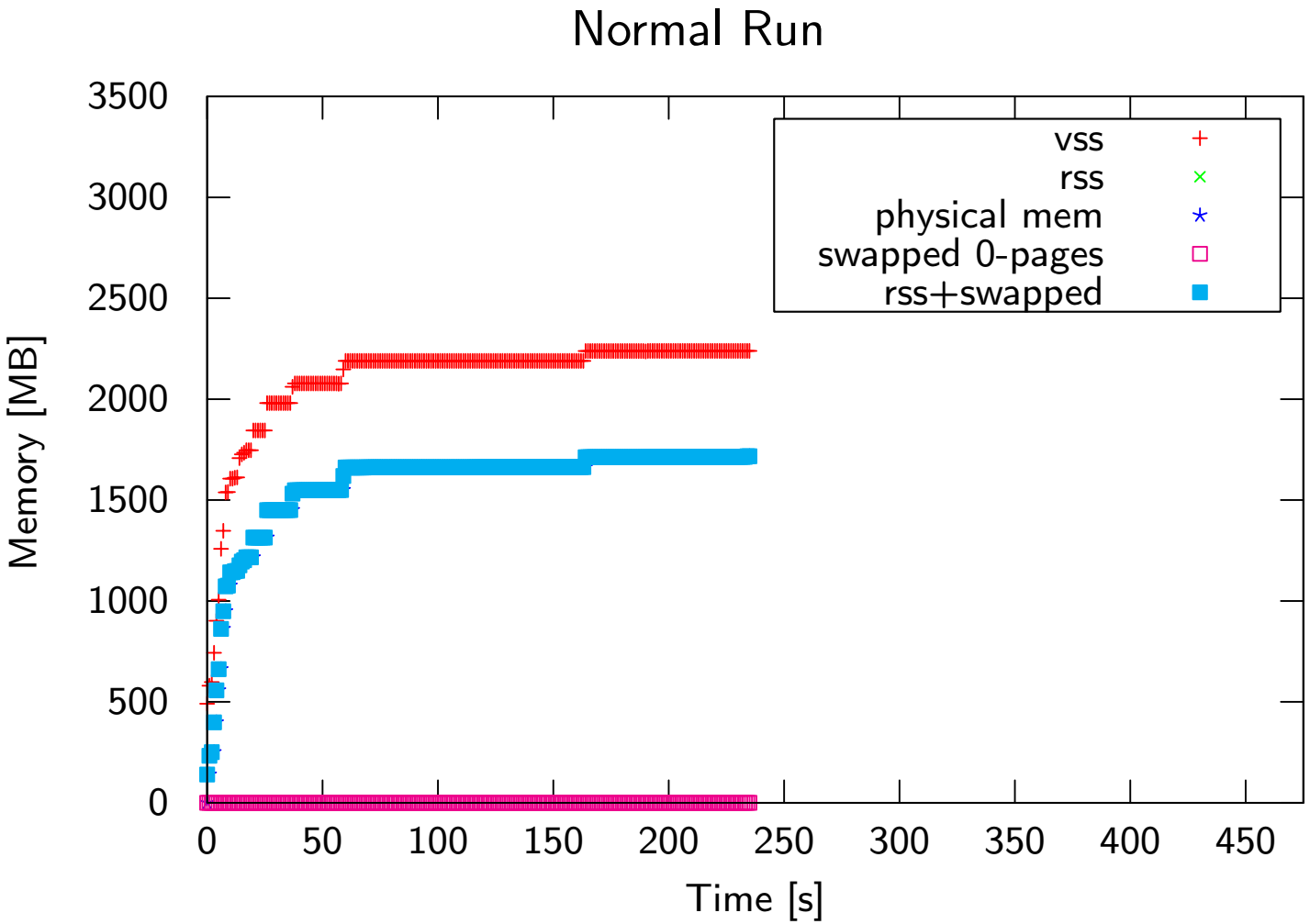
```
$ mkdir /sys/fs/cgroup/memory/restricted
$ echo $((150*1024*1024)) > \
    /sys/fs/cgroup/memory/restricted/memory.limit_in_bytes
$ echo $PID > /sys/fs/cgroup/memory/restricted/tasks
```

Problem: File system buffers for process' files are accounted for.

Here: files are pre-staged

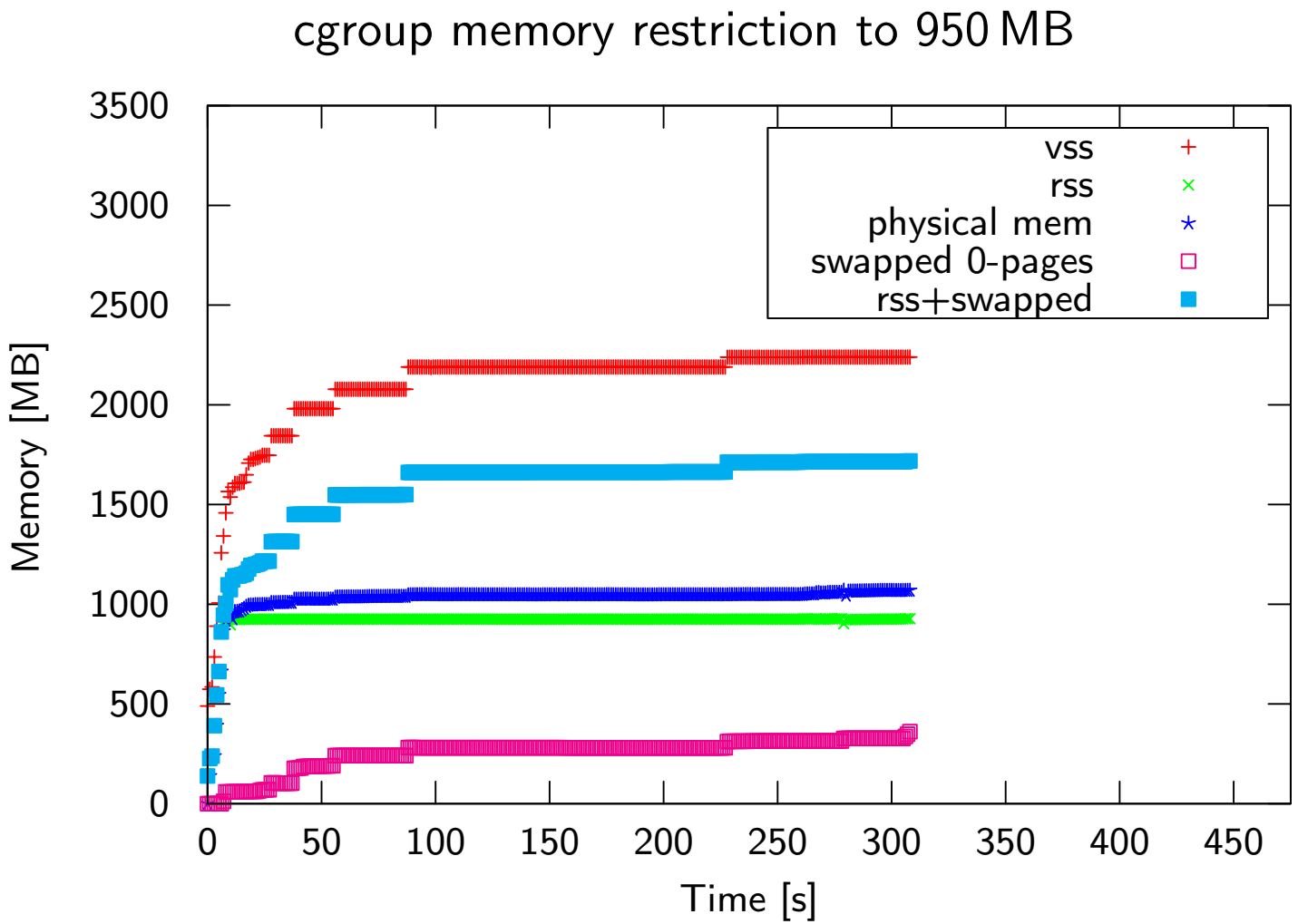
Kernel-compressed Memory – Figures

AliRoot reconstruction of 2 simulated pp Events (v5-04-25-AN)



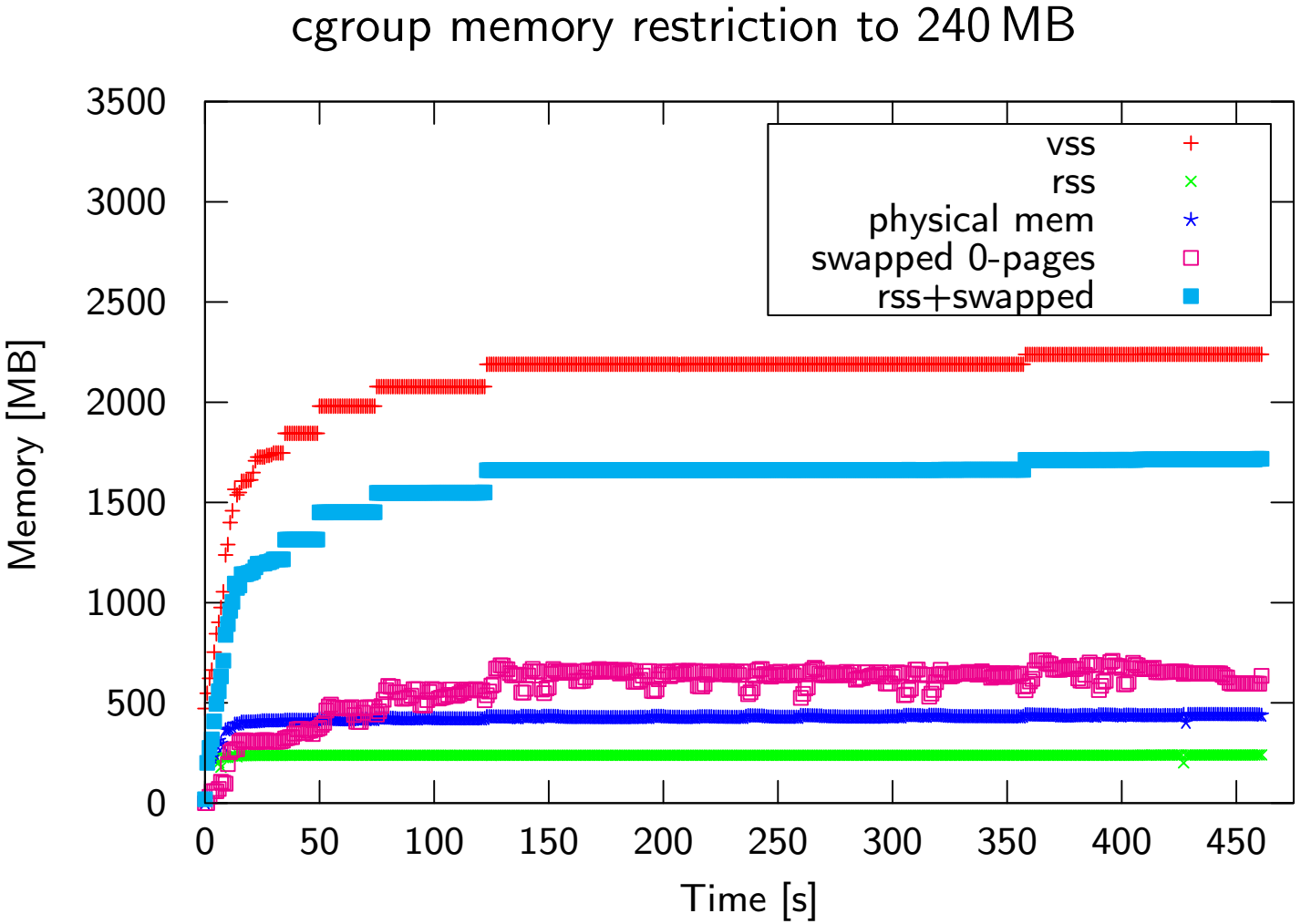
Kernel-compressed Memory – Figures

AliRoot reconstruction of 2 simulated pp Events (v5-04-25-AN)



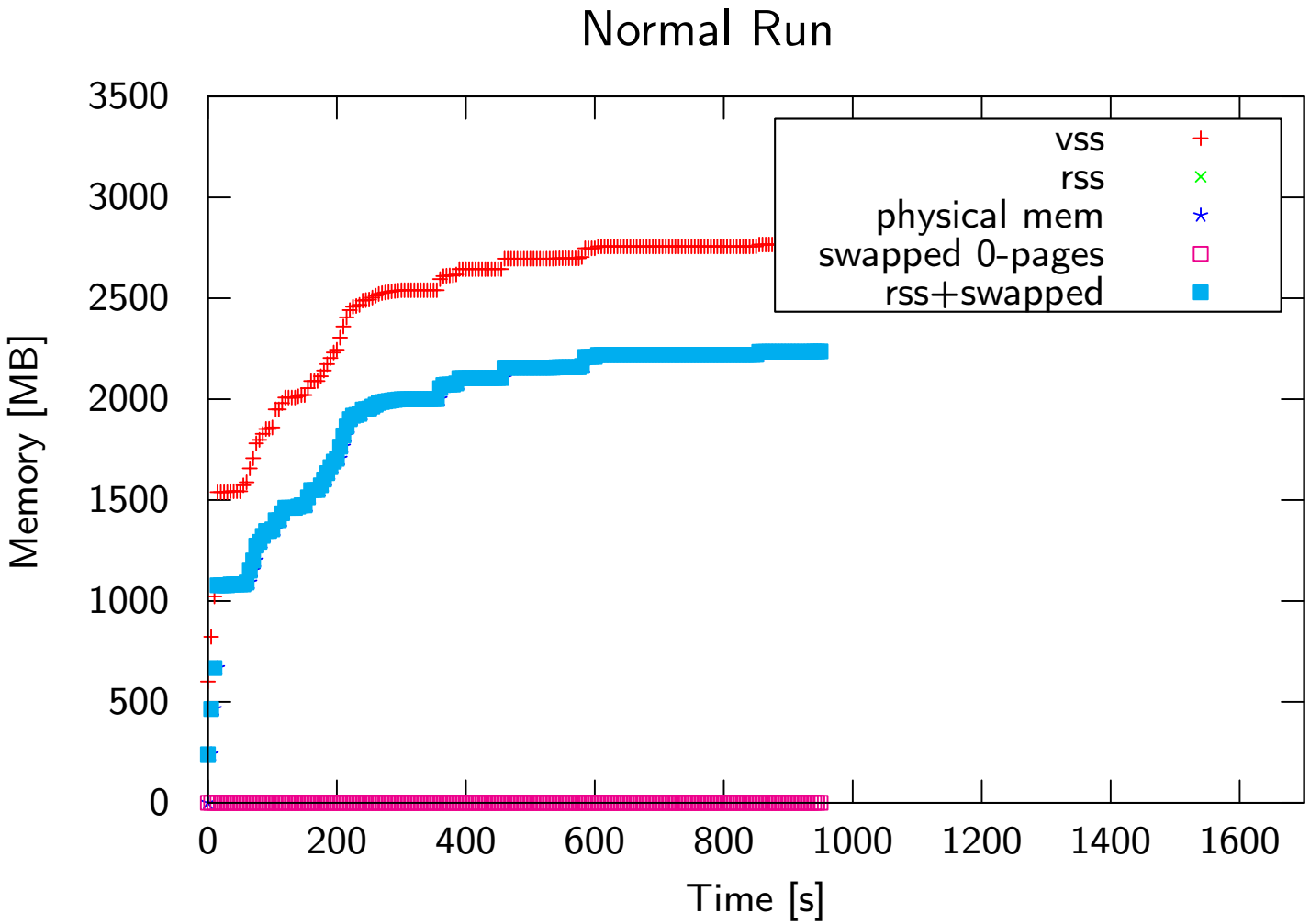
Kernel-compressed Memory – Figures

AliRoot reconstruction of 2 simulated pp Events (v5-04-25-AN)



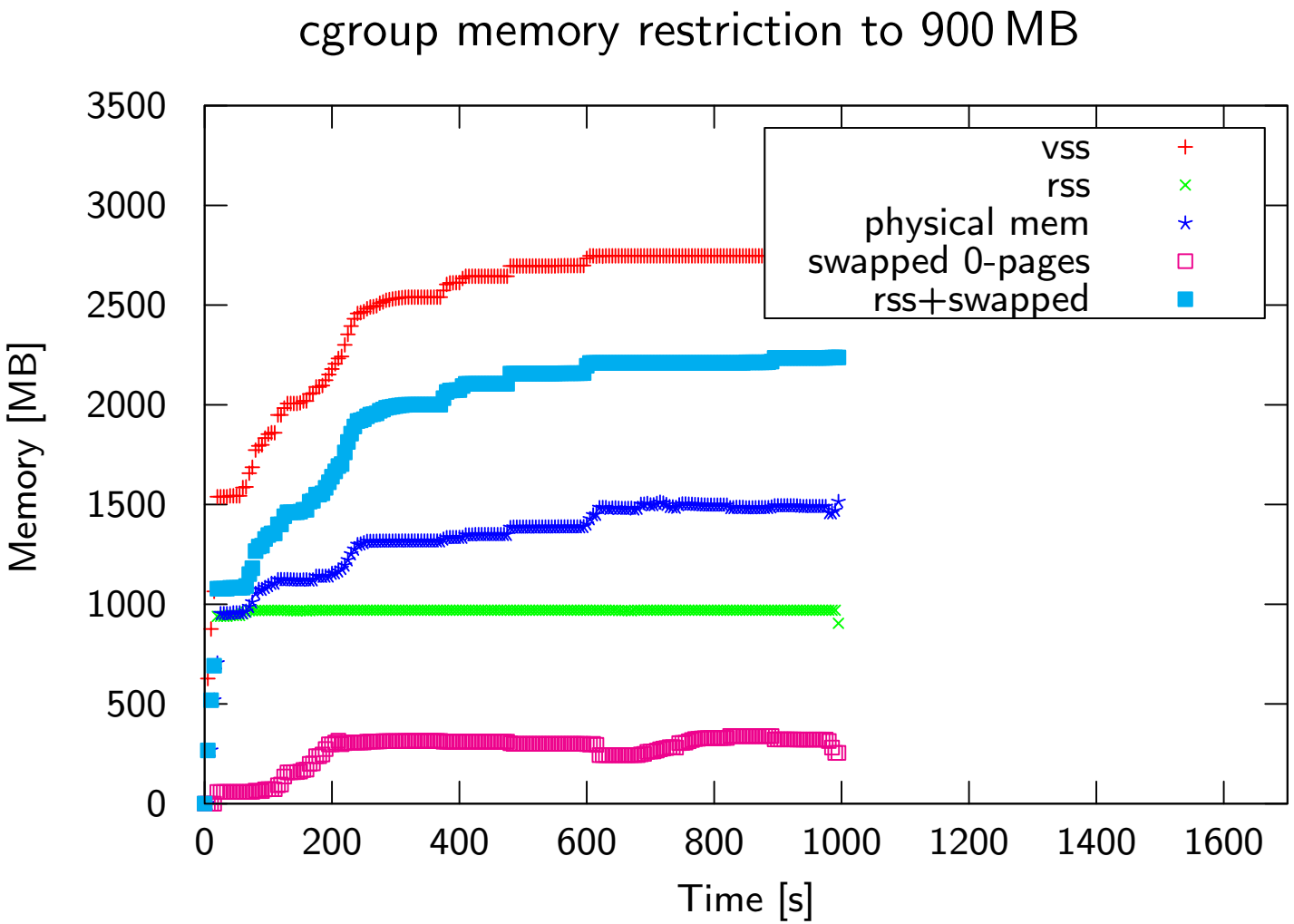
Kernel-compressed Memory – Figures

AliRoot reconstruction of 10 PbPb Events (v5-03-62-AN)



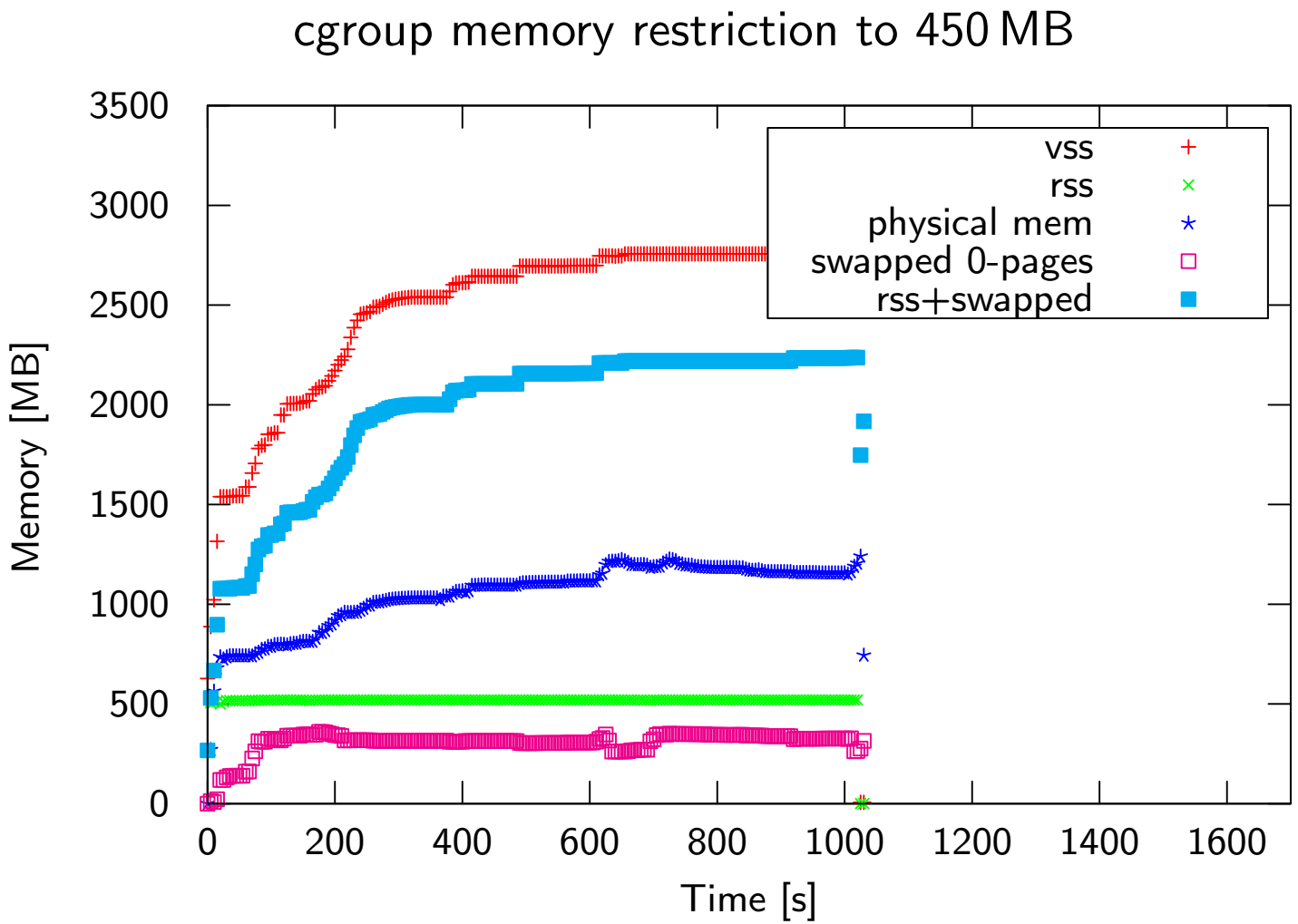
Kernel-compressed Memory – Figures

AliRoot reconstruction of 10 PbPb Events (v5-03-62-AN)



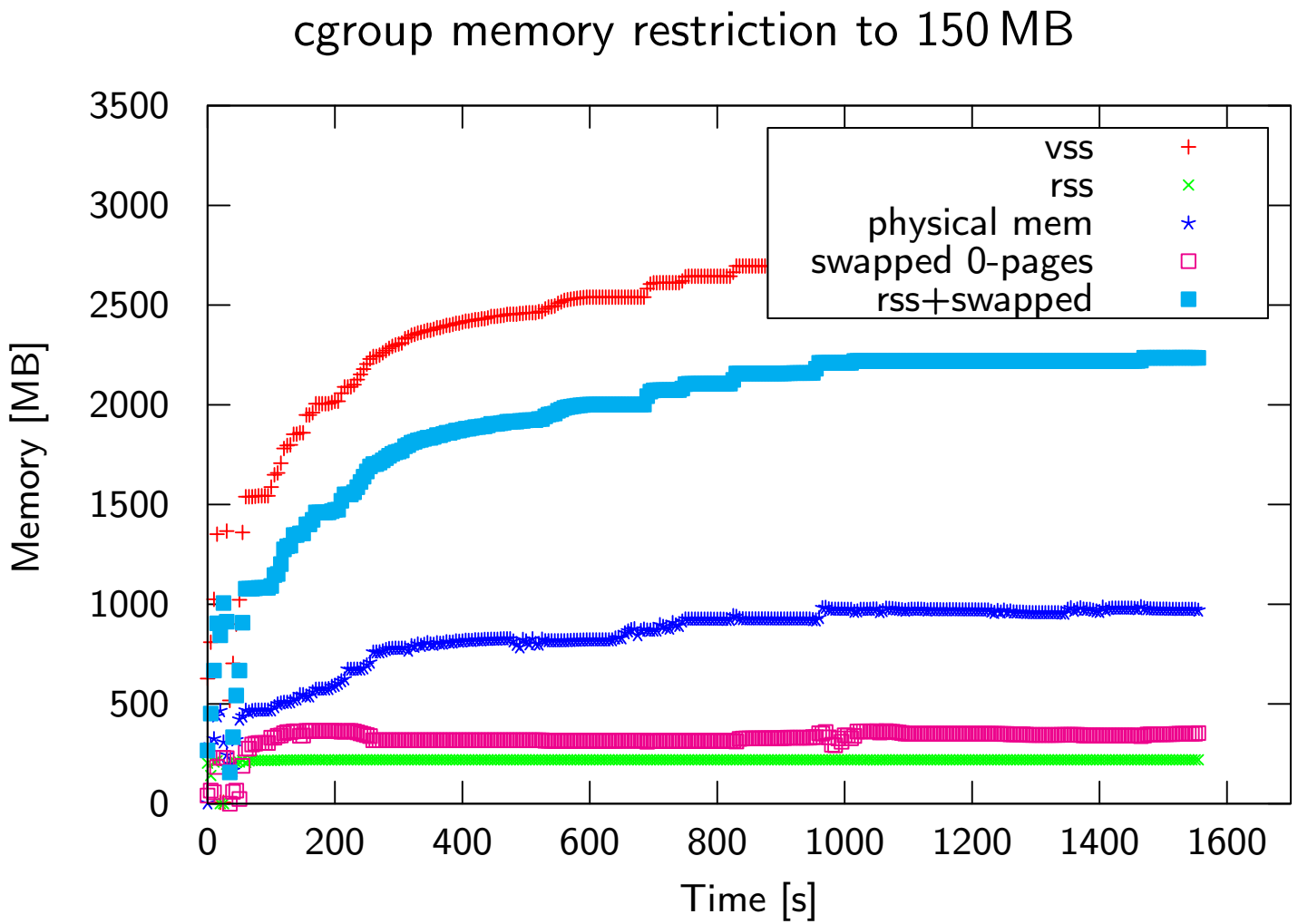
Kernel-compressed Memory – Figures

AliRoot reconstruction of 10 PbPb Events (v5-03-62-AN)



Kernel-compressed Memory – Figures

AliRoot reconstruction of 10 PbPb Events (v5-03-62-AN)



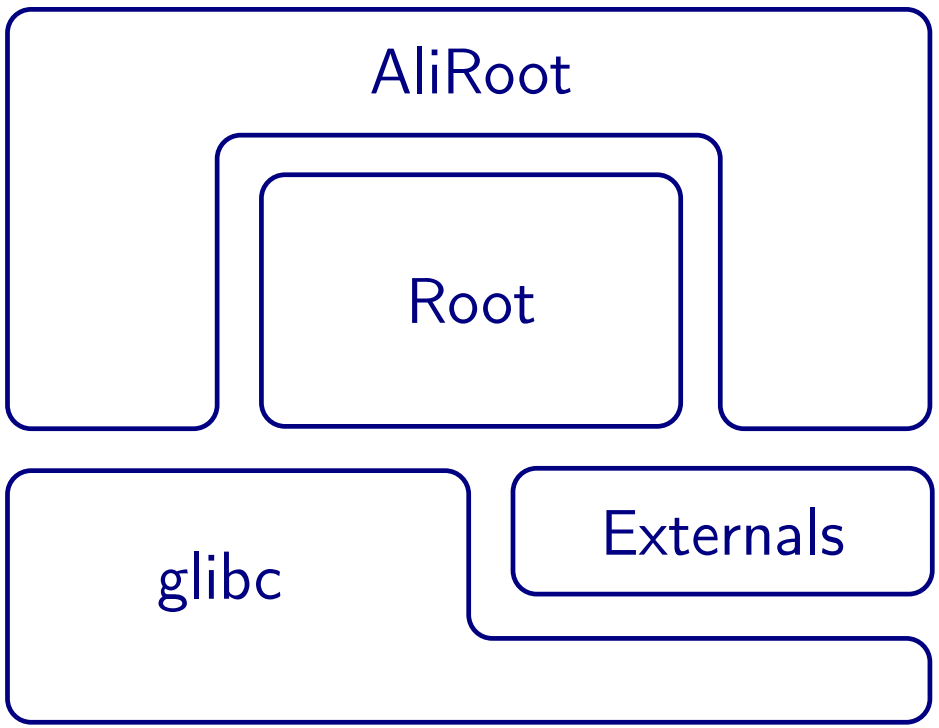
Zero Pages

X-Check: scan through a core dump of the application

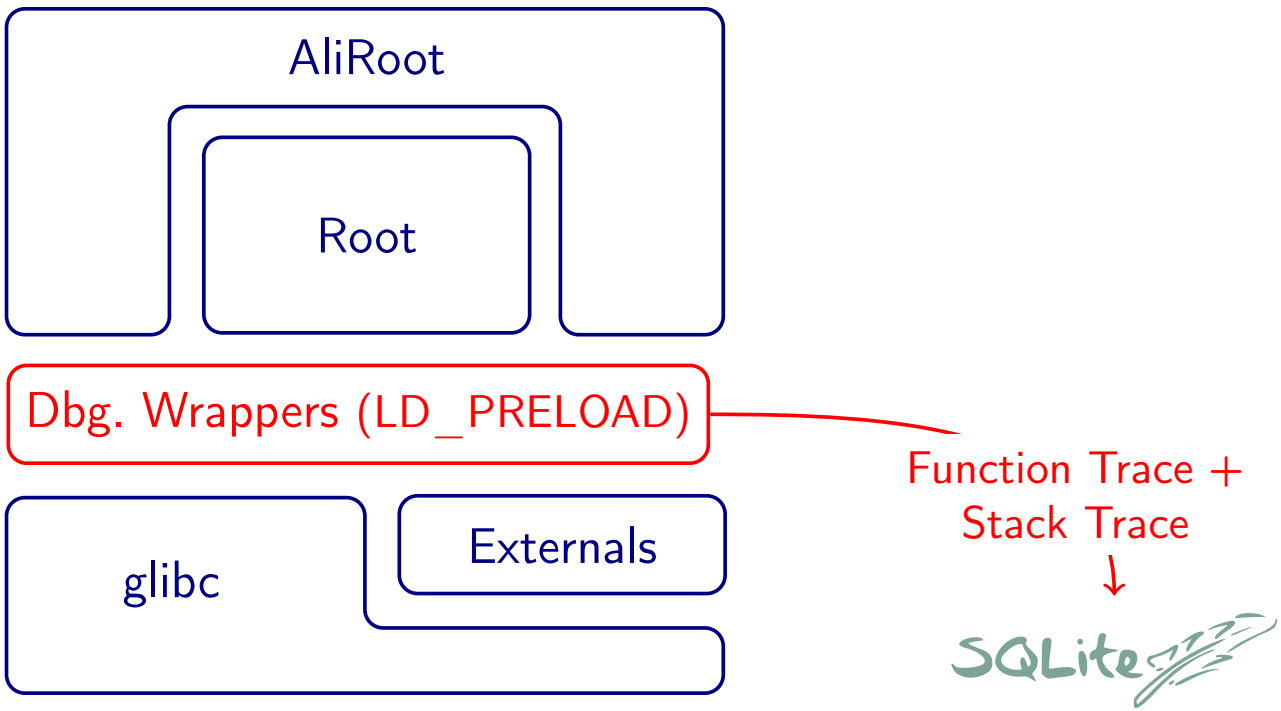
Can we get rid of these hundreds of Megabytes of continuous zeros?

- No change by using automatic garbage collection (Boehm's GC)
- Zero pages in LHCb DaVinci: $\approx$ 700 MB out of 2.3 GB
- Zero pages in CMS reconstruction
  - 180 MB out of 900 MB without output
  - 280 MB out of 1.4 GB with output

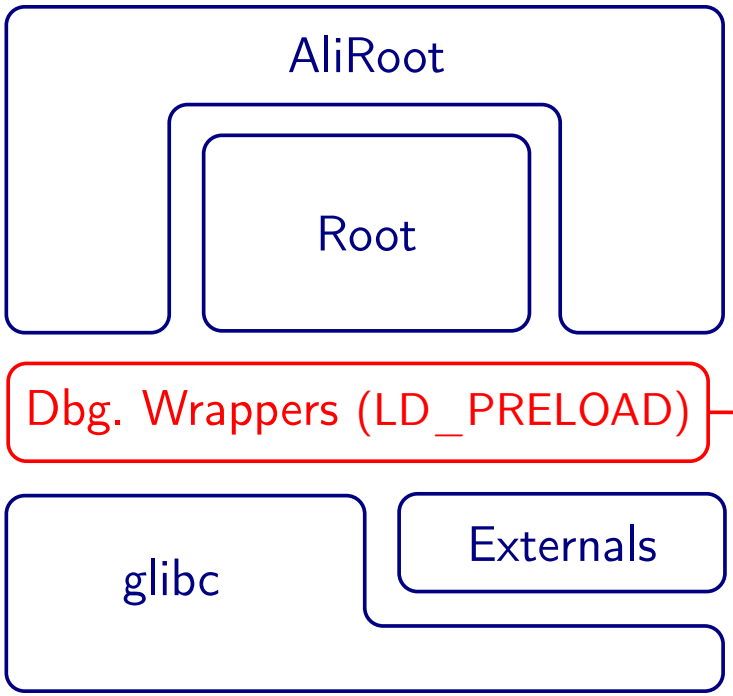
- Intercepting low-level library calls
- Valgrind-like approach (not profiling)



- Intercepting low-level library calls
- Valgrind-like approach (not profiling)



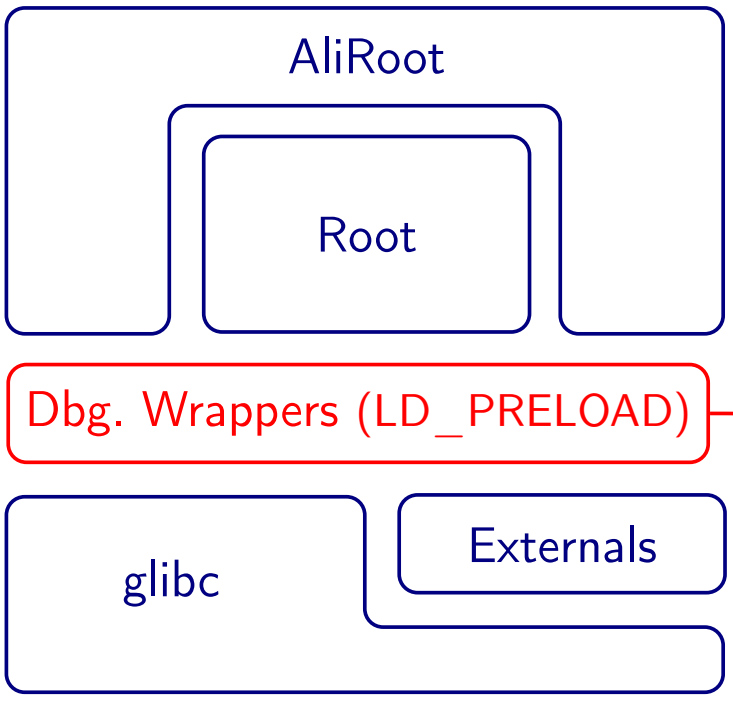
- Intercepting low-level library calls
- Valgrind-like approach (not profiling)




```
malloc, calloc, realloc
free
mmap
read
memset
```

Function Trace +
Stack Trace
↓
SQLite

- Intercepting low-level library calls
- Valgrind-like approach (not profiling)




```
malloc, calloc, realloc
free
mmap
read
memset
```

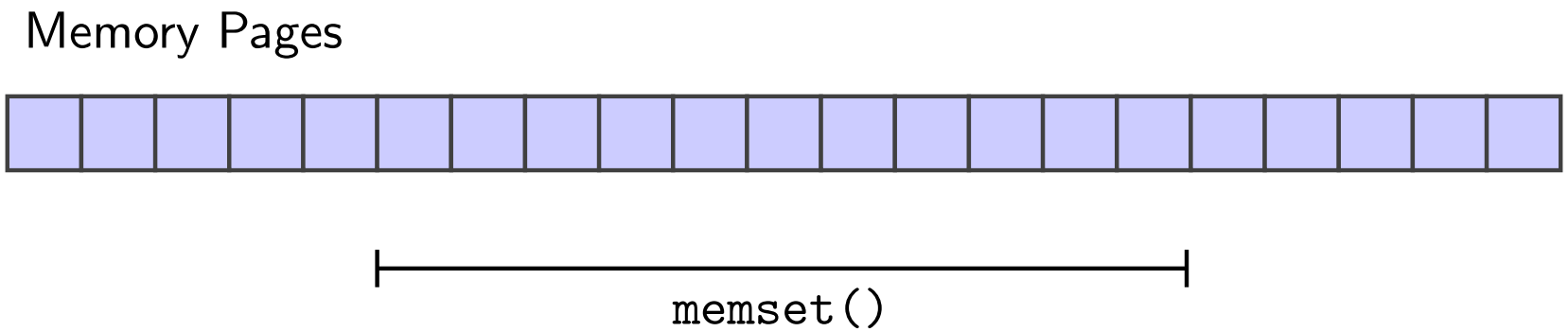
Function Trace +
Stack Trace
↓
SQLite

Idea: Inspect `memset()` calls >4 kB

Finding Unused Pages

Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

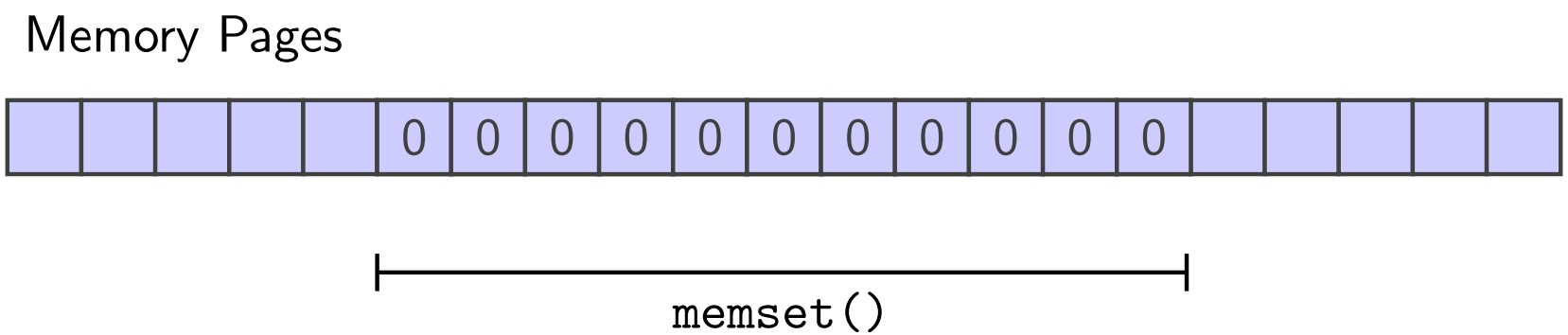
Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



Finding Unused Pages

Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

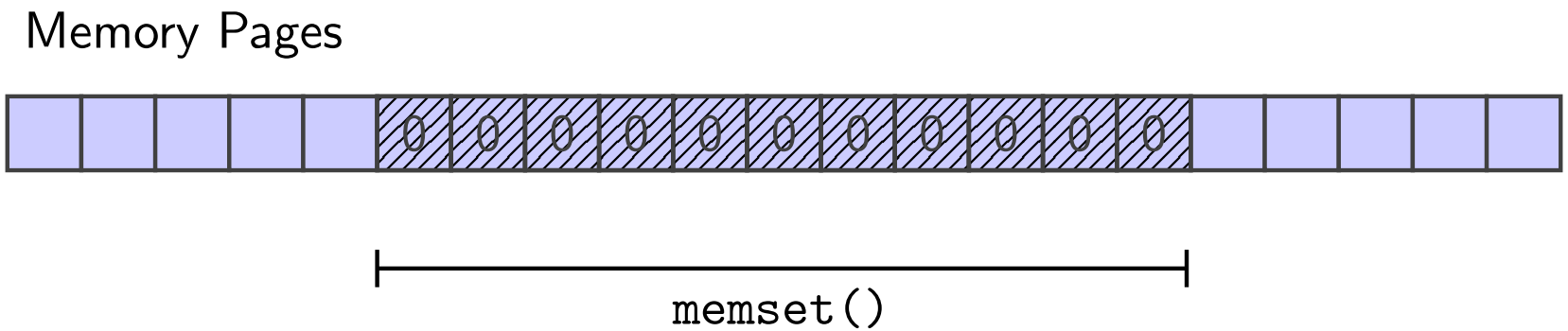
Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



Finding Unused Pages

Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

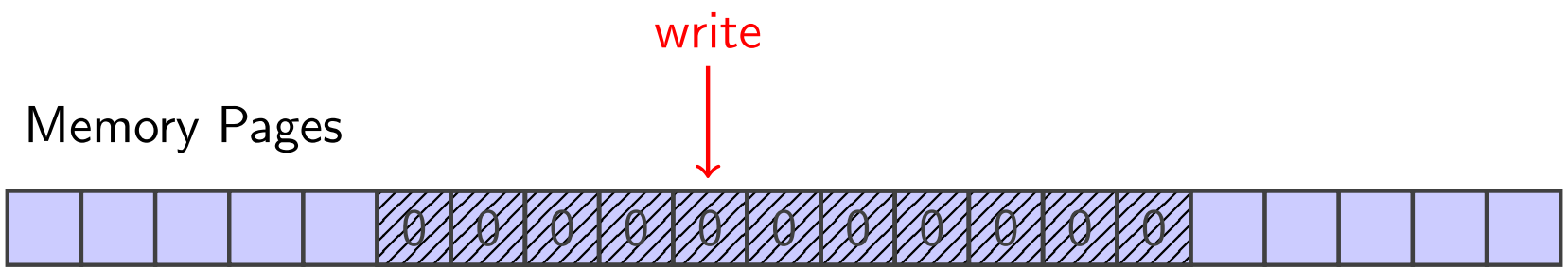
Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



Finding Unused Pages

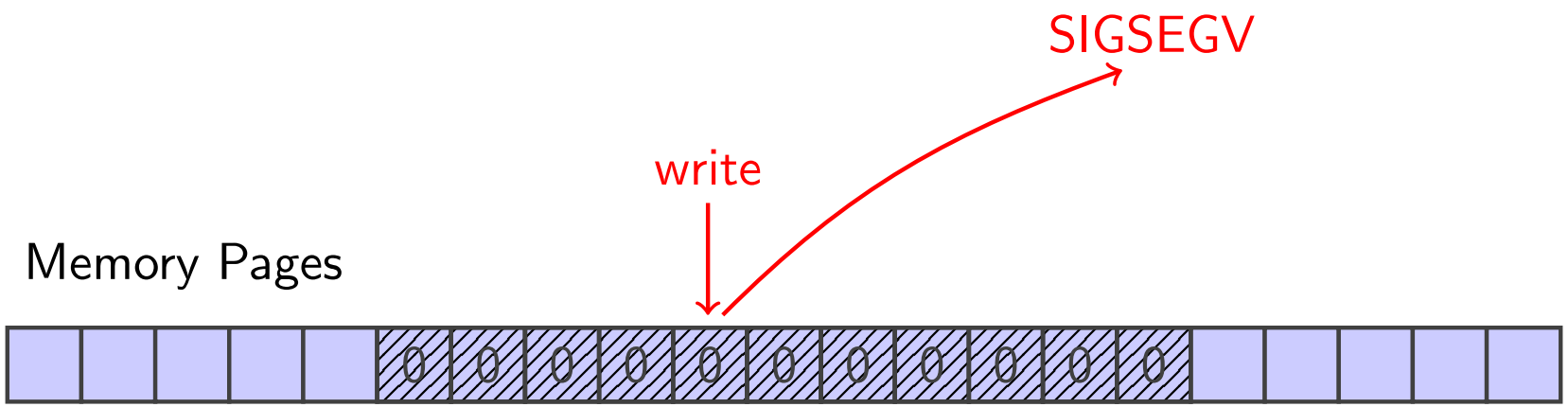
Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



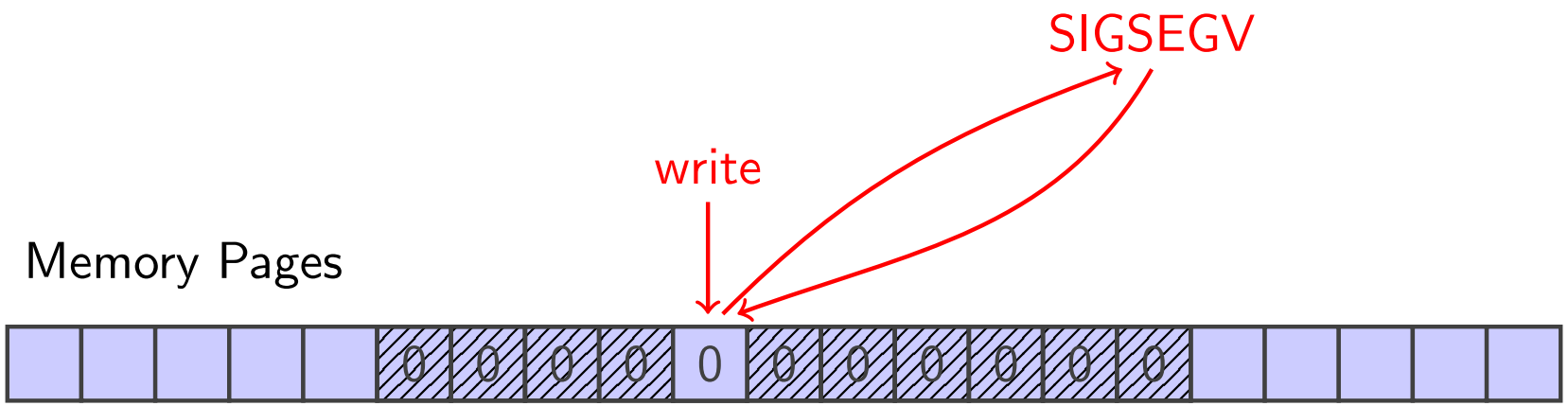
Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

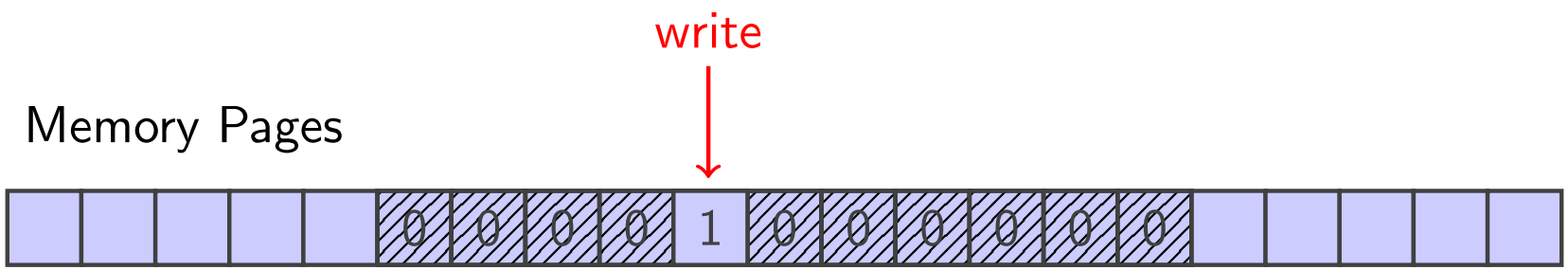
Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



Finding Unused Pages

Problem: How to distinguish the legitimate `memset()` calls from the wasteful ones

Approach: `memset()` $\mapsto$ `memset()` + `mprotect()` + SIGSEGV handler



Dead pages (AliRoot reco)

- $\approx$ 45 000 – 50 000 zero pages traced back to source code
- Breaks down to half a dozen memsets with high impact
- No hits after detector initialization
- Scattered over uses of TClonesArray

Remaining zero pages

- Excluded: `read()`, `mmap()`
- Other externals (libz, . . .) (unlikely)
- Measurement uncertainties at memset boundaries
- Only literal `memset()` covered, standard constructors:

```
int *a =
  new int[1024*1024]();
```

Open Issues

Intrinsic Improvements

Identifying the source of the remaining zero pages:

1. Track `malloc()` ($\geq 4\,\mathrm{kB}$) and `free()`
2. Stop the program after initialization (e. g. SIGQUIT)
3. Walk the heap and connect zero blocks to malloc stack traces

Automatic Memory Compaction

- How to choose zram parameters in an opportunistic way?
- How to account for file system buffers?
- Don't compress but only eliminate zero pages?
  (e. g. KSM, copy-on-write)