Monitoring Memory Usage Based on Kernel Trace Events

Progress Report Meeting

December 12, 2016

Houssem Daoud **Michel Dagenais**

École Polytechnique de Montréal

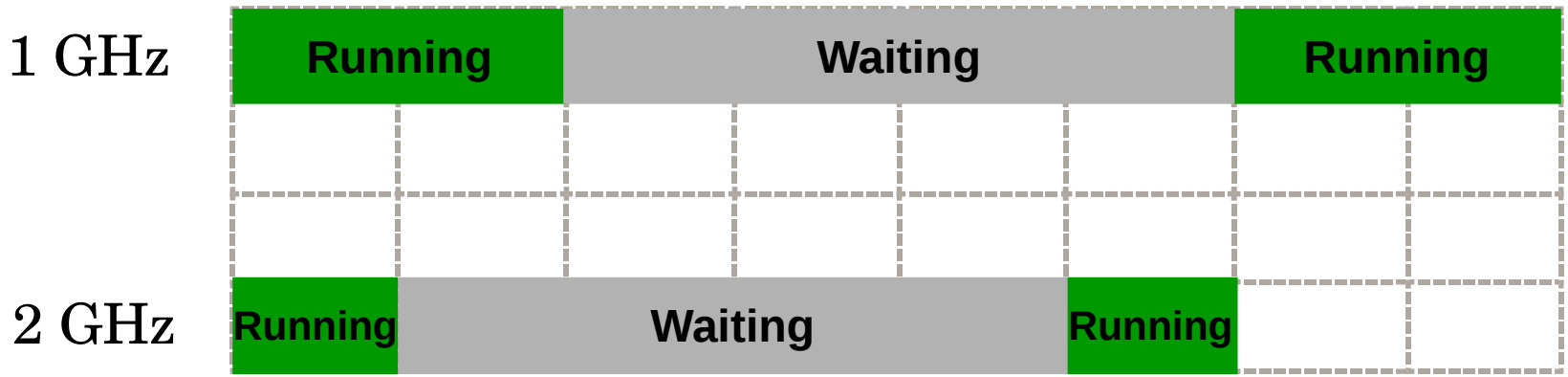
Laboratoire **DORSAL**

Agenda

- **Introduction and objectives**
- Literature review
- Memory usage monitoring
- Memory leak detection
- Conclusion

Introduction

- Using faster processors doesn't always improve the performance of the applications



A 2x faster processor gives an acceleration rate of 1.3

Introduction

Reasons of blocking:

- I/O operations (disk, network, etc.)
- Memory Access

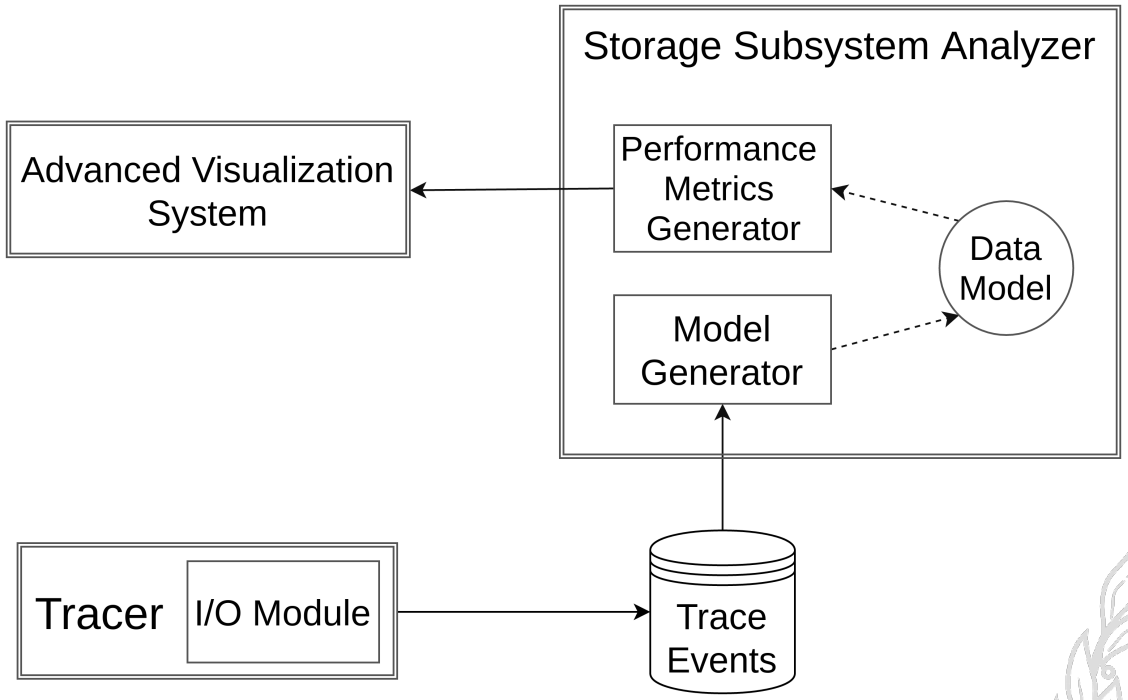
- The operating system uses many mechanisms to improve the performance of those peripherals: Disk scheduler, memory allocator, caching.

Our goal is to provide tools and algorithms able to evaluate the efficiency of those mechanisms and to detect performance problems.

Introduction

In a previous work, we proposed a tool that provides a low-level analysis of the storage subsystem based on Kernel traces.

Research Paper: "Recovering Disk Storage Metrics from Low-level Trace Events" (Submitted)



Introduction

Analyzing memory management mechanisms using tracing present the following challenges:

- The high frequency of memory operations makes the trace file huge

- It is not possible to target a single process using basic filtering techniques since physical memory releasing is usually done out of context

- Tracing can contribute in polluting the memory of the system

Introduction

RQ : Is it possible to analyze memory management mechanisms using tracing without perturbing the normal behavior of the system ?

Objectives

- Study the different memory management mechanisms
- Instrument the Kernel to get the required information
- Provide filtering and aggregation mechanisms to reduce the frequency of events.
- Generate metrics and visualization from the trace file

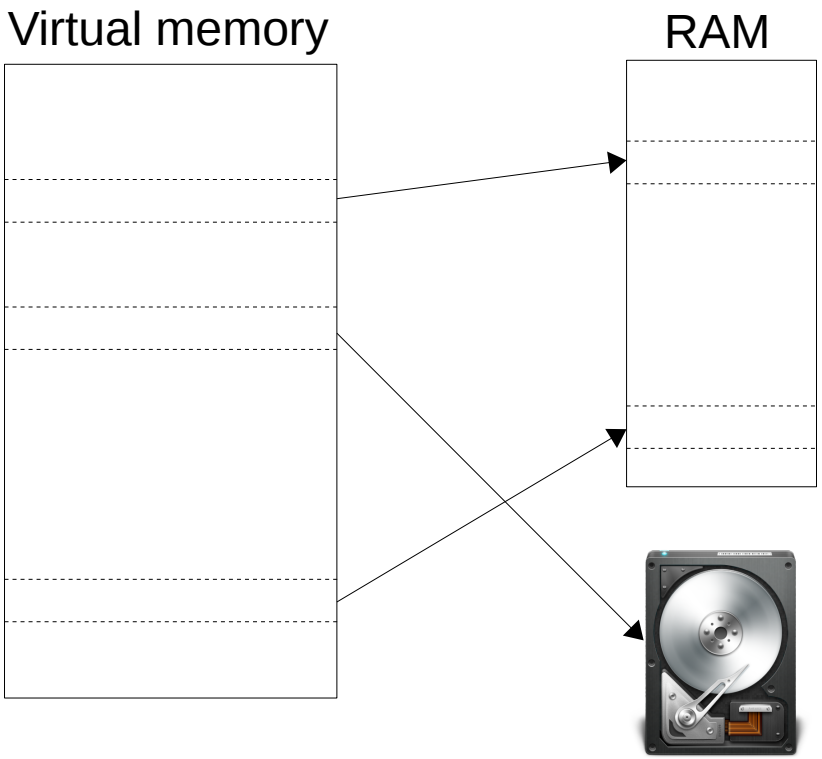
Agenda

- Introduction and objectives
- **Literature review**
- Memory usage monitoring
- Memory leak detection
- Conclusion

Literature review

Background

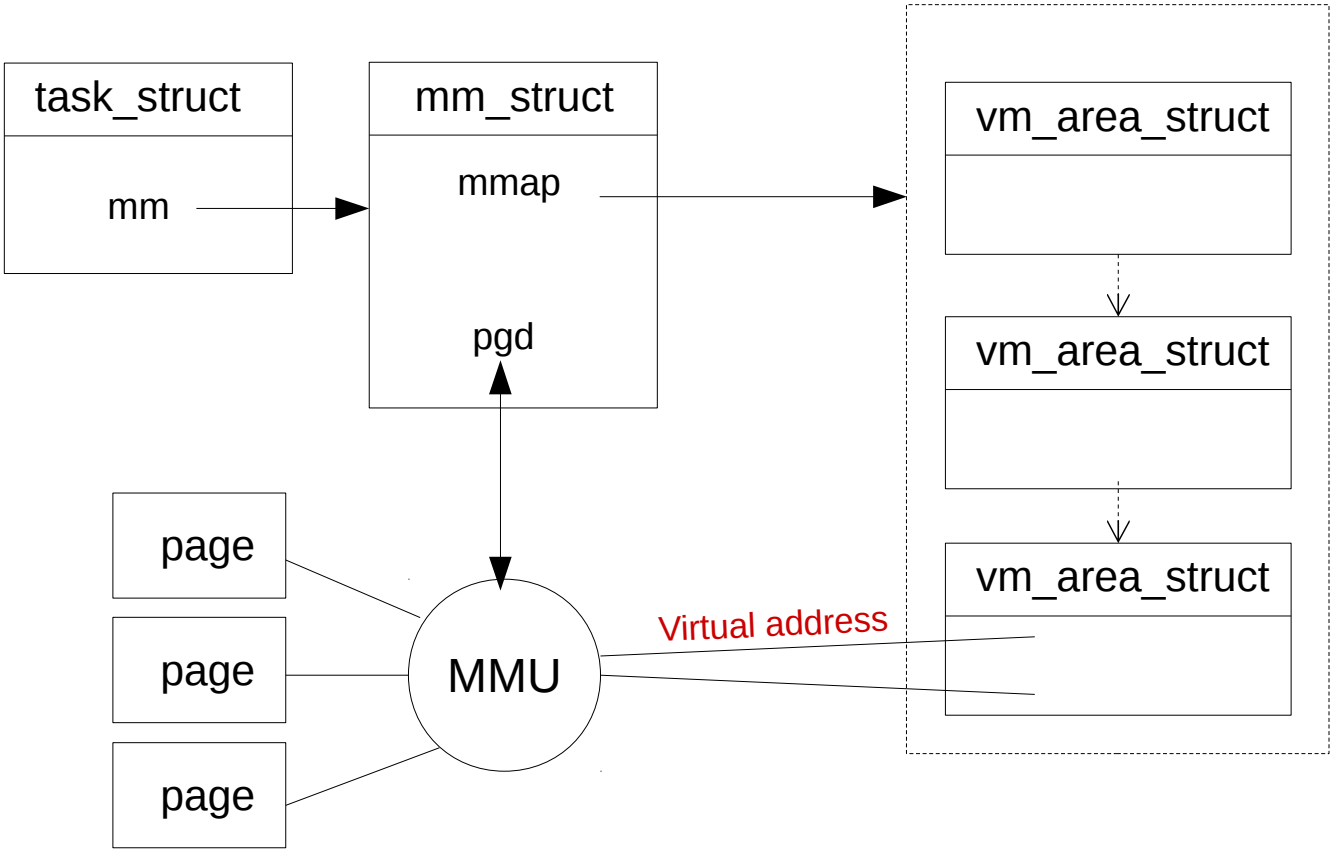
- Virtual memory is an abstraction used by the operating system to compensate the shortage of the physical memory



Literature review

Background

- The mapping between virtual and physical addresses is hardware-assisted (MMU)



Literature review

Memory usage monitoring

- *Printezis and al. : "GCspy, an adaptable heap visualization framework.*
  - Generic client-server architecture
  - Uses JVM instrumentation
  - Requires the instrumentation of the allocator
  - Uses animated images → Not appropriate for high frequency events.

- *Cheadle and al. : "Visualizing dynamic memory allocators"*
  - Extended GCspy to support dlmalloc, a Glibc memory allocator

- *Jurenz and al. : "Memory allocation tracing with VampirTrace"*
  - Require library pre-load instrumentation
  - Uses 2D charts, memory=f(time)

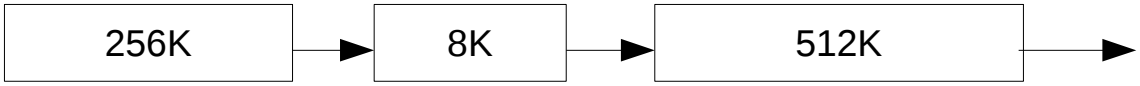
Literature review

Memory Fragmentation Detection

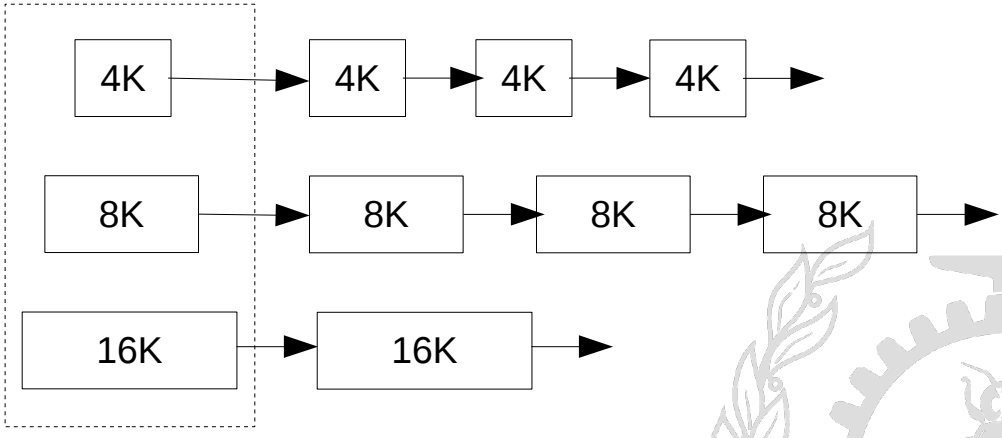
Dynamic allocation allows the applications to allocate and release memory at runtime.

Allocation strategies:

Sequential Fit:
(First Fit, Best Fit, ...)

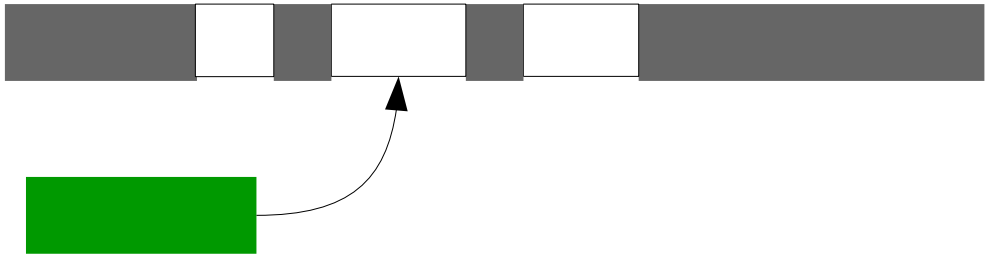


Segregated Free Lists:
(Buddy System)



Literature review

Memory Fragmentation Detection



- *"On the external fragmentation produced by First-Fit and Best-fit", Shore*
  - The authors used a simulator to compare the external fragmentation of First-Fit and Best-Fit.

- *"The memory fragmentation problem : Solved ?", Johnastone*
  - A tracer have been used to study the fragmentation of the heap
  - The fragmentation is estimated by comparing the memory usage to the total size of the heap

Literature review

Memory Leak Detection

- *M. Hauswirth and al. : "Low-overhead memory leak detection using adaptive statistical profiling"*
  - The instrumentation of memory operations is done at compile time.

- *Memcheck*
  - Provided as a Valgrind module
  - Uses the dynamic instrumentation of the application

- *Rubanov et al. : "KEDR, Runtime verification of linux kernel modules based on call interception"*
  - Detects memory leaks in kernel modules by instrumenting the functions kmalloc and kfree.

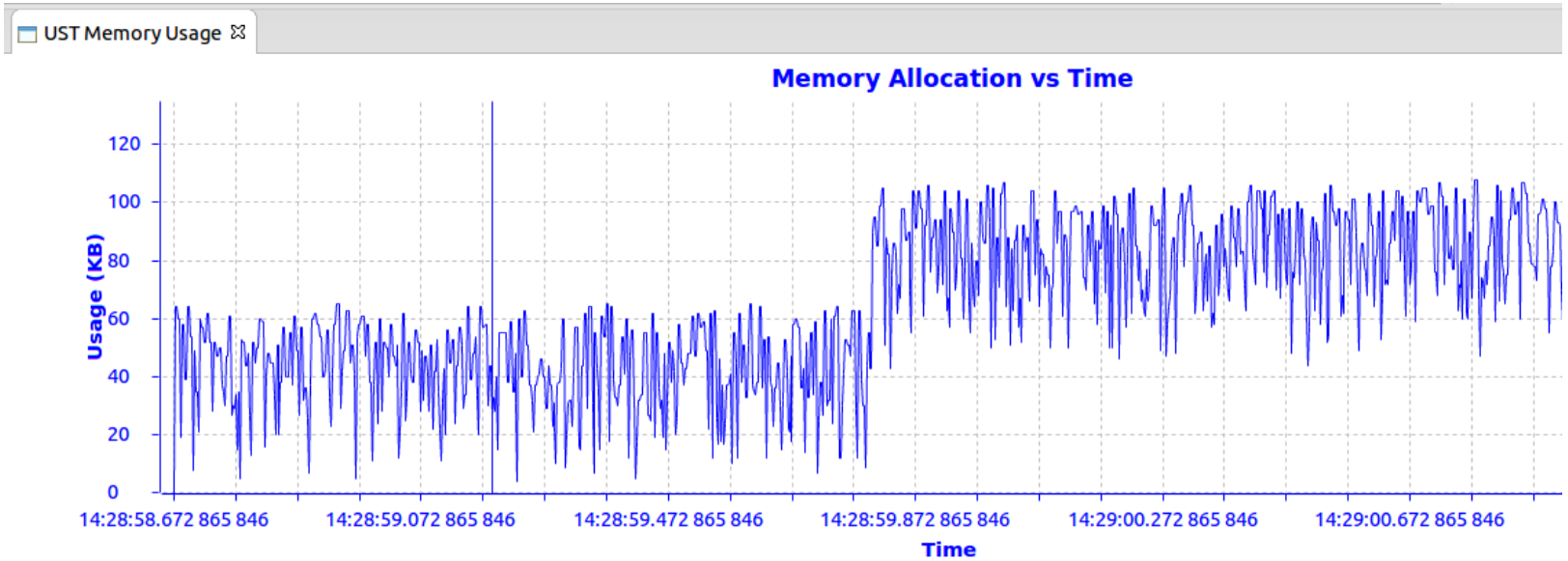
Agenda

- Introduction and objectives
- Literature review
- **Memory usage monitoring**
- Memory leak detection
- Conclusion

Monitoring memory usage

Monitoring virtual memory

- Tracecompass provides a view that shows the virtual memory usage of processes using Lttng-ust
- The instrumentation is based on **liblttng-ust-libc-wrapper**, which defines wrappers to the glibc functions.



Monitoring memory usage

Monitoring virtual memory

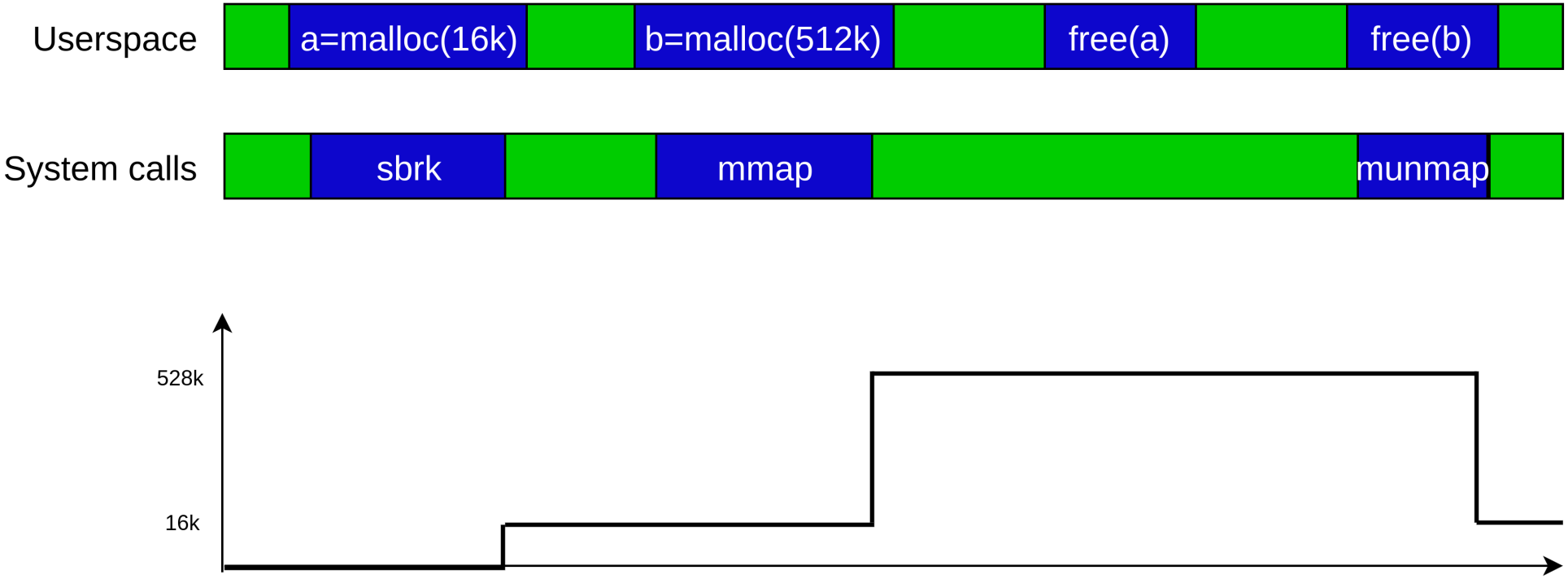
Only programs started with the LD_PRELOAD option can be analyzed.

To have a system-wide analysis, we created a similar view that uses the system calls mmap(), munmap(), and sbrk() as a source of information.

	size <= MMAP_THRESHOLD	size > MMAP_THRESHOLD
Malloc, calloc, realloc	sbrk(size)	mmap
free	None, or sbrk(negative) depending on M_TRIM_THRESHOLD	munmap

Monitoring memory usage

Monitoring virtual memory



Monitoring memory usage

Monitoring Physical memory

- Allocating a physical page is usually done in the context of the process that requires them (kmem_mm_page_alloc)
- Releasing physical pages can be done at any time by a kernel thread (kmem_mm_page_free)

→ Monitoring the physical memory usage of a specific process requires tracing all memory allocation/deallocation events, which are very frequent → lost events + very slow analysis.

Monitoring memory usage

Monitoring Physical memory

Sampling

- The Kernel keeps information about physical memory usage in the mm_struct data structure (RSS: Resident set size)

- This counter is adjusted each time a physical is inserted or removed from the page table of the process.

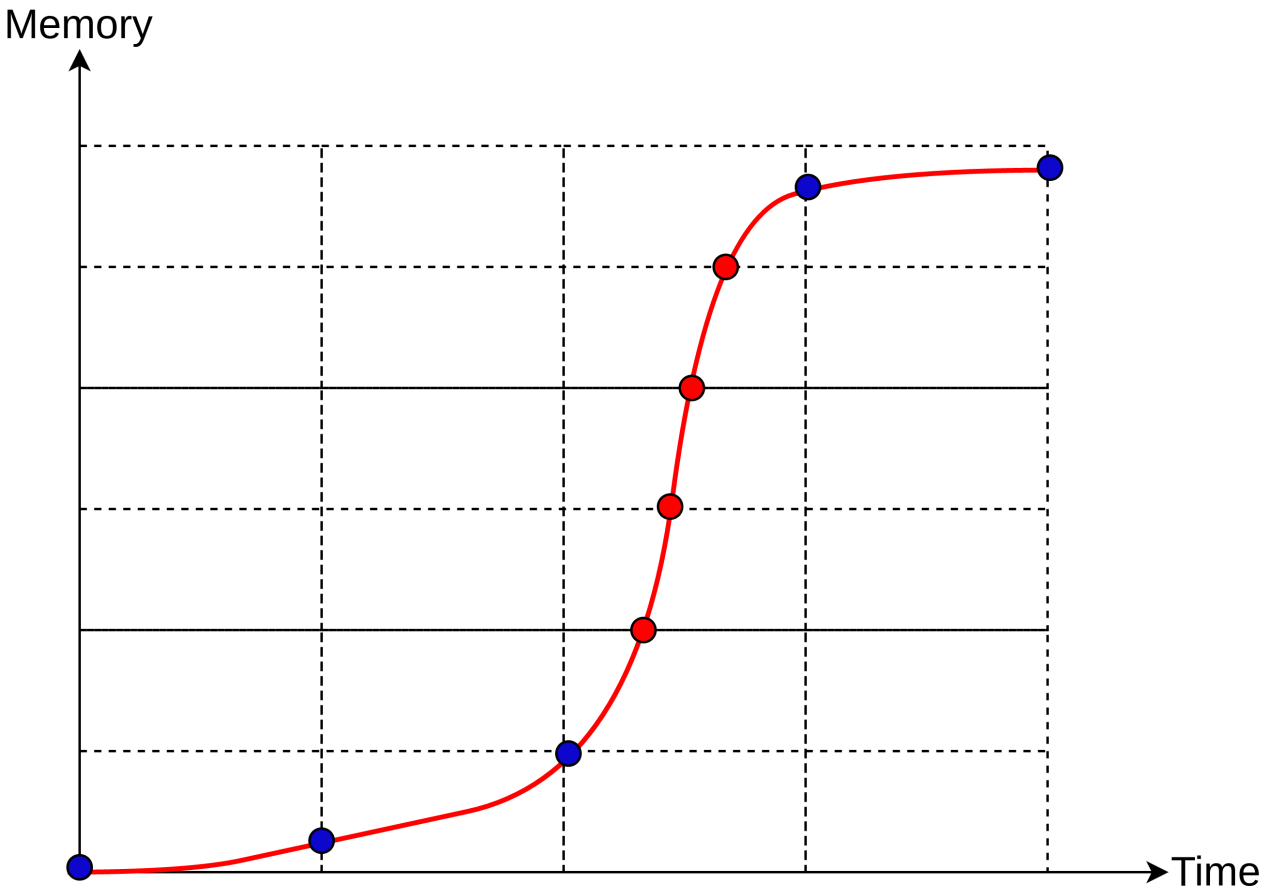
It is difficult to select good sampling rate

Monitoring memory usage

Monitoring Physical memory

(Time, Space) Sampling

An event is triggered if memory variability exceeds a certain threshold



Monitoring memory usage

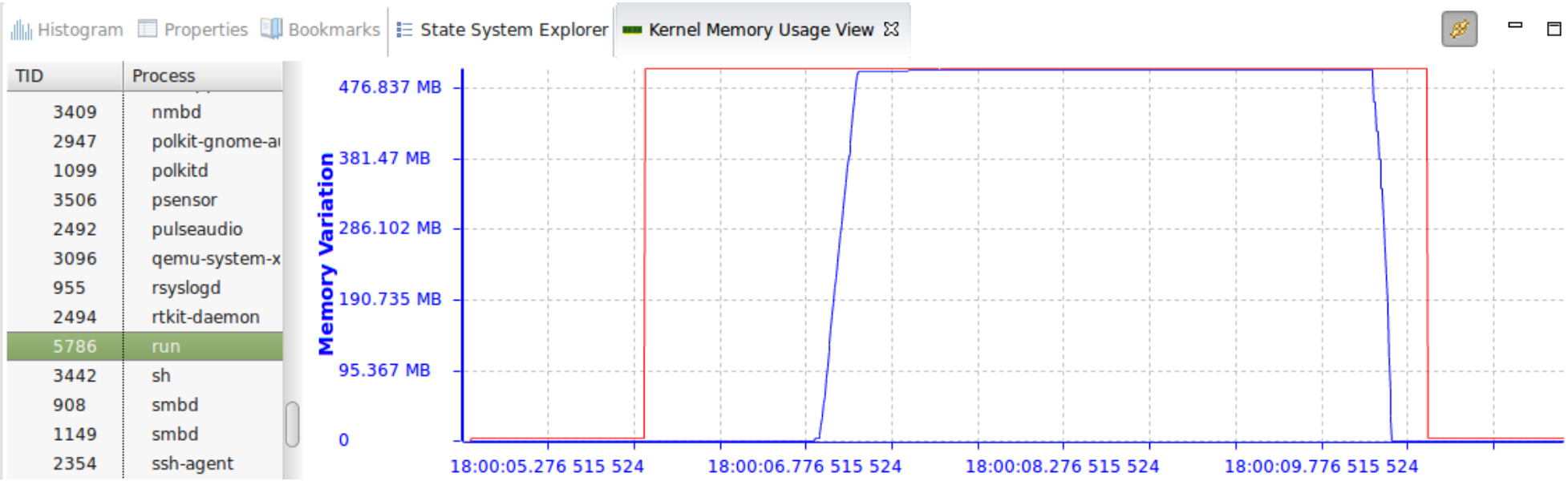
Monitoring Physical memory

(Time, Space) Sampling

- The sampling mechanism is implemented as a Kernel module.
- The module is configurable through the *proc* file system. (sampling rate, variability threashold.
- Lock-free data structures are used to provide a good scalability
  - RCU Hashmap is used to hold process information
  - Memory variability is defined as atomic_long

Monitoring memory usage

Monitoring Physical memory



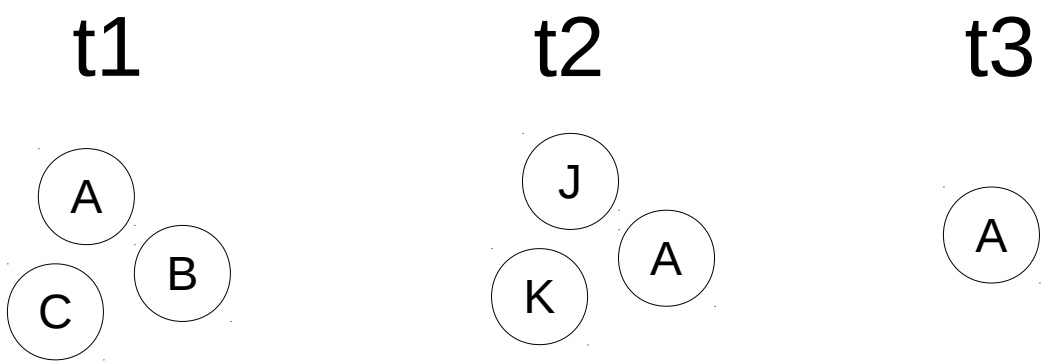
- Unified virtual/physical memory view

Agenda

- Introduction and objectives
- Literature review
- Memory usage monitoring
- **Memory leak detection**
- Conclusion

Memory leak detection

- A memory leak occurs when a computer program does not free memory objects that are no longer needed.
- Tracing can be used to track the potentially leaked objects, but only the programmer is able to confirm if an object is leaked or not.



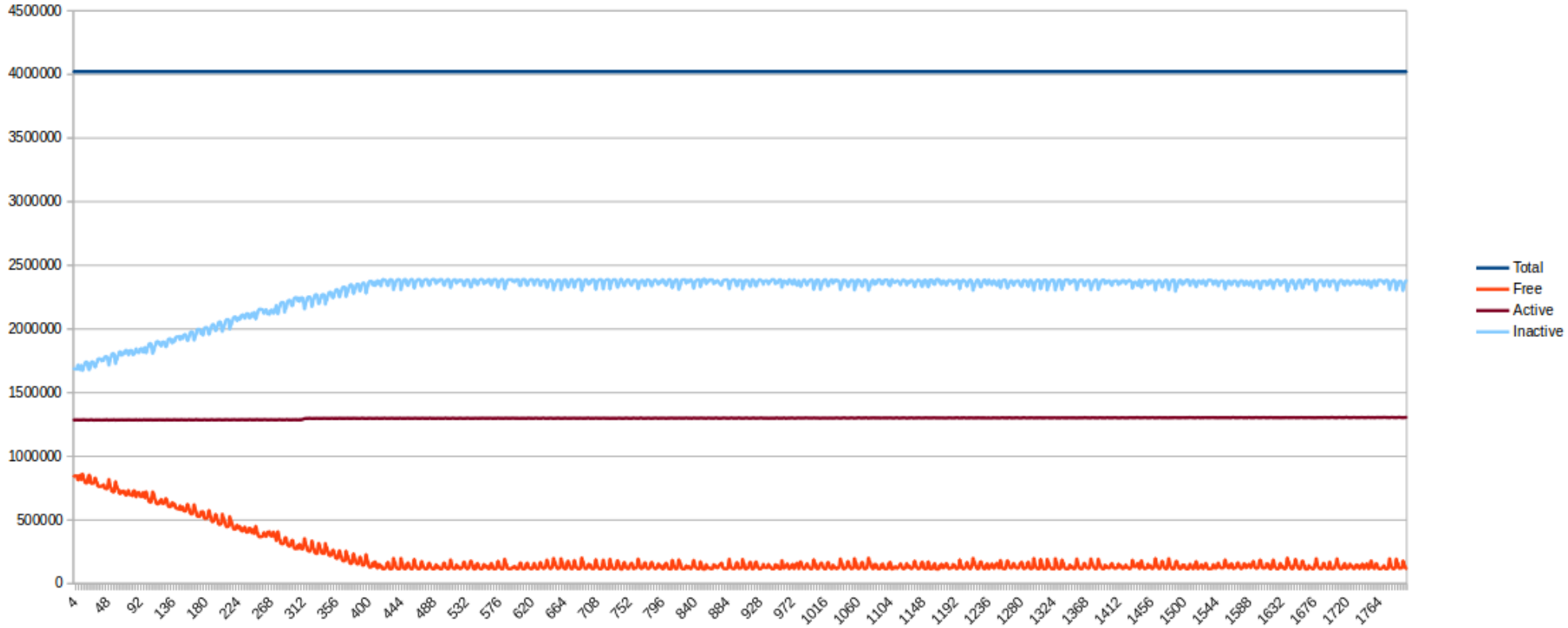
A is potentially a leaked object

Memory leak detection

Use case (Kernel 3.4)

```
#!/bin/bash

while true;
do
dd if=/dev/zero of=/output.dat  bs=10M count=1
done
```



Memory leak detection

Use case (Kernel 3.4)

- We used Lttng to find the pages that are allocated and never released.
- The events used are kmem_mm_page_alloc and kmem_mm_page_free with the call stack context.

→ We found out that the "leaked" pages are allocated in the following code path

```
'__alloc_pages_nodemask',
'alloc_pages_current',
'__page_cache_alloc',
'find_or_create_page',
'__getblk',
'jbd2_journal_get_descriptor_buffer',
'jbd2_journal_commit_transaction',
'kjournald2',
'kthread',
```

Memory leak detection

Use case (Kernel 3.4)

- We contacted the developers of the file system to confirm the problem. By looking at the call stack, they quickly recognized the problem and gave us the required patch.
- The problem is in the journal mode of the ext4 file system. It is solved in Kernel v4.4-rc5

Conclusion

- The preliminary results show that tracing can be used to detect interesting problems related to memory management mechanisms.
- The overhead of tracing should be carefully measured on different workloads to make sure that the tracer is not altering the normal behavior of the system.
- As a future work, we started to instrument the KSM mechanism to evaluate its efficiency in different contexts.

Thank You !