Out of Hypervisor (OoH)

Pr Alain Tchana - (alain.tchana@ens-lyon.fr)
ENS Lyon

EuroDW'22 - March, 5

About me

- Originate from Cameroon, in Africa
- Moved to France for an internship in 2008
- 2030, back to Cameroon
  - President of the republic: 2032 $\rightarrow \infty$

Some famous cameroonians

Roger Milla
Wold cup'90

Manu Dibango
plagiarised by Michael Jackson
("*mamasa mamako mamamakossa*")

Pierre Tchana
Salsa

Alain Tchana
Dakar 2021

My group

PhD students

- 2 females (S. Bitchebe, J. Kouam) and 3 males (K. Nguetchouang, Y. Kone, and T. Dubuc)

Interns

- 0 female (:() and 3 males (B. Teguia, A. Leberre, R. Colin)

OoH: Out of Hypervisor

Make nested virtualization practical

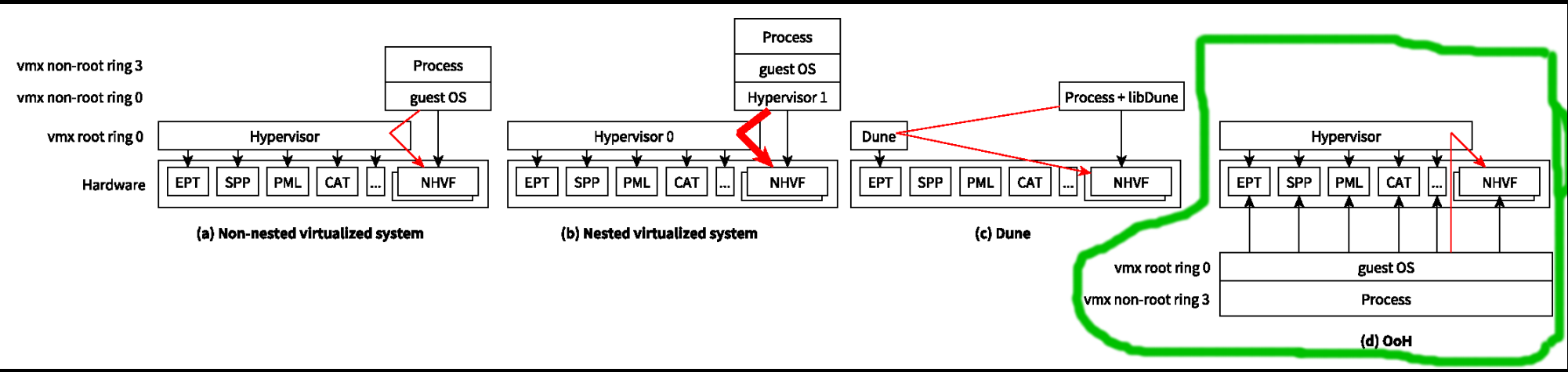
- OoH exposes relevant *hypervisor-oriented* hardware virtualization features to the guest OS so that its processes can also take benefit from them
- OoH is nested feature virtualization
  - nested virtualization (NV) is hypervisor stacking to run VMs inside a VM
  - NV is nice, but too ambitious, requires a lot of research efforts for very few use cases (testing, demo, continous delivery)
- OoH targets one hardware virtualization feature at a time
  - especially those which can improve applications

Muller-ization

Muller-ization

- OoH is about a new research axis

OoH in the design space



NHVF = Non Hardware Virtualization Features (e.g., set CR3 register on x86)

Dune is A. Belay et al. OSDI'12

OoH

Grants

- Stella Bitchebe
  - NEC Student Research Fellowship 2021
  - Microsoft Research PhD Fellowship 2021
  - French ANR Scalevisor

This talk illustrates OoH with 2 features

OoH for Intel SPP

- to reduce memory overhead of guard page-based secured heap memory allocators
- work in progress
- by Yves Kone, Augusta Mukam and Stella Bitchebe

OoH for Intel PML

- to improve process/container checkpointing, concurrent GCs
- work completed!
- by Stella Bitchebe

1 OoH

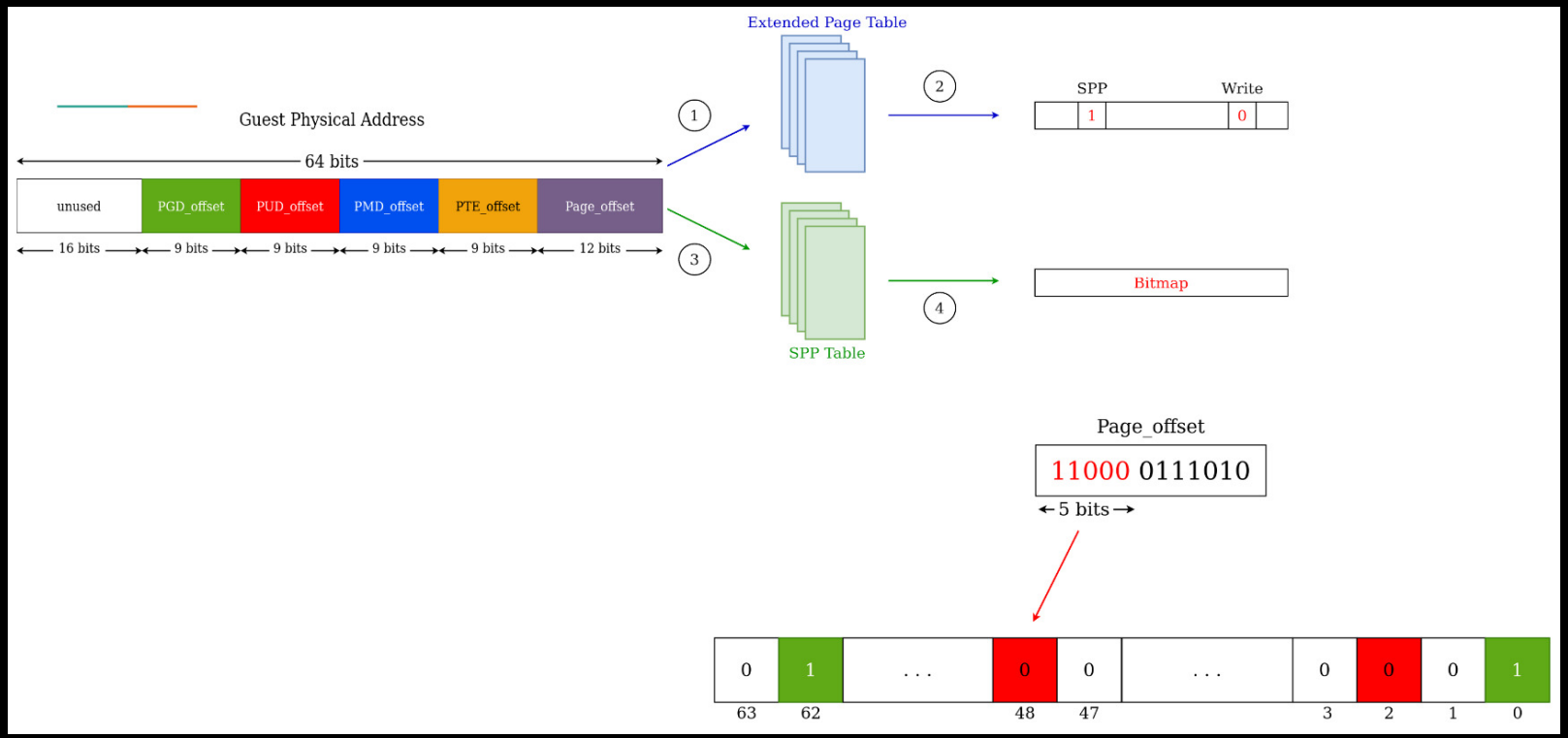
2 OoH for Intel SPP

3 OoH for Intel PML

Intel SPP

Overview

- EPT-based, 4KB page size, organised into 32 sub-pages (128 B each)

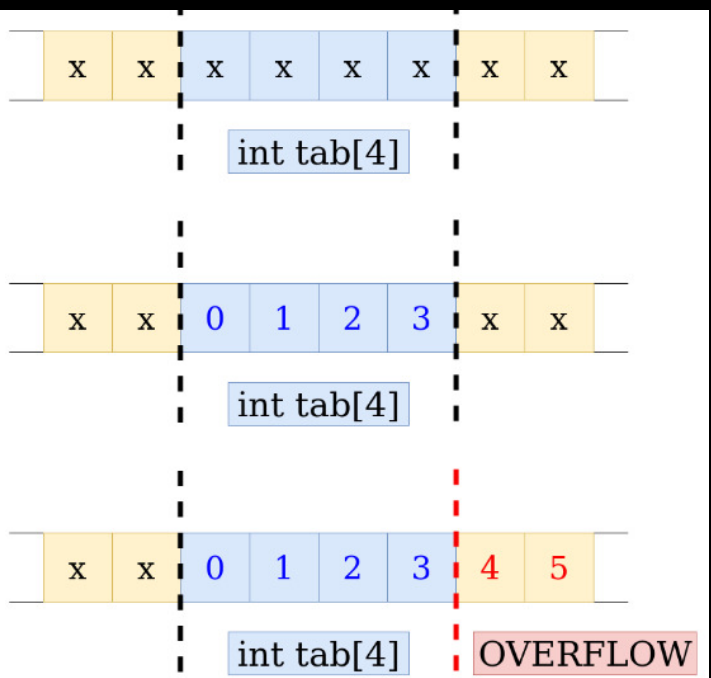


Buffer overflow vulerability

Unsafe languages such as C

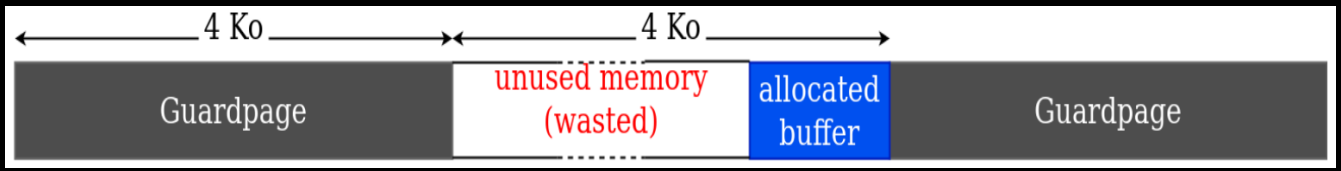
```
int *tab = (int *) malloc(sizeof(int) * 4);

for (int i = 0; i < 6; i++) {
    tab[i] = i;
}
```



Buffer overflow vulerability

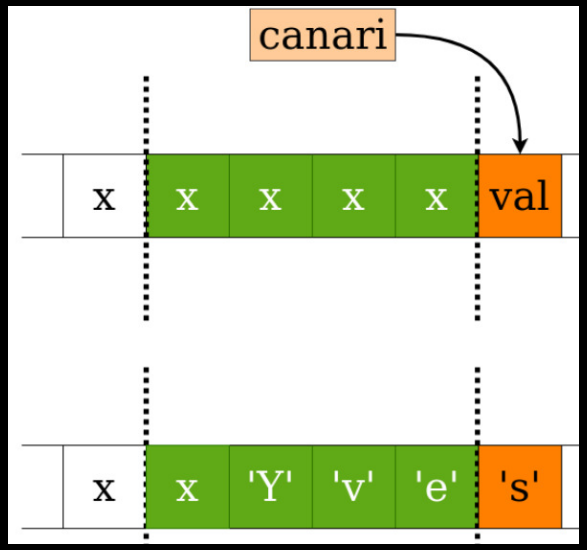
Guard pages



- Avantage: sync detection
- Limitations: memory waste

Buffer overflow vulerability

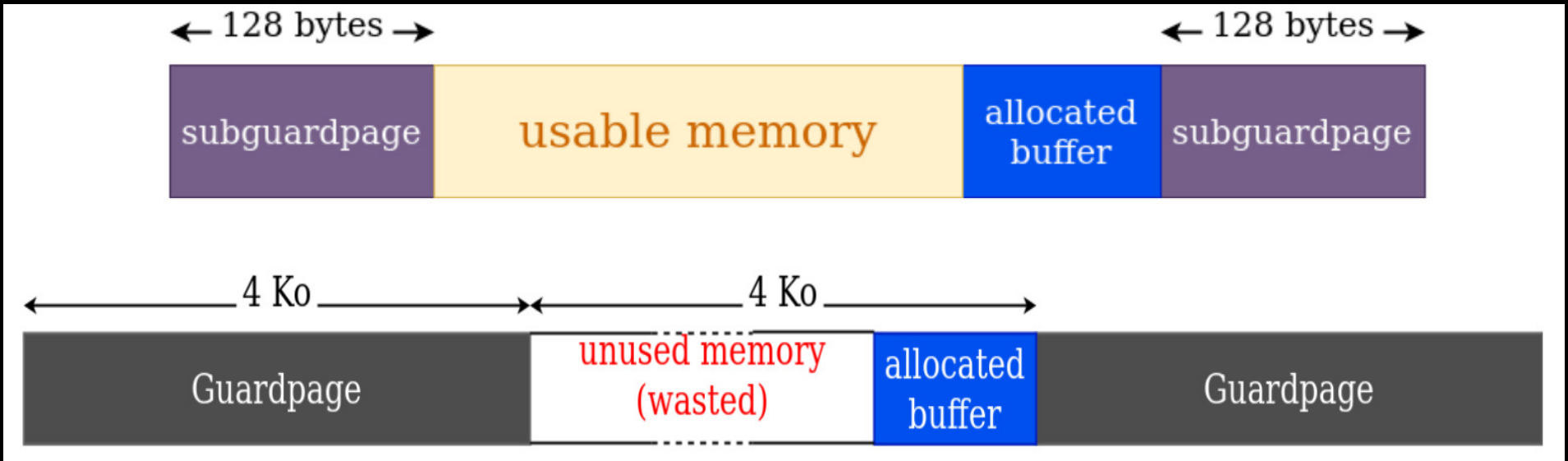
Canari



- Avantage: low memory waste
- Limitations: asyn detection

OoH for Intel SPP: Sub guard pages

overview

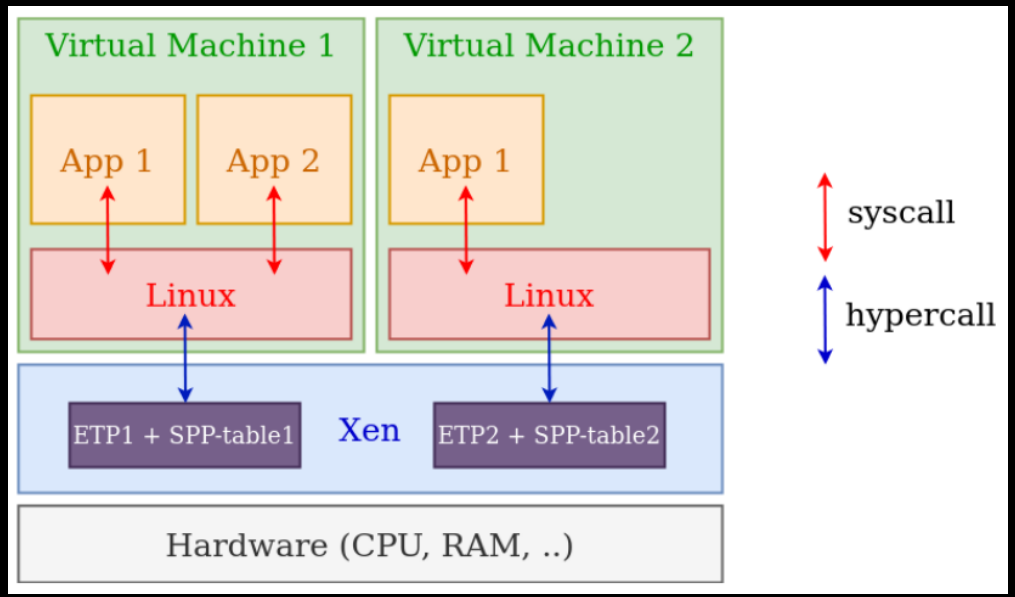


- Avantage: sync detection, low memory waste (maximum 128KB instead of 4KB $\rightarrow$ 32$\times$)

OoH for Intel SPP: Sub guard pages

Challenges

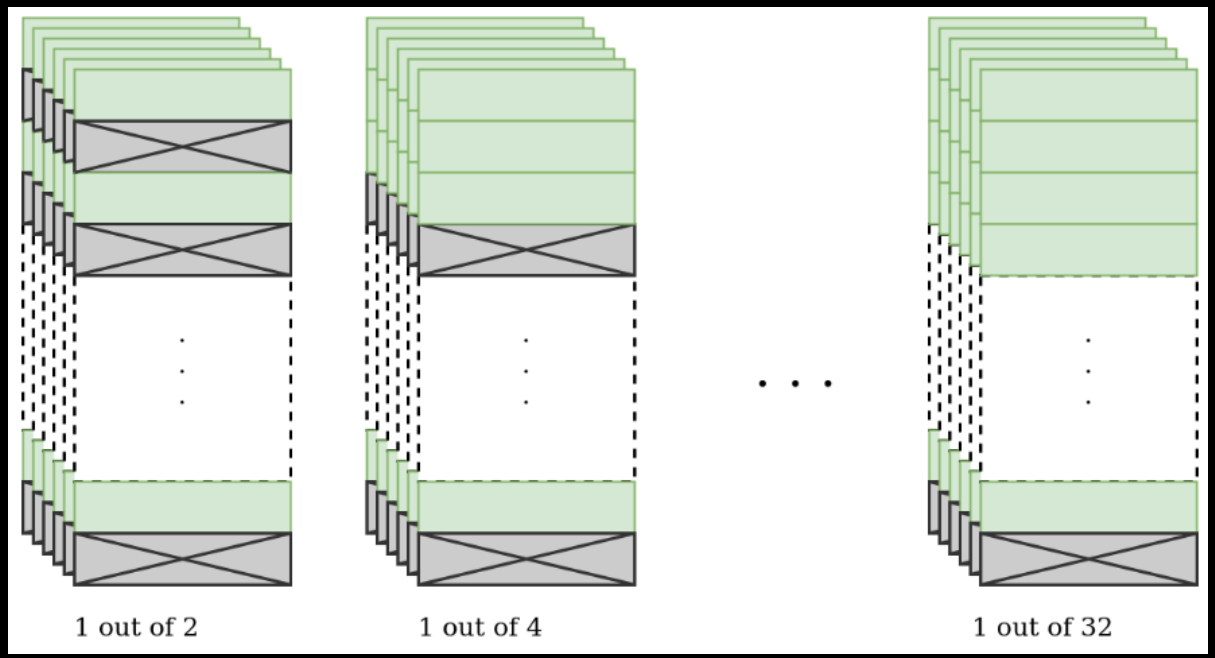
- SPP is only managed by the hypervisor
- A naive solution would lead to several costly hypercalls



OoH for Intel SPP: Sub guard pages

Approach: Pre-initialized SPP page pool

- On demand allocation of pre-initialized SPP pages
  - with different write protected sub-page frequencies



OoH for Intel SPP: Sub guard pages

In the guest kernel

- Addition of a new memory allocator: for SPP pages
- `struct page *spp_get_page(int freq)`: allocate an SPP pre-initialized page
- `int spp_put_page(struct page *page, int freq)`: the inverse
- Some VMAs are tagged spp
- New `madvise` flags: MADV_SPP2,...,MADV_SPP32

OoH for Intel SPP: Sub guard pages

In the heap memory alocator (e.g., Slimguard)

- `mmap` a buffer, which creates a new VMA
- `madvise` the buffer using MADV_SPP2,..., or MADV_SPP32
- use that buffer to allocate secured malloced buffers

OoH for Intel SPP: Sub guard pages

On page fault

- Check if the target VMS is tagged SPP
- If yes, use the new allocator, else use the default buddy allocator

OoH for Intel SPP: Sub guard pages

Implementation

- Xen hypervisor
- Linux guest kernel
- Slimguard secured heap memory allocator

OoH for Intel SPP: Sub guard pages

Evaluations

- In progress
- We expect
  - 32× memory waste reduction
  - with no performance overhead

Back to "Some famous cameroonians"

Roger Milla
Wold cup'90

Manu Dibango
plagiarised by Michael Jackson
("*mamasa mamako mamamakossa*")

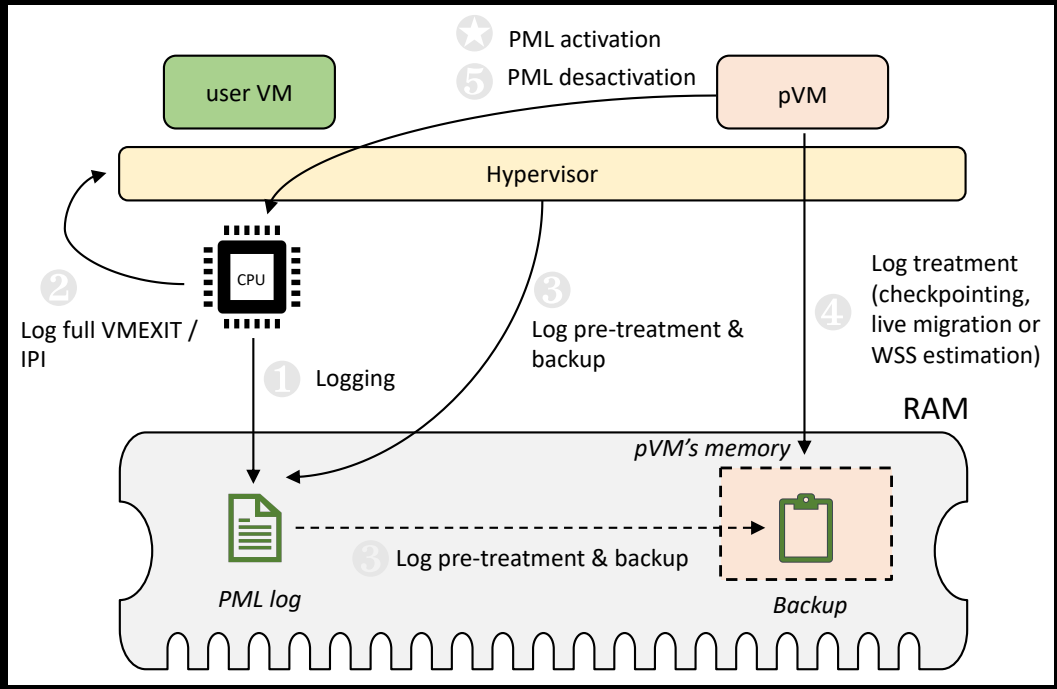
Pierre Tchana
Salsa

Alain Tchana
Dakar 2021

1. OoH
2. OoH for Intel SPP
3. OoH for Intel PML

Intel PML

Functioning



Intel PML

- to accelerate CRIU and Boehm GC

OoH PML: dirty page tracking in userspace

Some definitions

- Tracker: the monitoring thread
  - e.g., CRIU, Boehm GC (C and C++ applications)
- Tracked: the thread whose memory is monitored
  - any application

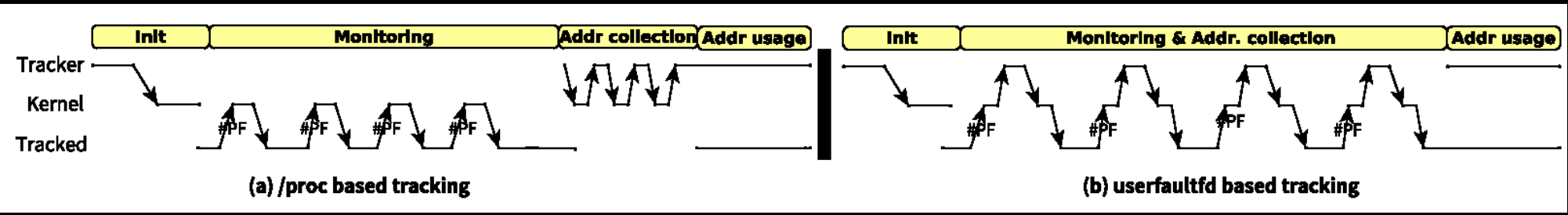
Current approach

- page write protection
- two main solutions
  - bit 55 of /proc/<PID>/pagemap
  - userfaultfd (ufd)

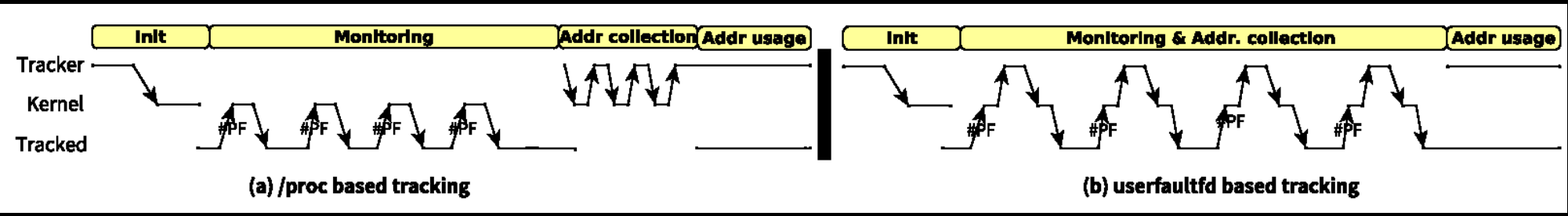
Dirty page tracking in userspace

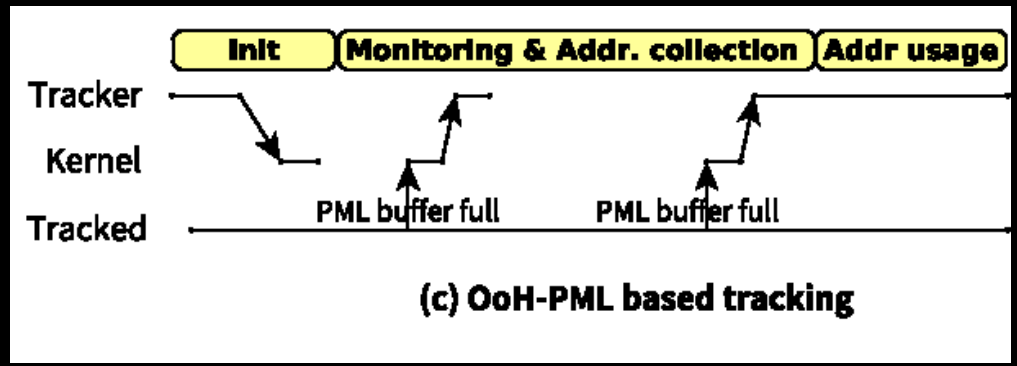
Overall functioning

- Tracker's activity can be organized in four phases:
  - the initialization of the tracking method,
  - the monitoring
  - the collection of dirty page addresses
  - the exploitation of the latter (e.g., for checkpointing)



Dirty page tracking in userspace





Page write protection

Limitations

- ufd
  - we measured an overhead of up to 1,462% and 1,349% for 1GB on Tracked and Tracker respectively.
  - due to page fault handling
- /proc
  - page table parsing and to flush the TLB (about 2.234ms when the monitored memory is 1GB), soft-dirty bit setting (about 33.5us)
  - /proc/PID/pagemap parsing in userspace (about 594.187ms for 1GB)

OoH for PML

Challenges

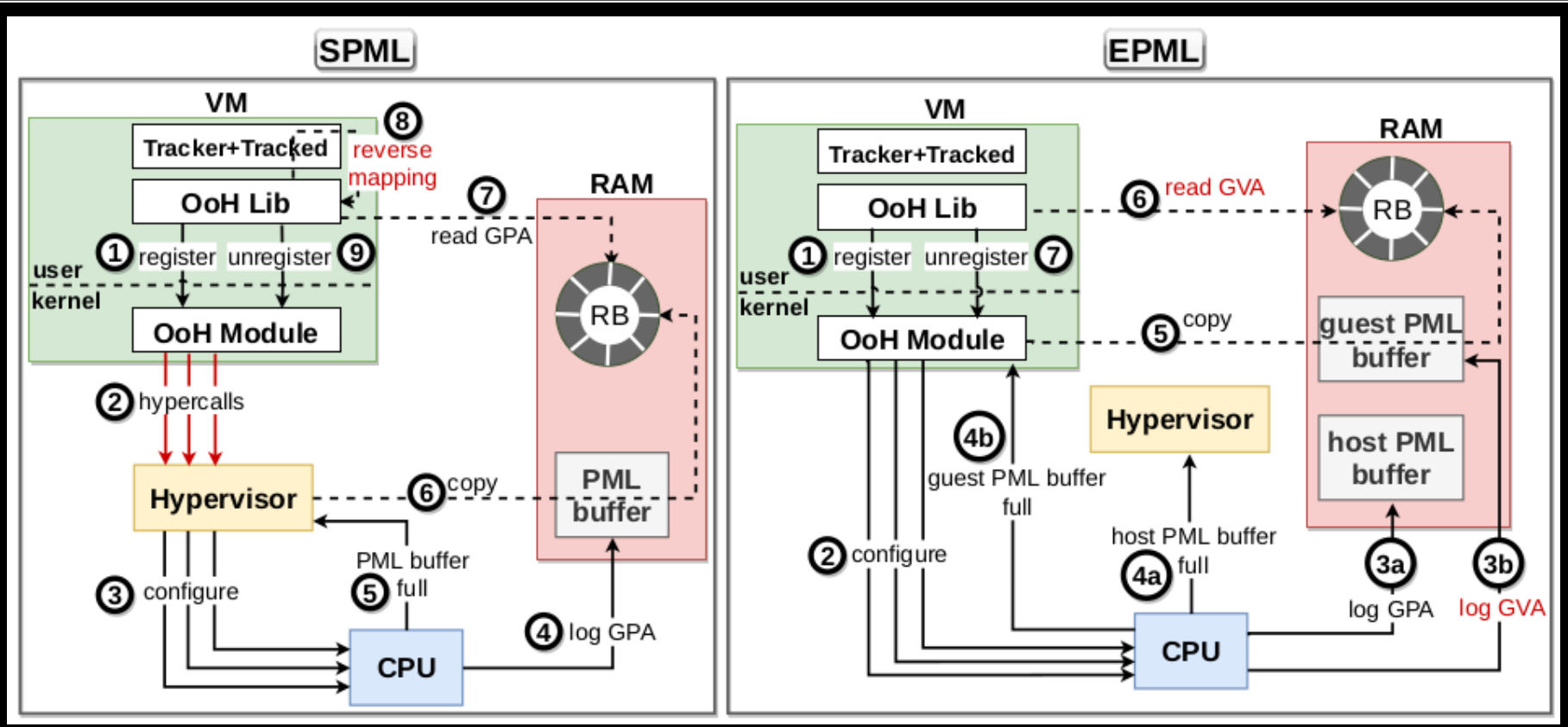
- (C_1) PML can only be managed by the hypervisor
- (C_2) PML now works at coarse-grained, that is it concerns the entire VM.
- (C_3) PML only logs GPA.

OoH for PML

Two solutions

- Extended PML (EPML): small hardware changes
- Shadow PML (SPML): no hardware modification
  - its huge overhead justifies EPML

OoH for PML


SPML
VM
Tracker+Tracked
OoH Lib
OoH Module
user
kernel
1 register
unregister 9
8 reverse mapping
7 read GPA
RAM
RB
2 hypercalls
Hypervisor
6 copy
PML buffer
3 configure
5 full
CPU
4 log GPA
EPML
VM
Tracker+Tracked
OoH Lib
OoH Module
user
kernel
1 register
unregister 7
6 read GVA
5 copy
RAM
RB
guest PML buffer
host PML buffer
Hypervisor
4b guest PML buffer full
2 configure
host PML buffer
4a full
CPU
3a log GPA
3b log GVA

OoH for PML

SPML limitations

- Costly reverse mapping (takes about 15.739 s for 1GB working set)
- Costly hypercalls (4.49μs to perform an empty hypercall)

OoH for PML

EPML

- We leverage VMCS shadowing to allow the guest to perform `vmread` and `vmwrite` instructions
- At load time, OoH Module calls the hypervisor to enable and configure VMCS shadowing

OoH for PML

Hardware changes

- We introduce in VMCS's VM-Execution Control area a new field (called `Guest PML Address`)
- We extend the page walk to make the processor log GVA to the guest-level PML buffer and the GPA to the hypervisor-level PML buffer.
- We modify the hardware so that when the guest-level PML buffer is full, the processor raises a virtual self-IPI (Inter-Processor Interrupt)

OoH for Intel PML

Implementation

- We implemented EPML's hardware changes in BOCHS
- We used Xen as the hypervisor and Linux as the guest OS
- We integrated OoH Lib with CRIU and Boehm GC

OoH for Intel PML

Evaluations

1. What is the potential overhead or improvement of SPML and EPML compared to `e/proc` and `ufd`?
2. What is the scalability of SPML and EPML?

OoH for Intel PML

Methodology

- We used DELL Intel Core i7-8565U (which provides PML) for experiments
- Micro-benchmarks: Array parsing and GCBench
- Macro-benchmarks: tkrzw **tkrzw** applications (key value store) and Phoenix applications (MapReduce)
- We considered three working set sizes (Small, Medium and Large)

OoH for Intel PML

Evaluations

- How to accurately evaluate EPML?
- Approach
  - build a formula
  - show the accuracy of that formula on other techniques, which are measurable
  - by construction, the formula is accurate for EPML

OoH for Intel PML

Evaluations

- The execution time of Tracker when it implements technique x

$$E(C_{tker}) = E(C_x) + E(C_p) + I(C_x, C_p) \quad (1)$$

where $E(C)$ is the execution time of code C and $I(C_1, C_2)$ is the impact of C_1 on C_2.

Evaluations

Impact on Tracker

$$
\begin{aligned}
E(C_{/proc}) &= E(C_{\text{echo 4 > /proc/PID/clear\_refs}}) \\
&\quad + E(C_{\text{page table walk in userspace}}) \\
E(C_{UFD}) &= E(C_{\text{ioctl write\_protect}}) \\
&\quad + E(C_{\text{ioctl register}}) \\
&\quad + E(C_{\text{ioctl write\_unprotect}}) \\
E(C_{SPML}) &= E(C_{\text{ring buffer copy}}) \\
&\quad + E(C_{\text{reverse mapping}}) \\
&\quad + E(C_{\text{enable/disable PML}}) \\
E(C_{EPML}) &= E(C_{\text{ring buffer copy}}) \\
&\quad + E(C_{\text{enable/disable PML}})
\end{aligned}
\tag{2}
$$

OoH for Intel PML

Evaluations

- The execution time of Tracked when it is monitored by a tracker using the technique x:

$$E(C_{tked\_tker}) = E(C_{tked}) + E(C_{tker}) + I(C_x, C_{tked}) \quad (3)$$

where $I(C_x, C_{tked})$ consists of page faults, vmexits, etc.. Thus, the overhead of x on Tracked is $E(C_{tker}) + I(C_x, C_{tked})$.

Evaluations

Impact on Tracked

$$
\begin{aligned}
I(C_{/proc}, C_{tked}) &= E(C_{PFH\ in\ kernelspace}) \\
&\quad + E(C_{context\ switch}) \\
I(C_{UFD}, C_{tked}) &= E(C_{PFH\ in\ userspace}) \\
&\quad + E(C_{context\ switch}) \\
I(C_{SPML}, C_{tked}) &= E(C_{vmexits}) \\
&\quad + N \times E(C_{vmread/vmwrite}) \\
I(C_{EPML}, C_{tked}) &= N \times E(C_{vmread/vmwrite})
\end{aligned}
\tag{4}
$$

For $I(C_{EPML}, C_{tked})$, we use SPML's N (the number of context switches) as it is the same (validated by running SPML and EPML under BOCHS).

Formula validation

Metric	Time (ms)
$E(C_{tker})$ **measured**	**5503.79**
$E(C_{tked\_tker})$ **measured**	**135255.35**
$E(C_p)$	251.35
$E(C_{copy\_rb})$	0.49
$E(C_{disable\ pml})$	2.06
$E(C_{rev.\ mapping})$	5419
$E(C_{tker})$ **estimated**	**5672.9**
$E(C_{vmexits})$	18000
N	39
$E(C_{vmread,vmwrite})$	1.73×10^{-3}
$E(C_{tked\_tker})$ **estimated**	**136919.85**

(a) SPML

Metric	Time (ms)
$E(C_{tker})$ **measured**	**1097.99**
$E(C_{tked\_tker})$ **measured**	**115283.98**
$E(C_p)$	251.35
$E(C_{clear\_refs})$	1.409
$E(C_{PTwalk})$	0.89
$E(C_{tker})$ **estimated**	**1116.09**
$E(C_{PFHuser})$	0.27
$E(C_{tked\_tker})$ **estimated**	**114418.58**

(b) /proc

Evaluations

Formula validation

An accuracy of about 96.34% and 99% respectively for SPML and /proc formulas

Evaluations

Micro-benchmark results

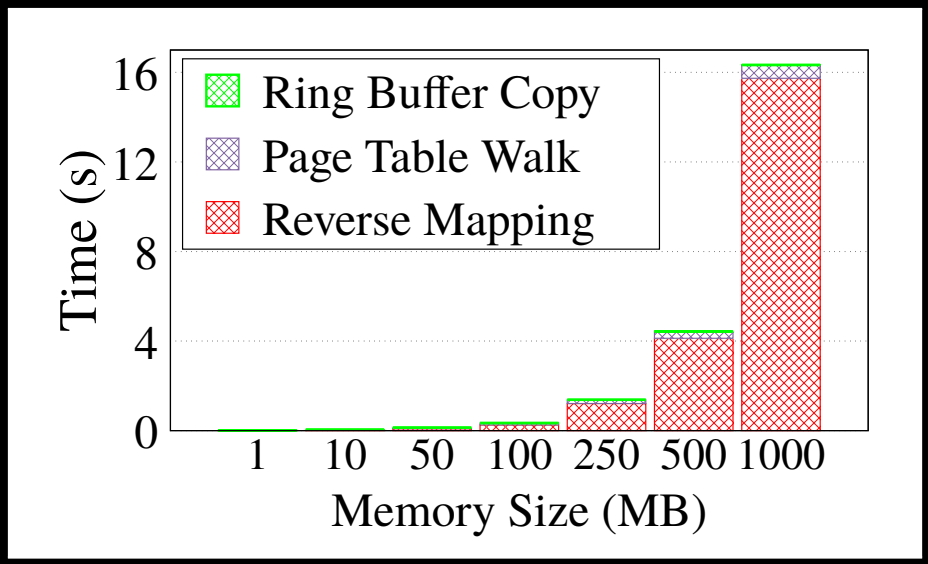


Figure: *Reverse mapping appears to be the bottleneck of SPML.*

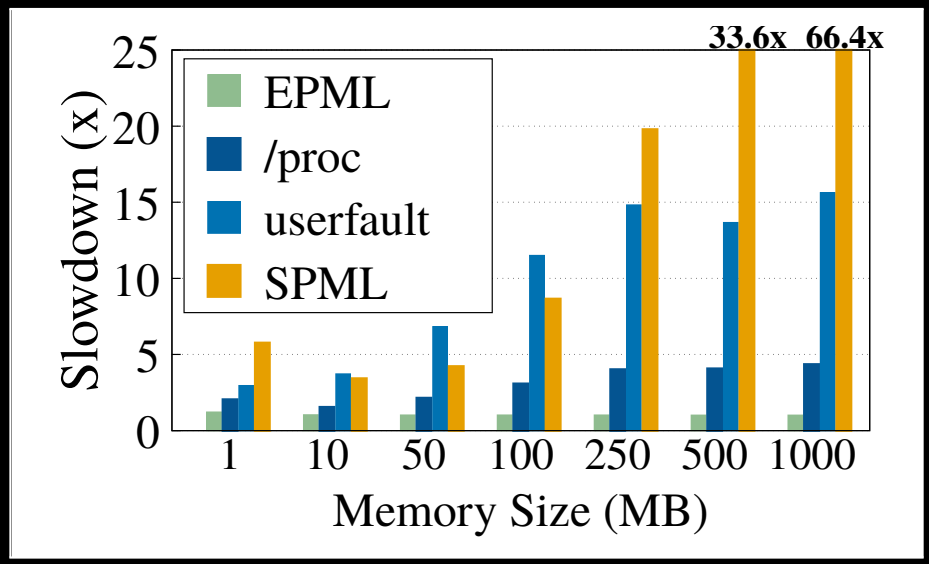


Figure: *Overhead of each tracking technique on the micro-benchmark.*

Evaluations

Impact on Boehm

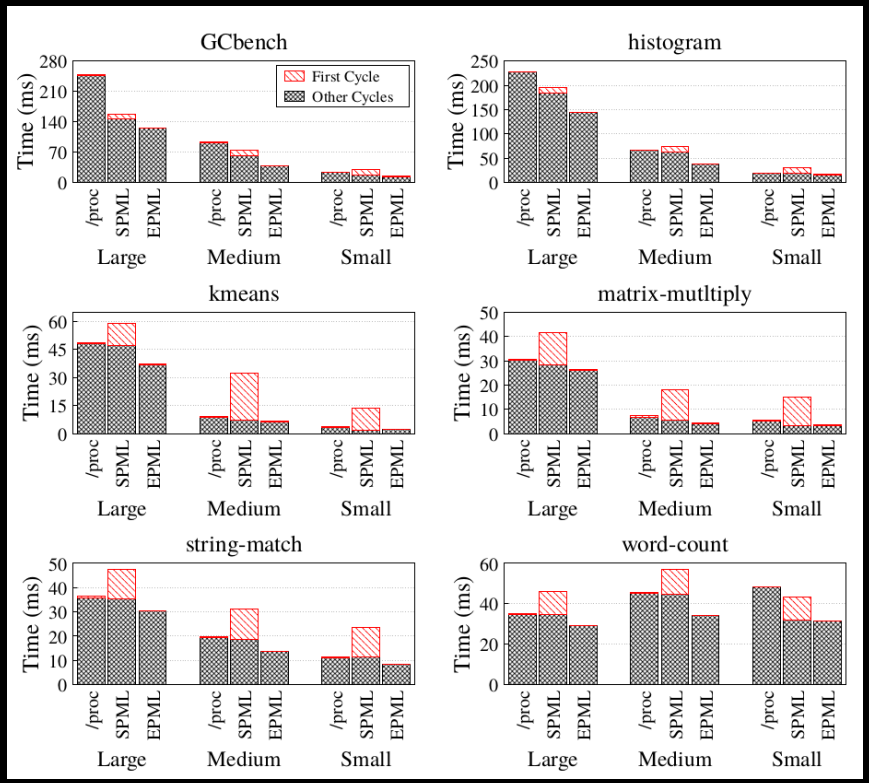


Figure: *We highlight the first GC cycle during which Boehm performs reverse mapping with SPML.*

Evaluations

Impact on applications

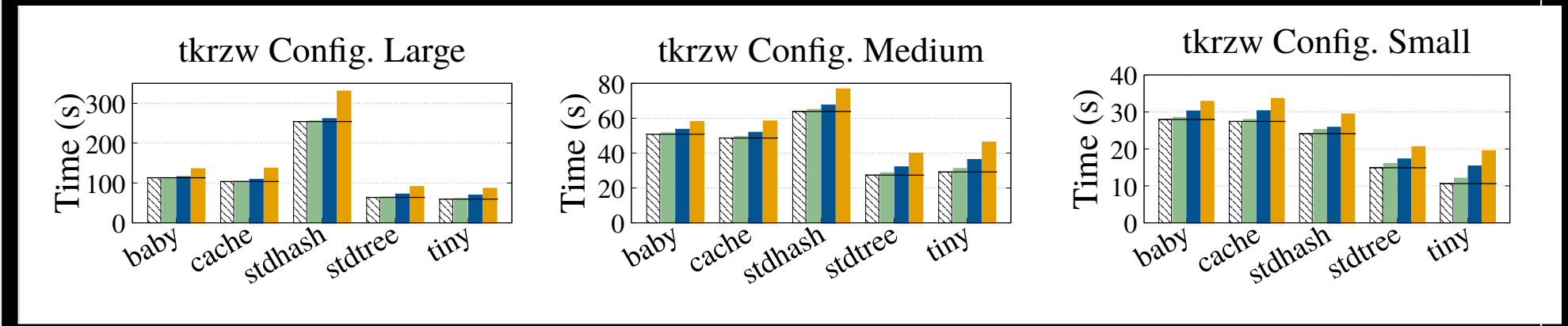


Figure: *Impact of Boehm GC.*

Evaluations

Impact on CRIU

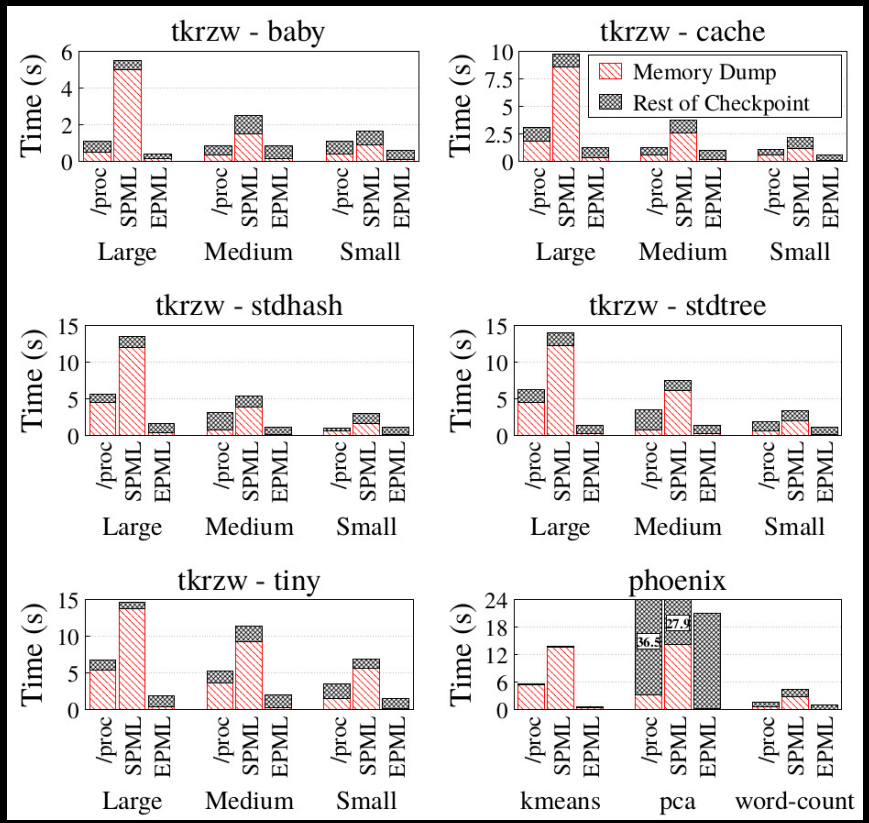
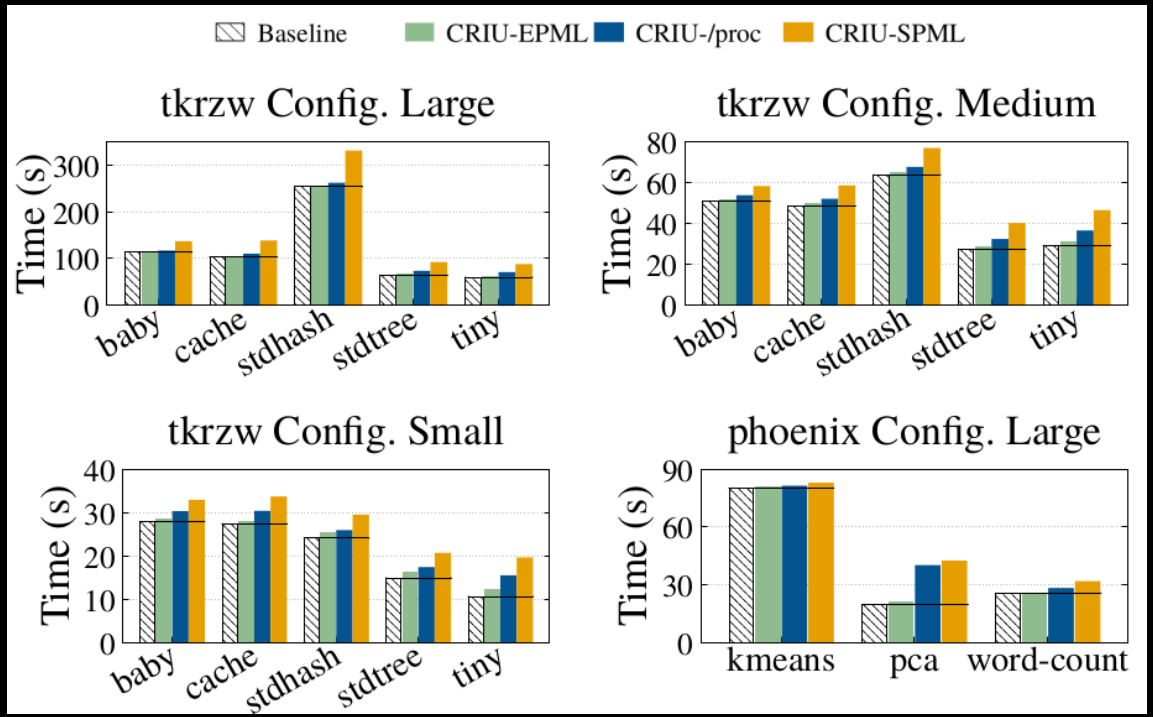


Figure: *We highlight MD phase during which CRIU performs reverse mapping with SPML.*

Evaluations

Impact on the checkpointed app



Evaluations

Summary

- Impact on the tracker
  - SPML induces up to 5× slowdown on CRIU and 3× slowdown on Boehm GC.
  - EPML brings up to 4× speedup compared to `/proc` and 13× speedup compared to SPML for CRIU; up to 2× speedup compared to `/proc` and up to 6× speedup compared to SPML.

Evaluations

Summary

- Impact on the application performance
  - `/proc`, which is the default solution implemented in both CRIU and Boehm, incurs an overhead of up to 102% with CRIU on the Phoenix `pca` application, and up to 232% with Boehm on the Phoenix `string-match` application.
  - The overhead of SPML is up to 114% with CRIU and 273% with Boehm on the same applications.
  - EPML leads to the lowest overhead, which is about 7% with CRIU and 24% with Boehm.

In this talk

OoH for Intel SPP

- for reducing memory overhead of guard page-based secured heap memory allocators
- work in progress

OoH for Intel PML

- for improving process/container checkpointing, concurrent GCs
- excellent results

The main message

- When thinking hypervisor-oriented hardware virtualization features, please think how they can be also used by processes from inside VMs, this may need few efforts

For questions

- Stella Bitchebe
  - mail: celestine-stella.ndonga-bitchebe@ens-lyon.fr
  - Phone: 00 33 6 54 63 42 74
  - Address: 3 Rue du Kiwi, Lyon.
- Yves Kone
  - mail: yves.kone@ens-lyon.fr
  - Phone: 00 33 6 55 65 45 75
  - Address: 3 Avenue du buffle, Lyon.