BUDAlloc: Defeating Use-After-Free Bugs by Decoupling Virtual Address Management from Kernel

Junho Ahn, Jaehyeon Lee, Kanghyuk Lee, Wooseok Gwak, Minseong Hwang, and Youngjin Kwon, *KAIST*

https://www.usenix.org/conference/usenixsecurity24/presentation/ahn

USENIX ‘24

Summary

- One-time allocator (BUDAlloc)
  - User-level allocator
  - Kernel

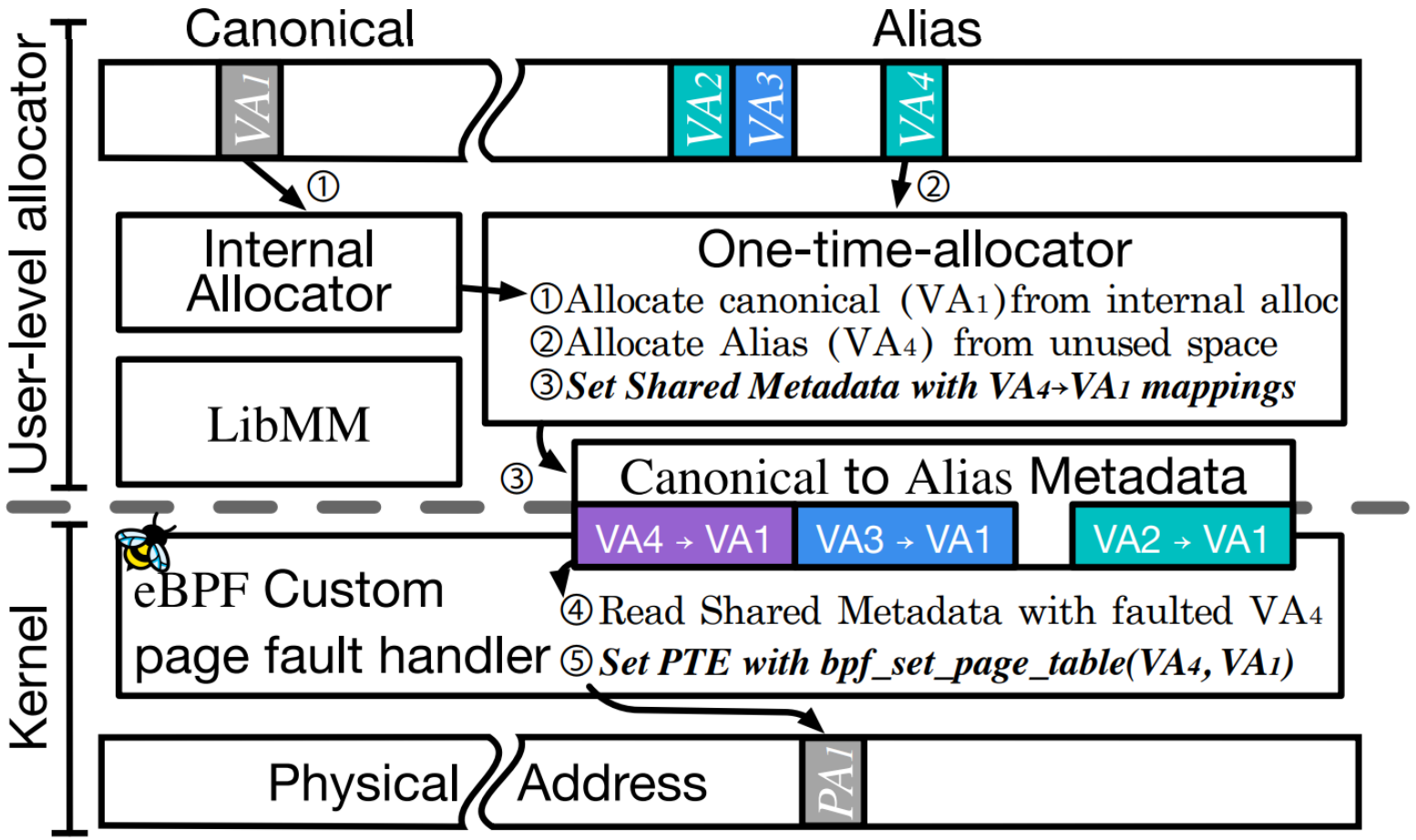
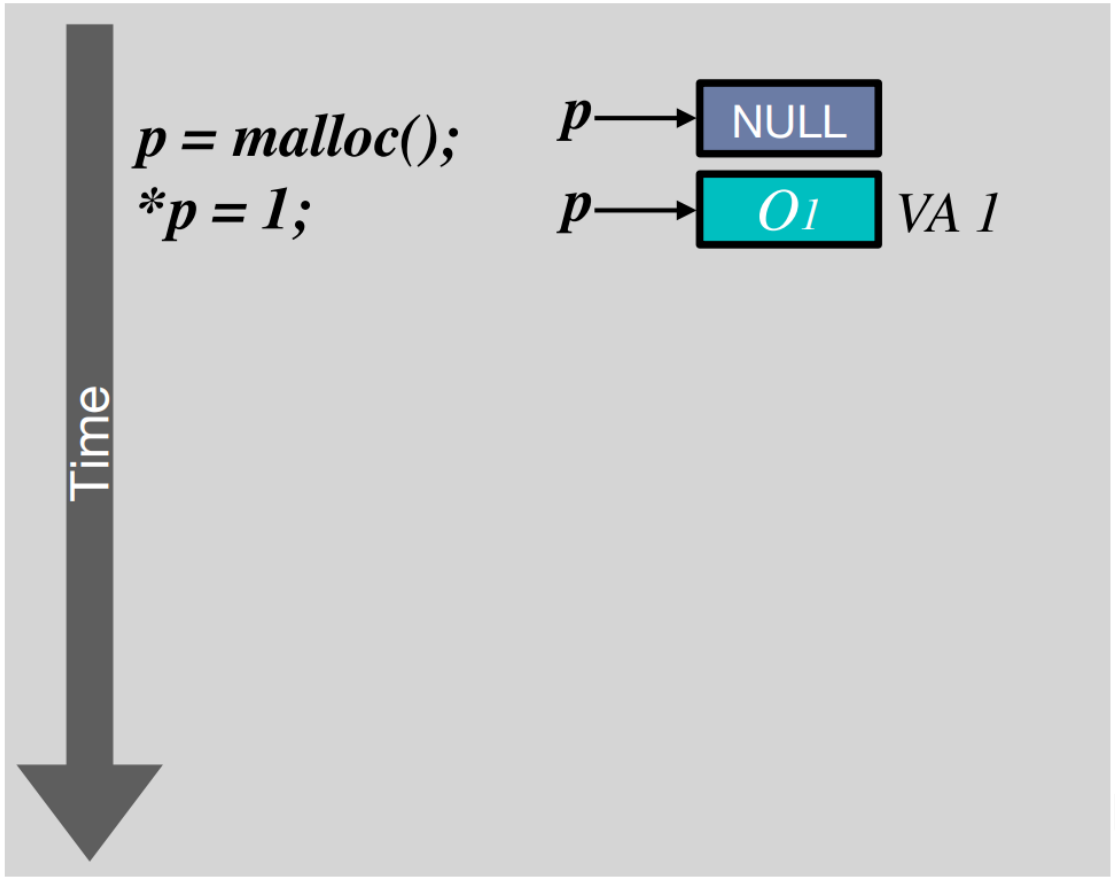
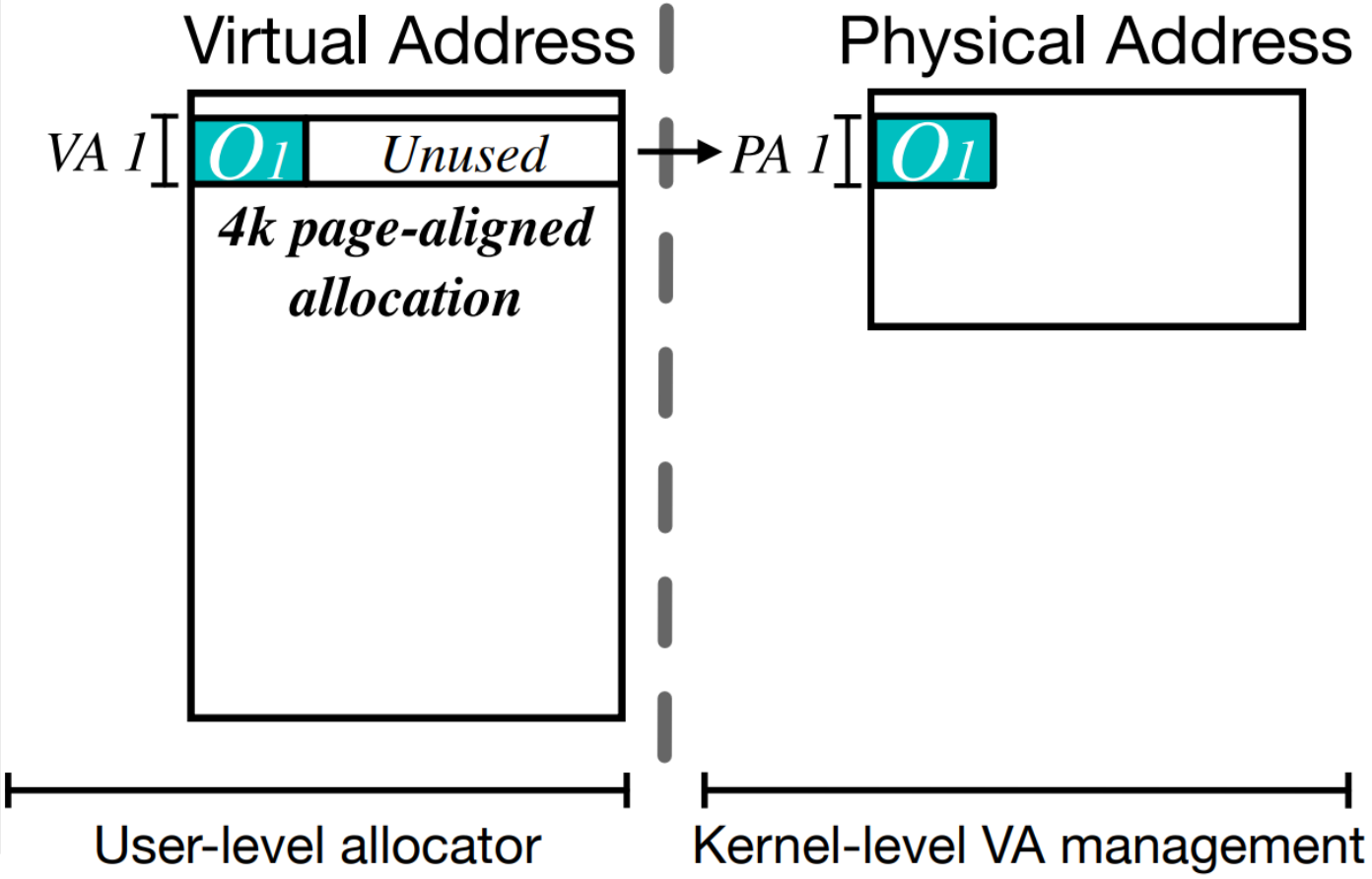


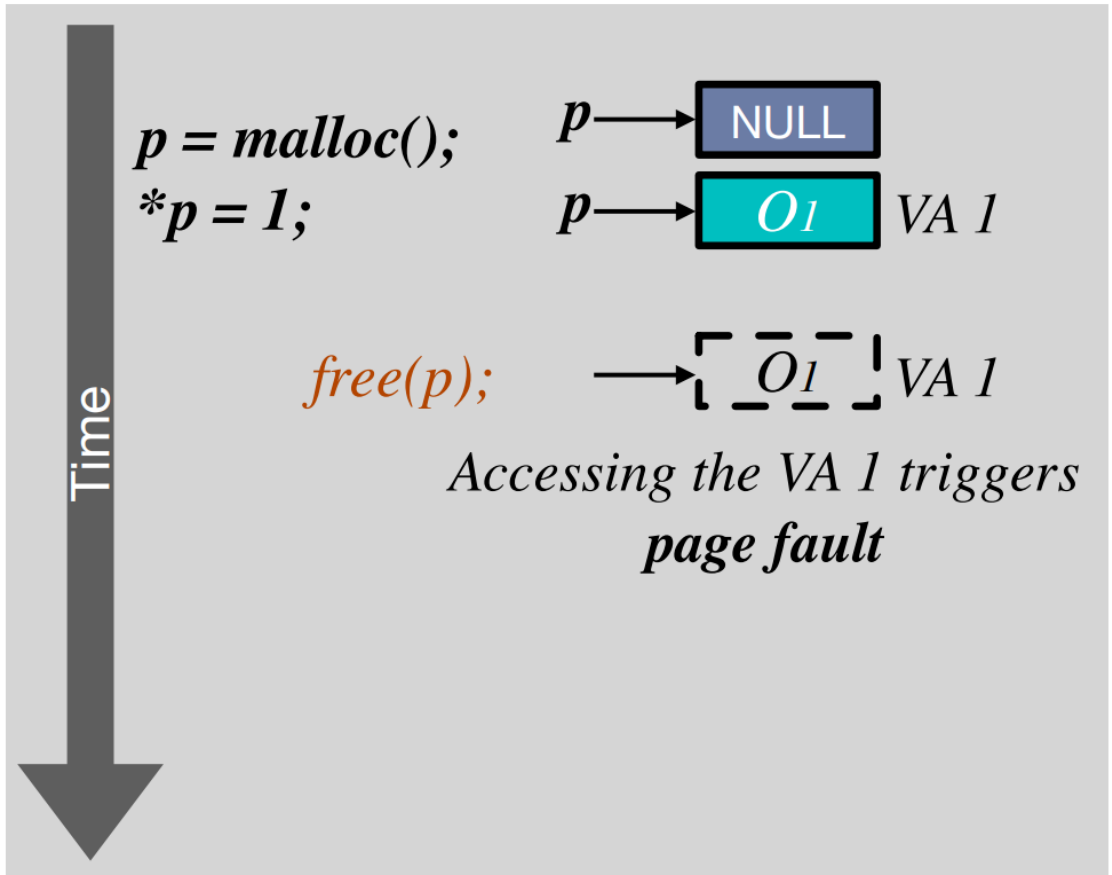
Figure 2: Overview of the BUDAlloc one-time-allocator.

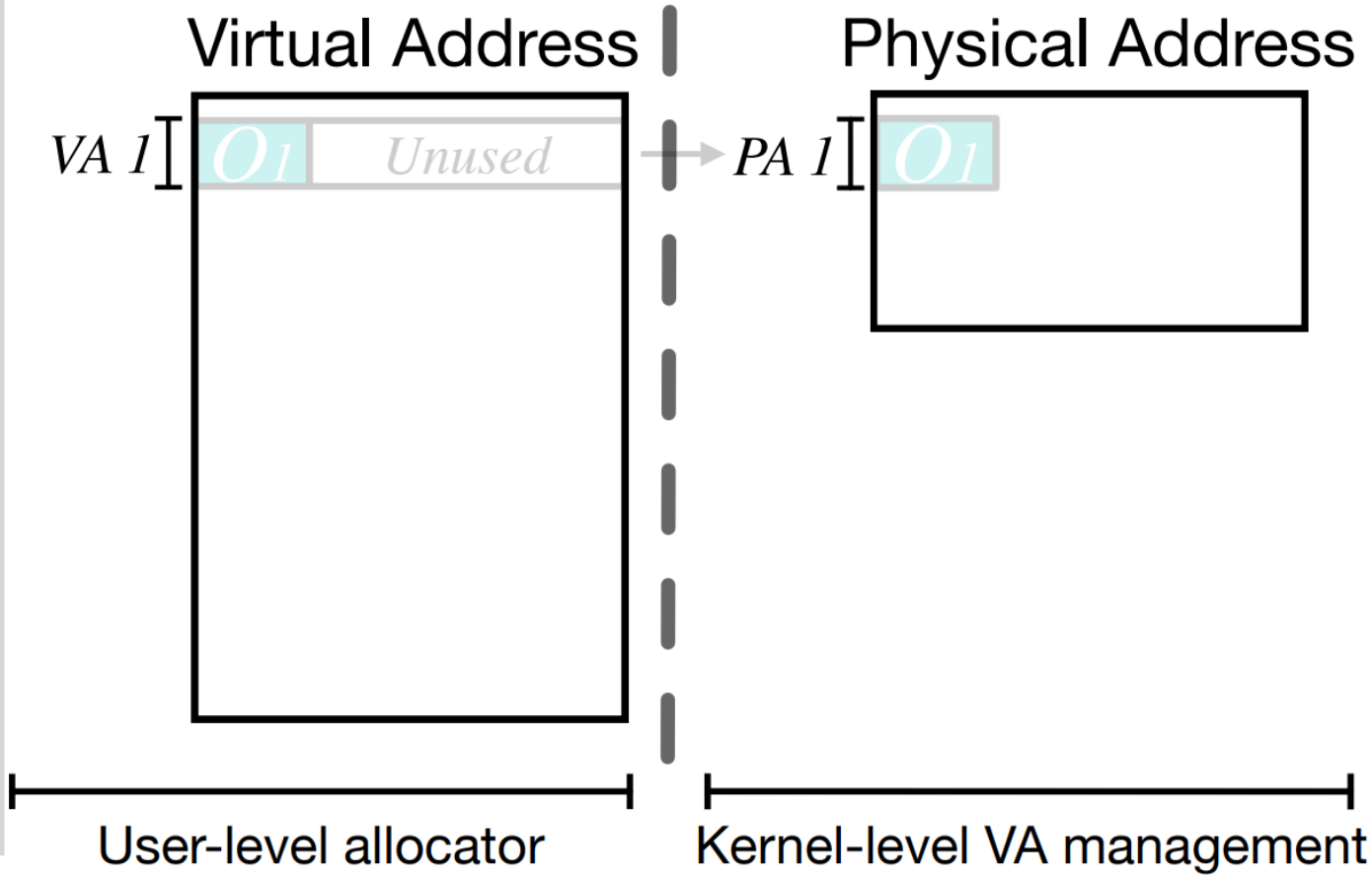
One-Time Allocator (OTA)

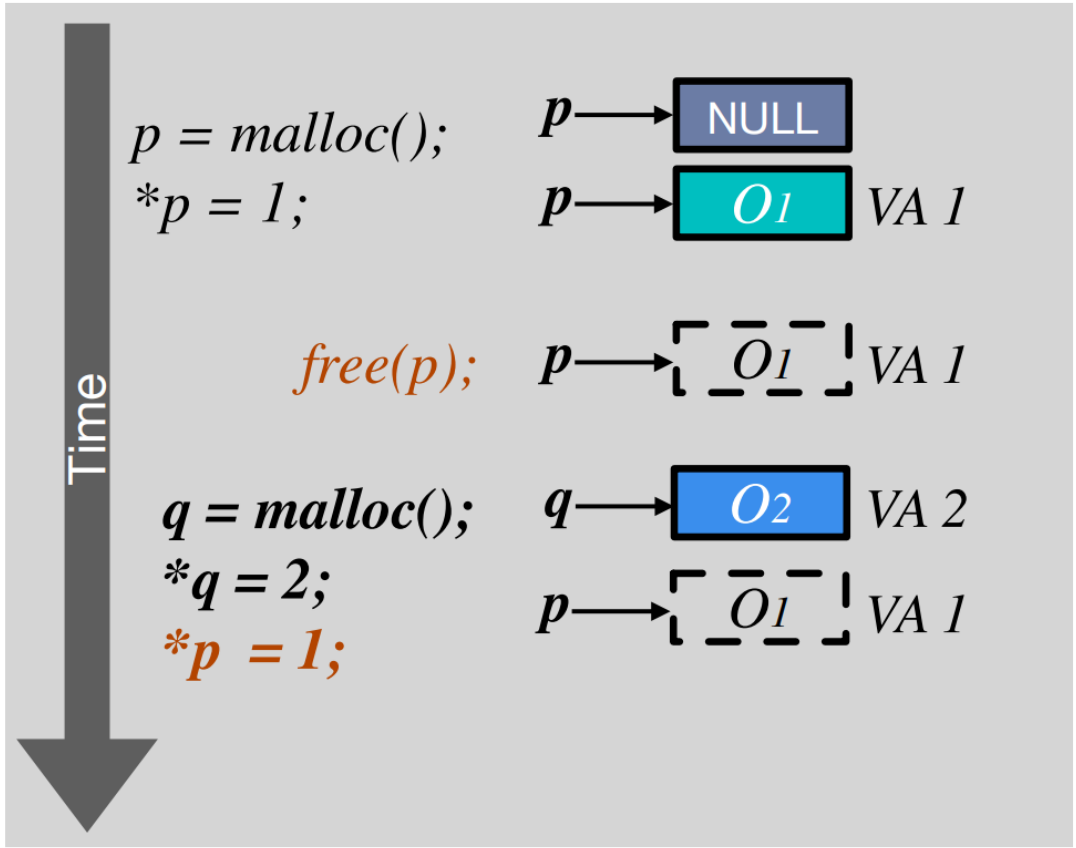
- “Never reusing allocated virtual addresses”

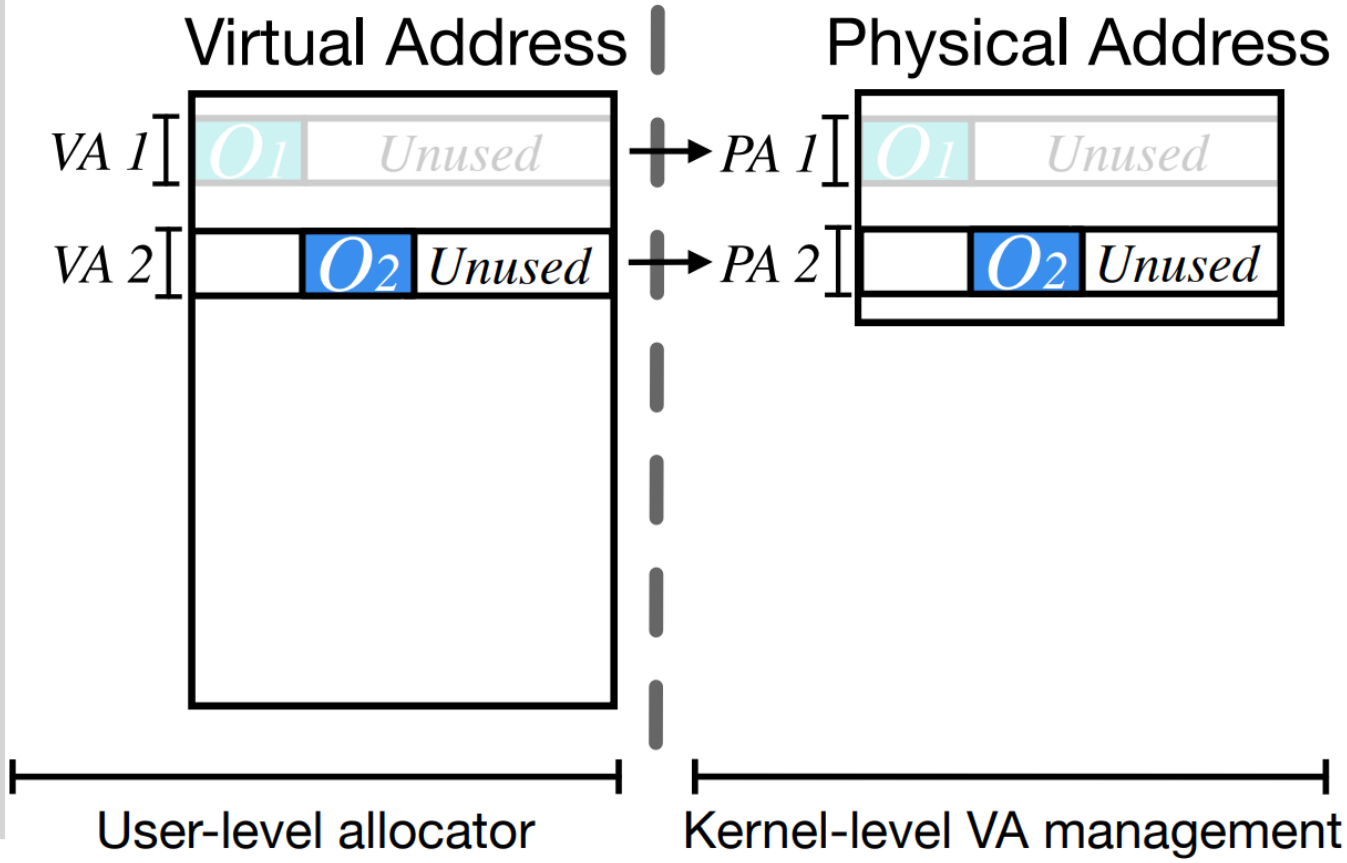





Time
p = malloc();
*p = 1;
p
NULL
p
O1
VA 1
free(p);
O1
VA 1
Accessing the VA 1 triggers
page fault

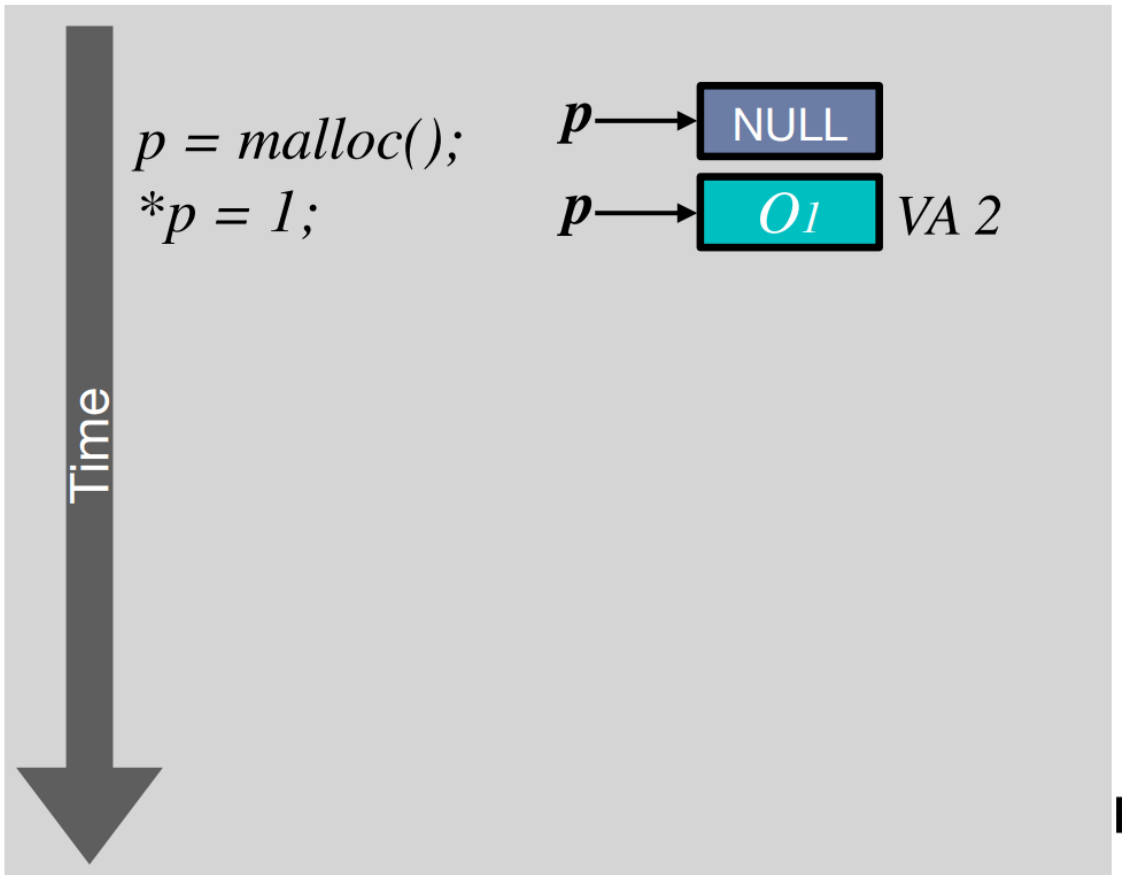

Virtual Address
Physical Address
VA 1
O1
Unused
PA 1
O1
User-level allocator
Kernel-level VA management

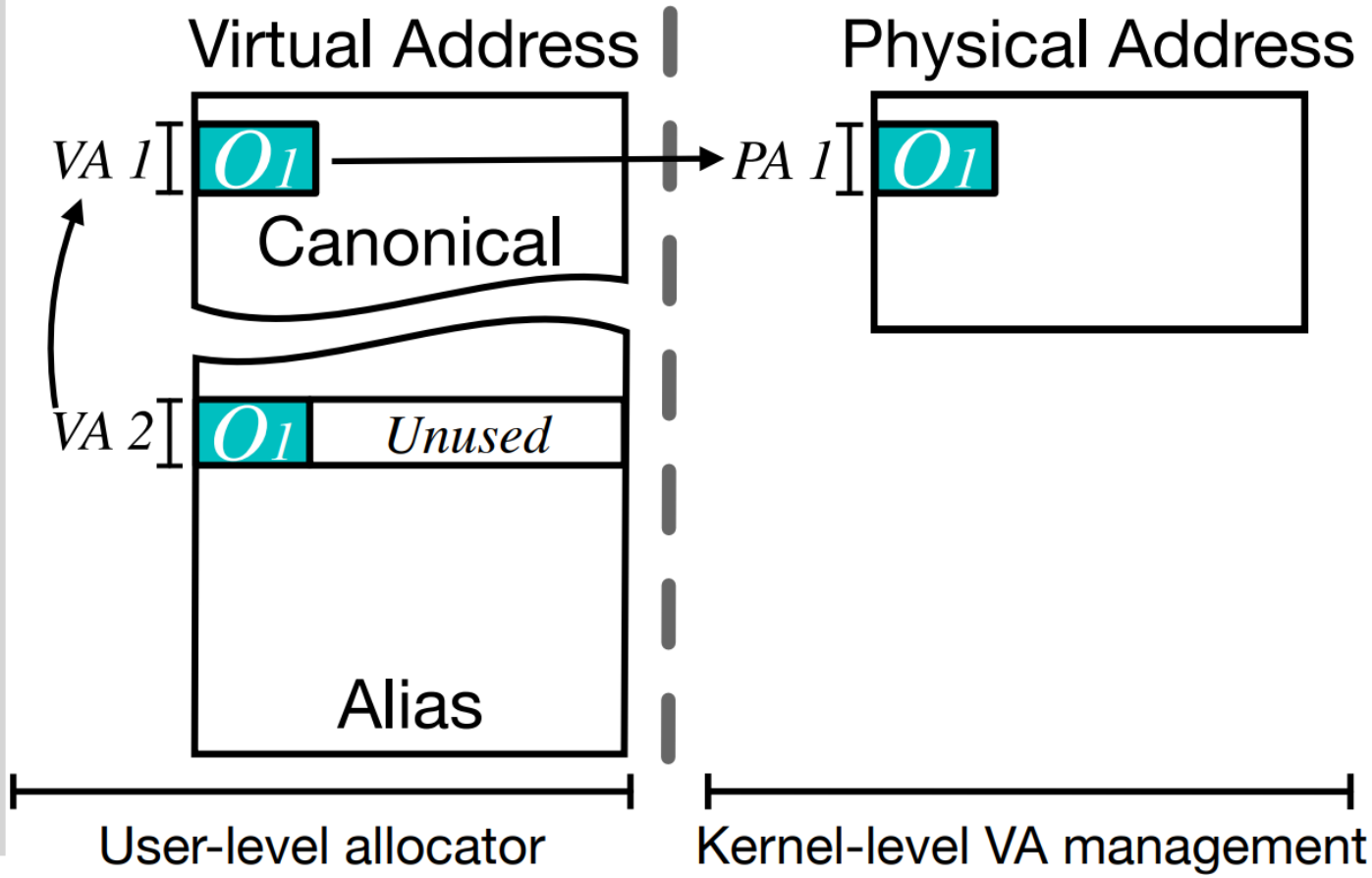

Time
p = malloc();
*p = 1;
p
NULL
p
O1
VA 1
free(p);
p
O1
VA 1
q = malloc();
*q = 2;
q
O2
VA 2
p
O1
VA 1
*p = 1;

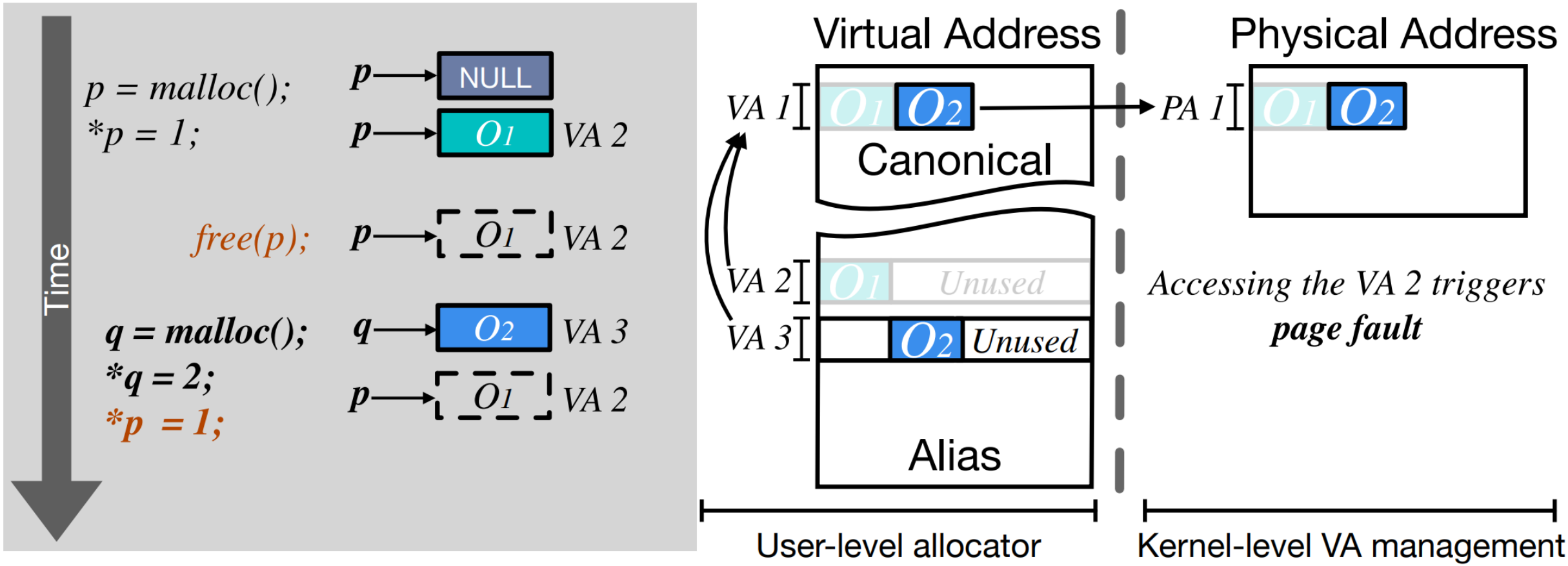

Virtual Address
Physical Address
VA 1
O1
Unused
PA 1
O1
Unused
VA 2
O2
Unused
PA 2
O2
Unused
User-level allocator
Kernel-level VA management

Problem

- Incurs high memory overhead (Fragmentation)
- Solution:
  - Virtual Aliasing


Time
p = malloc();
*p = 1;
p
NULL
p
O1
VA 2


Virtual Address
Physical Address
VA 1
O1
Canonical
PA 1
O1
VA 2
O1
Unused
Alias
User-level allocator
Kernel-level VA management


Time
p = malloc();
*p = 1;
p
NULL
p
O1
VA 2
free(p);
p
O1
VA 2
q = malloc();
*q = 2;
q
O2
VA 3
*p = 1;
p
O1
VA 2
Virtual Address
Physical Address
VA 1
O1
O2
Canonical
PA 1
O1
O2
VA 2
O1
Unused
VA 3
O2
Unused
Alias
Accessing the VA 2 triggers
page fault
User-level allocator
Kernel-level VA management

Problem: Semantic Gap

- Still incurs performance overhead due to system calls.

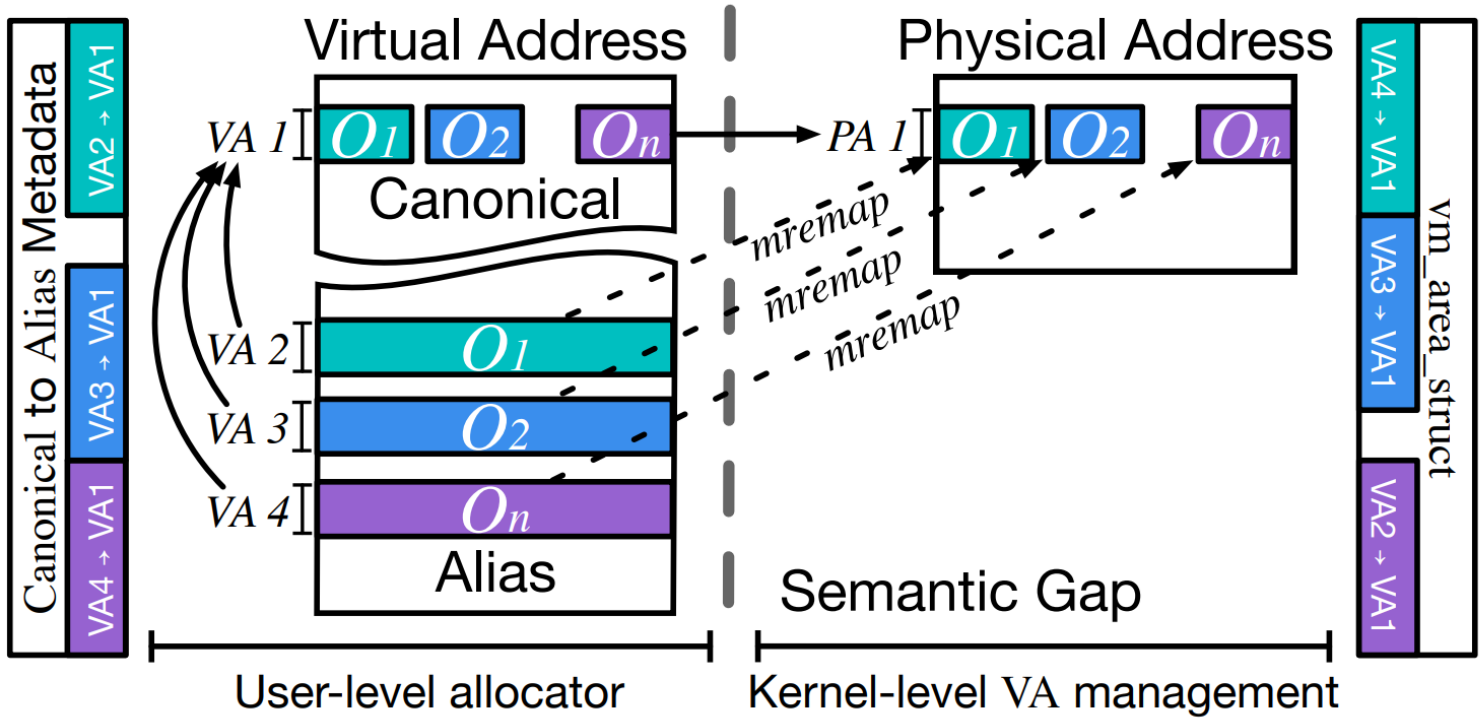


Figure 1: Overview of virtual aliasing.

Existing Approaches

Method	Oscar *USENIX Security 2017* Batching system calls	FFMalloc *USENIX Security 2021* Remove alias mapping	DangZero *ACM CCS 2022* Library operating system
Pros	+ Moderate memory overhead	+ Fast performance	+ Moderate memory overhead + High bug-detect precision
Cons	- Low performance - Low scalability - Cannot support copy-on-write	- High memory overhead - Low bug-detect precision	- Virtualization overhead - Low scalability - Cannot support copy-on-write - Lack of compatibilities

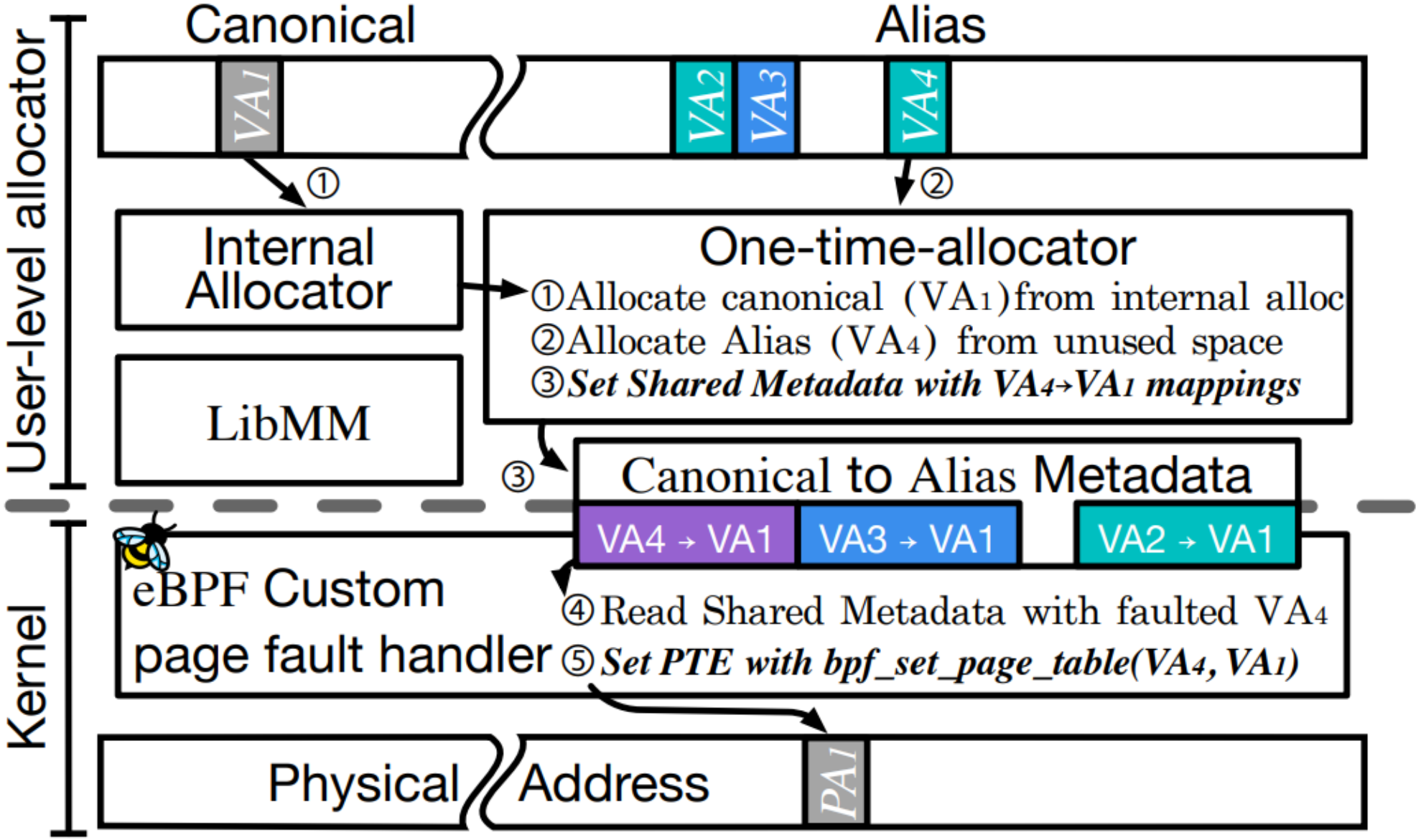


Figure 2: Overview of the BUDAlloc one-time-allocator.

Freeing an Object

- 1. Freeing an object => Essential for detecting UAF
- 2. Requires *unmap* (overhead)
- 3. BUDAlloc defines:
  - BUDAlloc-prevention (postpones freeing an object until next page fault)
  - BUDAlloc-detection (immediately frees an object)

BUDAlloc

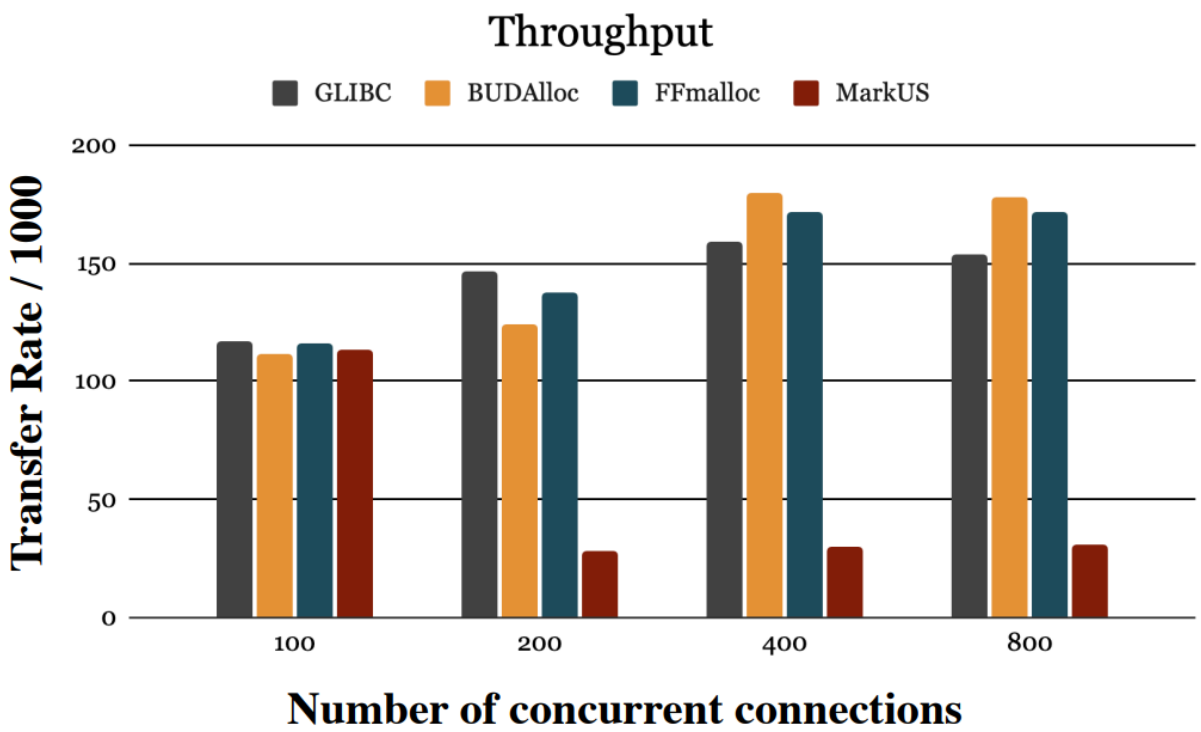
	Memory Bloat	Syscall Overhead	Scalability	Bug-detect Precision	Compatibility
No alias mapping [44]	Very High	Low	Very High	Very Low	Fully Compatible
Syscall-based [42]	Moderate	Very High	Low	Detector	No COW
LibOS-based [25]	Low	VM overhead	Single thread only	Detector	No COW, proc fs, etc
BUDAlloc-detection	Low	Low	Very High	Detector	Fully Compatible
BUDAlloc-prevention	Low	Very Low	Very High	High	Fully Compatible

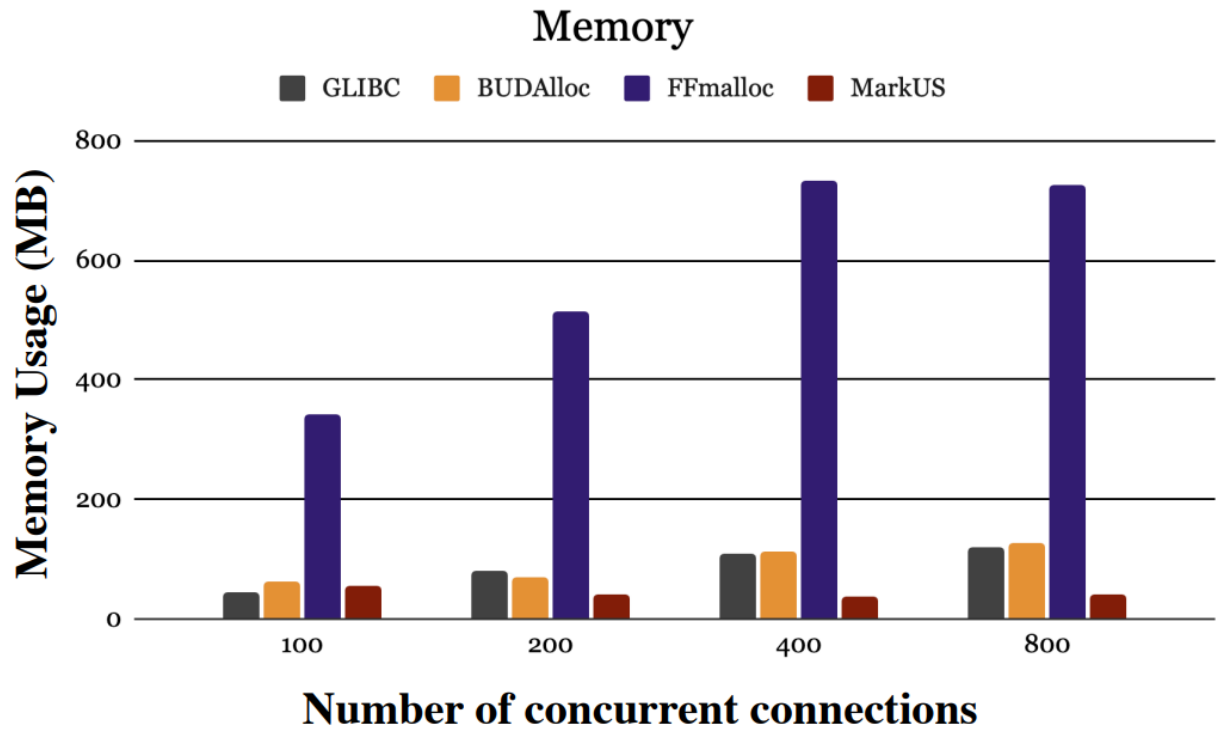
Table 1: Comparisons of previous OTAs.

Vulnerability	Program	BUDAlloc-p	BUDAlloc-d	FFmalloc	DangZero
UAFBench					
CVE-2016-3189	bzip2	●	●	◑	●
*CVE-2016-4487	cxxfilt	●	●	●	●
CVE-2017-10686	nasm	●	●	◑	●
CVE-2018-10685	lrzip	●	●	◑	●
CVE-2018-11496	lrzip	●	●	◑	●
*CVE-2018-11416	jpegoptim	●	●	●	●
CVE-2018-20623	readelf	●	●	◑	●
*CVE-2019-20633	patch	●	●	●	●
*CVE-2019-6455	rec2csv	●	●	●	●
Issue 74	giflib	●	●	◑	●
*Issue 122	gifsicle	●	●	●	●
Issue 73	mjs	●	●	◑	●
Issue 78	mjs	●	●	◑	●
Issue 91	yasm	●	●	◑	●
ffmalloc & DangZero					
CVE-2015-2787	PHP	●	●	◑	●
*CVE-2015-3205	libmimedir	●	●	●	●
CVE-2015-6835	PHP	●	●	◑	●
CVE-2016-5773	PHP	●	●	◑	●
Issue 3515	mruby	●	●	◑	●
Issue 24613	Python	●	●	◑	●
Exploit Database					
CVE-2019-6076	Lua	●	●	◑	●
CVE-2019-7703	Binaryen	●	●	◑	●
CVE-2019-8343	nasm	●	●	◑	●
CVE-2019-17582	libzip	●	●	◑	●
CVE-2020-24346	nginx	●	●	◑	●
CVE-2022-1934	mruby	●	●	◑	●
CVE-2022-1106	mruby	◑	●	◑	●
CVE-2022-35164	LibreDWG	●	●	◑	●
*BUG-66783	PHP	●	●	●	●
BUG-80927	PHP	●	●	◑	●
	●: Detect UAF bug		◑: Prevent UAF bug		

Performance of BUDAlloc

System	SPEC CPU 2006		SPEC CPU 2017	
	Perf.	Mem	Perf.	Mem
BUDAlloc-p	1.11×	1.31×	1.18×	1.24×
BUDAlloc-d	1.16×	1.25×	1.23×	1.20×
DangZero	1.28×	1.24×	1.31×	1.27×
FFmalloc	1.01×	2.08×	1.01×	1.90×
MarkUs	1.16×	1.27×	1.17×	1.28×


Throughput
GLIBC
BUDAlloc
FFmalloc
MarkUS
Transfer Rate / 1000
200
150
100
50
0
100
200
400
800
Number of concurrent connections


Memory
GLIBC
BUDAlloc
FFmalloc
MarkUS
Memory Usage (MB)
800
600
400
200
0
100
200
400
800
Number of concurrent connections

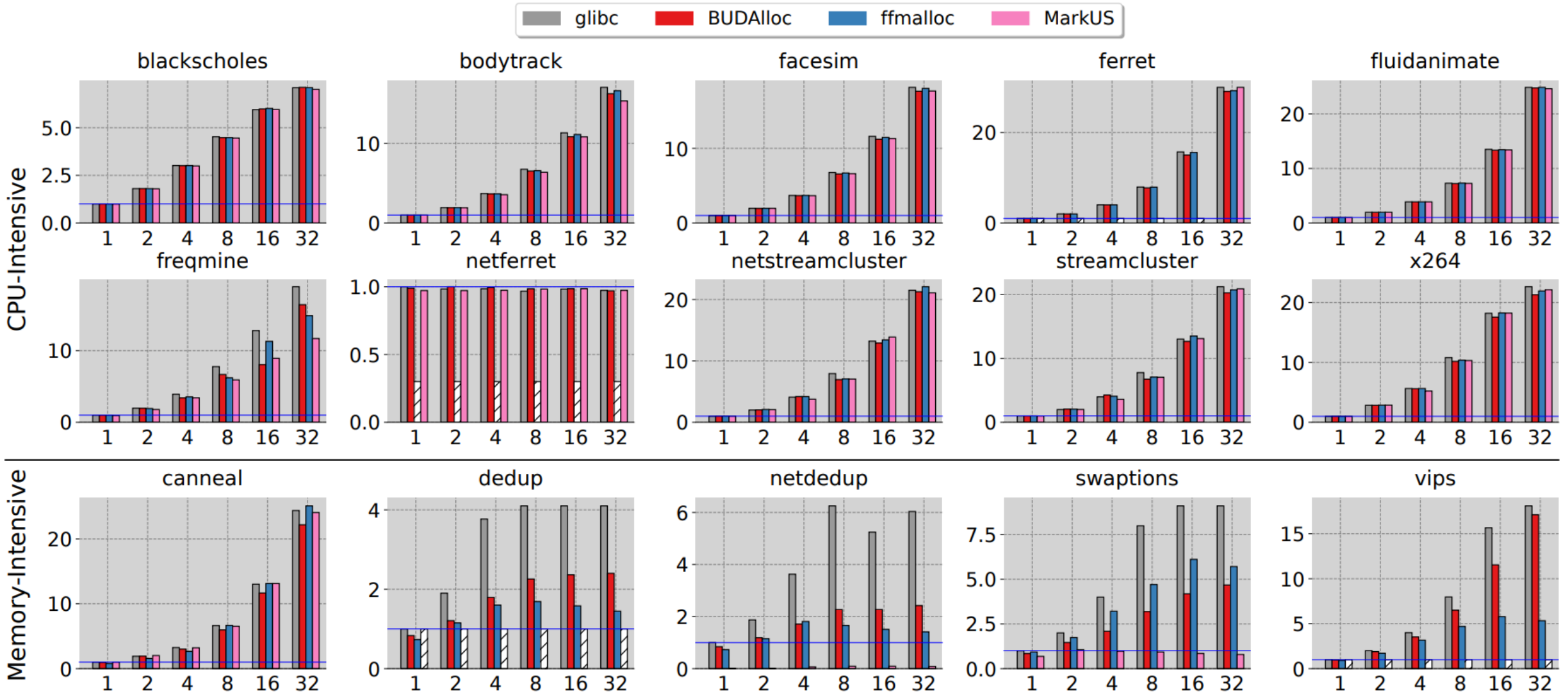
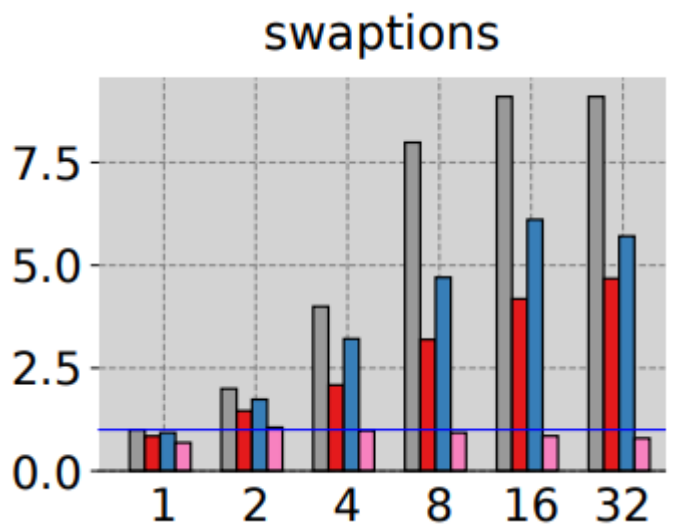


Figure 6: Speedups of PARSEC 3.0 based on the number of threads (higher is better). Performance is normalized to the GLIBC single thread. A white bar indicates that a specific allocator did not run.

Scalability Issue?

- swaptions frequently allocates and free large objects
  - Significant stress on alias to canonical mapping
  - FFmalloc mitigates these overheads by losing bug-detect precision
  - Adding similar configuration,
    - Improved performance by 38%, surpassing FFmalloc by 13%
    - May lead to significant memory overhead (did not use)



Limitation

- TLB Cache Miss
  - Allocates new alias page for each allocation