NDSS SYMPOSIUM

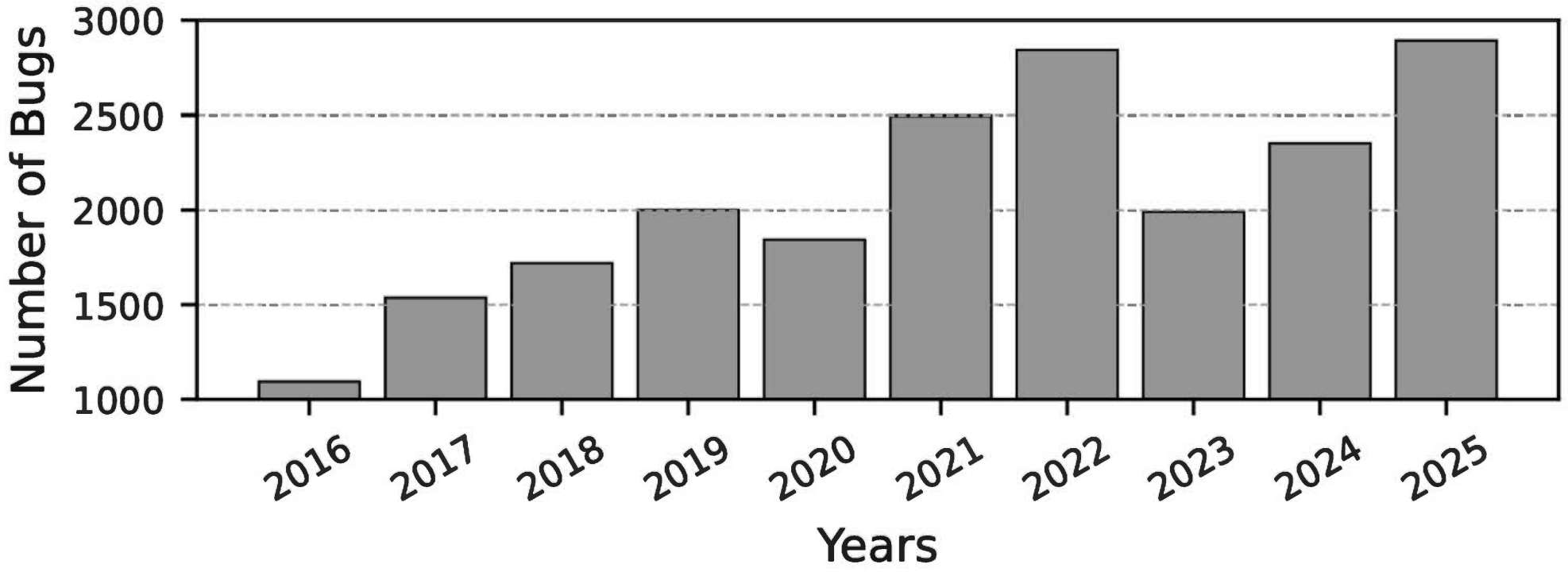
University of Pittsburgh

Fast Pointer Nullification for Use-After-Free Prevention

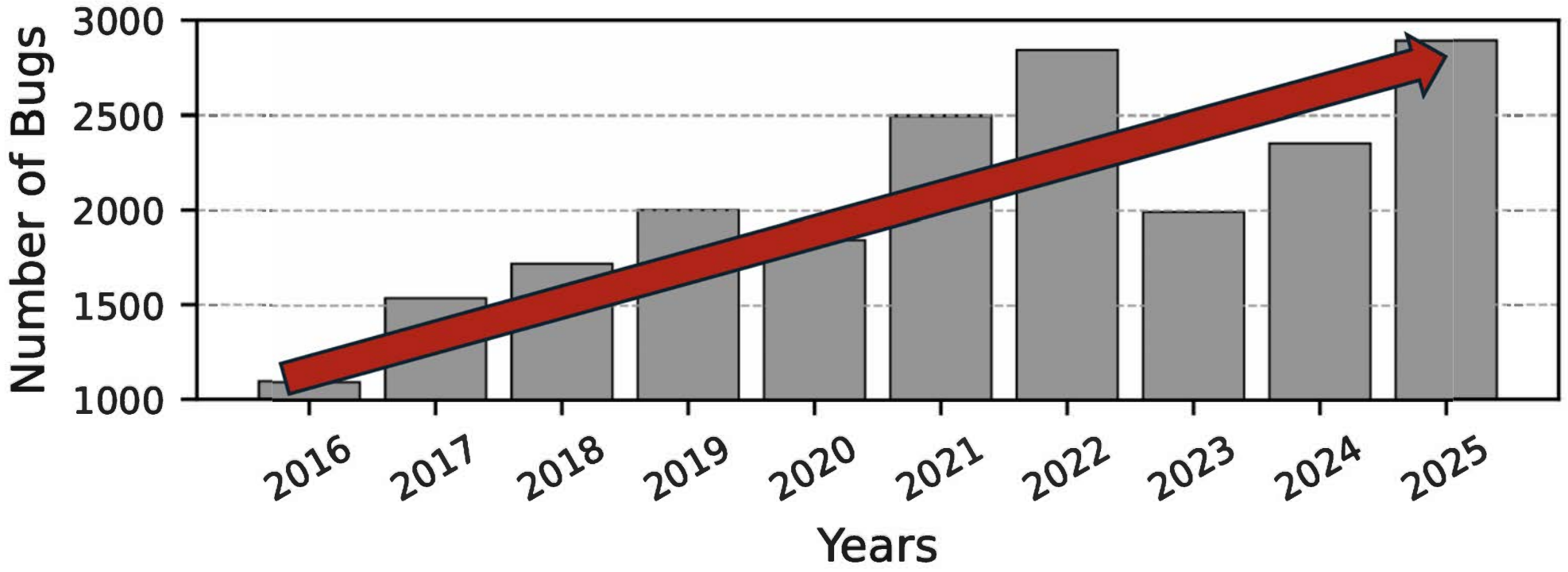
Yubo Du, Youtao Zhang, Jun Yang

University of Pittsburgh

Use-After-Free (UAF) : A Persistent and Growing Security Threat

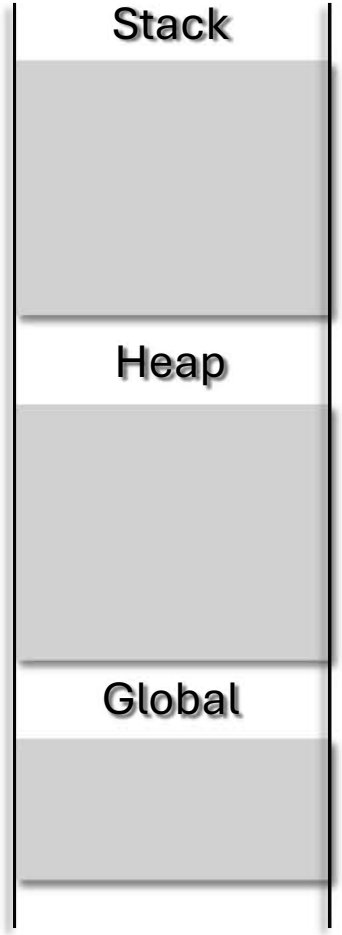


Use-After-Free (UAF) : A Persistent and Growing Security Threat



Root Cause of UAF

```
global char **ptr;
main(){
   char *ptr1 = malloc(32);
   void **ptr2 = malloc(16);
   *ptr2 = ptr1;
   *ptr = ptr1;
   free(ptr1);
}
```



Root Cause of UAF

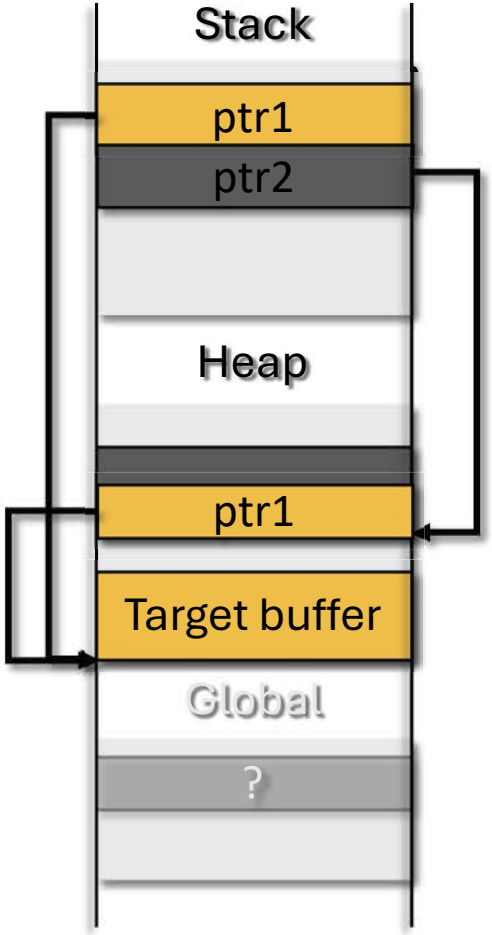
```
global char **ptr;
main(){
   char *ptr1 = malloc(32);
   void **ptr2 = malloc(16) + 8;
   *ptr2 = ptr1;
   *ptr = ptr1;
   free(ptr1);
}
```

Root Cause of UAF

```
global char **ptr;
main(){
  char *ptr1 = malloc(32);
  void **ptr2 = malloc(16);
  *ptr2 = ptr1;
  *ptr = ptr1;
  free(ptr1);
}
```

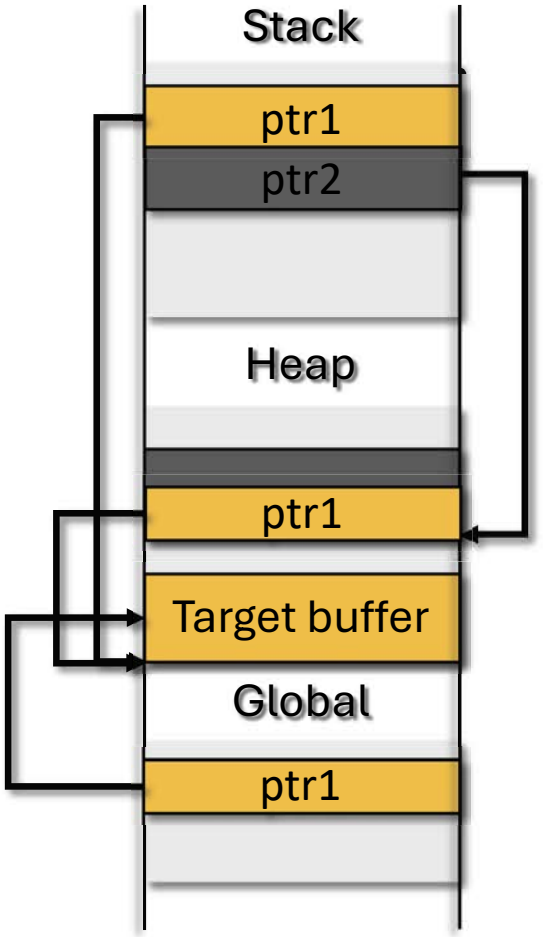
Root Cause of UAF

```
global char **ptr;
main(){
  char *ptr1 = malloc(32);
  void **ptr2 = malloc(16);
  *ptr2 = ptr1;
  *ptr = ptr1;
  free(ptr1);
}
```



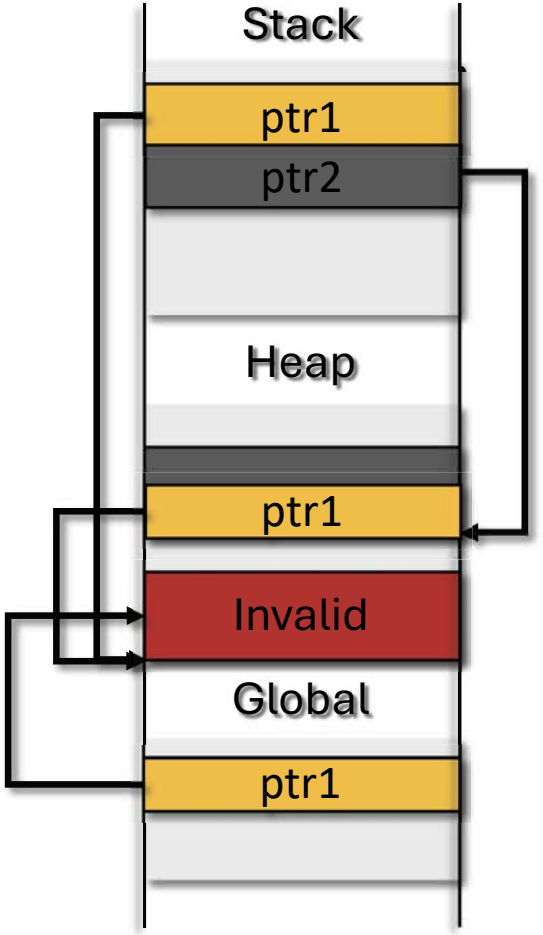
Root Cause of UAF

```
global char **ptr;
main(){
  char *ptr1 = malloc(32);
  void **ptr2 = malloc(16) + 8;
  *ptr2 = ptr1;
  *ptr = ptr1;
  free(ptr1);
}
```



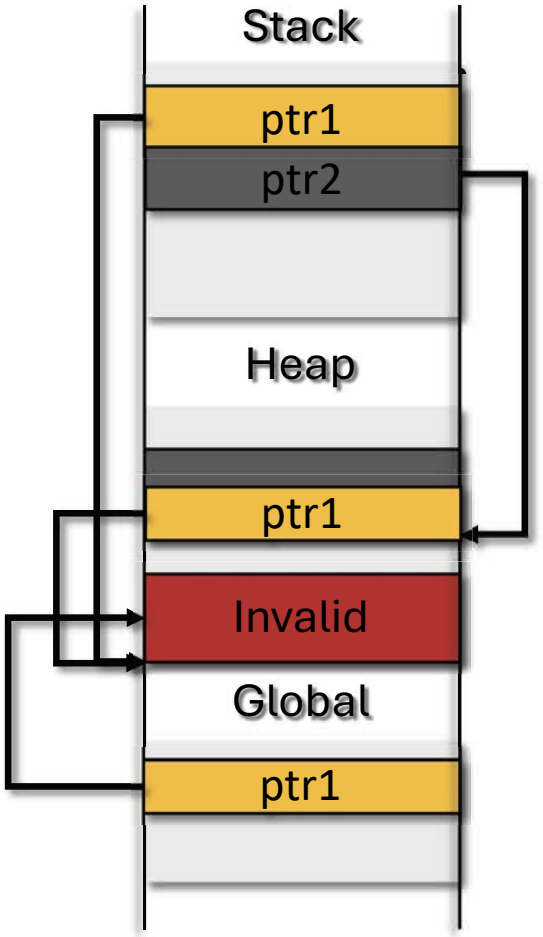
Root Cause of UAF

```
global char **ptr;
main(){
  char *ptr1 = malloc(32);
  void **ptr2 = malloc(16) + 8;
  *ptr2 = ptr1;
  *ptr = ptr1;
  free(ptr1);
}
```



Root Cause of UAF

```
global char **ptr;
main(){
  char *ptr1 = malloc(32);
  void **ptr2 = malloc(16) + 8;
  *ptr2 = ptr1;
  *ptr = ptr1;
  free(ptr1);
}
```

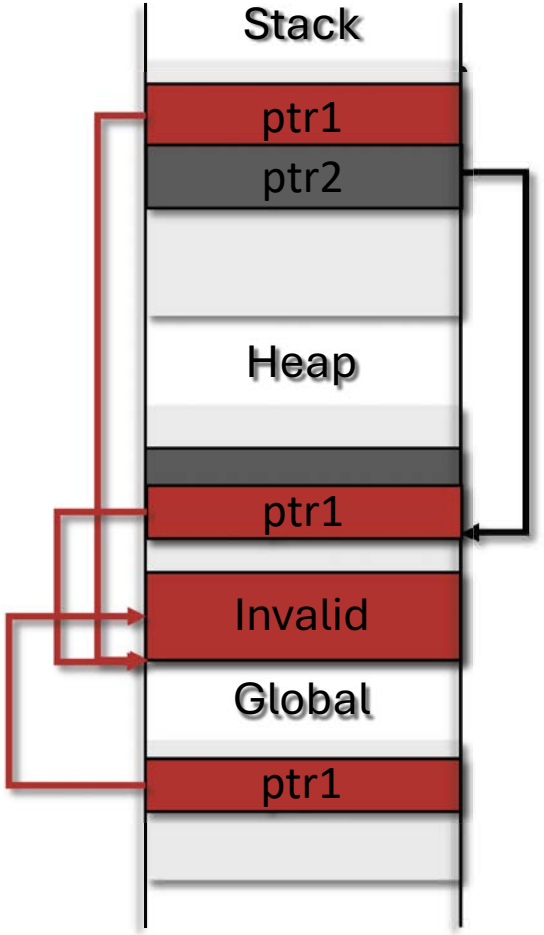


Root Cause of UAF

```
global char **ptr;
main(){
  char *ptr1 = malloc(32);
  void **ptr2 = malloc(16) + 8;
  *ptr2 = ptr1;
  *ptr = ptr1;
  free(ptr1);
}
```

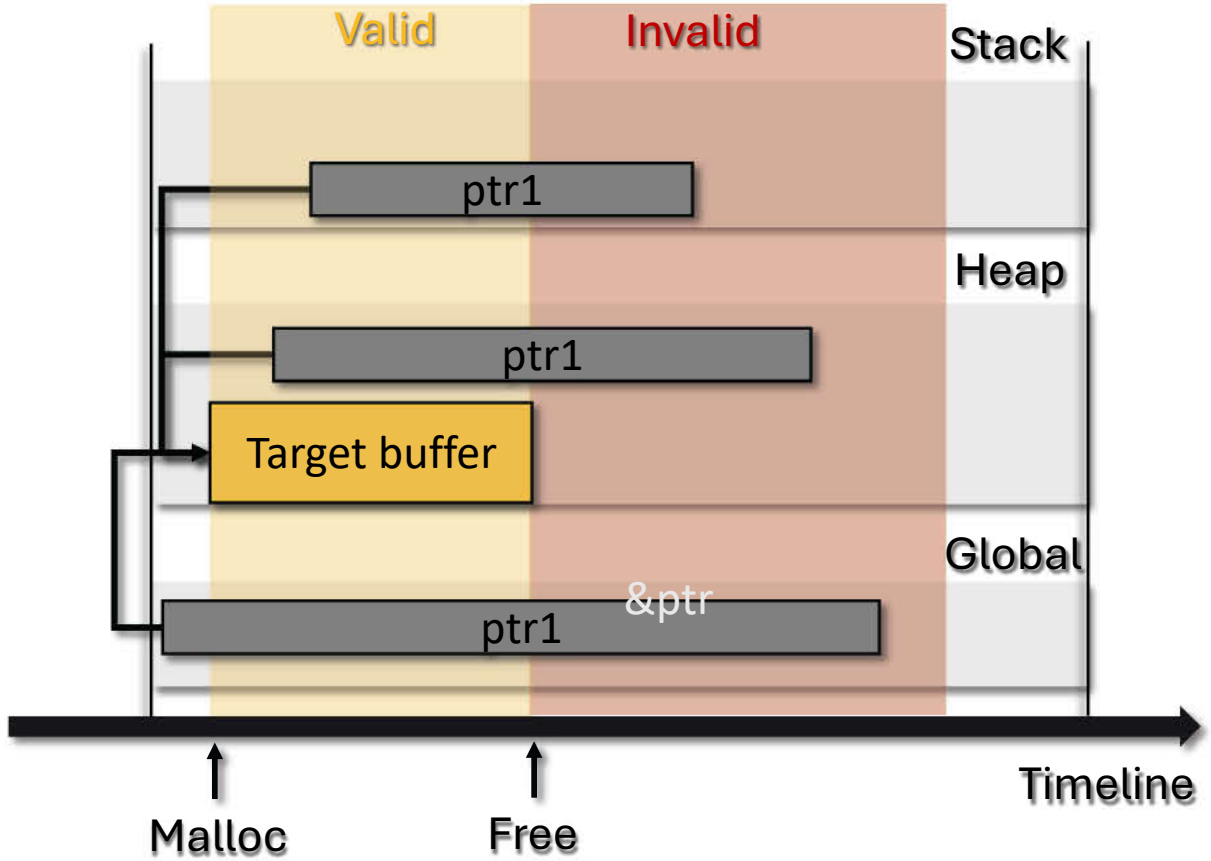
Dangling pointers are still available!

Referencing dangling pointers cause UAF.



Root Cause of UAF

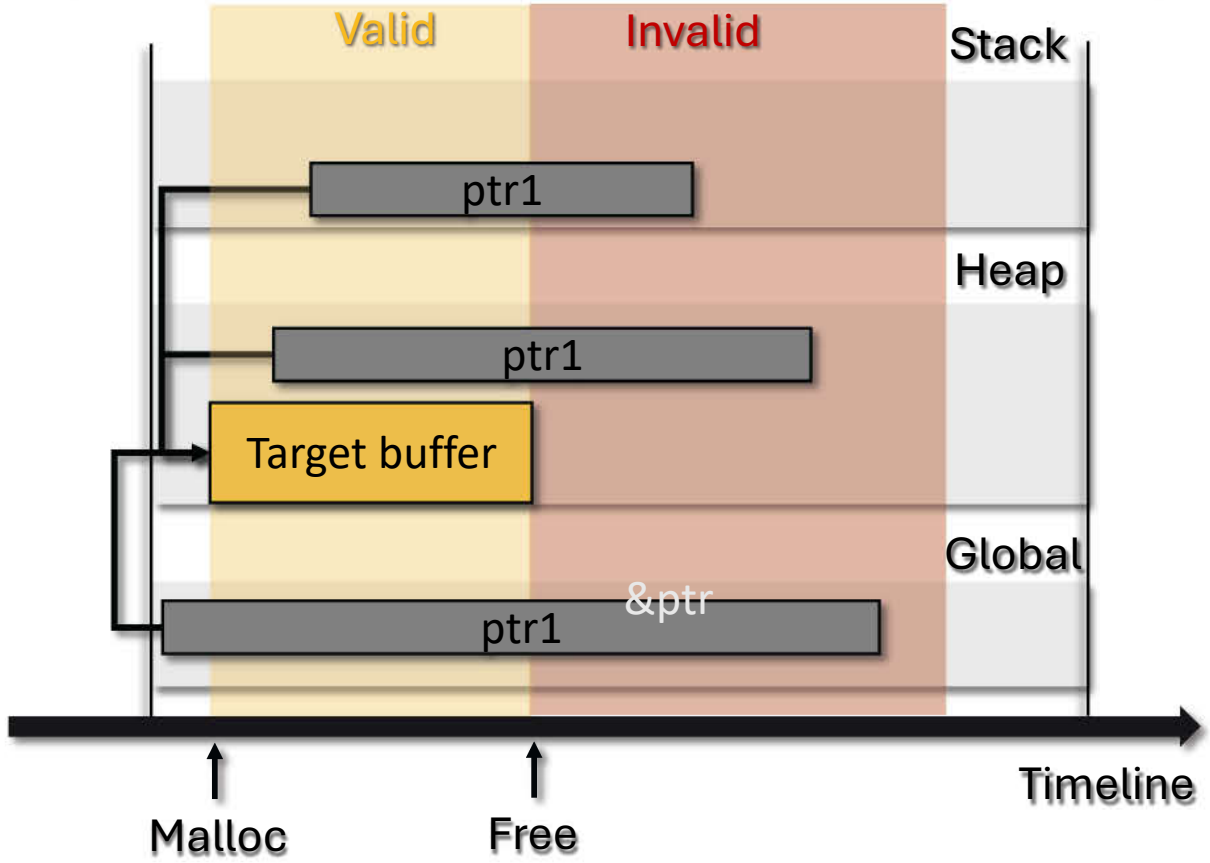
Heap pointers storage lifetime ends **later than** heap buffer lifetime.



Two UAF Preventions

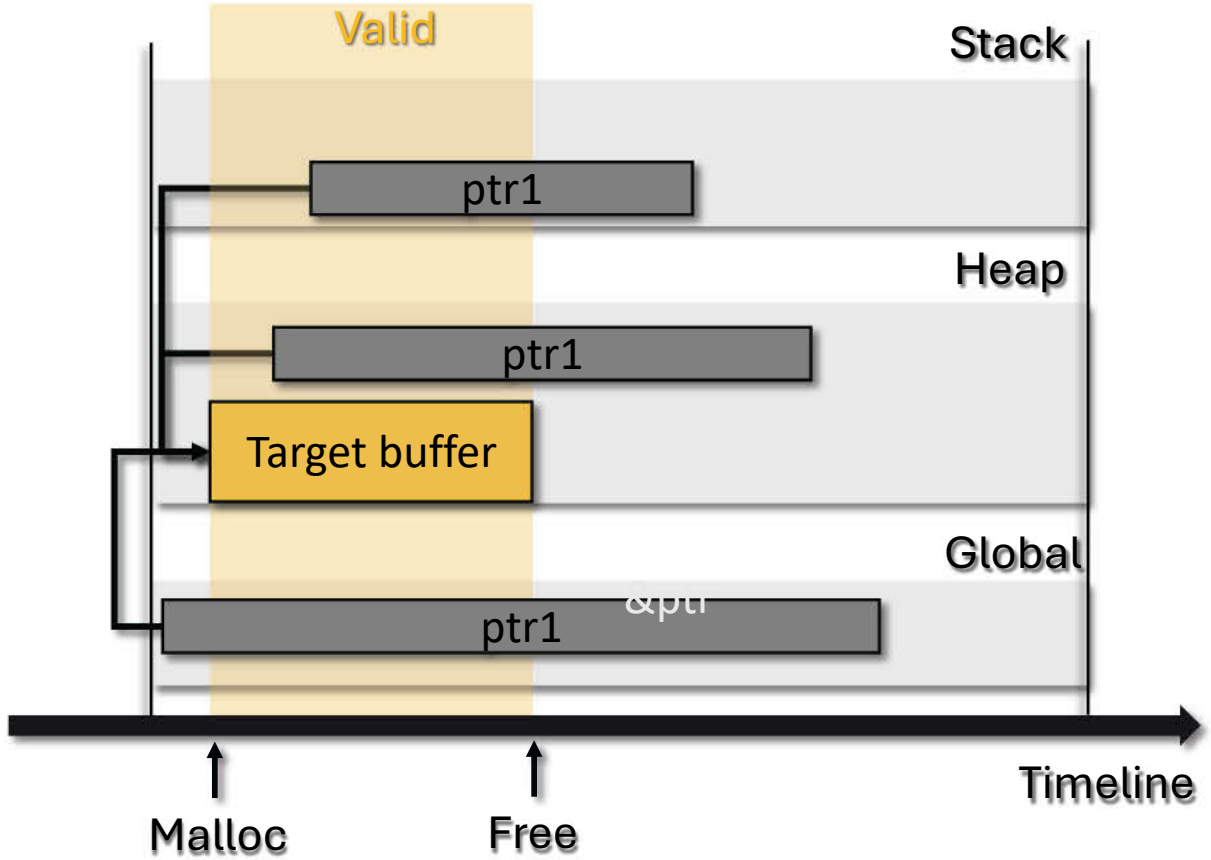
Extending heap buffer lifetime.

Shortening pointers' storage lifetime.



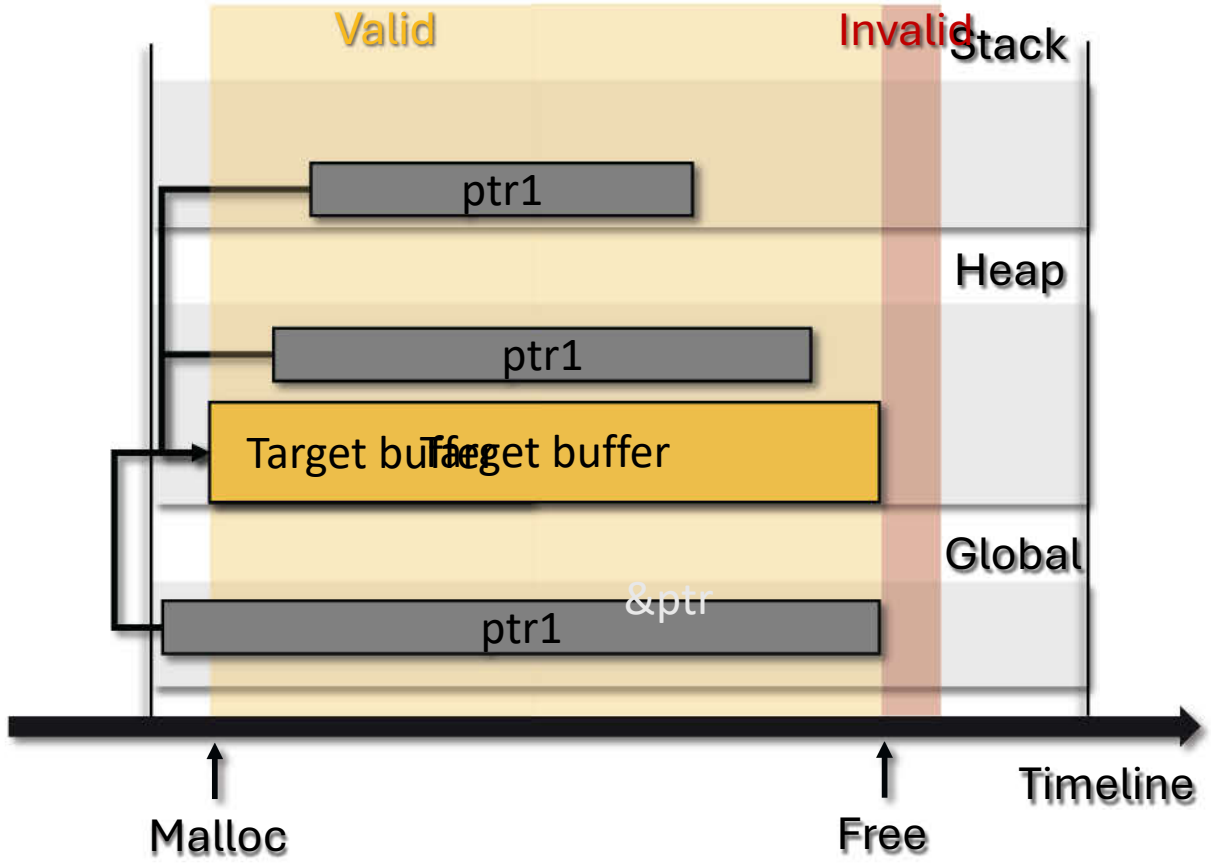
Extending Buffer Lifetime: Garbage Collectors (GCs)

GCs delay the free until no dangling pointers exist.



Extending Buffer Lifetime: Garbage Collectors (GCs)

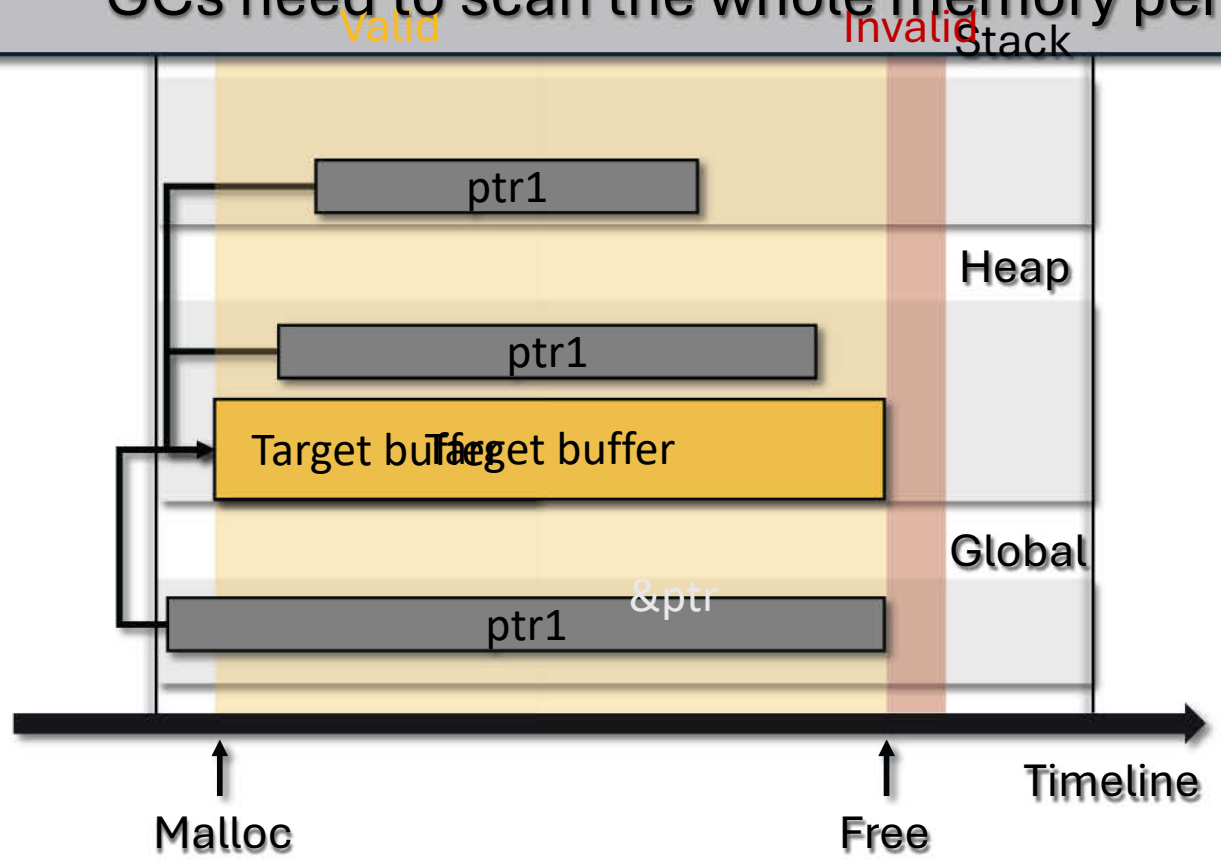
GCs delay the free until no dangling pointers exist.



Extending Buffer Lifetime: Garbage Collectors (GCs)

GCs delay the free until no dangling pointers exist.

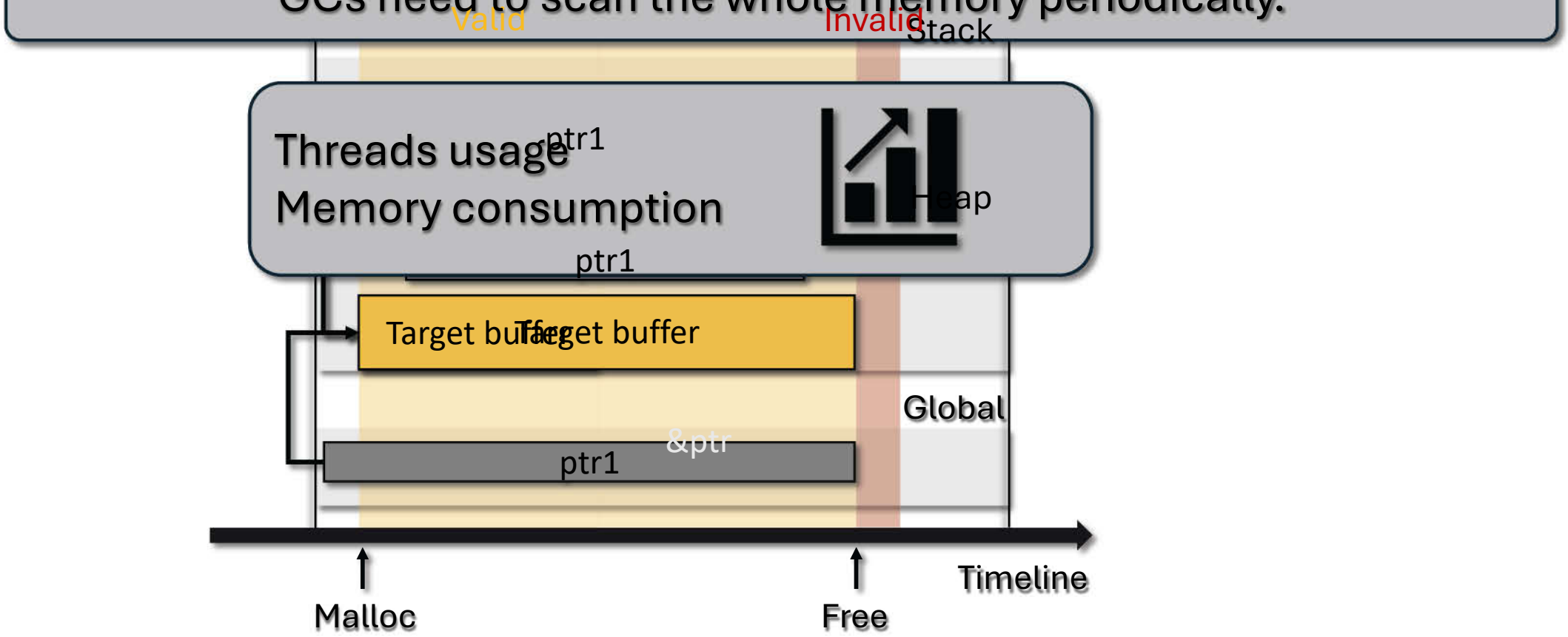
GCs need to scan the whole memory periodically.



Extending Buffer Lifetime: Garbage Collectors (GCs)

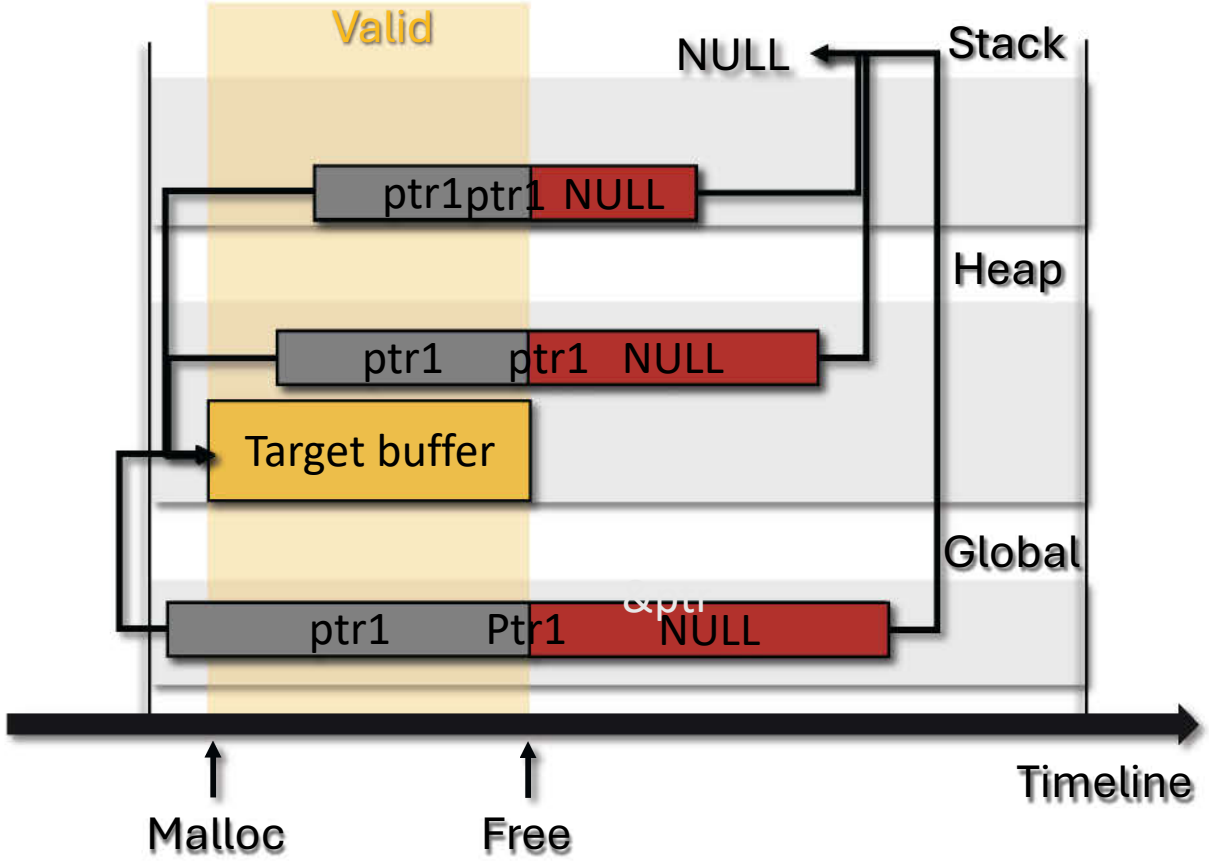
GCs delay the free until no dangling pointers exist.

GCs need to scan the whole memory periodically.



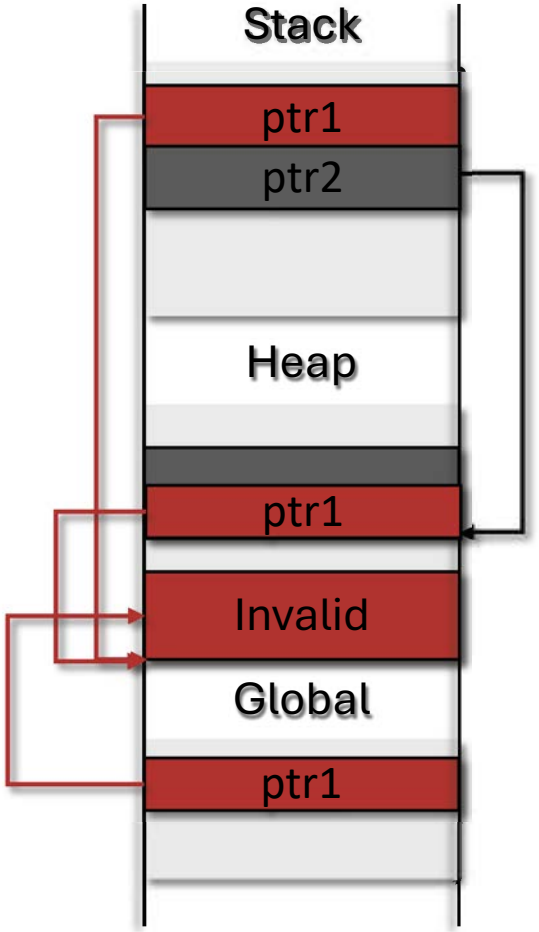
Shortening Pointer Lifetime: Pointer Nullifications (PNs)

PNs prevent UAF by **invalidating** all pointers when the target buffer is freed.



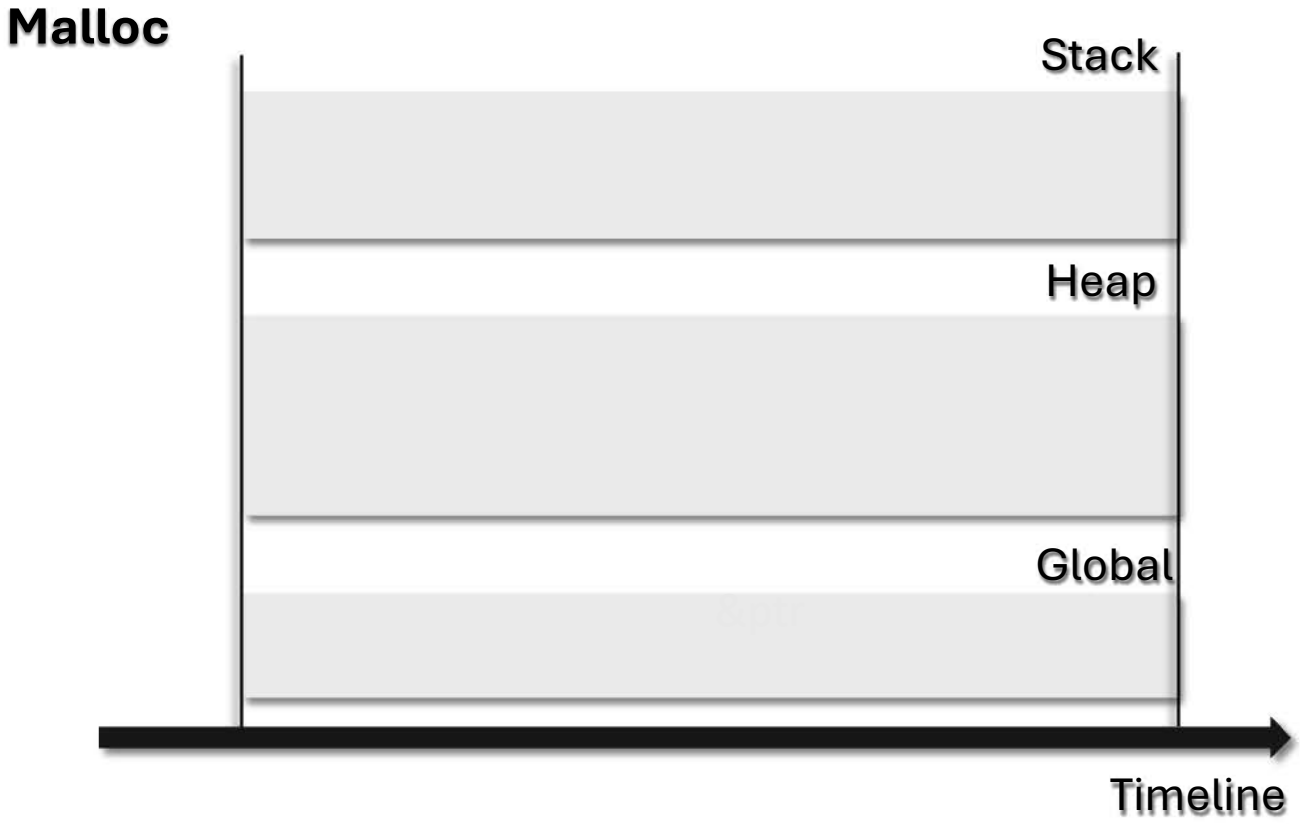
Why PNs Are Hard

- Pointers can be stored **anywhere**
- A buffer can be referenced by **many locations**
- These locations are **not known statically**
- They are **updated dynamically at runtime**



How PNs Work

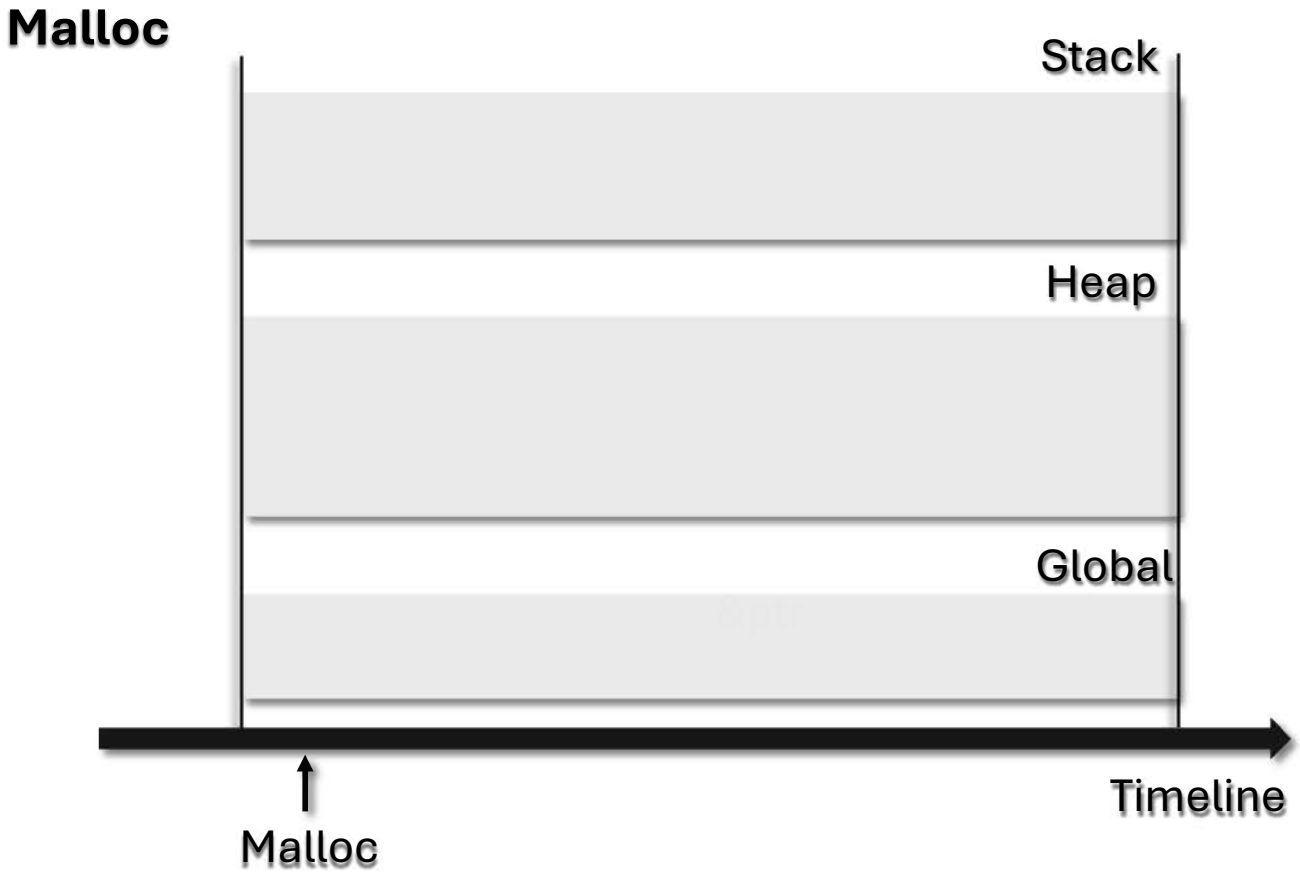
Set up metadata → Register pointer storage locations → Nullify dangling pointers



Metadata of PNs

How PNs Work

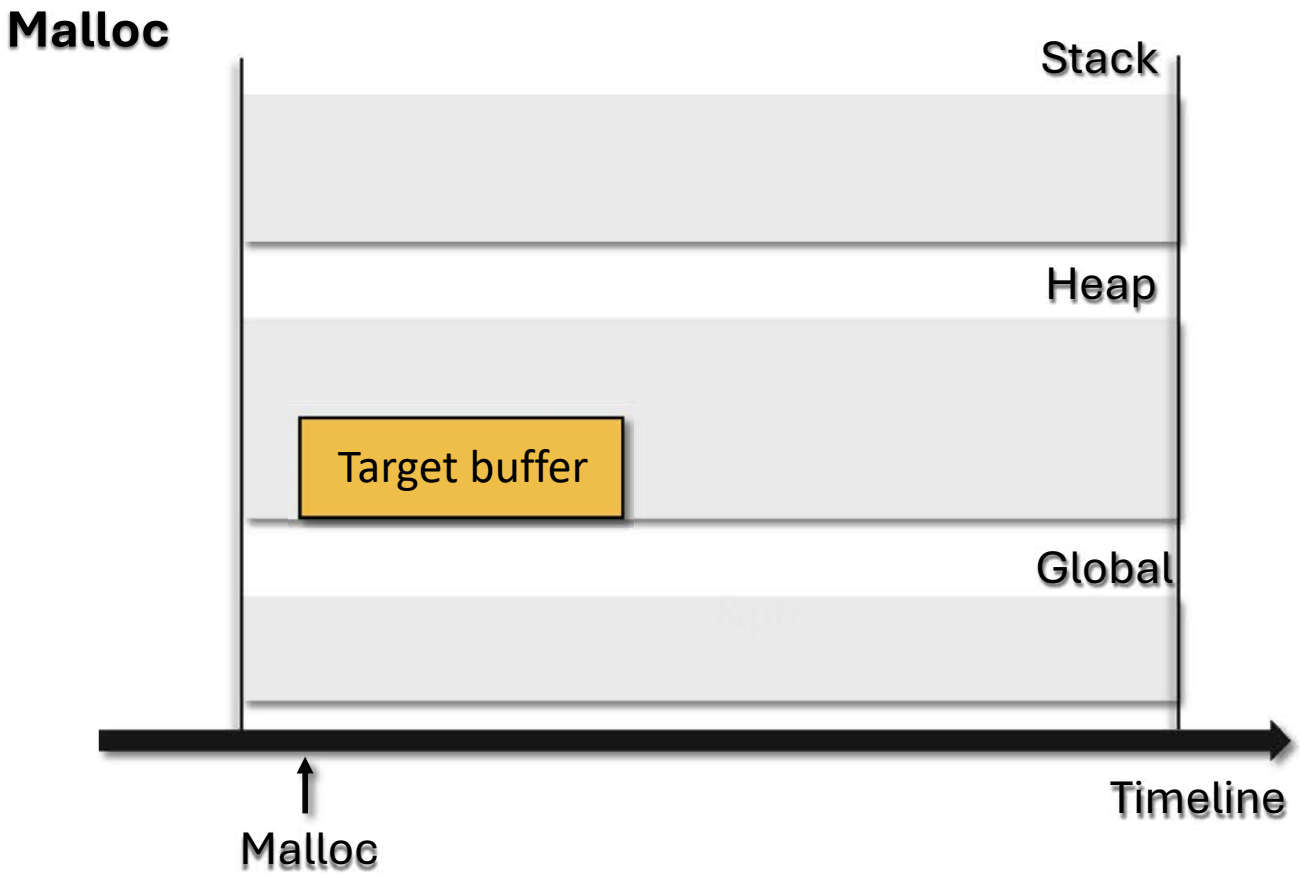
Set up metadata → Register pointer storage locations → Nullify dangling pointers



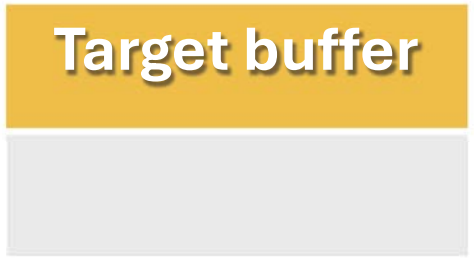
Metadata of PNs

How PNs Work

Set up metadata → Register pointer storage locations → Nullify dangling pointers

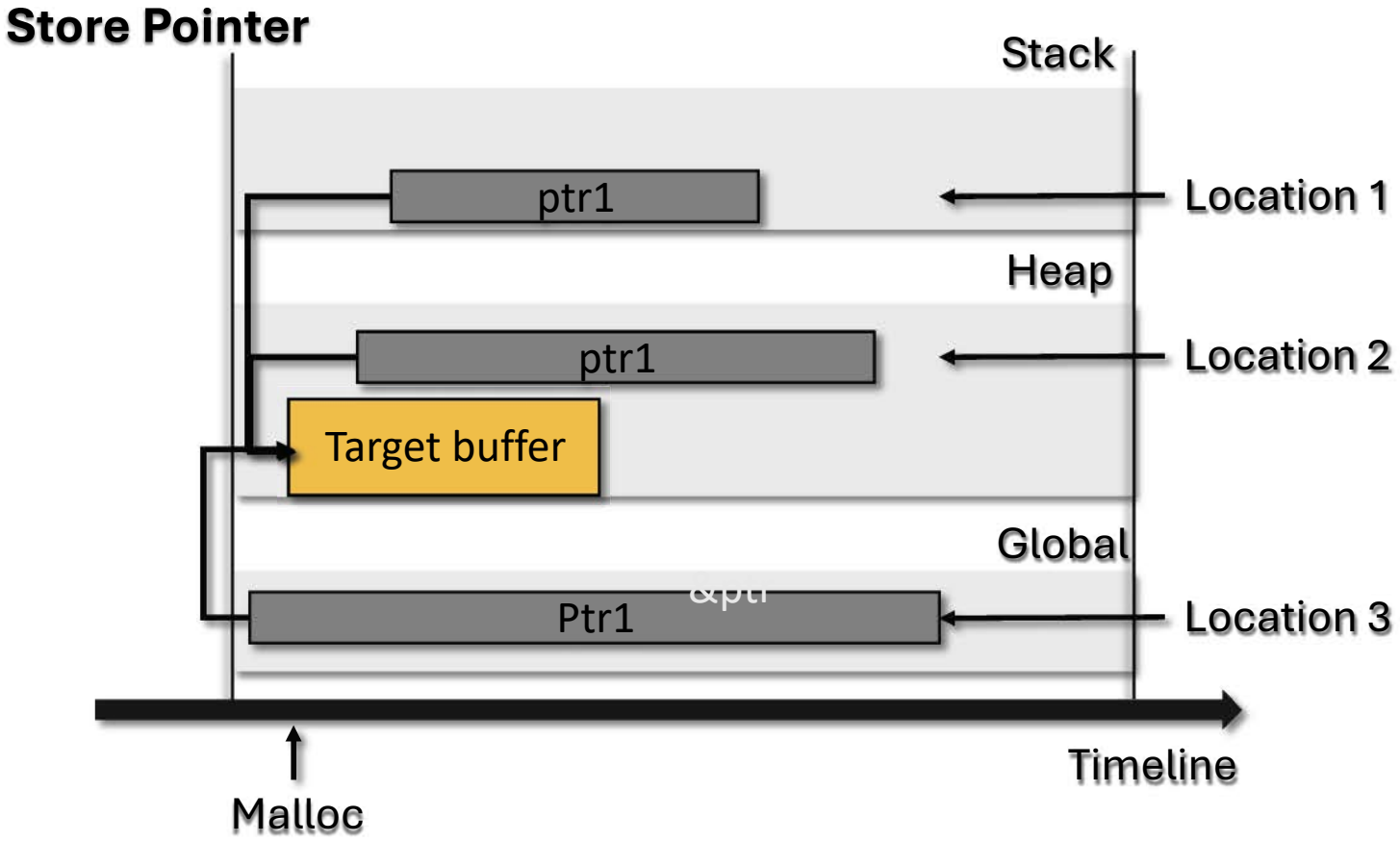


Metadata of PNs



How PNs Work

Set up metadata → **Register pointer storage locations** → **Nullify dangling pointers**

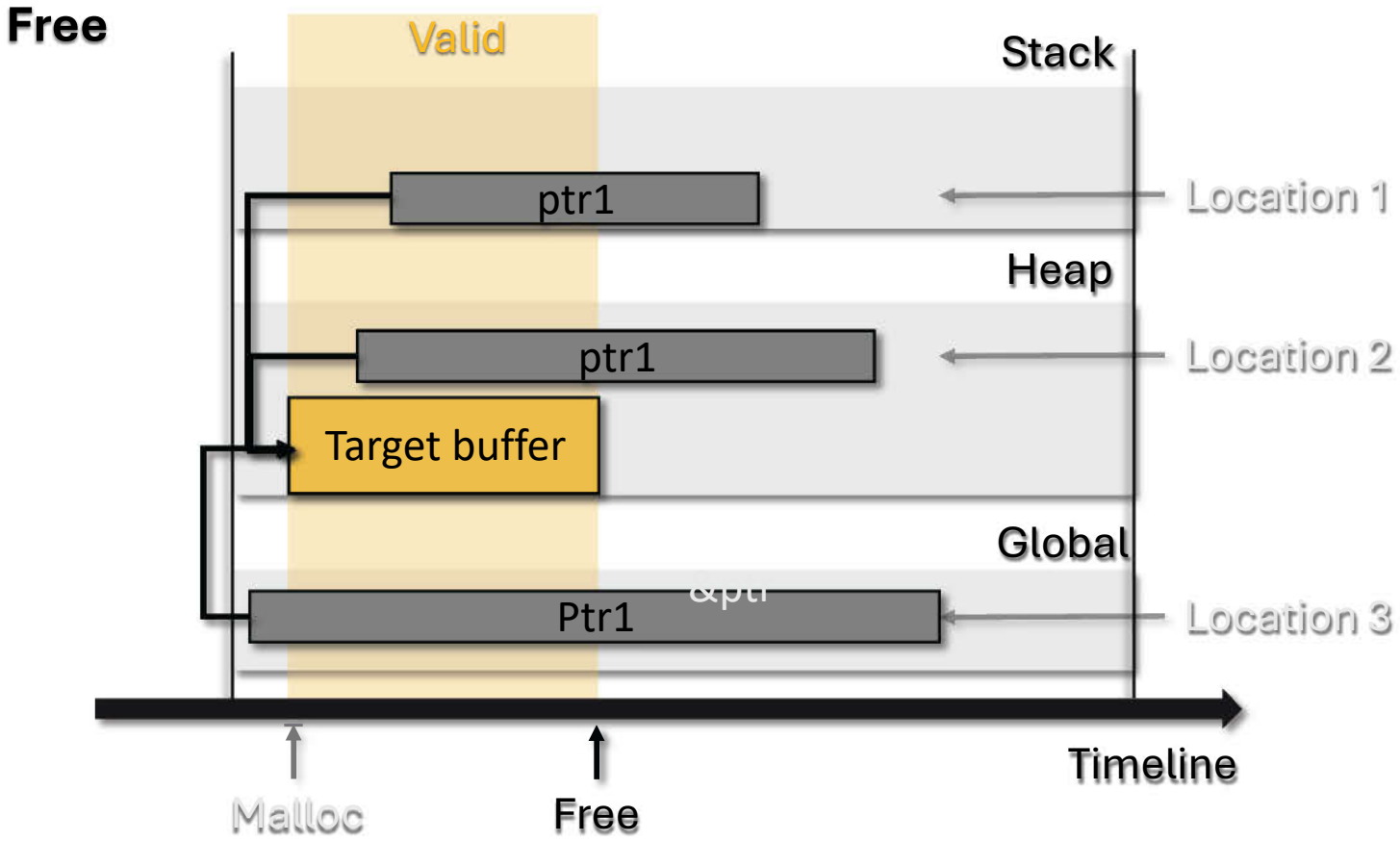


Metadata of PNs

Target buffer
Location 1
Location 2
Location 3

How PNs Work

Set up metadata → Register pointer storage locations → **Nullify dangling pointers**

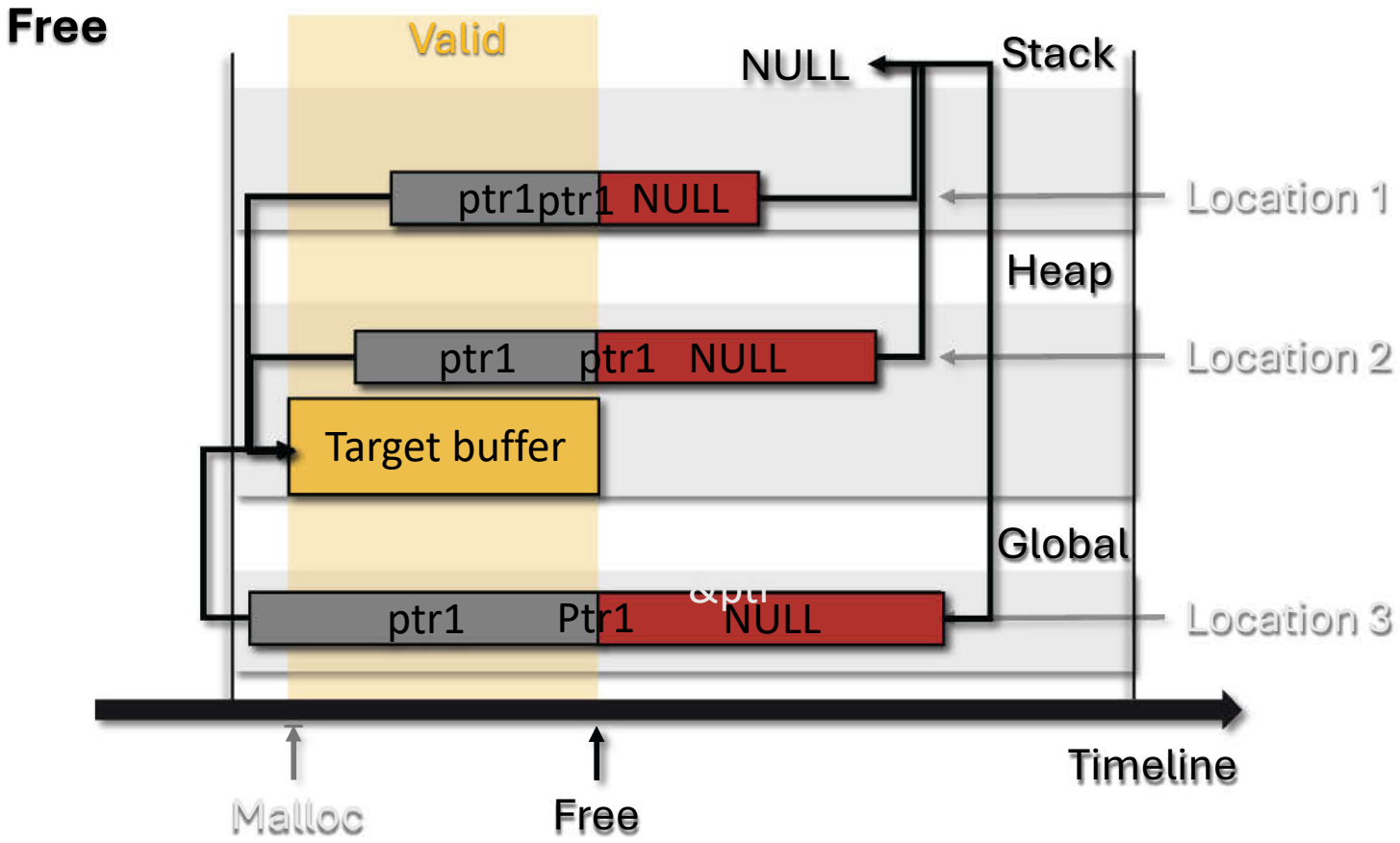


Metadata of PNs

Target buffer
Location 1
Location 2
Location 3

How PNs Work

Set up metadata → Register pointer storage locations → **Nullify dangling pointers**

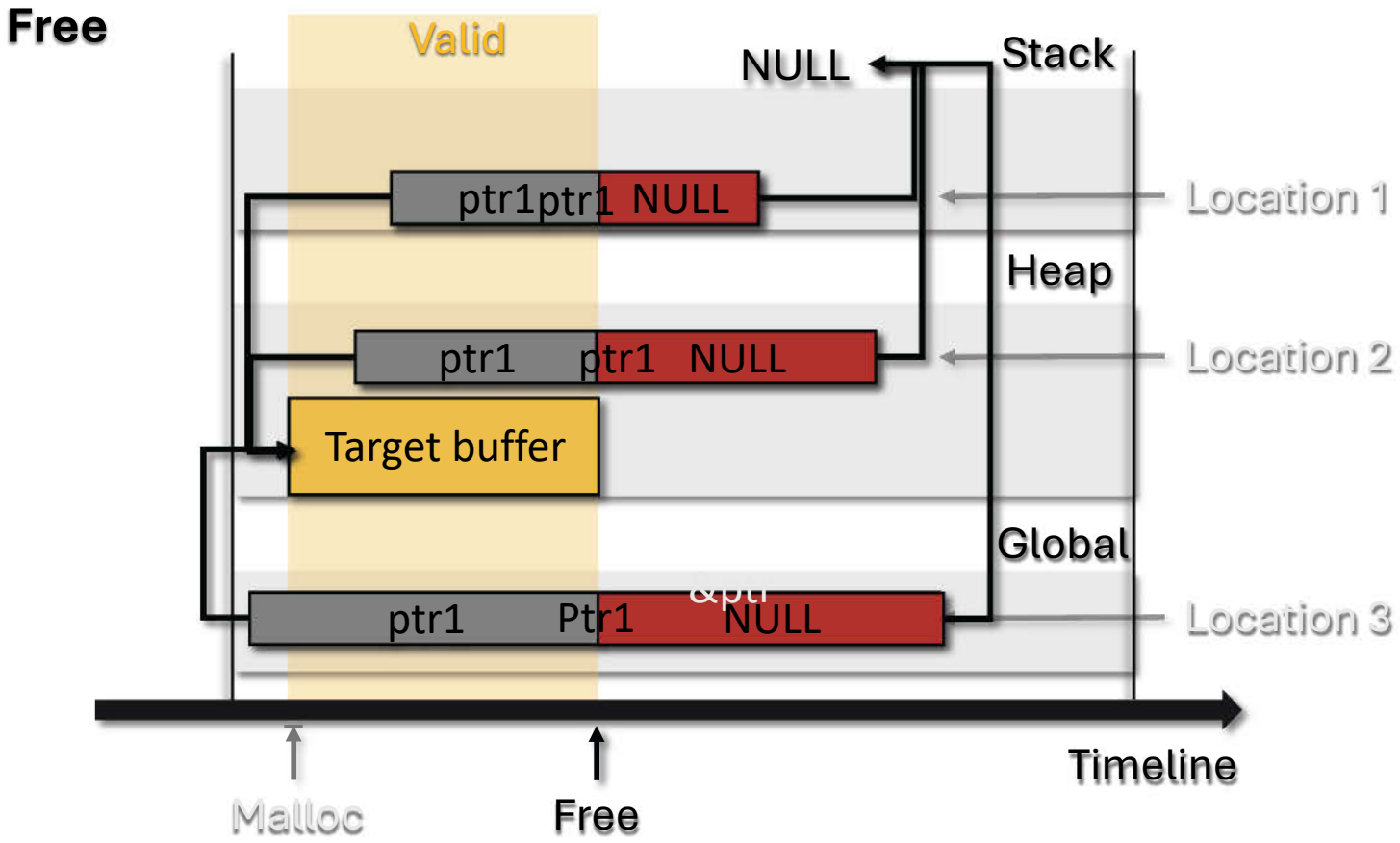


Metadata of PNs

Target buffer
Location 1
Location 2
Location 3

How PNs Work

Set up metadata → Register pointer storage locations → **Nullify dangling pointers**



Metadata of PNs

PNs are Effective - But Expensive

	SPEC 2017		SPEC 2006	
	Performance	**Memory**	**Performance**	**Memory**
FreeSentry [1]	1.32x	**2.60x**	1.24x	**2.37x**
DangSan [2]	1.36x*	1.29x*	1.23x	1.73x
CAMP [3]	**1.56x**	**2.73x**	1.40x	**3.10x**

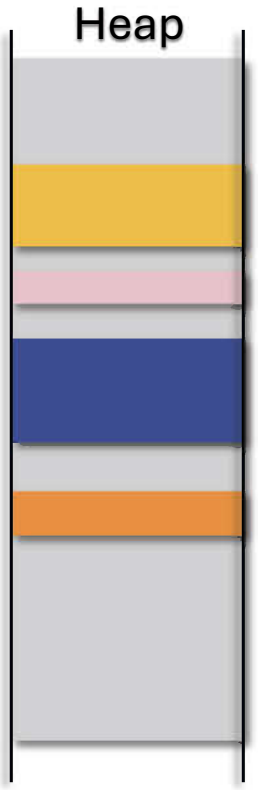
Despite strong UAF protection, prior PN systems incur **24–56%** runtime and **up to 200%** memory overhead

1. Metadata addressing is expensive.
2. Pointer registrations are excessive.

[1] Younan, Yves. "FreeSentry: protecting against use-after-free vulnerabilities due to dangling pointers." In *NDSS*. 2015.
[2] Van Der Kouwe, Erik, Vinod Nigade, and Cristiano Giuffrida. "Dangsan: Scalable use-after-free detection." In *Proceedings of the Twelfth European Conference on Computer Systems*, pp. 405-419. 2017.
[3] Lin, Zhenpeng, Zheng Yu, Ziyi Guo, Simone Campanoni, Peter Dinda, and Xinyu Xing. "{CAMP}: Compiler and Allocator-based Heap Memory Protection." In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 4015-4032. 2024.

Bottleneck I: Per-Store Metadata Addressing Is Costly

Expensive metadata lookup: Every pointer store must locate the metadata entry of its target buffer.



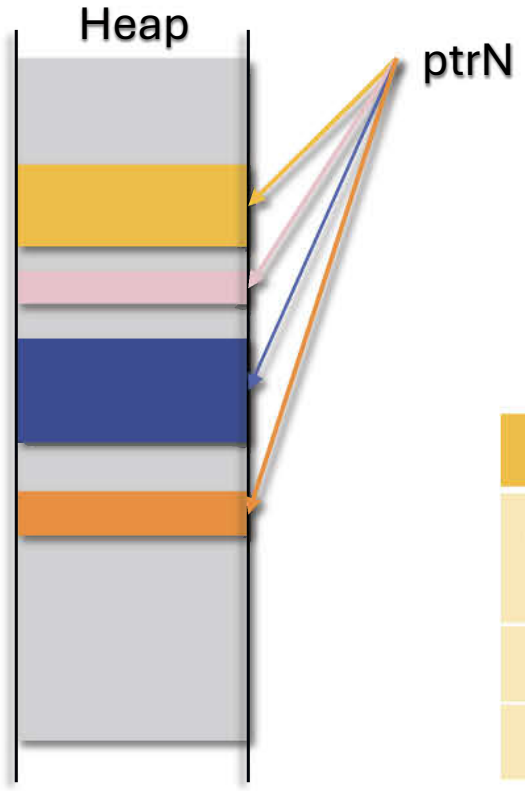
ptrN

Metadata of PNs

Target buffer1	Target buffer2	Target buffer3	Target buffer4
Start address 1 End address 1	Start address 2 End address 2	Start address 3 End address 3	Start address 4 End address 4

Bottleneck I: Per-Store Metadata Addressing Is Costly

Expensive metadata lookup: Every pointer store must locate the metadata entry of its target buffer.

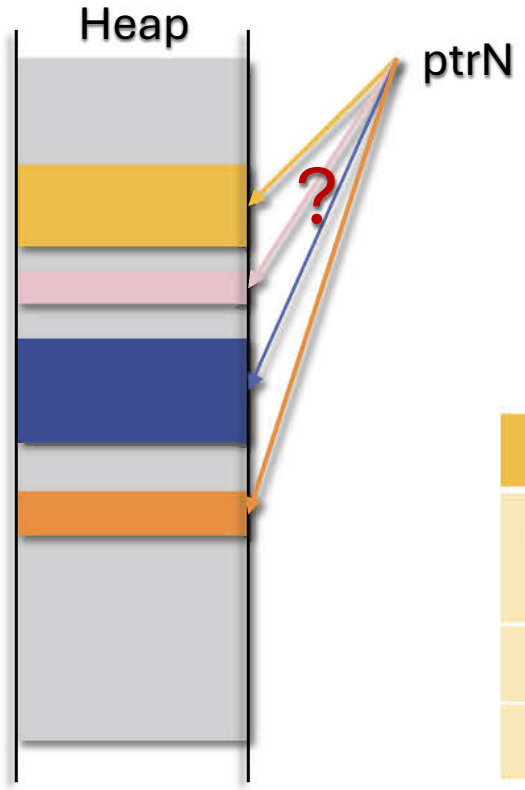


Metadata of PNs

Target buffer1	Target buffer2	Target buffer3	Target buffer4
Start address 1 End address 1	Start address 2 End address 2	Start address 3 End address 3	Start address 4 End address 4

Bottleneck I: Per-Store Metadata Addressing Is Costly

Expensive metadata lookup: Every pointer store must locate the metadata entry of its target buffer.

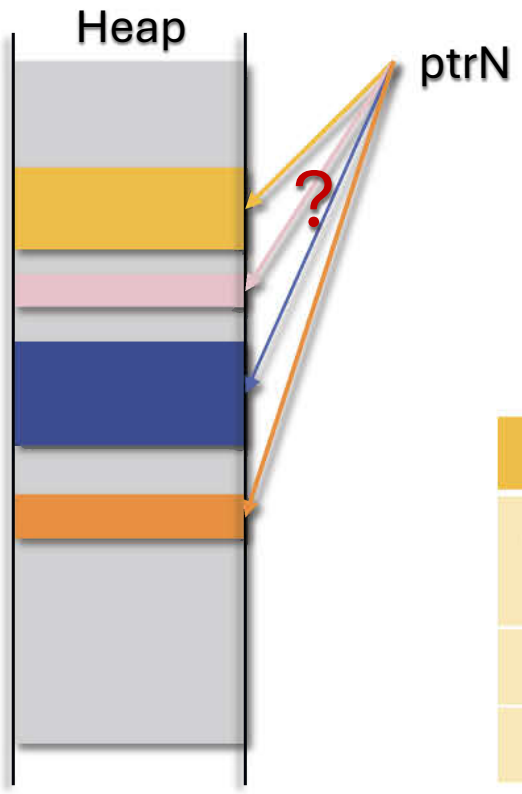


Metadata of PNs

Target buffer1	Target buffer2	Target buffer3	Target buffer4
Start address 1 End address 1	Start address 2 End address 2	Start address 3 End address 3	Start address 4 End address 4

Bottleneck I: Per-Store Metadata Addressing Is Costly

Expensive metadata lookup: Every pointer store must locate the metadata entry of its target buffer.




```
for( i = 1; i <= 4: i++) {
    if (ptrN > Start address i && ptrN < End address i)
        return i;
}
```

Metadata of PNs

Target buffer1	Target buffer2	Target buffer3	Target buffer4
Start address 1 End address 1	Start address 2 End address 2	Start address 3 End address 3	Start address 4 End address 4

Bottleneck I: Per-Store Metadata Addressing Is Costly

Expensive metadata lookup: Every pointer store must locate the metadata entry of its target buffer.

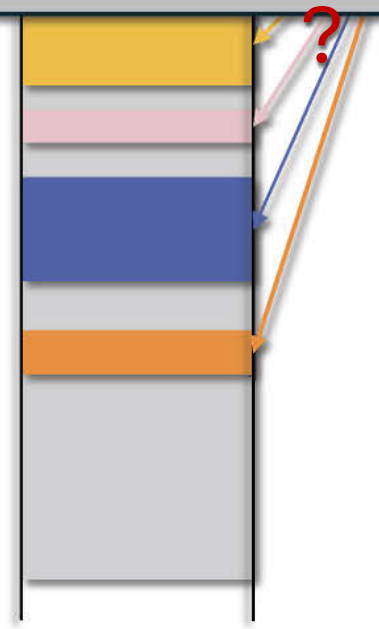
CAMP: 61% performance overhead comes from metadata addressing.

Heap

ptrN

Every pointer store must locate the metadata entry of its target buffer

```
for( i = 1; i <= 4; i++) {
    if (ptrN > Start address i && ptrN < End address i)
        return i;
}
```

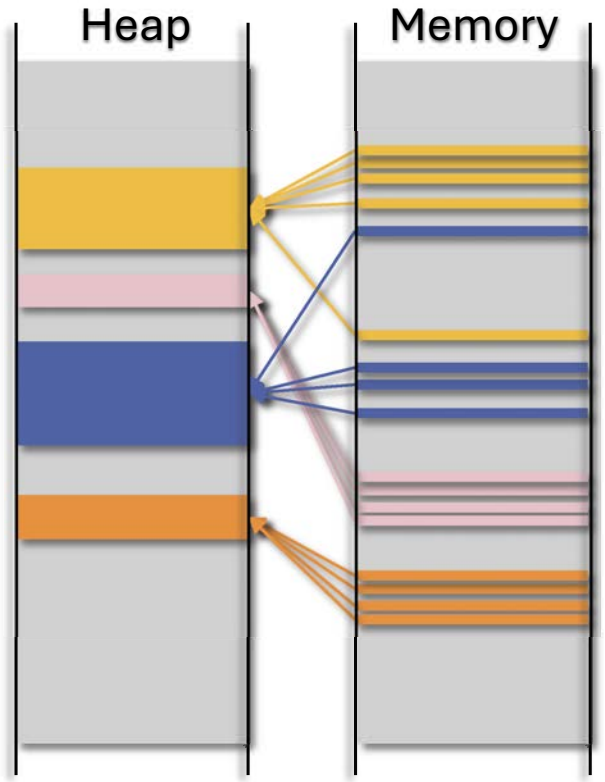


Metadata of PNs

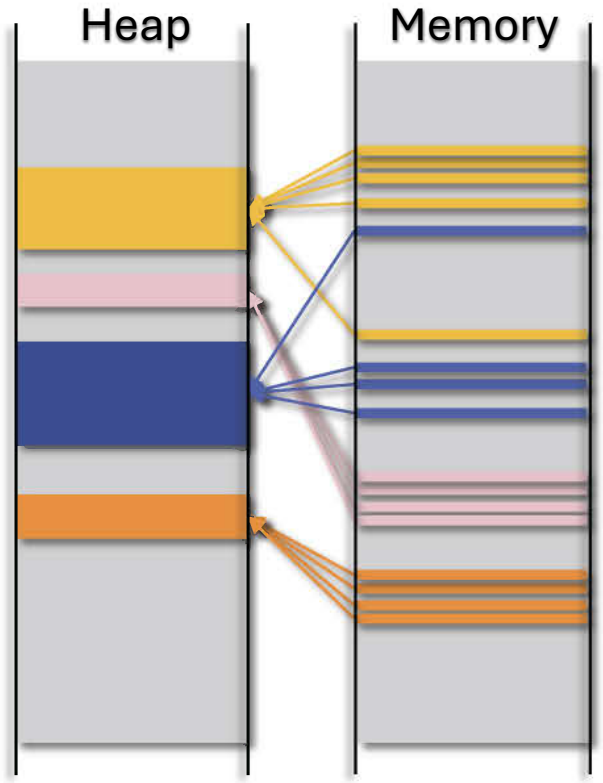
Target buffer1	Target buffer2	Target buffer3	Target buffer4
Start address 1 End address 1	Start address 2 End address 2	Start address 3 End address 3	Start address 4 End address 4

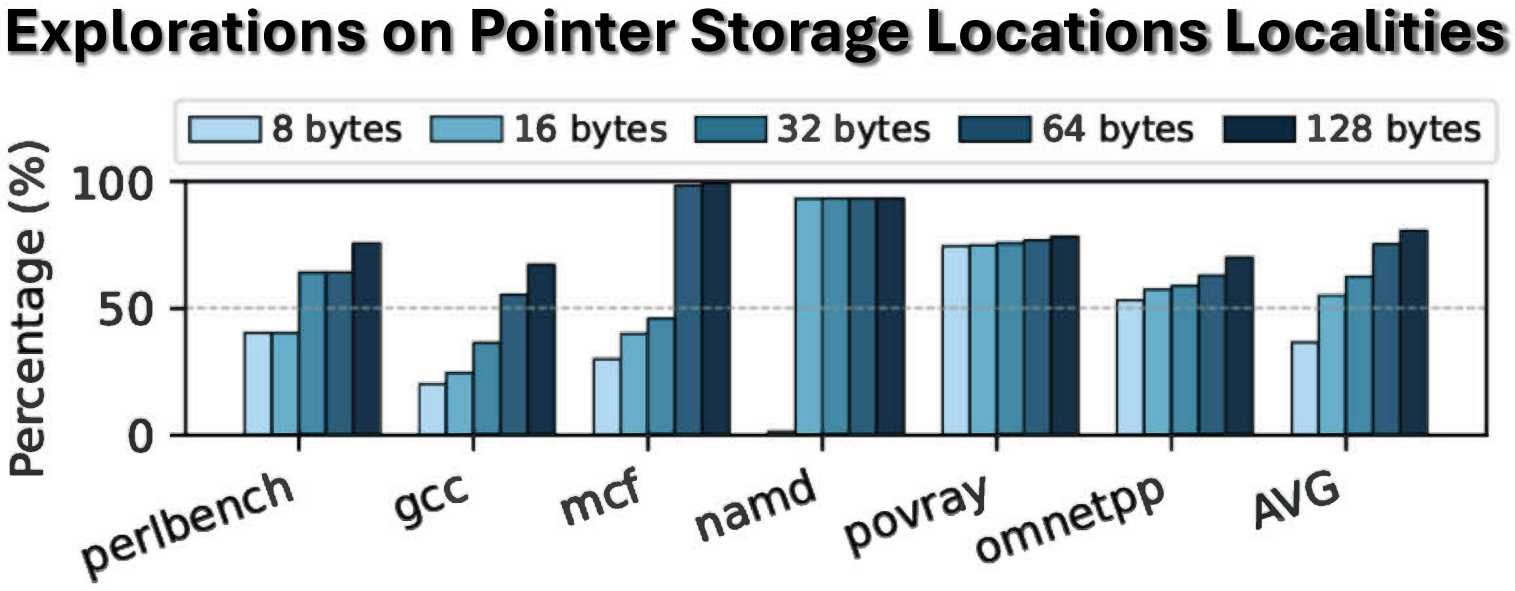
Bottleneck II: Pointer Locations Grow Unbounded

Excessive pointer registrations: registering each location independently, even when locations are spatially close.

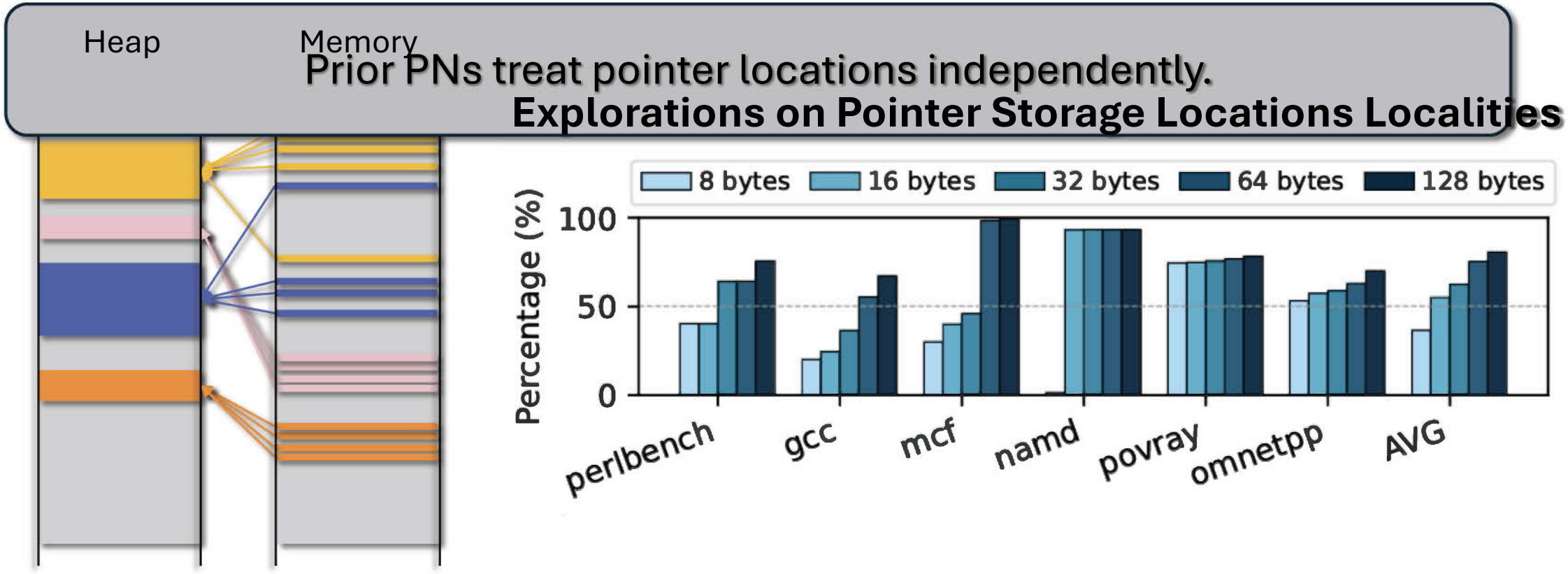


Key Observation: Strong Spatial Locality in Pointer Storage Locations





Key Observation: Strong Spatial Locality in Pointer Storage Locations

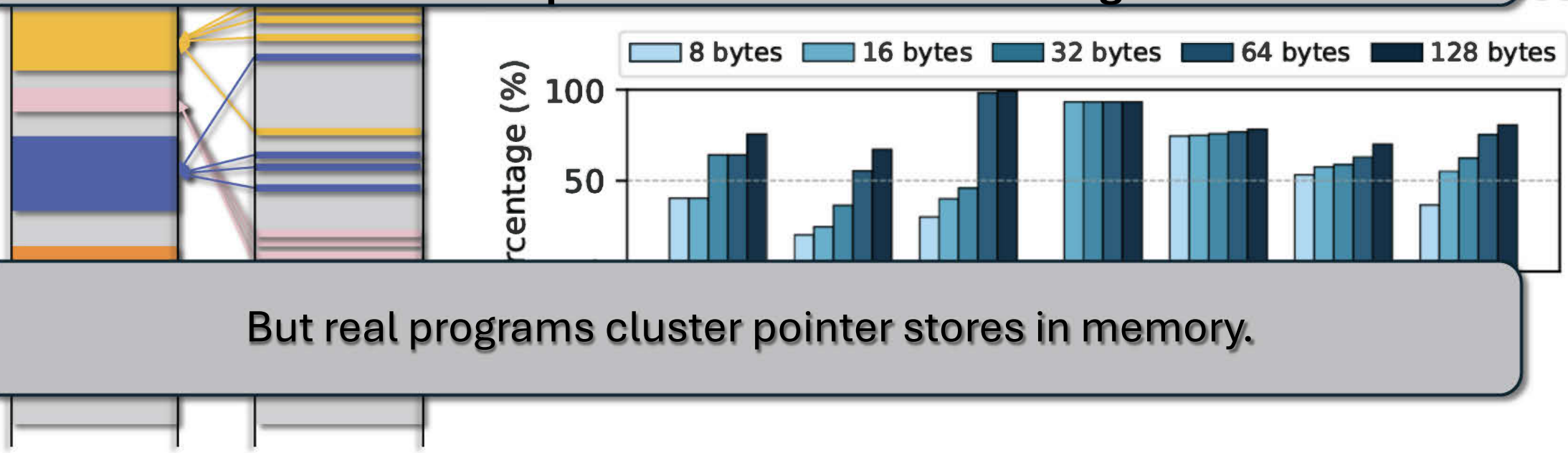


Key Observation: Strong Spatial Locality in Pointer Storage Locations

Heap Memory

Prior PNs treat pointer locations independently.

Explorations on Pointer Storage Locations Localities



But real programs cluster pointer stores in memory.

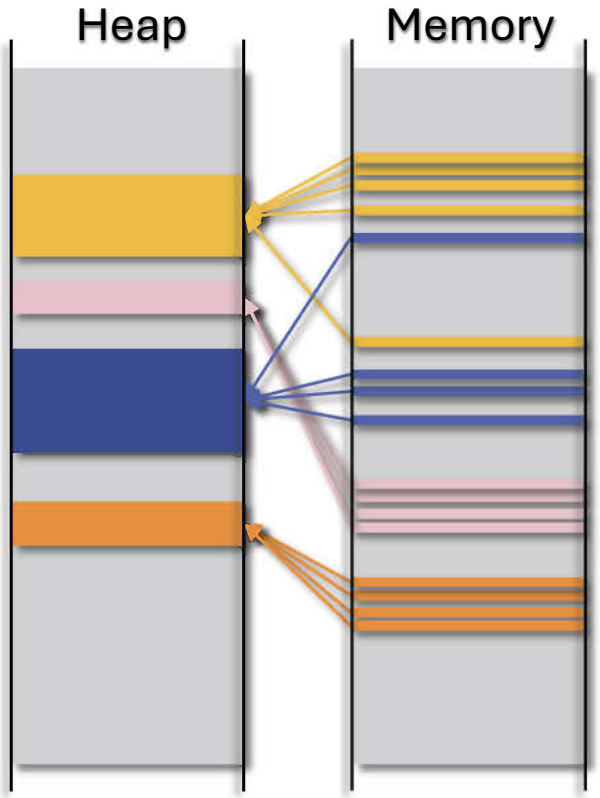
Motivation

Faster metadata addressing.

Fewer registrations by **exploiting locality.**

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

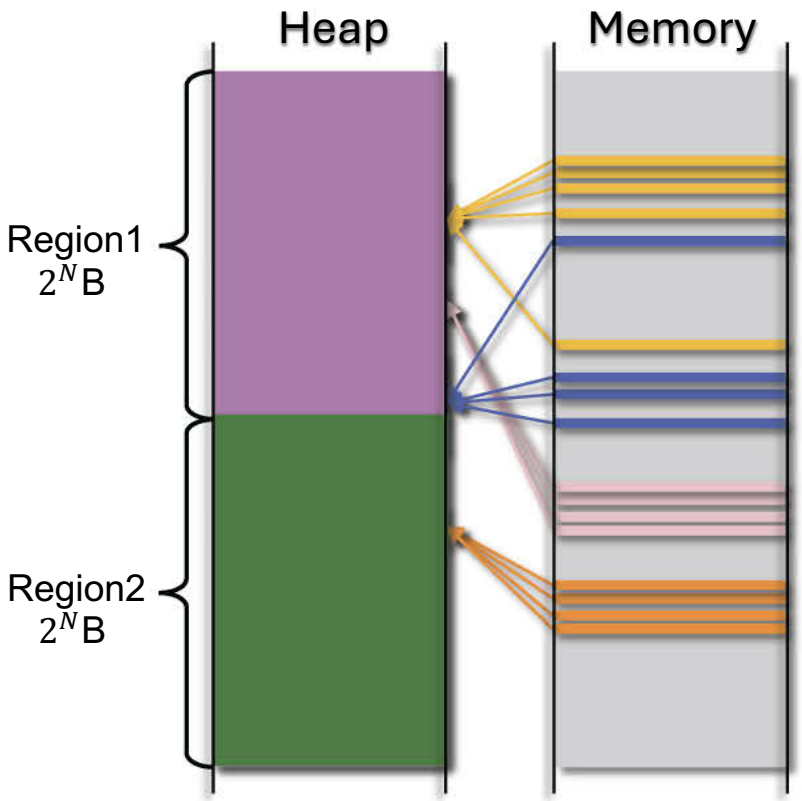
one bit shift + one table lookup

Metadata of FPN

Target buffer1	Target buffer2	Target buffer3	Target buffer4
Start address 1 End address 1	Start address 2 End address 2	Start address 3 End address 3	Start address 4 End address 4

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

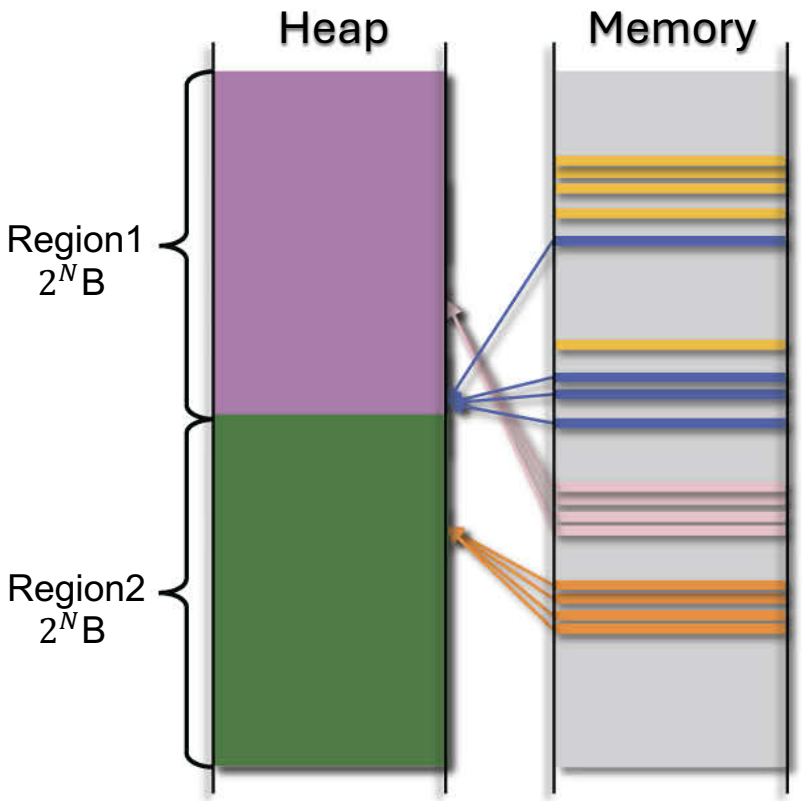
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

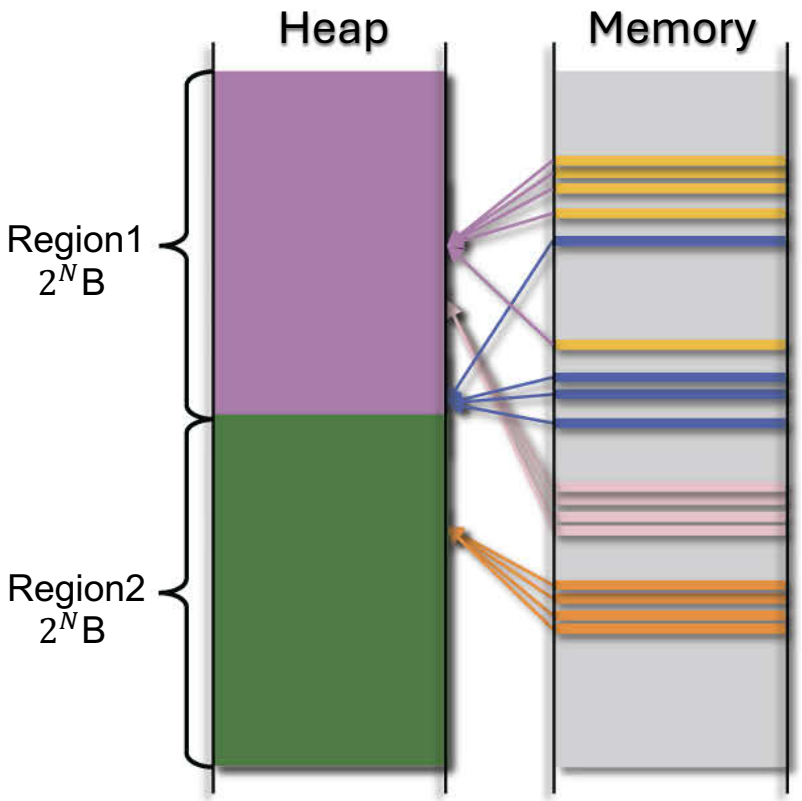
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

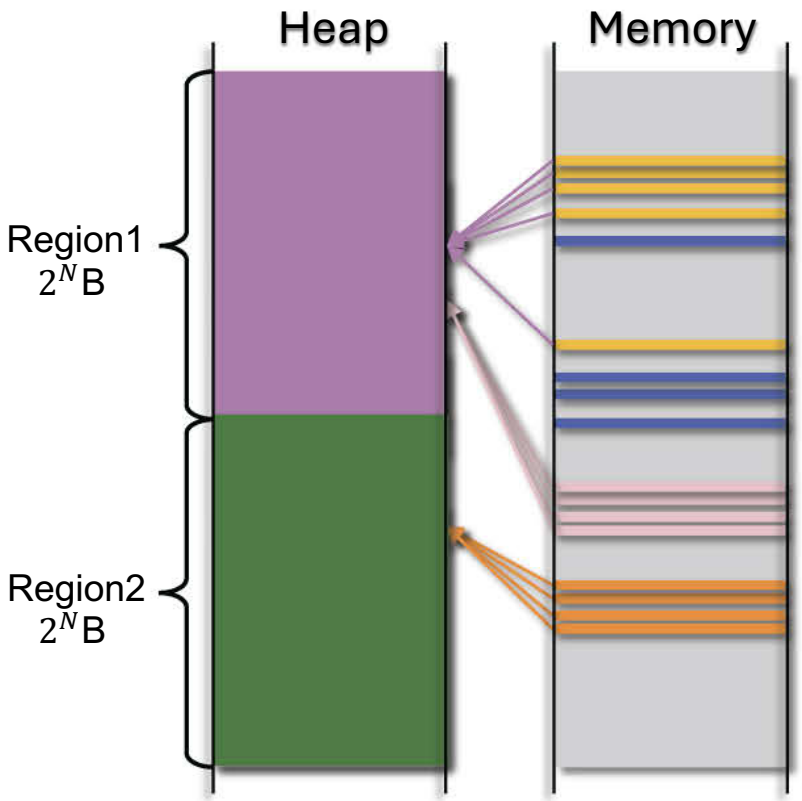
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

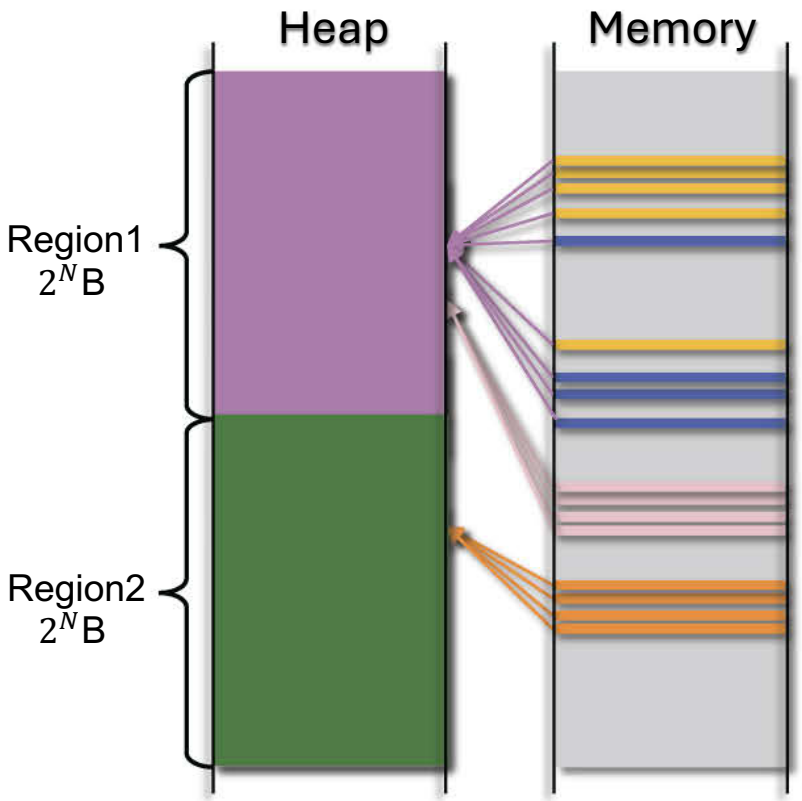
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

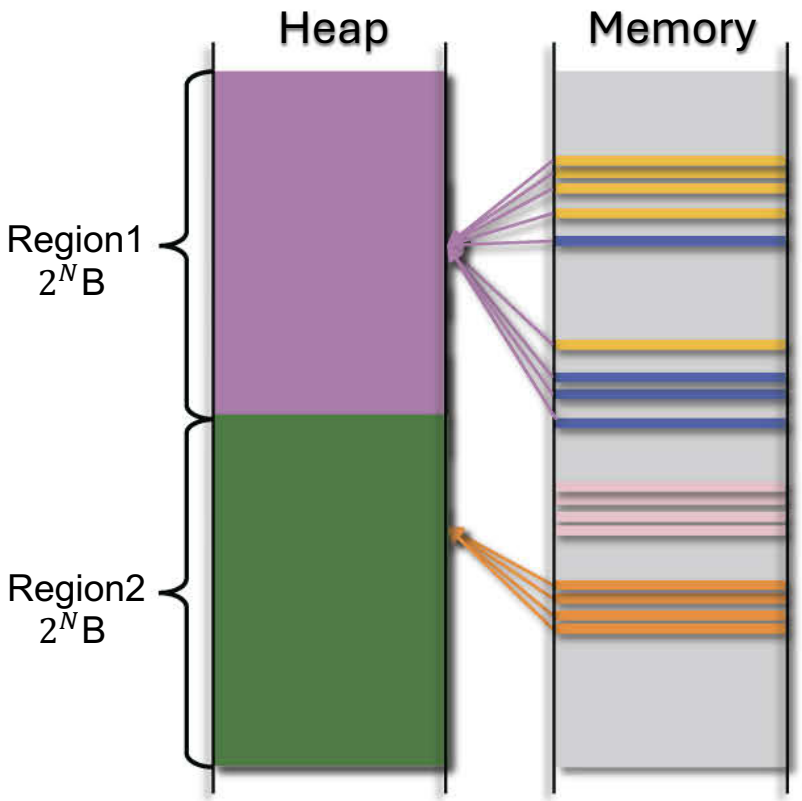
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

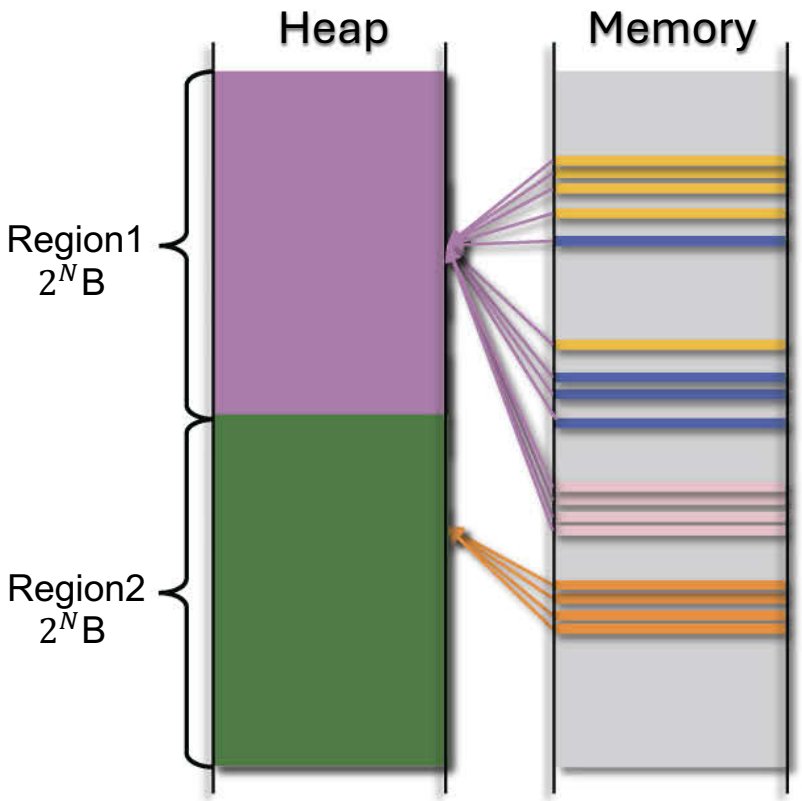
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

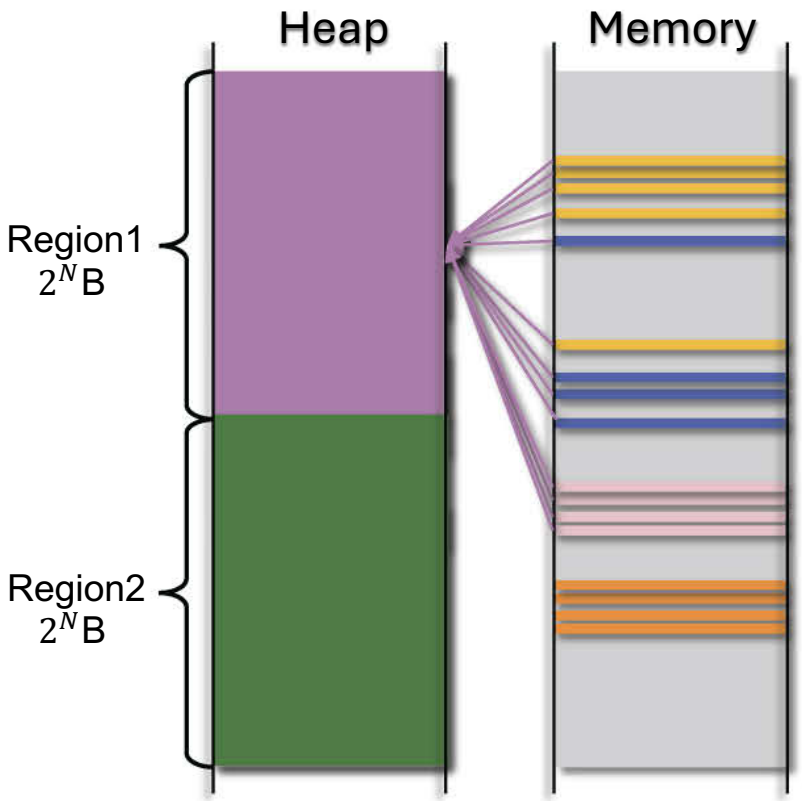
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

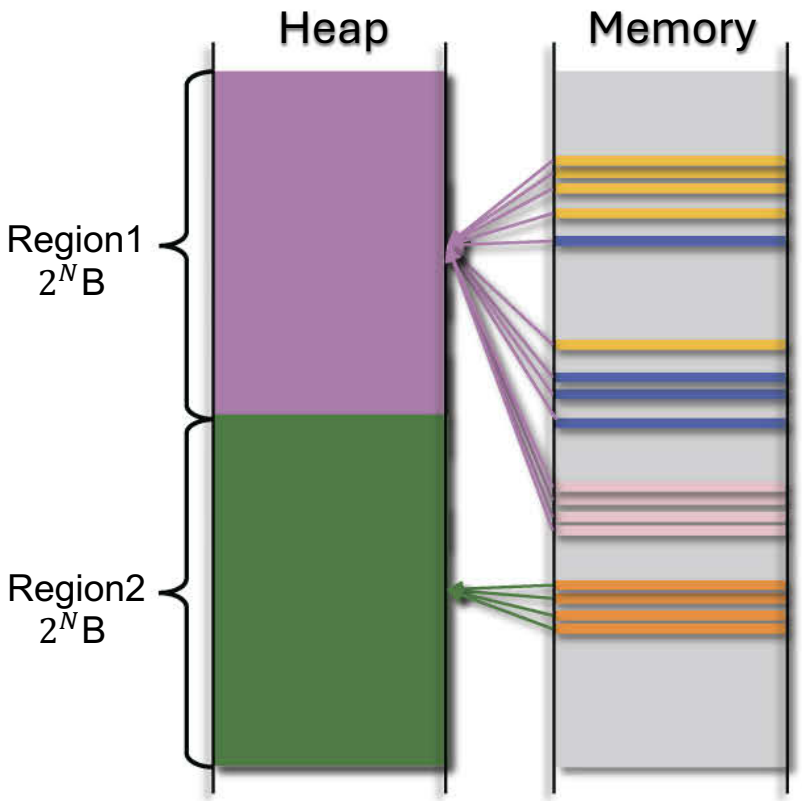
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

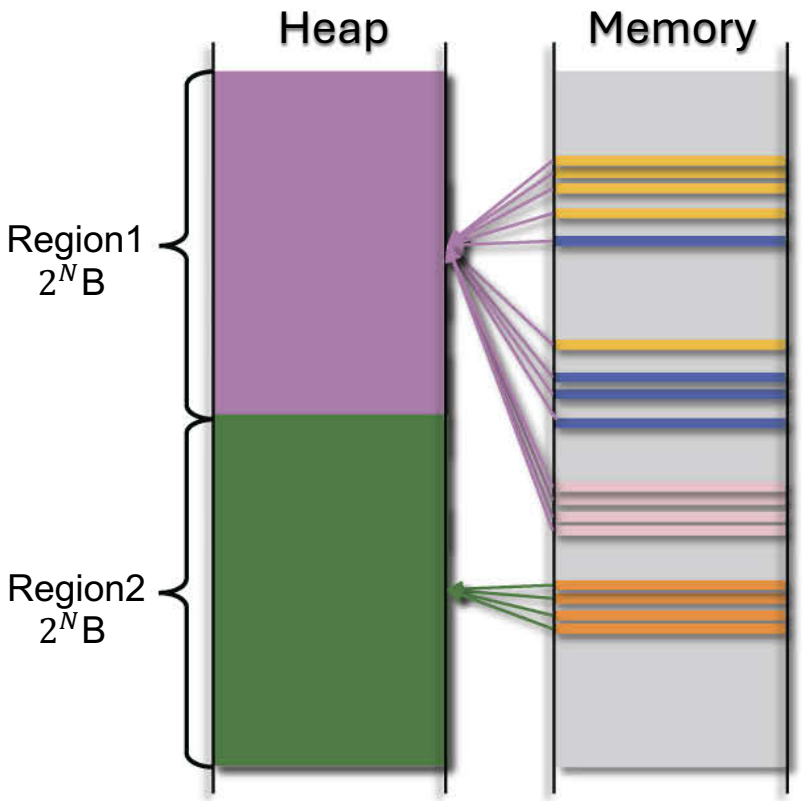
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

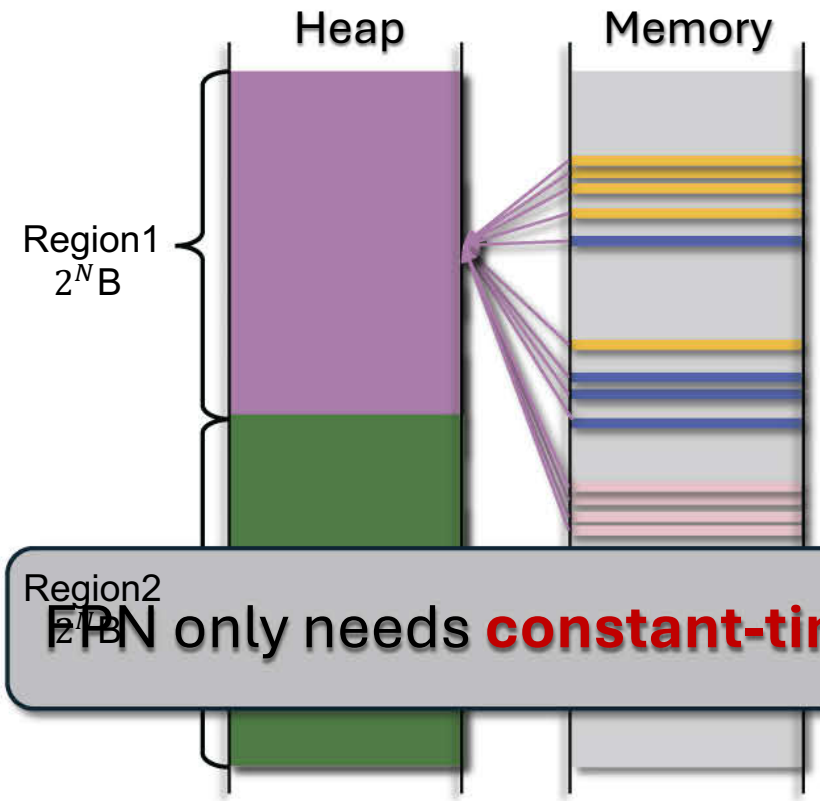
one bit shift + one table lookup

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Faster Metadata Addressing

1. **Aligned** region-based (**2^N Bytes**) metadata management.



ptrN points to ? Region

Region ID = ptrN >> N

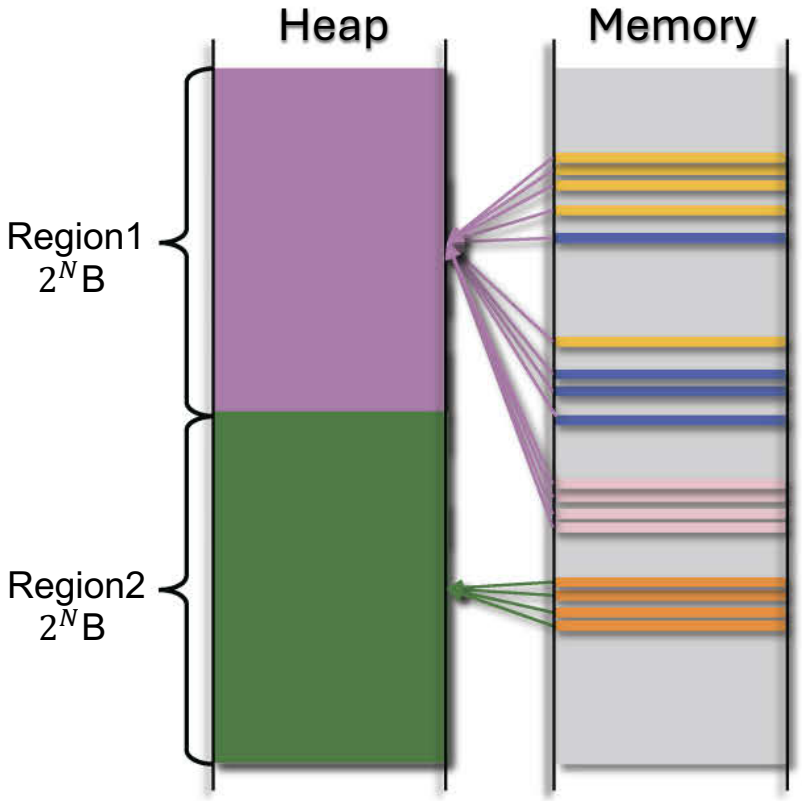
one bit shift + one table lookup

Metadata of FPN

FPN only needs **constant-time** metadata addressing (no tree traversal, no division)

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. **Aligned** block-based (**2^M Bytes**) storage location management.



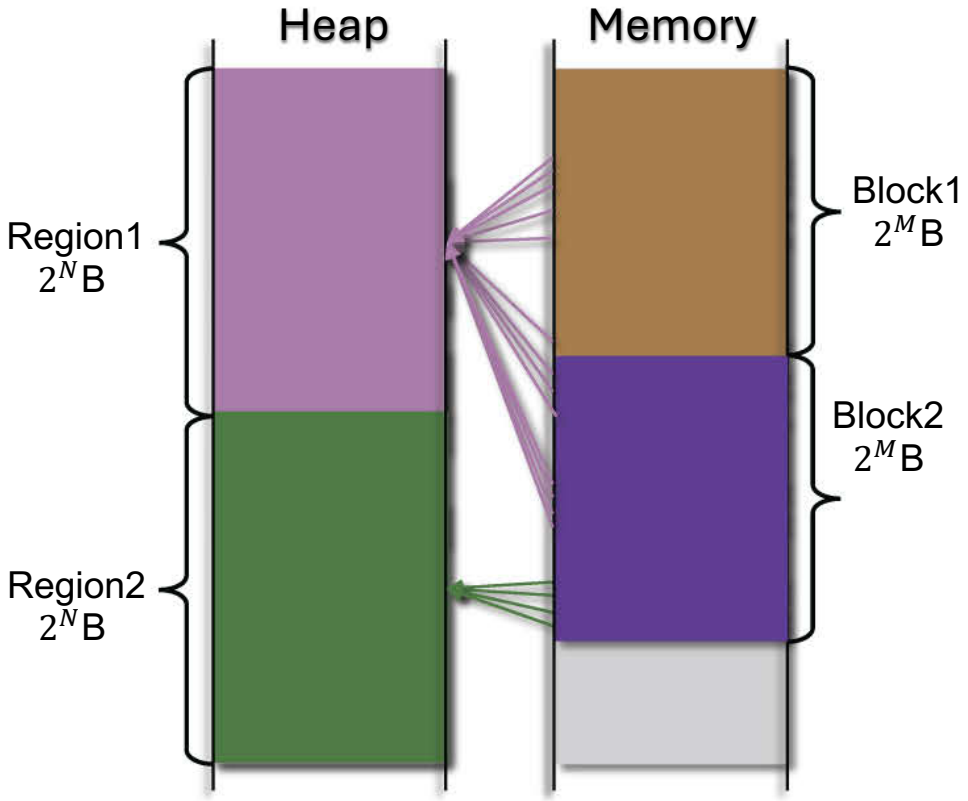
Instead of registering every pointer location, FPN registers a block instead.

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. **Aligned** block-based (**2^M Bytes**) storage location management.



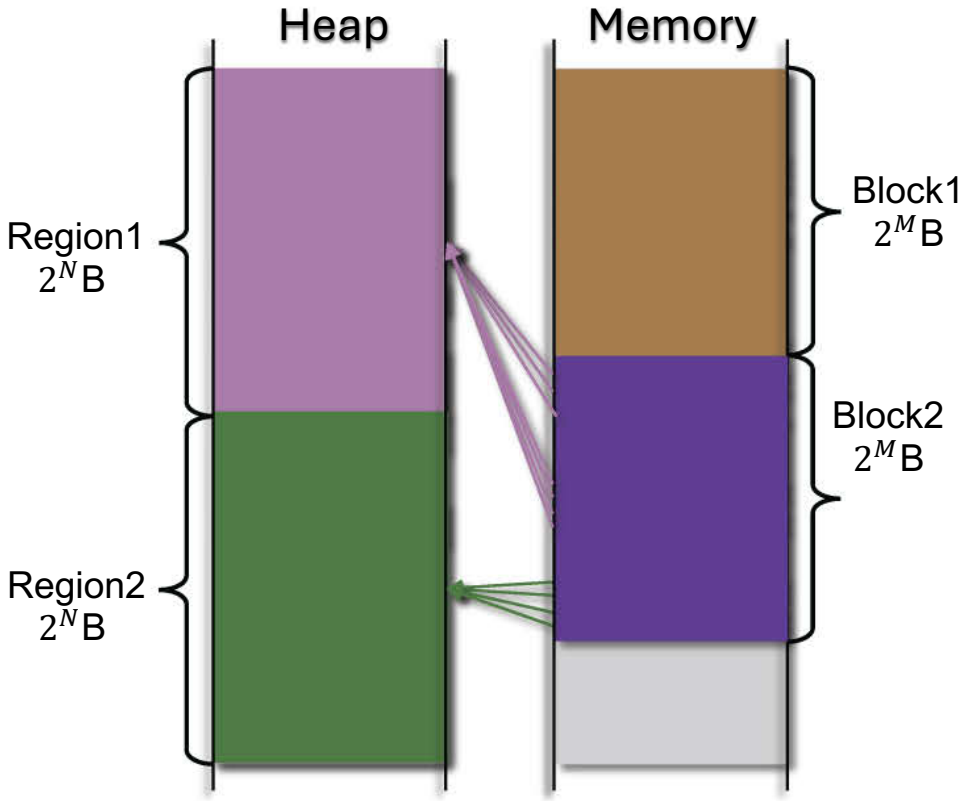
Instead of registering every pointer location, FPN registers a block instead.

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. Aligned block-based (2^M Bytes) storage location management.



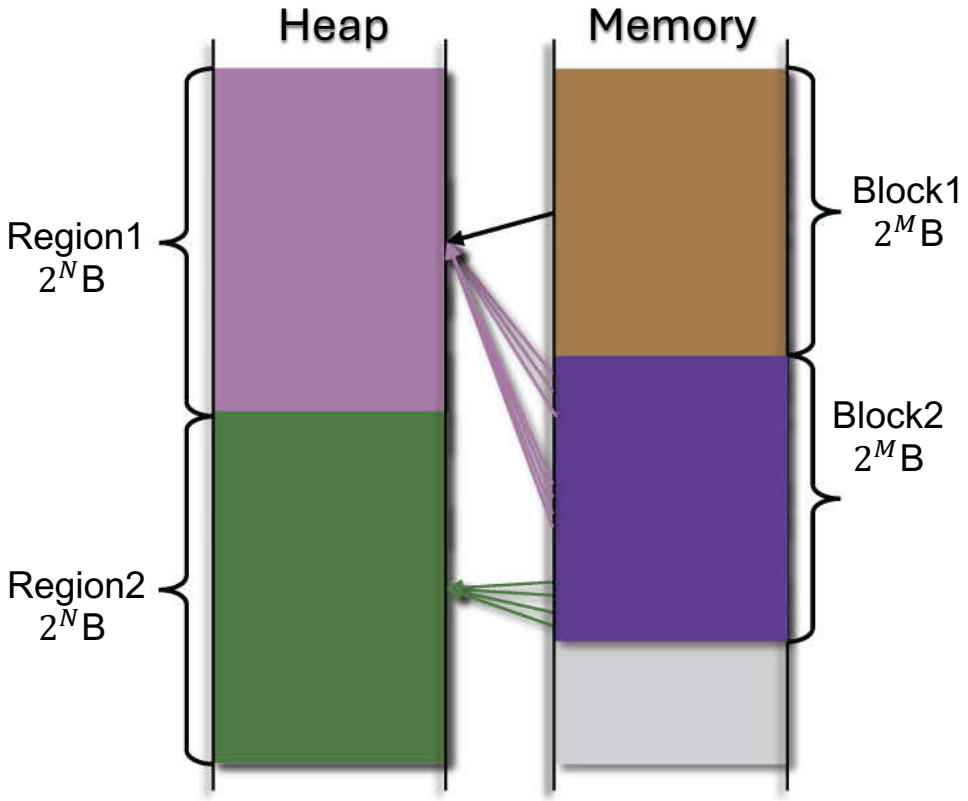
Instead of registering every pointer location, FPN registers a block instead.

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. Aligned block-based (2^M Bytes) storage location management.



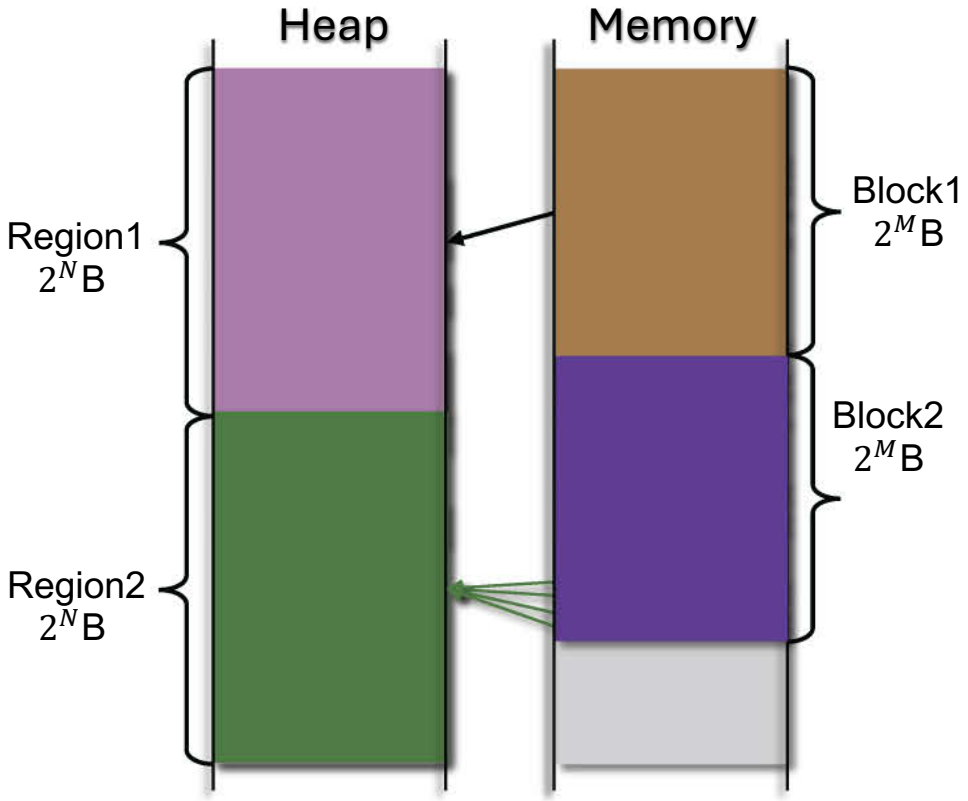
Instead of registering every pointer location, FPN registers a block instead.

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. Aligned block-based (2^M Bytes) storage location management.



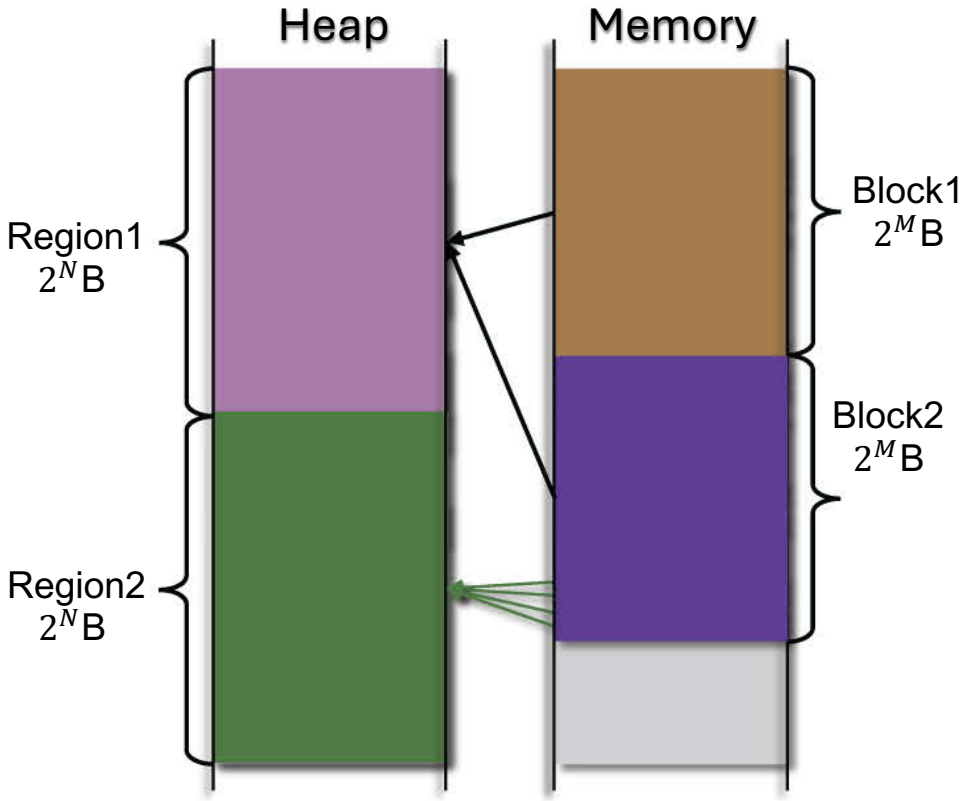
Instead of registering every pointer location, FPN registers a block instead.

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. Aligned block-based (2^M Bytes) storage location management.



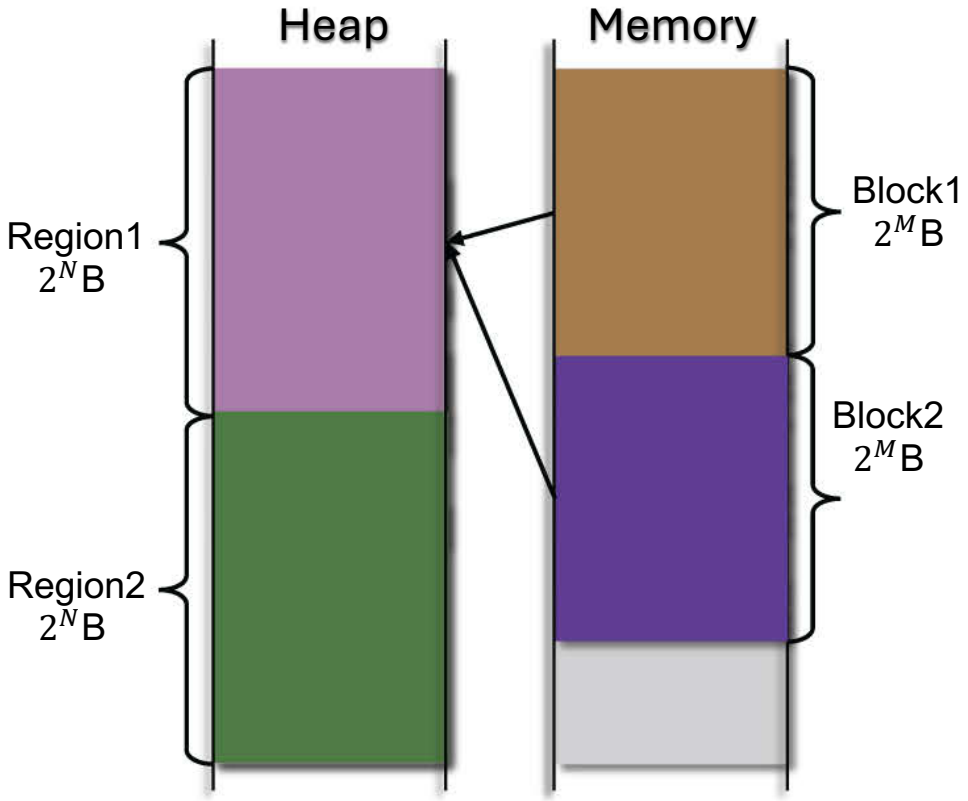
Instead of registering every pointer location, FPN registers a block instead.

Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. **Aligned** block-based (**2^M Bytes**) storage location management.



Instead of registering every pointer location, FPN registers a block instead.

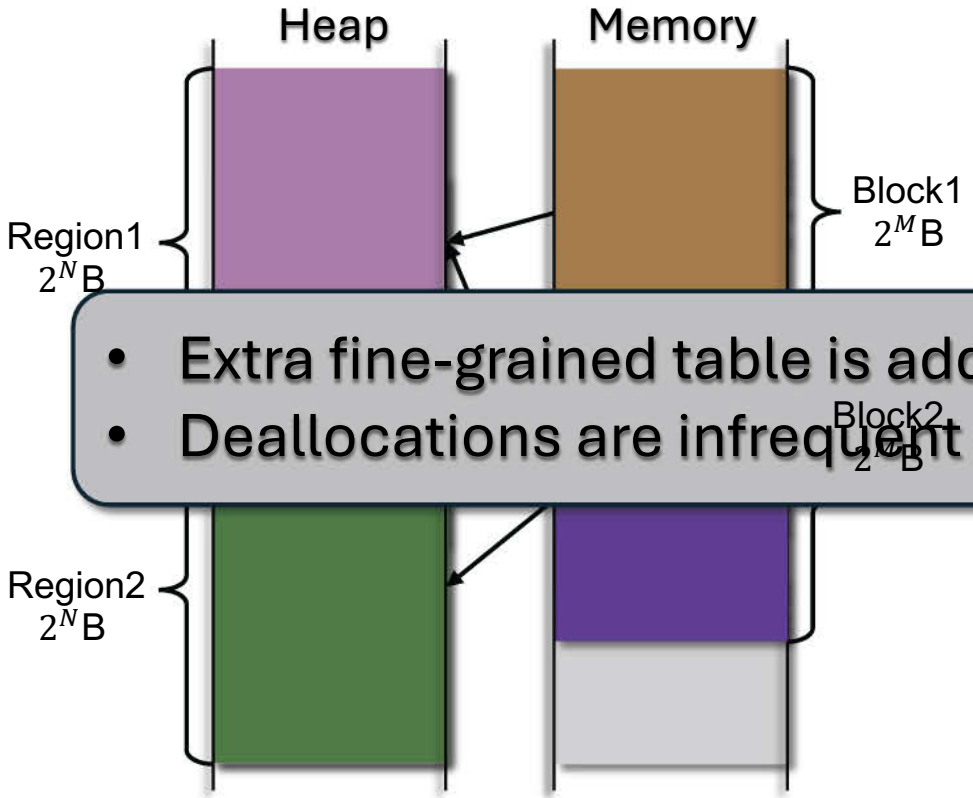
Metadata of FPN

Region1	Region2

Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. **Aligned** block-based (**2^M Bytes**) storage location management.

Instead of registering every pointer location, FPN registers a block instead.



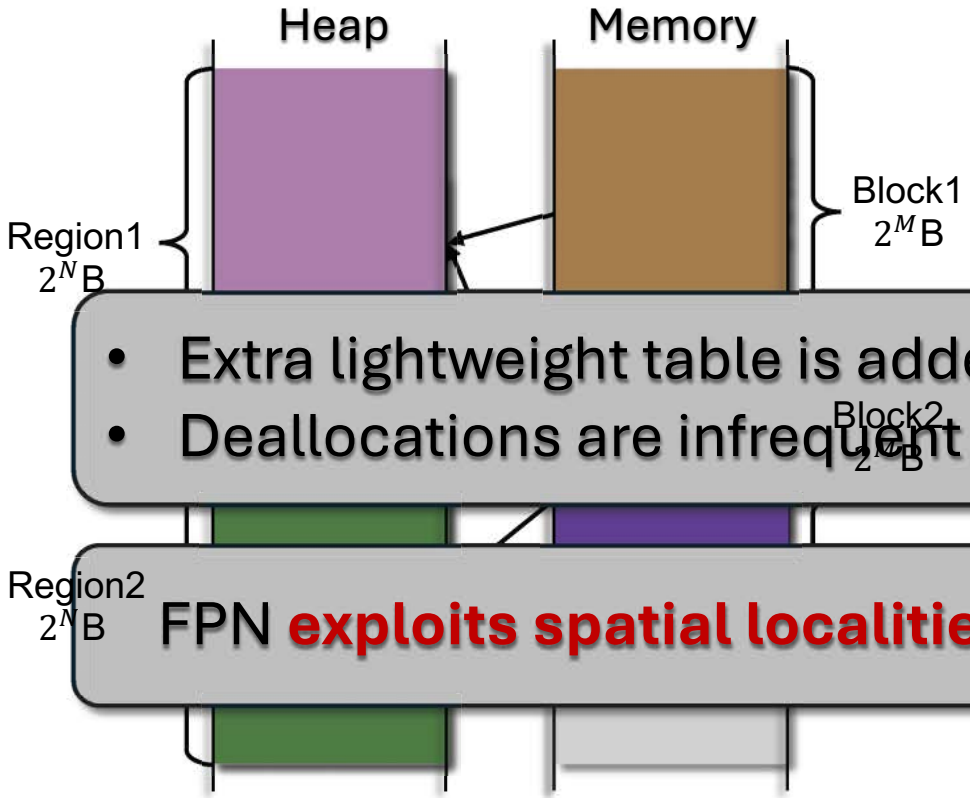
Metadata of FPN

- Extra fine-grained table is added to reduce false positives.
- Deallocations are infrequent -> scanning overhead is negligible.

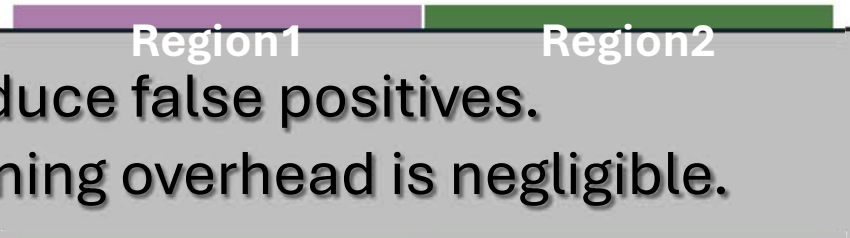
Fast Pointer Nullification (FPN): Coarse Grained Registrations

2. **Aligned** block-based (**2^M Bytes**) storage location management.

Instead of registering every pointer location, FPN registers a block instead.



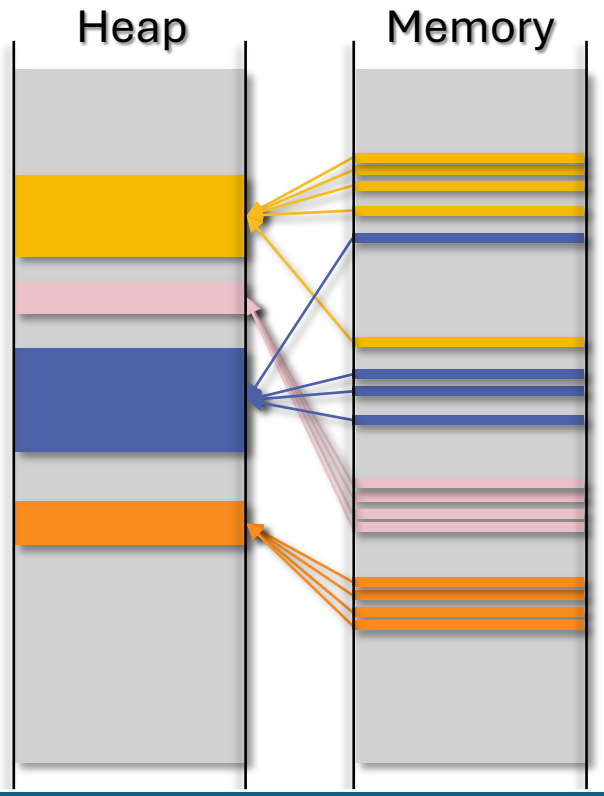
Metadata of FPN

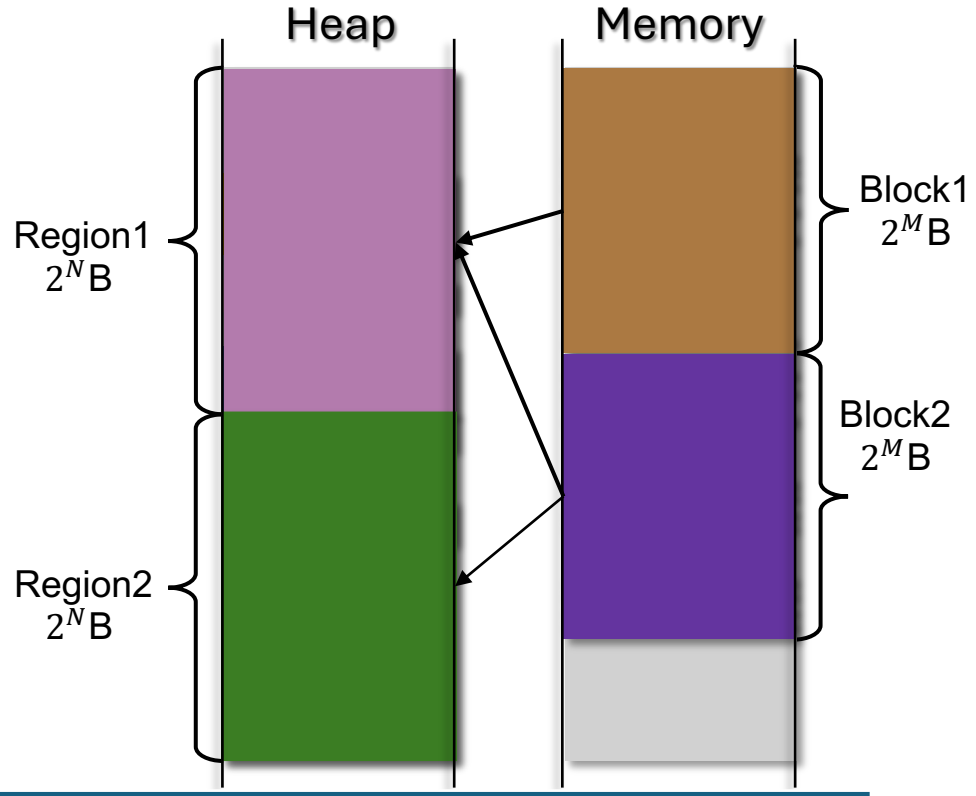


- Extra lightweight table is added to reduce false positives.
- Deallocations are infrequent -> scanning overhead is negligible.

FPN **exploits spatial localities** and reduces the number of registrations.

Comparison with Previous PNs



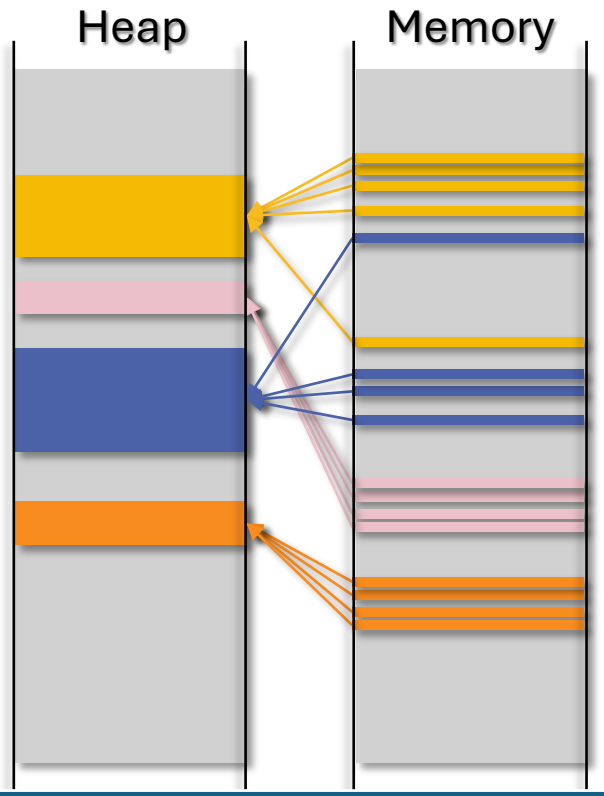


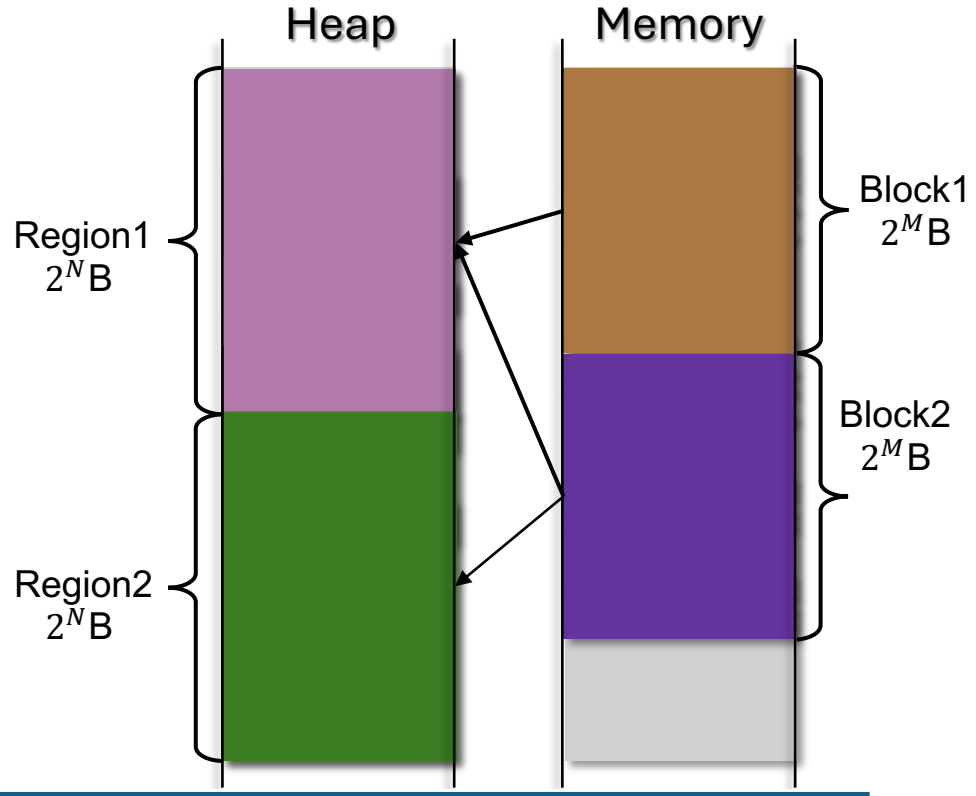
Previous	FPN
Addressing of metadata	
O(log(n)) [1] or complex arithmetic operations [2]	One bit shift operation + one table addressing
# of Registered Pointer Locations	
17	3

[1] Lee, Byoungyoung, Chengyu Song, Yeongjin Jang, Tielei Wang, Taesoo Kim, Long Lu, and Wenke Lee. "Preventing Use-after-free with Dangling Pointers Nullification." In *NDSS*. 2015.

[2] Lin, Zhenpeng, Zheng Yu, Ziyi Guo, Simone Campanoni, Peter Dinda, and Xinyu Xing. "{CAMP}: Compiler and Allocator-based Heap Memory Protection." In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 4015-4032. 2024.

Comparison with Previous PNs





Previous	FPN
# of Registered Pointer Locations	
O(nlog(n)) [1] or complex arithmetic operations [2]	Bit shift operation

FPN reduces both lookup cost and registration count.

[1] Lee, Byoungyoung, Chengyu Song, Yeongjin Jang, Tielei Wang, Taesoo Kim, Long Lu, and Wenke Lee. "Preventing Use-after-free with Dangling Pointers Nullification."

yi Guo, da, and and

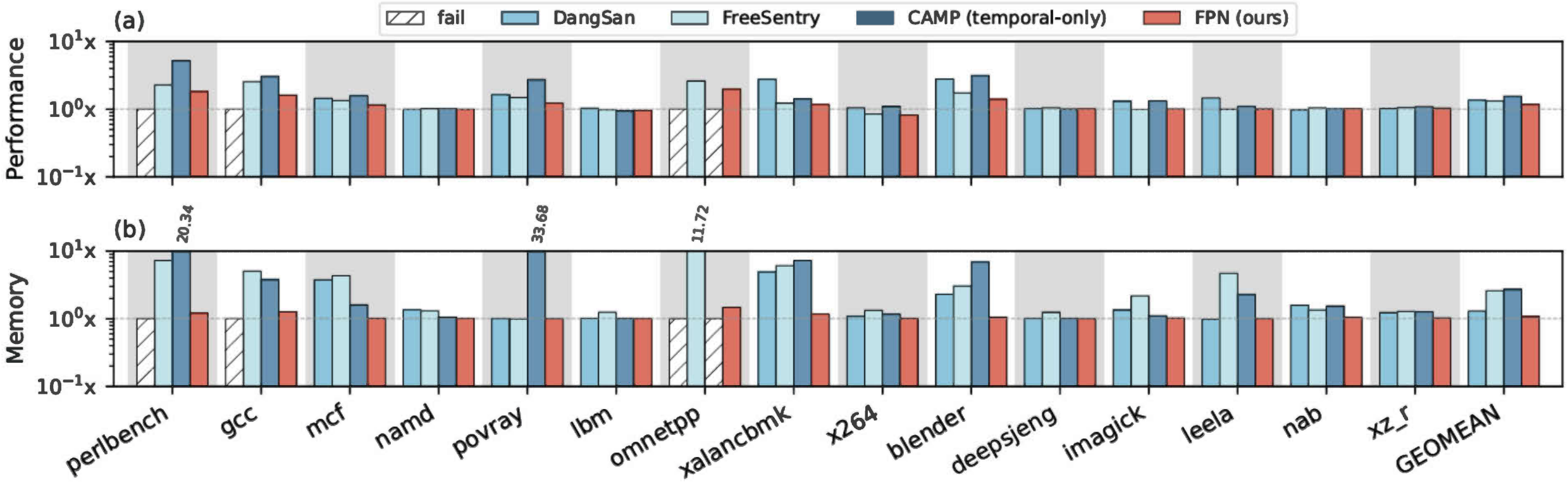
ecurity *Symposium (USENIX Security 24)*, pp. 4015-4032. 2024.

Security

CVE	Freesentry	Dangsan	CAMP	FPN (ours)
CVE-2015-3205	✖	✔	✔	✔
CVE-2015-2787	✔	✔	✔	✔
CVE-2015-6835	✖	✖	✔	✔
CVE-2016-5773	✔	✖	✔	✔
Issue-3515	✔	crash	✔	✔
CVE-2020-6838	✔	crash	✔	✔
CVE-2021-44964	✔	✔	✔	✔
CVE-2020-21688	✔	✔	✔	✔
CVE-2021-33468	✔	✔	✔	✔
CVE-2020-24978	✔	✔	✔	✔
Issue-1325664	✖	✖	✔	✔
CVE-2022-43286	✔	✔	✔	✔
CVE-2019-16165	✔	✔	✔	✔
CVE-2021-4187	✔	✔	✔	✔

FPN maintains the same security protection as other PNs

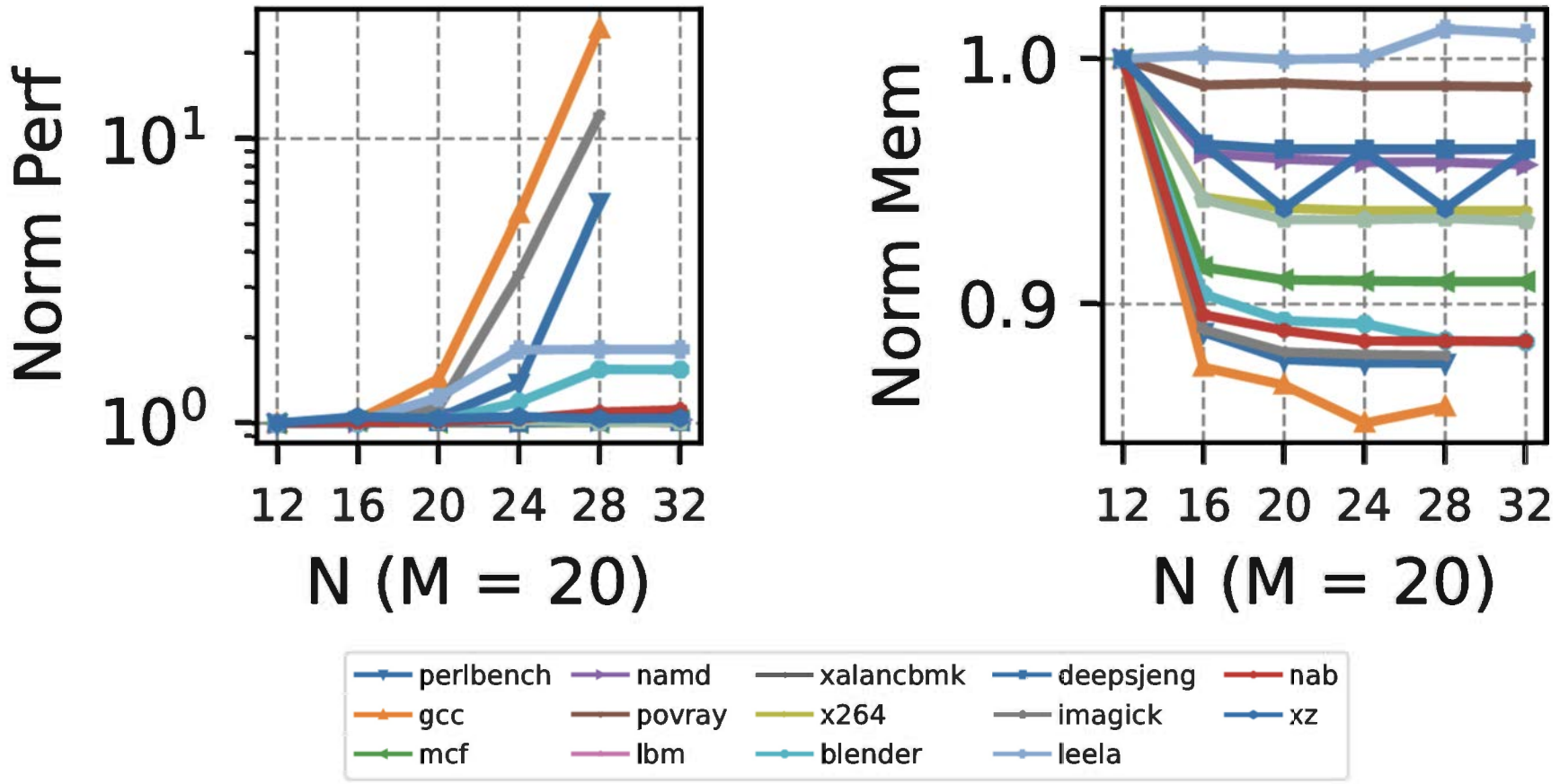
Performance & Memory



FPN has lower performance and memory overhead compared with other PNs

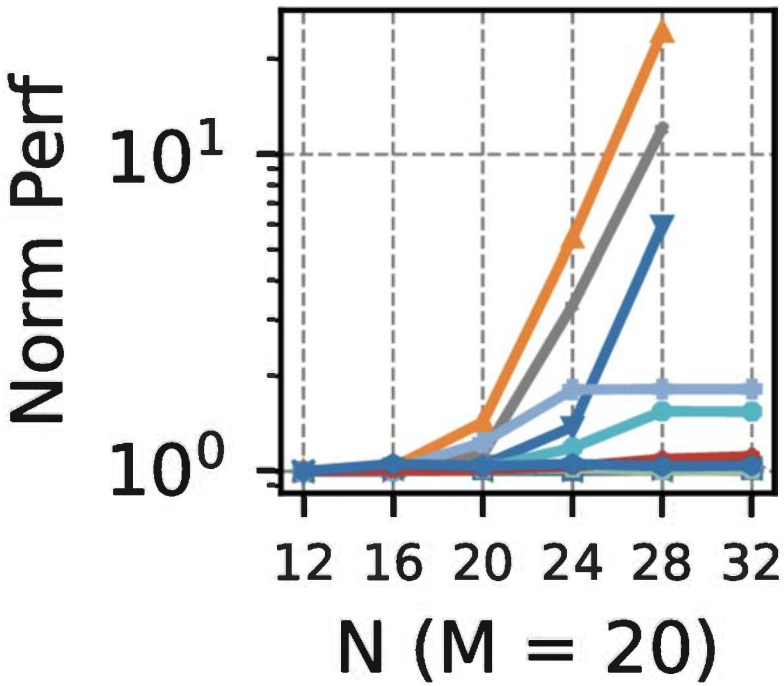
Explorations on Region Sizes and Block Sizes

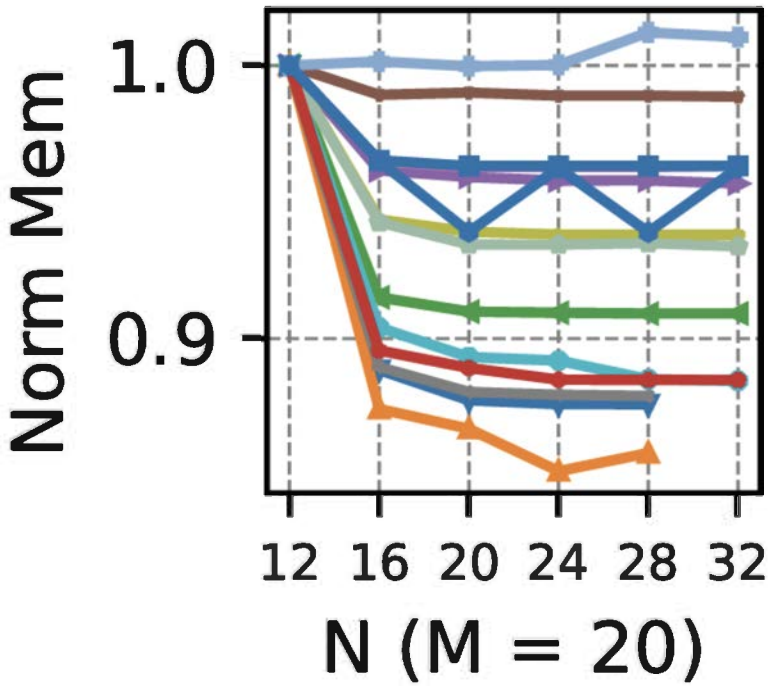
Region: aligned 2^N Bytes



Explorations on Region Sizes and Block Sizes

Region: aligned 2^N Bytes

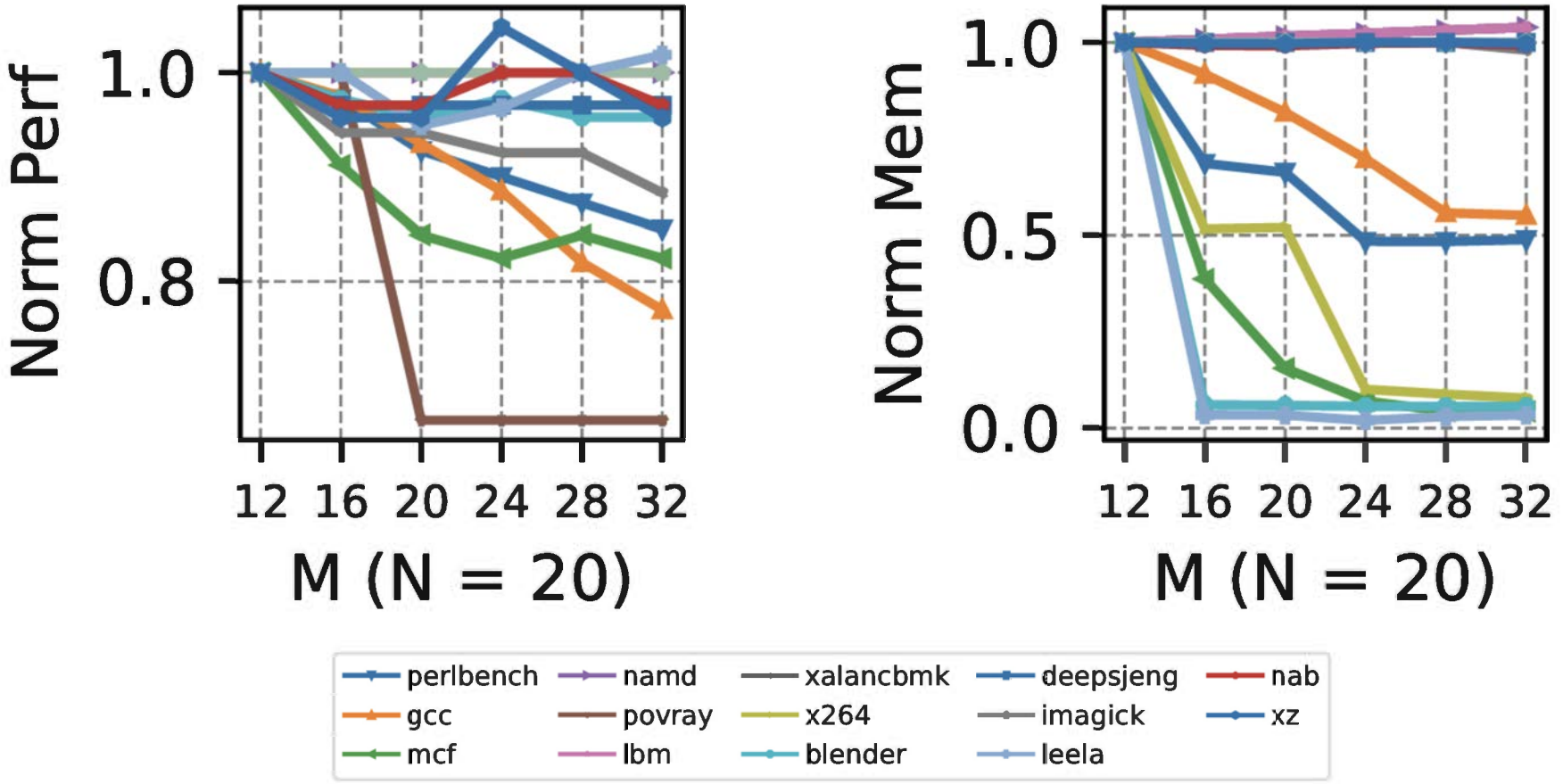




As N increases, more heap buffers share the same metadata table; FPN needs to scan a larger area.

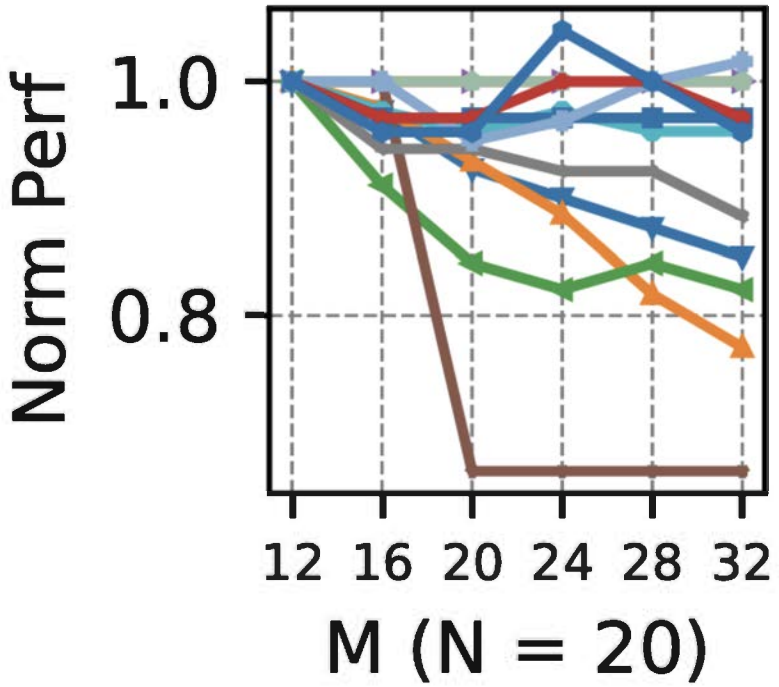
Explorations on Region Sizes and Block Sizes

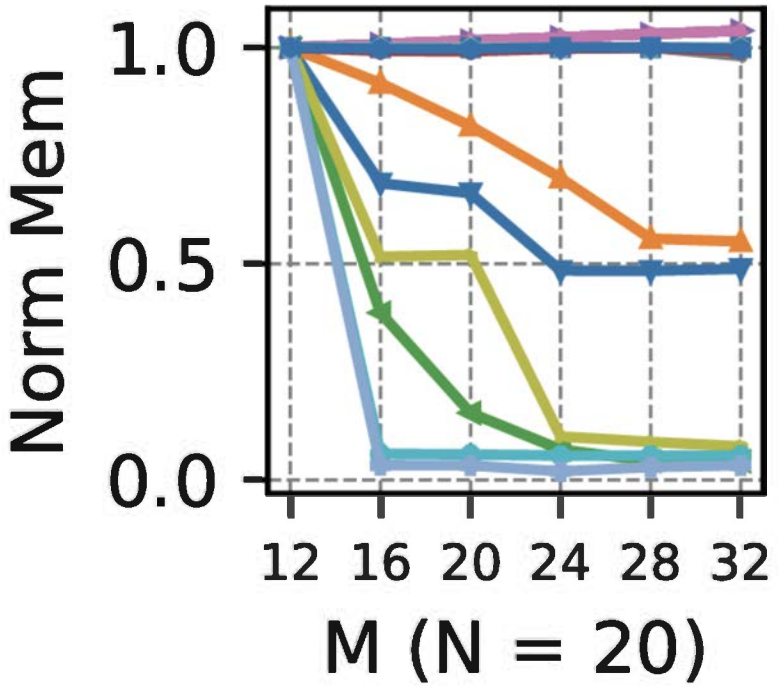
Block: aligned 2^M Bytes



Explorations on Region Sizes and Block Sizes

Block: aligned 2^M Bytes





As M increases, FPN converges to GC. But FPN has a lightweight shadow table to label the locations that need to be scanned.

Thank you!