Enhancing Memory Error Detection for Large-Scale Applications and Fuzz testing

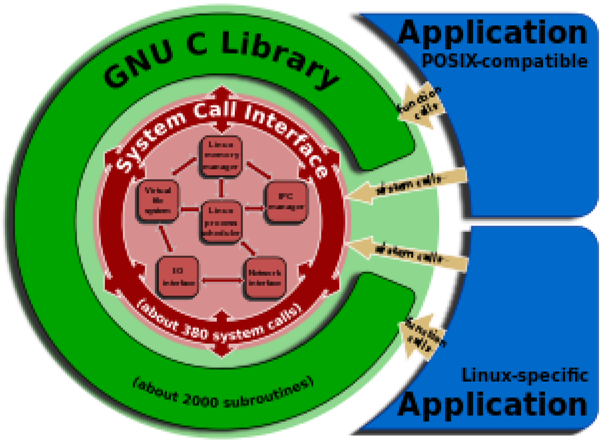
Wookhyun Han, Byunggil Joe, Byoungyoung Lee[*],
Chengyu Song[†], Insik Shin

KAIST, [*]Purdue, [†]UCR

Memory error

Heartbleed

Shellshock



glibc: getaddrinfo stack-based buffer overflow

- Information leakage – Heartbleed
- Privilege escalation – Shellshock
- Remote code execution – Shellshock, glibc, Conficker

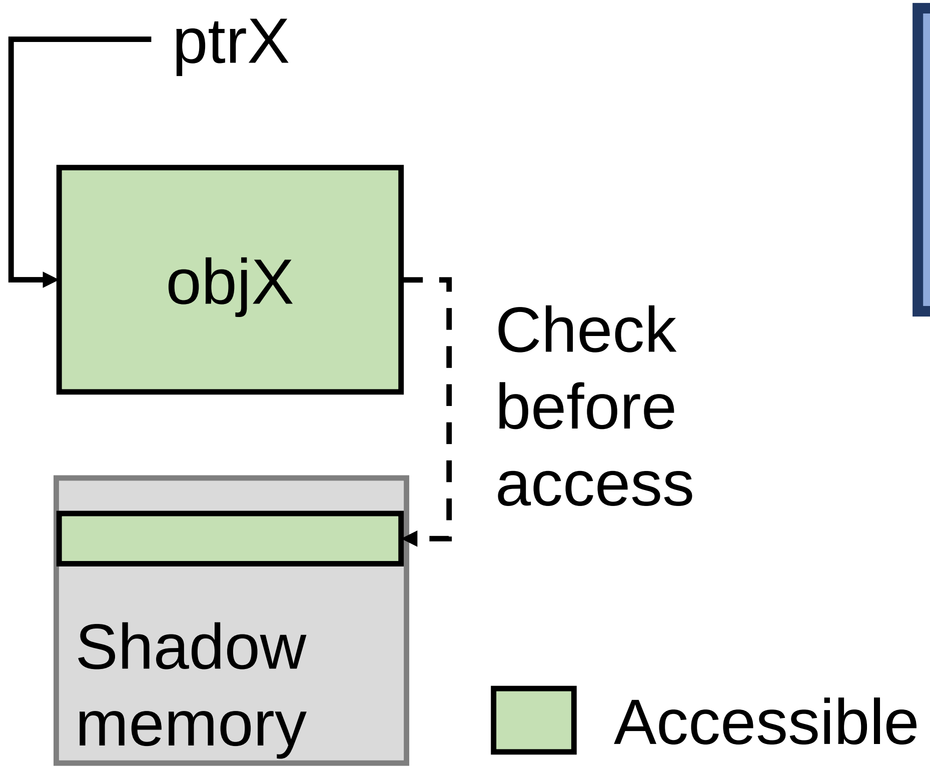
Memory error detection

- **Pointer-based [SoftBound+CETS, Intel MPX]**
  - Hardware support (cannot detect temporal memory errors)
  - Challenges to support complex applications

- **Redzone-based [AddressSanitizer (ASan)]**
  - **Compatible** to complex applications
  - **Most popular in practice**
    - ➔ Google Chrome, Mozilla Firefox, Linux Kernel
    - ➔ American Fuzzy Lop (AFL), ClusterFuzz, OSS-Fuzz

Redzone-based memory error detection

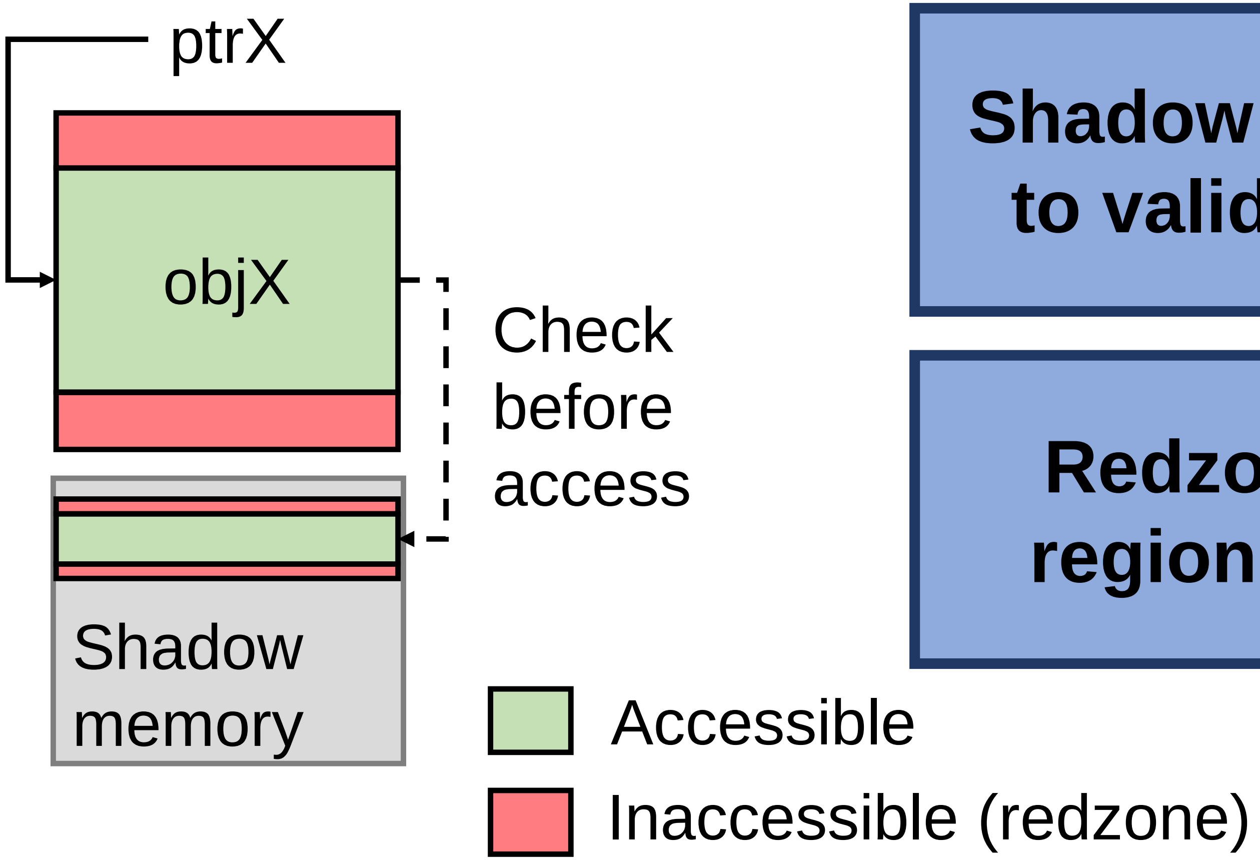
- **Buffer overflow (spatial memory errors)**



Shadow memory: a bitmap to validate all addresses

Redzone-based memory error detection

- **Buffer overflow (spatial memory errors)**

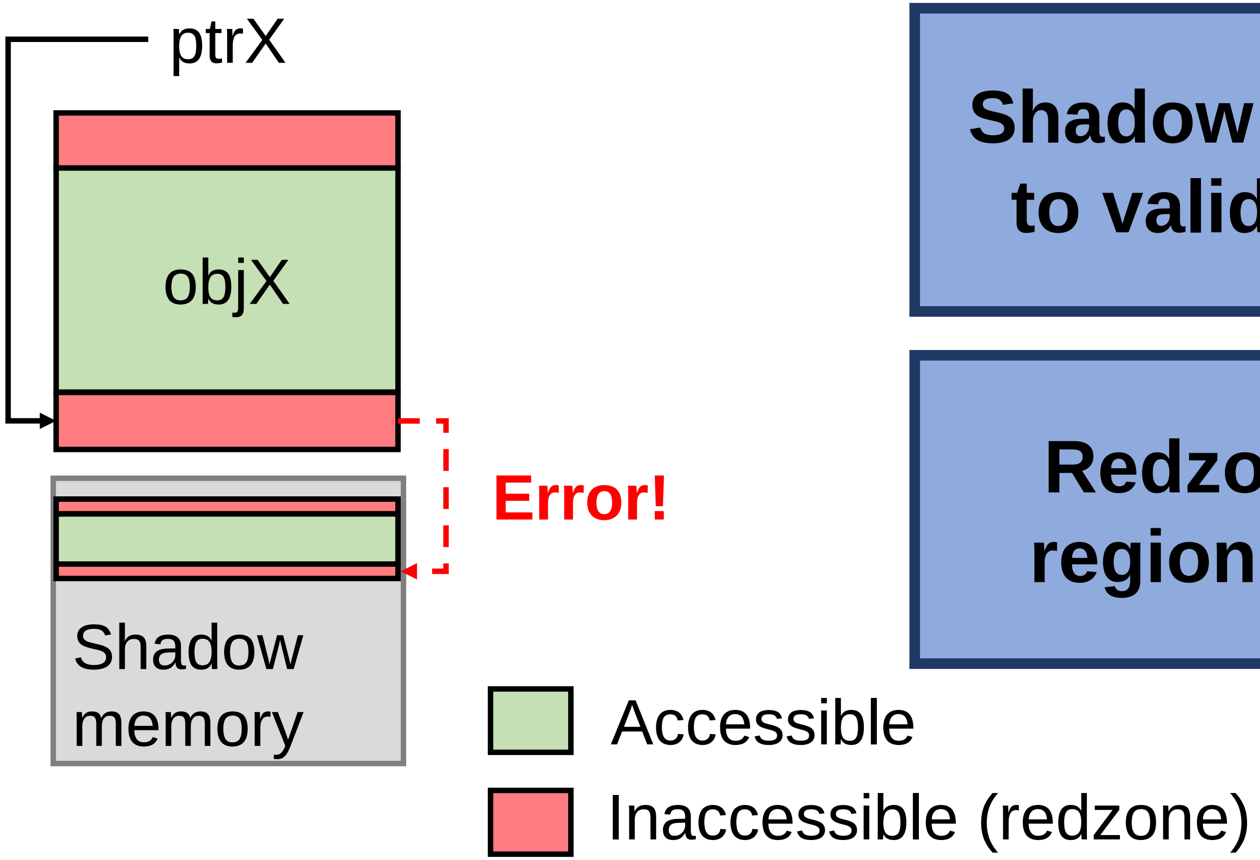


Shadow memory: a bitmap to validate all addresses

Redzone: inaccessible region between objects

Redzone-based memory error detection

- **Buffer overflow (spatial memory errors)**

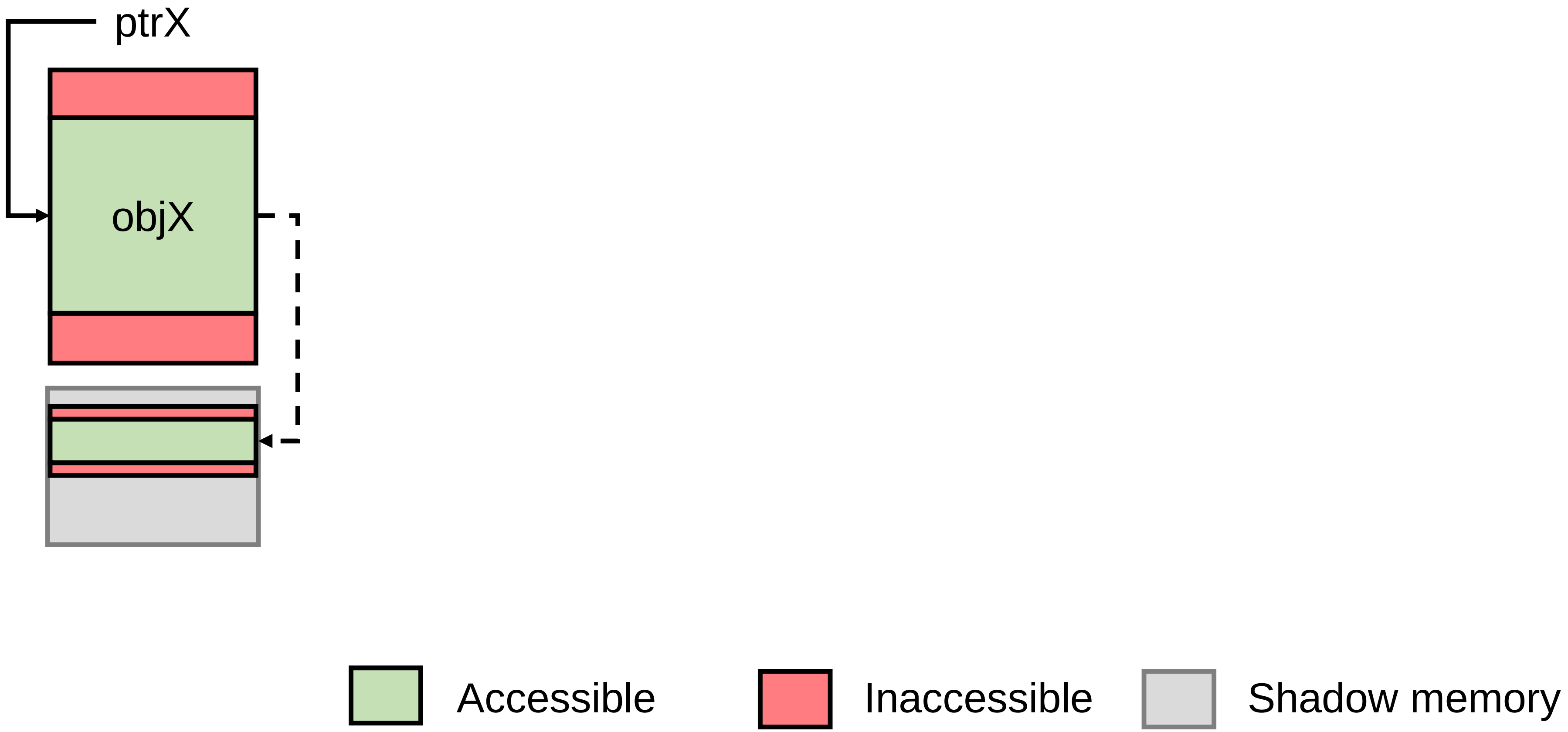


Shadow memory: a bitmap to validate all addresses

Redzone: inaccessible region between objects

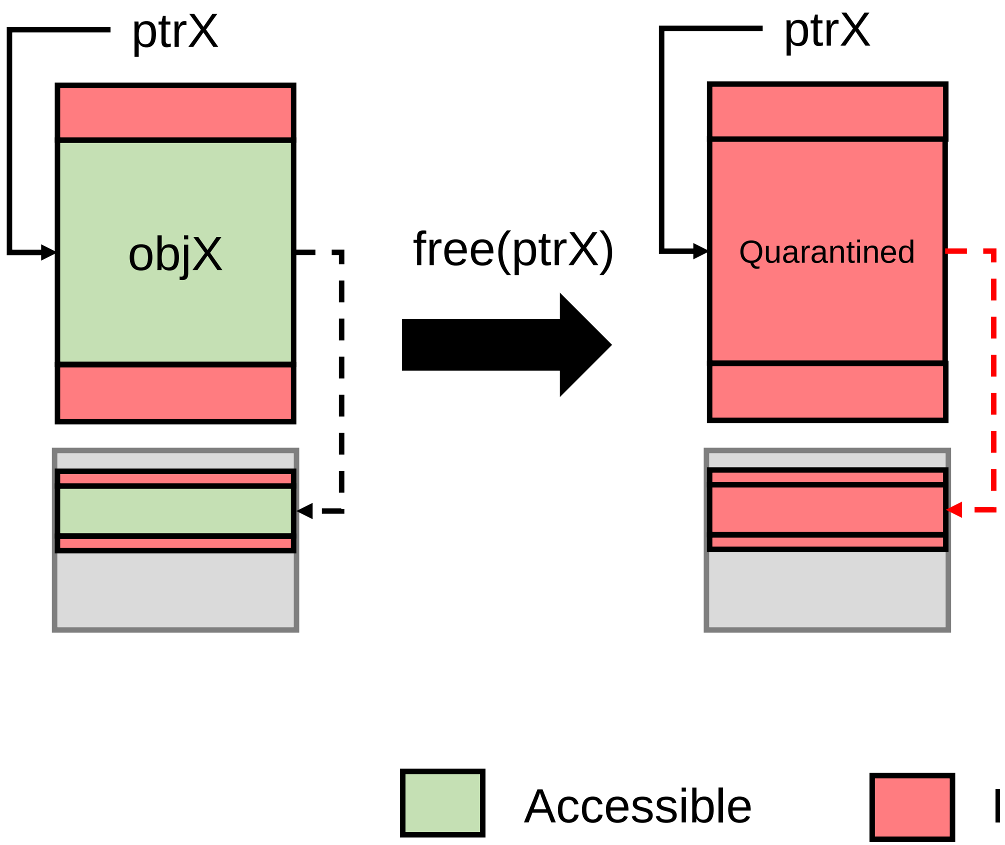
Redzone-based memory error detection

- **Use-after-free (temporal memory errors)**



Redzone-based memory error detection

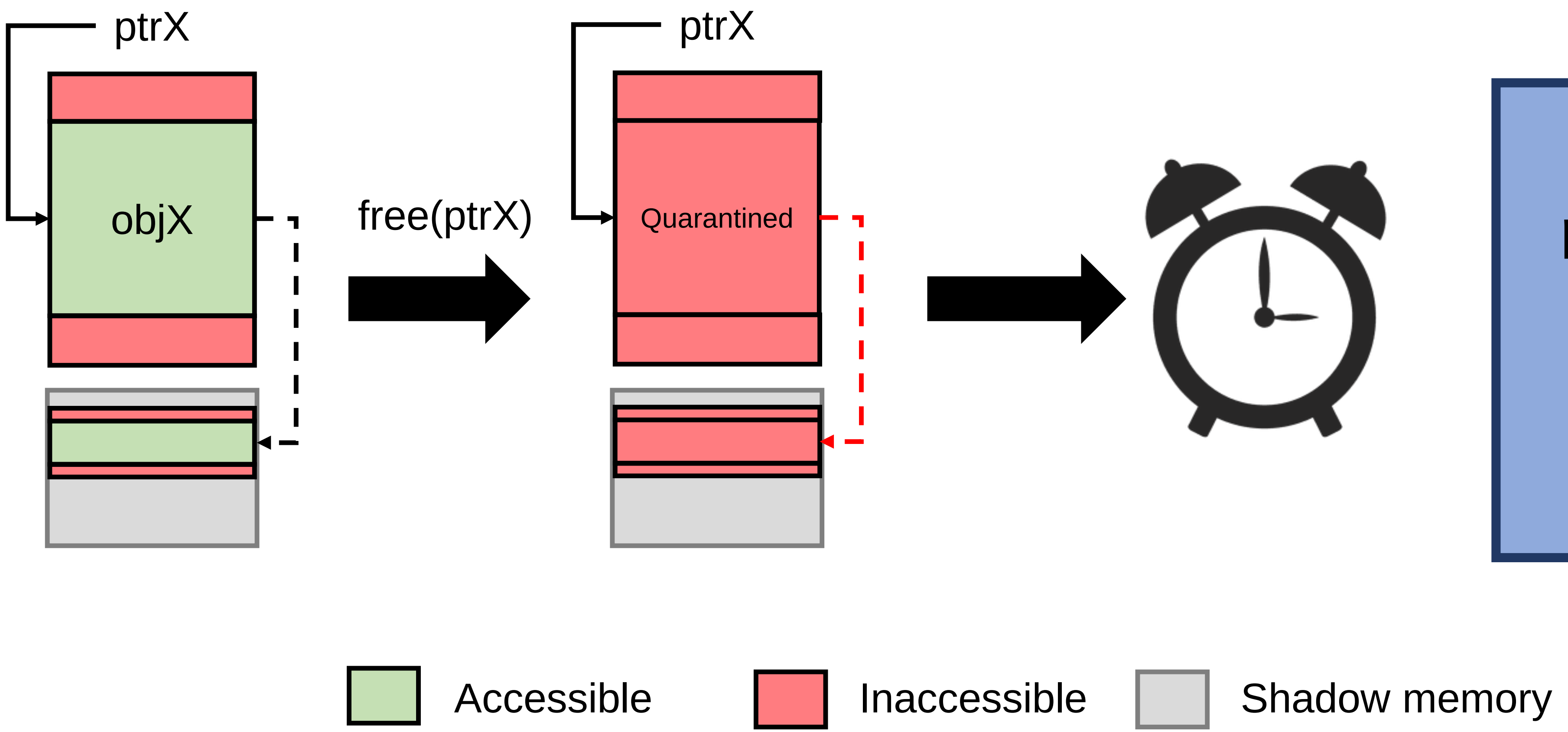
- **Use-after-free (temporal memory errors)**



Region is invalidated and quarantined, but not actually deallocated

Redzone-based memory error detection

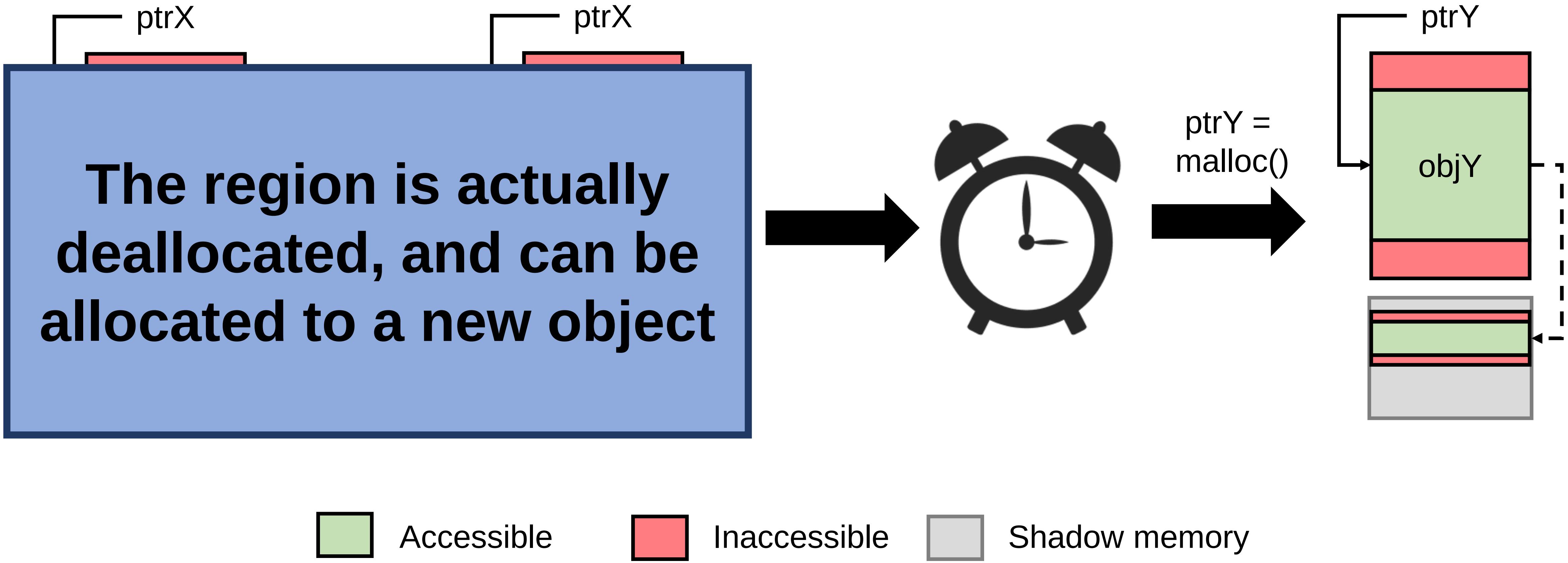
- **Use-after-free (temporal memory errors)**



Hold the region until quarantine zone is full (FIFO)

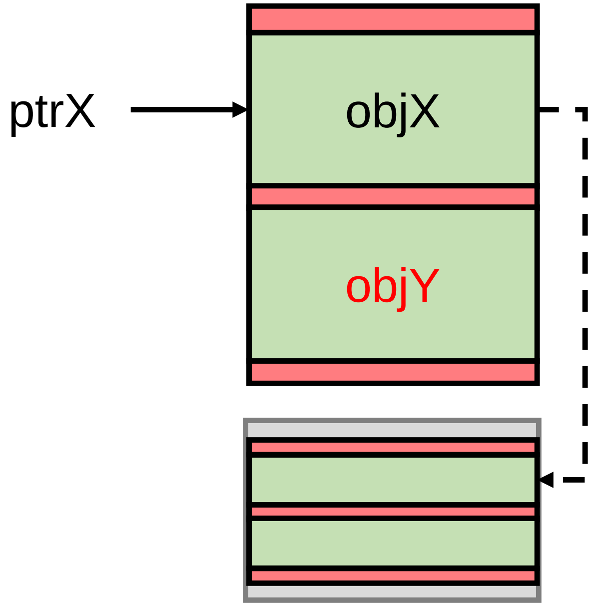
Redzone-based memory error detection

- **Use-after-free (temporal memory errors)**



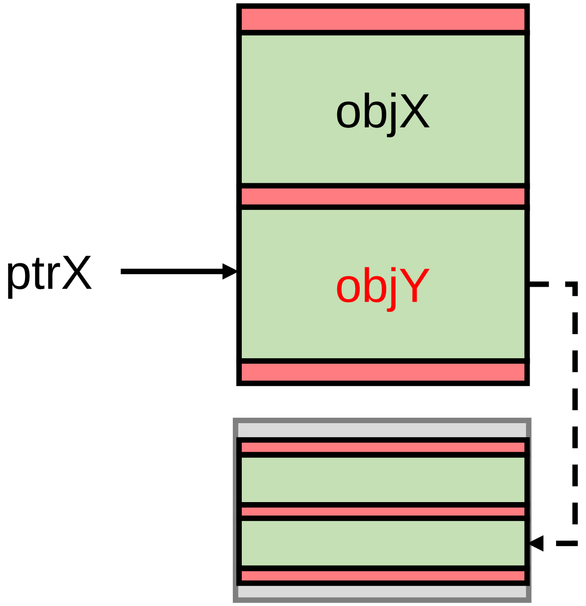
Limitations of redzone-based approach

1. What if a pointer accesses *beyond* redzone?



Limitations of redzone-based approach

1. What if a pointer accesses *beyond* redzone?

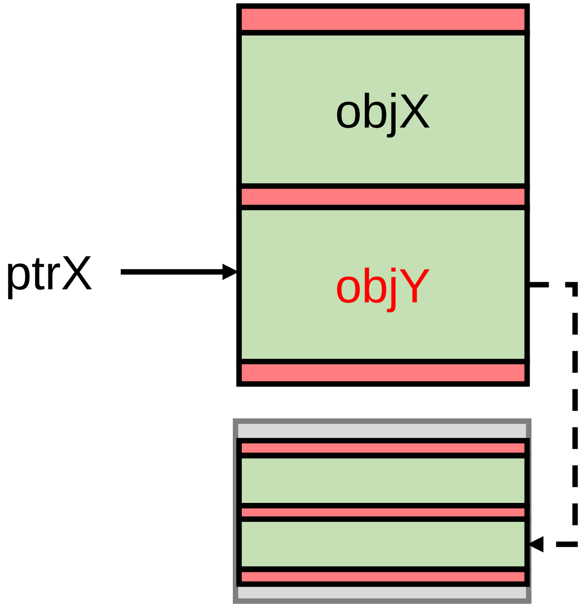


Spatial memory error

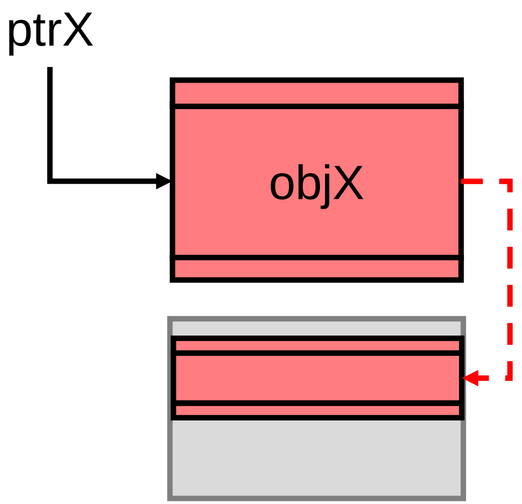
Limitations of redzone-based approach

1. What if a pointer accesses *beyond* redzone?

2. What if a dangling pointer accesses *after* another object is allocated in the region?



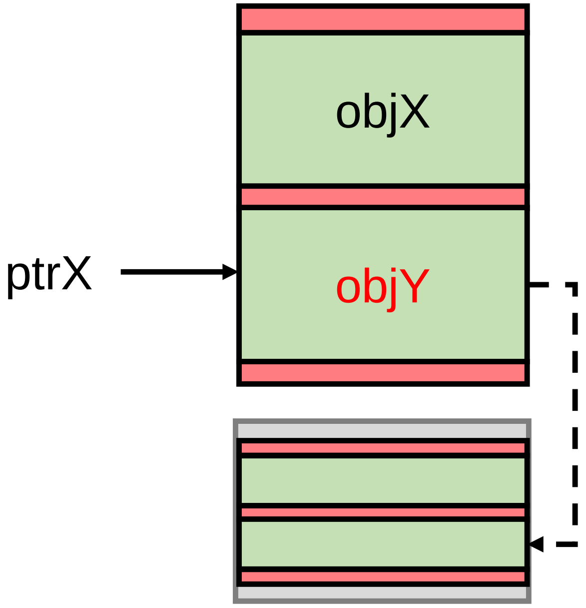
Spatial memory error



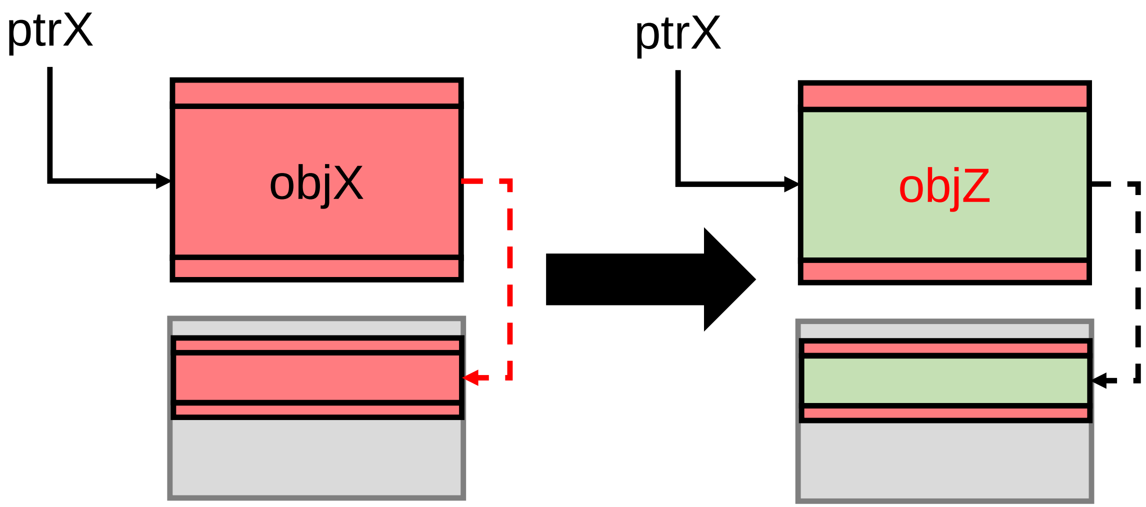
Limitations of redzone-based approach

1. What if a pointer accesses *beyond* redzone?

2. What if a dangling pointer accesses *after* another object is allocated in the region?



Spatial memory error

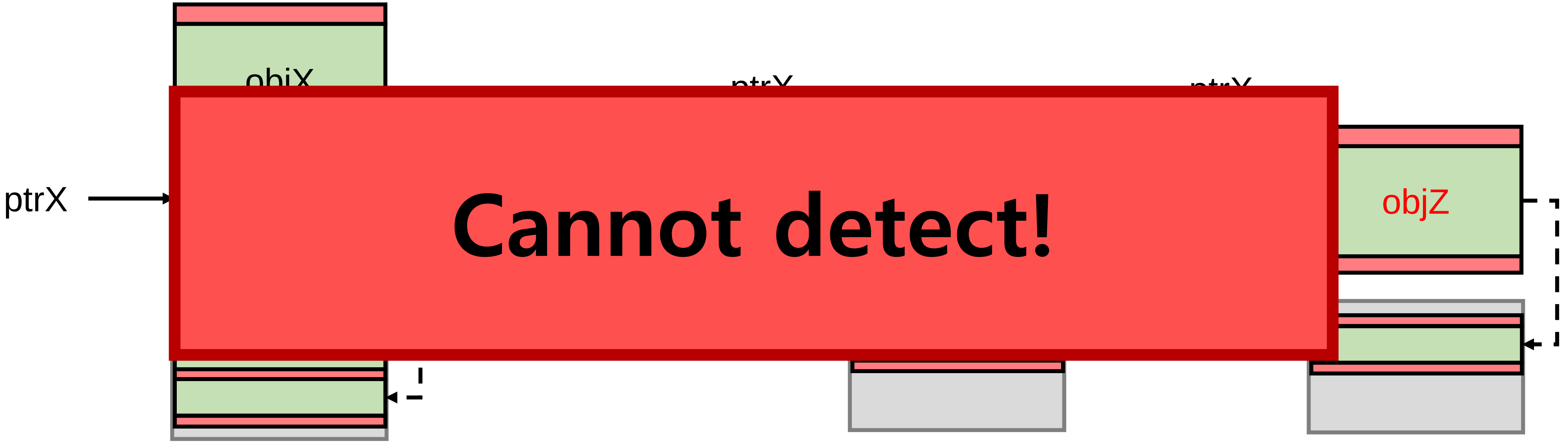


Temporal memory error

Limitations of redzone-based approach

1. What if a pointer accesses *beyond* redzone?

2. What if a dangling pointer accesses *after* another object is allocated in the region?

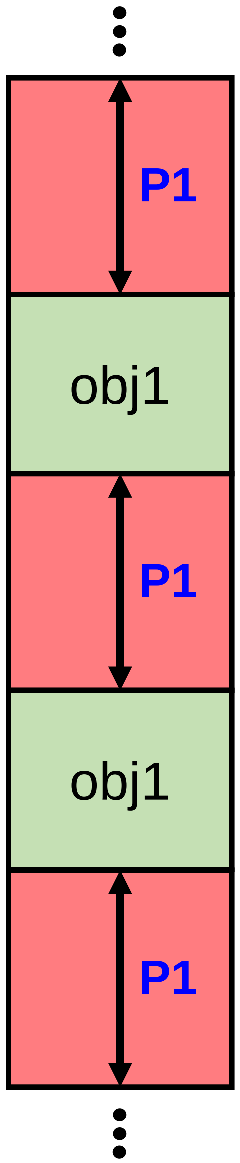


Spatial memory error

Temporal memory error

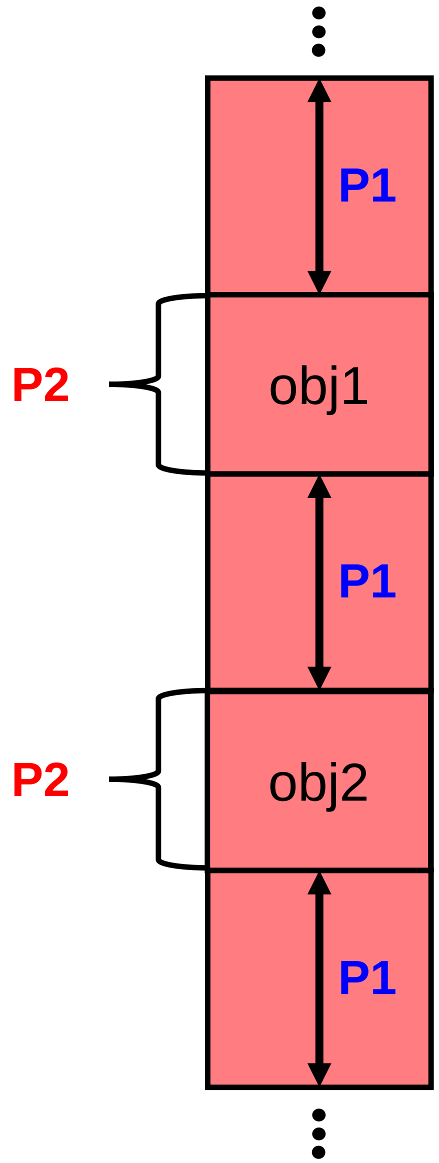
Motivation

- To enhance detectability of redzone-based memory error detection
  - **P1. Large gap** to detect spatial memory errors
  - **P2. Large quarantine zone** to detect temporal memory errors



Motivation

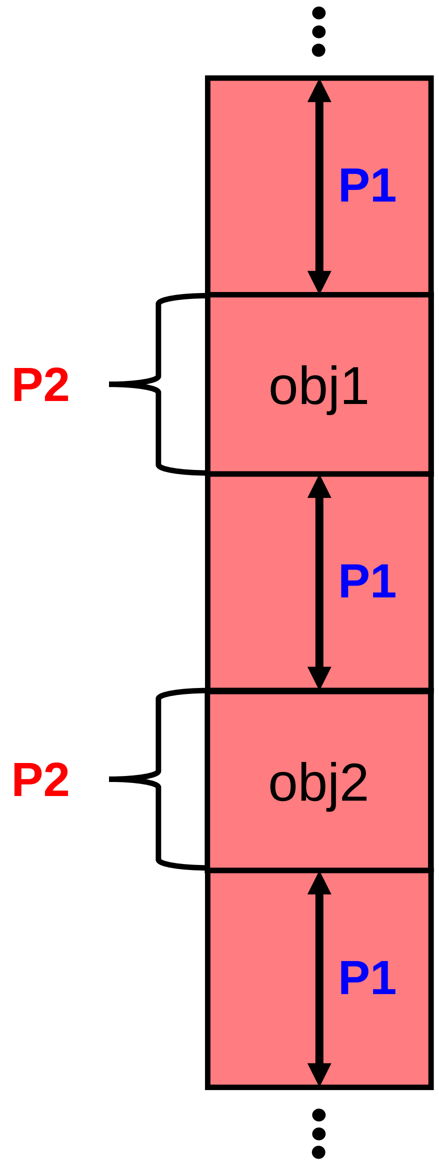
- To enhance detectability of redzone-based memory error detection
  - **P1. Large gap** to detect spatial memory errors
  - **P2. Large quarantine zone** to detect temporal memory errors



Motivation

- To enhance detectability of redzone-based memory error detection
  - **P1. Large gap** to detect spatial memory errors
  - **P2. Large quarantine zone** to detect temporal memory errors

Huge physical memory required



MEDS overview

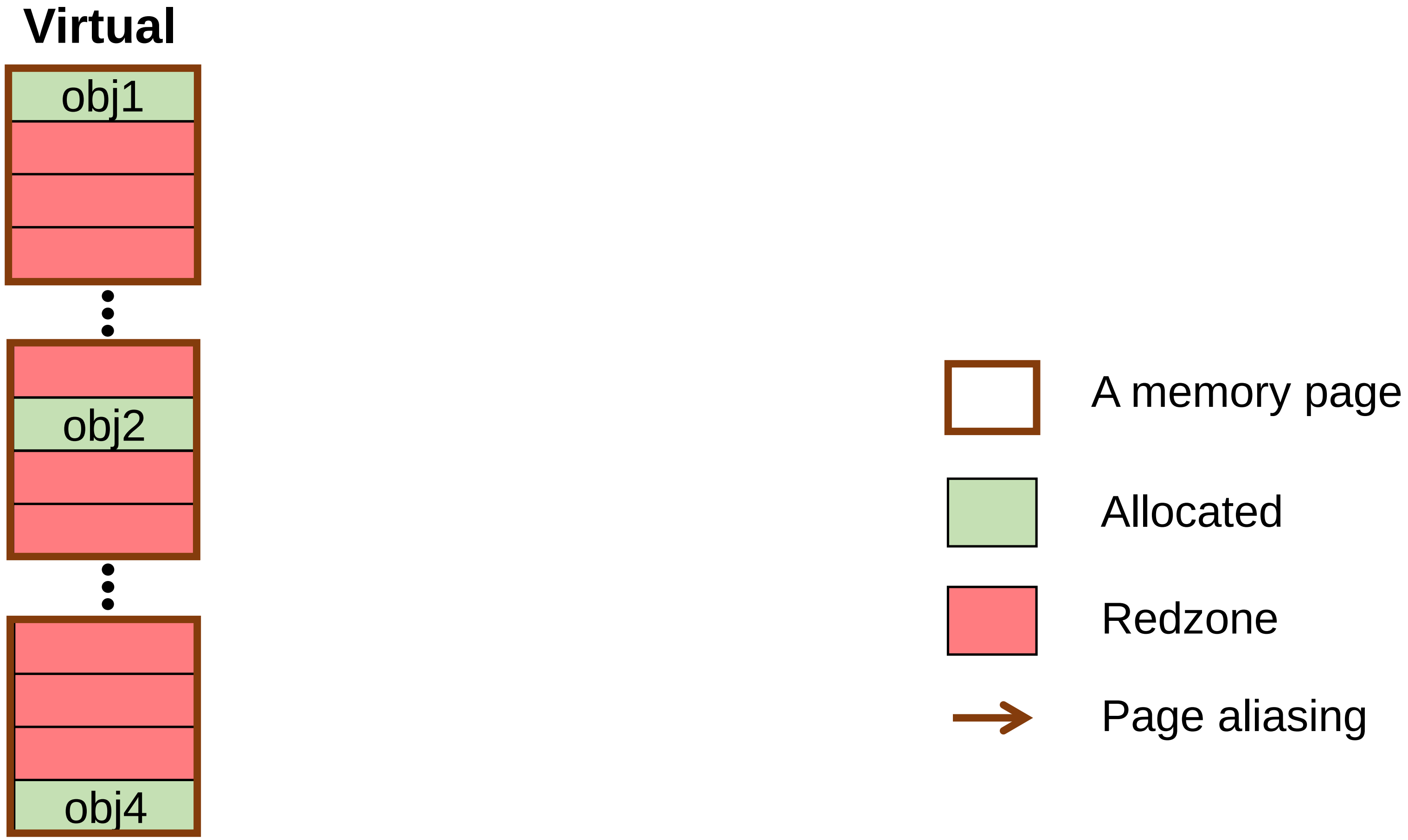
- Enhances detectability of redzone-based memory error detection

- **Idea: Fully utilize 64-bit virtual address** space to support
  - **P1. Large gap** to detect spatial error
  - **P2. Large quarantine zone** to detect temporal error

- **Approach**: minimize physical memory use
  - **Page aliasing** allocator and **page protection**
  - **Hierarchical** memory error detection

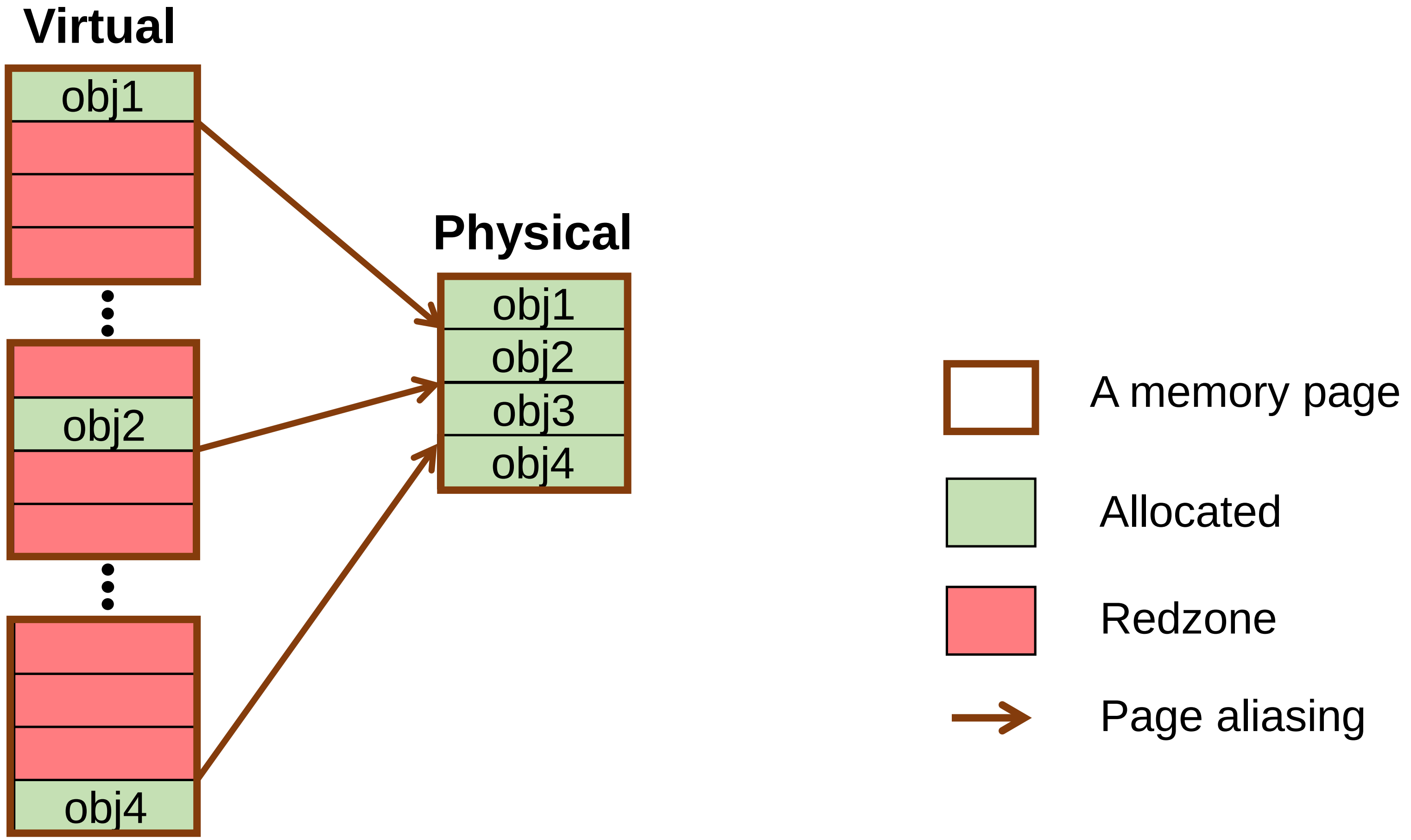
Page aliasing (P1)

- Maps multiple virtual pages to single physical page



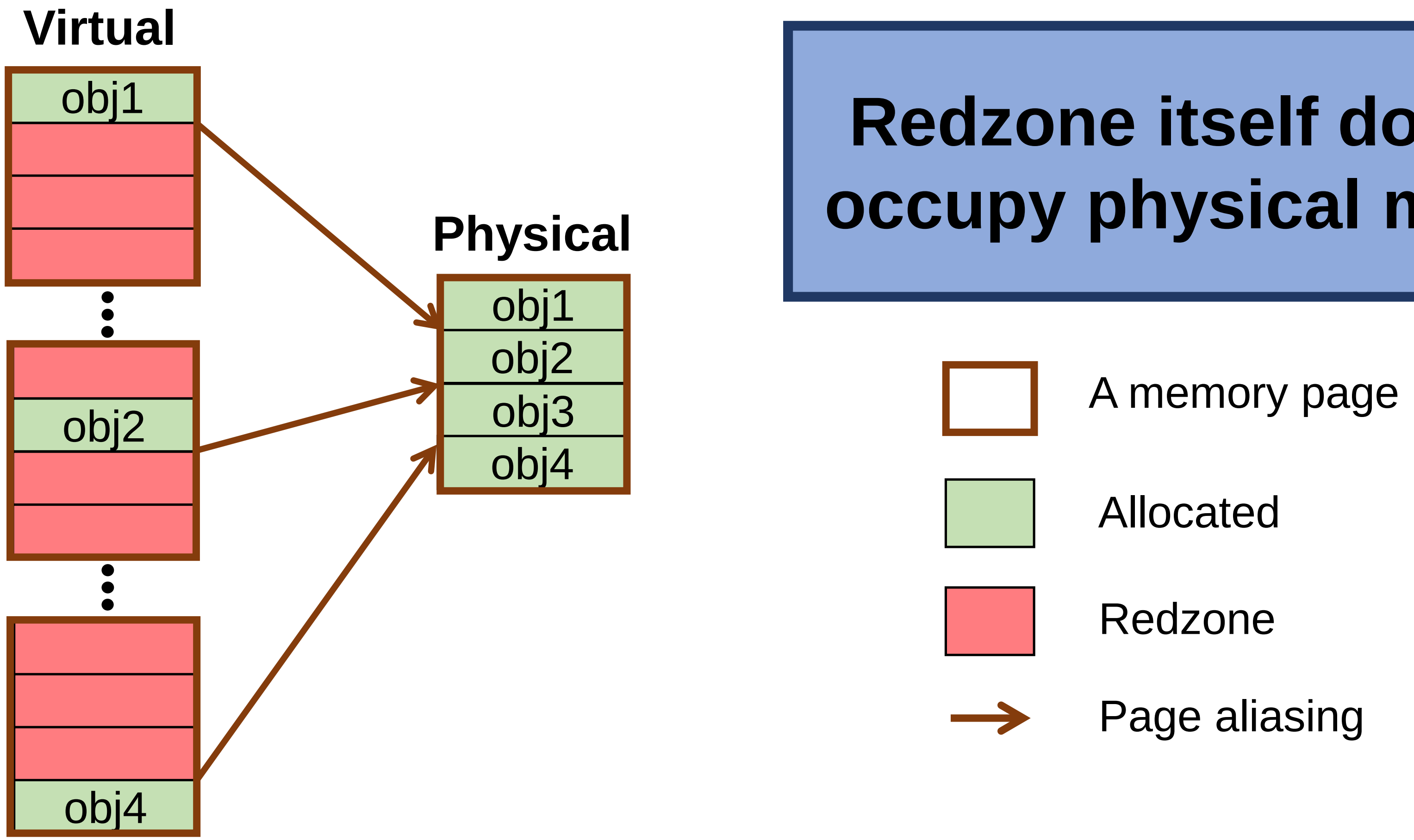
Page aliasing (P1)

- Maps multiple virtual pages to single physical page



Page aliasing (P1)

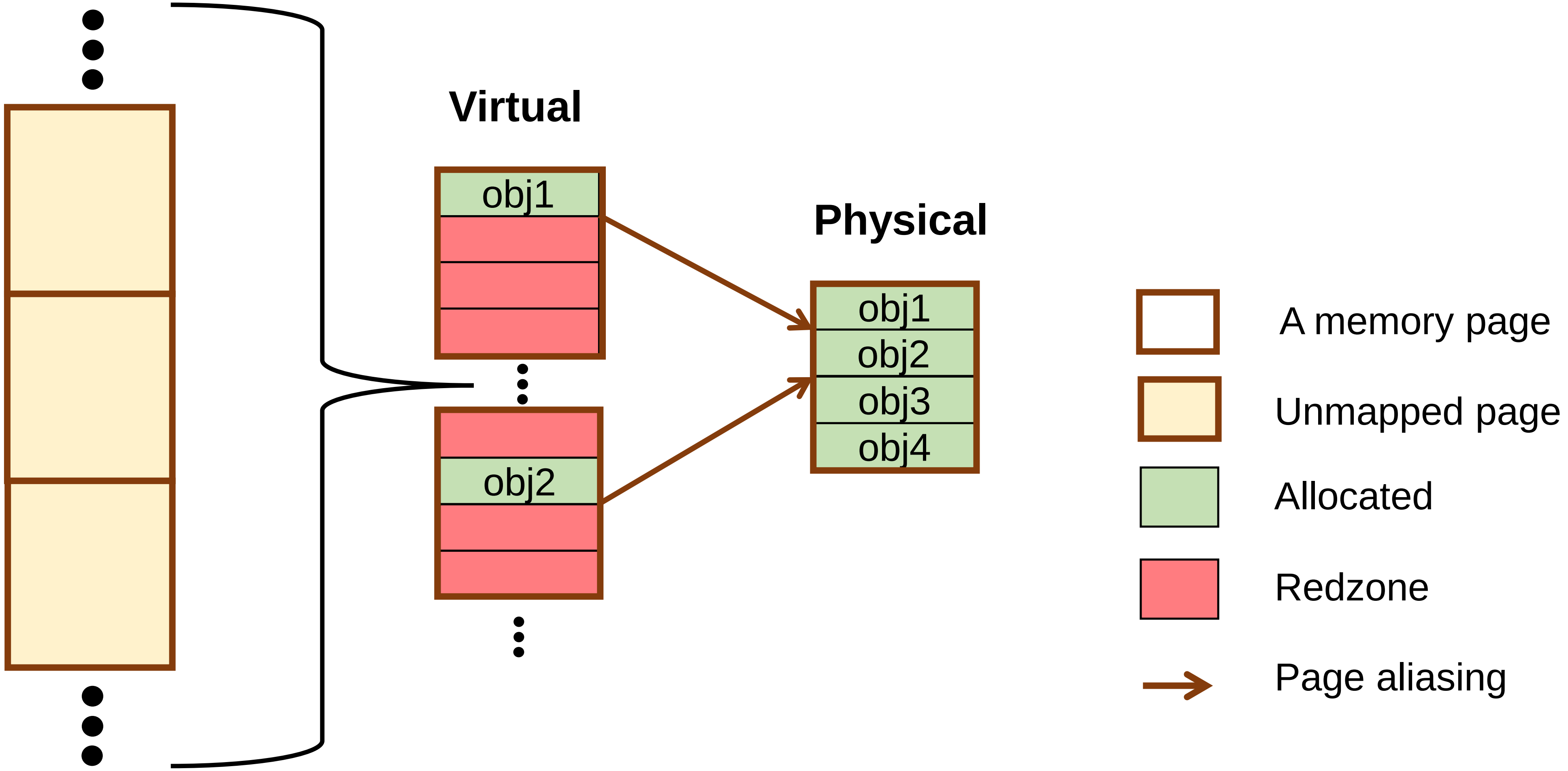
- Maps multiple virtual pages to single physical page



Redzone itself does not occupy physical memory

Page protection (P1)

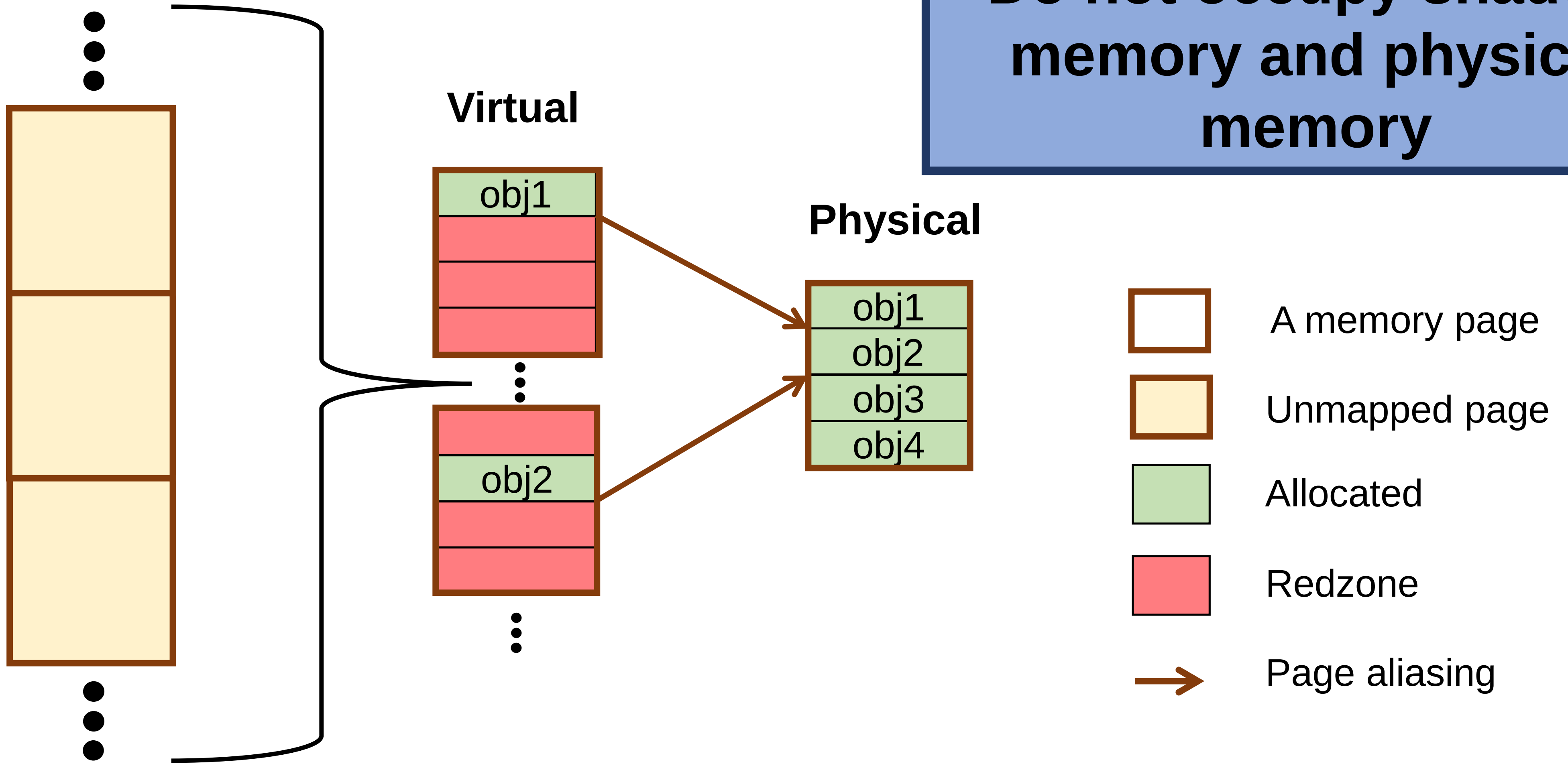
- Redzone only pages are unmapped



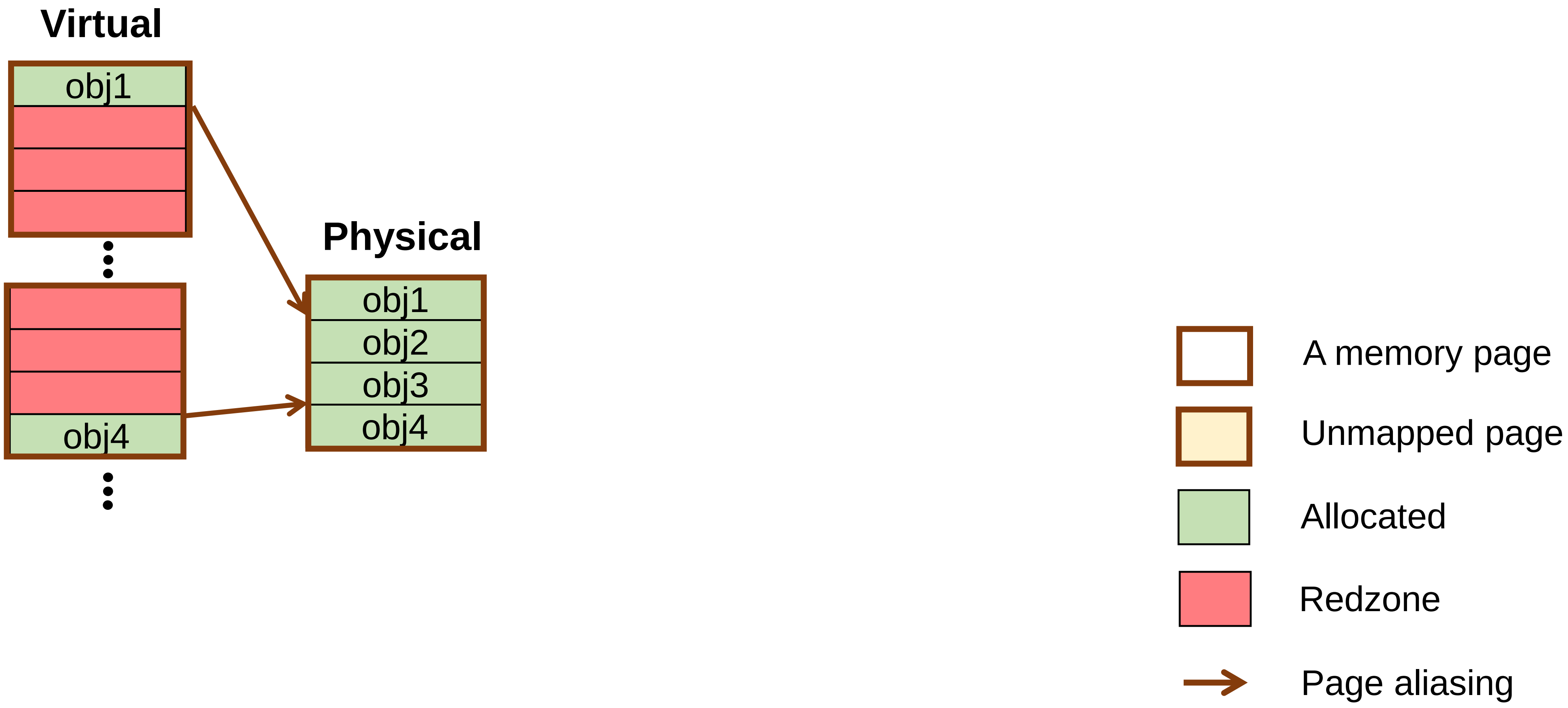
Page protection (P1)

- Redzone only pages are unmapped

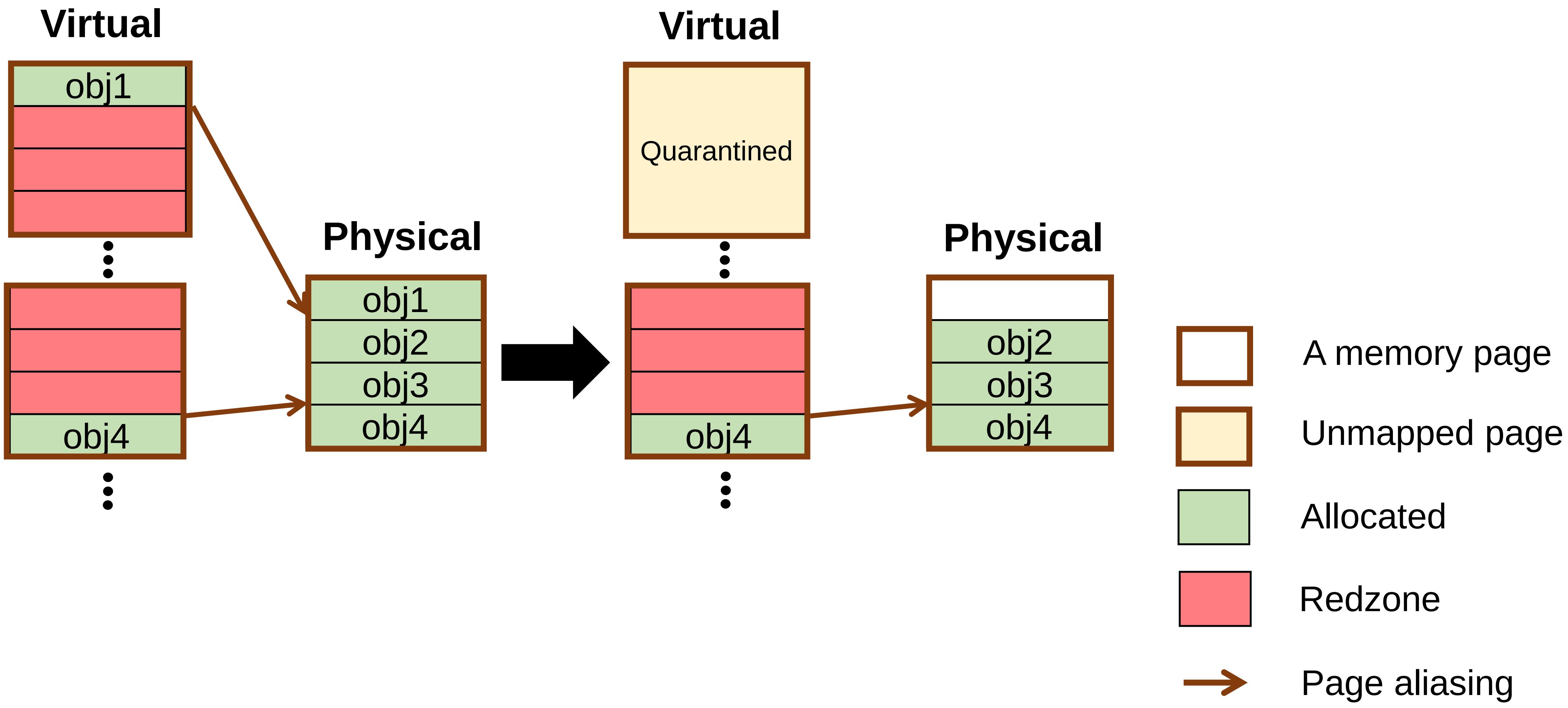
Do not occupy shadow memory and physical memory



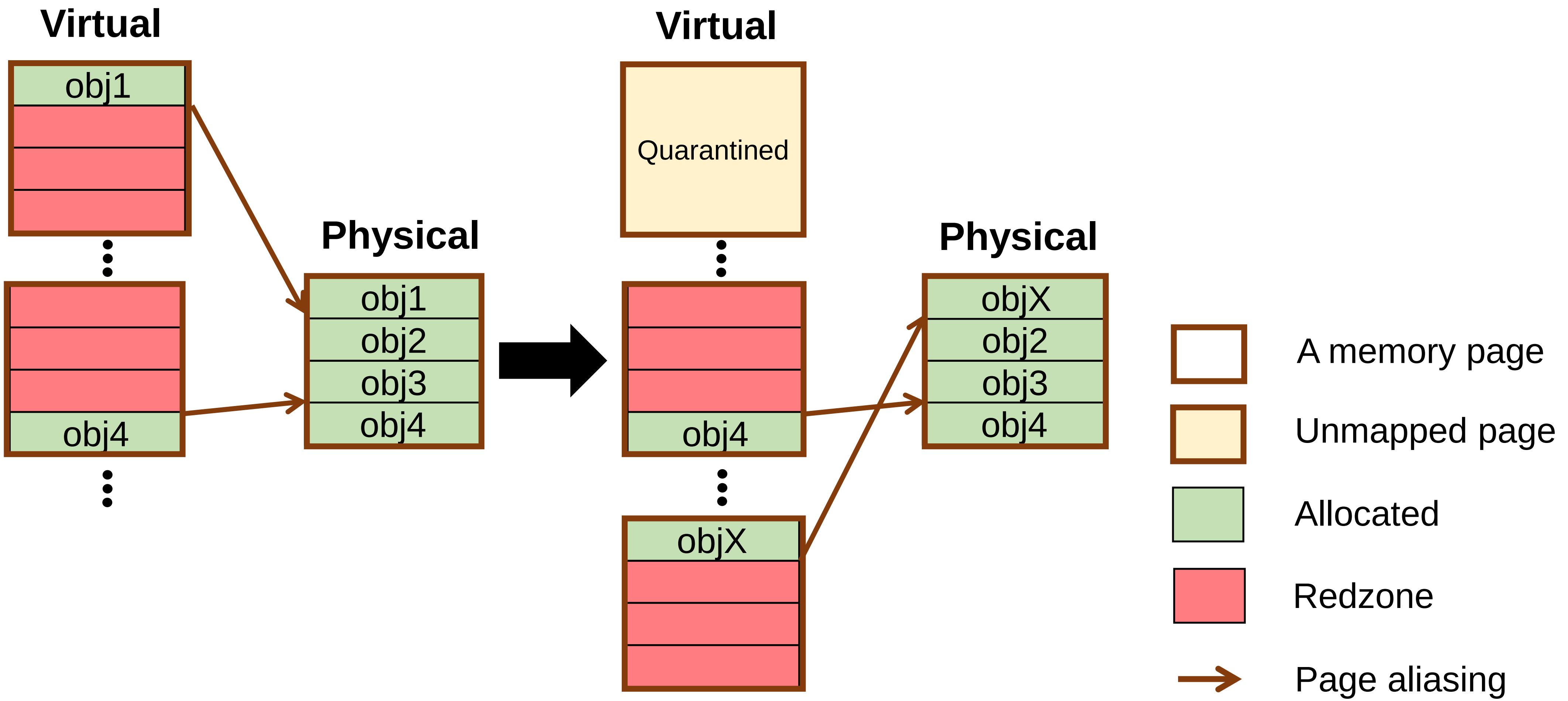
Page aliasing & Page protection (P2)



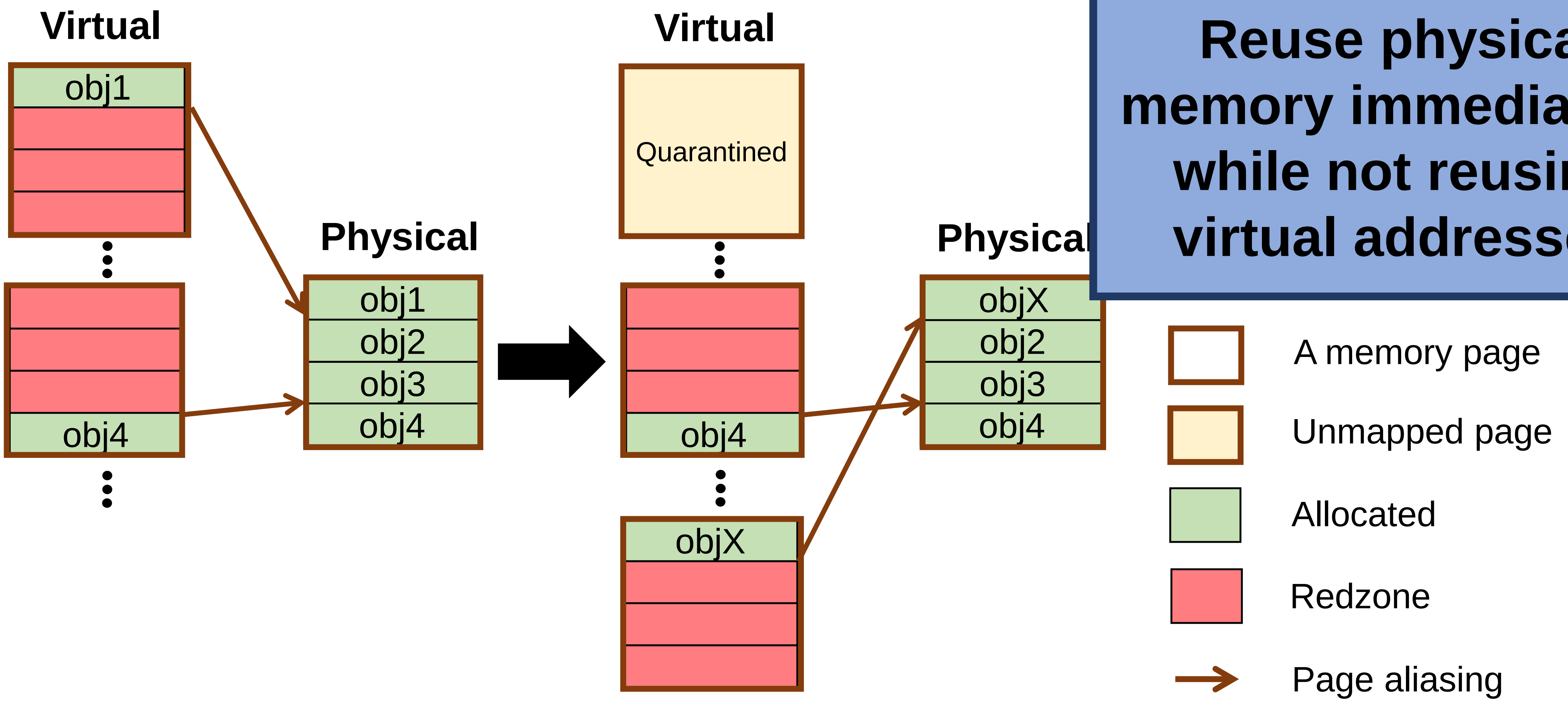
Page aliasing & Page protection (P2)



Page aliasing & Page protection (P2)



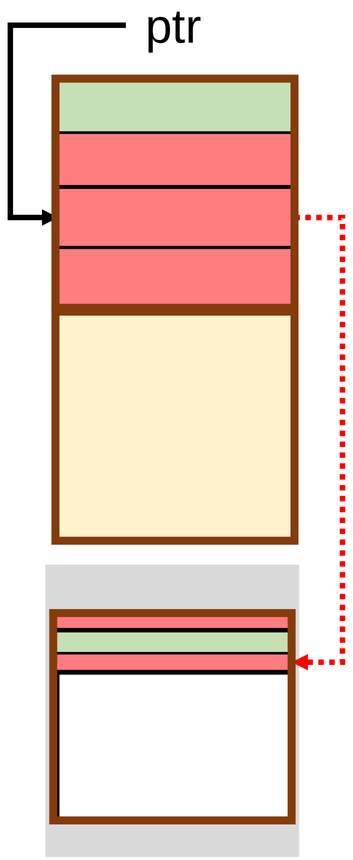
Page aliasing & Page protection (P2)



Reuse physical memory immediately, while not reusing virtual addresses

Hierarchical memory error detection

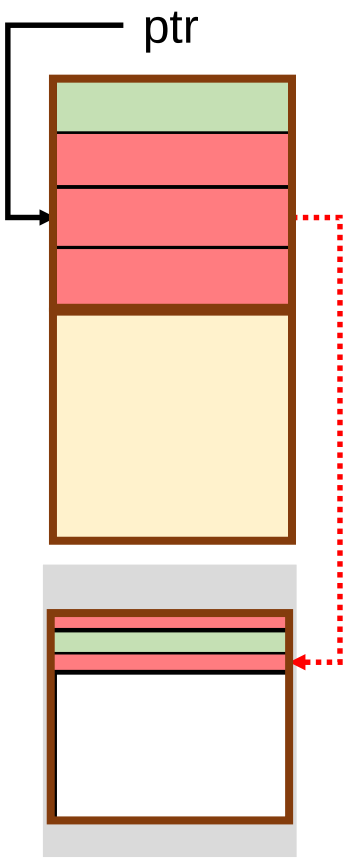
- Many different ways to represent redzones
  - ➔ Further optimizing physical memory uses



Hierarchical memory error detection

- Many different ways to represent redzones
  - ➔ Further optimizing physical memory uses

#1. Shadow memory is invalid

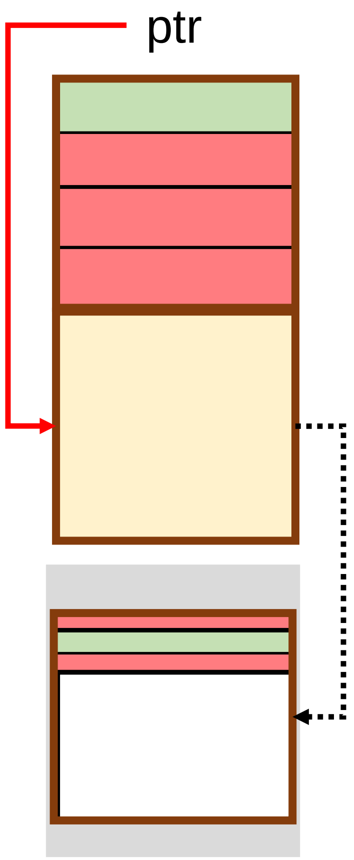


Hierarchical memory error detection

- Many different ways to represent redzones
  - ➔ Further optimizing physical memory uses

#1. Shadow memory is invalid

#2. Virtual page is unmapped



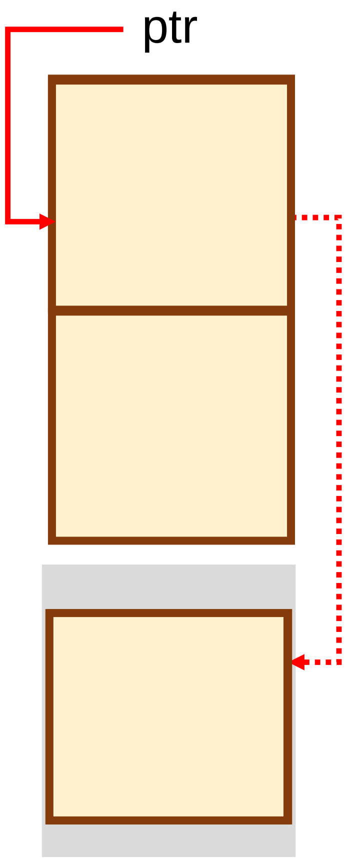
Hierarchical memory error detection

- Many different ways to represent redzones
  - ➔ Further optimizing physical memory uses

#1. Shadow memory is invalid

#2. Virtual page is unmapped

#3. Shadow memory is unmapped



Evaluation

- **Configuration**

	ASan	MEDS	Improv.
Redzone	8-1024 bytes	4MB	16,384x
Quarantine	128MB	80TB	65,536x

  - ASan cannot use configuration for MEDS (lack of memory)
- **Compatibility**
- **Performance:** 2 times slowdown
- **Detection (fuzz testing):** 68% more detection

Compatibility

- **Unit tests from real-world applications**
  - Test cases in Chrome, Firefox, Nginx
    - **All Passed**
- **Memory error unit tests**
  - ASan unit tests
    - **All Passed**
  - NIST Juliet test suites
    - **All Passed** except random access tests

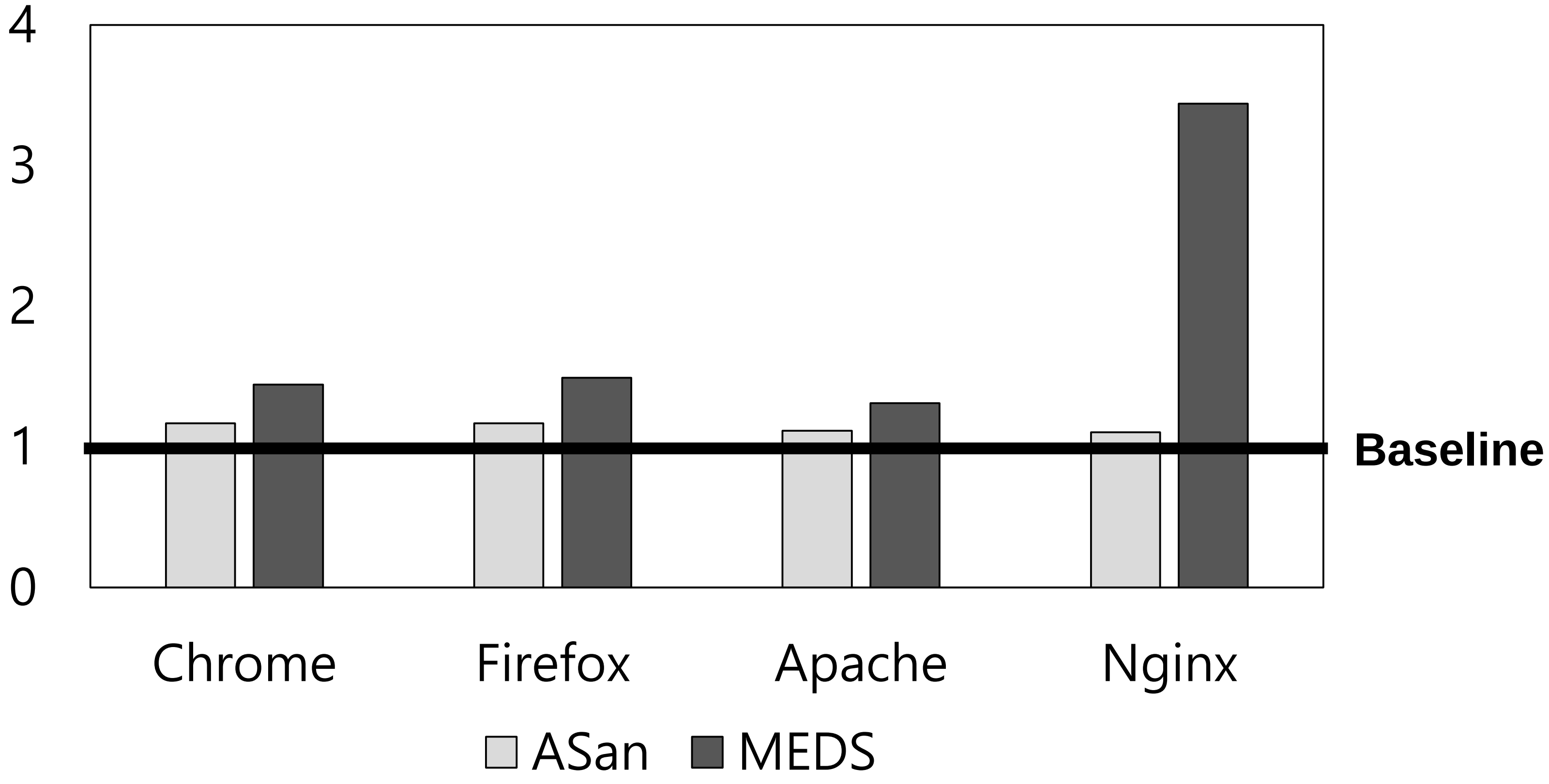
    ➔ ASan: 35% vs. MEDS: 98%

Micro-scale performance overhead

- **TLB misses**
  - 5 times more than ASan (more **virtual pages with page aliasing**)
- **Number of system calls**
  - mmap(), munmap(), and mremap()
  - 32 times more than ASan (**page aliasing** and **page protection**)
- **Memory footprint**
  - 218% more than baseline
  - 68% more than ASan (much larger **redzone** and **quarantine**)

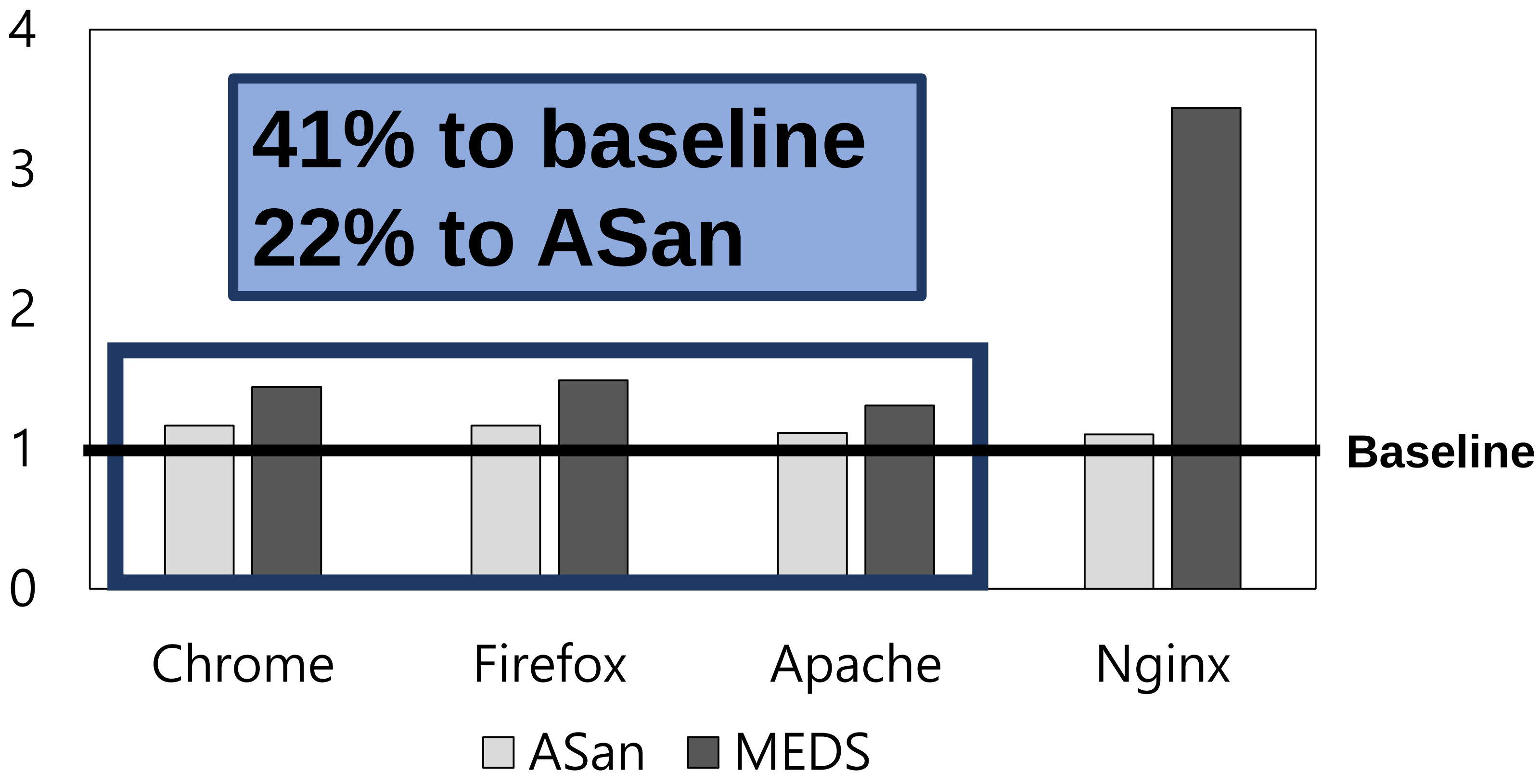
End-to-end performance overhead

- 108% compared to baseline, 86% to ASan



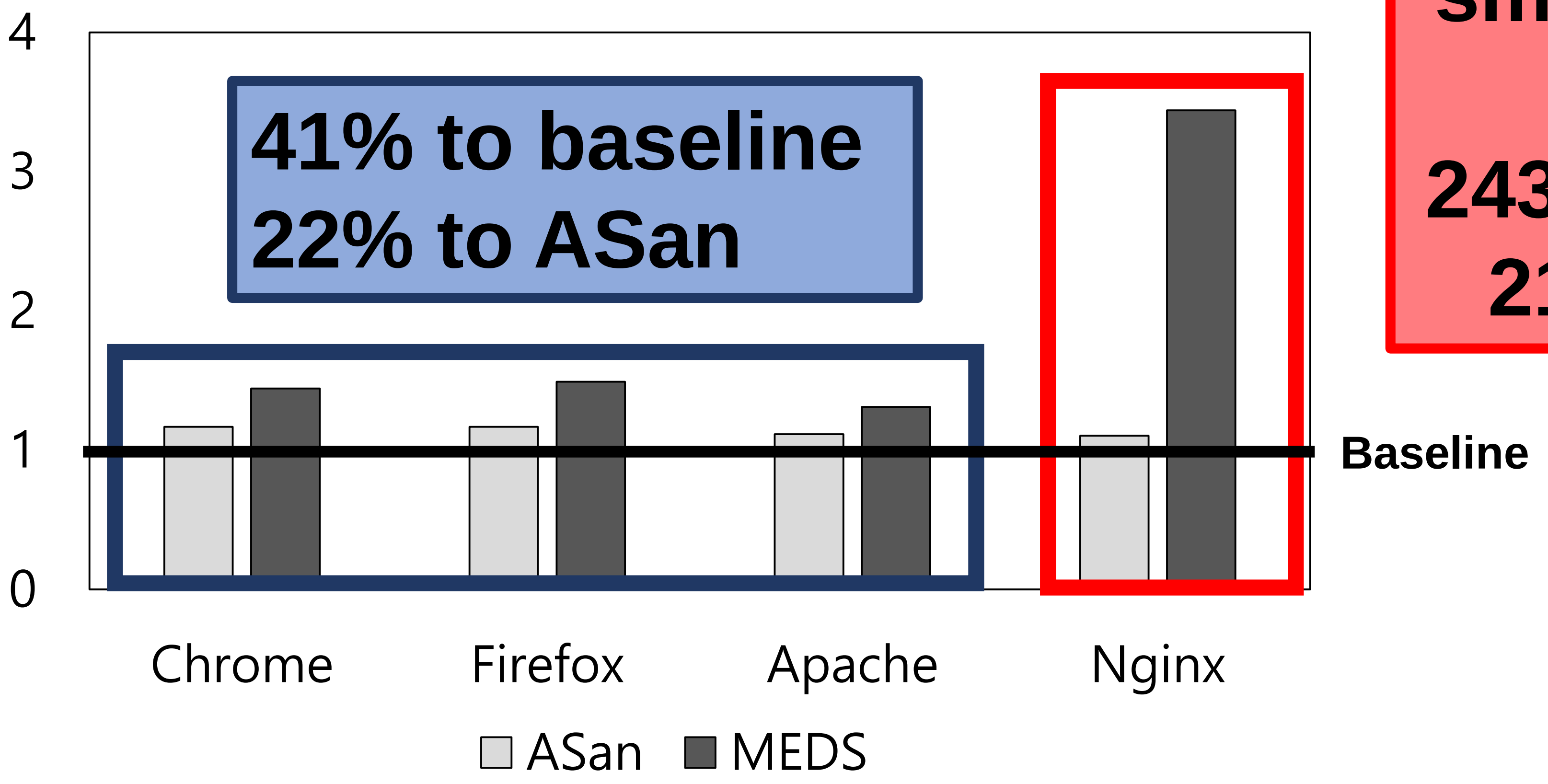
End-to-end performance overhead

- 108% compared to baseline, 86% to ASan



End-to-end performance overhead

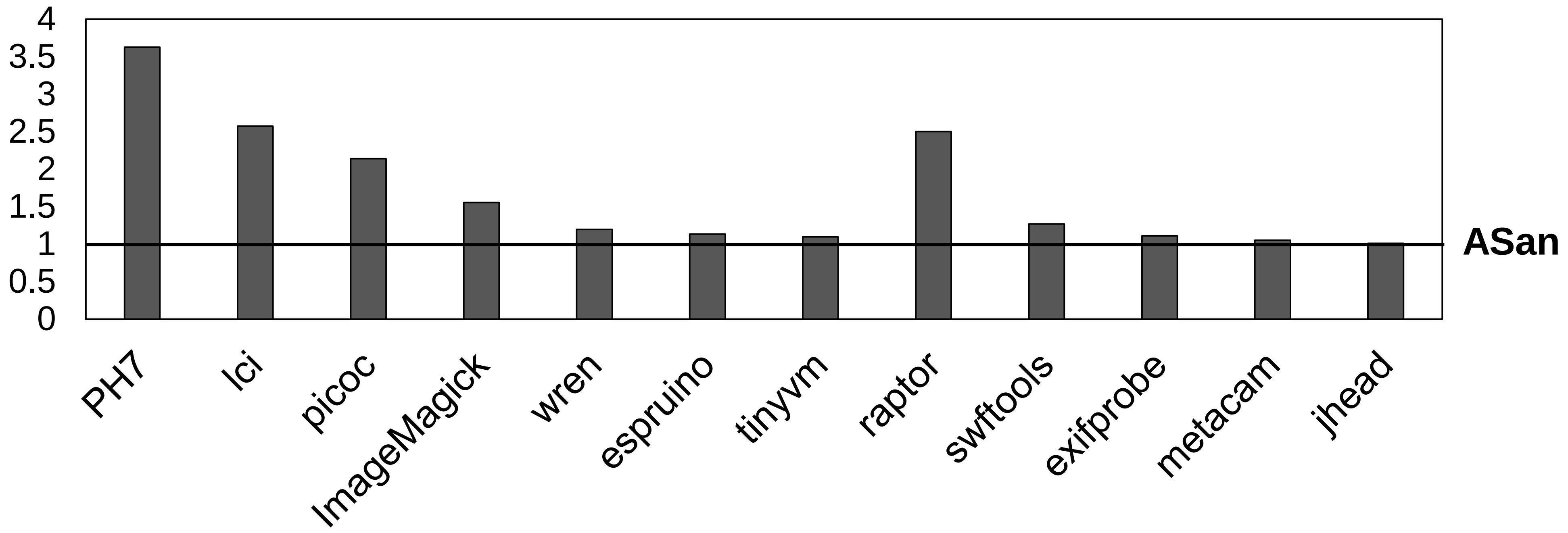
- 108% compared to baseline, 86% to ASan



Large number of small objects on stack
243% to baseline
211% to ASan

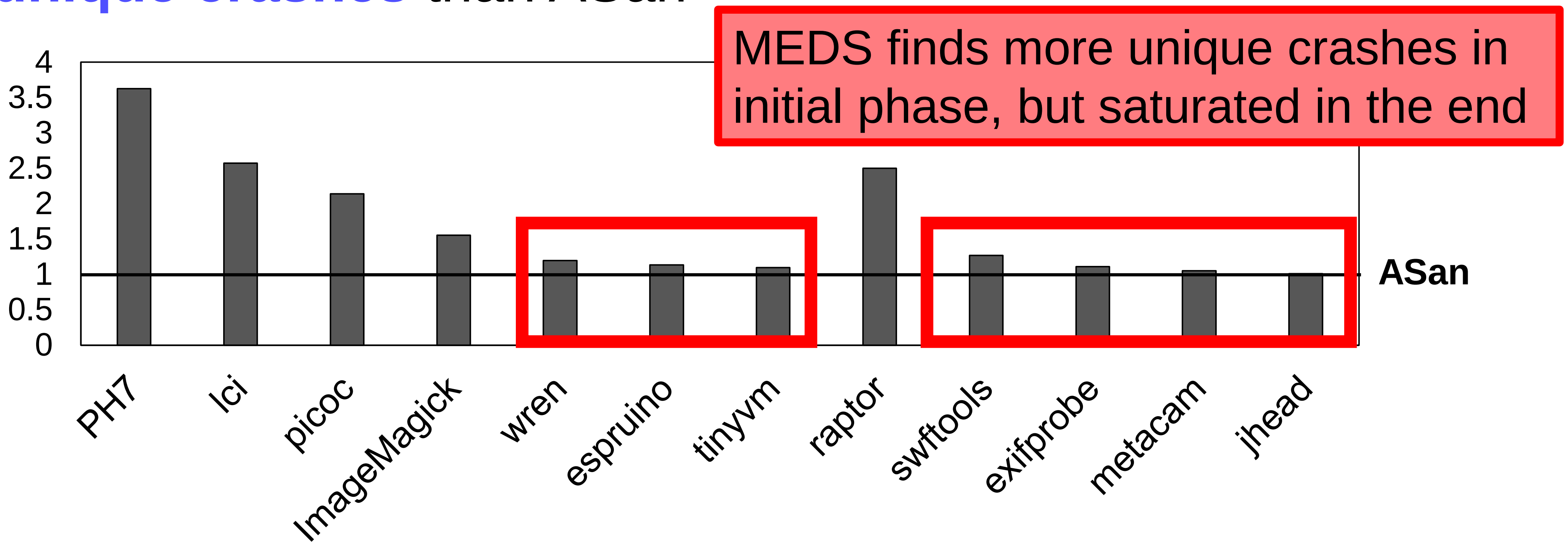
Detection (fuzz testing)

- Run AFL (8 cores, 6 hours)
- Despite the performance overhead, explore **68.3% more unique crashes** than ASan



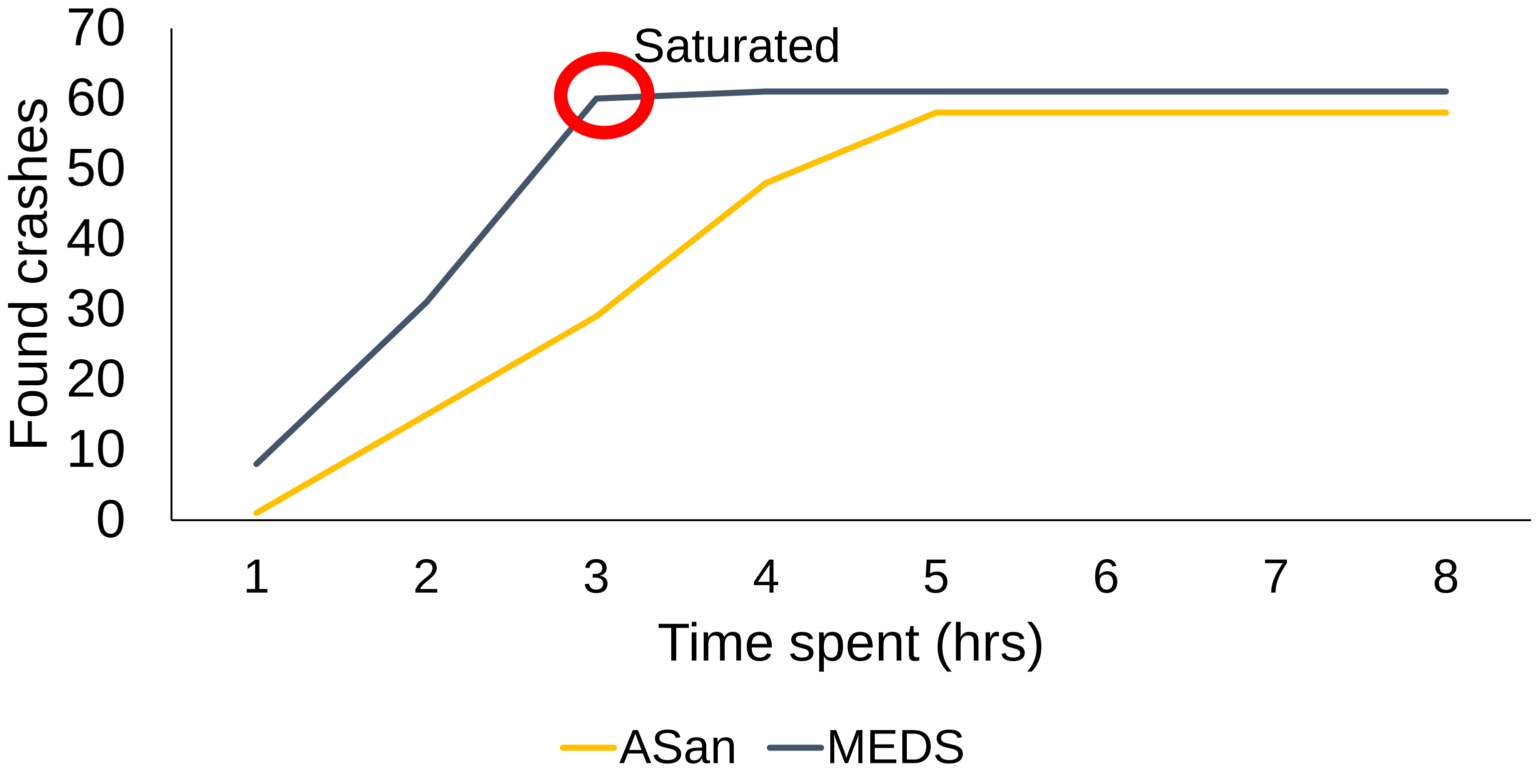
Detection (fuzz testing)

- Run AFL (8 cores, 6 hours)
- Despite the performance overhead, explore **68.3% more unique crashes** than ASan



Detection (fuzz testing)

- Number of unique crashes with time spent (metacam)



How MEDS explores more crashes?

- **More input sets can be detected**
  - **Higher probability** to detect
  - Bugs can be found **earlier** than ASan
  - Fuzzer can focus on **the other paths**

```
int a[10];
a[x] = x;
```

- **MEDS can detect the cases that ASan cannot detect**
  - Always **bypass** redzone
  - e.g., Miscalculation of structure array size
    - Size of the structure is larger than redzone size
    - Access to certain element cannot be detected.

```
struct A {
  int num[10];
};
struct A *a =
malloc(sizeof(struct A));
...
(a+i)->num[8] = i;
```

How MEDS explores more crashes?

- **More input sets can be detected**
  - **Higher probability** to detect
  - Bugs can be found **earlier** than ASan
  - Fuzzer can focus on **the other paths**

```
int a[10];
a[x] = x;
```

- **MEDS can detect the cases that ASan cannot detect**
  - Always **bypass** redzone
  - e.g., Miscalculation of structure array size
    - Size of the structure is larger than redzone size
    - Access to certain element cannot be detected.

```
struct A {
  int num[10];
};
struct A *a =
malloc(sizeof(struct A));
...
(a+i)->num[8] = i;
```

How MEDS explores more crashes?

- **More input sets can be detected**
  - **Higher probability** to detect
  - Bugs can be found **earlier** than ASan
  - Fuzzer can focus on **the other paths**

```
int a[10];
a[x] = x;
```

- **MEDS can detect the cases that ASan cannot detect**
  - Always **bypass** redzone
  - e.g., Miscalculation of structure array size
    - Size of the structure is larger than redzone size
    - Access to certain element cannot be detected.

```
struct A {
  int num[10];
};
struct A *a =
malloc(sizeof(struct A));
...
(a+i)->num[8] = i;
```

How MEDS explores more crashes?

- **More input sets can be detected**
  - **Higher probability** to detect
  - Bugs can be found **earlier** than ASan
  - Fuzzer can focus on **the other paths**

```
int a[10];
a[x] = x;
```

- **MEDS can detect the cases that ASan cannot detect**
  - Always **bypass** redzone
  - e.g., Miscalculation of structure array size
    - Size of the structure is larger than redzone size
    - Access to certain element cannot be detected.

```
struct A {
  int num[10];
};
struct A *a =
malloc(sizeof(struct A));
...
(a+i)->num[8] = i;
```

Conclusion

- **Idea**
  - Support **large gap** and **large quarantine zone**
- **Approach**
  - **Page aliasing** and **page protection**
  - **Hierarchical** memory error detection
- **Despite overhead (108%), MEDS finds more crashes during fuzz testing (68.3%)**
- **Open source – will be available soon**
  - **https://github.com/purdue-secomp-lab/MEDS**
  - Please use to detect bugs

Thank you for listening!