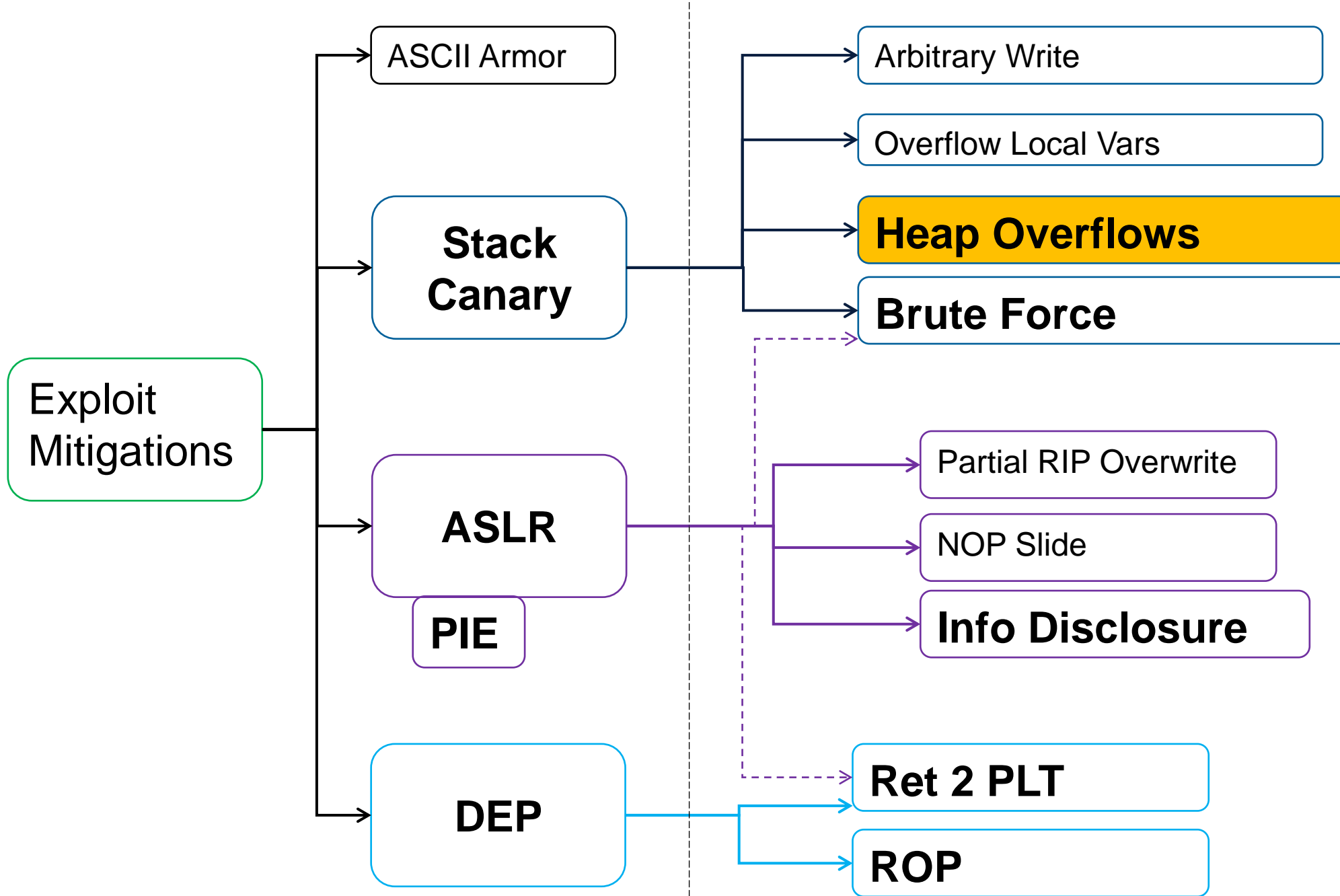# Defeat Exploit Mitigation

Heap Intro SHORT

Heap Exploitation

This slidedeck is not completely technically accurate

Should give an overview of heap exploitation concepts

Heap Introduction

What is a heap?

- malloc() allocations
- Fullfill allocating and deallocating of memory regions

Heap usage:

- Global variables (live longer than a function)
- Can be big (several kilobytes or even megabytes)

Reminder: Stack usage:

- Function-local variables
- Relatively small (usually <100 or <1000 bytes)
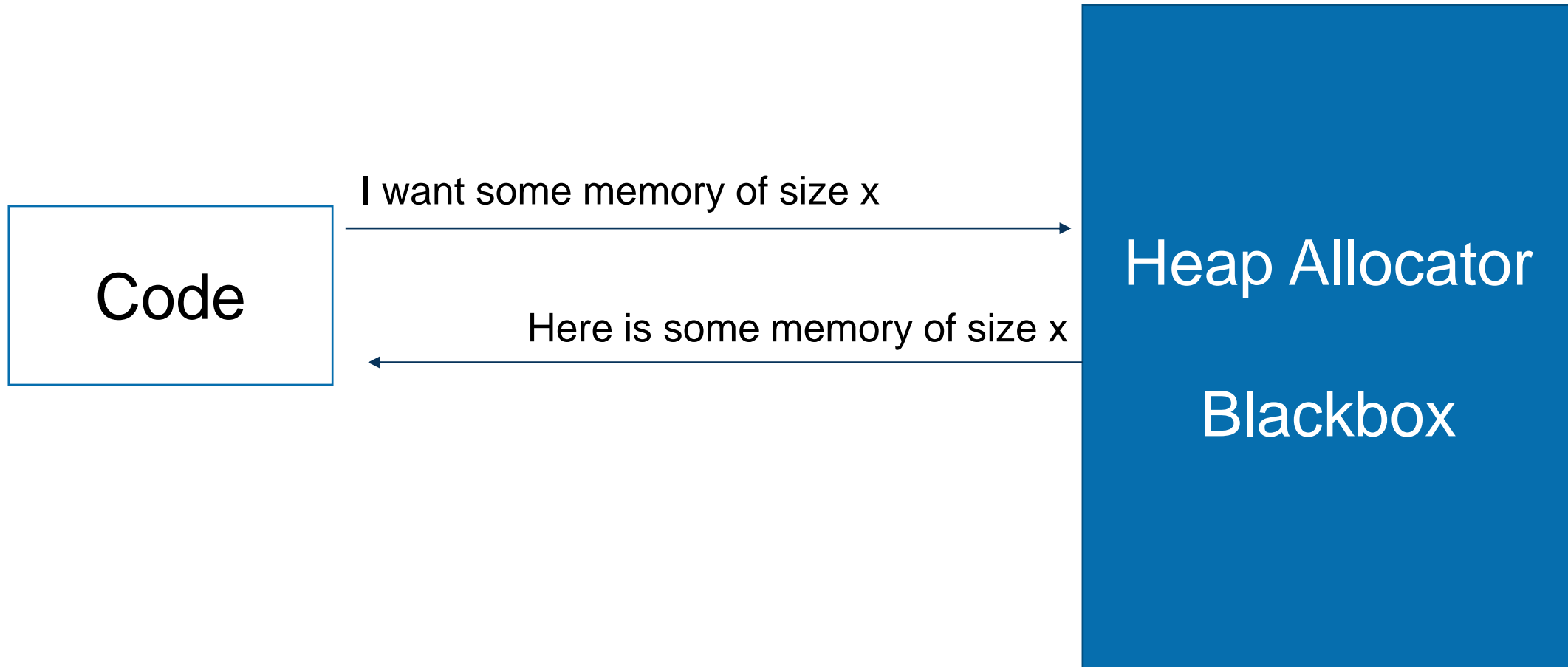
Heap Introduction

Heap:
- Dynamic memory (allocations at runtime)

- Objects, big buffers, structs, persistence, large things

- Slow, manually

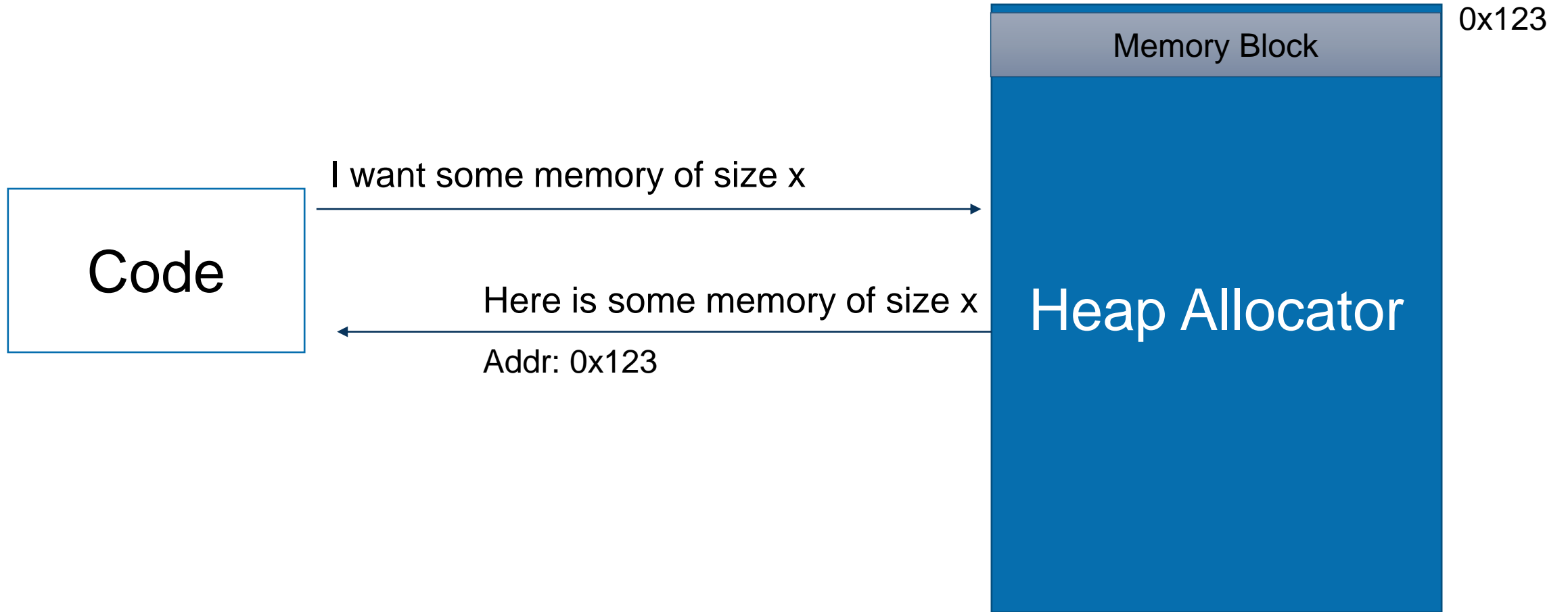Stack:
- Fixed memory allocations (known at compile time)

- Local variables, return addresses, function args

- Fast, automatic

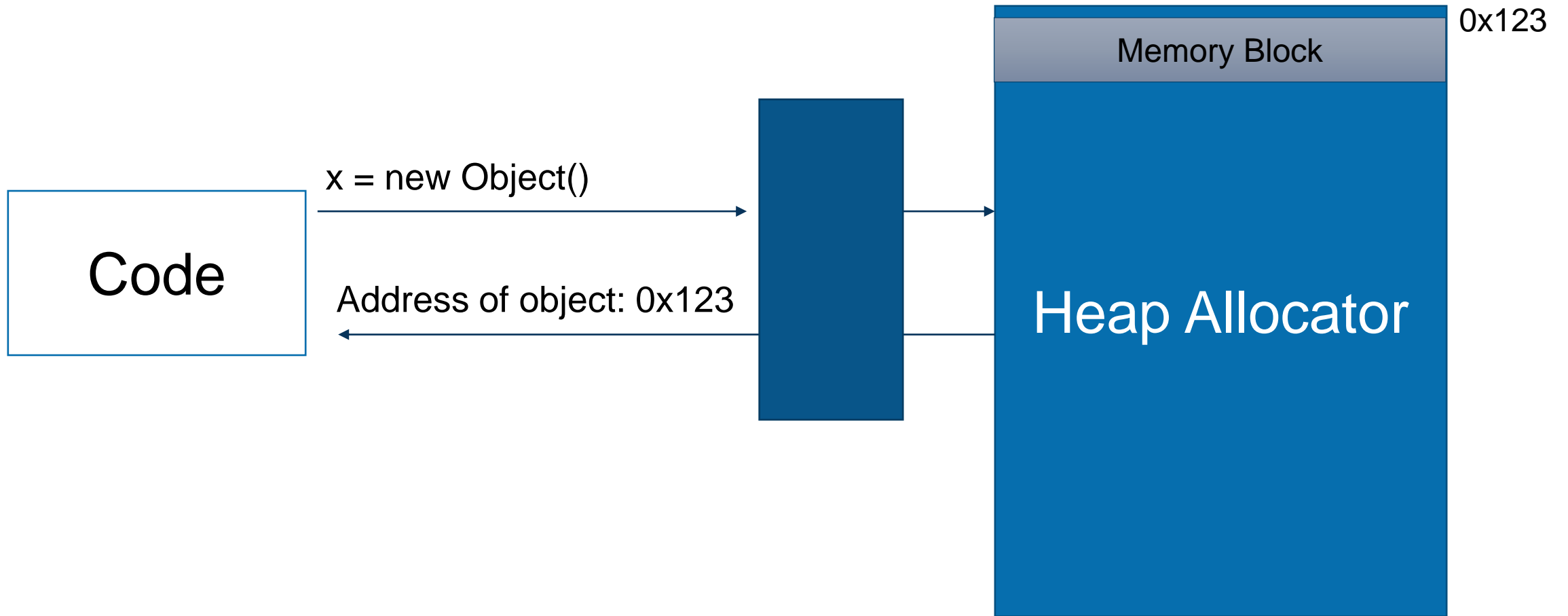# Heap Introduction

Code

I want some memory of size x

Here is some memory of size x

Heap Allocator

Blackbox

Heap Introduction

Code

Memory Block

0x123

I want some memory of size x

Here is some memory of size x

Addr: 0x123

Heap Allocator

Heap Introduction

Code

x = new Object()

Address of object: 0x123

Heap Allocator

Memory Block

0x123

Heap introduction

malloc(): Get a memory region

free(): Release a memory region

We only cover manual allocations
- Not: Automatic garbage collection
- (Garbage collection is just an automatic free() by using reference counting)

Heap Interface
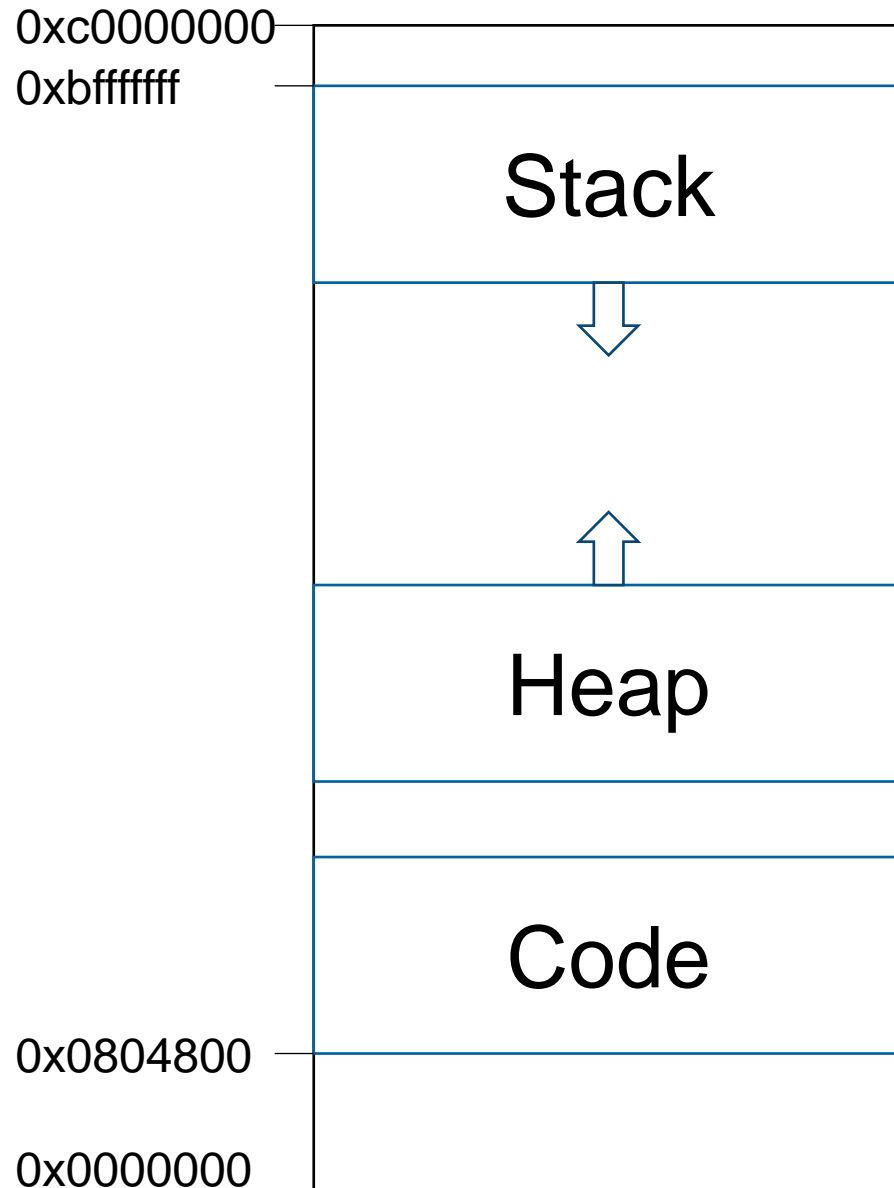
How does heap work?

void *ptr;

ptr = malloc(len)
- Allocated "len" size memory block
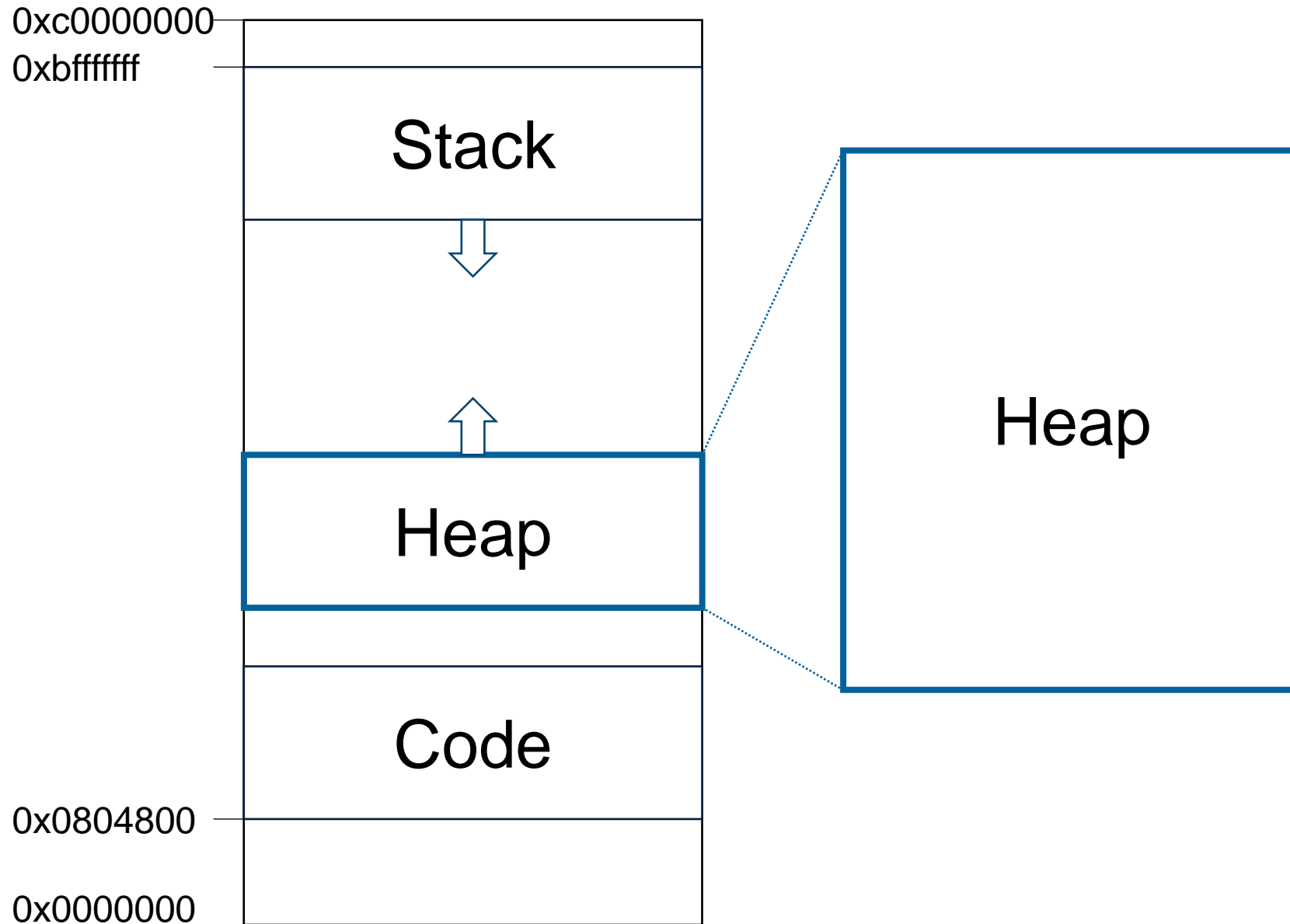- Returns a pointer to this memory block

free(ptr)
- Tells the memory allocator that the memory block can now be re-used
- Note: ptr is NOT NULL after a free()

Heap – Simplified

Heap: Memory Layout

0xc0000000

0xbfffffff

Stack

⇩

⇧

Heap

Code

0x0804800

0x0000000

Heap: Memory Layout

Heap

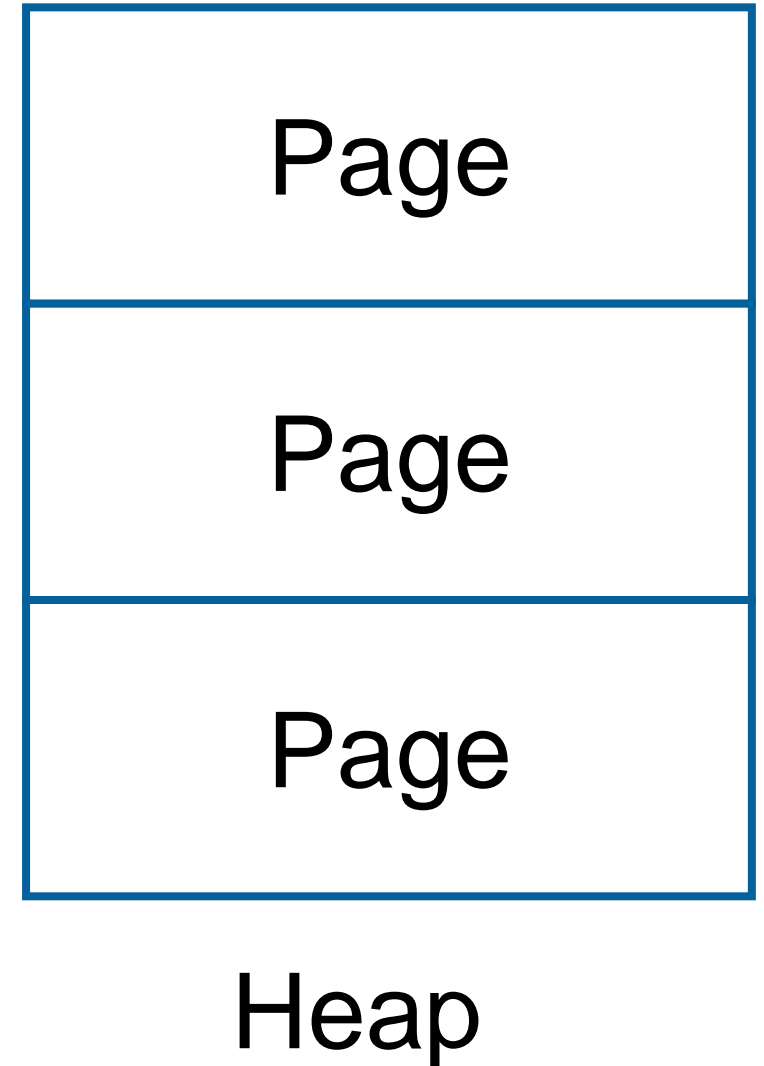What is a heap allocator doing?

- Allocate big memory **pages** from the OS

- Manage this **pages**

- Split the **pages** into smaller **bin's**

- each **bin** contain **chunks** of a specific size

- Make these **chunks** available to the program

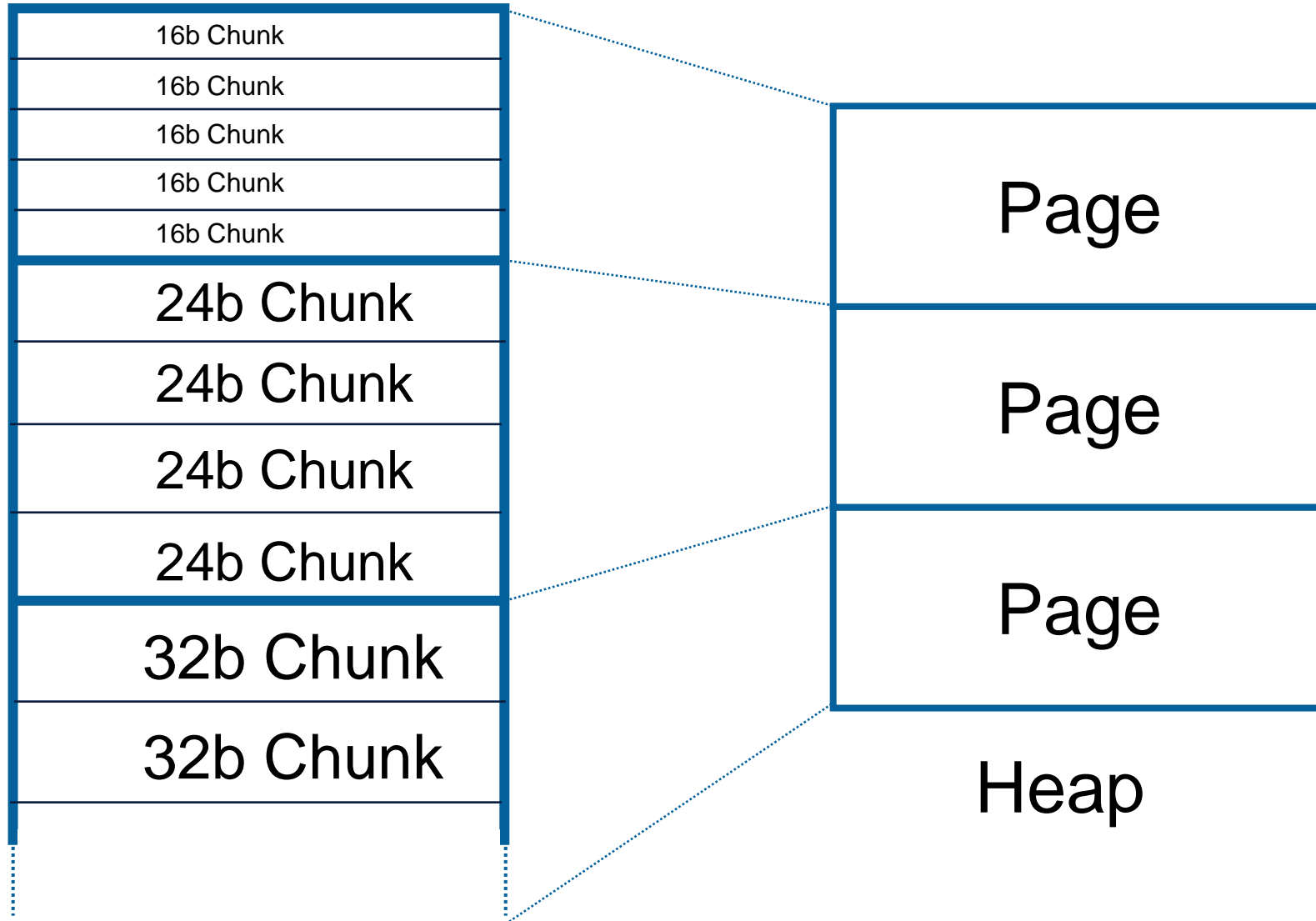# Heap: Memory Layout

Page:

- A memory page
- Usually **4k**
  - (Can also be 2 Megabytes or other, special)
- Allocated via sbrk() or mmap()
  - libc call which does a syscall to ask the OS for more "RAM"

Page
Page
Page

Heap

Heap: Memory Layout

16b Chunk
16b Chunk
16b Chunk
16b Chunk
16b Chunk

24b Chunk
24b Chunk
24b Chunk
24b Chunk

32b Chunk
32b Chunk

Page

Page

Page

Heap

Heap: Oversimplified example

Heap

16 Byte Bin

| 16b Chunk |
| 16b Chunk |
| 16b Chunk |
| 16b Chunk |
| 16b Chunk |

Page

24 Byte Bin

| 24b Chunk |
| 24b Chunk |
| 24b Chunk |
| 24b Chunk |

Page

32 Byte Bin

| 32b Chunk |
| 32b Chunk |

Page

Heap: Oversimplified example

Heap

ptr = malloc(16);

ptr

16

Page

Page

Page

Heap: Oversimplified example

Heap

Page

Page

Page

ptr2 = malloc(16);

ptr

ptr2

16

16

Heap: Oversimplified example

Heap

ptr3 = malloc(32);

ptr

ptr2

ptr3

16

16

32

Page

Page

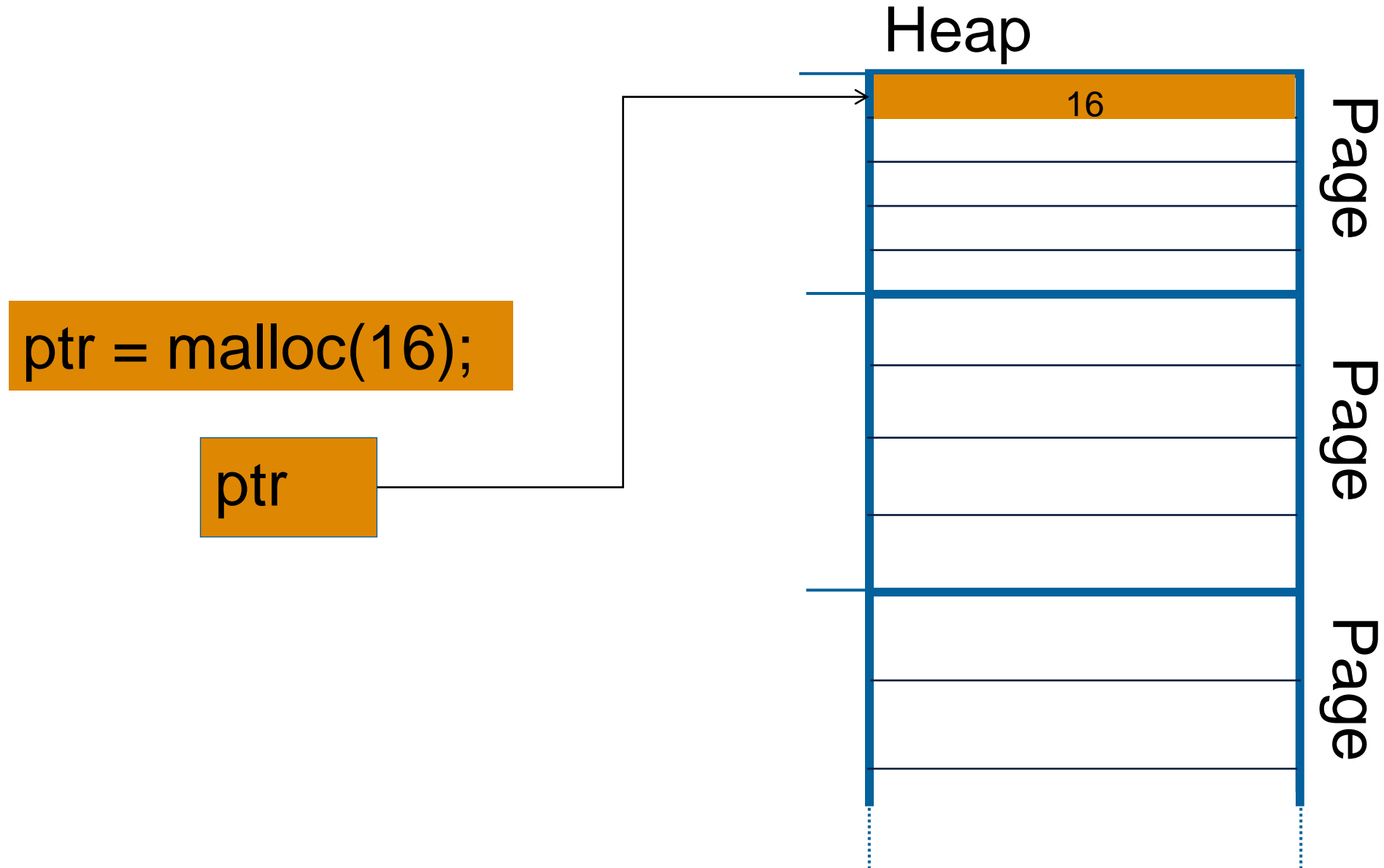Page

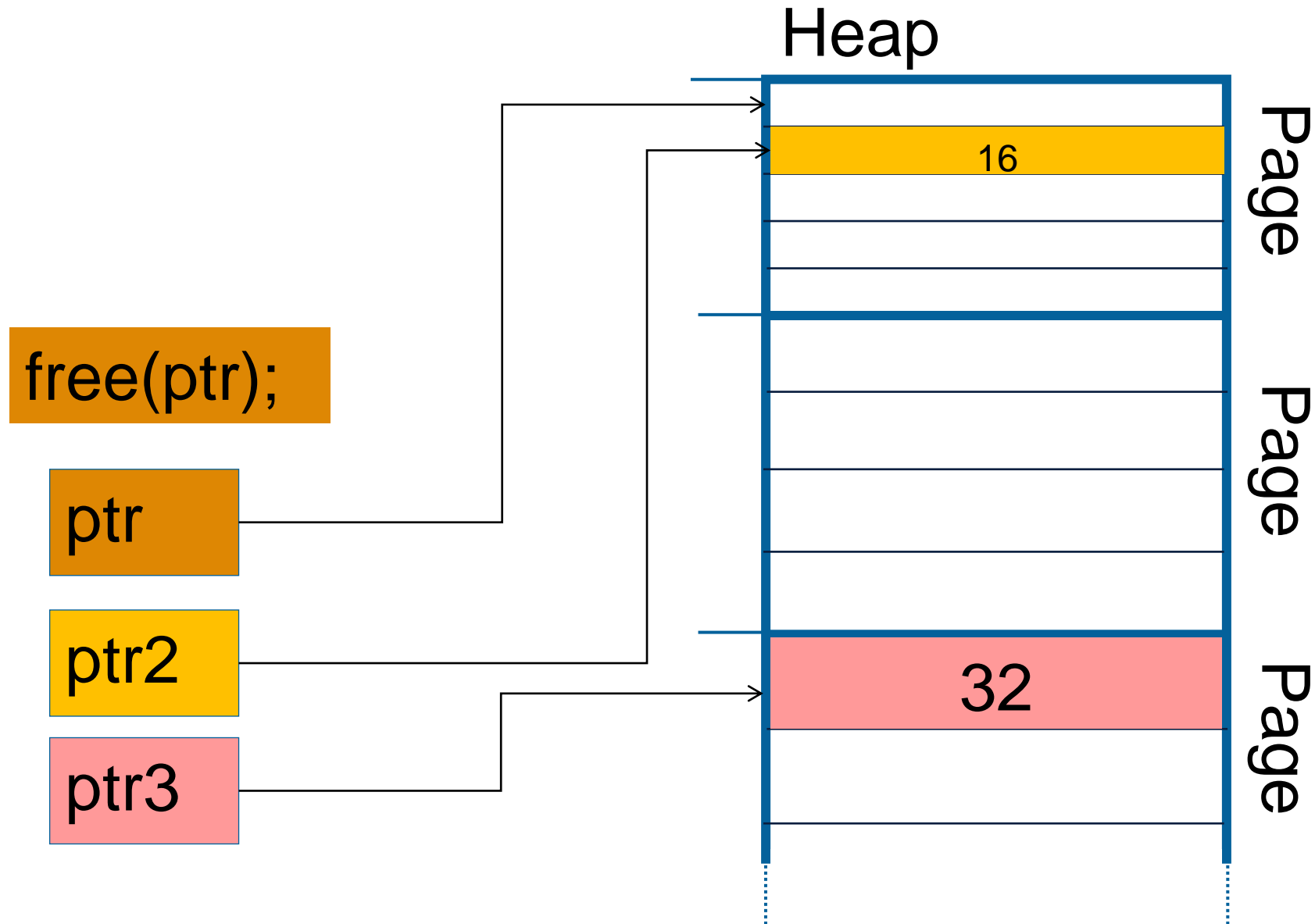# Heap: Oversimplified example

Heap: Oversimplified example

Heap Interface

Chunk →

Size of previous chunk
Size of chunk

ptr →

Space

Heap Interface

Chunk →

Size of previous chunk
Size of chunk

ptr →
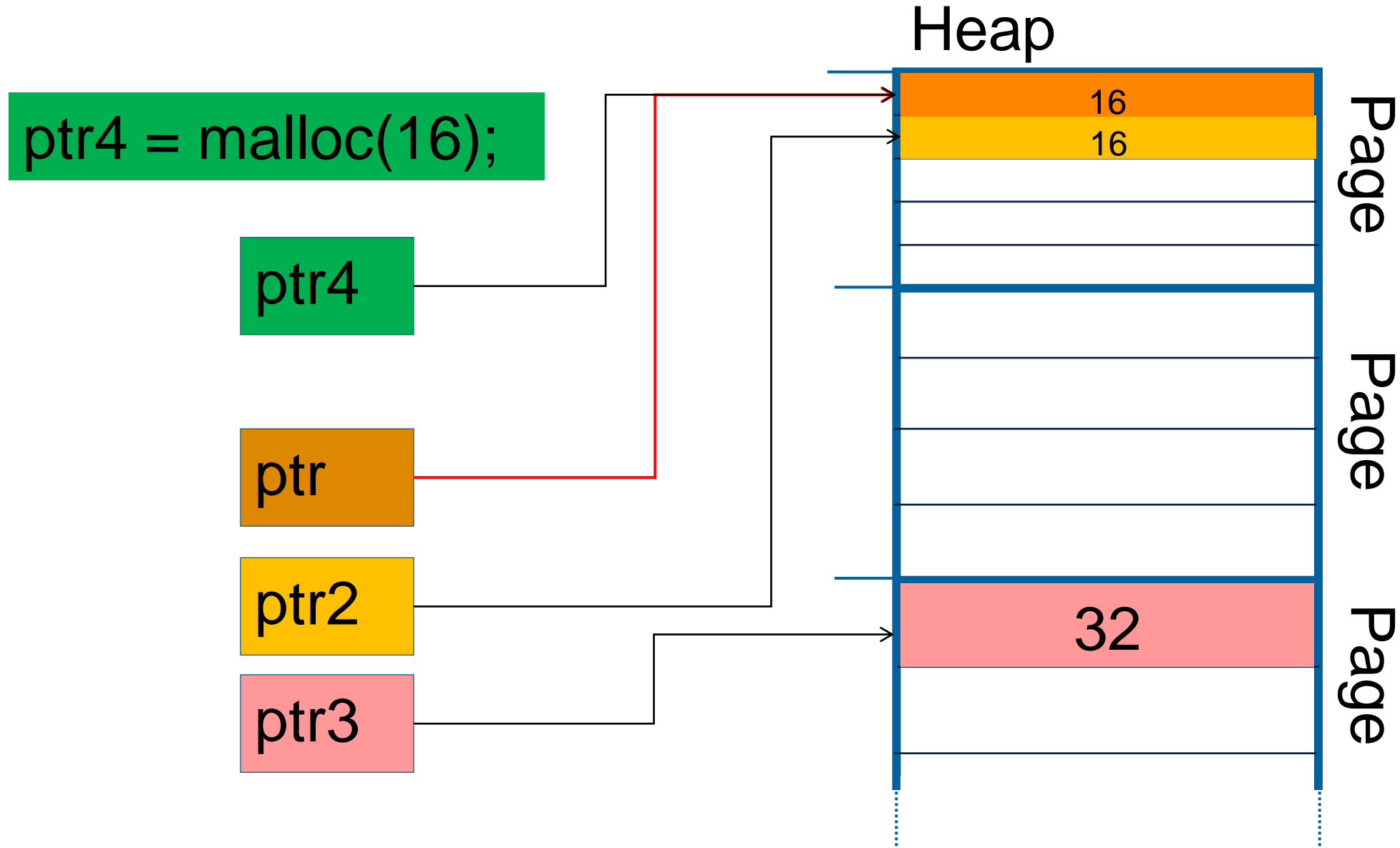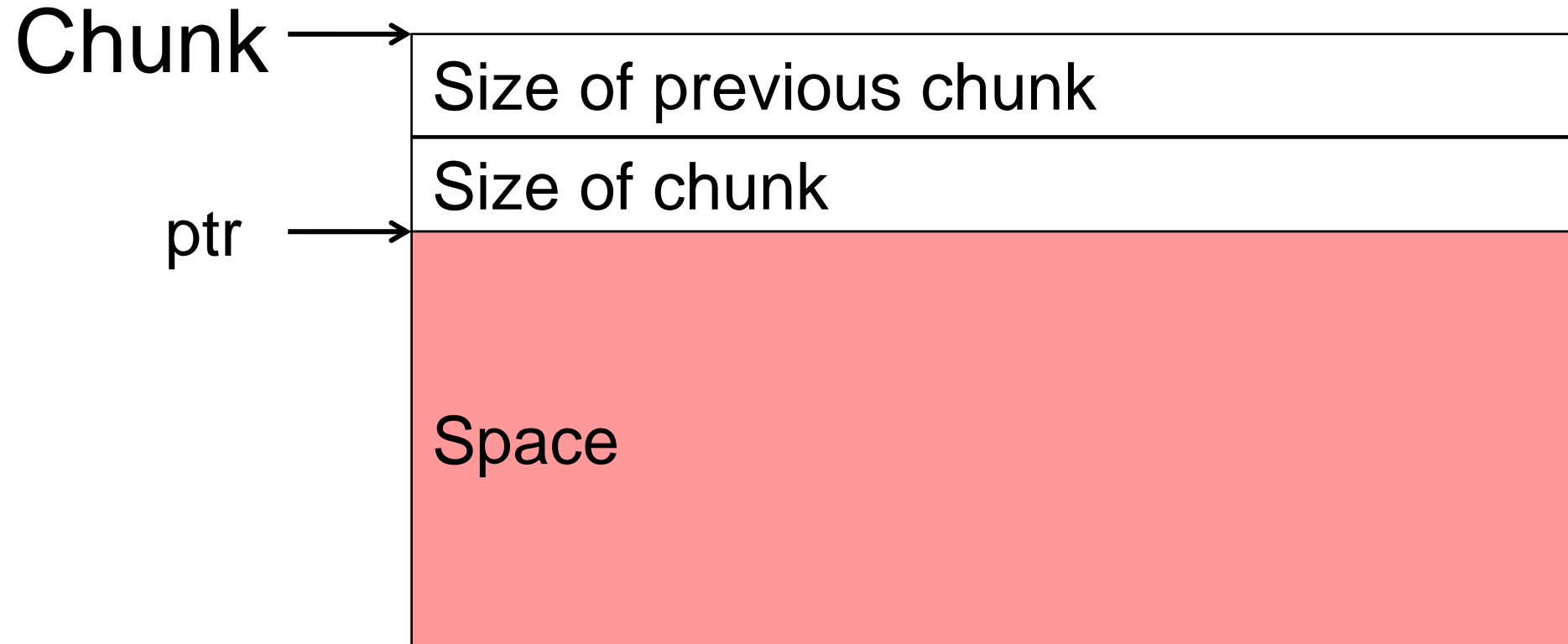
Space
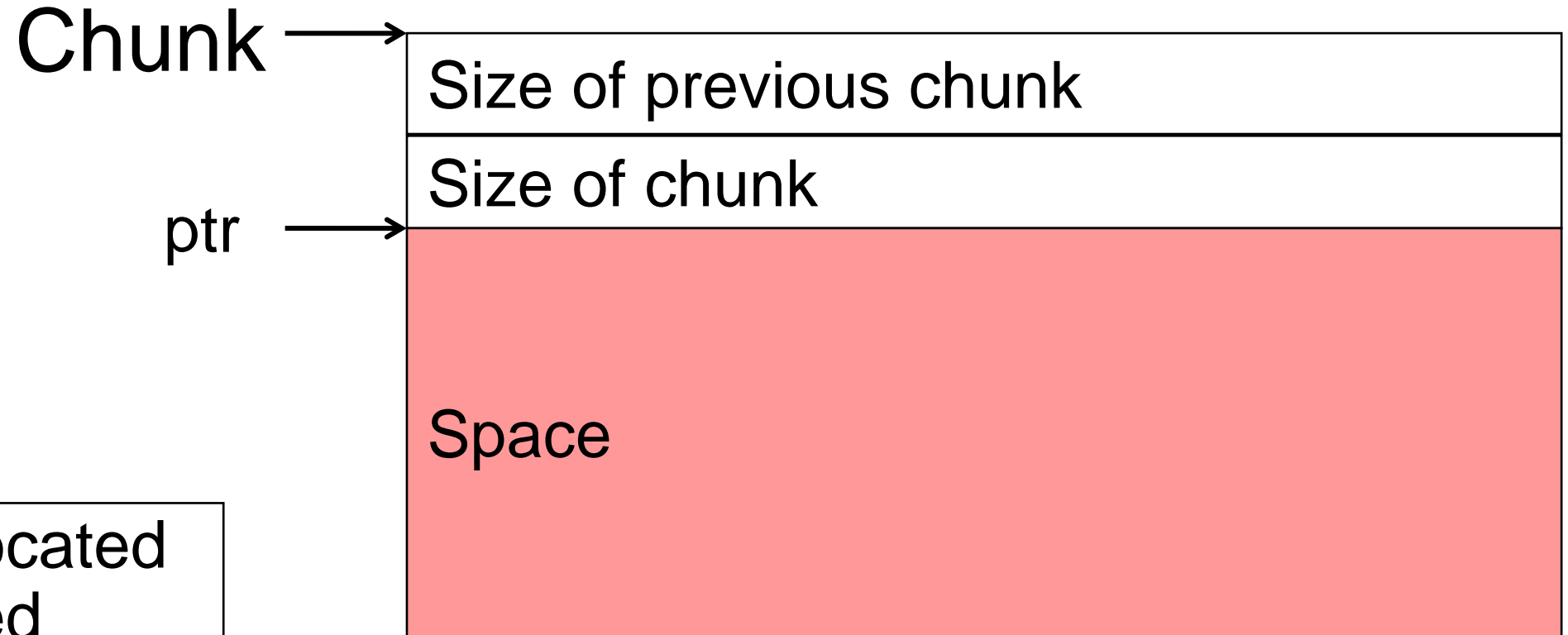
Chunk is allocated
Ptr is returned

Heap - Recap

Recap:

- Heap divides big (4k) memory **pages** into smaller **chunks**

- Heap gives these **chunks** to the program on request

- A **pointer** to a heap allocation points to the **data** part of the **chunk**

Heap attacks

Heap Attacks: Buffer overflow

Heap attack:

Inter-chunk overflow

Management chunk	Page
Chunk	
Chunk	
Chunk	
Chunk	
Management chunk	Page
Chunk	
Chunk	
Chunk	
Management chunk	Page
Chunk	
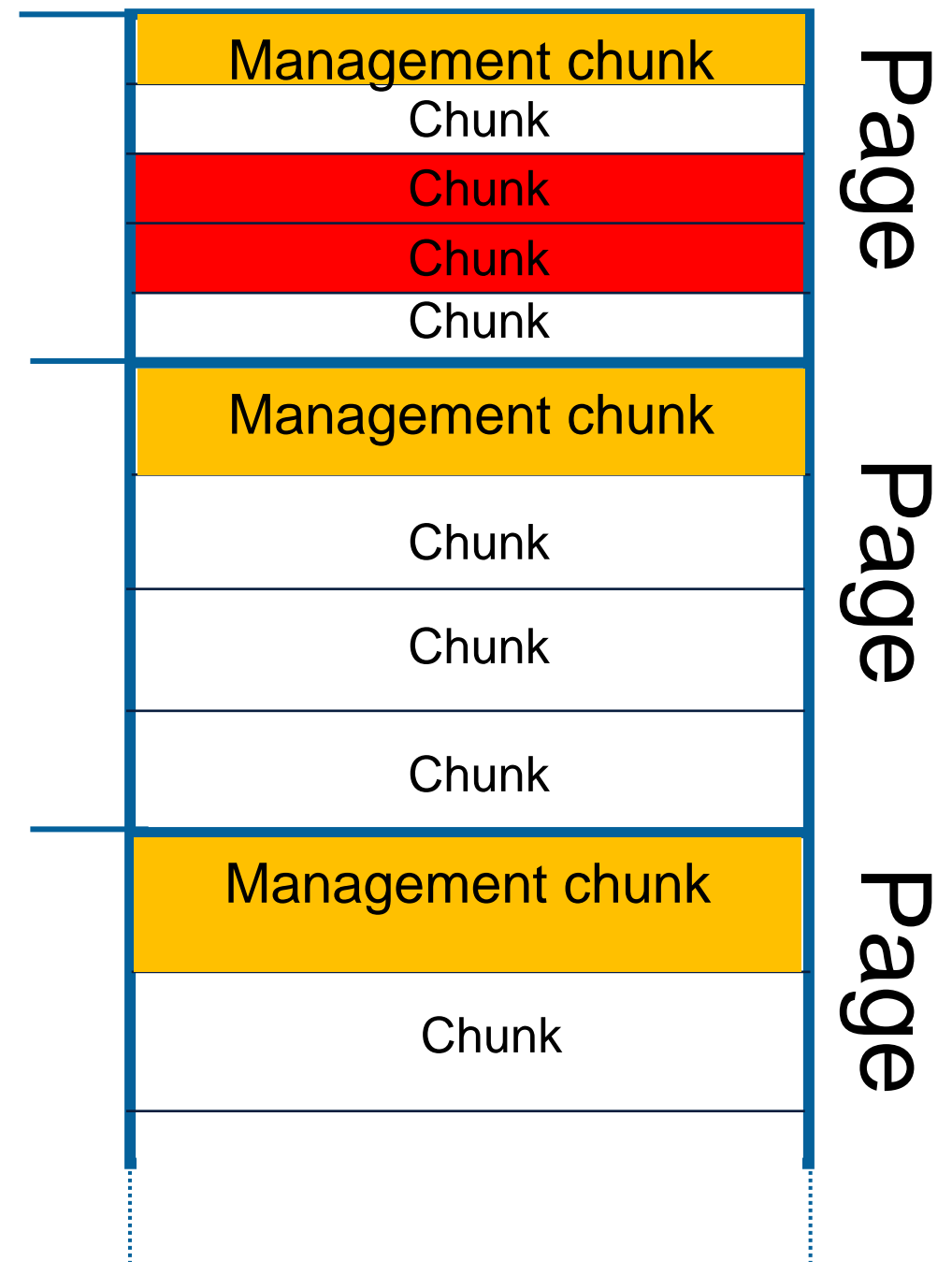
Heap Attacks: Buffer overflow

Heap attack:

Inter-chunk overflow with management chunk

Problem:
- In-band signalling (again)
- Can modify management data of heap allocator
- Therefore, can modify behaviour of heap allocator

Page
Management chunk
Chunk
Chunk
Chunk
Chunk

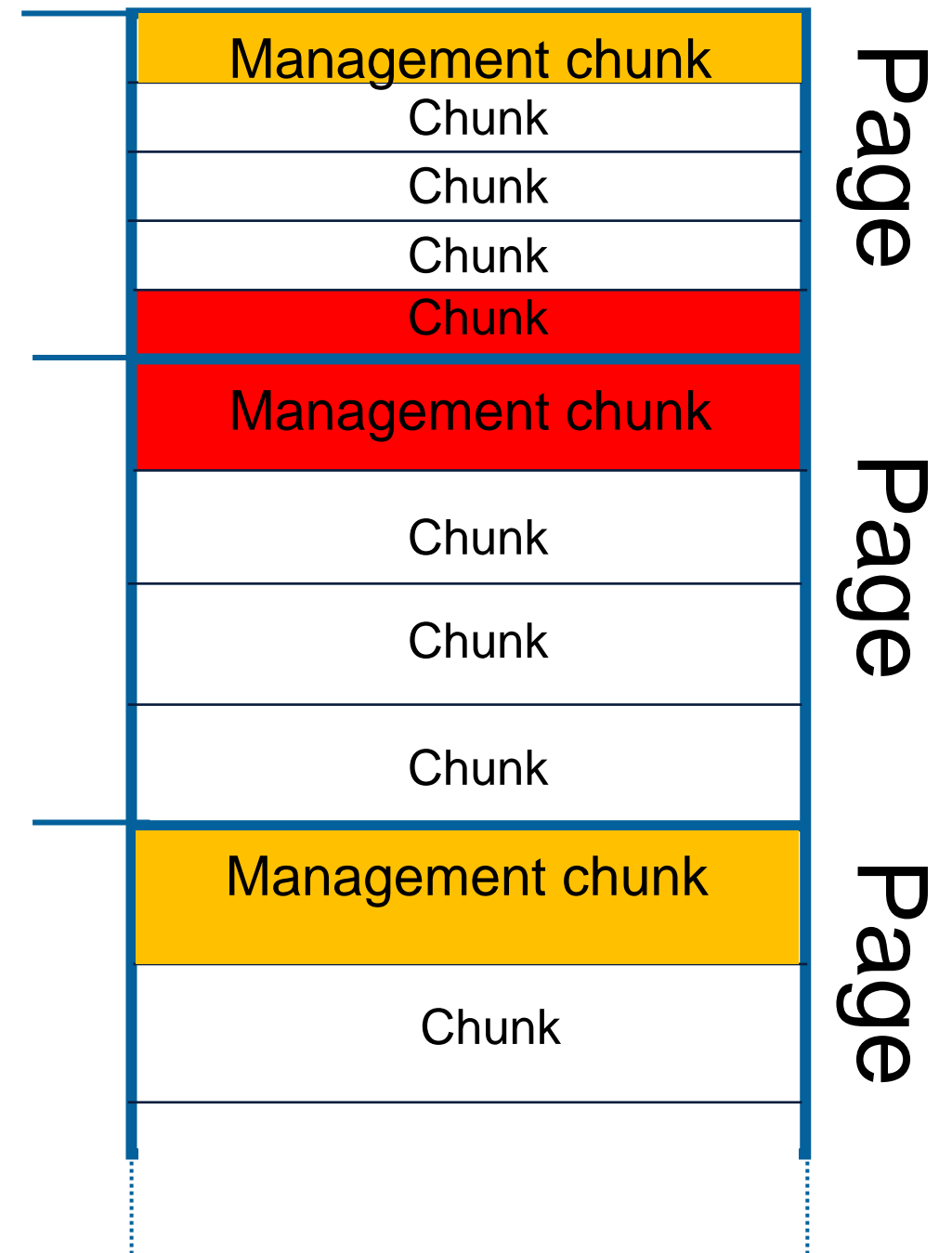| Page |
|------|
| **Management chunk** |
| Chunk |
| Chunk |
| Chunk |

Page
Management chunk
Chunk

Heap Attacks: Buffer overflow

Heap attack:

Inter-chunk overflow with chunk metadata

Problem:

- In-band signalling (again)
- Can modify management data of heap allocator
- Therefore, can modify behaviour of heap allocator
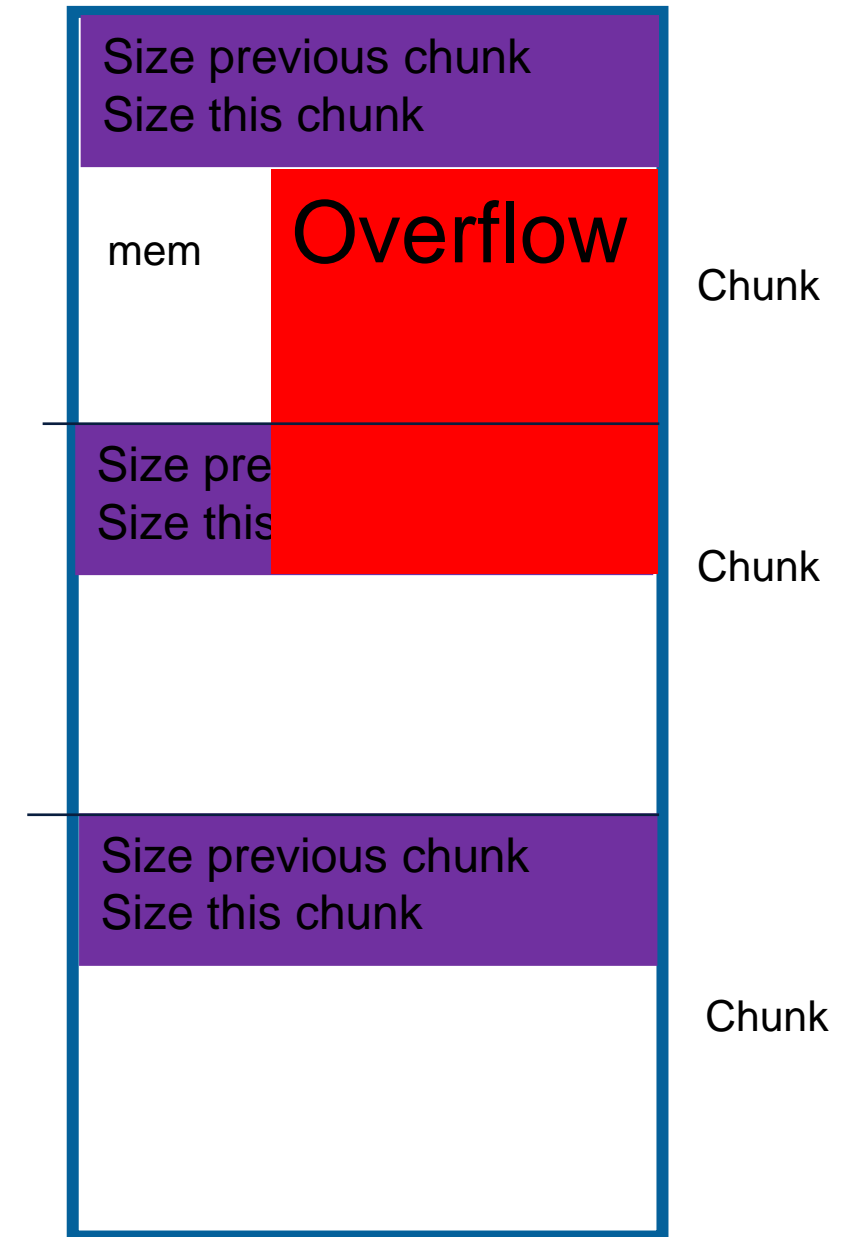 - Create fake chunks
  - Ptmalloc2: Write what where upon free

Size previous chunk
Size this chunk

mem

Overflow

Chunk

Size pre
Size this

Chunk

Size previous chunk
Size this chunk

Chunk

Heap Attacks: Use after free (UAF)

err

usr

Heap

| 16 |
| 16 |

Page

Page

Page
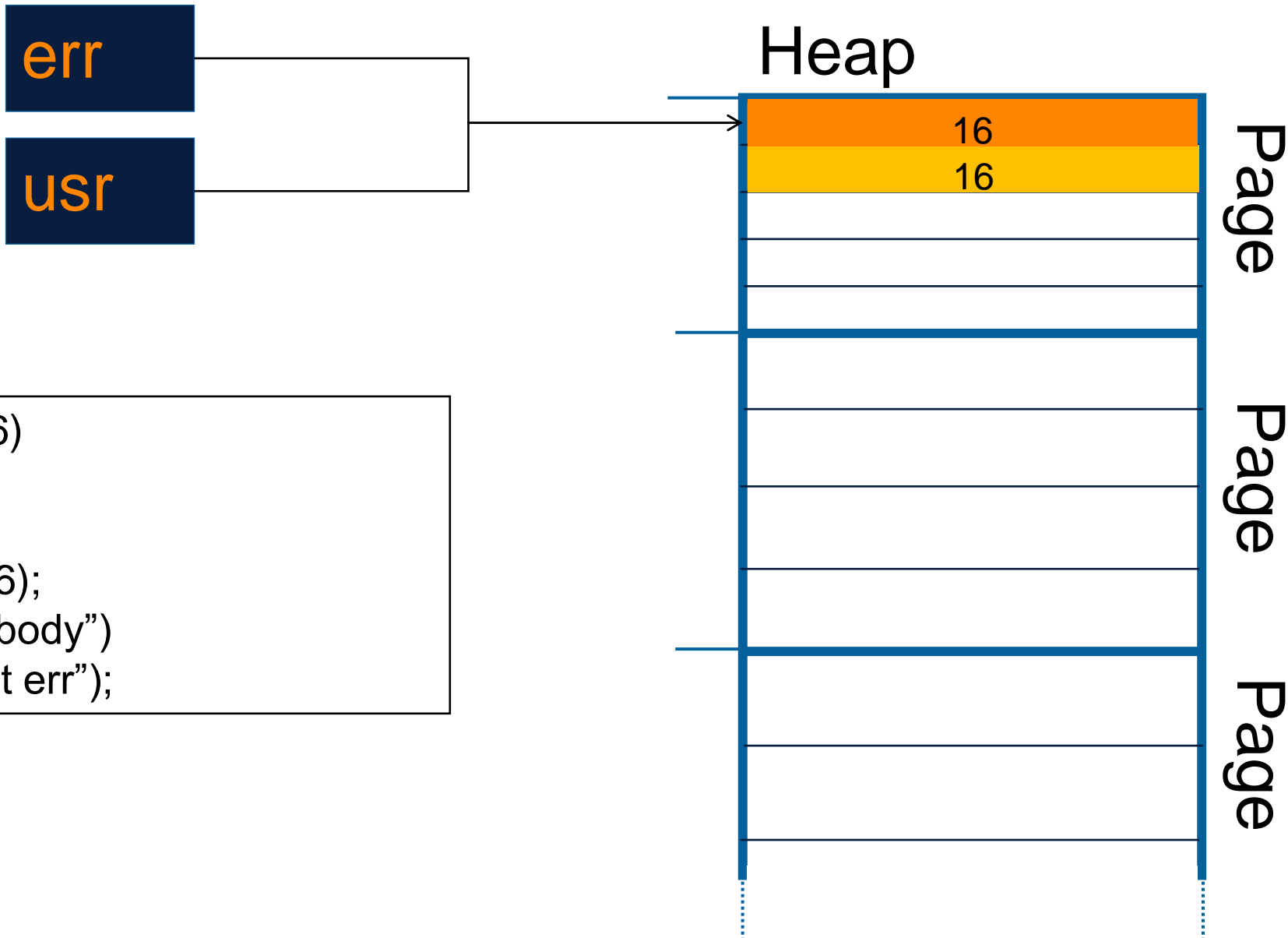
err = malloc(16)
free(err)

usr = malloc(16);
strcpy(usr, "nobody")
strcpy(err, "root err");

Heap Attacks:
Use After Free (UAF)

Heap Attack: UAF

UAF:

Use after free

Or more correctly:

Use an object, after the memory it has been pointing to has been freed, and now a different object is stored at that location

Object Oriented Languages

Dobin: *"OO ist just some fancy C structs with function pointers"*

OO in C:

```c
typedef struct animal {
        int (*constructor)(void *self);
        int (*write)(void *self, void *buff);
        void *data;
} AnimalClass;

AnimalClass animal;
animal.constructor = &constructor;
animal.data = malloc(…);
…
animal.constructor(&animal);
```

Garbage Collection

Dobin: *"Garbage collection is just fancy structs with reference counter"*

```
typedef struct animal {
        int (*constructor)(void *self);
        int (*write)(void *self, void *buff);
        void *data;
        int refCount;
} AnimalClass;


AnimalClass animal;
animal.refCount = 0;
…
Animal animal2 = &animal;
Animal.refCount++;
```

Use After Free

```
Class Animal {
    int hungry;
    int tired;

    void eat() {};
    void walk() {};
}
```

```
Class Car {
    int doors;
    int locked;
    int fuel;

    void drive() {};
}
```

Animal Class	Animal Object
int hungry	0
int tired	5
func eat()	0x123
func walk()	0x512

Car Class	Car Object
int doors	4
int locked	0
int fuel	10
func drive()	0xabc

Use After Free

animal

Animal

```
animal = new Animal()
if error:
  free(animal)

car = new Car()
car.drive(100000000)

animal.eat()
```

Animal Class	Animal Object
int hungry	0
int tired	5
func eat()	0x123
func walk()	0x512

Car Class	Car Object
int doors	4
int locked	0
int fuel	10
func drive()	0xabc

Use After Free

animal

Animal

```
animal = new Animal()
if error:
    free(animal)

car = new Car()
car.drive(100000000)

animal.eat()
```

Animal Class	Animal Object
int hungry	0
int tired	5
func eat()	0x123
func walk()	0x512

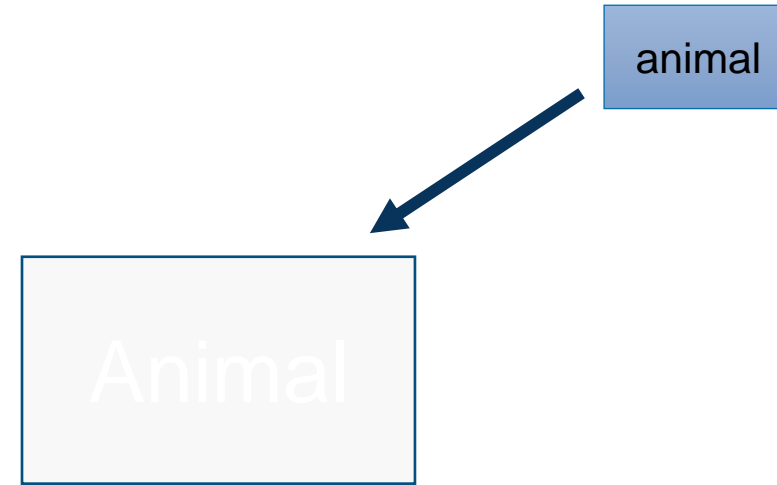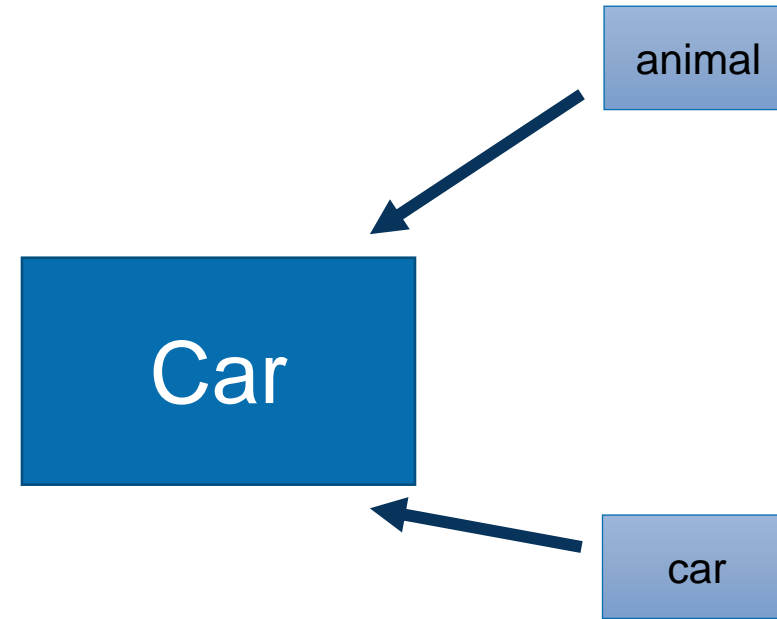| Car Class	Car Object
int doors	4
int locked	0
int fuel	10
func drive()	0xabc

Use After Free

```
animal = new Animal()
if error:
    free(animal)

car = new Car()
car.drive(100000000)

animal.eat()
```

animal

Car

car

Animal Class	Animal Object
int hungry	0
int tired	5
func eat()	0x123
func walk()	0x512

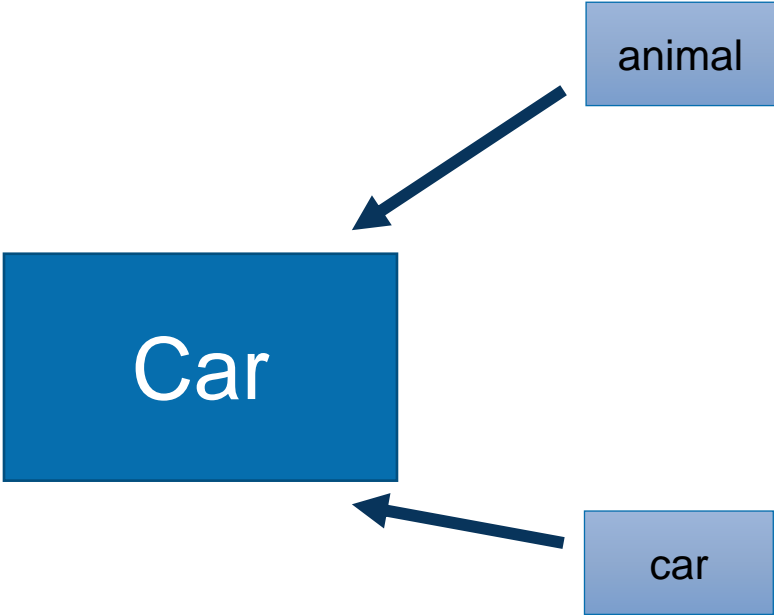| Car Class	Car Object
int doors	4
int locked	0
int fuel	10
func drive()	0xabc

Use After Free

```
animal = new Animal()
if error:
    free(animal)

car = new Car()
car.drive(100000000)

animal.eat()
```

```
Car::drive(distance):
    car.fuel -= distance
```

Car

animal

car

Animal Class	Animal Object
int hungry	0
int tired	5
func eat()	0x123
func walk()	0x512

Car Class	Car Object
int doors	4
int locked	0
int fuel	10
func drive()	0xabc

Use After Free

animal

Car

car

```
animal = new Animal()
if error:
    free(animal)

car = new Car()
car.drive(100000000)

animal.eat()
```

```
Car::drive(distance):
    car.fuel -= distance
```

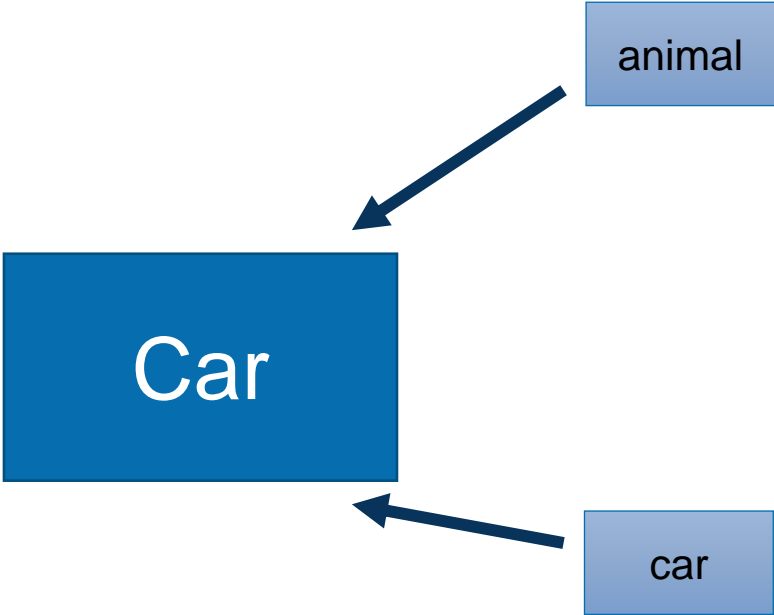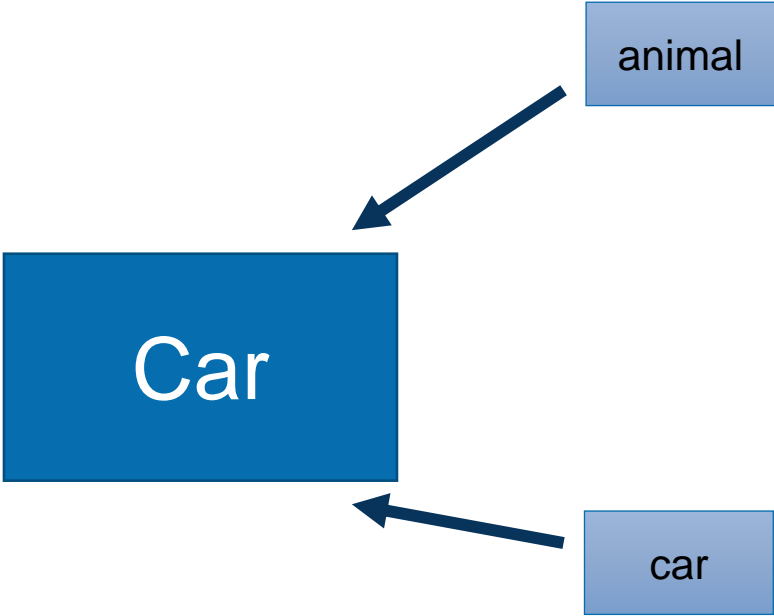| Animal Class	Animal Object		Car Class	Car Object
int hungry	0		int doors	4
int tired	5		int locked	0
func eat()	0xEFFFA285A		int fuel	-10000000
func walk()	0x512		func drive()	0xabc

Use After Free


```
animal = new Animal()
if error:
    free(animal)

car = new Car()
car.drive(100000000)
```

animal.eat()

```
Car::drive(distance):
    car.fuel -= distance
```

Animal Class	Animal Object	Car Class	Car Object
int hungry	0	int doors	4
int tired	5	int locked	0
func eat()	0xEFFFA285A	int fuel	-10000000
func walk()	0x512	func drive()	0xabc

Heap Attack: UAF

So, what is UAF?

- We have a pointer (of type A) to an **object**
- The **object** get's **free**()'d
  - This means that the memory allocater marks the object as free
  - The object will not be modified! (Similar to deleting a file on the harddisk)
  - The pointer is still valid
- **Another object** of type B (of similar size) get's **allocated**
- Memory allocator returns the previously free'd object memory space

- Attacker has now **a pointer (type A) to another object (type B)**!
- This object can be modified
  - Depending on the types A and B

Challenge 33

Challenge 33 UaF

Mongoose Web Server

- /login

- /logout

- /ping?asdf

Challenge 33 UaF

```
// An authenticated user
struct t_authenticated {
    int role;
    char sessionid[128];
    void (*logout_handler)();
};

// Only supports 1 authenticated user for now
struct t_authenticated *Authenticated;
```

+0	int role
+8	char sessionid[128]
+136	(*logout_handler)()

Challenge 33 UaF

Mongoose Web Server

- /login
  - Authenticated = malloc(sizeof(t_authenticated));

- /logout
  - free(Authenticated);

- /ping?asdf
  - malloc(sizeof(URL_QUERY));
  - free();

Challenge 33 UaF

```
// REST: /login
void rest_login(struct mg_connection *c, struct http_message *hm) {
    char response[RESPONSE_LEN];

    // Malloc and init for Authenticated struct
    Authenticated = malloc(sizeof(struct t_authenticated));
    Authenticated->logout_handler = &logout_handler;
    Authenticated->role = 23;
    strcpy(Authenticated->sessionid, "5"); // chosen by a fair dice roll

    strncpy(response, Authenticated->sessionid, 8);
    mg_send_head(c, 200, RESPONSE_LEN, "Content-Type: text/plain");
    mg_send(c, response, RESPONSE_LEN);
}
```

Challenge 33 UaF

```c
// REST: /logout
void rest_logout(struct mg_connection *c, struct http_message *hm) {
    char response[512];

    // Send answer
    sprintf(response, "Logout %i\r\n", Authenticated->role);
    mg_send_head(c, 200, strlen(response), "Content-Type: text/plain");
    mg_printf(c, "%s", response);

    // Cleanup
    (*Authenticated->logout_handler)();
    free(Authenticated);
}
```

Challenge 33 UaF

```c
// REST: /ping
void rest_ping(struct mg_connection *c, struct http_message *hm) {
    // Answer - copy query_string and send it back
    int len = (int)hm->query_string.len;
    void* qs = malloc(len);
    memcpy(qs, hm->query_string.p, hm->query_string.len);

    mg_send_head(c, 200, len, "Content-Type: text/plain");
    mg_send(c, qs, len);

    free(qs);
}
```

Challenge 33 UaF

```
curl localhost:8000/login
// Authenticated = malloc(sizeof(t_authenticated));
```
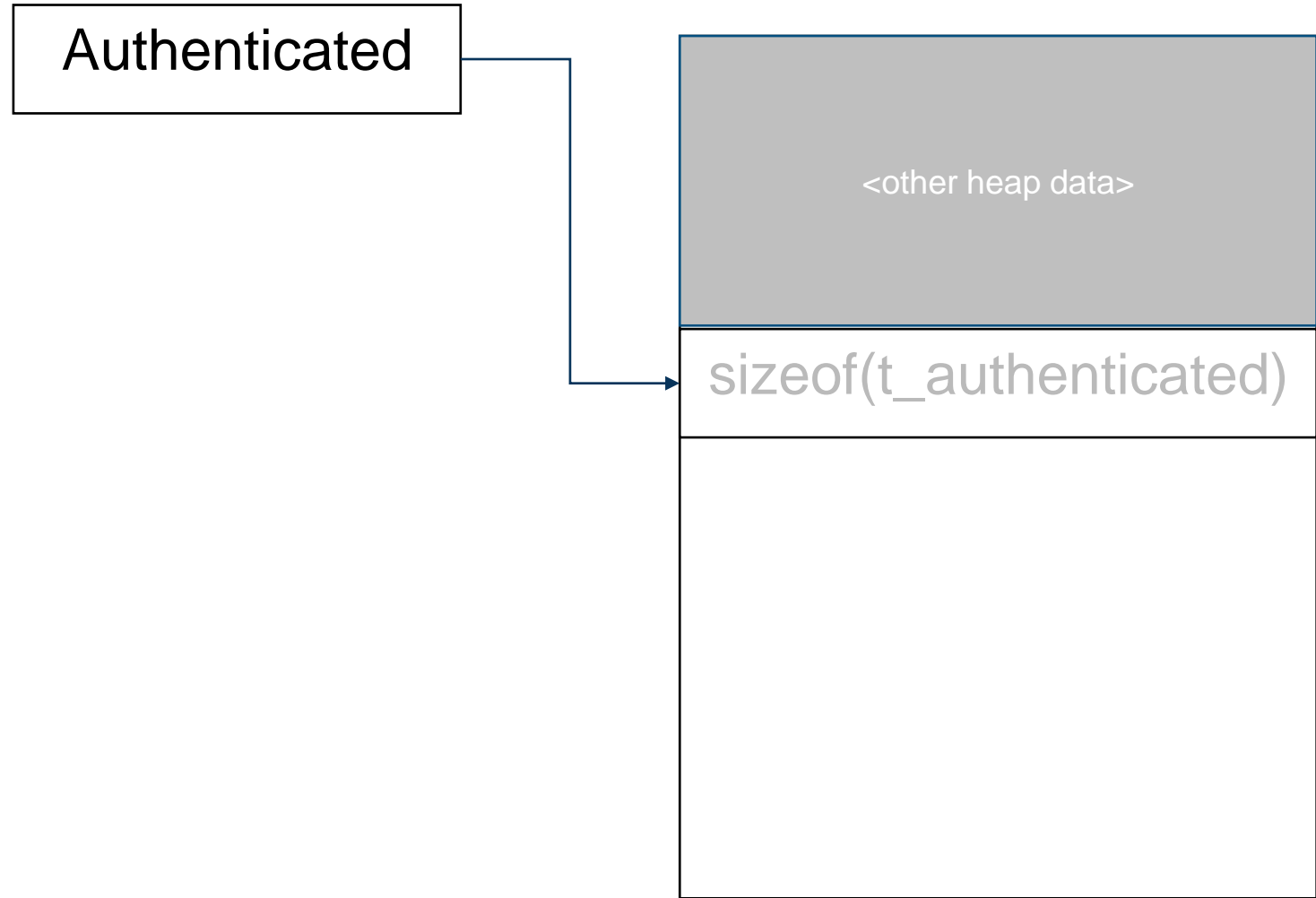
Authenticated

<other heap data>

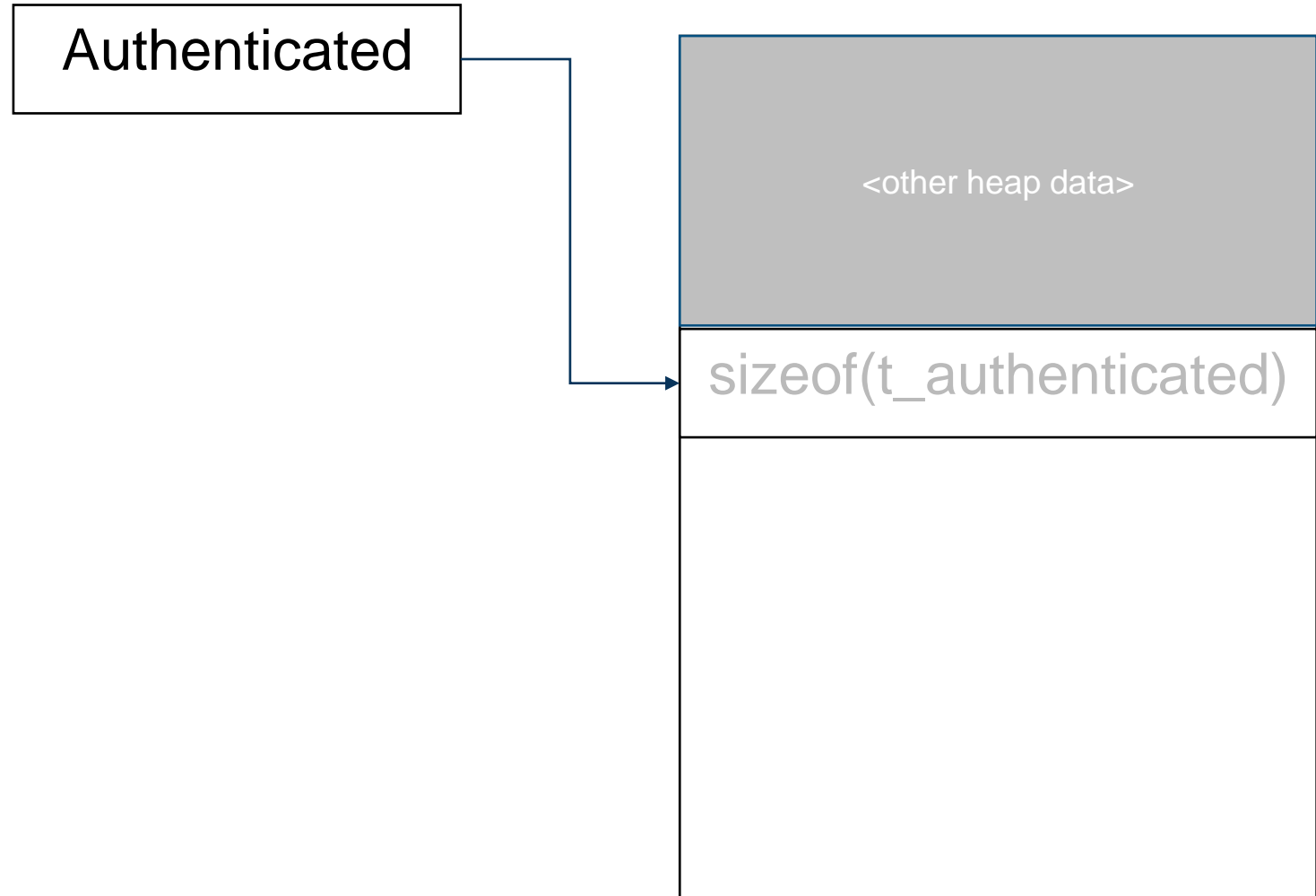sizeof(t_authenticated)

Challenge 33 UaF

```
curl localhost:8000/login
curl localhost:8000/logout
// Authenticated->function();
// free(Authenticated);
```

Authenticated

<other heap data>

sizeof(t_authenticated)

Challenge 33 UaF

```
curl localhost:8000/login
curl localhost:8000/logout
curl localhost:8000/logout
// Authenticated->function();
// free(Authenticated);
```

Authenticated

<other heap data>

sizeof(t_authenticated)

Challenge 33 UaF

Authenticated

```
curl localhost:8000/login
curl localhost:8000/logout
curl localhost:8000/ping?aaaaaaa...
// malloc(strlen("aaaa..."));
```

<other heap data>
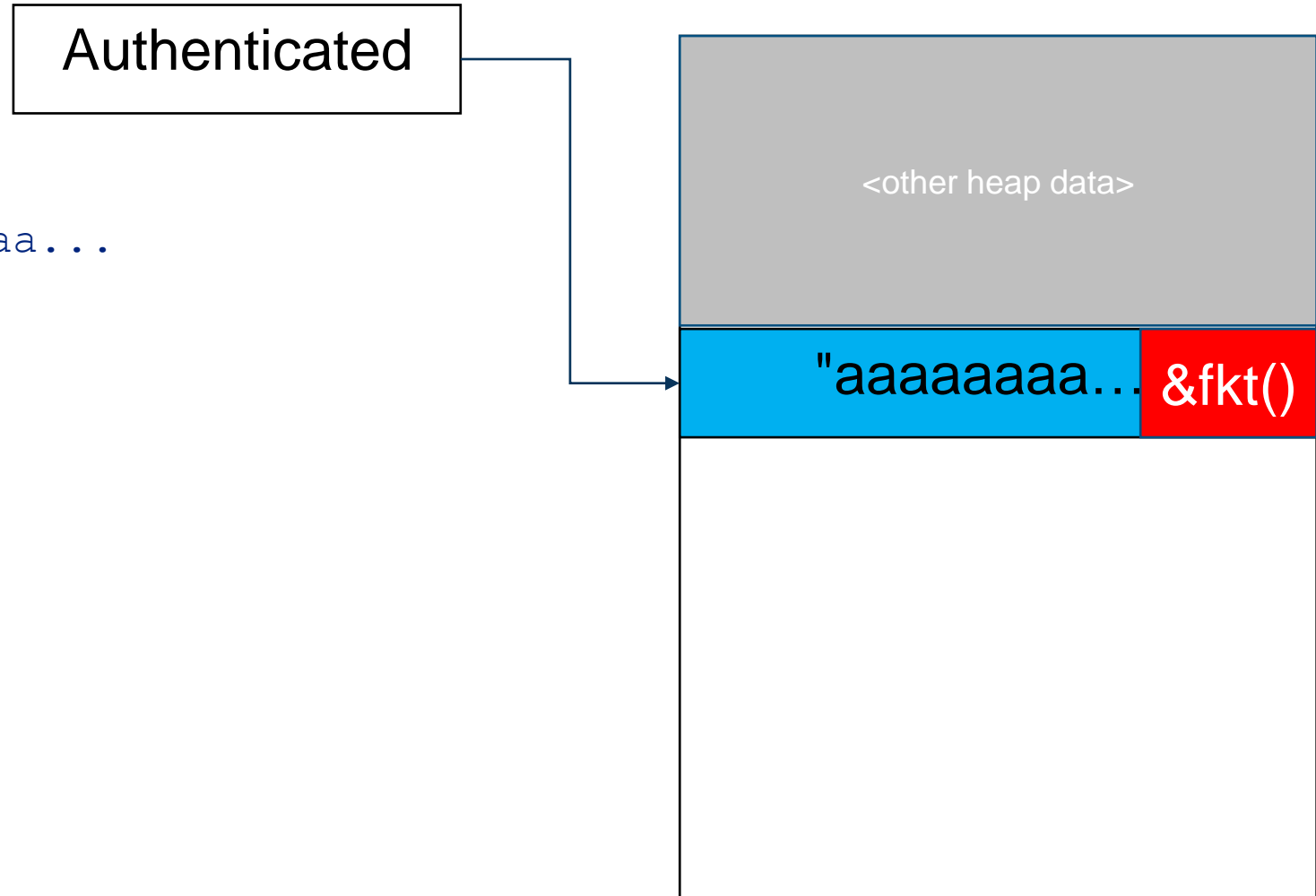
"aaaaaaa…."

Challenge 33 UaF

Authenticated

```
curl localhost:8000/login
curl localhost:8000/logout
curl localhost:8000/ping?aaaaaaa...
curl localhost:8000/logout
// Authenticated->function();
// free(Authenticated);
```

<other heap data>

"aaaaaaa... &fkt()

Conclusion

UaF:
Step 1: make two objects, with different types, point to the same object on the heap
Step 2: ???
Step 3: Profit!

Heap Attacks:
Use After Free (UAF)

Intermezzo

Use After Free

WebKit

Available for: iPhone 5 and later, iPad 4th generation and later, iPod touch 6th generation and later

Impact: Processing maliciously crafted web content may lead to arbitrary code execution

Description: A use after free issue was addressed through improved memory management.

CVE-2017-2471: Ivan Fratric of Google Project Zero

Kernel

Available for: iPhone 5 and later, iPad 4th generation and later, iPod touch 6th generation and later

Impact: An application may be able to execute arbitrary code with kernel privileges

Description: A use after free issue was addressed through improved memory management.

libc++abi

Available for: iPhone 5 and later, iPad 4th generation and later, iPod touch 6th generation and later

Impact: Demangling a malicious C++ application may lead to arbitrary code execution

Description: A use after free issue was addressed through improved memory management.

CVE-2017-2441

Use After Free

Fixed in Firefox 48

2016-84	Information disclosure through Resource Timing API during page navigation
2016-83	Spoofing attack through text injection into internal error pages
2016-82	Addressbar spoofing with right-to-left characters on Firefox for Android
2016-81	Information disclosure and local file manipulation through drag and drop
2016-80	Same-origin policy violation using local HTML file and saved shortcut file
2016-79	Use-after-free when applying SVG effects
2016-78	Type confusion in display transformation
2016-77	Buffer overflow in ClearKey Content Decryption Module (CDM) during video playback
2016-76	Scripts on marquee tag can execute in sandboxed iframes
2016-75	Integer overflow in WebSockets during data buffering
2016-74	Form input type change from password to text can store plain text password in session restore file
2016-73	Use-after-free in service workers with nested sync events
2016-72	Use-after-free in DTLS during WebRTC session shutdown
2016-71	Crash in incremental garbage collection in JavaScript
2016-70	Use-after-free when using alt key and toplevel menus
2016-69	Arbitrary file manipulation by local user through Mozilla updater and callback

use-after-free   1 of 76
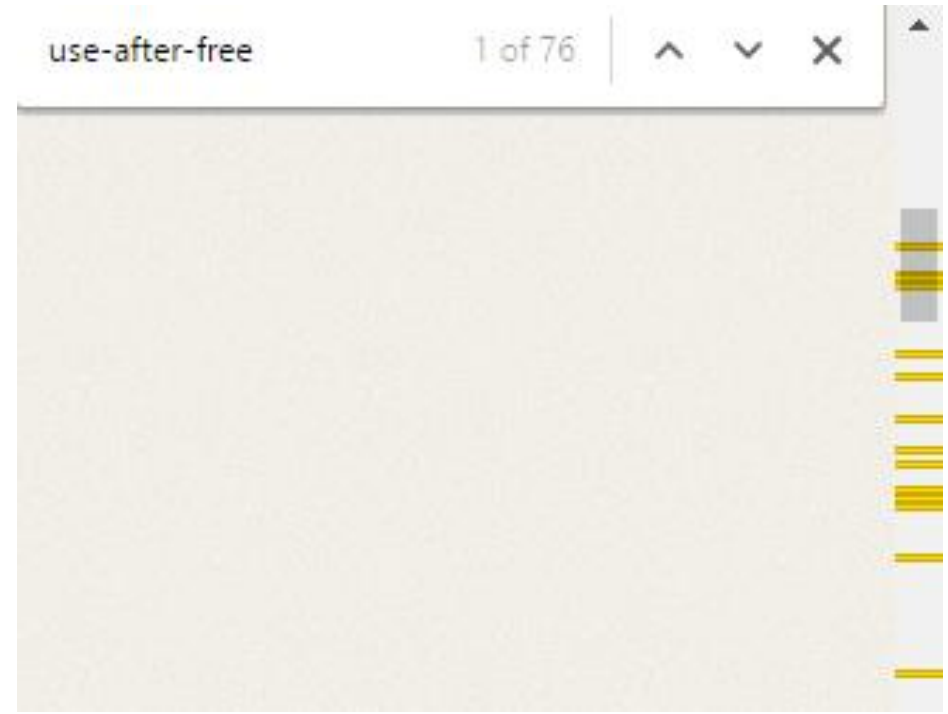
Use after free

Security Fixes and Rewards

Note: Access to bug details and links may be kept restricted until a majority of users are updated with a fix. We will also retain restrictions if the bug exists in a third party library that other projects similarly depend on, but haven't yet fixed.

This update includes 36 security fixes. Below, we highlight fixes that were contributed by external researchers. Please see the Chrome Security Page for more information.

[$7500][682194] **High** CVE-2017-5030: Memory corruption in V8. *Credit to Brendon Tiszka*
[$5000][682020] **High** CVE-2017-5031: Use after free in ANGLE. *Credit to Looben Yang*
[$3000][668724] **High** CVE-2017-5032: Out of bounds write in PDFium. *Credit to Ashfaq Ansari - Project Srishti*
[$3000][676623] **High** CVE-2017-5029: Integer overflow in libxslt. *Credit to Holger Fuhrmannek*
[$3000][678461] **High** CVE-2017-5034: Use after free in PDFium. *Credit to Ke Liu of Tencent's Xuanwu LAB*
[$3000][688425] **High** CVE-2017-5035: Incorrect security UI in Omnibox. *Credit to Enzo Aguado*
[$3000][691371] **High** CVE-2017-5036: Use after free in PDFium. *Credit to Anonymous*
[$1000][679640] **High** CVE-2017-5037: Multiple out of bounds writes in ChunkDemuxer. *Credit to Yongke Wang of Tencent's Xuanwu Lab (xlab.tencent.com)*
[$500][679649] **High** CVE-2017-5039: Use after free in PDFium. *Credit to jinmo123*
[$2000][691323] **Medium** CVE-2017-5040: Information disclosure in V8. *Credit to Choongwoo Han*
[$1000][642490] **Medium** CVE-2017-5041: Address spoofing in Omnibox. *Credit to Jordi Chancel*
[$1000][669086] **Medium** CVE-2017-5033: Bypass of Content Security Policy in Blink. *Credit to Nicolai Grødum*
[$1000][671932] **Medium** CVE-2017-5042: Incorrect handling of cookies in Cast. *Credit to Mike Ruddy*
[$1000][695476] **Medium** CVE-2017-5038: Use after free in GuestView. *Credit to Anonymous*
[$1000][683523] **Medium** CVE-2017-5043: Use after free in GuestView. *Credit to Anonymous*
[$1000][688987] **Medium** CVE-2017-5044: Heap overflow in Skia. *Credit to Kushal Arvind Shah of Fortinet's FortiGuard Labs*
[$500][667079] **Medium** CVE-2017-5045: Information disclosure in XSS Auditor. *Credit to Dhaval Kapil (vampire)*
[$500][680409] **Medium** CVE-2017-5046: Information disclosure in Blink. *Credit to Masato Kinugawa*

Security: Vulnerability lists

Intermezzo:

- Secure products:
  - Mention security fixes (don't hide it)
  - Have a website with all fixed security vulnerabilities
  - As pentest: Can see which vulnerabilities are in which versions
  - Vendor is open, up to date and ready for security issues

- Bad products:
  - Don't have a page with vulnerabilities
  - Don't mention security fixes in changelogs
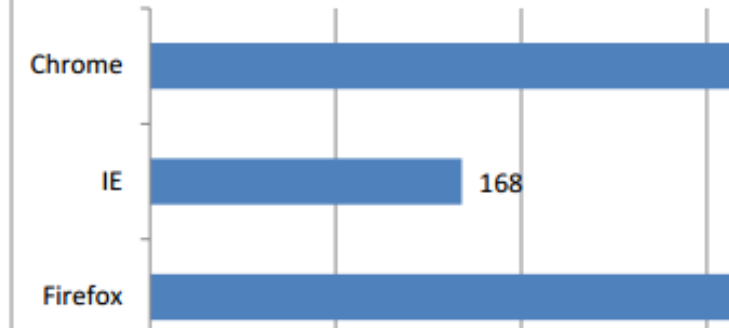 - Vendor hides, doesn't handle, obfuscate security issues

Security: CVE

CVE:

- Common Vulnerabilities and Exposures
- A vulnerability get a CVE (e.g. CVE-2017-1234)
  - Which software is affected
  - Which version
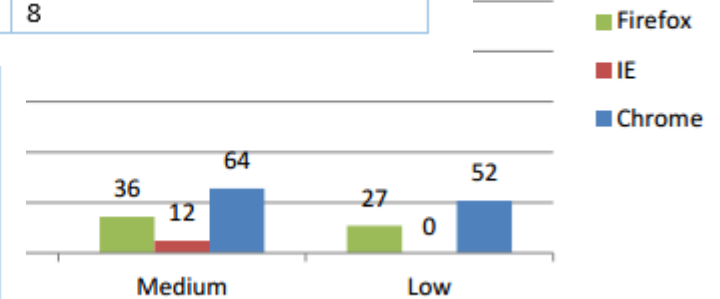 - When did it got fixed
  - …

Security: CVE

Vulnerabilites

rank	browser	number of vulnerabilities
1	Microsoft Internet Explorer	231
2	Google Chrome	187
3	Mozilla Firefox	178
4	Apple Safari	135
5	Mozilla Firefox Extended Support Release	94

rank	application	number of vulnerabilities
1	Adobe Flash Player	314
2	Adobe Air, SDK, and Compiler	246
3	Adobe Acrobat and Reader	129
4	Apple iTunes	100
5	Adobe Acrobat Document Cloud and Reader	97
6	Oracle Java Runtime Environment and JDK	80
7	Oracle MySQL	76
8	Oracle Fusion Middleware	68
9	Apple TV application	57
10	Oracle E-Business Suite	37
11	OpenSSL	34
12	Wireshark	33
13	MediaWiki	31
14	Mozilla Thunderbird	29
15	Oracle Database Server	29
16	Microsoft Office 2007	12
17	Microsoft Office 2010	11
18	Microsoft Office 2013	8

rank	operating system	
1	Apple OS X	
2	Microsoft Windows Server 2012	
3	Canonical Ubuntu Linux	
4	Microsoft Windows 8.1	
5	Microsoft Windows Server 2008	
6	Microsoft Windows 7	
7	Microsoft Windows 8	
8	Microsoft Windows Vista	135
9	openSUSE	121
10	Debian Linux	111
11	The Linux Kernel	77
12	Microsoft Windows 10	53
13	Fedora Linux	38
14	Microsoft Windows 2003	36
15	Xen OS	34

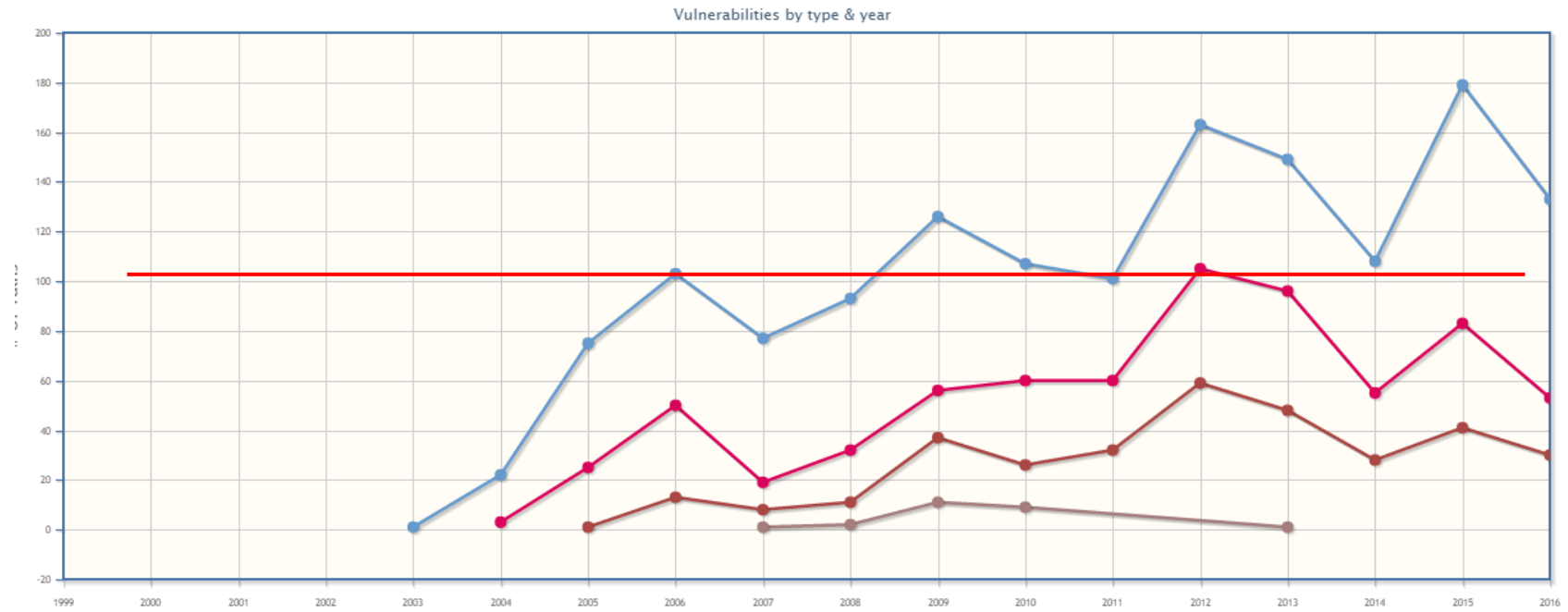
ilities by severity for each browser

Security: CVE

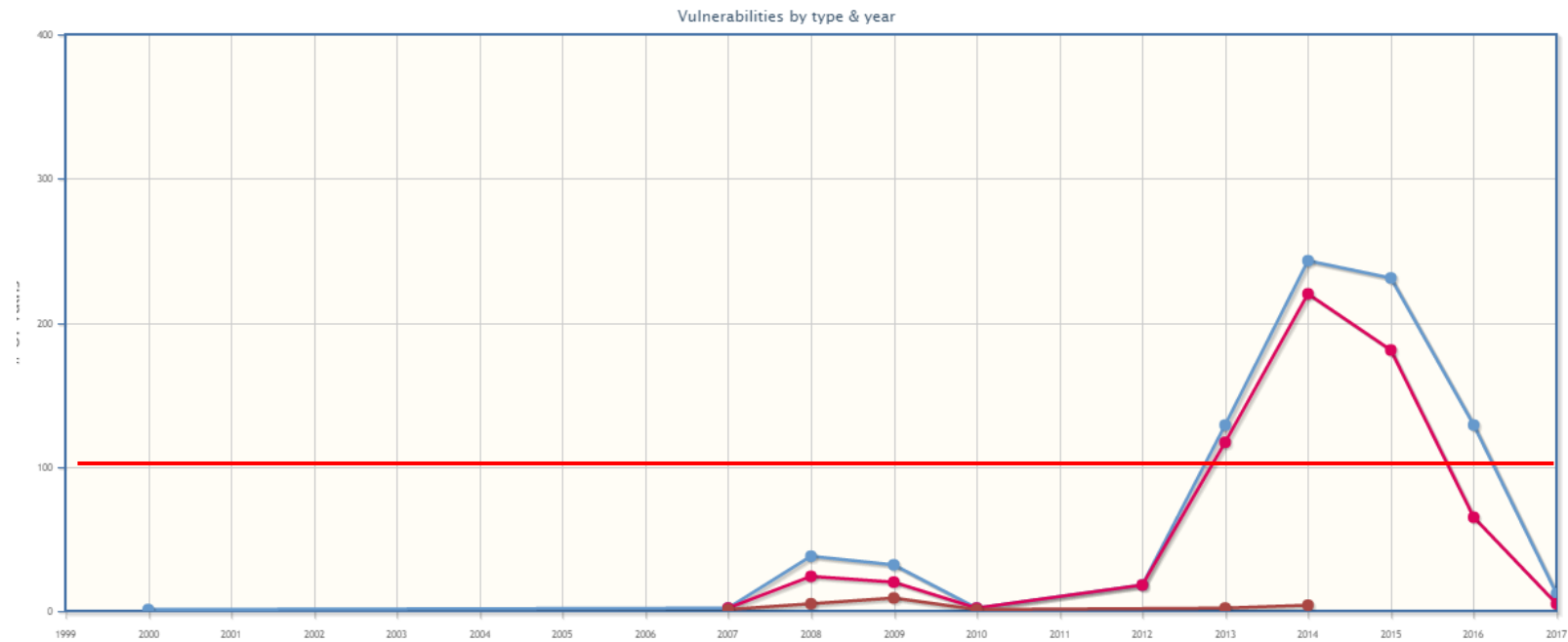Weakness comparison fails: (not just CVE)

- Scope: "Windows vs Linux"
  - What is in Linux? Linux Kernel? Suse? LIBC? Bash? Apache?
  - What is in Windows? Internet Explorer? IIS?
- Severity mismatch
  - When is a vulnerability "critical"? When is it "high"?
  - Microsoft categorizes differently than Mozilla, or Google
- Number of vulnerabilities in CVE / bulletin
  - 1 vulnerability, one CVE / securtiy bullettin ?
  - 1 CVE for each product affected? (Cisco: RCE in product x, y, z)
  - 1 CVE for each individual bug? (e.g. UAF in component x, y, z)
- Vulnerablity disclosure
  - CVE's for all the bugs found internally? (e.g. fuzzing)
  - CVE for all the bugs found by looking for similar bugs?
- ...

-> Don't compare different product's security issues by counting <-

References

Resources:

- http://homes.soic.indiana.edu/yh33/Teaching/I433-2016/lec13-HeapAttacks.pdf
- http://www.pwntester.com/blog/2014/03/23/codegate-2k14-4stone-pwnable-300-write-up/