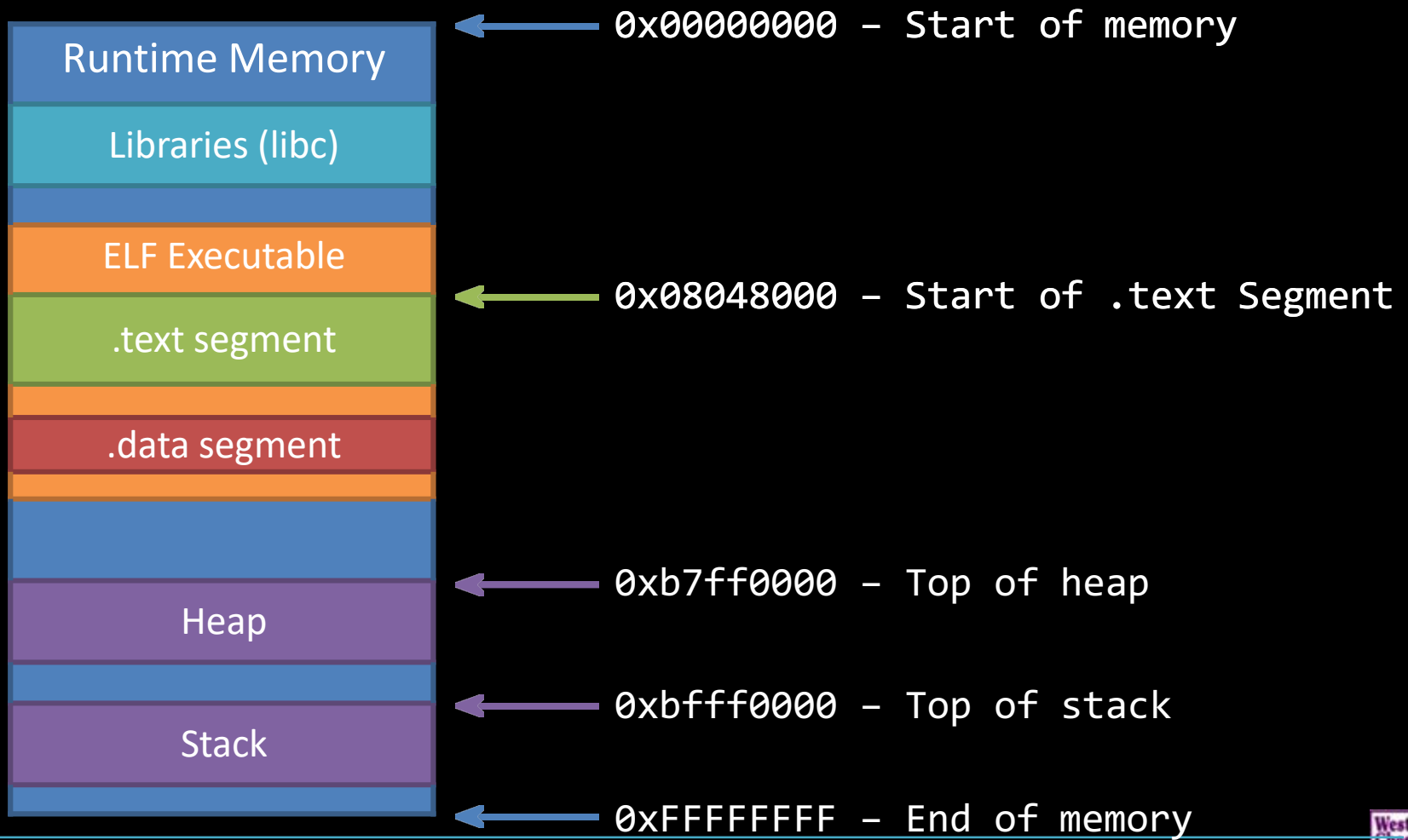
Class 21

CSC 472 Software Security
Heap Exploitation (3): House of Force

Dr. Si Chen (schen@wcupa.edu)

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 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

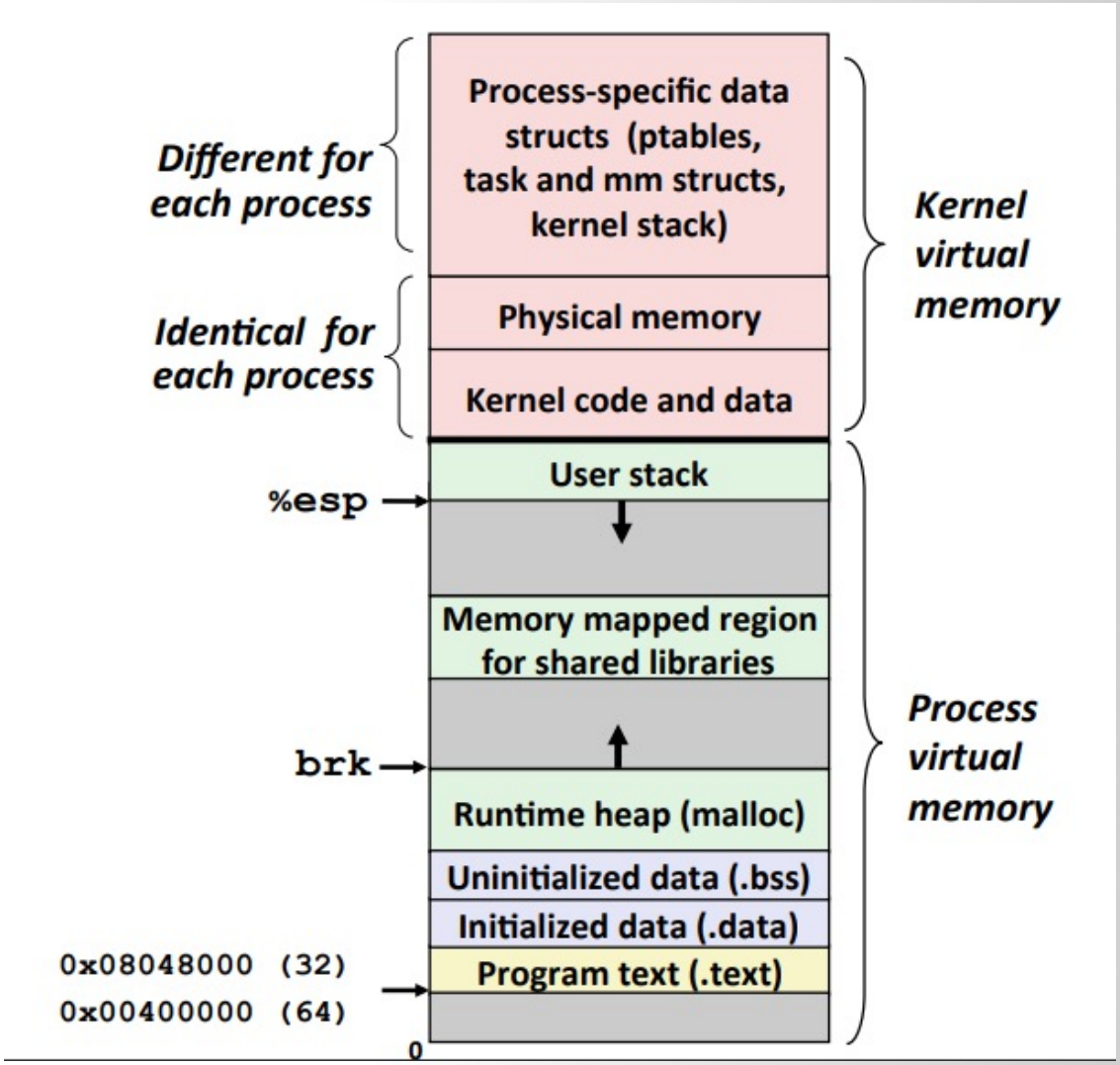
```
#include <sys/mman.h>

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).

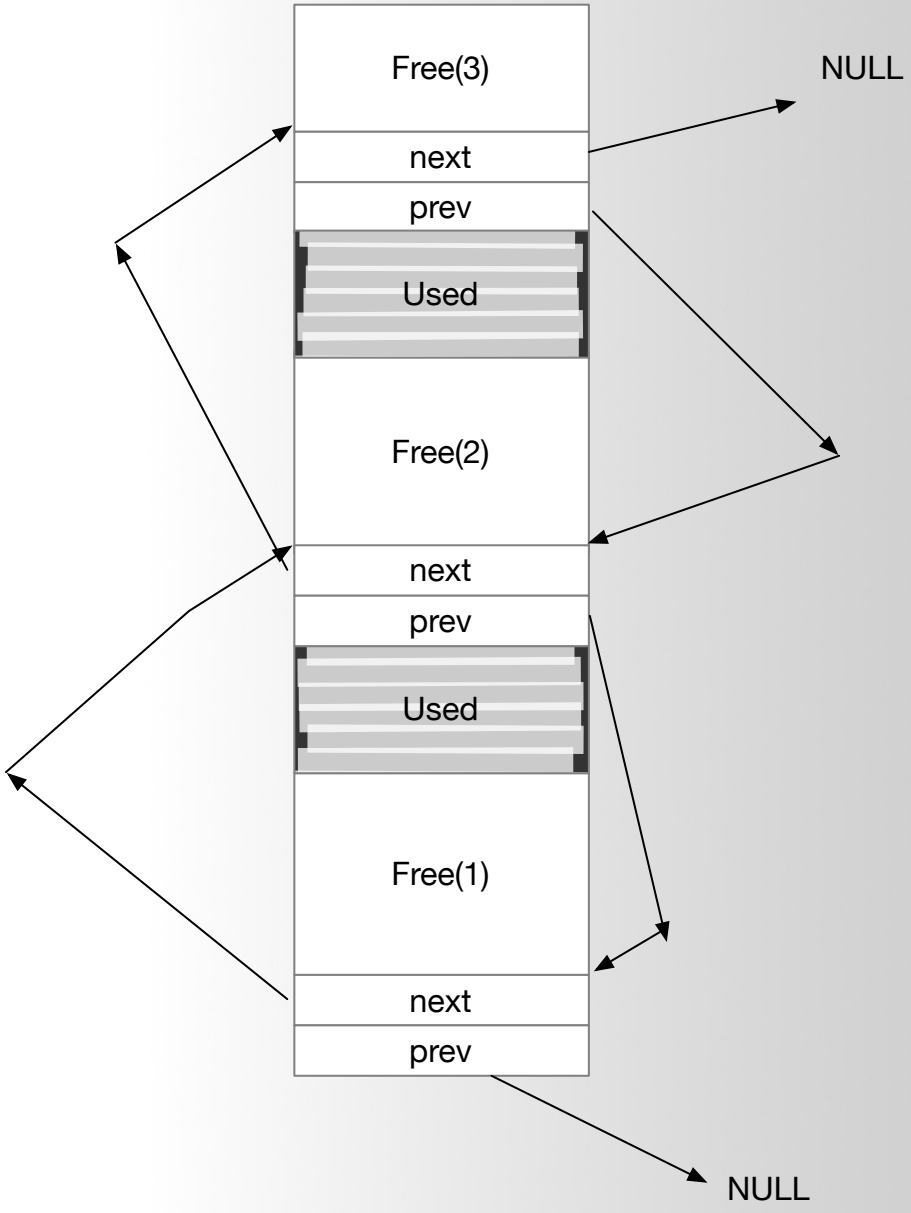
West Chester University

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

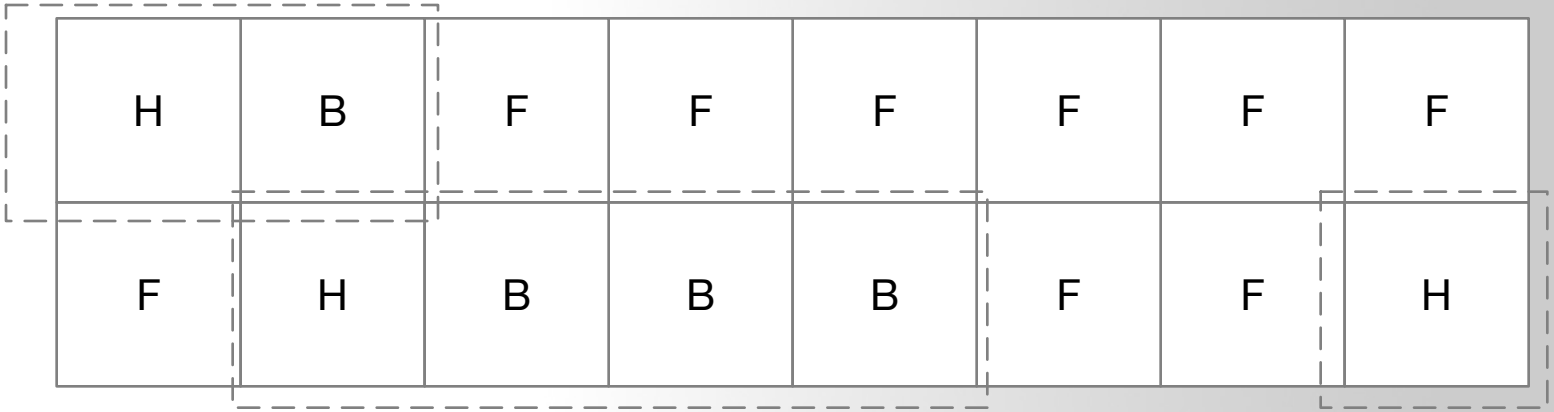
- Linked List



Design your own Heap management system

- bitmap

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



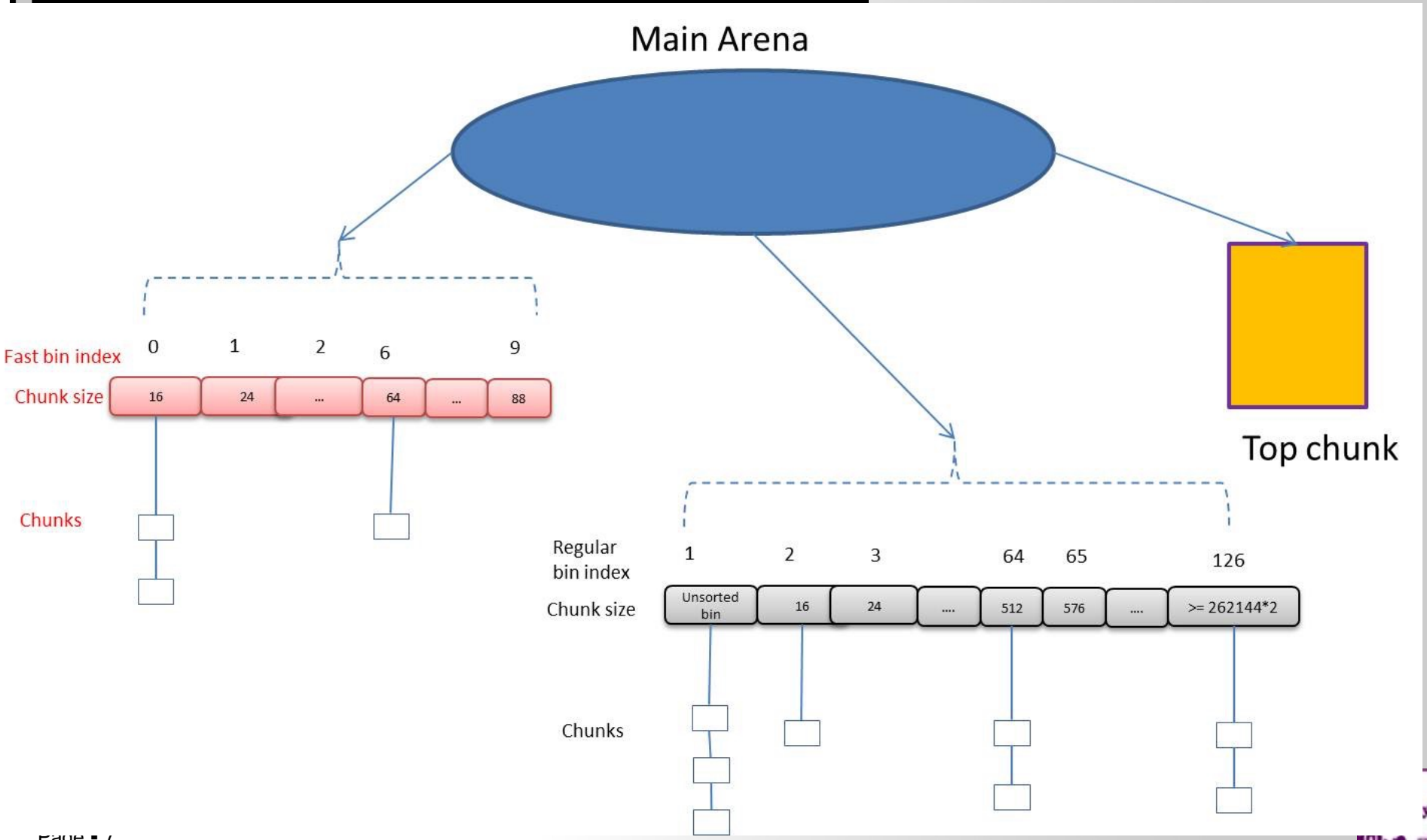
Bitmap representation:
(HIGH) 11 00 00 10 10 10 11 00 00 00 00 00 00 00 10 11 (LOW)

128 byte per chunk
1MB / 128 = 8k

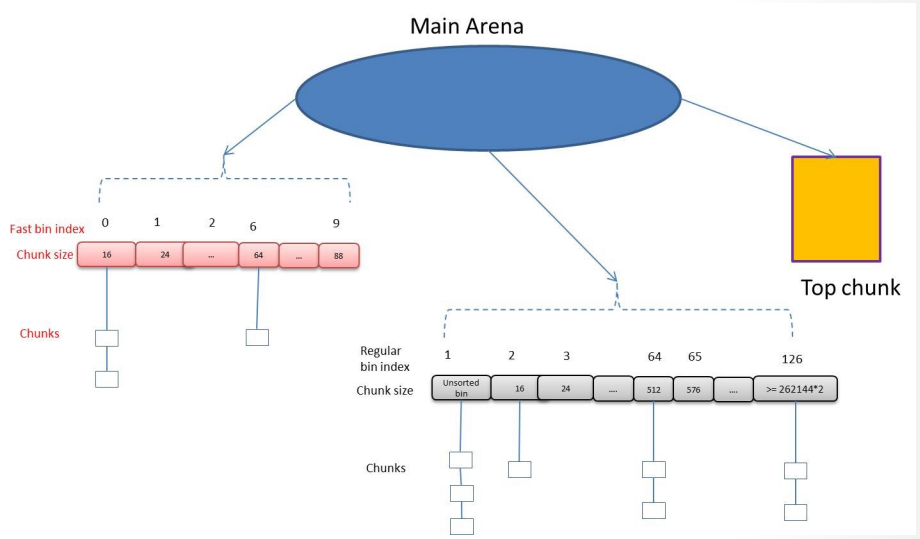
glibc source code

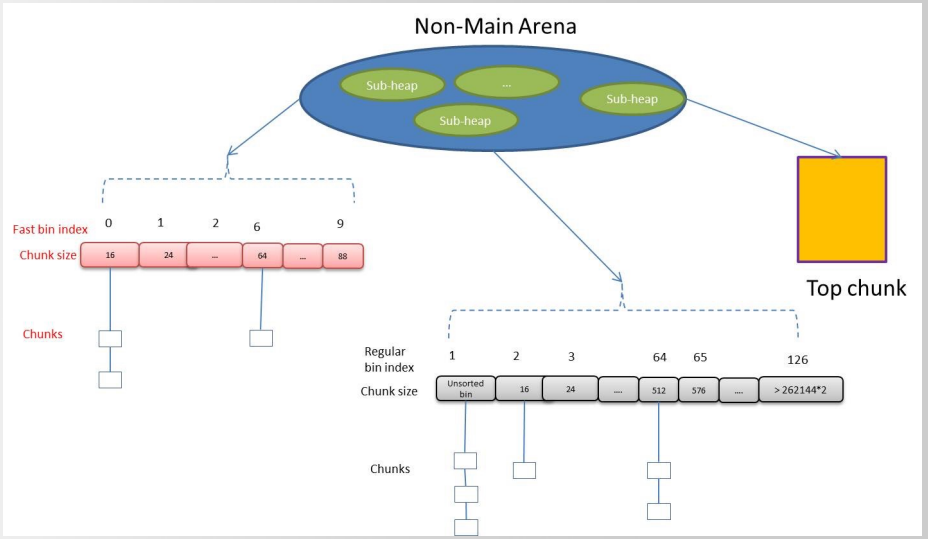
https://code.woboq.org/userspace/glibc/malloc/malloc.c.html

```
➜  ~ $ 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;
};
```

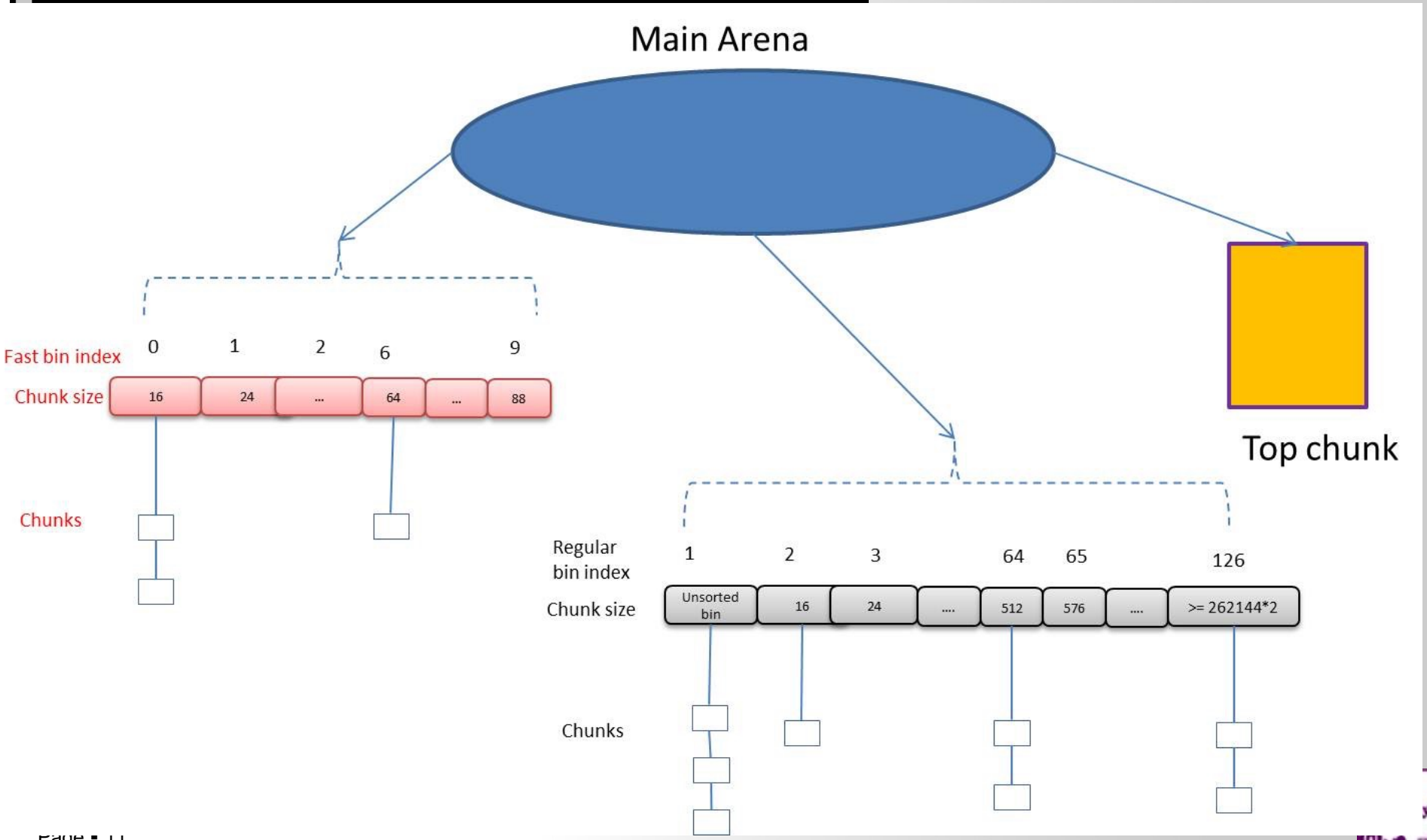
Arena

```
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

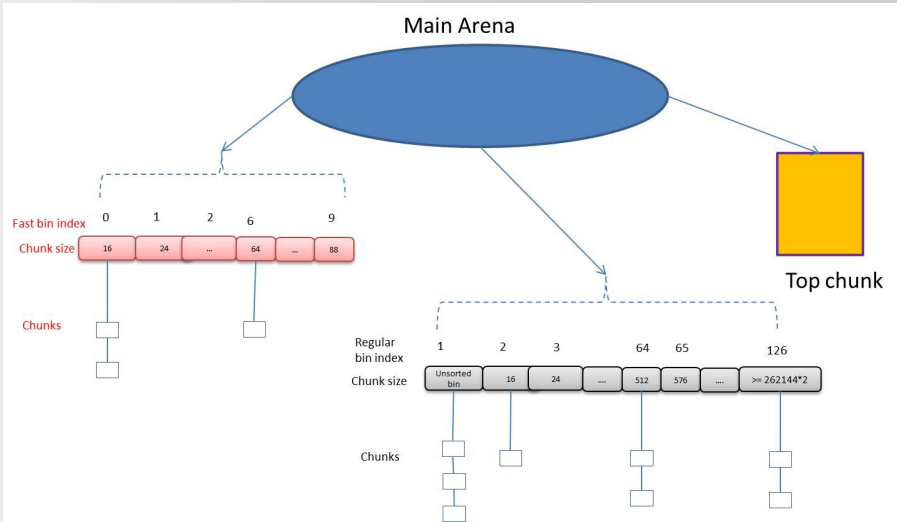
https://code.woboq.org/userspace/glibc/malloc/malloc.c.html

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



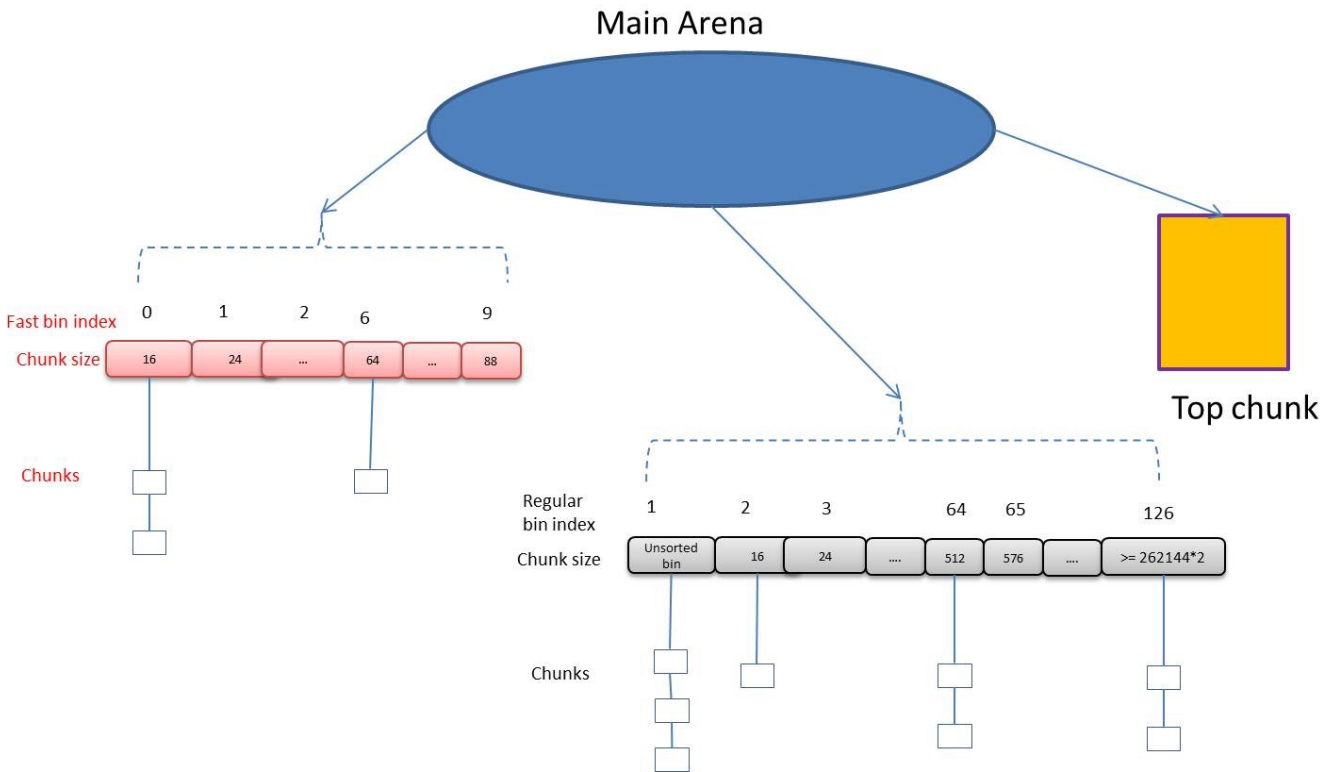
Bins and Chunks

- 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.



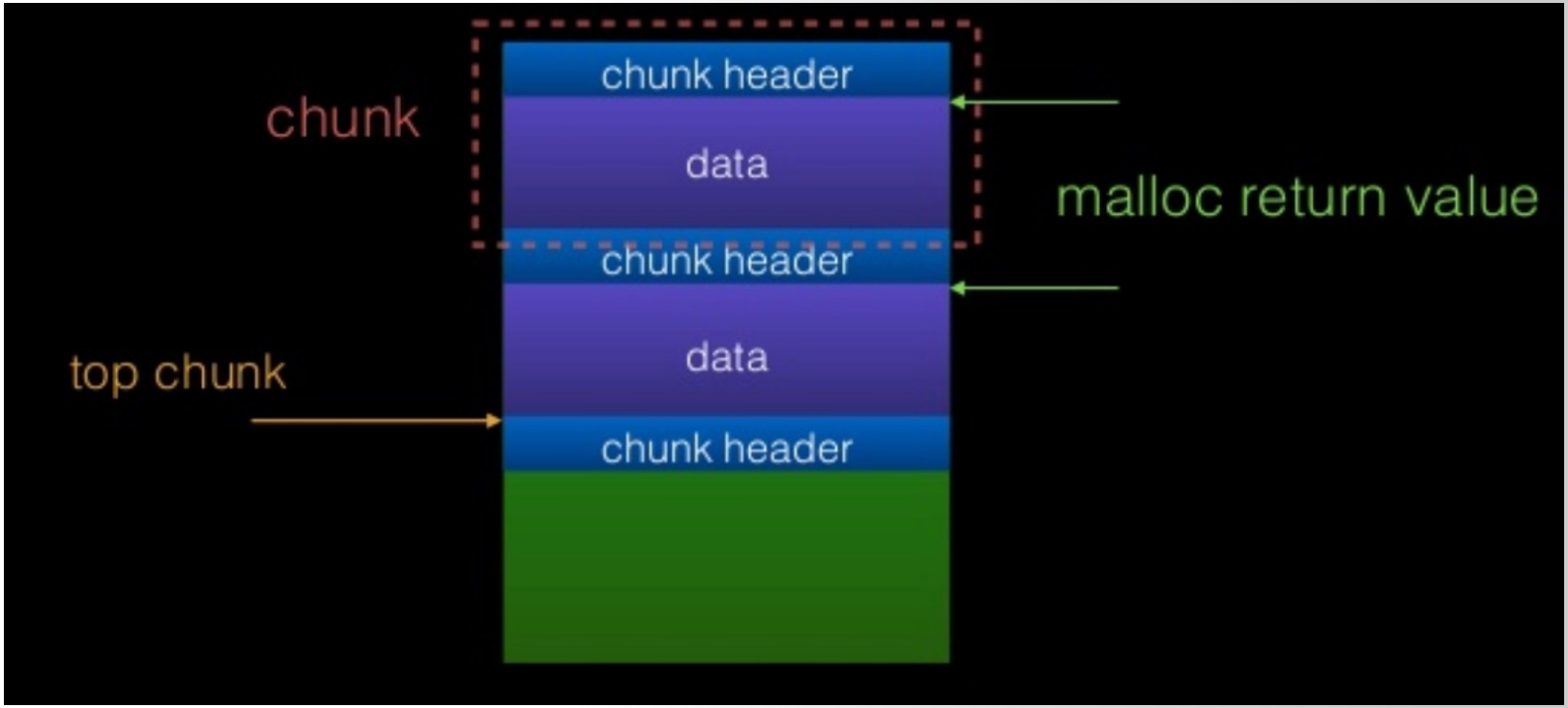
Bins and Chunks

- 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

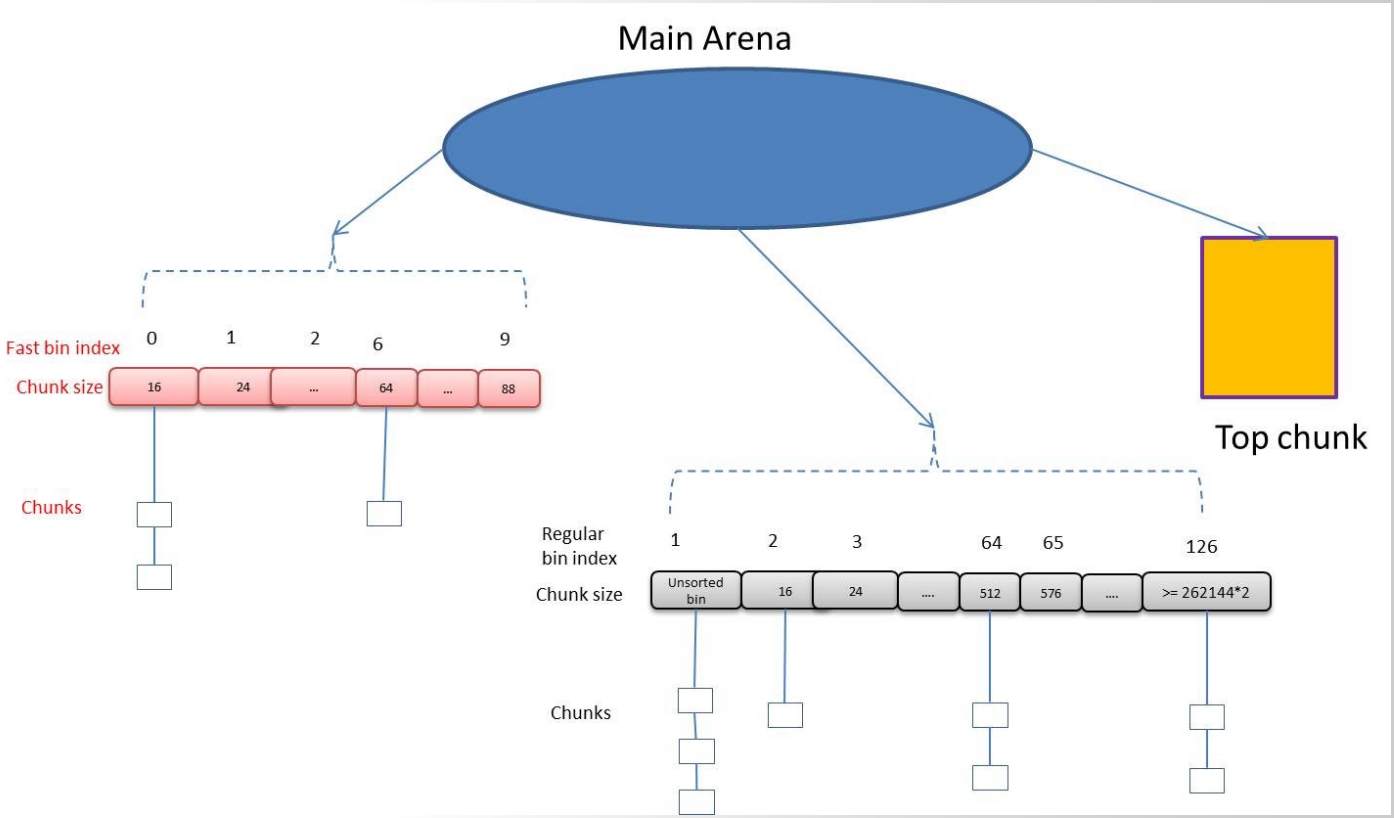
- *Top Chunk*: Chunk which is at the top border of an arena is called top chunk. It doesn't belong to any bin. Top chunk is used to service user request when there is NO free blocks, in any of the bins. If top chunk size is greater than user requested size top chunk is split into two:
  - User chunk (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

House of Force

- **House of Force**: In this technique, attacker abuses top chunk size and tricks ‘glibc malloc’ to service a very large memory request (greater than heap system memory size) using top chunk. Now when a new malloc request is made, GOT entry of free would be overwritten with shellcode address. Hence from now on whenever free is called, shellcode gets executed!!

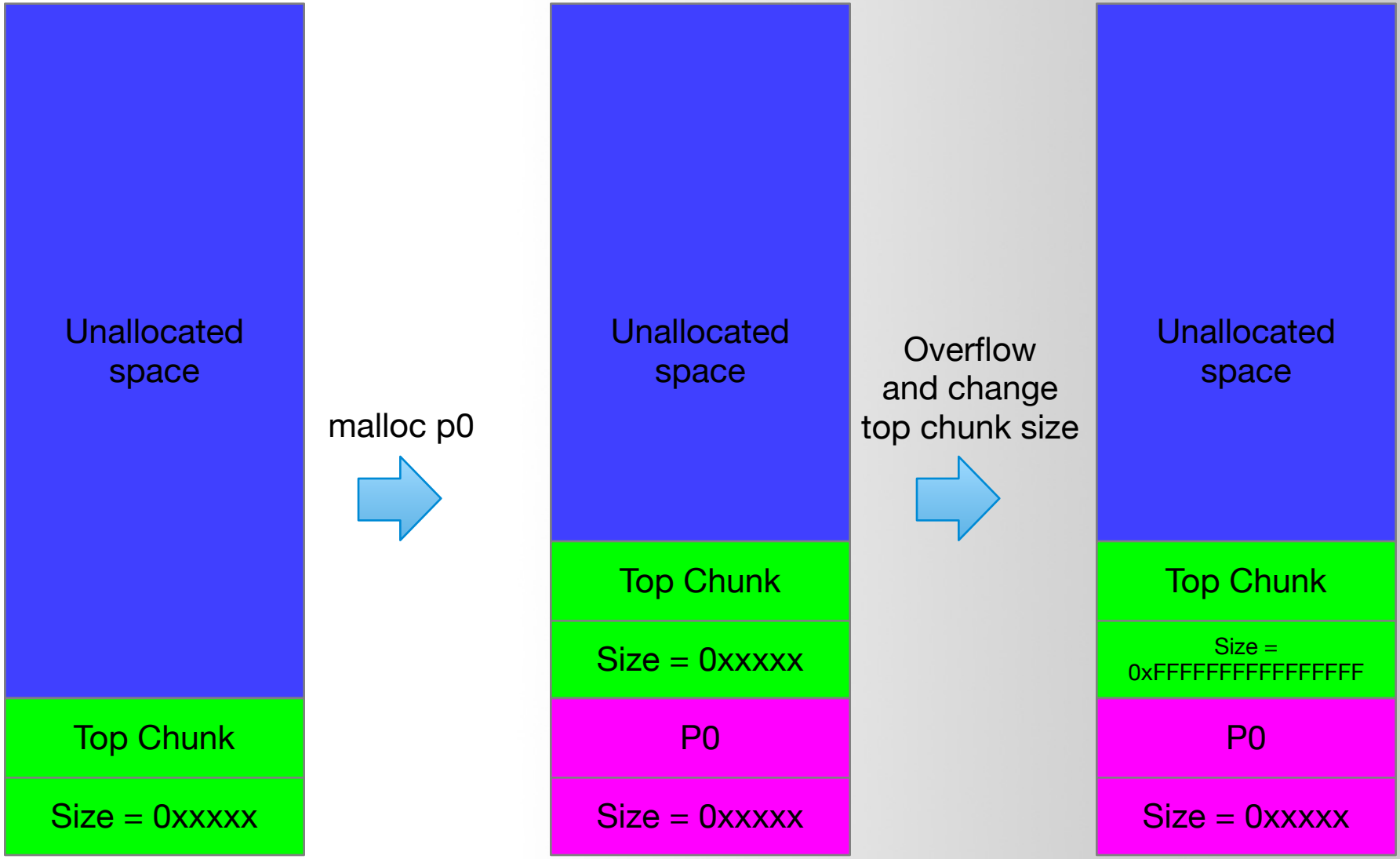


The Malloc Maleficarum (2004)

- 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~~

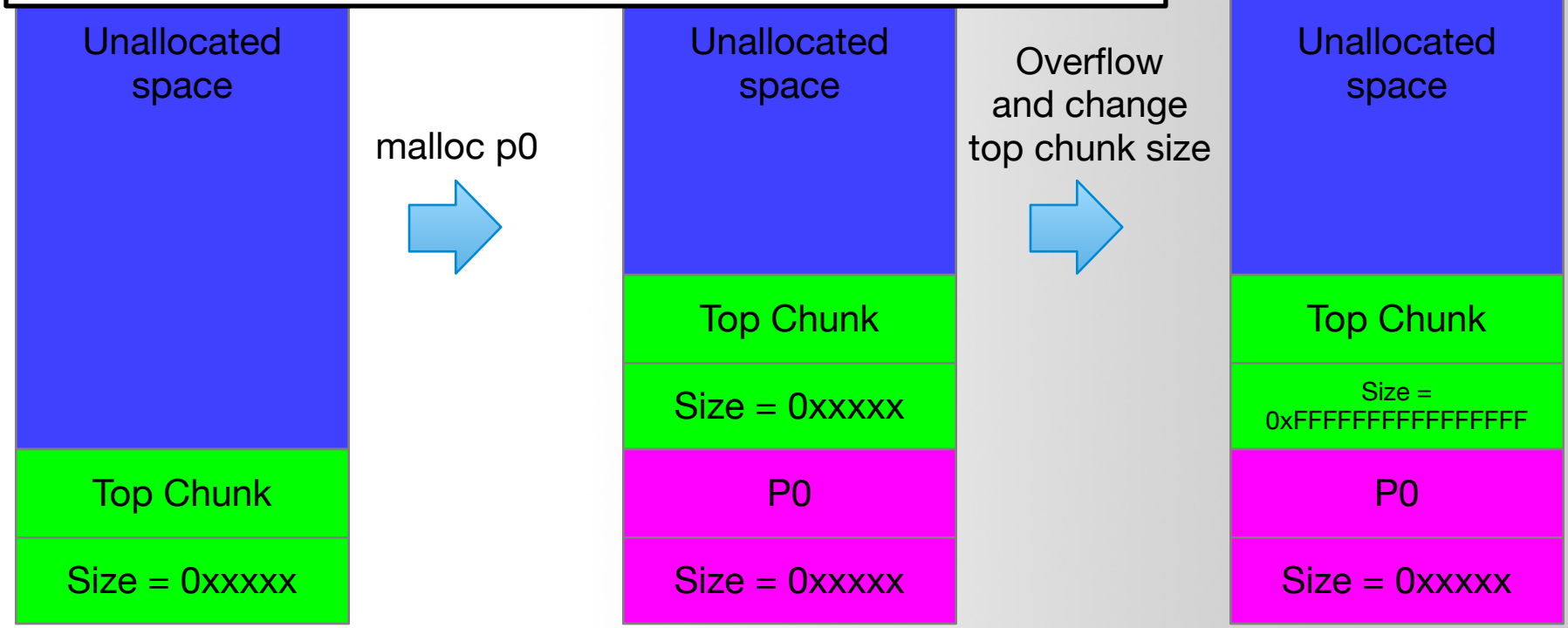
https://dl.packetstormsecurity.net/papers/attack/MallocMaleficarum.txt

House of Force



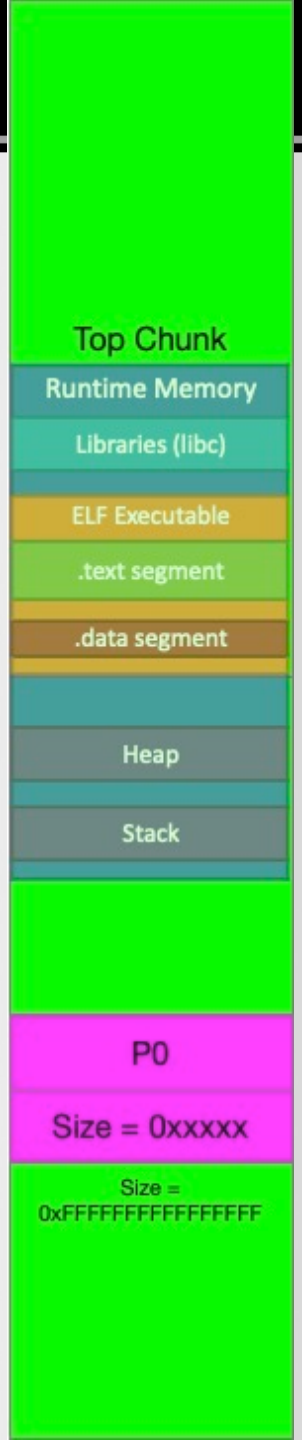
House of Force

- This attack assumes an overflow into the top chunk's header. The size is modified to a very large value (-1 in this example).
- This ensures that all initial requests will be services using the top chunk, instead of relying on mmap.
- On a 64 bit system, -1 evaluates to 0xFFFFFFFFFFFFFFFF.
- A chunk with this size can cover the entire memory space of the program.



House of Force

Let us assume that the attacker wishes 'malloc' to return address P. Now, any malloc call with the size of: &top_chunk - P will be serviced using the top chunk. Note that P can be after or before the top_chunk.



House of Force

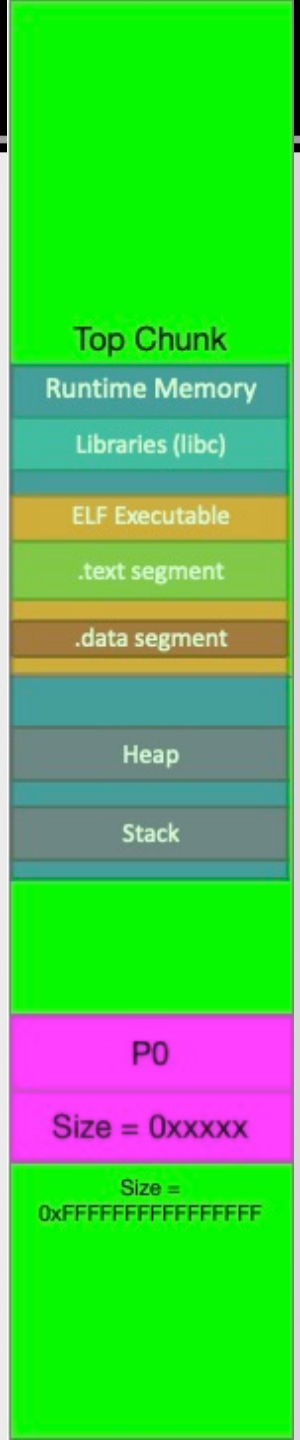
E.g. top_chunk=0x601200

malloc(0xffe00030)

0xffe00030 < top_chunk_size

0xffe00030+0x601200=0x100401230

top_chunk=0x401230



House of Force

- *Prerequisites*: Three malloc calls are required to successfully apply house of force as listed below:
  - Malloc 1: Attacker should be able to control the size of top chunk. Hence heap overflow should be possible on this allocated chunk which is physically located previous to top chunk.
  - Malloc 2: Attacker should be able to control the size of this malloc request.
  - Malloc 3: User input should be copied to this allocated chunk.

House of Force - example

```c
//gcc hof.c -o hof -no-pie
#include<stdio.h>
#include<stdlib.h>

char bss_var[] = "This is a string that we want to overwrite.";

int main()
{
    unsigned long *p1 = malloc(0xf0);
    unsigned long *top = (unsigned long*)( (unsigned char*)p1 + 0xf0);

    //overwrite the size of top chunk
    *(top+1) = -1;

    unsigned long evil_size = (unsigned long)bss_var - sizeof(long)*2 - (unsigned long)top - sizeof(long) * 2;
    malloc(evil_size);

    unsigned long *victim = malloc(0x100);

    printf("The victim pointer: %p\n", victim);

    return 0;
}
```

Exercise: BambooBox

Source Code:

- https://github.com/ctf-wiki/ctf-challenges/blob/master/pwn/heap/house-of-force/hitcontraning_lab11/bamboobox.c

Solution:

https://github.com/ctf-wiki/ctf-challenges/blob/master/pwn/heap/house-of-force/hitcontraning_lab11/exp.py

Q & A