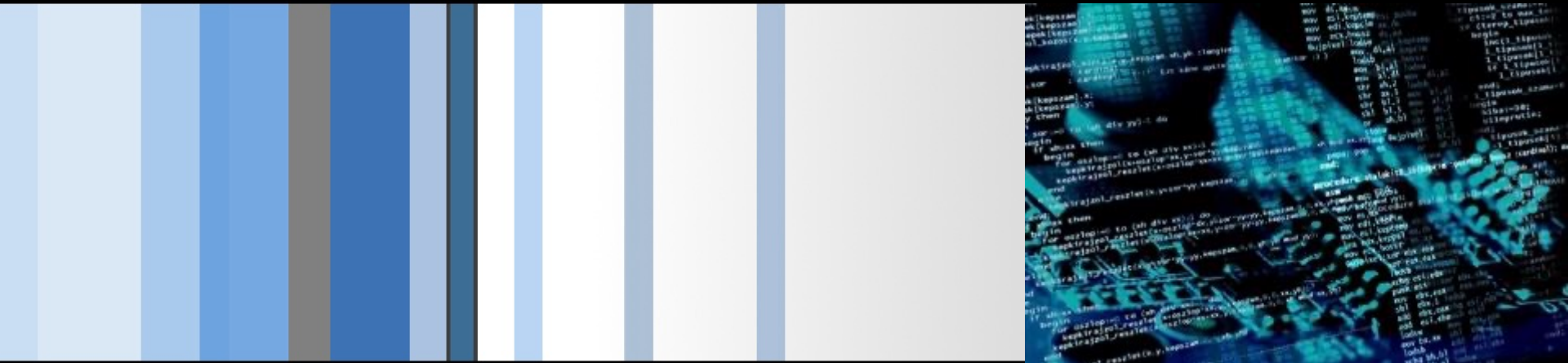


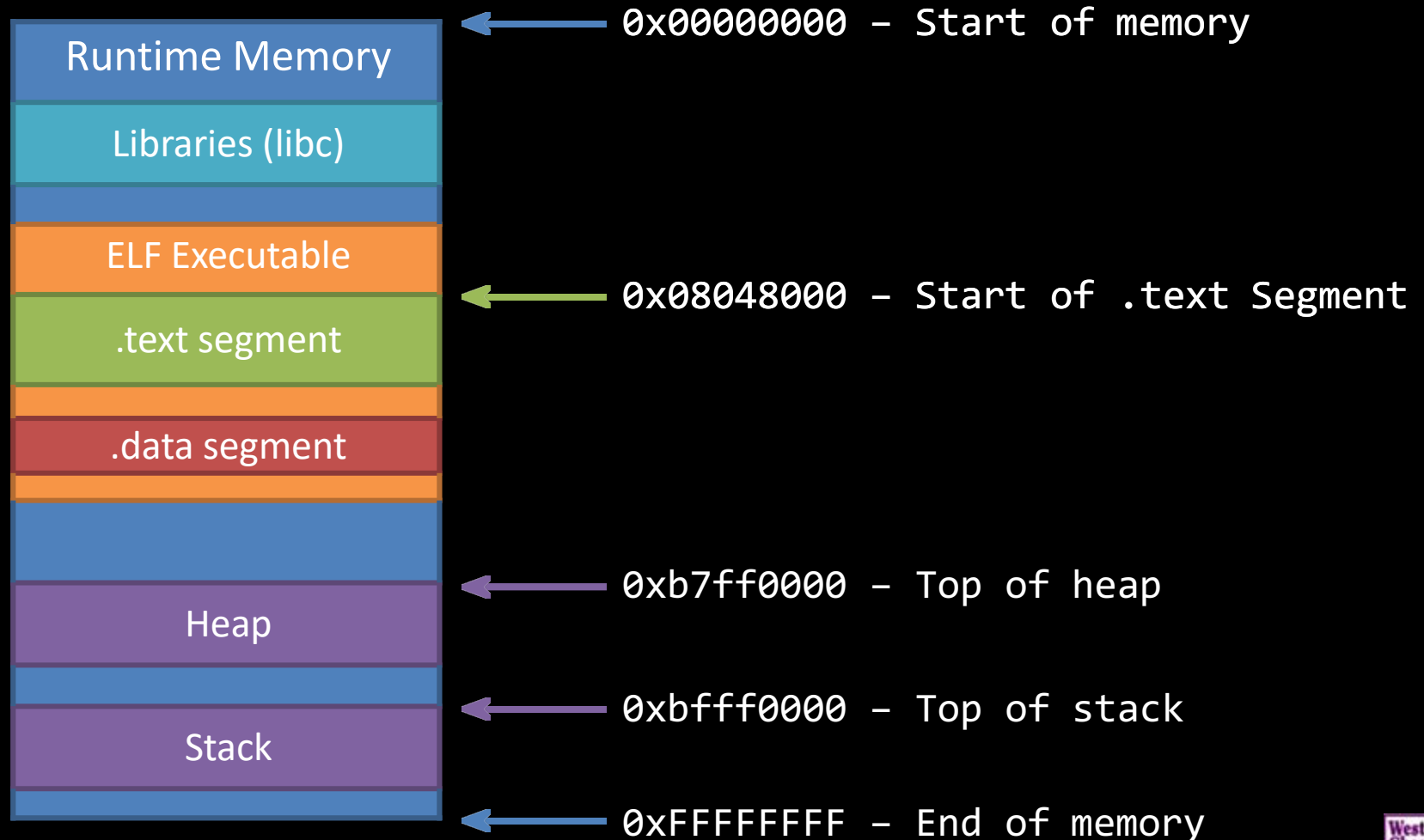
CSC 472/583 Topics of Software Security

Heap Exploitation (2): Unlink Attack

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>
```

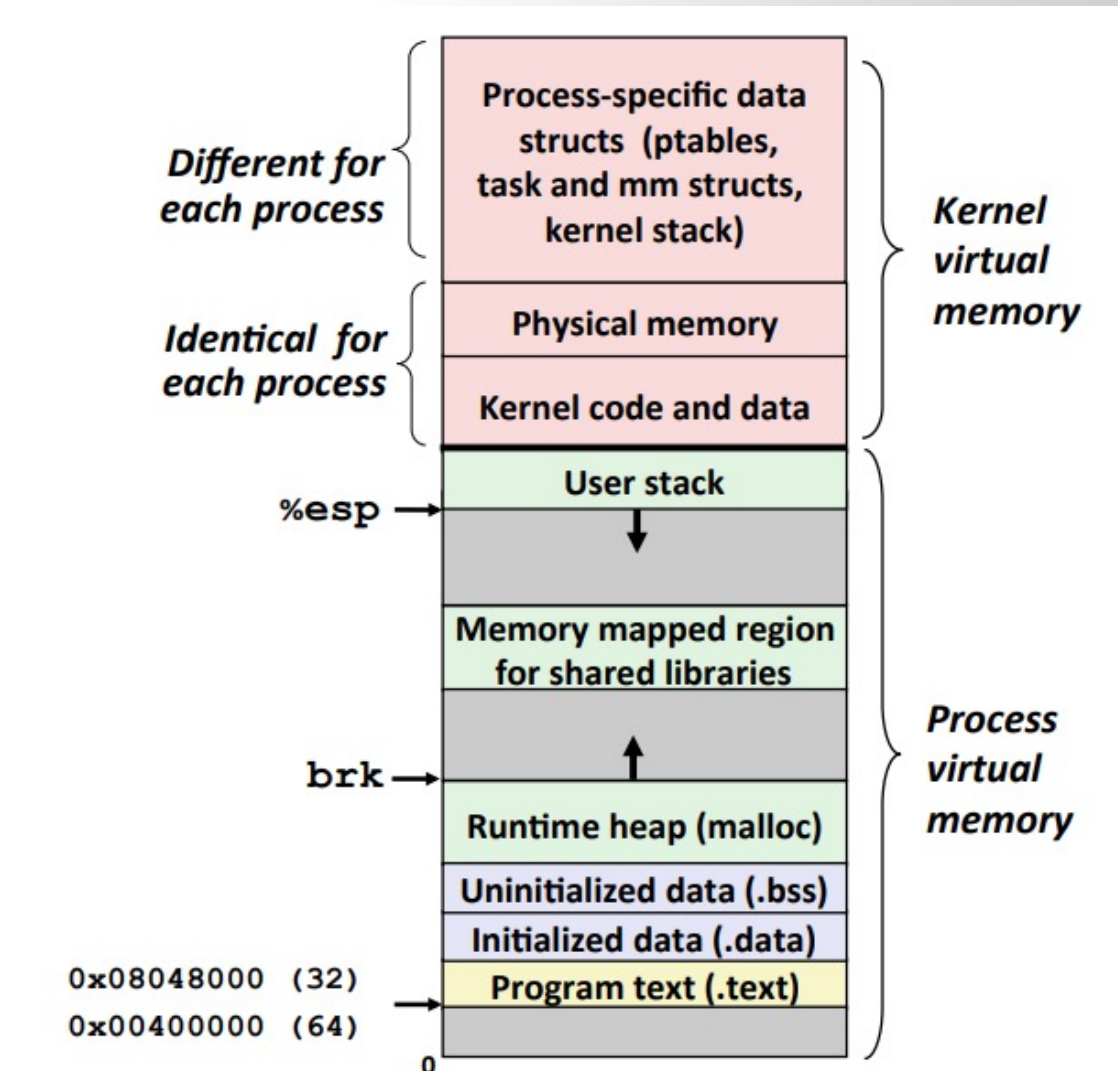
```
void *mmap(void *addr, size_t length, int prot, int flags,  
           int fd, off_t offset);  
int munmap(void *addr, size_t length);
```

See NOTES for information on feature test macro requirements.

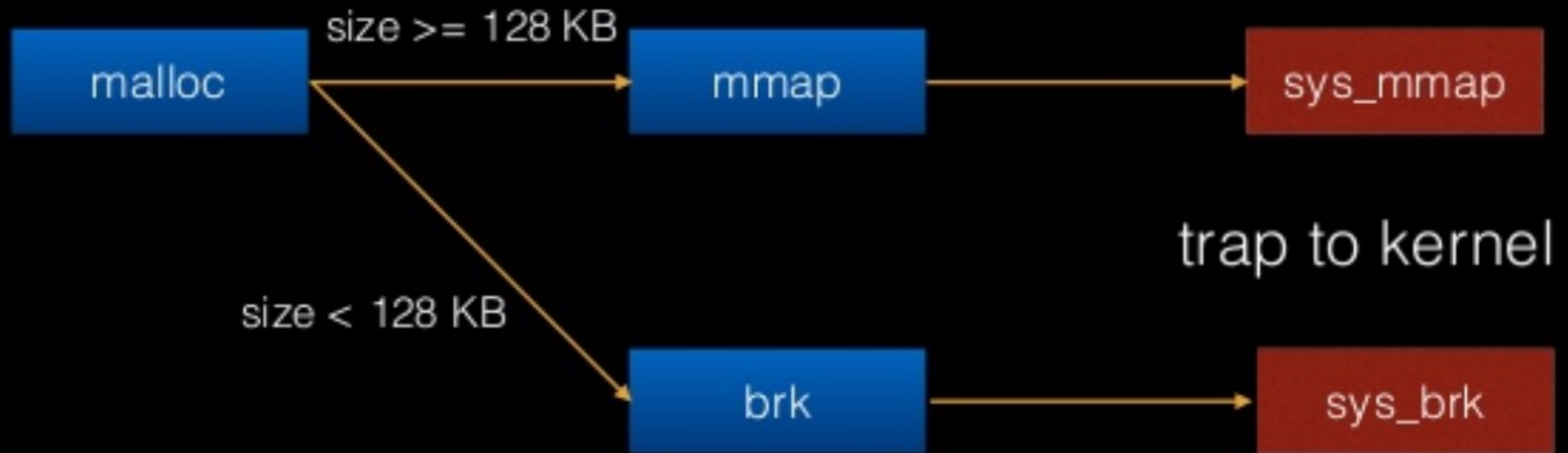
DESCRIPTION

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).

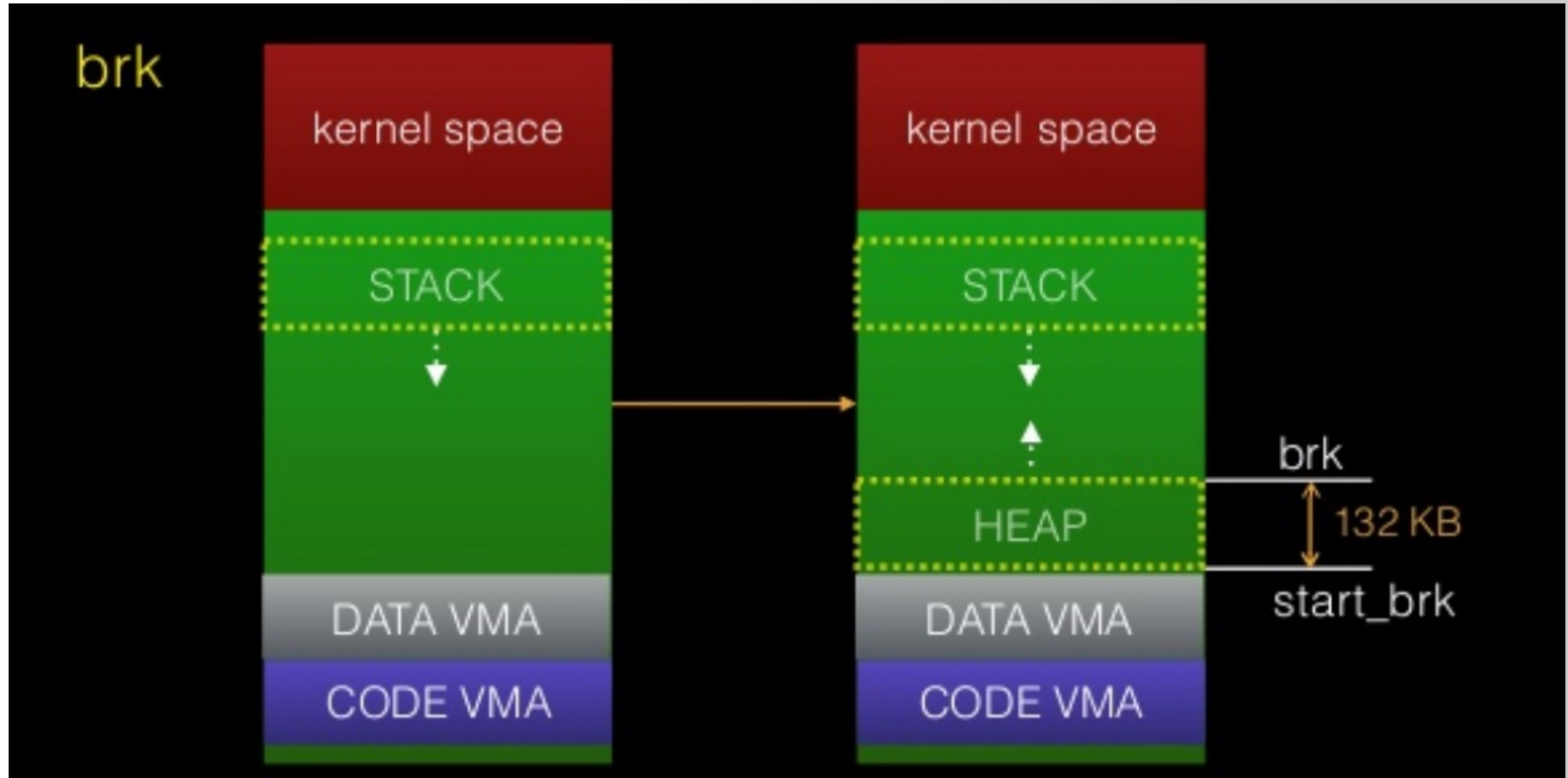
The Heap



The workflow of malloc()



The workflow of malloc()



If `malloc` size < 128KB then Kernel will return a 132KB heap segment (rw) which called main arena

Doug Lea's malloc Heap Chunks

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

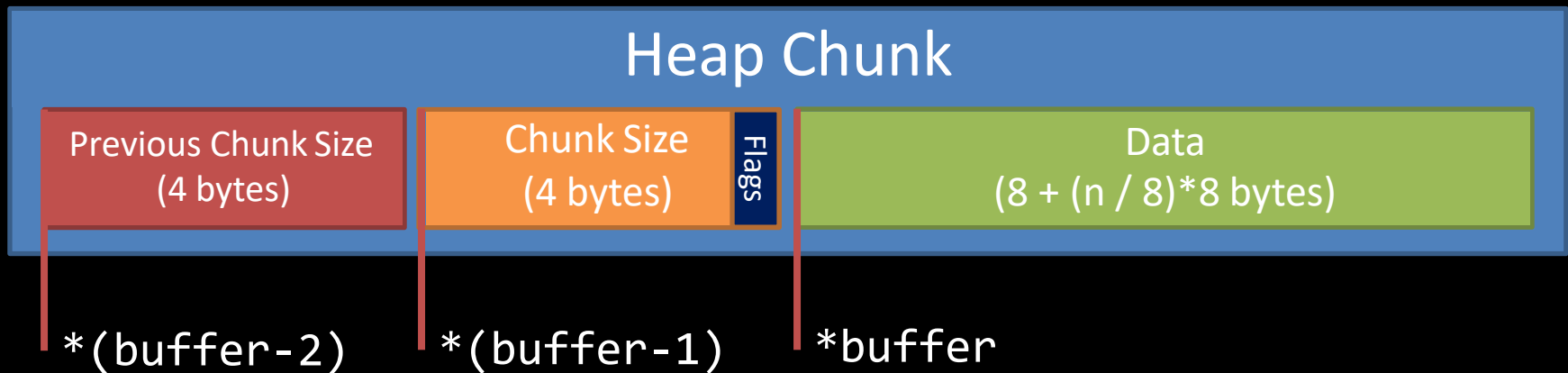
malloc chunk

```
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

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

//Out comes a heap chunk

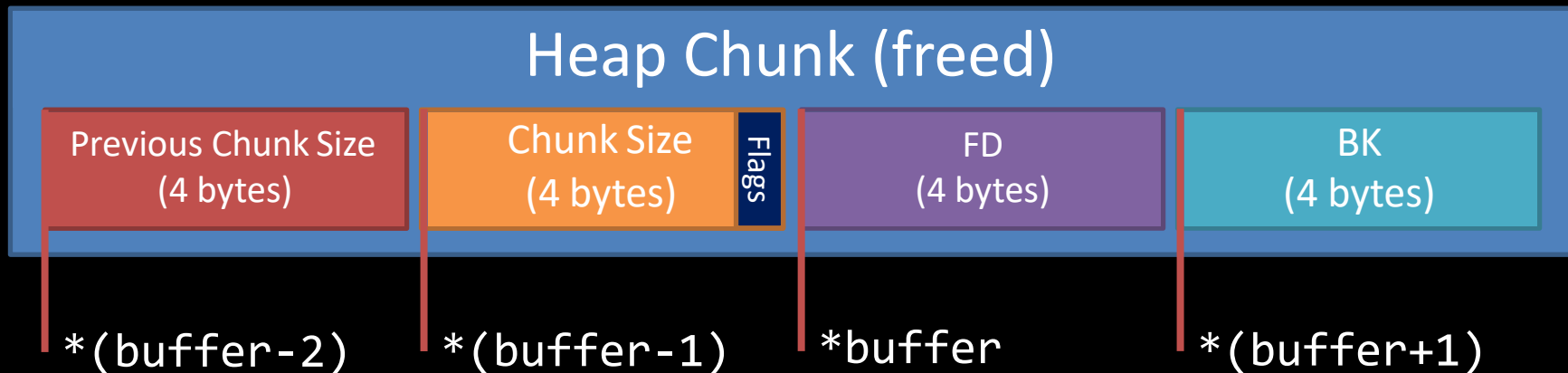


```
#define PREV_INUSE 0x1  
#define IS_MMAPPED 0x2  
#define NON_MAIN_ARENA 0x4
```

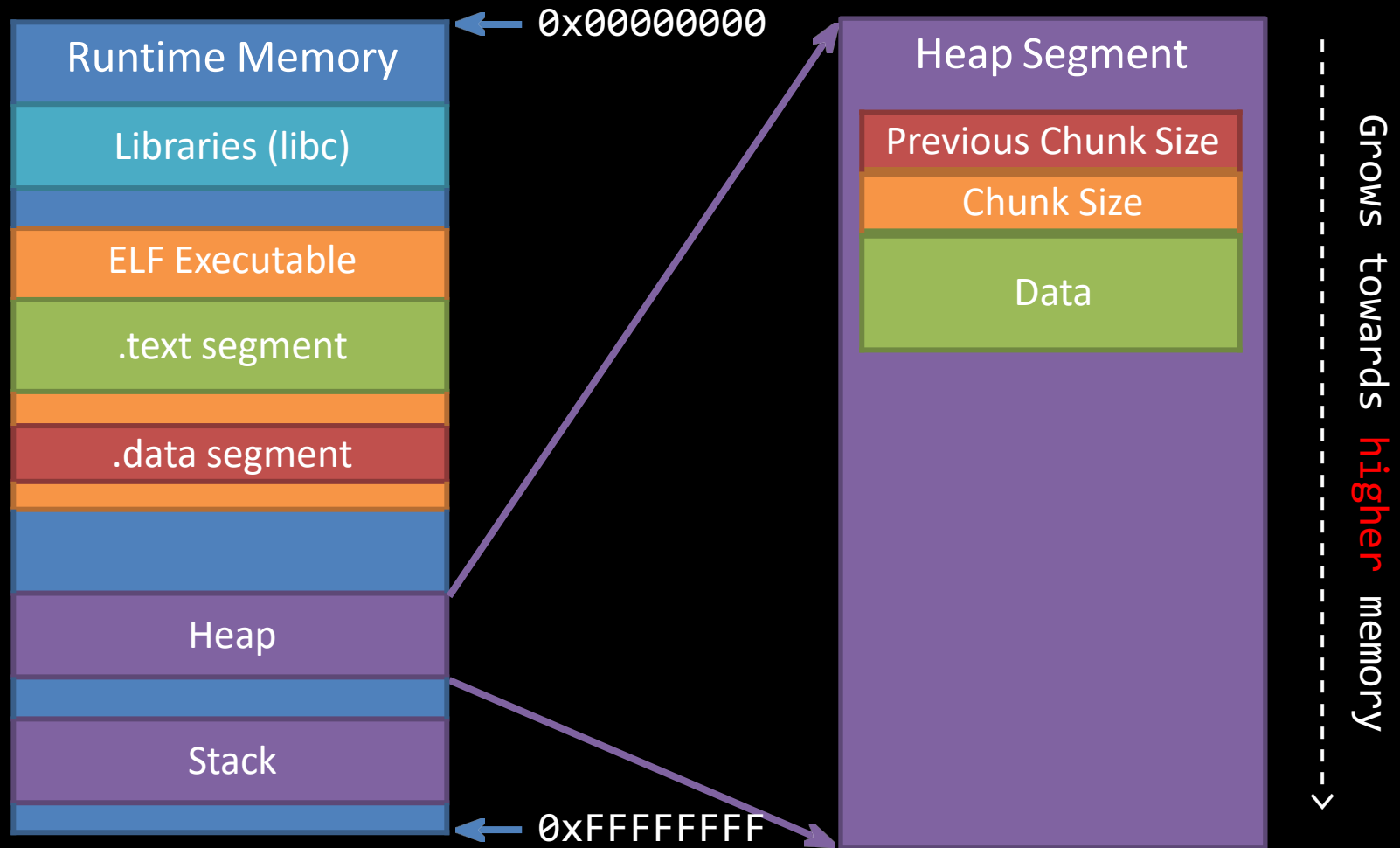
Heap Chunks – Freed

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

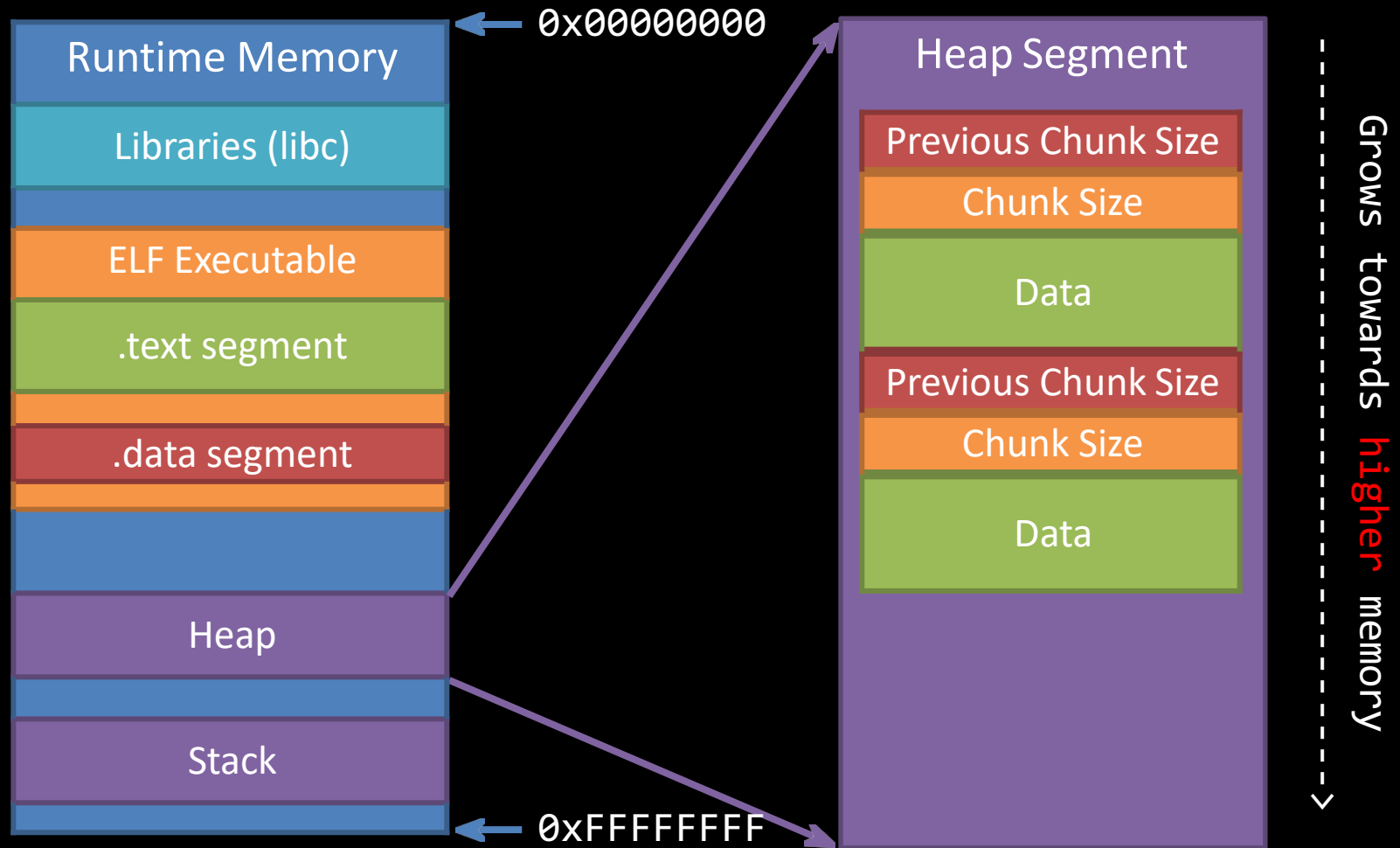
- Forward Pointer
 - A pointer to the next freed chunk
- Backwards Pointer
 - A pointer to the previous freed chunk



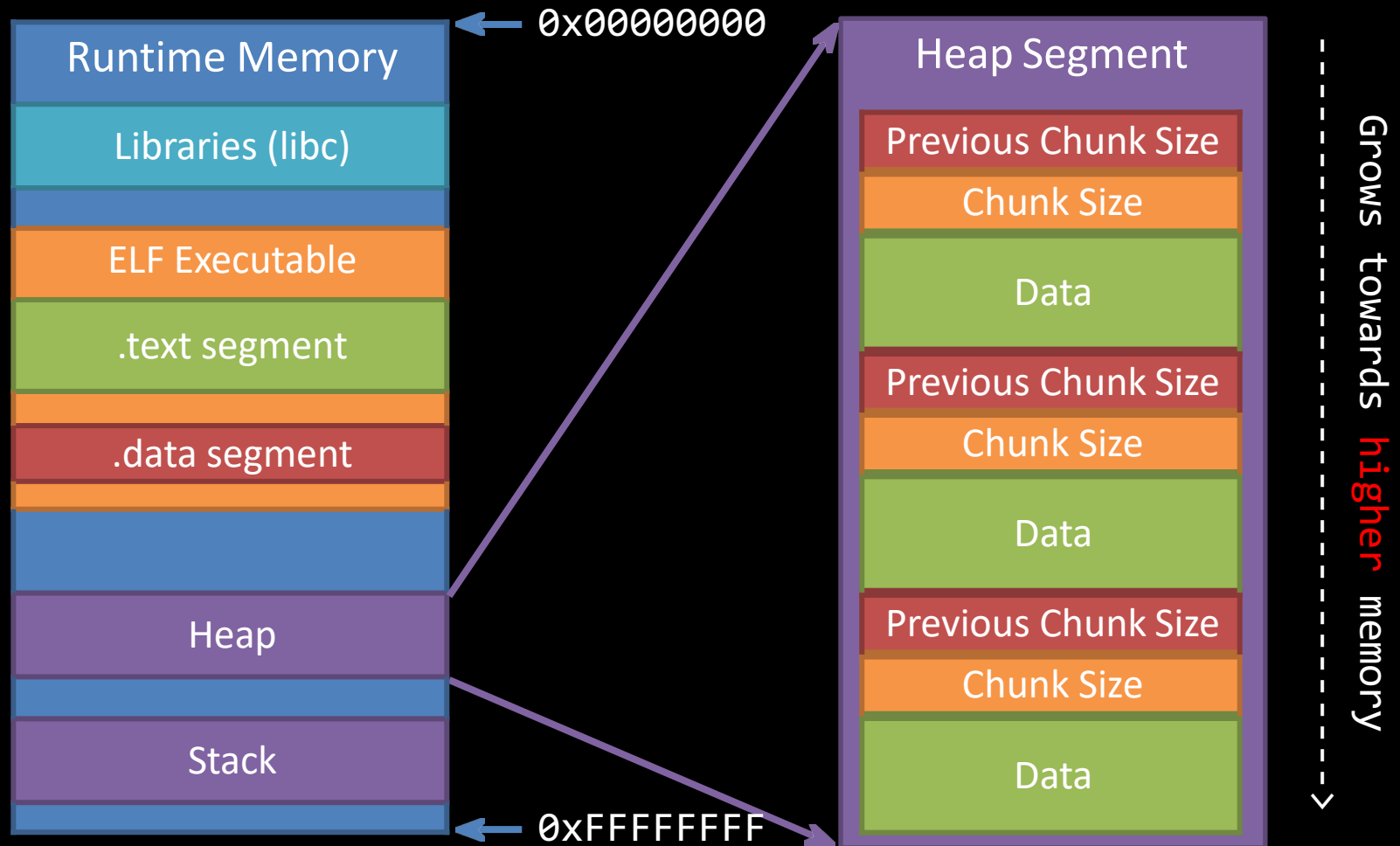
Heap Allocations



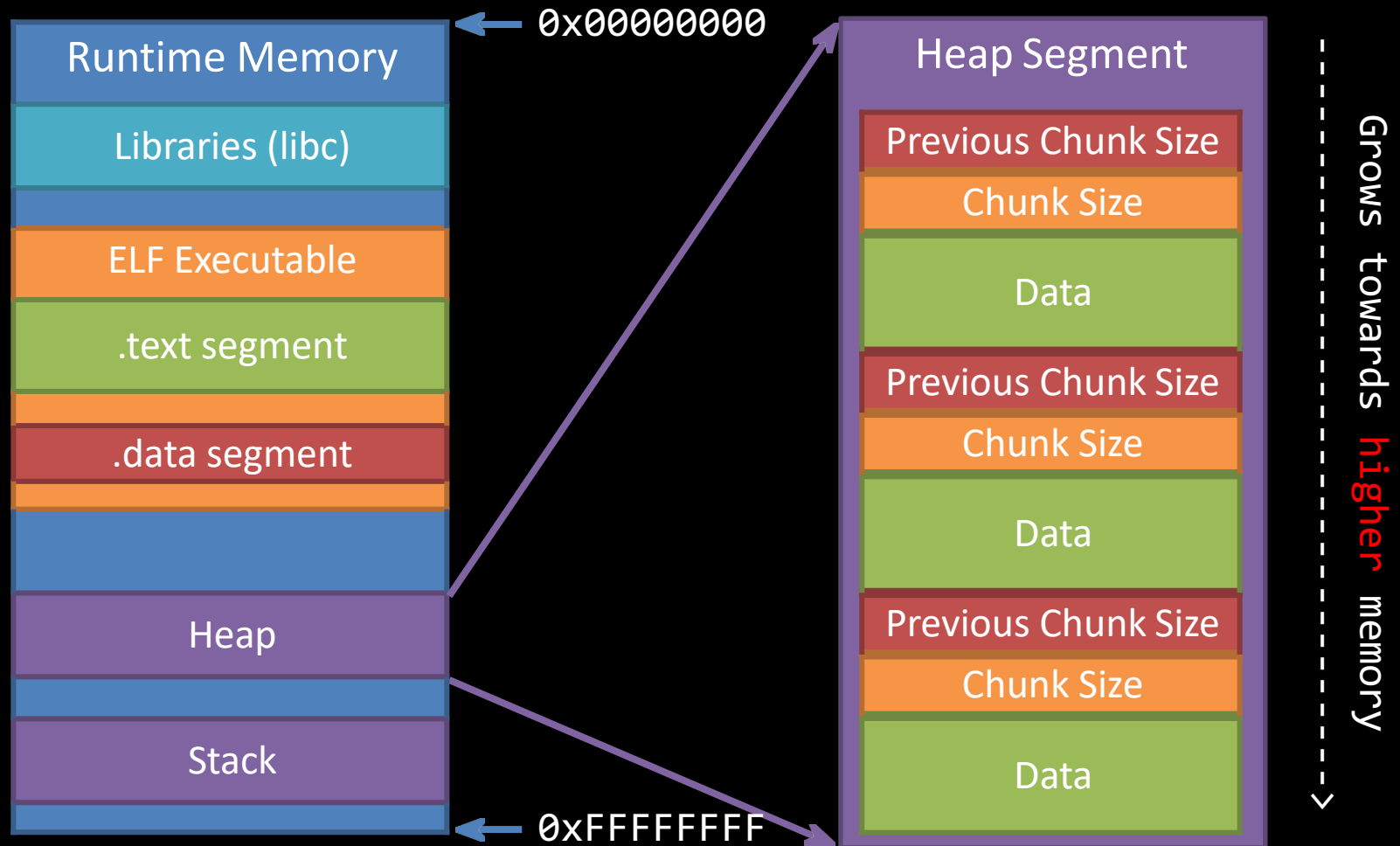
Heap Allocations



Heap Allocations

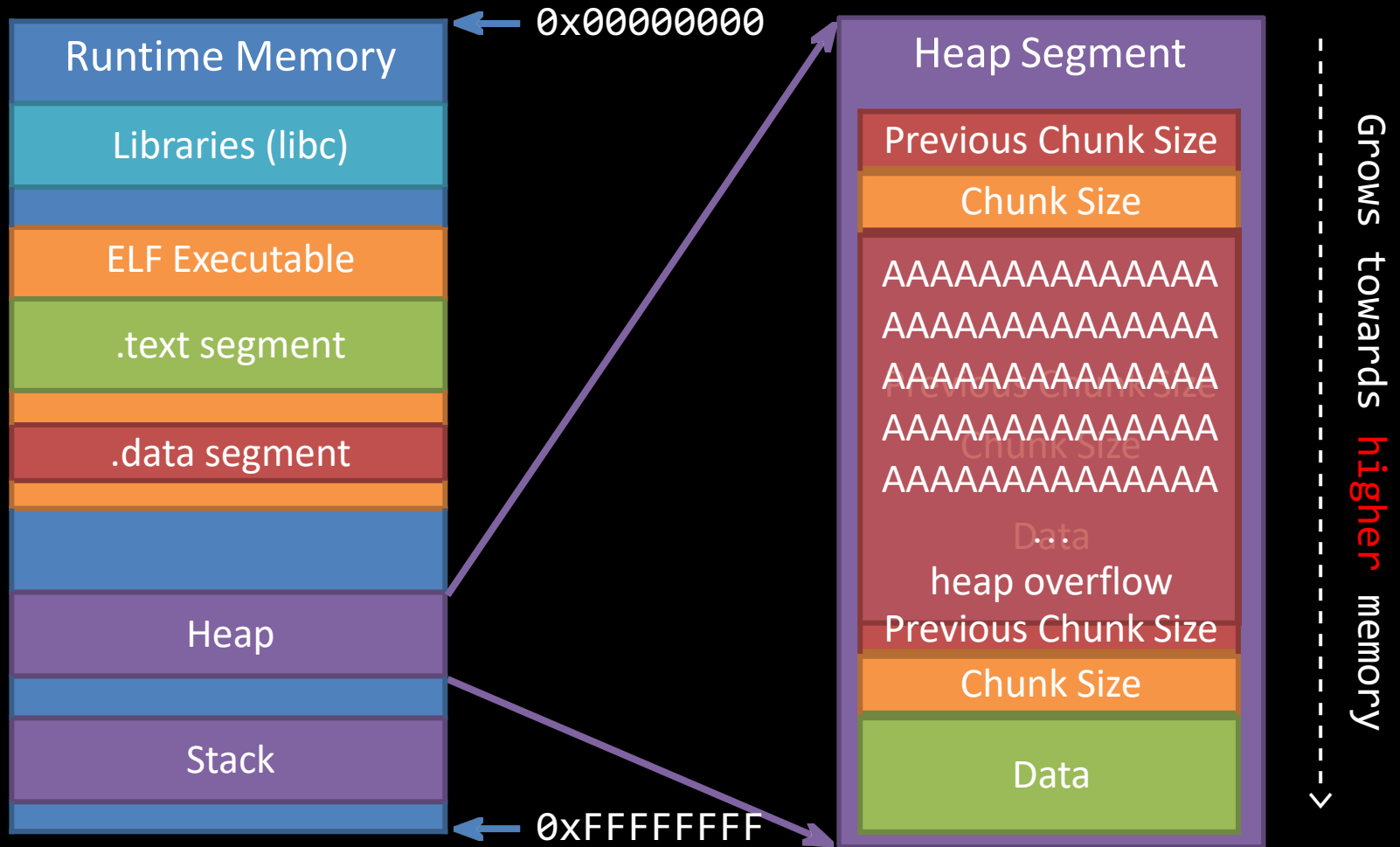


Heap Overflows



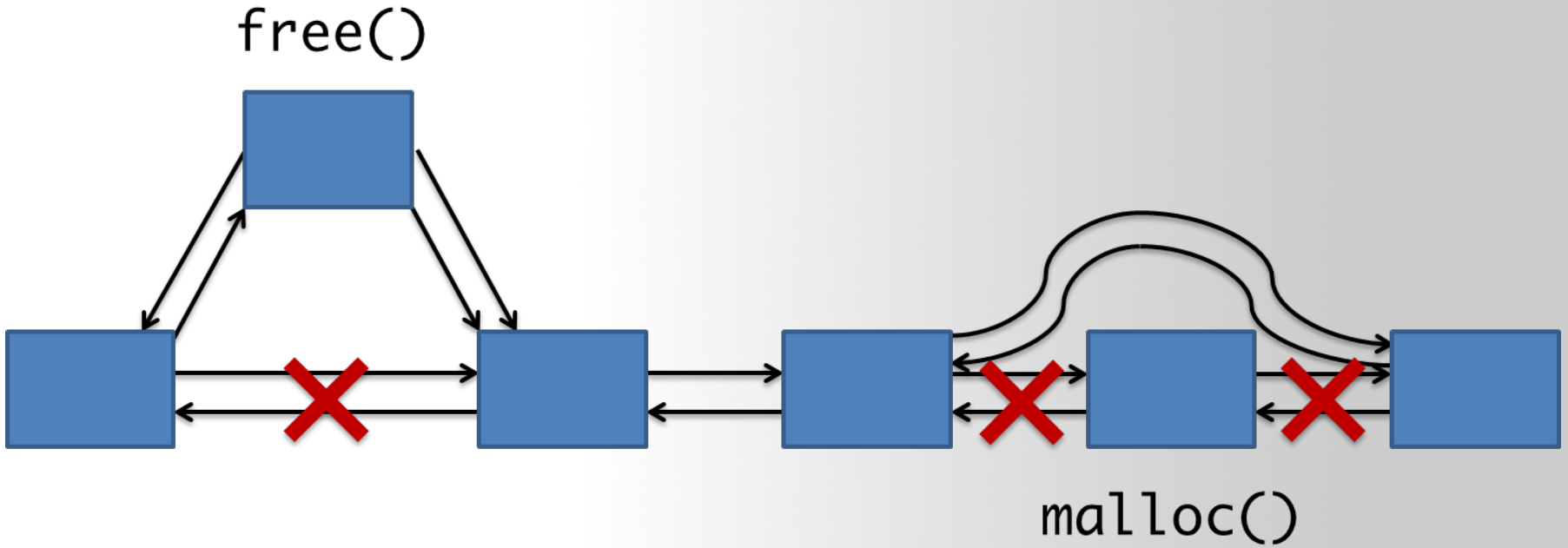
Heap Overflows

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



Unlink Attack

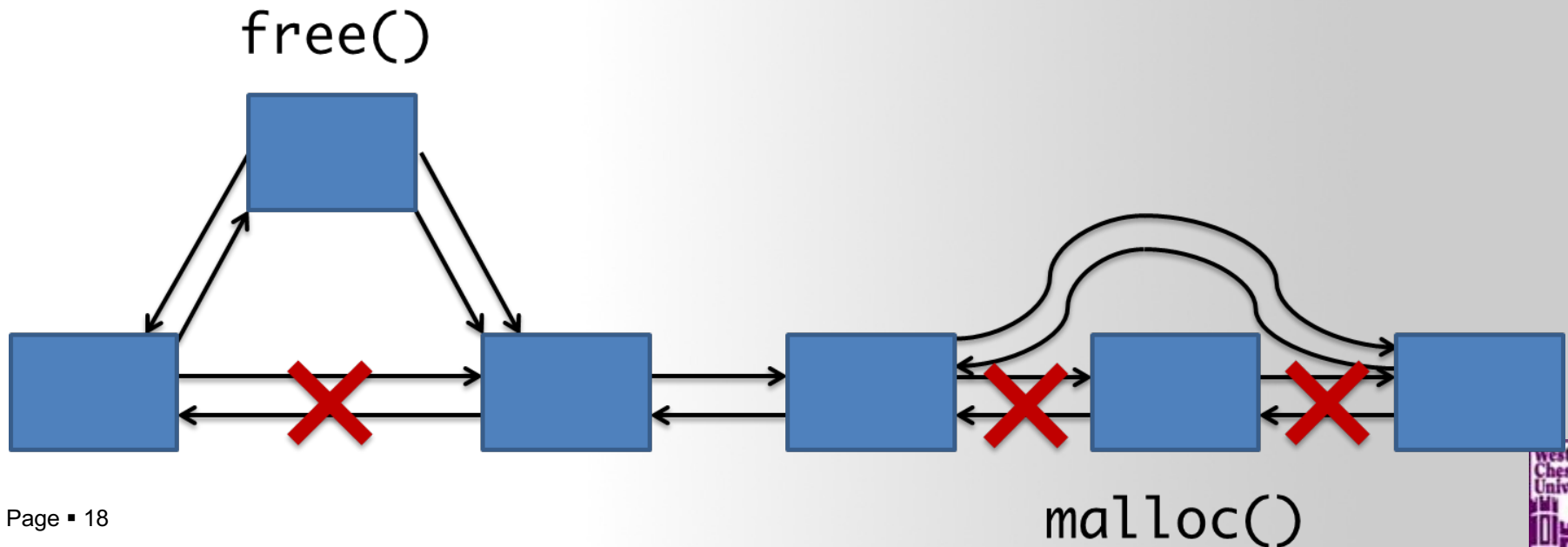
Unlink Attack: malloc & free



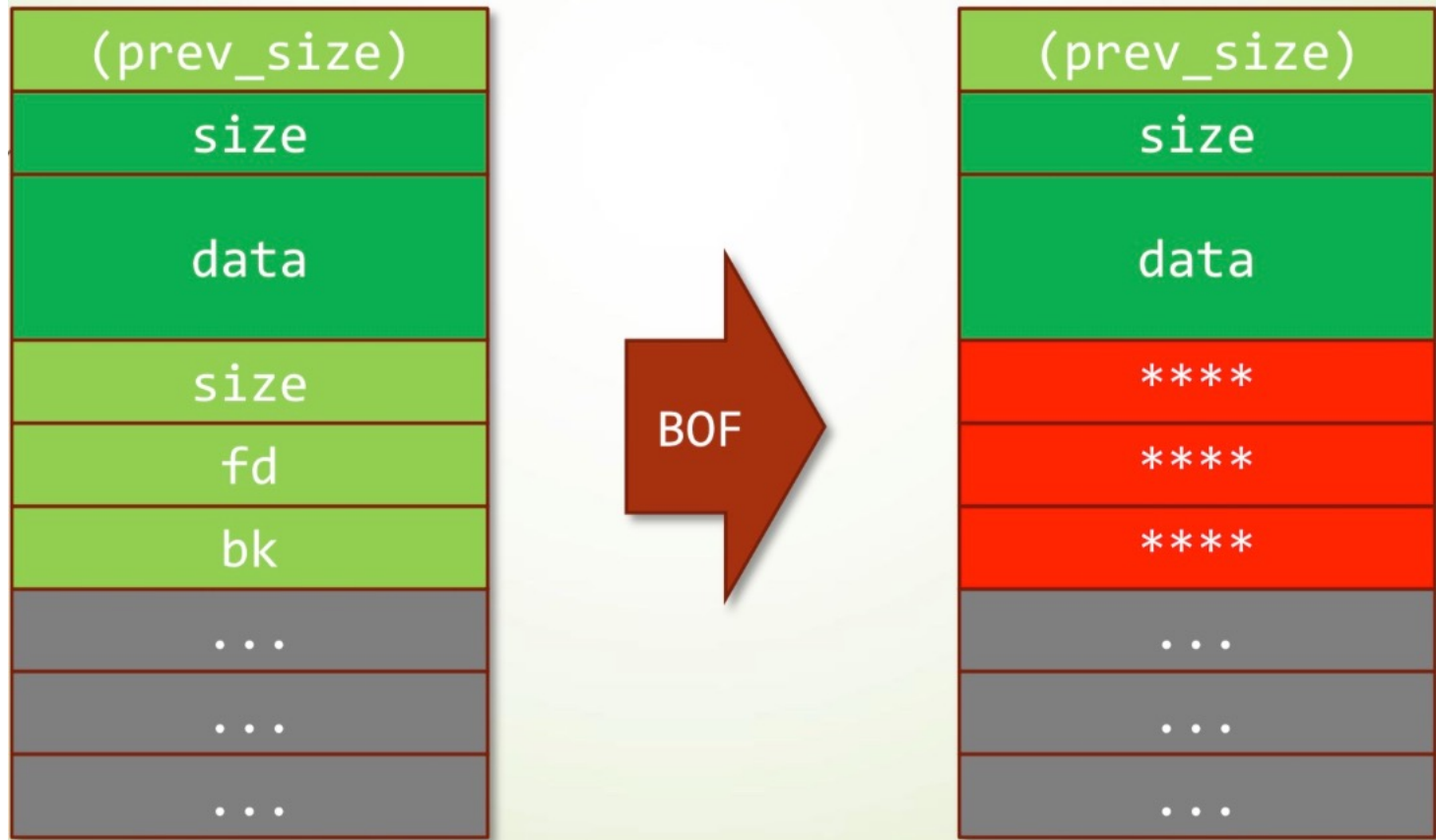
Unlink

- remove a chunk from linked list

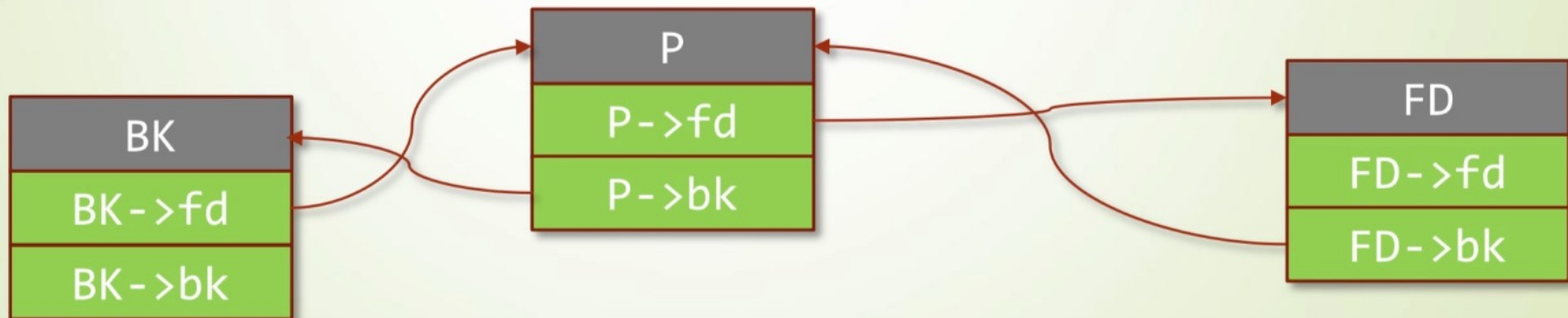
```
#define unlink(P, BK, FD) {  
    FD = P->fd;  
    BK = P->bk;  
    FD->bk = BK;  
    BK->fd = FD;  
}
```



Unlink



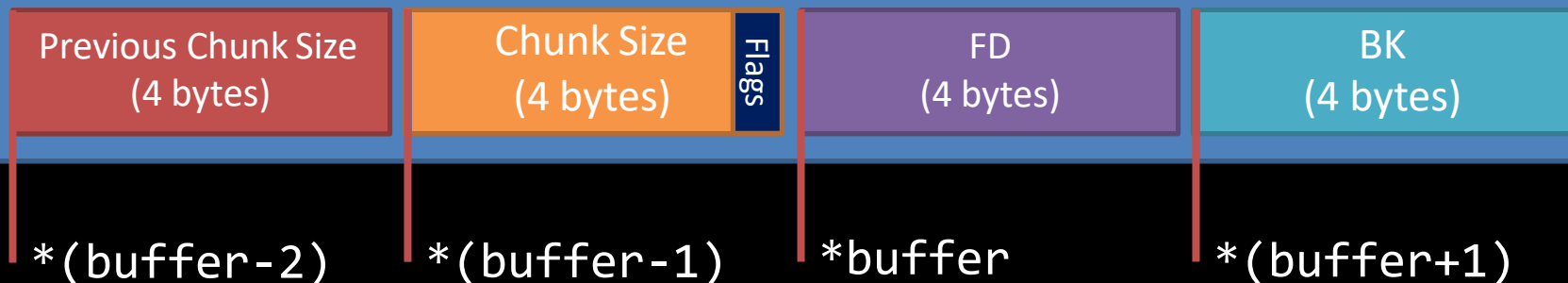
Freed Heap Chunks – Doubly linkedlist



`free(buffer);`

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

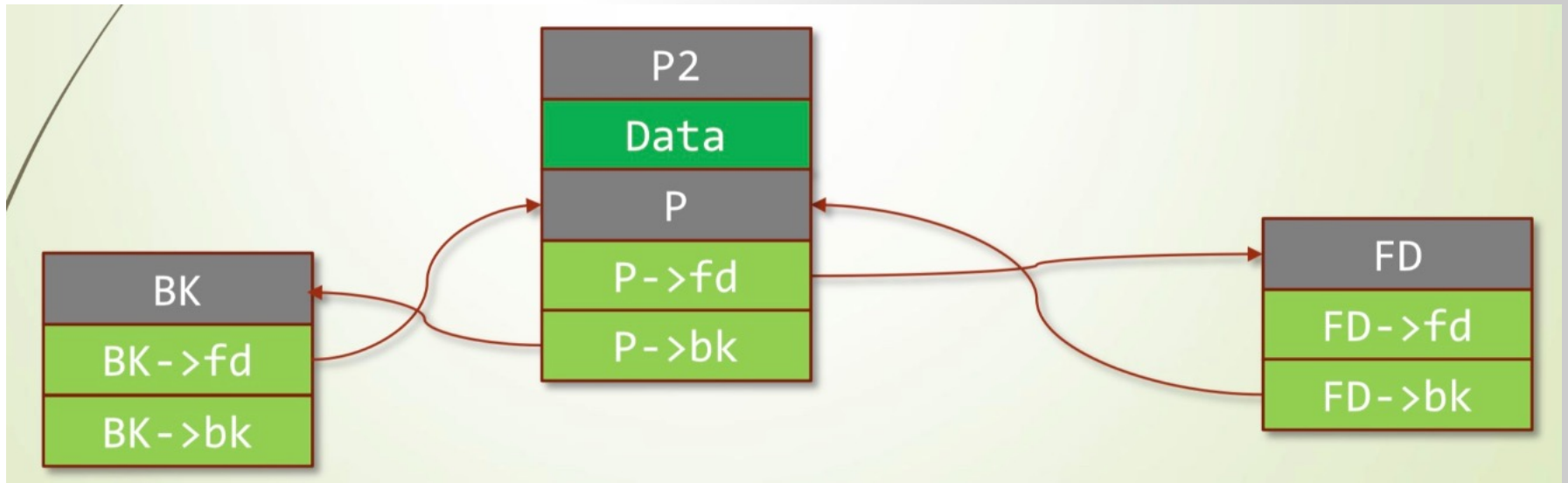
Heap Chunk (freed)



Normal Free()

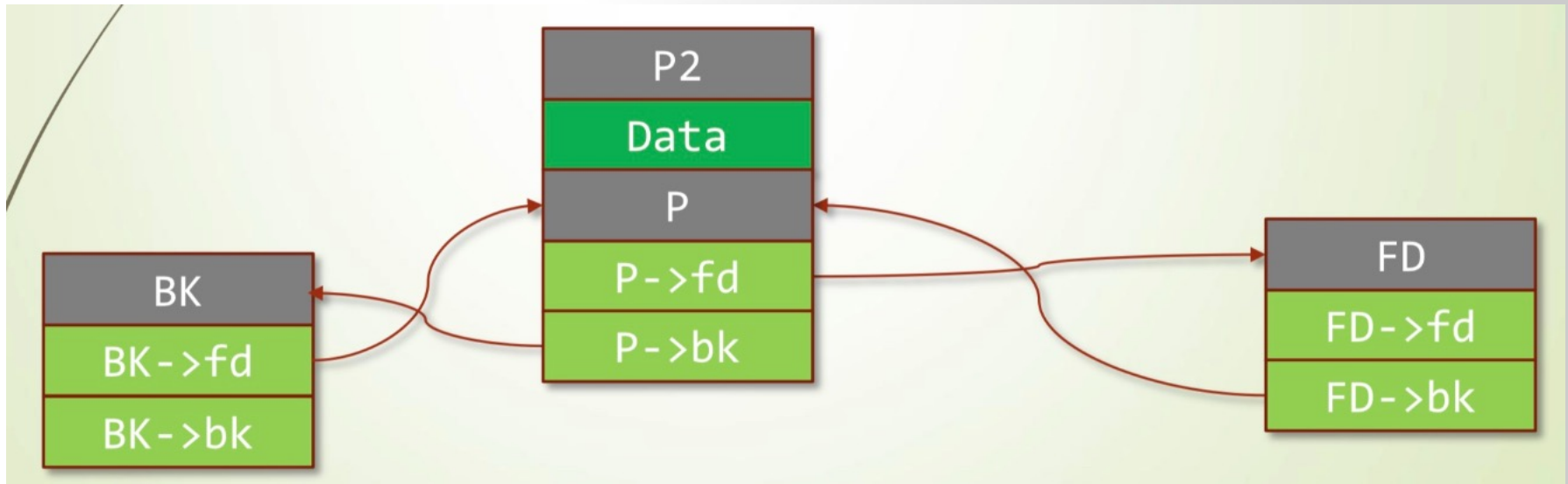
P2 is in **in-use** state, we can get **P's (freed) address** by using:

P2's address + P2's size



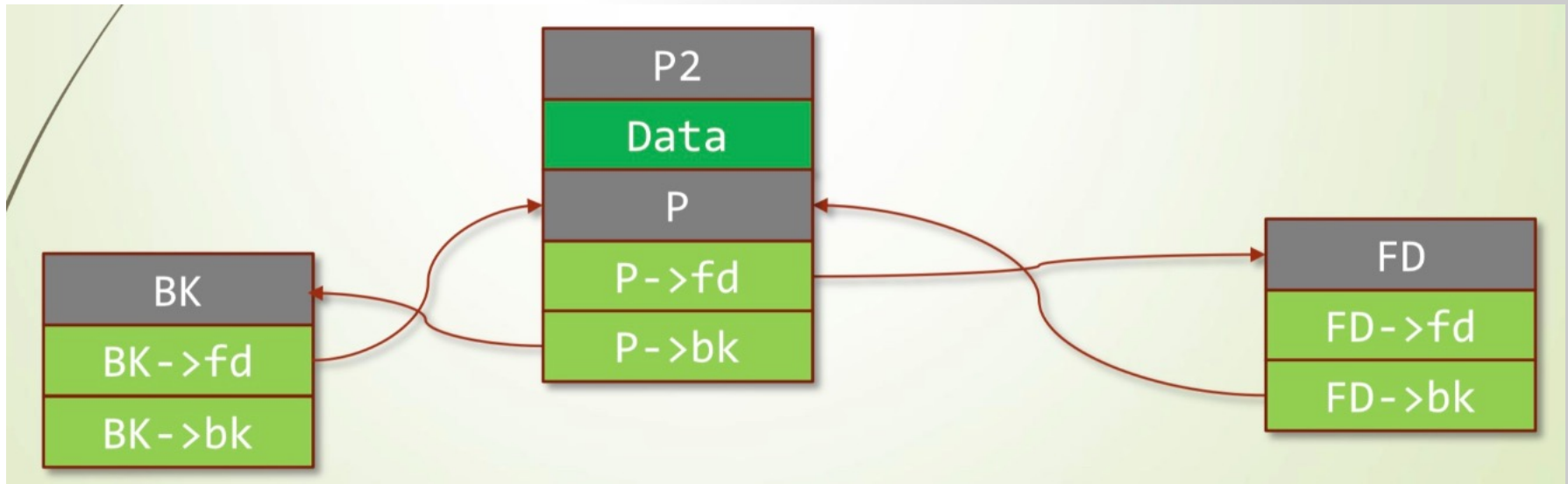
Normal Free()

- If we execute **free(P2)**. It will generate two contiguous free space
- OS will automatically combined two continues free chunks and **generate one big free chunk**
 - --> P will be **temporarily deleted from the doubly linked-list**



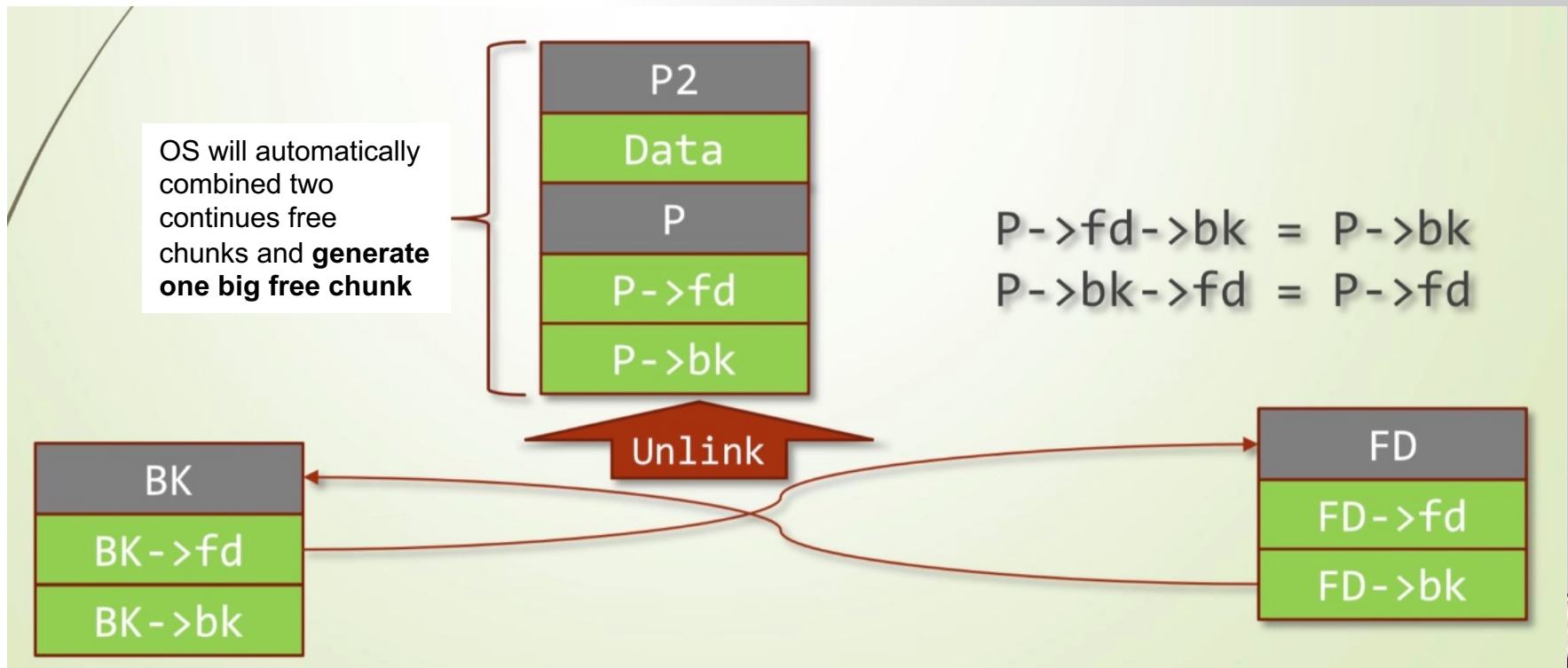
Normal Free()

- If we execute **free(P2)**. It will generate two contiguous free space
- OS will automatically combined two continues free chunks and **generate one big free chunk**
 - --> P will be **temporarily deleted from the doubly linked-list**



Normal Free()

- If we execute **free(P2)**. It will generate two contiguous free space
- OS will automatically combined two continues free chunks and **generate one big free chunk**
 - --> P will be **temporarily deleted from the doubly linked-list**
 - This is done inside free()
 - **When you free(P2). P will also get unlinked!!**

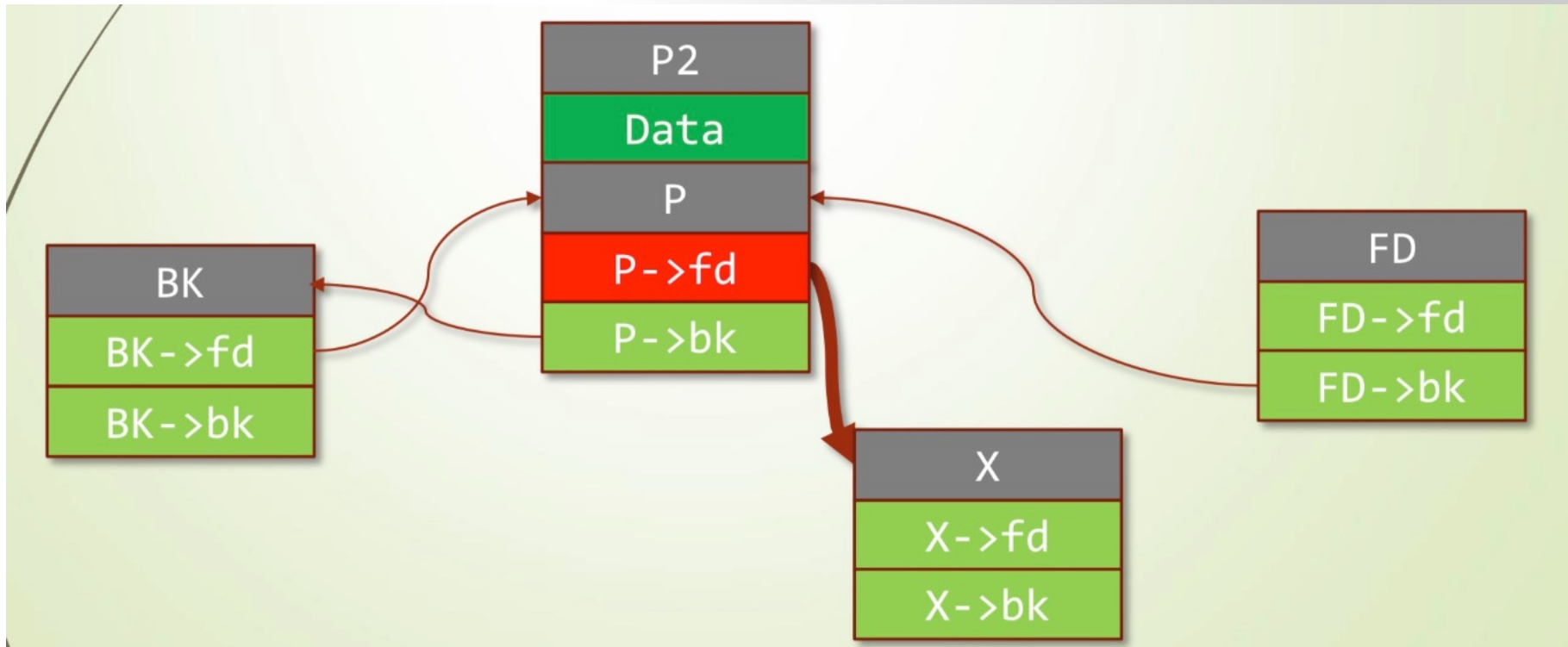


Consolidate Free Chunk

```
1  /* consolidate backward */
2  if (!prev_inuse(p)) {
3      prevsize = p->prev_size;
4      size += prevsize;
5      p = chunk_at_offset(p, -((long) prevsize)); /* previous chunk */
6      unlink(p, bck, fwd);
7  }
8
9  if (nextchunk != av->top) {
10     /* get and clear inuse bit */
11     nextinuse = inuse_bit_at_offset(nextchunk, nextsize);
12     /* consolidate forward */
13     if (!nextinuse) {
14         unlink(nextchunk, bck, fwd);
15         size += nextsize;
16     }
17 }
```

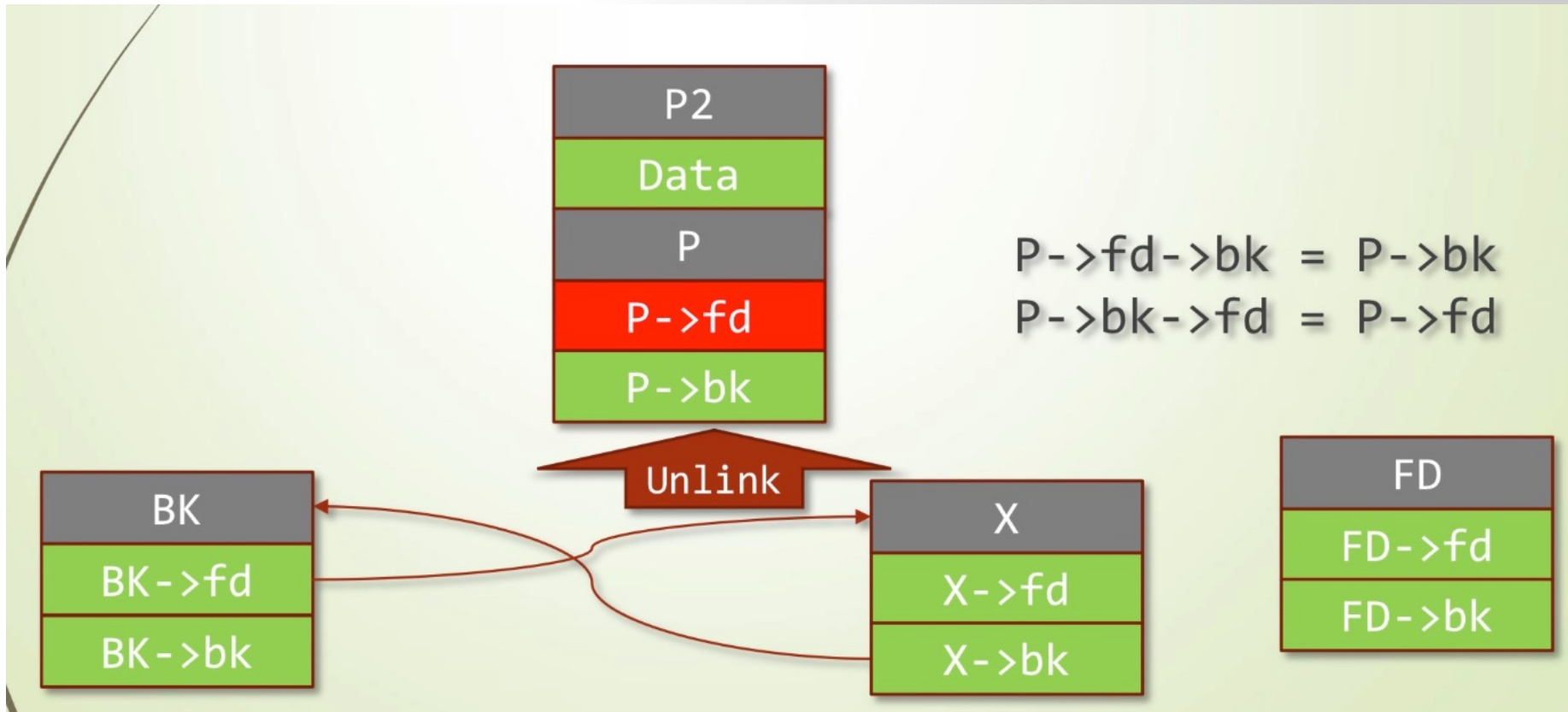
Unlink Attack

- Exploit P2, overwrite fd
 - Let it pointing to X



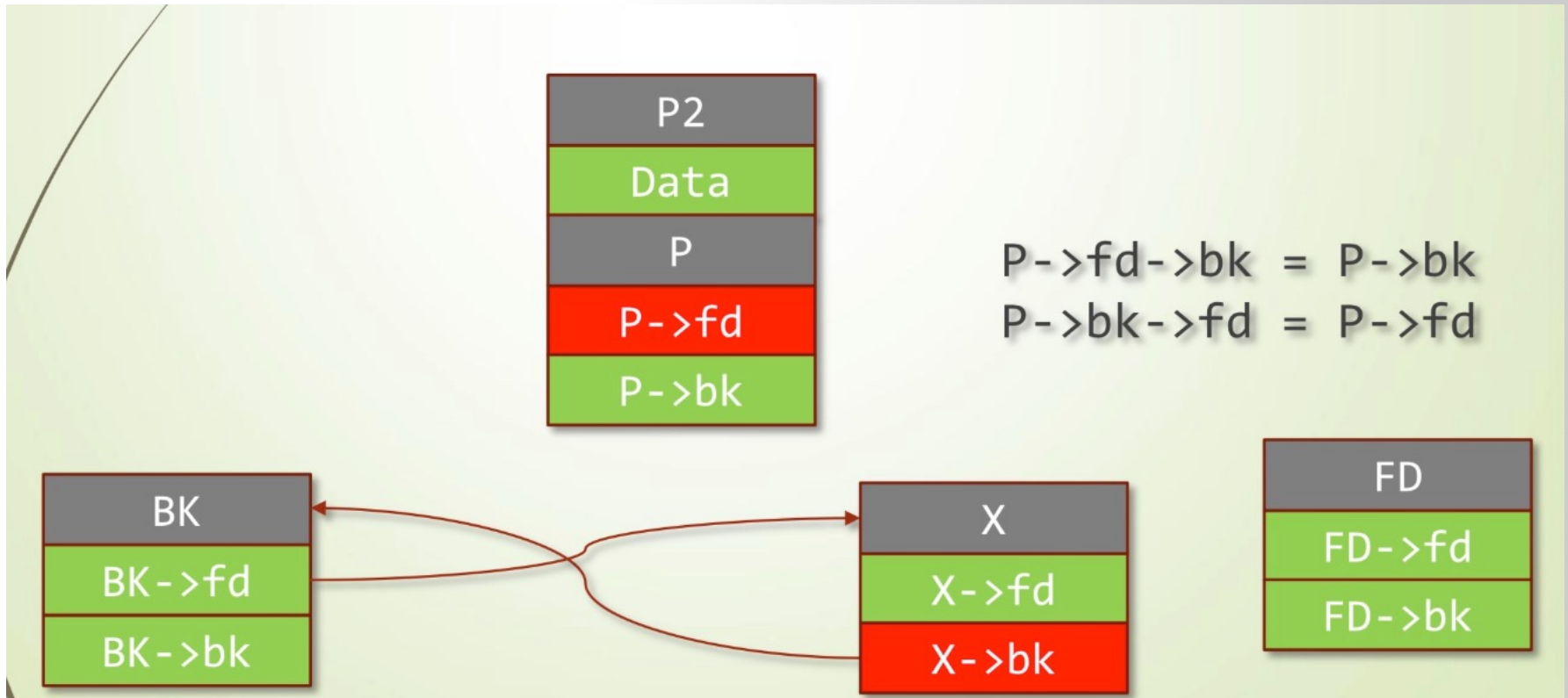
Unlink Attack

- When you free(P2). P will also be unlinked.



Unlink Attack

Then, BK and X will be connected.



Unlink Attack Example: DEFCON 2014 CTF-Baby's First-heap

```
Welcome to your first heap overflow...  
I am going to allocate 20 objects...  
Using Dougle Lee Allocator 2.6.1...  
Goodluck!
```

```
Exit function pointer is at 804C8AC address.
```

```
[ALLOC][loc=804D008][size=1246]  
[ALLOC][loc=804D4F0][size=1121]  
[ALLOC][loc=804D958][size=947]  
[ALLOC][loc=804DD10][size=741]  
[ALLOC][loc=804E000][size=706]  
[ALLOC][loc=804E2C8][size=819]  
[ALLOC][loc=804E600][size=673]  
[ALLOC][loc=804E8A8][size=1004]  
[ALLOC][loc=804EC98][size=952]  
[ALLOC][loc=804F058][size=755]  
[ALLOC][loc=804F350][size=260]  
[ALLOC][loc=804F458][size=877]  
[ALLOC][loc=804F7D0][size=1245]  
[ALLOC][loc=804FCB8][size=1047]  
[ALLOC][loc=80500D8][size=1152]  
[ALLOC][loc=8050560][size=1047]  
[ALLOC][loc=8050980][size=1059]  
[ALLOC][loc=8050DA8][size=906]  
[ALLOC][loc=8051138][size=879]  
[ALLOC][loc=80514B0][size=823]  
Write to object [size=260]:
```

DEFCON 2014 CTF-Baby's First-heap

```
Welcome to your first heap overflow...  
I am going to allocate 20 objects...  
Using Dougle Lee Allocator 2.6.1...  
Goodluck!
```

```
Exit function pointer is at 804C8AC address.
```

```
[ALLOC][loc=804D008][size=1246]  
[ALLOC][loc=804D4F0][size=1121]  
[ALLOC][loc=804D958][size=947]  
[ALLOC][loc=804DD10][size=741]  
[ALLOC][loc=804E000][size=706]  
[ALLOC][loc=804E2C8][size=819]  
[ALLOC][loc=804E600][size=673]  
[ALLOC][loc=804E8A8][size=1004]  
[ALLOC][loc=804EC98][size=952]  
[ALLOC][loc=804F058][size=755]  
[ALLOC][loc=804F350][size=260]  
[ALLOC][loc=804F458][size=877]  
[ALLOC][loc=804F7D0][size=1245]  
[ALLOC][loc=804FCB8][size=1047]  
[ALLOC][loc=80500D8][size=1152]  
[ALLOC][loc=8050560][size=1047]  
[ALLOC][loc=8050980][size=1059]  
[ALLOC][loc=8050DA8][size=906]  
[ALLOC][loc=8051138][size=879]  
[ALLOC][loc=80514B0][size=823]
```

```
Write to object [size=260]:
```

DEFCON 2014 CTF-Baby's First-heap

```
Welcome to your first heap overflow...  
I am going to allocate 20 objects...  
Using Dougle Lee Allocator 2.6.1...  
Goodluck!
```

```
Exit function pointer is at 804C8AC address.
```

```
[ALLOC][loc=804D008][size=1246]  
[ALLOC][loc=804D4F0][size=1121]  
[ALLOC][loc=804D958][size=947]  
[ALLOC][loc=804DD10][size=741]  
[ALLOC][loc=804E000][size=706]  
[ALLOC][loc=804E2C8][size=819]  
[ALLOC][loc=804E600][size=673]  
[ALLOC][loc=804E8A8][size=1004]  
[ALLOC][loc=804EC98][size=952]  
[ALLOC][loc=804F058][size=755]  
[ALLOC][loc=804F350][size=260]  
[ALLOC][loc=804F458][size=877]  
[ALLOC][loc=804F7D0][size=1245]  
[ALLOC][loc=804FCB8][size=1047]  
[ALLOC][loc=80500D8][size=1152]  
[ALLOC][loc=8050560][size=1047]  
[ALLOC][loc=8050980][size=1059]  
[ALLOC][loc=8050DA8][size=906]  
[ALLOC][loc=8051138][size=879]  
[ALLOC][loc=80514B0][size=823]
```

```
Write to object [size=260]:
```



DEFCON 2014 CTF-Baby's First-heap

```
Welcome to your first heap overflow...
I am going to allocate 20 objects...
Using Dougle Lee Allocator 2.6.1...
Goodluck!
```

```
Exit function pointer is at 804C8AC address.
```

```
[ALLOC][loc=804D008][size=1246]
[ALLOC][loc=804D4F0][size=1121]
[ALLOC][loc=804D958][size=947]
[ALLOC][loc=804DD10][size=741]
[ALLOC][loc=804E000][size=706]
[ALLOC][loc=804E2C8][size=819]
[ALLOC][loc=804E600][size=673]
[ALLOC][loc=804E8A8][size=1004]
[ALLOC][loc=804EC98][size=952]
[ALLOC][loc=804F058][size=755]
[ALLOC][loc=804F350][size=260]Array[10]
[ALLOC][loc=804F458][size=877]Array[11]
[ALLOC][loc=804F7D0][size=1245]
[ALLOC][loc=804FCB8][size=1047]
[ALLOC][loc=80500D8][size=1152]
[ALLOC][loc=8050560][size=1047]
[ALLOC][loc=8050980][size=1059]
[ALLOC][loc=8050DA8][size=906]
[ALLOC][loc=8051138][size=879]
[ALLOC][loc=80514B0][size=823]
Write to object [size=260]:
```



memcpy() lead to heap overflow

Array[11] is in in-use state:

Let's overflow Array[10]

And set Array[11] to freed state

DEFCON 2014 CTF-Baby's First-heap

```
Welcome to your first heap overflow...
I am going to allocate 20 objects...
Using Dougle Lee Allocator 2.6.1...
Goodluck!
```

```
Exit function pointer is at 804C8AC address.
```

```
[ALLOC][loc=804D008][size=1246]
[ALLOC][loc=804D4F0][size=1121]
[ALLOC][loc=804D958][size=947]
[ALLOC][loc=804DD10][size=741]
[ALLOC][loc=804E000][size=706]
[ALLOC][loc=804E2C8][size=819]
[ALLOC][loc=804E600][size=673]
[ALLOC][loc=804E8A8][size=1004]
[ALLOC][loc=804EC98][size=952]
[ALLOC][loc=804F058][size=755]
[ALLOC][loc=804F350][size=260]Array[10]
[ALLOC][loc=804F458][size=877]Array[11]
[ALLOC][loc=804F7D0][size=1245]
[ALLOC][loc=804FCB8][size=1047]
[ALLOC][loc=80500D8][size=1152]
[ALLOC][loc=8050560][size=1047]
[ALLOC][loc=8050980][size=1059]
[ALLOC][loc=8050DA8][size=906]
[ALLOC][loc=8051138][size=879]
[ALLOC][loc=80514B0][size=823]
Write to object [size=260]:
```



memcpy() lead to heap overflow

Array[11] is in in-use state:

Let's overflow Array[10]

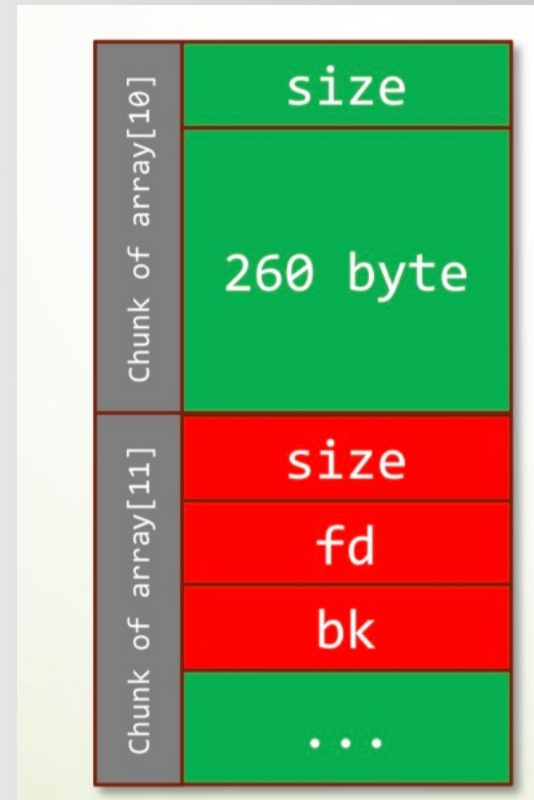
And set Array[11] to freed state

DEFCON 2014 CTF-Baby's First-heap

```
Welcome to your first heap overflow...
I am going to allocate 20 objects...
Using Dougle Lee Allocator 2.6.1...
Goodluck!
```

```
Exit function pointer is at 804C8AC address.
```

```
[ALLOC][loc=804D008][size=1246]
[ALLOC][loc=804D4F0][size=1121]
[ALLOC][loc=804D958][size=947]
[ALLOC][loc=804DD10][size=741]
[ALLOC][loc=804E000][size=706]
[ALLOC][loc=804E2C8][size=819]
[ALLOC][loc=804E600][size=673]
[ALLOC][loc=804E8A8][size=1004]
[ALLOC][loc=804EC98][size=952]
[ALLOC][loc=804F058][size=755]
[ALLOC][loc=804F350][size=260]Array[10]
[ALLOC][loc=804F458][size=877]Array[11]
[ALLOC][loc=804F7D0][size=1245]
[ALLOC][loc=804FCB8][size=1047]
[ALLOC][loc=80500D8][size=1152]
[ALLOC][loc=8050560][size=1047]
[ALLOC][loc=8050980][size=1059]
[ALLOC][loc=8050DA8][size=906]
[ALLOC][loc=8051138][size=879]
[ALLOC][loc=80514B0][size=823]
Write to object [size=260]:
```



memcpy() lead to heap overflow

Array[11] is in in-use state:

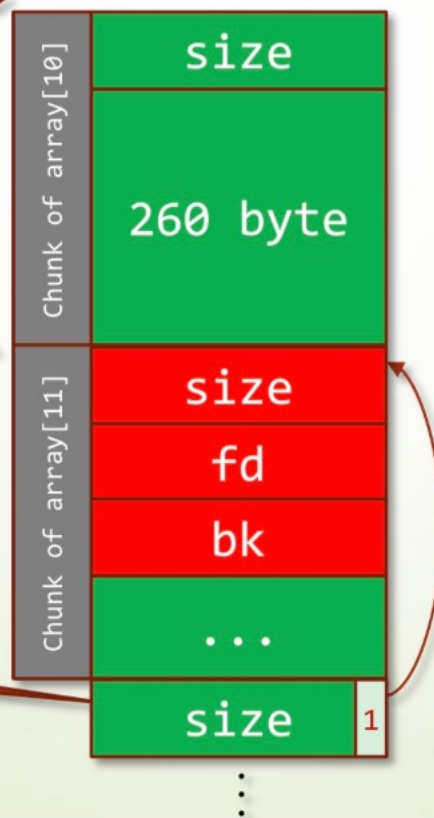
Let's overflow Array[10]

And set Array[11] to freed state

DEFCON 2014 CTF-Baby's First-heap

Check **PREV_INUSE**
from the previous chunk

Check **PREV_INUSE**
bit of the next chunk →
→ Array[12]



1: In-use
0: Freed

Heap Chunk (freed)

Previous Chunk Size
(4 bytes)

Chunk Size
(4 bytes)

Flags

FD
(4 bytes)

BK
(4 bytes)

*(buffer-2)

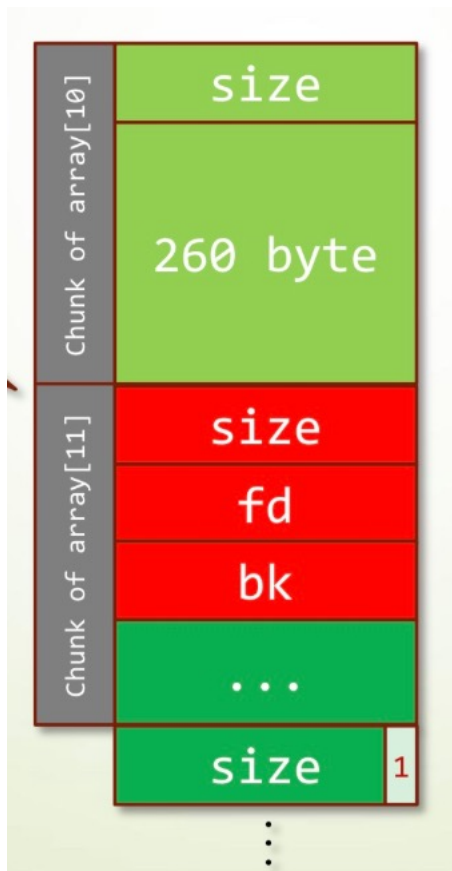
*(buffer-1)

*buffer

*(buffer+1)

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

DEFCON 2014 CTF-Baby's First-heap



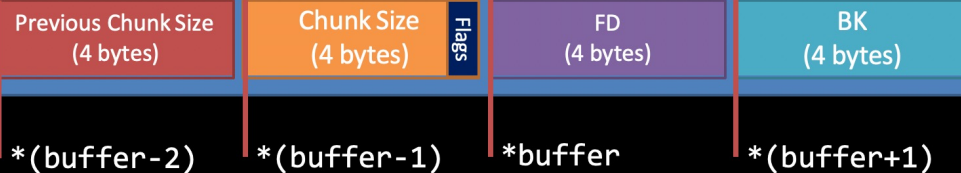
How to find the address of **Chunk Size** for both Array[11] and Array[12]?

$\text{Addr_Array}[10] + \text{chunk_size}[10] == \text{addr_Array}[11]$
 $\text{Addr_Array}[11] + \text{chunk_size}[11] == \text{addr_Array}[12]$

1: In-use
0: Freed

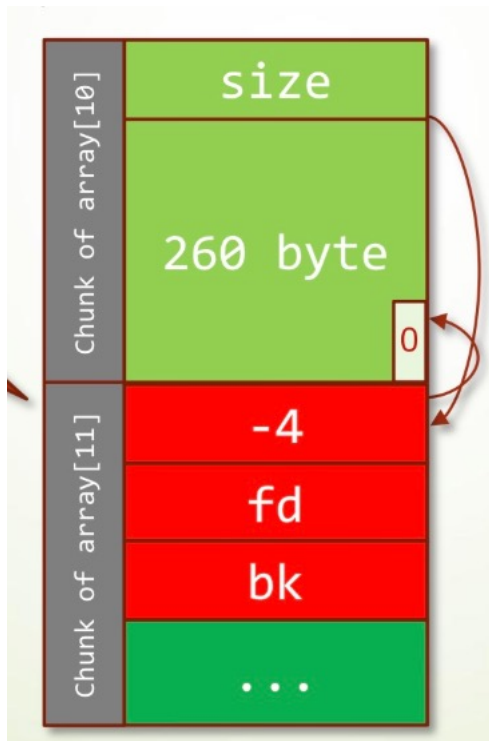


Heap Chunk (freed)



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

DEFCON 2014 CTF-Baby's First-heap



If we change Array[11]'s size to -4

Aka. 0xffffffffc

Then, Array[10] thought array[11] is in freed state

Then in order to find Array[12], computer use:

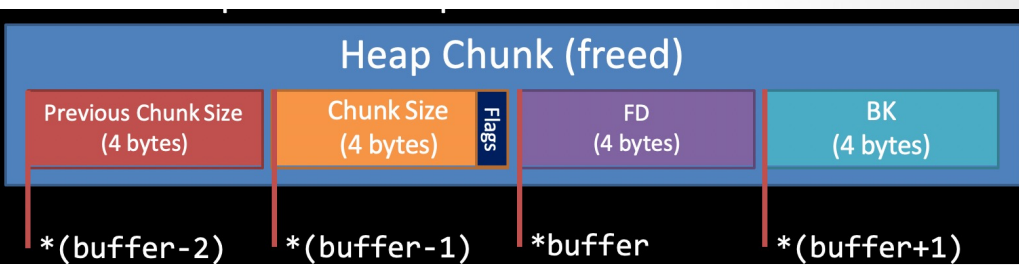
$\text{Addr_Array}[11] + \text{chunk_size}[11] == \text{addr_Array}[12]$

Where chunk_size[11] is negative 4.

So the addr_array[12] is pointing to the end of Array[10]

And it's PREV_INUSE is 0. So system think Array[11] is in freed state

1: In-use
0: Freed



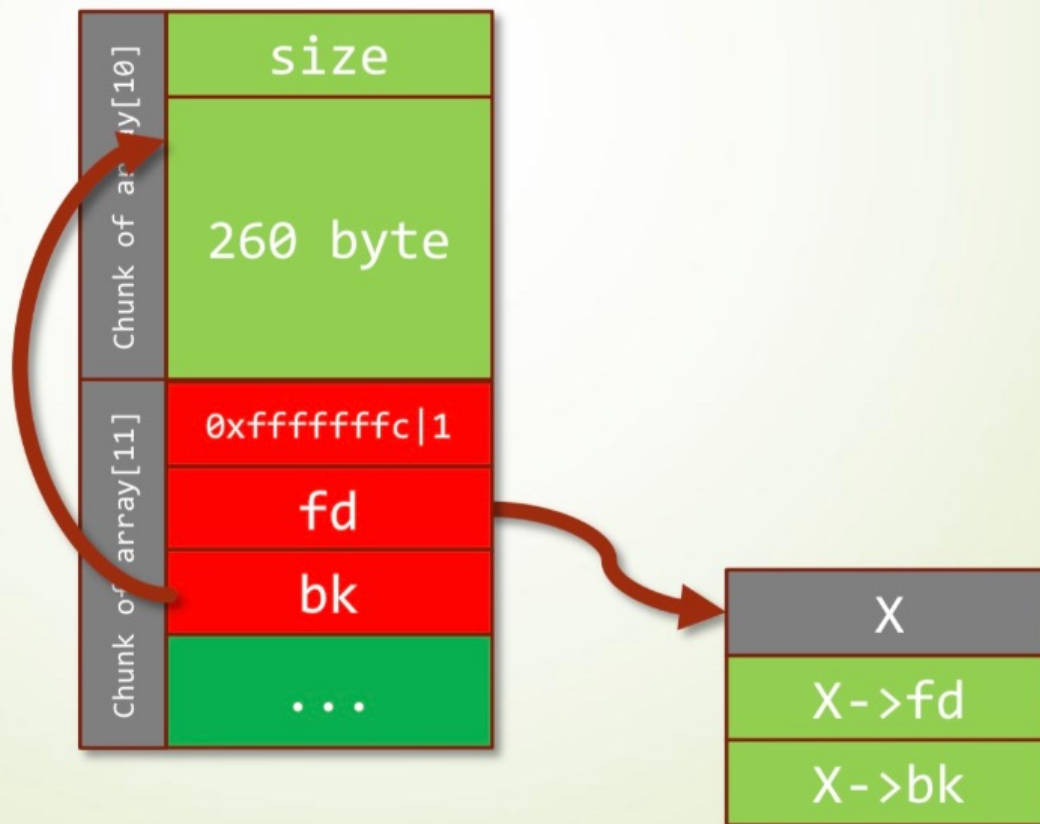
```
#define PREV_INUSE 0x1
#define IS_MMAPPED 0x2
#define NON_MAIN_ARENA 0x4
```

DEFCON 2014 CTF-Baby's First-heap

P : Array[11]

➡ P->fd->bk = P->bk

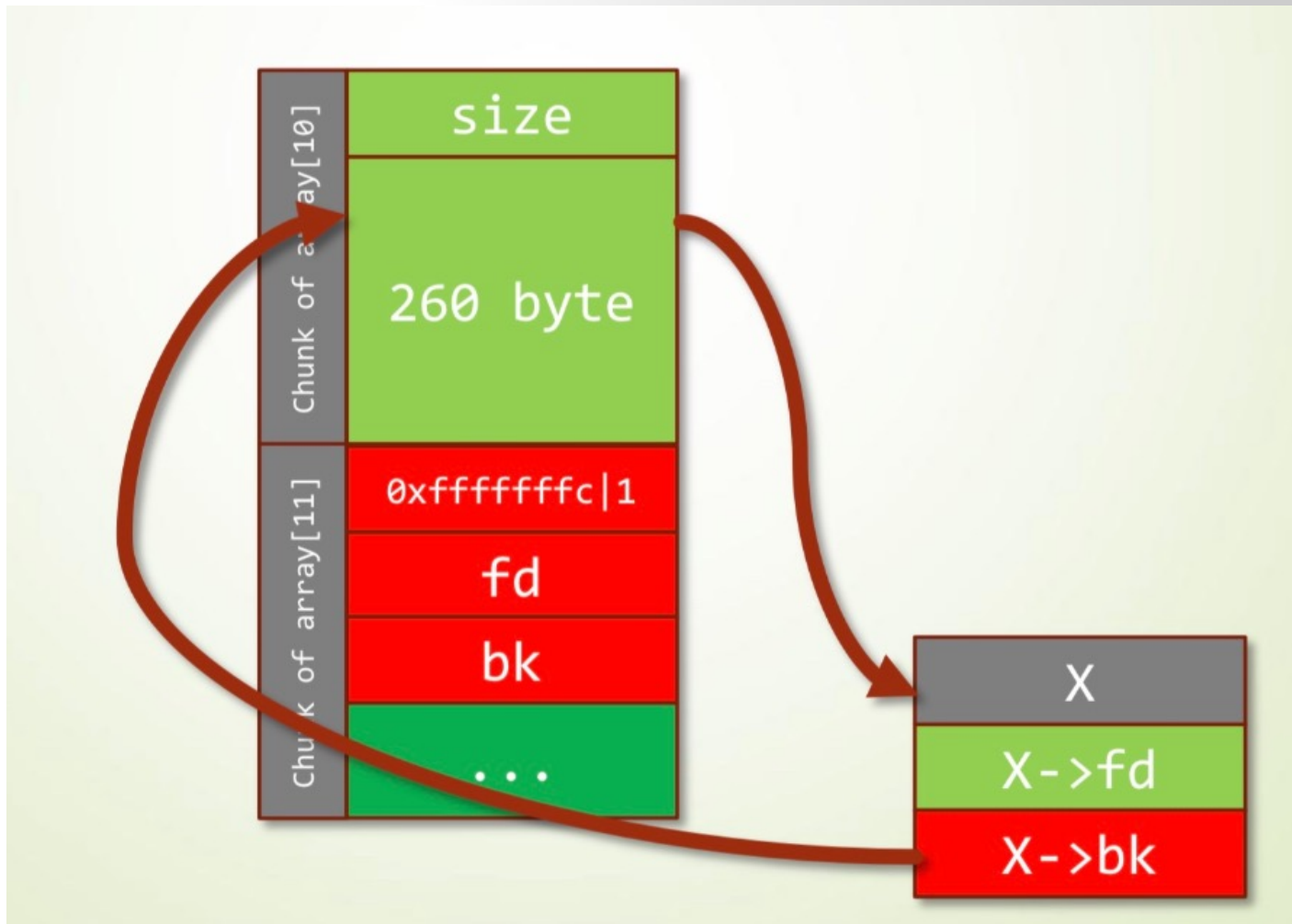
➡ P->bk->fd = P->fd



Launch unlink attack!

DEFCON 2014 CTF-Baby's First-heap

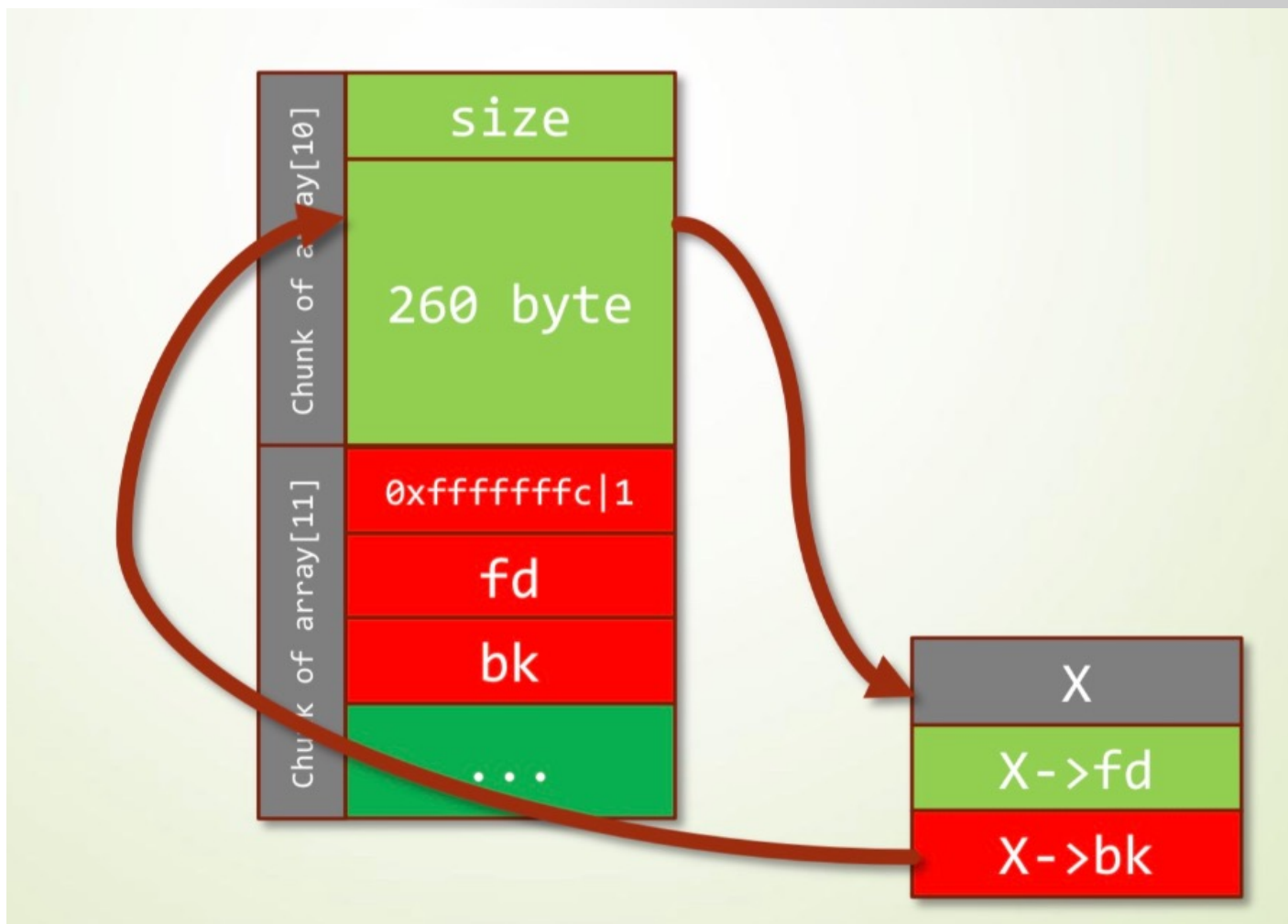
P : Array[11]



Launch unlink attack!

DEFCON 2014 CTF-Baby's First-heap

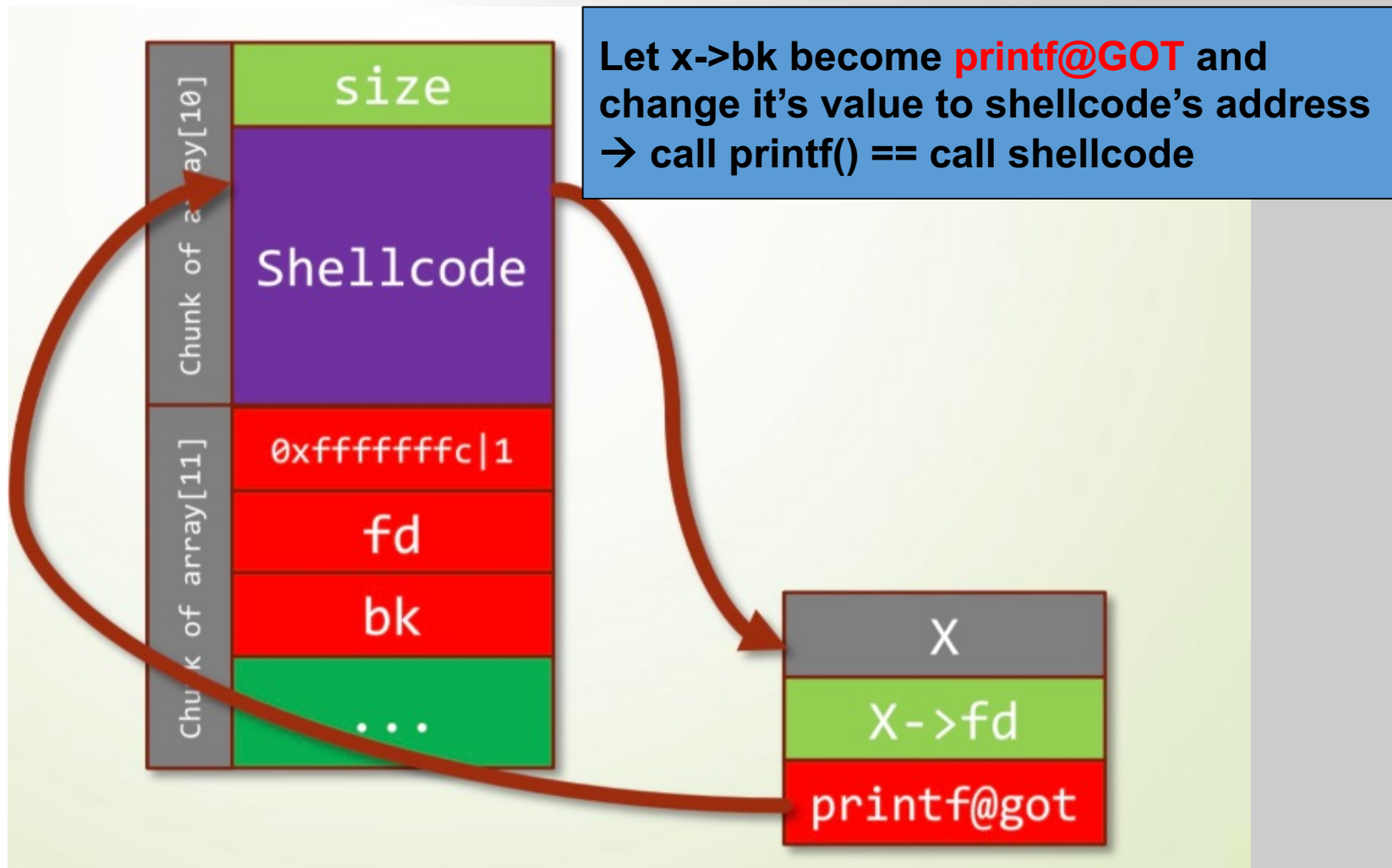
P : Array[11]



Let x->bk become printf@GOT

DEFCON 2014 CTF-Baby's First-heap

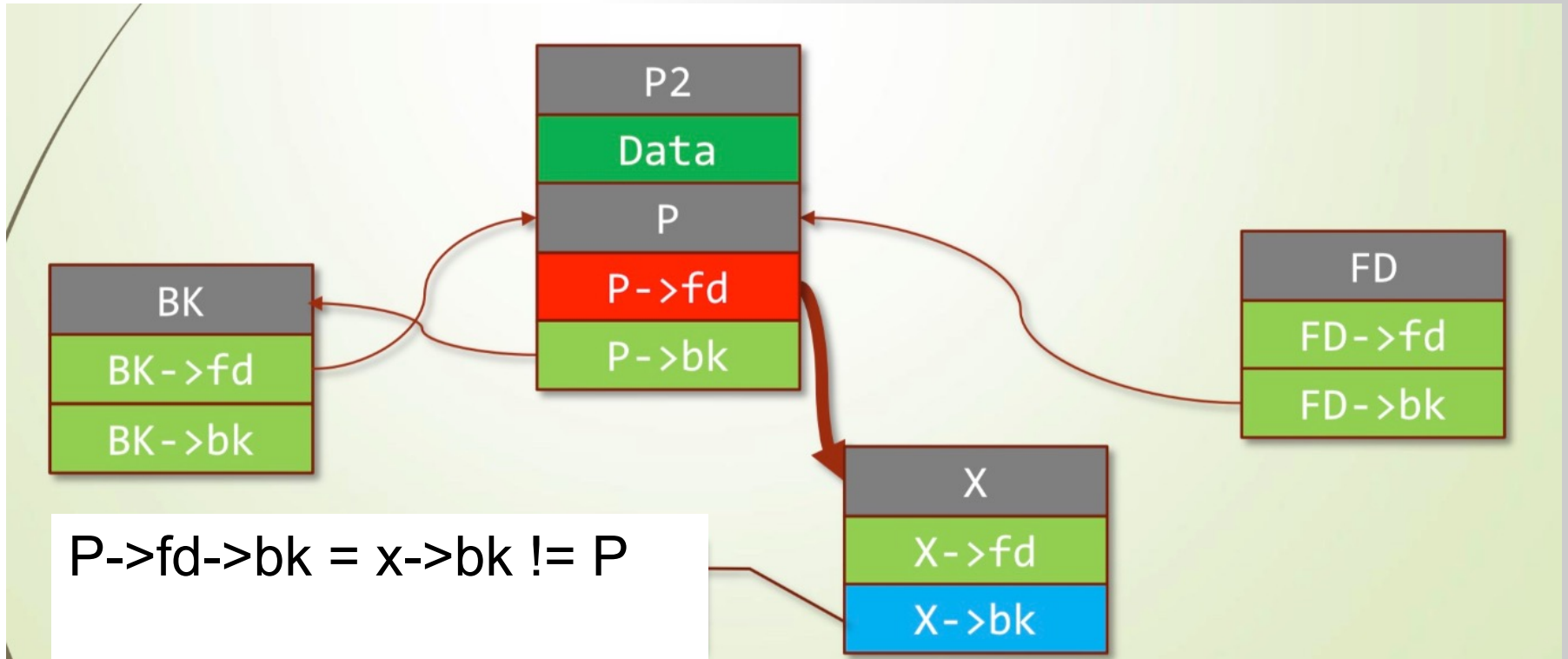
P : Array[11]



Write shellcode into array[10]

Unlink Attack – After 2004

After 2004, libc updates it's free() function. Added the following judgement:

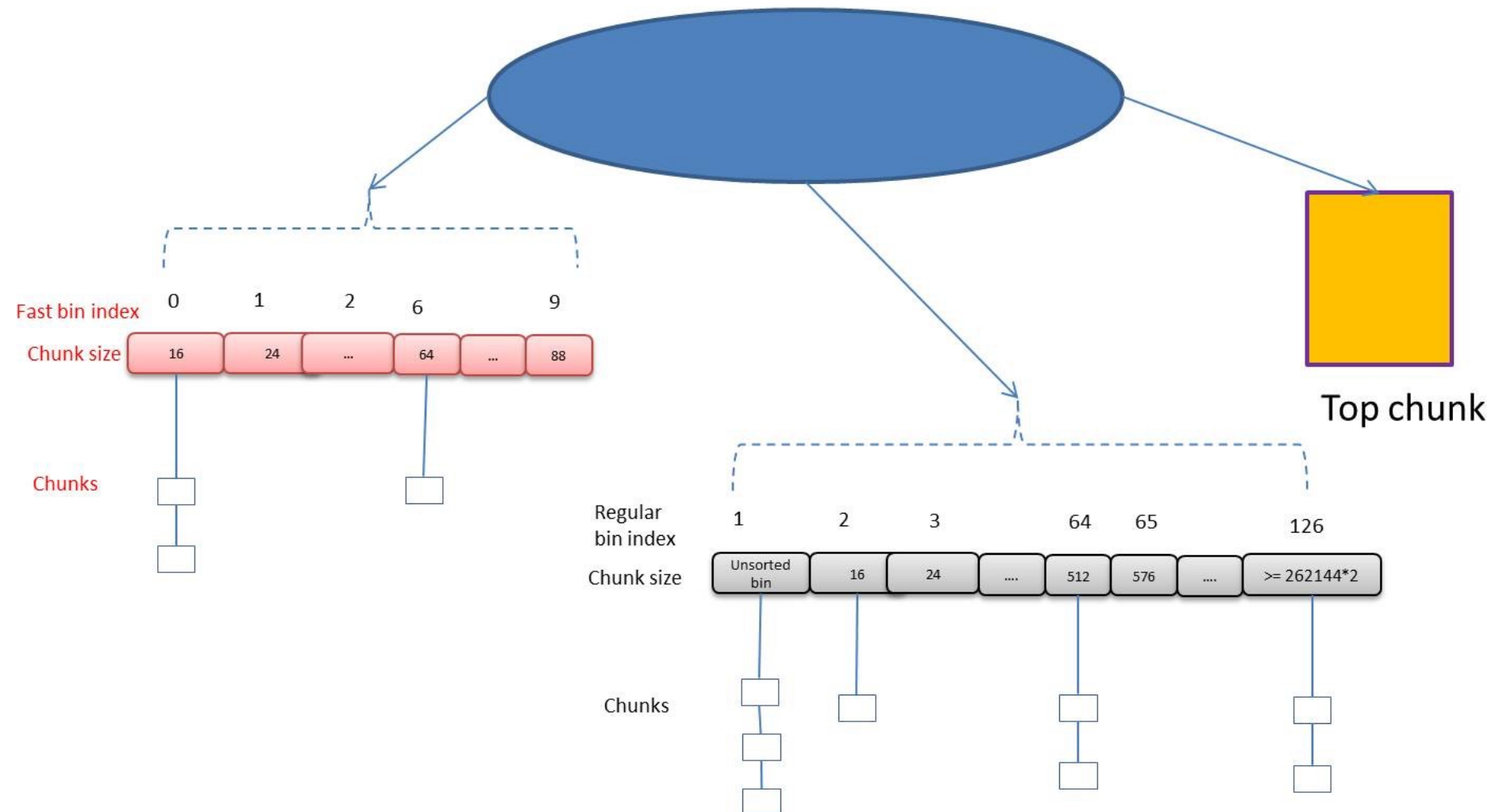
$$P \rightarrow fd \rightarrow bk == P$$
$$P \rightarrow bk \rightarrow fd == P$$


Unlink Attack – After 2004

```
#define unlink(P, BK, FD) {                                \
    FD = P->fd;                                              \
    BK = P->bk;                                              \
    if (FD->bk != P || BK->fd != P)                          \
        malloc_printerr (check_action, "corrupted d...", P); \
    else {                                                  \
        FD->bk = BK;                                         \
        BK->fd = FD;                                         \
    }                                                       \
}
```

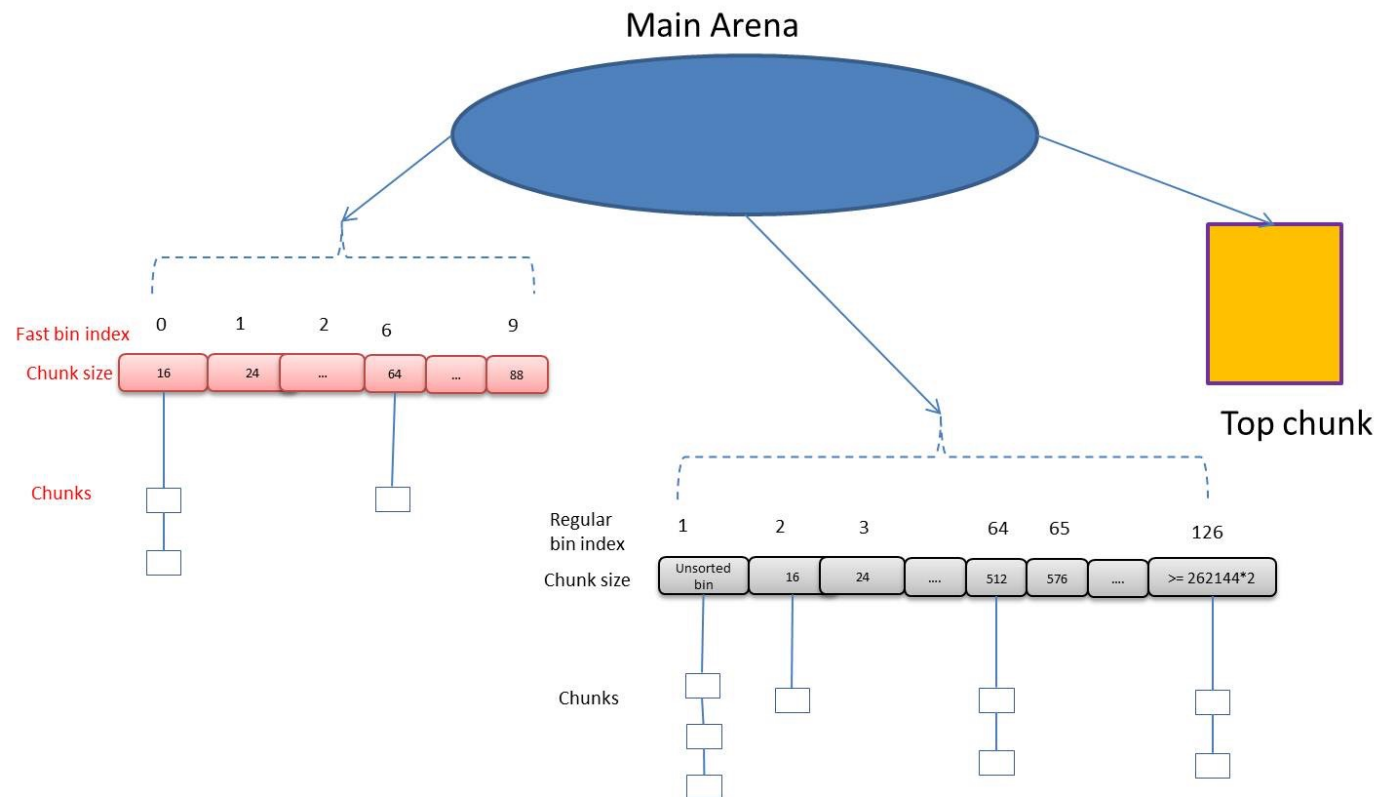
```
➔ ~ $ git clone git://sourceware.org/git/glibc.git
```

Main Arena



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



Q & A