

Linux Binary Exploitation

Advanced heap exploitation
angelboy@chroot.org

Outline

- Fastbin corruption
- Shrink the chunk
- Extend the chunk
- Unsorted bin attack

Fastbin corruption

- 假設程式存在著 **double free** 的漏洞
- 目的：
 - 我們可以利用 fastbin chunk 改掉 fd 使得**下下次** malloc 該 chunk 時可以取得自己想要的位置
- 為了利用 double free 的漏洞來改變 free chunk 中的 fd，我們可以利用一些 fastbin 的特性達到我們的目的，但我們也必須通過一些 chunk 的檢查

Fastbin corruption

- fastbin 的檢查
- in free(p) :
 - chunk address $< - \text{size}$ 且 alignment
 - chunk size $\geq \text{MINSIZE}(0x20)$ 且為 $0x10$ 的倍數
 - nextchunk->size
 - 大於 $\text{MINSIZE}(0x20)$
 - 小於 $\text{system_mem}(0x21000)$
 - 檢查屬於該 size 的 fastbin 中的第一塊 chunk 與 p 是否不同

Fastbin corruption

- fastbin 的檢查
- in malloc(bytes) :
 - 根據 bytes 大小取得 index 後，到對應的 fastbin 找，取出後檢查該 chunk 的 (unsigned long) size 是否屬於該 fastbin
 - 但實際比較的時候是先以 fastbin 中第一塊 size 取得 fastbin 的 index，再去用這個 index 跟剛剛算的 index 是否相同，不過這取 index 的方式是用 unsigned int (4 byte)

Fastbin corruption

- fast bin
 - a singly linked list
 - chunk size $\leq 0x80$ byte
 - 不取消 inuse flag
 - 依據 bin 中所存的 chunk 大小，在分為 10 個 fast bin 分別為 size 0x20,0x30,0x40...
 - LIFO
 - 當下次 malloc 大小與這次 free 大小相同時，會從相同的 bin 取出，也就是會取到相同位置的 chunk

Fastbin corruption

- fastbin layout

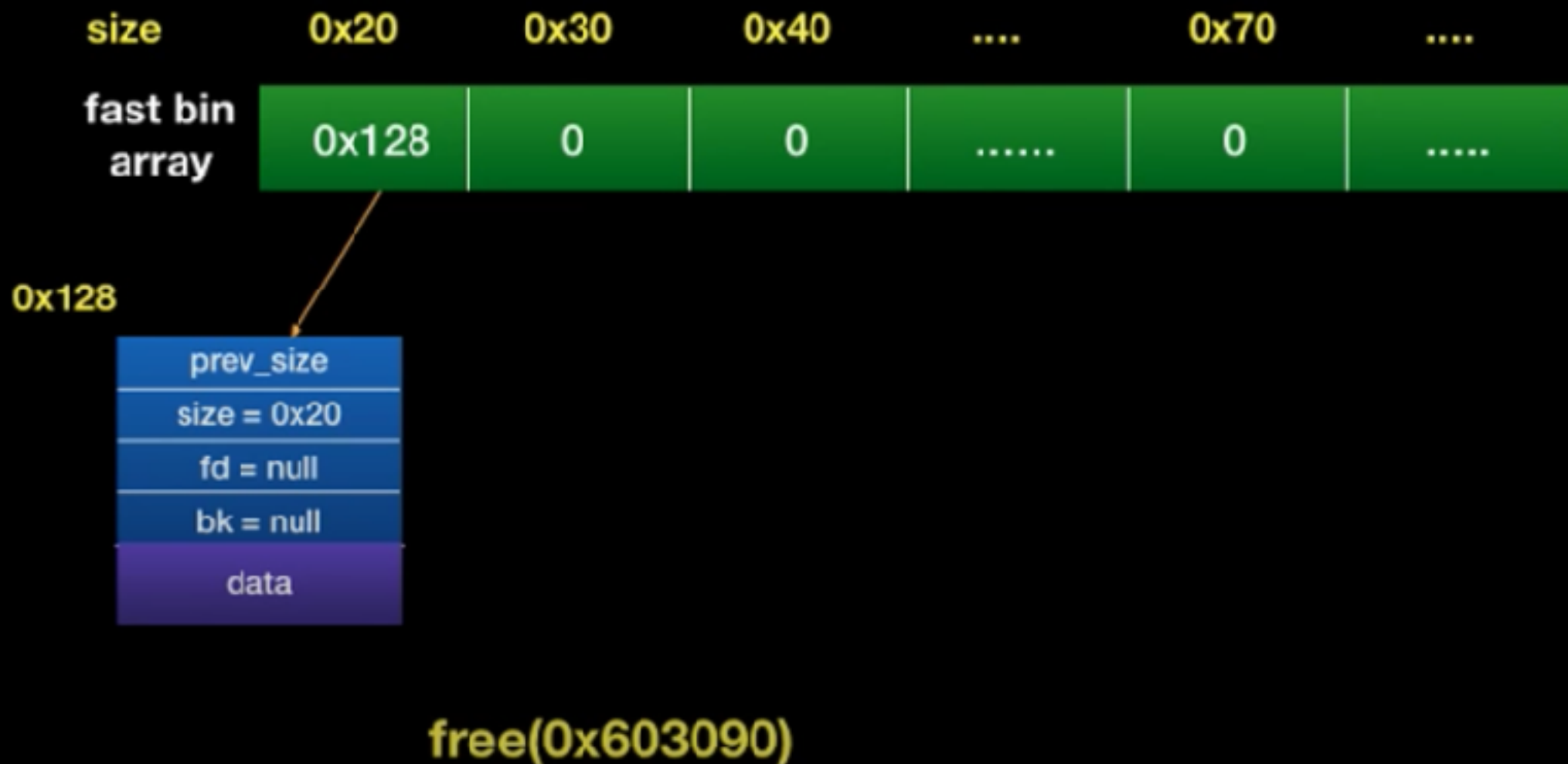


Fastbin corruption

- 我們可以利用 fastbin 的 chunk 在 free 的時候只檢查屬於該 size 的 fastbin 中的第一塊 chunk 與 p 是否不同這項特性，來創造 overlaps chunk
- 創造出一個 circle 的 singly linked list，這樣就可以達到類似 UAF 的效果

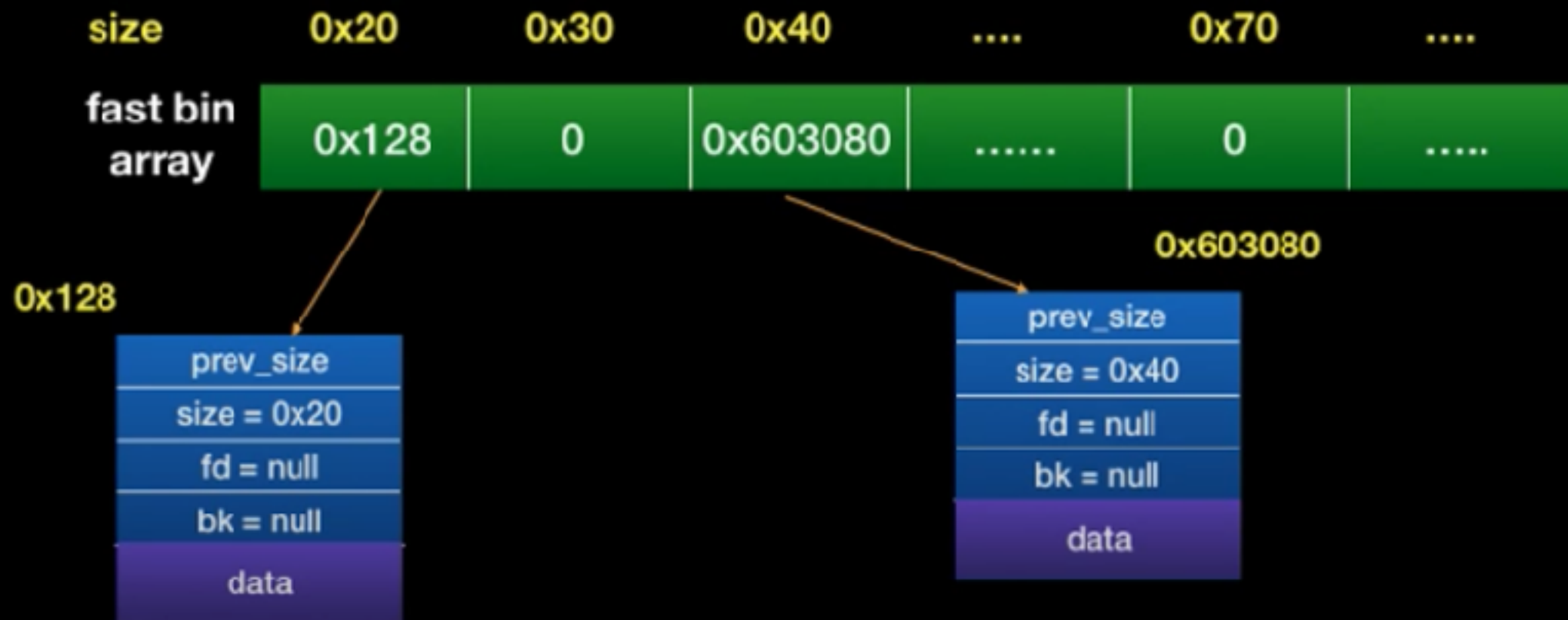
Fastbin corruption

- fastbin layout



Fastbin corruption

- fastbin layout



free(0x6030d0)

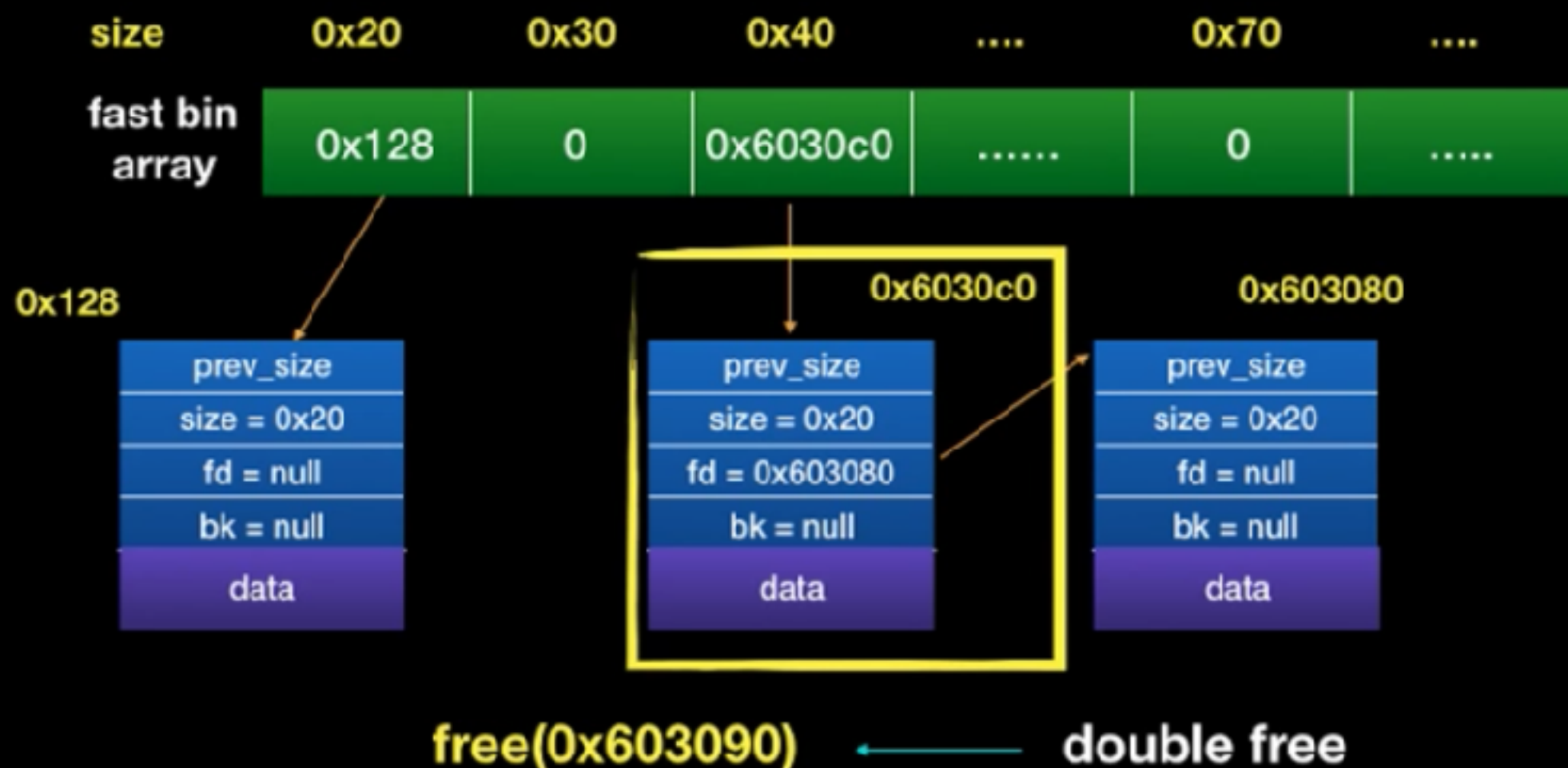
Fastbin corruption

- fastbin layout



Fastbin corruption

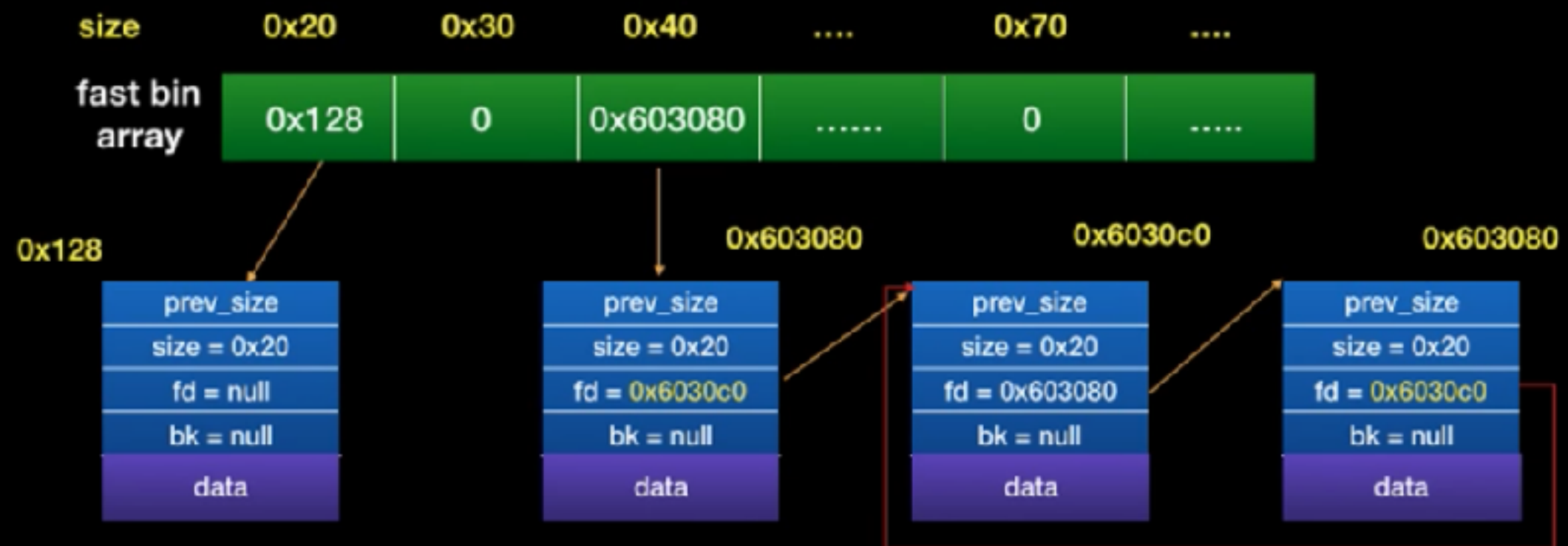
- fastbin layout



但因為 fastbin 只會檢查 fastbin 的第一塊，故通過檢查

Fastbin corruption

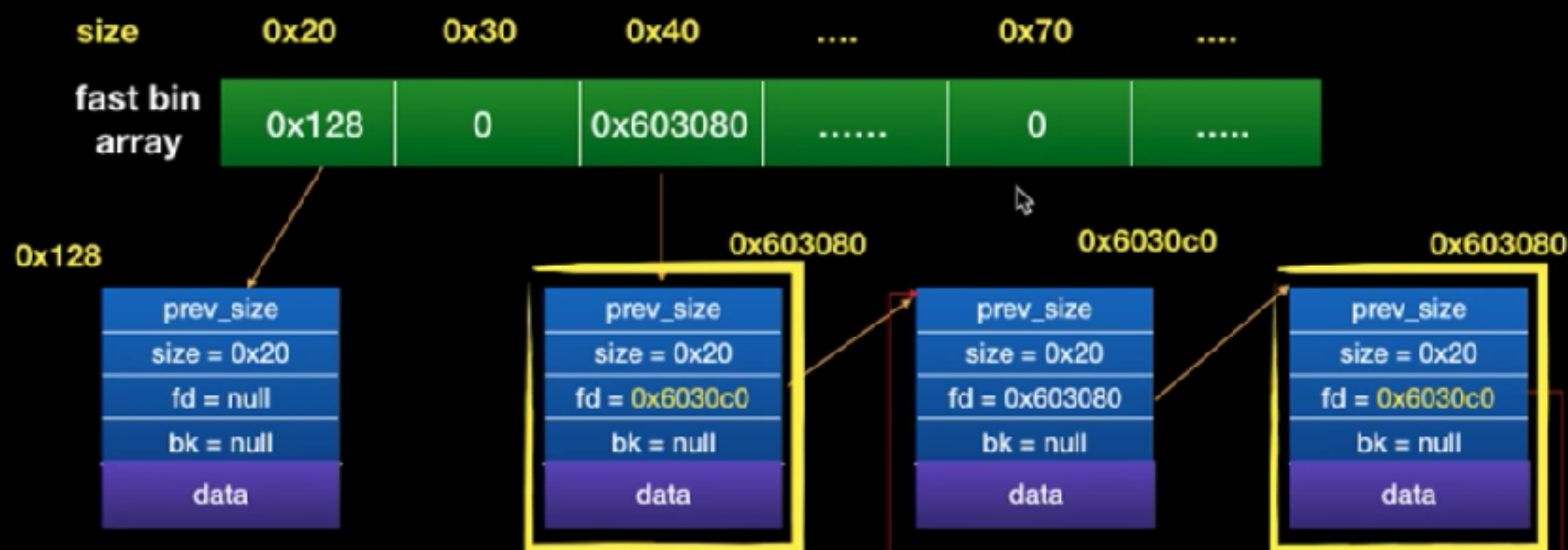
- fastbin layout



Circle singly linked list

Fastbin corruption

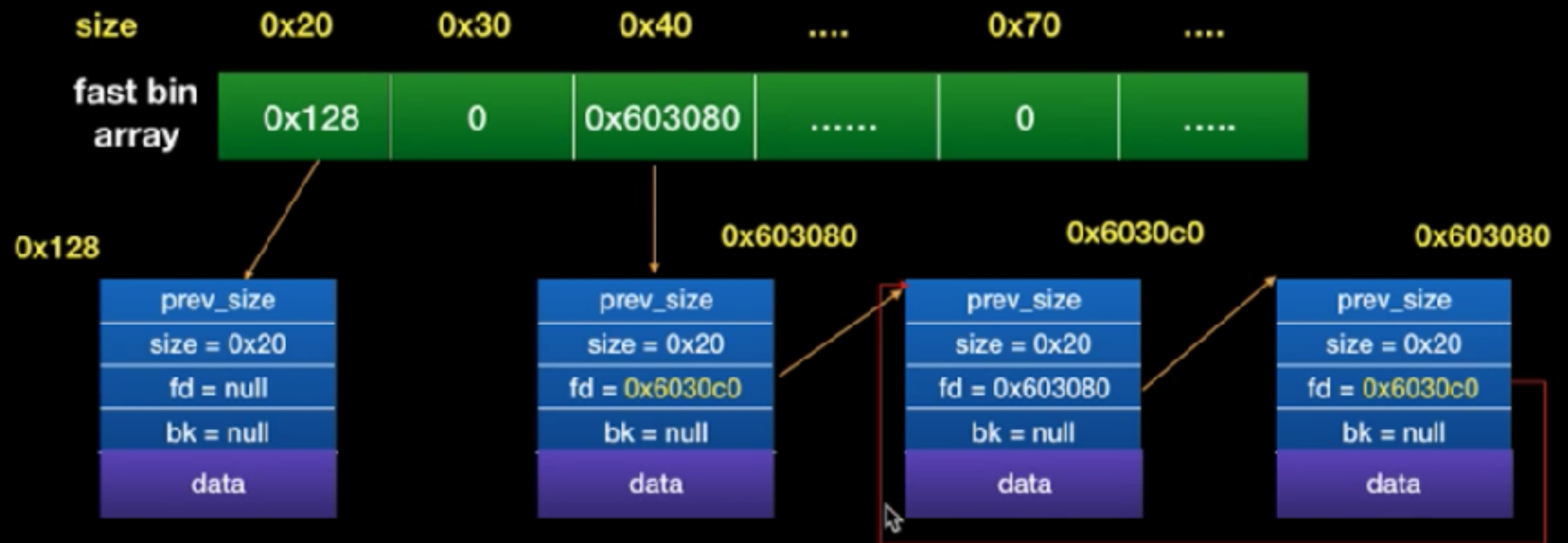
- fastbin layout



Same chunk in the fastbin
這樣在之後 malloc 時會有 overlap 情形發生

Fastbin corruption

- fastbin layout



malloc(0x30)

Fastbin corruption

- fastbin layout



You will get the 0x603080 chunk ,and overwrite the fd
But the chunk is also in the fastbin

Fastbin corruption

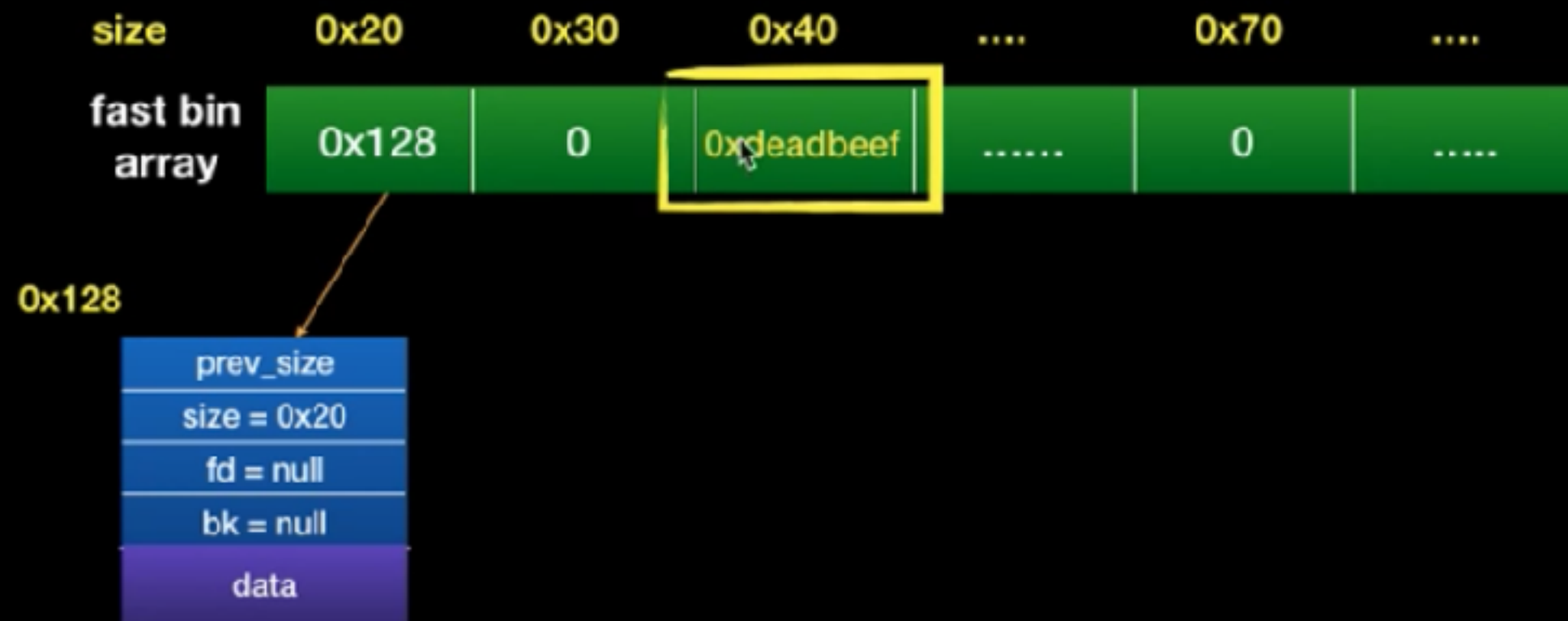
- fastbin layout



malloc(0x30)

Fastbin corruption

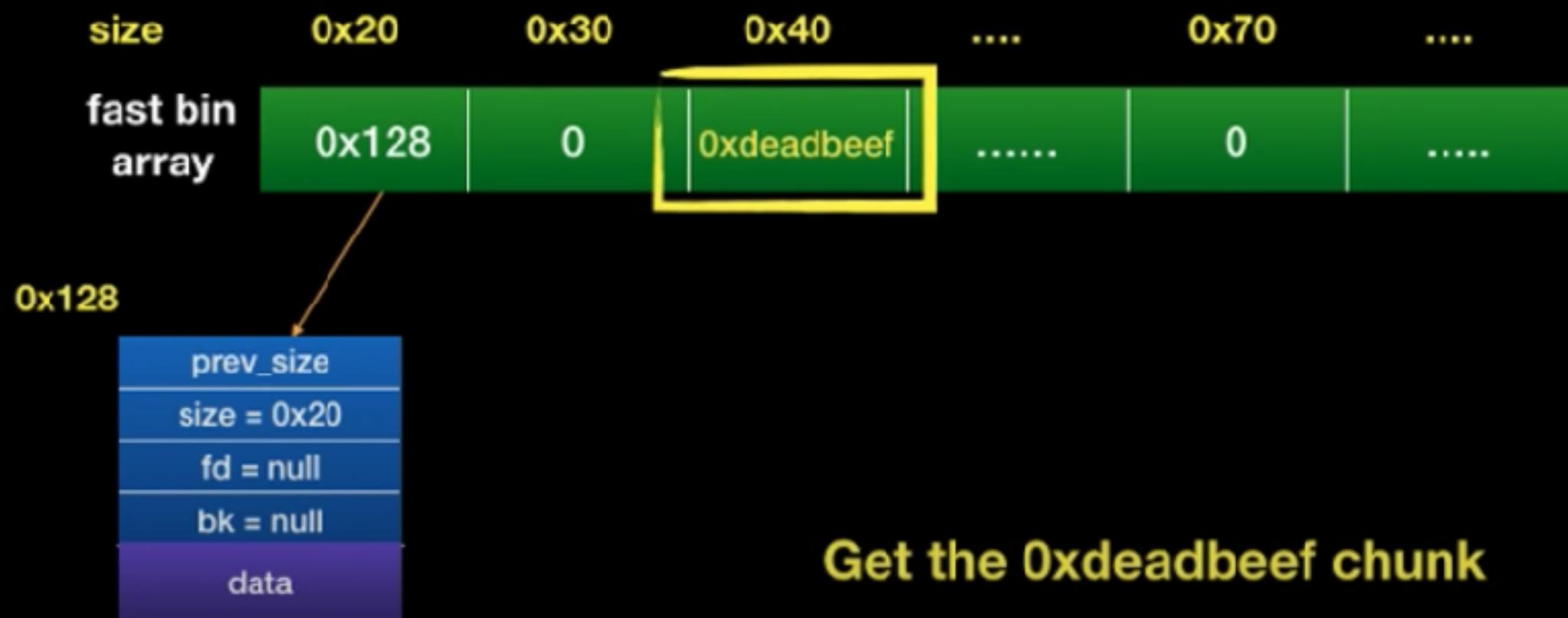
- fastbin layout



`malloc(0x30)`

Fastbin corruption

- fastbin layout



Get the 0xdeadbeef chunk

Fastbin corruption

- fd 只要符合 size 是否屬於該 chunk 就可以通過 malloc 檢查即可，因此只要想寫入的地址附近有屬於該 bin 的 size 就可以讓 malloc 分配到該位置
- 根據 bytes 大小取得 index 後，到對應的 fastbin 找，取出後檢查該 chunk 的 (unsigned long) size 是否屬於該 fastbin
 - 但實際比較的時候是先以 fastbin 中第一塊 size 取得 fastbin 的 index，再去比 index 跟剛剛算的 index 是否相同，不過這取 index 的方式是用 unsigned int (4 byte)，所以偽造時不用滿足 8 byte
- 因此沒有檢查 alignment 所以不一定要以八的倍數作為 chunk 的 address

Outline

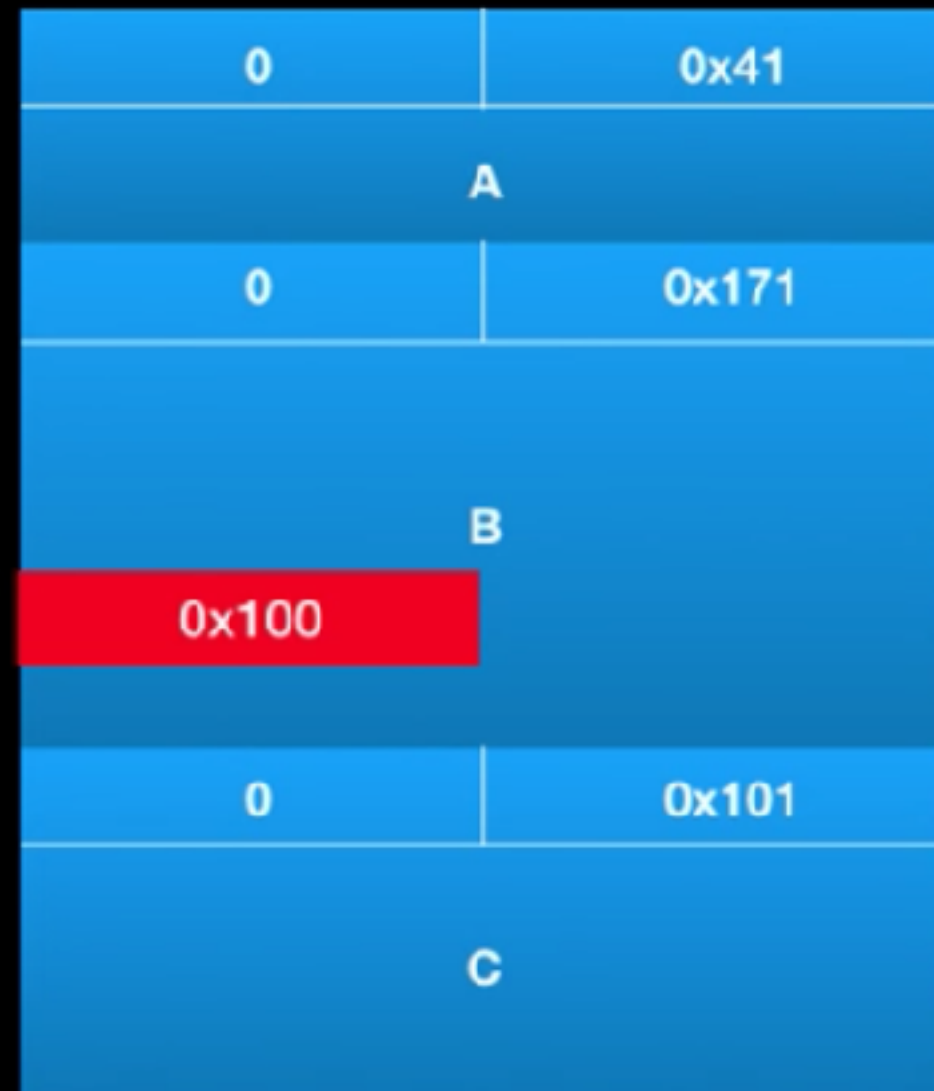
- Fastbin corruption
- Shrink the chunk
- Extend the chunk
- Unsorted bin attack

Shrink the chunk

- 假設存在一個 off-by-one null byte 的漏洞
- 目的：
 - 創造出 **overlap chunk**，進而更改其他 chunk 中的內容
- 主要利用 unsorted bin ,smallbin 會 unlink 做合併的特性來達到我們的目的

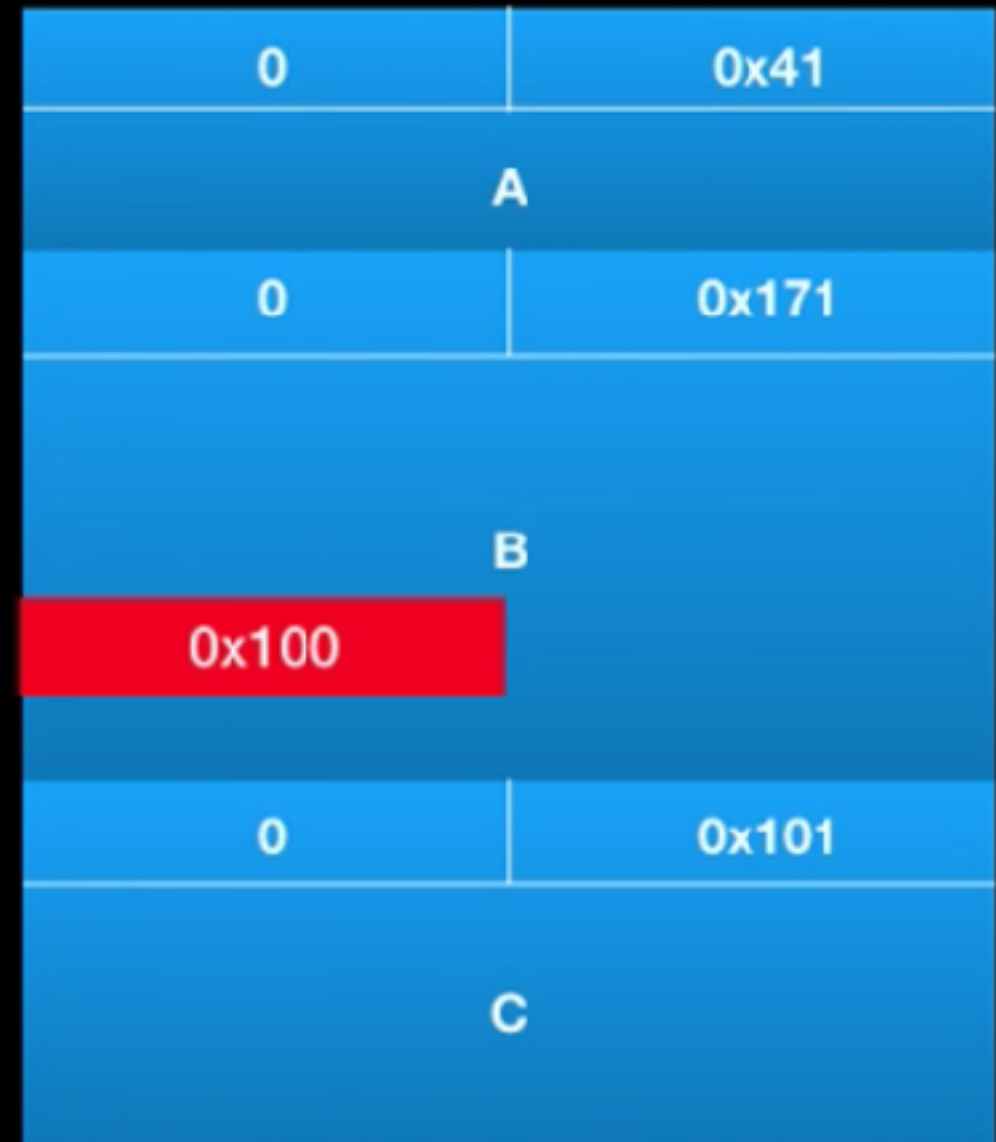
Shrink the chunk

- 一開始先 malloc 3 塊 chunk 至 heap 段，且 fastbin 是空的
 - 在 B 中的 data 裡，我們要在 offset 0xf0 位置上填上 0x100
 - 其目的是為了之後能夠通過 `prev_size == size` 的檢查



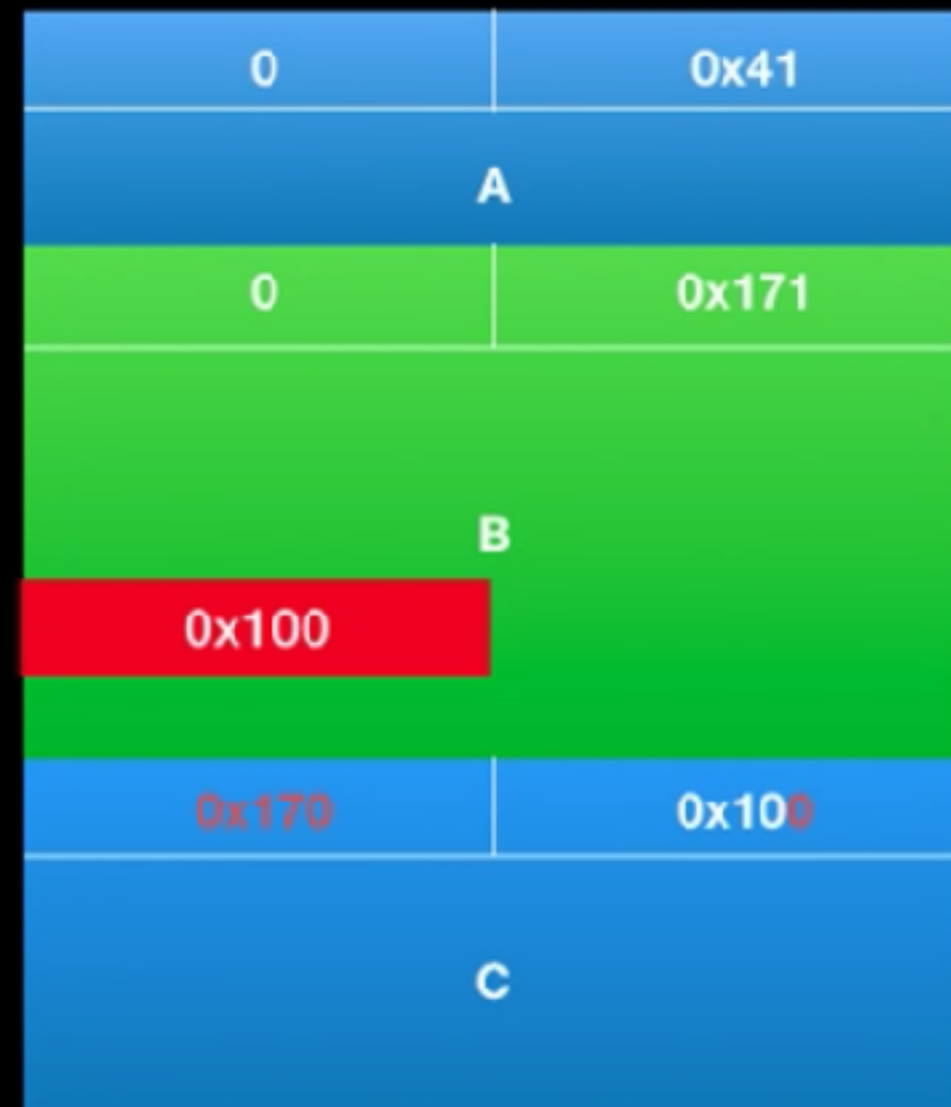
Shrink the chunk

- `free(B)`



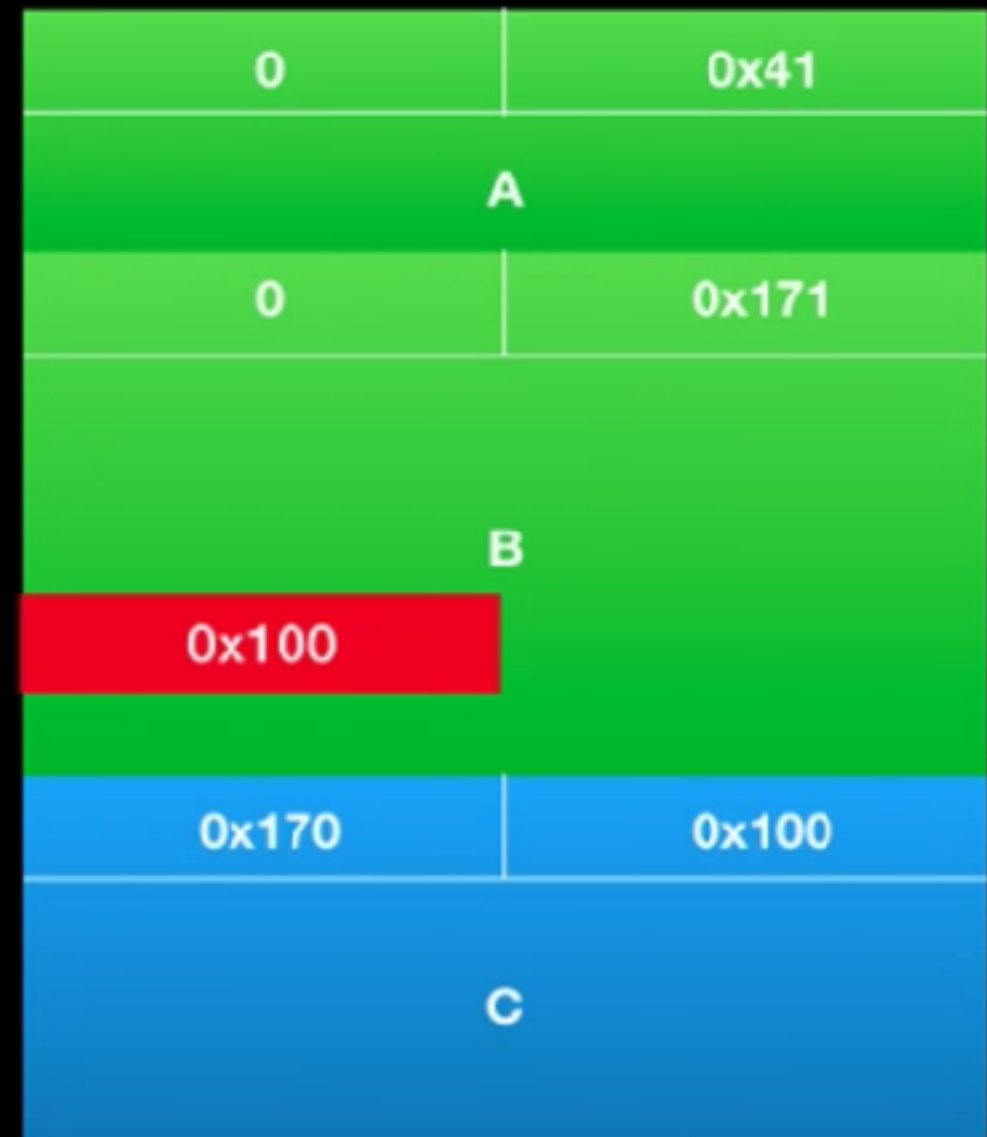
Shrink the chunk

- `free(A)`



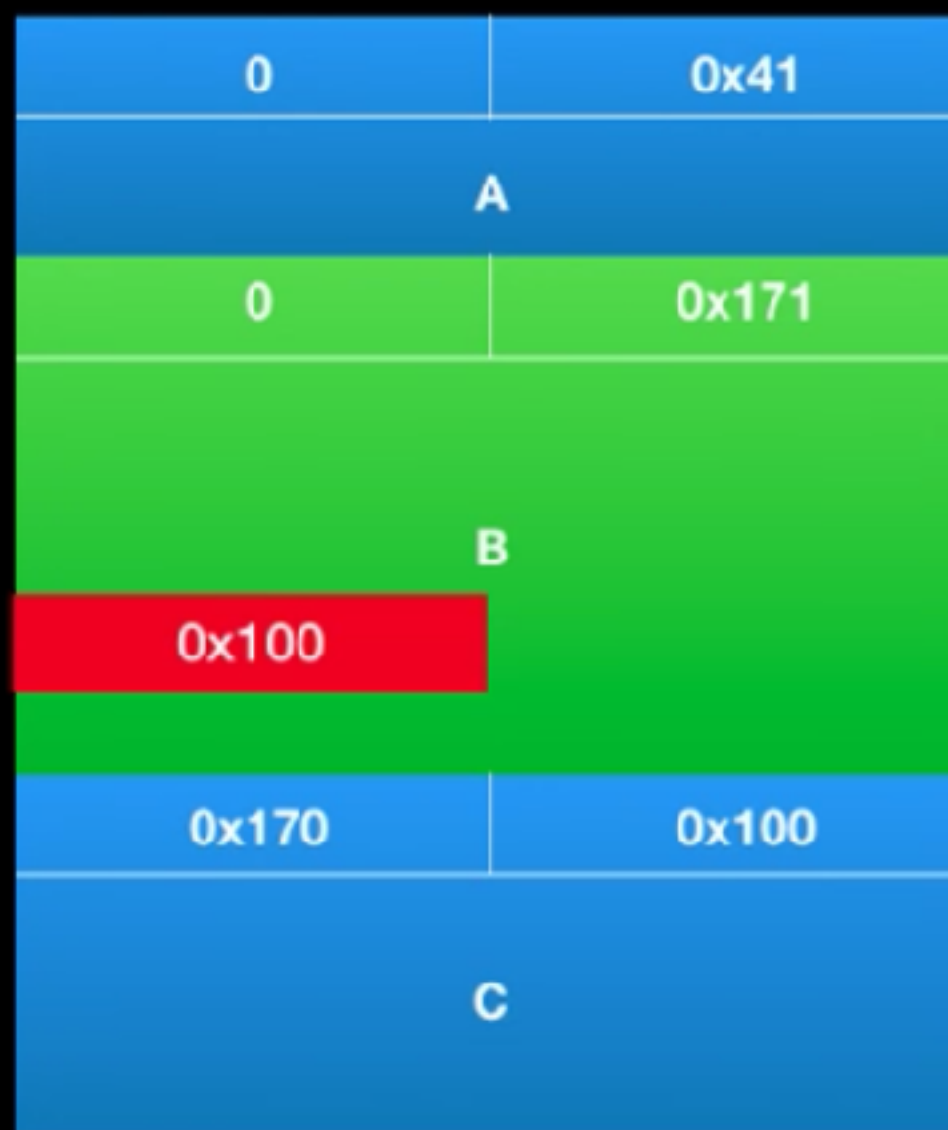
Shrink the chunk

- `malloc(0x38)`



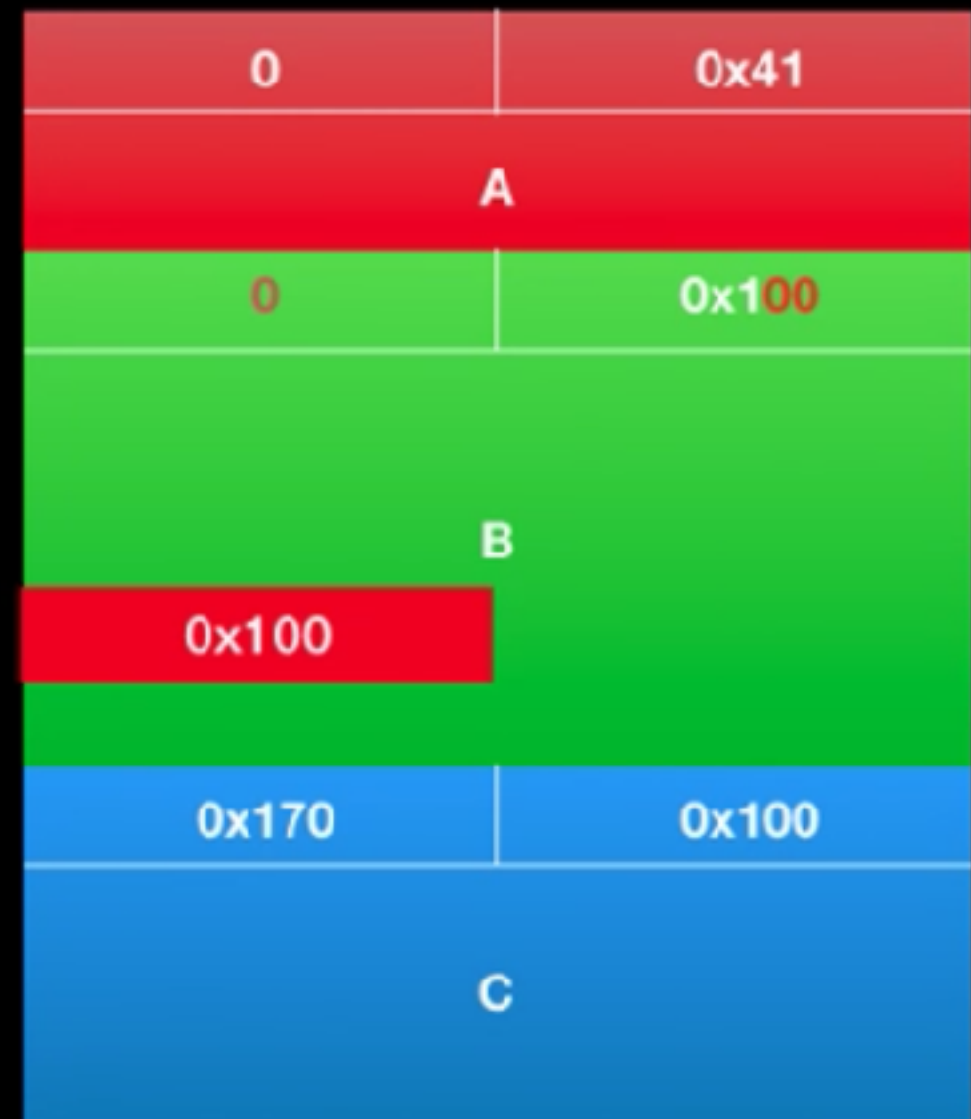
Shrink the chunk

- read data to A and off-by-one overflow



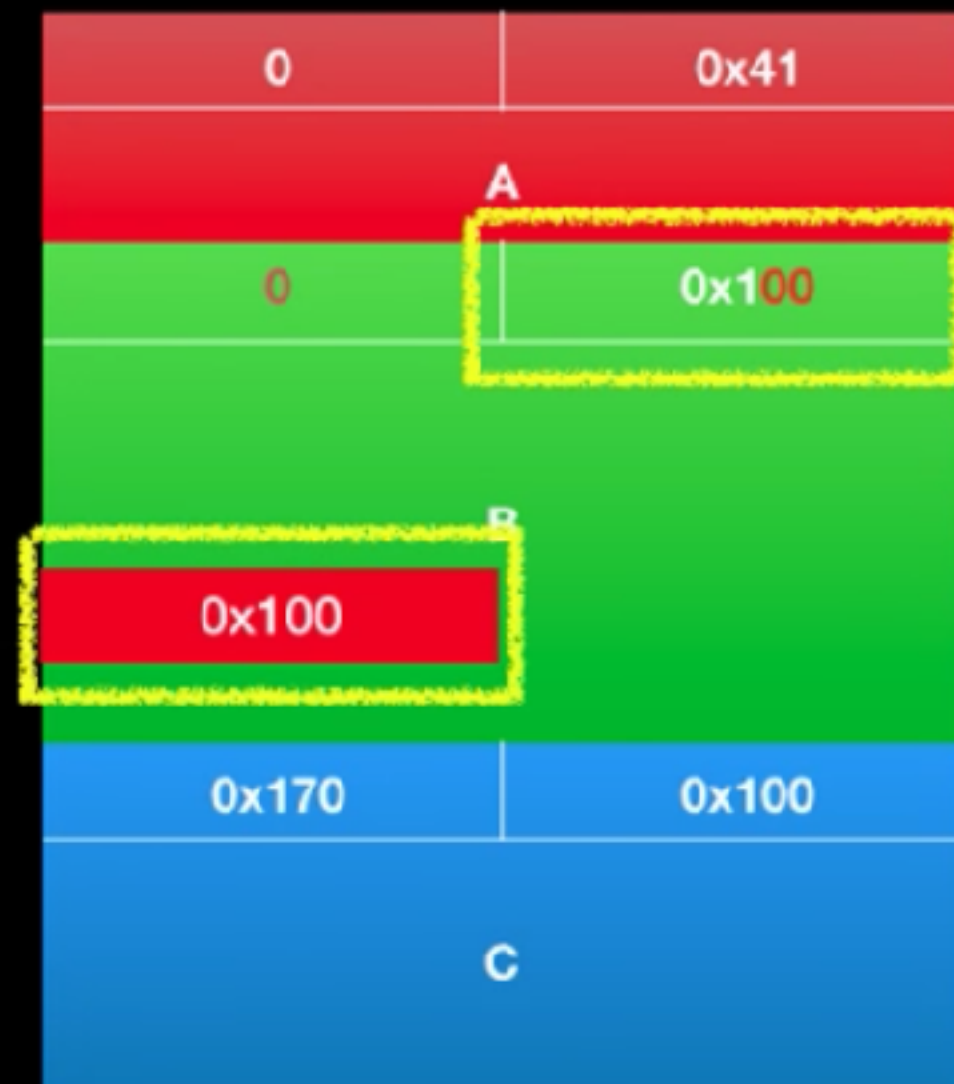
Shrink the chunk

- `malloc(0x80)`



Shrink the chunk

- malloc(0x80)
- 這邊在 malloc 時，因為要從 unsorted bin 拿出來，所以需要 unlink 因此會對 prev_size 和 size 做檢查
- 一開始放入 0x100 目的在這



Shrink the chunk

- `malloc(0x30)`



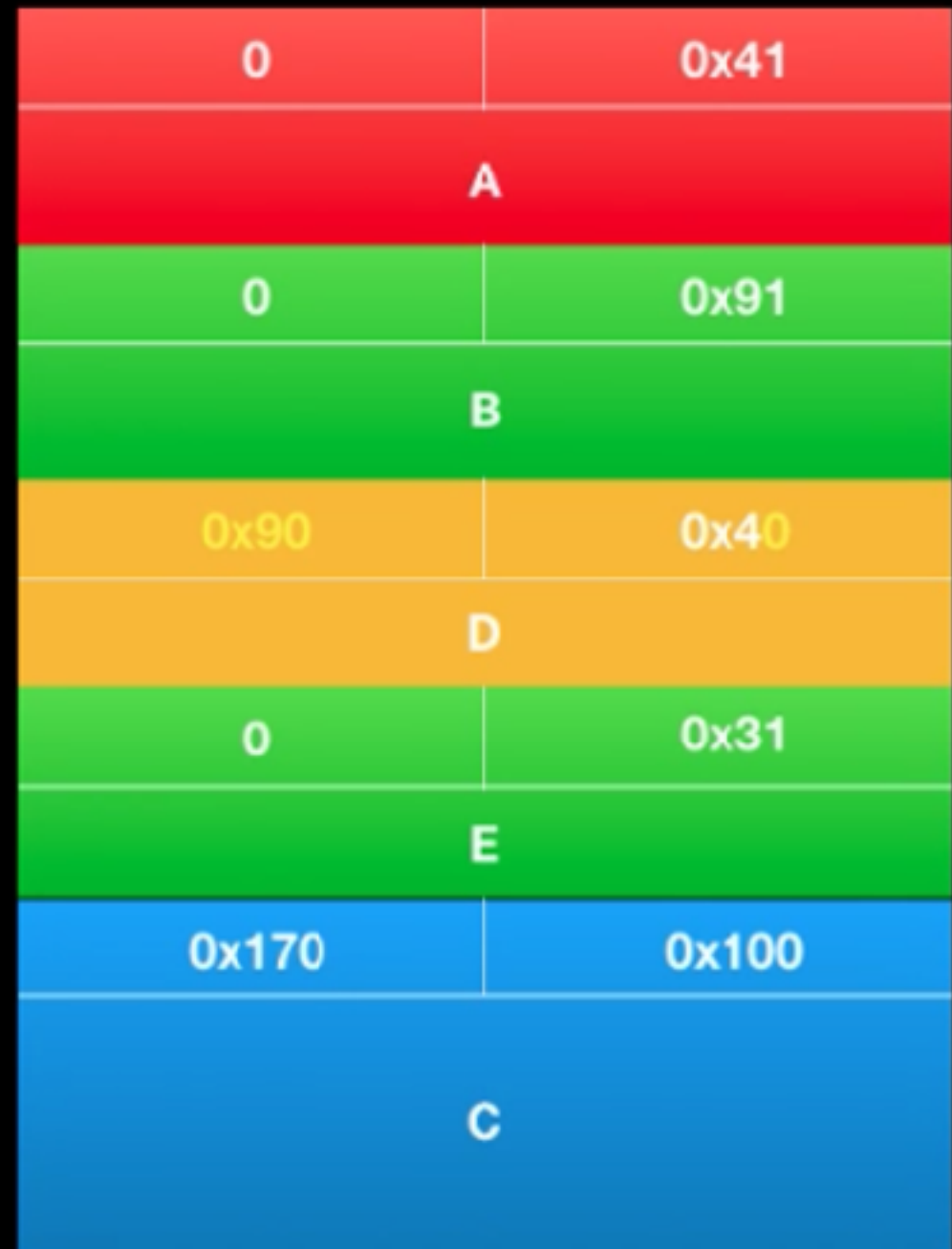
Shrink the chunk

- `free(B)`



Shrink the chunk

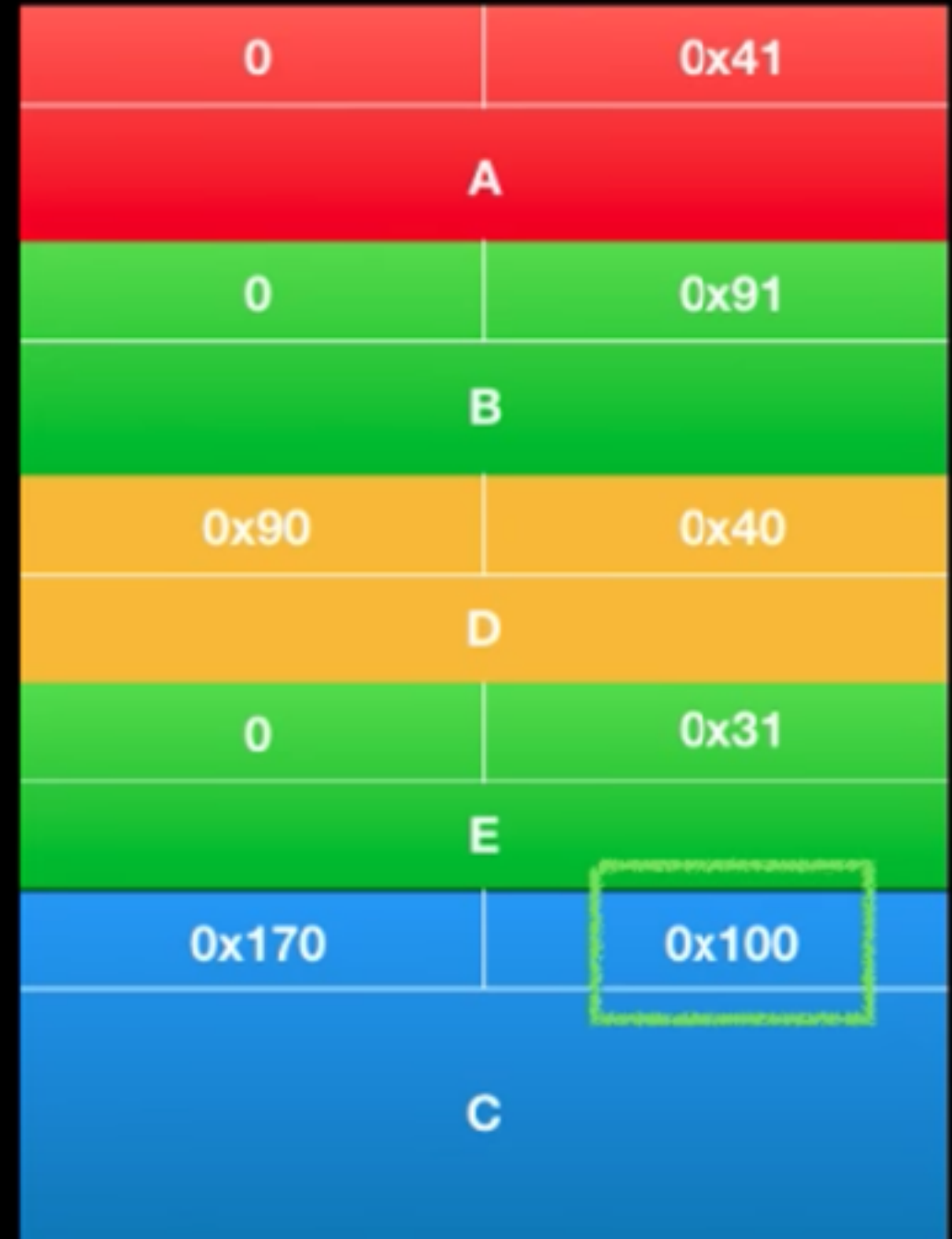
- free(C)



Shrink the chunk

- free(C)

此時會因為 C 是 smallbin
的大小的關係，
所以會檢測上一塊是否 inused



Shrink the chunk

- free(C) - merge

如果上一塊是 freed 就會
根據prev_size
去找上一塊 chunk
的 header 做
合併及 unlink



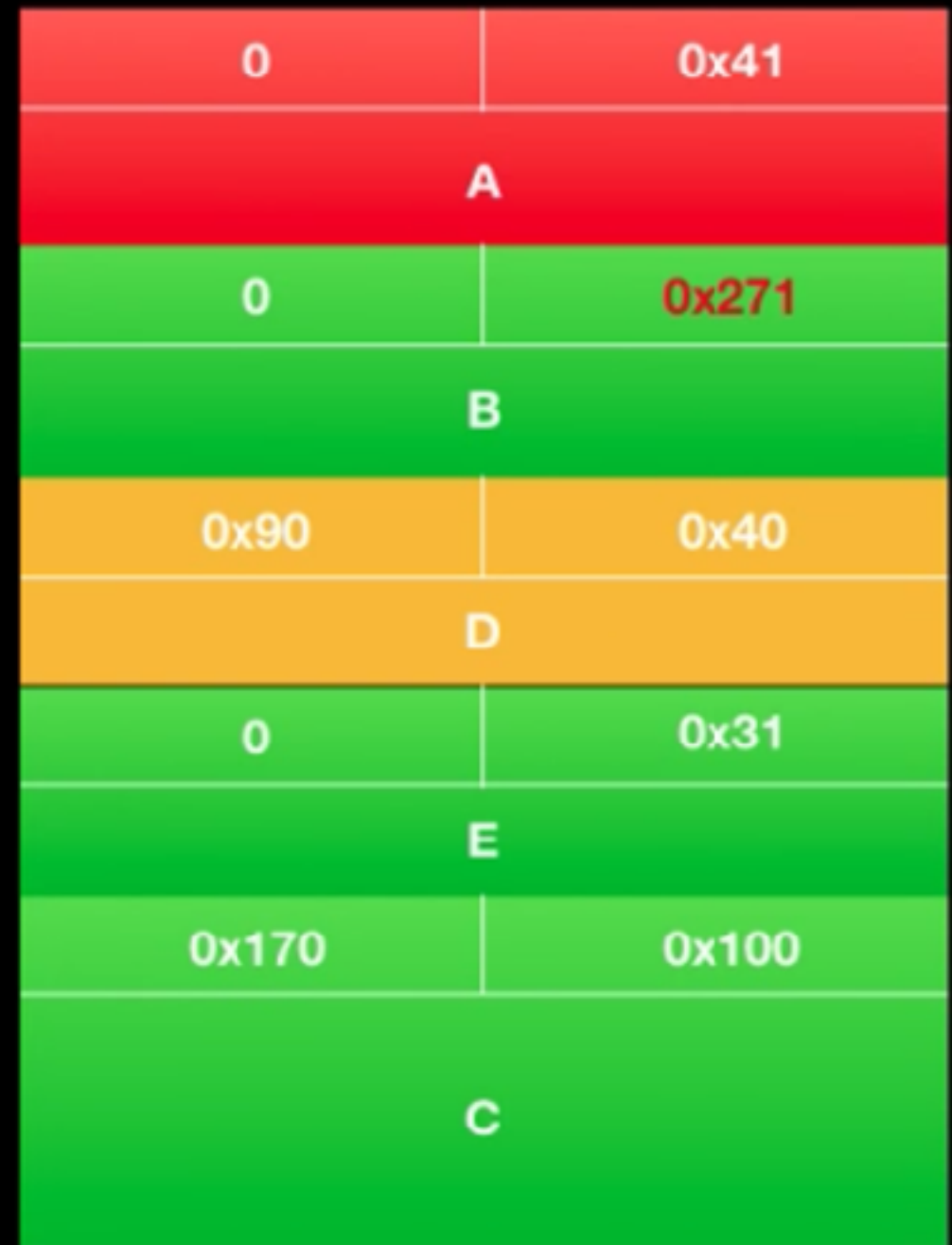
Shrink the chunk

此時 **unsorted bin** 存這
這塊大 chunk，
所以下次
malloc 會用這一塊
先份配給 **user**



Shrink the chunk

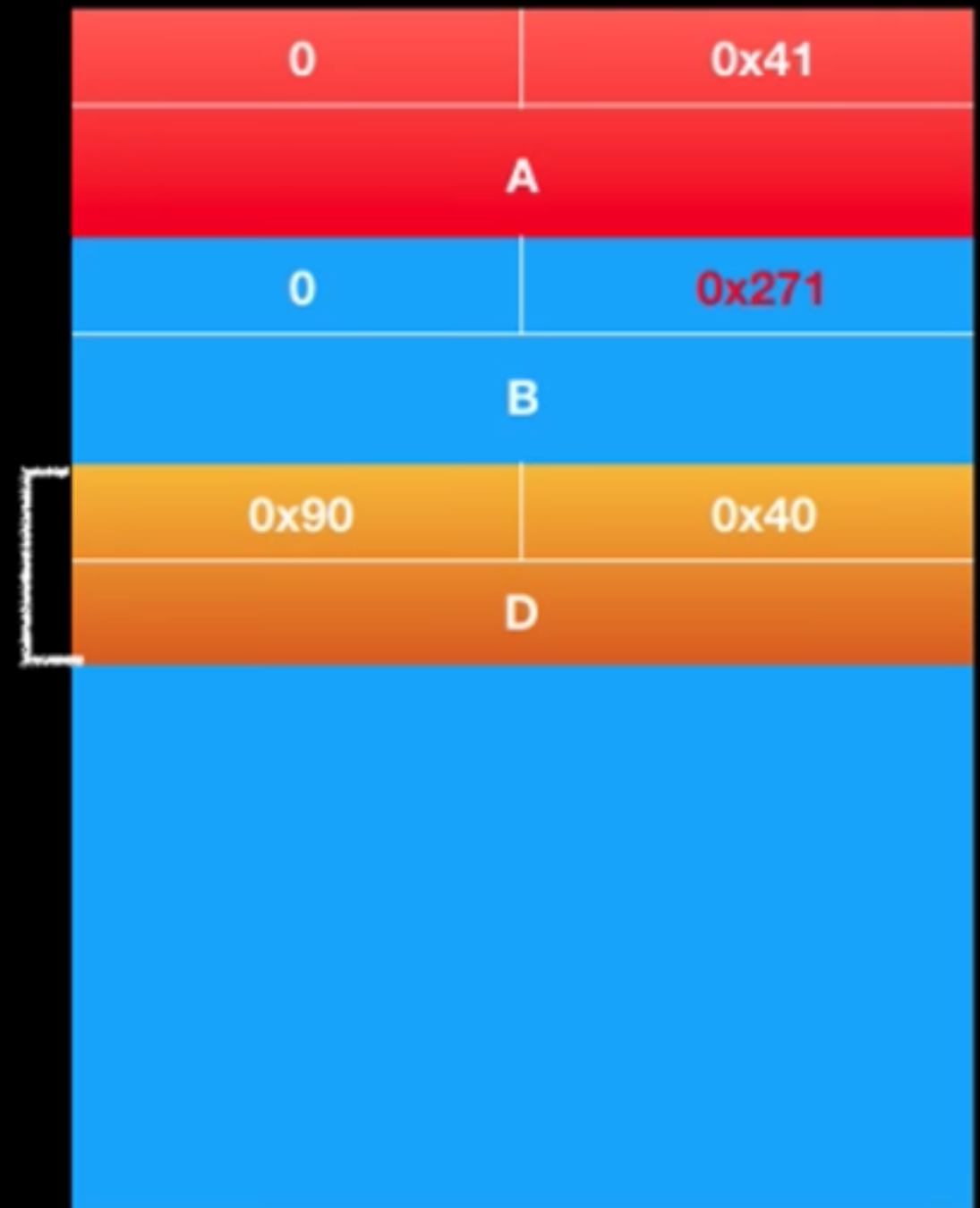
- `malloc(0x260)`



Shrink the chunk

- 此時可任意改 D

overlap chunk



Shrink the chunk

- overlap 的情況其實還蠻 powerful 的，如果中間 overlap 部分有 function pointer 或者是其他有用的 struct 可間接控制程式流程
- 也可以配合 fastbin freed chunk 更改 fd

Outline

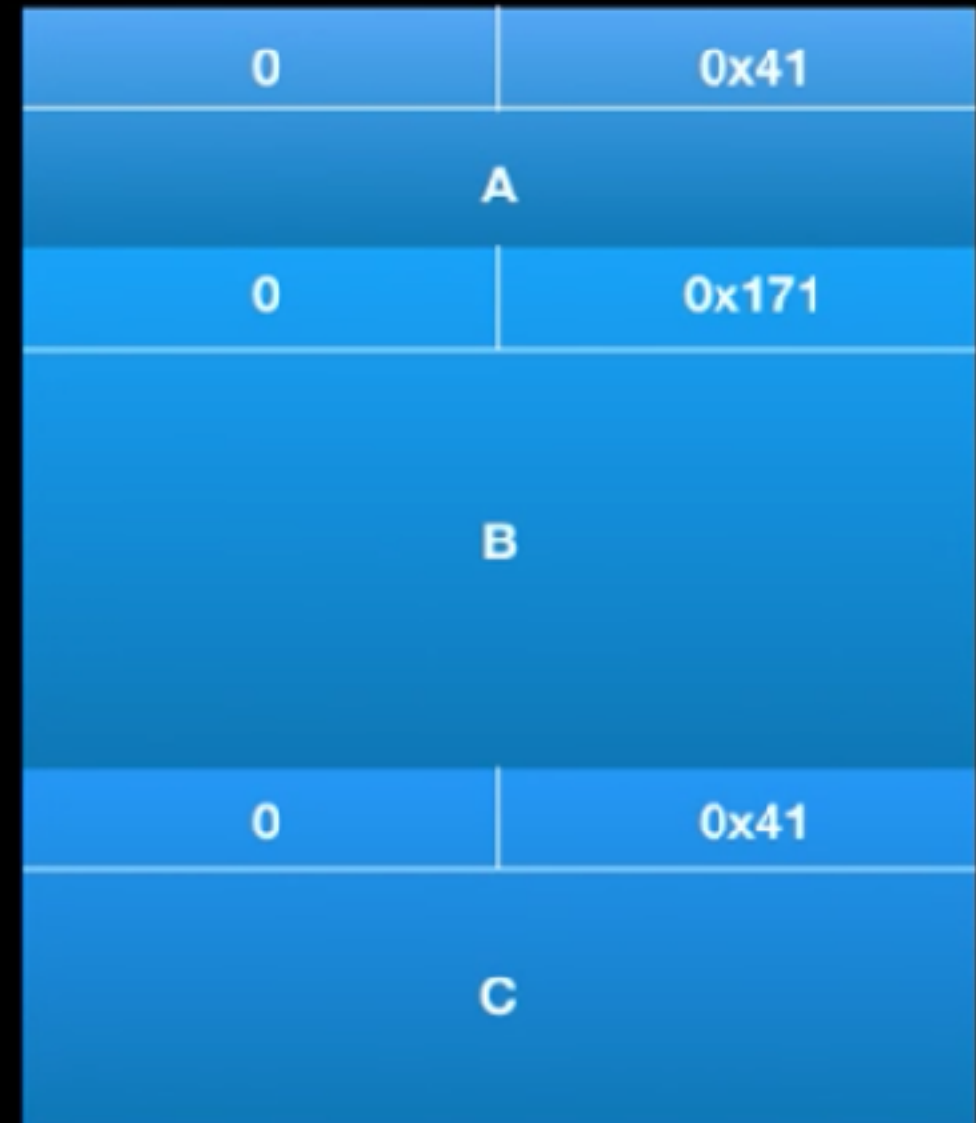
- Fastbin corruption
- Shrink the chunk
- **Extend the chunk**
- Unsorted bin attack

Extend the chunk

- 假設存在一個 off-by-one 的漏洞
- 目的：
 - 一樣是創造出 **overlap chunk**，進而更改其他 chunk 中的內容
- 跟 shrink 很像，但主要是加大 size 直接吃掉後面的 chunk，只要後面的 chunk header 有對上就好

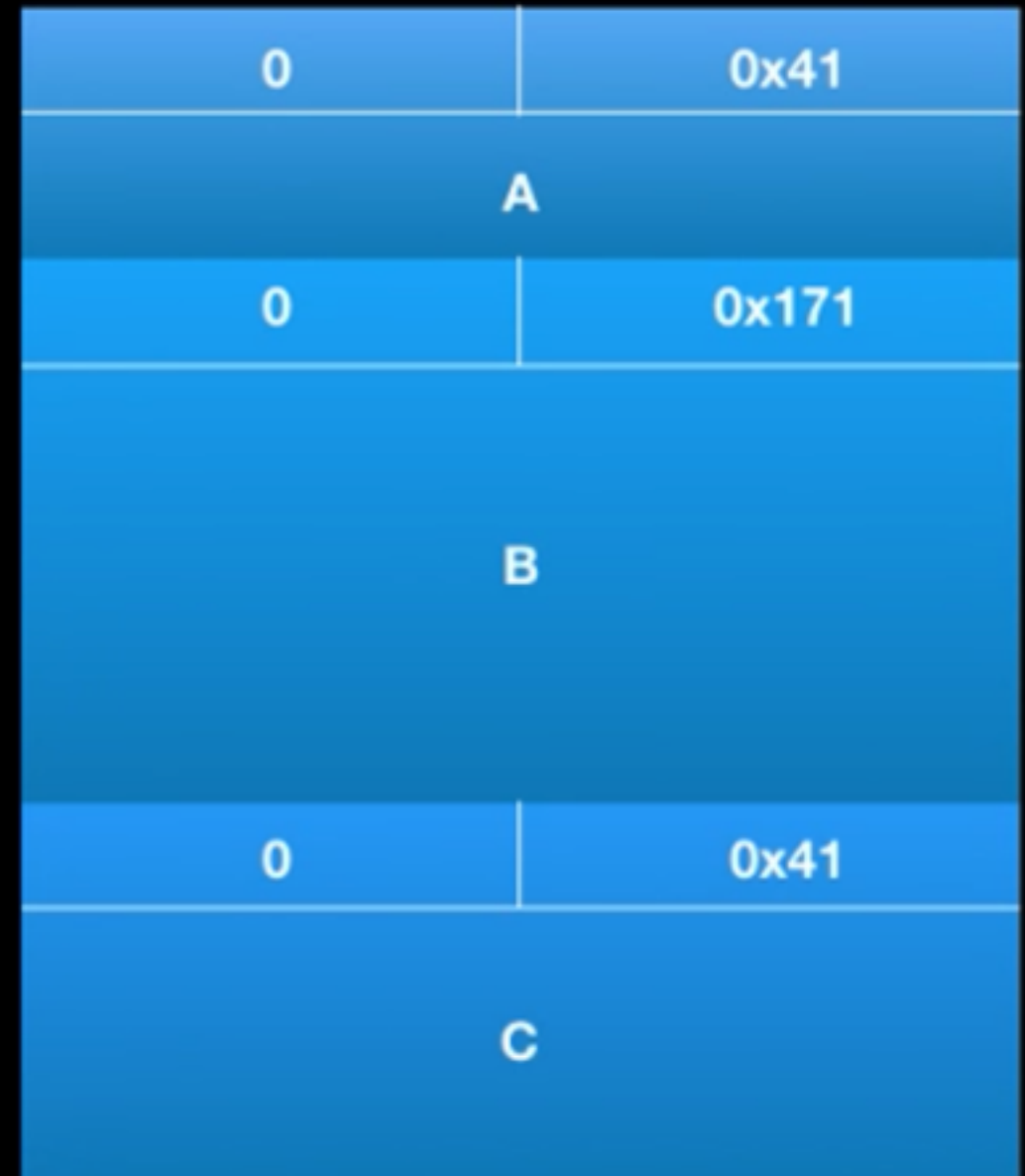
Extend the chunk

- 一開始先 malloc 3 塊 chunk 至 heap 段



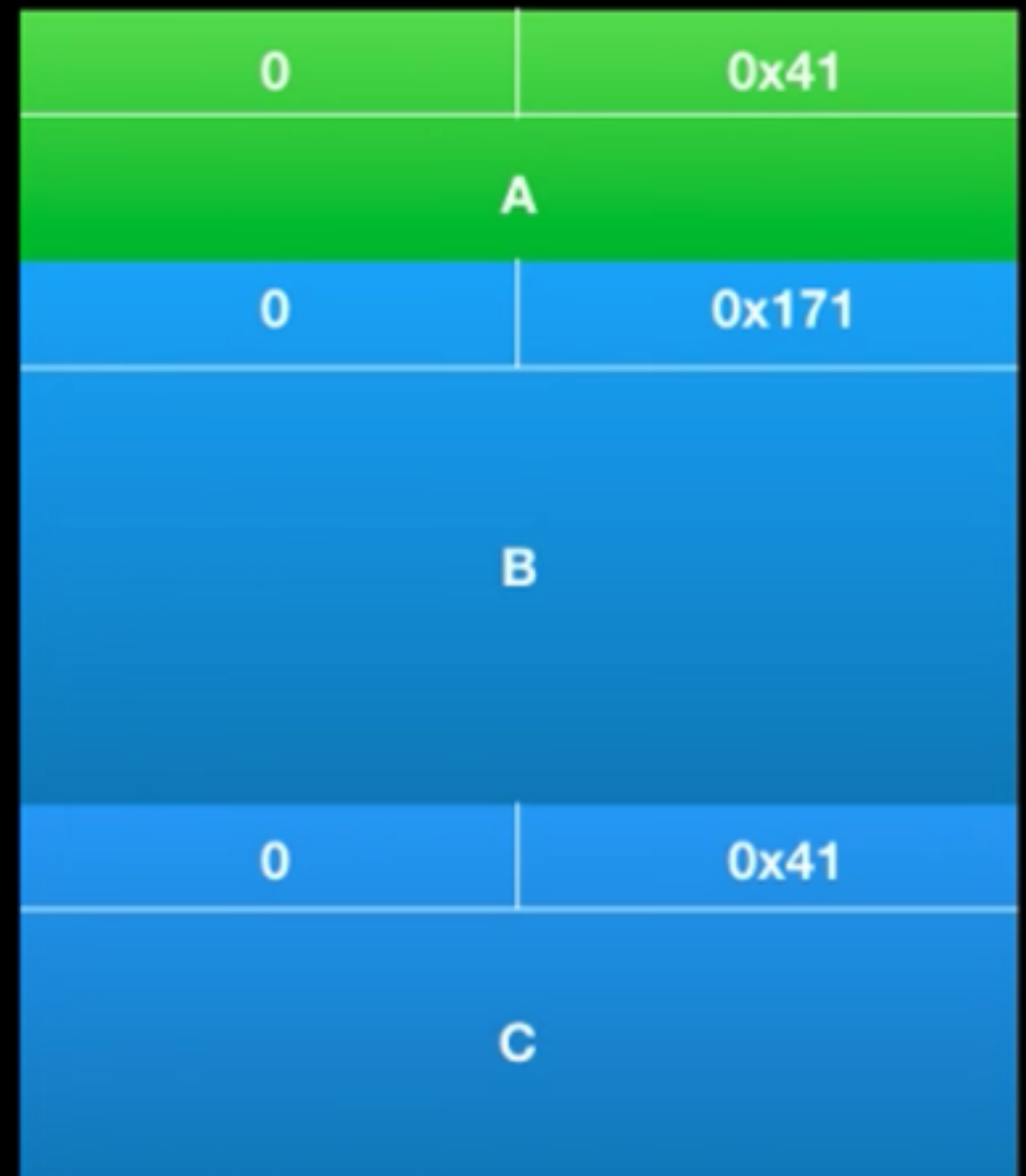
Extend the chunk

- `free(A)`



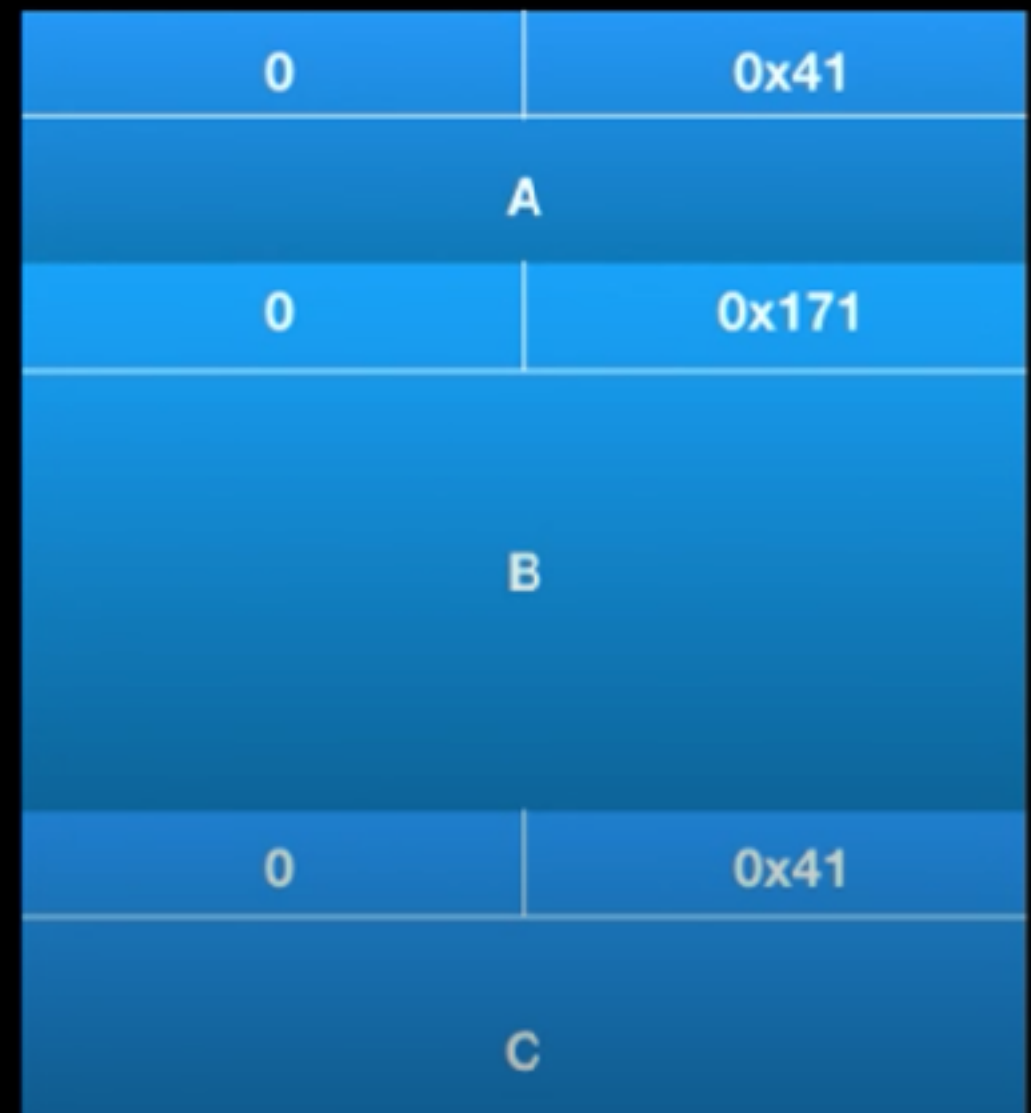
Extend the chunk

- `malloc(0x38)`



Extend the chunk

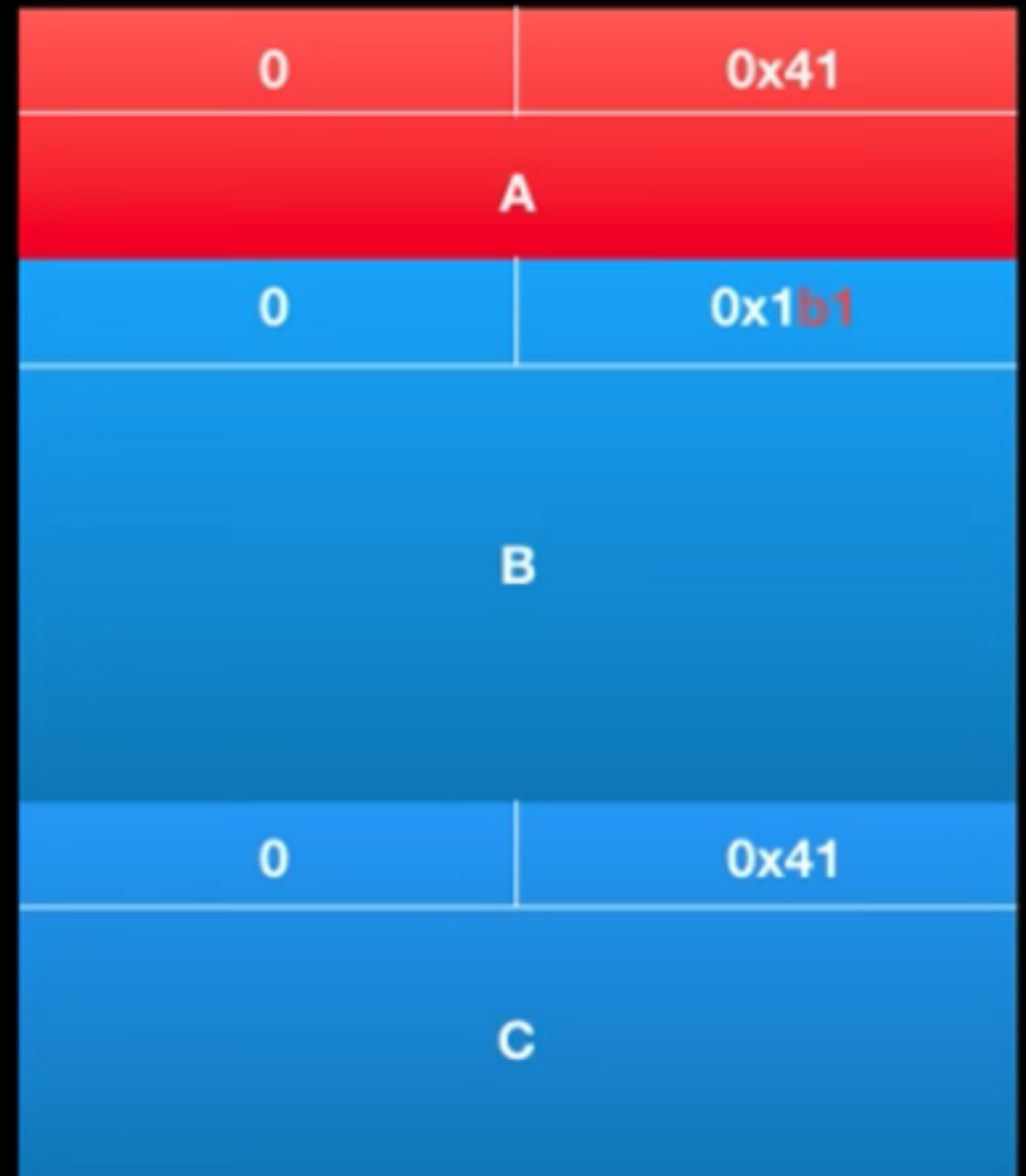
- read data to A and off-by-one overflow



Extend the chunk

- `free(B)`

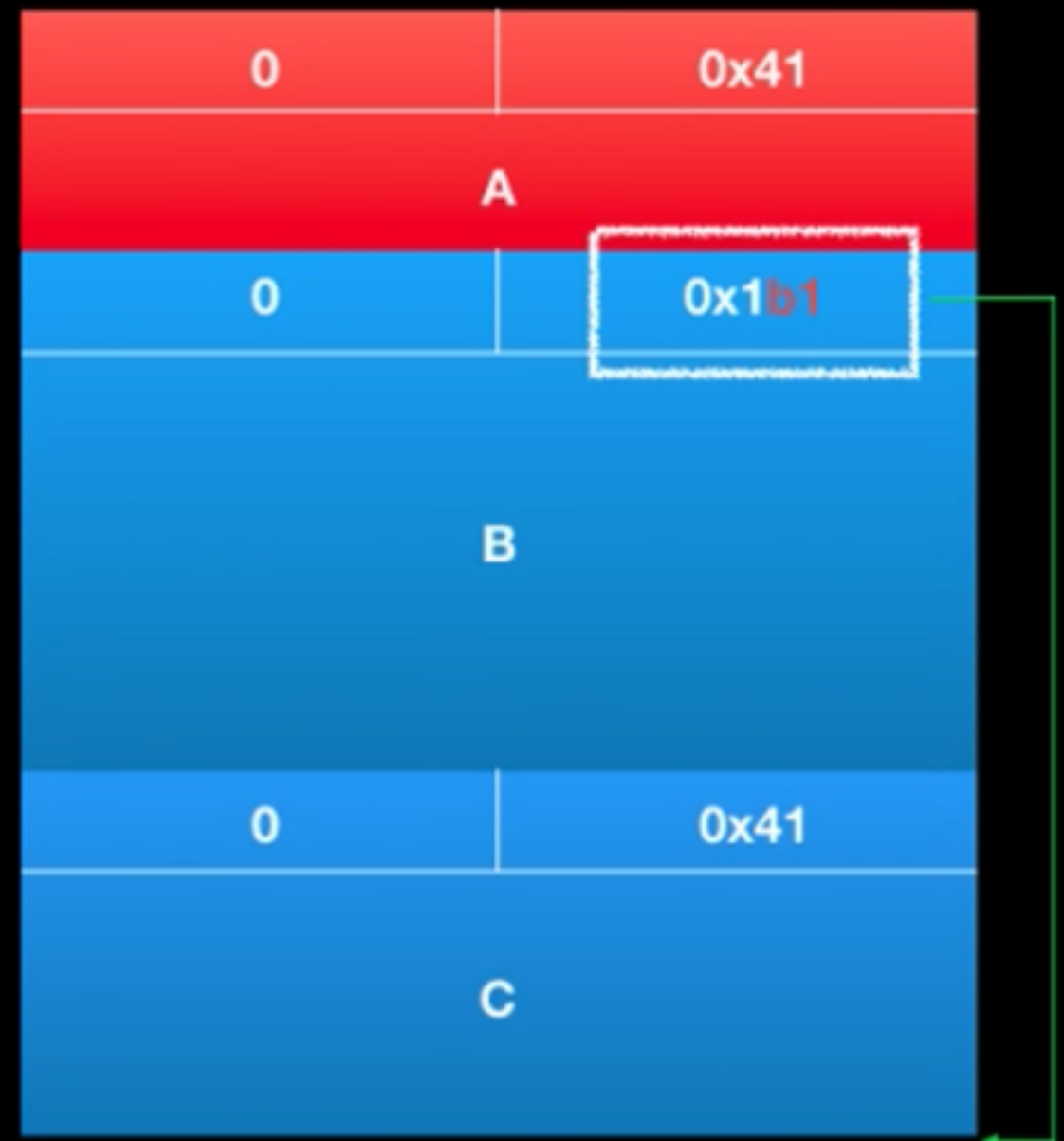
$0x1b1 = 0x171 + 0x40$
為了之後要把 C 吃進去



Extend the chunk

- free(B)

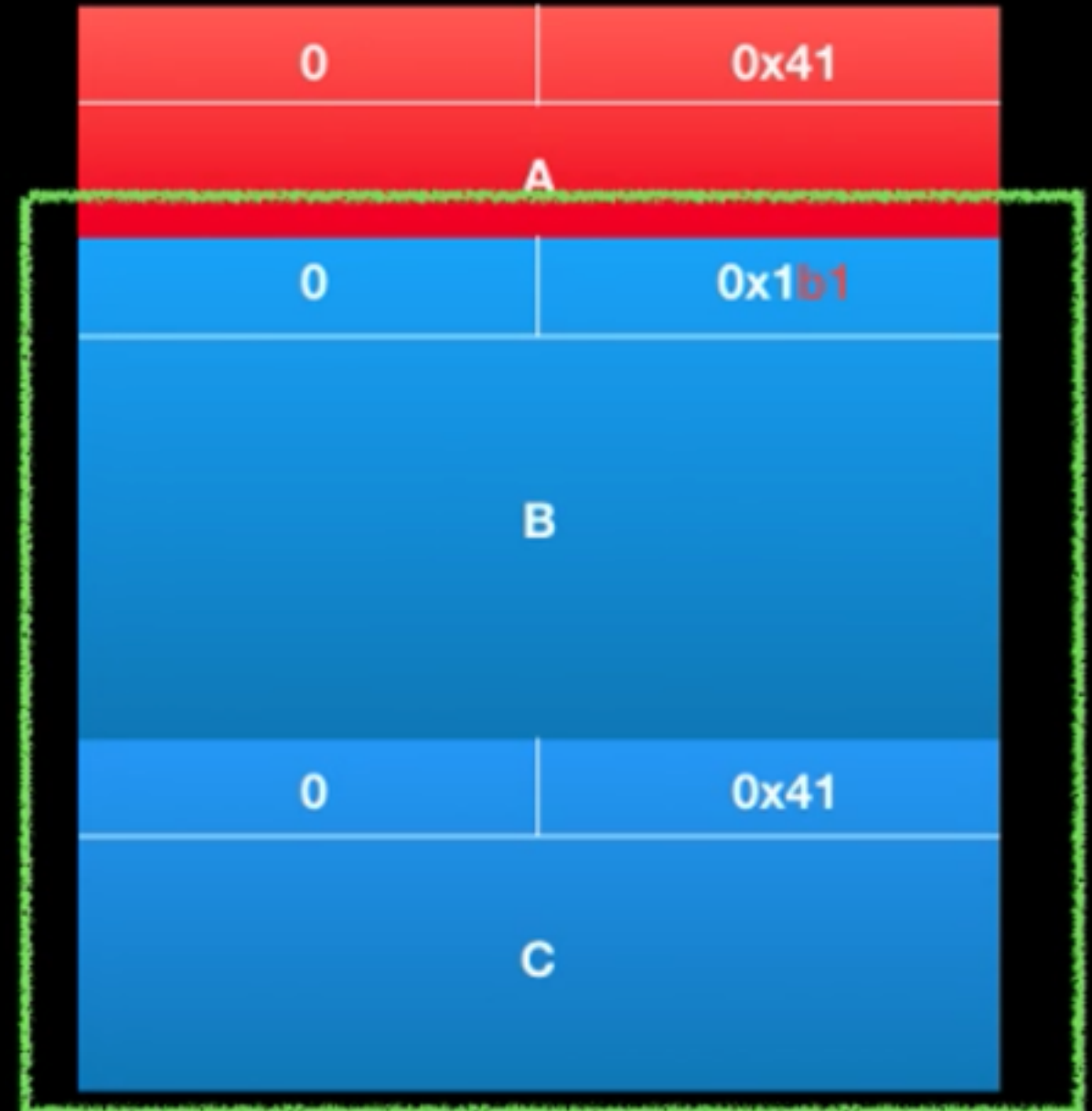
在此時會根據 B 的 size
去找下一塊 chunk 的 header
因剛剛偽造的 size 會去抓到
C 的下一塊 chunk header
做 inused bit 檢查
但 C 是 inused 所以會通過



Extend the chunk

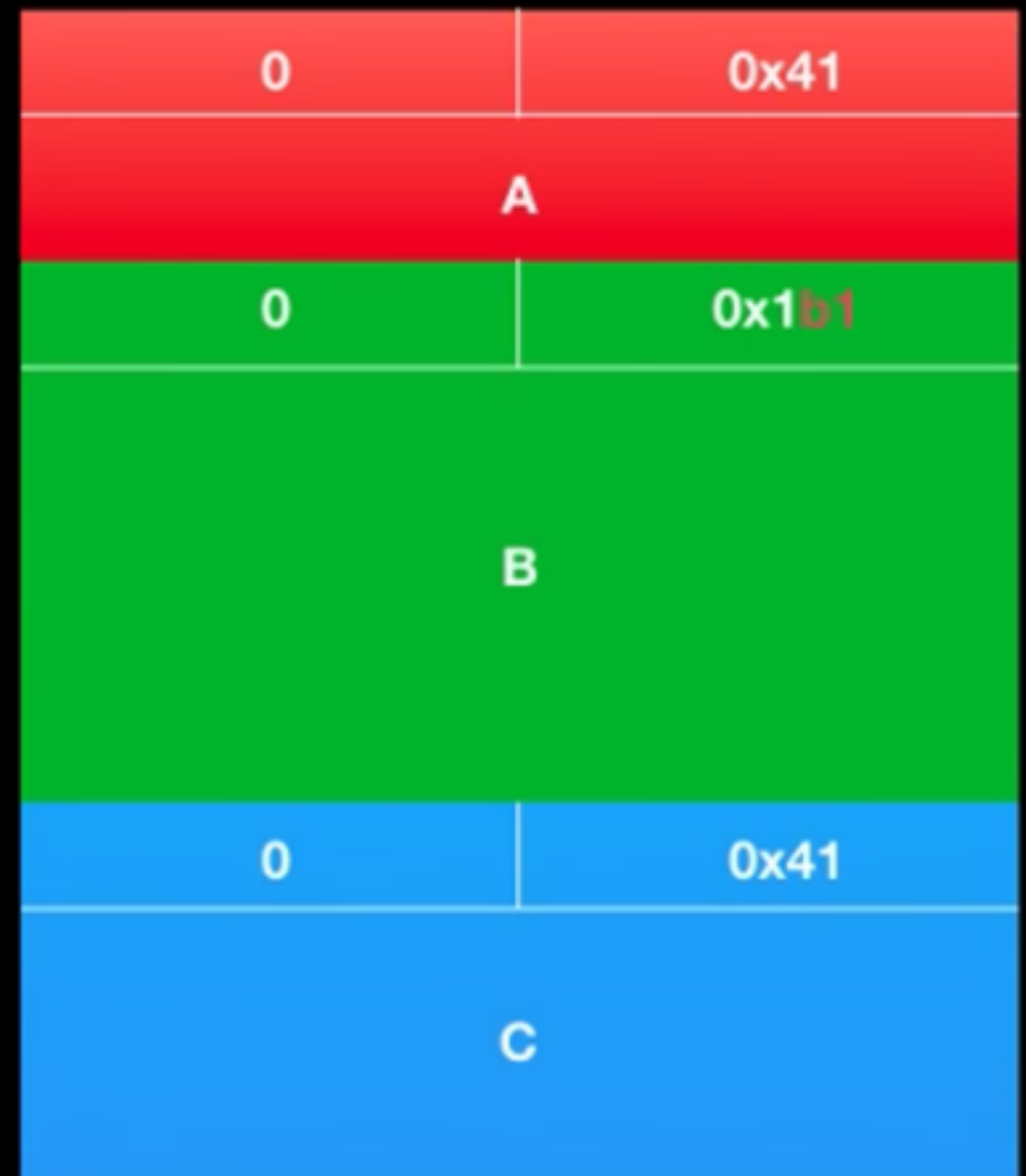
- free(B)

此時這塊 chunk
會被合併到 top
(如果 C 後面是 top)
或是加到 unsorted bin 中



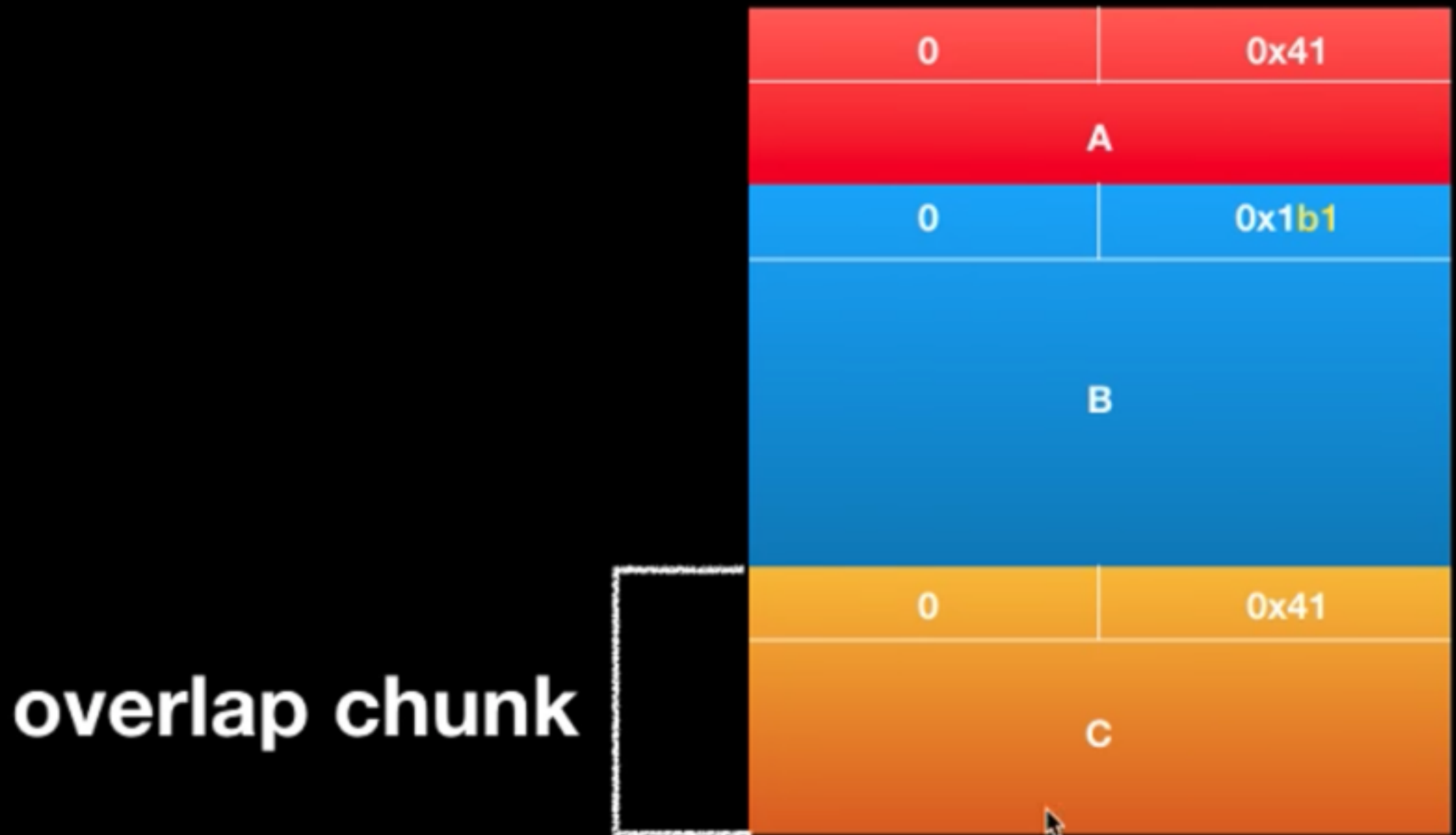
Extend the chunk

- `malloc(0x1a0)`



Extend the chunk

- 此時可以任意更改 C



Extend the chunk

- 不過跟 Shrink 比起來相對限制比較多一點，必須要可控 off-by-one 的那個 byte，Shrink 只要是 null byte 即可，相對比較常見，但也不好發現就是了

Outline

- Fastbin corruption
- Shrink the chunk
- Extend the chunk
- Unsorted bin attack

Unsorted bin attack

- 在 malloc 時，不論 unsorted bin 是否有剛好大小的 chunk，都會對 unsorted bin 的 chunk 做 unlink
 - 有剛好大小，從 unsorted bin 移除給使用者
 - 不是剛好大小，從 unsorted bin 移除，放到相對應的 bin 中

Unsorted bin attack

- 但這個 unlink 跟平常的 unlink 方式有點不同，並沒有針對 double linked list 做檢查
- 移除時會先取出 unsorted bin 最後一塊 chunk 做為 victim，然後將這一塊的 bk 指向的 chunk 的 fd 改成 unsorted bin 位置

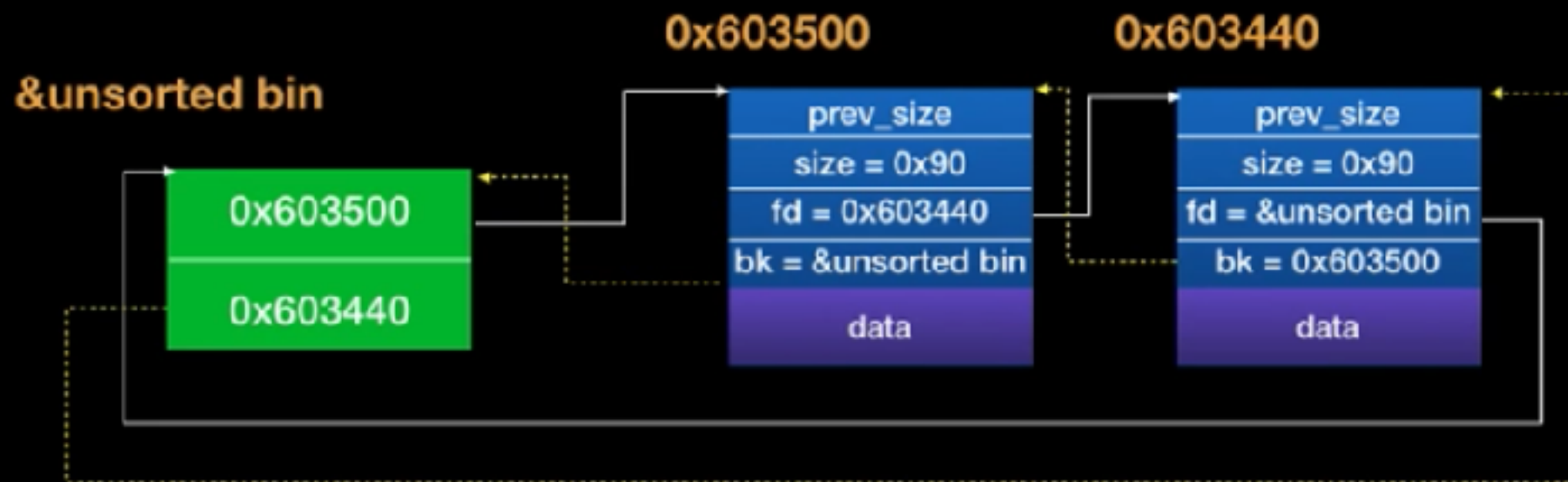
```
int iters = 0;
while ((victim = unsorted_chunks (av)->bk) != unsorted_chunks (av))
{
    bck = victim->bk;
```

```
/* remove from unsorted list */
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```

Unsorted bin attack

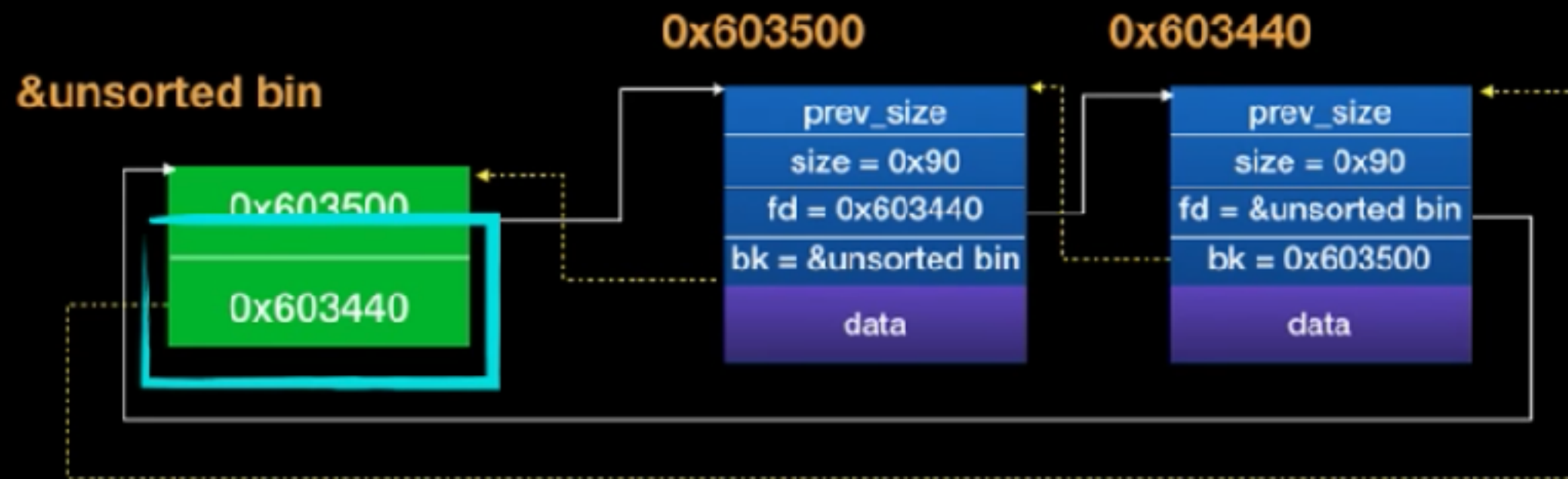
- 所以只要將最後一塊的 bk 改寫成任意位置，就可以在任意位置上寫上 unsorted bin 的位置
- 理論上會是一個很大的數字
- 通常會寫道 glibc 中 `global_max_fast` 這個變數，該變數主要用來判斷是否為 fastbin 的 chunk，如果 chunk size 小於這變數就會認為是 fastbin
- 可以配合 fastbin corruption attack

Process unsorted chunks



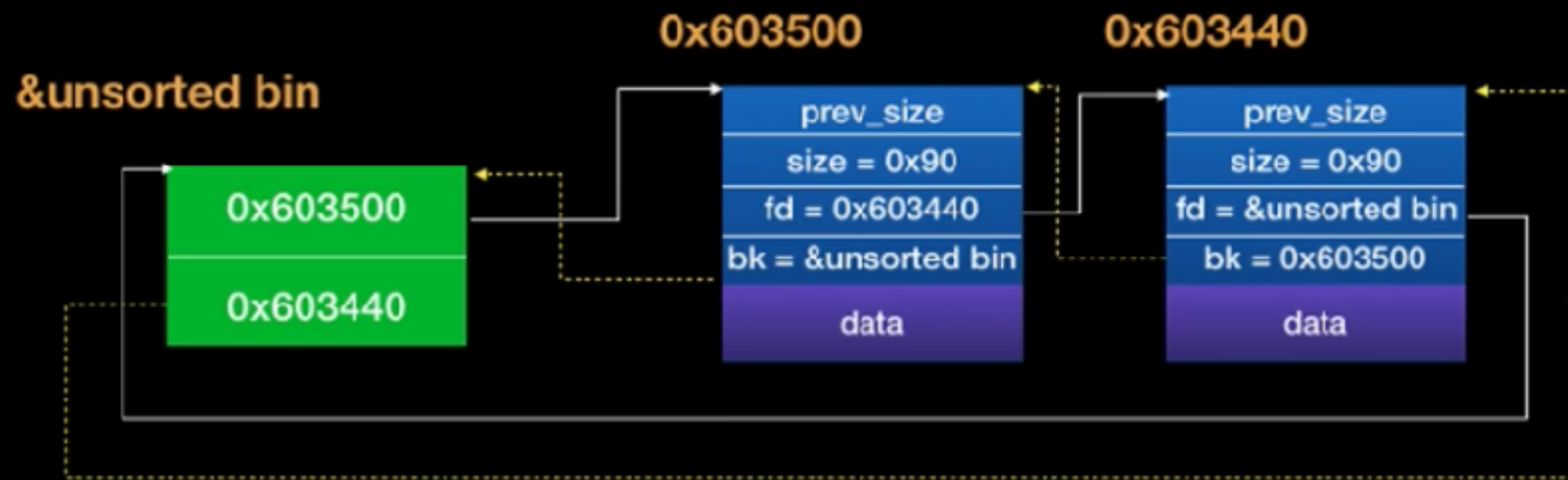
- call `malloc(0x80)`
 - remove the 0x603440 from unsorted bin
 - return 0x603450 to user

Process unsorted chunks



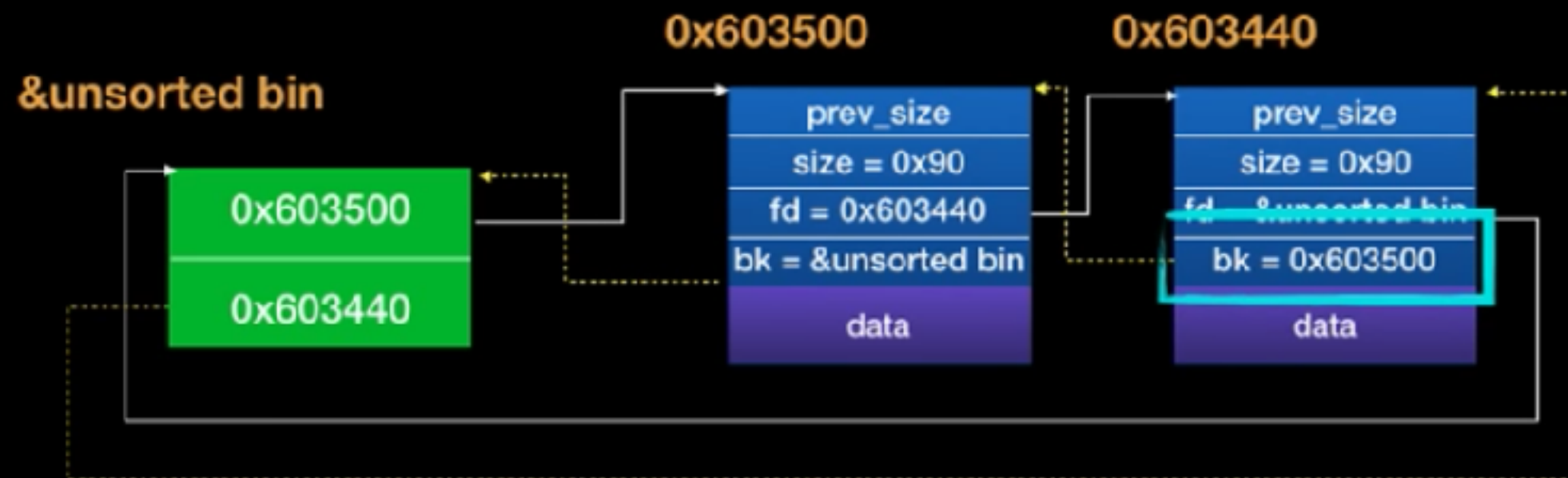
- `victim = unsorted_chunks(av)->bk`

Process unsorted chunks



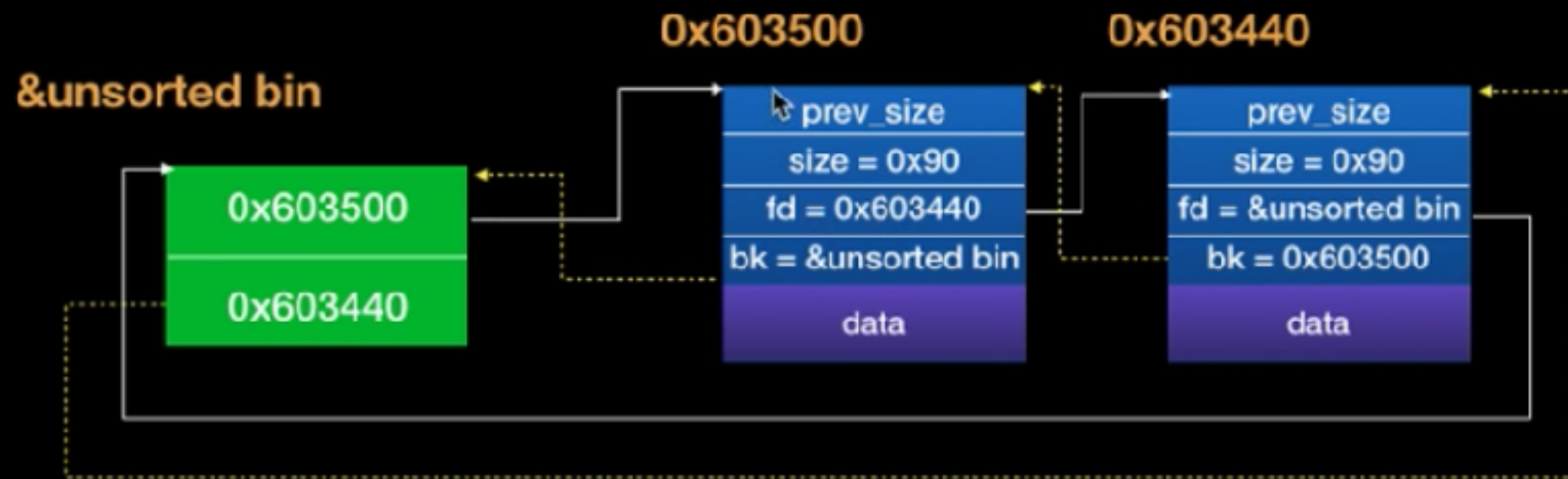
- `victim = unsorted_chunks(av)->bk`
- `victim = 0x603440`

Process unsorted chunks



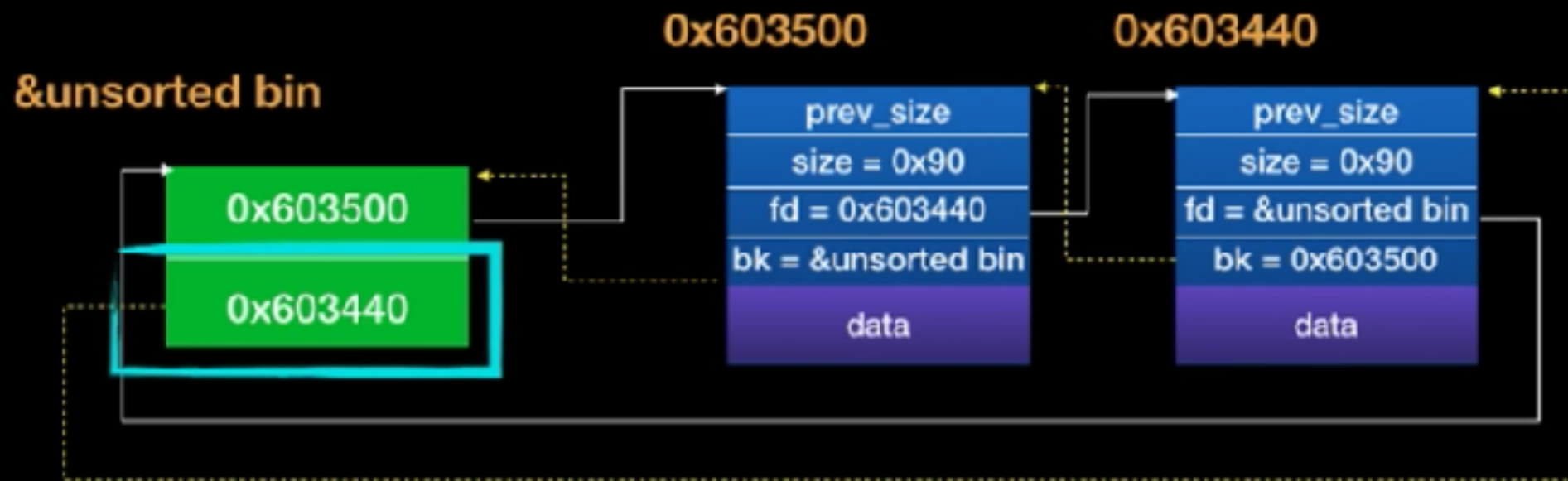
- victim = 0x603440
- bck = victim->bk

Process unsorted chunks



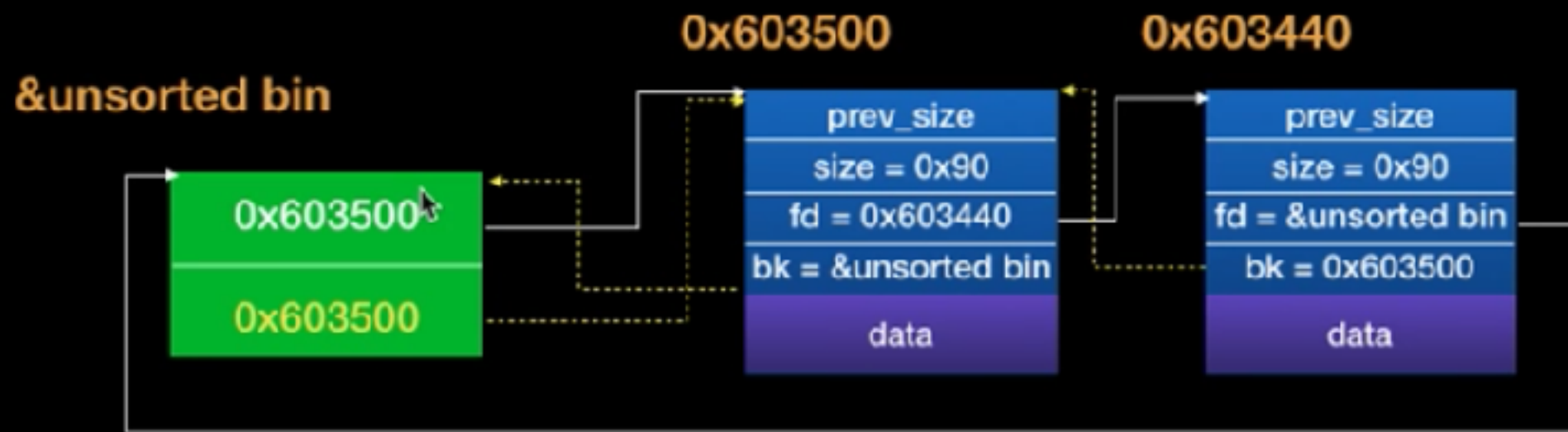
- victim = 0x603440
- bck = 0x603500

Process unsorted chunks



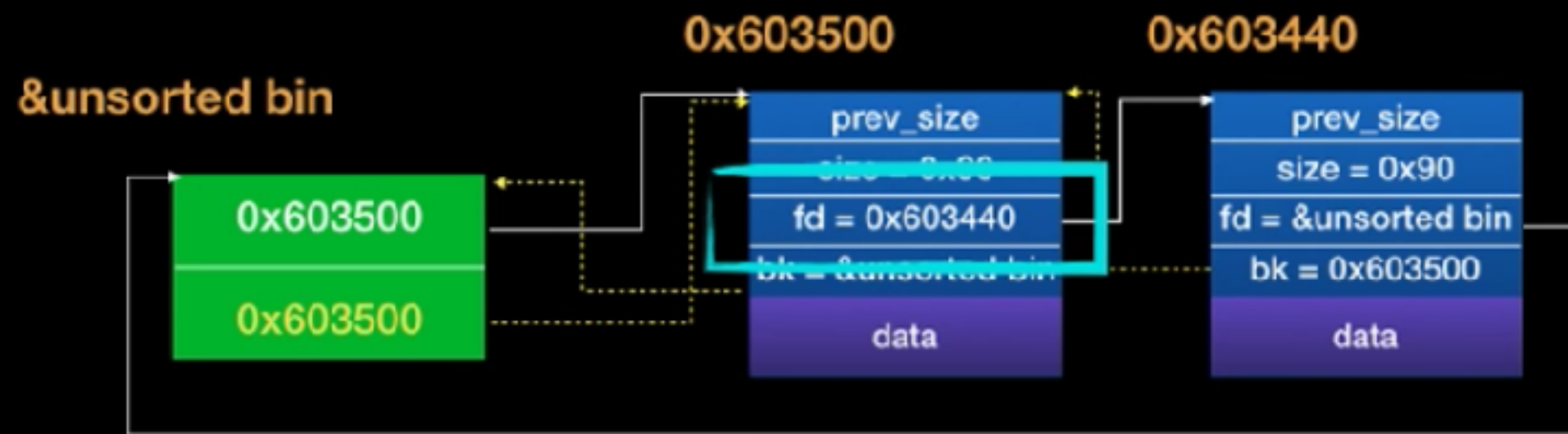
- `bck = 0x603500`
- `unsorted_chunks(av)->bk = bck`

Process unsorted chunks



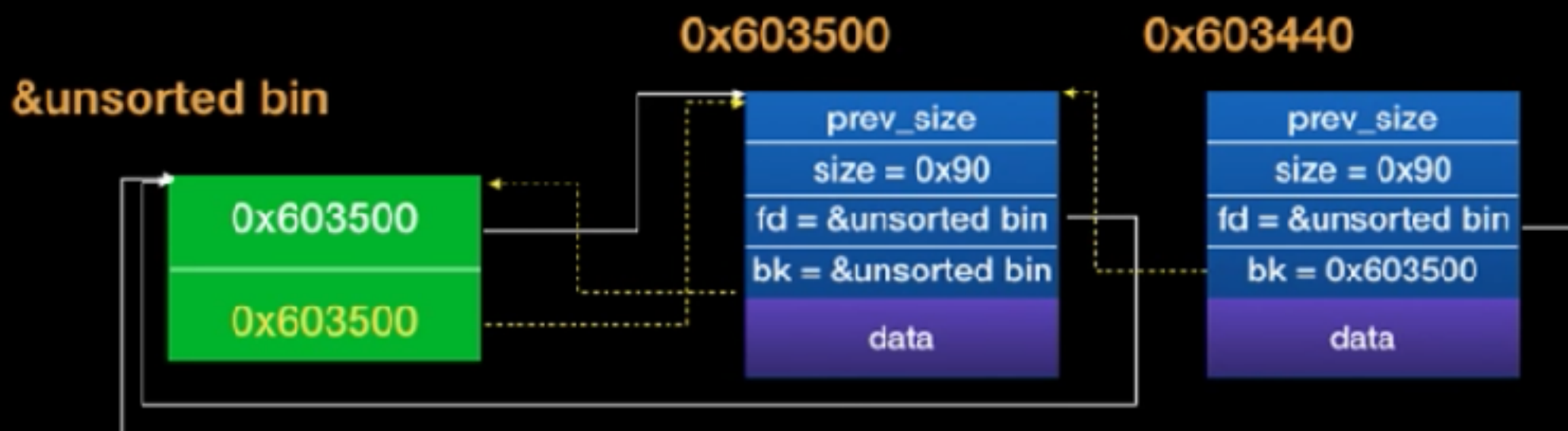
- `bck = 0x603500`
- `unsorted_chunks(av)->bk = bck`

Process unsorted chunks



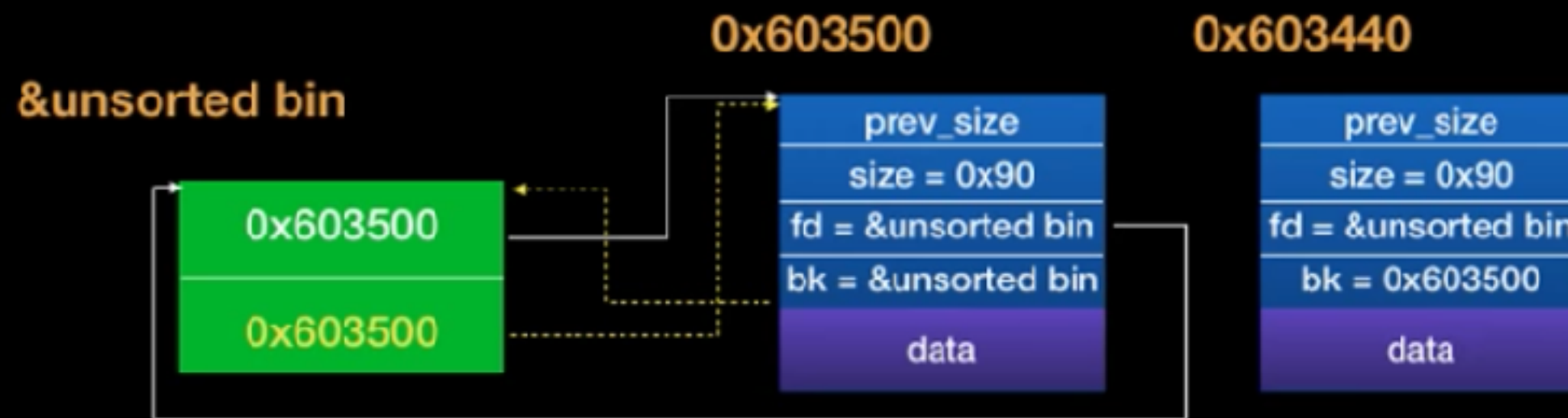
- `bck = 0x603500`
- `unsorted_chunks(av)->bk = bck`
- `bck->fd = unsorted_chunks(av)`

Process unsorted chunks



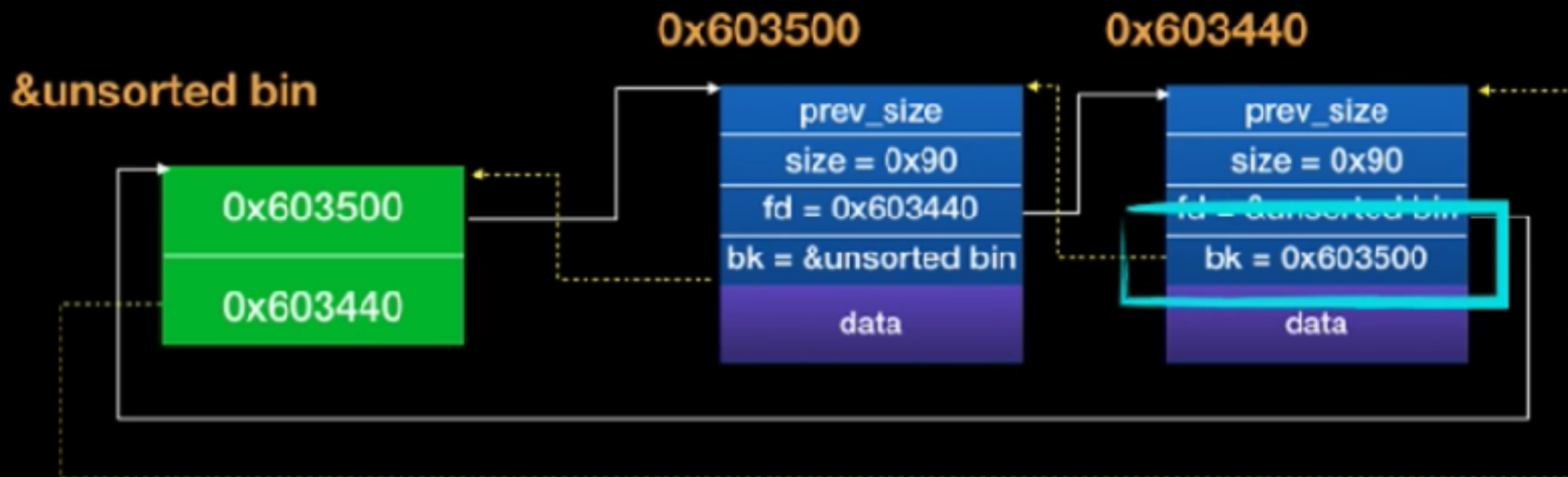
- `bck = 0x603500`
- `unsorted_chunks(av)->bk = bck`
- `bck->fd = unsorted_chunks(av)`

Process unsorted chunks



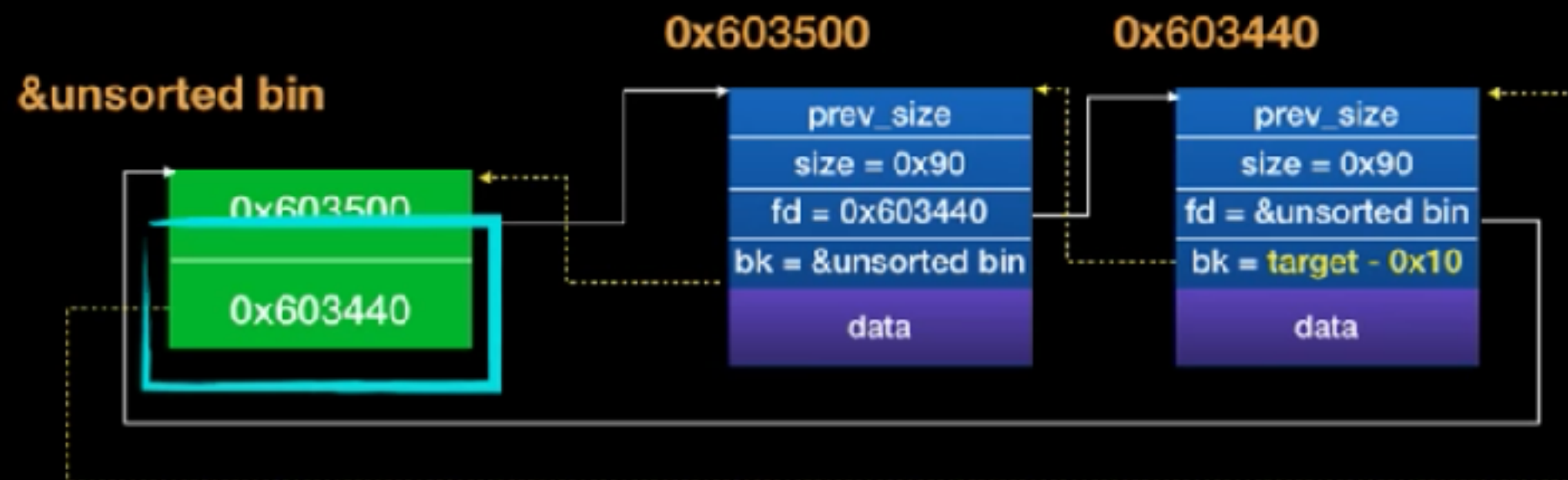
- Return 0x603450 to user
- If (nb != victim->size)
 - put 0x603440 to smallbin

Unsorted bin attack



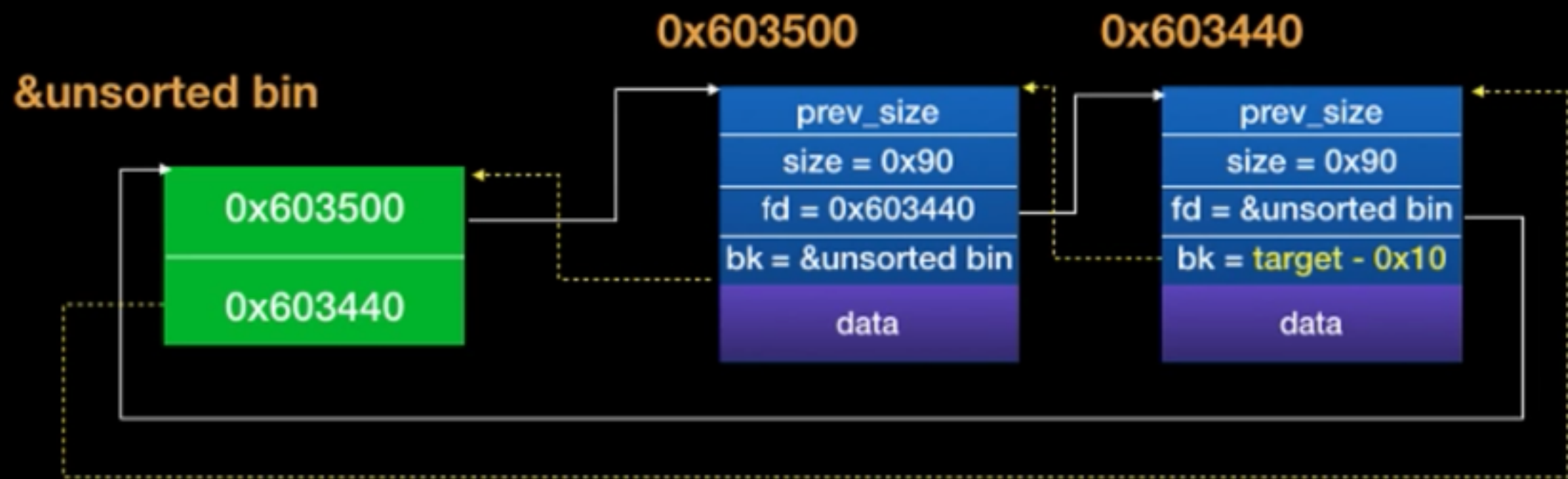
- Corruption in 0x603440

Unsorted bin attack



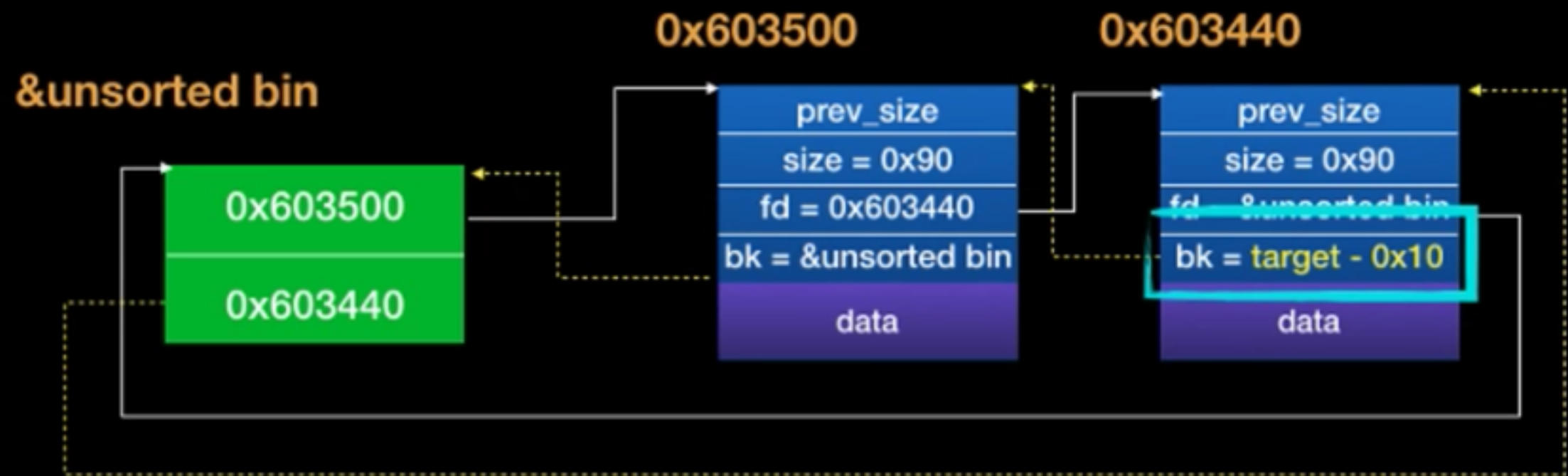
- `victim = unsorted_chunks(av)->bk`

Unsorted bin attack



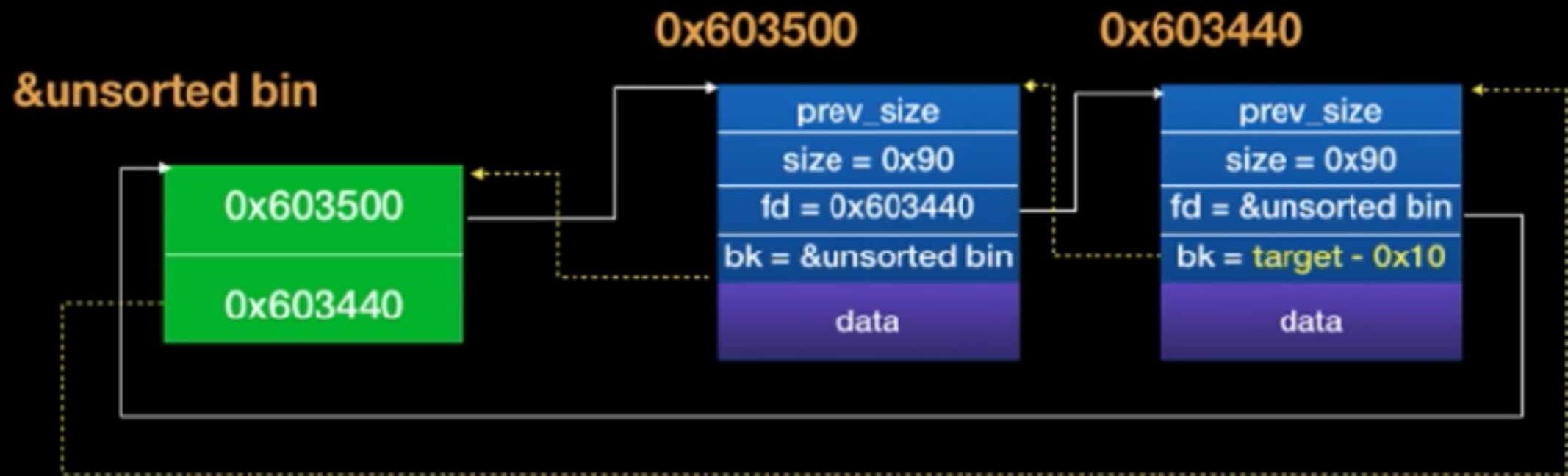
- victim = unsorted_chunks(av)->bk
- victim = 0x603440

Unsorted bin attack



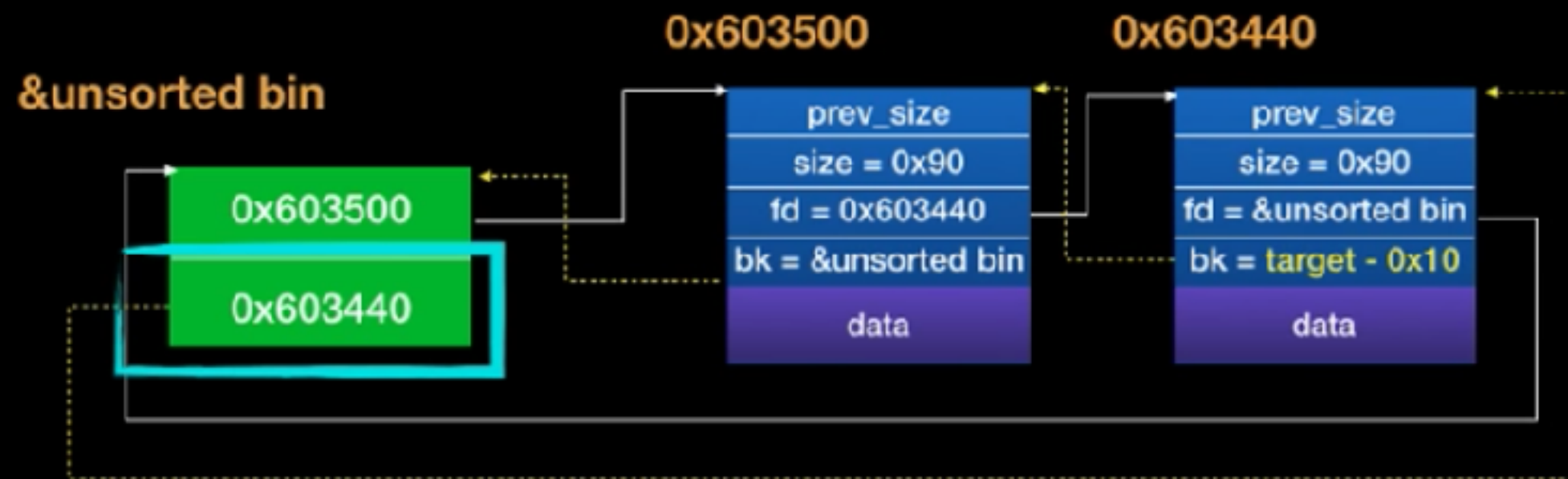
- victim = 0x603440
- bck = victim->bk

Unsorted bin attack



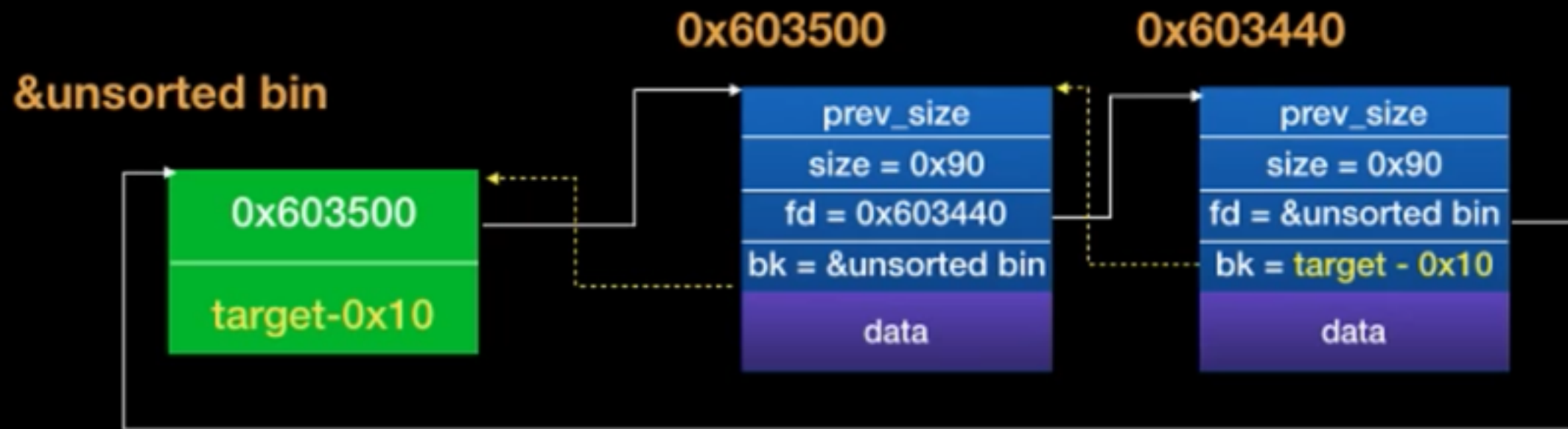
- victim = `0x603440`
- `bck = target - 0x10`

Unsorted bin attack



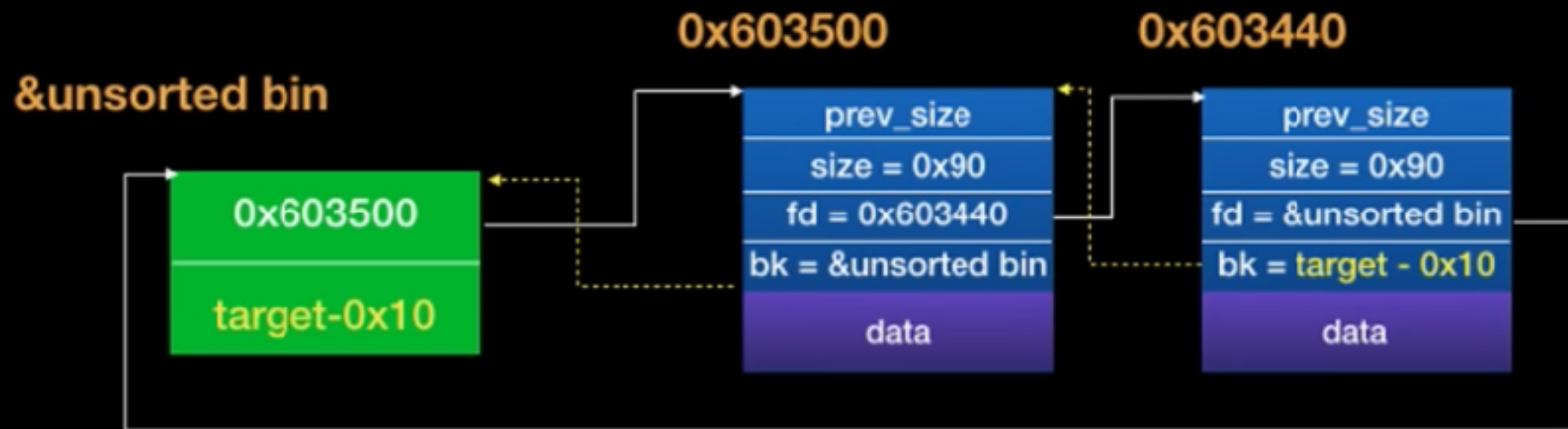
- `bck = target - 0x10`
- `unsorted_chunks(av)->bk = bck`

Unsorted bin attack



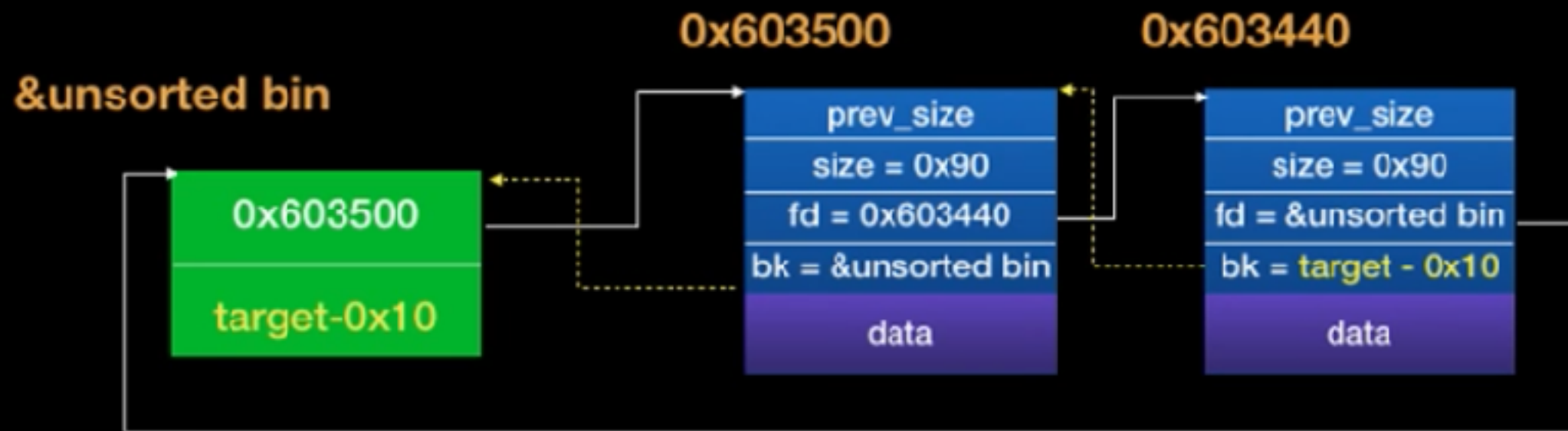
- `bck = target - 0x10`
- `unsorted_chunks(av)->bk = bck`

Unsorted bin attack



- `bck = target - 0x10`
- `unsorted_chunks(av)->bk = bck`
- `bck->fd = unsorted_chunks(av)`

Unsorted bin attack



- `target = unsorted_chunks(av)`
- `if target == global_fast_max`
 - `global_fast_max = unsorted_chunks(av)`