Tcache in glibc

New mechanism of libc malloc

Angelboy

Outline

- New Structure
- Tcache
- Make Heap Exploitation Easy Again
  - Weakness in tcache

Overview

- 增進記憶體管理的效能
- 實作在 glibc 2.26
  - Ubuntu 17.10 之後

Outline

- New Structure
- Tcache
- Make Heap Exploitation Easy Again
  - Weakness in tcache

New Structure

- tcache_entry
  - 類似 fastbin 中的 chunk
  - 在 freed 時，用 linked list 串起來，指向的是 chunk data 的部分

```
typedef struct tcache_entry
{
  struct tcache_entry *next;
} tcache_entry;
```

New Structure

- tcache_perthread_struct
  - 一個 thread 一個 tcache_perthread_struct
  - 在該 thread 第一次 malloc 時初始化
  - Count : 對應每個 tcache 中，chunk 的數量

```
typedef struct tcache_perthread_struct
{
  char counts[TCACHE_MAX_BINS];
  tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;
```

New Structure

- tcache_perthread_struct
  - 根據大小分成多個不同的 tcache
    - 只要是 smailbin 範圍大小的 chunk 都會使用 tcache

```
typedef struct tcache_perthread_struct
{
  char counts[TCACHE_MAX_BINS];
  tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;
```

Outline

- New Structure
- Tcache
- Make Heap Exploitation Easy Again
  - Weakness in tcache

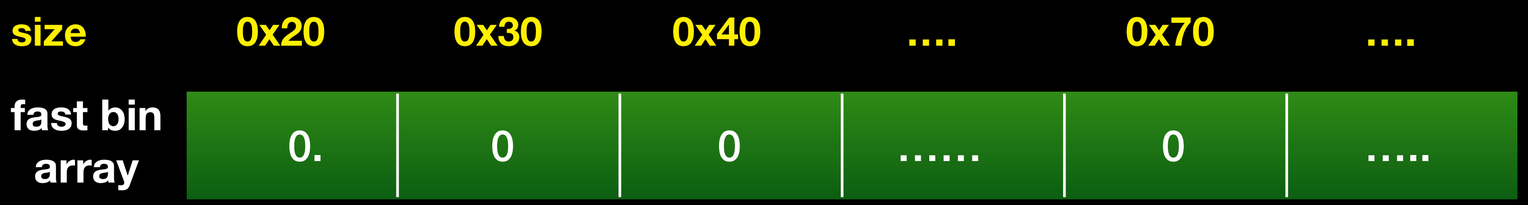
tcache

- 第一次 malloc 時，會先 malloc 一塊記憶體區塊，用來存放 tcache_perthread_struct
- 在之後 small bin chunk size 的 malloc 都會先以存放在 tcache 中的為主，幾乎與 fastbin 相似
  - 比較不同的事 fastbin 中的 fd 是指向 chunk 開頭位置
    而 tcache 中則是指向 user data 的部分，也就是 header 之後

tcache

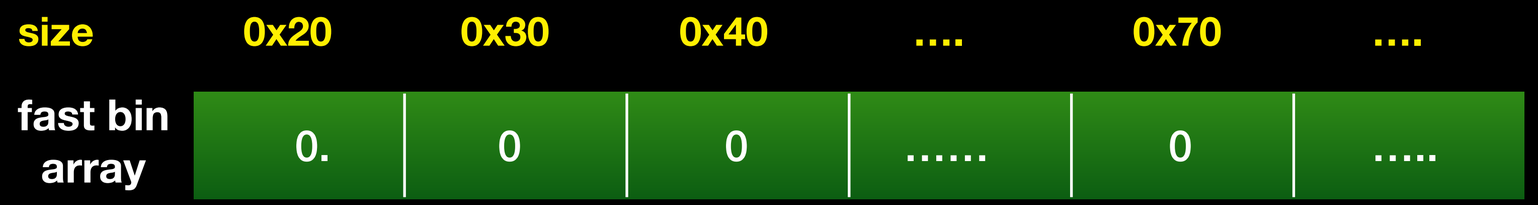
- 當 free 掉一塊 chunk 時，且 chunk 小於 small bin size 時
  - 以往來說都會先放到 fastbin 或是 unsorted bin 中
  - 在有 tcache 之後
    - 會先放到相對應的 tcache 中，直到 tcache 被塞滿到 7 個
    - 塞滿之後，其他的 chunk 就跟以前一樣放到 fastbin 或是 unsorted bin中
    - tcache 中的 chunk 不會合併（不取消 inused bit)

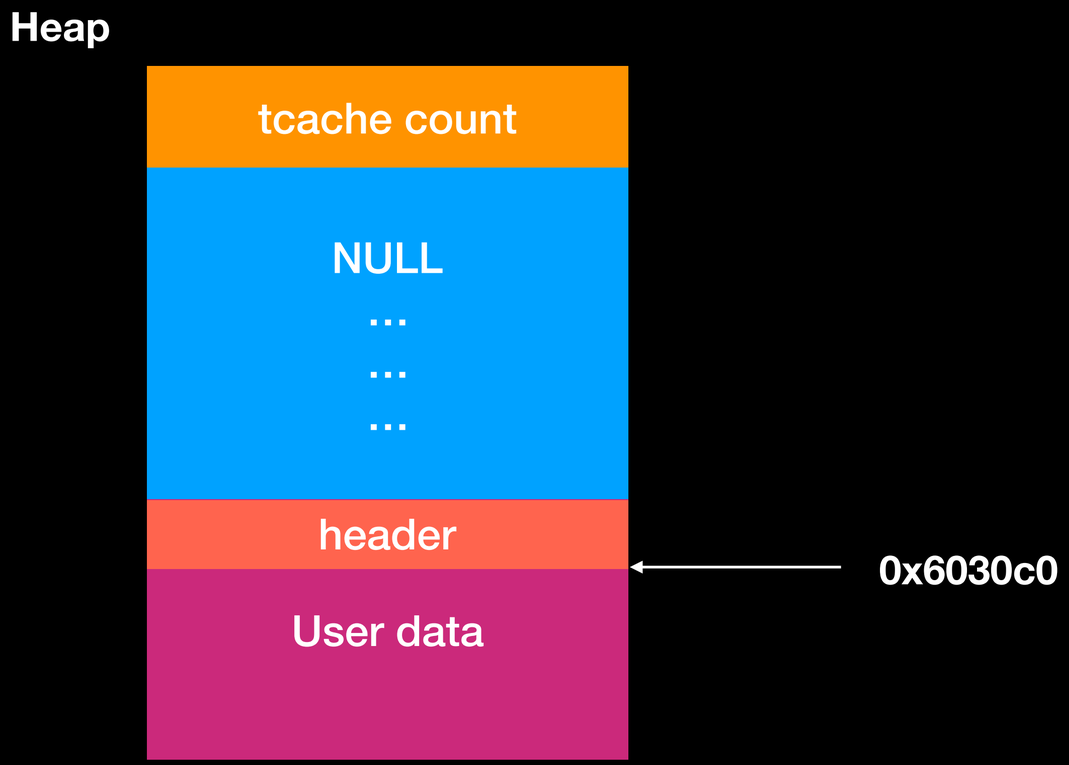
tcache



First time malloc(0x60)

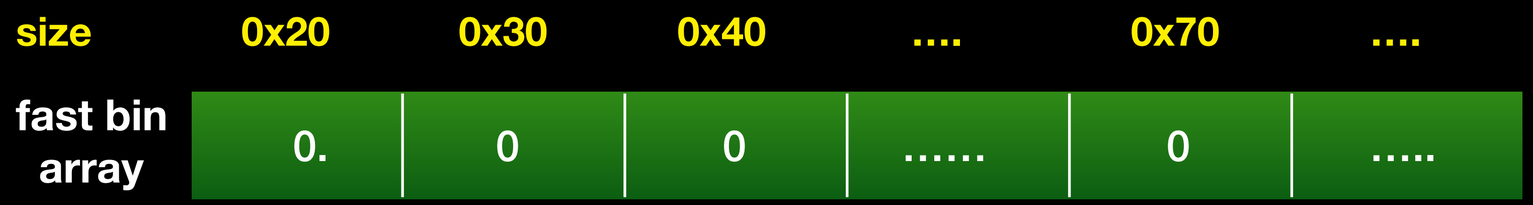
tcache

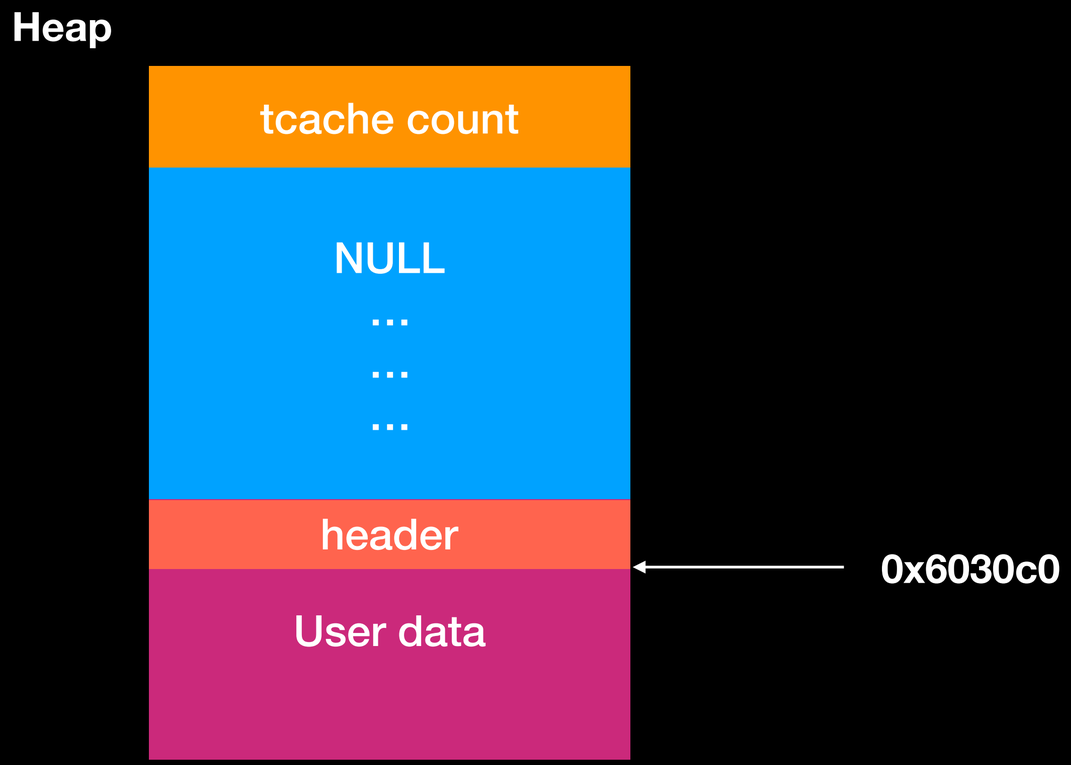




First time malloc(0x60)

tcache



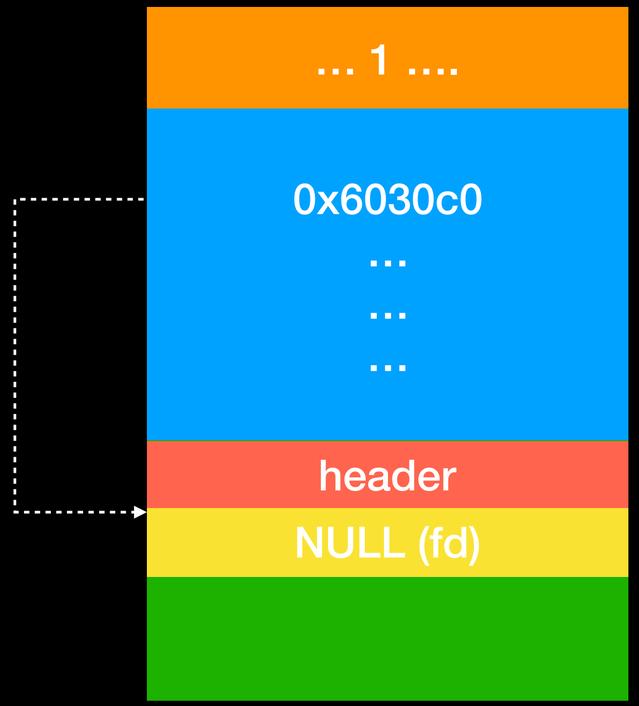


free(0x6030c0)

tcache

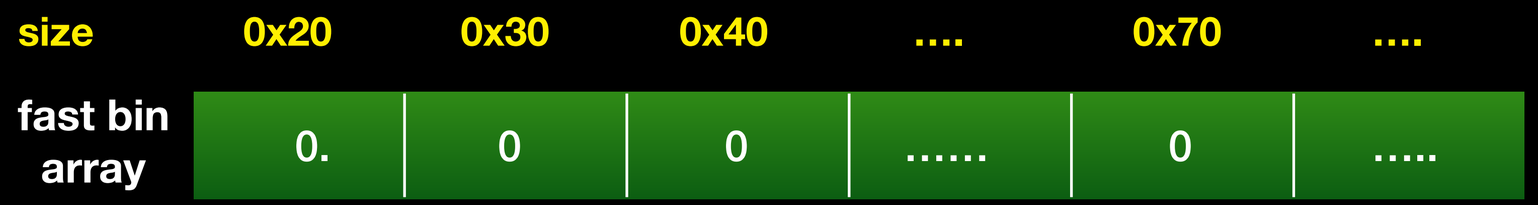
size	0x20	0x30	0x40	....	0x70	....
fast bin array	0.	0	0	.....	0	.....

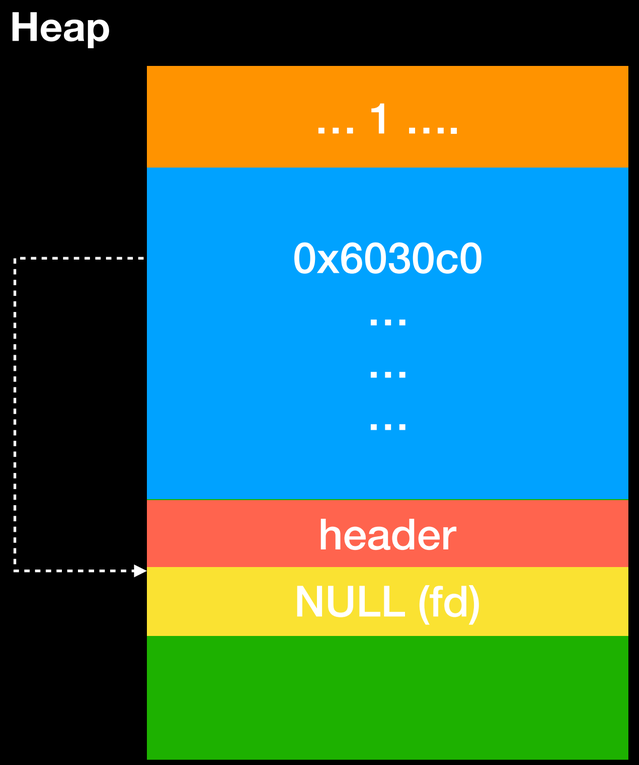
Heap



free(0x6030c0)

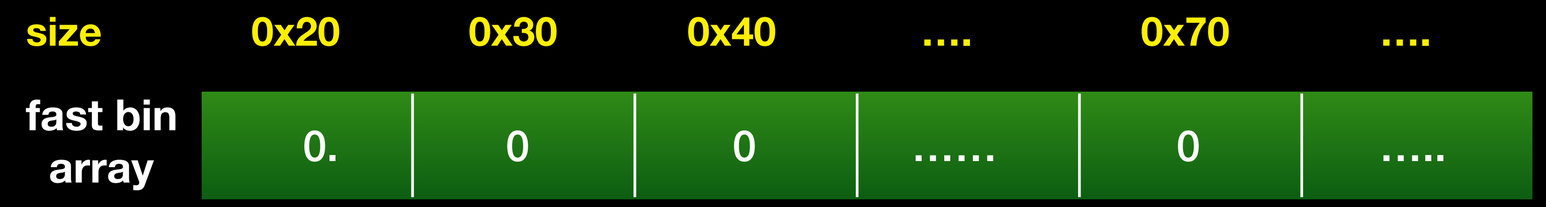
tcache

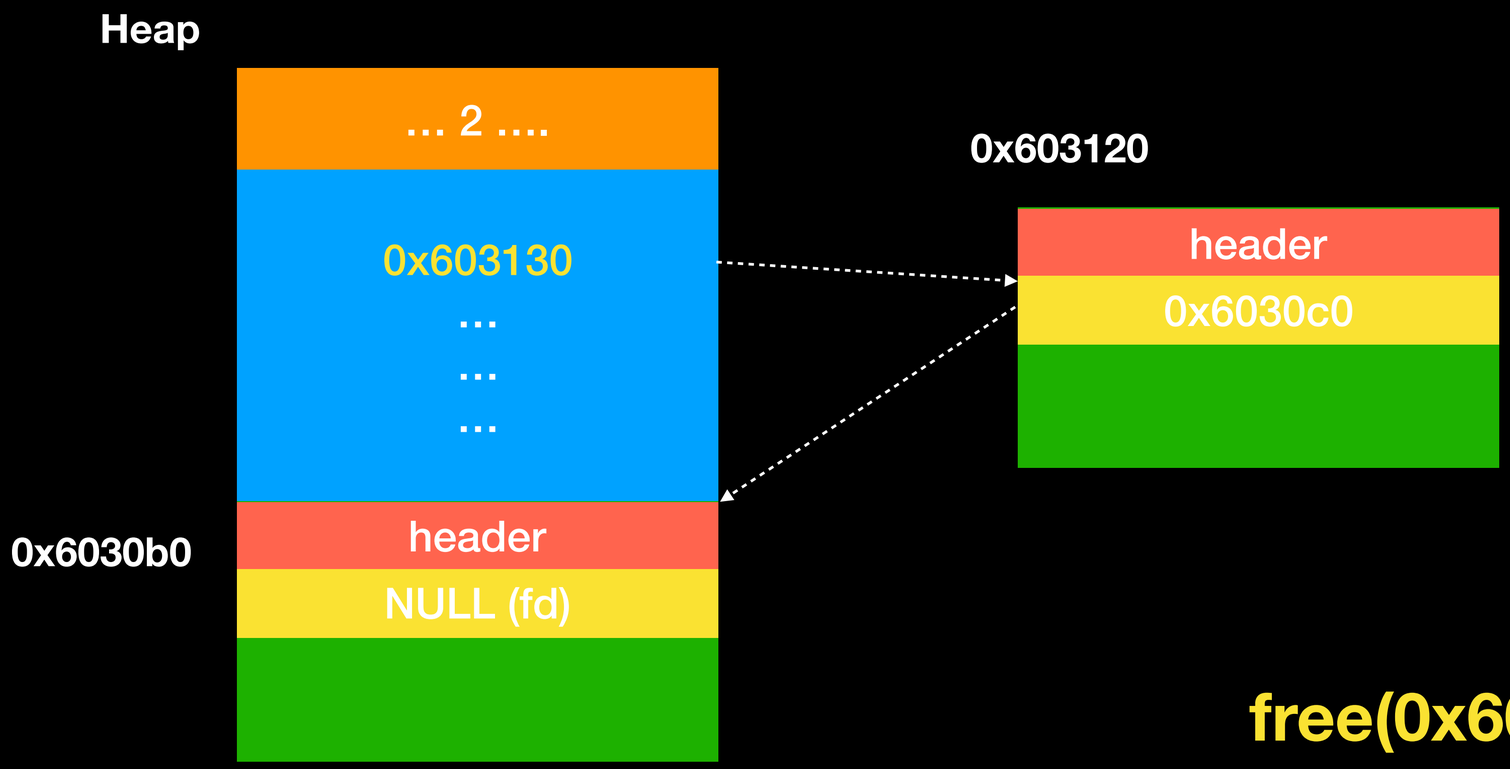




free(0x603130) (size:0x70)

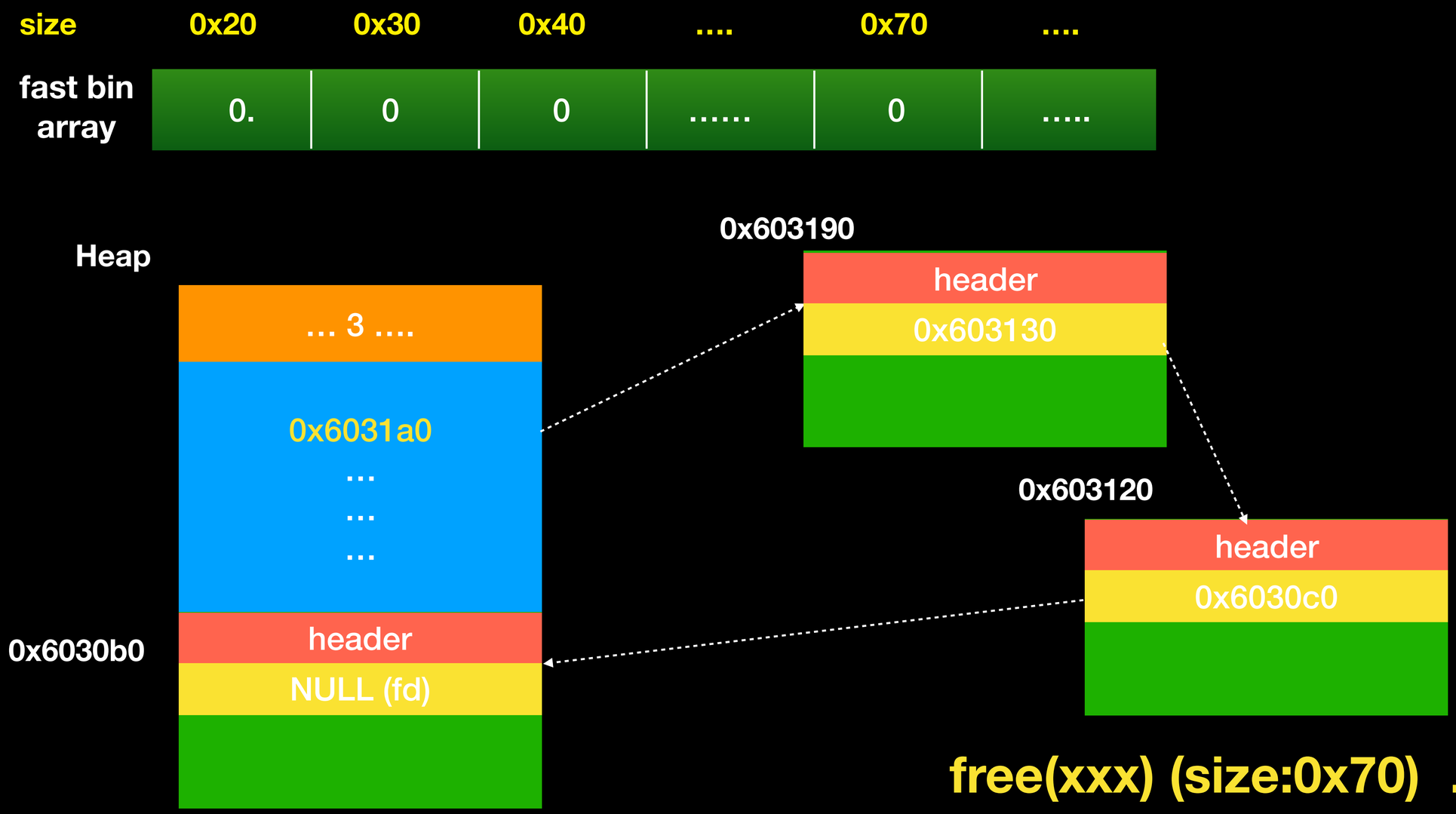
tcache



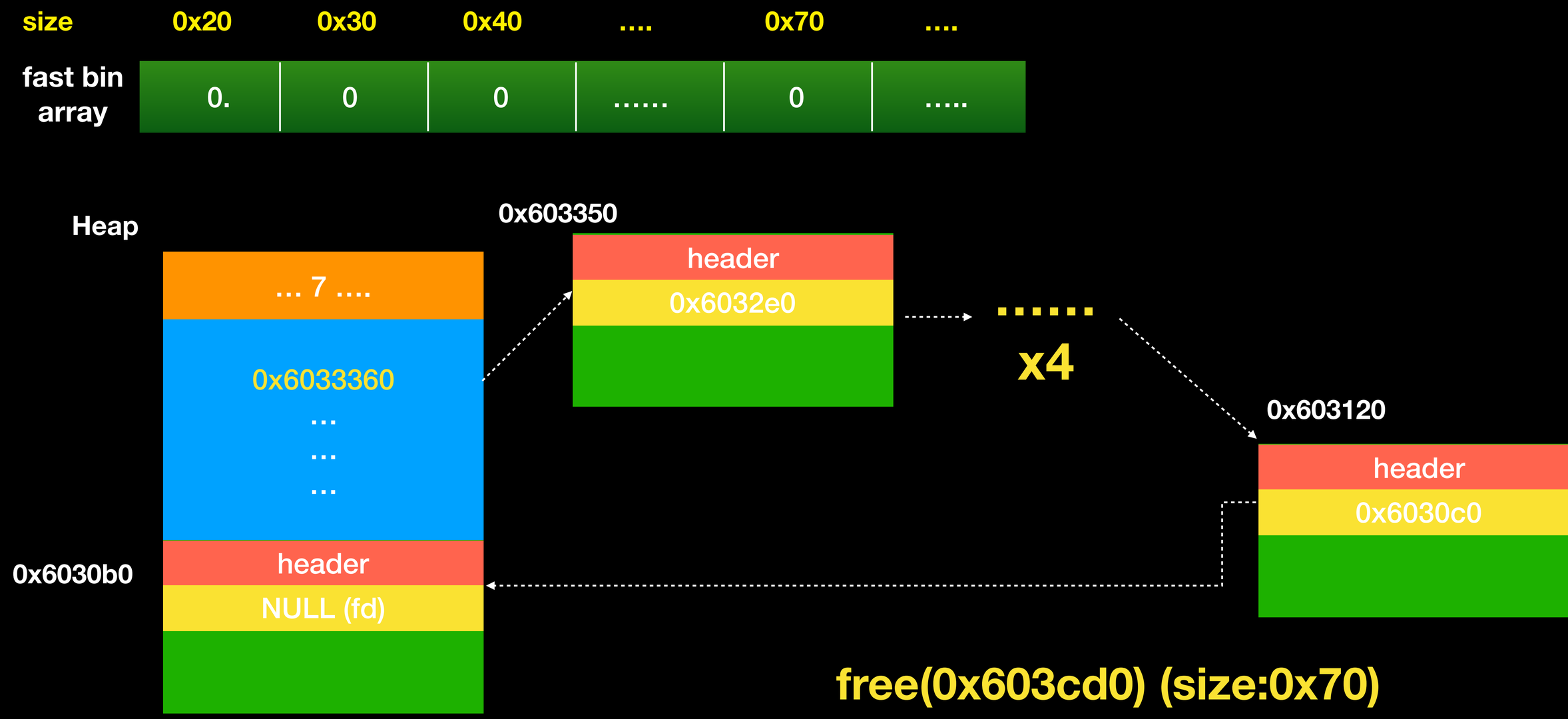


free(0x6031a0) (size:0x70)

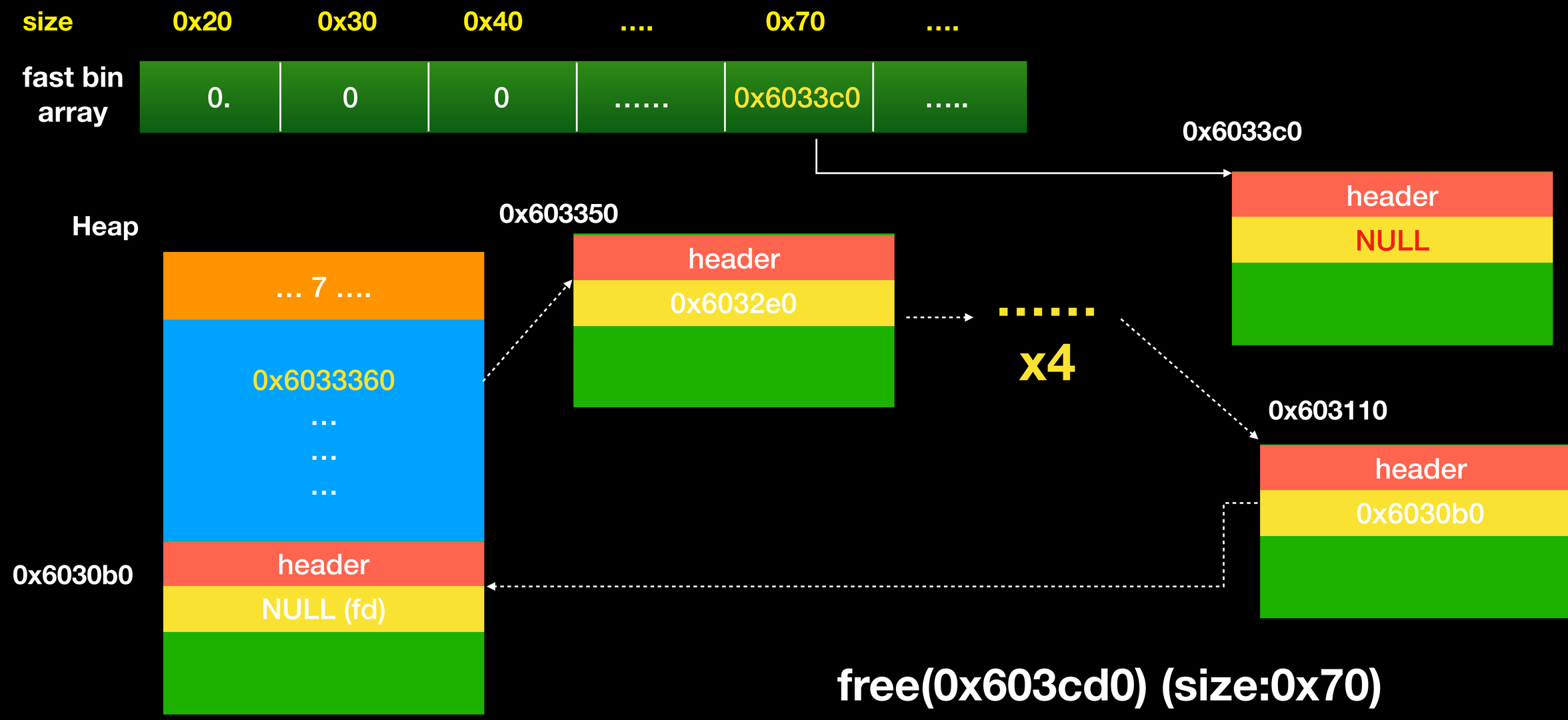
tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0
.....
Heap
0x603190
header
0x603130
... 3 ....
0x6031a0
...
...
...
0x603120
header
0x6030c0
0x6030b0
header
NULL (fd)
free(xxx) (size:0x70) ……

tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0
.....
Heap
0x603350
... 7 ....
0x6033360
...
...
...
header
0x6032e0
.....
x4
0x603120
header
0x6030c0
0x6030b0
header
NULL (fd)
free(0x603cd0) (size:0x70)

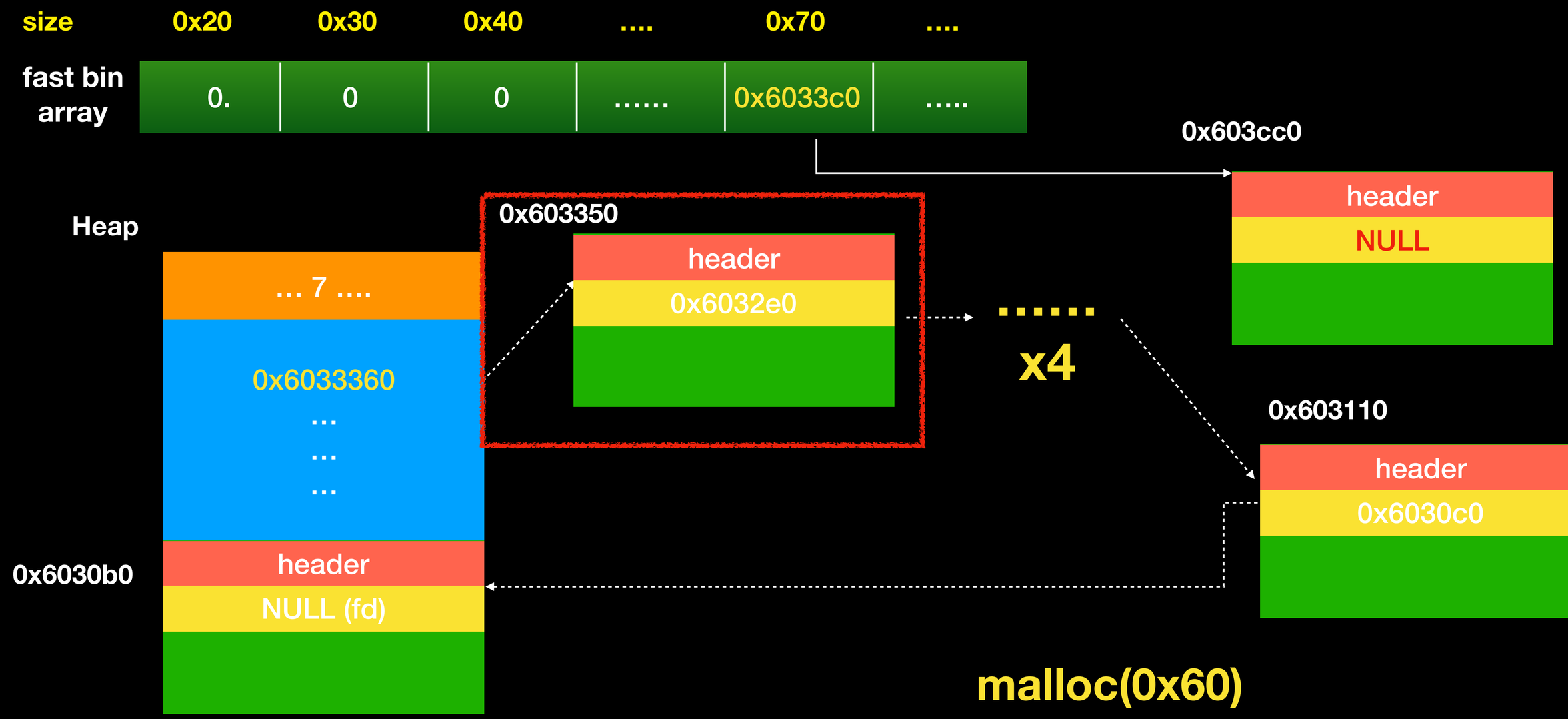
tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0x6033c0
.....
0x6033c0
header
NULL
Heap
0x603350
header
0x6032e0
.....
x4
... 7 ....
0x6033360
...
...
...
0x603110
header
0x6030b0
0x6030b0
header
NULL (fd)
free(0x603cd0) (size:0x70)

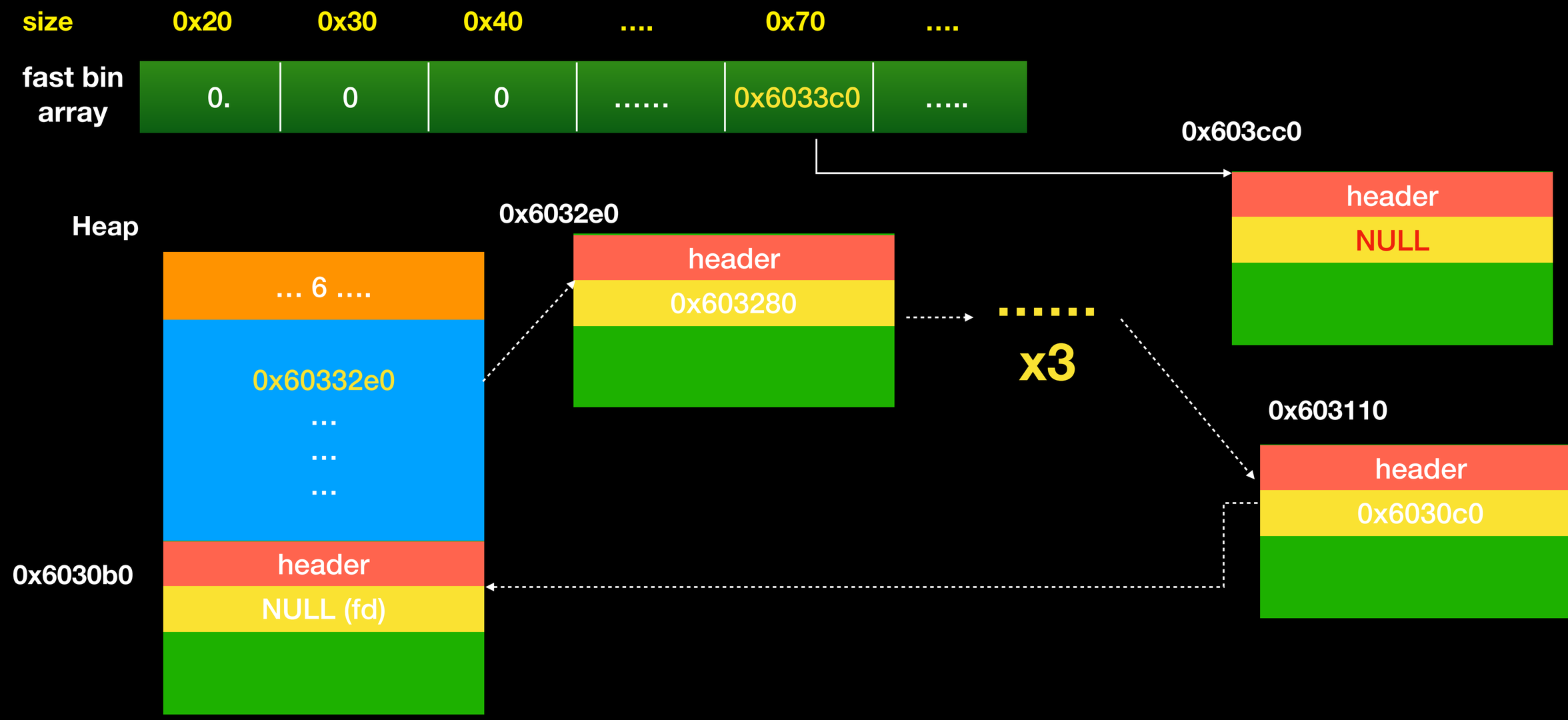
tcache

- Malloc 時
  - 優先從 tcache 取出，直到該 tcache 為空，才會從原本的 bin 開始找
  - Tcache 為空時，如果 fastbin/smallbin/unsorted bin 有剛好 size 的 chunk 時，會先將該 fastbin/smallbin/unsroted bin 中的 chunk 填補至 tcache 中，直到填滿為止，再從 tcache 相對應的 tcache 中取出
    - 所以在 bin 中，與在 tcache 中的順序會反過來
      - 但在 fastbin 中第一個會先作為 victim 後面的順序才是反的

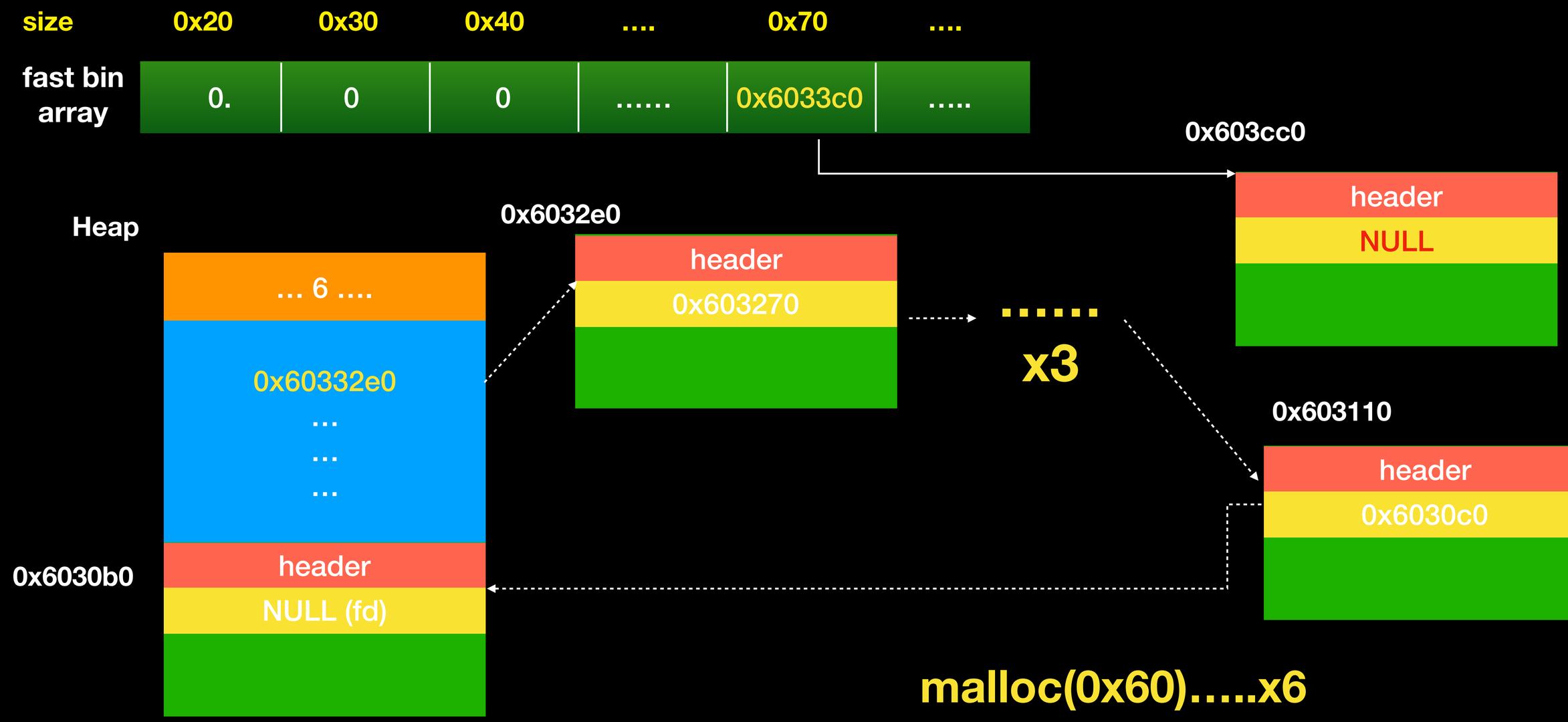
tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0x6033c0
.....
0x603cc0
header
NULL
Heap
0x603350
header
0x6032e0
.....
x4
... 7 ....
0x6033360
...
...
...
0x603110
header
0x6030c0
0x6030b0
header
NULL (fd)
malloc(0x60)

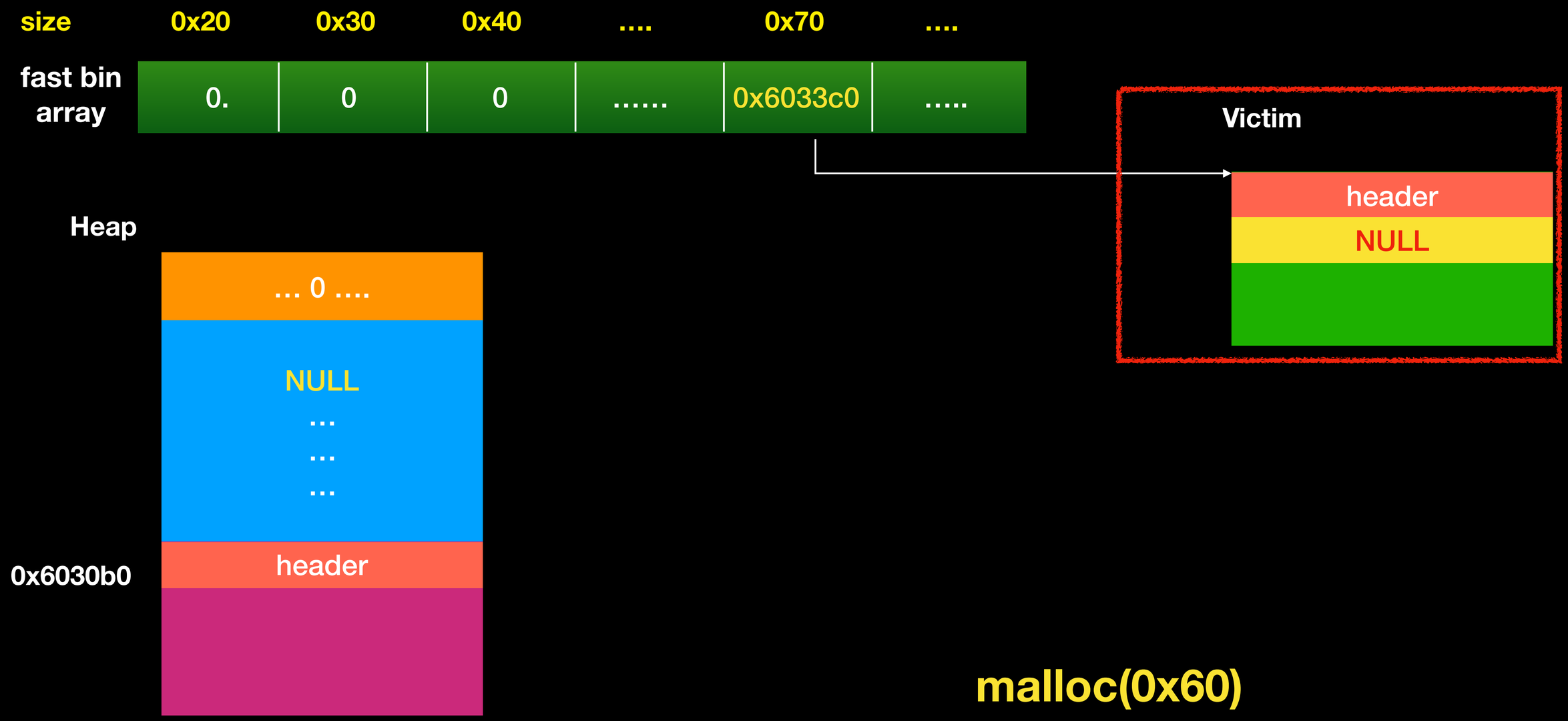
tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0x6033c0
.....
0x603cc0
header
NULL
Heap
0x6032e0
header
0x603280
.....
x3
... 6 ....
0x60332e0
...
...
...
0x603110
header
0x6030c0
0x6030b0
header
NULL (fd)

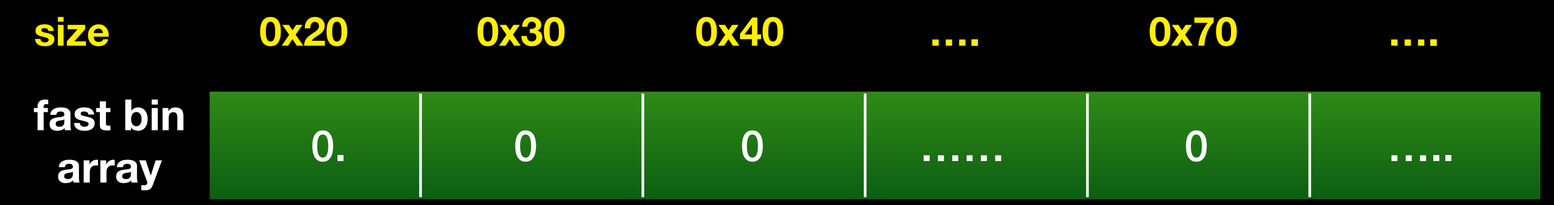
tcache

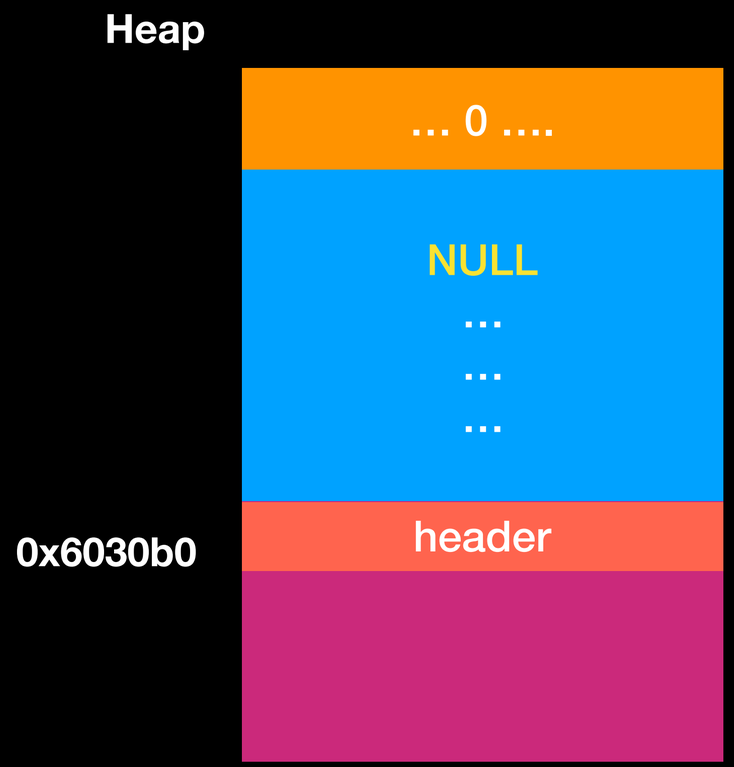

size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0x6033c0
.....
0x603cc0
header
NULL
Heap
0x6032e0
header
0x603270
.....
x3
... 6 ....
0x60332e0
...
...
...
0x603110
header
0x6030c0
0x6030b0
header
NULL (fd)
malloc(0x60).....x6

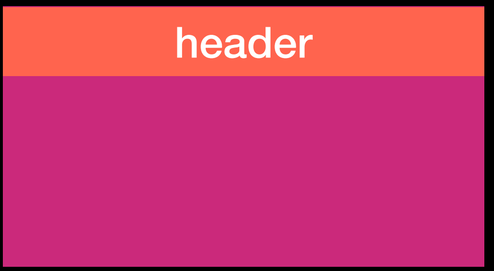
tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0x6033c0
.....
Victim
header
NULL
Heap
... 0 ....
NULL
...
...
...
0x6030b0
header
malloc(0x60)

tcache

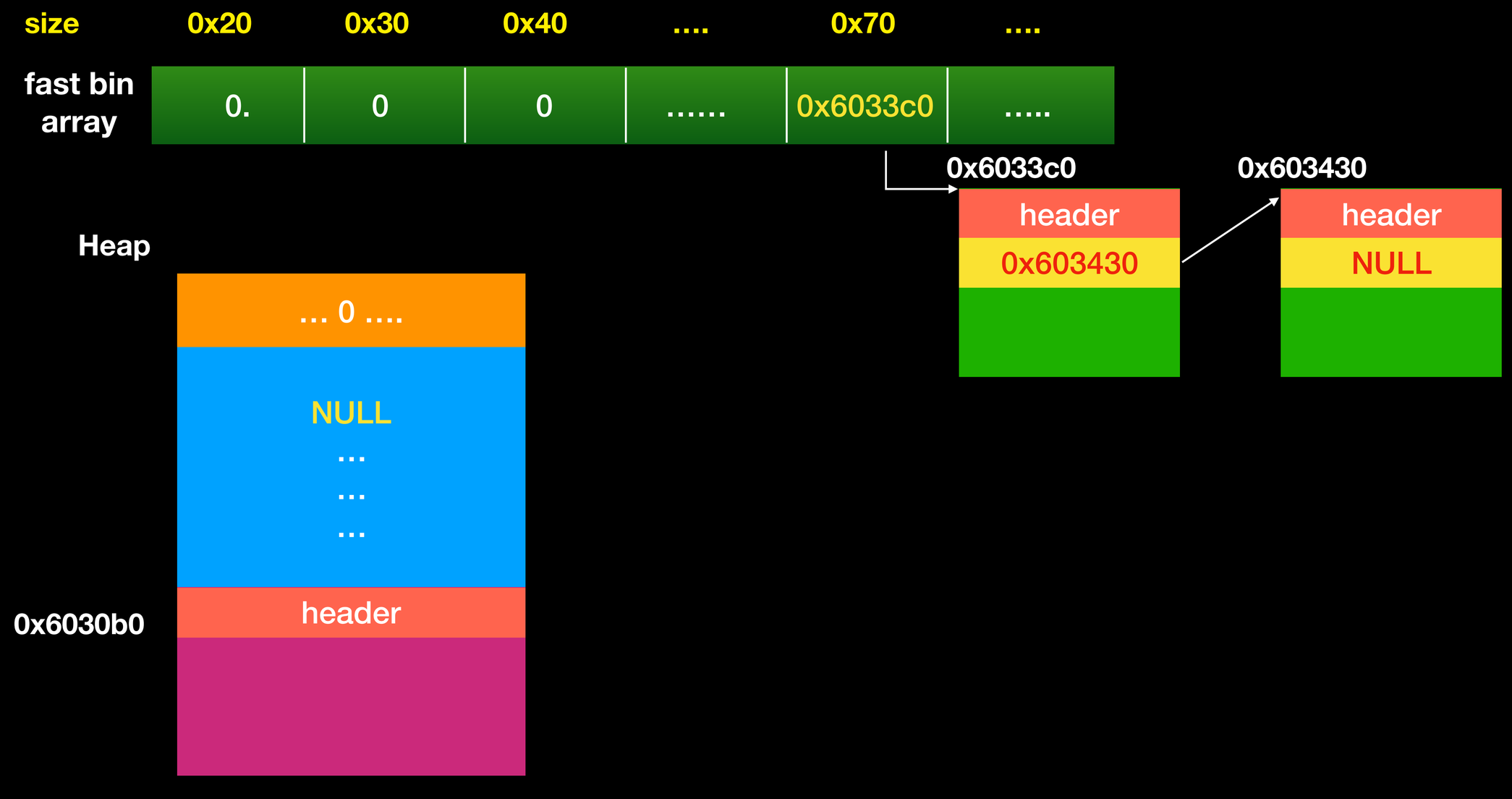




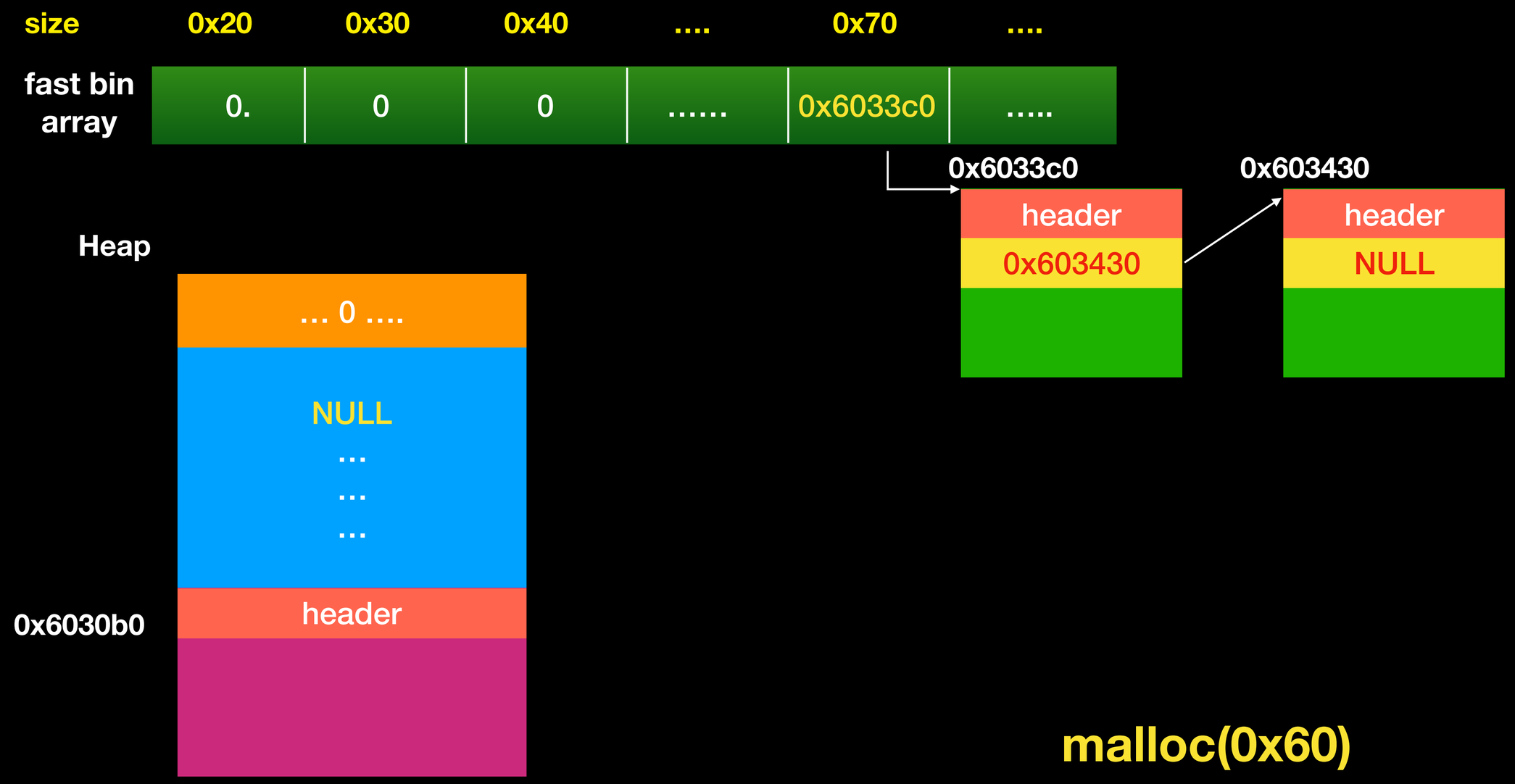


Return to user

tcache

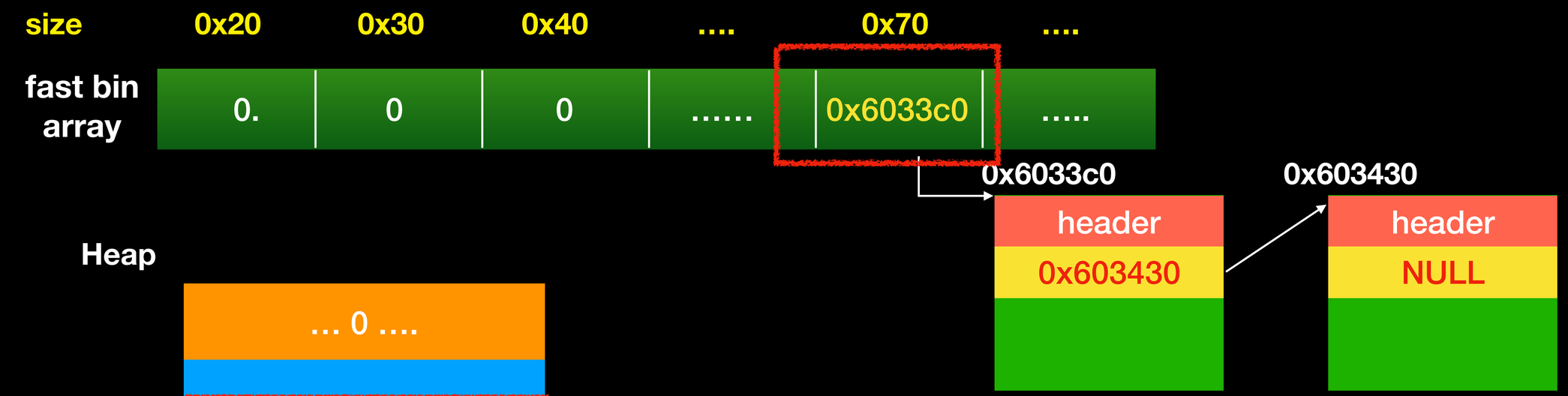

size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0x6033c0
.....
0x6033c0
header
0x603430
0x603430
header
NULL
Heap
... 0 ....
NULL
...
...
...
0x6030b0
header

tcache



malloc(0x60)

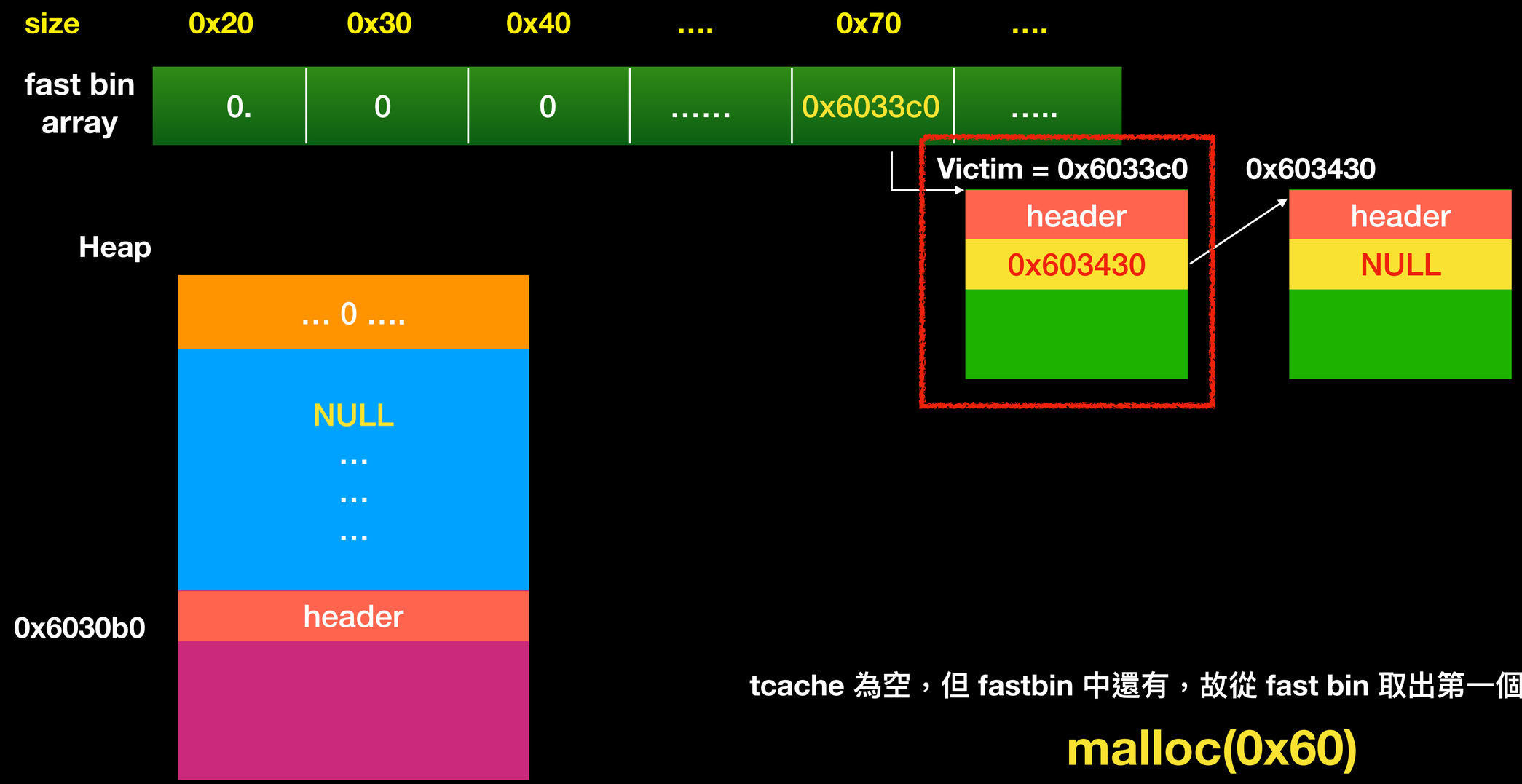
tcache



tcache 為空，但 fastbin 中還有，故從 fast bin 取出第一個並填充

malloc(0x60)

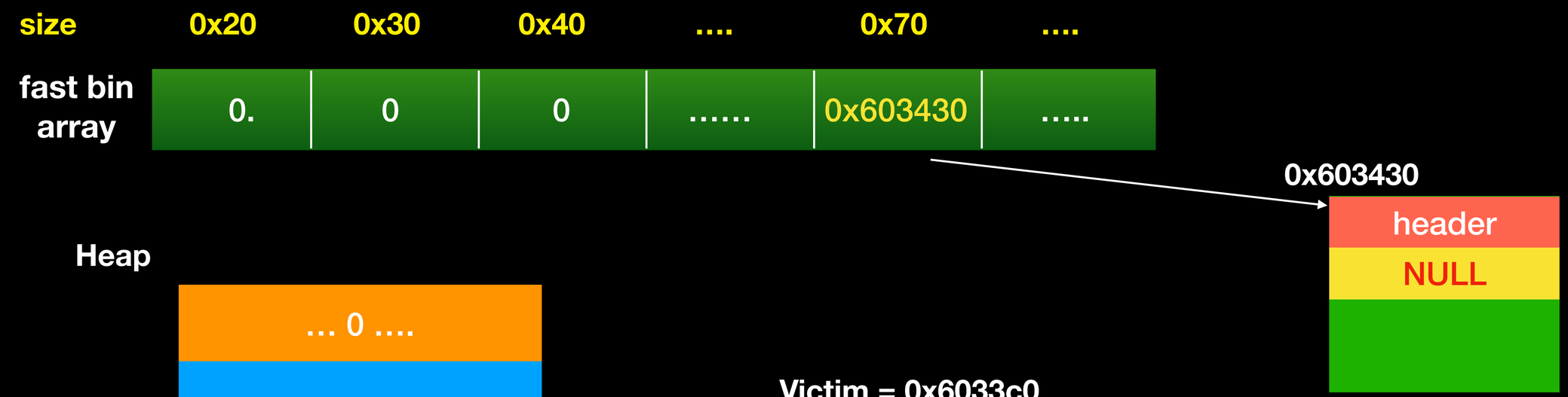
tcache



tcache 為空，但 fastbin 中還有，故從 fast bin 取出第一個並填充

malloc(0x60)

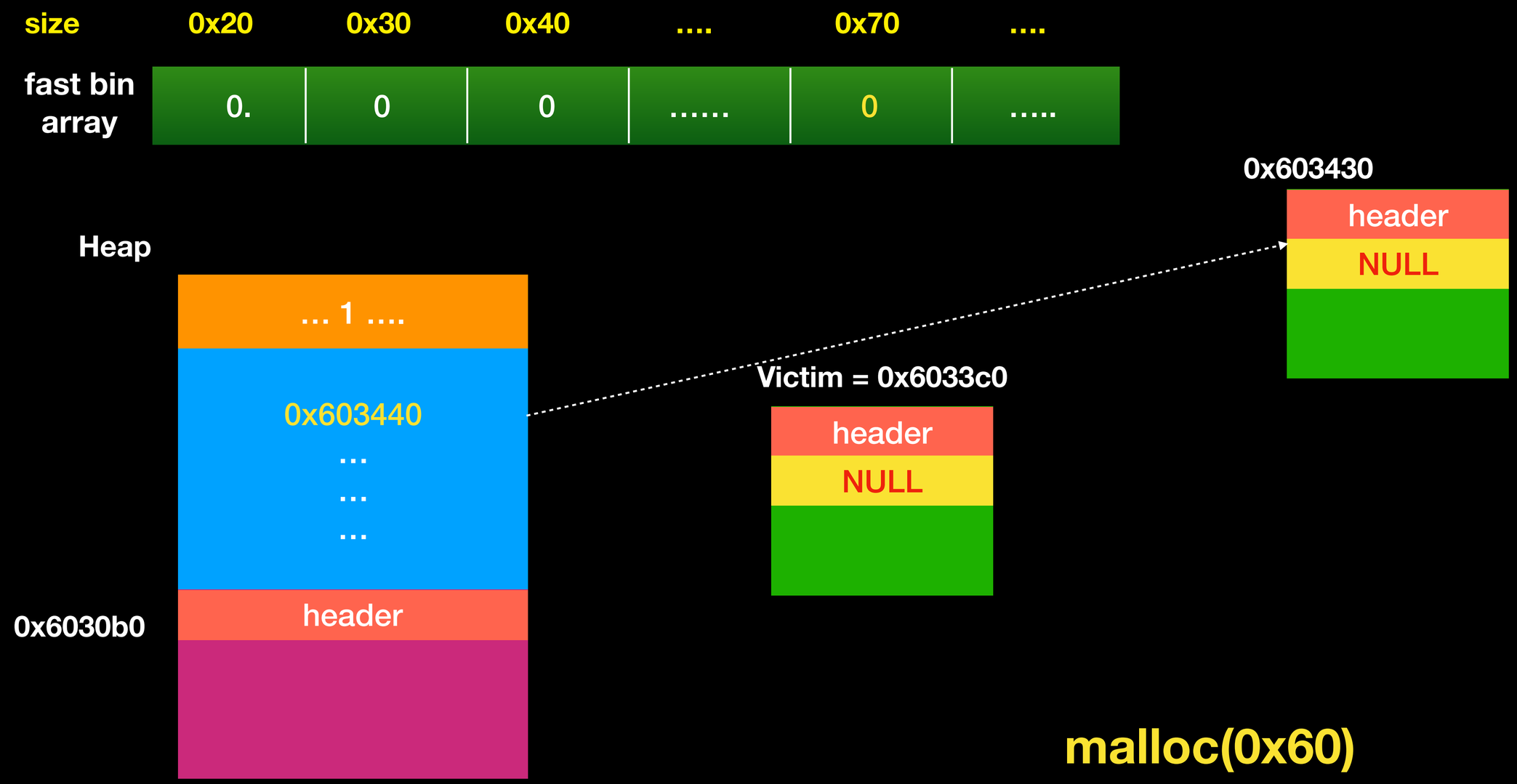
tcache



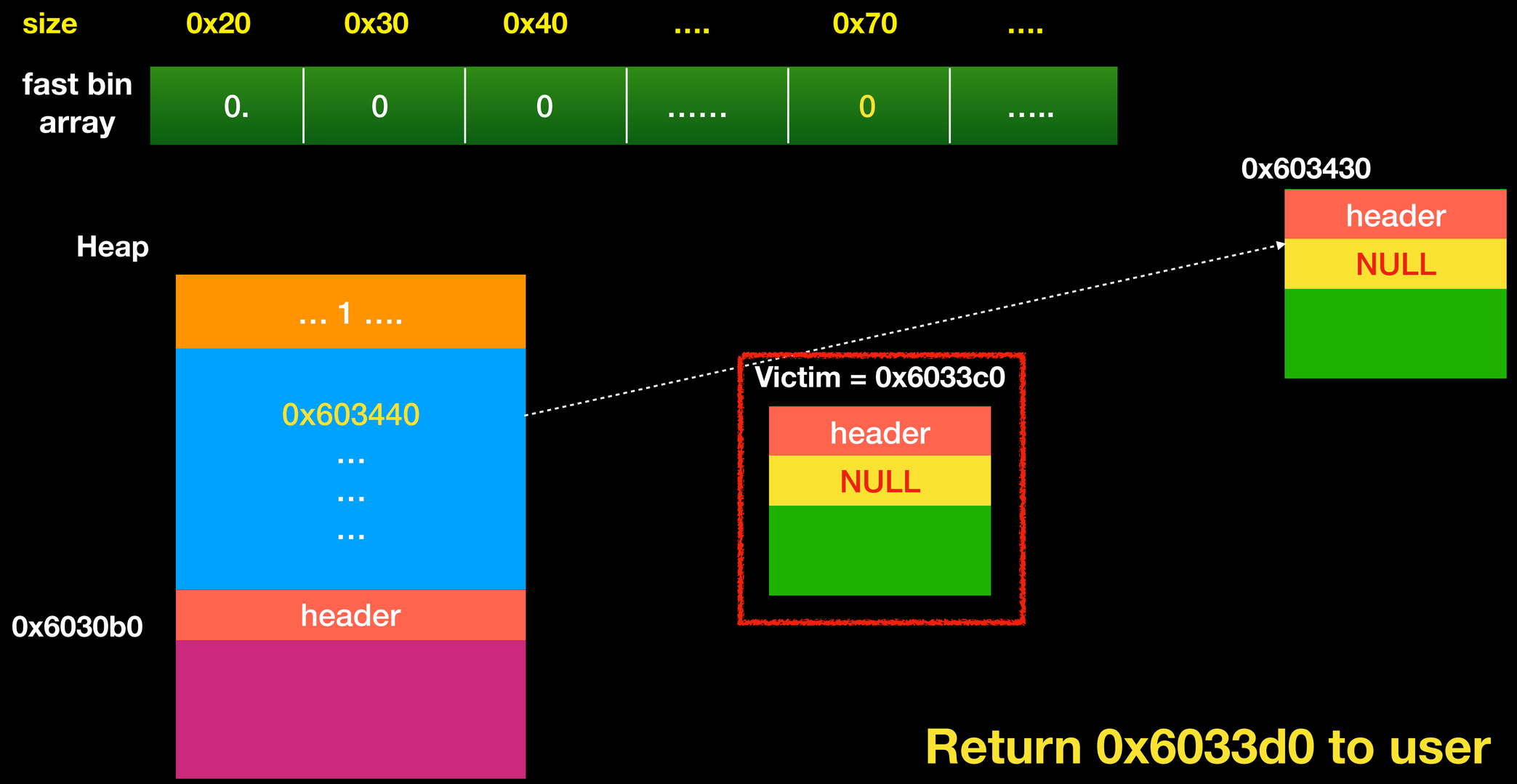
只要還沒滿，且 bin 中還有，就會繼續填充

malloc(0x60)

tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0
.....
0x603430
header
NULL
Heap
... 1 ....
0x603440
...
...
...
Victim = 0x6033c0
header
NULL
0x6030b0
header
malloc(0x60)

tcache


size
0x20
0x30
0x40
....
0x70
....
fast bin array
0.
0
0
.....
0
.....
0x603430
header
NULL
Heap
... 1 ....
0x603440
...
...
...
Victim = 0x6033c0
header
NULL
0x6030b0
header
Return 0x6033d0 to user

Outline

- New Structure
- Tcache
- Make Heap Exploitation Easy Again
  - Weakness in tcache

Weakness in tcache

- Malloc

```
static void *
tcache_get (size_t tc_idx)
{
  tcache_entry *e = tcache->entries[tc_idx];
  assert (tc_idx < TCACHE_MAX_BINS);
  assert (tcache->entries[tc_idx] > 0);
  tcache->entries[tc_idx] = e->next;
  --(tcache->counts[tc_idx]);
  return (void *) e;
}
```

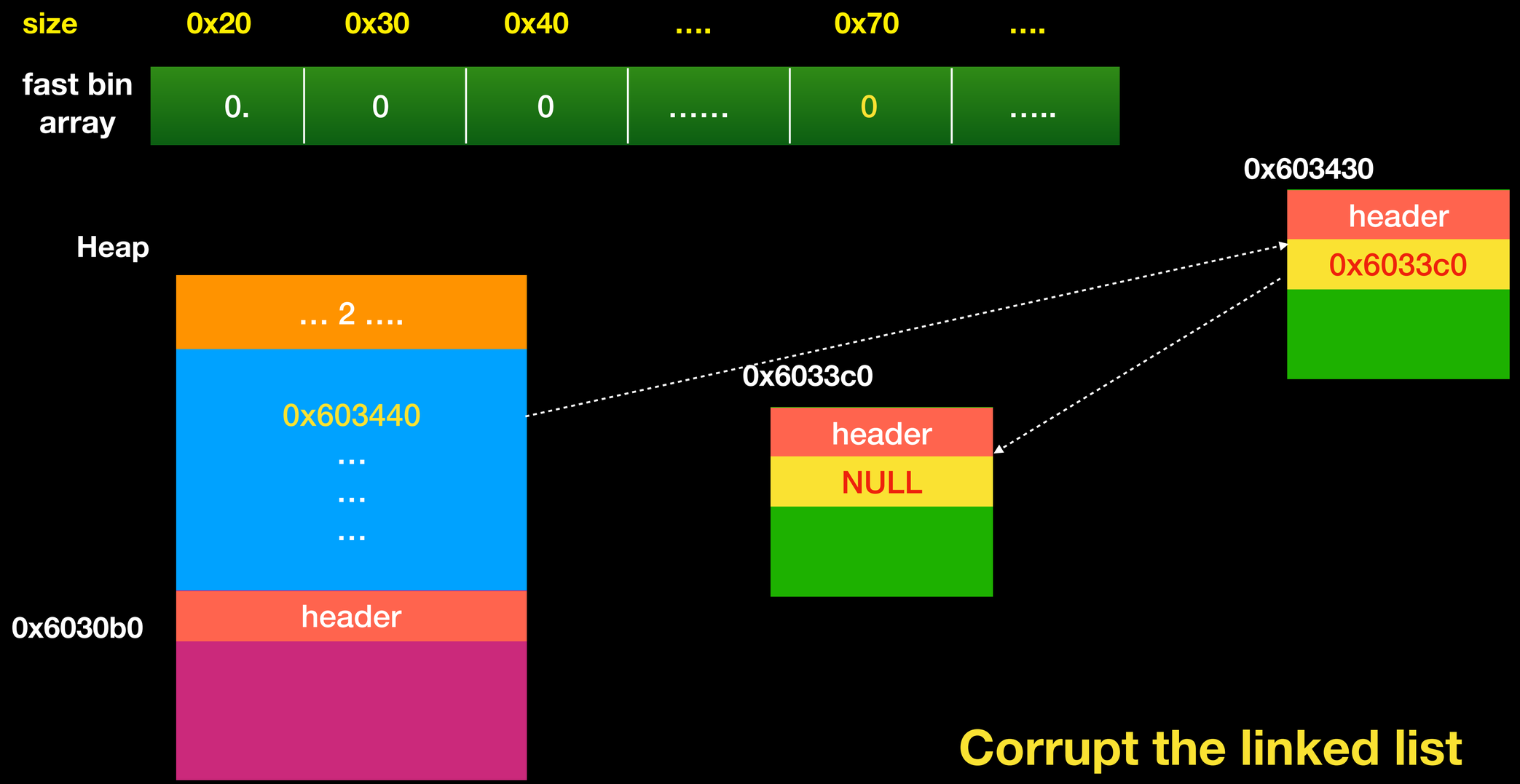
Weakness in tcache

- Malloc
  - 沒有任何檢查

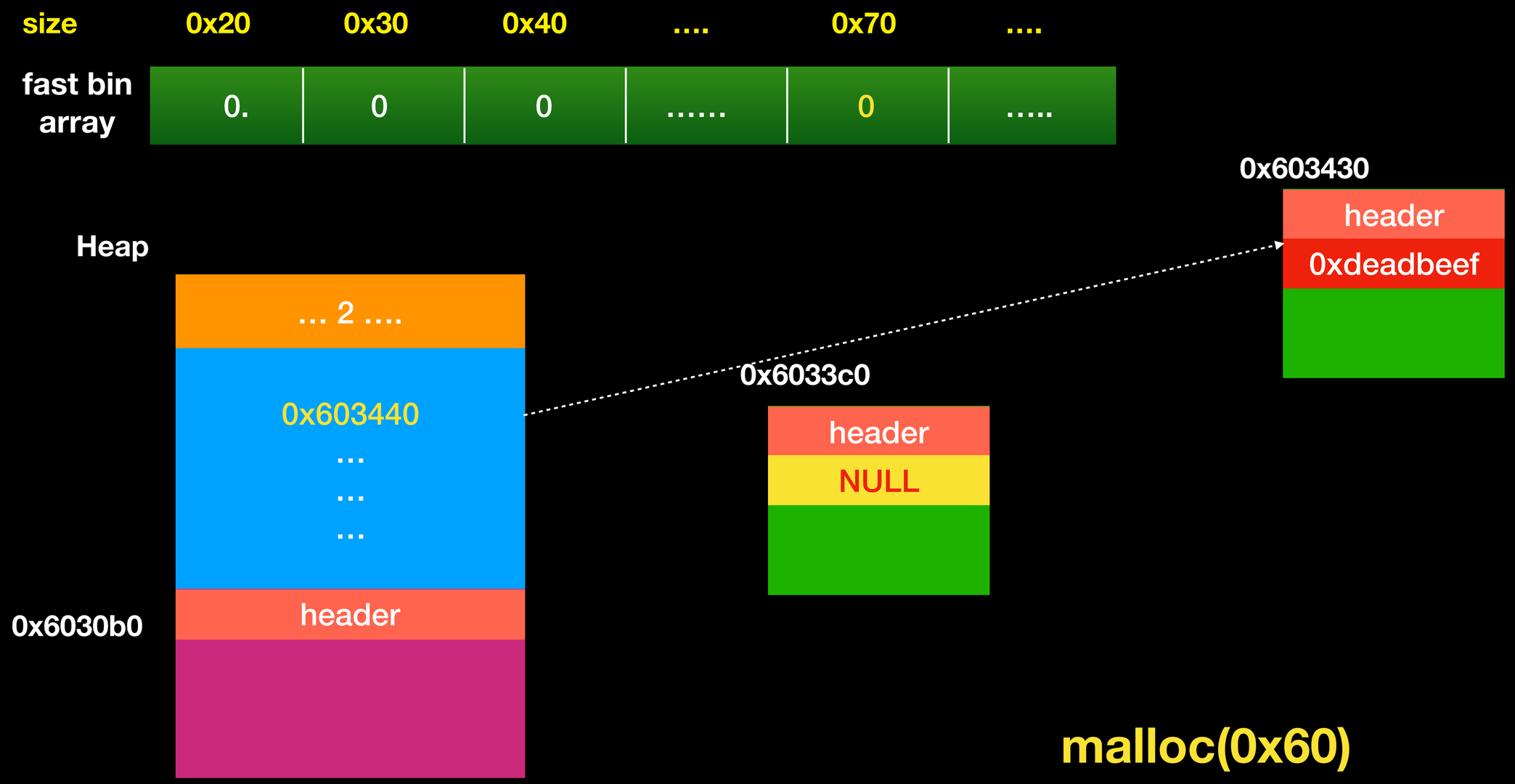
Weakness in tcache

- tcache corruption
- 利用任意 memory corruption 覆蓋 tcache 中的 next
    - 不需偽造任何 chunk 結構就可以拿到任意記憶體位置

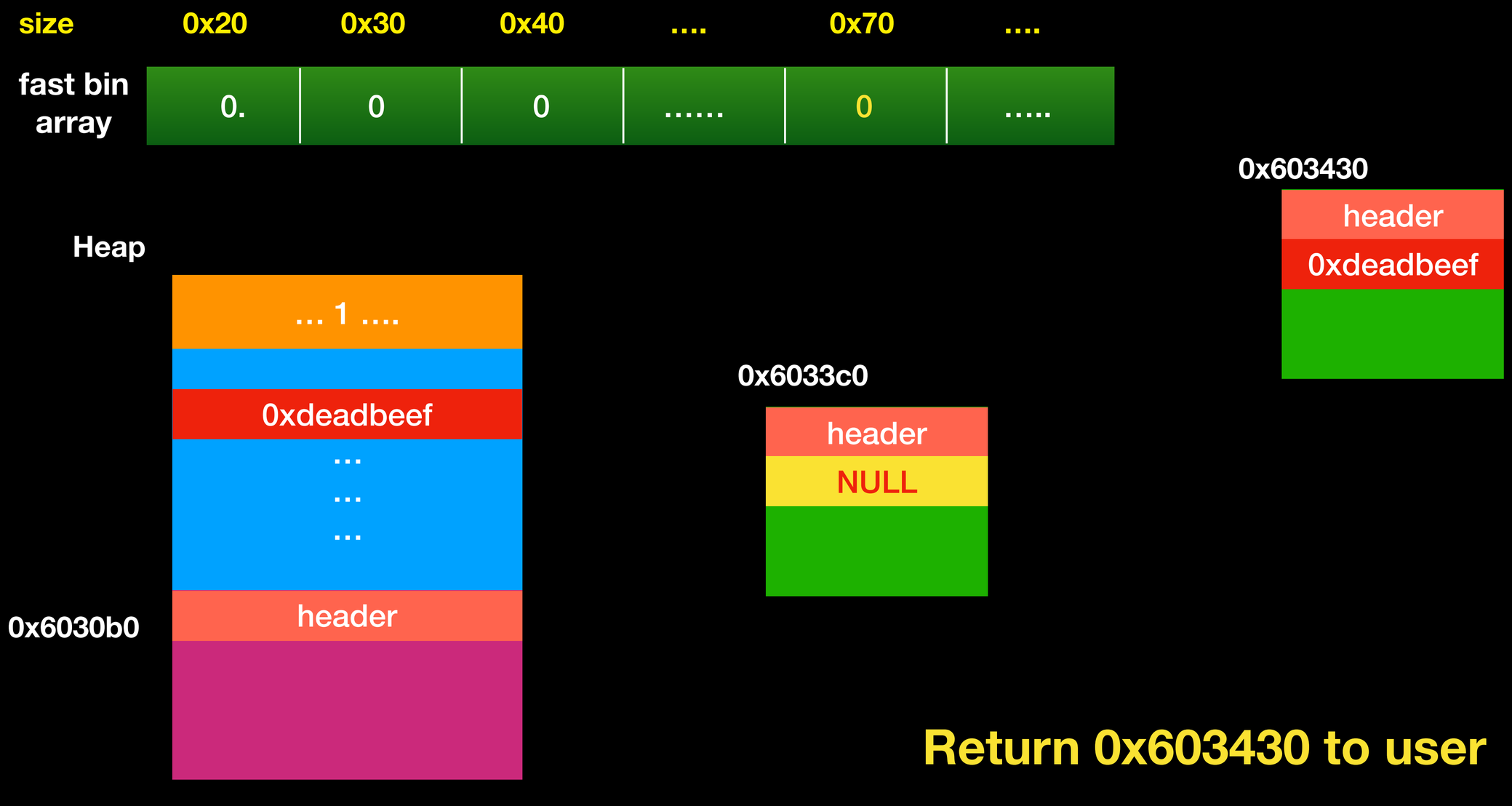
tcache corruption



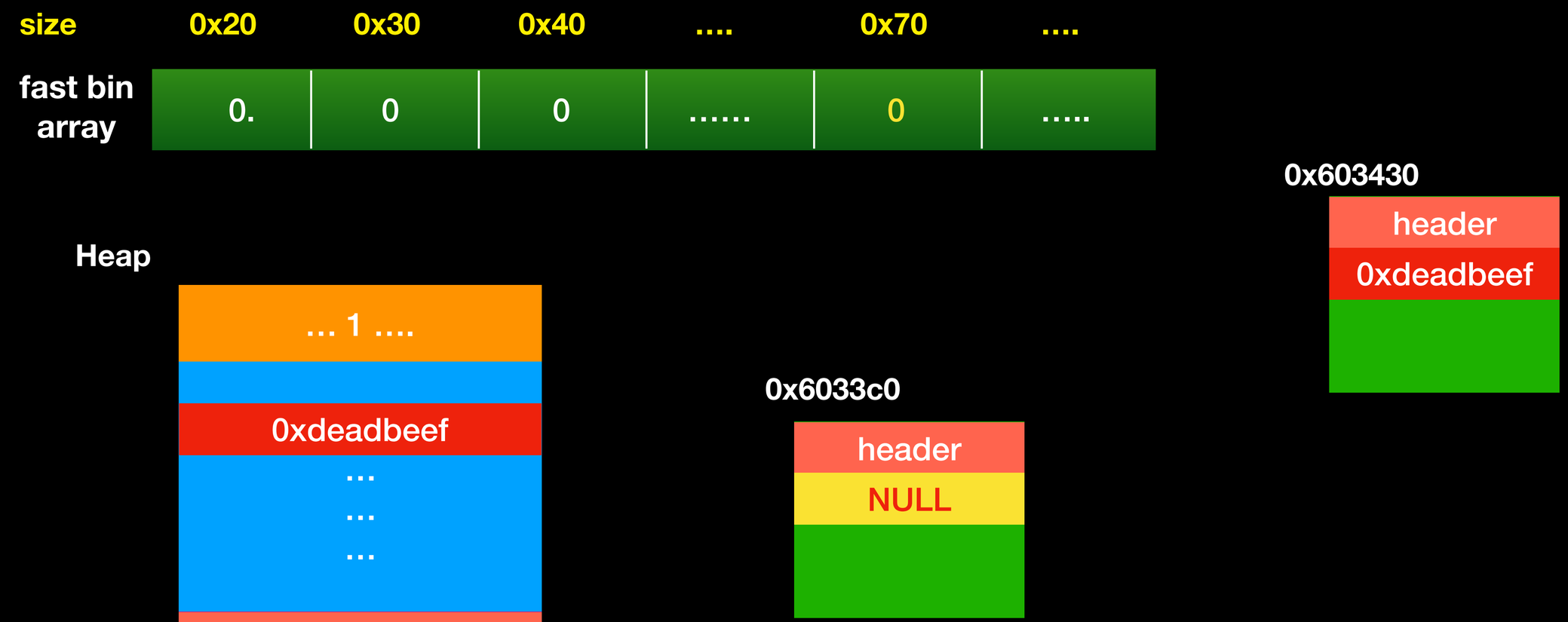
tcache corruption



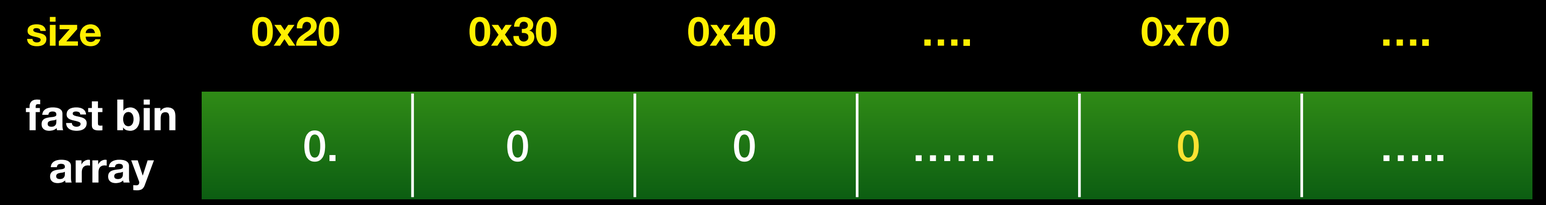
tcache corruption



tcache corruption

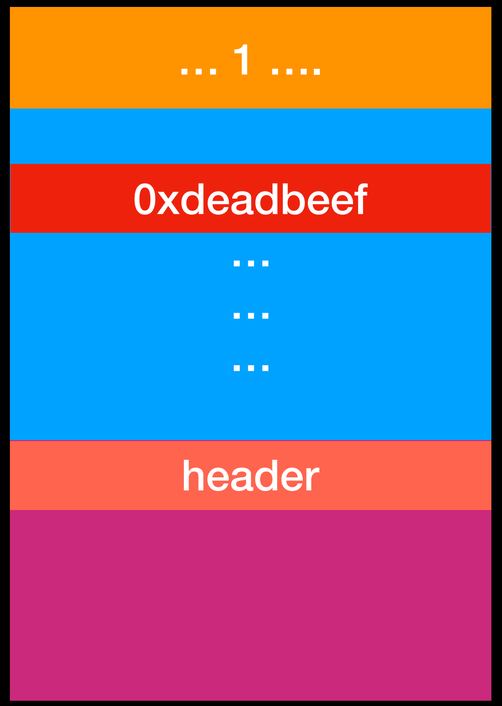


tcache corruption

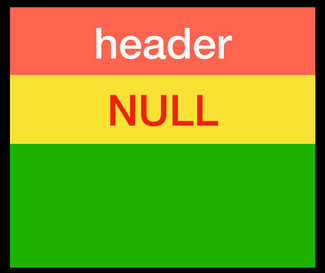


Heap

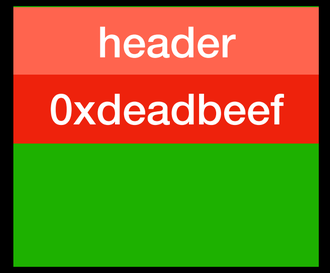
0x6030b0



0x6033c0



0x603430



Return 0xdeadbeef to user

Weakness in tcache

- Free
  - 在存入 tcache 之前會檢查
    - size 的合法性
      - size > MINSIZE && -size < p
    - 位置是否有對齊
      - 必須對其 8 的倍數

Weakness in tcache

- Free
  - Double free ?
  - Linked list 完整性 ?

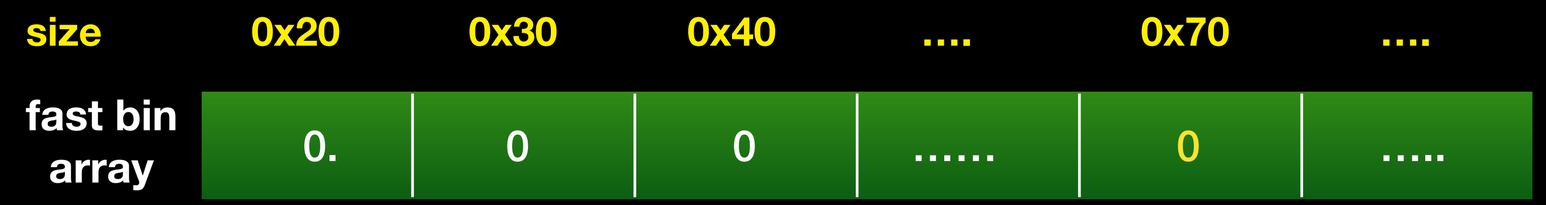
Weakness in tcache

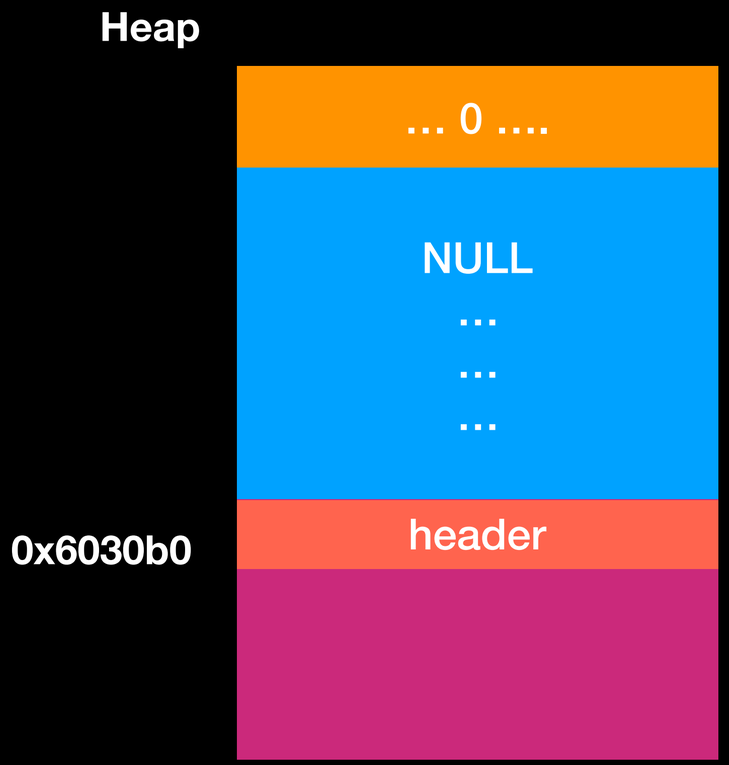
- Free
  - ~~Double free ?~~
  - ~~Linked list 完整性 ?~~
  - 這些通通都沒有檢查

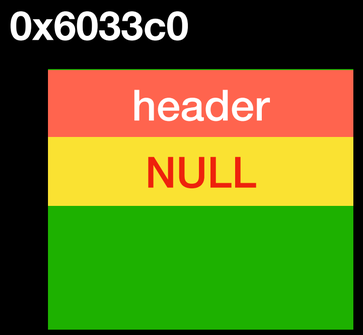
Overlap tcache

- 因為沒有檢查 double free
  - 所以我們可以不斷對同一記憶體區塊 free
  - 我們可以利用此特性來達成 fastbin attack 的效果，甚至更好用，因為不須符合 chunk 的結構
  - 另外他也沒有檢查 tcache 中的 count

Overlap tcache

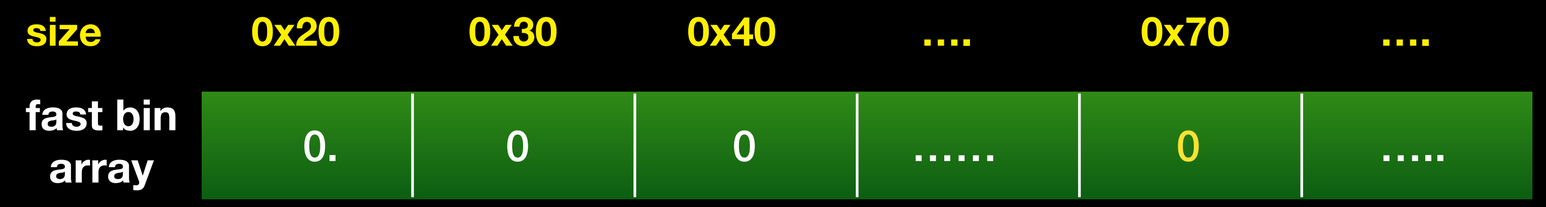


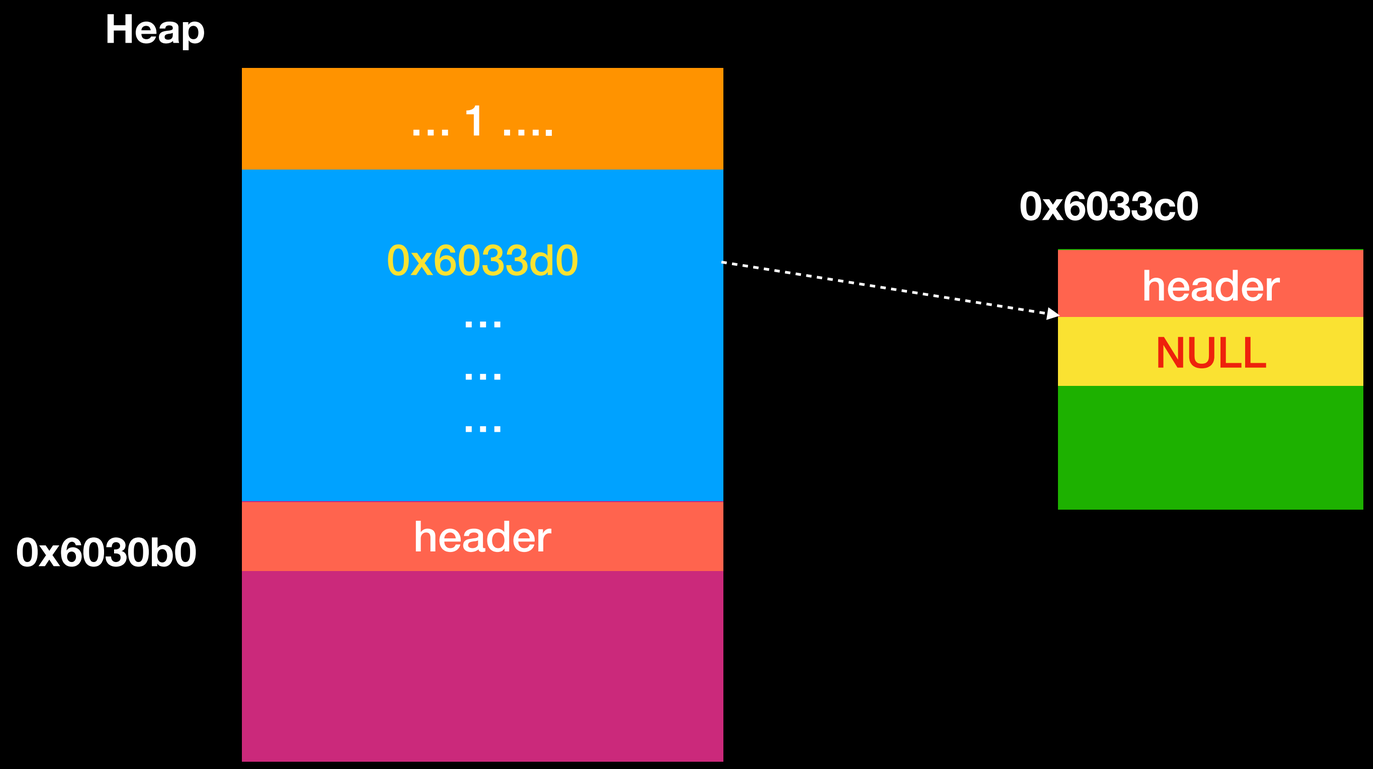




free(0x6033d0)

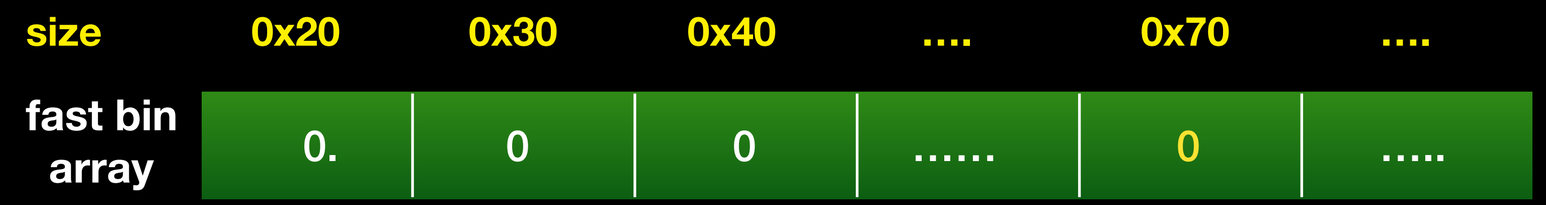
Overlap tcache

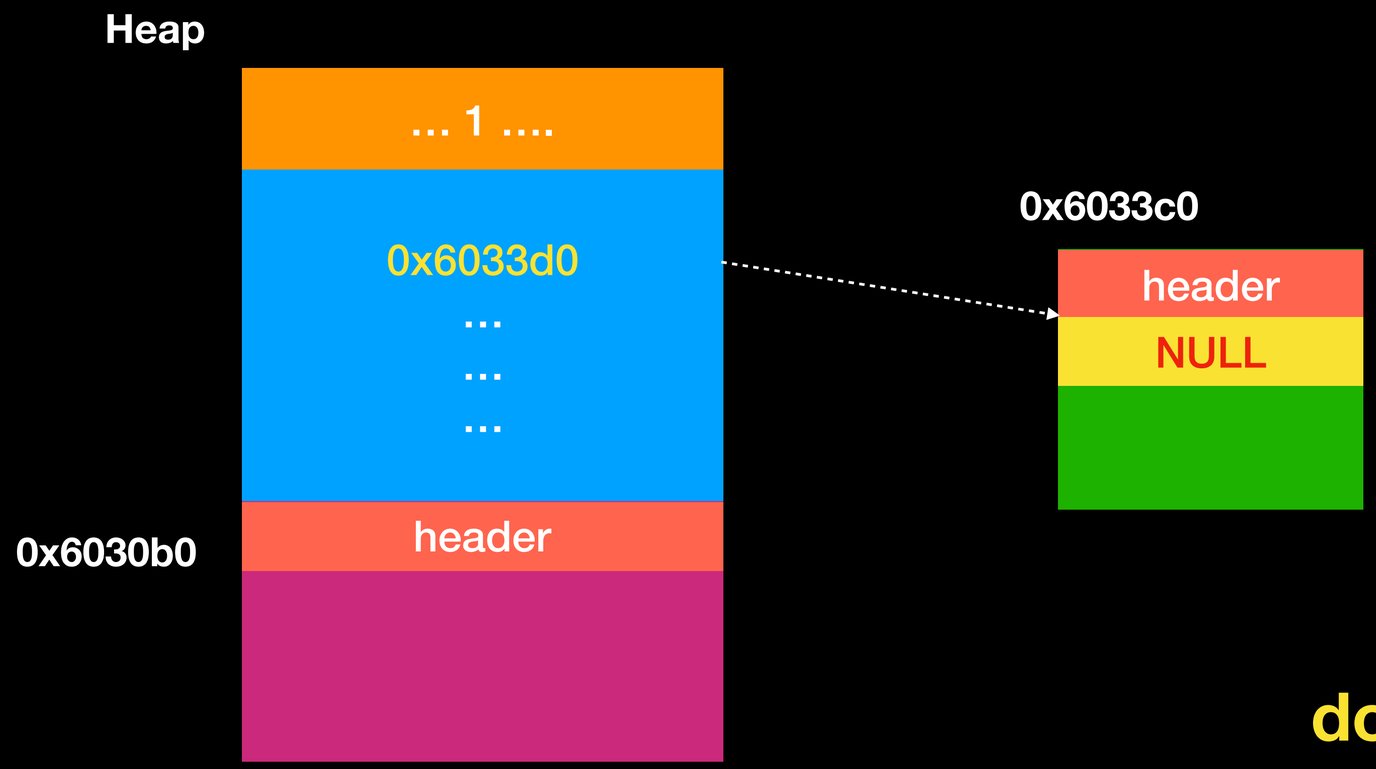




free(0x6033d0)

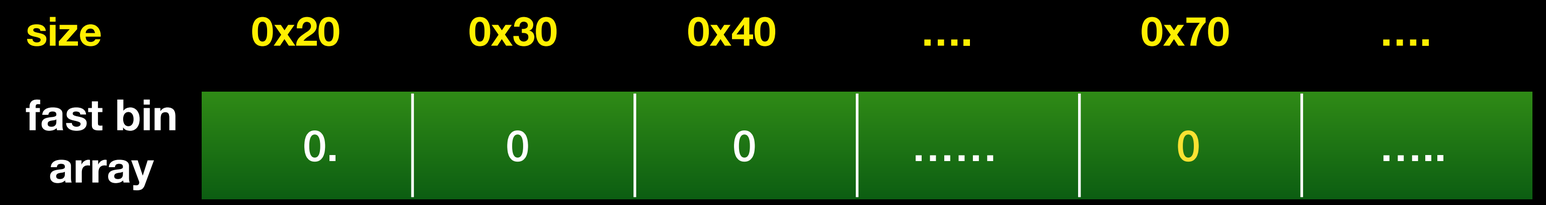
Overlap tcache

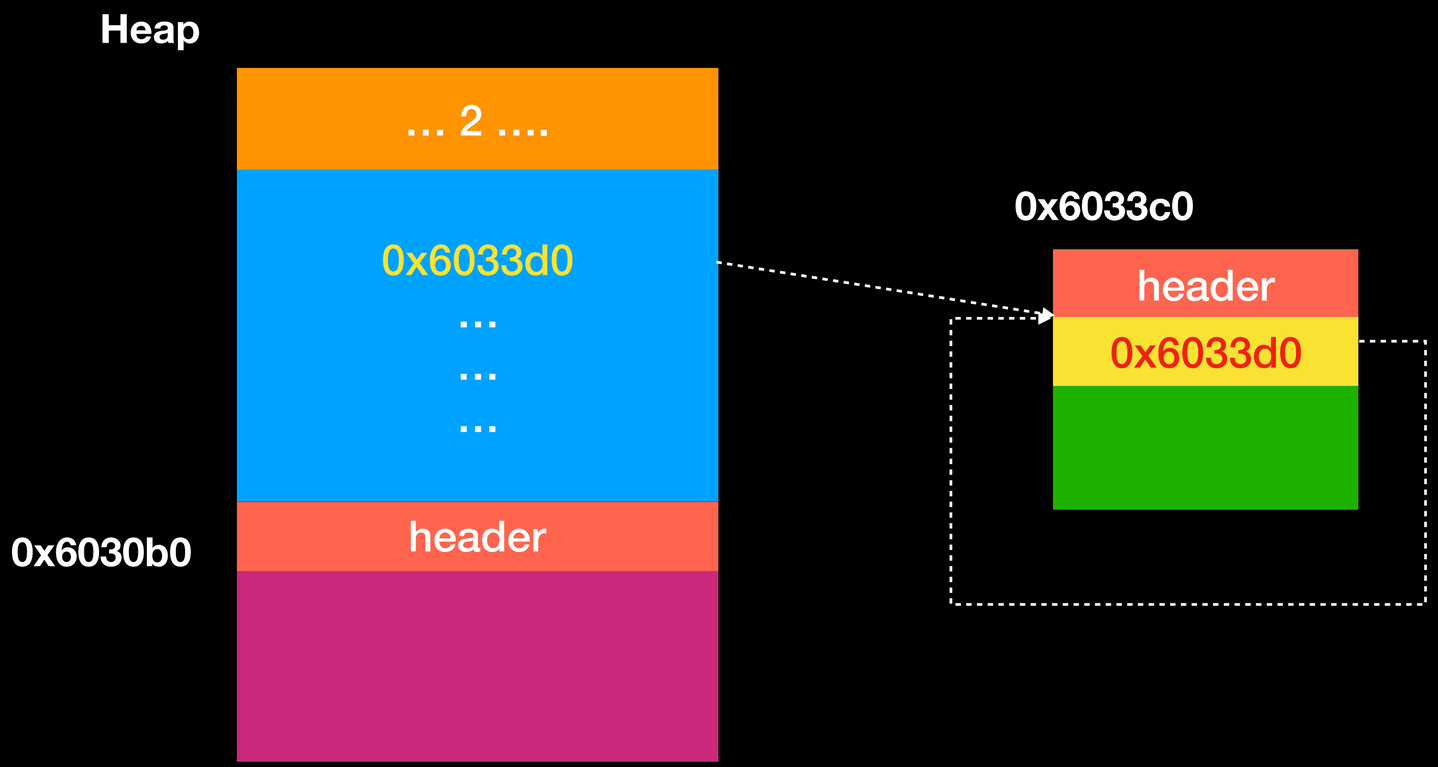




double free(0x6033d0)

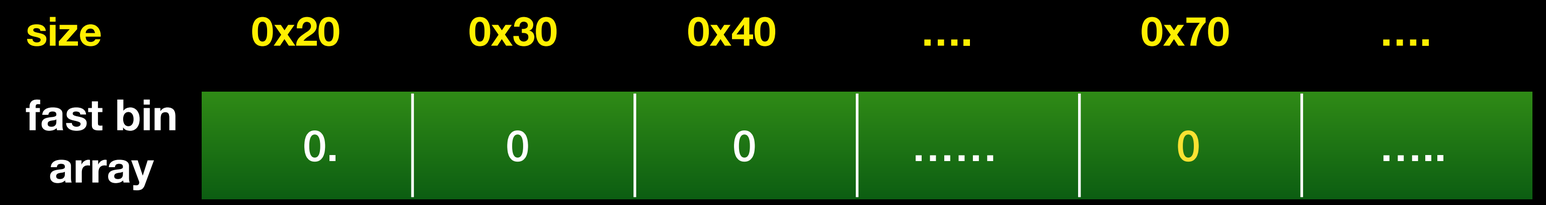
Overlap tcache

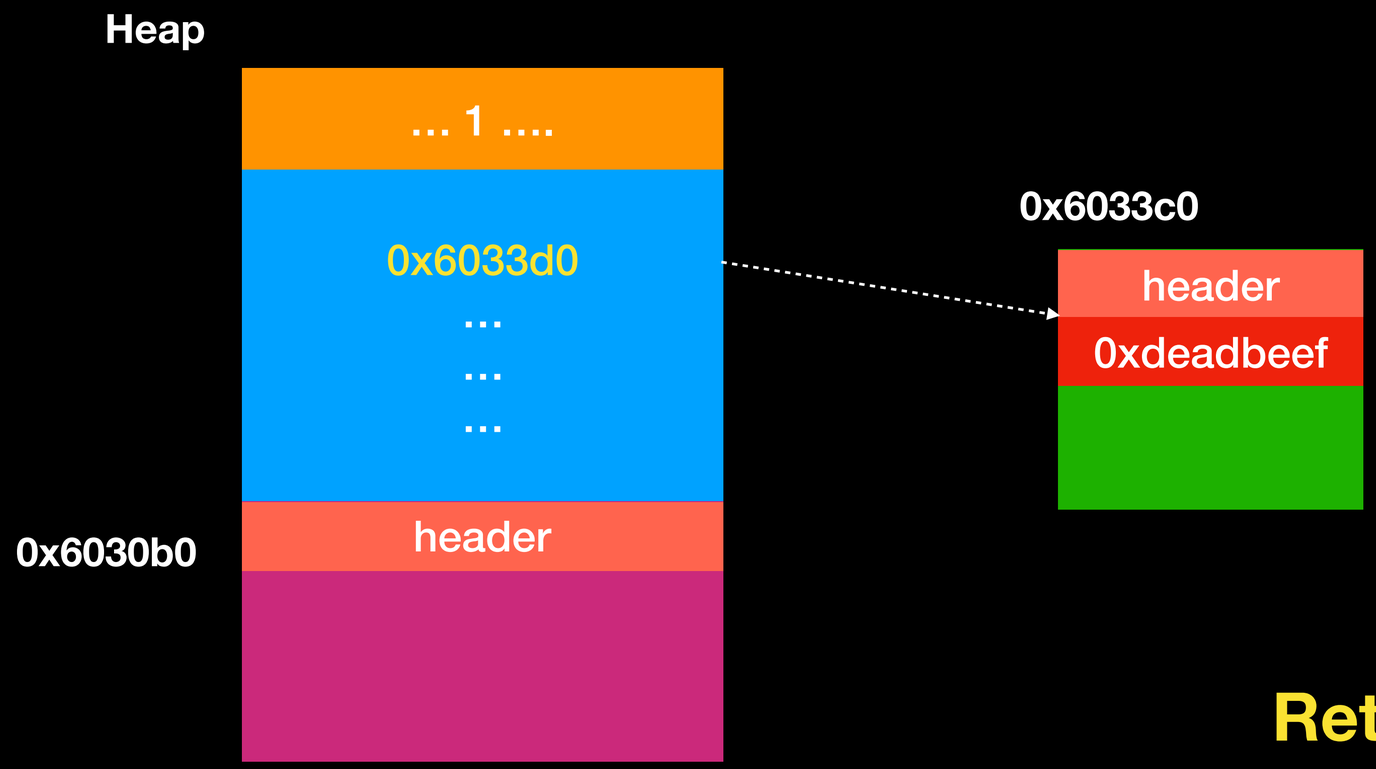




malloc(0x60)

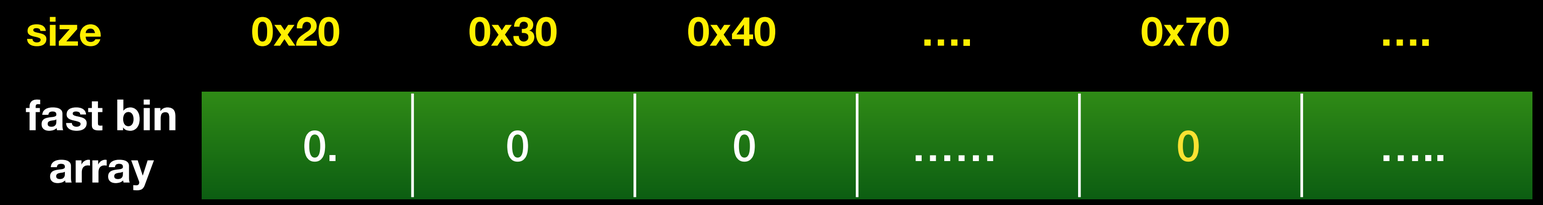
Overlap tcache

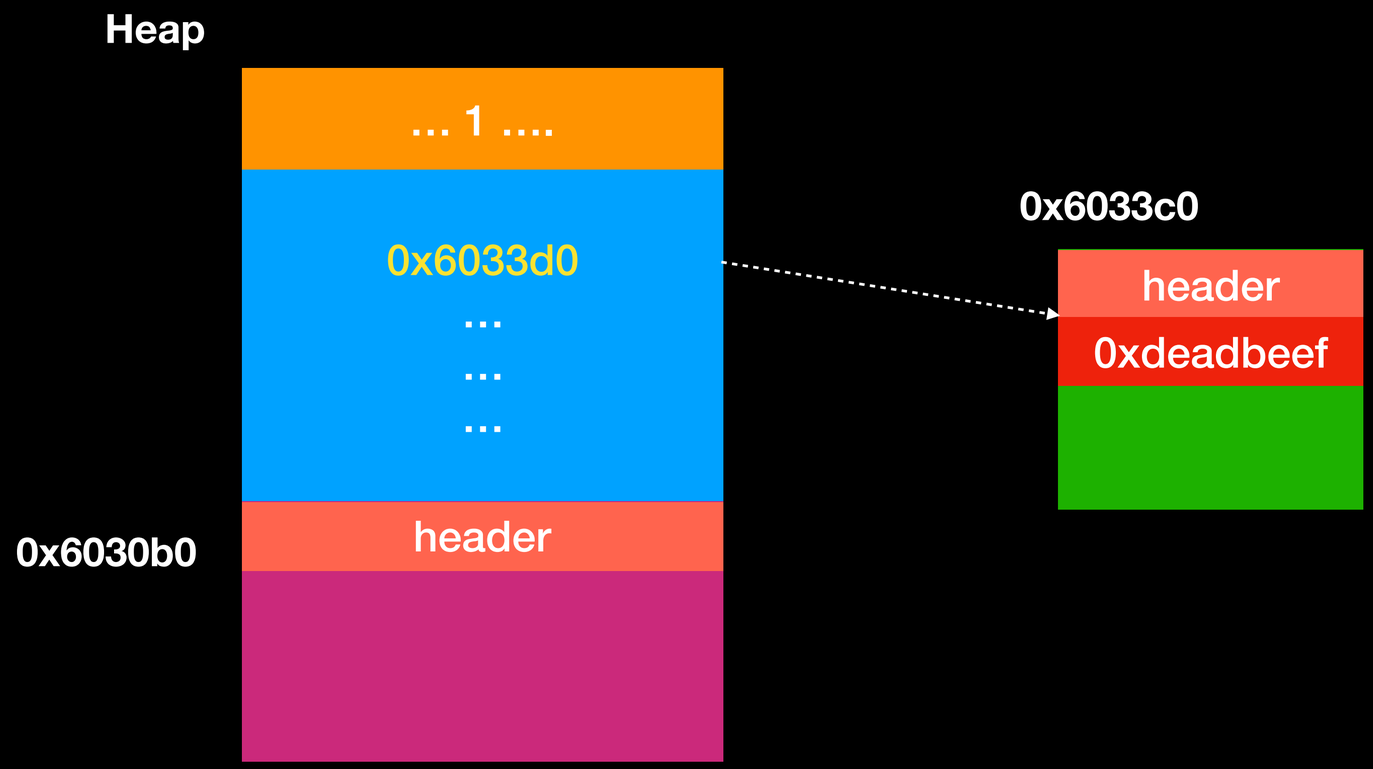




Return 0x6033d0 to user

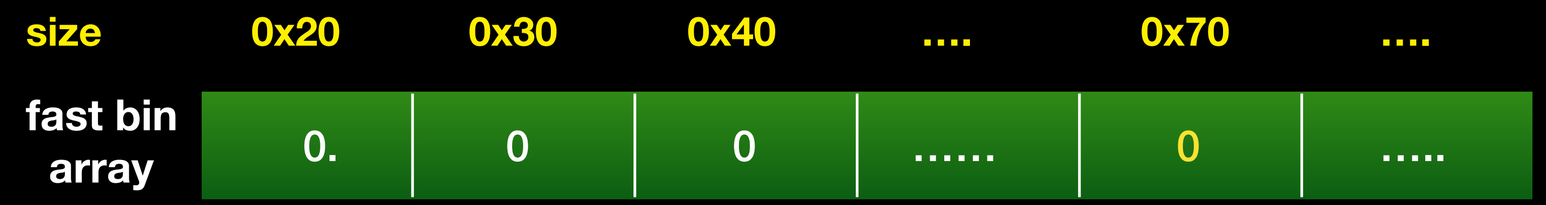
Overlap tcache

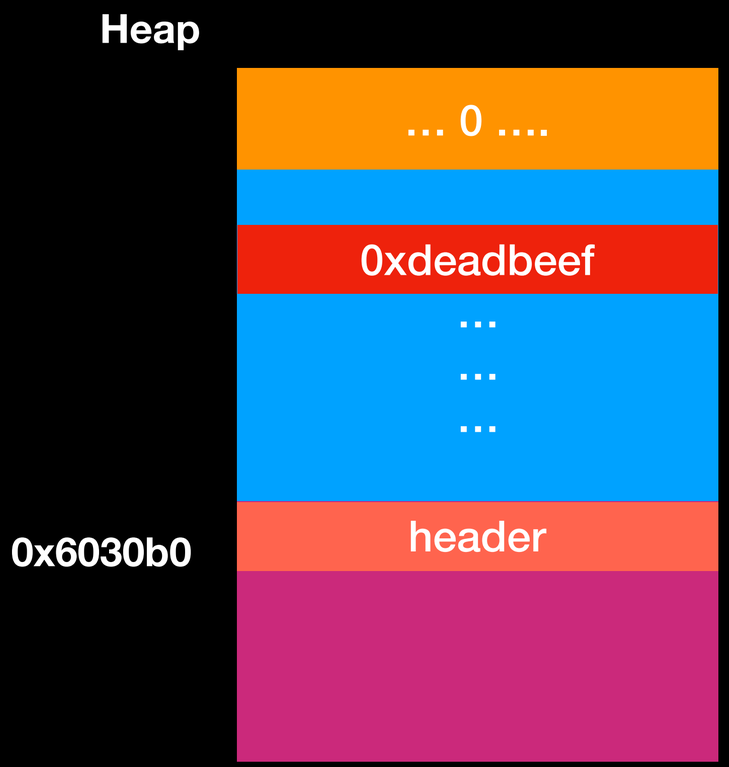


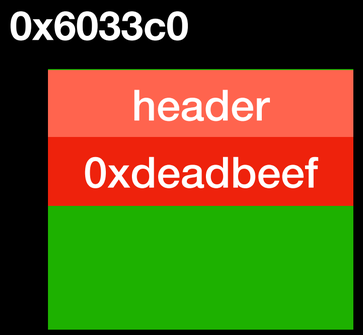


malloc(0x60)

Overlap tcache

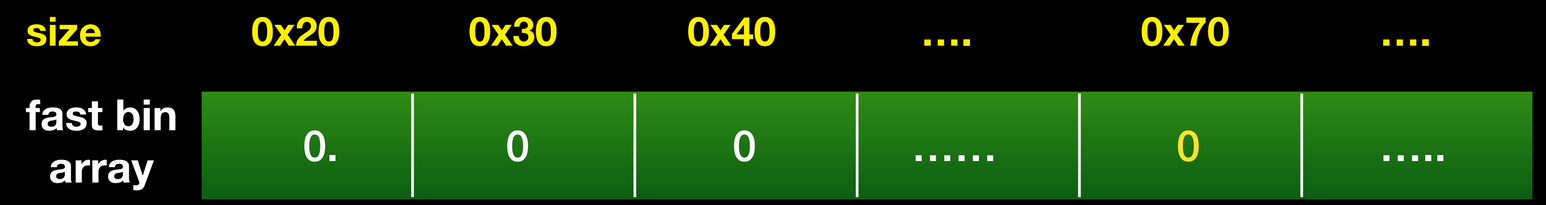


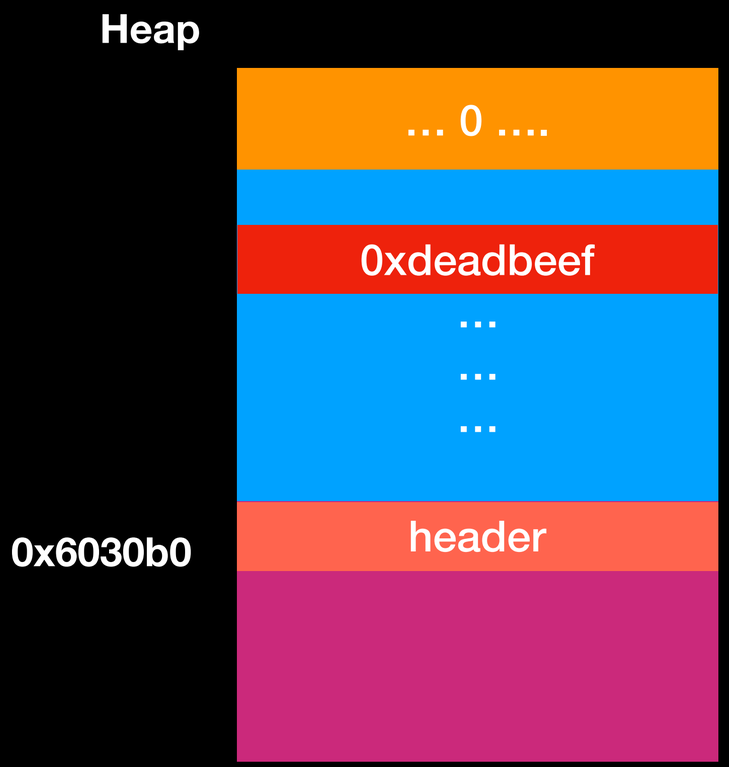


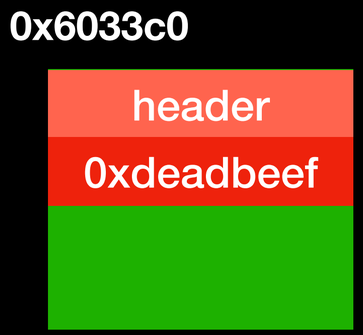


Return 0x6033d0 to user

Overlap tcache

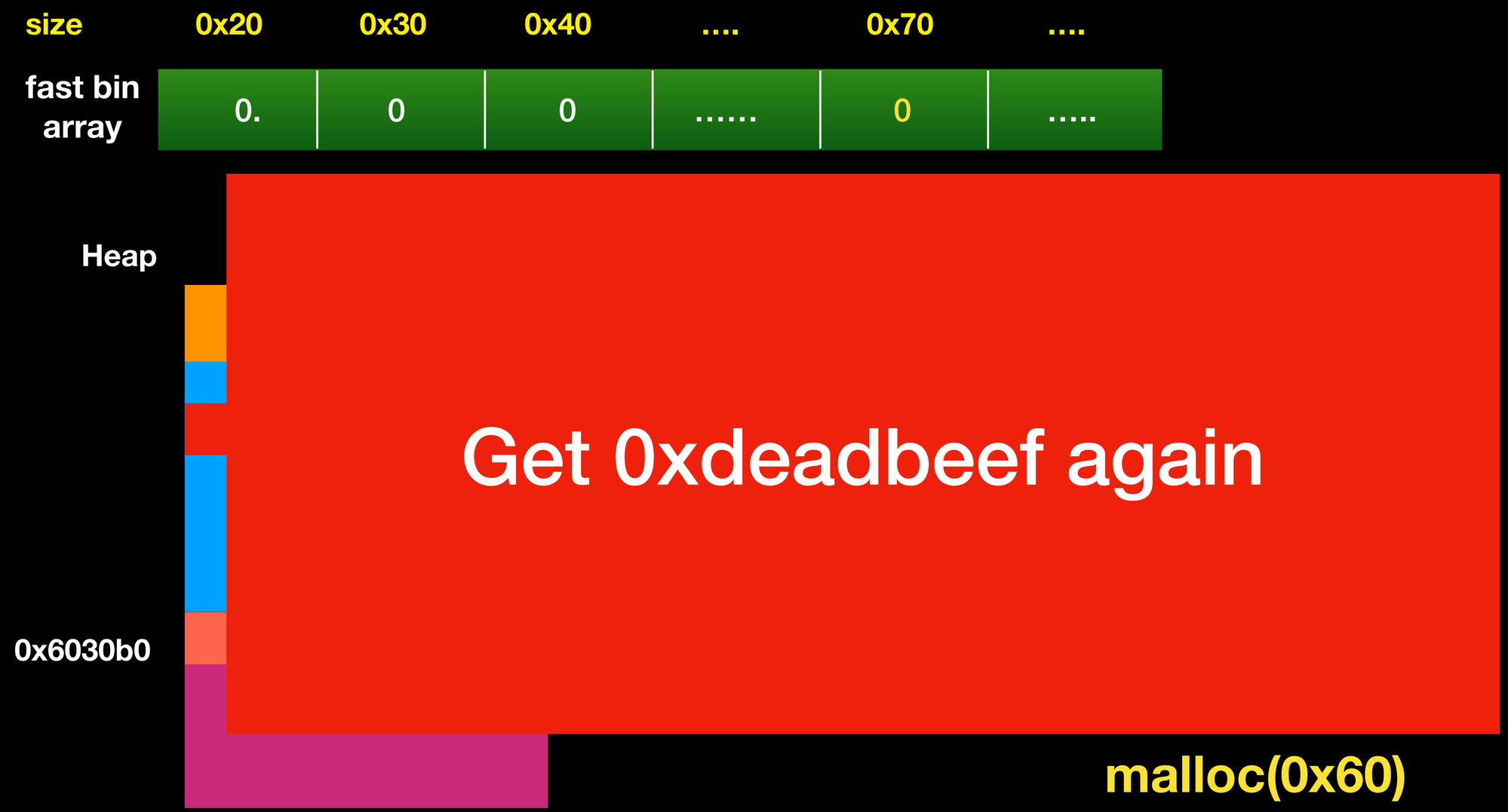






malloc(0x60)

Overlap tcache



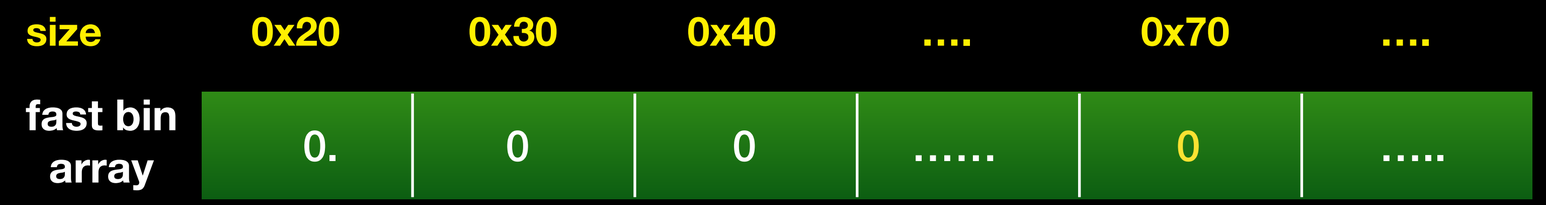
Small bin attack

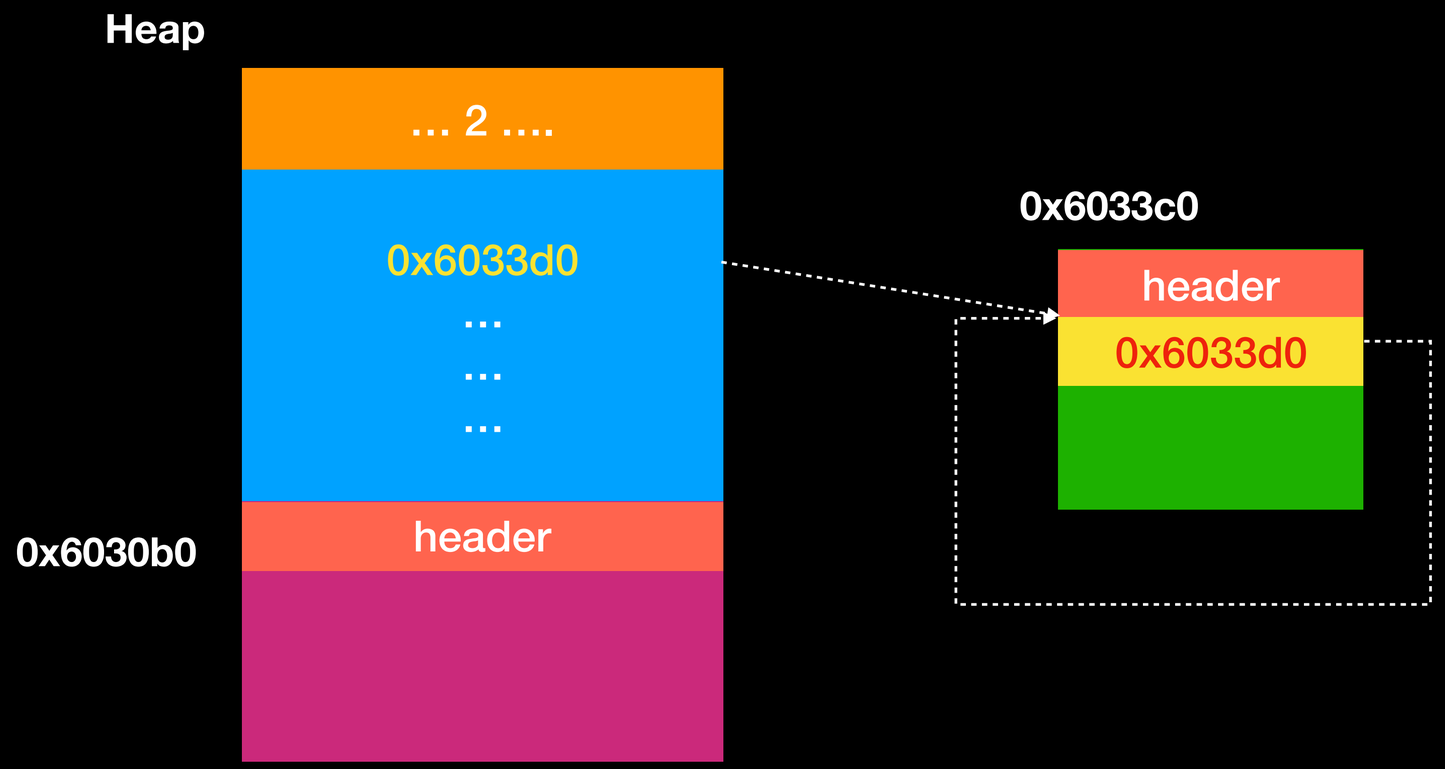
- 在從 small bin 填充至 tcache 時
  - 需要對 small bin 做 unlink
    - 正常來講，應該要對該 chunk 做 double linked list 的檢查
    - 但是這邊卻沒檢查，所以可以做類似 unsorted bin attack 的效果
      - 唯一要注意的是 count 數要剛好滿，不然會導致無窮迴圈或者存取到非法區段而 segment fault

tcache perthread corruption

- 該結構掌管整個 tcache 機制
- 配合前面漏洞複寫該結構
  - 就可控制整個 tcache ，不管 malloc 大小多少，變成整個都是可控的
  - 很多情況下只要利用前面漏洞做 partial overwrite 就可以間接控制該結構

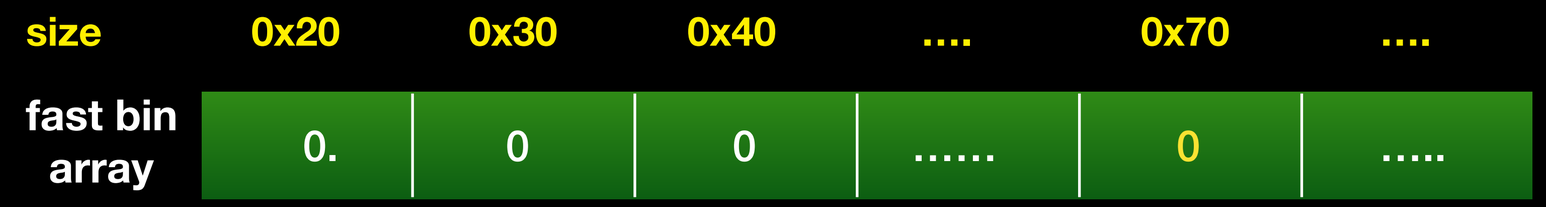
tcache perthread corruption

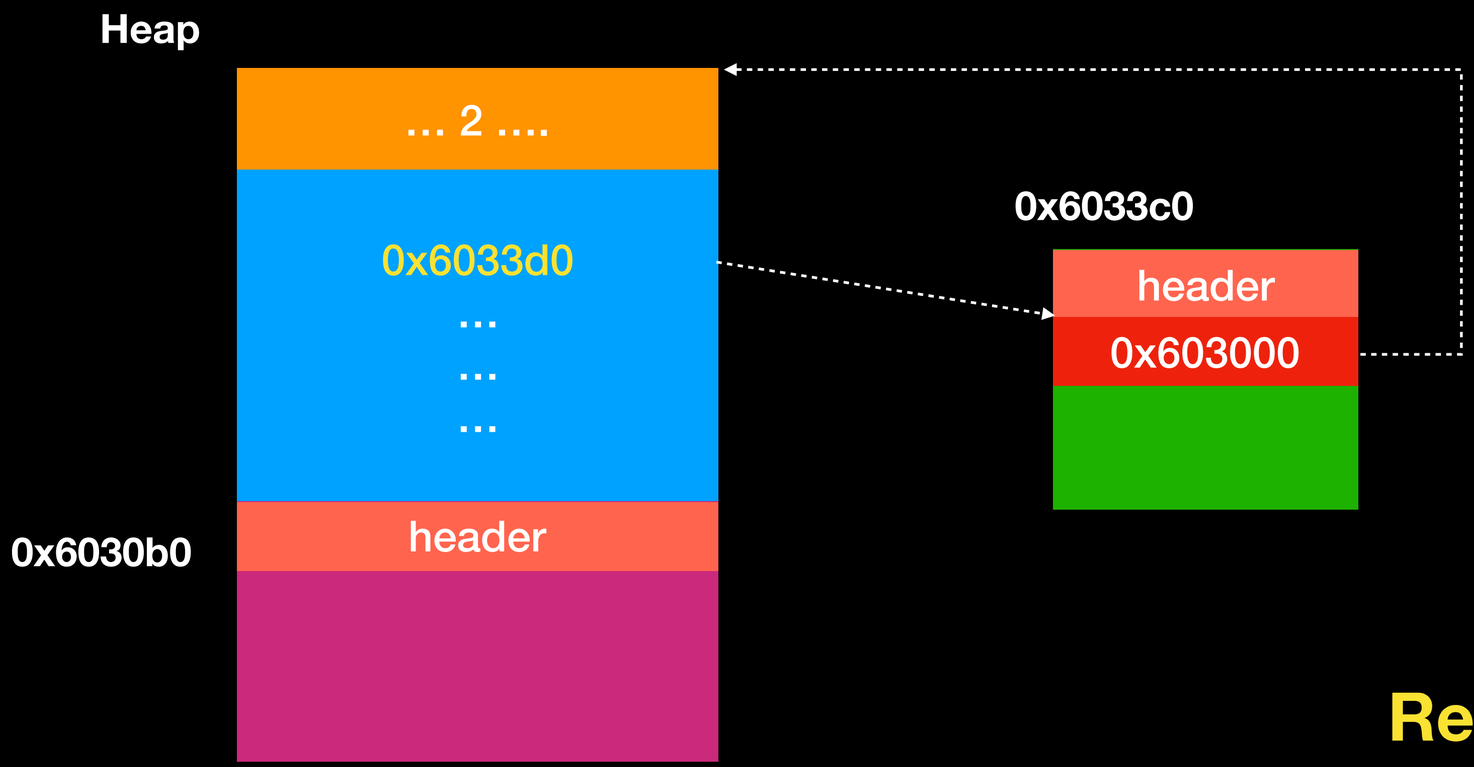




malloc(0x60)

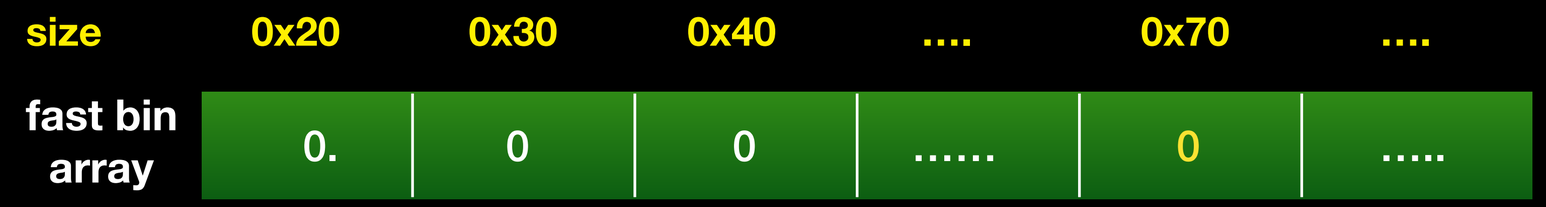
tcache perthread corruption

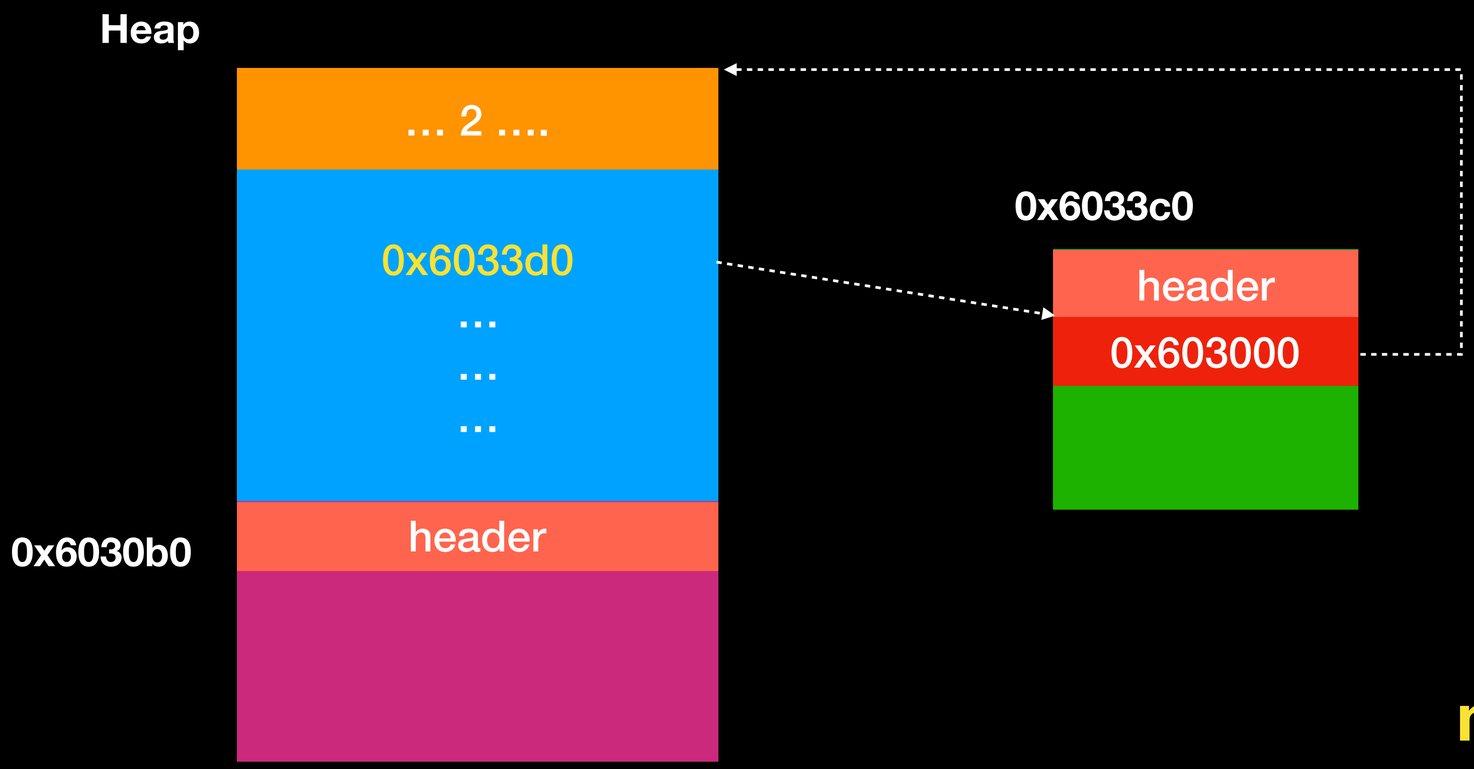




Return 0x6033d0

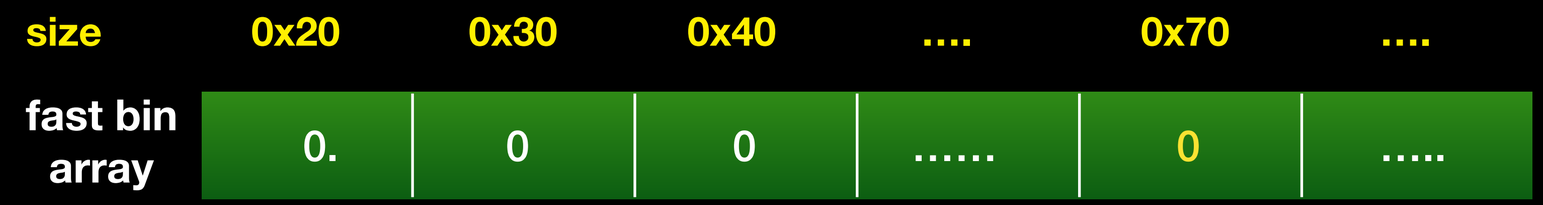
tcache perthread corruption

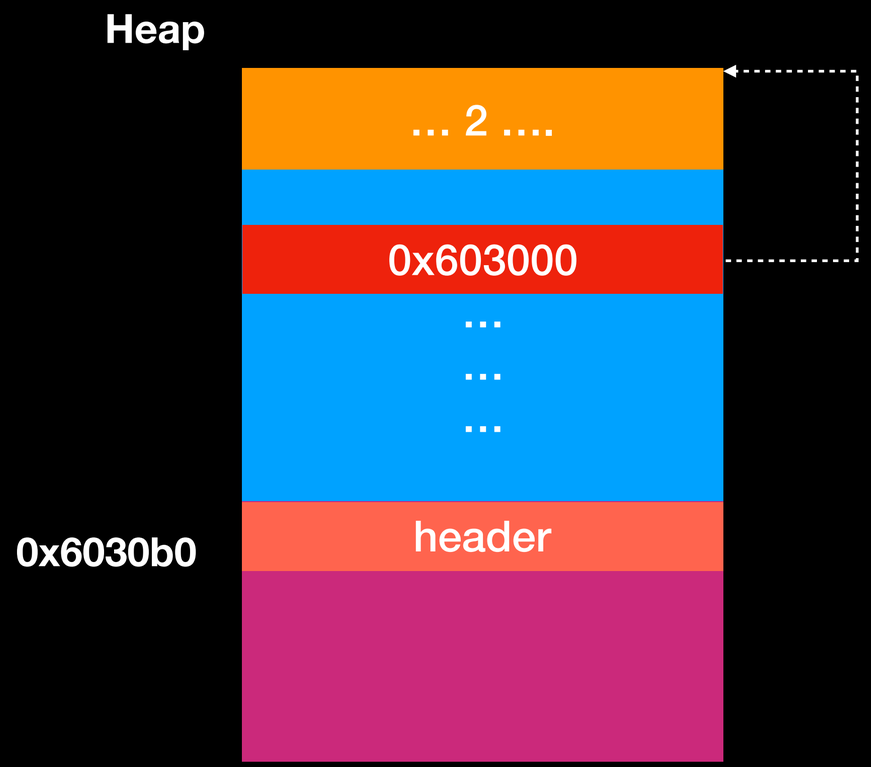


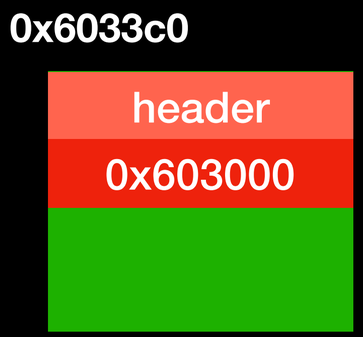


malloc(0x60)

tcache perthread corruption

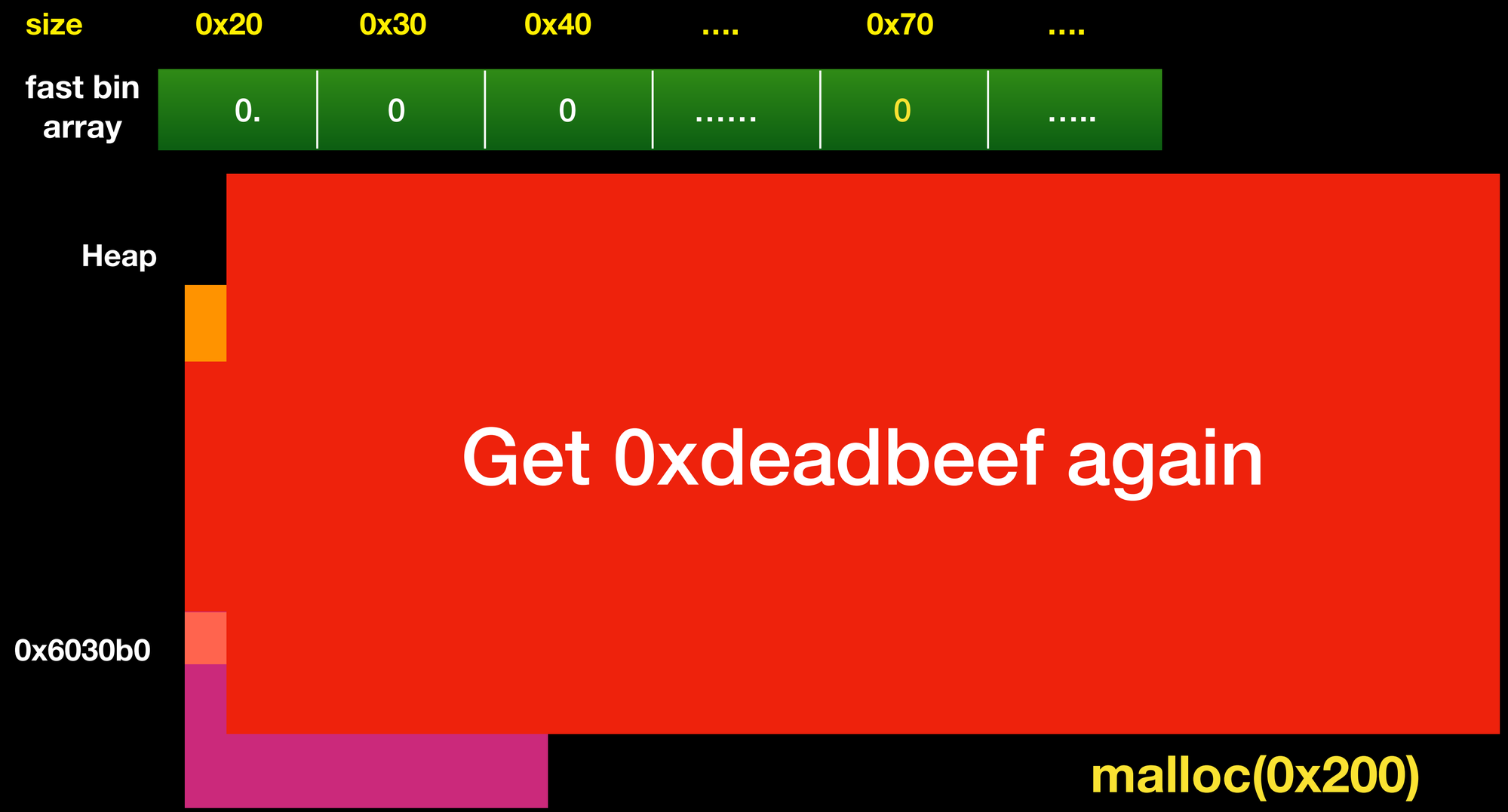






malloc(0x60)

tcache perthread corruption



Reference

- http://tukan.farm/2017/07/08/tcache/

Q & A