Heap Exploitation

Binary Exploitation 2019/2020

Heap Overview

- The Heap is a pool of memory used for dynamic allocation at runtime.
  - malloc – grabs memory from the heap
  - free – releases memory on the heap

- It is a data segment, just like the .stack or the .bss

Heap Overview

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x555555554000     0x555555555000 r-xp     1000 0      /home/vagrant/share/heap/hi_mom
    0x555555754000     0x555555755000 r--p     1000 0      /home/vagrant/share/heap/hi_mom
    0x555555755000     0x555555756000 rw-p     1000 1000   /home/vagrant/share/heap/hi_mom
    0x555555756000     0x555555777000 rw-p    21000 0      [heap]
    0x7ffff79e4000     0x7ffff7bcb000 r-xp   1e7000 0      /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7bcb000     0x7ffff7dcb000 ---p   200000 1e7000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dcb000     0x7ffff7dcf000 r--p     4000 1e7000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dcf000     0x7ffff7dd1000 rw-p     2000 1eb000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dd1000     0x7ffff7dd5000 rw-p     4000 0
    0x7ffff7dd5000     0x7ffff7dfc000 r-xp    27000 0      /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7dd5000     0x7ffff7dfc000 rwxp    27000 0      <explored>
    0x7ffff7fe9000     0x7ffff7feb000 rw-p     2000 0
    0x7ffff7ff7000     0x7ffff7ffa000 r--p     3000 0      [vvar]
    0x7ffff7ffa000     0x7ffff7ffc000 r-xp     2000 0      [vdso]
    0x7ffff7ffc000     0x7ffff7ffd000 r--p     1000 27000  /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000 28000  /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000 0
    0x7ffffffde000     0x7ffffffff000 rw-p    21000 0      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp     1000 0      [vsyscall]
```

Heap Overview - Usage

```
int main() {
    char * buffer = NULL;

    /* get 0x100 bytes from memory from the heap */
    buffer = (char*) malloc(sizeof(char) * 0x100);

    fgets(stdin, buffer, 0x100);
    printf("%s", buffer);

    /* release the allocated memory */
    free(buffer);

    return 0;
}
```

Heap Chunks

- The heap is made of **Heap Chunks**, and there are different types:
  - Allocated chunk
  - Free chunk
  - Top chunk
  - Last Remainder chunk

Chunk Implementation

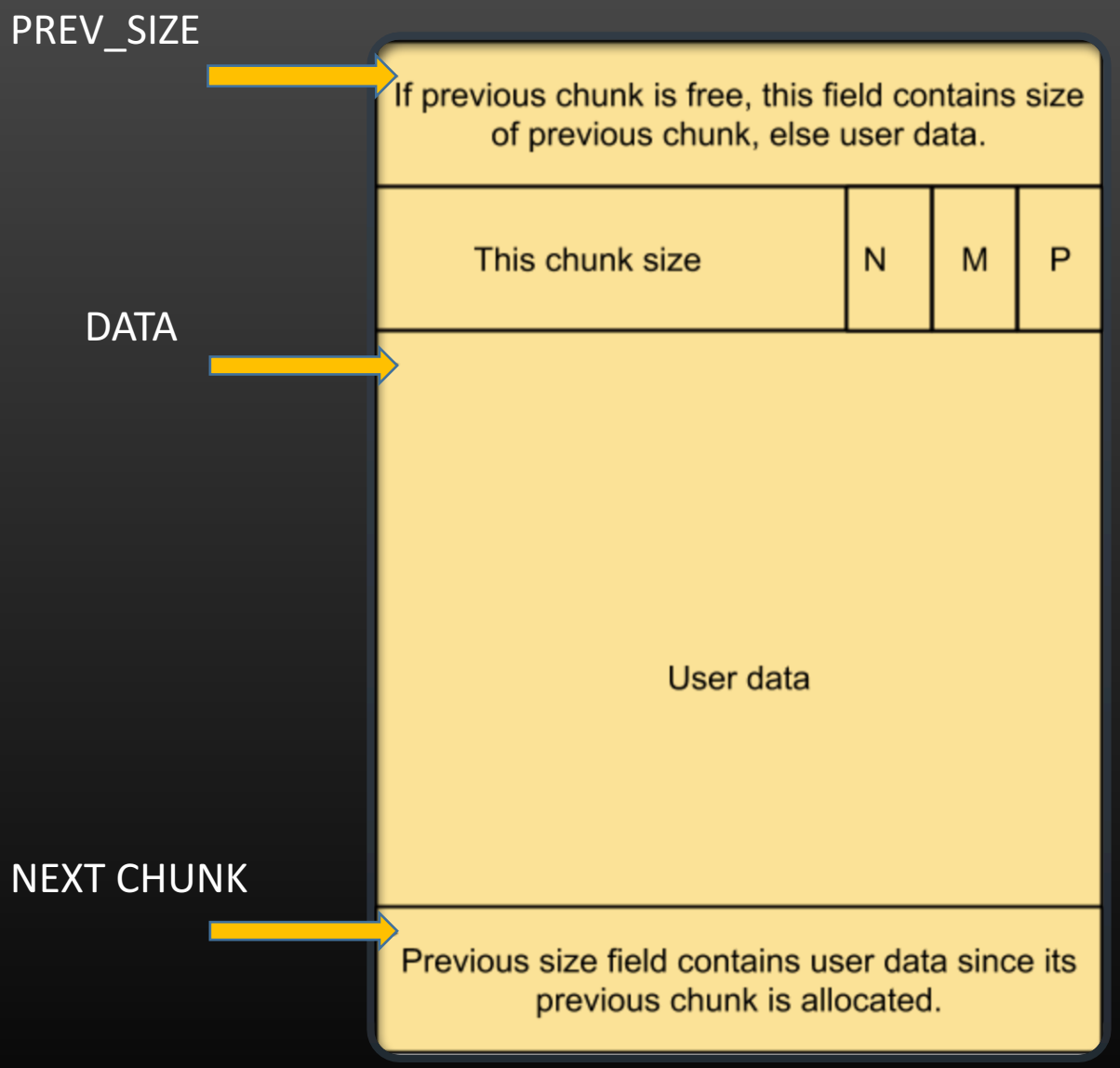
```
struct malloc_chunk {
    INTERNAL_SIZE_T     prev_size;    /* Size of previosu chunk (if free).  */
    INTERNAL_SIZE_T     size;         /* Size in bytes, including overhead. */

    struct malloc_chunk* fd;          /* double links -- used only if free. */
    struct malloc_chunk* bk;

    /* Only used for large blocks: pointer to next larger size. */
    struct malloc_chunk* fd_nextsize;
    struct malloc_chunk* bk_nextsize; /* double links -- used only if free. */
}
```

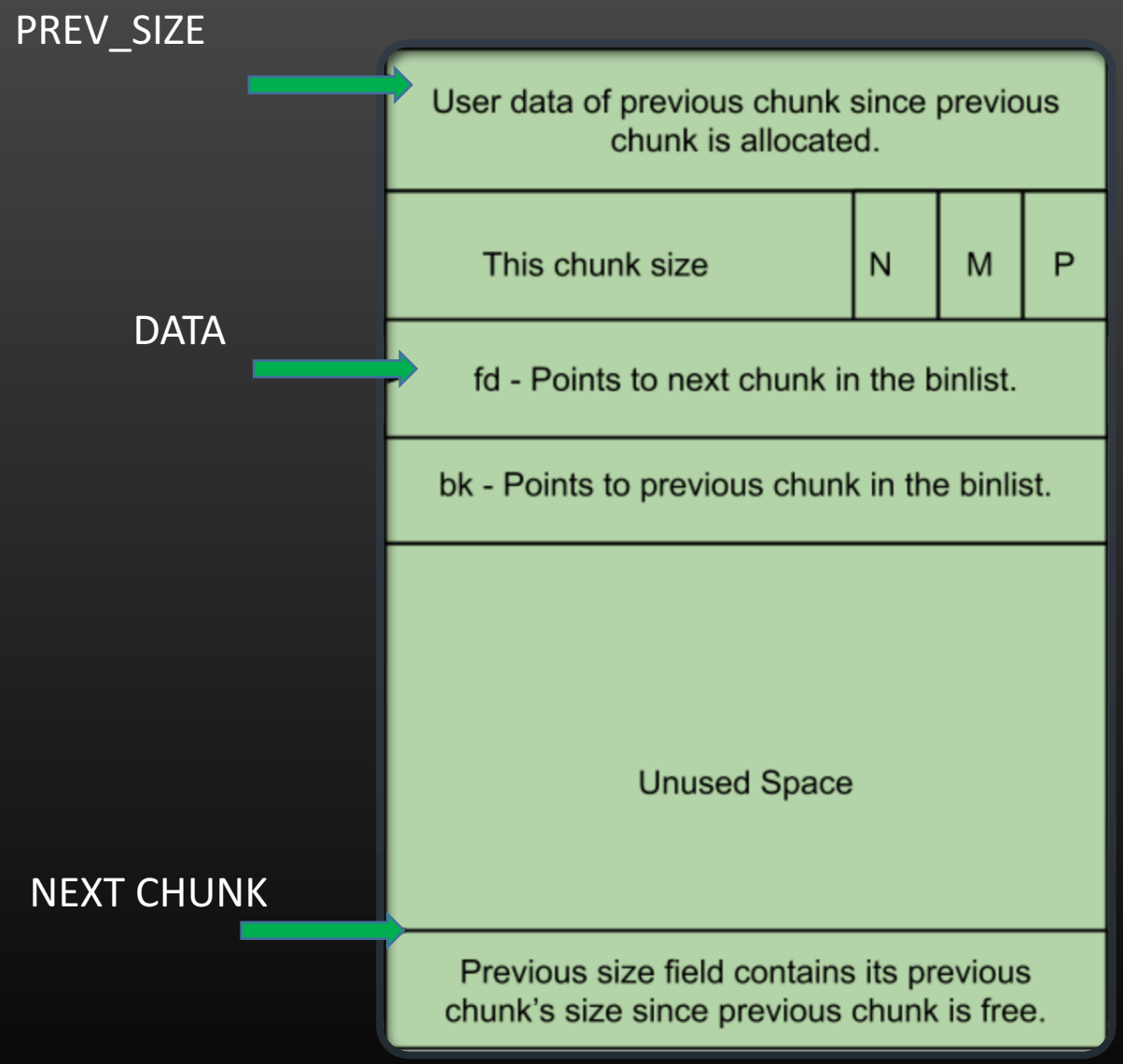
Allocated Chunk

- **PREV_SIZE:** size of previous chunk if allocated.
- **SIZE**: size of the chunk. The 3 lsbits are ignored (always 0) for size purposes.
- **PREV_INUSE(P)**: bit is set if previous chunk is in use
- **IS_MMAPED(M)** – bit set if chunk was mmap'd
- **NON_MAIN_ARENA (N)** – bit set when chunk belongs to a thread arena



Free Chunk

- **PREV_SIZE**: previous chunk's user data
- **SIZE**: size of the chunk. The 3 lsbits are ignored (always 0) for size purposes.
- **PREV_INUSE(P)**: bit is set if previous chunk is in use
- **IS_MMAPED(M)** – bit set if chunk was mmap'd
- **NON_MAIN_ARENA (N)** – bit set when chunk belongs to a thread arena



Top Chunk

- Used to service user request when there are **NO FREE CHUNKS** in any bin.
- Features:
  - If top_chunk->size > requested->size, it is split in two
    - User chunk (requested size)
    - Remainder chunk (of remaining size)
  - If not, the top chunk is extended using the sbrk or mmap syscalls.

Last Remainder Chunk

- This chunk is the remainder from the most recent split
- Why?
  - Helps to improve locality of reference, increasing performance

Coalescing

- Two chunks which are free can’t be adjacent, so they are combined into a single **Free chunk**.

- Why?
  - It eliminates external fragmentation, but it slows up free!!!

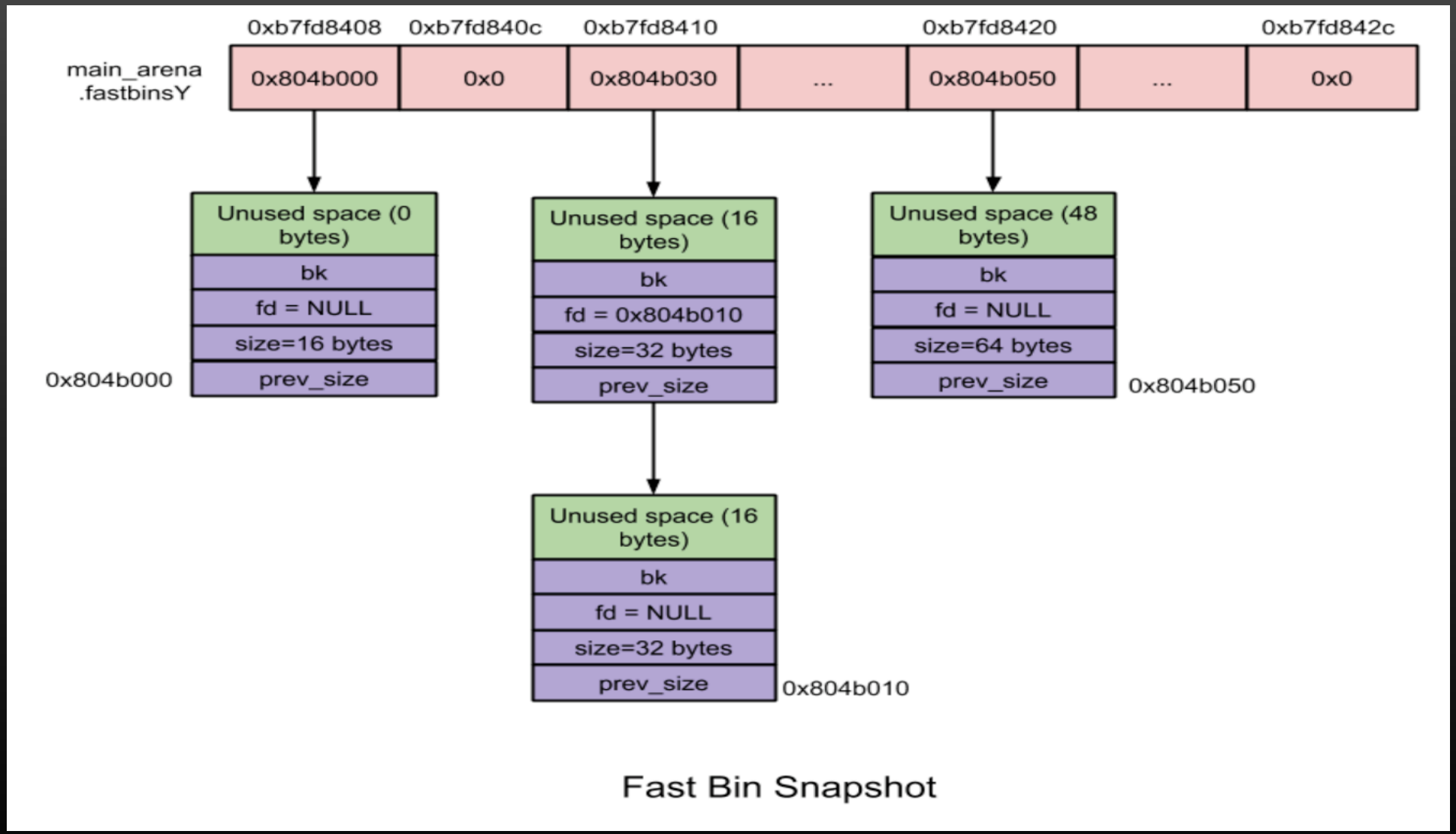
Main Arena - Bins

- Bins are the free-list data structures.
- They are used to hold free chunks.
- Based on the chunk size, different bins are available:
  - Fast bin
  - Unsorted bin
  - Small bin
  - Large bin

Fast Bins

- Singly-linked list
- 16 bytes <= chunk->size <= 80 bytes
- Non Coalescing
  - Chunks that belong to a Fast bin don’t coalesce

Fast Bins


0xb7fd8408
0xb7fd840c
0xb7fd8410
0xb7fd8420
0xb7fd842c
main_arena
.fastbinsY
0x804b000
0x0
0x804b030
...
0x804b050
...
0x0
Unused space (0 bytes)
bk
fd = NULL
size=16 bytes
prev_size
0x804b000
Unused space (16 bytes)
bk
fd = 0x804b010
size=32 bytes
prev_size
Unused space (48 bytes)
bk
fd = NULL
size=64 bytes
prev_size
0x804b050
Unused space (16 bytes)
bk
fd = NULL
size=32 bytes
prev_size
0x804b010
Fast Bin Snapshot

Small Bins

- Doubly-linked list
- 80 bytes < chunk->size <= 512 bytes
- Coalescing

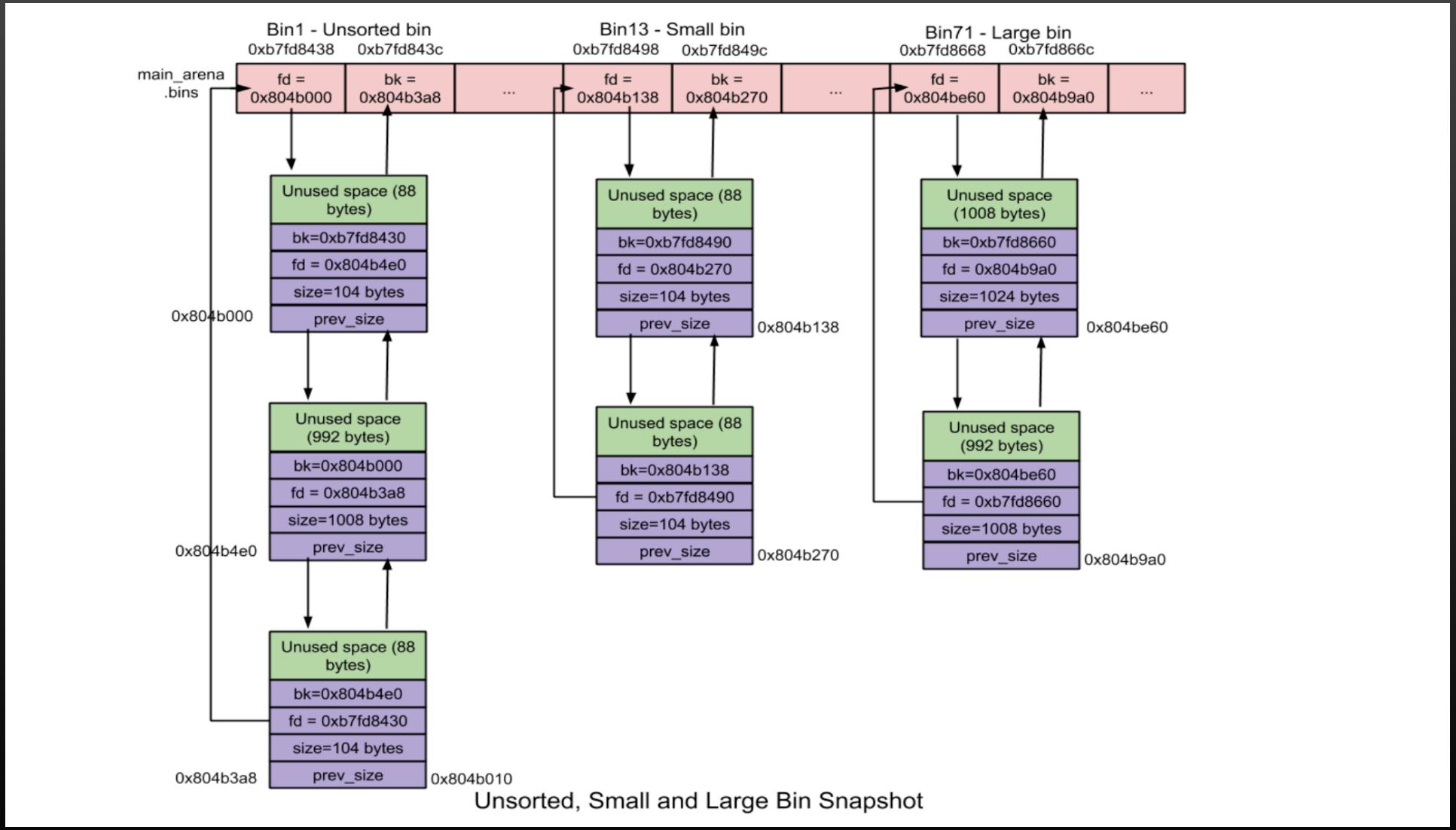
Large Bins

- Doubly-linked list
- 512 bytes < chunk->size
- Coalescing

Unsorted Bin

- When small or large chunks get freed, they are added to the Unsorted Bin
- Why?
  - Helps speed up memory allocation
- Doubly-linked list
- No size requirement

Main Arena - Bins



Unsorted, Small and Large Bin Snapshot

Tcache

- Structure introduced in glibc 2.26 in order to improve the heap's performance.
- The Tcache is a per-thread structured, stored on the heap, and it's very similar to the fast bin's structure.

Tcache Structure

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


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

Put a chunk in tcache

```
static void
tcache_put (mchunkptr chunk, size_t tc_idx) {
  tcache_entry *e = (tcache_entry *) chunk2mem (chunk);
  assert (tc_idx < TCACHE_MAX_BINS);
  e->next = tcache->entries[tc_idx];
  tcache->entries[tc_idx] = e;
  ++(tcache->counts[tc_idx]);
}
```

Get chunks from tcache

```
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;
}
```

Historical Vulnerabilities and Attacks

- Use After Free
- Double Free
- Poison null byte
- Unsafe Unlink
- Fastbin dup/poison
- Tcache dup/poison