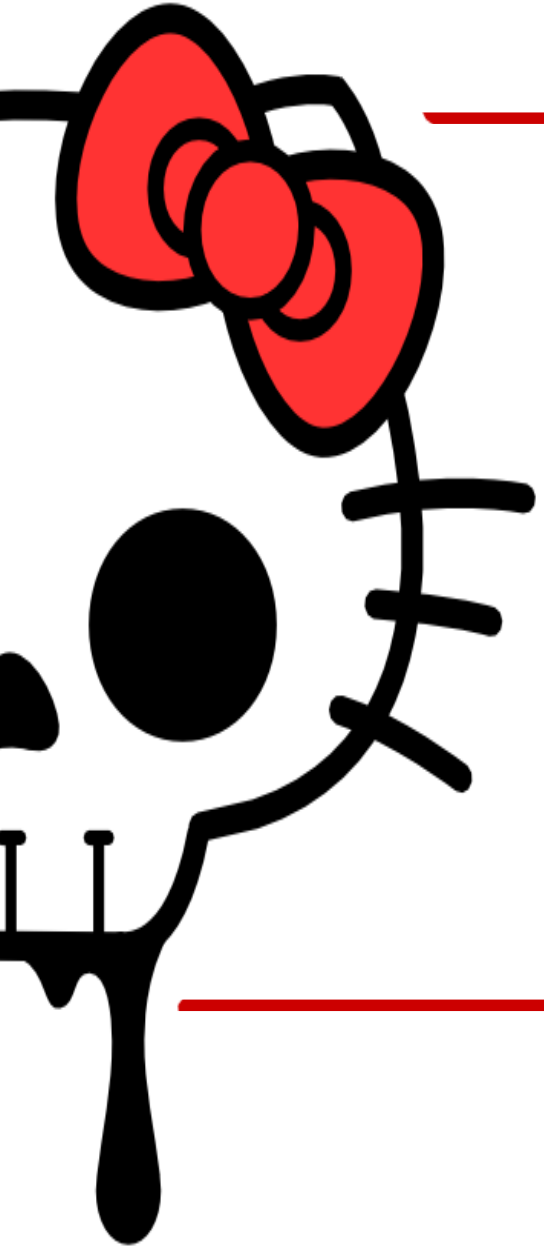
# Glibc Heap Analysis in Linux Systems with Radare2

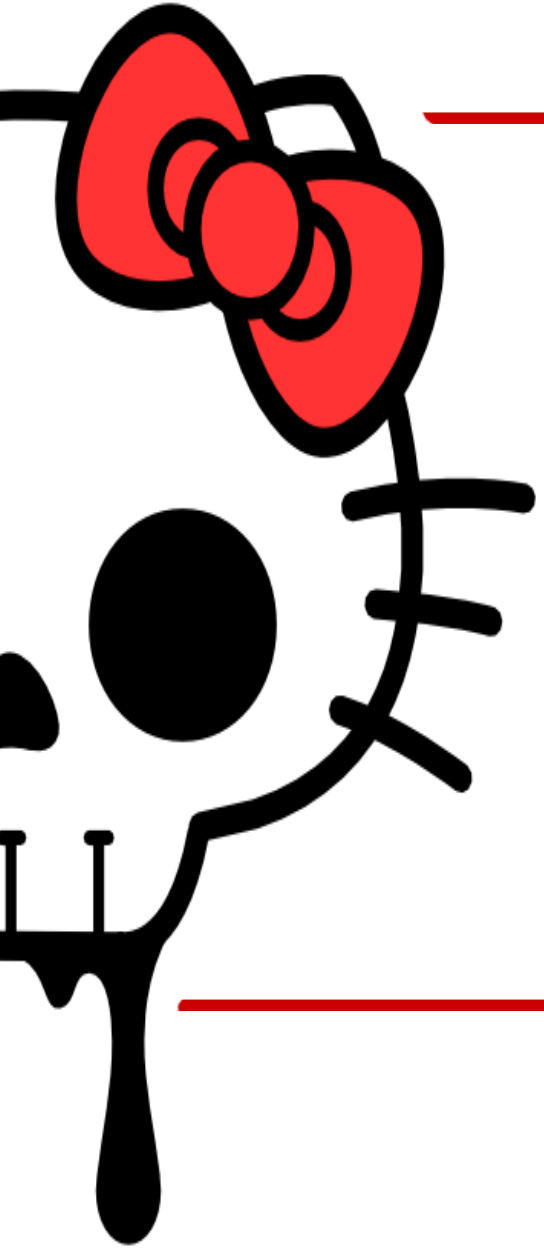

# Introduction



## n4x0r@r2con:~/r2_heap$ whoami

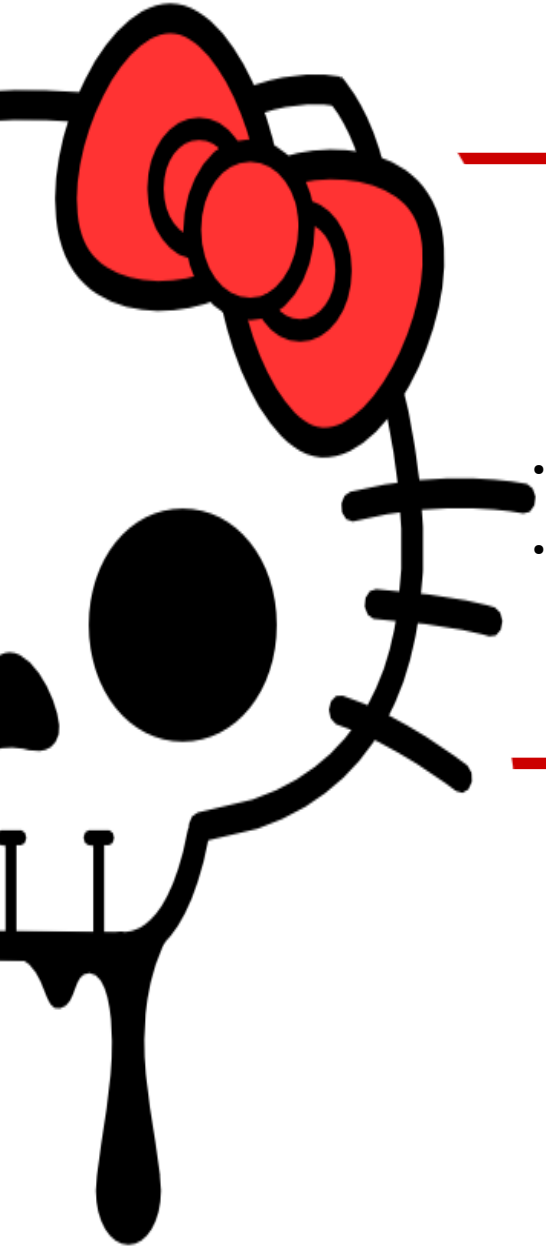
- CTF player at am3s1a_team.
- Computer Science Undergraduate at The University of Liverpool, UK.
- Mainly focused on binary Exploitation and Reverse Engineering.

# Talk Layout



1. Contributors
2. Motivation
3. Glibc malloc Heap Essentials
4. Dmh command family
5. Conclusion
6. Questions

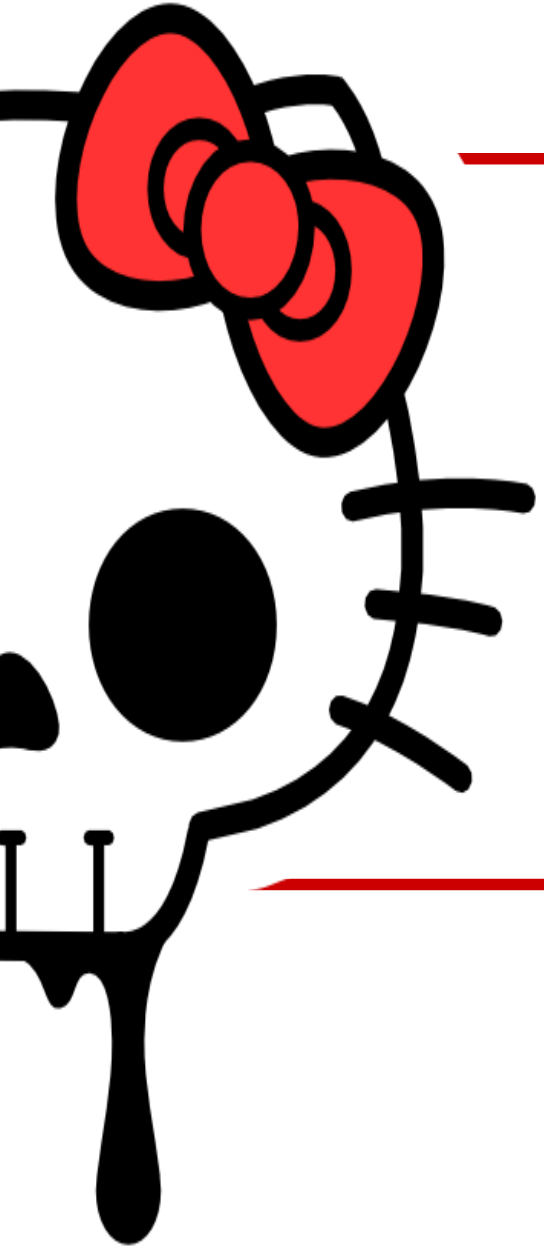
# Contributors



## n4x0r@r2con:~/r2_heap$ ls -l

- -rwxr--r-- 1 @javierprtd @amn3s1a_team 373 Jul 9 23:50 Javier_Partido
- -rwxr--r-- 1 @n4x0r_ @amn3s1a_team 373 Jul 9 23:50 Ignacio_Sanmillan

# Motivation



- Mainly focused on Exploitation and Reverse Engineering in CTF comptetions.
- Seeking for a tool to analyze glibc malloc Heap.
- Found lack of quality features in known debuggers to analyze the Heap efficiently.
- GDB plugins did not meet expectations.

- Outcome: To Implement something on our own.
- Problem: Choosing framework to implement it.
- Solution: Radare2

# Glibc malloc
# Heap Essentials

Glibc malloc Heap Essentials
- Heap Overview

- Heap creation

- Heap structures
  + Malloc_chunk
  + Malloc_state
  + Heap_Info

- Intro to glibc malloc algorithm

# Heap Overview

- **What is the heap?**

  **Region of memory created at runtime intended to allocate variables which size is not deterministic at compile time.**

  - **2 functions which make up the foundations of heap manipulation. This functions are:**

## Malloc:

- Prototype: void* malloc(size_t size)
- Functonality:
  - First time malloc is called in thread:
    - Allocates a reasonable amount of memory,
    - Creates a heap segment or equivalent,
    - Return the caller a pointer to a memory region within it of suitable for demand.
  - Not the first call to malloc in thread:
    - malloc will just return a pointer to a region within the current heap segment (or equivalent) of suitable size for the caller demand.

## Free:

- Prototype: void free(void *ptr)
- Functionality:
  - Set a memory region previously returned from malloc as not in use.

- **A little history about allocators**

  **The heap segment has to be created at runtime by an algorithm.**

  **Algorithms that allocate memory dynamically and administrate it are denominated allocators.**

- **There are various allocators that use malloc, free ...etc, however even though their functionality resembles similar, their implementation is quite different.**

**Some known allocators are:**

- **Dlmalloc: General purpose allocator**
- **Ptmalloc: (ptmalloc2) Glibc**
- **Jemalloc: FreeBSD and Firefox**
- **Tcmalloc: Google Chrome**
- **Magazine malloc: IOS/OSX**

**This talk is based on the glibc malloc algorithm (glibc version 2.24) since is the most wide spreaded within linux distributions.**

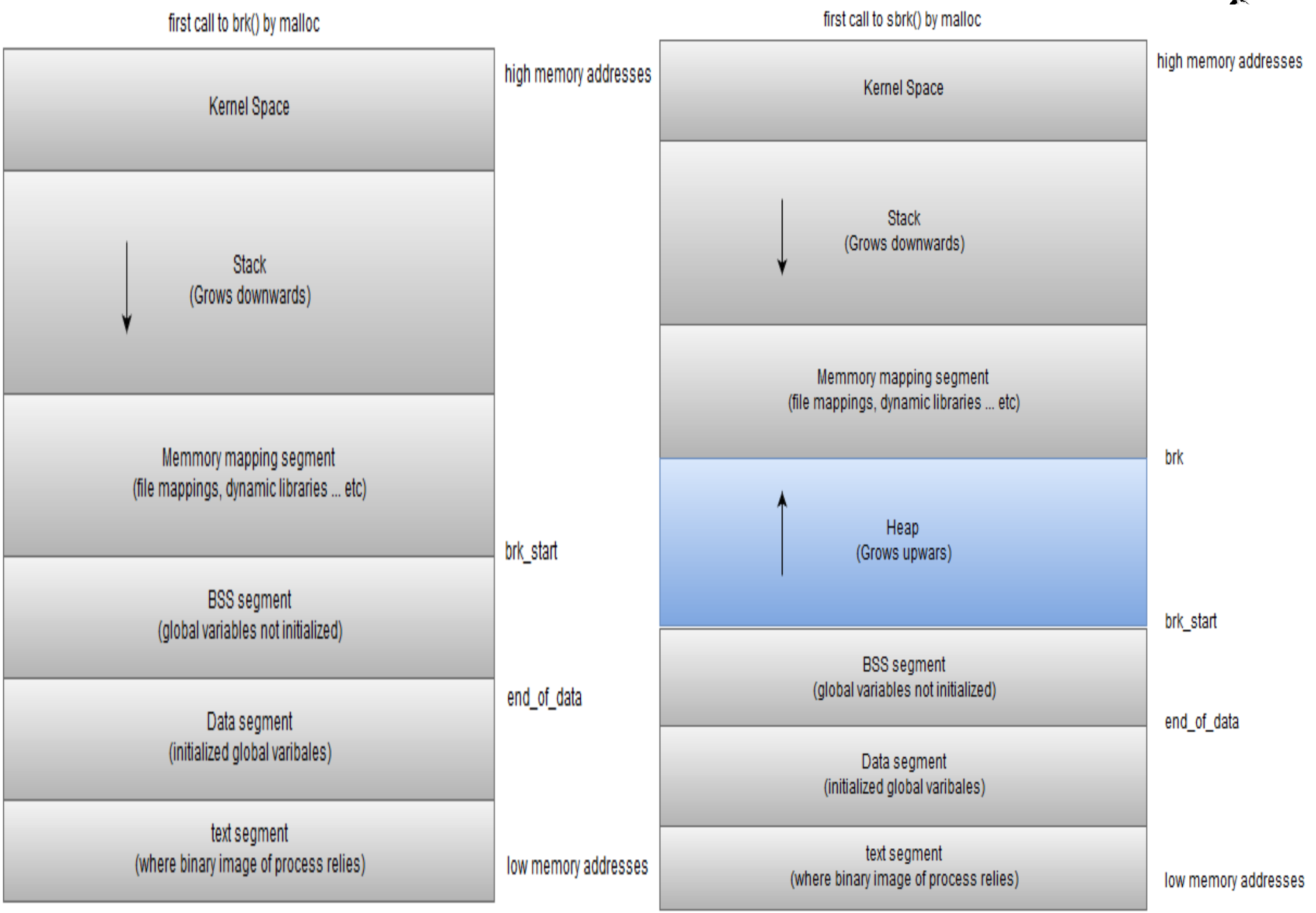
- **A little history about glibc malloc:**
  - **Based on the ptmalloc allocator (pthreads malloc) version 2, which was originally derived from dlmalloc allocator.**
  - **Mayor differences between ptmalloc2 and dlmalloc:**
    - **ptmalloc2 supports heaps on different threads aswell as multiple heaps in child threads.**
  - **The actual implementation of glibc malloc may vary from ptmalloc2 implementation due to different changes in glibc malloc from its ptmalloc2 integration.**

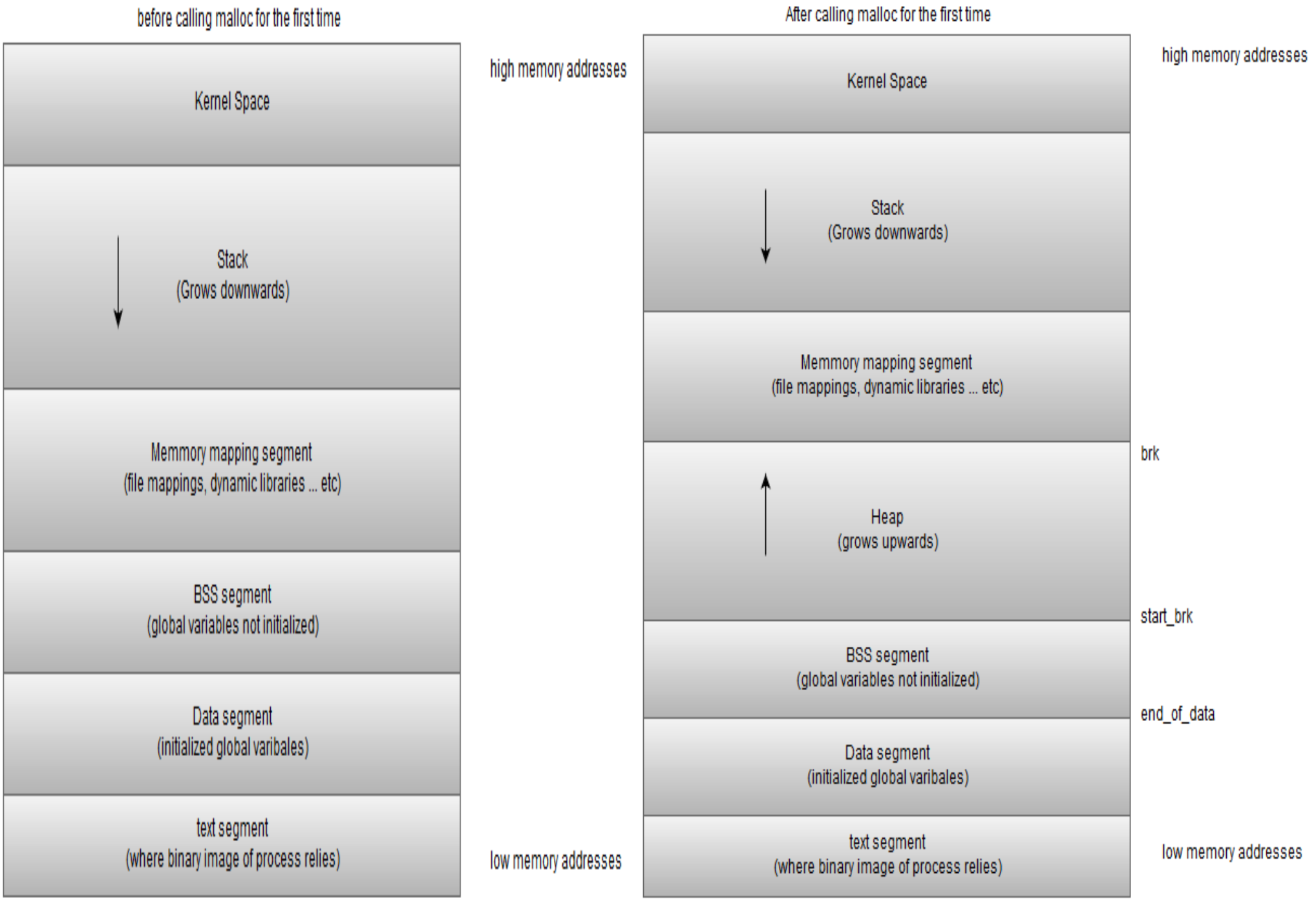
# Heap Creation

- **Some definitions:**
  - **End_data:** pointer which sets the end of the data segment.
  - **start_brk:** pointer which denotes the beginning of the heap segment.
  - **Brk / program break:** pointer that denotes the end of the heap segment.
- **Allocators communicate with the kernel in order to retrieve memory from the OS via systemcalls. In glibc malloc these system calls are the following:**

- **The brk() systemcall:**
  - **Prototype: int brk(void *addr);**
  - **Initializes:**
    - **end_data to the value specified by addr.**
    - **start_brk to the next available segment after the data segment.**
    - **brk = start_brk.**
  - **In order to expand or shrink the dimensions of the heap segment, an additional glibc function is used. This function is sbrk().**

- **The sbrk() function:**
  - **Prototype: void *sbrk(intptr_t increment);**
  - **wrapper function around brk(),**
  - **increments the program break by increment bytes, therefore giving dimensions to the heap segment.**
  - **Calling sbrk() with an argument of 0 can be used to find the current location of the program break.**


first call to brk() by malloc
Kernel Space
Stack
(Grows downwards)
Memmory mapping segment
(file mappings, dynamic libraries ... etc)
BSS segment
(global variables not initialized)
Data segment
(initialized global varibales)
text segment
(where binary image of process relies)
high memory addresses
brk_start
end_of_data
low memory addresses
first call to sbrk() by malloc
Kernel Space
Stack
(Grows downwards)
Memmory mapping segment
(file mappings, dynamic libraries ... etc)
Heap
(Grows upwars)
BSS segment
(global variables not initialized)
Data segment
(initialized global varibales)
text segment
(where binary image of process relies)
high memory addresses
brk
brk_start
end_of_data
low memory addresses


before calling malloc for the first time
Kernel Space
Stack
(Grows downwards)
Memmory mapping segment
(file mappings, dynamic libraries ... etc)
BSS segment
(global variables not initialized)
Data segment
(initialized global varibales)
text segment
(where binary image of process relies)
high memory addresses
low memory addresses
After calling malloc for the first time
Kernel Space
Stack
(Grows downwards)
Memmory mapping segment
(file mappings, dynamic libraries ... etc)
Heap
(grows upwards)
BSS segment
(global variables not initialized)
Data segment
(initialized global varibales)
text segment
(where binary image of process relies)
high memory addresses
brk
start_brk
end_of_data
low memory addresses

**Brk() / sbrk() only create the heap segment for the main thread!.**

- **The mmap() systemcall:**
  - **Prototype: void * mmap (void *address, size_t length, int protect, int flags, int filedes, off_t offset)**
  - **Functionality:**
    - **Mmap() will be called in 2 different scenarios:**

      **1) Demanded size for a malloc call in the main thread exceeded the macro DEFAULT_MMAP_THRESHOLD which has a value of 128 * 1024**

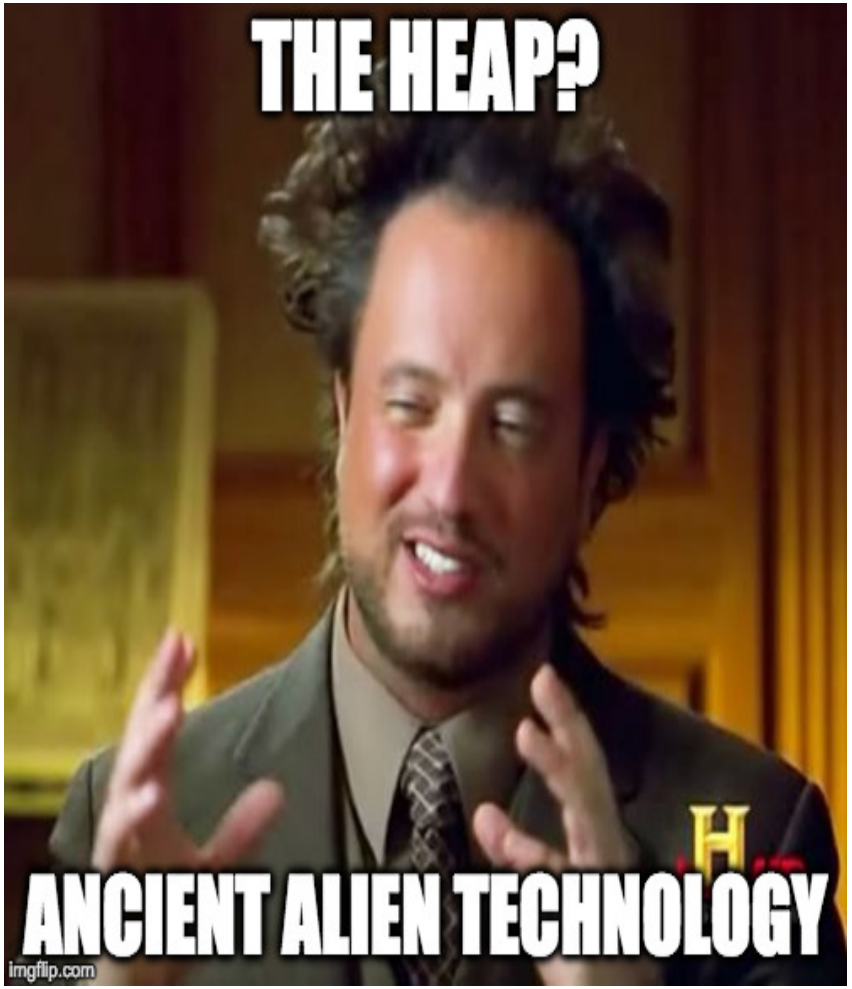
      **2) Call to malloc proceeded form a non main thread.**

- **How does mmap() work?:**
  - **treats child “heap segment” as private anonymous allocations.**
  - **Allocates a region of memory in the memory mappings segment filled with 0s, unlinked from any file and being private for each thread.**
  - **Mmap si capable to do this with the following flags:**
    - **MAP_ANON**
    - **MAP_PRIVATE**

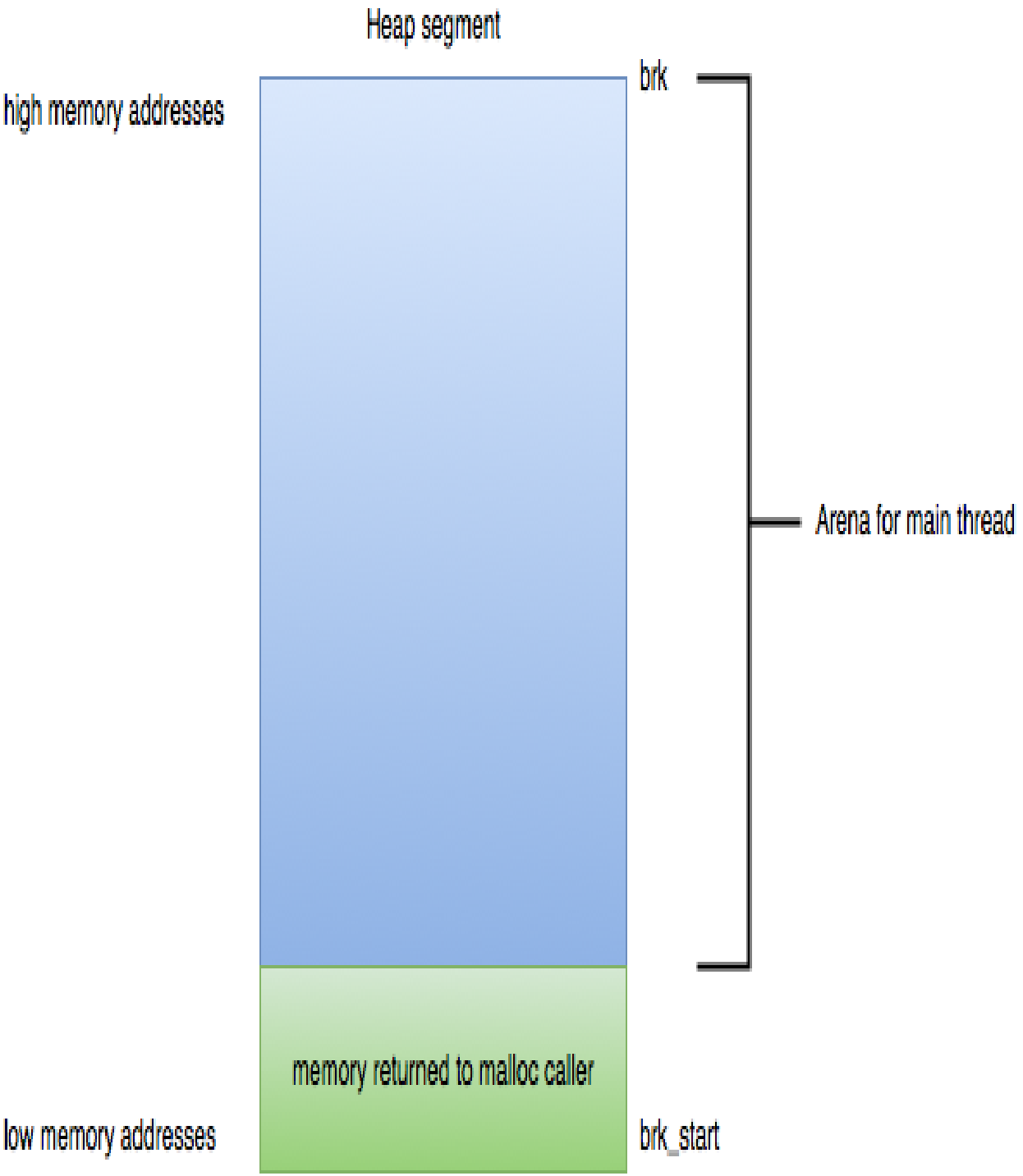

Concurrent application with multiple Heaps
Kernel Space
high memory addresses
Stack
(Grows downwards)
Memory map segment
Heap for child thread
Heap Segment
(Just for main Thread if demanded size siutable)
BSS segment
(global variables not initialized)
Data segment
(initialized global varibales)
text segment
(where binary image of process relies)
low memory addresses

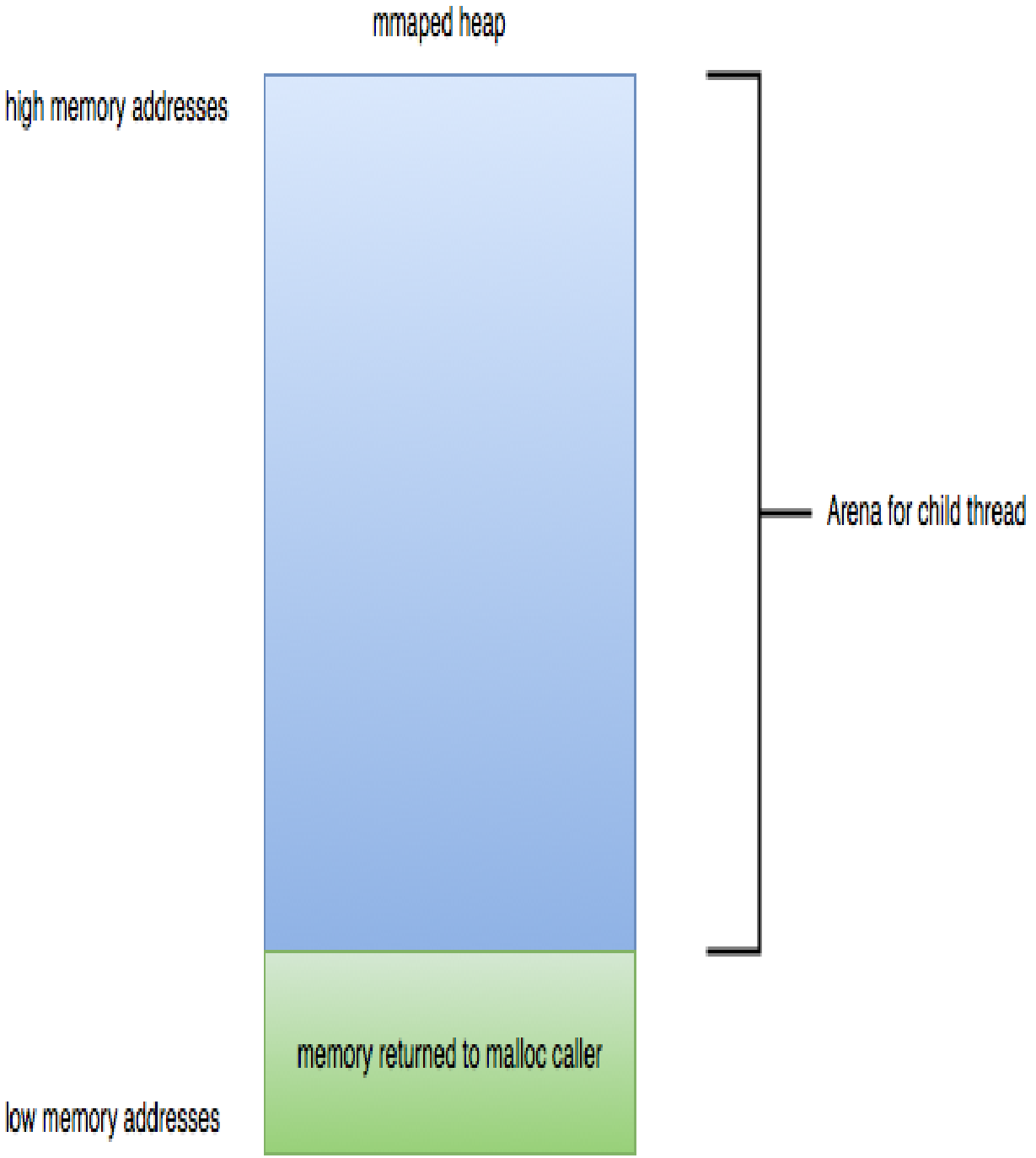
# Heap Organization

- **Before digging any further into glibc malloc heap internas, first we have to understand a few concepts:**



- **Arena:**
  - **Product of optimization procedure to avoid unecessary interaction with the kernel while requesting memory for dynamic allocation.**
  - **An arena is all the available memory left that was retrieved from the OS to serve all potential malloc calls demands for a particular thread.**


Heap segment
high memory addresses
brk
Arena for main thread
memory returned to malloc caller
low memory addresses
brk_start


mmaped heap
high memory addresses
Arena for child thread
memory returned to malloc caller
low memory addresses

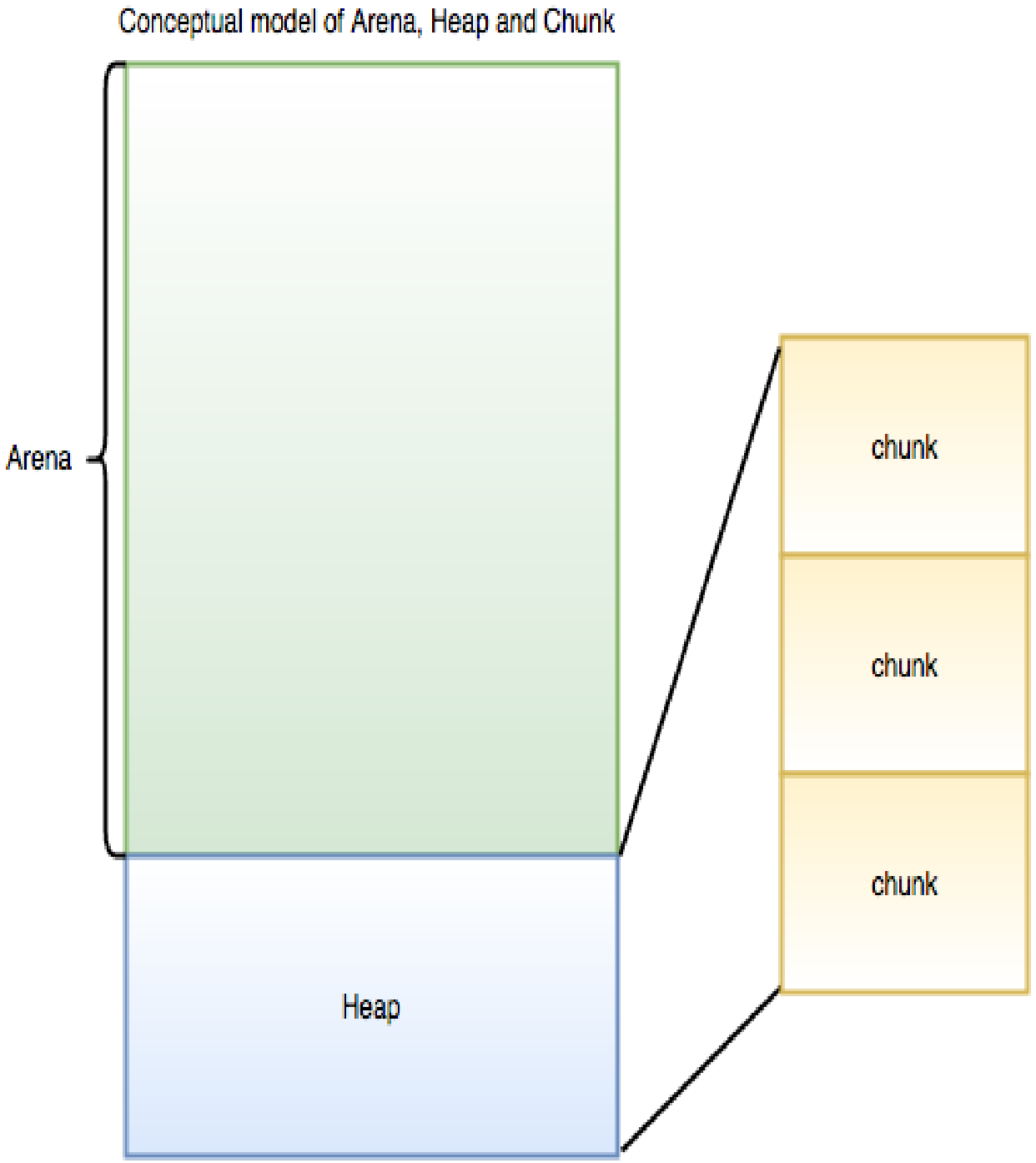
- **There is a limit to the number of arenas that a concurrent application may support.**
- **This limit is determined based on the number of cores the system has.**
  - **limit = (32bit) ? number of cores * 2 : number of cores * 8.**
- **This limit can also be controlled using the MALLOC_ARENA_MAX environment variable.**

- **Heap**

  **Nothing but a contiguous region of memory that is composed of pieces called chunks.**
  **Each heap belongs to exactly one arena.**
  - **Heap != Heap segment.**

- **Chunk**

  **Smallest unit of memory administration in dynamic allocation.**
  **When malloc is called, it returns a pointer to a chunk with suitable size for the particular call.**
  **Each chunk exists in one heap and belongs to one arena.**


Conceptual model of Arena, Heap and Chunk
Arena
Heap
chunk
chunk
chunk

# Heap Structures

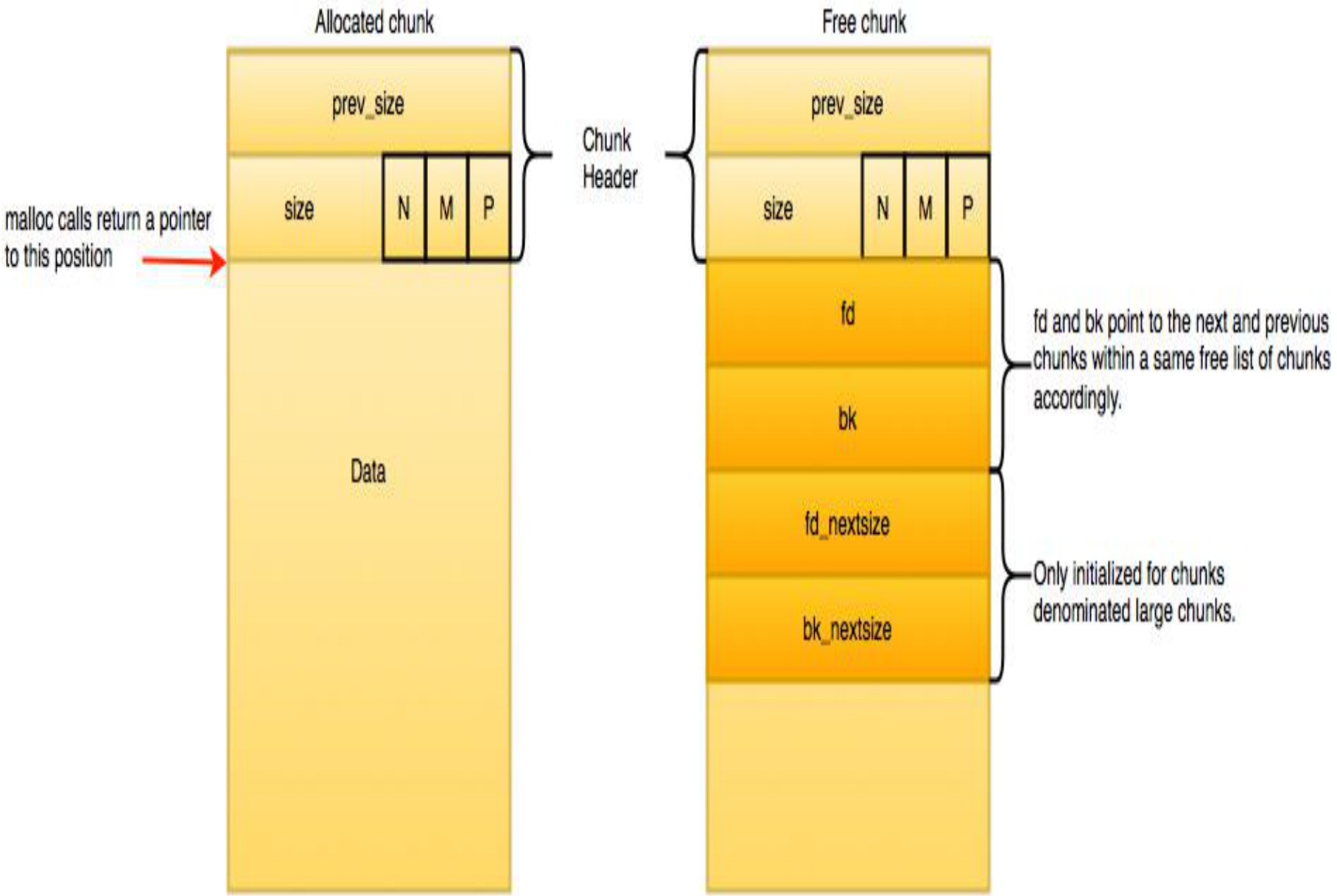
- **The previous concepts are implemented in the form of structures. A few of the most relevant structures in glibc malloc are the following:**
  - **malloc_chunk :** **all the elements that makeup a chunk.**
  - **malloc_info:** **all elements that interact with the heap or heaps for a particular child thread.**
  - **malloc_state:** **all elements that compose an arena.**

**malloc_chunk**:

```
  struct malloc_chunk {

    INTERNAL_SIZE_T     prev_size;  /* Size of previous chunk (if free).  */
    INTERNAL_SIZE_T     size;       /* Size in bytes, including header. */

    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; /* double links -- used only if free. */
    struct malloc_chunk* bk_nextsize;
};
```

- **Chunks can either be free or allocated.**
- **Their structure change accordingly**
  - **prev_size field is only initialized if chunk is free.**
  - **When chunks are freed, they get insterted in linked list of free chunks.**
    - **Fd, bk are then initialized.**
    - **Link a particular chunk to others within a same linked list.**
    - **(more about this in coming slides).**

- **3 LSB of size field represent chunk flags, these are:**
  - **PREV_INUSE (P) : bit set when previous chunk in heap is allocated.**
  - **IS_MMAPPED (M) : bit set when chunk is being mmap'd.**
  - **NON_MAIN_ARENA (N): bit set when chunk does not belong to heap segment**


Allocated chunk
prev_size
size
N
M
P
Data
malloc calls return a pointer to this position
Chunk Header
Free chunk
prev_size
size
N
M
P
fd
bk
fd_nextsize
bk_nextsize
fd and bk point to the next and previous chunks within a same free list of chunks accordingly.
Only initialized for chunks denominated large chunks.

**malloc_state**:

- **Two possible types of instances for this structure:**
  - **main_arena: instance for the main thread arena. Held in glibc data segment as a global variable.**
  - **thread_arena: instance for any other arena which does not belong to the main thread.**

# Heap Structures

```
typedef struct malloc_state{

        mutex_t mutex;                          // serialize access
        int flags;                                      // flags
        mfastbinptr fastbinsY[NFASTBINS];   //array of single linked list of free chunks

        mchunkptr top;                          // base of top chunk
        mchunkptr last_remainder;               // base of remainder chunk

        mchunkptr bins[NBINS * 2 – 2];          // array of double linked list of free chunks
        unsigned int binmap[BINMAPSIZE]; // bitmap of bins
        struct malloc_state *next;                  // arena single linked list
        struct malloc_state *next_free;         // free arena linked list

        INTERNAL_SIZE_T system_mem;         // current memory allocated for this arena
        INTERNAL_SIZE_T max_system_mem;
};
```

```
mfastbinptr fastbinsY[NFASTBINS];  //array of single linked list of
                                                free chunks

mchunkptr top;                          // base of top chunk
mchunkptr last_remainder;               // base of remainder chunk

mchunkptr bins[NBINS * 2 – 2];          // array of double linked list of
                                        free chunks
```

## Free Lists:

**What happens when free() is called?:**

**Chunk is marked as free, and its reference will be placed in one of these two data structures:**

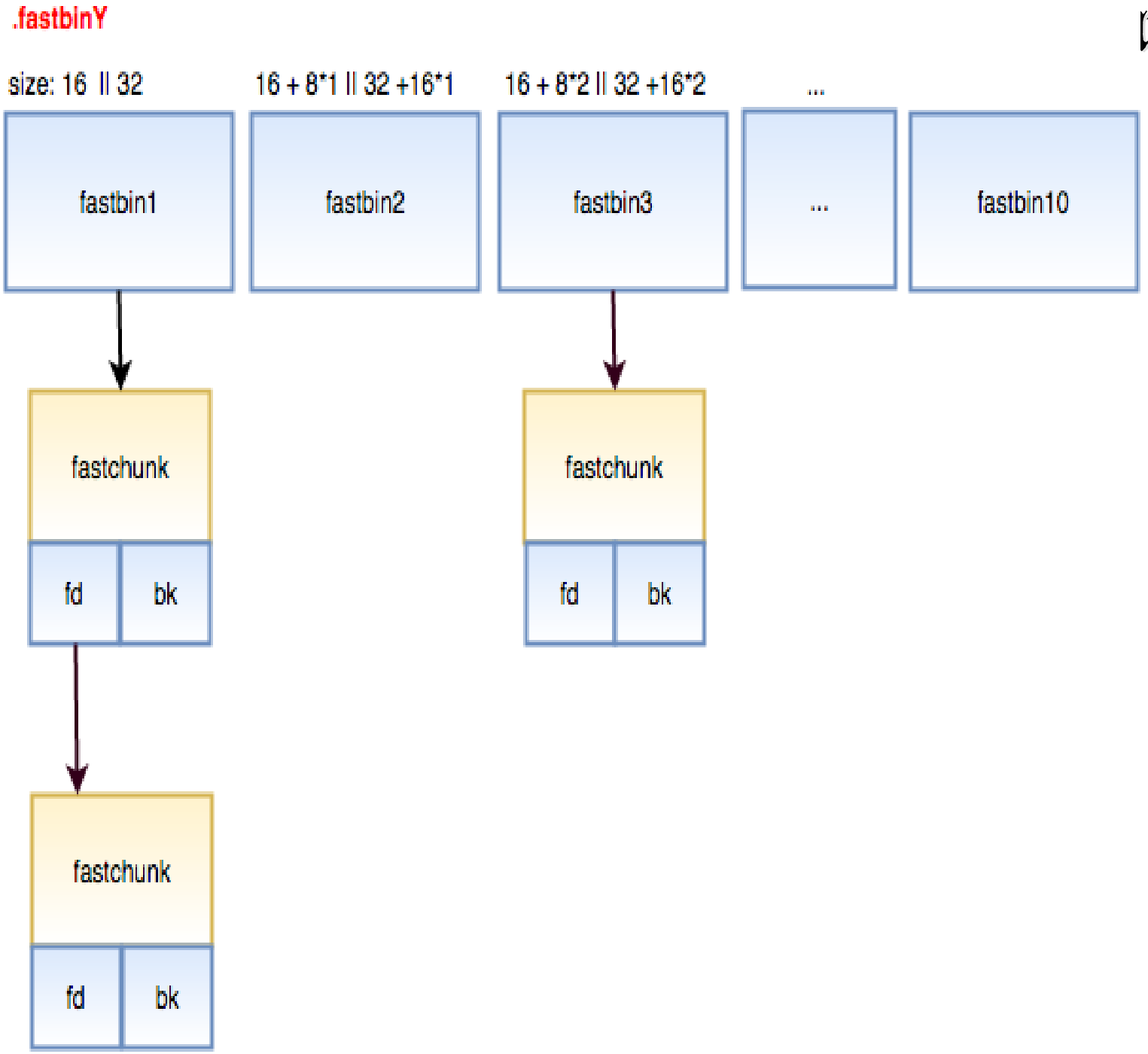
- **fastbinsY**
- **bins**

- **The purpose of these two data structures is to recycle already allocated chunks by malloc to serve future malloc calls.**

- **Less kernel interaction == More efficiency**

## FastbinY:

- Array of LIFO single linked lists holding recently freed chunks denominated fastchunks.
- Fastchunks may have sizes rangin from:
  - Fastchunk max size:
    - 80 bytes – 32bit,
    - 160 bytes – 64bit.
  - Fastchunk min size:
    - 16 bytes – 32bit.
    - 32 bytes – 64bit.
- List that hold fastchunks is denominated a Fastbin.
- Fastbin number: 10.
- Very fast for allocation and deallocation.
- Each fastbin hold chunks of the same size
- Chunk's sizes in different fastbins are spaced by:
  - 8 bytes in 32bit
  - 16 bytes in 64bit.


.fastbinY
size: 16 || 32
16 + 8*1 || 32 +16*1
16 + 8*2 || 32 +16*2
...
fastbin1
fastbin2
fastbin3
...
fastbin10
fastchunk
fd
bk
fastchunk
fd
bk
fastchunk
fd
bk

## Bins:

- **Each index in array is a FIFO double linked list of free chunks known as bin.**
- **Bin number: 128**
- **A particular bin holds free chunks of specific size/sizes depending on index.**

- **Bins for chunk sizes < 512 || 1024 bytes (small chunk) are denominated Small bins.**
  - **Chunks in small bins are called small chunks.**
  - **A Small bin will contain small chunks of all the same size.**
    - **Spaced 8 bytes - 32bit**
    - **Spaced 16 bytes - 64 bit**
  - **Exact fit search.**
  - **Placed from index 2 – 64 in bins.**

- **Bins for chunks sizes >= 512 || 1024 (large chunk) bytes are denominated Large bins.**
  - **Large bins contain chunks called large chunks.**
  - **Are approximately logarithmically spaced.**
  - **They will contain chunks within a particular size range.**
  - **Best fit search.**
  - **Placed from index 65 – 127 in bins.**

**Recap:**

- **bins[2] to bins[64] – Small bins.**
- **bins[65] to bins[127] – Large bins.**

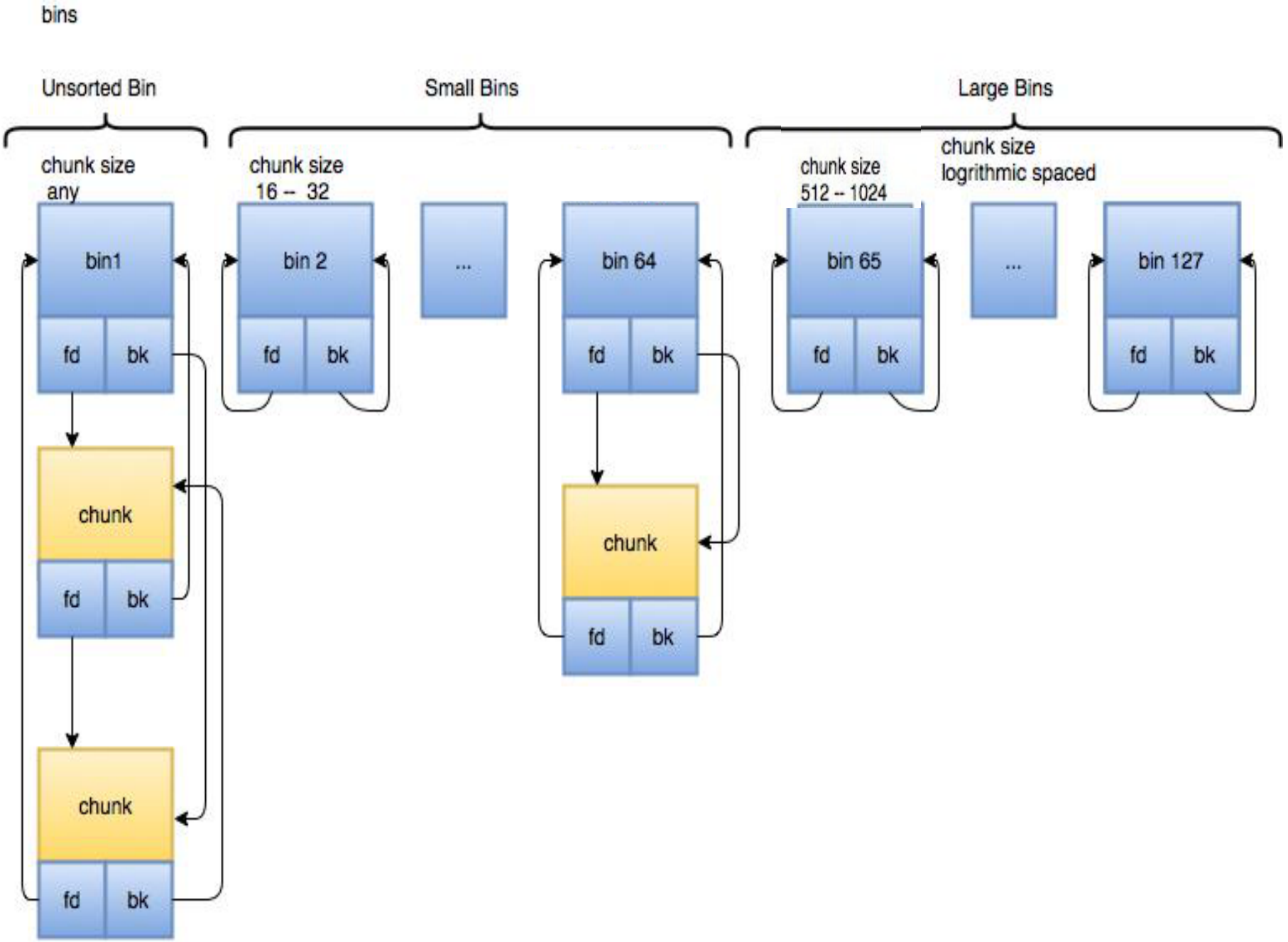
**What about bins[0] and bins[1]?**

- **bins[0] does not exist. (malloc.c #1459)**
- **bins[1] is the Unsorted bin.**

## Unsorted bin:

- Intended to reuse recent freed non fastchunks of not particular size.
- For all deallocations:
  - If chunk is a fastchunk: once freed is placed in a fastbin.
  - Else: once freed chunk is first placed in the unsorted bin.
- Chunks in the Unsorted bin are not sorted by chunk size as in other bins.
- Malloc will sort the unsorted bin in chunk requests in which:
  - Malloc did not find an exact fit in fastbins or small bins for demanded chunk.
  - Malloc can’t consolidate any fastchunk to return the user a suitiable chunk from for demand.
    (more details in coming slides).

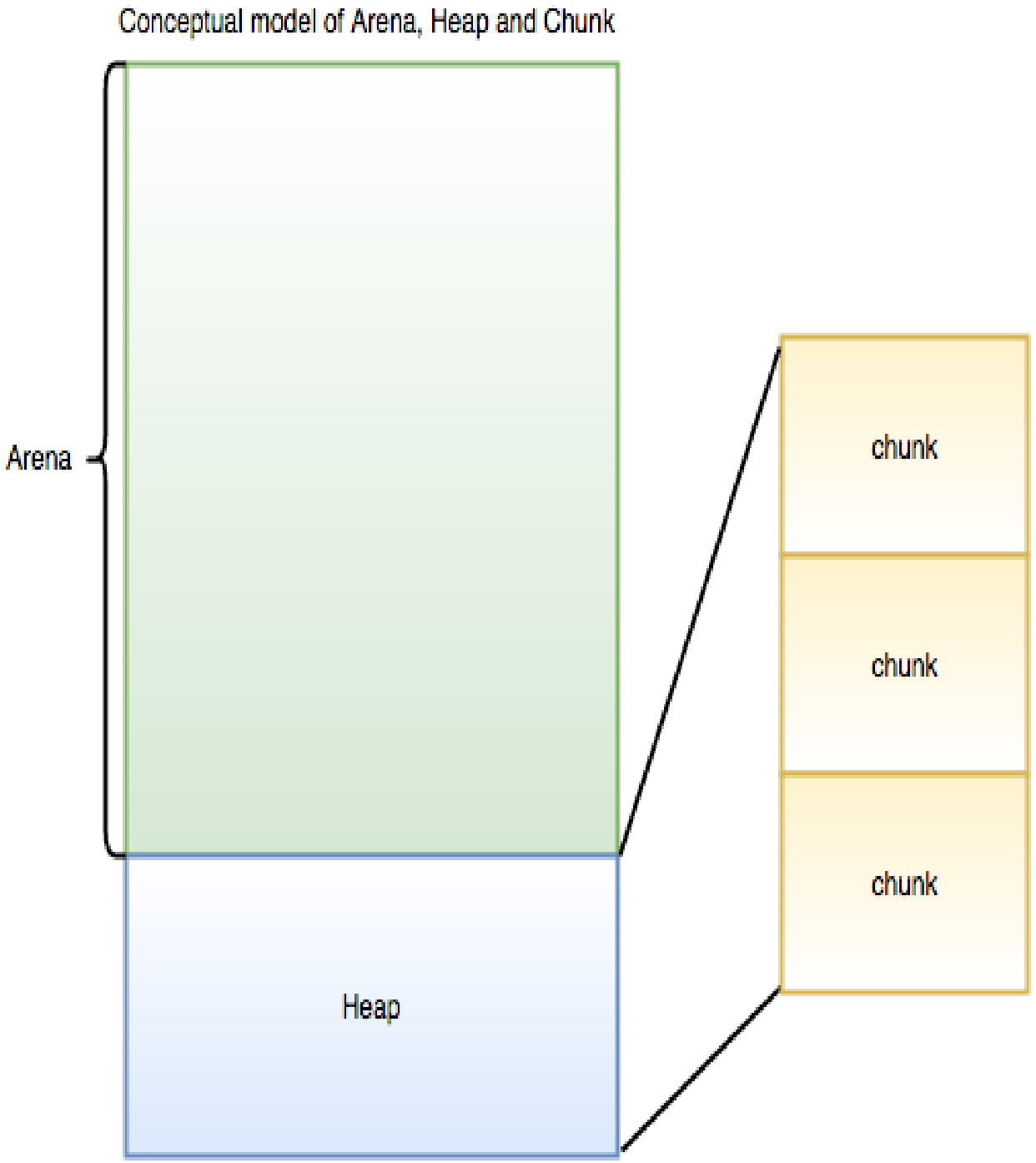
# Heap Structures

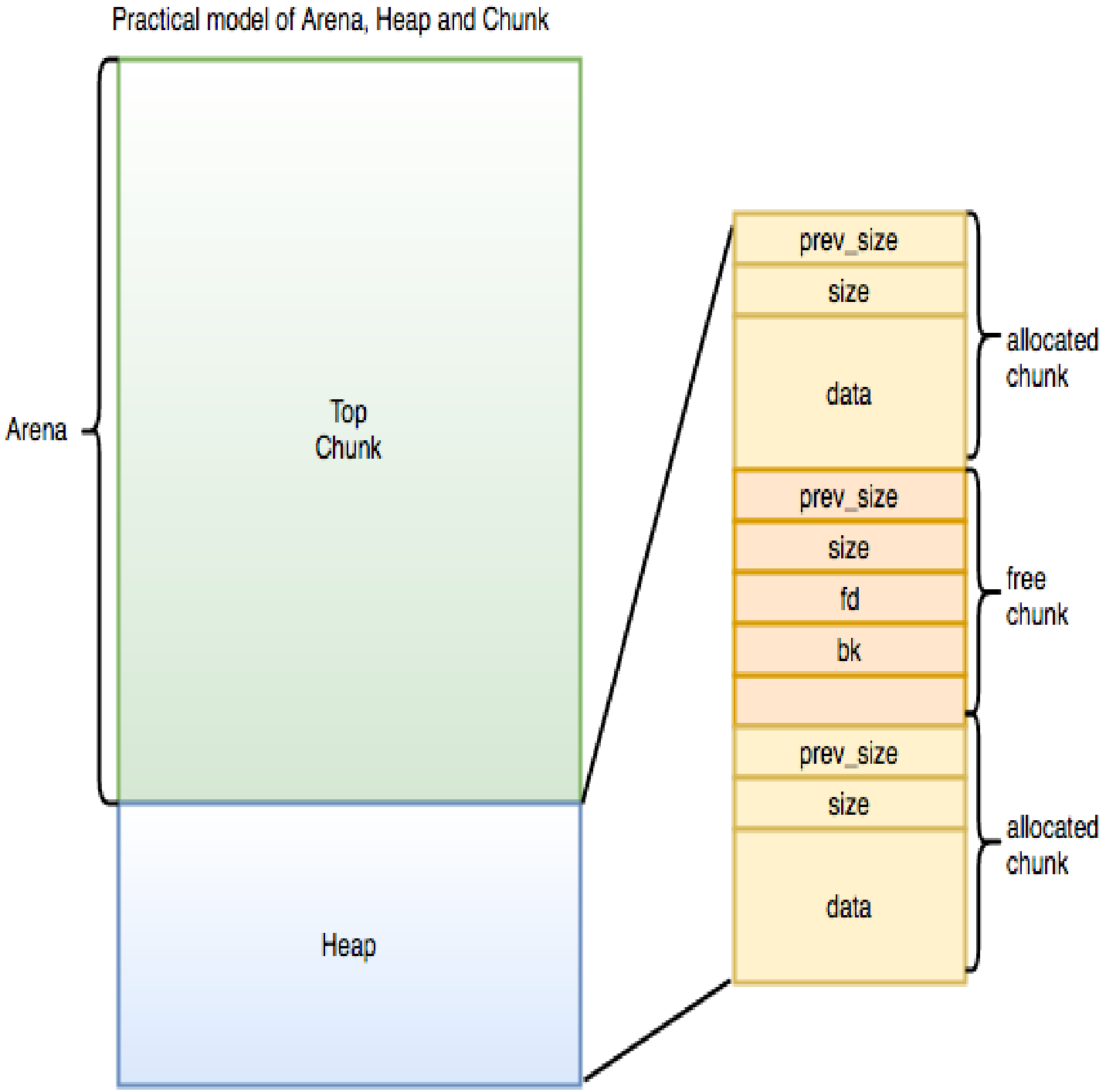


# Heap Structures

- If ideal chunk for malloc demand is not found in Free Lists, a chunk will need to be retrieved from the thread's arena in order to satify malloc's demand.

- How is an arena actually implemented?
  - The arena of a thread is implemented in the form of a chunk.
  - This chunk receives the name of top chunk.
    - It is never included in any bin.
    - is used only if no other chunk is available.
    - Top chunk is independent of heaps.

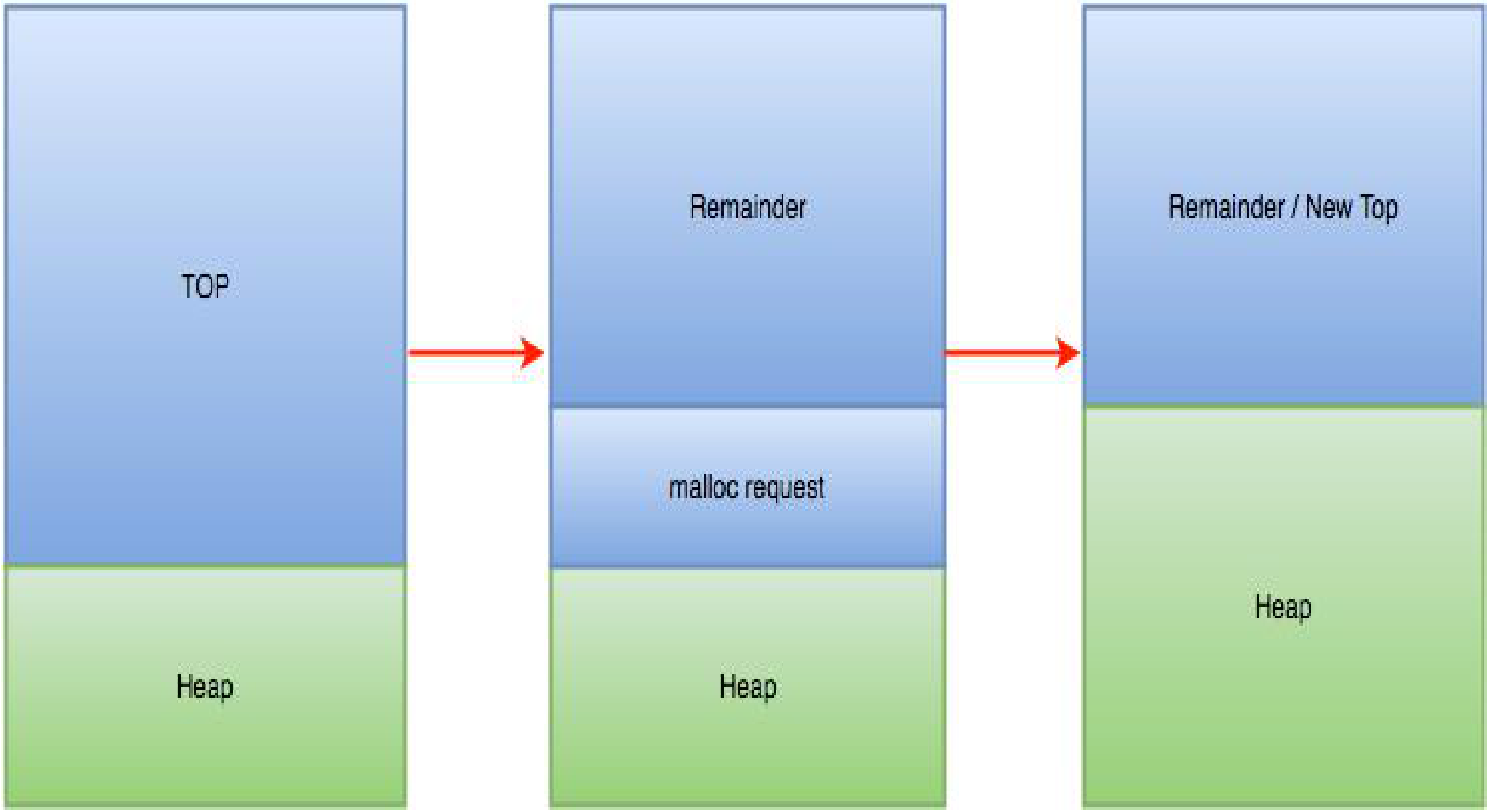

Conceptual model of Arena, Heap and Chunk
Arena
Heap
chunk
chunk
chunk


Practical model of Arena, Heap and Chunk
Arena
Top
Chunk
Heap
prev_size
size
data
allocated
chunk
prev_size
size
fd
bk
free
chunk
prev_size
size
data
allocated
chunk

# Heap Structures

- **The remainder chunk.**
  - **Product of retrieveing a chunk from arena**
  - **If ideal chunk for malloc demand was not found in free lists,memory will need to be retrieved from the thread's arena.**
    - **top chunk has to be splitted.**
  - **splitted into two parts:**
    - **Malloc requested size chunk**
    - **Remainder chunk (new top).**

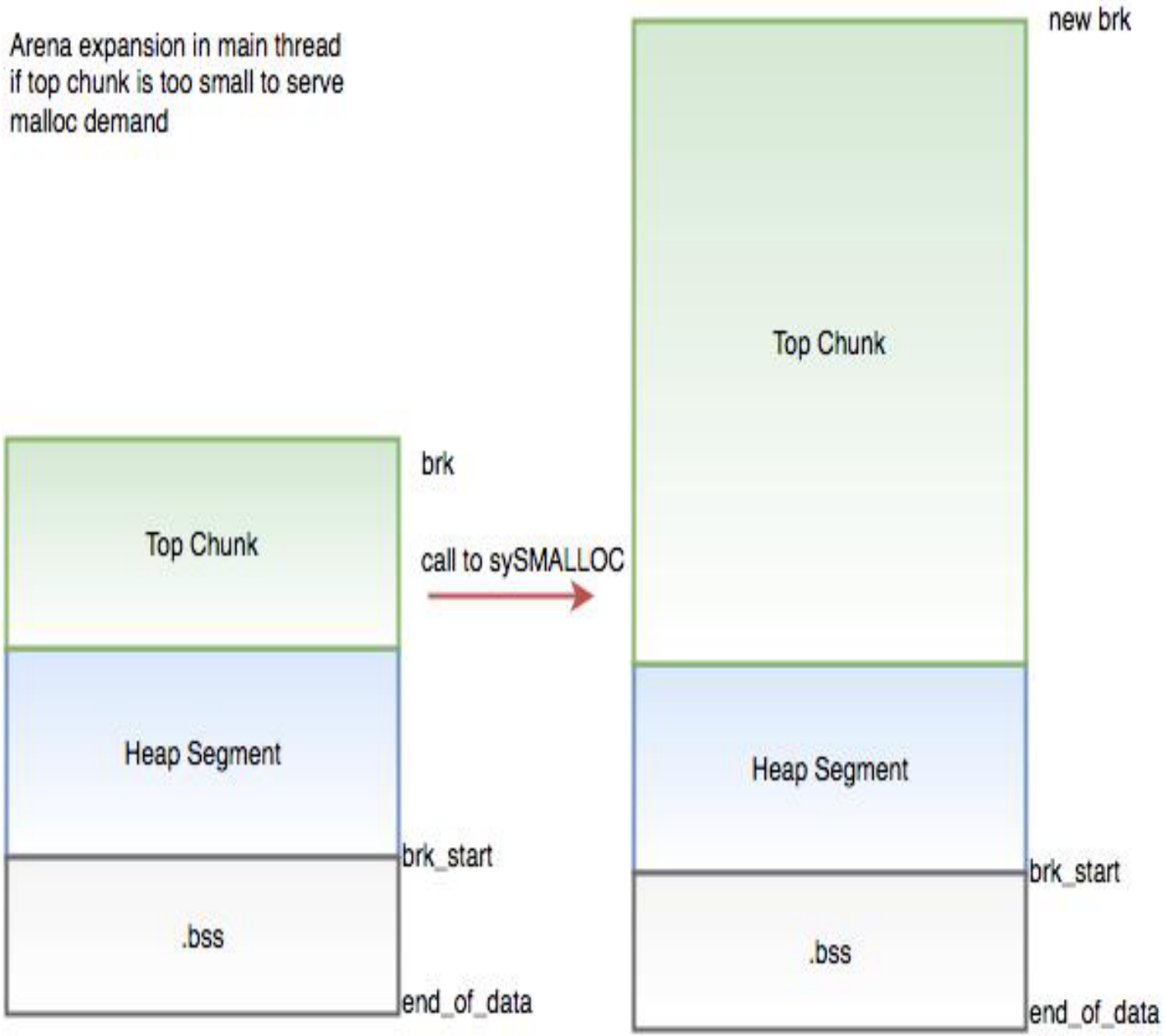
Retrieving a chunk from the heap segments's top chunk



# Heap Structures

- What about if the thread's arena is not big enough to serve a particular malloc demand?
  - Thread's arena will need to be expanded.
  - This process is different in child threads than it is for the main thread.
  - For the main thread:
    - Arena can be expanded by calling sbrk(), therefore expanding the dimensions of the heap segment by incrementing program break.

# Main thread’s arena expansion



# Heap Structures

- For child threads:
  - New mmap() call will need to be made, meaning:
    - New arena will be created and combined with previous arena.
      - New top chunk will be generated.
      - Previous top chunk will become a free chunk.
      - New Heap will be created in the newer mmaped region context.
      - These two mmaped regions will become part of the same arena, however they most likely will not be adjacent in memory.
    - To manage multiple heaps in child threads an additional structure is used. This structure is **heap_info.**

# Heap Structures

- **heap_info**

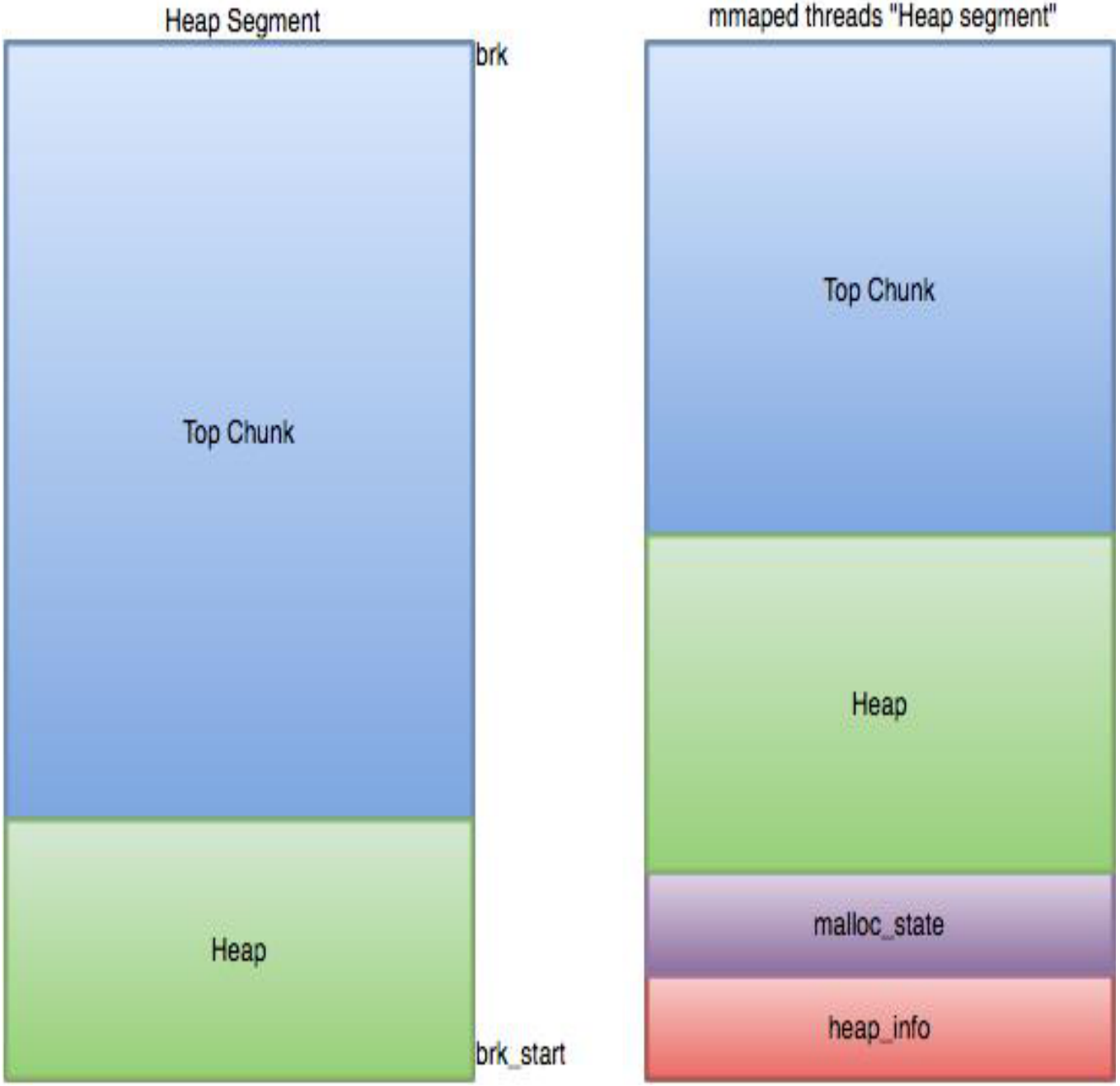
```
typedef struct _heap_info {
     mstate ar_ptr;                    /* Arena for this heap. */
     struct _heap_info *prev;          /* Previous heap. */
     size_t size;                      /* Current size in bytes. */
     size_t mprotect_size;             /* Size in bytes that has
     been                                 mprotected
     PROT_READ|
     PROT_WRITE.  */
     char pad[-6 * SIZE_SZ & MALLOC_ALIGN_MASK];
//alignment
} heap_info;
```

# Heap Structures

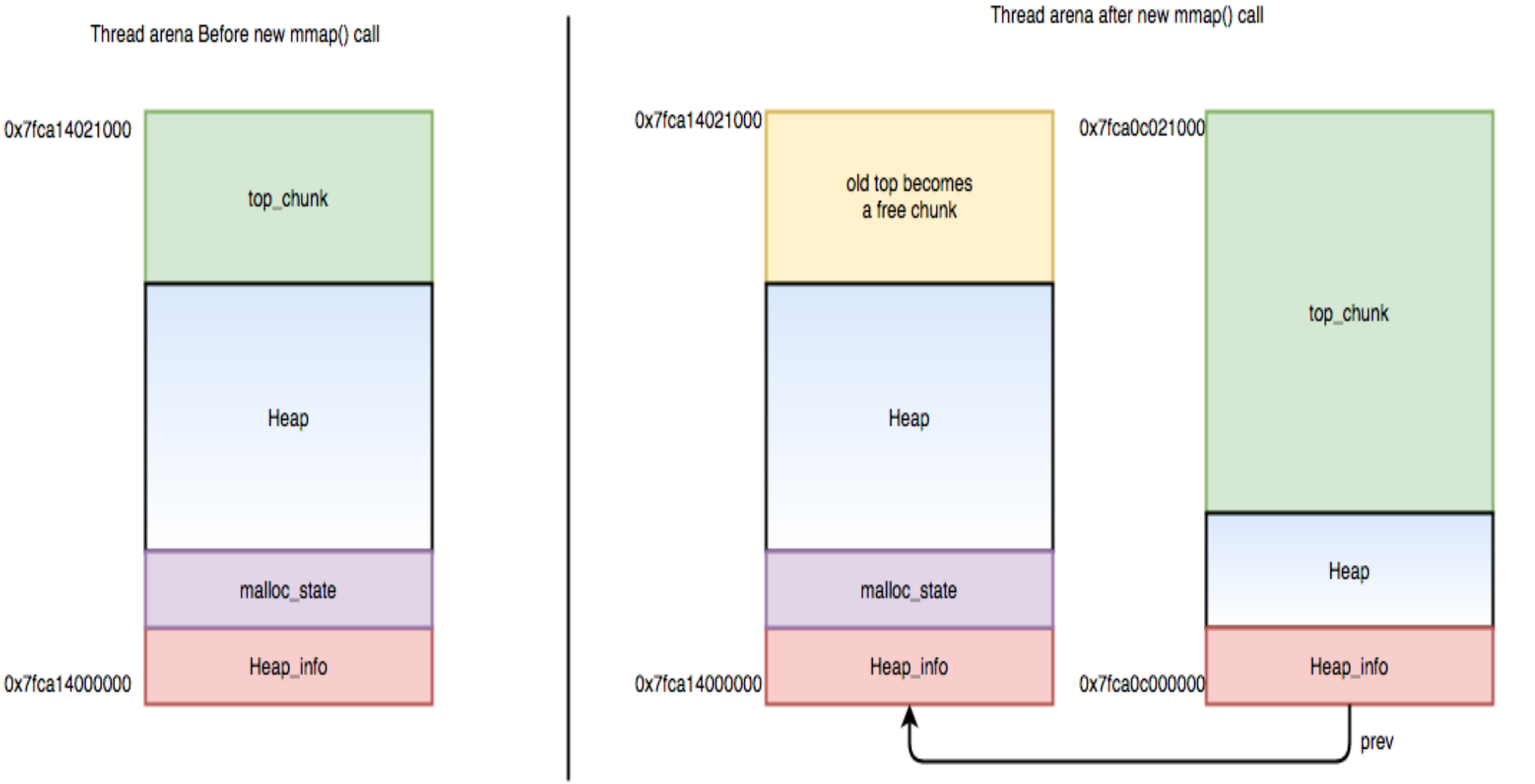
- **heap_info**

```
mstate ar_ptr;              /* Arena for this heap. */
struct _heap_info *prev;    /* Previous heap. */
size_t size;                /* Current size in bytes. */
```

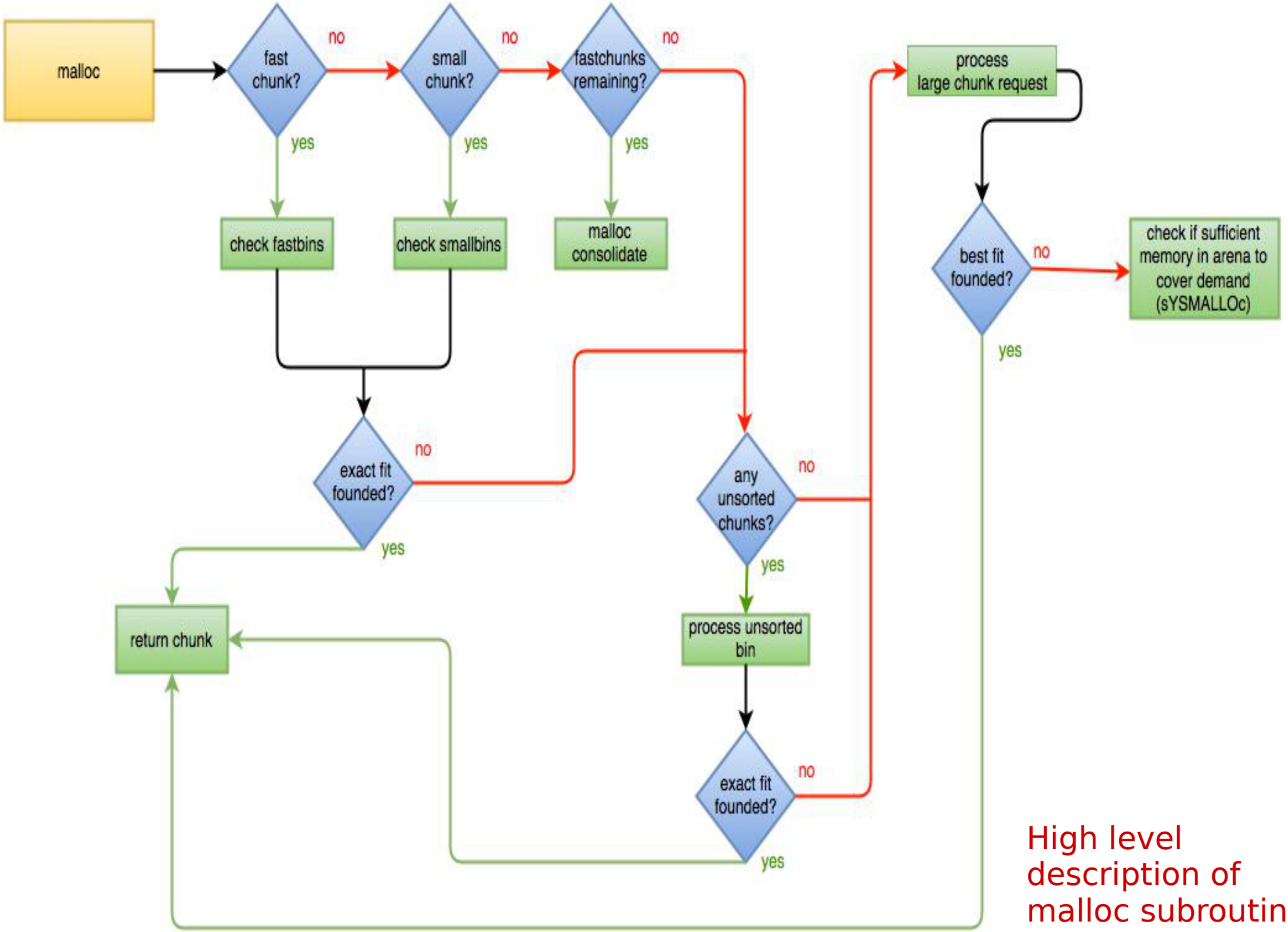
- **Intended to keep track of multiple heaps in child threads.**
- **Multiple heaps are not supported in the main thread.**
  - **heap_info does not exist in single threaded applications.**
- **The main thread also uses a different instance of malloc_state (main_arena).**
- **Instances for malloc_state and malloc_info in child threads are located at the beginning of the their particular arenas.**


Heap Segment
brk
Top Chunk
Heap
brk_start
mmaped threads "Heap segment"
Top Chunk
Heap
malloc_state
heap_info

# Heap Structures



# Introduction to glibc malloc algorithm

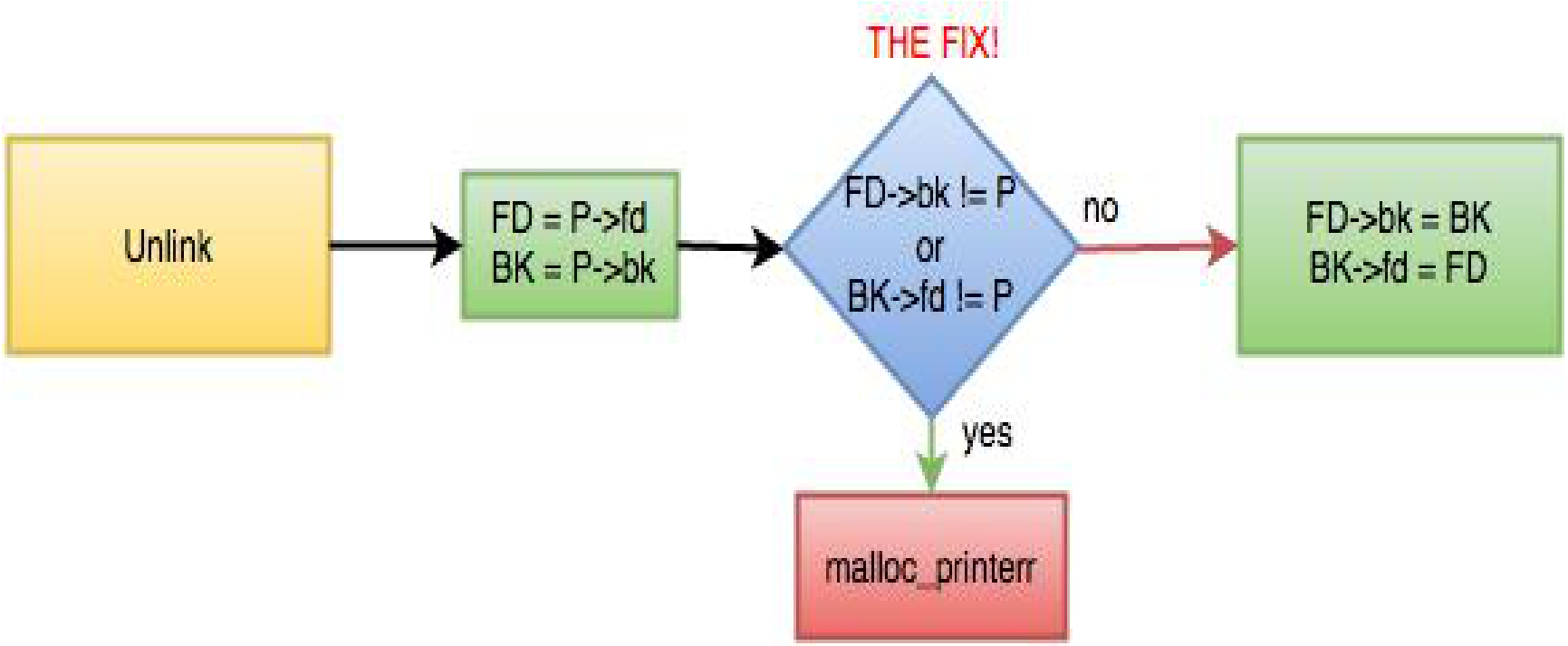



High level description of malloc subroutine

## The unlink macro:

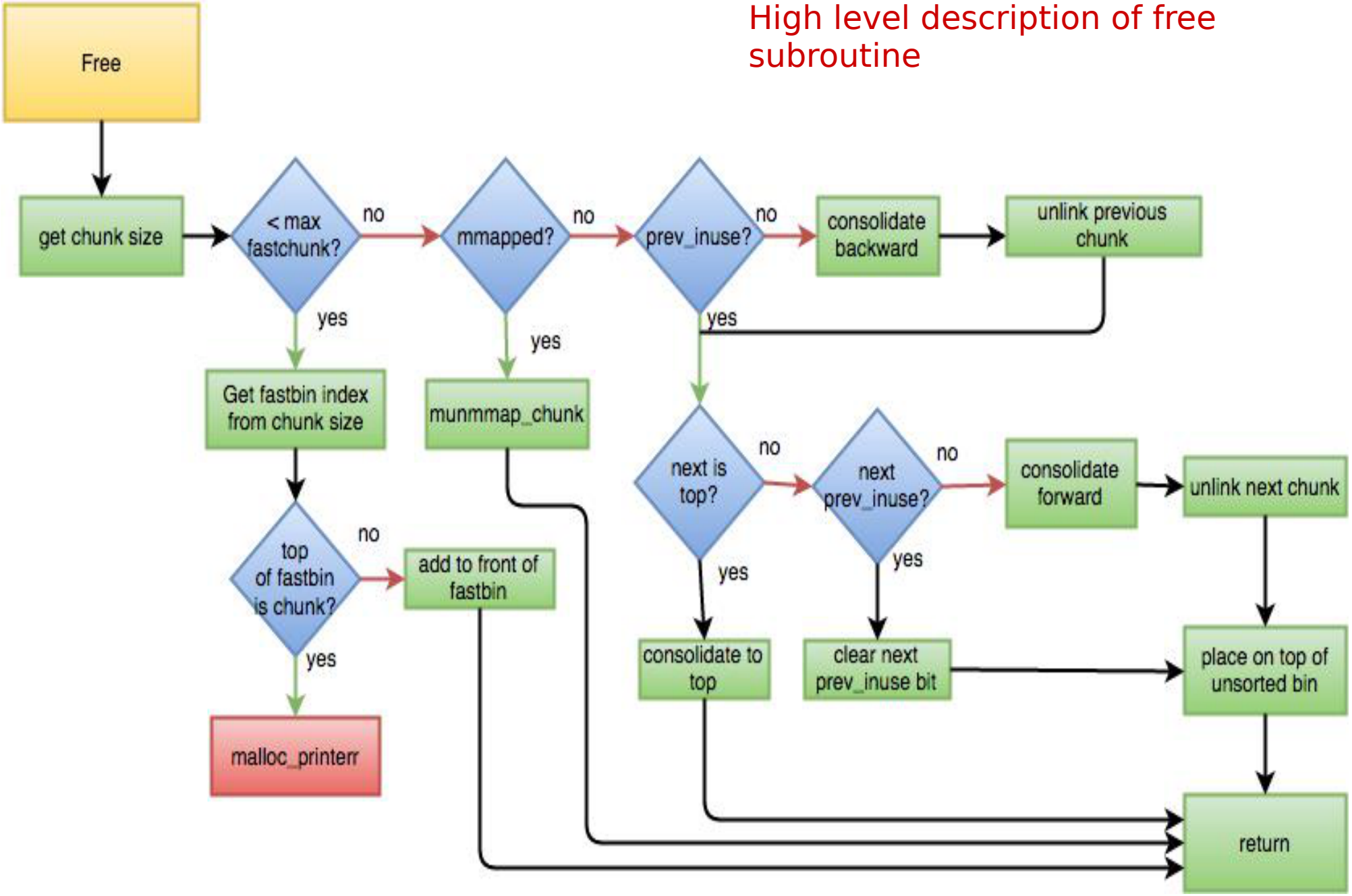
- Intended to take a chunk off a free list.
- Highly exploited in previous version of glibc malloc.
- Fixed (not really).

# Introduction to glibc malloc algorithm

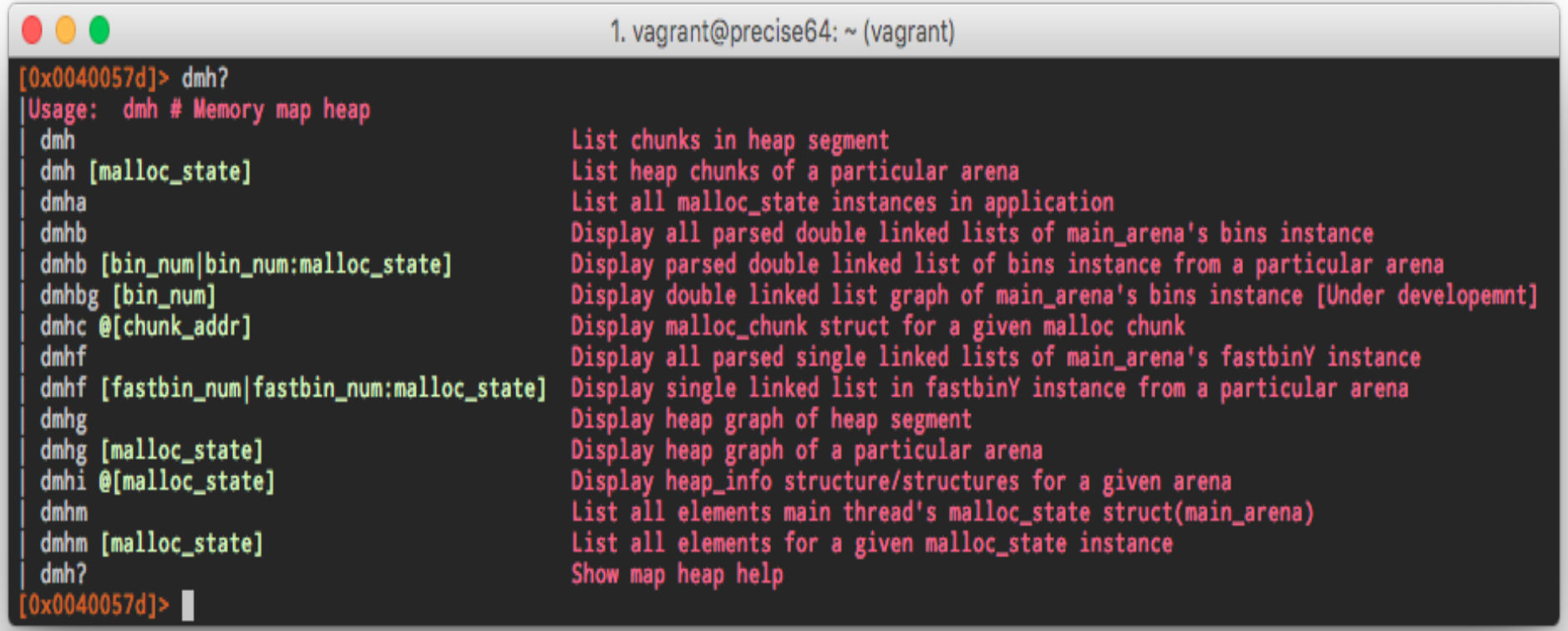



High level description of unlink macro

# High level description of free subroutine

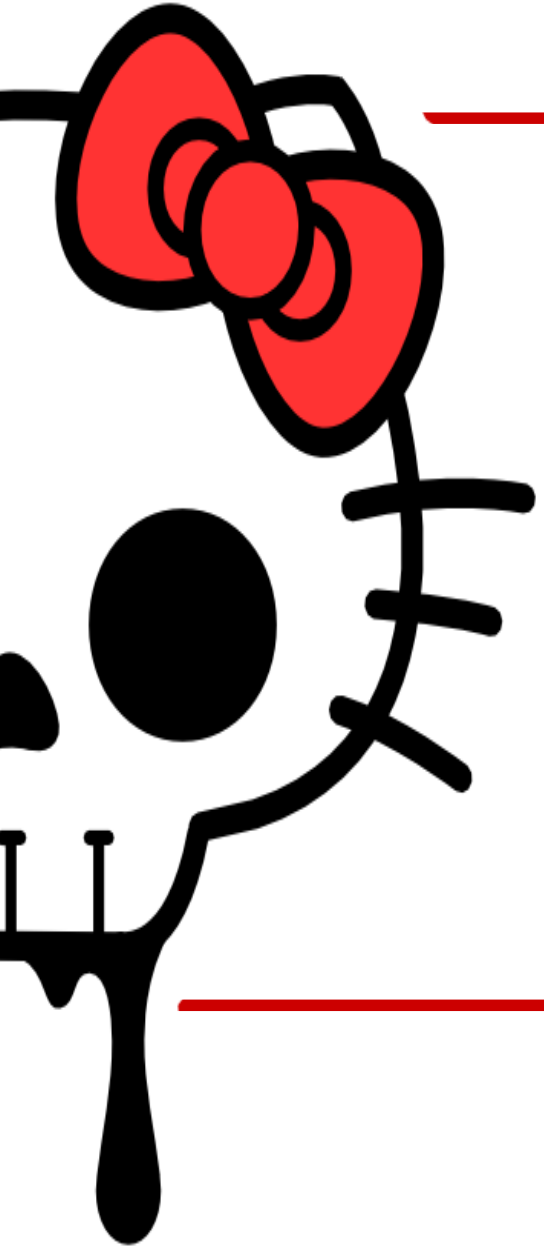


# Dmh command family



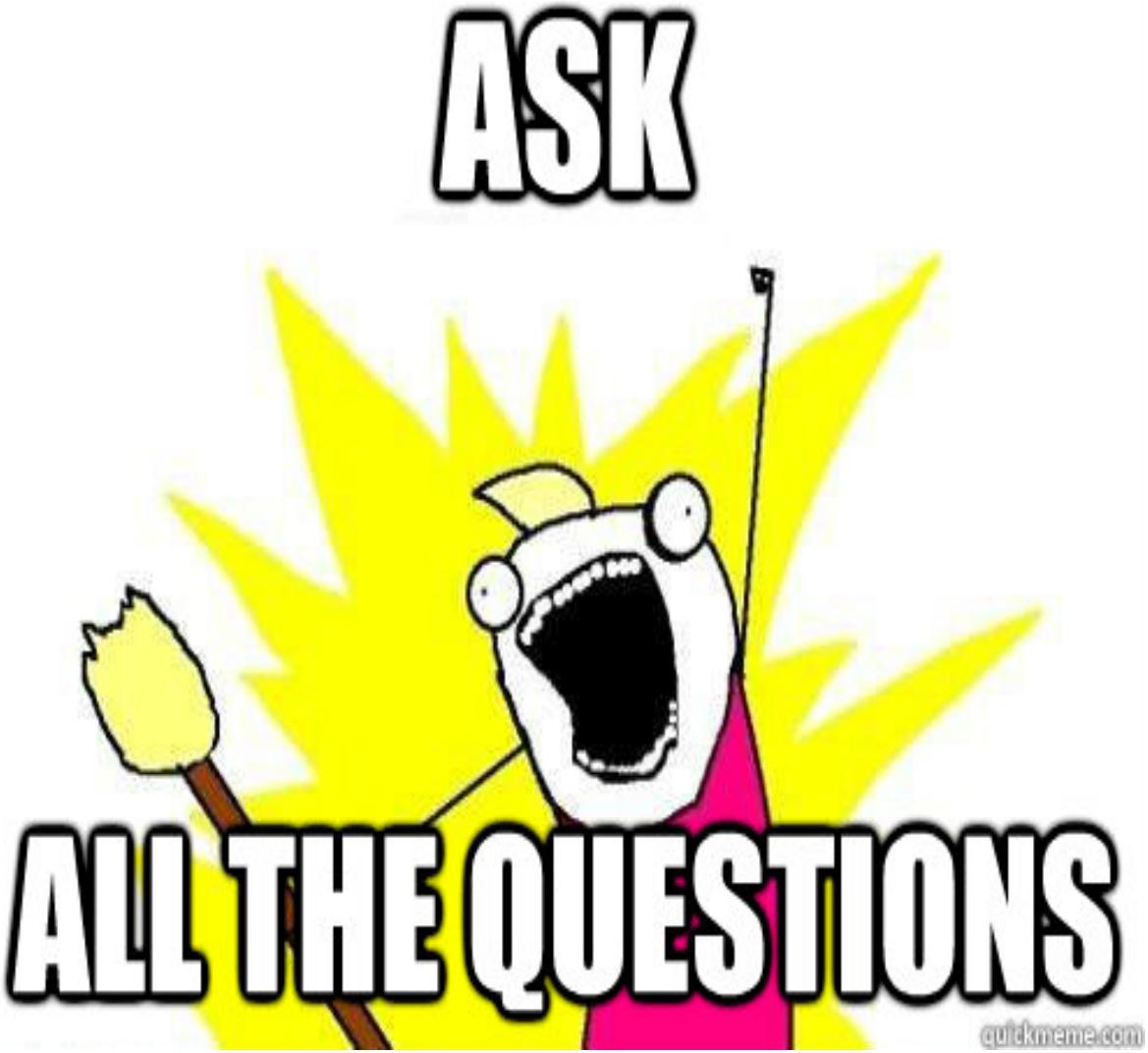
# Dmh command family

# Conclusion

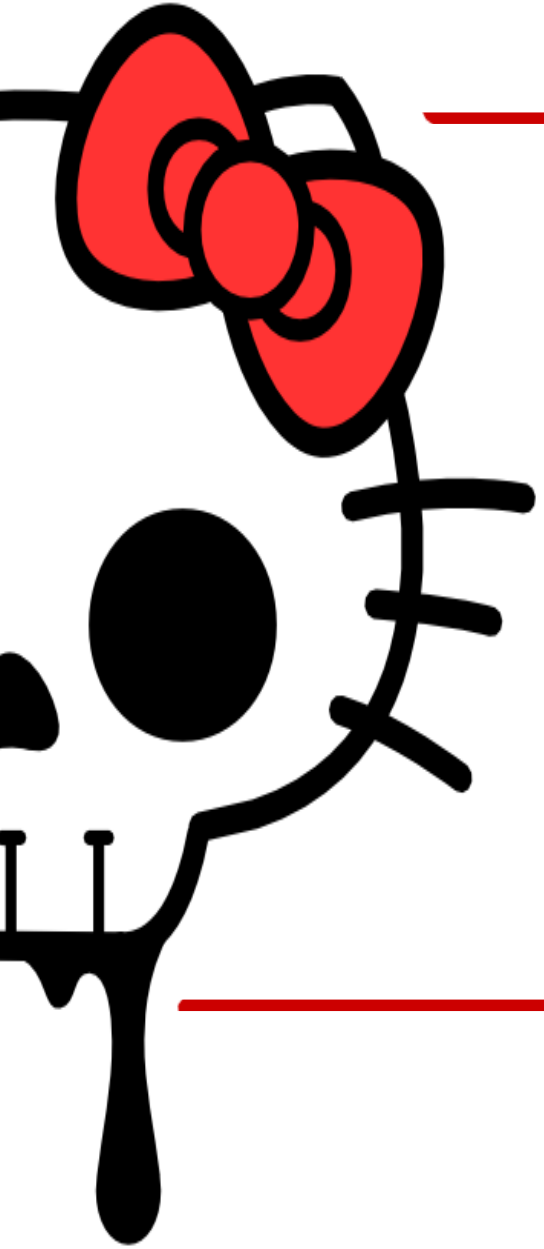


- The extension does give a more convenient view of the heap.
- Overall experience was satisfactory.
  - Learned to defend ourselves with Radare2 API
  - Learned in great extend how the glibc heap works.
- Extension has not reach its maximum potential.
  - Not all strucutres have been implemented yet (malloc_par).
- Planning to do similar extensions for different allocators such as:
  - Magazine.
  - Jemalloc.

# Questions?



# References



- **Glibc malloc source code:**
  - https://fossies.org/dox/glibc-2.24/malloc_8c_source.html.
- **Relevant documentation:**
  - https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/
  - https://sourceware.org/glibc/wiki/MallocInternals
  - http://files.cnblogs.com/files/hseagle/demo.pdf
  - http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/17-allocation-basic.pdf