

OS 2018

Homework 4: Memory Allocator Implementation

(Due date 01/03 23:59:59)

Objective

- Understand how **malloc()** and **free()** work
- Understand how to **manage Heap**

Requirements

Requirements

1. Implement a **Memory Allocator Library** for user application (*slides 4-16*)
2. Write a **User Application** to test the memory allocator library (*slides 17,18*)
3. Please follow the **Input/output format** (*slides 19,20*)

1. Memory Allocator Library Requirements

- The library must provide the following **3 functions**, and you should follow the format in *next slide* :
 1. **hw_malloc()** (*slides 6-13*)
 2. **hw_free()** (*slides 14-16*)
 3. **hw_get_start_brk()**
- Use **chunk** (*slides 7,8*), **bin** (*slides 9,10*) and **sbrk()** to **manage heap**
- Use **chunk** (*slides 7,8*) to **manage every mmap-allocated memory**

Functions format

1. `void *hw_malloc(size_t bytes)`

- bytes : the required memory size in bytes
- $\text{Return} = \begin{cases} \text{the valid virtual address} \\ \text{(starting address of the data part)} & , \text{ if success} \\ \text{NULL} & , \text{ otherwise} \end{cases}$

2. `int hw_free(void *mem)`

- mem : starting address of the data part
- $\text{Return} = \begin{cases} 1 & , \text{ if success} \\ 0 & , \text{ otherwise} \end{cases}$

3. `void *hw_get_start_brk()`

- Return the starting address of the heap

hw_malloc() requirements

- Use **mmap_threshold** to decide the memory allocate method. If **the allocated size(data size + chunk header size) > mmap_threshold** , use mmap allocation method ; else use Heap allocation method.
 - **mmap_threshold** is **initial : 32 KiB** (32 * 1024)

mmap allocation method :

- Use **mmap()** system call to allocate the space
- Use **chunk** (slides 7,8) to manage the allocated space

The allocate size = request size (data size) + chunk header size

- Use **mmap_alloc_list** (slide 11) to manage allocated mmap chunks

Heap allocation method:

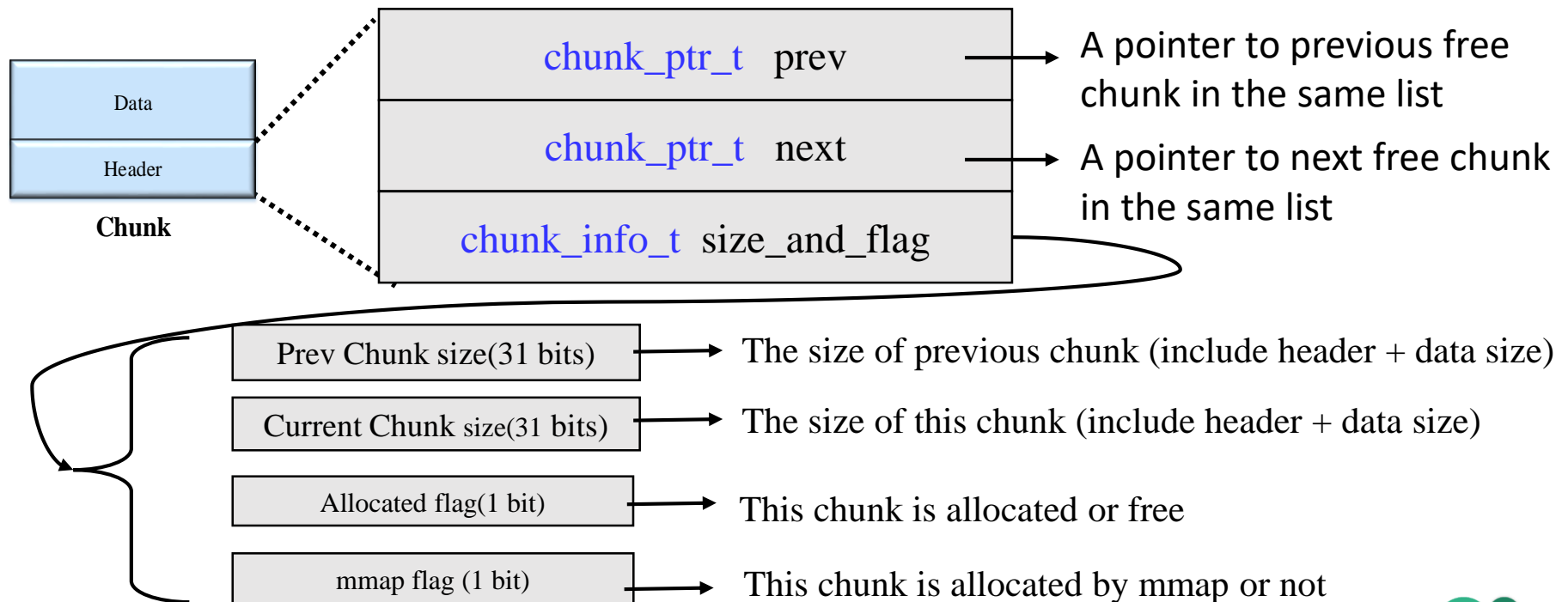
- Use **chunk** (slides 7,8) and **bin** (slides 9,10) to manage heap
- Should follow the rules of **Heap initialization** (slide 12)
- Should follow the rules of **Split** (slide 13)
- **The allocated size (data size + chunk header size) should be the best fit size**

Chunk requirements

- The continuous heap space is split into chunk(s) for management
- Each chunk contains two parts, header and data (*in next slide*)
 - Header (lower address)
 - Data (higher address), the actual memory space return to caller

Chunk header format

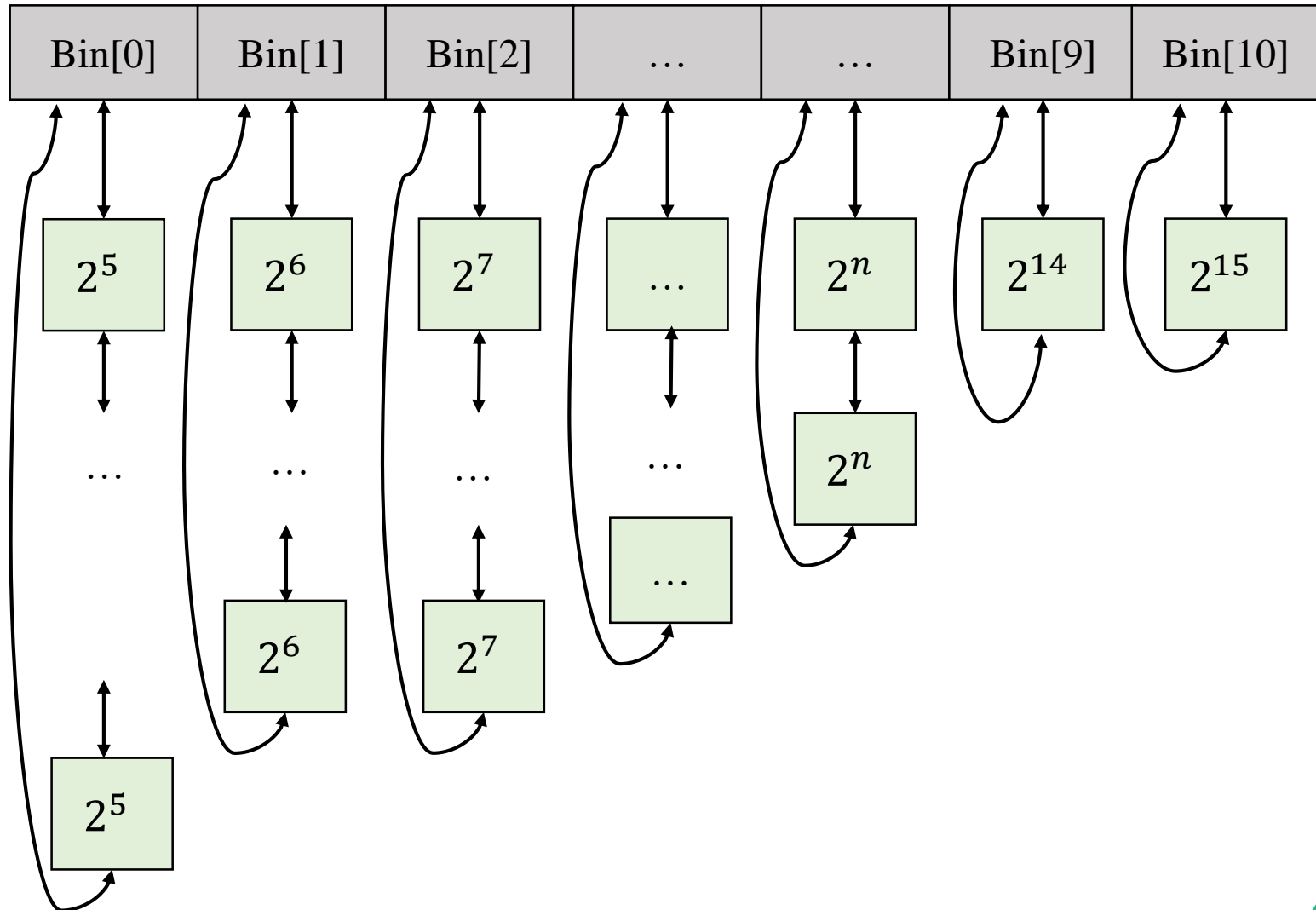
- Chunk header (24 bytes)
 - There are 3 members in the header
 - `chunk_ptr_t` and `chunk_info_t` can be defined by yourself, but each of them should be 8 bytes
 - `chunk_info_t` should include 4 information (Prev Chunk size, Current Chunk size, Allocates flag and mmap flag)



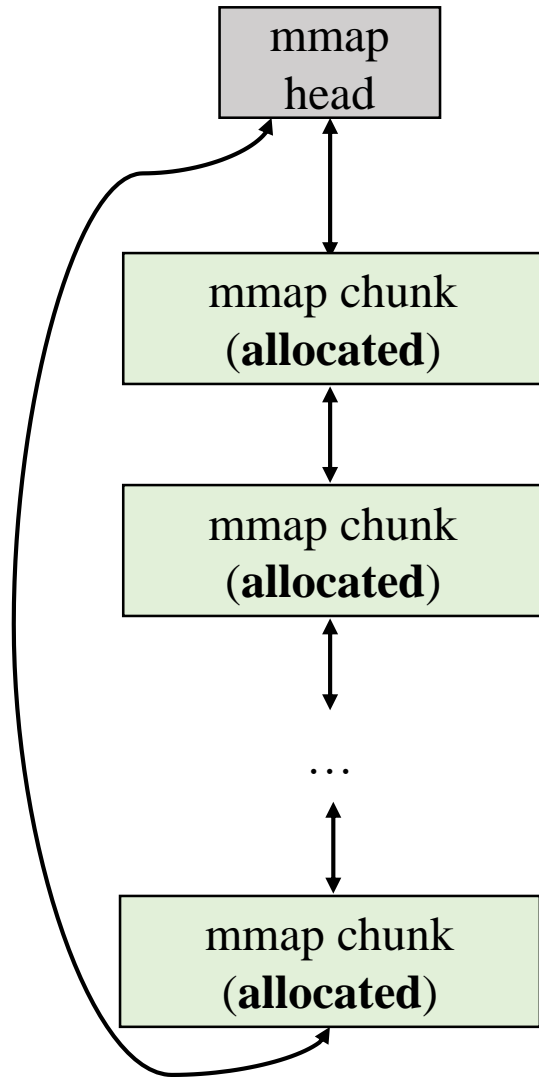
Bins requirements

- Bins are use to manage free chunks of Heap's segment.
- A bin is a circular doubly-linked list of free chunk(s) (*next slide*)
- Add the chunk to **rear** of the bin.
- You should manage 11 bins
 - bin[0]-bin[10] hold chunks with fixed size (*next slide*)
 - Every chunk size should be **the nth power of 2** (n is a number of 5 to 15)
- Use the best fit size to select a chunk during memory allocation
 - If there are multiple chunks with the same size, **select the one with the lowest address**

Bin example



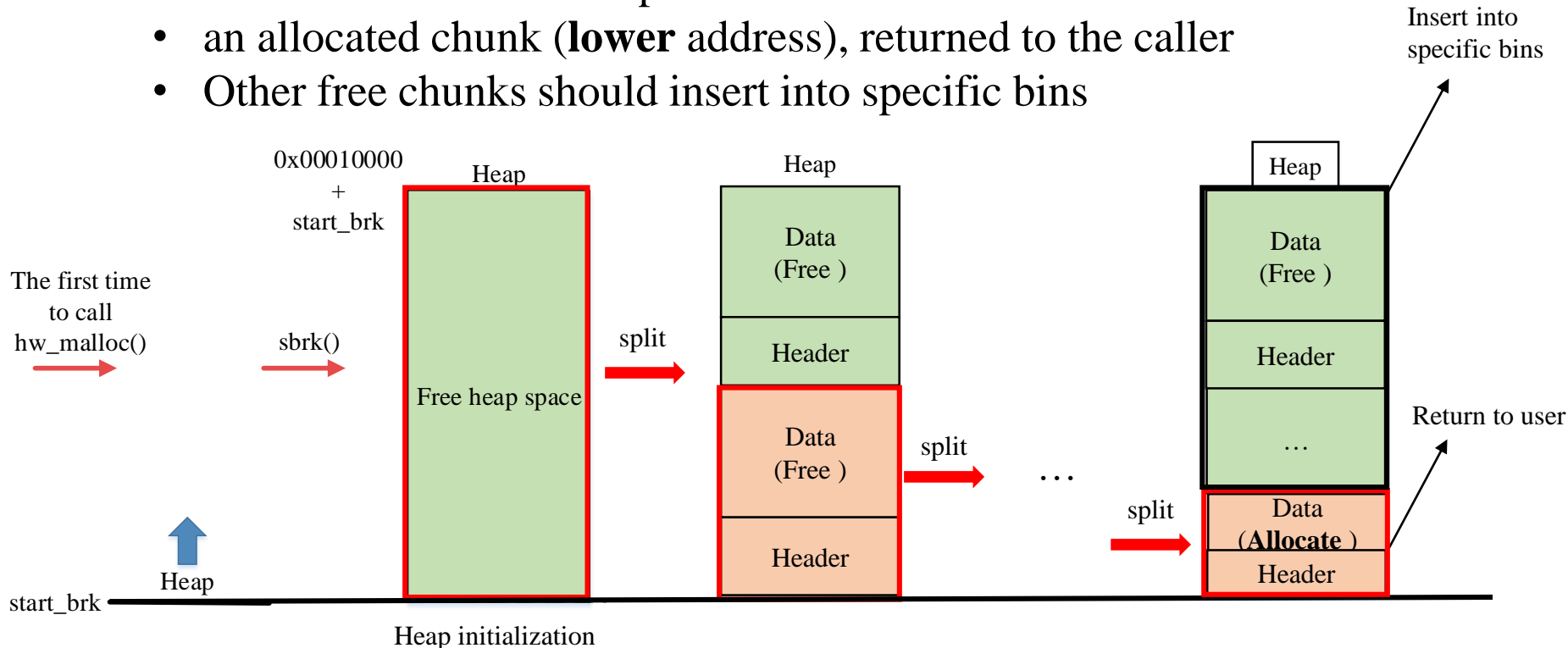
mmap_alloc_list



- `mmap_alloc_list` is a circular doubly-linked list of mmap allocated chunk(s)
- mmap chunk header is same as [slide 8](#) (use `chunk_ptr_t`)
- `mmap_alloc_list` should be ordered by size (ascending)
- If there is/are multiple chunk(s) of the same size, add new chunk **after** it/them.

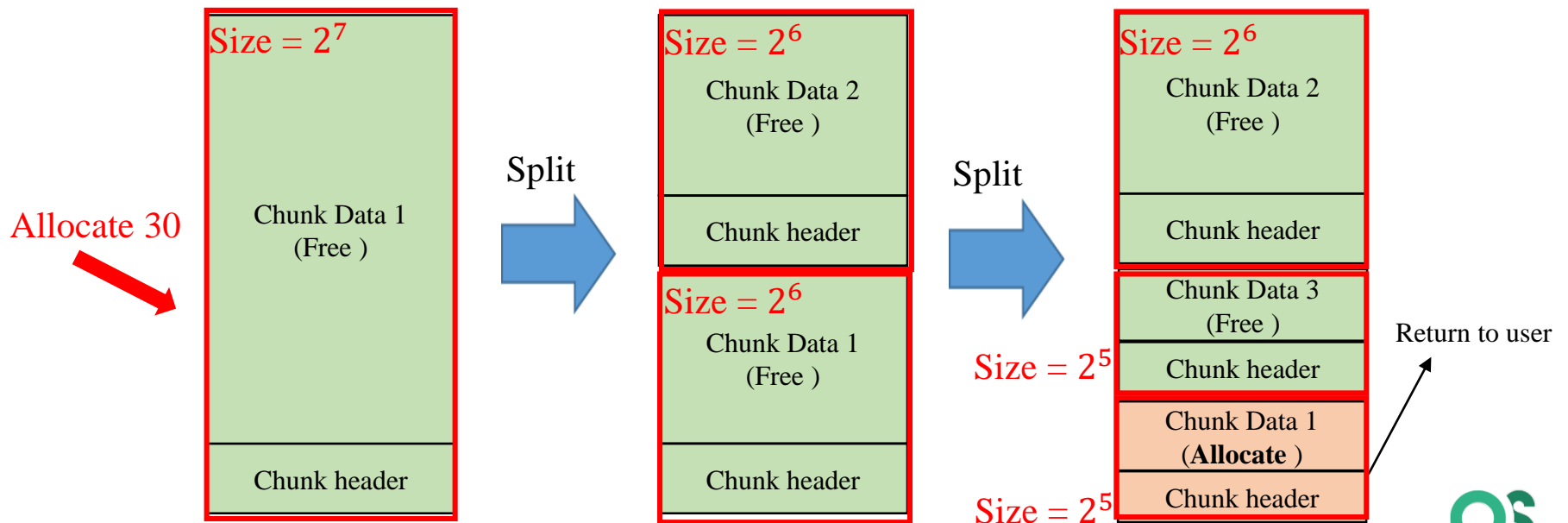
Heap initialization & First-time Heap allocation

- Initialize the Heap before its first use :
 - Use `sbrk()` to allocate a **64KiB** heap
- After the heap initialization, Split (*next slide*) until the allocated size match the best fit of the nth power of 2
 - an allocated chunk (**lower** address), returned to the caller
 - Other free chunks should insert into specific bins



Split

- When Heap allocation method is used, *split* may be performed :
 - If chunk size is too large (not the best fit for the allocation size), they must be split into two equal-sized chunks
 - Should split the lowest address chunk
 - Chunk size should always be the nth power of the 2
 - Must be **split until it reach the best fit for the allocation size**
- Example : (hw_malloc(6), allocation size =30 , Chunk1 =2⁷)



hw_free() requirements

- Use Chunk header information (*slides 7, 8*) to check the address was allocated by mmap or Heap allocation method.

mmap free method :

- Use **munmap()** system call to free the space
- **The chunk header should be free too.**
- This free chunk **do not need to be added into bin.**
(bin is not used in mmap allocation method)

Heap free method :

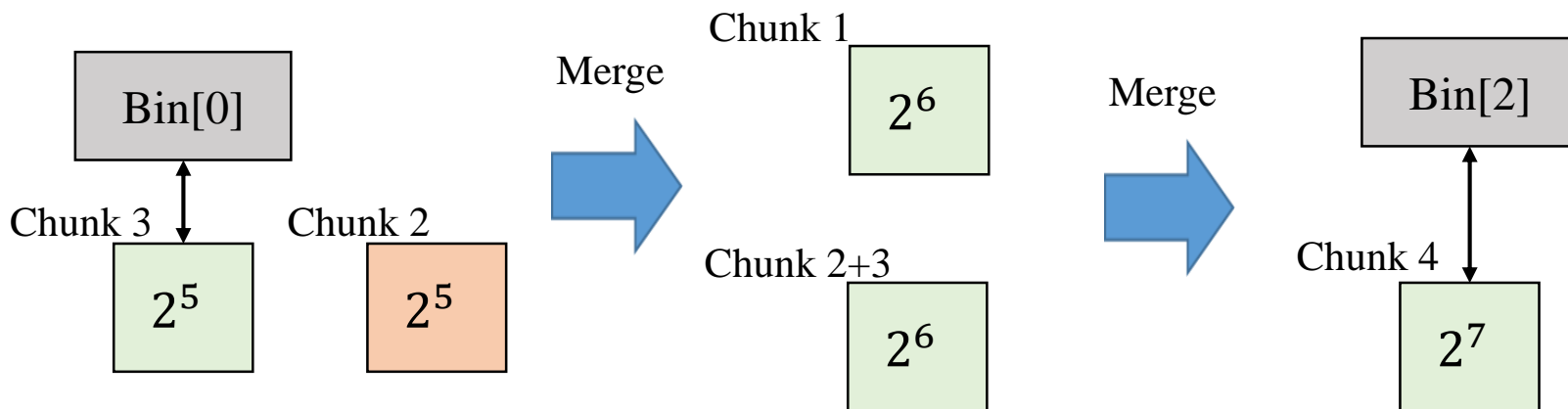
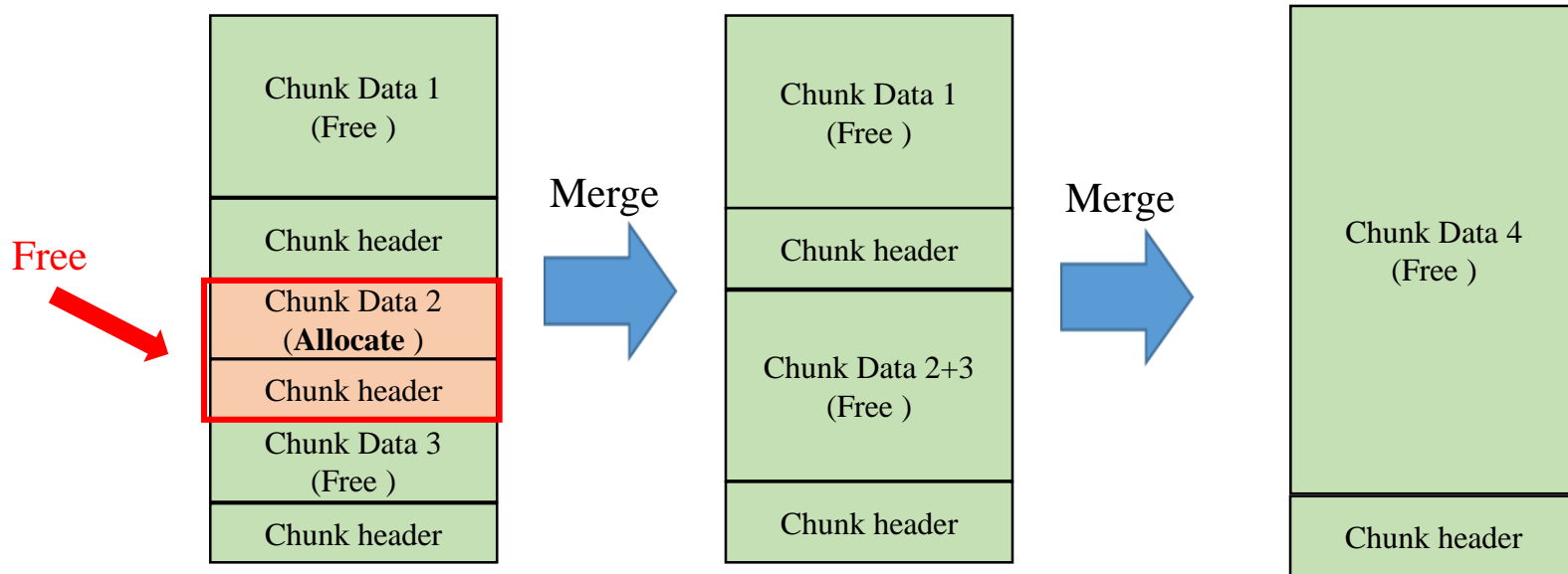
- Use bin (*slides 9,10*) to manage free chunks
- Should follow the rules of **Merge** (*slides 15,16*)

Merge

- When Heap free method is called , adjacent free chunks must be merged into one if they have the same size :
 - **Merge operations should be repeated until the two adjacent free chunks do not have the same size**

Merge Example

- Example: (Chunk 1 size = 2^6 , Chunk 2 size = Chunk 3 size = 2^5)



2. User Application Requirements

Write a user application to test the memory allocator library

- Should receive **4 kinds of commands** :
 1. **alloc** *N*
 2. **free** *ADDR*
 3. **print** *BIN*
 4. **print** *mmap_alloc_list*
- Continuously receive commands from stdin until **EOF** (*Ctrl+D*)
 - Should successfully run
“cat testfile.txt | ./hw4_mm_test > outputfile.txt”

Commands format

1. **alloc N**

- Call `hw_malloc(N)` to allocate N bytes of data memory
- Print relative data address (i.e., offset between `start_brk` and the address returned by `hw_malloc()`)

2. **free ADDR**

- Call `hw_free()` to free the memory at (`start_brk + ADDR`)
- Print either “success” or “fail”

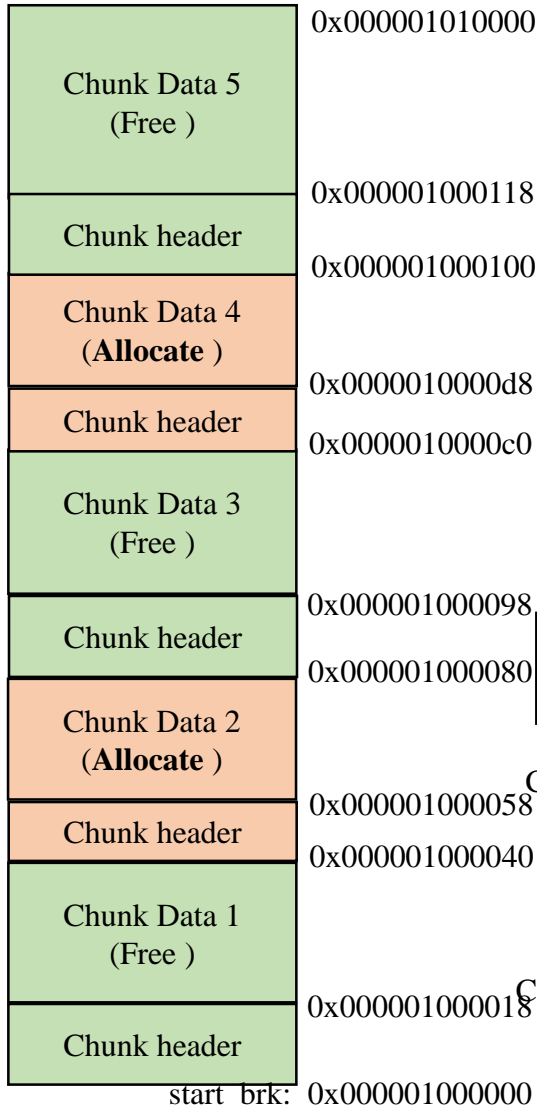
3. **print bin[i]**

- Print relative data address and size information of a given bin
 - `bin[i]` can be `bin[0]`, `bin[1]` `bin[10]`

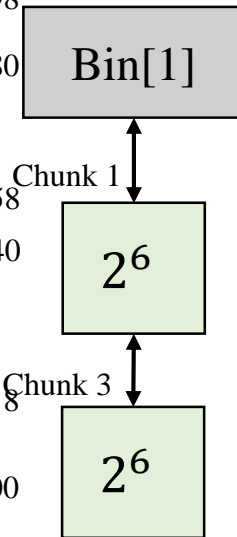
4. **print mmap_alloc_list**

- Print data address and size information.

3. Input/output format



- alloc
 - print (relative) **data** address in a line
- free
 - print success/fail in a line
- print bin[i]
 - print (relative) **chunk** address and **size** of each free chunk in the given bin, from the front to the rear
 - print a line for each chunk; pad 8 dash signs between the address and size



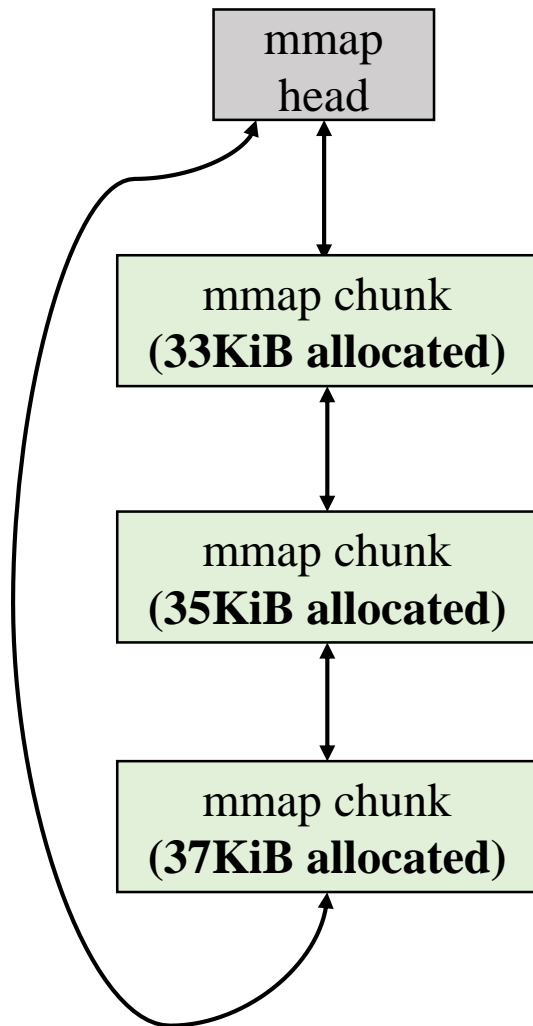
Input example

```
alloc 16
alloc 16
alloc 16
alloc 16
free 0x00000000000018
free 0x00000000000098
print bin[1]
```

Output example

```
0x00000000000018
0x00000000000058
0x00000000000098
0x000000000000d8
success
success
0x00000000000000-----64
0x00000000000080-----64
```

3. Input/output format



- print `mmap_alloc_list`
 - print **chunk address** and **size** of each free chunk in the list, from the front to the rear
 - print a line for each chunk; pad 8 dash signs between the address and size

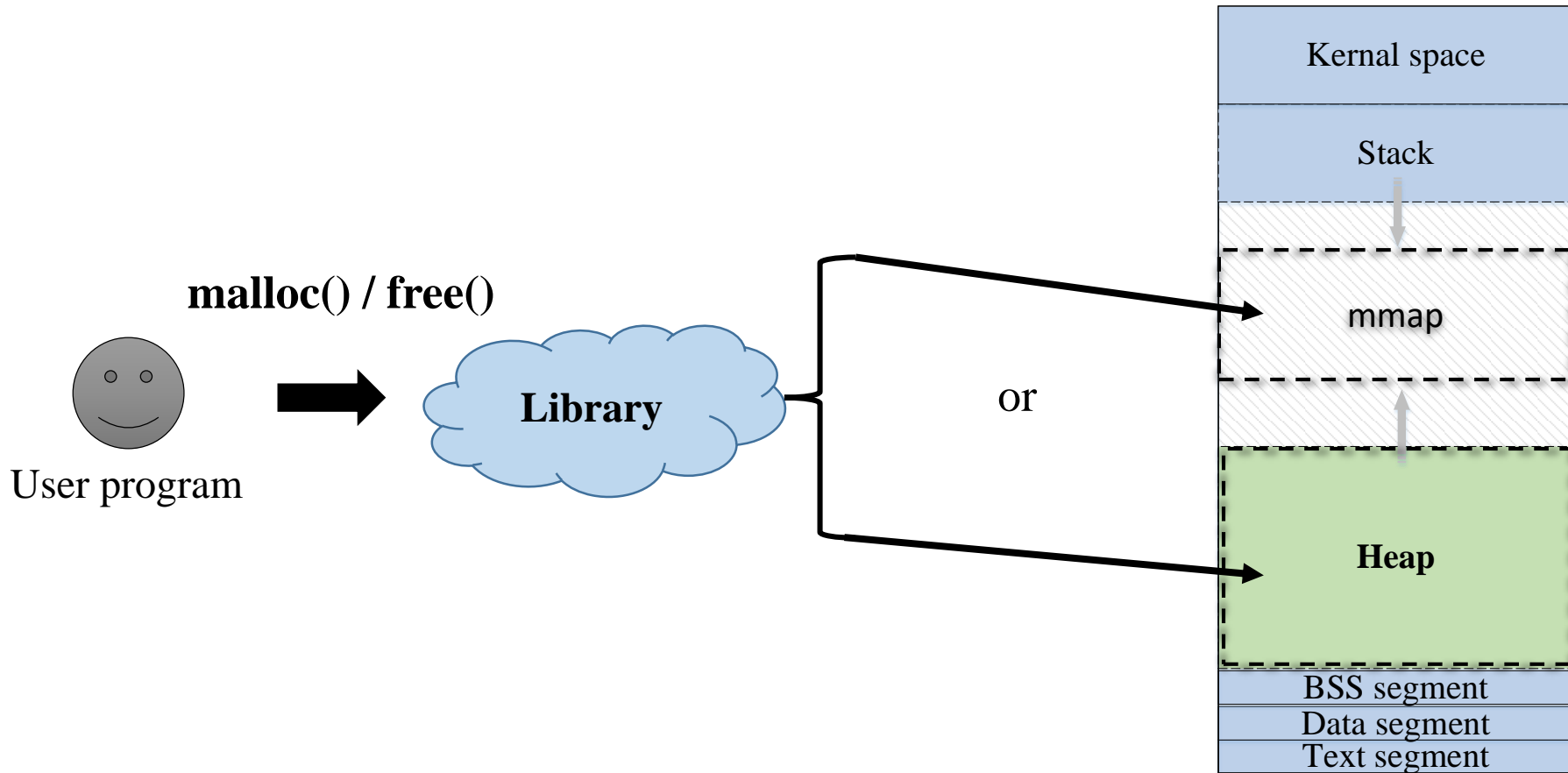
Input example

```
alloc 33768
alloc 37864
alloc 35816
print mmap_alloc_list
```

Output example

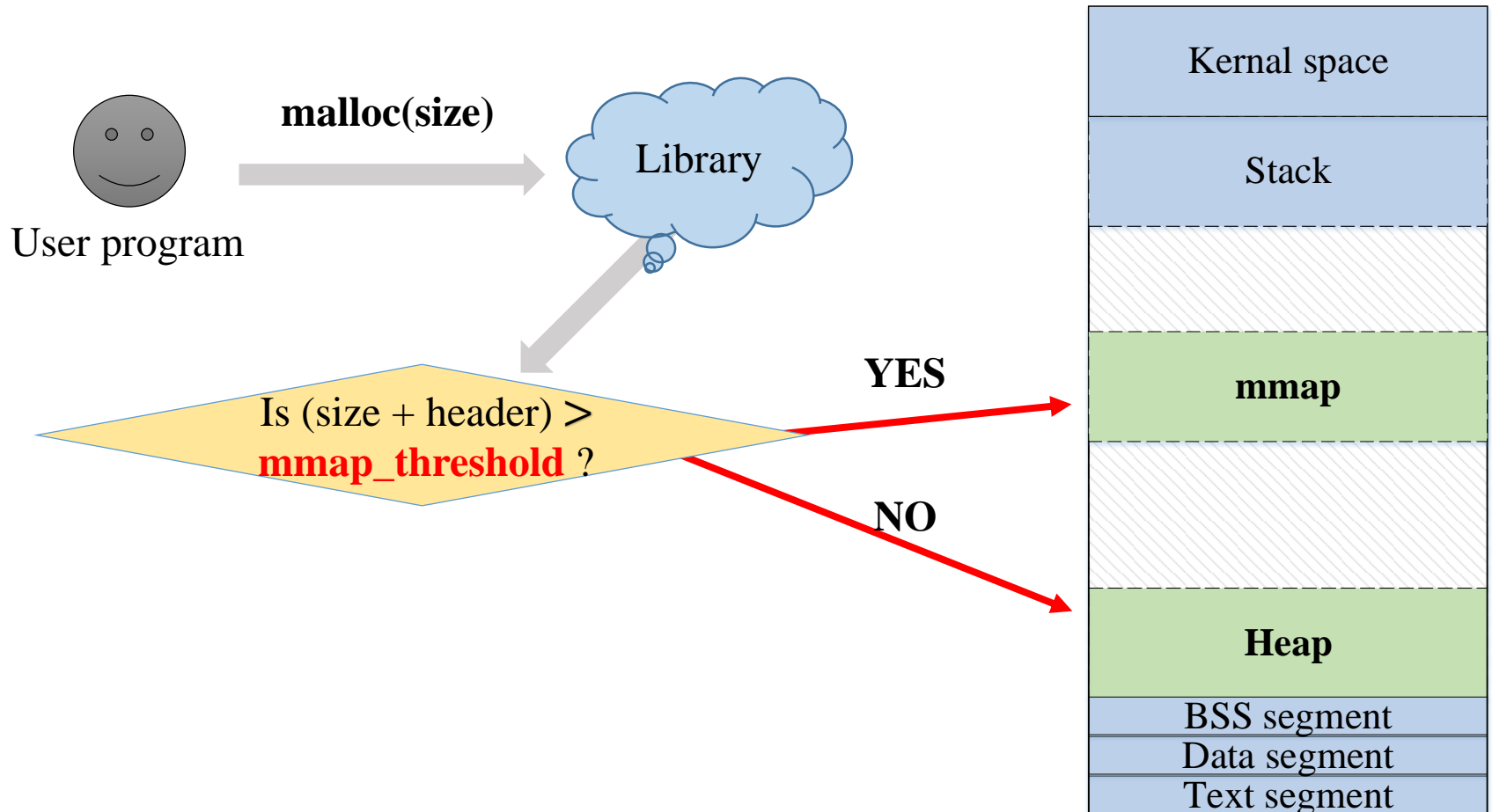
```
0x7f0cdcf00018
0x7f0cecff0f018
0x7f0bfcff0f018
0x7f0cdcf00000-----33792
0x7f0bfcff0f000-----35840
0x7f0cecff0f000-----37888
```

Concepts (malloc / free)

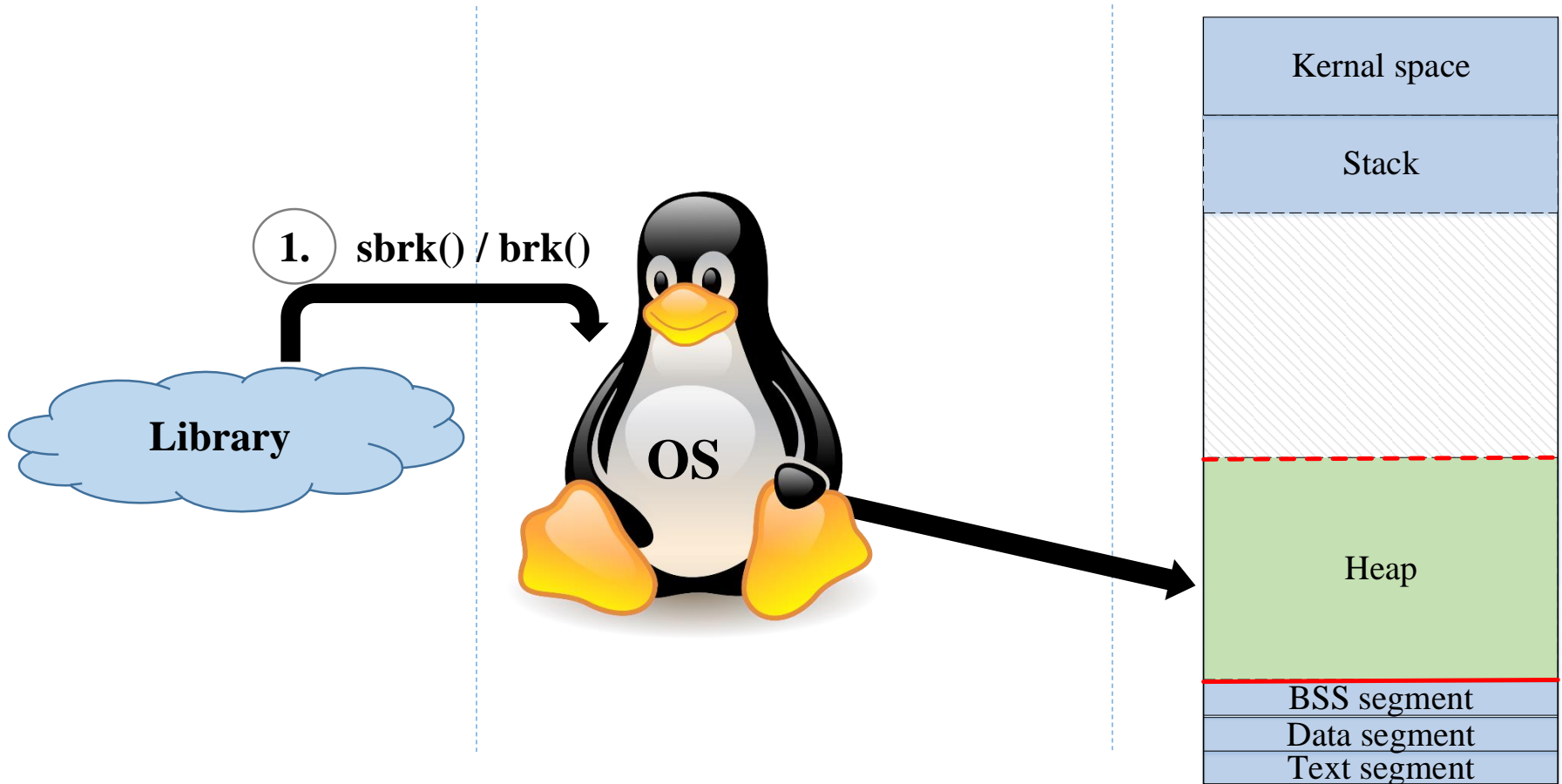


Concepts (mmap_threshold)

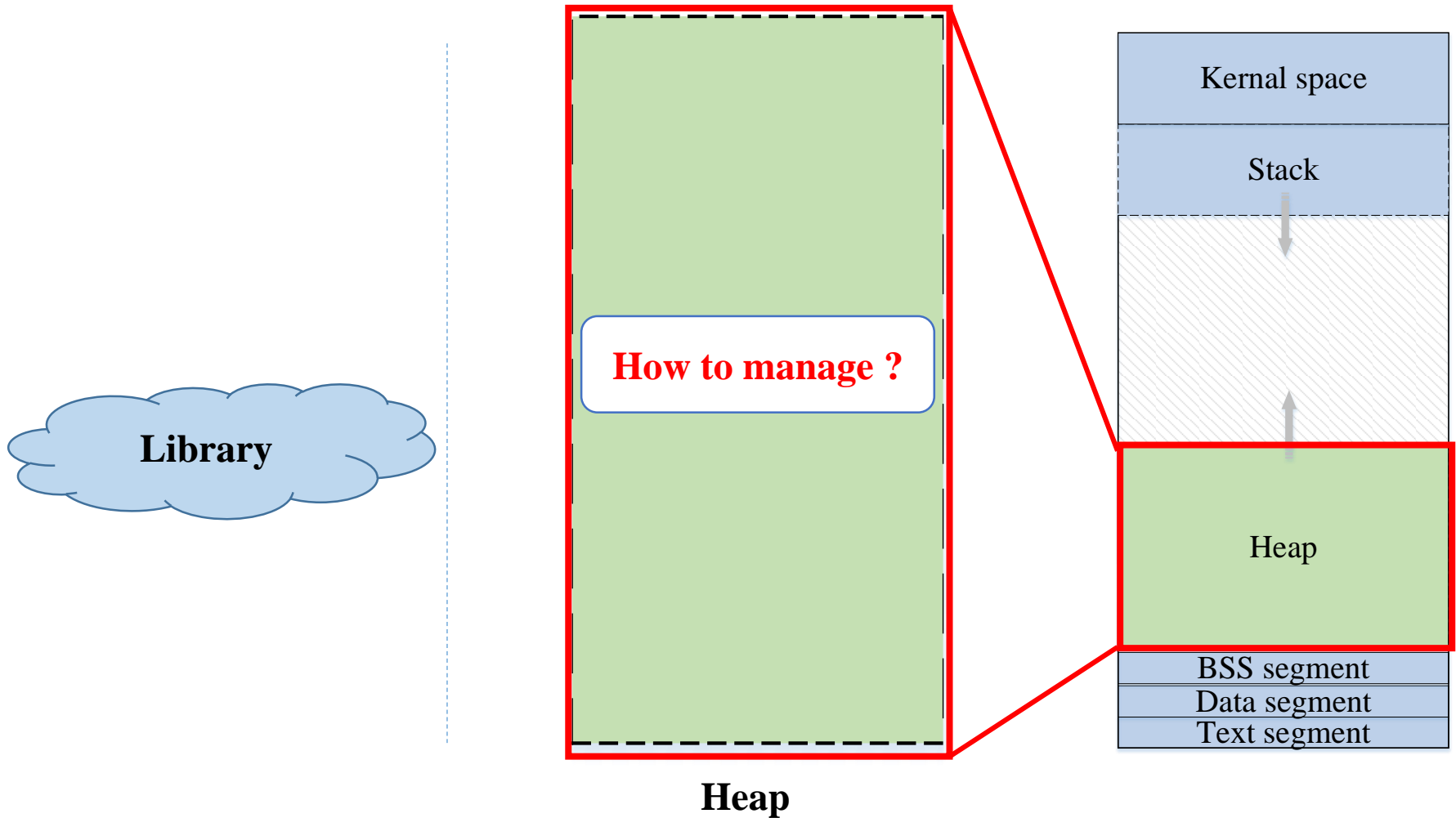
- Use `mmap_threshold` to decide the memory allocate method



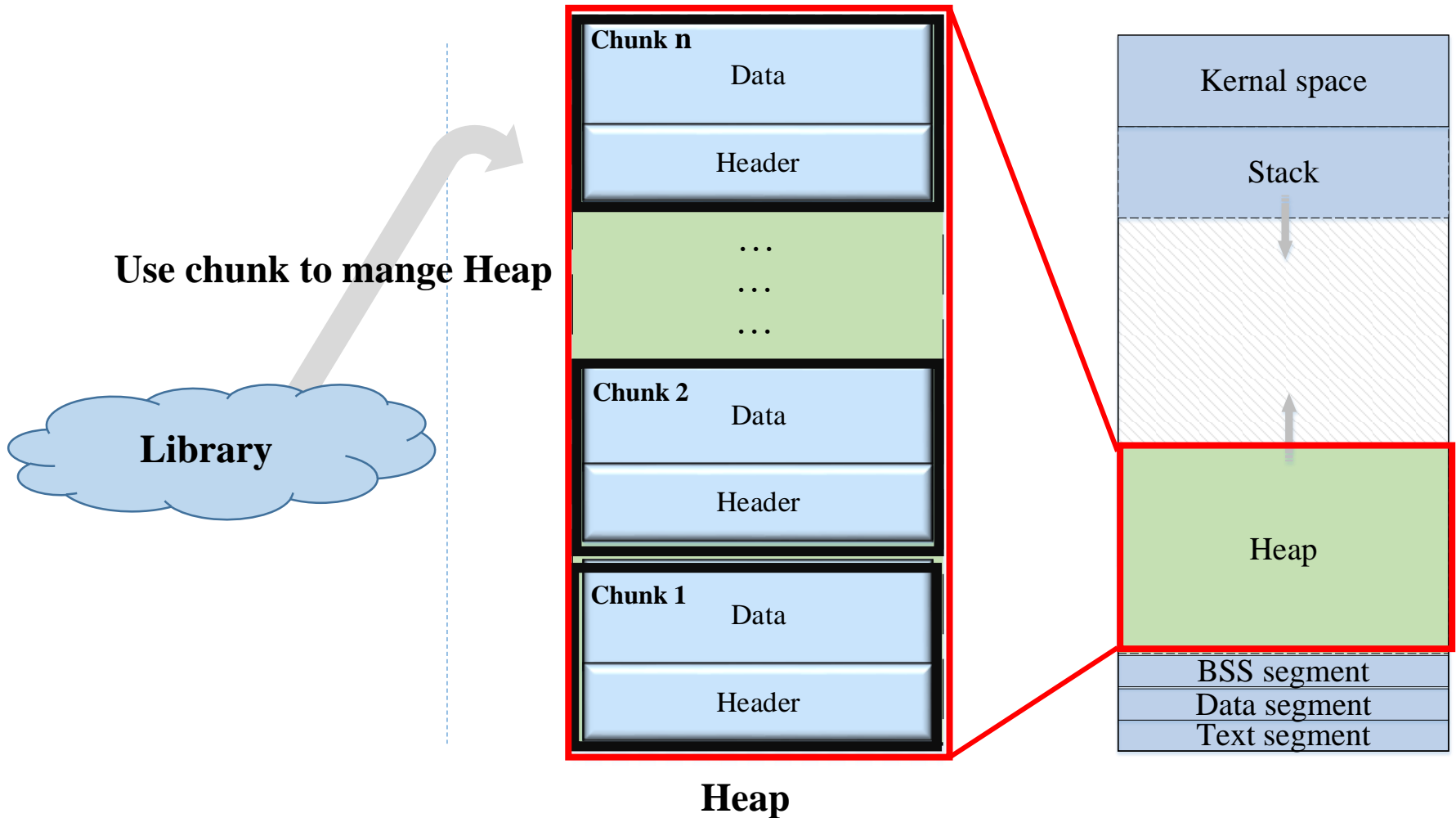
Concepts (Heap initialization)



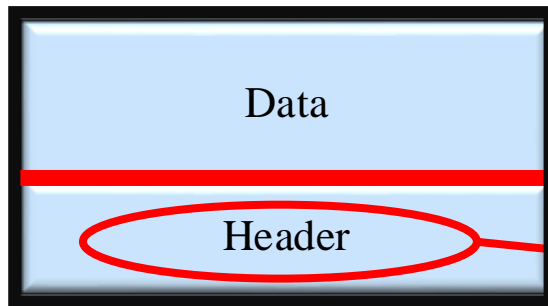
Concepts (heap)



Concepts (heap segment)



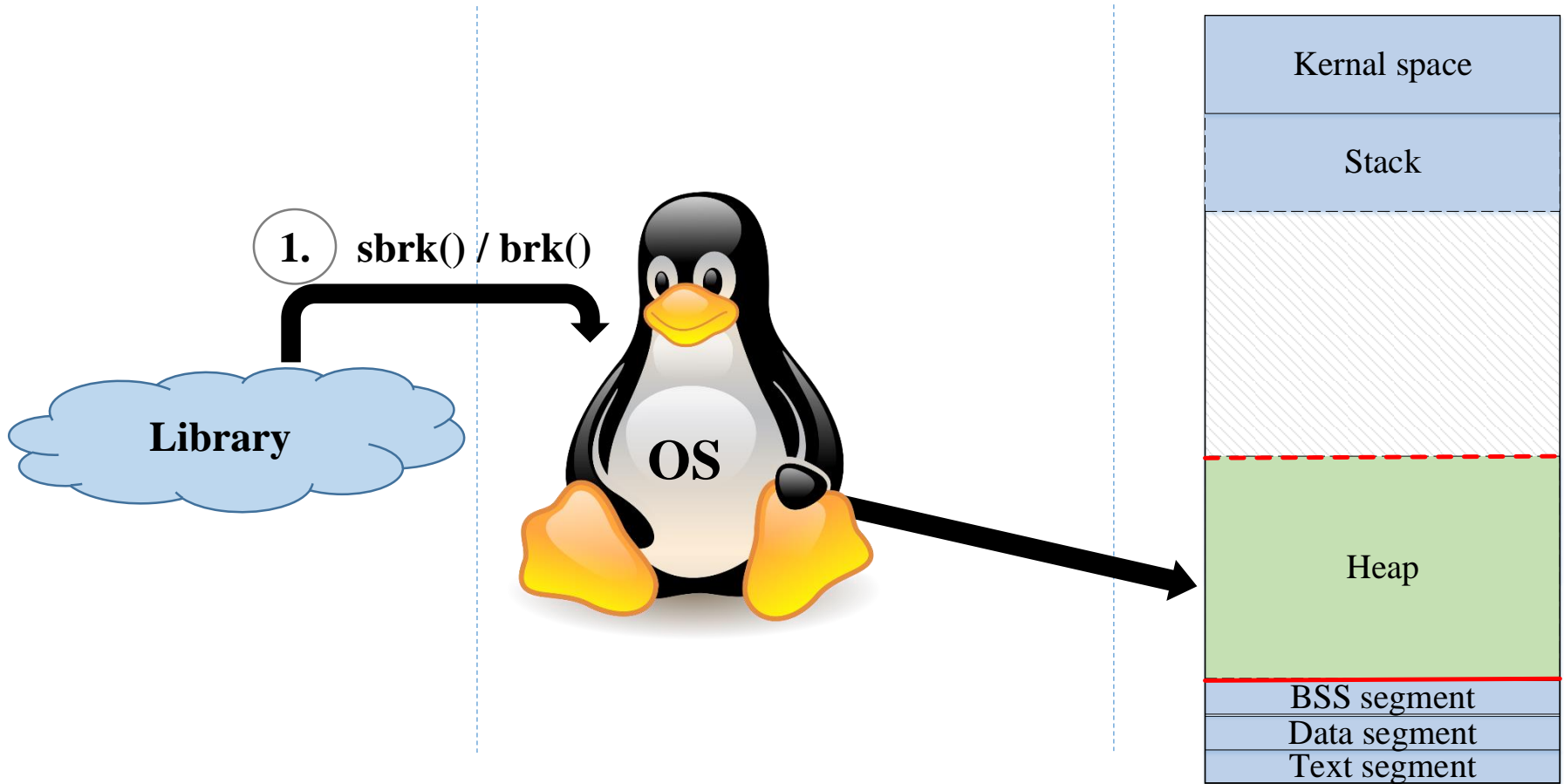
Concepts (chunk)



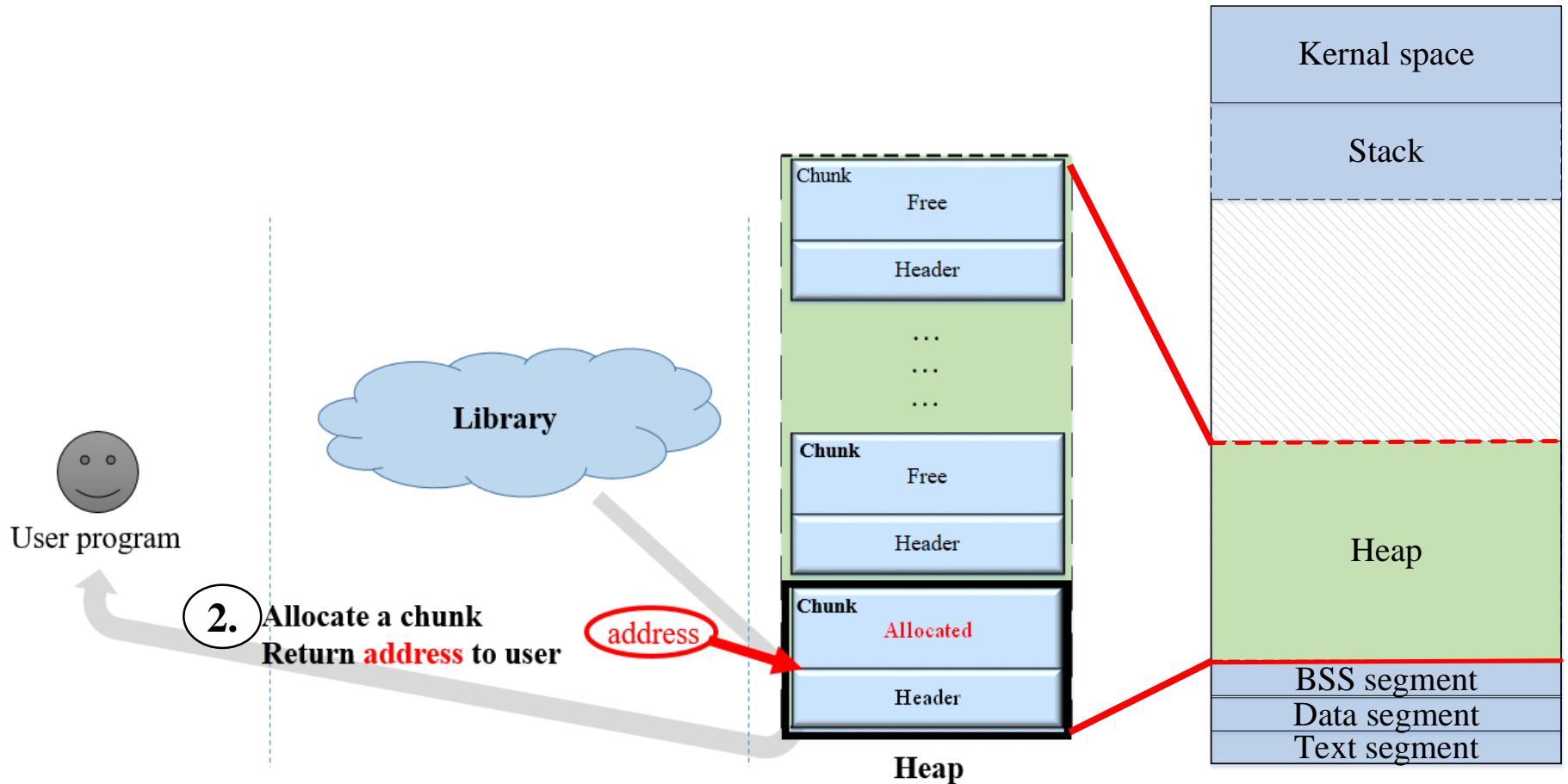
Chunk

- **Return address**
(starting address of the data part)
- **Chunk information**
(e.g: size, flag, prev, next...)

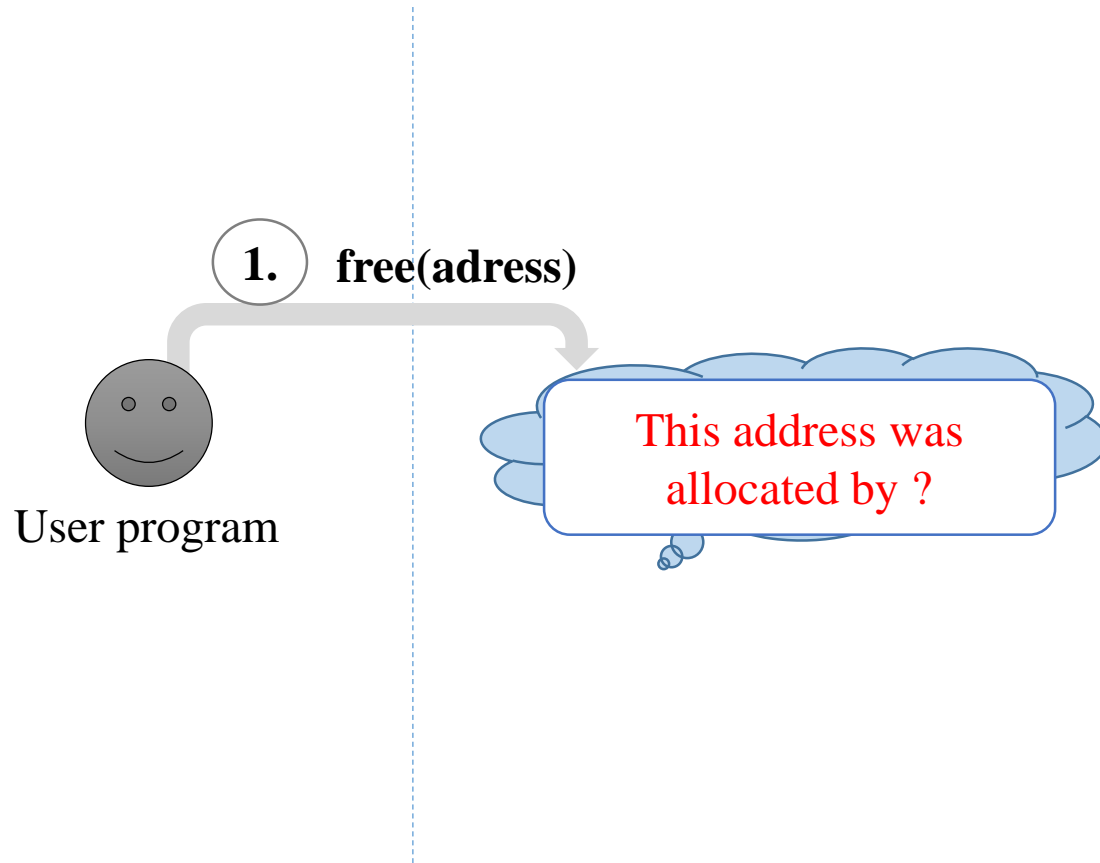
Concepts (Heap initialization)



Concepts (heap segment)

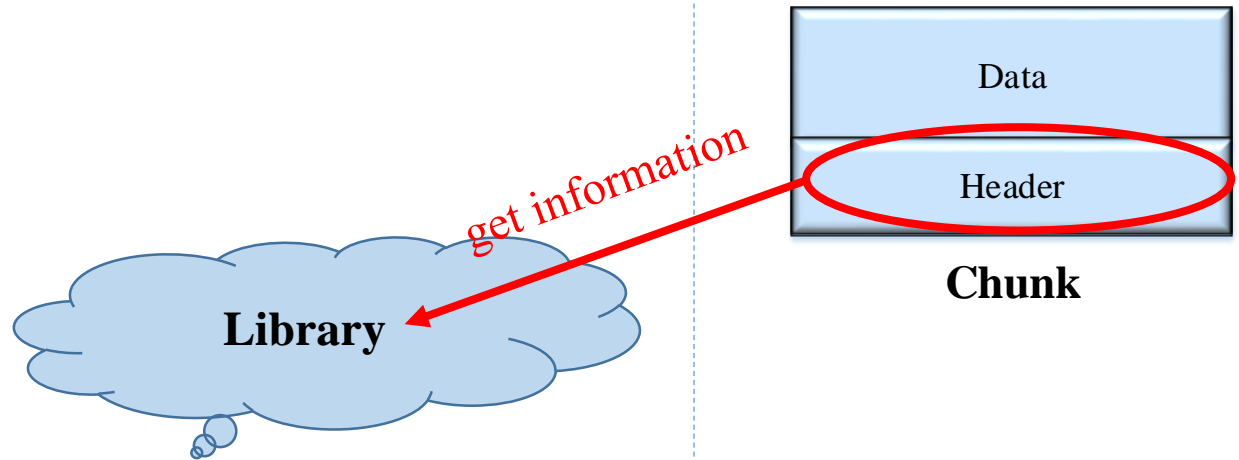
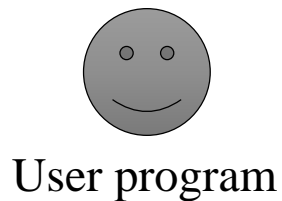


Concepts (free)

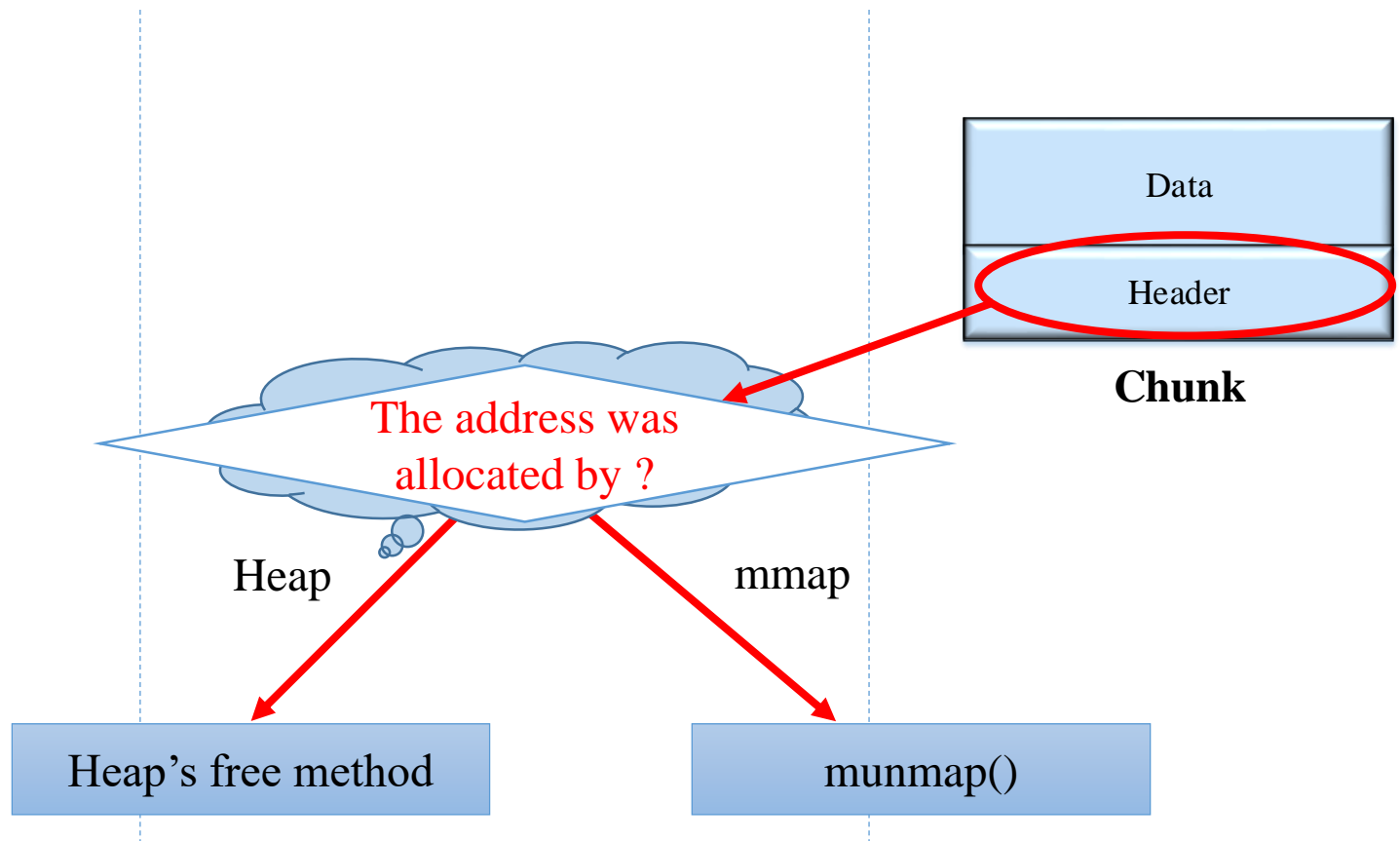


Concepts (free)

- Get information from Chunk header to know which memory allocate method was use

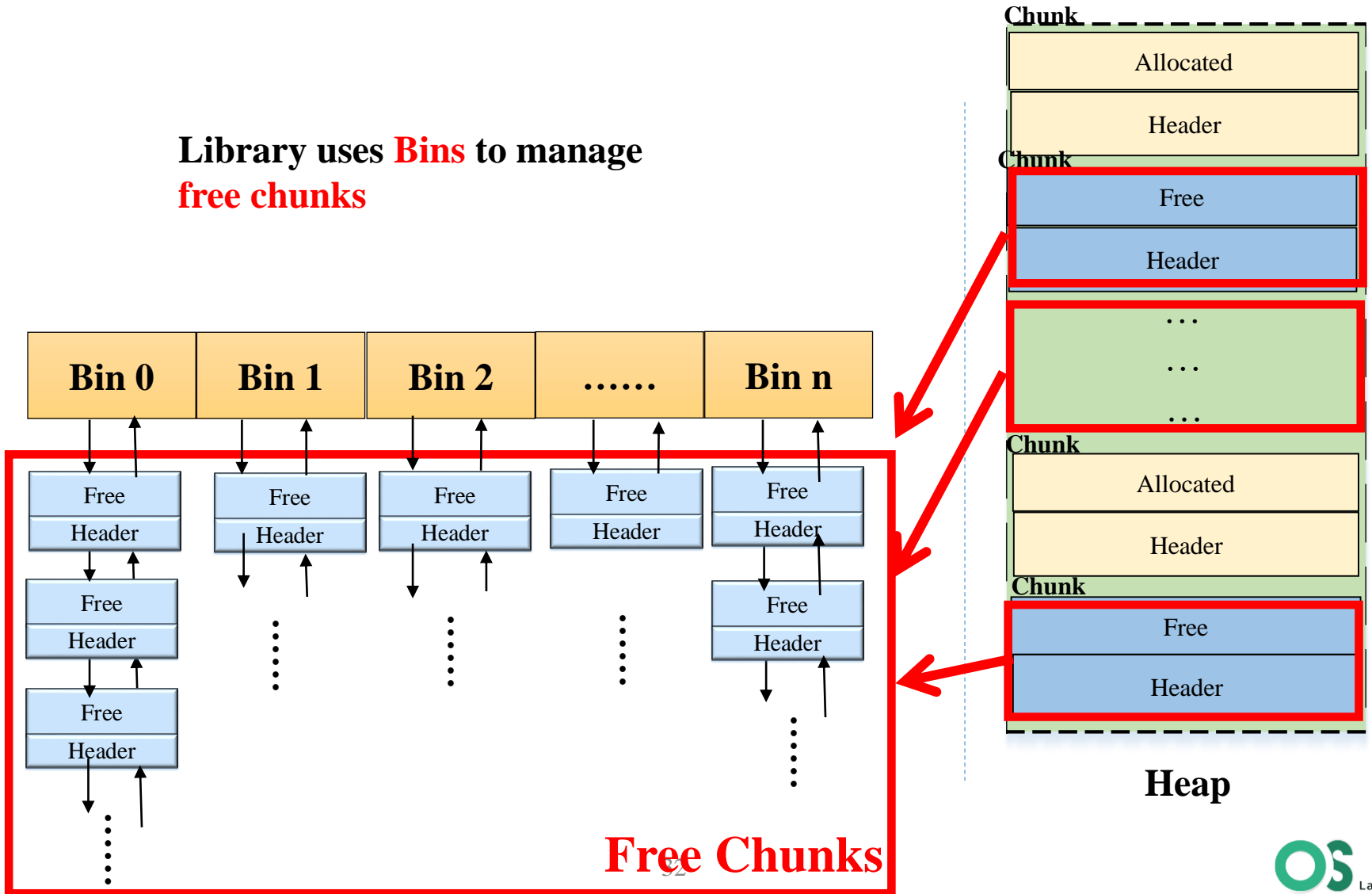


Concepts (free)

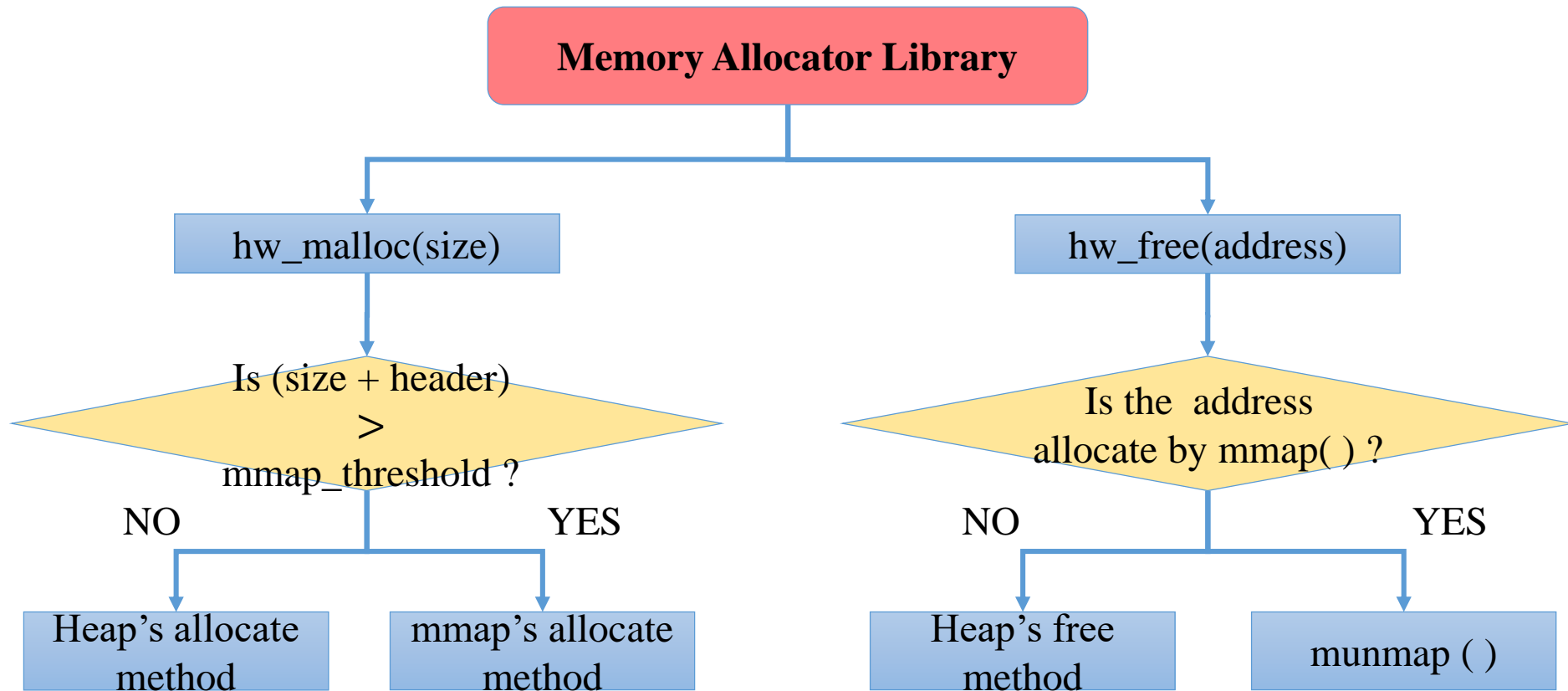


Concepts (Heap's free method)

Library uses **Bins** to manage
free chunks



Concepts (Architecture)



References

- sbrk()
 - [Linux man page](#)
- Streams, pipes, and redirects
 - [IBM](#)