OS 2017

Homework4: memory allocator implementation

(Due date 01/18 23:59:59)

Lab @NCKU

Objective

- Understand how to manage heap
- Understand how malloc() and free() work

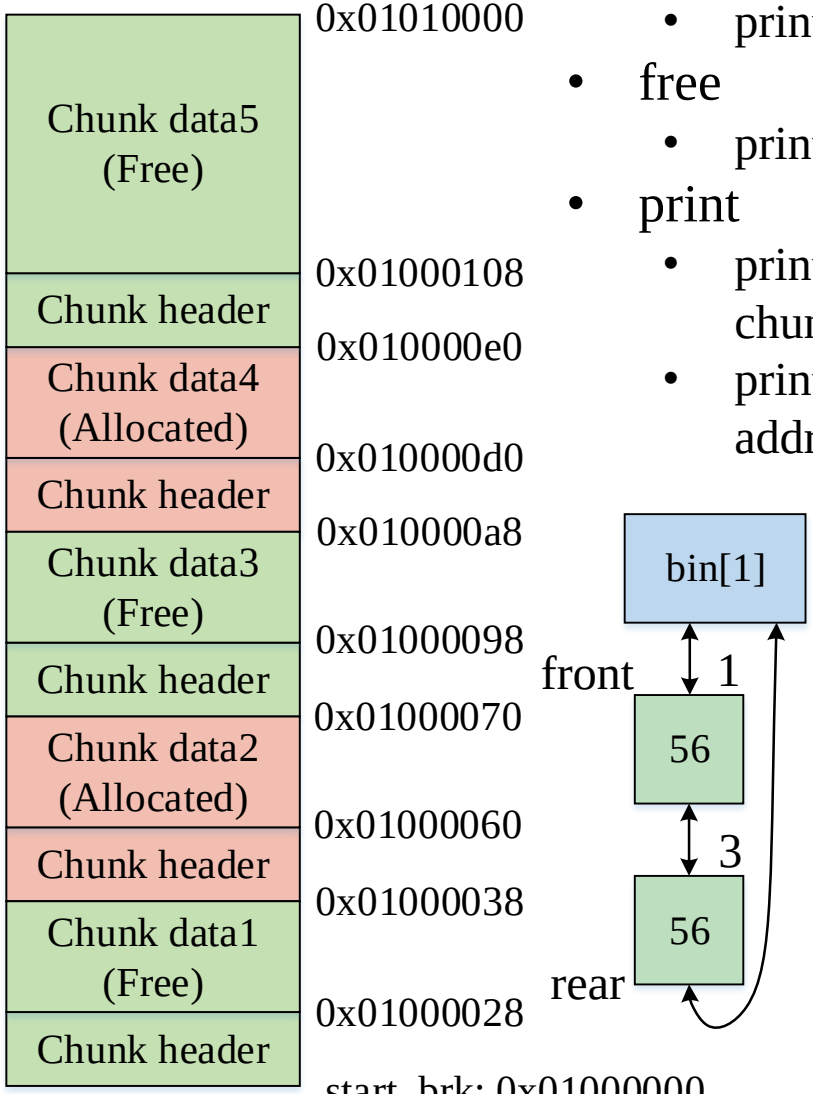
Requirements (1/2)

1. Implement a memory allocator library for user application
   - The library must provide the following 3 functions as the interface
     - void *hw_malloc(size_t *bytes*);
       - *bytes*: the required memory size in bytes
       - Return the valid virtual address (starting address of the ***data*** part) if success; Otherwise, return NULL.
     - int hw_free(void *mem);
       - free the virtual memory
       - *mem*: starting address of the ***data*** part
       - Success: return 1; Fail: return 0.
     - void *hw_get_start_brk();
       - Return the starting address of the heap
   - Use chunk (*slides 5-7*), bin (*slides 8-9*) and sbrk() to manage heap

Requirements (2/2)

2. Write a user application to test the memory allocator library
   - Should receive 3 kinds of commands
     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***
        - Print address and size information of a given bin
        - ***BIN*** can be **bin[0]**, **bin[1]**, **bin[2]**, **bin[3]**, **bin[4]**, **bin[5]**, or **bin[6]**
   - Continuously receive commands from stdin until ***EOF*** (***Ctrl+D***)
     - Should successfully run "cat testfile.txt | hw4_mm_test > outputfile.txt"
     - Command input/output format is shown in *slide 4*

Input/output format



- alloc
  - print **(relative) data address** in a line
- free
  - print success/fail in a line
- print
  - 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	Output example
alloc 16	0x00000028
alloc 16	0x00000060
alloc 16	0x00000098
alloc 16	0x000000d0
free 0x00000028	success
free 0x00000098	success
print bin[1]	0x00000000--------56
	0x00000070--------56

Chunk (1/3)

- The continuous heap space is split into chunk(s) for management
  - Chunk size should be multiple of 8 bytes

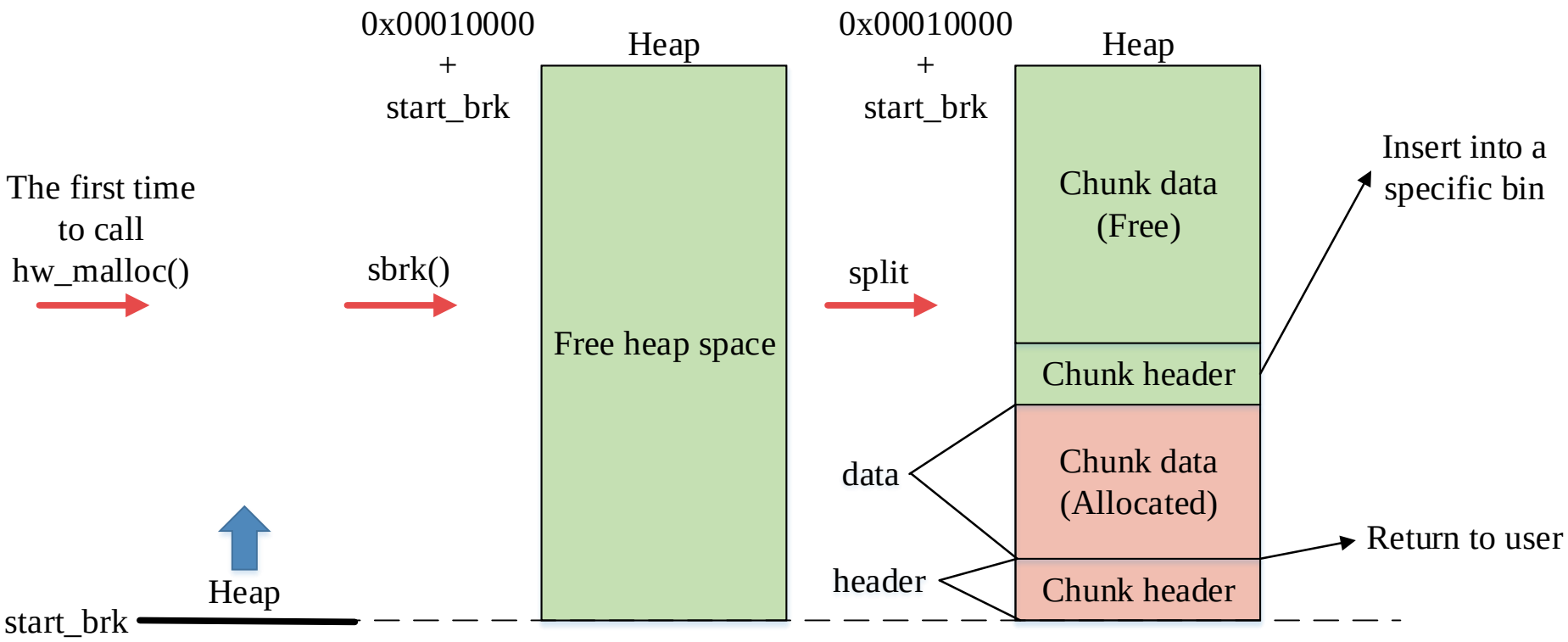
- When hw_malloc() is called for the first time, use sbrk() to allocate a **64KB** heap, and then split it into two chunks (*as in slides 6 and 10*)
  - an allocated chunk (lower address), returned to the caller
  - a free chunk (higher address), inserted in a specific bin

- Each chunk contains two parts, header and data
  - Header (lower address) (*as in slide 7*)
  - Data (higher address), the actual memory space return to caller

- Each free chunk resides in a specific bin (described in *slides 8-9*)

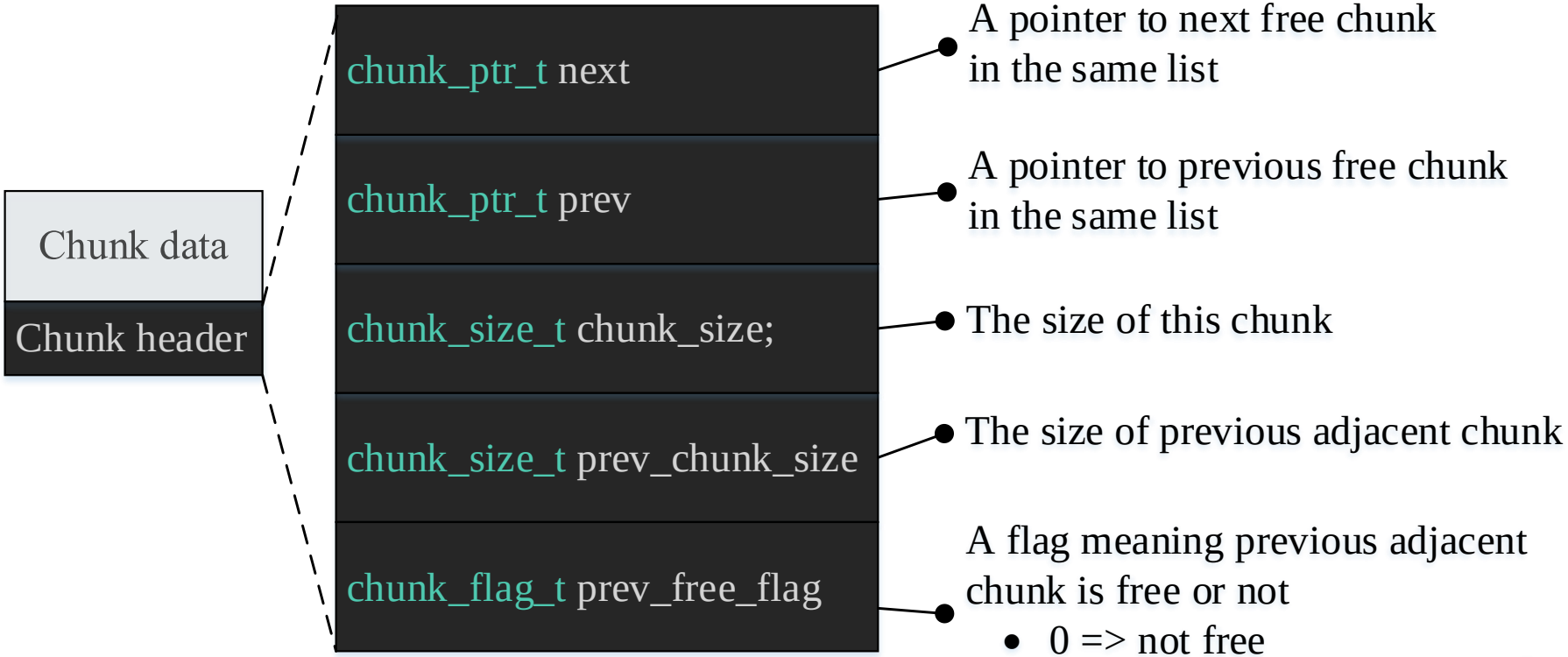
- Adjacent free chunks must be merged (described in *slide 11*)

Chunk (2/3)


0x00010000
+
start_brk
Heap
Free heap space
The first time
to call
hw_malloc()
sbrk()
Heap
start_brk
0x00010000
+
start_brk
Heap
split
Chunk data
(Free)
Chunk header
Chunk data
(Allocated)
Chunk header
data
header
Insert into a
specific bin
Return to user

Chunk (3/3)

- Chuck header (40 bytes)
  - There are 5 members in the header
    - chunk_ptr_t, chunk_size_t, and chunk_flag_t can be defined by yourself, but each of them should be 8 bytes

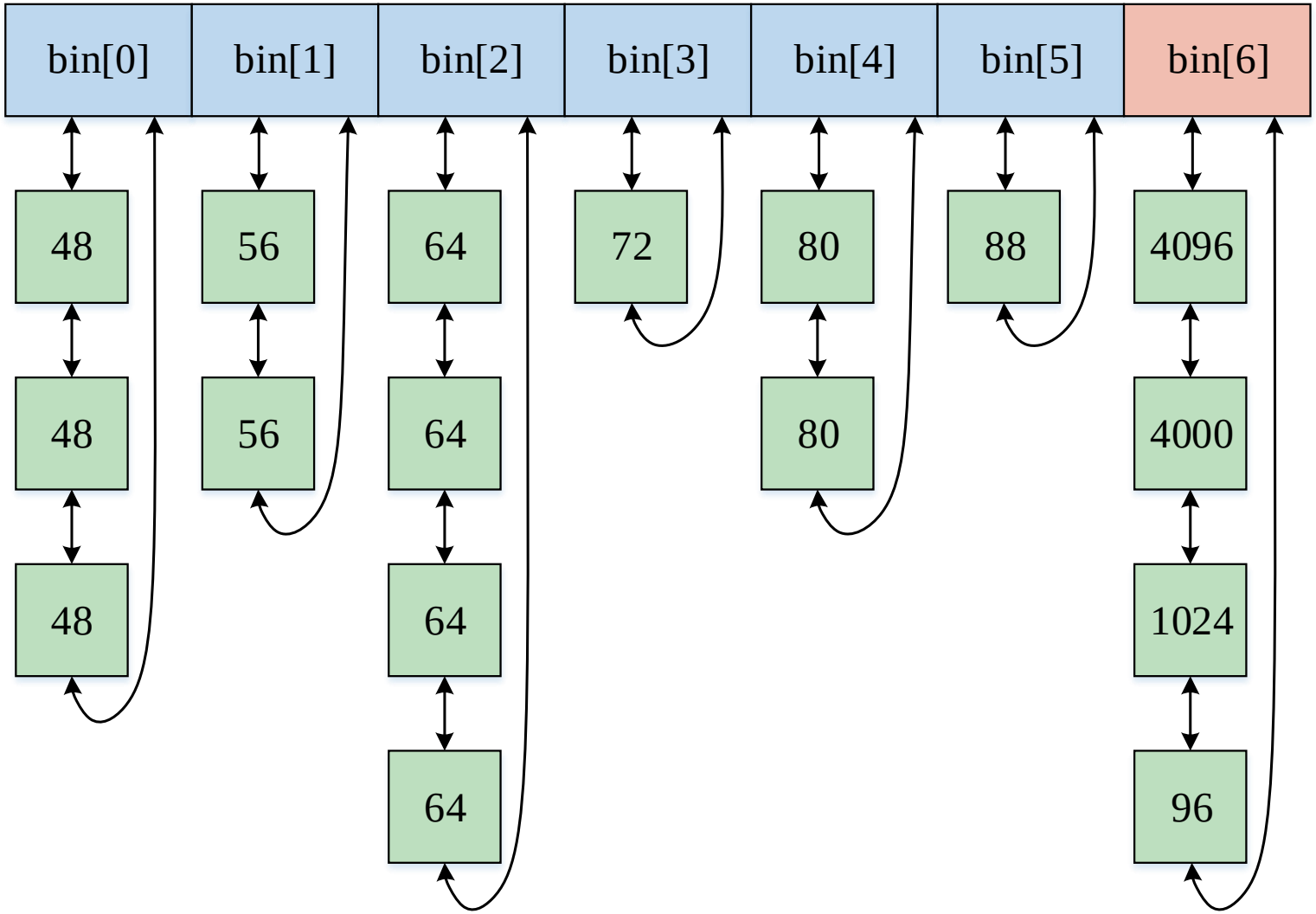


Bins

- A bin is a circular doubly-linked list of **free** chunk(s) (*slide 9*)
- You should manage 7 bins
  - bin[0]-bin[5] hold chunks with fixed size (*as in slide 9*)
  - bin[6] hold chunks with sizes > 88 bytes
    - Chunks in this bin is sorted by chunk size (in **descending** order)

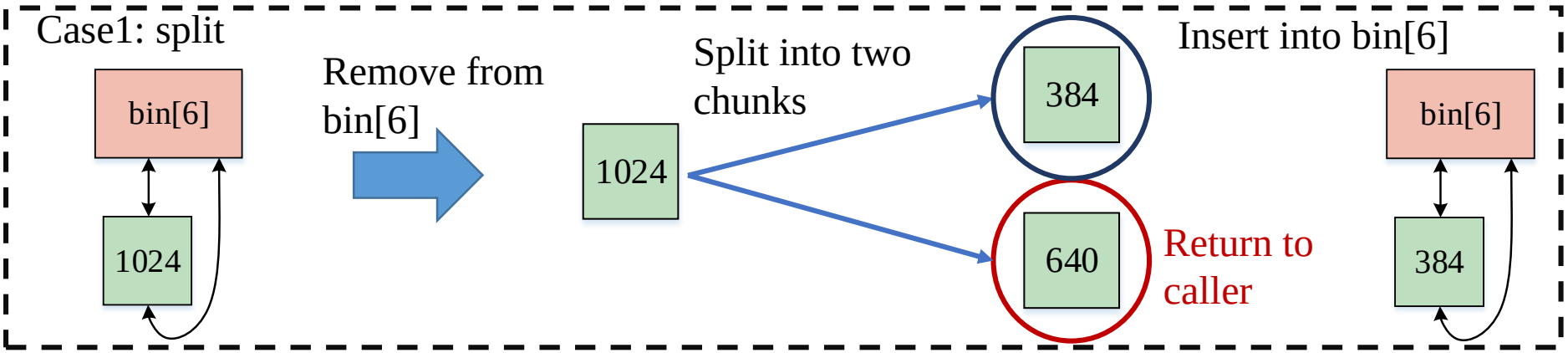
- Use **best fit** to select a chunk during memory allocation
  - If there are multiple chunks with the same size, select the one that is the nearest to the front(insert at the rear)
  - You may need to split the selected chunk (*slide 10*)

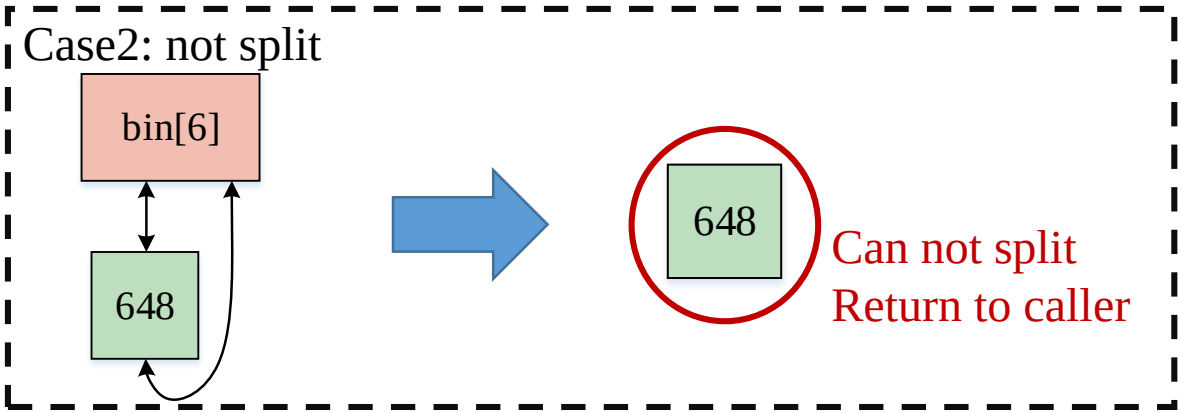
Bin example


bin[0]
bin[1]
bin[2]
bin[3]
bin[4]
bin[5]
bin[6]
48
48
48
56
56
64
64
64
72
80
80
88
4096
4000
1024
96

Split

- When hw_malloc() is called, *split* may be performed on the chunk you wish to return to the caller; If split occurs, return the chunk with the **lower** address
- A chunk in bin[0]-bin[5] **CANNOT** be split
- A chunk **CANNOT** be split if the remaining size **after split** < 48 bytes (40+8)
- A chunk in bin[6] may be split
  - e.g., allocate a 640-byte chunk (40-byte header + 600-byte data) in the following 2 cases



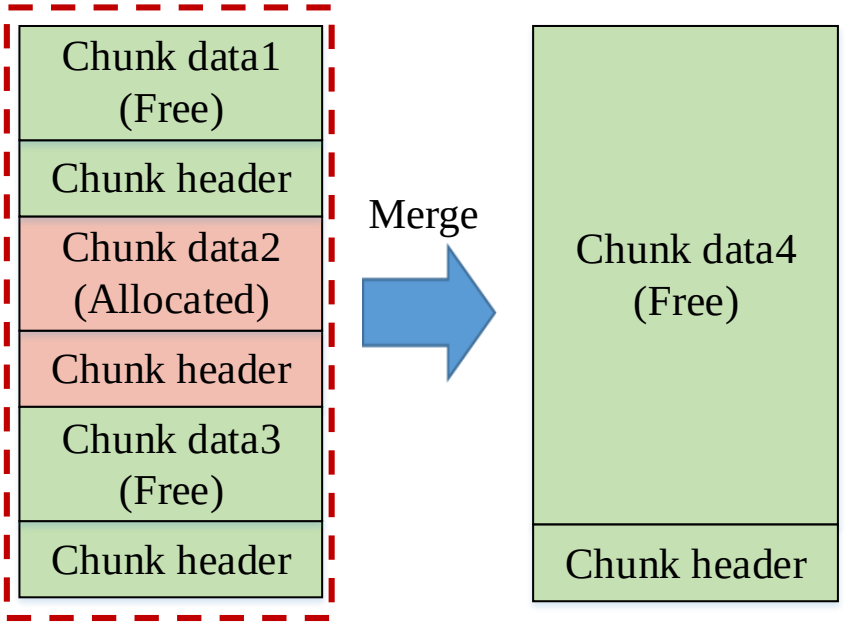


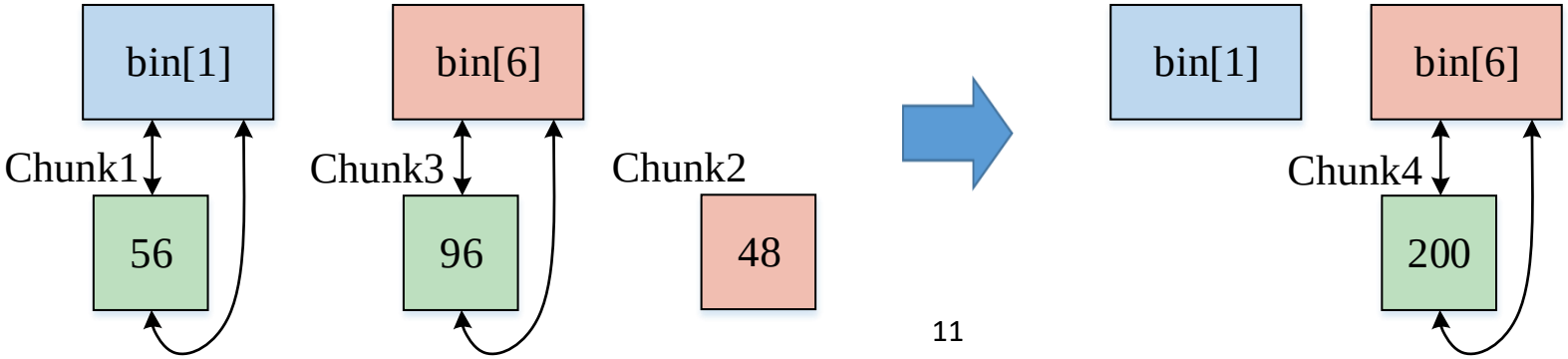
Merge

- When hw_free() is called, adjacent free chunks must be merged into one

For example,

- Chunk2 (48 bytes) is going to be freed
- Chunk1 and Chunk3 are both free and adjacent to Chunk2.
- Chunk1, Chunk2, Chunk3 should be merged (become Chunk4)





Bonus

- Implement a dynamically growing/shrinking memory allocator library
- Implement a memory allocator library for multi-threaded process

References

- sbrk()
  - Linux man page

- Streams, pipes, and redirects
  - IBM