PA#3: Malloc Dynamic Memory Allocator

SCE213 (Operating Systems)
Spring 2022

TA: Jinwoo Jeong (정진우)
Email: jjw8967@ajou.ac.kr

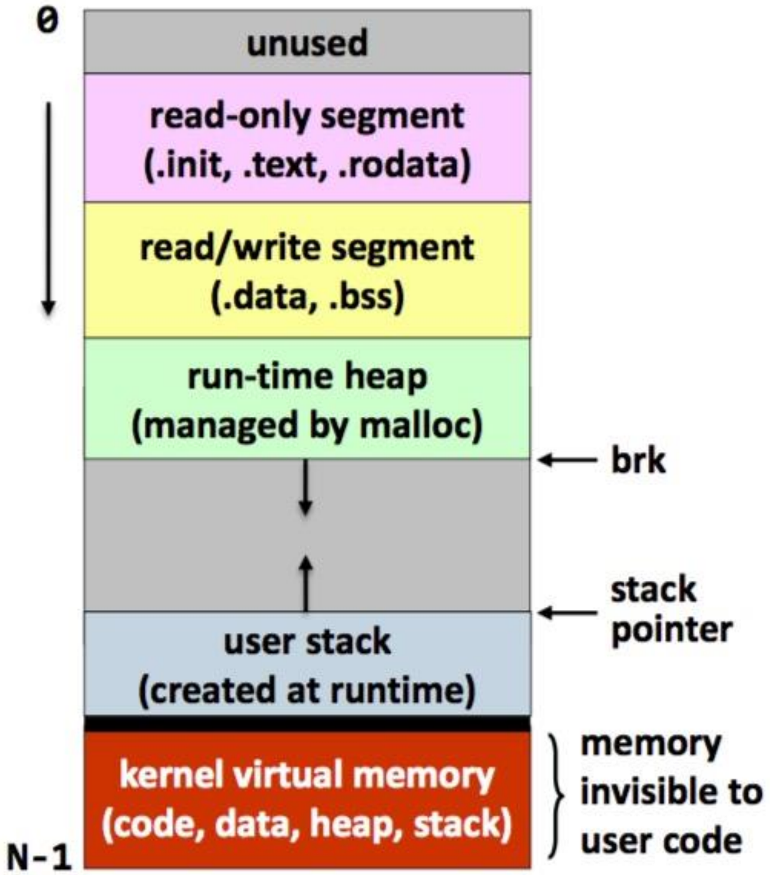
Adapted slides from Changwoo Min@VT

Goal

- Dynamic Memory Allocator
  - Implement the dynamic memory allocation functions:
    - my_malloc()
    - my_realloc()
    - my_free()
  - The functions behave similarly to the POSIX standard functions, such as malloc(), realloc(), free()

Process Address Space

- Process's abstract view of memory
  - Static area
    - Code & Data
  - Dynamic area
    - Allocated at runtime
    - Can grow or shrink
    - Stack & Heap



0
unused
read-only segment (.init, .text, .rodata)
read/write segment (.data, .bss)
run-time heap (managed by malloc)
brk
stack pointer
user stack (created at runtime)
kernel virtual memory (code, data, heap, stack)
memory invisible to user code
N-1

Dynamic Area (Stack & Heap)

- Stack
  - The region of the stack that provides the execution environment of a *particular* call to a function
    - The function arguments and local variables is stored in stack

- Heap
  - Unlike stack memory, heap memory is allocated explicitly by programmers and it won’t be deallocated until it is explicitly freed
    - Heap memory is managed by malloc(), realloc(), free(), etc.

External Fragmentation in Heap

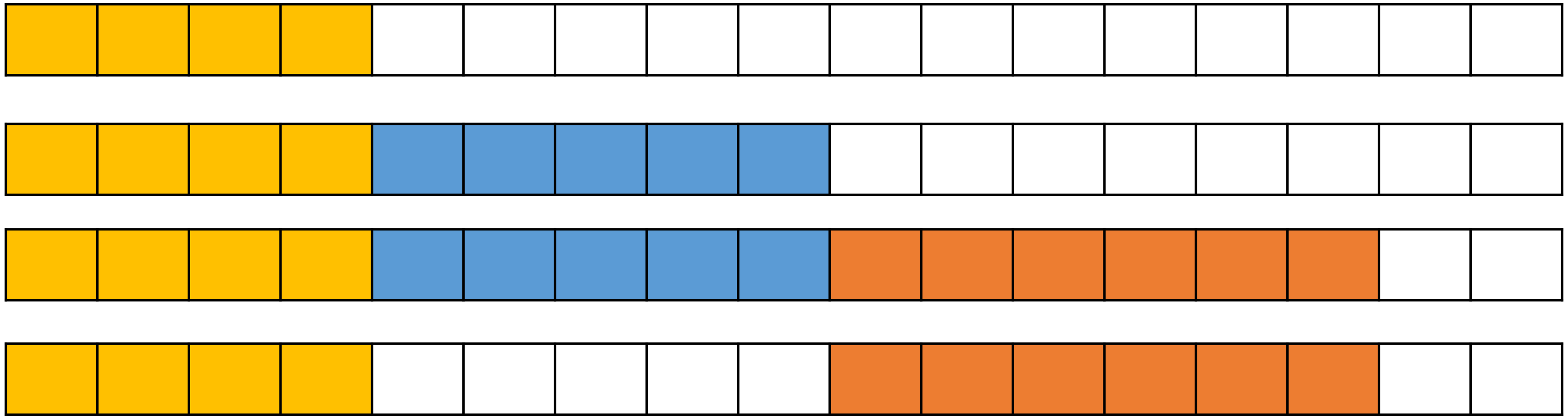
- There is enough aggregate heap memory, but no single free block is large enough

p1 = malloc(4)

p2 = malloc(5)

p3 = malloc(6)

free(p2)



p4 = malloc(6)

Oops! (what would happen now?)

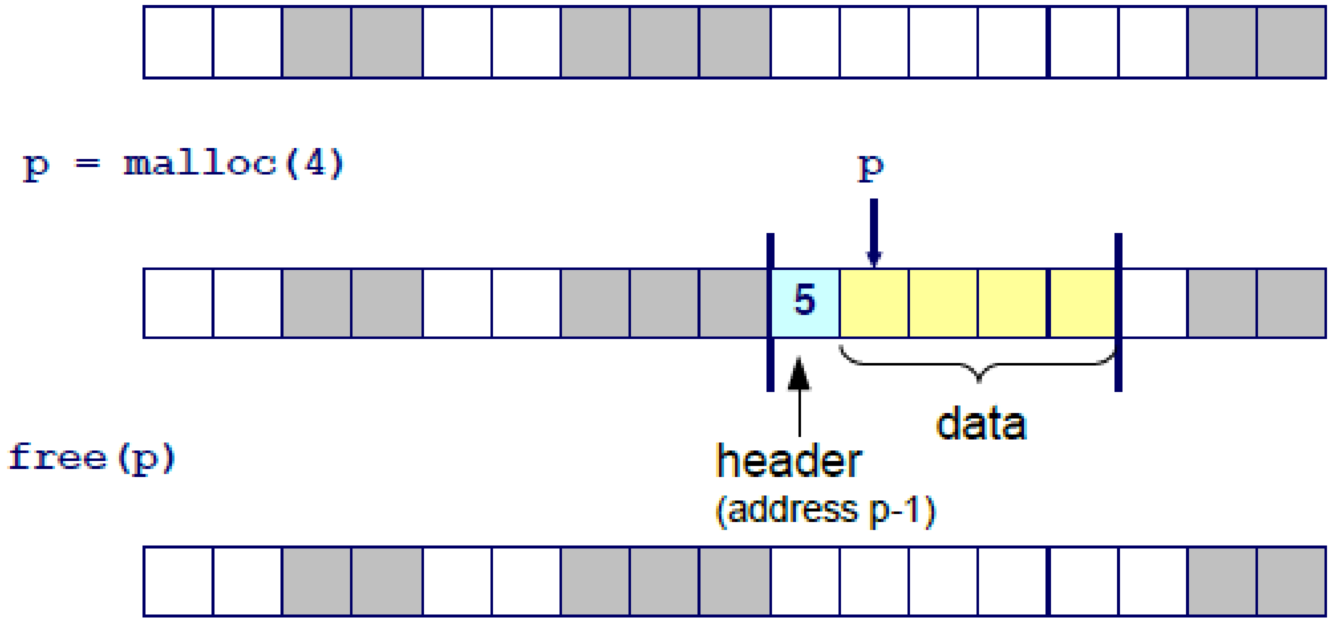
Depends on the pattern of future requests
Difficult to plan for

Implementation issues you need to solve!

- How do I know how much memory to free just given a pointer?

- Keep the length of the block in the header preceding the block

- Requires an extra word for every allocated block



Keeping Track of Free Blocks

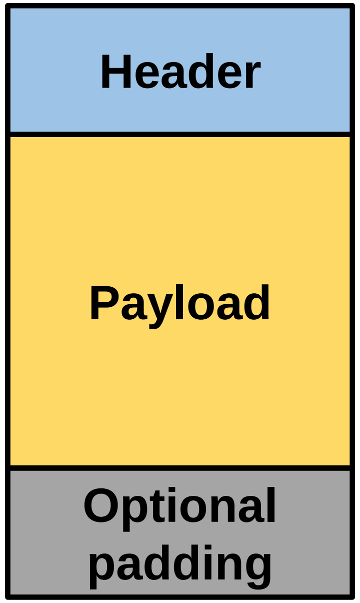
- One of the biggest jobs of an allocator is knowing where the free memory is

- The allocator's approach to this problem affects:
  - Throughput - time to complete a **malloc()** or **free()**
  - Space utilization - amount of extra metadata used to track location of free memory

- There are many approaches to free space management

Free List

- For each block we need both size and allocation status
- **Header** contain metadata
  - Allocation status
  - Size
  - List_head
- Payload: application data

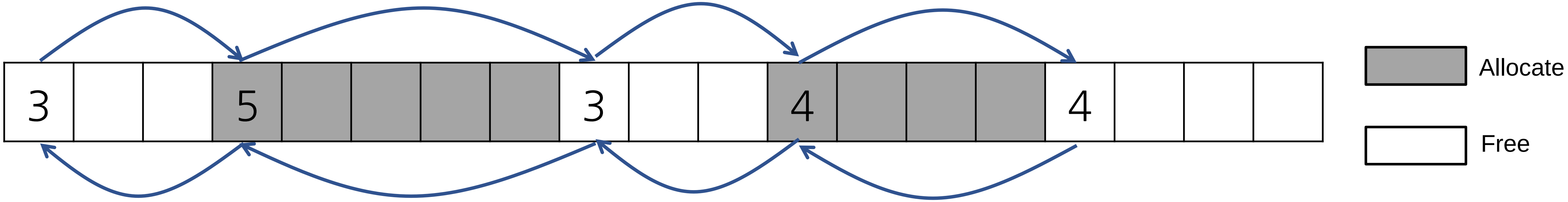


Header

Payload

Optional padding

Memory Block



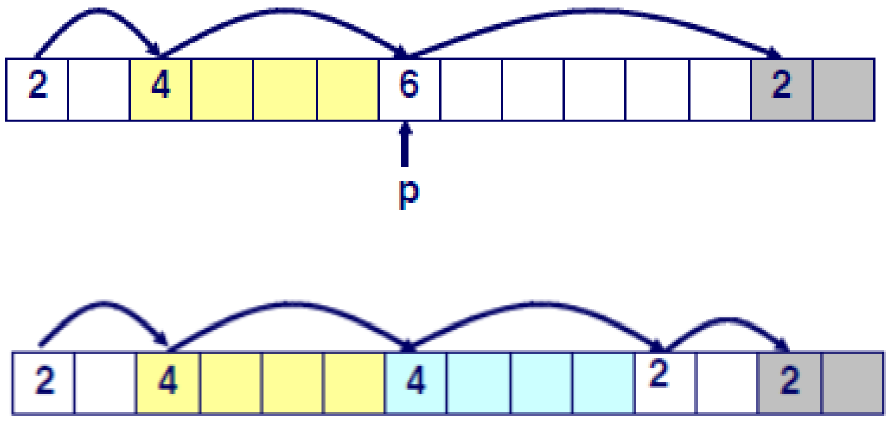
3 5 3 4 4

Allocate

Free

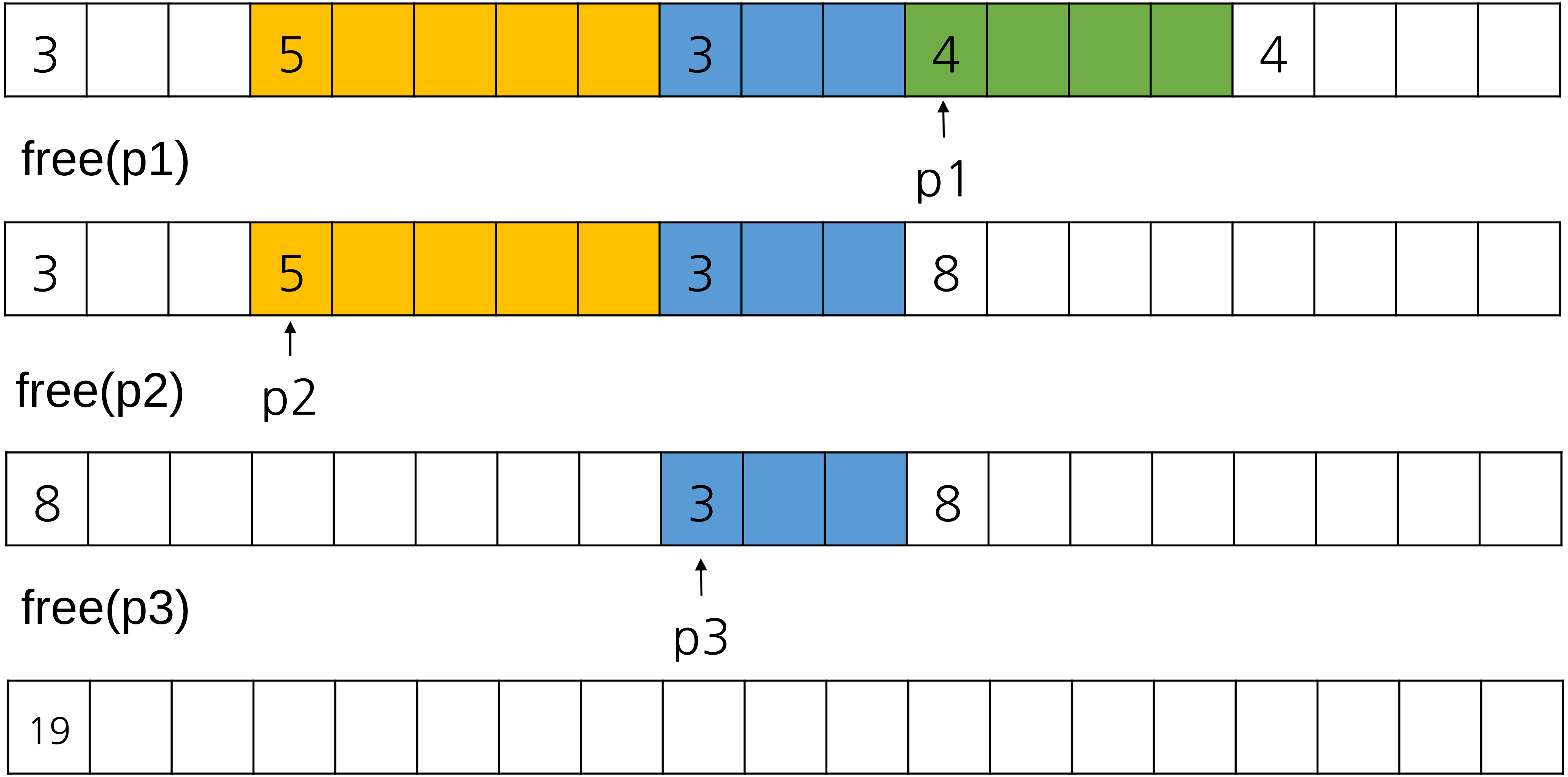
Allocating a Block

- Splitting free blocks
  - Since allocated space might be smaller than free space,
    we may need to split the free block that we're allocating within



place(p, 4)

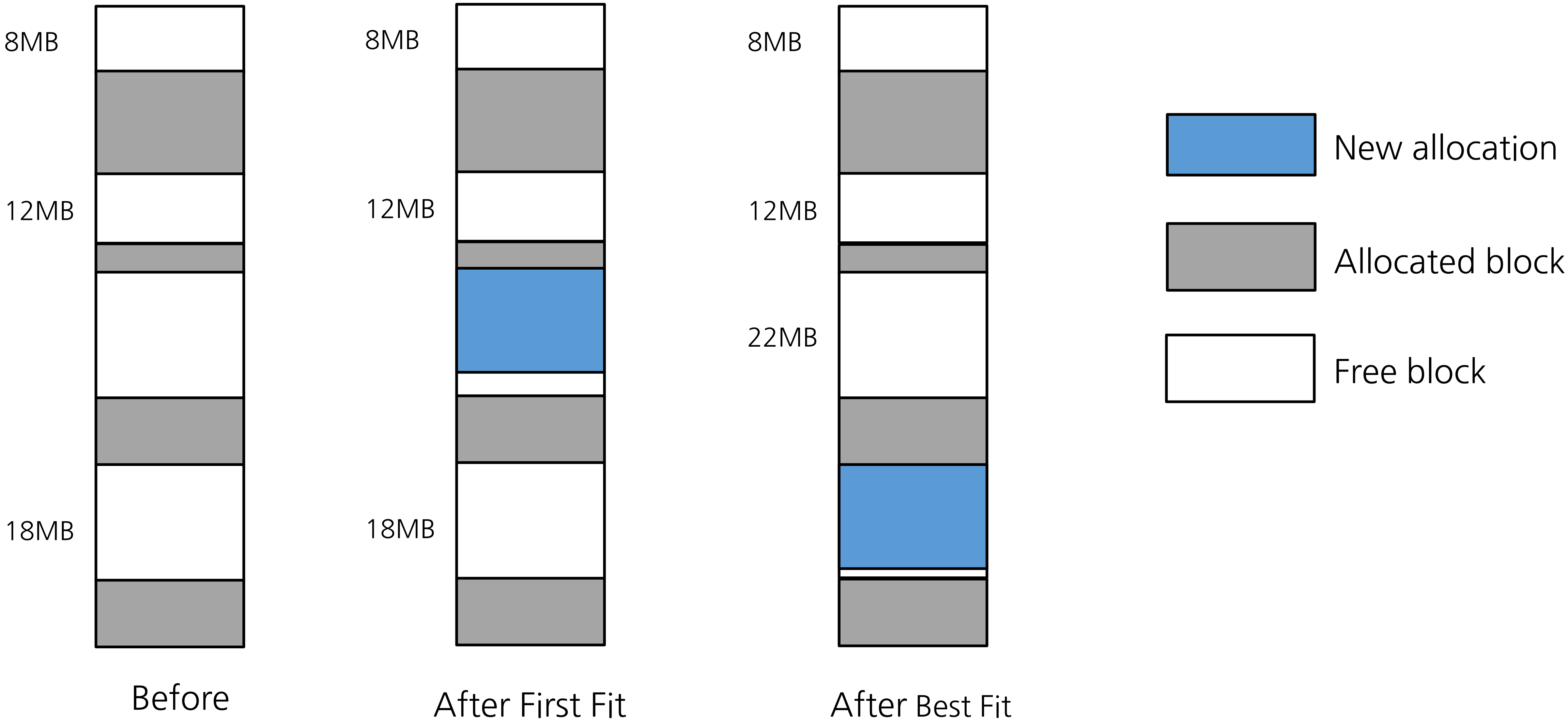
Free List: Bidirectional Coalescing



How do we find a free block?

- There are dynamic partitioning placement algorithm
  - First Fit
    - Start scanning from the beginning of the heap
    - Traverse each block until (a) we find a free block and (b) the block is large enough to handle the request.
  - Best Fit
    - Traverse all free blocks and considers the smallest free block that is adequate.
    - Try to find a free block which is close to actual memory size needed.

Placement Algorithm



So what should I do for PA#3?

- Implement dynamic memory allocation functions
  - **my_malloc()**
  - **my_realloc()**
  - **my_free()**

- You should only sbrk() to implement **my_malloc()** and **my_realloc()** instead **malloc()** family functions
  - sbrk() increments the program's data space by increment bytes

  - You will get 0 pts if you use it

- All allocation sizes are aligned by **32 bytes**

PA#3: Deliverables

- Submission by June 27th, 11:59 PM
  - Submit only **malloc.c** for the code

- PAsubmit
  - Start project by cloning PA#3 repository

```
git clone https://github.com/csl-ajou/sce213-project3
```