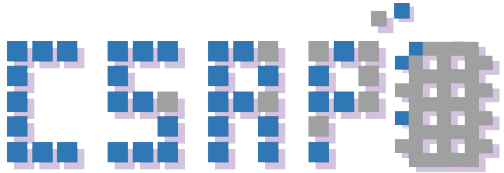
System Programming
Lab Session #4

Memory Lab

2019/10/15

sysprog@csap.snu.ac.kr



Computer Systems and Platforms Laboratory
School of Computer Science and Engineering
Seoul National University

Overview

- Handout / Handin instructions
- Simulated heap explanations
- Dynamic memory allocator specifications
- An example to get start with
- Q&A

Handout

- **Fork** the https://git.csap.snu.ac.kr/sysprog/memorylab repository to your personal account.
- Set your project to private visibility.
- A detailed handout and this presentation are on etl, as usual.

Handin

- You will have to submit your **report** and `mm.c` source file via your repository.
- **Please** copy/rename your files when submitting to follow the following naming format: 20XX-XXXXX_mm.c and 20XX-XXXXX_report.pdf

Simulated Heap

Simulated Heap

- This lab uses of a **simulated heap.**

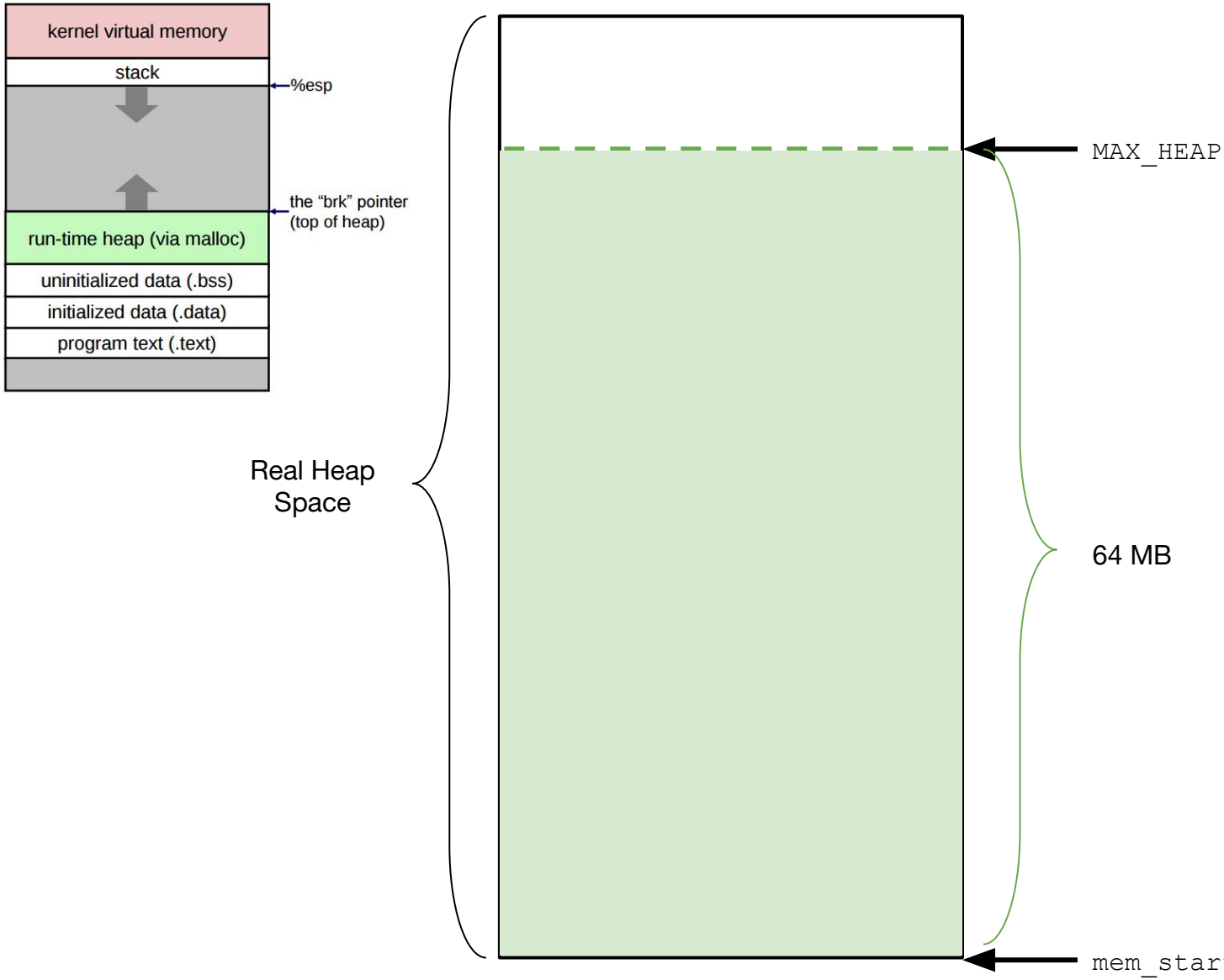
- Your `mm_malloc` and `mm_free` functions will allocate/deallocate blocks within this simulated heap.

- Important variables:
  - `*mem_start_brk` points to first byte of the heap
  - `*mem_brk` points to last byte of the heap
  - `*mem_max_addr` largest legal heap address

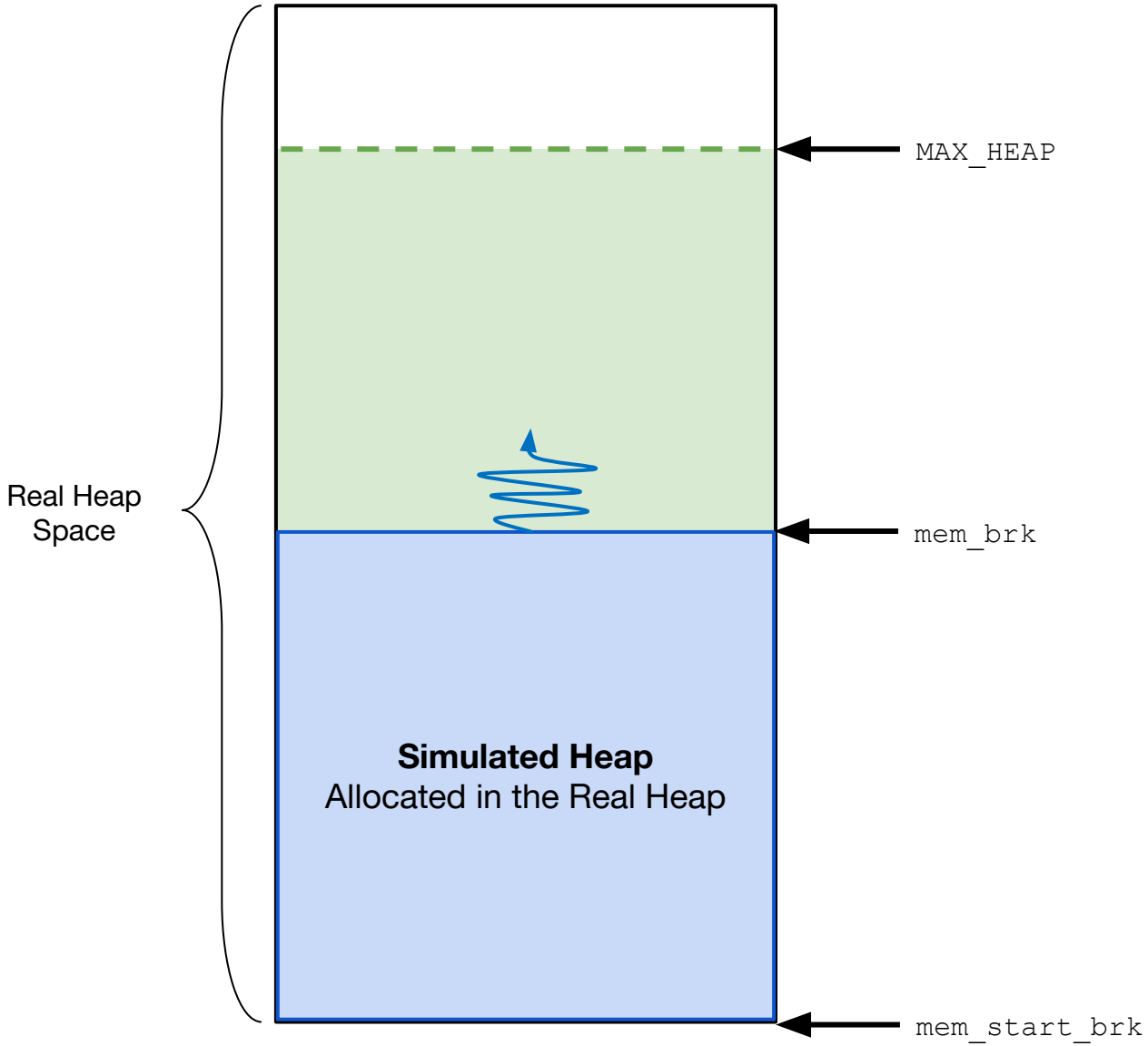
- They can be accessed via:
  - `void* mem_heap_lo()` -> `mem_start_brk`
  - `void* mem_heap_hi()` -> `mem_brk`
  - `void* mem_heapsize()` -> `mem_brk - mem_start_brk`

- You should use `mem_sbrk(N)` to extend your simulated heap by `N` Bytes.

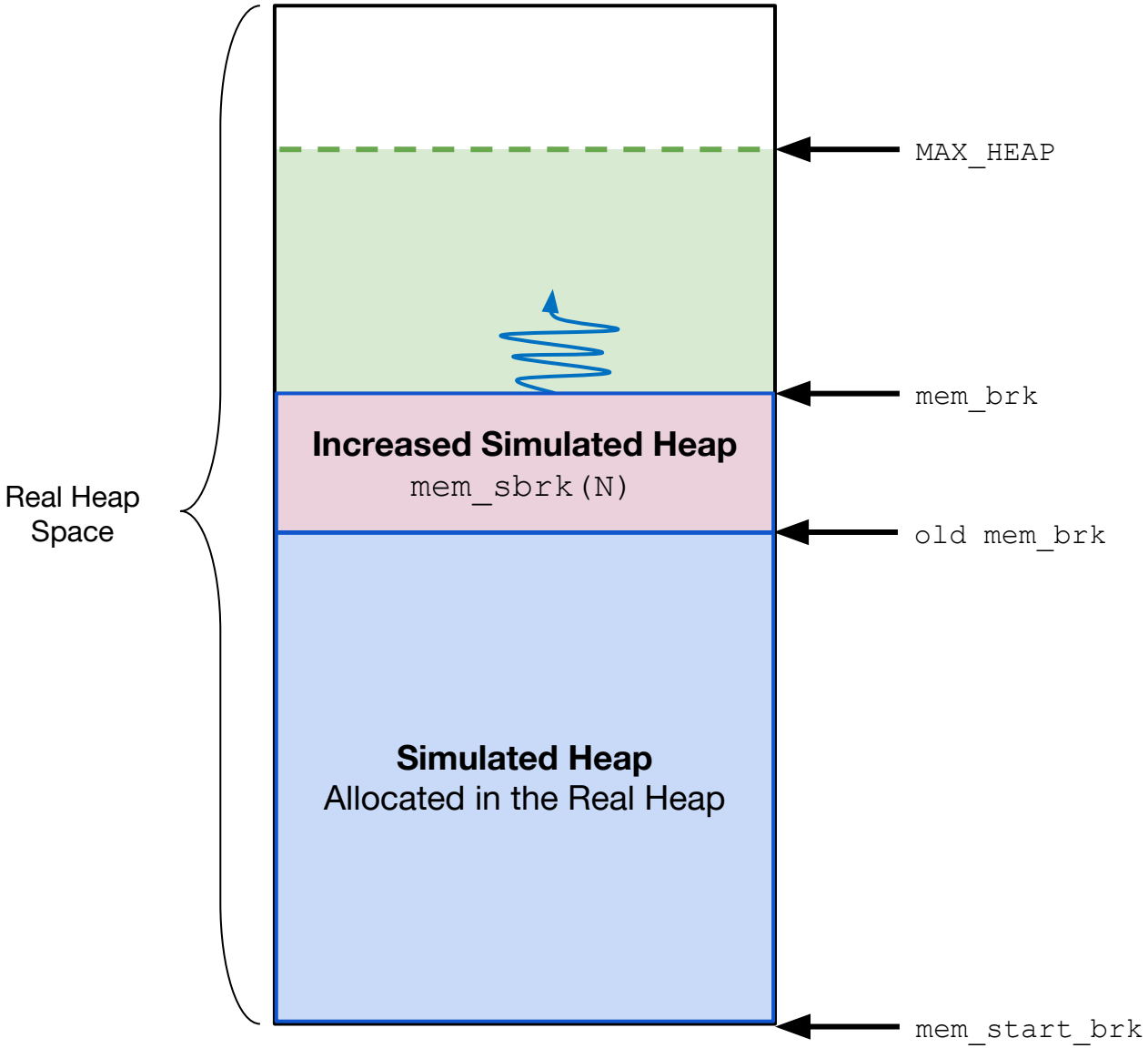
Simulated Heap: Init



Simulated Heap: In Function



Simulated Heap: Making More Space



Dynamic Memory Allocator

Dynamic Memory Allocator

- In this lab you will implement your own dynamic memory allocator.

- The fundamental methods you need to implement are:
  - `void *mm_malloc(void)`
  - `void *realloc(void *ptr, size_t size)`
  - `void mm_free(size_t size)`
  - `int  mm_init(void *ptr)`
  - `void mm_exit(void)`

- Let's have a look at these functions and a few more that will be helpful

Dynamic Memory Allocator

- `mm_init`: performs the necessary initializations, like allocating the initial heap area in the simulated heap and setting up the block/list structures you use.

- `mm_malloc`: returns a pointer to an allocated memory block payload of *size* Bytes. This block should be within the simulated heap region and not overlapping with any other block.

  Tasks:

  1. maintain 8B alignment
  2. find an available free block or make more space
  3. place the block

- `place`: update the info of headers and footers or any other structure you are using.

Dynamic Memory Allocator

- `find_fit`(several algorithms):

  first fit
  next fit
  best fit

- `mm_free`: frees a block pointed by ptr. This ptr should have been returned by an earlier call to mm_malloc. This function should fail if the if called on an unallocated block (exit the program).

  Tasks:

  1. unallocate the block reference by ptr
  2. update the info of header and footer
  3. coalesce if it's in your policy
  4. fail if the block was unallocated

Dynamic Memory Allocator

- `coalesce`: (4 types in textbook)
  merges adjacent free blocks to avoid external fragmentation. Might be used or not depending on your memory management policy

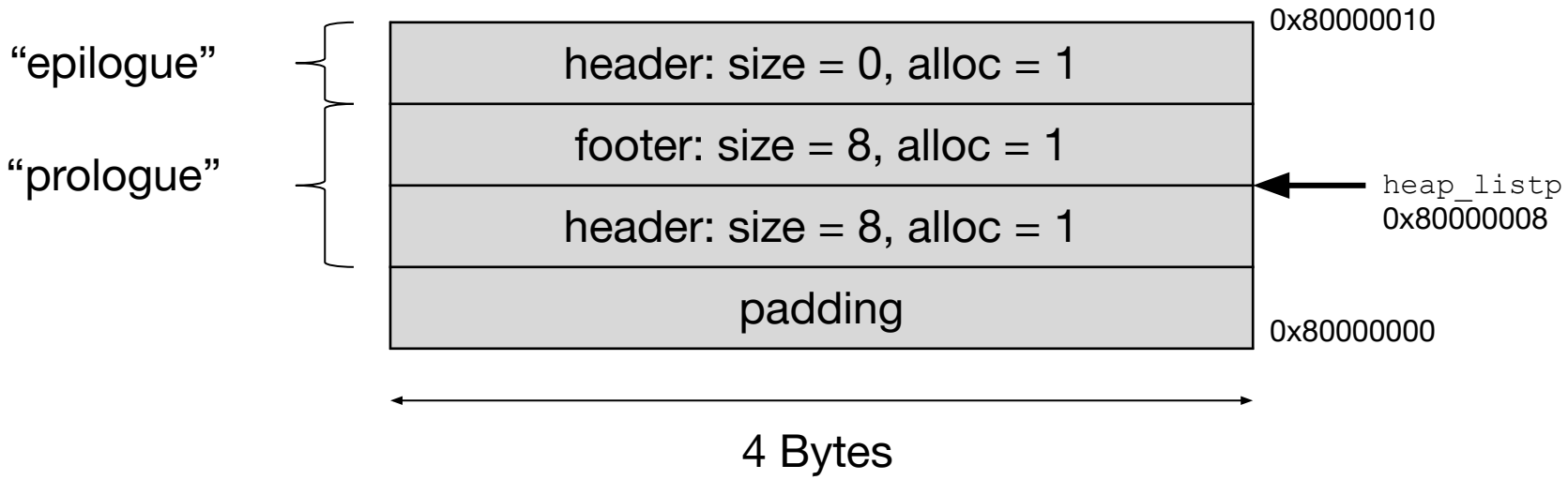
- `mm_realloc`: you do not need to implement this function. If you want to learn more about its behaviour, refer to the handout.

- `mm_exit`: handles memory leaks. Free all the unfreed blocks in the simulated heap.

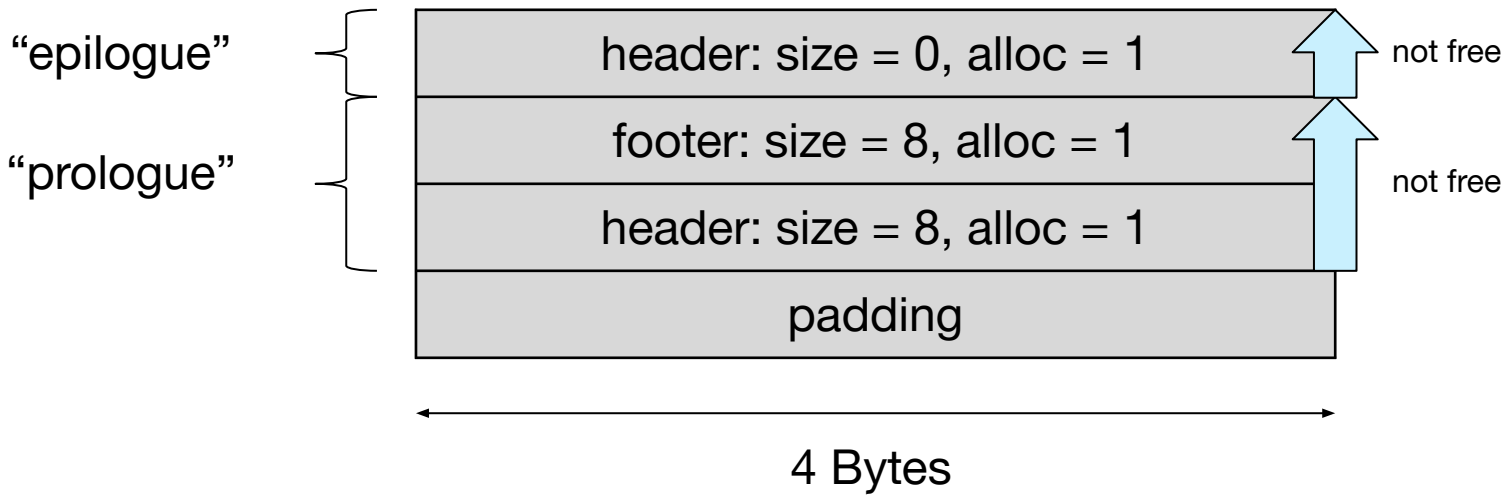
Implicit free list example based on chapter 9.9 in the Textbook

Getting Started

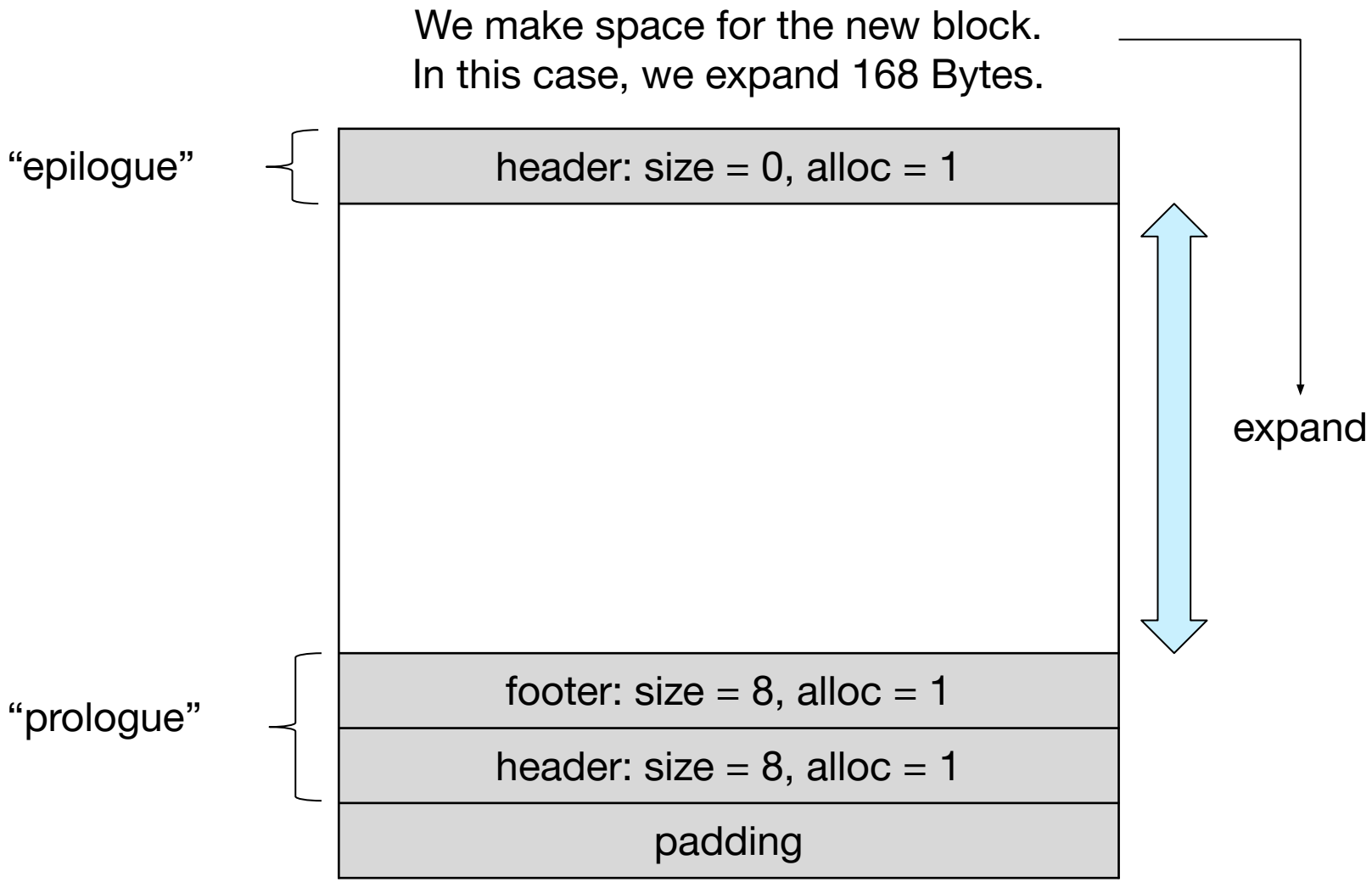
mm_init()


“epilogue”
“prologue”
header: size = 0, alloc = 1
footer: size = 8, alloc = 1
header: size = 8, alloc = 1
padding
0x80000010
heap_listp
0x80000008
0x80000000
4 Bytes

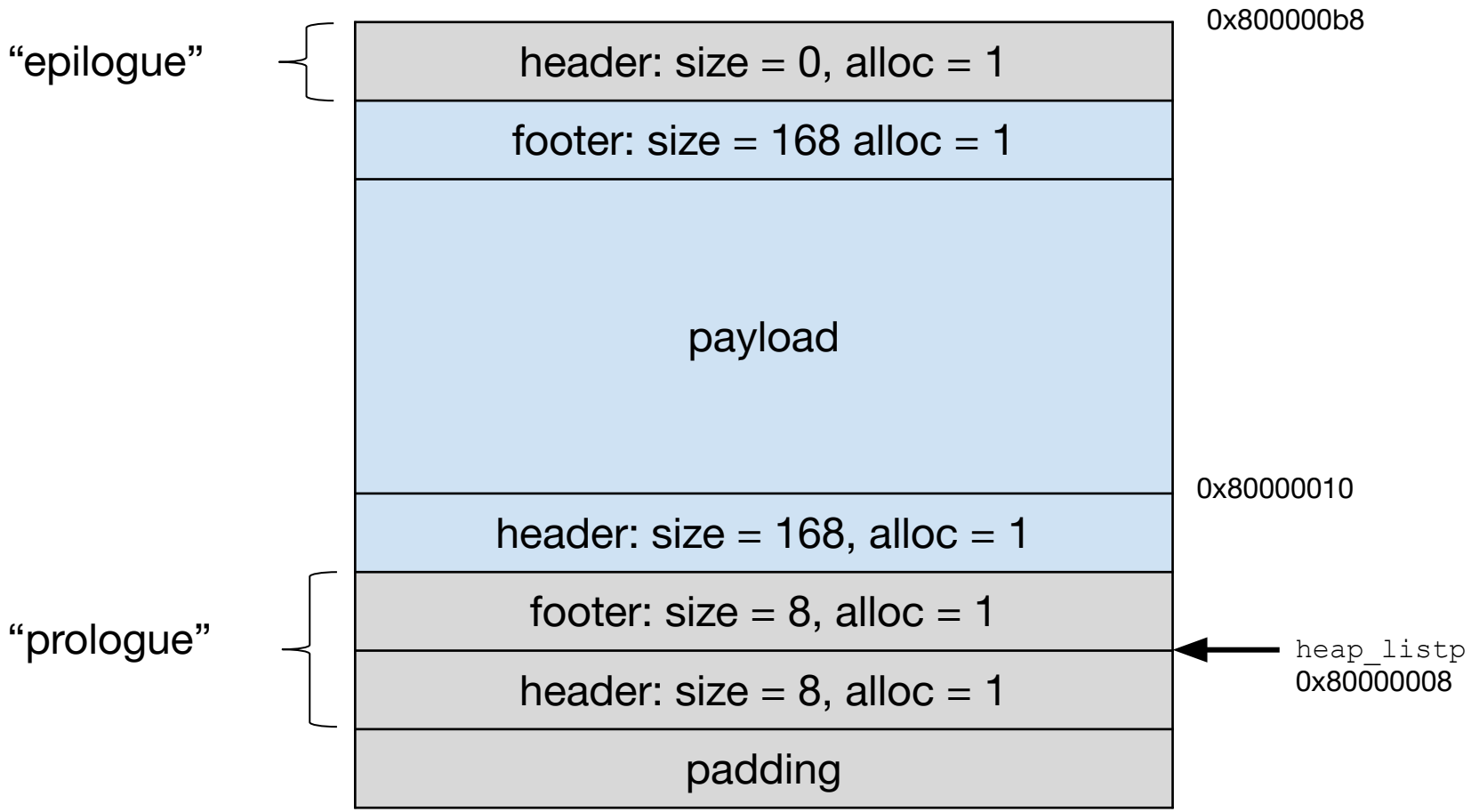
mm_malloc(160)



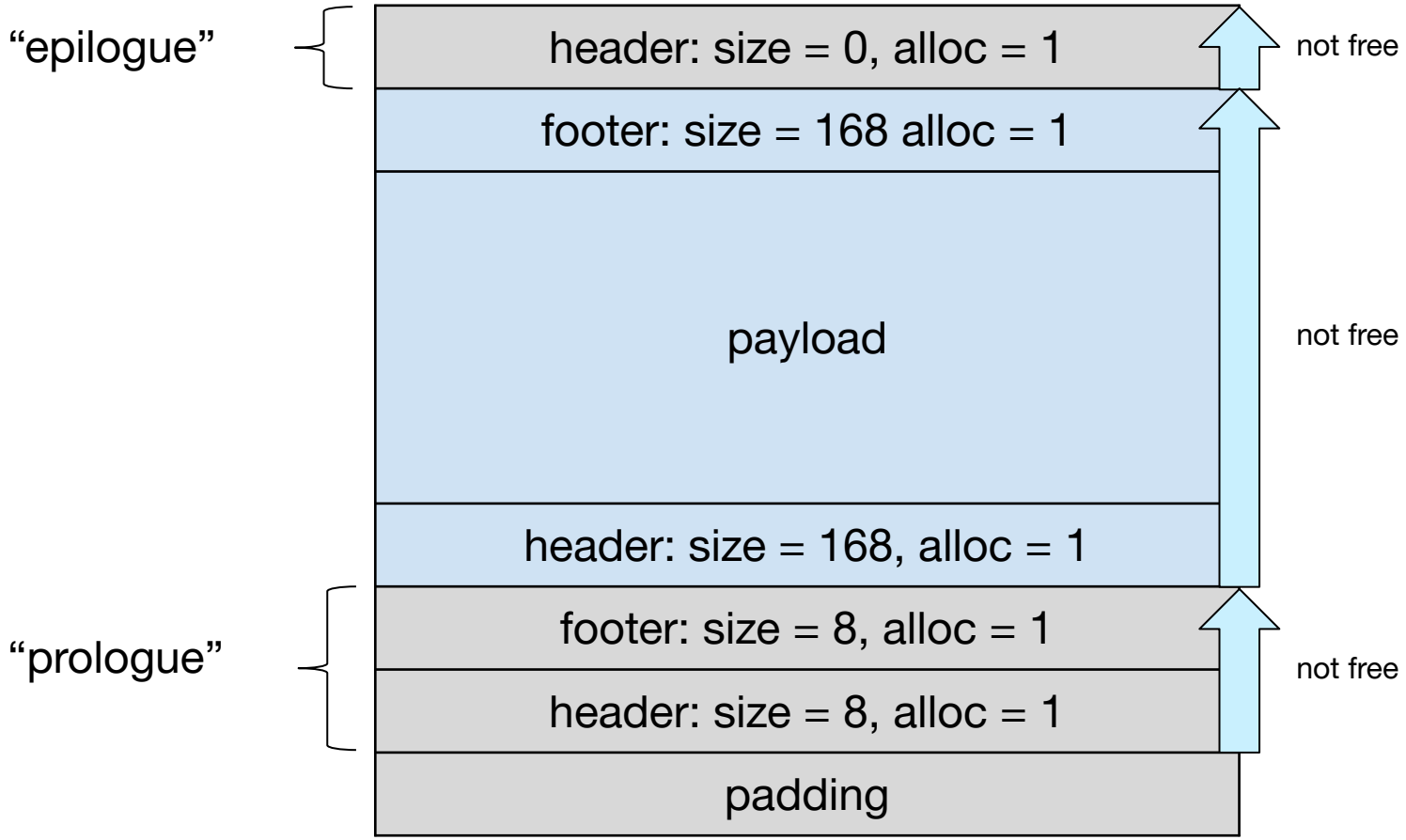
mm_malloc(160)



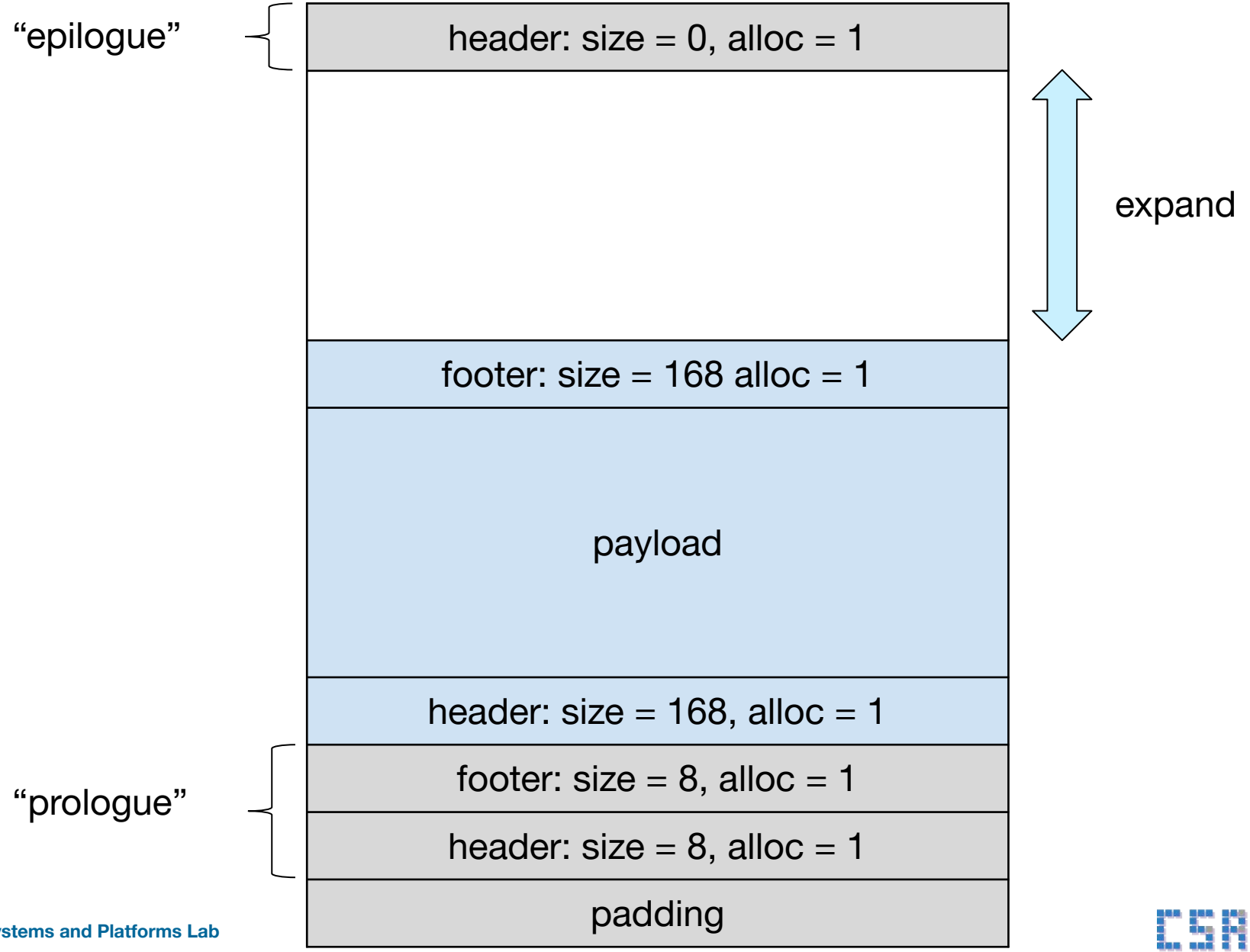
mm_malloc(160)



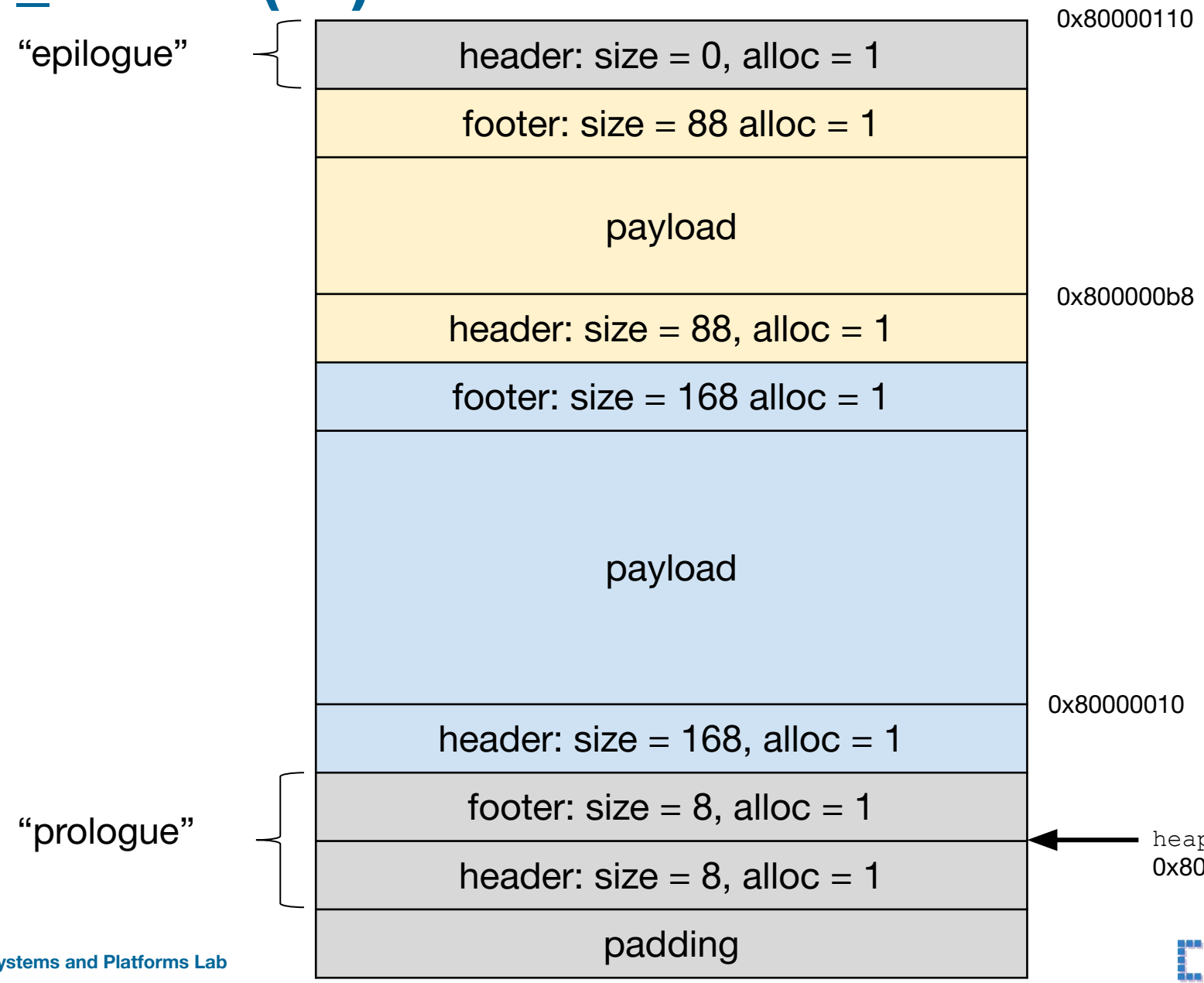
mm_malloc(80)


“epilogue”
header: size = 0, alloc = 1
not free
footer: size = 168 alloc = 1
payload
not free
header: size = 168, alloc = 1
“prologue”
footer: size = 8, alloc = 1
not free
header: size = 8, alloc = 1
padding

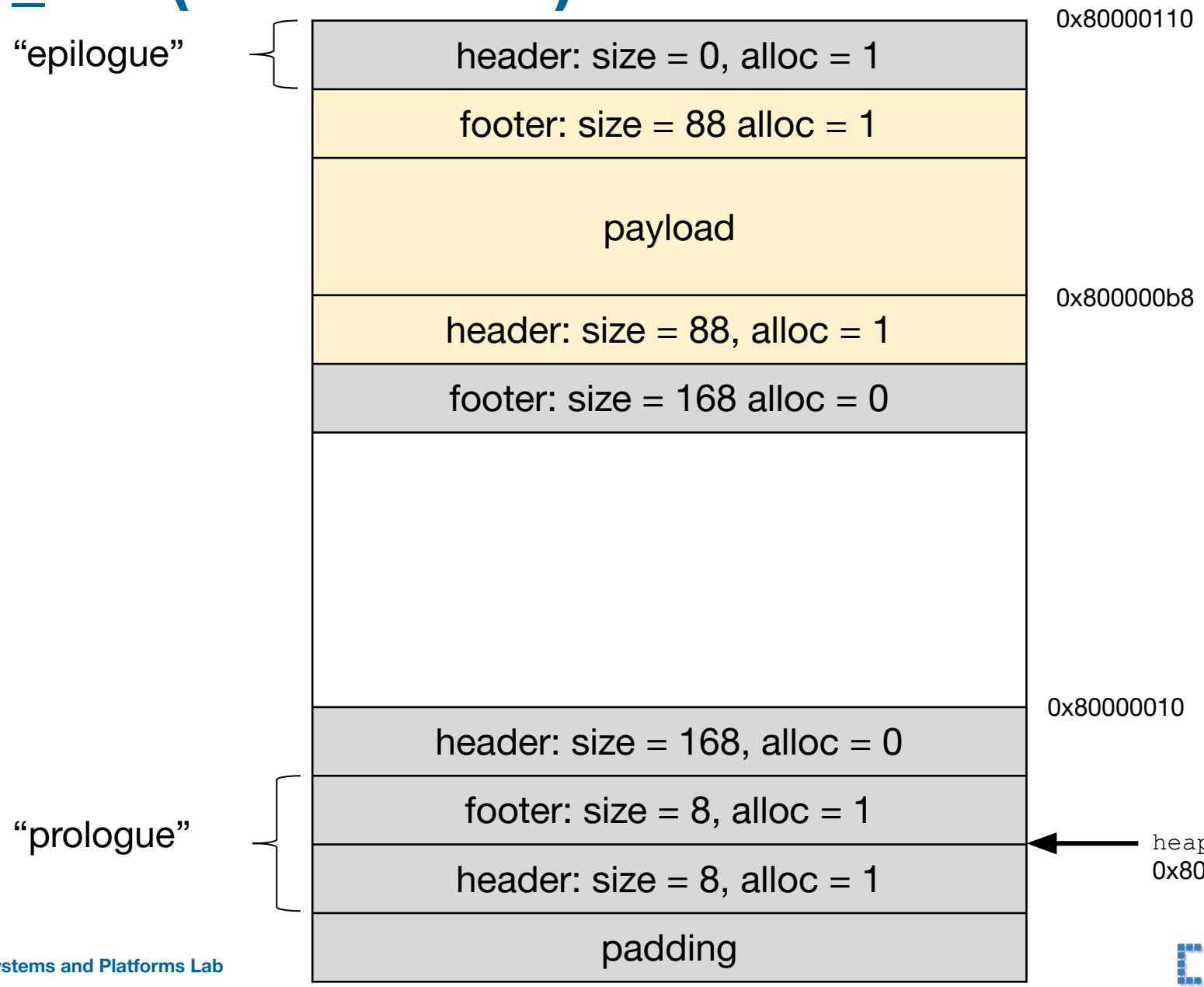
mm_malloc(80)


“epilogue”
header: size = 0, alloc = 1
expand
footer: size = 168 alloc = 1
payload
header: size = 168, alloc = 1
“prologue”
footer: size = 8, alloc = 1
header: size = 8, alloc = 1
padding

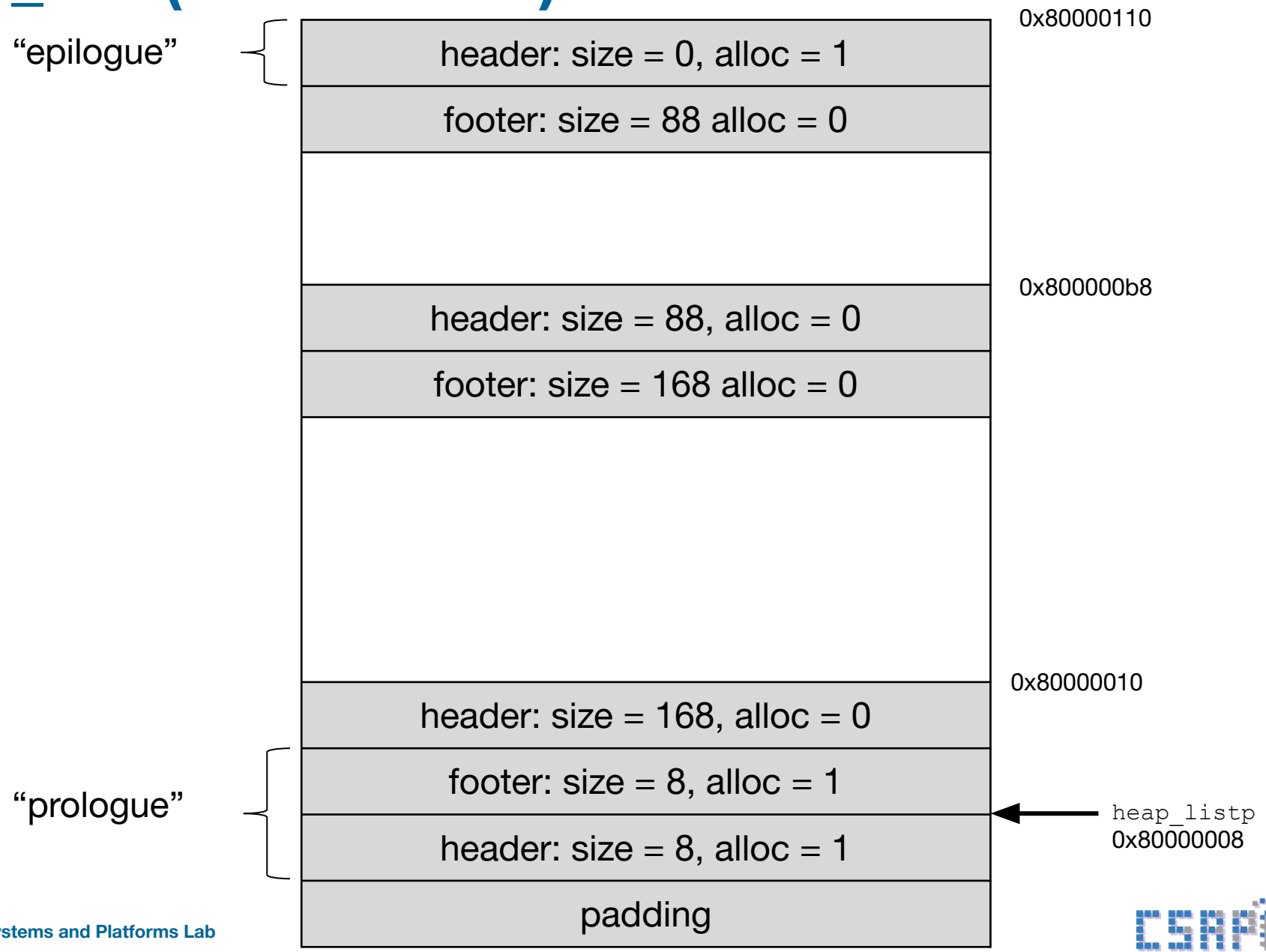
mm_malloc(80)


“epilogue”
header: size = 0, alloc = 1
footer: size = 88 alloc = 1
payload
header: size = 88, alloc = 1
footer: size = 168 alloc = 1
payload
header: size = 168, alloc = 1
“prologue”
footer: size = 8, alloc = 1
header: size = 8, alloc = 1
padding
0x80000110
0x800000b8
0x80000010
heap_listp
0x80000008

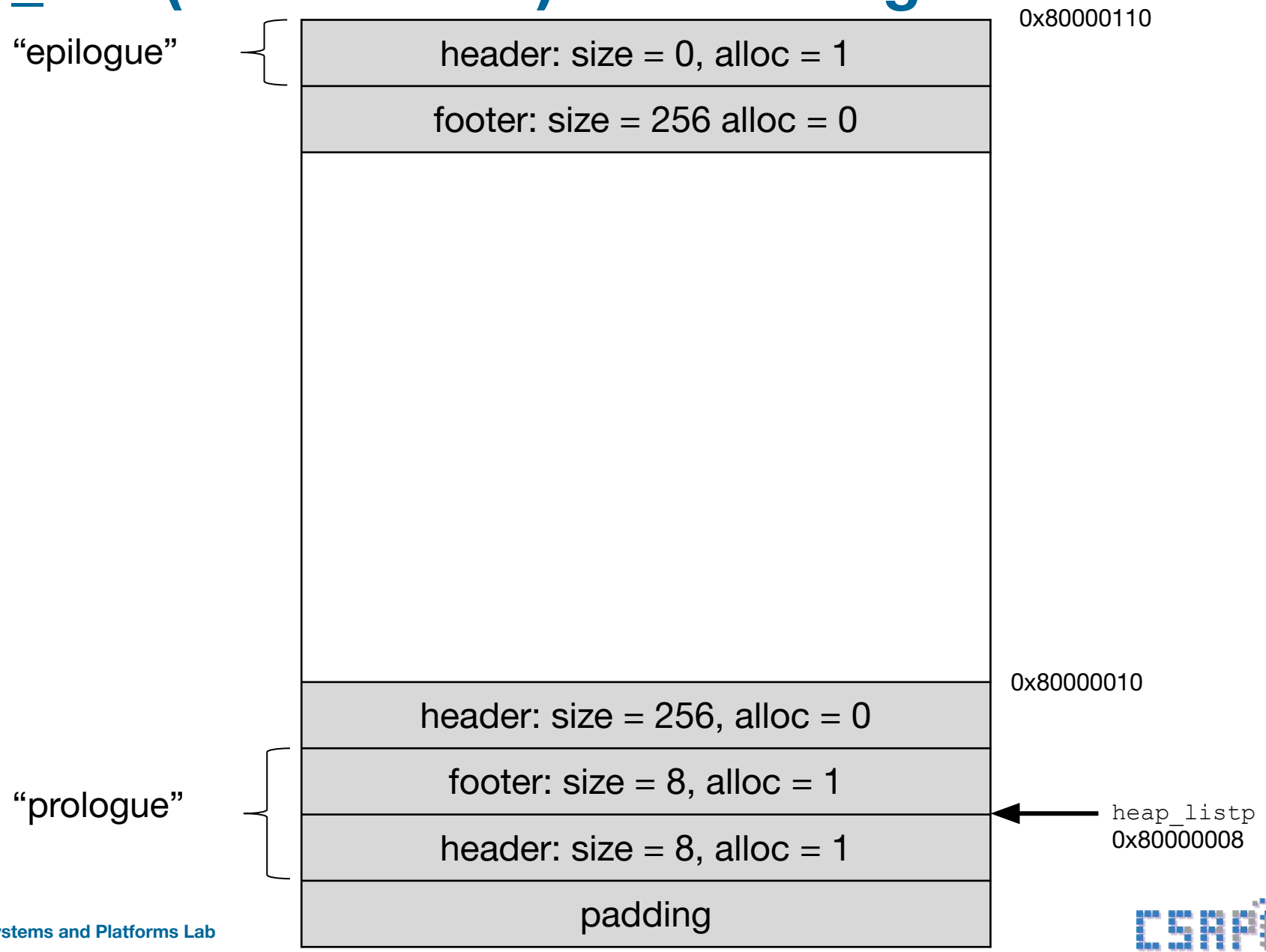
mm_free(0x80000010)



mm_free(0x800000b8)



mm_free(0x800000c0): coalescing



`mdriver` and traces

Test your memory allocator

The Test Driver

- Build everything you need with `make`. **Note**: Provide your ID in `mm.c` beforehand or rebuild once you added it.
- Use the `-f` option at first to run individual traces. `short1-bal.rep` and `short2-bal.rep` are good starting points.
- Your allocator's performance will be evaluated as explained in the handout.
- Run `./mdriver -g` to get an idea of you performance score. It might be slightly different during our evaluation. Here is how it will look like:

```
Student: SysProg TA (2019-11111)
Using default tracefiles in ./
Perf index = 50 (util) + 10 (thru) = 60/100
correct:15
perfidx:60
```

Trace Files Format

```
20000          Simulated Heap Size
6                 Total number of allocations
8                 Total number of distinct
  operations
1                 // Unused
a 0 2040       allocate 2040 bytes for block 0
a 1 2040       allocate 2040 bytes for block 1
f 1            free block 1
a 2 48         allocate 48 bytes for block 2
a 3 4072       allocate 4072 bytes for block 3
f 3            free block 3
a 4 4072       allocate 4072 for block 4
a 5 4072       allocate 4072 for block 5
```

Q&A until 21:00

Good luck!