A practical course on

Advanced systems programming in C/Rust

Dimitrios Stavrakakis

Today's topic!

Memory Management

Outline

- Memory organisation
  - Physical, virtual memory

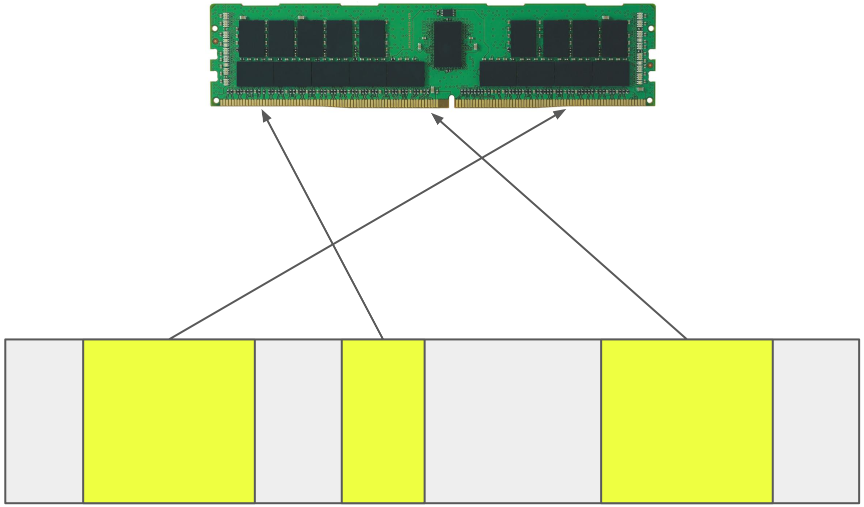
- Virtual memory management
  - paging, segmentation
  - heap, stack

- Memory allocation
  - sbrk(), malloc(), free(), mmap()
  - selection & allocation policies
  - fragmentation

Physical & Virtual memory

- Physical memory:
  - The actual DRAM device
  - Byte-addressable directly accessed
  - Limited in size

- Virtual memory:
  - Memory management technique
  - Convenience to the programmers
  - Managed by the OS
  - Address translation required



Virtual memory of a process

Virtual address translation

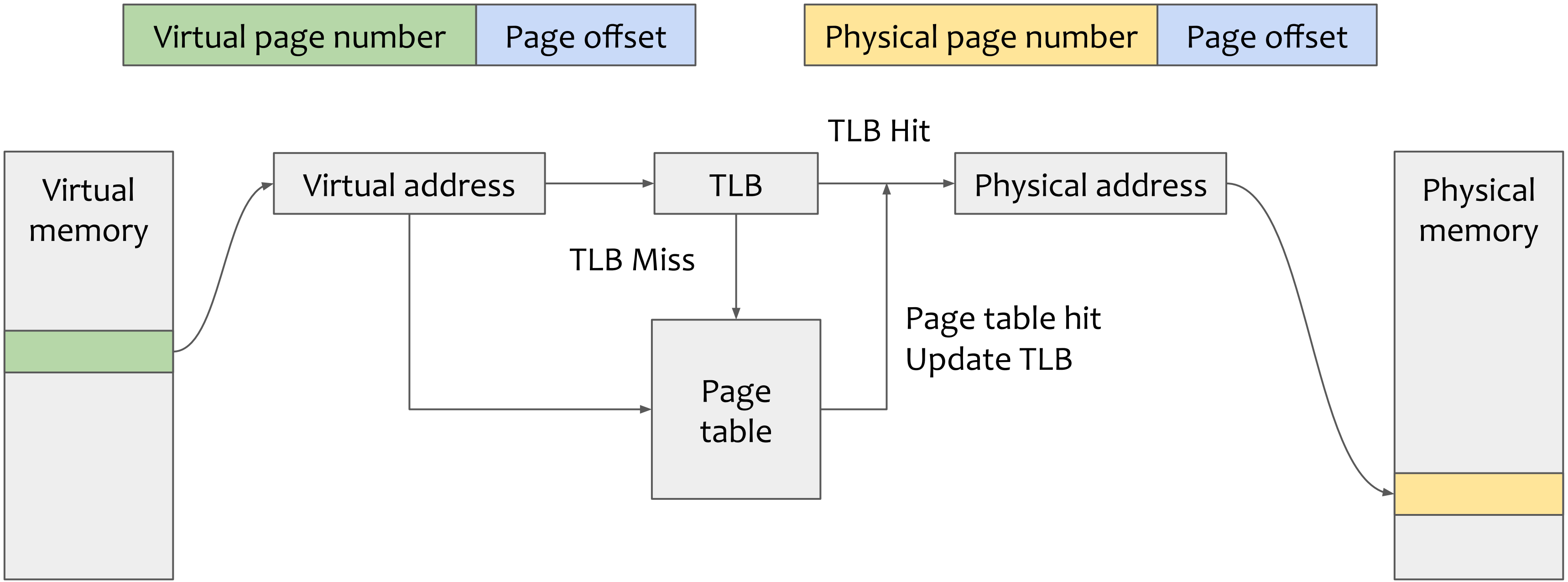
- Virtual address space is not mapped 1 by 1 to physical memory

- Need for address translation

- 2 most common approaches:
  - Paging
  - Segmentation

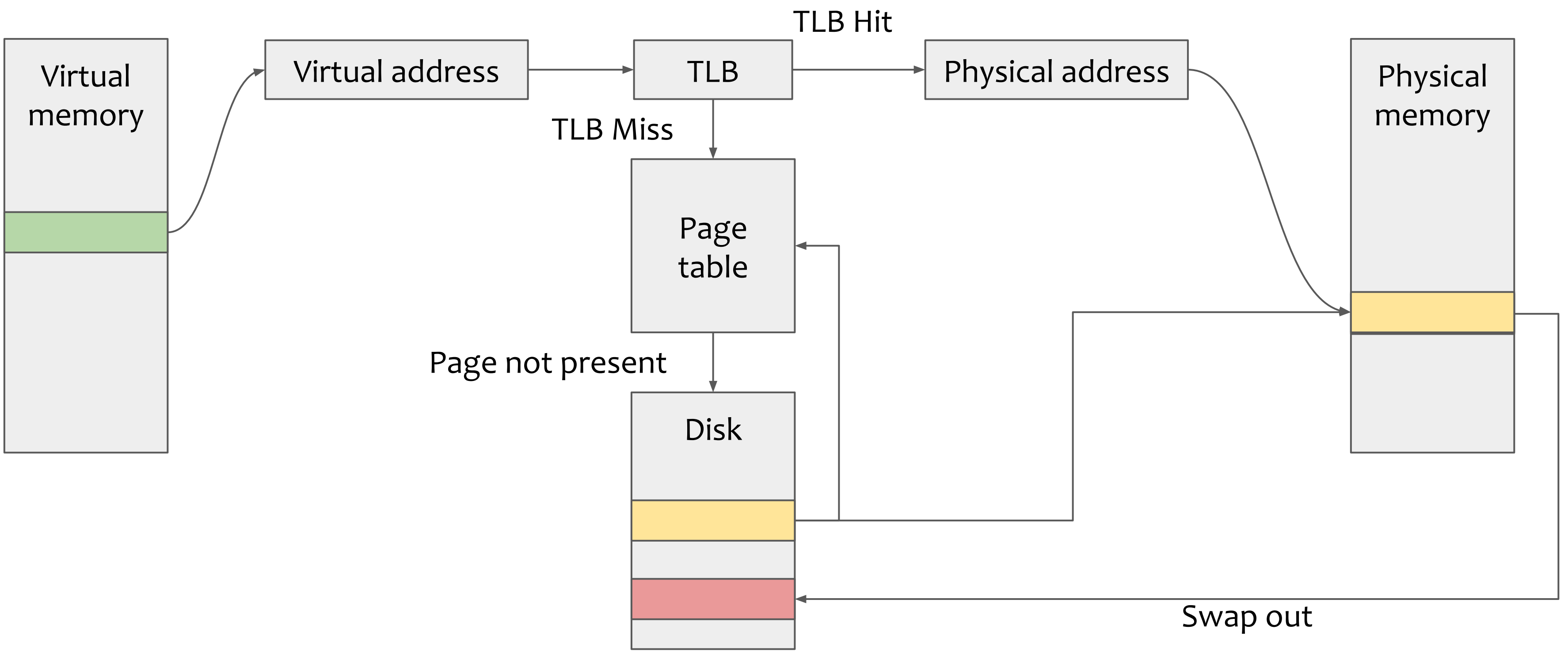
Paging

- Virtual address space of a process is mapped to the physical memory in pages
  - Typical page size is 4KB
- How address translation lookup process works:



Page faults

- What happens when the requested page is not found during the translation?
  - Page fault! - potentially expensive I/O operation



Segmentation

- Virtual address space is divided into segments of variable length
  - Each segment represents different logical address space of the program

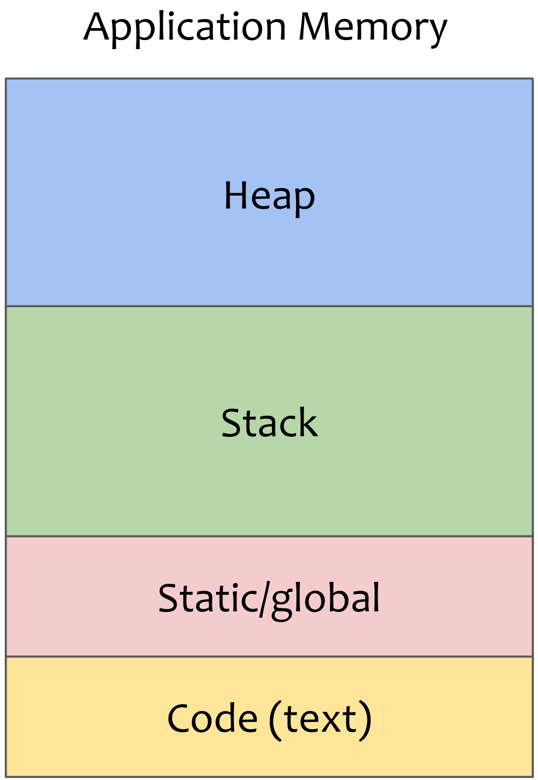
- Segments of a process are loaded into memory at run time - not necessarily contiguously

- Similar translation with paging
  - Segment table instead of page table

Segment ID	Segment size	Physical memory address
0	100	400
1	200	800
2	100	600

Stack & Heap

- Stack
  - Contiguous block of memory
  - Allocation & deallocation performed by the compiler
  - Fixed size during run-time
- Heap
  - Heap memory can be allocated in any random order
  - Allocation & deallocation is a programmer's task
  - Heap can grow during the lifetime of an app

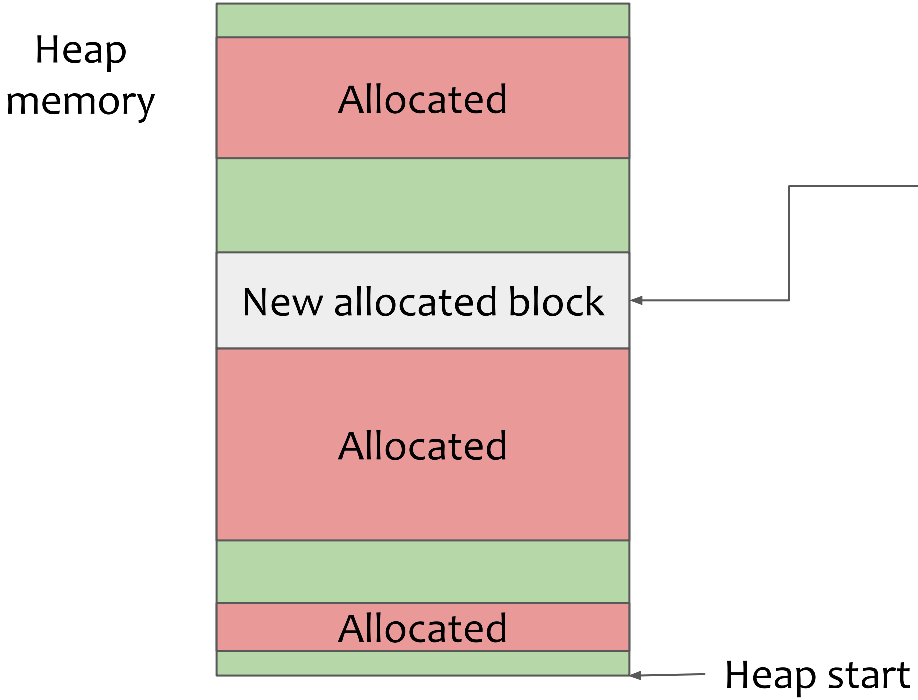


Heap memory allocation interface

- **sbrk**(intptr_t increment) :
  - Extends the heap region by the requested increment size
- **malloc**(size_t size) :
  - Allocates a memory block of <u>at least</u> the requested size
- **realloc**(void *ptr, size_t size)
  - Reuse the same block if size fits otherwise allocate a new block and copy the data
- **free**(void *ptr) :
  - Mark the occupied block pointer by ptr as free & reusable
- **mmap**() - **munmap**()
  - Maps files or devices directly into virtual memory address space
  - Uses demand paging
  - Byte-to-byte correlation

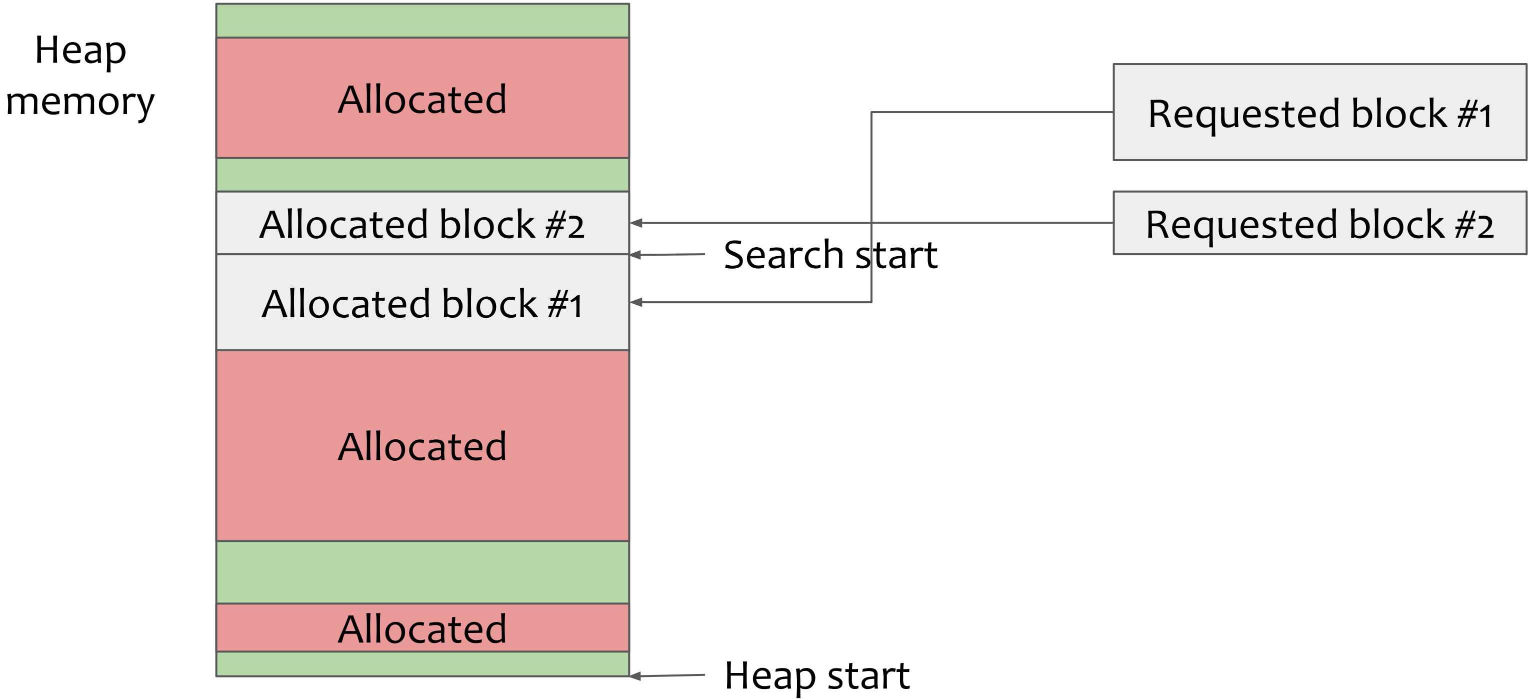
Selection policies

- **First fit:**
  - Commence the search from the start address of the heap
  - Allocate the first block that can accommodate the requested memory size



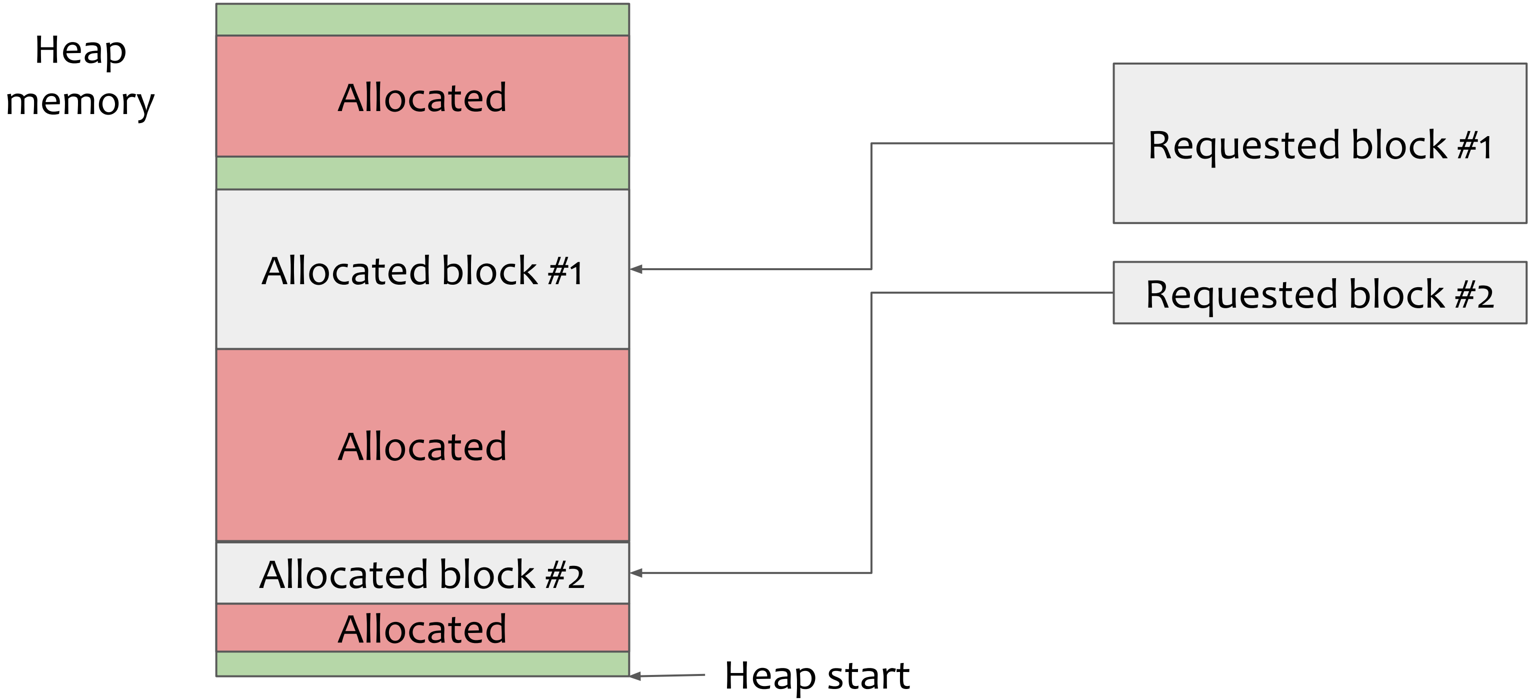
Selection policies

- **Next fit:**
  - Commence the search from the address of the most recently allocated block
  - Allocate the first block that can accommodate the requested memory size



Selection policies

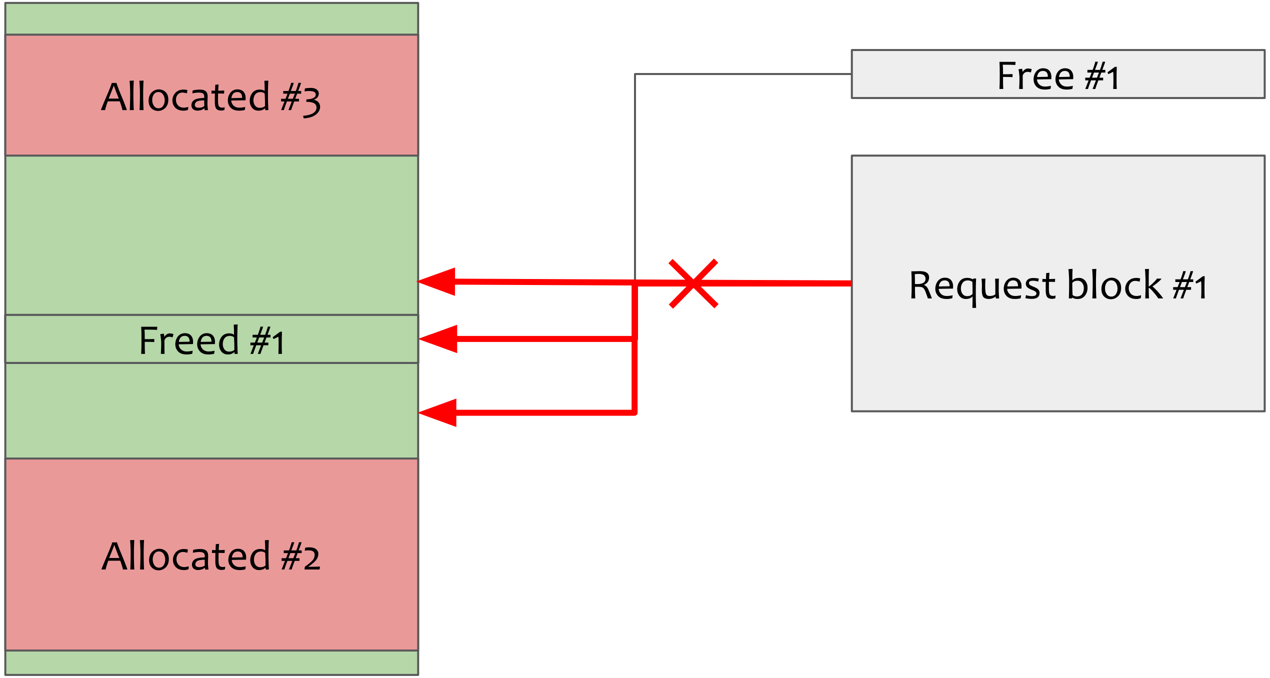
- **Best fit:**
  - Commence the search from the start address of the heap
  - Allocate the smallest free block that can accommodate the requested memory size



Fragmentation

External fragmentation problem

Heap memory

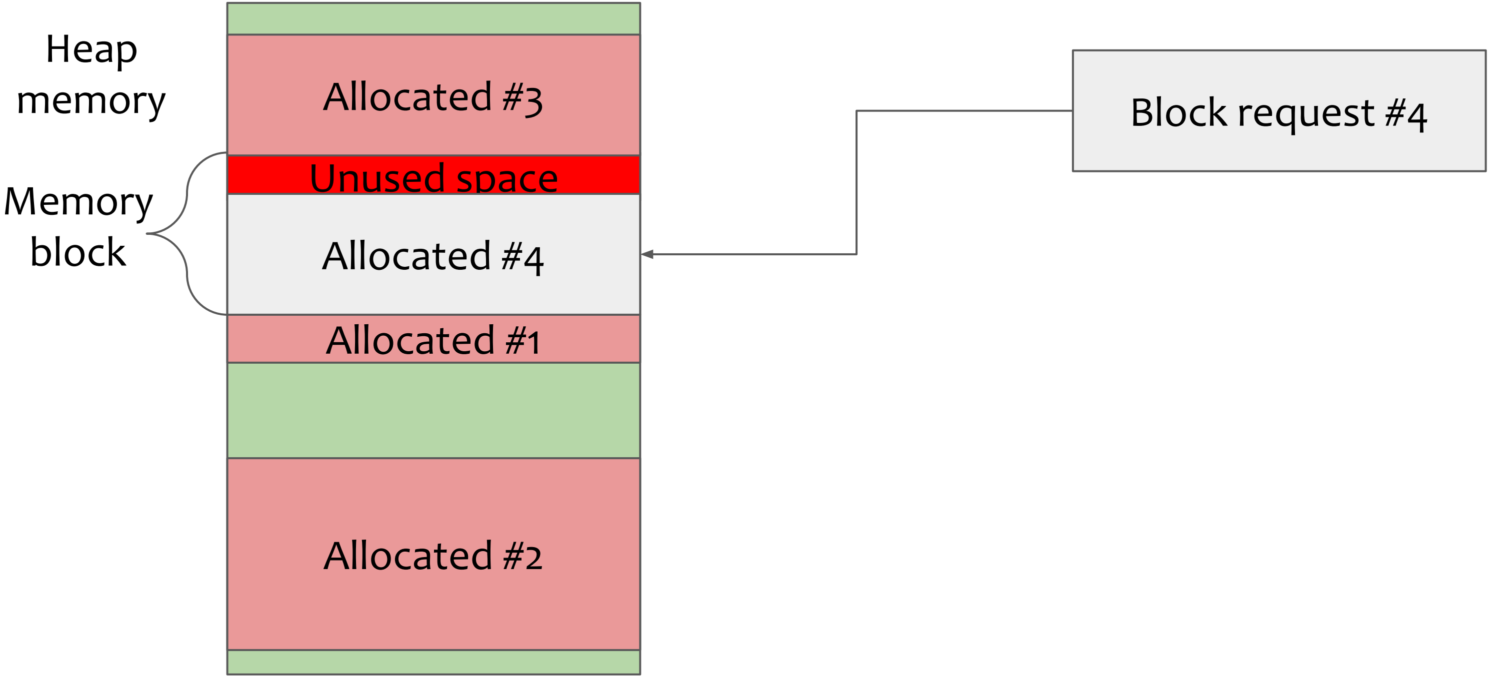


There is enough memory space, but not in a single block

Coalescing: merging adjacent free blocks

Fragmentation

Internal fragmentation problem



Internal fragmentation: depends on the allocation strategy

Possible allocation strategies

- Free Bitmaps
  - The state of a memory block is represented by a bit in a bitmap
- Free lists
  - Linked list that maintains the location and size of the free memory blocks
- Segregated fits
  - Maintain separate free lists of size classes referring to different sized blocks
- Buddy system allocation
  - Each block may be split into two sub-blocks (buddies) being power of 2 in size

Assignment

Tasks:

- Implement your own heap allocation functions
  - malloc(), realloc(), free(), calloc()
- Make use of different allocation policies to lessen fragmentation and efficiently use the allocated heap size
  - first_fit, next_fit, best_fit
- Extend your allocator to make it suitable for multi-threading environment

Useful tools:

- Valgrind, AddressSanitizer, ThreadSanitizer

Thank you for listening!

See you in the Q&A session