Memory Management

Instructor: Linyi Li
Slides adapted from Dr. B. Fraser

Topics

1) What is the layout of memory?
2) How does the heap work?

a) Getting space from the OS
b) Tracking free space
c) Freeing allocated space

Context

- Memory allocation / deallocation
  - Heap is used for dynamically allocated memory.
    - Usually use: malloc() or calloc(), and free().
  - How could we actually implement malloc() / free()?

  (This will help us really understand low-level memory management)

- *We are not talking about physical memory here.*
  User processes can only use virtual memory, not physical memory.

Details

- Can find more info in OSTEP book
  (more depth than we require)
  - Chapter 13 The Abstraction: Address Spaces
    https://pages.cs.wisc.edu/~remzi/OSTEP/vm-intro.pdf
  - Chapter 14 Interlude: Memory API
    https://pages.cs.wisc.edu/~remzi/OSTEP/vm-api.pdf
  - Chapter 15 Free-Space Management
    https://pages.cs.wisc.edu/~remzi/OSTEP/vm-freespace.pdf

Prerequisites

What you already know

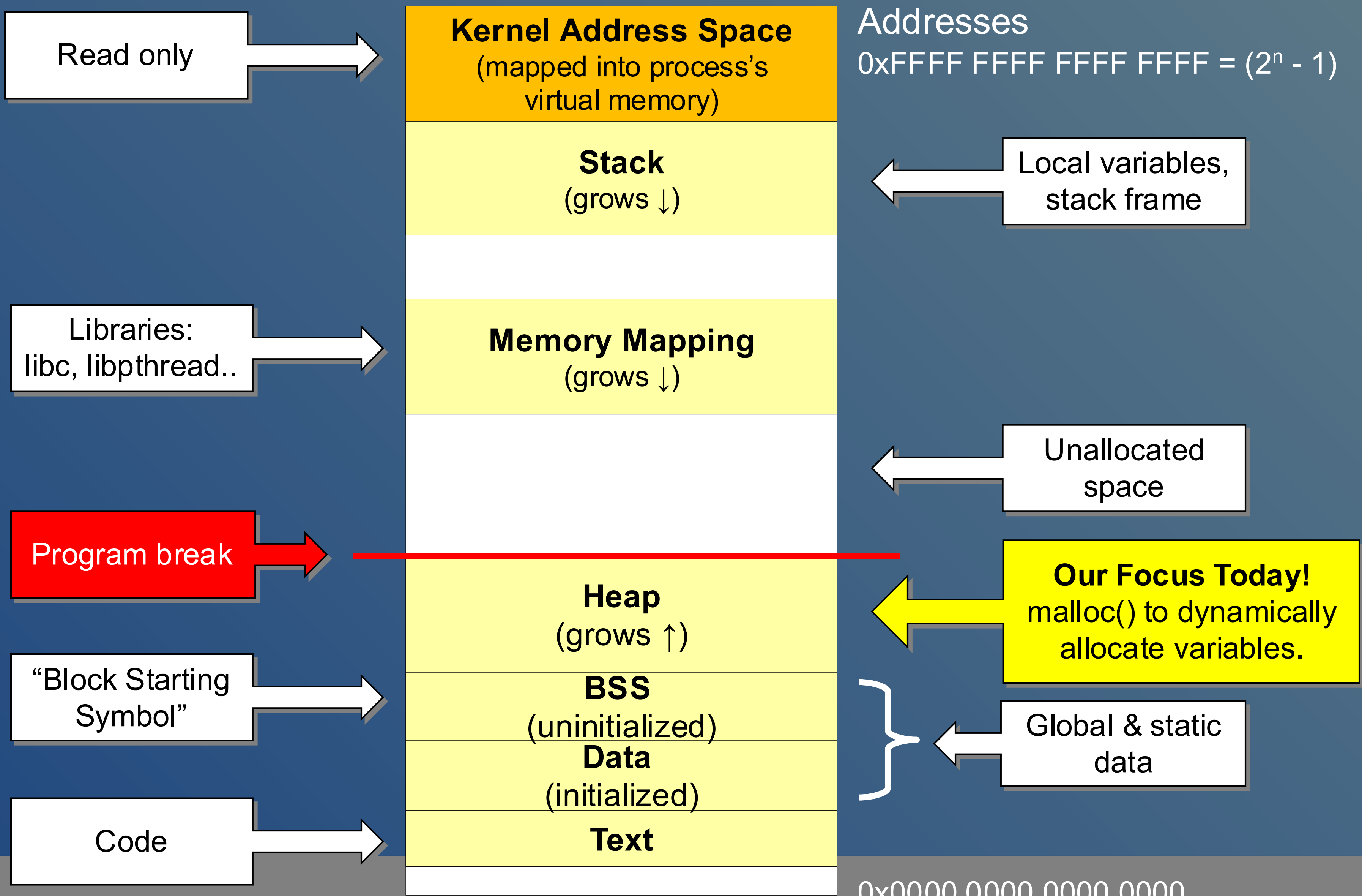
- This lecture assumes you know:
  - Data structures used for memory management: array, struct, linked lists
  - Able to use and understand malloc() and free() in C.
  - How to implement a singly- and doubly-linked list in C.
  - The stack and the heap:
    - How a program's variables use stack and heap in C
    - How variables are placed in the stack and heap.

Linked Lists

```c
struct Node {
   int data;
   struct Node *next;
};

// Create a new node with the given data
struct Node *createNode(int data) {
   struct Node *newNode
      = malloc(sizeof(*newNode));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}


// Insert a new node at the end of list
void append(struct Node **head, int data) {
   // Code together!
}

// Traverse and print the linked list
void traverse(struct Node *head) {
   // Code together!
}
```

```c
int main() {
   struct Node *head = NULL;

   // Append elements to the list
   append(&head, 1);
   append(&head, 2);
   append(&head, 3);

   // Traverse and print the list
   printf("Linked List: ");
   traverse(head);

   // Remember: free memory when done
   struct Node *current = head;
   while (current != NULL) {
      struct Node *temp = current;
      current = current->next;
      free(temp);
   }
   head = NULL;

   return 0;
}
```

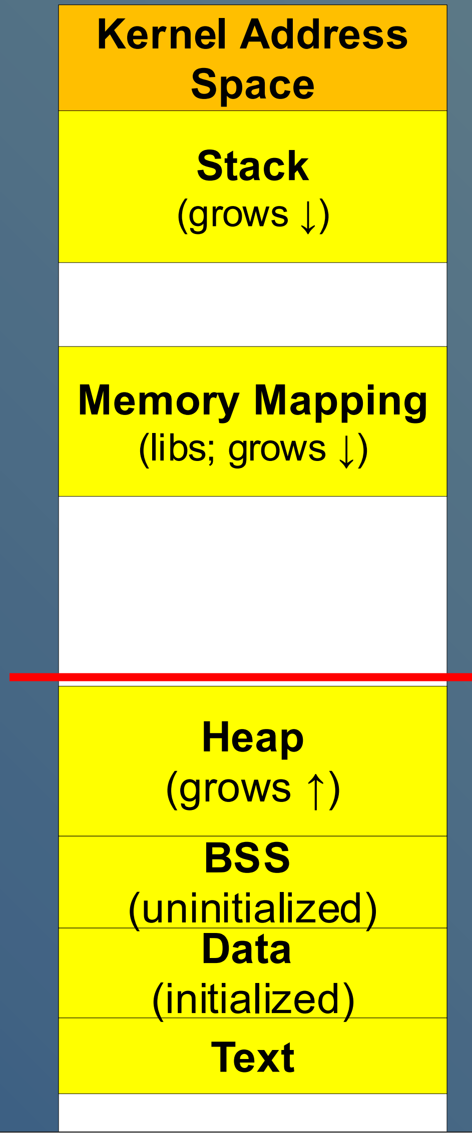
Memory Layout



- brk() and sbrk()

Getting More Memory

- Program Break
  - Used by Linux to mark end of heap (actually end of BSS; but grows to be heap)
  - Above the Program Break is unallocated space.
- More Space
  - OS moves the program break higher to expand the heap
  - Linux uses brk() and sbrk() to move the program break.



man sbrk

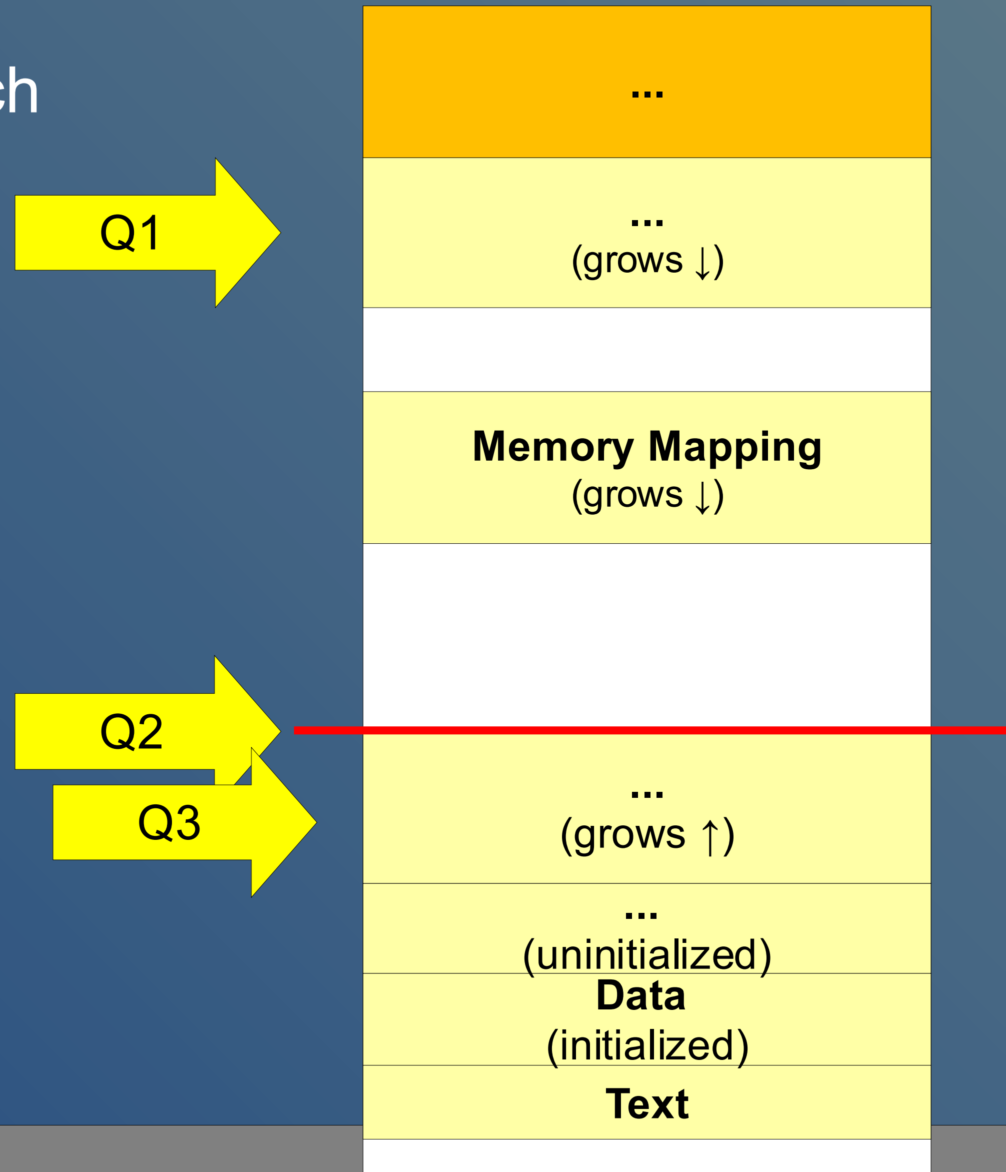
- man sbrk
  - OS increases size heap.
  - It's a syscall: overhead!
- Don't call sbrk() often
  - malloc() (user-level) calls sbrk() (kernel) to..
    - get big block of memory
  - malloc()..
    - hands out small pieces of memory for each request.
- How can malloc() do that?
  - Allocation strategies!
  - Deallocation strategies!

```
brk(2)                    System Calls Manual                    brk(2)

NAME
       brk, sbrk - change data segment size

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>

       int brk(void *addr);
       void *sbrk(intptr_t increment);

DESCRIPTION
       brk() and sbrk() change the location of  the  pro‐
       gram break, which defines the end of the process's
       data segment (i.e., the program break is the first
       location  after  the end of the uninitialized data
       segment).  Increasing the program  break  has  the
       effect  of  allocating  memory to the process; de‐
       creasing the break deallocates memory.

       brk() sets the end of  the  data  segment  to  the
       value  specified  by addr, when that value is rea‐
       sonable, the system has  enough  memory,  and  the
       process does not exceed its maximum data size (see
       setrlimit(2)).

       sbrk()  increments the program's data space by in‐
       crement bytes.  Calling sbrk() with  an  increment
       of  0  can be used to find the current location of
       the program break.

RETURN VALUE
       On success, brk() returns zero.  On error,  -1  is
       returned, and errno is set to ENOMEM.

       On  success,  sbrk()  returns the previous program
       break.  (If the break  was  increased,  then  this
```

ABCD: Memory Layout

- What is the name of each memory segment?

(a) BSS
(b) Heap
(c) Program Break
(d) Stack



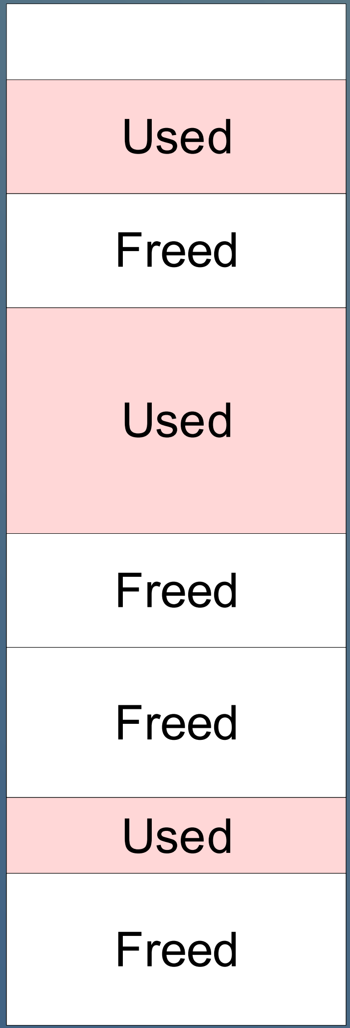
Managing Dynamic Memory

Overview

Memory Allocator

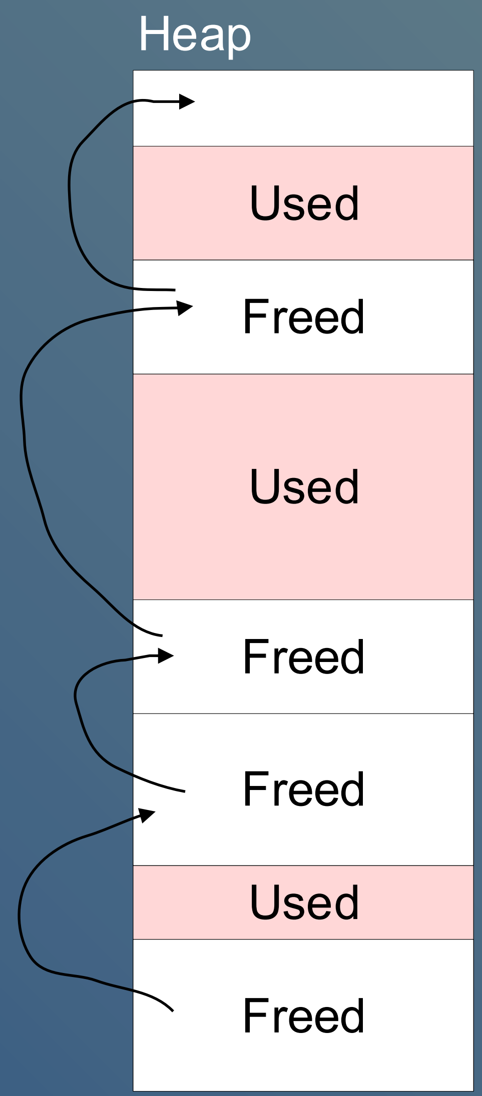
- Memory Allocator: manages the heap
  - For each allocation request,
    .. it returns a pointer to an unused (or *free*) region inside the heap.
  - It tracks of which parts of the heap are not used.
- Fragmentation
  - Over time the application allocates and frees memory regions.
  - This fragments memory into
    .. broken up pockets of used and freed memory.

Heap

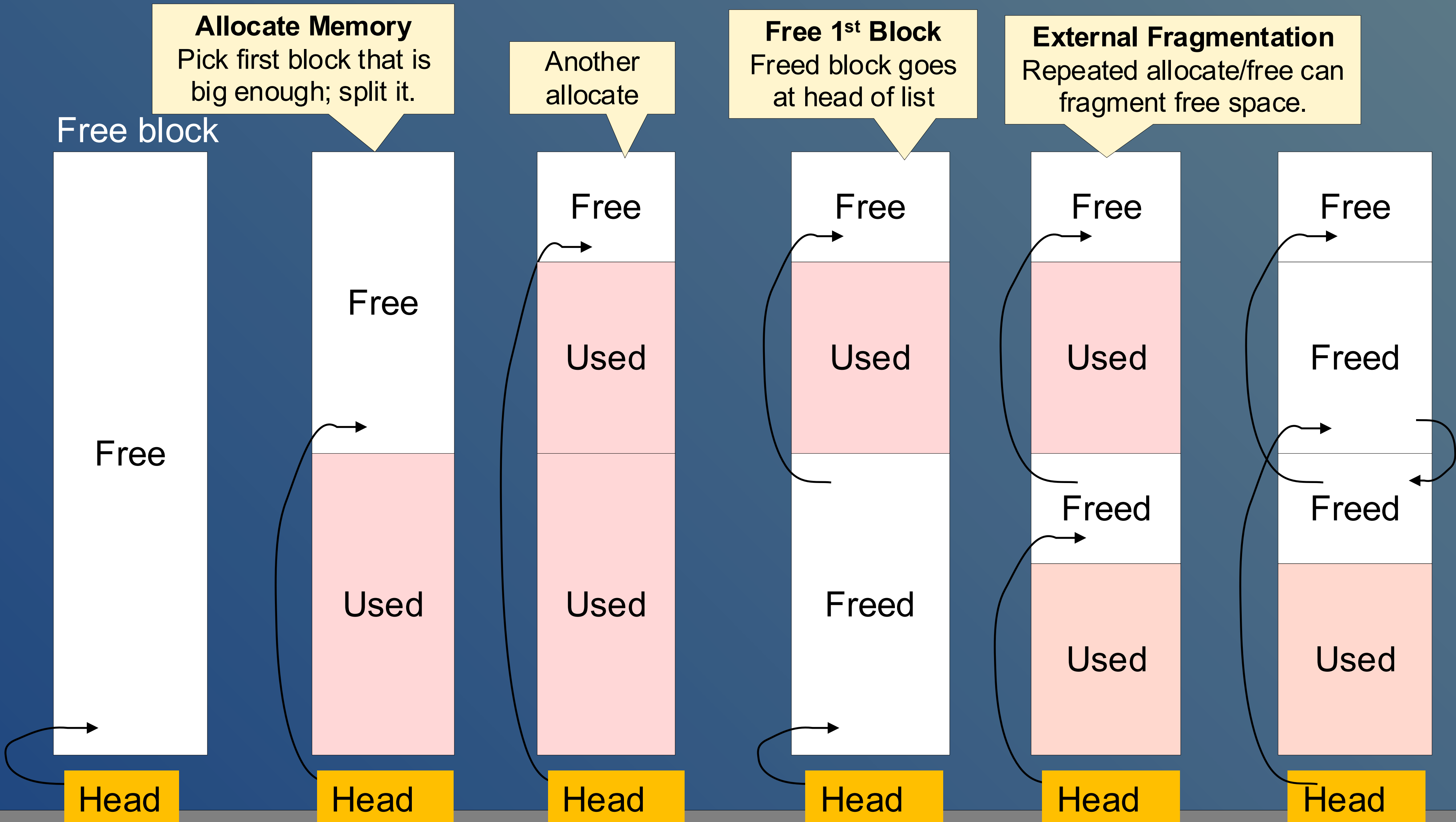


Track Free Space

- Track free regions (blocks) in ..a linked list of free blocks.
  - We don't track used regions; we are given back regions from calls to free().



Linked List Management



Linked List Management

- Free Blocks Linked List
  - We have a linked list of free blocks.
  - .. Head points to the most recent free block.
- Basics of Allocation - malloc()
  - .. Pick a free block from the linked list.
  - Remove it from the linked list.
  - Split the free block into two blocks: allocated and free.
  - Insert the new free block back into the *head* of the linked list.
  - Return the allocated block to the caller.
- Basics of Deallocation - free()
  - Inserting the given block at head of the linked list.

Linked-list Without Dynamic Allocation

- Linked List of Free Memory

–We've see how to manage free memory using a linked list of free blocks.

–But, how do we normally create nodes in a Linked List?

.. Dynamic allocation!

–So, how do we create a linked list without dynamic allocation?

- In-Place linked List

–..Create a header on each free block

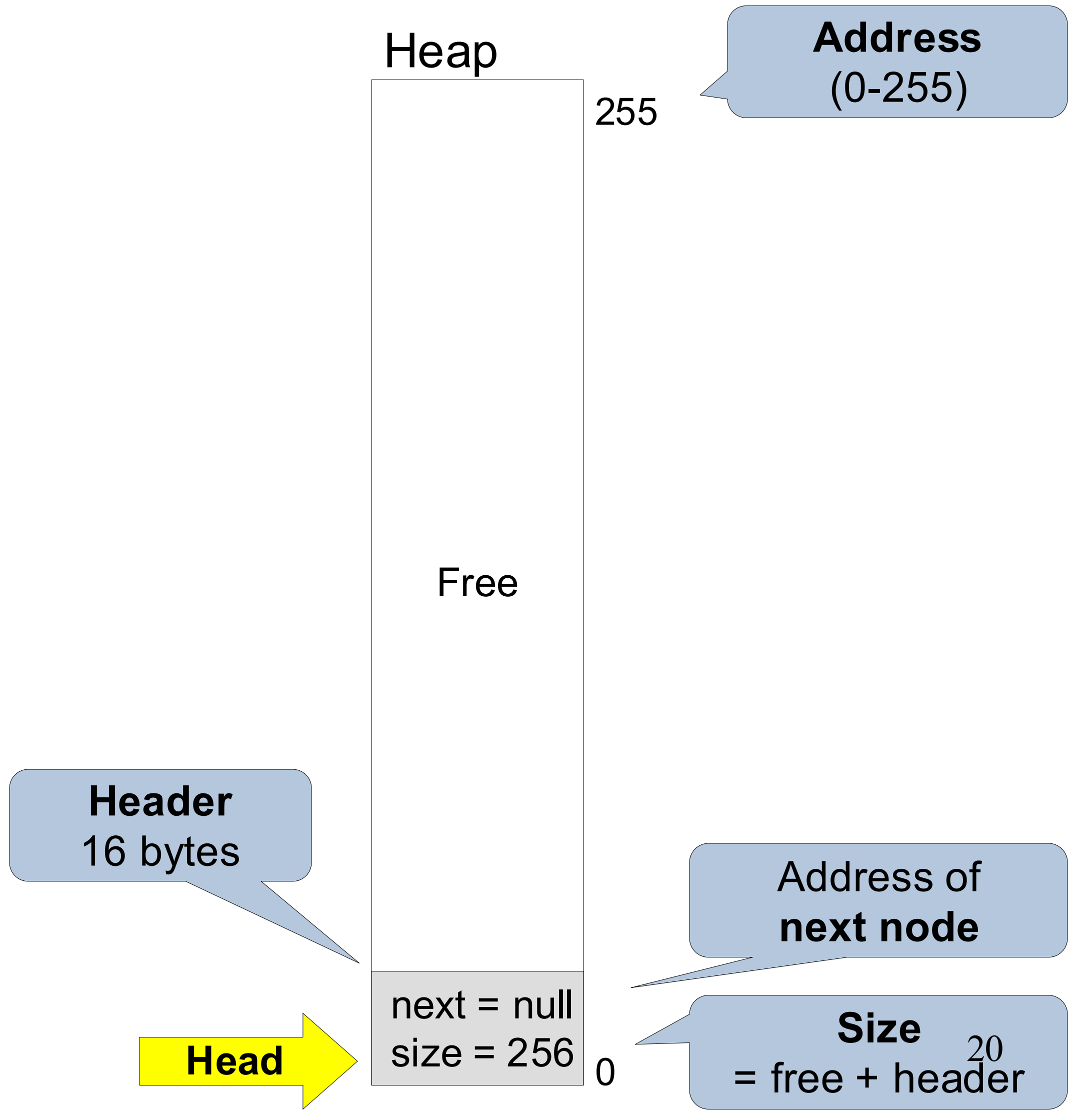
to track size of the block and pointer to next free block

–Perform coalescing:

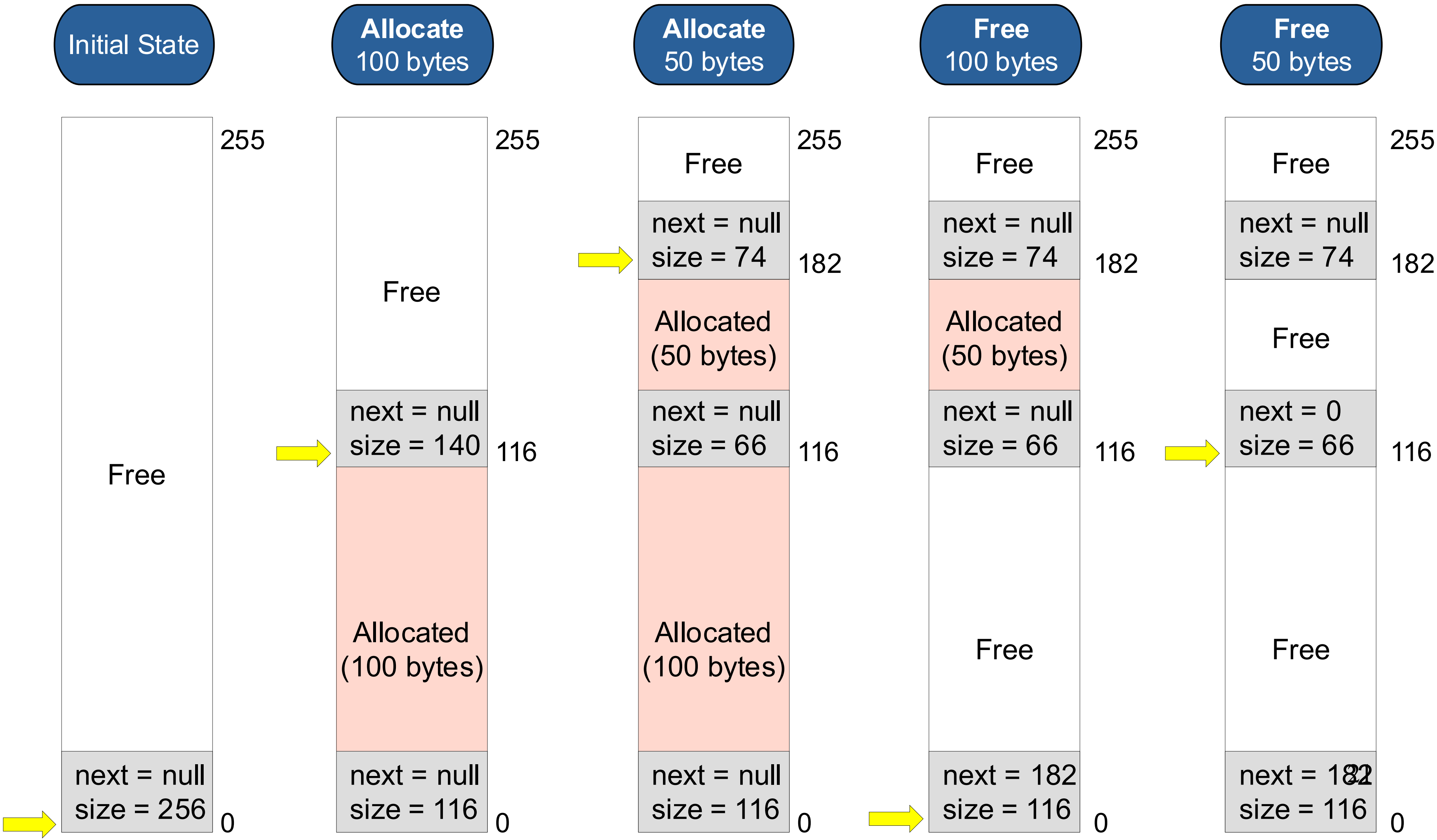
combines consecutive free blocks into a larger single free block.

In-Place Linked List

- Example with the heap size of 256 bytes.
- Build linked-list of blocks.
- Each free and allocated block has a header
  - Assume size and next are 8 bytes each.



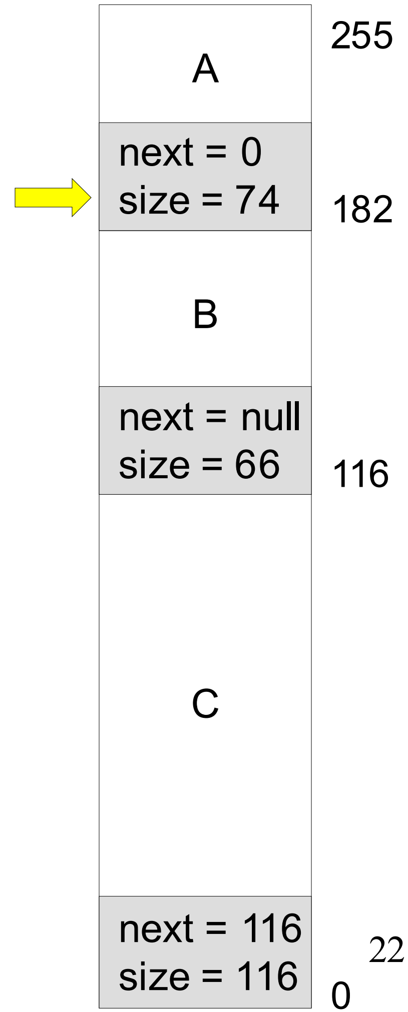
Example: In-Place Linked List



ABCD: Linked List

- What was the order in which these blocks were freed?
(Listed in order of first freed to last freed)

(a) A then B then C
(b) A then C then B
(c) B then C then A
(d) C then B then A

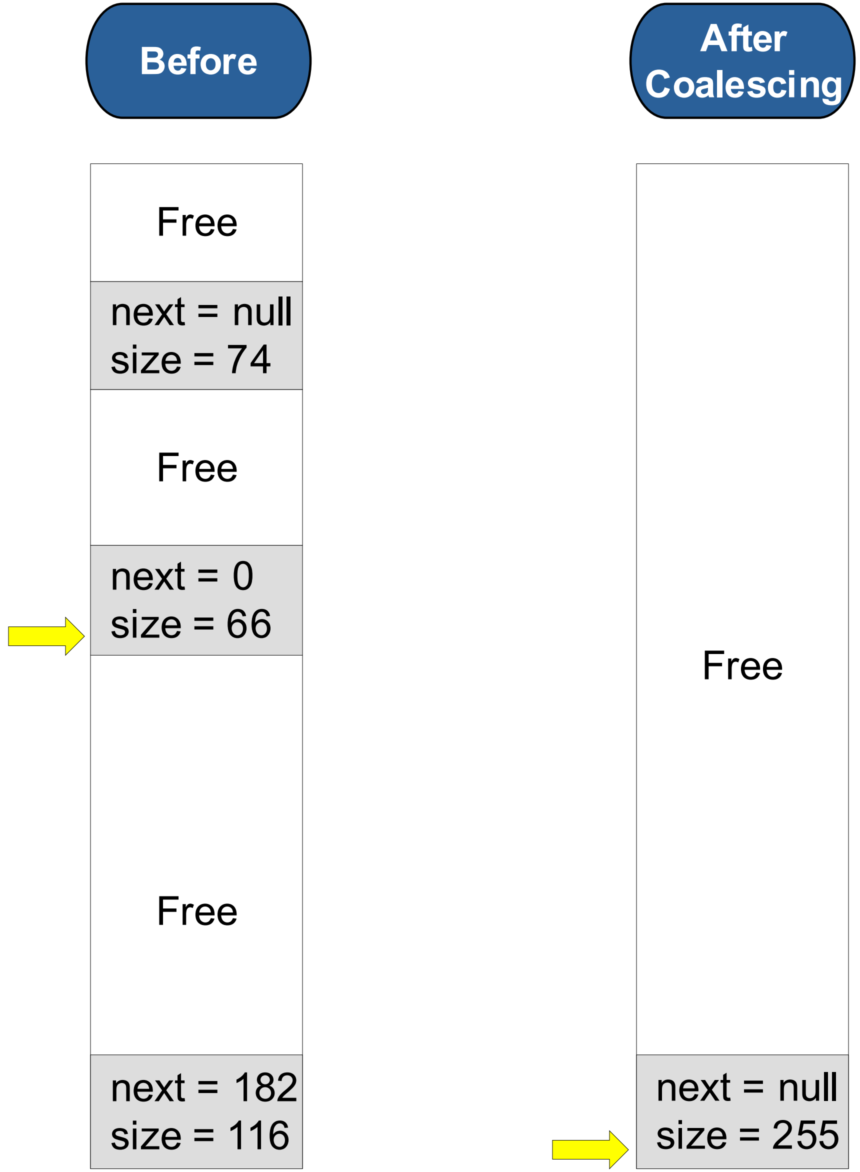


External Fragmentation

- External Fragmentation
  - .. Free memory is fragmented into smaller blocks.
  - But each allocation request can only be satisfied by a single block (cannot split it up).
  - Even if total free memory is enough, may not have one contiguous free block to satisfy an allocation request.
  ..
- Coalescing
  - Process of combining consecutive free blocks into bigger blocks.
- Internal Fragmentation
  - Similar problem of unused space inside blocks; More during virtual memory.

Coalescing

- Merge consecutive free blocks.

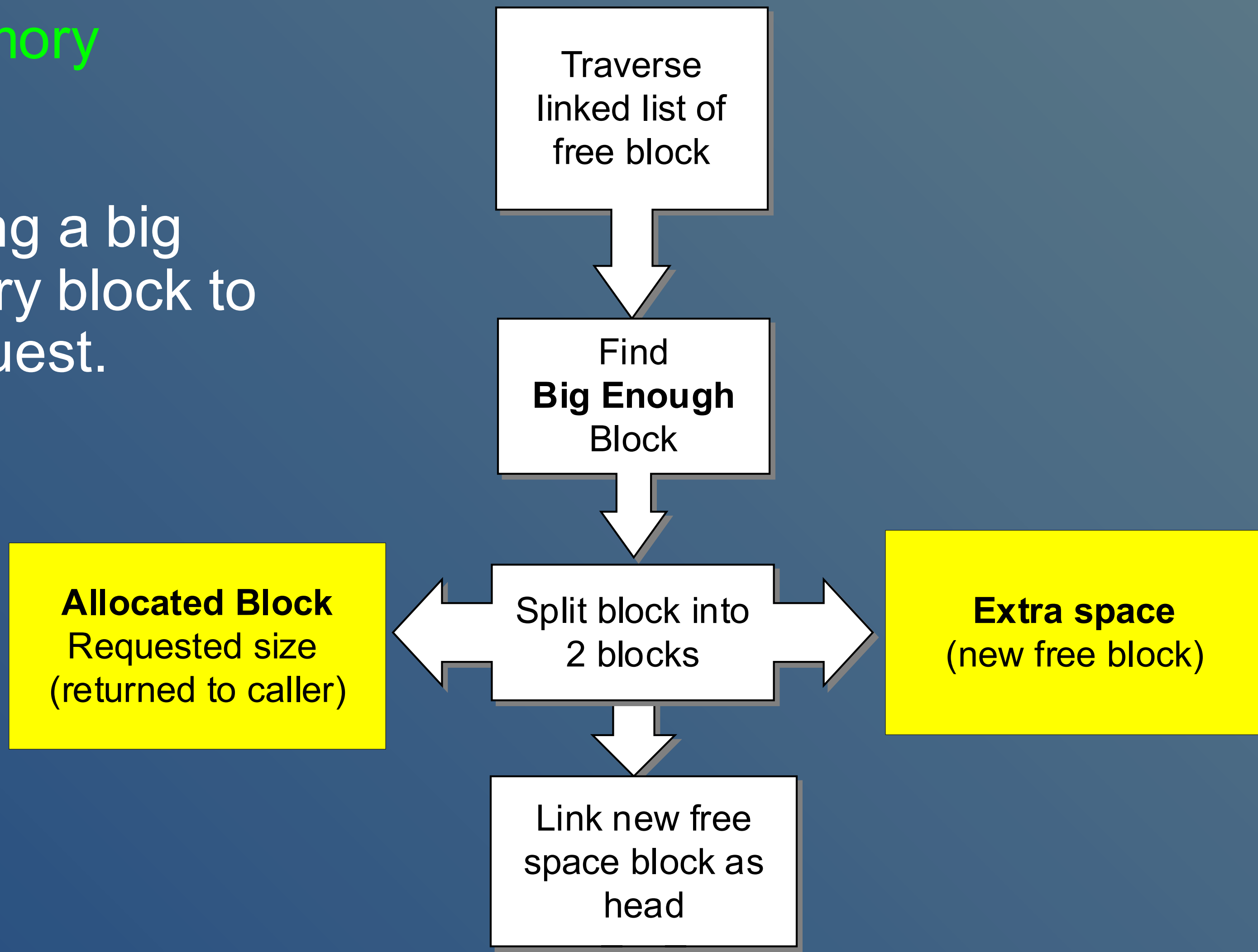


Finding a Free Block

Allocating Memory

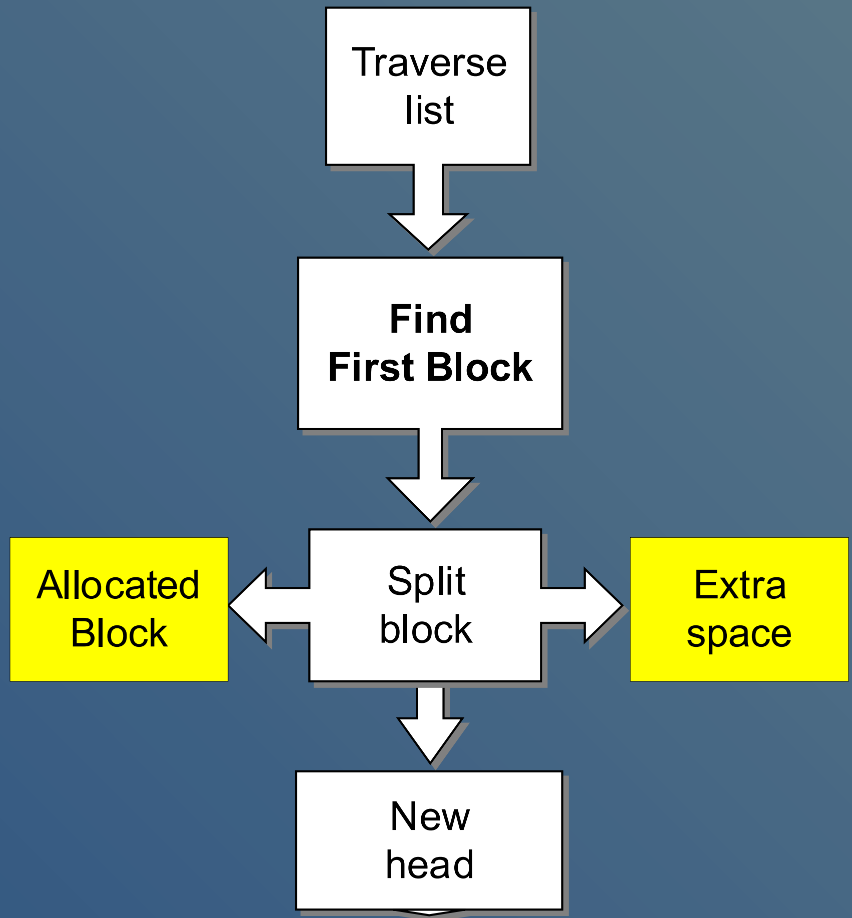
- Allocating Memory
  e.g., malloc()

  Requires finding a big enough memory block to satisfy the request.



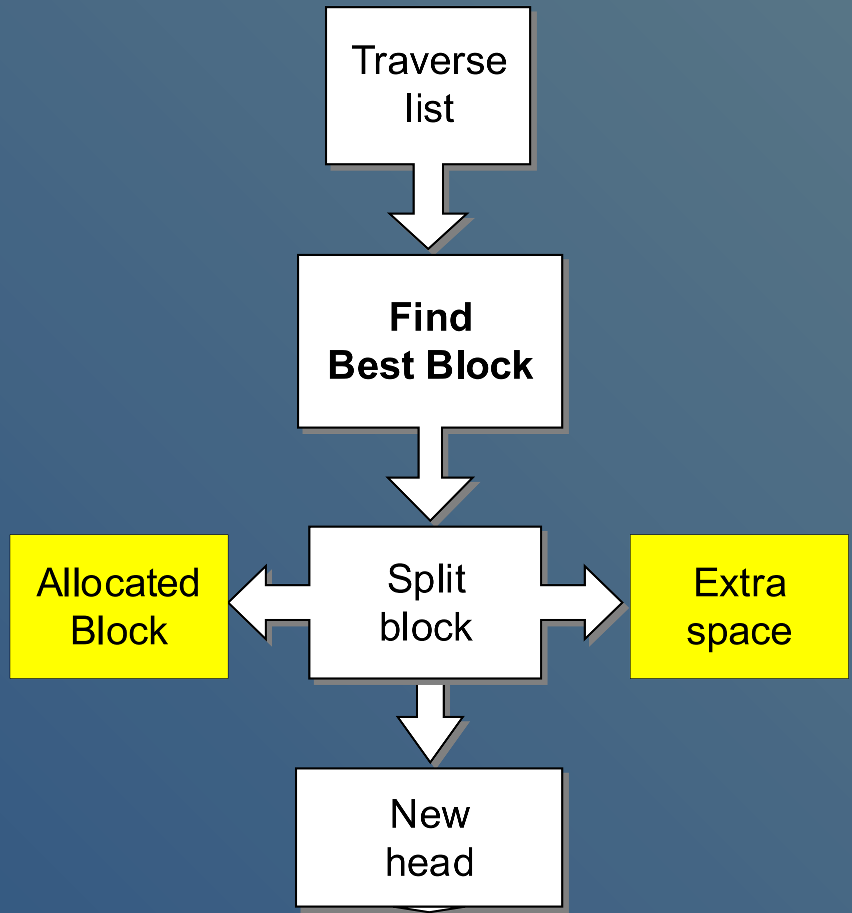
Allocating Memory: First Fit

- First-fit
  - .. Find the first block that is big enough.
- Advantage
  - implementation simplicity
  - fast: it only needs to find the first big enough block.
- Disadvantage
  - can pollute the beginning of the free list with small blocks
  - leads to more search time for bigger allocation requests.



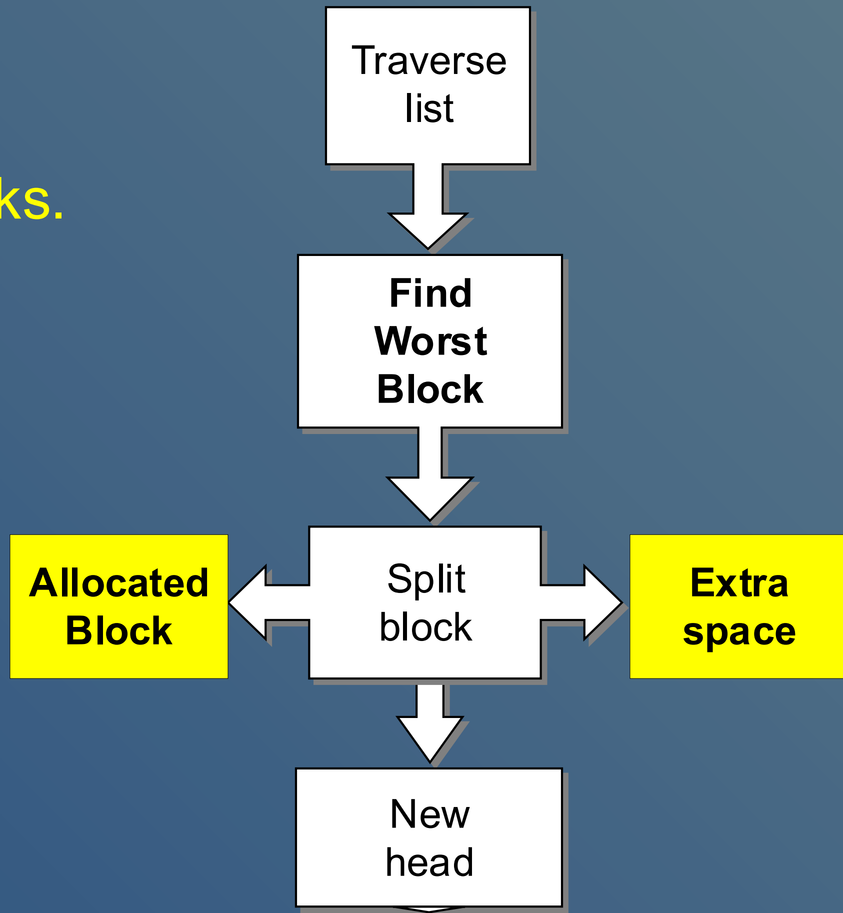
Allocating Memory: Best Fit

- Best-fit
  - Find the smallest free block that is big enough.
- Advantage
  - .. reduces wasted memory space.
- Disadvantage
  - **Speed**
    must search the entire list (unless ordered by size which has additional implementation complexity).
  - **Fragmentation**
    may create many small free blocks, leading to more chances of external fragmentation.



Allocating Memory: Worst Fit

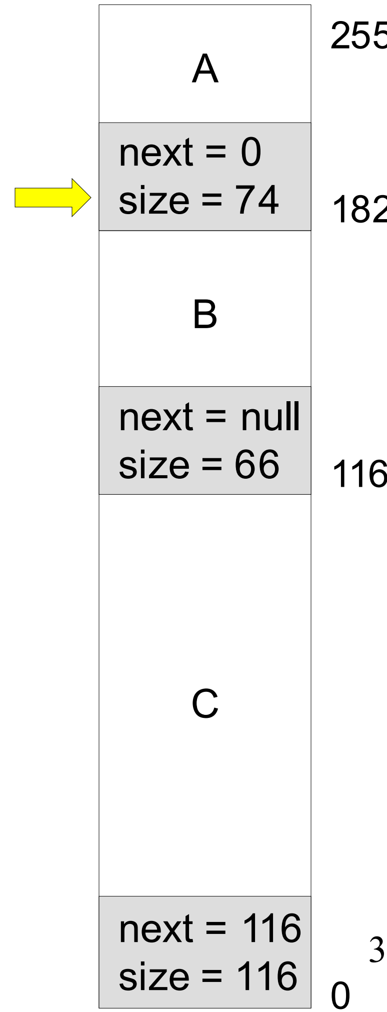
- Worst-fit
  - Find the largest free block.
- Advantage
  - .. produces large leftover free blocks.
- Disadvantage
  - must search the entire list.



ABCD: Free Space

- A memory allocation system is asked to allocate 50 bytes. Which block is allocated if it is using...
  - First fit
  - Worst Fit
  - Best Fit

(a) A
(b) B
(c) C
(d) None of them.



Summary

- Memory Segments
  - text, data, BSS, heap, memory mapped, stack, kernel.
  - Program break and effect of brk() and sbrk()
- Memory Allocator
  - Linked list of free memory
  - New free blocks go first in the list
- Fragmentation
  - External Fragmentation
  - Coalescing algorithm
- Block selection algorithms
  - (first, smallest, biggest) fit