Operating System Labs

Yuanbin Wu
CS@ECNU

Operating System Labs

- Project 2 Due
  - 21:00, Nov. 7

- Project 2
  - group of 3
  - You now have 3 "late days", but start early!
  - Oral test at week 12 (Nov. 22)
  - We encourage using **git** for your group assignments (geeks' choice).

Operating System Labs

- C Memory API
- Free Memory Management

C Memory API

- Type of memory
  - Stack
  - Heap

C Memory API

- Stack
  - Allocated / Deallocate automatically
  - By the compiler
  - Automatic memory

C Memory API

- Stack
  - Example (local variable)

```
void func()
{
        int x = 0;
    ...
}
```

  - You only declare the variable
  - Compiler will allocate it when someone calls the function
  - Also will deallocate it when func returns

C Memory API

- Heap
  - Allocated / Deallocate explicitly
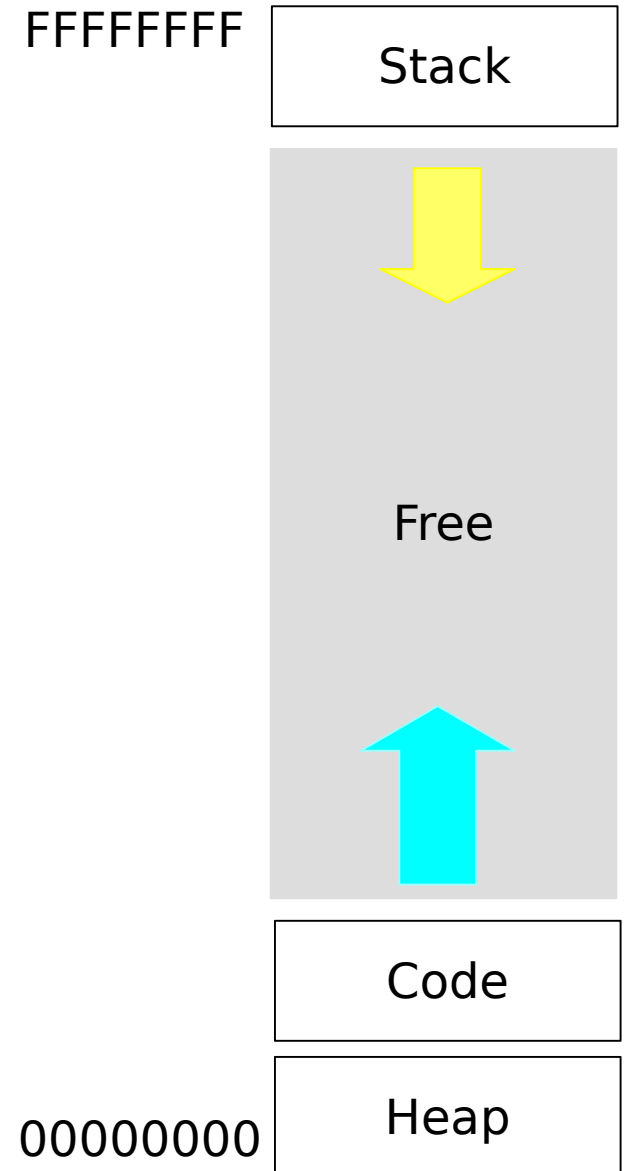 - By you, the programmer

C Memory API

- Heap
  - Example (malloc)

```
void func()
{
        int *ptr = (int*)malloc(sizeof(int));
    ...
}
```
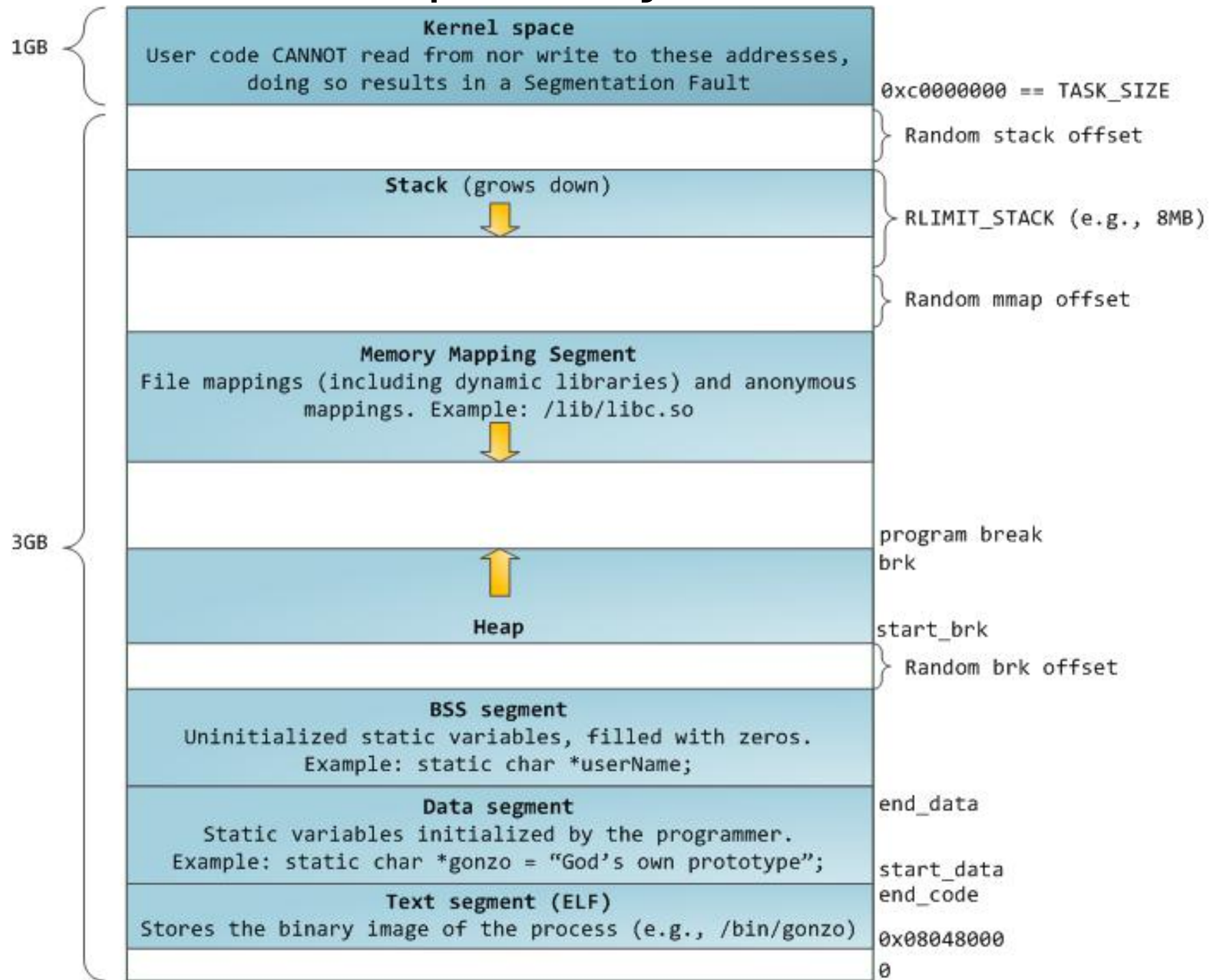
  - Both stack and heap allocation
  - When func returns,
    - Stack memory will be deallocated
    - Heap memory is still there

C Memory API

- **Stack and Heap**

    - Heap
        - From low addr to high addr
    - Stack
        - From high addr to low addr

- **Let's see**

FFFFFFFF

Stack

Free

Code

Heap

00000000

A real address space layout

Kernel space User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault	0xc0000000 == TASK_SIZE
	Random stack offset
Stack (grows down) ⬇	RLIMIT_STACK (e.g., 8MB)
	Random mmap offset
Memory Mapping Segment File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so ⬇	
	program break brk
⬆	
Heap	start_brk
	Random brk offset
BSS segment Uninitialized static variables, filled with zeros. Example: static char *userName;	
Data segment Static variables initialized by the programmer. Example: static char *gonzo = "God's own prototype";	end_data start_data
Text segment (ELF) Stores the binary image of the process (e.g., /bin/gonzo)	end_code 0x08048000
	0

- 1GB (Kernel space)
- 3GB (user space)

C Memory API

- ## Malloc

  ```
  #include <stdlib.h>
  void *malloc(size_t size);
  ```
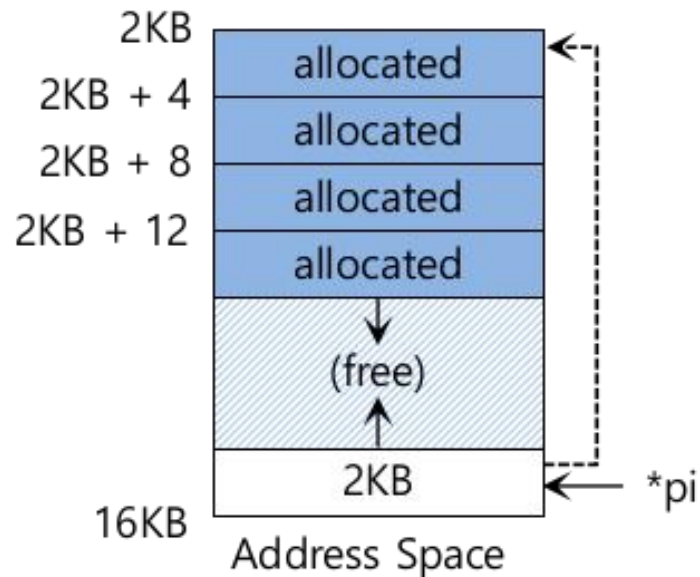
  - If failed, return NULL

- ## Free

  ```
  #include <stdlib.h>
  void free(void* ptr);
  ```
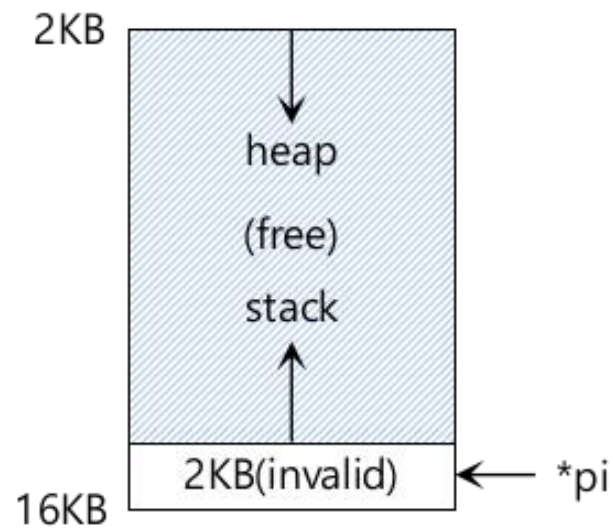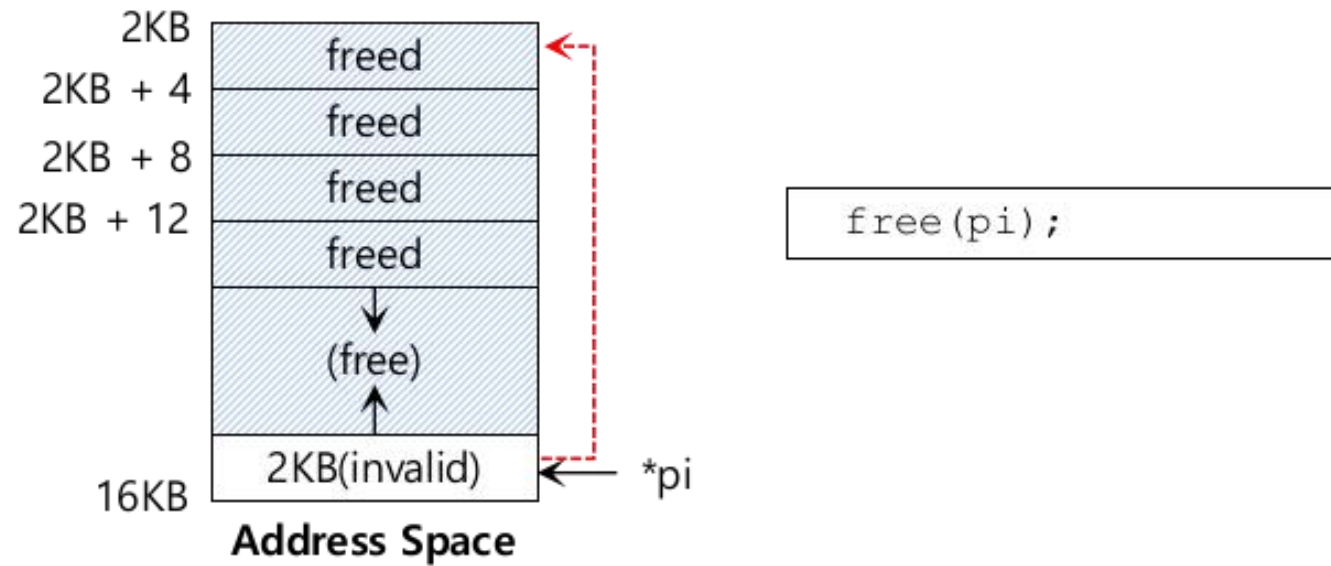
Allocation
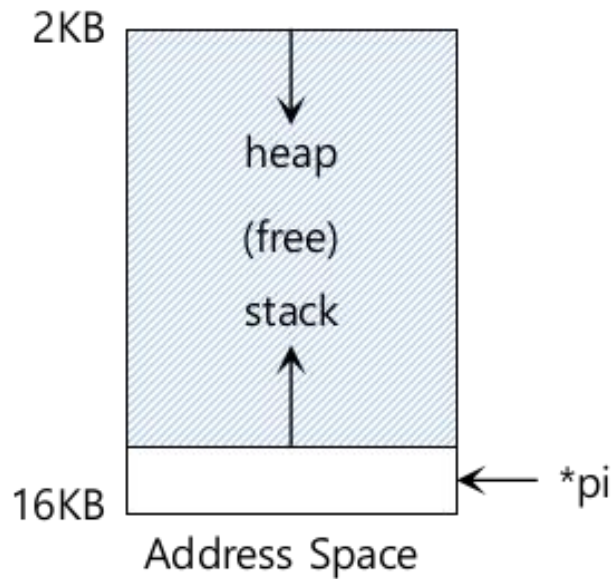
int *pi; // local variable

pi = (int *)malloc(sizeof(int)*
4);

Free


```
free(pi);
```

C Memory API

- Segment fault

```
char *src = "hello";
char *dst;
strcpy(dst, src);
```
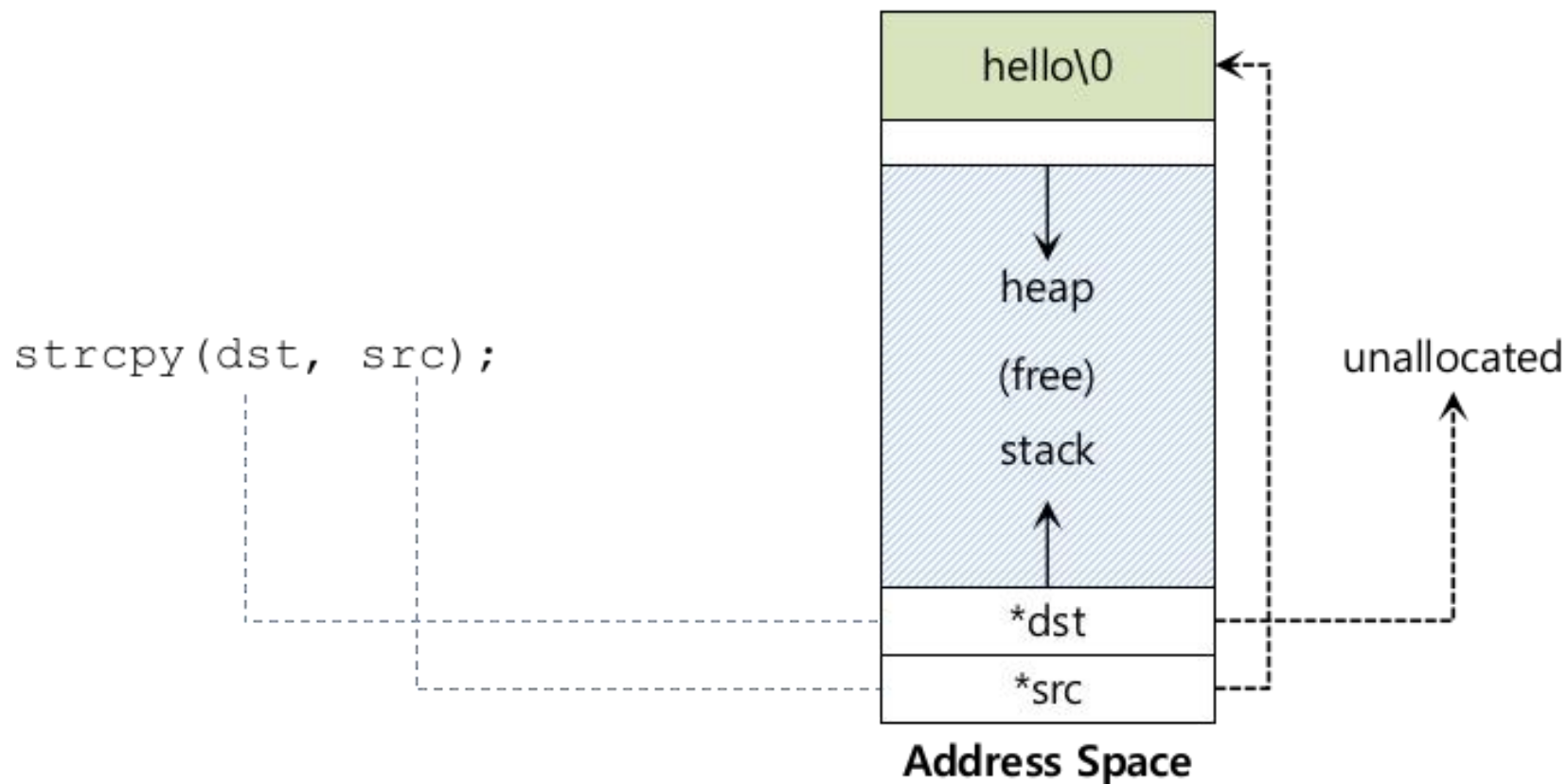
- run this code, it will likely lead to a

  segmentation fault

- It is a fancy term for

  YOU DID SOMETHING WRONG WITH MEMORY YOU
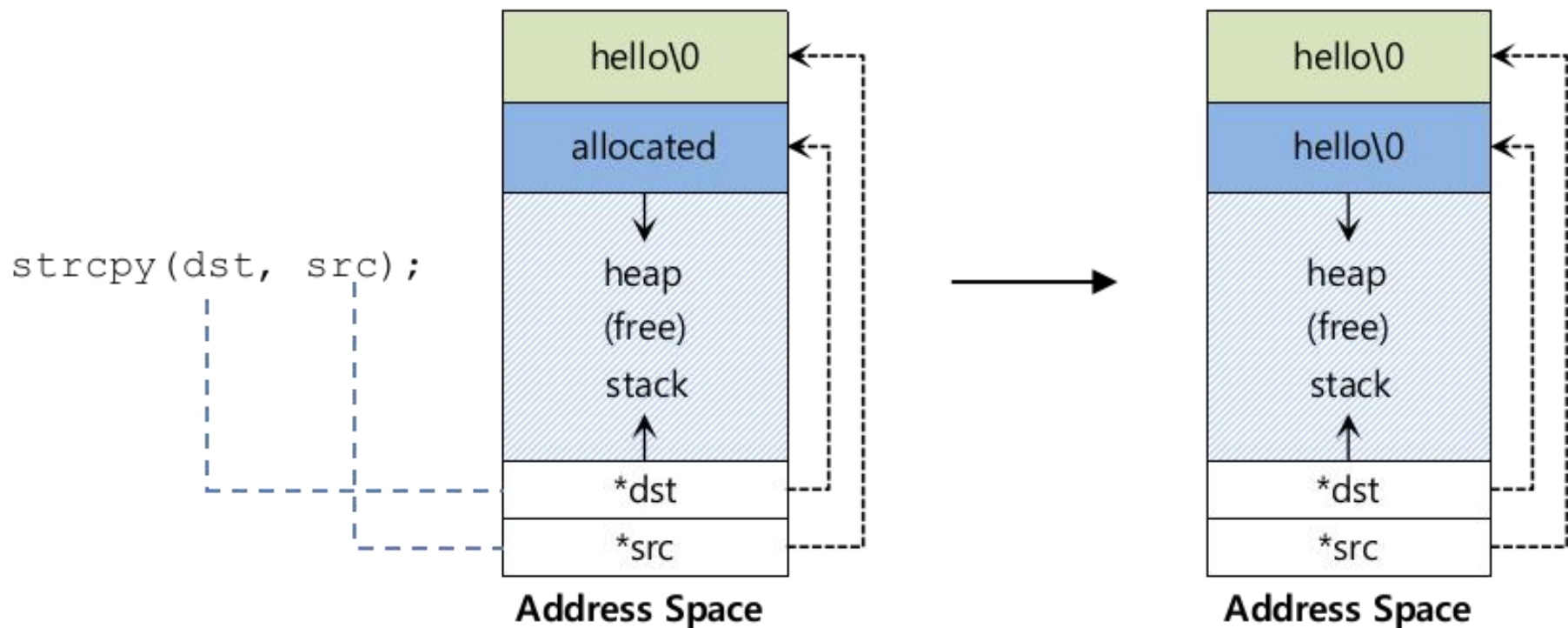 FOOLISH PROGRAMMER AND I AM ANGRY.

Segmentation Fault

```
char *src = "hello";   //character string constant
char *dst;             //unallocated
strcpy(dst, src);      //segfault and die
```

strcpy(dst, src);

hello\0

heap

(free)

stack

unallocated

*dst

*src
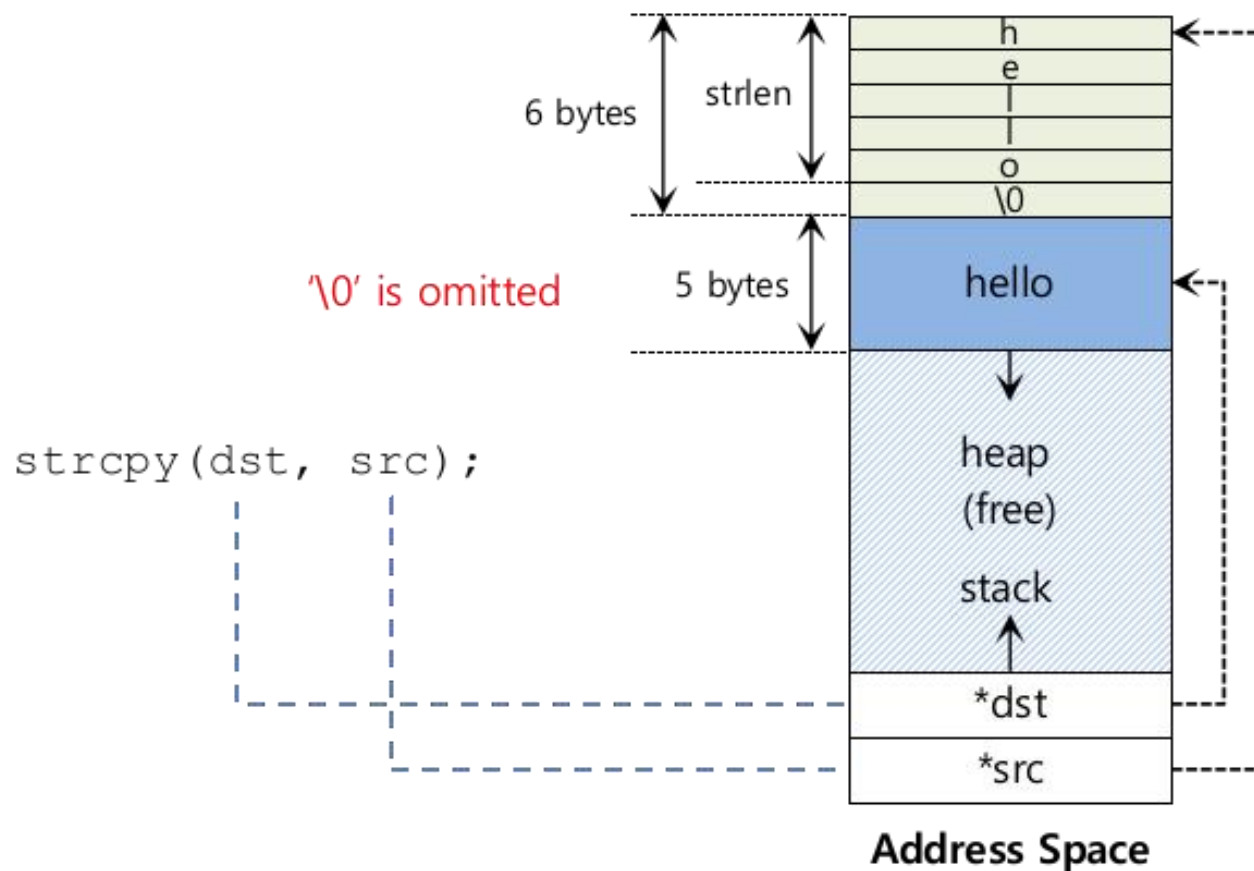
Address Space

Correct Code

```
char *src = "hello";    //character string constant
char *dst (char *)malloc(strlen(src) + 1 ); // allocated
strcpy(dst, src);       //work properly
```
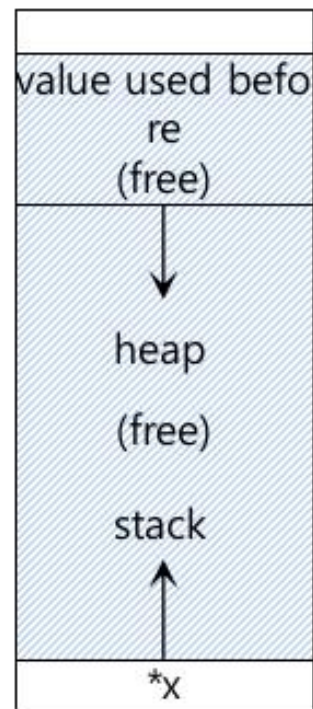
strcpy(dst, src);

hello\0	hello\0
allocated	hello\0
heap (free) stack	heap (free) stack
*dst	*dst
*src	*src
Address Space	**Address Space**

Works, but buggy

```
char *src = "hello"; //character string constant
char *dst (char *)malloc(strlen(src)); // too small
strcpy(dst, src);      //work properly
```
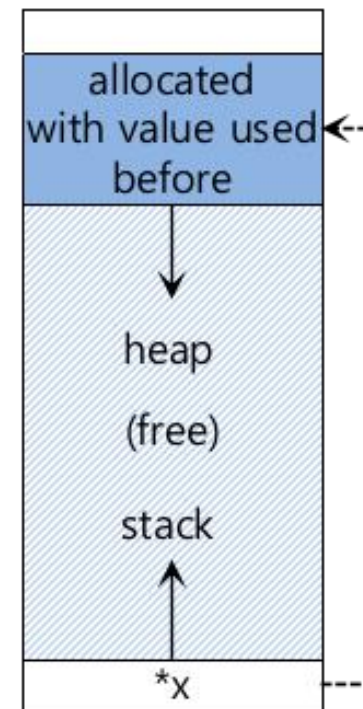

'\0' is omitted

```
strcpy(dst, src);
```

6 bytes · strlen · h e l l o \0
5 bytes · hello
heap (free)
stack
*dst
*src

Address Space

Uninitialized Read

- Wild pointer

```
int *x = (int *)malloc(sizeof(int)); // allocated
printf("*x = %d\n", *x); // uninitialized memory access
```
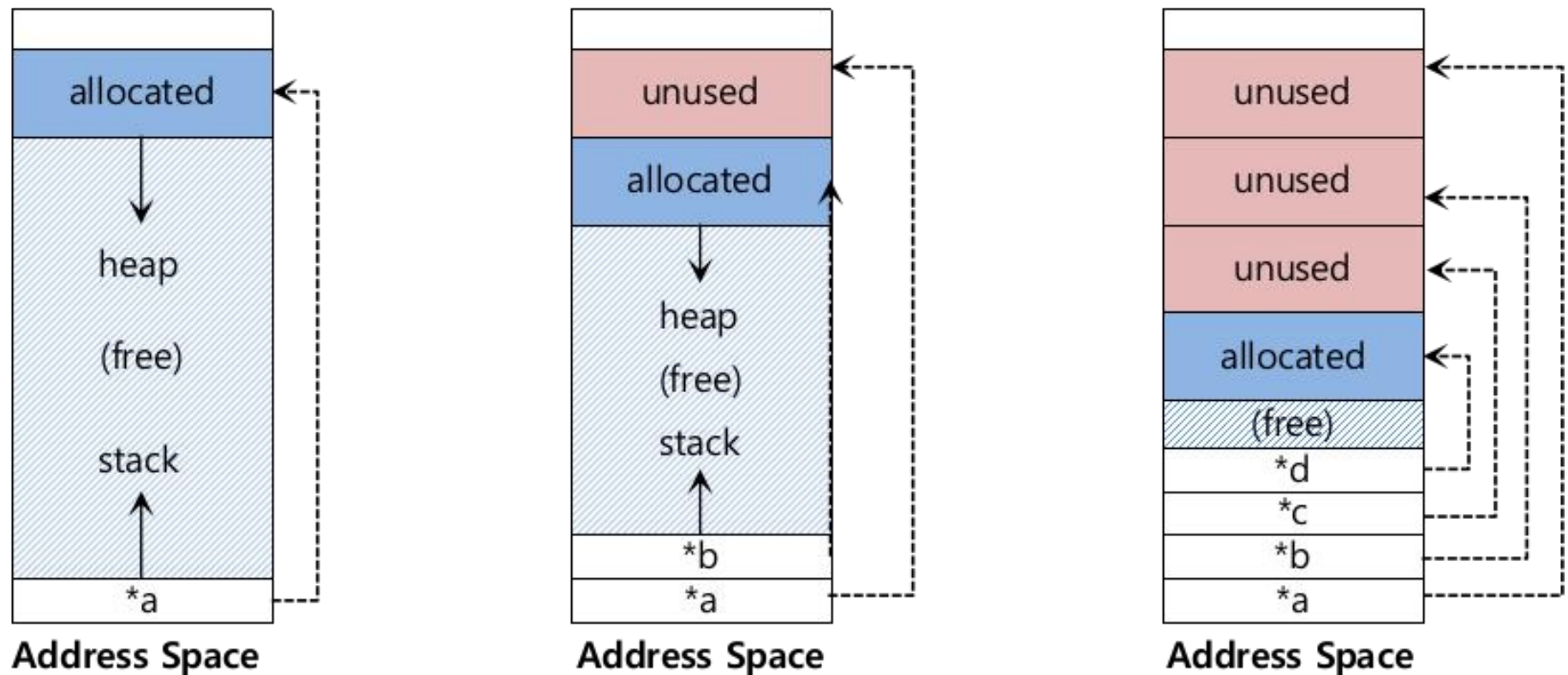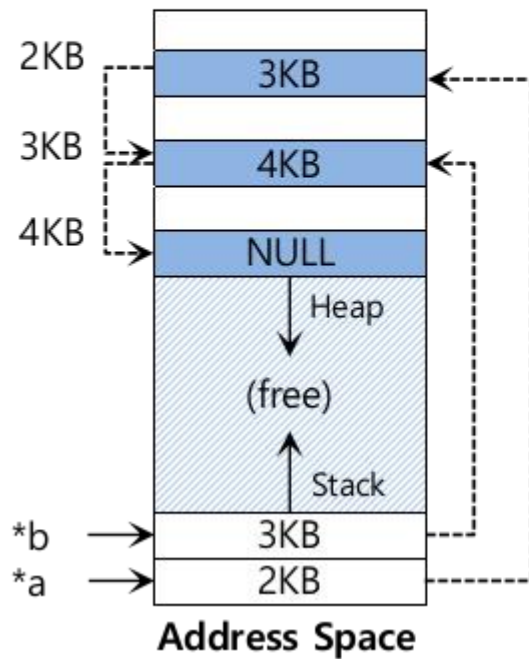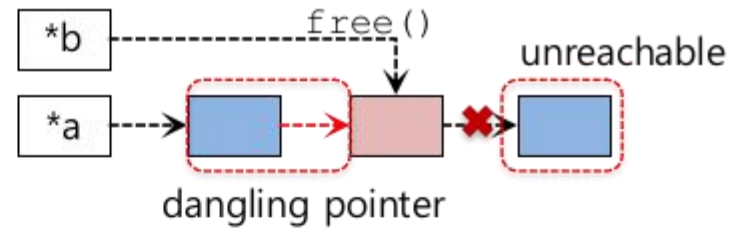
value used before (free)

heap (free)

stack

*x

Address Space

allocated with value used before
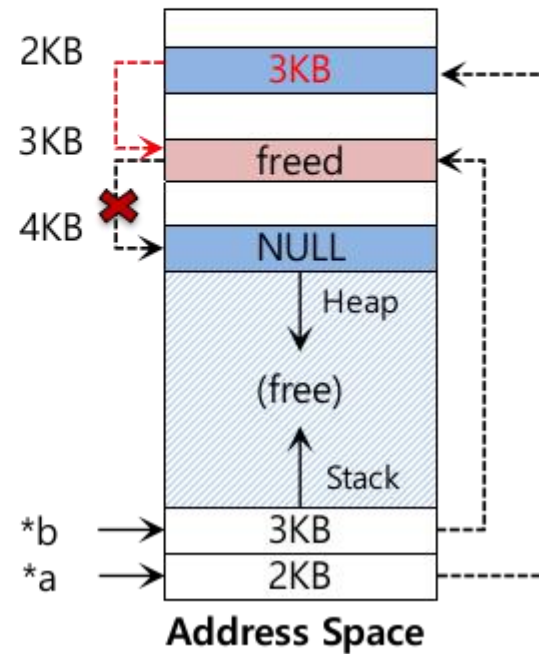
heap (free)

stack

*x

Address Space

Memory Leak

Dangling Pointer
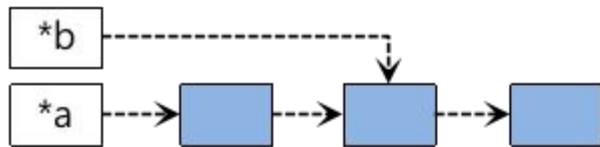
C Memory API

- ## Standard library
  - malloc(), realloc(), free()
- ## System calls
  - brk(), sbrk()
  - mmap()
- ## For comparison
  - printf() and write()
  - "Buffer the system call"

System calls: brk(), sbrk()
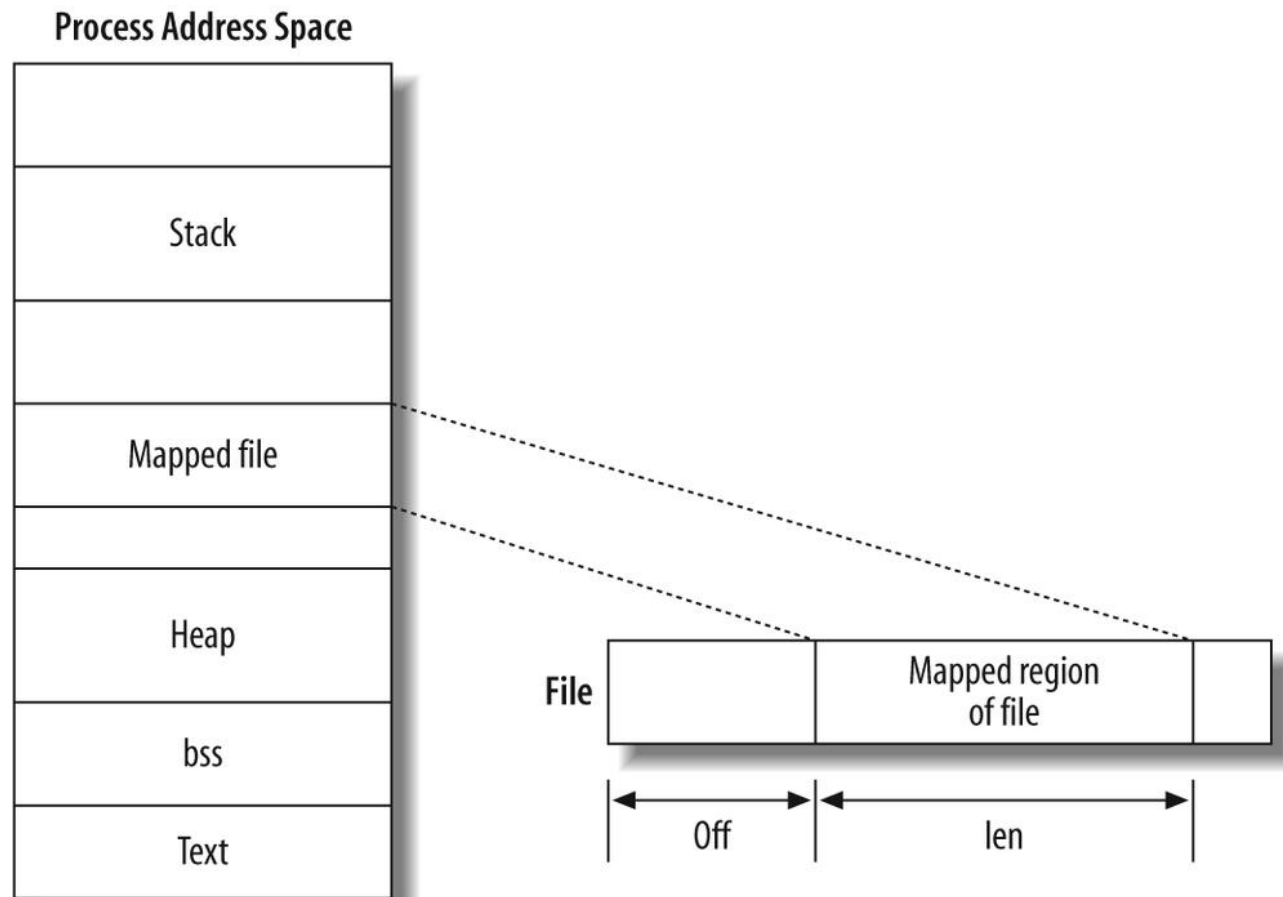
```
#include <unistd.h>

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

- brk/sbrk: expand the program's break.

  - break: The location of the end of the heap in address space

System calls: mmap()

#include <sys/mman.h>

void *mmap(void *ptr, size_t length, int port, int flags, int fd, off_t offset)

Process Address Space

Stack
Mapped file
Heap
bss
Text

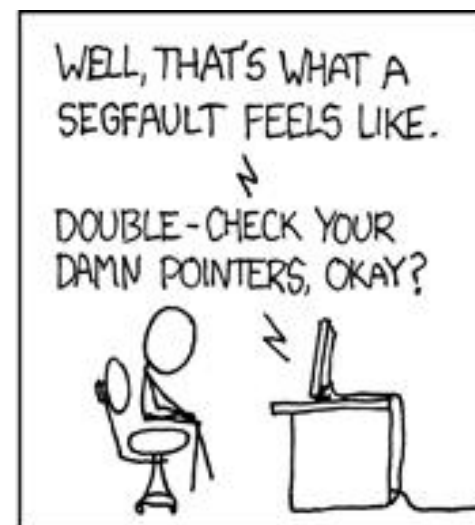File | | Mapped region of file | |

Off ← → len

C Memory API

- Summary: common errors
  - Forget to allocate memory
  - Not allocating enough memory
  - Forget to initialize allocated memory
  - Forget to free memory
  - Free memory before you are done with it
  - Free memory repeatedly
  - Call free() incorrectly

Free Memory Management

Dark Forest of Pointers

Free Memory Management

- Fixed-size unit
  - Paging
  - Problem: internal fragmentation
- Variable-size unit
  - User level memory allocation library
  - Kernel level: VM implemented with segmentation
  - Problem: external fragmentation

free	used	free
0	10	20	30

Free Memory Management

- Free memory management

    - How to manage variable-size free memory units

    - How to implement

        - malloc(size_t size)
   - free(void *ptr)

Free Memory Management

- Assumptions
  - Focus on external fragmentation
  - No compaction
  - Manage a contiguous region of bytes (by mmap() system call)

Free Memory Management

- Low-level Mechanisms

  - Splitting and Coalescing

  - Tracking allocated regions

  - Implementation of a free list

- High-level Intelligence

  - Best fit

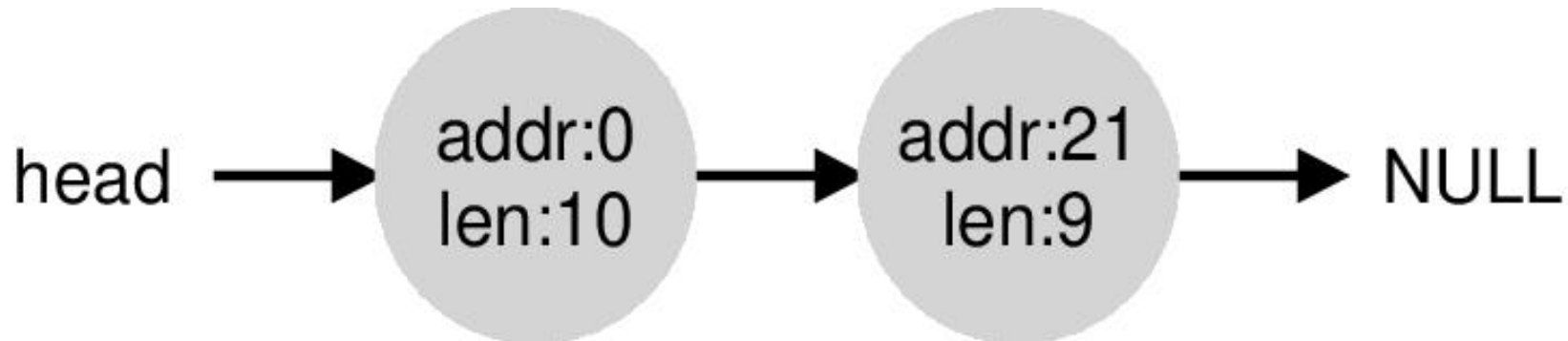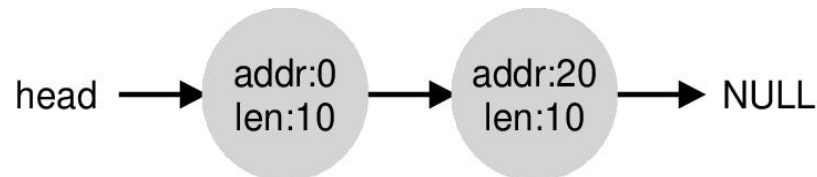 - Worst fit

  - First fit

  - Next fit

Free Memory Management

- Splitting and Coalescing
    - Free list: a set of free chunks
    - Two chunks (10 bytes each)
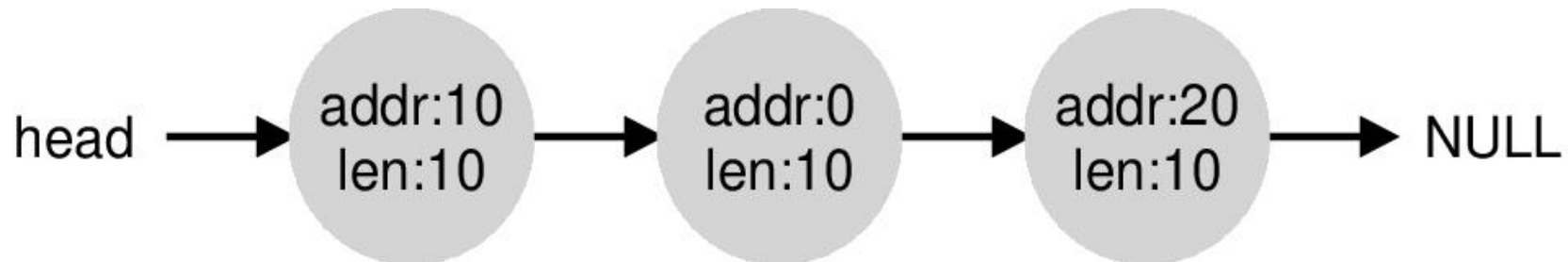
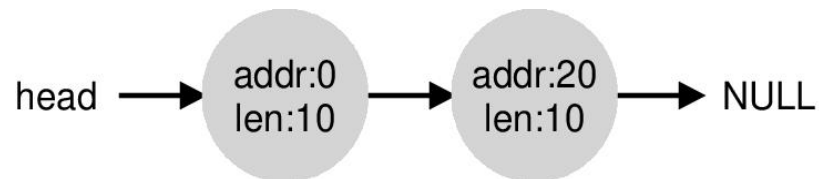| free	used	free
0	10 20	30

head → addr:0 len:10 → addr:20 len:10 → NULL
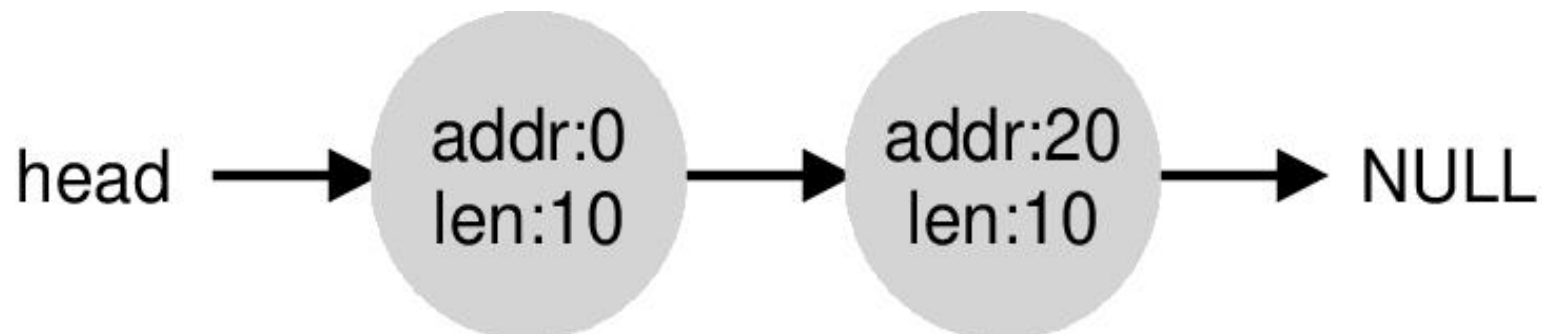
Free Memory Management

- Splitting and Coalescing
    - request less than 10 bytes? (e.g. malloc(1))
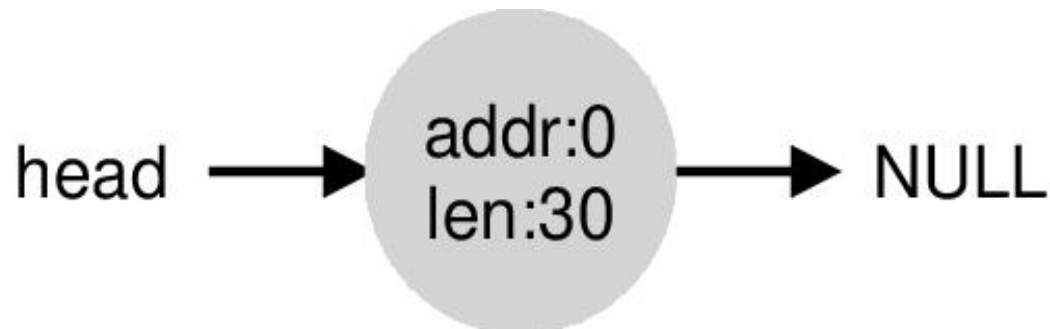   - Splitting

Free Memory Management

- Splitting and Coalescing
  - Free a chunk?

  - Malloc(20)?
  - Coalescing

Free Memory Management

- Tracking Allocated Regions
  - Observation on free(void *ptr)
    - No size parameter
  - Given a pointer, the malloc library could determine the size of region
  - How?
    - Some extra information
    - header of a memory block
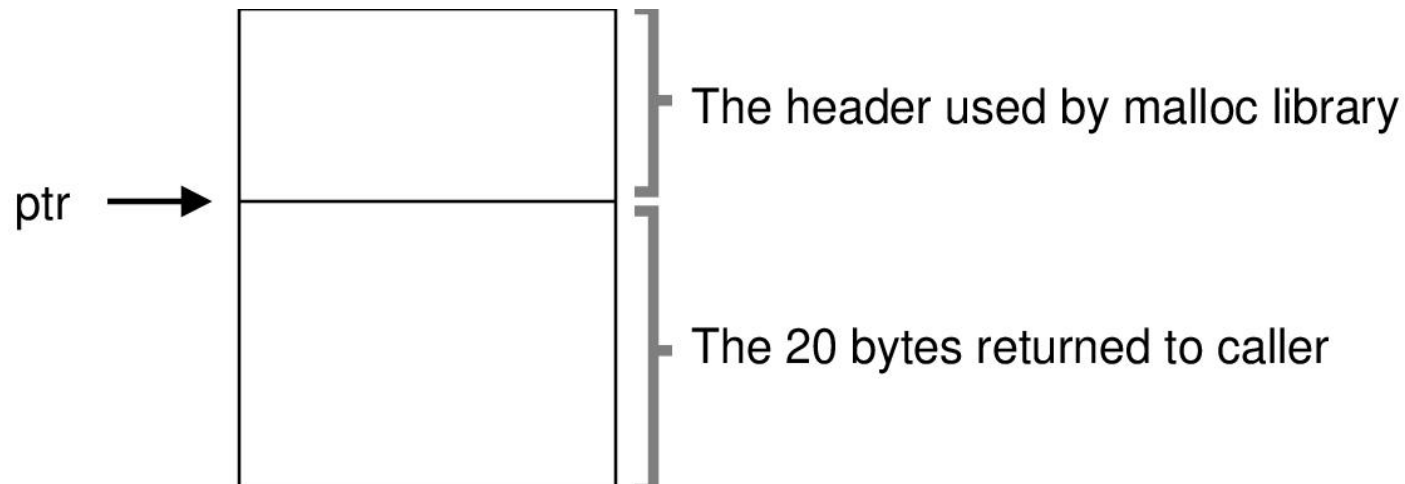
Free Memory Management

- Tracking Allocated Regions
  - header

    ```
    typedef struct __header_t {
            int size;
            int magic;
    } header_t;
    ```

  - malloc(20)

The header used by malloc library

The 20 bytes returned to caller

Free Memory Management

- Tracking Allocated Regions
  - header: example
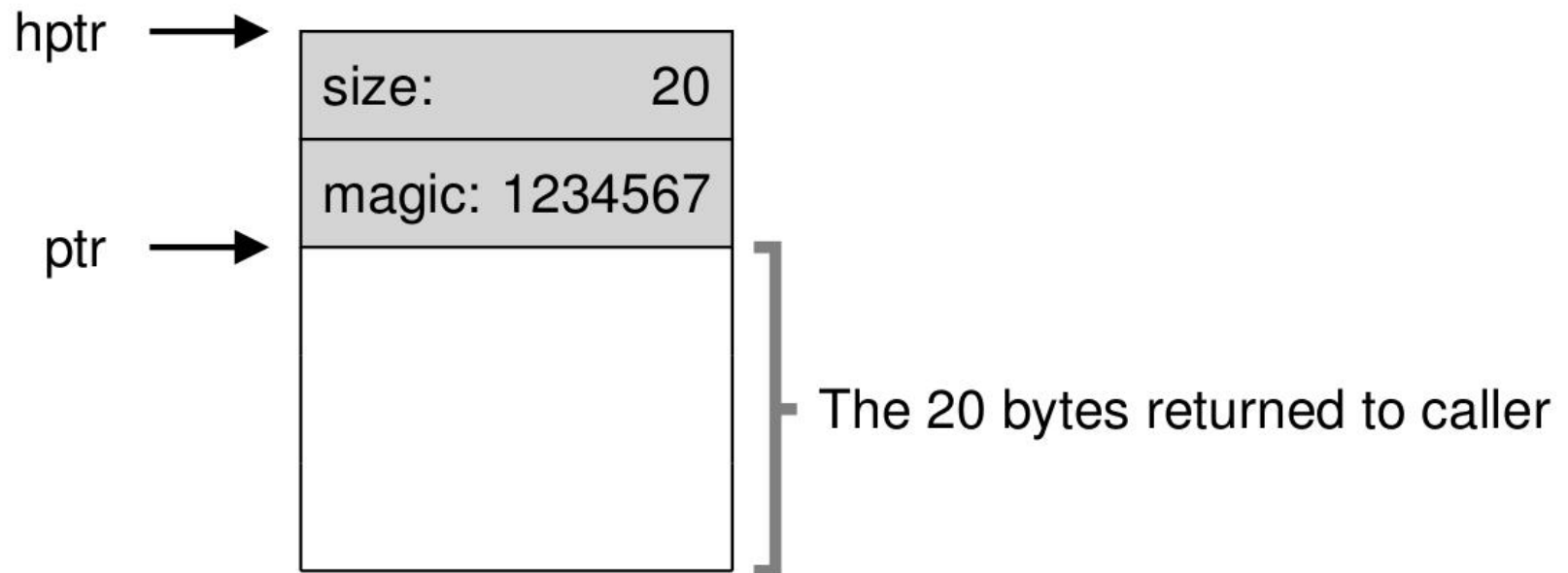
Free Memory Management

- Tracking Allocated Regions
  - free(ptr)
    - Get the size of the region

    ```
    void free(void *ptr) {
            header_t *hptr = (void *)ptr - sizeof(header_t);
    }
    ```

    - Check whether ptr is valid

    ```
    assert(hptr->magic == 1234567)
    ```

Free Memory Management

- Implementation of the Free List

  - Free list
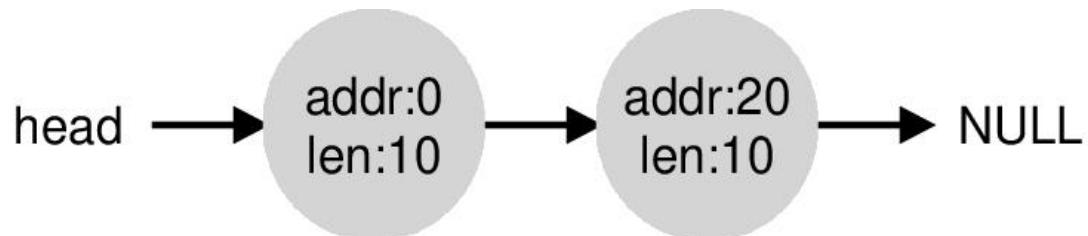
  - Implementation

    - List node (allocate a node when needed)

    - Can NOT do this here! All you have is a given free space

  - How to build a free list **inside** the free space?
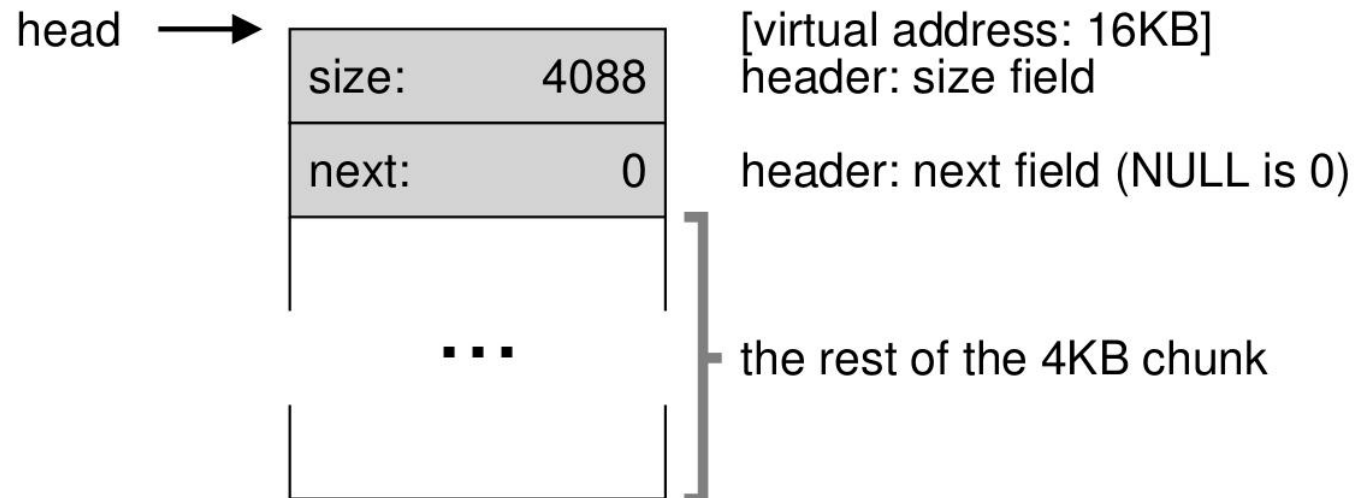
Free Memory Management

- Implementation of the Free List
  - Node in free list

```
typedef struct __node_t {
        int size;
        struct __node_t *next;
} node_t;
```

Free Memory Management

- ## Implementation of the Free List

  - ### Initialization (e.g. 4096)

```
// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|
                         PROT_WRITE, MAP_ANON|MAP_PRIVATE, -1, 0);
head->size = 4096 - sizeof(node_t);
head->next = NULL;
```


head →

size:	4088
next:	0

[virtual address: 16KB]
header: size field

header: next field (NULL is 0)

...

the rest of the 4KB chunk

Free Memory Management

- Implementation of the Free List
  - malloc(100)

- malloc(100)*3

[Virtual address: 16KB]

| size: | 100 |
| magic: | 1234567 |

··· } 100 bytes still allocated

| size: | 100 |
| magic: | 1234567 |

sptr →

··· } 100 bytes still allocated (but about to be freed)

| size: | 100 |
| magic: | 1234567 |

··· } 100-bytes still allocated

head →

| size: | 3764 |
| next: | 0 |

··· The free 3764-byte chunk

- Free(16500)
  - 16384+108+8

size:	100
magic: 1234567	

. . .   ⎤ 100 bytes still allocated

head →

size:	100
next:	16708

sptr →

. . .   (now a free chunk of memory)

size:	100
magic: 1234567	

. . .   ⎤ 100-bytes still allocated

size:	3764
next:	0

. . .   ⎤ The free 3764-byte chunk

- Free()*3
- Coalesce
  - Merge adjacent
  - chunks

[Virtual address: 16KB]

| size: | 100 |
| next: | 16492 |

(now free)

| size: | 108 |
| next: | 16708 |

(now free)

head →

| size: | 100 |
| next: | 16384 |

(now free)

| size: | 3764 |
| next: | 0 |

The free 3764-byte chunk
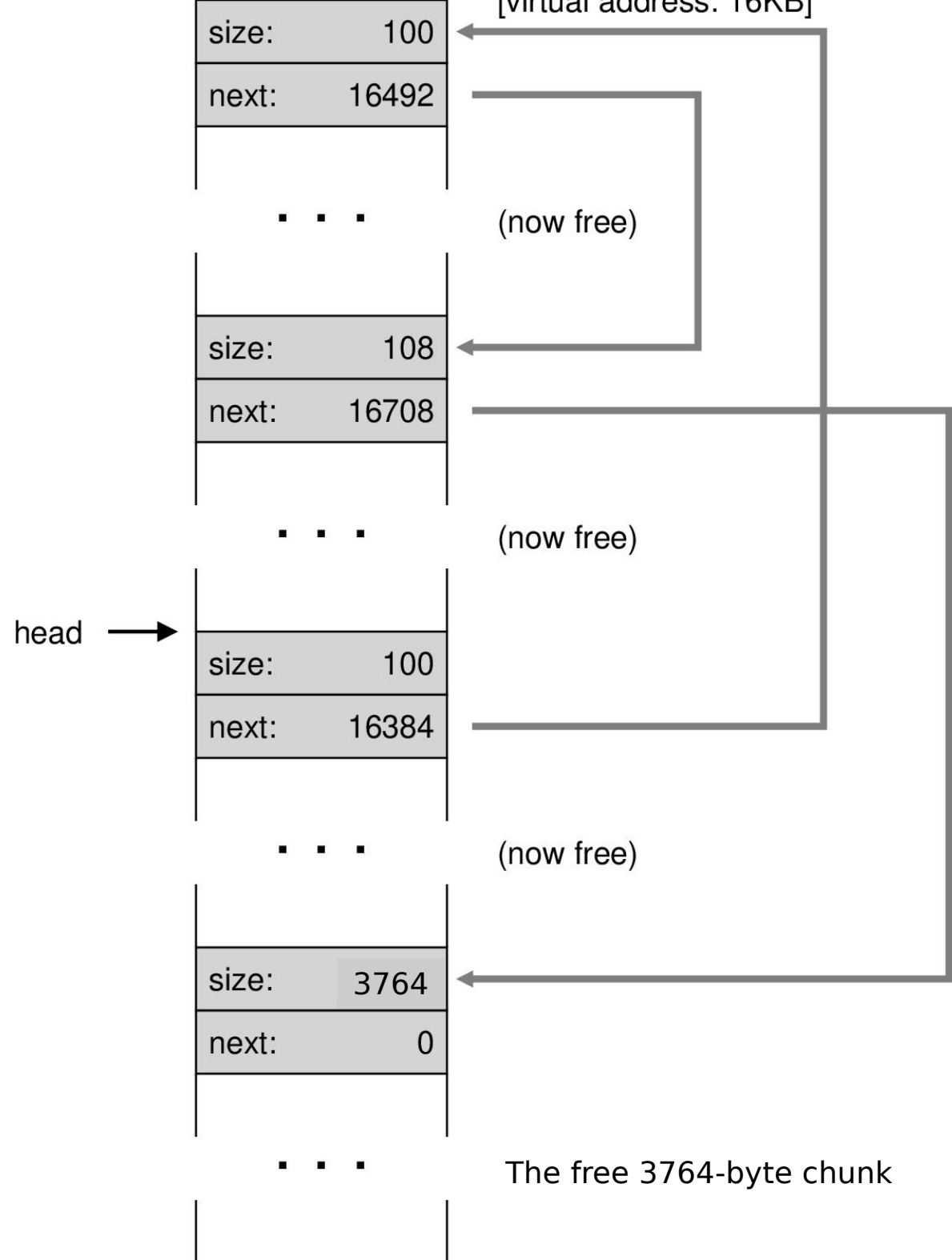
Free Memory Management

- Growing the Heap
  - What if the heap runs out of space?
    - Return NULL
  - Increase the size of heap
    - OS find free physical pages
    - Map them into address space of the process

Free Memory Management

- Summary of low-level Mechanisms
  - Splitting and Coalescing
  - Tracking allocated regions
  - Implementation of a free list
  - Growing the heap
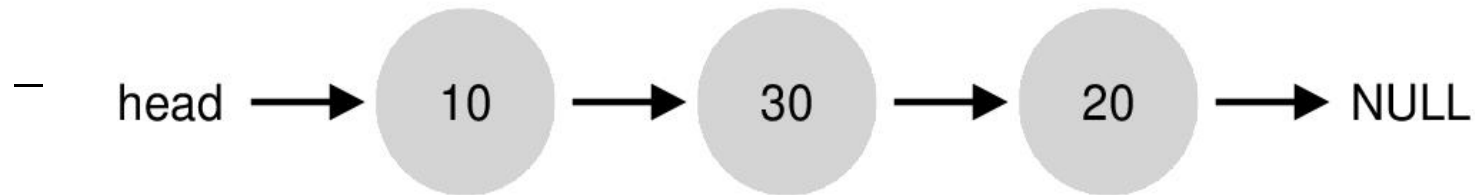
Free Memory Management

- High-level intelligence
  - How to find the proper nodes in the free list?
    - Less fragmentation
    - Fast allocation
  - Some simple strategies
    - The stream of allocation and free requests can be arbitrary
    - Any strategy could be arbitrarily bad/good

Free Memory Management

- Best Fit

  - Find the smallest feasible node

- Worst Fit

  - Find the largest feasible node

- First Fit

  - Find the first feasible node
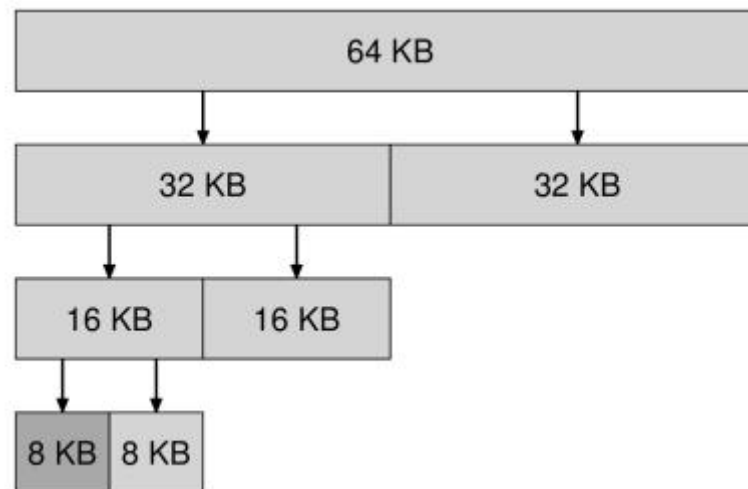
Free Memory Management

- Example

  - Best fit

  - Worst fit

Free Memory Management

- Other approaches
  - Segregated List
    - Slab allocator
  - Buddy Allocation
    - Binary search tree

Free Memory Management

- Dlmalloc (Doug Lea allocator)

  - Segregated list for small size allocations

  - Search the free list

  - sbrk and mmap

  - ...

- http://g.oswego.edu/dl/html/malloc.html

- https://cs61.seas.harvard.edu/wiki/images/e/e2/Lec 11-Dynamic_memory_2.pdf