Operating System Labs

Yuanbin Wu
CS@ECNU

Operating System Labs

- Project 2 Due
  - 21:00, Oct. 29
- Project 3
  - Group of 3
  - For each group: email group members to TAs
  - If you can not find a partner, drop us an email
  - You now have 3 “late days”, but start early!
  - We will have oral test at week 12 (Nov. 27)

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 call the function
  - Also deallocate it when func returns

C Memory API

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

C Memory API

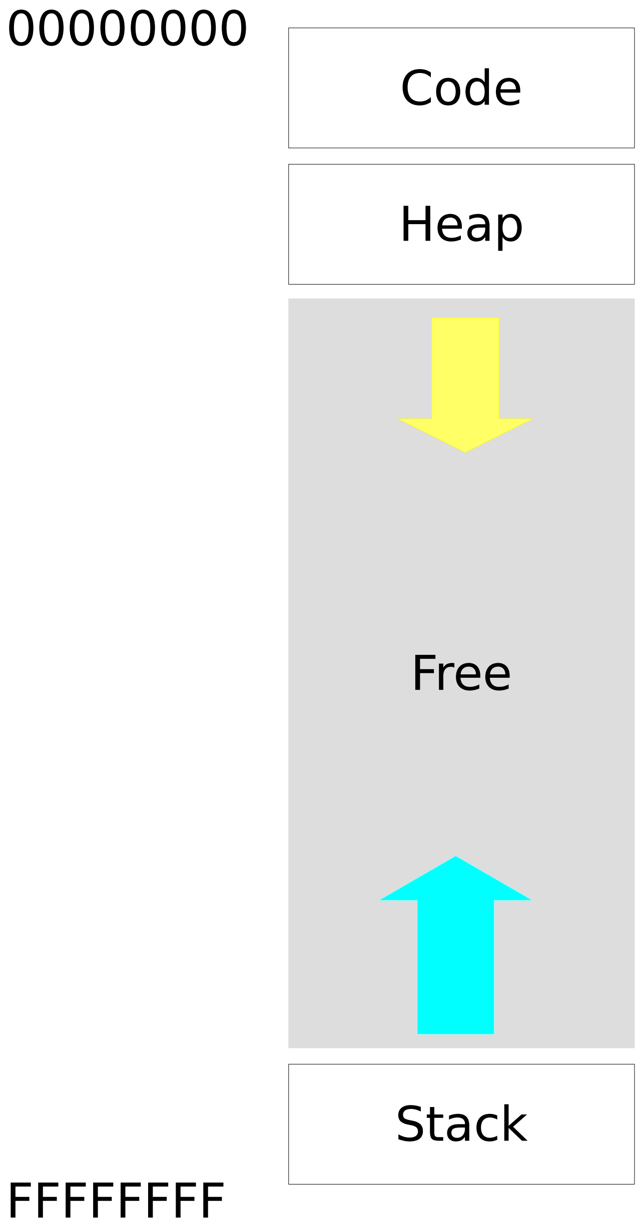
- Heap
  - Example (malloc)

```
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



C Memory API

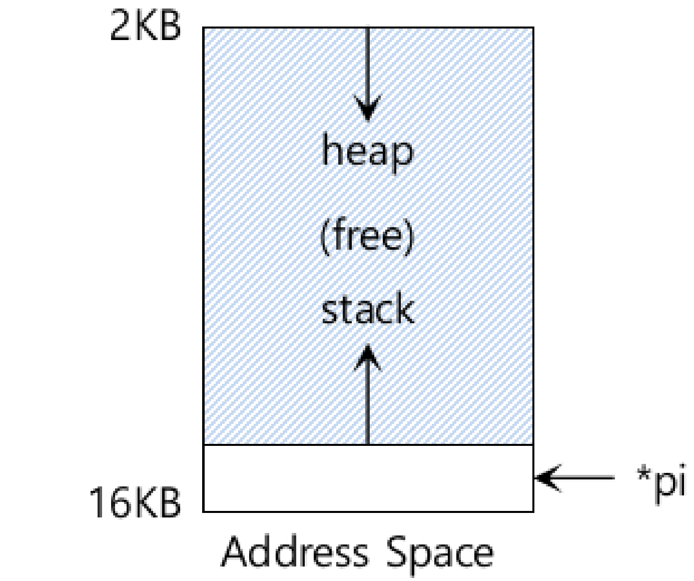
- Malloc

  ```
  #include <stdlib.h>
  void *malloc(size_t size);
  ```

  - If failed, return NULL
- Free

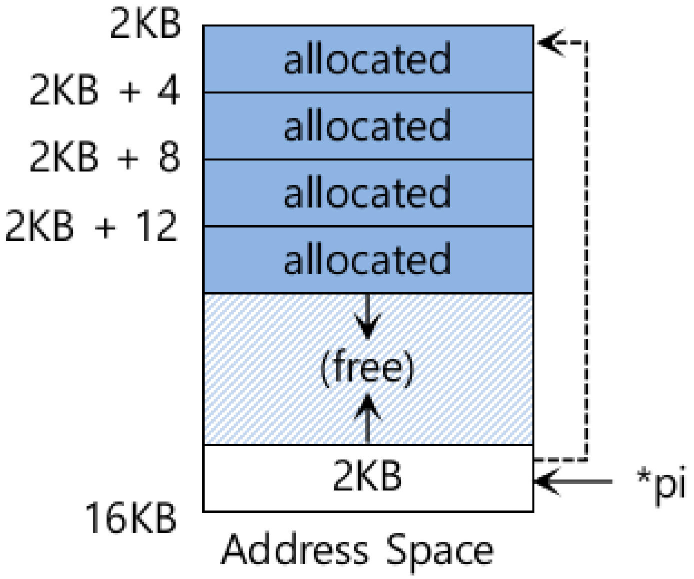
  ```
  #include <stdlib.h>
  void free(void* ptr);
  ```

Allocation



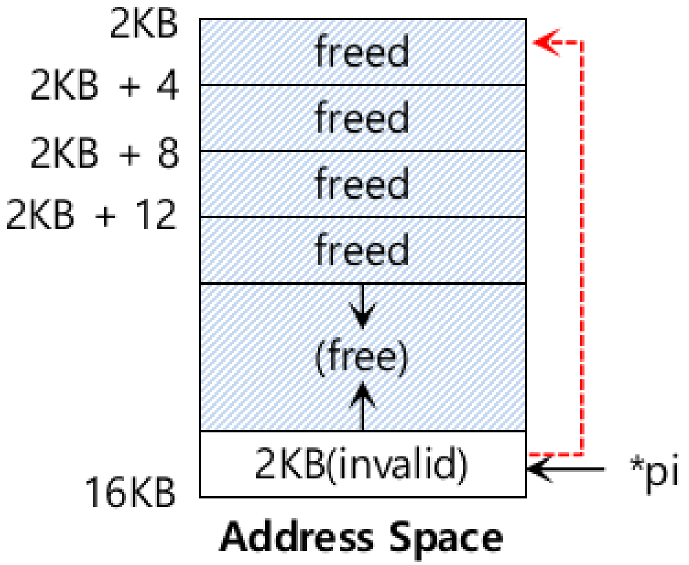
---------> pointer

```
int *pi; // local variable
```

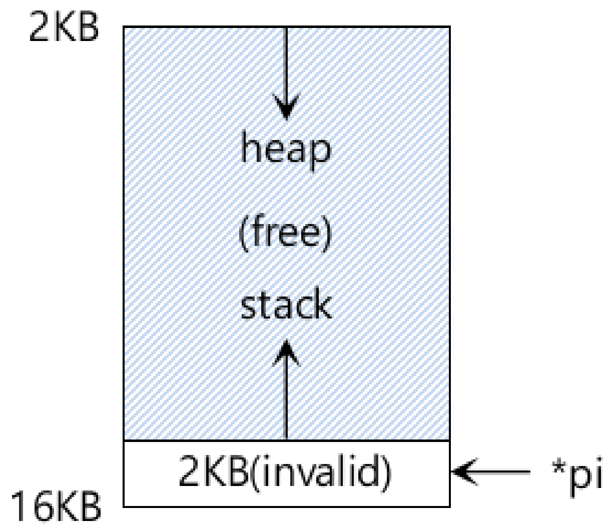


```
pi = (int *)malloc(sizeof(int)*
4);
```

Free


2KB
2KB + 4
2KB + 8
2KB + 12
16KB
freed
freed
freed
freed
(free)
2KB(invalid)
*pi
Address Space

```
free(pi);
```


2KB
16KB
heap
(free)
stack
2KB(invalid)
*pi

C Memory API

- Segment fault

```
char *src = "hello";
char *dst;
strcpy(dst, src);
```

- run this code, it will likely lead to a

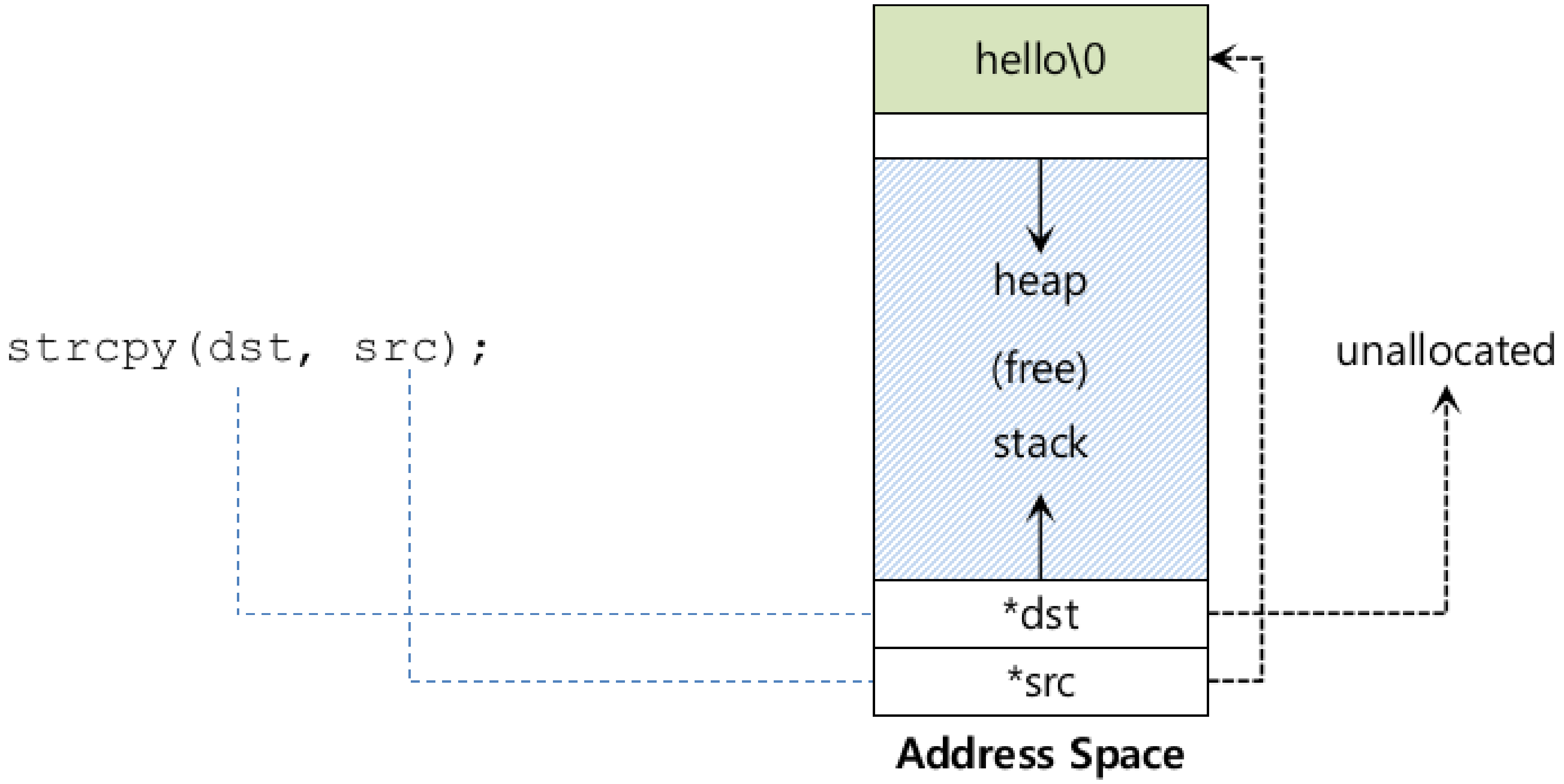
segmentation fault

- It is a fancy term for

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

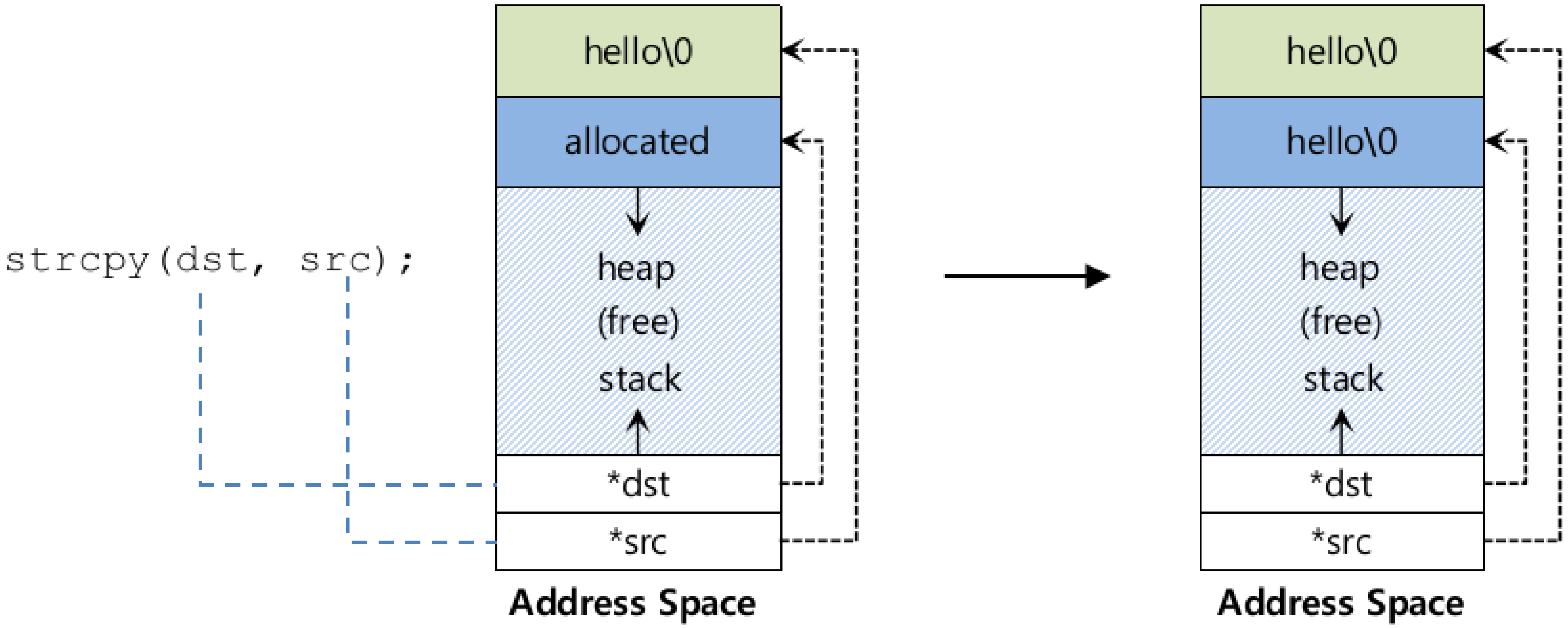
Segmentation Fault

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



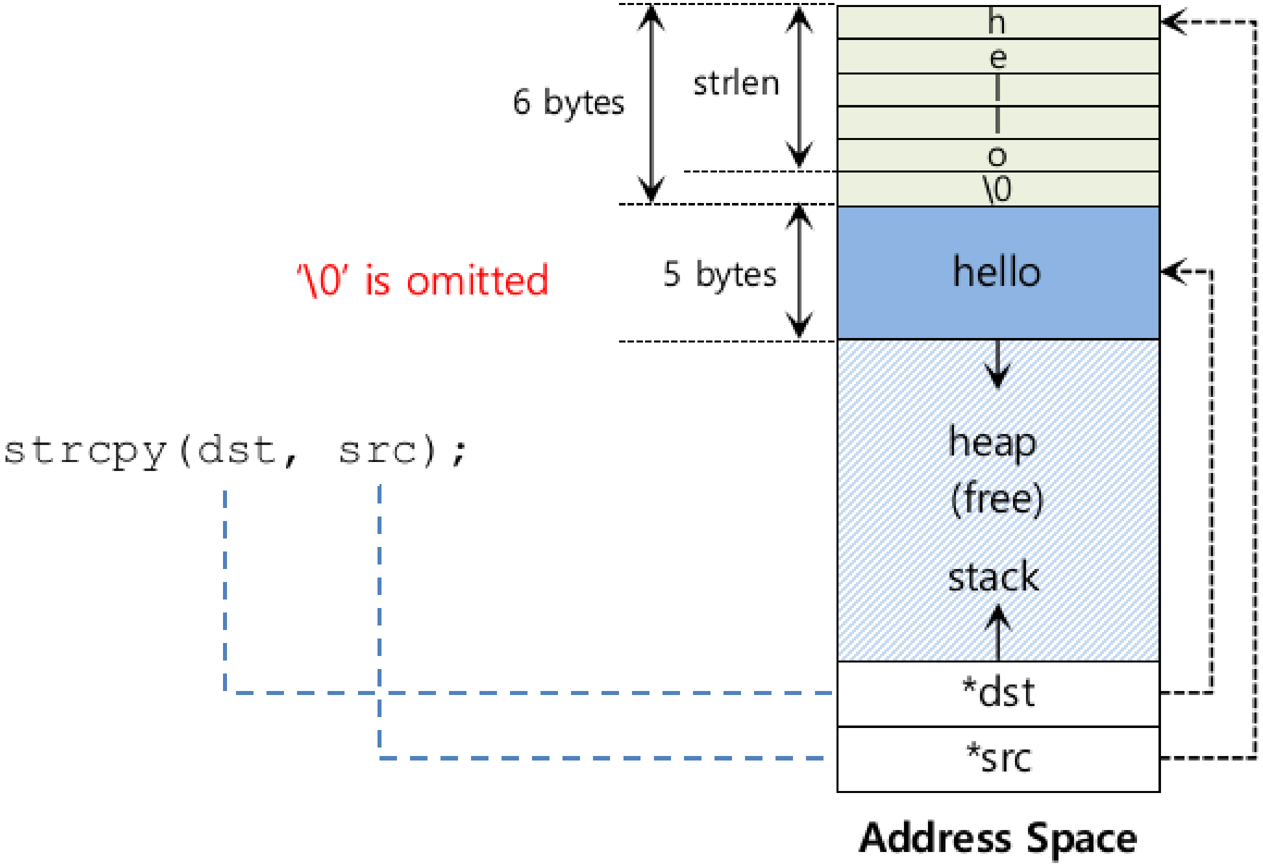
Correct Code

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



Works, but buggy

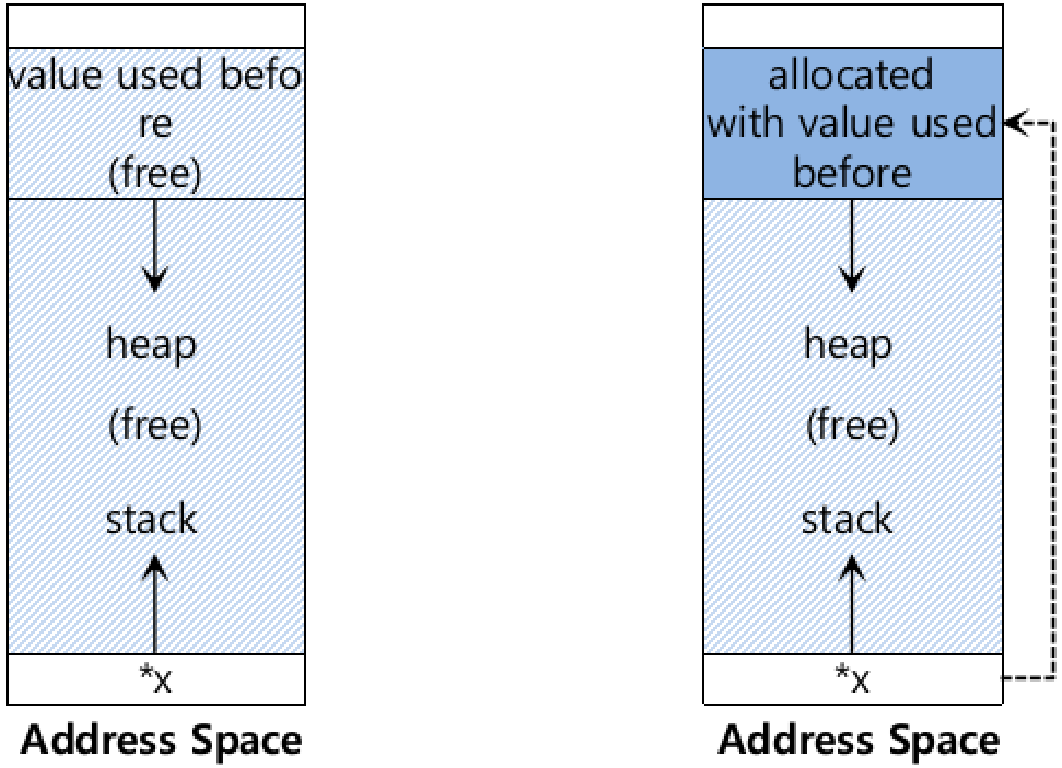
```
char *src = "hello"; //character string constant
char *dst (char *)malloc(strlen(src)); // too small
strcpy(dst, src);     //work properly
```



Uninitialized Read

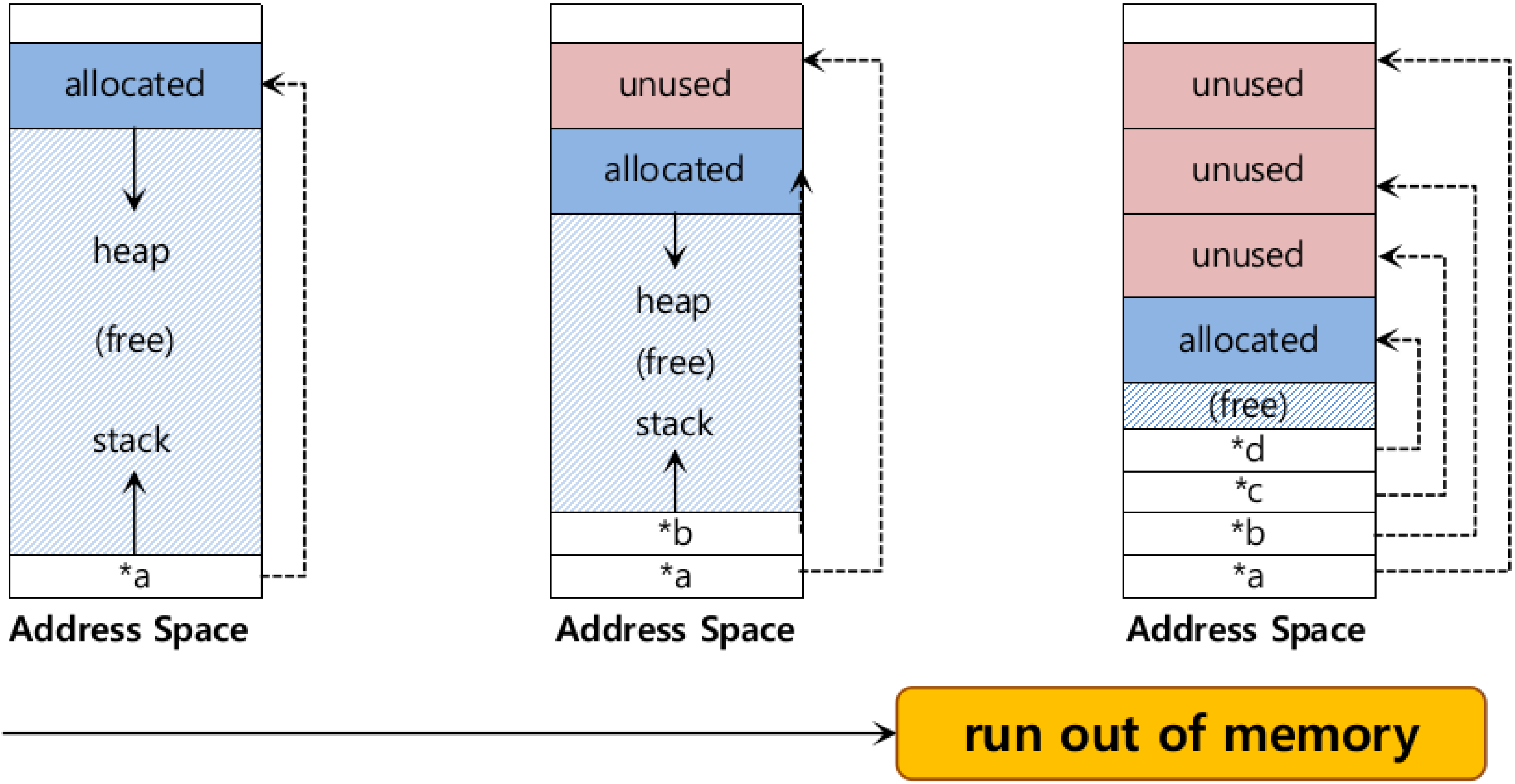
- Wild pointer

```
int *x = (int *)malloc(sizeof(int)); // allocated
printf(“*x = %d\n”, *x); // uninitialized memory access
```

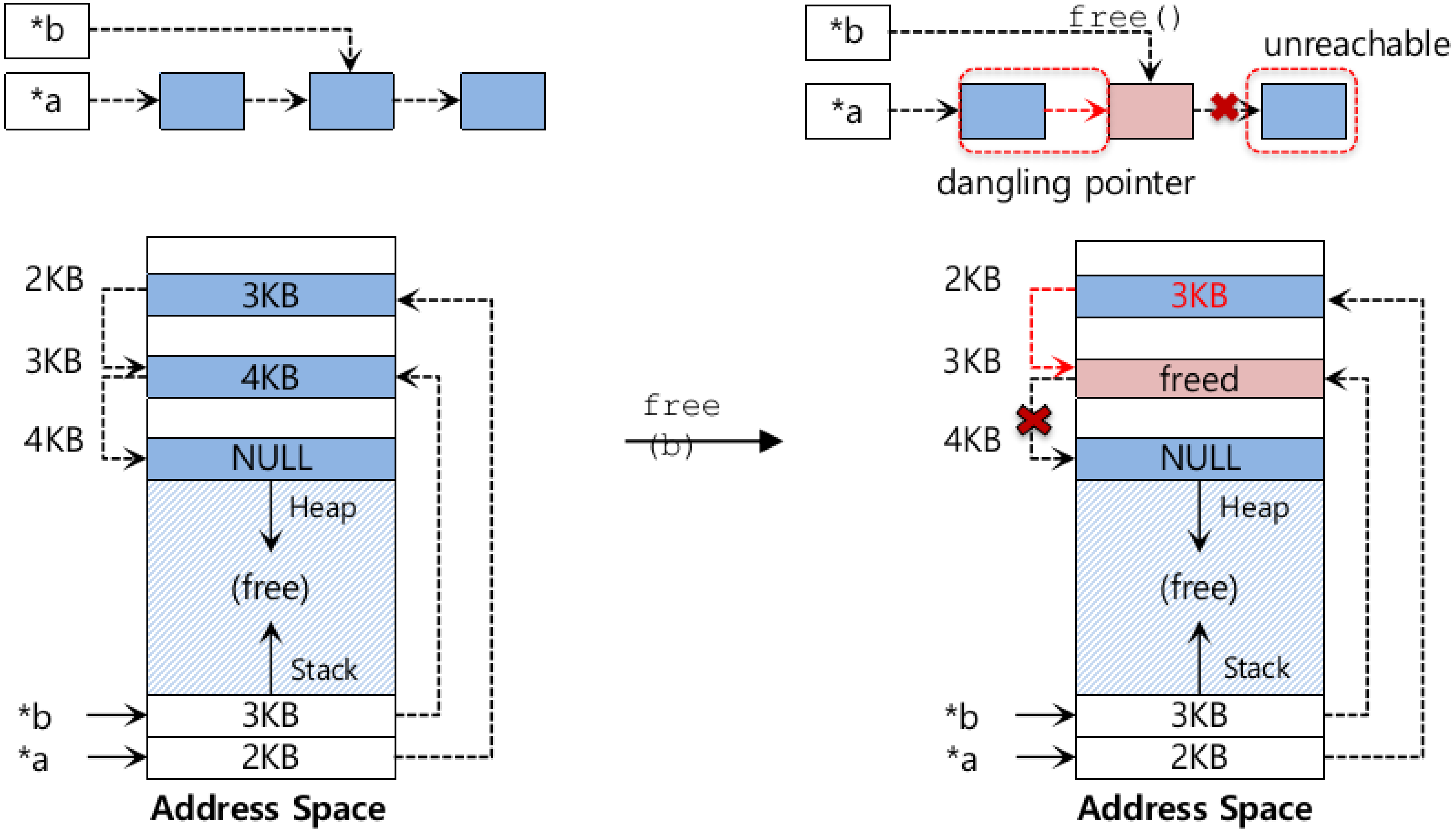


Memory Leak

unused : unused, but not freed



Dangling Pointer



C Memory API

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

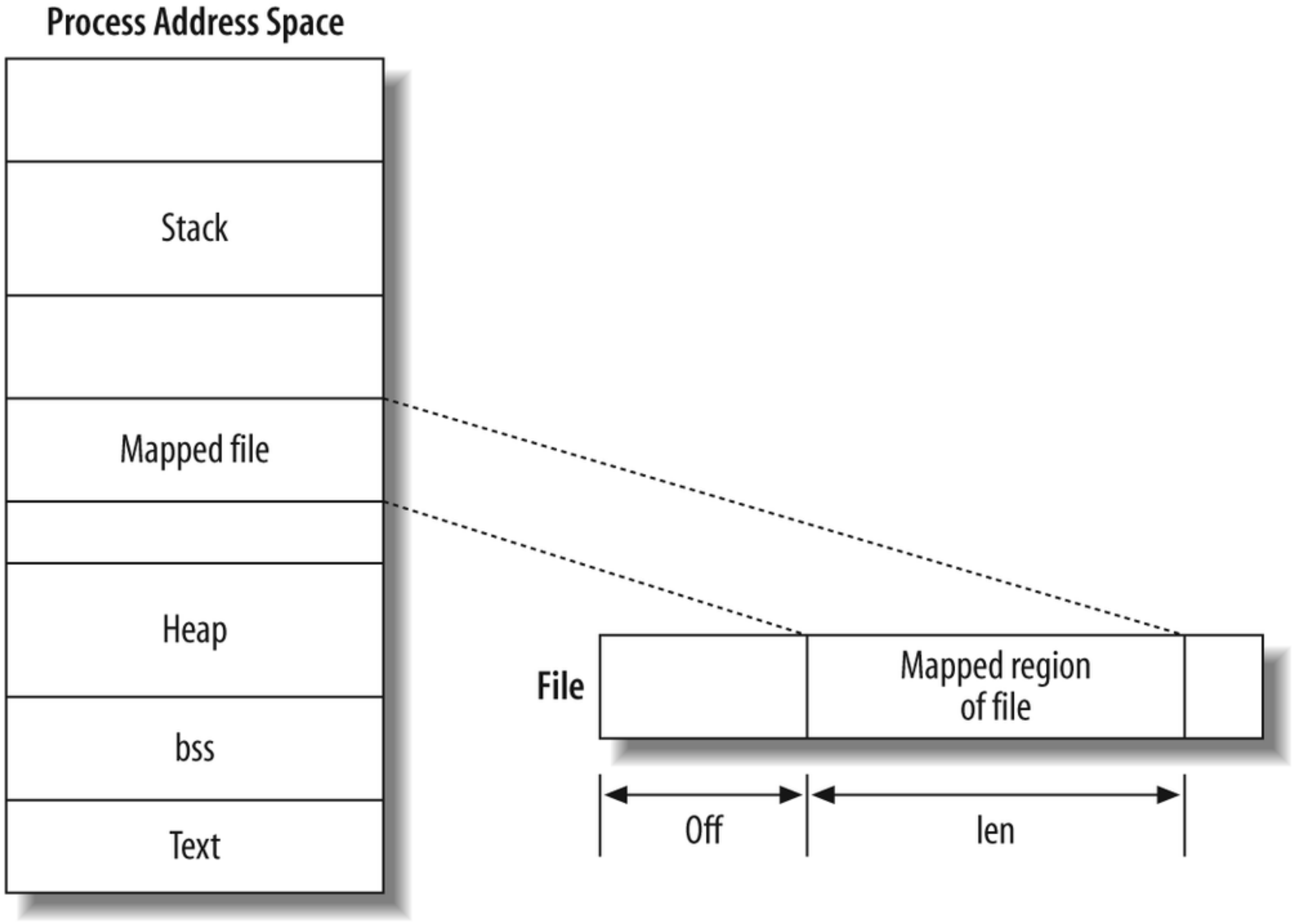
System calls: brk(), sbrk()

```
#include <unistd.h>

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

- brk is called to expand the program’s break.
  - break: The location of the end of the heap in address space
- sbrk is an additional call similar with brk.
- Programmers should never directly call either brk or sbrk

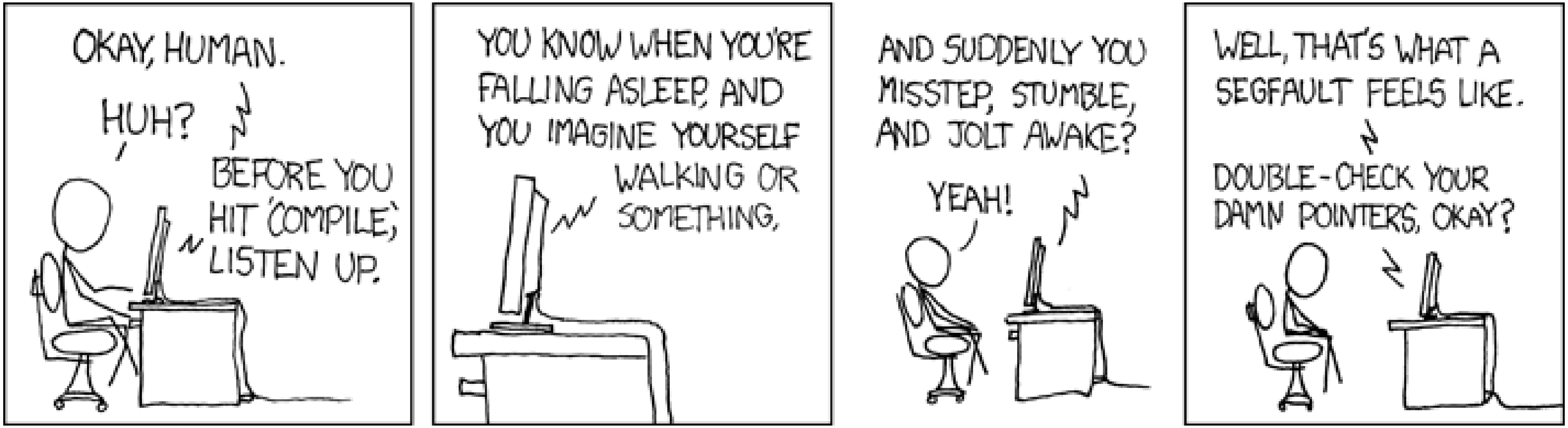
System calls: mmap()

```
#include <sys/mman.h>

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



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

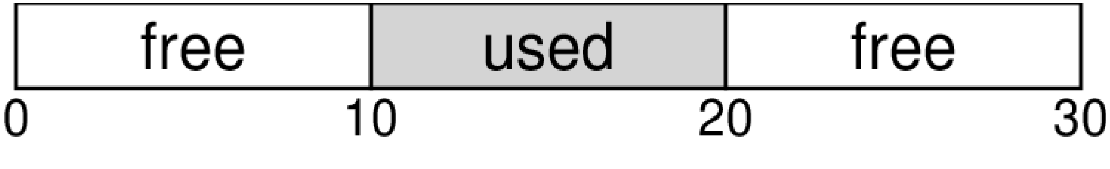

OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE,' LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF
WALKING OR SOMETHING,
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

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 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

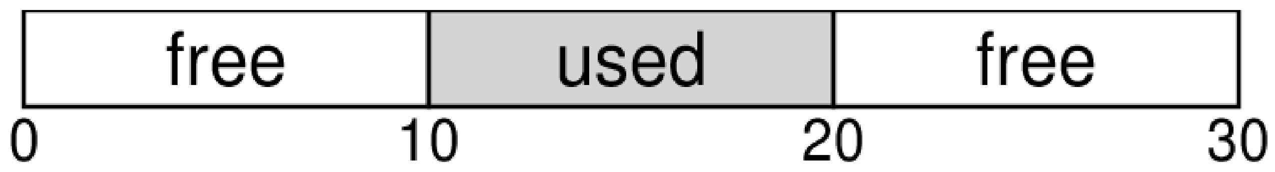
- Assumptions
  - Focus on external fragmentation
  - No compaction
  - Manage a contiguous region of bytes (by mmap() system call)

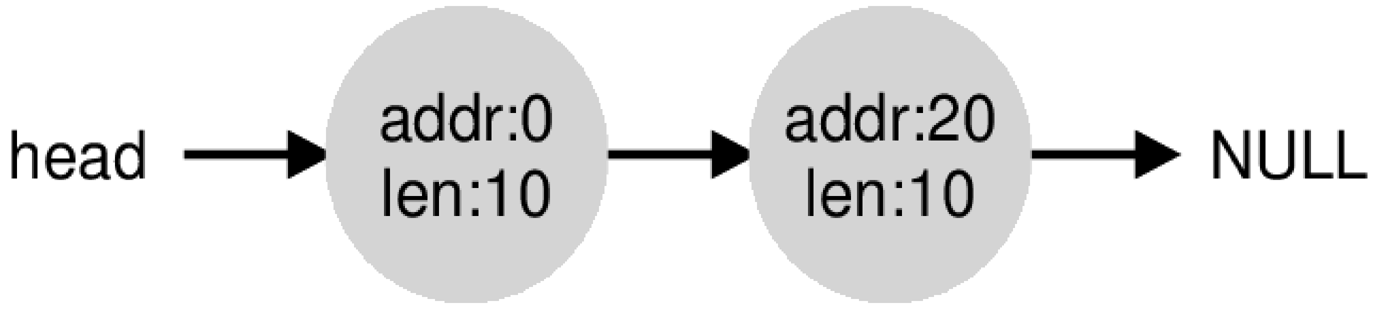
Free Memory Management

- 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

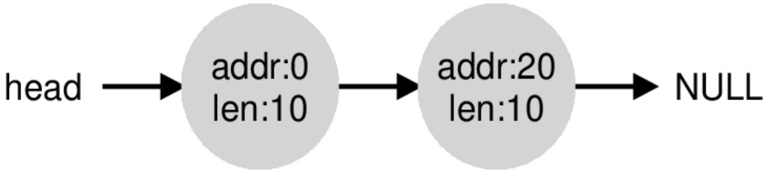
- Splitting and Coalescing
  - Free list: a set of free chunks
  - Two chunks (10 bytes each)

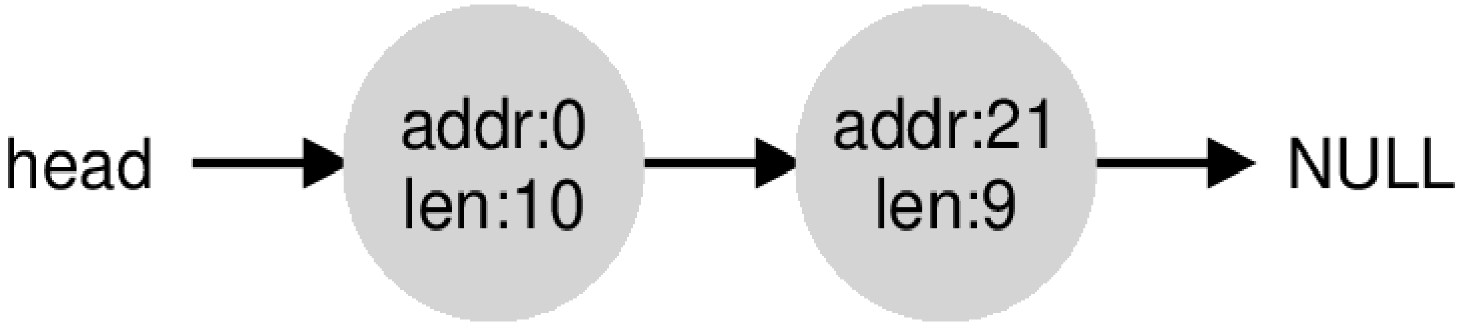




Free Memory Management

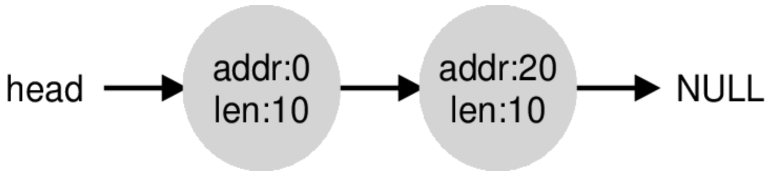
- Splitting and Coalescing
  - request less than 10 bytes? (e.g. malloc(1))
  - Splitting

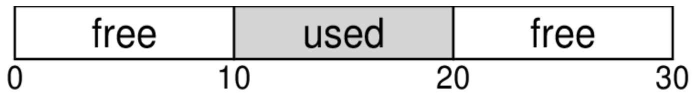


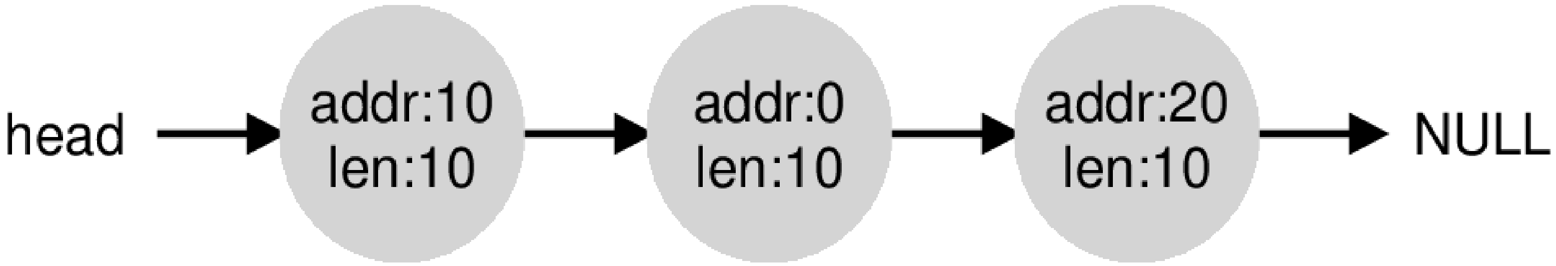


Free Memory Management

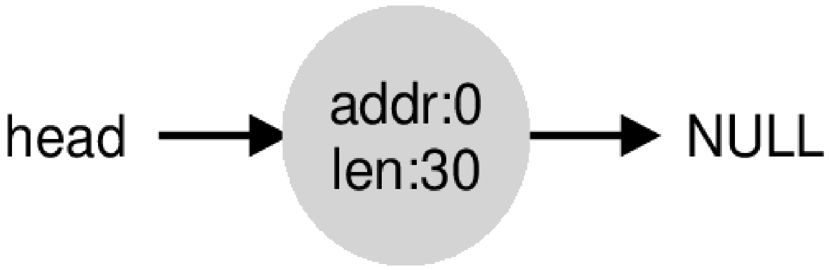
- Splitting and Coalescing
  - Free a chunk?







  - Malloc(20)?
  - Coalescing



Free Memory Management

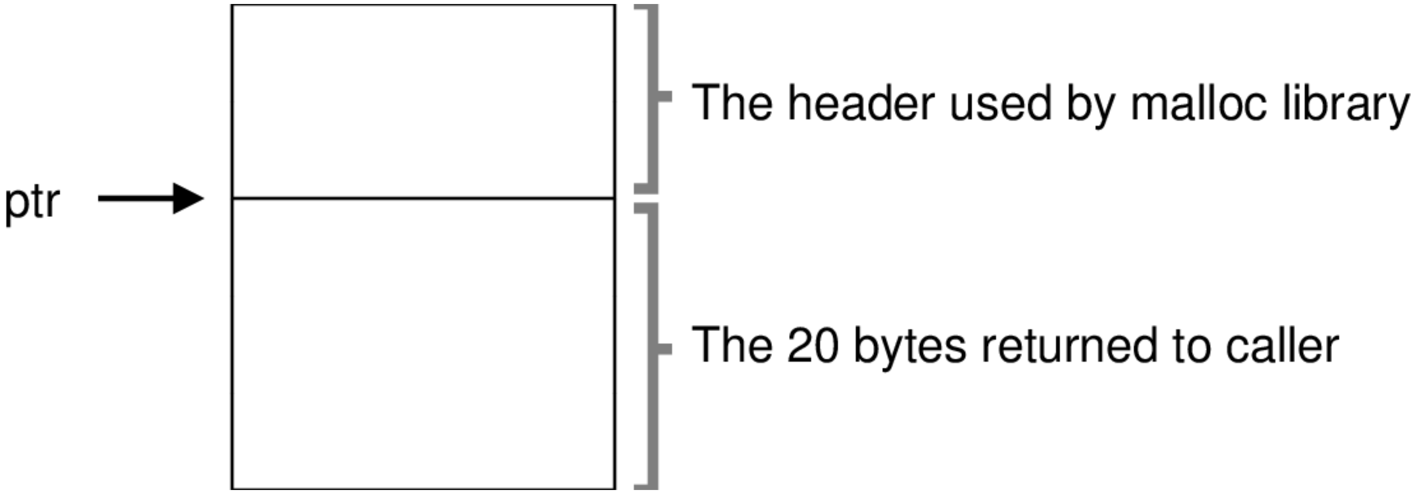
- Tracking Allocated Regions
  - Observation on free(void *ptr)
    - No size parameter
  - Given a pointer, the malloc library could determine the size of region
  - How?
    - Some extra information
    - header of a memory block

Free Memory Management

- Tracking Allocated Regions
  - header

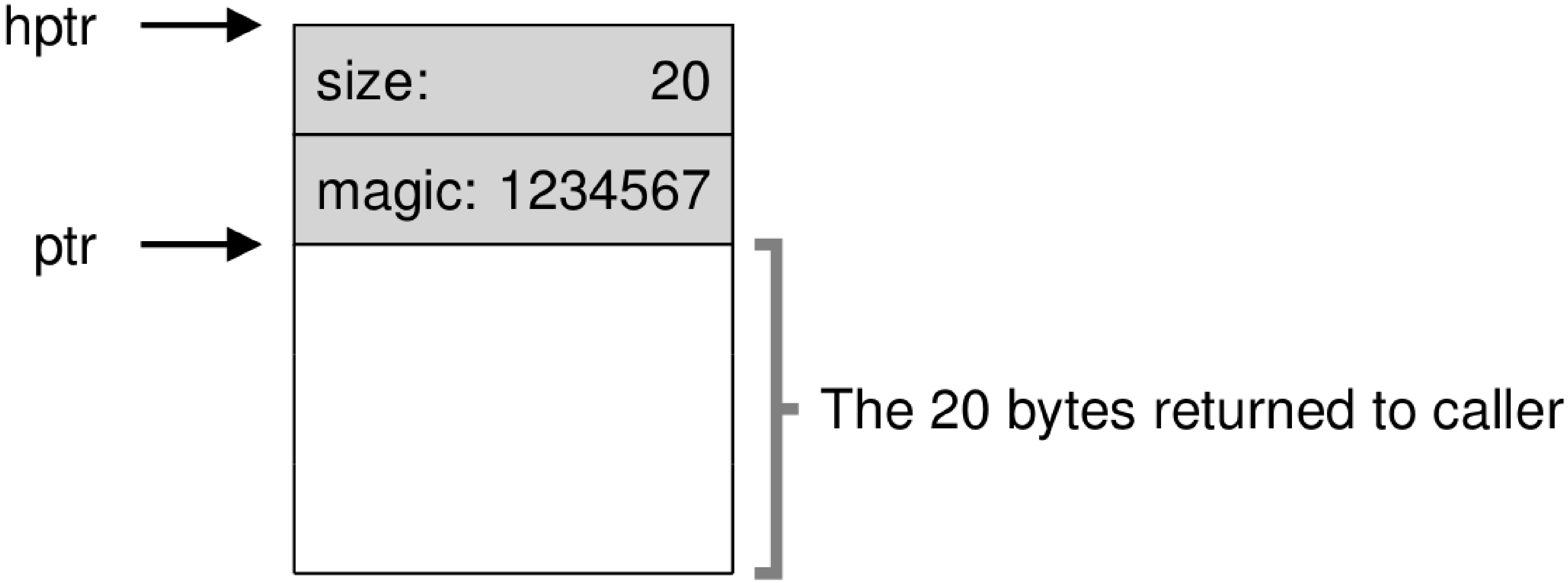
```
typedef struct __header_t {
     int size;
     int magic;
} header_t;
```

  - malloc(20)


The header used by malloc library
ptr
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

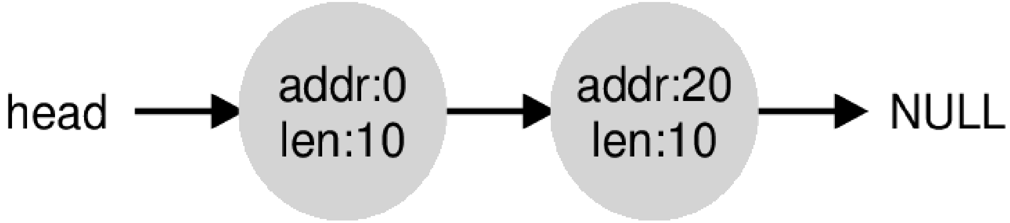
```
void free(void *ptr) {
    header_t *hptr = (void *)ptr - sizeof(header_t);
}
```

    - 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

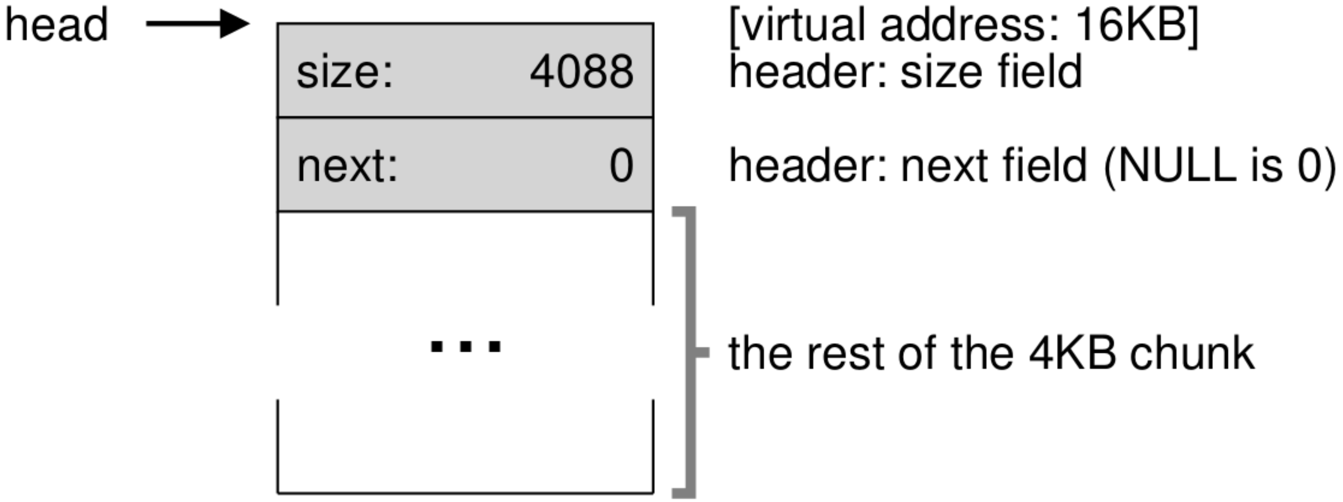
- Implementation of the Free List
  - Node in free list

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

Free Memory Management

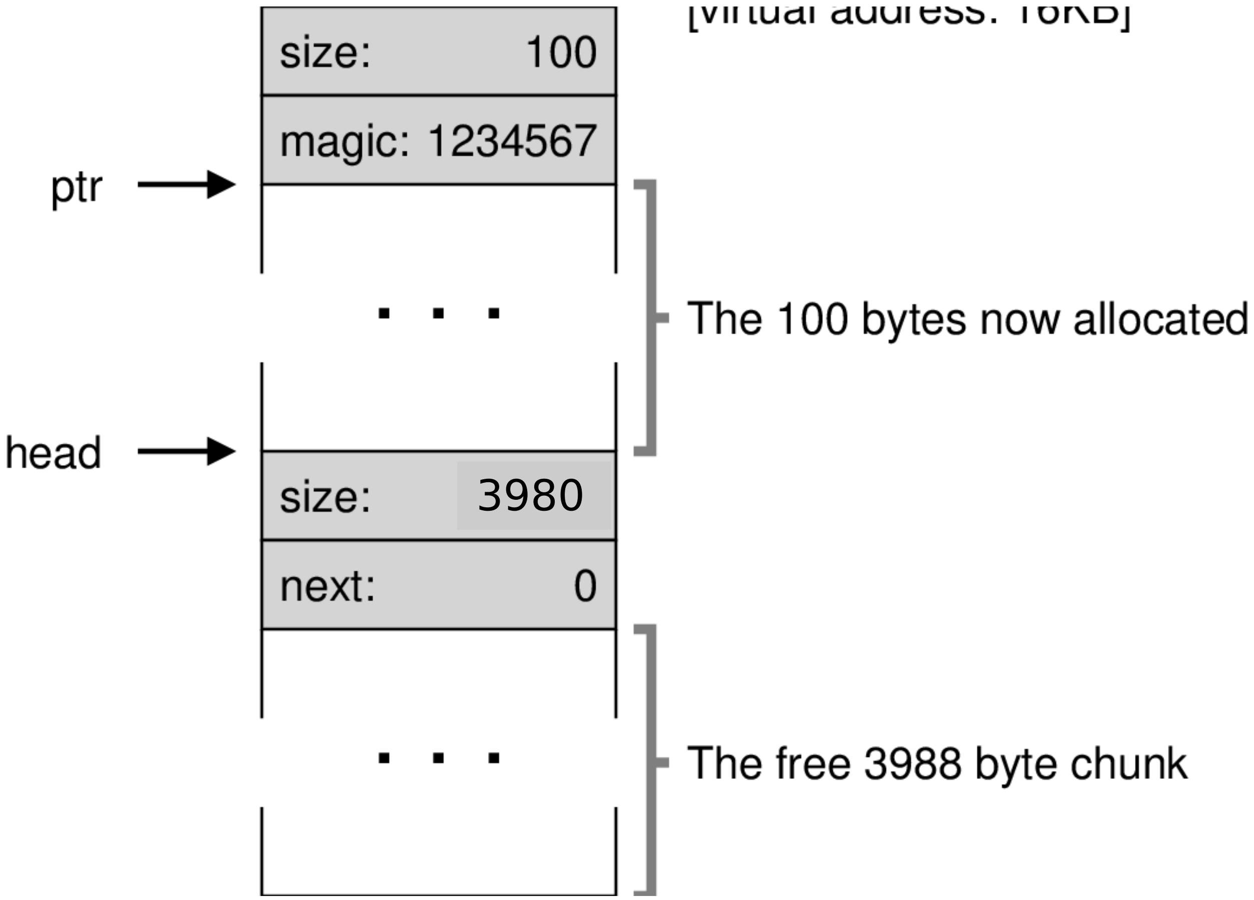
- Implementation of the Free List
  - Initialization (e.g. 4096)

```
// 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;
```

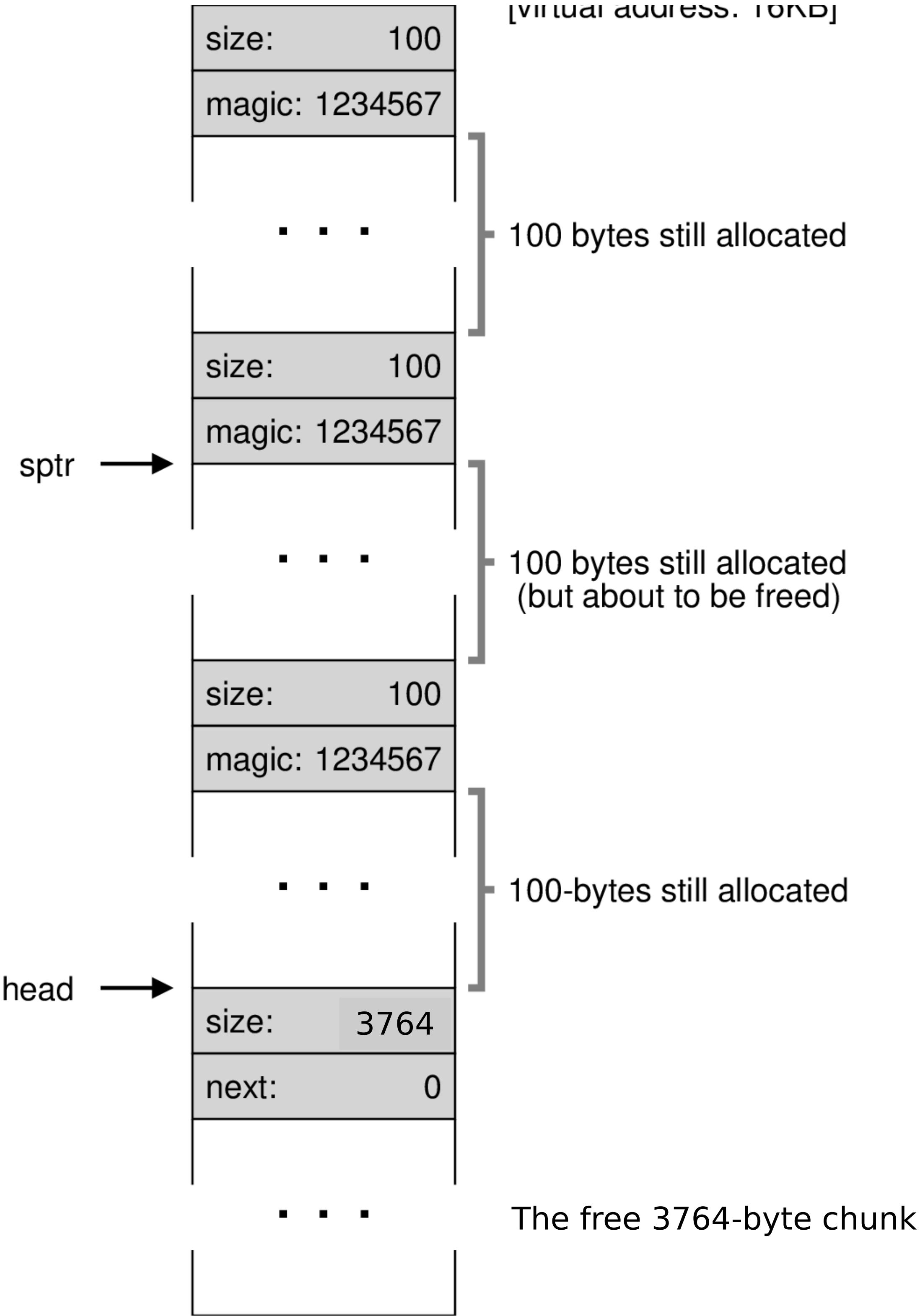


Free Memory Management

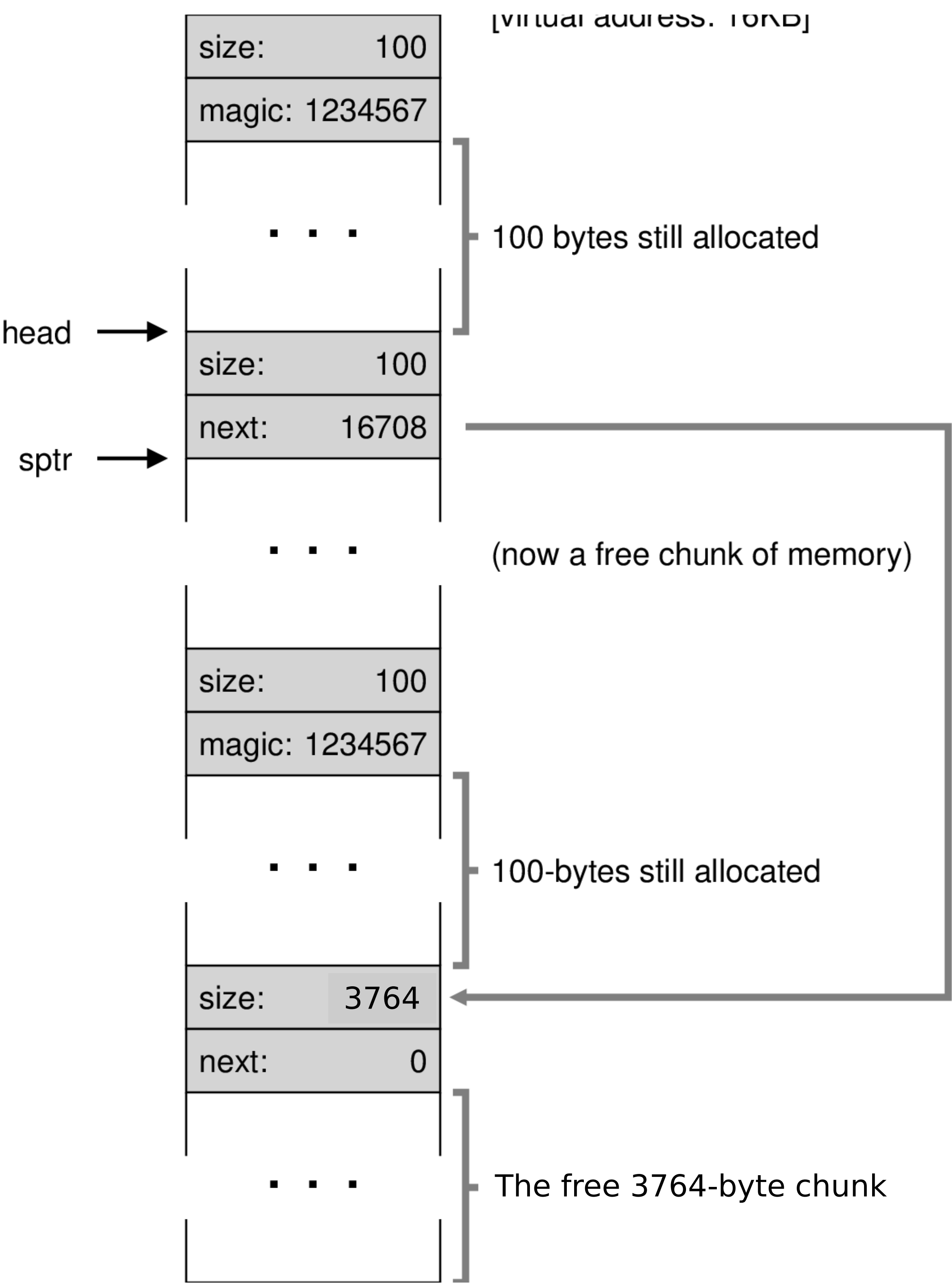
- Implementation of the Free List
  - malloc(100)



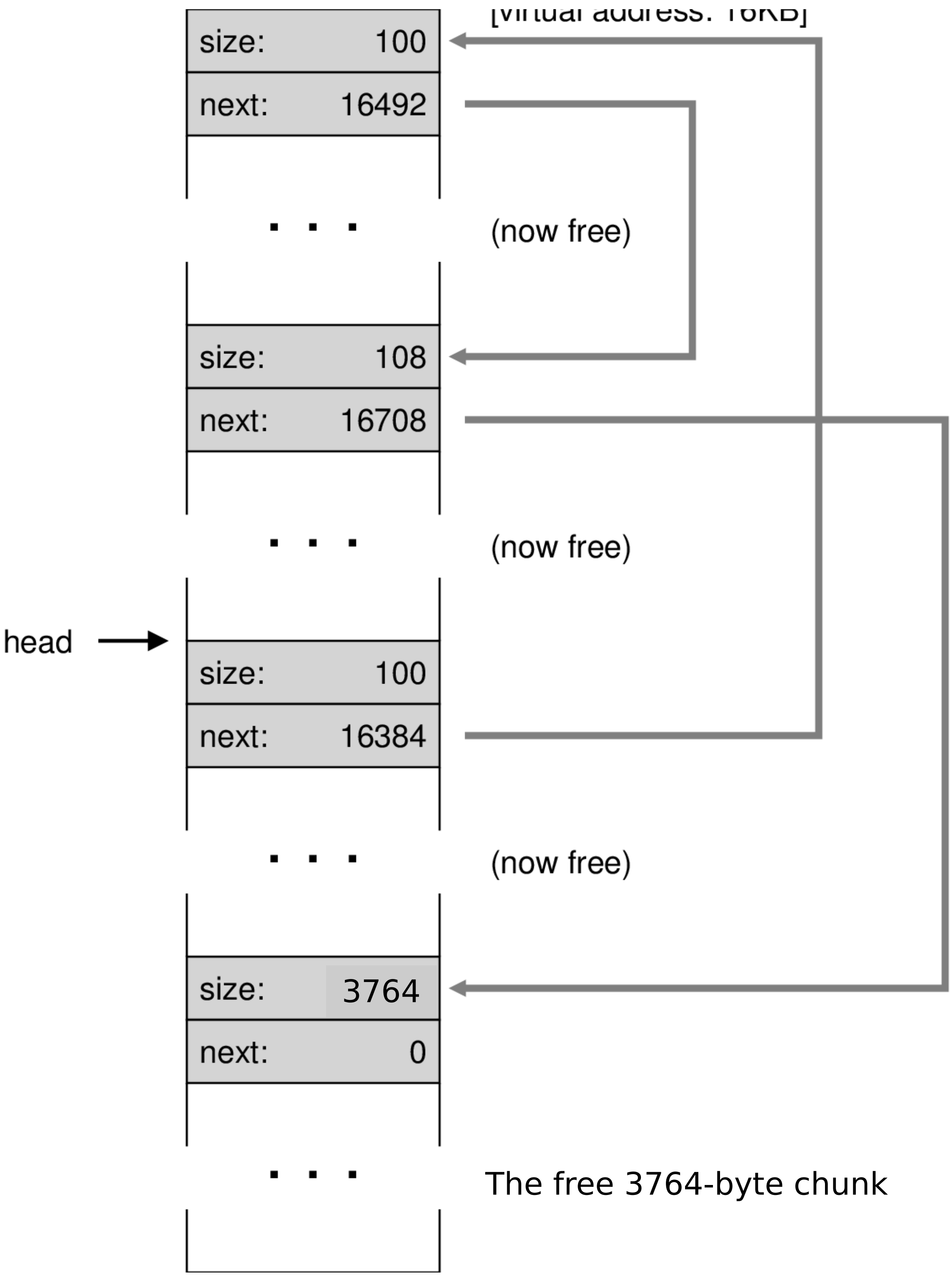
- malloc(100)*3


[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



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



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

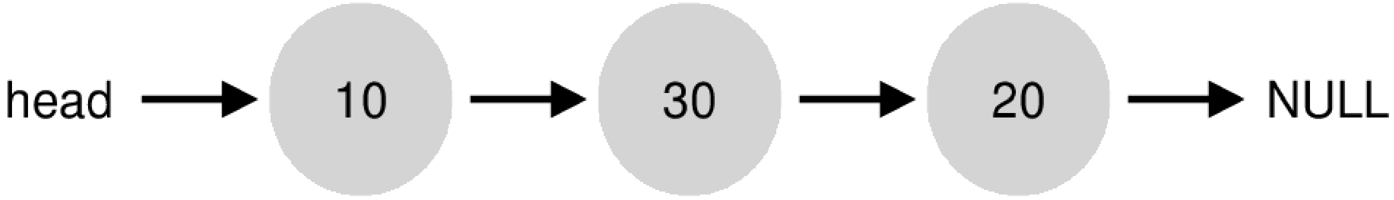
- High-level intelligence
  - How to find the proper nodes in the free list?
    - Less fragmentation
    - Fast allocation
  - Some simple strategies
    - The stream of allocation and free requests can be arbitrary
    - Any strategy could be arbitrarily bad/good

Free Memory Management

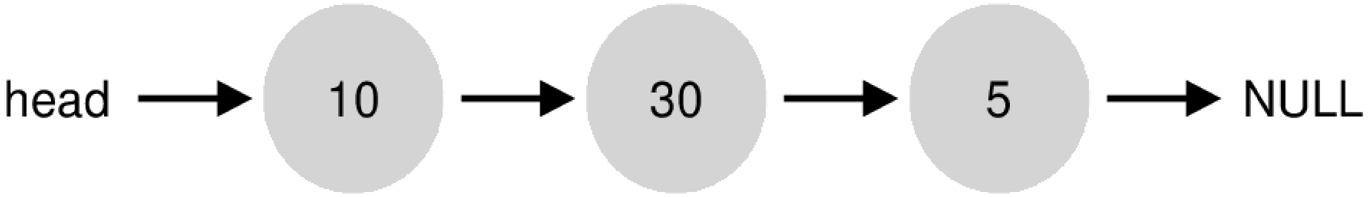
- Best Fit
  - Find the smallest feasible node
- Worst Fit
  - Find the largest feasible node
- First Fit
  - Find the first feasible node

Free Memory Management

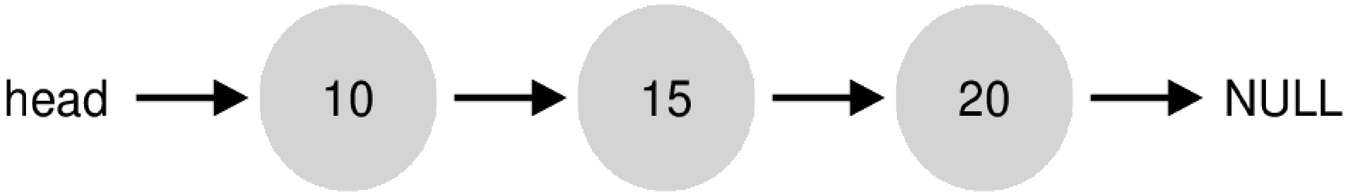
- Example
  -



  - Best fit

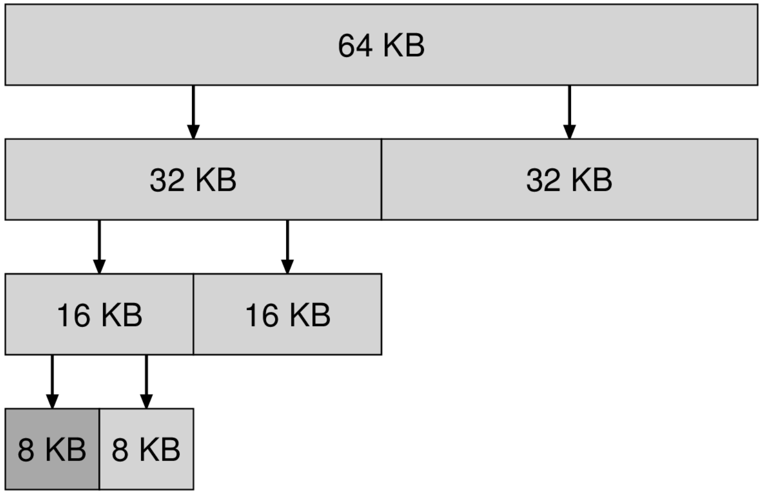


  - Worst fit



Free Memory Management

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


64 KB
32 KB
32 KB
16 KB
16 KB
8 KB
8 KB