CS 340
Computer Systems

#6: Memory Allocation and malloc

Feb. 2, 2023 · Wade Fagen-Ulmschneider

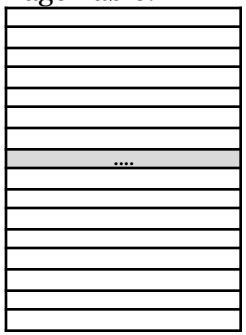
Memory Allocation

At a system level, the page table is a series of pointers to RAM (or other storage). From a process level, we organize our private page table to store data:

06/memory-addr.c

```
 5  int val;
 6  printf("&val: %p\n", &val);
 7
 8  void *ptr = malloc(0x1000);
 9  printf("&ptr: %p\n", &ptr);
10  printf(" ptr: %p\n", ptr);
11
12  void *ptr2 = malloc(0x1000);
13  printf("&ptr2: %p\n", &ptr2);
14  printf(" ptr2: %p\n", ptr2);
15
16  int arr[4096];
17  printf("&arr: %p\n", &arr);
18
19  return 0;
```

Page Table:



As a programmer, we talk about these different regions of memory as different "types" of memory:

_____ **Memory** | _____ **Memory**

Q1: What if we access memory beyond the end of our heap? (Or any other region not allocated in our page table?)

Memory Address Components:

Address:		

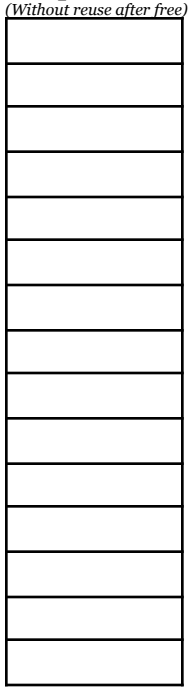
Efficient Use of Heap Memory

During the lifetime of a single process, we will allocate and free memory many times. Consider a simple program:

06/heap.c

```
 5  int *a = malloc(4096);
 6  printf("a = %p\n", a);
 7  free(a);
 8
 9  int *b = malloc(4096);
10  printf("b = %p\n", b);
11
12  int *c = malloc(4096);
13  printf("c = %p\n", c);
14
15  int *d = malloc(4096);
16  printf("d = %p\n", d);
17
18  free(b);
19  free(c);
20
21  int *e = malloc(5000);
22  printf("e = %p\n", e);
23
24  int *g = malloc(10);
25  printf("g = %p\n", g);
26
27  int *g = malloc(10);
28  printf("g = %p\n", g);
```

Heap v1: *(Without reuse after free)*



Heap v2: *(With reuse after free)*

Q2: How much memory is used if we **do not** reuse memory?

Q3: How much memory is used with **optimal** reuse of memory?

- What happens to our memory over time?

- When we have "holes" in our heap, how do we decide what hole to use?