Lecture 19: Memory Allocation Implementation

CSE 29: Systems Programming and Software Tools

Aaron Schulman (Shalev)

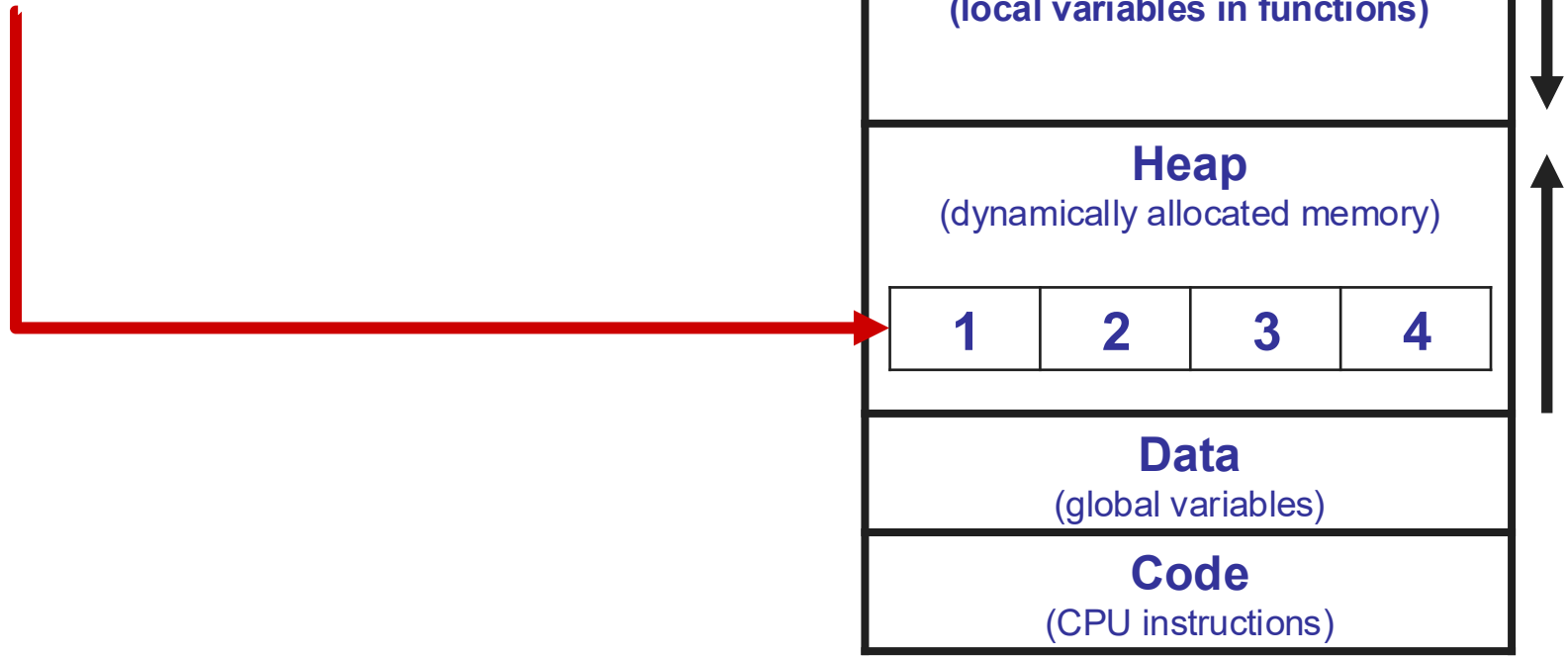


Logistics

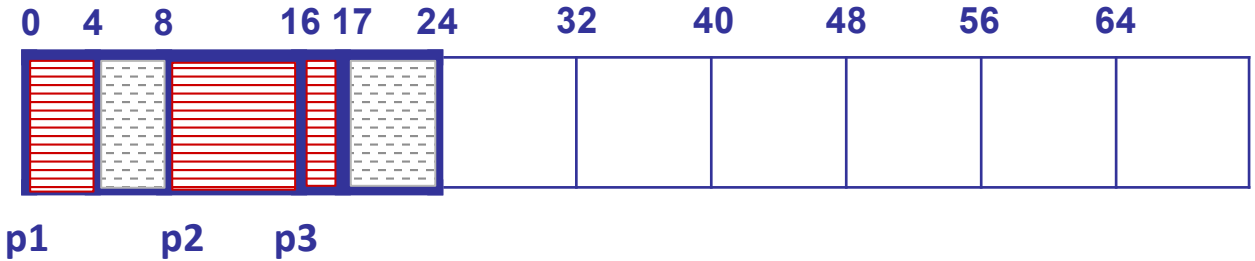
- *Beginning a new phase of the class today:*
  - First half of the class was focused on **low-level programming in C**
  - Second half of the class is focused on **how a computer system works ("systems")**
    - » How does memory allocation work?
    - » How does information get stored on a disk?
    - » Intro to Networking and OS: How do you build a webserver?

What does malloc() do?

```
int iarr* = malloc(len * sizeof(int));
```



Malloc Implementation: Heap structure



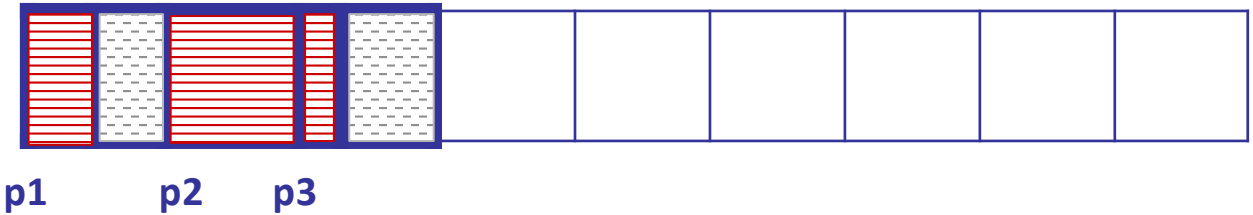

```
int* p1 = malloc(4); // 0x00
int* p2 = malloc(8); // 0x08
int* p3 = malloc(1); // 0x10
```

Alignment Requirement for 64-bit machine:
8 bytes (every heap allocation should be aligned to 8 bytes)
Why do we need alignment?

Malloc Implementation: Requirements

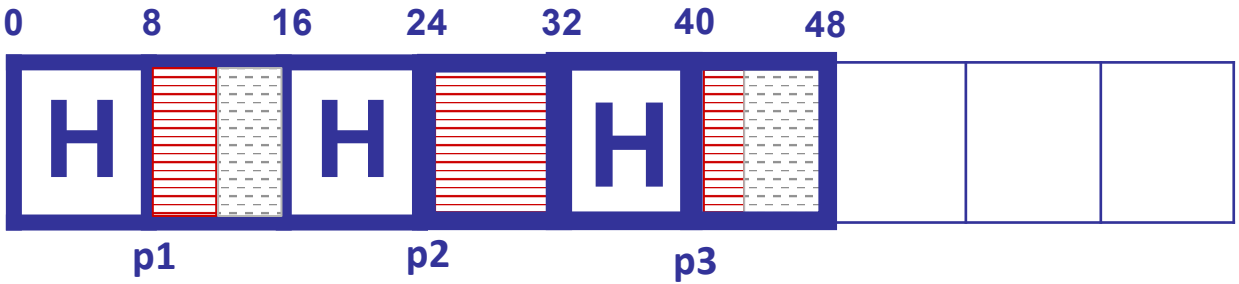
Requirements:

1. Only use information in the heap to manage the heap
2. Can not modify previously allocated blocks



We **also** need to store metadata also in the heap about each allocation

Headers: Storing metadata on the heap



Add 8 byte headers before each allocation on the heap

```
int* p1 = malloc(4); // 0x00
int* p2 = malloc(8); // 0x08
int* p3 = malloc(1); // 0x10
```

Header implementation

- Header contains a **length** and a **flag bit** representing free/allocated
  - Even number: representing a *free block*
  - Odd number: representing a *malloc'd, allocated, busy block*

- In both cases, the number with LSB set to 0 is the block size in bytes

- header = 33 → allocated block of 32 bytes (24 byte payload)
- header = 108 → free block of 108 bytes (100 byte payload)

- "size of block" = full size including header
- Always make block sizes divisible by 8 (or 8-byte aligned)

How do you decide what part of the heap an allocation should be in?



- https://elixir.bootlin.com/glibc/glibc-2.32/source/malloc/malloc.c

Why use this malloc?

This is not the fastest, most space-conserving, most portable, or most tunable malloc ever written. However it is among the fastest while also being among the most space-conserving, portable and tunable. Consistent balance across these factors results in a good general-purpose allocator for malloc-intensive programs.

Review Quiz

1. What is the heap?
2. What is heap alignment and why does the heap need to be aligned?
3. What information must the heap store to track allocations?