Section 17
Implicit List Heap Allocator

CS 107A, Autumn 2021
Andrew Benson (adbenson@)

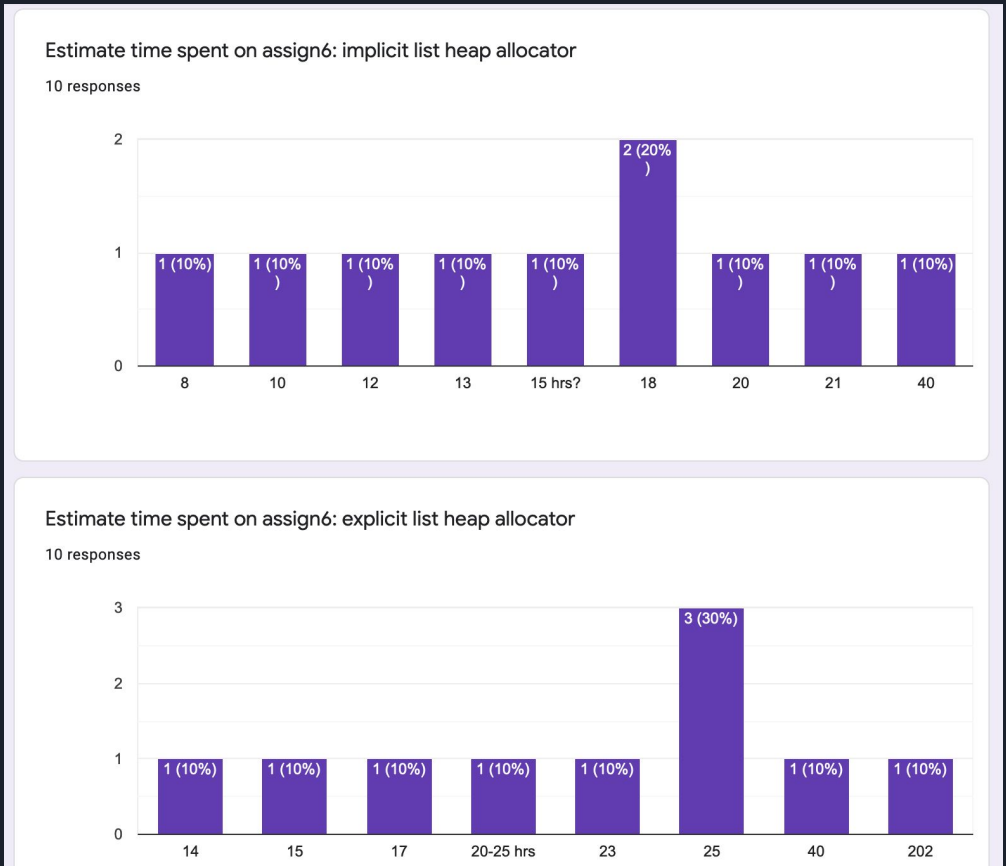
Don’t forget to start recording

Bird's Eye View

Day	Week 9 Monday	Tuesday	Wednesday	Thursday	Friday	Week 10 Monday	Tuesday	Wednesday
CS 107A		Section: Implicit List Heap Allocators		Section: Explicit List Heap Allocators			Section: Beyond CS 107	
CS 107	Lecture: Optimization		Lab: Code and Memory Optimization		Lecture: Additional Topics	Lecture: Wrap-up		No Lab
CS 107 assignments	assign5 due, assign6 out							assign6 due Friday

assign6: Heap Allocator

assign6: Heap Allocator



Announcements

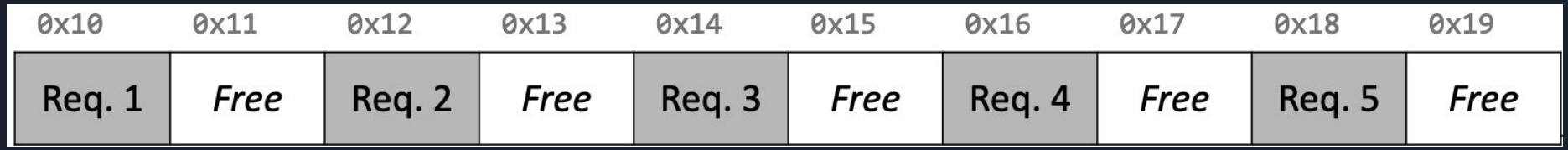
- Feedback on weekend OH
- Please start heap allocator early, it's big
- Heap Allocator Office Hours
  - Friday (?)
  - No OH next week, Slack remains ~asynchronous~
  - Aim to finish implicit heap allocator this week

Agenda

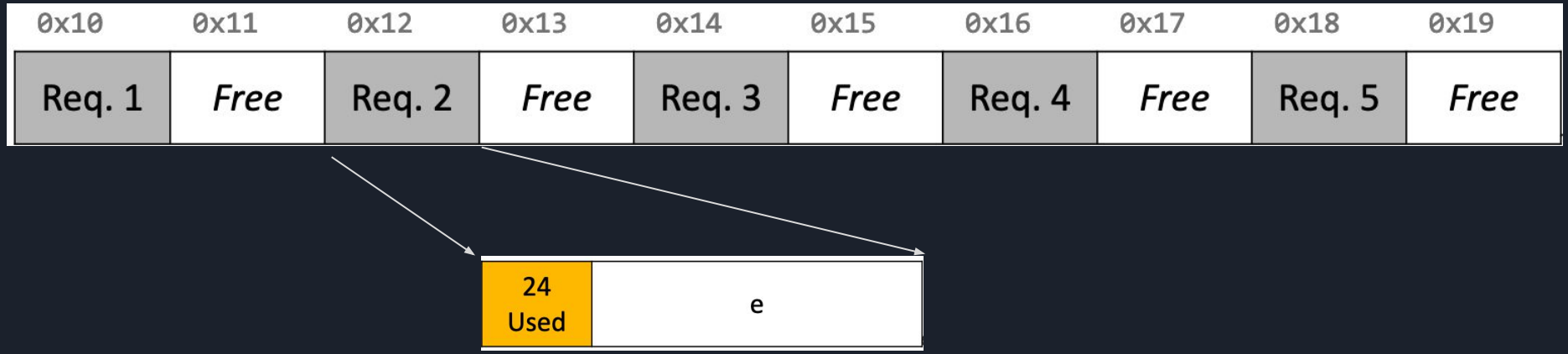
- Implicit List Terminology
- Headers
- Blocks
- Implicit List

Implicit List Terminology

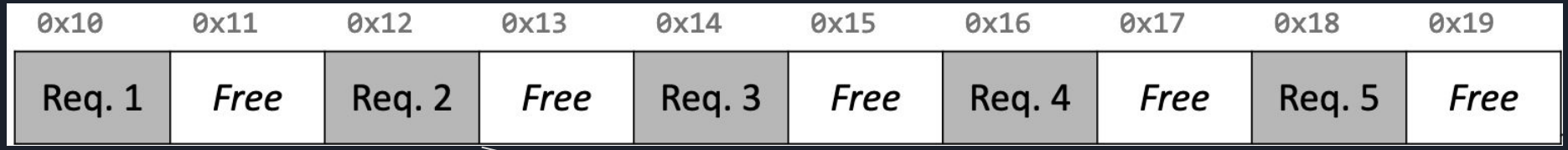
The heap is made up of an implicit list of blocks



Each block consists of a header and a payload

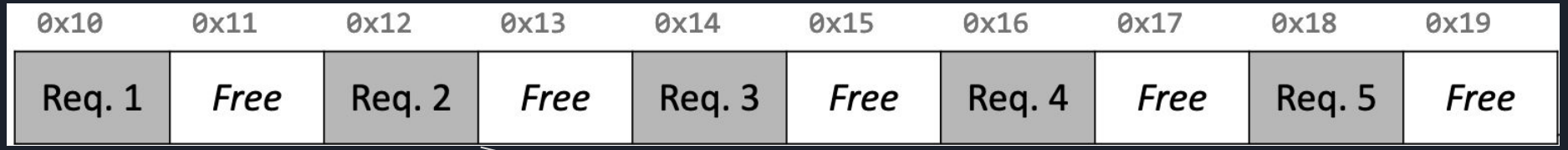


Each header consists of the size and the allocated bit



0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxxxxx
0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxx00a

Implicit List operations work at 3 levels



Work within the implicit list

Work within a block

0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxxxxx
0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxxxxxx 0bxxxxx00a

Work within a header

Headers

Headers

- Required by the handout:
  - 8 bytes, and must be able to hold any size `malloc` takes in
    - So what type should a header be?

Headers

- Required by the handout:
  - 8 bytes, and must be able to hold any size `malloc` takes in
  - `typedef size_t header_t;`

Headers

- Required by the handout:
  - 8 bytes, and must be able to hold any size `malloc` takes in
  - `typedef size_t header_t;`
  - Sizes must be a multiple of 8 (use the `ALIGNMENT` constant)
  - An “allocated bit” must be stored within as well
  - Wait, we have to hold 64 bits for the size, and also 1 bit for whether it’s allocated - do we need 65 bits?

Some size_t numbers that are multiples of 8

0	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000
8	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00001000
16	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00010000
24	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00011000

Some size_t numbers that are multiples of 8

0	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000
8	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00001000
16	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00010000
24	0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00000000 0b00011000

Use the LSB as the allocated bit

- Arbitrarily decide that 0 = FREE, 1 = ALLOC (you can reverse this if you like, as long as you always follow the rule!)

25

```
0b00000000 0b00000000 0b00000000 0b00000000
0b00000000 0b00000000 0b00000000 0b00011001
```

- This has an actual value of 25. This value is meaningless.
- We know that when we extract out the size, the LSB should be 0.
- We can extract the actual size by zeroing out the LSB.
- We can extract the allocated bit by reading only the LSB.

24

```
0b00000000 0b00000000 0b00000000 0b00000000
0b00000000 0b00000000 0b00000000 0b00011000
```

1

```
(allocated bit = ALLOC)
```

Header-level Operations

- Write:
  - `bool is_free(header_t *header);` (group)
  - `void set_alloc(header_t *header);` (group)
  - `size_t get_size(header_t *header);` (partner)
- You may assume:
  - `typedef size_t header_t;`
  - the constants `FREE` and `ALLOC` are defined (one of them is 1, the other is 0)

Blocks

Pointers within a Block

header pointer (same as block pointer)

payload pointer (only used to return to user - you will never write to the payload for the implicit list!)

next block pointer

Block-level Operations

- Write:
  - `void *header2payload(header_t *header);` (group)
  - `header_t *payload2header(void *payload);` (partner)
  - `header_t *next_block(header_t *header);` (partner)
  - Is it possible to write `prev_block`?
- You may assume:
  - The existence of the header-level functions we wrote earlier (don't examine the values of headers all over again!)

Implicit List

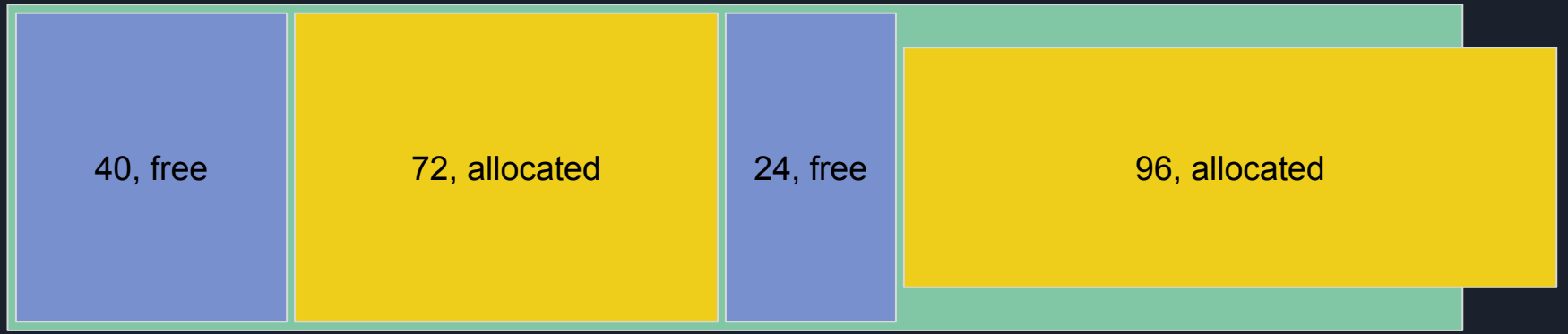
The implicit list must fit the entire heap exactly

Note: at the very beginning, the heap is just one big giant free block

This is after we've done some `mallocs` and `frees`

Overlaps NOT allowed! (header pointer + size must point to next block)

The last block must take up exactly the heap remainder (last block pointer + size must be end of heap)



Don’t treat these slides as a list of validity characteristics, by the way. Sizes must be a multiple of 8, too, for example.

`next_block`, which we already wrote, helps us iterate through blocks

Implicit list-level Operations

- Write:
  - `size_t count_blocks(header_t *start);` (group)
  - `header_t *find_block_of_64(header_t *start);` (group)
- You may assume:
  - The existence of the block-level functions we wrote earlier (don't parse pointers all over again!)