Section 18
Explicit List Heap Allocator

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

Don’t forget to start recording

Bird's Eye View

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

Announcements

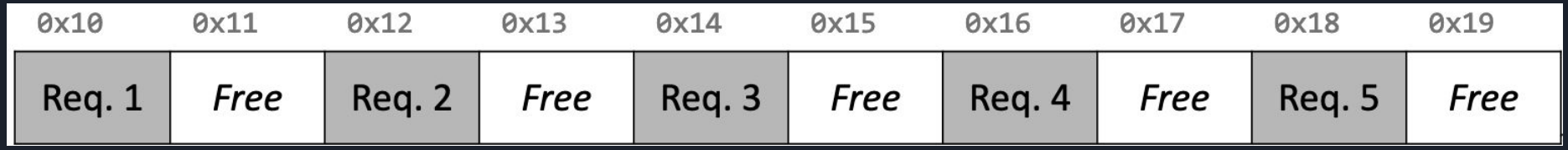
- Friday OH will happen
- Final Review Session Preferences
- Obligatory plea to start heap allocator
- Walkthrough posted, forgot to say on Tuesday

Agenda

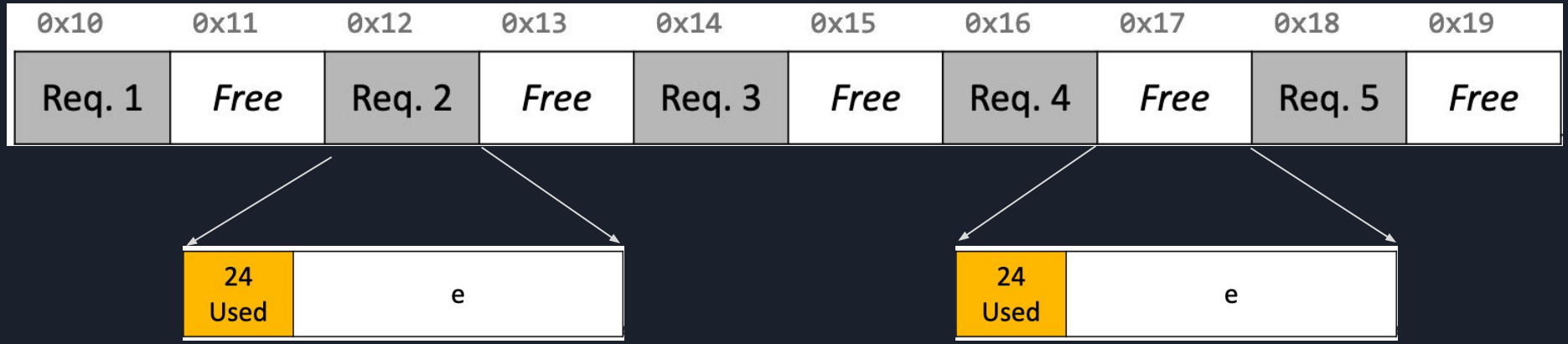
- Explicit List Overview
- Headers
- Free Payloads
- Blocks
- Implicit List

Explicit List Overview

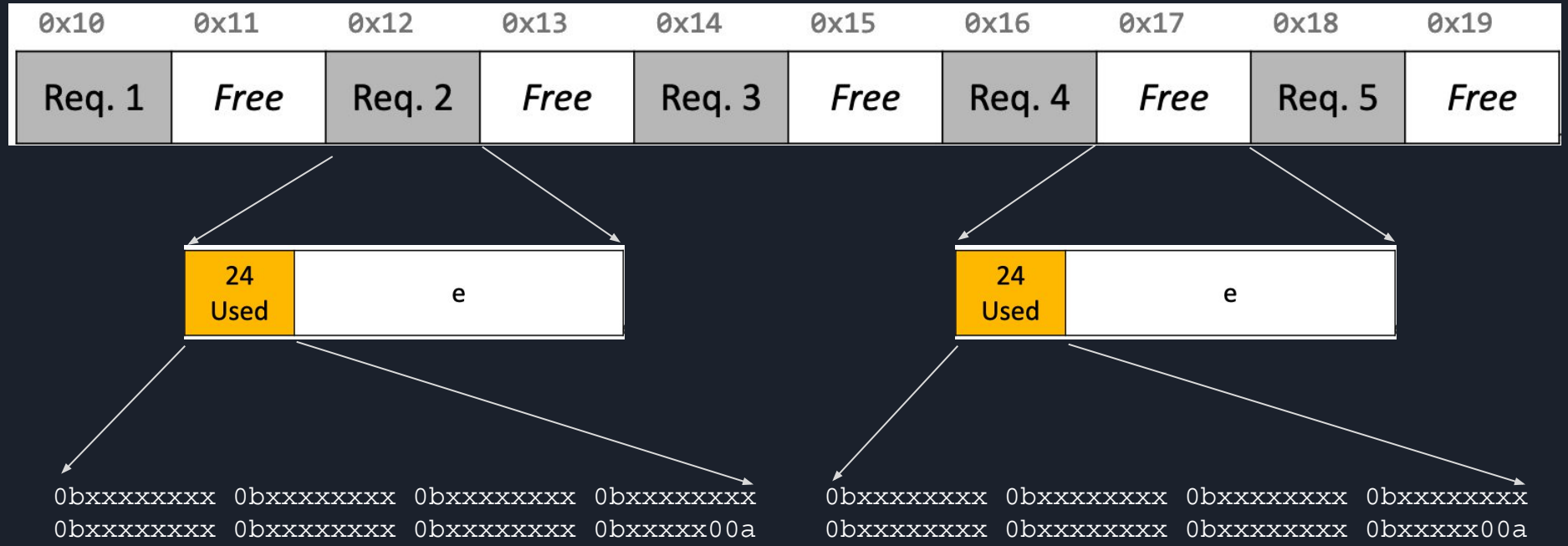
[Unchanged from implicit list, not a typo!]
The heap is made up of an implicit list of blocks



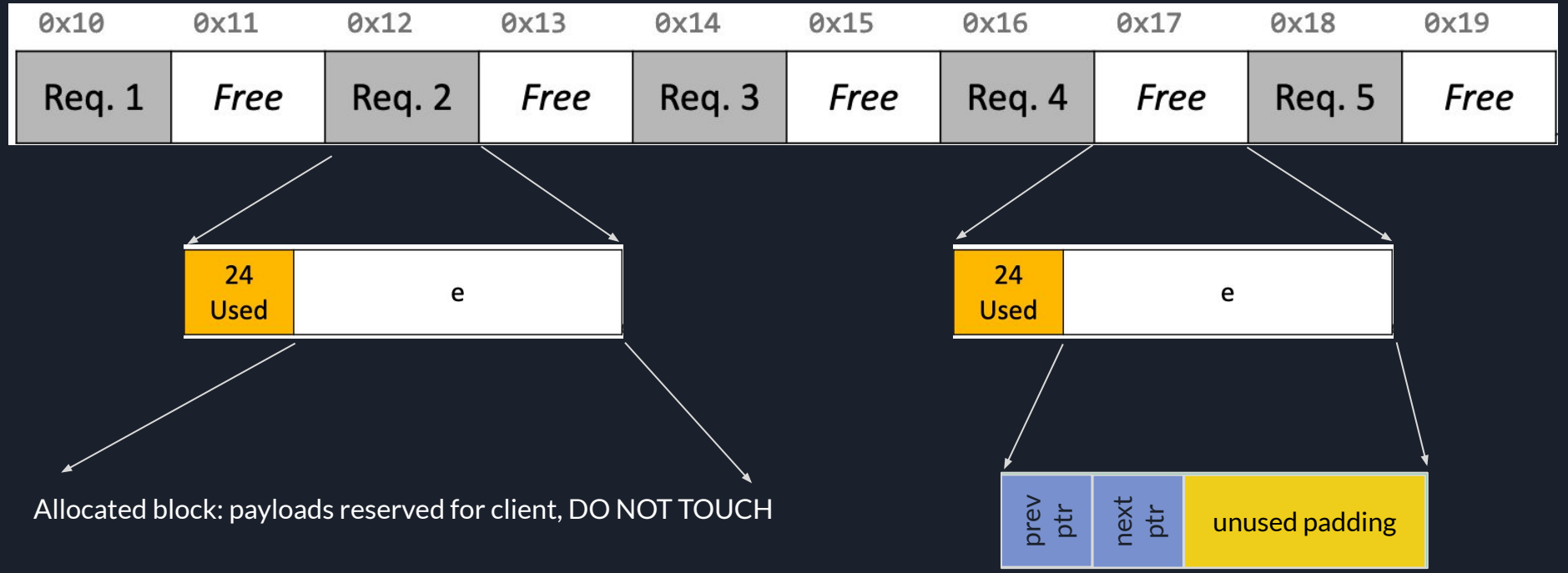
[Also unchanged from implicit list]
Each [free/allocated] block consists of a header and a payload



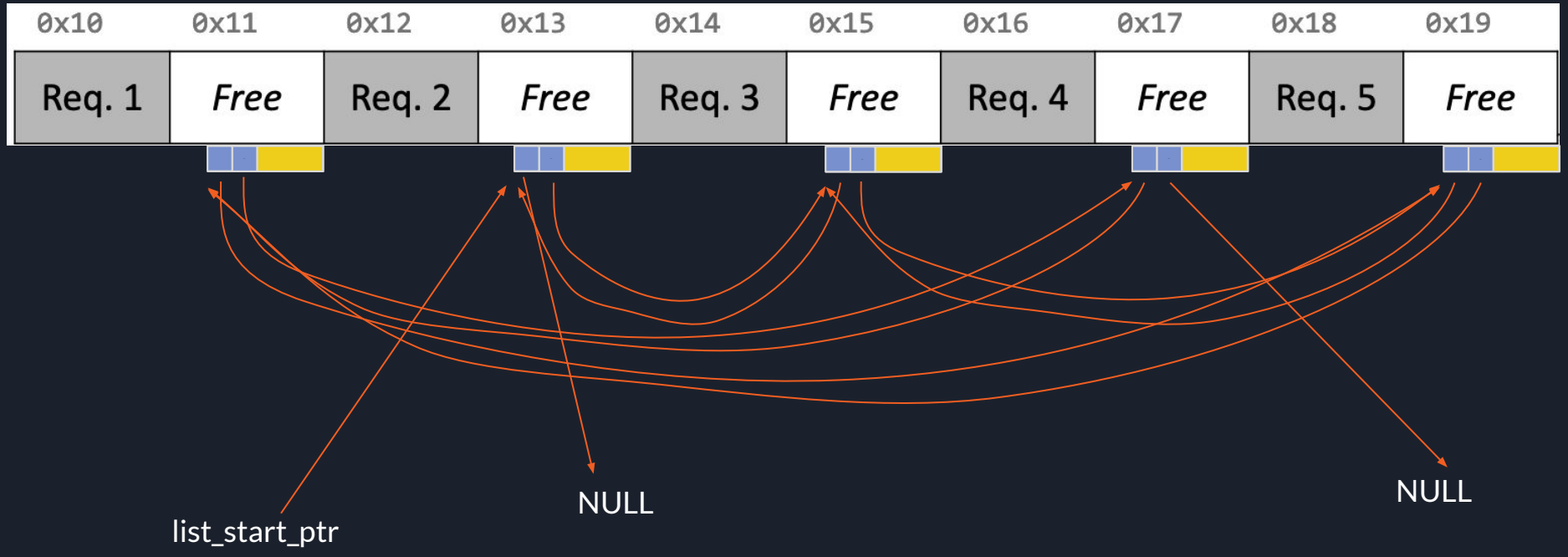
[Still unchanged from implicit list]
Each header is the bitwise combination of a size and an allocated bit



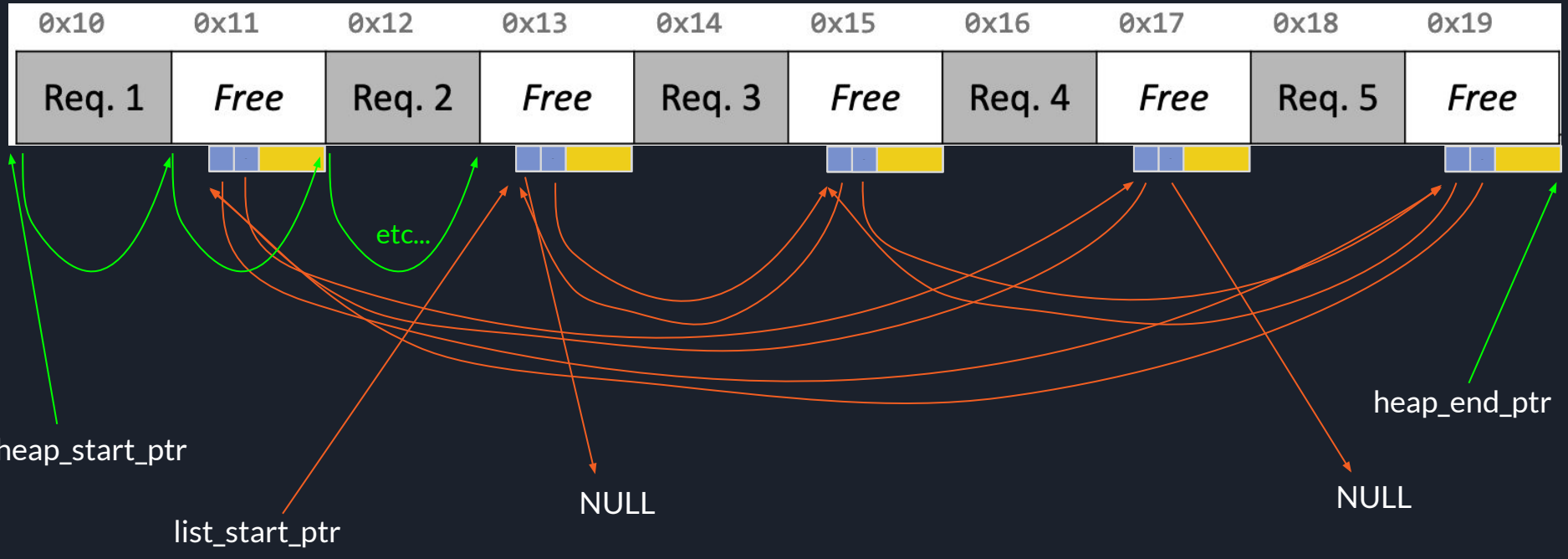
[NEW for explicit list]
Payloads differ between free and allocated blocks



[NEW for explicit list]
Payloads differ between free and allocated blocks



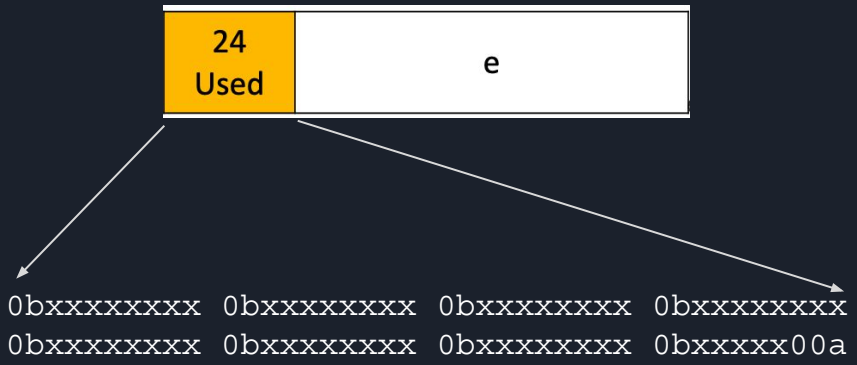
Two ways to traverse the heap! Implicit (all blocks) and explicit (only free blocks)



Headers

Reuse from implicit list heap allocator!

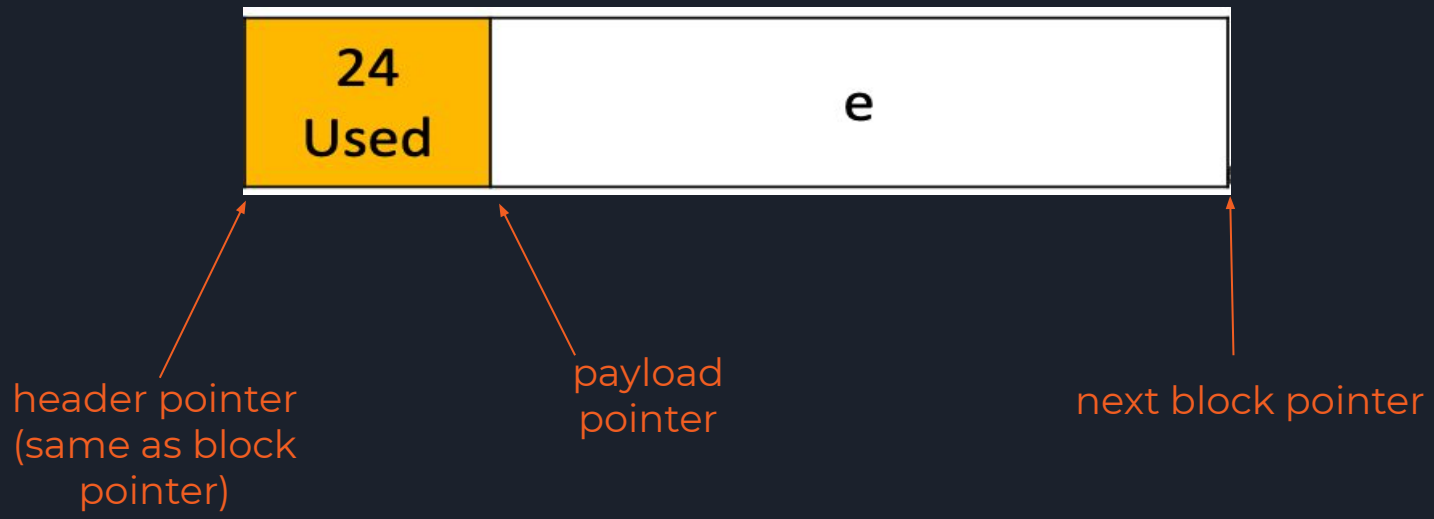
- `bool is_free(header_t *header);`
- `bool is_alloc(header_t *header);`
- `void set_free(header_t *header);`
- `void set_alloc(header_t *header);`
- `size_t get_size(header_t *header);`



Blocks

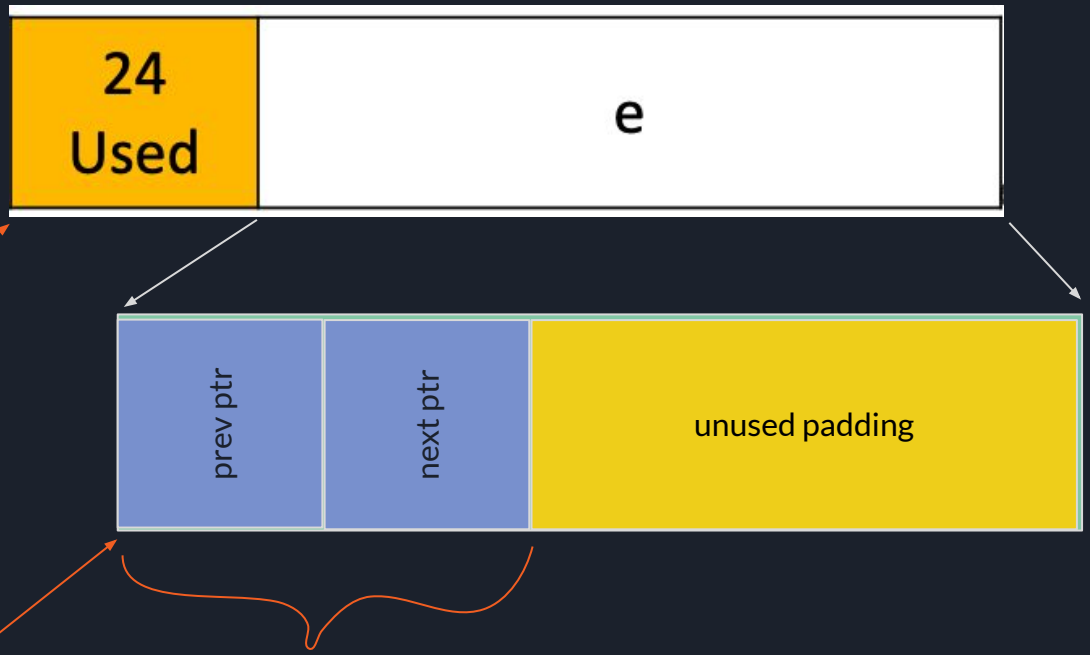
Reuse from implicit list heap allocator!

- `void *hdr2pl(header_t *header);`
- `header_t *pl2hdr(void *payload);`
- `header_t *next_block(header_t *header);`

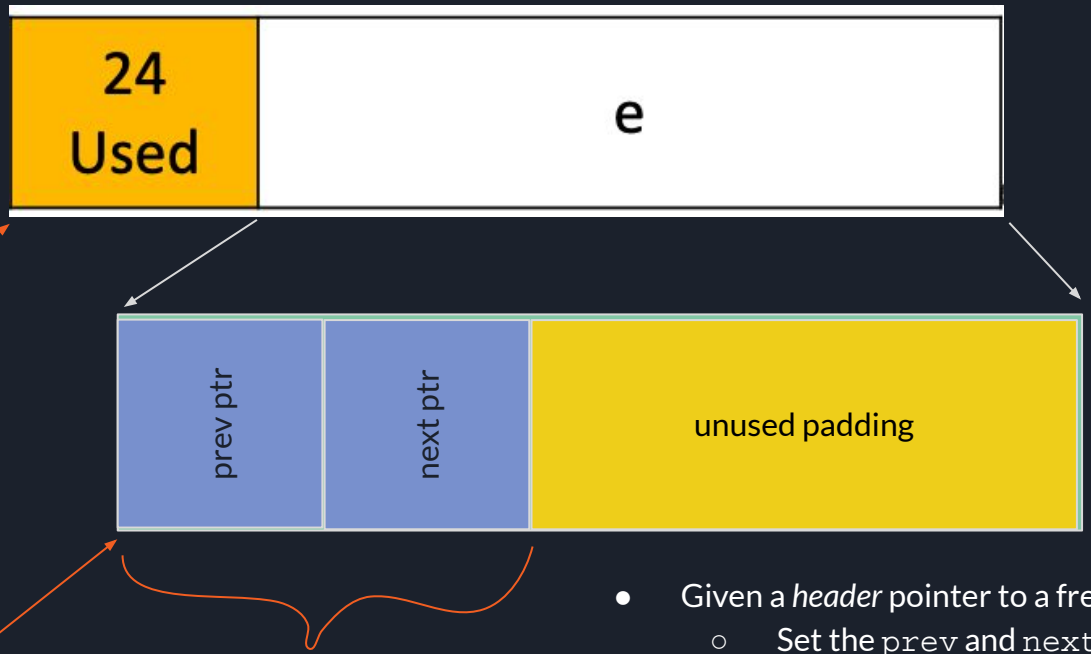


Free Payloads

Pointers within a Block



Pointers within a Block

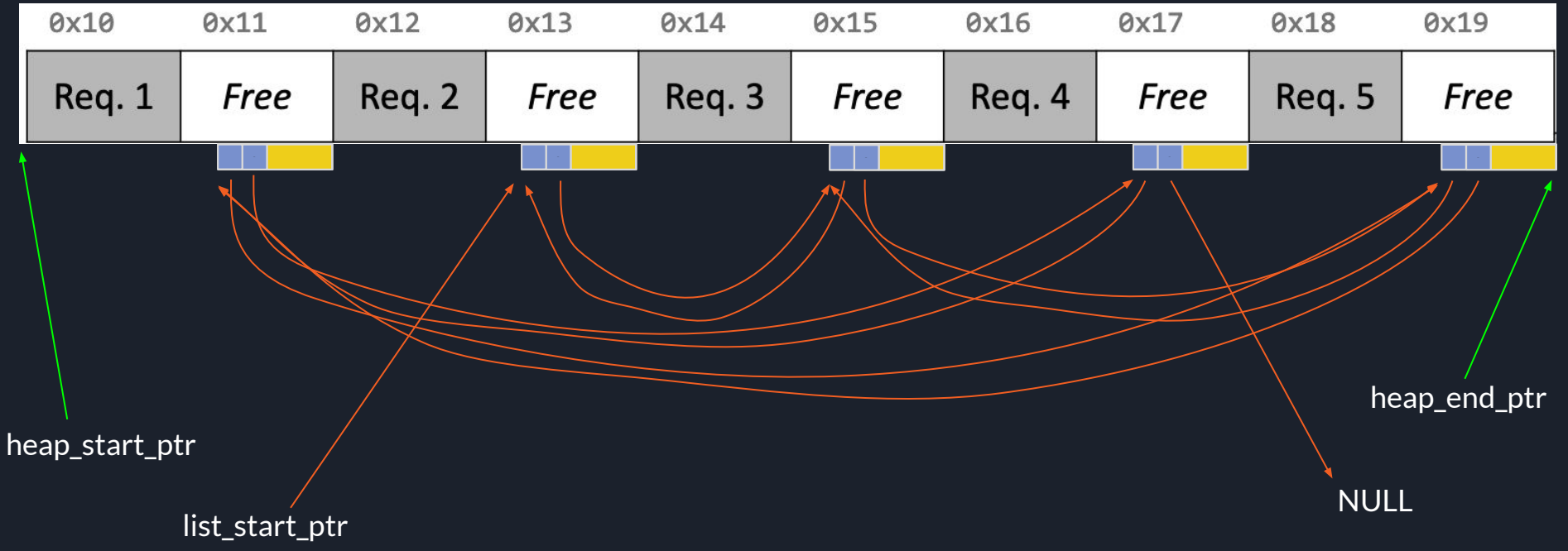



```
struct node {
  struct node *prev;
  struct node *next;
};
```

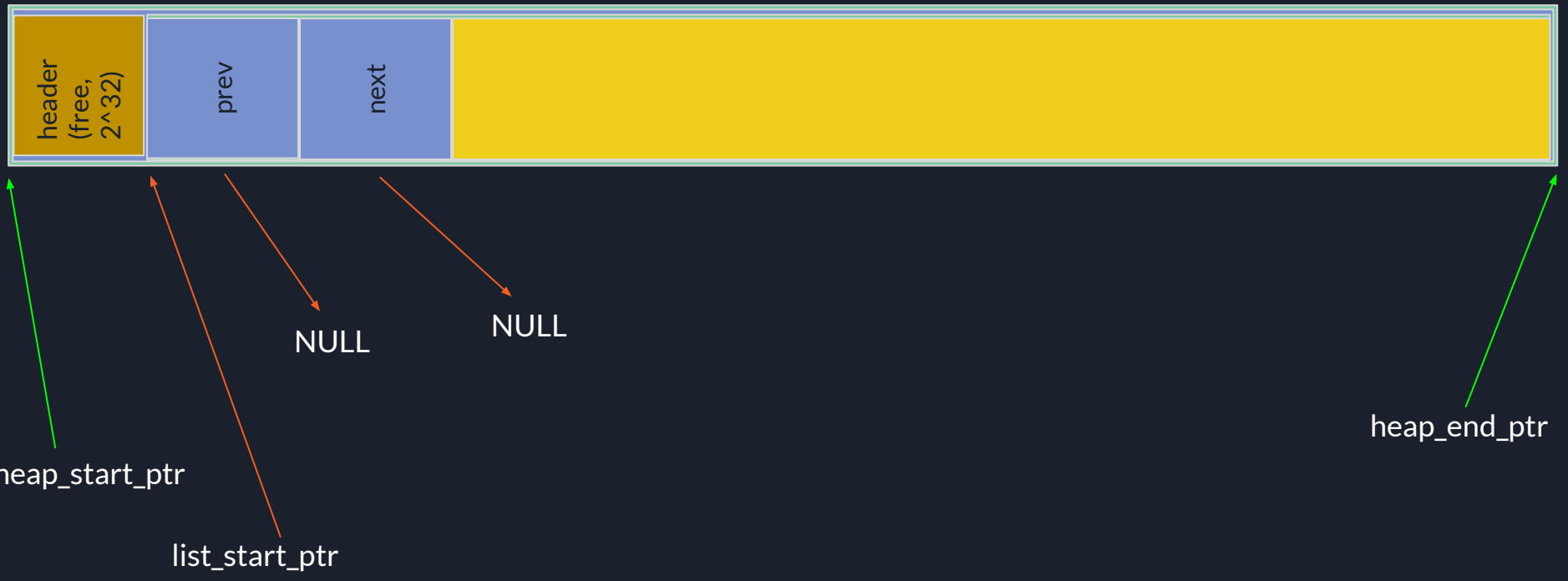
- Given a *header* pointer to a free block, how would you
  - Set the `prev` and `next` pointers to NULL?
  - Check if the next block's prev pointer points back to this block?
  - Check both `prev` and `next` for a self-loop?

Doubly Linked Lists

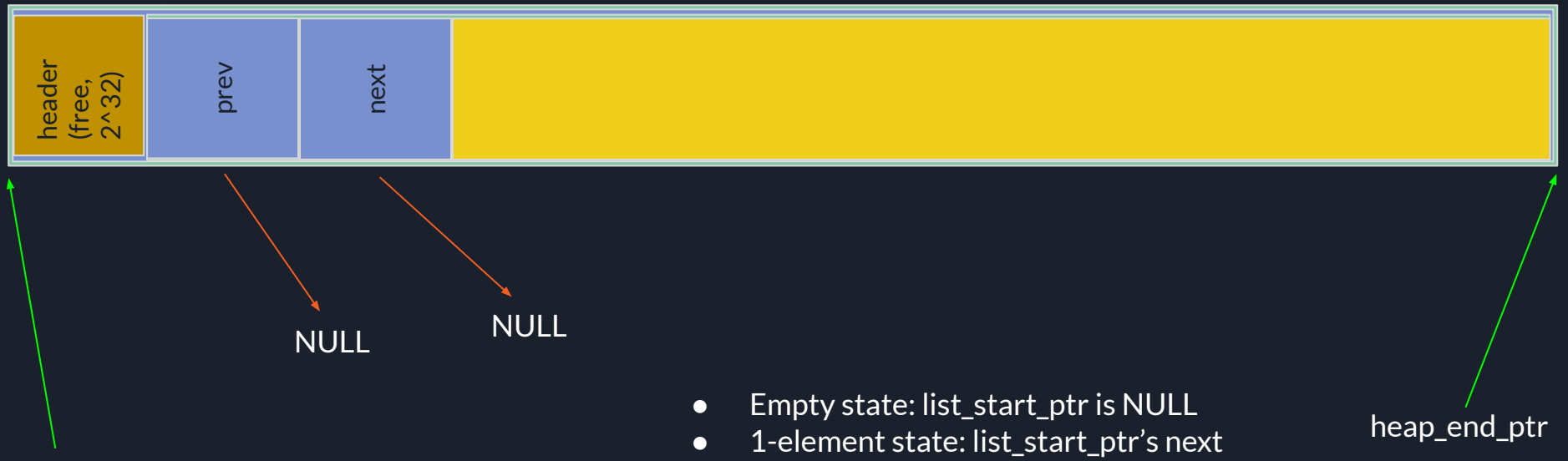
This is a doubly linked list, a la CS 106B



Initial state (after `my_init`) (length 1)



Possible Intermediate Empty State (length 0)



- Empty state: list_start_ptr is NULL
- 1-element state: list_start_ptr's next and prev are both NULL
- 2+ element state: list_start_ptr's next is not NULL

Explicit list-level Operations

- Iterate through the explicit list
- `void add_free_block(struct node *new_free_payload);`
- `void detach_free_block(struct node *free_payload);`

(Why?) Malloc algorithm:

- Foreach free block in explicit list:
  - If big enough
    - Detach from free list
    - Split block or take entire block
    - Add split free block back in, if there is one
    - Return