Recitation 2.2

Outline

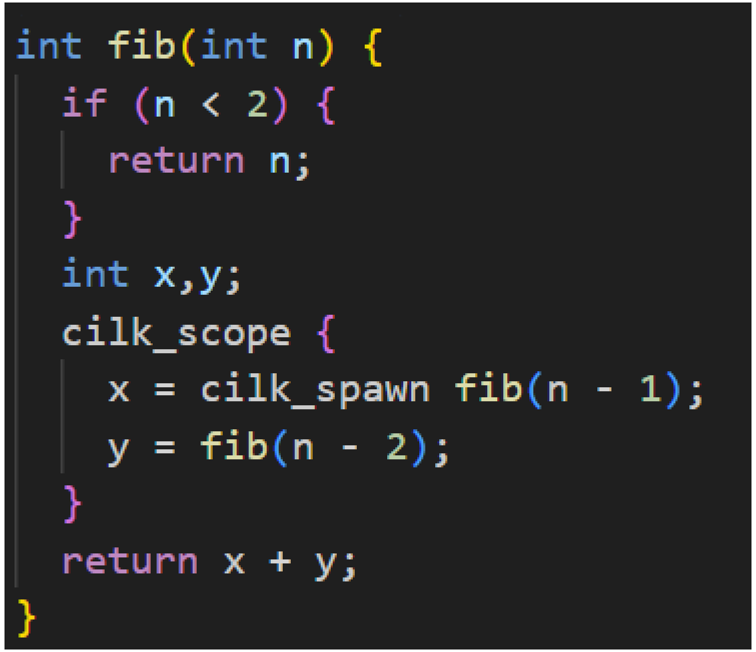
- A quick recap of project 2 optimizations
- An example write-up
- Some remarks from homework 4
- Homework 5 Introduction
- Attendance
- Remaining time is used as “lab time” to make progress in hw5/ project 2

- **Announcement**: The class will be held via Zoom on April 2 (Wednesday)

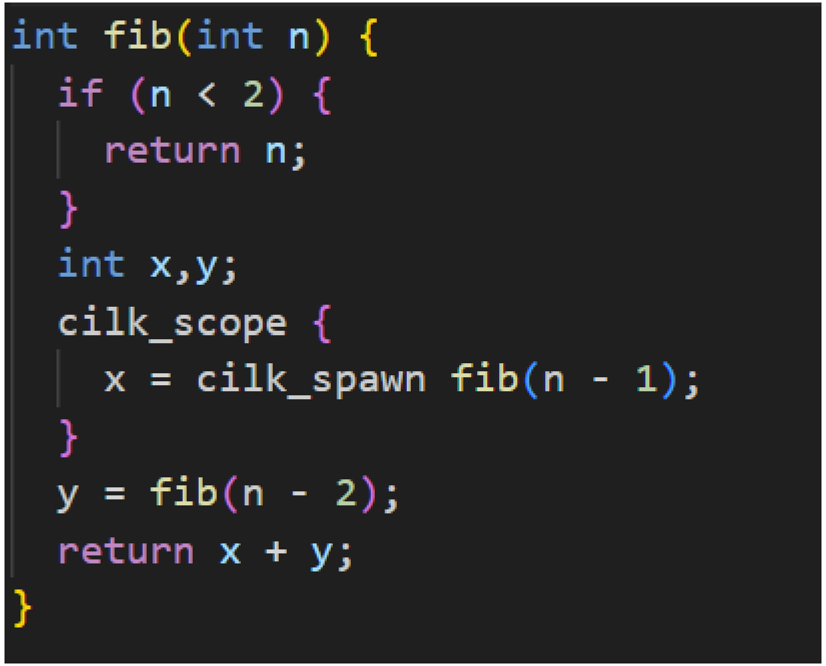
An example of a good project write up

- *Opens a pdf*

Common hw4 mistake – Which one is parallelized?

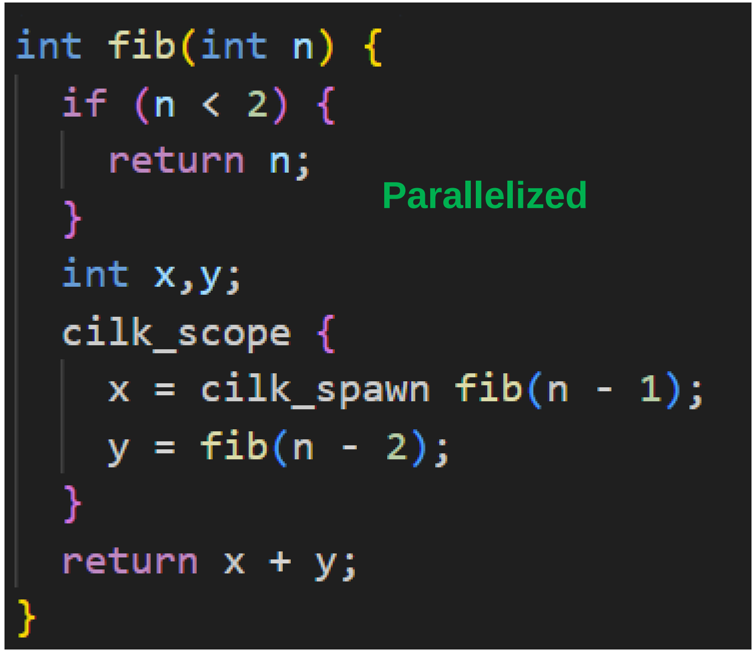


```c
int fib(int n) {
  if (n < 2) {
    return n;
  }
  int x,y;
  cilk_scope {
    x = cilk_spawn fib(n - 1);
    y = fib(n - 2);
  }
  return x + y;
}
```



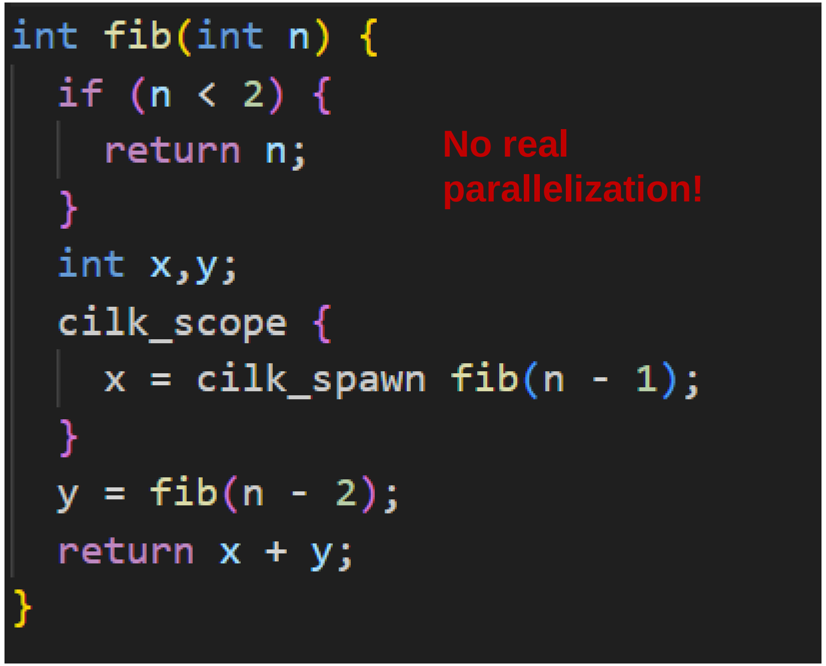
```c
int fib(int n) {
  if (n < 2) {
    return n;
  }
  int x,y;
  cilk_scope {
    x = cilk_spawn fib(n - 1);
  }
  y = fib(n - 2);
  return x + y;
}
```

Common mistake in HW4



```
int fib(int n) {
  if (n < 2) {
    return n;
  }
  int x,y;
  cilk_scope {
    x = cilk_spawn fib(n - 1);
    y = fib(n - 2);
  }
  return x + y;
}
```

Parallelized

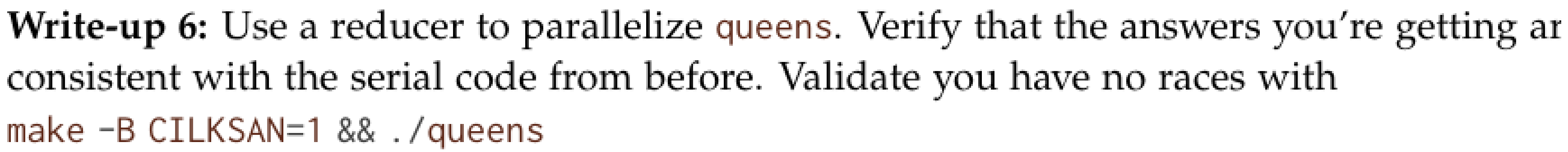


```
int fib(int n) {
  if (n < 2) {
    return n;
  }
  int x,y;
  cilk_scope {
    x = cilk_spawn fib(n - 1);
  }
  y = fib(n - 2);
  return x + y;
}
```

No real parallelization!

Answering Write-ups

- Your code might not be perfect. Don’t just refer me to Git, but show proofs of execution for write-ups like the following:



Write-up 6: Use a reducer to parallelize `queens`. Verify that the answers you’re getting ar
consistent with the serial code from before. Validate you have no races with
`make -B CILKSAN=1 && ./queens`

Malloc and Free

Malloc, Free, and Realloc

```
void* addr = malloc(size_t size)
```

- Allocates a chunk of memory of size `size`

```
void free(void* addr)
```

- Frees the allocated chunk of memory starting at `addr`

Free Lists

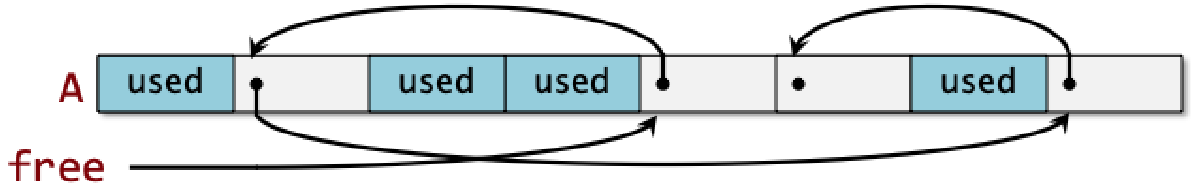
Free Lists

- Keeps track of deallocated memory
- Allows us to reuse memory
- Most memory allocators use a freelist of some sort
- Can implement as a singly linked list as seen in lecture

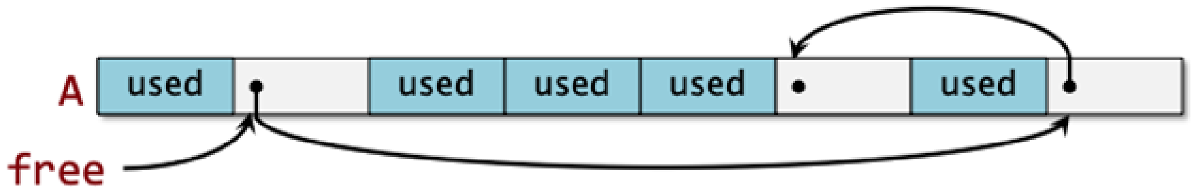
Allocating Memory w/ Free Lists
(fixed size blocks)

Free-List: Allocating

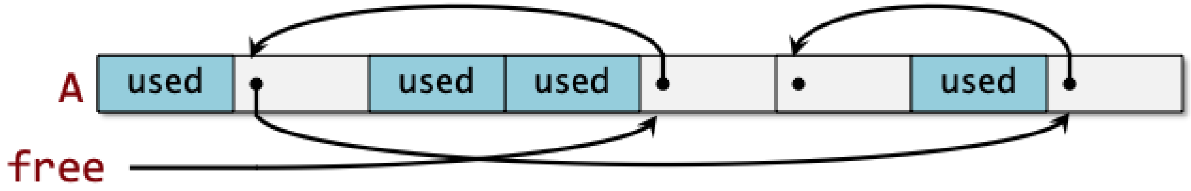
Before:



After:



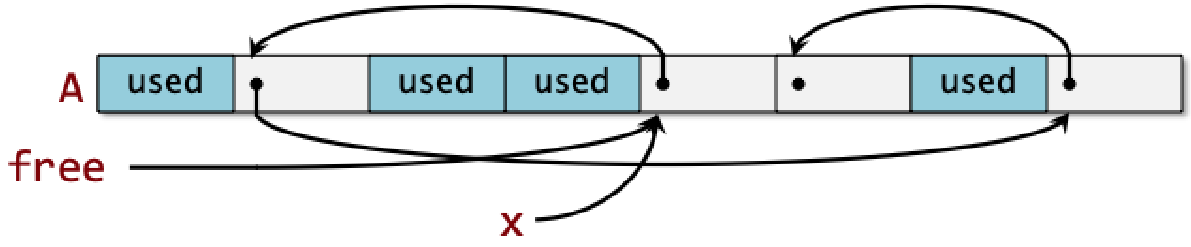
Free-List: Allocating



Allocate 1 object

```
x = free;
free = free->next;
return x;
```

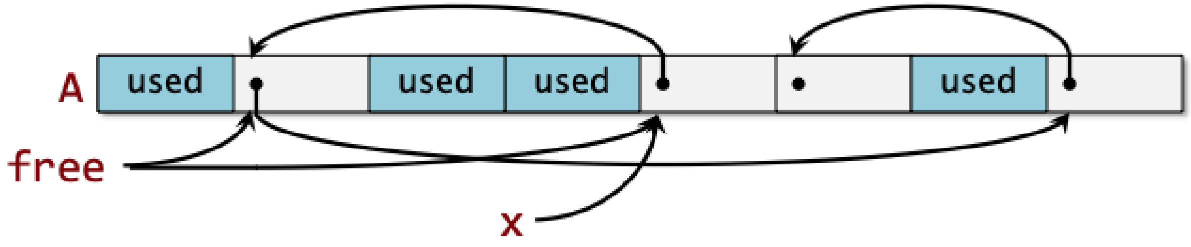
Free-List: Allocating



Allocate 1 object

```
x = free;
free = free->next;
return x;
```

Free-List: Allocating

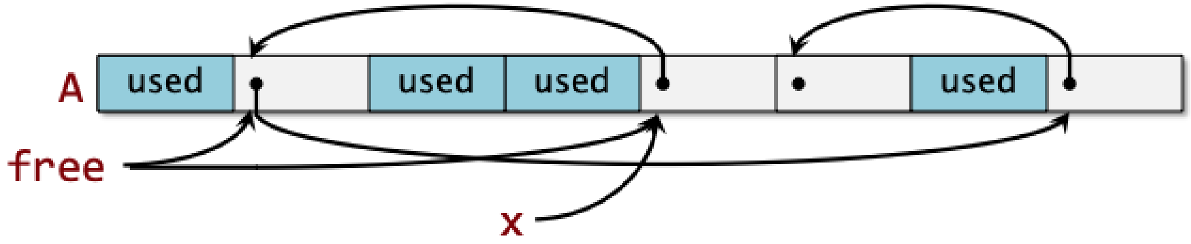


Allocate 1 object

```
x = free;
free = free->next;
return x;
```

Should check `free != NULL`.

Free-List: Allocating

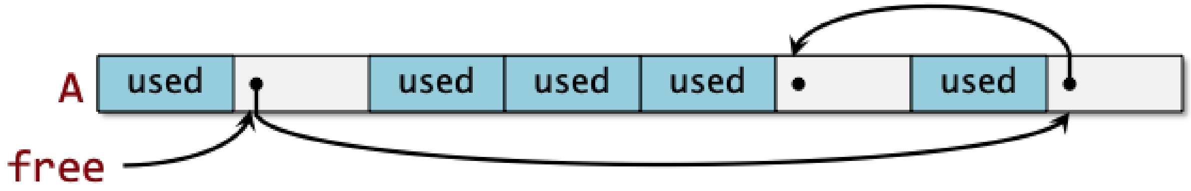


Allocate 1 object

```
x = free;
free = free->next;
return x;
```

Should check
`free != NULL`.

Free-List: Allocating



Allocate 1 object

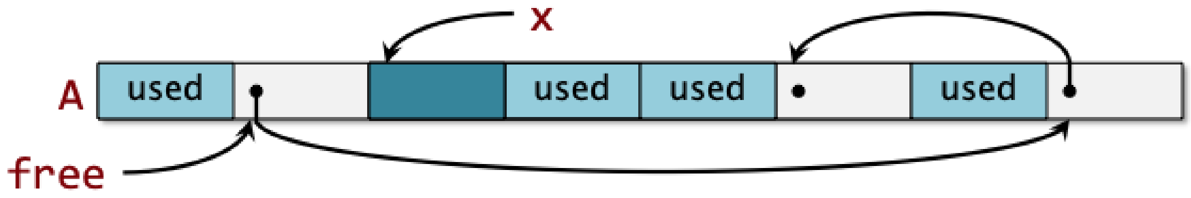
```
x = free;
free = free->next;
return x;
```

Freeing Memory w/ Free Lists

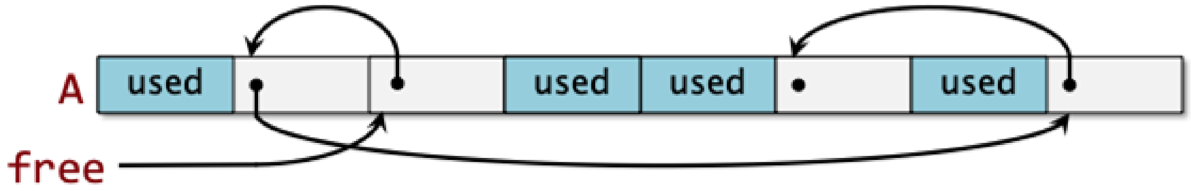
(fixed size blocks)

Free-List: Deallocating

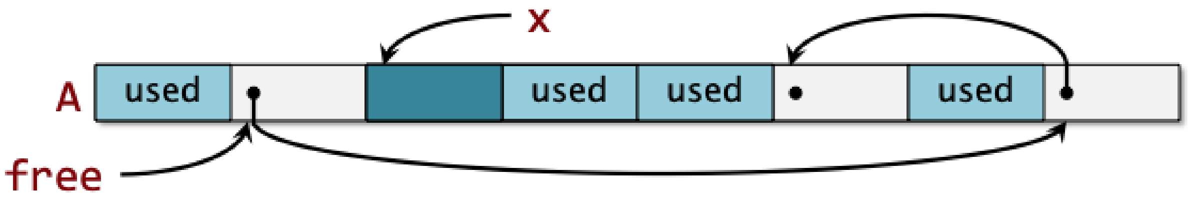
Before:



After:



Free-List: Deallocating



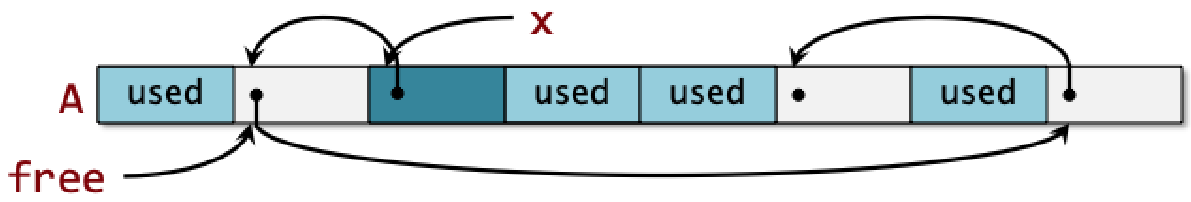
Allocate 1 object

```
x = free;
free = free->next;
return x;
```

free object x

```
x->next = free;
free = x;
```

Free-List: Deallocating



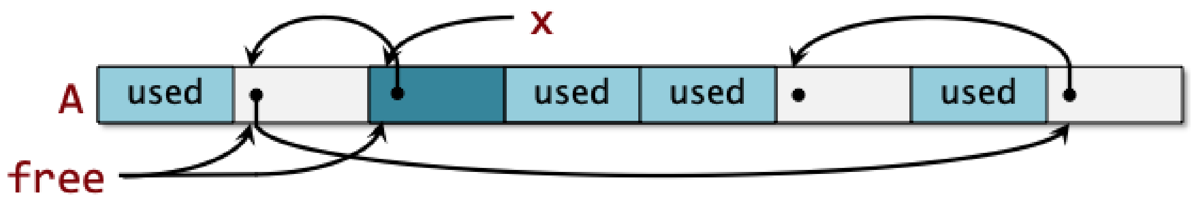
Allocate 1 object

```
x = free;
free = free->next;
return x;
```

free object x

```
x->next = free;
free = x;
```

Free-List: Deallocating



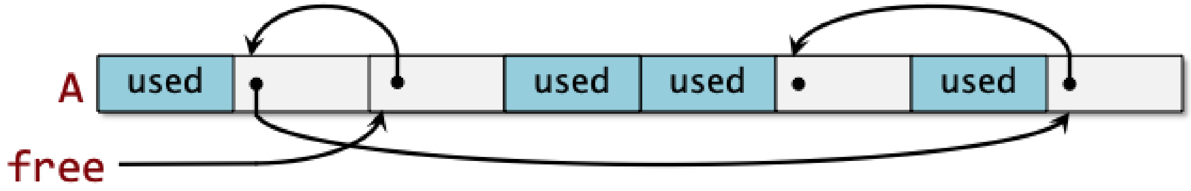
Allocate 1 object

```
x = free;
free = free->next;
return x;
```

free object x

```
x->next = free;
free = x;
```

Free-List: Deallocating



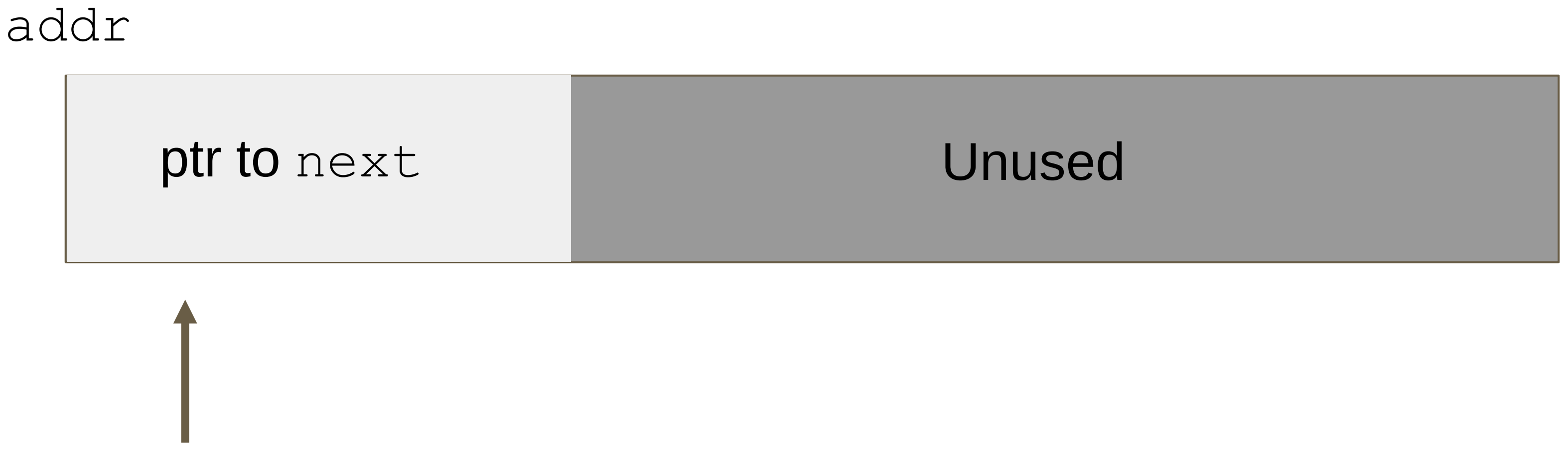
Allocate 1 object

```
x = free;
free = free->next;
return x;
```

free object x

```
x->next = free;
free = x;
```

What does a freed block look like?



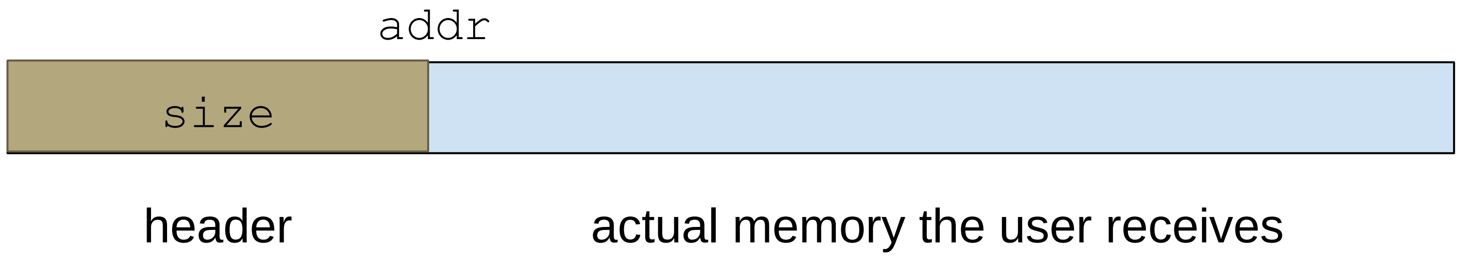
Freelist node struct

*Need to make sure Freelist node struct is smaller than the size of the block

Binned Free Lists

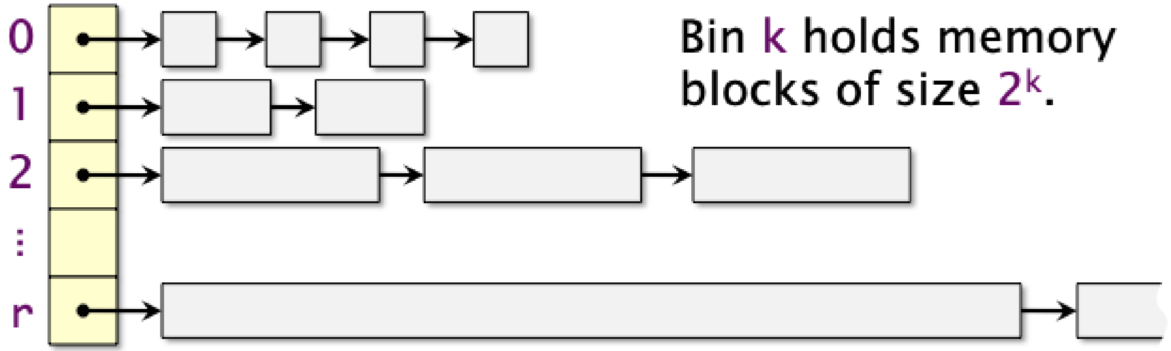
Binned Free Lists

- Allocate chunks of memory at specific sizes
  (i.e. round up user's requested size to the next power of 2)
- Maintain free lists for these different sizes
- Need to keep track of chunk sizes
  The user will only give us the pointer, not the size!
- Store this information in **headers**.



Binned Free Lists

- Leverage the efficiency of free lists.
- Accept a bounded amount of internal fragmentation.

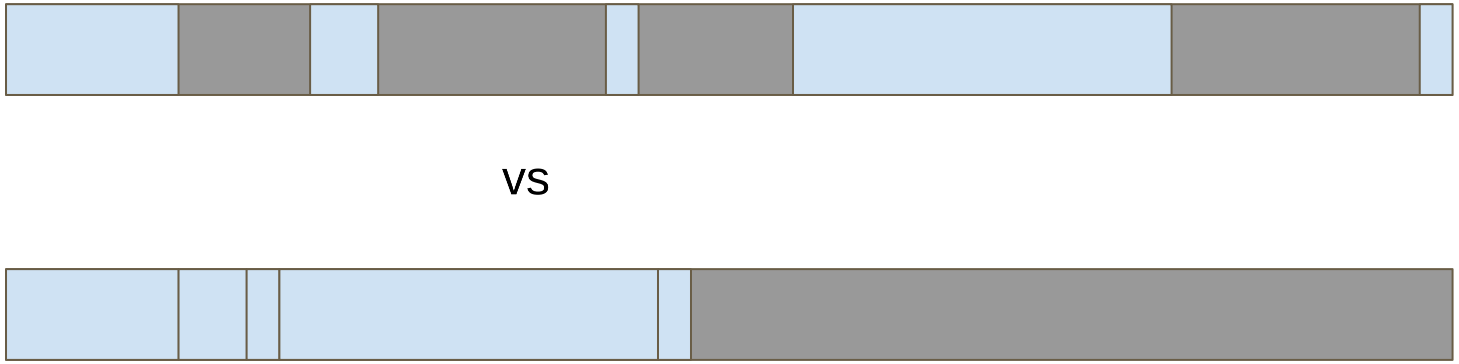


Bin k holds memory blocks of size 2^k.

Fragmentation

What is fragmentation?

- Memory is broken apart into many pieces
- Even if you have X amount of memory available, if it’s not contiguous, you can’t allocate it as a chunk of memory of size X.



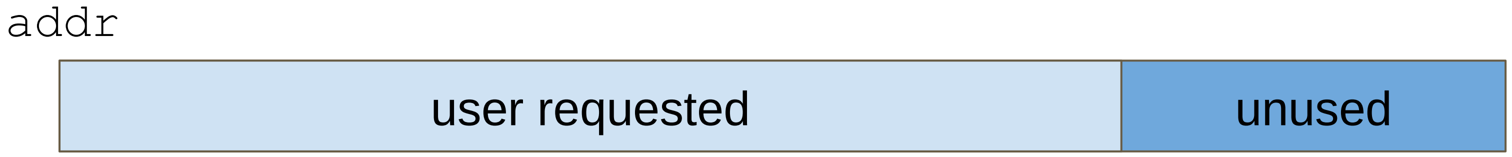
Types of Fragmentation

External fragmentation:

- Blocks are scattered across virtual memory, making remaining memory non-contiguous (previous slide)

Internal fragmentation:

- The difference in how much memory the user requested and how much we actually allocated (i.e. due to headers)



Strategies for Mitigating Fragmentation

- Splitting : dividing a large free block into smaller pieces, depending on how much memory the user requested
(allows you to “fill in” large gaps of free memory in your heap)

- Coalescing : merging together adjacent free blocks into a single, large free block