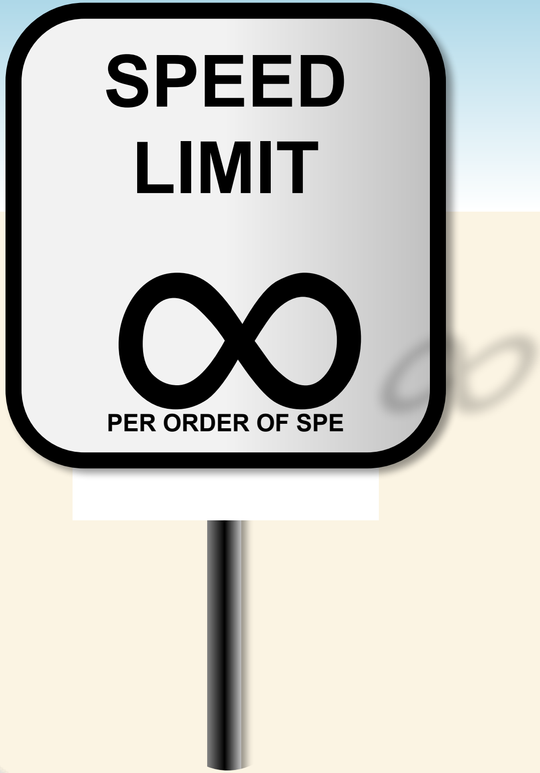
Software Performance Engineering



Recitation 2.3

Sophia Sun
Tuesday, November 4, 2025

Some due dates

- Homework 5 is due next Monday, November 10.
- Project 2 Beta is due next Thursday, November 13.
  - Remember to include your project log in the write up.
  - Remember to divide the commits evenly across all team members, so that everyone contributes to the project.

- Project 3A and 3B are due Monday, December 8.
  - If you haven't yet, please let me know which one you choose so I can prepare your repo, and you can start ASAP.

Malloc and Free

Malloc, Free, and Realloc

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

- Allocates a chunk of memory of size `size`

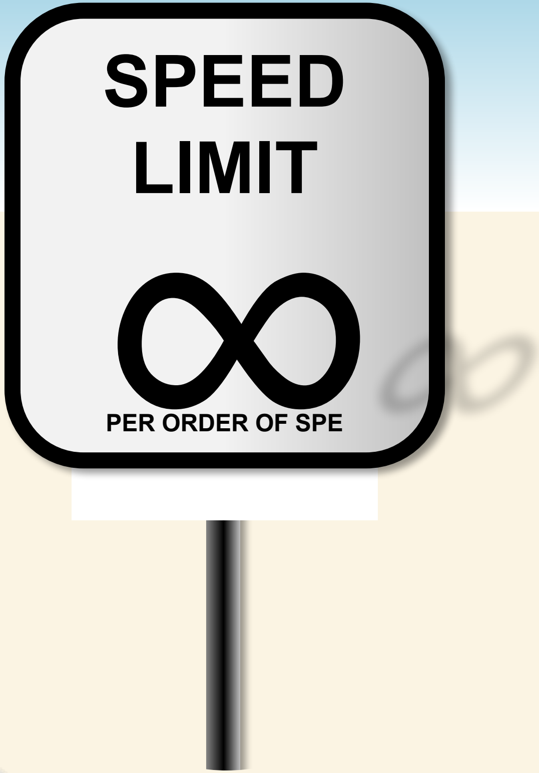
```
void free(void* addr)
```

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

Free Lists

Free Lists

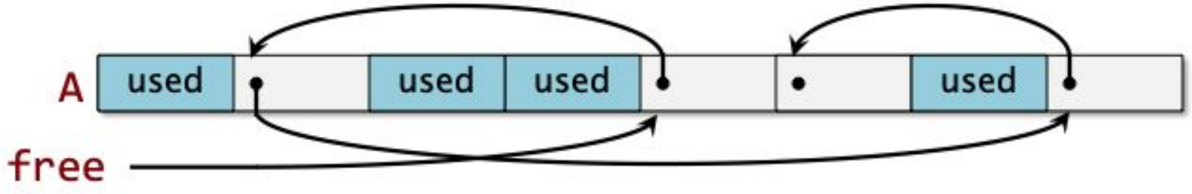
- Can implement as a singly linked list
- Keeps track of deallocated memory
- Allows us to reuse memory
- Most memory allocators use a freelist of some sort



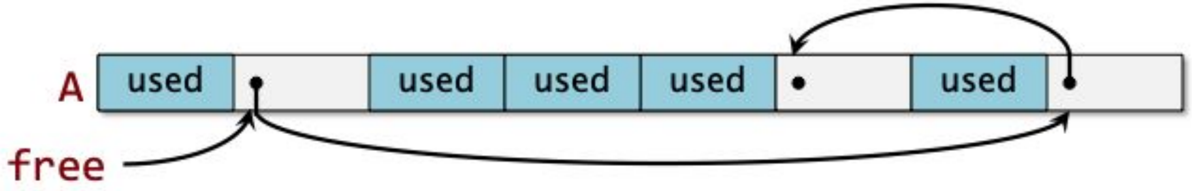
Allocating Memory w/ Free Lists (fixed size blocks)

Free-List: Allocating

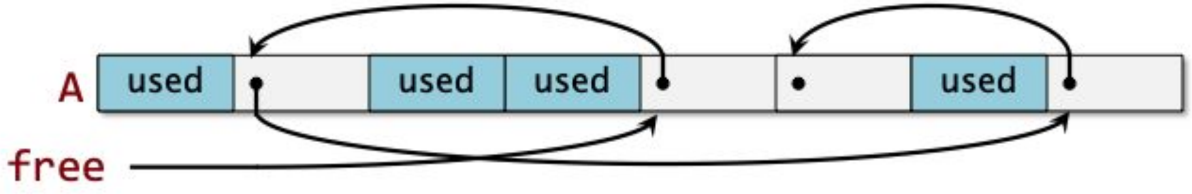
Before:



After:



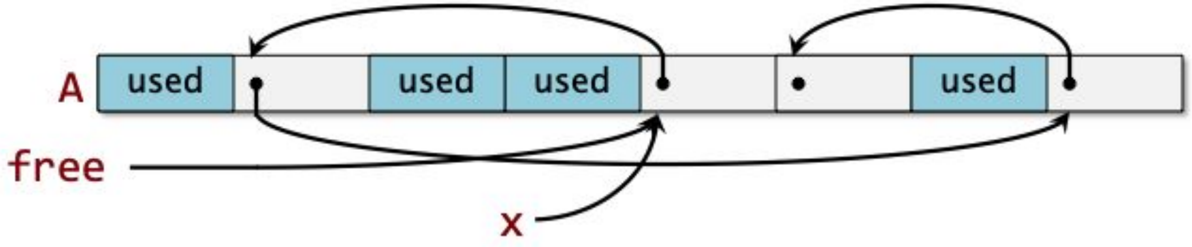
Free-List: Allocating



Allocate 1 object

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

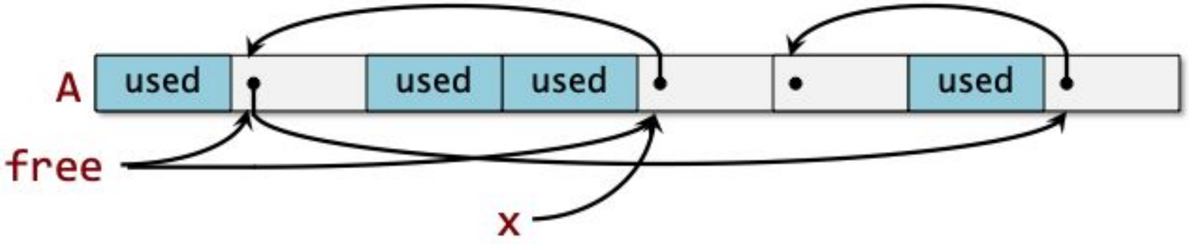
Free-List: Allocating



Allocate 1 object

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

Free-List: Allocating

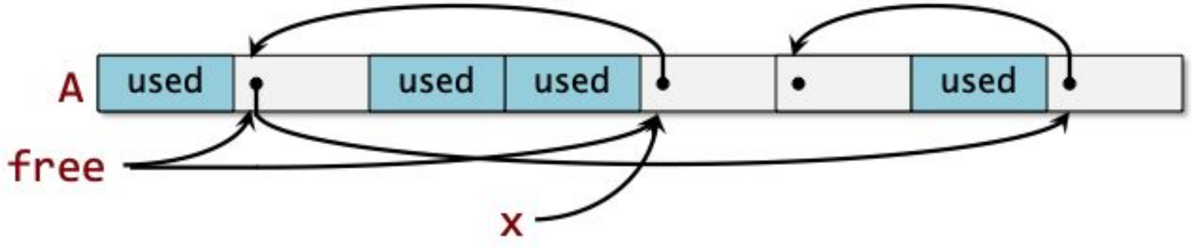


Allocate 1 object

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

Should check `free != NULL`.

Free-List: Allocating

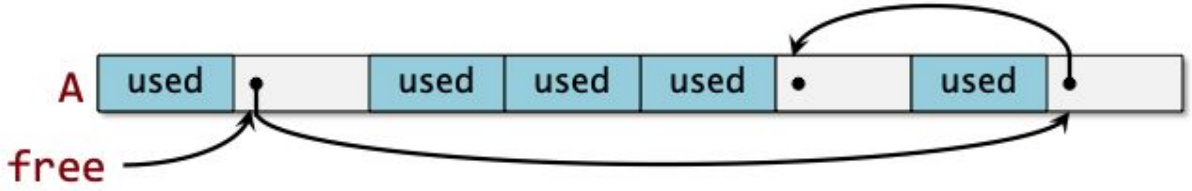


Allocate 1 object

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

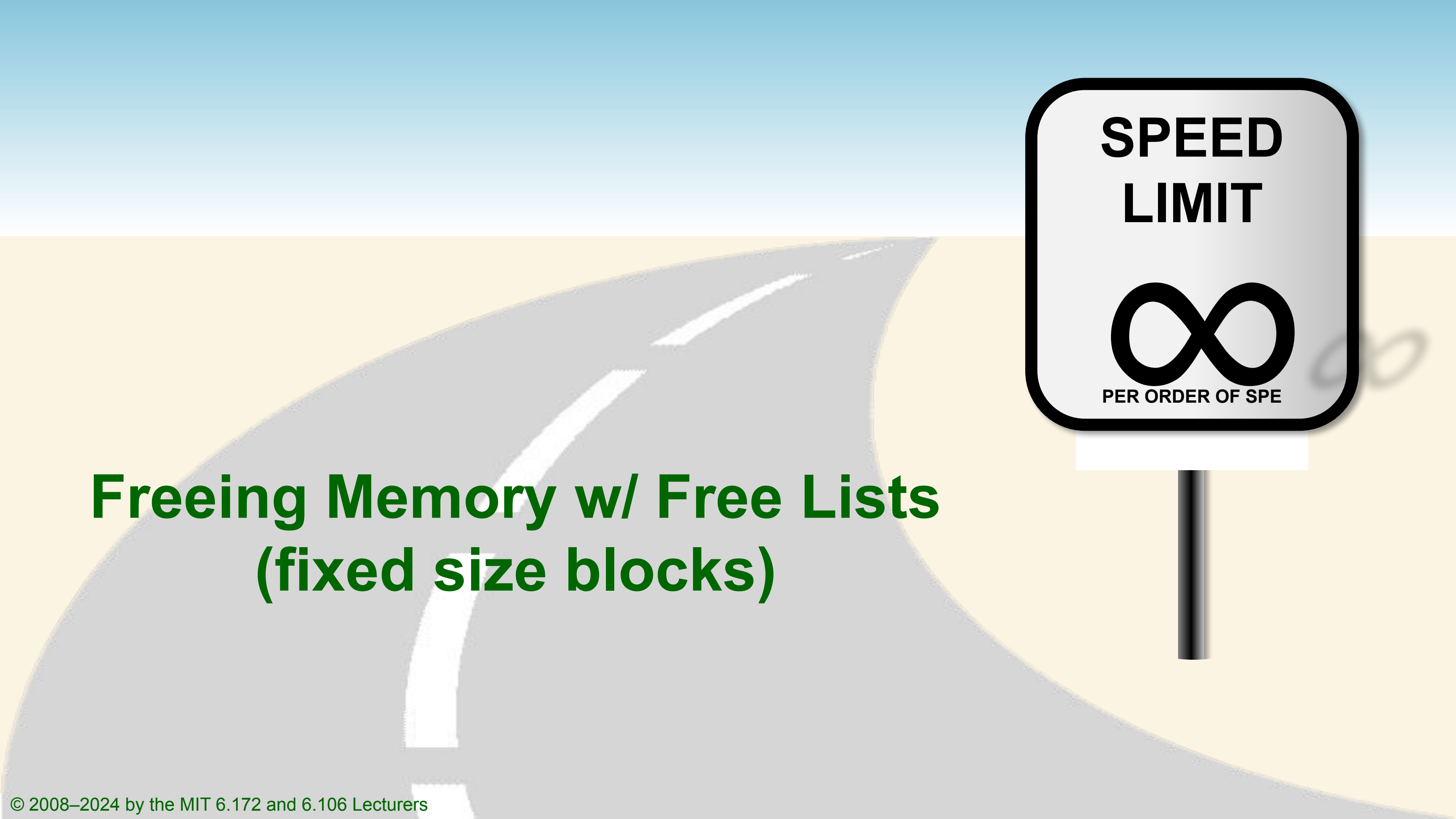
Should check `free != NULL`.

Free-List: Allocating



Allocate 1 object

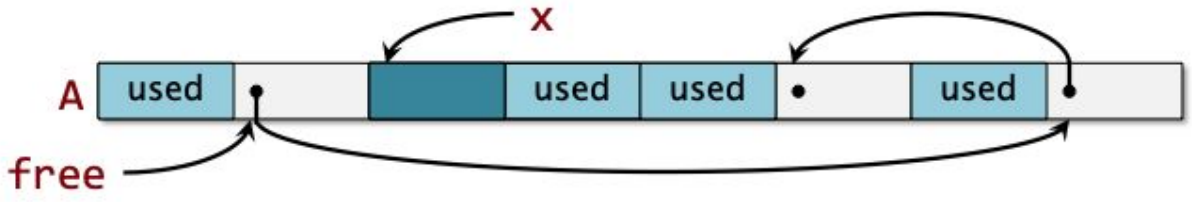
```
x = free;
free = free->next;
return x;
```



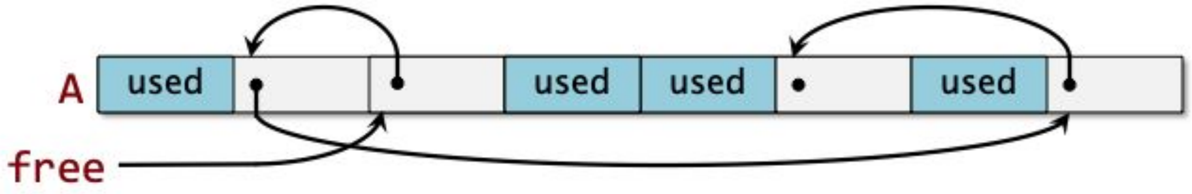
Freeing Memory w/ Free Lists (fixed size blocks)

Free-List: Deallocating

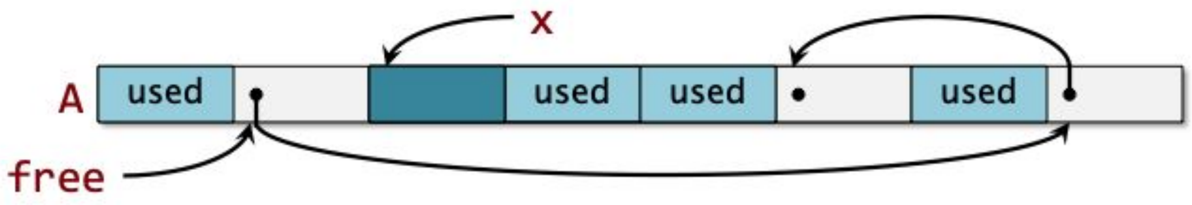
Before:



After:



Free-List: Deallocating



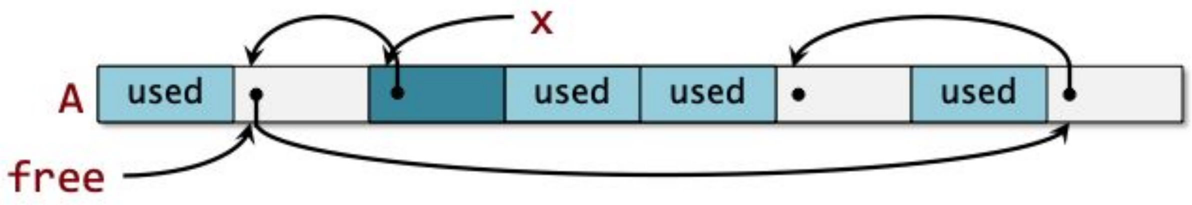
Allocate 1 object

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

free object x

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

Free-List: Deallocating



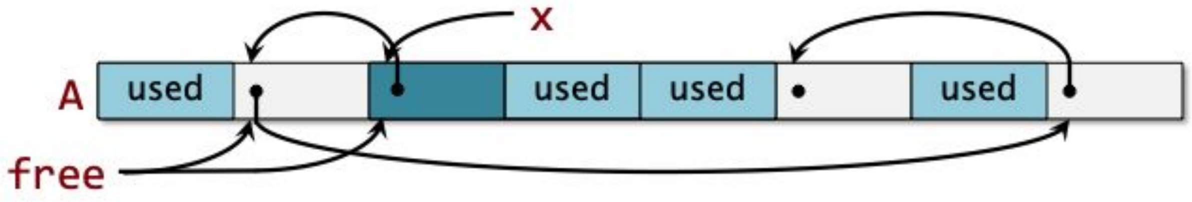
Allocate 1 object

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

free object x

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

Free-List: Deallocating



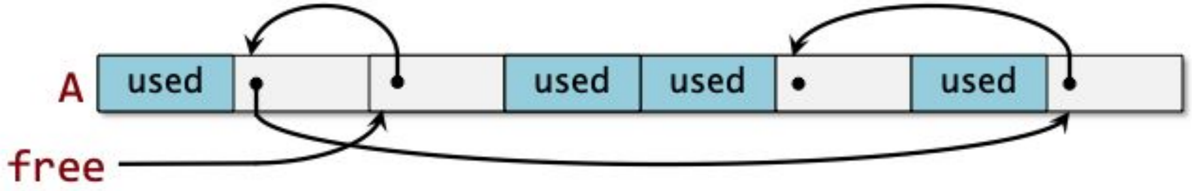
Allocate 1 object

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

free object x

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

Free-List: Deallocating



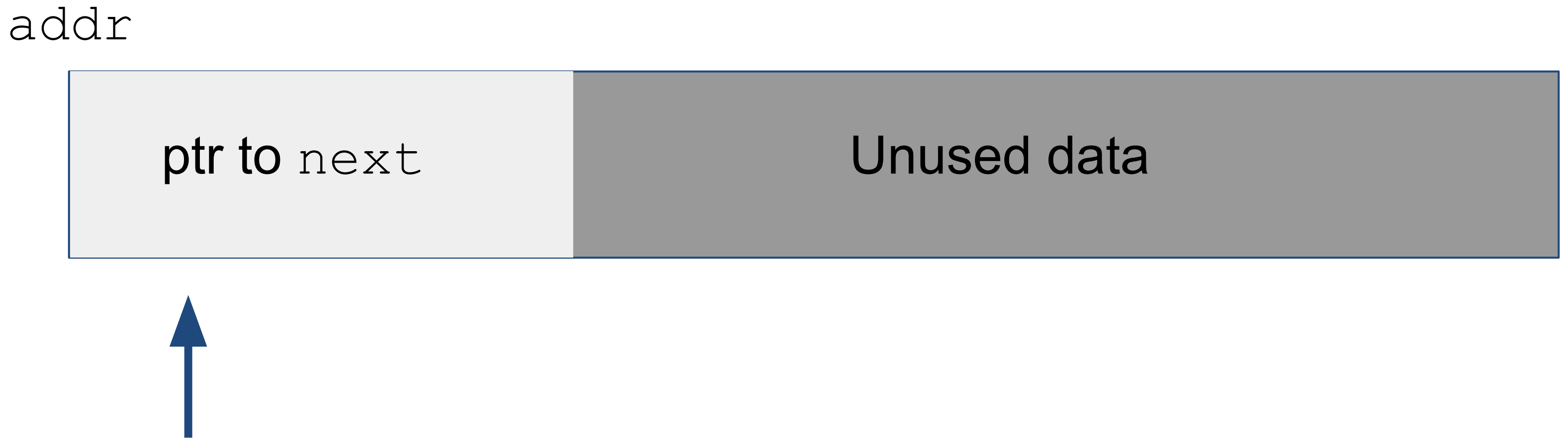
Allocate 1 object

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

free object x

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

What does a freed block look like?



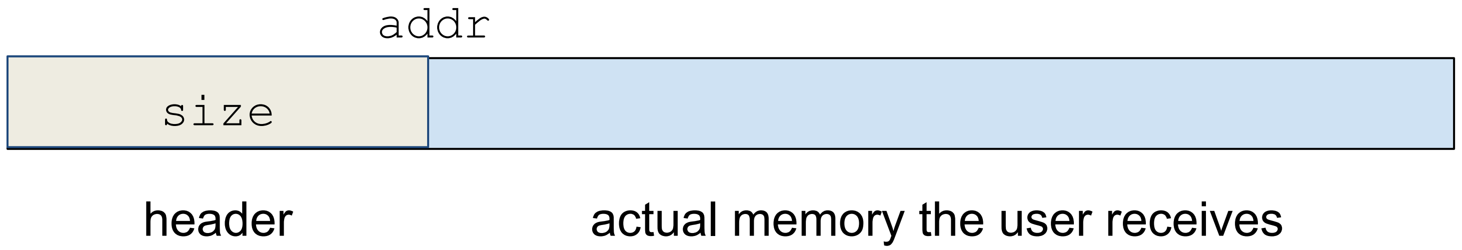
Freelist node struct

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

Binned Free Lists

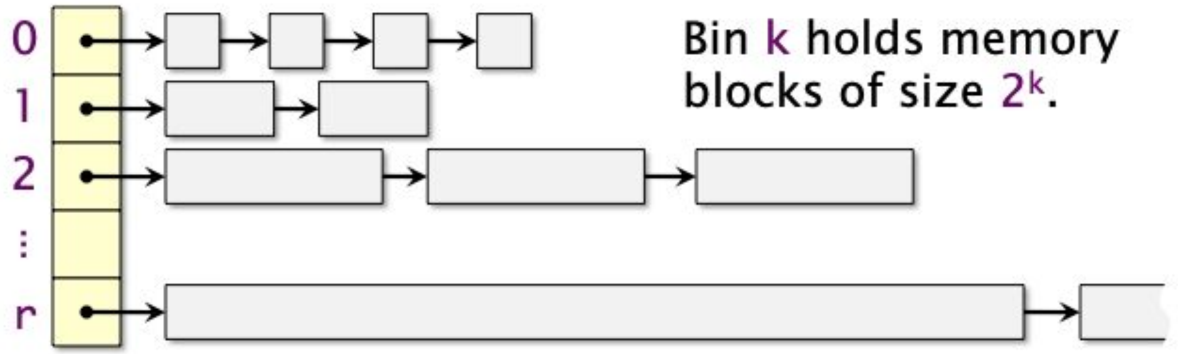
Binned Free Lists

- 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**.



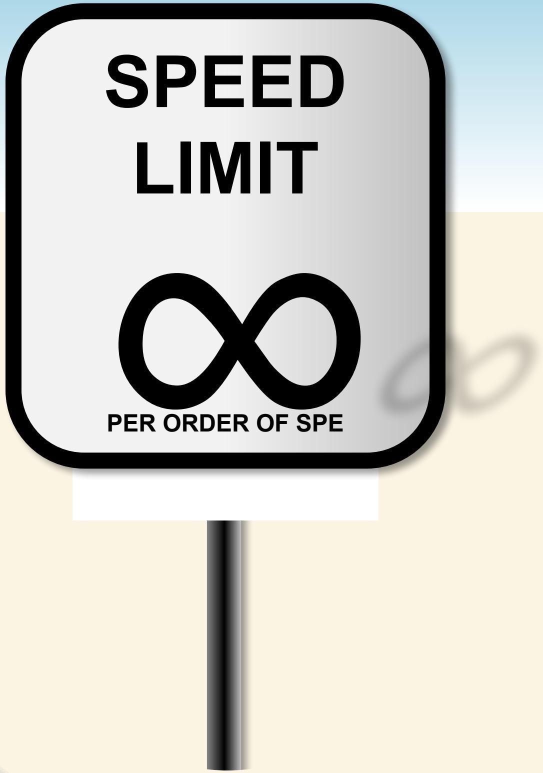
Binned Free Lists

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



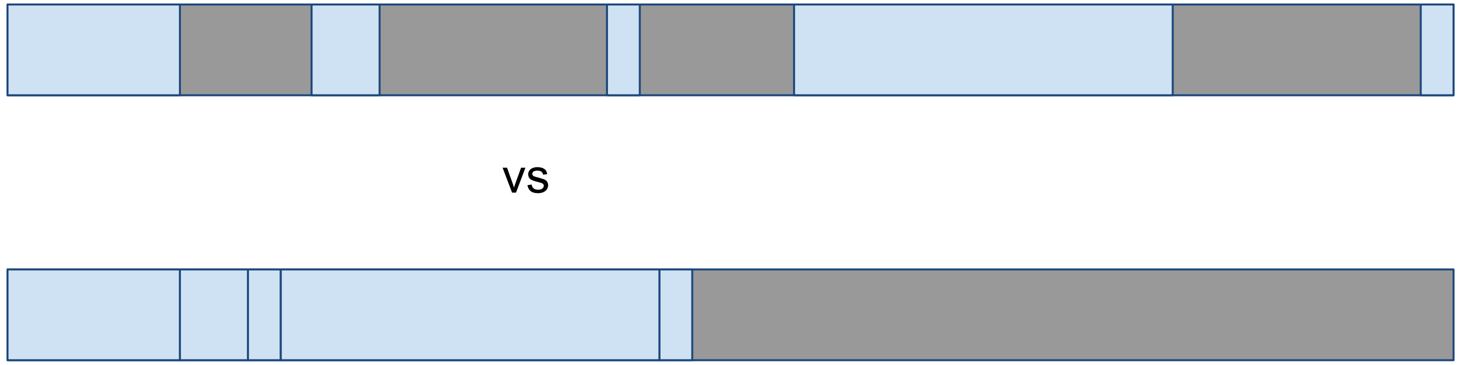
Bin k holds memory blocks of size 2^k.

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.



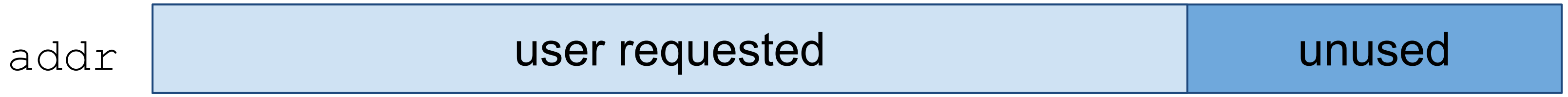
Types of Fragmentation

External fragmentation:

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

Internal fragmentation:

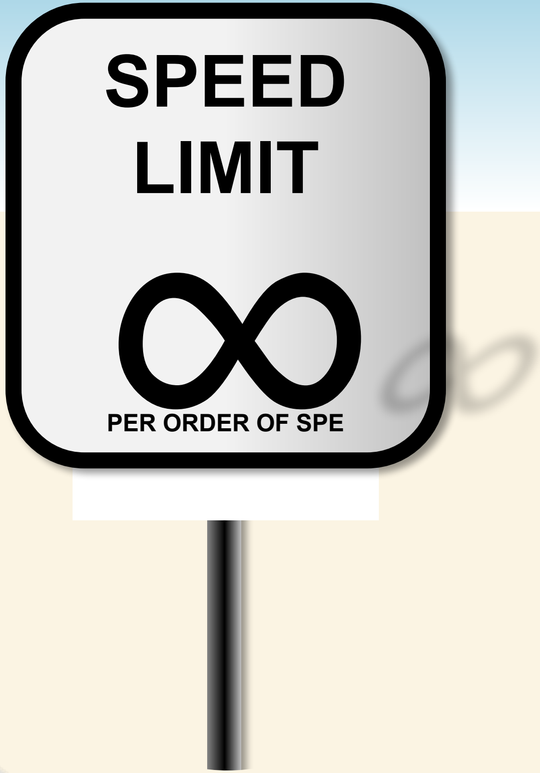
- The difference in how much memory the user requested and how much we actually allocated (i.e. due to headers)



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



Coding time!