Section 9: Memory Allocation

The Heap

- Memory allocated dynamically by the programmer (`malloc`)
- Must be explicitly freed (`free`)
  - Free it as soon as you don't need it!
- Distinct from normal variables, which are always on the stack

Use-cases:

- Variable-length data, like arrays or strings (think: Java's ArrayList)
- Long-lived data passed between functions

The interface of the heap in C

Signatures:

```
void* malloc(size_t length);

void free(void* ptr);
```

Usage:

```
int* array = (int*) malloc(10 * sizeof(int));

free(array);
```

Allocator internals: Finding a free block

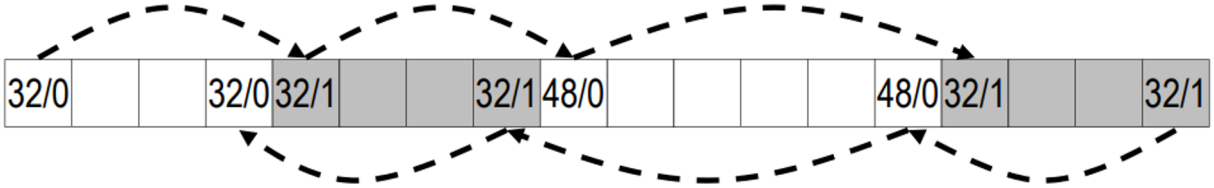
Two options:

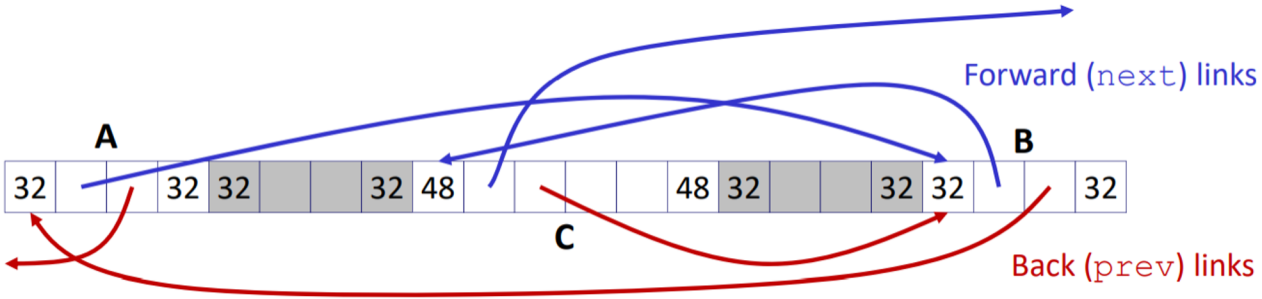
***Implicit* free list**

- Iterate through *all* the blocks until you find one that's free

***Explicit* free list**

- Each free block stores pointers to *other free blocks*





Lab 5 uses an explicit free list!

Reminder: Implicit/explicit free-lists are separate from implicit and explicit allocators.

Comparison: free-lists

Implicit

- Find the next block via incrementing by the current block's length
- It may or may not be free
  - Potentially lots of extra blocks in the way!
- Requires only knowledge of each block's size

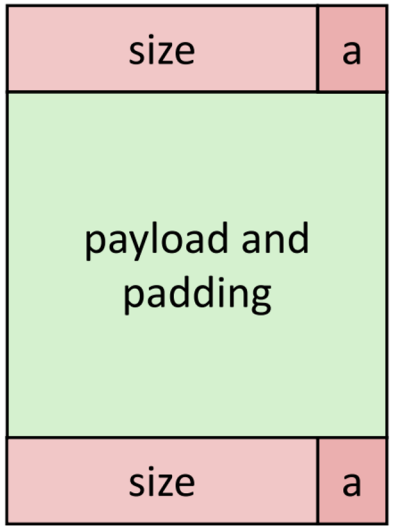
Explicit

- Find the next block by following a pointer
- All blocks in the free-list are guaranteed to be *free*
- Requires space in each free block to store pointers to the blocks before/after it

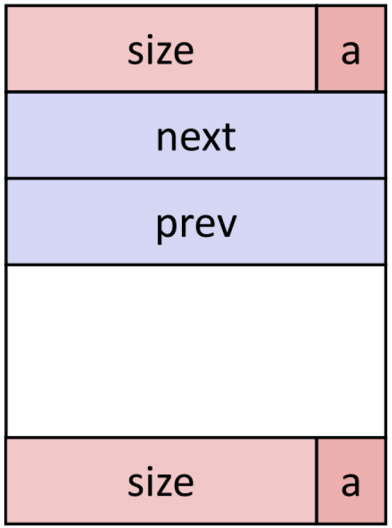
For the remainder of this section, we'll be looking at explicit free-lists.

Anatomy of a block (explicit free-list)

Allocated block:



Free block:



We will see a change to this later in section!

Boundary tags (header/footer) vs. free-list

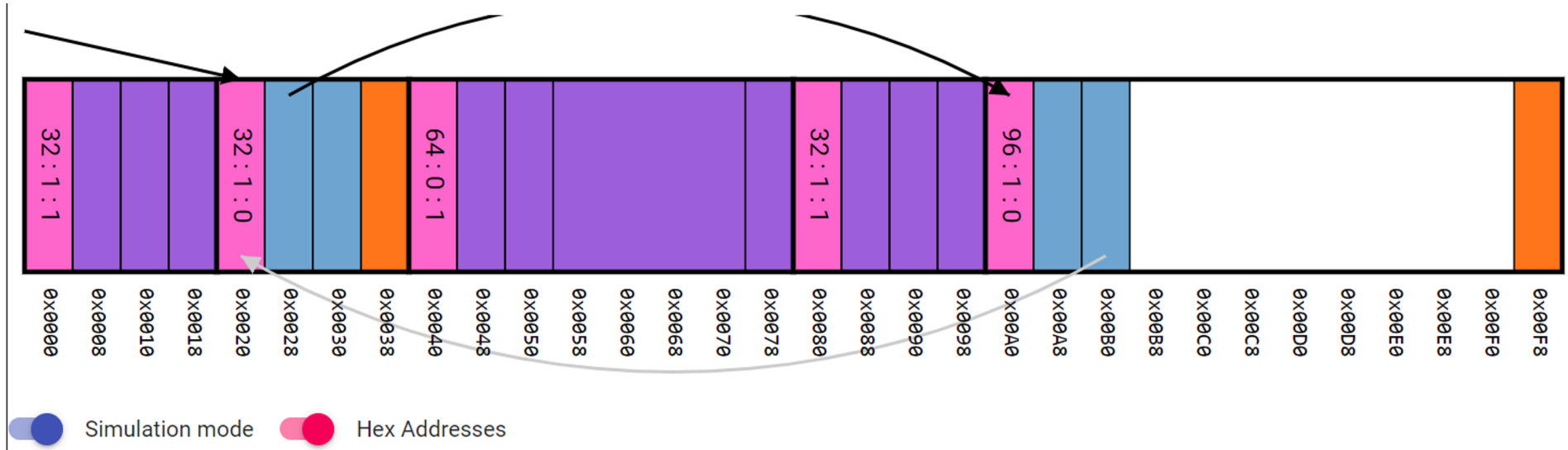
- Boundary tags (header and footer) are for the block immediately before/after in memory
  - Used for combining free blocks together
  - Facilitate checking whether those blocks are free

- Free-list (next and prev pointers) is used for finding an available free block
  - Entirely unrelated to physical memory layout
  - Used only when looking for a block to allocate

Key steps

- Allocation
  - Search for a block of sufficient size
  - If sufficient space for another block, split into two
  - Remove selected block from free-list
  - Mark the allocated block as allocated
  - Return a pointer to the *payload*

- Deallocation (freeing)
  - Mark as free
  - Coalesce with adjacent blocks if possible
  - Add new larger block to free-list
    - If using LIFO insertion policy, this free block becomes the new "root"




```
void *block4 = malloc(10);
```

GO

block0	0x0008	FREE
block2	0x0048	FREE
block3	0x0088	FREE