CSE 351 - Section 9:
VM Wrap-Up + Memory Allocation

Aaron Miller
David Cohen
Spring 2011

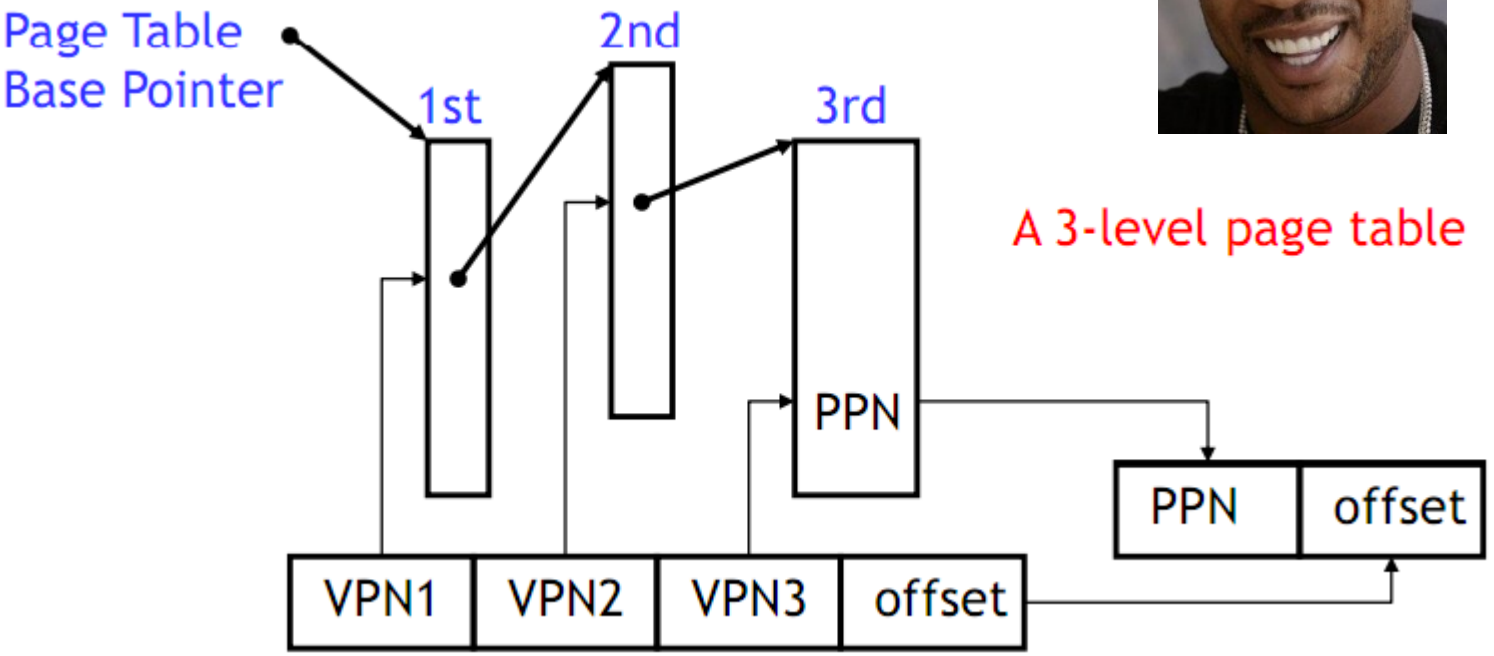
Some slides graciously adapted from Paul Pham

Section Outline

- Lingering HW 3 questions
- Hierarchical Page Tables
- Memory Allocation

Hierarchical Page Tables

- Idea: use indirection to reduce size of page table
- Goal: hold many first-level page tables in a page
- How: page your page tables!

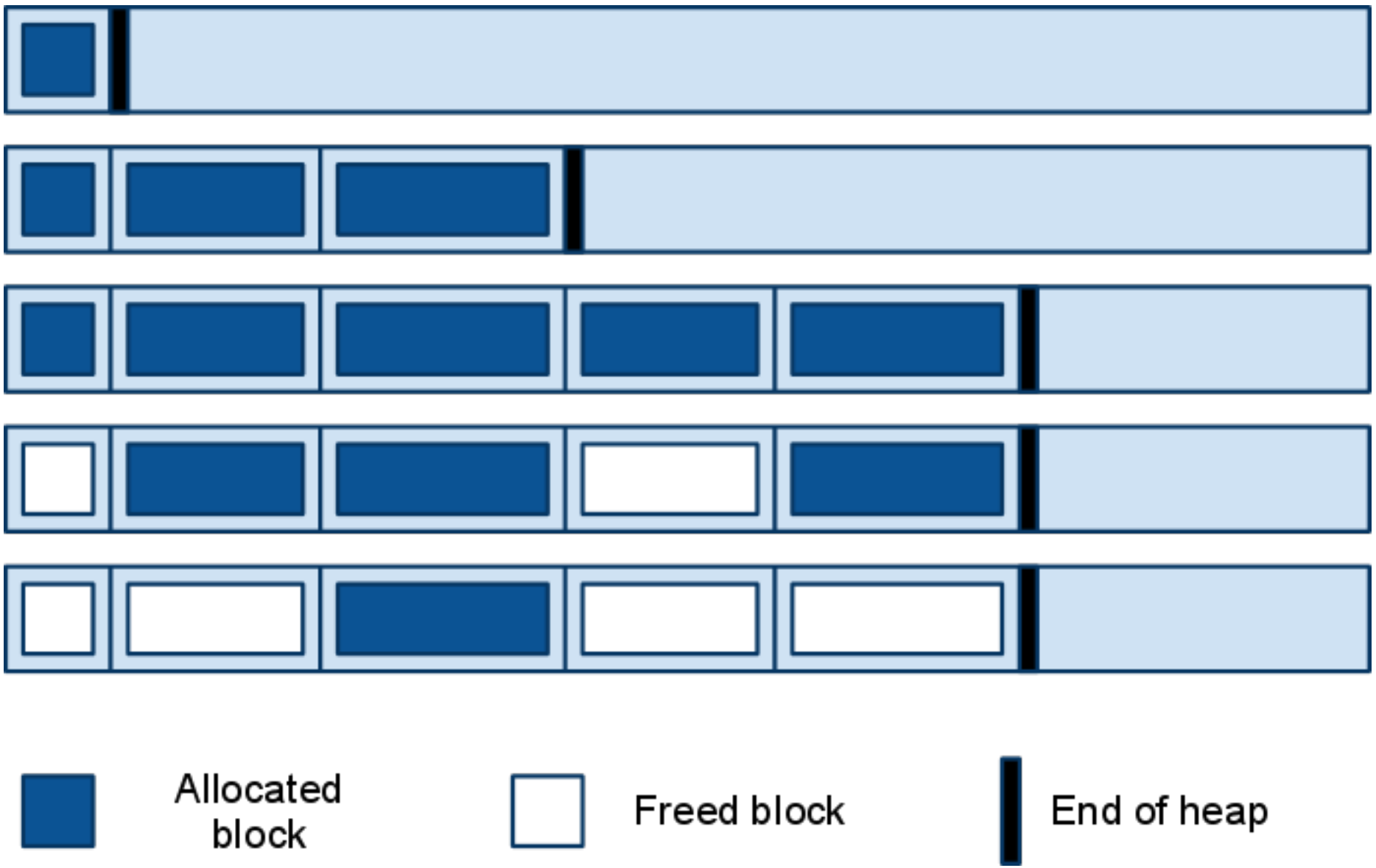


A 3-level page table

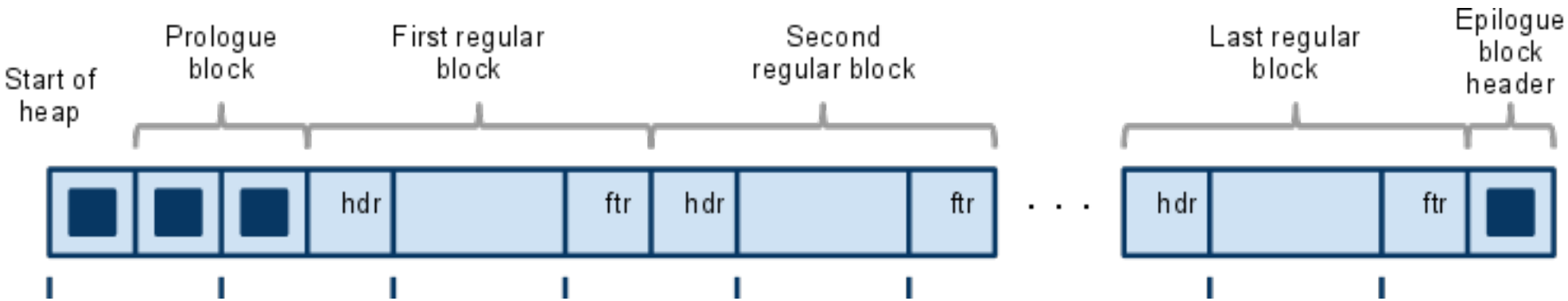
Hierarchical Page Table - Example

- Specs:
  - 48-bit virtual addys, 32-bit physical
  - 8 KiB page size, 4 B / entry
  - 3-level system
    - Level 3: 4 pages, Level 2: 2 pages
- Questions
  - How many bits for physical page offset (PPO)?
  - How many entries in each page table level?
  - How many bits for indexing each page table level?
  - How many first-level tables fit w/in a page?
  - What's the expression for translating VA → PA?

The Heap Never Shrinks



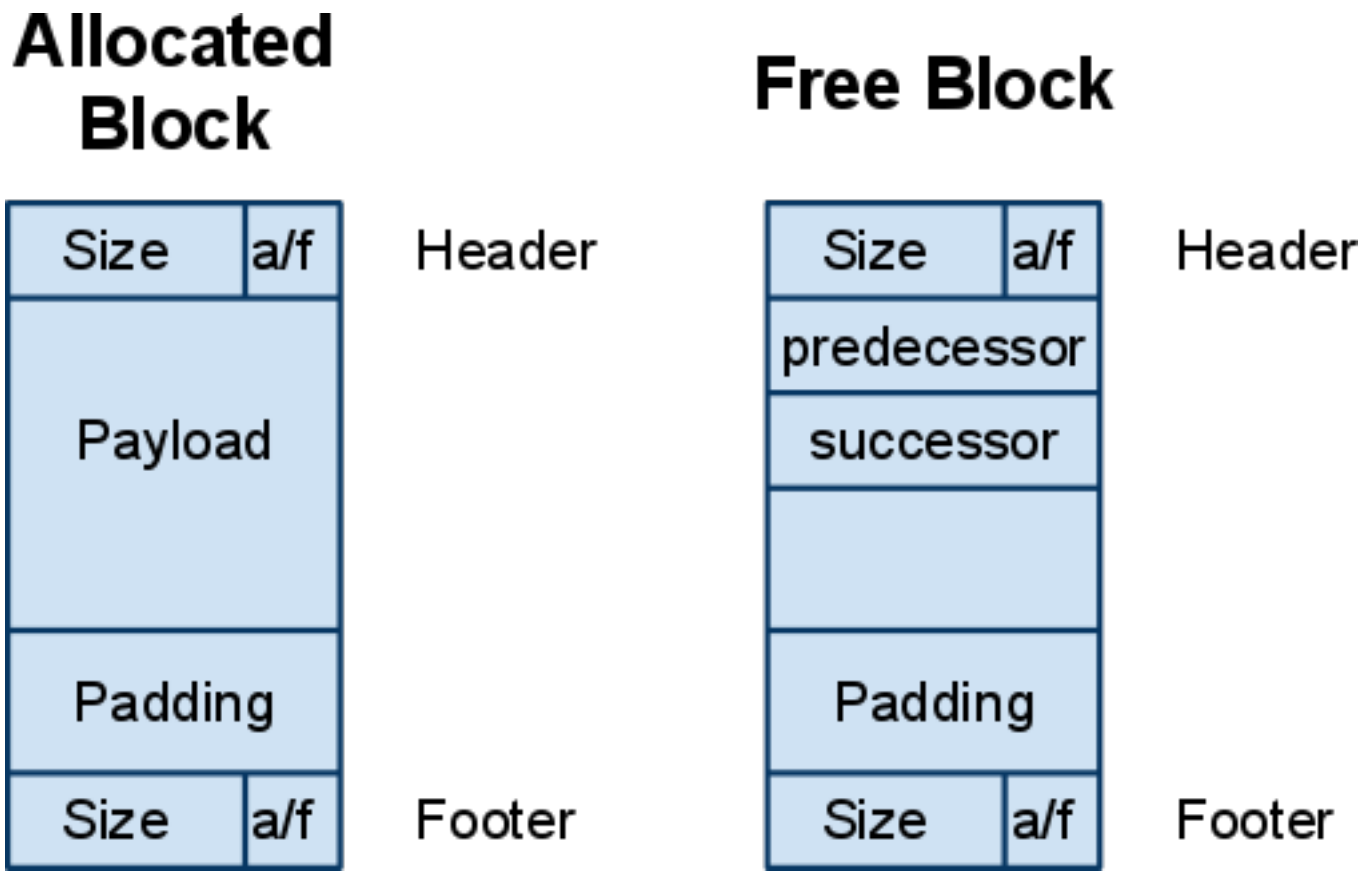
Questions for an Implicit Free List



- Why do payloads have to be 8-byte aligned?
- Why don't the block headers have to be 8-byte aligned?
- Why do we need prologue and epilogue blocks?

Blocks in an Explicit Free List

- We only need a payload for allocated blocks.
- We only need pred/succ pointers for free blocks.



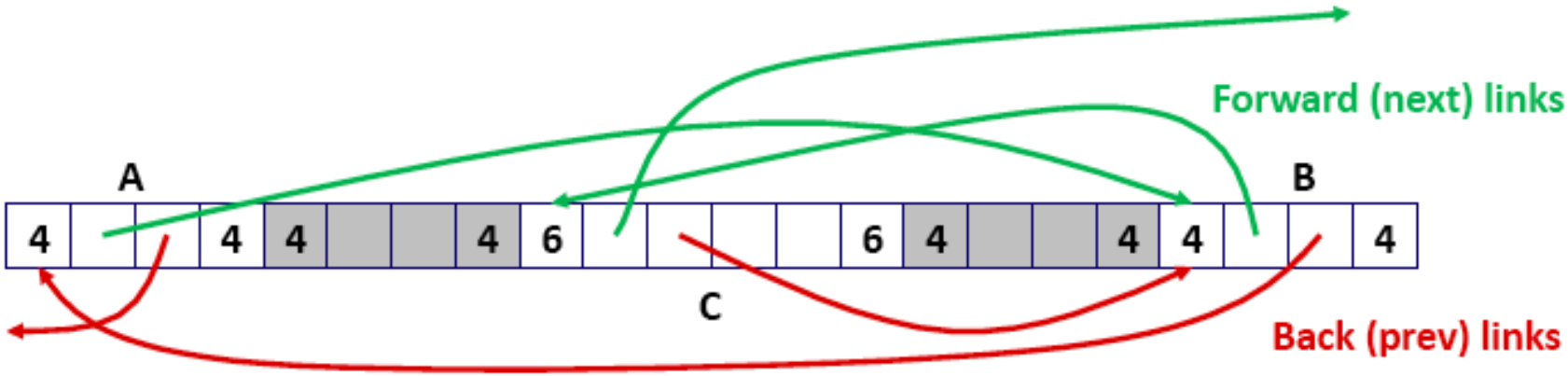
Explicit Free List: Logical vs. Physical

- **Logically (doubly-linked lists):**

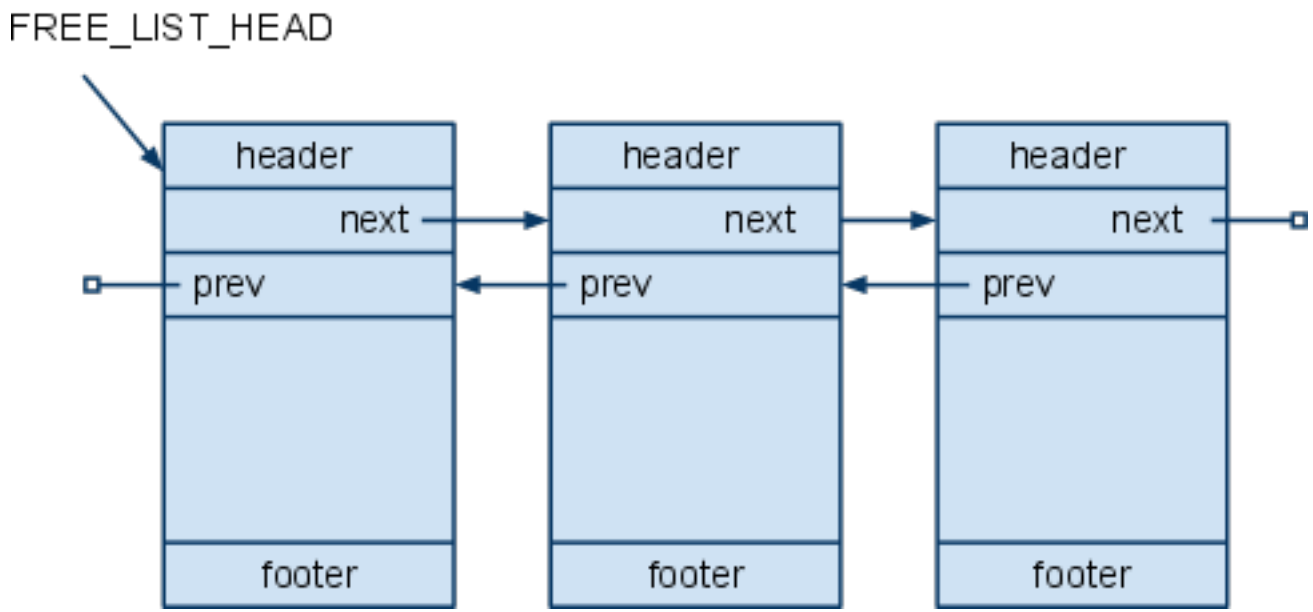
We can search for free blocks using this picture

- **Physically: blocks can be in any order**

We need to know actual neighboring blocks to coalesce



Structure of an Explicit Free List



- How do we insert blocks into or remove blocks from this list?
- How do we coalesce blocks in this list?

Lab 5: DIY Memory Allocator

Goal: write alternate dynamic memory allocator to the GNU libc implementation

Purpose:	GNU Function:	Your Function:
Increase heap size	sbrk()	mem_sbrk()
Allocate memory	malloc()	mm_alloc() **
Free memory	free()	mm_free() **

** You will implement these functions

BlockInfo Struct

```
struct BlockInfo {
  int sizeAndTags;
  struct BlockInfo* next;
  struct BlockInfo* prev;
};

typedef struct BlockInfo BlockInfo;
```

Low-Level Functions

// Called by requestMoreSpace to increase heap size
```
void *mem_sbrk(int incr);
```

// Resets the entire heap, the nuclear option
```
void mem_reset_brk(void);
```

// Beginning of heap, used in FREE_LIST_HEAD macro
```
void *mem_heap_lo(void);
```

// End of heap, you don't really have to worry about this
```
void *mem_heap_hi(void);
```

Constants and Macro Definitions

```
#define ALIGNMENT 8

#define WORD_SIZE sizeof(void*)

#define POINTER_ADD(p,x) ((char*)p + x)
#define POINTER_SUB(p,x) ((char*)p - x)
#define FREE_LIST_HEAD *((BlockInfo**)mem_heap_lo())

#define MIN_BLOCK_SIZE 4*WORD_SIZE
#define SIZE(x) (x & ~(ALIGNMENT - 1))

#define TAG_USED 1
#define TAG_PRECEDING_USED 2
```

Functions We Provide You

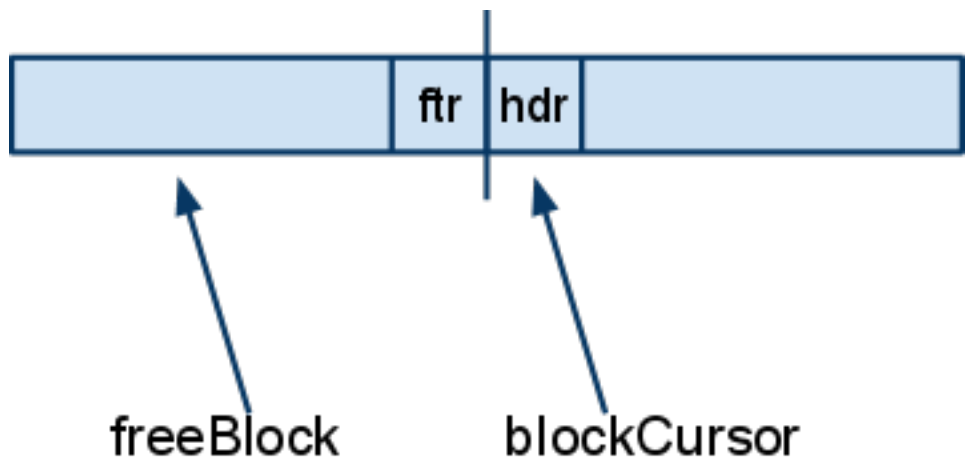
```
static void * searchFreeList(int reqSize);

static void insertFreeBlock(BlockInfo* freeBlock);
static void removeFreeBlock(BlockInfo* freeBlock);
static void coalesceFreeBlock(BlockInfo* oldBlock);

static void requestMoreSpace(int reqSize);
int mm_init();
```

Coalesce Function Overview

```
static void coalesceFreeBlock(BlockInfo* oldBlock) {

  /* 1. Coalesce with previous block, if possible */

  /* 2. Coalesce with next block, if possible */

  /* 3. Insert new free block, if coalesced */

}
```

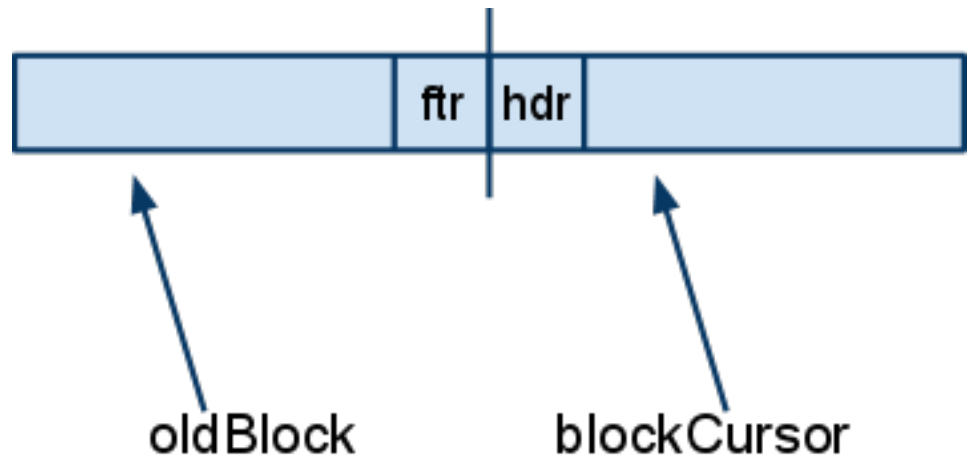
1. Coalesce With Previous Block

```
blockCursor = oldBlock;
while ((blockCursor->sizeAndTags & TAG_PRECEDING_USED)==0) {
  int size = SIZE(*((int*)POINTER_SUB(blockCursor, WORD_SIZE)));
  freeBlock = (BlockInfo*)POINTER_SUB(blockCursor, size);
  removeFreeBlock(freeBlock);

  newSize += size;
  blockCursor = freeBlock;
}
newBlock = blockCursor;
```



2. Coalesce With Next Block

```
blockCursor = (BlockInfo*)POINTER_ADD(oldBlock, oldSize);
while ((blockCursor->sizeAndTags & TAG_USED)==0) {
  // While the block is free:

  int size = SIZE(blockCursor->sizeAndTags);
  // Remove it from the free list.
  removeFreeBlock(blockCursor);
  // Count its size and step to the following block.
  newSize += size;
  blockCursor = (BlockInfo*)POINTER_ADD(blockCursor, size);
}
```



3. Insert New Coalesced Block

```
if (newSize != oldSize) {
  removeFreeBlock(oldBlock);

  newBlock->sizeAndTags = newSize | TAG_PRECEDING_USED;
  *(int*)POINTER_SUB(blockCursor, WORD_SIZE) =
      newSize | TAG_PRECEDING_USED;

  insertFreeBlock(newBlock);
}
return;
```

Some Final Parting Tips

- Understand the implicit free list allocator in the textbook first.
- Do the simplest possible thing that could work first.
  - Don't start out with complicated policies
- Optimize later.
- Debug / step through your code in GDB
- You don't need any other global variables.
  - You can traverse everywhere using existing pointers.
  - FREE_LIST_HEAD, block pointers, sizes
- Get started early.