CS 354 - Machine Organization & Programming
Tuesday, Feb 28 and Thurs March 2nd, 2023

Project p3: Released DUE on or before Friday March 24

Activities Week 6 and p3 Practice available

Homework 3: DUE on or before Monday March 6

Exam 1: Scores posted by Thursday (I hope)

Last Week

Posix `brk` & `unistd.h` C's Heap Allocator & `stdlib.h` Meet the Heap Allocator Design Simple View of Heap	Free Block Organization Implicit Free List Placement Policies **MIDTERM EXAM 1**

This Week

Placement Policies Free Block - Too Large/Too Small Coalescing Free Blocks Free Block Footers	Explicit Free List Explicit Free List Improvements Heap Caveats Memory Hierarchy
Next Week: Locality and Designing Caches B&O 6.4.2	

p3 Progress Dates
- complete Week 6 activity as soon as possible
- review source code functions before lecture Tuesday
- implement alloc by Friday this week and submit progress
- implement free by Tuesday next week and submit progress
- implement coalesce by Thursday next week and submit progress
- complete testing and debugging by Friday next week and complete final submission

Free Block - Too Large/Too Small

What happens if the free block chosen is bigger than the request?

- Entire block is used

  - mem util: more internal fragmentation

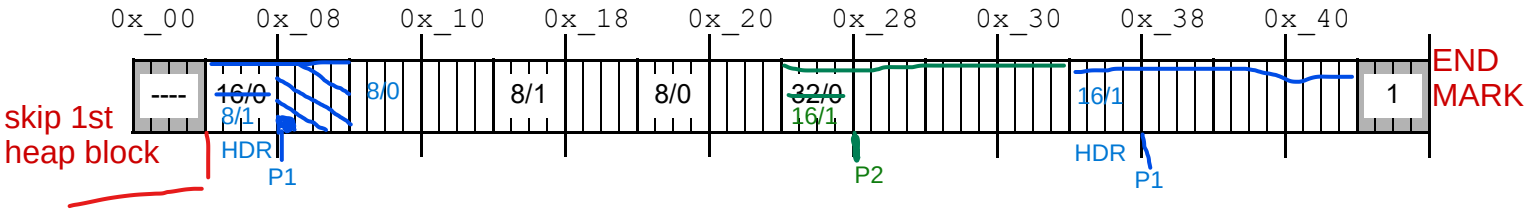
  + thruput: fast and simple code

- Split the block

  + mem util: less internal fragmentation due to less padding or "wasted space"

  - thruput: more heap blocks, slower to search

Run 4: Heap First-Fit Allocation with Splitting



→ Diagram how the heap above is modified by the 4 mallocs below.
For each, what address is assigned to the pointer?
If there is a new free block, what is its address and size?

	pay+HDR+pad	PTR	New Free Block
1) `p1 = malloc(sizeof(char));`	1 + 4 + 3 = 8	0x_08	0x_0C to 0x_13
2) `p2 = malloc(11 * sizeof(char));`	11 + 4 + 1 = 16	0x_28	0x_34 to 0x_43
3) `p3 = malloc(2 * sizeof(int));`	8 + 4 + 4 = 16	0x_38	No free block
4) `p4 = malloc(5 * sizeof(int));`	20 + 4 = 24	ALLOC FAILS	0x0 (NULL)

What happens if there isn't a large enough free block to satisfy the request?

also in p3 1st. coalesce or "merge" adjacent free blocks

→ Can allocated blocks be moved out of the way to create larger free areas? NO!

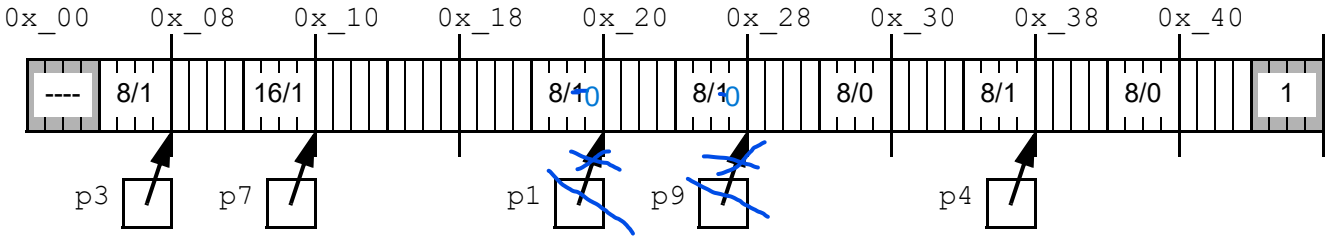
2nd. ask kernel for more space for our heap (not in p3)

(p3) 3rd. Alloc FAILS!, return NULL (0x0)

"merging"

Coalescing Free Blocks

Run 5: Heap Freeing without Coalescing



→ What's the problem resulting from the following heap operations?

```
1) free(p9); p9 = NULL;
2) free(p1); p1 = NULL;
3) p1 = malloc(4 * sizeof(int));
```
16 + 4 + 4 = 24 ALLOC FAILS

Problem? FALSE FRAGMENTATION

enough contiguous free space, but in blocks that are too small

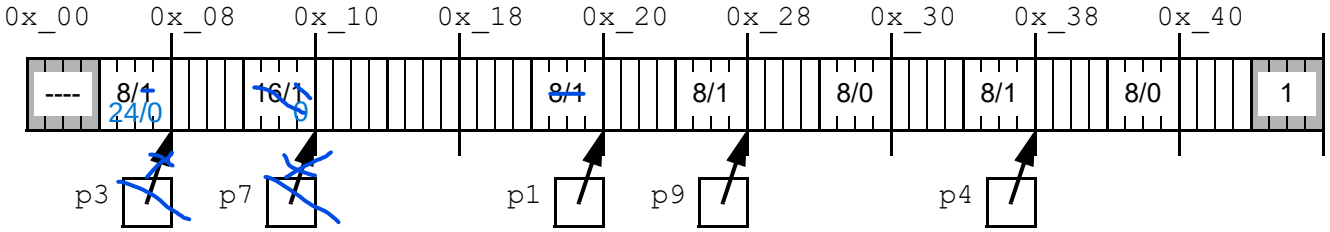
Solution? coalesce adjacent free blocks

immediate: coalesce next and previous when block is freed (if prev and next are free)

delayed: coalesce on alloc when needed and possible

p3 when asked:

Run 6: Heap Freeing with Immediate Coalescing



→ Given the heap above, what is the size in bytes of the freed heap block? 16, NO COALESCE

```
1) free(p7); p7 = NULL;
```

→ Given a pointer to a payload, how do you find its block header?

p4 = 0x_38 p4 HDR p4 - 4 p4 - (sizeof(blockHDR))

→ Given a pointer to a payload, how do you find the block header of the NEXT block?

ptr - sizeof(blockHDR) + "size of curr block"

(char *)ptr
(void *) ptr S.F. = 1
block HDR => S.F. = 4

❋ *Use type casting* to set correct scale factor

→ Given the modified heap above, what is the size in bytes of the freed heap block when immediate coalescing is used?

```
2) free(p3); p3 = NULL;
```
0 + 8 + 16 = 24 bytes

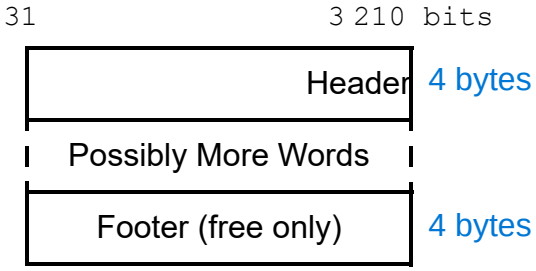
```
3) free(p1); p1 = NULL;
```
24 + 8 + 0 = 32 bytes

→ Given a pointer to a payload, how do you find the block header of the PREVIOUS block?

Free Block Footers

- *The last word of each free block* is a footer containing free block size
  - → Why don't allocated blocks need footers? they can't be coalesced
  - → If only free blocks have footers, how do we know if previous block will have a footer? contain a p-bit (previous block alloc'd bit)
- *Free and allocated block headers*

Layout 2: Heap Block with Headers & Free Block Footers



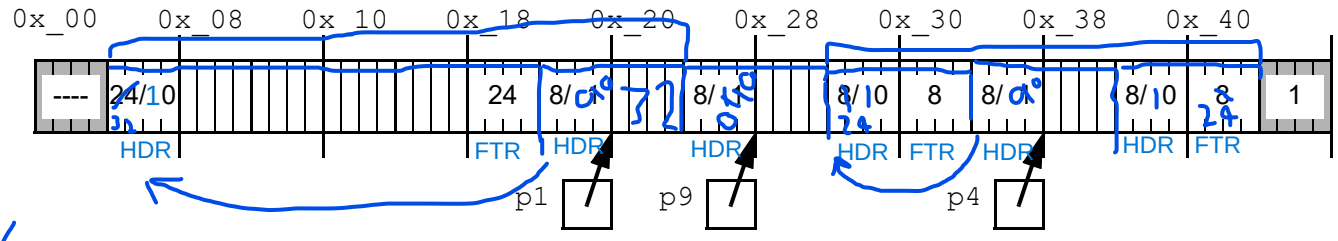
→ What integer value will the header have for an allocated block that is: p-bit a bit

1) 8 bytes in size and prev. block is free? 8 + 0 + 1 = 9
2) 8 bytes in size and prev. block is allocated? 8 + 2 + 1 = 11
   0 0 0 0 0 … 0 0 0 1 0 1 1
3) 32 bytes in size and prev. block is allocated? 32 + 2 + 1 = 35
   … 0 0 0 0 1 0 0 0 1 1
4) 64 bytes in size and prev. block is free? 64 + 0 + 1 = 65
   … 0 0 0 1 0 0 0 0 0 1

→ Given a pointer to a payload, how do you get to the header of a previous block if it's free?

1. get header of current — ptr - 4
2. check p-bit of current — if (p-bit is 0) prev is free
3. get to previous block footer — ptr - 8 or hdr - 4
4. get to previous block header — (ptr - 4) - prev block size

Run 7: Heap Freeing with Immediate Coalescing using p-bits and Footers



→ Given the heap above, what is the size in bytes of the freed heap block?

```
1) free(p1); p1 = NULL;
```
prev + curr + next = 24 + 8 + 0 = 32

→ Given the modified heap above, what is the size in bytes of the freed heap block?

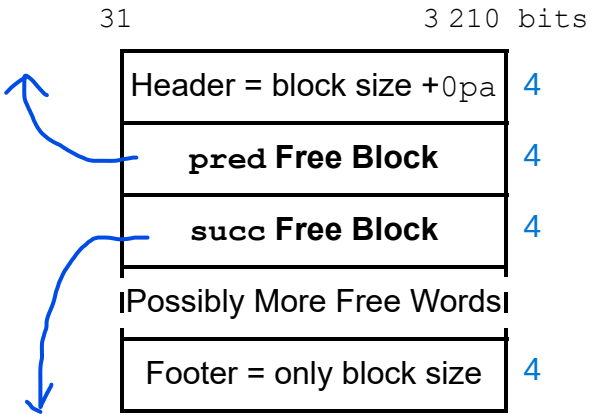
```
2) free(p4); p4 = NULL;
```
8 + 8 + 8 = 24

- *Don't forget to update* header of free block, footer of free block, p-bit of next block (if next block is not the end mark)
  - ➢ Is coalescing done in a fixed number of steps (constant time) or is it dependent on the number of heap blocks (linear time)?
    O(1) constant

Explicit Free List

❋ *An allocator using an explicit free list* only keeps a list of free blocks
It can be integrated and stored in heap

Explicit Free List Layout: Heap Free Block with Footer



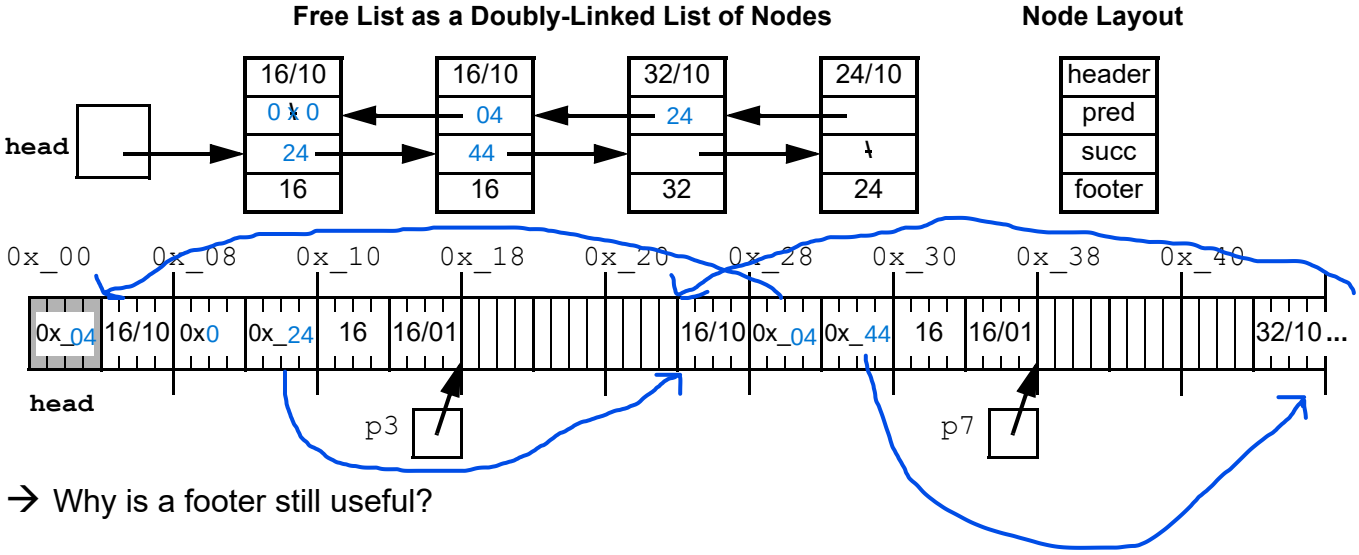
Free Block Links

pred address of previous free block

succ address of next free block

min size 16 blocks

→ Complete the addresses in the partially shown heap diagram below.



→ Why is a footer still useful?

Faster coalescing with prev adj block

→ Does the order of free blocks in the free list need to be the same order as they are found in the address space? NO

Explicit Free List Improvements

Free List Ordering

address order : maintain list in order from low to high address

+ malloc with FF has better memory utilization than "lastin" order

- free O(N) where N is number of free blocks, must insert

last-in order: place most recently freed block at end of doubly-linked list

- malloc with FF slower, must go through most recently

+ free 0(1), just link at end of E.F.L. (explicit free list?)

Free List Segregation keep an array of free list - a separate E.F.L for each size free block
malloc chooses correct list to search for free block

simple segregation: one for each block size

structure simple, no need for header, free block only need next address "successor"

+ malloc pick desired E.F.L

if free list is empty ask for more mem, coalesce from smaller or split larger

+ free 0(1), add to correct list

- problem internal fragmentation because of splitting
external fragmentation because of no coalescing

C's solution *fitted segregation*: One E.F.L for each size range (sm, med, lg)

+mem util - as good as best fit
+thruput - search only part of heap

fitting use first fit (FF) of appropriate E.F.L, if fail search next large range

splitting put the newly created free block into appropriate E.F.L

coalescing put the coalesced free block into appropriate E.F.L

Heap Caveats

Consecutive heap allocations don't result in contiguous payloads!

→ Why?

- payloads are interspersed with heap structure and possibly padding
- placement policies and heap structure can scatter allocations throughout heap

Don't assume heap memory is initialized to 0!

OS initially clears heap pages for security,
but your recycled heap memory will have your old data unless you use calloc()

Do free all heap memory that your program allocates!

→ Why are memory leaks bad?

They slowly kill your program's performance by cluttering heap with garbage blocks
bad leaks could ultimately consume your entire heap

→ Do memory leaks persist when a program ends?

No, heap pages are returned to the OS for other uses

Don't free heap memory more than once!

→ What is the best way to avoid this mistake?

set freed pointers to NULL

Don't read/write data in freed heap blocks!

→ What kind of error will result?

Intermittent error

Don't change heap memory outside of your payload!

→ Why?

You'll trash the heap's internal structure and/or another block's payload

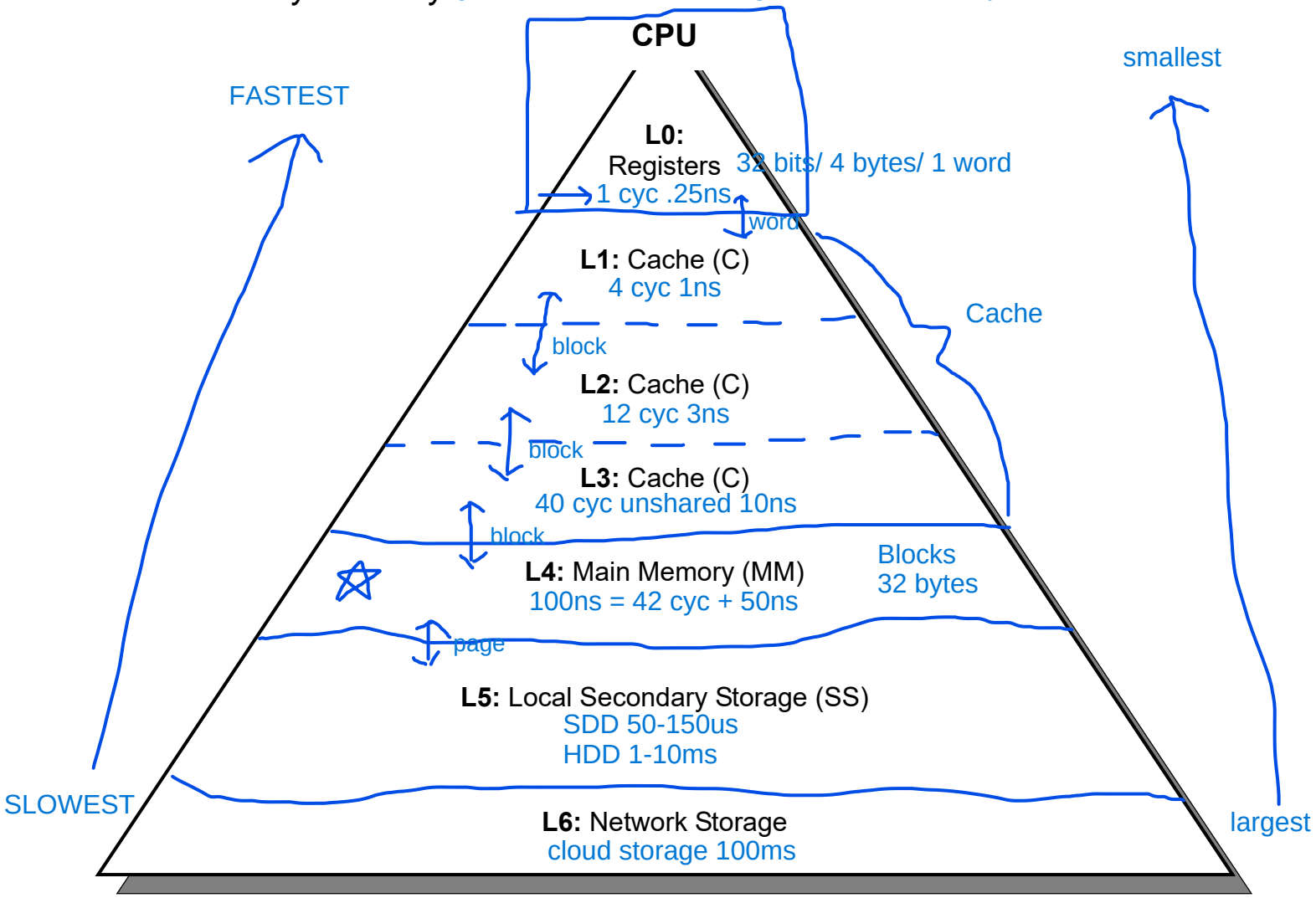
Do check if your memory intensive program has run out of heap memory!

→ How?

Check that allocator's return value is not NULL

Memory Hierarchy

❋ *The memory hierarchy* gives the illusion of having lots of fast memory



Cache

is a smaller faster memory that acts as a staging area for data stored in a larger slower memory

Memory Units

4 bytes	*word*: size used by CPU	transfer between L1 & CPU
32 bytes	*block*: size used by C	transfer between C levels & MM (Main Memory)
size of MM	*page*: size used by MM	transfer between MM & SS (Secondary Storage)

Memory Transfer Time

cpu cycles: used to measure time

latency: memory access time (delay)