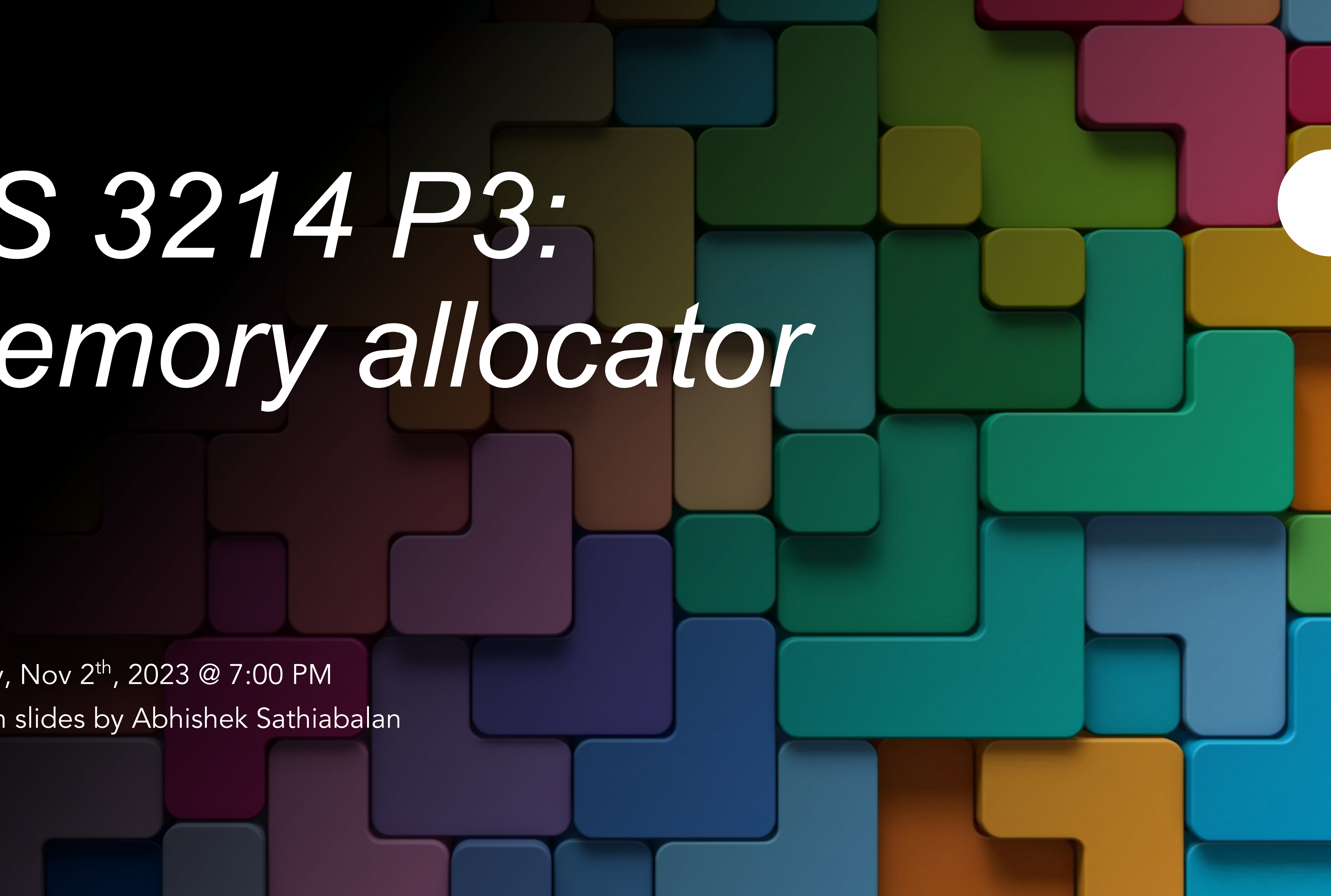
CS 3214 P3: Memory allocator



Thursday, Nov 2th, 2023 @ 7:00 PM

Based on slides by Abhishek Sathiabalan

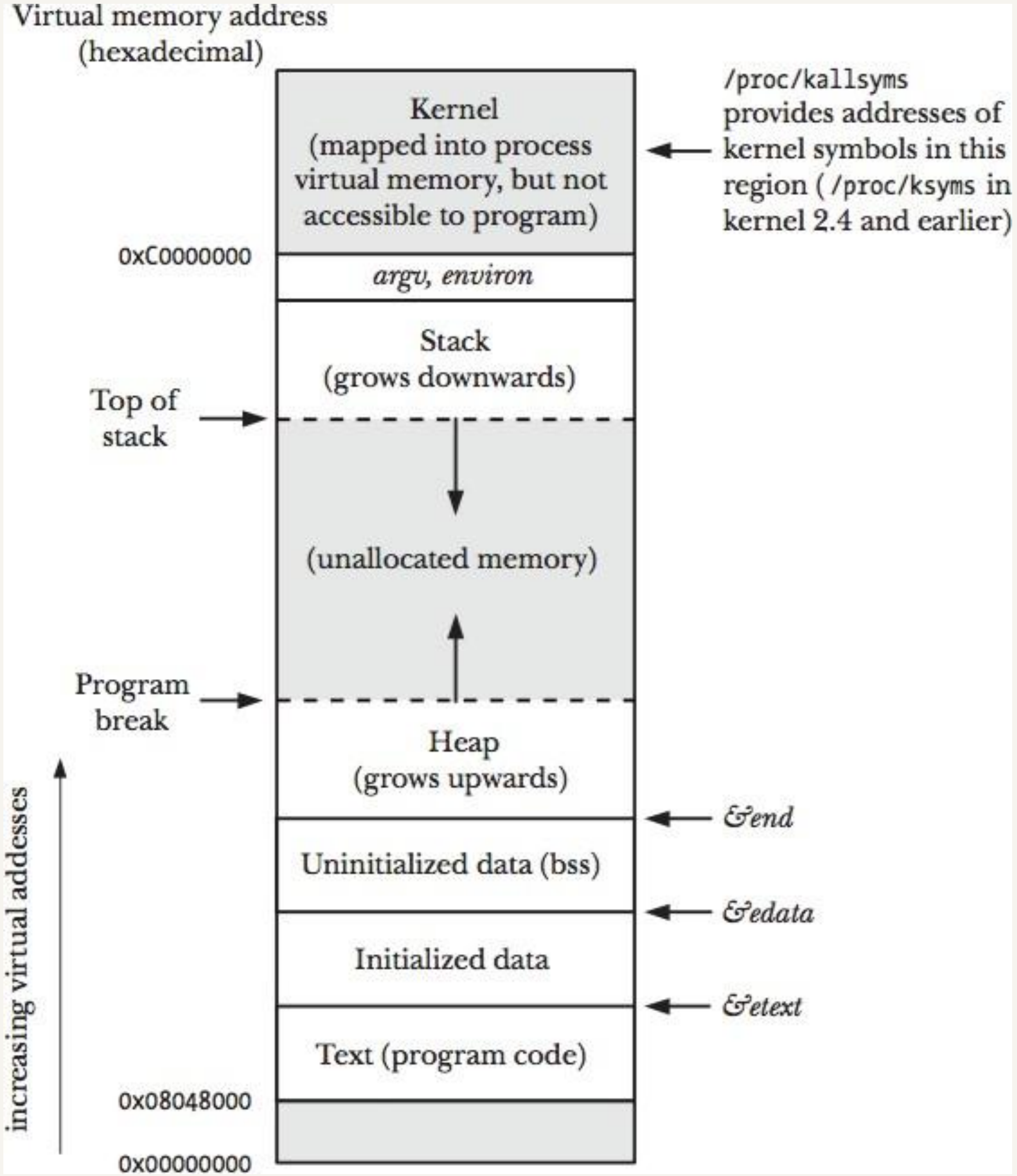
Topics

- Overview of Memory Management
- Intro to P3
  - *How to Start Malloc*
- Project Structure
- Debugging & Performance Tools
- Logistics
  - *Grading*
  - *Testing Framework*

Overview of Memory Management

The Heap

- Persistent, unmanaged memory granted to processes
    - Memory leak
        - Hold onto memory for too long
    - Memory Corruption
        - Free memory too early
- Sometimes memory allocation strategies are coupled with garbage collectors
- Managed by the `malloc()` family in `stdlib`


Virtual memory address
(hexadecimal)
Kernel
(mapped into process virtual memory, but not accessible to program)
/proc/kallsyms provides addresses of kernel symbols in this region (/proc/ksyms in kernel 2.4 and earlier)
0xC0000000
argv, environ
Stack
(grows downwards)
Top of stack
(unallocated memory)
Program break
Heap
(grows upwards)
&end
Uninitialized data (bss)
&edata
Initialized data
&etext
Text (program code)
increasing virtual addresses
0x08048000
0x00000000

The Goal

Lots of allocators are available

- Hoard
- Google’s TCMalloc
- Glibc

Resource tradeoff

- Time
  - Instantly access an available block
- Space
  - Find a block that fits exactly

Intro to P3

Getting Started

Fork

Fork the repo

- https://git.cs.vt.edu/cs3214-staff/malloclab
- Set to private
- You WILL be graded on git usage

Review

Review the sample implementation

- mm-gback-implicit.c
- Take a close look at design decisions and function preconditions
- Be mindful of conversions between word and bytes

Create

Create and write mm.c

Provided Functions

1

void* mem_sbrk(int incr);

- Extend the heap by incr bytes and return the start address

2

void* mem_heap_lo(void);

- Return the start address of the heap

3

void* mem_heap_hi(void);

- Return the end address of the heap

4

size_t mem_heapsize(void);

- Return the current size of the heap

5

size_t mem_pagesize(void);

- Return the system's page size in bytes

Client-Side Functions

Build off the implicit implementation from Dr. Back

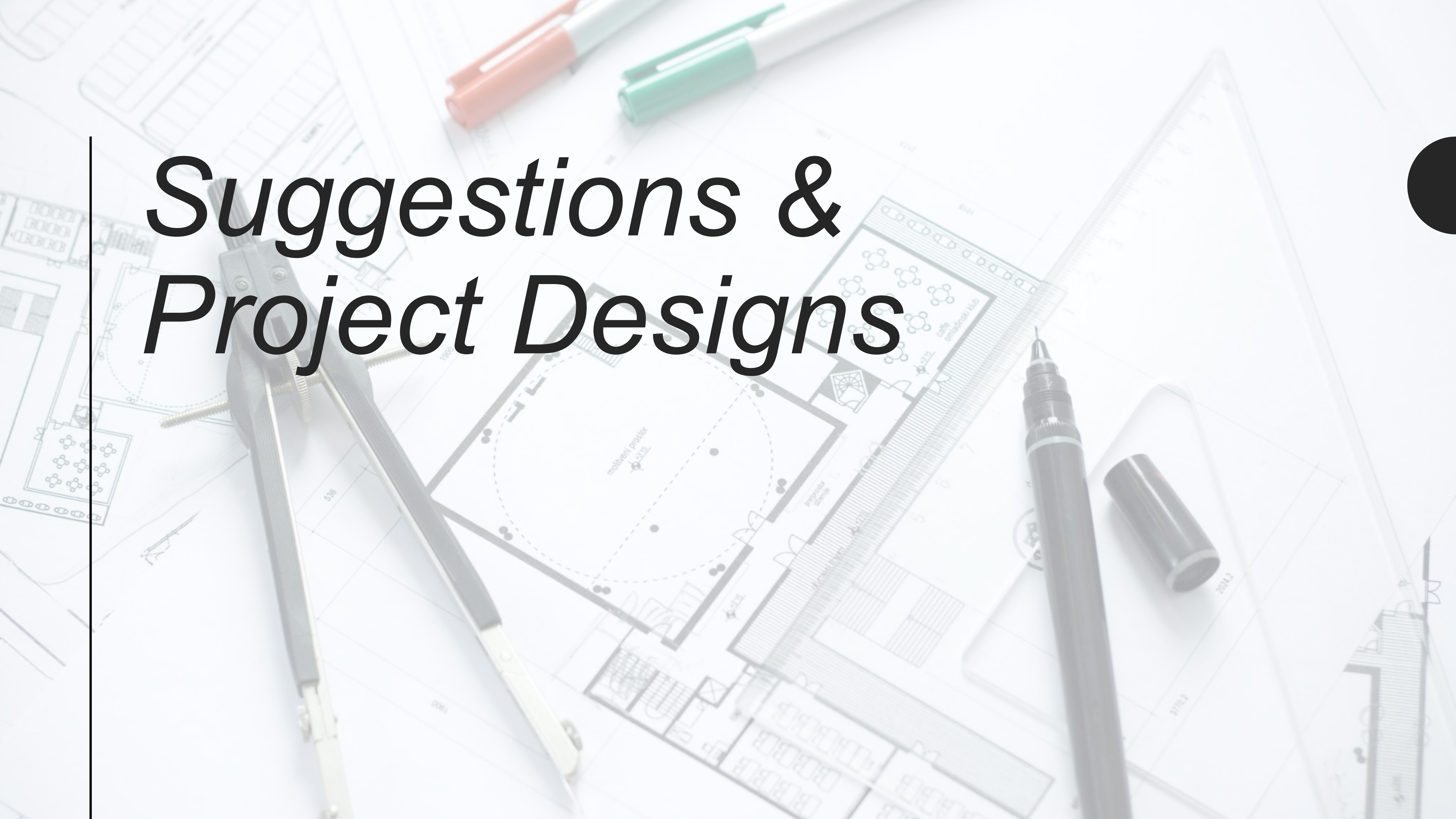
- Int Mm_init(void);
- Void* Mm_malloc(size_t size);
- Void Mm_free(void*ptr);
- Void* mm_realloc(void* ptr, size_t size);

Any helper methods you find suitable

- find_fit()
- place()
- coalesce()

You must be able to handle a wide variety of sizes

Suggestions & Project Designs

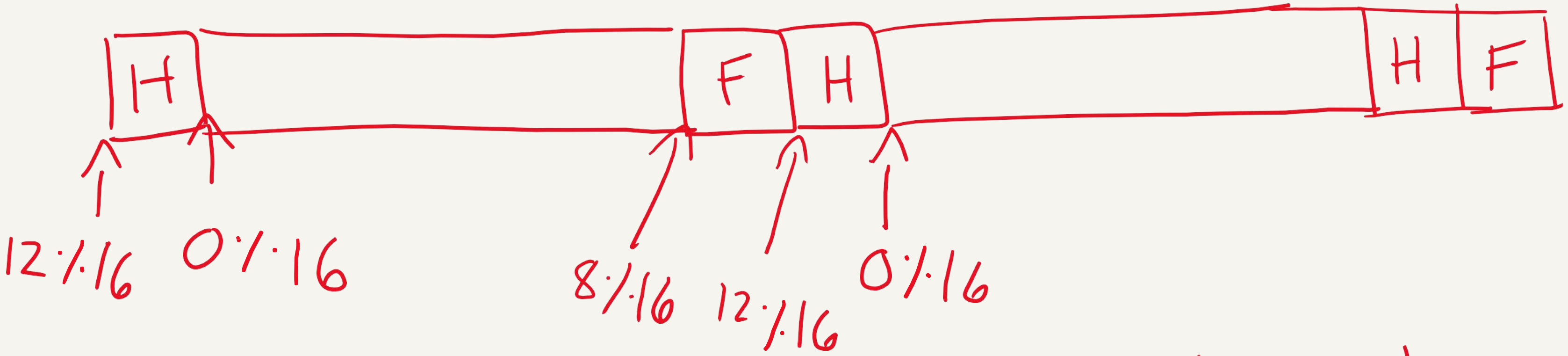


Suggestions

Performance	Consider performance implications from the start •Do extra structs/fields require more memory? •What edge conditions are important? •Avoid high time complexity operations whenever possible!
Asserts	Use assert statements liberally •Ensure alignment •Test for pre- and post-conditions often •Figure out where the bug occurs, rather than a side effect •You will need at least 5 assert statements in your design
Timeline	Start early and Implement in stages •Play with different designs

Link to lecture slides:
https://docs.google.com/presentation/d/1IC6Kghz-y2OMlzZrI8HRJU4RoDgMr6n7c0lG5tSIpWs/edit#slide=id.g120f7216323_2_722

Alignment

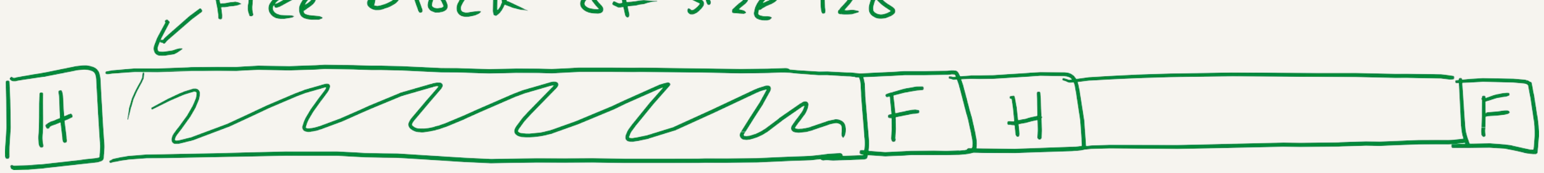


All payloads must be aligned at 0 %16.

Since headers and footers are 4 bytes each, need to pad and align accordingly

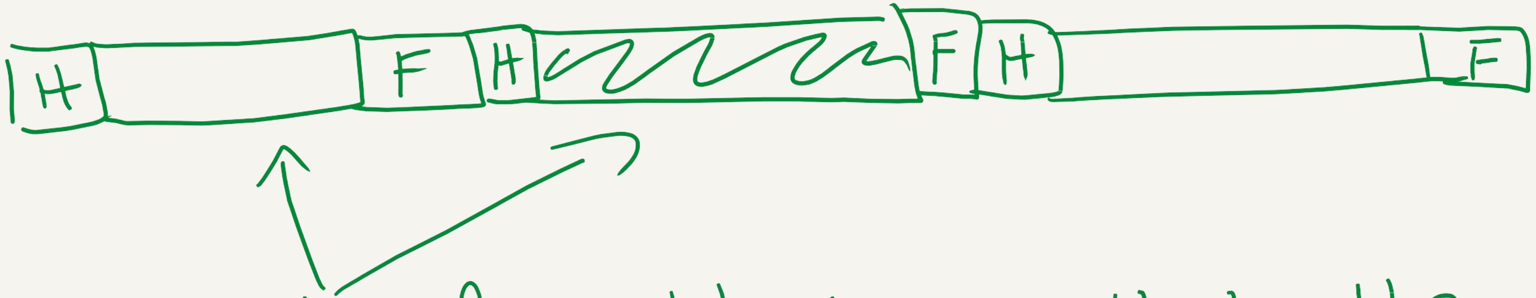
As a side note, when you add pointers remember rlogin is a 64-bit system and thus the pointers are 8 bytes in length.

Splitting



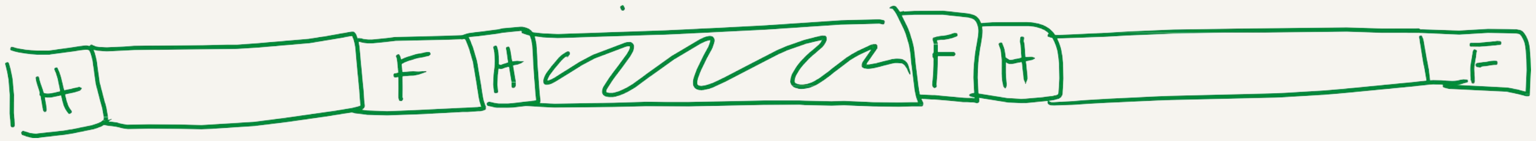
Free block of size 128

User makes malloc(16) call

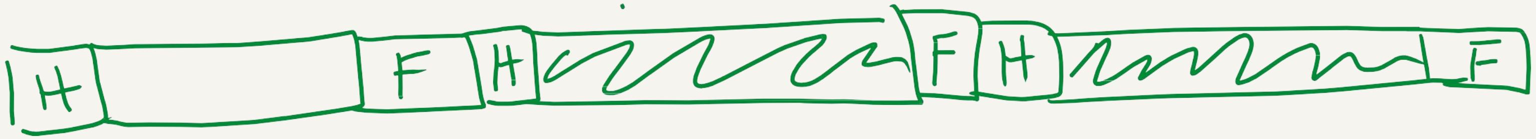


Split the free block so that the unused space from the 16 byte payload can still be used for other data.

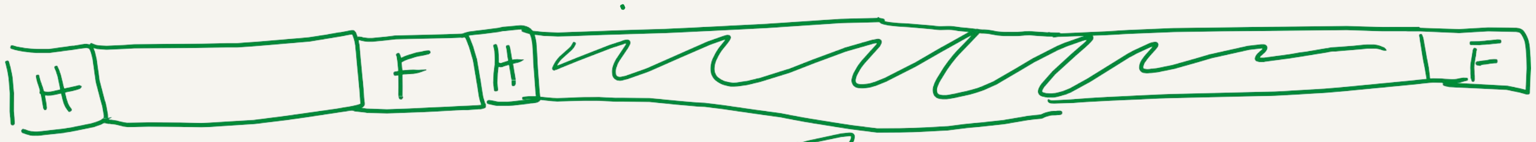
Coalescing



The user makes a free call to the last block



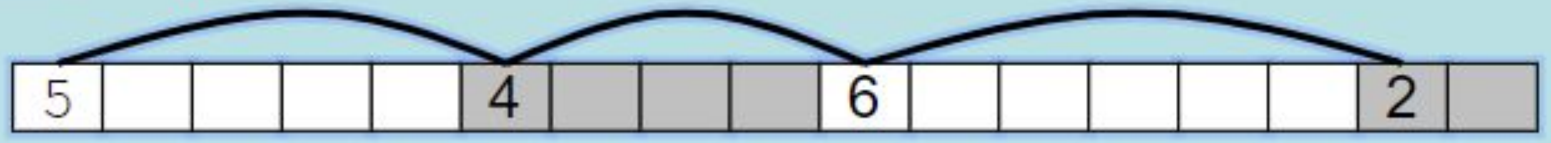
We now have two adjacent free blocks.
If we combine them together, the free block is able to hold larger blocks of. data.



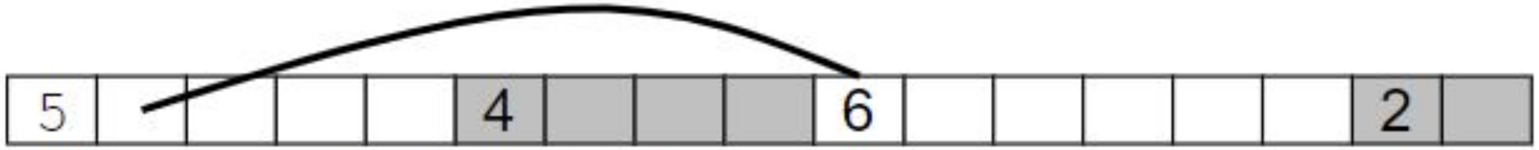
No header and footer in middle

Keeping Track of Free Blocks

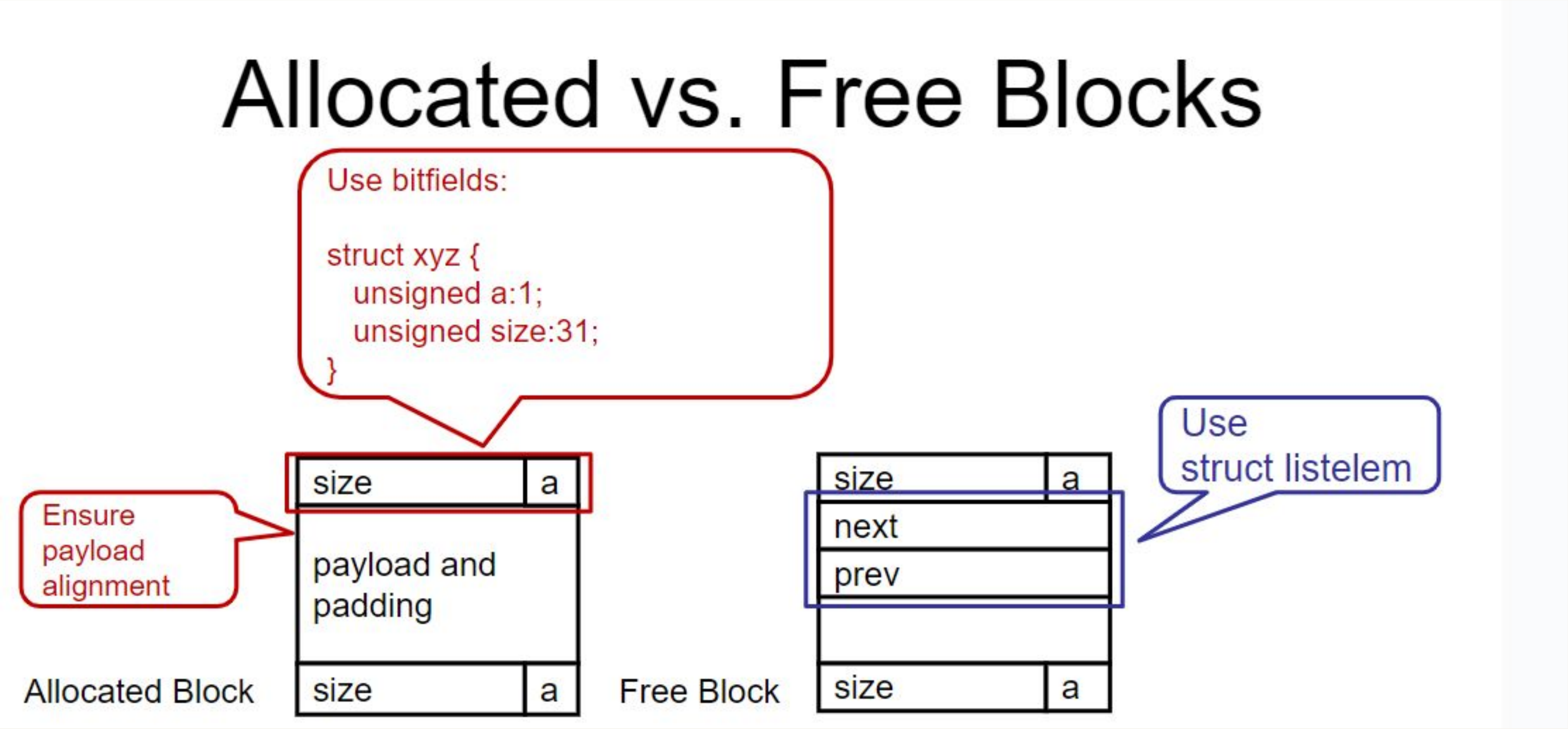
- ***Method 1***: *Implicit list* using lengths -- links all blocks



- ***Method 2***: *Explicit list* among the free blocks using pointers within the free blocks



- ***Method 3***: *Segregated free list*
  - Different free lists for different size classes
- ***Method 4***: Blocks sorted by size
  - Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key



Allocated vs. Free Blocks

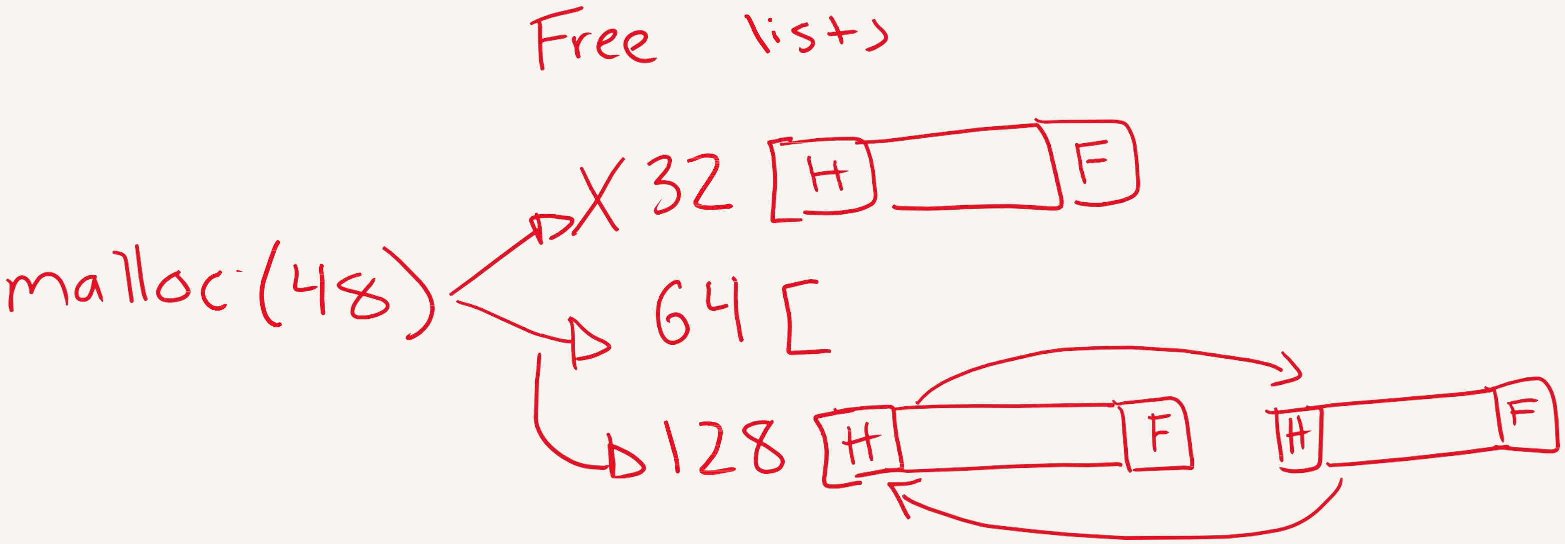
Use bitfields:

```
struct xyz {
  unsigned a:1;
  unsigned size:31;
}
```

Ensure payload alignment

Use struct listelem

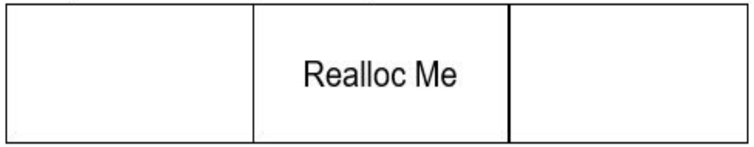
Segregated Free Lists



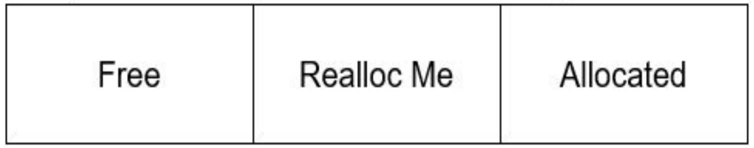
We know a block of size 48 won't fit in a 32-byte block, so we look through the 64-byte block free list. Since that list is empty, we go on to the next biggest list no until we find a free block or determine no free blocks.

Realloc Optimizations

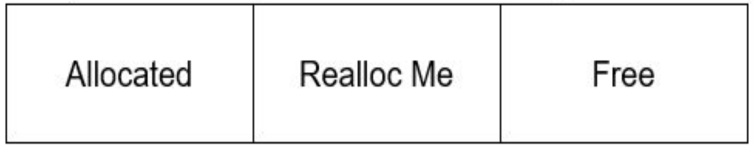
Case 0
Original Size: [********************]
Requested Size: [*********]



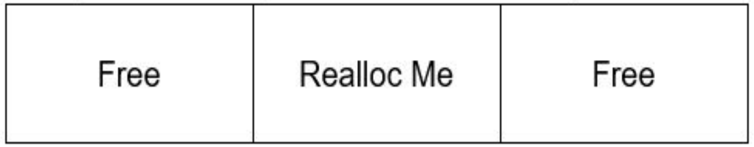
Case 1
Original Size: [********************]
Requested Size: [*************************]



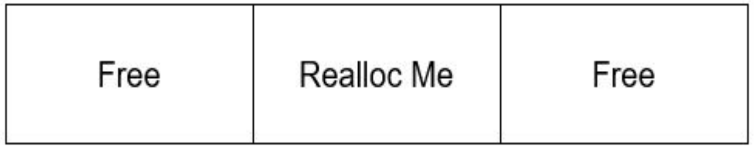
Case 2
Original Size: [********************]
Requested Size: [*************************]



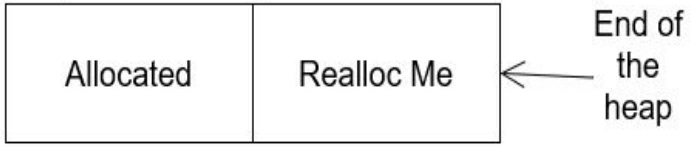
Case 3
Original Size: [*******************]
Requested Size: [*************************]



Case 4
Original Size: [********************]
Requested Size: [***]



Case 5
Original Size: [********************]
Requested Size: [****************************]



Debugging & Performance Tools

Debugging

mm_checkheap()

- Internal mechanism to check the integrity of the heap through linear iteration
- You will have to implement this to fit your design

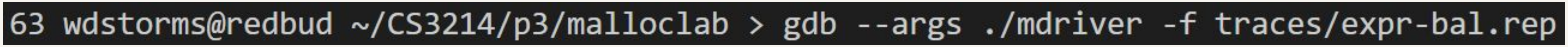
GDB

- Check the actual values of variables

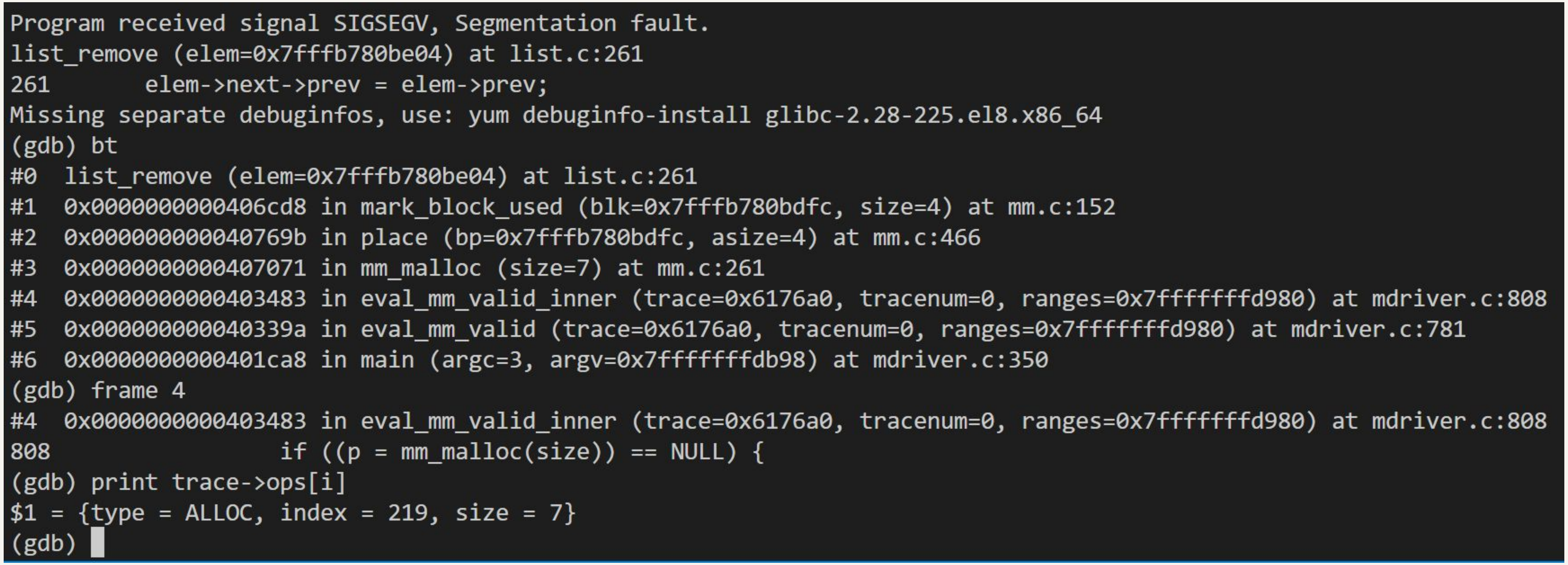
Performance Tools

- `gprof`
  - Tool that counts function calls and exec time, creating `gmon.out`
  - Requires `–pg` flag
    - Remove this flag during performance testing
  - Check output using
    - `gprof mdriver gmon.out > prof_output`
- `perf`
  - Same thing basically, but without `–pg` flag
  - Called with `perf record` then `perf report`

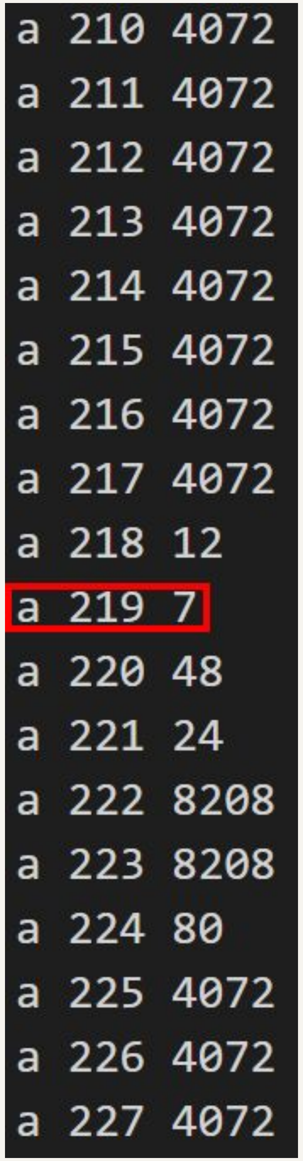
Debugging Demo



```
63 wdstorms@redbud ~/CS3214/p3/malloclab > gdb --args ./mdriver -f traces/expr-bal.rep
```



```
Program received signal SIGSEGV, Segmentation fault.
list_remove (elem=0x7fffb780be04) at list.c:261
261       elem->next->prev = elem->prev;
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-225.el8.x86_64
(gdb) bt
#0  list_remove (elem=0x7fffb780be04) at list.c:261
#1  0x0000000000406cd8 in mark_block_used (blk=0x7fffb780bdfc, size=4) at mm.c:152
#2  0x000000000040769b in place (bp=0x7fffb780bdfc, asize=4) at mm.c:466
#3  0x0000000000407071 in mm_malloc (size=7) at mm.c:261
#4  0x0000000000403483 in eval_mm_valid_inner (trace=0x6176a0, tracenum=0, ranges=0x7fffffffd980) at mdriver.c:808
#5  0x000000000040339a in eval_mm_valid (trace=0x6176a0, tracenum=0, ranges=0x7fffffffd980) at mdriver.c:781
#6  0x0000000000401ca8 in main (argc=3, argv=0x7fffffffdb98) at mdriver.c:350
(gdb) frame 4
#4  0x0000000000403483 in eval_mm_valid_inner (trace=0x6176a0, tracenum=0, ranges=0x7fffffffd980) at mdriver.c:808
808                 if ((p = mm_malloc(size)) == NULL) {
(gdb) print trace->ops[i]
$1 = {type = ALLOC, index = 219, size = 7}
(gdb) 
```



```
a 210 4072
a 211 4072
a 212 4072
a 213 4072
a 214 4072
a 215 4072
a 216 4072
a 217 4072
a 218 12
a 219 7
a 220 48
a 221 24
a 222 8208
a 223 8208
a 224 80
a 225 4072
a 226 4072
a 227 4072
```

Project Logistics

Logistics

Submit code that compiles

- Test using the driver locally before submitting

Grading

- Tests will be run 3-5 times, taking the average
- If a single failure occurs, you get a 0
- Components
  - Correctness (40%)
  - Performance (40%)
    - Throughput
    - Space Utilization
  - Design/Documentation/Git (20%)
    - At least 5 assert statements

Driver

- `./mdriver`
  - `Flags`
  - `-v` for verbose
  - `-V` for MORE verbose
  - `-f` to customize traces
  - `-s` to vary allocation size
  - `-h` for these (and more) flags

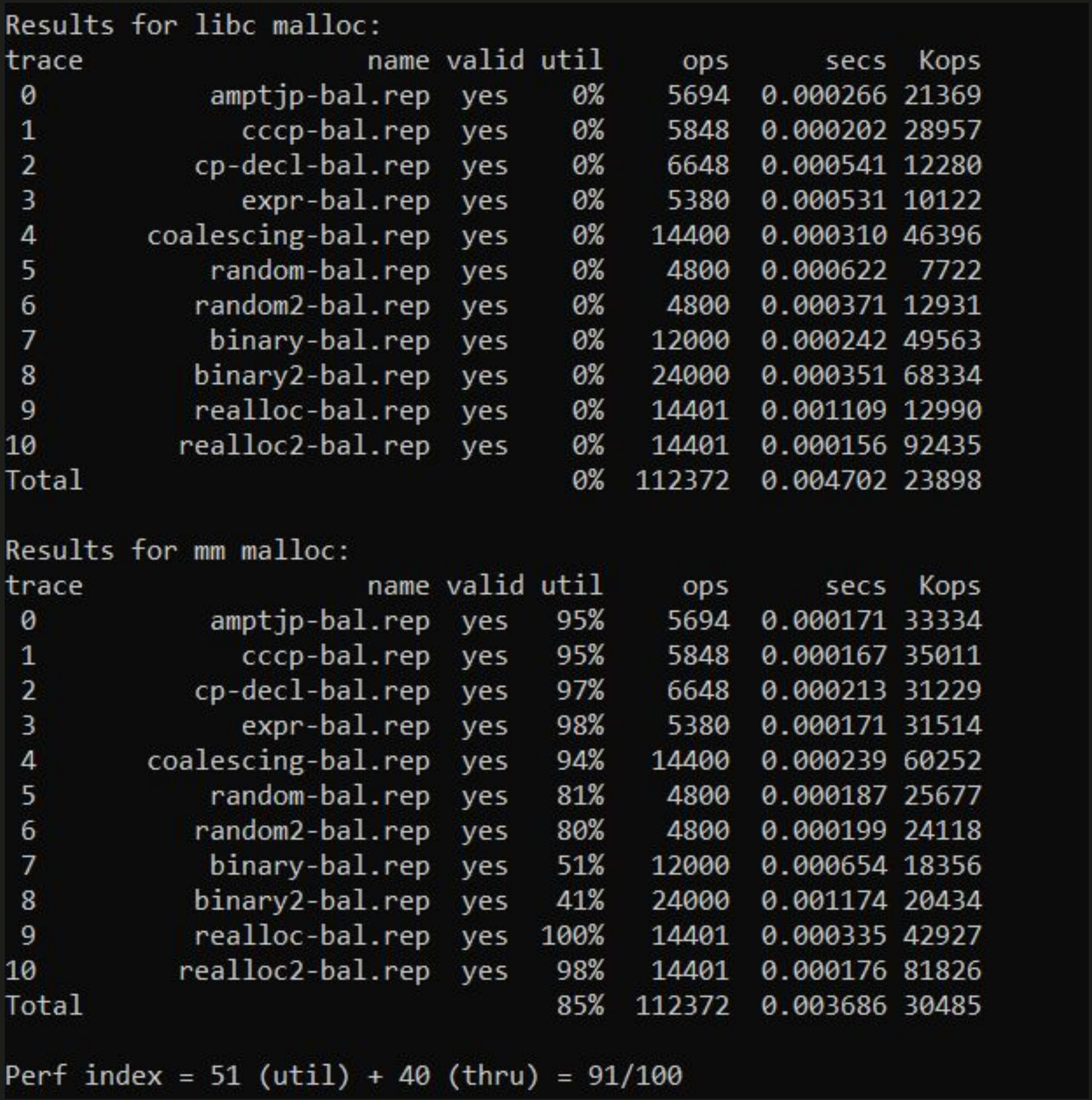
Performance

Throughput

- Number of requests per second

Utilization

- How much space the heap has been expanded by versus the space user data takes
- Overhead
- Fragmentation



```
Results for libc malloc:
trace                  name valid util     ops      secs  Kops
 0           amptjp-bal.rep  yes    0%    5694  0.000266 21369
 1             cccp-bal.rep  yes    0%    5848  0.000202 28957
 2          cp-decl-bal.rep  yes    0%    6648  0.000541 12280
 3             expr-bal.rep  yes    0%    5380  0.000531 10122
 4       coalescing-bal.rep  yes    0%   14400  0.000310 46396
 5           random-bal.rep  yes    0%    4800  0.000622  7722
 6          random2-bal.rep  yes    0%    4800  0.000371 12931
 7           binary-bal.rep  yes    0%   12000  0.000242 49563
 8          binary2-bal.rep  yes    0%   24000  0.000351 68334
 9          realloc-bal.rep  yes    0%   14401  0.001109 12990
10         realloc2-bal.rep  yes    0%   14401  0.000156 92435
Total                               0%  112372  0.004702 23898

Results for mm malloc:
trace                  name valid util     ops      secs  Kops
 0           amptjp-bal.rep  yes   95%    5694  0.000171 33334
 1             cccp-bal.rep  yes   95%    5848  0.000167 35011
 2          cp-decl-bal.rep  yes   97%    6648  0.000213 31229
 3             expr-bal.rep  yes   98%    5380  0.000171 31514
 4       coalescing-bal.rep  yes   94%   14400  0.000239 60252
 5           random-bal.rep  yes   81%    4800  0.000187 25677
 6          random2-bal.rep  yes   80%    4800  0.000199 24118
 7           binary-bal.rep  yes   51%   12000  0.000654 18356
 8          binary2-bal.rep  yes   41%   24000  0.001174 20434
 9          realloc-bal.rep  yes  100%   14401  0.000335 42927
10         realloc2-bal.rep  yes   98%   14401  0.000176 81826
Total                              85%  112372  0.003686 30485

Perf index = 51 (util) + 40 (thru) = 91/100
```

Test Trace Files

```
3000000        // Heap size
2847           // Unique identifiers
5694           // Number of operations
1              // Weight of trace
a 0 2040
f 0
```

- Located in `/home/courses/cs3214/malloclab/traces`

Reference

[L-MEM1] Dynamic Memory Management (malloc/free)

Implicit vs Explicit

Fragmentation

Coalescing Policies

Questions?

Thank you for attending!