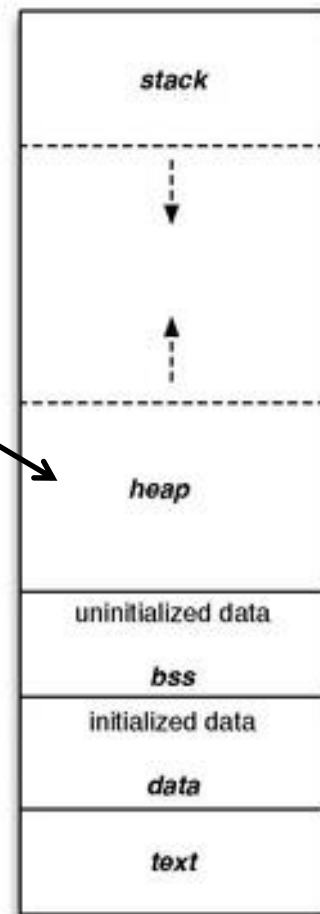


# Recitation 10: Malloc Lab

# What's malloc?

- **A function to allocate memory during runtime (dynamic memory allocation).**
  - More useful when the size or number of allocations is unknown until runtime (e.g. data structures)
- **There's a segment of memory addresses reserved almost exclusively for malloc to use.**
  - Your code directly manipulates the bytes of memory in this section.

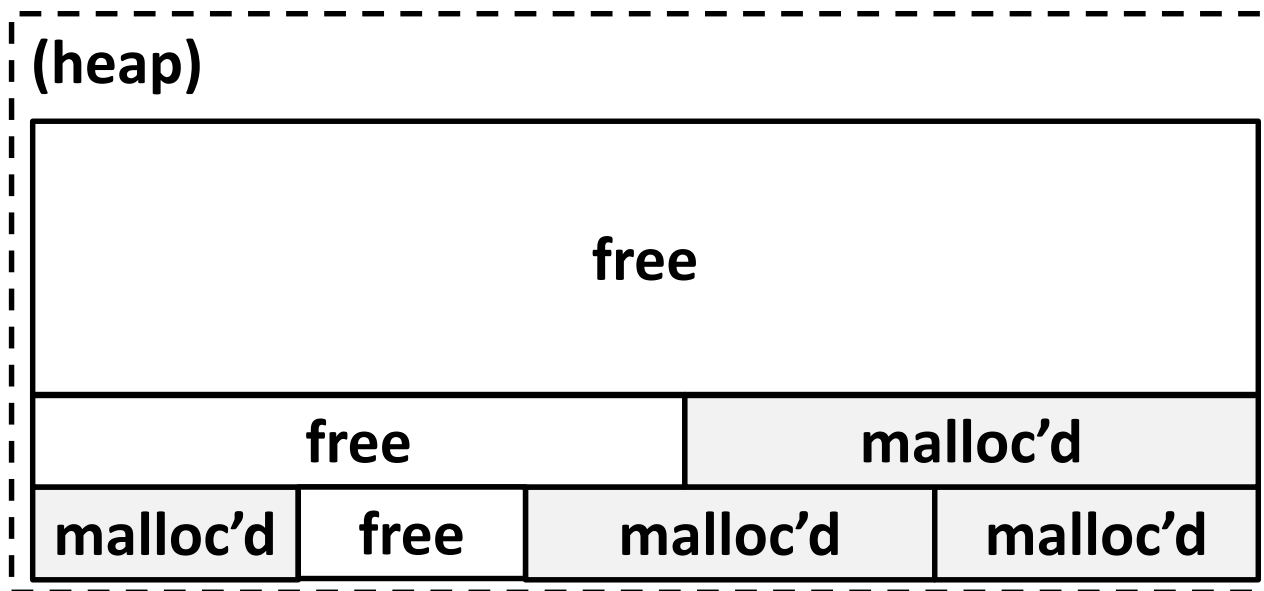


# Outline

- **Concept**
- **How to choose blocks**
- **Metadata**
- **Debugging / GDB Exercises**

# Malloc Internals

- The heap consists of blocks of memory



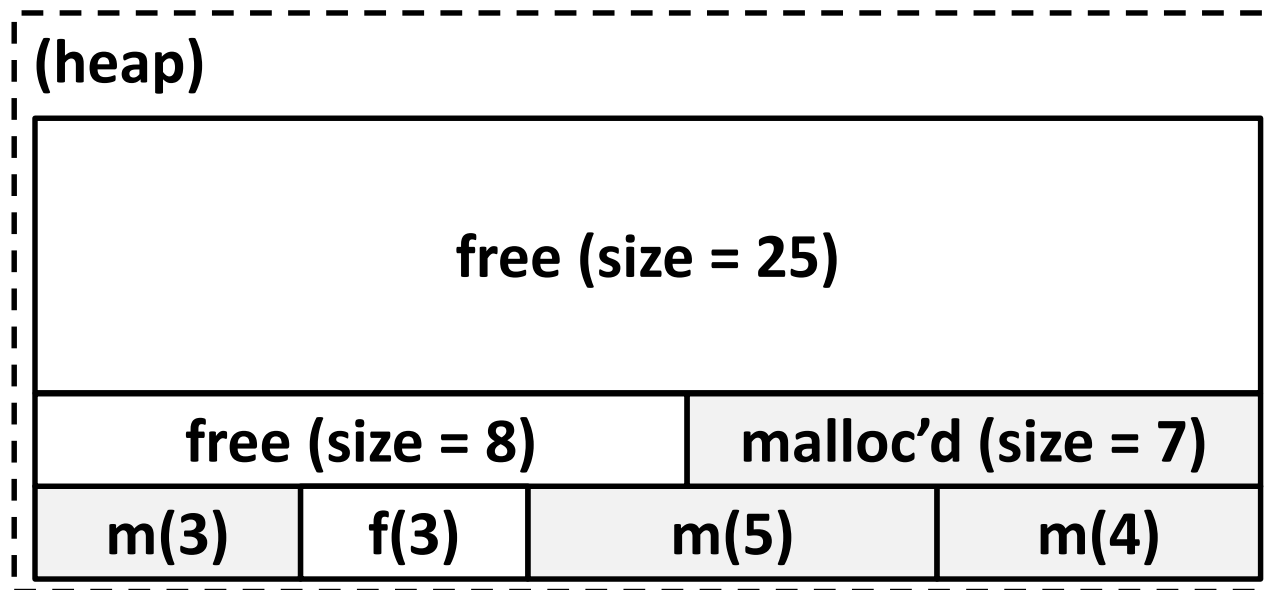
# Concept

- Really, malloc only does three things:
  1. Organize all blocks and store information about them in a structured way.
  2. Using the structure made in 1), choose an appropriate location to allocate new memory.
  3. Update the structure made in 1) when the user frees a block of memory.

This process occurs even for a complicated algorithm like segregated lists.

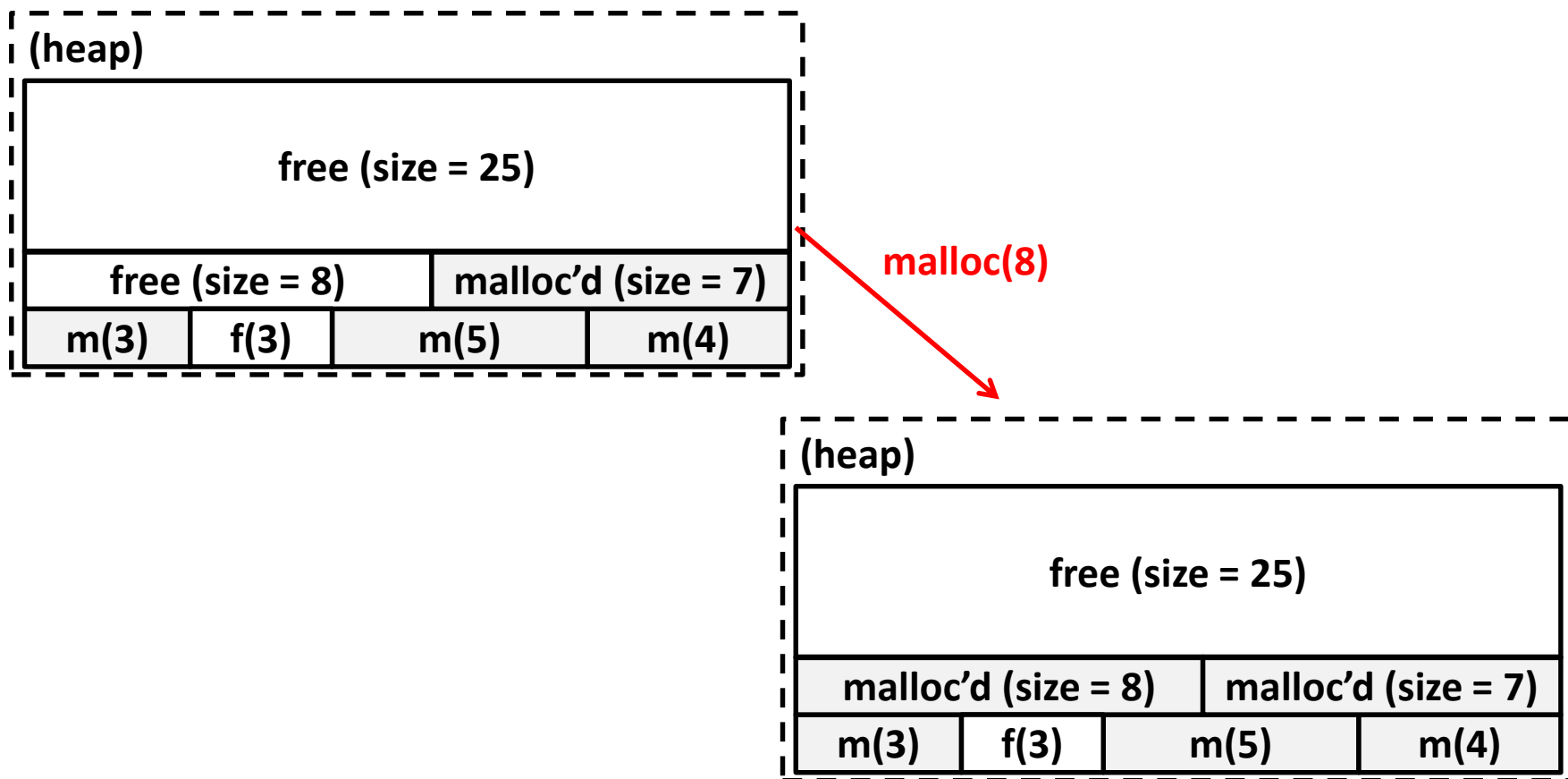
# Concept (Implicit list)

1. Organize all blocks and store information about them in a structured way.



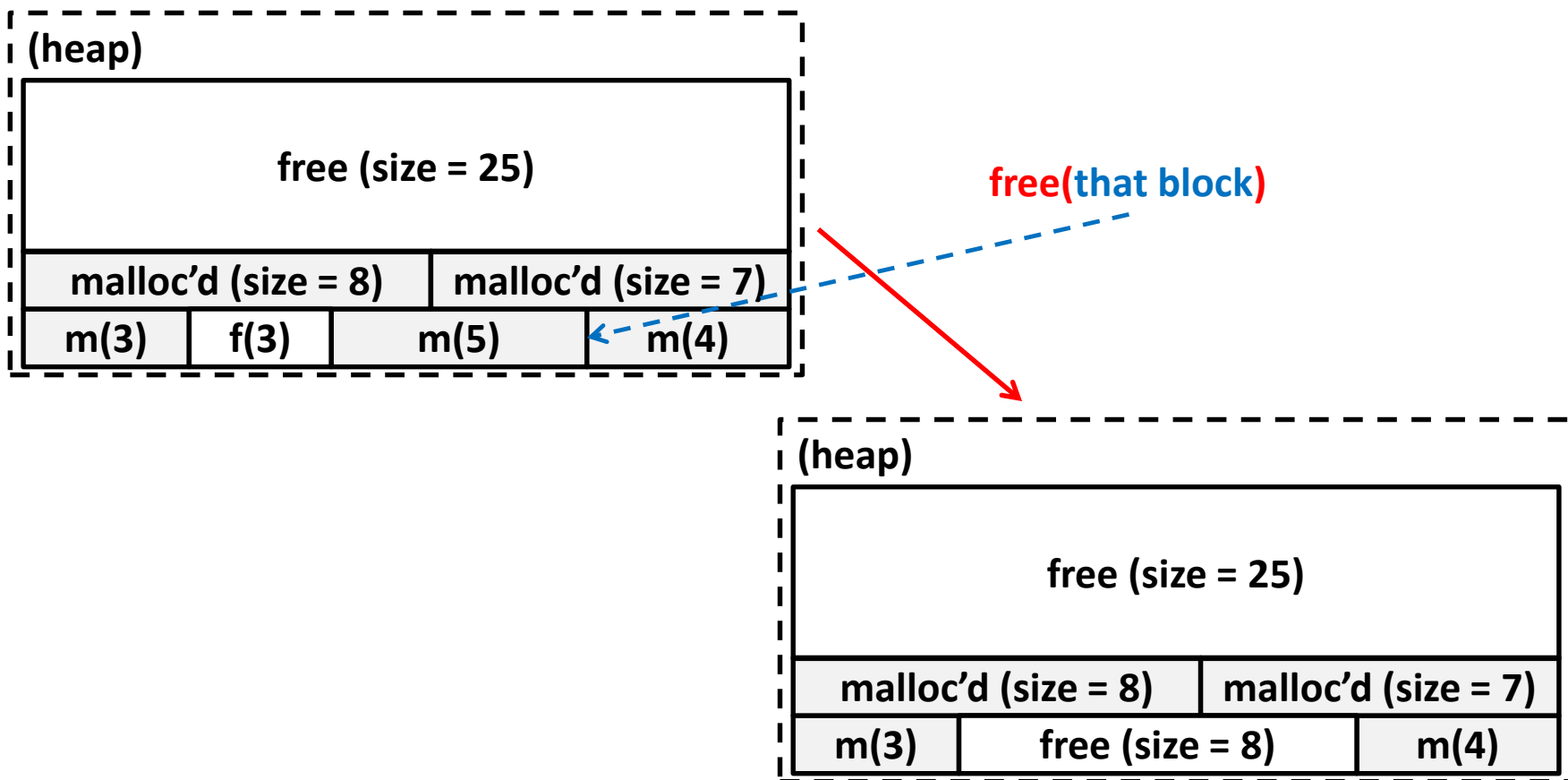
# Concept (Implicit list)

2. Using the structure made in 1), choose an appropriate location to allocate new memory.



# Concept (Implicit list)

3. Update the structure made in 1) when the user frees a block of memory.





# Goals

- **Run as fast as possible**
- **Waste as little memory as possible**
  - Seemingly conflicting goals, but with cleverness you can do very well in both areas.
- **The simplest implementation is the implicit list.**  
**mm-baseline uses this method.**
  - Unfortunately...

```
[dalud@angelshark:~/../15213/s17/malloclabcheckpoint-handout] $ ./mdriver -p
Found benchmark throughput 13090 for cpu type Intel(R)Xeon(R)CPUE5520@2.27GHz, benchmark checkpoint

Throughput targets: min=2618, max=11781, benchmark=13090
.....
Results for mm malloc:
 valid   util    ops    msecs   Kops  trace
  yes    78.4%    20     0.002   9632  ./traces/syn-array-short.rep
  yes    13.4%    20     0.001  25777  ./traces/syn-struct-short.rep
  yes    15.2%    20     0.001  24783  ./traces/syn-string-short.rep
  yes    73.1%    20     0.001  19277  ./traces/syn-mix-short.rep
  yes    16.0%    36     0.001  31192  ./traces/ngram-fox1.rep
  yes    73.6%   757     0.145   5237  ./traces/syn-mix-realloc.rep
* yes    62.0%  5748     3.925   1464  ./traces/bdd-aa4.rep
* yes    58.3%  87830  1682.766    52  ./traces/bdd-aa32.rep
* yes    58.0%  41080   410.385   100  ./traces/bdd-ma4.rep
* yes    58.1% 115380  4636.711    25  ./traces/bdd-nq7.rep
* yes    56.6%  20547   26.677   770  ./traces/cbit-abs.rep
* yes    55.8%  95276   675.303   141  ./traces/cbit-parity.rep
* yes    58.0%  89623   611.511   147  ./traces/cbit-satadd.rep
* yes    49.6%  50583   185.382   273  ./traces/cbit-xyz.rep
* yes    40.6%  32540    76.919   423  ./traces/ngram-gulliver1.rep
* yes    42.4% 127912  1284.959   100  ./traces/ngram-gulliver2.rep
* yes    39.4%  67012   338.591   198  ./traces/ngram-moby1.rep
* yes    38.6%  94828   701.305   135  ./traces/ngram-shake1.rep
* yes    90.9%  80000  1455.891    55  ./traces/syn-array.rep
* yes    88.0%  80000   915.167    87  ./traces/syn-mix.rep
* yes    74.3%  80000   914.366    87  ./traces/syn-string.rep
* yes    75.2%  80000   812.748    98  ./traces/syn-struct.rep
16 16    59.1% 1148359 14732.604    78

Average utilization = 59.1%. Average throughput = 78 Kops/sec
Checkpoint Perf index = 20.0 (util) + 0.0 (thru) = 20.0/100
```

# In case you didn't preview

- **Allocation methods, in a nutshell**
- **Implicit list: A list is implicitly formed by jumping between blocks, using knowledge about their sizes.**
- **Explicit list: Free blocks explicitly point to other blocks, like in a linked list.**
  - Understanding explicit list requires understanding implicit list
- **Segregated list: Multiple linked lists, each containing blocks in a certain range of sizes.**
  - Understanding segregated lists requires understanding explicit list

# Choices

- **What kind of implementation to use?**
  - Implicit list, explicit list, segregated lists, binary tree methods ...etc
  - Can use specialized strategies depending on the size of allocations
  - Adaptive algorithms are fine, though not necessary to get 100%.
    - But please, don't directly test for which trace file is running.
  
- **What fit algorithm to use?**
  - Best fit: choose the smallest block that is big enough to fit the requested allocation size
  - First fit / next fit: search linearly starting from some location, and pick the first block that fits.
  - Which one's faster, and which one uses less memory?
  
- **This lab has many more ways to get an A+ than, say, Cache lab part 2**

# Finding a Best Block

- **Suppose you have implemented the explicit list approach**
  - You were using best fit with explicit lists
- **You experiment with using segregated lists instead. Still using best fits.**
  - Will your memory utilization score improve?

*Note: you don't have to implement seglists and run mdriver to answer this. That's, uh, hard to do within one recitation session.*

- What other advantages does segregated lists provide?
- **Losing memory because of the way you choose your free blocks is called external fragmentation.**

# Metadata

- **All blocks need to store some data about themselves in order for `malloc` to keep track of them (e.g. headers)**
  - This takes memory too...
  - Losing memory for this reason is called internal fragmentation.
- **What data might a block need?**
  - Does it depend on the malloc implementation you use?
  - Is it different between free and allocated blocks?
- **Can we use the extra space in free blocks?**
  - Or do we have to leave the space alone?
- **How can we overlap two different types of data at the same location?**

# Hey, your malloc worked! GJ.

- Setting up the blocks, metadata, lists... etc (500 LoC)
- + Finding and allocating the right blocks (500 LoC)
- + Updating your heap structure when you free (500 LoC) =

```
[dalud@angelshark:~/../15213/s17/malloclabcheckpoint-handout] $ ./mdriver
Found benchmark throughput 13056 for cpu type Intel(R)Xeon(R)CPUE5520@2.270

Throughput targets: min=6528, max=11750, benchmark=13056
.....
Results for mm malloc:
  valid   util    ops   msec   Kops  trace
   yes    78.1%    20    0.004   5595 ./traces/syn-array-short.rep
   yes     3.2%    20    0.004   5273 ./traces/syn-struct-short.rep
*  yes    96.0%  80000   17.176   4658 ./traces/syn-array.rep
*  yes    93.2%  80000    6.154  12999 ./traces/syn-mix.rep
*  yes    86.4%  80000    3.717  21521 ./traces/syn-string.rep
*  yes    85.6%  80000    3.649  21924 ./traces/syn-struct.rep
16 16     74.2% 1148359   55.949  20525

Average utilization = 74.2%. Average throughput = 20525 Kops/sec
Perf index = 60.0 (util) + 40.0 (thru) = 100.0/100
```

# Nope. Have fun debugging your code!

- Setting up the blocks, metadata, lists... etc (500 LoC)
- + Finding and allocating the right blocks (500 LoC)
- + Updating your heap structure when you free (500 LoC)
- + **One bug, somewhere lost in those 1500 LoC =**

```
[dalud@angelshark:~/.../15213/s17/malloclabcheckpoint-handout] $ ./mdriver
Found benchmark throughput 13056 for cpu type Intel(R)Xeon(R)CPUE5520@2.27

Throughput targets: min=6528, max=11750, benchmark=13056
.....Segmentation fault
[dalud@angelshark:~/.../15213/s17/malloclabcheckpoint-handout] $ █
```



# GDB Practice

- Using GDB well in malloclab can save you HOURS\* of debugging time
  - Average 20 hours using GDB for “B” on malloclab
  - Average 23 hours not using GDB for “B” on malloclab
- Form pairs
  - Login to a shark machine
  - wget <http://www.cs.cmu.edu/~213/activities/rec11.tar>
  - tar xf rec11.tar
  - cd rec11
  - make
- Two buggy mdriivers

# First things first

## ■ Try running `$ make`

- If you look closely, our code compiles your `malloc` implementation with the `-O3` flag.
- This is an optimization flag. `-O3` makes your code run as efficiently as the compiler can manage, but also makes it horrible for debugging (almost everything is “optimized out”).

```
[dalud@angelshark:~/.../15213/s17/rec11] $ make
gcc -Wall -Wextra -Werror -O3 -g -DDRIVER -Wno-unused-function -Wno-u
./macro-check.pl -f mm.c
clang -Wall -Wextra -Werror -O3 -g -DDRIVER -Wno-unused-function -Wno
gcc -Wall -Wextra -Werror -O3 -g -DDRIVER -Wno-unused-function -Wno-u
(gdb) print block
$3 = <optimized out>
(gdb) print asize
$4 = <optimized out>
```

# Debugging mdriver

```
$ gdb --args ./mdriver -c traces/syn-mix-short.rep
```

```
(gdb) run
```

```
(gdb) backtrace
```

```
(gdb) list
```

Optional: Type Ctrl-X Ctrl-A to see the source code. Don't linger there for long, since this visual mode is buggy. Type that key combination again to go back to console mode.

- 1) What function is listed on the top of backtrace?
- 2) What line of code crashed?
- 3) How did that line cause the crash?

# Debugging mdriver

- **(gdb) x /10gx block**
  - Shows the memory contents within the block
  - In particular, look for the header.
- **Remember the output from (gdb) bt?**
- **(gdb) frame 1**
  - Jumps to the function one level down the call stack (aka the function that called `write_footer`)
  - Ctrl-X, Ctrl-A again if you want to see visuals
- **What was the caller function? What is its purpose?**
  - Was it writing to `block` or `block_next` when it crashed?

# Thought process while debugging

- `write_footer` crashed because it got the wrong address for the footer...
- The address was wrong because the header of the block was some garbage value
  - Since `write_footer` uses `get_size(block)` after all
- **But why in the world does the header contain garbage??**
  - The crash happened in `place`, which basically splits a free block into two and uses the first one to store things.
  - Hm, `block_next` would be the new block created after the split? The one on the right?
  - The header would be in the middle of the original free block actually. Wait, but I wrote a new header before I wrote the footer!
    - Right? ...Oh, I didn't. Darn.

# Heap consistency checker

- **mm-2.c activates debug mode, and so mm\_checkheap runs at the beginning and end of many of its functions.**

```
106 /*
107  * If DEBUG is defined, enable printing on dbg_printf and contracts.
108  * Debugging macros, with names beginning "dbg_" are allowed.
109  * You may not define any other macros having arguments.
110  */
111 #define DEBUG // uncomment this line to enable debugging
112
113 #ifdef DEBUG
114 /* When debugging is enabled, these form aliases to useful functions */
115 #define dbg_printf( ... ) printf( VA_ARGS )
```

- **The next bug will be a total nightmare to find without this heap consistency checker\*.**

# Now you try debugging this

```
$ gdb --args ./mdriver-2 -c traces/syn-array-short.rep
```

`mm_checkheap` will fail. What reason does it cite?

Where's the footer? Use `x /gx` and some arithmetic

Track changes in the header and the footer:

```
(gdb) watch *[header address]
```

```
(gdb) watch *[footer address]
```

When does the footer's value turn inconsistent? What function was running at the time? Which part of that function was wrong? Use `backtrace` and `frame`.

# MallocLab Checkpoint

- Due Thursday
- Checkpoint should take a bit less than half of the time
- Read the writeup. Slowly. Carefully.
- Use GDB
- Ask us for debugging help
  - Only after you implement `mm_checkheap` though



# Appendix: Advanced GDB Usage

- **backtrace**: Shows the call stack
- **frame**: Lets you go to one of the levels in the call stack
- **list**: Shows source code
- **print <expression>**:
  - Runs any valid C command, even something with side effects like `mm_malloc(10)` or `mm_checkheap(1337)`
- **watch <expression>**:
  - Breaks when the value of the expression changes
- **break <function / line> if <expression>**:
  - Only stops execution when the expression holds true
- **Ctrl-X Ctrl-A for visualization**

# Appendix: Building O0

- Edit the file named `Makefile` and make it use `-O0`

```
4 # Regular compiler
5 CC = gcc
6 # Compiler for mm.c
7 CLANG = clang
8 # Change this to -O0 (big-Oh, numeral zero) if you need to use a debugger on your code
9 COPT = -O0
10 CFLAGS = -Wall -Wextra -Werror $(COPT) -g -DDRIVER -Wno-unused-function -Wno-unused-parameter
11 LIBS = -lm -lrt
12
```

- Then run `$ make -B`
  - Alternative: `$ make clean $ make`
  - Just running `make` won't work because it'll say nothing new needs to be compiled. So we force it to recompile.
- Remember to set it back to `-O3` when you're done to test throughput, since `-O0` makes your code much slower.