Memory Allocator Discussion

5/9/22: The Curtain Behind malloc()

A Brief Motivation

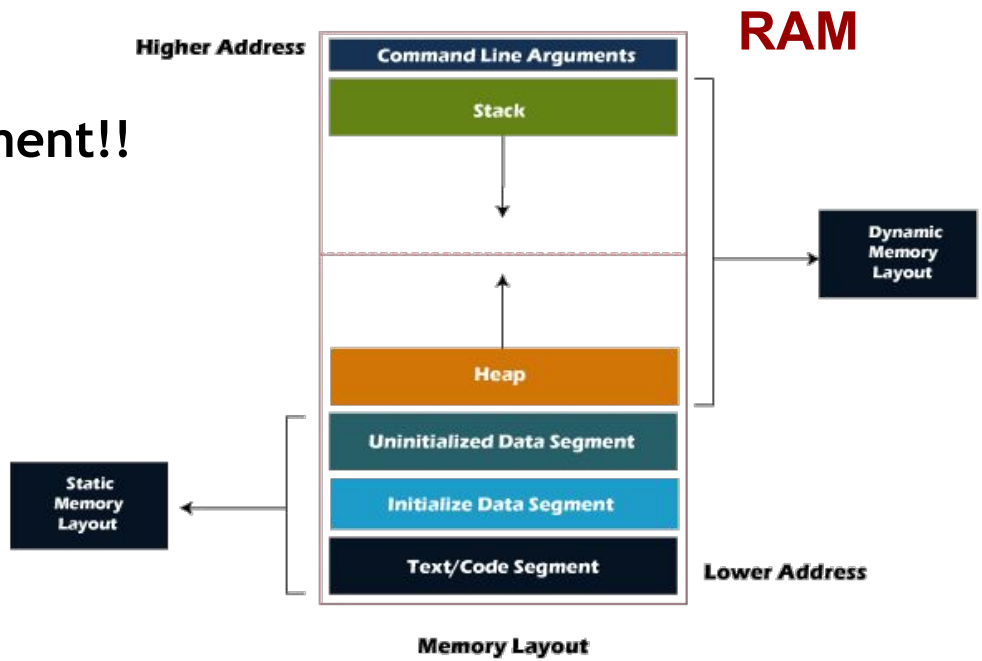
There are 3 “locations” data for a program can reside in: .data segment, stack, and **heap.**

Example Code:

```
unsigned int fifo = 0;
void loop() {
    char input[128];
    …
}
```

.data segment!!

stack!!



Spaces are allocated on the heap with an explicit “malloc” call.

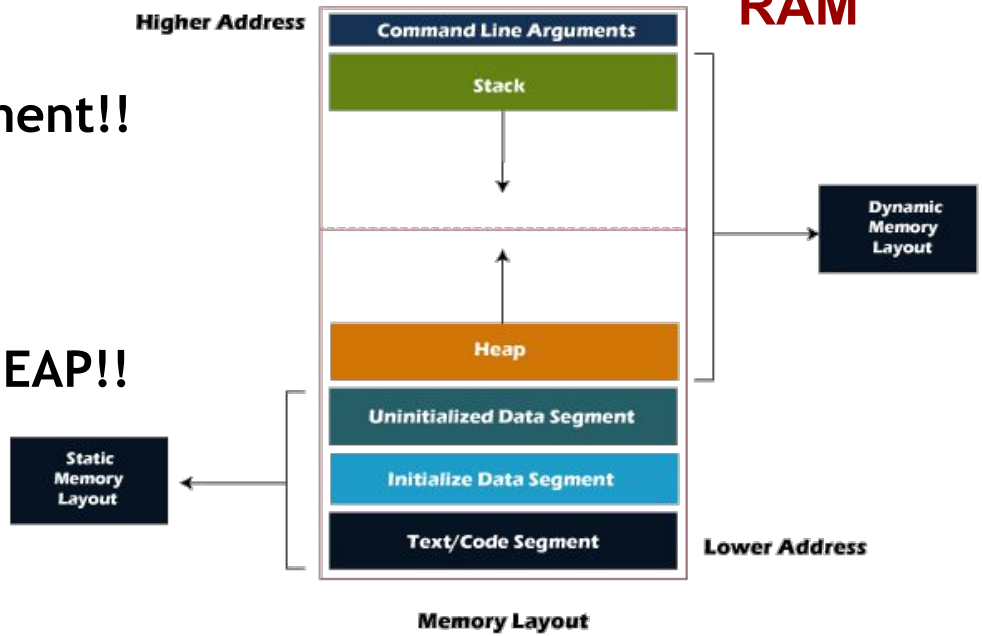
Example Code:

```
unsigned int fifo = 0;
void loop() {
    char input[128];
    char *ex = (char*) malloc(4);
    …
}
```

.data segment!!

stack!!

HEAP!!



Spaces allocated in .data segment and stack **must be** determinable at compile-time.

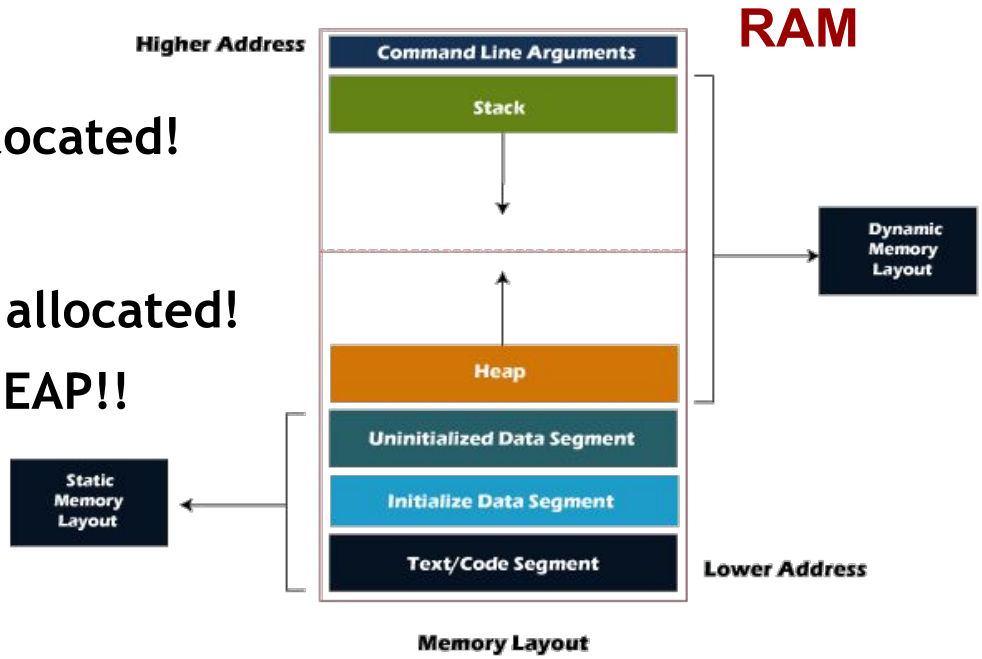
Example Code:

```
unsigned int fifo = 0;
void loop() {
    char input[128];
    char *ex = (char*) malloc(4);
    ...
}
```

4 bytes allocated!

128 bytes allocated!

HEAP!!



Spaces allocated in .data segment and stack **must be** determinable at compile-time.

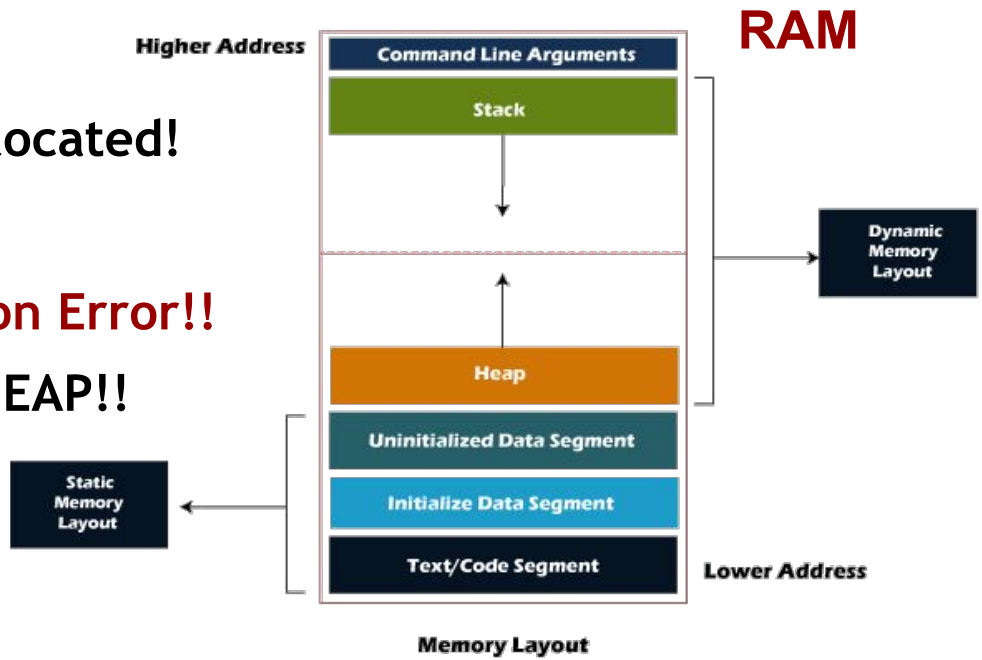
Example Code:

```
unsigned int fifo = 0;
void loop(int size) {
    char input[size];
    char *ex = (char*) malloc(4);
    ...
}
```

4 bytes allocated!

Compilation Error!!

HEAP!!



Spaces allocated on the heap **can be** determined at run-time instead.

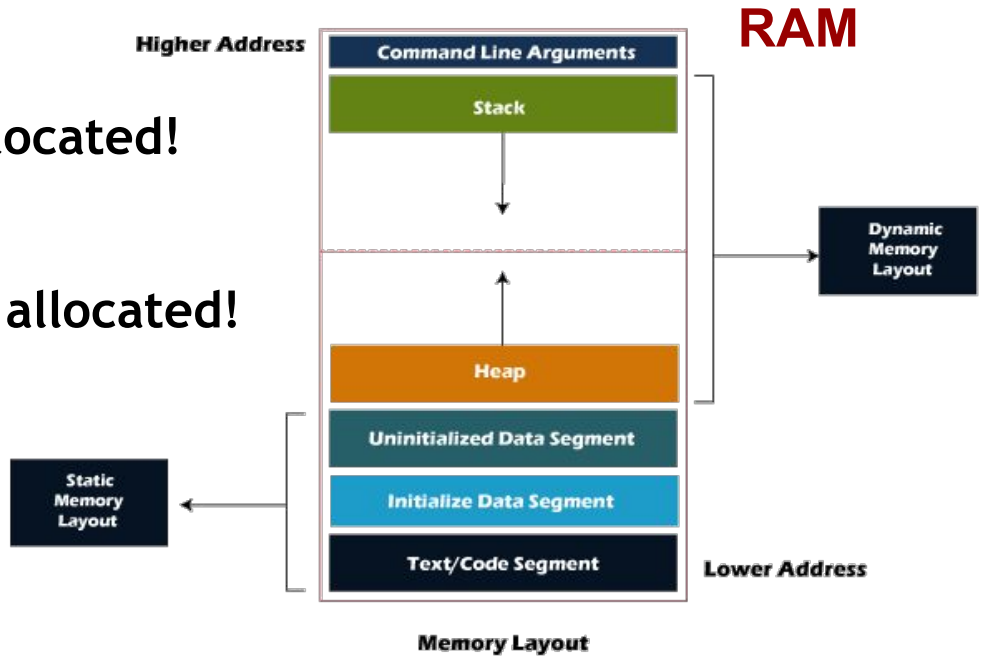
Example Code:

```
unsigned int fifo = 0;
void loop(int size) {
    char input[128];
    char *ex = (char*) malloc(size);
    ...
}
```

4 bytes allocated!

128 bytes allocated!

‘size’ bytes allocated!



Why Use Heap?

Sometimes we don't know how much memory spaces we need until runtime!

For Example:

For Assignment 2 (the shell assignment), we have a constraint where there will be at most 5 processes running concurrently.

What if we remove that constraint? How do we keep track of all the processes? Could be 5, 10, 20, 30, 31, …

Use a linked list where each linked list node lives on the heap!

Assignment 4: Implementing Your Own ‘malloc’

Memory Allocator is what manages the heap during program runtime to allow for dynamic allocation.

However, it presents a new set of challenges and design space…

For the purpose of Assignment 4, here are the terms you should know:

- **Block Splitting**
- **Block Coalescing**
- **Implicit Free List**
- **First-Fit Strategy**

But first, the design of Assignment 4’s heap…

Assignment 4: The Heap Design

Your allocator is managing a “virtualized” heap.

Your “virtualized” heap lives in the .data segment (declared as a global variable):

```
unsigned char heap[64];
```

Think of indexes into “heap” as memory pointers.

Assignment 4: Heap Organized As Implicit Free List

You identify free block by traversing the whole heap. No explicit data structure keeping track of the free blocks (in contrast with *explicit free list*).

Use **first-fit strategy** to allocate the block: allocate the first free block that fits the requested size.

How to traverse the heap:

```
void allocateBlock(int sz){
    int i = 0; // pointer to beginning of heap
    int found = 0; // a flag
    while (!found && i < 64){
        if (blockIsFree(heap[i]) && (getSize(heap[i])-2)>= sz)
            // block is free and fits the requested size
            found = 1; break;
        i += getSize(heap[i]); // go to header of next block
    }
}
```

Assignment 4: Header and Footer Structure

Header and footer are one byte each. The one byte contains two information: block size (most significant 7 bits) and allocation status (least significant bit):

Supposed a block with size 20 (header + payload + footer) and an allocated status, how is it represented by its header and footer?

Representation Of The Byte In Binary:

Decimal 20 == 0x14 == 0b10100

0 0 1 0 1 0 0 1

Block Size

Allocation Status

Assignment 4: Header and Footer Structure

Use bit-masking and/or bitwise shifts to isolate either the block size or the allocation status!

```
// to get block size
// - bitwise AND with 0b 1111 1110
// - bitwise shift to remove allocation status
(header & 0xFE) >> 1
```

Representation Of The Byte In Binary:

Decimal 20 == 0x14 == 0b10100

0 0 1 0 1 0 0 1

Block Size

Allocation Status

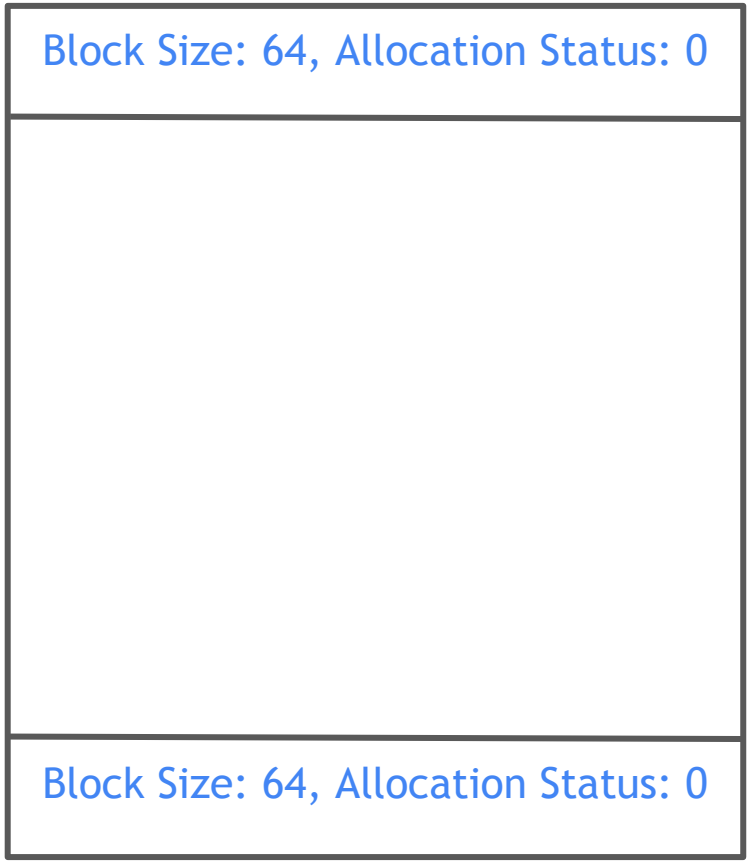
Assignment 4: Block Splitting

Once you find a free block, you need to split the block if free block size minus allocated block size is greater than 2:

- allocated block size = header + payload + footer
- header + footer takes 2 bytes. *If after splitting, there's only 2 bytes of free space left, then there is no point splitting since there is no space for payload.*

Assignment 4: Block Splitting Example

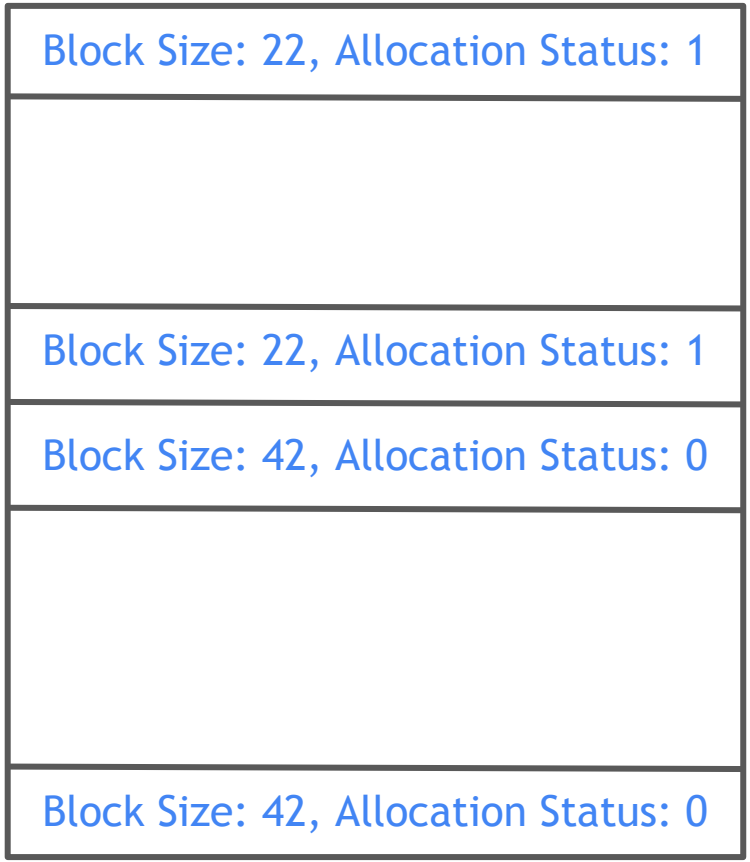
Your Heap Initially:



Assignment 4: Block Splitting Example

After malloc(20):

Note: *blocks (allocated or not) are contiguous!*



Assignment 4: Block Coalescing

WHY: reduce **external fragmentation.** You need to do forward and backward coalescing!

To determine if you need forward coalescing:

```
// find next block's header
int next_header = current_footer + 1
if (current_footer < 63 && blockIsFree(heap[next_header])){
    // perform forward coalescing
    // (1) change current_header's block size to account
    //     for next_header's block size
    // (2) update next_footer's block size. Current_footer
    //     now becomes part of the payload
    // (3) zero'd out each byte of payload. NOTE: this is an
    //     requirement for this assignment. Real malloc will
    //     not do this because of speed!
}
```

Assignment 4: Block Coalescing

WHY: reduce **external fragmentation.** You need to do forward and backward coalescing!

To determine if you need backward coalescing:

```
// find previous block's footer
int previous_footer = current_header - 1
if (current_header > 0 && blockIsFree(heap[previous_footer])){
    // perform backward coalescing
    // (1) update previous_header's block size. current_header
    //     now becomes part of the payload
    // go to previous header
    int previous_header = previous_footer + 1 - previous_block_size
    // (2) change current_footer's block size to account
    //     for previous_header's block size
    // (3) zero'd out each byte of payload. NOTE: this is an
    //     requirement for this assignment. Real malloc will
    //     not do this because of speed!
}
```

Questions?