## Notes

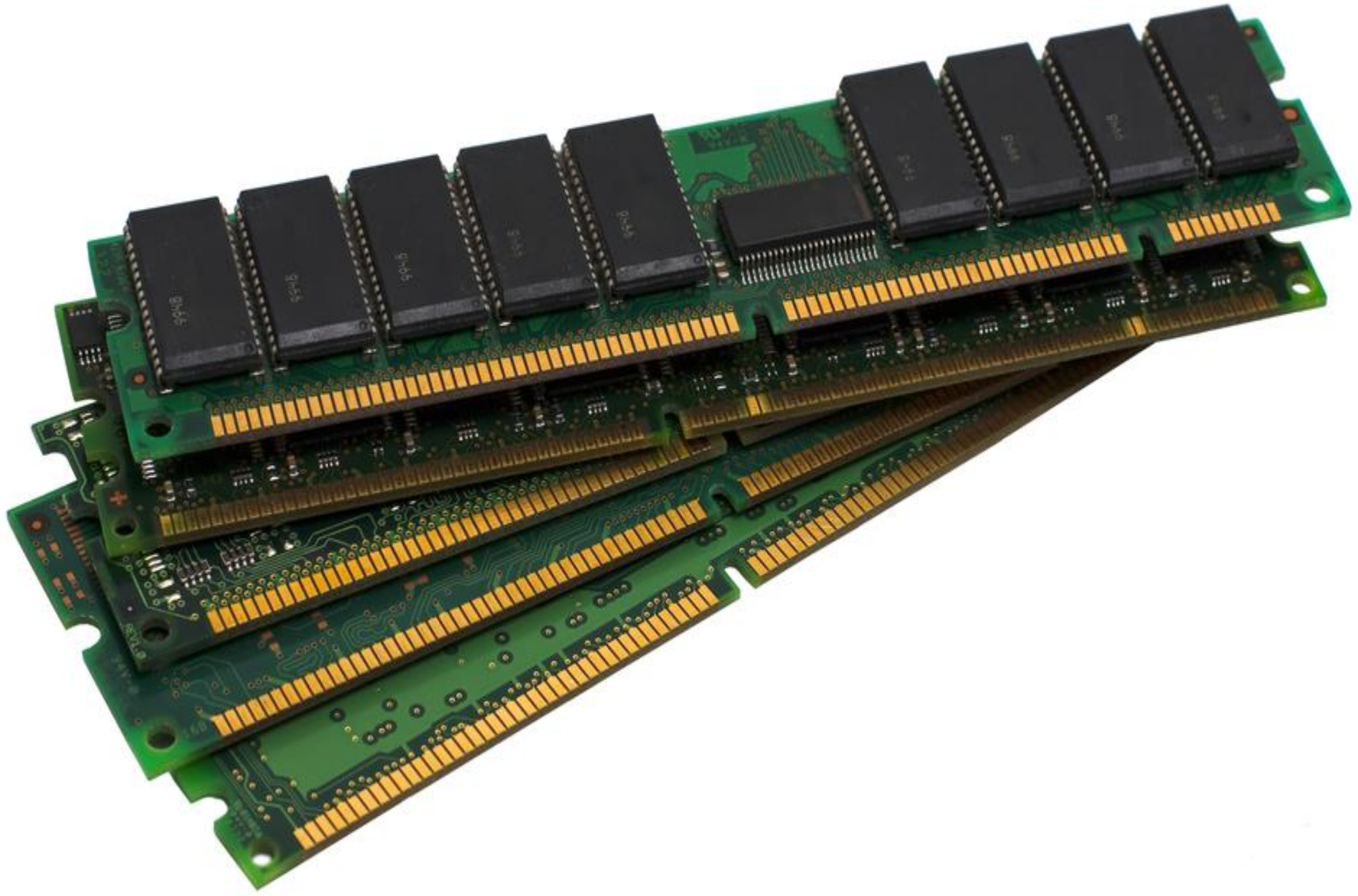
Move Paul's **Thursday** Zoom office hours to **Tuesday**, this week only.

No class next Monday.

A panel talk on DEI in STEM fields, which includes a few CS profs

- Thurs Feb 13, 11:30am-1pm
- RSVP here

# CS221
# C and Systems Programming



# REVIEW: Computer Memory

| Address | Value |
|---|---|
| 0x1000 0000 | 0x05 |
| 0x1000 0001 | 0xfa |
| 0x1000 0002 | 0x00 |
| 0x1000 0003 | 0x1d |
| 0x1000 0004 | 0x44 |
| 0x1000 0005 | 0x8e |
| 0x1000 0006 | 0x00 |
| … | |

Each bit of physical memory location consists of 1-3 transistors

Each byte has address and 8-bit value

We interpret addresses to be variable names (or method names)

We interpret values to be values of chars, ints, doubles, strings, CPU instructions, etc

## Organization of Computer Memory

Some memory reserved for hardware use

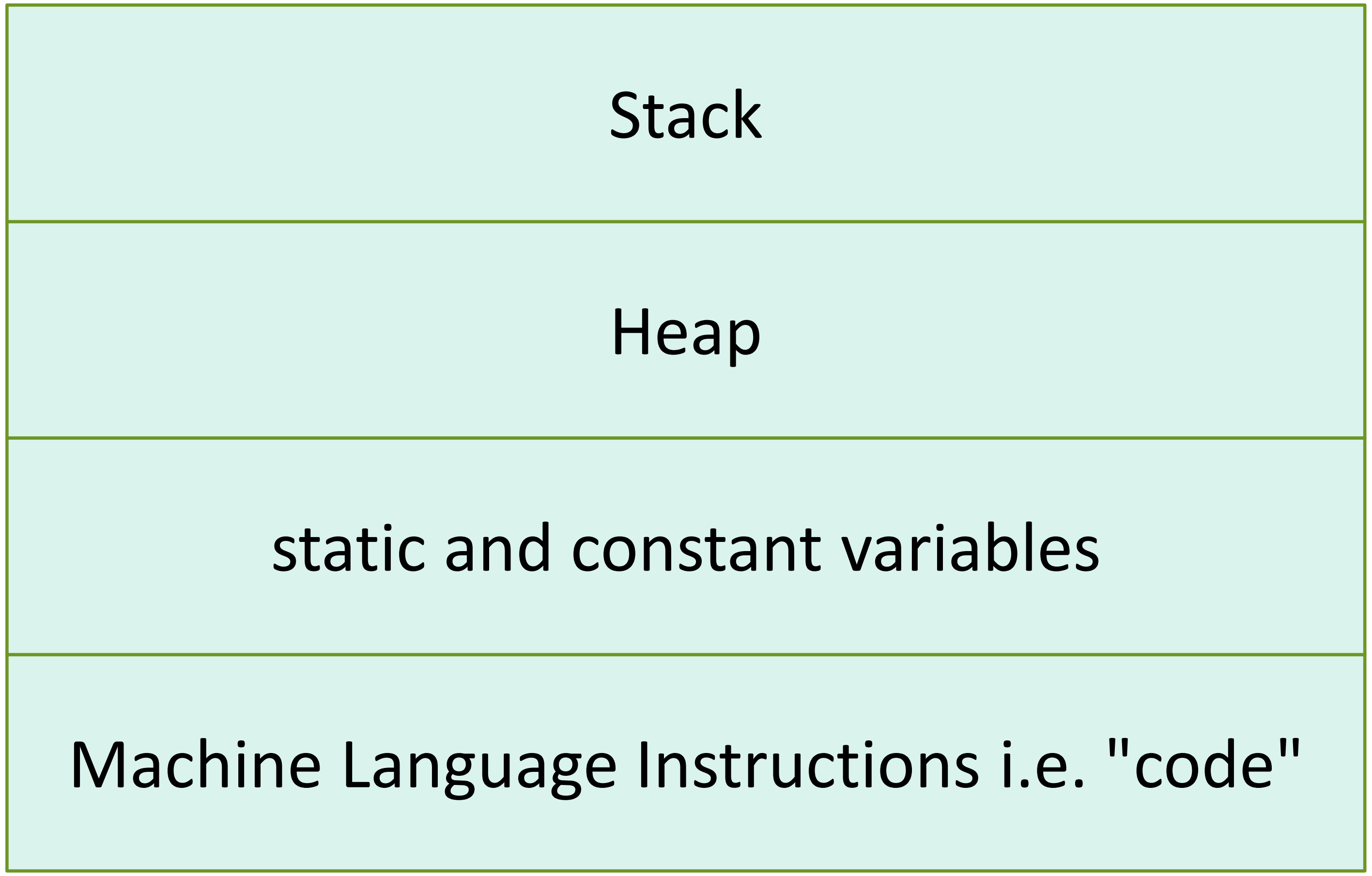
- Wi-fi radio, USB devices, etc

Some memory reserved for use by Operating System

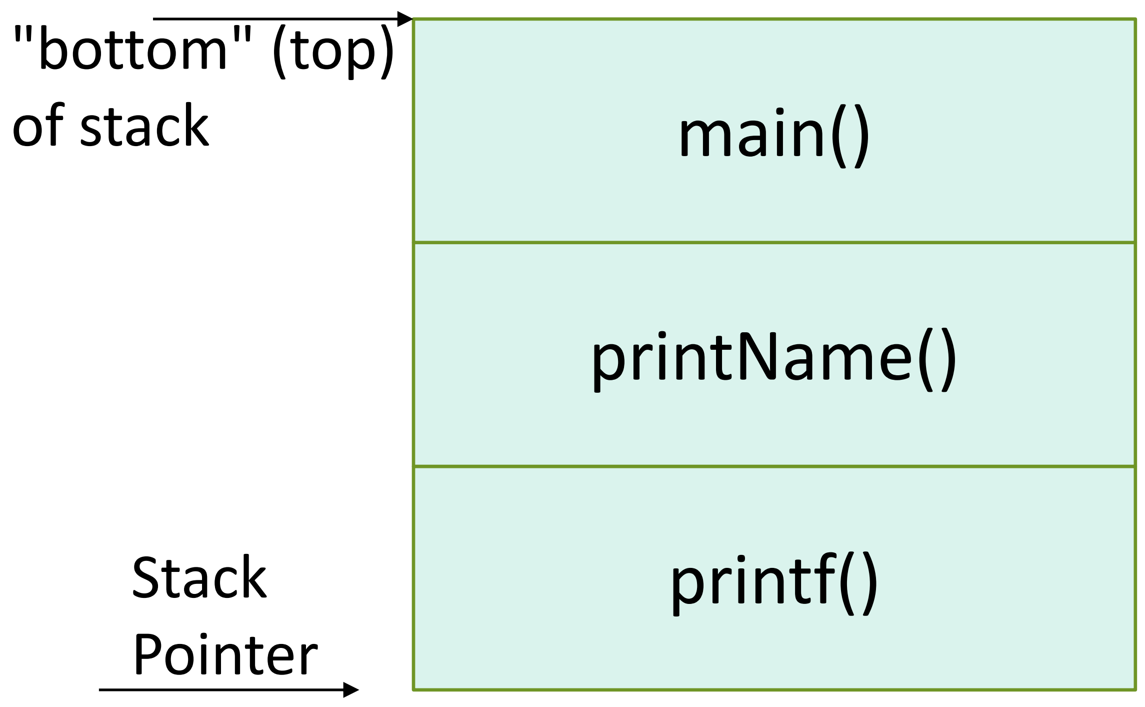
- CS 326 !

Each running <u>program</u> has its own memory region, assigned by Operating System

# REVIEW: Program Memory Organization


Stack
Heap
static and constant variables
Machine Language Instructions i.e. "code"

# REVIEW: Stack



Each running method has a "stack frame"

The Stack Frame stores:

- Return value
- Local variables
- Input variables

To call a method:

- "Push" input vars to stack memory
- Update Stack Pointer
- Jump to method's code

Return from a method

- Update Stack Pointer…
- …leaving Return Value in stack memory

## string.h

## String manipulation functions

```
#include <string.h>


strcat(), strcpy(), strcmp(), strchr(), strrchr()
```

## New C syntax - pointers

## Pointer variables

```
short sVar1 = 1, sVar2 = 2;
short* sPointer = &sVar1;
sPointer = &sVar2;
*sPointer = 333;
printf("%hd\n", sVar2);
```

What is **type** of `sPointer`?

What is **value** of `sPointer`?

Another tool the Java designers decided was too dangerous

## Some details

Often use `#define NULL (0)`

- Address 0 belongs to the OS. If a program tries to dereference it, causes a crash

<u>Value of an array variable</u> is <u>pointer to the first element</u>.

```
short ss;
short* sPtr = &ss;
char* cPtr = sPtr; // ???
```

# C Pointers vs Java References

C

- Any variable of any type can have a pointer to it…
- …including pointers
    - `int ii; int* pi = &ii; int** ppi = &pi;`
- Can pass pointer-to-variable to a method, and method can change underlying value

Java

- Built-in types are only passed by value
- Object types are only passed by reference
- Strings and Wrappers have (weird) immutability

## Keyboard Input in C

`scanf()`

- Format codes like `printf()`
- Give pointers-to-variables as arguments, so `scanf()` can change the value

## REVIEW: file input and output redirection

On the command line, we can replace a program's keyboard input with the contents of a file

```
myProgram < pretendKeyboardInput.txt
```

We can "redirect" a program's terminal output to a file

```
prog2 > ouputSavedHere.txt
```

## Dynamic memory allocation

## Wouldn't it be nice if…

…we had some way to call `new` to create new objects and variables?

C has `malloc()`

- We must give size of object.  Use `sizeof()` method
- Objects are created on the heap.  They stay there until we remove them with `free()`
- No automatic garbage collection!
- `#include <stdlib.h>`

## More scary details

`sizeof()`

- `int* ptr = malloc(1000); sizeof(ptr);`
- `int ptr2[1000]; sizeof(ptr2);`

- `int* iPtr; sizeof(iPtr);`
- `double* dPtr; sizeof(dPtr);`
- `char* cPtr; sizeof(cPtr);`

- `malloc() vs calloc()`

## Some examples

```
const int SIZE = getRandom();
int* intArray = malloc(SIZE * sizeof(int));
intArray[SIZE-1] = -1;

char* bigString = malloc(strlen(InputString));
strcpy(bigString, InputString);
```