Pointers

Lecture Overview

- Quick Review
- Lecture
  - Computer Overview (High Level)
  - Memory Layout (High Level)
  - Data storage
    - Variable Type vs Data Type
- Before We Code
  - Pointers
- Sample Programs

QUICK REVIEW

Variables and Equations

- Some commonly used properties (not all):

Addition Property: if a=b, then a+c=b+c 3x=2y -> 3x+7=2y+7

Subtraction Property: if a=b, then a-c=b-c 3x=2y -> 3x-5=2y-5

Distributive property: if $a(b + c) = ab + ac$ 3(x+2)=3x+6 *(6=3*2)*

Division Property: If $a = b$, then $\frac{a}{c} = \frac{b}{c}$ 3x=2y -> 3x/4=2y/4

(You can look up a full list)

Variables and Equations

Let's now take a look at the following real world scenario:

Jane spent $42 for shoes. This was $14 less than twice what she spent for a blouse. How much was the blouse?

Using the variable b, let's convert this real world scenario into a mathematical equation (basically looking for quantities and operations on these quantities):

We will set our equation equal to 42 since the price of shoes equals everything else

less implies a subtraction operation

Jane spent **$42 for shoes**. This was **$14 less** than **twice what she spent for a blouse**. How much was the blouse?

Twice what she spent for b (our unknown blouse price) implies a multiplication operation on b

Variables and Equations

Finally:

Zaza spent $70 for shoes. This was $10 less than three times what she spent for a blouse. How much was the blouse?

- Now we have to handle 3x as much, no longer 2x as much. Let's create a variable called t:

s=tb-l

We now have a general equation.

By generalizing the whole equation (and using variables), we can now have a computer program that can handle **any** of the preceding situations. We simply assign values to the variables accordingly.

Our program is not limited.

Looking Up Info

- You may have some assignments where you will have to look up information
  - I will **NOT** specify to you that you need to look something up
  - I will of course **always** help with instructions and concepts for the assignments themselves
    - Help breaking up a problem, for example
  - Future clients will not always explicitly tell you what something is

LECTURE

-**Computer Overview (High Level)**
-Memory Layout (High Level)
-Computer storage
    Variable Type vs Data Type

Computer Overview

- C allows the programmer to actively manage memory (in Java this was more behind the scenes unless you wanted to)
  - Historically, this type of management was crucial since computers were limited in size
  - In this class, since our computers are large in size, we won't feel the full power of C-it will be more practice
  - If you get into things like embedded systems, you will get to see more the importance of C features

Computer Overview

- Inside of a computer (in class we will only watch 2:00-3:15, but watch the whole thing on your own if you would like):

- https://www.youtube.com/watch?v=yRmPTbGBqVI *(Youtube: How To Identify The Components Inside Your Computer)*

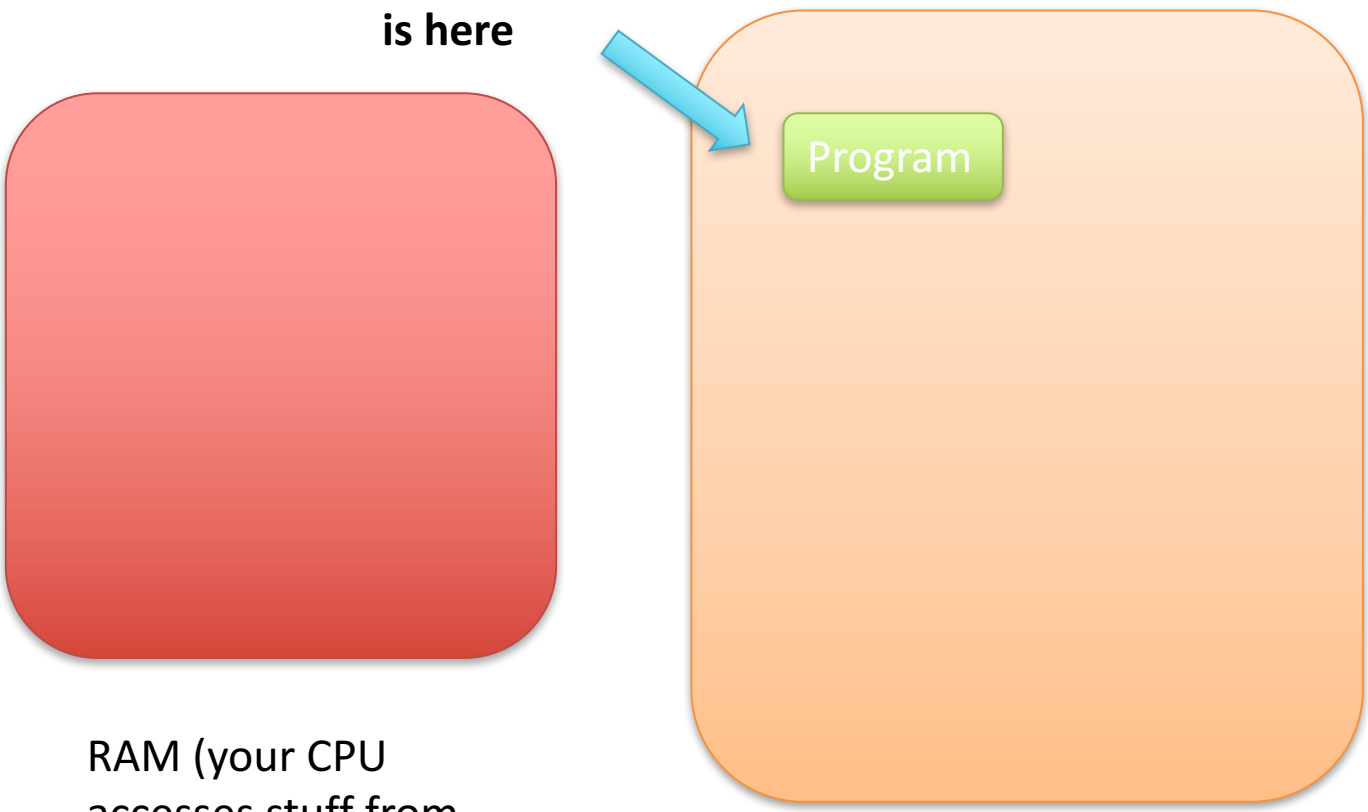
Computer Overview

- Main points to understand (high level viewpoint):
  - Your programs are executed by the CPU (after becoming machine language of course)
  - RAM (memory) vs Hard Drive
    - When our program is being run, it is in the RAM
    - When we talk about memory, we're talking about RAM (volatile storage)
      - Hard drive (non-volatile storage) is long term storage and usually much larger
  - Processor (CPU) speed
    - The faster your processor is, the more instructions it can execute

Computer Overview

(This is a VERY high level explanation- many important details are not being discussed)

1. Your program is here



CPU

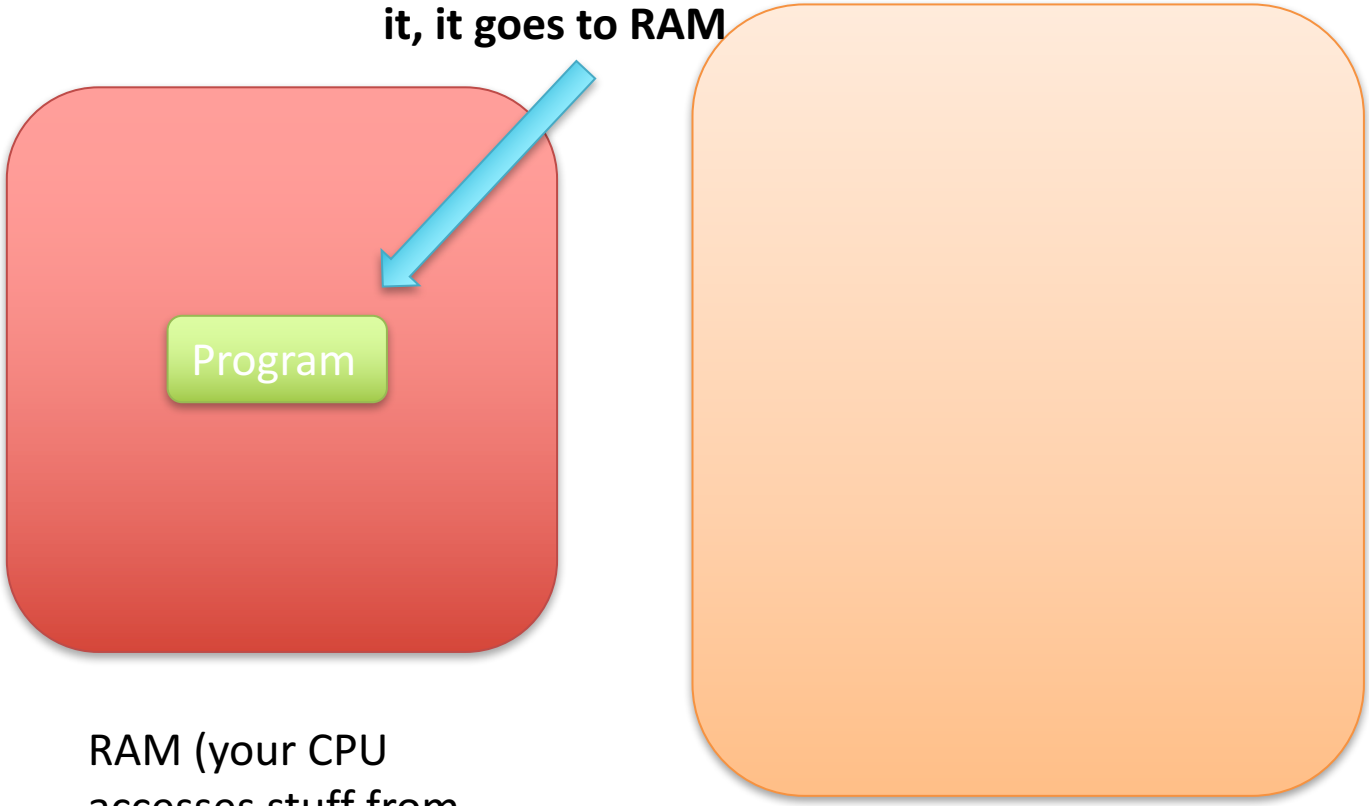
RAM (your CPU accesses stuff from here-including your program)

Long term storage (most people call this the hard drive)

Computer Overview

(This is a VERY high level explanation- many important details are not being discussed)

2. When you run it, it goes to RAM



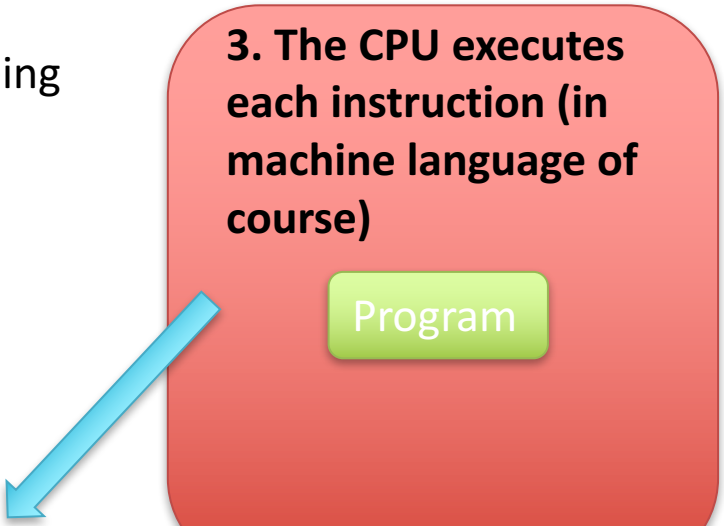
CPU

RAM (your CPU accesses stuff from here-including your program)

Long term storage (most people call this the hard drive)

Computer Overview

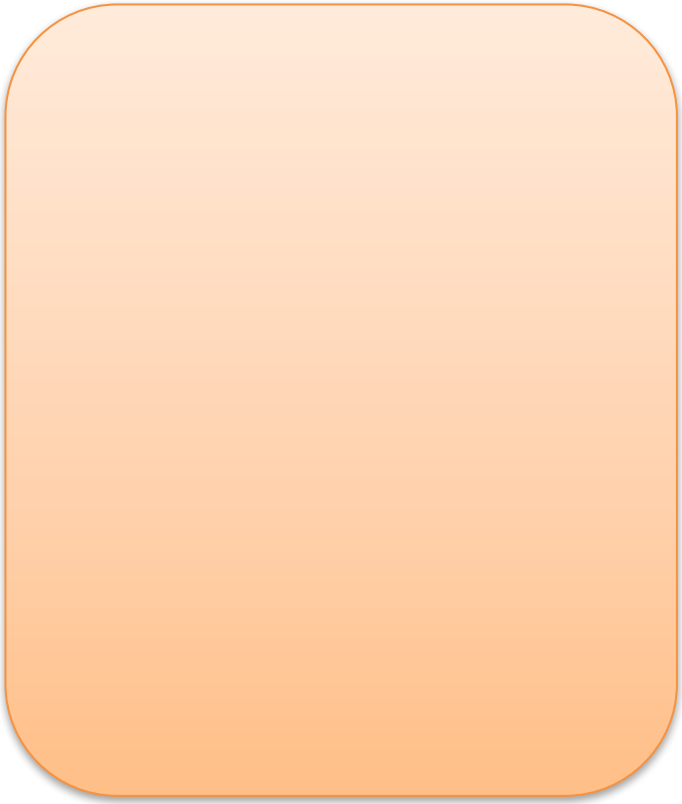
(This is a VERY high level explanation- many important details are not being discussed)



CPU

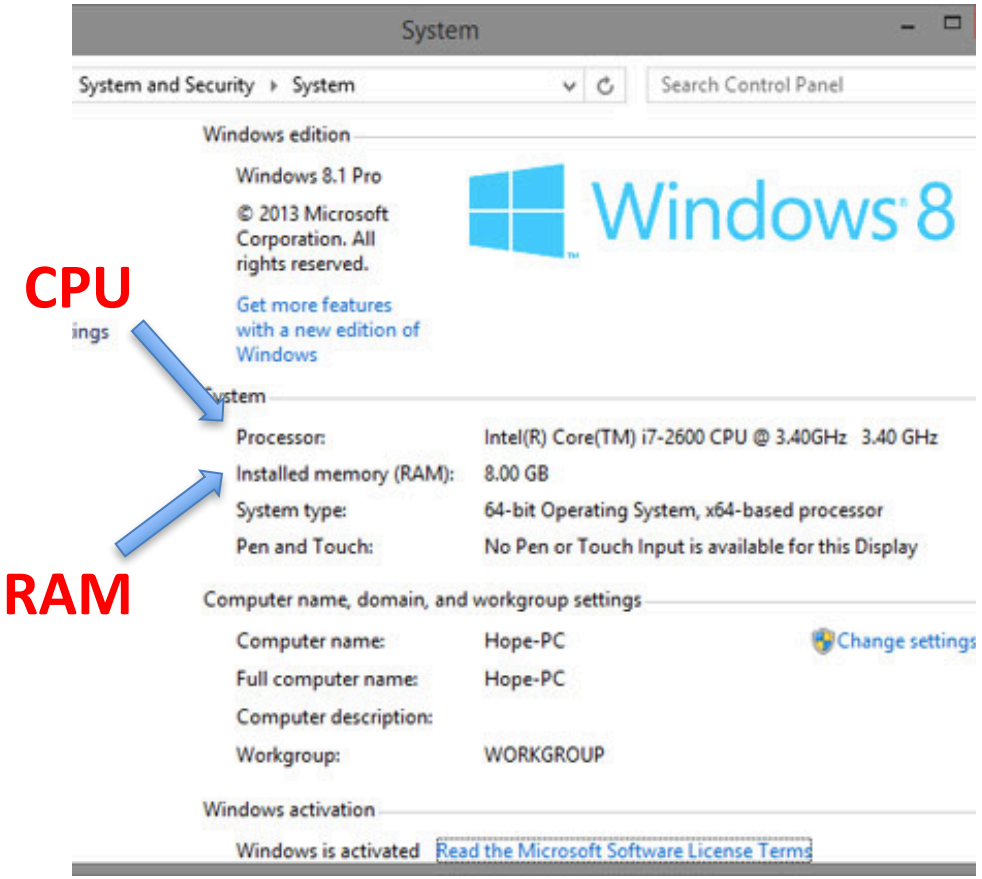
The faster the CPU is, the more instructions in a given point of time it can execute

RAM (your CPU accesses stuff from here-including your program)

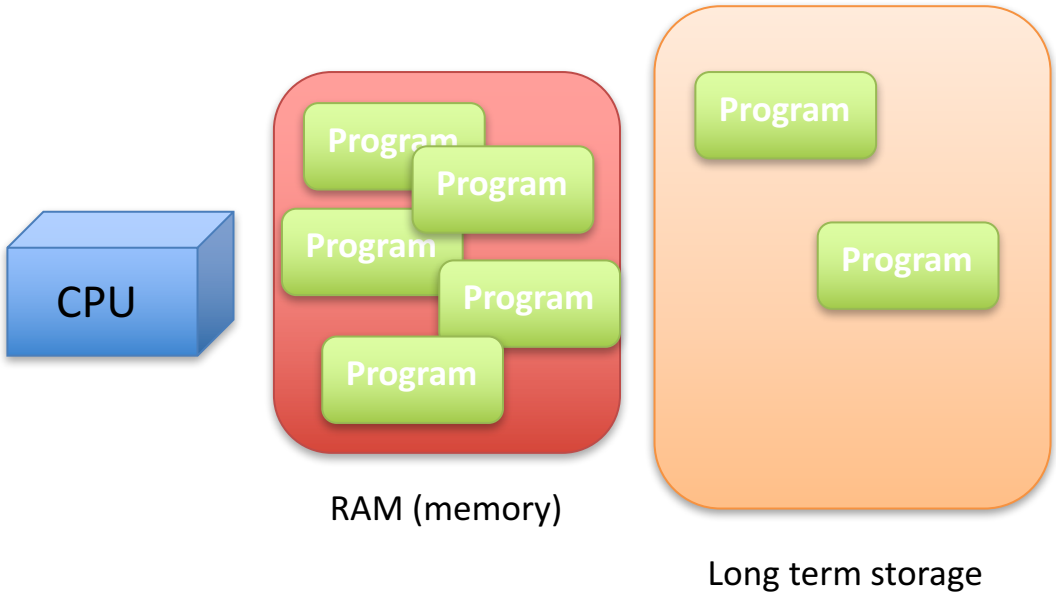


Long term storage (most people call this the hard drive)

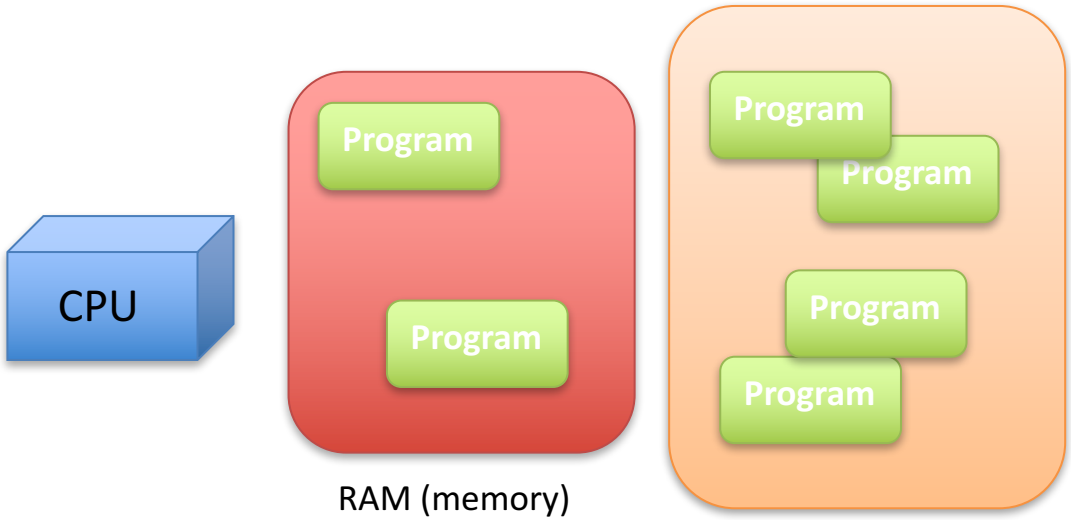
Computer Overview



When you have many things open and running on your computer it looks like this:



Otherwise, it looks more like this:



-Computer Overview (High Level)

-Memory Layout (High Level)

-Computer storage

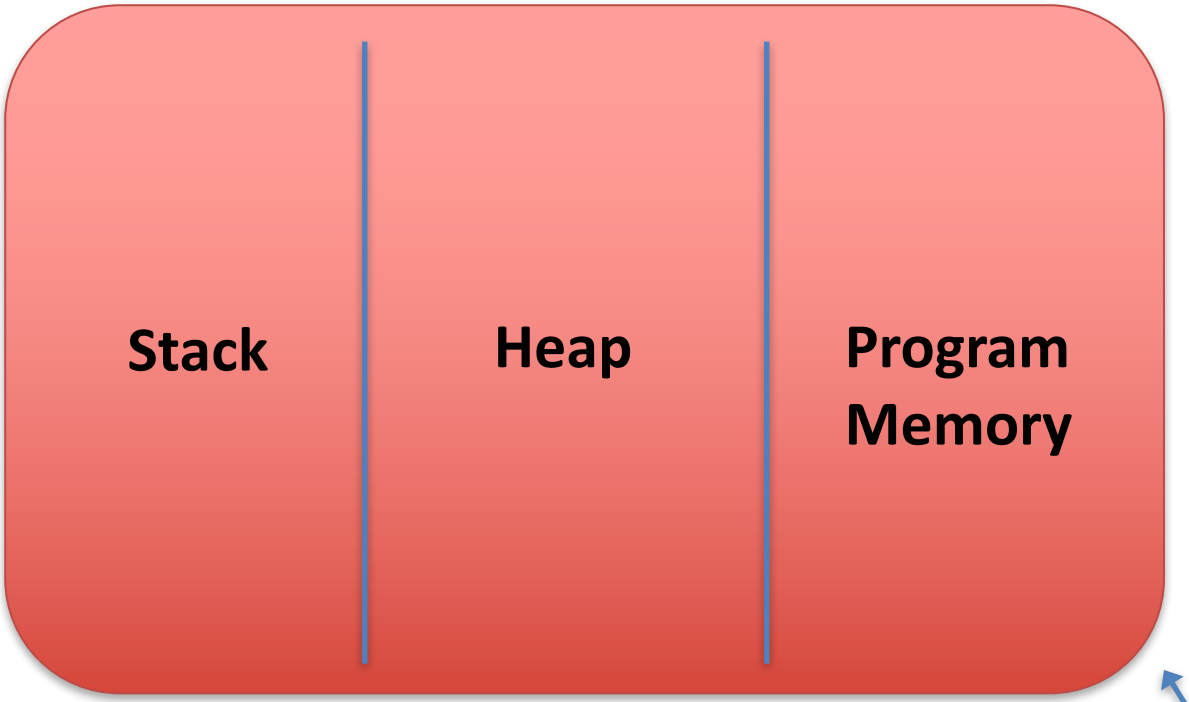
Variable Type vs Data Type

Memory Layout

- Memory is usually divided into three types:
  - Stack memory
    - Holds data
  - Heap memory
    - Holds data
  - Program memory
    - Holds machine code of computer programs-remember our programs need to be “translated” down to machine language for the computer to understand it

Memory Layout

(High level concept)



Holds data

Holds machine code of computer programs

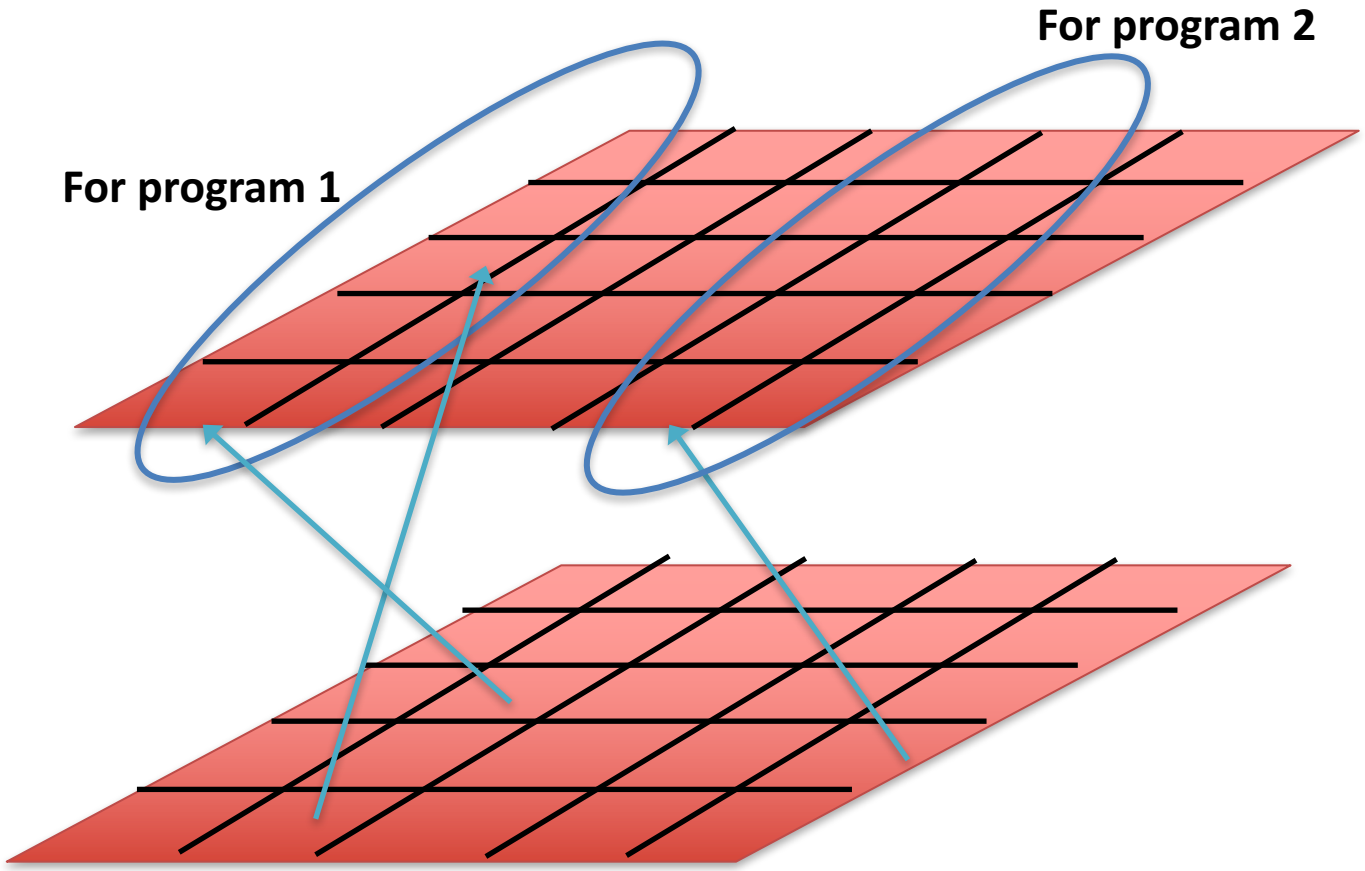
Memory Layout

- We will be discussing both heap and stack memory this semester
  - Very important concepts when using a language like C
  - Active memory management by the programmer in C is one of things that make it such a powerful language
    - In Java, memory management was more “behind the scenes”

Memory Layout

- One final concept to understand is that the addresses we will be seeing in our programs will be virtual addresses, not physical addresses
  - Every time we run a program, a certain part of memory is set aside for our program
    - We address our memory so we can access it
  - To us and our program it looks like one giant contiguous block of memory
    - Think of it like an array for you to do your work for now
    - Addresses can look sequential to us
- We could of course see physical addresses if we wanted, but that is not a concept we will cover in this class
  - We won't be actually going that low into the computer

Memory Layout



What we are dealing with in our programs (virtual address space)
We can view it in our C programs

Physical address space (we're not going this low into the computer in this class)-the ACTUAL physical computer

https://www.tutorialspoint.com/operating_system/os_memory_management.htm

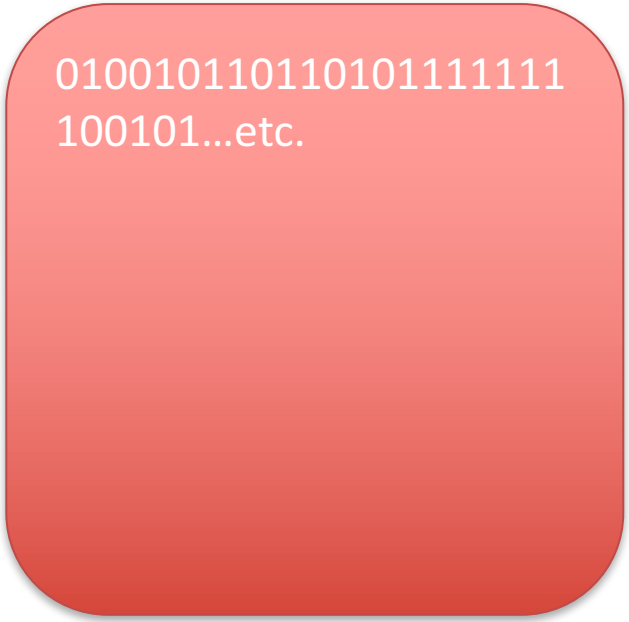
-Computer Overview (High Level)
-Memory Layout (High Level)
-**Computer storage**
    Variable type vs Data Type

Computer Storage

- Everything on your computer can be reduced down to a 0 and 1
  - That is the basic storage for a computer
  - All information in your computer is somehow encoded into 0s and 1s
  - Memory (and the hard drive in your computer) is just a bunch of 0s and 1s (bits)

These are bits. They can either have the value 0 or 1.

Computer Storage



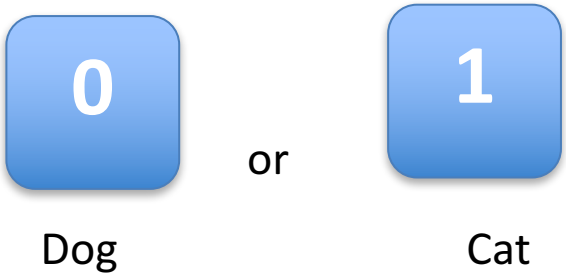
RAM (volatile memory)

Hard drive (non-volatile memory)

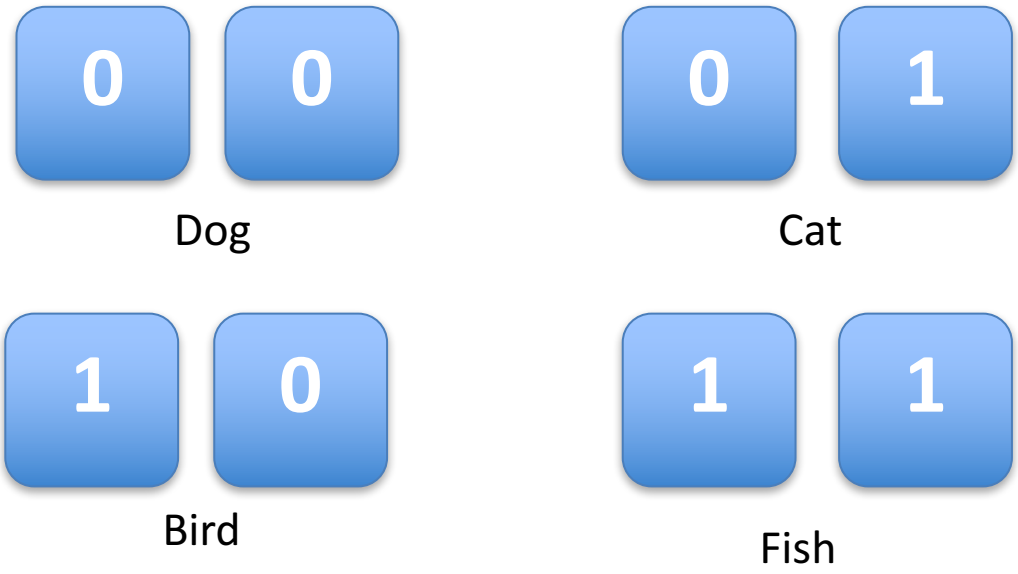
Computer Storage

- The basic unit of storage is a **bit**. It can hold either a 0 or 1.
  - You can remember this by **B**inary dig**IT**
- A standard size for **byte** is 8 bits. It would look something like 01101100
- A standard size for **word** is 32 bits (4 bytes) or 64 bits (8 bytes)
  - A CPU typically handles data/info in word sizes
    - You can say in a 32-bit computer that the CPU handles data in 32 bit chunks (so 2^{32} different combos)
    - You can say in a 64-bit computer that the CPU handles data in 64 bit chunks (so 2^{64} different combos)
  - Think about it-it would be pointless for a computer to only handle/ manipulate 1 or 2 bits of information

Computer Storage



A single bit can encode **2** things: 0 and 1. We can let this represent whatever we want.



Two bits can encode **4** different things

(remember this concept for later in the semester)

Computer Storage

A byte is a collection of 8 bits (standard)

A byte is 8 bits, so it can encode 2^8 (256) different combinations.
We can let each of these mean something different.

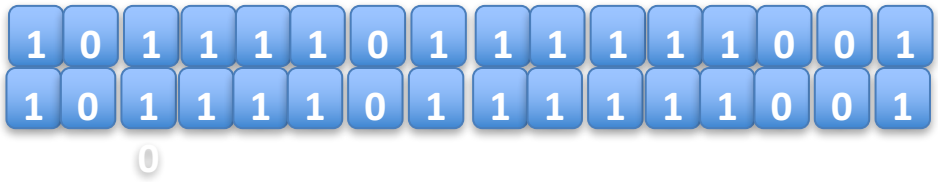
Note about the calculation:
The base (2) is number of possibilities and exponent (8) is number of spaces

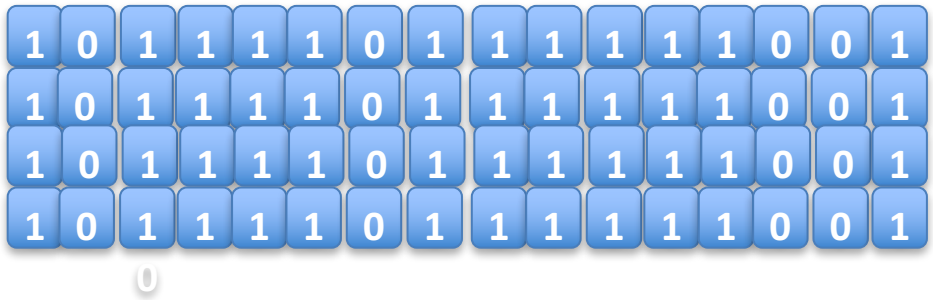
Number of possibilities: 2 (0 or 1) number of spaces 8 (8 bits) so 2^8

Example: If we had a ternary system instead of binary (so if we had 0, 1, 2) then we could code 3^8 (6,561) different combinations.

Computer Storage

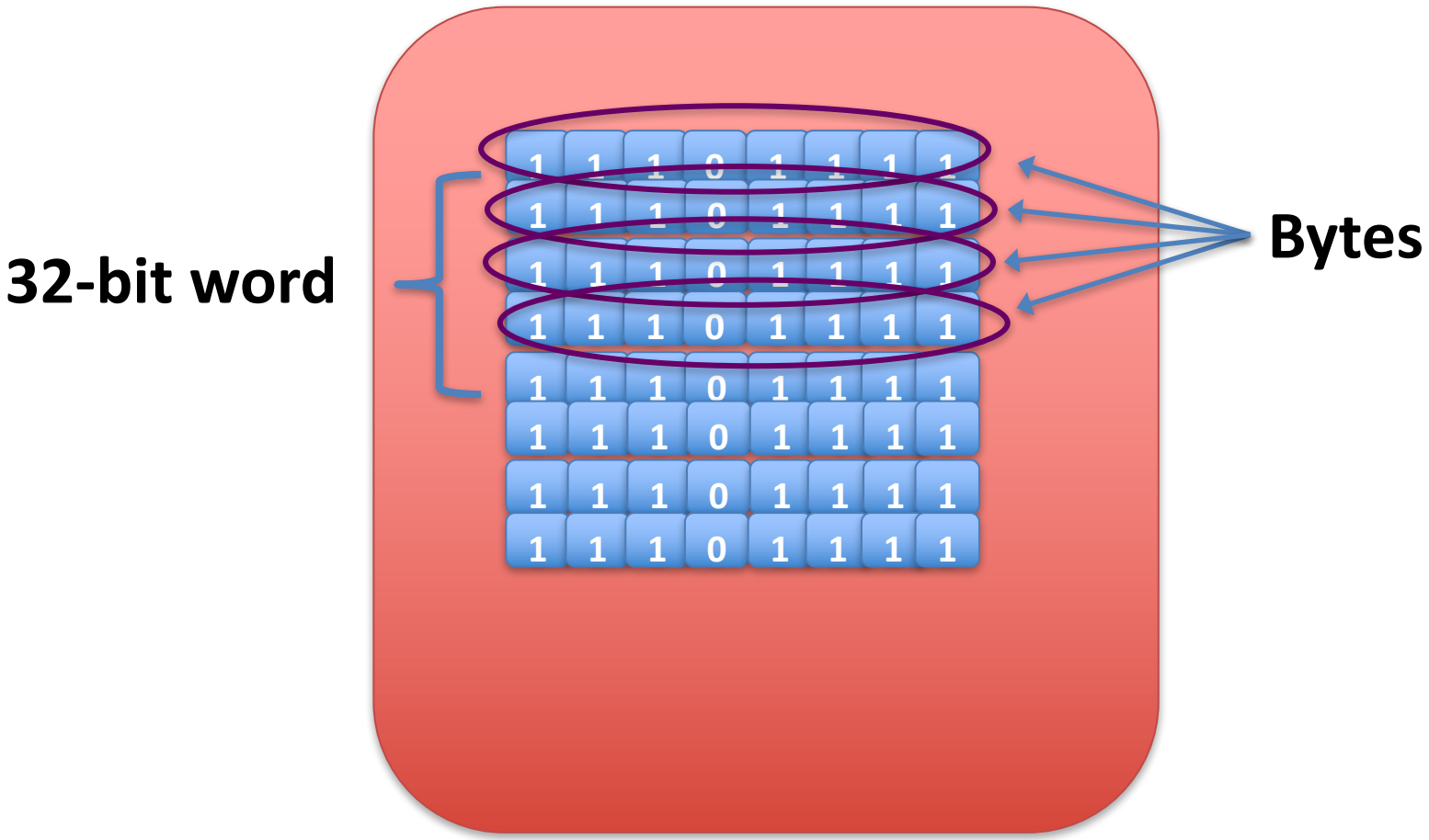
A byte is a collection of 8 bits (standard)





A word size can vary- usually you will see 32-bit (4 bytes) or 64-bit (8 bytes)

Computer Storage



Ram (volatile memory)

Variable Type vs Data Type

Use the letter f in my program

char c='f'; 

-Variable type is a character
-C has a built in data type to handle characters.
It is called char and the size is 1 byte.

Variable type: type of info being stored-is it a whole number? A character? This is more of a concept.

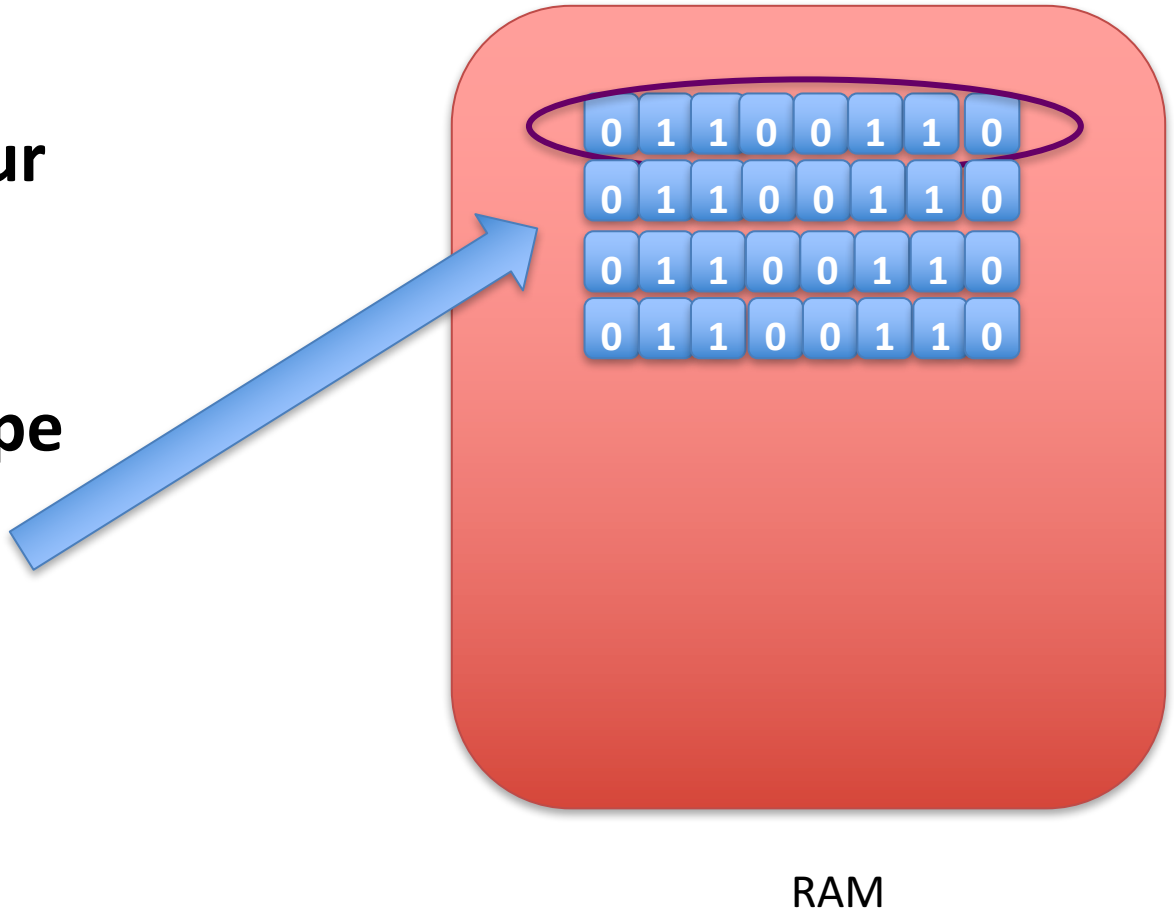
Data type: the actual physical "container" of the variable in the computer, measured in the storage units (bits, bytes etc)

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

Variable Type vs Data Type

1. **We want to use the letter f in our program**

2. **Use the data type char (1 byte): char c='f';**



Variable Type vs Data Type

- Since the char data type is a byte, that means we can hold 2^8 (256) different combinations
- We can see the different combinations in the ASCII Table (next slide)
  - **Note:** you will see that we only have 128 different characters represented, so 7 bits would have been sufficient (2^7). We have an "extra bit" for our storage. This can account for growth (more symbols) and the fact that 7 bits is not a standard size.
  - **Note 2:** There is something called the extended ASCII table. We won't address that in this class (total symbols go up to 256)

Variable Type vs Data Type

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

The char *f* is represented by 102.

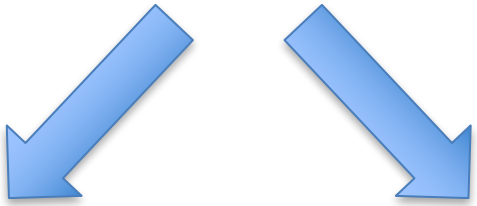
In binary, this is 1100110 (so 7 bits).

In our computer, we would have 01100110 (8 bits, 1 byte) to hold our char.

Variable Type vs Data Type

Different languages have different data types and built-in features to handle information

Handle binary option variables like:
Yes/no
True/false
Right/Wrong



Java has a boolean data type

C does not have a boolean data type (although later versions added in some support for booleans)

Variable Type vs Data Type

- A pointer is simply a data type to hold memory addresses
  - Which memory address are we actually talking about? That is the variable
  - The data type is the actual space used to hold the address
    - The size varies-usually 32 bit or 64 bit
      - We will talk in a future class about the importance of this size difference
    - An address is actually a number
      - Usually represented in base 16 (hexadecimal)

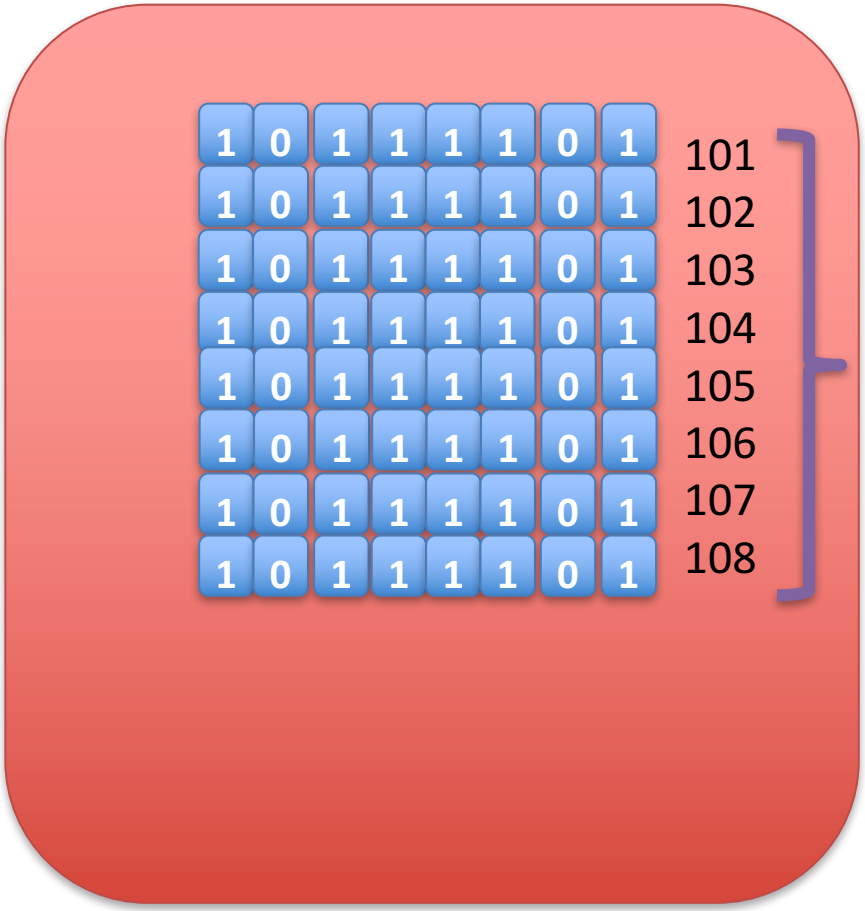
Variable Types vs Data Types

- One final point
  - People tend to use variable types and data types in the same way in general conversation
    - I do it also-you will see me do this often. Just remember variable is more of the concept (like in math) and data type is more about the actual space being used in the computer
  - The sizes of data types can depend on things like the machine and compiler you are using
  - We will use the sizeof() operator today in our code
    - We will talk about operators in the next class

Computer Storage

We can “name” chunks of our memory so we can access it directly. This is called addressing.

We usually view it as hexadecimal values.



Ram (volatile memory)

BEFORE WE CODE

Note

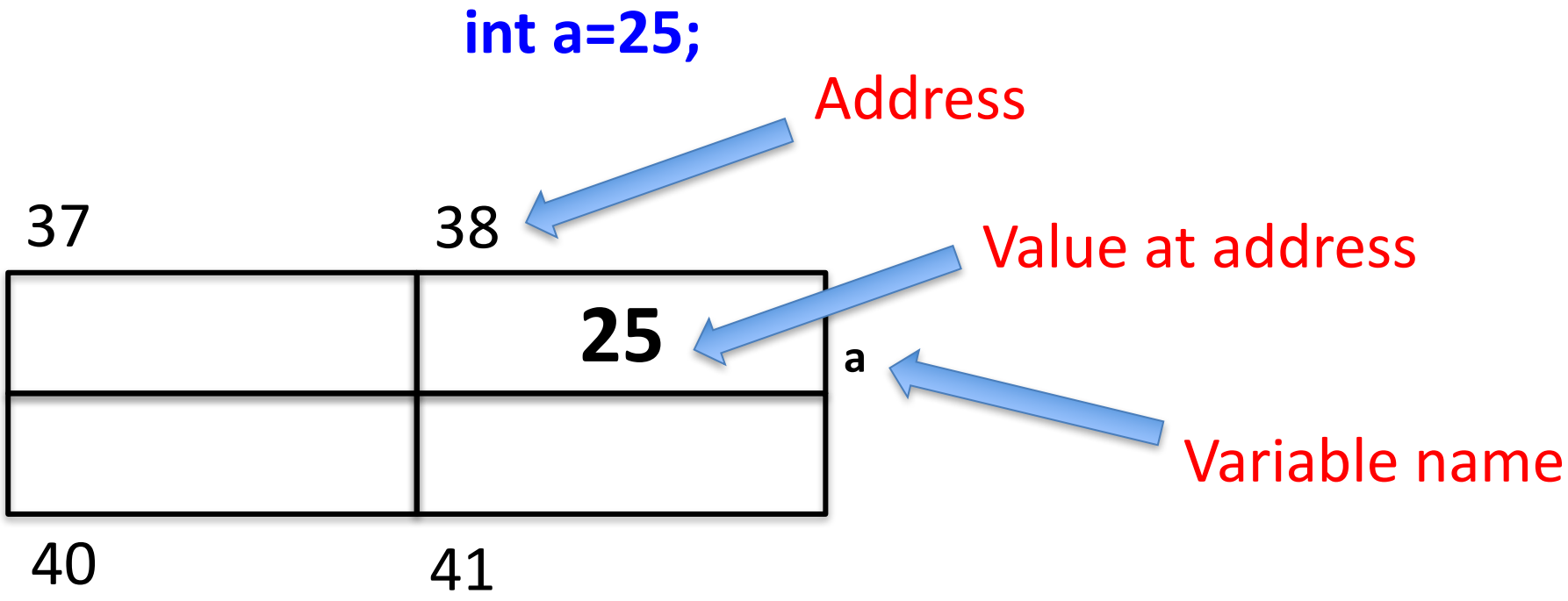
- Before we begin, I want to mention that many people learning C say that pointers are one of the most difficult things to learn
- My take is that pointers are not inherently difficult, but you must 100% understand how to use them or you can't successfully use them at all
  - You can't "kinda" get pointers
  - We will be using them for the rest of the semester, so don't think you can just avoid them
  - I draw a lot of pictures to help you visualize what's going on
    - I encourage you to do the same

Note

- Don’t worry if you don’t get them in a week or so
  - It takes lots of practice to fully understand how to use them
- In most cases, it is my opinion that successfully using C means successfully using pointers
  - I will do a small intro today and we will build up from there
  - Make sure to KEEP UP WITH ALL LECTURES!!!

Before We Code

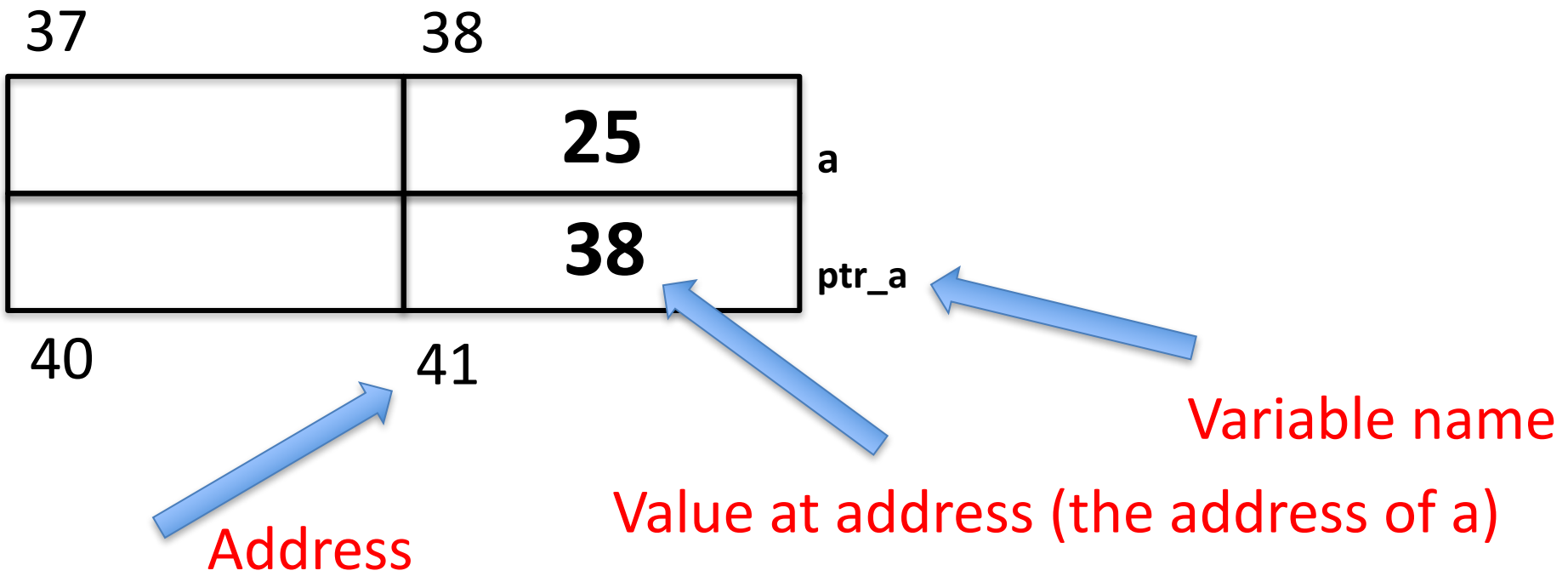
You can visualize pointers in a computer like the following:



Before We Code

You can visualize pointers in a computer like the following:

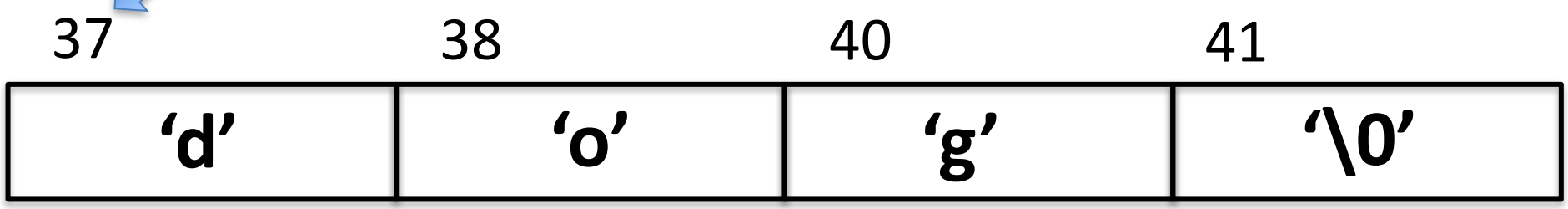
```
int a=25;
int* ptr_a=&a;
```



Before We Code

word (an array) is really represented by the address of the first char in the char array

char word[]="dog";



Before We Code

- Notice with char arrays we don't need & with scanf

```
int a;
scanf("%d", &a);
```

We are putting the value at the address of the variable a (& is the address operator)

```
char word [];
scanf("%s", word);
```

Since the array is really an address at the first element, this works (no &)

Before We Code

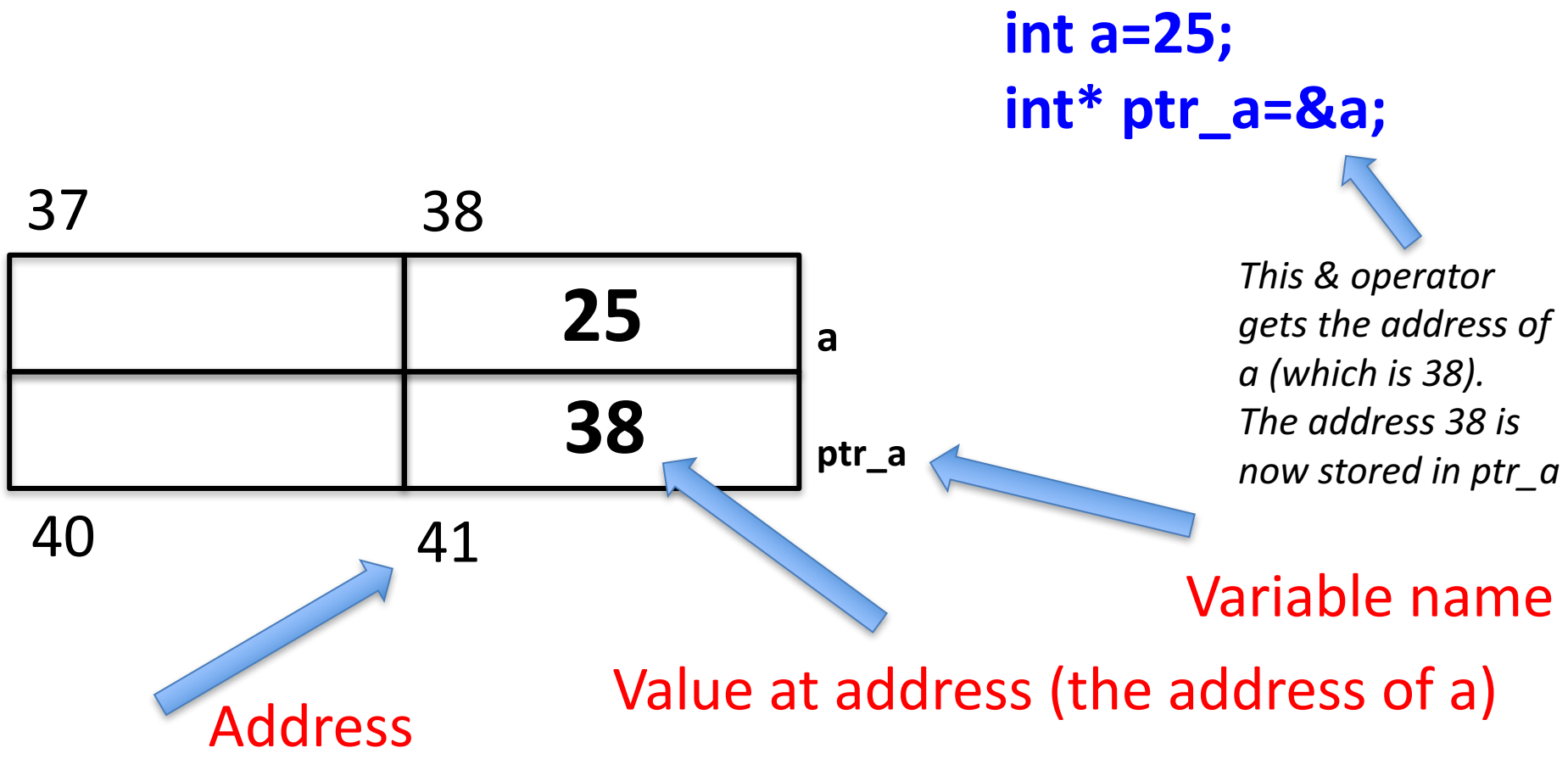
- Operators:
  - You learned about operators (like **&&** and **<** for example) in 1310
  - An operator is a special symbol that performs an operation (and can return us a value)
    - You see them in math (+ for example will return us the sum of two values)
- Two operators in C we will use today:
  - The address operator: **&**
  - The dereference operator: ***

Before We Code

- The address operator: **&**
  - This operator is applied on a variable and returns the address of that variable
    - If you apply this operator to the variable data type, you get the the address of that variable
- The dereference operator: *
  - This operator is applied on a pointer variable and returns the value of the variable's address stored in the pointer
    - If you apply this operator to a pointer variable, you get back the value at the address that is held

Before We Code

- The address operator: **&**

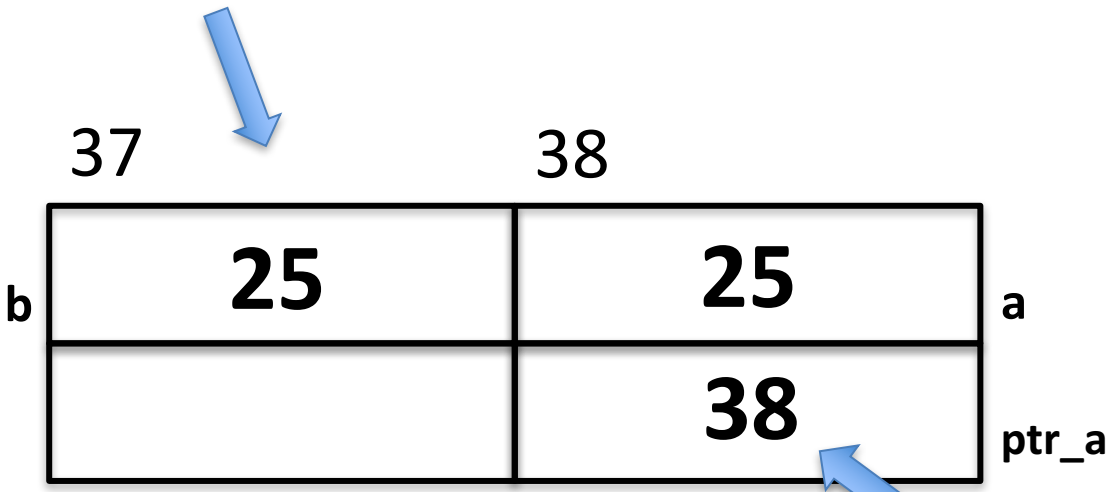


Before We Code

- The dereference operator: *

```
int a=25;
int* ptr_a=&a;
int b= *ptr_a;
```

The value stored at address in ptr_a is 25 (value of a)



*This * operator goes to whichever address is in ptr_a and gets the value stored at the address*

Variable name

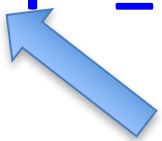
Value at address (the address of a)

Address

Before We Code

- DO NOT confuse the dereference operator * with the * used in declaring a pointer variable:

```
int a=25;
int *ptr_a=&a;
```



Here, this * is used to declare a pointer (more of a signal to us that this is a pointer variable)

VS

```
printf("%d", *ptr_a);
```

Here, this is a * deref operator (returning to us what is at the address in the pointer)

Before We Code

- https://www.tutorialspoint.com/cprogramming/c_operators.htm

SAMPLE PROGRAMS

Sample Programs

1. Pointer examples
   - Operators & and deref
2. Scanf using &
3. Pointers in functions
4. Sizeof
5. Sample Program