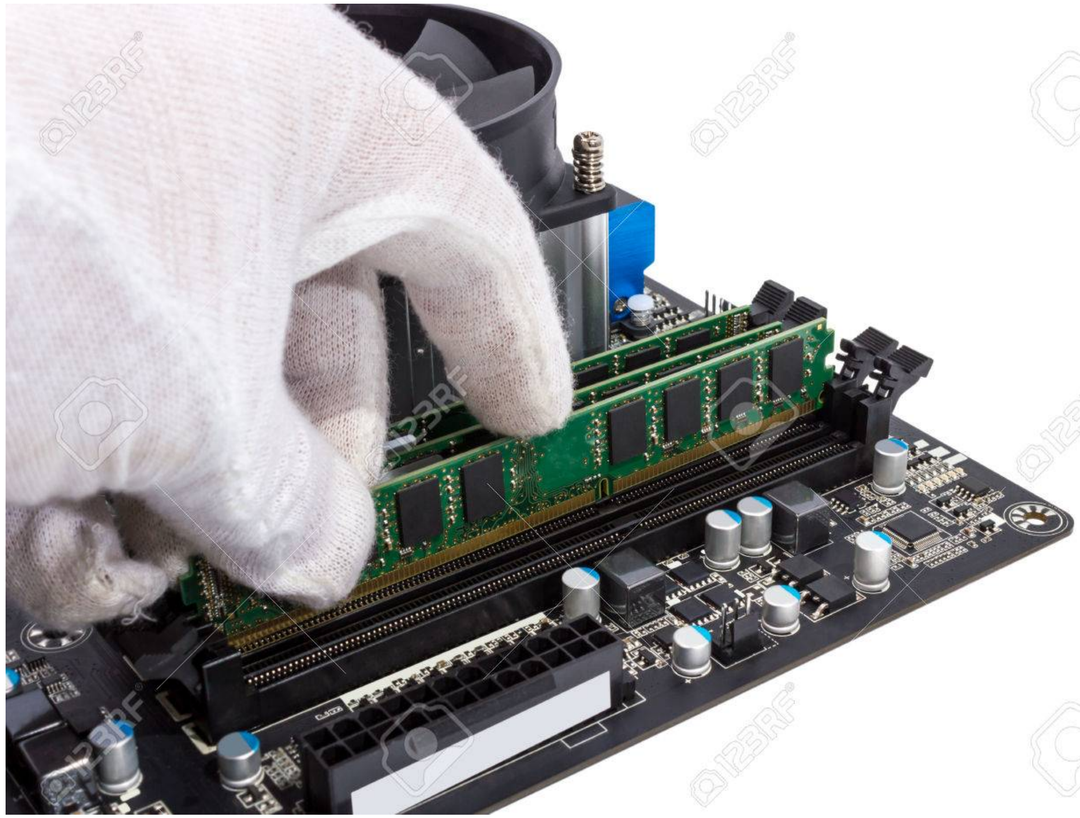
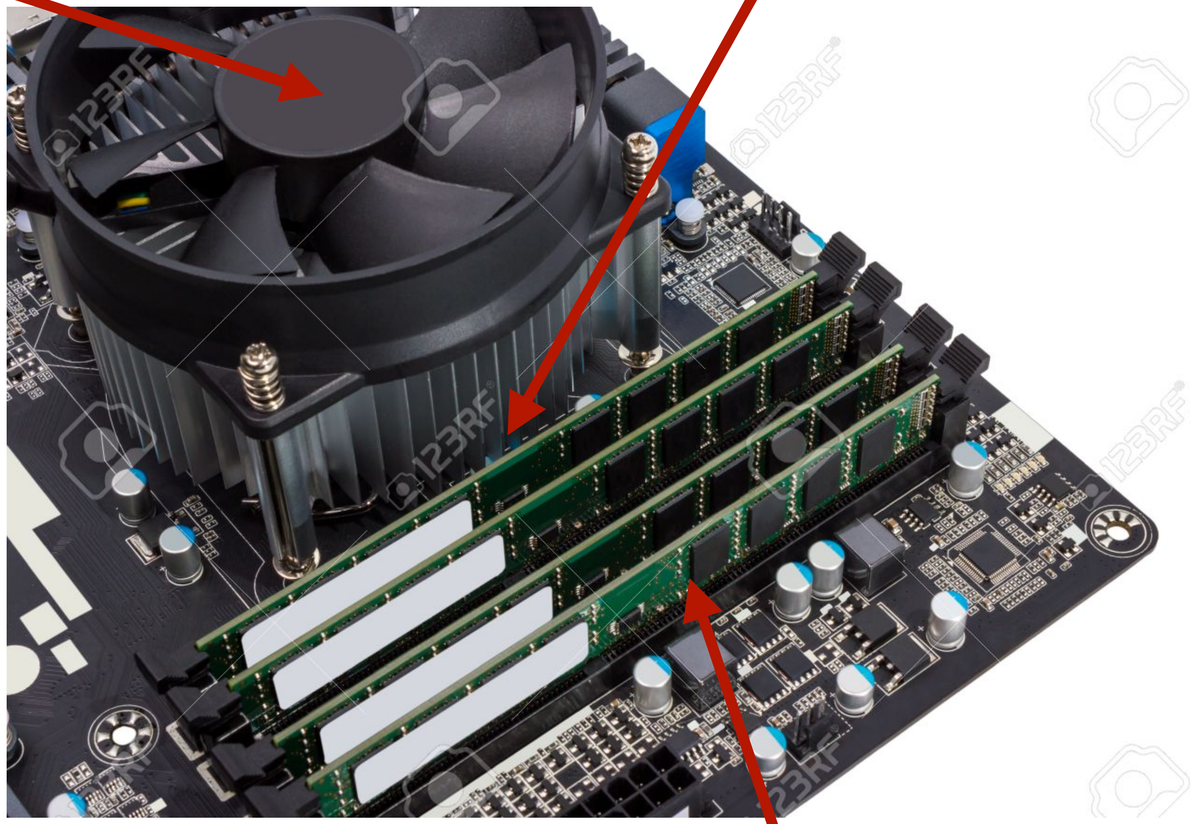
Memory & C Pointers

Yunhao Zhang
Cornell University

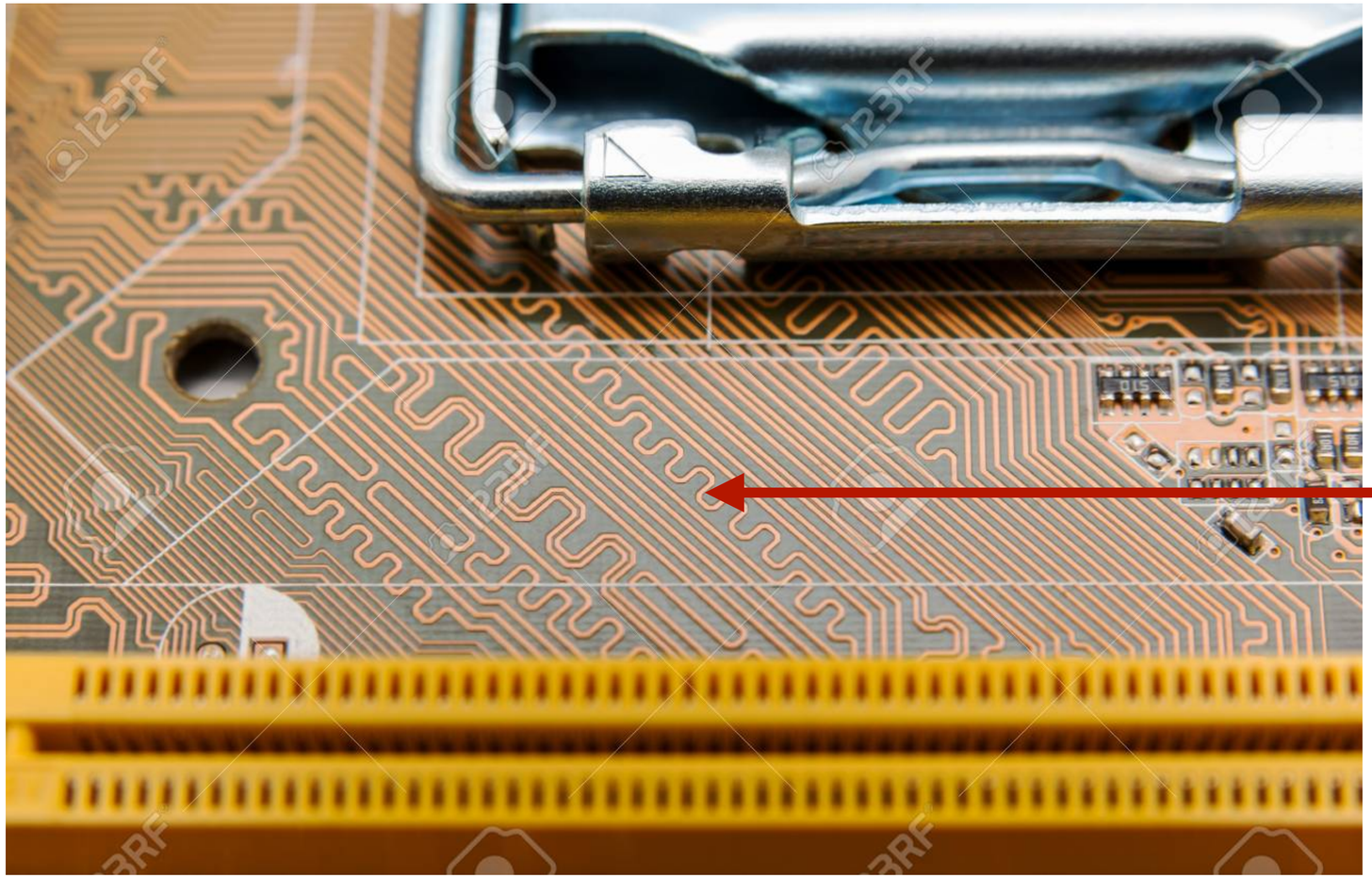
What is memory?





* Images from https://www.123rf.com/

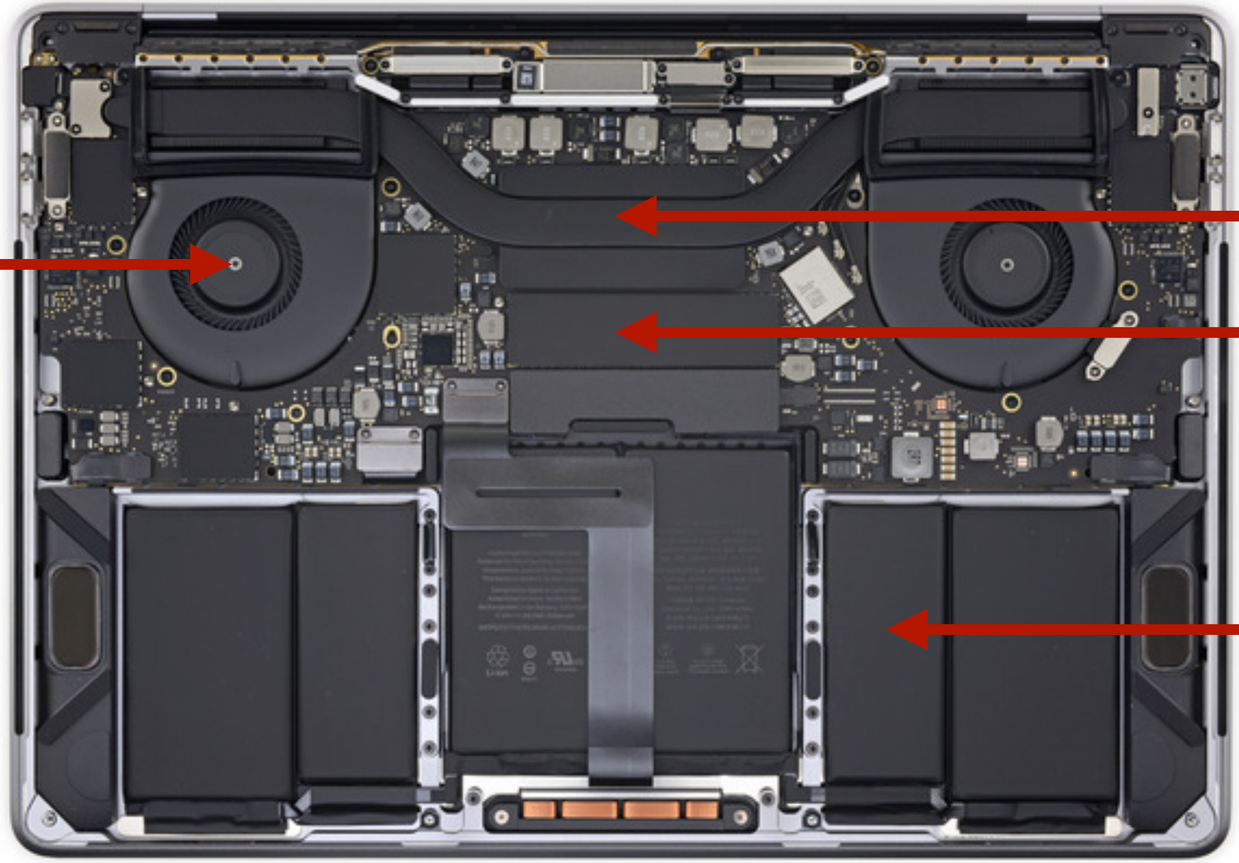
What is memory?



Circuits connecting
CPU and memory

* Images from https://www.123rf.com/

What is memory?



* Images from https://www.computerrepairsoftware.com/macbook-pro-2018-teardown-more-than-just-a-new-keyboard/

Memory size

Size	Size in 2^n	Address space
1KB	2^10 bytes	#0, #1, …, #2^10-1
2MB	2^21 bytes	#0, #1, …, #2^21-1
8GB	2^33 bytes	#0, #1, …, #2^33-1

- Memory contains bytes and each byte is 8 bits.
- 2^**10** is 1KB; 2^**20** is 1MB; 2^**30** is 1GB

OS controls CPU and memory

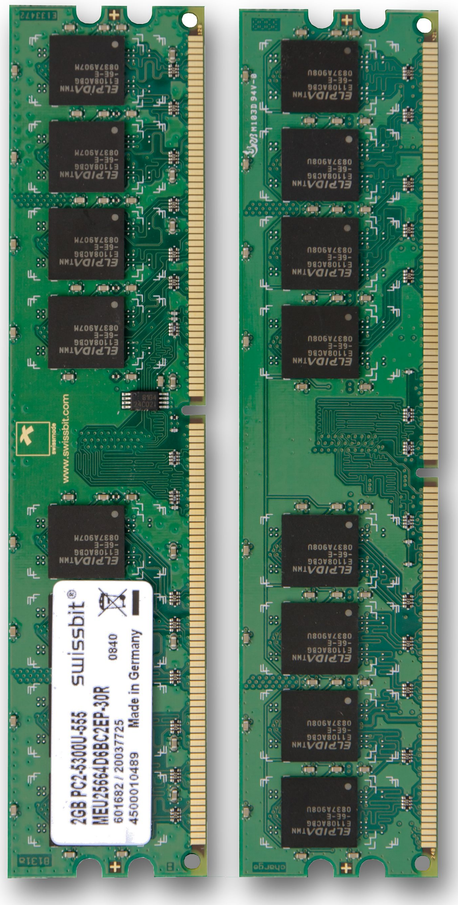
- We have seen how CPU and memory exist in the real-world.
- Those circuits are fun to see, but operating systems do NOT need to know the circuit details (CS3410 deals with that)!
- The power of **abstraction**:
  - represent memory with a simple math model.

Abstraction

Art in 19th century

Art in 20th century

Memory hardware



Abstract math model

2^n -1	8bits
...	
...	
#2	8bits
#1	8bits
#0	8bits

* Images from Wikipedia and Lorenzo's slides: https://www.cs.cornell.edu/courses/cs5414/2017fa/notes/week12.pdf

Memory address space

- **n** is usually 32 or 64 meaning that the first column of the math model requires 4 bytes or 8 bytes to represent.
- Example of modifying a single byte in memory:

```
// address is usually represented
// in hexadecimal
char* loc = (char*) 0x1234abcd;
*loc = 0x89;
// putting byte 0x89 to address 0x1234abcd
```

2^n -1	8bits
...	
...	
#2	8bits
#1	8bits
#0	8bits

Pointer

```
char* loc = (char*) 0x1234abcd;
*loc = 0x89;
// putting 0x89 to address 0x1234abcd
```

- We call `loc` a pointer.
- Compiler decides the position of `loc`.
- `loc` occupies 4 bytes of memory (i.e., n=32) and stores an address.

2^n -1	...
...	...
0x 1234abcd	0x 89
...	...
position of loc + 3	0x 12
position of loc + 2	0x 34
position of loc + 1	0x ab
position of loc	0x cd
...	...

Pointer and Types

Type	sizeof(Type)	Type	sizeof(Type) (n = 32)	sizeof(Type) (n = 64)
char	1	char*	4	8
int	4	int*	4	8
long long	8	long long*	4	8
float	4	float*	4	8
double	8	double*	4	8

Pointer and Array

```
char* loc = (char*) 0x1234abcd;
loc[0] = 0x89;
// same as *loc = 0x89
loc[1] = 0x12;
// same as *(loc + 1) = 0x12
loc[2] = 0xaa;
// same as *(loc + 2) = 0xaa
```

...	...
0x 1234abcf	0x aa
0x 1234abce	0x 12
0x 1234abcd	0x 89
...	...
position of loc + 3	0x 12
position of loc + 2	0x 34
position of loc + 1	0x ab
position of loc	0x cd
...	...

Operating system vs. User application

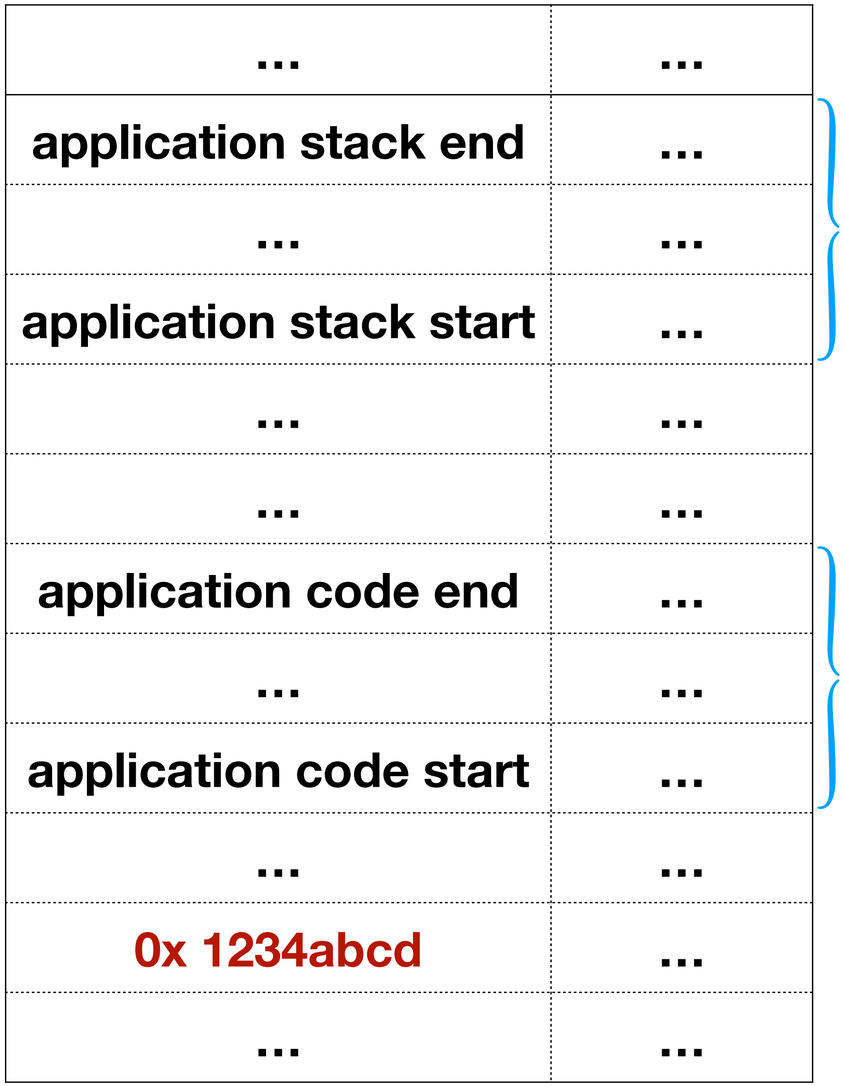
```
int main() {
        char* loc = (char*) 0x1234abcd;
        loc[0] = 0x89;
        loc[1] = 0x12;
        loc[2] = 0xaa;

        return 0;
}
```

- This function can work well as operating systems code.
- But it crashes if you write a user application like this.

Operating system vs. User application

- CPU has privileged mode and unprivileged mode (specified by a CPU internal register).
  - Operating systems run in the privileged mode.
  - User applications run in the unprivileged mode.
- In privileged mode, code is free to access all memory addresses.
- In unprivileged mode, code can only access memory addresses that operating systems have allowed.

OS controls Application memory access

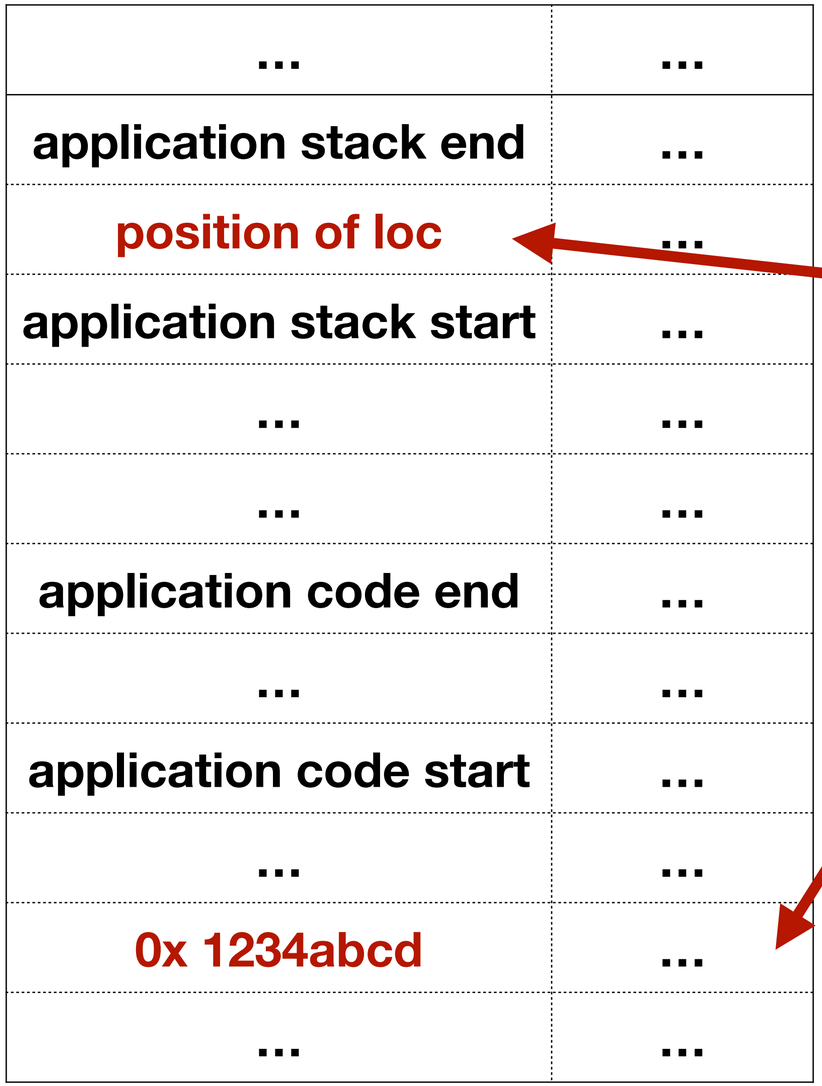


OS allows user applications to access this region holding local variables in functions.

OS allows user applications to access this region holding the binary executable code.

OS disallows user application to access!

Operating system vs. User application




```
int main() {
    char* loc = (char*) 0x1234abcd;
    loc[0] = 0x89;
    loc[1] = 0x12;
    loc[2] = 0xaa;

    return 0;
}
```

allowed!

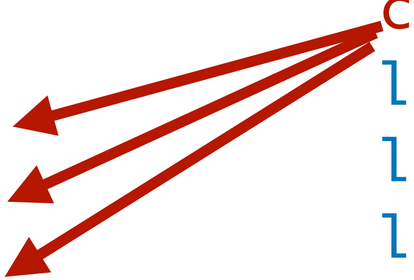
disallowed!

OS controls Application memory access

- We will discuss the control mechanisms later this semester.
- For now, the take-aways are simple
  - OS controls which memory regions in the address space that applications are allowed to access.
  - You used to implement `malloc` in CS3410. `malloc` is a mechanism that application can request access to a piece of memory dynamically from the operating system.

Request memory dynamically from OS

...	...
application stack end	...
...	...
application stack start	...
...	...
position of loc + 2	...
position of loc + 1	
position of loc	...
...	...
application code end	...
...	...
application code start	...
...	...



```
int main() {
        // char* loc = (char*) 0x1234abcd;
        char* loc = (char*) malloc(3);
        loc[0] = 0x89;
        loc[1] = 0x12;
        loc[2] = 0xaa;

        return 0;
}
```

The code now works!

Homework

- We will release the first project P0 today. P0 is due next Friday (Sep 11).
- Implement a queue data structure and the related operations
  - create/free a queue
  - append/dequeue elements to the queue
- Please read the instructions carefully before asking questions on Piazza.