

# Computer System Organization

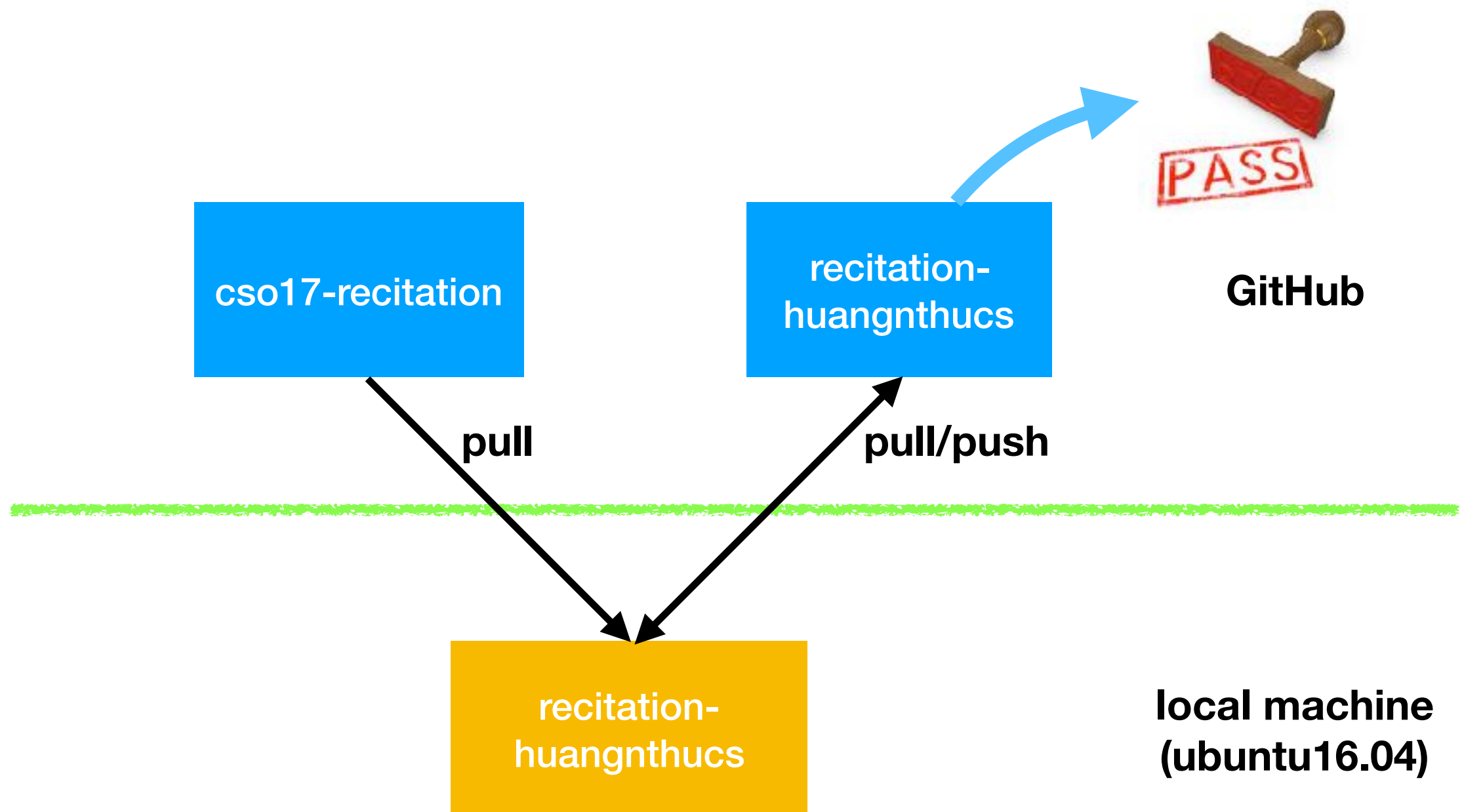
## Recitation

### [Fall 2017]

CSCI-UA 201-006

R3: pointer/array

# Git status for our recitation and labs

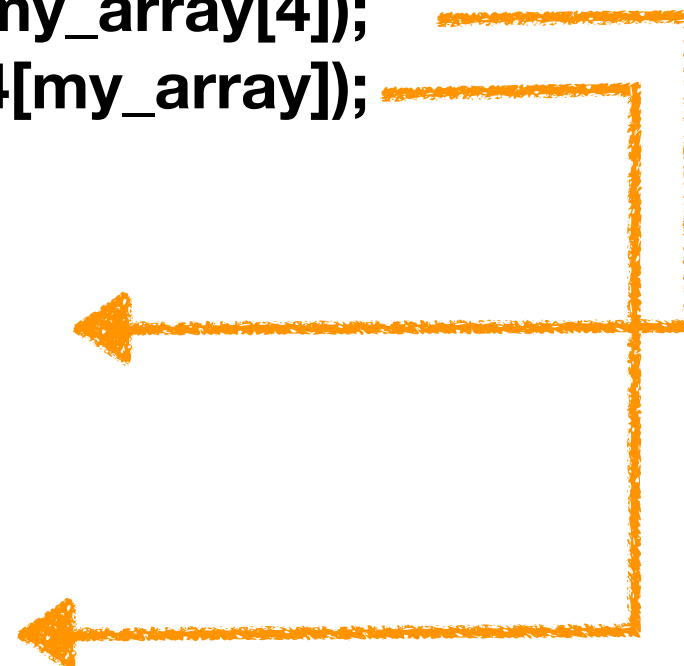


# Array access and pointer

```
int main (int argc, char ** argv) {  
    int my_array[] = {1, 2, 3, 4, 5, 6};  
    printf("%d", *(my_array + 4));  
    printf("%d", my_array[4]);  
    printf("%d", 4[my_array]);  
}
```

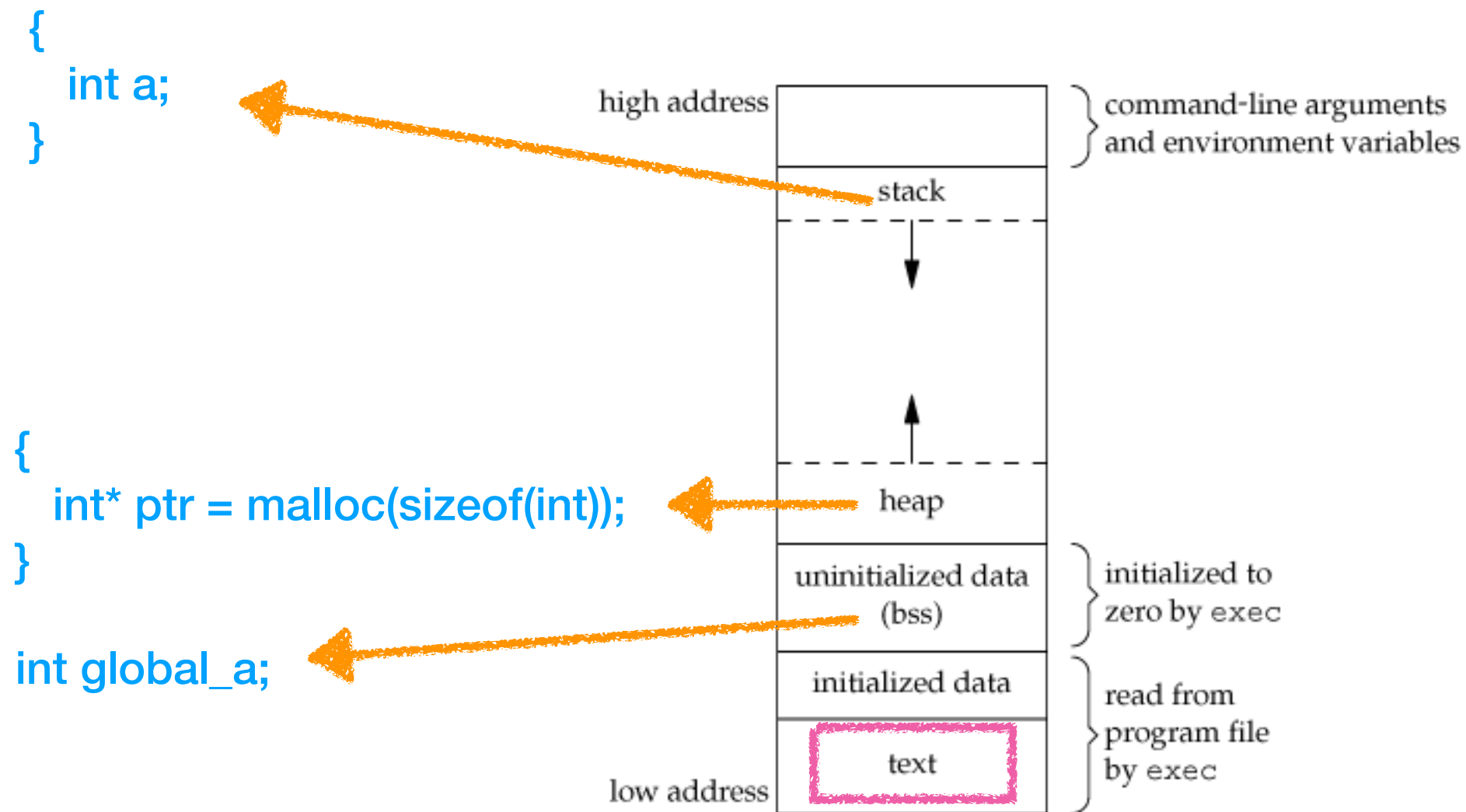
**\*(my\_array + 4)**

**\*(4 + my\_array)**



1	0x1000F000 (my_array)
2	0x1000F004
3	0x1000F008
4	0x1000F00C
5	0x1000F010
6	0x1000F014

# Memory layout of C programs



# Function Pointer in C

- A function pointer is just a variable with a unique interpretation.
  - The memory address(ADDR) of that variable.
  - The number of bytes used by the variable.
  - How to interpret the content stored in ADDR.
    - ▶ It represents a memory address.
    - ▶ The content in the memory address is part of the program.

# Function Pointer in C

```
int add_two (int val) {  
    return vale + 2;  
}  
  
int main (int argc, char ** argv) {  
    int (*func_ptr)(int) = add_two;  
    printf("%d\n", add_two(5));  
}
```

```
int main (int argc, char ** argv) {  
    int (func_ptr*)(int) = add_two;  
    printf("%d\n", add_two(5));  
}
```

```
int add_two (int val) {  
    return vale + 2;  
}
```

**0x1000**  
**(main)**

**0x2000**  
**(add\_two)**

**0x1000F008**  
**(func\_ptr)**

**0x2000**

**0x1000F00C**

# Function object in C and Python

```
int add_two (int val) {  
    return val + 2;  
}  
  
int main (int argc, char ** argv) {  
    int (*func_ptr)(int) = add_two;  
    printf("%d\n", add_two(5));  
}
```

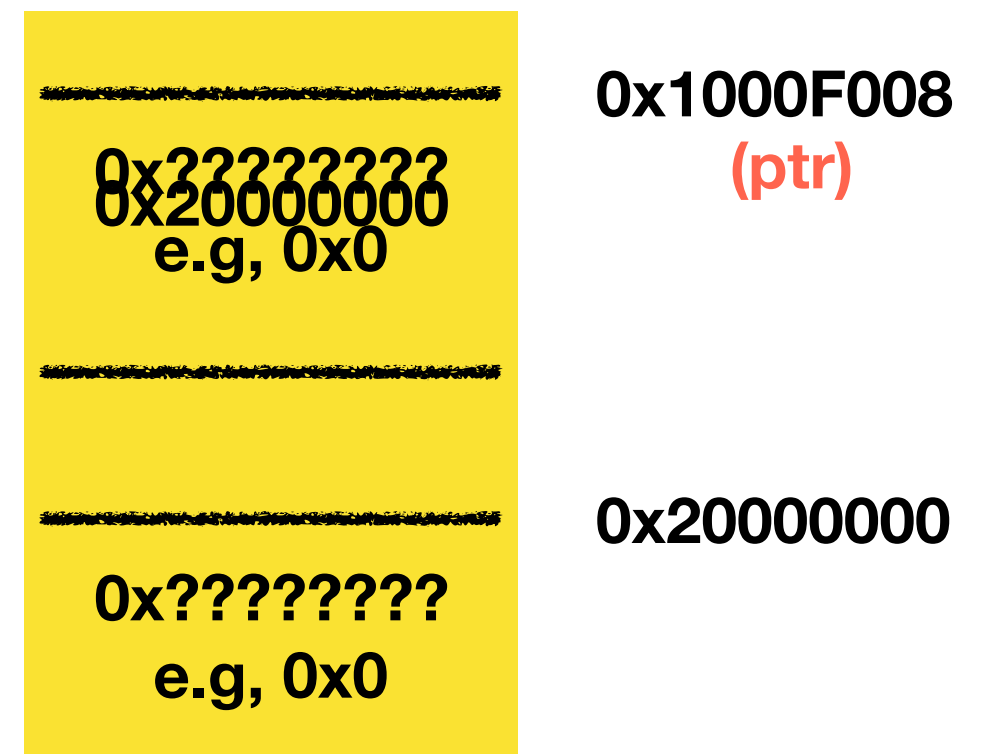
```
def add_two(val):  
    return val + 2  
  
def main():  
    func_ptr = add_two  
    printf(func_ptr(5))  
  
if __name__ == '__main__':  
    main()
```

# Segmentation Fault

- In computing, a segmentation fault (often shortened to segfault) or access violation is a fault, or failure condition, raised by hardware with memory protection, notifying an operation system the software has attempted to access a restricted area of memory (a memory access violation)  
— — Wikipedia

```
int main (int argc, char ** argv) {  
    int val = 5;  
    int *ptr;  
    printf("%d\n", *ptr);  
}
```

```
int *ptr = &val;  
int *ptr = malloc(sizeof(int));
```





# Core Dump

- In computing, a core dump(in Unix parlance), memory dump, or system dump consists of the recorded state of working memory of a computer program at a specific time, generally when the program has crashed or otherwise terminated abnormally.
- Let's see how it works.

# C v.s Python (loop)

```
int main (int argc, char ** argv) {  
    for (int i = 0; i < 10; i++) {  
        printf("%d\n", i);  
    }  
}
```

```
def main():  
    for i in range(10):  
        print(i)
```

# C v.s Python (variable)

```
int main (int argc, char **argv) {  
    int i = 1;  
    int j ;  
    float k = 1.5;  
}
```

```
def main():  
    i = 1  
    i = 1.5
```

# C v.s Python (variable)

```
int main (int argc, char **argv) {  
    void* var = null;  
  
    char* __temp1 = "12345";  
    var = (void*)__temp1;  
    printf("%s\n", (char*) var);  
  
    float __temp2 = 1.5;  
    var = (void*) &__temp2;  
    printf("%f\n", *((float*) var));  
  
}
```

```
def main():  
    var = "12345"  
    print(var)  
  
    var = 1.5  
    print(var)
```

