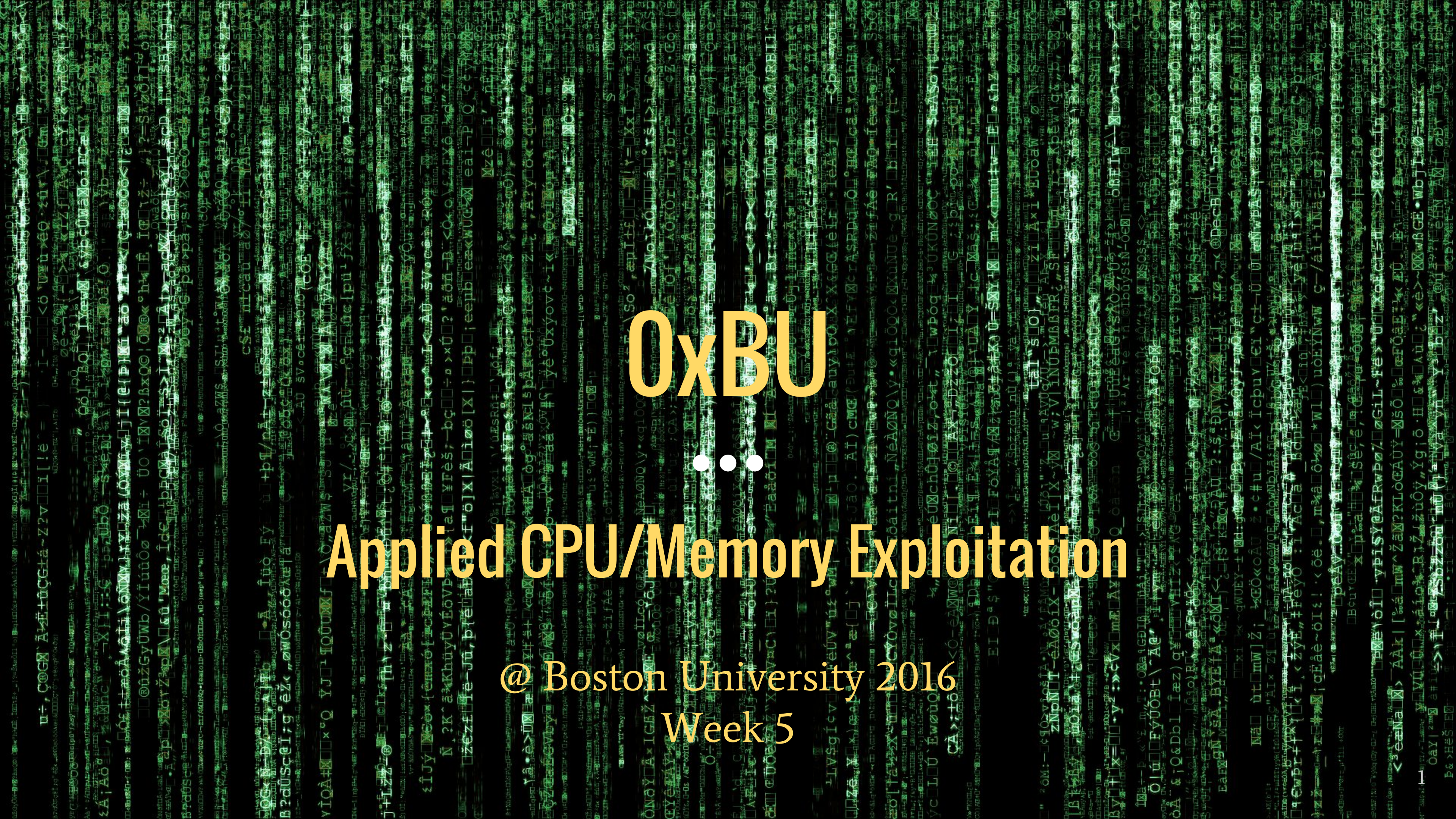


0xBU

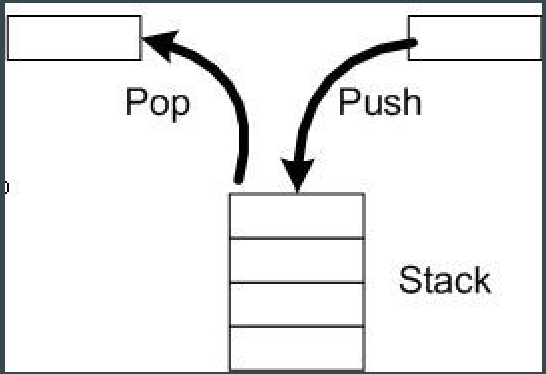
●●●

Applied CPU/Memory Exploitation

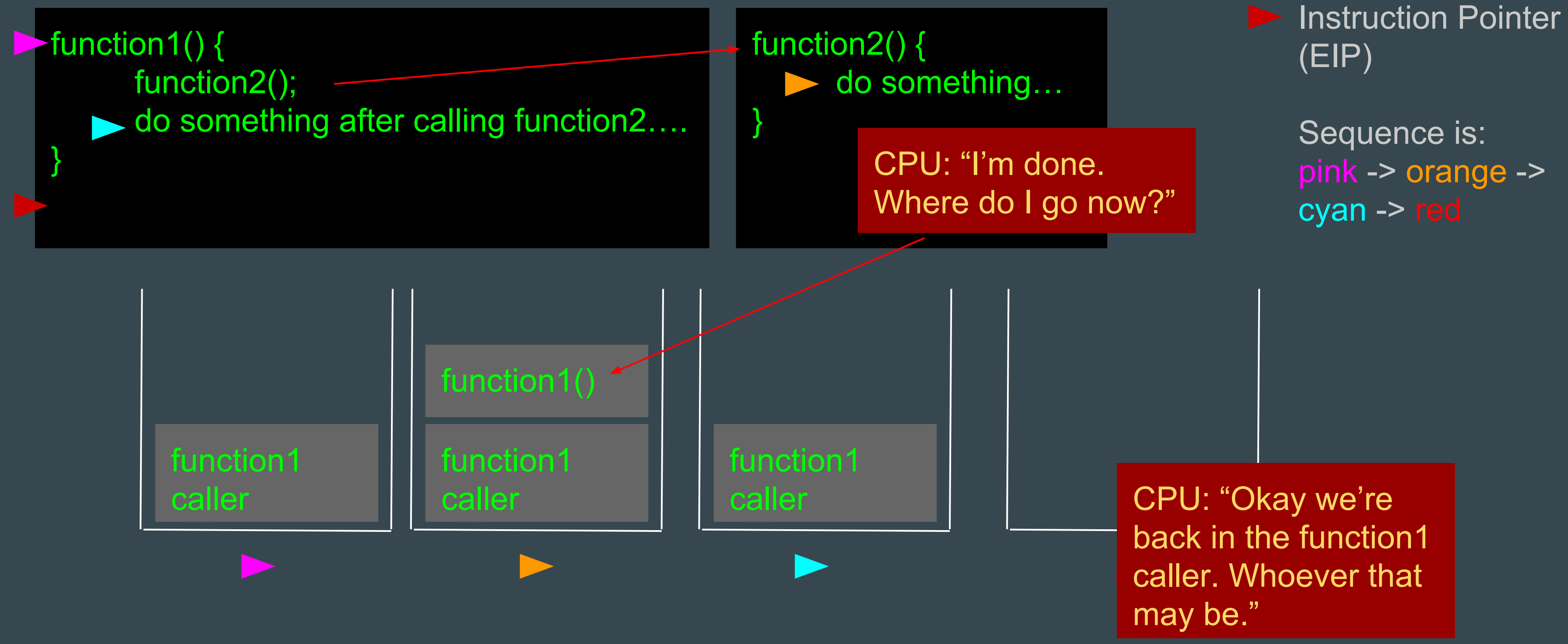
@ Boston University 2016
Week 5

The stack

- An abstract data structure that is similar to a real world “stack” of magazines
  - The last object in, is the first object out, LIFO
  - Two operations to alter the stack, push and pop
    - push something onto the top of the stack
    - pop the item at the top of the stack off
  - Computer scientists love them
- LIFO property allows simple and efficient way to backtrack to previous function caller
- Used to pass local variables to functions
- Used to keep track of scope



Stack from 100 feet away



```
►function1() {
        function2();
     ►  do something after calling function2….
}
►
```

```
function2() {
   ►  do something…
}
```

CPU: “I’m done. Where do I go now?”

CPU: “Okay we’re back in the function1 caller. Whoever that may be.”

► Instruction Pointer (EIP)

Sequence is:
pink -> orange -> cyan -> red

More complete stack view

```
► function1(a, b) {
          c = 5;
        ► d = function2(a, b, c);
          return d;
  }
►
```

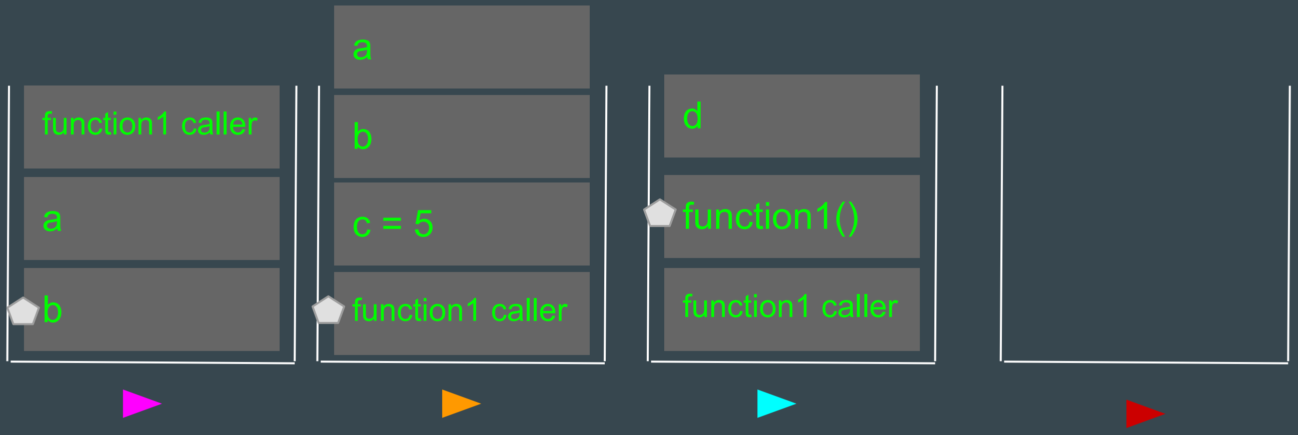
```
function2(a, b, c) {
          d = a + b + c;
        ► return d;
}
```

CPU: “I’m done. Good thing I saved where to go in the base pointer.”

⬟ Base Pointer (EBP)

► Instruction Pointer (EIP)

Sequence is:
pink -> orange -> cyan -> red



EAX: d

(Return register)

What controls a program

- Instruction Pointer (IP)
    - The current instruction being executed by the CPU
        - e.g. mov eax, 0x10; add eax, 0x5, etc...
- Base Pointer (BP)
    - Points to the current frame. Goes hand and hand with the stack pointer (SP).
        - e.g. After 20 nested calls (func1 -> func2 -> func3... func20), the BP is used to get a frame of reference for the local vars relating to func20.
- The Stack
    - Local variables are passed to functions on the stack.
        - e.g. The function "int add_two_numbers(int a, int b)" receives "a" and "b" off of the stack.
- The Registers (EAX, EBX, ECX, EDX, ESI, EDI, Special Purpose Registers, etc.)
    - The registers are used to store data that functions, or the code in general needs
        - e.g. Input vars, output vars, data being accessed from global, etc, etc...

How can *we* control the program?

- Programs are affected by us, and the landscape (also potentially us)
    - User input
        - e.g. mv <file1> <file2>; ping <ip>
    - Operating environment
        - e.g. Full hard drive; Server is getting hammered with requests, etc.
    - Shell environment
        - e.g. PATH, CWD, SPECIFIC_ENV_VAR_TO_A_PROGRAM, etc.

Control IP, BP, the stack, or the Regs control the world!

Let’s see it in action

$> Live @ `0xBU`

__libc_fini