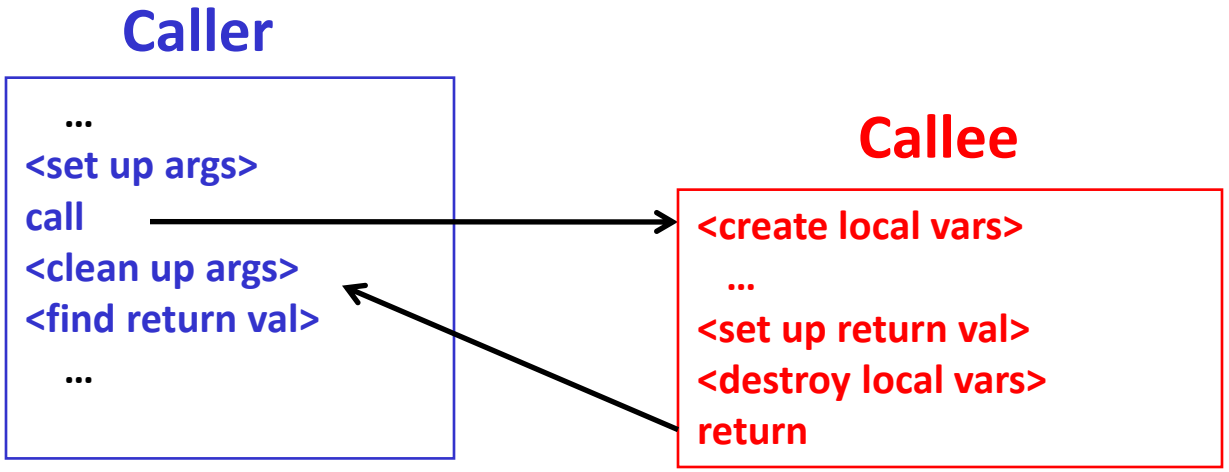
Section 5: Procedures & Stacks

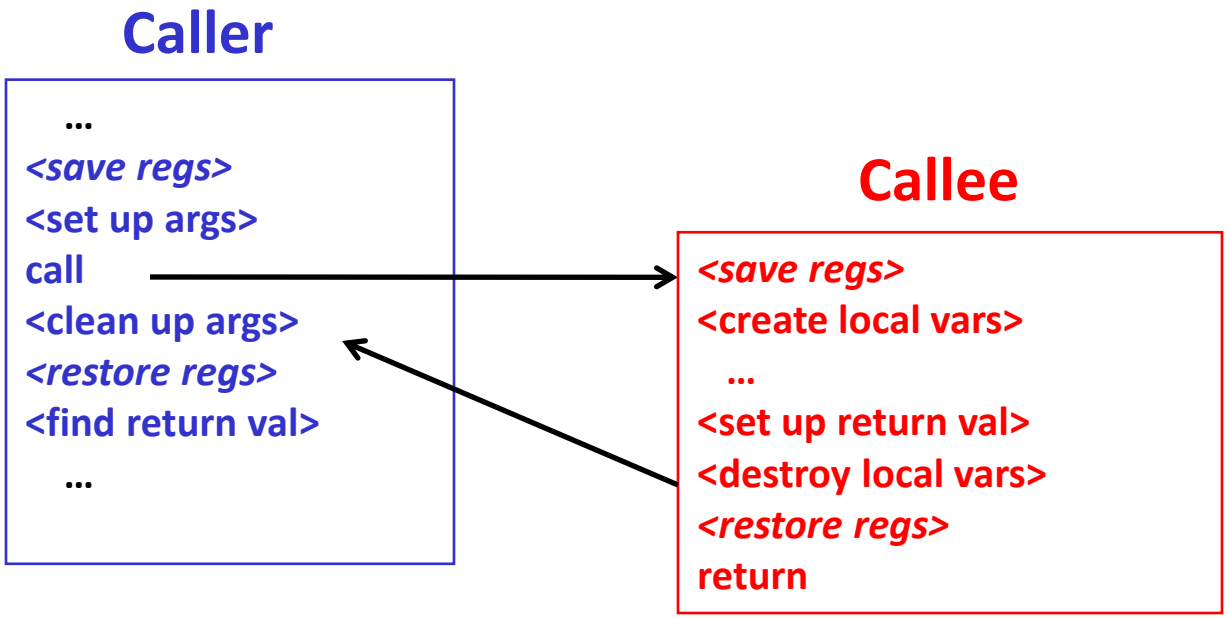
- Stacks in memory and stack operations
- The stack used to keep track of procedure calls
- Return addresses and return values
- Stack-based languages
- The Linux stack frame
- Passing arguments on the stack
- Allocating local variables on the stack
- Register-saving conventions
- Procedures and stacks on x64 architecture

Procedure Call Overview



- Callee must know where to find args
- Callee must know where to find "return address"
- Caller must know where to find return val
- Caller and Callee run on same CPU → use the same registers
  - Caller might need to save registers that Callee might use
  - Callee might need to save registers that Caller has used

Procedure Call Overview



- **The convention of where to leave/find things is called the procedure call linkage**
  - Details vary between systems
  - We will see the convention for IA32/Linux in detail
  - What could happen if our program didn't follow these conventions?

Procedure Control Flow

- **Use stack to support procedure call and return**
- **Procedure call: `call` *`label`***
  - Push return address on stack
  - Jump to *`label`*

Procedure Control Flow

- **Use stack to support procedure call and return**
- **Procedure call: `call` *`label`***
  - Push return address on stack
  - Jump to *`label`*
- **Return address:**
  - Address of instruction after `call`
  - Example from disassembly:

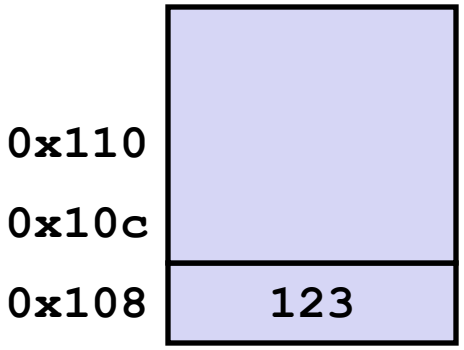
```
804854e:   e8 3d 06 00 00    call   8048b90 <main>
8048553:   50                pushl  %eax
```

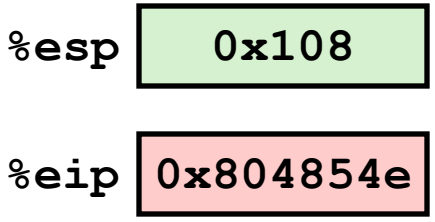
  - Return address = `0x8048553`
- **Procedure return: `ret`**
  - Pop return address from stack
  - Jump to address

Procedure Call Example

```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```

`call 8048b90`

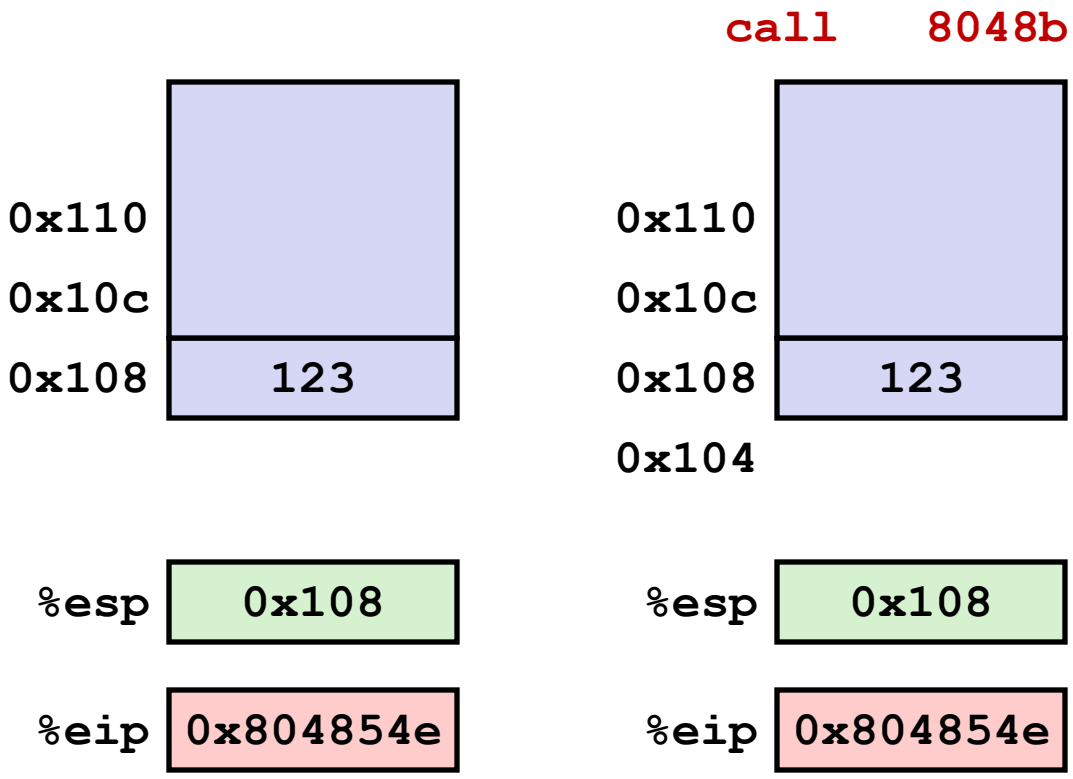




`%eip`: program counter

Procedure Call Example

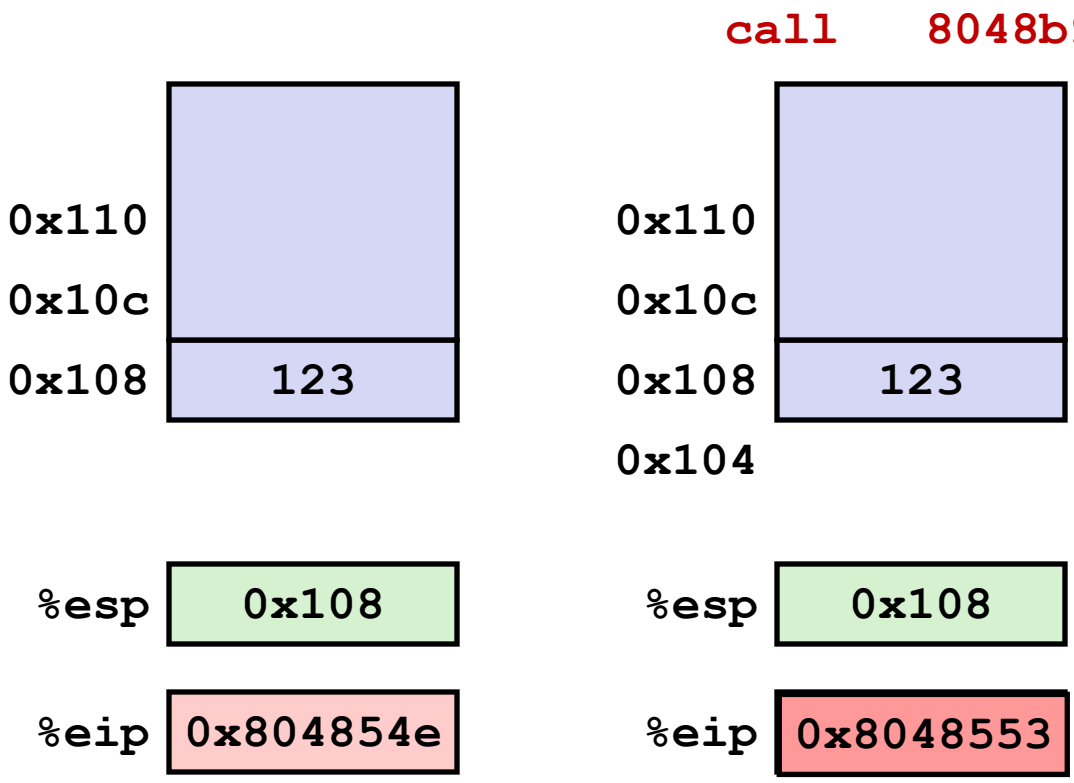
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

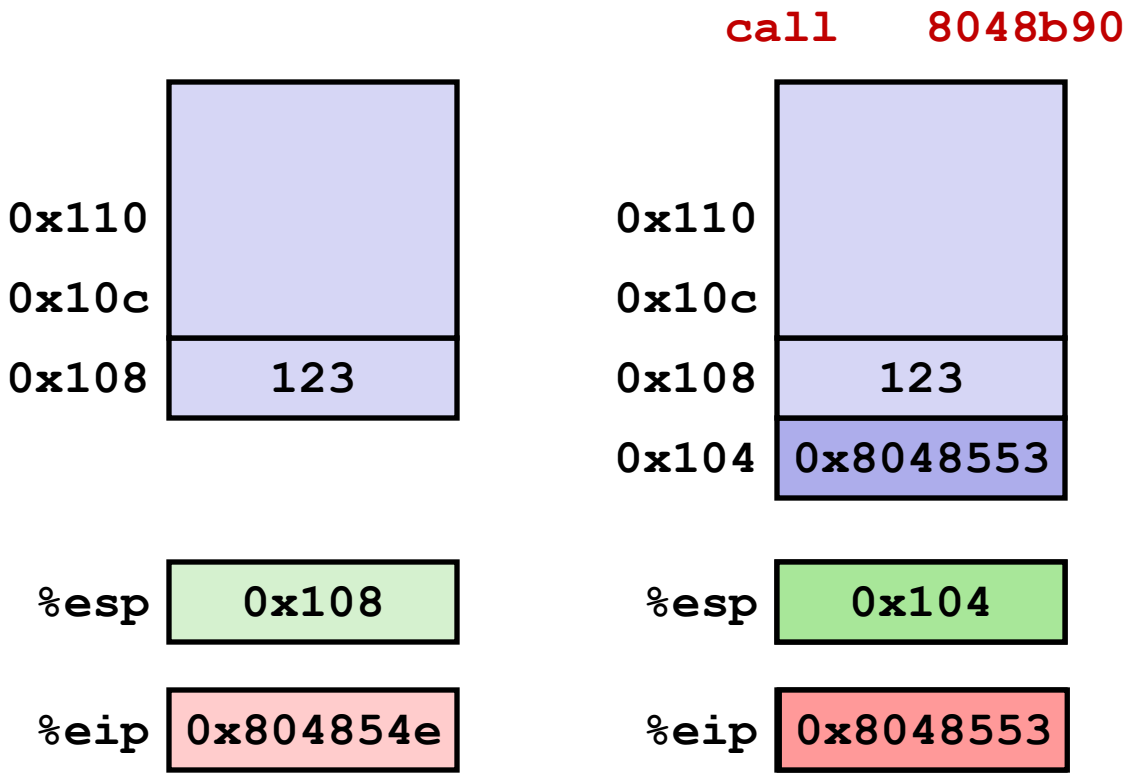
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

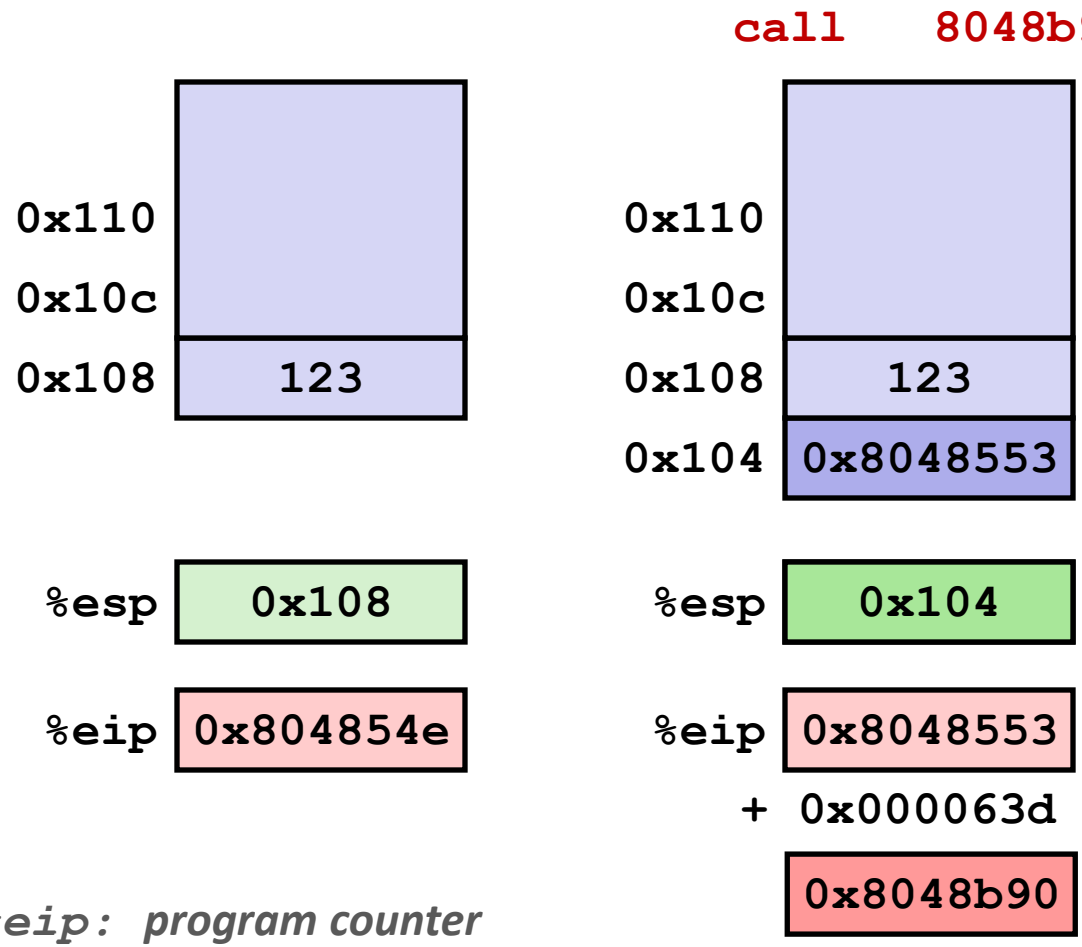
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```

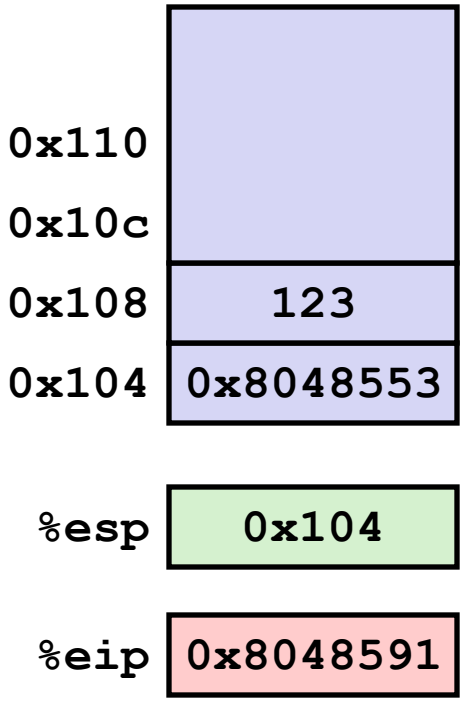


%eip: program counter

Procedure Return Example

```
8048591:      c3                              ret
```

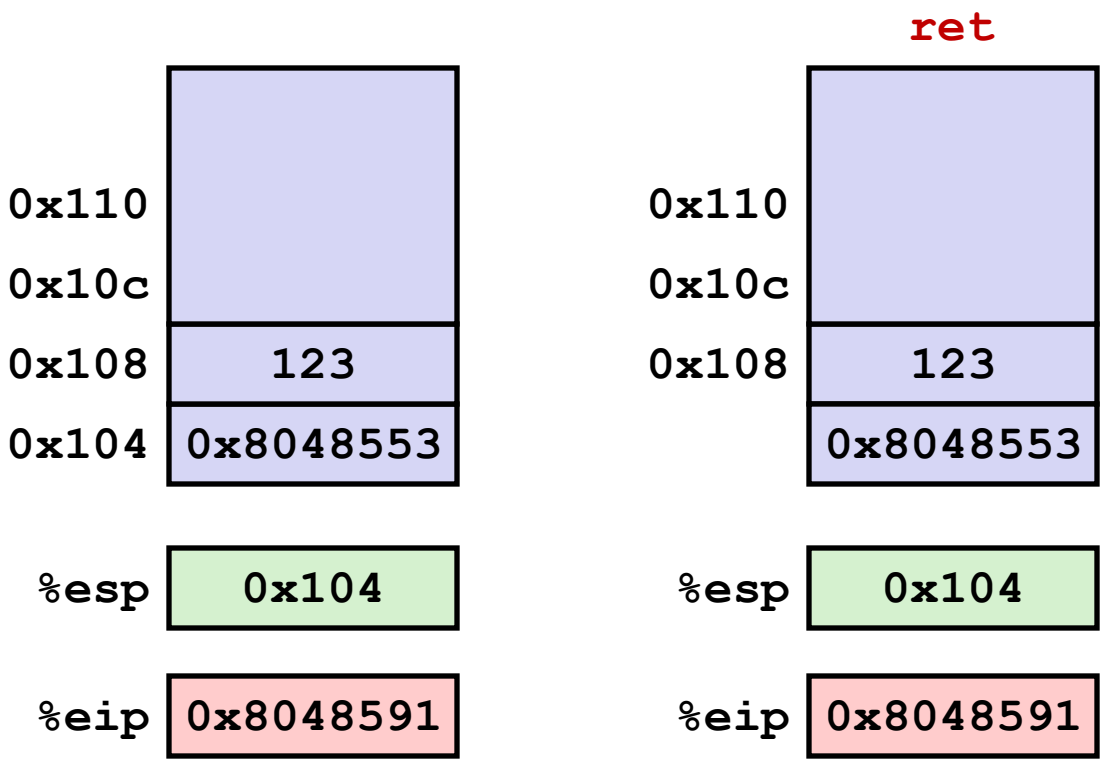
ret



%eip: program counter

Procedure Return Example

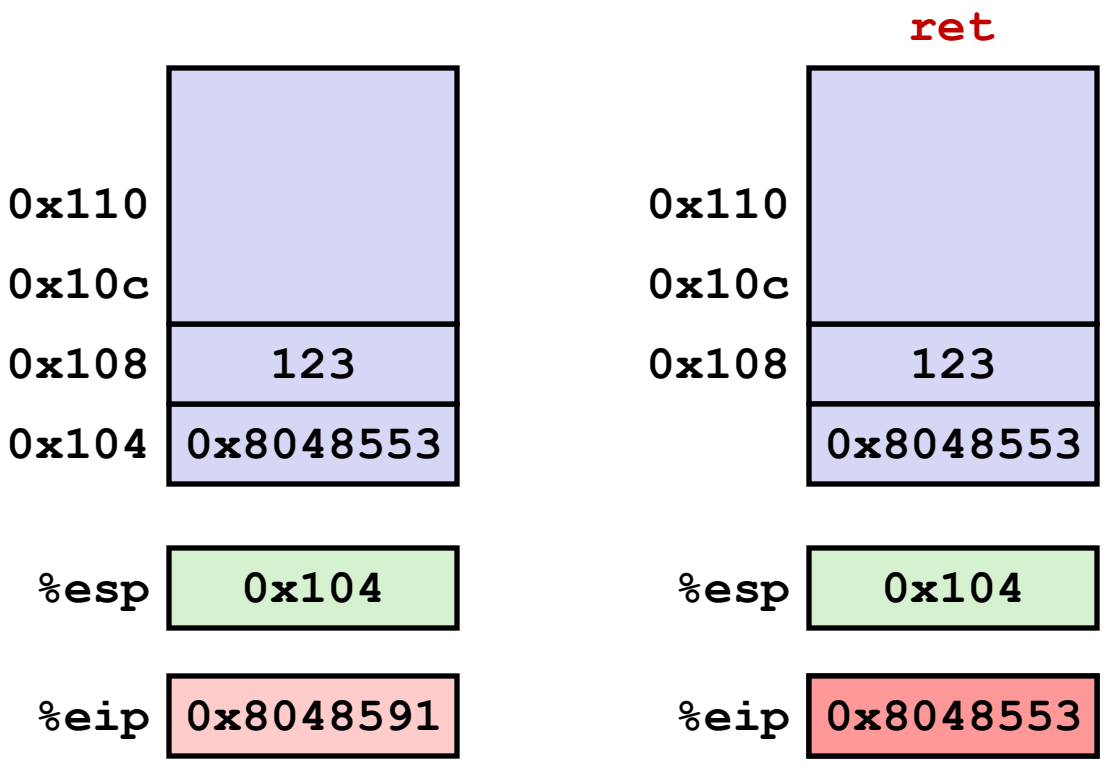
```
8048591:      c3                              ret
```



%eip: program counter

Procedure Return Example

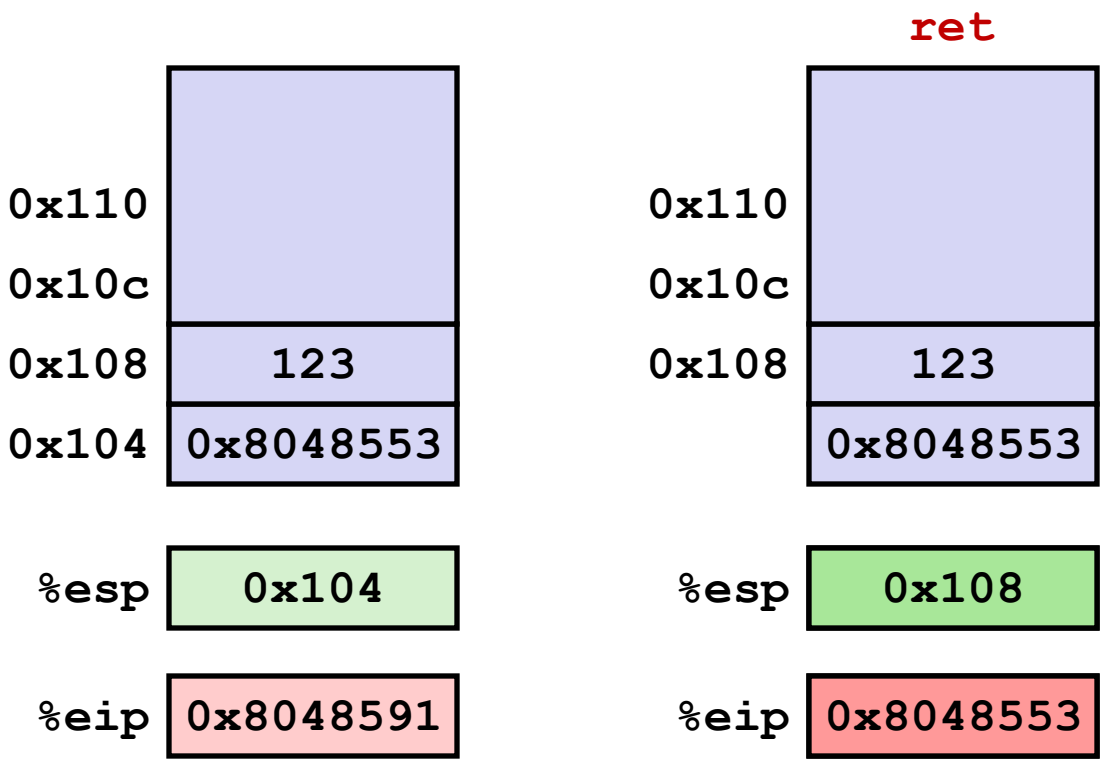
```
8048591:      c3                              ret
```



%eip: program counter

Procedure Return Example

```
8048591:      c3                              ret
```



%eip: program counter

Return Values

- **By convention, values returned by procedures are placed in the %eax register**
  - Choice of %eax is arbitrary, could have easily been a different register
- **Caller must make sure to save that register before calling a callee that returns a value**
  - Part of register-saving convention we'll see later
- **Callee placed return value (any type that can fit in 4 bytes – integer, float, pointer, etc.) into the %eax register**
  - For return values greater than 4 bytes, best to return a pointer to them
- **Upon return, caller finds the return value in the %eax register**