CSSE132
Introduction to Computer Systems

16 : Procedure calls

March 28, 2013

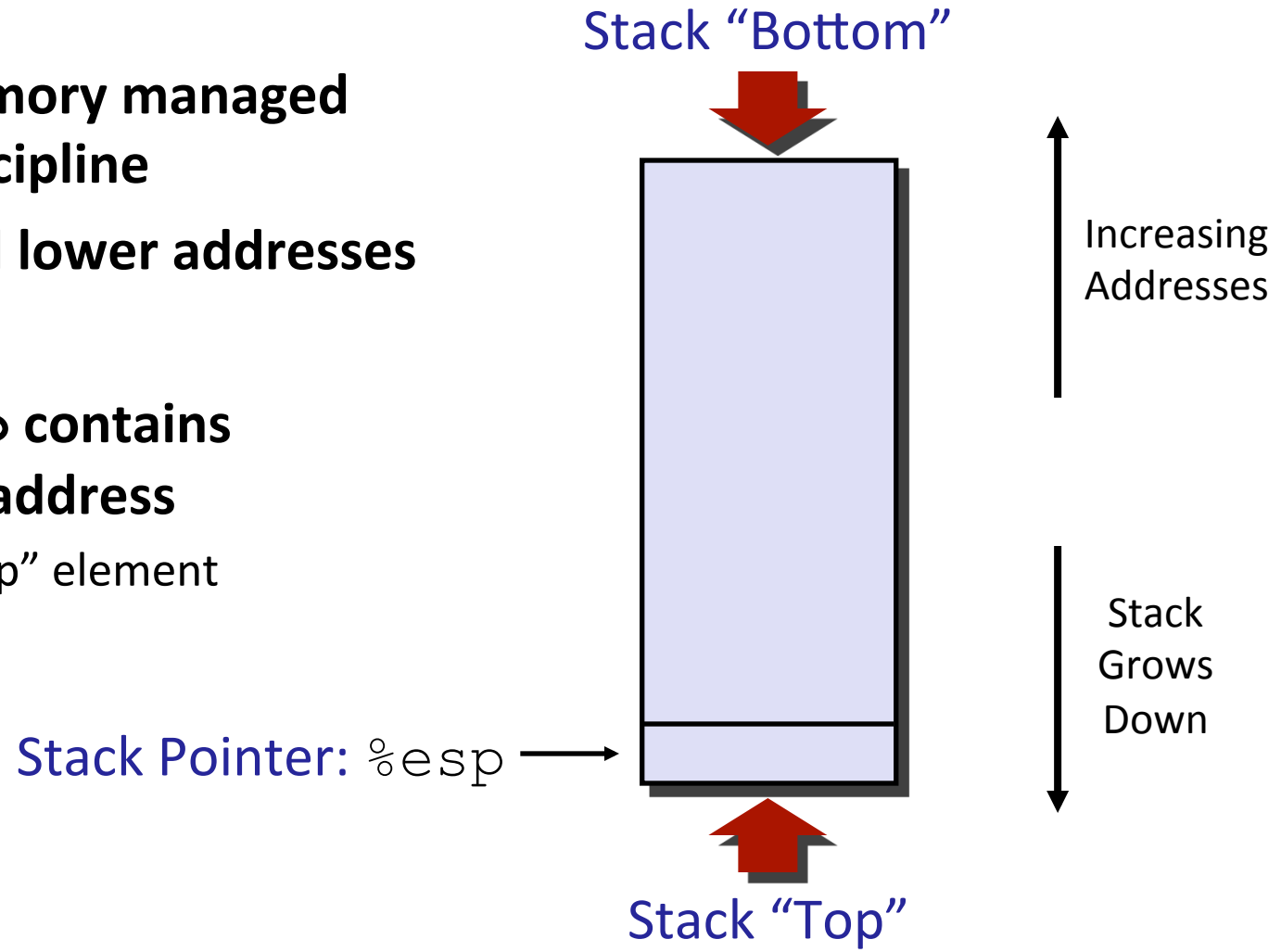
Overview

- Procedures and stack
- Calling conventions
- Stack details

IA32 Stack

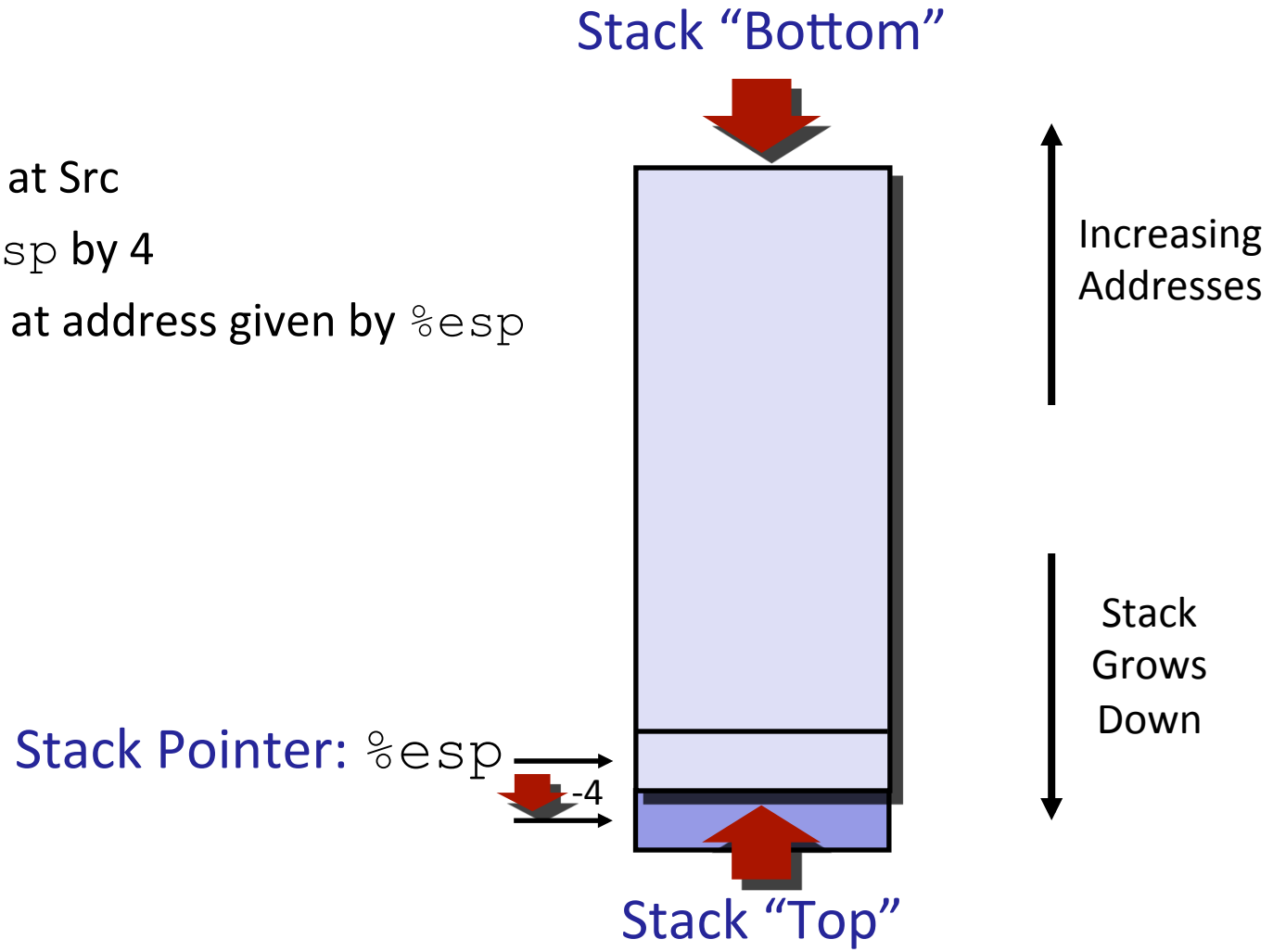
- **Region of memory managed with stack discipline**
- **Grows toward lower addresses**

- **Register `%esp` contains lowest stack address**
  - address of "top" element

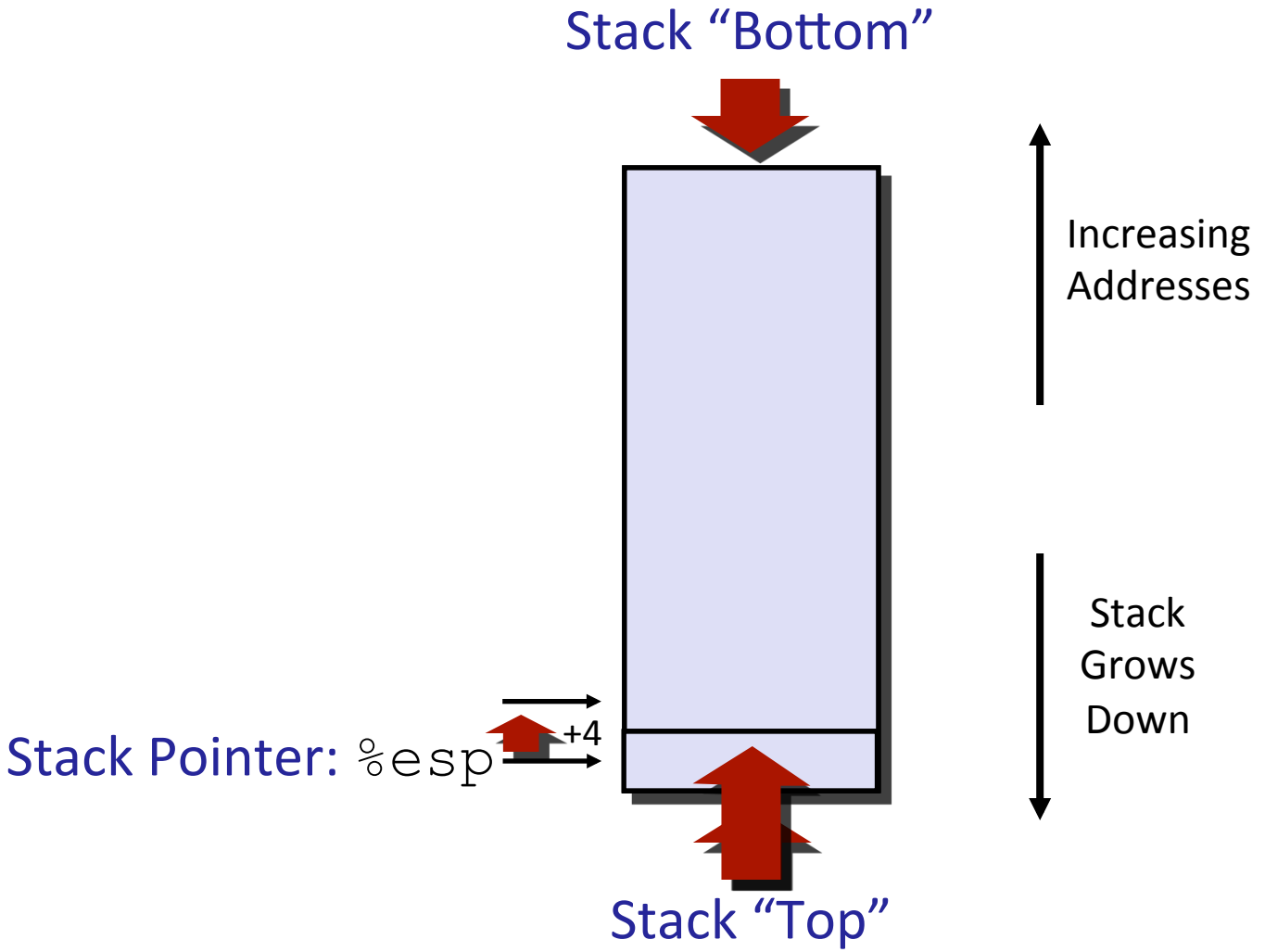


IA32 Stack: Push

- **`pushl Src`**
  - Fetch operand at Src
  - Decrement `%esp` by 4
  - Write operand at address given by `%esp`



IA32 Stack: Pop



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 the next instruction right after call
  - Example from disassembly

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

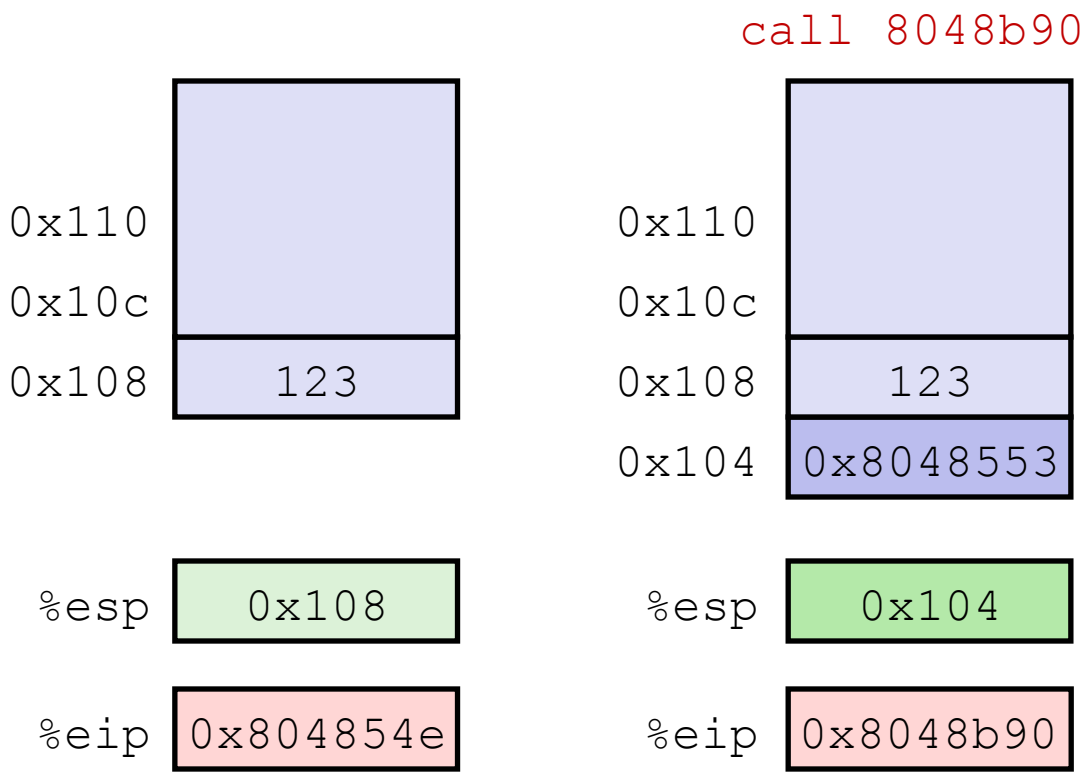
  - Return address = `0x8048553`
- **Procedure return: `ret`**
  - Pop address from stack
  - Jump to address

`leave` instruction : prepare for return

```
movl %ebp, %esp
popl %ebp
```

Procedure Call Example

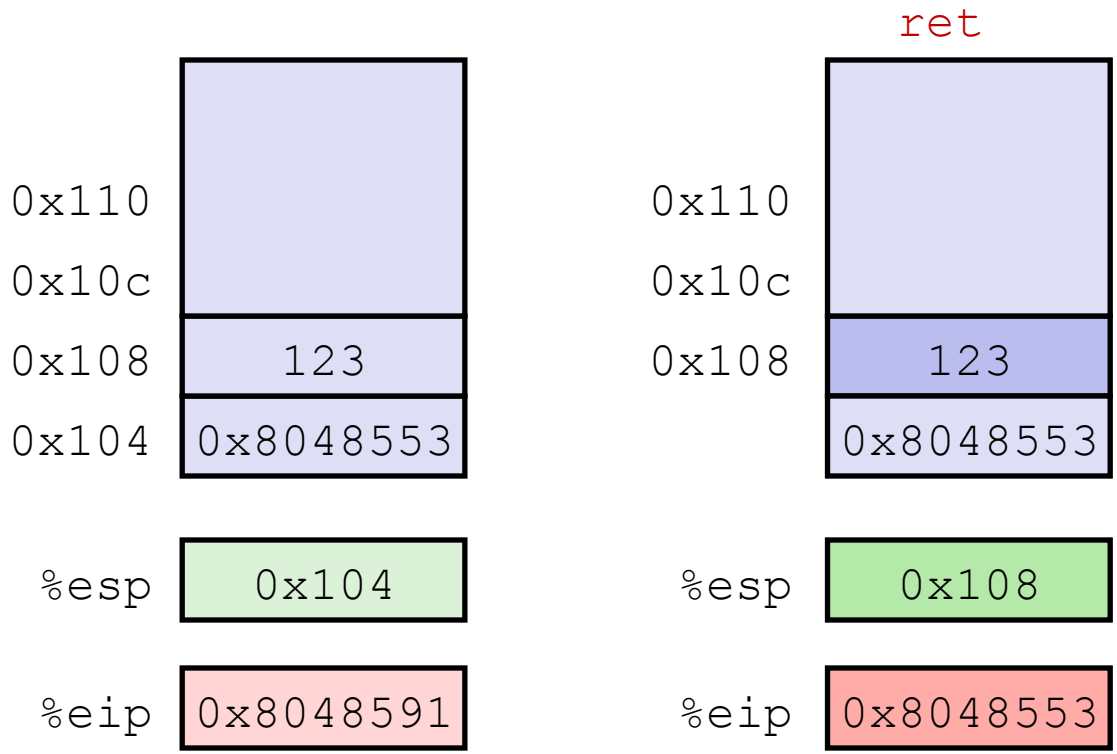
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Return Example

```
8048591:    c3                  ret
```



`%eip:` program counter

Stack-Based Languages

- **Languages that support recursion**
  - e.g., C, Pascal, Java
  - Code must be "Reentrant"
    - Multiple simultaneous instantiations of single procedure
  - Need some place to store state of each instantiation
    - Arguments
    - Local variables
    - Return pointer
- **Stack discipline**
  - State for given procedure needed for limited time
    - From when called to when return
  - Callee returns before caller does
- **Stack allocated in Frames**
  - state for single procedure instantiation

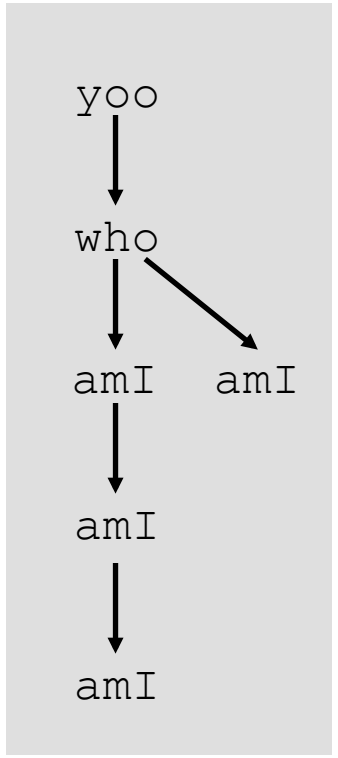
Call Chain Example

```
yoo(…)
{
   •
   •
   who();
   •
   •
}
```

```
who(…)
{
   • • •
   amI();
   • • •
   amI();
   • • •
}
```

```
amI(…)
{
   •
   •
   amI();
   •
   •
}
```

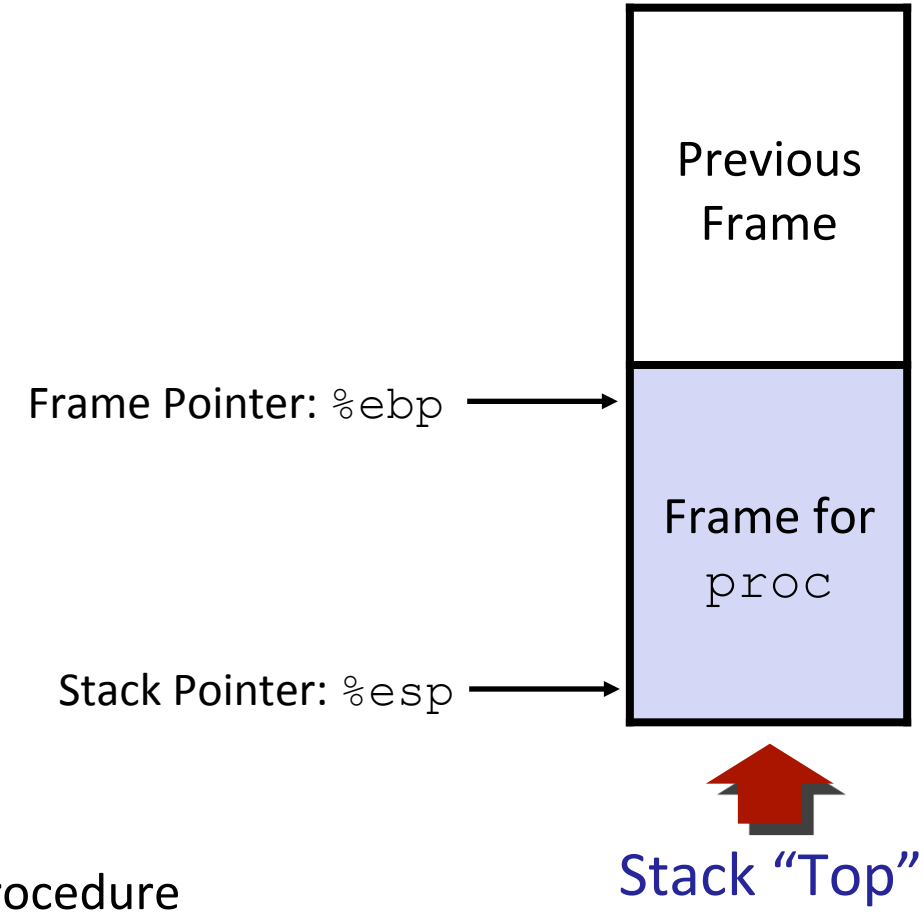
Example
Call Chain



Procedure `amI()` is recursive

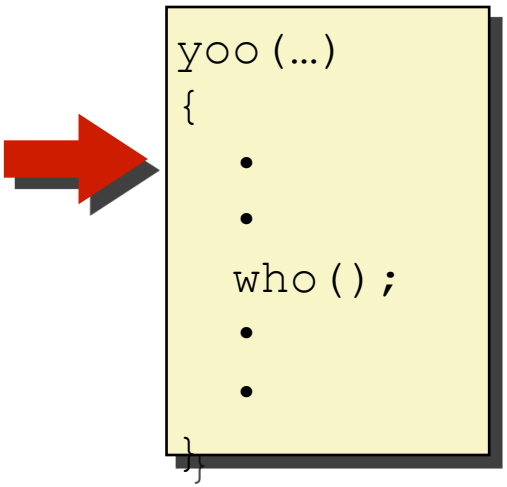
Stack Frames

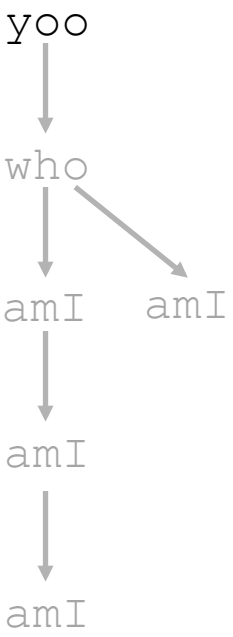
- **Contents**
  - Local variables
  - Return information
  - Temporary space

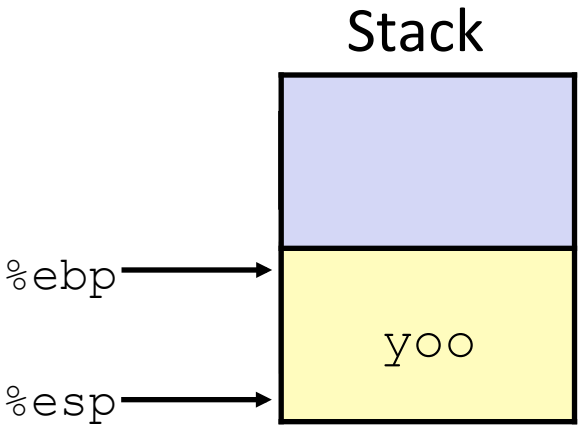


- **Management**
  - Space allocated when enter procedure
    - "Set-up" code
  - Deallocated when return
    - "Finish" code

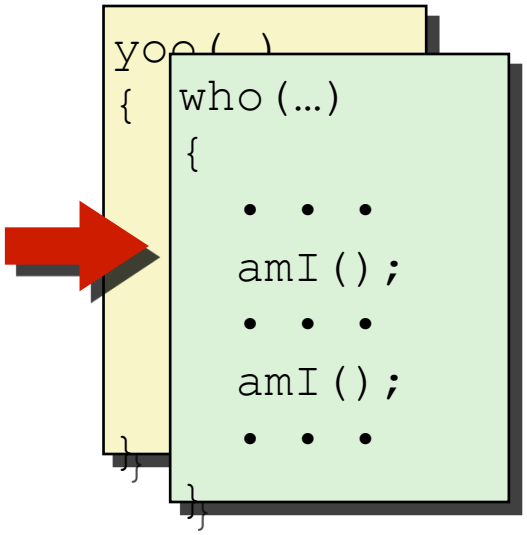
Example

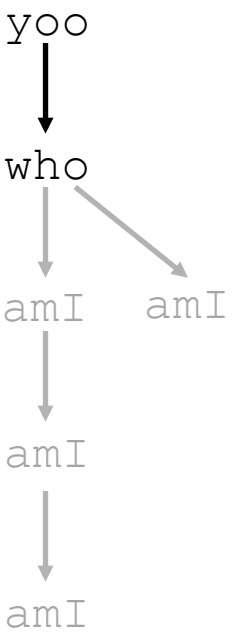


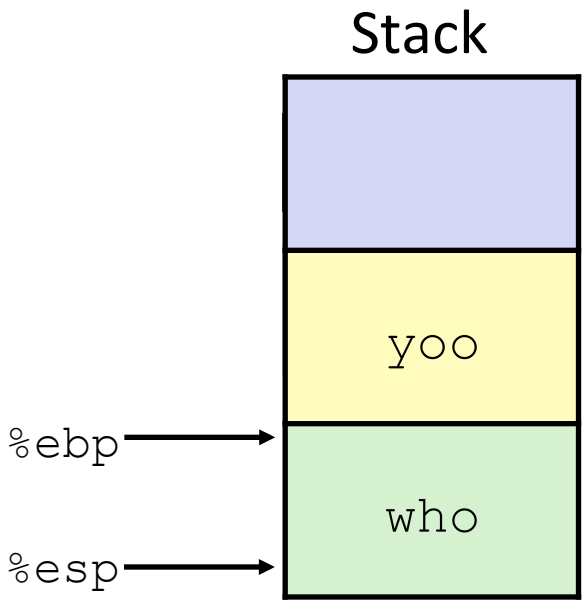




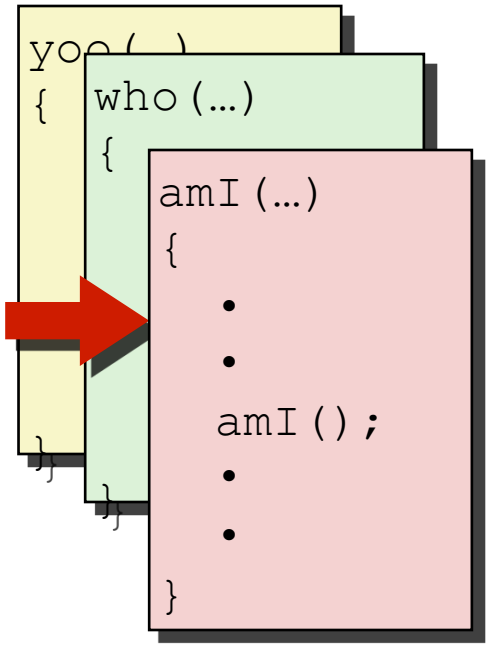
Example

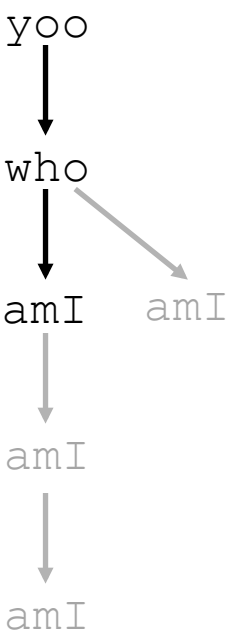


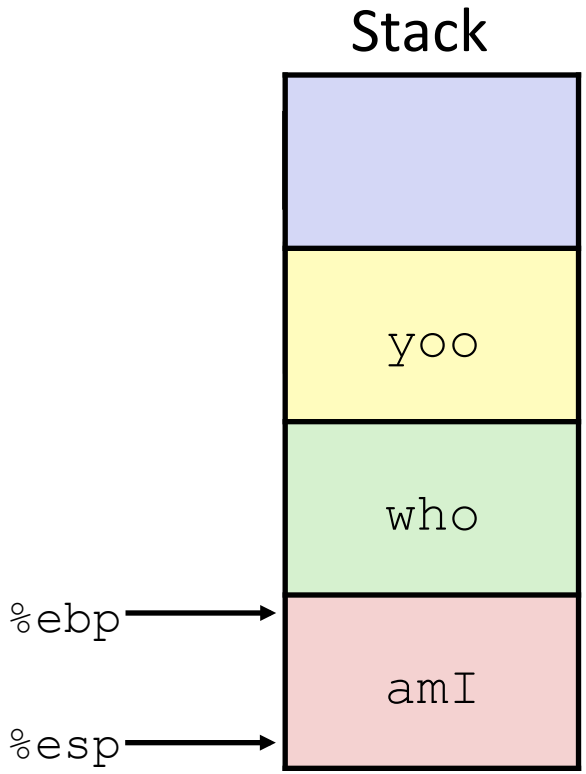




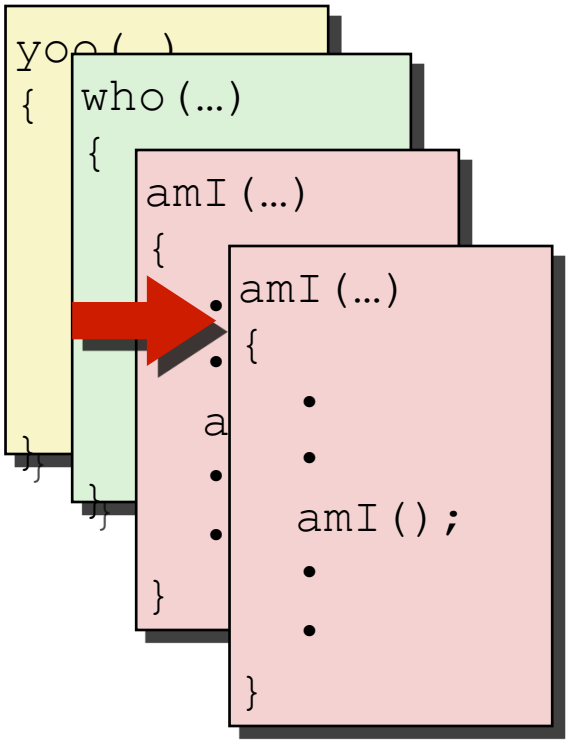
Example

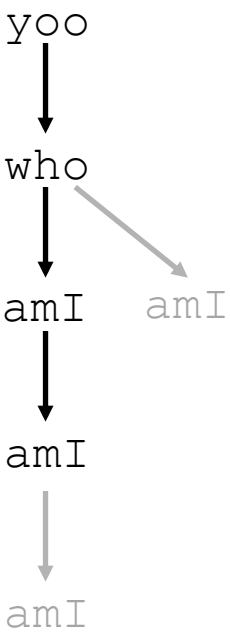

yoo( )
{
who(…)
{
amI(…)
{
•
•
amI();
•
•
}
}
}

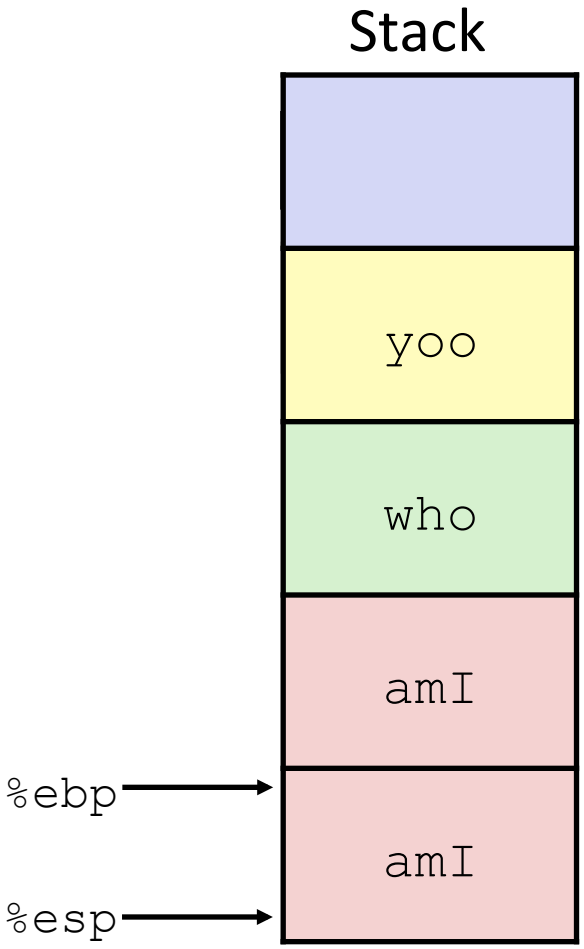

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
%ebp
%esp

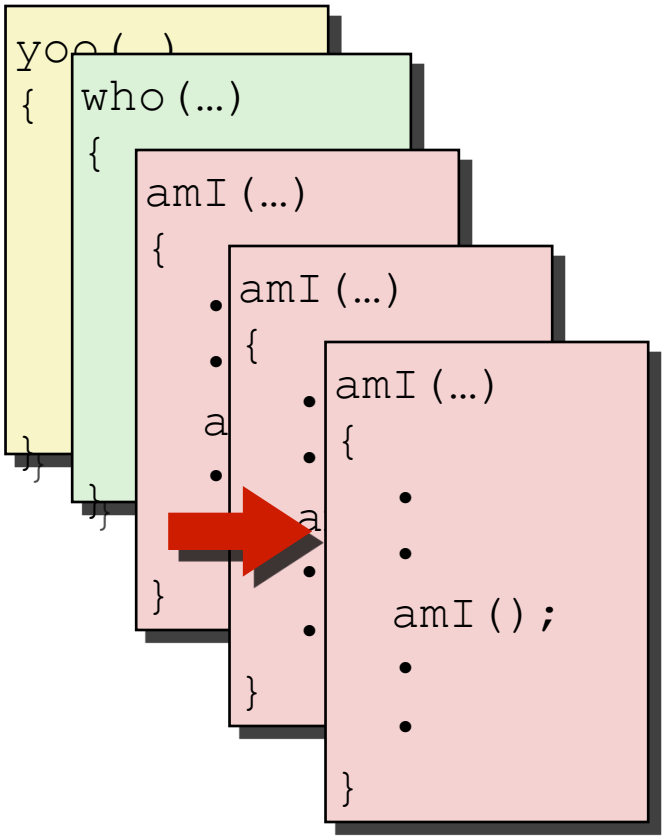
Example

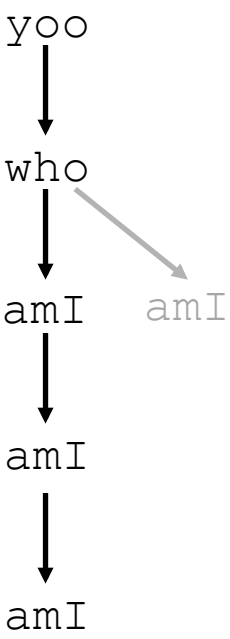

who(…)
{
amI(…)
{
amI(…)
{
amI();
}

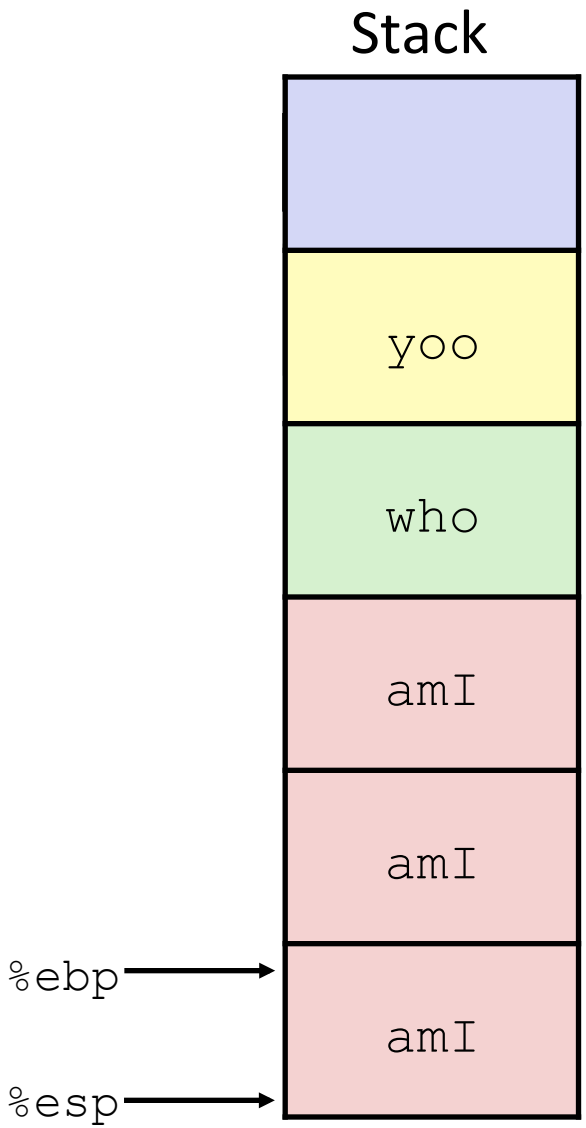

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
amI
%ebp
%esp

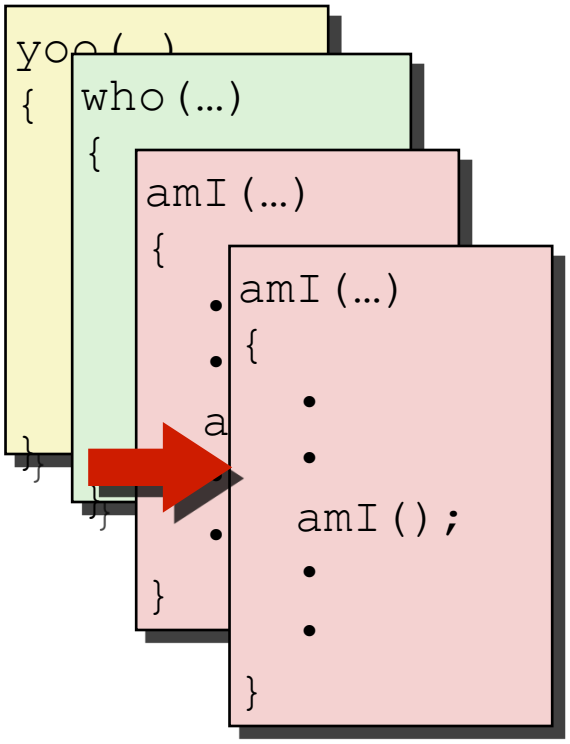
Example

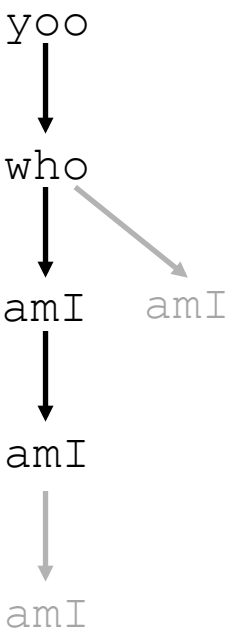

who (…)
{
amI (…)
{
amI (…)
{
amI();
}

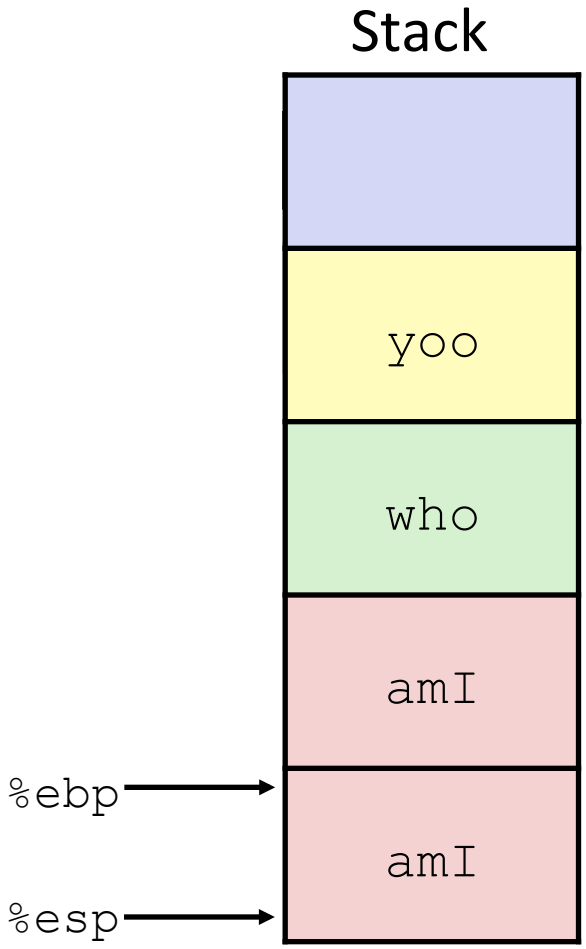

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
amI
amI
%ebp
%esp

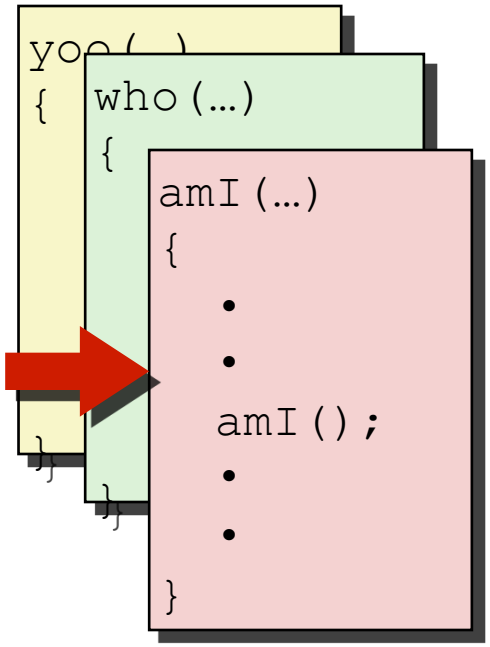
Example

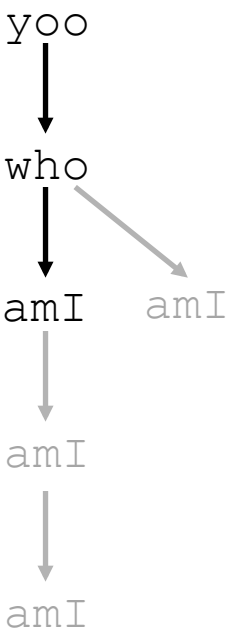

who(…)
{
amI(…)
{
amI(…)
{
amI();
}

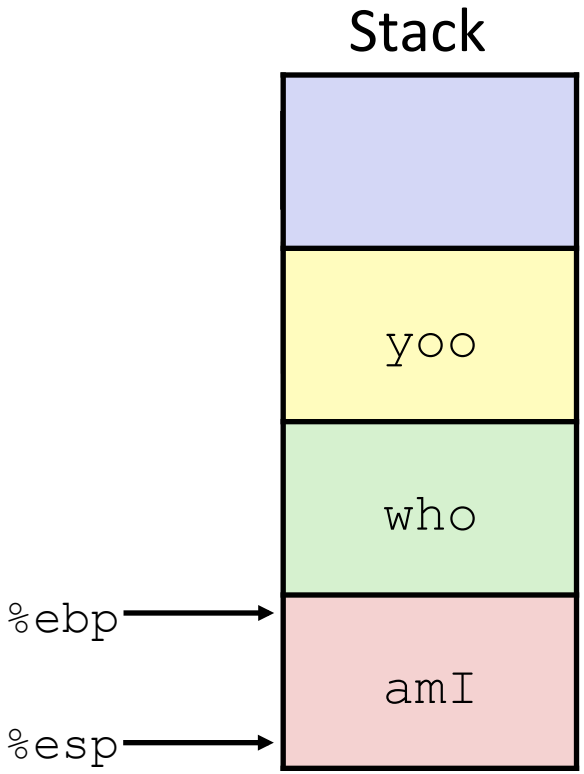

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
amI
%ebp
%esp

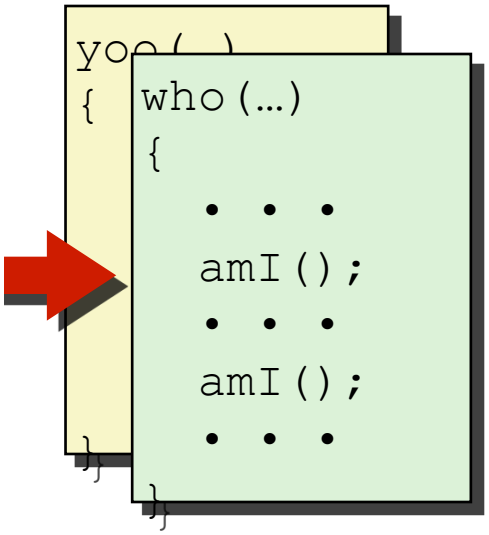
Example

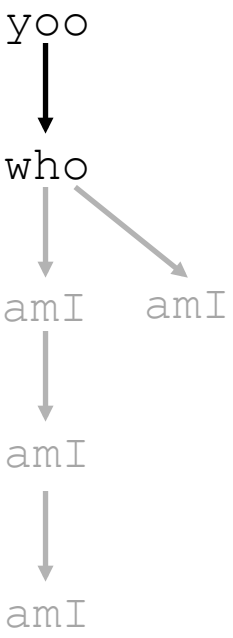

yoo( )
{
who(…)
{
amI(…)
{
•
•
amI();
•
•
}
}
}

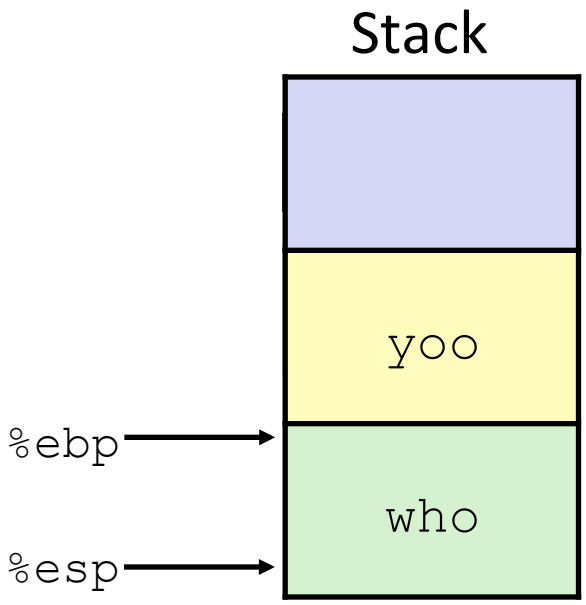

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
%ebp
%esp

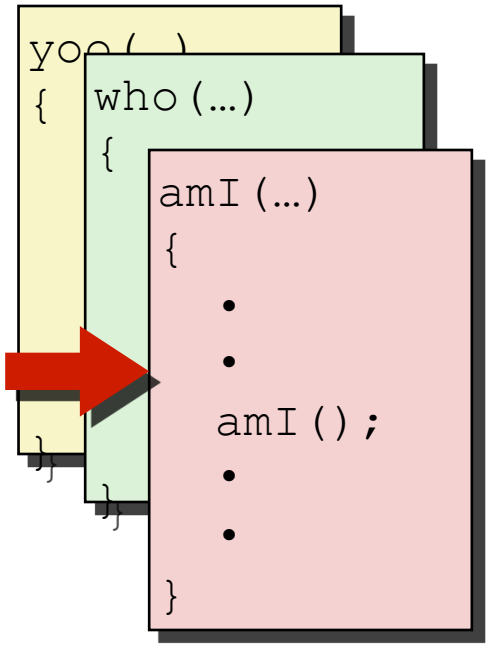
Example

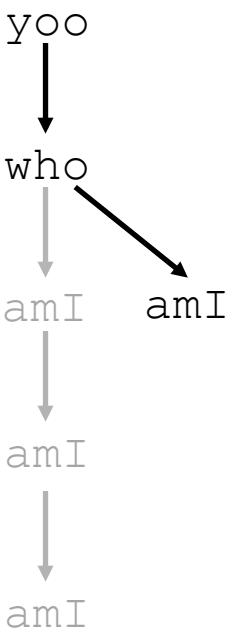


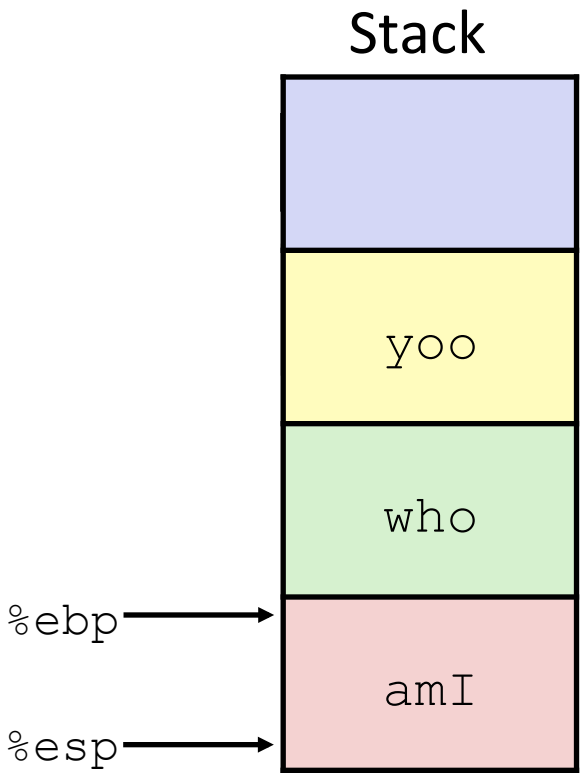




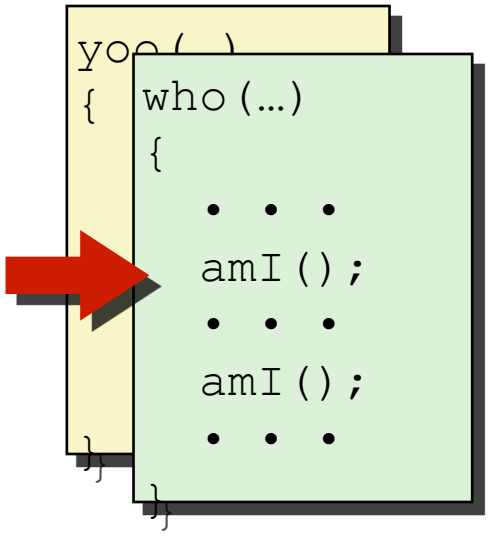
Example

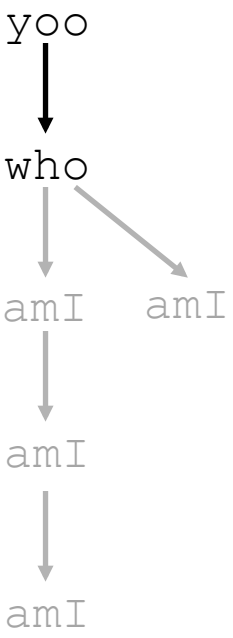

yoo( )
{
who(…)
{
amI(…)
{
•
•
amI();
•
•
}
}
}

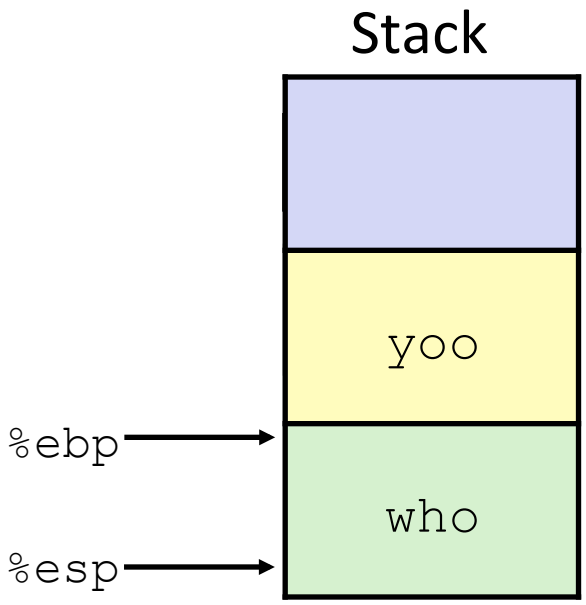

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
%ebp
%esp

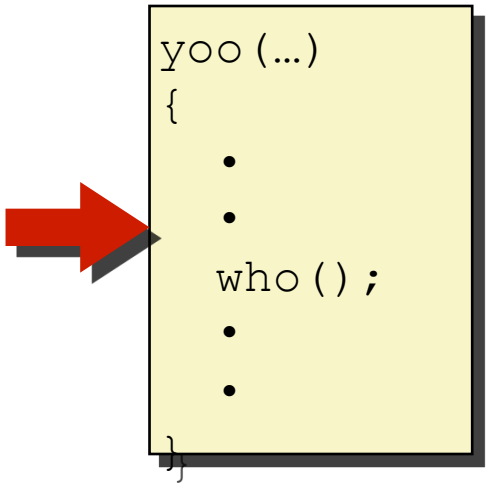
Example

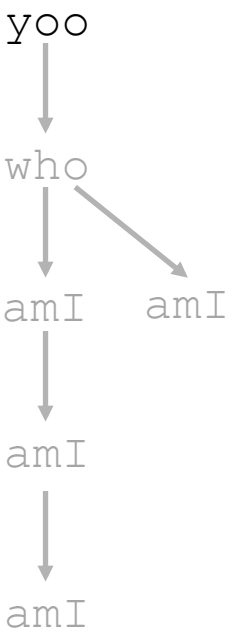


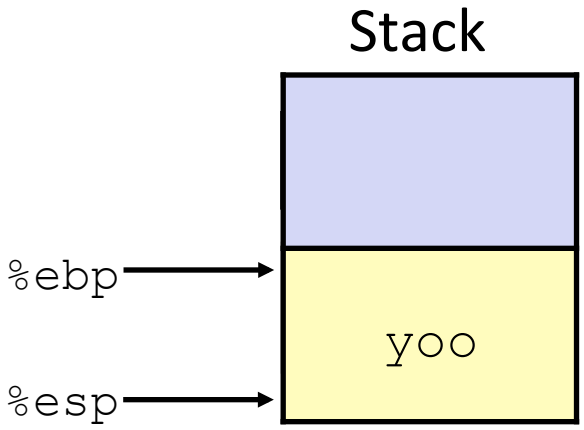




Example







Overview

- Procedures and stack
- Calling conventions
- Stack details

Register Saving Conventions

- **When procedure `yoo` calls `who`:**
  - `yoo` is the caller
  - `who` is the callee
- **Can register be used for temporary storage?**

```
yoo:
    • • •
    movl $15213, %edx
    call who
    addl %edx, %eax
    • • •
    ret
```

```
who:
    • • •
    movl 8(%ebp), %edx
    addl $18243, %edx
    • • •
    ret
```

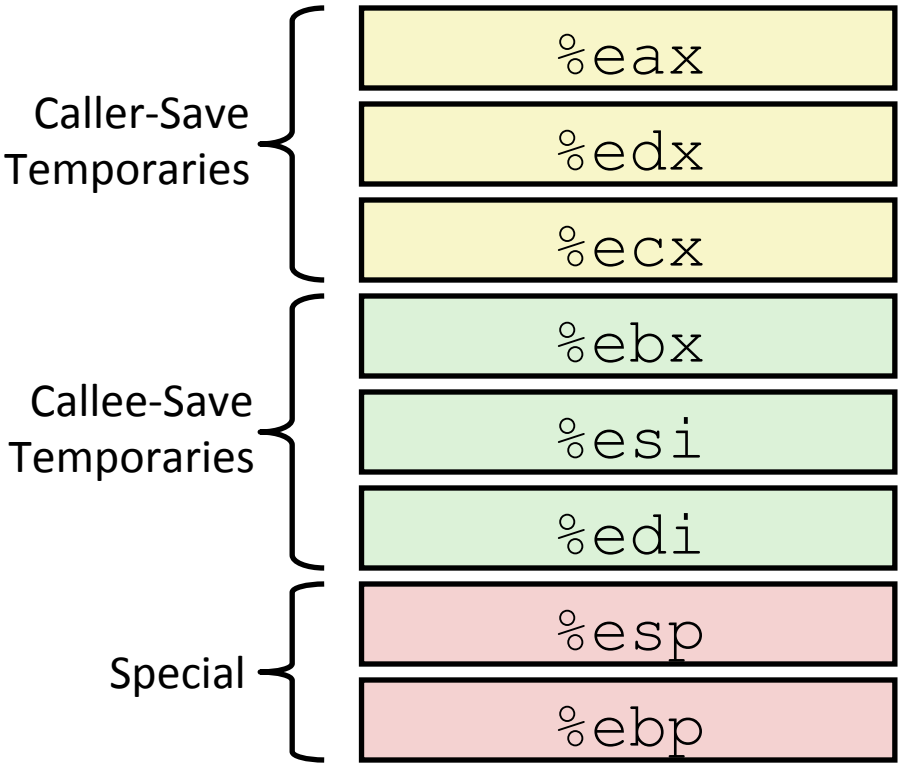
  - Contents of register `%edx` overwritten by `who`
  - This could be trouble ➙ something should be done!
    - Need some coordination

Register Saving Conventions

- **When procedure `yoo` calls `who`:**
  - `yoo` is the caller
  - `who` is the callee
- **Can register be used for temporary storage?**
- **Conventions**
  - "Caller Save"
    - Caller saves temporary values in its frame before the call
  - "Callee Save"
    - Callee saves temporary values in its frame before using

IA32/Linux+Windows Register Usage

- **`%eax`, `%edx`, `%ecx`**
  - Caller saves prior to call if values are used later
- **`%eax`**
  - also used to return integer value
- **`%ebx`, `%esi`, `%edi`**
  - Callee saves if wants to use them
- **`%esp`, `%ebp`**
  - special form of callee save
  - Restored to original values upon exit from procedure



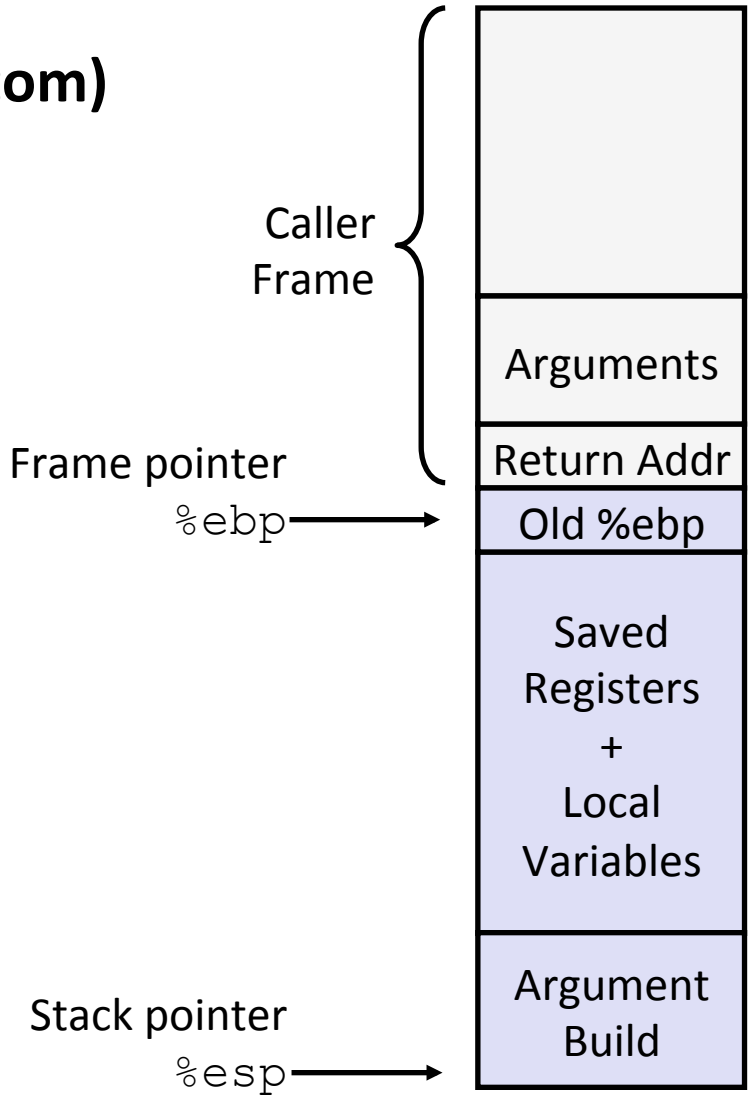
Overview

- Procedures and stack
- Calling conventions
- Stack details

IA32/Linux Stack Frame

- **Current Stack Frame ("Top" to Bottom)**
  - "Argument build:"
    Parameters for function about to call
  - Local variables
    If can't keep in registers
  - Saved register context
  - Old frame pointer

- **Caller Stack Frame**
  - Return address
    - Pushed by `call` instruction
  - Arguments for this call



Revisiting swap

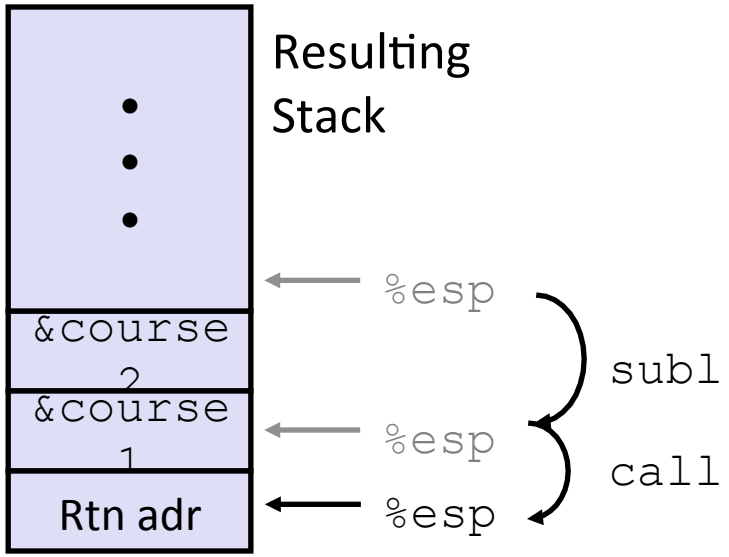
```
int course1 = 15213;
int course2 = 18243;

void call_swap() {
  swap(&course1, &course2);
}
```

Calling `swap` from `call_swap`

```
call_swap:
    • • •
    subl    $8, %esp
    movl    $course2, 4(%esp)
    movl    $course1, (%esp)
    call    swap
    • • •
```

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



Revisiting swap

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl  %esp, %ebp
    pushl %ebx

    movl  8(%ebp), %edx
    movl  12(%ebp), %ecx
    movl  (%edx), %ebx
    movl  (%ecx), %eax
    movl  %eax, (%edx)
    movl  %ebx, (%ecx)

    popl  %ebx
    popl  %ebp
    ret
```

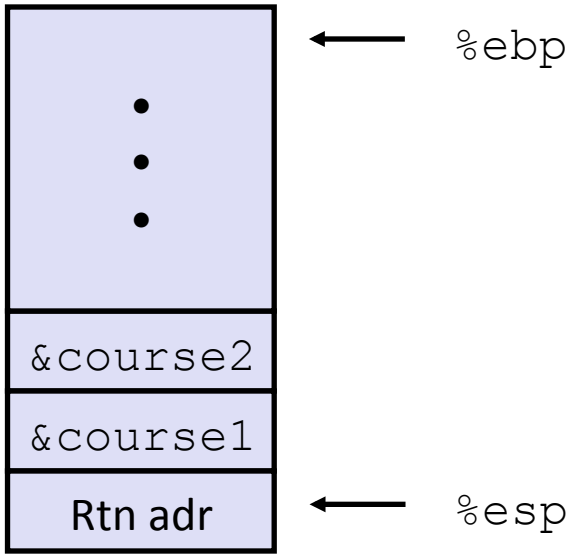
Set Up

Body

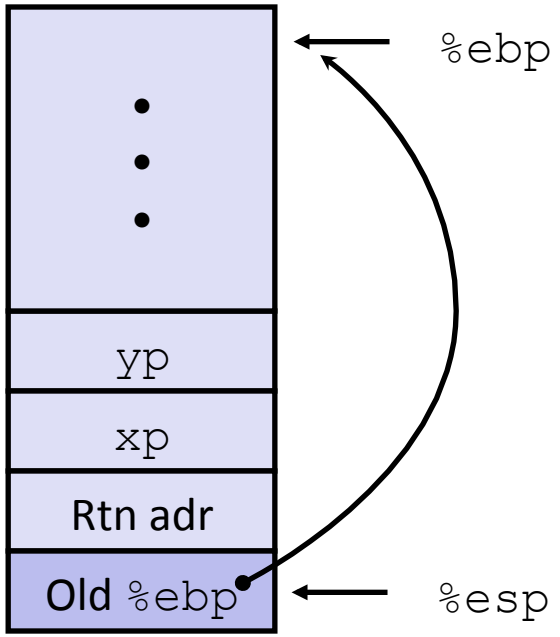
Finish

swap Setup #1

Entering Stack



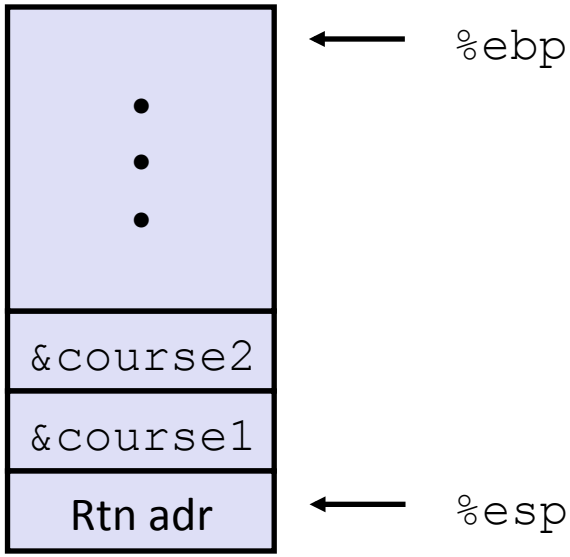
Resulting Stack



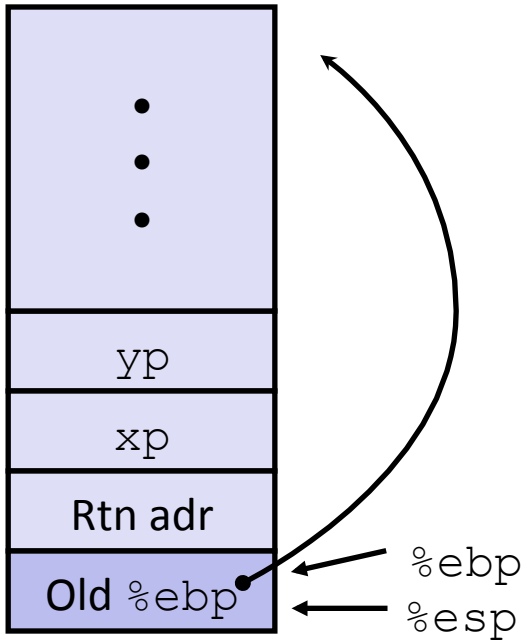

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

swap Setup #2

Entering Stack



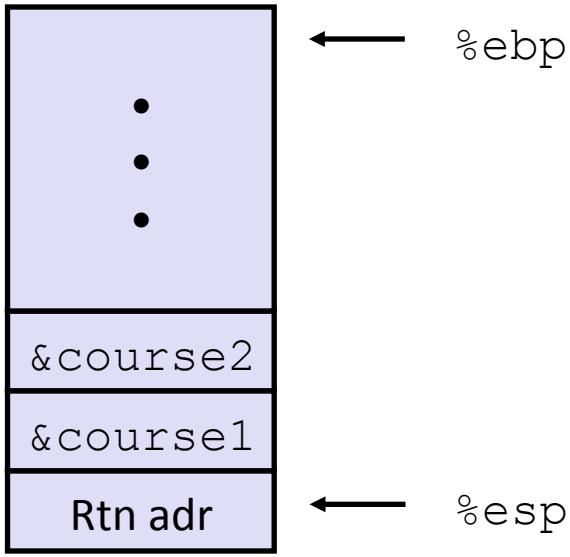
Resulting Stack



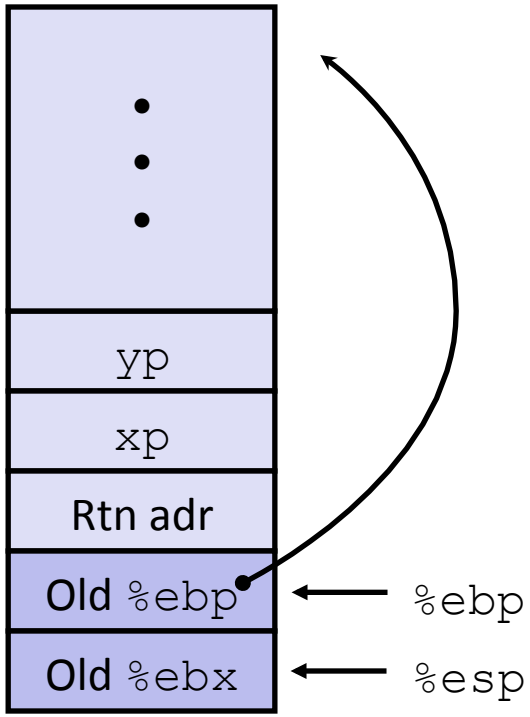

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

swap Setup #3

Entering Stack

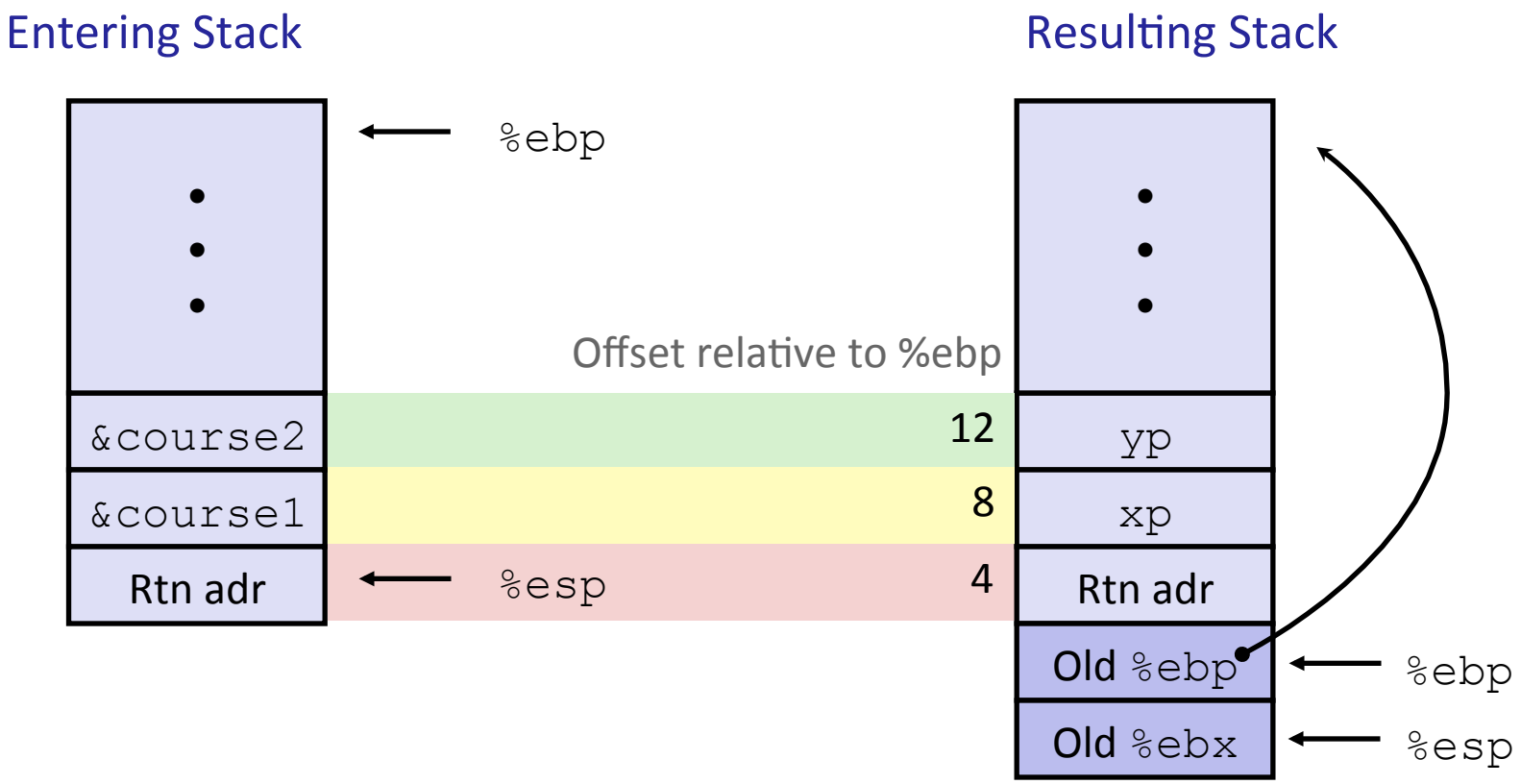


Resulting Stack




```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

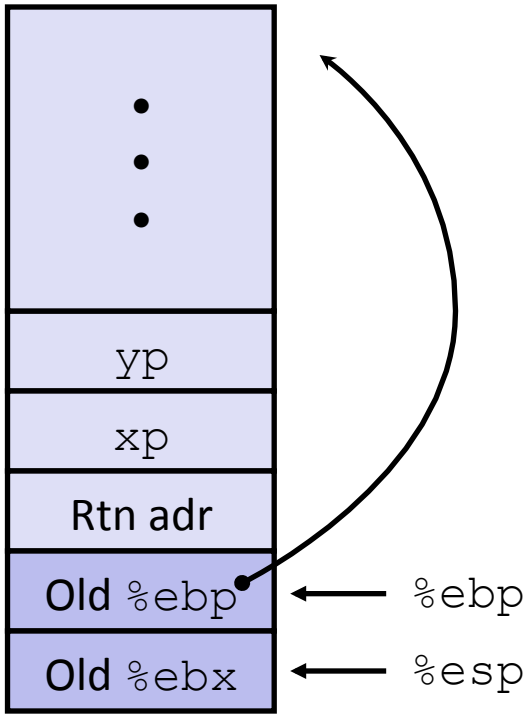
swap Body




```
movl 8(%ebp),%edx    # get xp
movl 12(%ebp),%ecx   # get yp
. . .
```

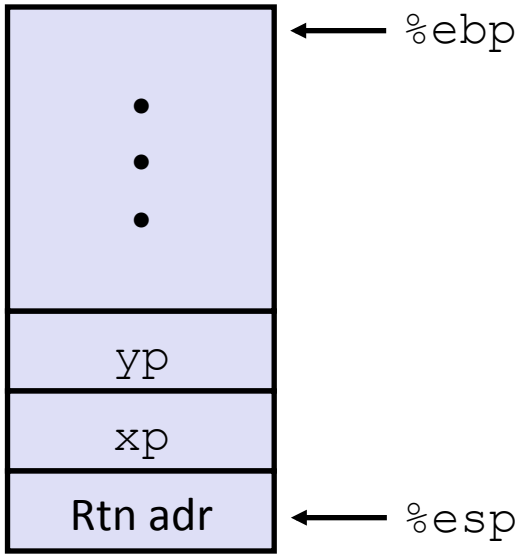
swap Finish

Stack Before Finish




```
popl    %ebx
popl    %ebp
```

Resulting Stack



- Observation
  - Saved and restored register `%ebx`
  - Not so for `%eax`, `%ecx`, `%edx`

Disassembled swap

```
08048384 <swap>:
 8048384:  55                    push   %ebp
 8048385:  89 e5                 mov    %esp,%ebp
 8048387:  53                    push   %ebx
 8048388:  8b 55 08              mov    0x8(%ebp),%edx
 804838b:  8b 4d 0c              mov    0xc(%ebp),%ecx
 804838e:  8b 1a                 mov    (%edx),%ebx
 8048390:  8b 01                 mov    (%ecx),%eax
 8048392:  89 02                 mov    %eax,(%edx)
 8048394:  89 19                 mov    %ebx,(%ecx)
 8048396:  5b                    pop    %ebx
 8048397:  5d                    pop    %ebp
 8048398:  c3                    ret
```

Calling Code

```
 80483b4:  movl   $0x8049658,0x4(%esp) # Copy &course2
 80483bc:  movl   $0x8049654,(%esp)    # Copy &course1
 80483c3:  call   8048384 <swap>       # Call swap
 80483c8:  leave                       # Prepare to return
 80483c9:  ret                         # Return
```