Announcements

- Exam 2
  - Due Friday, October 24
  - Covers Lectures 9-18.
- Assignment 4
  - Posted at 5 p.m. today
  - Due 5 p.m., Monday, October 27
- Graded lab next week

Lecture 18

Control: Procedures

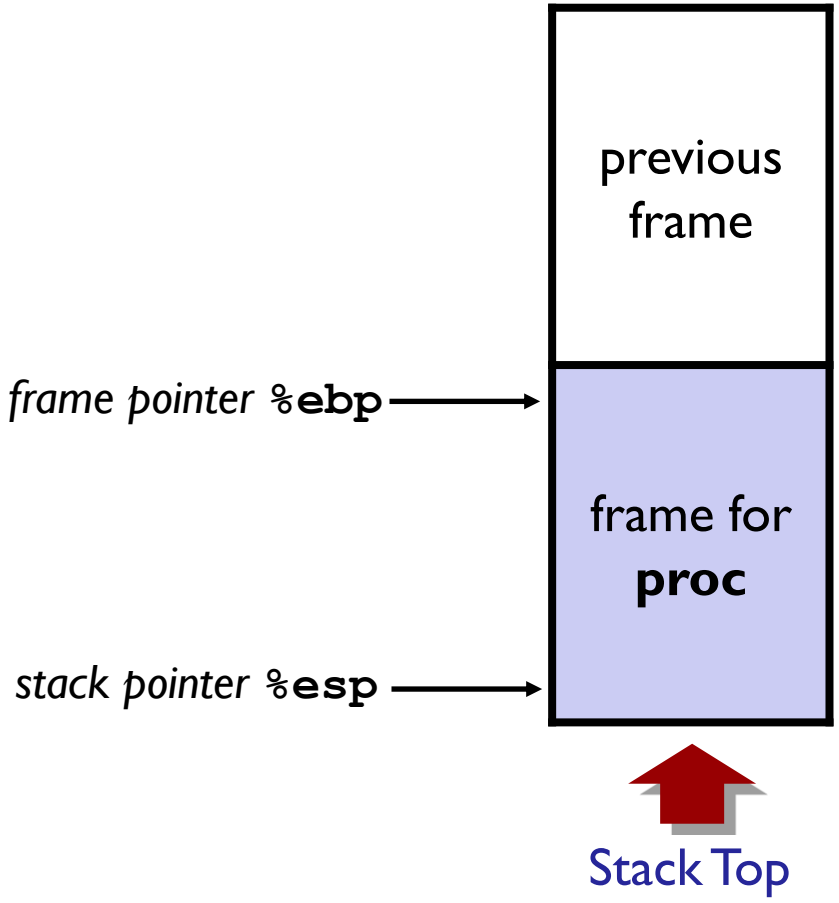
CPSC 275
Introduction to Computer Systems

Stack-Based Languages

- Pascal, C, Java, etc.
- Need some place to store state of each instantiation (or *activation*), including
  - arguments
  - local variables
  - return address
- Stack allocated in *frames*
  - state of single procedure instantiation

Stack Frames

- Contents
  - local variables
  - return information
  - temporary space
- Management
  - Space allocated when enter procedure
    - "Set-up" code
  - Deallocated when return
    - "Finish" code



Call Chain Example

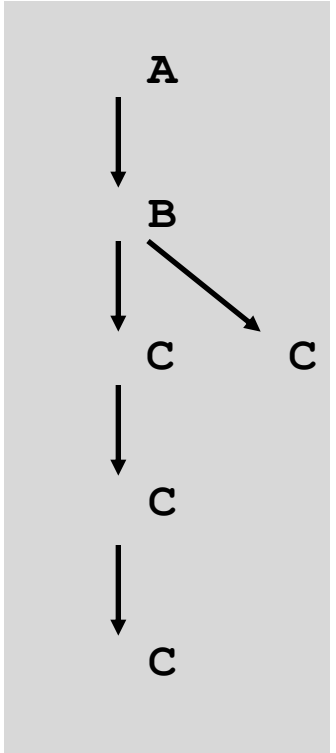
```
A(…)
{
   •
   •
   B();
   •
   •
}
```

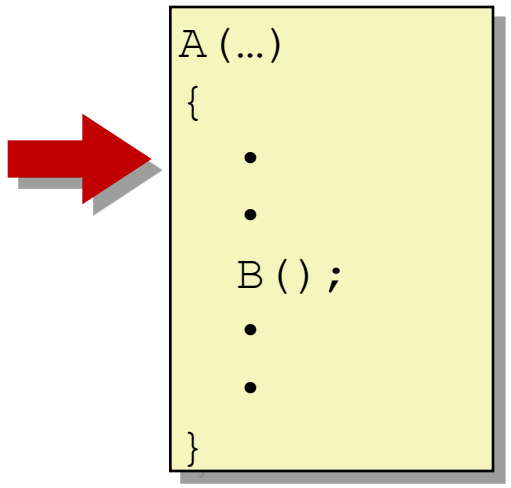
```
B(…)
{
   • • •
   C();
   • • •
   C();
   • • •
}
```

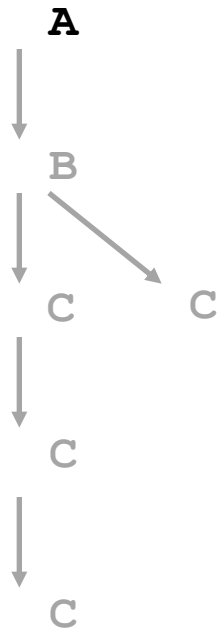
```
C(…)
{
   •
   •
   C();
   •
   •
}
```

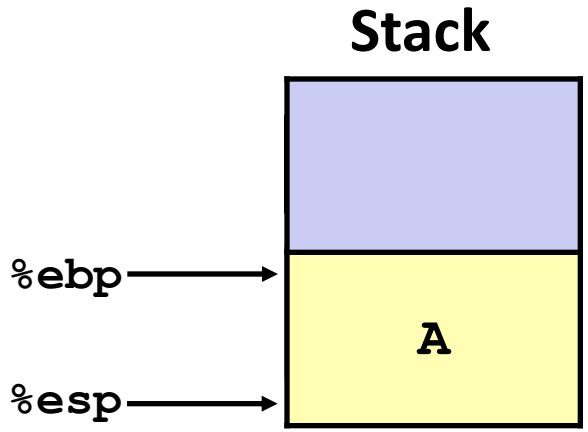
Procedure C() is recursive

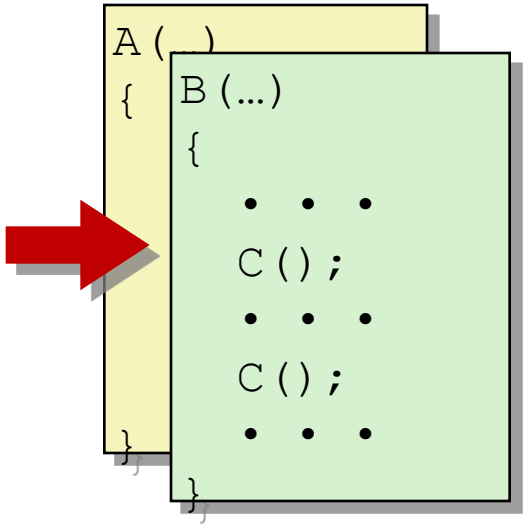
Example Call Chain

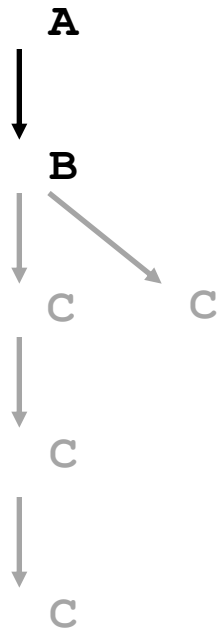


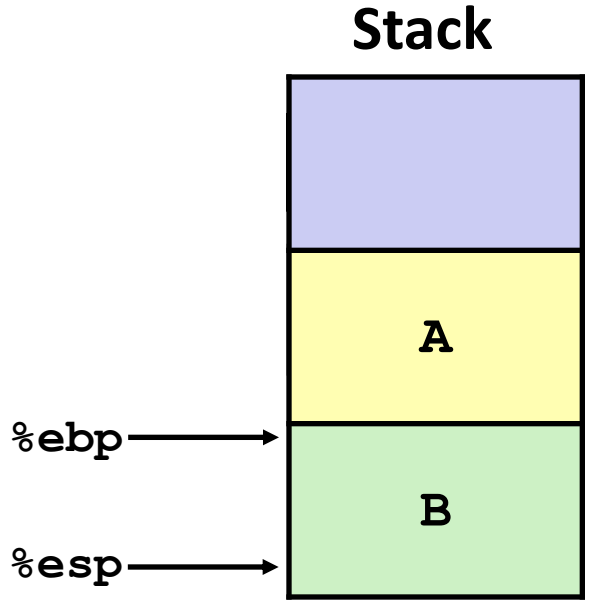

A(…)
{
•
•
B();
•
•
}

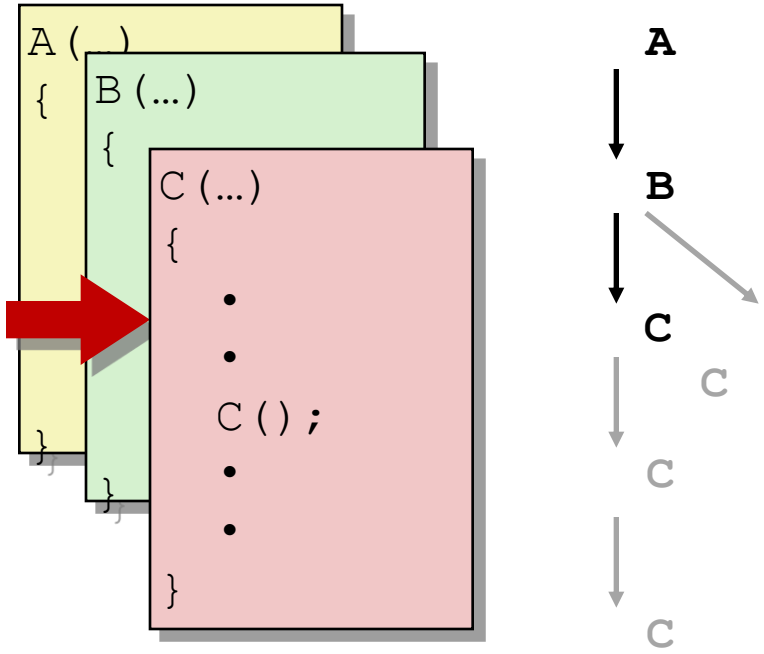

A
B
C
C
C
C


Stack
%ebp
%esp
A

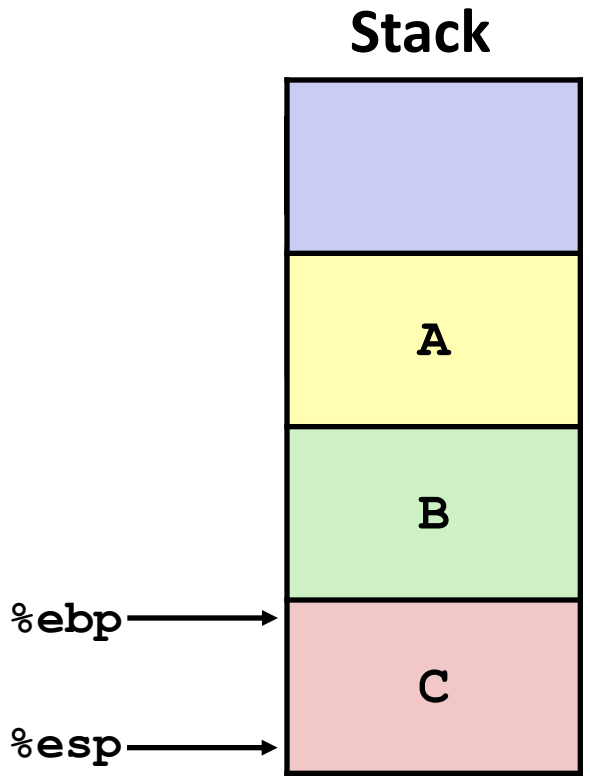

A(…)
{
B(…)
{
C();
C();
}

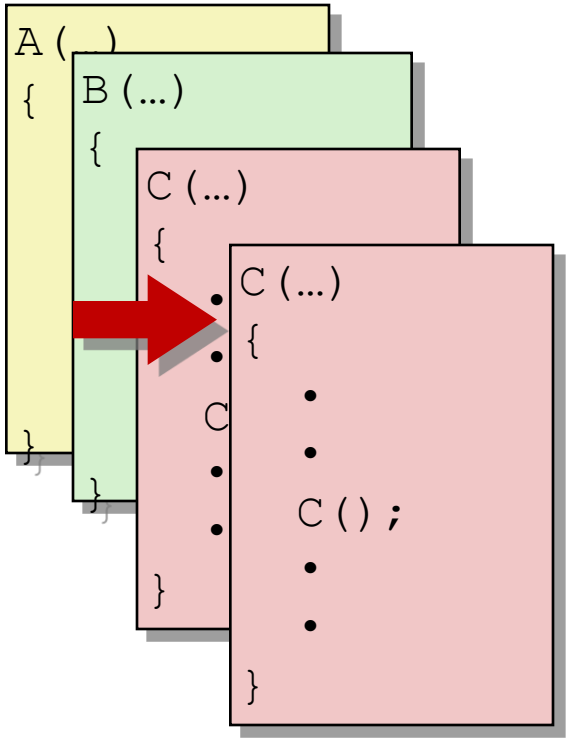

A
B
C
C
C
C

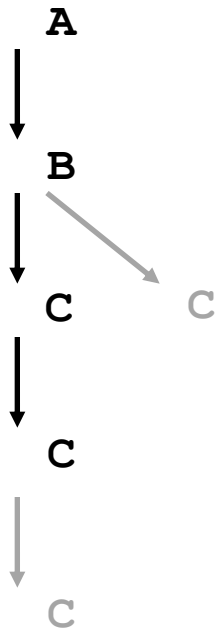

Stack
A
B
%ebp
%esp

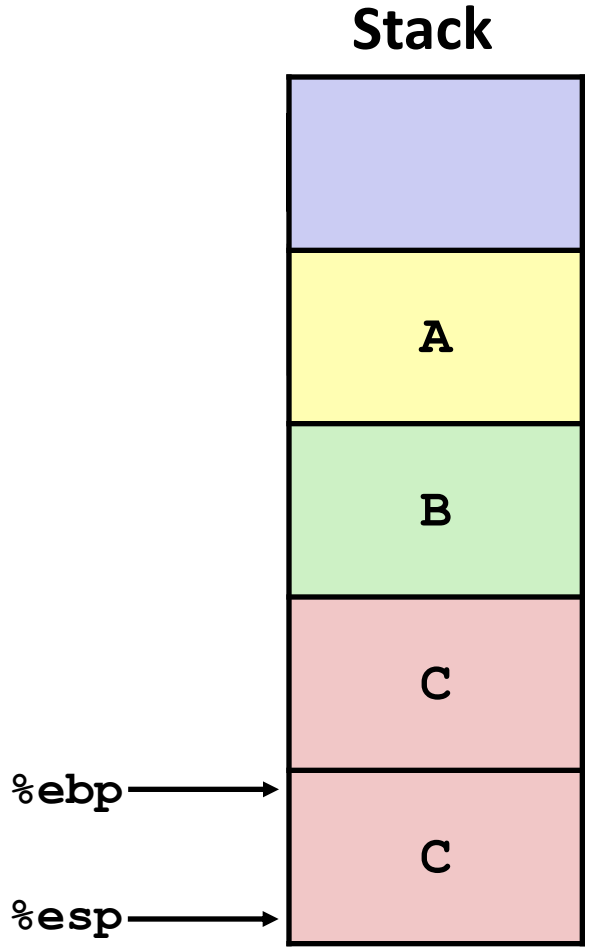

A ( )
{
B (…)
{
C (…)
{
C ( ) ;
}

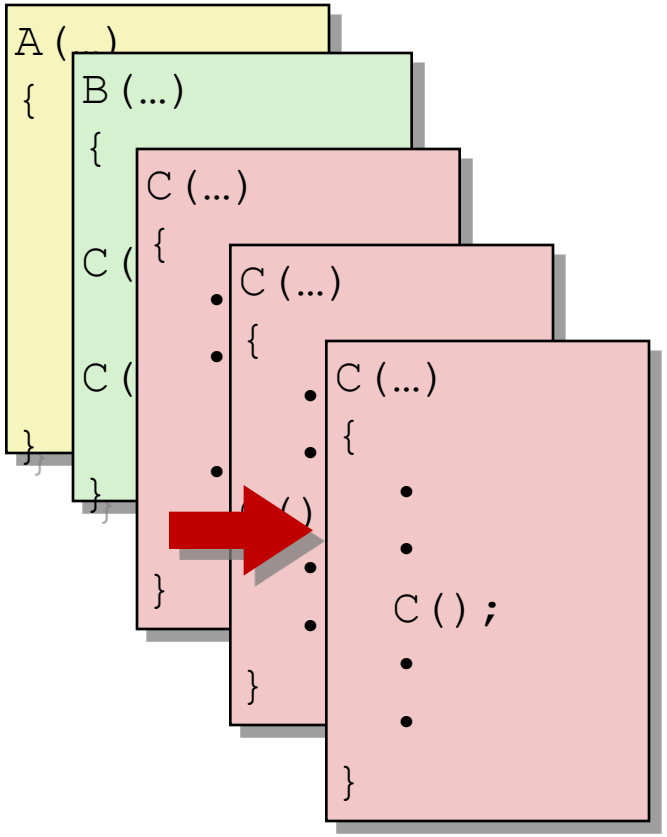
A
B
C
C
C
C

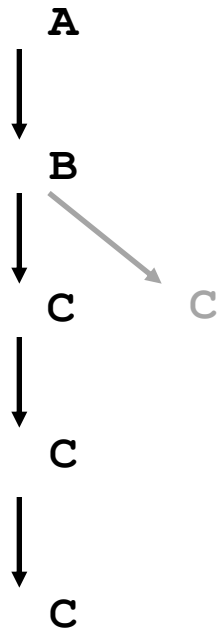

Stack
A
B
C
%ebp
%esp

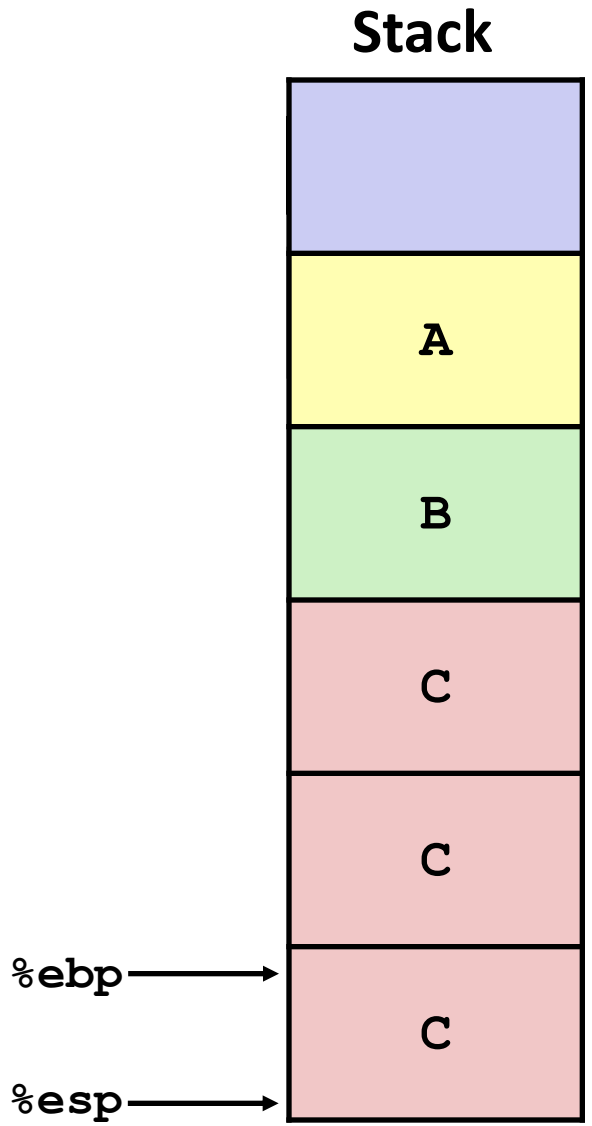

A (...)
{
}
B (...)
{
}
C (...)
{
C
}
C (...)
{
C();
}

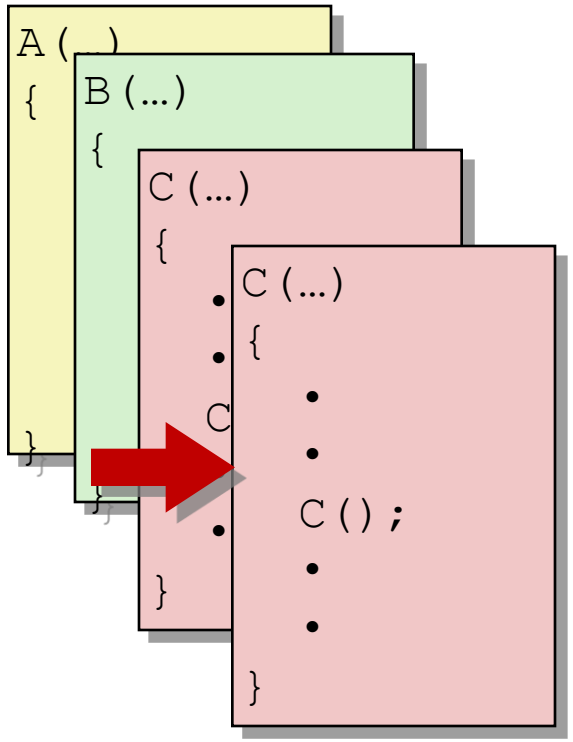

A
B
C
C
C
C

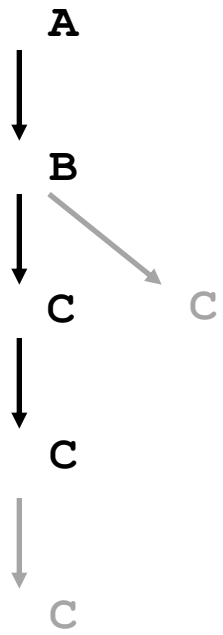

Stack
A
B
C
C
%ebp
%esp

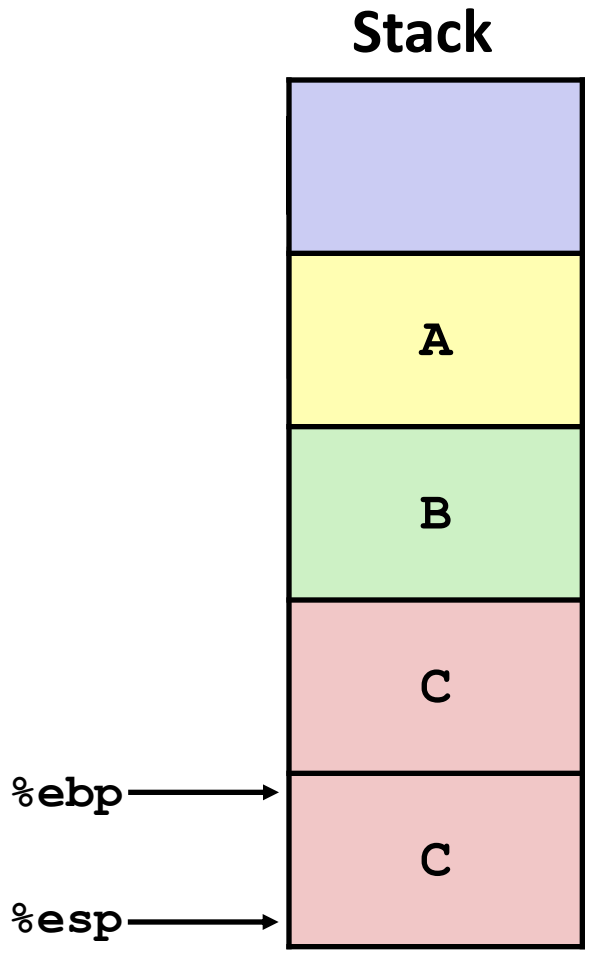

A ( )
{
B (...)
{
C (...)
{
C (...)
{
C (...)
{
C ();
}

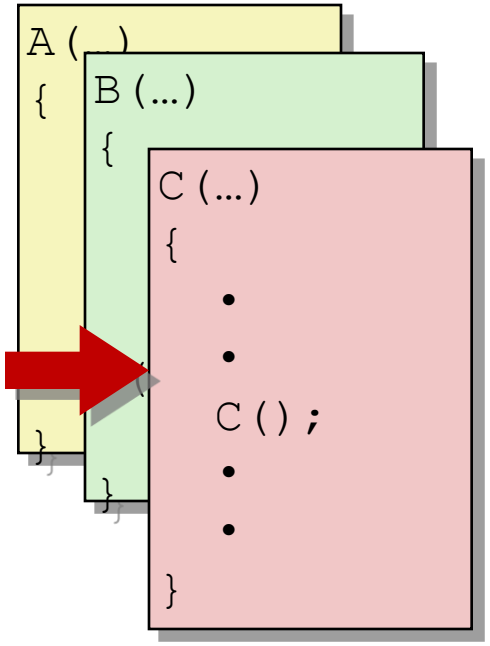

A
B
C
C
C
C

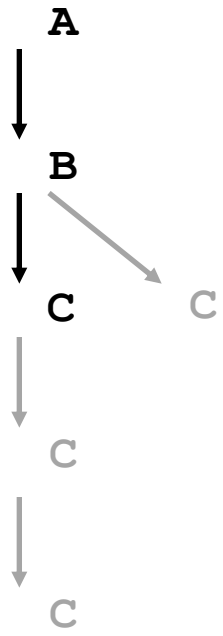

Stack
A
B
C
C
C
%ebp
%esp

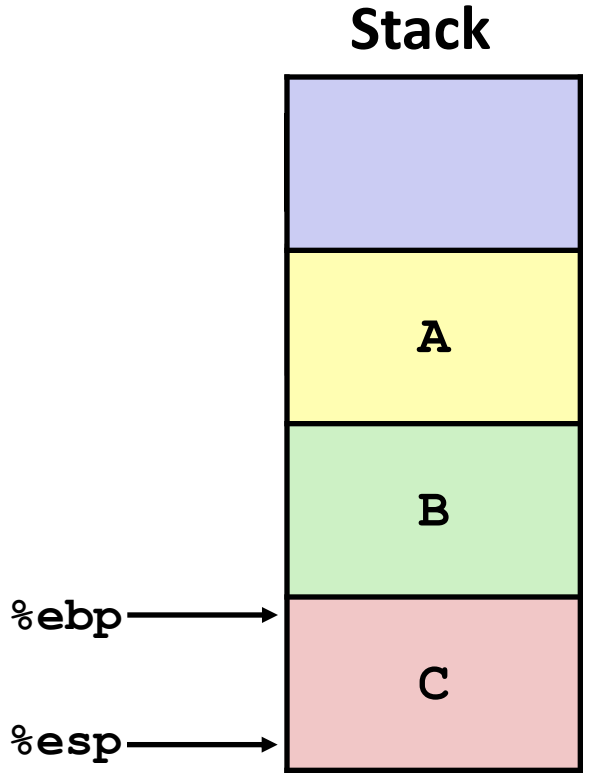

A (...)
{
B (...)
{
C (...)
{
C (...)
{
C ();
}

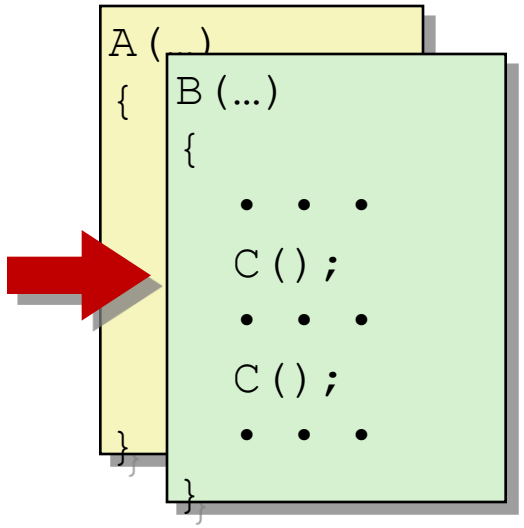

A
B
C
C
C
C

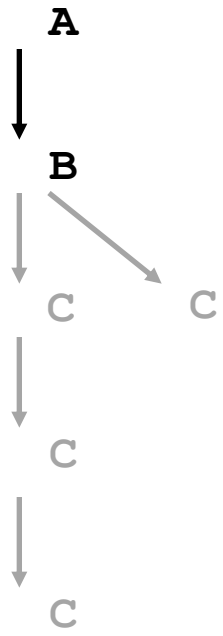

Stack
A
B
C
C
%ebp
%esp

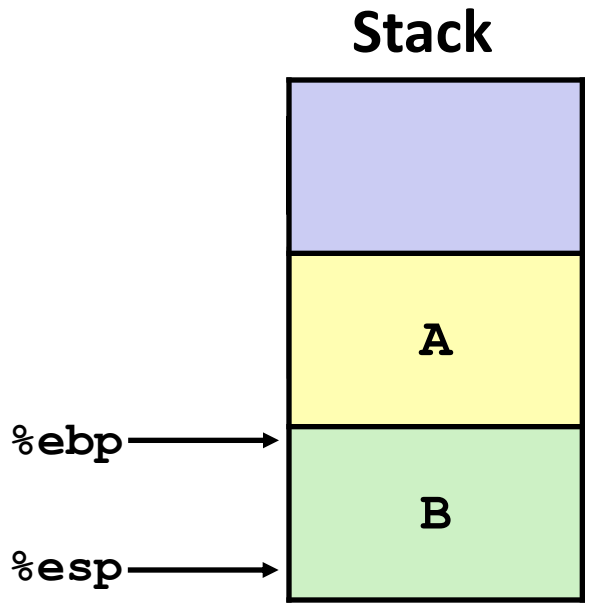

A(...)
{
B(...)
{
C(...)
{
C();
}
}
}

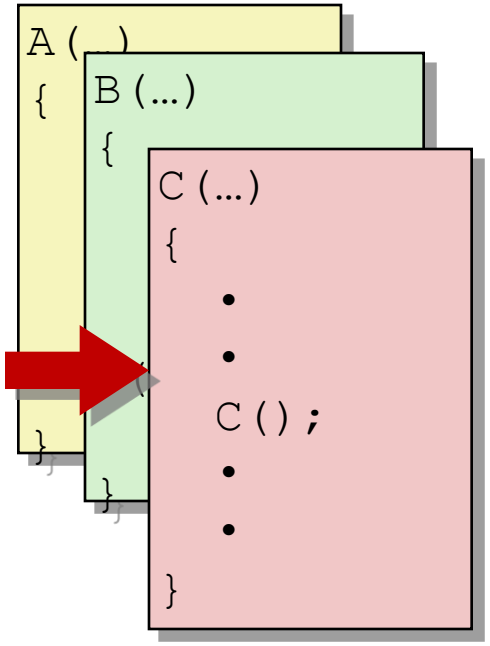

A
B
C
C
C
C

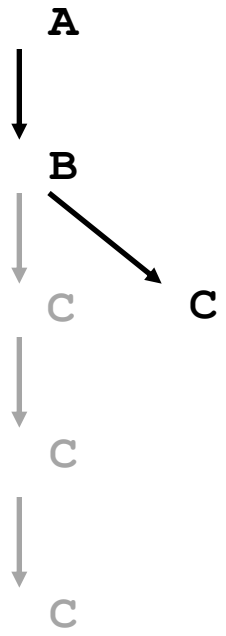

Stack
A
B
C
%ebp
%esp

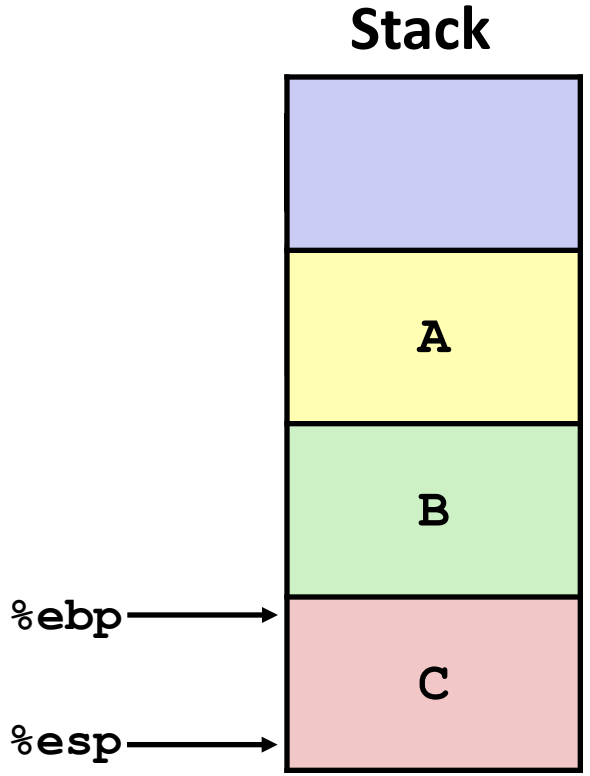

A(…)
{
B(…)
{
C();
C();
}

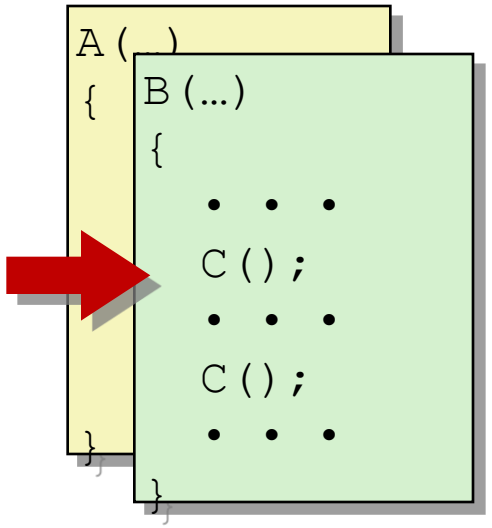

A
B
C
C
C
C

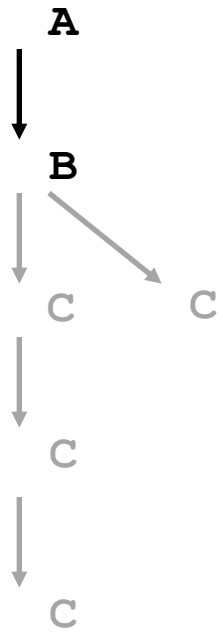

Stack
A
B
%ebp
%esp

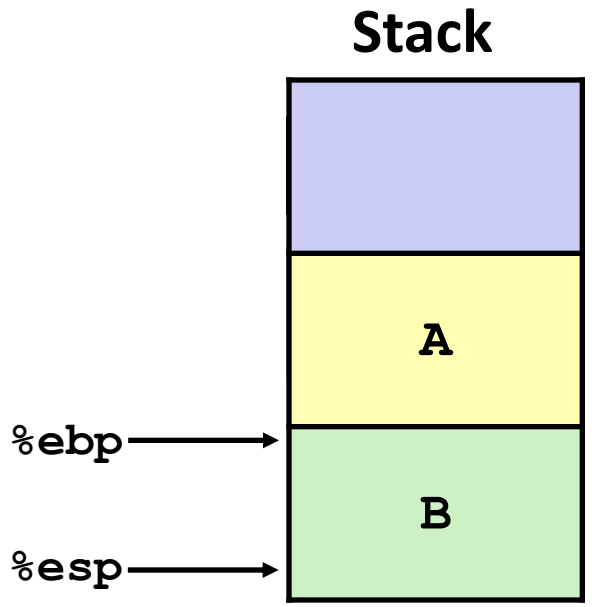

A(...)
{
B(...)
{
C(...)
{
C();
}
}
}

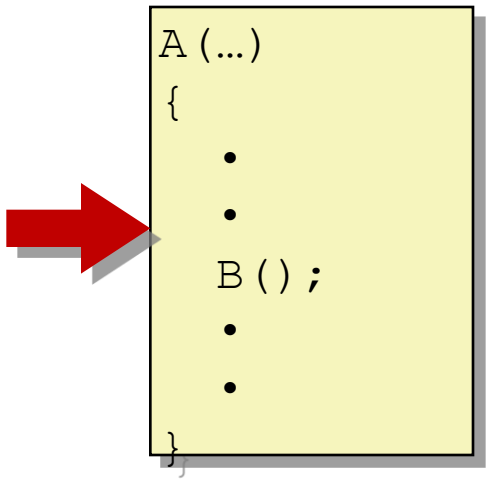

A
B
C
C
C

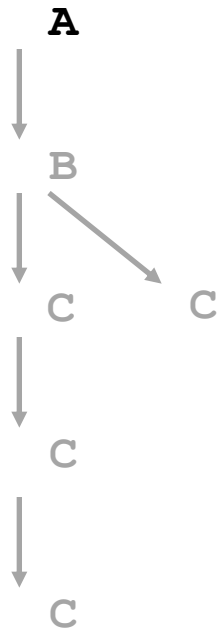

Stack
A
B
C
%ebp
%esp

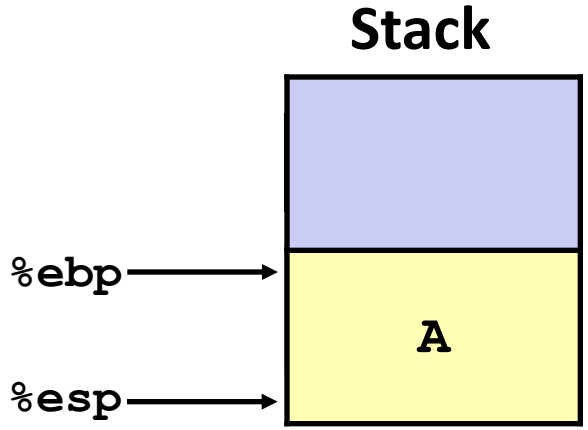

A(...)
{
B(...)
{
• • •
C();
• • •
C();
• • •
}
}


A
B
C
C
C
C


Stack
A
B
%ebp
%esp

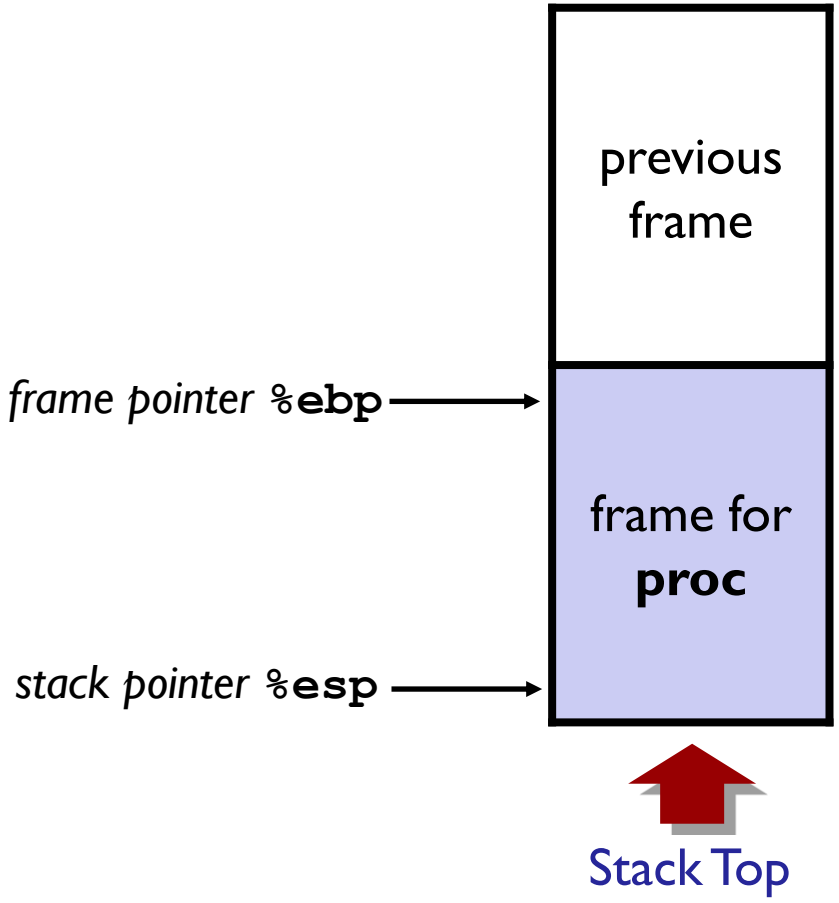

A(…)
{
•
•
B();
•
•
}


A
B
C
C
C
C


Stack
%ebp
%esp
A

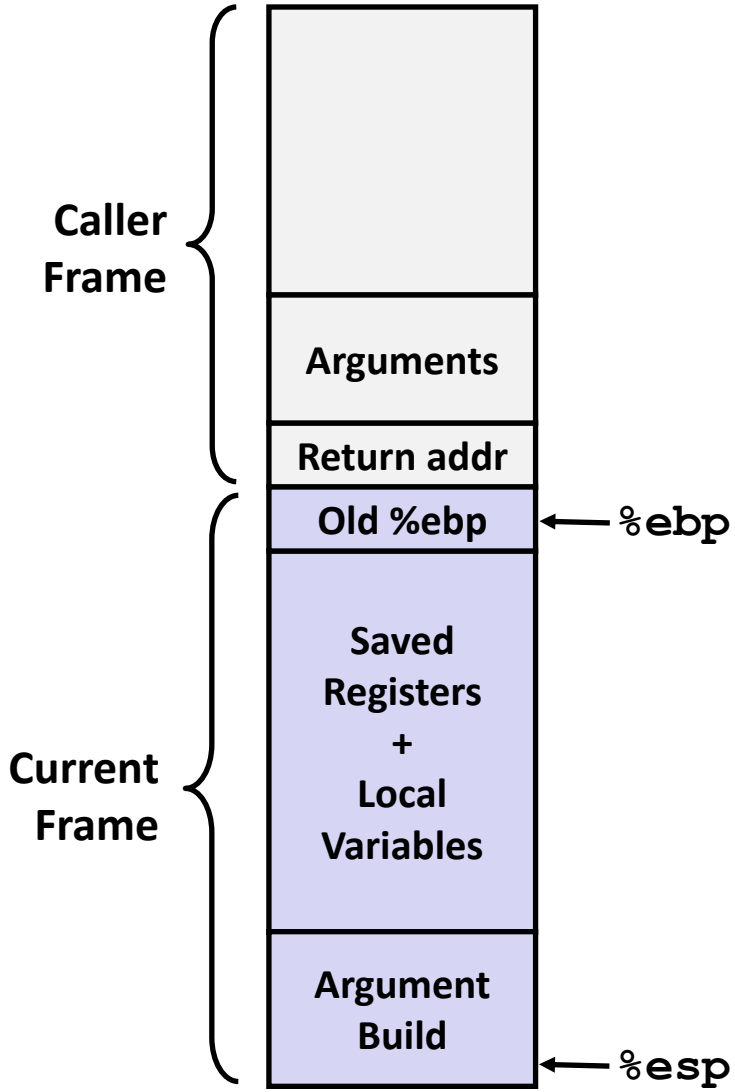
Stack Frames

- Contents
  - local variables
  - return information
  - temporary space
- Management
  - Space allocated when enter procedure
    - "Set-up" code
  - Deallocated when return
    - "Finish" code



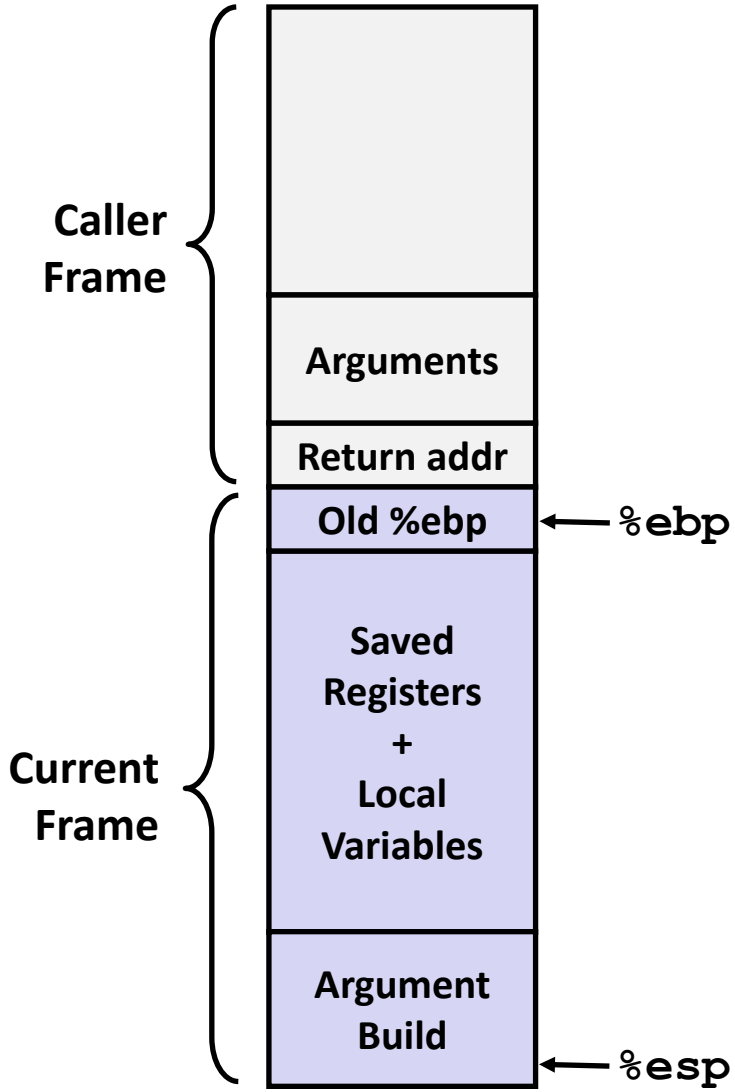
IA-32/Linux Stack Frame

- Current stack frame (top to bottom)
  - parameters for function about to call (*argument build*)
  - 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



IA-32/Linux Stack Frame

- Current stack frame (top to bottom)
  - arguments for the function about to call (*argument build*)
  - 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



The swap function

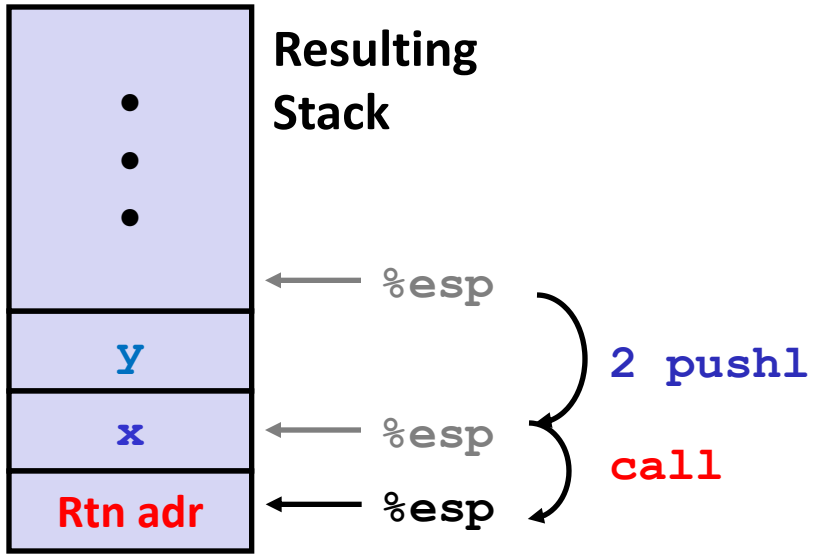
```
int x = 15213; // global var
int y = 18243;

void f() {
  swap(&x, &y);
}
```

Calling swap from f()

```
f:
    . . .
    pushl   $y
    pushl   $x
    call    swap
    . . .
```

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



The **swap** function

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

Set Up

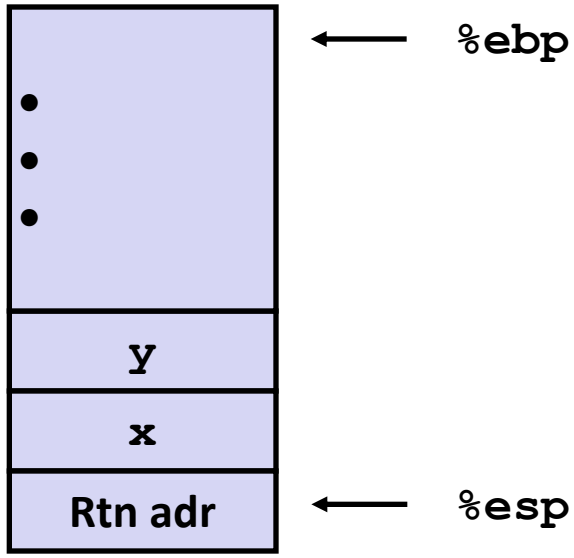
Body

Finish

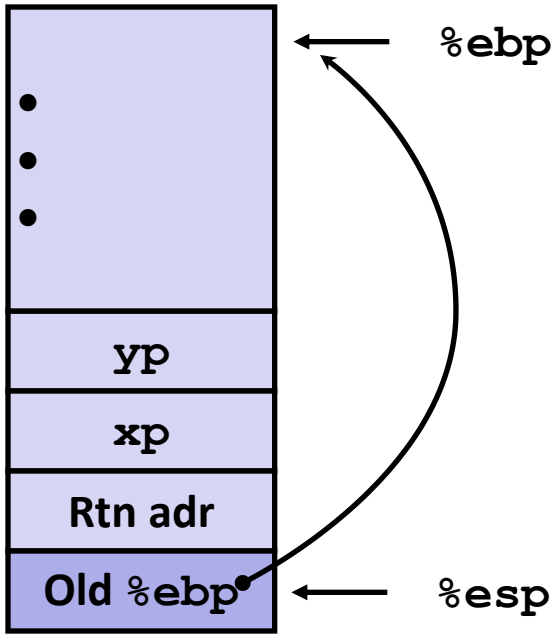
→ will be explained shortly.

swap Setup #1

Entering Stack



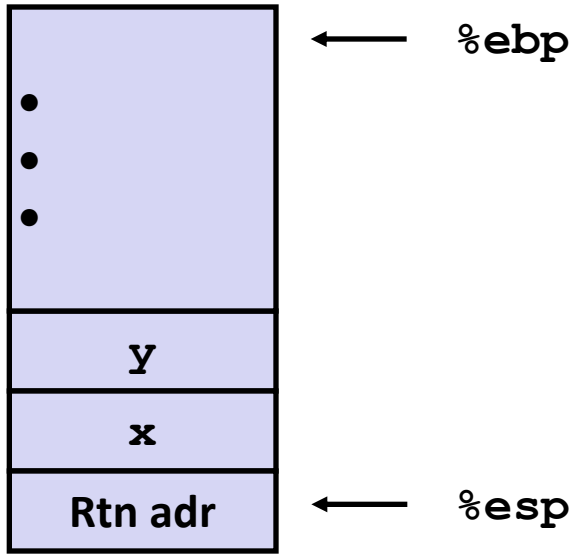
Resulting Stack



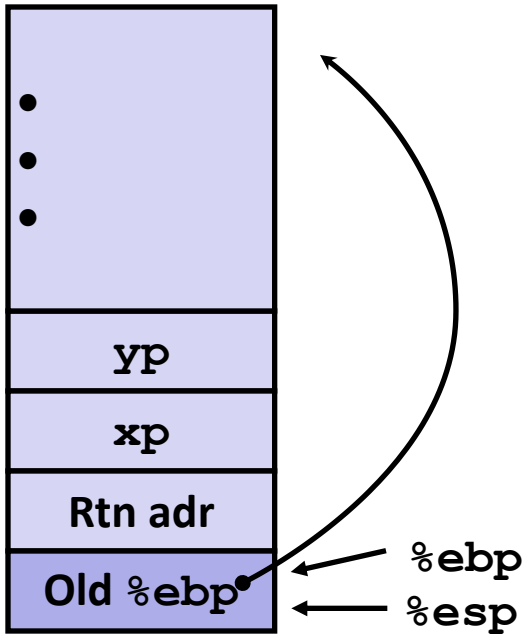

```
swap:
    pushl   %ebp
    movl    %esp,%ebp
    pushl   %ebx    #  to be explained
```

swap Setup #2

Entering Stack



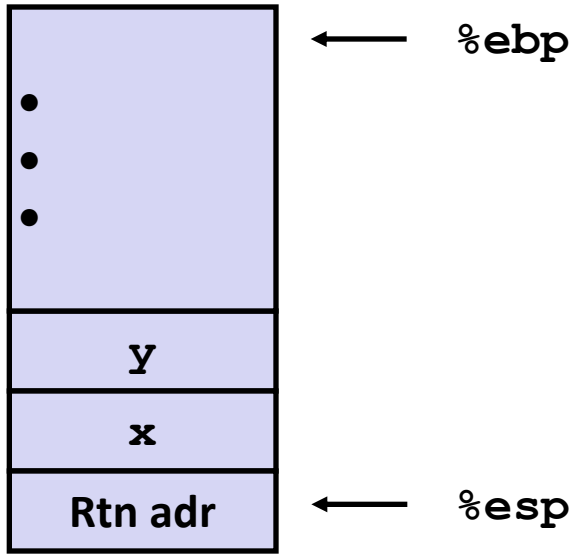
Resulting Stack



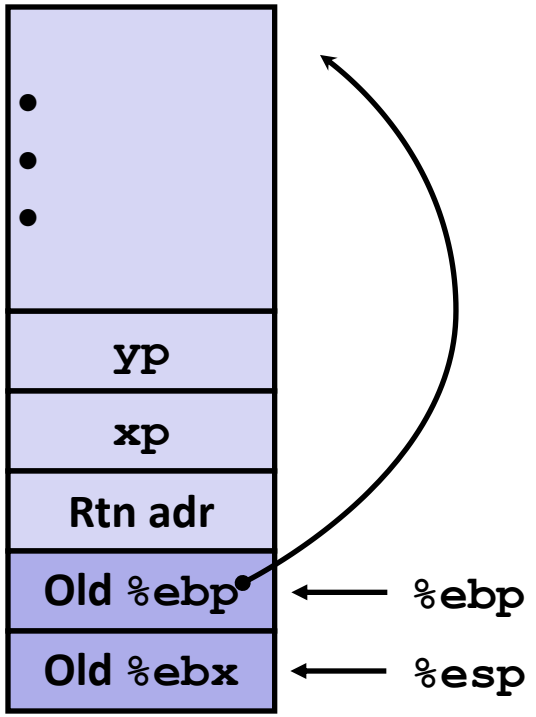

```
swap:
    pushl   %ebp
    movl    %esp,%ebp
    pushl   %ebx    #  to be explained
```

swap Setup #3

Entering Stack

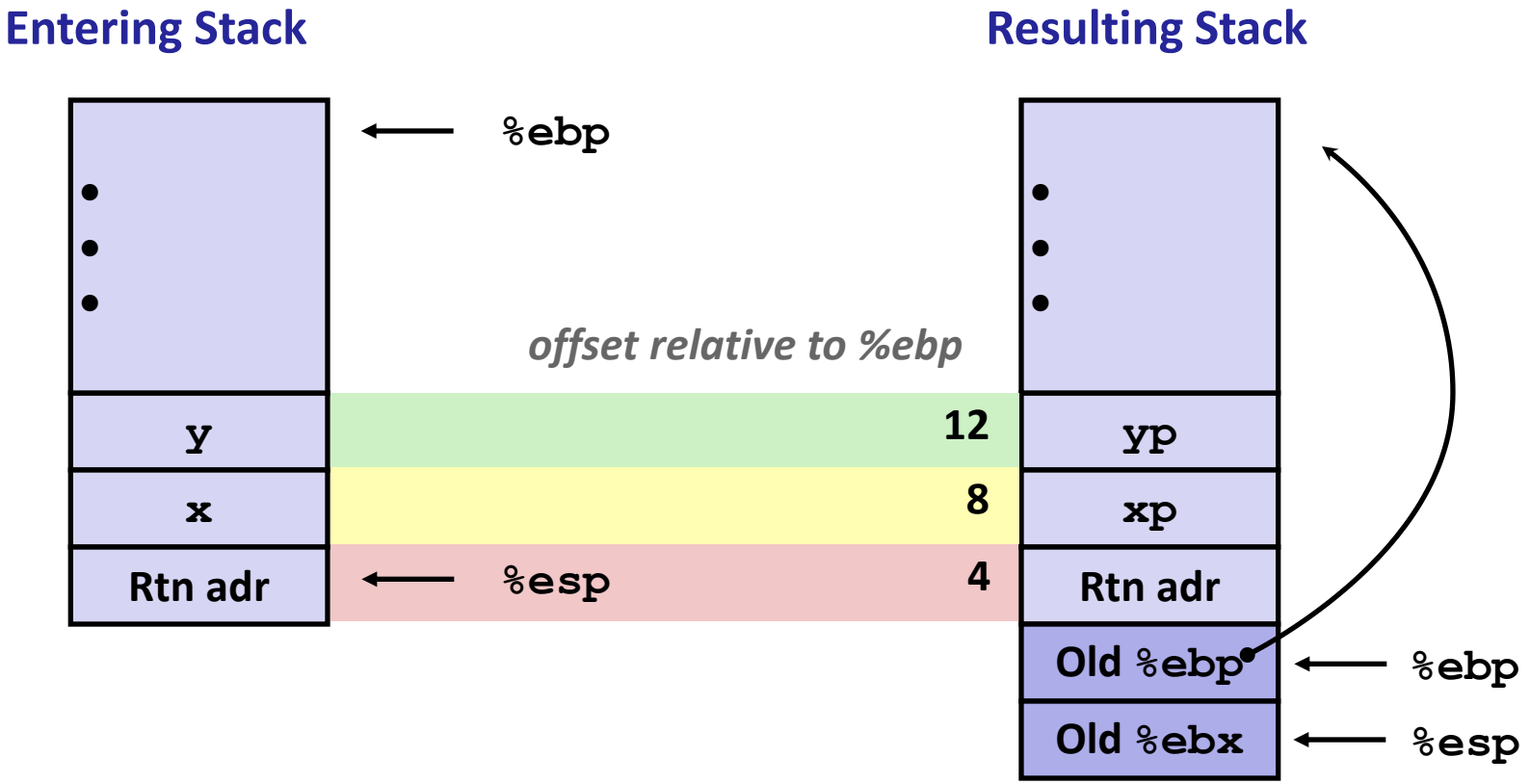


Resulting Stack



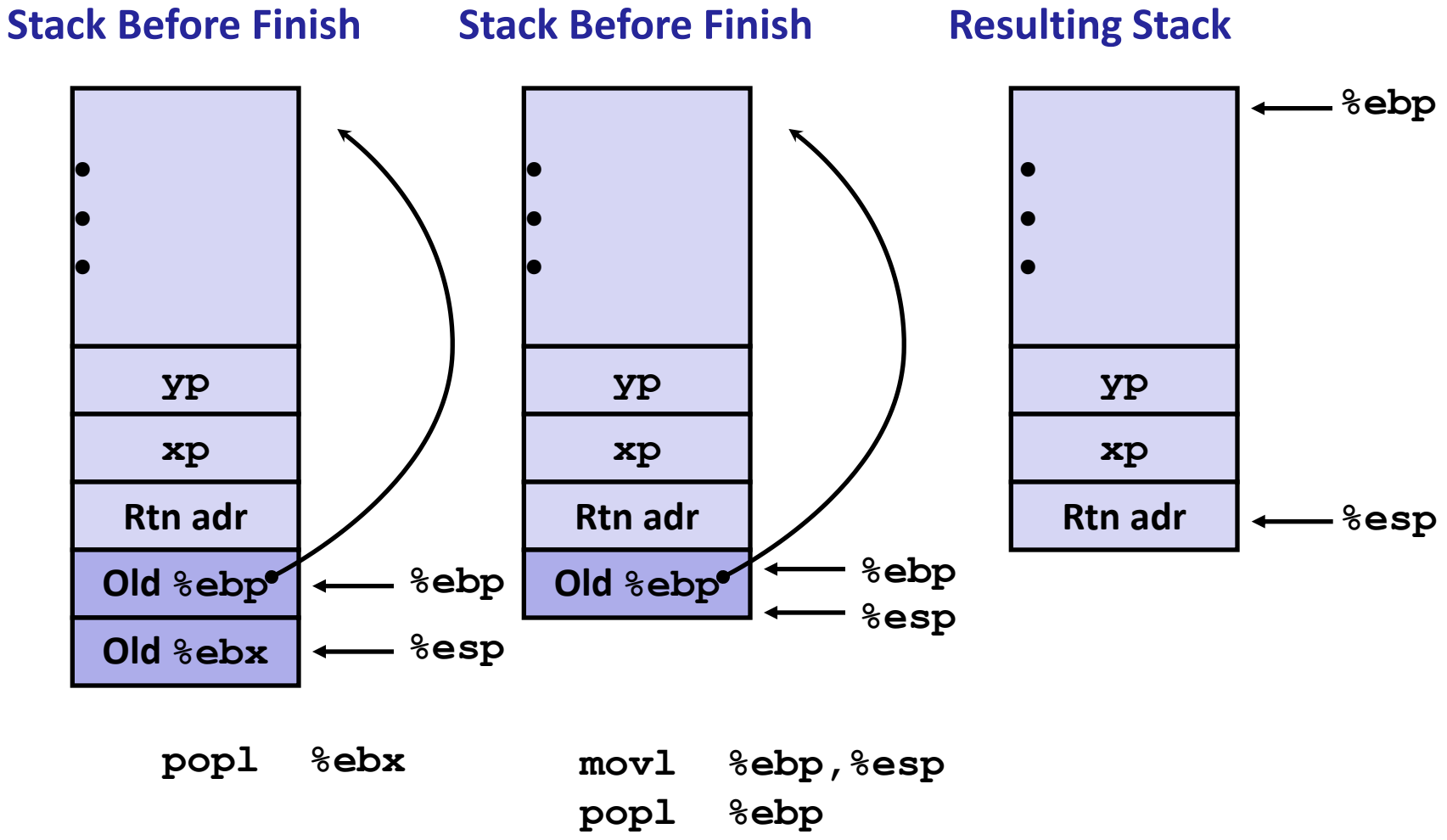

```
swap:
    pushl   %ebp
    movl    %esp,%ebp
    pushl   %ebx    #  to be explained
```

swap Body




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

swap Finish


Stack Before Finish
Stack Before Finish
Resulting Stack
yp
xp
Rtn adr
Old %ebp
Old %ebx
%ebp
%esp
yp
xp
Rtn adr
Old %ebp
%ebp
%esp
yp
xp
Rtn adr
%ebp
%esp
popl %ebx
movl %ebp,%esp
popl %ebp

`swap` Finish

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

- **`leave`** instruction may replace:

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

Register saving conventions

- When procedure **A** calls **B**:
  - **A** is the ***caller***
  - **B** is the ***callee***
- Q: Can a register be used for temporary storage?

```
A:
    • • •
    movl      $15213,%edx
    call B
    addl      %edx,%eax
    • • •
```

```
B:
    • • •
    movl      8(%ebp),%edx
    incl      %edx
    • • •
    ret
```

  - Contents of register **%edx** overwritten by **B**
  - This could be trouble → something should be done!

Register saving conventions

- Conventions
  - ***"Caller Save"***

    - Caller saves temporary values in its frame before the call

  - ***"Callee Save"***

    - Callee saves temporary values in its frame before using

IA-32 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 it wants to use them
- **`%esp`, `%ebp`**
  - special form of callee save
  - Restored to original values upon exit from procedure

Exam 2 will cover up to here.