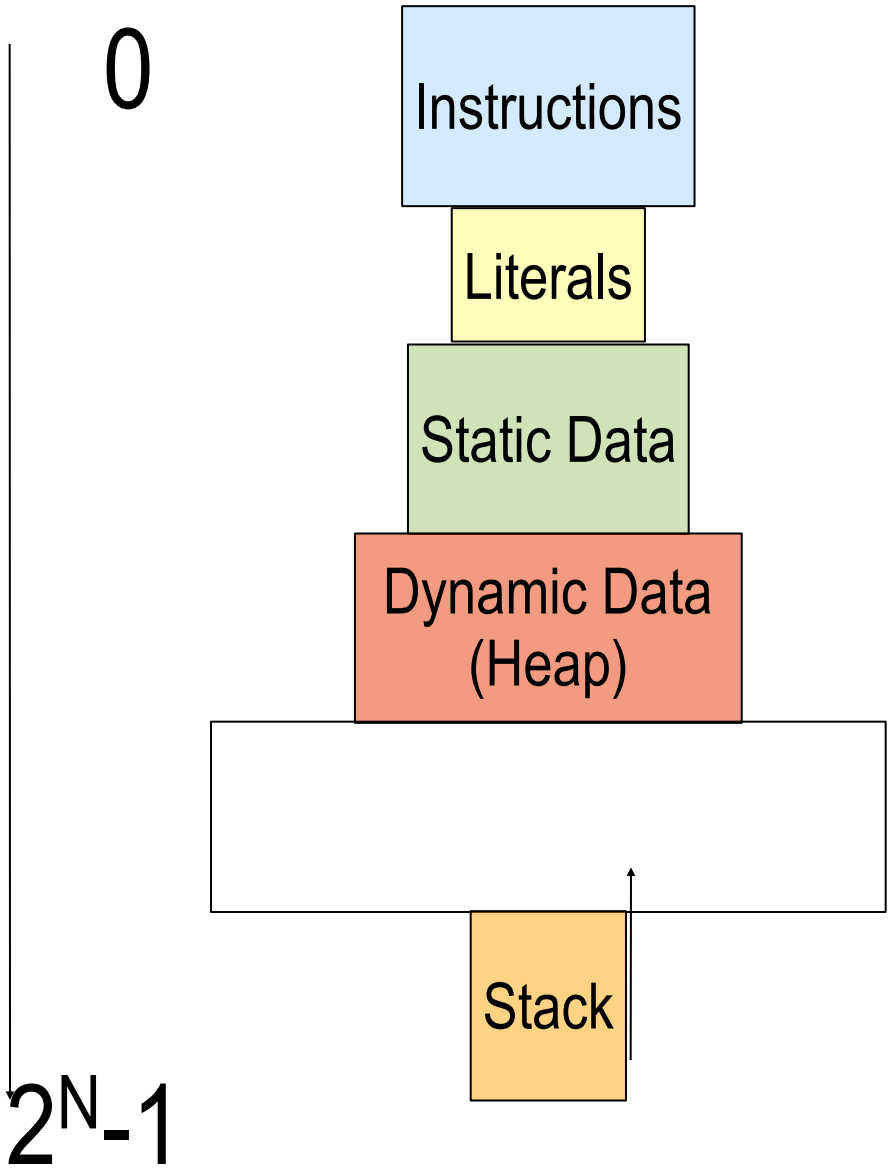
Procedures/Stacks

- Stacks
- Procedures
- Parameter passing

What is the stack for?

- Why a stack?

Memory Layout



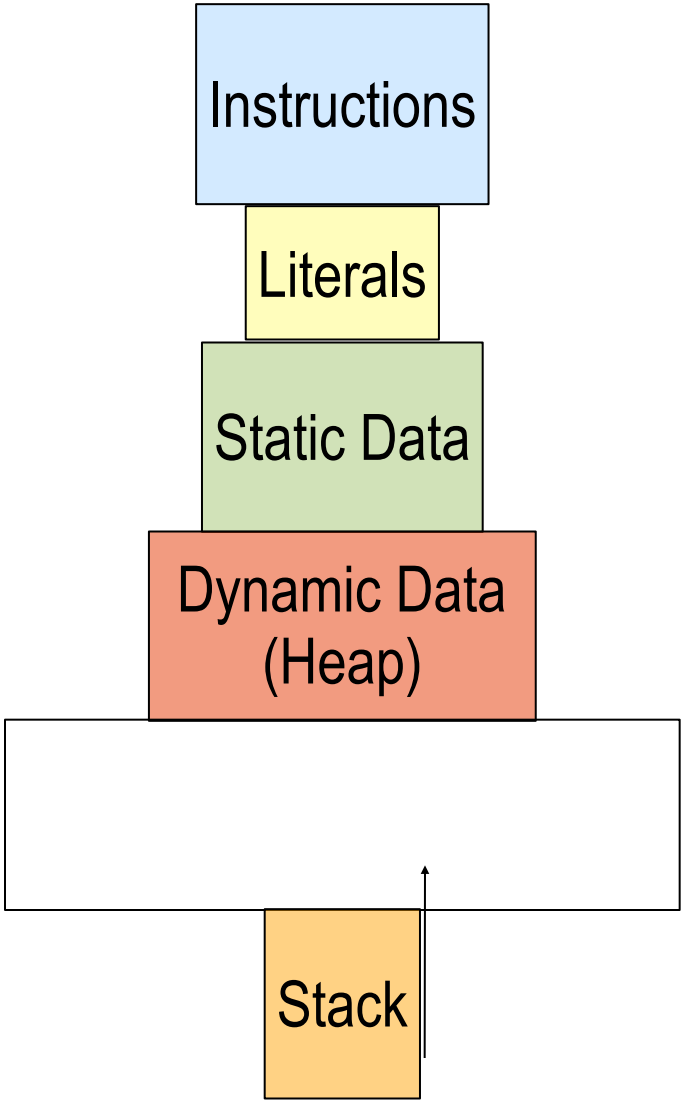
literals (e.g., “example”)

static variables
(including global variables (C))

new'ed variables

local variables

Memory Layout



Read-only; executable

Execution lifetime; immutable

Read-only; not executable

Execution lifetime; immutable

writable; not executable

Execution lifetime; mutable

writable; not executable

Programmer controlled lifetime; mutable

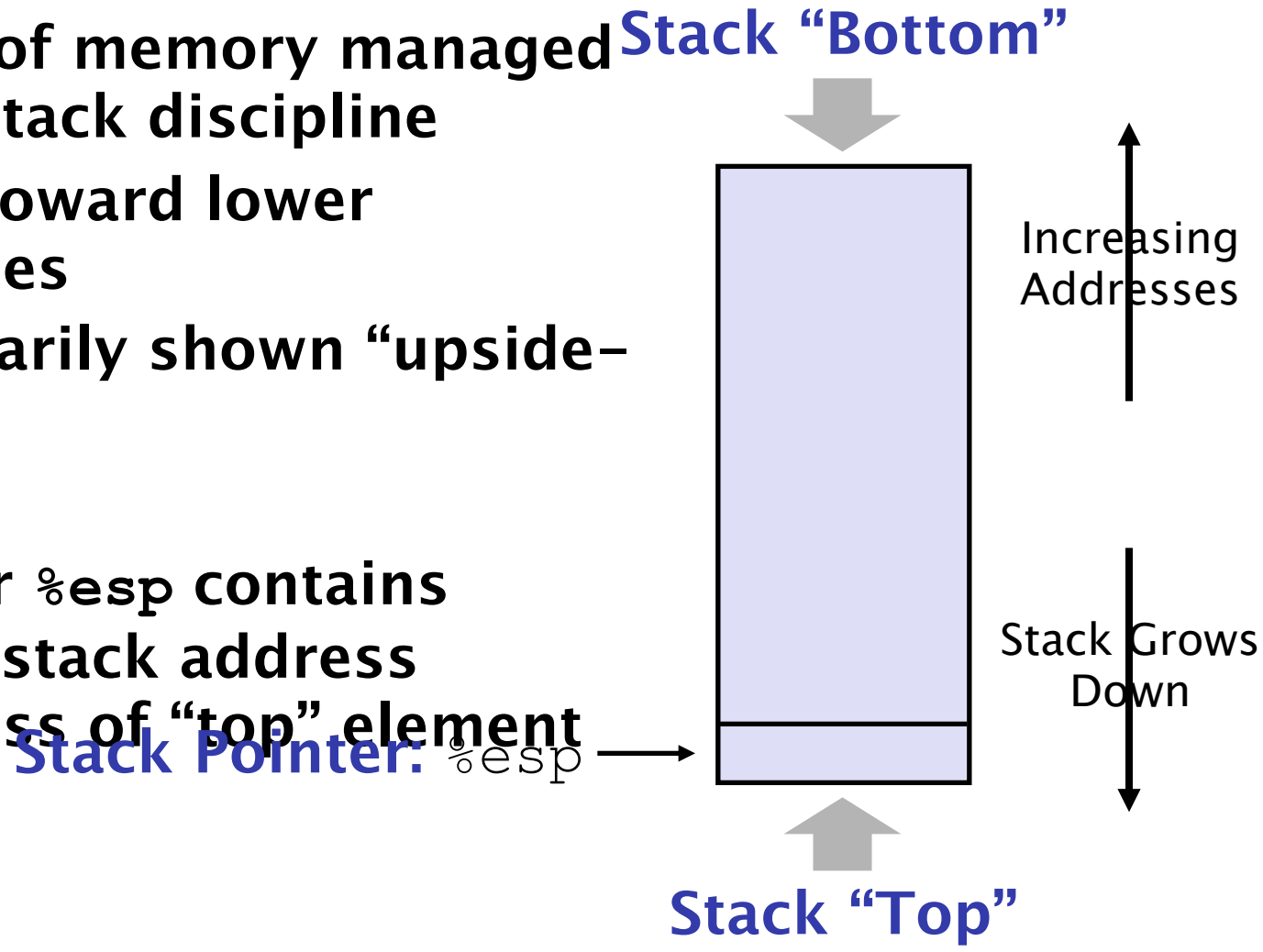
writable; not executable

"Automatic" lifetime; mutable

IA32 Stack

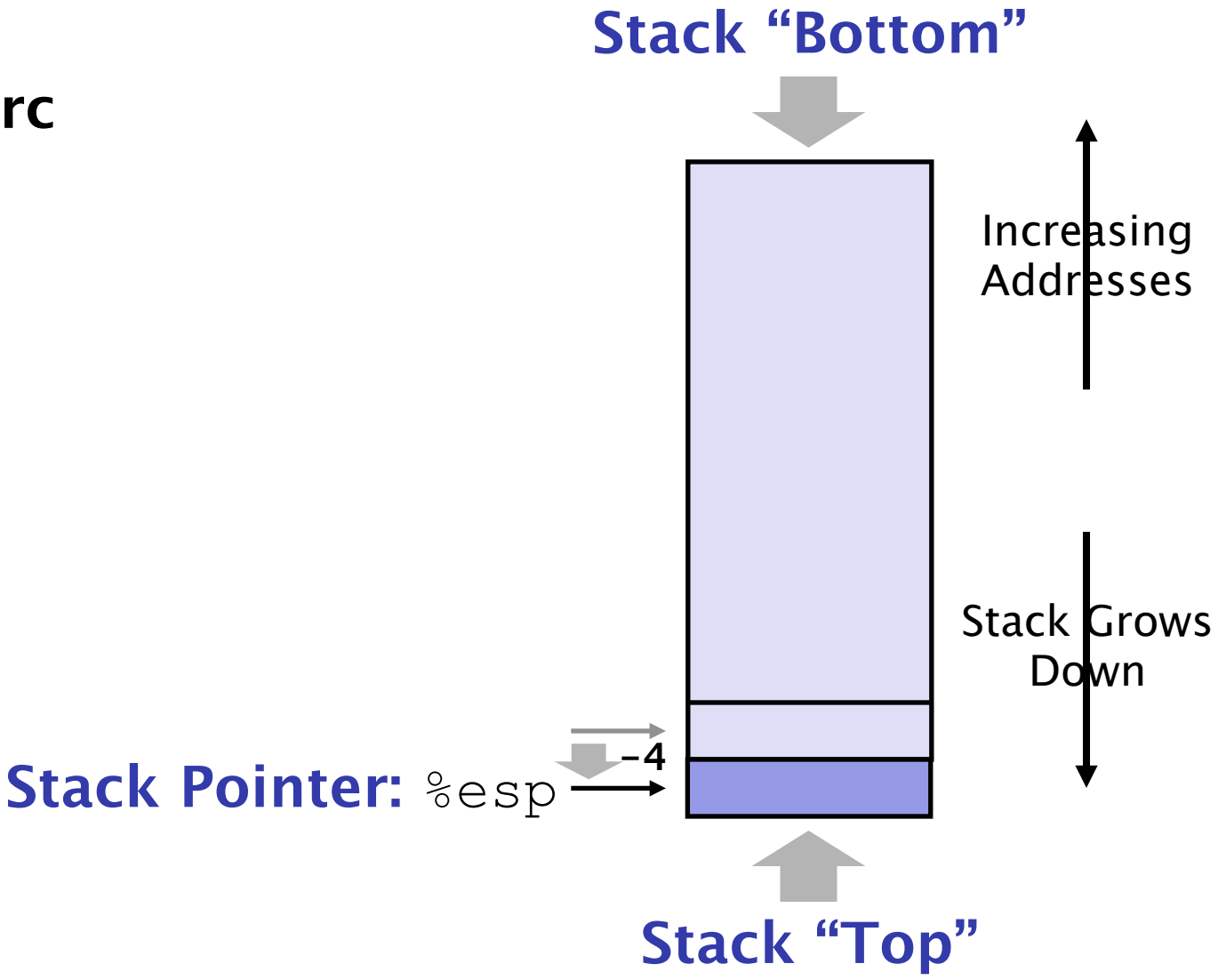
- Region of memory managed with a stack discipline
- Grows toward lower addresses
- Customarily shown "upside-down"

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



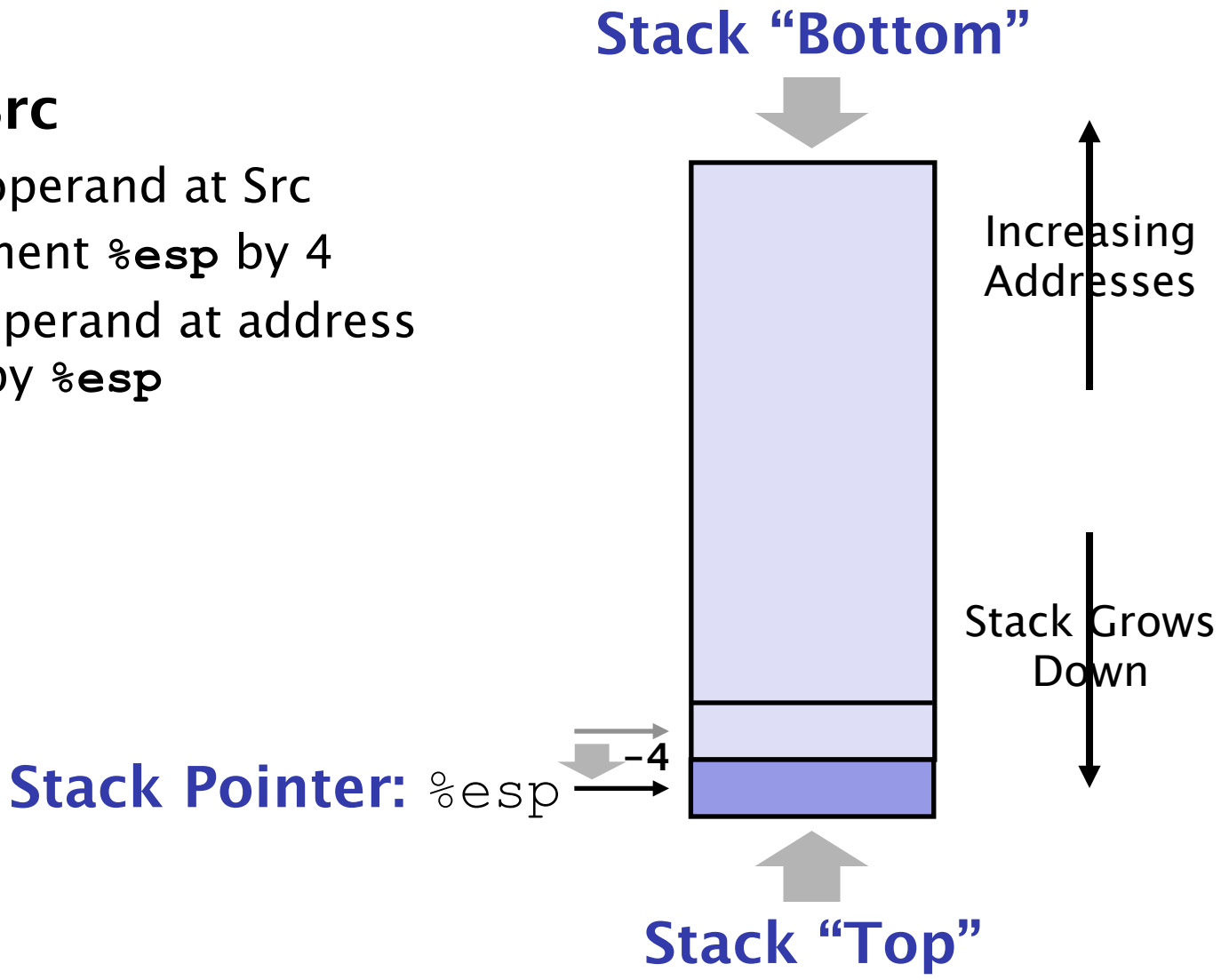
IA32 Stack: Push

- `pushl Src`

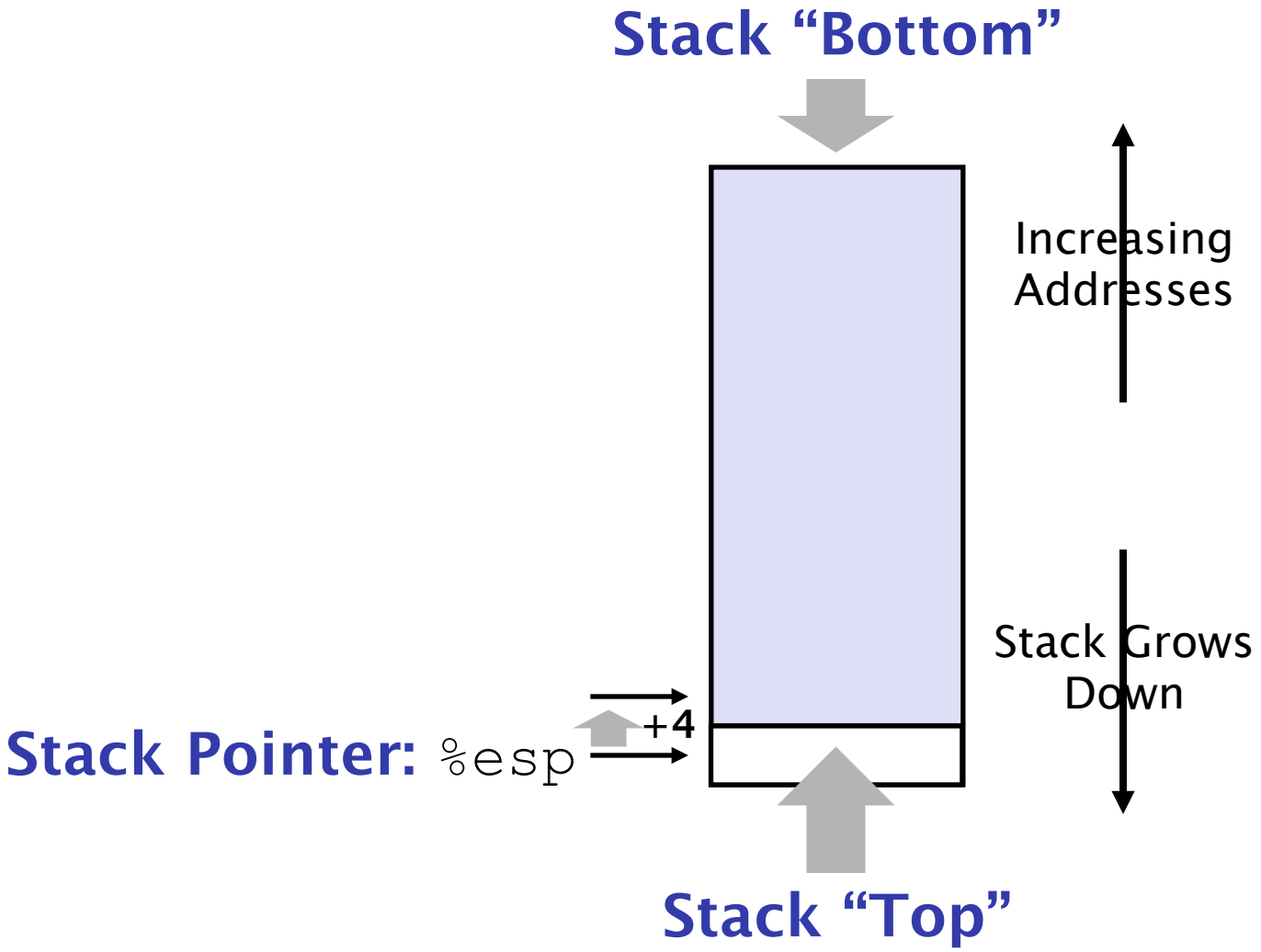


IA32 Stack: Push

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

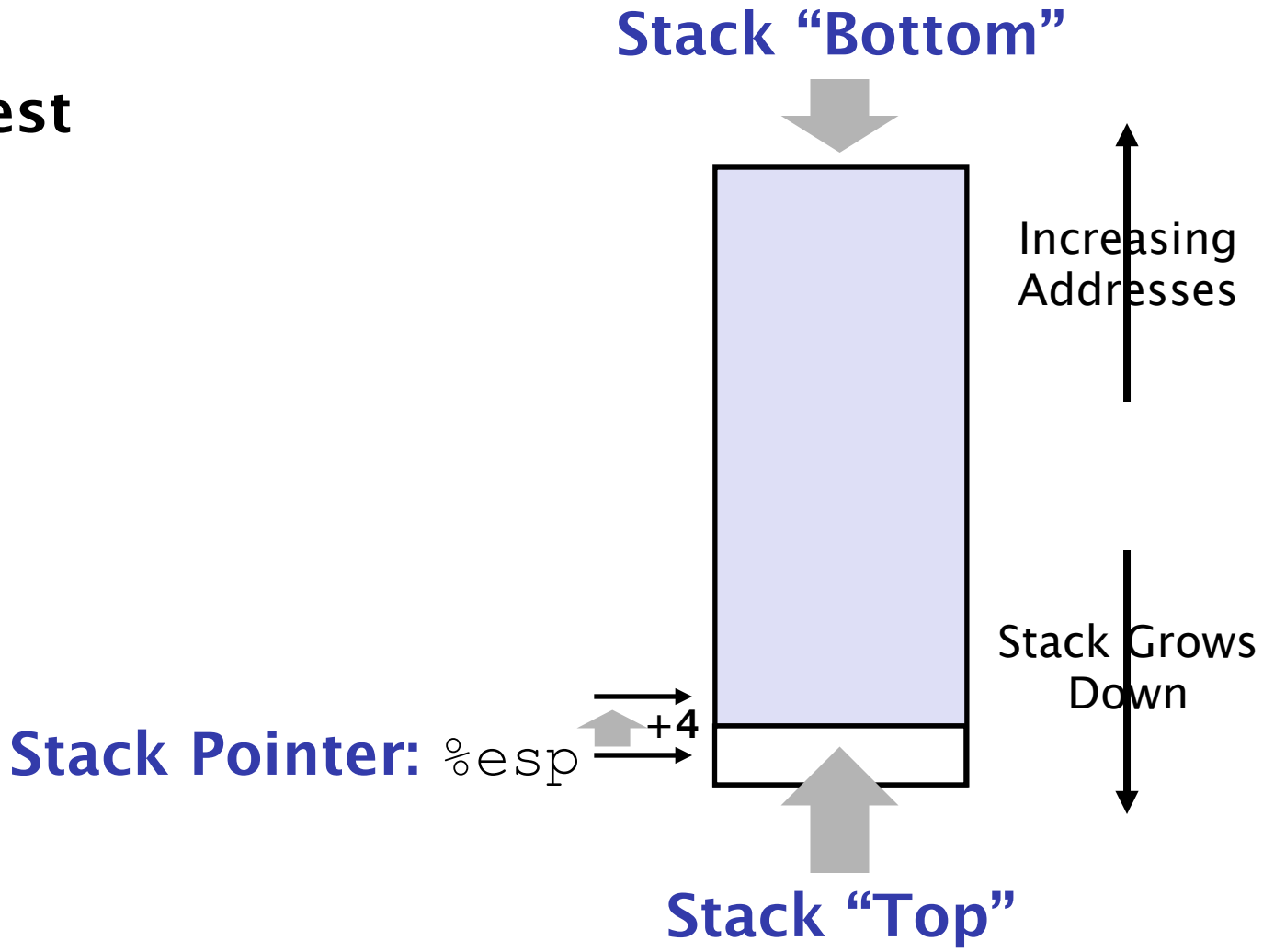


IA32 Stack: Pop

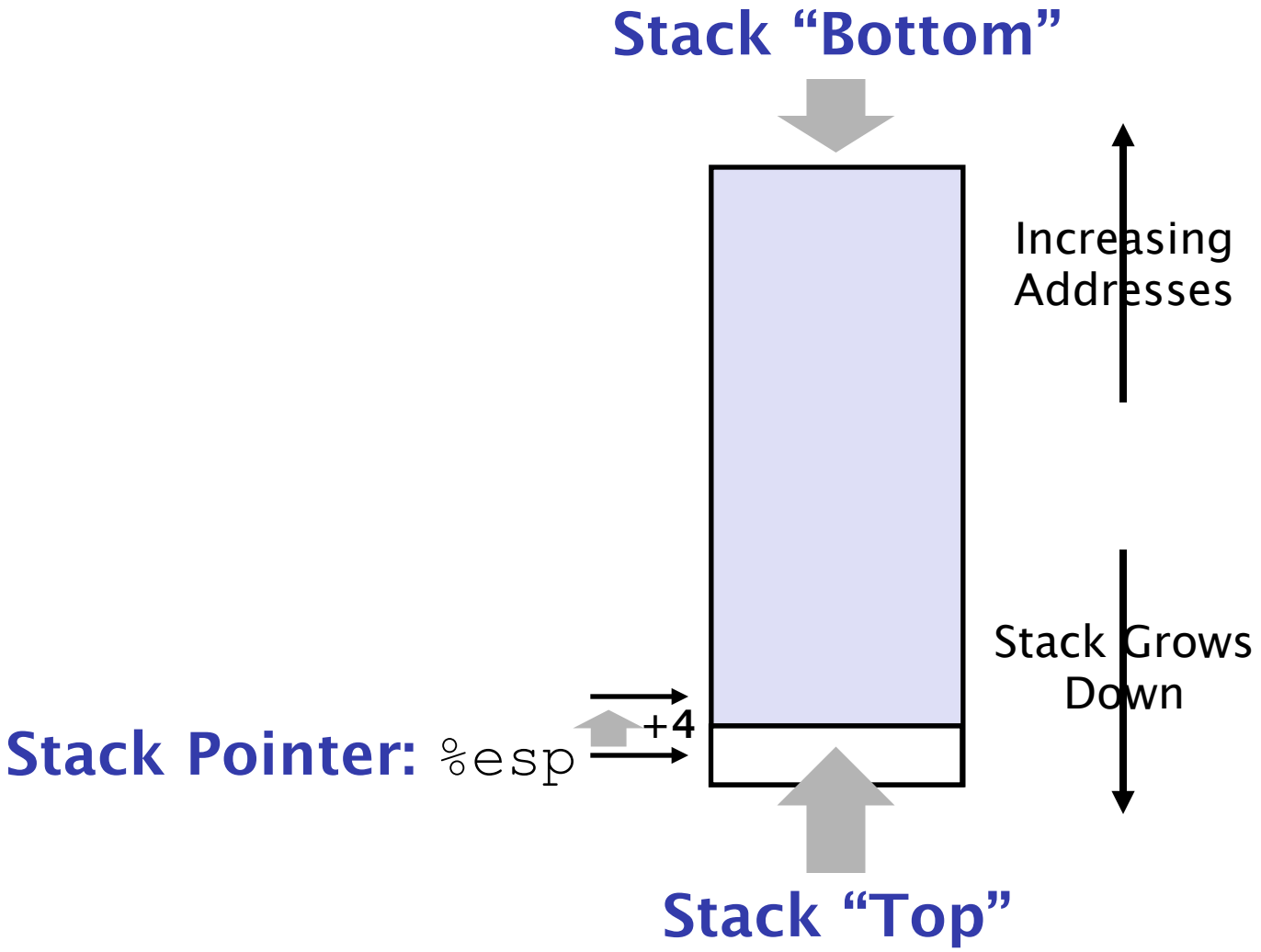


IA32 Stack: Pop

- `popl` Dest

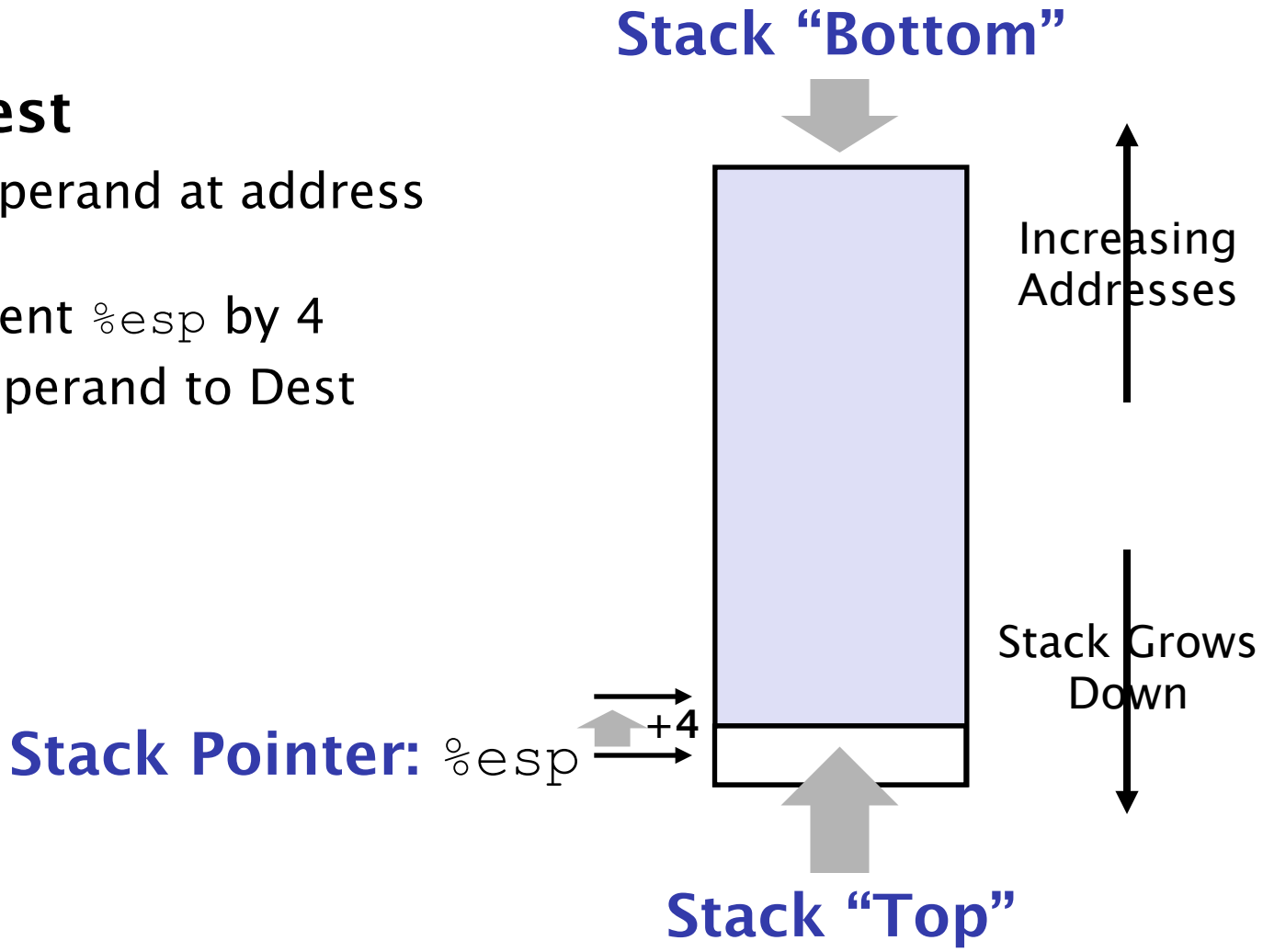


IA32 Stack: Pop



IA32 Stack: Pop

- **`popl` Dest**
  - Read operand at address `%esp`
  - Increment `%esp` by 4
  - Write operand to Dest



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

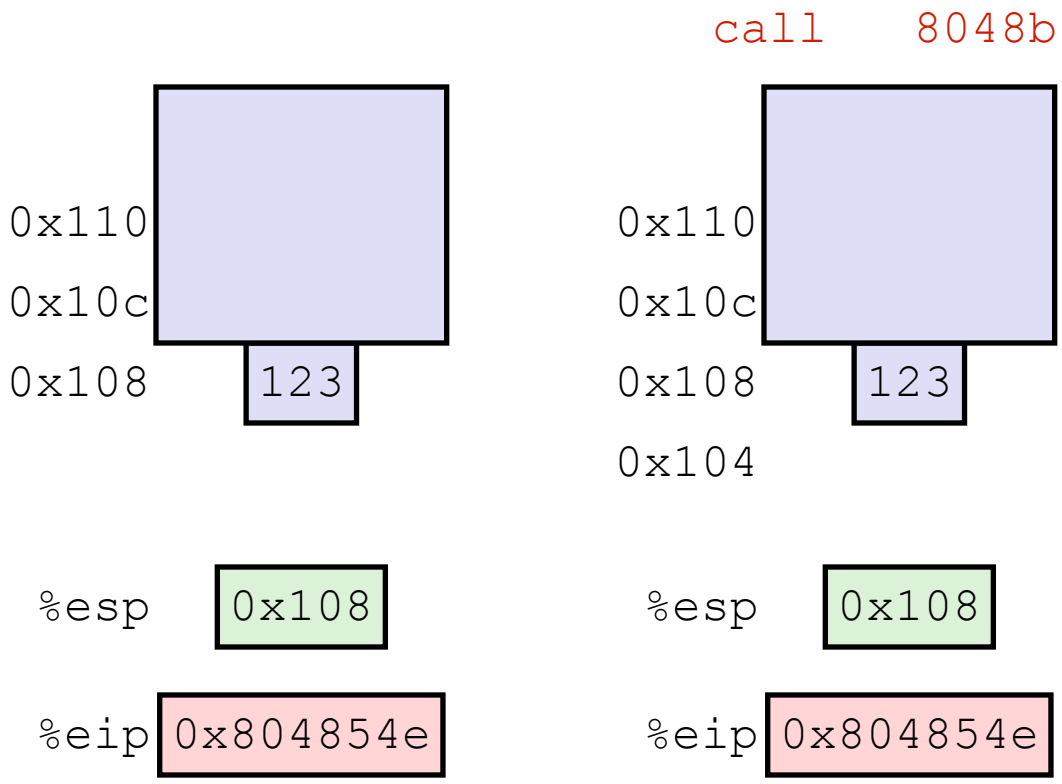
- **Use stack to support procedure call and return**
- **Procedure call:** `call` ***`label`***
  - Push return address on stack
  - Jump to ***`label`***
- **Return address:**
  - Address of instruction beyond `call`
  - Example from disassembly

```
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

Procedure Call Example

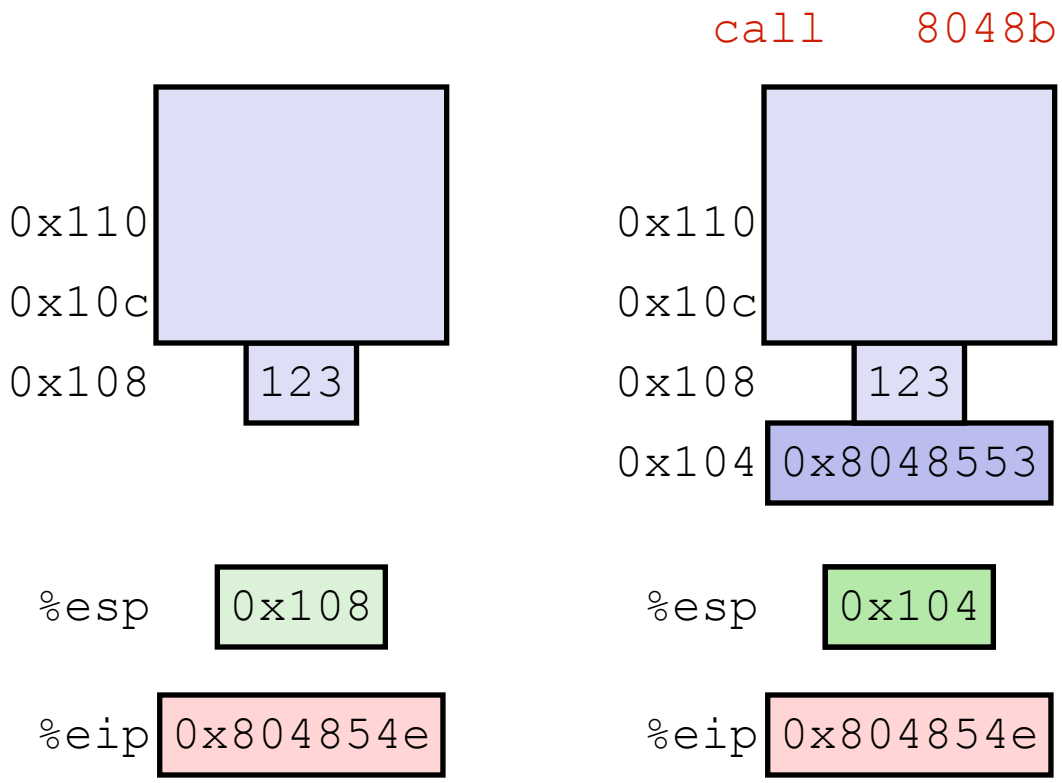
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

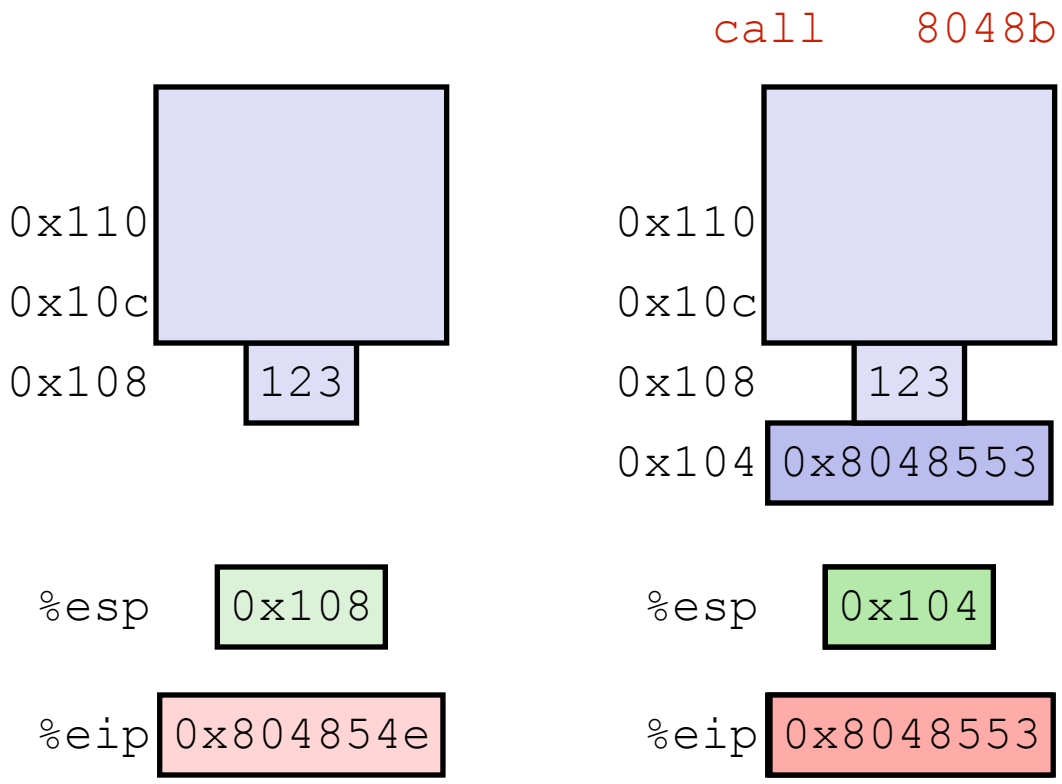
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

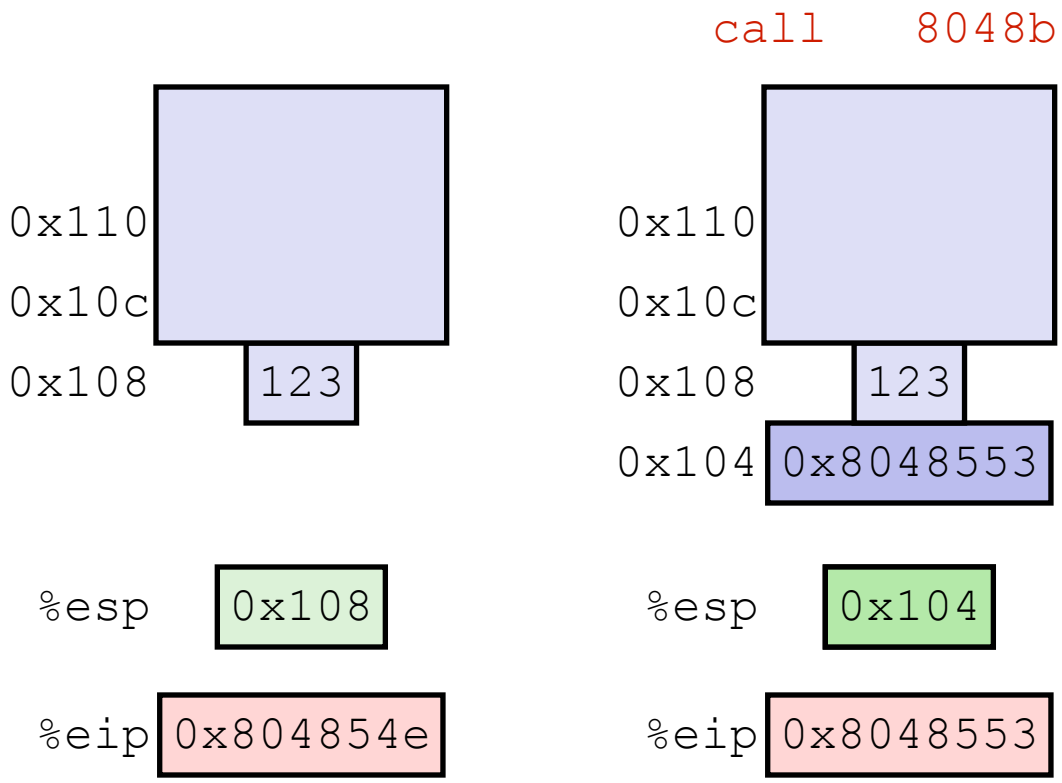
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

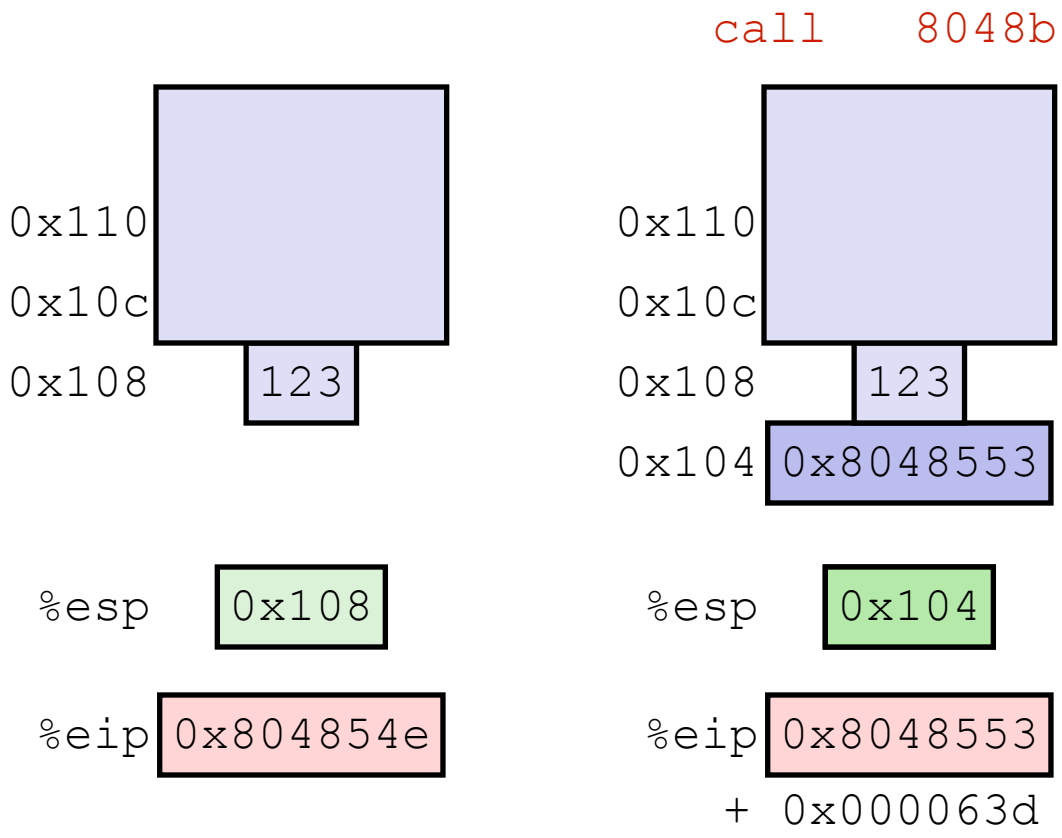
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

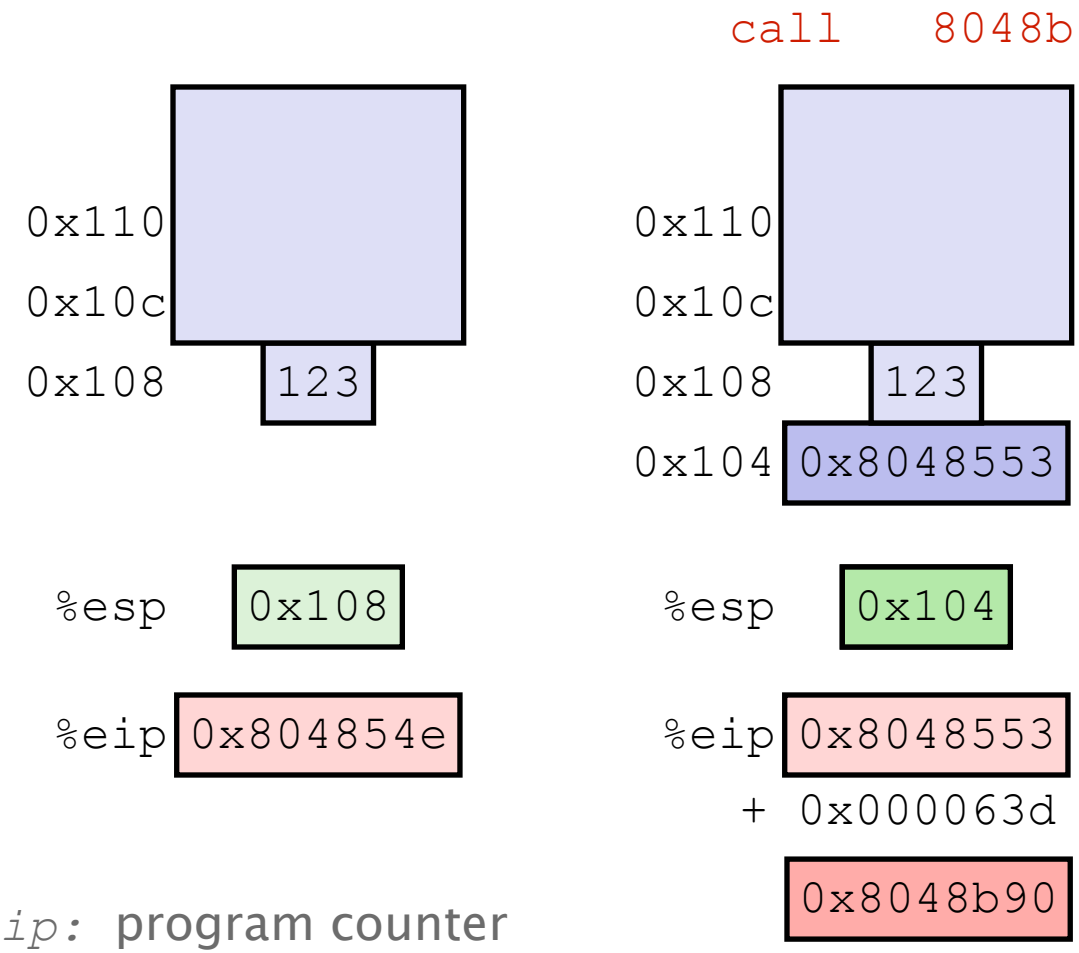
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

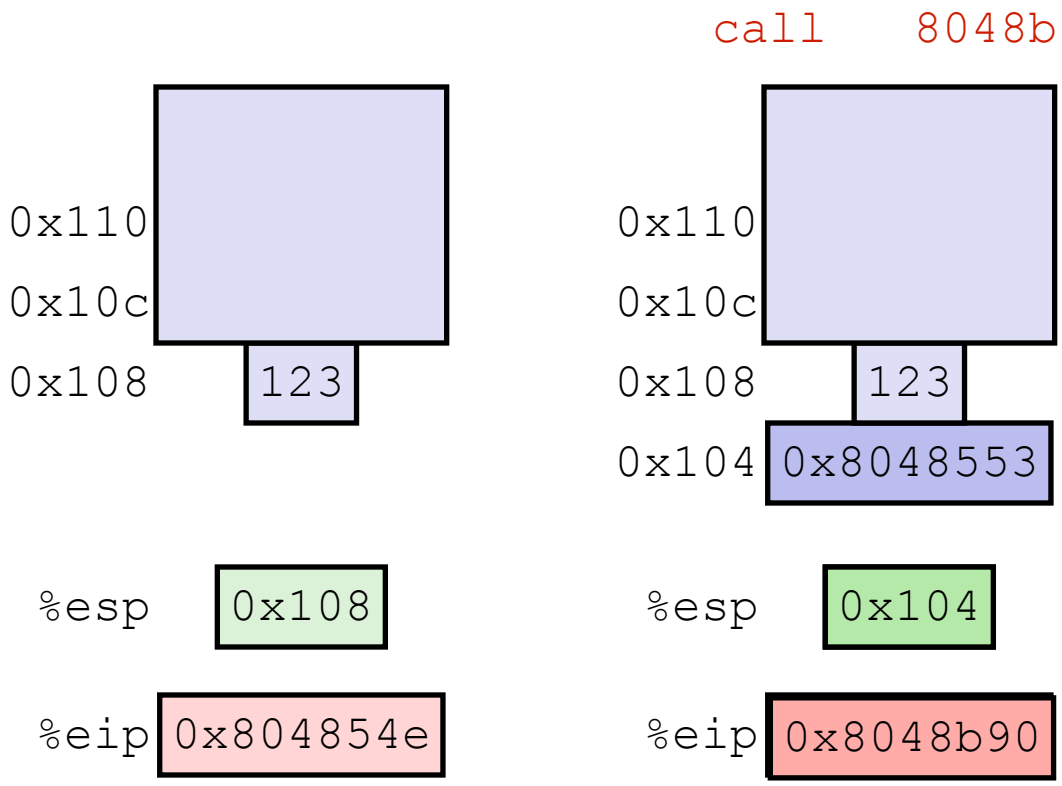
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Call Example

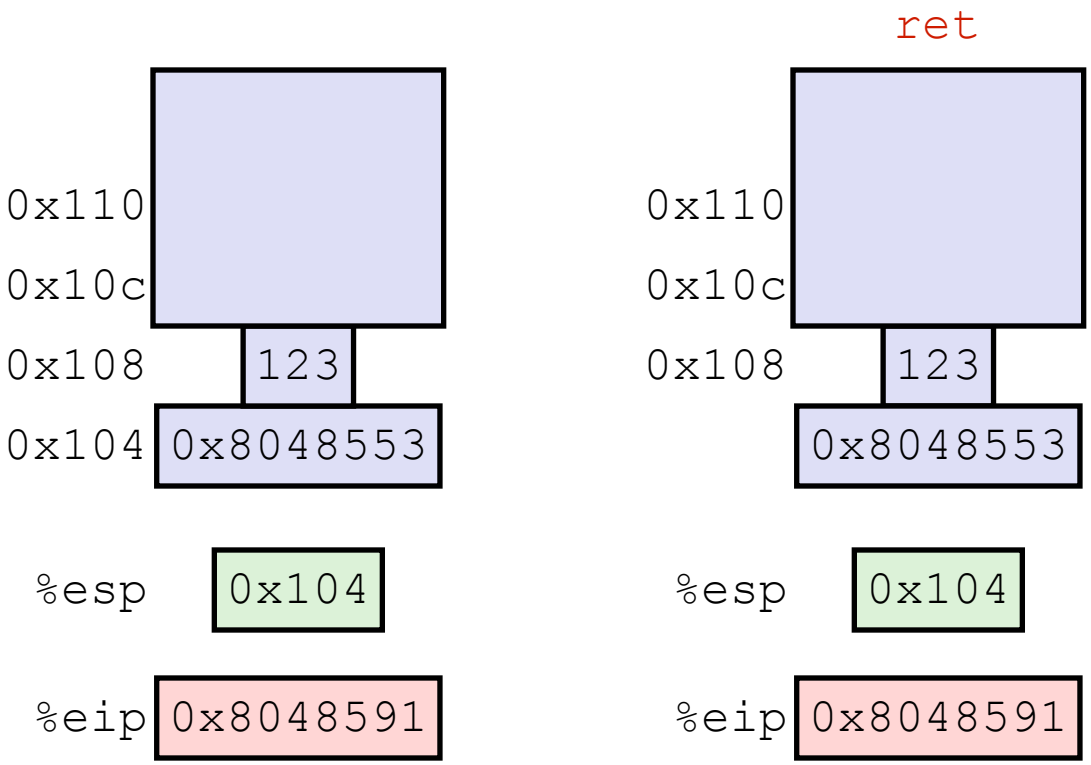
```
804854e:    e8 3d 06 00 00        call   8048b90 <main>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Return Example

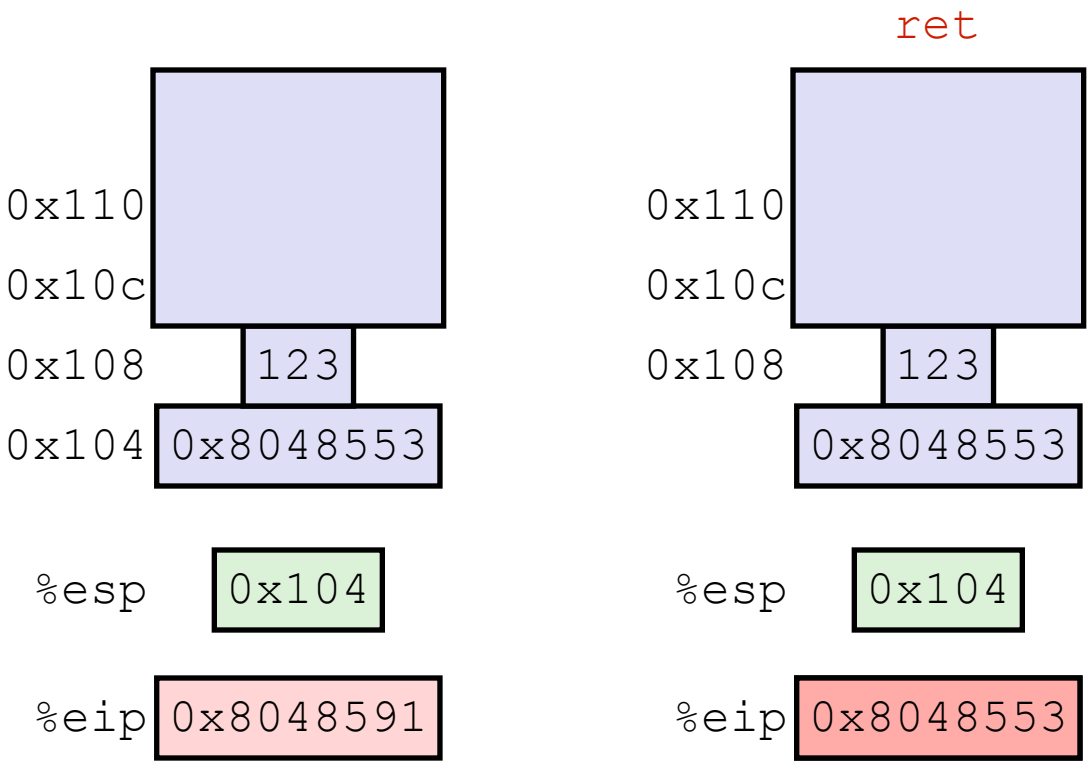
```
8048591:     c3                          ret
```



%eip: program counter

Procedure Return Example

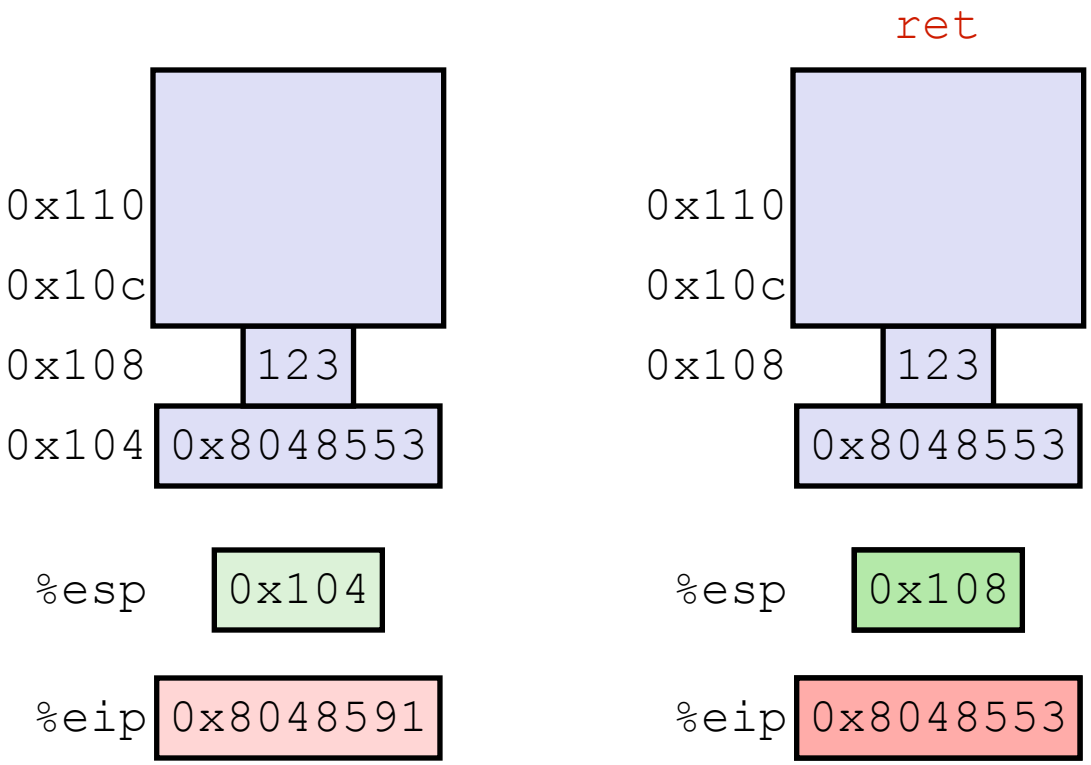
```
8048591:     c3                          ret
```



%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 re-entrant
    - Multiple simultaneous instantiations of single procedure
      - What would happen if code could not be reentrant?
  - Need some place to store state of each instantiation
    - Arguments
    - Local variables
    - Return pointer
- **Stack discipline**
  - State for a given procedure needed for a limited time
    - Starting from when it is called to when it returns
  - Callee always returns before caller does
- **Stack allocated in frames**
  - State for a single procedure instantiation

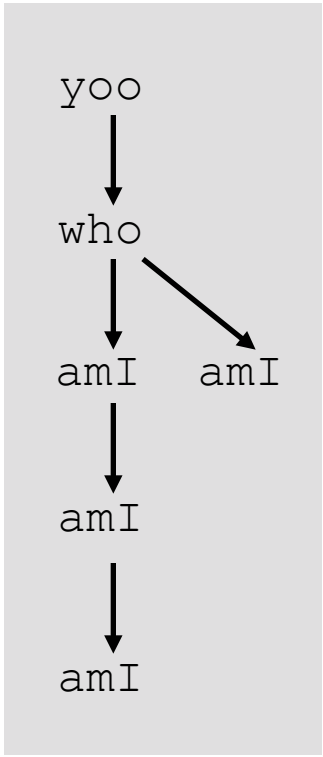
Call Chain Example

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

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

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

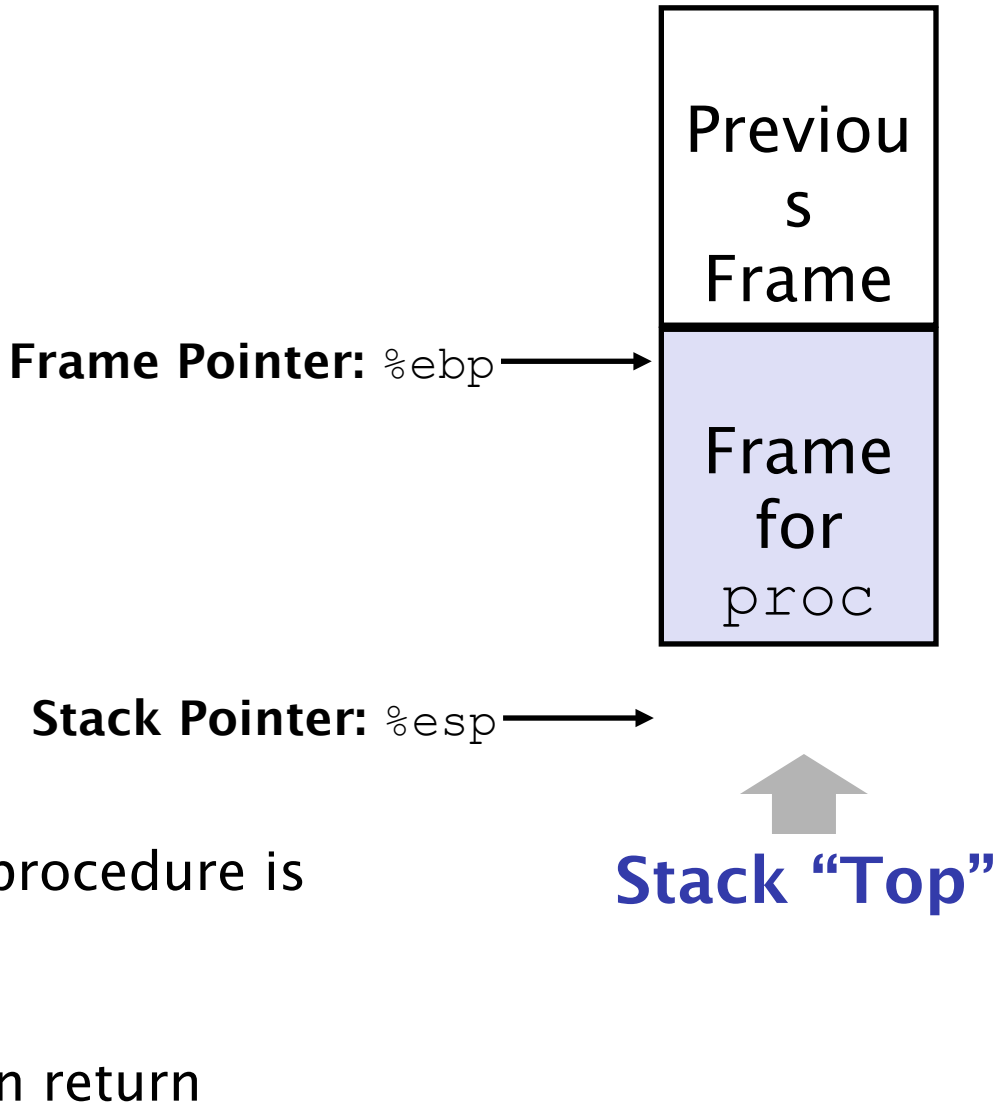
Example Call Chain



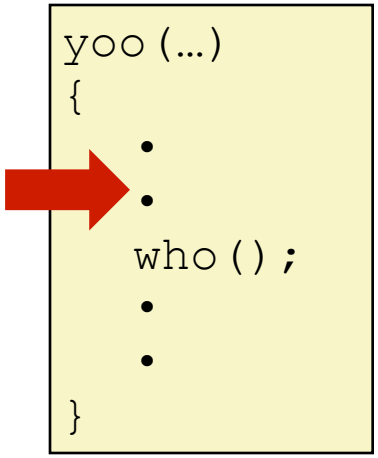
Procedure `amI` is recursive (calls itself)

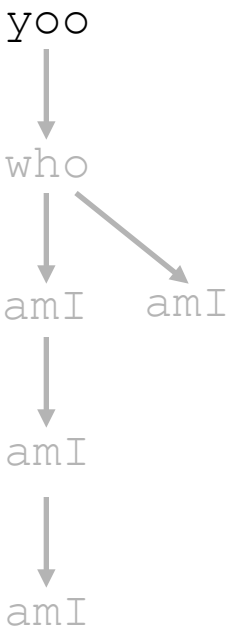
Stack Frames

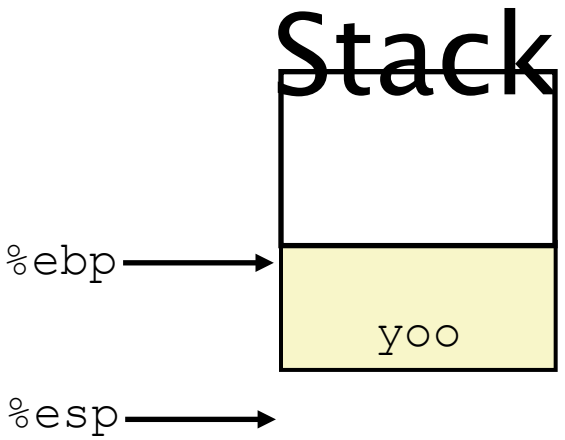
- **Contents**
  - Local variables
  - Return information
  - Temporary space
- **Management**
  - Space allocated when procedure is entered
    - "Set-up" code
  - Space deallocated upon return
    - "Finish" code



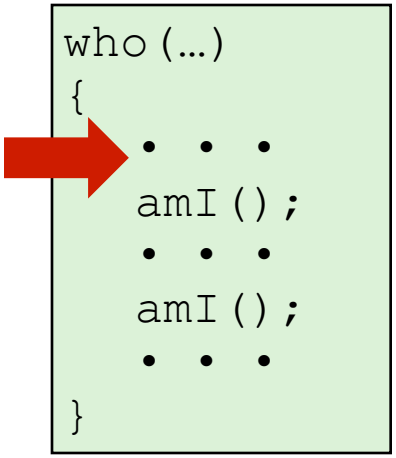
Example

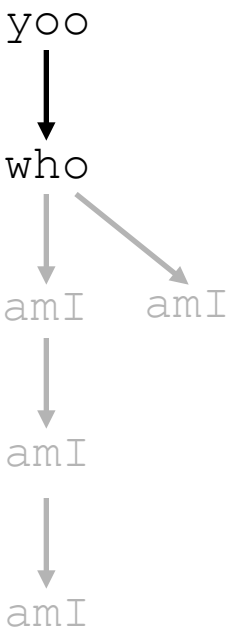


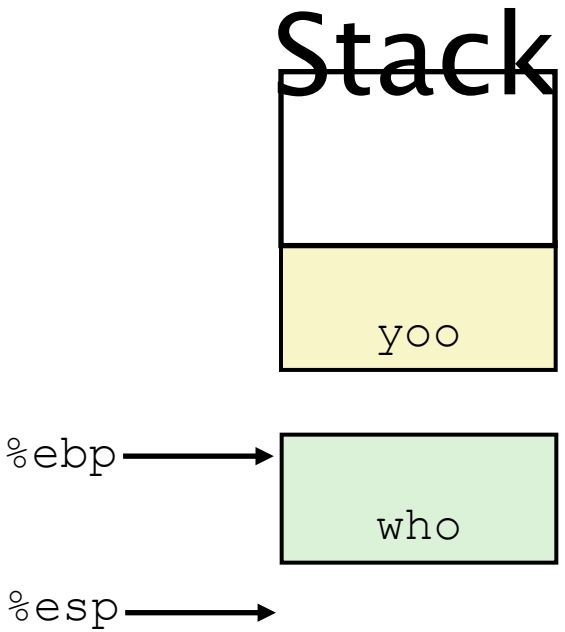




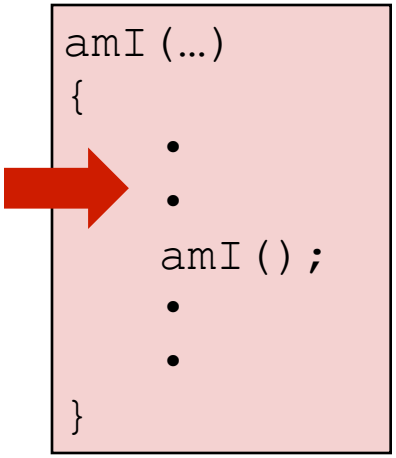
Example

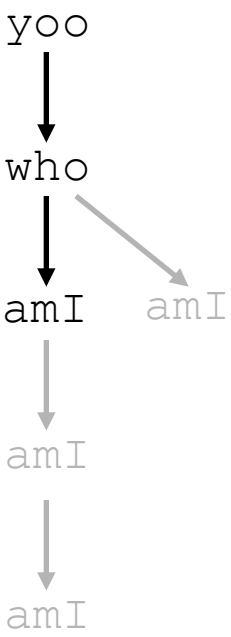


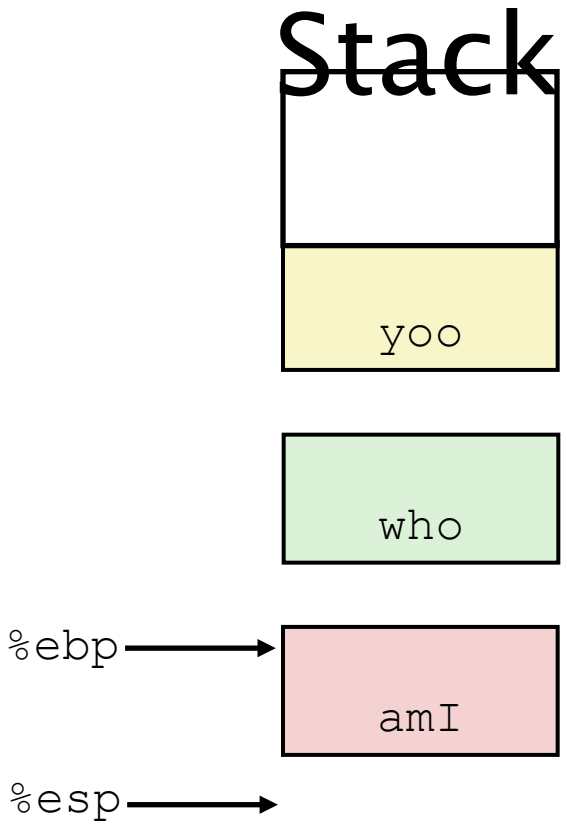




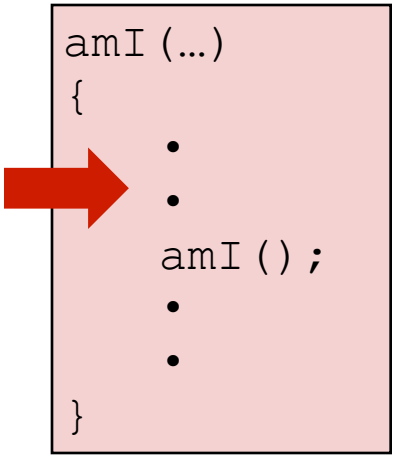
Example

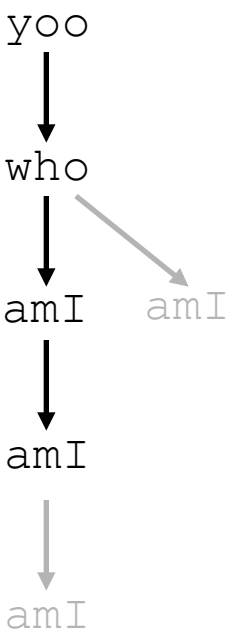


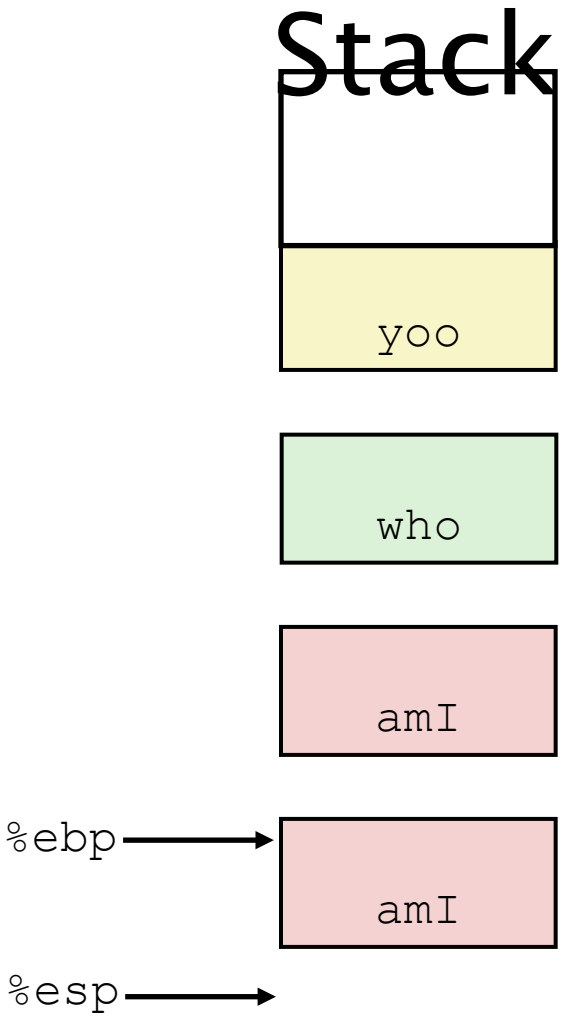




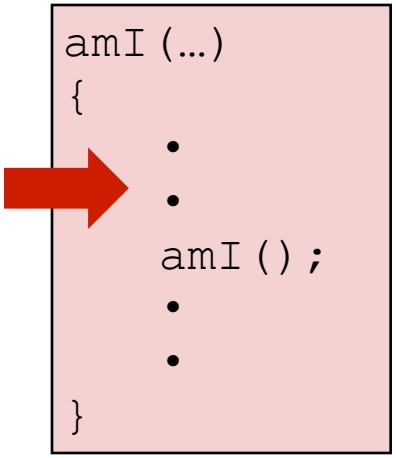
Example

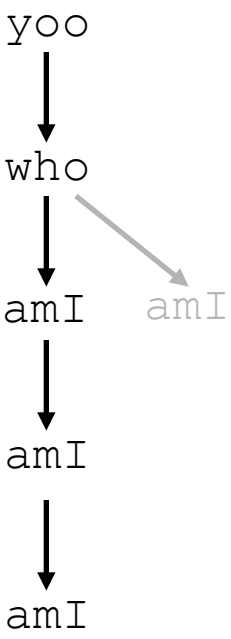


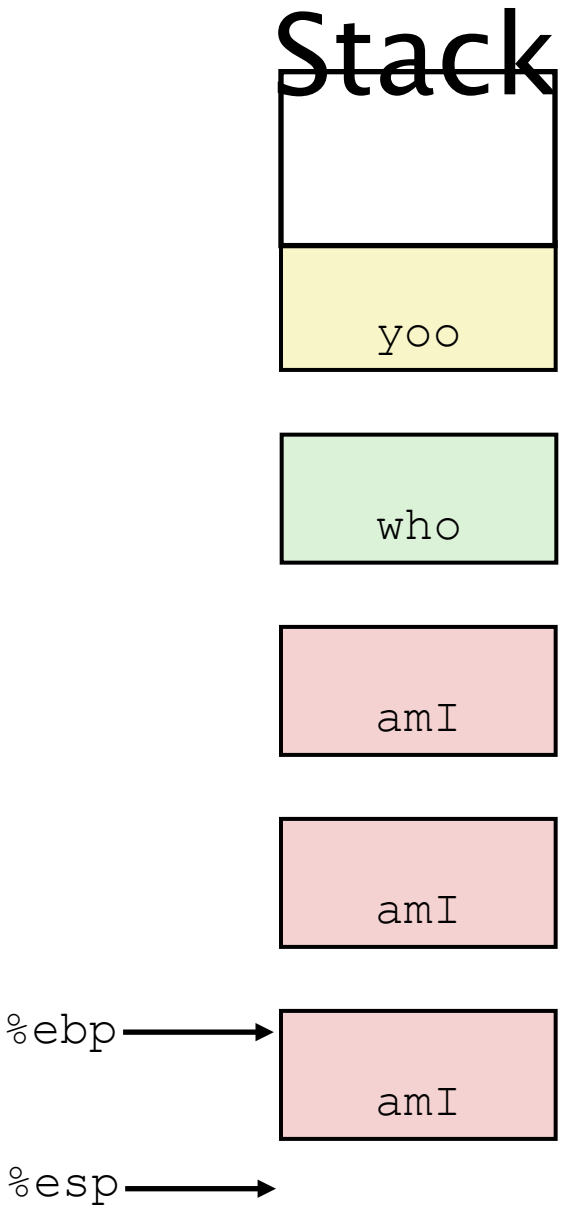




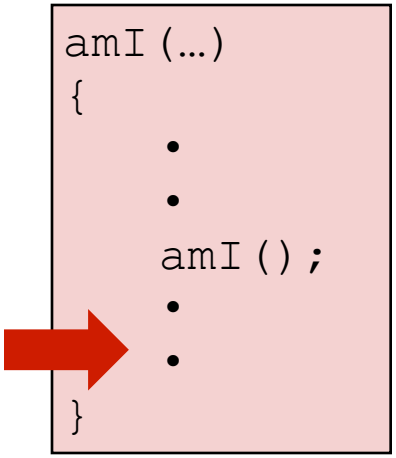
Example

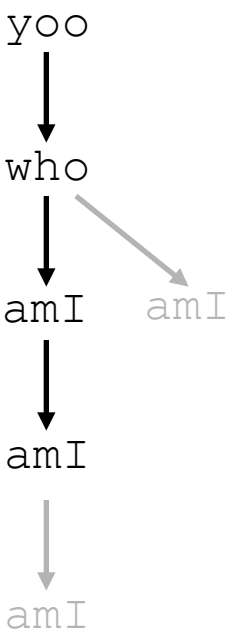

amI(…)
{
•
•
amI();
•
•
}

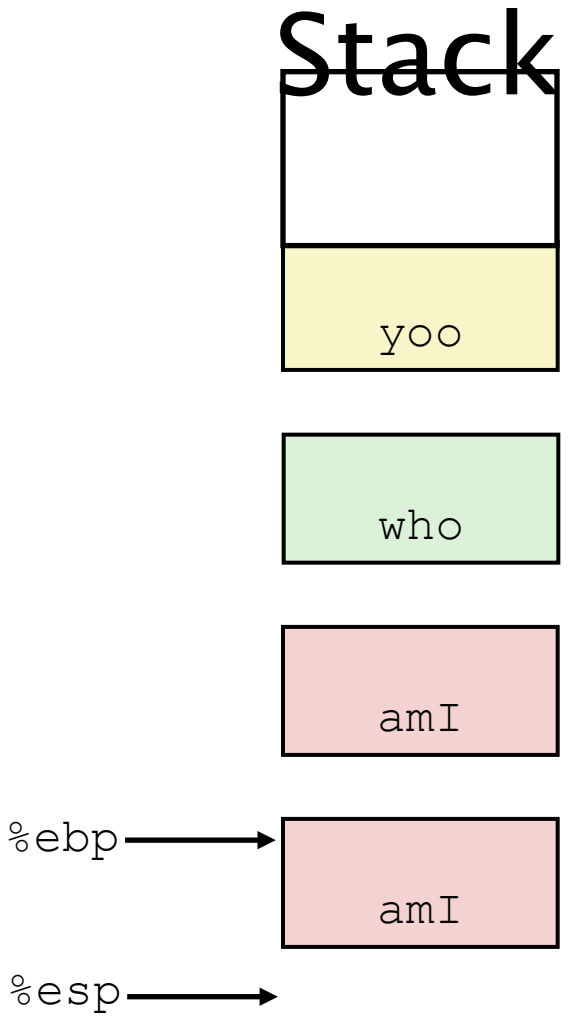

yoo
who
amI
amI
amI
amI


Stack
yoo
who
amI
amI
%ebp
amI
%esp

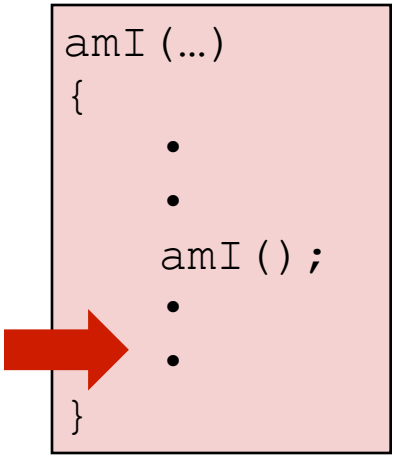
Example

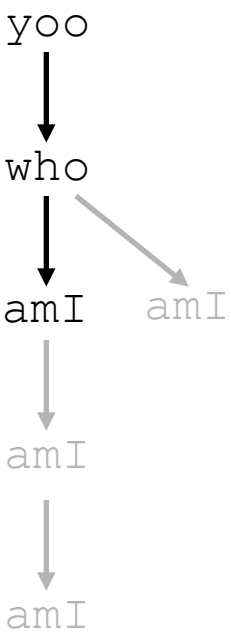


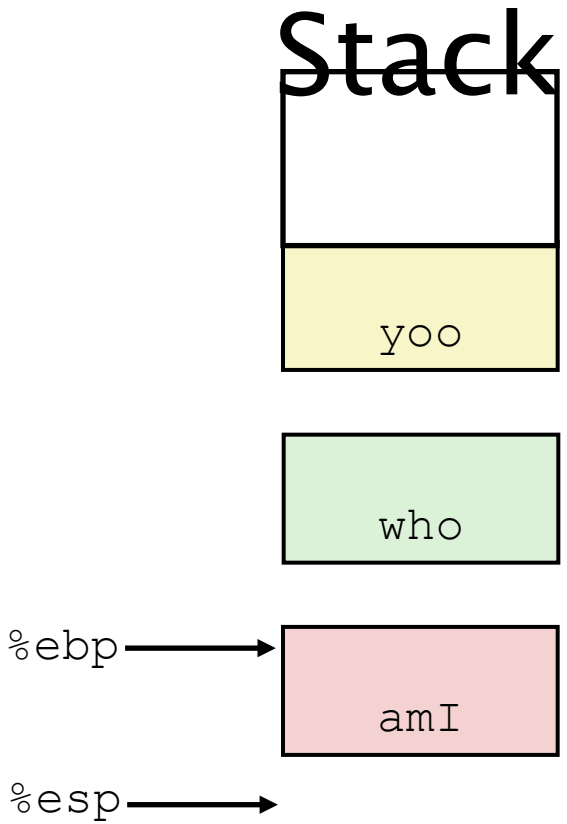




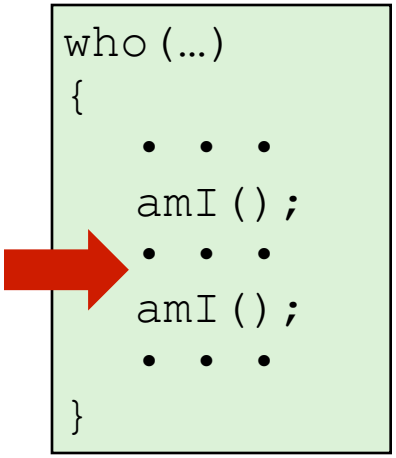
Example

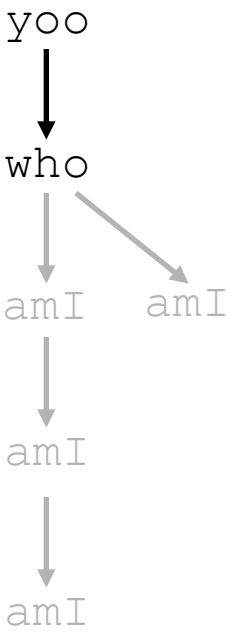


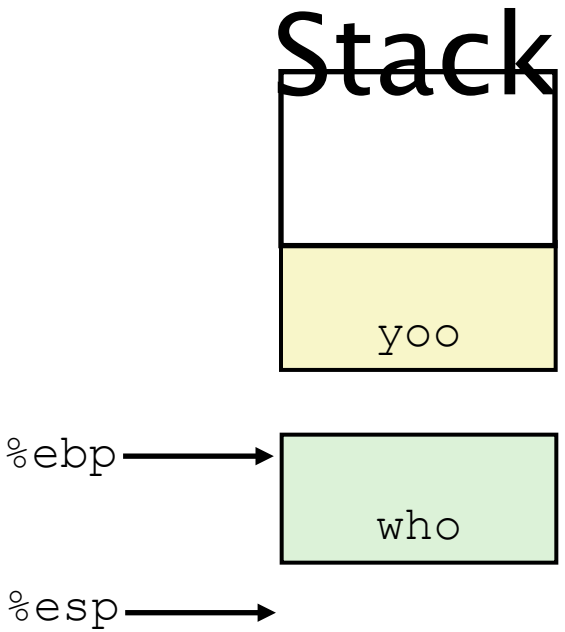




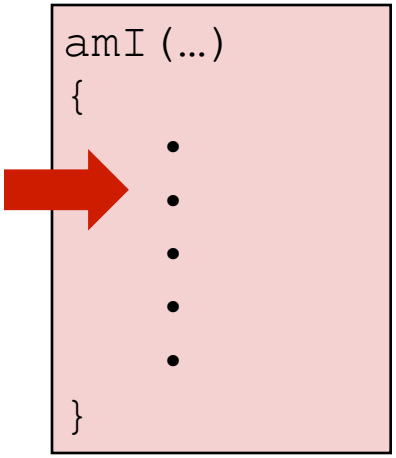
Example

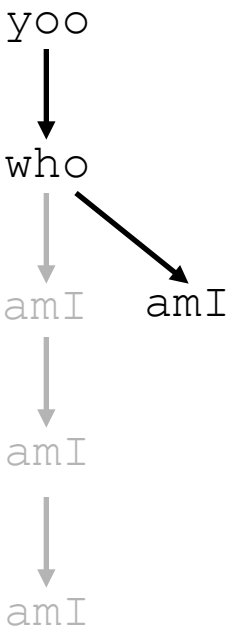


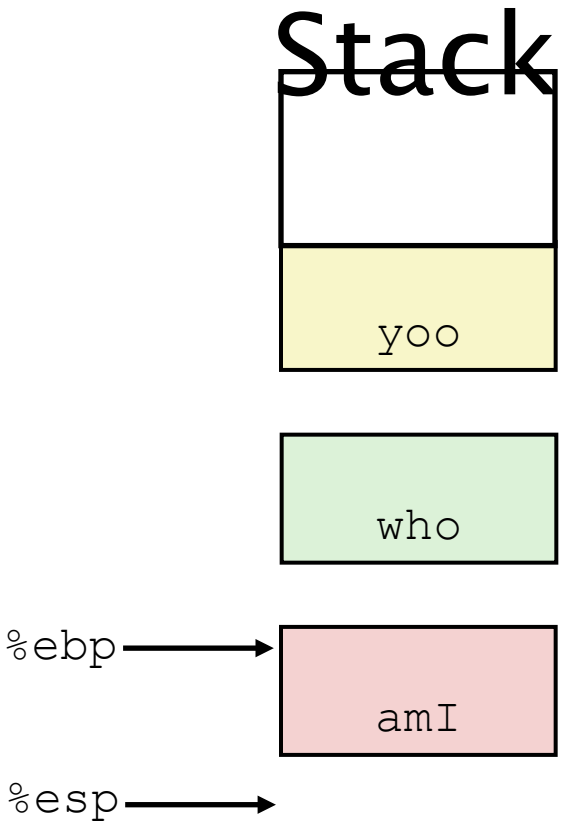




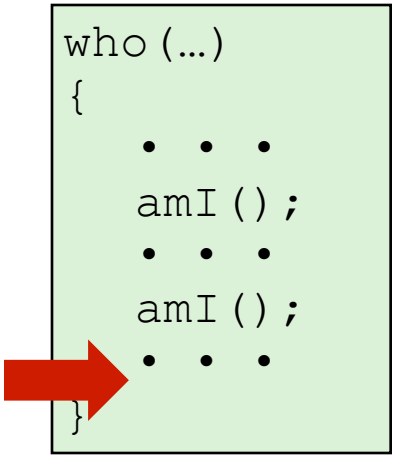
Example

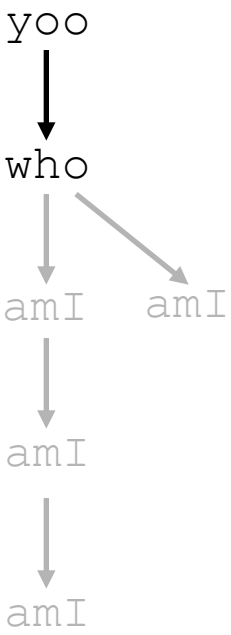


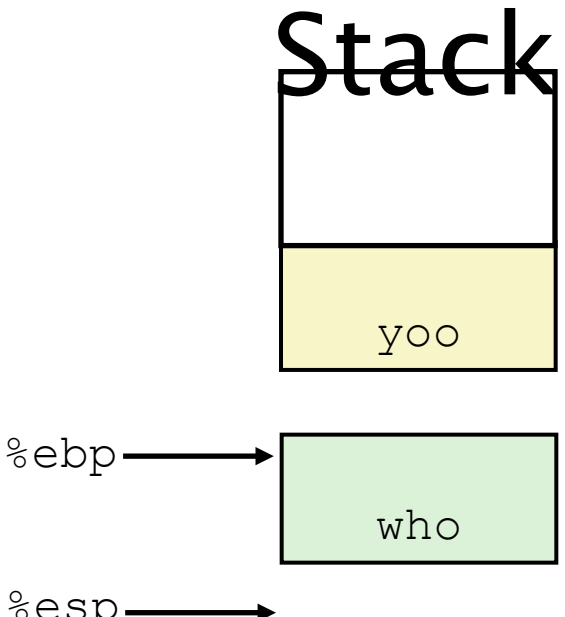




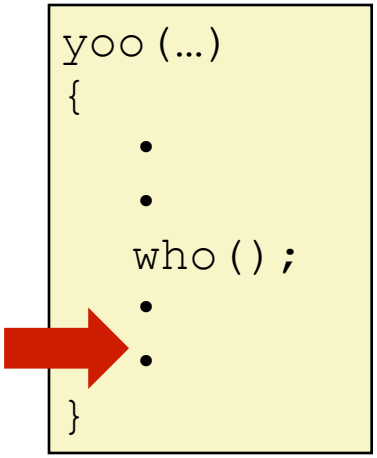
Example

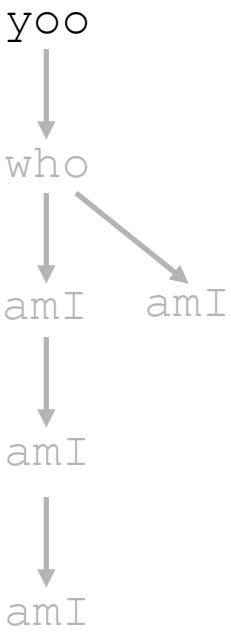


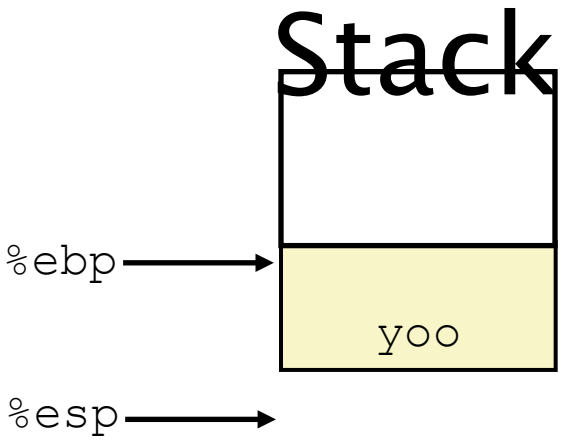




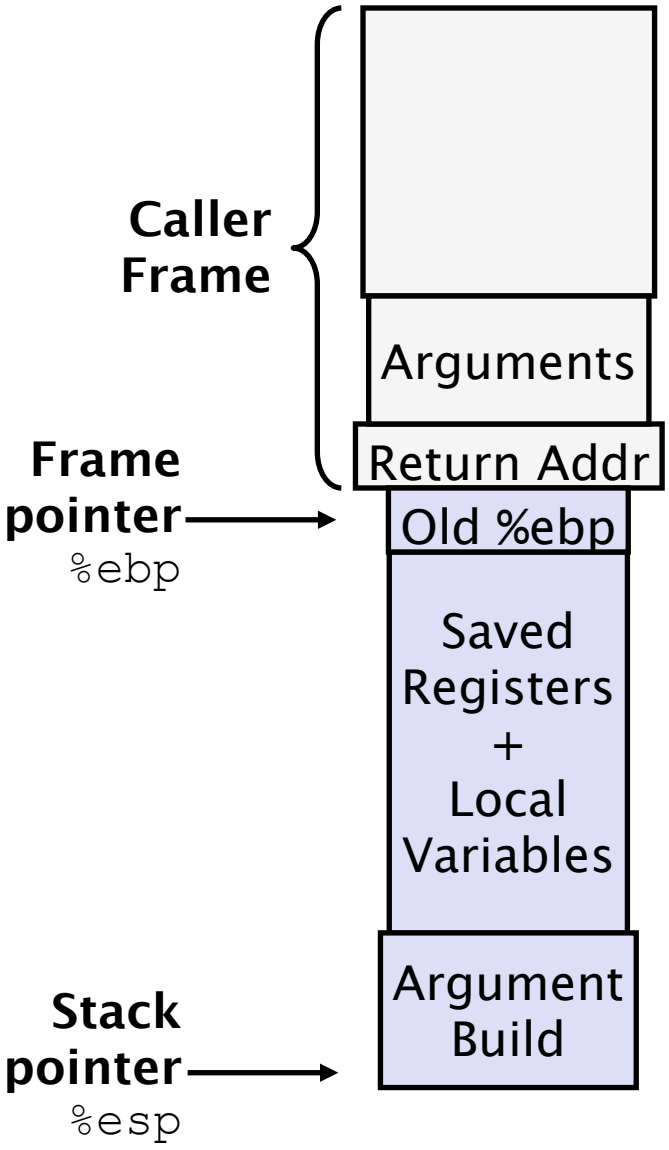
Example





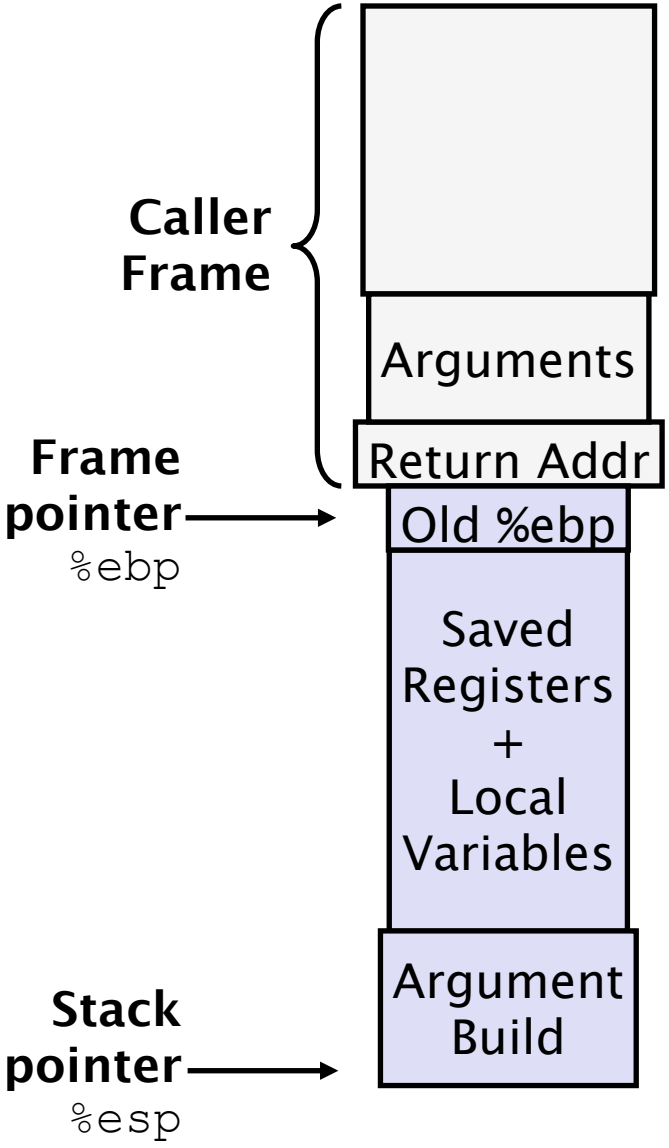


IA32/Linux Stack Frame



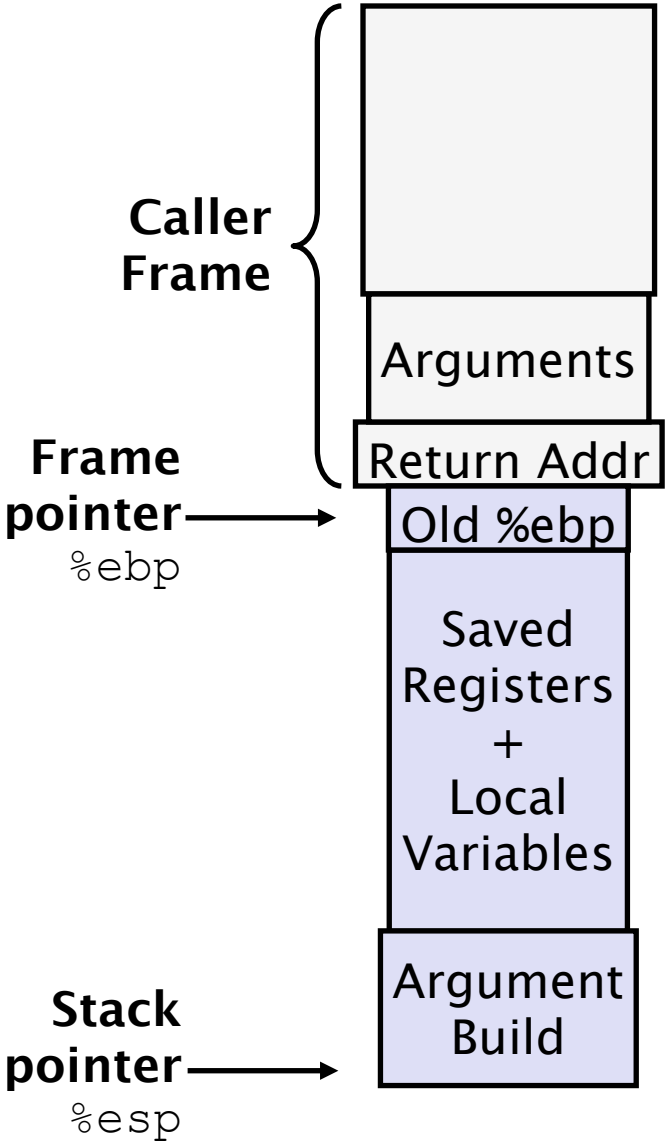
IA32/Linux Stack Frame

- **Current Stack Frame ("Top" to Bottom)**
  - Old frame pointer
  - Local variables
    If can't be just kept in registers
  - Saved register context
    When reusing registers
  - "Argument build area"
    Parameters for function about to be called



IA32/Linux Stack Frame

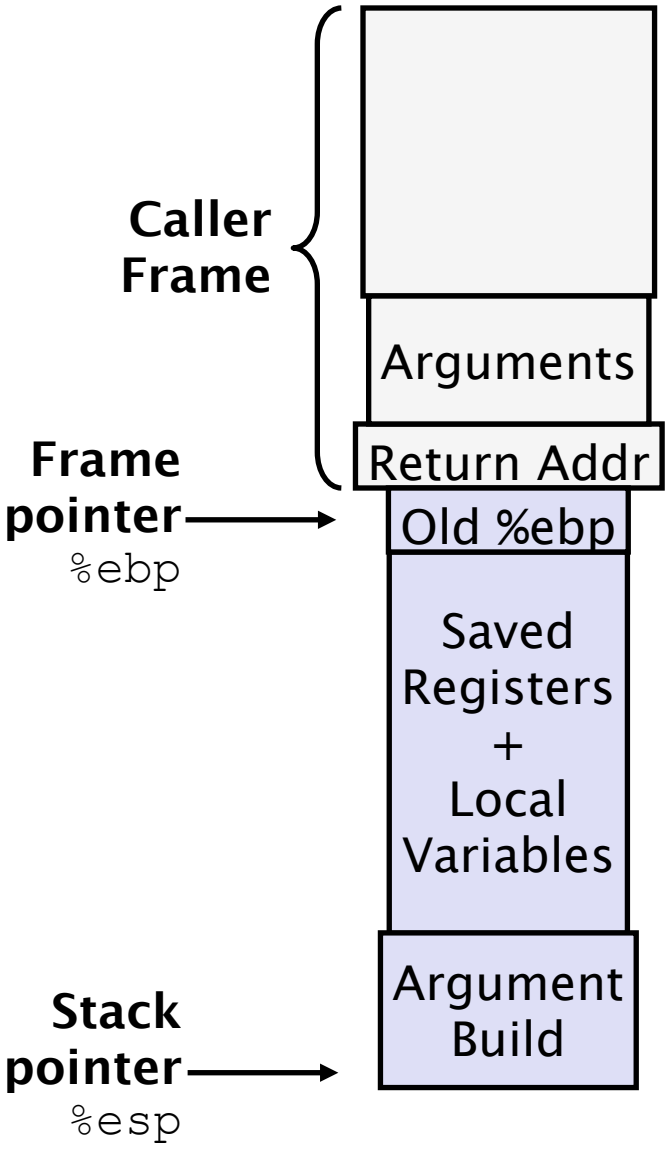
- **Current Stack Frame ("Top" to Bottom)**
  - Old frame pointer
  - Local variables
    If can't be just kept in registers
  - Saved register context
    When reusing registers
  - "Argument build area"
    Parameters for function about to be called



IA32/Linux Stack Frame

- **Current Stack Frame ("Top" to Bottom)**
  - Old frame pointer
  - Local variables
    If can't be just kept in registers
  - Saved register context
    When reusing registers
  - "Argument build area"
    Parameters for function about to be called

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



Revisiting swap

```
int zip1 = 15213;
int zip2 = 98195;

void call_swap()
{
  swap(&zip1, &zip2);
}
```

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

Revisiting swap

```
int zip1 = 15213;
int zip2 = 98195;

void call_swap()
{
  swap(&zip1, &zip2);
}
```

Calling `swap` from `call_swap`

```
call_swap:
    . . .
    pushl $zip2    # Global Var
    pushl $zip1    # Global Var
    call swap
    . . .
```

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

Revisiting swap

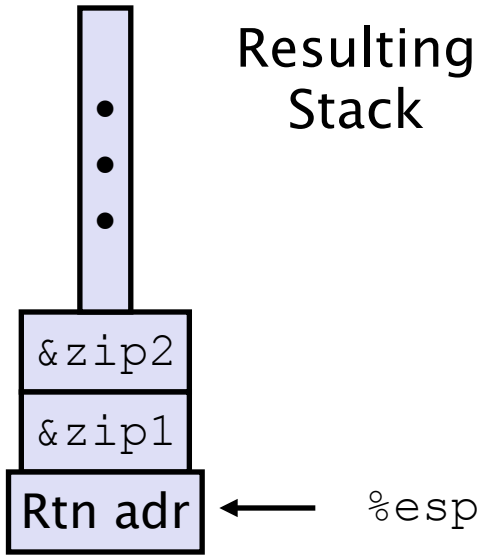
```
int zip1 = 15213;
int zip2 = 98195;

void call_swap()
{
  swap(&zip1, &zip2);
}
```

Calling `swap` from `call_swap`

```
call_swap:
    • • •
    pushl $zip2    # Global Var
    pushl $zip1    # Global Var
    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
```

Set Up

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

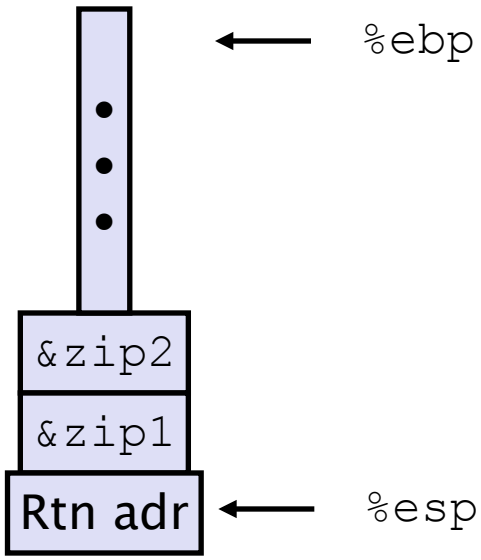
Body

```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
      ret
```

Finish

swap Setup #1

Entering Stack

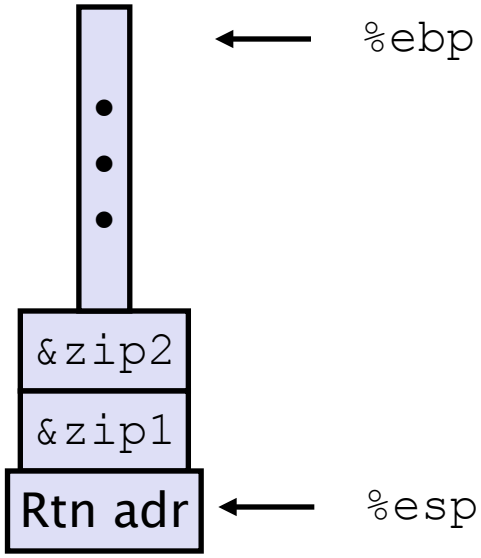


Resulting Stack?

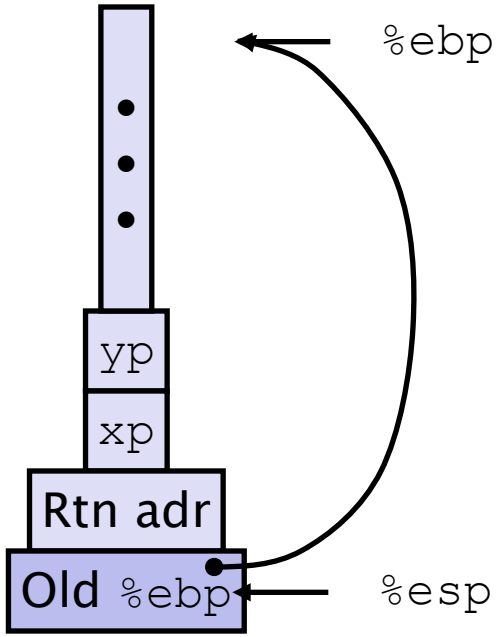
```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

swap Setup #1

Entering Stack



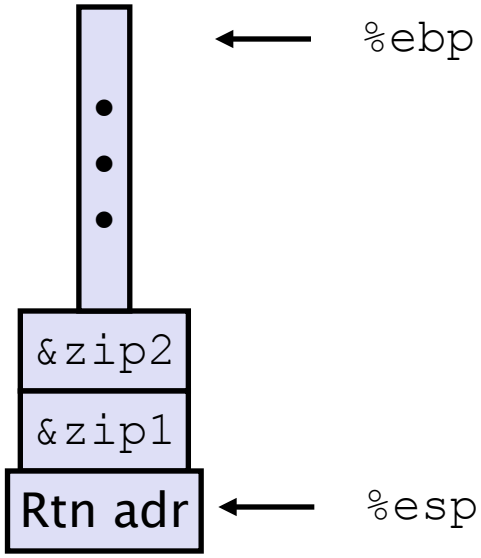
Resulting Stack

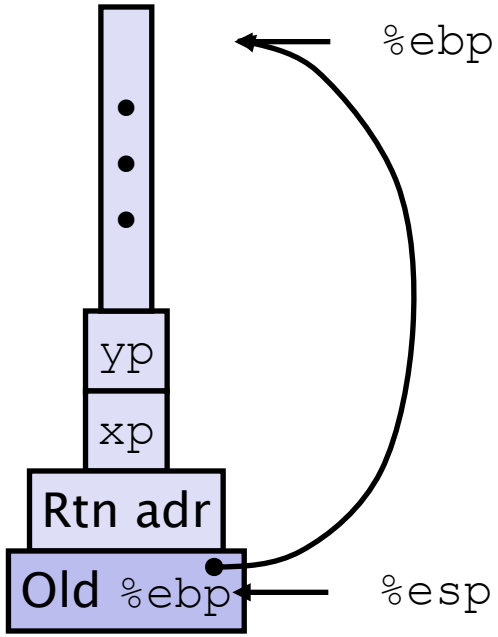



```
swap:
  pushl %ebp
movl %esp,%ebp
  pushl %ebx
```

swap Setup #1

Entering Stack

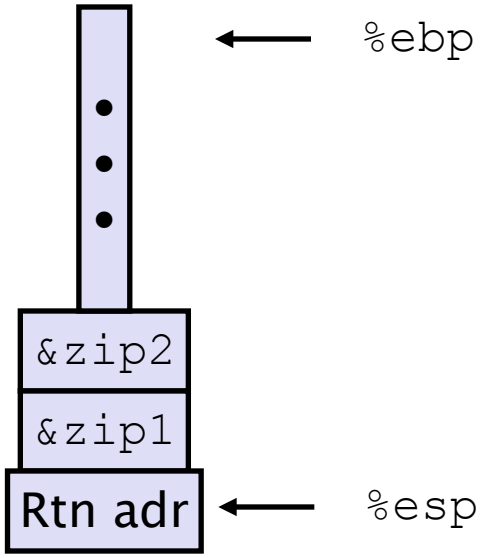




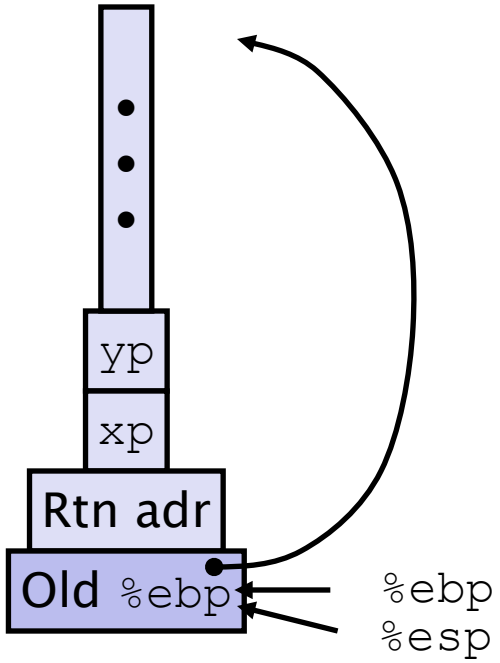

```
swap:
    pushl  %ebp
    movl   %esp,%ebp
    pushl  %ebx
```

swap Setup #1

Entering Stack



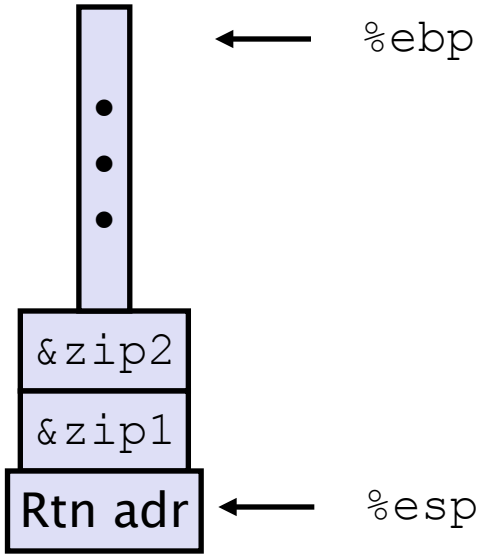
Resulting Stack

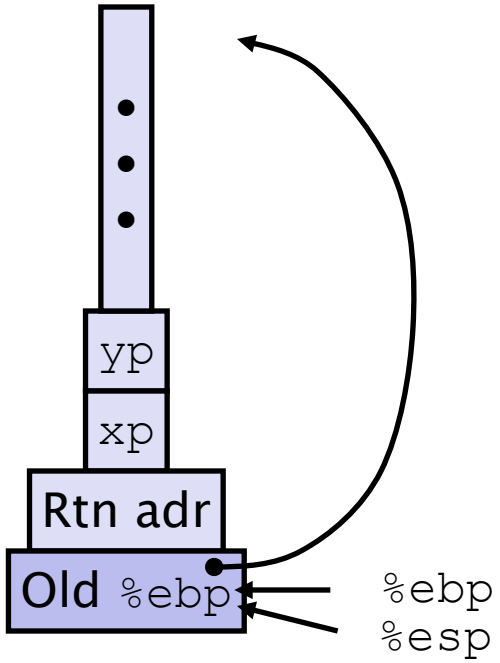



```
swap:
    pushl  %ebp
    movl   %esp,%ebp
    pushl  %ebx
```

swap Setup #1

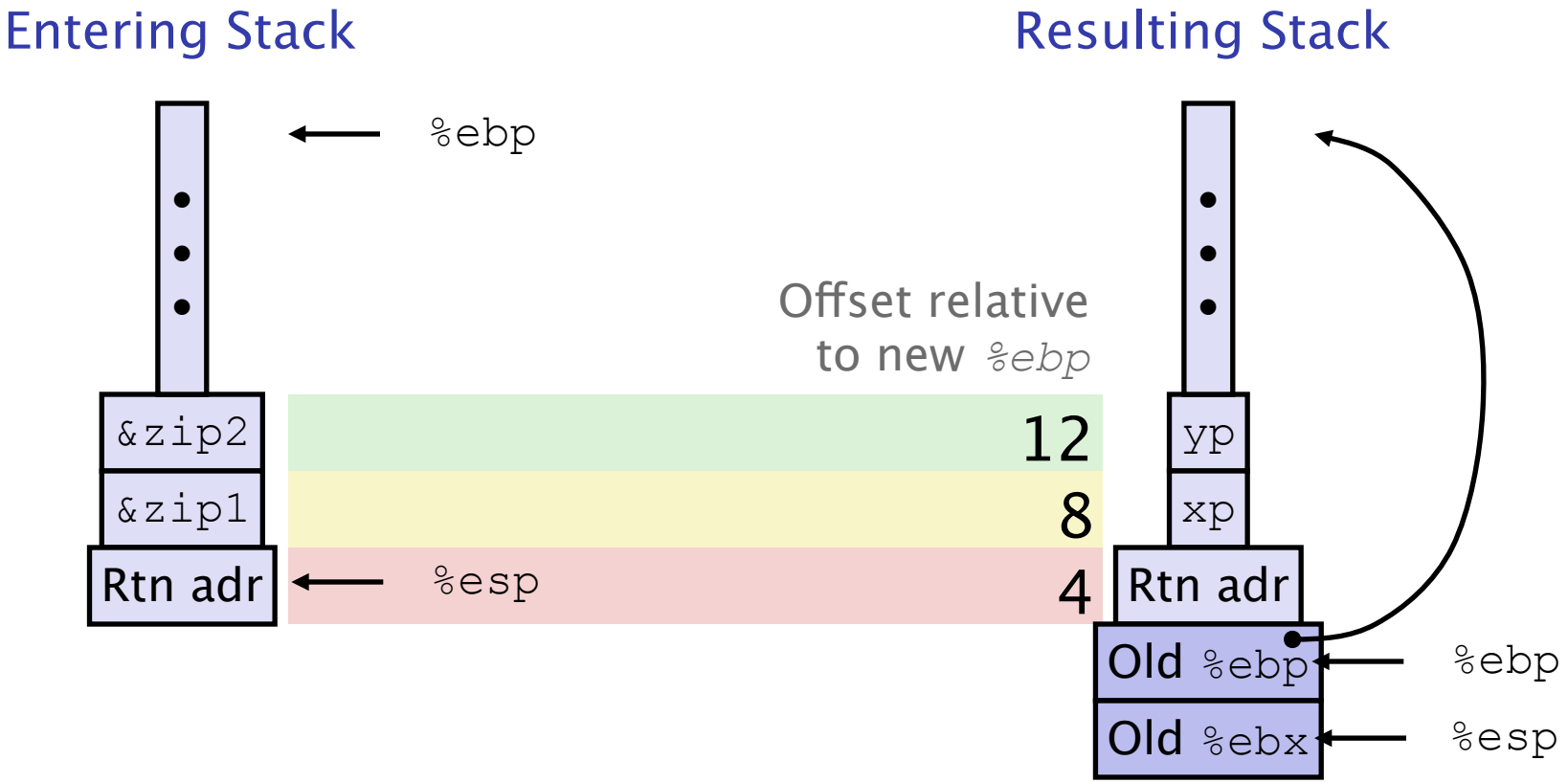
Entering Stack






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

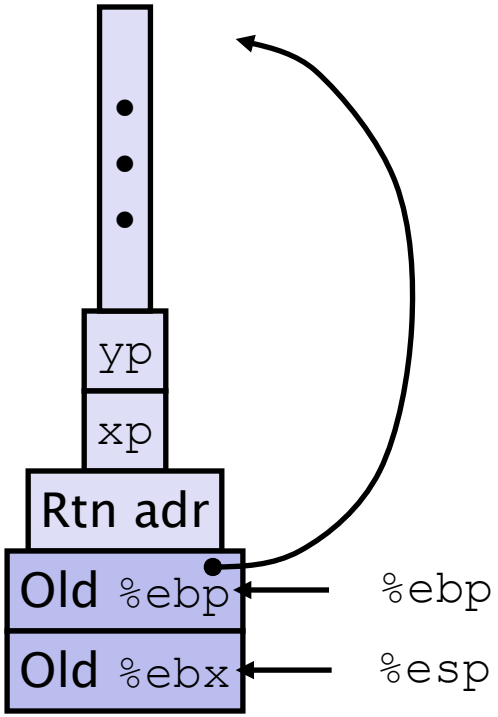
swap Setup #1




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

swap Finish #1

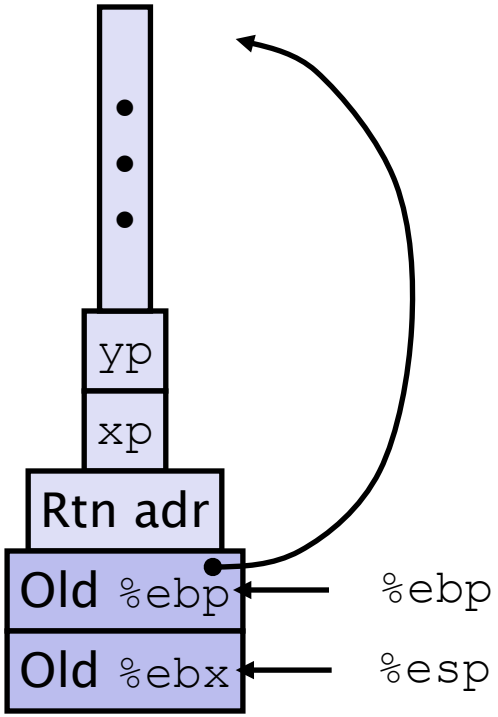
swap's Stack




```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #1

`swap's` Stack

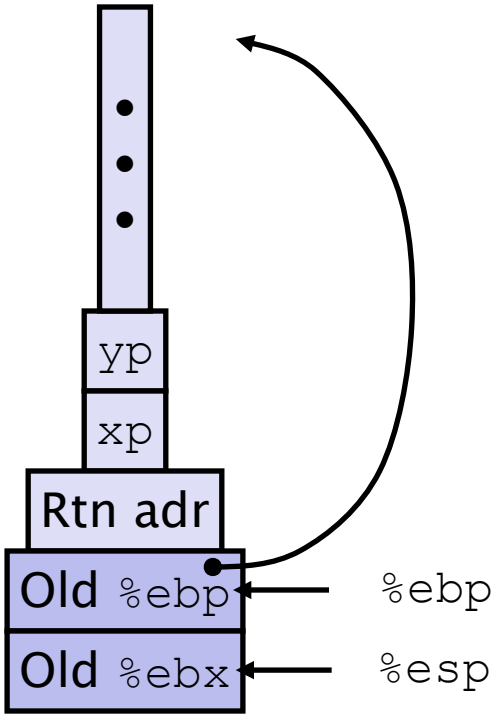


Resulting Stack

```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #1

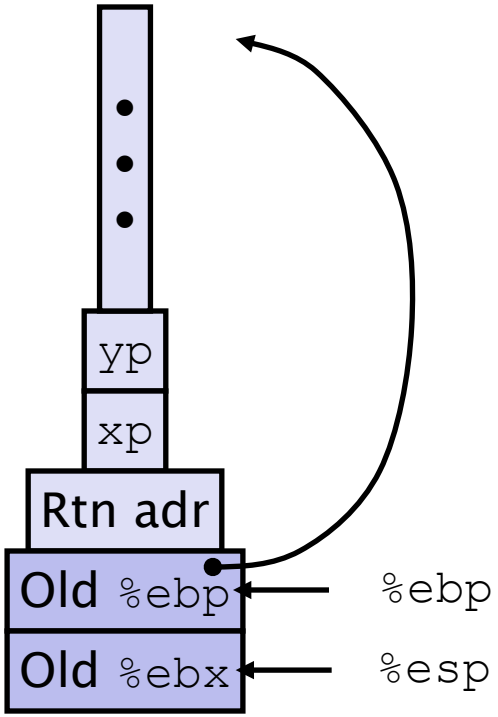
swap's Stack



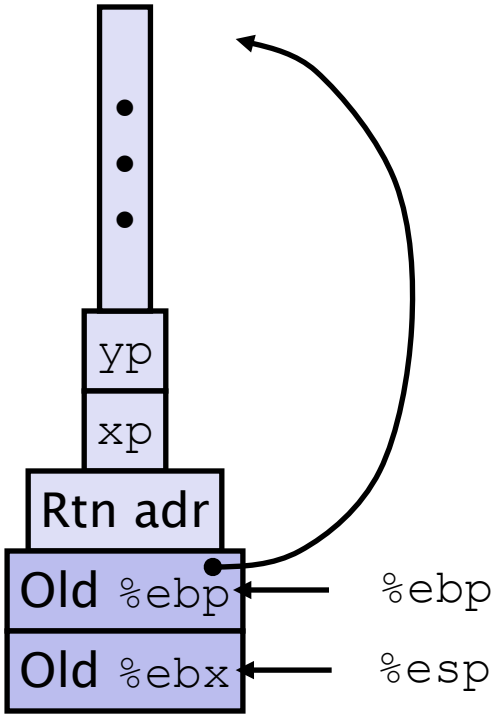

```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #1

`swap`'s Stack



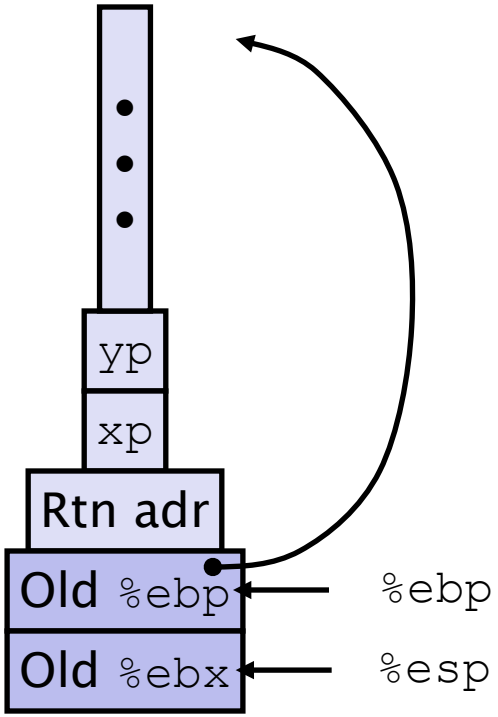
Resulting Stack



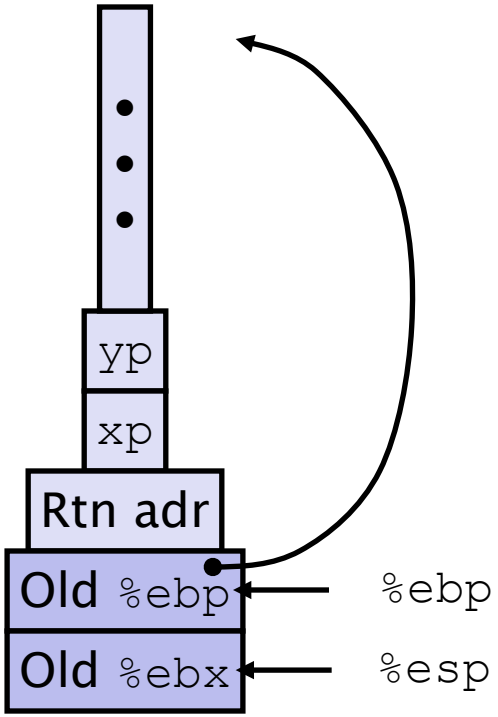

```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #1

swap's Stack



Resulting Stack

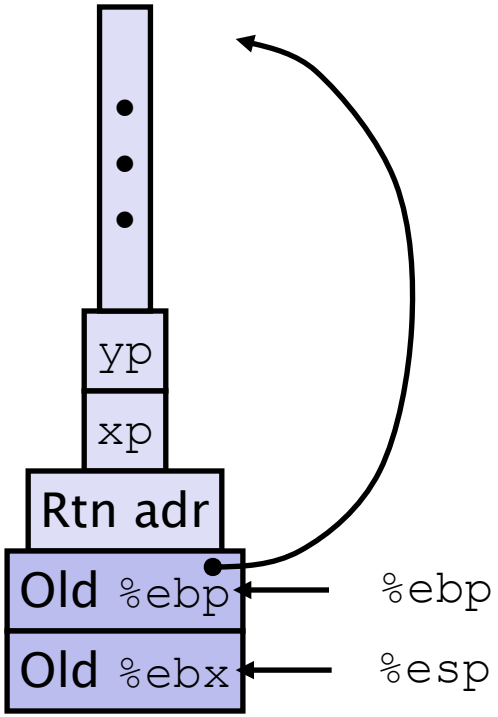


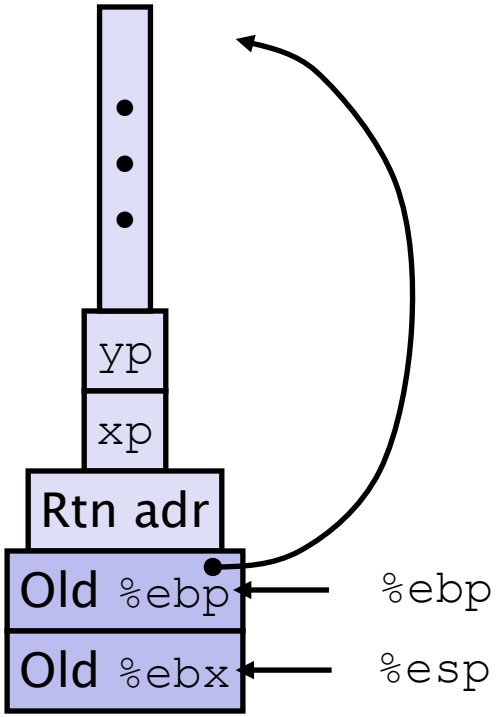

```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

Observation: Saved and restored register %ebx

swap Finish #2

swap's Stack

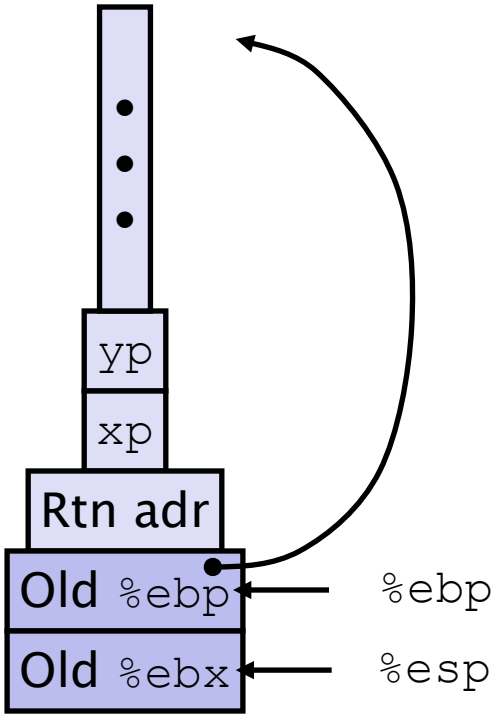




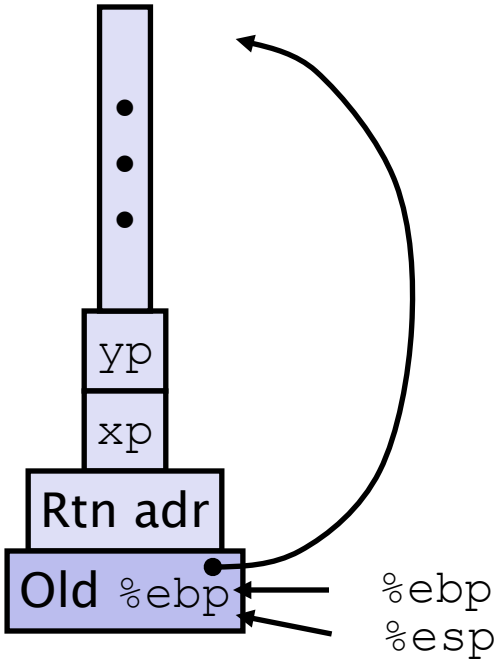

```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

swap Finish #2

`swap`'s Stack



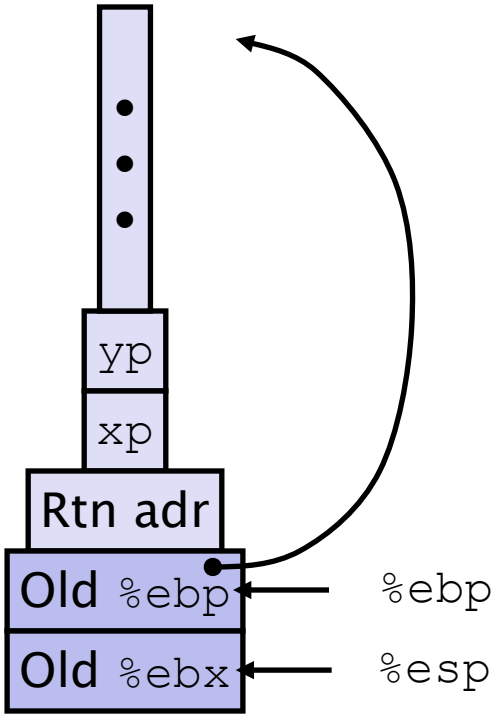
Resulting Stack

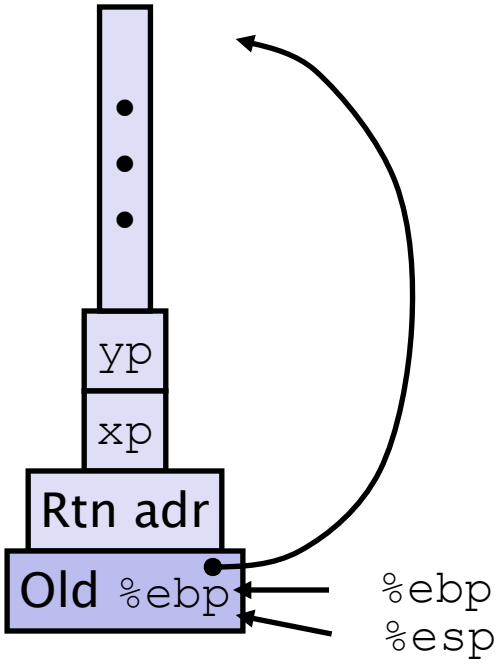



```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

swap Finish #2

swap's Stack

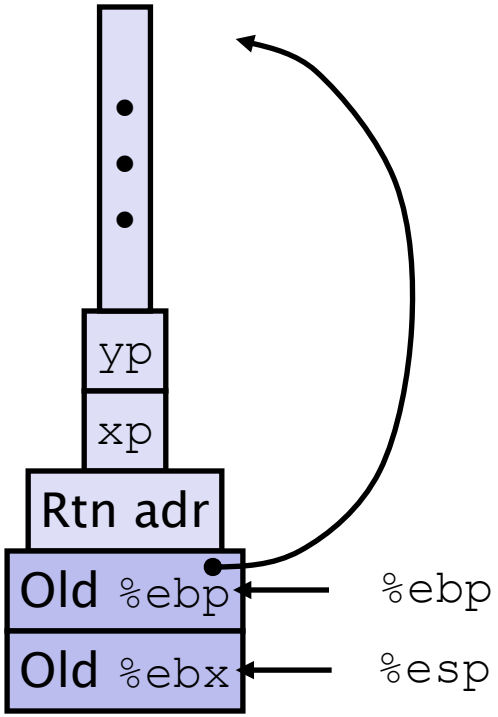





```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #3

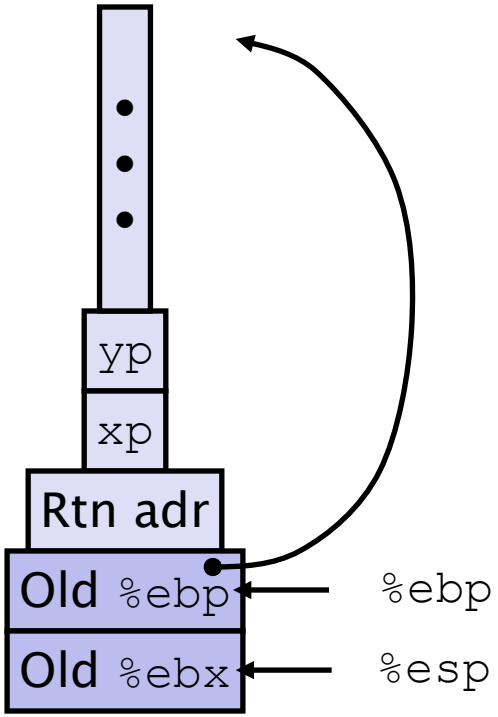
swap's Stack



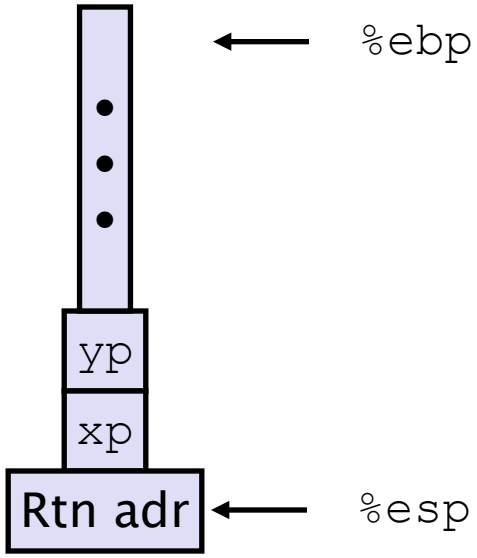

```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #3

swap's Stack



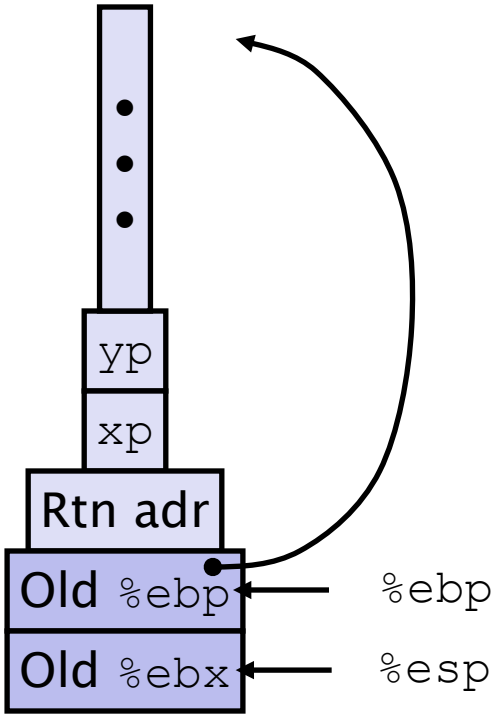
Resulting Stack

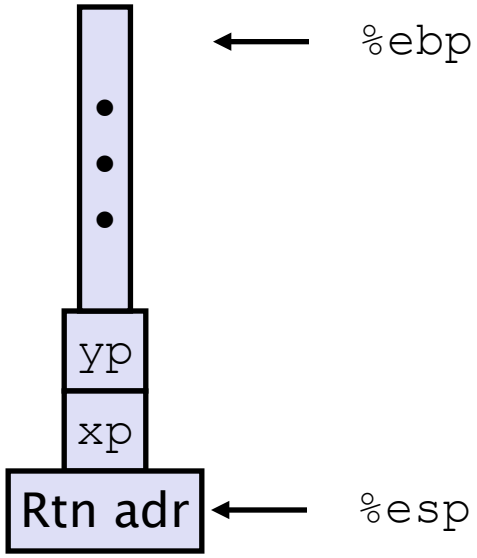



```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #4

swap's Stack

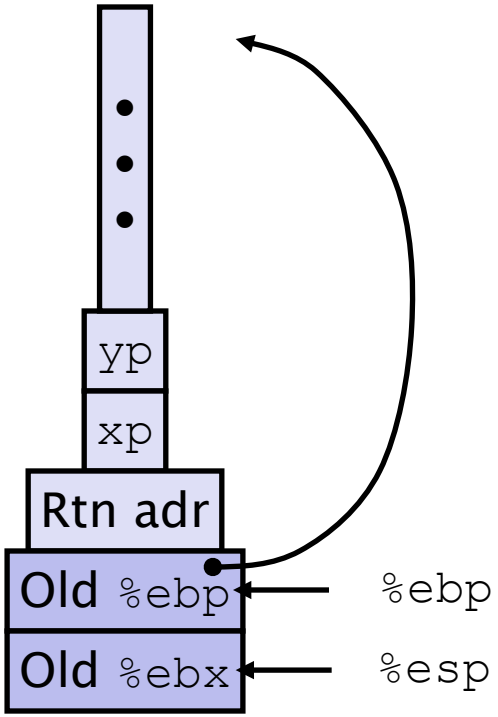




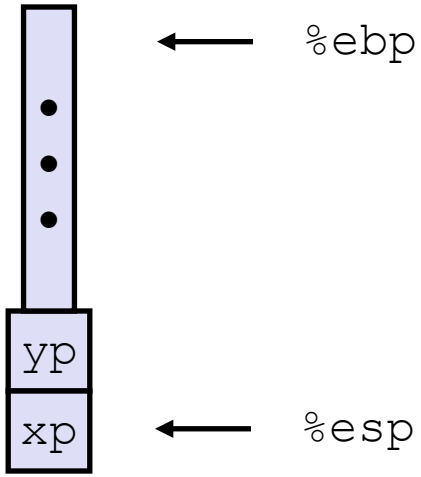

```
movl -4(%ebp),%ebx
  movl %ebp,%esp
    popl %ebp
       ret
```

swap Finish #4

swap's Stack



Resulting Stack




```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

- **Observation**
  - Saved & restored register `%ebx`
  - Didn't do so for `%eax`, `%ecx`, or `%edx`

Disassembled swap

```
080483a4 <swap>:
 80483a4:   55              push   %ebp
 80483a5:   89 e5           mov    %esp,%ebp
 80483a7:   53              push   %ebx
 80483a8:   8b 55 08        mov    0x8(%ebp),%edx
 80483ab:   8b 4d 0c        mov    0xc(%ebp),%ecx
 80483ae:   8b 1a           mov    (%edx),%ebx
 80483b0:   8b 01           mov    (%ecx),%eax
 80483b2:   89 02           mov    %eax,(%edx)
 80483b4:   89 19           mov    %ebx,(%ecx)
 80483b6:   5b              pop    %ebx
 80483b7:   c9              leave
 80483b8:   c3              ret
```

Calling Code

```
 8048409:   e8 96 ff ff ff   call 80483a4 <swap>
804840e:   8b 45 f8         mov  0xfffffff8(%ebp),%eax
```

Disassembled swap

```
080483a4 <swap>:
 80483a4:   55              push   %ebp
 80483a5:   89 e5           mov    %esp,%ebp
 80483a7:   53              push   %ebx
 80483a8:   8b 55 08        mov    0x8(%ebp),%edx
 80483ab:   8b 4d 0c        mov    0xc(%ebp),%ecx
 80483ae:   8b 1a           mov    (%edx),%ebx
 80483b0:   8b 01           mov    (%ecx),%eax
 80483b2:   89 02           mov    %eax,(%edx)
 80483b4:   89 19           mov    %ebx,(%ecx)
 80483b6:   5b              pop    %ebx
 80483b7:   c9              leave
 80483b8:   c3              ret
```

```
mov    %ebp,%esp
pop    %ebp
```

Calling Code

```
 8048409:   e8 96 ff ff ff   call 80483a4 <swap>
804840e:   8b 45 f8         mov  0xfffffff8(%ebp),%eax
```

Disassembled swap

```
080483a4 <swap>:
 80483a4:   55              push   %ebp
 80483a5:   89 e5           mov    %esp,%ebp
 80483a7:   53              push   %ebx
 80483a8:   8b 55 08        mov    0x8(%ebp),%edx
 80483ab:   8b 4d 0c        mov    0xc(%ebp),%ecx
 80483ae:   8b 1a           mov    (%edx),%ebx
 80483b0:   8b 01           mov    (%ecx),%eax
 80483b2:   89 02           mov    %eax,(%edx)
 80483b4:   89 19           mov    %ebx,(%ecx)
 80483b6:   5b              pop    %ebx
 80483b7:   c9              leave
 80483b8:   c3              ret
```

```
mov    %ebp,%esp
pop    %ebp
```

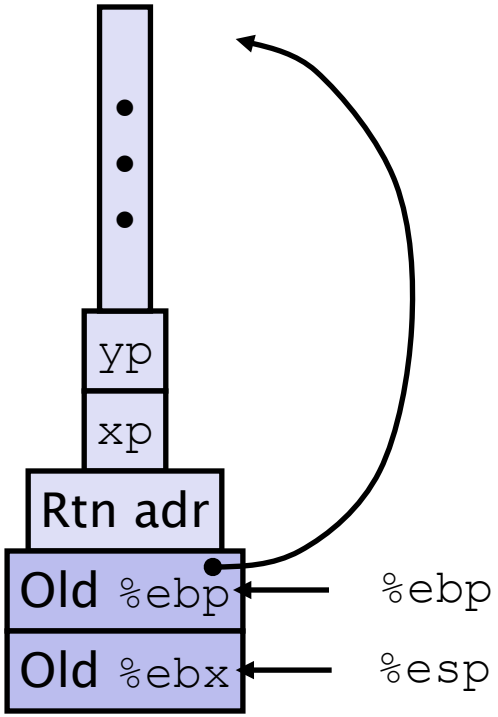
Calling Code

```
8048409:   e8 96 ff ff ff   call 80483a4 <swap>
804840e:   8b 45 f8         mov  0xfffffff8(%ebp),%eax
```

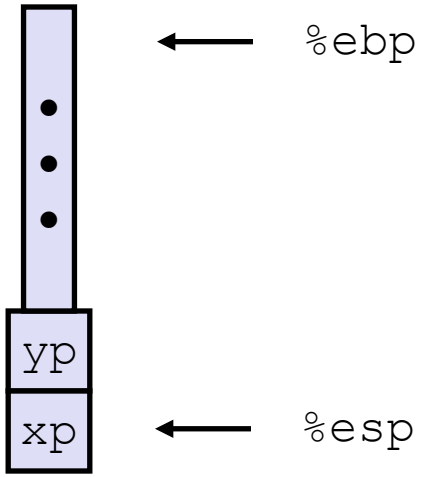
0x0804840e + 0xffffff96 = 0x080483a4

swap Finish #4

swap's Stack



Resulting Stack




```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

- **Observation**
  - Saved & restored register `%ebx`
  - Didn't do so for `%eax`, `%ecx`, or `%edx`

Register Saving Conventions

- **When procedure `yoo` calls `who`:**
  - `yoo` is the **caller**
  - `who` is the **callee**

- **Can Register be used for temporary storage?**

Register Saving Conventions

- **When procedure `yoo` calls `who`:**
  - `yoo` is the **caller**
  - `who` is the **callee**

- **Can Register be used for temporary storage?**

Register Saving Conventions

- **When procedure `yoo` calls `who`:**
  - `yoo` is the **caller**
  - `who` is the **callee**

- **Can Register be used for temporary storage?**

Register Saving Conventions

- **When procedure `yoo` calls `who`:**
  - `yoo` is the **caller**
  - `who` is the **callee**

- **Can Register be used for temporary storage?**

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 $98195, %edx
    • • •
    ret
```

  - Contents of register **`%edx`** overwritten by **`who`**

Saving registers

- When should you save them?
- When should you not save them?
  - Why not save all of them?

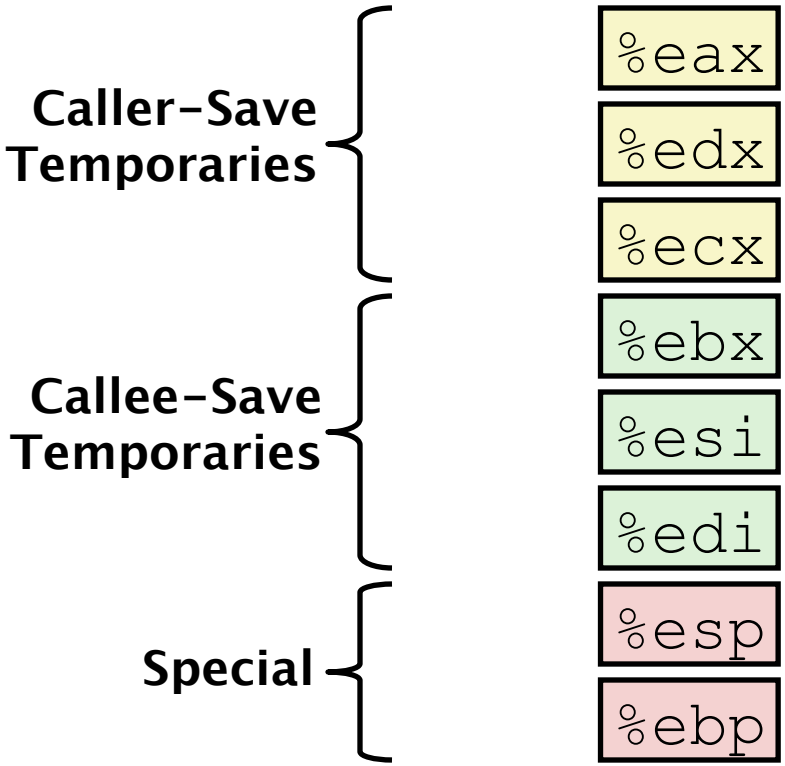
Register Saving Conventions

- **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 in its frame before calling
  - **"Callee Save"**
    - Callee saves temporary in its frame before using
- **Why do we have these conventions?**

IA32/Linux 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



Recursive Factorial

```
int rfact(int x)
{
  int rval;
  if (x <= 1)
    return 1;
  rval = rfact(x-1);
  return rval * x;
}
```

```
rfact:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ebx
    cmpl $1,%ebx
    jle .L78
    leal -1(%ebx),%eax
    pushl %eax
    call rfact
    imull %ebx,%eax
    jmp .L79
    .align 4
.L78:
    movl $1,%eax
.L79:
    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

Recursive Factorial

```
int rfact(int x)
{
  int rval;
  if (x <= 1)
    return 1;
  rval = rfact(x-1);
  return rval * x;
}
```

- **Registers**
  - **`%ebx`** used, but saved at beginning & restored at end
  - **`%eax`** used without first saving
    - expect caller to save
    - pushed onto stack as parameter for next call

```
rfact:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ebx
    cmpl $1,%ebx
    jle .L78
    leal -1(%ebx),%eax
    pushl %eax
    call rfact
    imull %ebx,%eax
    jmp .L79
    .align 4
.L78:
    movl $1,%eax
.L79:
    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

Pointer Code

Recursive Procedure

```
void s_helper
  (int x, int *accum)
{
  if (x <= 1)
    return;
  else {
    int z = *accum * x;
    *accum = z;
    s_helper (x-1,accum);
  }
}
```

Top-Level Call

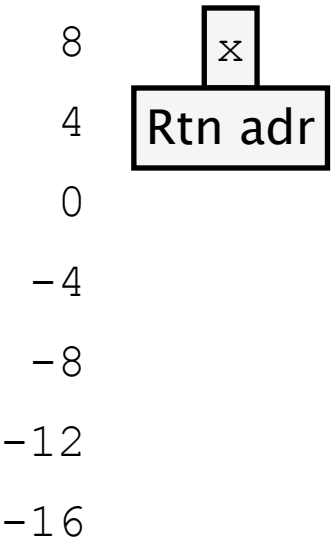
```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Pass pointer to update location**

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



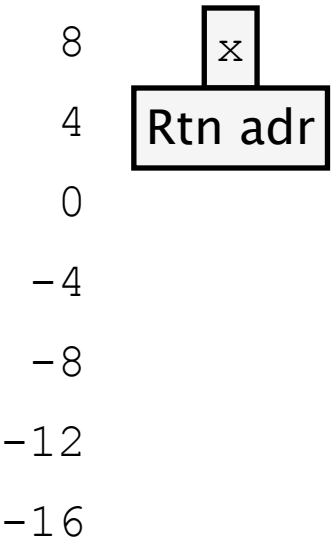
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



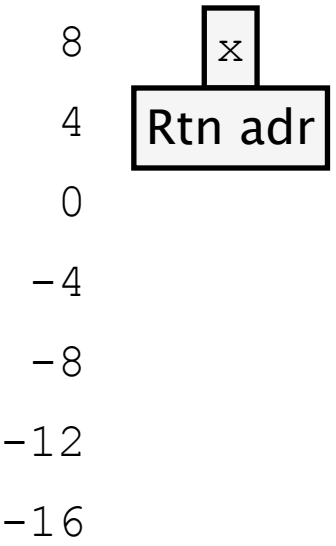
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



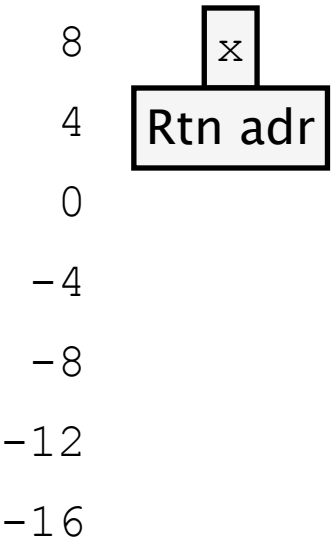
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



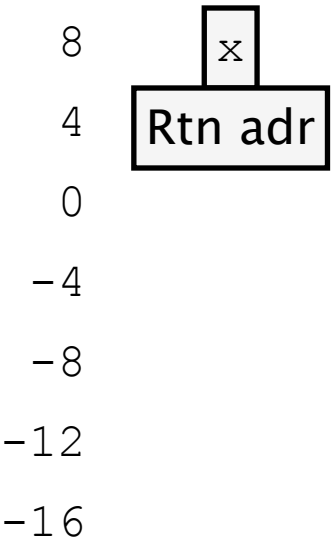
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



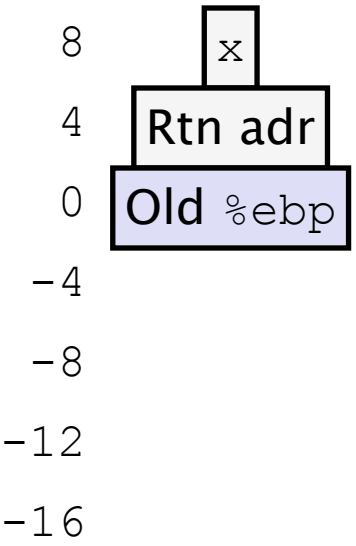
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



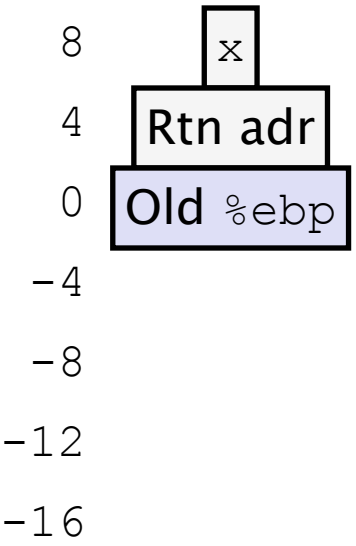
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



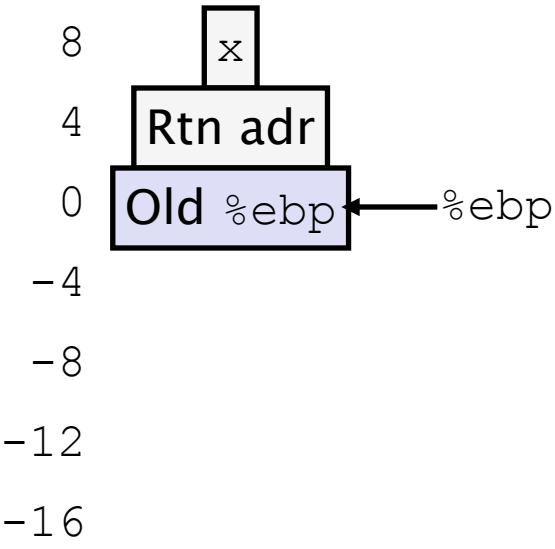
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



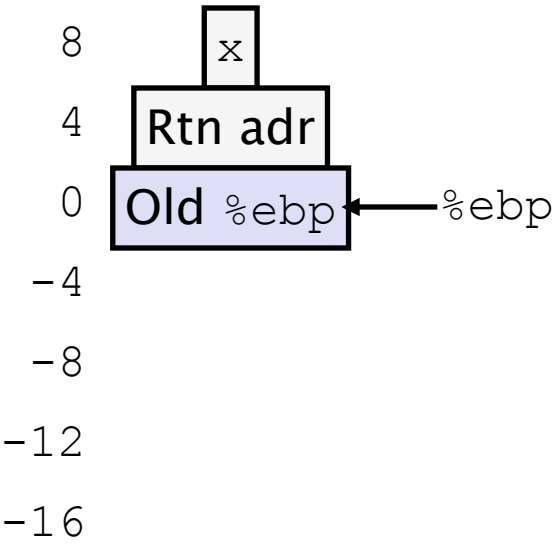
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



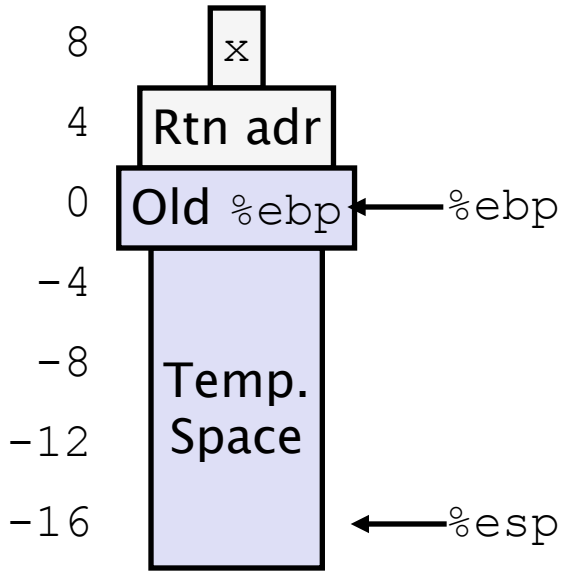
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

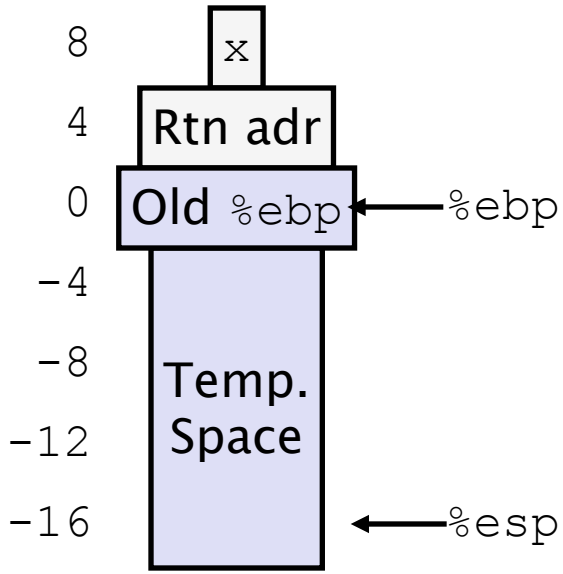
Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**

Initial part of `sfact`

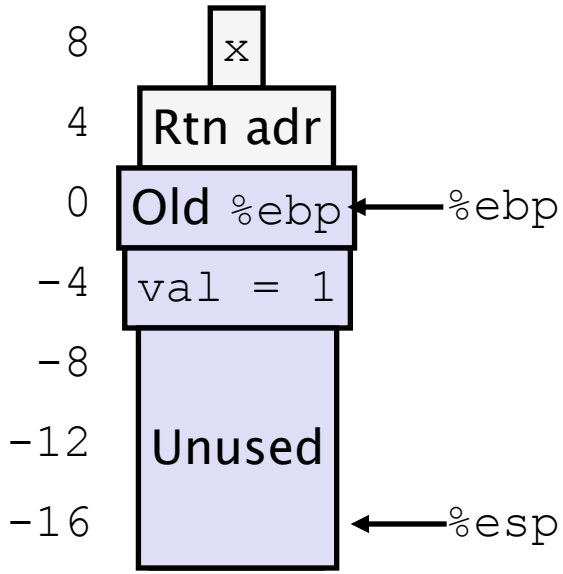
```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```



Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



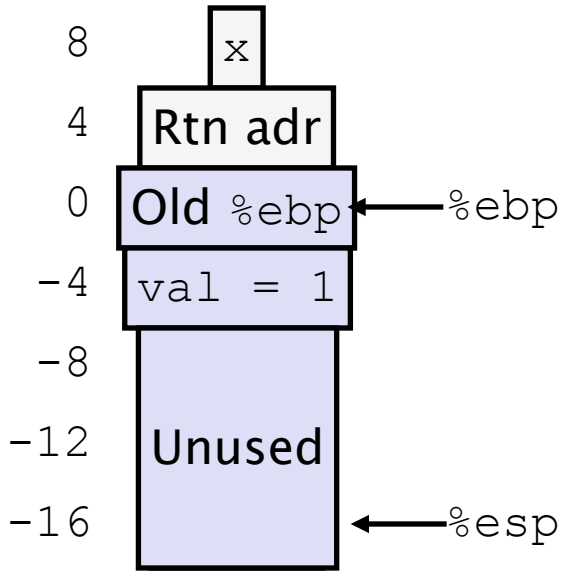
Initial part of `sfact`

```
_sfact:
   pushl %ebp
   movl %esp,%ebp
   subl $16,%esp
   movl 8(%ebp),%edx
   movl $1,-4(%ebp)
```

Creating & Initializing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

- **Variable `val` must be stored on stack**
  - Because: Need to create pointer to it
- **Compute pointer as `-4(%ebp)`**
- **Push on stack as second argument**



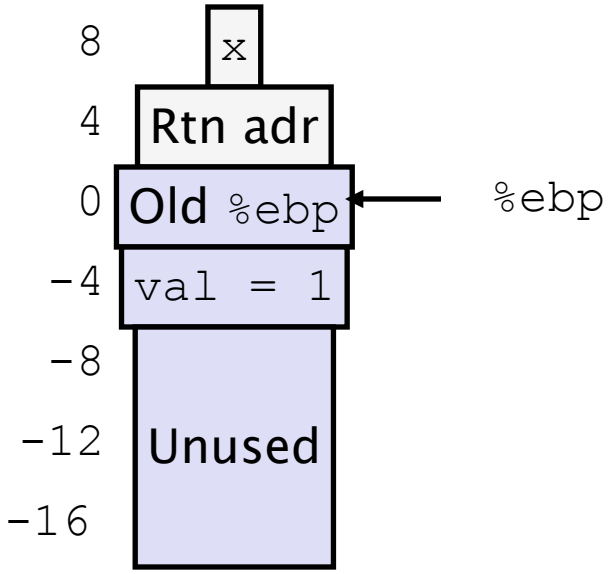
Initial part of `sfact`

```
_sfact:
   pushl %ebp          # Save %ebp
   movl %esp,%ebp      # Set %ebp
   subl $16,%esp       # Add 16 bytes
   movl 8(%ebp),%edx   # edx = x
   movl $1,-4(%ebp)    # val = 1
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



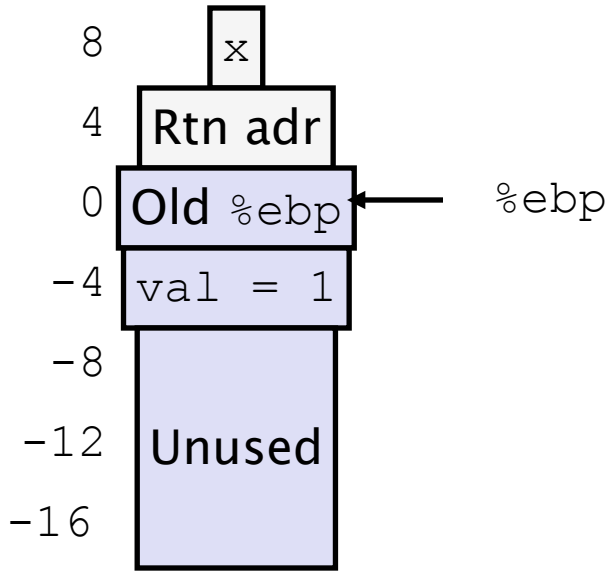
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



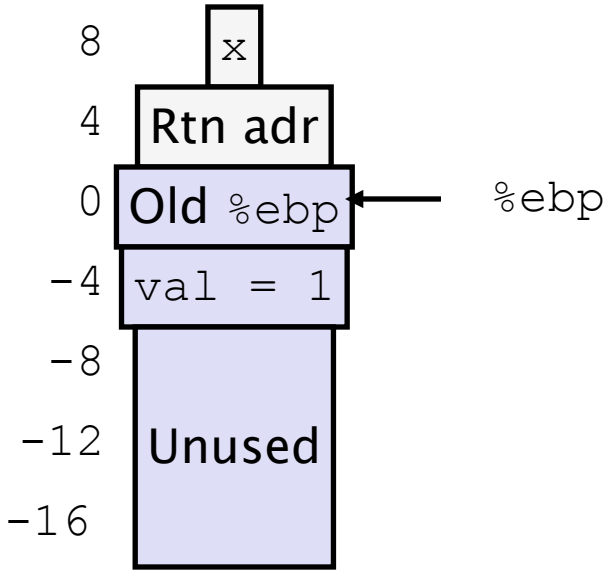
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



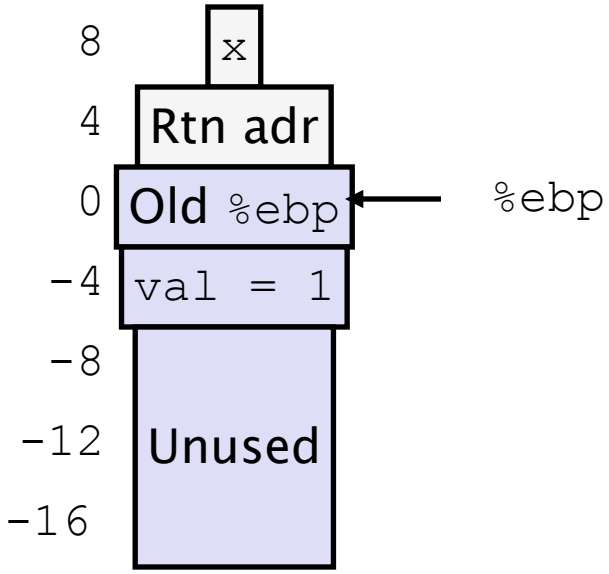
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



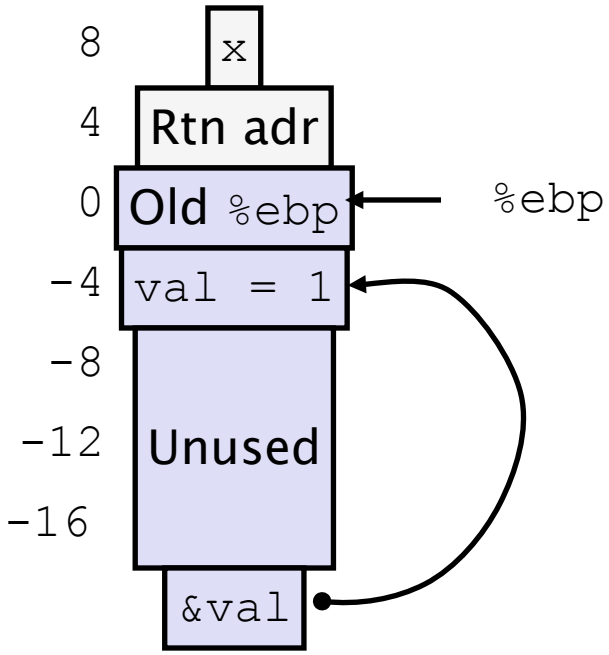
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



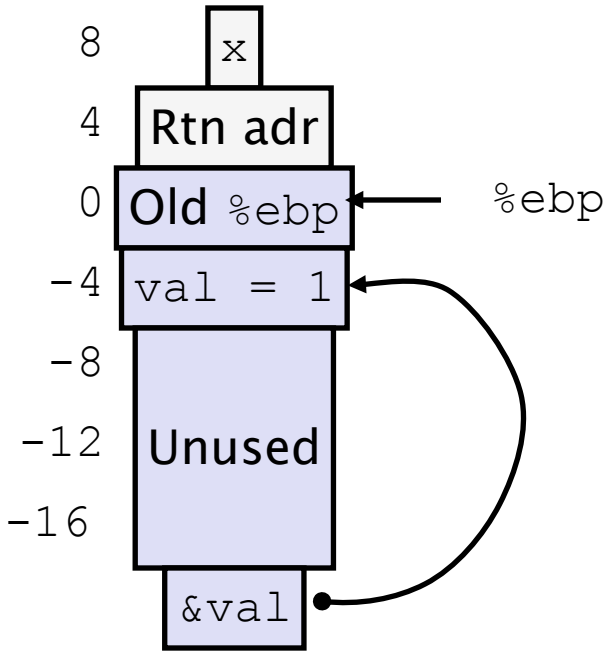
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



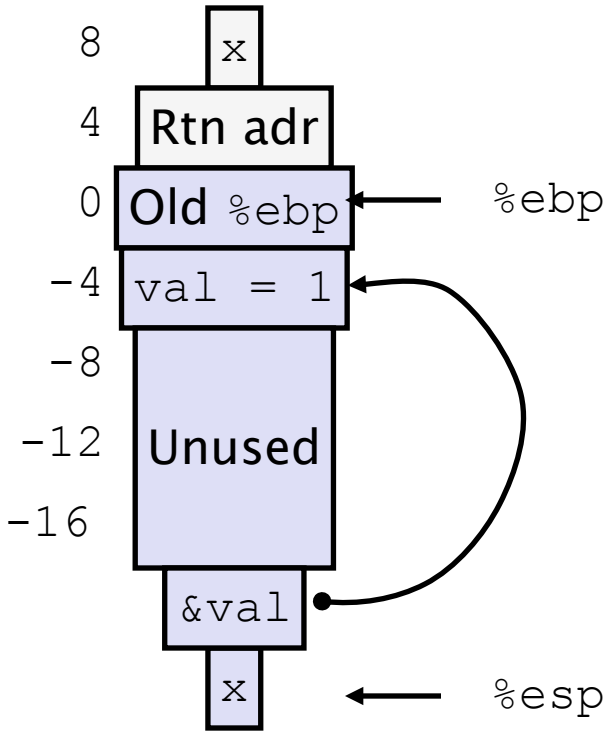
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



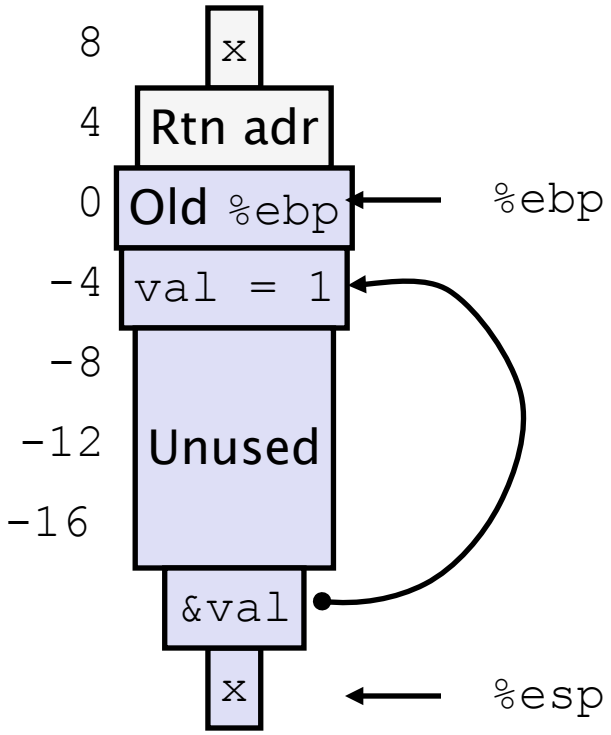
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



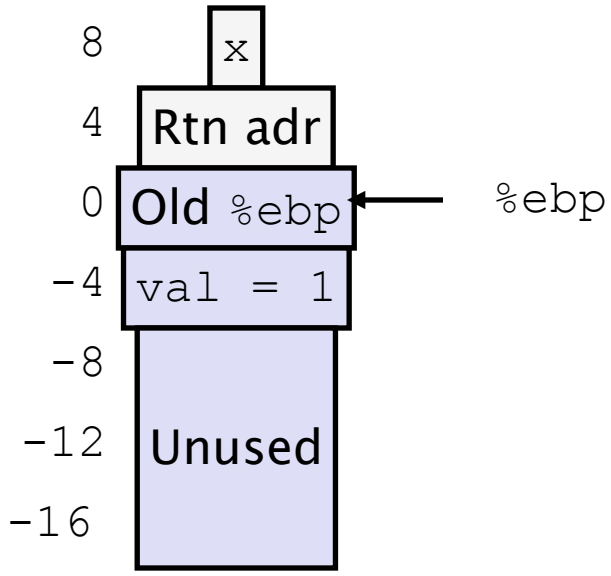
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



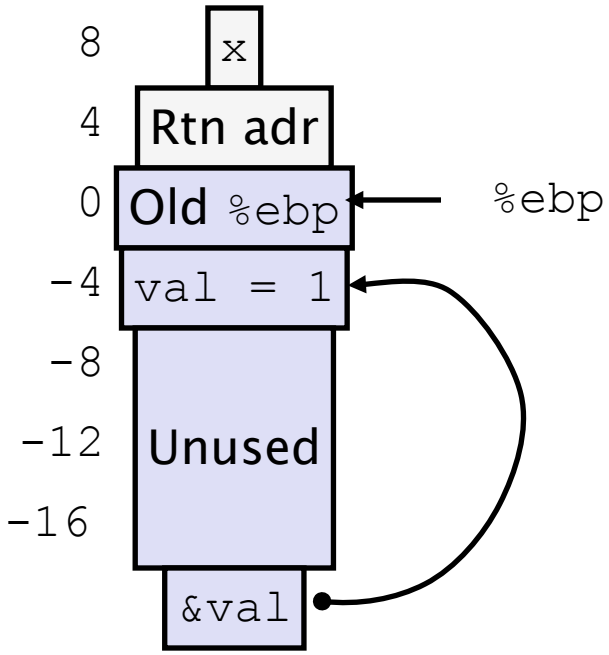
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



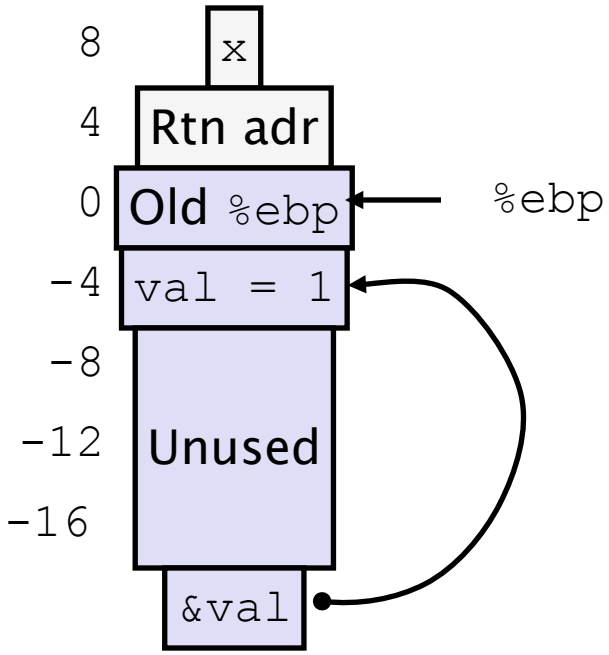
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



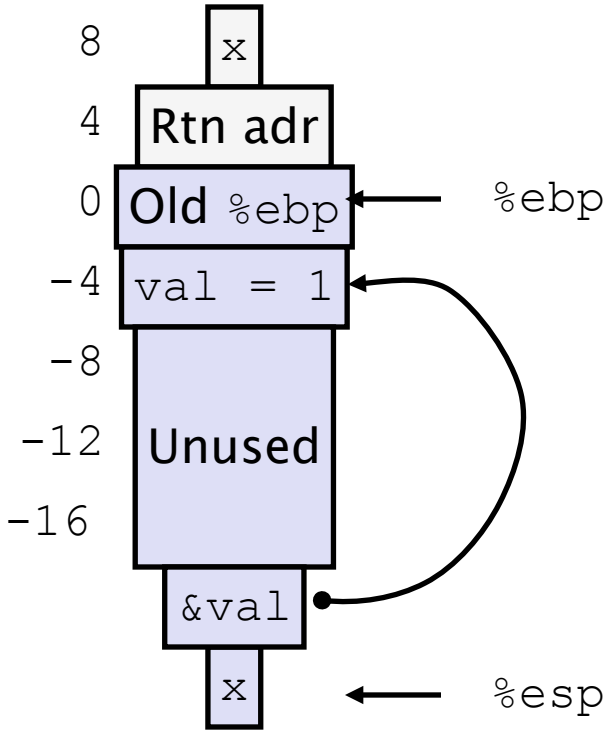
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



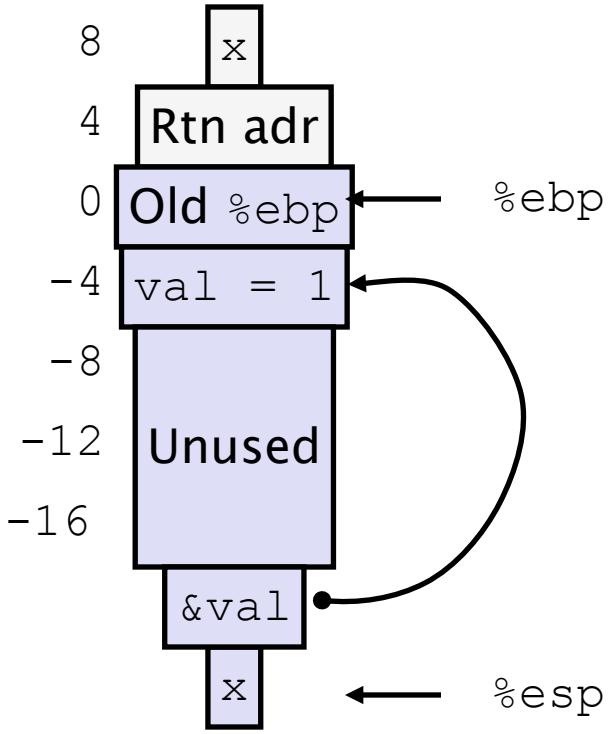
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



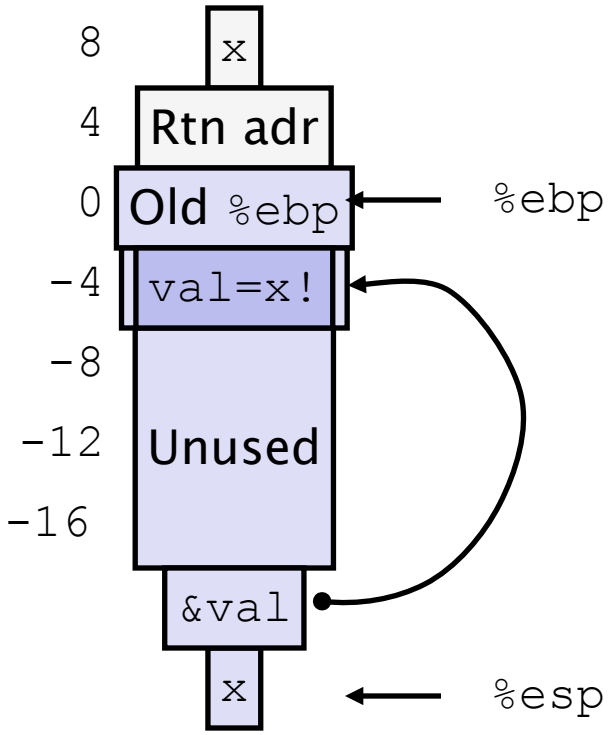
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

Stack at time of call



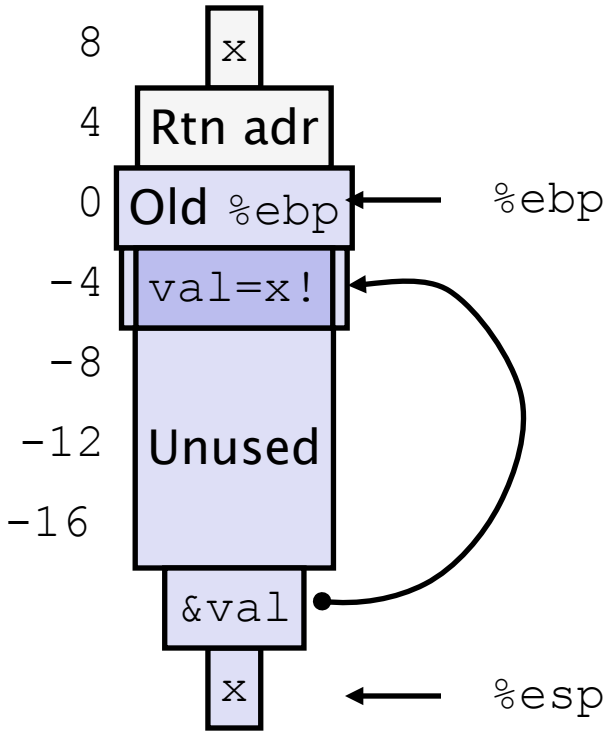
Calling `s_helper` from `sfact`

```
  leal -4(%ebp),%eax
  pushl %eax
  pushl %edx
  call s_helper
  movl -4(%ebp),%eax
  • • •
```

Passing Pointer

```
int sfact(int x)
{
  int val = 1;
  s_helper(x, &val);
  return val;
}
```

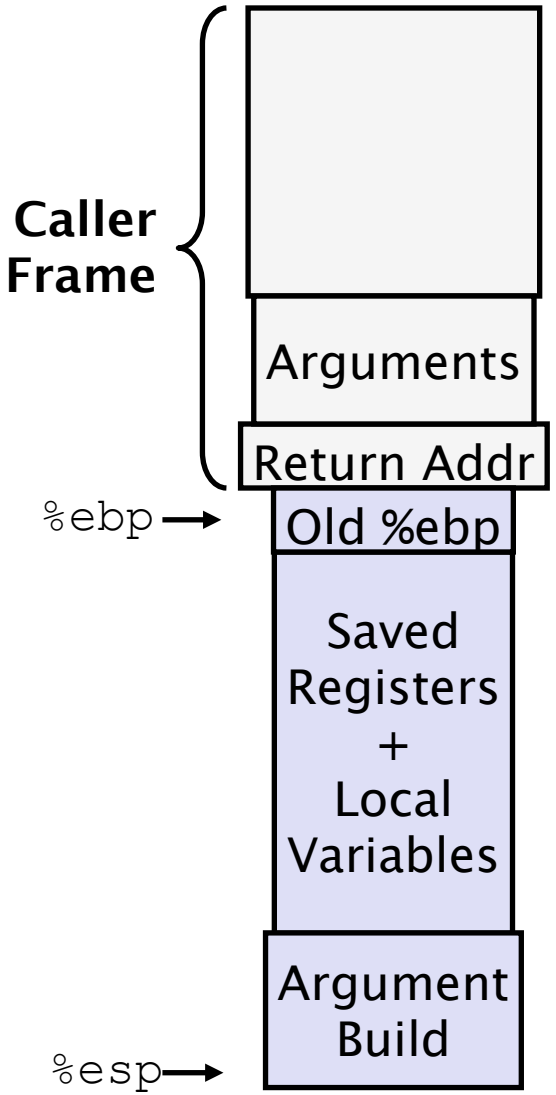
Stack at time of call



Calling `s_helper` from `sfact`

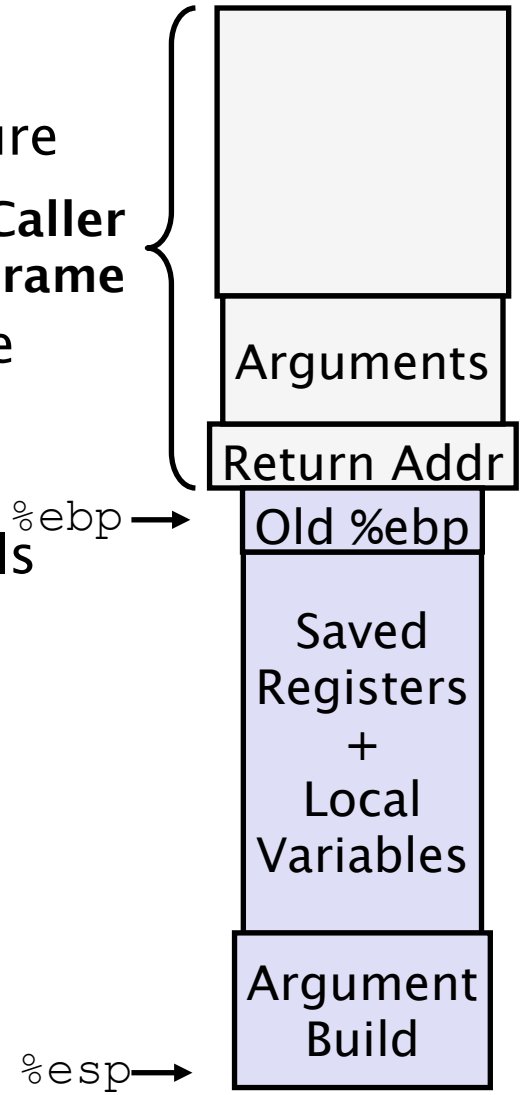
```
   leal -4(%ebp),%eax  # Compute &val
   pushl %eax          # Push on stack
   pushl %edx          # Push x
   call s_helper       # call
   movl -4(%ebp),%eax  # Return val
   • • •               # Finish
```

IA 32 Procedure Summary



IA 32 Procedure Summary

- **Stack makes recursion work**
  - Private storage for each instance of procedure call
    - Instantiations don't clobber each other
    - Addressing of locals + arguments can be relative to stack positions
  - Managed by stack discipline
    - Procedures return in inverse order of calls



IA 32 Procedure Summary

- **Stack makes recursion work**
  - Private storage for each instance of procedure call
    - Instantiations don't clobber each other
    - Addressing of locals + arguments can be relative to stack positions
  - Managed by stack discipline
    - Procedures return in inverse order of calls
- **IA32 procedures**
  Combination of Instructions + Conventions
  - call / ret instructions
  - Register usage conventions
    - caller / callee save
    - **`%ebp`** and **`%esp`**