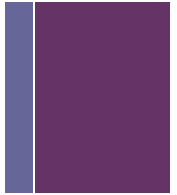


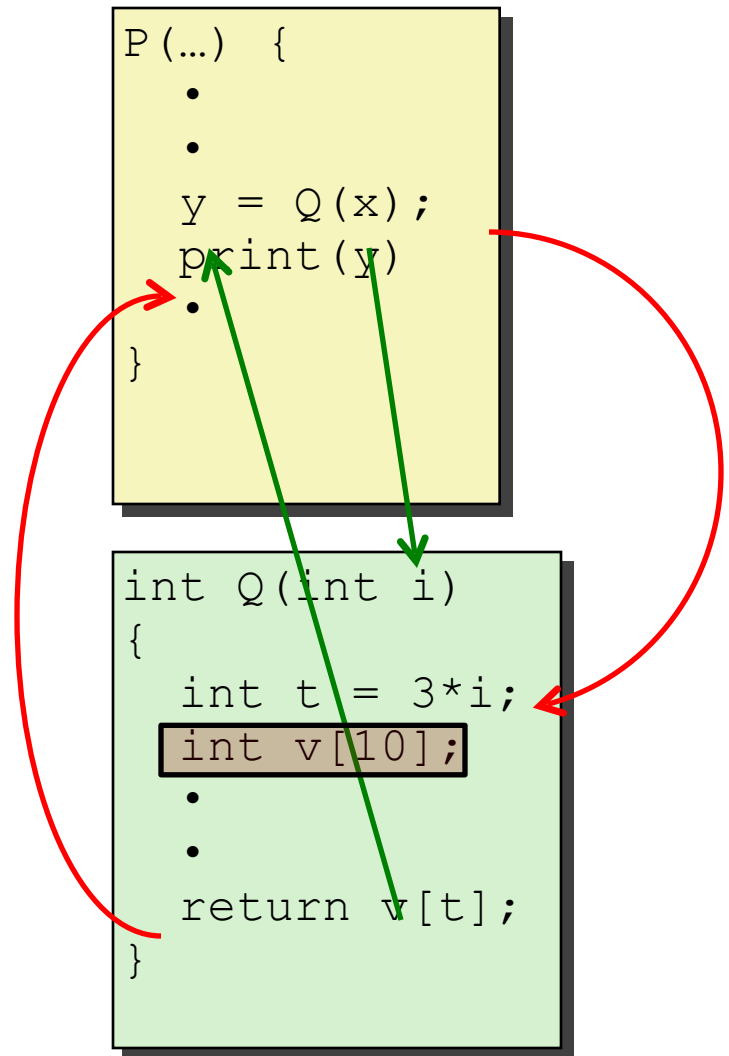


# Procedures

# + Mechanisms in Procedures

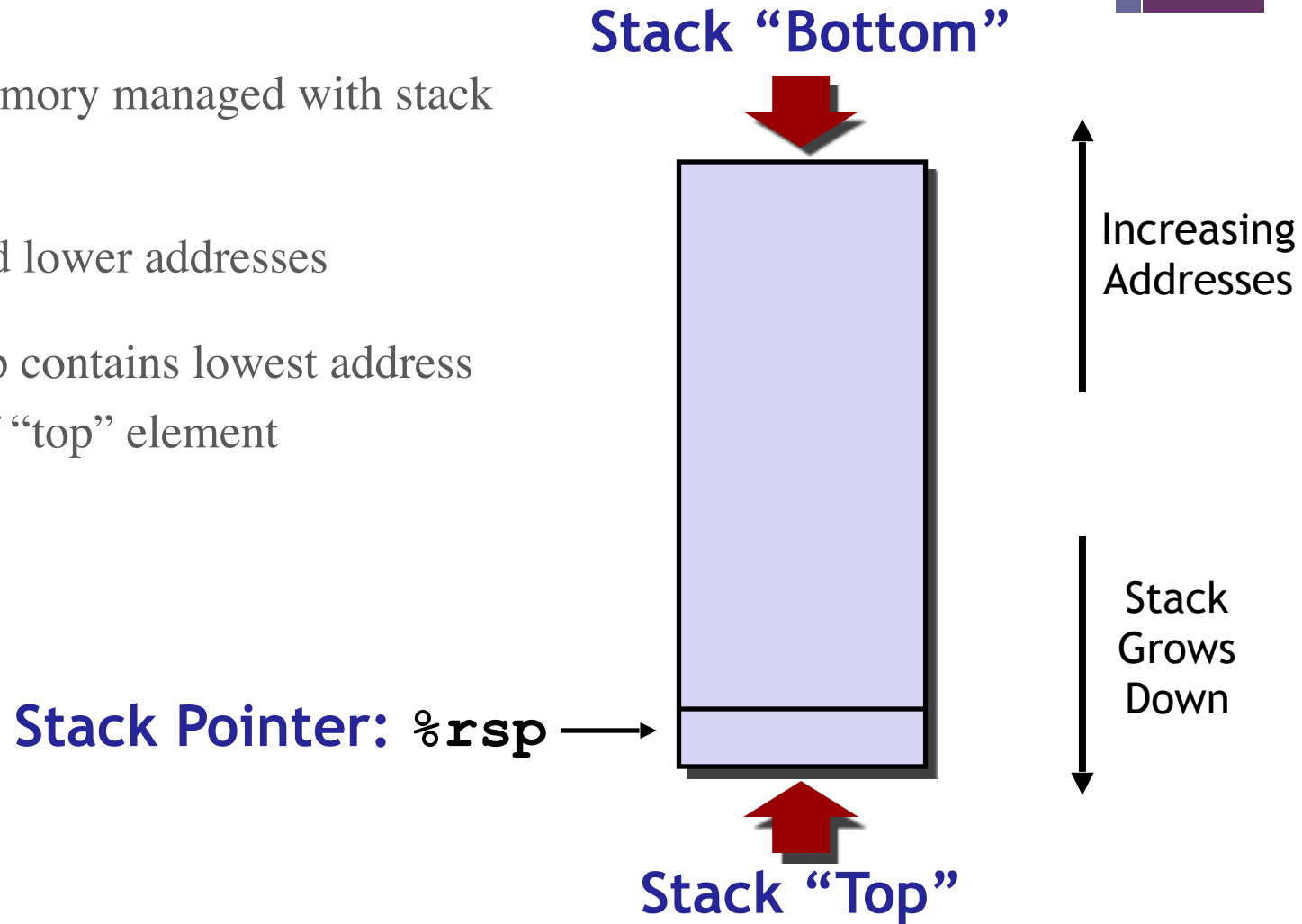


- **Passing control**
  - To beginning of procedure code
  - Back to return point
- **Passing data**
  - Procedure arguments
  - Return value
- **Memory management**
  - Allocate during procedure execution
  - Deallocate upon return
- **Mechanisms all implemented with machine instructions**
- **x86-64 implementation of a procedure uses only those mechanisms required**



# + x86-64 Stack

- Region of memory managed with stack discipline
- Grows toward lower addresses
- Register `%rsp` contains lowest address
  - address of “top” element



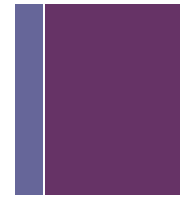
# + x86-64 Stack: Push

- `pushq src`
  - Fetch operand at *src*
  - Decrement `%rsp` by 8
  - Write operand at address given by `%rsp`

Stack Pointer: `%rsp`



Stack "Top"



Increasing  
Addresses

Stack  
Grows  
Down

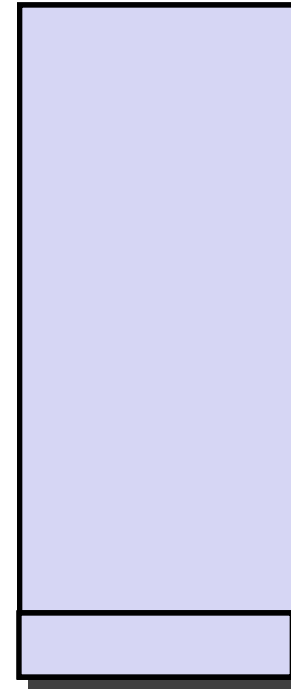
# + x86-64 Stack: Pop

- `popq dest`
  - Read value at address given by `%rsp`
  - Increment `%rsp` by 8 bytes
  - Store value at `dest` (must be register)

Stack Pointer: `%rsp`



Stack “Bottom”

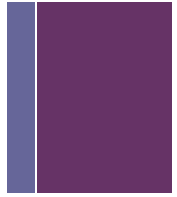


Increasing  
Addresses



Stack  
Grows  
Down

Stack “Top”





+

Passing Control

# + Code Examples



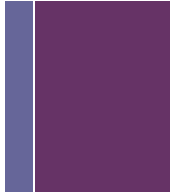
```
void multstore (long x, long y, long *dest){  
    long t = mult2(x, y);  
    *dest = t;  
}
```

```
0000000000400540 <multstore>:  
    400540: push    %rbx           # Save %rbx  
    400541: mov     %rdx,%rbx      # Save dest  
    400544: callq   400550 <mult2> # mult2(x,y)  
    400549: mov     %rax,(%rbx)    # Save at dest  
    40054c: pop     %rbx           # Restore %rbx  
    40054d: retq                    # Return
```

```
long mult2(long a, long b){  
    long s = a * b;  
    return s;  
}
```

```
0000000000400550 <mult2>:  
    400550: mov     %rdi,%rax      # a  
    400553: imul    %rsi,%rax      # a * b  
    400557: retq                    # return
```

# + Procedure Control Flow



- Use stack to support procedure call and return
- **Procedure `call` with label**
  - Pushes *return address* on stack
    - Address of the next instruction right after call
  - Jumps to *label*
- **Procedure return: `ret`**
  - Pops *return address* from stack
  - Jumps to *return address*



# + Control Flow Example #1



```
00000000000400540 <multstore>:
```

•  
•  
•

```
400544: callq 400550 <mult2>
```

```
400549: mov    %rax, (%rbx)
```

•  
•

```
00000000000400550 <mult2>:
```

```
400550: mov    %rdi,%rax
```

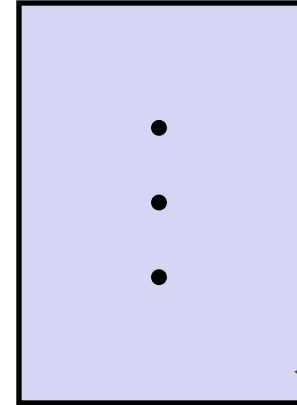
•  
•

```
400557: ret
```

0x130

0x128

0x120

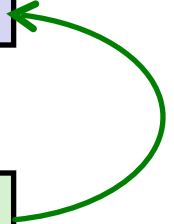


%rsp

0x120

%rip

0x400544



# + Control Flow Example #2



```
00000000000400540 <multstore>:
```

•  
•  
•  
•  
•

```
400544: callq 400550 <mult2>
```

```
400549: mov    %rax, (%rbx)
```

0x130

0x128

0x120

0x118

0x400549

%rsp

0x118

%rip

0x400550

```
00000000000400550 <mult2>:
```

```
400550: mov    %rdi,%rax
```

•  
•

```
400557: ret
```

# + Control Flow Example #3



00000000000400540 <multstore>:

•  
•  
•  
•  
•

400544: callq 400550 <mult2>

400549: mov %rax, (%rbx)

0x130

0x128

0x120

0x118

0x400549

%rsp

0x118

%rip

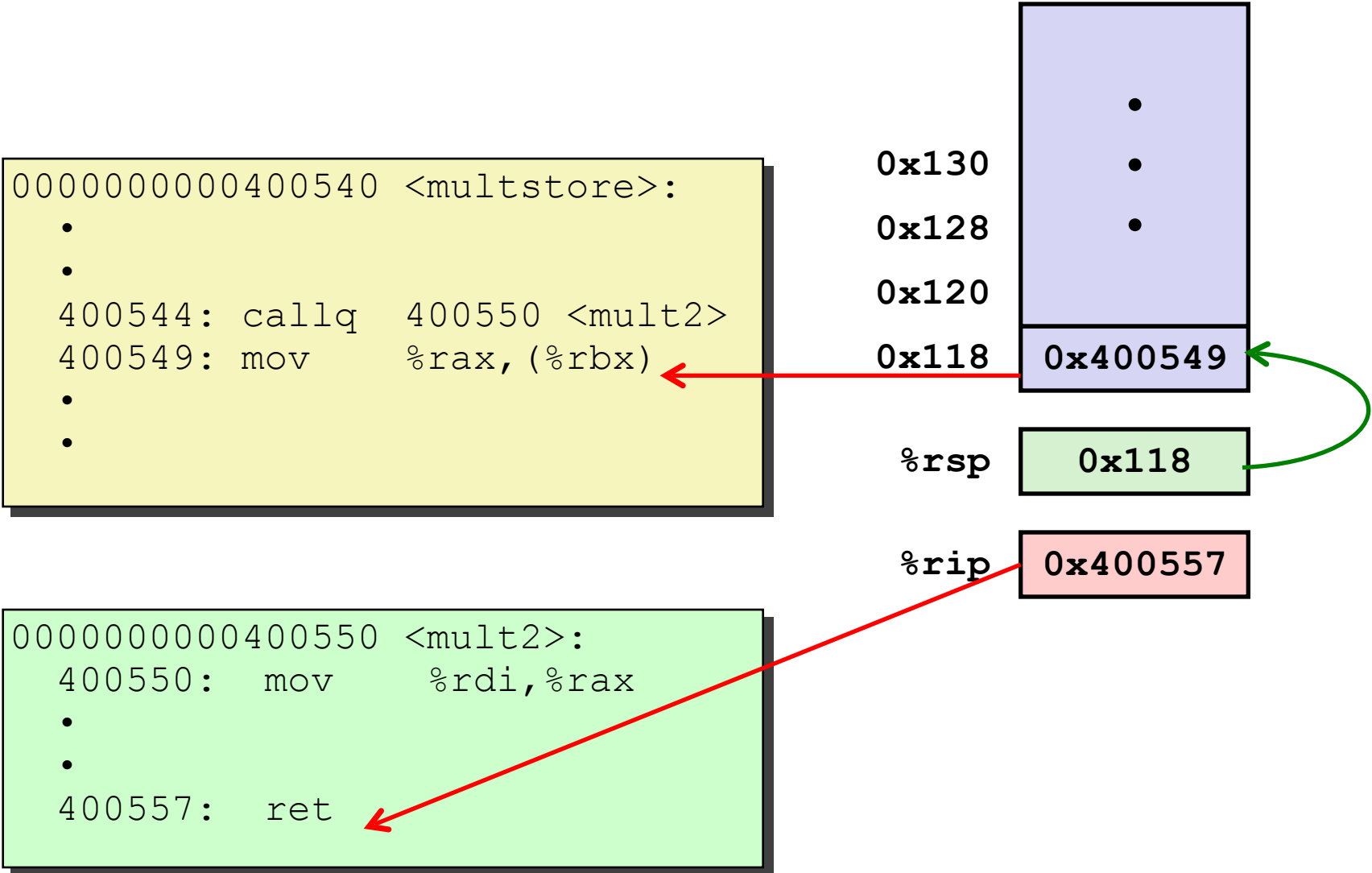
0x400557

00000000000400550 <mult2>:

400550: mov %rdi, %rax

•  
•

400557: ret



# + Control Flow Example #4

```
00000000000400540 <multstore>:
```

•  
•  
•  
•  
•

```
400544: callq 400550 <mult2>
```

```
400549: mov    %rax, (%rbx)
```

```
00000000000400550 <mult2>:
```

```
400550: mov    %rdi, %rax
```

•  
•

```
400557: ret
```

0x130

0x128

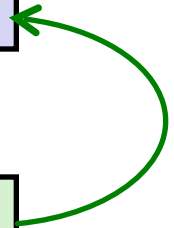
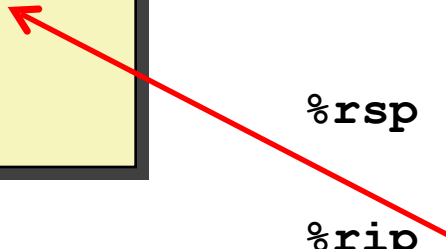
0x120

%rsp

0x120

%rip

0x400549





# Passing Data

# + Procedure Data Flow



## Registers

First 6 arguments

<code>%rdi</code>
<code>%rsi</code>
<code>%rdx</code>
<code>%rcx</code>
<code>%r8</code>
<code>%r9</code>

Return value

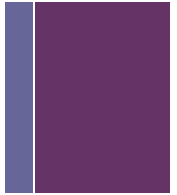
<code>%rax</code>
-------------------

## Stack

...
Arg <i>n</i>
...
Arg 8
Arg 7

- ❖ Only allocate stack space when needed

# + Data Flow Examples



```
void multstore (long x, long y, long *dest){  
    long t = mult2(x, y);  
    *dest = t;  
}
```

```
0000000000400540 <multstore>:  
    # x in %rdi, y in %rsi, dest in %rdx  
    . . .  
400541: mov     %rdx,%rbx      # Save dest  
400544: callq   400550 <mult2>  # mult2(x,y)  
    # t in %rax  
400549: mov     %rax,(%rbx)     # Save at dest  
    . . .
```

```
long mult2(long a, long b){  
    long s = a * b;  
    return s;  
}
```

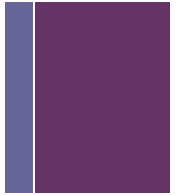
```
0000000000400550 <mult2>:  
    # a in %rdi, b in %rsi  
400550:  mov     %rdi,%rax      # a  
400553:  imul    %rsi,%rax      # a * b  
    # s in %rax  
400557:  ret                    # return
```



# Managing Local Data



# + Stack Frames



- **Functions have “instantiations”**
  - Every function call is a distinct execution with distinct data.
  - Need some place to store state of each instantiation
    - Arguments
    - Local variables
    - Return pointer (next instruction in caller)
- **Stack allocated in *frames***
  - State for single procedure instantiation
  - Moreover, an allocation of memory holding all the data for some function call.
- **Recursion**
  - Supported by this idea of *instantiation* and *stack discipline*.

# + Call Chain Example

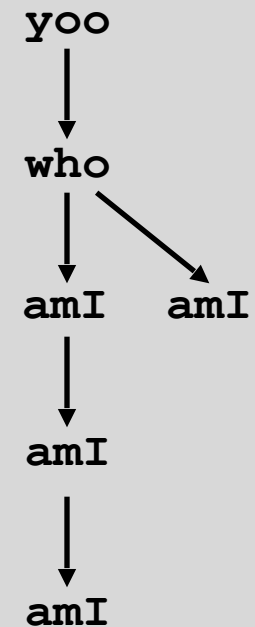


```
yoo (...)  
{  
  .  
  .  
  who ();  
  .  
  .  
}
```

```
who (...)  
{  
  . . .  
  amI ();  
  . . .  
  amI ();  
  . . .  
}
```

```
amI (...)  
{  
  .  
  .  
  amI ();  
  .  
  .  
}
```

## Example Call Chain



Procedure `amI ()` is recursive

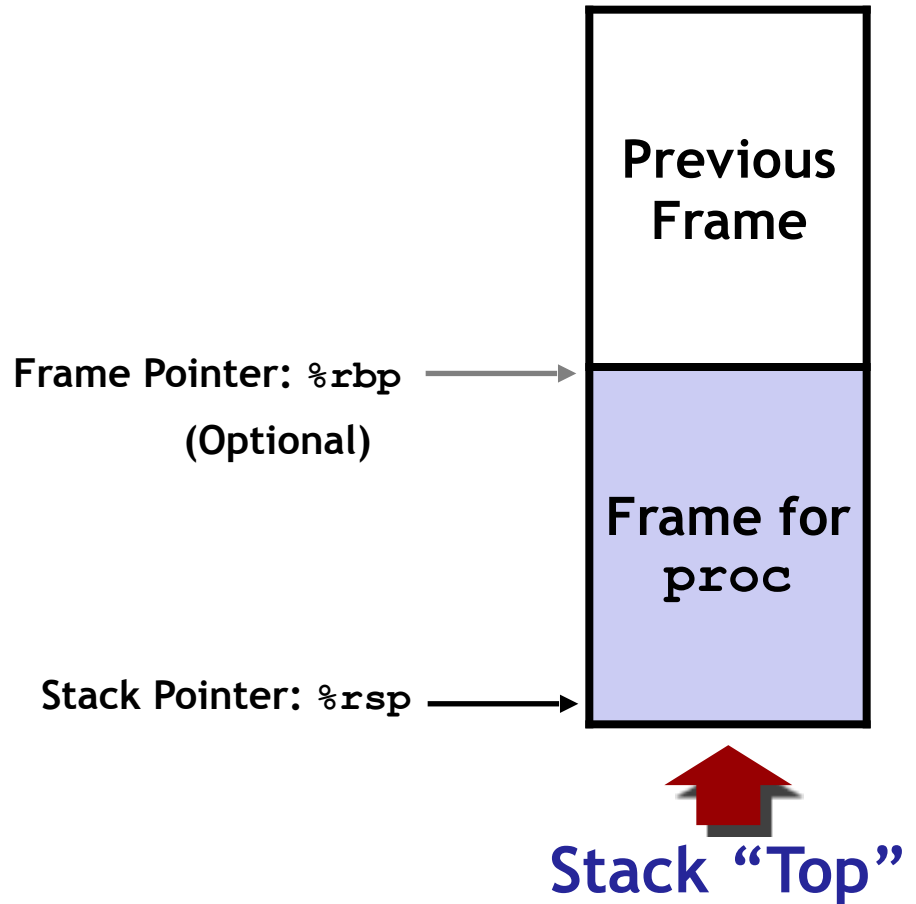
# + Stack Frames

- **Contents**

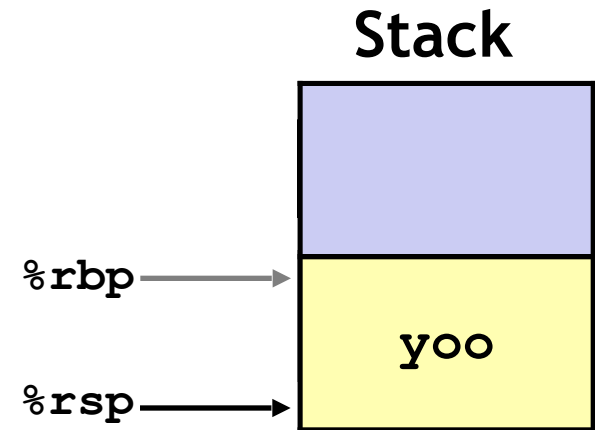
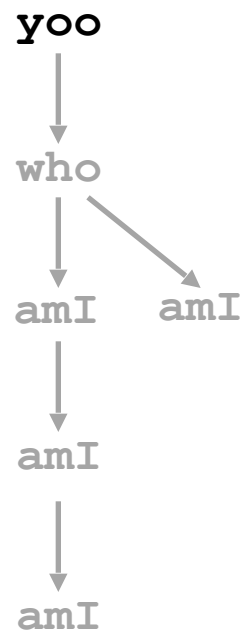
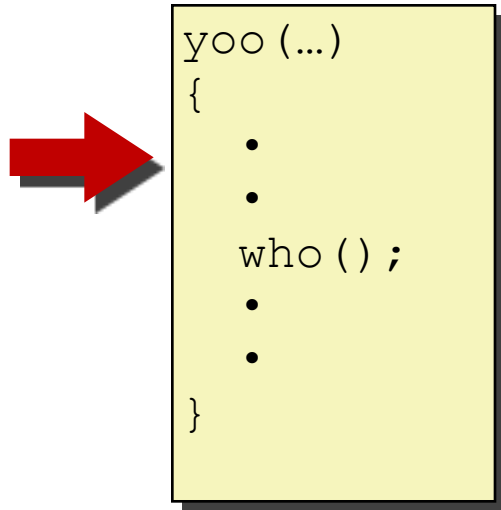
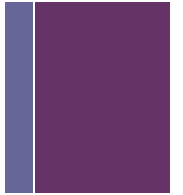
- Return information
- Local storage (if needed)

- **Management**

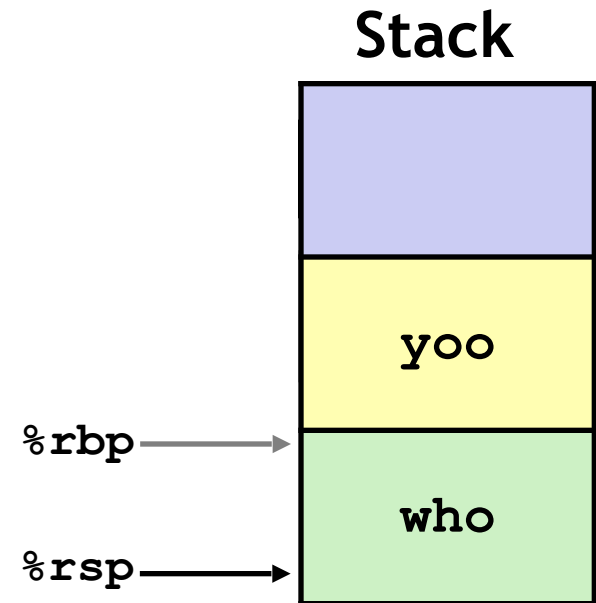
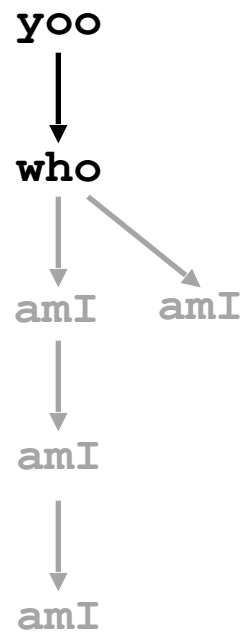
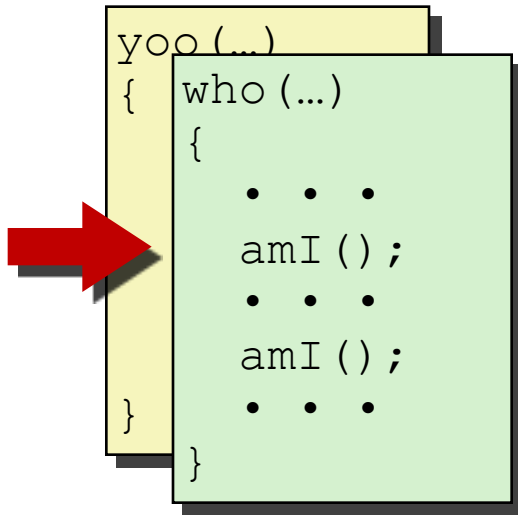
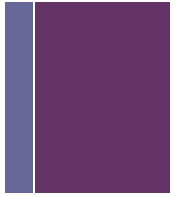
- Space allocated on procedure call
  - push by **call** instruction
- Space deallocated on return
  - pop by **ret** instruction



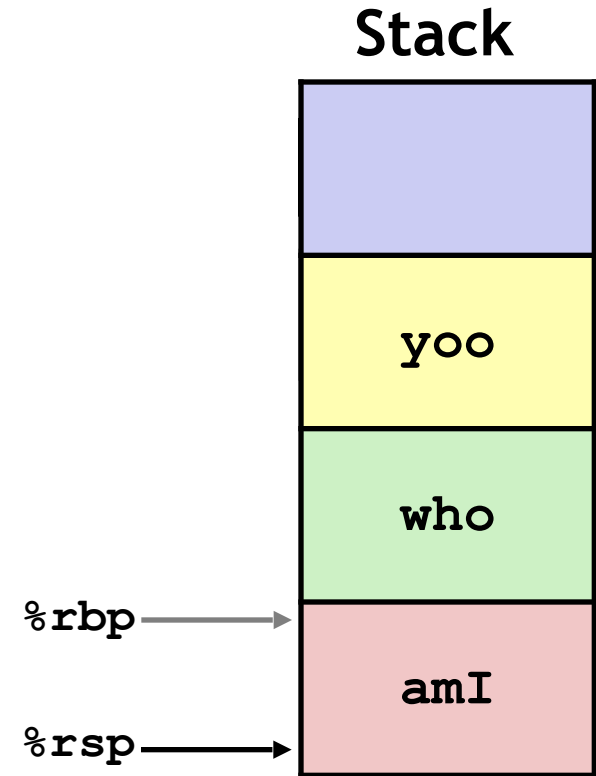
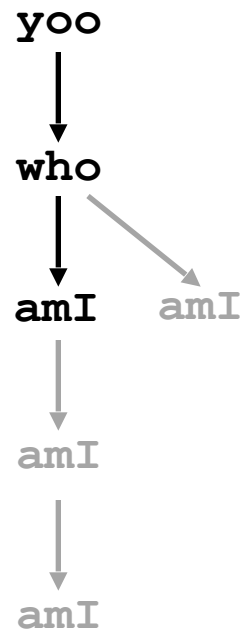
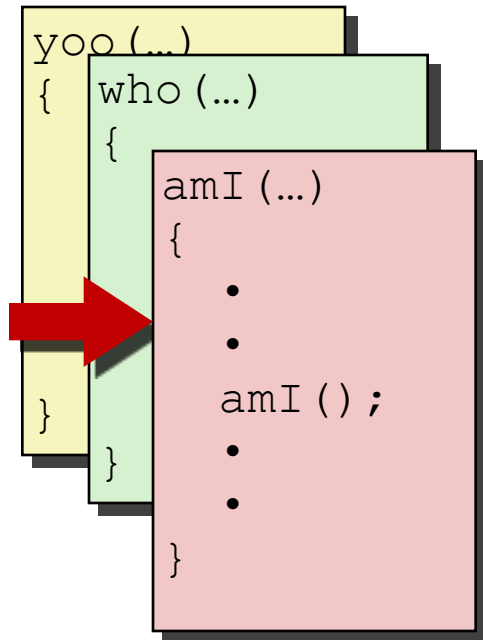
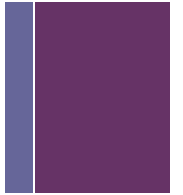
# + Example



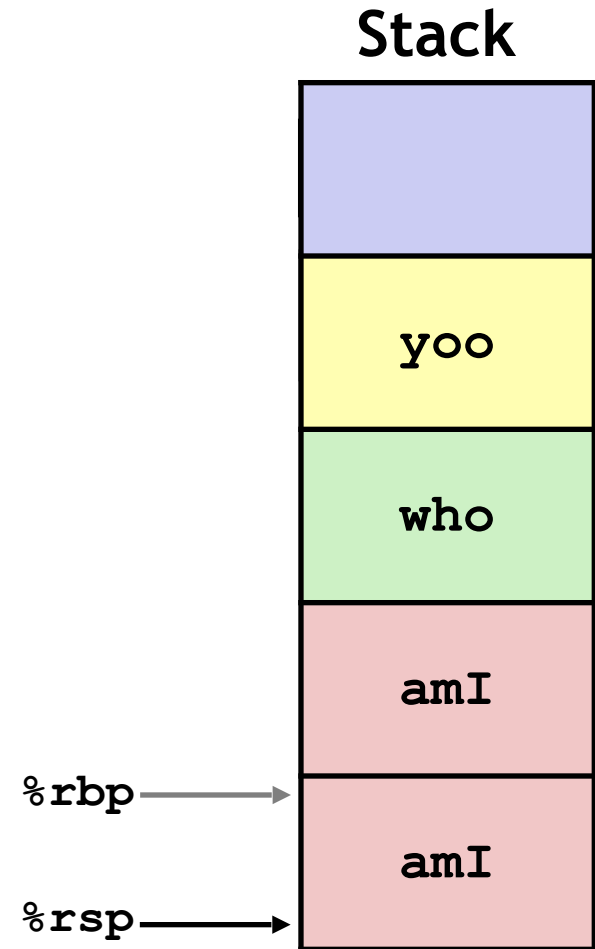
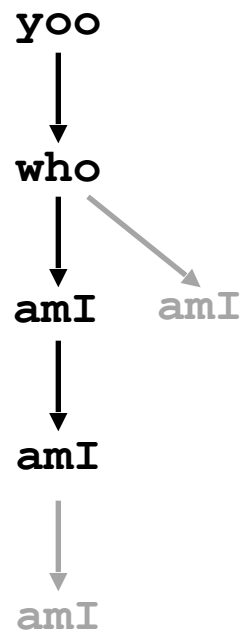
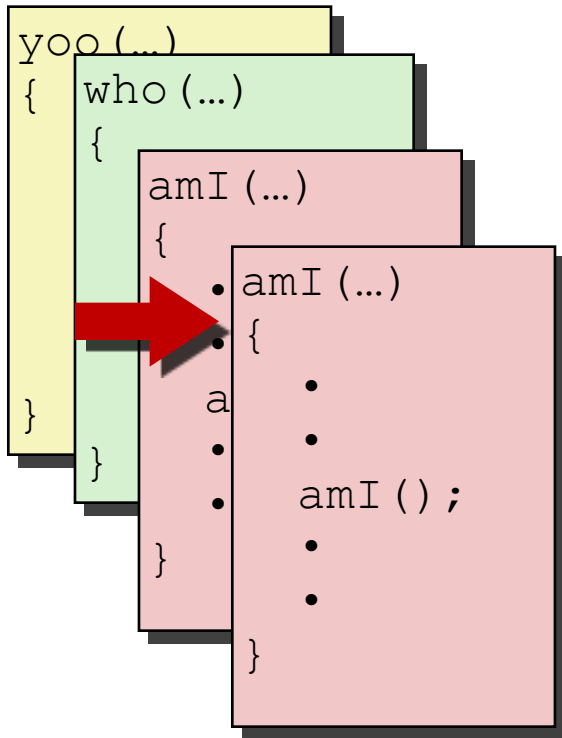
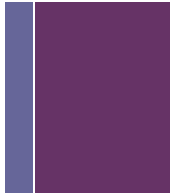
# + Example

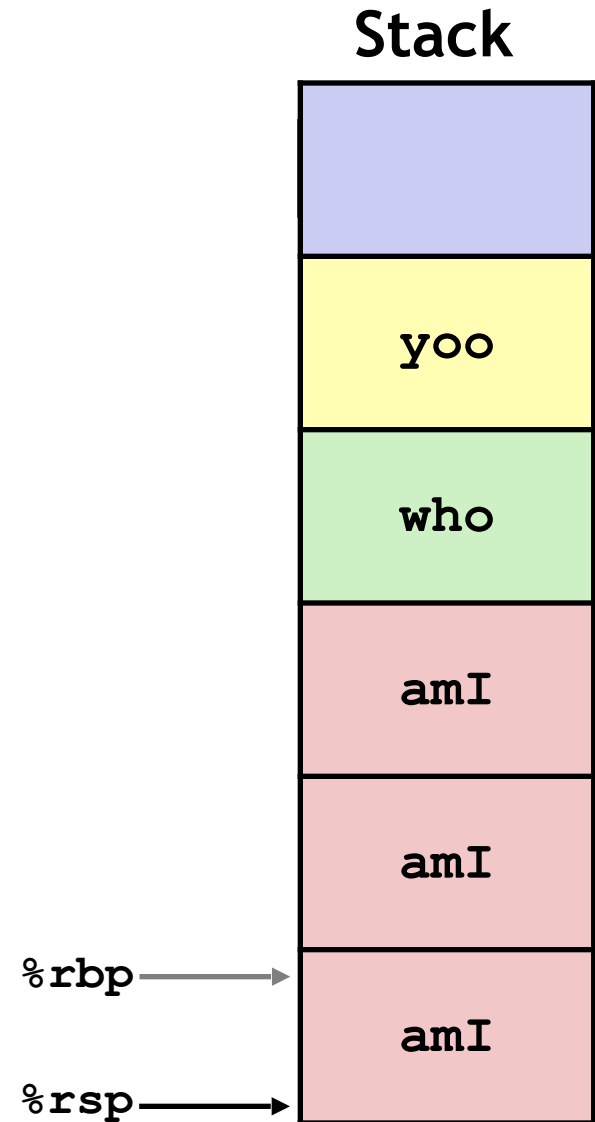
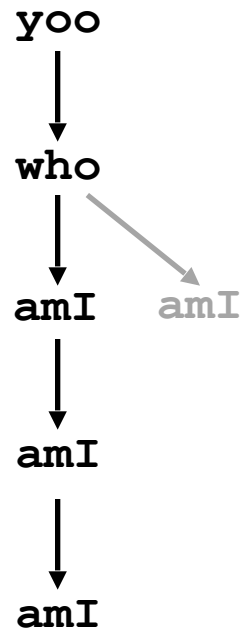


# + Example



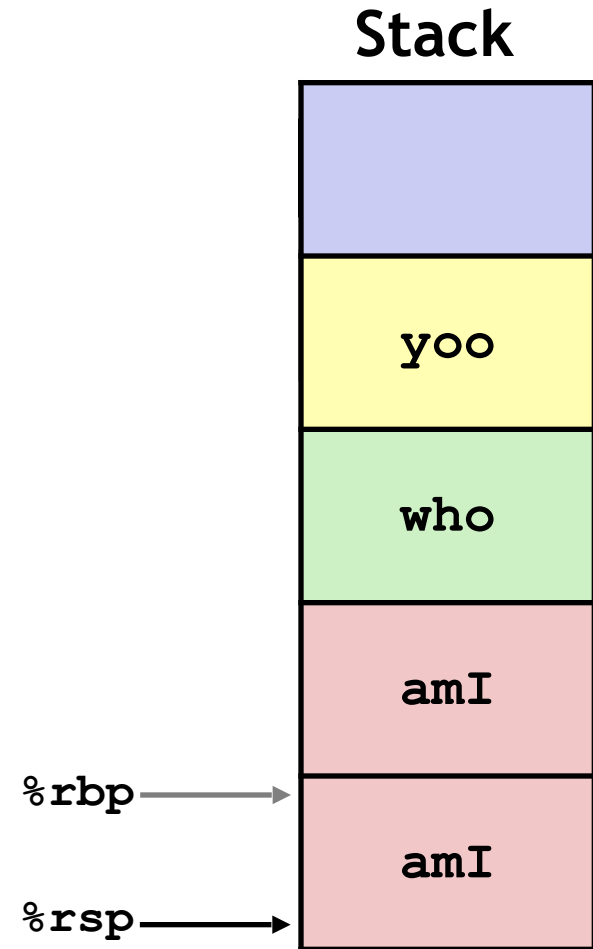
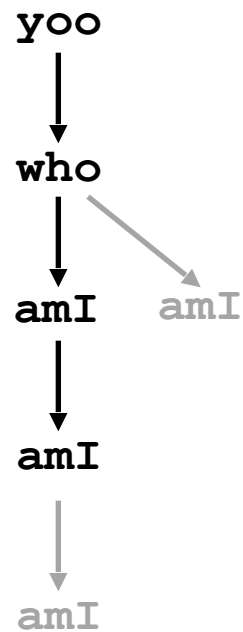
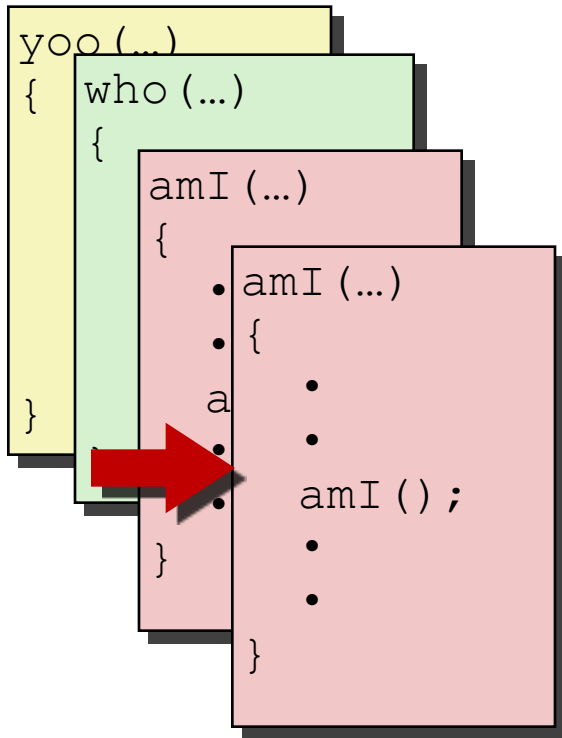
# + Example



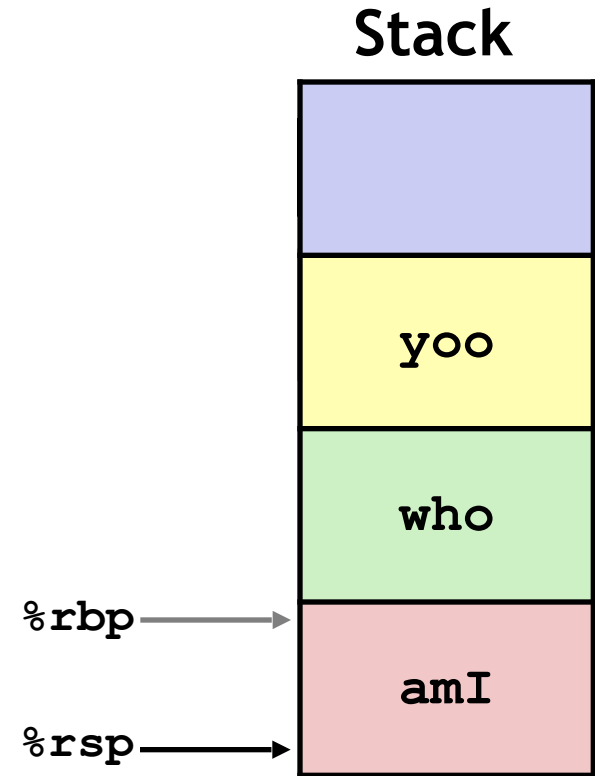
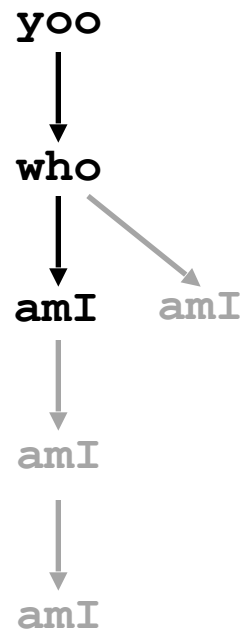
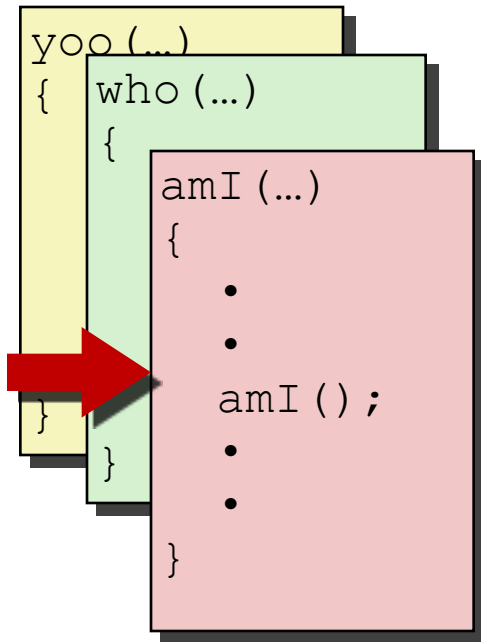
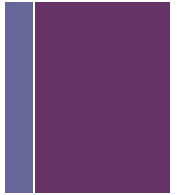




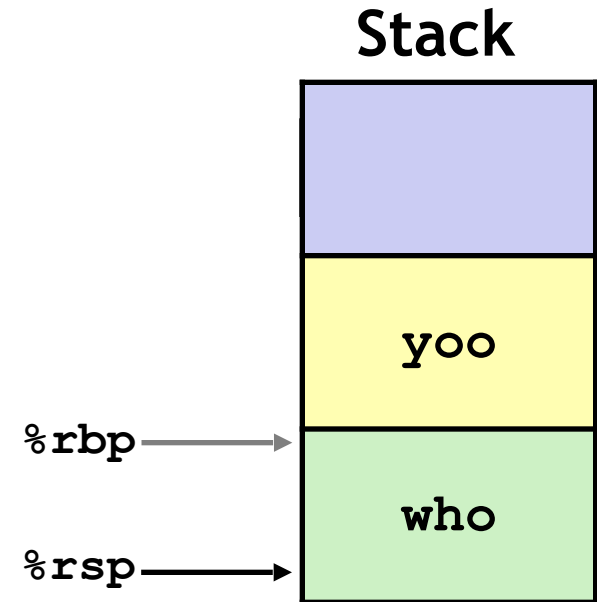
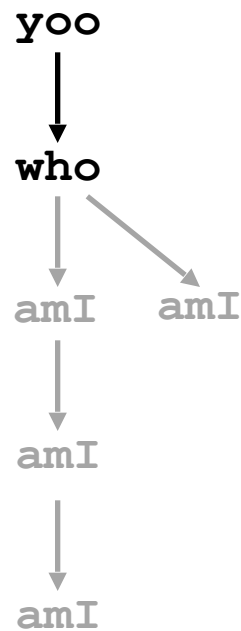
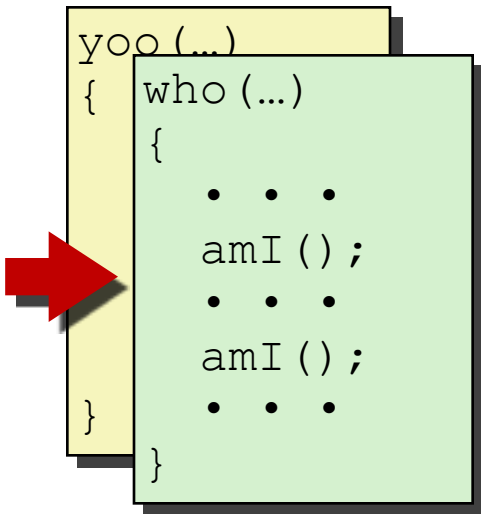
# + Example



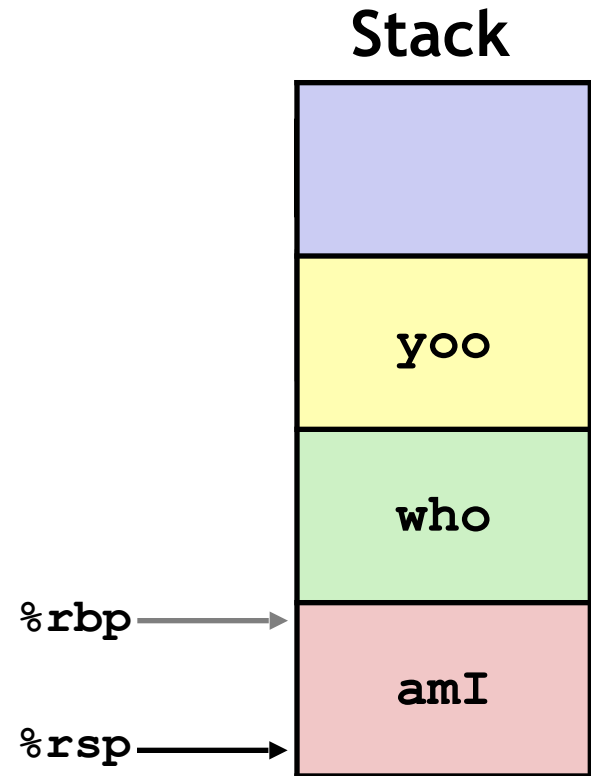
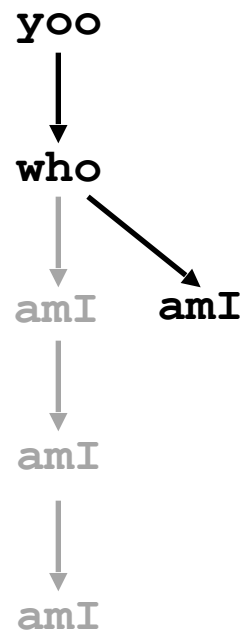
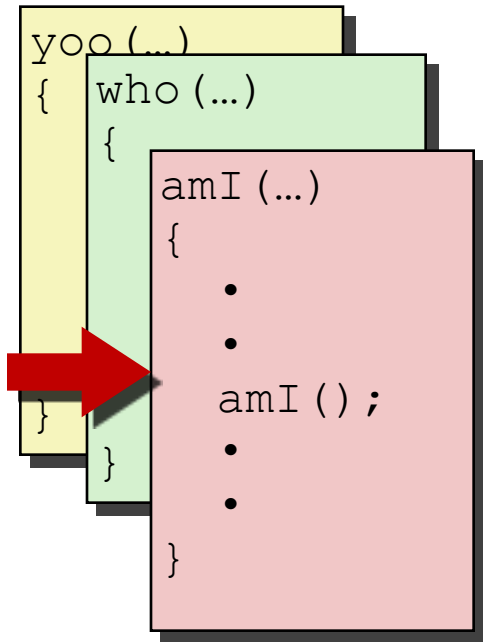
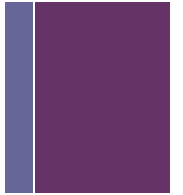
# + Example



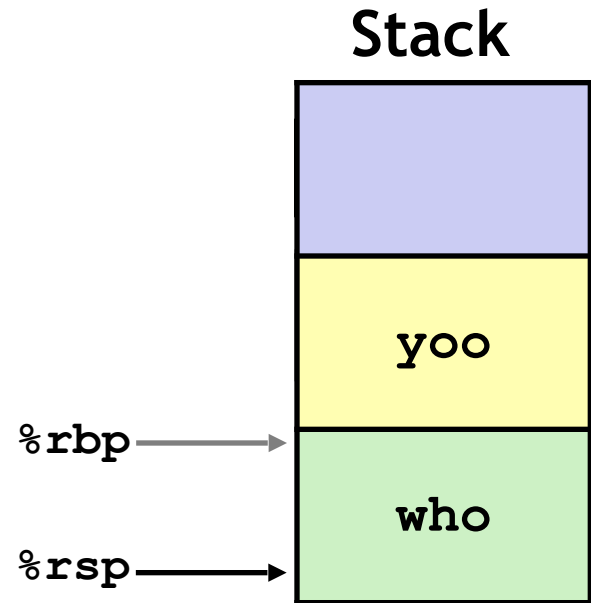
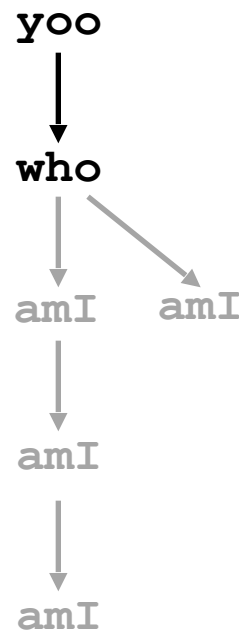
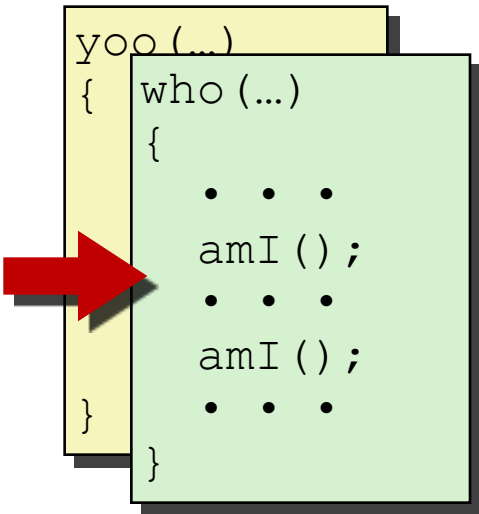
# + Example



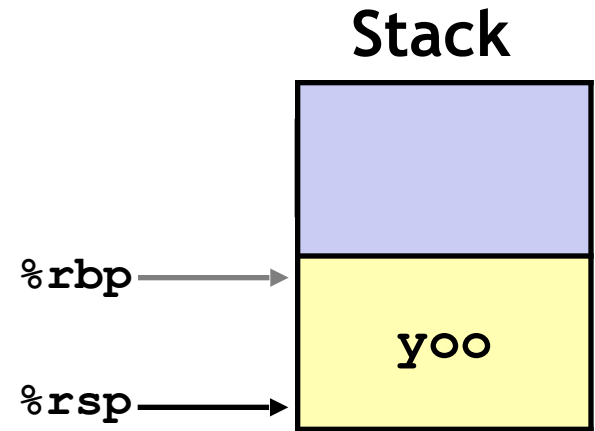
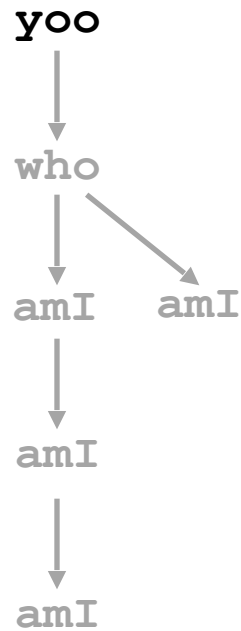
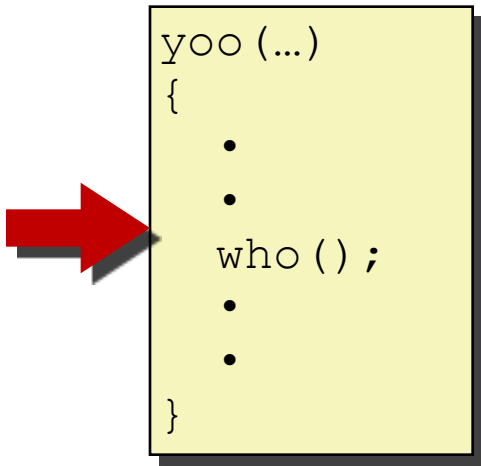
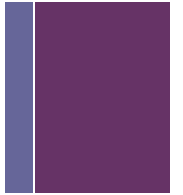
# + Example



# + Example



# + Example



# + x86-64/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
- *Old frame pointer* (optional)

- **Caller Stack Frame**

- Return address
  - Pushed by **call** instruction
- Arguments for this call

