

Procedures and the Call Stack

Topics

- Procedures
- Call stack
- Procedure/stack instructions
- Calling conventions
- Register-saving conventions

Implementing Procedures

How does a caller pass **arguments** to a procedure?

How does a caller receive a **return value** from a procedure?

Where does a procedure store **local variables**?

How does a procedure know **where to return**
(what code to execute next when done)?

How do procedures **share limited registers** and **memory**?

Call Chain

```
yoo (...)
```

```
{
```

```
.
```

```
.
```

```
who () ;
```

```
.
```

```
.
```

```
}
```

```
who (...)
```

```
{
```

```
• • •
```

```
ru () ;
```

```
• • •
```

```
ru () ;
```

```
• • •
```

```
}
```

```
ru (...)
```

```
{
```

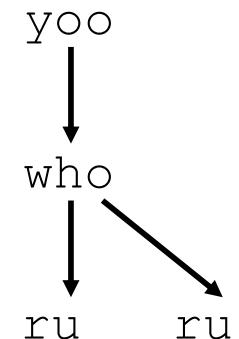
```
.
```

```
.
```

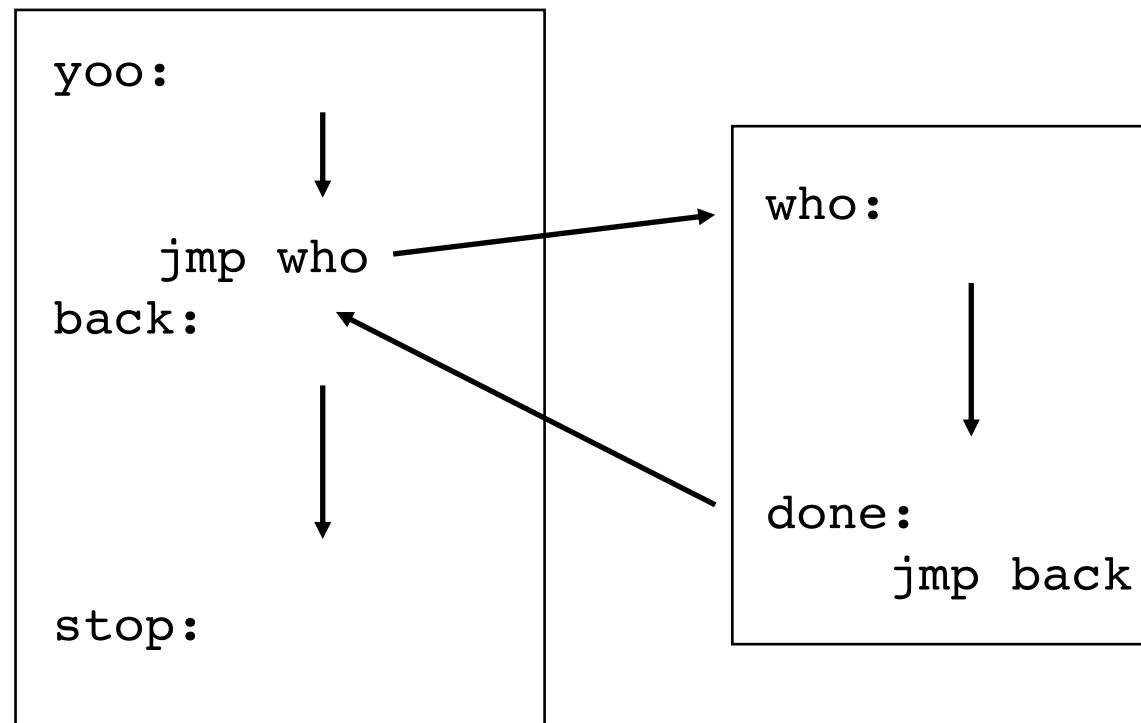
```
.
```

```
}
```

Example
Call Chain

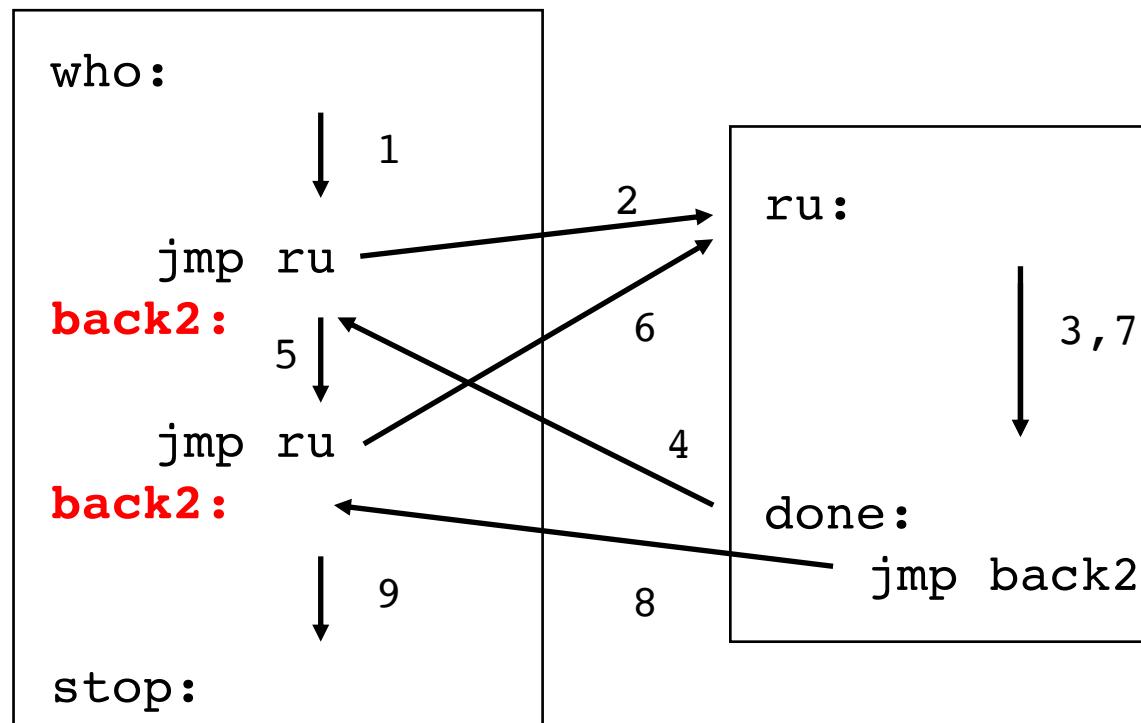


First Try...



But what if we want to call a function from multiple places in the code?

First Try... broken!



Implementing Procedures

How does a caller pass **arguments** to a procedure?

How does a caller receive a **return value** from a procedure?

Where does a procedure store **local variables**?

How does a procedure know **where to return**
(what code to execute next when done)?

How do procedures **share limited registers** and **memory**?

All these need **separate storage *per call!***
(not just per procedure)

Procedure Control Flow Instructions

Procedure call: `callq label`

1. Push return address on stack
2. Jump to *label*

Return address: Address of instruction after `call`. Example:

```
400544: callq 400550 <mult2>
          ↓
400549: movq    %rax, (%rbx)
```

Procedure return: `retq`

1. Pop return address from stack
2. Jump to return address

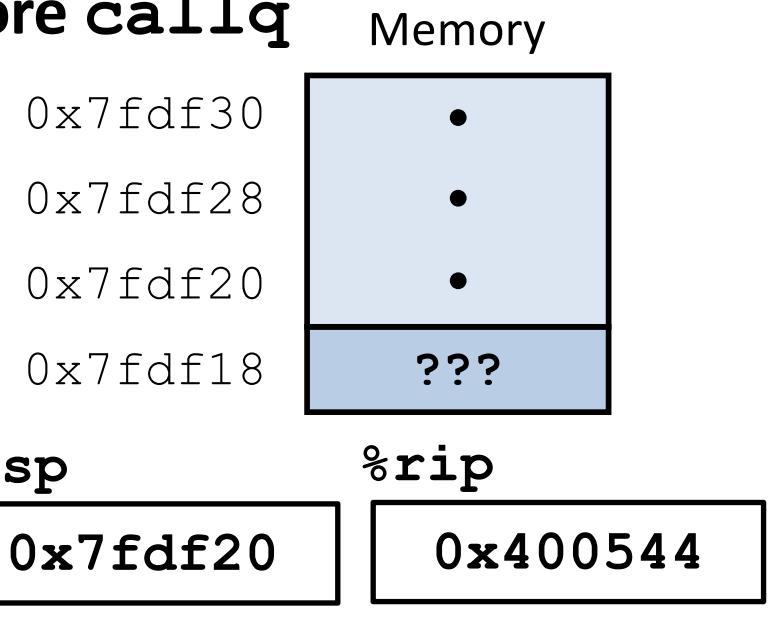
Call Example

```
0000000000400540 <multstore>:  
.  
. .  
400544: callq 400550 <mult2>  
400549: mov    %rax, (%rbx)  
. .
```

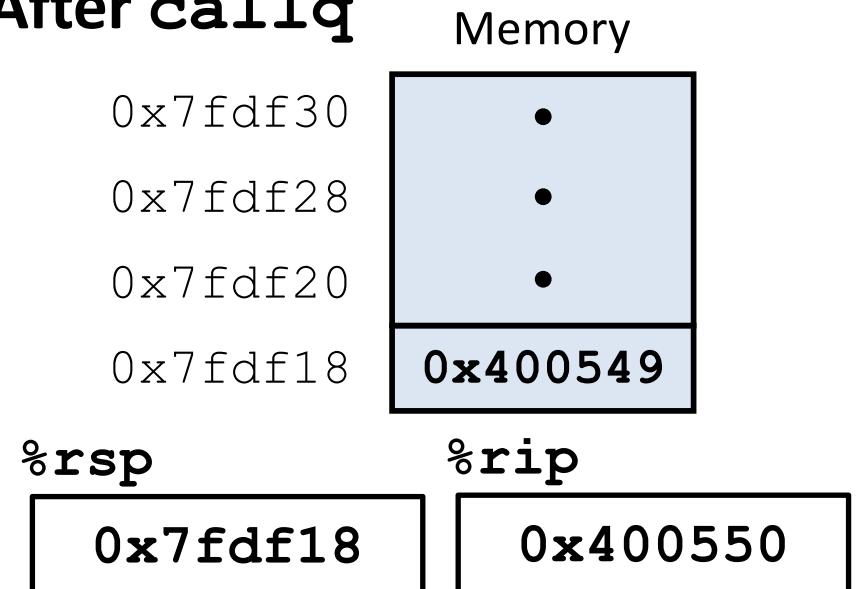
```
0000000000400550 <mult2>:  
400550: mov    %rdi, %rax  
. .  
400557: retq
```



Before `callq`



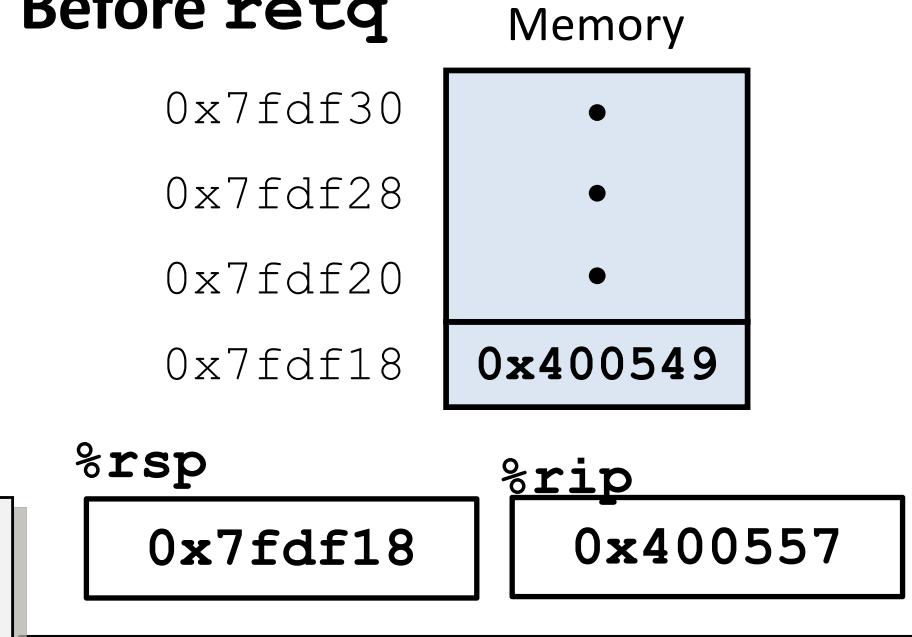
After `callq`



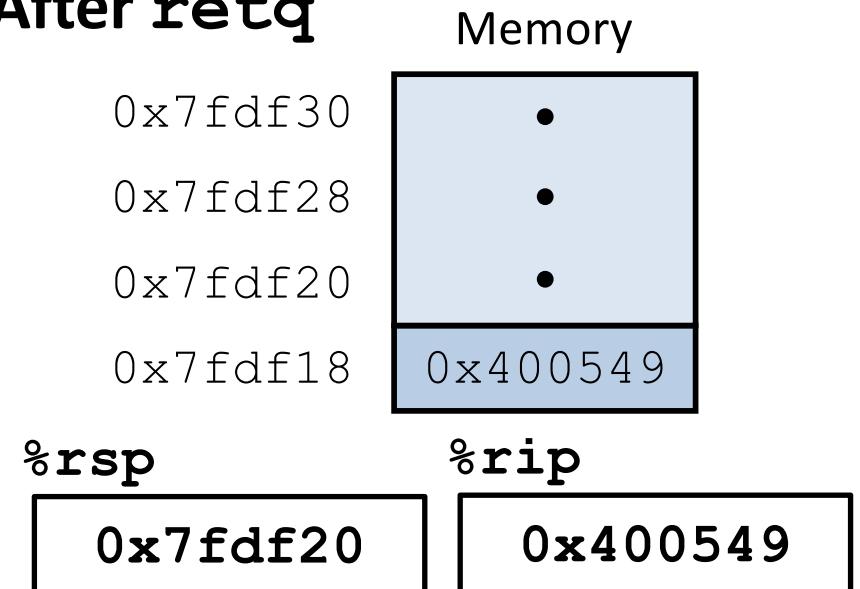
Return Example

```
0000000000400540 <multstore>:  
.  
. .  
400544: callq 400550 <mult2>  
400549: mov %rax, (%rbx)  
. .  
0000000000400550 <mult2>:  
400550: mov %rdi, %rax  
. .  
400557: retq
```

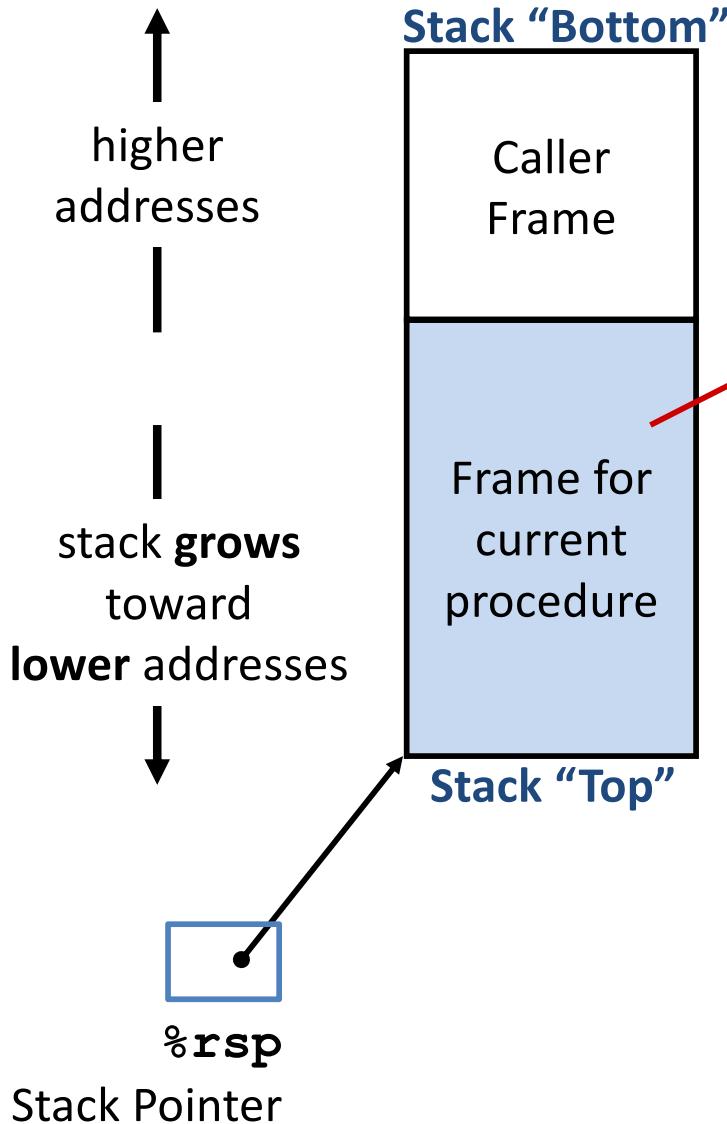
Before **retq**



After **retq**



Stack frames support procedure calls.



Contents

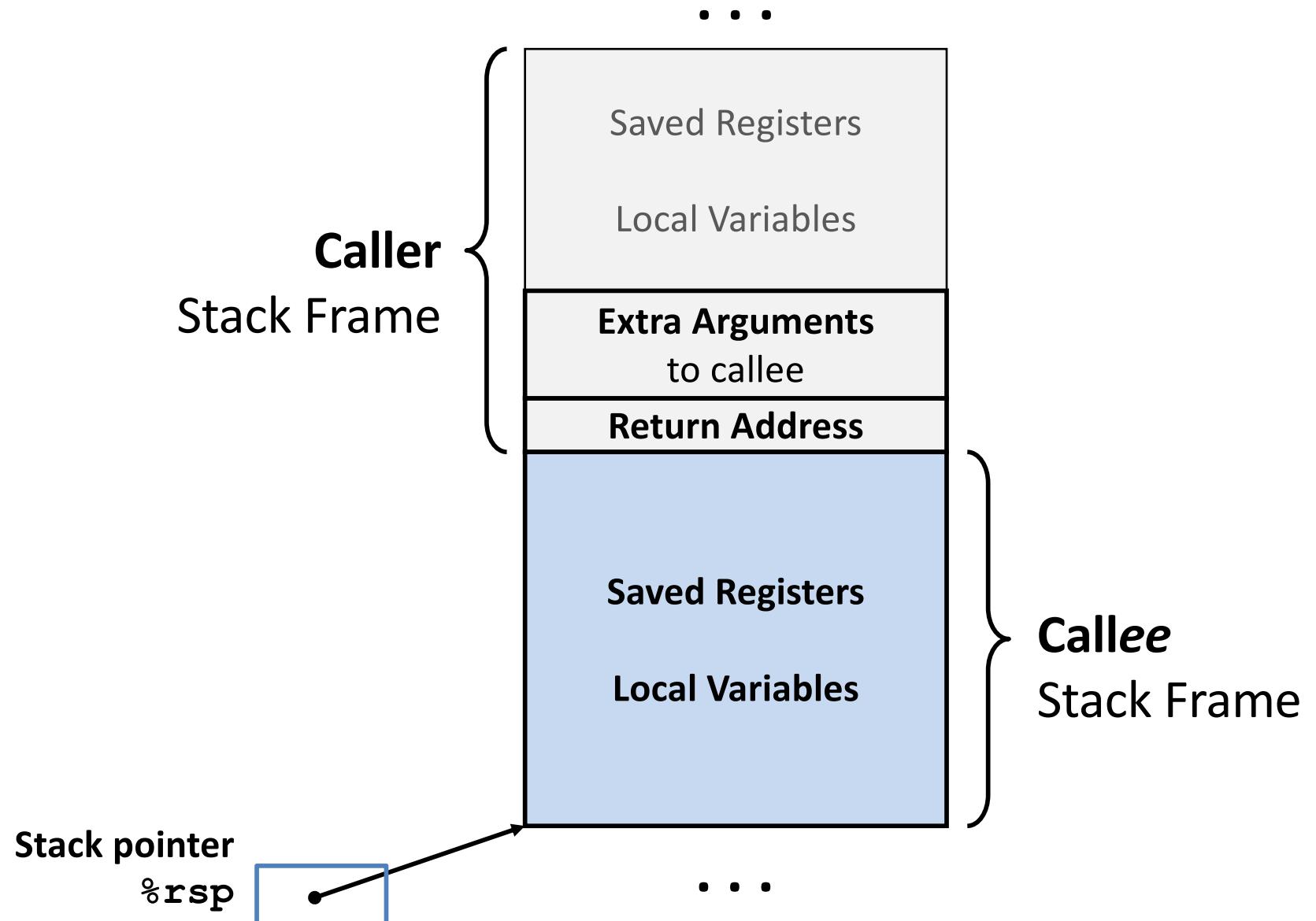
- Local variables
- Function arguments (after first 6)
- Return address
- Temporary space

Management

- Space allocated when procedure is entered
 - “Setup” code
- Space deallocated before return
 - “Finish” code

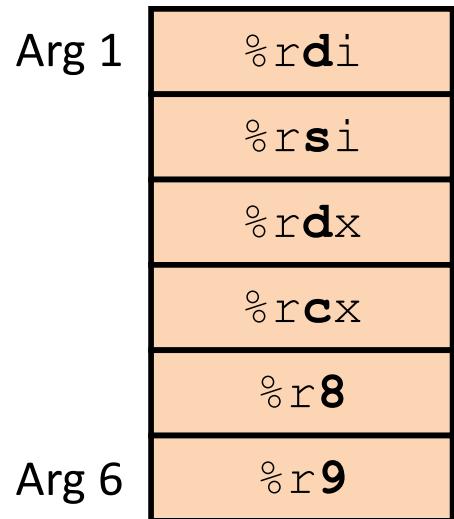
Why not just give every *procedure* a permanent chunk of memory to hold its local variables, etc?

Stack Frames

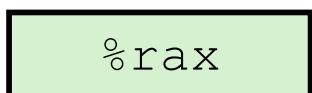


Procedure Data Flow Conventions

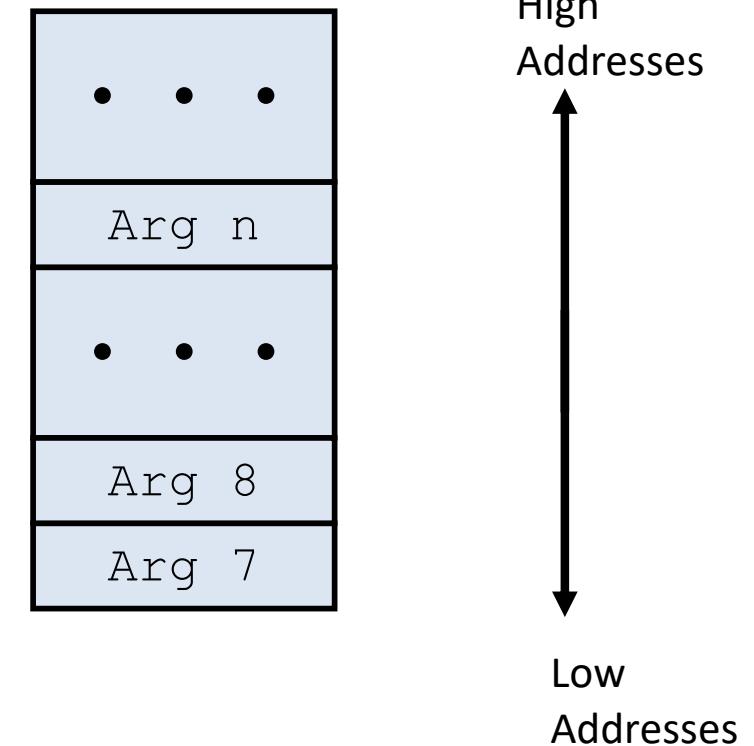
First 6 arguments passed
in registers



Return value



Remaining arguments passed
on stack (in memory)



Allocate stack space only when needed.

A Puzzle

C function body:

```
*p = d;  
return x - c;
```

assembly:

```
movsbl %dl,%edx  
movl %edx,(%rsi)  
movswl %di,%edi  
subl %edi,%ecx  
movl %ecx,%eax
```

Write the C function header, types, and order of parameters.

movsbl = move sign-extending a byte to a long (4-byte)

movswl = move sign-extending a word (2-byte) to a long (4-byte)

Procedure Call / Stack Frame Example

call_incr:

```
400509: subq    $8,  %rsp  
40050d: movq    $240, (%rsp)  
400515: movq    %rsp, %rdi  
400518: movl    $61, %esi  
40051d: callq   4004cd <increment>  
400522: addq    (%rsp), %rax  
400526: addq    $8,  %rsp  
40052a: retq
```

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

Passes address of local variable (in stack).

Uses memory through pointer.

increment:

```
4004cd: movq    (%rdi), %rax  
4004d0: addq    %rax, %rsi  
4004d3: movq    %rsi, (%rdi)  
4004d6: retq
```

```
long increment(long* p, long val) {  
    long x = *p;  
    long y = x + val;  
    *p = y;  
    return x;  
}
```

Procedure Call Example (step 0)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

call_incr:

```
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

increment:

```
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

main called call_incr

Stack
Frames



Memory

0x7fdf28

0x40053b

<main+8>

0x7fdf20

0x7fdf18

%rax



%rdi



%rsi



%rsp



%rip



Procedure Call Example (step 1)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

```
increment:  
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Allocate space
for local vars

Stack
Frames

main
call_incr



0x7fdf28

0x7fdf20

0x7fdf18

Memory

...

0x40053b
<main+8>

240
v1

%rax



%rdi



%rsi



%rsp

0x7fdf20

%rip

0x400515

Procedure Call Example (step 2)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

```
increment:  
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Set up args for call
to increment

Stack
Frames

main
call_incr

0x7fdf28

0x7fdf20

0x7fdf18

Memory

...

0x40053b

<main+8>

240

v1

%rax

%rdi

0x7fdf20

%rsi

61

%rsp

0x7fdf20

%rip

0x40051d

Procedure Call Example (step 3)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

increment:

```
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Call increment

Stack
Frames

main
call_incr

0x7fdf28
0x7fdf20
0x7fdf18

Memory

...
0x40053b <main+8>
240 v1
0x400522 <call_incr+25>

%rax %rdi %rsi

0x7fdf20
61

%rsp %rip

0x7fdf18
0x4004cd

Procedure Call Example (step 4)

Run increment

```
long call incr() {  
    long increment(long* p, long val) {  
        long x = *p;  
        long y = x + val;  
        *p = y;  
        return x;  
    }  
}
```

```
400500: subq %o, %1sp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

increment:

```
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Stack
Frames

call_incr main

0x7fdf28

0x7fdf20

0x7fdf18

Memory

...

0x40053b

<main+8>

301

v1

0x400522

<call_incr+25>

%rax

%rdi

%rsi

240

0x7fdf20

301

%rsp

%rip

0x7fdf18

0x4004d6

Procedure Call Example (step 5)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

```
increment:  
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Stack
Frames

main
call_incr

Return from increment
to call_incr

Memory

0x7fdf28	0x40053b <main+8>
0x7fdf20	301 v1
0x7fdf18	0x400522 <call_incr+25>

%rax	%rdi	%rsi
240	0x7fdf20	301
%rsp	%rip	
0x7fdf20	0x400522	

Procedure Call Example (step 6)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

```
increment:  
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Prepare call_incr result

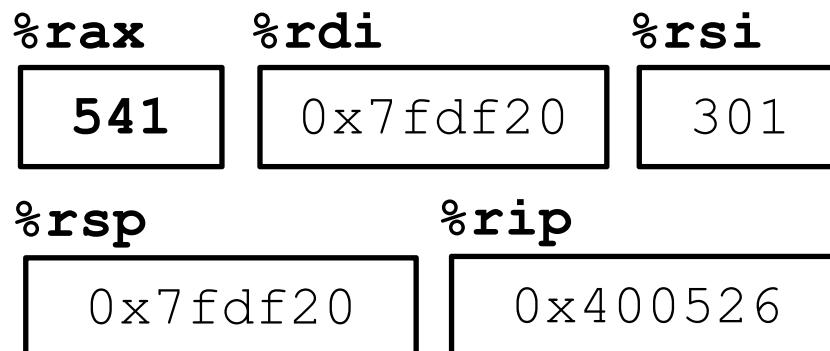
Stack
Frames

main
call_incr

0x7fdf28
0x7fdf20
0x7fdf18

Memory

...
0x40053b <main+8>
301 v1
0x400522 <call_incr+25>



Procedure Call Example (step 7)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

```
increment:  
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Stack
Frames

main

Deallocate space
for local vars

Memory

0x7fdf28	0x40053b <main+8>
0x7fdf20	301 v1
0x7fdf18	0x400522 <call_incr+25>

%rax

541

%rdi

0x7fdf20

%rsi

301

%rsp

0x7fdf28

%rip

0x400526

Procedure Call Example (step 8)

```
long call_incr() {  
    long v1 = 240;  
    long v2 = increment(&v1, 61);  
    return v1+v2;  
}
```

```
call_incr:  
400509: subq $8, %rsp  
40050d: movq $240, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq (%rsp), %rax  
400526: addq $8, %rsp  
40052a: retq
```

```
increment:  
4004cd: movq (%rdi), %rax  
4004d0: addq %rax, %rsi  
4004d3: movq %rsi, (%rdi)  
4004d6: retq
```

Stack
Frames

main [

]

Memory

0x7fdf28	0x40053b <main+8>
0x7fdf20	301 v1
0x7fdf18	0x400522 <call_incr+25>

%rax

541

%rdi

0x7fdf20

%rsi

301

%rsp

0x7fdf30

%rip

0x40053b

Register Saving Conventions

yoo calls who:

Caller *Callee*

Will register contents still be there after a procedure call?

```
yoo:  
  . . .  
  movq $12345, %rbx  
  call who  
  addq %rbx, %rax  
  . . .  
  ret
```

```
who:  
  . . .  
  addq %rdi, %rbx  
  . . .  
  ret
```

Conventions:

Caller Save

Callee Save

x86-64 64-bit Register Conventions

%rax	Return value – Caller saved	%r8	Argument #5 – Caller saved
%rbx	Callee saved		
%rcx	Argument #4 – Caller saved	%r10	Caller saved
%rdx	Argument #3 – Caller saved	%r11	Caller Saved
%rsi	Argument #2 – Caller saved	%r12	Callee saved
%rdi	Argument #1 – Caller saved	%r13	Callee saved
%rsp	Stack pointer	%r14	Callee saved
%rbp	Callee saved	%r15	Callee saved

Callee-Save Example (step 0)

```
long call_incr2(long x) {  
    long v1 = x;  
    long v2 = increment(&v1, 61);  
    return x + v2;  
}
```

call_incr2:

```
400504: pushq %rbx  
400506: movq %rdi, %rbx  
400509: subq $16, %rsp  
40050d: movq %rdi, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq %rbx, %rax  
400525: addq $16, %rsp  
400529: popq %rbx  
40052b: retq
```

main called call_incr2(240)

Stack
Frames
main

0x7fdf28
0x7fdf20
0x7fdf18
0x7fdf10
0x7fdf08

Memory

...

0x40053b

%rbx

3

%rax

%rdi

240

%rsi

%rsp

0x7fdf28

%rip

0x400504

Callee-Save Example (step 1)

```
long call_incr2(long x) {  
    long v1 = x;  
    long v2 = increment(&v1, 61);  
    return x + v2;  
}
```

```
call_incr2:  
400504: pushq %rbx  
400506: movq %rdi, %rbx  
400509: subq $16, %rsp  
40050d: movq %rdi, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq %rbx, %rax  
400525: addq $16, %rsp  
400529: popq %rbx  
40052b: retq
```

Register save

Stack
Frames

call_incr2 main

0x7fdf28	0x40053b
0x7fdf20	3
0x7fdf18	
0x7fdf10	
0x7fdf08	

Memory

...

3

%rbx

3

%rax

%rdi

240

%rsi

%rsp

0x7fdf20

%rip

0x400506

Callee-Save Example (step 2)

```
long call_incr2(long x) {  
    long v1 = x;  
    long v2 = increment(&v1, 61);  
    return x + v2;  
}
```

```
call_incr2:  
400504: pushq %rbx  
400506: movq %rdi, %rbx  
400509: subq $16, %rsp  
40050d: movq %rdi, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq %rbx, %rax  
400525: addq $16, %rsp  
400529: popq %rbx  
40052b: retq
```

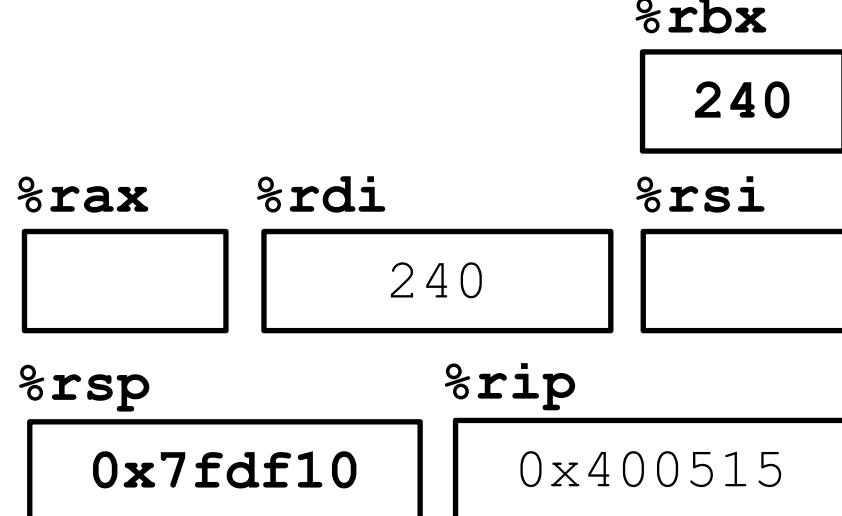
Stack frame setup
(extra slot for alignment)

Stack
Frames

call_incr2 main

0x7fdf28	0x40053b
0x7fdf20	3
0x7fdf18	for alignment
0x7fdf10	240
0x7fdf08	

Memory



Callee-Save Example (step 3)

```
long call_incr2(long x) {  
    long v1 = x;  
    long v2 = increment(&v1, 61);  
    return x + v2;  
}
```

```
call_incr2:  
400504: pushq %rbx  
400506: movq %rdi, %rbx  
400509: subq $16, %rsp  
40050d: movq %rdi, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq %rbx, %rax  
400525: addq $16, %rsp  
400529: popq %rbx  
40052b: retq
```

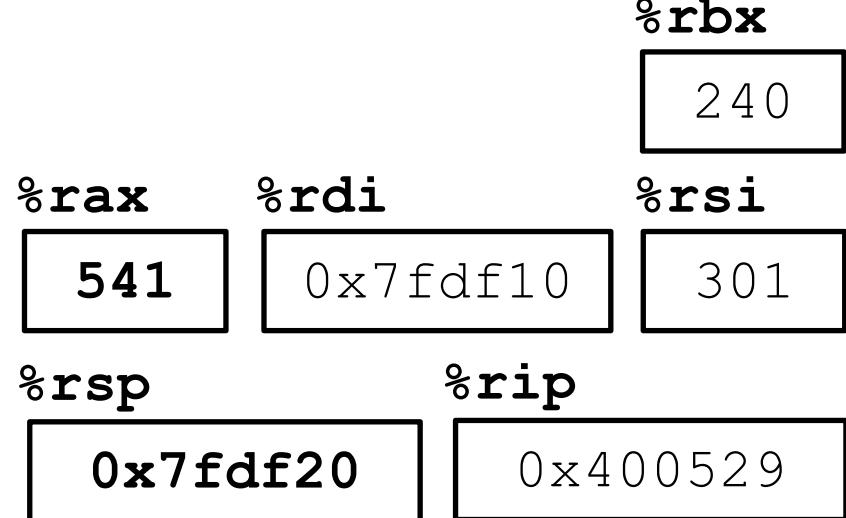
Prepare return value
and tear down stack frame

Stack
Frames

call_incr2 main

0x7fdf28	0x40053b
0x7fdf20	3
0x7fdf18	<i>for alignment</i>
0x7fdf10	301
0x7fdf08	0x400522

Memory



Callee-Save Example (step 4)

Register restore

```
long call_incr2(long x) {  
    long v1 = x;  
    long v2 = increment(&v1, 61);  
    return x + v2;  
}
```

```
call_incr2:  
400504: pushq %rbx  
400506: movq %rdi, %rbx  
400509: subq $16, %rsp  
40050d: movq %rdi, (%rsp)  
400515: movq %rsp, %rdi  
400518: movl $61, %esi  
40051d: callq 4004cd <increment>  
400522: addq %rbx, %rax  
400525: addq $16, %rsp  
400529: popq %rbx  
40052b: retq
```

Stack
Frames
main

Memory

0x7fdf28	0x40053b
0x7fdf20	3
0x7fdf18	<i>for alignment</i>
0x7fdf10	301
0x7fdf08	0x400522

%rbx

3

%rax

541

%rdi

0x7fdf10

%rsi

301

%rsp

0x7fdf28

%rip

0x40052b

Recursion Example: code

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

pcount:

The diagram illustrates the assembly code for the `pcount` function. It highlights the base case/condition, the recursive case, and the save/restore logic for the `%rbx` register.

base case/condition: The base case is checked at address 4005e2 with the instruction `testq %rdi, %rdi`. If `x == 0`, control jumps to the end of the function at address 4005fb.

recursive case: The recursive case is handled by the loop starting at address 4005f1. It involves the following steps:

- Save `%rbx`: `pushq %rbx` (at 4005e7).
- Set `%rbx` to `x & 1`: `movq %rdi, %rbx` (at 4005e8).
- Check for 1: `andl $1, %ebx` (at 4005eb).
- Shift right: `shrq %rdi` (at 4005ee).
- Call `pcount`: `callq pcount` (at 4005f1). This step is annotated with "x&1 in %rbx across call".
- Add results: `addq %rbx, %rax` (at 4005f6).
- Restore `%rbx`: `popq %rbx` (at 4005f9).

save/restore: The `%rbx` register is marked as callee-saved. It is saved at 4005e7 and restored at 4005f9.

Labels:

- `.L6:` The label for the base case jump point.
- `4005fa:` The label for the `rep` instruction.
- `4005fb:` The label for the `retq` instruction.

Recursion Example: pcount(2)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

pcount:

```
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb: retq
```

Stack

Frames

main [

0x7fdf38
0x7fdf30
0x7fdf28
0x7fdf20
0x7fdf18
0x7fdf10
0x7fdf08

Memory

0x4006ed

%rax

0

%rdi

2

%rbx

42

%rsp

0x7fdf38

%rip

0x4005dd

Recursion Example: pcount(2)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

```
pcount:  
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb: retq
```

Stack

Frames

pc(2) main

0x7fdf38
0x7fdf30
0x7fdf28
0x7fdf20
0x7fdf18
0x7fdf10
0x7fdf08

Memory

0x4006ed

%rax

0

%rdi

2

%rbx

42

%rsp

0x7fdf38

%rip

0x4005e7

Recursion Example: pcount(2)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

```
pcount:  
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb: retq
```

Stack

Frames

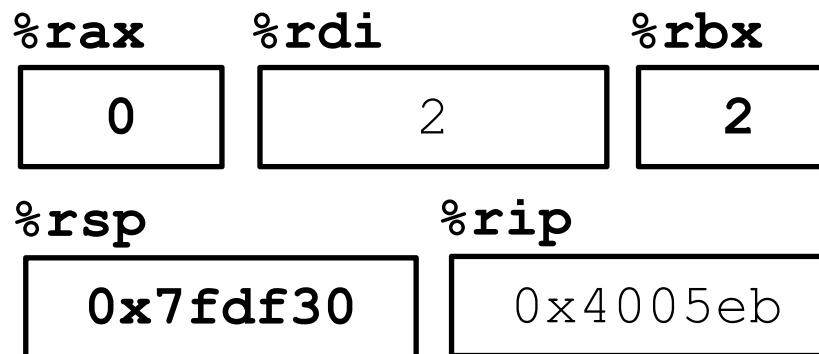
pc(2) main

[0x7fdf38
 0x7fdf30
 0x7fdf28
 0x7fdf20
 0x7fdf18
 0x7fdf10
 0x7fdf08]

Memory

0x4006ed

42



Recursion Example: pcount(2)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

```
pcount:  
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb:  retq
```

Stack

Frames

pc(2) main

[0x7fdf38
 0x7fdf30
 0x7fdf28
 0x7fdf20
 0x7fdf18
 0x7fdf10
 0x7fdf08]

Memory

[0x4006ed]

[42]

%rax

[0]

%rdi

[1]

%rbx

[0]

%rsp

[0x7fdf30]

%rip

[0x4005f1]

Recursion Example: pcount(2) → pcount(1)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

pcount:

```
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb: retq
```

Stack

Frames

pc(2) main

0x7fdf38

0x7fdf30

0x7fdf28

0x7fdf20

0x7fdf18

0x7fdf10

0x7fdf08

Memory

0x4006ed

42

0x4005f6

%rax

0

%rdi

1

%rbx

0

%rsp

0x7fdf28

%rip

0x4005dd

Recursion Example: pcount(2) → pcount(1)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```

Stack

Frames

pc(2) main

0x7fdf38

0x7fdf30

0x7fdf28

0x7fdf20

0x7fdf18

0x7fdf10

0x7fdf08

Memory

0x4006ed

42

0x4005f6

%rax

%rdi

%rbx

0

1

0

%rsp

%rip

0x7fdf28

0x4005e7

Recursion Example: pcount(2) → pcount(1)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```
4005dd:    movl $0, %eax
4005e2:    testq %rdi, %rdi
4005e5:    je 4005fa <.L6>
4005e7:    pushq %rbx
4005e8:    movq %rdi, %rbx
4005eb:    andl $1, %ebx
4005ee:    shrq %rdi
4005f1:    callq pcount
4005f6:    addq %rbx, %rax
4005f9:    popq %rbx
.L6:
4005fa:    rep
4005fb:    retq
```

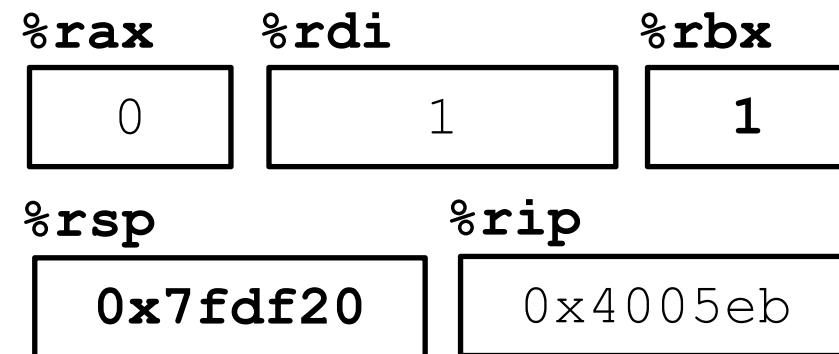
Stack

Frames

pc(1) **pc(2)** **main**

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	
0x7fdf10	
0x7fdf08	



Recursion Example: pcount(2) → pcount(1)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```
4005dd:    movl $0, %eax
4005e2:    testq %rdi, %rdi
4005e5:    je 4005fa <.L6>
4005e7:    pushq %rbx
4005e8:    movq %rdi, %rbx
4005eb:    andl $1, %ebx
4005ee:    shrq %rdi
4005f1:    callq pcount
4005f6:    addq %rbx, %rax
4005f9:    popq %rbx
.L6:
4005fa:    rep
4005fb:    retq
```

Stack

Frames

pc(1) pc(2) main

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	
0x7fdf10	
0x7fdf08	

%rax

0

%rdi

0

%rbx

1

%rsp

0x7fdf20

%rip

0x4005f1

Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```

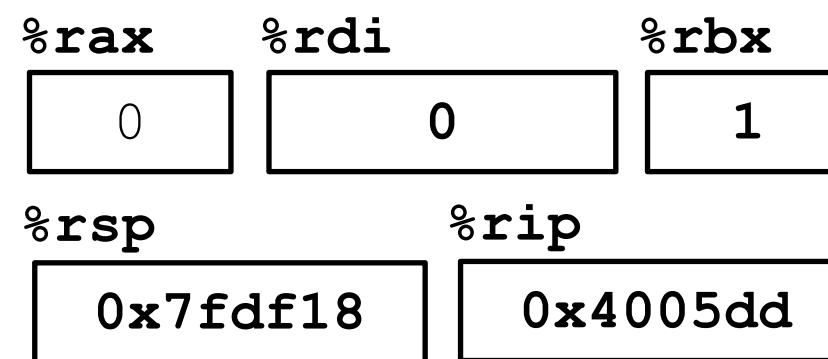
Stack

Frames

pc(1) pc(2) main

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	



Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```
4005dd: movl $0, %eax
4005e2: testq %rdi, %rdi
4005e5: je 4005fa <.L6>
4005e7: pushq %rbx
4005e8: movq %rdi, %rbx
4005eb: andl $1, %ebx
4005ee: shrq %rdi
4005f1: callq pcount
4005f6: addq %rbx, %rax
4005f9: popq %rbx
.L6:
4005fa: rep
4005fb: retq
```

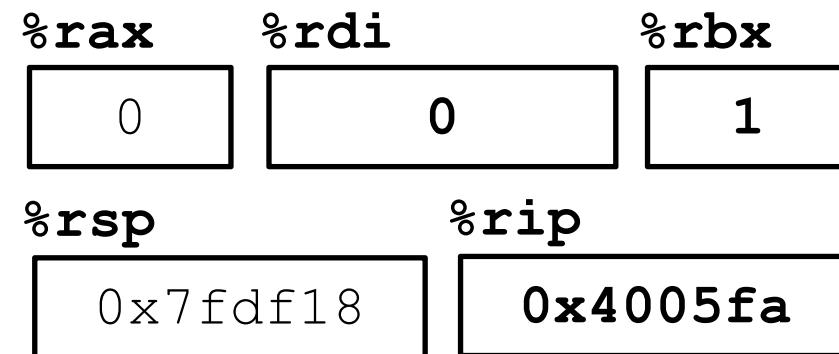
Stack

Frames

pc(1) pc(2) main

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	



Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```
4005dd:  movl $0, %eax
4005e2:  testq %rdi, %rdi
4005e5:  je 4005fa <.L6>
4005e7:  pushq %rbx
4005e8:  movq %rdi, %rbx
4005eb:  andl $1, %ebx
4005ee:  shrq %rdi
4005f1:  callq pcount
4005f6:  addq %rbx, %rax
4005f9:  popq %rbx
.L6:
4005fa:  rep
4005fb:  retq
```

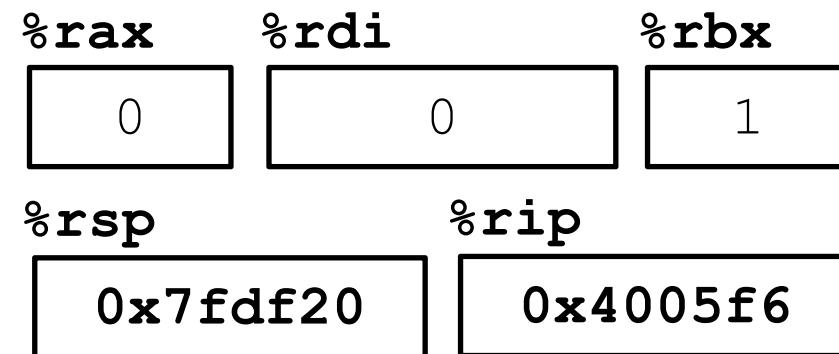
Stack

Frames

pc(1) pc(2) main

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	



Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

```
pcount:
4005dd:    movl $0, %eax
4005e2:    testq %rdi, %rdi
4005e5:    je 4005fa <.L6>
4005e7:    pushq %rbx
4005e8:    movq %rdi, %rbx
4005eb:    andl $1, %ebx
4005ee:    shrq %rdi
4005f1:    callq pcount
4005f6:    addq %rbx, %rax
4005f9:    popq %rbx
.L6:
4005fa:    rep
4005fb:    retq
```

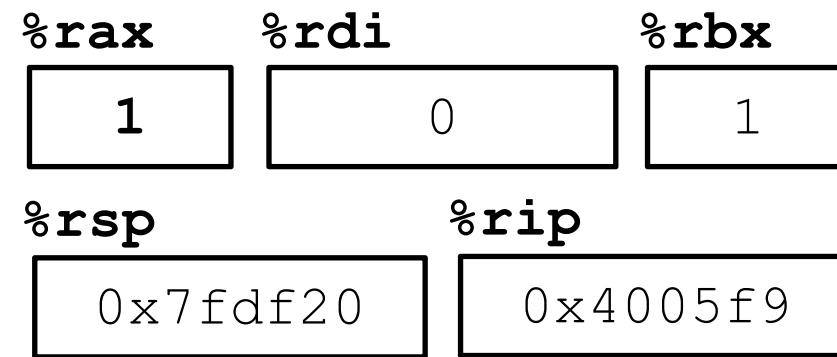
Stack

Frames

pc(1) pc(2) main

Memory

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	



Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

pcount:

```

4005dd:  movl $0, %eax
4005e2:  testq %rdi, %rdi
4005e5:  je 4005fa <.L6>
4005e7:  pushq %rbx
4005e8:  movq %rdi, %rbx
4005eb:  andl $1, %ebx
4005ee:  shrq %rdi
4005f1:  callq pcount
4005f6:  addq %rbx, %rax
4005f9:  popq %rbx
.L6:
4005fa:  rep
4005fb:  retq

```

Stack

Frames

pc(2) main

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	

Memory



Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

```
pcount:  
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb: retq
```

Stack

Frames

pc(2) main

0x7fdf38

Memory

0x4006ed

0x7fdf30

42

0x7fdf28

0x4005f6

0x7fdf20

0

0x7fdf18

0x4005f6

0x7fdf10

0x7fdf08

%rax

1

%rdi

0

%rbx

0

%rsp

0x7fdf30

%rip

0x4005f6

Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {
    if (x == 0) {
        return 0;
    } else {
        return (x & 1) + pcount(x >> 1);
    }
}
```

```
pcount:
4005dd:    movl $0, %eax
4005e2:    testq %rdi, %rdi
4005e5:    je 4005fa <.L6>
4005e7:    pushq %rbx
4005e8:    movq %rdi, %rbx
4005eb:    andl $1, %ebx
4005ee:    shrq %rdi
4005f1:    callq pcount
4005f6:    addq %rbx, %rax
4005f9:    popq %rbx
.L6:
4005fa:    rep
4005fb:    retq
```

Stack

Frames

pc(2) main

0x7fdf38
0x7fdf30
0x7fdf28
0x7fdf20
0x7fdf18
0x7fdf10
0x7fdf08

Memory

0x4006ed
42
0x4005f6
0
0x4005f6

%rax

1

%rdi

0

%rbx

0

%rsp

0x7fdf30

%rip

0x4005f9

Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

```
pcount:  
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb:  retq
```

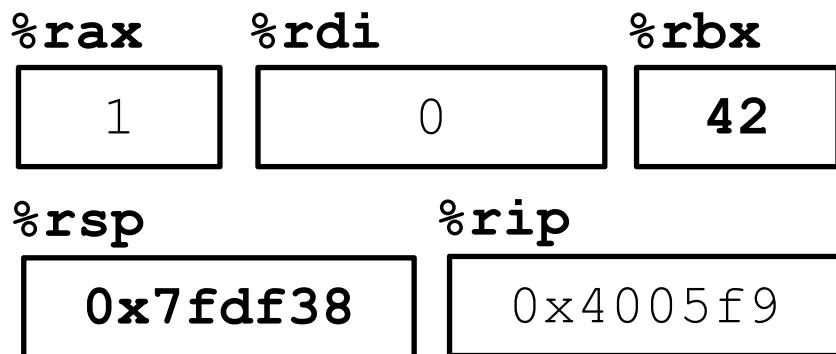
Stack

Frames

main

0x7fdf38	0x4006ed
0x7fdf30	42
0x7fdf28	0x4005f6
0x7fdf20	0
0x7fdf18	0x4005f6
0x7fdf10	
0x7fdf08	

Memory



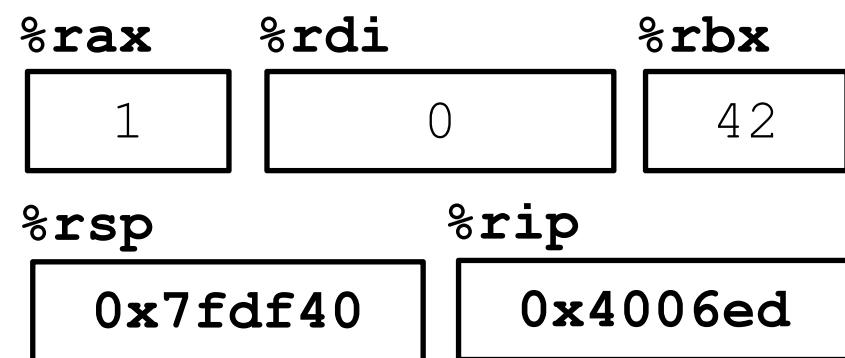
Recursion Example: pcount(2) → pcount(1) → pcount(0)

```
long pcount(unsigned long x) {  
    if (x == 0) {  
        return 0;  
    } else {  
        return (x & 1) + pcount(x >> 1);  
    }  
}
```

```
pcount:  
4005dd:  movl $0, %eax  
4005e2:  testq %rdi, %rdi  
4005e5:  je 4005fa <.L6>  
4005e7:  pushq %rbx  
4005e8:  movq %rdi, %rbx  
4005eb:  andl $1, %ebx  
4005ee:  shrq %rdi  
4005f1:  callq pcount  
4005f6:  addq %rbx, %rax  
4005f9:  popq %rbx  
.L6:  
4005fa:  rep  
4005fb: retq
```

Stack
Frames
main ↗

Memory
0x4006ed
42
0x4005f6
0
0x4005f6
0x7fdf10
0x7fdf08



x86-64 stack storage example

(1)

```
long int call_proc()
{
    long x1 = 1;
    int x2 = 2;
    short x3 = 3;
    char x4 = 4;
    proc(x1, &x1, x2, &x2,
          x3, &x3, x4, &x4);
    return (x1+x2) * (x3-x4);
}
```

```
call_proc:
    subq $32,%rsp
    movq $1,16(%rsp) # x1
    movl $2,24(%rsp) # x2
    movw $3,28(%rsp) # x3
    movb $4,31(%rsp) # x4
    • • •
```

Return address to caller of call_proc

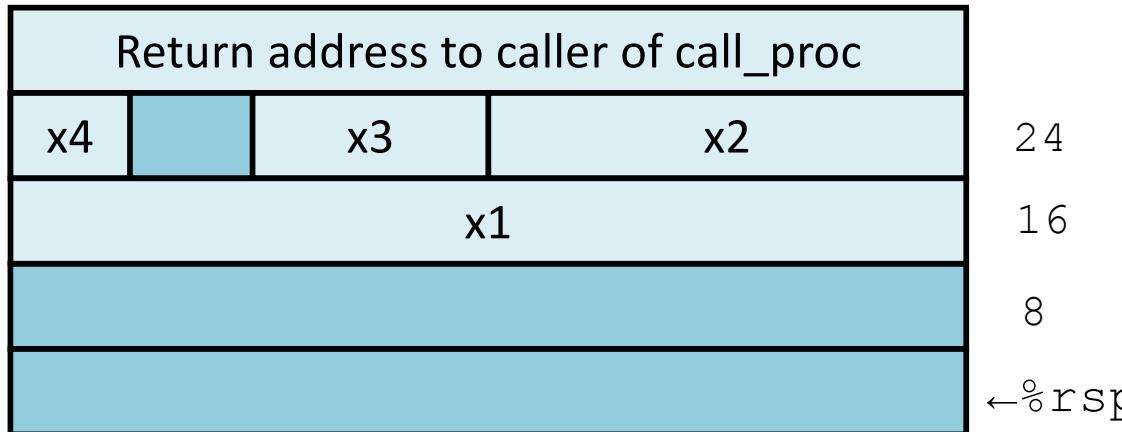
←%rsp

x86-64 stack storage example

(2) Allocate local vars

```
long int call_proc()
{
    long x1 = 1;
    int  x2 = 2;
    short x3 = 3;
    char x4 = 4;
    proc(x1, &x1, x2, &x2,
          x3, &x3, x4, &x4);
    return (x1+x2)*(x3-x4);
}
```

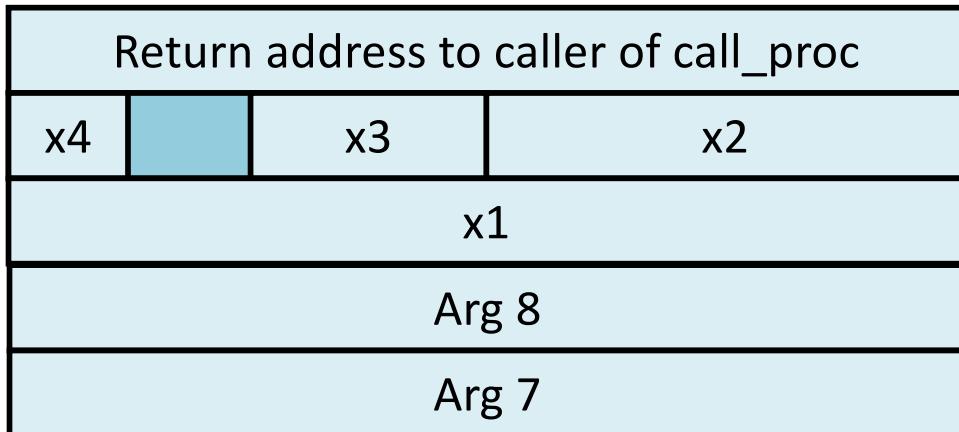
```
call_proc:
    subq $32,%rsp
    movq $1,16(%rsp) # x1
    movl $2,24(%rsp) # x2
    movw $3,28(%rsp) # x3
    movb $4,31(%rsp) # x4
    • • •
```



x86-64 stack storage example

(3) setup args to proc

```
long int call_proc()
{
    long x1 = 1;
    int x2 = 2;
    short x3 = 3;
    char x4 = 4;
    proc(x1, &x1, x2, &x2,
          x3, &x3, x4, &x4);
    return (x1+x2) * (x3-x4);
}
```



```
call_proc:
```

```
    • • •  

    leaq  24(%rsp),%rcx # &x2  

    leaq  16(%rsp),%rsi # &x1  

    leaq  31(%rsp),%rax # &x4  

    movq %rax,8(%rsp) # ...  

    movl $4,(%rsp)      # 4  

    leaq  28(%rsp),%r9 # &x3  

    movl $3,%r8d        # 3  

    movl $2,%edx         # 2  

    movq $1,%rdi        # 1  

    call proc  

    • • •
```

24

16

8

←%rsp

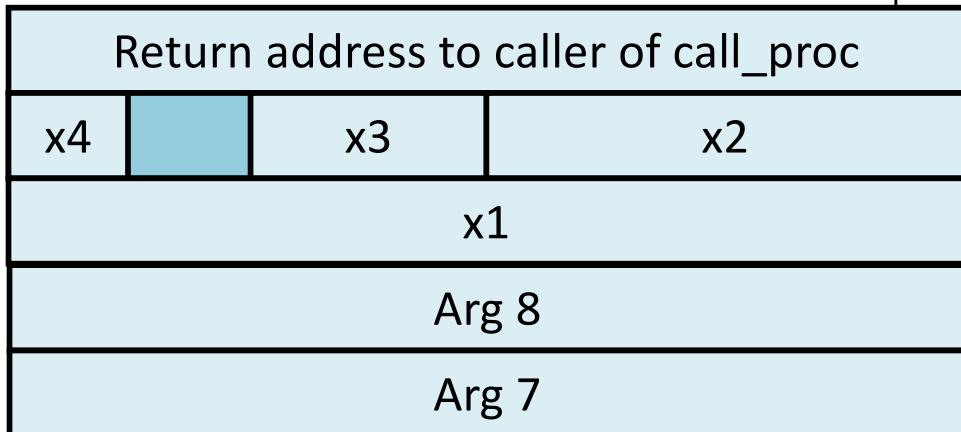
Arguments passed in (in order): rdi,
rsi, rdx, rcx, r8, r9

x86-64 stack storage example

(4) after call to proc

```
long int call_proc()
{
    long x1 = 1;
    int x2 = 2;
    short x3 = 3;
    char x4 = 4;
    proc(x1, &x1, x2, &x2,
          x3, &x3, x4, &x4);
    return (x1+x2) * (x3-x4);
}
```

```
call_proc:
    . . .
    movswl 28(%rsp),%eax # x3
    movsb1 31(%rsp),%edx # x4
    subl %edx,%eax # x3-x4
    cltq # sign-extend %eax->rax
    movslq 24(%rsp),%rdx # x2
    addq 16(%rsp),%rdx # x1+x2
    imulq %rdx,%rax # *
    addq $32,%rsp
    ret
```



24
16
8
←%rsp

x86-64 stack storage example

(5) deallocate local vars

```
long int call_proc()
{
    long x1 = 1;
    int x2 = 2;
    short x3 = 3;
    char x4 = 4;
    proc(x1, &x1, x2, &x2,
          x3, &x3, x4, &x4);
    return (x1+x2) * (x3-x4);
}
```

```
call_proc:
    . . .
    movswl 28(%rsp), %eax
    movsb1 31(%rsp), %edx
    subl    %edx, %eax
    cltq
    movslq 24(%rsp), %rdx
    addq    16(%rsp), %rdx
    imulq   %rdx, %rax
    addq   $32, %rsp
    ret
```

Return address to caller of call_proc

←%rsp

Procedure Summary

call, ret, push, pop

Stack discipline fits procedure call / return.

If P calls Q: Q (and calls by Q) returns before P

Conventions support arbitrary function calls.

Register-save conventions.

Stack frame saves extra args or local variables.

Result returned in **%rax**

%rax	Return value – Caller saved
%rbx	Callee saved
%rcx	Argument #4 – Caller saved
%rdx	Argument #3 – Caller saved
%rsi	Argument #2 – Caller saved
%rdi	Argument #1 – Caller saved
%rsp	Stack pointer
%rbp	Callee saved

%r8	Argument #5 – Caller saved
%r9	Argument #6 – Caller saved
%r10	Caller saved
%r11	Caller Saved
%r12	Callee saved
%r13	Callee saved
%r14	Callee saved
%r15	Callee saved

