

Welcome to 213 recitation!

15/18-213: Introduction to Computer Systems
11st Recitation, Sept. 26, 2011

TA: Pablo Chavez (pchavez)
Section E, 1:30pm – 2:20pm GHC 5222

Slide material from Kevin Su

Outline

- Reminders
- Stack discipline (x86)
- Stack discipline (x86_64)
- Data lab handback

Reminders

Stack x86

Stack x86_64

Data lab

Questions

Reminder – Bomb lab

- Due tomorrow, at midnight
- Feel free to ping the staff list with questions
 - 15-213-staff@cs.cmu.edu
- Office hours
 - Sun – Thurs 5:30pm – 8:30pm @ WeH 5207

Stacks



Reminders

Stack x86

Stack x86_64

Data lab

Questions

Stacks



Reminders

Stack x86

Stack x86_64

Data lab

Questions

Stacks (x86)

ebp->
esp->



Reminders

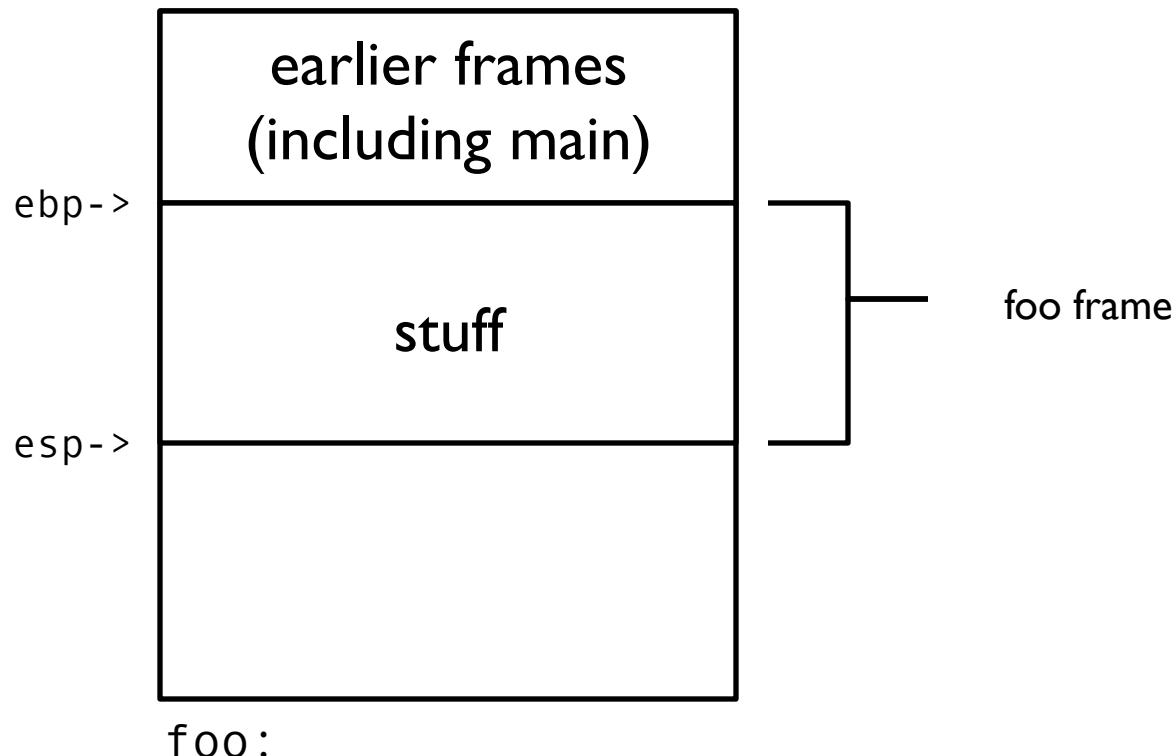
Stack x86

Stack x86_64

Data lab

Questions

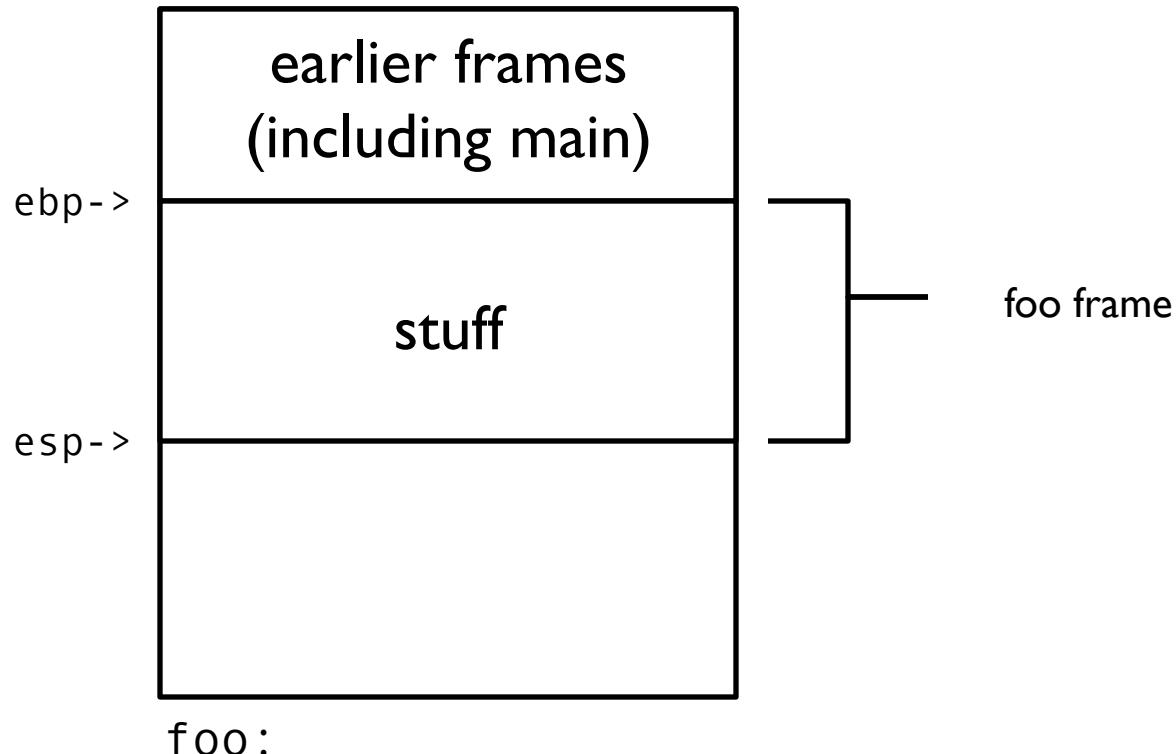
Stack Operations (x86) – Push



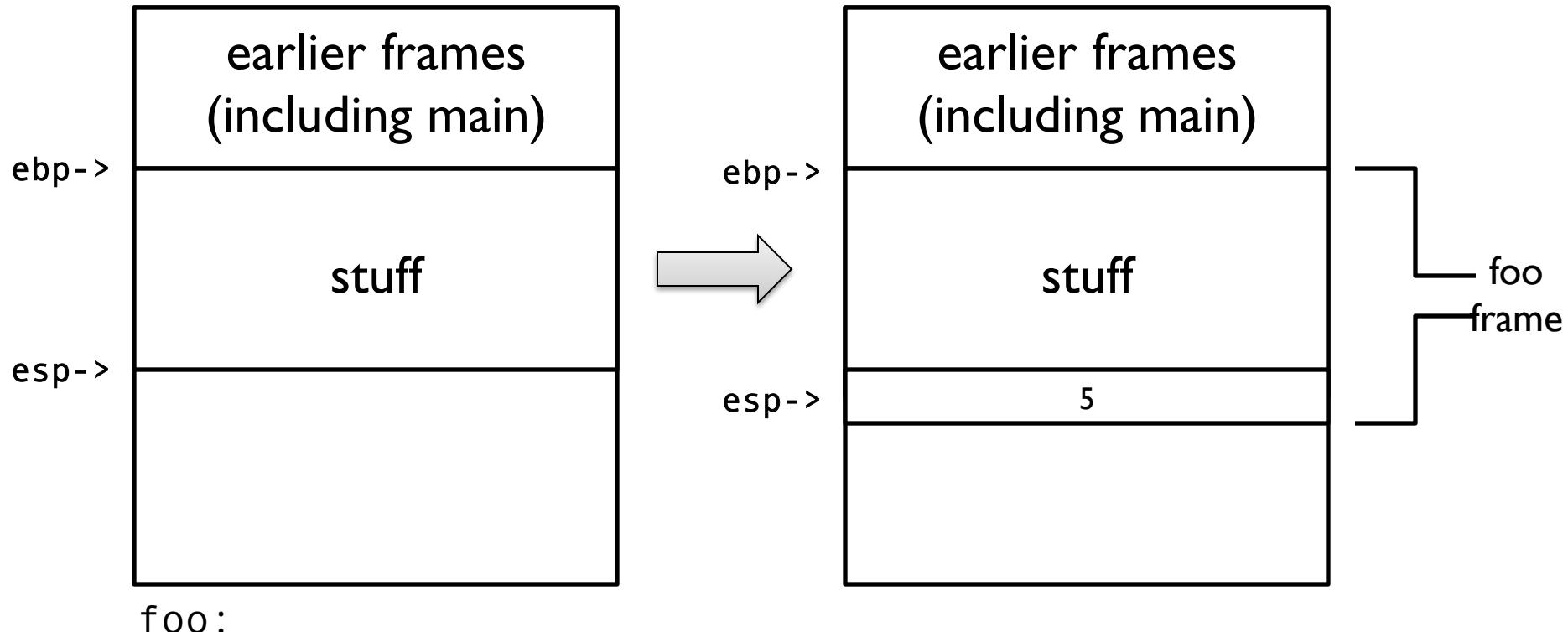
foo:

```
...
    movl $0x5, %edx
    push %edx
```

Stack Operations (x86) – Push (2)



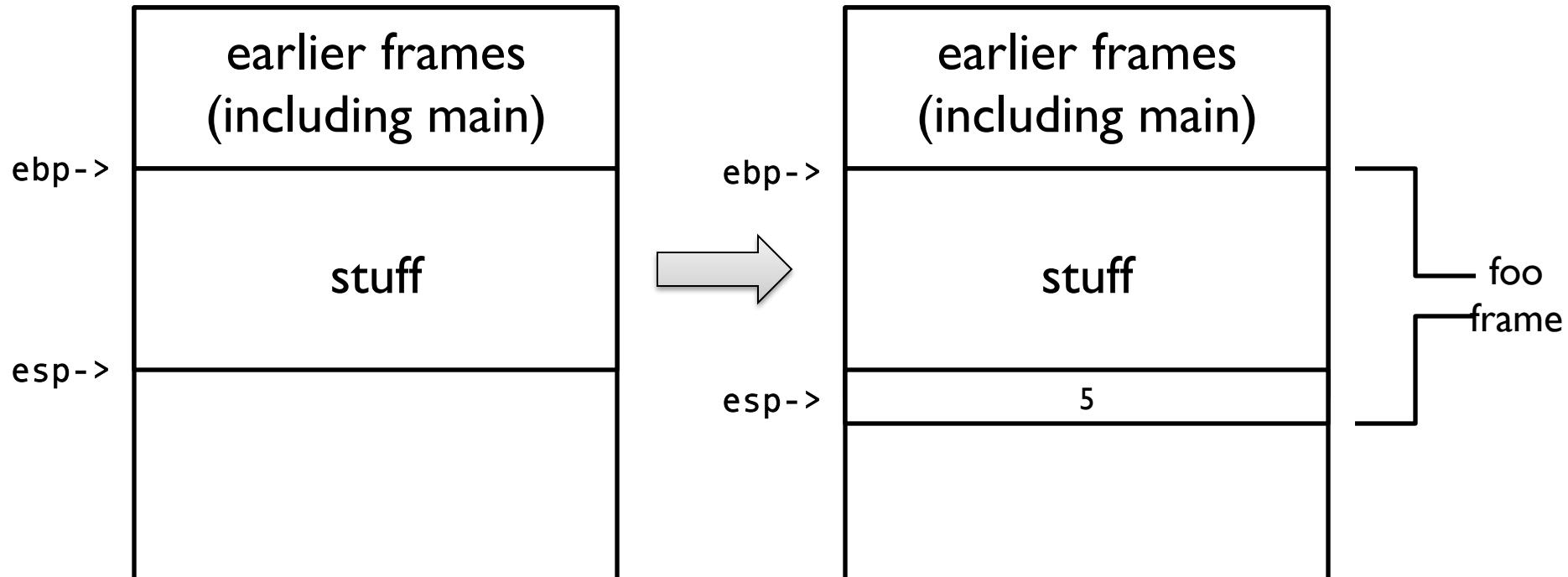
Stack Operations (x86) – Push (3)



foo:

```
...
    movl $0x5, %edx
    push %edx
```

Stack Operations (x86) – Push (4)



foo:

...
movl \$0x5, %edx
push %edx

What instructions are equivalent to
push %edx?

Stack Operations (x86) – Pop

Reminders

Stack x86

Stack x86_64

Data lab

Questions

Stack Operations (x86) – Pop

Run previous slides backwards

Stack (x86) – High Level

Disclaimer: Next slides are for your conceptual understanding only.
They are not representative of anything about order of execution.

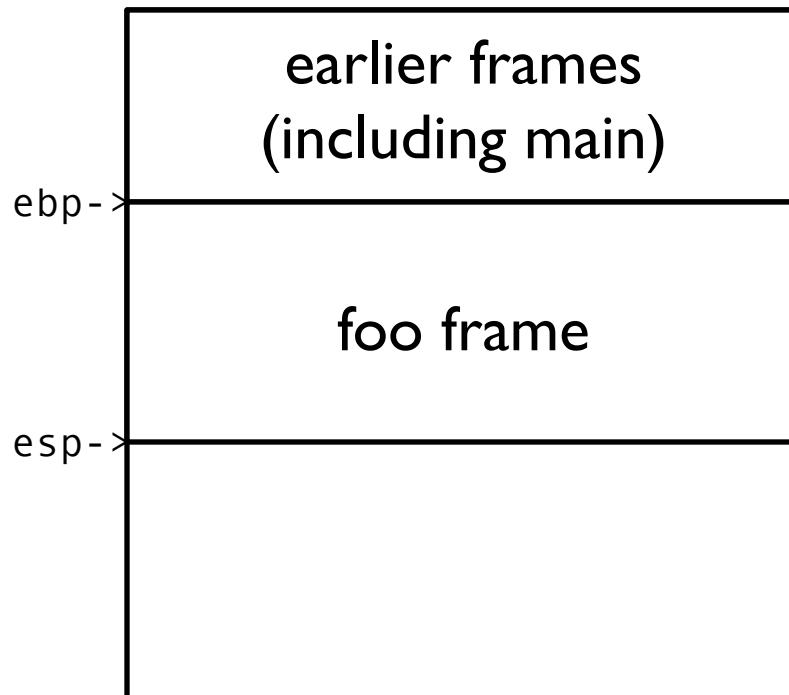
The views, opinions, positions or strategies expressed by the authors and those providing comments are theirs alone, and do not necessarily reflect the views, opinions, positions or strategies of Millennium Promise, the Earth Institute, UNDP or any employee thereof. Millennium Promise, the Earth Institute and UNDP make no representations as to accuracy, completeness, currentness, suitability, or validity of any information on this site and will not be liable for any errors, omissions, or delays in this information or any losses, injuries, or damages arising from its display or use.

The Millennium Villages blog reserves the right to delete, edit, or alter in any manner it sees fit blog entries or comments that it, in its sole discretion, deems to be obscene, offensive, defamatory, threatening, in violation of trademark, copyright or other laws, or is otherwise unacceptable.

Stack (x86) – High Level

```
int add3(int a, int b, int c)
{
    return a + b + c;
}
```

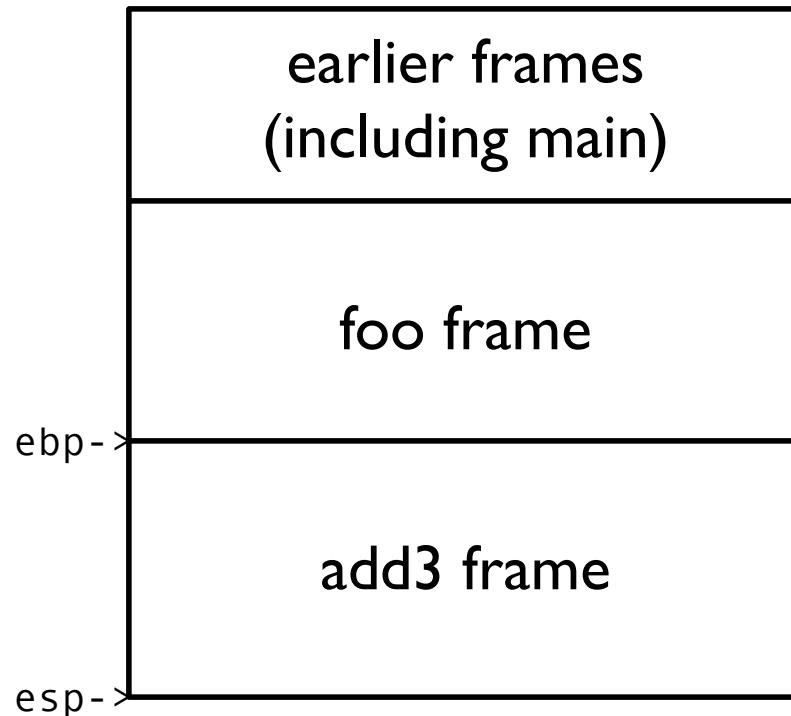
```
int foo(int x, int y)
{
    int b = 5;
    int s = add3(x, y, b);
    return x >> s;
}
```



Stack (x86) – High Level

```
int add3(int a, int b, int c)
{
    return a + b + c;
}
```

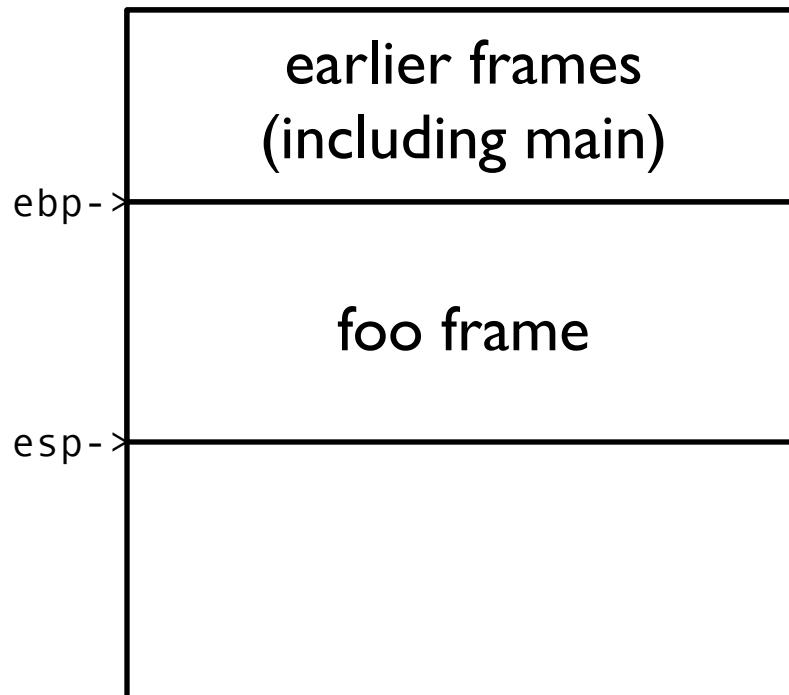
```
int foo(int x, int y)
{
    int b = 5;
    int s = add3(x, y, b);
    return x >> s;
}
```



Stack (x86) – High Level

```
int add3(int a, int b, int c)
{
    return a + b + c;
}
```

```
int foo(int x, int y)
{
    int b = 5;
    int s = add3(x, y, b);
    return x >> s;
}
```



Stack (x86) – Zoomed in

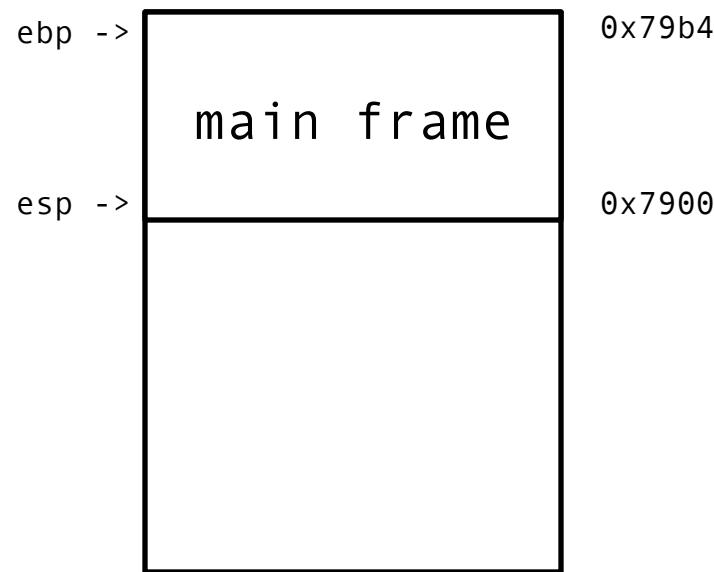
```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
    movl %eax, sum  
    movl sum, %eax  
    leave  
    ret
```

```
int sum;  
int add3(int a, int b, int c)  
{  
    return a + b + c;  
}  
  
int foo(void)  
{  
    sum = add3(3, 4, 5);  
    return sum;  
}
```

example stolen from princeton lecture slides

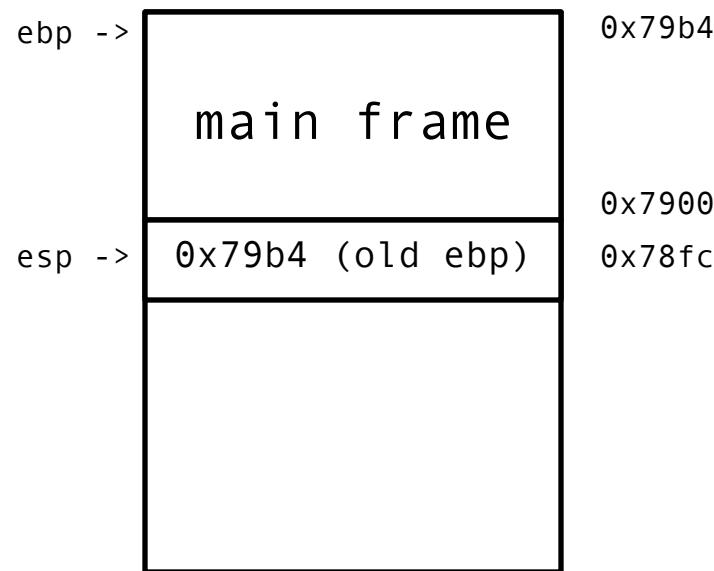
Stack (x86) – Zoomed in

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
    movl %eax, sum  
    movl sum, %eax  
    leave  
    ret
```



Stack (x86) – Zoomed in

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
    movl %eax, sum  
    movl sum, %eax  
    leave  
    ret
```



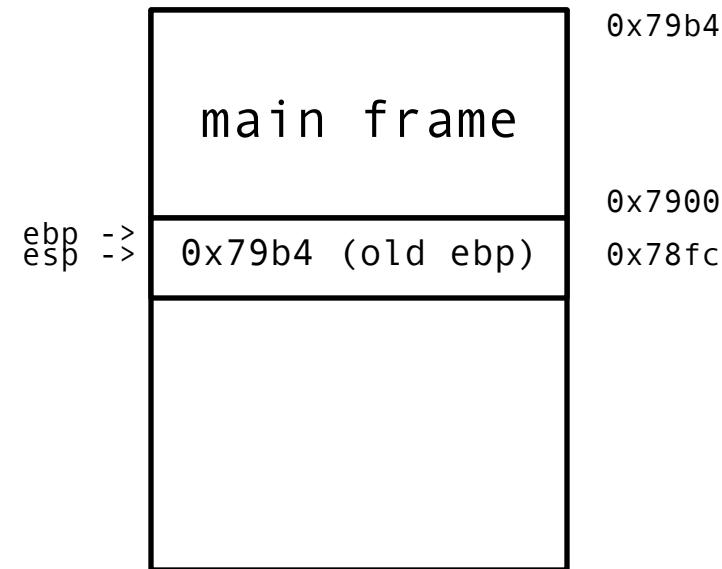
Stack (x86) – Zoomed in

add3:

```
pushl %ebp  
movl %esp, %ebp  
movl 12(%ebp), %edx  
movl 8(%ebp), %eax  
addl %edx, %eax  
addl 16(%ebp), %eax  
popl %ebp  
ret
```

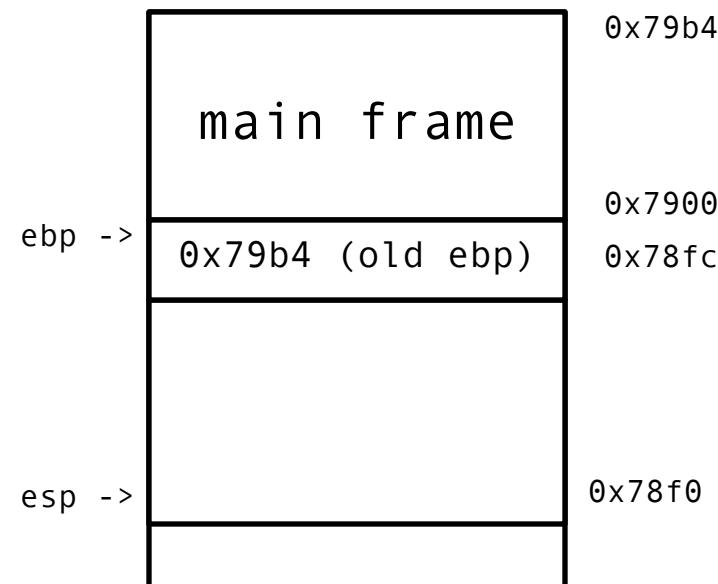
foo:

```
pushl %ebp  
movl %esp, %ebp  
subl $12, %esp  
movl $5, 8(%esp)  
movl $4, 4(%esp)  
movl $3, (%esp)  
call add3  
movl %eax, sum  
movl sum, %eax  
leave  
ret
```



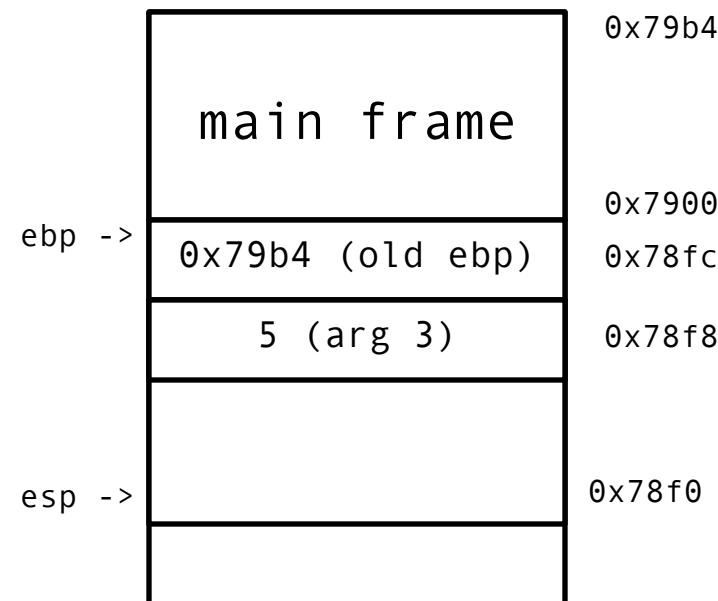
Stack (x86) – Zoomed in

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
    movl %eax, sum  
    movl sum, %eax  
    leave  
    ret
```



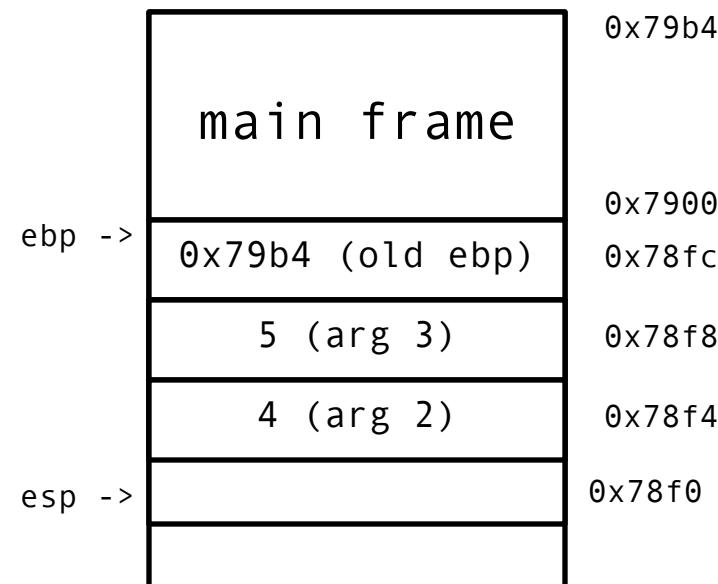
Stack (x86) – Zoomed in

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
    movl %eax, sum  
    movl sum, %eax  
    leave  
    ret
```



Stack (x86) – Zoomed in

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
    movl %eax, sum  
    movl sum, %eax  
    leave  
    ret
```



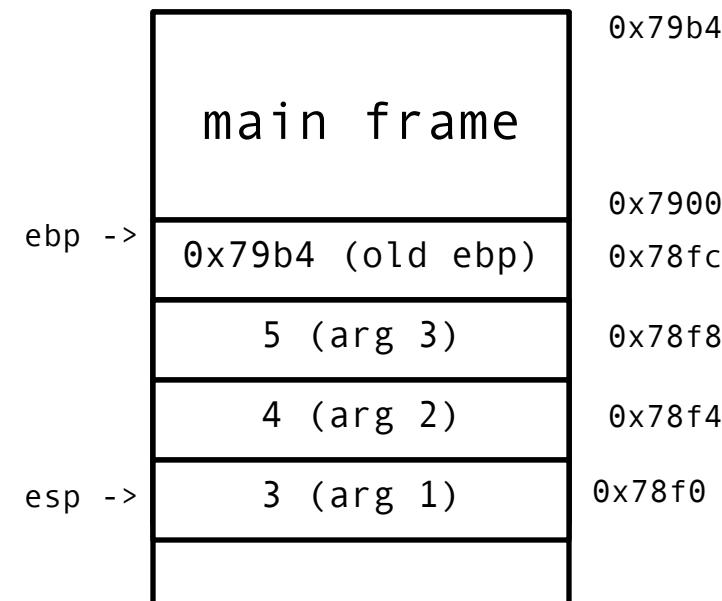
Stack (x86) – Zoomed in

add3:

```
pushl %ebp  
movl %esp, %ebp  
movl 12(%ebp), %edx  
movl 8(%ebp), %eax  
addl %edx, %eax  
addl 16(%ebp), %eax  
popl %ebp  
ret
```

foo:

```
pushl %ebp  
movl %esp, %ebp  
subl $12, %esp  
movl $5, 8(%esp)  
movl $4, 4(%esp)  
movl $3, (%esp)  
call add3  
movl %eax, sum  
movl sum, %eax  
leave  
ret`
```

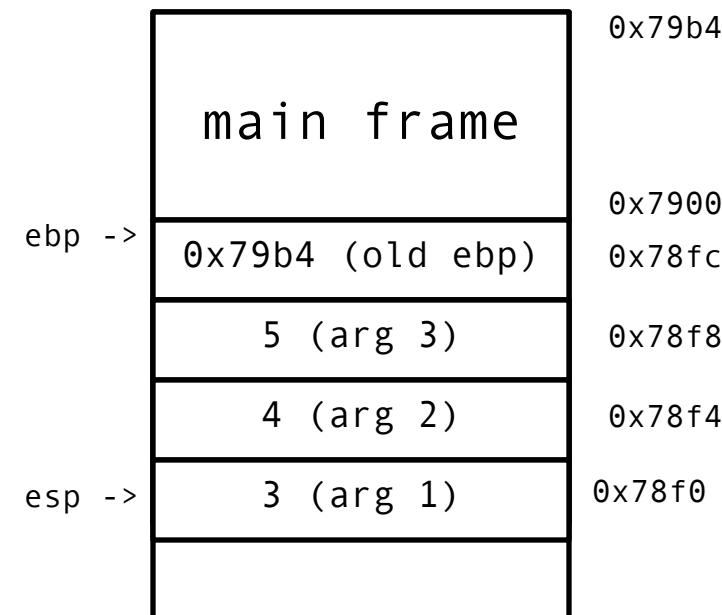


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

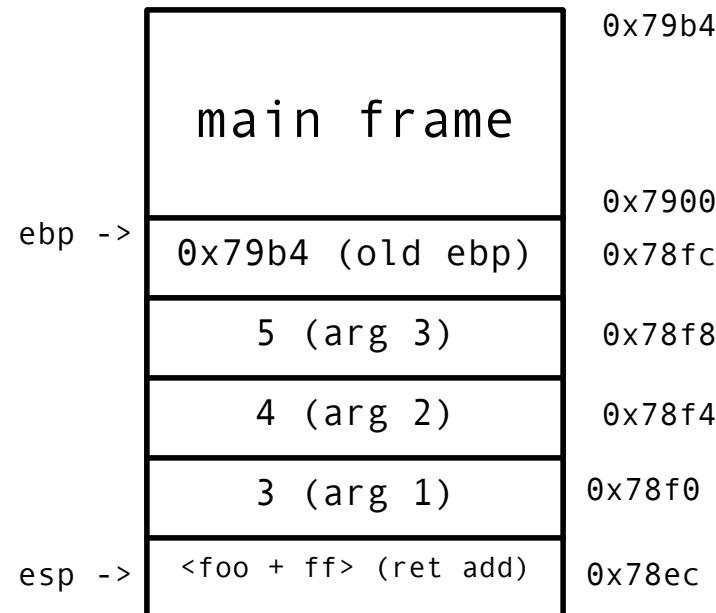
foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
            movl sum, %eax
            leave
            ret`
```



What happens
here?

Stack (x86) – Zoomed in

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
<foo+ff>  movl %eax, sum  
    movl sum, %eax  
    leave  
    ret`
```

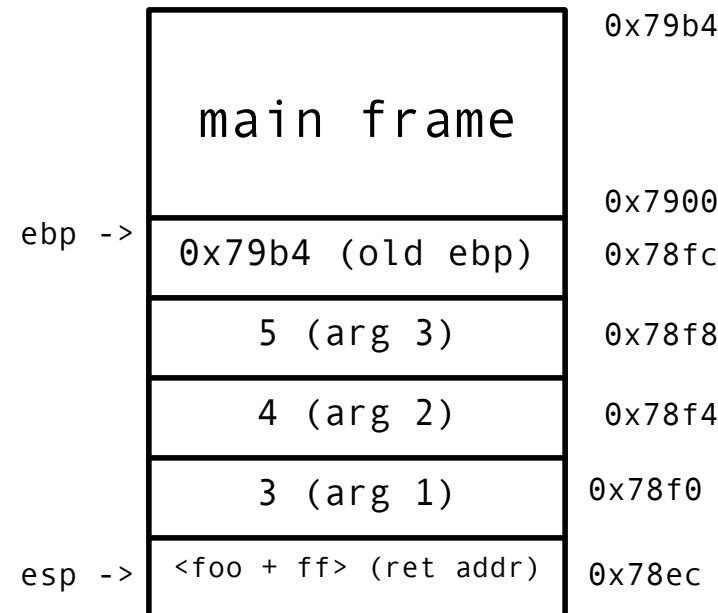


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
            movl sum, %eax
            leave
            ret`
```



What two
instructions was
that equivalent to?

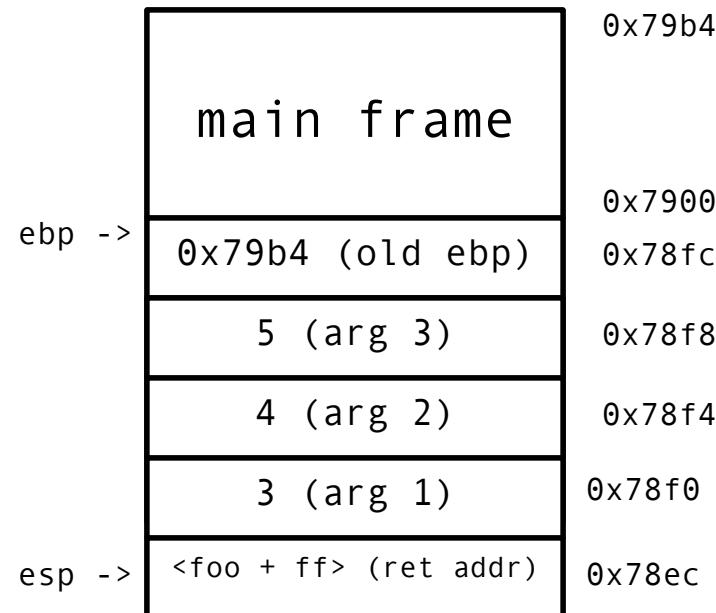
Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
            movl sum, %eax
            leave
            ret

```

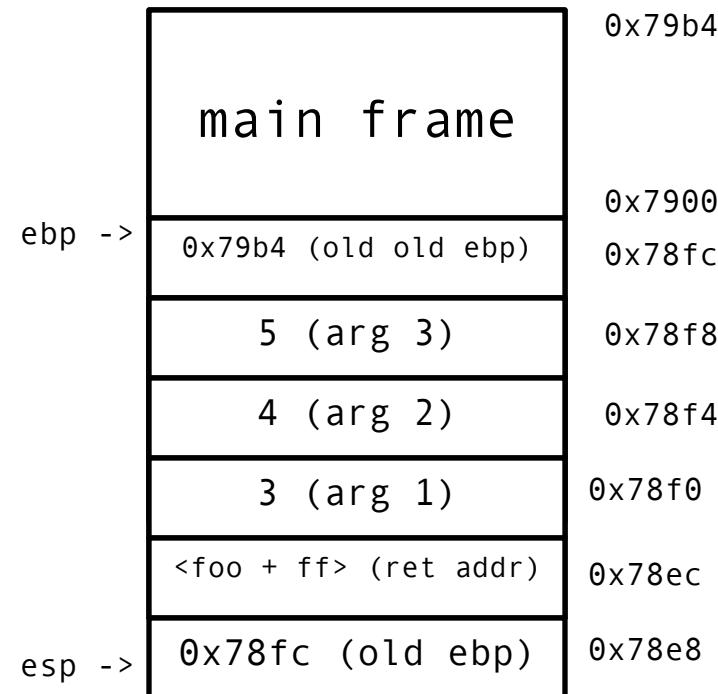


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret`
```

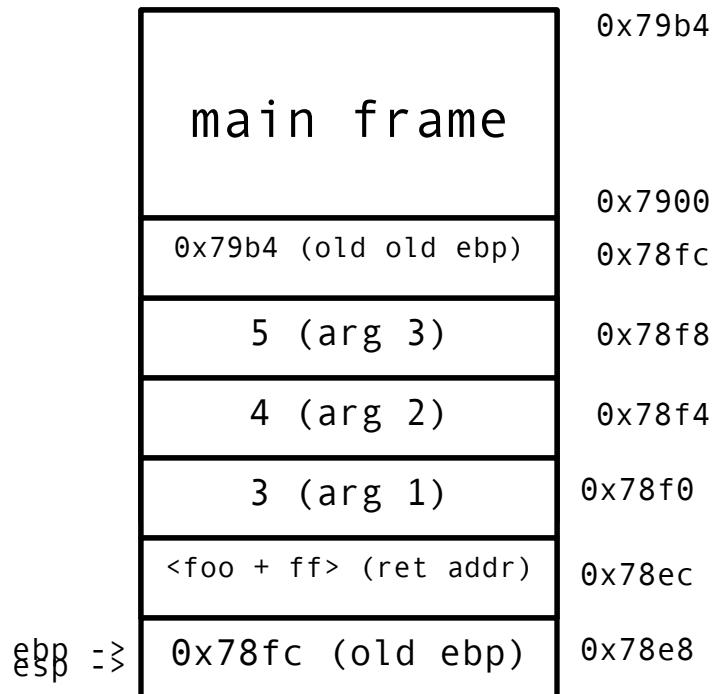


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret`
```

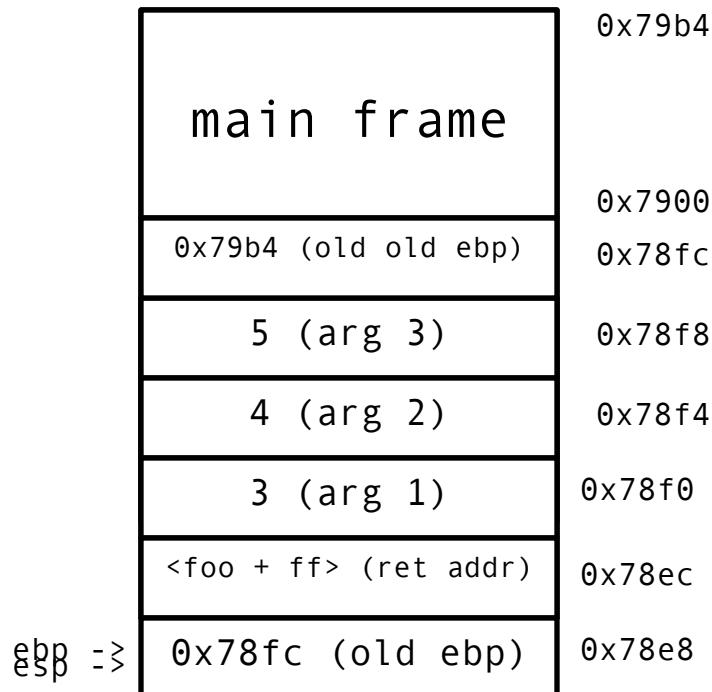


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp ←
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret`
```

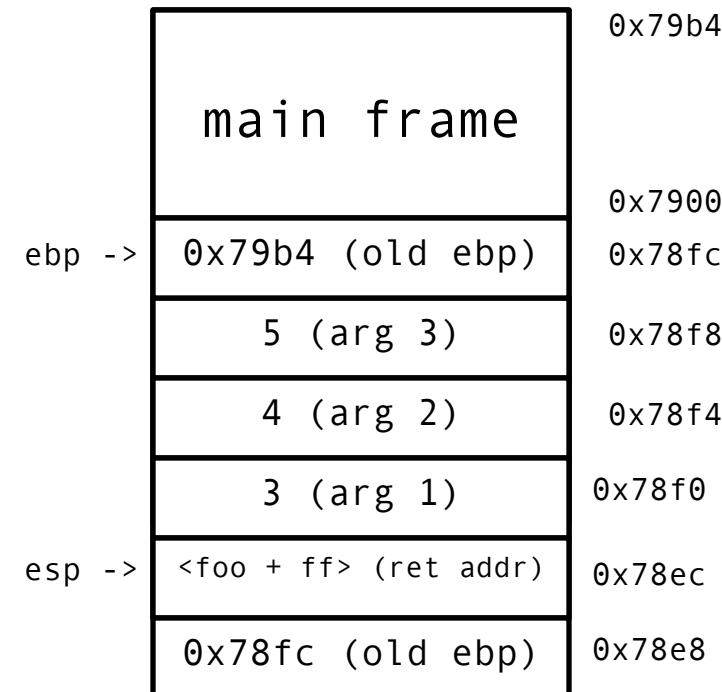


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret`
```

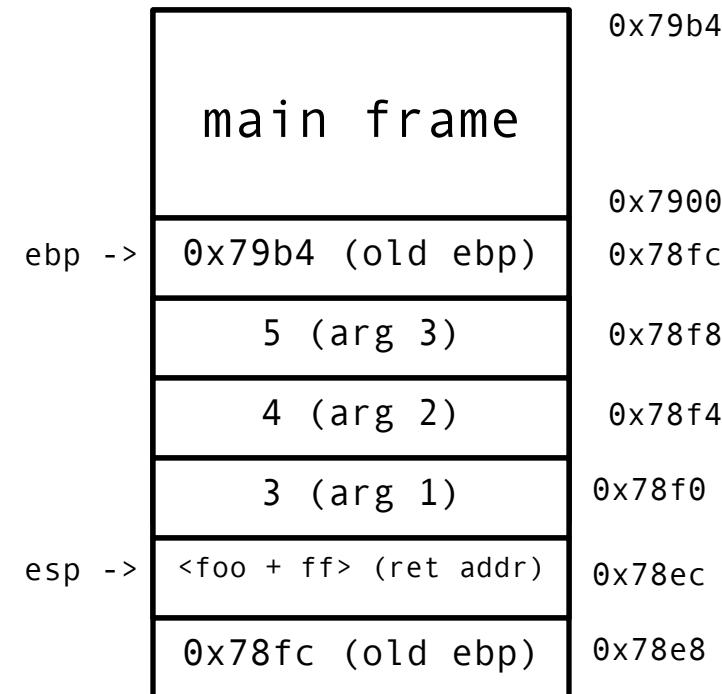


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret
  
```



What does `ret`
need to do?

Reminders

Stack x86

Stack x86_64

Data lab

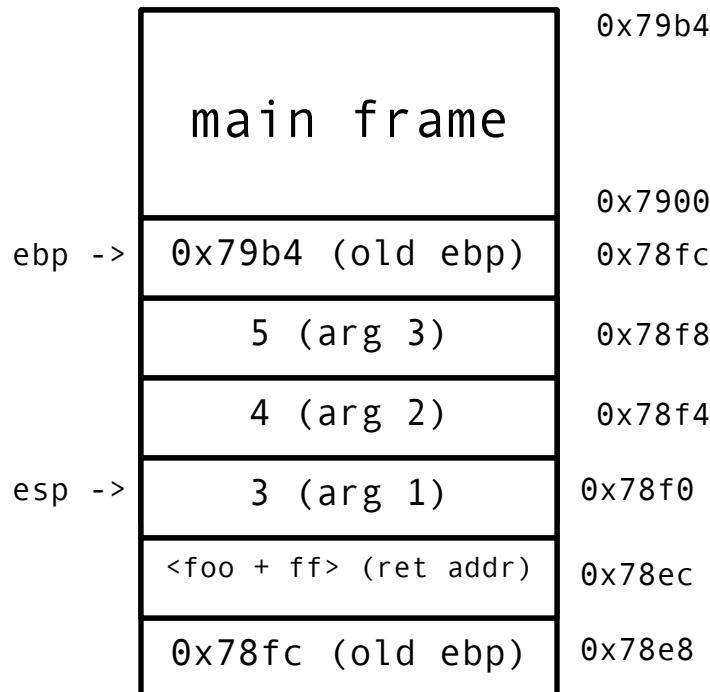
Questions

Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum <
    movl sum, %eax
    leave
    ret`
```

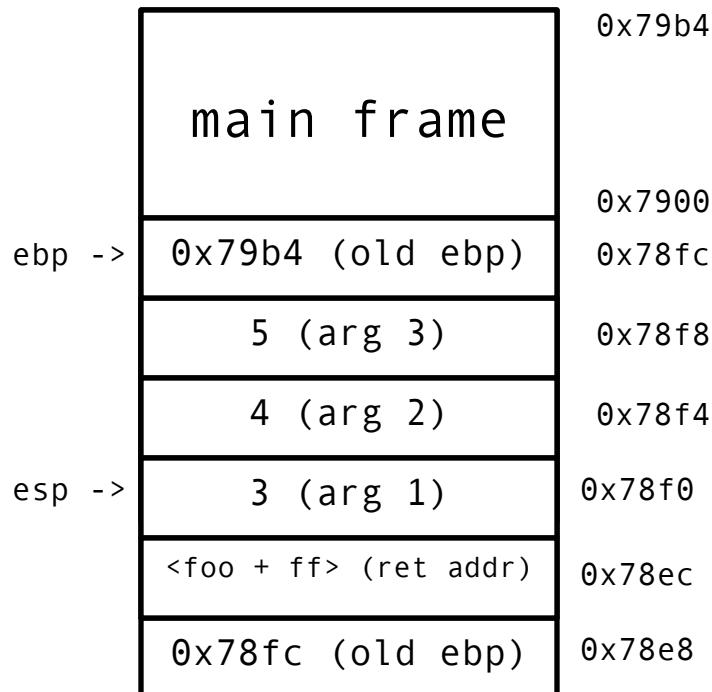


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum <
    movl sum, %eax
    leave
    ret`
```

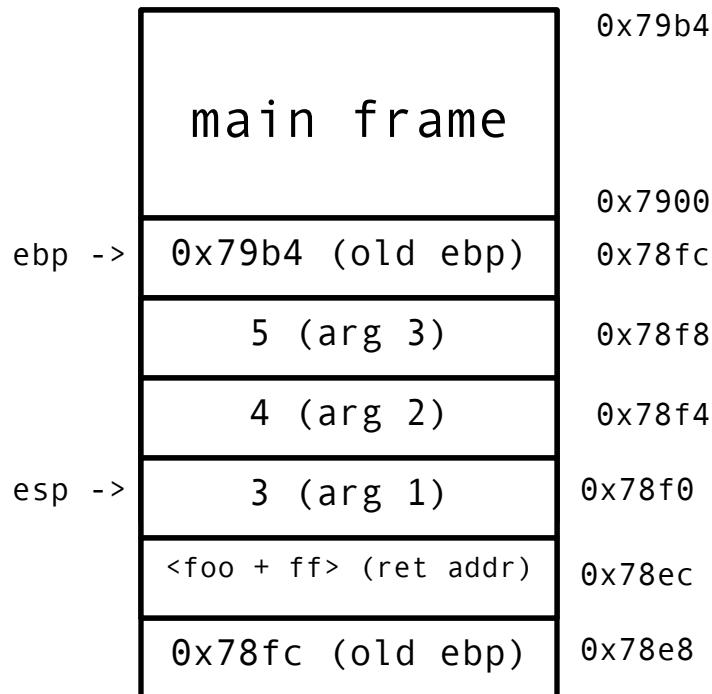


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
            movl sum, %eax
            leave
            ret`
```

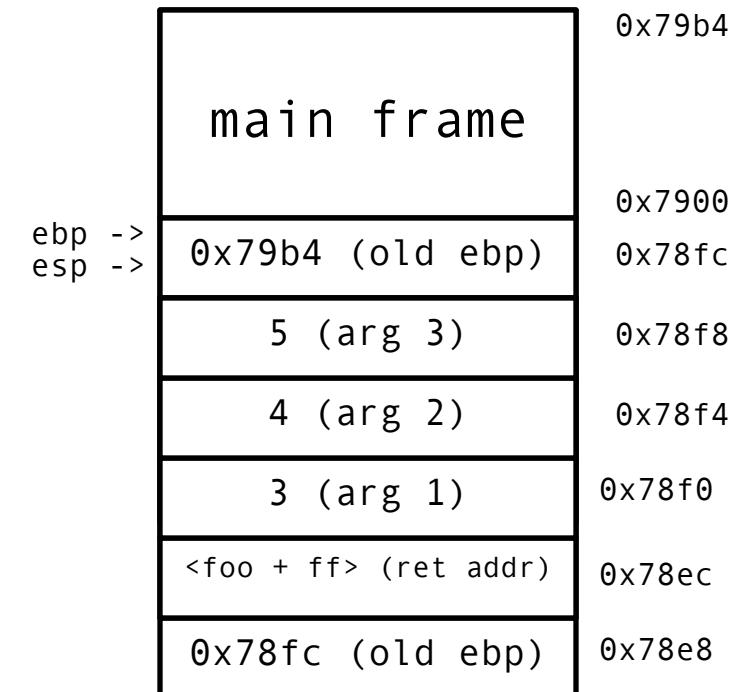


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret`
```

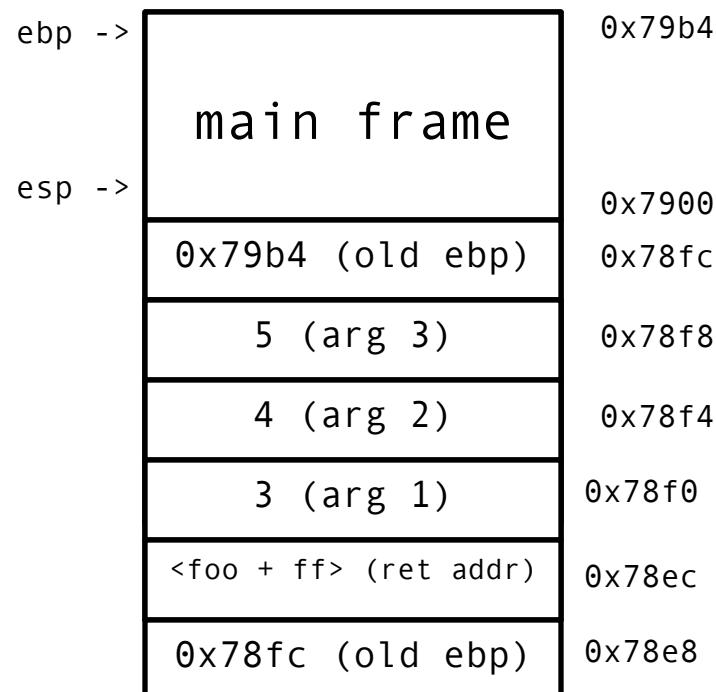


Stack (x86) – Zoomed in

```

add3:
    pushl %ebp
    movl %esp, %ebp
    movl 12(%ebp), %edx
    movl 8(%ebp), %eax
    addl %edx, %eax
    addl 16(%ebp), %eax
    popl %ebp
    ret

foo:
    pushl %ebp
    movl %esp, %ebp
    subl $12, %esp
    movl $5, 8(%esp)
    movl $4, 4(%esp)
    movl $3, (%esp)
    call add3
<foo+ff>  movl %eax, sum
    movl sum, %eax
    leave
    ret`
```



Stack (x86) – Zoomed in

We're about to (finally) move on. Any questions about that diagram?

Stack (x86) – Local variables

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
<foo+ff>  movl %eax, sum  
    movl sum, %eax  
    leave  
    ret`  
  
int sum;  
  
int add3(int a, int b, int c)  
{  
    return a + b + c;  
}  
  
int foo(void)  
{  
    sum = add3(3, 4, 5);  
    return sum;  
}  
  
example stolen from princeton lecture  
slides
```

Stack (x86) – Local variables

```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    movl 12(%ebp), %edx  
    movl 8(%ebp), %eax  
    addl %edx, %eax  
    addl 16(%ebp), %eax  
    popl %ebp  
    ret  
  
foo:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $16, %esp  
    movl $5, 8(%esp)  
    movl $4, 4(%esp)  
    movl $3, (%esp)  
    call add3  
<foo+ff>  movl %eax, 0xc(%esp)  
    movl 0xc(%esp), %eax  
    leave  
    ret`  
  
int sum;  
  
int add3(int a, int b, int c)  
{  
    return a + b + c;  
}  
  
int foo(void)  
{  
    int sum;  
    sum = add3(3, 4, 5);  
    return sum;  
}  
  
example stolen from princeton lecture  
slides
```

Stack (x86) – Local variables

Show: unsafe.c

Stack (x86) – Registers

- eax
 - return value
- eax, ecx, edx
 - caller save
- ebx, edi, esi
 - callee save

Stack (x86_x64) – Registers

- A simplification of x86
- `rax`
 - return value
- `rax, r10, r11`
 - caller save
- `rbx, rbp, r12 – r15`
 - callee save
- `rdi – arg1`
- `rsi – arg2`
- `rdx – arg3`
- `rcx – arg4`
- `r8 – arg5`
- `r9 – arg6`

Stack (x86_x64) – Stack Allocation

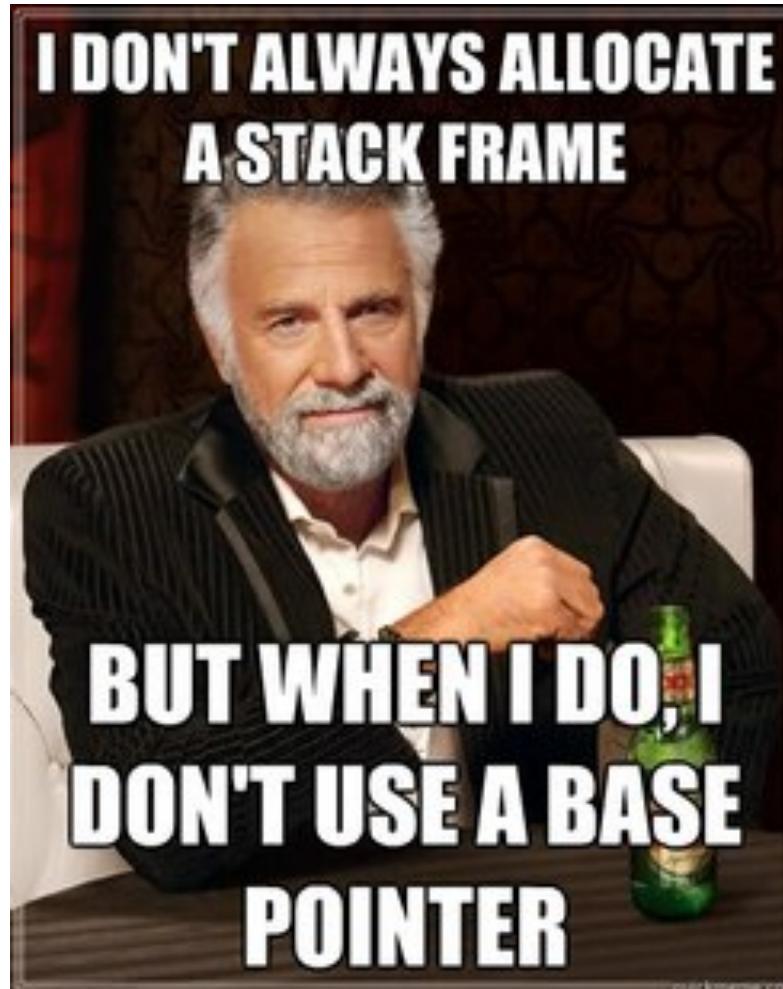
- Often none.

Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp

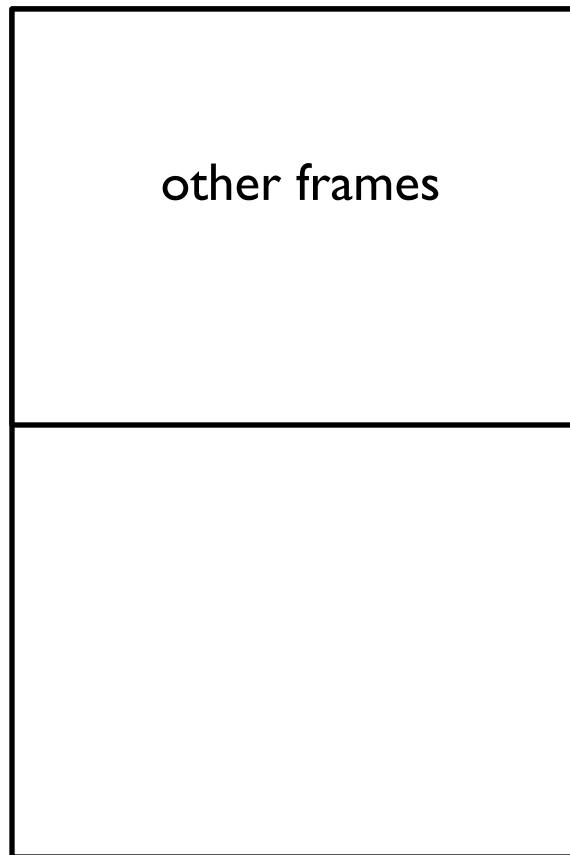
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp



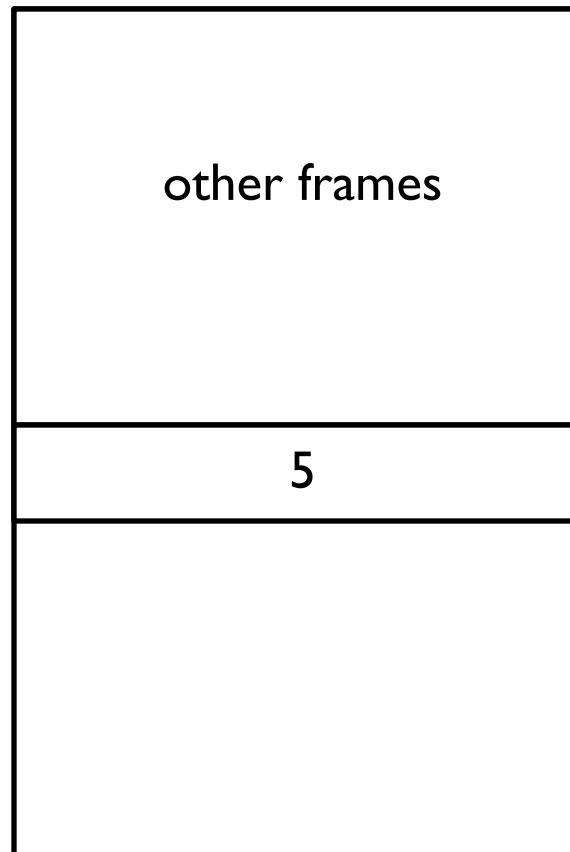
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- Say need \$5, \$4, \$3 on stack:
 $\text{rsp} - >$



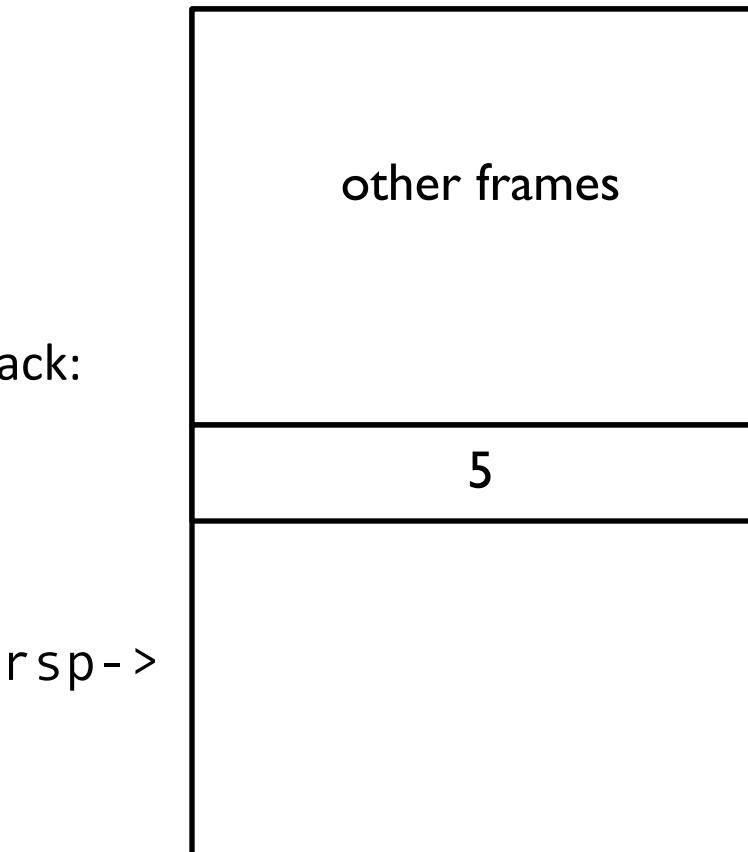
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- Say need \$5, \$4, \$3 on stack:
 $\text{rsp} ->$



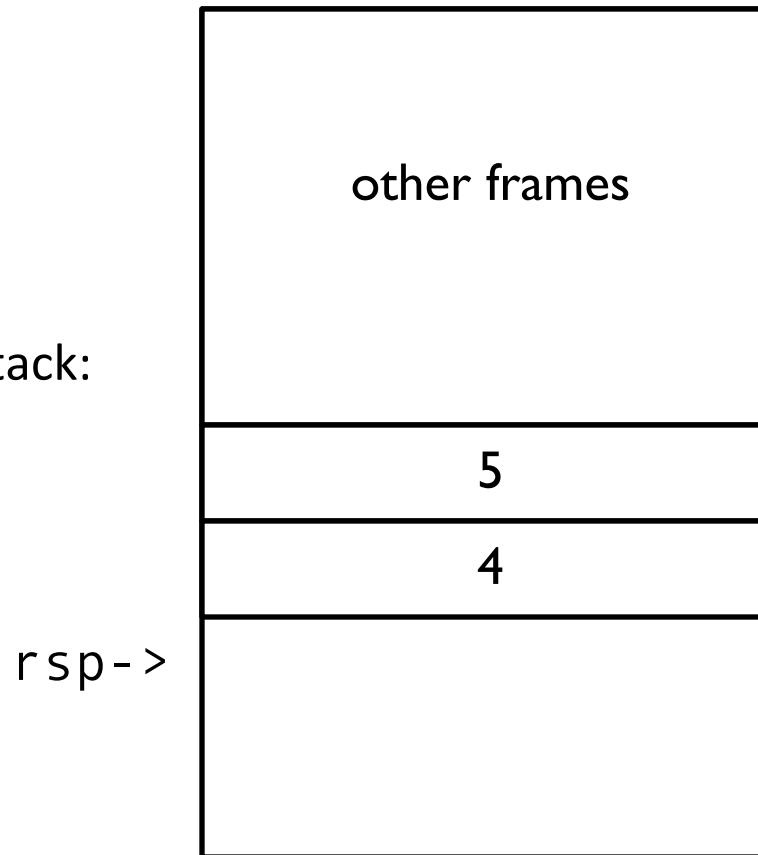
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- Say need \$5, \$4, \$3 on stack:



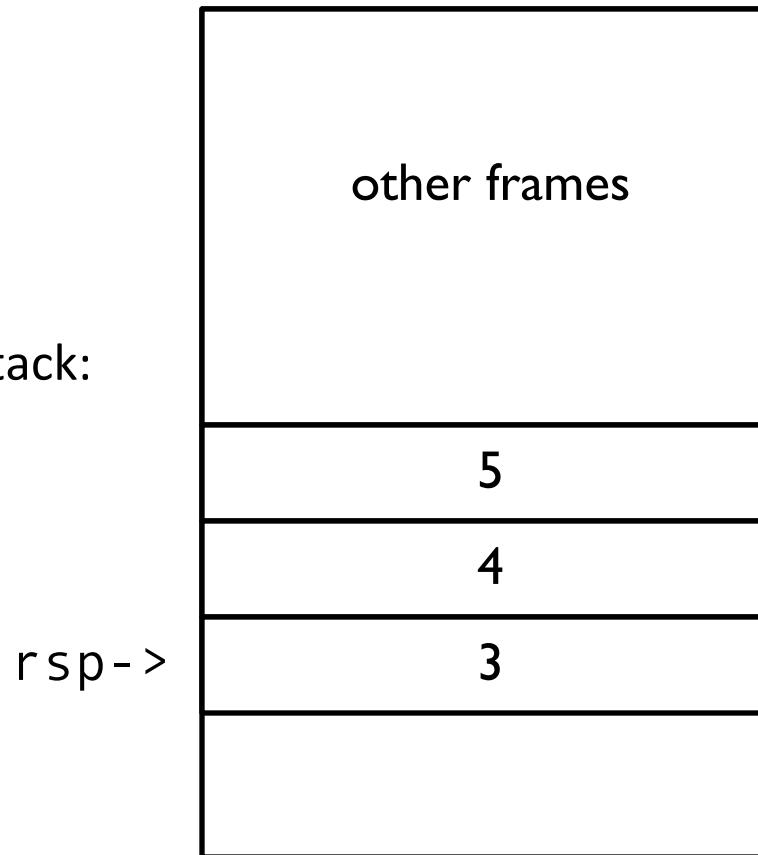
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- Say need \$5, \$4, \$3 on stack:



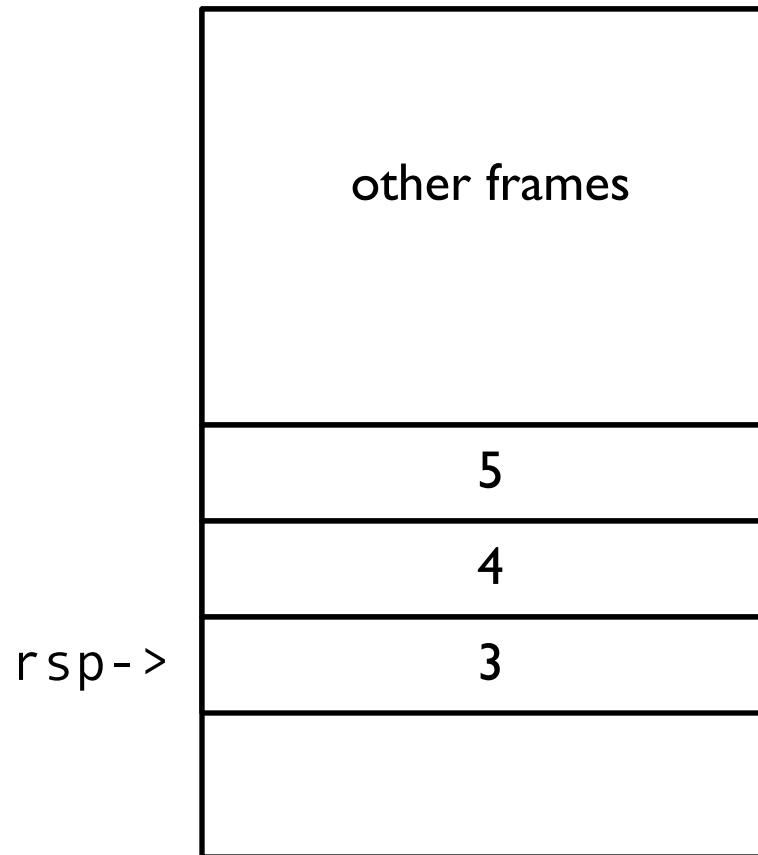
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- Say need \$5, \$4, \$3 on stack:



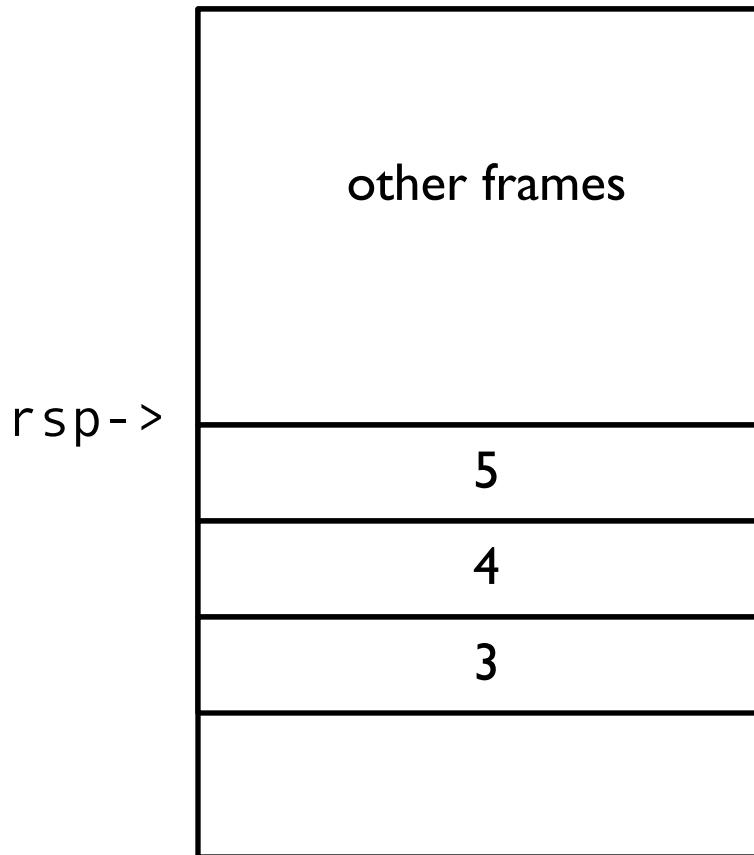
Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- When done?



Stack (x86_x64) – Stack Allocation

- Often none.
- No use of %rbp
- When done:



Data lab handback

Common things:

wrap lines to 80 characters!

```
/* calculate result */
return ~((x >> 31) | (!!x << 31)) >> n
+~0) & 0x841fffff;

if(x)
{
    return 0;
}
else
{
    statement1;
    statement2;
    ...
    statement52;
}
```

```
int x0, x1, x2, x3, x53;
if(x0)
{
    while(x1)
    {
        //printf("%d\n", x2);
        int absolute_value_of_input = |x3|;
        int thisMask = 0x583924;
    }
}
```

Reminders

Stack x86

Stack x86_64

Data lab

Questions

Questions?