Stacks

15-213: Introduction to Computer Systems
Recitation 5: February 9, 2014

Grant Skudlarek

Section D

Today: Stacks

- News
- Stack discipline review
  - Quick review of registers and assembly
  - Stack frames
  - Function calls
  - x86 (IA32) and x86-64
- Demo

News

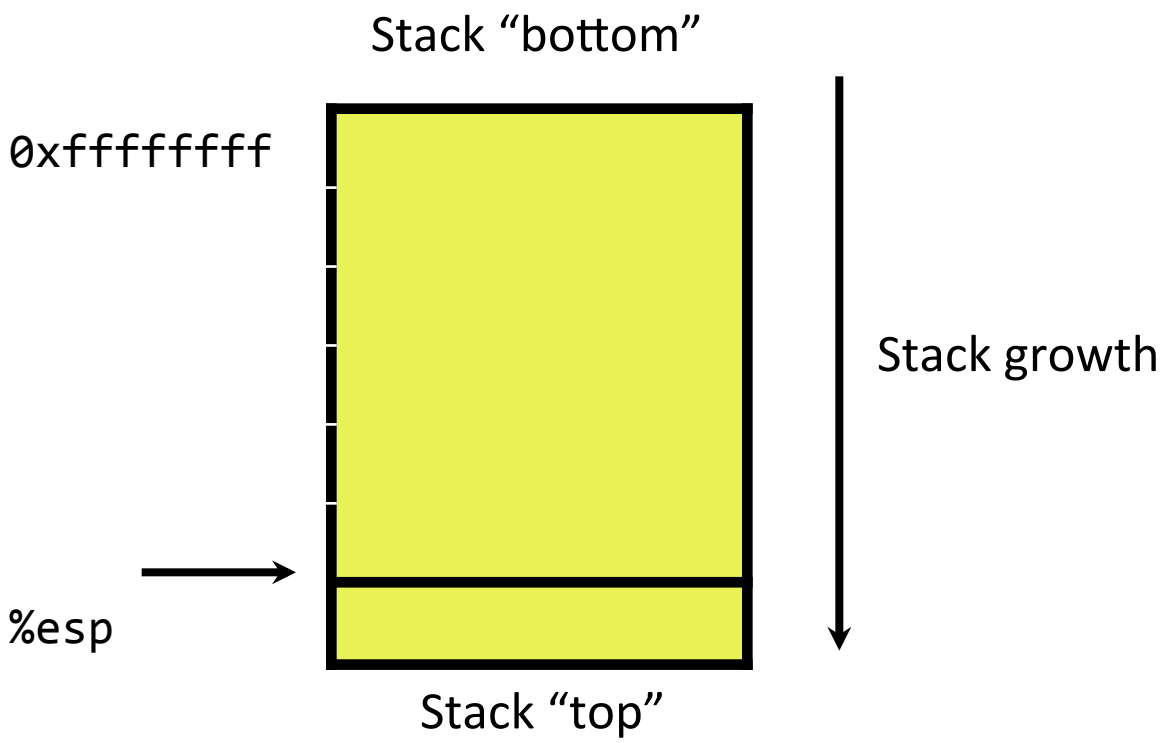
- bomblab is due tomorrow night
  - Don't use your late days yet
  - "If you wait till the last minute, it only takes a minute!"
- buflab is coming out tomorrow night
  - All about stacks
- Pro-tip: we love stack questions on exams
- We're here to help!
  - Office hours: Sunday – Thursday, 6 – 8 PM
  - Mailing List: 15-213-staff AT cs.cmu.edu

Quick review of registers (IA32)

- Caller saved: %eax, %ecx, %edx
  - You must save these before a function call if you need them
- Callee saved: %ebx, %edi, %esi
  - You must save these before any work if you need them
- Base pointer: %ebp
  - Points to the “bottom” of a stack frame
- Stack pointer: %esp
  - Points to the “top” of a stack frame
- Instruction pointer: %eip
  - Generally don’t need to worry about this one

IA32 stack

- This is a memory region that grows down
- Confusingly, refer to the bottom of the stack as the “top”
- %esp refers to the lowest stack address



pushing and popping

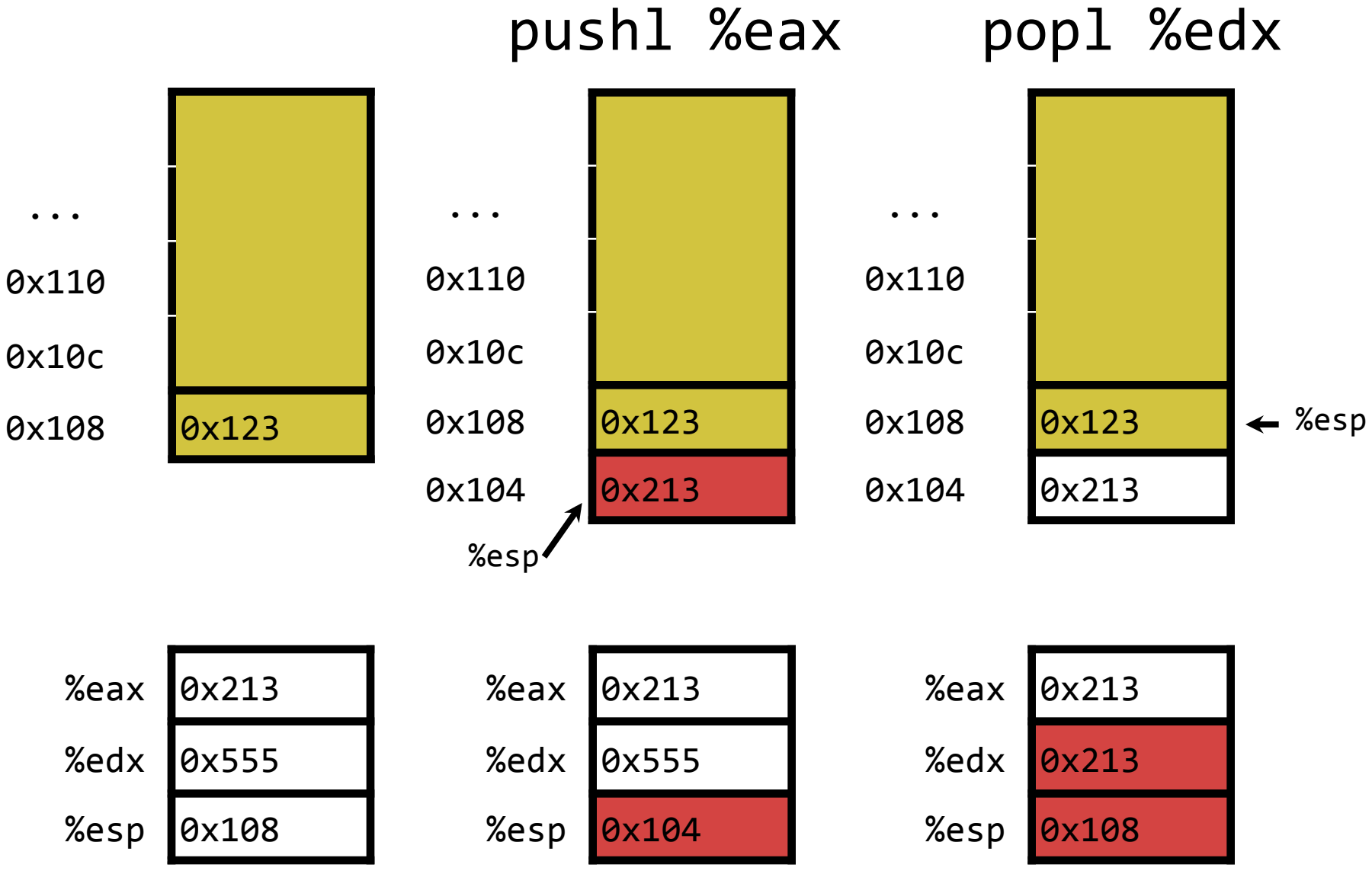
- It may be helpful to remember this correspondence (IA32)
  - Note: This is probably not how it actually works

```
pushl src  ⟶  subl $4,%esp
              movl src,(%esp)

popl dest  ⟶  movl (%esp),dest
              addl $4,%esp
```

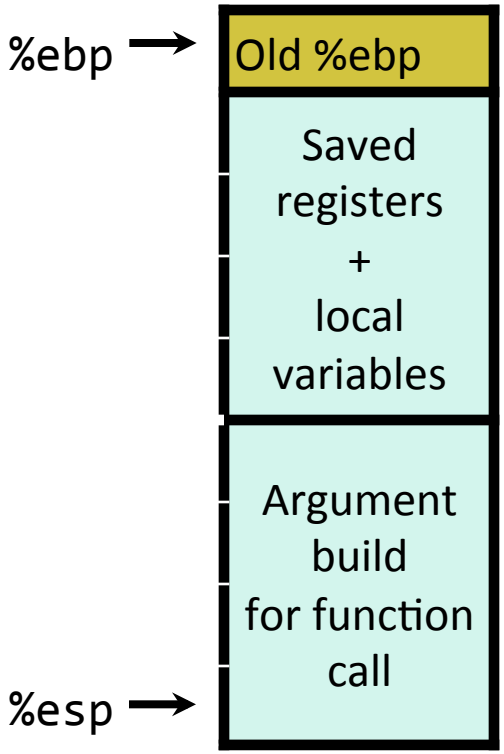
- %esp "points" to the top value on the stack

Quick example



Stack frames

- Every function call is given a stack frame
- What does a C function need?
  - Local variables (scalars, arrays, structs)
    - Scalars: if the compiler couldn't allocate enough registers
  - Space to save callee saved registers
  - Space to put computations
  - A way to give arguments and call other functions
  - A way to grab arguments
- Use the stack!

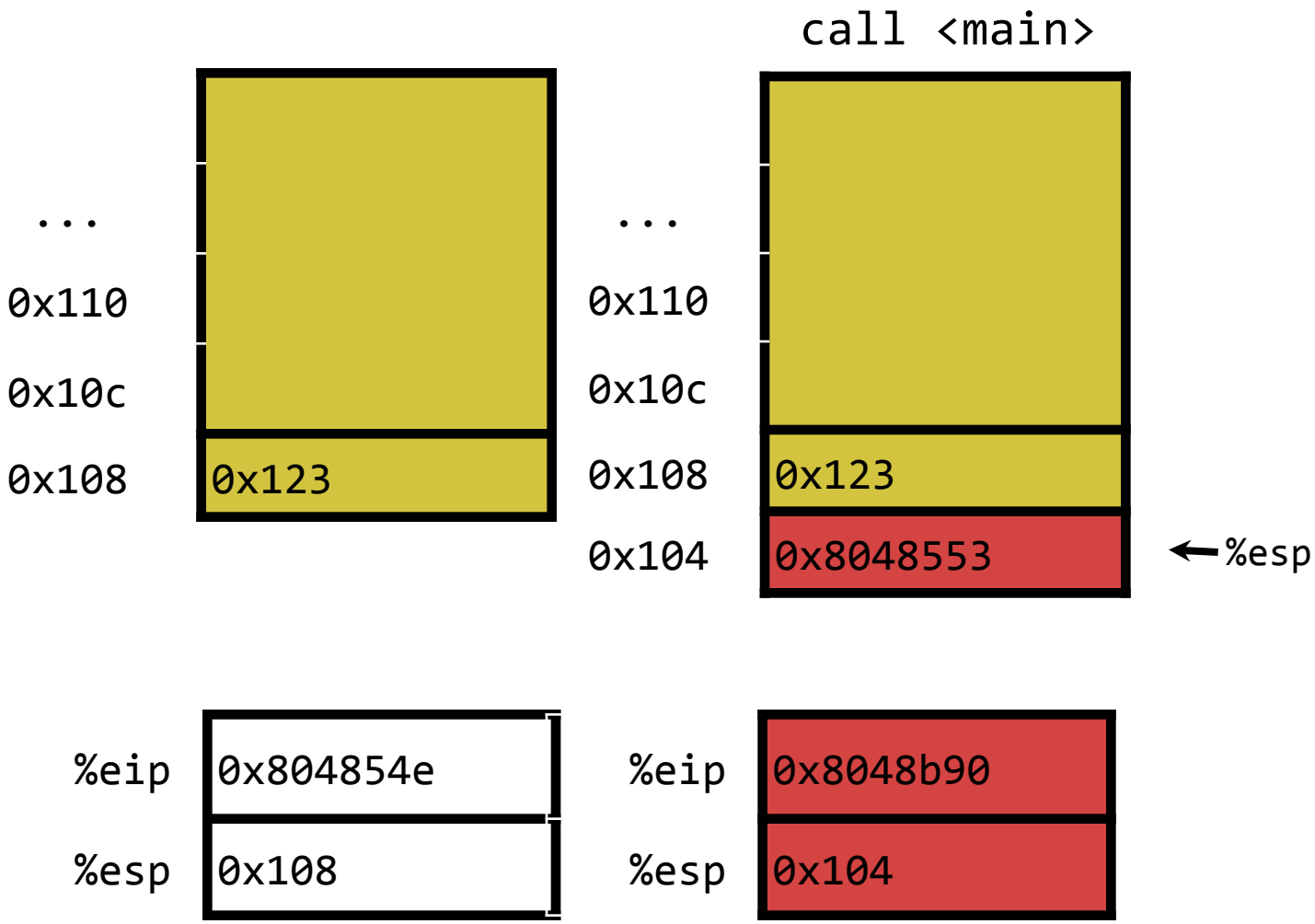


Function calls

- Use the stack for function calls
- Function call
  - `call label` Push "return address" on stack, jump to label
- Return address
  - Address of the instruction immediately after the `call`
  - Example from disassembly:
    - `804854e: e8 3d 06 00 00  call 8048b90 <main>`
    - `8048553: 50  pushl %eax`
  - Return address is `0x8048553`
- Returning from a function call
  - `ret` Pop return address [(%esp)] into %eip, keep running
  - Remember that the function's actual return value must be in %eax

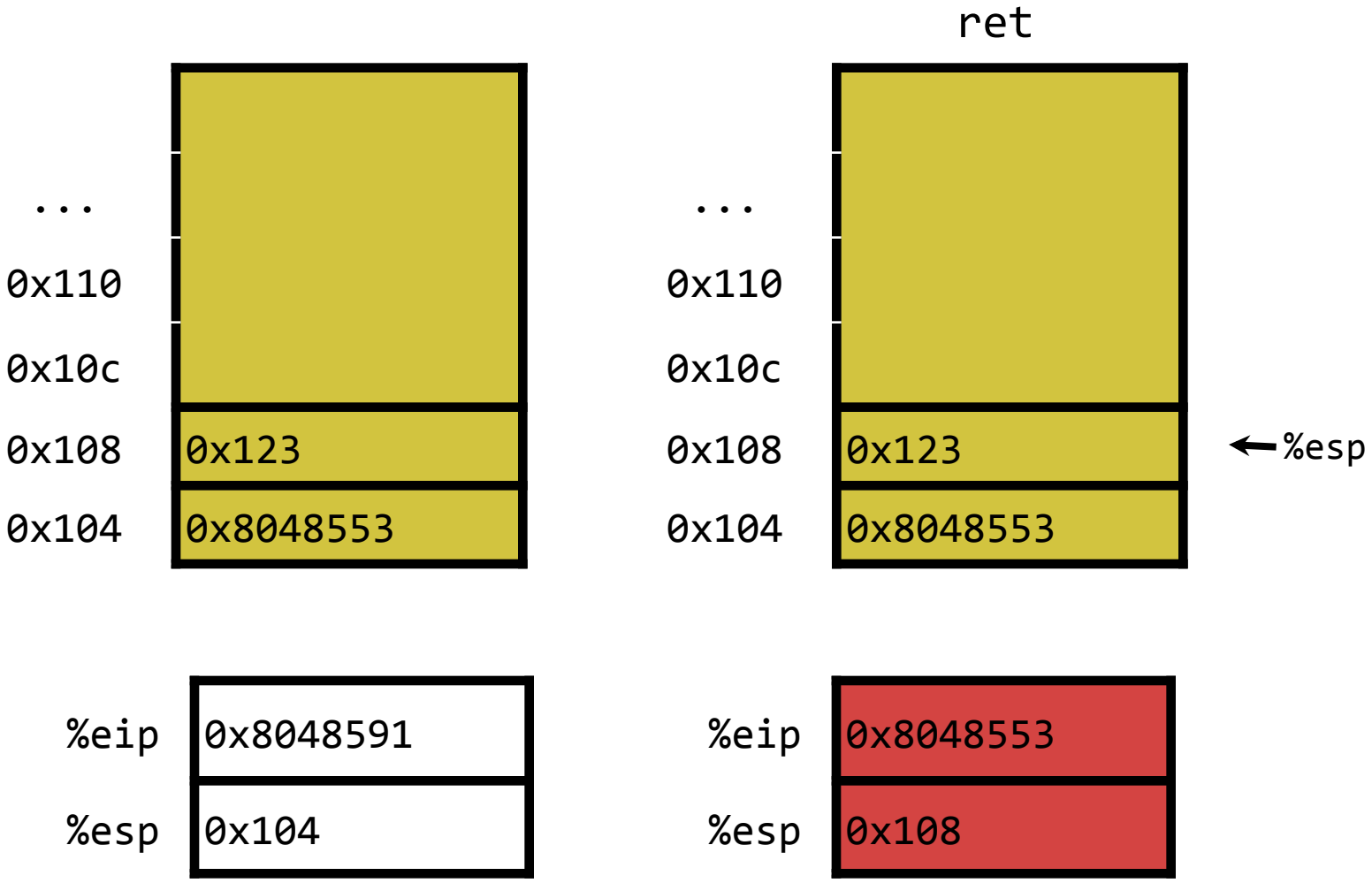
What does this look like?

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



Returning

- 8048591: c3 ret

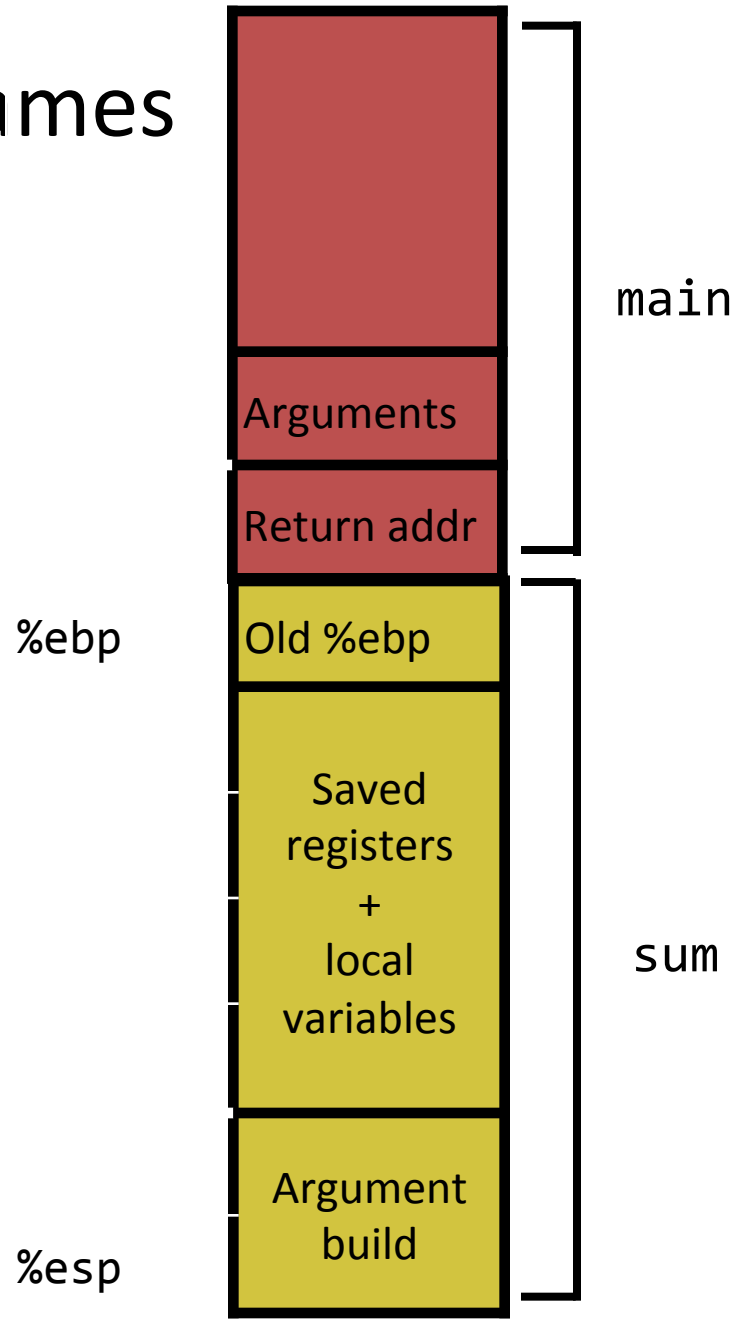


Function calls and stack frames

- Suppose you have

```
int main(void)
{
    int x = 3;
    return sum(x, 0);
}
```

- sum grabs arguments by reaching up the caller's stack frame!
- If we scale up this example, we see that arguments should be pushed in reverse order



Demo Time!

Buflab Summary

- You have one week for this lab
- With a solid grasp of stack discipline, shouldn't take very long
- Premise: exploit stack structure to gain control of the program, machine

Another Demo!

IA32 vs x86-64

- Remember in 64-bit this stuff is even easier
  - No more frame pointer (you are free to use %ebp/%rbp)
  - Many arguments are passed in registers
  - More registers = less stack space needed
- Overall a lot less stack usage
  - Good for performance - see memory hierarchy
- You are expected to know how the stack works for 64-bit
  - Even if no labs exercise these skills

Questions?

(stacks, bomblab, what is buflab)

(come to office hours if you need help)