Class 3

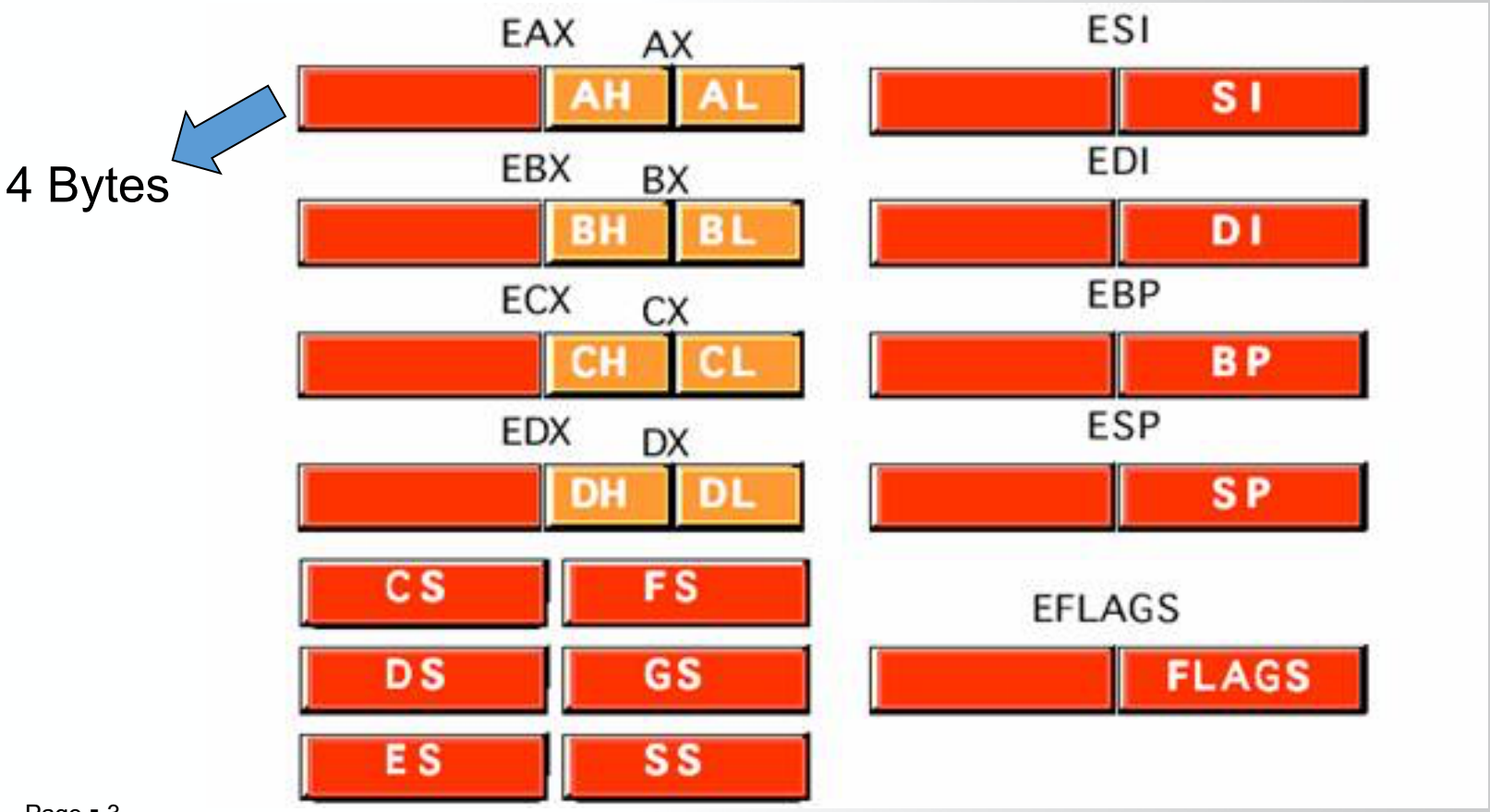
CSC 495/583 Topics of Software Security
X86 Assembly & Stack & Stack Frame

Dr. Si Chen (schen@wcupa.edu)

Review

General-purpose Registers

- The **eight** 32-bit general-purpose data registers are used to hold operands for logical and arithmetic operations, operands for address calculations and memory pointers



X86 ASM

MOV

- Move **reg/mem** value to **reg/mem**
  - mov A, B is "Move B to A" (A=B)
  - Same data size

mov eax, 0x1337

mov bx, ax

mov [esp+4], bl

0x1234

More About Memory Access

- mov ebx, [esp + eax * 4] **Intel**
- mov (%esp, %eax, 4), %ebx **AT&T**
- mov BYTE [eax], 0x0f
  You must indicate the data size: BYTE/WORD/DWORD

ADD / SUB

- ADD / SUB
- Normallly "reg += reg" or "reg += imm"
- Data size should be equal
  - add eax, ebx
  - sub eax, 123
  - sub eax, BL ; Illegal

Jump

- Unconditional jump: jmp
- Conditional jump: je/jne
  and ja/jae/jb/jbe/jg/jge/jl/jle ...
- Sometime with ”cmp A, B” -- compare these two values and set eflags
- Conditional jump is decided by some of the eflags bits.

ref

The JMP Instruction

- JMP (jump) instruction causes an **unconditional jump**
- Syntax is:

JMP destination/target_label

- JMP can be used to get around the range restriction [126/127 byte]
- Flags – no change

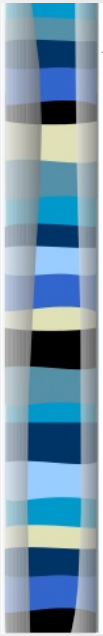
```
TOP:

; body of the loop, say 2 instructions
DEC     CX      ; decrement counter
JNZ     TOP     ; keep looping if CX > 0
MOV     AX, BX
```

```
TOP:

; the loop body contains so many instructions
; that label TOP is out of range for JNZ. Solution is-
        DEC     CX
        JNZ     BOTTOM
        JMP     EXIT
BOTTOM:
        JMP     TOP
EXIT:
        MOV     AX, BX
```

Section 6-3: Assembly Language Programming



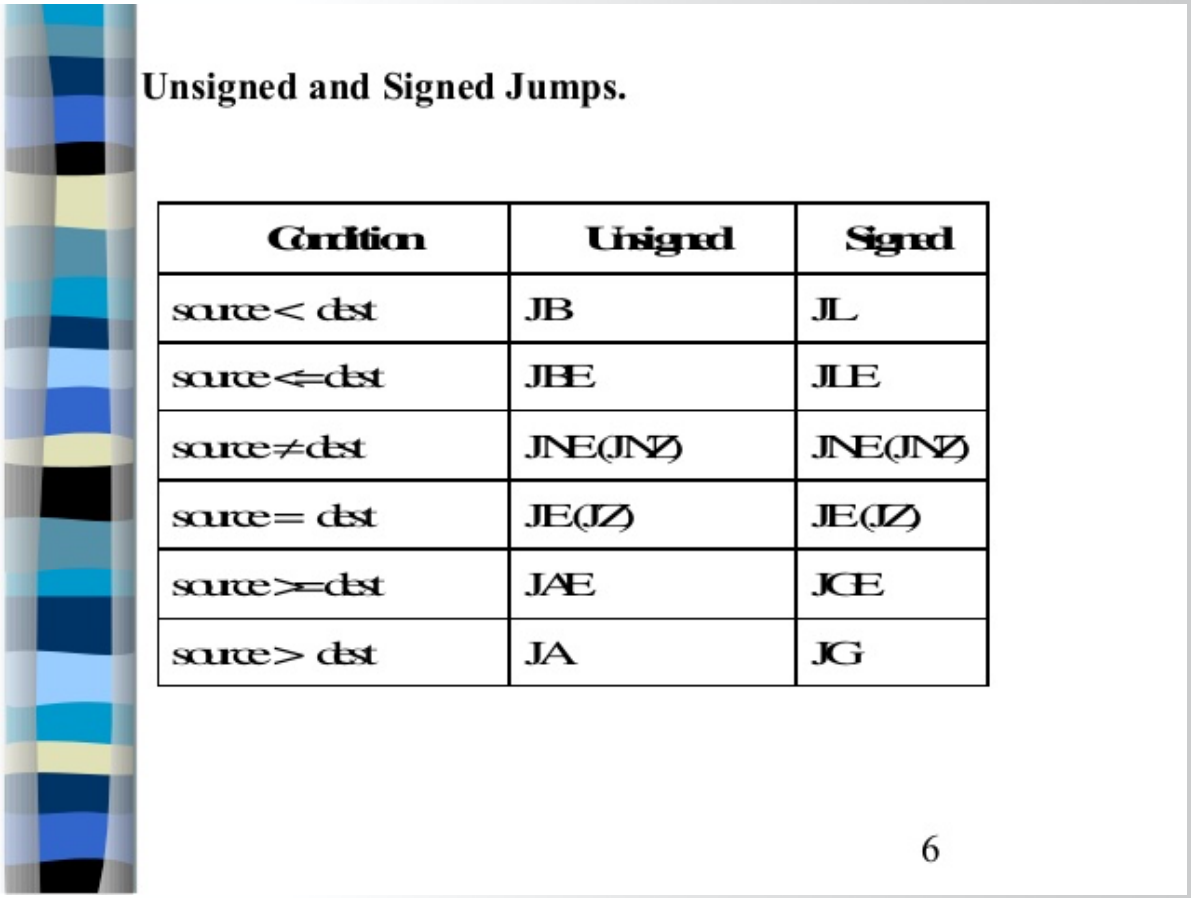
Unsigned and Signed Jumps.

Condition	Unsigned	Signed
source < dst	JB	JL
source <= dst	JBE	JLE
source ≠ dst	JNE(JNZ)	JNE(JNZ)
source = dst	JE(JZ)	JE(JZ)
source >= dst	JAE	JGE
source > dst	JA	JG

6

Jump

- ja/jae/jb/jbe are unsigned comparison
- jg/jge/jl/jle are signed comparison

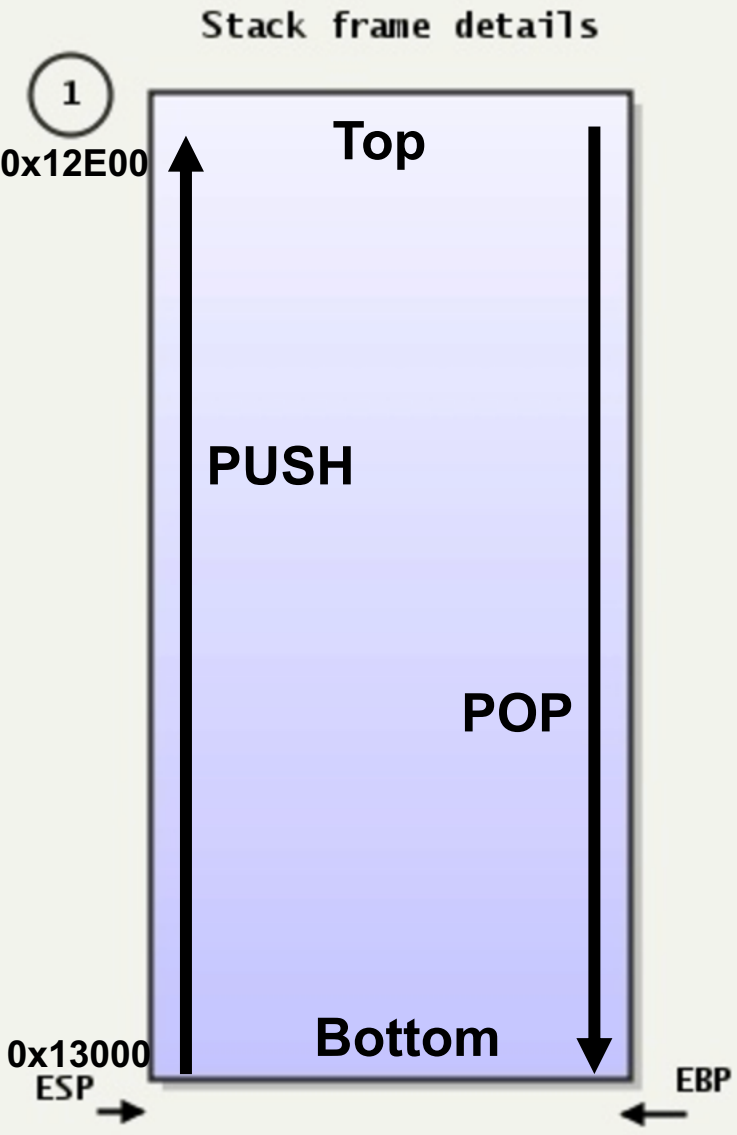


Unsigned and Signed Jumps.

Condition	Unsigned	Signed
source< dst	JB	JL
source≤dst	JBE	JLE
source≠dst	JNE(JNZ)	JNE(JNZ)
source= dst	JE(JZ)	JE(JZ)
source≥dst	JAE	JGE
source> dst	JA	JG

6

The Stack



Stack:

- A special region of your computer's memory that **stores temporary variables** created by each functions
- The stack is a "**LIFO**" (last in, first out) data structure
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

Properties:

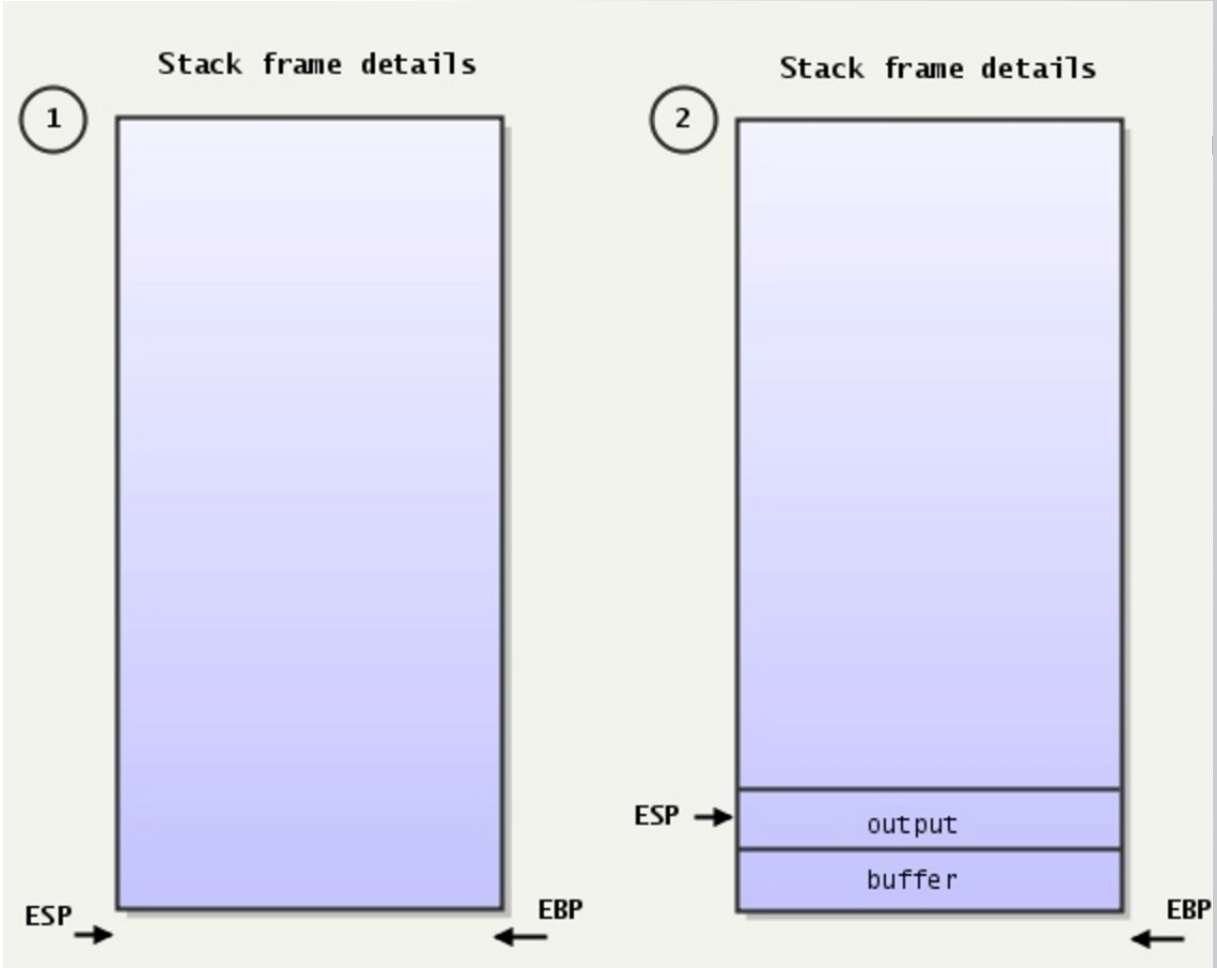
- the stack grows and shrinks as functions **push and pop** local **variables**
- there is no need to manage the memory yourself, variables are allocated and freed **automatically**
- the **stack has size limits**
- stack variables only exist while the function that created them, is running

EBP—Pointer to data on the stack
ESP—Stack pointer

The Stack

Stack:

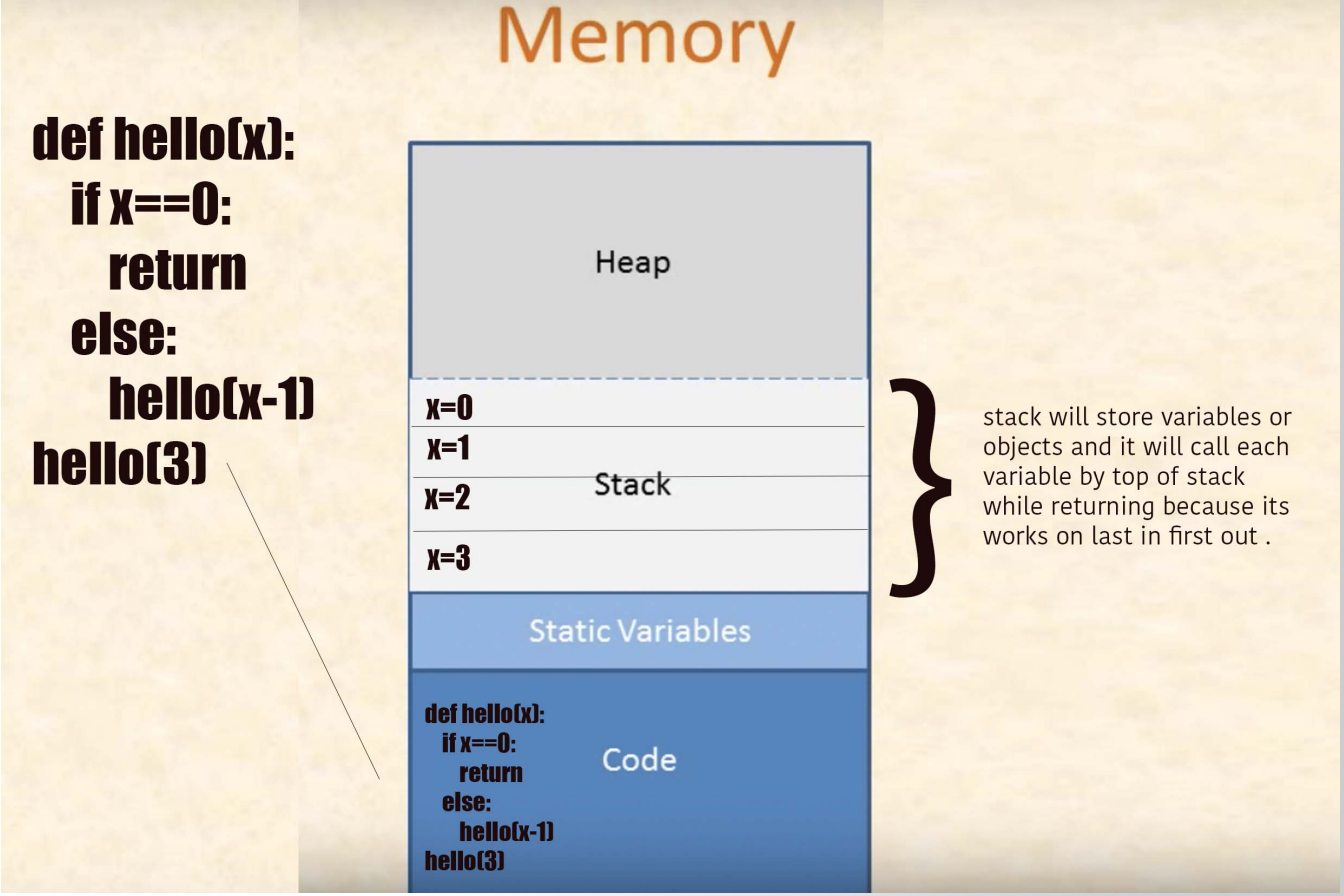
- A special region of your computer's memory that **stores temporary variables** created by each functions
- The stack is a "**LIFO**" (last in, first out) data structure
- Once a stack variable is freed, that region of memory becomes available for other stack variables.



Stack Frame

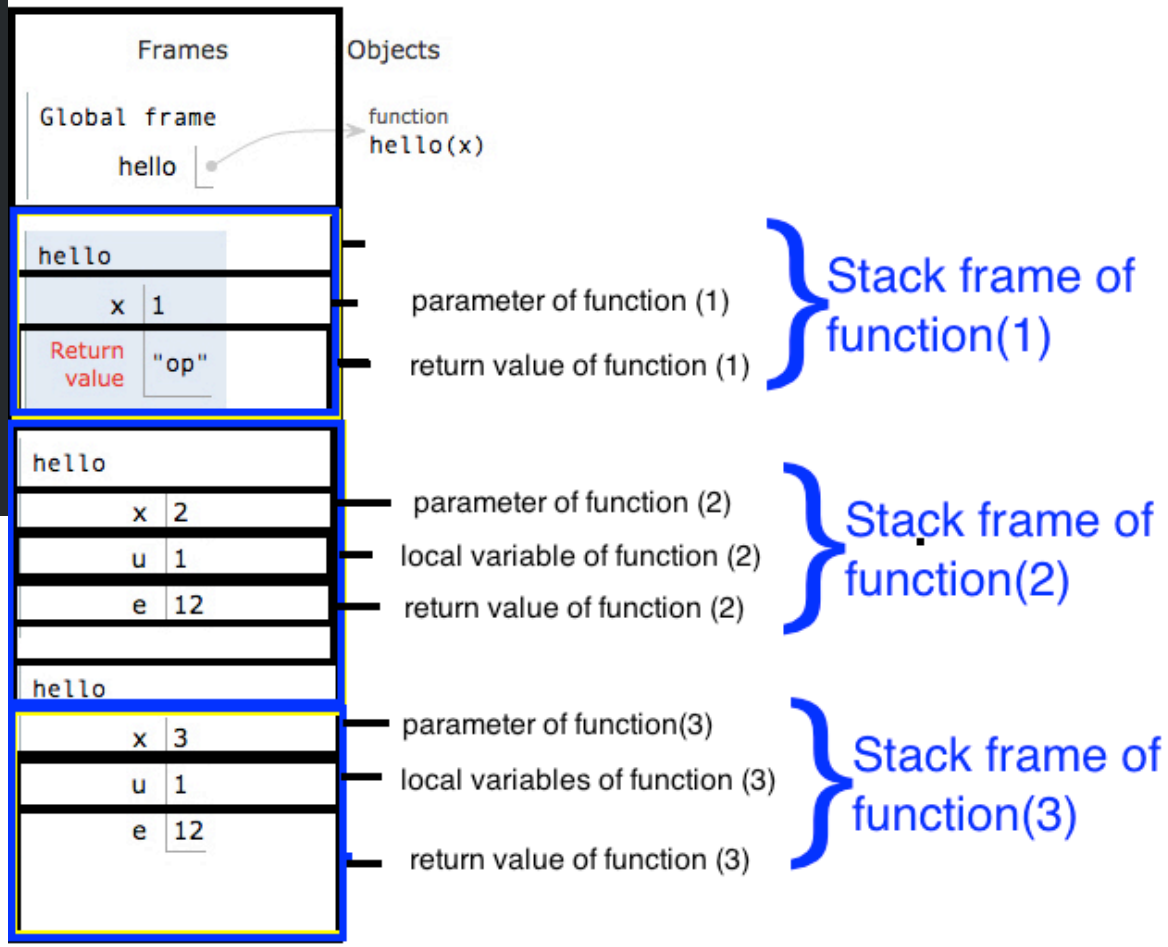
Stack Frame

- A stack frame is **a frame of data that gets pushed onto the stack**.
- In the case of a **call stack**, a stack frame would represent **a function call and its argument data**.

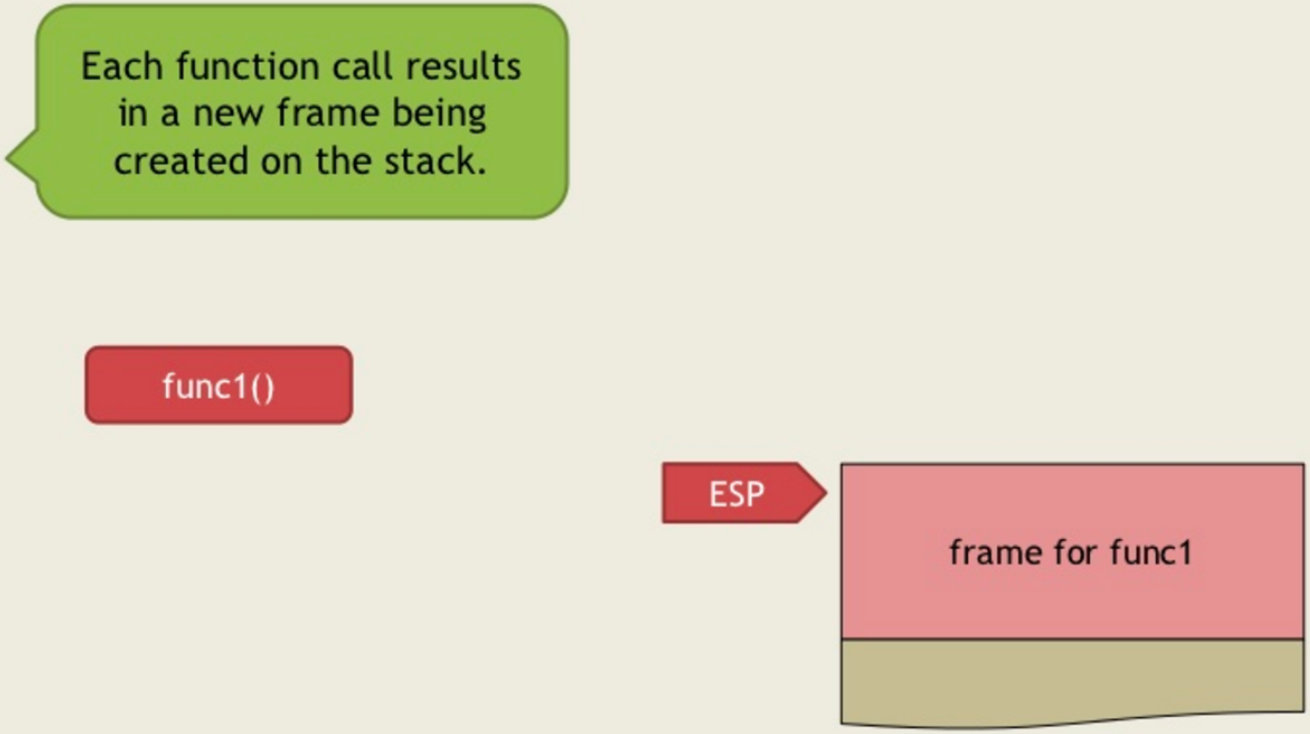


Stack Frame

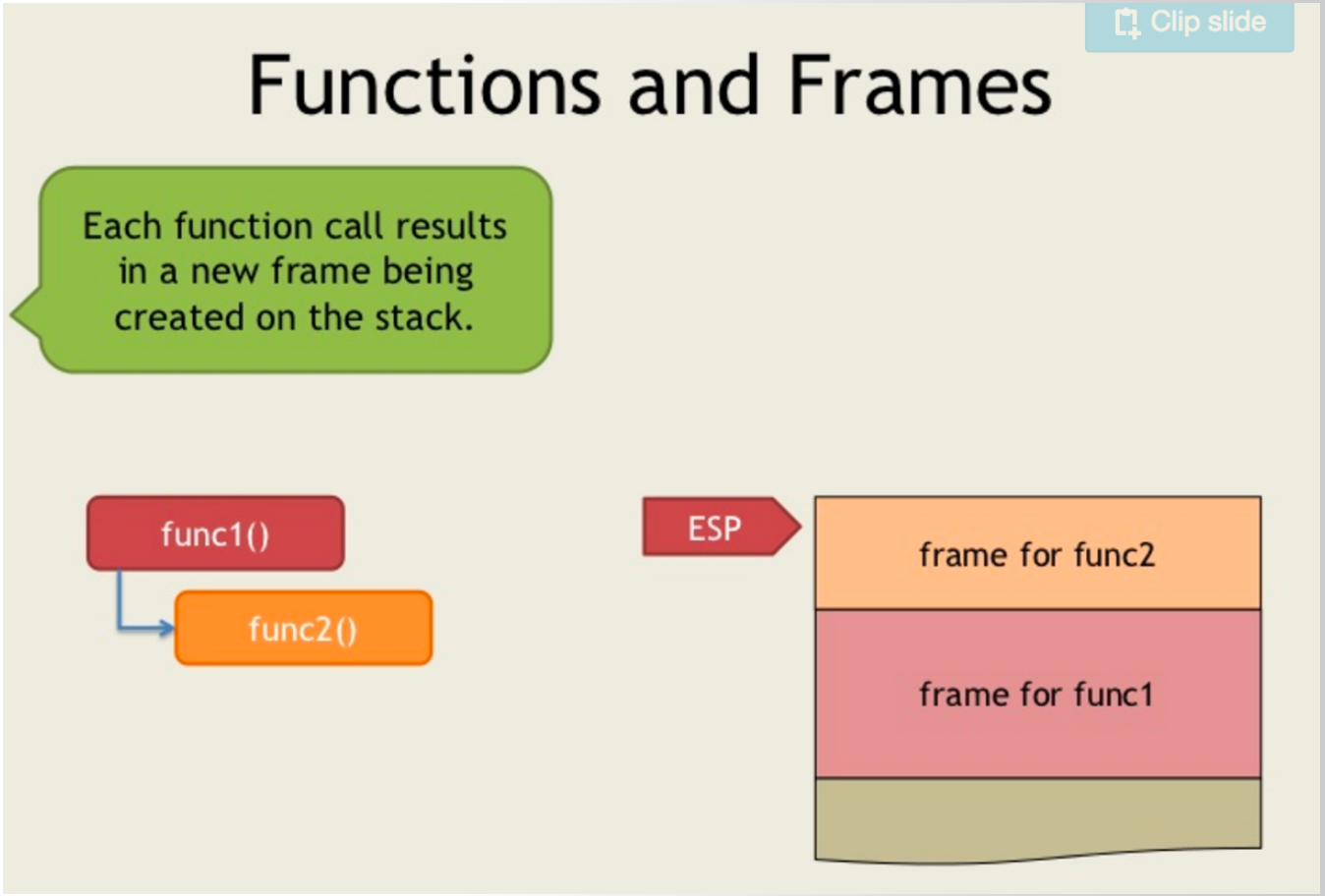
```
1  def hello(x):
2      if x == 1:
3          return "op"
4      else:
5          u = 1
6          e = 12
7          s = hello(x - 1)
8          e += 1
9          print(s)
10         print(x)
11         u += 1
12     return e
13
14
15 hello(3)
```



Functions and Frames

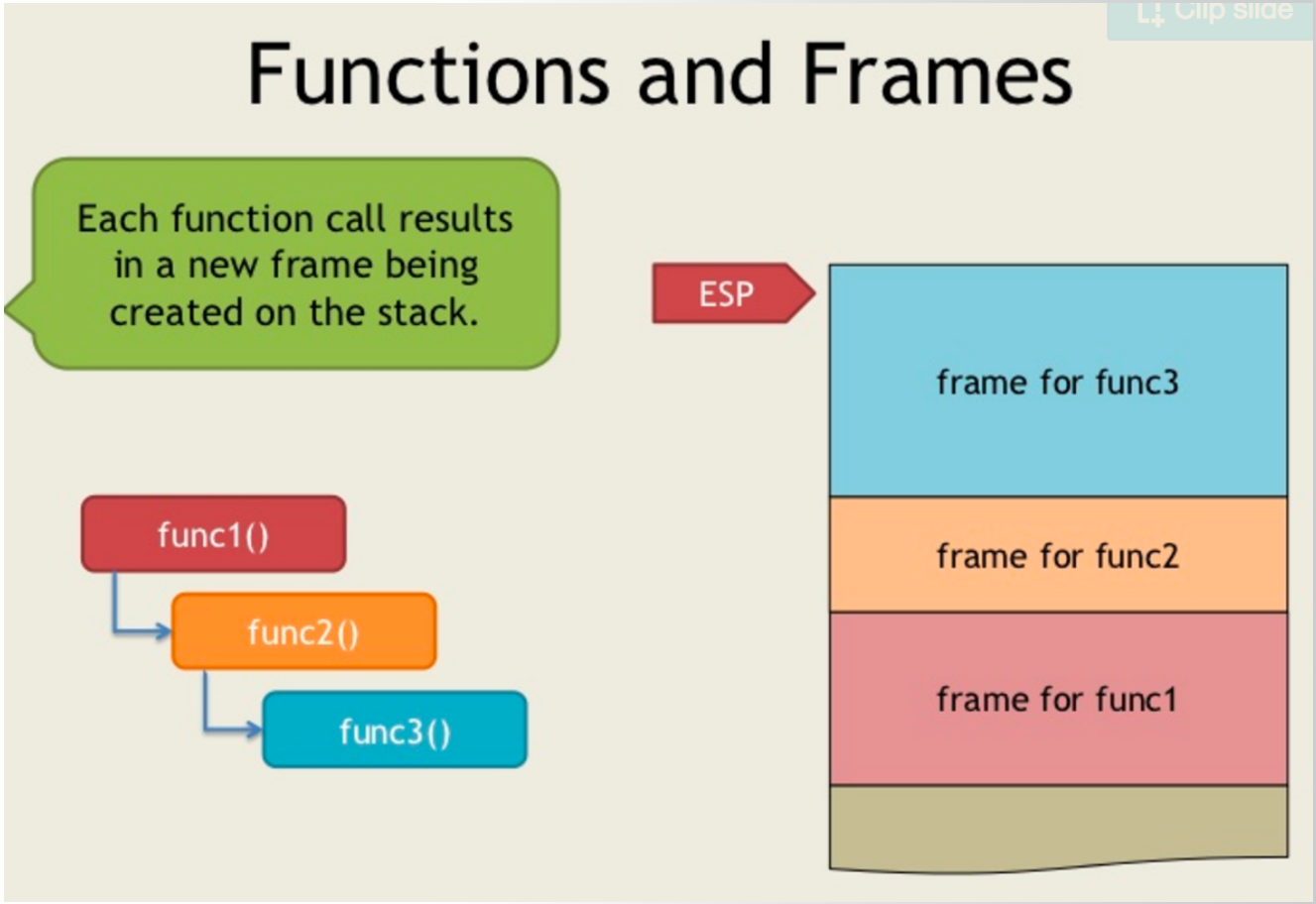


https://www.slideshare.net/saumilshah/how-functions-work-7776073



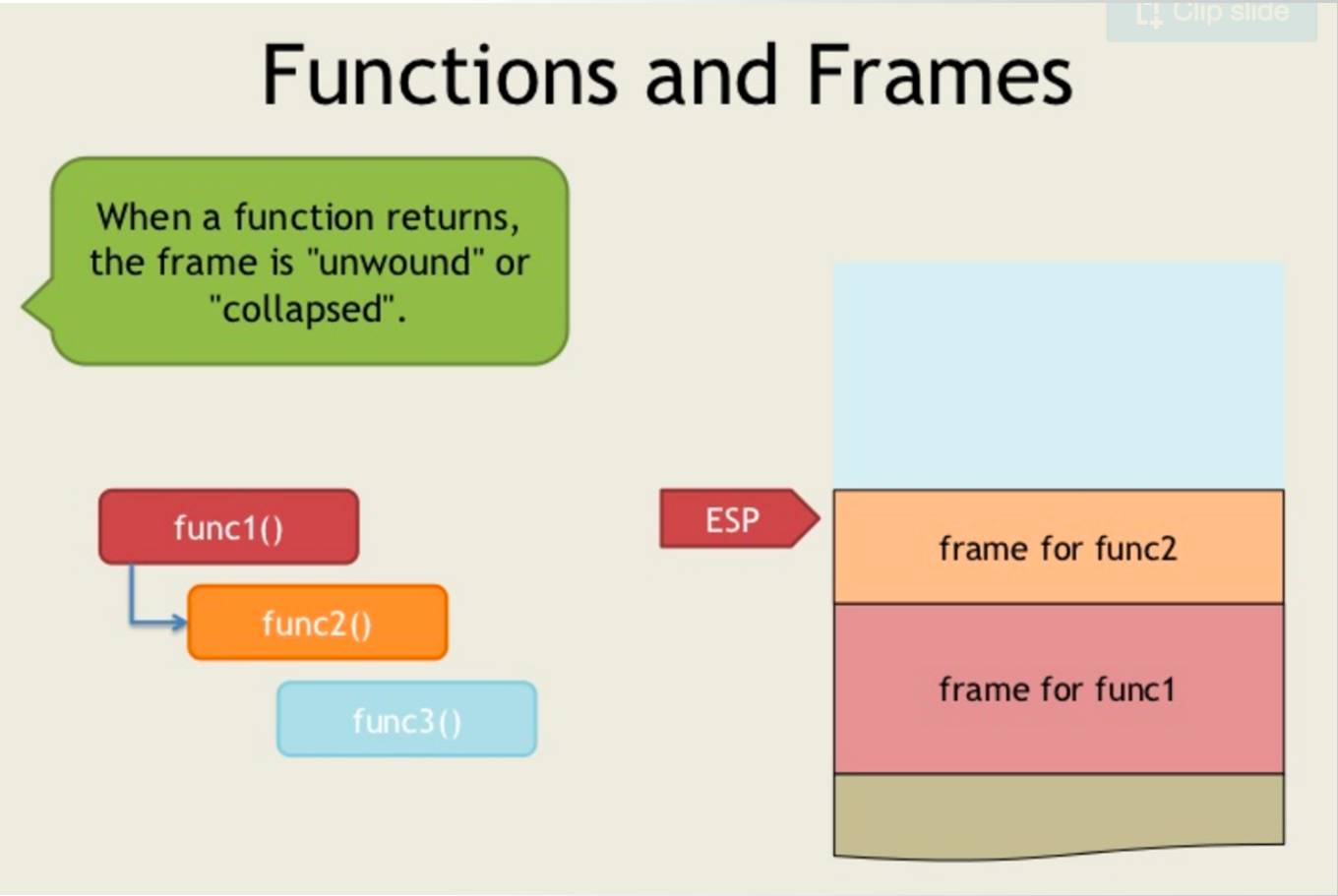
https://www.slideshare.net/saumilshah/how-functions-work-7776073

Functions and Frames



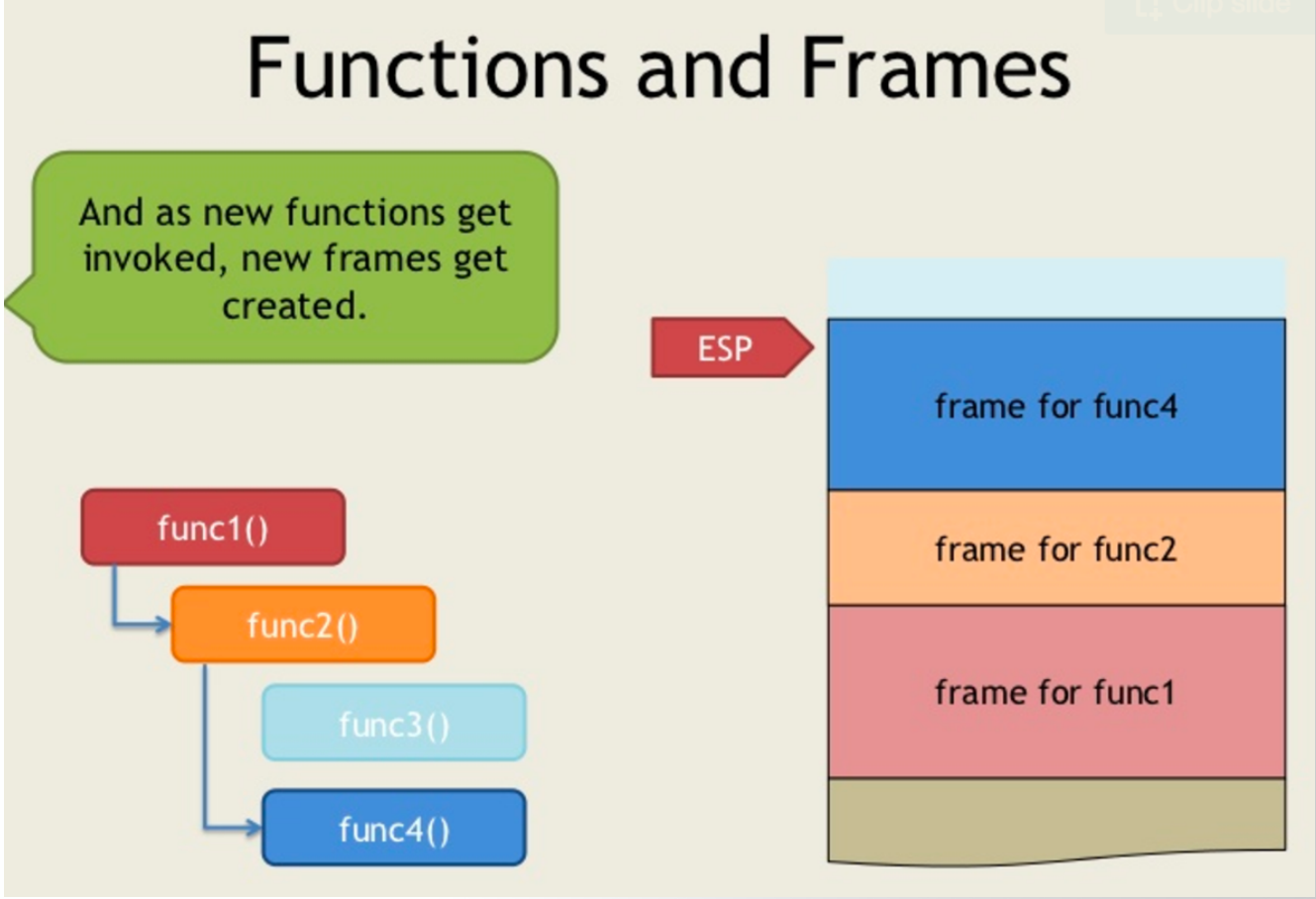
https://www.slideshare.net/saumilshah/how-functions-work-7776073

Functions and Frames



https://www.slideshare.net/saumilshah/how-functions-work-7776073

Functions and Frames



https://www.slideshare.net/saumilshah/how-functions-work-7776073

Stack Frame

File Edit View Terminal Tabs Help

```
PUSH EBP          ; start of the func (save current EBP to stack)
MOV EBP, ESP      ; save current ESP to EBP

....              ; function body
                  ; no matter how ESP changes, the EBP remains unchanged


MOV ESP, EBP      ; move the saved function start addr back to ESP
POP EBP           ; before return the func, pop the stored EBP
RETN              ; end of the func

~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                    12,1          All
```

StackFrame.c

```
1 StackFrame.c +
  1 #include "stdio.h"
  2
  3 long add(long a, long b)
  4 {
  5     long x = a, y = b;
  6     return (x + y);
  7 }
  8
  9 int main(int argc, char* argv[])
 10 {
 11     long a = 1, b = 2;
 12     printf("%d\n", add(a,b));
 13     return 0;
 14 }
 15
```

Q & A