

# Outline

---

- ▶ **Review: Memory Layout and Function Call Convention**
- ▶ **Buffer Overflow Vulnerability**

# Outline

---

- ▶ **Review: Memory Layout and Function Call Convention**
- ▶ Buffer Overflow Vulnerability

# Memory Layout of a Program (x86)

## Code

- ▶ The program code: fixed size and read only

## Static data

- ▶ Statically allocated data, e.g., variables, constants

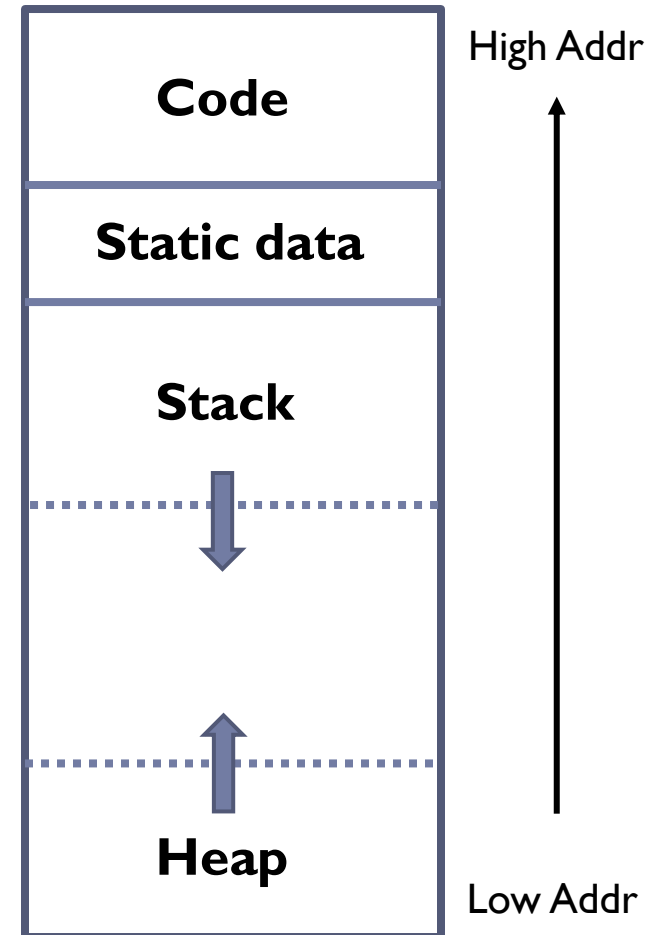
## Stack

- ▶ Parameters and local variables of methods as they are invoked.
- ▶ Each invocation of a method creates one frame which is pushed onto the stack
- ▶ Grows to lower addresses

## Heap

- ▶ Dynamically allocated data, e.g., class instances, data array
- ▶ Grows towards higher addresses

## Memory layout



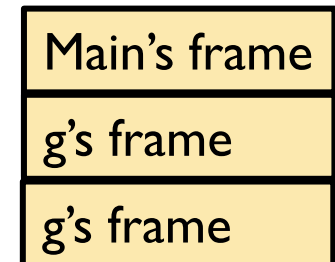
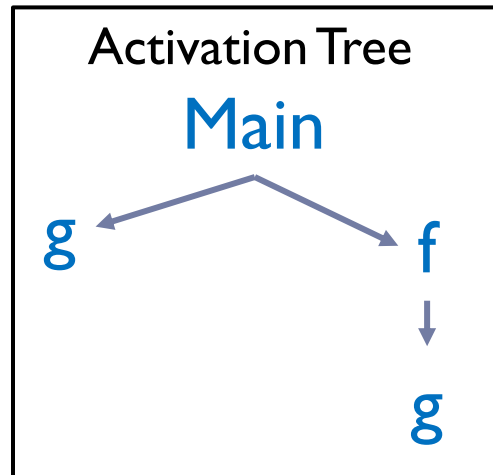
# Stack

Store local variables (including method parameters) and intermediate computation results

A stack is subdivided into multiple **frames**:

- ▶ A method is invoked: a new frame is pushed onto the stack to store local variables and intermediate results for this method;
- ▶ A method exits: its frame is popped off, exposing the frame of its caller beneath it

```
Main( ) {  
    g( );  
    f( );  
}  
f( ) {  
    return g( );  
}  
g( ) {  
    return 1;  
}
```



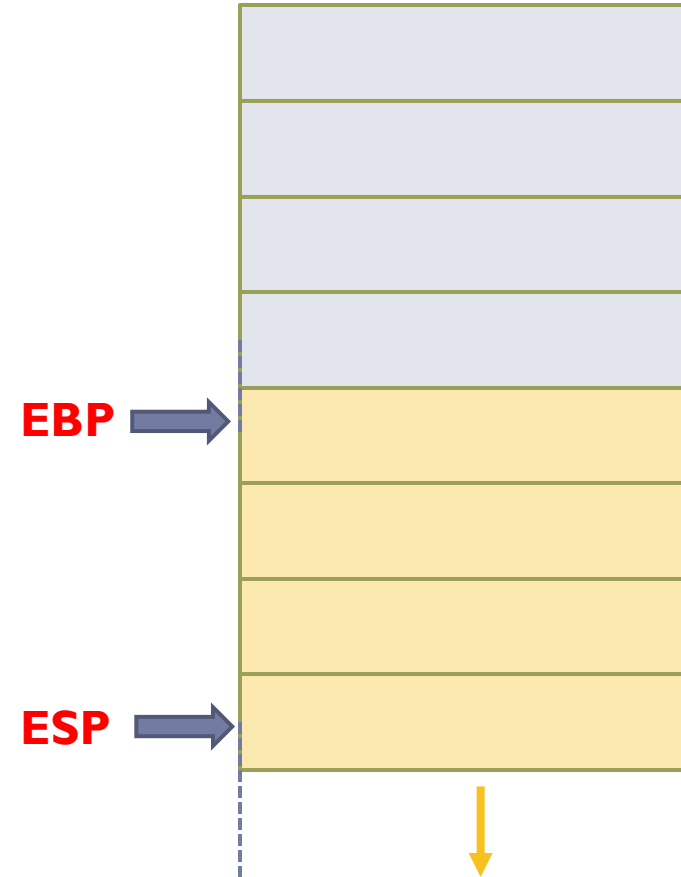
# Inside a Frame for One Function

## Two pointers:

- ▶ **EBP**: base pointer. Fixed at the frame base
- ▶ **ESP**: stack pointer. Current pointer in frame (current lowest value on the stack)

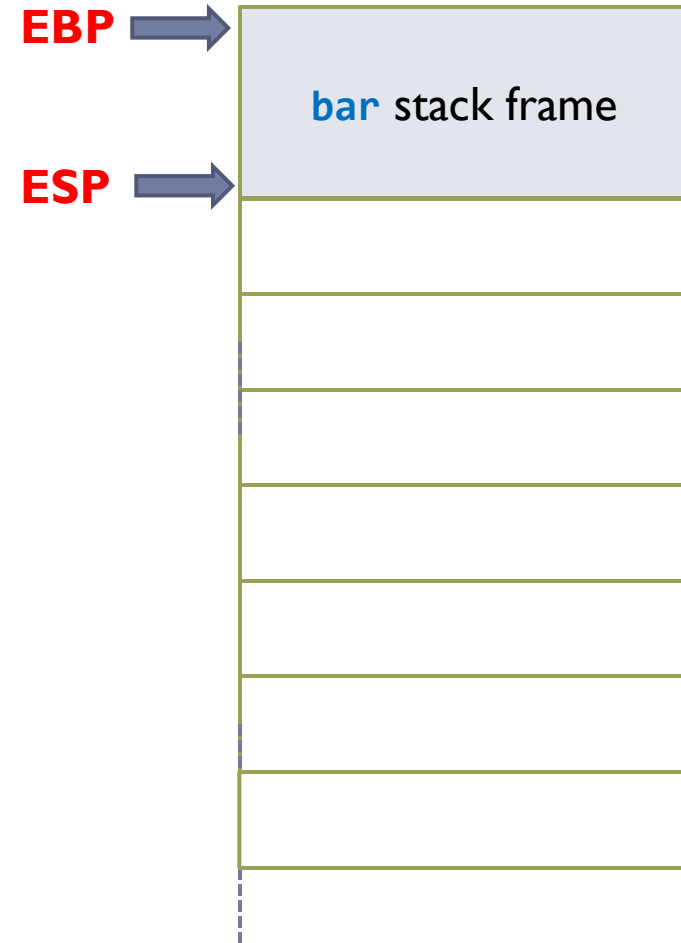
## A frame consists of the following parts:

- ▶ Function parameters
- ▶ Return address of the caller function
  - ▶ When the function is finished, execution continues at this return address
- ▶ Base pointer of the caller function
- ▶ Local variables
- ▶ Intermediate operands



# Function Call Convention

Initially: **EBP** and **ESP** point to the top and bottom of the bar stack frame.



```
void bar( ) {  
    foo(1, 2);  
}  
int foo(int x, int y){  
    int z = x + y;  
    return z;  
}
```

# Function Call Convention

## Step 1: Push function parameters to the stack.

- ▶ Function parameters are stored in reverse order.
- ▶ **ESP** is updated to denote the lowest stack location due to the push operation.

```
void bar( ) {  
    foo(1, 2);  
}  
int foo(int x, int y){  
    int z = x + y;  
    return z;  
}
```

