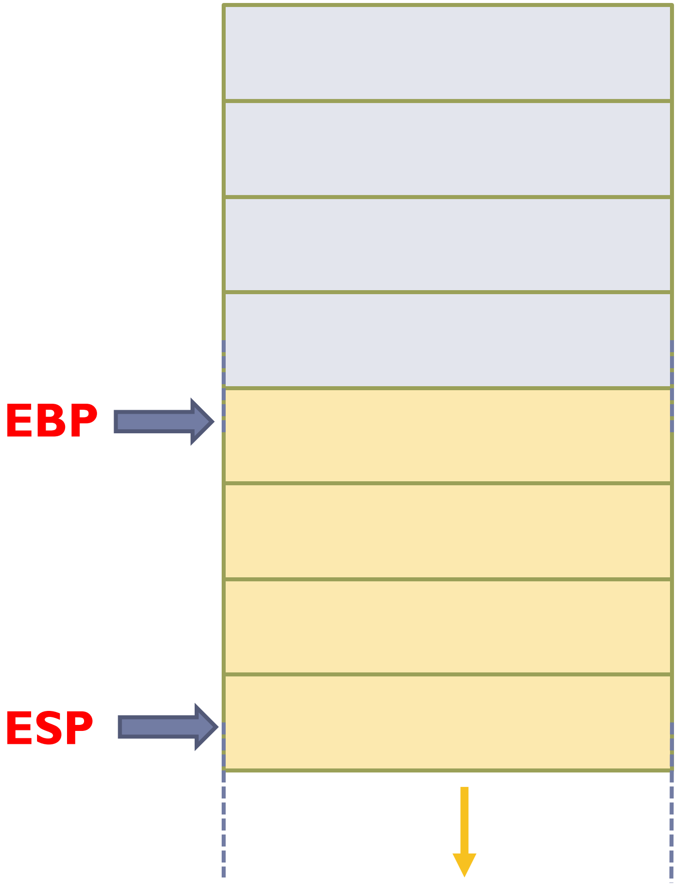
Inside a Frame for One Function

Two pointers:

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

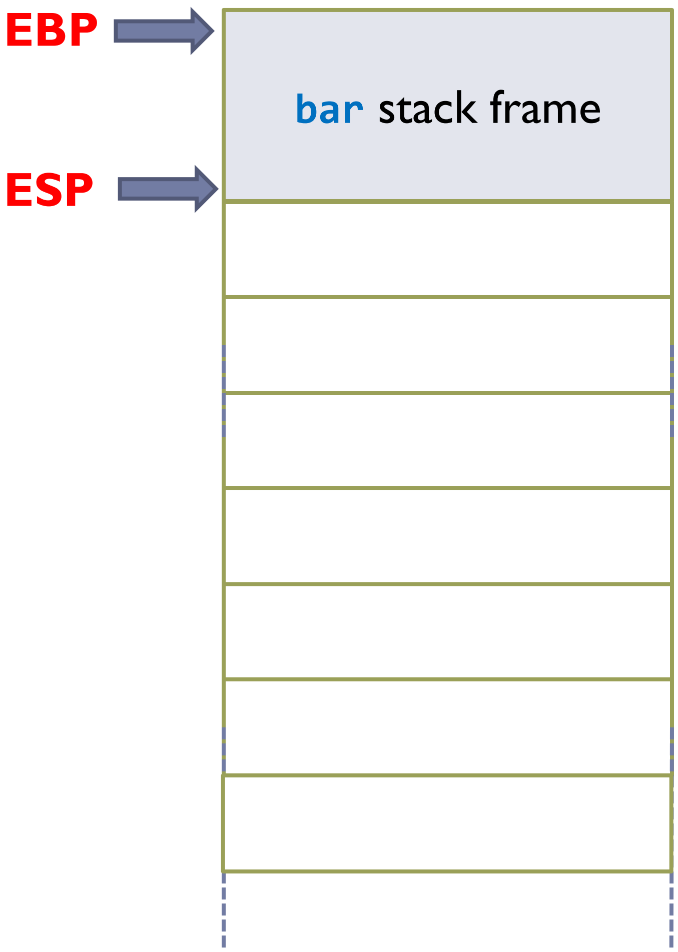
A frame consists of the following parts:

- 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.



EBP

ESP

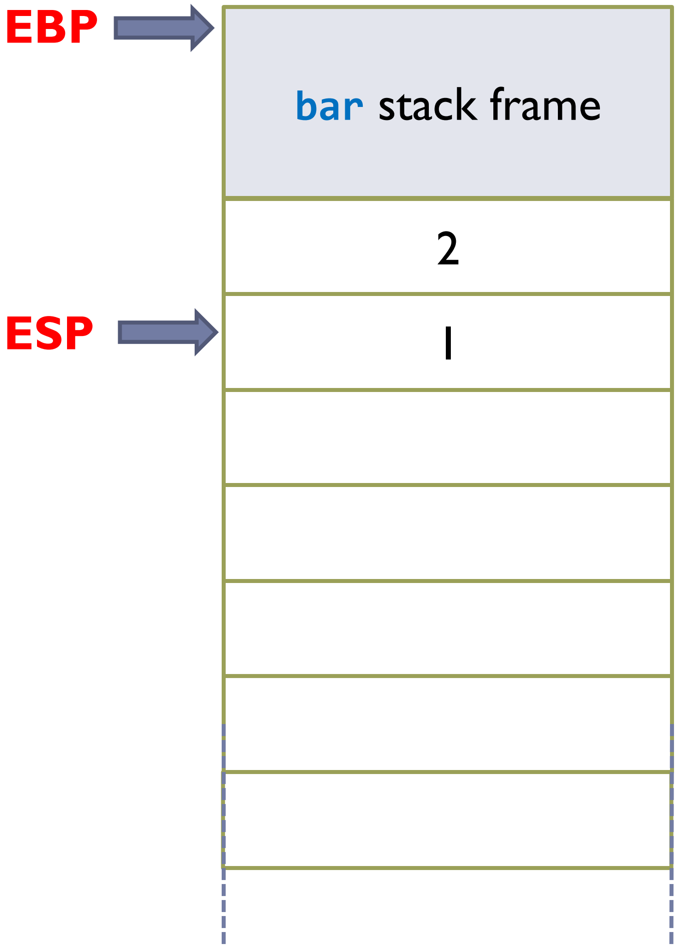
bar stack frame

```
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.

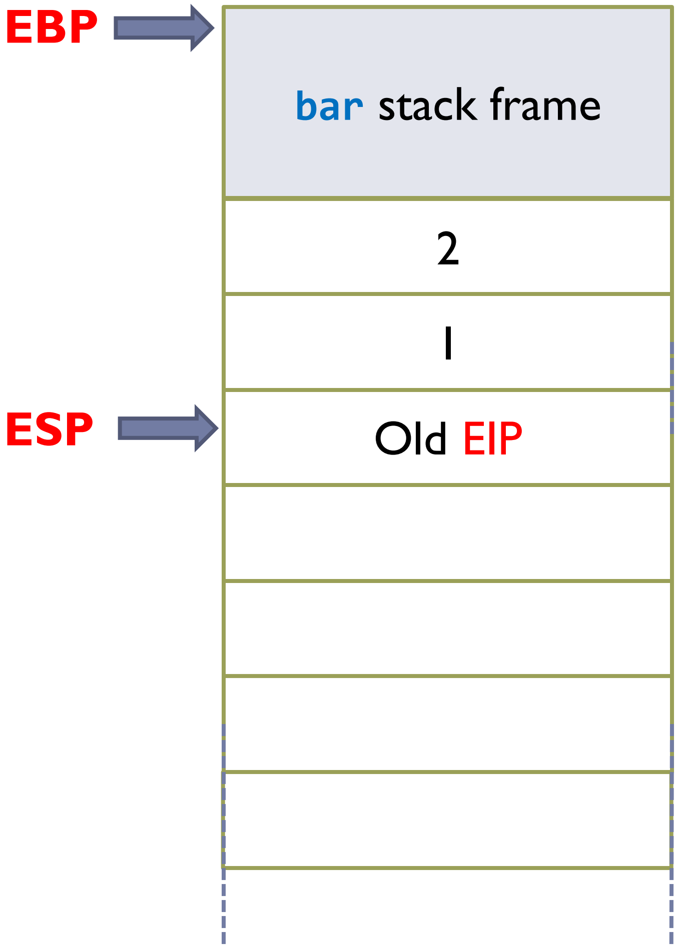


```
void bar( ) {
  foo(1, 2);
}
int foo(int x, int y){
  int z = x + y;
  return z;
}
```

Function Call Convention

Step 2: Push the current instruction pointer (EIP) to the stack.

- This is the return address in function bar after we finish function foo.
- ESP is updated to denote the lowest stack location due to the push operation.

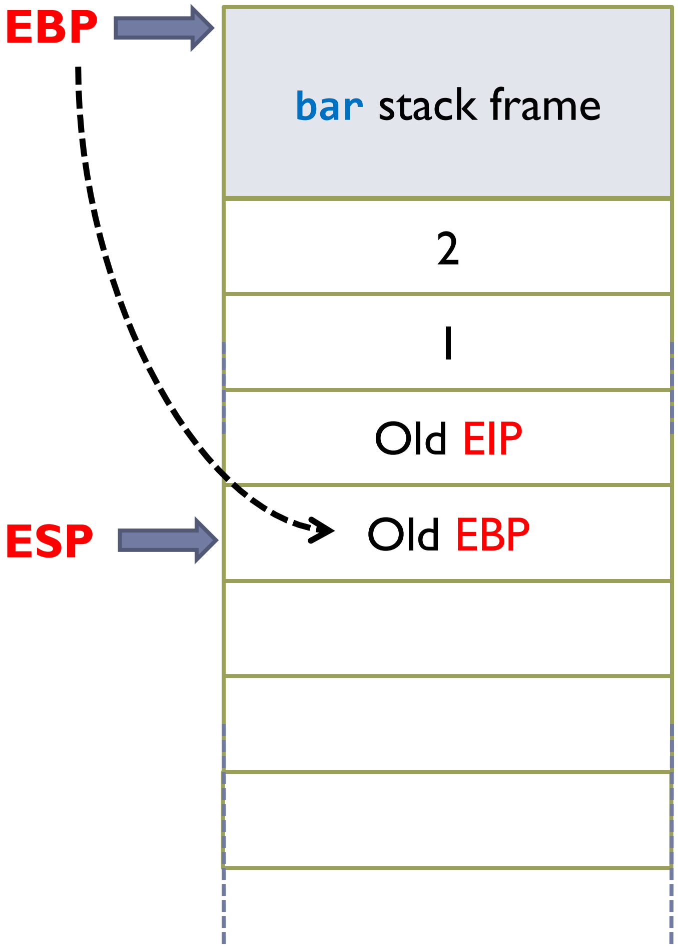


```
void bar( ) {
  foo(1, 2);
}
int foo(int x, int y){
  int z = x + y;
  return z;
}
```

Function Call Convention

Step 3: Push the EBP of function bar to the stack.

- This can help restore the top of function **bar** stack frame when we finish function **foo**.
- ESP is updated to denote the lowest stack location due to the push operation.



EBP

bar stack frame

2

1

Old EIP

ESP

Old EBP

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