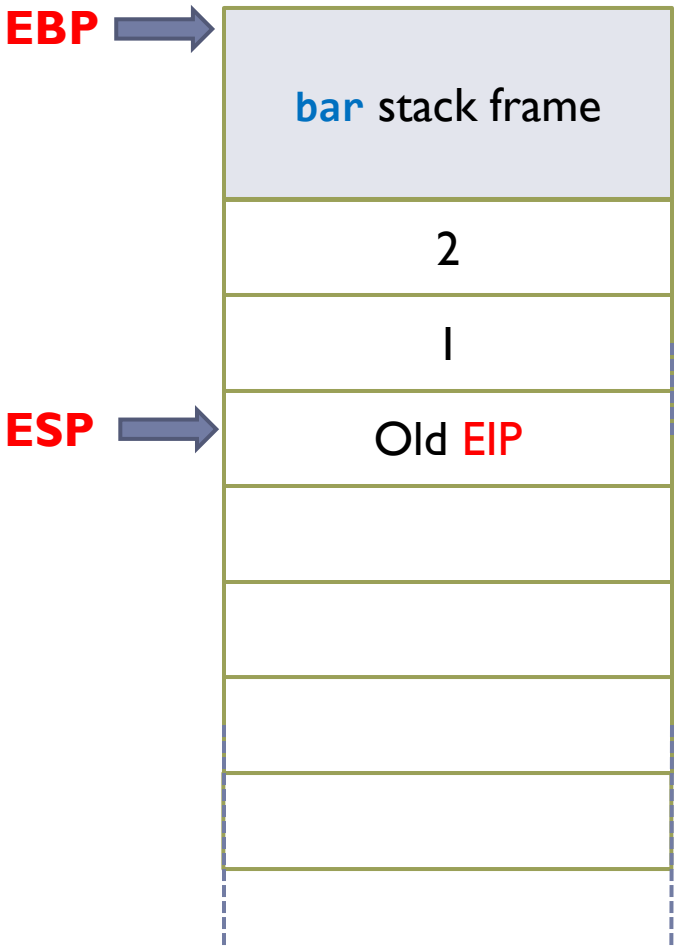
# Function Call Convention

## 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.

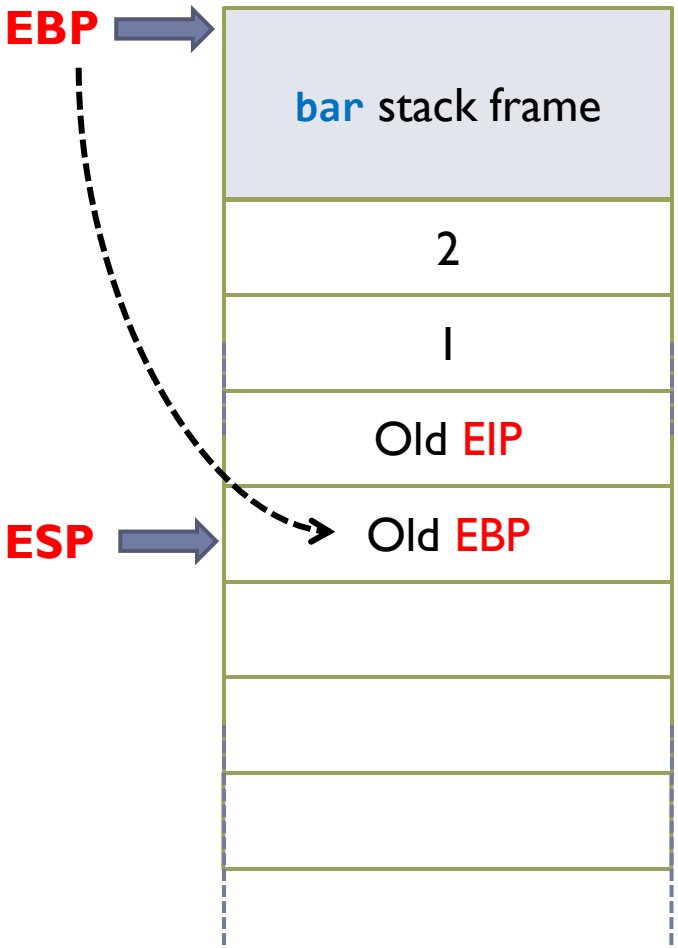



```
void bar( ) {
  foo(1, 2);
}
int foo(int x, int y){
  int z = x + y;
  return z;
}
```

# 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.

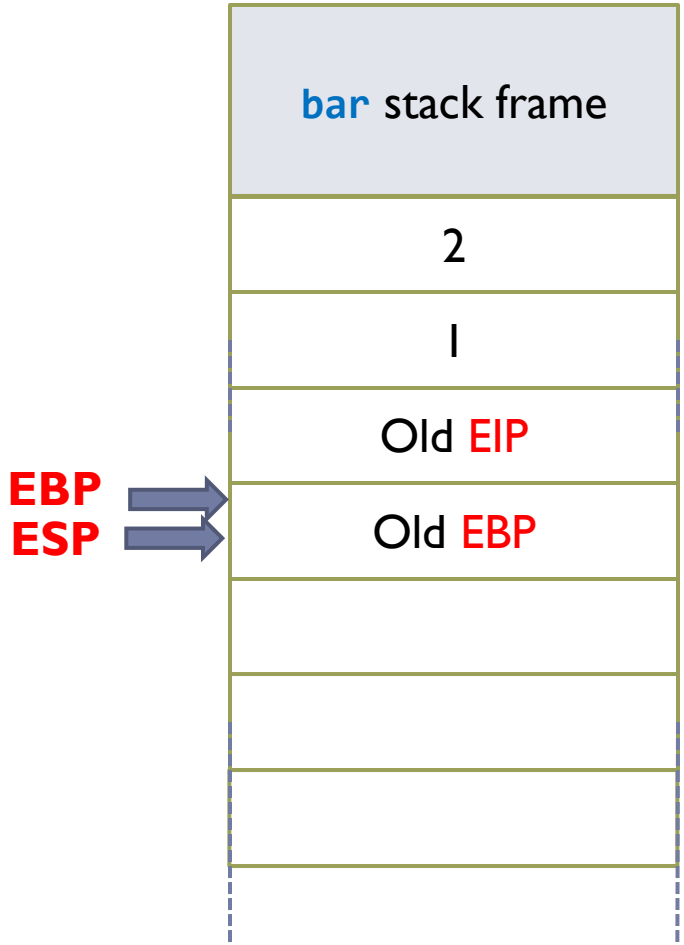



```
void bar( ) {
  foo(1, 2);
}
int foo(int x, int y){
  int z = x + y;
  return z;
}
```

# Function Call Convention

## Step 4: Adjust EBP for function foo stack frame.

- Move EBP to ESP of `bar` stack frame



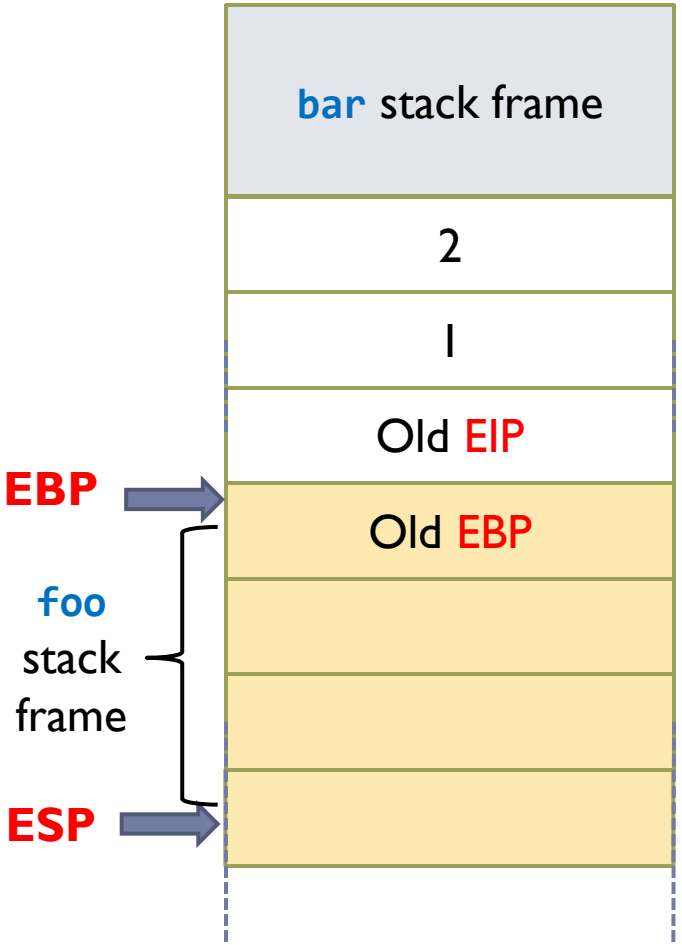

```
void bar( ) {
  foo(1, 2);
}
int foo(int x, int y){
  int z = x + y;
  return z;
}
```

# Function Call Convention

## Step 5: Adjust ESP for function foo stack frame.

- Move ESP to some location below to create a new stack frame for function **foo**
- The stack space for function **foo** is pre-calculated based on the source code. It is used for storing the local variables and intermediate results.

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



# Function Call Convention

## Step 6: Execute function foo within its stack frame.

- The returned result will be stored in the register EAX.

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