COEN 175

Lecture 14: Stack Frames

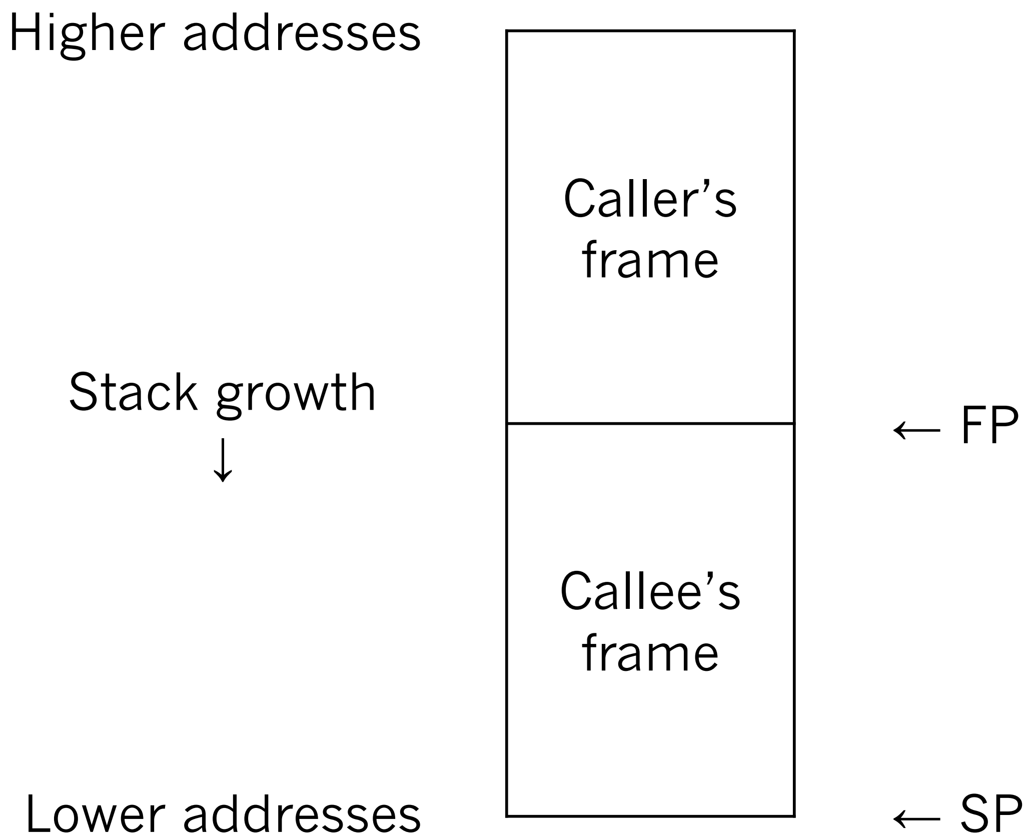
Stack Allocation

- Each function is responsible for allocating and deallocating memory for its stack-allocated data.
- Memory management is done per function, not per scope within a function.
- Stack-allocated data can include:
  - Local variables
  - Parameters
  - Intermediate results from computations
  - Saved program state such as registers

Terminology

- A function's block of memory on the stack is called an activation record or **stack frame**.
- The function making the function call is the **caller**, and the function being called is the **callee**.
- The processor reserves two registers for our use.
  - The **stack pointer** (SP) points to the top of the stack.
  - The **frame pointer** (FP) points to the start of the frame.
  - Since stacks typically grow down, the "top" of the stack is actually the lowest memory address.

Example: Stack Growth



Important Tasks

- Stack frame management
  - Who allocates the callee's stack frame?
  - Who deallocates the callee's stack frame?
- Transfer of control
  - How is control transferred to the callee?
  - How is control transferred back to the caller?
- Parameter passing
  - How are parameters passed to the callee?
  - How does the callee return a value to the caller?

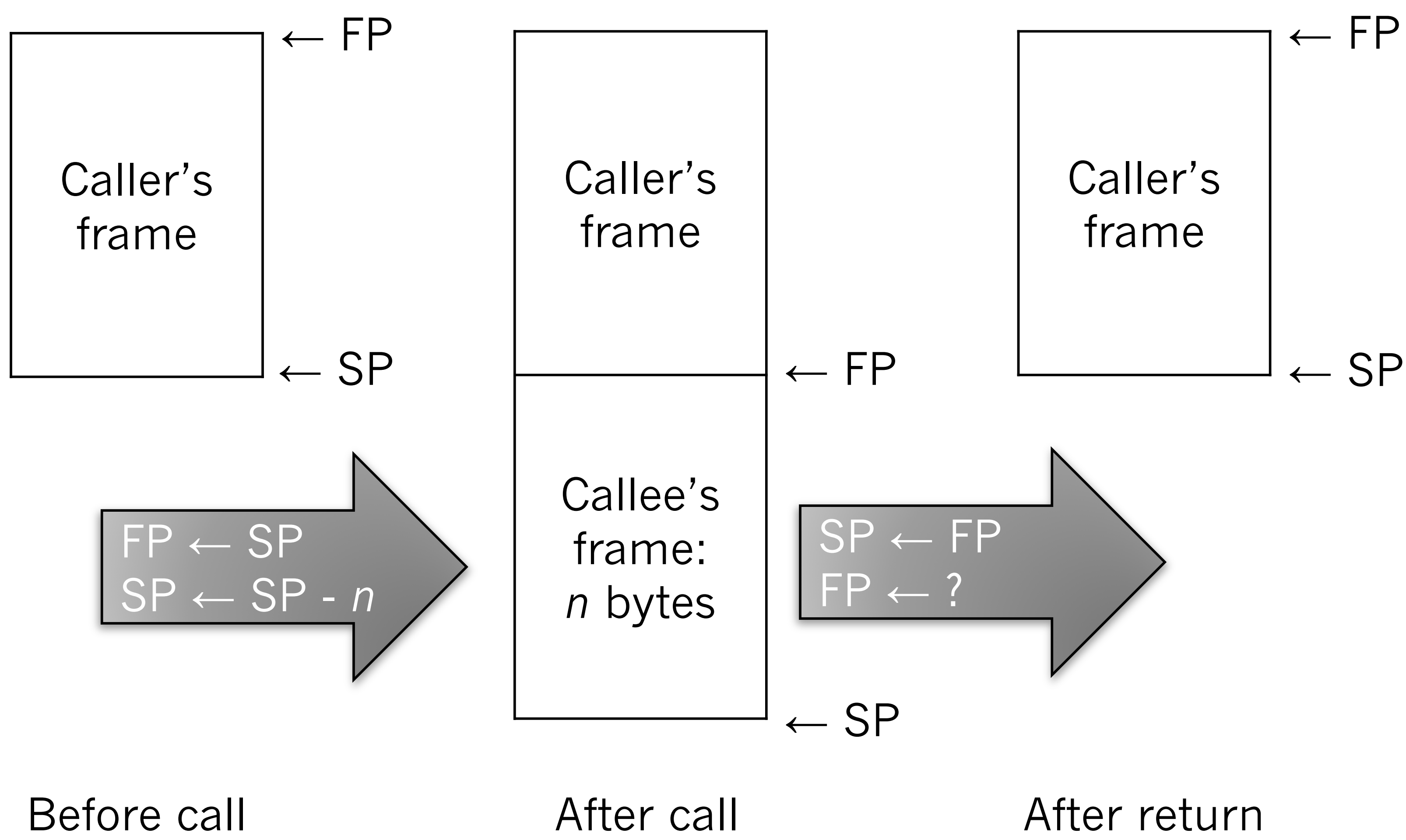
Frame Management

- Who allocates the callee's stack frame?
  - The callee does: it is the only one who knows how much space it requires.
- Who deallocates the callee's stack frame?
  - The callee does: only it knows how much space it has allocated.
- The callee does not know the identity of the caller.
  - Separate compilation of source files means that the source code of the caller is not available.

Assembly Code Layout

- A function begins execution by allocating its frame.
  - This piece of assembly code is called the **prologue**.
  - Thus, a call to the function does not in fact immediately begin execution of the function body.
- Next comes the body of the function.
  - This is the code that the programmer actually wrote.
- A function ends execution by deallocating its frame.
  - This piece of assembly code is called the **epilogue**.
  - Thus, a return from a function is actually a jump to the epilogue of the function.

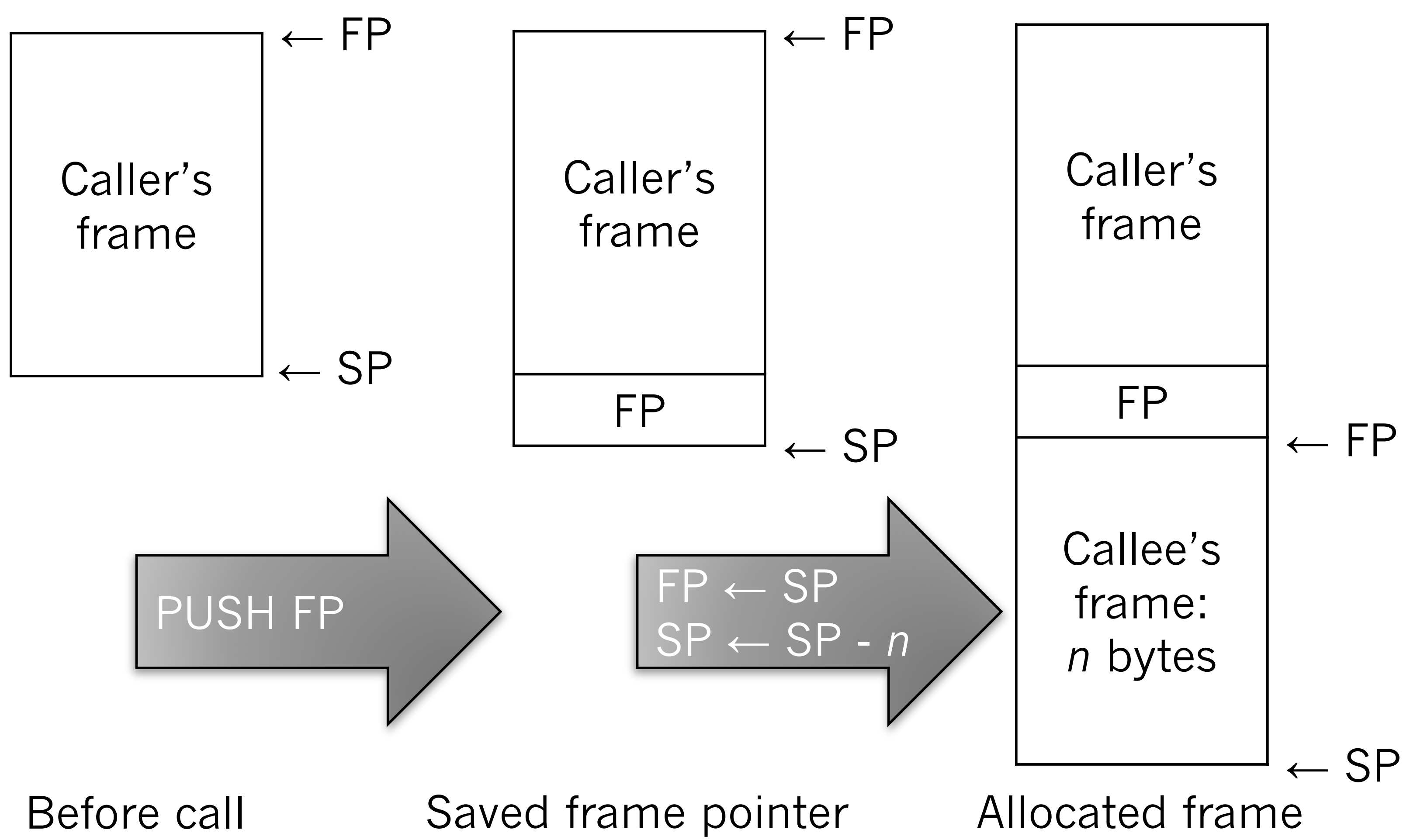
Example: Call and Return



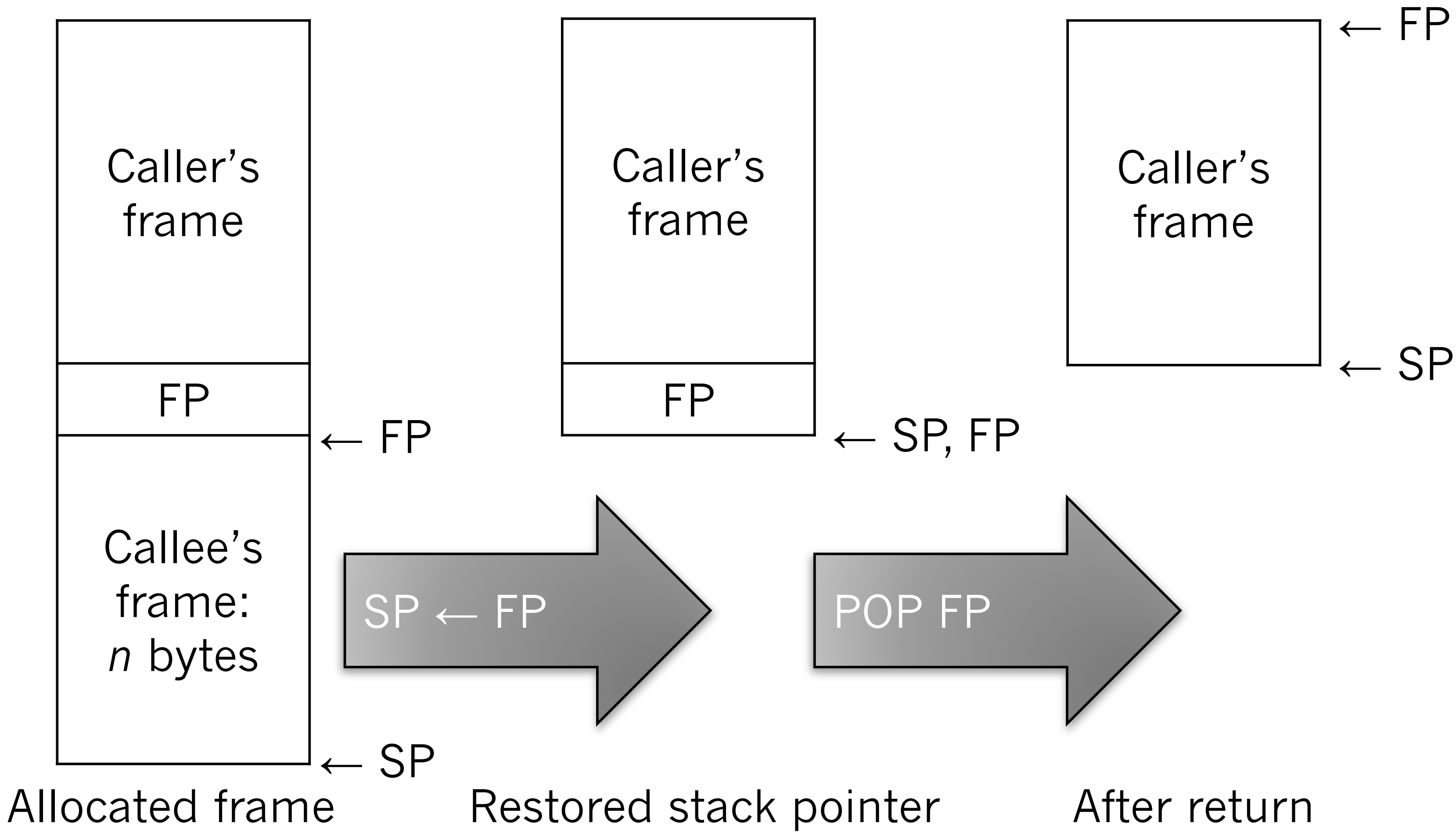
Saving the Frame Pointer

- Deallocating the stack frame requires us to restore the old frame pointer, which we have lost.
  - The solution is to first save the old frame pointer.
  - Where do we save it? On the stack, of course.
  - Note that we cannot use static memory because our function might be recursive.
- The prologue will first push the frame pointer.
- The epilogue will pop the old value before returning.

Revised: Call and Return



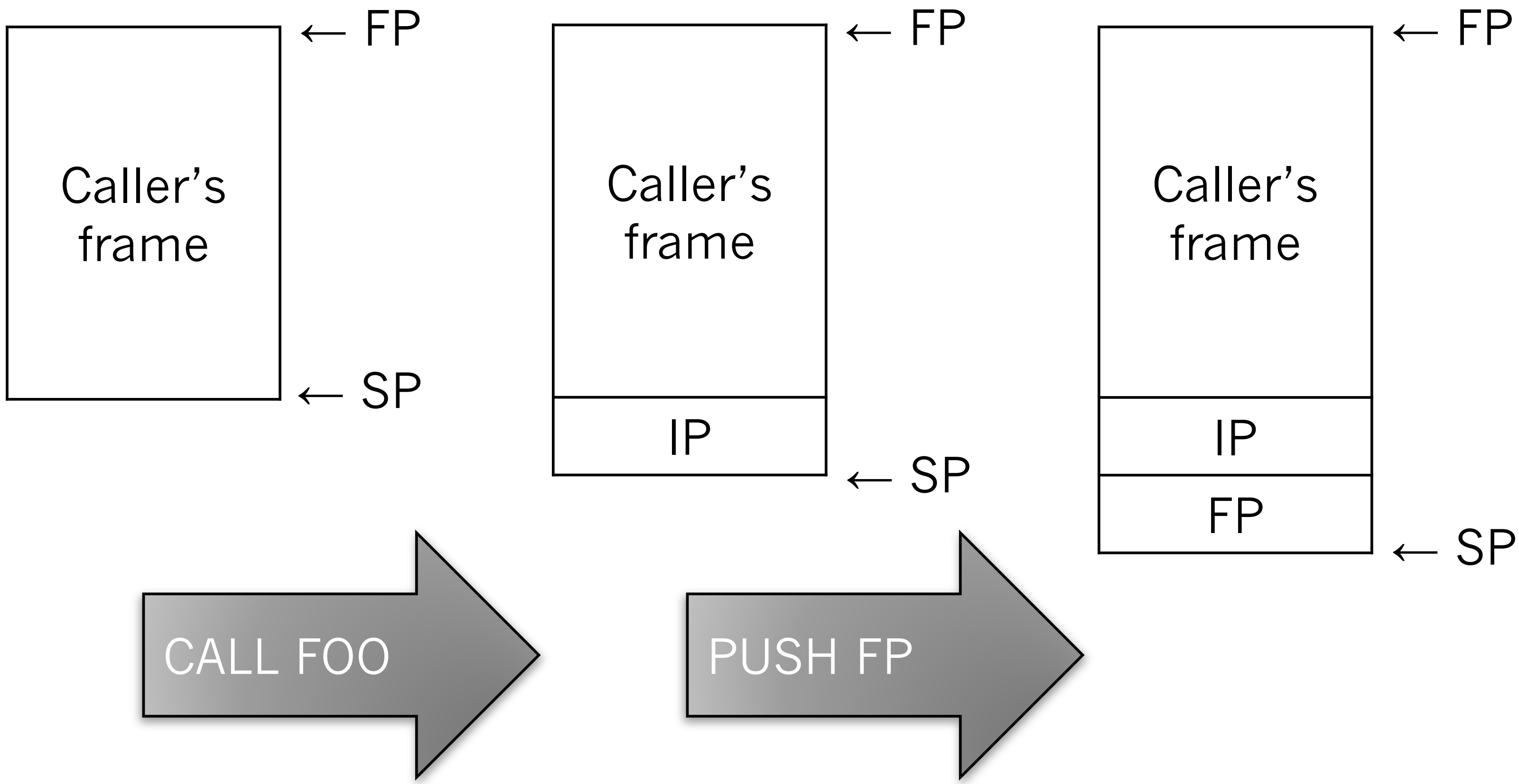
Revised: Call and Return



Transfer of Control

- How is control transferred to the callee?
  - The address of the current instruction is stored in a special register called the program counter or **instruction pointer**.
  - The caller uses the `call` instruction to change the value of the instruction pointer (IP).
  - The `call` instruction must first save the current value of the instruction pointer on the stack.
- How is control transferred back to the caller?
  - The callee uses the `return` instruction to pop the instruction pointer and return control.

Reality: Call and Return

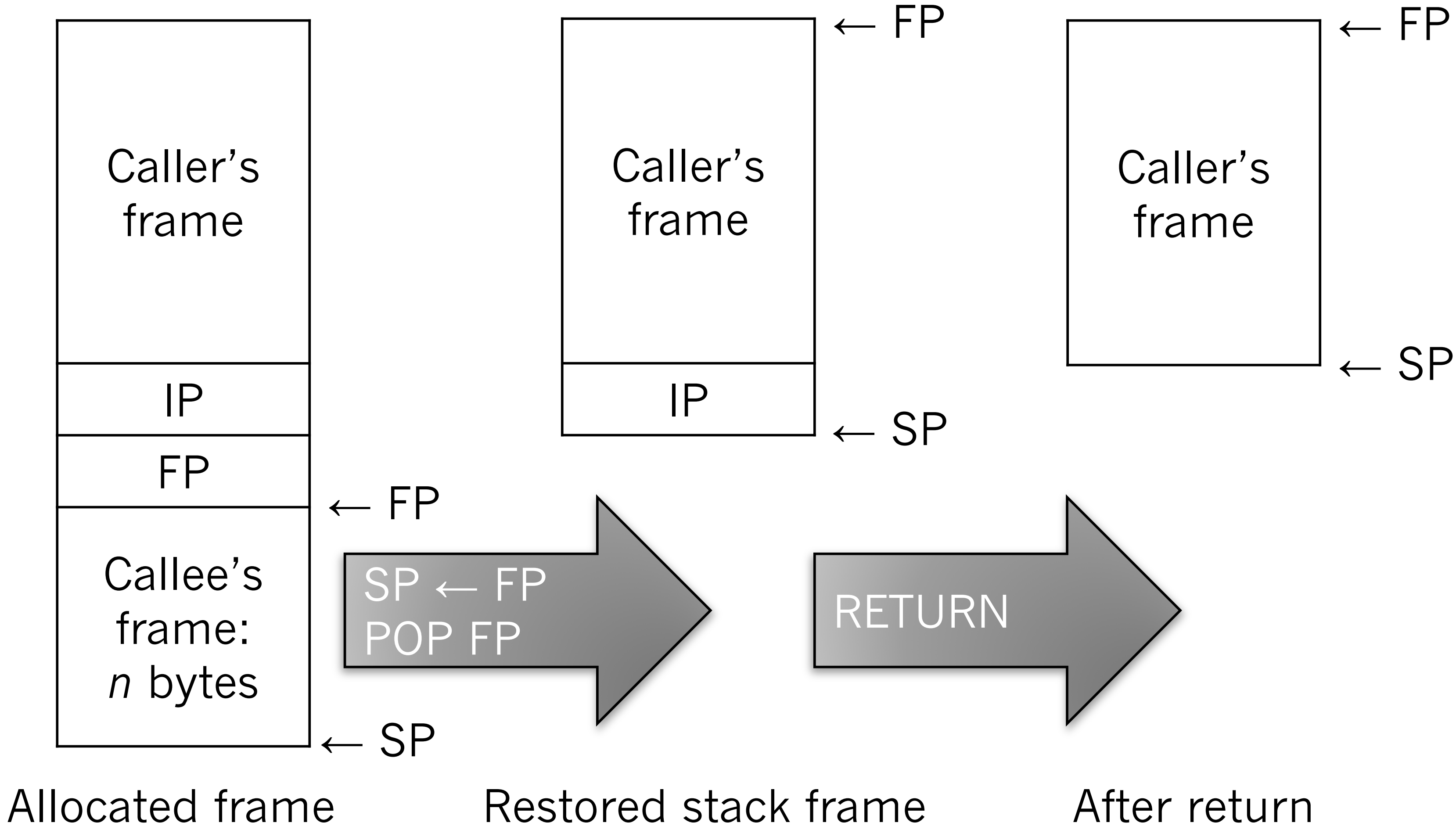


Before call

Saved instruction pointer

Saved frame pointer

Reality: Call and Return



Assembler Syntax

- Registers are prefixed with a percent sign.
- Immediate values are prefixed with a dollar sign.
- Opcodes are suffixed to indicate operand size:
  - `b` – byte (8-bit) operand
  - `w` – word (16-bit) operand
  - `l` – long (32-bit) operand
  - `q` – quad (64-bit) operand
- Base offset addressing is denoted by *offset*(*register*).
- The destination operand is on the right.

Intel Registers

- The Intel 32-bit architecture has 8 registers.
  - The stack pointer is called `%esp`.
  - The frame or **base pointer** is called `%ebp`.
- In the original 16-bit architecture the stack pointer was simply `%sp`.
- The 32-bit register is called `%esp` (for "extended") and `%sp` is simply the lower 16-bits.
- On the 64-bit architecture, the 64-bit register is called `%rsp`, with `%esp` being the lower 32-bits of it.

Intel Assembly

- Standard prologue on 32-bit Intel architecture, where *n* is the number of bytes required:

```
pushl  %ebp
movl   %esp, %ebp
subl   $n, %esp
```

- Standard epilogue on 32-bit Intel architecture:

```
movl   %ebp, %esp
popl   %ebp
ret
```