Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:

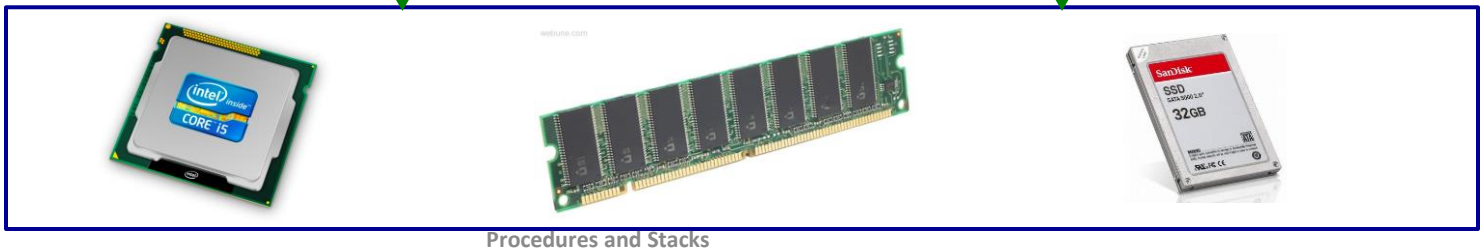
```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
11000001111110100011111
```

OS:

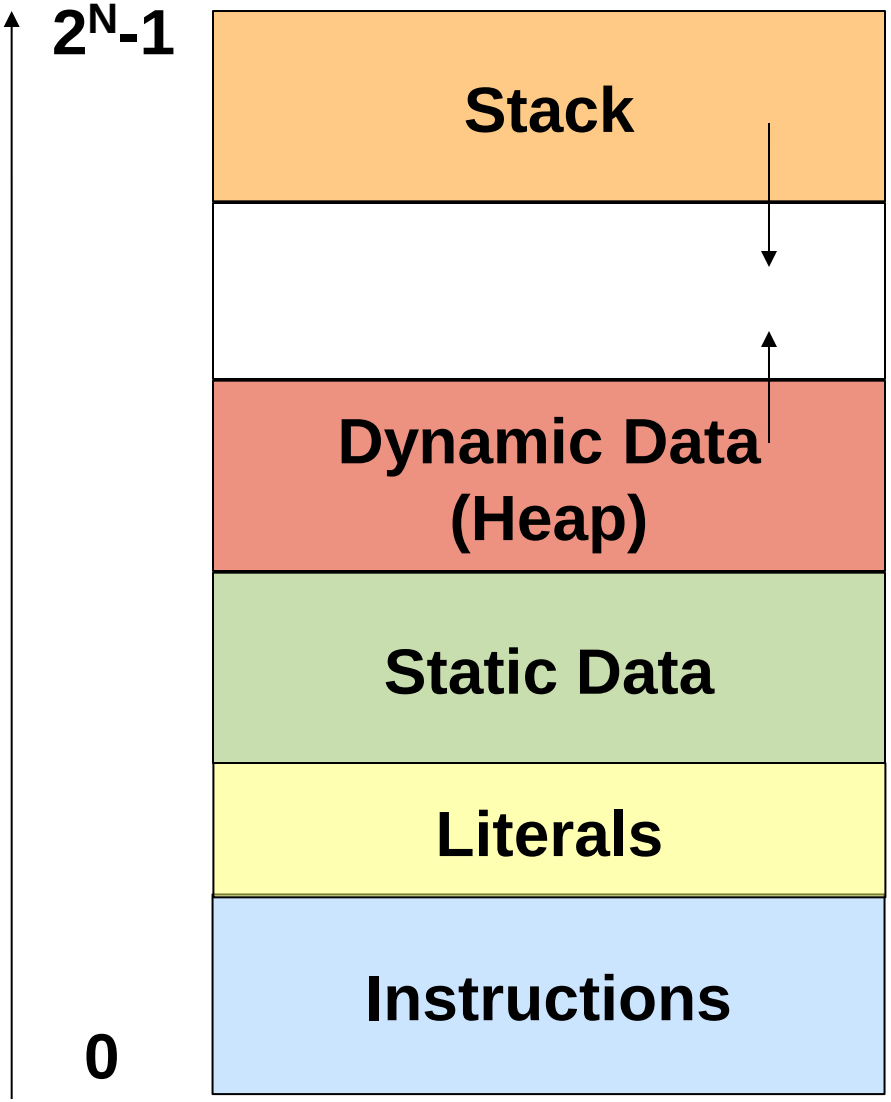
Computer system:



Section 5: Procedures & Stacks

- Stacks in memory and stack operations
- The stack used to keep track of procedure calls
- Return addresses and return values
- Stack-based languages
- The Linux stack frame
- Passing arguments on the stack
- Allocating local variables on the stack
- Register-saving conventions
- Procedures and stacks on x64 architecture

Memory Layout



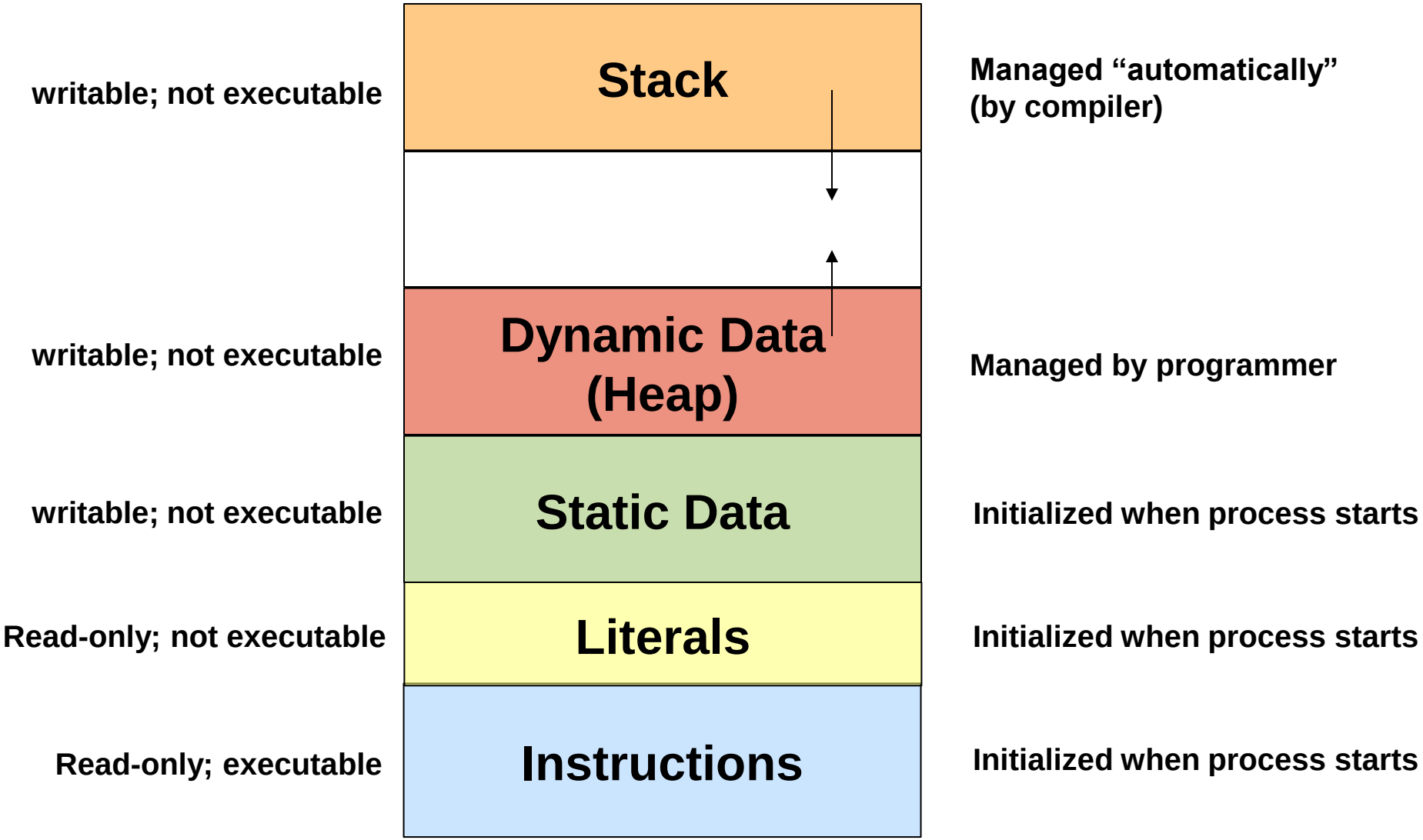
local variables; procedure context

variables allocated with *new* or *malloc*

static variables (including global variables (C))

literals (e.g., “example”)

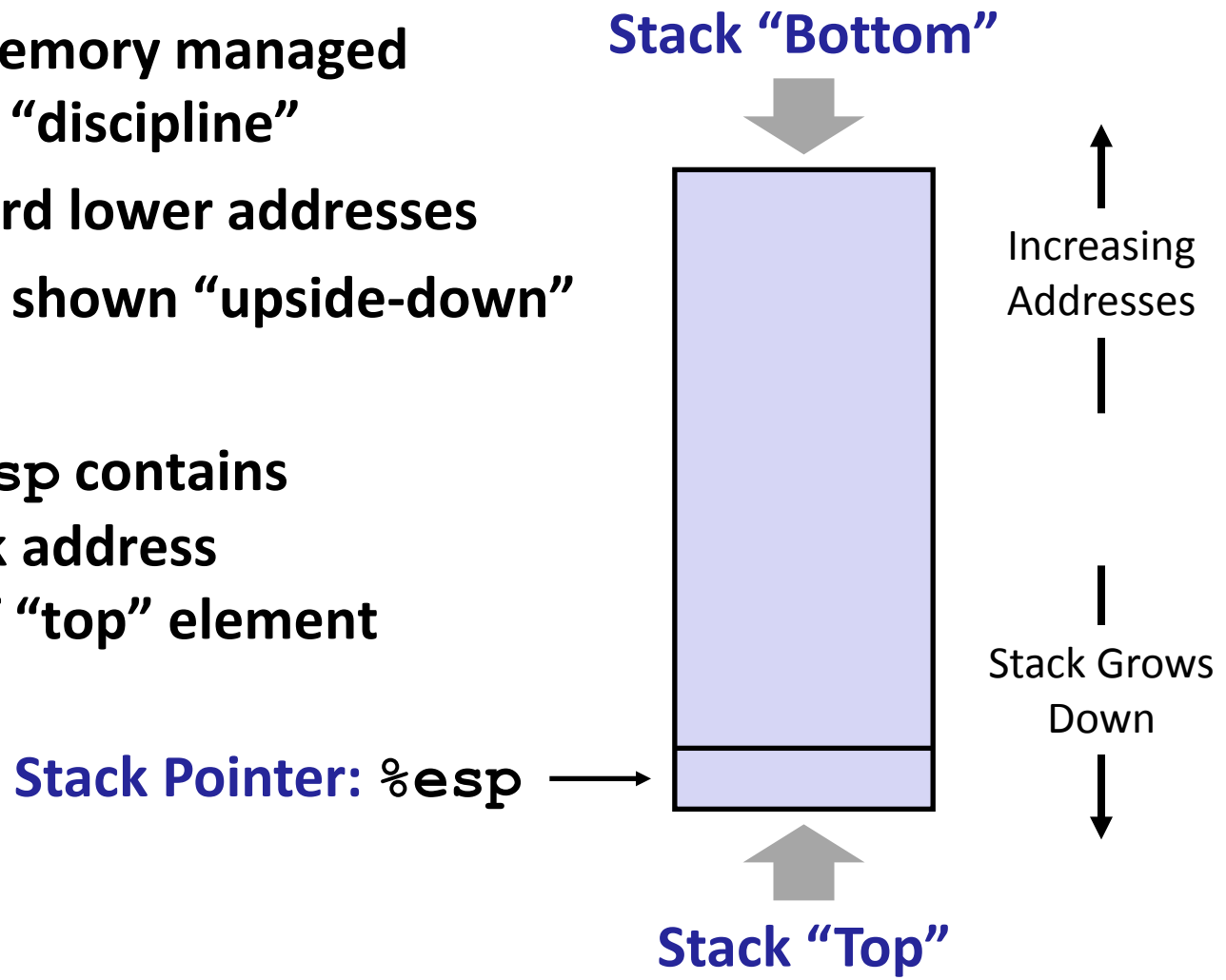
Memory Layout



IA32 Call Stack

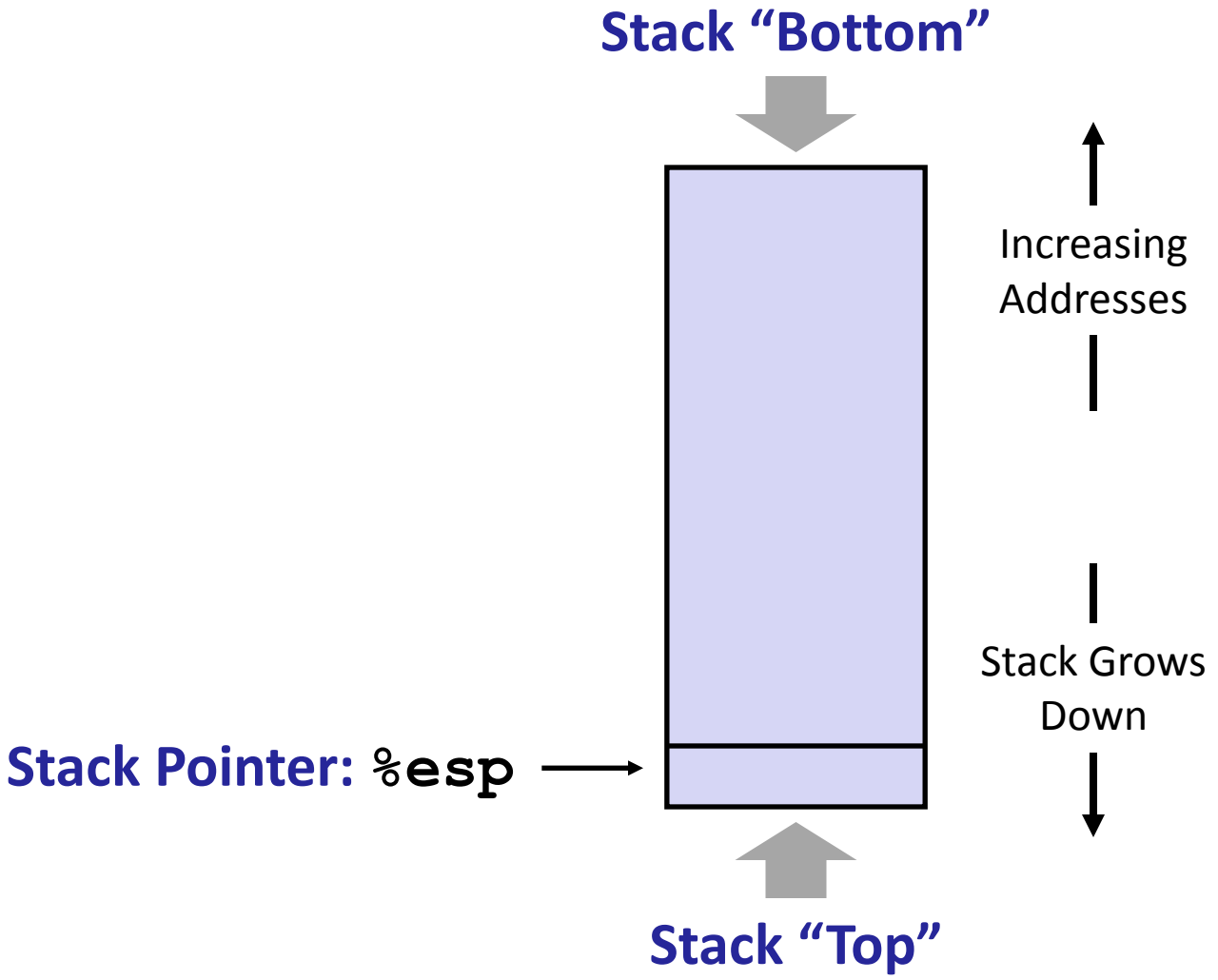
- Region of memory managed with a stack "discipline"
- Grows toward lower addresses
- Customarily shown "upside-down"

- Register `%esp` contains lowest stack address = address of "top" element



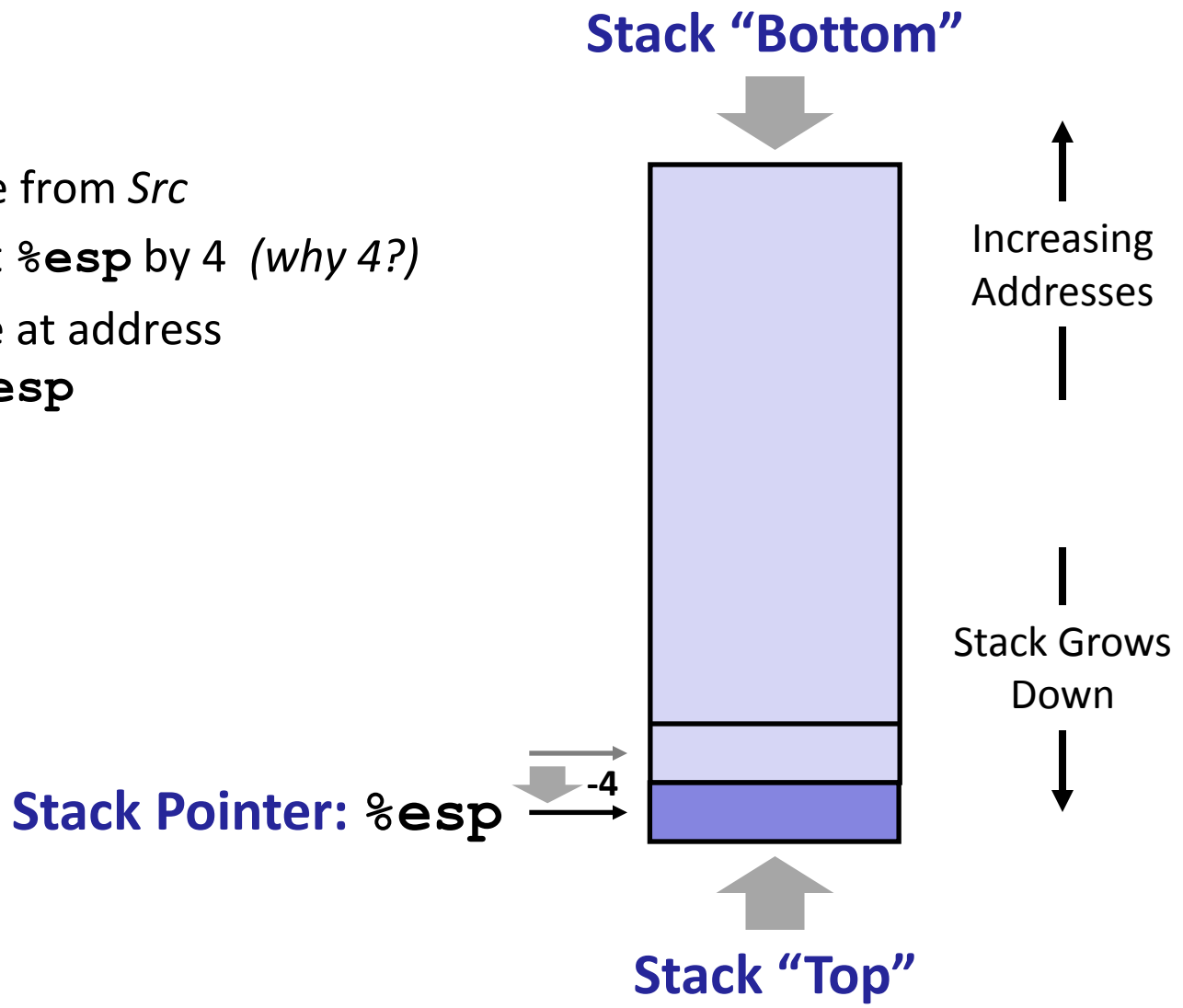
IA32 Call Stack: Push

- `pushl` *Src*



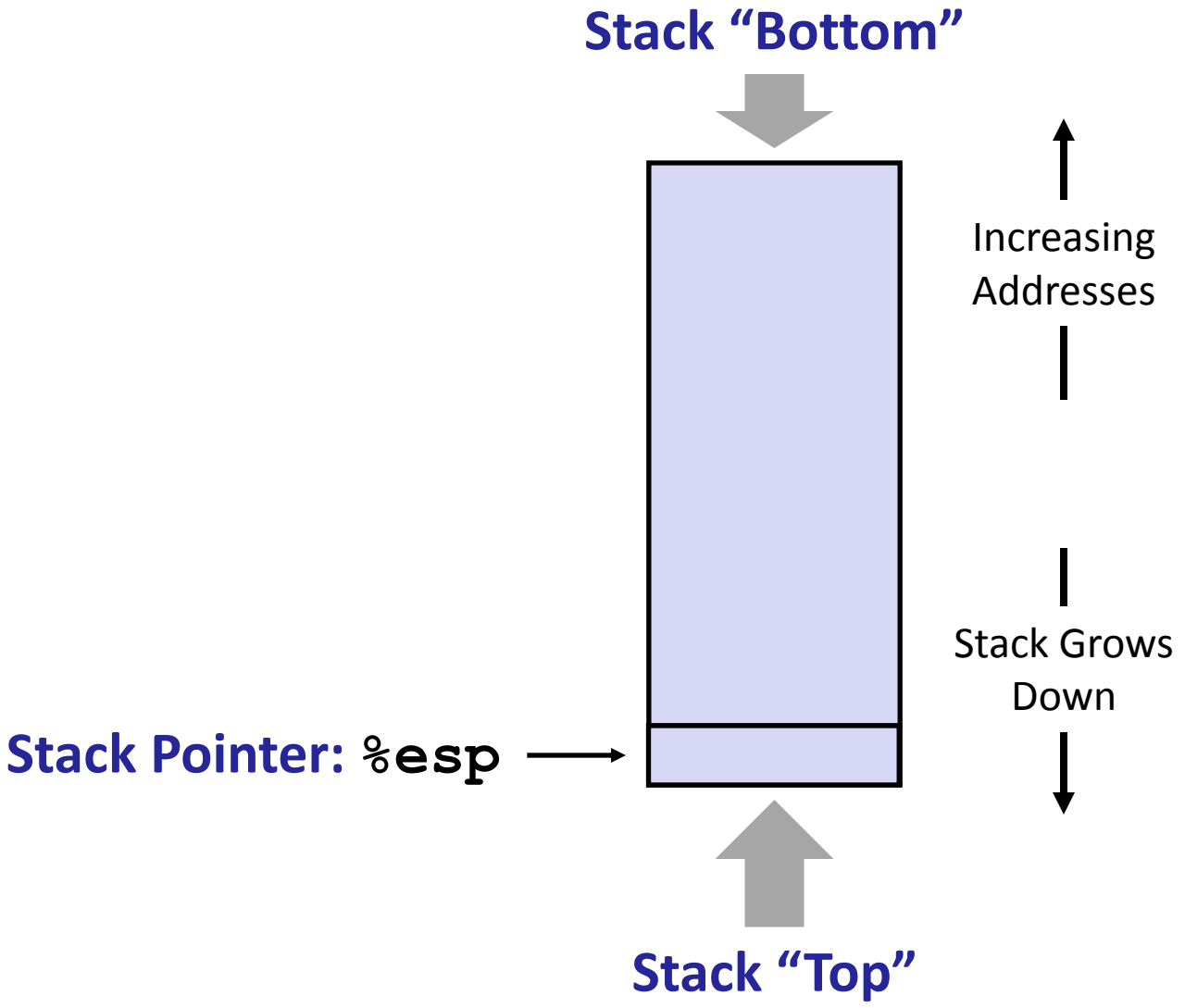
IA32 Call Stack: Push

- **`pushl` *Src***
  - Fetch value from *Src*
  - Decrement **`%esp`** by 4 *(why 4?)*
  - Store value at address given by **`%esp`**



IA32 Call Stack: Pop

- `popl` *Dest*



IA32 Call Stack: Pop

- **`popl` *Dest***
  - Load value from address **`%esp`**
  - Write value to *Dest*
  - Increment **`%esp`** by 4