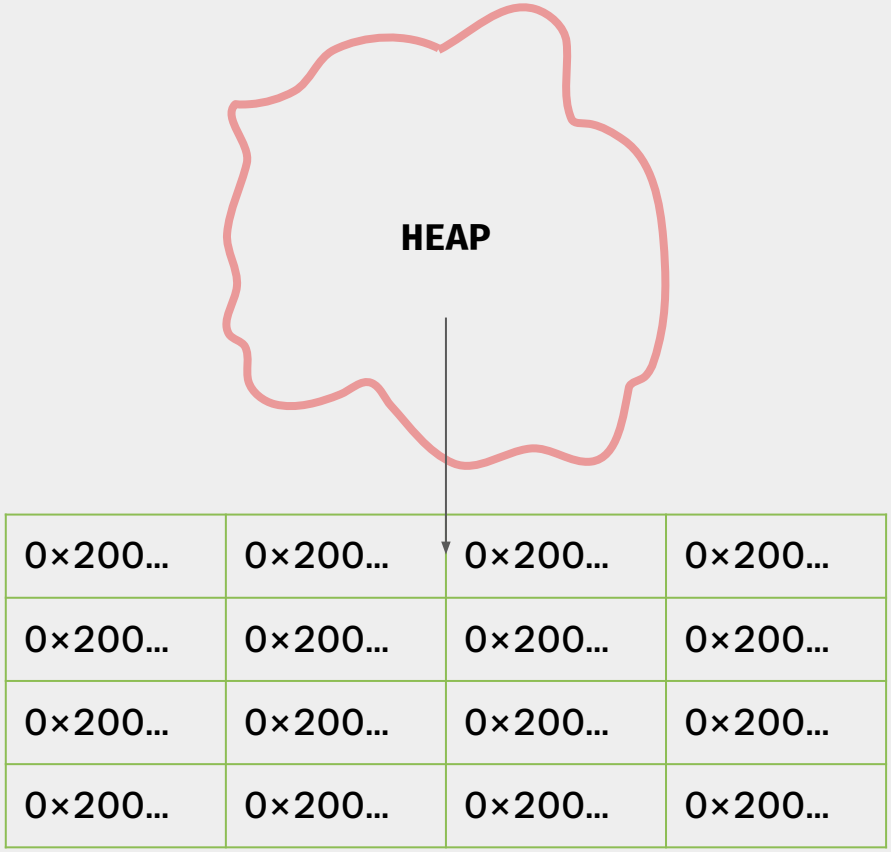
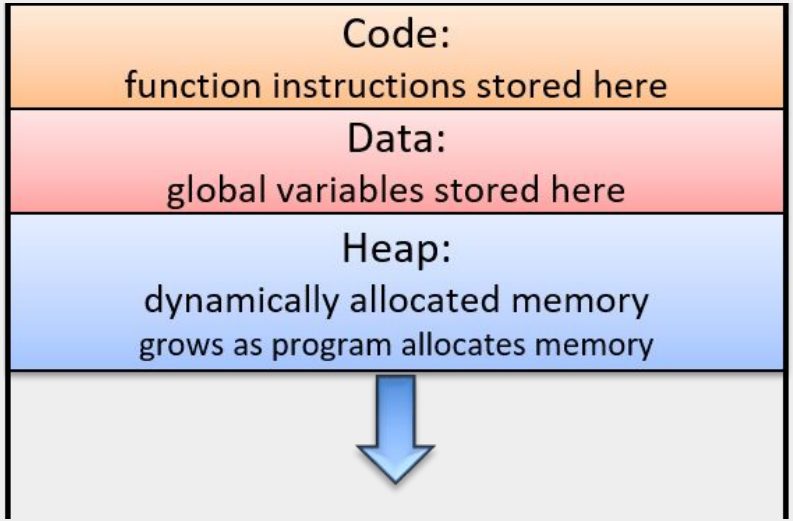
OX CB BARE METAL

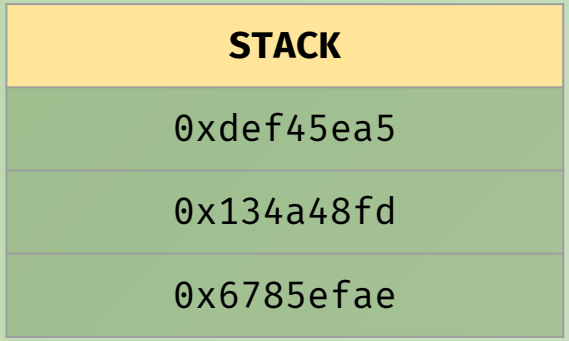
Memory and the heap



The Heap

Grows every time we create a .fill and STR/H/B data

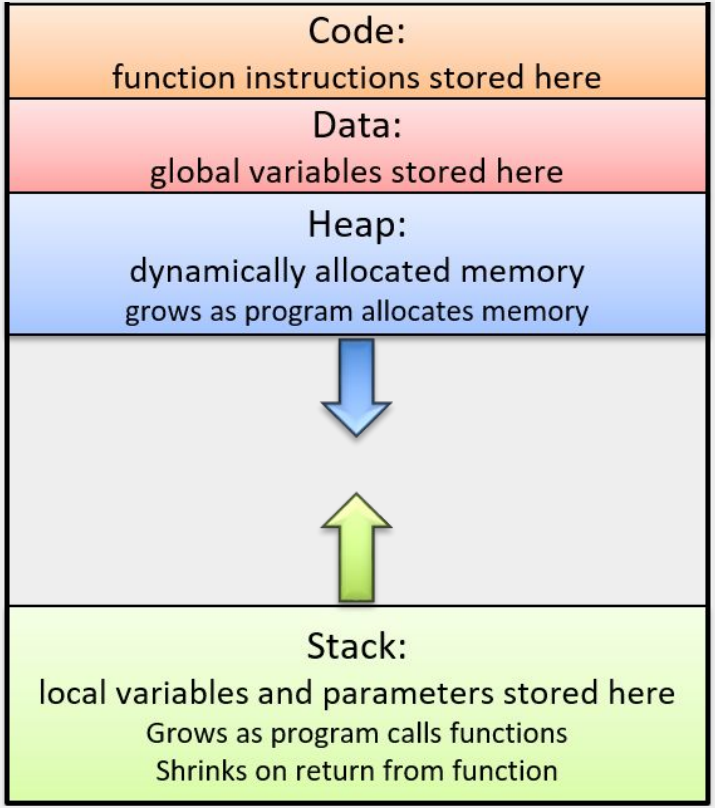




Lawful good

Chaotic neutral

Stack and Heap



Brief intro the the Stack

LIFO

Last In, First Out

VALUE
0xdef45ea5
0x134a48fd
0x6785efae

“grows down”

Brief intro the the Stack

- A linear data structure
- Largely used at the machine level to:
  - Keep track of function and subroutine addresses
  - Organize and keep track of function and subroutine arguments

Using the Stack

PUSH: fills the stack with a singular value or set of values

REGISTERS	
R0	0xdef45ea5
R1	0x134a48fd
R2	0x6785efae

PUSH {R0-R2}

STACK
0xdef45ea5
0x134a48fd
0x6785efae

0x BARE C0 METAL

Using the Stack

PUSH: Retrieves a single value or set of values

STACK
0xdef45ea5
0x134a48fd
0x6785efae

POP {R0}

REGISTERS	
R0	0x6785efae
R1	
R2	

Using the Stack

POP: Retrieves a single value or set of values

STACK
0xdef45ea5
0x134a48fd
0x6785efae

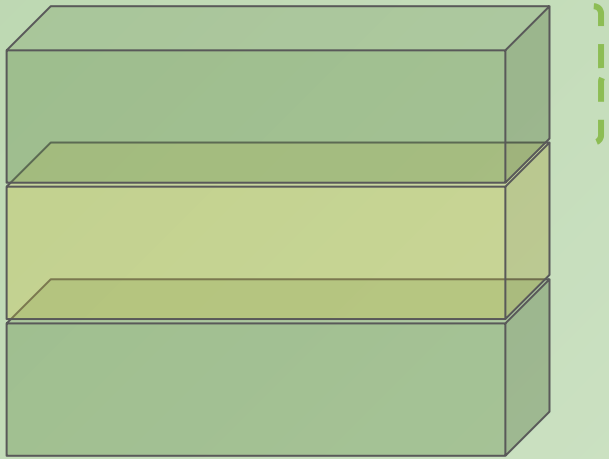
POP {R1-R2}

REGISTERS	
R0	0x6785efae
R1	0x134a48fd
R2	0xdef45ea5

PUSH and POP

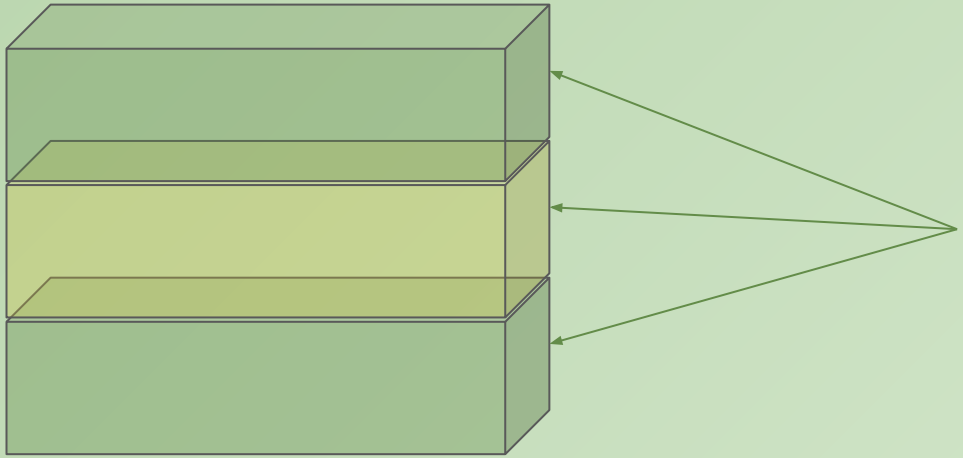
Argument formulae

PUSH	$\{R_M, R_D, \ldots\}$
PUSH	$\{R_M - R_D\}$
POP	$\{R_M, R_D, \ldots\}$
POP	$\{R_M - R_D\}$



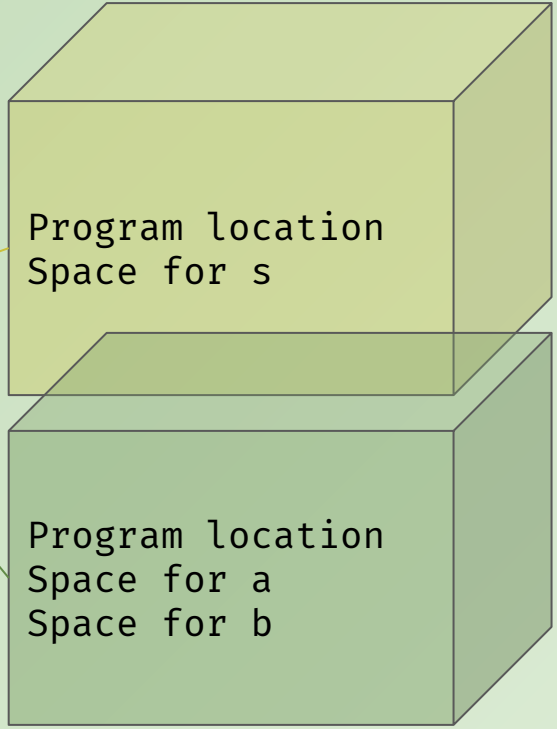
Individual stack “frame”

Contains all PUSH’d, POP’d values for a given subroutine



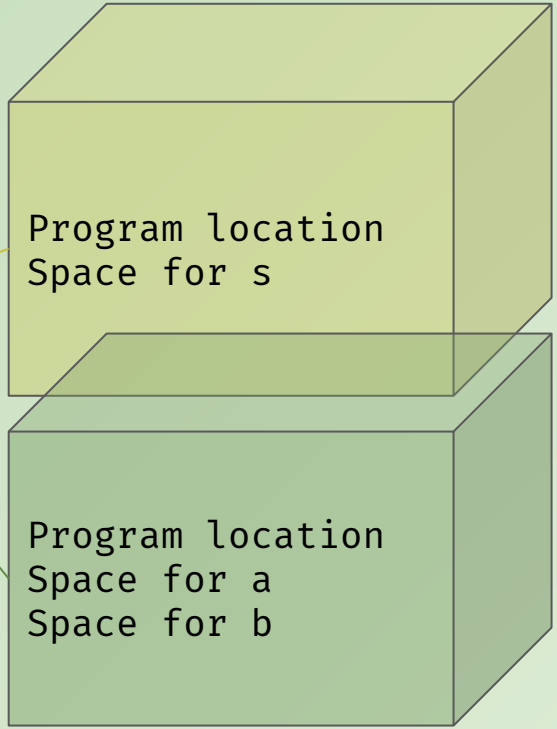
Implies: each subroutine can have its own stack “frame”

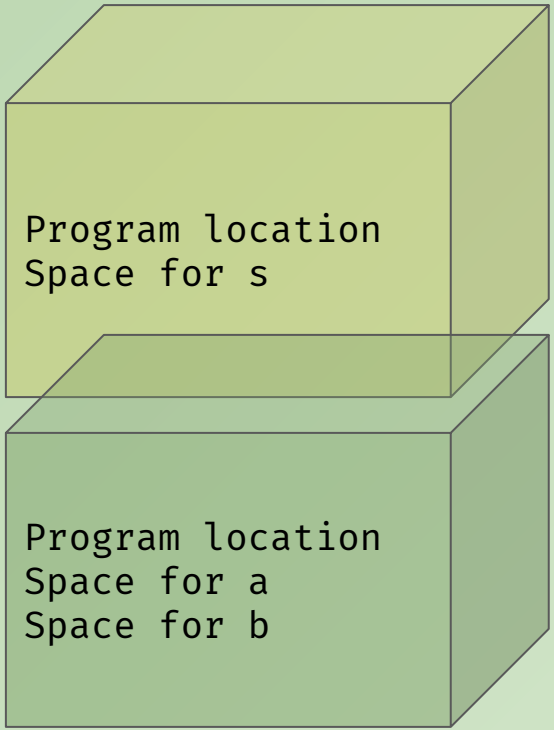
```
def add(a: int, b: int) -> int:
    c = a + b
    return c

s = add(2, 3)
print(s)
```



```
def add(a: int, b: int) -> int:

    c = a + b

    return c


s = add(2, 3)

print(s)
```

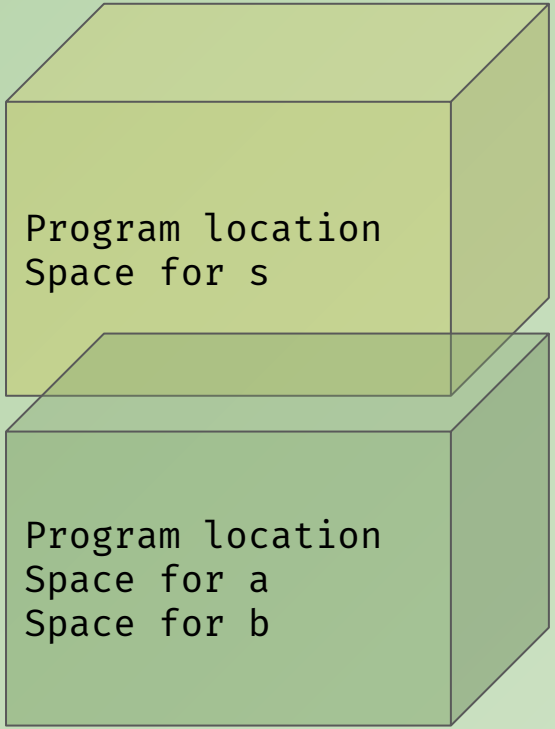
But we can't do this, can we?

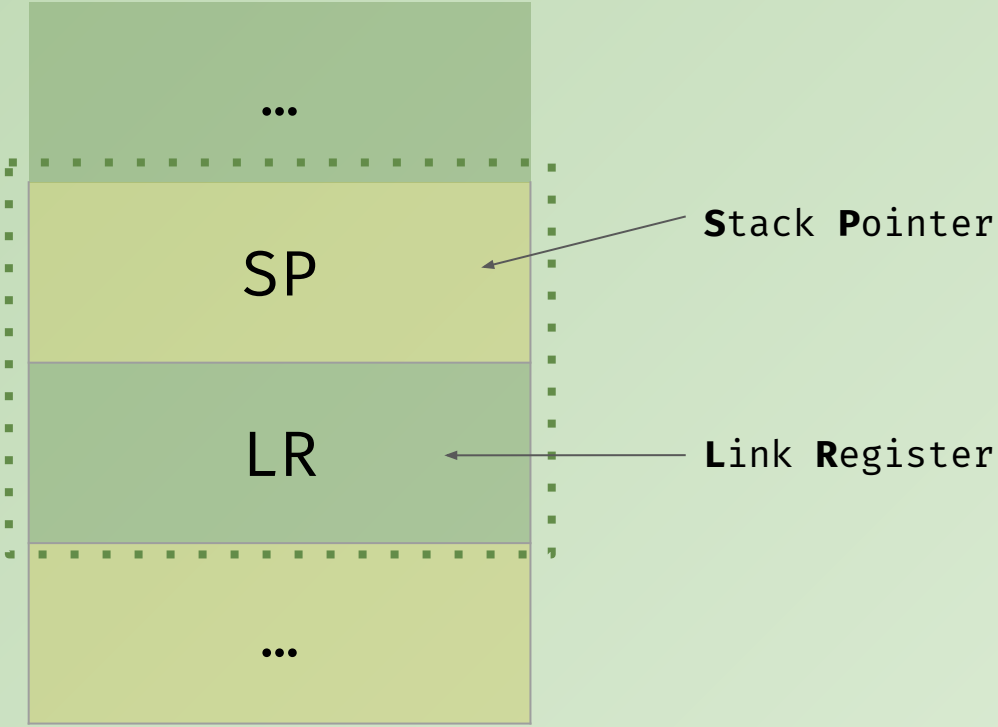




Back to here?

How do we get
from here…





BL	**B**ranch and **L**ink	Jump to a label; store location in **LR**
BX	**B**ranch and e**X**ecute	Jump back to a memory location and continue to execute

BL, BX

BL	LABEL
BX	R_D