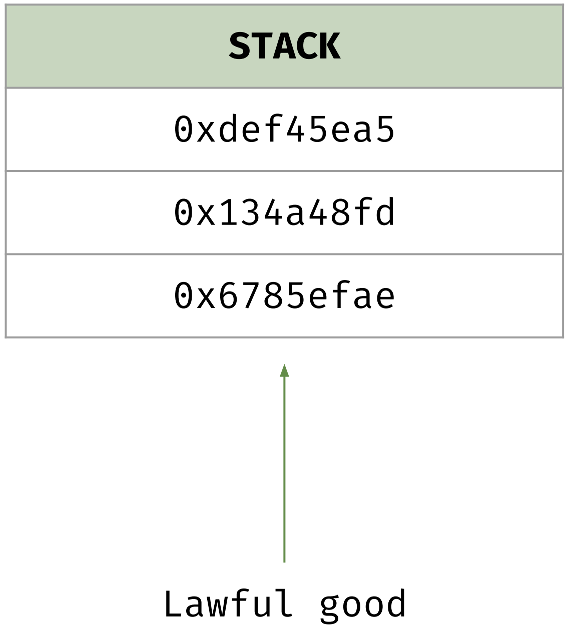
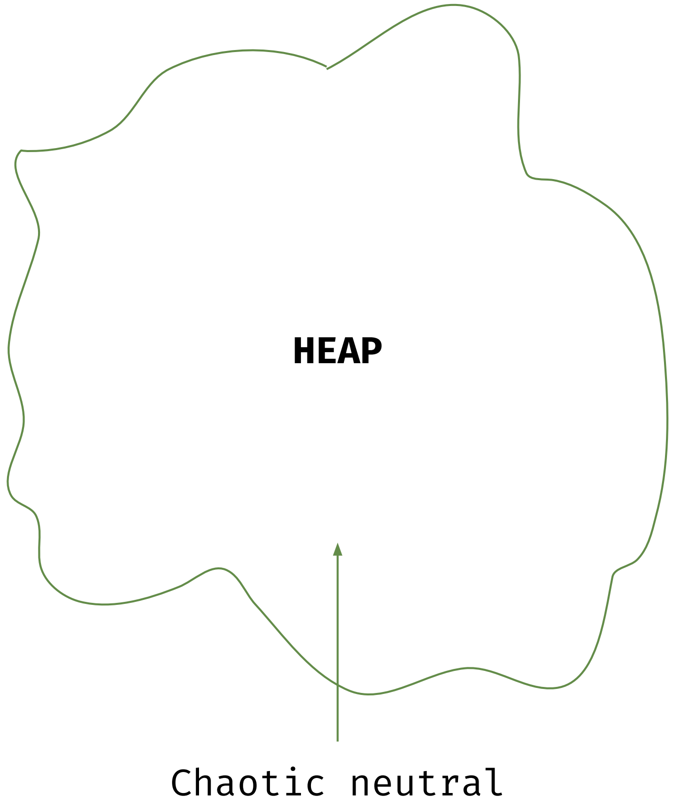
0X BARE

CMPSC 0XC8 ALLEGHENY COLLEGE 0X7E7

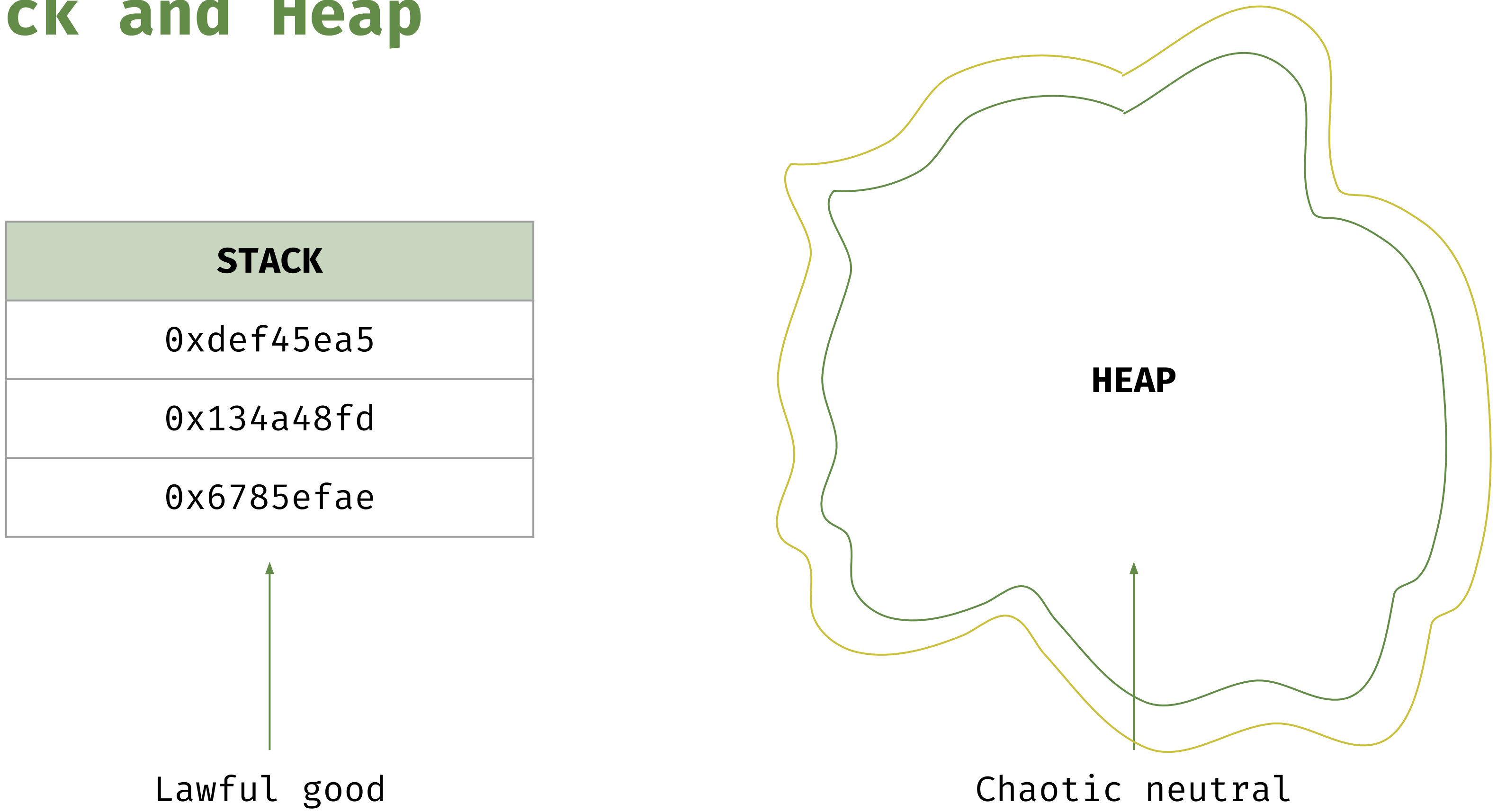
C8 METAL

Stack and Heap

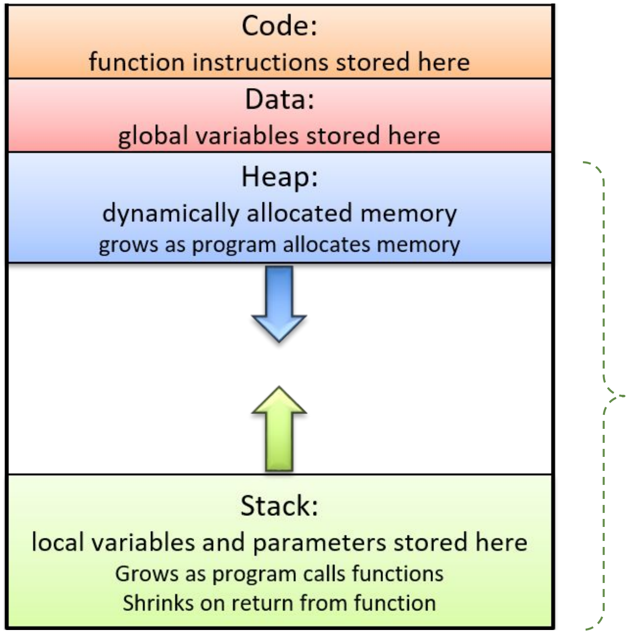




Stack and Heap



Stack and heap

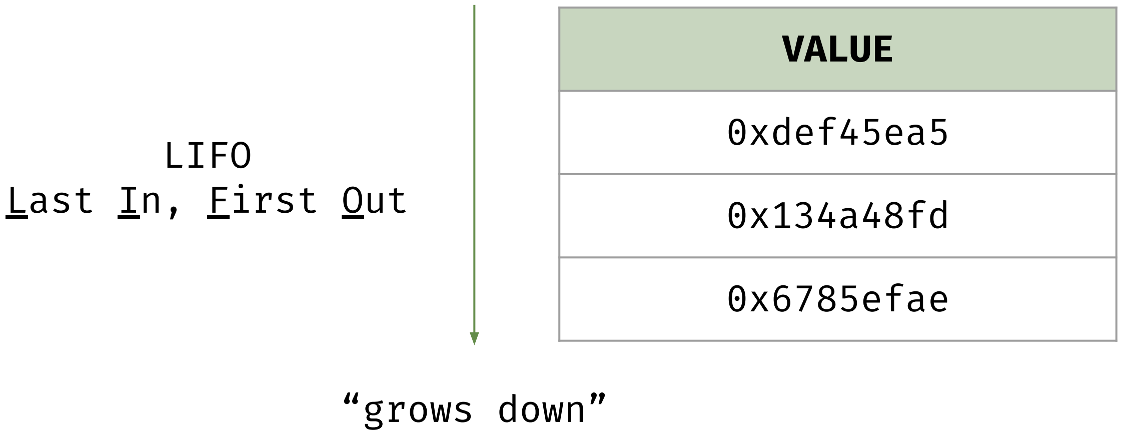

Code:
function instructions stored here
Data:
global variables stored here
Heap:
dynamically allocated memory
grows as program allocates memory
Stack:
local variables and parameters stored here
Grows as program calls functions
Shrinks on return from function

Dynamically allocated; can grow or shrink depending on usage.

(Can collide!)

Brief introduction to the “stack”

There exists another quasi-temporary spot for memory which can get us out of a tricky jam: the *stack*.



VALUE
0xdef45ea5
0x134a48fd
0x6785efae

Brief introduction to the “stack”

PUSH
Places a value in the stack *in order*

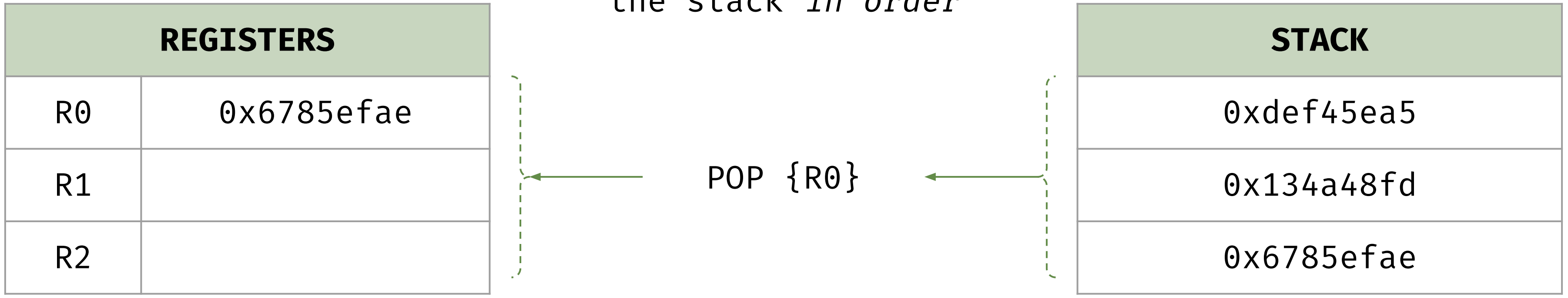
REGISTERS	
R0	0xdef45ea5
R1	0x134a48fd
R2	0x6785efae

$\longrightarrow$ PUSH {R0-R2} $\longrightarrow$

STACK
0xdef45ea5
0x134a48fd
0x6785efae

Brief introduction to the “stack”

POP
Extracts values from the stack *in order*



REGISTERS	
R0	0x6785efae
R1	
R2	

```
POP {R0}
```

STACK
0xdef45ea5
0x134a48fd
0x6785efae

Brief introduction to the “stack”

POP
Extracts values from the stack *in order*

REGISTERS	
R0	0x6785efae
R1	0x134a48fd
R2	0xdef45ea5

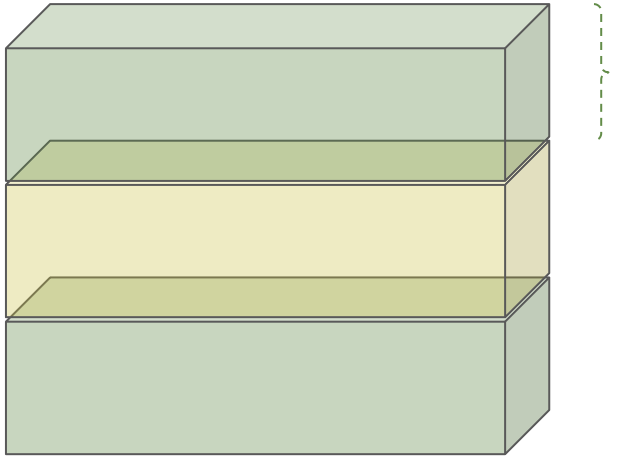
$\leftarrow$ POP {R1,R2} $\leftarrow$

STACK
0xdef45ea5
0x134a48fd
0x6785efae

Brief introduction to the “stack”

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

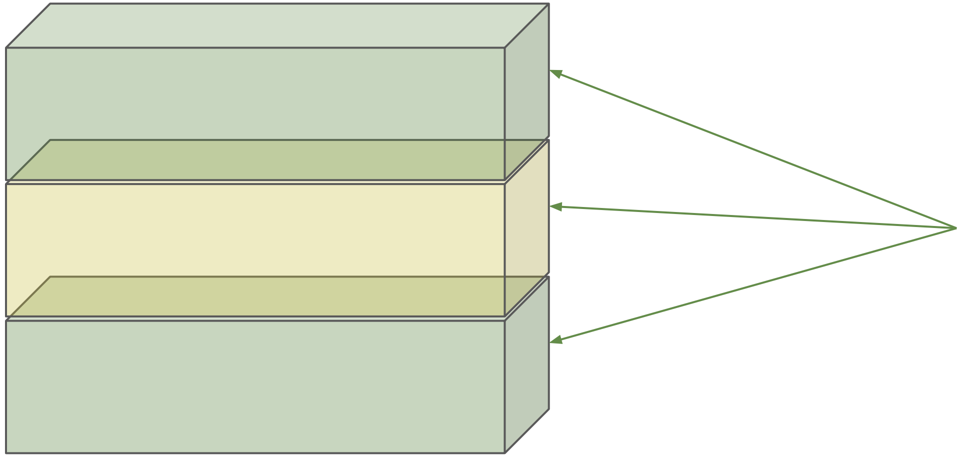
Getting framed



Individual stack "frame"

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

Getting framed

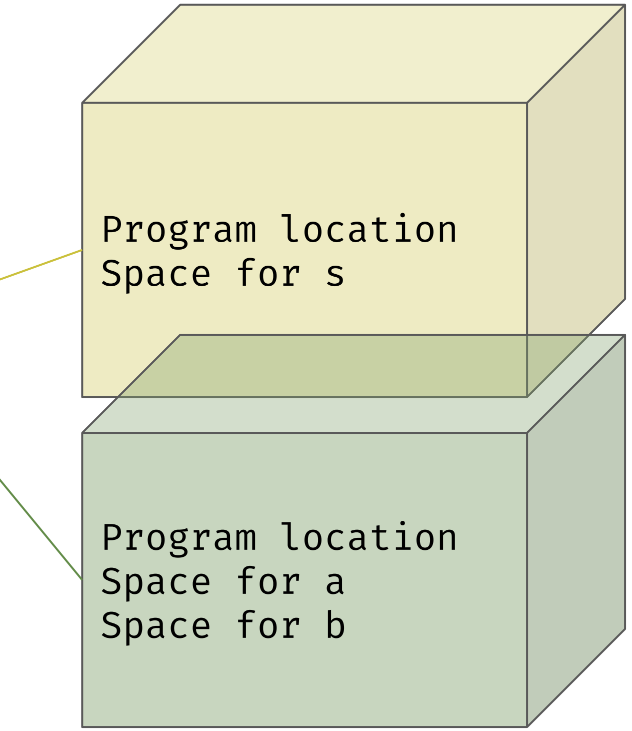


Implies: each subroutine can have its own stack "frame"

Getting framed

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


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

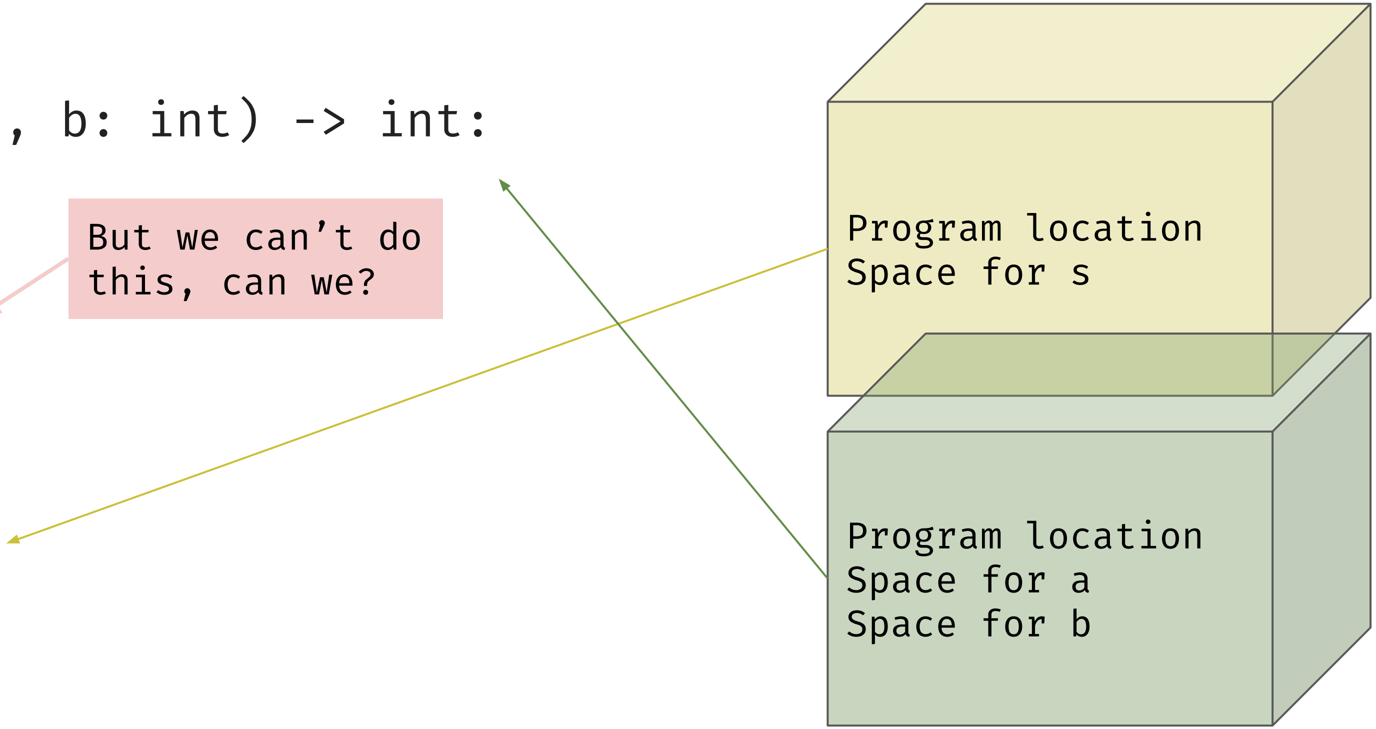


Getting framed

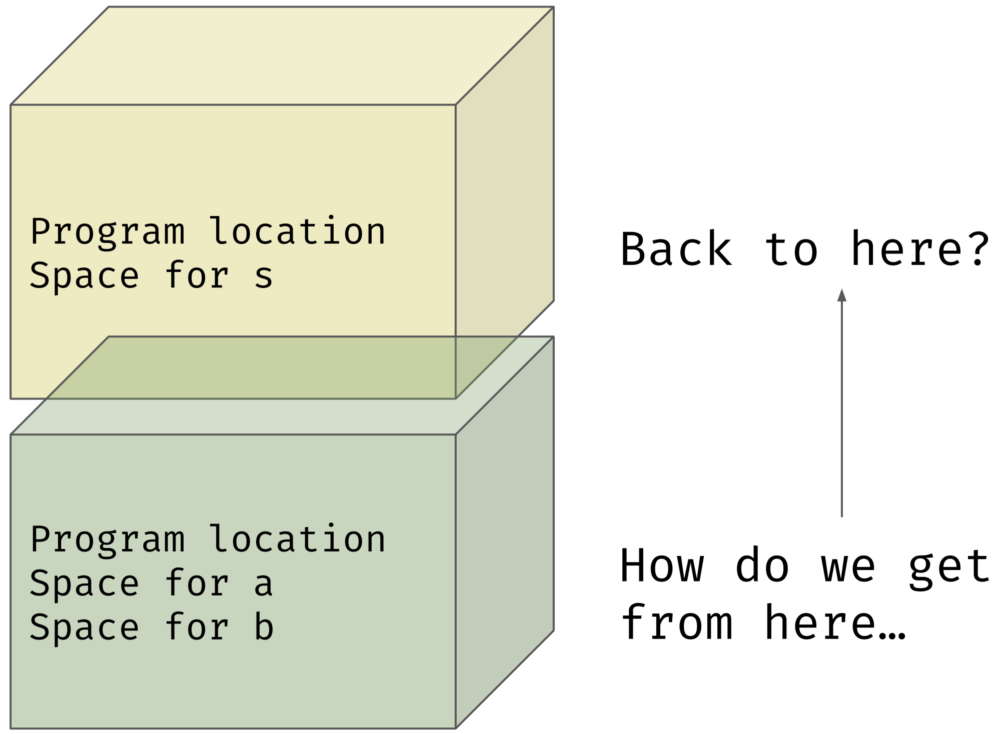
```
def add(a: int, b: int) -> int:
    c = a + b
    return c

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

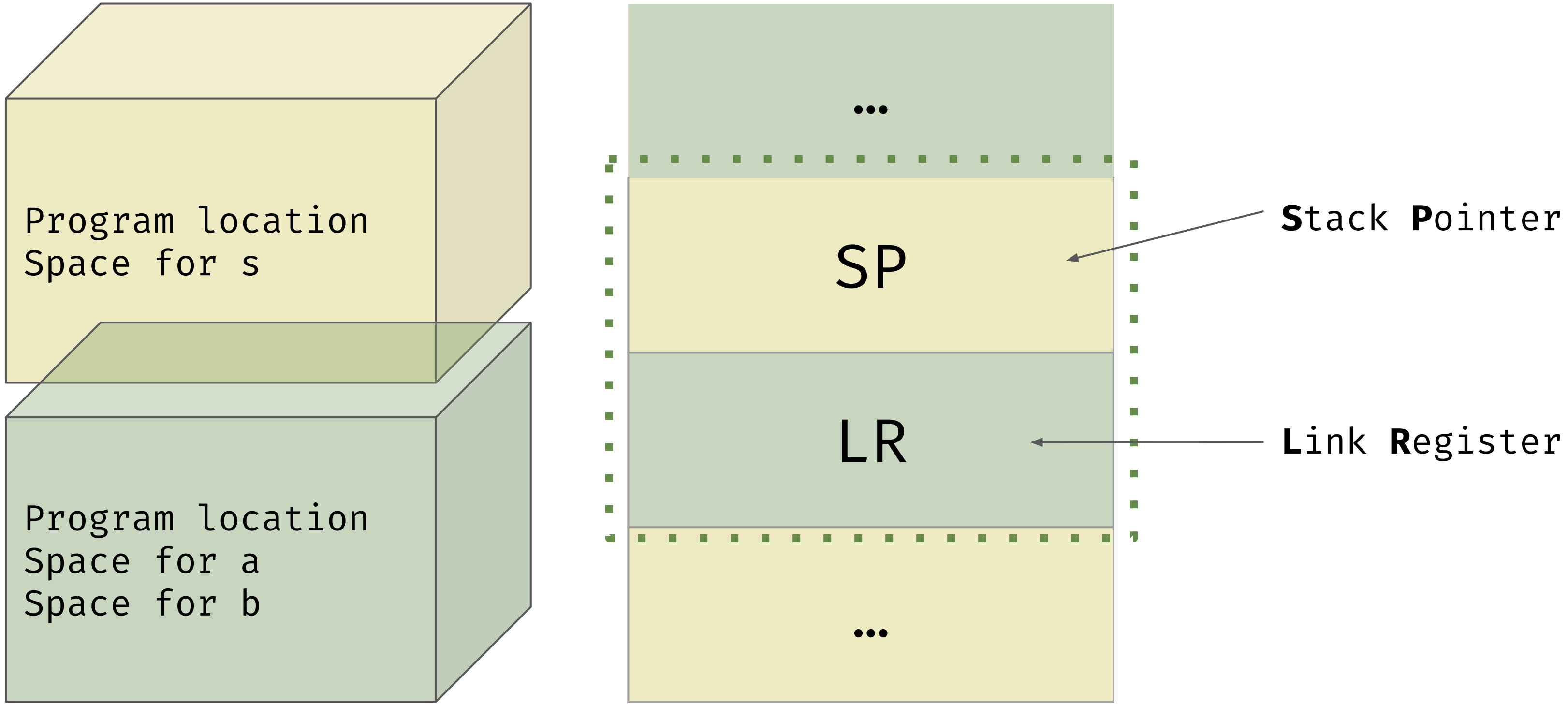
But we can't do this, can we?



Getting framed

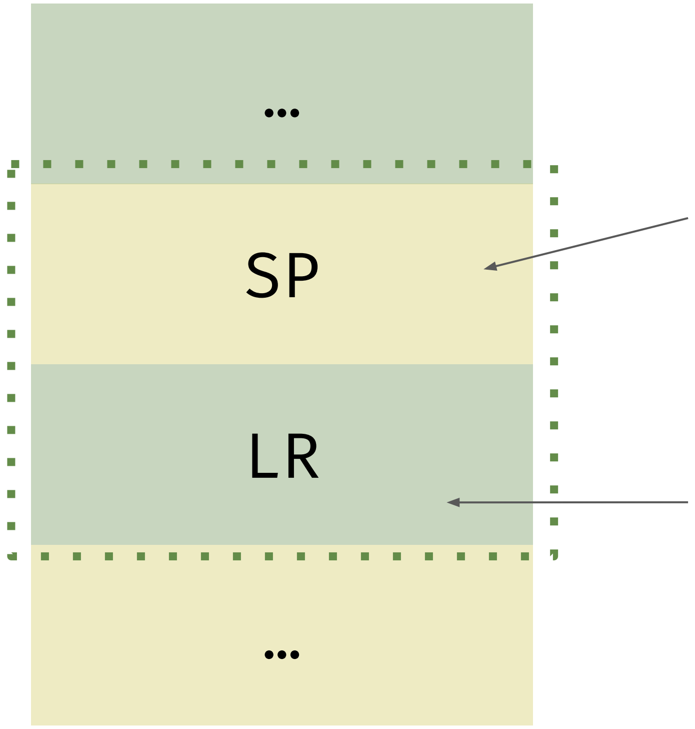


Getting framed



Getting framed

Seems like the CARDIAC had
something like this…



Stack **P**ointer

Link **R**egister: stores memory location of the previous point in memory that we branched from.

Going out on a limb…

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

Going out on a limb…

```
BL      LABEL

BX      R_D
```

Has to be a memory location stored in a register…

But can't POP {LR}

Getting shifty

Logical **S**hift **R**ight

LSR R_D, R_X, $\#_{BITS}$

LSL R_D, R_X, $\#_{BITS}$

Logical **S**hift **L**eft

```
LSR     R5, R5, #24
```

0xff000000 $\longrightarrow$ 0x000000ff

```
LSL     R5, R5, #24
```

0x000000ff $\longrightarrow$ 0xff000000