Processor control flow

Simon Hollis, COMS12600

Part 2

Allowing modular programming

Sub-routines

- *Sub-routines* (or procedures, methods, functions) are a key component of modern modular programming.
- Supporting them efficiently in hardware is therefore very important.
- Most architectures, therefore provide some primitive support for sub-routines.

The Link Register (LR)

- Most processor implementations make use of a component called a ***link register (lr)***.
- It's a register, like any other.
- It is the same length as the PC.
- It is used on sub-routine calls and exits, where it stores addresses.
- It may or may not be available for general purpose usage as well.

Register-based calling

- Most processors support instructions for calling and returning from sub-routines:
  - e.g.
    - **`CALL foo`** = copy return address to special return (or *link*) register, then branch to ‘foo’
    - **`RETURN`** = branch to link register address

Register linking

Link register

PC 0

0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

CALL	**ADD**	

next current prev

Register linking

Link register

1+1=2

PC 1

0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

MOV	CALL	ADD

next current prev

Register linking

Link register

2

PC 100

RET	SUB	CALL

next current prev

0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

Register linking

Link register

2

PC 101

NOP	RET	SUB

next current prev

0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

Register linking

Link register

2

PC 102

0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

NOP	RET	SUB

next current prev

Register linking

Link register

2

PC 2

0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

…	**MOV**	**RET**

next current prev

Supporting multiple calls

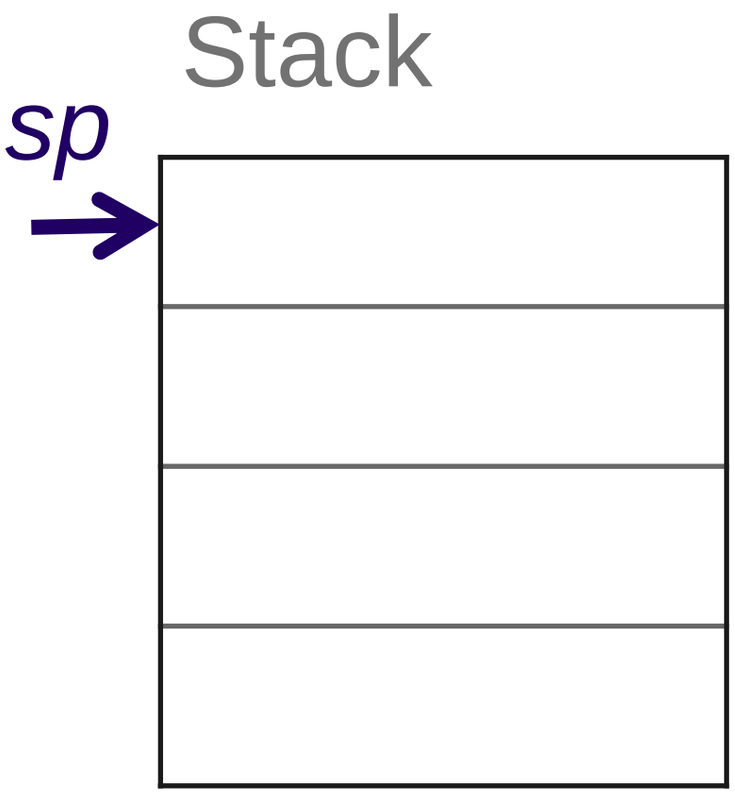
- How to allow for:
  - Multiple levels of sub-routine call?
  - Recursive sub-routine calls?

Stack-based linking

- Produce a **stack** containing a list of return addresses.
- When calling, **`PUSH`** a return address onto the stack.
- When returning **`POP`** a return address off the stack.

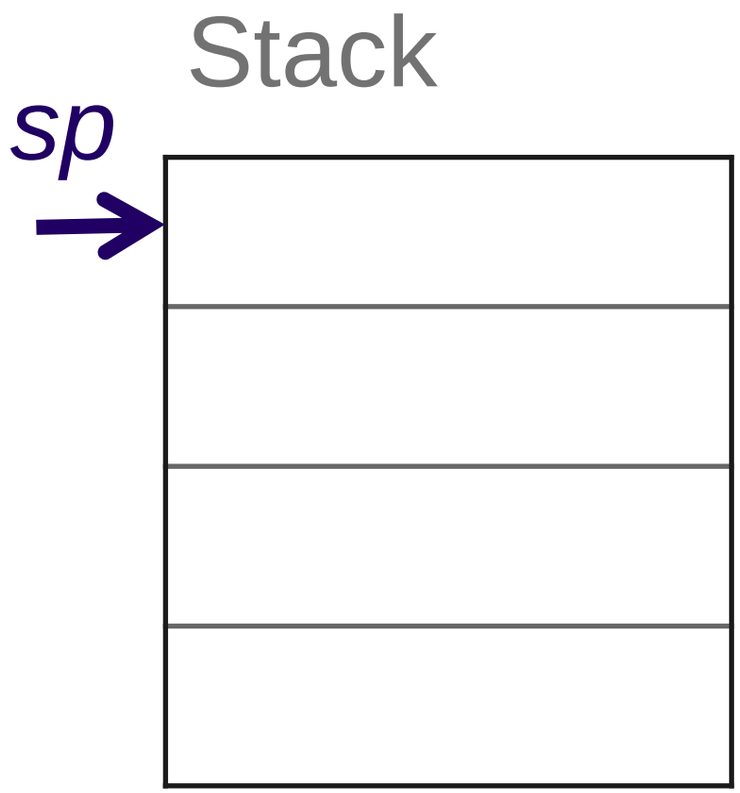
Recall the stack

- A stack is like a stack of plates.
- New items go on the top and push down the previous ones.
- You can access the items at the top, but not those at the bottom
- The amount of items is tracked by a *'stack pointer'*

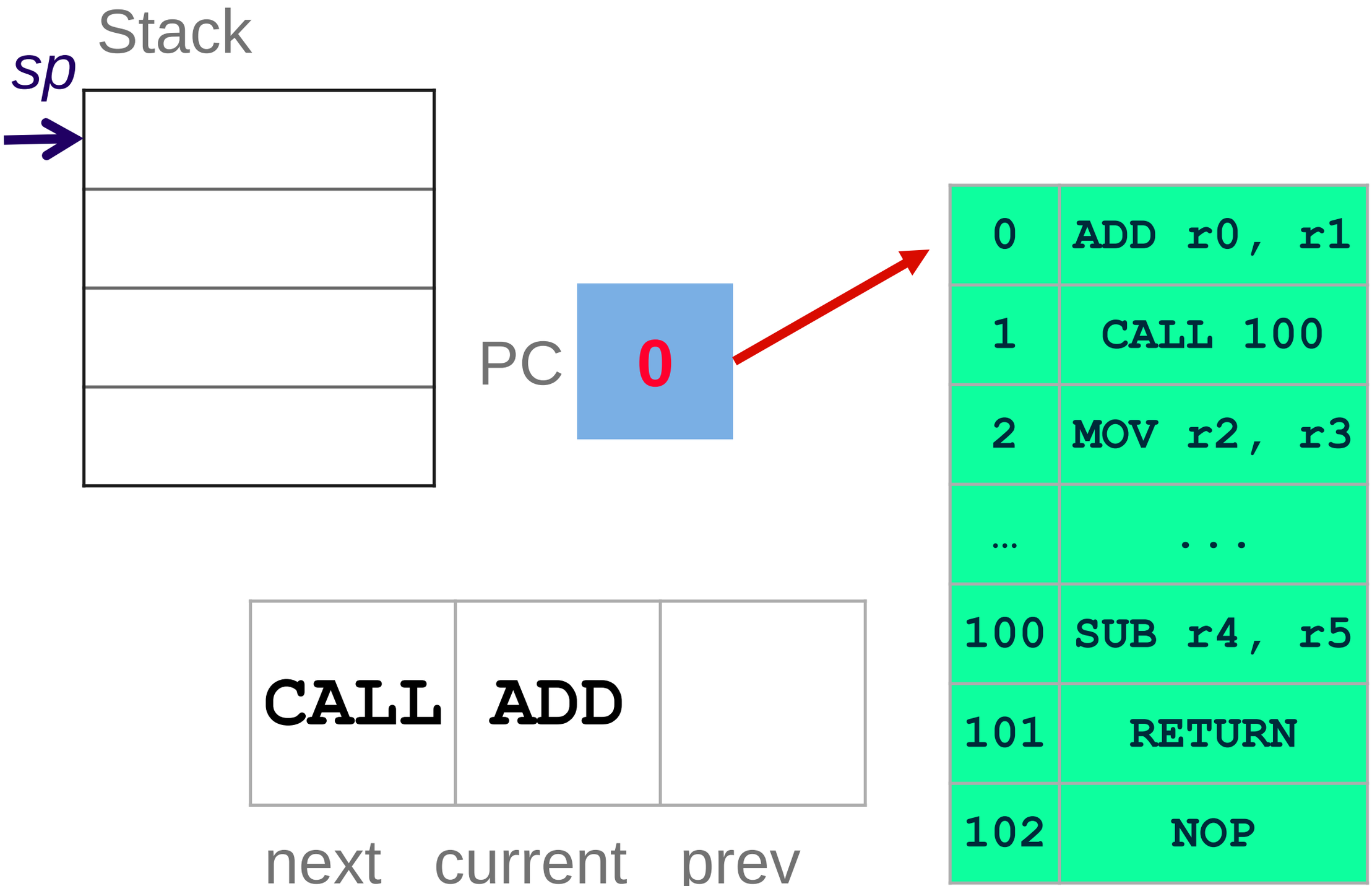


Stack example

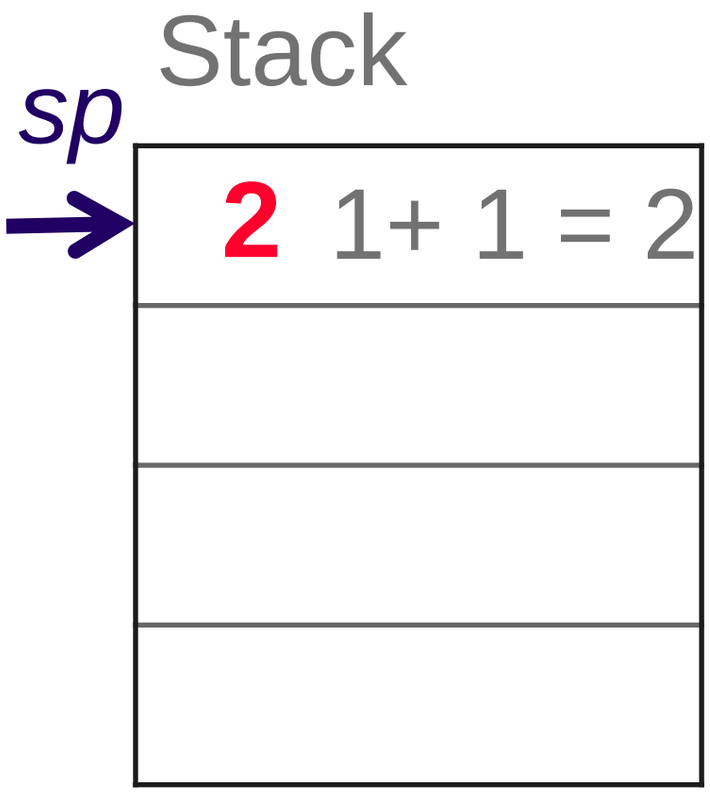
- Let’s add the following values in order to the stack: 1, 2, 3.
- We’ll use an ‘*empty*’ paradigm for the *sp*.

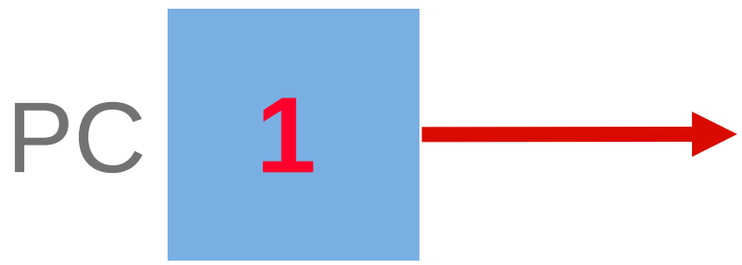


Stack-based linking



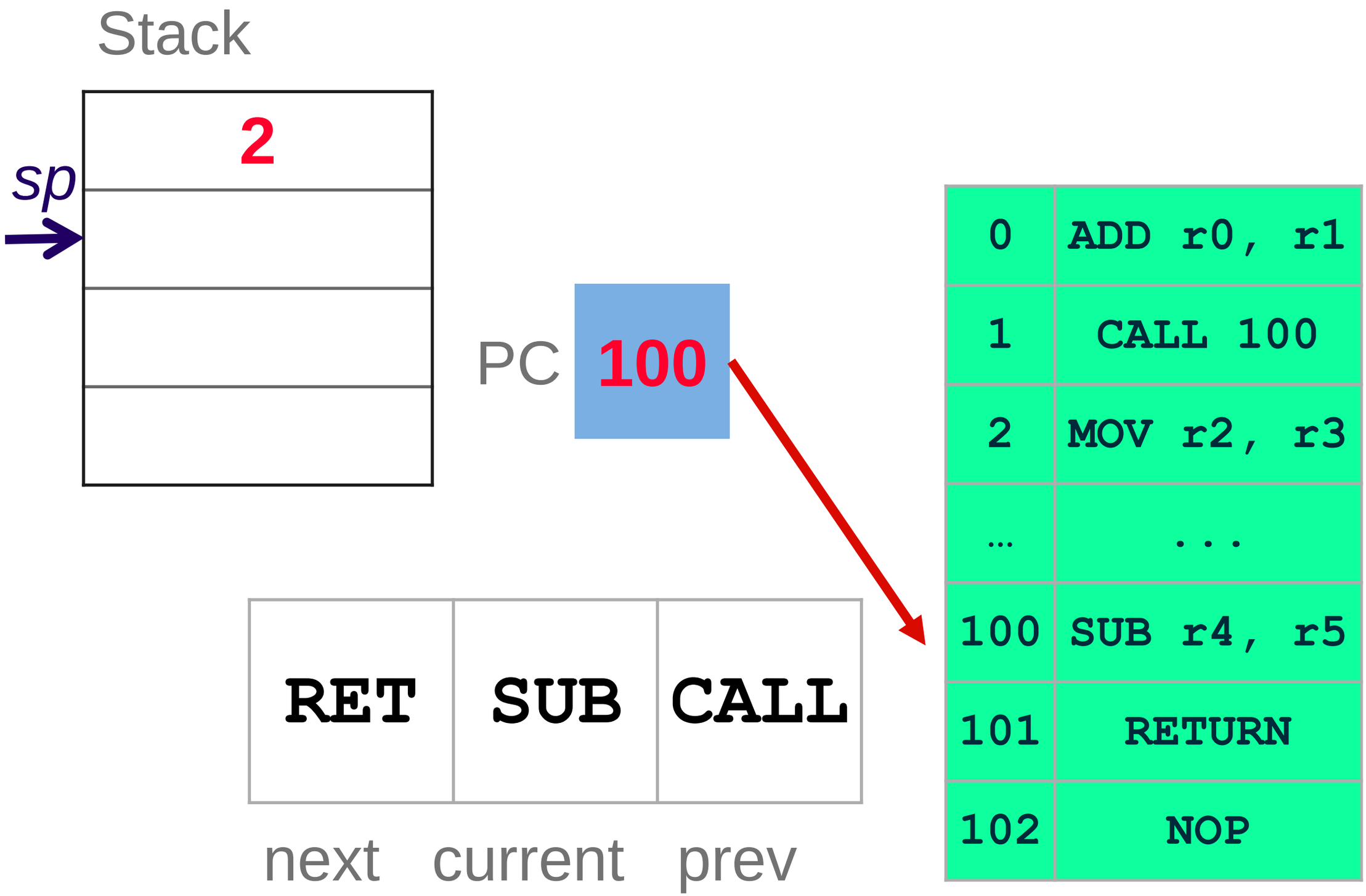
Stack-based linking



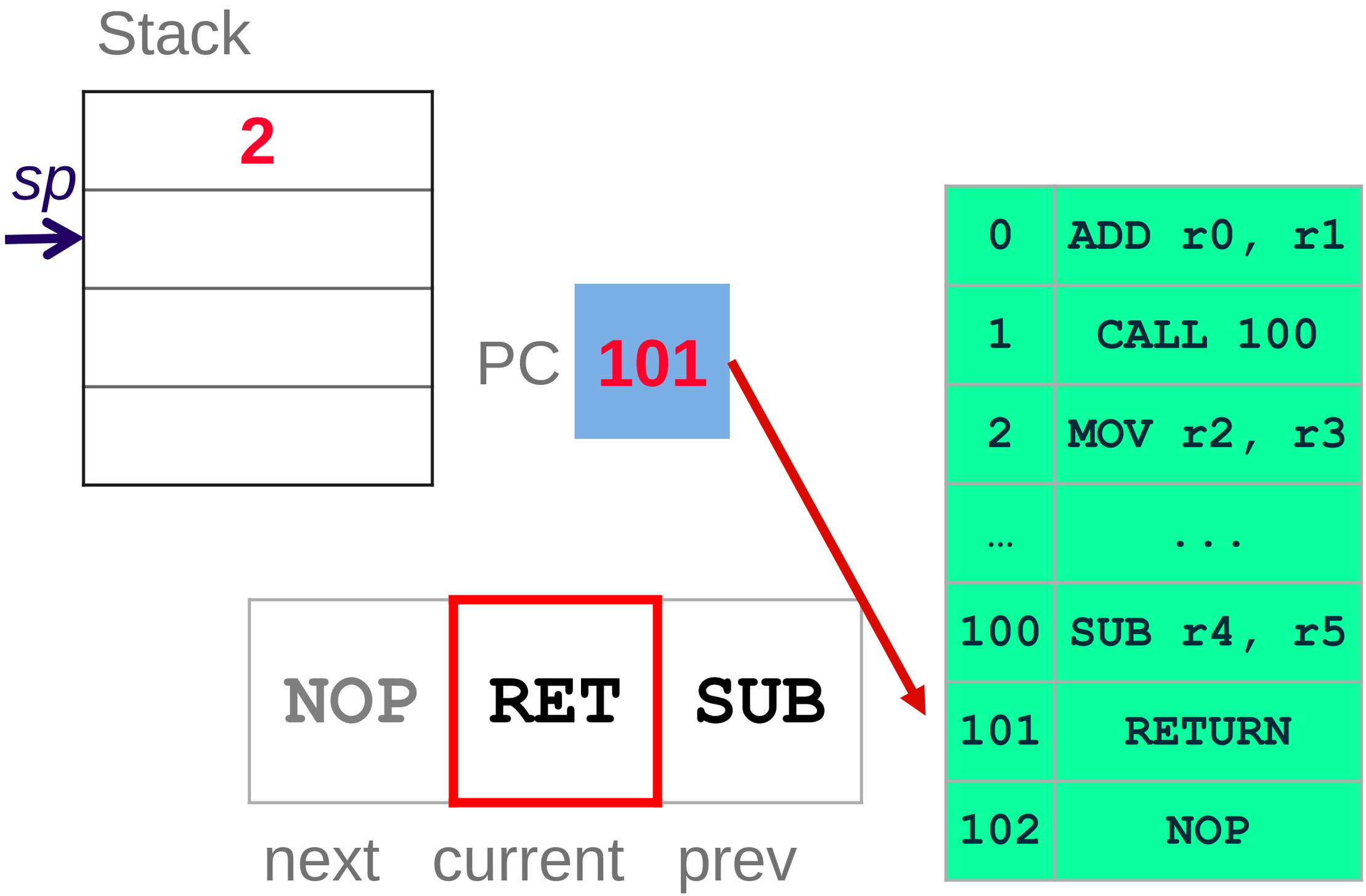


0	ADD r0, r1
1	CALL 100
2	MOV r2, r3
…	. . .
100	SUB r4, r5
101	RETURN
102	NOP

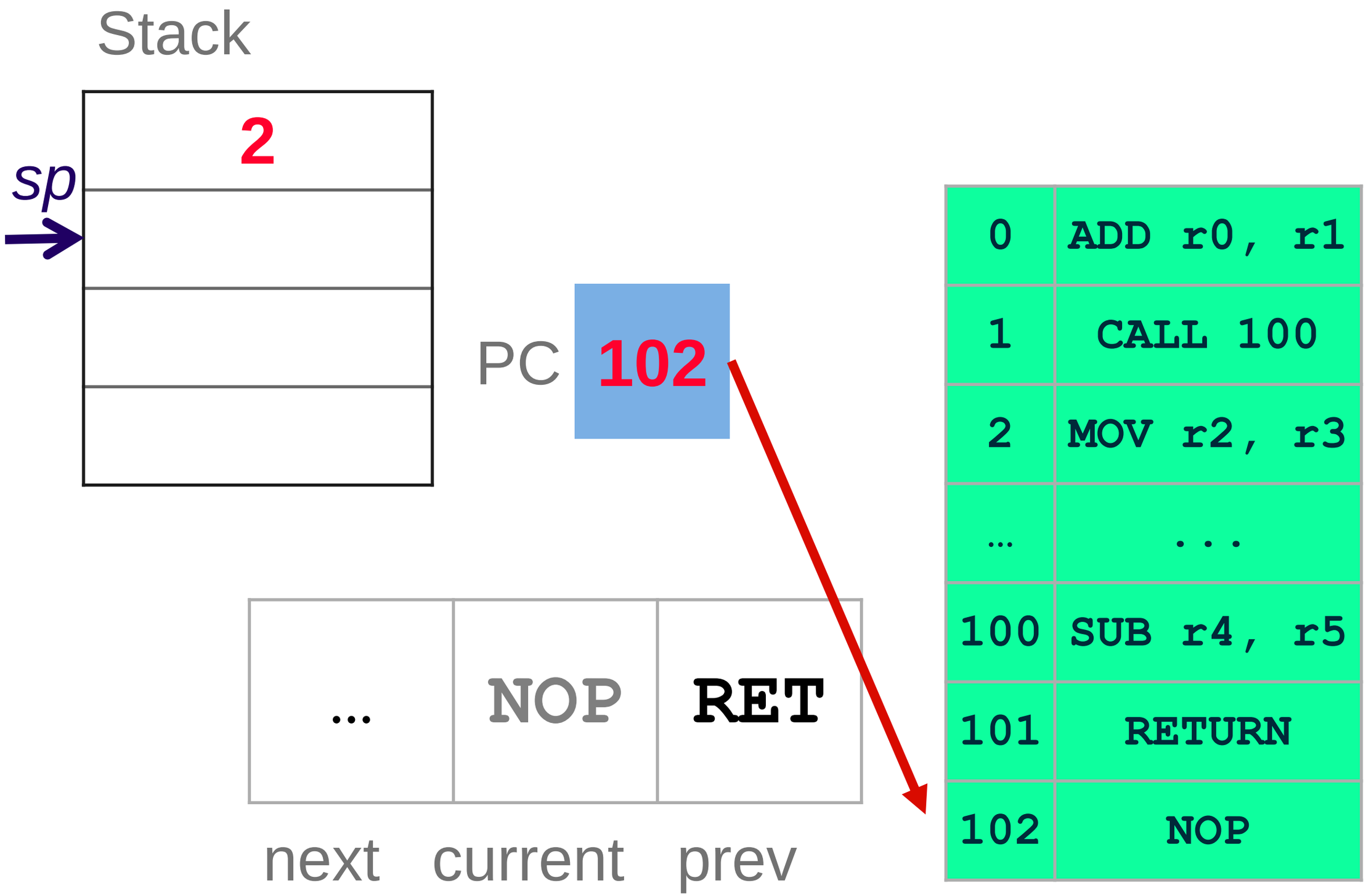
Stack-based linking



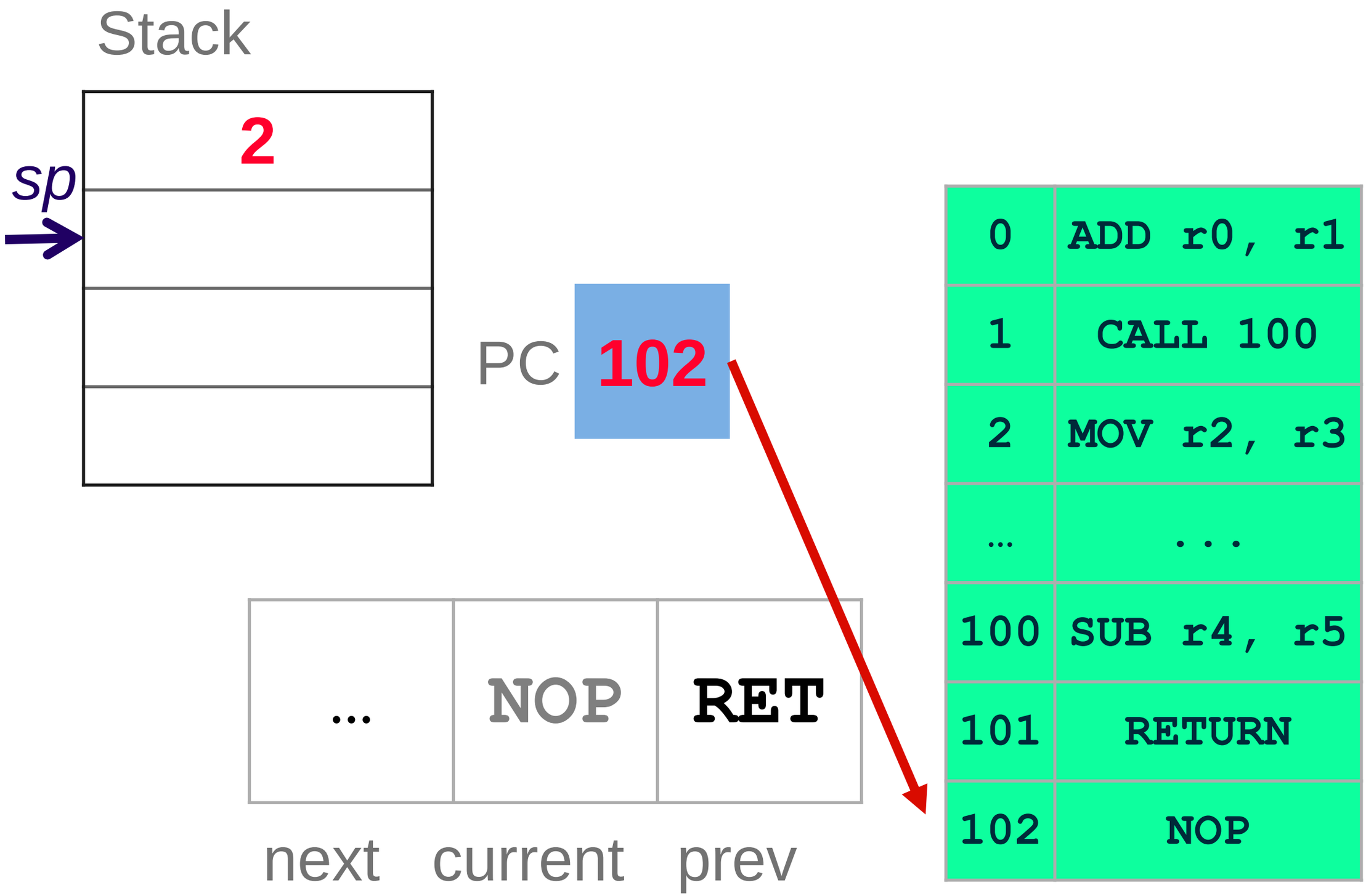
Stack-based linking



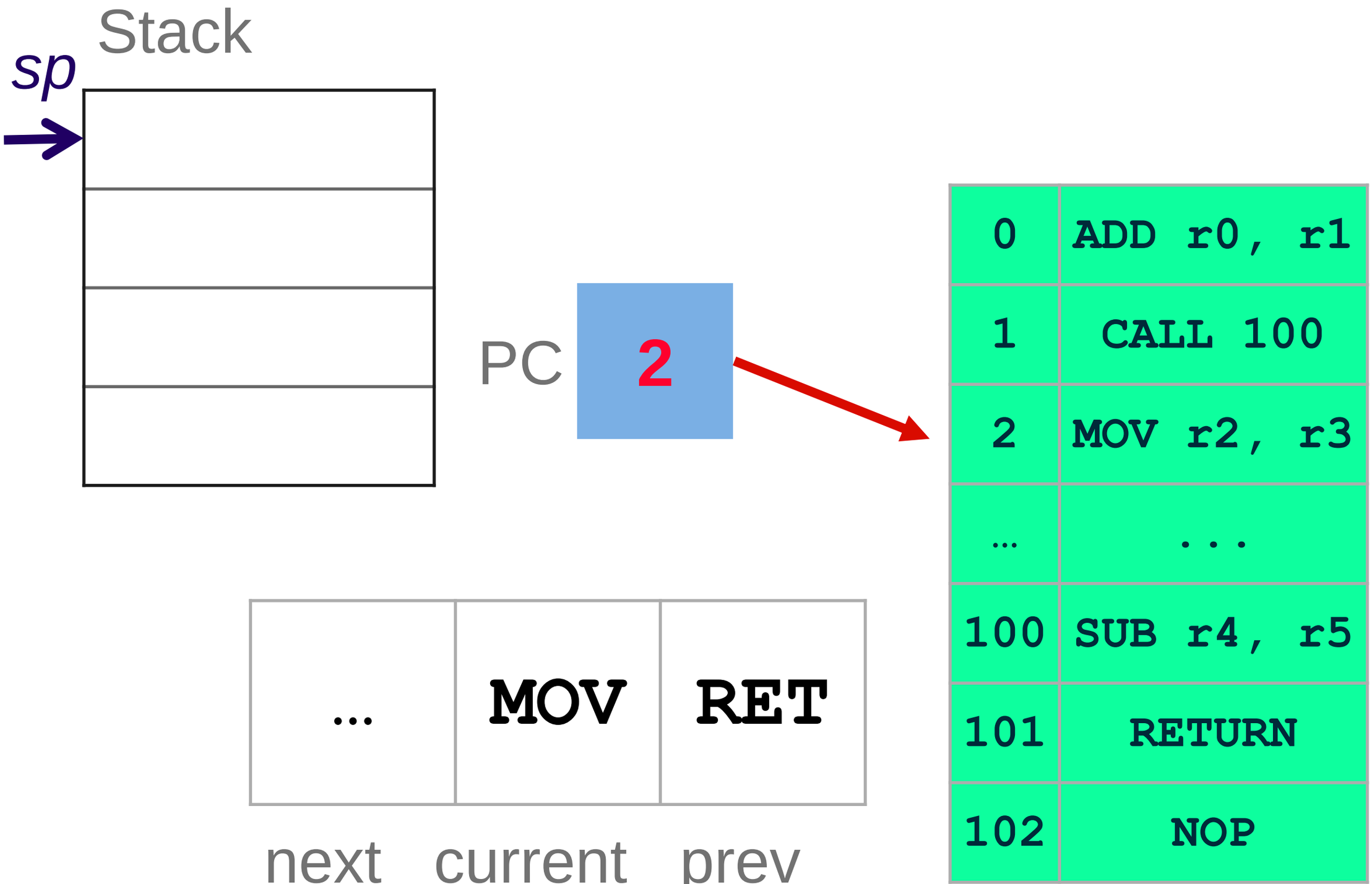
Stack-based linking



Stack-based linking



Stack-based linking



Summary

- We have seen that ***control flow*** is a vital part of a modern programming flow.
- Therefore, we need hardware support to allow its use and provide efficient execution.
- We have seen:
  - Branches
  - Conditional flow
  - Sub-routine linking