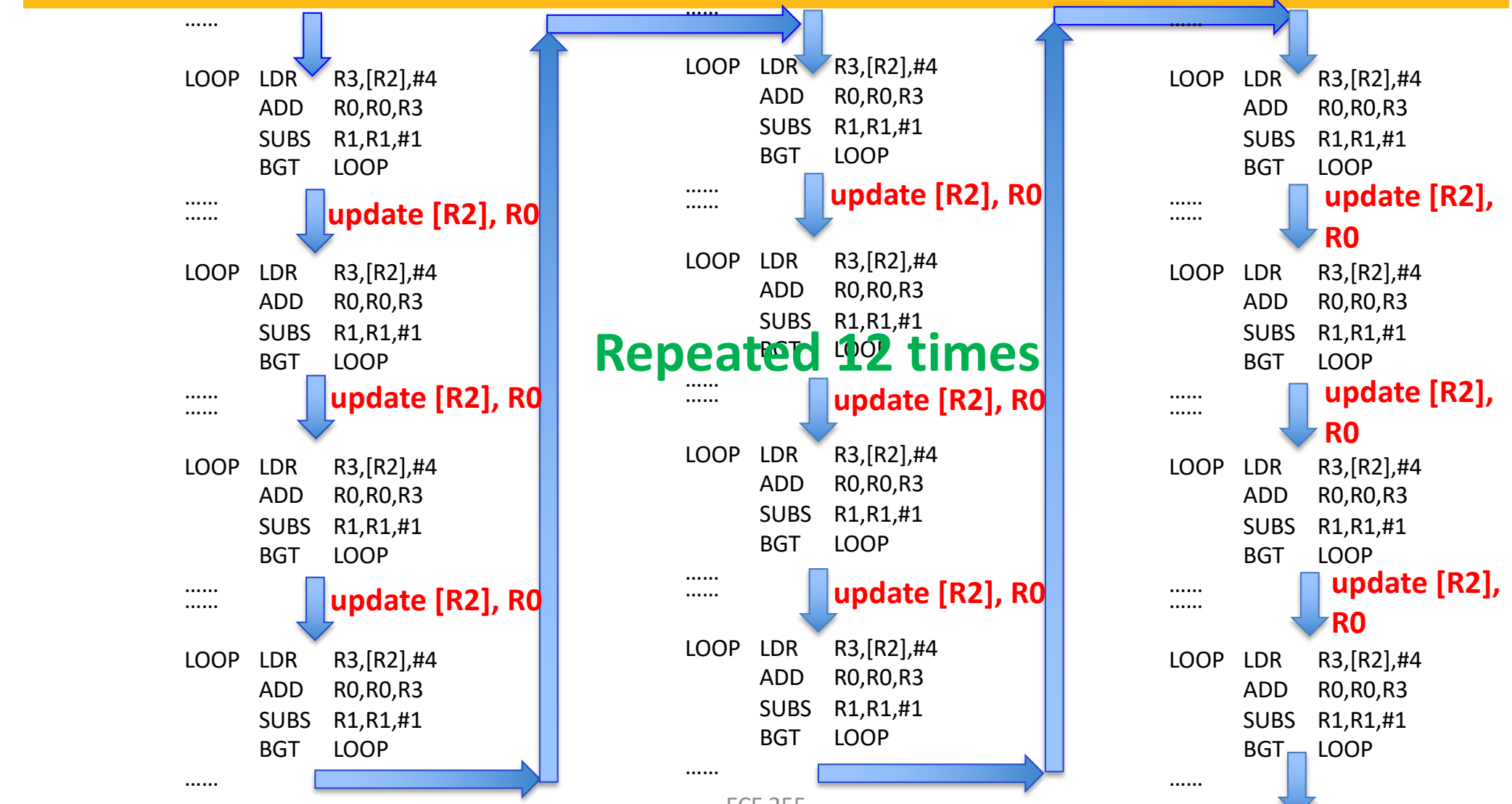
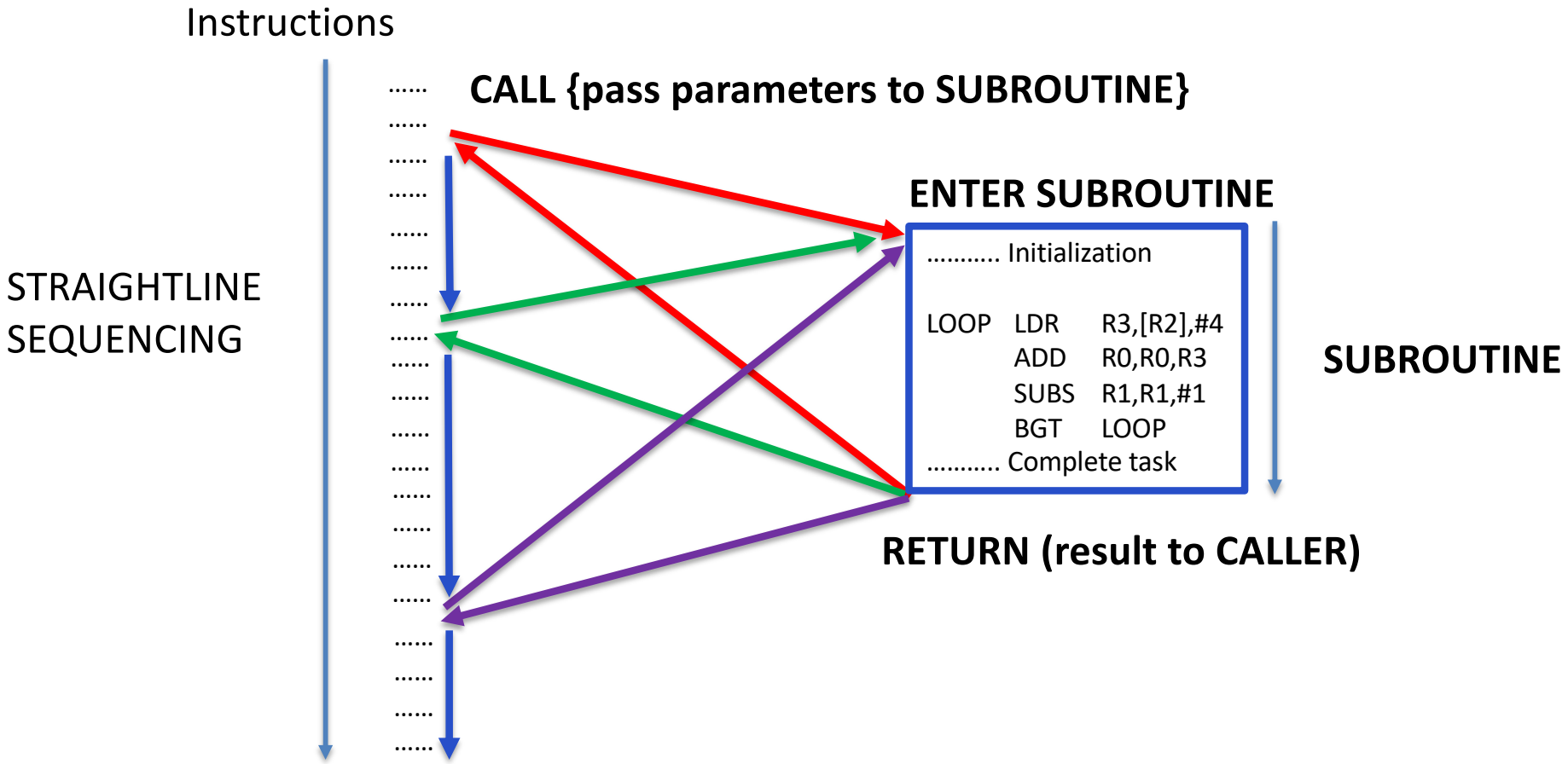
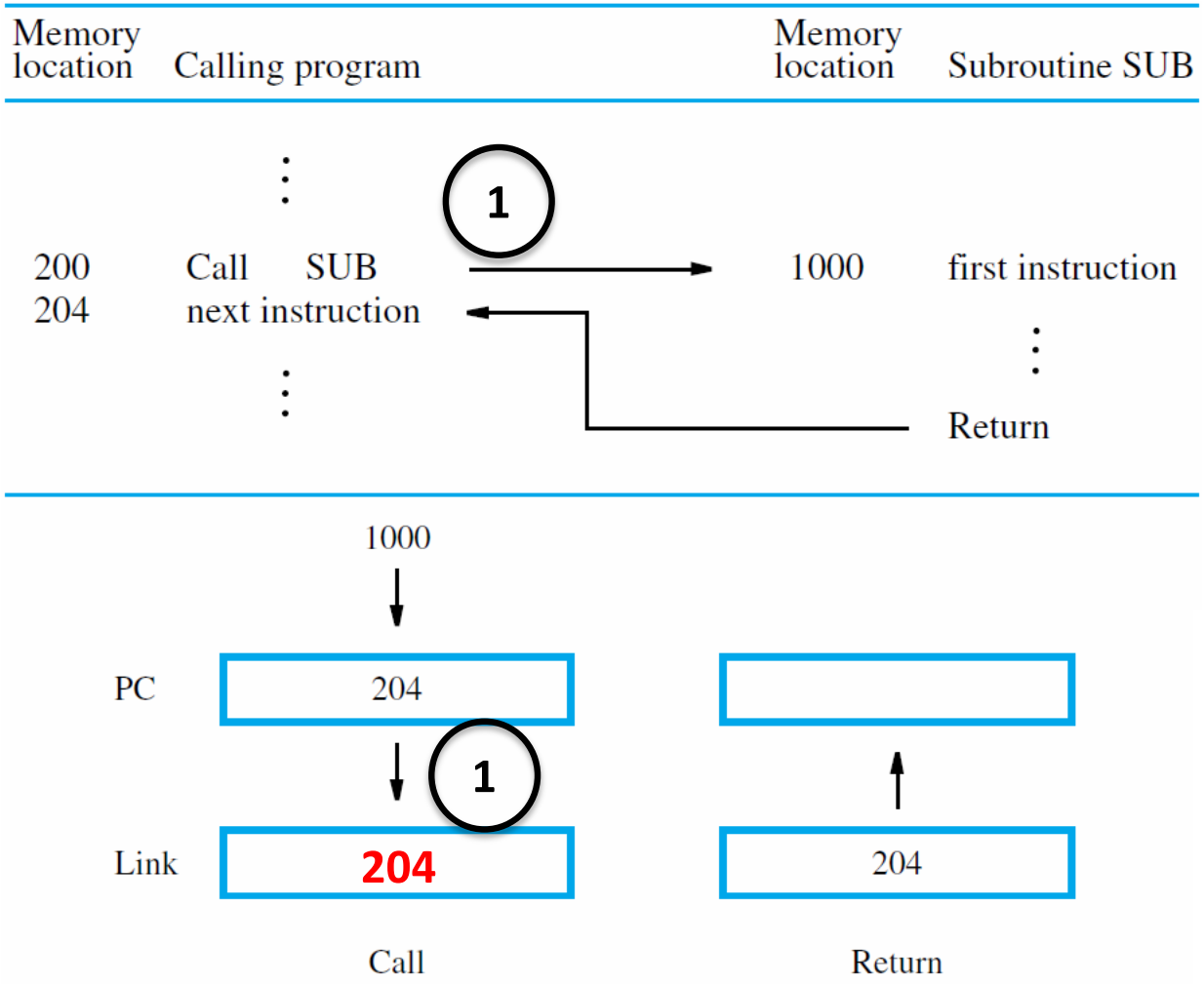
Repeated Task: Different Data



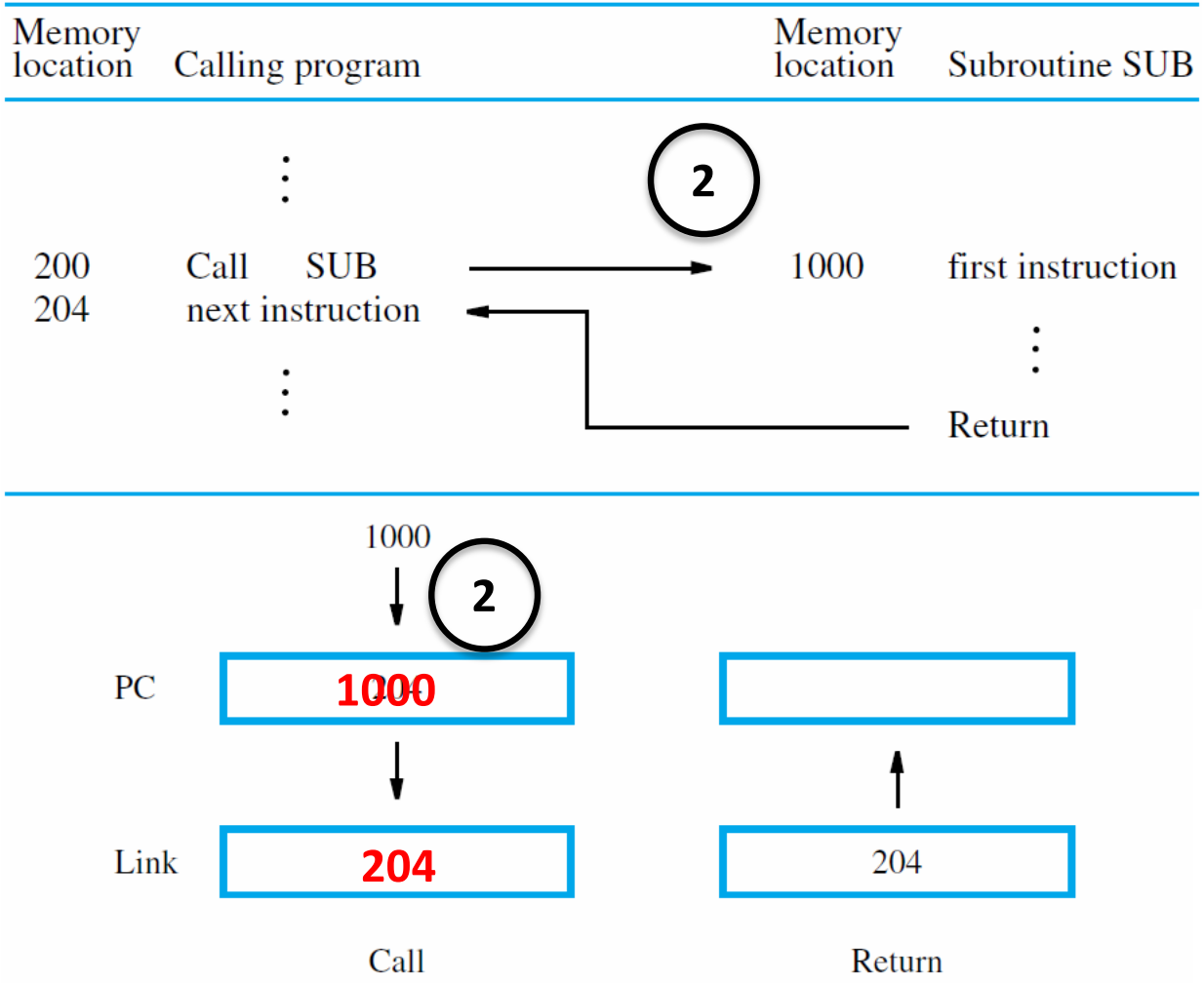
Special Branch



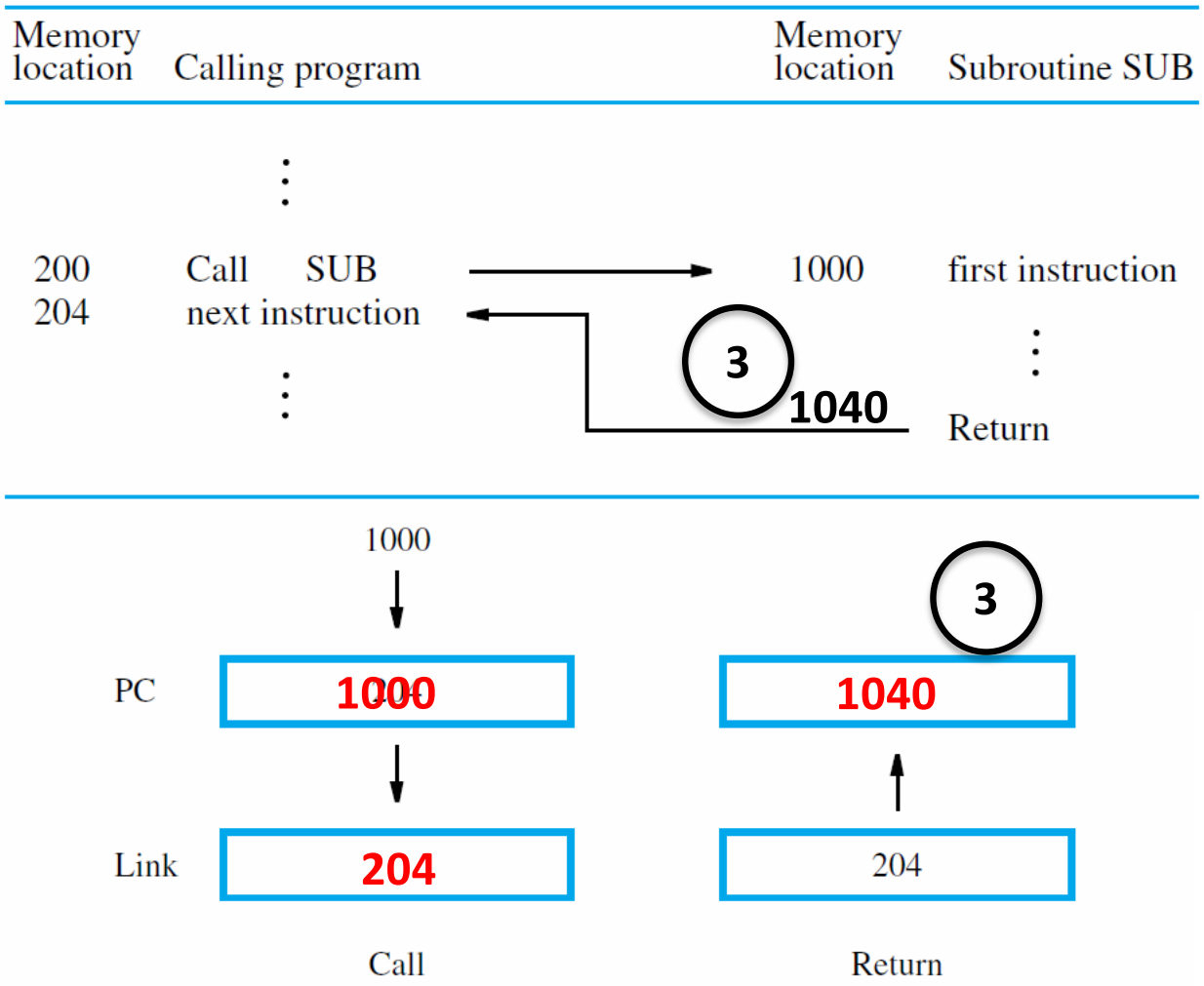
Caller and Callee



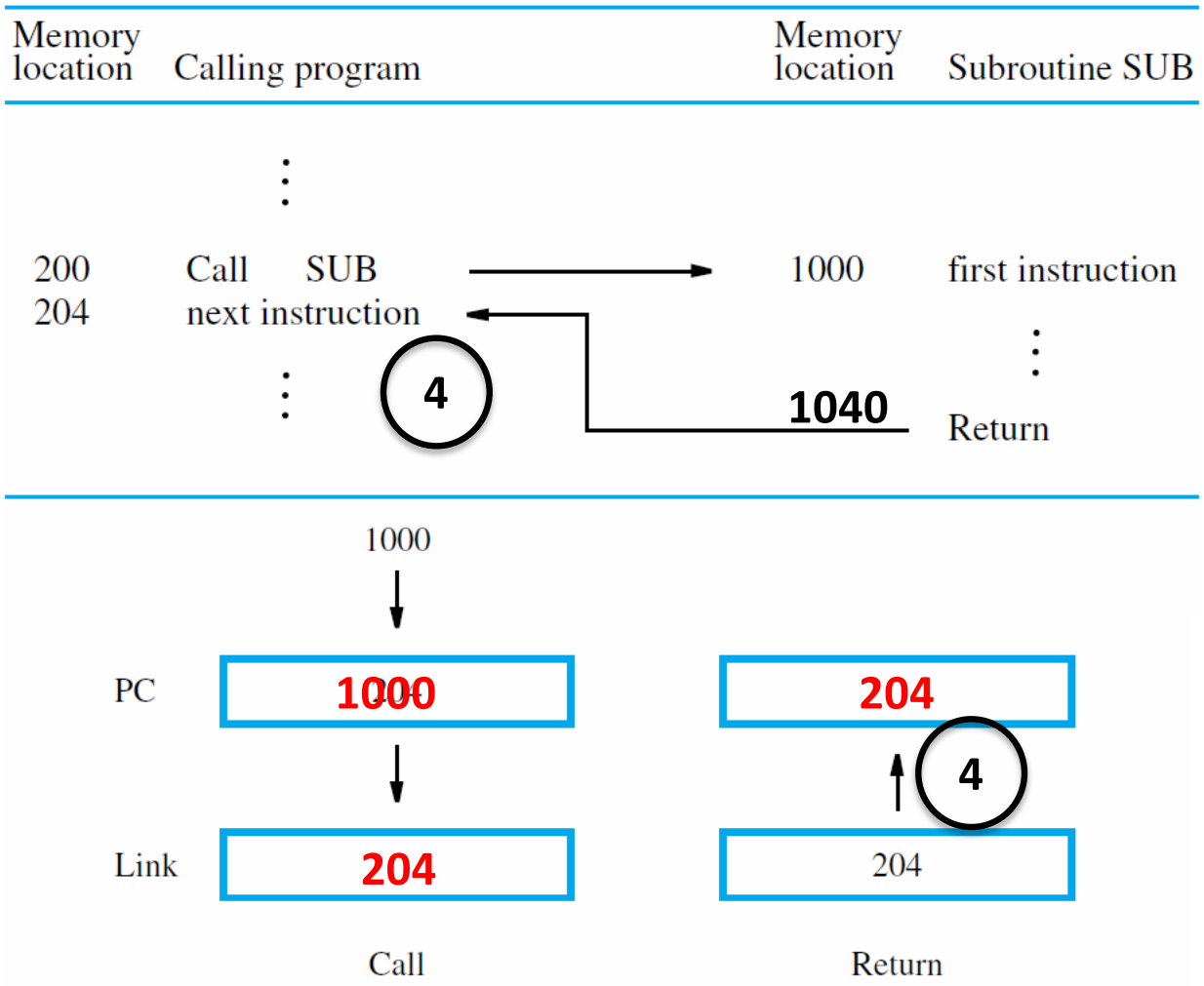
Caller and Callee



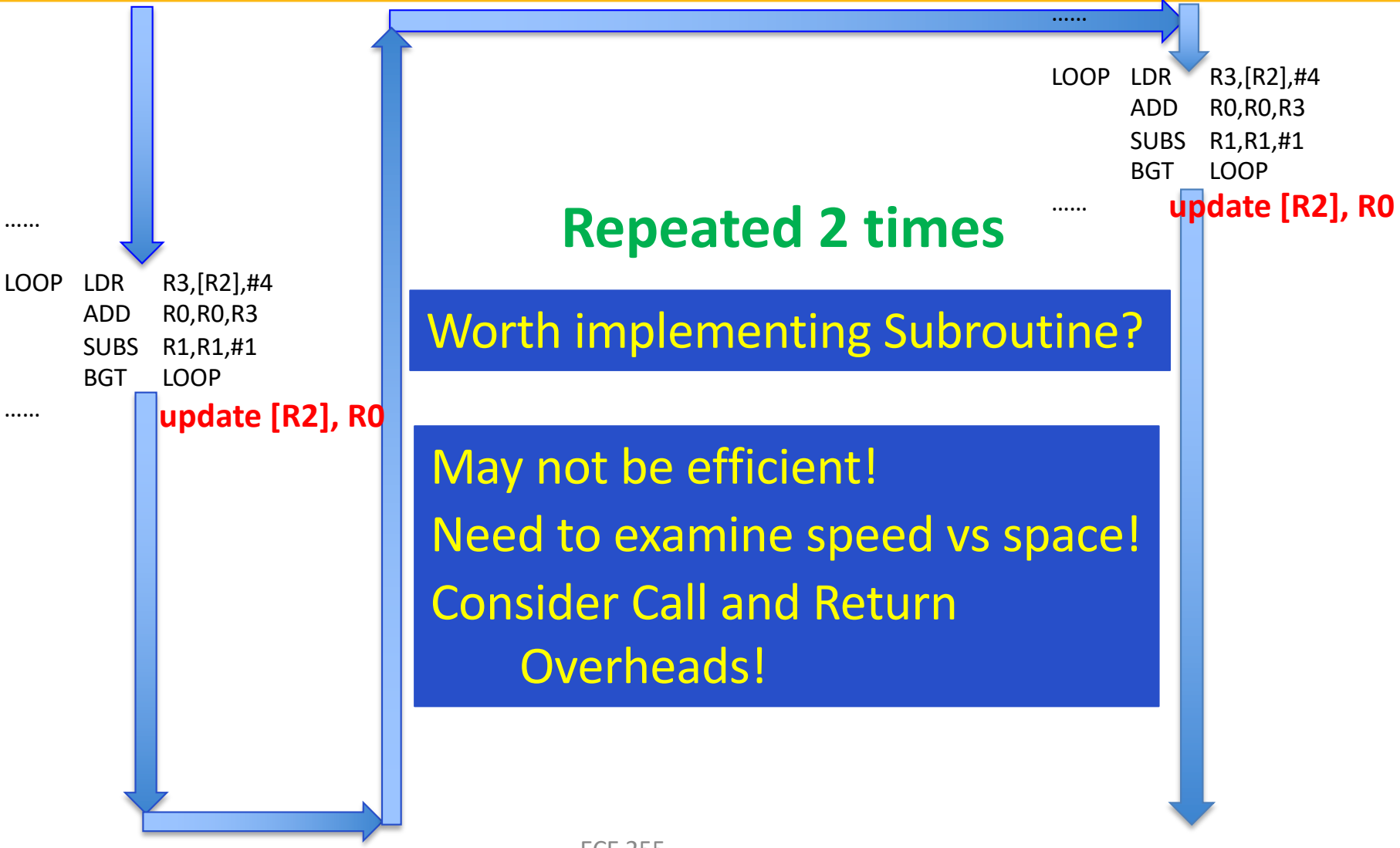
Caller and Callee



Caller and Callee



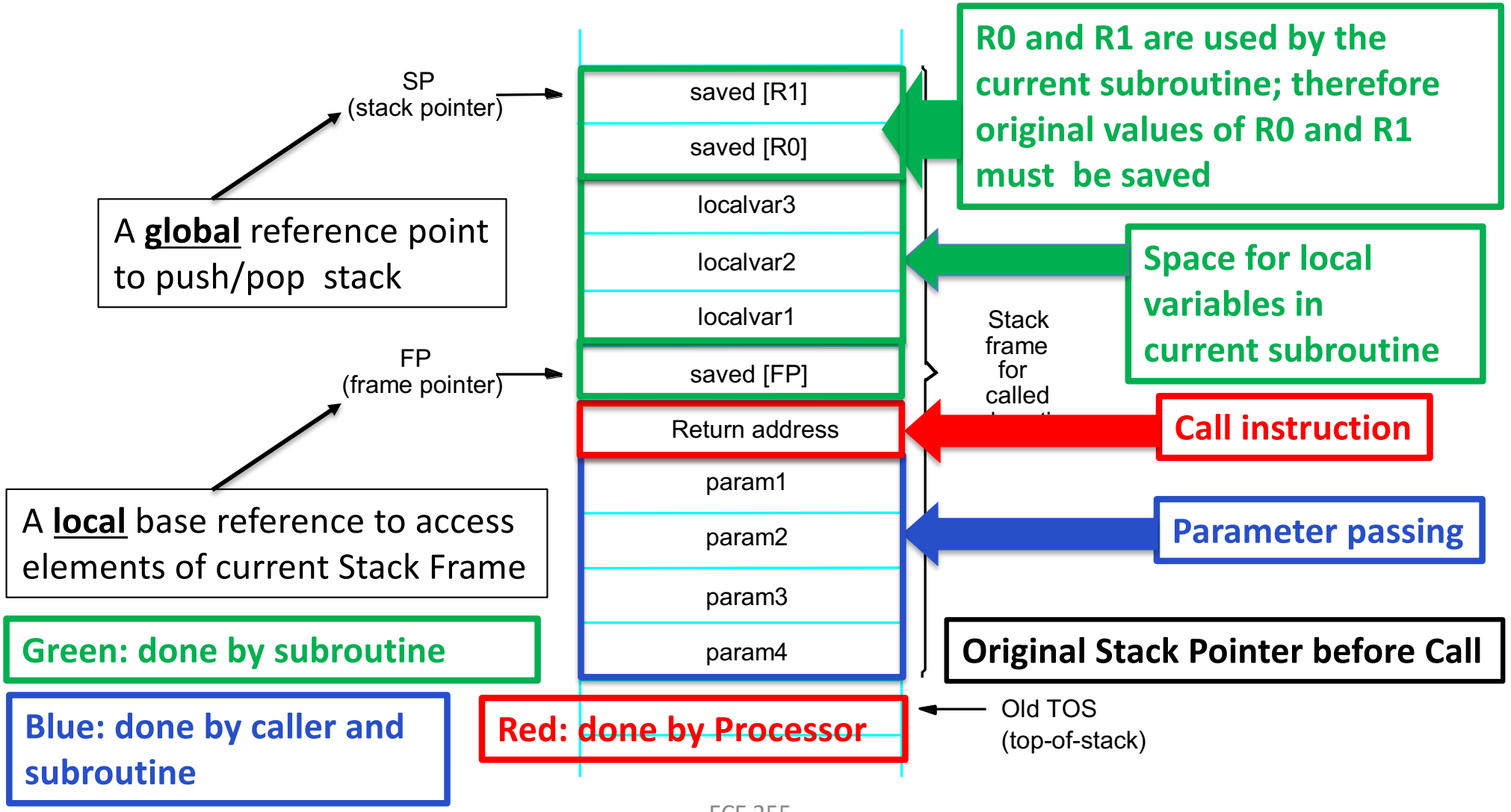
Repeated Task: Different Data



2.7.2 Parameter Passing

- **Parameter passing with Subroutine Call**:
  - Information exchange to/from a subroutine
  - Different input to subroutine, yield different results
- Use the stack for:
  1. Parameters
  2. Allocate space for local variables in subroutine
  3. Save registers before using them in subroutine
  4. Return address to caller

Fig 2.20 Stack Frame Layout



Call Process (re: Fig. 2.20)

[Ri] = Content of R_i

Caller:

1. Pass 4 parameters
   a. Push 4 parameters

Processor:

2. Call instruction executed
   a. Return address on stack (↑SP: stack pointer)

Subroutine:

3. FP is a (borrowed) GP register and saved on stack by
   a. Move FP,-(SP) ; full stack, so auto-decrement first
   b. Move SP,FP ; ↑SP = ↑FP = original value of FP
4. Allocate local variables space
   a. Subtract #12,SP ; simply move SP
5. Will use R1, R0 so save original on stack; ↑SP = old[R1]

Return Process (re: Fig. 2.20)

[Ri] = Content of R_i

Subroutine:

6. Before Return:
   a. Pop R0, R1 ; restore their original values
   b. Add #12,SP ; remove used local variables by moving SP
   c. Pop old[FP] back to FP ; restore FP original value

      ; ↑SP = return address

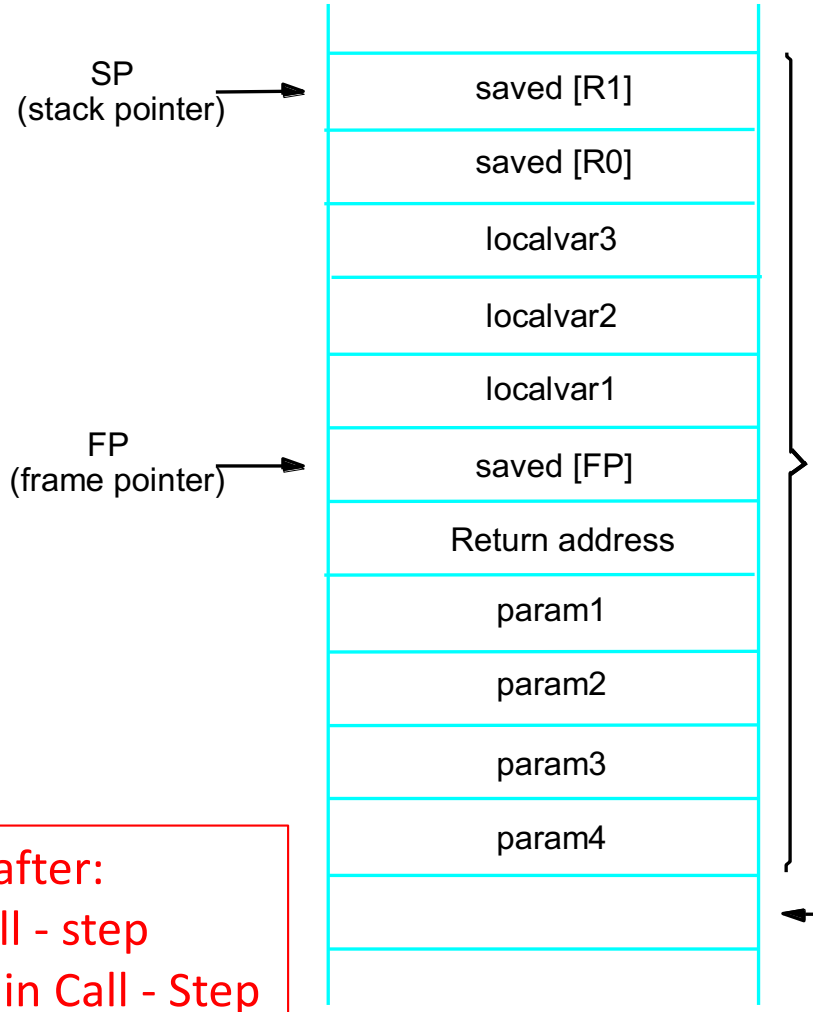
Processor:

7. Return instruction executed

   Caller:

   8. After Return:
      a. Pop result; ↑SP = old TOS

Fig. 2.20 Stack Layout-Call-A



[Ri] = Content of R_i

Caller:

1. Pass 4 parameters
   a. Push 4 parameters

Processor:

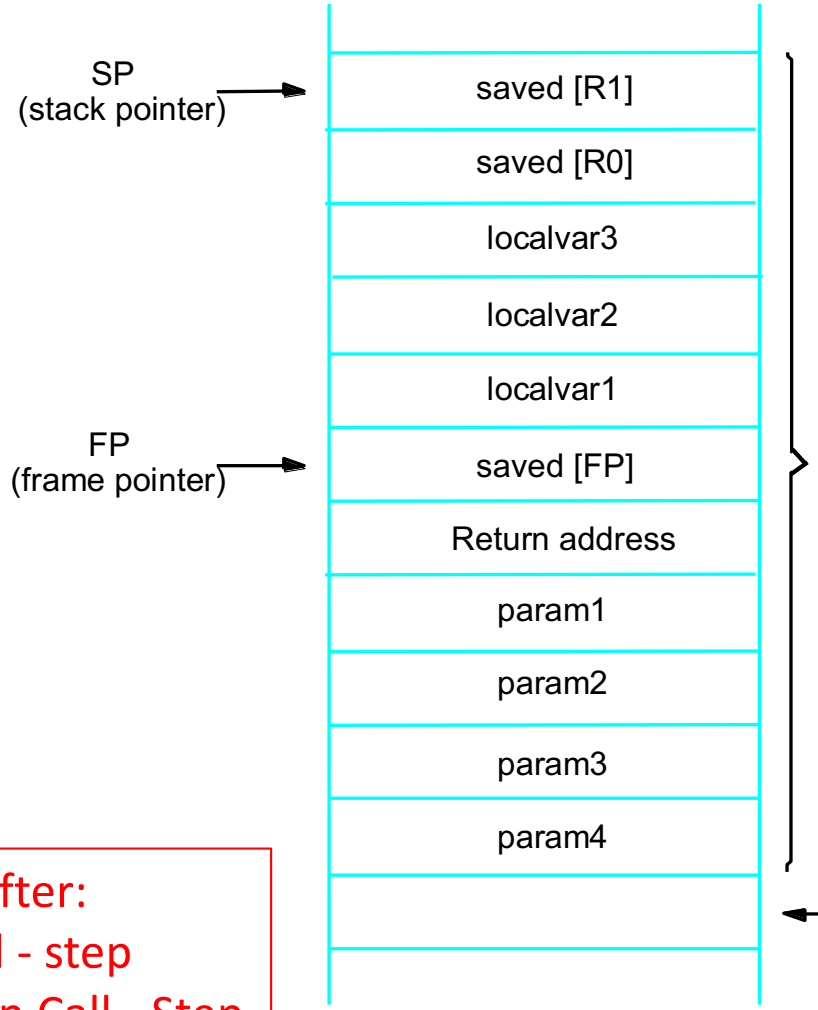
2. Call instruction executed
   a. Return address on stack (↑SP: stack pointer)

Subroutine:

3. FP is a (borrowed) GP register and saved on stack by
   a. Move FP,-(SP) ; full stack, so auto-decrement first
   b. Move SP,FP ; ↑SP = ↑FP = original value of FP
4. Allocate local variables space
   a. Subtract #12,SP ; simply move SP
5. Will use R1, R0 so save original on stack; ↑SP = old[R1]

—> Stack Pointer after:
CC-X = Caller in Call - step
SC-X = Subroutine in Call - Step

Fig. 2.20 Stack Layout-Call-B



Caller:

1. Pass 4 parameters
   a. Push 4 parameters

Processor:

2. Call instruction executed
   a. Return address on stack (↑SP: stack pointer)

Subroutine:

3. FP is a (borrowed) GP register and saved on stack by
   a. Move FP,-(SP) ; full stack, so auto-decrement first
   b. Move SP,FP ; ↑SP = ↑FP = original value of FP
4. Allocate local variables space
   a. Subtract #12,SP ; simply move SP
5. Will use R1, R0 so save original on stack; ↑SP = old[R1]

CC-1a

1. Pass 4 parameters
   a. Push 4 parameters

—> Stack Pointer after:
CC-X = Caller in Call - step
SC-X = Subroutine in Call - Step

Fig. 2.20 Stack Layout-Call-C

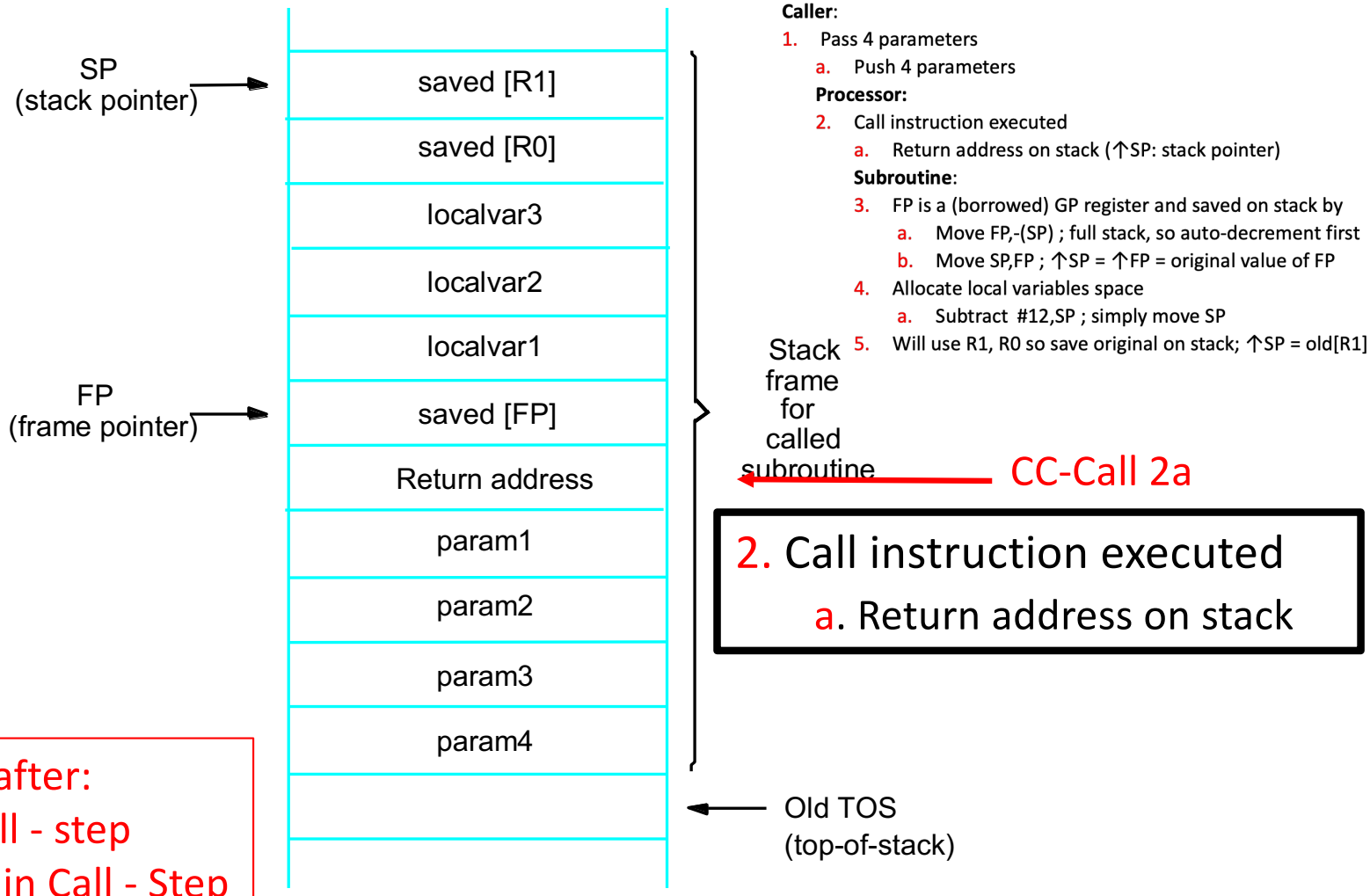


Fig. 2.20 Stack Layout-Call-D

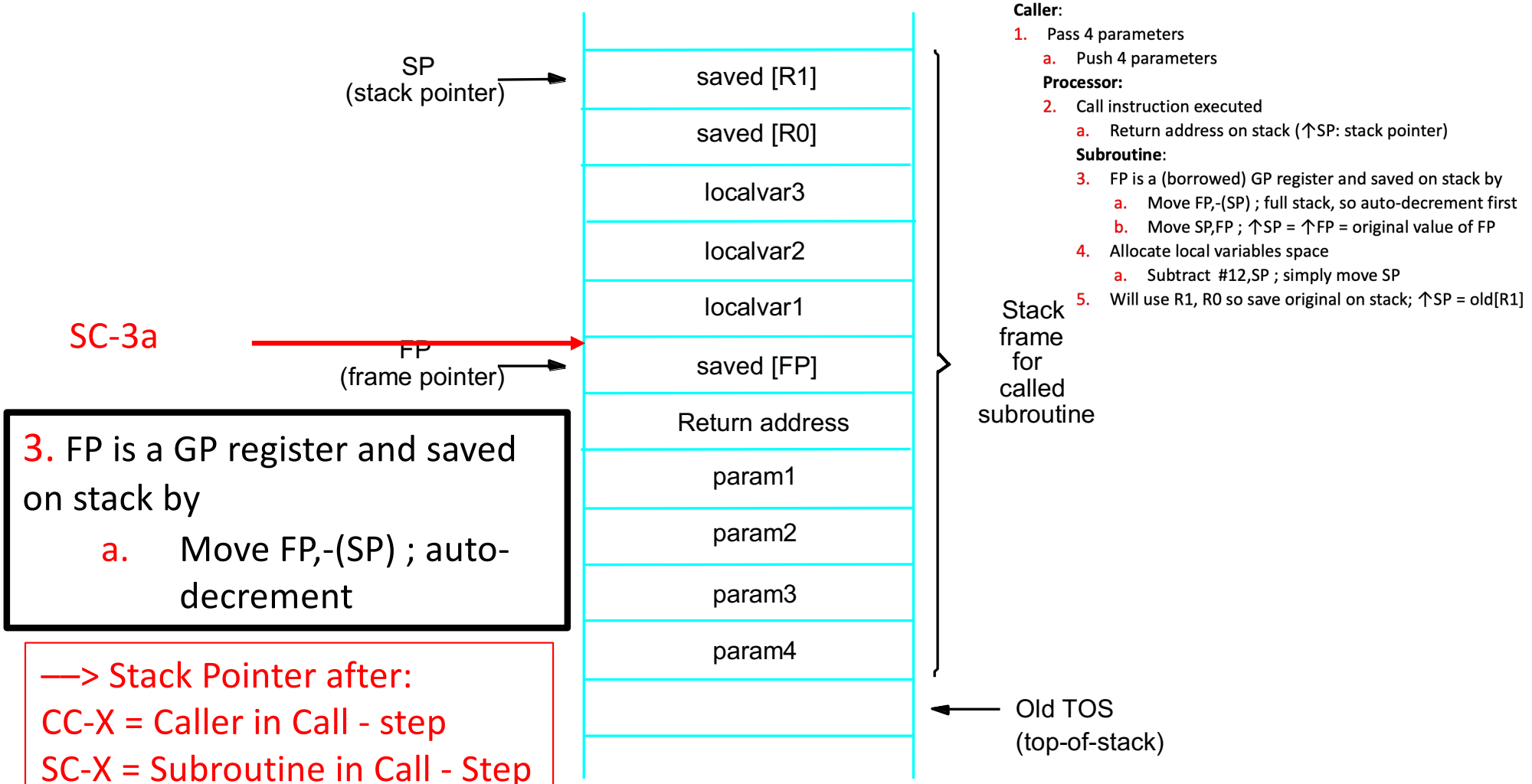


Fig. 2.20 Stack Layout-Call-E

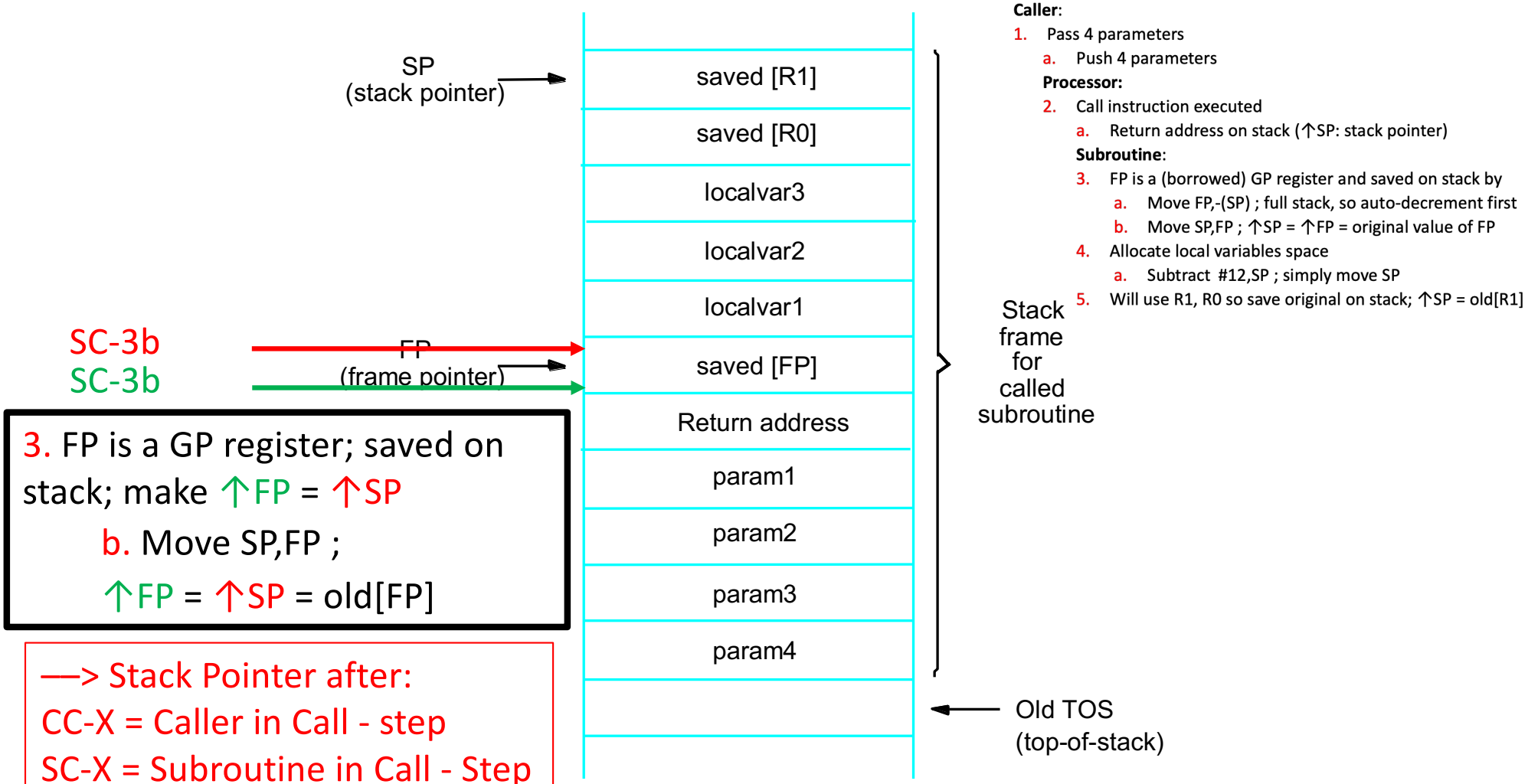


Fig. 2.20 Stack Layout-Call-F

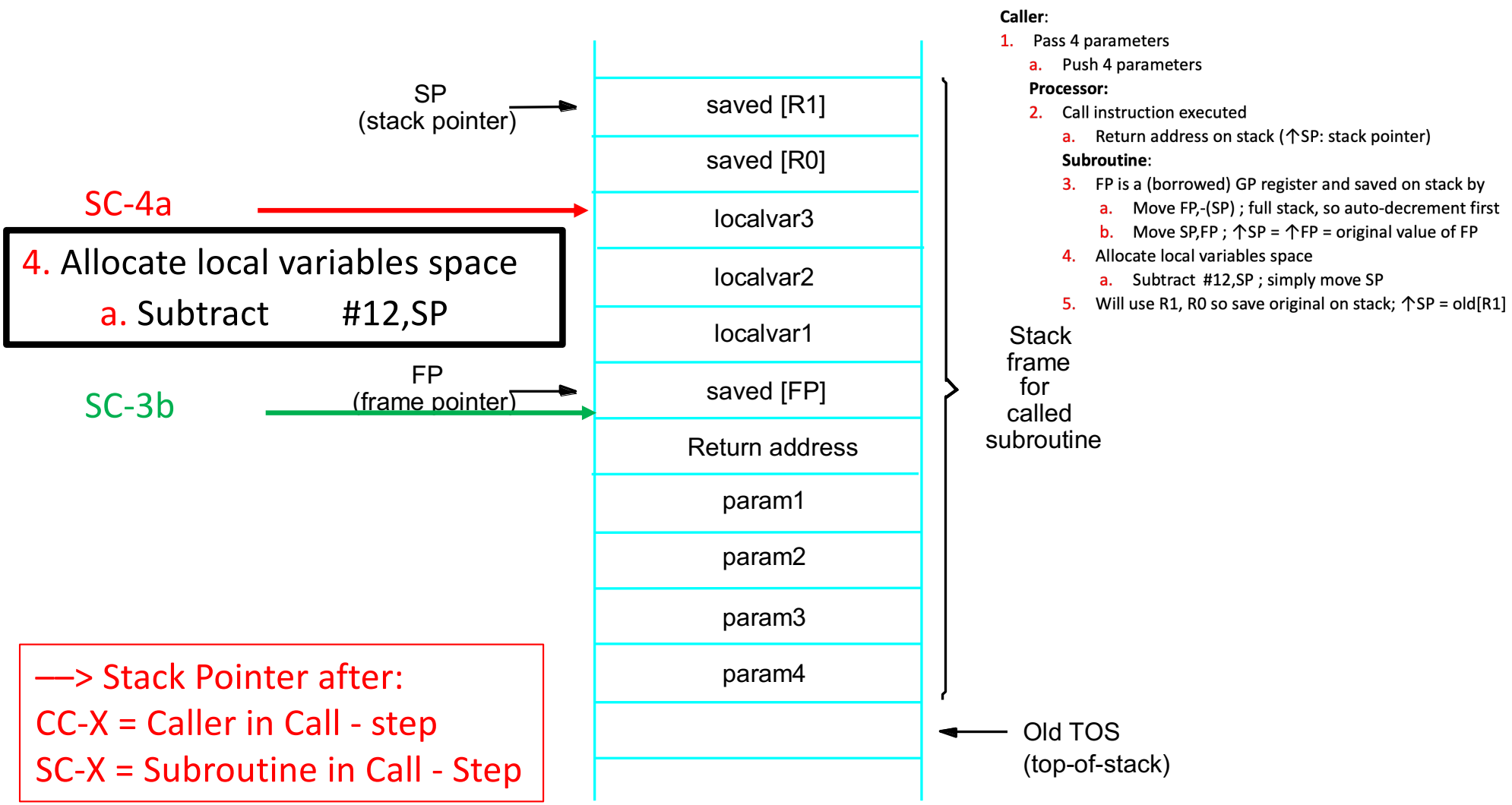


Fig. 2.20 Stack Layout-Call-G

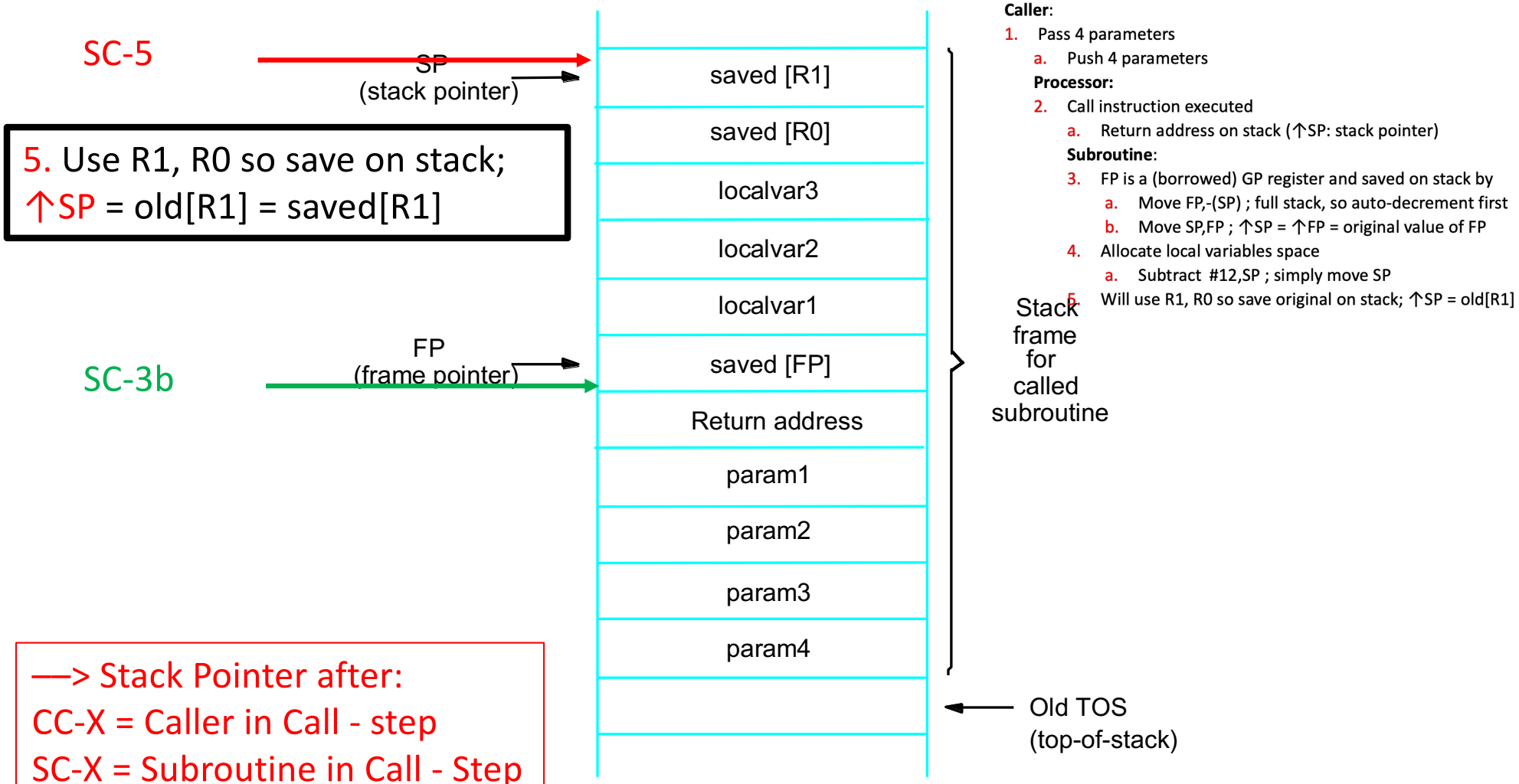


Fig. 2.20 Stack Layout-Return-A

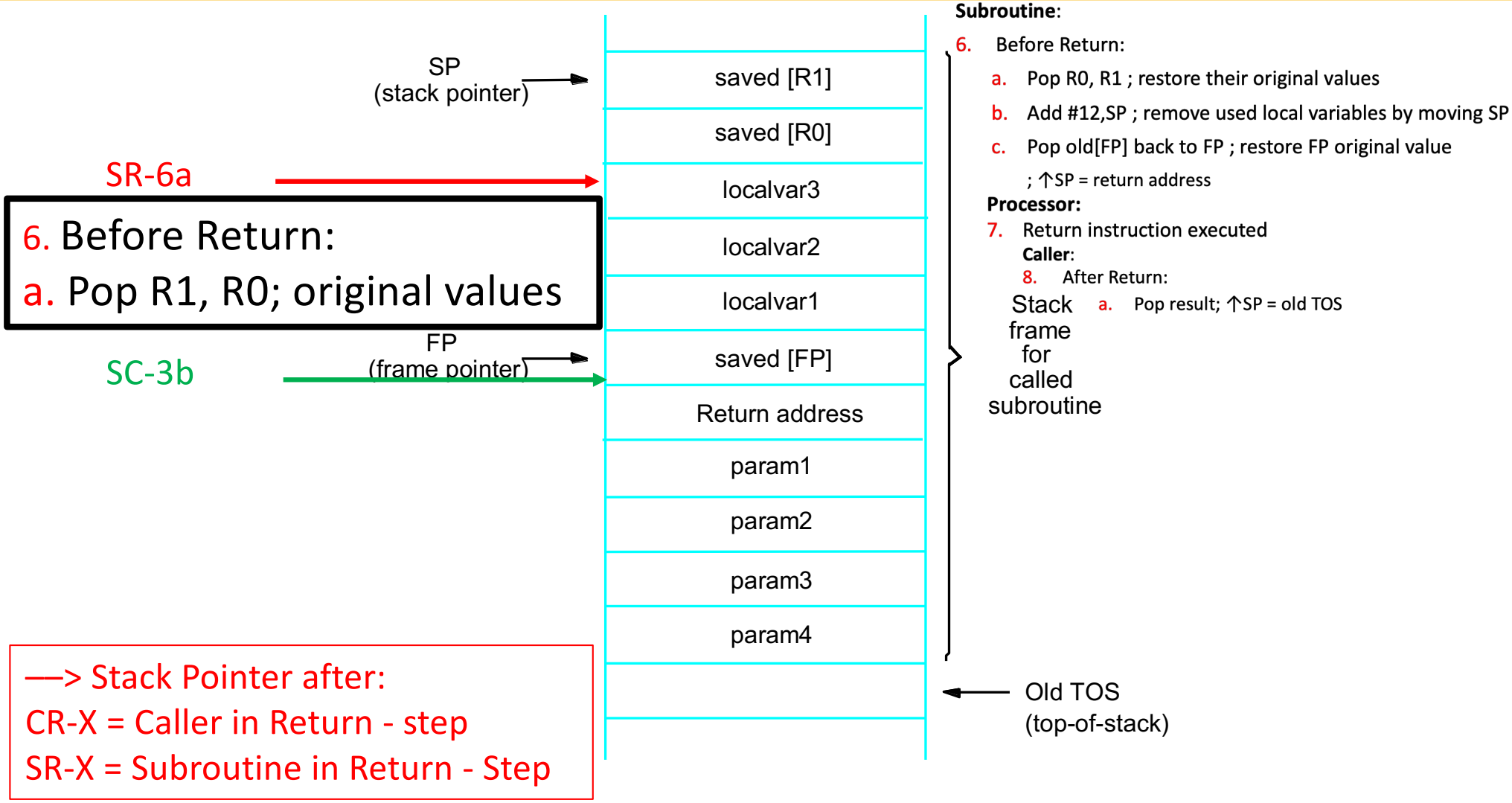


Fig. 2.20 Stack Layout-Return-B

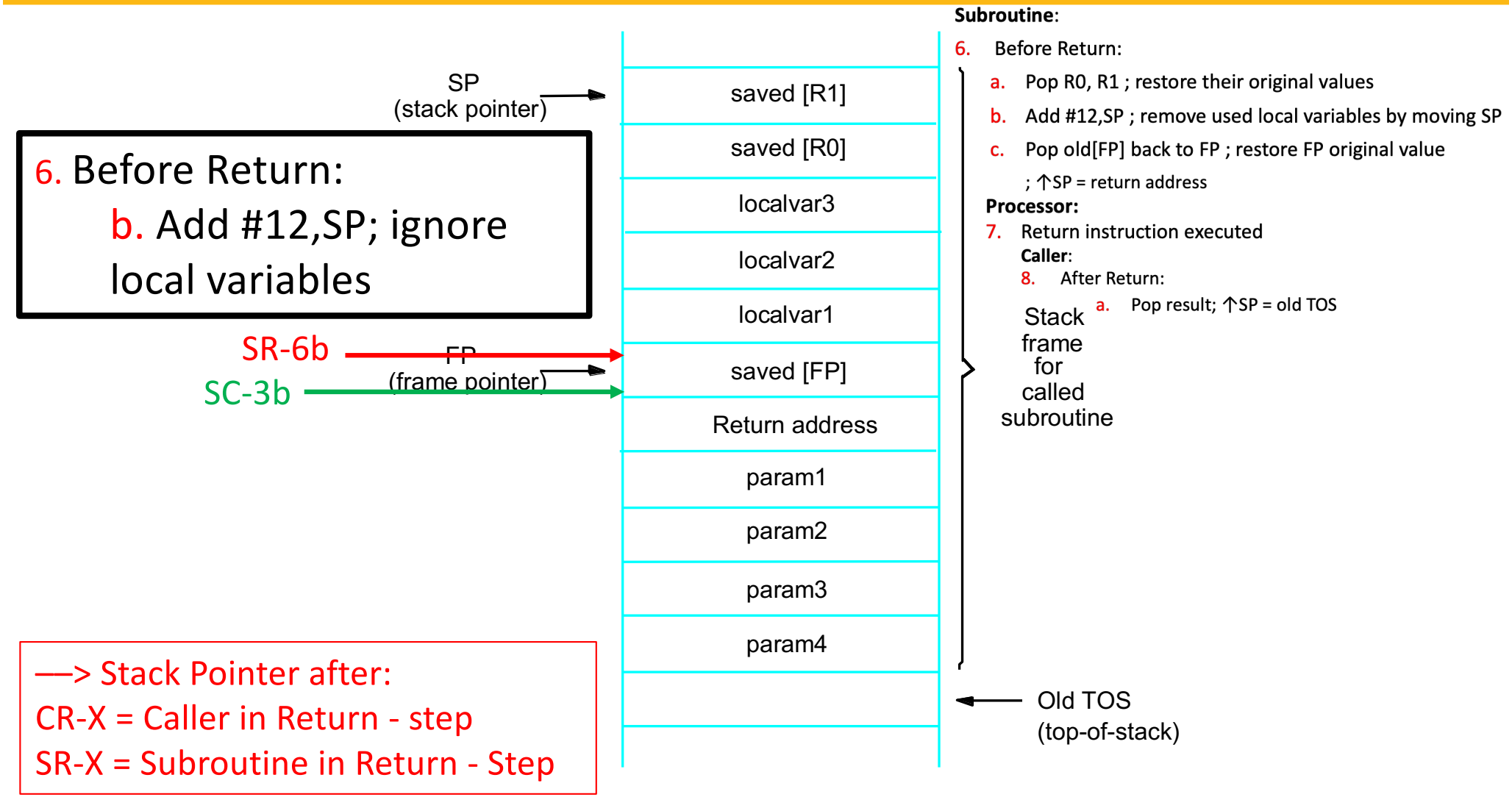


Fig. 2.20 Stack Layout-Return-C

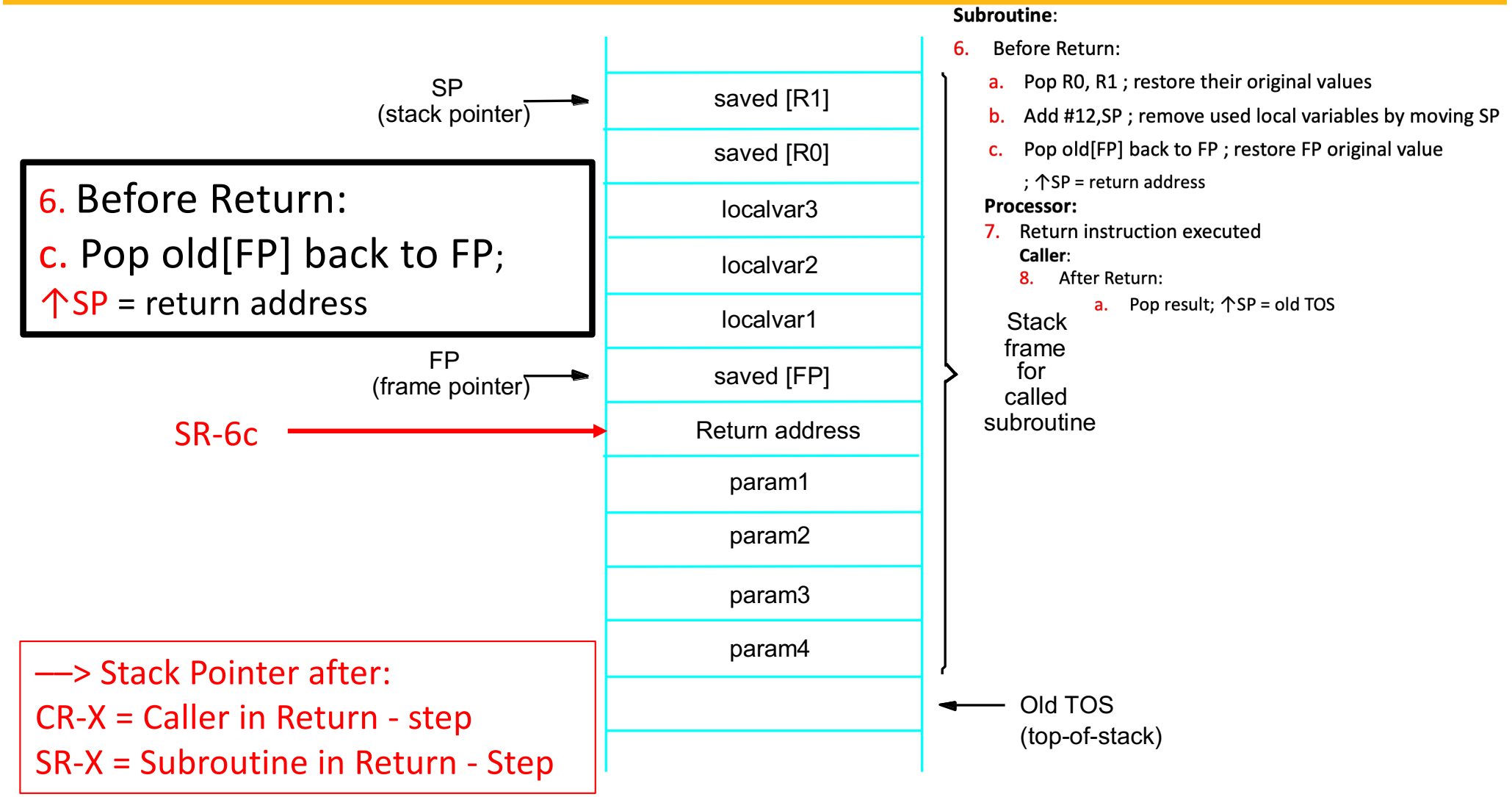


Fig. 2.20 Stack Layout-Return-D

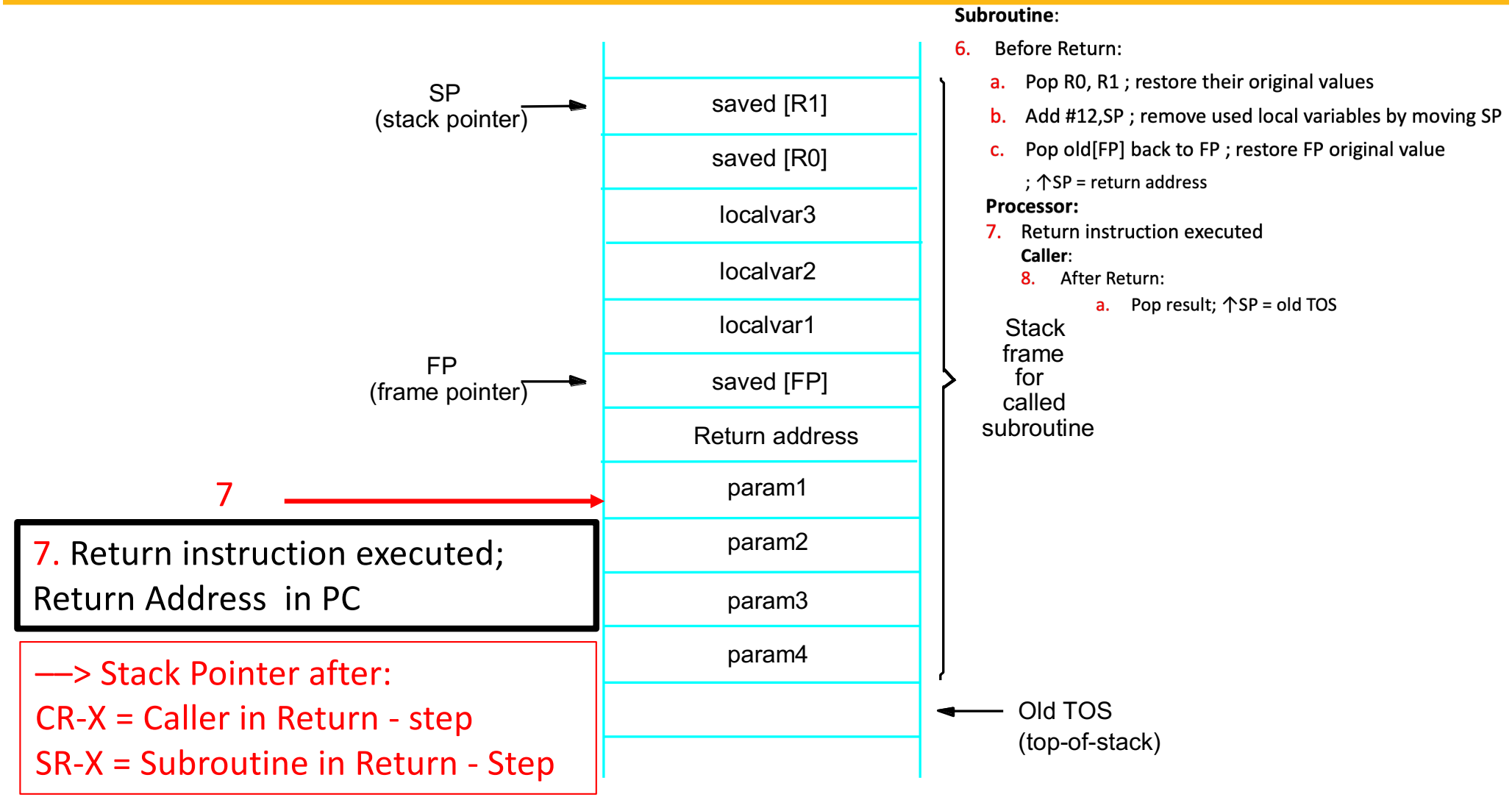
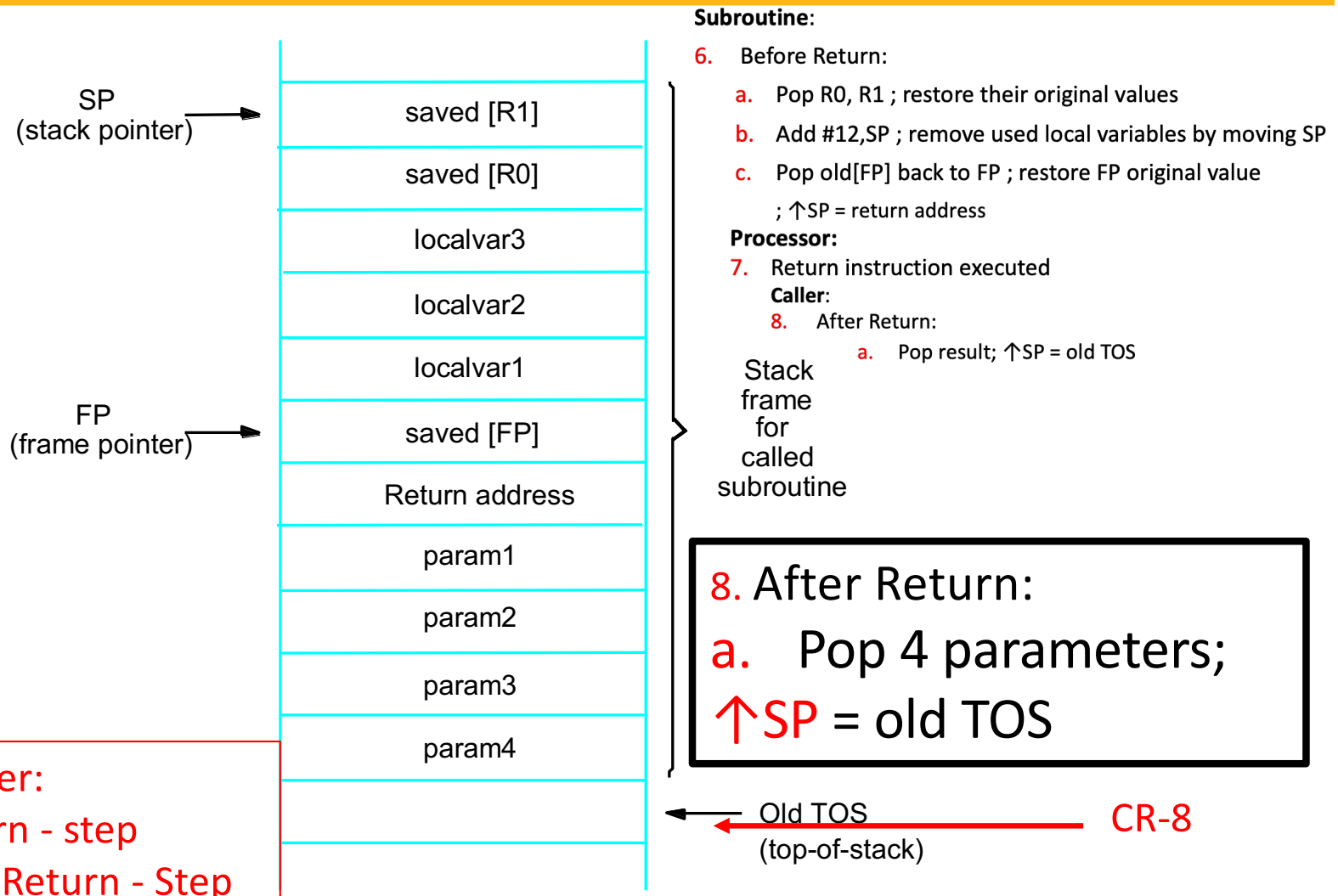


Fig. 2.20 Stack Layout-Return-E



—> Stack Pointer after:
CR-X = Caller in Return - step
SR-X = Subroutine in Return - Step

Fig. 2.20 Stack Frame Layout