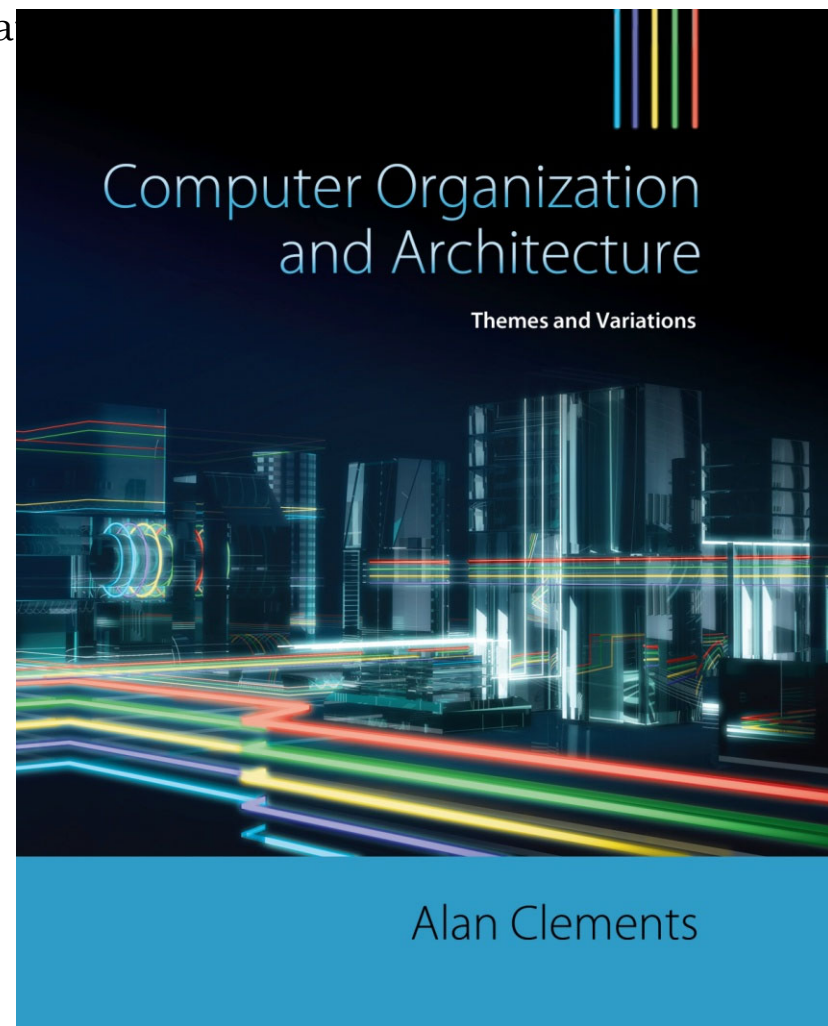


Part 2

CHAPTER 4

Computer Organization and Architecture

1



These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides are for personal use only.
Students must destroy these copies within 30 days after receipt of final course evaluations.

ARM's Subroutine Example with Stack Frame

```
AREA TestProg, CODE, READONLY
ENTRY      ;This is the calling environment.
           ;subroutine code is on the next slide.
```

Main	ADR	sp , Stack	;set up r13 as the stack pointer
	MOV	r0 , #124	;set up a dummy parameter in r0
	MOV	fp , #123	;set up dummy frame pointer
	<u>STR</u>	<u>r0, [sp, #-4]!</u>	;push the parameter
	BL	Sub	;call the subroutine
	LDR	r1 , [sp], #4	;pop the parameter
Loop	B	Loop	;wait here (endless loop)

Missing the post
update value
in page 237

Bold is not correct in page 237

ARM's Subroutine Example with Stack Frame

```

Sub  STMFD sp!, {fp,lr}    ;push frame-pointer and link-register
    MOV    fp, sp          ;frame pointer at the bottom of
                               ;the frame

    SUB     sp, sp, #4       ;create the stack frame (one word)
    LDR     r2, [fp, #8]      ;get the pushed parameter
    ADD     r2, r2, #120      ;do a dummy operation on
                               ;the parameter
    STR     r2, [fp, #-4]    ;store it in the stack frame
    ADD     sp, sp, #4       ;clean up the stack frame
    LDMFD   sp!, {fp, pc}  ;restore frame pointer and return

    DCD     0x0000            ;clear memory
    DCD     0x0000
    DCD     0x0000
    DCD     0x0000
Stack DCD   0x0000            ;start of the stack

```

Bold is not correct in page 238

END

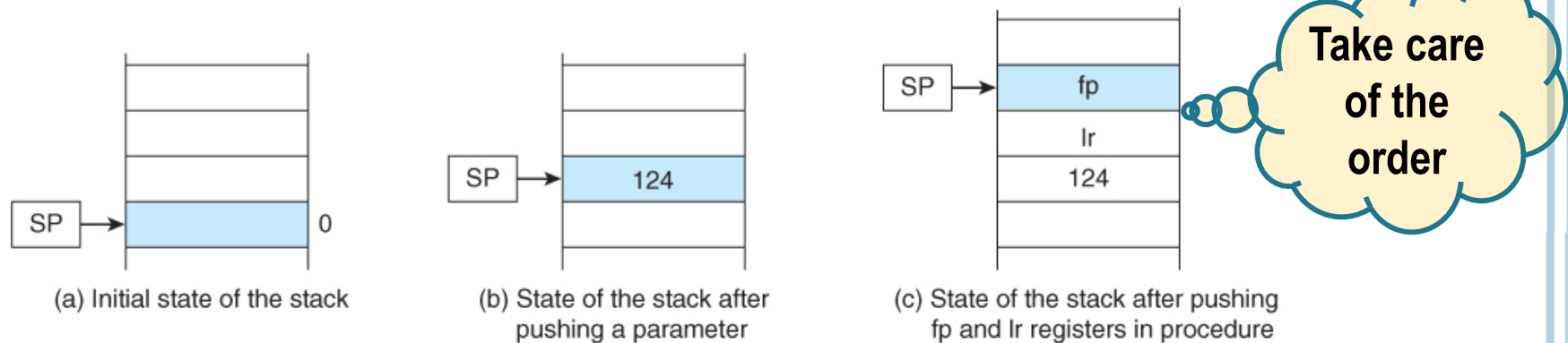
**R2 has been changed inside the subroutine.
Hence, it should be saved at the beginning of the
subroutine and restored at the end.**

ARM's Subroutine Example with Stack Frame

- Figure 4.6 demonstrates the behavior of the stack during the code's execution. Figure 4.6a depicts the stack's initial state. In Figure 4.6b the parameter has been pushed onto the stack. In Figure 4.6c the frame pointer and link register have been stacked by `STMFD sp!, {fp, lr}`.

FIGURE 4.6

The behavior of the stack during the execution of the code

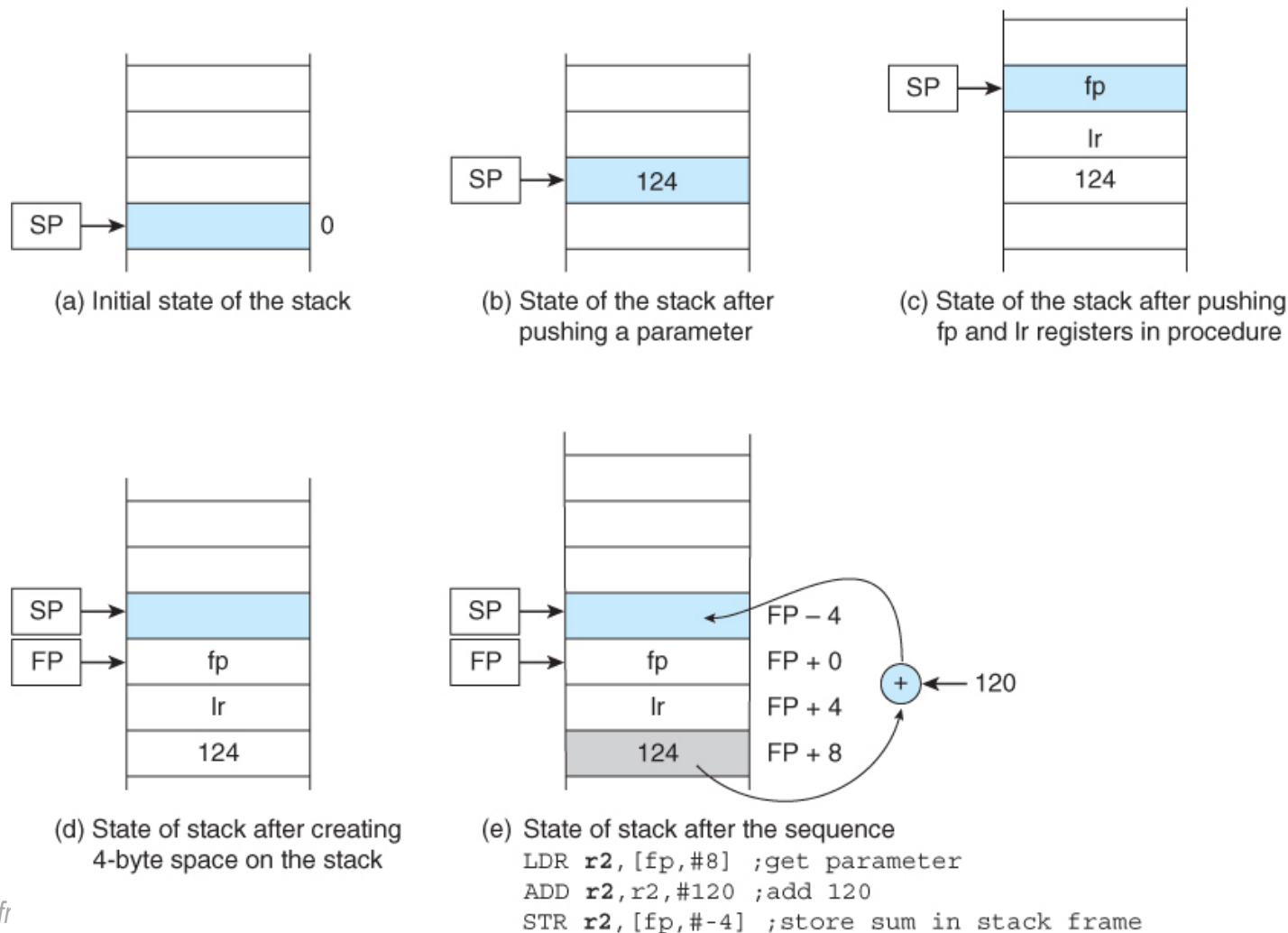


ARM's Subroutine Example with Stack Frame

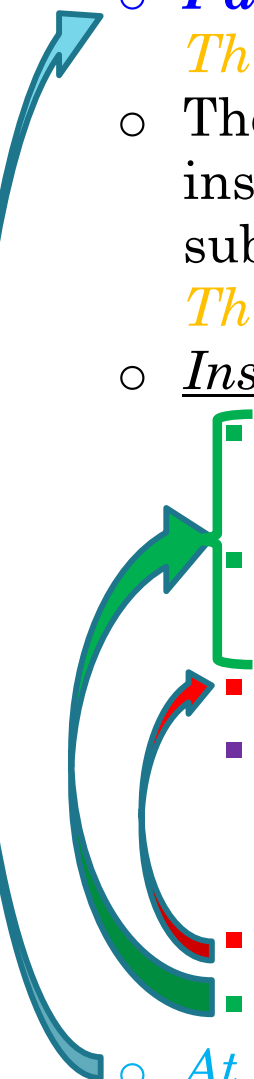
- ❑ In Figure 4.6d a 4-byte word has been created at the top of the stack. Finally, Figure 4.6e demonstrates how the pushed parameter is accessed and moved to the new stack frame using register indirect addressing with the frame pointer.

FIGURE 4.6

The behavior of the stack during the execution of the code



Calling a Subroutine Step-by-Step

- ❑ To call a subroutine, the following steps need to be performed:
 - **Parameters** need to be *passed* from the caller to the subroutine.
This can be performed via the stack.
 - The **address** of the instruction immediately after the calling instruction needs to be *saved in a safe place BEFORE* branching to the subroutine.
This can be performed by using BL instruction or via the stack, or both.
 - Inside the subroutine, we need to:
 - Push the values of all registers to be used inside the subroutine, as well as the FP (R11) and LR (R14).
 - Make the FP (R11) point to the bottom of the frame by copying the value of the SP (R13) to the FP (R11).
 - Create a space inside the stack for local variables.
 - Perform the subroutine instructions.
The addresses of parameters and local variables are calculated relative to the value of the FP (R11).
 - At the end of the subroutine, deallocate all created local variables.
 - Pop all pushed registers but use PC (R15) instead of LR (R14).
 - At the caller program, all pushed parameters need to be popped.
- 

Passing Parameters via the Stack

- ❑ You can pass a parameter to a subroutine
 - *by value*
 - *by reference*

- ❑ When passed *by value*, the subroutine receives a copy of the parameter.
 - Passing a parameter by value causes the *parameter to be cloned* and the *cloned version of the parameter* to be used by the subroutine.
 - If the parameter is modified by the subroutine, the new value does not affect the value of the parameter elsewhere in the program.

- ❑ When passed *by reference*, the subroutine receives a pointer, (i.e., an address) to the parameter.
 - *There is only one copy of the parameter*, and the subroutine can access this value because it knows the address of the parameter.
 - If the subroutine modifies the parameter, it is modified the original value.

Passing Parameters via the Stack

- ❑ The subroutine `swap(int a, int b)` *intends* to exchange two values.
- ❑ Let's examine how parameters are passed to this subroutine.

```
void swap(int a, int b)  /* swaps the value of a and b */
{
    int temp;
    temp = a;             /* copy a    to temp    */
    a     = b;            /* copy b    to a,    and */
    b     = temp;         /* copy temp to b      */
}
```

```
void main(void)
{
    int x = 2, y = 3;
    swap (x, y);          /* swap a and b */
}
```



Will it work?

Passing Parameters via the Stack

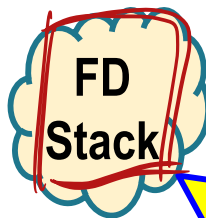
```
AREA SwapVal, CODE, READONLY
```

```
ENTRY
```

```
ADR    sp, STACK          ;set up stack pointer
MOV    fp, #0xFFFFFFFF    ;set up dummy fp for tracing
B       main                ;jump to the function main
```

```
SPACE 0x20
```

```
STACK DCD 0
```



You need to re-do it yourself using the other stack types.

```
; void swap (int a, int b)
; Parameter a is at [fp]+4
; Parameter b is at [fp]+8
; Variable temp is at [fp]-4
```

FD
Stack

Passing Parameters via the Stack

You need to re-do it yourself using the other stack types.

```

; {
swap SUB    sp, sp, #4      ; Create stack frame: decrement sp
    STR    fp, [sp]        ; push the frame pointer onto the stack
    MOV    fp, sp          ; frame pointer points at the base
;   int temp;
    SUB    sp, sp, #4      ; move sp up 4 bytes for temp
;   temp = a;
    LDR    r0, [fp, #4]     ; get parameter a from the stack
    STR    r0, [fp, #-4]    ; copy a to temp onto the stack frame
;   a = b;
    LDR    r0, [fp, #8]     ; get parameter b from the stack
    STR    r0, [fp, #4]     ; copy b to a
;   b = temp;
    LDR    r0, [fp, #-4]    ; get temp from the stack frame
    STR    r0, [fp, #8]     ; copy temp to b
; }

    MOV    sp, fp          ; Collapse stack frame created for swap
    LDR    fp, [sp]        ; restore the stack pointer
    ADD    sp, sp, #4      ; restore old frame pointer from stack
    MOV    pc, lr          ; move stack pointer down 4 bytes
                           ; return by loading LR into PC

```

Passing Parameters via the Stack

```

; void main(void)
; {
main
SUB    sp, sp, #4      ;Create stack frame in main for x, y
STR    fp, [sp].       ;move the stack pointer up
MOV    fp, sp          ;push the frame pointer onto the stack
; the frame pointer points at the base;
; int x = 2, y = 3;
SUB    sp, sp, #8      ;move sp up 8 bytes for 2 integers
MOV    r0, #2          ;x = 2
STR    r0, [fp, #-4]   ;put x in stack frame
MOV    r0, #3          ;y = 3
STR    r0, [fp, #-8]   ;put y in stack frame
; swap(x, y);
LDR    r0, [fp, #-8]   ;get y from stack frame
STR    r0, [sp, #-4]!  ;push y on stack
LDR    r0, [fp, #-4]   ;get x from stack frame
STR    r0, [sp, #-4]!  ;push x on stack
BL     swap            ;call swap, save return address in LR
ADD    sp, sp, #8      ;Clean the stack from the parameters
; }
MOV    sp, fp          ;restore the stack pointer
LDR    fp, [sp]         ;restore old frame pointer from stack
ADD    sp, sp, #4      ;move stack pointer down 4 bytes
Loop B   Loop
END

```

Bold is not correct in page 244

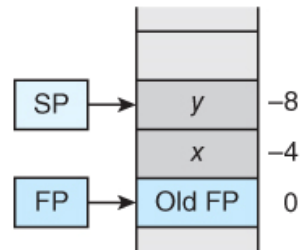
Passing Parameters via the Stack

- ❑ This code swaps the variables inside the stack frame
- ❑ When the return is made, the stack frame will be collapsed, and the effect of the swap will be lost.
- ❑ The variables in the calling environment are not affected.

Passing Parameters via the Stack

FIGURE 4.8

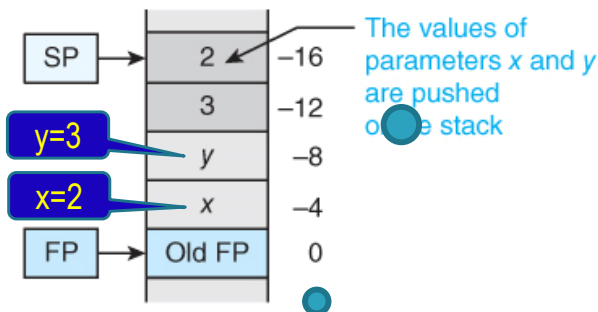
Passing values to a subroutine by value



(a) State of the stack in main after creating stack frame with:

```
SUB sp, sp, #4
STR fp, [sp]
MOV fp, sp
SUB sp, sp, #8
```

FP not SP.
Not correct in
page 245

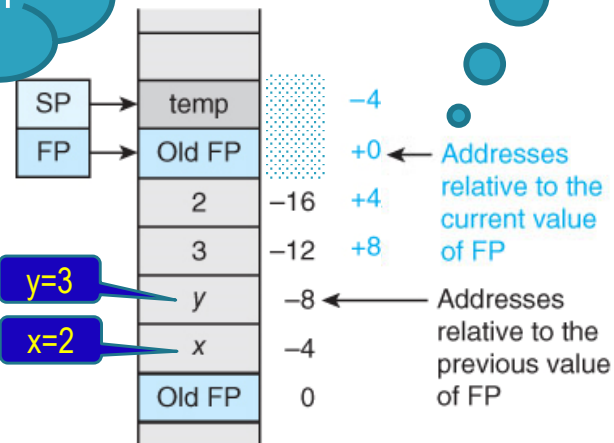


(b) The stack in main after putting two parameters in the stack frame with:

```
MOV r0, #2
STR r0, [fp, #-4]
MOV r0, #3
STR r0, [fp, #-8]
```

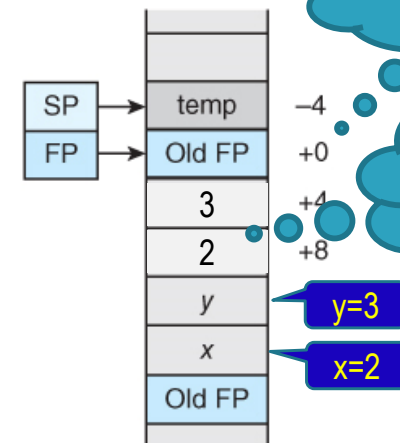
Then pushing two parameters on the stack

```
LDR r0, [fp, #-8]
STR r0, [sp, #-4]!
LDR r0, [fp, #-4]
STR r0, [sp, #-4]!
```



(c) The stack after the creation of a stack frame in swap. The new stack frame is four bytes deep and holds the variable temp. The frame is created by:

```
SUB sp, sp, #4
STR fp, [sp]
MOV fp, sp
SUB sp, sp, #4
```



(d) The stack after executing the body of swap. Note that all data is referenced to FP.

```
LDR r0, [fp, #4]
STR r0, [fp, #-4]
LDR r0, [fp, #8]
STR r0, [fp, #4]
LDR r0, [fp, #-4]
STR r0, [fp, #8]
```

The location of the addresses relative to the FP needs to be shifted down one cell. It is not correct in page 245

Same thing

The values of the parameters are not correct in page 245

Passing Parameters via the Stack

❑ In the next example, we pass parameters by reference

```
void swap(int *a, int *b)    /* A function to swap two parameters
                               in calling program          */
{
    int temp;                /* copy *a    to temp          */
    temp = *a;                /* copy *b    to *a,  and */
    *a    = *b;                /* copy temp to *b          */
    *b    = temp;
}

void main(void)
{
    int x = 2, y = 3;
    swap(&x, &y);              /* call swap and pass
                               addresses of parameters */
}
```

Passing Parameters via the Stack

```
AREA SwapVal, CODE, READONLY
```

```
ENTRY
```

```
ADR    sp, STACK           ;set up stack pointer
MOV    fp, #0xFFFFFFFF      ;set up dummy fp for tracing
B      main                 ;jump to main function
```

```
SPACE 0x20
```

```
STACK DCD 0
```

```
; void swap (int *a, int *b)
; Parameter *a is at [fp]+4
; Parameter *b is at [fp]+8
; Variable temp is at [fp]-4
```


Passing Parameters via the Stack

```

; {
swap SUB    sp, sp, #4      ;Create stack frame: decrement sp
STR     fp, [sp]           ;push the frame pointer onto the stack
MOV     fp, sp             ;frame pointer points at the base
;   int temp;
SUB     sp, sp, #4         ;move sp up 4 bytes for temp
;   temp = *a;
LDR     r1, [fp, #4]       ;get address of parameter a
LDR     r2, [r1]           ;get value of parameter a (i.e., *a)
STR     r2, [fp, #-4]      ;store *a in temp in stack frame
;   *a = *b;
LDR     r0, [fp, #8]       ;get address of parameter b
LDR     r3, [r0]           ;get value of parameter b (i.e., *b)
STR     r3, [r1]           ;store *b in *a
;   *b = temp;
LDR     r3, [fp, #-4]      ;get temp
STR     r3, [r0]           ;store temp in *b
; }

MOV     sp, fp             ;Collapse stack frame created for swap
LDR     fp, [sp]           ;restore the stack pointer
ADD     sp, sp, #4         ;restore old frame pointer from stack
MOV     pc, lr             ;move stack pointer down 4 bytes
                                ;return by loading LR into PC

```

Missing the *
in page 247

Passing Parameters via the Stack

```

; void main(void)
; {
main
SUB    sp, sp, #4      ;Create stack frame in main for x, y
STR    fp, [sp].       ;move the stack pointer up
MOV    fp, sp          ;push the frame pointer onto the stack
; int x = 2, y = 3;    ;the frame pointer points at the base;
SUB    sp, sp, #8      ;move sp up 8 bytes for 2 integers
MOV    r0, #2          ;x = 2
STR    r0, [fp, #-4]   ;put x in stack frame
MOV    r0, #3          ;y = 3
STR    r0, [fp, #-8]   ;put y in stack frame
; swap(&x, &y);
SUB    r0, fp, #8      ;get address of y in stack frame
STR    r0, [sp, #-4]!  ;push address of y on stack
SUB    r0, fp, #4      ;get address of x in stack frame
STR    r0, [sp, #-4]!  ;push address of x on stack
BL     swap            ;call swap, save return address in LR
ADD    sp, sp, #8      ;Clean the stack from the parameters
; }
MOV    sp, fp          ;restore the stack pointer
LDR    fp, [sp]         ;restore old frame pointer from stack
ADD    sp, sp, #4      ;move stack pointer down 4 bytes
Loop B   Loop          ;Stop
END

```

Bold is not correct in page 244

Passing Parameters via the Stack

- ❑ In the function main, the addresses of the *parameters are pushed onto the stack* by means of the following instructions:

```
SUB    r0, fp, #8      ;get address of y in stack frame
STR    r0, [sp, #-4]!  ;push address of y on stack
SUB    r0, fp, #4      ;get address of x in stack frame
STR    r0, [sp, #-4]!  ;push address of x on stack
```

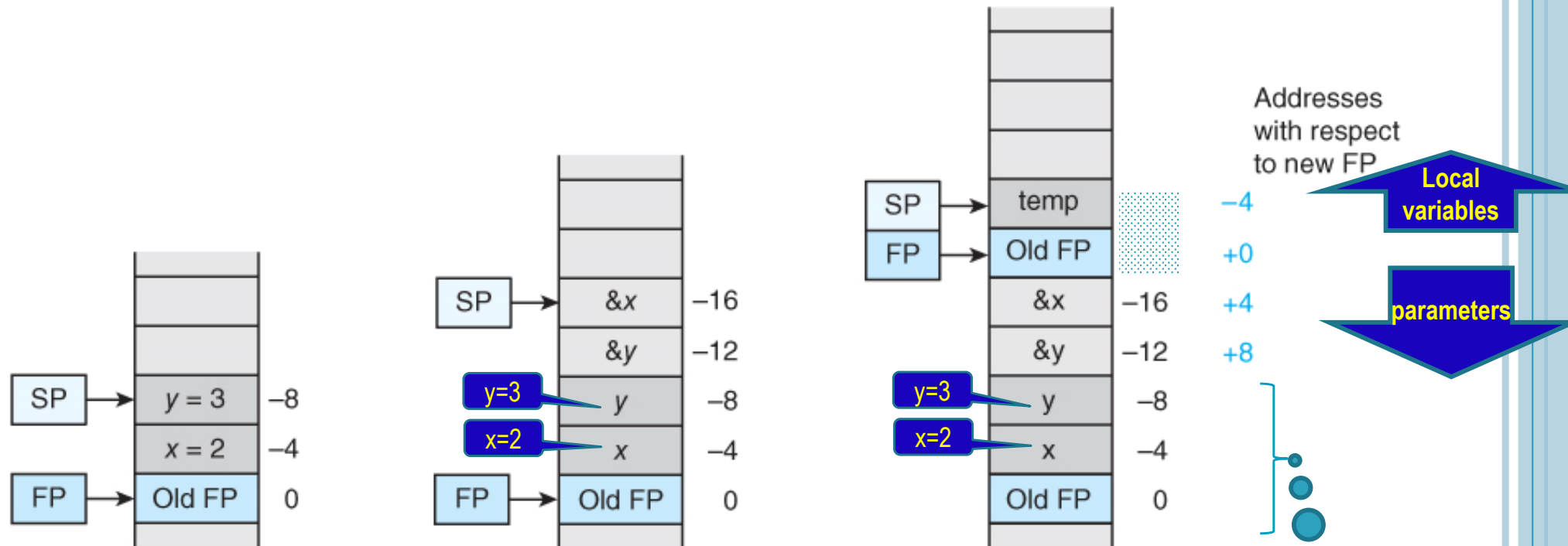
- ❑ In the function swap, the addresses of *parameters are read from the stack* by means of

```
;    temp = *a;
LDR    r1, [fp, #4]    ;get address of parameter a
LDR    r2, [r1]        ;get value of parameter a (i.e., *a)
STR    r2, [fp, #-4]   ;store *a in temp in stack frame
;    *a = *b;
LDR    r0, [fp, #8]    ;get address of parameter b
LDR    r3, [r0]        ;get value of parameter b (i.e., *b)
STR    r3, [r1]        ;store *b in *a
;    *b = temp;
LDR    r3, [fp, #-4]   ;get temp
STR    r3, [r0]        ;store temp in *b
```

Passing Parameters via the Stack

FIGURE 4.9

Passing values to a subroutine by reference



(a) State of the stack after

```
SUB sp, sp, #4
STR fp, [sp]
MOV fp, sp
SUB sp, sp, #8
MOV r0, #2
STR r0, [fp, #-4]
MOV r0, #3
STR r0, [fp, #-8]
```

in function main

(b) State of the stack after

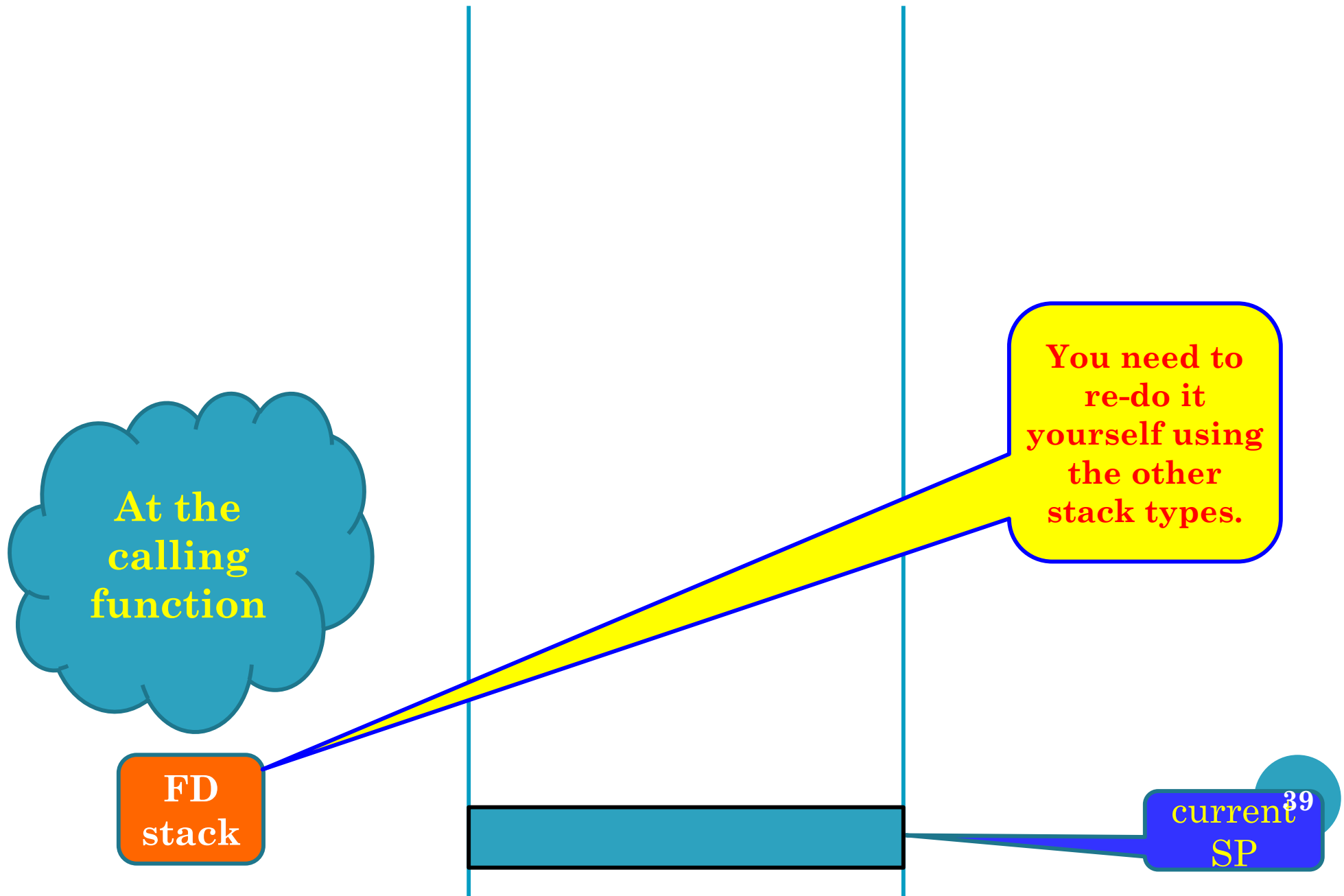
```
pushing parameter
addresses by
SUB r0, fp, #8
STR r0, [sp, #-4]!
SUB r0, fp, #4
STR r0, [sp, #-4]!
```

(c) State of the stack after

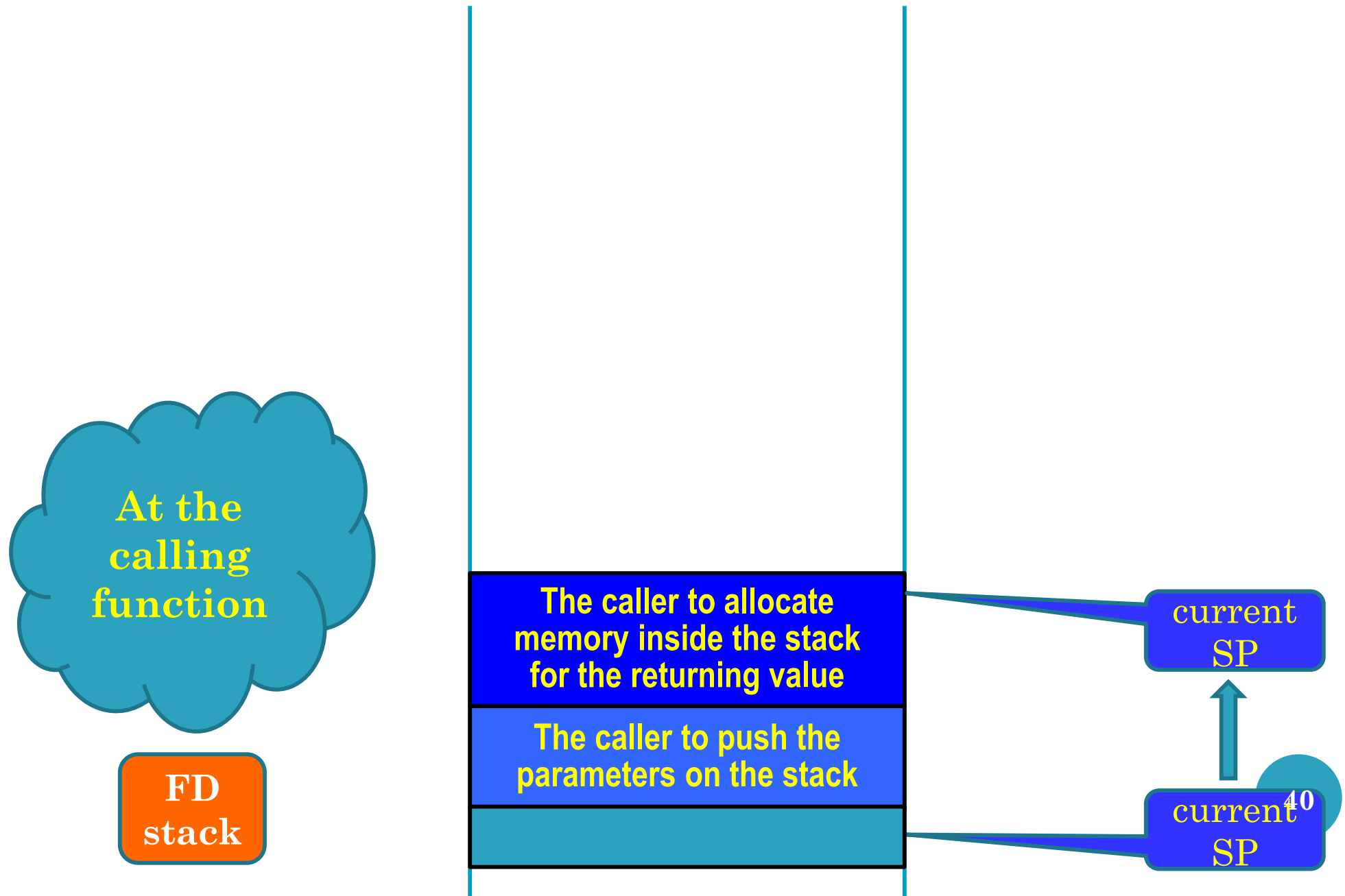
```
subroutine call and
stack frame created by
SUB sp, sp, #4
STR fp, [sp]
MOV fp, sp
SUB sp, sp, #4
```

The swap function should not have a direct access to x and y

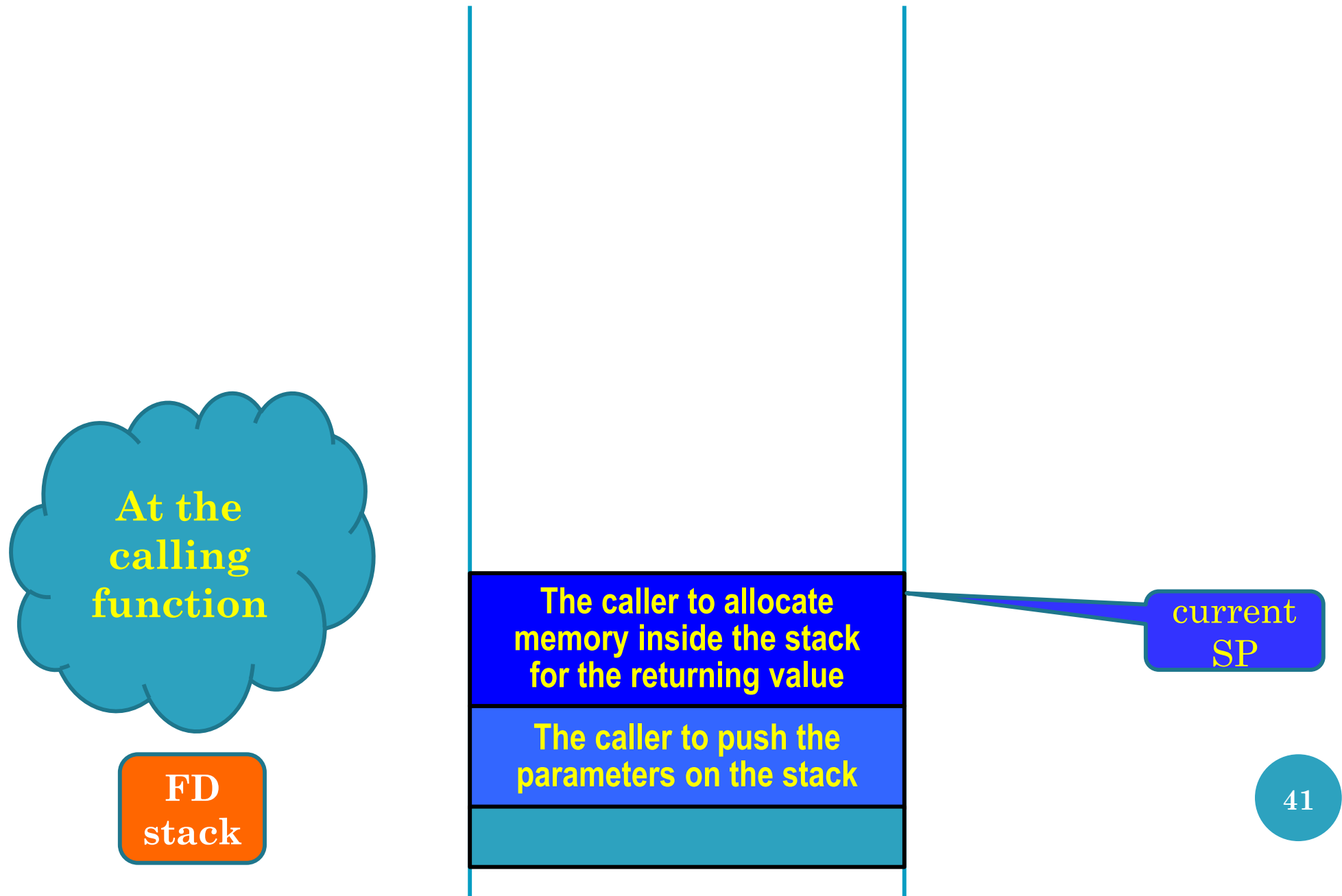
The Traditional Call/Return Mechanism



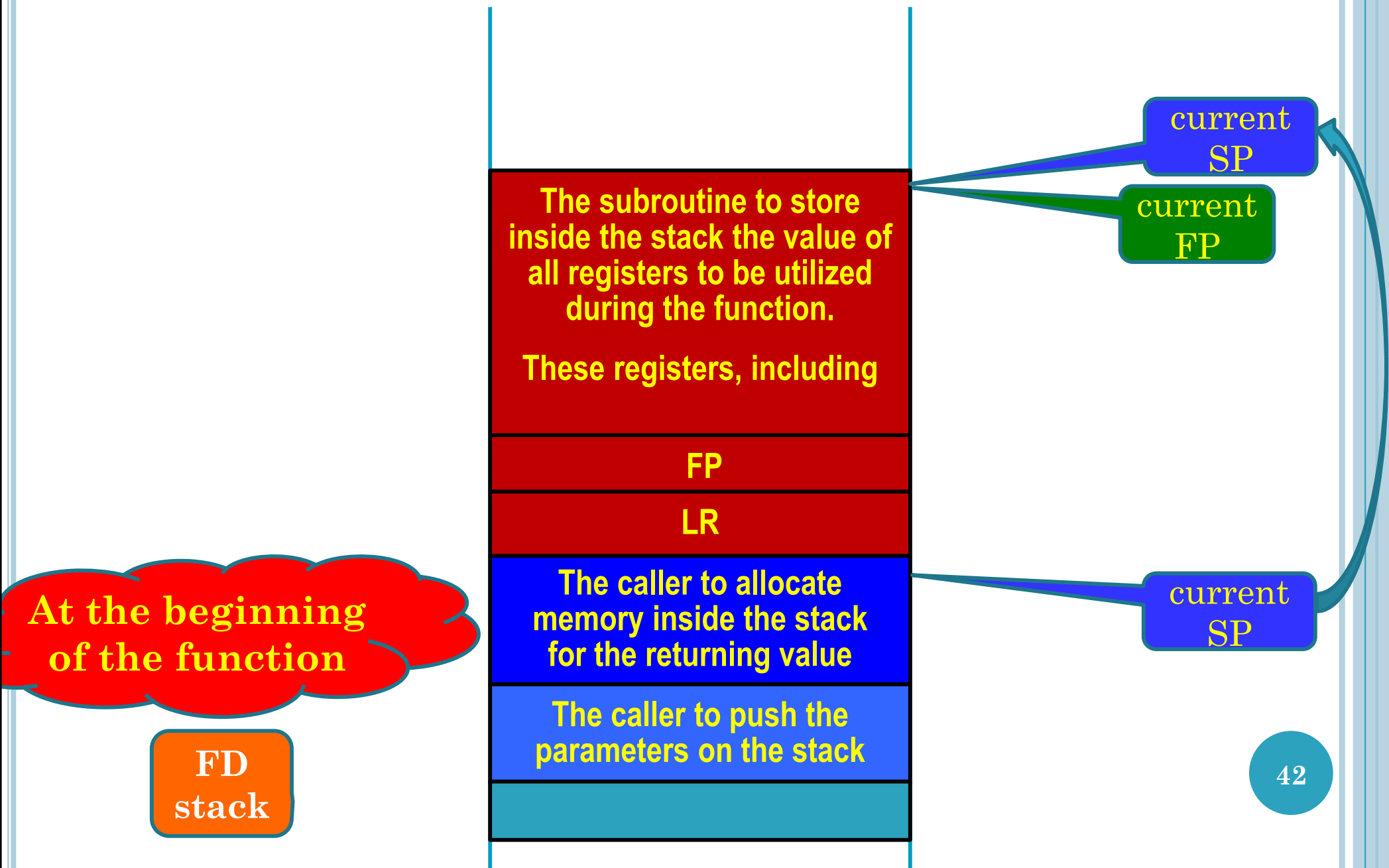
The Traditional Call/Return Mechanism



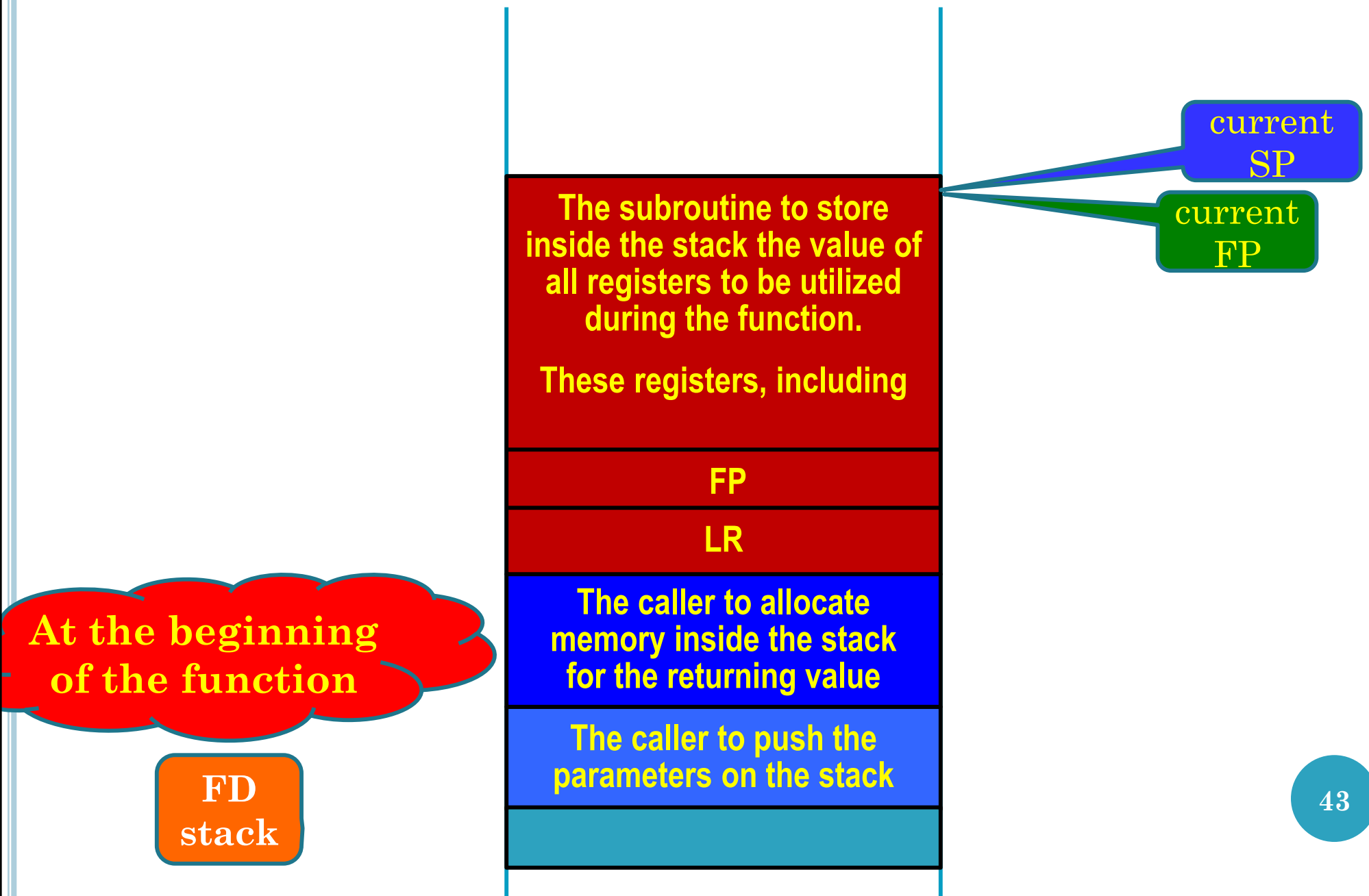
The Traditional Call/Return Mechanism



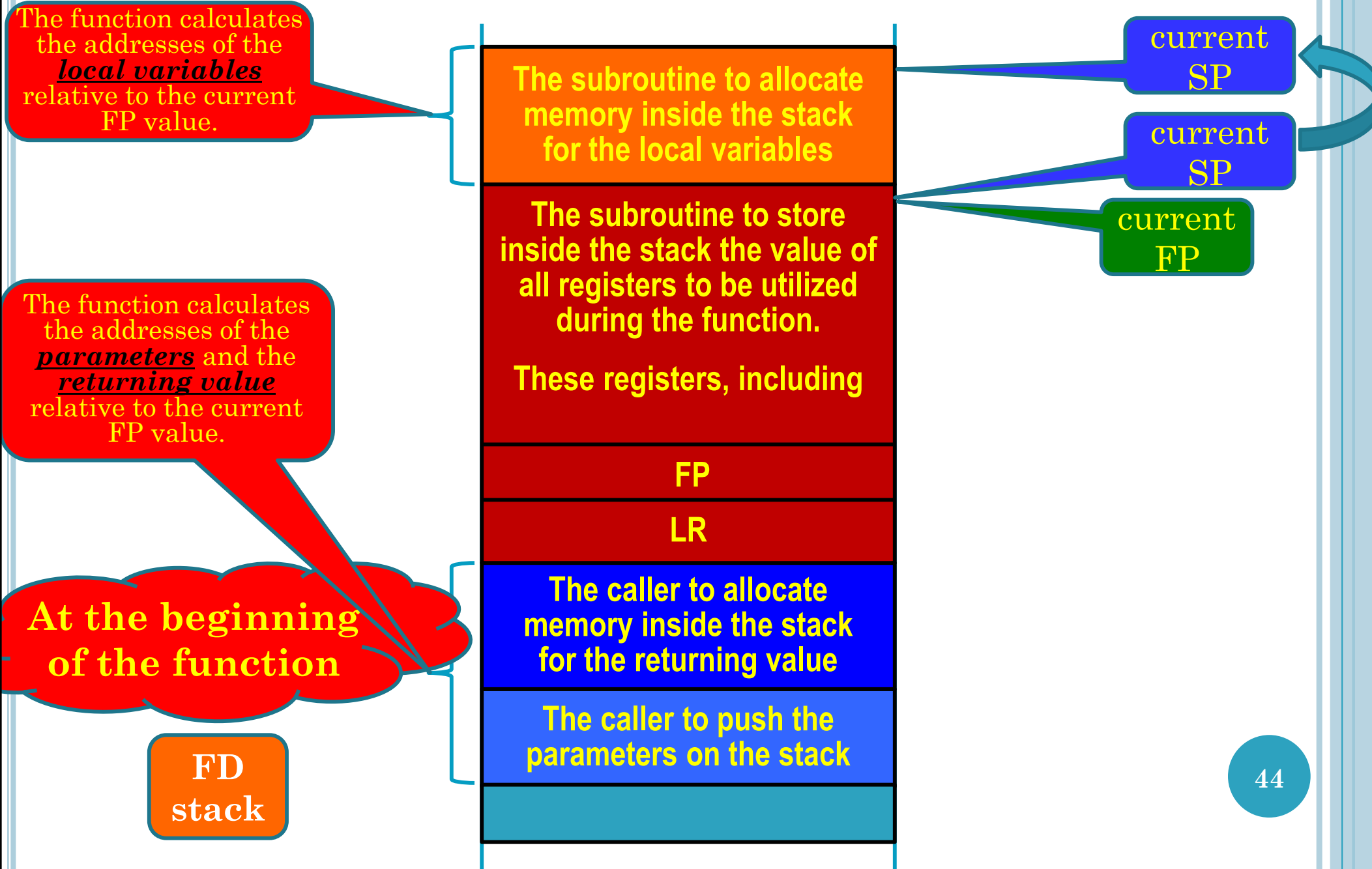
The Traditional Call/Return Mechanism



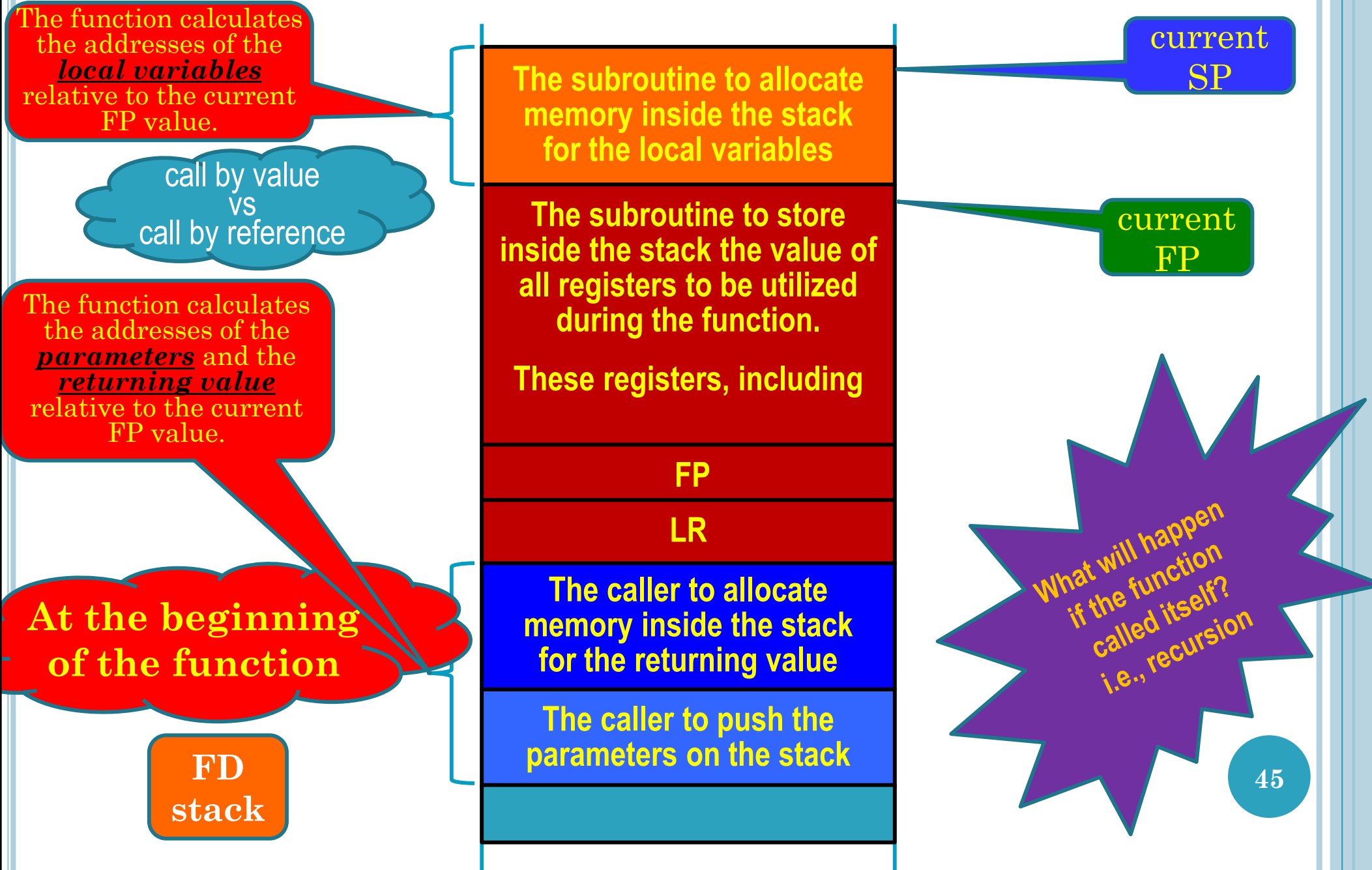
The Traditional Call/Return Mechanism



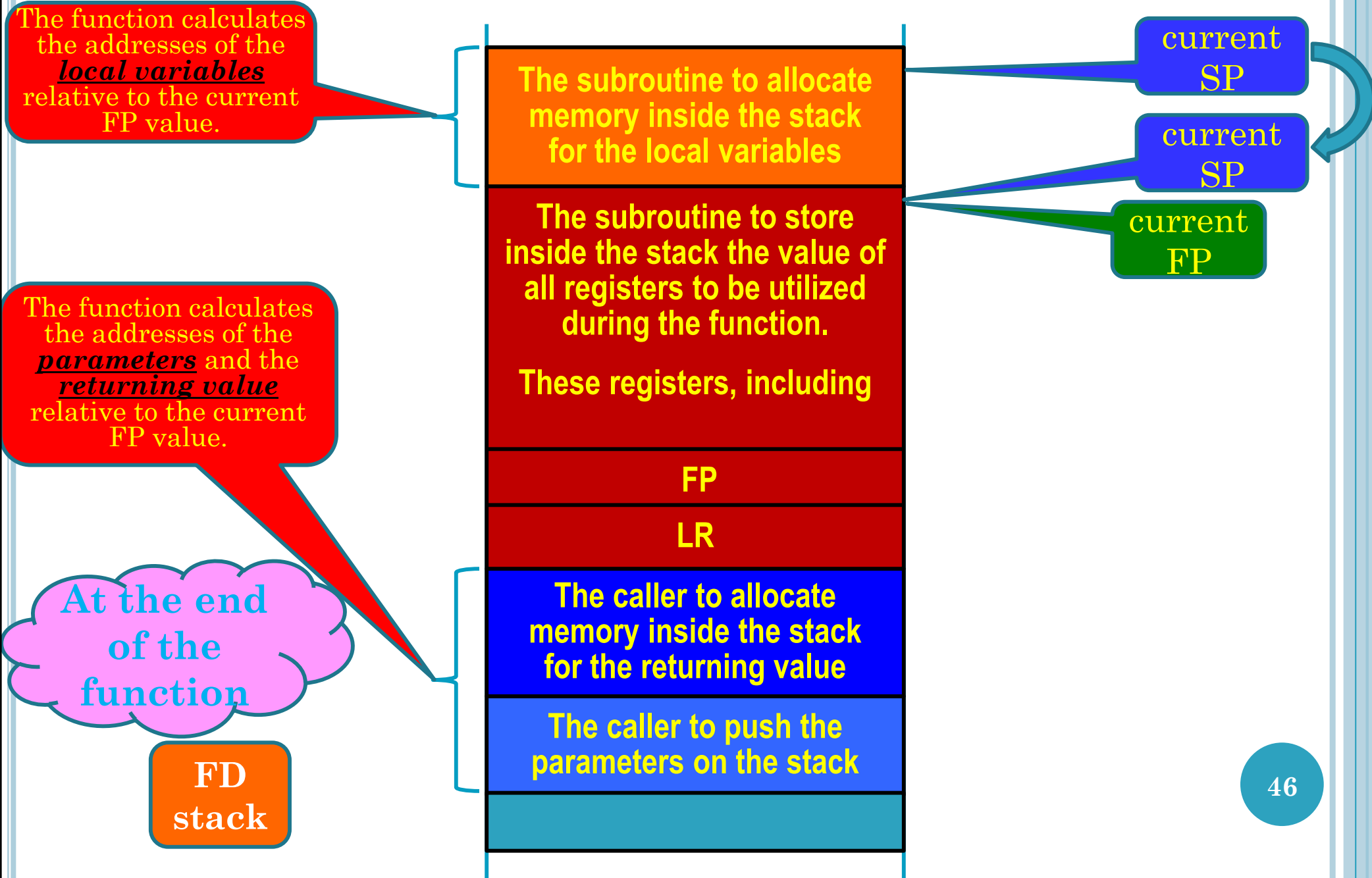
The Traditional Call/Return Mechanism



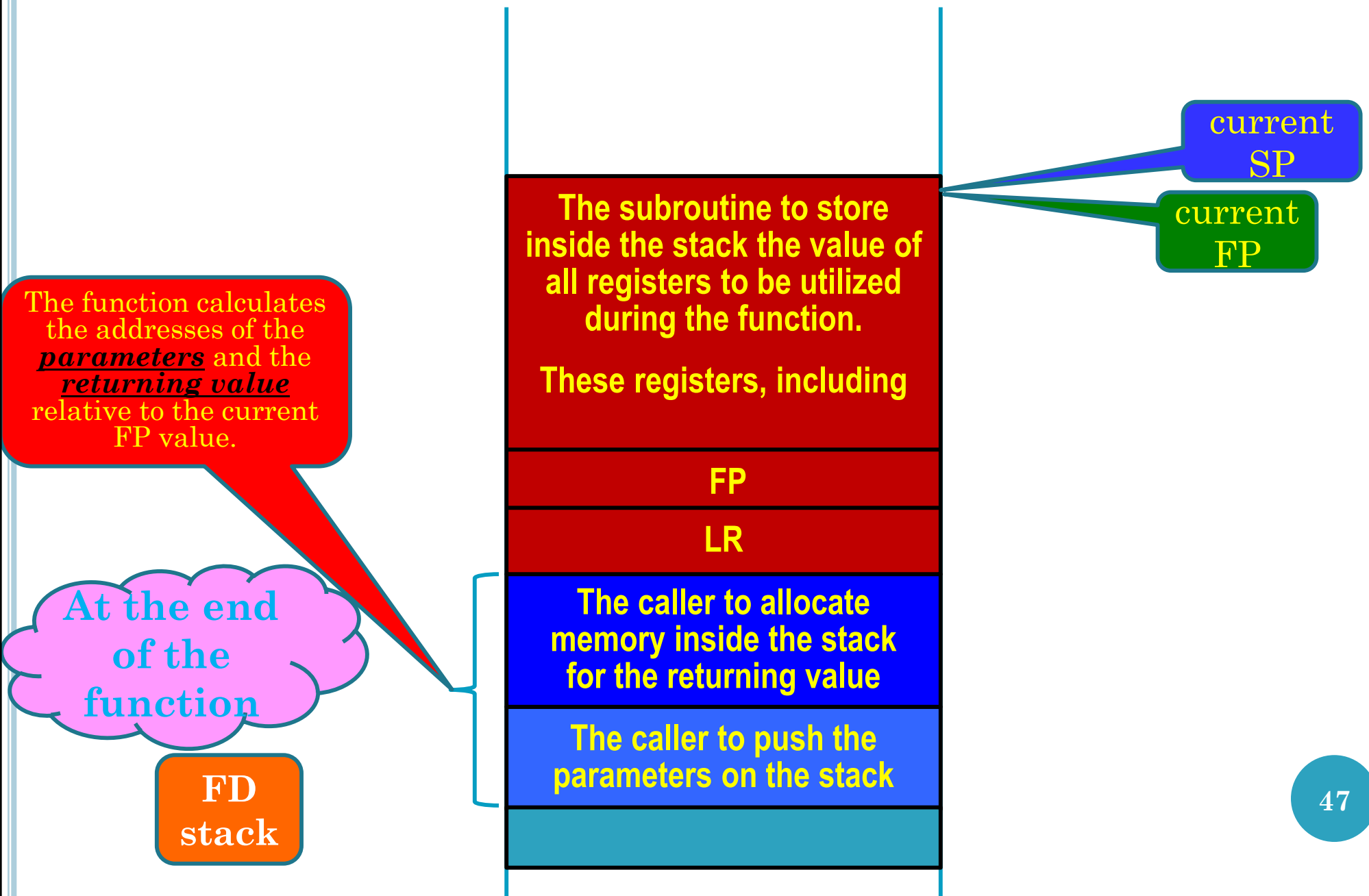
The Traditional Call/Return Mechanism



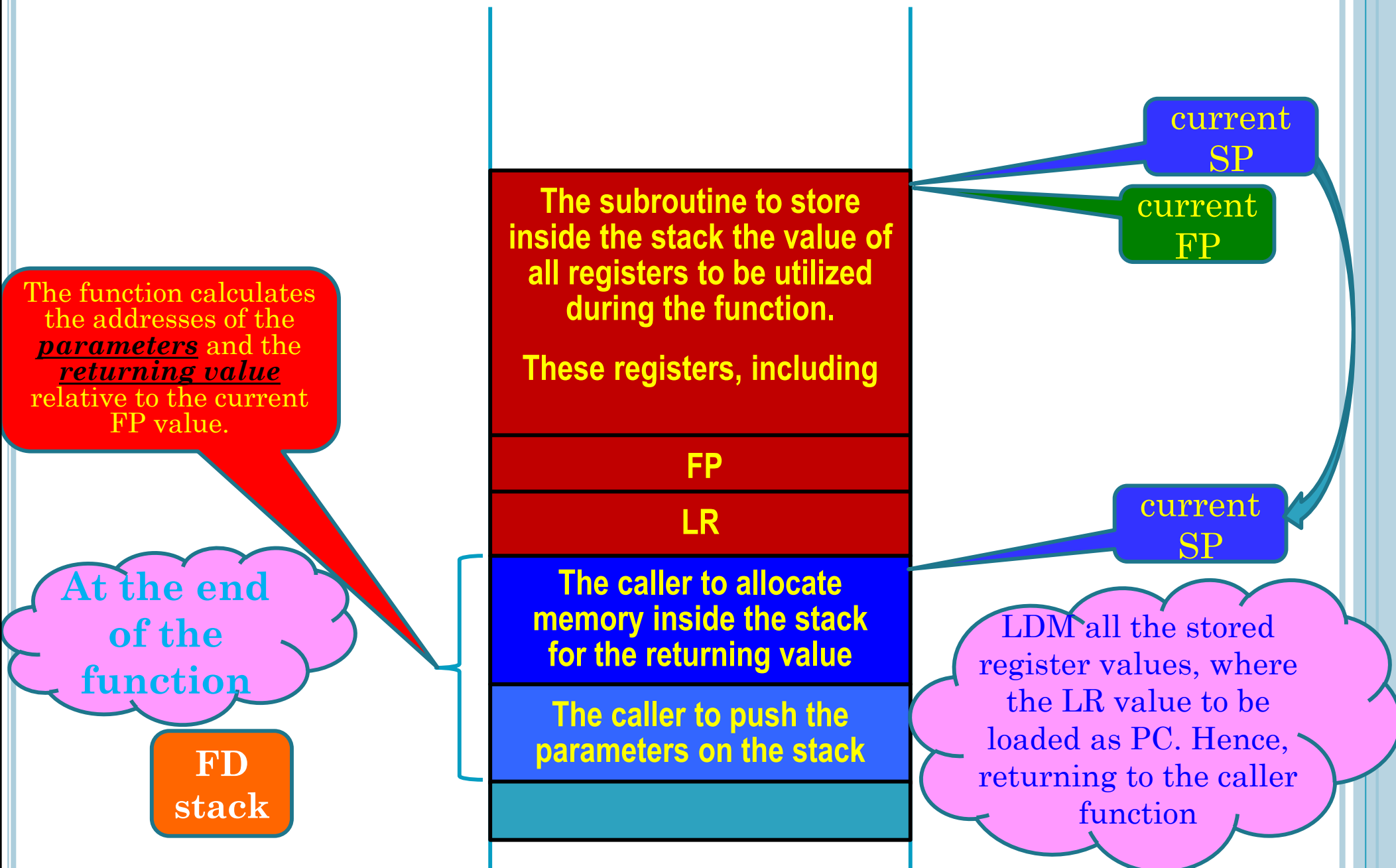
The Traditional Call/Return Mechanism



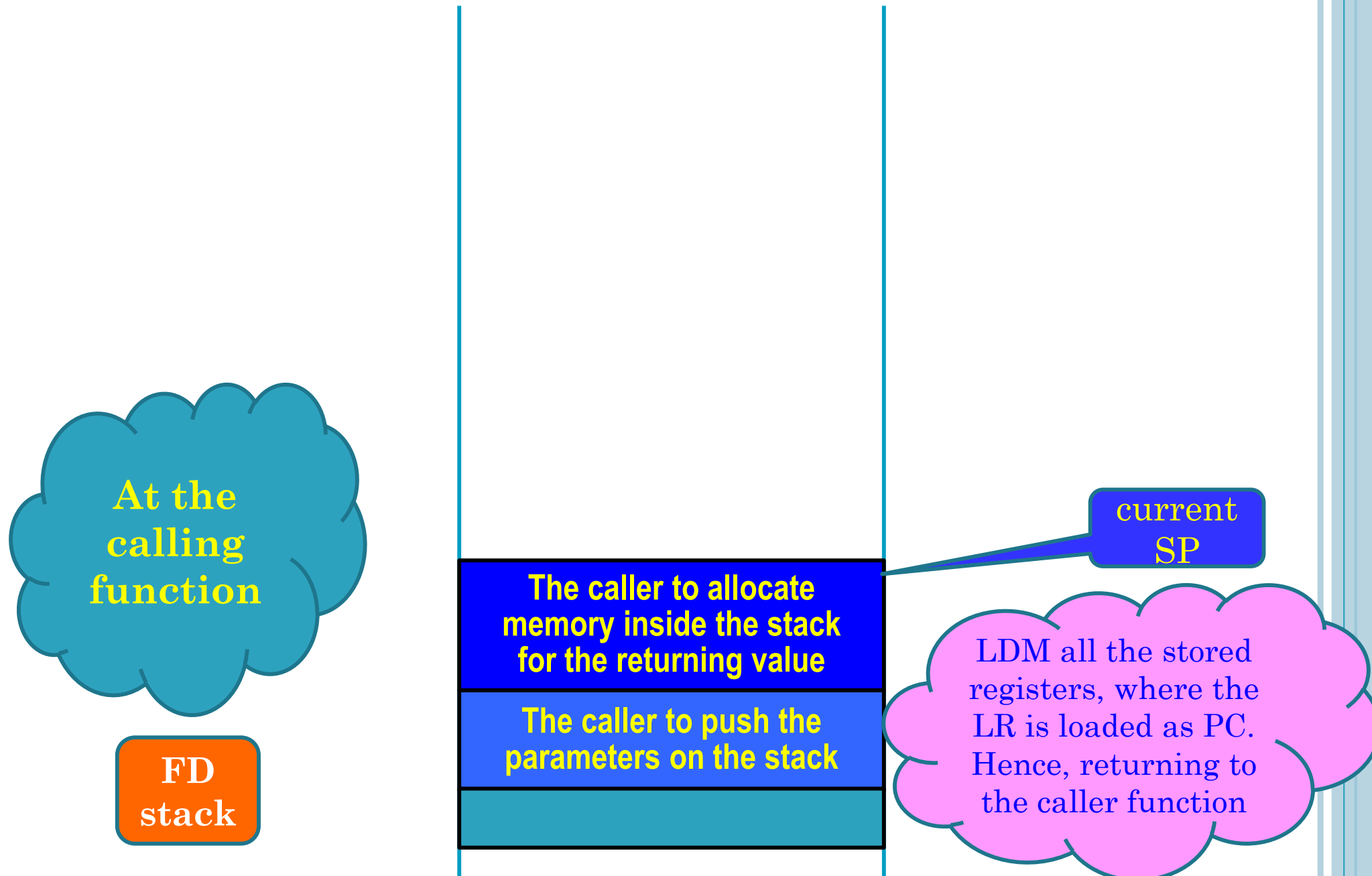
The Traditional Call/Return Mechanism



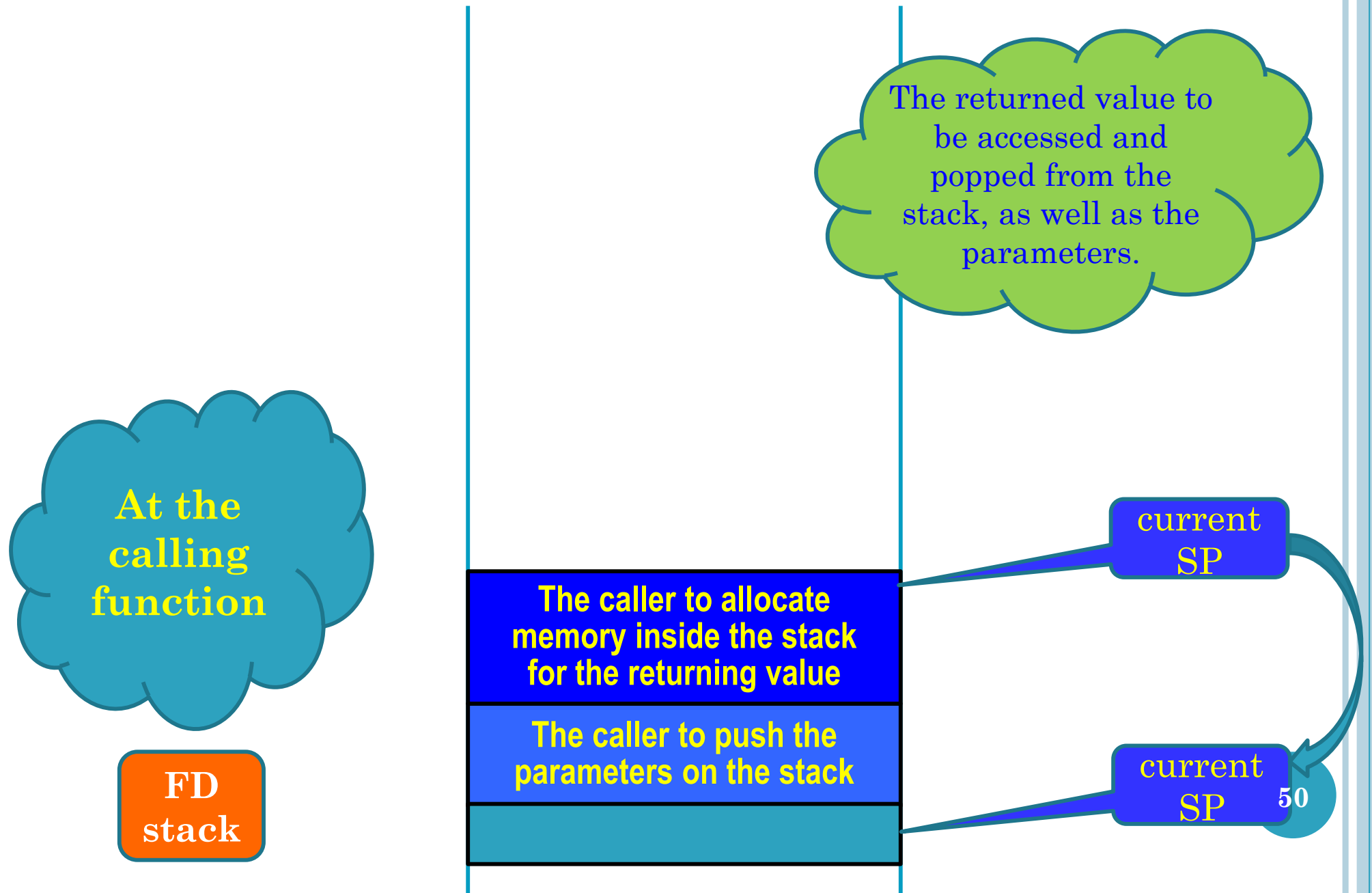
The Traditional Call/Return Mechanism



The Traditional Call/Return Mechanism



The Traditional Call/Return Mechanism



The Traditional Call/Return Mechanism

