Computer Architecture and System Programming Laboratory

TA Session 7

Co-Routines

co-routines

co-routine is assembly implementation of user-level thread

each co-routine decides to which co-routine to pass a control

We would implement simple example of two co-routines round robin scheduling:

main() → scheduler ⇄ co-routine1

⇄ co-routine2

⇄ co-routine1

⇄ co-routine2

→ **main()** → **exit program**

note: scheduler is also co-routine

co-routine state

- stack content
- registers
- flags (EFLAGS)
- stack pointer (ESP)
- instructions pointer (EIP)

- co-routine must save its current state before suspending itself (in order to continue the execution later)

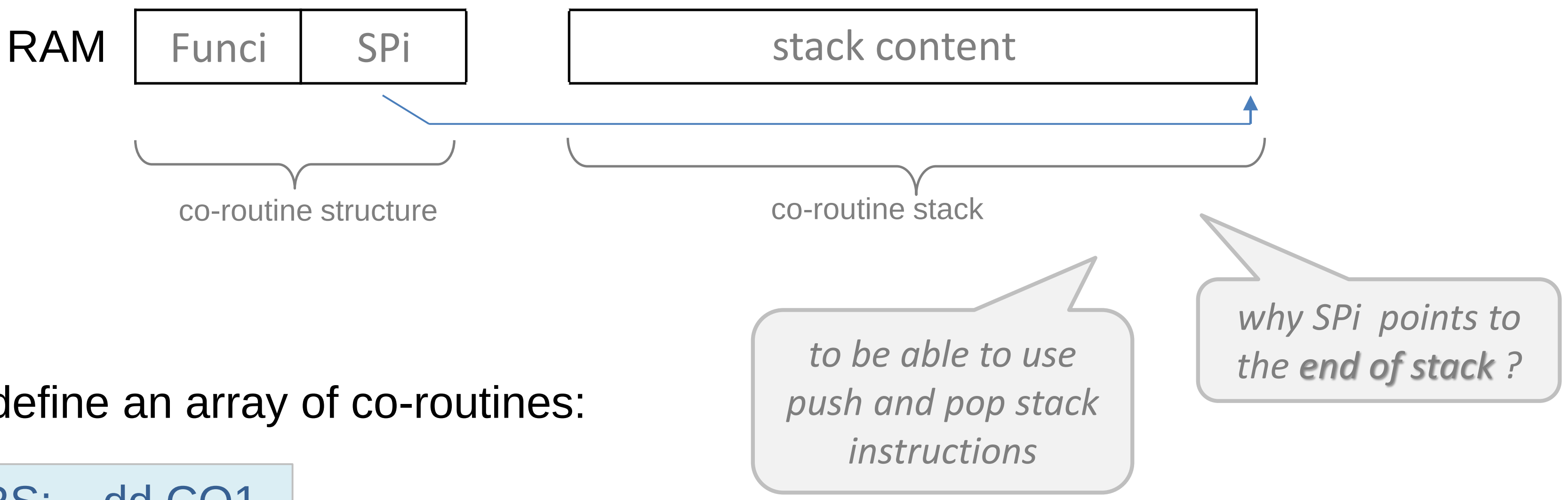
- co-routine suspends itself after some time slice
- co-routine resumes a scheduler

co-routine struct

For each co-routine COi we allocate the following struct:

```
STKSIZE  equ 16*1024                ;16 Kb
STKi:       resb STKSIZE            ; memory allocation for stack
```

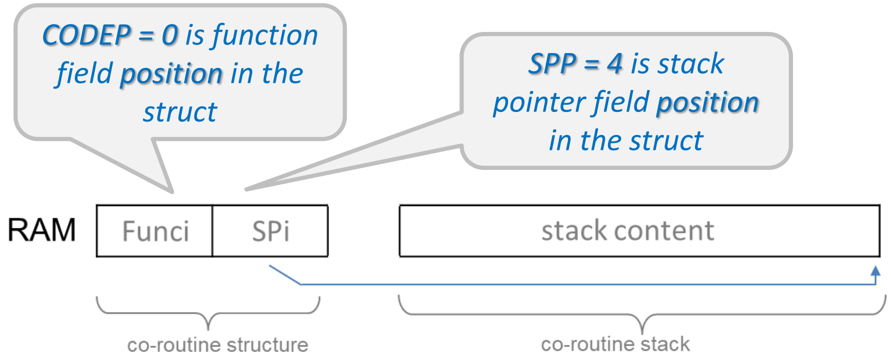
```
COi:        dd  Funci               ; pointer to co-routine function
            dd  STKi + STKSIZE      ; pointer to the beginning of co-routine stack
```


RAM
Funci
SPi
stack content
co-routine structure
co-routine stack
to be able to use push and pop stack instructions
why SPi points to the end of stack ?

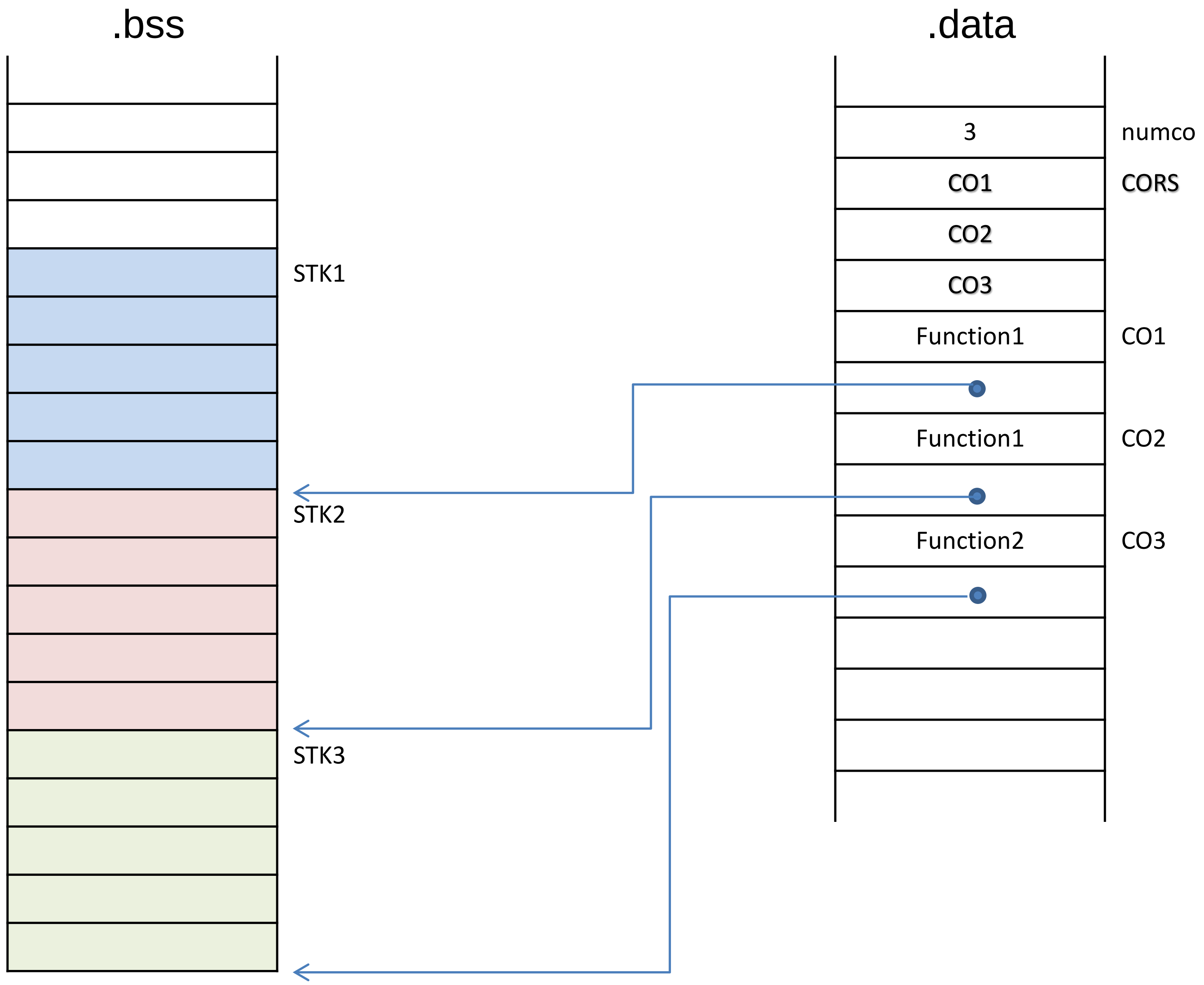
We define an array of co-routines:

```
CORS:    dd CO1
         dd CO2
         dd CO3
```

```
CODEP   equ     0           ; offset of pointer to co-routine function in co-routine struct
SPP     equ     4           ; offset of pointer to co-routine stack in co-routine struct

section .data
        global  numco
numco:  dd      3
CO1:    dd      Function1   ; struct of first co-routine
        dd      STK1+STKSZ
CO2:    dd      Function1   ; struct of second co-routine
        dd      STK2+STKSZ
CO3:    dd      Function2   ; struct of scheduler
        dd      STK3+STKSZ
CORS:   dd      CO1
        dd      CO2
        dd      CO3

section .bss
CURR:   resd    1
SPT:    resd    1   ; temporary stack pointer
SPMAIN: resd    1   ; stack pointer of main
STKSZ   equ     16*1024     ; co-routine stack size
STK1:   resb    STKSZ
STK2:   resb    STKSZ
STK3:   resb    STKSZ
```


CODEP = 0 is function field position in the struct
SPP = 4 is stack pointer field position in the struct
RAM
Funci
SPi
stack content
co-routine structure
co-routine stack

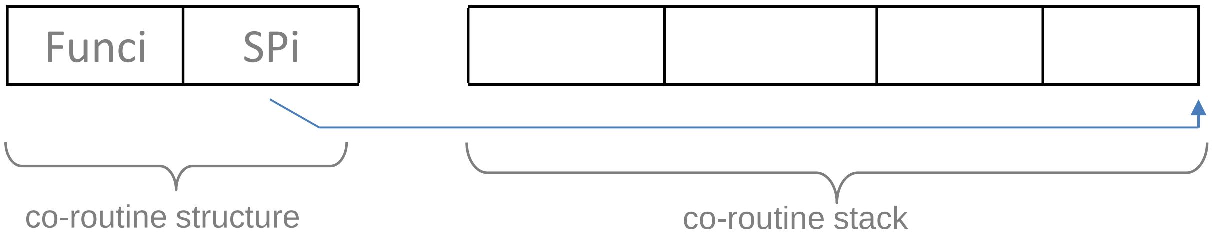
data declaration



Co-routines initialization - create initial co-routines state

```
/* initialize co-routines*/
for i=0 to numco
      initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

```
initCo:
  →  mov ebx, [ebp+8]          ; get co-routine ID number
  →  mov ebx, [4*ebx + CORS]   ; get pointer to COi struct
  →  mov eax, [ebx+CODEP]      ; get initial EIP value – pointer to COi function
  →  mov [SPT], ESP            ; save ESP value
     mov esp, [EBX+SPP]        ; get initial ESP value – pointer to COi stack
     push eax                  ; push initial “return” address
     pushfd                    ; push flags
     pushad                    ; push all other registers
     mov [ebx+SPP], esp        ; save new SPi value (after all the pushes)
     mov ESP, [SPT]            ; restore ESP value
```

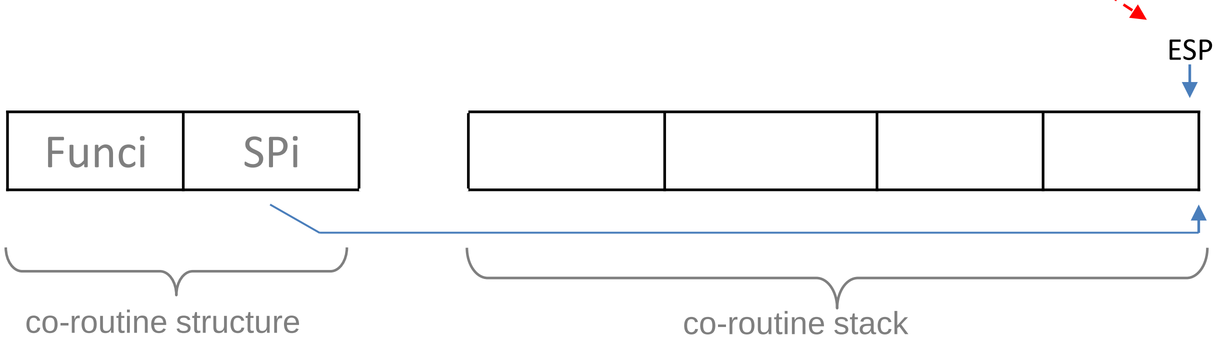


Co-routines initialization - create initial co-routines state

```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

initCo:

```
    mov ebx, [ebp+8]            ; get co-routine ID number
    mov ebx, [4*ebx + CORS]     ; get pointer to COi struct
    mov eax, [ebx+CODEP]        ; get initial EIP value – pointer to COi function
    mov [SPT], ESP              ; save ESP value
    mov esp, [EBX+SPP]          ; get initial ESP value – pointer to COi stack
    push eax                    ; push initial “return” address
    pushfd                      ; push flags
    pushad                      ; push all other registers
    mov [ebx+SPP], esp          ; save new SPi value (after all the pushes)
    mov ESP, [SPT]              ; restore ESP value
```

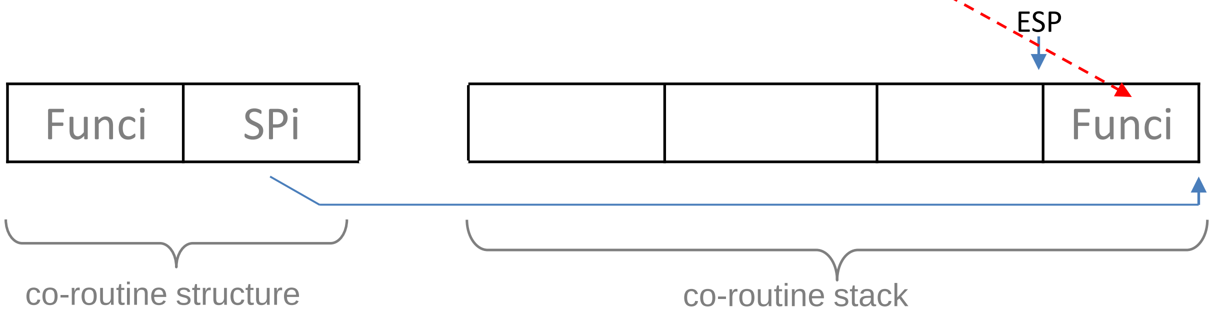


Co-routines initialization - create initial co-routines state

```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

initCo:

```
    mov ebx, [ebp+8]             ; get co-routine ID number
    mov ebx, [4*ebx + CORS]      ; get pointer to COi struct
    mov eax, [ebx+CODEP]         ; get initial EIP value – pointer to COi function
    mov [SPT], ESP               ; save ESP value
    mov esp, [EBX+SPP]           ; get initial ESP value – pointer to COi stack
    push eax                     ; push initial “return” address
    pushfd                       ; push flags
    pushad                       ; push all other registers
    mov [ebx+SPP], esp           ; save new SPi value (after all the pushes)
    mov ESP, [SPT]               ; restore ESP value
```

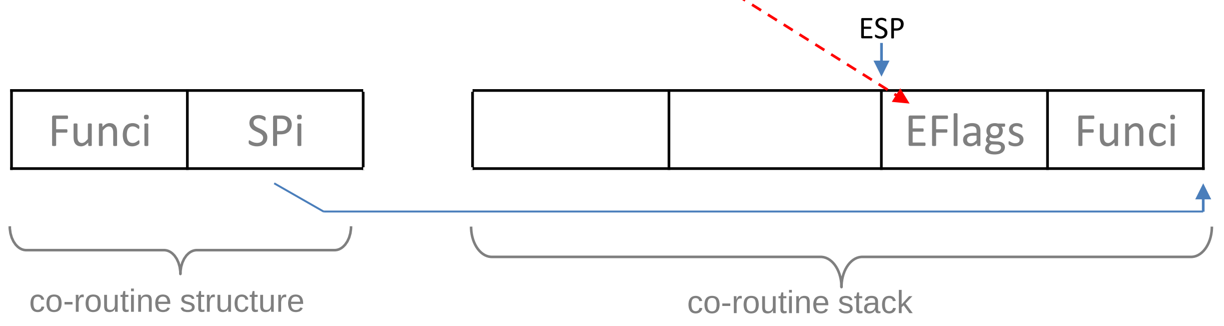


Co-routines initialization - create initial co-routines state

```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

initCo:

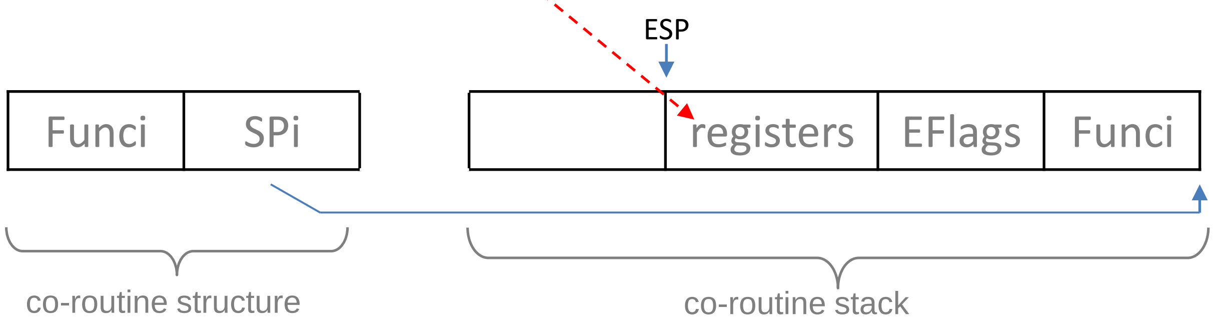
```
    mov ebx, [ebp+8]              ; get co-routine ID number
    mov ebx, [4*ebx + CORS]       ; get pointer to COi struct
    mov eax, [ebx+CODEP]          ; get initial EIP value – pointer to COi function
    mov [SPT], ESP                ; save ESP value
    mov esp, [EBX+SPP]            ; get initial ESP value – pointer to COi stack
    push eax                      ; push initial "return" address
    pushfd                        ; push flags
    pushad                        ; push all other registers
    mov [ebx+SPP], esp            ; save new SPi value (after all the pushes)
    mov ESP, [SPT]                ; restore ESP value
```



Co-routines initialization - create initial co-routines state

```
/* initialize co-routines*/
for i=0 to numco
      initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

```
initCo:
        mov ebx, [ebp+8]            ; get co-routine ID number
        mov ebx, [4*ebx + CORS]     ; get pointer to COi struct
        mov eax, [ebx+CODEP]        ; get initial EIP value – pointer to COi function
        mov [SPT], ESP              ; save ESP value
        mov esp, [EBX+SPP]          ; get initial ESP value – pointer to COi stack
        push eax                    ; push initial "return" address
        pushfd                      ; push flags
        pushad                      ; push all other registers
        mov [ebx+SPP], esp          ; save new SPi value (after all the pushes)
        mov ESP, [SPT]              ; restore ESP value
```

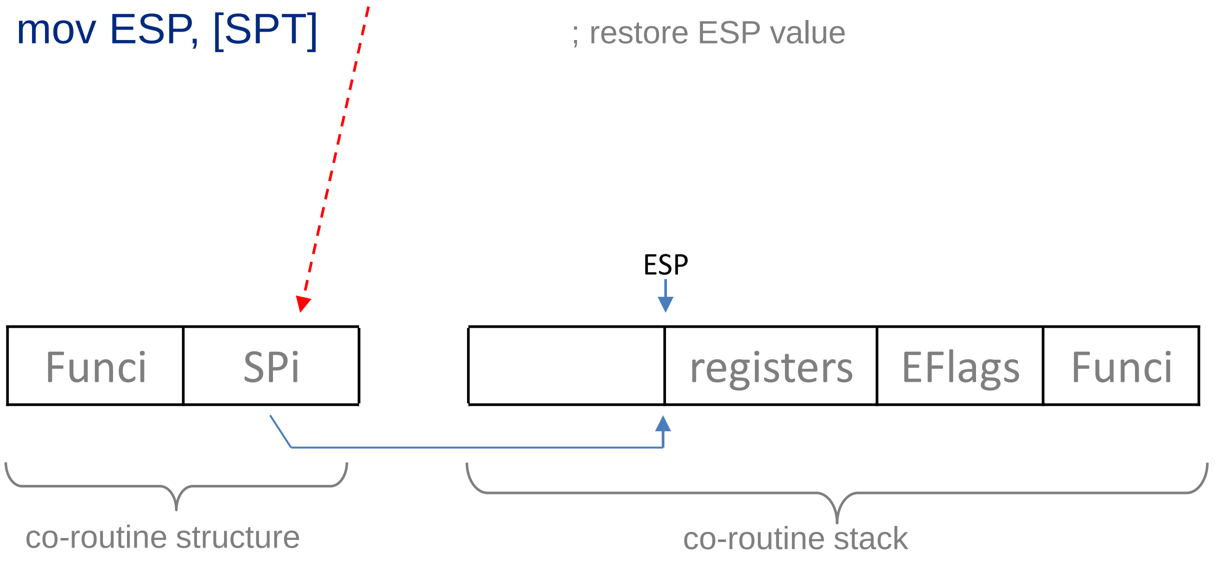


Co-routines initialization - create initial co-routines state

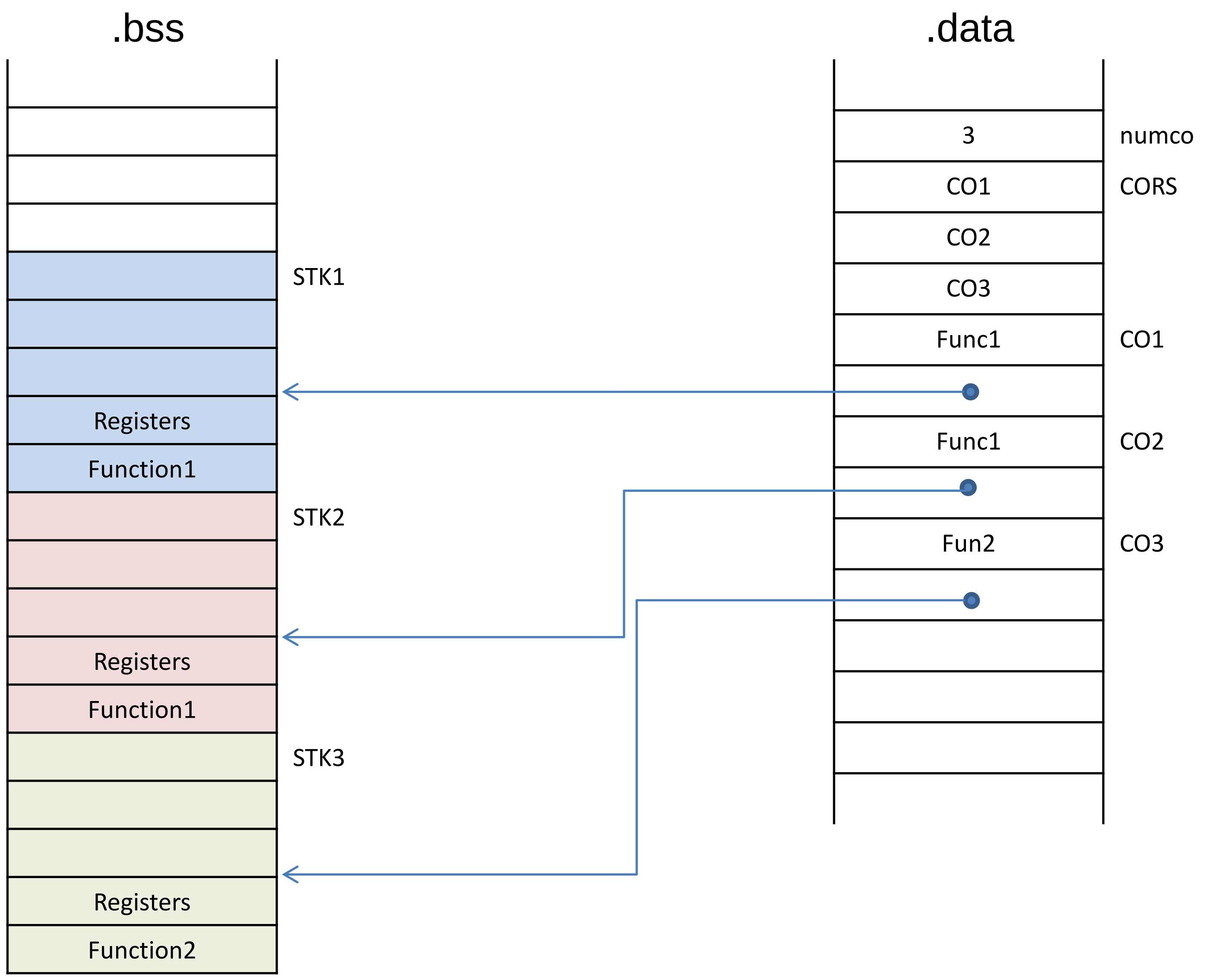
```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

initCo:

```
mov ebx, [ebp+8]            ; get co-routine ID number
mov ebx, [4*ebx + CORS]     ; get pointer to COi struct
mov eax, [ebx+CODEP]        ; get initial EIP value – pointer to COi function
mov [SPT], ESP              ; save ESP value
mov esp, [EBX+SPP]          ; get initial ESP value – pointer to COi stack
push eax                    ; push initial "return" address
pushfd                      ; push flags
pushad                      ; push all other registers
mov [ebx+SPP], esp          ; save new SPi value (after all the pushes)
mov ESP, [SPT]              ; restore ESP value
```



After co-routine initialization



Start scheduler co-routine

We start scheduling by suspending main() and resuming a scheduler co-routine.

```
startCo:
    pushad                          ; save registers of main ()
    mov [SPMAIN], ESP               ; save ESP of main ()
    mov EBX, [EBP+8]                ; gets ID of a scheduler co-routine
    mov EBX, [EBX*4 + CORS]         ; gets a pointer to a scheduler struct
    jmp do_resume                   ; resume a scheduler co-routine
```

```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

Start scheduler co-routine

We start scheduling by suspending main() and resuming a scheduler co-routine.

```
startCo:
    pushad                          ; save registers of main ()
    mov [SPMAIN], ESP               ; save ESP of main ()
    mov EBX, [EBP+8]                ; gets ID of a scheduler co-routine
    mov EBX, [EBX*4 + CORS]         ; gets a pointer to a scheduler struct
    jmp do_resume                   ; resume a scheduler co-routine
```

```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

Start scheduler co-routine

We start scheduling by suspending main() and resuming a scheduler co-routine.

```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

```
startCo:
    pushad                            ; save registers of main ()
    mov [SPMAIN], ESP                 ; save ESP of main ()
    mov EBX, [EBP+8]                  ; gets ID of a scheduler co-routine
    mov EBX, [EBX*4 + CORS]           ; gets a pointer to a scheduler struct
    jmp do_resume                     ; resume a scheduler co-routine
```

We end scheduling and go back to main().

```
endCo:
    mov   ESP, [SPMAIN]               ; restore ESP of main()
    popad                             ; restore registers of main()
```

Start scheduler co-routine

We start scheduling by suspending main() and resuming a scheduler co-routine.

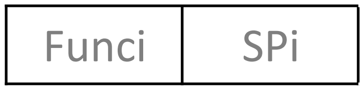
```
startCo:
    pushad                            ; save registers of main ()
    mov [SPMAIN], ESP                 ; save ESP of main ()
    mov EBX, [EBP+8]                  ; gets ID of a scheduler co-routine
    mov EBX, [EBX*4 + CORS]           ; gets a pointer to a scheduler struct
    jmp do_resume                     ; resume a scheduler co-routine
```

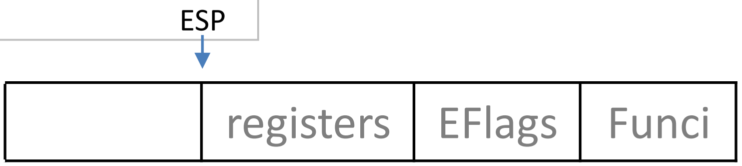
```
/* initialize co-routines*/
for i=0 to numco
    initCo(i)
/* start a scheduler co-routine*/
startCo(2)
```

```
resume:     ; save state of current co-routine
            pushfd
            pushad
            mov EDX, [CURR]
            mov [EDX+SPP], ESP   ; save current ESP
do_resume:  ; load ESP for resumed co-routine
            mov ESP, [EBX+SPP]
            mov [CURR], EBX
            popad   ; restore resumed co-routine state
            popfd
            ret     ; "return" to resumed co-routine
```

EBX points to the struct of the ***co-routine to be resumed***

CURR points to the struct of the ***current co-routine***





Round Robin scheduler

```
do N times
    if meets stop condition
        jmp endCo                ; resume main()
    pick up next thread i
    mov EBX, [CORS + i*4]        ; resume COi
    call resume
```

any other co-routine

```
do N times
    do some work
    mov EBX, [CORS + 8] ; resumes scheduler
    call resume
```

EBX** points to the struct of the **co-routine to be resumed.** **CURR** points to the struct of the **current co-routine

```
resume:                                  ; save state of current co-routine
        pushfd
        pushad
        mov     EDX, [CURR]
        mov     [EDX+SPP], ESP           ; save current ESP
do_resume:                               ; load ESP for resumed co-routine
        mov     ESP, [EBX+SPP]
        mov     [CURR], EBX
        popad                            ; restore resumed co-routine state
        popfd
        ret                              ; "return" to resumed co-routine
```

__note__: after 'call resume' return address of the current co-routine is pushed automatically into this co-routine stack. Thus, we only should save EFLAGS, ESP, and registers

Function2

This function used as scheduler code

```
FMT2: db "Function2, co %lx, called by %lx, pass %ld", 10, 0
```

Function2:

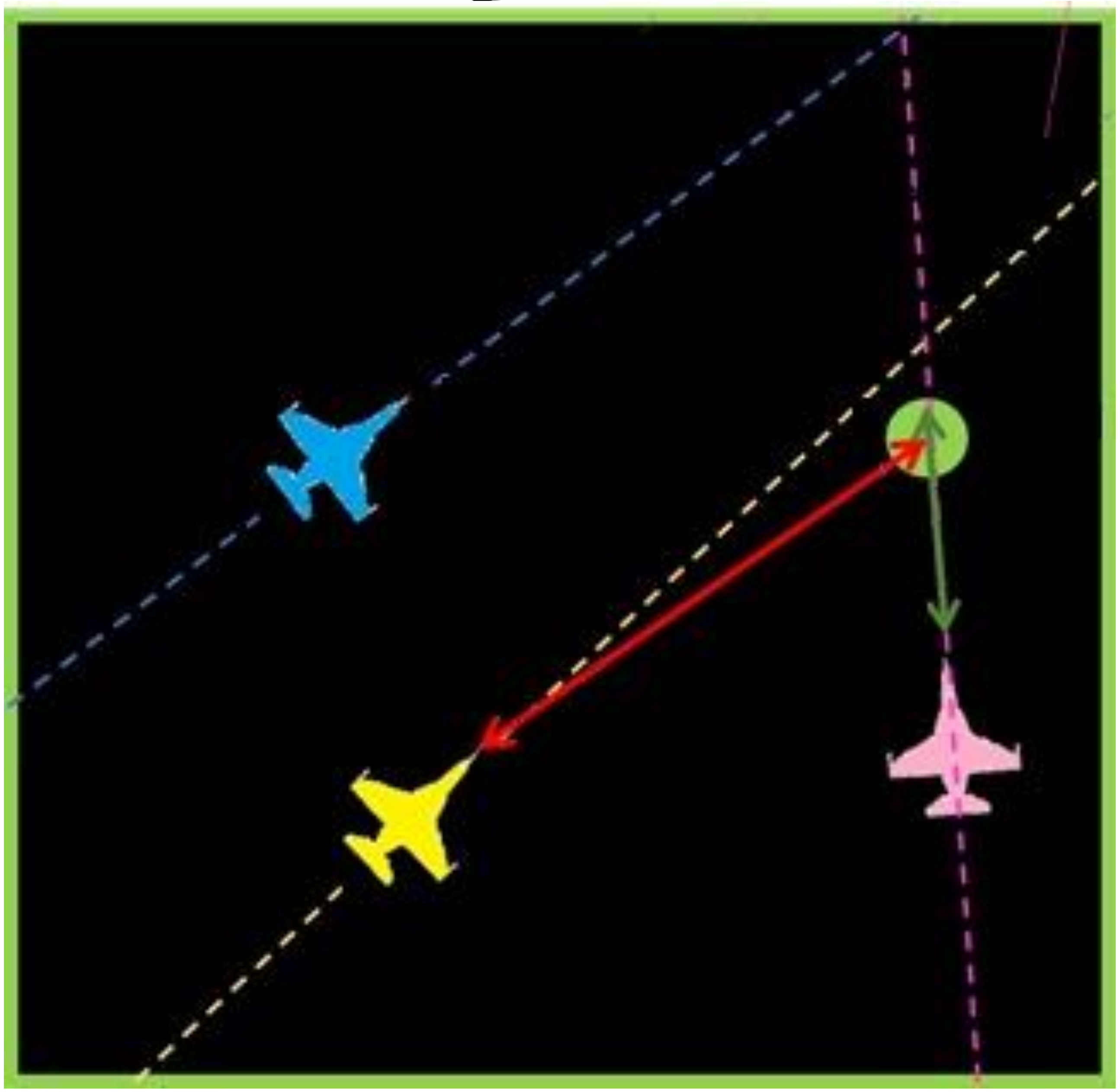
```
Function2:
        push    dword   1
        push    dword [CORS] ; indeed, called by main
        push    dword [CURR]
        push    dword FMT2
        call    printf
        add     ESP, 16
        mov     EBX, [CORS]      ; resume CO1
        call    resume

        push    dword   2
        push    dword [CORS]
        push    dword [CURR]
        push    dword FMT2
        call    printf
        add     ESP, 16
        mov     EBX, [CORS+4]    ; resume CO2
        call    resume
        push    dword   3
        push    dword [CORS+4]
        push    dword [CURR]
        push    dword FMT2
        call    printf
        add     ESP, 16
        mov     EBX, [CORS]      ; resume CO1
        call    resume
        push    dword   4
        push    dword [CORS]
        push    dword [CURR]
        push    dword FMT2
        call    printf
        add     ESP, 16
        mov     EBX, [CORS+4]    ; resume CO2
        call    resume
        jmp end_co  ; resume main
```

Function1

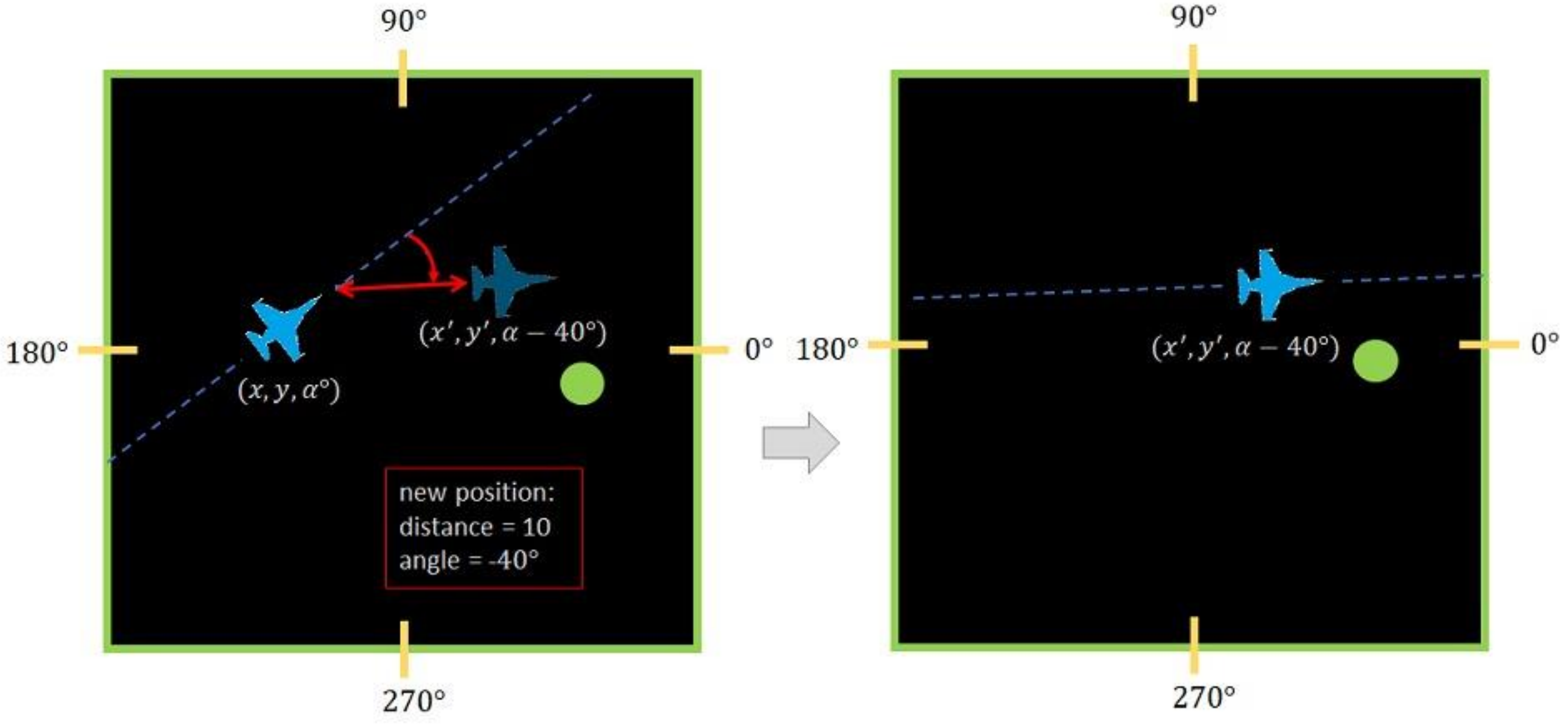
This function used as code for co-routines 1 and 2

```
FMT1: db "Function1, co %lx, called by %lx, pass %ld", 10, 0

Function1:
        push    dword   1
        push    dword [CORS+8]
        push    dword [CURR]
        push    dword FMT1
        call    printf
        add     ESP, 16
        mov     EBX, [CORS+8]           ; resume a scheduler
        call    resume

        push    dword   2
        push    dword [CORS+8]
        push    dword [CURR]
        push    dword FMT1
        call    printf
        add     ESP, 16
        mov     EBX, [CORS+8]           ; resume a scheduler
        call    resume
```

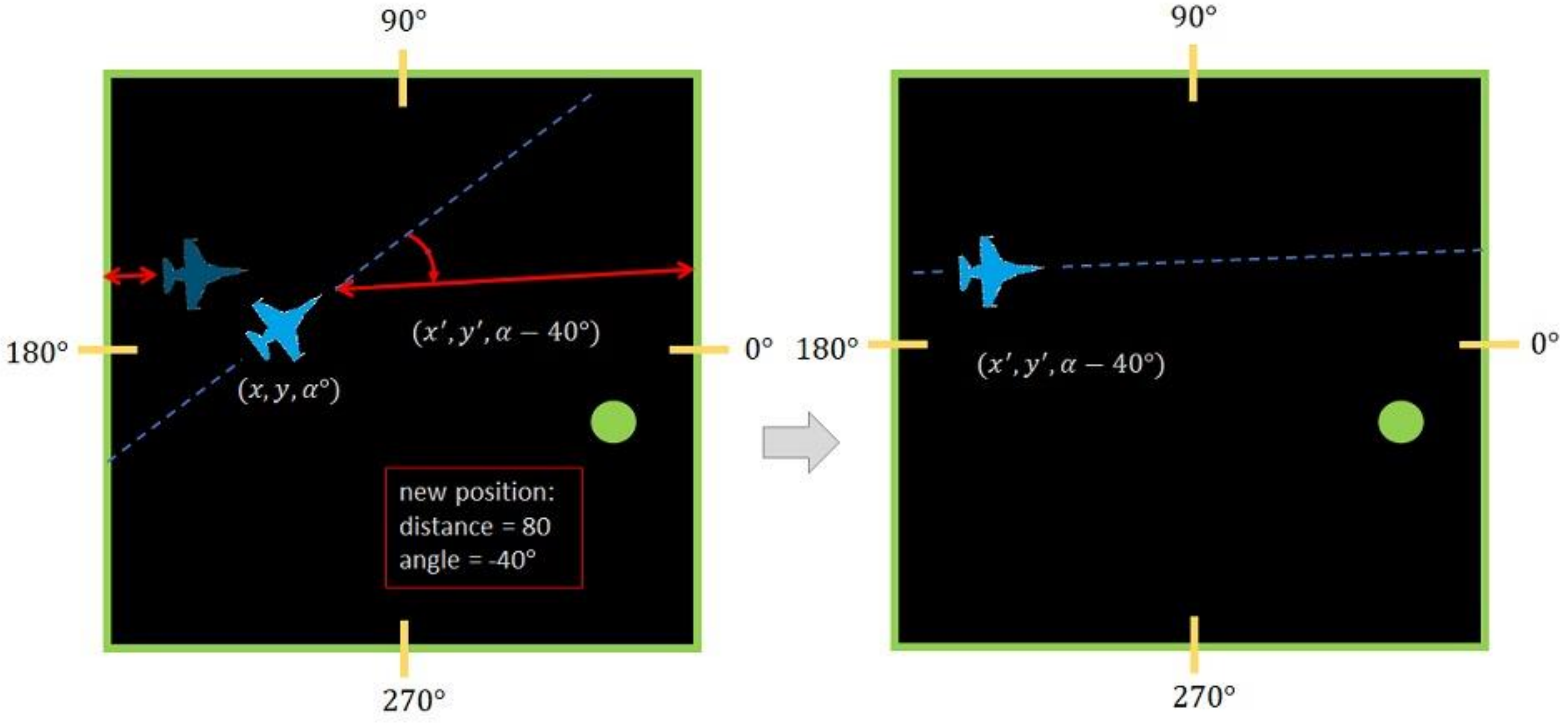
Assignment 3



Assignment 3



Assignment 3


90°
180°
0°
270°
$(x, y, \alpha°)$
$(x', y', \alpha - 40°)$
new position:
distance = 80
angle = -40°
90°
180°
0°
270°
$(x', y', \alpha - 40°)$