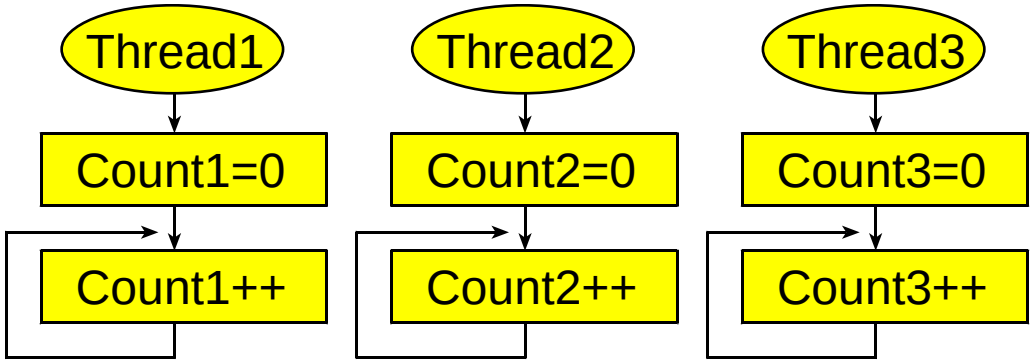
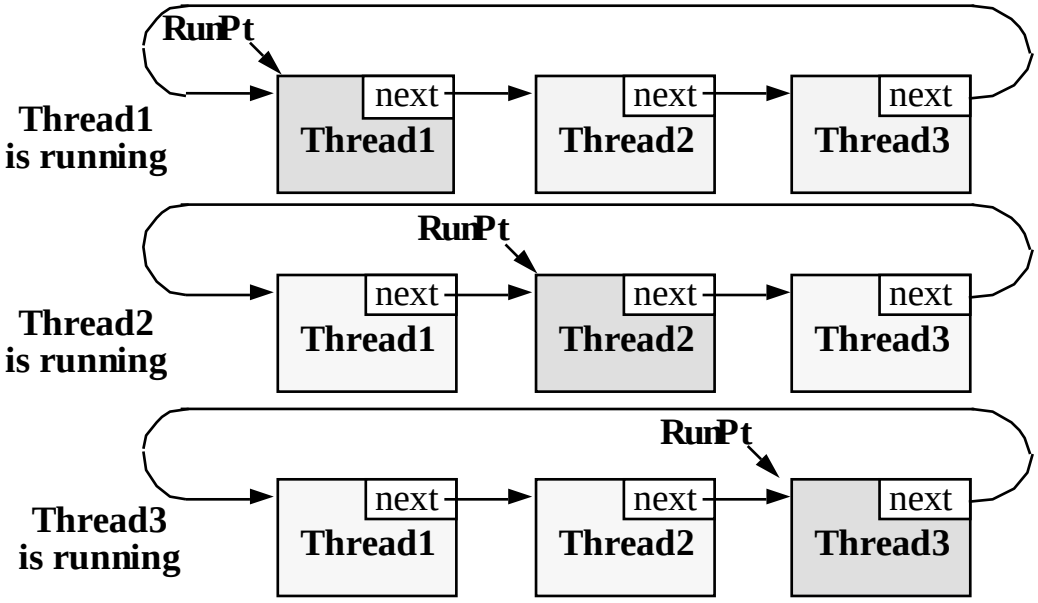
Threads

TCB

Stacks

Scheduler





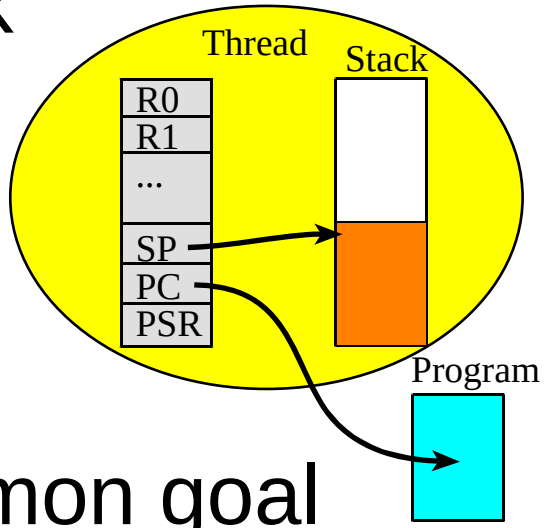
See Testmain1

See Testmain2

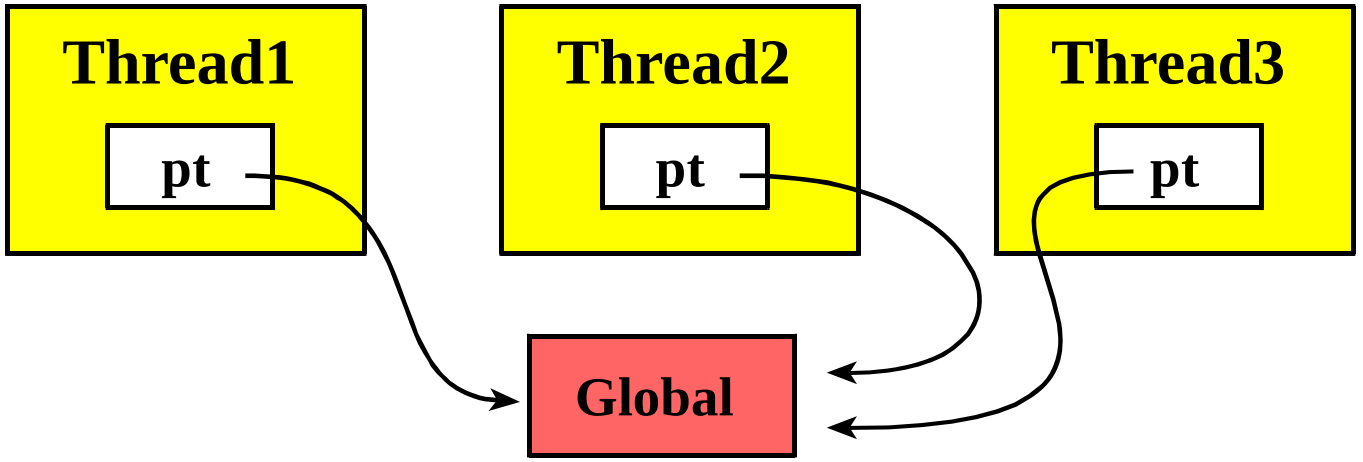
Reference book, chapter 4

Thread or Light-weight process

- Execution of a software task
- Has its own registers
- Has its own stack
- Local variables are private
- Threads cooperate for common goal
- Private global variables
  - Managed by the OS
  - Allocated in the TCB (e.g., **`Id`**)

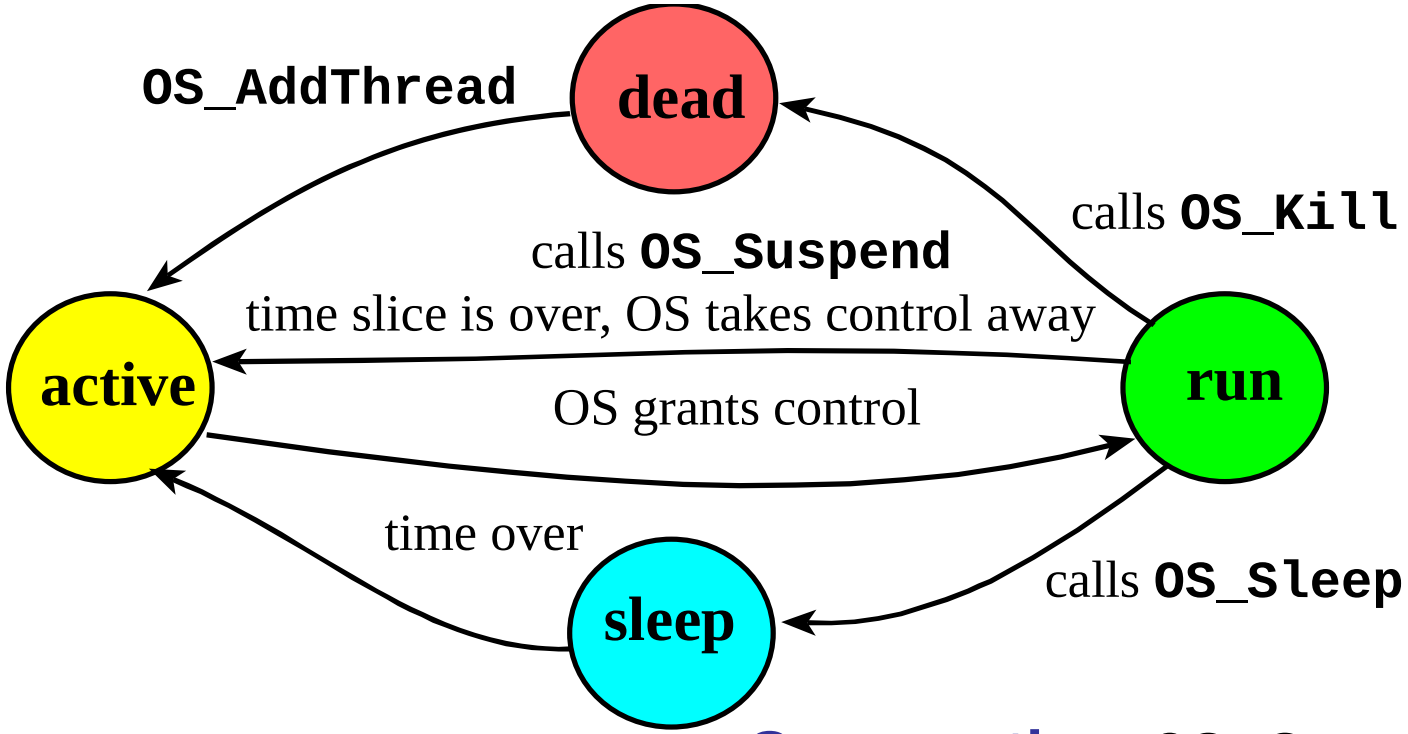


Communication/sharing



- Shared Globals
- Mailbox (Lab 2)
- FIFO queues (Lab 2)
- Message (Lab 6)

Thread States



Lab 3 will add **Blocked**

Cooperative, `OS_Suspend`
Round robin (Lab 2)
Weighted round robin
Priority (Lab 3)

Thread Control Block

- Stack pointer
- Next or Next/Previous links
- Id
- Sleep counter
- Blocked pt (Lab 3)
- Priority (Lab 3)

Where are the registers saved?

```
struct TCB {
   // order??, types??
};
typedef struct TCB
TCBType;
typedef TCBType * TCBPtr;
```

Look at TCB in **uCOS-II ucos_ii.h**

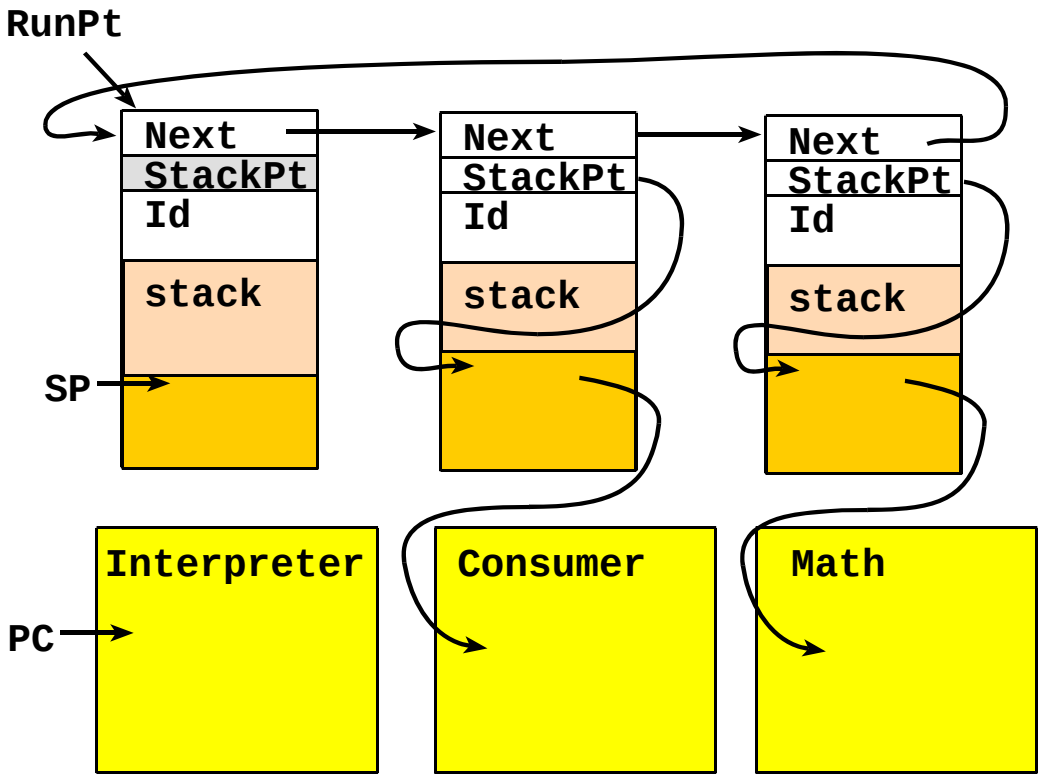
Micrium\Software\uCOS-II\Source

`os_tcb`

Round Robin

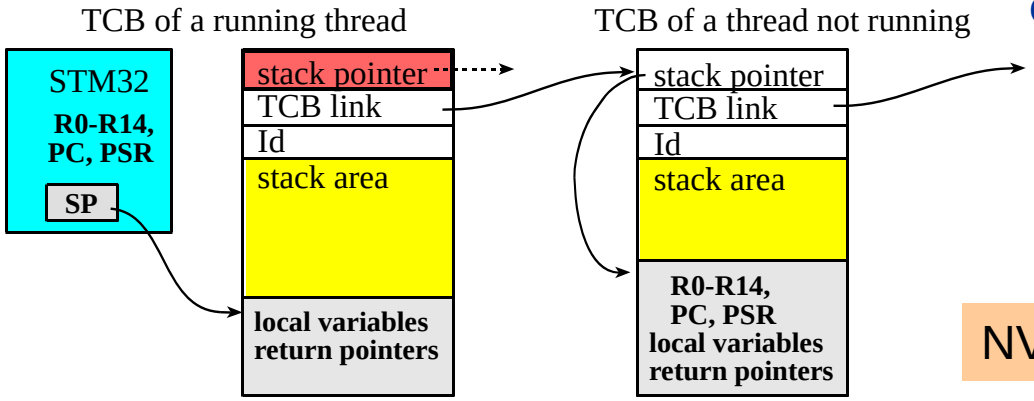
```
OS_AddThread(&Interpreter);
OS_AddThread(&Consumer);
OS_AddThread(&Math);
OS_Launch(TIMESLICE); // doesn't return
```

RunPt



Thread Switch

- Prevent switching out background tasks
- PendSV handler
- Give PendSV handler lowest priority
- Use C code to find next thread
- Trigger PendSV



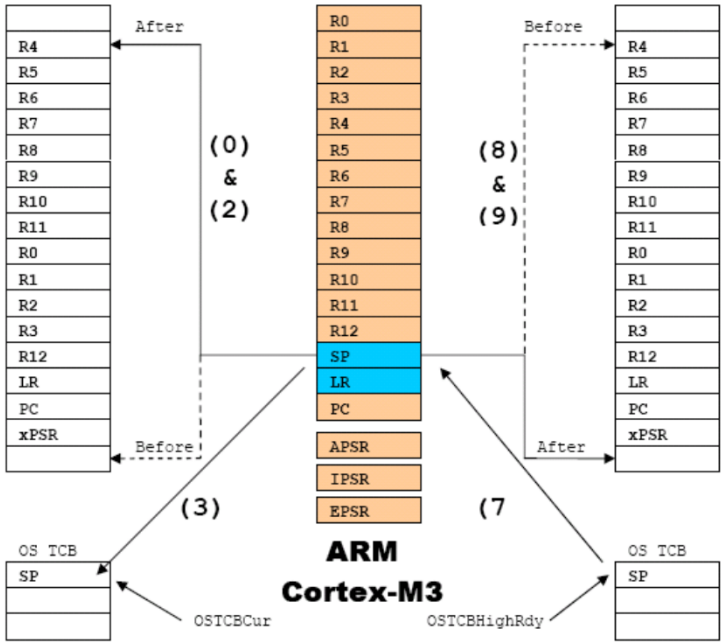

```
NVIC_INT_CTRL     EQU     0xE000ED04
NVIC_PENDSVSET  EQU     0x10000000
ContextSwitch
    LDR     R0, =NVIC_INT_CTRL
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

```
NVIC_INT_CTRL_R = 0x10000000;
```

PendSV Thread Switch

1) disable interrupts
2) save registers R4 to R11 on the user stack
3) save stack pointer into TCB
4) choose next thread
5) retrieve stack pointer from
6) restore registers R4 to R11
7) reenable interrupts
8) return from interrupt

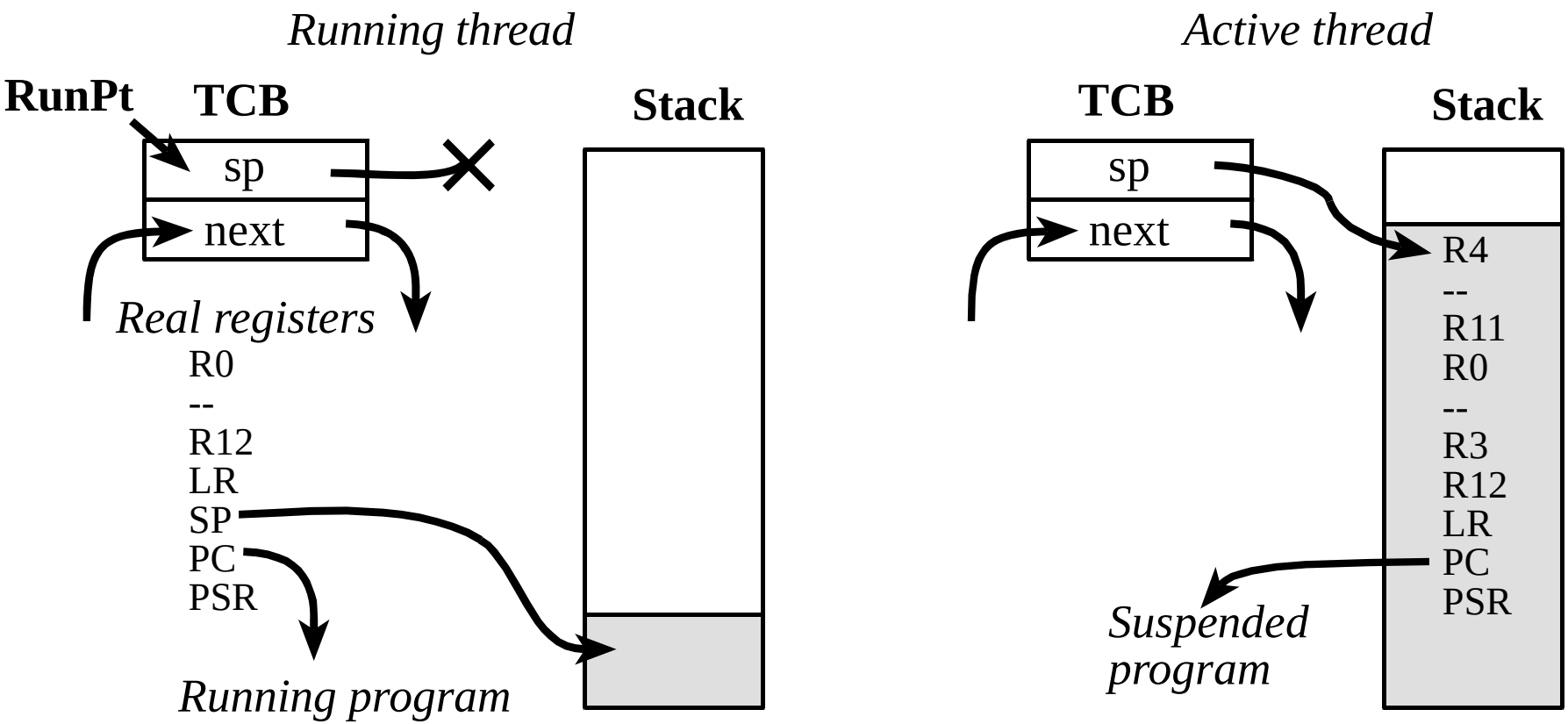
Run Testmain1
-Show TCB chain
-Show stacks
-Explain switch



Micrium\Software\uCOS-II\Ports\ARM-Cortex-M3\Generic\RealView

os_cpu_a.asm

Thread Switch



Assembly Thread Switch

```
SysTick_Handler         ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID I             ; 2) Make atomic
    PUSH  {R4-R11}      ; 3) Save remaining regs r4-11
    LDR   R0, =RunPt    ; 4) R0=pointer to RunPt, old
    LDR   R1, [R0]      ;    R1 = RunPt
    STR   SP, [R1]      ; 5) Save SP into TCB
    LDR   R1, [R1,#4]   ; 6) R1 = RunPt->next
    STR   R1, [R0]      ;    RunPt = R1
    LDR   SP, [R1]      ; 7) new thread SP; SP=RunPt->sp;
    POP   {R4-R11}      ; 8) restore regs r4-11
    CPSIE I             ; 9) tasks run enabled
    BX    LR            ; 10) restore R0-R3,R12,LR,PC,PSR
```

This code is in the book

Decisions

- MSP or MSP/PSP?
- Trap or regular function call?
  - How do you link OS to user code?
- Protection versus speed?
  - MSP/PSP
  - Check for stack overflow
  - Check for valid parameters

NVIC

- Set priorities
  - PendSV low
  - Timer1 high
- Trigger PendSV

```
NVIC_INT_CTRL_R = 0x10000000;
```

Page 158 of
tm4c123gh6pm.pdf

Launch

- Set SysTick period
- Set PendSV priority
- Using RunPt
  - Pop initialize Reg
- Enable interrupts
- Branch to user

To do first (1)

- Debugging
- Interrupts
- OS_AddThread
- Assembly
- NVIC
- PendSV
- OS_Suspend
- OS_Launch

To do last (2)

- Stack size
- FIFO size
- Timer1 period
- SysTick period
- Semaphores
- PSP
  - Just use MSP

Lab 2 Part 1 (1)

- Debugging
  - How to breakpoint, run to, dump, heartbeat
- Interrupts
  - How to arm, acknowledge, set vectors
  - What does the stack look like? What is in LR?
- OS_AddThread
  - Static allocation of TCBs and Stack
  - Execute 1,2,3 times and look at TCBs and Stack
- Assembly
  - PendSV, push/pull registers, load and store SP
  - Enable, disable interrupts
  - Access global variables like RunPt

Lab 2 Part 1(2)

- NVIC
  - Arm/disarm, priority
- PendSV
  - How to trigger
  - Write a PendSV handler to switch tasks
- OS_Suspend (scheduler and PendSV)
- OS_Launch *(this is hard)*
  - Run to a line at the beginning of the tread
  - Make sure TCB and stack are correct

Debugging tips

- Visualize the stacks
- Dumps and logs
- Logic analyzer