

# Interrupts (2)

Lecture 10

Yeongpil Cho

Hanyang University

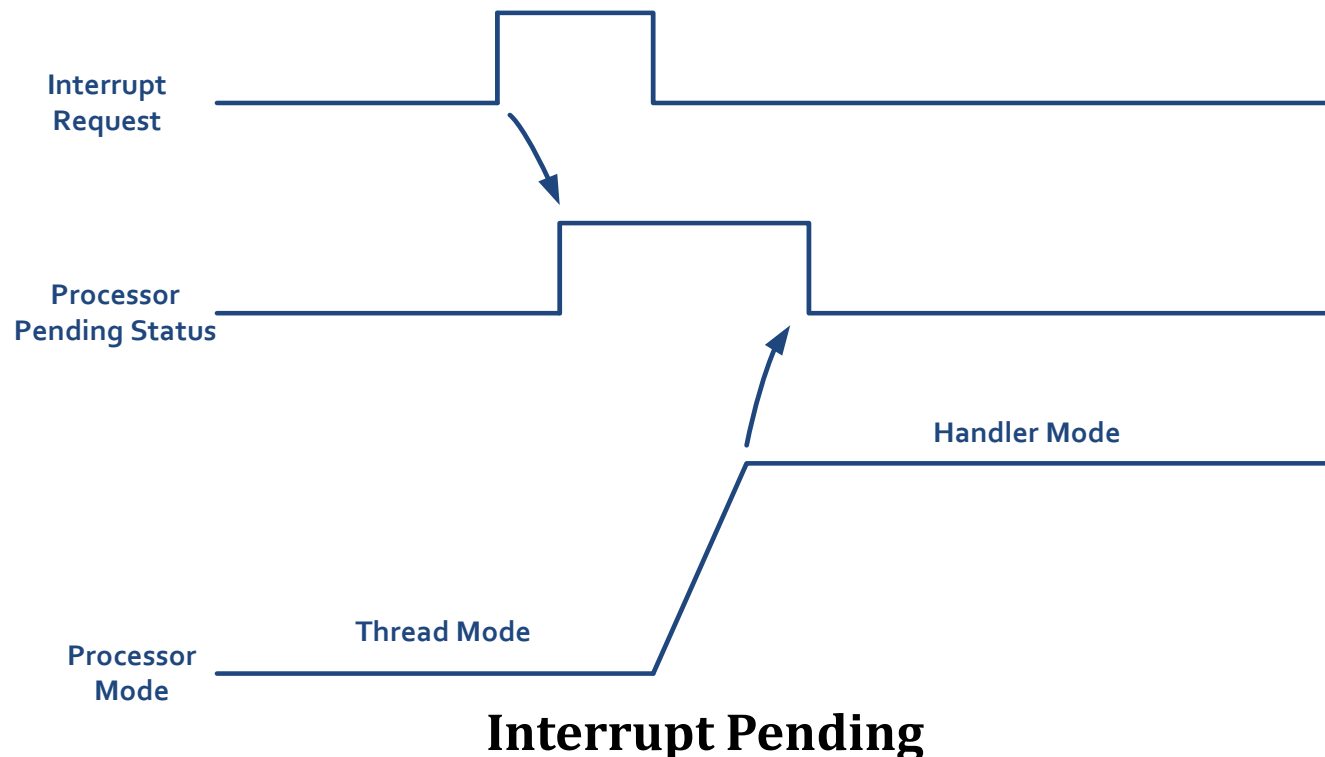
# Topics

- Pending Status
- Faults and Supervisor Call Exceptions

Pending Status

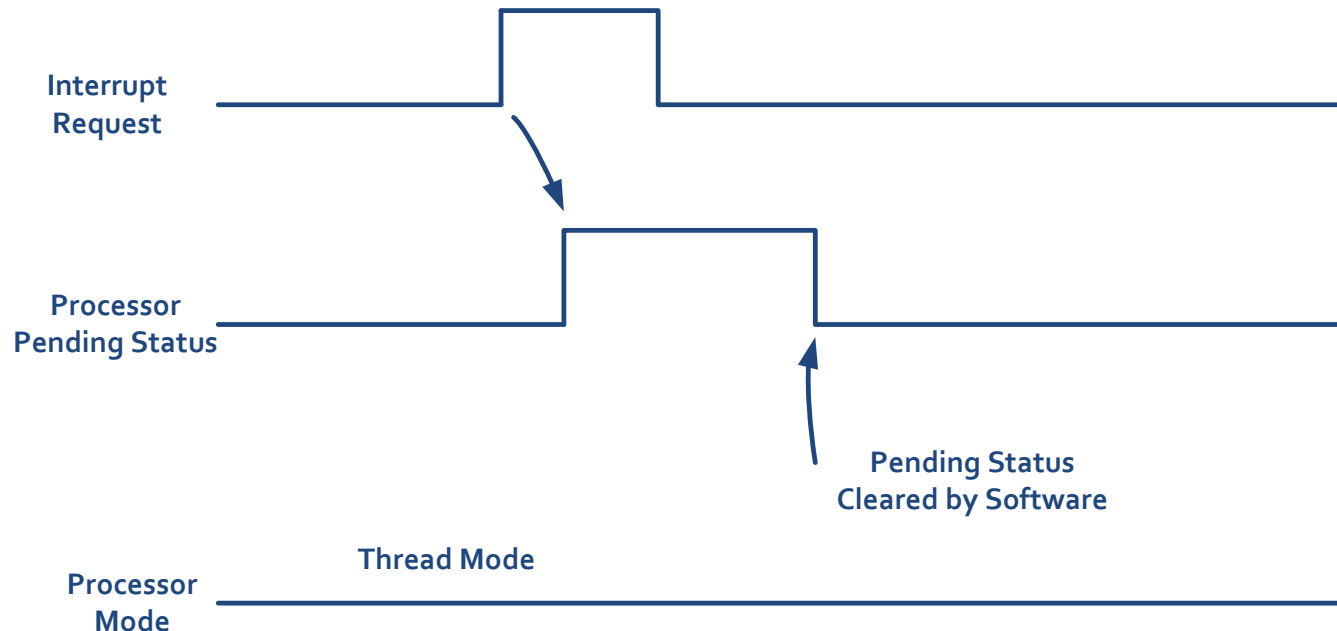
# Interrupt Inputs and Pending Behavior

- When an interrupt input is asserted, it can be pended
  - Interrupt-Set-Pending Registers (ISPRs) hold pending status
- Even if the interrupt source de-asserts the interrupt, the pended interrupt status will still cause the interrupt handler to be executed.



# Interrupt Inputs and Pending Behavior

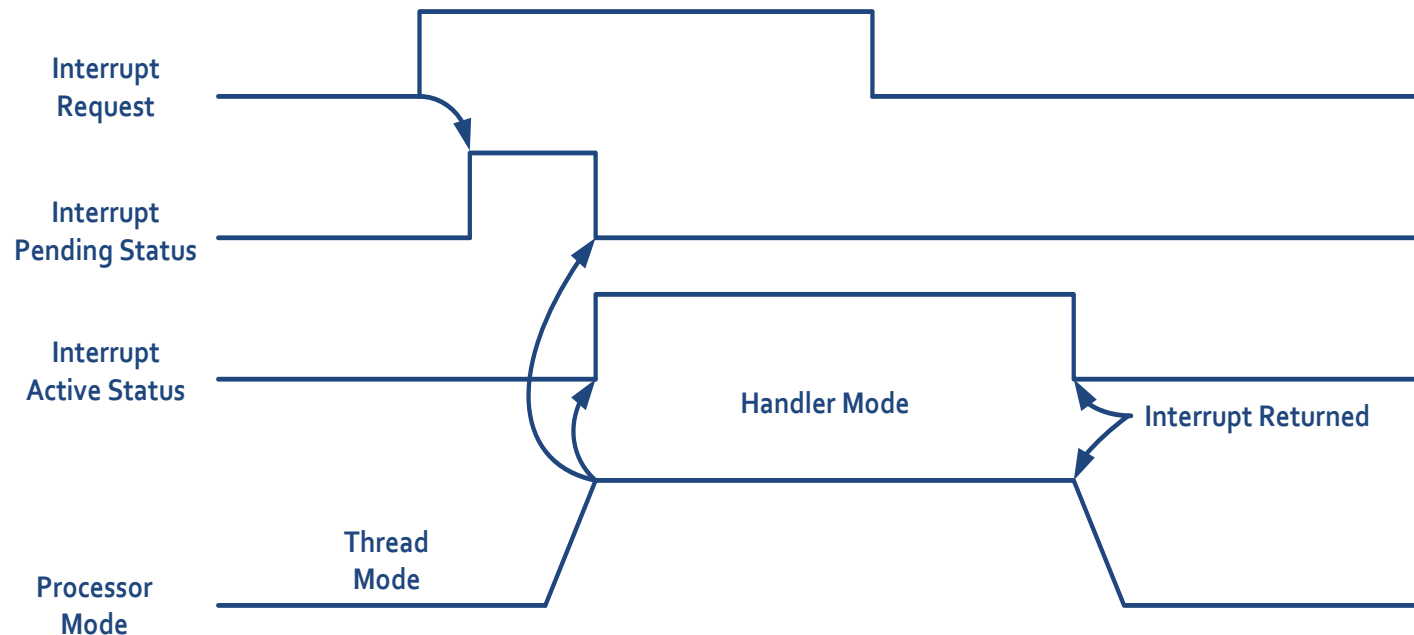
- If the pending status is cleared before the processor starts responding to the pending interrupt,
  - the interrupt can be canceled
- Note:
  - Pending status is automatically cleared when a handler is started.
  - You can even use pending status to raise software interrupts.



**Interrupt Pending Cleared Before Processor Takes Action**

# Interrupt Inputs and Pending Behavior

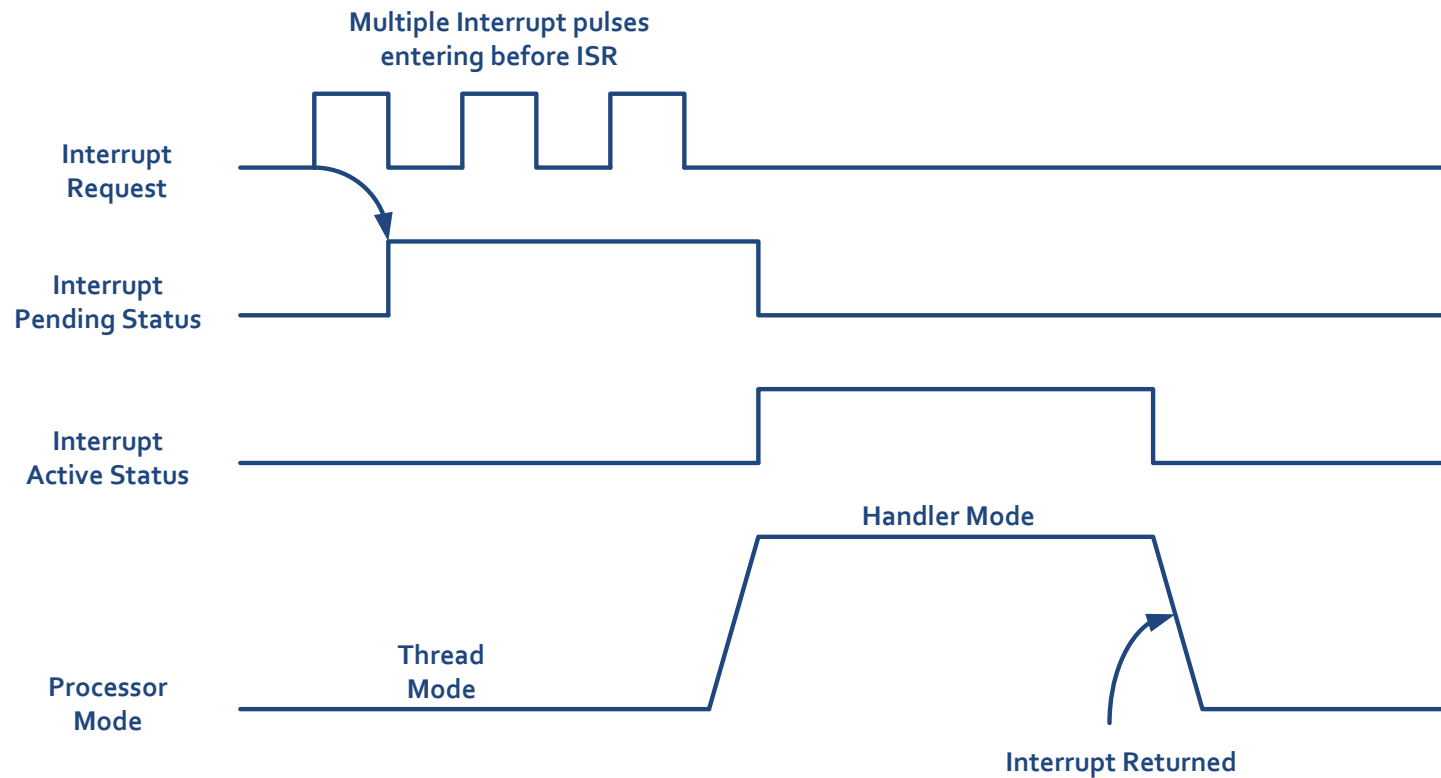
- When the processor starts to execute an interrupt
  - The interrupt becomes active
  - The pending bit will be cleared automatically.



**Interrupt Active Status Set as Processor Enters Handler**

# Interrupt Inputs and Pending Behavior

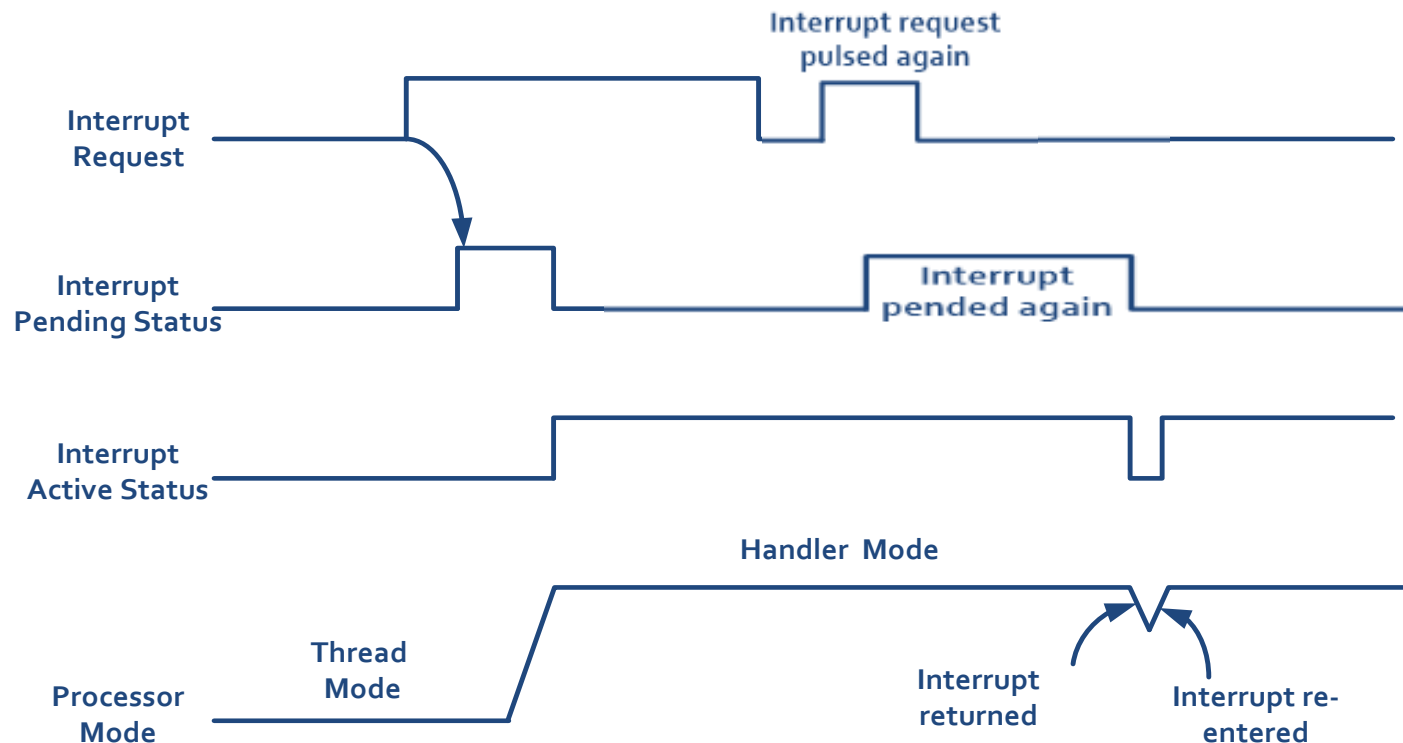
- If an interrupt is pulsed several times before the processor starts processing it,
  - it will be treated as one single interrupt request.



**Interrupt Pending Only Once, Even with Multiple Pulses Before the Handler**

# Interrupt Inputs and Pending Behavior

- If an interrupt is de-asserted and then pulsed again during the interrupt service routine,
  - it will be pended again.

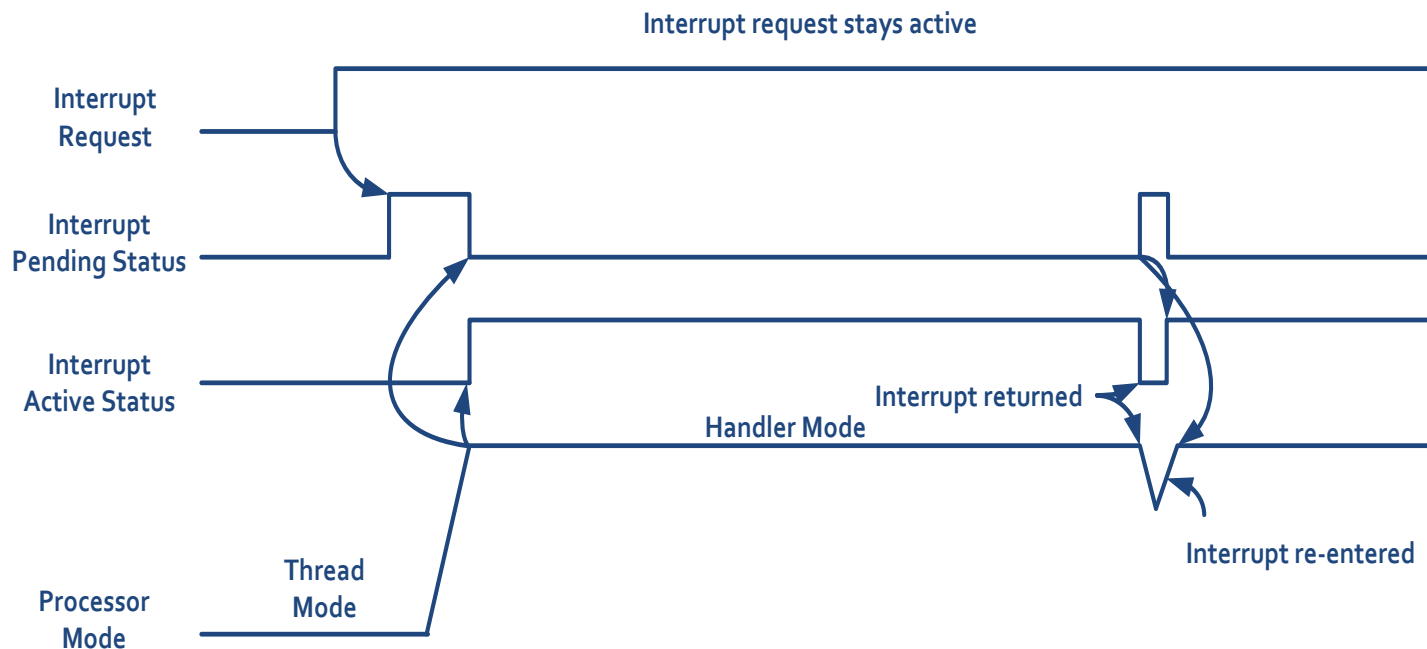


**Interrupt Pending Occurs Again During the Handler**



# Interrupt Inputs and Pending Behavior

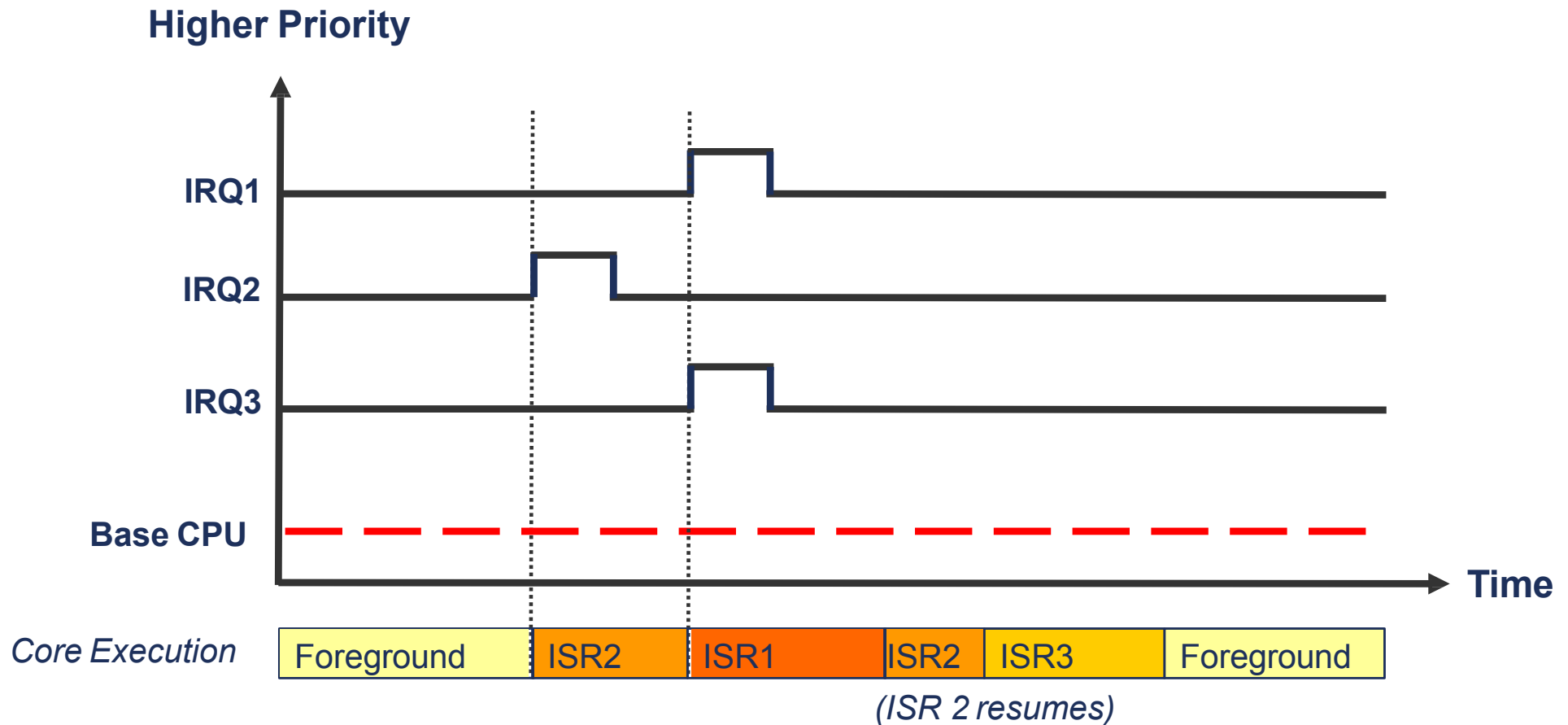
- If an interrupt source continues to hold the interrupt request signal active,
  - the interrupt will be pended again at the end of the interrupt service routine.



**Continuous Interrupt Request Pends Again After Interrupt Exit**

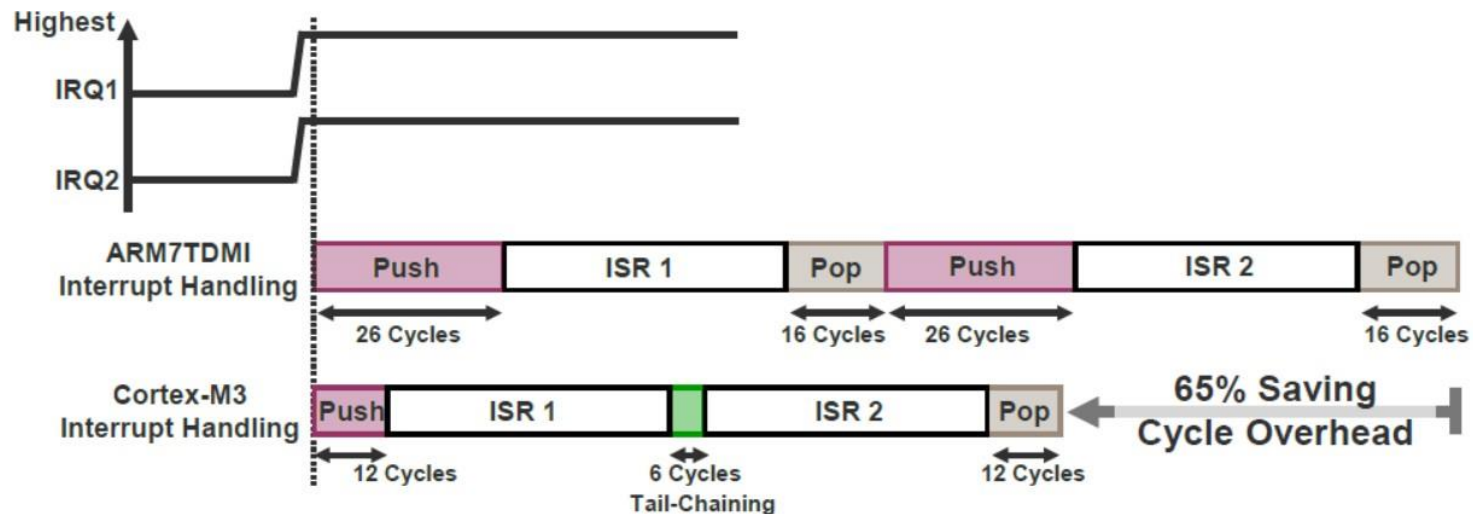
# Nested Interrupts

- NVIC supports nested interrupts as its names.



# Interrupt Response – Tail Chaining

- The processor skips the unstacking and stacking steps and enters the exception handler of the pended exception as soon as possible



## ARM7TDMI

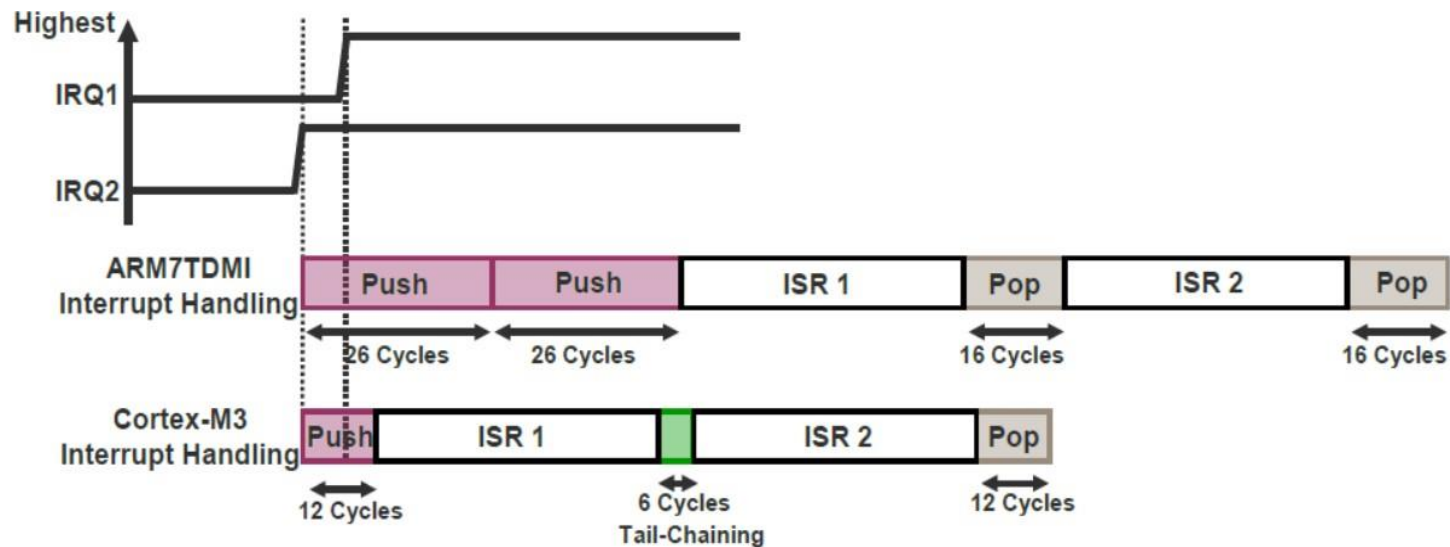
- 26 cycles from IRQ1 to ISR1 (up to 42 cycles if in LSM)
- 42 cycles from ISR1 exit to ISR2 entry
- 16 cycles to return from ISR2

## Cortex-M3

- 12 cycles from IRQ1 to ISR1 (Interruptible/Continual LSM)
- 6 cycles from ISR1 exit to ISR2 entry
- 12 cycles to return from ISR2

# Interrupt Response – Late Arriving

- An interrupt with a higher priority will be serviced first even if it arrives late during the stacking operation of the previous one.



## ARM7TDMI

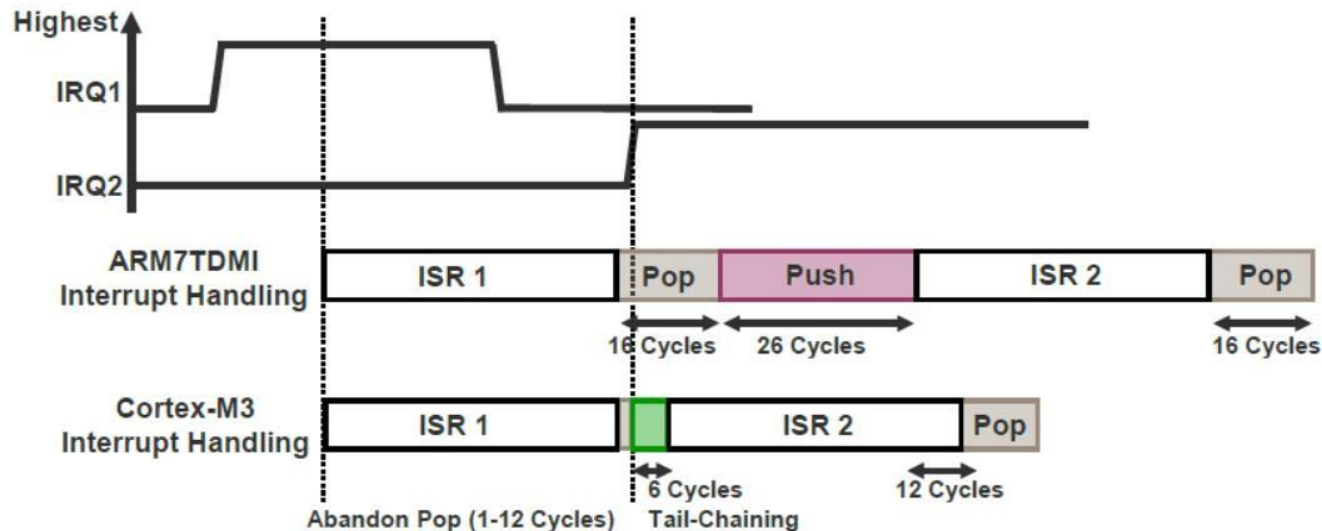
- 26 cycles to ISR2 entered
- Immediately pre-empted by IRQ1  
Additional 26 cycles to enter ISR1.
- ISR 1 completes  
Additional 16 cycles return to ISR2.

## Cortex-M3

- 12 cycles to ISR entry
- Parallel stacking & instruction fetch
- Target ISR may be changed until last cycle (PC is set)
- When IRQ1 occurs new target ISR set

# Interrupt Response – Pop Pre-emption

- If an exception arrives during the unstacking process of another exception, the unstacking would be abandoned and the next exception service begins



## ARM7TDMI

- Load multiple not interruptible
- Core must complete the recovery of the stack then re-stack to enter the ISR

## Cortex-M3

- Hardware un-stacking interruptible
- If interrupted only 6 cycles required to enter ISR2

# Faults and Supervisor Call Exceptions

# Faults and Supervisor Call Exceptions

- Bus Faults
  - Memory Management Faults
  - Usage Faults
  - Hard Faults
- 
- SVC
  - PendSV

# Bus Faults

- Bus faults are produced when an error response is received during a transfer on the AHB interfaces.
  - Prefetch abort (Instruction prefetch)
  - Data abort (data read/write)
  - Stacking error
  - Unstacking error
- Bus fault due to: (refer to Bus Fault Status Registers)
  - Attempt to access invalid memory region
  - Attempt to access target devices that are not ready yet
  - Attempt to access target devices with unsupported transfer sizes or privilege levels



# Memory Management Faults

- Common memory manage faults include:  
(refer to Memory management Fault Status Register)
  - Access to memory regions not defined in MPU setup.
  - Execute code from nonexecutable memory regions.
  - Writing to read-only regions.
  - An access in the user state to a region defined as privileged access only.

# Usage Faults

- Usage faults can be caused by:  
(refer to Usage Fault Status Register)
  - Undefined instructions
  - Coprocessor instructions
    - the Cortex-M3 processor does not support a coprocessor
  - Trying to switch to the ARM state
    - PC is set to a new value with the LSB equal to 0)
  - Invalid interrupt return
    - Link Register contains invalid/incorrect values
  - Unaligned memory accesses using multiple load/store
  - Returning to Thread mode with any active interrupts
- It is possible, by setting up certain control bits in the NVIC, to generate usage faults for:
  - Divide by zero
  - Any unaligned memory accesses

# Hard Faults

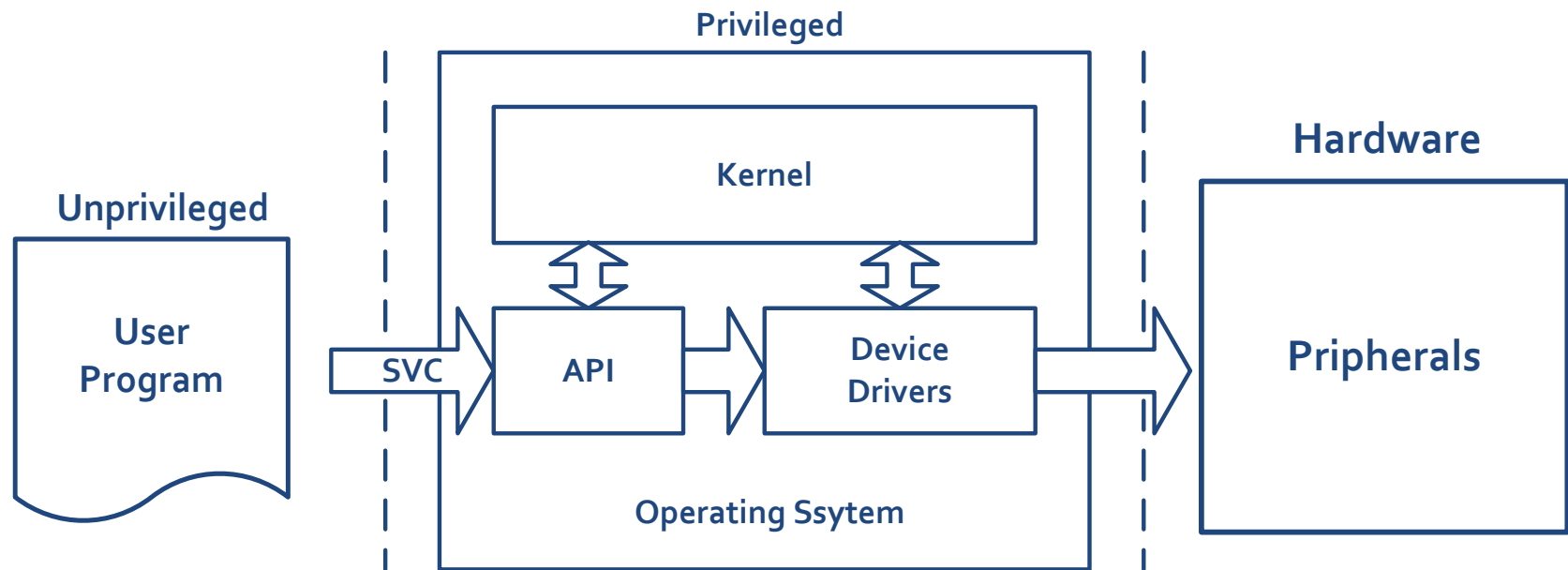
- Occurs when usage faults, bus faults, and memory management faults cannot be handled.
- Example (refer to Hard Fault Status Registers)
  - Fail to fetch the vector table due to bus faults
  - The three faults happen in a handler of other faults with higher priority.

# SVC and PendSV

- SVC (Supervisor Call) and PendSV (Pended Supervisor Call) are two exceptions targeted at operating systems.
- SVC
  - generated by SVC instruction
- PendSV
  - generated by PENDSVSET-bit of Interrupt Control and State Register (ICSR)

# SVC

- SVC can make software more portable because the user application does not need to know the programming details of the hardware.



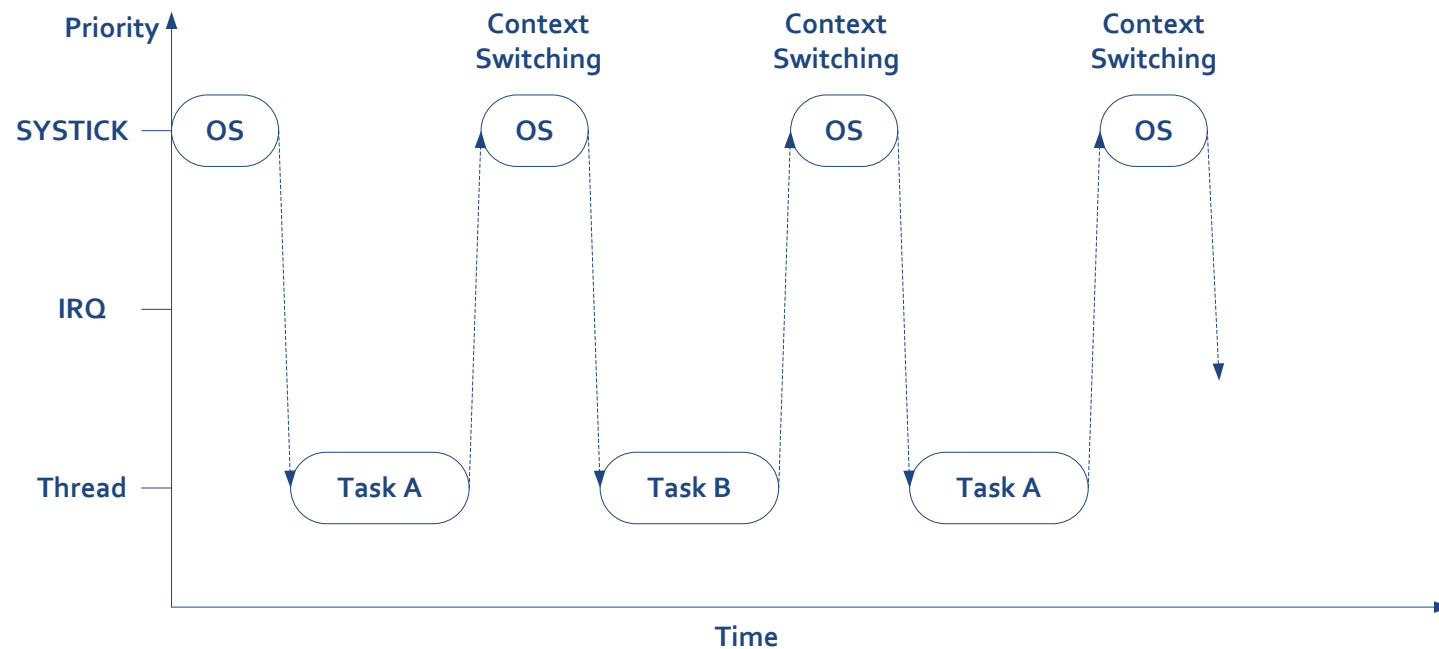
**SVC as a Gateway for OS Functions**

# PendSV

- PendSV has the lowest privileged level
- Useful for an OS to pend an exception so that an action can be performed after other important tasks (with higher priority).
  - A typical use is context switching.

# PendSV

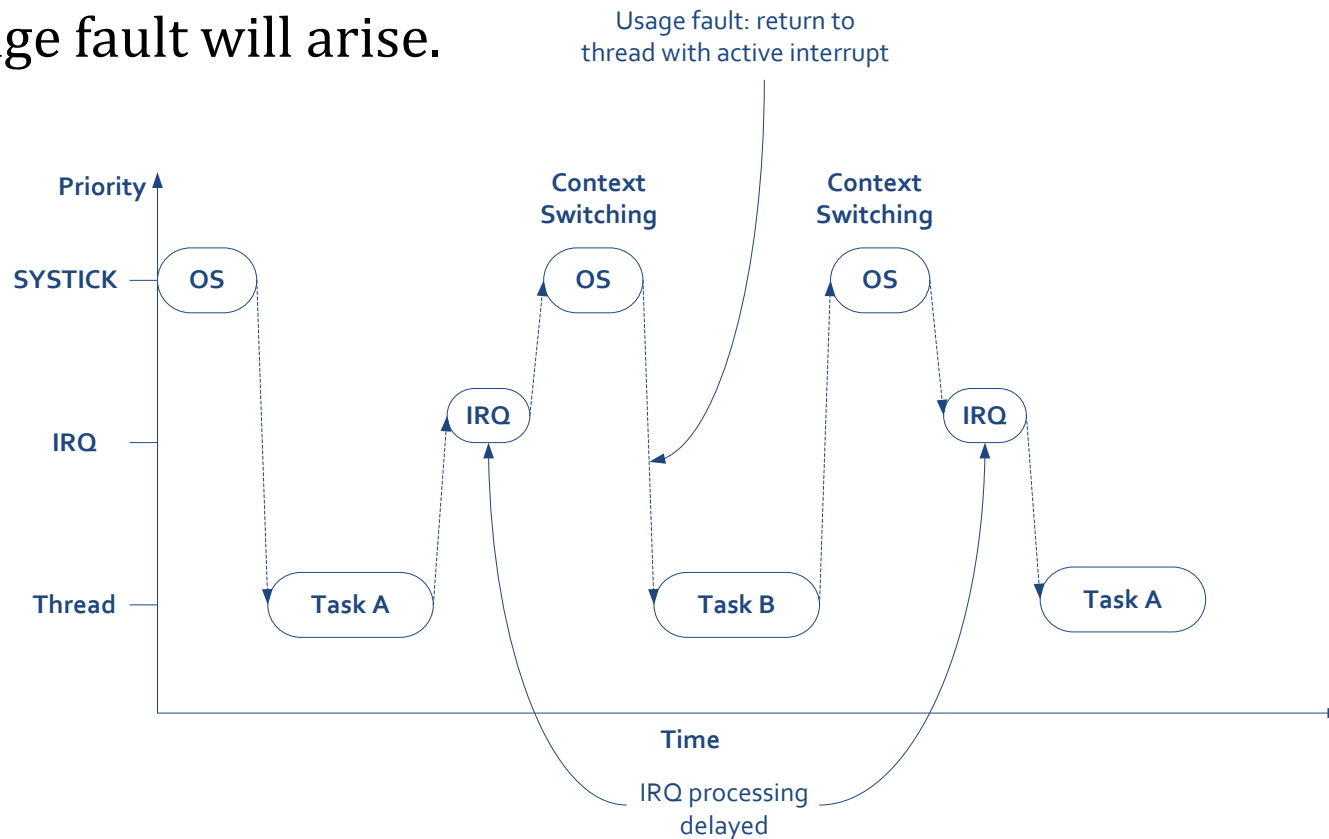
- Assumed that SYSTICK triggers context-switching



**A Simple Scenario Using SYSTICK to Switch Between Two Tasks**

# PendSV

- If SYSTICK happens during an interrupt is handled and context-switching is triggered,
  - Usage fault will arise.



## Problems at Context Switching at the IRQ



# PendSV

- One solution
  - OS performs context switching only when none of interrupt handlers are being executed.
  - The context-switching can be delayed so long if the interrupts arise with the frequency close to that of the SYSTICK.

# PendSV

- Better solution: using PendSV
  - Make SYSTICK exceptions do not perform context-switching by themselves.
    - Only pends PendSV exception with a lowest priority
  - PendSV handler will be executed and performs context-switching safely after all interrupts with higher priorities handled.

