Introduction to Real-Time Kernels

Scheduling and Context Switching

2013-07-17

Jean J. Labrosse

CEO, **Micriμm**

Outline

- **Scheduling**
  - What is scheduling?
  - What is round-robin scheduling?
  - When does scheduling happen?
  - What is the outcome?
- **Context Switching**
  - What is a task's context?
  - How does context Switching work?
- **Servicing Interrupt**
  - Priorities of interrupts
  - Anatomy of an ISR
  - Kernel Aware vs Non-Kernel Aware ISRs

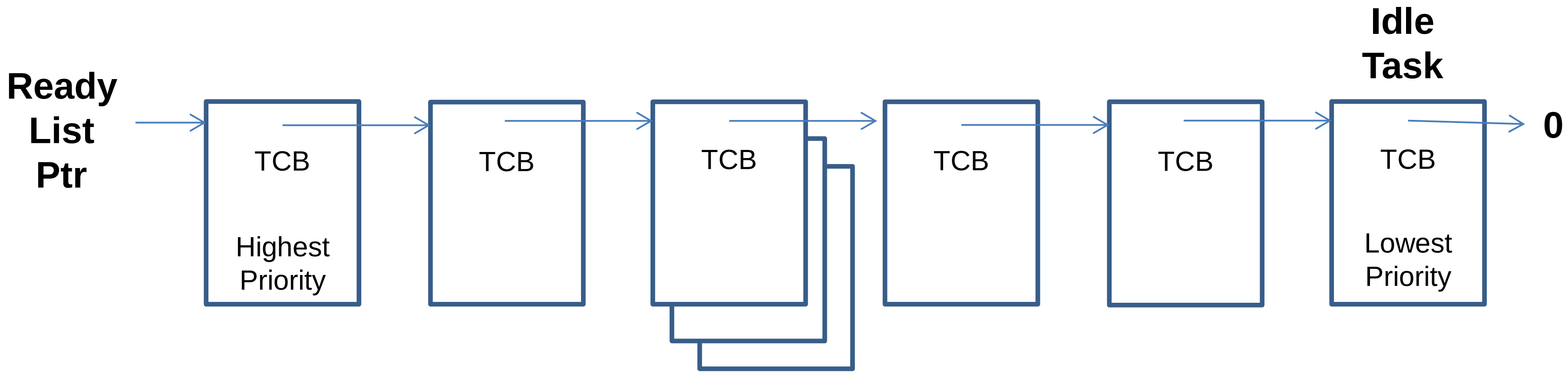
Task Priorities

- **Each task is assigned a priority when it's created**
  - Based on the importance of the task in your application
  - In general high priority task are assigned to the functions of your product
    - Control systems, communications, user interface, etc.

- **Always run the highest priority task ready**

- **Ready-to-run tasks are placed in a 'Ready-list'**

The Ready List



What is Scheduling?

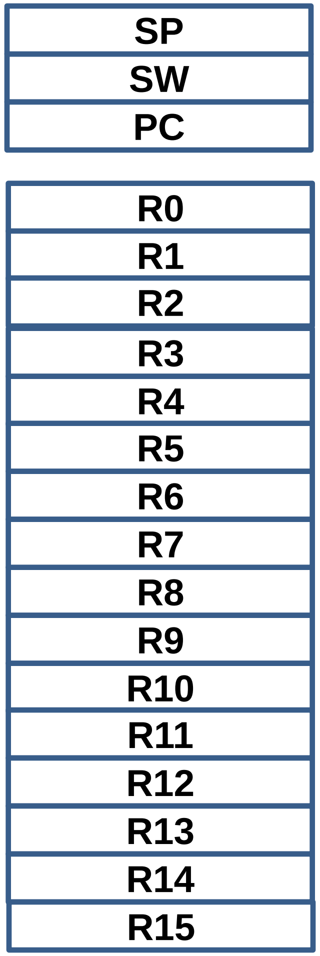
- **Deciding whether a more important task needs to run**

- **When does scheduling occur?**
  - When a task decides to wait for time to expire
  - When a task or an ISR ‘signals’ or notifies another task about an event
  - When a task or an ISR ‘sends’ a message to another task
  - When the priority of a task is changed
  - When a task is suspended

- **What’s the outcome**
  - Possibly a ‘Context Switch’

Context Switch

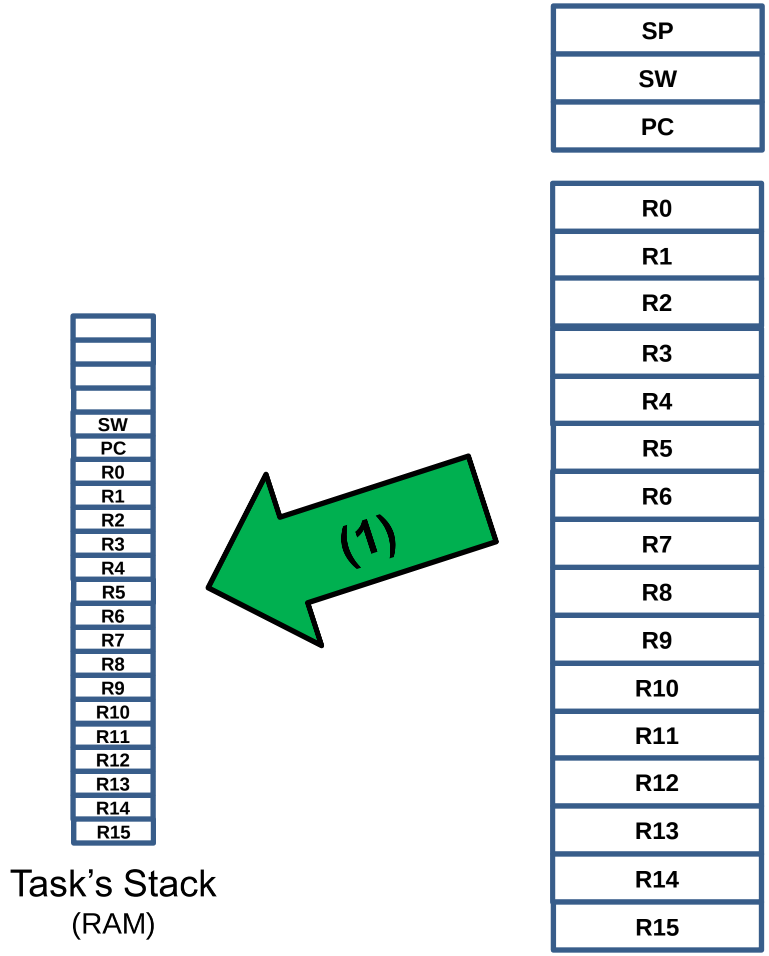
CPU Registers



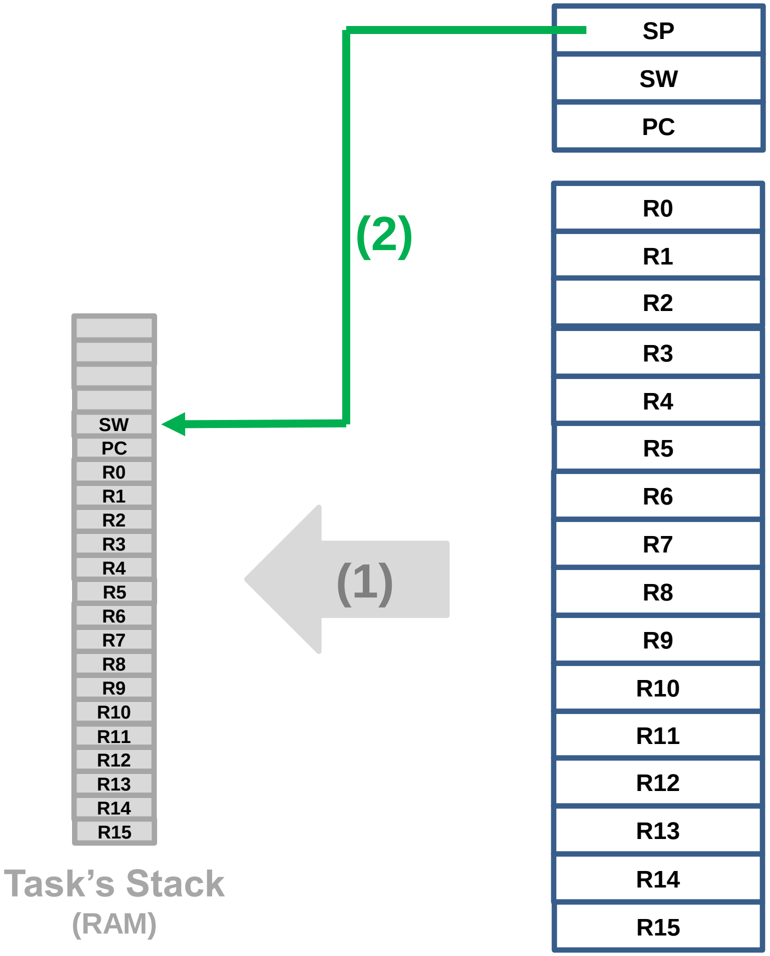
SP
SW
PC

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15

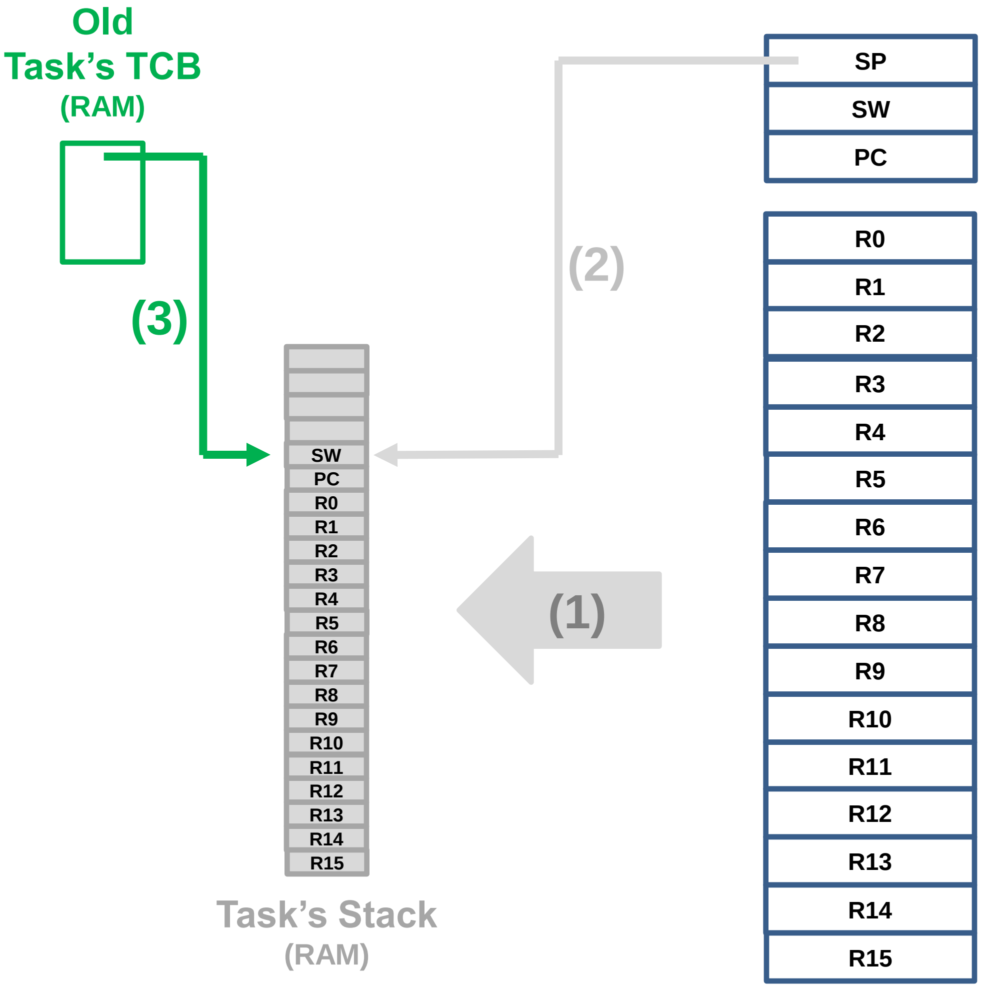
Context Switch



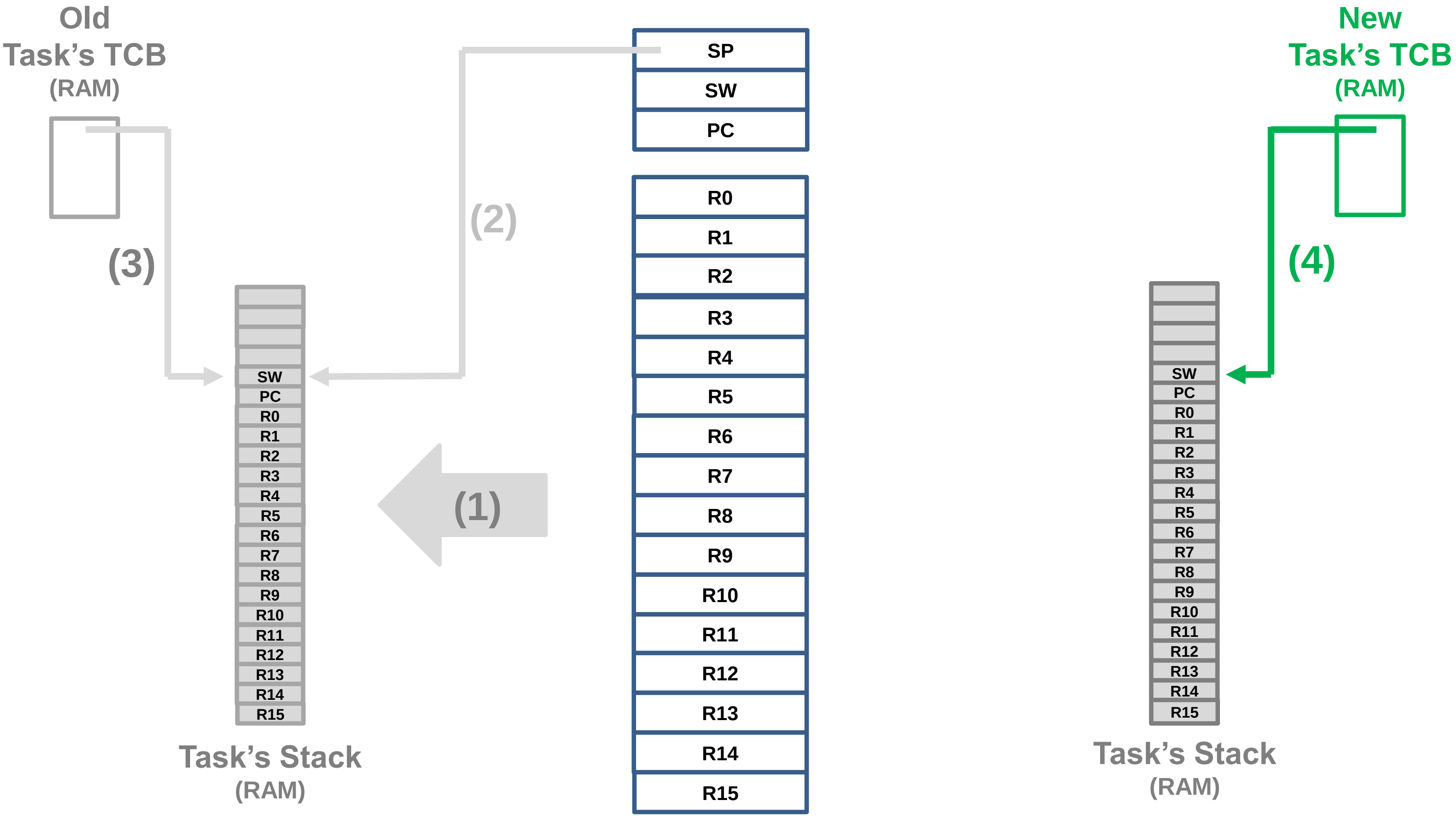
Context Switch



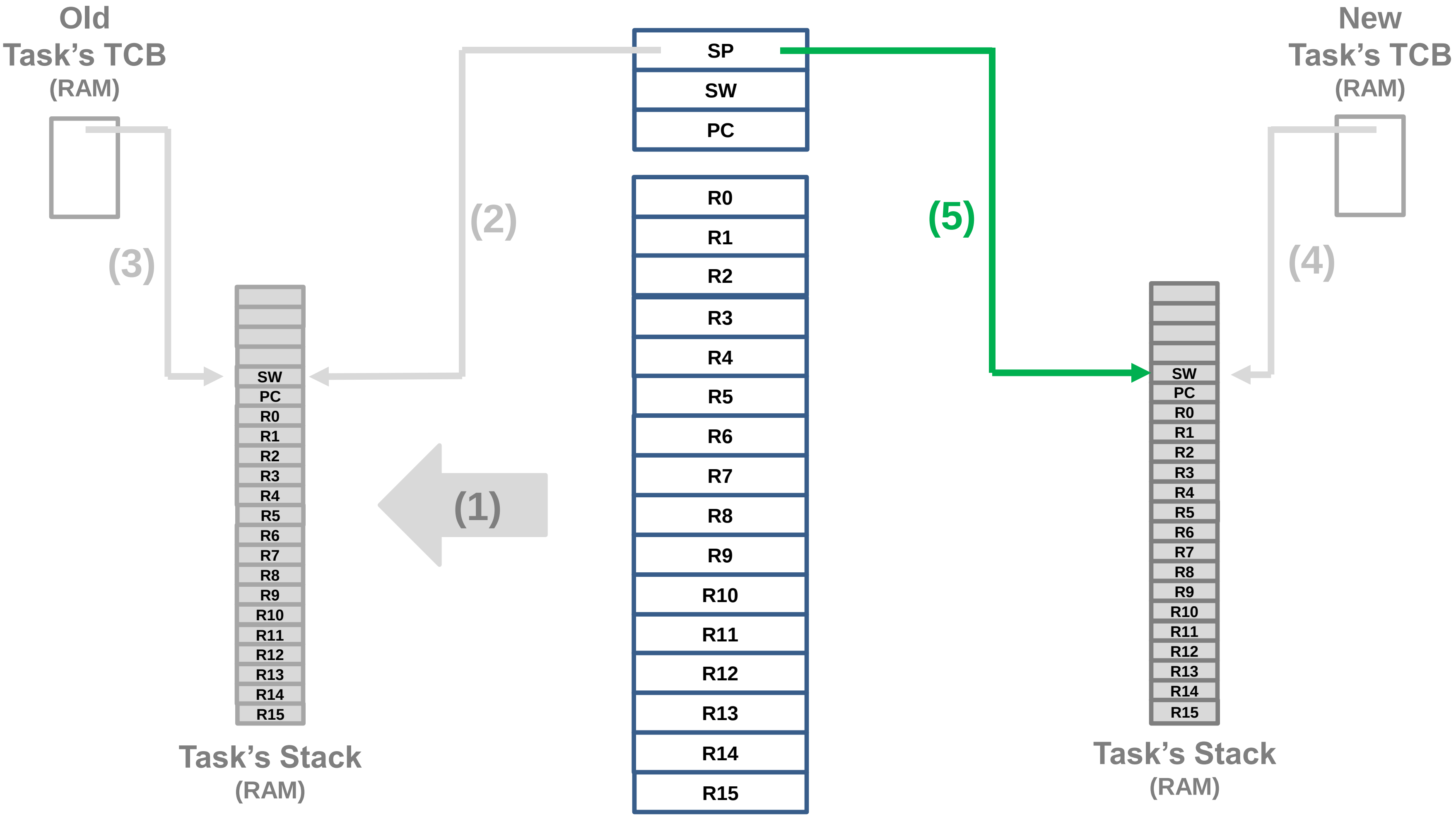
Context Switch



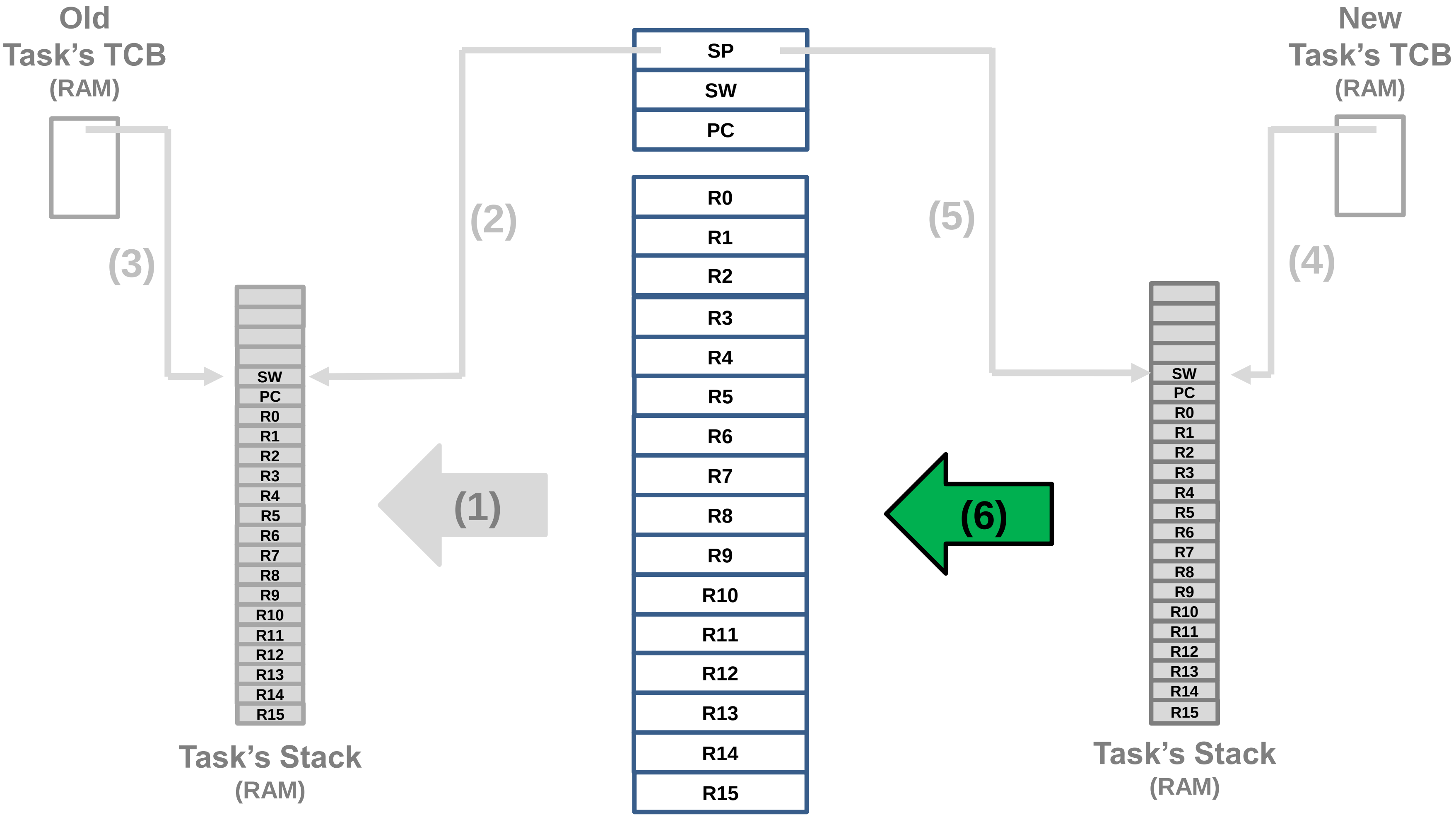
Context Switch



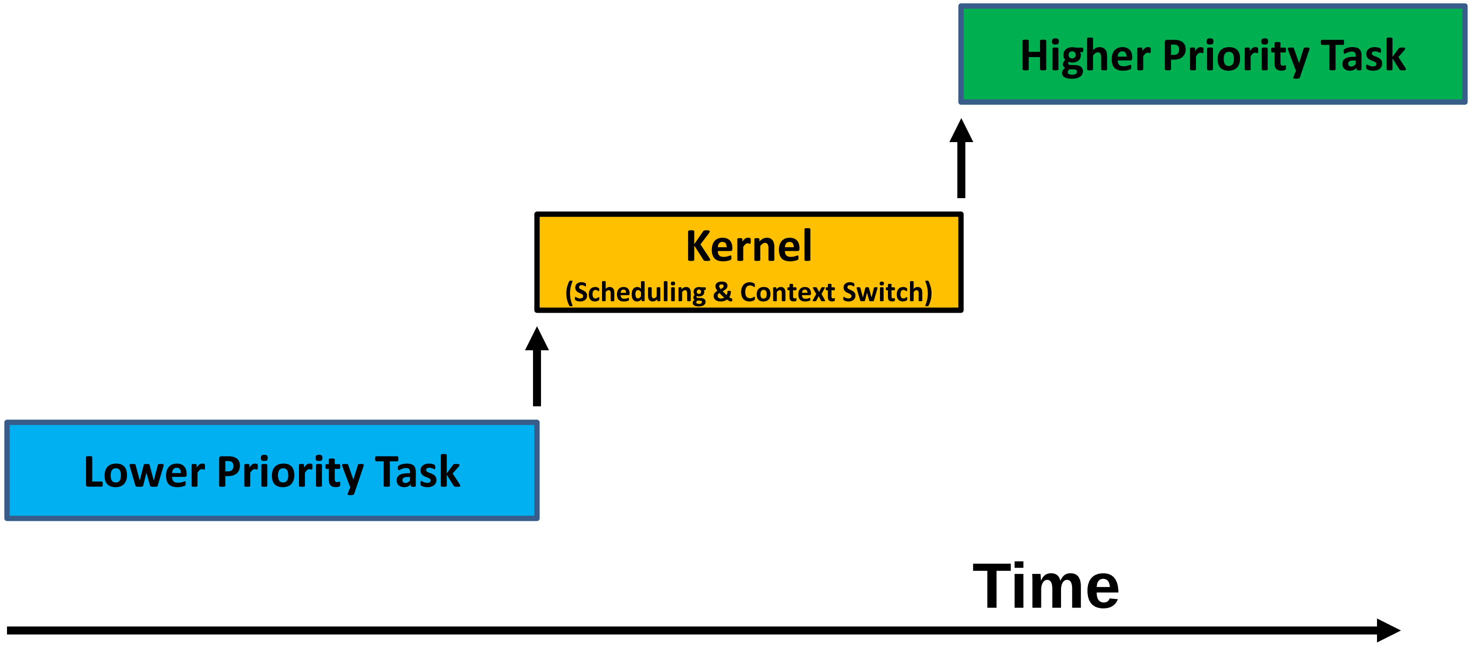
Context Switch



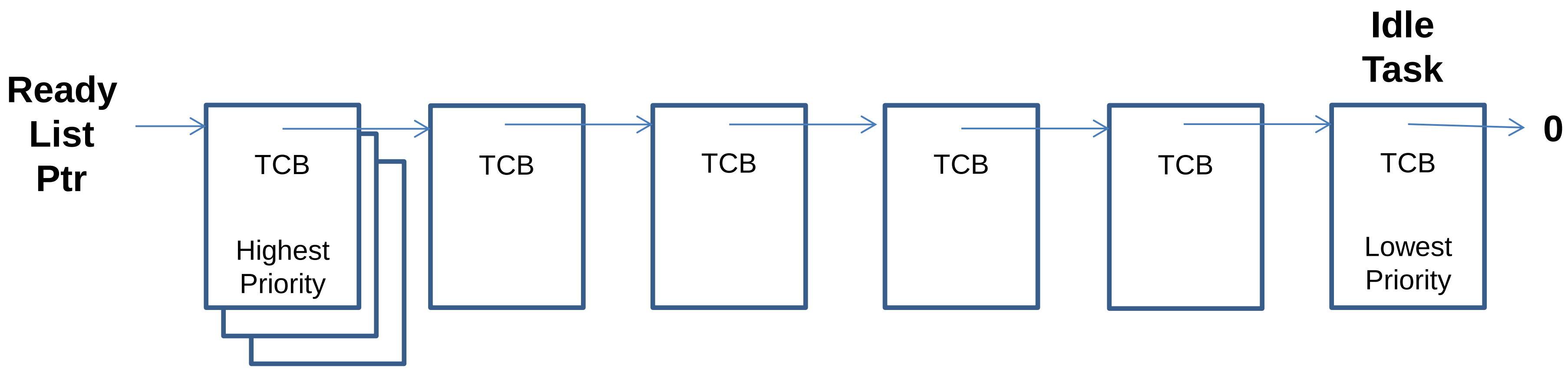
Context Switch



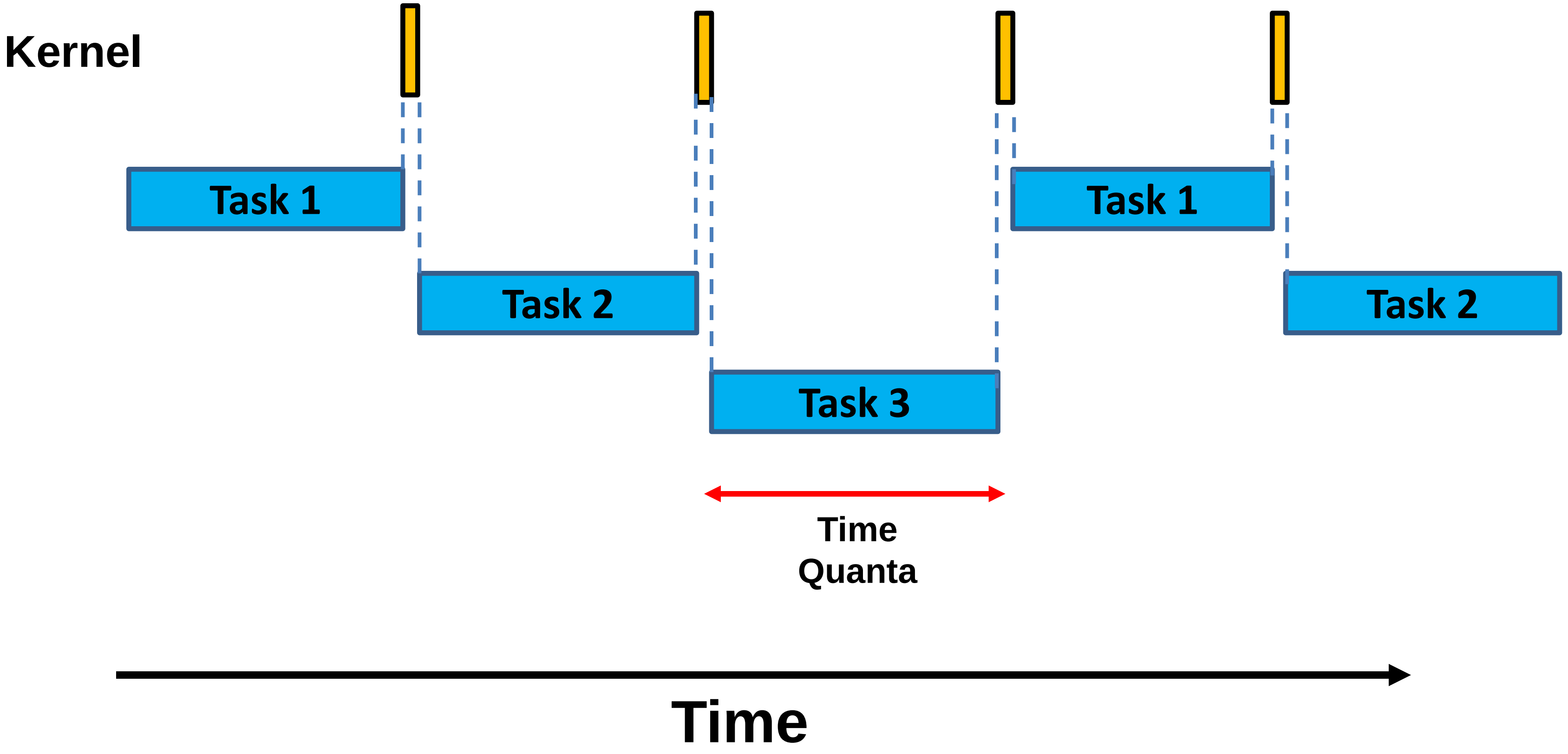
Context Switch



Round-Robin Scheduling



Round-Robin Scheduling



Servicing Interrupts

- **Interrupts are always more important than tasks.**

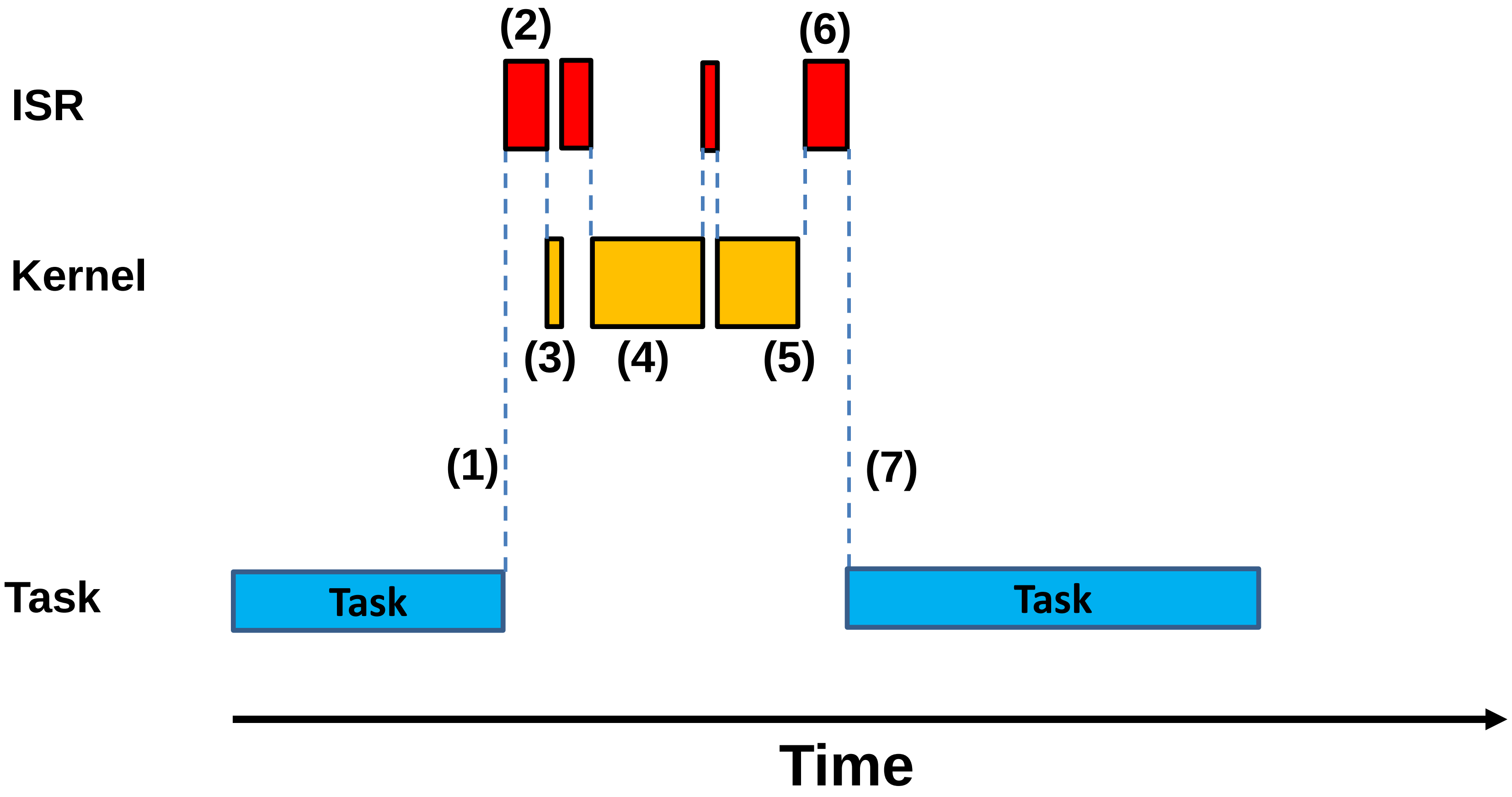
- **Interrupts are always recognized unless …**
  - … Your application disables interrupts
  - … Or, the kernel disables interrupts

- **Interrupt Service Routines (ISRs) should be kept as short as possible**

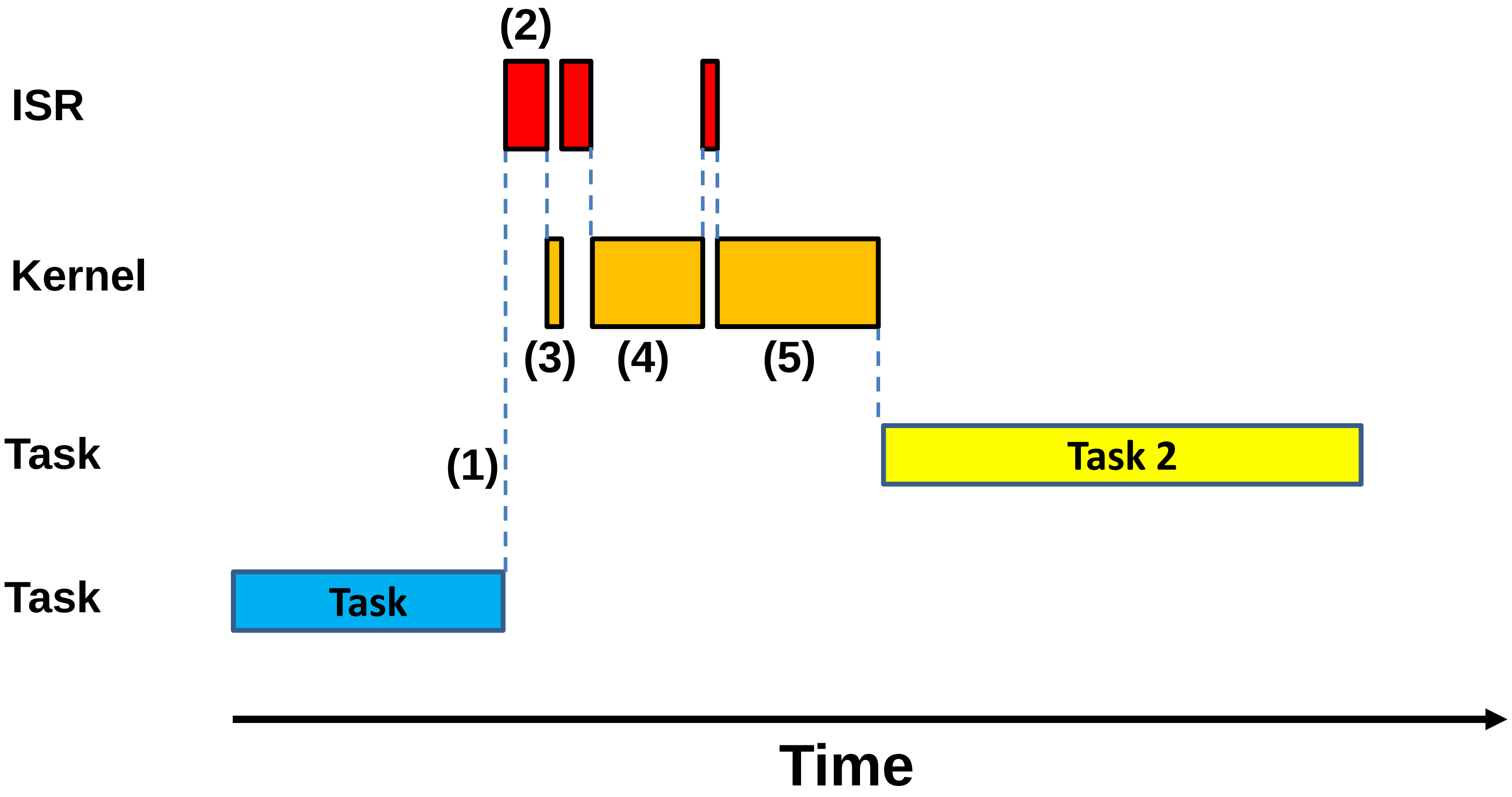
Interrupt Handlers

```
MyISR:                                                    (1)
   Save CPU registers;                                    (2)
   Notify the kernel that an interrupt is being processed; (3)
   Signal a task that its event occurred;                 (4)
   Notify the kernel that the ISR is done;                (5)
   Restore saved CPU registers;                           (6)
   Return from interrupt;                                 (7)
```

Resuming the interrupted task



Running a more important task



Kernel Aware vs Non-Kernel Aware Interrupts

- **Interrupts that tasks are waiting for are called 'Kernel Aware' interrupts.**
  - Most of the ISRs in an application will be Kernel Aware

- **Interrupts that don't need to notify tasks are 'Non-Kernel Aware' interrupts.**
  - e.g. An ISR that simply reloads the value of a PWM register
  - Pseudo-code:

```
MyISR:
    Save CPU registers;
    Service interrupting device;
    Restore CPU registers;
    Return from Interrupt;
```

Next Class

- **The Tick ISR**
  - Time Delays
  - Timeouts
- **Soft Timers**
- **Resource sharing and Mutual Exclusion**
  - Priority Inversions
  - Priority Inheritance