Digi-Key CORPORATION CONTINUING EDUCATION CENTER

Introduction to Real-Time Kernels

What is a Real-Time Kernel?

2013-07-15

Jean J. Labrosse

CEO, **Micriμm**

Outline

Foreground/Background Systems

Real-Time Kernels

- What is it?
- A subset of an Real-Time Operating System (RTOS)
- Multitasking
- Preemptive Kernel
- Benefits and Drawbacks

Foreground/Background
(a.k.a. Super Loops)

Background (i.e. Tasks)

```
int  main (void)
{
    Perform initializations;
    while (1) {
        ADC_Read();
        DI_Read();
        USB_Packet();
        LCD_Update();
        Audio_Decode();
        File_Write();
        Etc;
    }
}
```

Foreground (i.e. ISRs)

```
void  USB_ISR (void)
{
    Clear interrupt;
    Read packet;
}
```

Foreground/Background

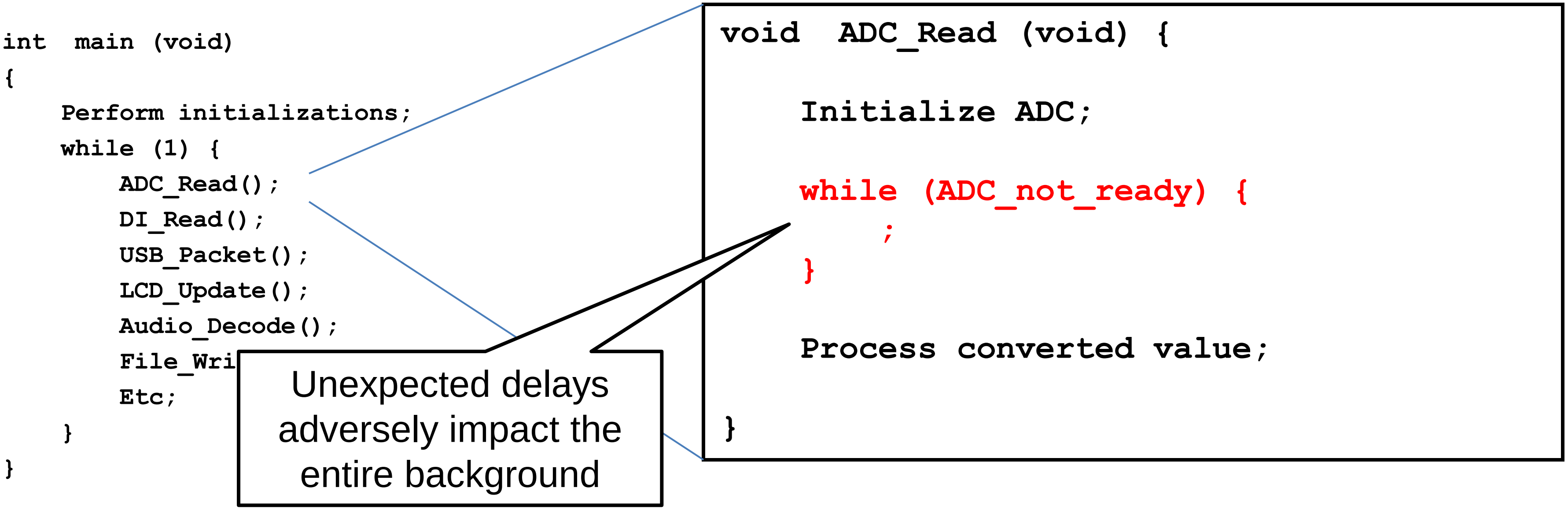
Benefits

- **No upfront cost**
- **Minimal training required**
  - Developers don’t need to learn a kernel’s API
- **No additional memory to accommodate the kernel**
  - There’s a small amount of overhead with a kernel
- **Minimal interrupt latency**

Foreground/Background

Drawbacks (1)

- **Difficult to ensure that each operation will meet its deadline**
  - All the code in the 'background' has the same priority
  - One function can affect the responsiveness of the whole system!



```
int  main (void)
{
    Perform initializations;
    while (1) {
        ADC_Read();
        DI_Read();
        USB_Packet();
        LCD_Update();
        Audio_Decode();
        File_Wri
        Etc;
    }
}
```

```
void  ADC_Read (void) {

    Initialize ADC;

    while (ADC_not_ready) {
          ;
    }

    Process converted value;

}
```

Unexpected delays adversely impact the entire background

Foreground/Background
Drawbacks (2)

- **High-priority code must be placed in ISRs**
  - Long ISRs will impact the responsiveness of the system
  - Coordinating the foreground and background is difficult
    - The developer must implement foreground-background communications services

```
while (1) {
    ADC_Read();
    SPI_Read();
    USB_Packet();
    LCD_Update();
    Audio_Decode();
    File_Write();
}
```

```
void  USB_ISR (void) {
    Clear interrupt;
    Read packet;
}
```

If a USB packet is received **immediately after** this function returns, the response time will be lengthy.

Foreground/Background
Drawbacks (3)

- **Code is difficult to maintain with multiple developers**
  - The efforts of all developers must be closely coordinated in order to ensure that proper application’s timing requirements will be met
- **Expanding the application can prove difficult**
  - … Even with a single developer
  - Changes to one portion of the code may negatively impact the rest of the code
  - As the application grows, inefficient use of resources may not be avoidable

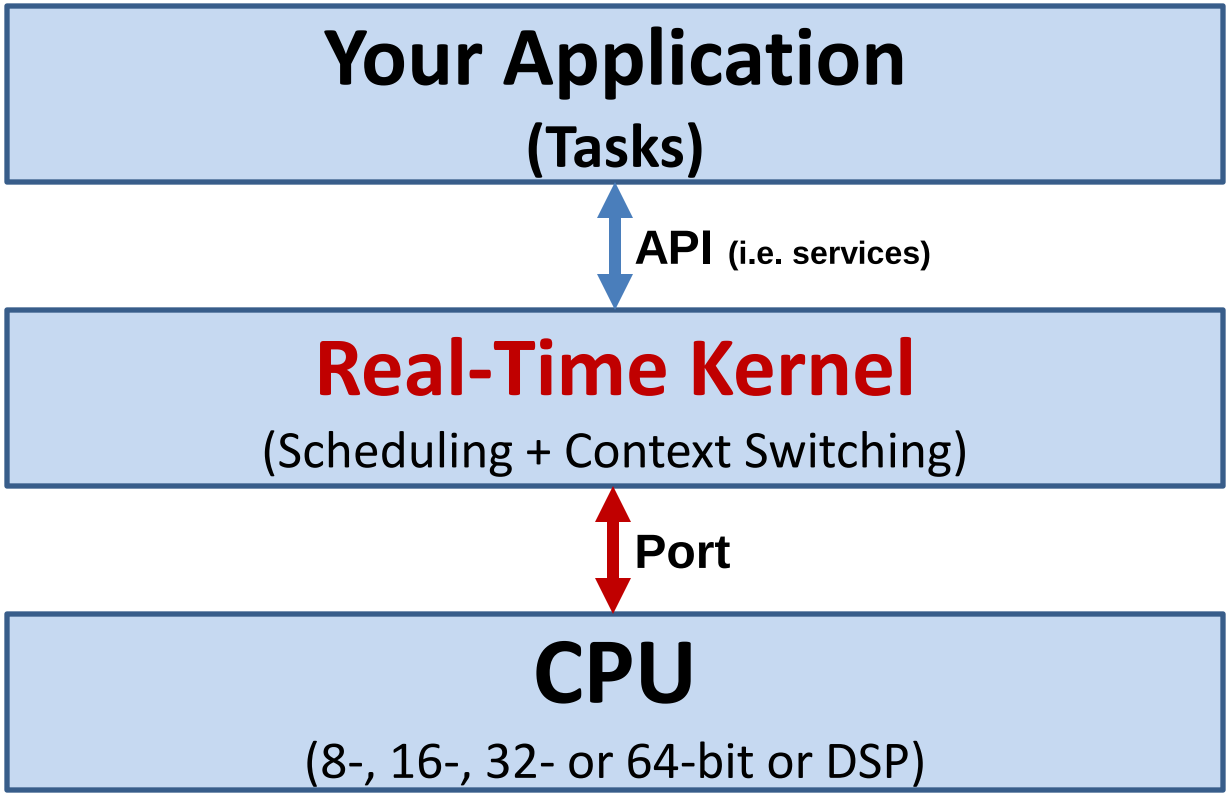
Real-Time Kernel

What is it?

- **It's software**
  - That manages the *time* and *resources* of a CPU or MCU-based application
  - It ensures that more important code runs before less important code!
- **It provides 'Multitasking' capabilities**
  - You break down the application into smaller tasks
  - You tell the kernel which tasks are more important
    - The kernel will try to satisfy your requirements at run-time
  - Each task it 'thinks' it has its own CPU
- **It provides 'Services'**
  - Task management, resource sharing, time management, synchronization, communications, etc.

Real-Time Kernels

Your code sees an API



Your Application

(Tasks)

API **(i.e. services)**

Real-Time Kernel

(Scheduling + Context Switching)

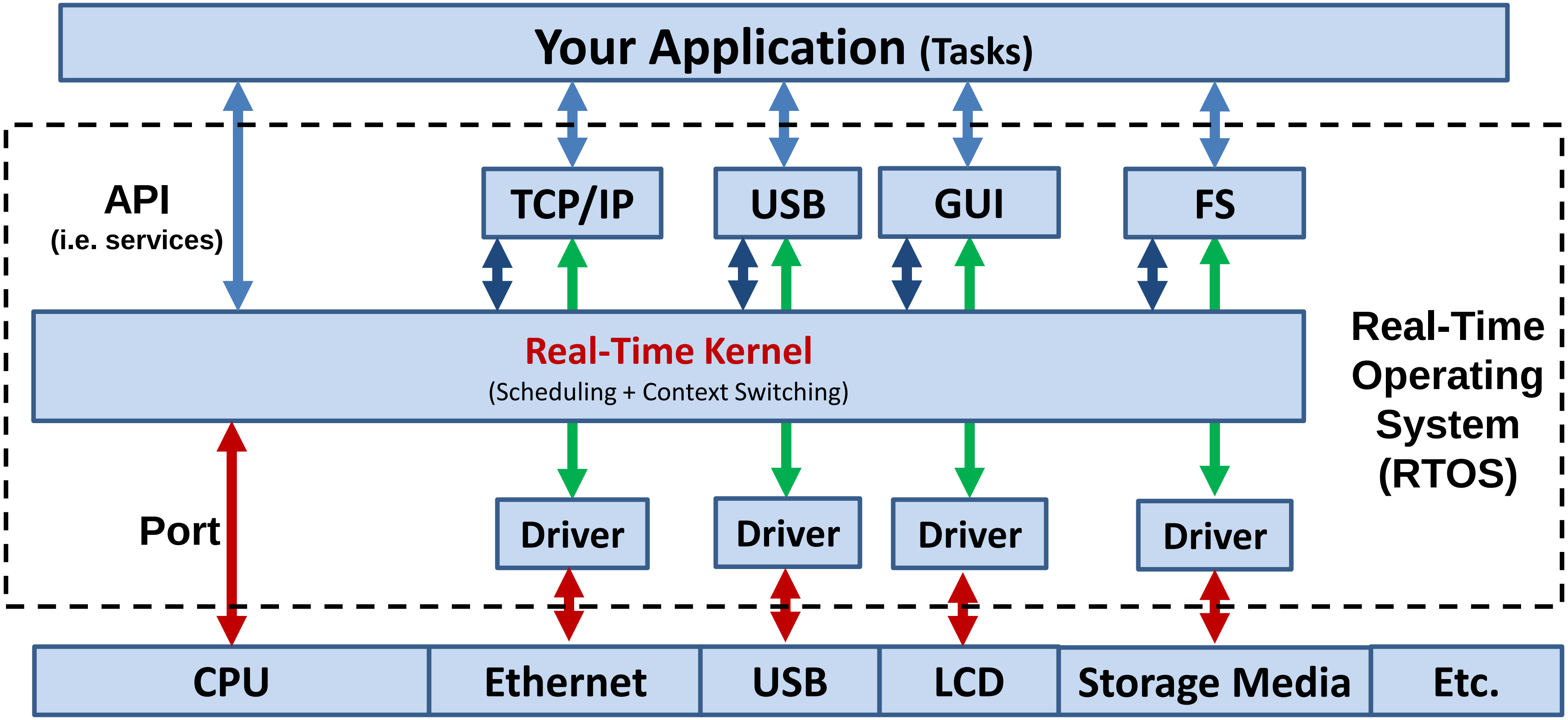
Port

CPU

(8-, 16-, 32- or 64-bit or DSP)

Real-Time Kernels

Are a subset of an RTOS



Your Application (Tasks)

API
(i.e. services)

TCP/IP

USB

GUI

FS

Real-Time Kernel
(Scheduling + Context Switching)

Real-Time Operating System (RTOS)

Port

Driver

Driver

Driver

Driver

CPU

Ethernet

USB

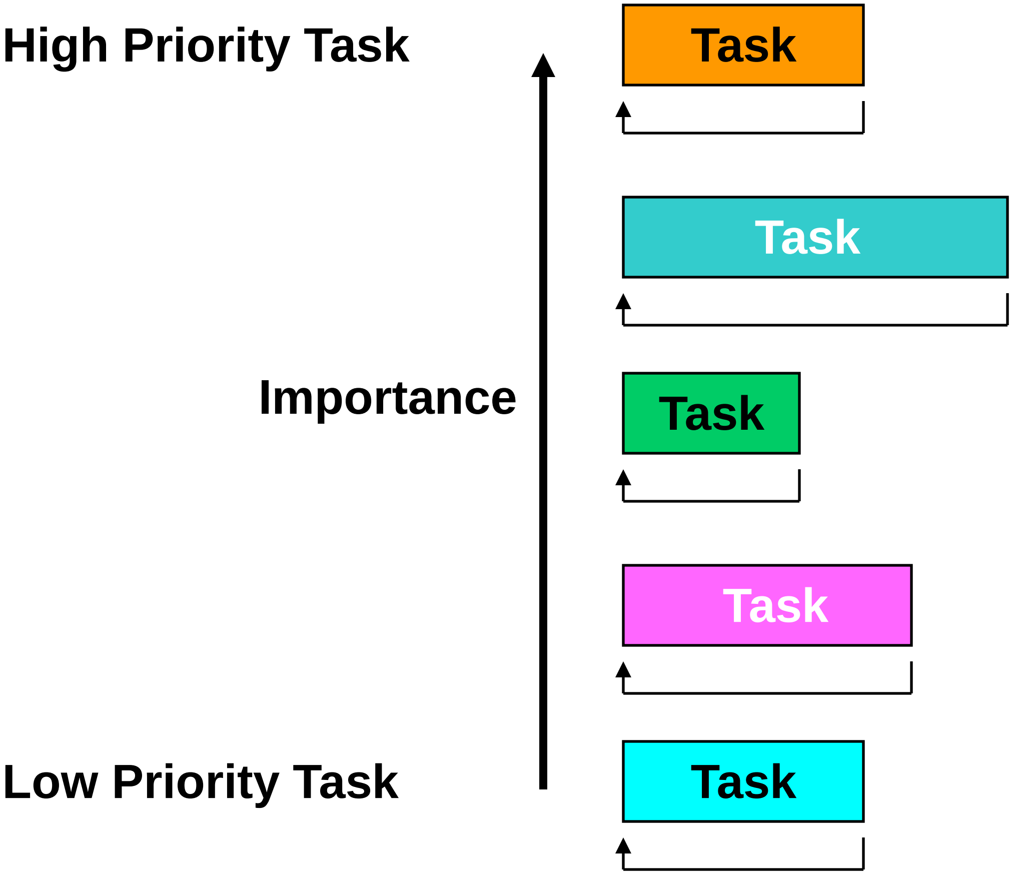
LCD

Storage Media

Etc.

Multitasking

Splitting an application into Tasks (2)



High Priority Task

Importance

Low Priority Task

Task

Task

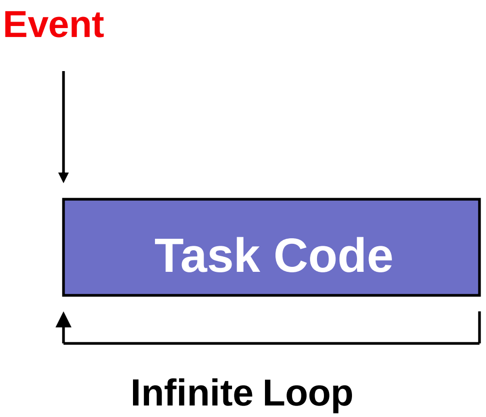
Task

Task

Task

Multitasking

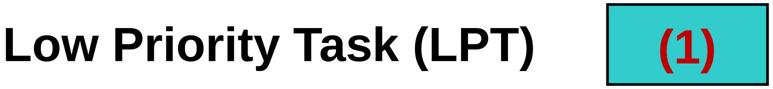
Splitting an application into Tasks (1)



```
MyTask ()
{
     while (1) {
          Wait for Event;
          Task code;          // YOUR code
     }
}
```

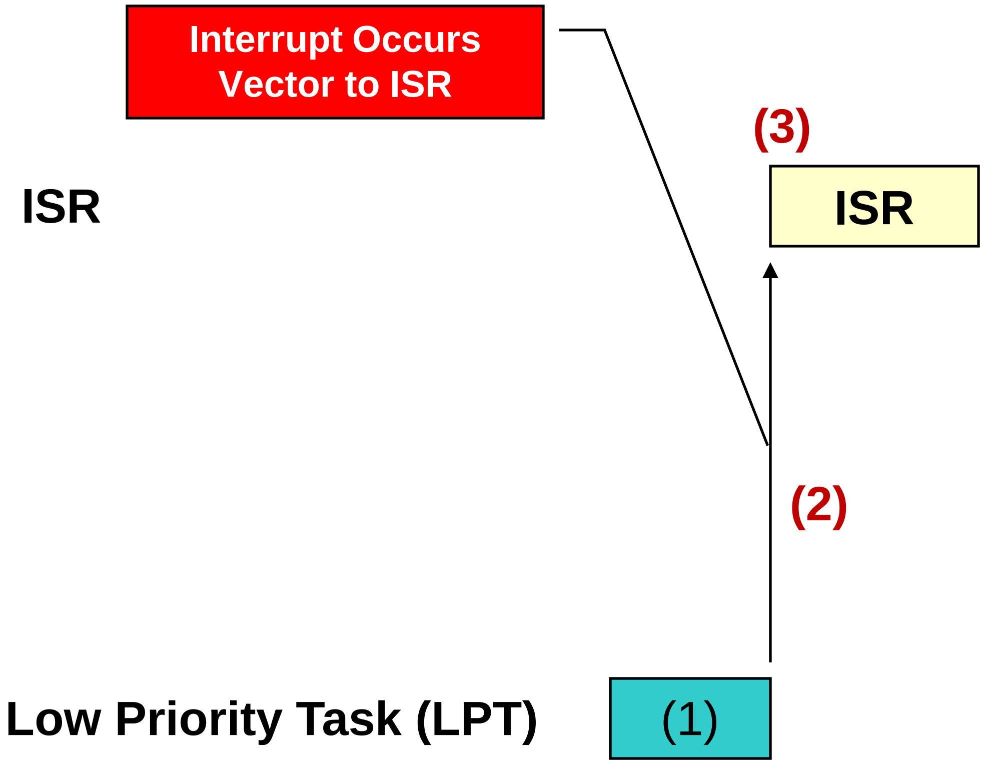
Real-Time Kernels

Preemptive



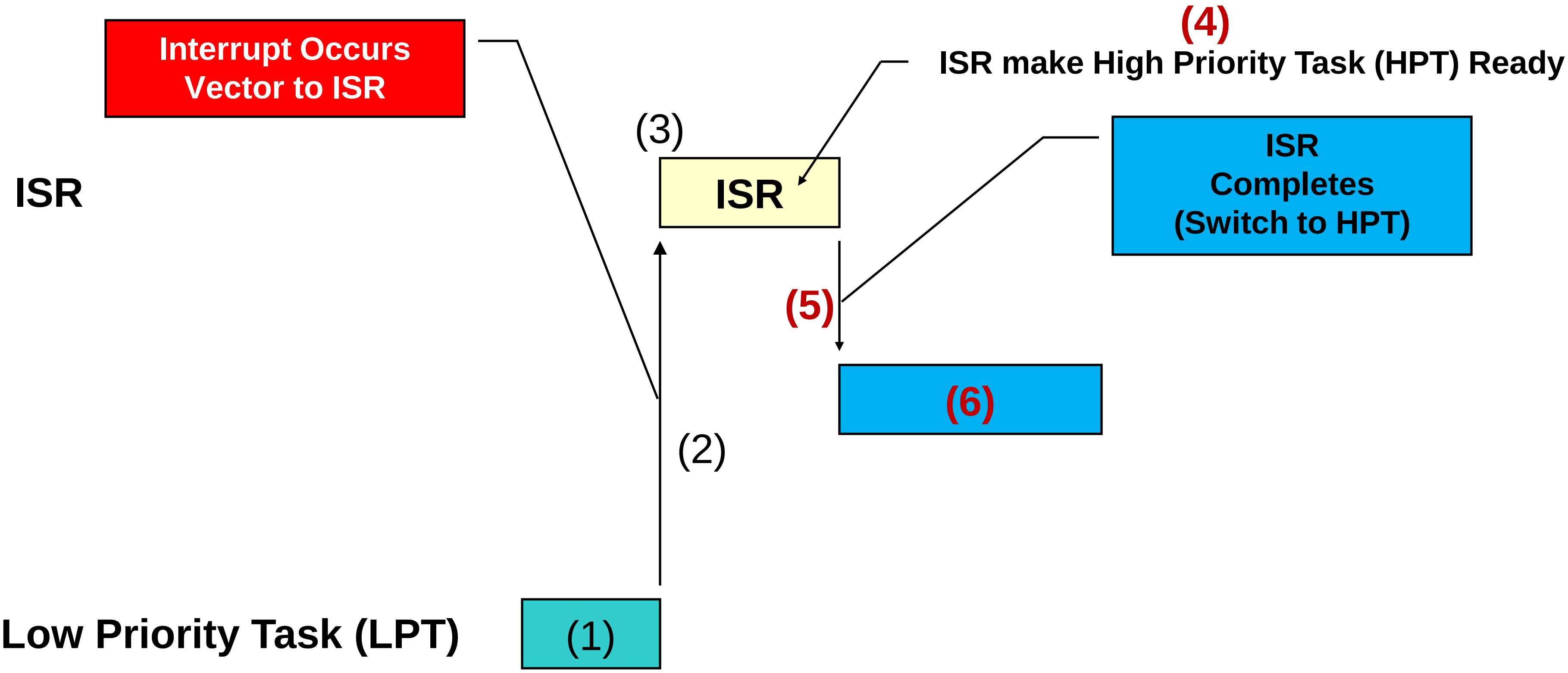
Real-Time Kernels

Preemptive



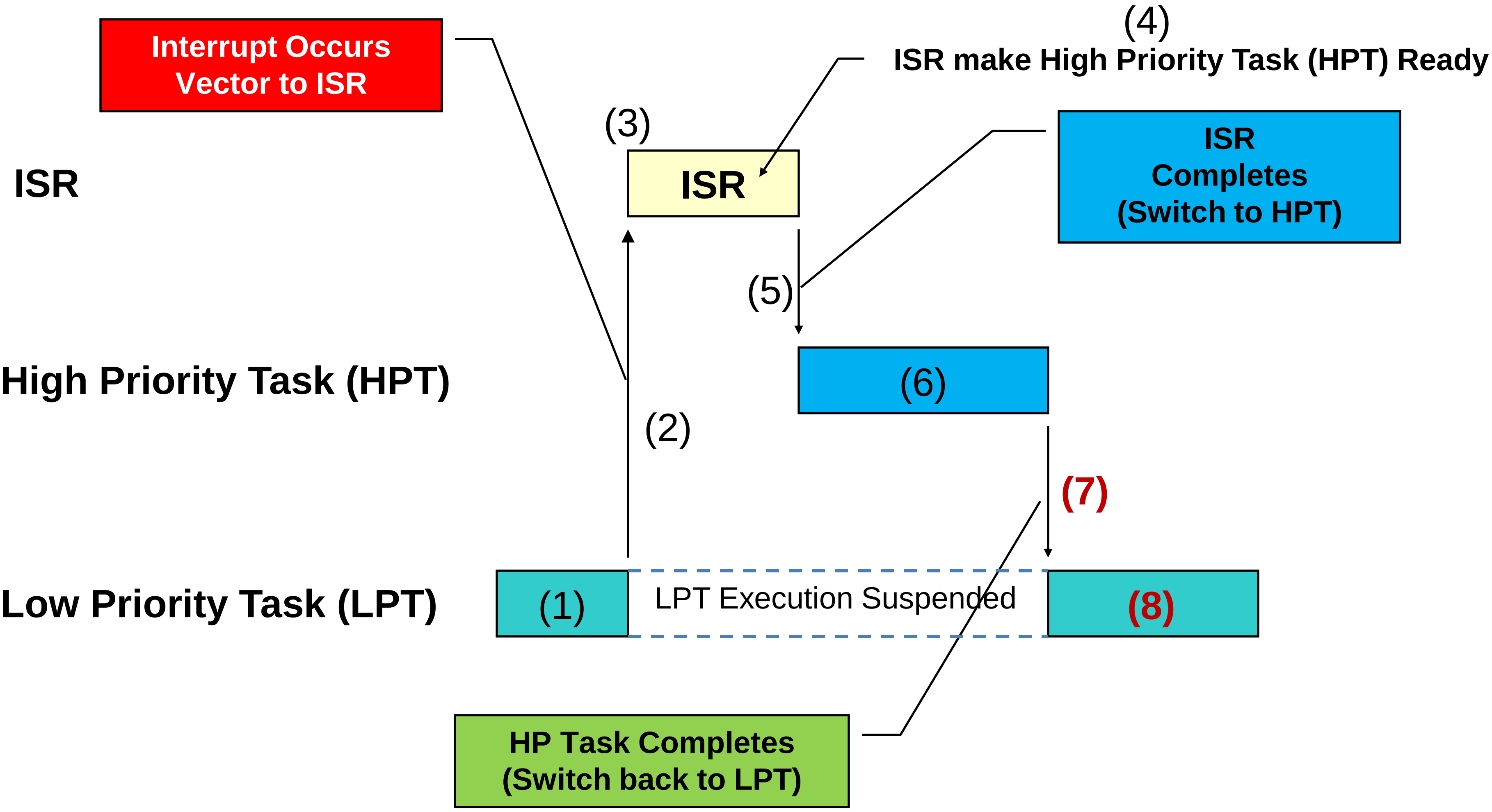
Real-Time Kernels

Preemptive



Real-Time Kernels

Preemptive



Real-Time Kernel
Benefits

- **A kernel:**
  - Enables Multitasking:
    - Breaks (i.e. split) the application into simpler code
    - Allows for easier system expansion
    - Simplifies maintenance
    - Allows different programmers to work on different aspects of the product
  - Provides services to your application
  - Allows you to prioritize the work done by the CPU
  - Is responsive to real-time events
    - Often deterministic
  - Provides a ‘framework’ for your application

Real-Time Kernel

Drawbacks

- **A kernel increases your code and RAM size**
  - Typ. 8K-24K bytes of Code, a few hundred bytes of RAM plus RAM for task stacks
- **A kernel add overhead**
  - Typically 2-4% of the CPU's time
- **A kernel possibly adds cost**
  - Commercial kernels typically require licensing
- **A kernel requires reentrant functions**
- **A kernel will disable interrupts for critical sections**
- **You have to be careful with shared resources**
  - I'll cover that in a different session

Next Class

- **I'll provide more details about task management:**
  - Task resources
  - Task states
  - Task stacks
    - Setting the size
  - Creating tasks
  - Deleting tasks
  - Changing the priority of a task at run-time