Real Time Operating Systems RTOS

Dr.R.Sundaramurthy

Department of EIE
Pondicherry Engineering College

sundar@pec.edu

RTOS

RTOS = Real Time + Operating System

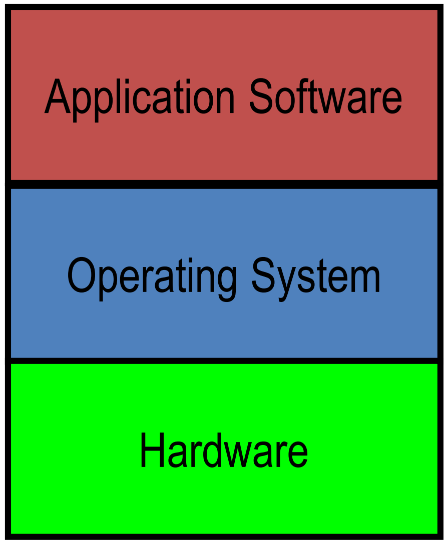
RTOS is a multitasking operating system for Embedded System Applications to meet

1. Time deadlines

2. Functioning with real time constraints

Operating System

- An Operating system (OS) is nothing but a collection of system calls or functions which provides an interface between hardware and application programs.



Real Time

- Real-time systems are those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.

Types of Real Time Systems

- Two types exist
  - Soft real-time
    - Late completion of jobs is undesirable but not fatal.
    - System performance degrades as more & more jobs miss deadlines
    - Example: Online Databases
  - Hard real-time
    - Tasks have to be performed on time
    - Failure to meet deadlines is fatal
    - Example : Flight Control System

RTOS

- RTOS is an operating system that supports real-time applications by providing logically correct result within the deadline required.
- Basic Structure is similar to regular OS but, in addition, it provides mechanisms to allow real time scheduling of tasks.

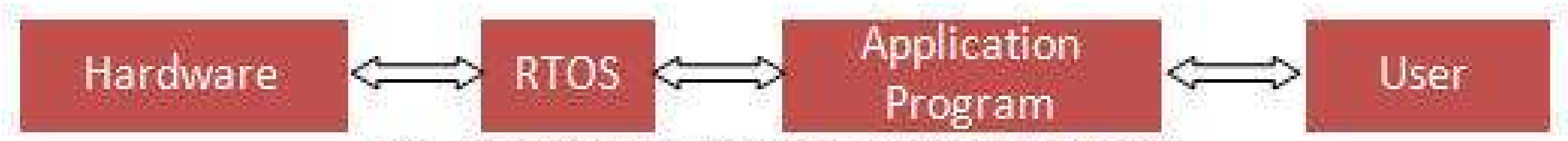


Fig. Real time embedded system with RTOS

Single Tasking Vs MultiTasking

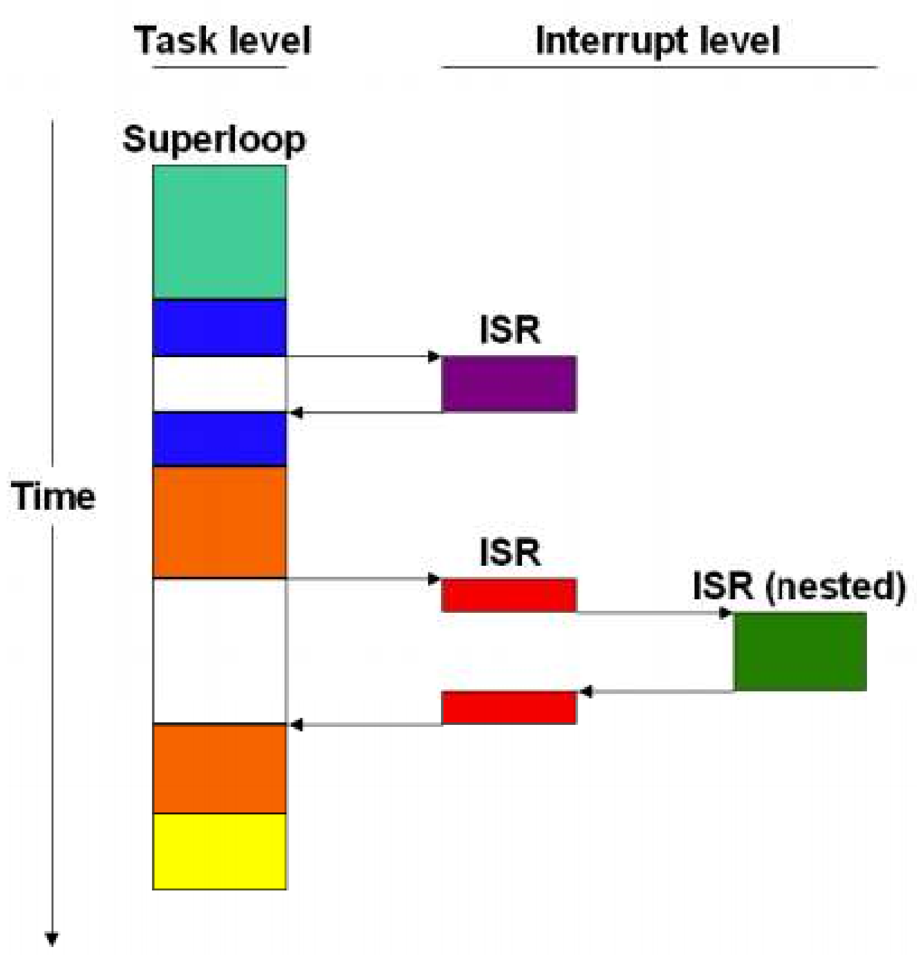
- a task is a program running on the CPU core of a microcontroller.
- Without a multitasking kernel (an RTOS), only one task can be executed by the CPU at a time. This is called a single-task system.
- A real-time operating system allows the execution of multiple tasks on a single CPU. All tasks execute as if they completely own the entire CPU.
- The tasks are scheduled for execution, meaning that the RTOS

  can activate and deactivate each task according to its priority.
- Always the highest priority task being executed in general.

Single-task systems (superloop)

```
while(1)
{
task1();
task2();
task3();
task4();
}
```

```
void myISR() __irq
{
interrupttask();
}
```



Single Task Systems

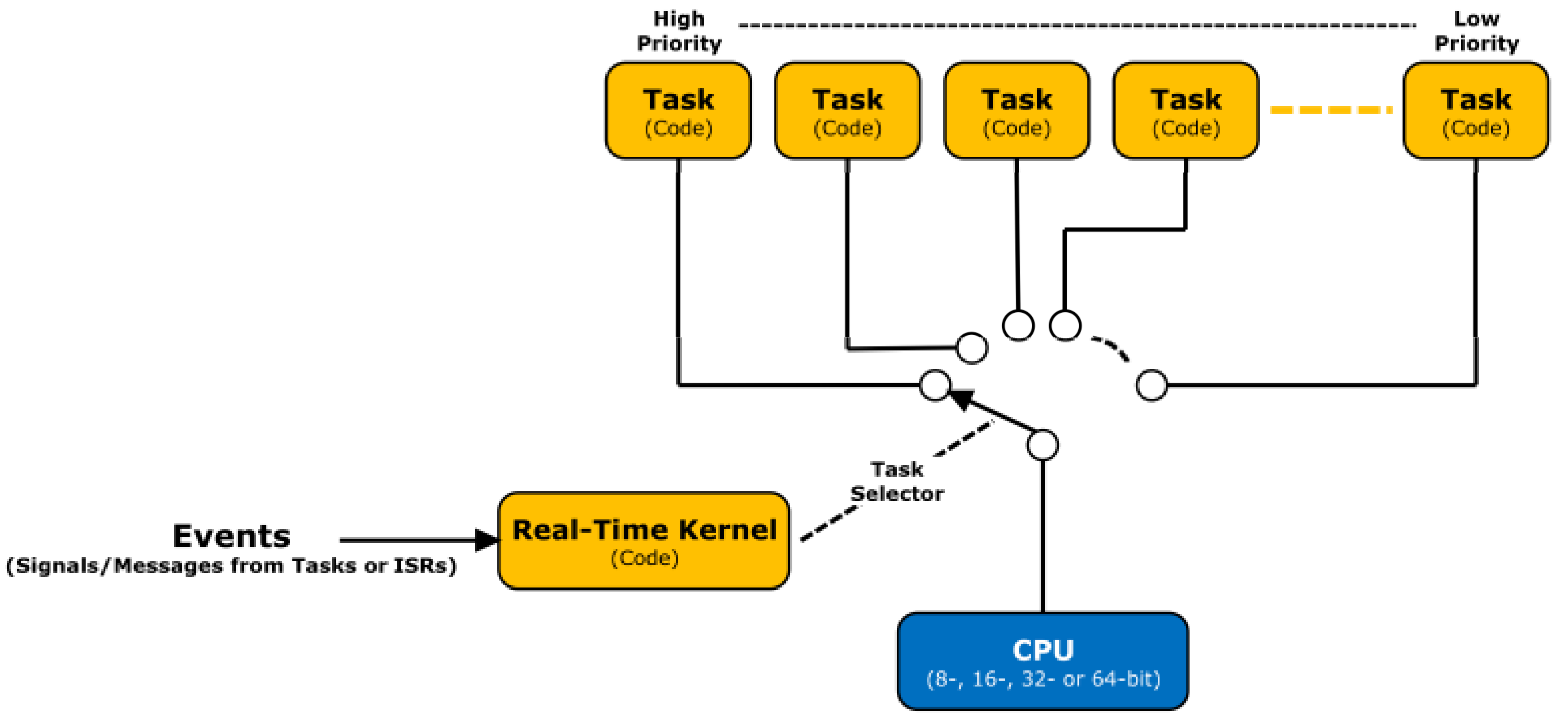
- "superloop design" is the classic way of designing embedded systems does not use the services of an RTOS.
- no real time kernel is used, so interrupt service routines (ISRs) are used for the real-time parts of the application(critical operations).
- This type of system is typically used in small, simple systems or if real-time behavior is not critical.

Nature of Single Tasking Systems

- Typically, because no real-time kernel and only one stack is used
- Both ROM and RAM size for simple applications are smaller when compared to using an RTOS.
- There are no inter-task synchronization problems with a superloop application.
- However, superloops can become difficult to maintain if the program becomes too large or uses complex interactions.
- As sequential processes cannot interrupt themselves, reaction times depend on the execution time of the entire sequence, resulting in a poor real-time behavior.

Multitasking Systems

In a multitasking system, the CPU time is distributed amongst different tasks.



Scheduling in RTOS

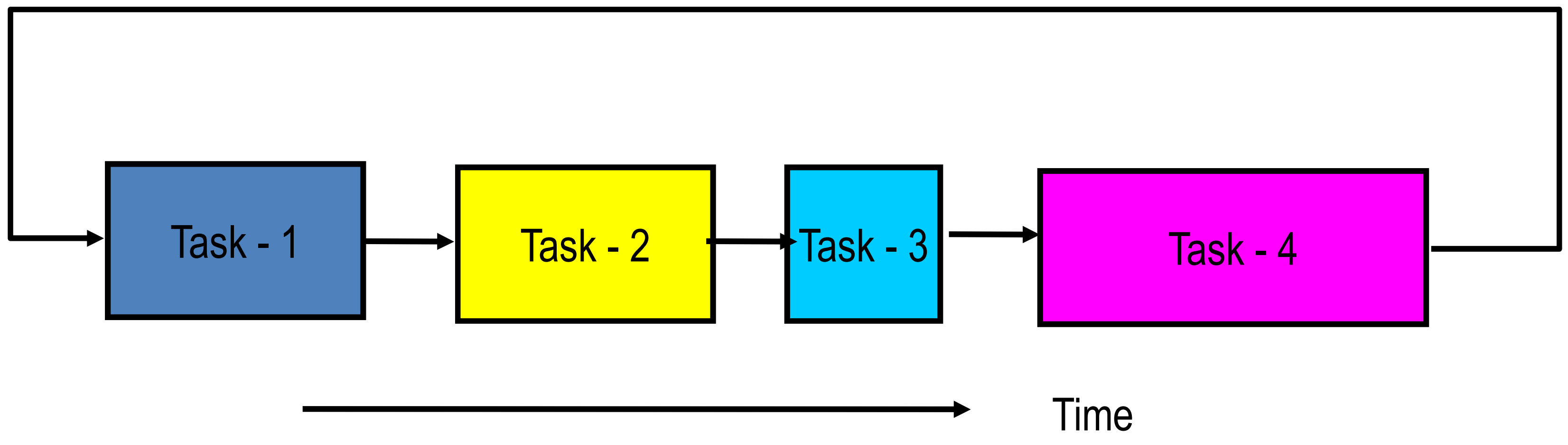
- scheduling is the process of distributing the CPU time amongst different tasks.
- To schedule the tasks, More information about the tasks should be known
  - Number of tasks
  - Resource Requirements
  - Execution time
  - Deadlines

Scheduling Algorithms in RTOS

- Non-Priority based Scheduling
  - Cyclic executive
  - Shortest Task First
  - Round Robin
- Priority Scheduling
  - Cooperative Scheduling
  - Preemtive Scheduling

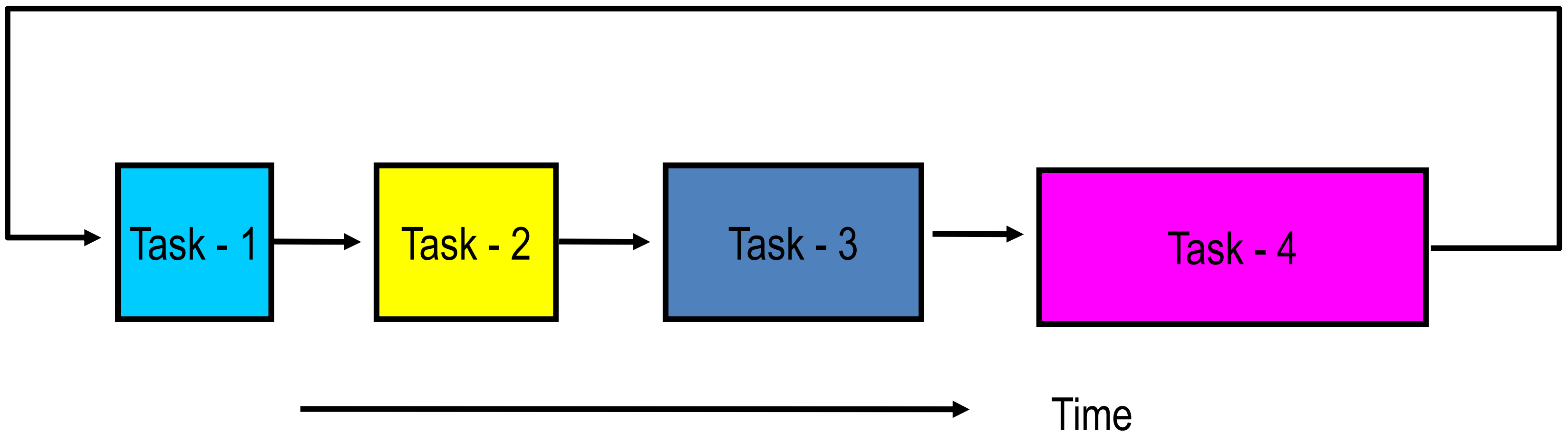
Cyclic executive

- Tasks are arranged in Cyclic Fashion.
- Each Task has to wait until the completion of previous task.



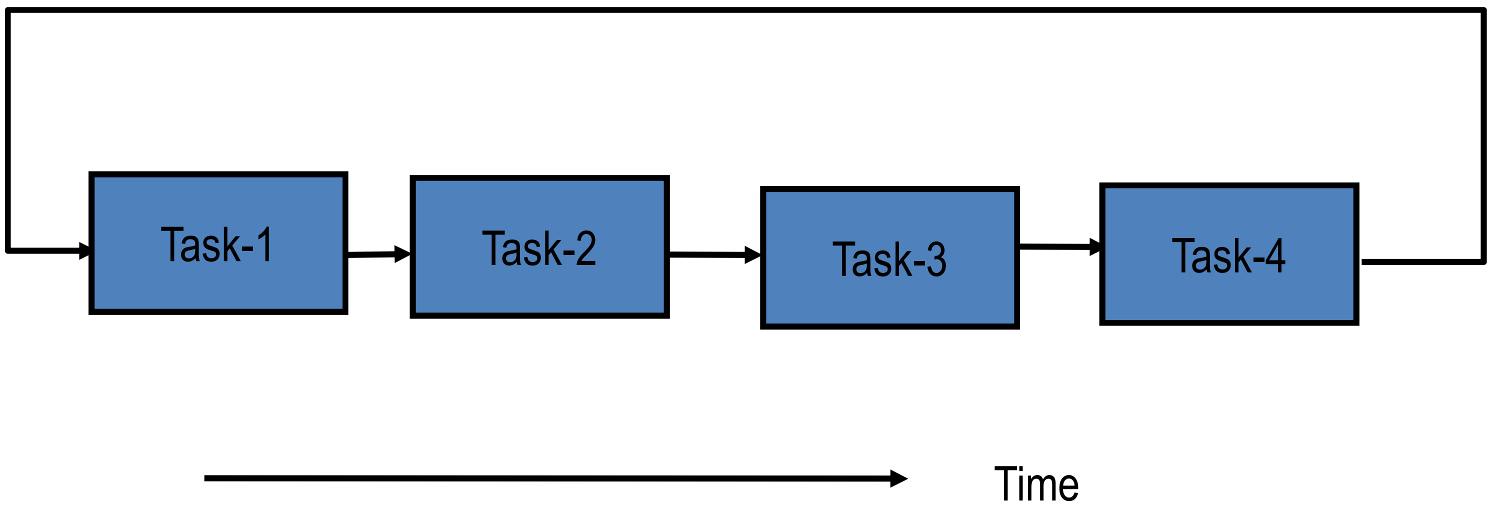
Shortest Task First

- Tasks with shortest Execution time are executed first.
- It requires precise knowledge of how long a task will run



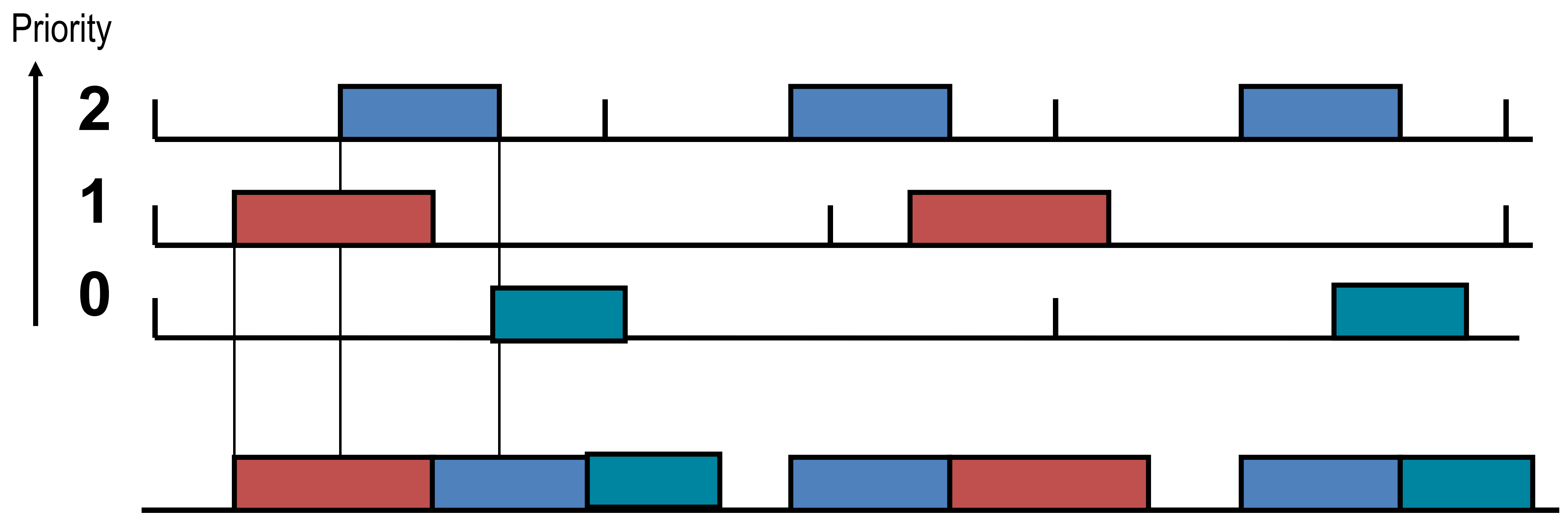
Round Robin Scheduling

- Equal Time share of the CPU is given for all the Tasks.
- CPU is given to the next task if
  - The current task completes before time slice
  - The current task uses the full time slice

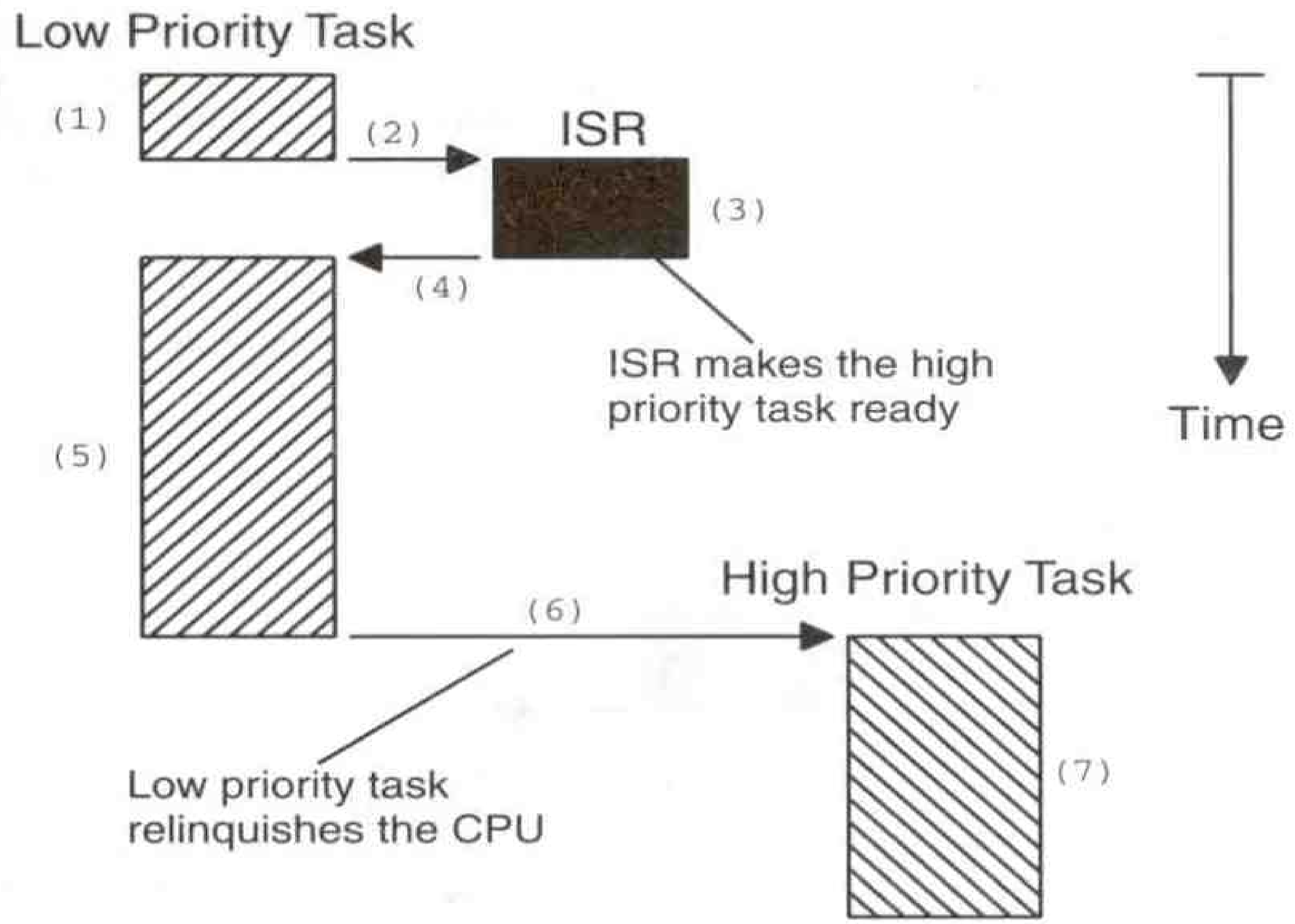


Coperative Scheduling

- Tasks are Prioritized.
- Always High Priority Executes First.
- In the event of a High Priority Task interrupting a Low Priority Task (Which is currently running), the Low Priority Task First Completes itself before giving the control to High Priority Task.

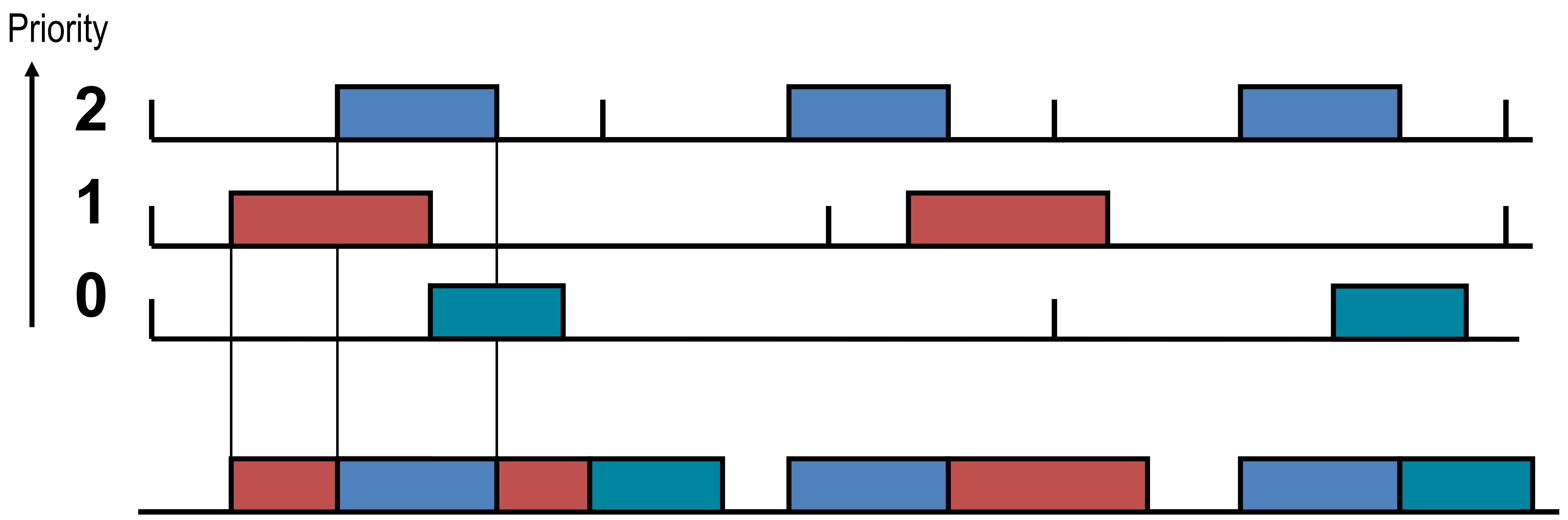


Coperative Scheduling

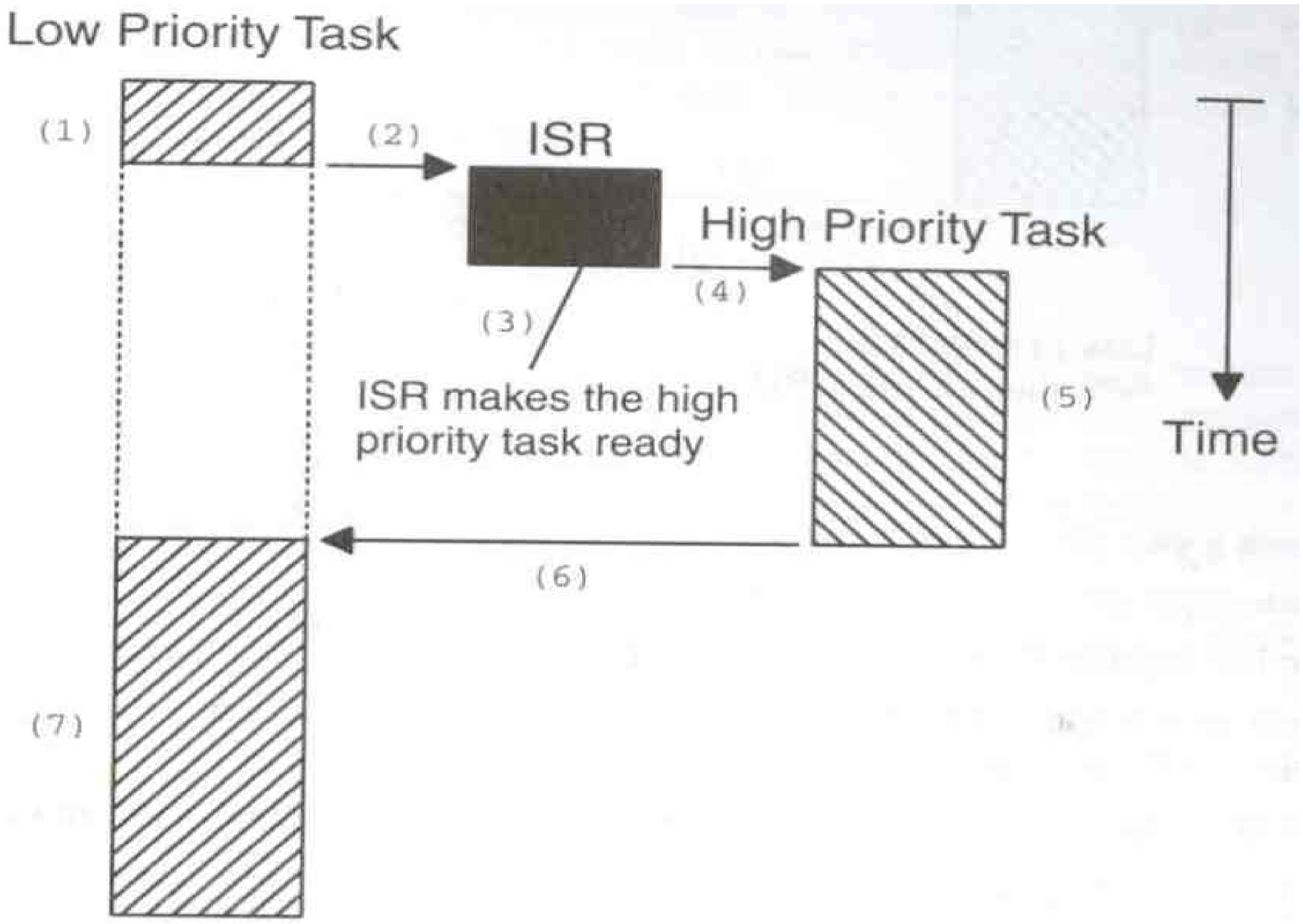

Low Priority Task
(1)
(2)
ISR
(3)
(4)
ISR makes the high priority task ready
Time
(5)
High Priority Task
(6)
Low priority task relinquishes the CPU
(7)

Preemtive Scheduling

- Tasks are Prioritized.
- Always High Priority Executes First.
- In the event of a High Priority Task interrupting a Low Priority Task (Which is currently running), the High Priority Task **<u>Preemts</u>** the Low Priority Task and takes the control.



Preemtive Scheduling



List of RTOS

Open Source RTOS

- Linux
- eCos
- uClinux
- **FreeRTOS**
- RTAI
- coscox
- Rocket OS

Properitary RTOS

- QNX
- VxWorks
- INTEGRITY
- ThreadX
- MicroC/OS2
- embOS
- SafeRTOS

End of Session

sundar@pec.edu