Introduction to Real-Time Operating Systems

A Training for Software Development Professionals

Training Goals

- Understanding Real-Time Concepts
- Gaining the Knowledge of Real-Time Operating Systems and Internals

Contacts/Resources

- Özgür Aytekin
  - Embedded Software Engineer at Collins Aerospace
  - ozguraytekin@gmail.com

Chapter 1

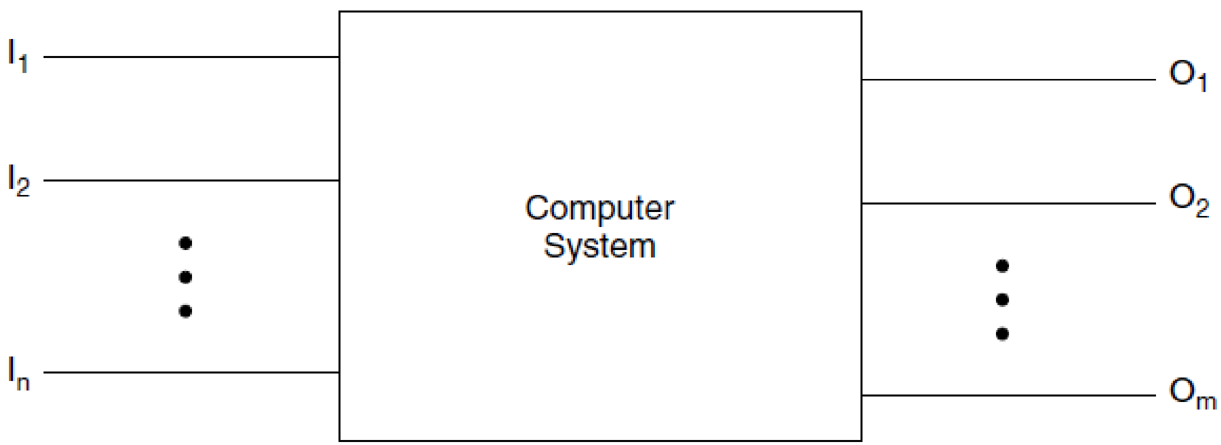
Understanding the Basic Concepts

Agenda

- ➢ **What is a system?**
- ❑ Fundamentals of Real-Time Systems

Definition - 1

"A system is a mapping of a set of inputs into a set of outputs."



Definition - 2

- A set of integrated components that interact with each other and depend upon each other, to achieve a complex function together.

- A system can be decomposed into smaller subsystems or components

- A system may be one of the components for a larger system.

Agenda

- ❑ What is a system?
- ➢ **Fundamentals of Real-Time Systems**

Definition

A real-time system is a computer system that must satisfy bounded response time constraints or risk severe consequences, including failure.

Definition (cont)

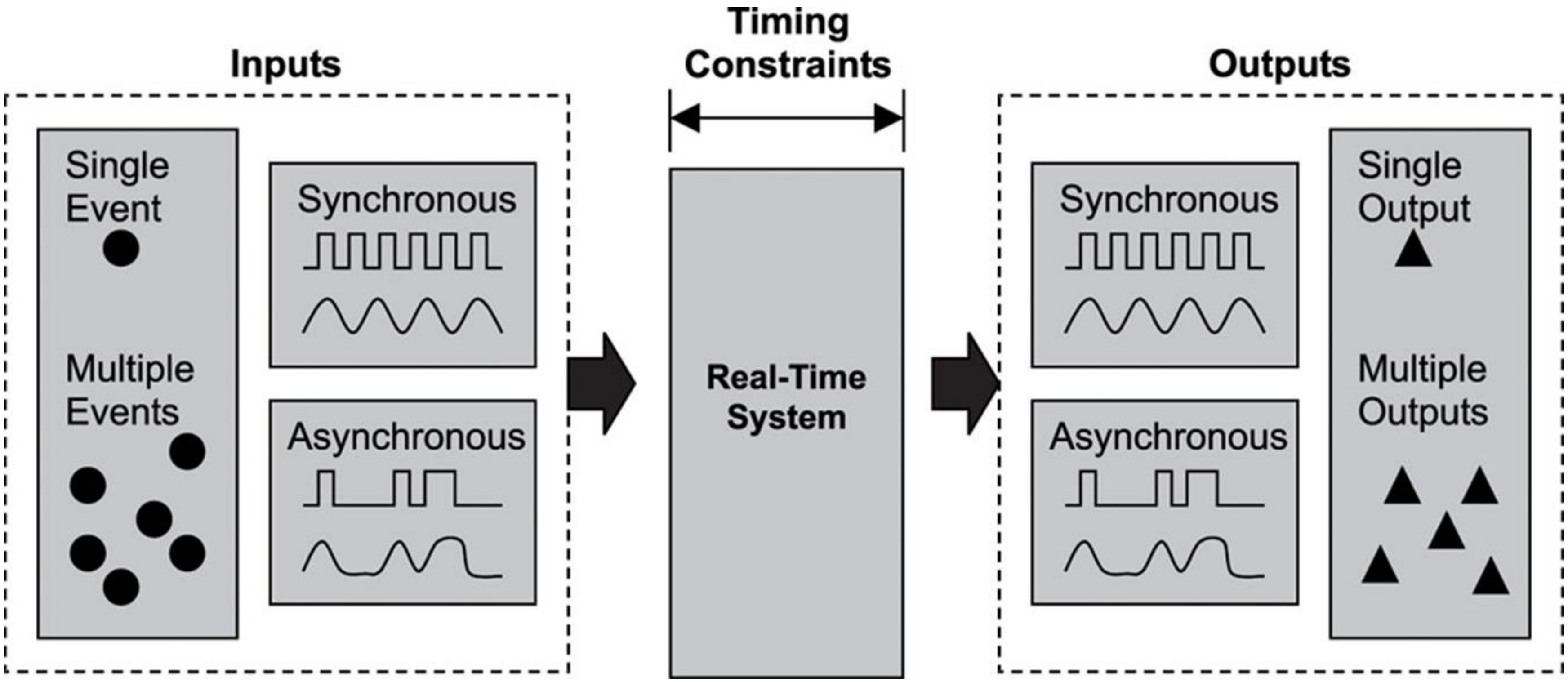
- Real-time systems are defined as those systems in which the overall correctness of the system depends on both the functional correctness and the timing correctness.

- The timing correctness is at least as important as the functional correctness.

Response Time of a System

The time between the presentation of a set of inputs to a system (stimulus) and the realization of the required behavior (response), including the availability of all associated outputs, is called the response time of the system.

A Simple view of real-time systems



Failed System

A failed system is a system that cannot satisfy one or more of the requirements stipulated in the system requirements specification

Terminology

- Hard Real-Time
  - Timing requirements must be met precisely
  - Failure to meet requirements leads to significant failure
  - The penalty incurred for a missed deadline is catastrophe.
- Soft Real-Time
  - Missing deadlines does not cause catastrophic effects
  - Costs can rise in proportion to the delay, depending on the application
  - There is some flexibility in the timing constraints

Hard or Soft?

- Are these applications Hard or Soft real-time?

Flight Management System	Hard
Bottling Plant Production Line	Hard
Automatic Train Protection	Hard
Anti-Lock Braking System	Soft
Airline Reservation System	Soft
Cellular Phone RF Reception	Hard
Vending Machine	Soft
Telephony Switch	Hard
Missile Guidance System	Hard
Smart TV	Soft

Constraints

- Where does the constraints come from?
- Who determines the constraints?

Constraints as Requirements

- Timing constraints should be defined as requirements/specification
- No timing requirement no real-time system
- System/Software Requirements should include real-time requirements

Real vs. Artificial Constraints

- Real
  - There is no flexibility in the timing constraint. If it is changed or relaxed, the system will fail.
  - Eg. 150 msec window for filling bottle on conveyor belt.
- Artificial
  - Timing constraint was determined somewhat arbitrarily.
  - E.g. 40 Hz rate for cruise control.

<u>Important to distinguish between Real and Artificial</u>:

- Artificial constraints can be refined, possibly improving other design factors (like cost & functionality)
- Real constraints must be met at any cost.

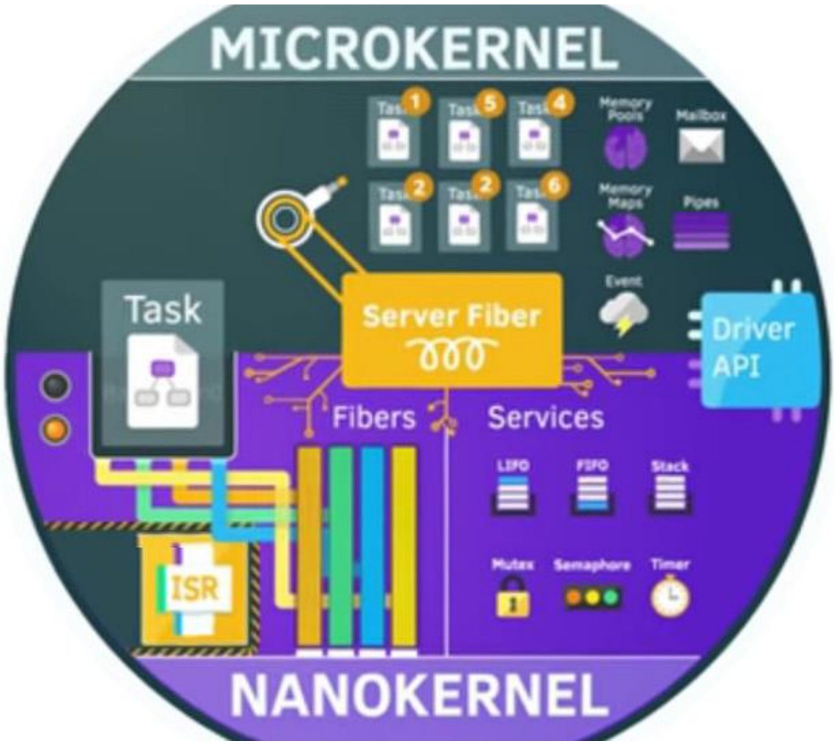
Common Misconceptions

- Real-time computing is equal to fast computing
- There is no science in real-time system design
- Length of the deadline makes a real-time system hard or soft
- Advances in computer hardware will take care of real-time requirements
- Real-time system research is equal to performance search
- Real-time systems are fail-safe systems

Chapter 2

Introduction to Operating Systems

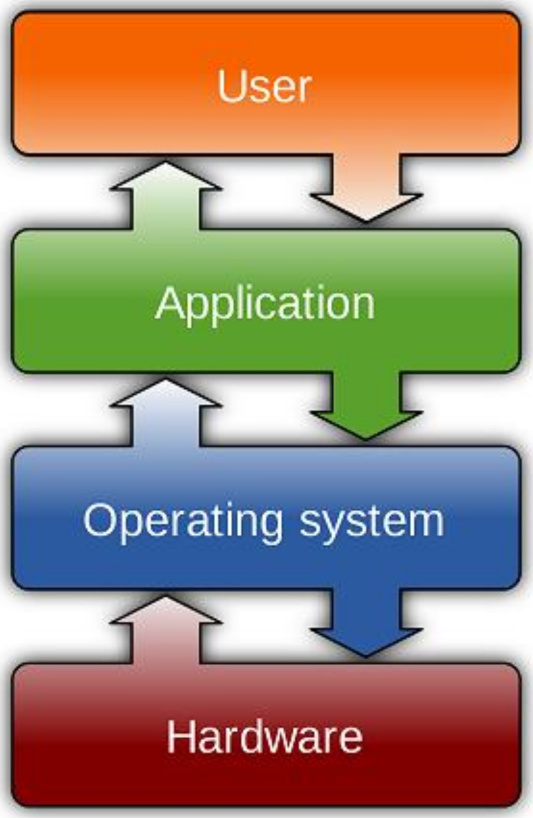
Agenda



- ➢ **What is an Operating System?**
- ❑ Operating Systems Concepts
- ❑ Operating Systems Types
- ❑ Defining an RTOS

What is an Operating System?

- An OS is a program that controls the execution of application programs, and acts as an interface between applications and the computer hardware



What is an Operating System? (cont)

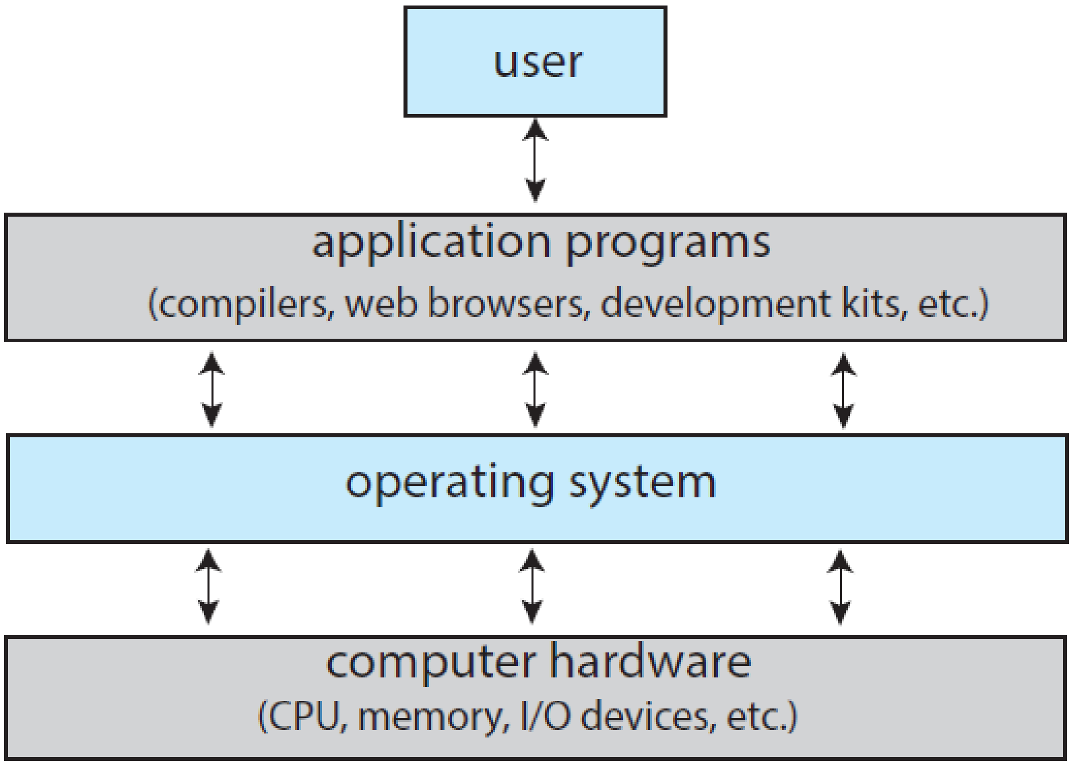
- The OS can be defined in the following ways:
  - A software that acts as an interface between the users and hardware of the computer system
  - A software that provides a working environment for the users' applications
  - A resource manager that manages the resources needed for all the applications in the background

Two viewpoints

- To understand more fully the operating system's role, operating systems may be explored from two viewpoints:
  - User view
  - System view

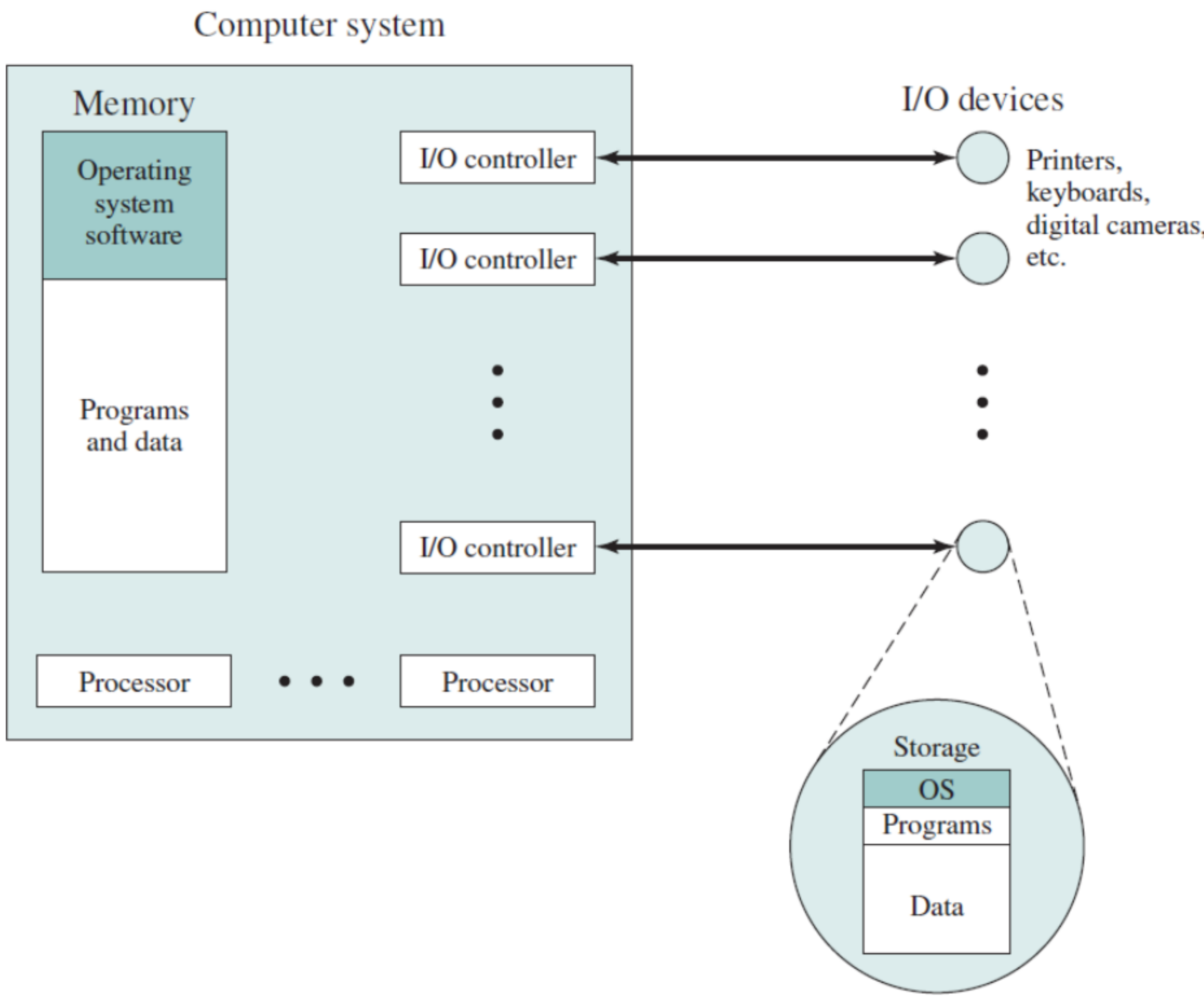
User View

- Operating Systems provide abstractions to user programs



System View

- The operating system is a resource allocator



System View (cont)

- The OS is responsible for controlling the use of a computer's resources such as
  - I/O
  - Memory Space
  - System's CPU
  - File Storage Space
  - Disk Storage

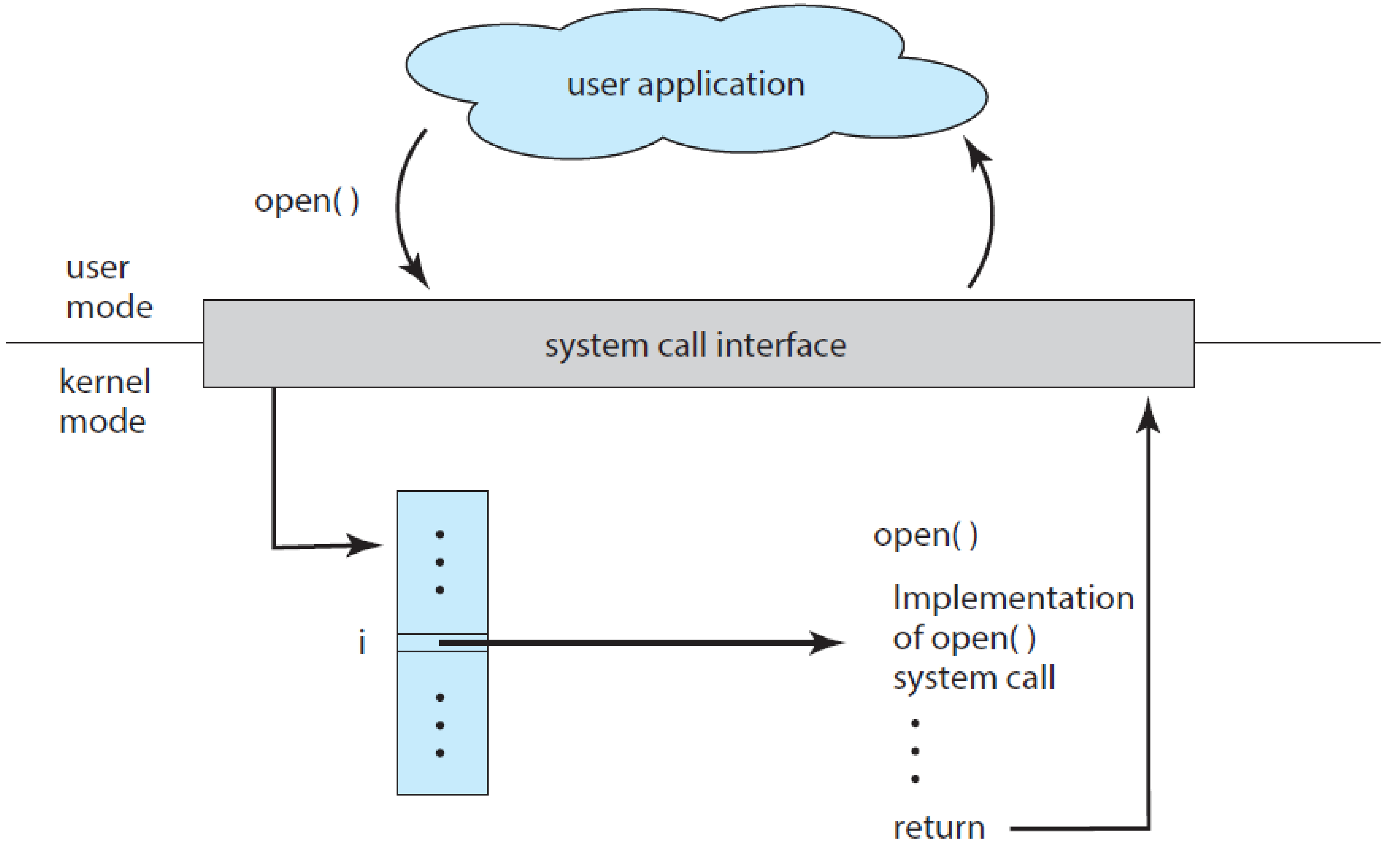
System Call

- Operating systems have two main functions:
  - Providing abstractions to user programs
  - Managing the computer's resources

- The resource management functionality is largely transparent to the users and done automatically.

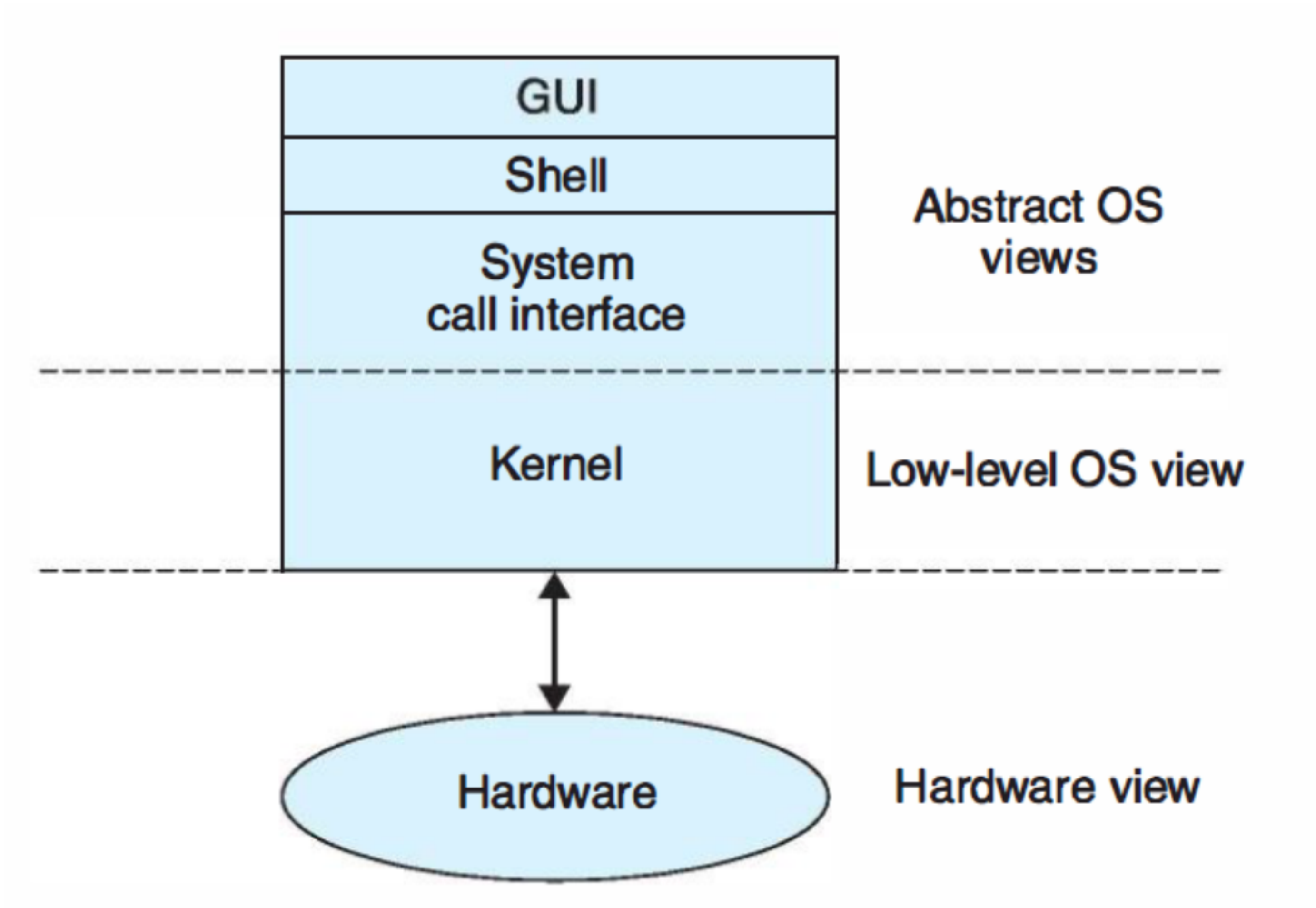
- System calls provide an abstraction interface to the services made available by an operating system.

System Call (cont)



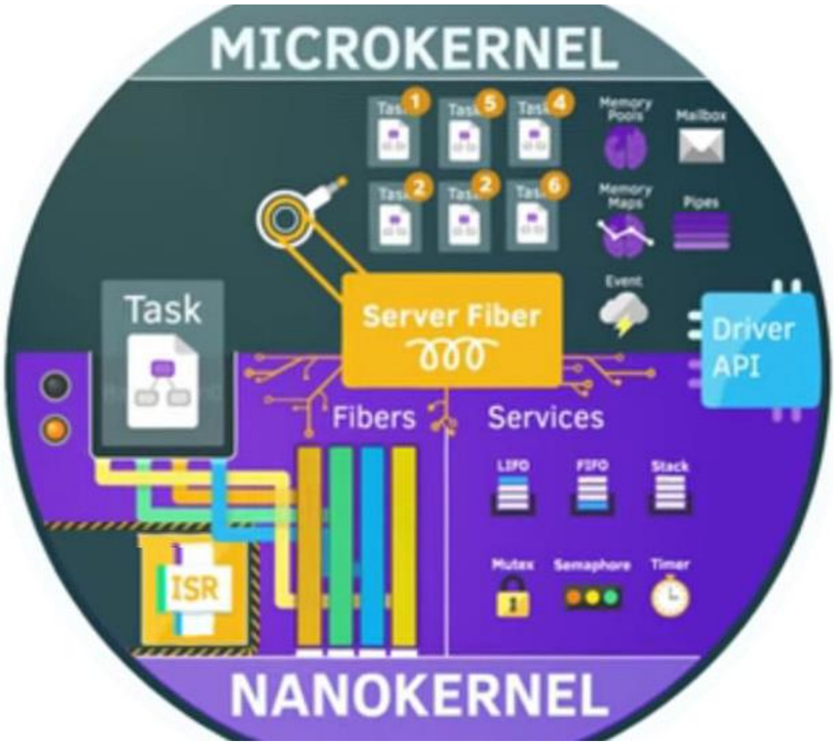
All Content Copyright Edge Software

System Call (cont)



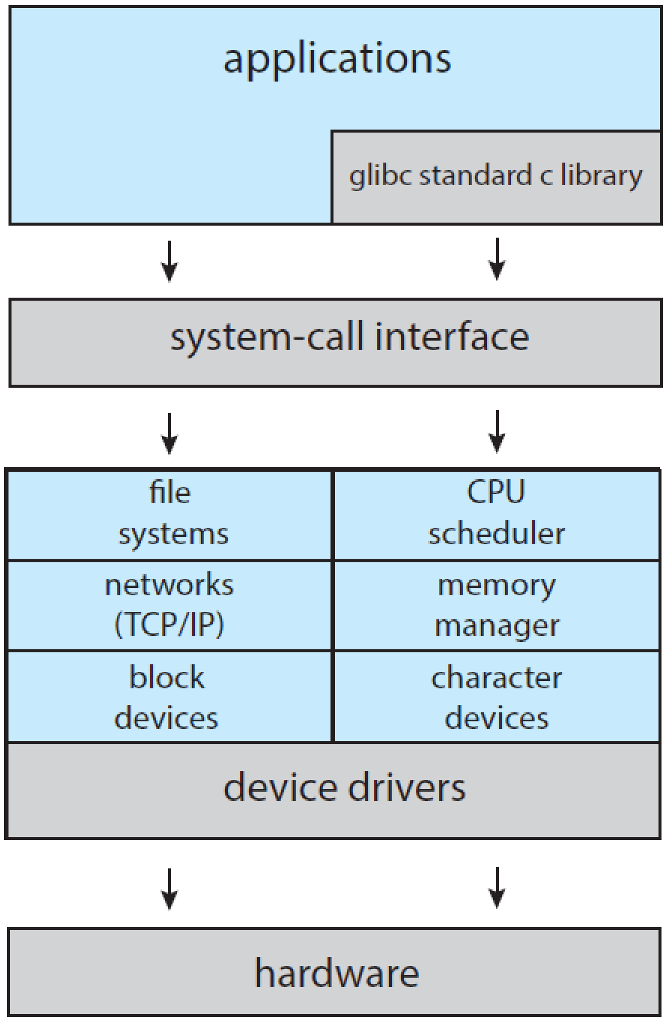
All Content Copyright Edge Software

Agenda



- What is an Operating System?
- **Operating Systems Types**
- Why Study Operating Systems?
- Defining an RTOS

Monolithic Kernel

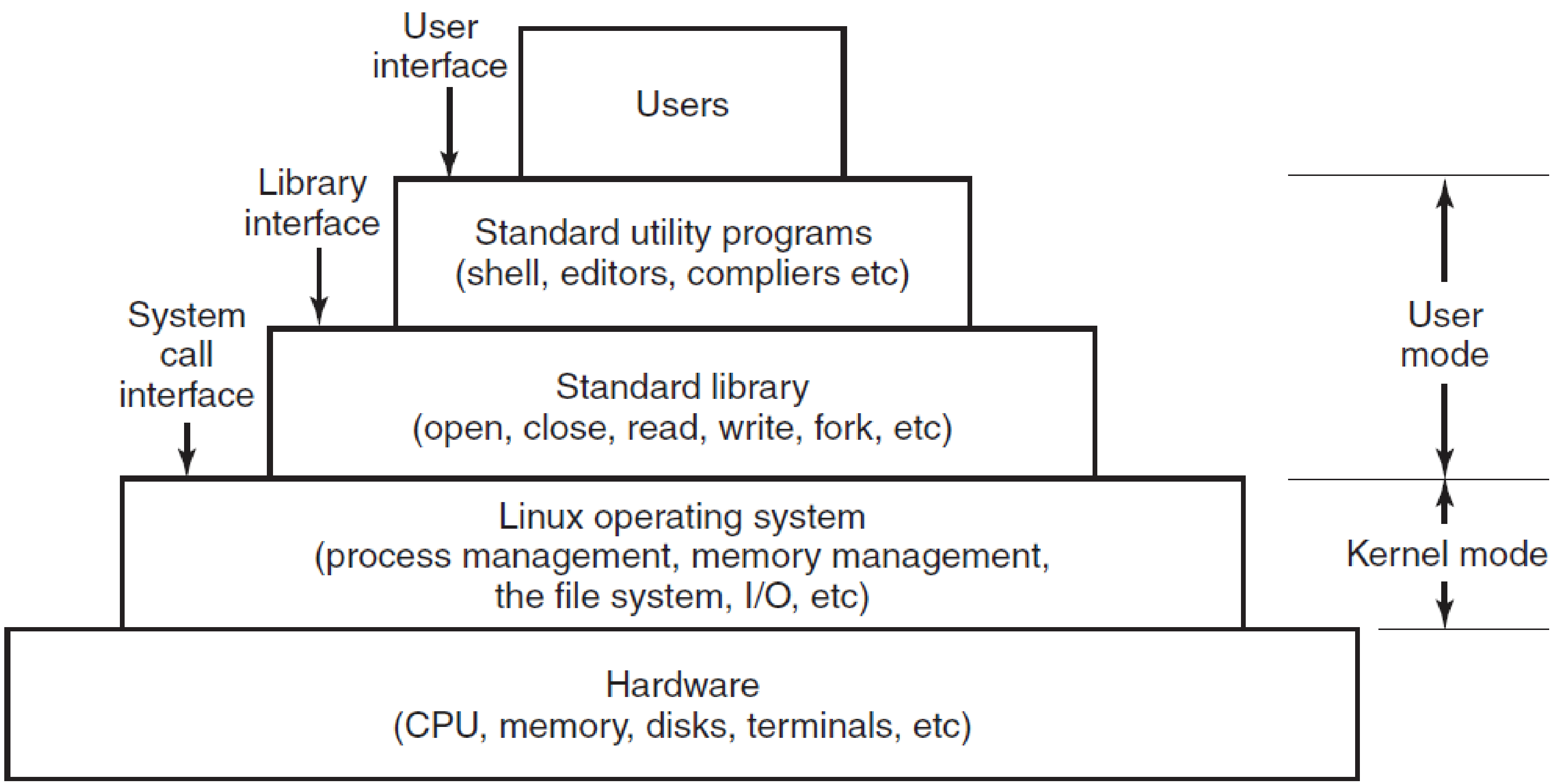


- In the monolithic approach the entire operating system runs as a single program in kernel mode

- The operating system is written as a collection of procedures, linked together into a single large executable binary program.

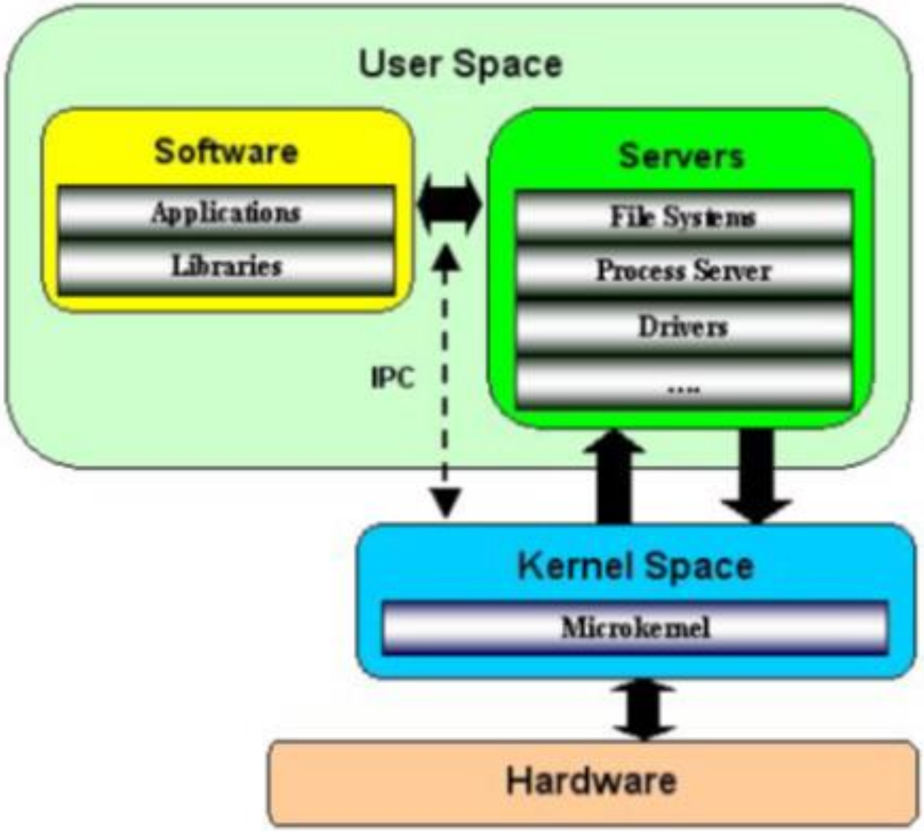
All Content Copyright Edge Software

Linux


User interface
Users
Library interface
Standard utility programs
(shell, editors, compliers etc)
System call interface
Standard library
(open, close, read, write, fork, etc)
Linux operating system
(process management, memory management,
the file system, I/O, etc)
Hardware
(CPU, memory, disks, terminals, etc)
User mode
Kernel mode

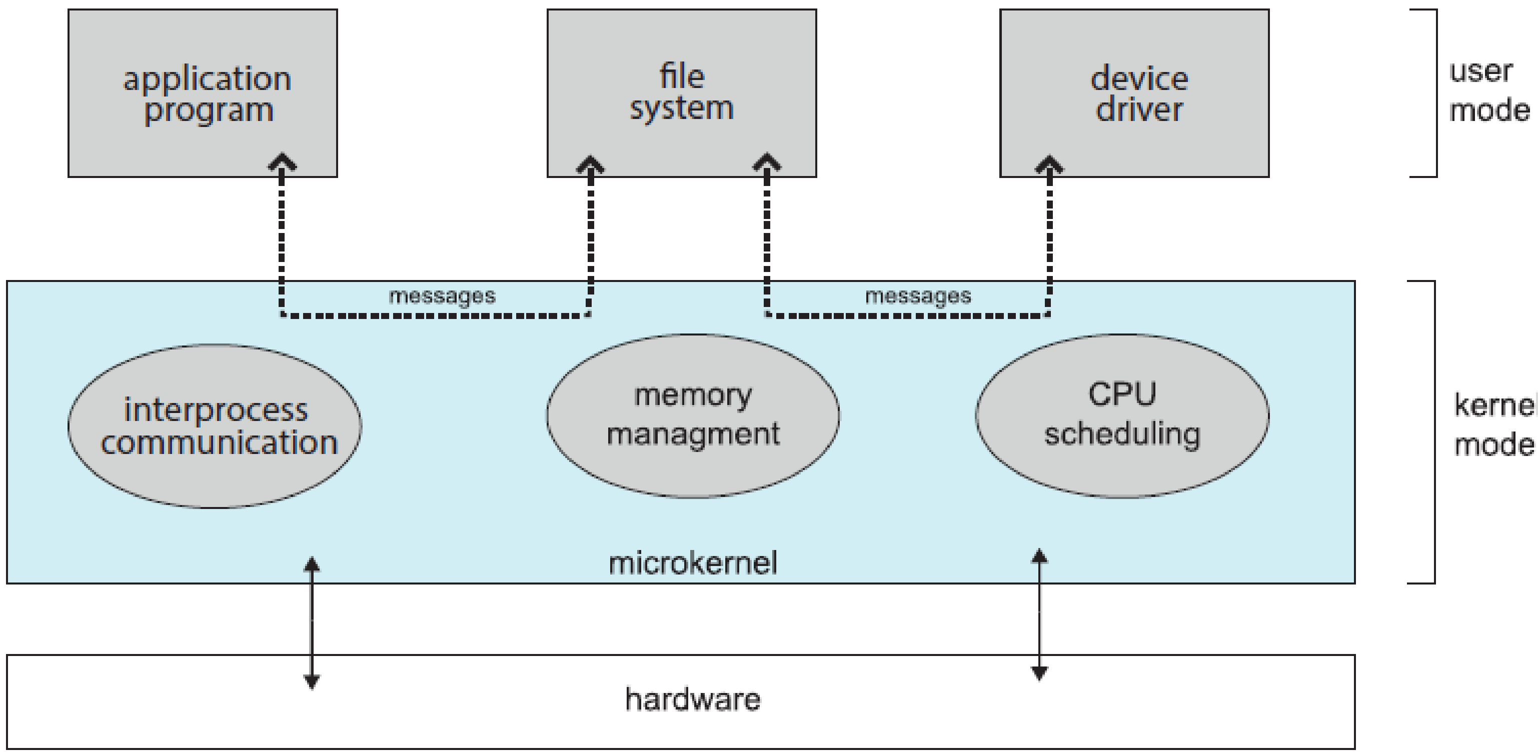
Microkernel

- It runs only basic process communication and I/O control.
- The other system services) reside in user space in the form of servers.



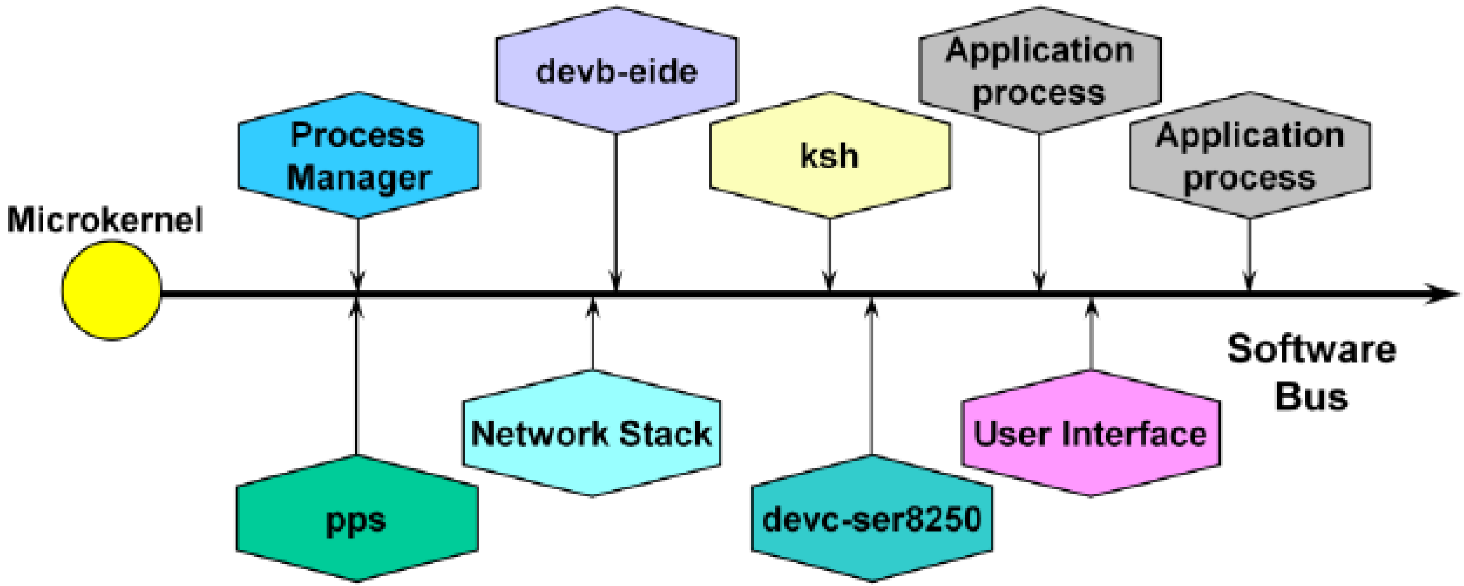
All Content Copyright Edge Software

Architecture of a typical microkernel



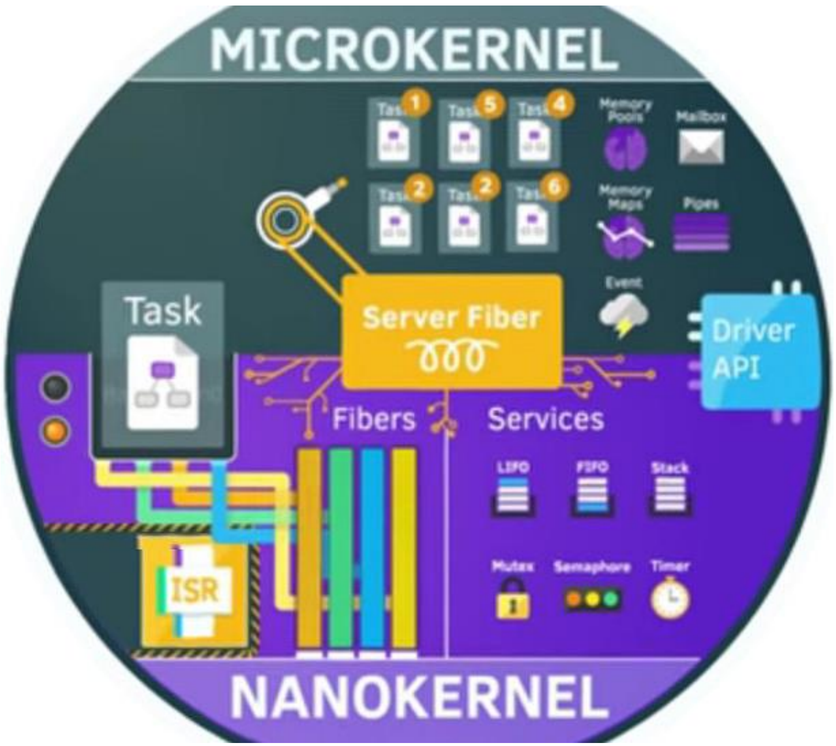
All Content Copyright Edge Software

QNX Microkernel



All Content Copyright Edge Software

Agenda



- ❑ What is an Operating System?
- ❑ Operating Systems Types
- ➢ **Why Study Operating Systems?**
- ❑ Defining an RTOS

All Content Copyright Edge Software

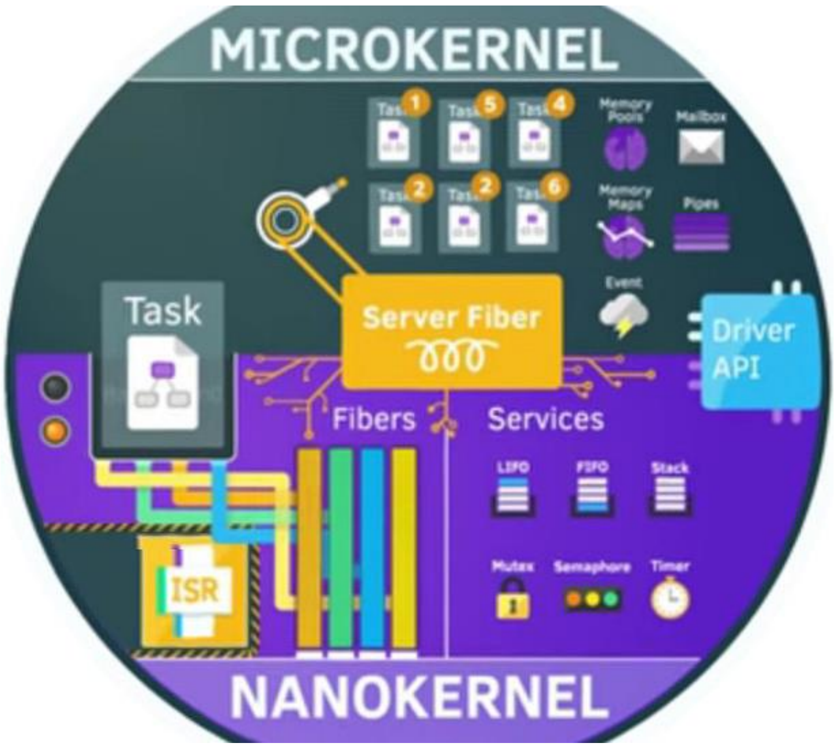
Why Study Operating Systems?

- Almost all code runs on top of an operating system

- Knowledge of how operating systems work is crucial to proper, efficient, effective, and secure programming

- Understanding the fundamentals of operating systems is essential to those who write programs on them and use them.

Agenda



- ❑ What is an Operating System?
- ❑ Operating Systems Types
- ❑ Why Study Operating Systems?
- ➢ **Defining an RTOS**

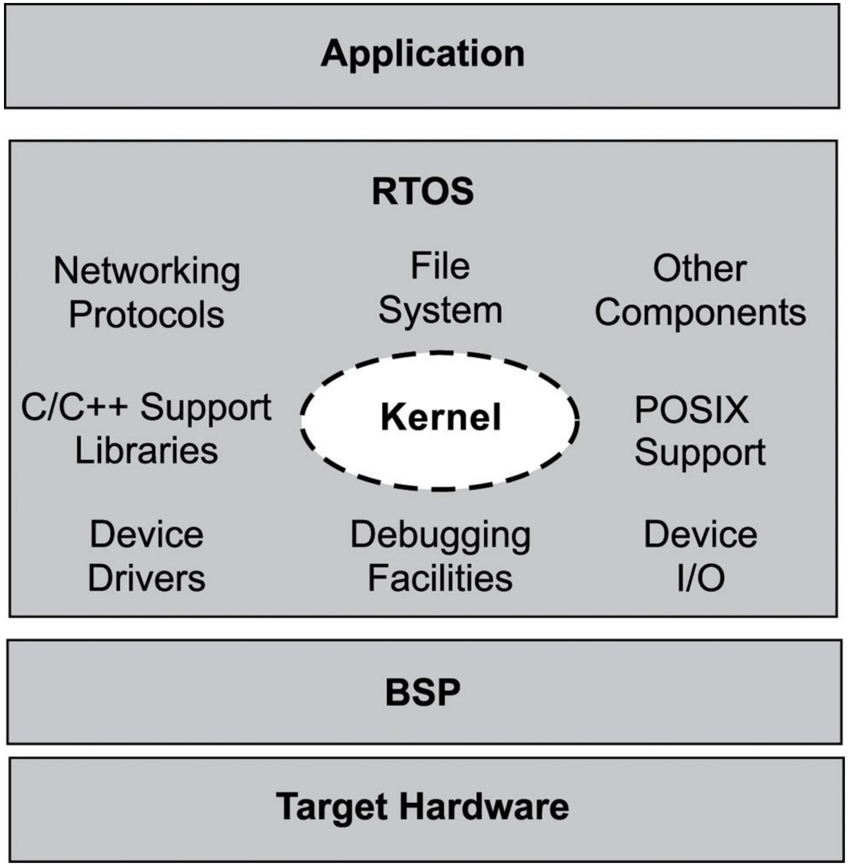
What is an RTOS?

- A real-time operating system (RTOS) is a program that
  - Schedules execution in a timely manner
  - Manages system resources
  - Provides a consistent foundation for developing application code.

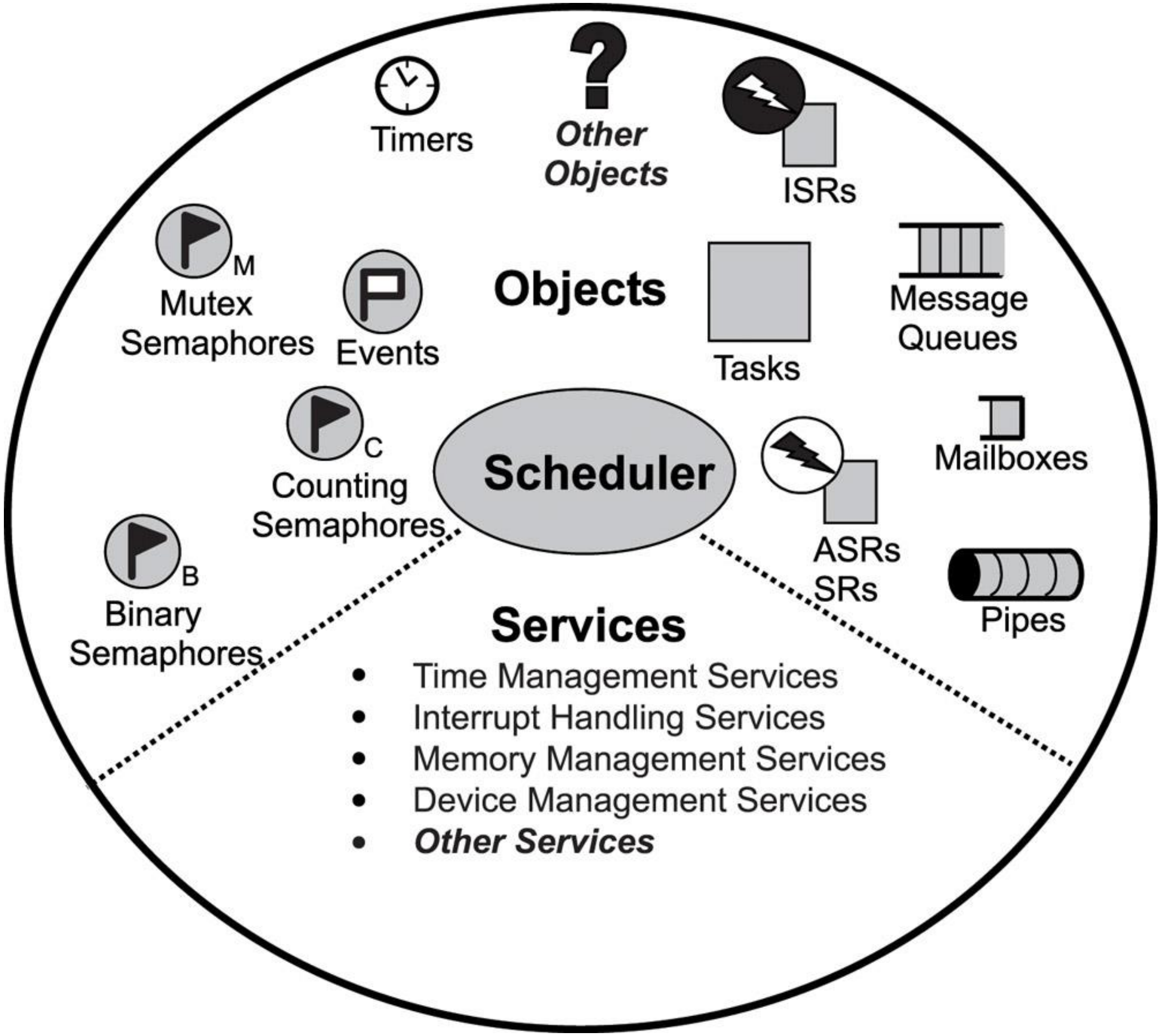
- An RTOS is a kind of operating system with special features such as
  - Priority based pre-emptive scheduling
  - Constant and very low scheduling latency
  - Very low and measurable interrupt latency

High-level view of an RTOS

- RTOS can be a combination of various modules, including
  - the kernel,
  - a file system,
  - networking protocol stacks,
  - Components required for a particular application



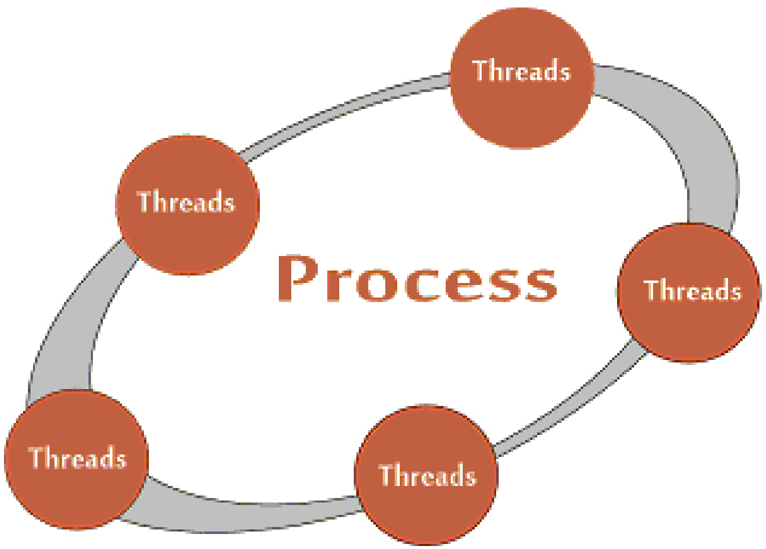
Common components in an RTOS



Chapter 2

Process, Threads and Scheduling

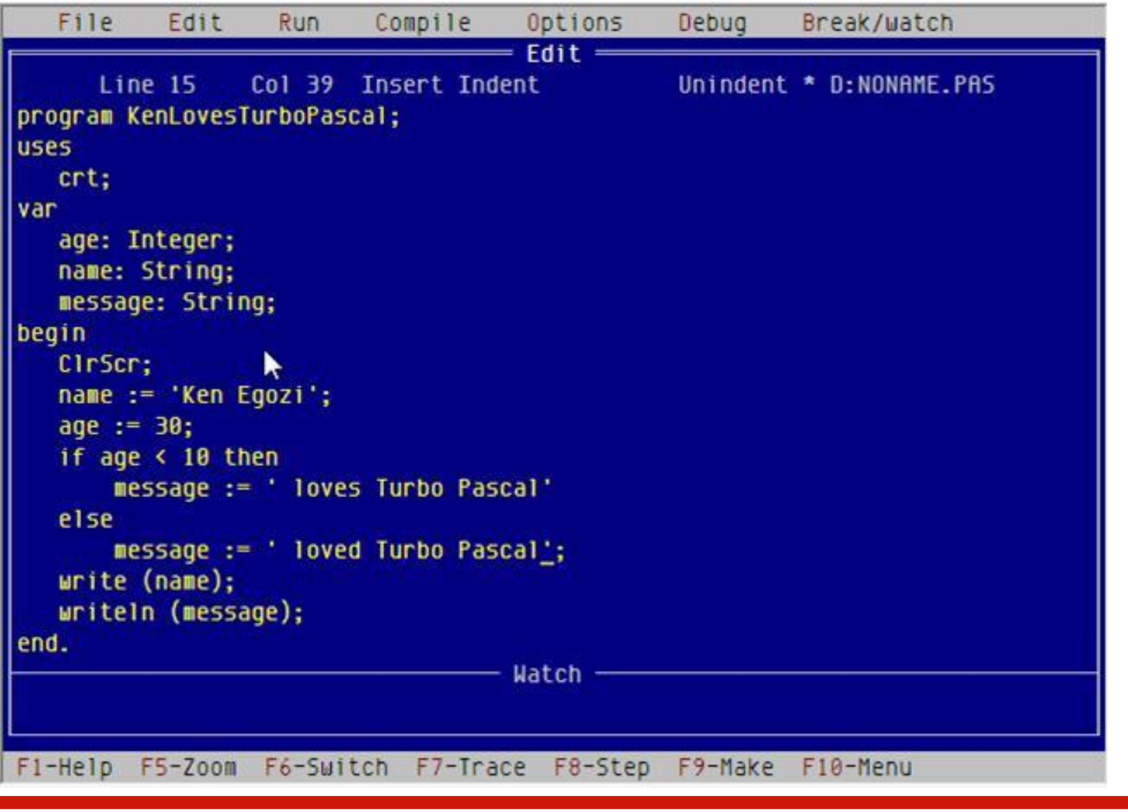
Agenda



- ➢ **Process and Threads**
- ❑ Scheduling

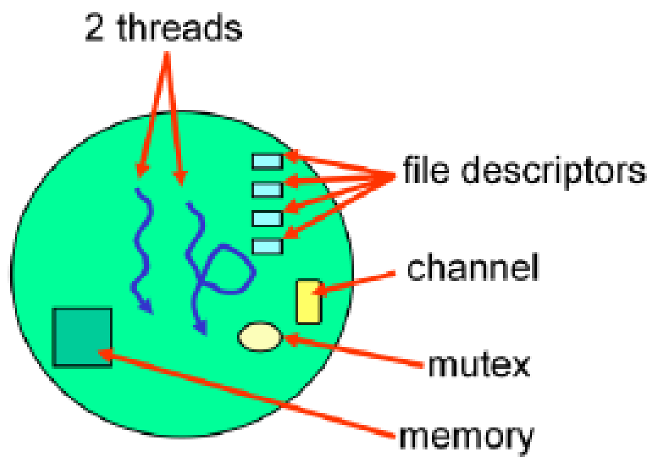
Program

- A set of instructions the user/programmer has written
- A passive entity and continues to exist at a place
- Written by a computer programmer



What is a Process?

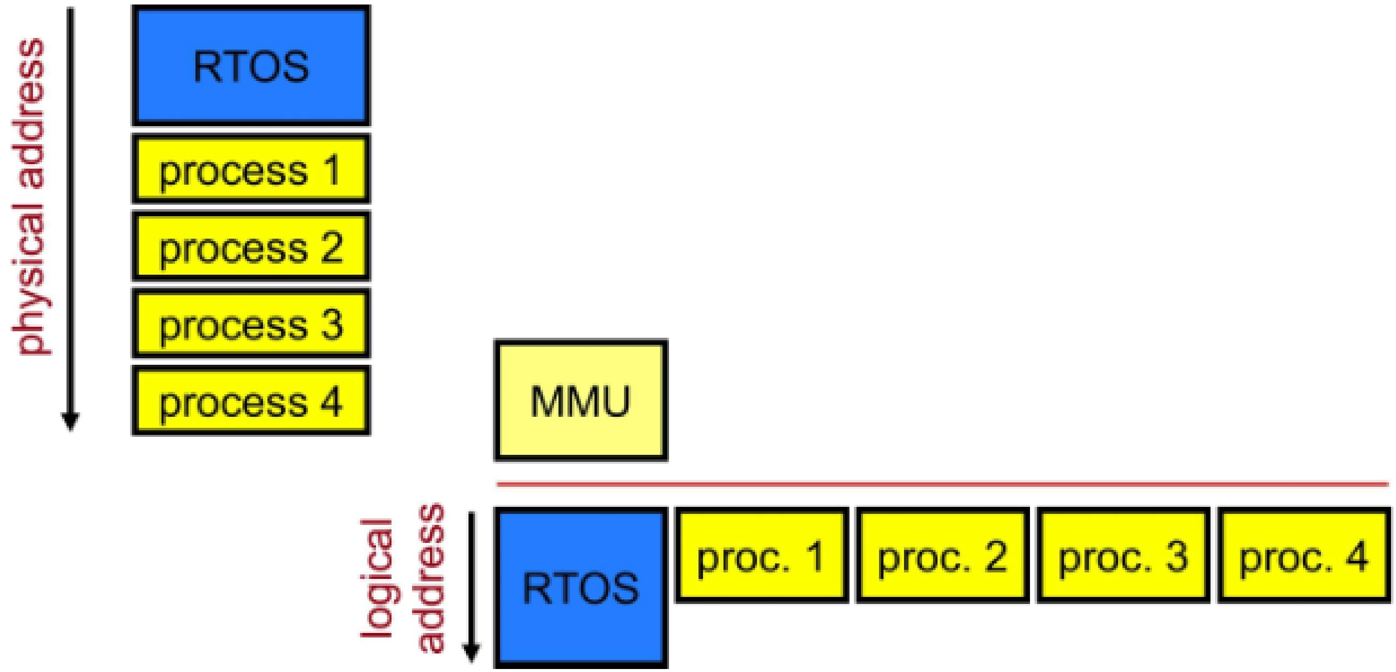
- A program loaded into memory
- A program in execution
- Own resources
  - Memory
  - Open Files
  - Identity
  - Timers
  - And more



- A resource manager with its own address space
- Resources owned by one process are protected from other process

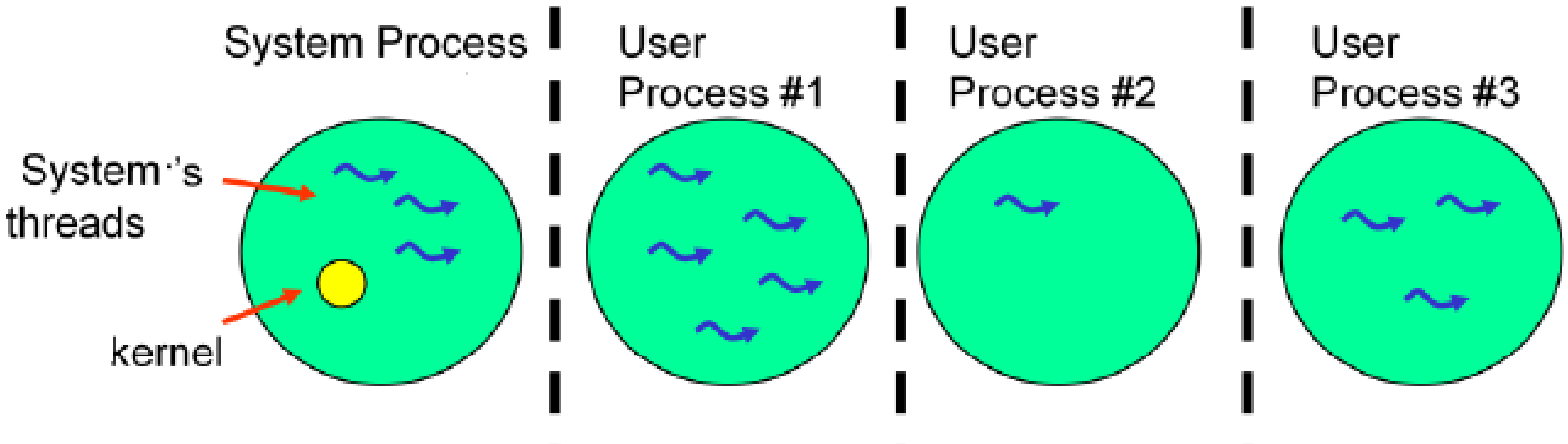
What is a Process? (cont)

- Each process has its own address space
- Hardware support for memory protection is required



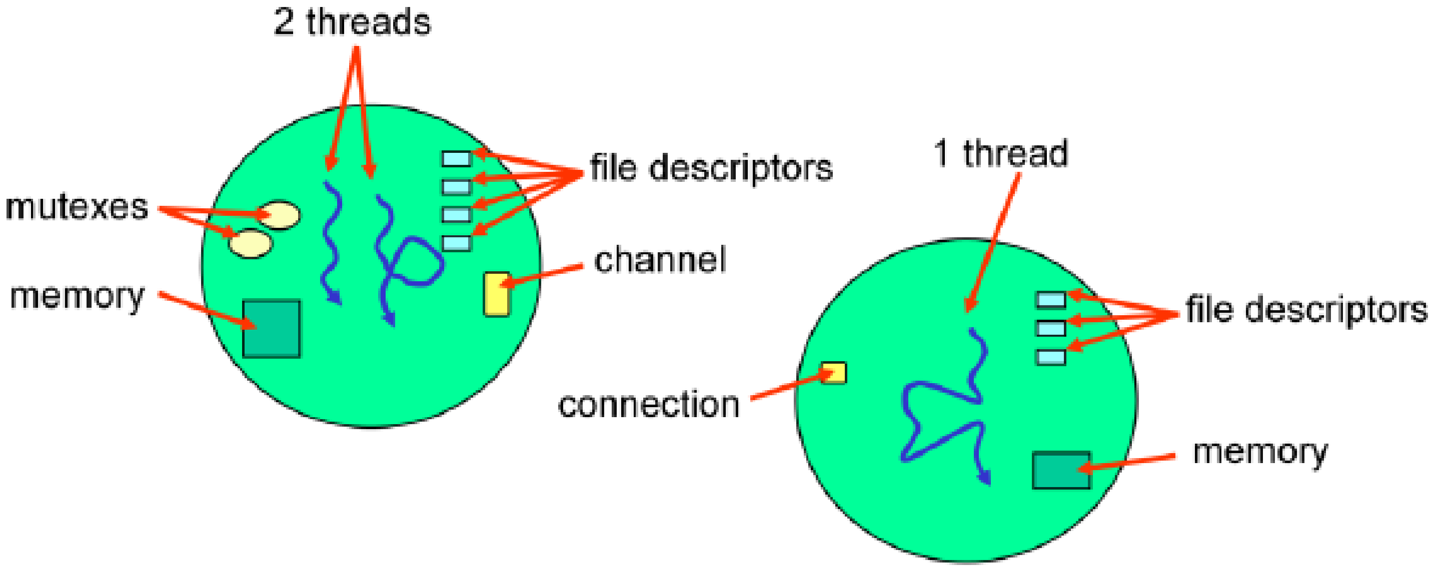
Virtual Address Space

- Each process run in its own protected virtual address space
- Pointers that you deal with contain virtual addresses
- Physical they all share the same address space



Threads

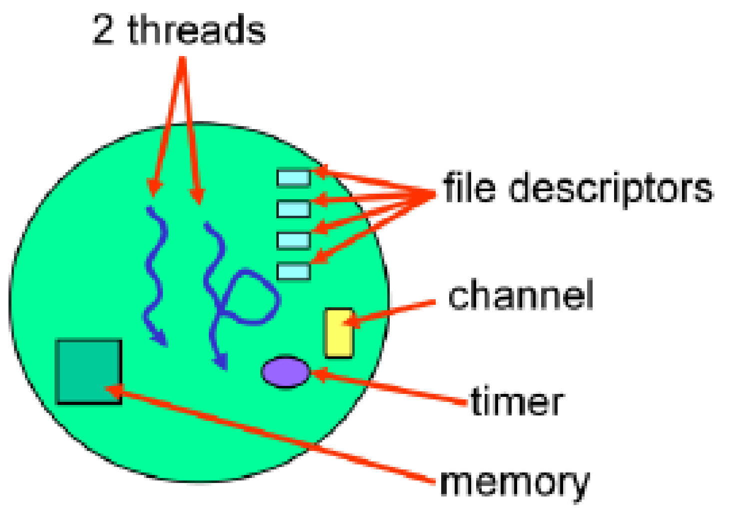
- Threads run in a process
  - A process must have at least one thread
  - Threads in a process share all resources
  - There is no protection between threads



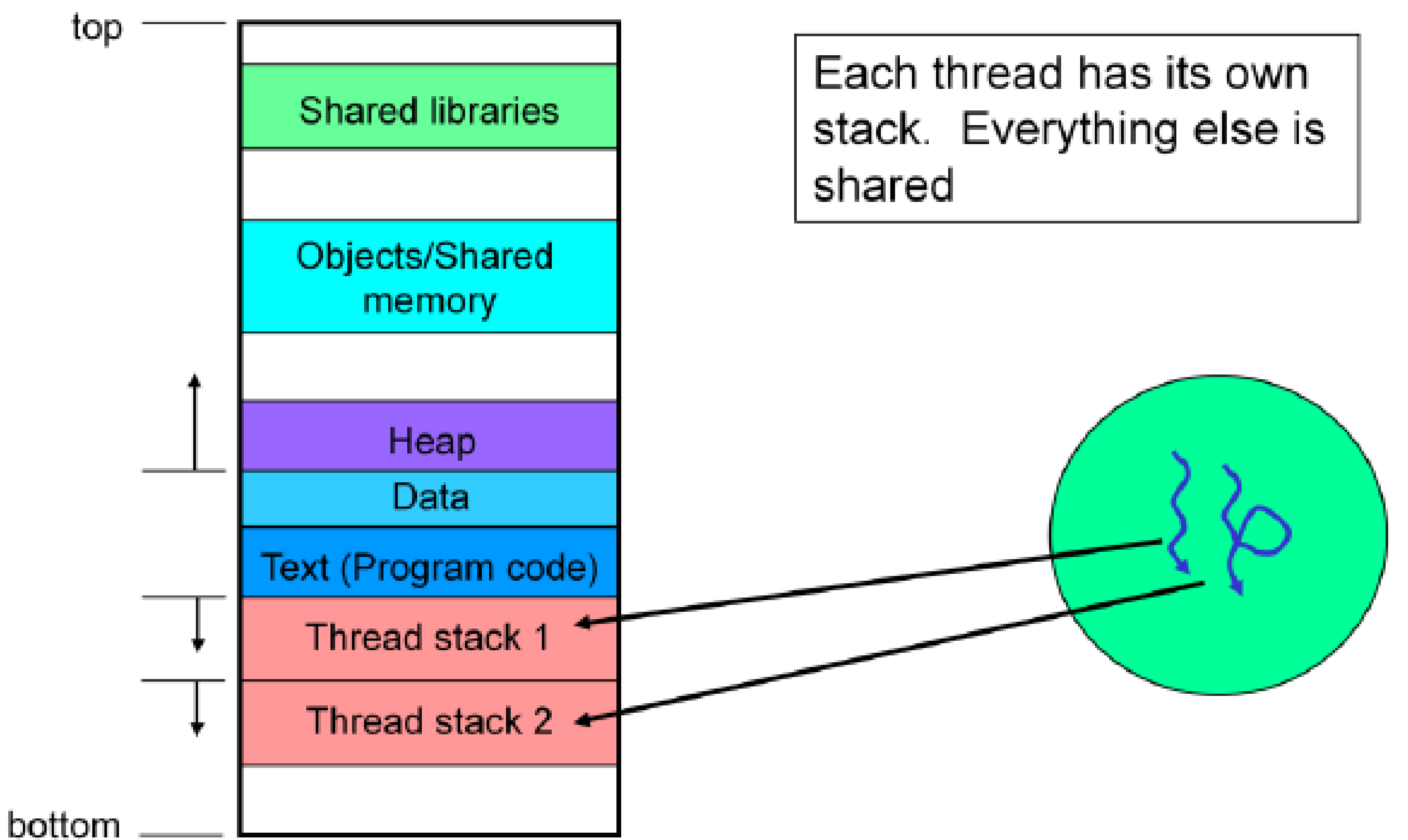
- Threads run code, process own resources

Threads (cont)

- Threads within a process share :
  - Timers
  - Channels
  - Connections
  - Memory Access
  - File pointers / descriptors
  - Signal Handlers



Virtual Address Space



Each thread has its own stack. Everything else is shared

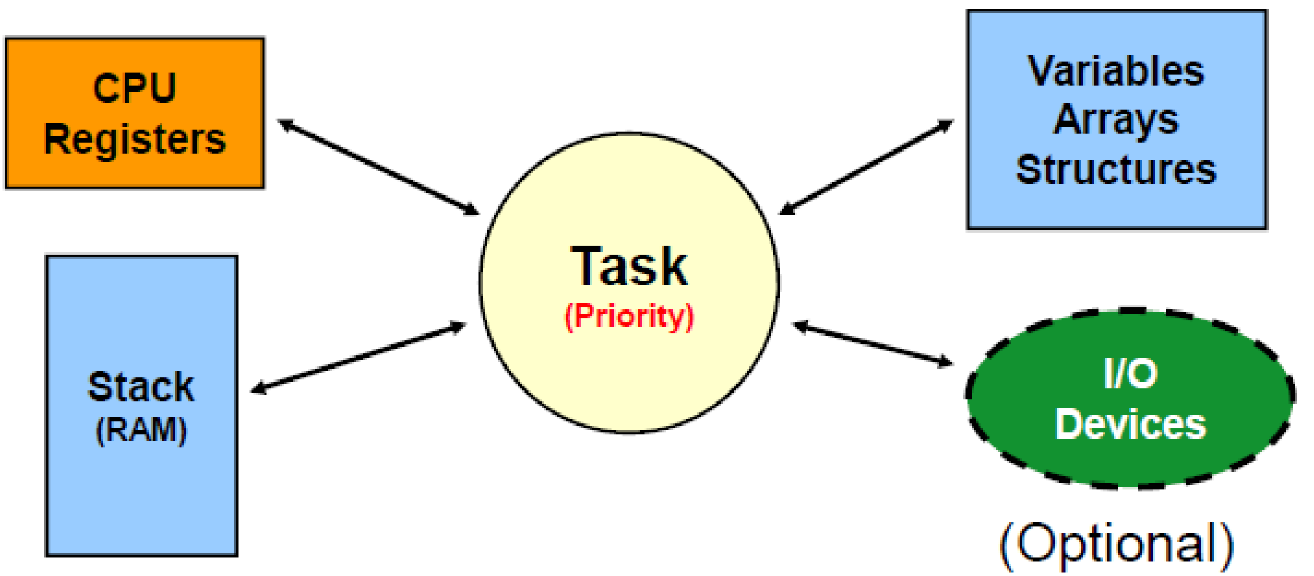
Task vs Threads

- Tasks and threads can be used interchangeably
- Most RTOSes use the word “task”
- In this training you may hear both words

All Content Copyright Edge Software

Task

- A task is a simple program that thinks it has the CPU all to itself.
- A task contains the application code



Task Types - Run to completion

```
void Task (void *p_arg)
{
    Do something with 'argument' p_arg;
    Task initialization;
    /* Processing (Your Code)                        */
    Wait for event;     /* Optional!                 */
                        /* Time to expire ...        */
                        /* Signal/Msg from ISR ...   */
                        /* Signal/Msg from task ...  */
    /* Processing (Your Code)                        */
    Delete Self;
}
```

Task Types - Infinite loop

```
void Task (void *p_arg)
{
    Do something with 'argument' p_arg;
    Task initialization;
    for (;;) {
        /* Processing (Your Code)                          */
        Wait for event;    /* Time to expire ...          */
                           /* Signal/Msg from ISR ...     */
                           /* Signal/Msg from task ...    */
        /* Processing (Your Code)                          */
    }
}
```

When can a task be created?

- At Startup

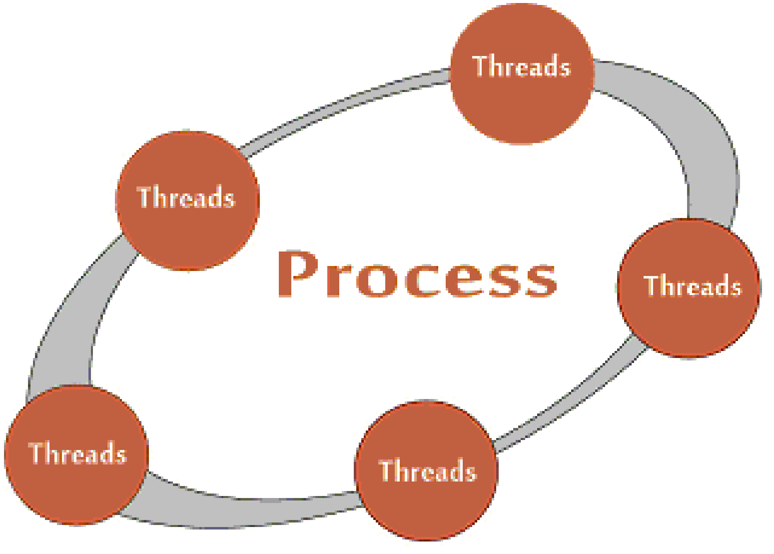
```
void main (void)
{
      :
      OSInit();
      :
      :
      OSTaskCreate(...);
      OSTaskCreate(...);
      :
      :
      OSStart();
}
```

When can a task be created?

- At Run-Time

```
void MyTask (void *p_arg)
{
    :
    for (;;) {
        :
        :
        OSTaskCreate(...);
        OSTaskCreate(...);
        :
        :
    }
}
```

Agenda



- ❑ Process and Threads
- ➢ **Scheduling**

The Scheduler

- The scheduler is at the heart of every kernel
- The scheduler is also called the dispatcher
- It determines which will run next and when

Schedulable Entities

- A schedulable entity is a kernel object that can compete for execution time on a system

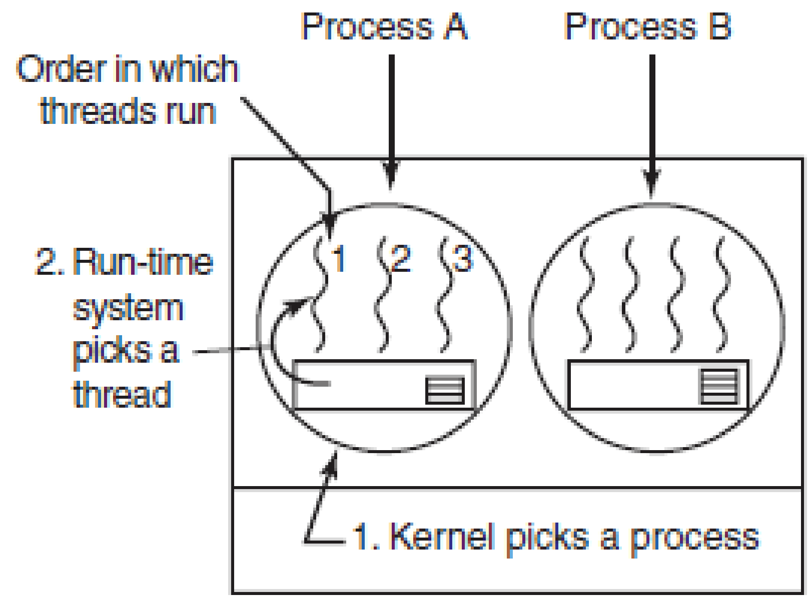
- Tasks/threads and process are all examples of schedulable entities

- Most kernels schedule tasks/threads only

- If process scheduling is provided, a second level scheduler is also required to schedule the thread to execute(Process is not a runnable entity)

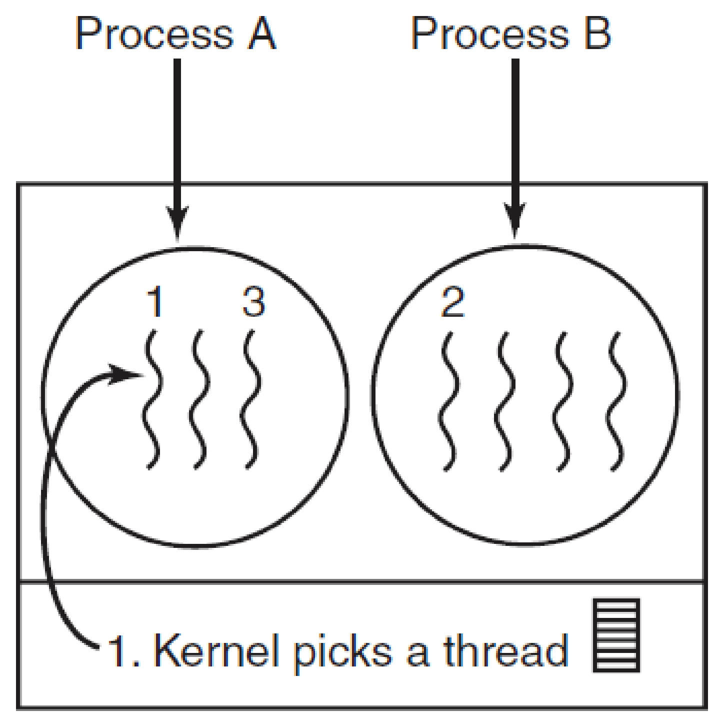
Schedulable Entities (cont)

- Two level scheduling
  - First a process is selected
  - Then a thread of the process is selected to run next
  - Example for two level scheduling : VxWorks 653



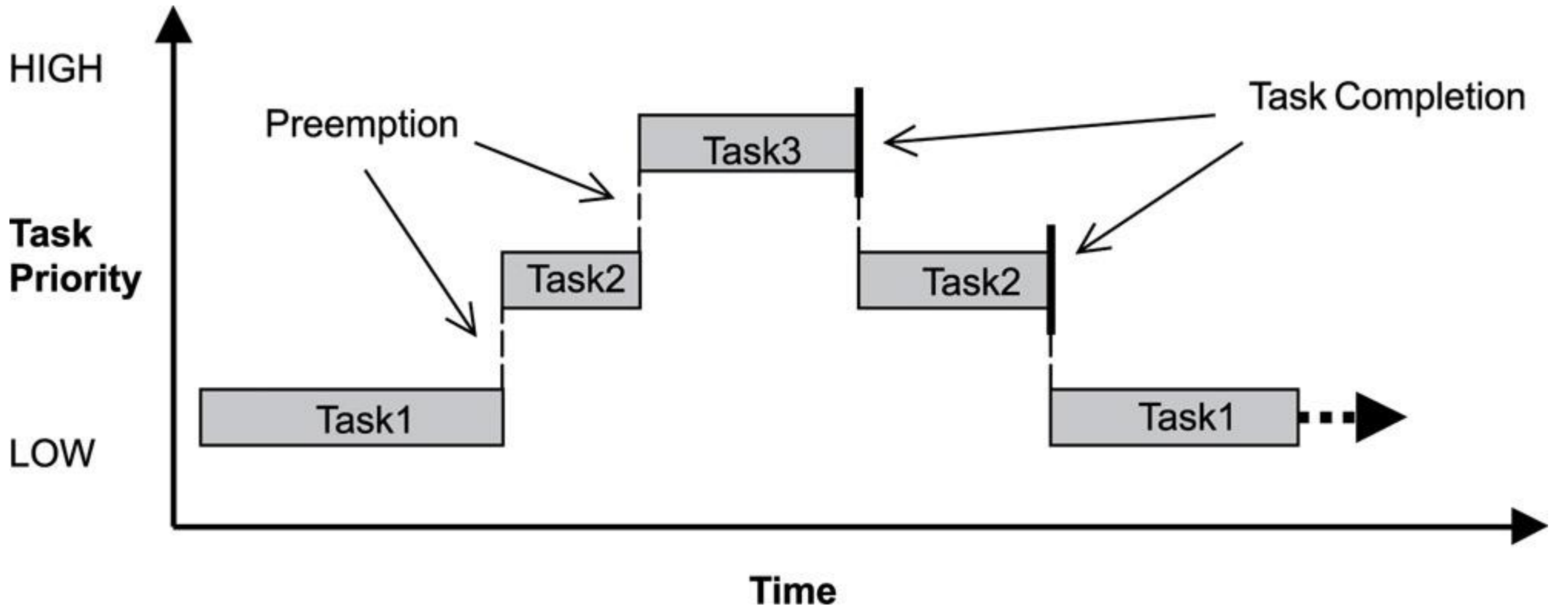
Schedulable Entities (cont)

- Single level scheduling
  - Kernels picks a thread
  - It does not have to take into account which process the thread belongs to
  - Linux, Windows, Integrity, QNX



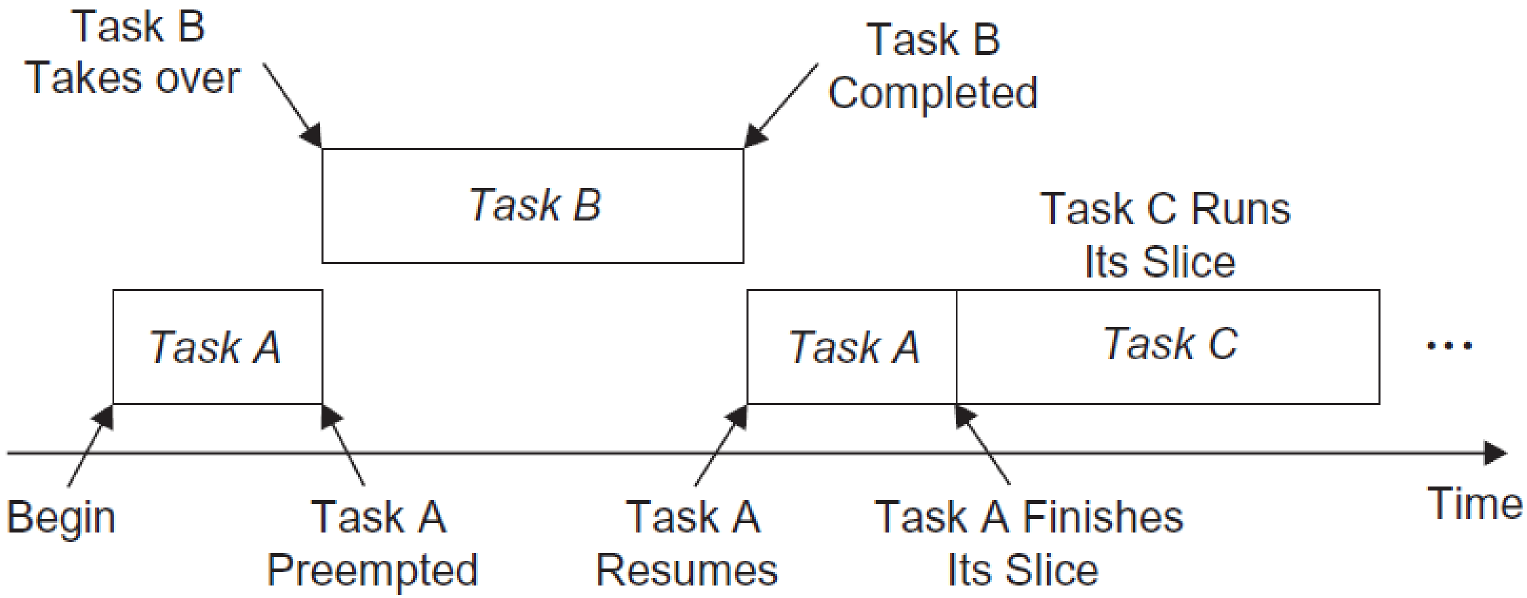
Preemptive priority-based scheduling

- All real-time kernels use preemptive priority-based scheduling by default.
- The highest priority READY thread is the one which gets the CPU



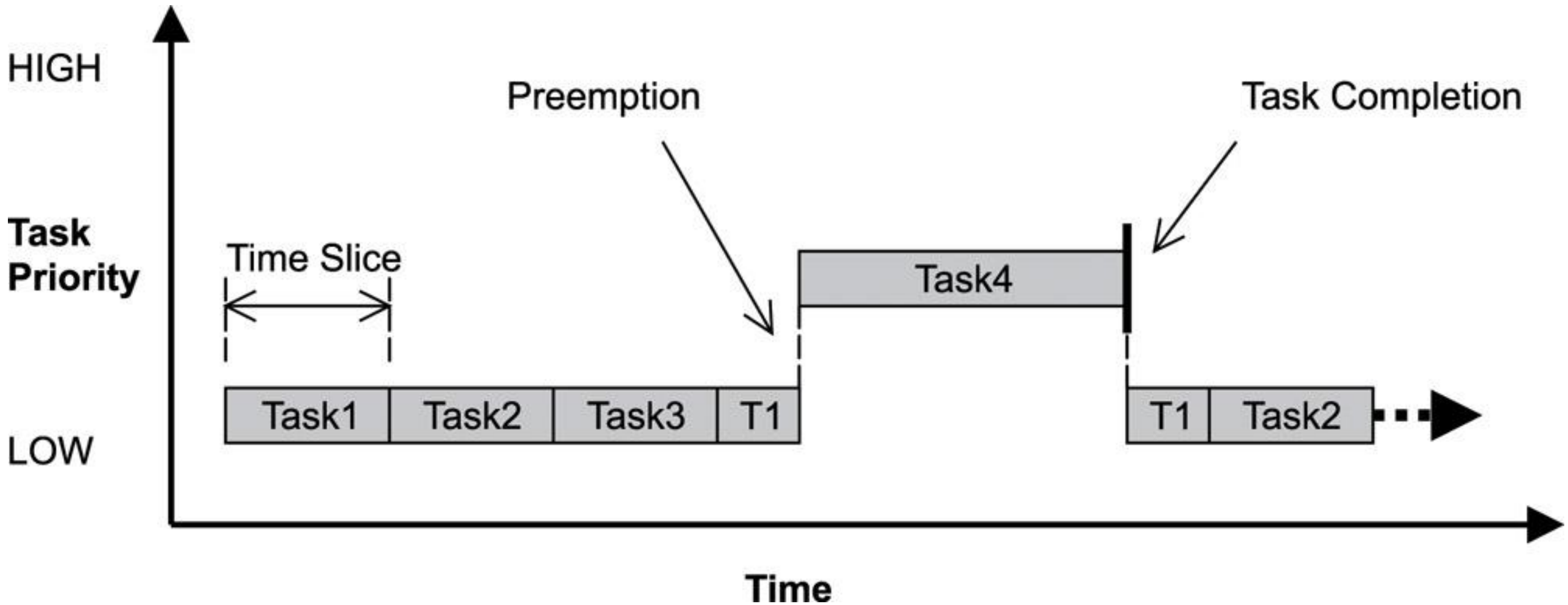
Round-Robin Scheduling

- Preemptive, priority-based scheduling can be augmented with round-robin scheduling which uses time slicing to achieve equal allocation of the CPU for tasks of the <u>same priority</u>



Round-Robin Scheduling (cont)

- Each executable task is assigned a fixed time quantum called a time slice
- A fixed rate clock is used to initiate an interrupt at a rate corresponding to the time slice.



The Context Switch

- Each task has its own context

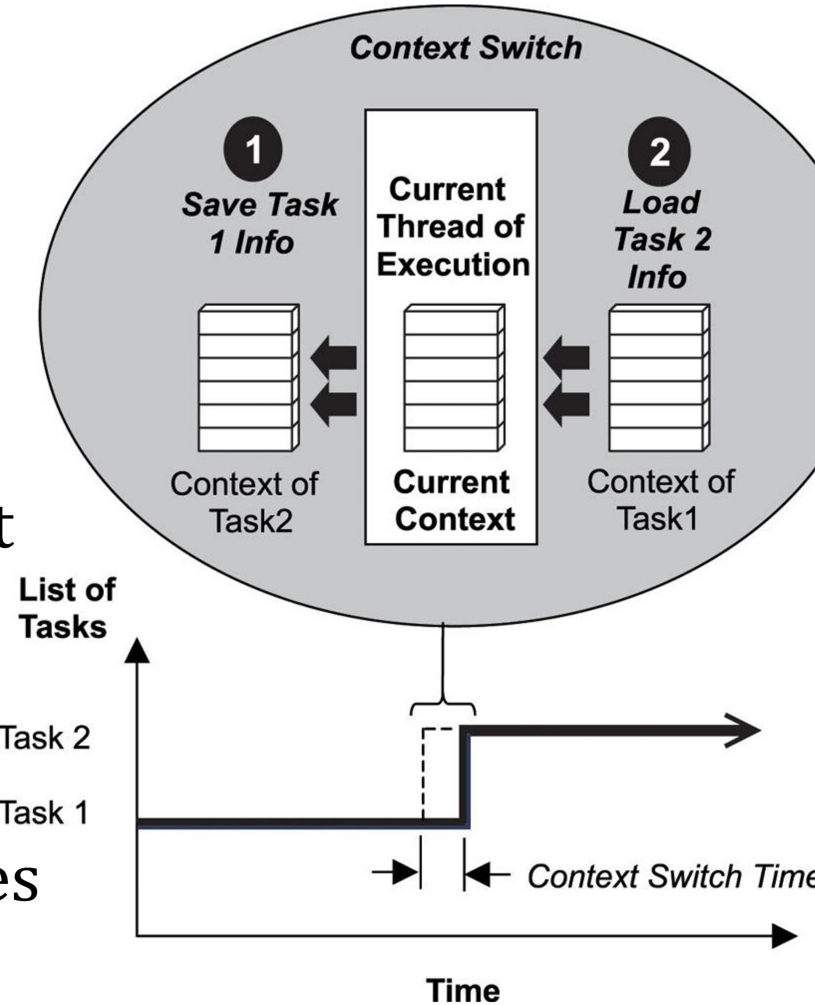
- A context switch occurs when the scheduler switches from one task to another

- When a task is running, its context is highly dynamic

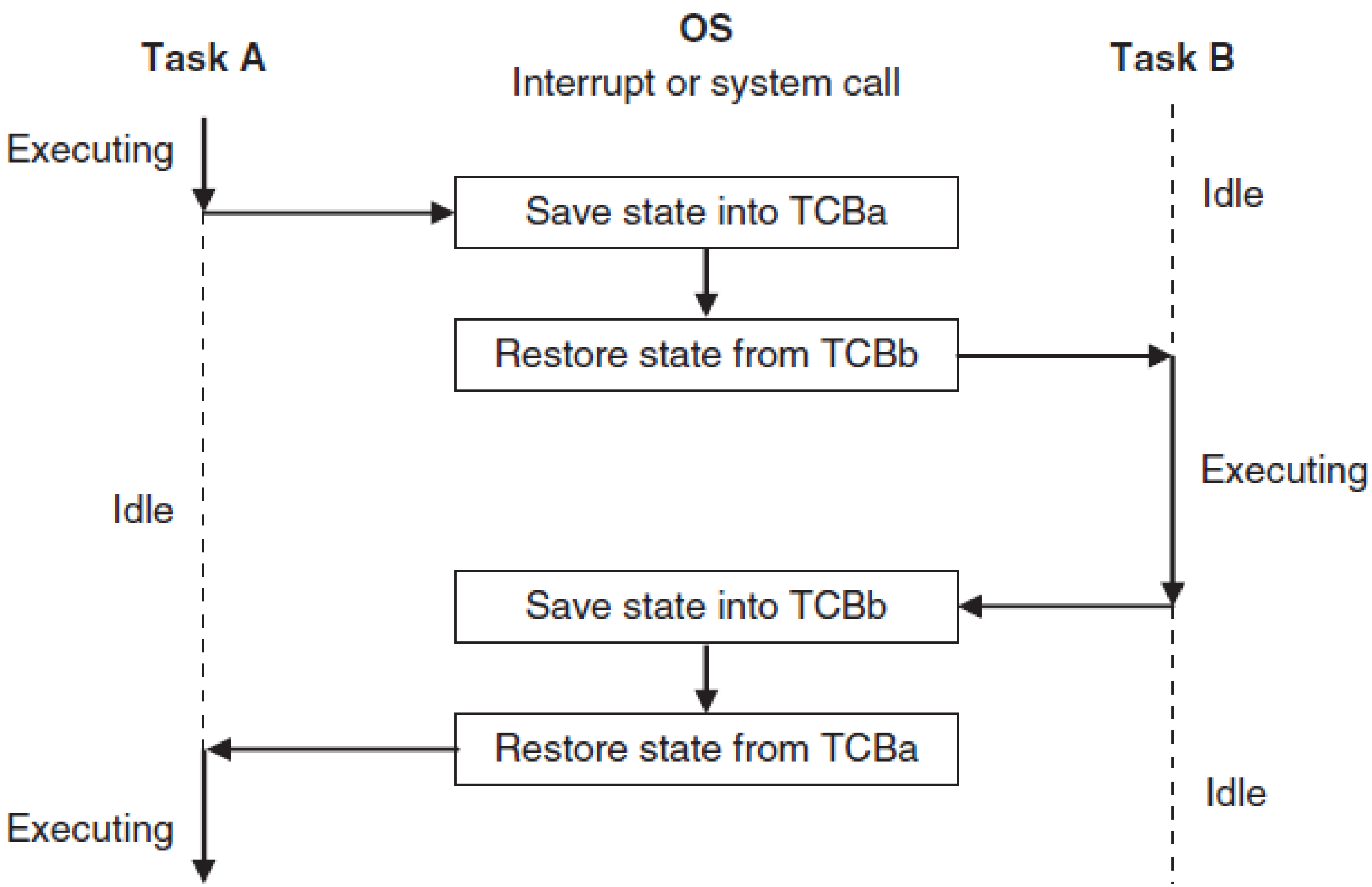
- When the task is not running, its context is frozen within the TCB(or stack), to be restored the next time the task runs

The Context Switch (cont)

- ❑The kernel saves task 1's context information in its TCB.
- ❑It loads task 2's context information from its TCB, which becomes the current thread of execution.
- ❑The context of task 1 is frozen while task 2 executes



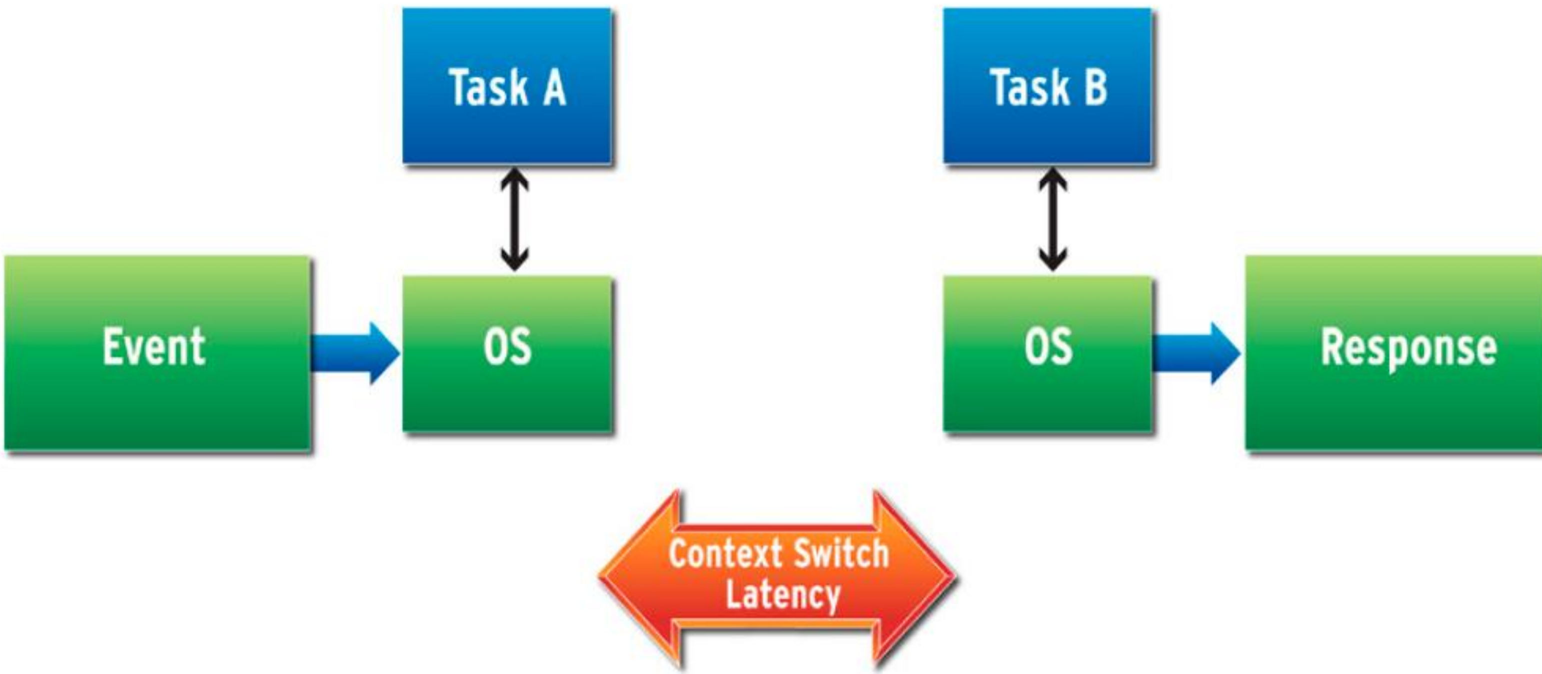
Context switch between tasks A and B



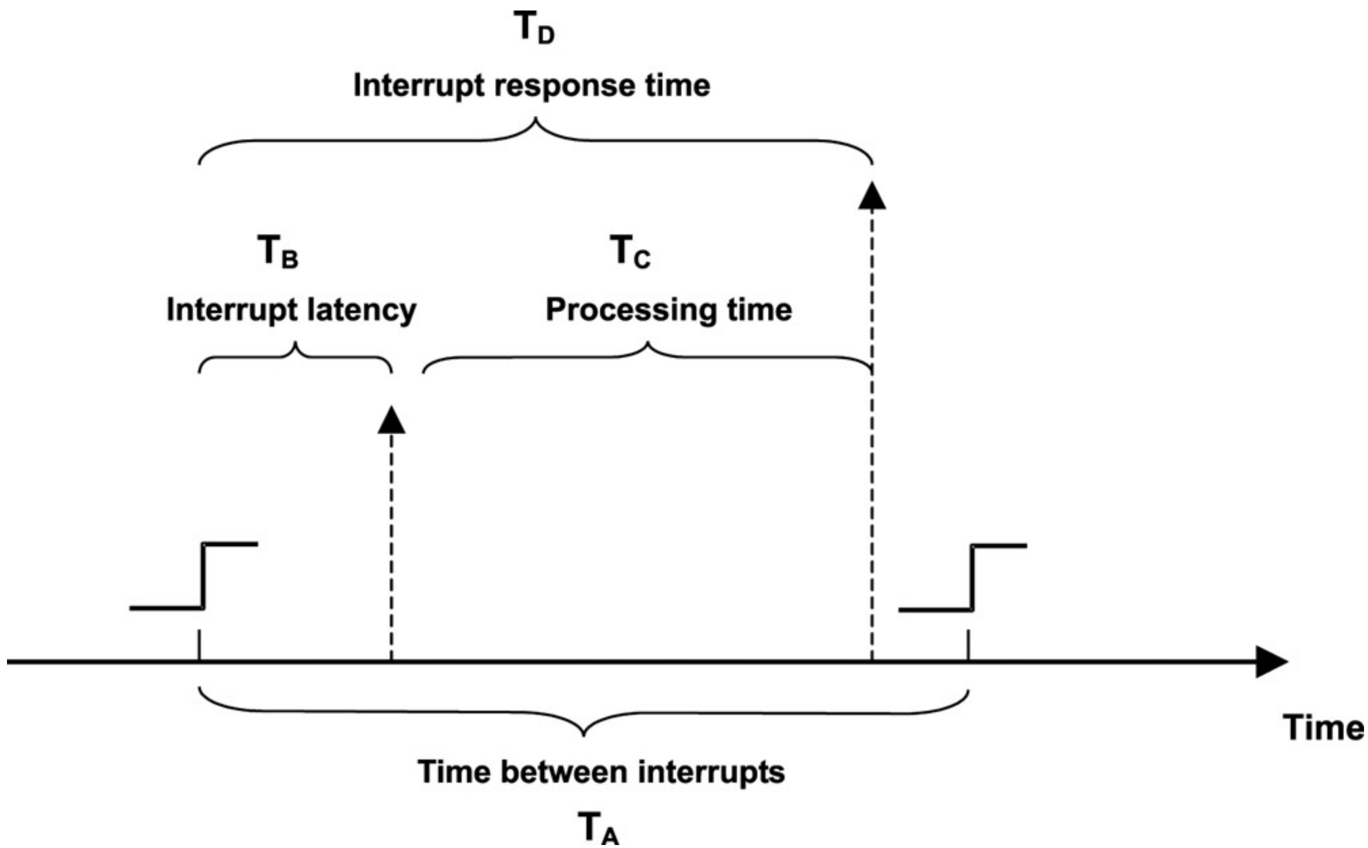
Context Switch Time

- The time it takes for the scheduler to switch from one task to another is the context switch time
- Context switch time is relatively insignificant compared to most operations that a task performs
- If an application’s design includes frequent context switching the application can incur unnecessary performance overhead

Context Switch Time (cont)



Interrupt Response Time

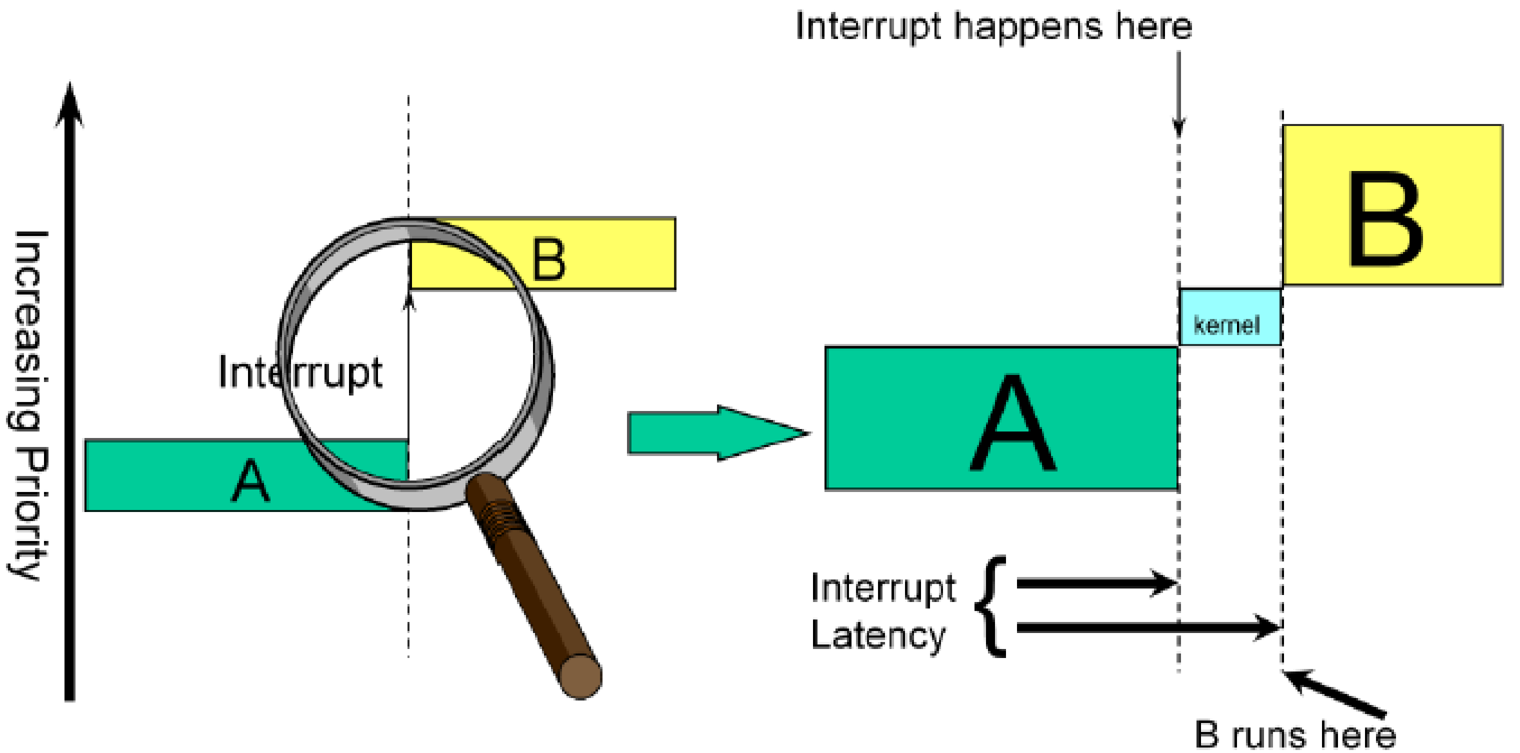


The interrupt response time is $T_D = T_B + T_C$.

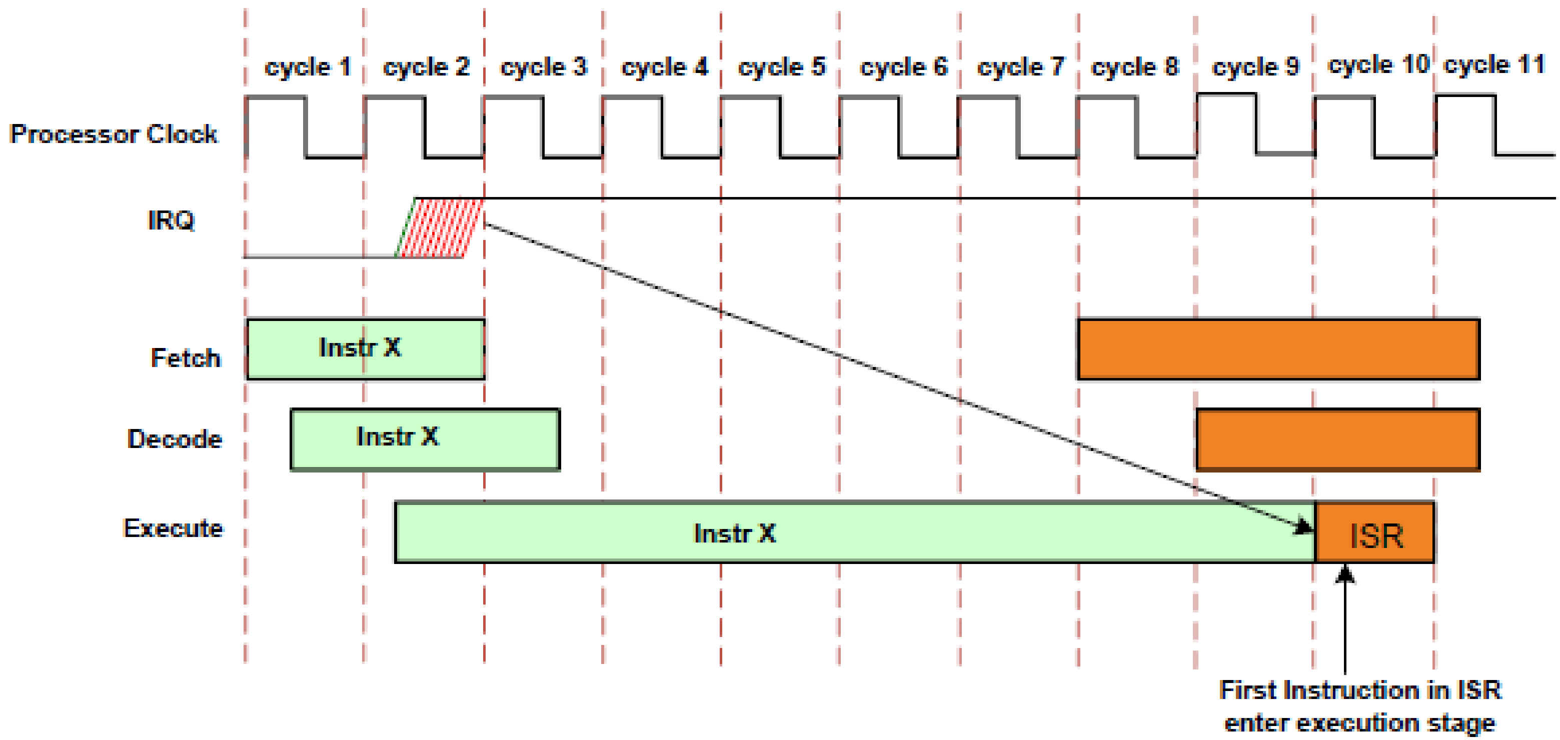
What affects Interrupt Latency?

- Time spent with interrupts disabled
- Time spent in an equal or higher priority interrupt handler
- Time spent in other handlers for this interrupt
- Time spent with the particular interrupt level disabled

Interrupt Latency



Interrupt Latency in terms of processor clock cycles



Interrupt and Scheduling Relation

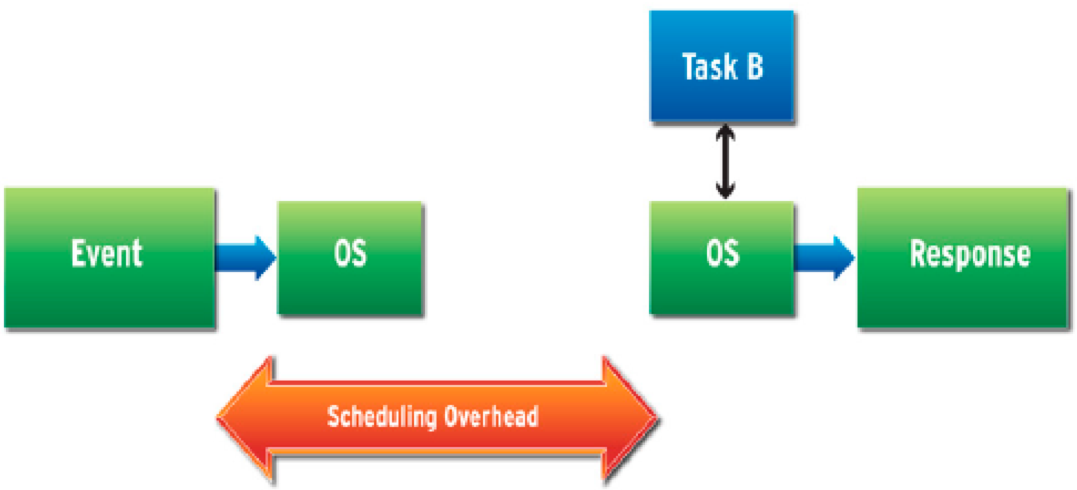
- An interrupt always has higher priority than any thread priority in the system

- Time spent in an interrupt handler has a direct impact on scheduling

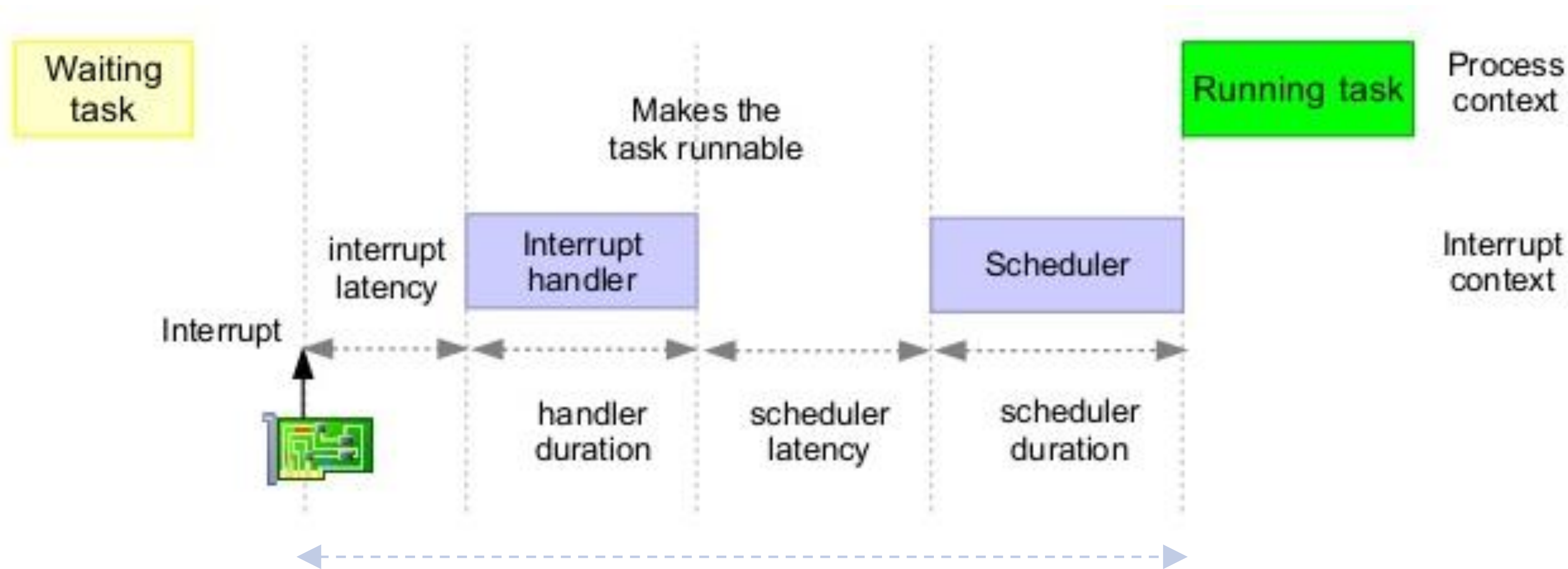
All Content Copyright Edge Software

Scheduling Latency

- Real-time threads need to be scheduled as soon as they have something to do
- However, there is always a delay from the point at which the wake-up event occurs
- This is called scheduling latency.

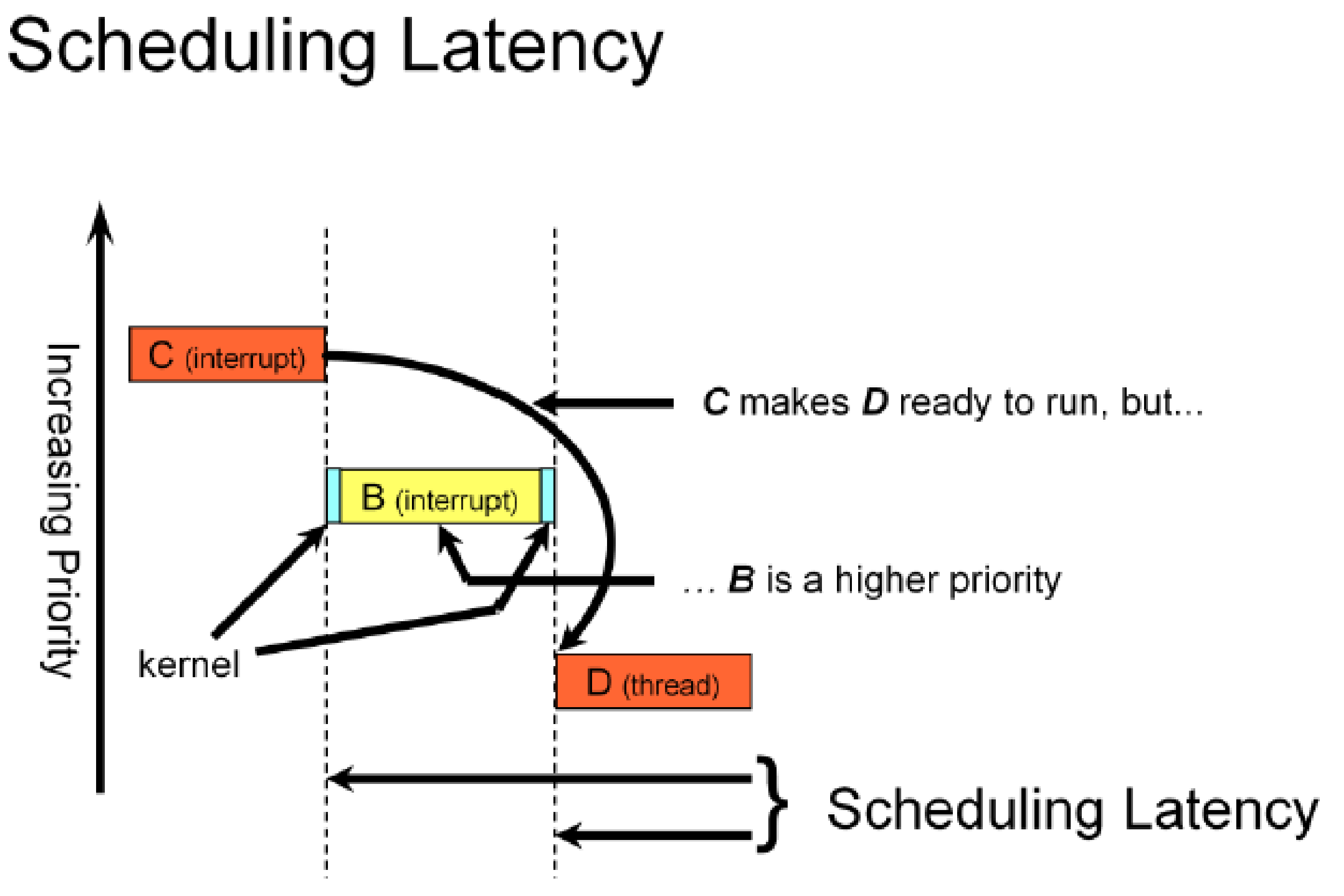


Scheduling Latency (cont)



Scheduling Latency

Scheduling Latency (cont)

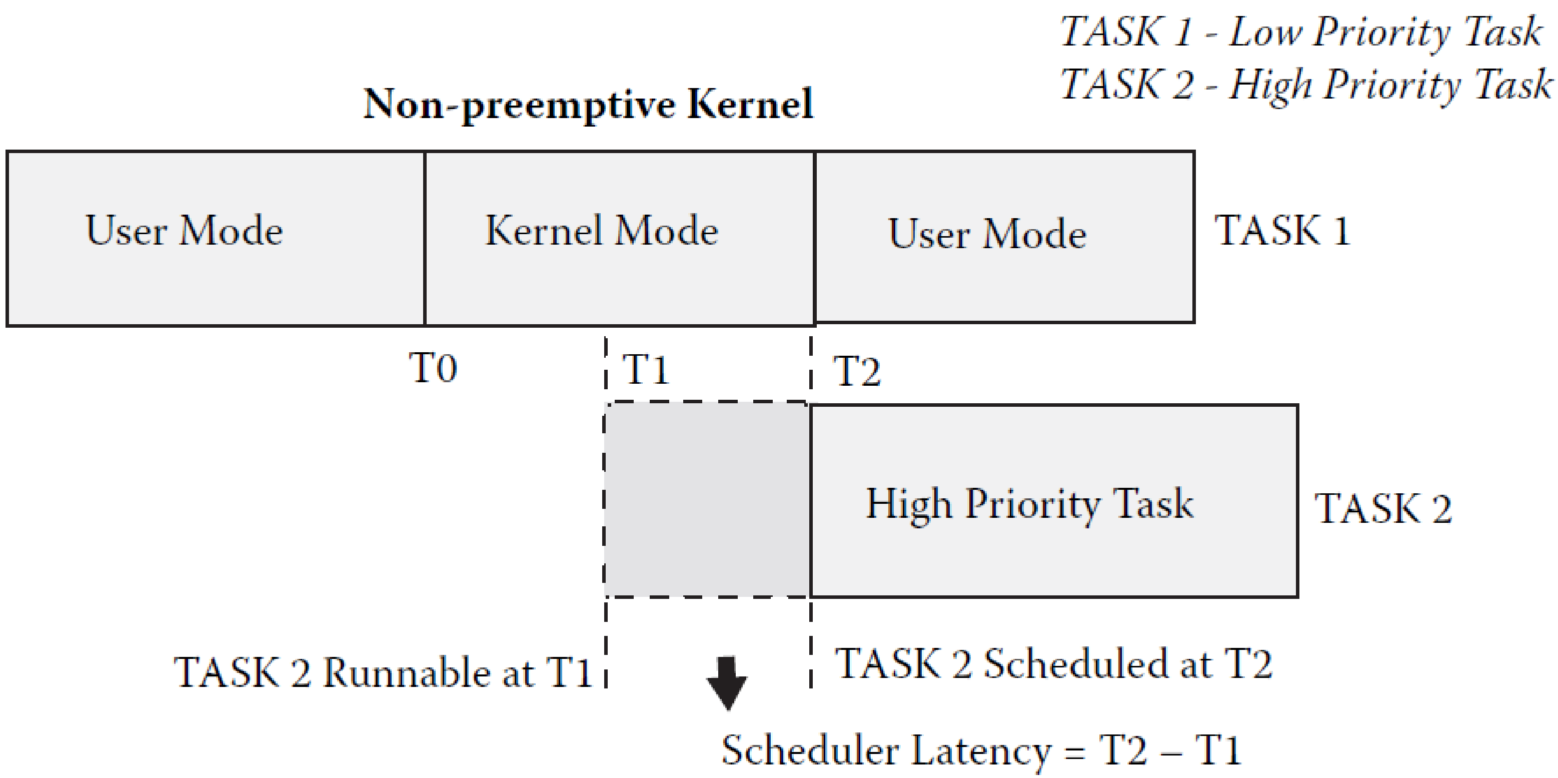


Scheduling Latency (cont)

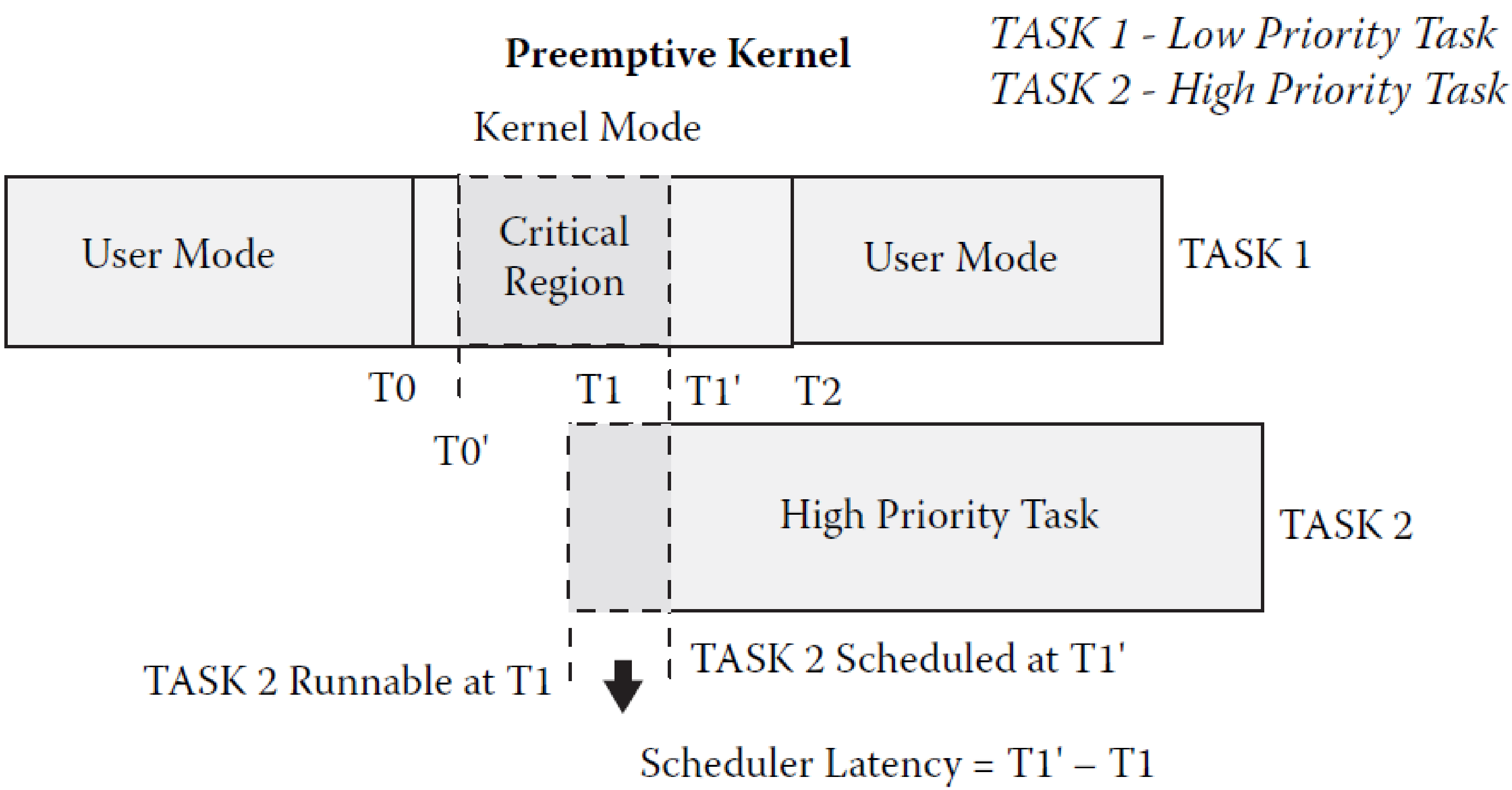
- What affects scheduling latency
  - Time spent in the interrupt handlers
  - Priority of the thread scheduled
  - Context switch time
  - Scheduler latency
  - Time spent in higher priority threads that are also ready

All Content Copyright Edge Software

Non-preemptive Kernel



Preemptive Kernel



Scheduler Duration

- Scheduler duration is the time taken by the scheduler to select the next task for execution and context switch to it

- The scheduler duration should be constant regardless of the number of tasks in the system

The idle task

- When there is “nothing to do”, the processor still executes instructions

- Most kernels create an internal task called the idle task

- The idle task basically runs when no other application task is able to run

- The idle task is the lowest-priority task in the application and is a “true” infinite loop that never calls functions to “wait for an event”.

Priority levels

- All kernels allow you to assign priorities to tasks based on their importance in your application

- The number of different priority levels greatly depends on the implementation of the kernel.

- Most kernels allow the priority of tasks to be changed dynamically at run-time through a “change a task’s priority” service

Who schedules the scheduler ?

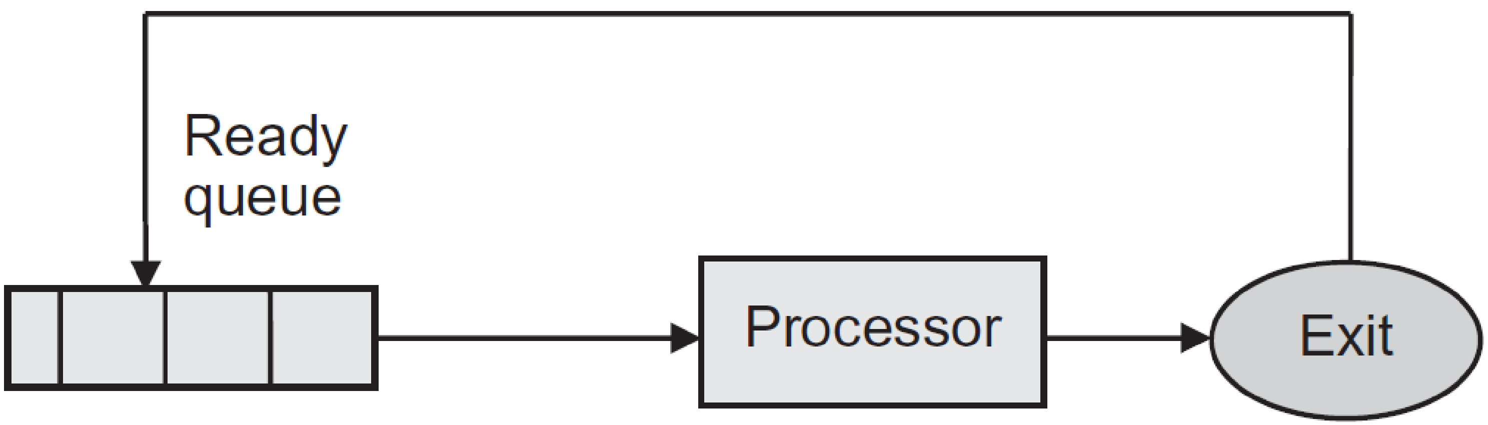
- Scheduler is not scheduled
- <u>There is not a scheduler task</u>
- Scheduler is executed
  - During a system call
  - After an ISR processing
- Scheduling occurs at predefined scheduling points

Who schedules the scheduler ? (cont)

- Nothing special must be done in the application code since scheduling occurs automatically based some conditions

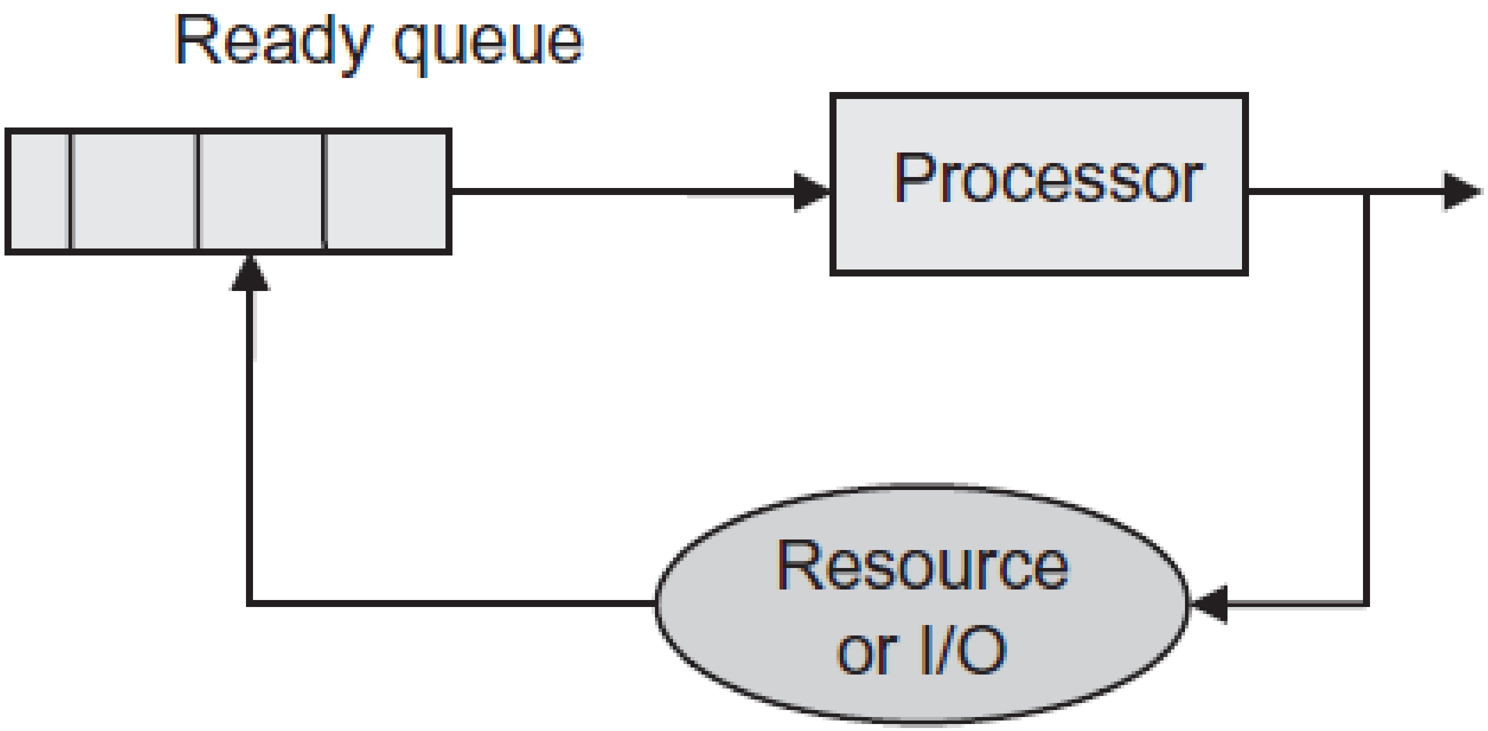
When to Schedule?

- Scheduling reason: Running thread exits



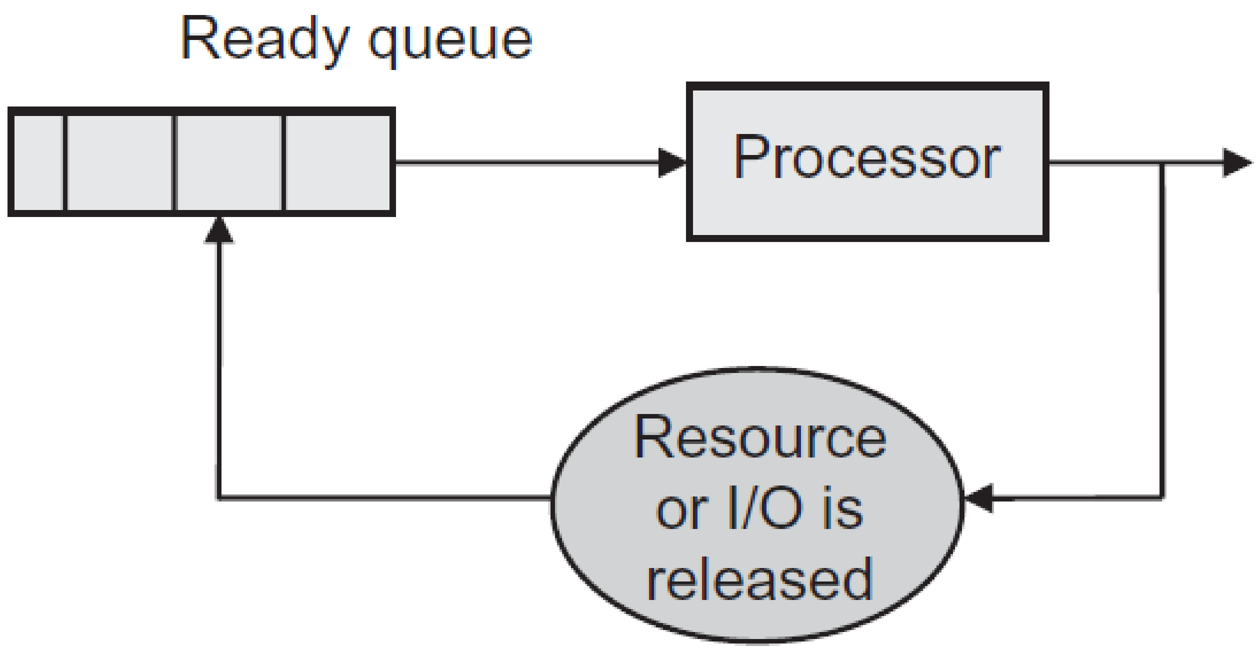
When to Schedule? (cont)

- Scheduling reason: Running thread enters in a wait



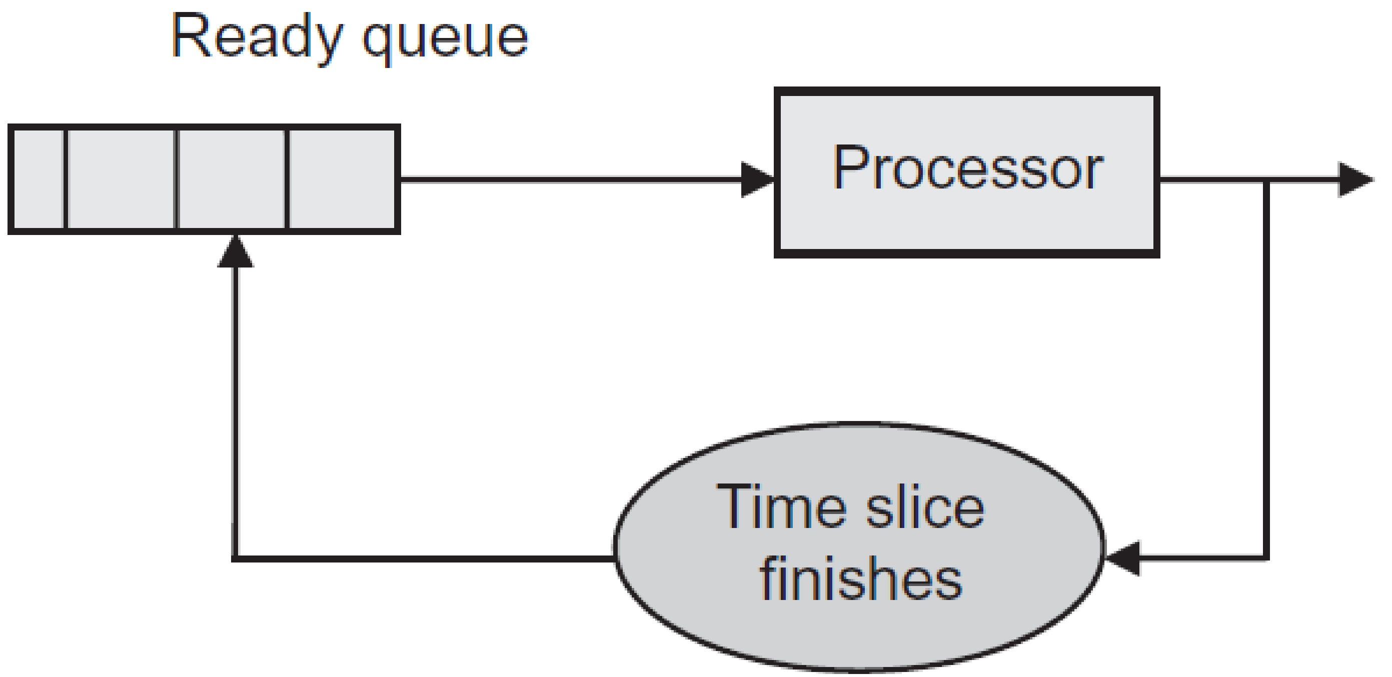
When to Schedule? (cont)

- Scheduling reason: I/O or resource is released



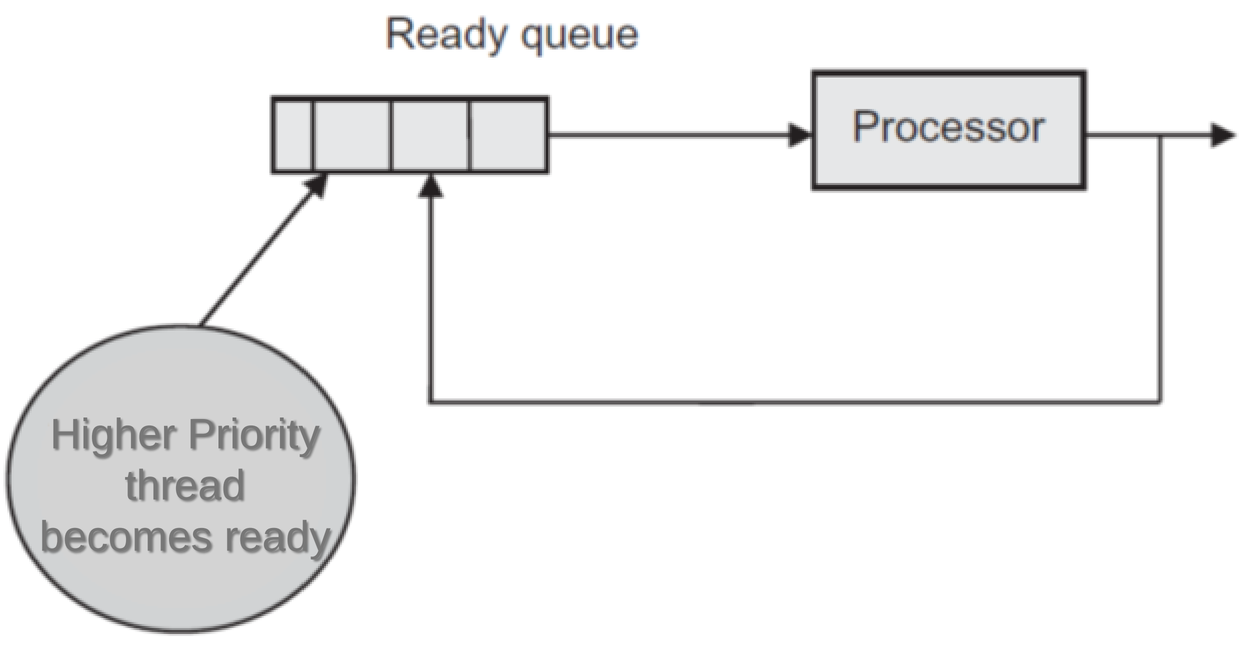
When to Schedule? (cont)

- Scheduling reason: Time slice of running thread finishes



When to Schedule? (cont)

- Scheduling reason: Higher priority thread becomes ready



Chapter 3

RTOS vs GPOS

RTOS vs GPOS

- In a GPOS, the scheduler typically uses a fairness policy to dispatch threads and processes onto the CPU.

- Such a policy enables the high overall throughput

- Most GPOSs have unbounded dispatch latencies

- General Purpose Operating systems are very good at what they are designed for

RTOS vs GPOS (cont)

- In an RTOS priority based preemptive scheduling policy is employed
- An RTOS can sacrifice throughput for being deterministic
- In an RTOS <u>dispatch latencies are constant</u>
- RTOSes are best suited for real-time embedded systems

Chapter 4

RTOS is only a tool

A Famous Quote from Grady Booch

A fool with a tool is still a fool

All Content Copyright Edge Software

RTOS is only a tool

- Using an RTOS does not make a system a Real-Time system.
- Incorrect usage of an RTOS may end up with a system failure
- Porting an application from a GPOS to an RTOS may require rearchitecting(API porting is totally dangerous)

All Content Copyright Edge Software

RTOS is only a tool

- Improper preemption usage may cause deadlock
- A GPOS may tolerate design errors but an RTOS most probably not
- RTOSes provide a toolset to model a Real-Time System

RTOS is only a tool

- If the model or the design has problems the final product will have the same problems

- Try to decrease latencies and meet the deadlines of the system. This may decrease the total throughput of the system

- A Real-Time system is not a system with the maximum performance

All Content Copyright Edge Software