BCSE305L-EMBEDDED SYSTEM DESIGN

MODULE-5

REAL-TIME OPERATING SYSTEM

MODULE-6

Introduction to Real-Time Concepts

RTOS Internals & Real Time Scheduling, Performance Metrics of RTOS, Task Specifications, Schedulability Analysis, Application Programming on RTOS.

RTOS - INTRODUCTION

RTOS - INTRODUCTION

NEED FOR RTOS

Why not GPOS or EOS?

RTOS - INTRODUCTION

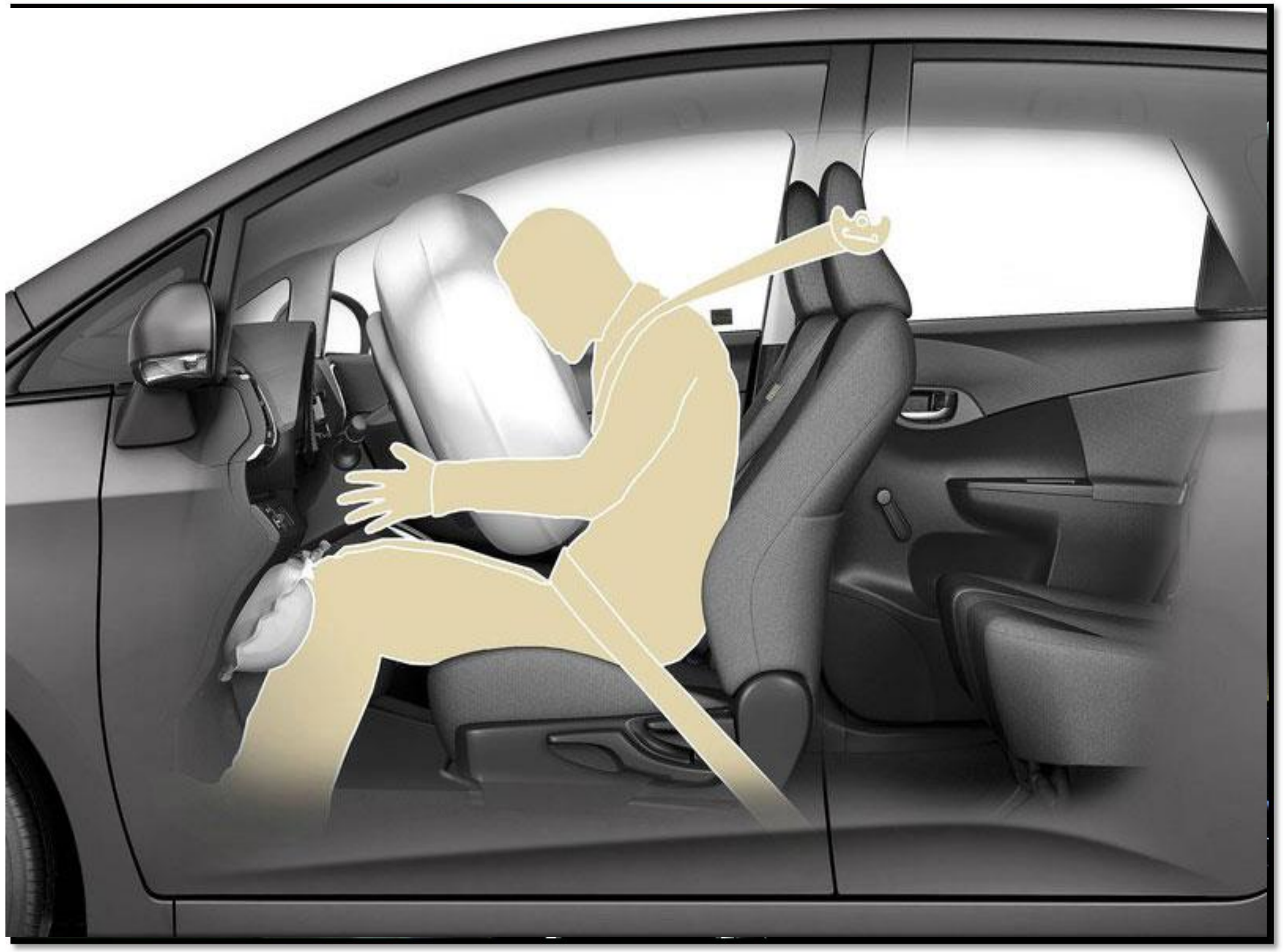
NEED FOR RTOS

Microsoft Windows
Windows Explorer has stopped working
Windows is checking for a solution to the problem...
Cancel

When a fighter jet control system runs on GPOS encountered a hill, and you attempted to avoid…

RTOS - INTRODUCTION

NEED FOR RTOS



While you drive your car at high speed, suddenly collided with another vehicle if airbag activation system runs on EOS opens only after 10 sec....

RTOS - INTRODUCTION

WHAT IS RTOS?

- "Real time in operating systems is the ability of the OS to provide a required level of service in a bounded response time." - POSIX Standard 1003.1
- "A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time application requests." - Wikipedia
- "RTOS: Any OS where interrupts are guaranteed to be handled within a certain specified time, there by making it suitable for control hardware in embedded system & time critical applications." - Dictionary.com
- A real-time operating system (RTOS) is an operating system that guarantees a certain capability within a specified time constraint to serve real time embedded applications.

RTOS - INTRODUCTION

RTOS - FEATURES

- **Multitasking and Pre-emptibility:** An RTOS must be multi-tasked and pre-emptible to support multiple tasks in real-time applications.
- **Task Priority:** In RTOS, pre-emption capability is achieved by assigning individual task with the appropriate priority level.
- **Inter Task Communication:** For multiple tasks to communicate in a timely manner and to ensure data integrity among each other, reliable and sufficient inter-task communication and synchronization mechanisms are required.
- **Priority Inheritance:** To allow applications with stringent priority requirements to be implemented, RTOS must have a sufficient number of priority levels when using priority scheduling.

RTOS - INTRODUCTION

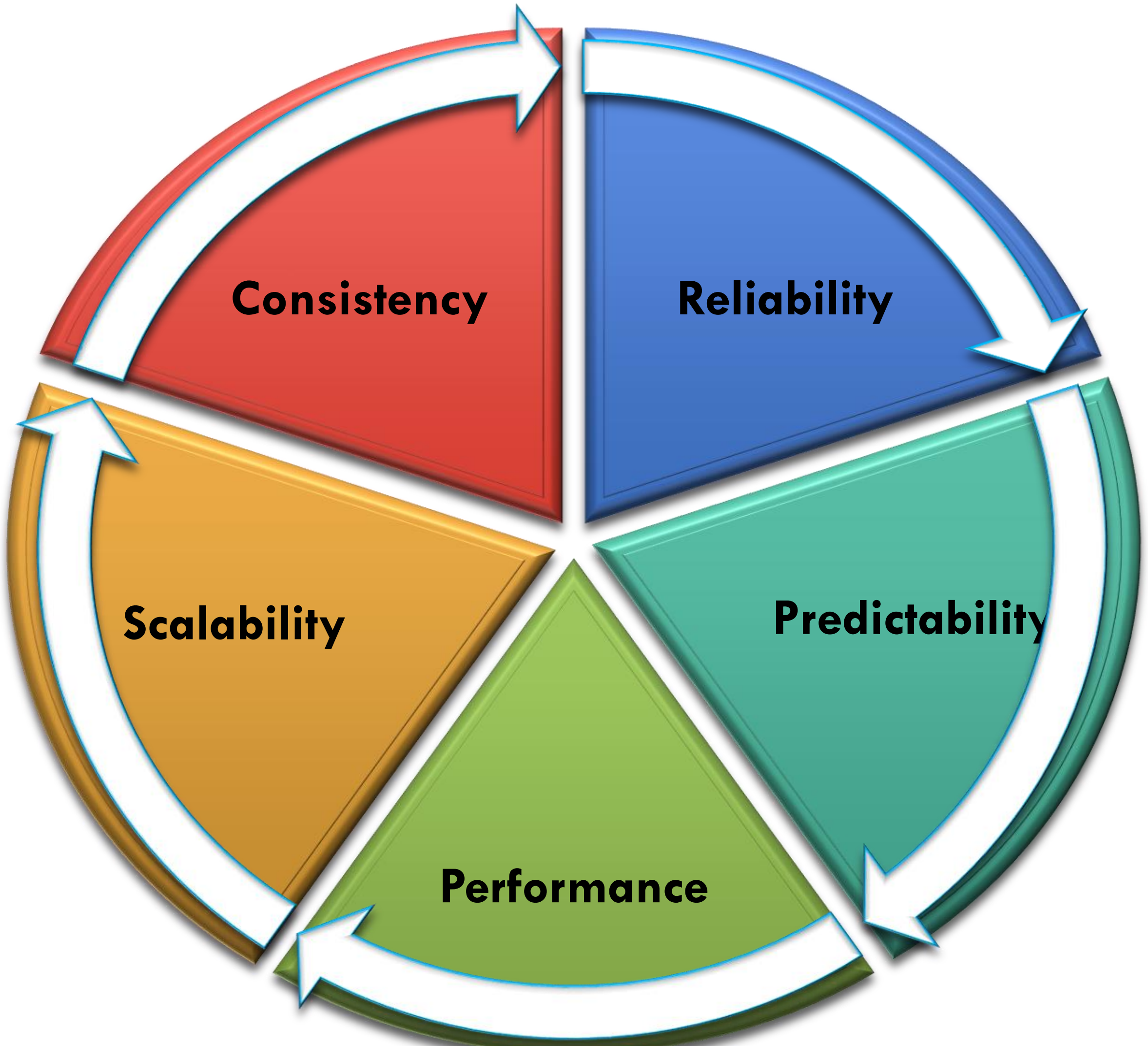
RTOS - FEATURES

- **Control of Memory Management:** To ensure predictable response to an interrupt, an RTOS should provide way for task to lock its code and data into real memory.

- **Predefined Short Latencies:**
  - **Task switching latency:** The time needed to save the context of a currently executing task and switching to another task is desirable to be short.
  - **Interrupt latency:** The time elapsed between execution of the last instruction of the interrupted task and the first instruction in the interrupt handler.
  - **Interrupt dispatch latency:** The time from the last instruction in the interrupt handler to the next task scheduled to run.

RTOS - INTRODUCTION

RTOS - CHARACTERISTICS



RTOS - INTRODUCTION

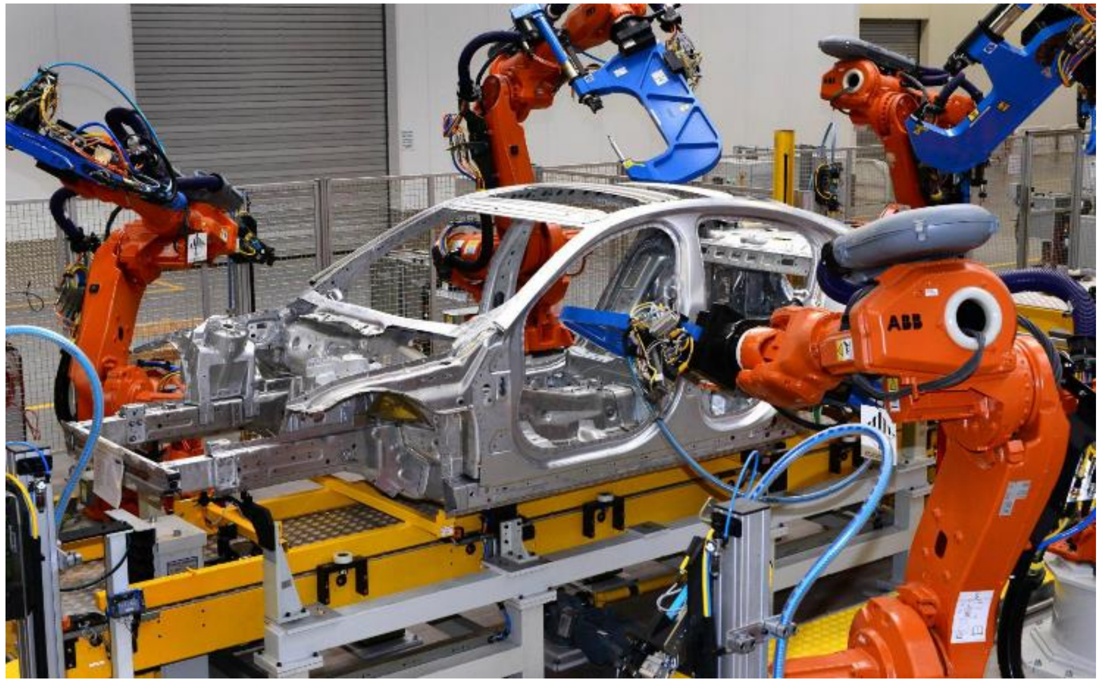
RTOS - TYPES

- HARD RTOS strictly adhere to the deadline associated with the tasks and the degree of tolerance for missed deadlines is negligible.
- A missed deadline can result in catastrophic failure of the system
- Examples:
  - Missile Navigation Systems
  - Vehicle Air Bags Control
  - Nuclear Power Plant Control

RTOS - INTRODUCTION

RTOS - TYPES

- FIRM RTOS tolerates a low occurrence of missing a deadline.
- Missing a deadline may result in an unacceptable reduction in quality of a product not lead to failure of the complete system.
- Examples:
  - Robot in car assembly section
  - Food processing control system
  - Weather monitoring system



RTOS - INTRODUCTION

RTOS - TYPES

- SOFT RTOS allows for frequently missed deadlines, and as long as tasks are timely executed their results continue to have value.
- Even the soft real time systems cannot miss the deadline for every task or process according to the priority it should meet the deadline. (Best effort system)
- Examples:
  - Multimedia transmission & reception
  - Digital cameras & mobile phones
  - Computer games

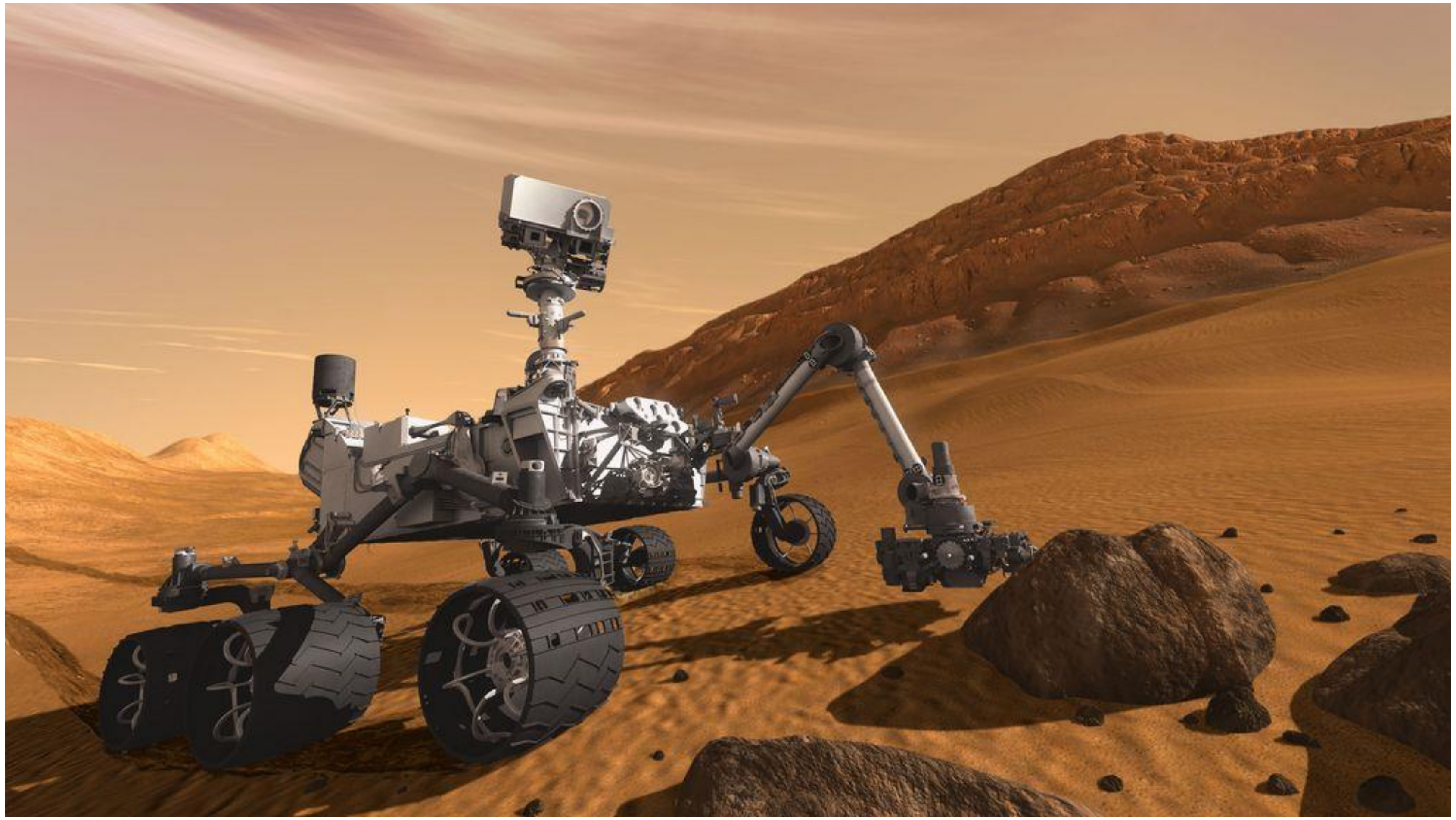
RTOS - INTRODUCTION

RTOS - APPLICATIONS

MILITARY/ AEROSPACE
SECURE CLIENTS
AVIONICS
NETWORK SECURITY
MEDICAL
EMBEDDED DEVICES

RTOS - INTRODUCTION

RTOS - APPLICATIONS



Mars Curiosity Rover, Uses ***VxWorks*** (For Split-second decision taking) and ***μc/os-II*** (For Sample Analysis)

RTOS - INTRODUCTION

RTOS - APPLICATIONS

F-35 Fighter aircraft uses a ***Integrity*** DO-178B, a POSIX based RTOS developed by Green Hills Software

RTOS - INTRODUCTION

RTOS - APPLICATIONS

International medical technology group Elekta is basing its new generations of equipment on the ***LynxOS-SE*** (RTOS)

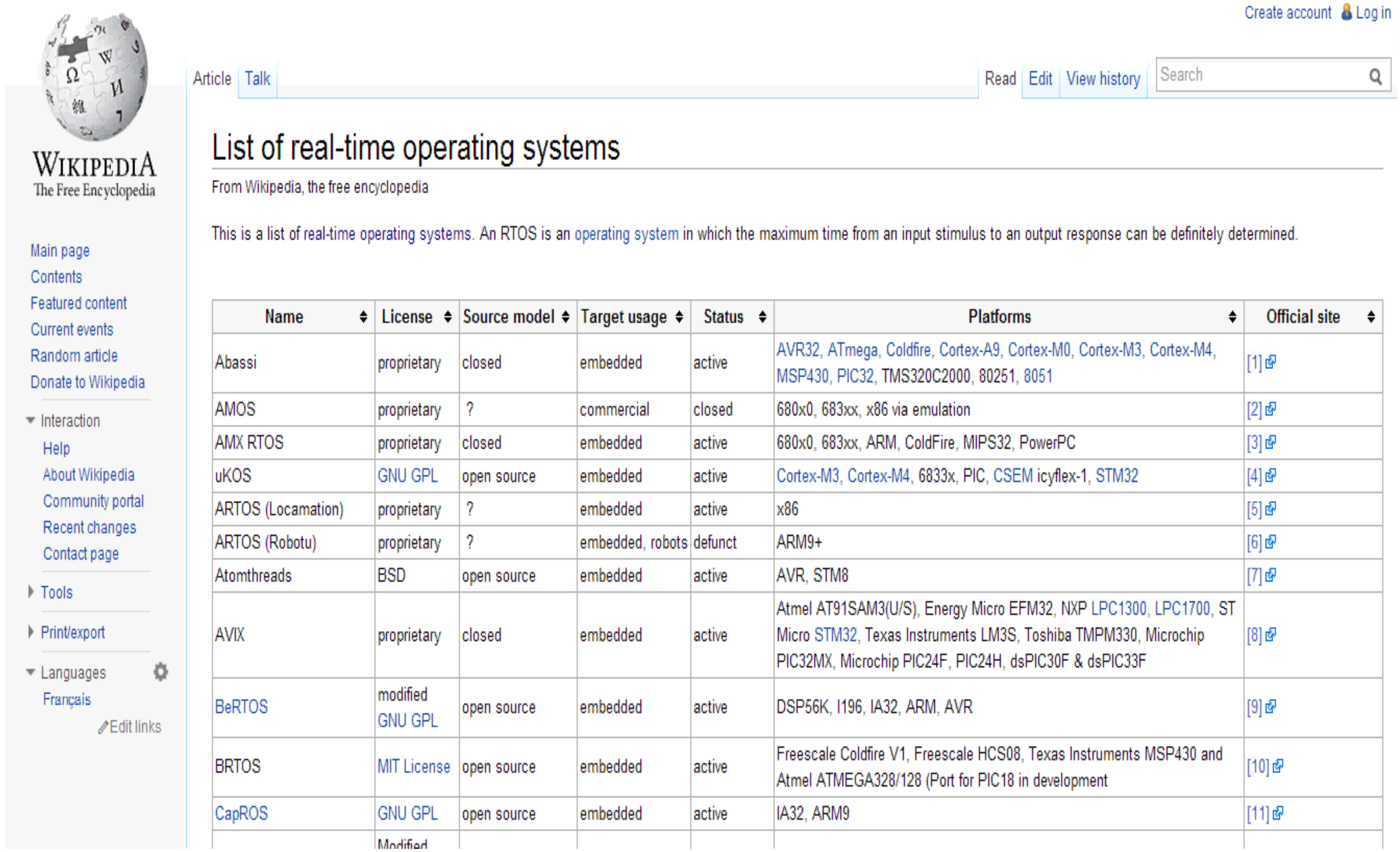
RTOS - INTRODUCTION

RTOS - APPLICATIONS

ThreadX RTOS is used by HP (All Printers) & Honeywell (Advanced security systems) in consumer electronics devices

RTOS - INTRODUCTION

RTOS IN EMBEDDED MARKET



List of real-time operating systems

From Wikipedia, the free encyclopedia

This is a list of real-time operating systems. An RTOS is an operating system in which the maximum time from an input stimulus to an output response can be definitely determined.

Name	License	Source model	Target usage	Status	Platforms	Official site
Abassi	proprietary	closed	embedded	active	AVR32, ATmega, Coldfire, Cortex-A9, Cortex-M0, Cortex-M3, Cortex-M4, MSP430, PIC32, TMS320C2000, 80251, 8051	[1]
AMOS	proprietary	?	commercial	closed	680x0, 683xx, x86 via emulation	[2]
AMX RTOS	proprietary	closed	embedded	active	680x0, 683xx, ARM, ColdFire, MIPS32, PowerPC	[3]
uKOS	GNU GPL	open source	embedded	active	Cortex-M3, Cortex-M4, 6833x, PIC, CSEM icyflex-1, STM32	[4]
ARTOS (Locamation)	proprietary	?	embedded	active	x86	[5]
ARTOS (Robotu)	proprietary	?	embedded, robots	defunct	ARM9+	[6]
Atomthreads	BSD	open source	embedded	active	AVR, STM8	[7]
AVIX	proprietary	closed	embedded	active	Atmel AT91SAM3(U/S), Energy Micro EFM32, NXP LPC1300, LPC1700, ST Micro STM32, Texas Instruments LM3S, Toshiba TMPM330, Microchip PIC32MX, Microchip PIC24F, PIC24H, dsPIC30F & dsPIC33F	[8]
BeRTOS	modified GNU GPL	open source	embedded	active	DSP56K, I196, IA32, ARM, AVR	[9]
BRTOS	MIT License	open source	embedded	active	Freescale Coldfire V1, Freescale HCS08, Texas Instruments MSP430 and Atmel ATMEGA328/128 (Port for PIC18 in development	[10]
CapROS	GNU GPL	open source	embedded	active	IA32, ARM9	[11]
	Modified					

There are more than 100 RTOS currently available in embedded market

RTOS - INTRODUCTION

MOST COMMONLY USED RTOS

freeRTOS

VxWorks® WIND RIVER

LynxOS FROM LYNUXWORKS

µC/OS-III® The Real-Time Kernel

RTLinux

INTEGRITY

Microsoft® Windows CE

eCOS

NUCLEUS

µClinux

QNX

THREADX®

RTOS - INTRODUCTION

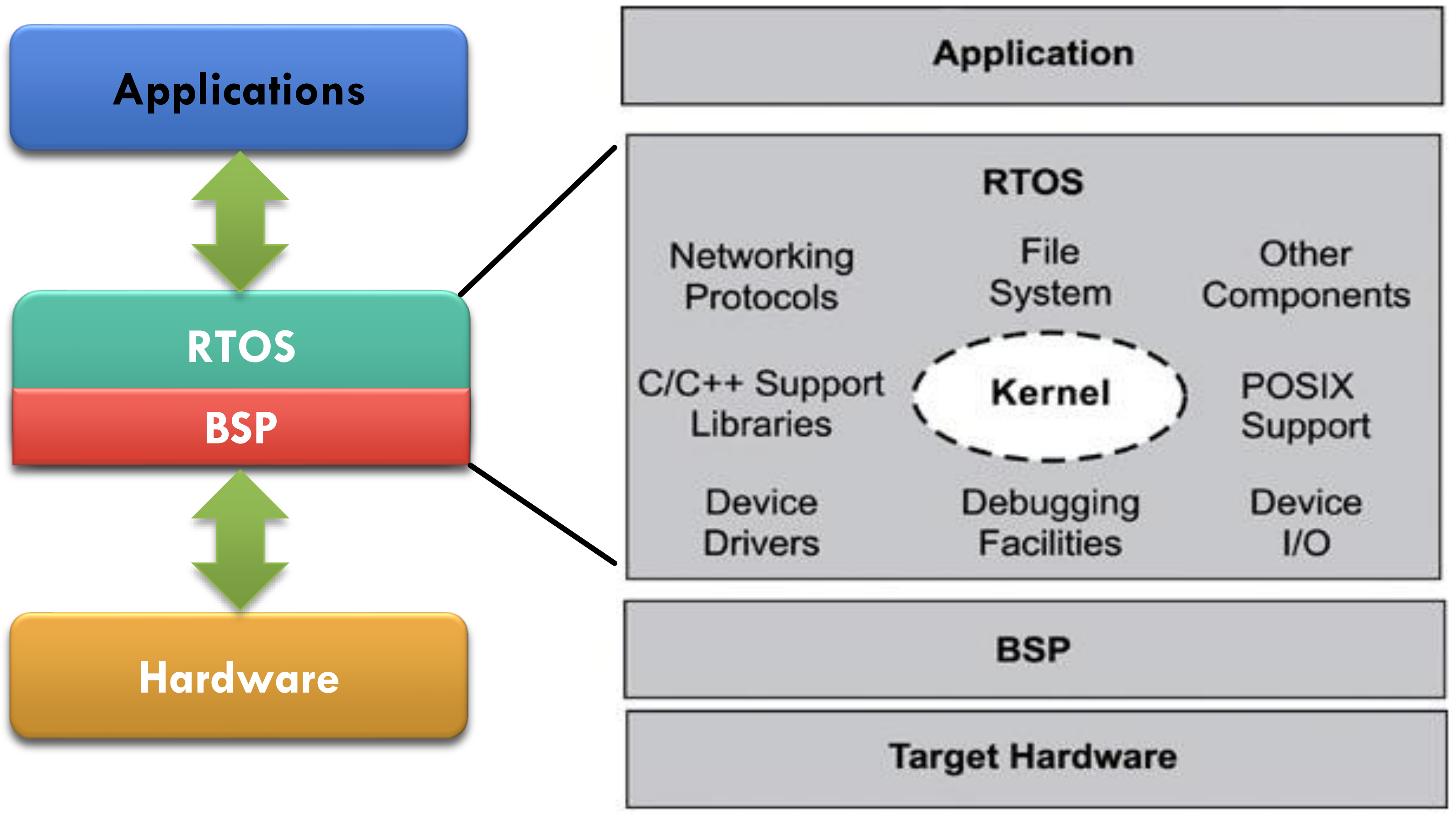
RTOS IN IOT ERA

- The Internet of Things (IoT) is accelerating the use of third-party software stacks, especially the TCP/IP and USB stacks and these third-party stacks and tools are often compatible with various RTOS solutions.
- Moreover, popular RTOS solutions like Azure RTOS, Amazon FreeRTOS, and Segger's embOS are being qualified for MCUs from large suppliers.
- This out-of-box integration is crucial in managing software complexity and lowering the barriers to entry for smaller embedded design outfits.
- However, the use of RTOS also increases the overall system complexity and may introduce new types of bugs. So, developers must have adequate knowledge of how to implement RTOS-based programs effectively.

RTOS - INTERNALS

RTOS - ARCHITECTURE

RTOS – ARCHITECTURE



RTOS - ARCHITECTURE

RTOS – KERNEL

- Kernel is the smallest and core component of an operating system.
- Its services include managing memory and devices and also to provide an interface for software applications to use the resources.
- Additional services such as managing protection of programs and multitasking may be included depending on architecture of operating system.
- There are three categories of kernel models available: Monolithic, Micro and Hybrid kernel

RTOS - ARCHITECTURE

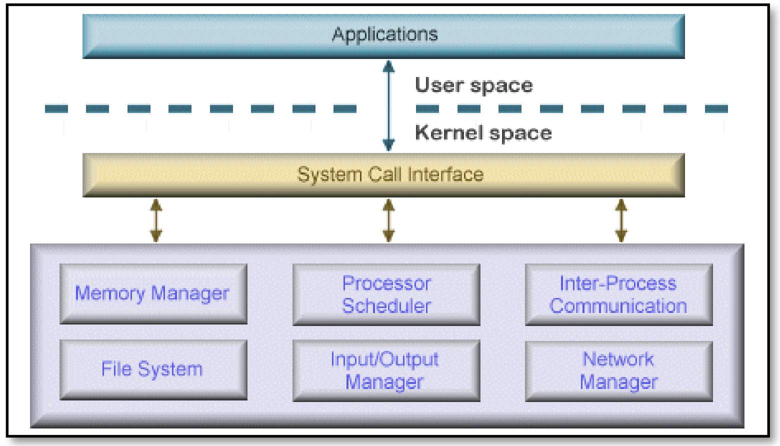
RTOS - KERNEL

- Monolithic kernel
  - Entire OS (device drivers, file system, and IPC) is working in kernel space.
  - Each component of Monolithic kernel could communicate directly with any other component, and had unrestricted system access.
- Micro kernel
  - Includes very small number of services within the kernel in an attempt to keep it small and scalable.
  - The services typically include low-level memory management, inter-process communication (IPC), basic process synchronization to enable processes to cooperate.
- Hybrid kernel
  - It has a structure similar to microkernels, but implemented as a monolithic kernel.
  - Hybrid kernels have the ability to pick and choose what they want to run in user mode and what they want to run in kernel mode.

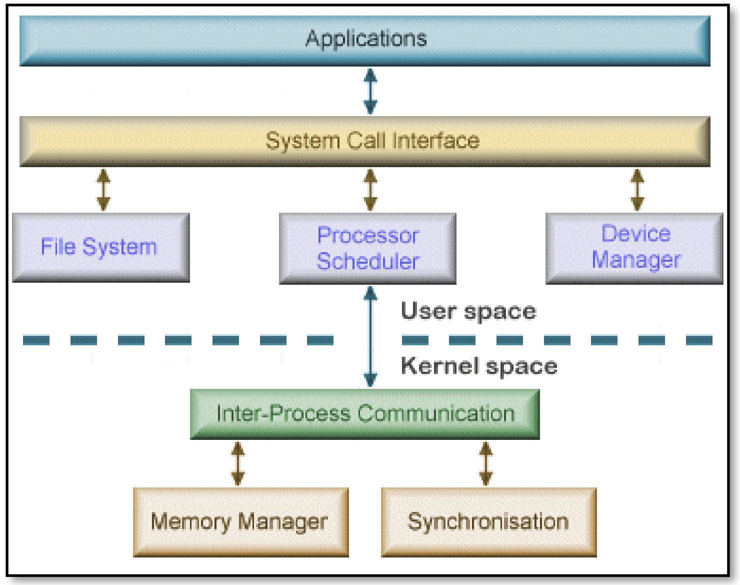
RTOS - ARCHITECTURE

EOS - KERNEL

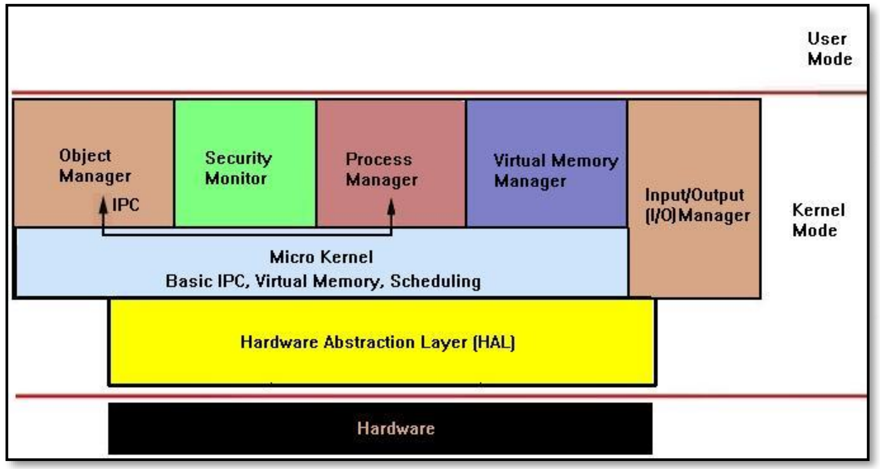
Monolithic Kernel



Microlithic Kernel



Hybrid Kernel



RTOS - ARCHITECTURE

MONOLITHIC VS MICRO VS HYBRID KERNEL

BASIS FOR COMPARISON	MONOLITHIC KERNEL	MICROKERNEL	HYBRID KERNEL
Basic	In monolithic kernel, both user services and kernel services are kept in the same address space.	In microkernel user services and kernel, services are kept in separate address space.	Hybrid kernel is a kernel combining aspects from both Monolithic and Microkernel designs
Size	larger than microkernel	Microkernel are smaller in size.	smaller than monolithic kernel
Execution	Fast execution.	Slow execution.	Moderate speed
Extendible	The monolithic kernel is hard to extend.	The microkernel is easily extendible.	Dynamically loadable module
Security	If a service crashes, the whole system crashes in monolithic kernel.	If a service crashes, it does effect on working of microkernel.	It depends on service called
Code	To write a monolithic kernel, more code is required.	To write a microkernel, less code is required.	Moderate code size
Example	Linux, BSDs, Windows (95,98,Me), Solaris, OS-9, AIX, HP-UX, DOS.	QNX, Symbian, L4Linux, Singularity, K42, Integrity, PikeOS,.	Windows NT and above, Mac OS X, BeOS, ReactOS, Syllable.

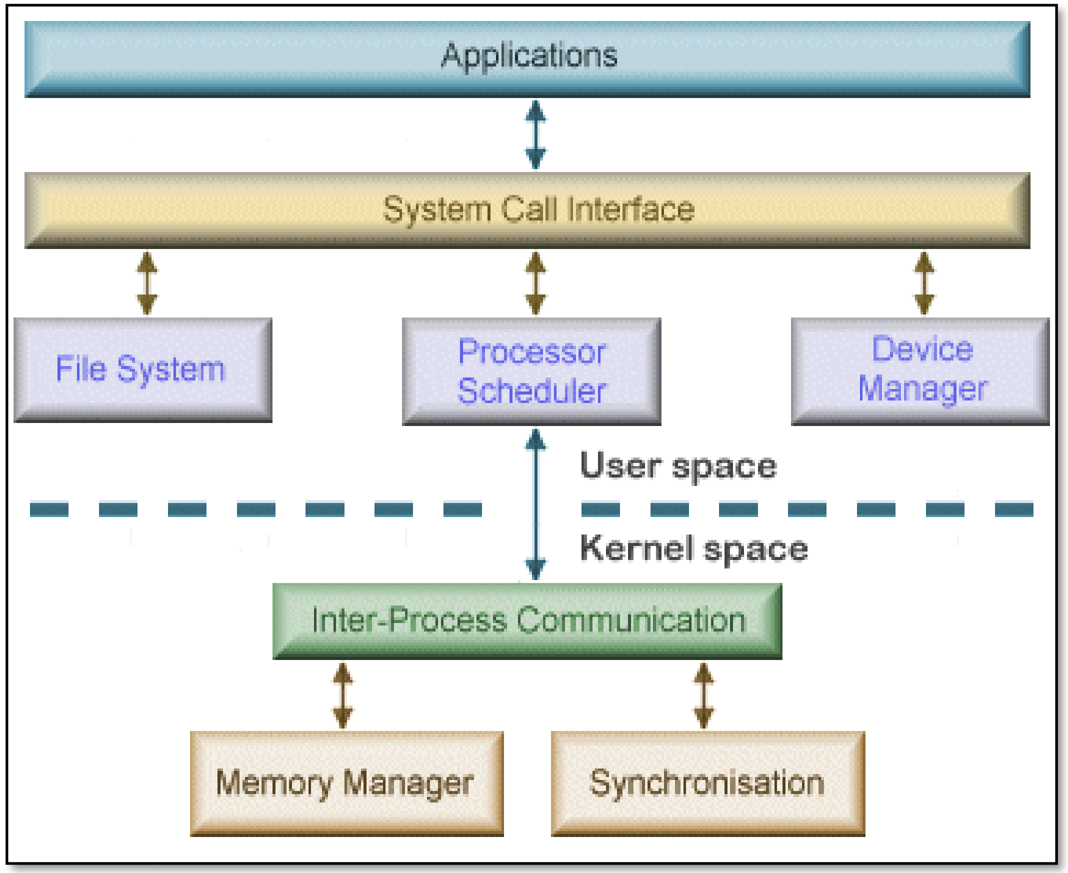
RTOS - ARCHITECTURE

KERNEL

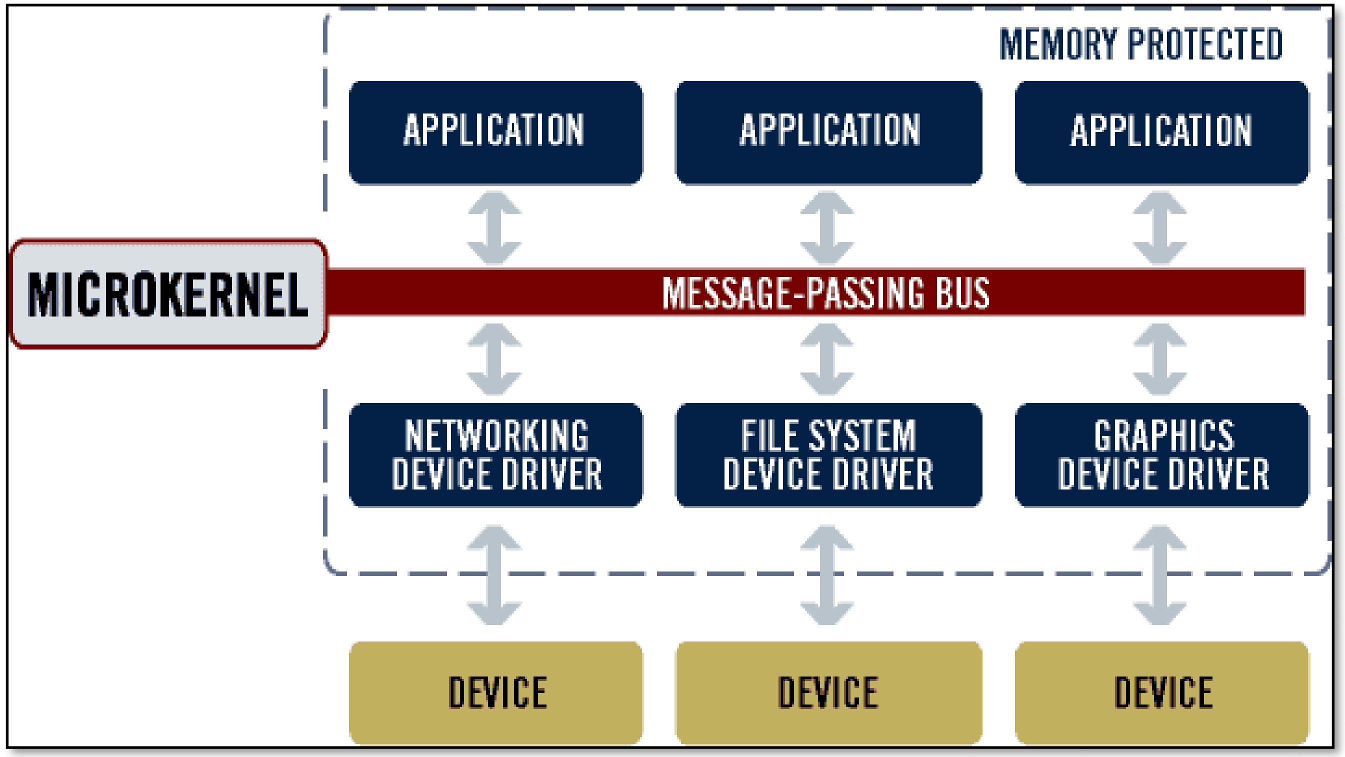
- An RTOS generally avoids implementing the kernel as a monolithic program.
- The kernel is developed instead as a micro-kernel with added configurable functionalities like scheduling, Managing memory protection and IPC.
- All other basic services can be made part of user space and can be run in the form of servers.
- This implementation gives resulting benefit in increase system configurability, as each embedded application requires a specific set of system services with respect to its characteristics.
- QNX, VxWorks OS follows the Microkernel approach

RTOS - ARCHITECTURE

RTOS – ARCHITECTURE



Microkernel Model



QNX Microkernel

RTOS - ARCHITECTURE

BSP

- In embedded systems, a board support package (BSP) is the layer of software containing hardware-specific drivers and other routines

- This allow a particular operating system (traditionally a RTOS) to function in a particular hardware environment integrated with the RTOS itself.

- Third-party hardware developers who wish to support a particular RTOS must create a BSP that allows that RTOS to run on their platform.

- BSPs are typically customizable, allowing the user to specify which drivers and routines should be included in the build based on their selection of hardware and software options.

RTOS - ARCHITECTURE

BSP

- For instance, a particular single-board computer might be paired with any of several graphics cards; in that case the BSP might include a driver for each.
- While building the BSP image the user would specify which graphics driver to include based in his choice of hardware.
- BSP is supposed to perform the following operations
  - Initialize the processor
  - Initialize the bus
  - Initialize the interrupt controller
  - Initialize the clock
  - Initialize the RAM settings
  - Load and run boot loader from flash

PERFORMANCE METRICS OF RTOS

PERFORMANCE METRICS OF RTOS

RTOS PERFORMANCE METRICS

- There are three areas of interest when looking at the performance and usage characteristics of an RTOS: (1) Memory (2) Latency (3) Kernel services

- Memory:
  - How much ROM and RAM does the kernel need and how is this affected by options and configuration?
  - Factors that affect the memory footprint includes static or dynamic configuration of Kernel, number of instruction related to processor, global variable declaration etc.,
  - With the help of optimization setting in embedded compilers code size can be reduced, but that will most likely affect performance.
  - Most RTOS kernels are scalable, but some RTOSes, scalability only applies to the kernel. For others, scalability is extended to the rest of the middleware.

PERFORMANCE METRICS OF RTOS

RTOS PERFORMANCE METRICS

- Latency:
  - Interrupt latency: It is the sum of the hardware dependent time, which depends on the interrupt controller as well as the type of the interrupt, and the OS induced overhead.
  - Ideally, it should include the best and worst case scenarios.
  - To measure a time interval, like interrupt latency, with any accuracy, requires a suitable instrument and the best tool to use is an oscilloscope.
  - Scheduling latency: Being real time, the efficiency at which threads or tasks are scheduled is of some importance and the scheduler is at the core of an RTOS.
  - It is hard to get a clear picture of performance, as there is a wide variation in the techniques employed to make measurements and in the interpretation of the results.
  - There are really two separate measurements to consider: (1) The context switch time (2) The time overhead that the RTOS introduces when scheduling a task

PERFORMANCE METRICS OF RTOS

RTOS PERFORMANCE METRICS

- Performance of kernel services:
  - An RTOS is likely to have many API (application program interface) calls, probably numbering into the hundreds.
  - To assess timing, it is not useful to try to analyse the timing of every single call. It makes more sense to focus on the frequently used services.
  - How long does it take to perform specific actions?
  - For most RTOSes, there are four key categories of service call:
    - Threading services
    - Synchronization services
    - Inter-process communication services
    - Memory services

PERFORMANCE METRICS OF RTOS

RTOS CONSIDERATIONS

- What are the real-time capabilities? Is it soft or hard real-time interrupt handling
- What are the preemptive scheduling services?
- What is the target processor, and does it support the RTOS you have in mind?
- How large is the RTOS memory footprint?
- What are the RTOS interrupt latencies?
- How long does it take the RTOS to switch contexts?
- Does your processor development board have a BSP with your RTOS?

PERFORMANCE METRICS OF RTOS

RTOS CONSIDERATIONS

- Which development environments and debugging tools work with the RTOS?
- How much does the RTOS cost, is it open source or royalty-free?
- Does the RTOS have good documentation and/or forums?
- What is the RTOS supplier's reputation?
- How flexible is the choice of scheduling algorithms in the RTOS? E.g., FIFO, round Robin, rate-monotonic, sporadic, etc.
- Are there tools for remote diagnostics?

TASK SPECIFICATIONS

TASK SPECIFICATIONS

REAL-TIME TASK

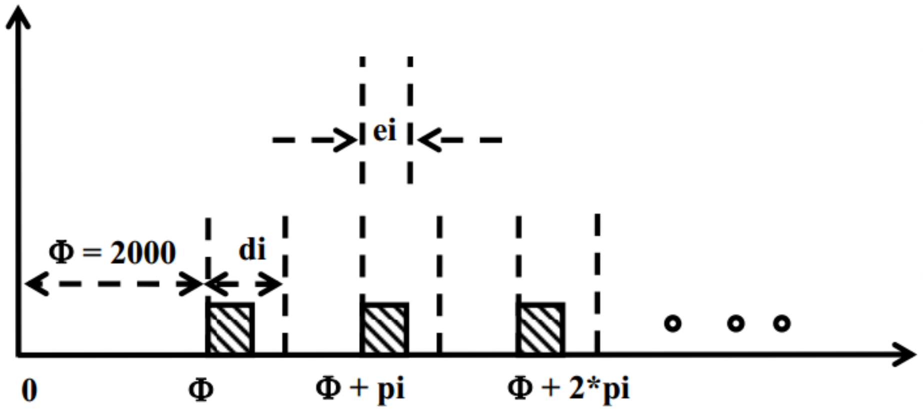
- Real time tasks normally recur a large number of times at different instants of time depending on event occurrence time.
- Ex.: A temperature sensing task in chemical plant might recur indefinitely with a certain period because the temperature is sampled periodically, whereas a task handling a device interrupt might recur at random instants.
- A real-time task is defined to be one in which there is a hard deadline to meet; failure to meet the deadline can lead to disaster, Ex: task include process control such as a nuclear reactor, automatic flight control, etc.
- Based on the way real-time tasks recur over a period of time, it is possible to classify them into (1) Periodic tasks (2) Sporadic tasks (3) Aperiodic tasks

TASK SPECIFICATIONS

REAL-TIME TASK - PERIODIC

- A periodic task is one that repeats after a certain fixed time interval also referred as clock-driven tasks.
- A vast majority of the tasks present in a typical real-time system are periodic, for example, monitoring certain conditions, polling information from sensors at regular intervals to carry out certain action at regular intervals.
- Periodic task Ti can be represented using four tuples: (Øi, pi, ei, di)

  Where,
  - Øi - phase of the task
  - pi - Period of task
  - ei - Worst case execution time of the task
  - di - Relative deadline of the task



TASK SPECIFICATIONS

REAL-TIME TASK - SPORADIC

- A sporadic task is one that recurs at random instants.
- E.g. In a robot a task that gets generated to handle an obstacle that appears suddenly is a sporadic task. In a factory , the task that handles fire condition is a sporadic task.
- Sporadic task Ti can be represented using three tuples: (ei, gi, di)

  Where, ei - Worst case execution time of the task
  gi - the minimum separation between two consecutive instances of the task.
  di - Relative deadline of the task
- The minimum separation gi indicates that once an instance of a sporadic task occurs , the next instance cannot occur before gi time unit have elapsed.

TASK SPECIFICATIONS

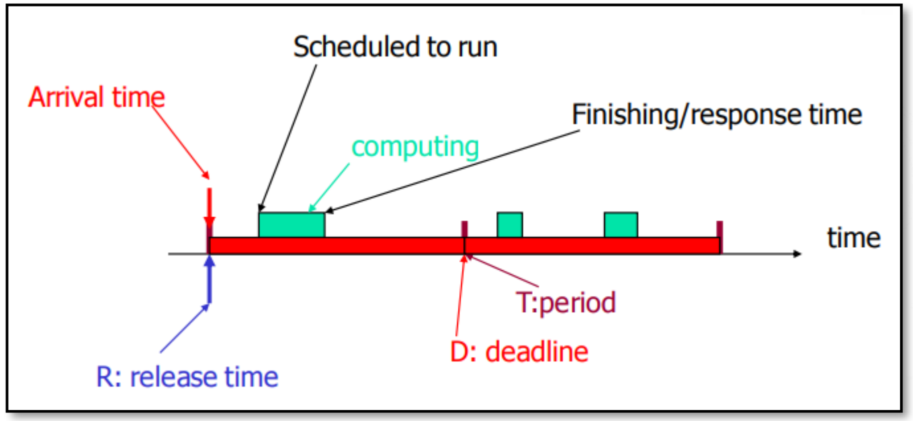
REAL-TIME TASK - APERIODIC

- An aperiodic task can arise at random instants.
- Two or more instances of an aperiodic task might occur at the same time instant because minimum separation gi, between two consecutive instances can be 0.
- Aperiodic task can recur in quick succession, therefore it is very difficult to meet the deadlines of all instances of an aperiodic task.
- Soft real time systems can tolerate a few deadline misses. so, aperiodic tasks are generally soft real time tasks.
- E.g. Logging task in a distributed systems. The logging task can be started by different tasks running on different nodes.

TASK SPECIFICATIONS

REAL-TIME TASK - ASSUMPTIONS

- A1: Tasks are periodic i.e activated at a constant rate
- A2: All instance of a periodic task (Ti) have same computation time (Ci)
- A3: All tasks have the same deadline which is equal to the period (Di=Pi)
- A4: All task are independent i.e no dependency on other task or any resources
- A5: All task are pre-emptible
- A6: No task can suspend itself
- A7: All task are released as soon as they arrive
- A8: All overhead in the kernel is assumed to be zero



SCHEDULABILITY ANALYSIS

SCHEDULABILITY ANALYSIS

SCHEDULING CRITERIA

1. CPU utilization: CPU should be working most of the time (Ideally 100% all the time)
2. Throughput: total number of processes completed per unit time(10 tasks/second)
3. Turnaround time (TAT): amount of time taken to execute a particular process, TATi=CTi–ATi (Where CTi → Completion Time, ATi → Arrival Time)
4. Waiting time(WT): time periods spent waiting in the ready queue by a process to acquire get control on the CPU, WTi = TATi – BTi (Where BTi → CPU burst time)
5. Load average: average number of processes residing in the ready queue waiting for their turn to get into the CPU
6. Response time: Amount of time it takes from when a request was submitted until the first response is produced

SCHEDULING OBJECTIVE: **Max →CPU utilization, Throughput**
Min → Turnaround time, Waiting time, Load average, Response time

SCHEDULABILITY ANALYSIS

TASK MODEL

- A task = (C, P)
  - C: worst case execution time/computing time (C<=P!)
  - P: period (D=P)
  - C/P is CPU utilization of a task

U>1 (overload): some task will fail to meet its deadline
U<=1 : it will depend on the scheduling algorithms
U=1 : CPU is kept busy, all deadlines will be met

- A task set: (Ci,Pi)
  - All tasks are independent
  - The periods of tasks start at 0 simultaneously
  - U=Σ(Ci/Pi) is CPU utilization of a task set

- CPU utilization is a measure on how busy the processor could be during the shortest repeating cycle: P1*P2*...*Pn

SCHEDULABILITY ANALYSIS

SCHEDULABILITY TEST

- Schedulability test determine whether a given task set is feasible to schedule?
- Necessary test:
  - If test is passed, tasks may be schedulable but not necessarily
  - If test is not passed, tasks are definitely not schedulable
- Sufficient test:
  - If test is passed, then task are definitely schedulable
  - If test is not passed, tasks may be schedulable but not necessarily
- Exact test: (Necessary test + Sufficient test)
  - The task set is schedulable if and only if it passes the test

SCHEDULABILITY ANALYSIS

SCHEDULABILITY TEST

- Necessary test

$$U = \sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1$$

- Sufficient test (Utilization Bound test)

$$U = \sum_{i=1}^{N} \left(\frac{C_i}{PT_i}\right) \leq N(2^{\frac{1}{N}} - 1)$$

N	B(N)
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
∞	0.693

REAL TIME SCHEDULING

REAL TIME SCHEDULING

- Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running on the system.
- Real-time scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU in RTOS.
- By switching the CPU among processes, the operating system can make the embedded system is more productive.
- Scheduling algorithm is the method by which threads, processes or data flows are given access to system resources
- The need for a scheduling algorithm arises from requirement for most modern systems to perform multitasking.

REAL TIME SCHEDULING

REAL-TIME SCHEDULING ALGORITHM

- The purpose of a real-time scheduling algorithm is to ensure that critical timing constraints, such as deadlines and response time, are met.
- Real-time systems use preemptive multitasking with priorities are assigned to tasks, and the RTOS always executes the task with highest priority.
- Most algorithms are classified as
  - Fixed-priority: Algorithm assigns priorities at design time, and those priorities remain constant for the lifetime of the task. Ex.: Rate-Monotonic Scheduling (RMS)
  - Dynamic-priority: Assigns priorities dynamically at runtime, based on execution parameters of tasks, such as upcoming deadlines. Ex.: Earliest Deadline First (EDF)
  - Mixed-priority: This algorithm has both static and dynamic components.

REAL TIME SCHEDULING

TYPES OF SCHEDULING ALGORITHM

- Non pre-emptive Algorithm:
  - First come First served (FCFS)
  - Priority(Non-pre-emptive)
  - Shortest Job First(SJF)

- Pre-emptive Algorithm:
  - Shortest remaining Time First(SRTF)
  - Round Robin(RR)
  - Priority(Pre-emptive)
  - Rate Monotonic Scheduling (RMS)
  - Earliest Deadline First (EDF)

REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS)

- Concept: Task with the shortest period executes with the highest priority.
- Working :
  - Rate-monotonic is a fixed priority based scheduling.
  - The scheduling scheme is pre-emptive; it ensures that a task is pre-empted if another task with a shorter period is expected to run.
  - Used in embedded systems where the nature of the scheduling is deterministic.
  - Consider three tasks with a period Task-1(10ms), Task-2(15ms), Task-3 (20ms), then as per RMS priority of the tasks can be assigned as:

priority (task1) > priority (task2) > priority (task3)

REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) – EXAMPLE-1

- Consider set of task running on a Automotive control system as follows
  - Speed measurement Task (T1): C=20ms, P=100ms, D=100ms
  - ABS control Task (T2): C=40ms, P=150ms, D=150ms
  - Fuel injection Task (T3): C=100ms, P=350ms, D=350ms
  - Other software with soft deadlines e.g. audio, air condition etc.,

Necessary Test

$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1}$$

U=(20/100)+(40/150)+(100/350)
= 0.2 + 0.2666 + 0.2857 = 0.7517 ≤ 1

Necessary Test is passed hence given task set must be tested under sufficient test to conclude the Schedulability. But, if necessary test is failed we may conclude given task set is definitely not schedulable by any scheduling algorithm.

Sufficient Test

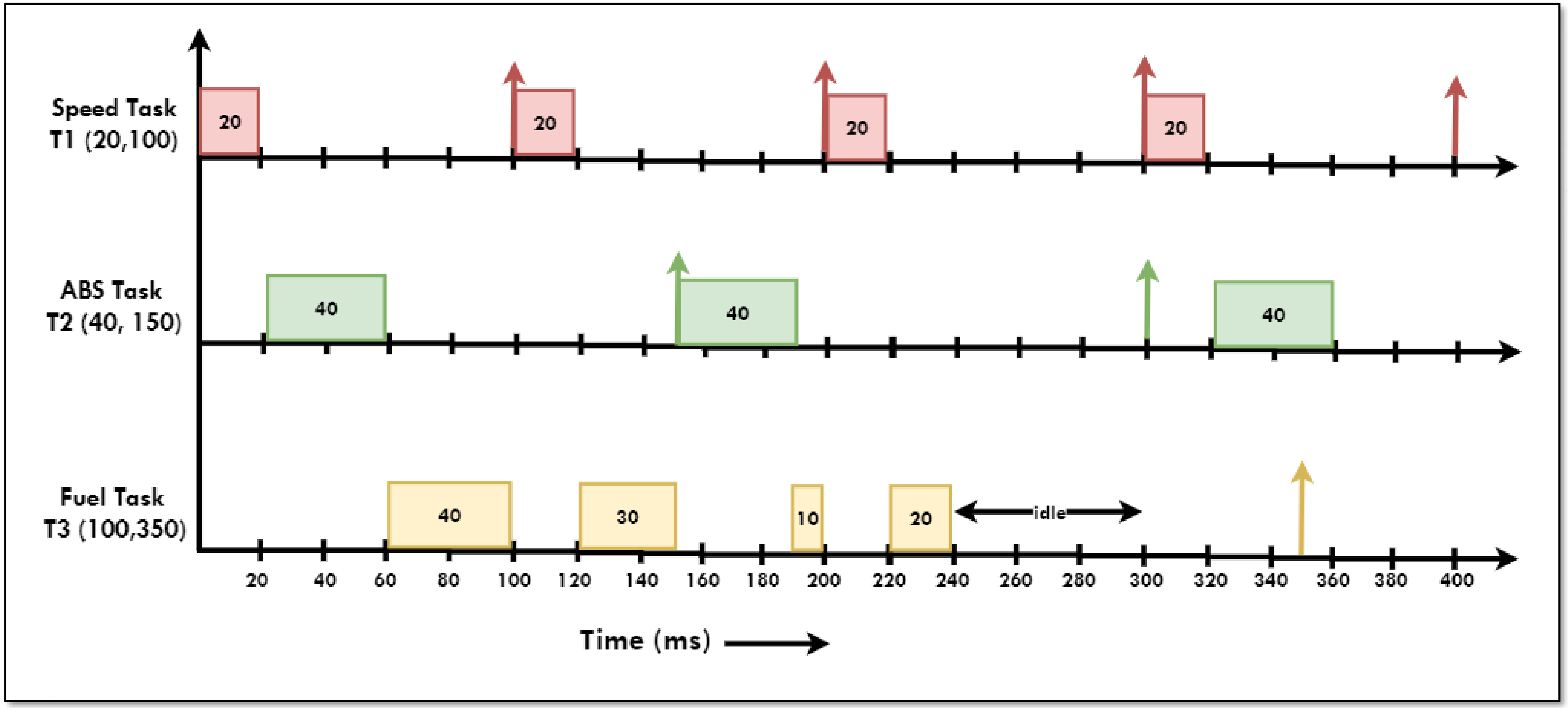
$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{PT_i}\right) \leq N(2^{\frac{1}{N}} - 1)}$$

B(3)=0.779
U=0.7517 ≤ B(3) ≤ 1

Since given task set passes Sufficient test we may conclude it is definitely schedulable under RMS

REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) – EXAMPLE-1



This is only for the first periods. But this is enough to conclude that the task set is schedulable

REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) – EXAMPLE-2

- Consider set of four task (Ci, Pi) running on embedded system as follows, T1= (1, 3) , T2= (1, 5) , T3= (1, 6) , T4= (2, 20)

Necessary Test

$$\boxed{U = \sum_{i=1}^{N}\left(\frac{C_i}{P_i}\right) \le 1}$$

U=(1/3)+(1/5)+(1/6)+(2/10)=0.899 ≤ 1

Necessary Test is passed hence given task set must be tested under sufficient test to conclude the Schedulability. But, if necessary test is failed we may conclude given task set is definitely not schedulable by any scheduling algorithm.

Sufficient Test

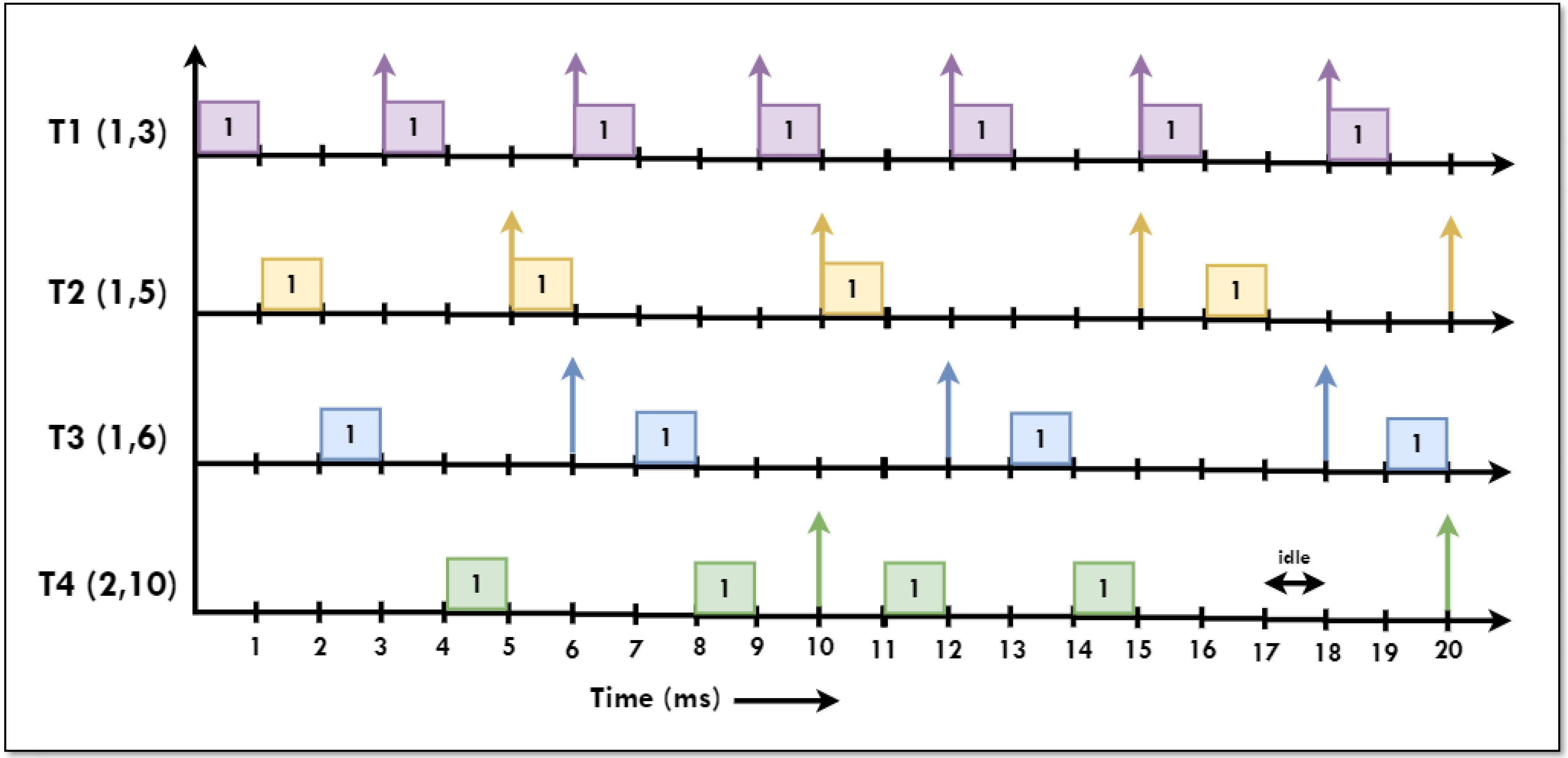
$$\boxed{U = \sum_{i=1}^{N}\left(\frac{C_i}{PT_i}\right) \le N(2^{\frac{1}{N}} - 1)}$$

B(4)=0.756

The given task set fails in the Sufficient test due to U=0.899 > B(4) we can't conclude precisely whether given task set is schedulable or not under RMS

REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) – EXAMPLE-2



REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) – EXAMPLE-3

- Consider set of three task (Ci, Pi) running on embedded system as follows, T1= (10, 30) , T2= (15, 40) , T3= (5, 50)

Necessary Test

$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1}$$

U=(10/30)+(15/40)+(5/50)=0.808 ≤ 1

Sufficient Test

$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{PT_i}\right) \leq N(2^{\frac{1}{N}} - 1)}$$

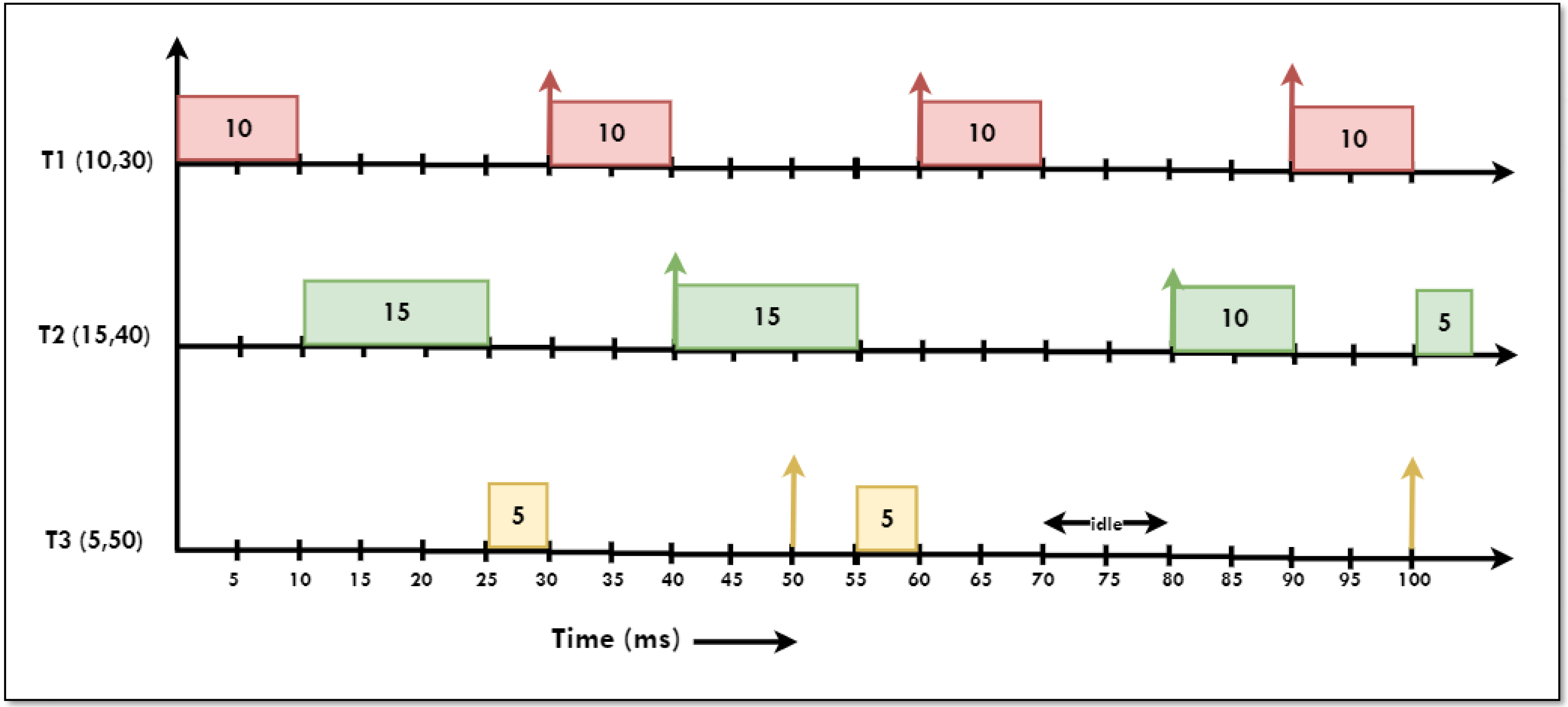
B(3)=0.779

Necessary Test is passed hence given task set must be tested under sufficient test to conclude the Schedulability. But, if necessary test is failed we may conclude given task set is definitely not schedulable by any scheduling algorithm.

The given task set fails in the Sufficient test due to U=0.808 > B(3) we can't conclude precisely whether given task set is schedulable or not under RMS

REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) – EXAMPLE-3



REAL TIME SCHEDULING

RATE MONOTONIC SCHEDULING(RMS) - PROs & CONs

- **PROs:**
  - Simple to understand
  - Easy to implement (static/fixed priority assignment)
  - Stable: though some of the lower priority tasks fail to meet deadlines, others may meet deadlines
- **CONs:**
  - Lower CPU utilization
  - Requires D=T
  - Only deal with independent tasks
  - Non-precise Schedulability analysis

REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF)

- Concept: Task closest to the end of its period assigned the highest priority
- Working :
  - Earliest deadline first is a dynamic priority based scheduling.
  - The scheduling scheme is pre-emptive; it ensures that a task is pre-empted if another task having a nearest deadline is expected to run.
  - EDF is optimal scheduling i.e it can schedule the task set if any other scheduling algorithm can schedule
  - If two task have the same deadlines, need to chose one if the two at random but in most of the case it proceed with the same task which is currently executing on the CPU to avoid context switching overhead.

REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-1

- Consider set of task running on a weather monitoring station as follows
  - Temperature measurement Task (T1): C=1ms, P=4ms
  - Humidity measurement Task (T2): C=2ms, P=5ms
  - Co2 measurement Task (T3): C=2ms, P=7ms

Necessary Test

$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1}$$

U=(1/4)+(2/5)+(2/7)

= 0.25 + 0.4+ 0.2857

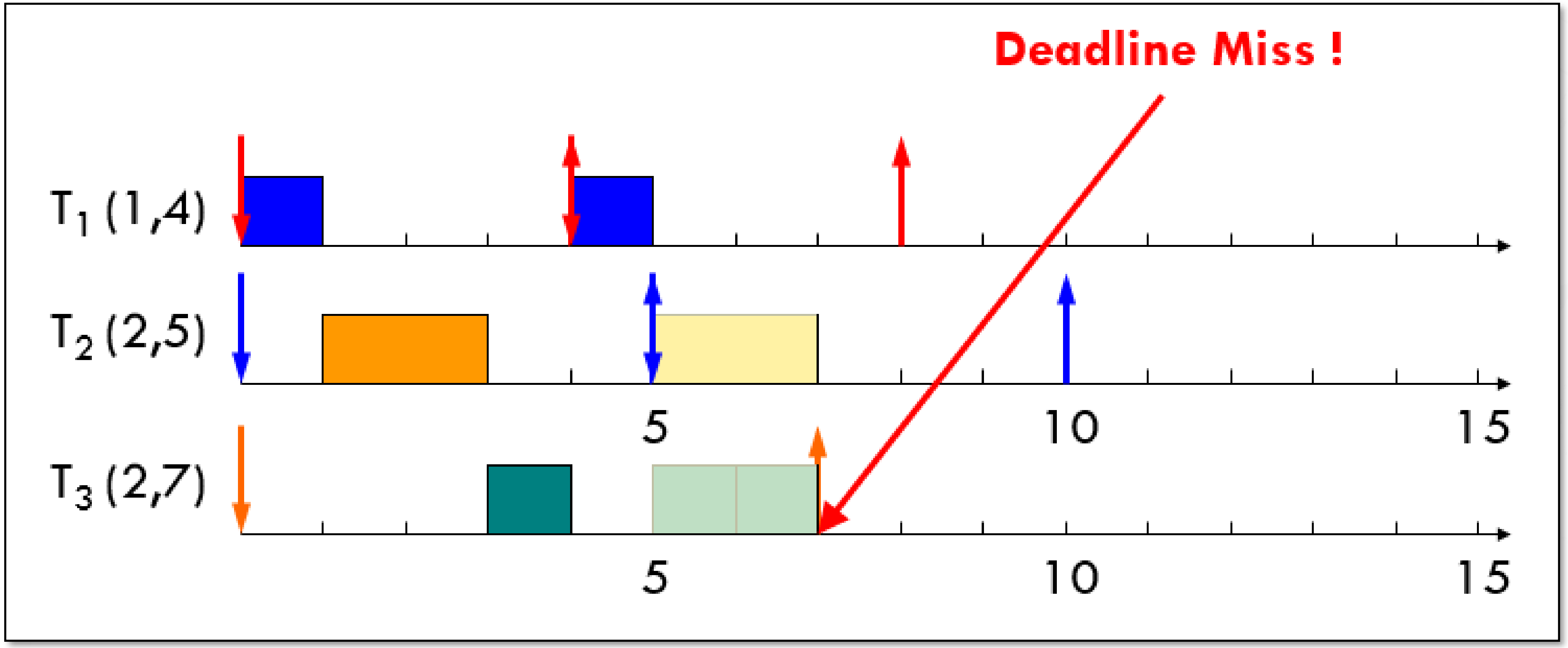
= 0.935 ≤ 1

Necessary Test is passed hence given task set must be schedulable by EDF

REAL TIME SCHEDULING

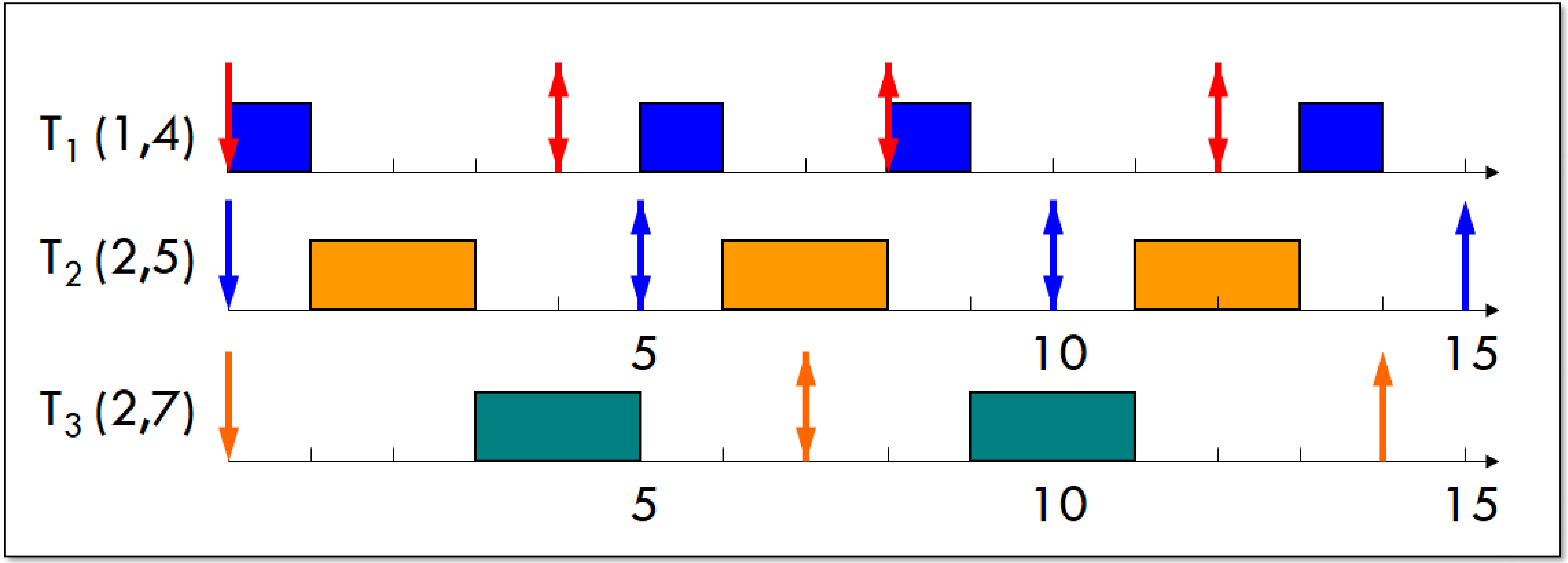
EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-1

Let's first try with RMS



REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-1



Although given task set failed to schedule under RMS, it can scheduled by EDF

REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-2

- Consider set of four task (Ci, Pi) running on embedded system as follows, T1= (1, 3) , T2= (1, 5) , T3= (1, 6) , T4= (2, 20)

Necessary Test

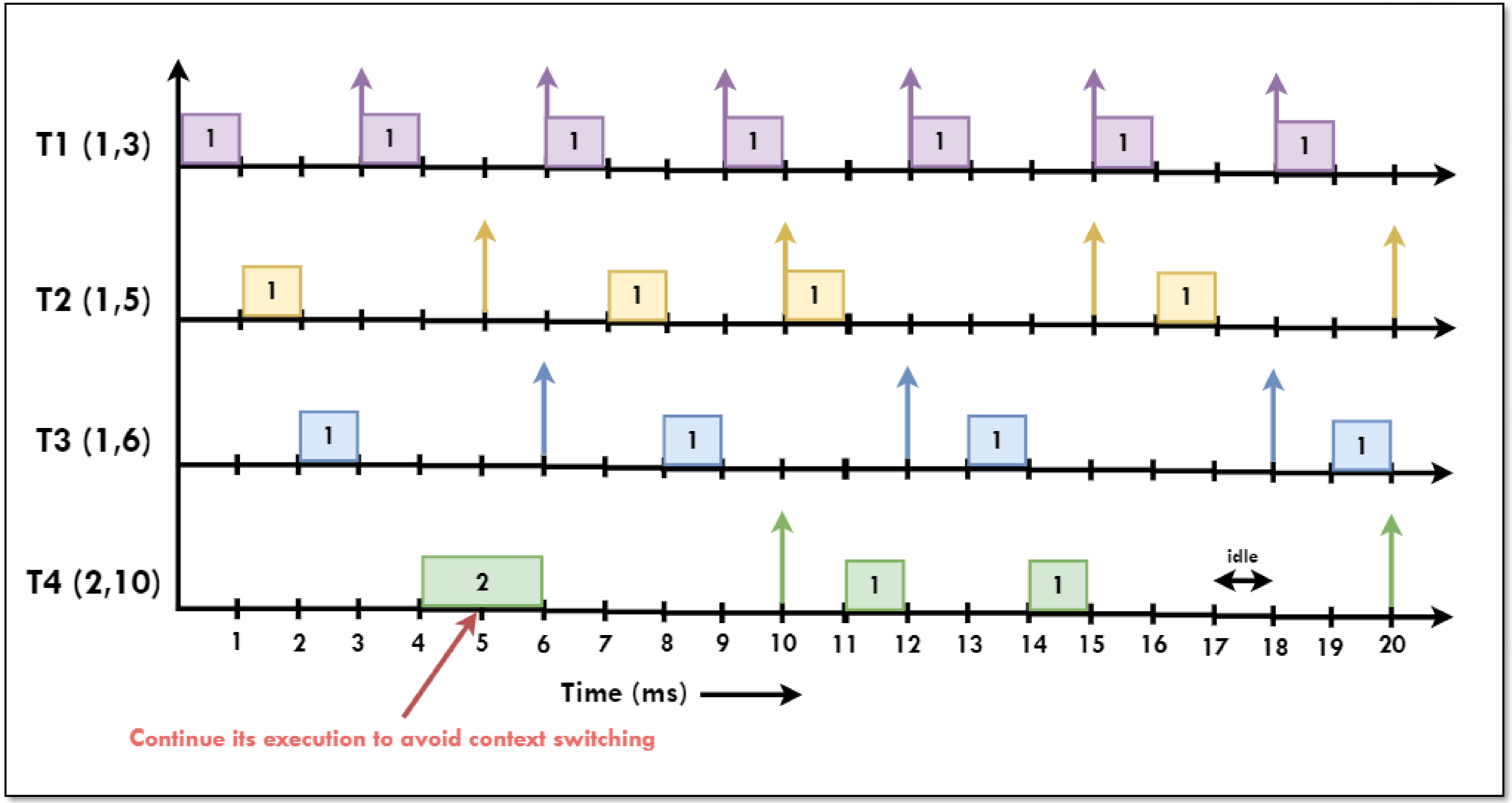
$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1}$$

U=(1/3)+(1/5)+(1/6)+(2/10)=0.899 $\leq$ 1

Necessary Test is passed hence given task set must be schedulable by EDF

REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-2



REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-3

- Consider set of three task (Ci, Pi) running on embedded system as follows, T1= (5, 20) , T2= (10, 40) , T3= (40, 80)

Necessary Test

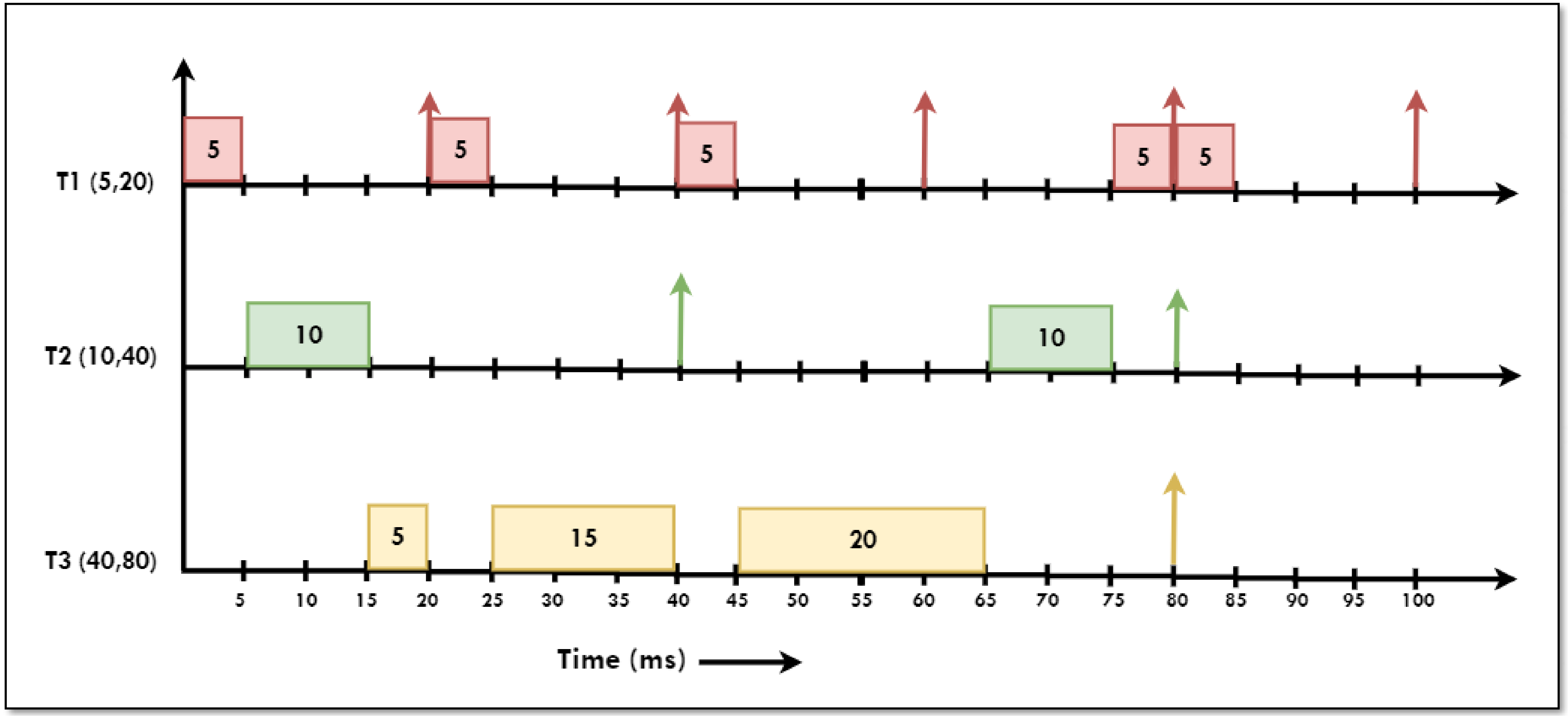
$$\boxed{U = \sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1}$$

U= (5/20)+(10/40)+(40/80)
= 0.25 + 0.25 + 0.5 = 1 ≤ 1

If cumulative CPU utilization is 1 then the CPU will be utilized 100% without idle condition

REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) – EXAMPLE-3



At time instance 80ms all 3 tasks are having same deadline. When multiple tasks having deadline at same instance of a time then one of the task will be chosen randomly and get executed.

REAL TIME SCHEDULING

EARLIEST DEADLINE FIRST (EDF) - PROs & CONs

- PROs:
  - It works for all types of tasks: periodic or non periodic
  - Simple Schedulability test
  - Best CPU utilization
  - Optimal scheduling algorithm
- CONs:
  - Difficult to implement due to the dynamic priority-assignment.
  - Any task could get the highest priority even if the task is not important
  - Non stable because if any task fails to meet its deadline, the system is not predictable