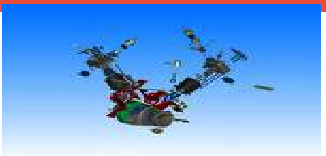
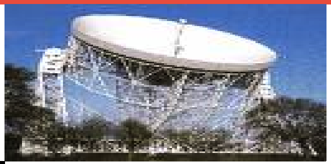
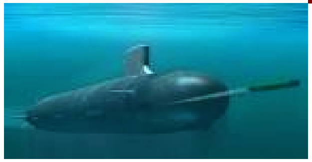
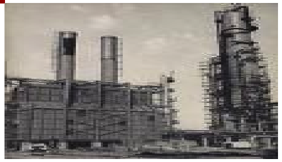





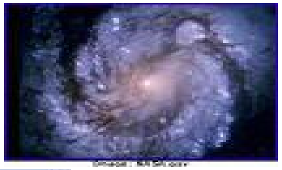
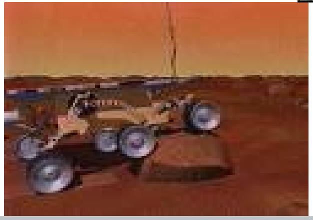




Real Time Systems & Real Time OS Concepts



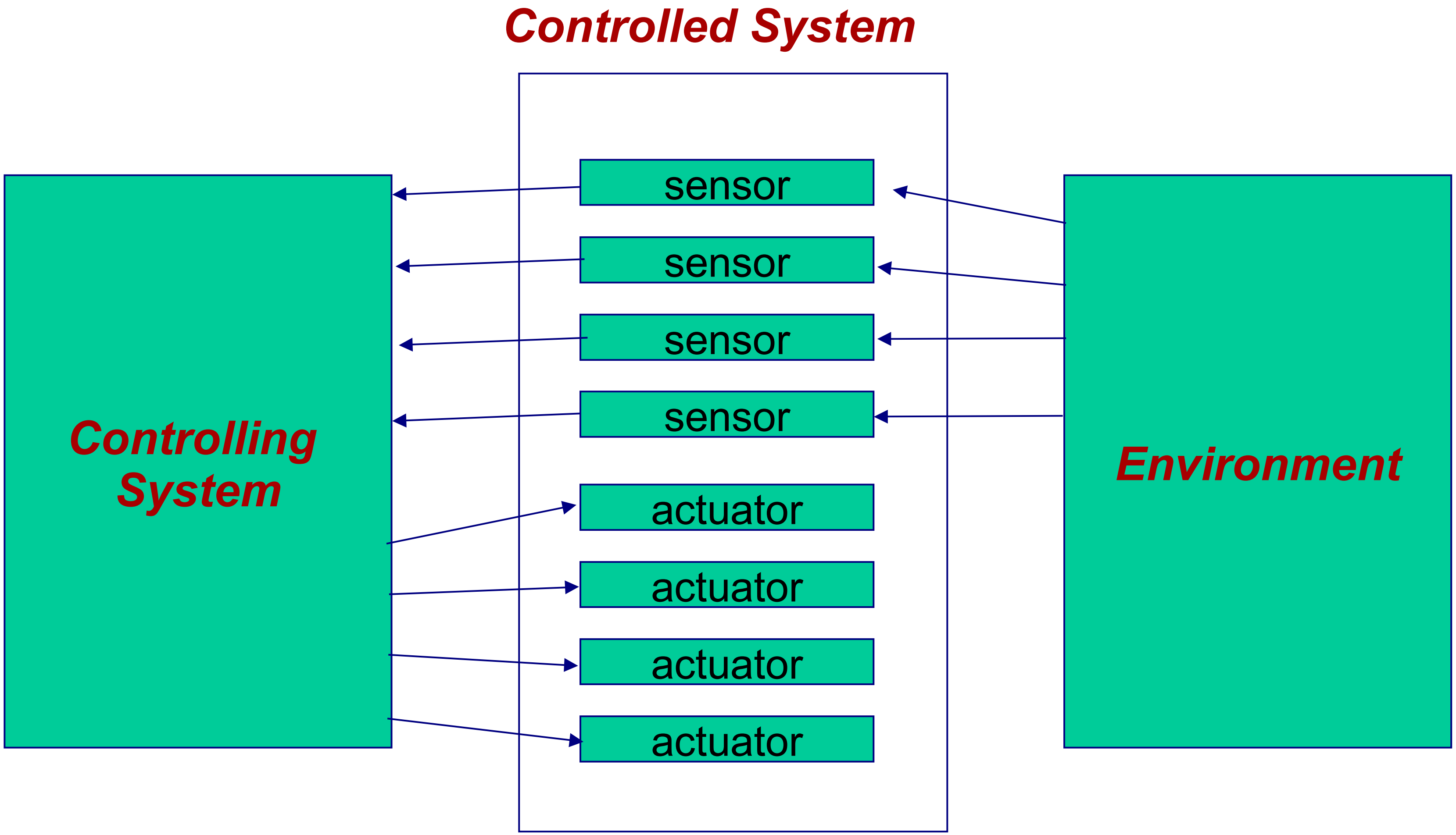
Agenda

- **Introduction to Real Time Systems (RTS)**
- **Introduction to Real Time Operating Systems (RTOS)**
  - How they differ: RTOS vs GPOS
  - RTOS Building Blocks: Tasks, Scheduler, Services
- **Overview of popular RTOS**

Real-Time System definition

- **Real-time systems have been defined as: "those systems in which the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced";**
  - The misconception that real time means fast is wrong
  - J. Stankovic, "Misconceptions About Real-Time Computing," IEEE Computer, 21(10), October 1988.

Typical Real-Time System



Real Time System Characteristics

- **Correctness**
  - The correctness of the system depends not only on the logical result, but also on the time at which the results are produced.
- **Predictability**
  - Possible to show at “design time” that all the timing constraints of the application will be met.
- **Deterministic**
  - For each possible state and each set of inputs, a unique set of outputs and next state of the system can be determined.

Real Time Tasks

- **Periodic tasks**
  - Time-driven (pi, ci)
  - Eg: Task monitoring temperature of a patient in an ICU
- **Sporadic Tasks**
  - Event-driven (ci, gi, di)
  - Minimum separation (gi) between two consecutive instances of the task implies that once an instance of a sporadic task occurs, the next instance cannot occur before gi.
  - Hard deadline. Eg: Reporting of fire conditions
- **Aperiodic tasks**
  - Similar to Sporadic tasks
  - Minimum separation between two consecutive instances can be 0
  - Soft deadline Eg: Interactive commands issued by users

ci : worst case exec time, pi : task period, di : deadline, gi : minimum separation time

Classification of Real Time Systems

- **Soft real-time system**
  - Performance is degraded but not destroyed by failure to meet response-time constraints.
  - Eg: Mutlimedia, Interactive video games, Online transaction systems
- **Hard real-time system**
  - Failure to meet a single deadline may lead to catastrophic failure
  - Eg: Aircraft Control Systems, Nuclear Power Stations, Chemical Plants, Life support systems
- **Synchronous real-time system**
  - Periodic (Time driven) – task activity occurs repeatedly
- **Asynchronous real-time system**
  - Aperiodic (Event driven) – task activities are asynchronous
- **Combination of both – Some tasks are periodic others, aperiodic**

Real Time Operating Systems

Goals of an Operating System

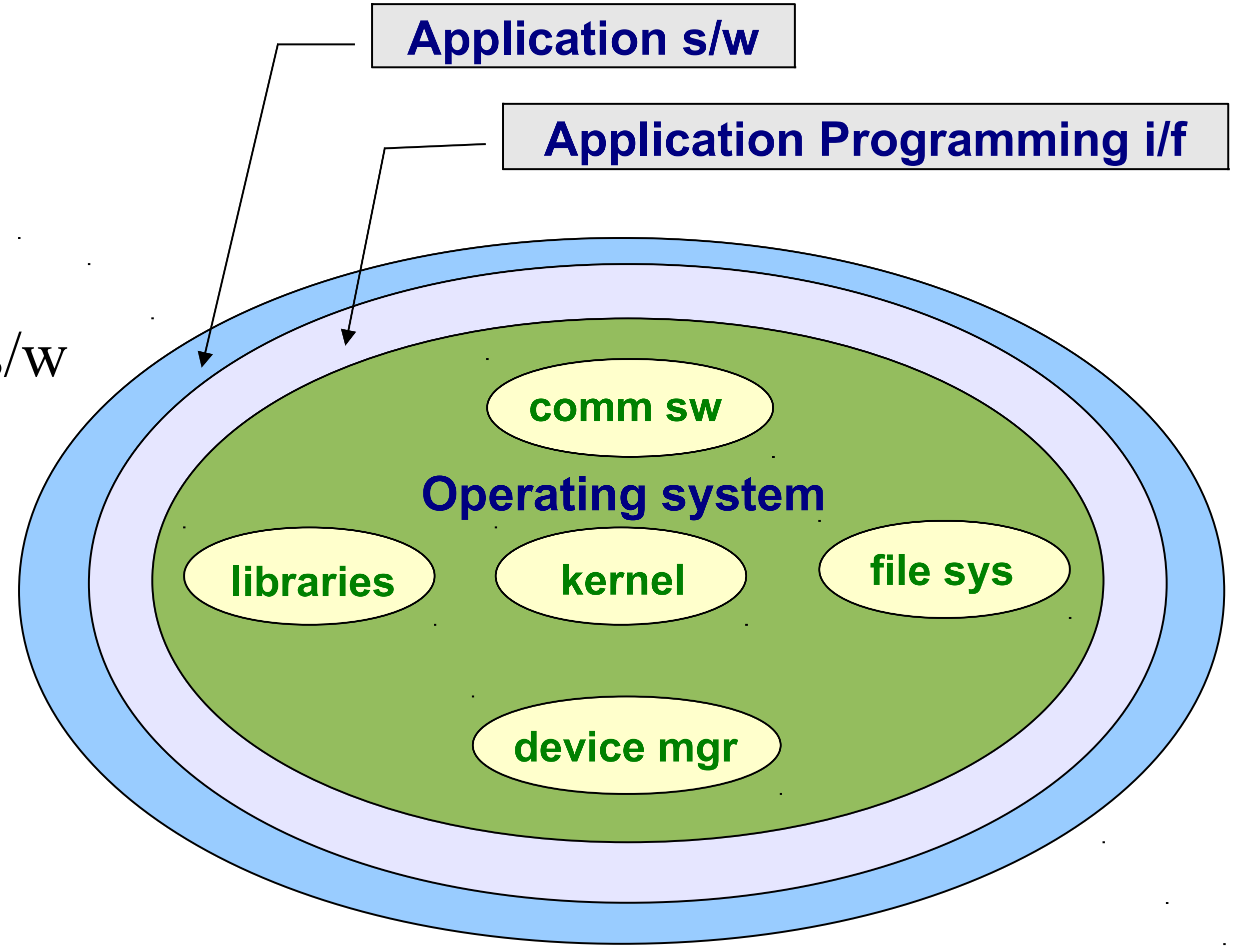
- **General Purpose Operating System**
  - Maximum Throughput and CPU Utilization
  - User experience through enhanced graphics
- **Real Time Operating System**
  - Handling timing constraints imposed by the application
  - Respond to events quickly
  - Determinism

RTOS Vs GPOS

#	Evaluation Metrics	General Purpose OS	Real Time OS
1	Determinism	• Non Deterministic	• All RTOS functions should execute in a fixed/deterministic amount of time
2	Load Independent Timing	• Not Applicable • Response becomes sluggish as number of tasks increase	• Remains Constant • Irrespective of system load, performance of the RT system should remain predictable
3	Task Level Scheduling	• Generally Round Robin Scheduling • Sometimes Priority based scheduling • Efforts are made to ensure that all tasks get a chance to execute	• Generally Priority based Preemptive Scheduling . • Time slicing only among tasks that hold the same priority • Ensure that the Highest Priority task is always executed, even if it is the most frequent
4	Interrupt Management	• Nesting may be disabled. • Interrupt latency, response and recovery are not performance metrics	• Nesting is always enabled. • Interrupt latency, response and recovery are very important performance metrics

RTOS Architecture

- An RTOS consists of
  - Kernel
  - Device Manager
  - Networking protocol s/w
  - Libraries
  - File system (optional)



RTOS Services

- **Multitasking, Task & Thread Management - creation and scheduling of tasks, priority based preemptive scheduling**
- **Vectored Interrupt Service Routines - avoid polling**
- **Inter Task Synchronization and Inter Task Communications through mutual exclusions, signals, messages, shared memory, etc**
- **Timer Services such as periodic and aperiodic interrupts**

RTOS Services

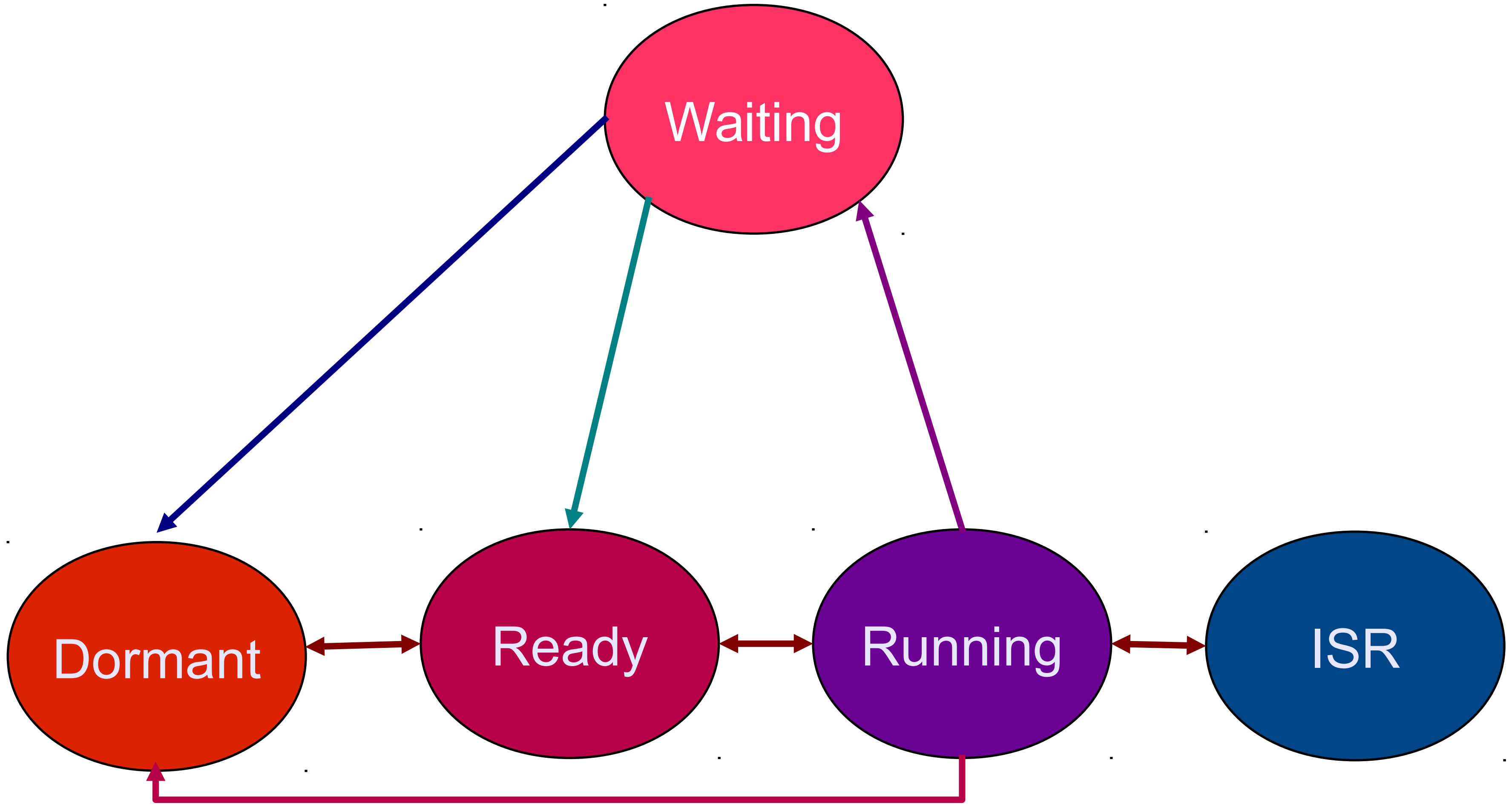
- **Device drivers to service the needed special devices**
- **Communication Protocol software (networking through Ethernet, wireless, etc)**
- **Application Programming Interface (APIs) to access kernel Services**

Real Time Tasks
&
Real Time Scheduling

Real Time Tasks

- **An RT application splits the work into Tasks.**
- **A real time task, also called a thread, is a simple program that thinks it has the CPU all to itself.**
- **It is generally implemented as an infinite loop (periodic tasks).**
- **Each Task is assigned its own set of CPU Registers, Stack Area and a Priority.**
- **A Task may be visualized through 3 logical components**
  - **The Task Function** (Logic) – What the task is out to achieve
  - **The Task Control Block** – Holds configuration information of the task like Priority, State, Period, Stack Pointer
  - **The Task Stack** – Maintains context information of the task during switches

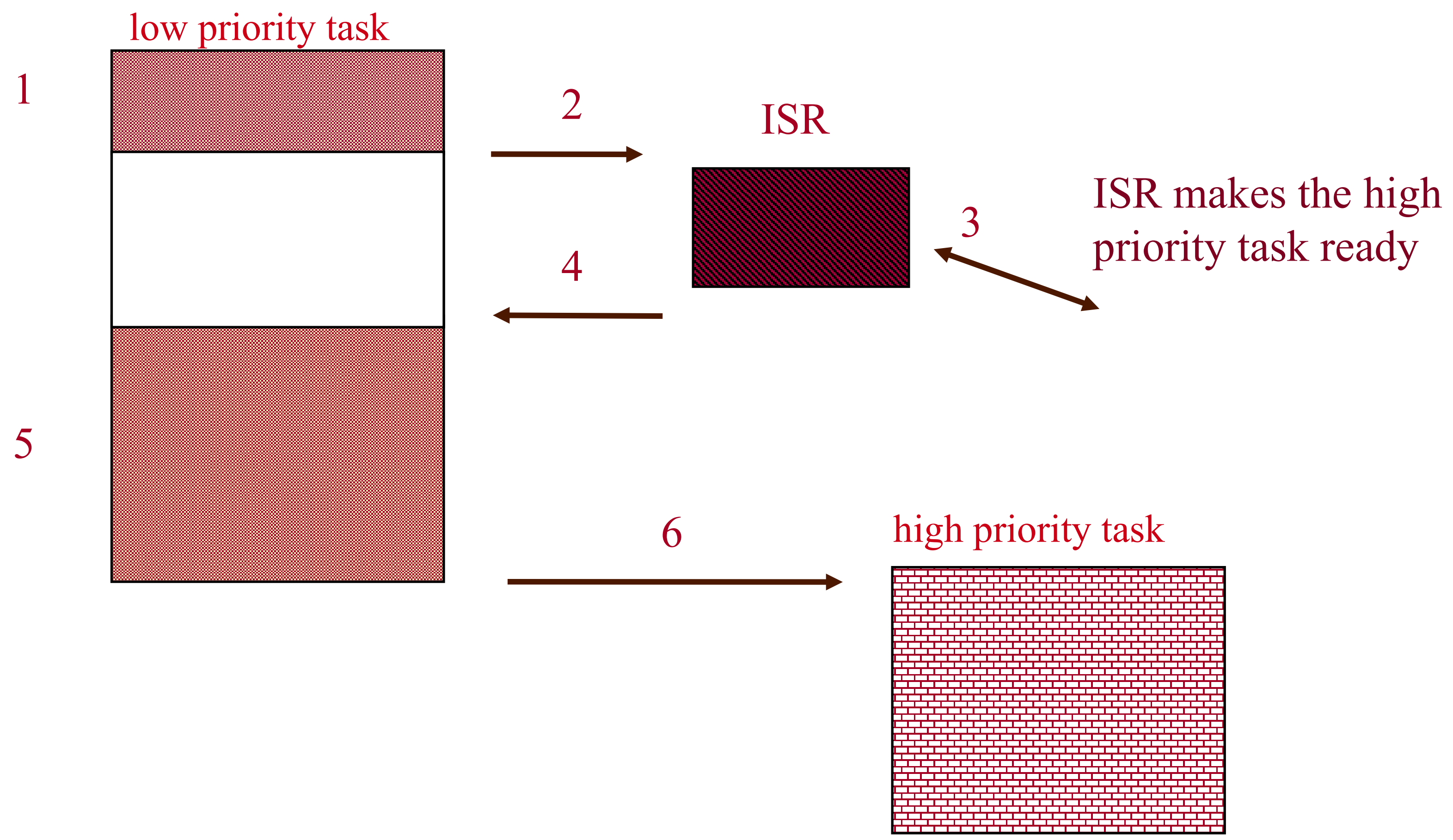
Task States



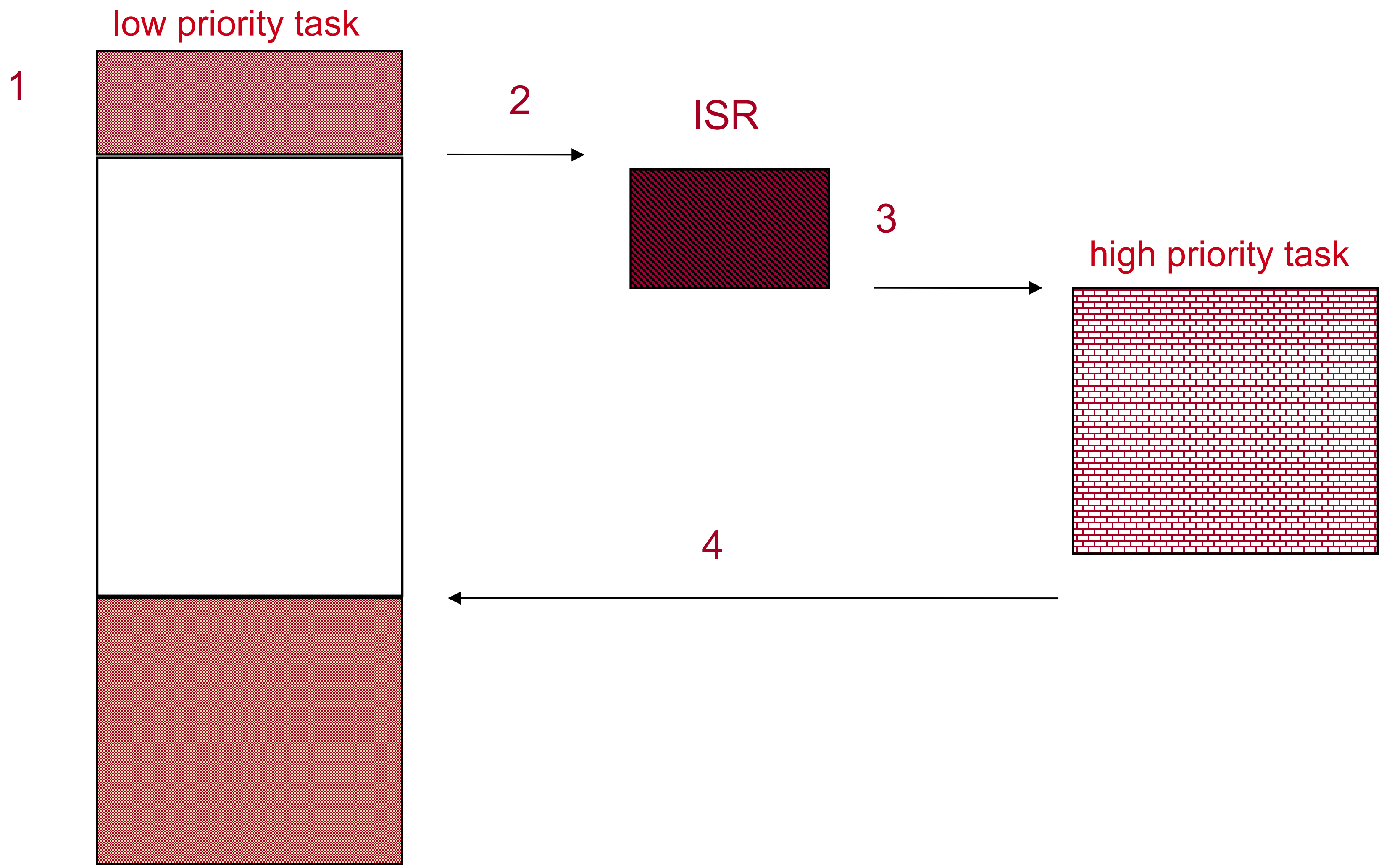
Real-time Scheduling

- **Goals**
  - Meeting the timing constraints of the system
  - Preventing simultaneous access to shared resources and devices
  - Attaining a high degree of CPU utilization while satisfying the timing constraints of the system
  - Reducing the cost of context switches caused by preemption

Kernel Types – Non Preemptive Kernel



Kernel Types – Preemptive Kernel

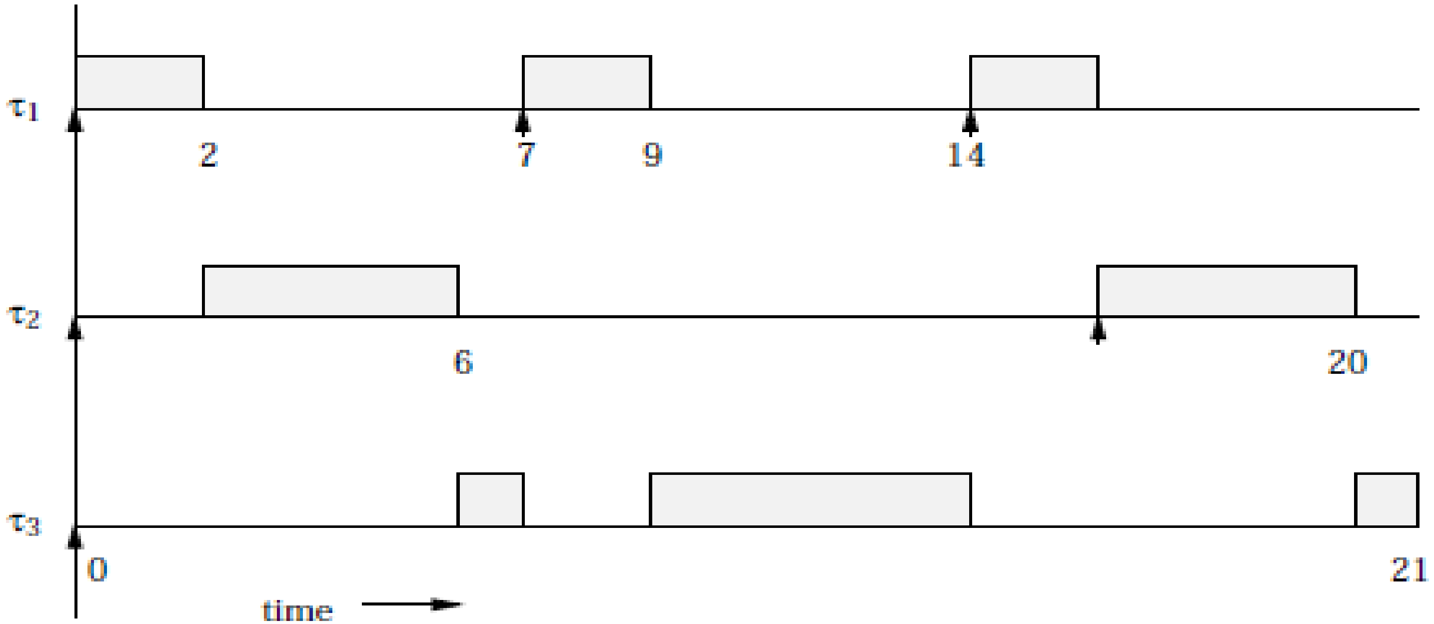


Task Timing Properties

- **Each task(Ti) occurring in a real-time system has some timing properties, which should be considered when scheduling tasks.**
  - Release time (or ready time)
  - Deadline(Di)
  - Worst case execution time(Ci)
  - Period(Ti or Pi) (For periodic tasks)

Schedulability analysis

- **Timing diagrams provide a good way to visualize and even to calculate the timing properties of simple programs.**



Schedulability analysis

- **A better method of analysis is to derive conditions to be satisfied by the timing properties of a program, to meet its deadlines**
- **Necessary Conditions:**
  - Worst cast execution time must be less than period (***Ci < Ti***)
  - CPU Utilization factor ***Ui*** for a periodic task ***Ti*** is (***Ci/Pi***)
  - Overall system utilization (U) = ***∑ Ui ≤ 1***

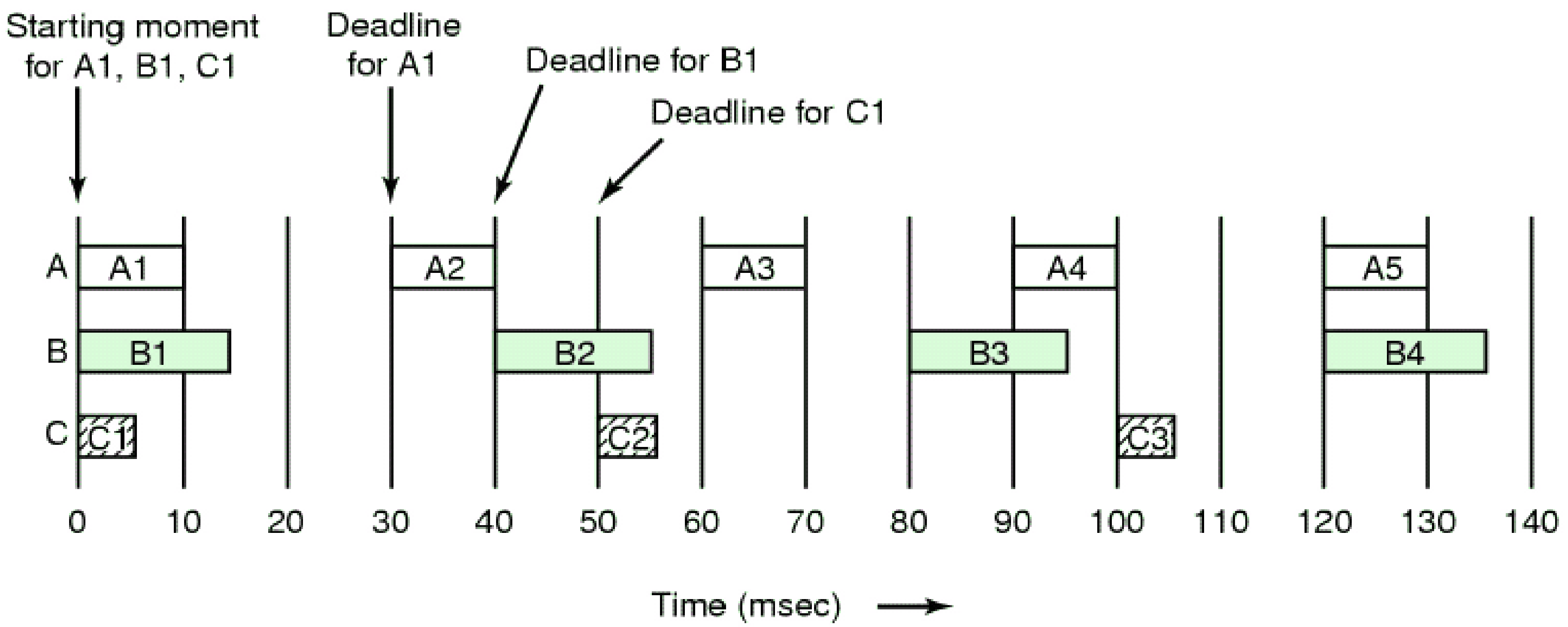
Utilization (%)	**Zone Type**	**Typical Application**
0-25	significant excess processing power – CPU may be more powerful than necessary	various
26-50	very safe	various
51-68	safe	various
69	theoretical limit	embedded systems
70-82	questionable	embedded systems
83-99	dangerous	embedded systems
100+	overload	stressed systems

RM, EDF, LLF Scheduling

Rate Monotonic	**Earliest Deadline First**	**Least Laxity First**
– Static-priority preemptive scheme – Assumes that all tasks are periodic – The priorities are inversely proportional to the period of the task.	– Dynamic-priority preemptive scheme – Higher priority is assigned to the task that has earlier deadline	– Dynamic priority preemptive scheme – The laxity of a task is deadline minus remaining computation time. – Highest priority is given to the task with least laxity

Scheduling Example

- **Consider 3 tasks with the following timing properties**



Scheduling Example

- **Are the tasks schedulable ??**

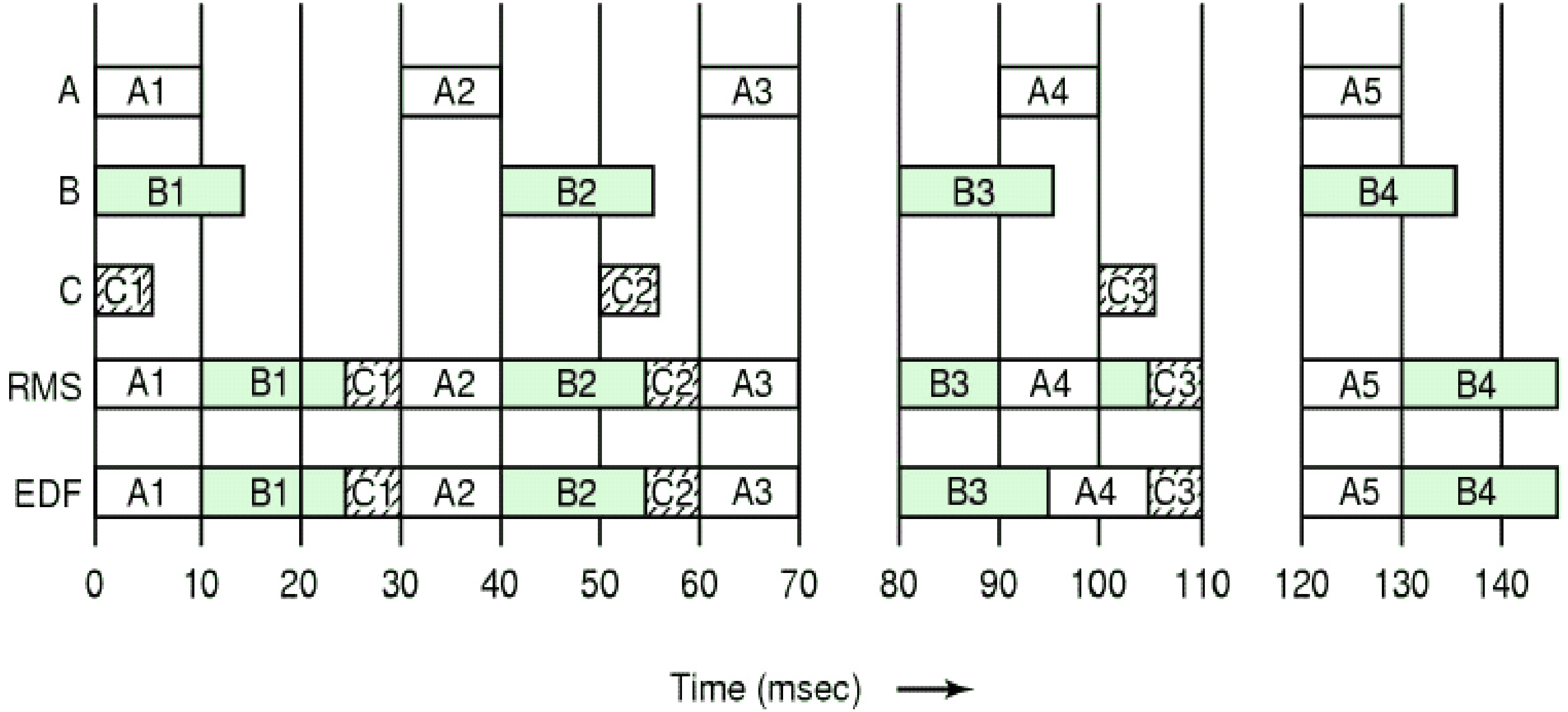
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

$$\frac{10}{30} + \frac{15}{40} + \frac{5}{50} = 0.808$$

- YES

Scheduling with RM and EDF

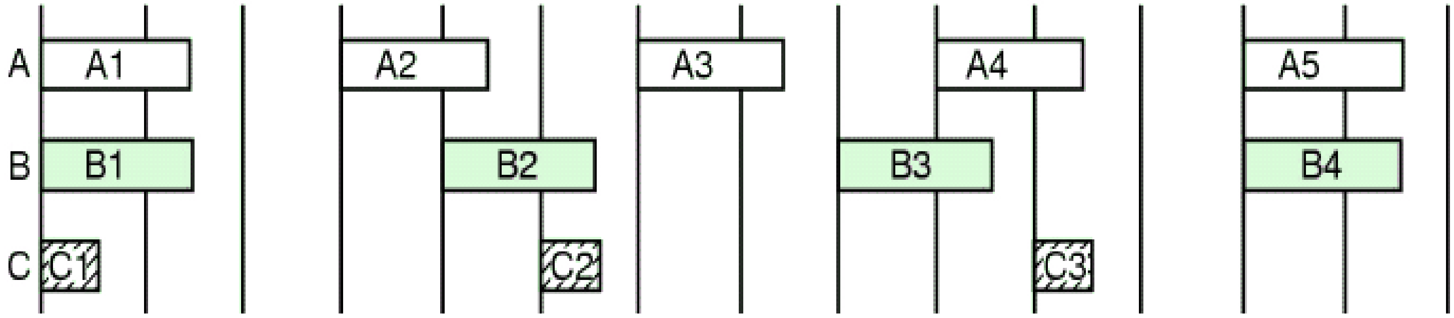
- **Analyze using timing diagram**



Modify Example

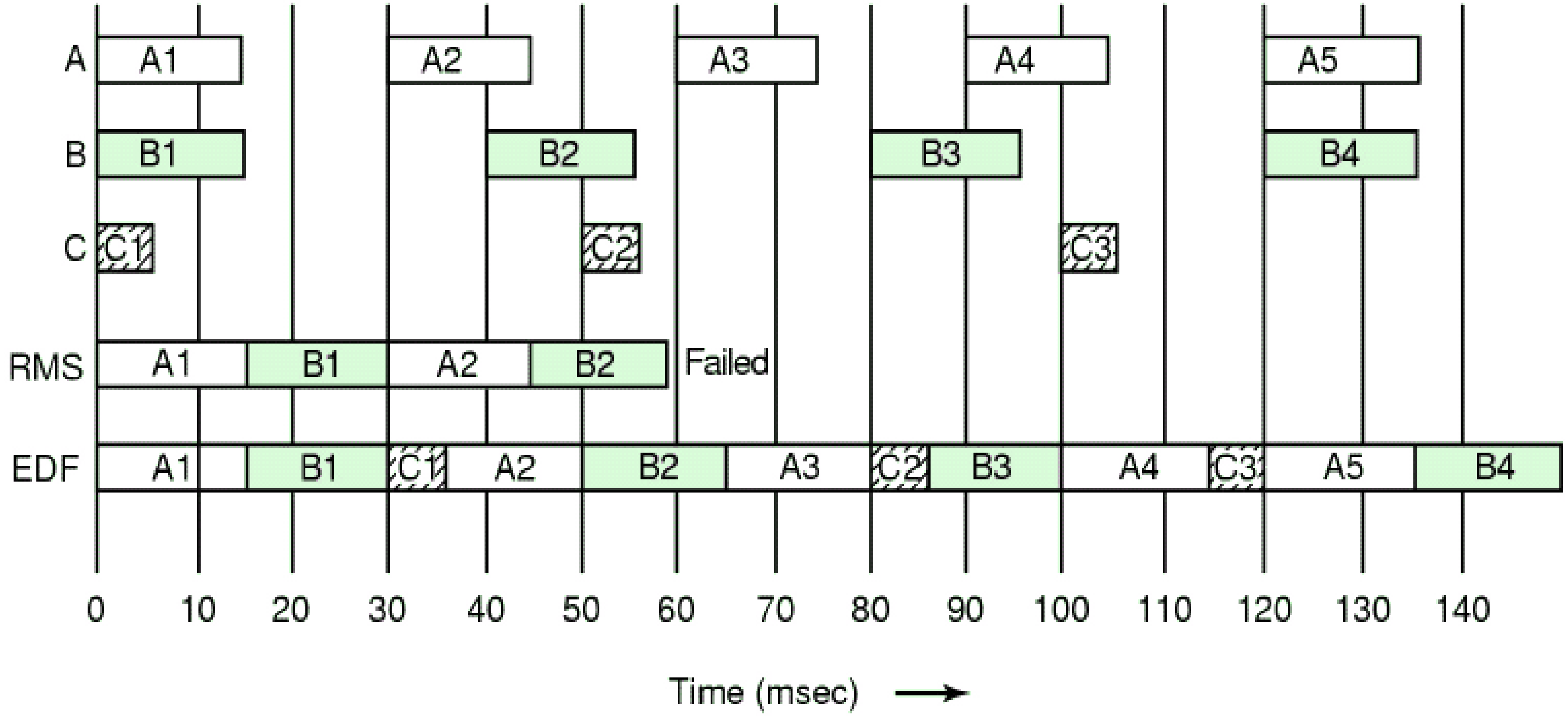
- **Increase A's CPU requirement to 15 milli sec**
- **The system is still schedulable**

$$\frac{15}{30}+\frac{15}{40}+\frac{5}{50}=0.975$$



Scheduling using RM and EDF

- **Analyze using timing diagram**



RM Scheduling Condition

- **RM Scheduling is guaranteed to work if the CPU utilization is not too high**
- **Condition of Schedulability**

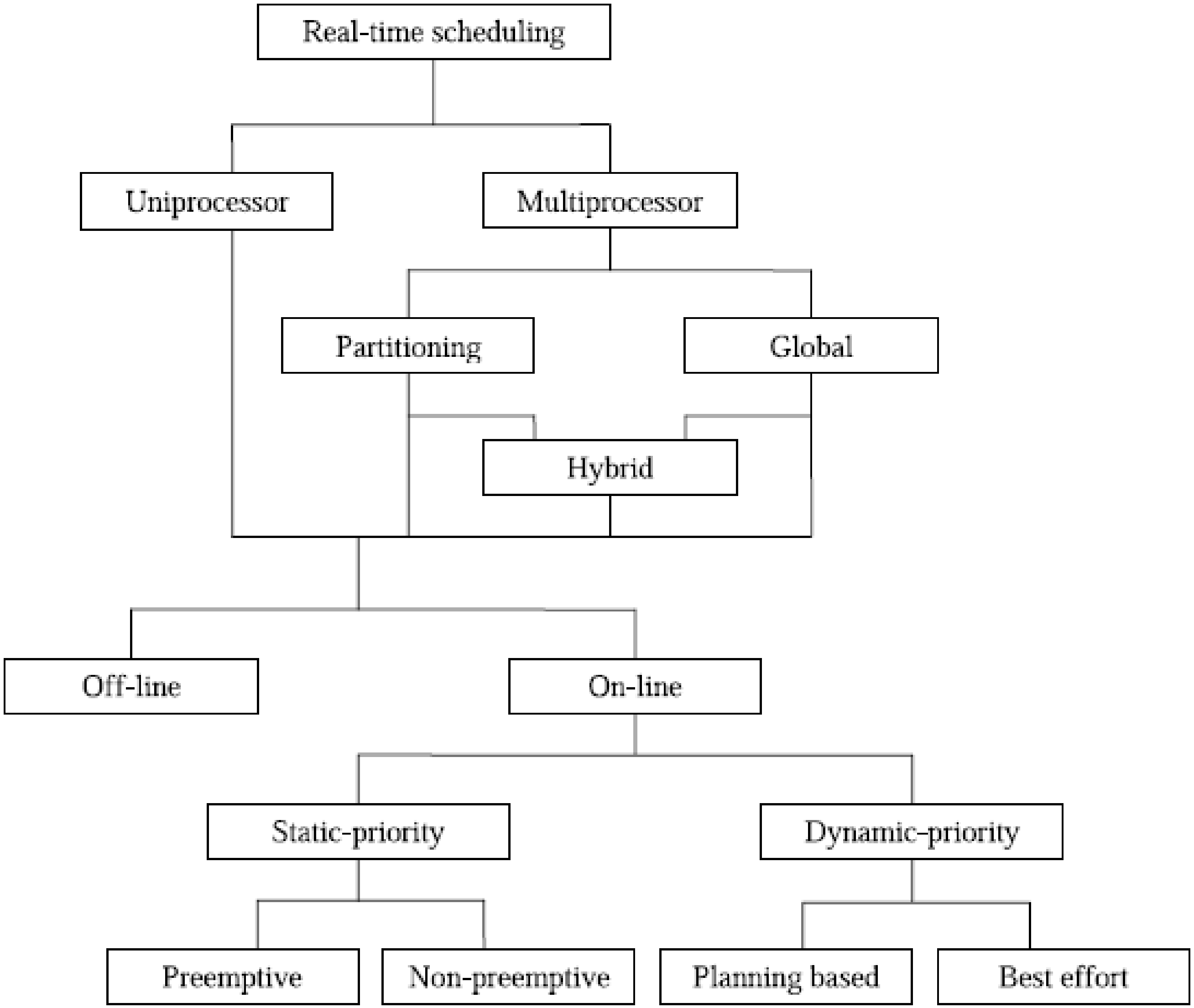
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$$

  - Where m = number of tasks
  - For three tasks, CPU utilization must be less than 0.780

Performance Comparison

- **RM Scheduling which is simple and easy to implement can be used for systems with low CPU utilization**
- **EDF Scheduling always works for any schedulable set of tasks. Upto 100% CPU utilization**
- **LLF Scheduling is similar in properties to EDF**
  - Takes into account that laxity time is more meaningful than deadline for tasks with mixed computing sizes

Classification of Real-time Scheduling



Multiprocessor Scheduling Algorithms
Global Scheduling Algorithms

- **Store the tasks that have arrived in a queue, which is shared among all processors**
- **Each processor maintains**
  - Status table of tasks it has committed to run.
  - Table of the surplus computational capacity at every other processor. (Each processor regularly sends to others the fraction of the next window that is free)
- **A Overloaded processor selects a processor that is most likely to be able to successfully execute that task by its deadline and ships the tasks out**
- **Eg: Focused addressing and bidding algorithm**

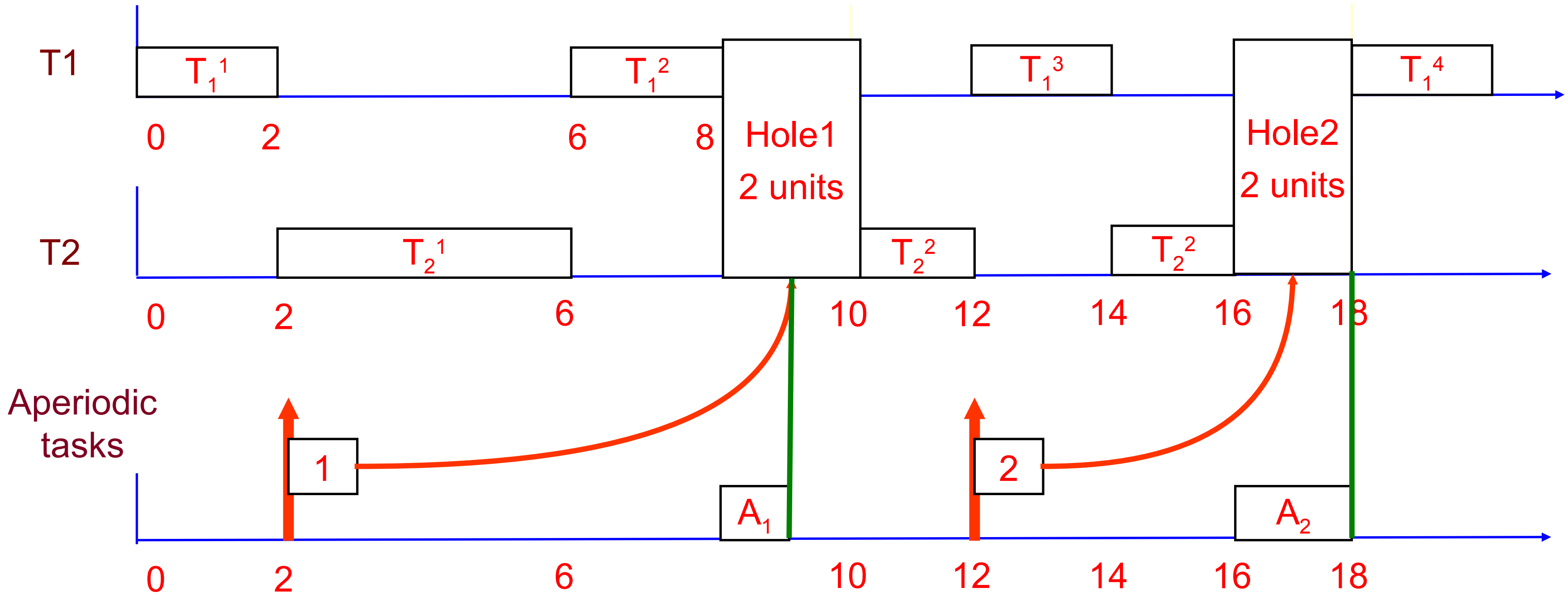
Multiprocessor Scheduling Algorithms
Partitioning Scheduling Algorithms

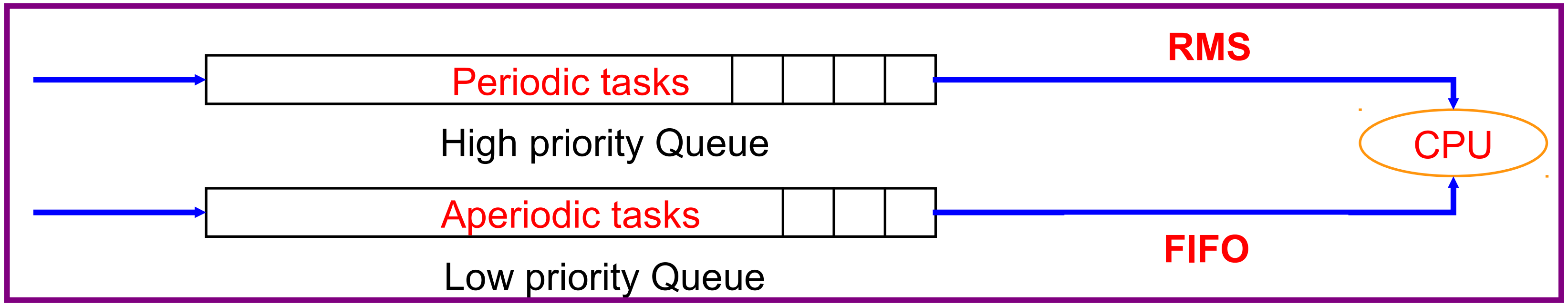
- **Tasks are partitioned such that all tasks in a partition are assigned to the same processor**
  - Tasks of same class, are guaranteed to satisfy the RM schedulability on one processor
  - Tasks are not allowed to migrate
- **Partitioning scheduling has a low scheduling overhead compared to global scheduling, because tasks do not need to migrate across processors**
- **Eg: Next fit algorithm for RM scheduling**

Scheduling of Aperiodic Tasks

- **Background: scheduled when processor is idle**
- **Interrupt-driven: scheduled on arrival**
- **Periodic server: defined by (ps, es) for processing aperiodic tasks. Budget replenishes at ps intervals.**
  - Bandwidth non-preserving server
    - If scheduled and queue empty then budget set to 0.
    - Polling server
  - Bandwidth preserving server
    - Improves on the polling server by preserving budget (bandwidth) when aperiodic queue is empty.
    - Deferrable servers
    - Priority Exchange and Deferred Servers

Background Scheduling: Example

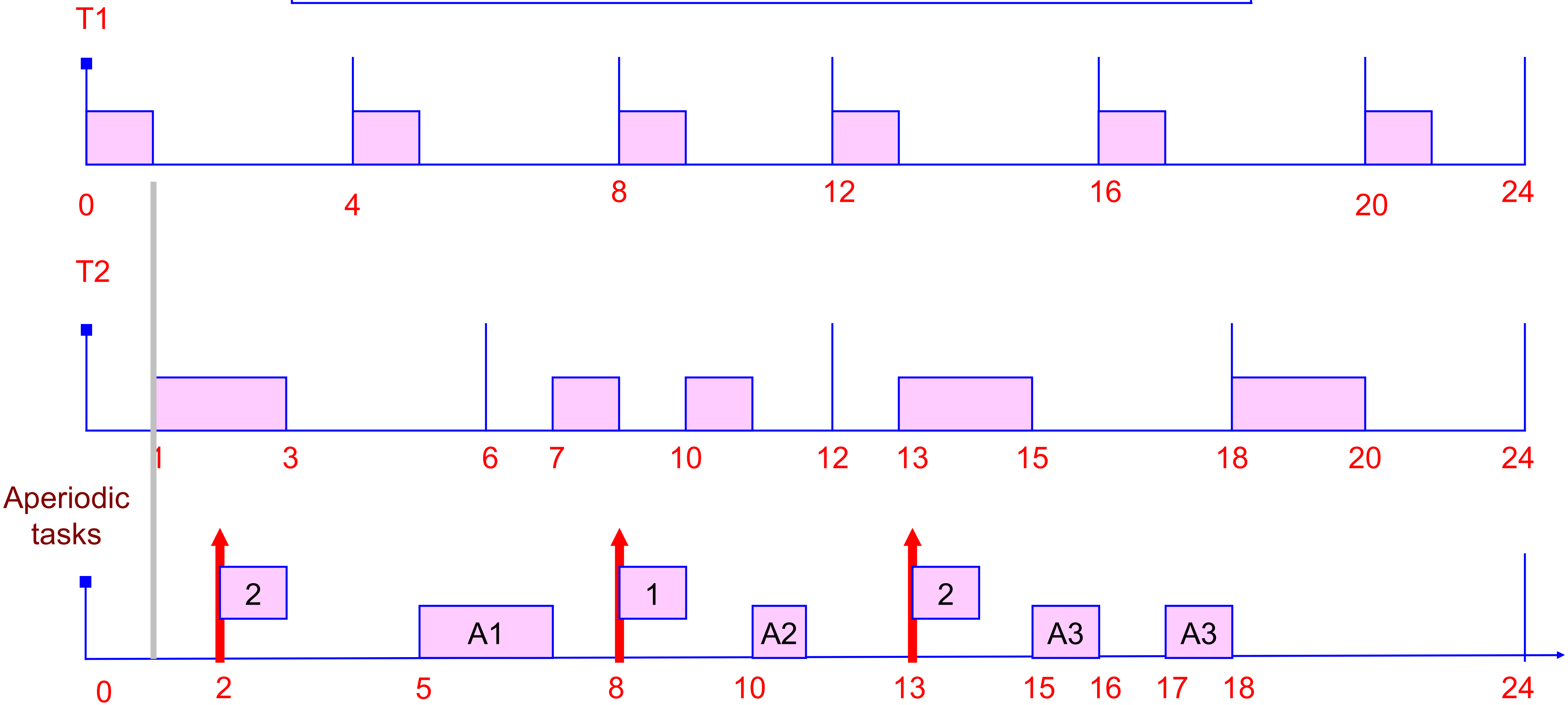




Polling server: Example

Task set: $T_i = (c_i, p_i)$

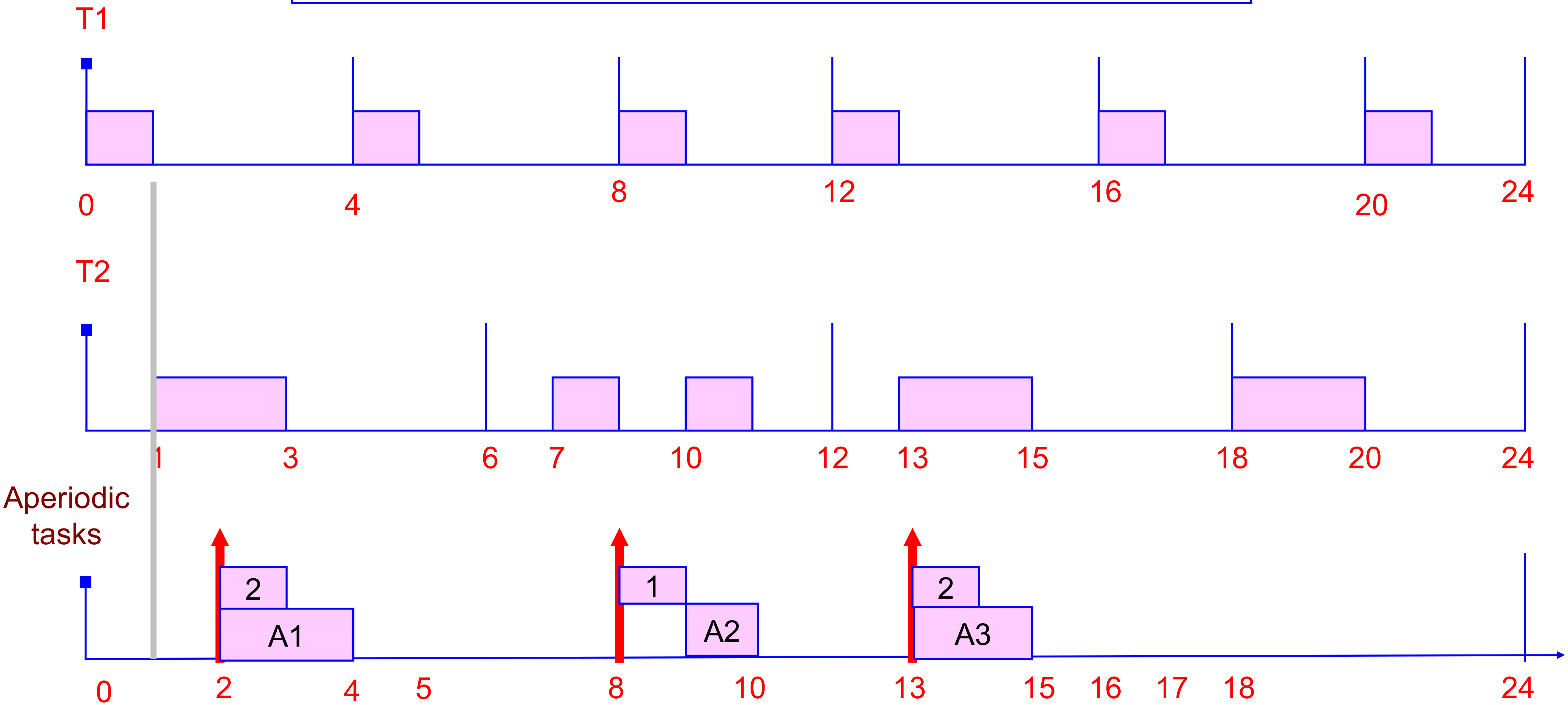
T1 = (1,4) , T2 = (2,6) and Ts = (2,5)



Deferred server: Example

Task set: $T_i = (c_i, p_i)$

T1 = (1,4) , T2 = (2,6) and Ts = (2,5)

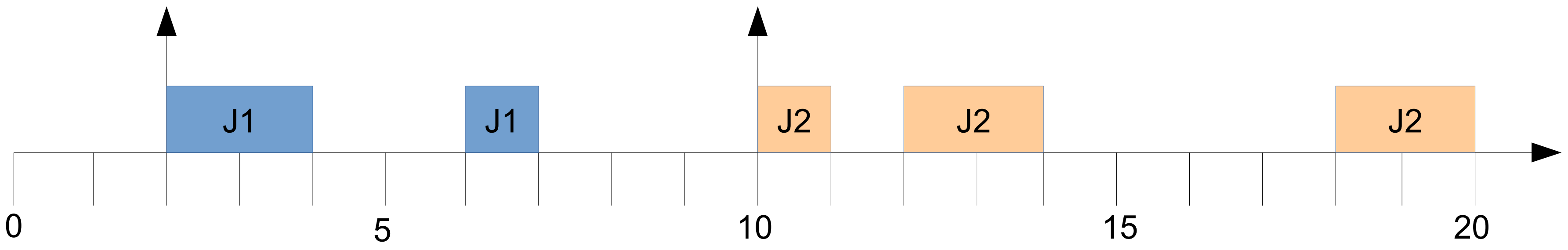


Scheduling of Sporadic Tasks

- **Methods**
  - Consider sporadic tasks as periodic tasks with a period equal to their minimum inter-arrival time
  - Define a fictitious periodic task of highest priority and when the task is scheduled, run any sporadic task awaiting service
  - The Deferred server
    - When sporadic tasks are scheduled and none is awaiting service, it schedules periodic tasks in order of priority
    - Wastes less bandwidth

Scheduling of Sporadic Tasks

- **Consider a deferred server js (2, 6) for a collection of sporadic tasks**
  - Sporadic task j1 with c1 = 3 arrives at time 2
  - Sporadic task j2 with c2 = 5 arrives at time 10

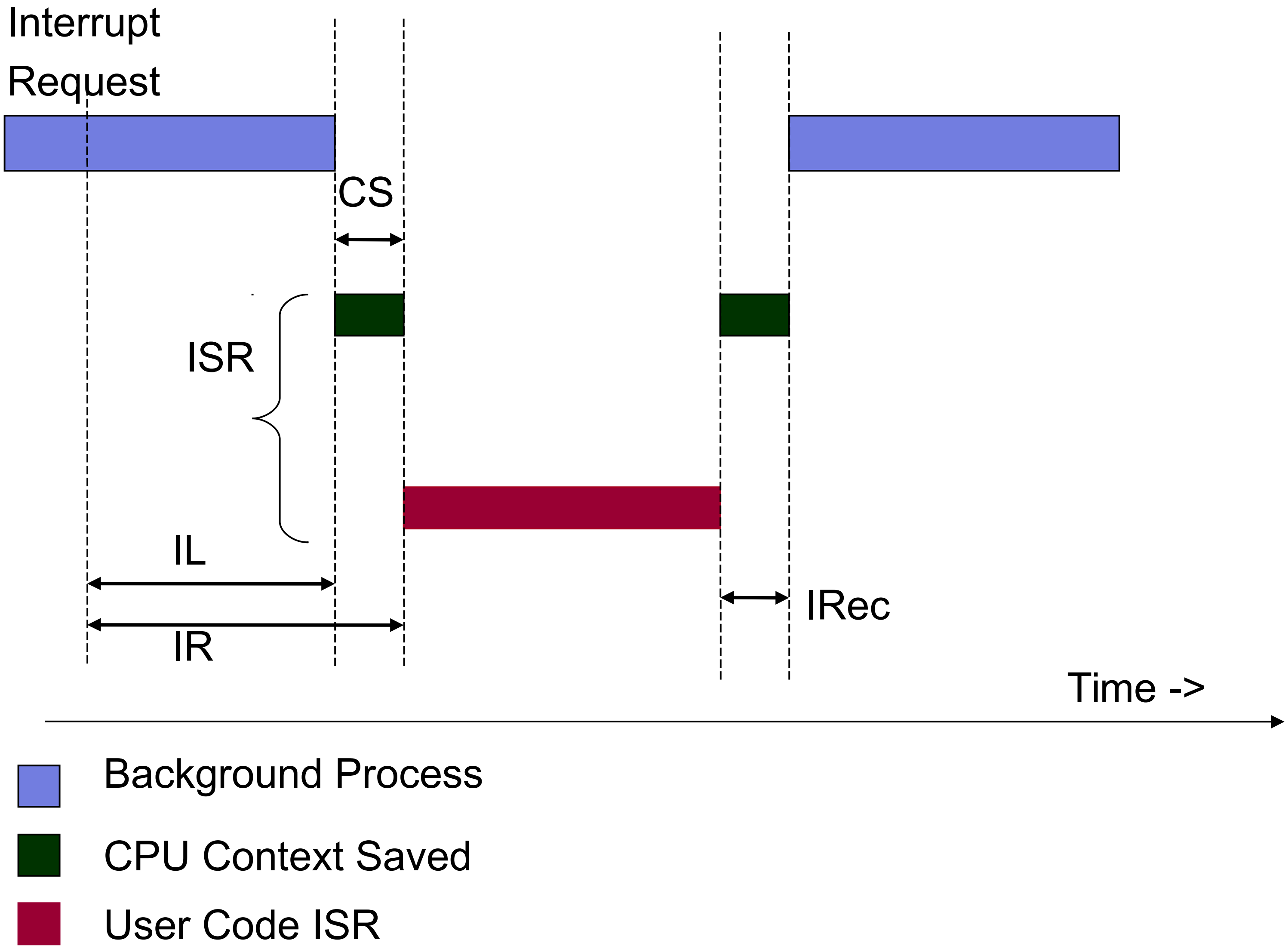


Interrupt Service Routines

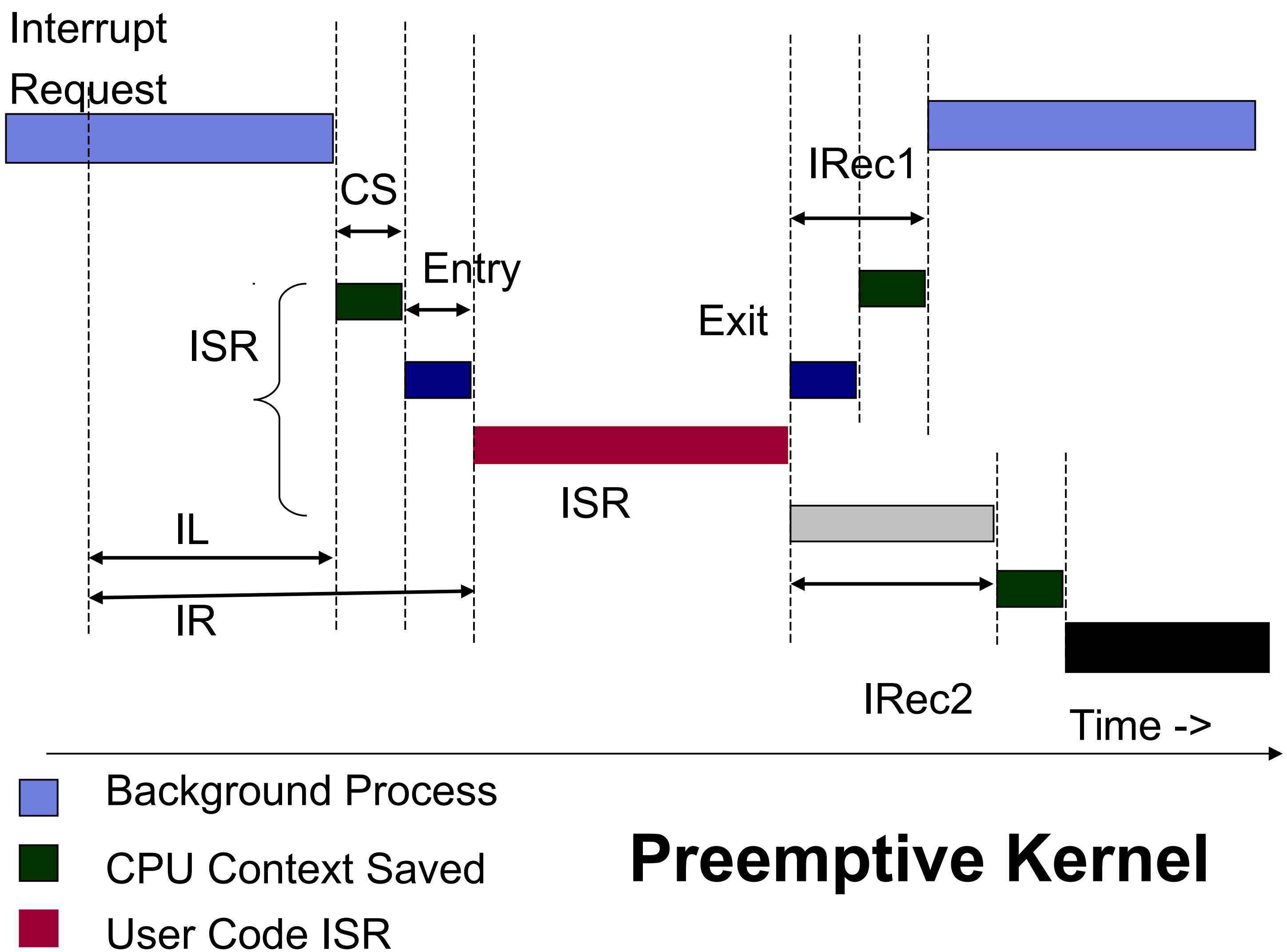
Interrupts

- **When an interrupt occurs**
  - CPU saves its context on the stack, Jumps to the Interrupt Servicing Routine (ISR), Executes ISR and Returns
  - **Interrupt Latency**
    - max time interrupts are disabled + time to begin servicing the interrupt
  - **Interrupt Response Time**
    - Interrupt Latency + time to start execution of 1st instruction in ISR
  - **Interrupt Recovery Time**
    - time for CPU to return to interrupted code / highest priority task

Interrupts in Non Preemptive Kernels

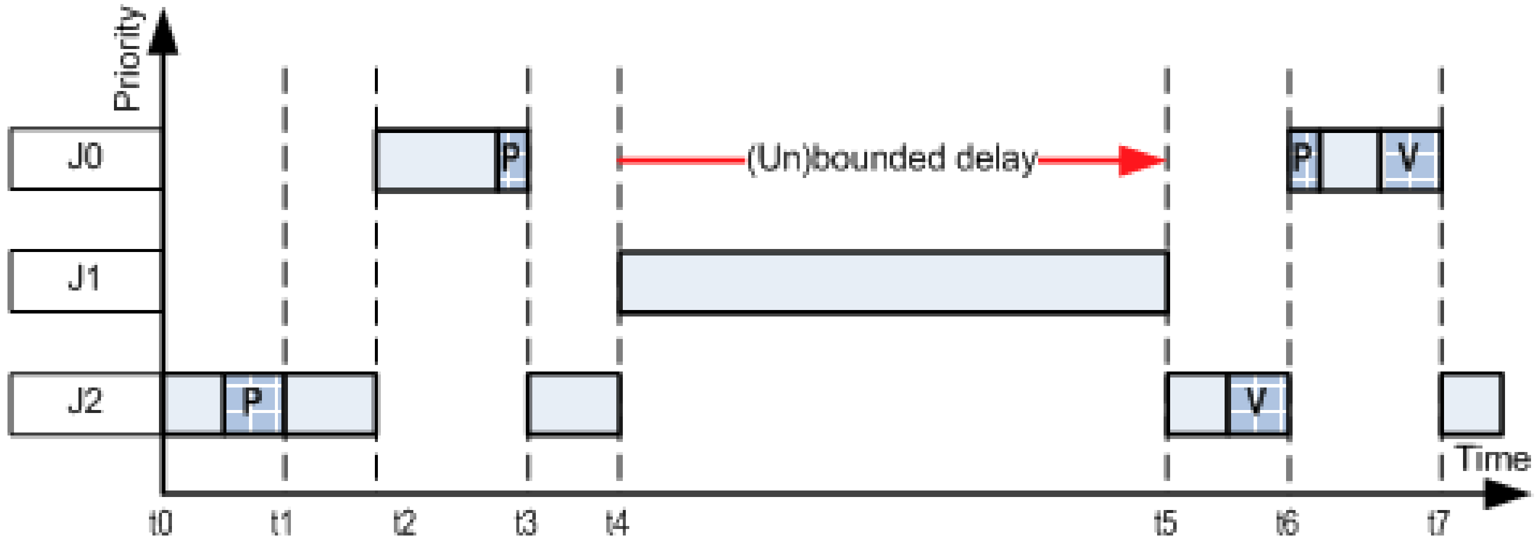


Interrupts in a preemptive kernel



Inter Task Synchronization

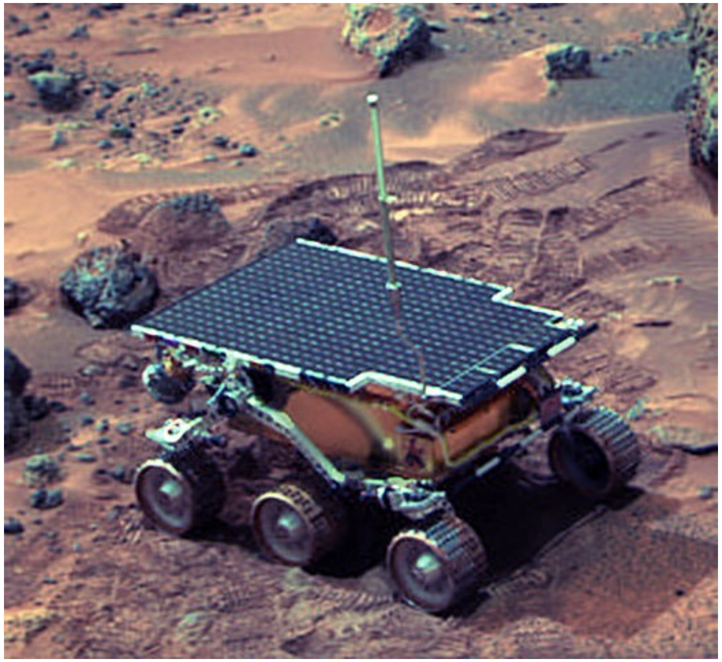
Priority Inversion Problem



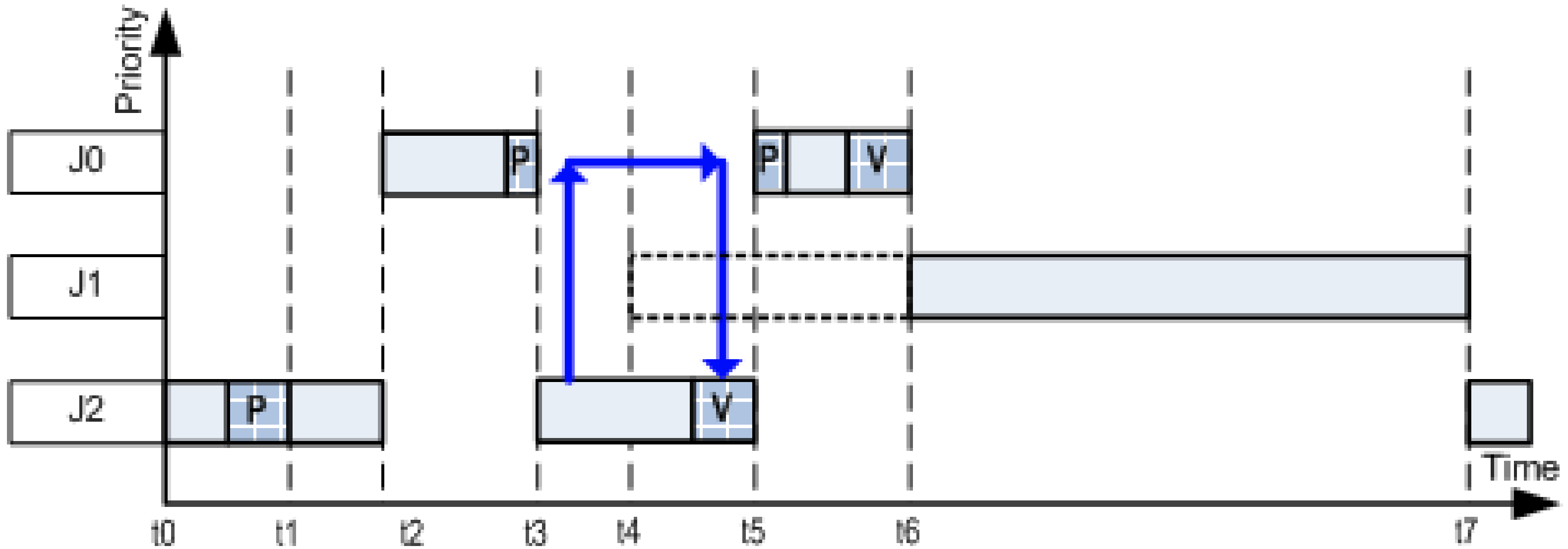
- **Effectively, the high priority task, J0, is blocked by medium priority task J1. This condition is called priority inversion.**
- **Primary techniques to solve priority-inversion**
  - Priority Inheritance Protocol
  - Priority Ceiling Protocol

Mars Pathfinder

- **The flow of information and images was interrupted by a series of system resets**
  - The Pathfinder's applications were scheduled by the VxWorks RTOS using pre-emptive priority scheduling
  - The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus synchronized with mutual exclusion locks
  - A very high priority bus management task, also accessed the bus with the same mutexes
  - A long-running communications task, having higher priority than the meteorological task, but lower than the bus management task, prevented it from running.
  - Watchdog timer noticed that the bus management task had not been executed for some time, concluded that something had gone wrong, and ordered a total system reset

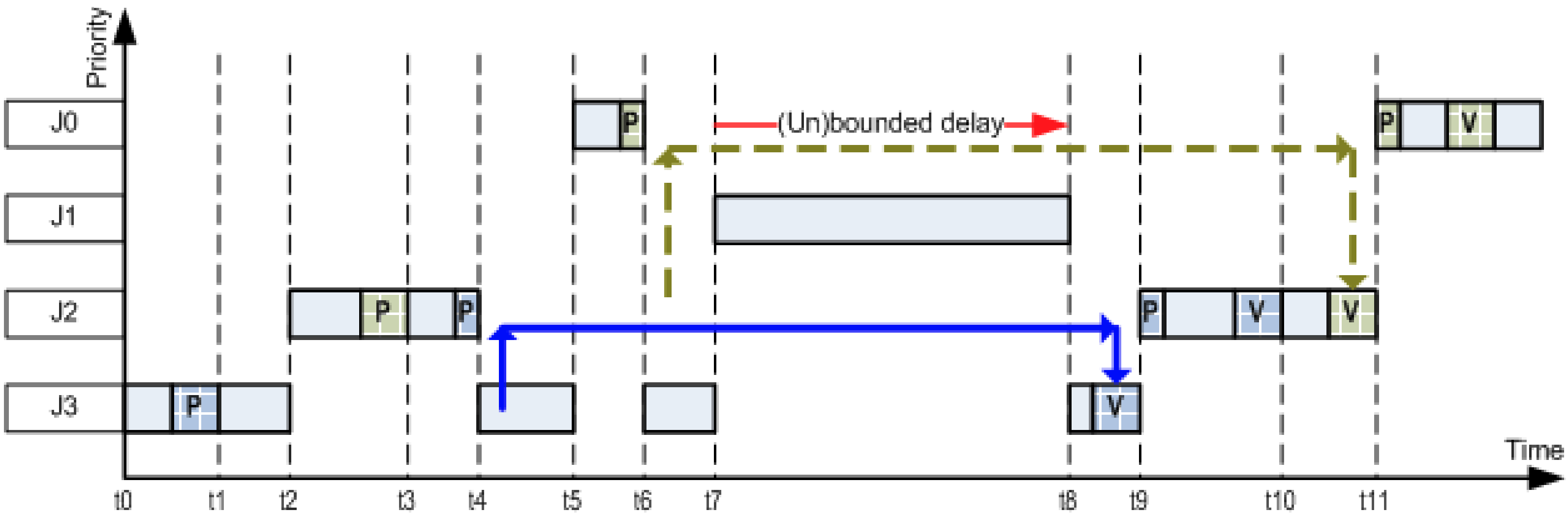


Simple Priority Inheritance Protocol

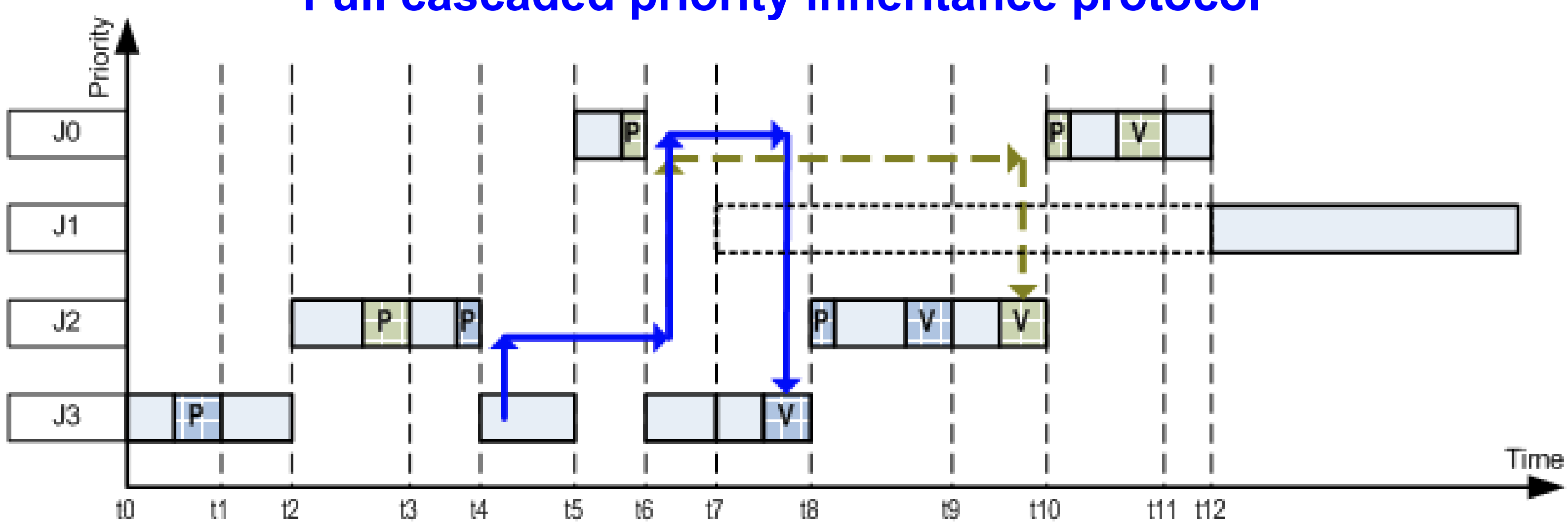


- **Prevents priority inversion by temporarily raising the priority of the task holding the mutex.**
- **Reactive protocol. Priority is raised to the priority of the task that is waiting on the mutex, if that is greater.**
- **No support for multiple mutexes.**

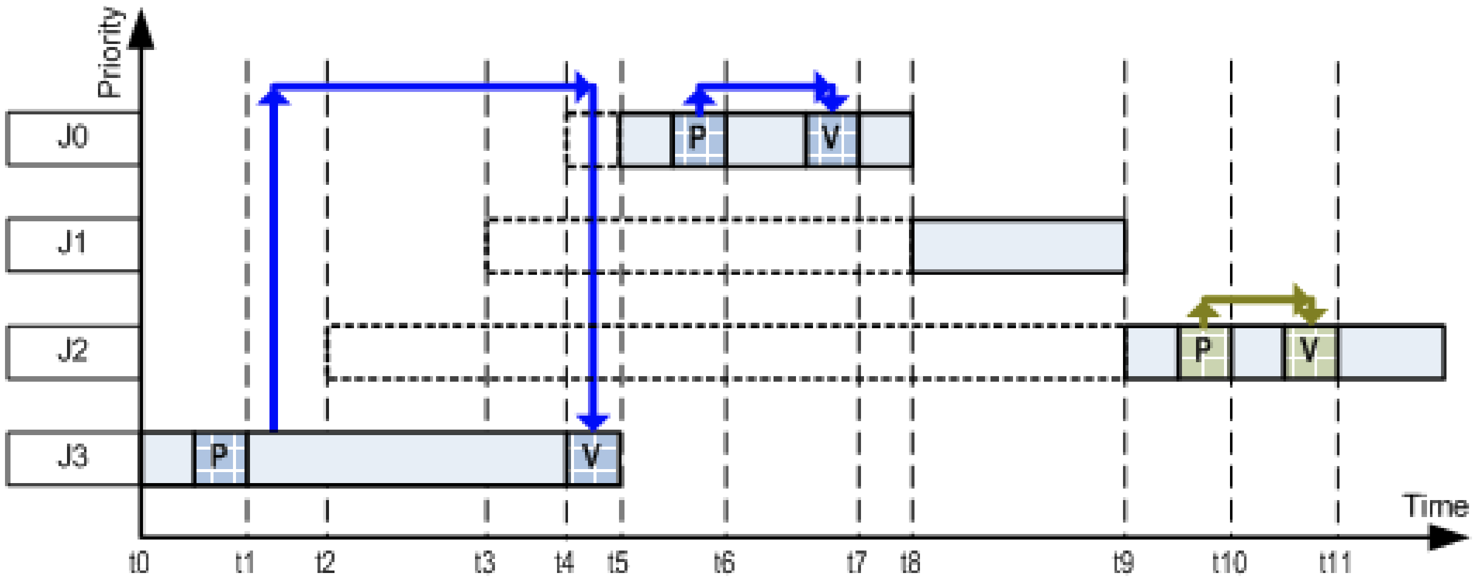
Priority Inheritance (with mutliple mutexes)



Full cascaded priority inheritance protocol



Priority Ceiling Protocol



- **Called Priority Protect Protocol in POSIX.**
- **Every mutex has a priority ceiling. The priority ceiling of a mutex should be greater than or equal to the priority of any task that might use it.**
- **Proactive protocol. When a task locks a mutex, its priority is immediately raised to the priority ceiling of the mutex.**
- **It can eliminate deadlock!**

Overview of available RTOS's

- **Three categories of real-time operating systems:**
  - Small, proprietary kernels. e.g. VRTX32, pSOS, VxWorks, Windows CE, MicroC-OS/III*
  - Real-time Linux extensions: RT-Linux, Xenomai, RTAI
  - Research kernels: MARS, ARTS, Spring, Polis, MicroC-OS/II

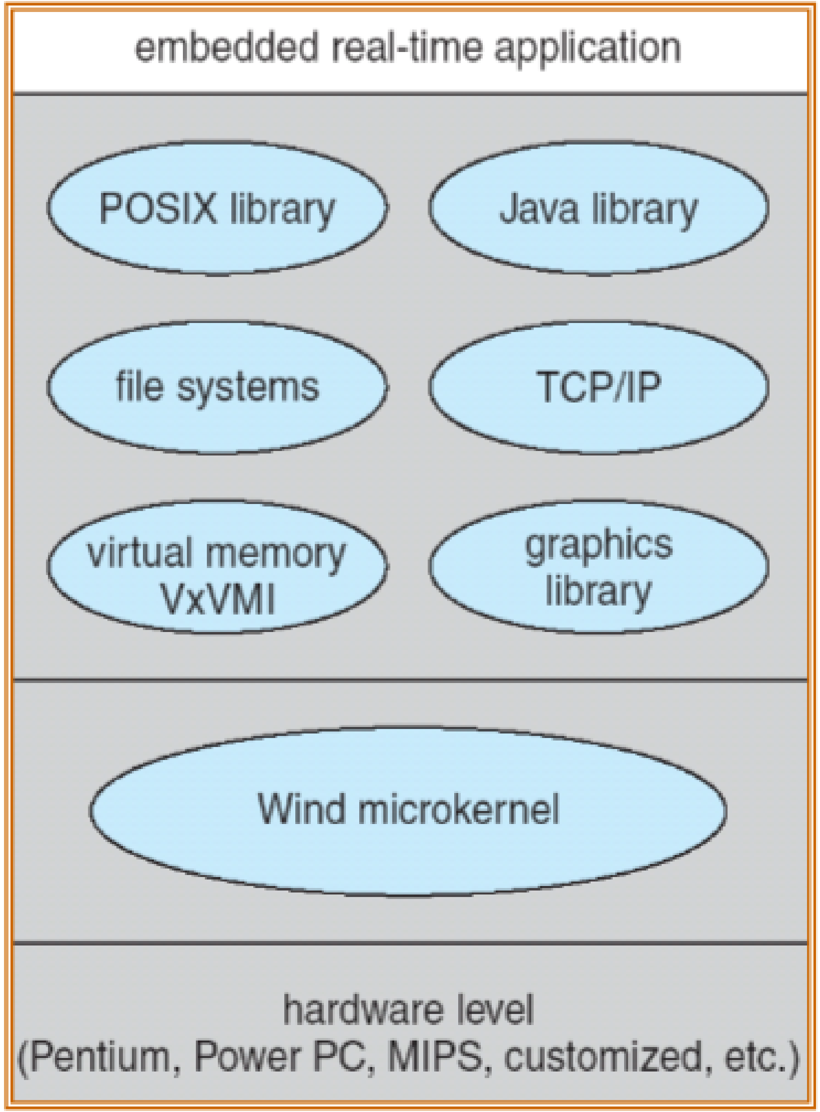
Hard	Real Time Operating System	Vx Works, Lynx, QNX Intime, HyperKernel, RTX Windows CE
Soft	General Purpose Operating System	Linux, NT

QNX Neutrino

- **Developed by Gordon Bell and the students at the University of Waterloo in 1980**
- **Supported on ARM, MIPS, Power PC, x86 and Pentium**
- **Micro-kernel based, and most of the OS is run in the form of a number of small tasks, known as servers**
- **Configurable to small size (64 K kernel ROM, 32 K kernel RAM)**
- **Supports Symmetric multiprocessing and strict priority-preemptive scheduling**
- **IEEE1003 real-time std compliant and POSIX threads**
- **Finds Applications in Embedded systems for over 20 years in mission and life critical systems, medical instrumentation, aviation and space systems, process control systems**
- **Avg Interrupt latency – 1.6us**

Vx Works

- **Developed by Wind River Systems of California**
- **Pentium, Motorola, Power PC, ARM**
- **Micro-kernel based**
- **Preemptive and non-preemptive scheduling**
- **Manages interrupts with bounded interrupt and dispatch latency times**
- **POSIX Compliant threads**
- **Shared memory and message passing for inter process communications**
- **Used in automobiles, routers, switches, Mars Pathfinder**
- **Avg interrupt latency – 1.7us**



LynxOS

- Lynx is a Unix like real time operating system
- Developed for Motorola 68010, ported to x86, ARM, Power PC
- Support hard real time applications, due to extremely fast interrupt routines known as Multiple Priority Light Weight kernel Task-based Interrupt Handling
- Mostly used in embedded systems in avionics, aerospace, communications

Microsoft Windows CE

- Supported on x86, MIPS, ARM processors for embedded systems
- Supports threads, priority inheritance
- Non - POSIX compliant, 10% of Win32 APIs
- Applications can be developed on Visual Studio
- Windows Mobile, Pocket PC, Smart Phone are based on CE
- Avg interrupt latency – 2.4us

Variants of Real-Time Linux Extensions

Real Time Linux (RTLinux)	Real Time Application Interface (RTAI)	Xenomai Framework
– Developed at the New Mexico Institute of Mining and Technology – RTOS Micro kernel running entire Linux in fully preemptive mode – Runs special real-time tasks and interrupt handlers – FiFo, Shared memory, Semaphores. POSIX mutexes and threads – Avg interrupt latency - 15us – Used to control robots, data acquisition systems, manufacturing plants, and other time-sensitive instruments and machines	– Developed by programmers at the Department of Aerospace Engineering, Milano – Adeos based patch over Linux kernel, with native real time tasks, interrupt handlers and services – Platforms - MIPS, x86-64, PowerPC, ARM – Semaphores, mailboxes, FIFOs, shared memory, RPCs – POSIX 1003.1c & POSIX 1003.1b(pqueues) – Avg interrupt response - 20us	– Implementing and migrating real time applications – Based on emulators for proprietary RTOS interfaces/apis, such as VxWorks and pSOS – Linux-hosted dual kernel, with pure Adeos. – User space RT tasks – Platfroms - x86, ARM, POWER, IA-64, Blackfin, nios