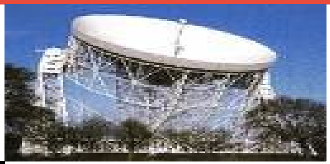
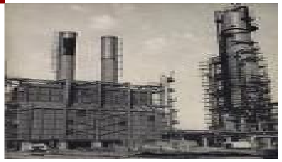







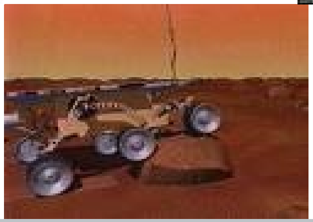


Real Time Systems & Real Time OS Concepts







Agenda

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

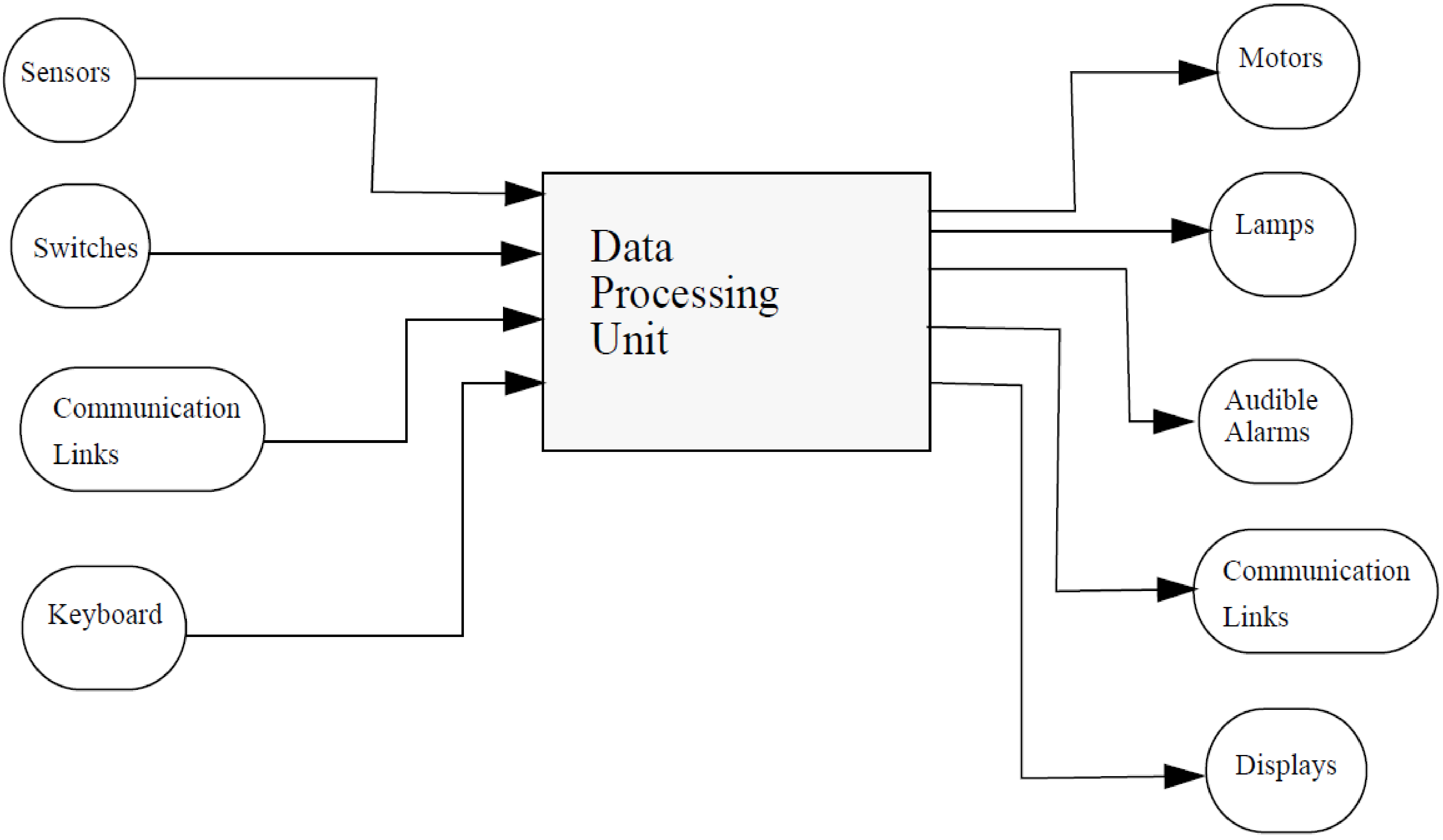
Real-Time System definition

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*";**

Real Time System Characteristics

- **Embedded Computer Systems**
  - Usually built around a micro-controller
  - Reads the input of various sensors, applies various filtering, calibration, and processing algorithms on the input data and produces output data to various actuators
- **Concurrent processing**
  - Involves processing of multiple inputs over the same time interval
  - Task scheduling is important for managing concurrency
  - Priority scheduling in which tasks with more stringent deadlines will be given a higher priority
- **Predictability**
  - Possible to show at “design time” that all the timing constraints of the application will be met. Systems require **determinism** to ensure predictable behavior

Typical Real-Time System

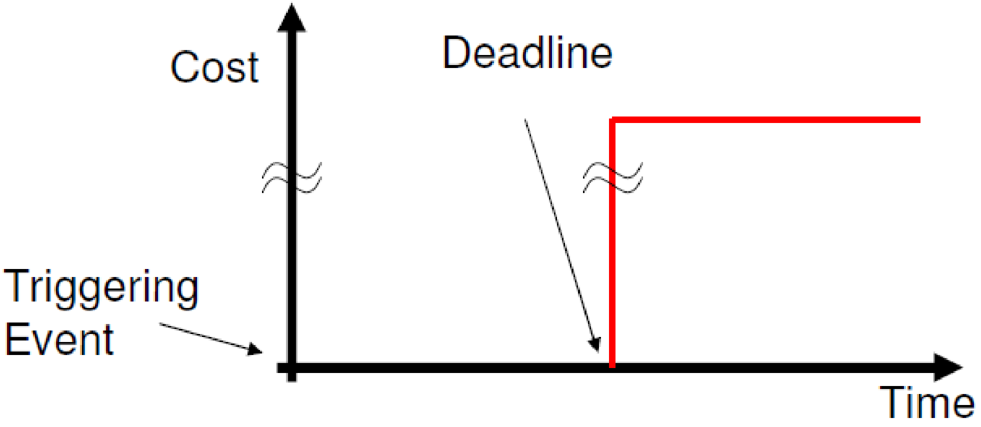


Real Time Tasks

- **Periodic tasks**
  - Time-driven, characteristics are known a prior (pi, ci)
  - Eg: Task monitoring temperature of a patient in an ICU
- **Aperiodic tasks**
  - Event-driven, characteristics not known a prior (ai, ri, ci, di)
  - Task activated on detecting change in patient's condition
- **Sporadic Tasks**
  - Aperiodic tasks with known minimum inter-arrival time.
  - pi : task period ai : arrival time ri : ready time di : deadline ci : worst case exec time

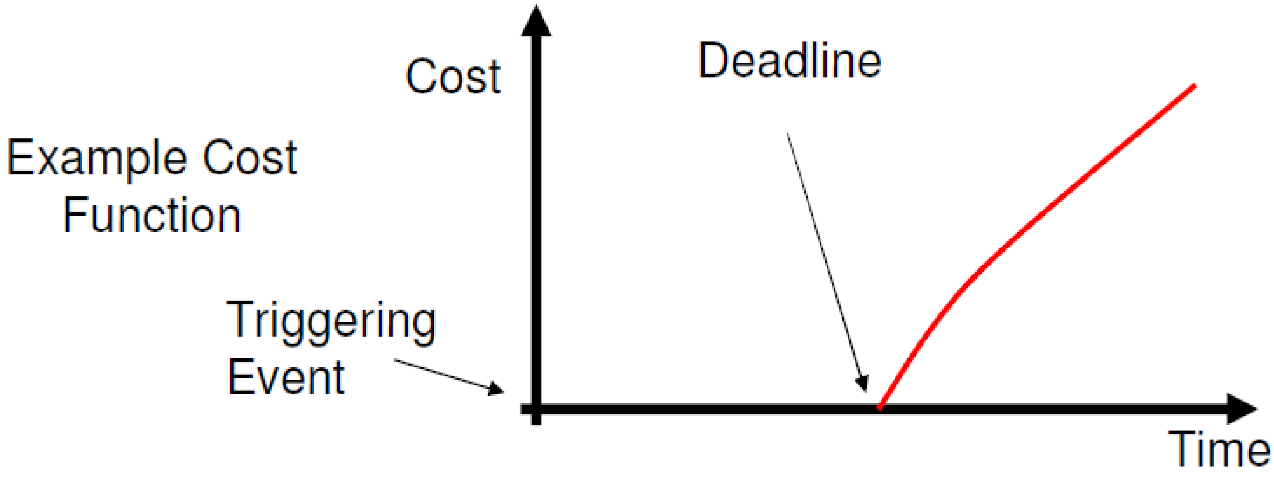
Hard Real-Time Systems

- An overrun in response time leads to a catastrophe (potential loss of life and/or big financial damage)
- Many of these systems are considered to be safety critical.
- Sometimes they are “only” mission critical, with the mission being very expensive.
- In general there is a cost function associated with the system.
- Eg: Aircraft Control Systems, Nuclear Power Stations, Chemical Plants, Life support systems

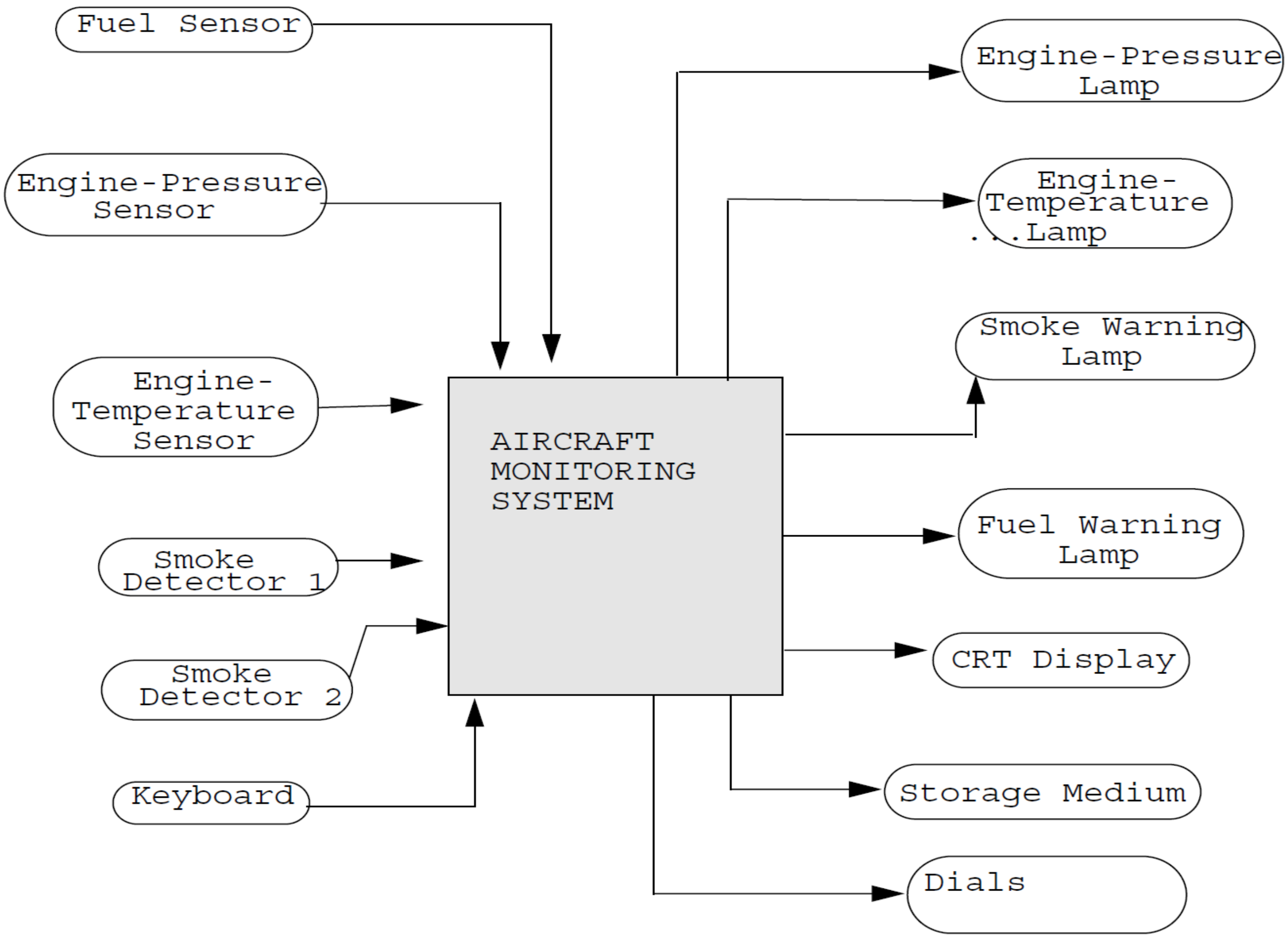


Soft Real-Time Systems

- An Deadline overruns are tolerable, but not desired.
- There are no catastrophic consequences of missing one or more deadlines.
- There is a cost associated to overrunning, but this cost may be abstract.
- Often connected to Quality-of-Service (QoS)
- Eg: Mutlimedia, Interactive video games



Aircraft Monitoring System (AMS), a Case Study



Real Time Scheduling Algorithms

Real-time System Properties

- Hard/Soft real-time tasks
- Periodic/Aperiodic/Sporadic tasks
- Preemptive/Non-preemptive tasks
- Multiprocessor/Single processor systems
- Fixed/Dynamic priority tasks
- Flexible/Static systems
- Independent/Dependent tasks

Goals of Real-time Scheduling

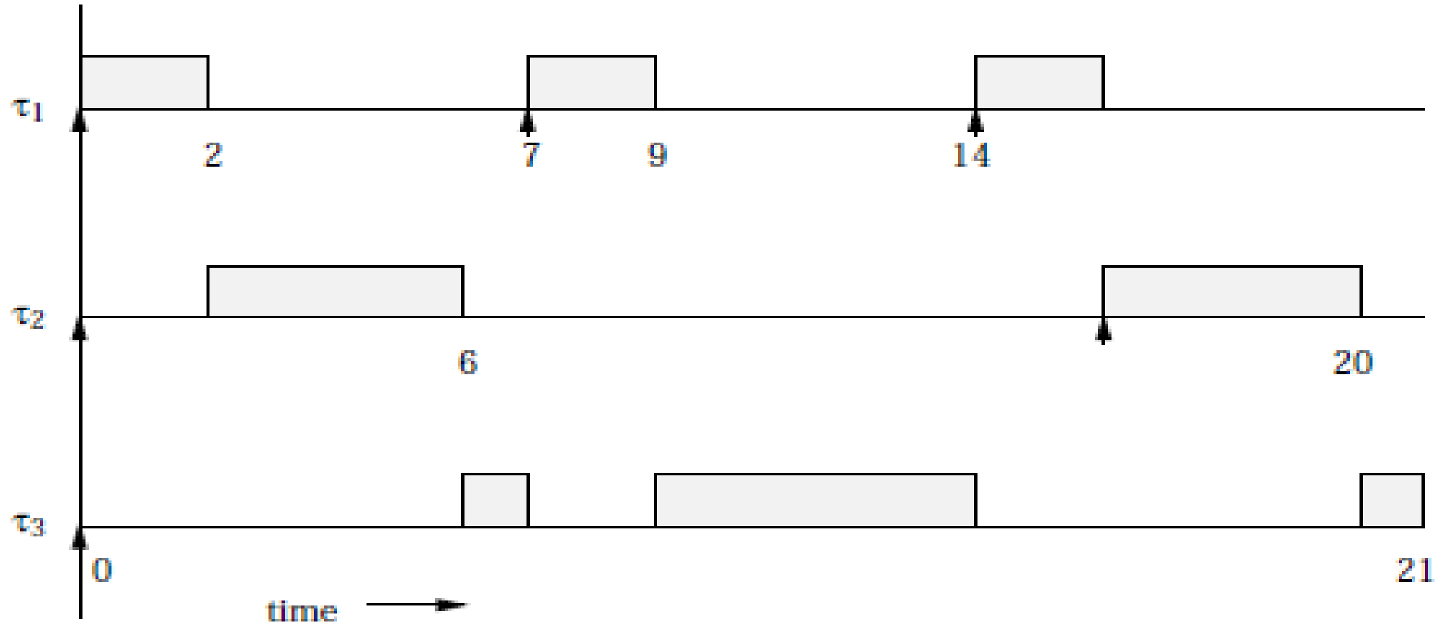
- Meeting the timing constraints of the system
- Preventing simultaneous access to shared resources and devices
- Attaining a high degree of utilization while satisfying the timing constraints of the system
- Reducing the cost of context switches caused by preemption

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)

Sample methods of analysis

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



Sample methods of analysis

- **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) = $\sum Ui \leq 1$

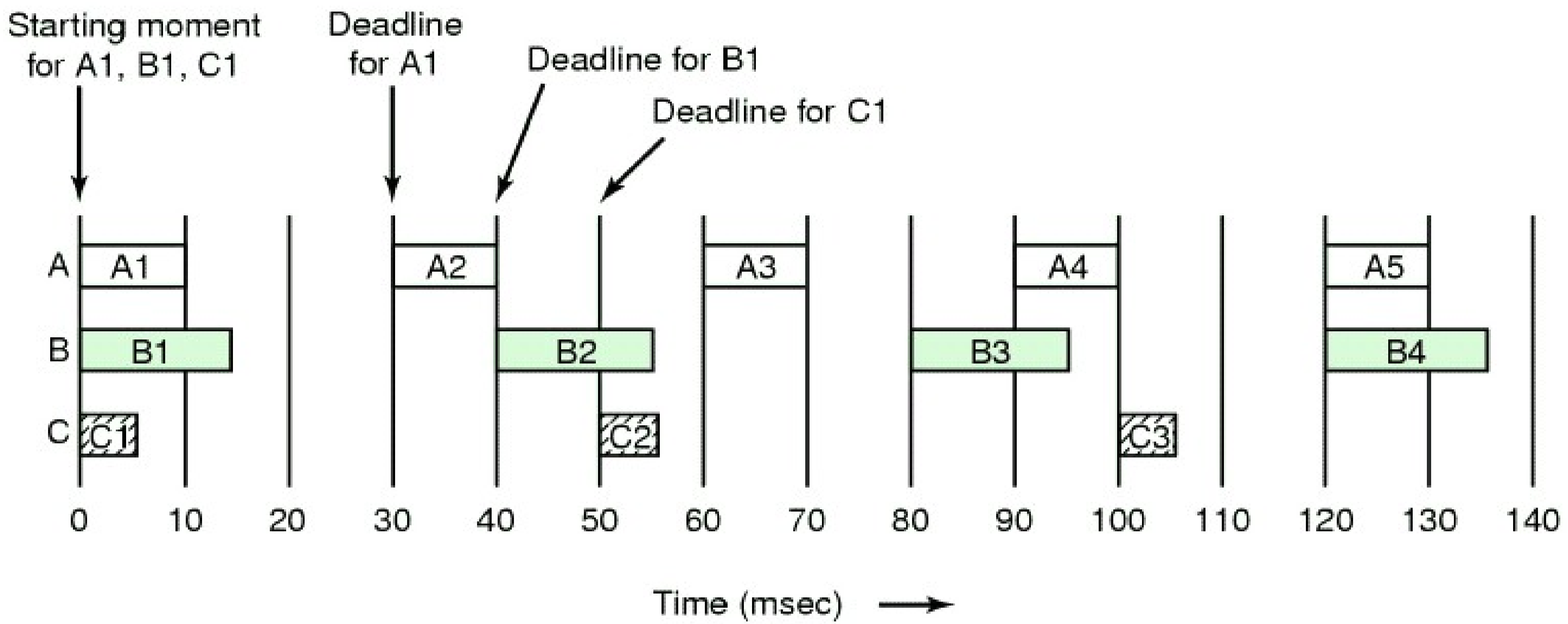
Utilization (%)	**Zone Type**	**Typical Application**
0-25	significant excess processing power – CPU may be more powerful than necessary	various
26-50	very safe	various
51-68	safe	various
69	theoretical limit	embedded systems
70-82	questionable	embedded systems
83-99	dangerous	embedded systems
100+	overload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

RM, EDF, LLF Scheduling

- **Rate Monotonic**
  - Static-priority preemptive scheme
  - Assumes that all tasks are periodic
  - The priorities are determined only by the period of the task.
- **Earliest Deadline First**
  - Dynamic-priority preemptive scheme
  - Higher priority is assigned to the task that has earlier deadline
  - Also called Deadline-Monotonic Scheduling
- **Least Laxity First**
  - Dynamic priority preemptive scheme
  - The laxity of a process is defined as the deadline minus remaining computation time.
  - Highest priority is given to the active job with the smallest 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 Example

- **Are the tasks schedulable ??**

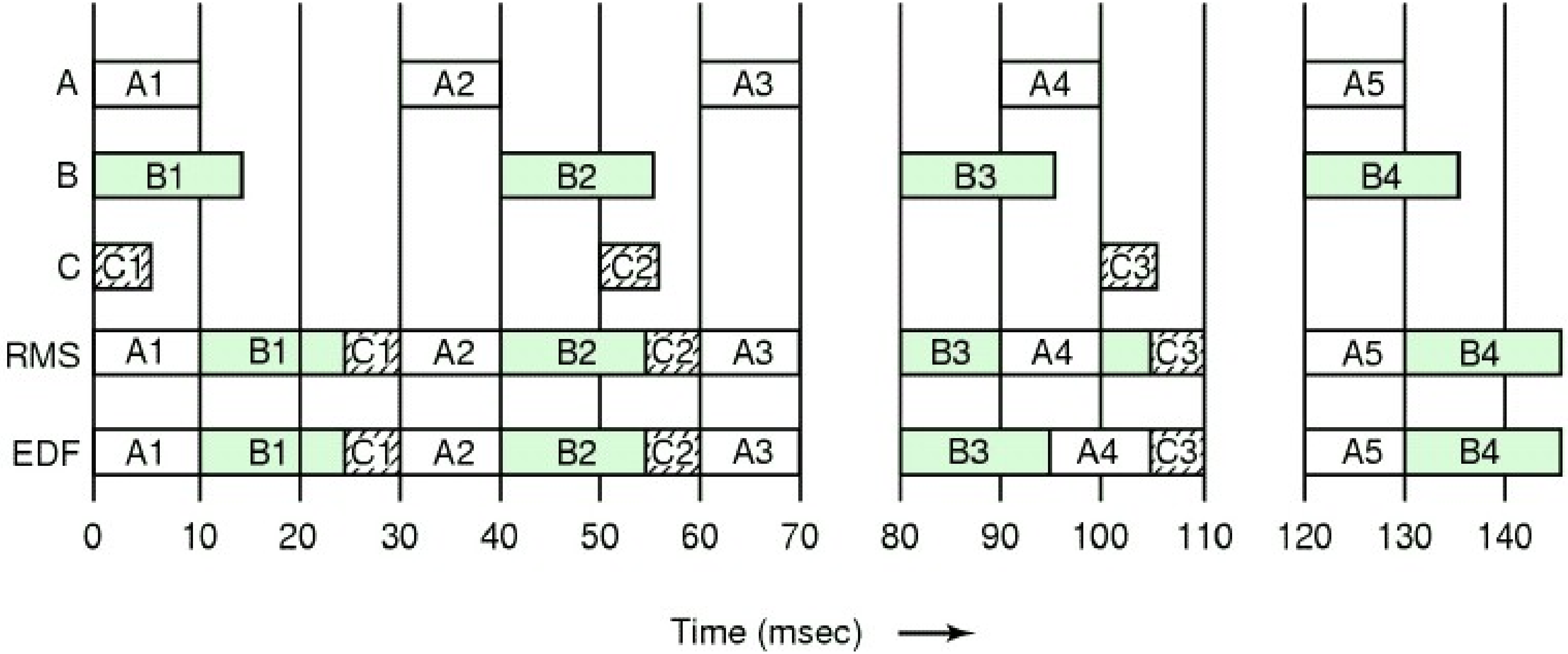
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

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

- YES

Scheduling with RM and EDF

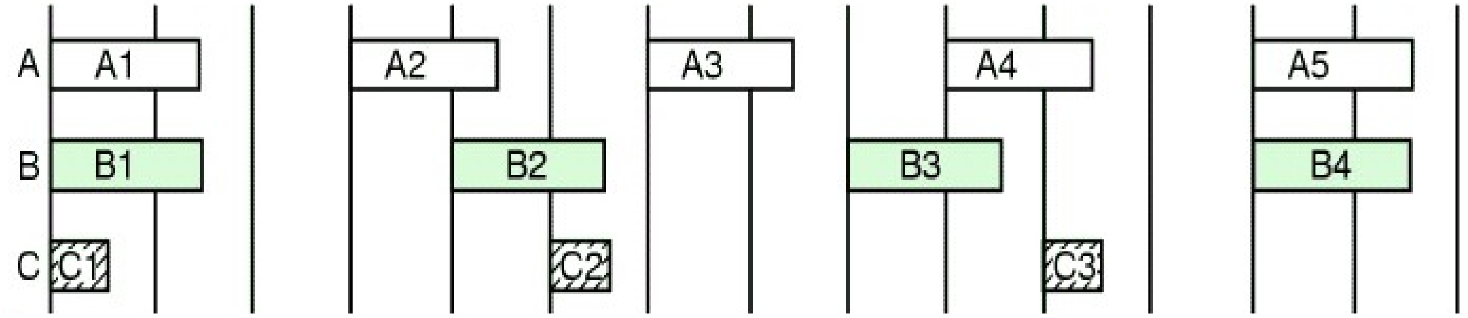
- **Analyze using timing diagram**



Modify Example

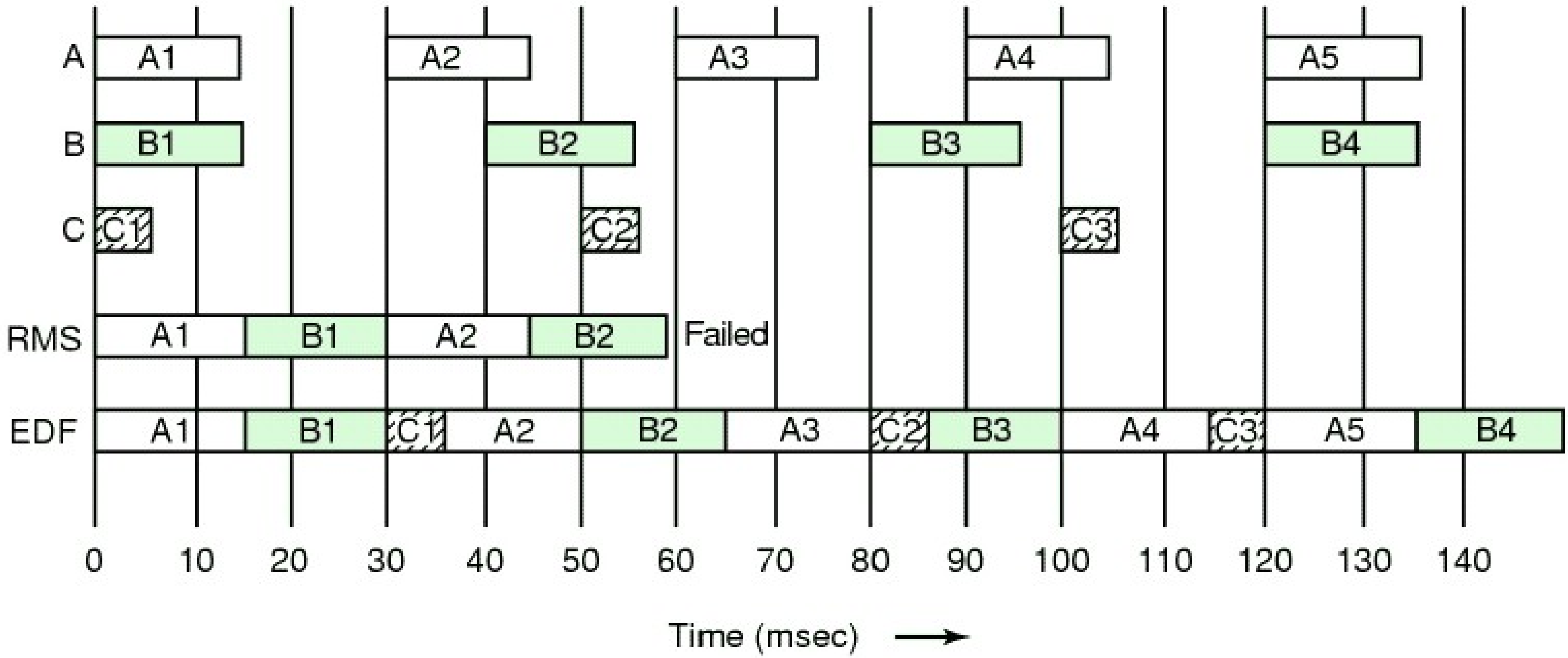
- **Increase A's CPU requirement to 15 milli sec**
- **The system is still schedulable**

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



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**

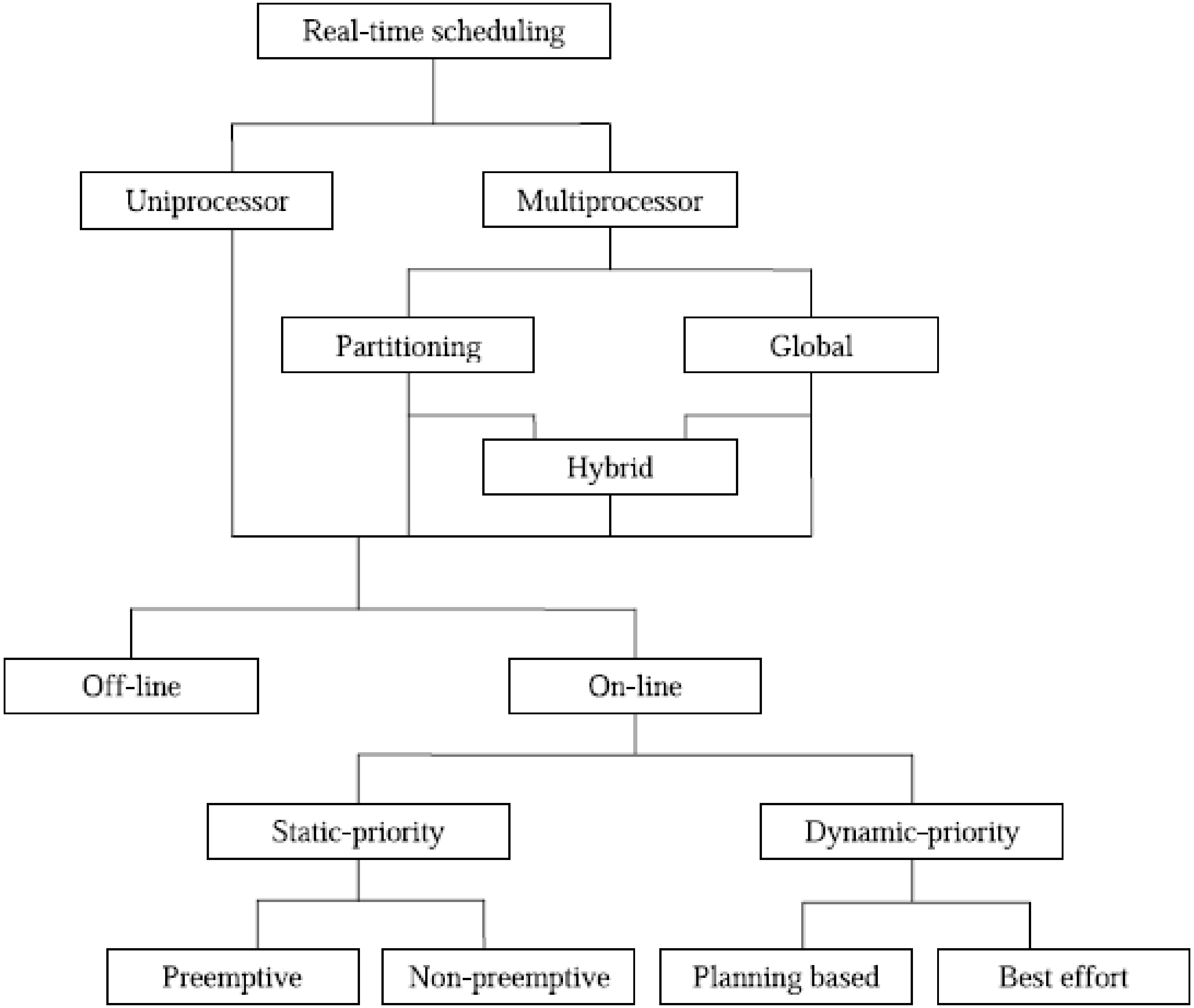
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$$

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

Performance Comparison

- **RM Scheduling 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



Off-line (Pre-run-time scheduling)

- **Generate scheduling information prior to system execution and utilize it during run-time.**
  - Uses precedence relations and prevents simultaneous access to shared resources
  - Reduces the cost of context switches by choosing algorithms that do not result in a large number of preemptions, such as the EDF algorithm
- **Suitable for applications where all characteristics are known prior and change very infrequently**
  - Ready times, execution times, deadlines are known and environment is completely predictable
- **Pros -** Significant reduction in run-time resources, including processing time for scheduling
- **Cons -** Any change requires re-computing the entire schedule

Online Scheduling

- **Generate scheduling information while the system is running**
- **Pros**
  - No requirement to know tasks characteristics in advance
  - Flexible and easily adaptable to environment changes
- **Cons**
  - Require a large amount of run-time processing time.

On-line Scheduling
Static-priority based algorithms

- **Work well with fixed periodic tasks but do not handle aperiodic tasks particularly well**
- **Preemptive**
  - RM Scheduling
- **Non-preemptive**
  - Used when the execution order is known prior and each task completes before another task starts
  - Avoids the overhead associated with multiple context switches per task

Online Scheduling
Dynamic-priority based algorithms

- **Require a large amount of on-line resources, but are flexible**
- **Better response to aperiodic tasks or soft tasks while still meeting the timing constraints of the hard periodic tasks**
- **Two subsets:**
  - Planning based - feasibility check at run-time, schedule produced. Eg: EDF, LLF
  - Best effort - No feasibility check, attempts to meet deadlines but no guarantee.

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 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 Aperiodic Tasks

- **Methods**
  - A Background Server executes at low priority, and makes use of CPU idle time to schedule aperiodic tasks. No guarantee of service
  - The Polling Server executes as a high-priority periodic task, and every cycle checks if an event needs to be processed
  - Priority Exchange and Deferred Servers
    - A periodic server(task) with highest priority is added. When there are no aperiodic tasks to service, exchange its priority with the next highest priority task, to allow it to execute

Real Time Operating Systems

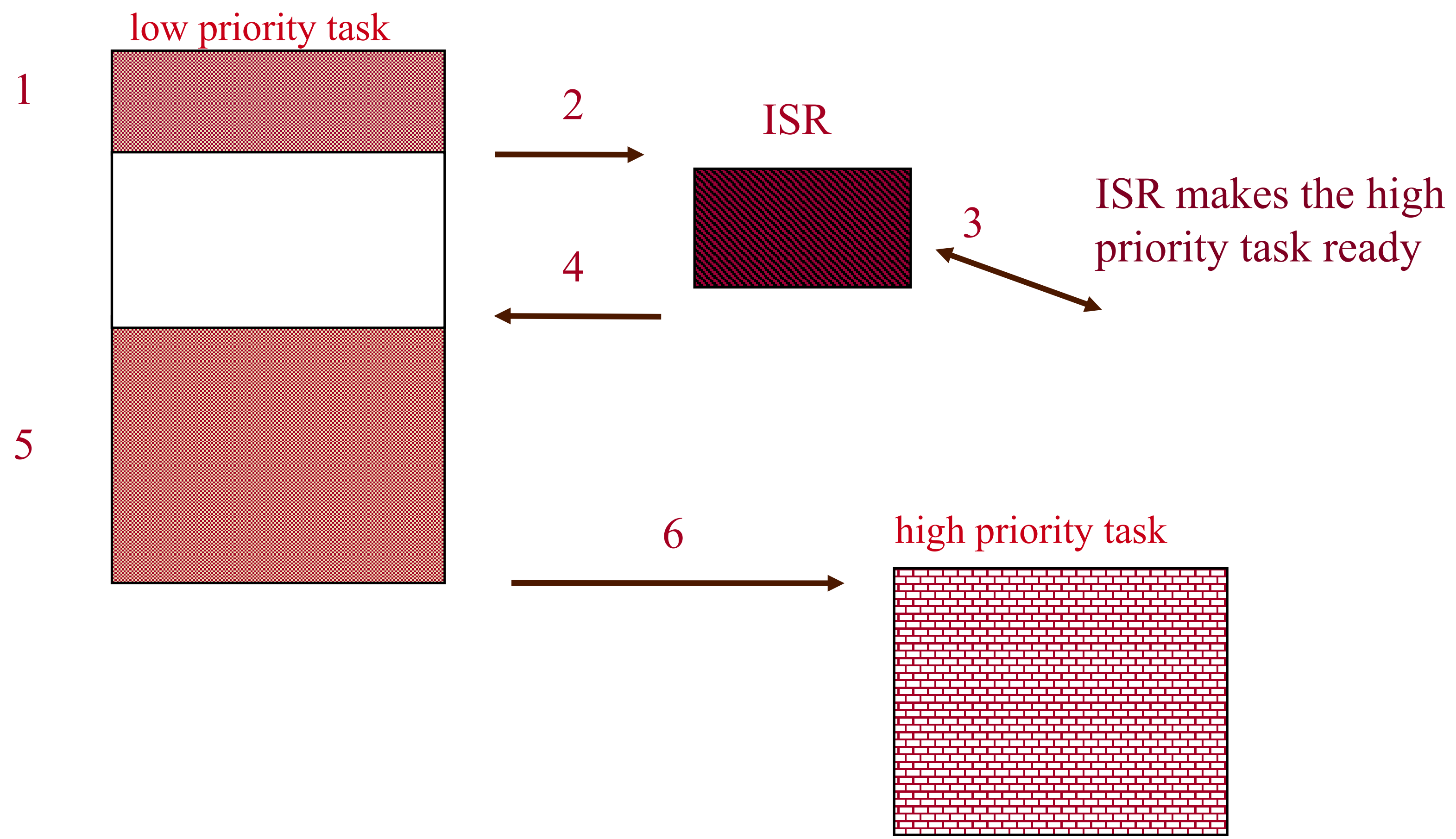
Goals of an Operating System

- **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

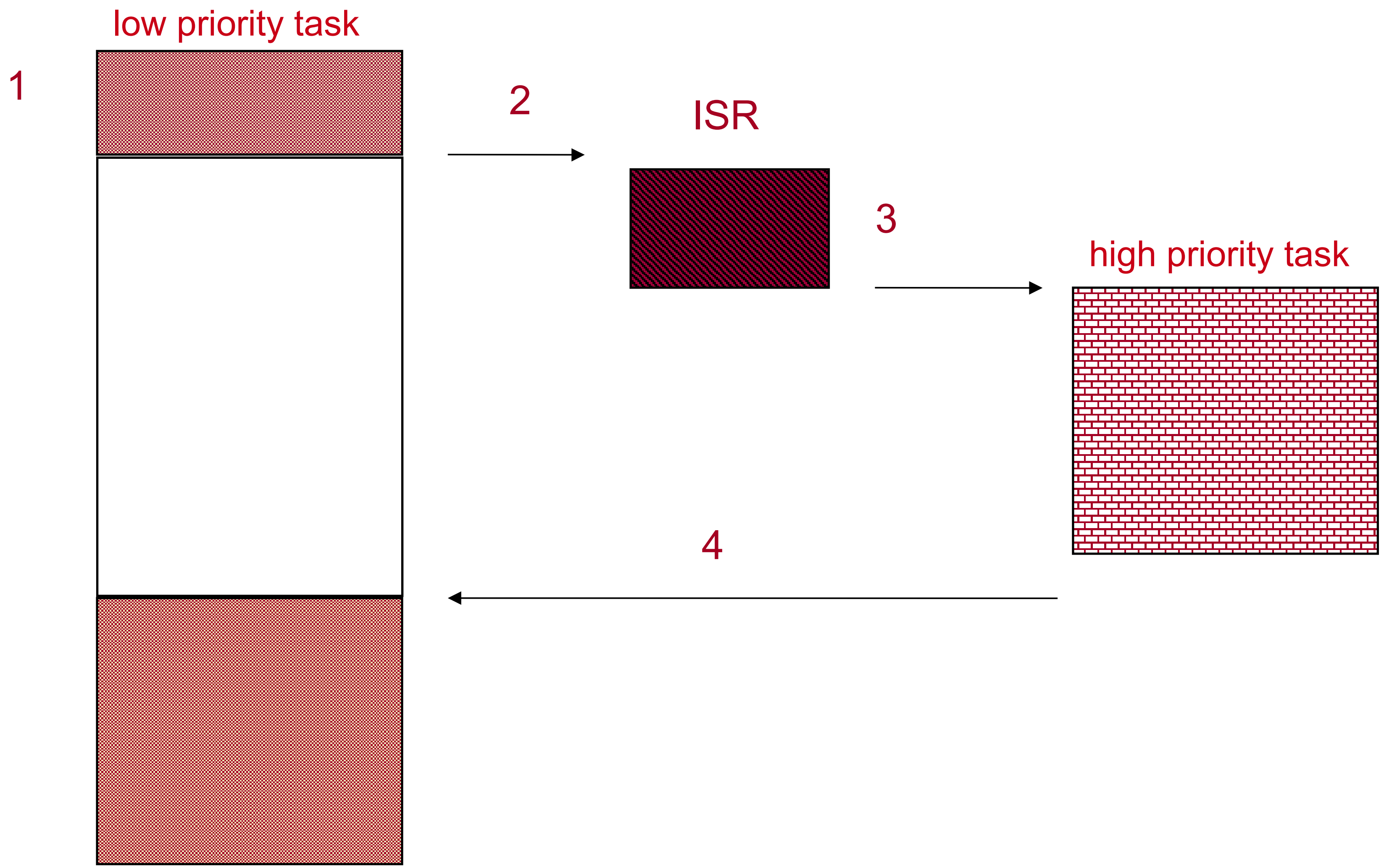
RTOS Vs GPOS

#	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

Kernel Types – Non Preemptive Kernel



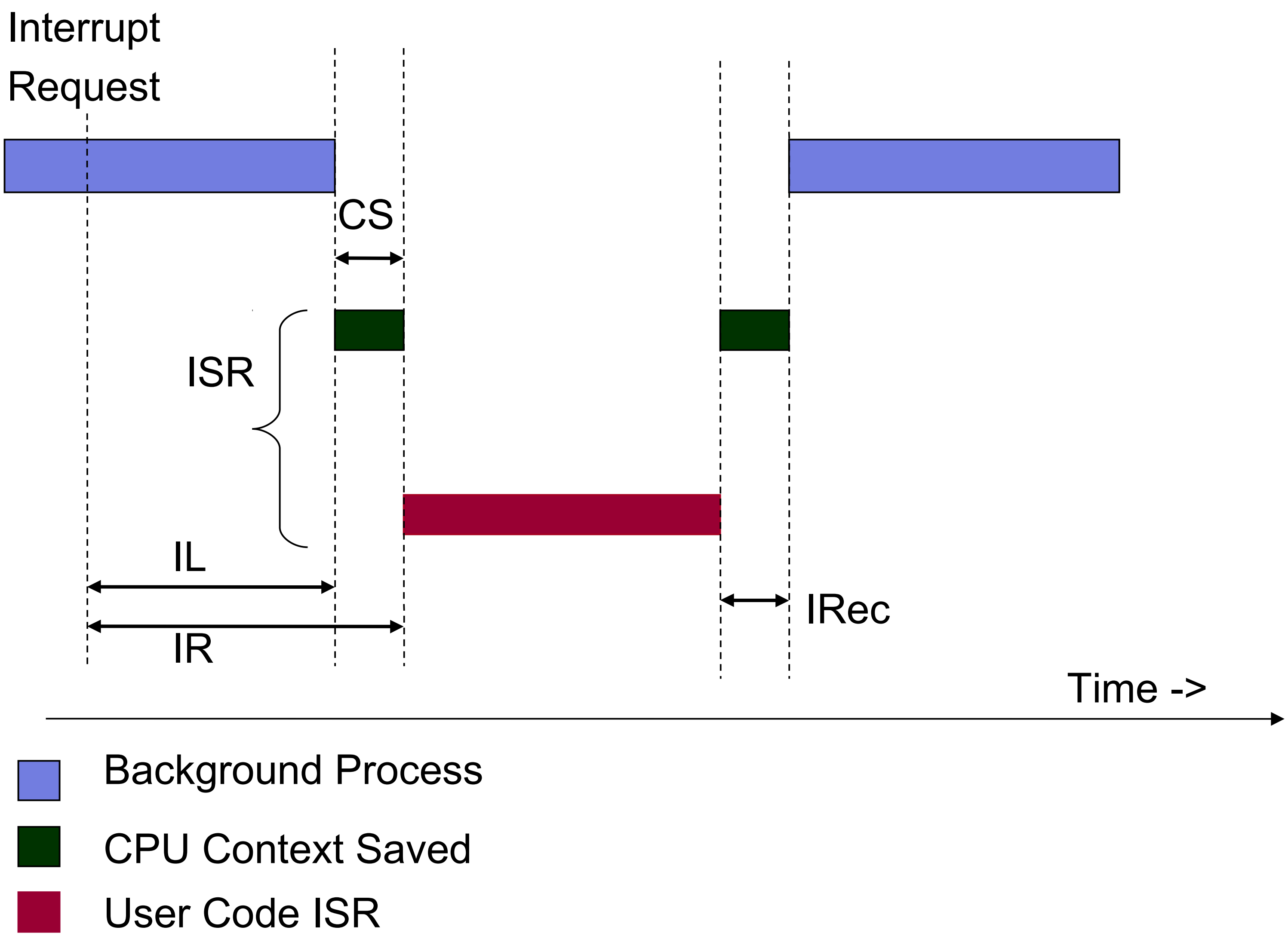
Kernel Types – Preemptive Kernel



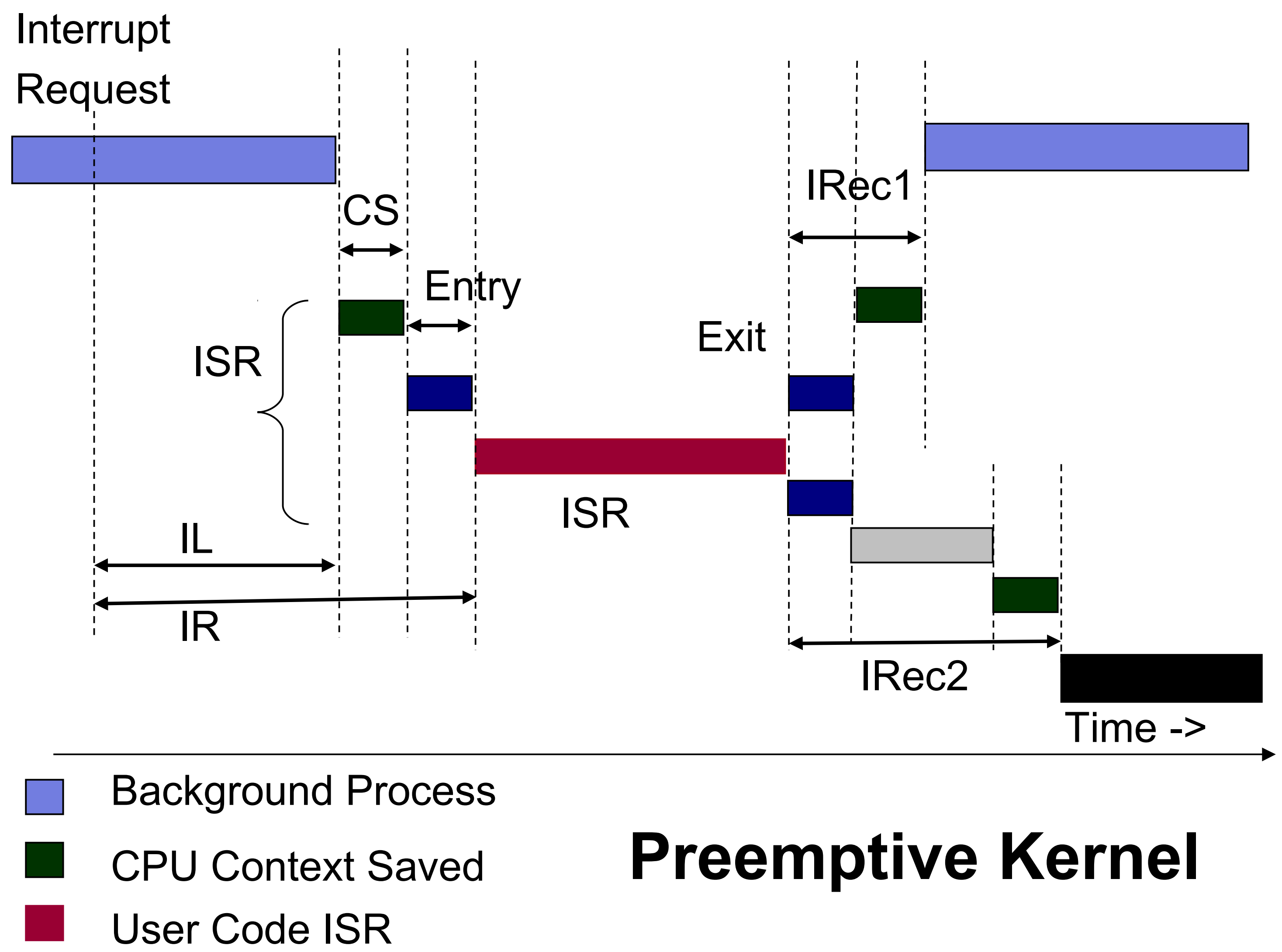
Interrupts

- **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

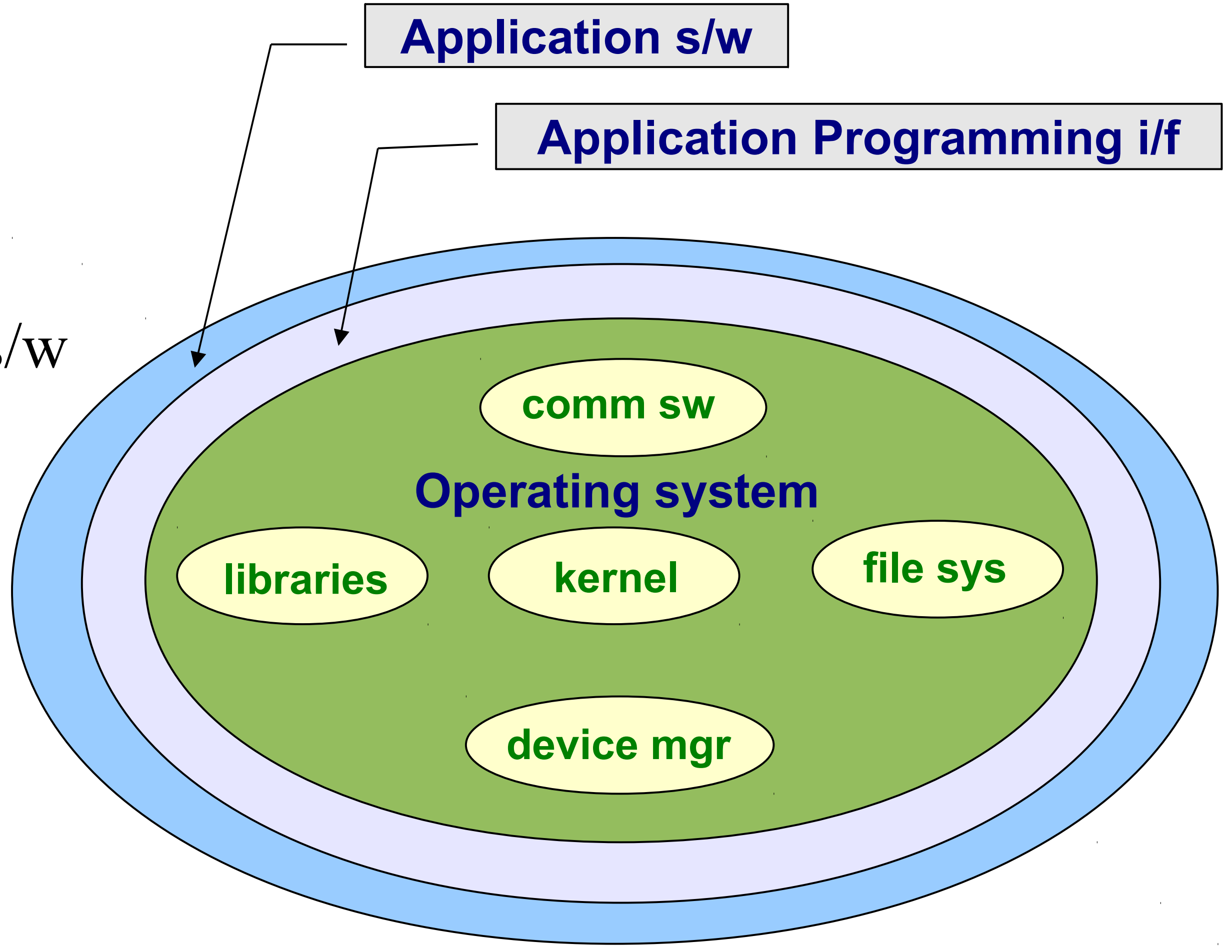


Interrupts in a preemptive kernel



RTOS Architecture

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



RTOS Services

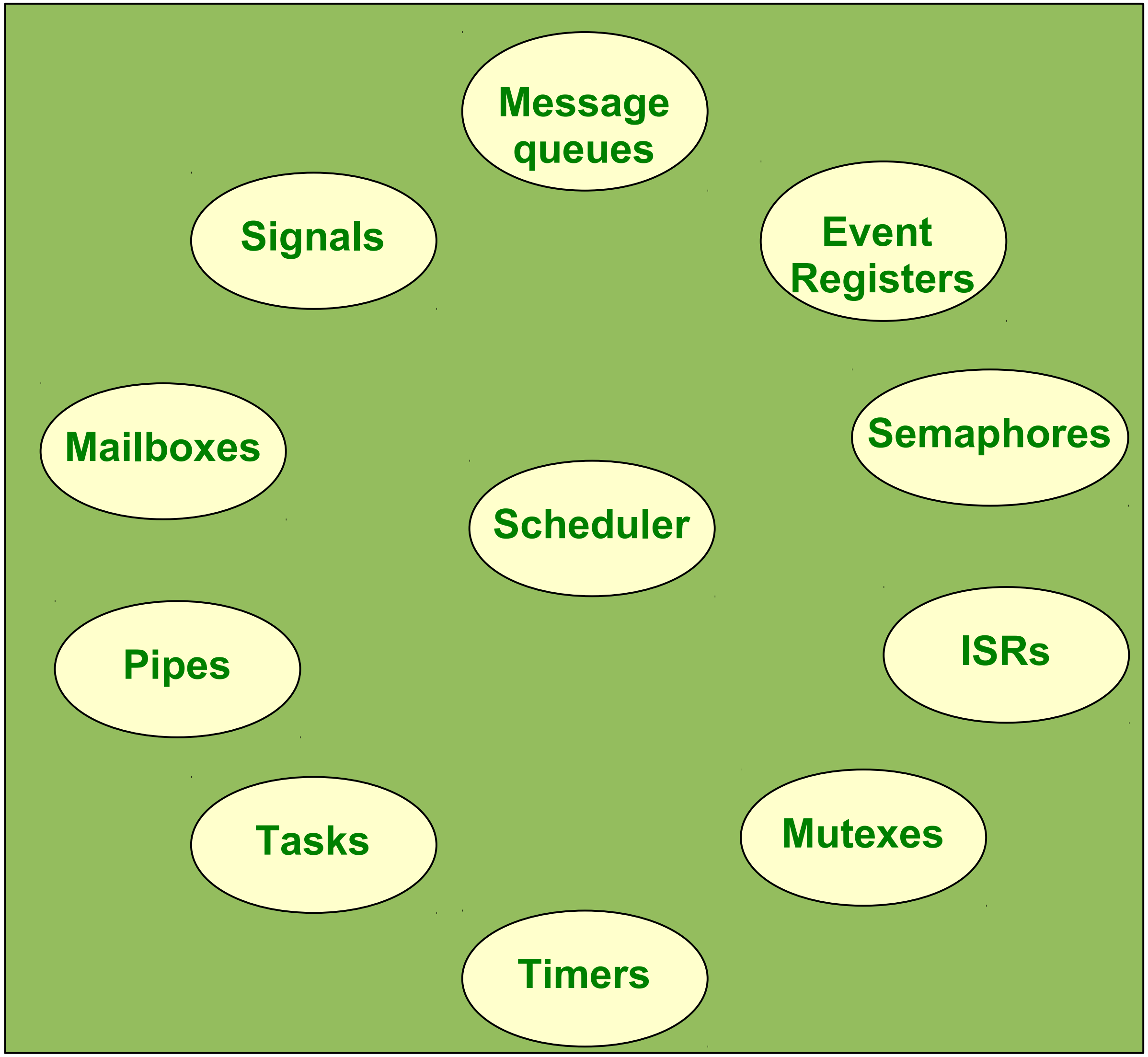
- **Multitasking, Task & Thread Management**
  - Creation and scheduling of tasks.
  - Priority based preemptive scheduling
- **Vectored Interrupt Service Routines**
- **Inter Task Synchronization and Inter Task Communications**
  - Mutual exclusions, signals, messages, shared memory, etc
- **Timer Services**
  - 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

Kernel Objects

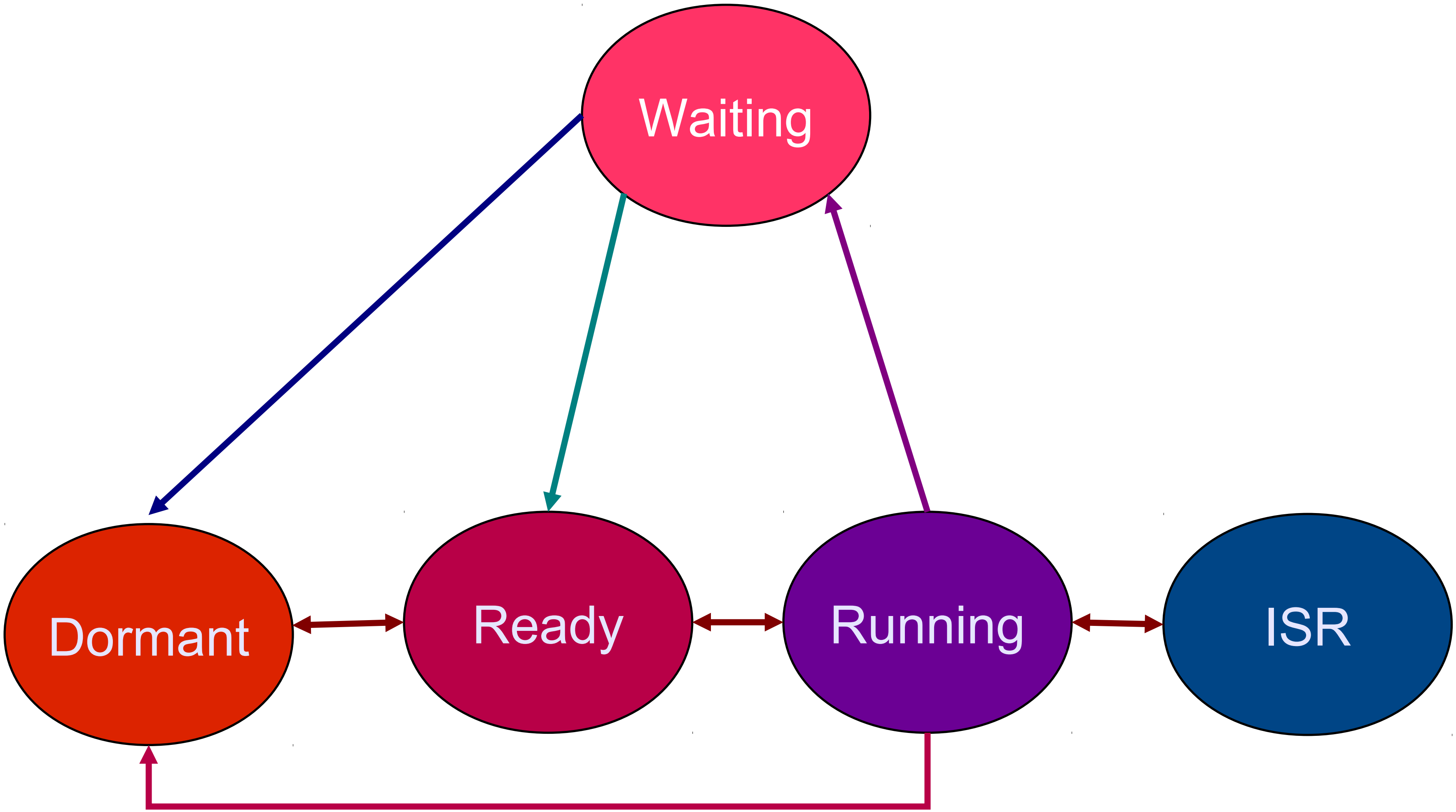
- Scheduler
- Tasks
- Interrupt Service Routines
- Semaphores
- Mutexes
- Mail boxes
- Message queues
- Pipes
- Event registers
- Signals
- Timers



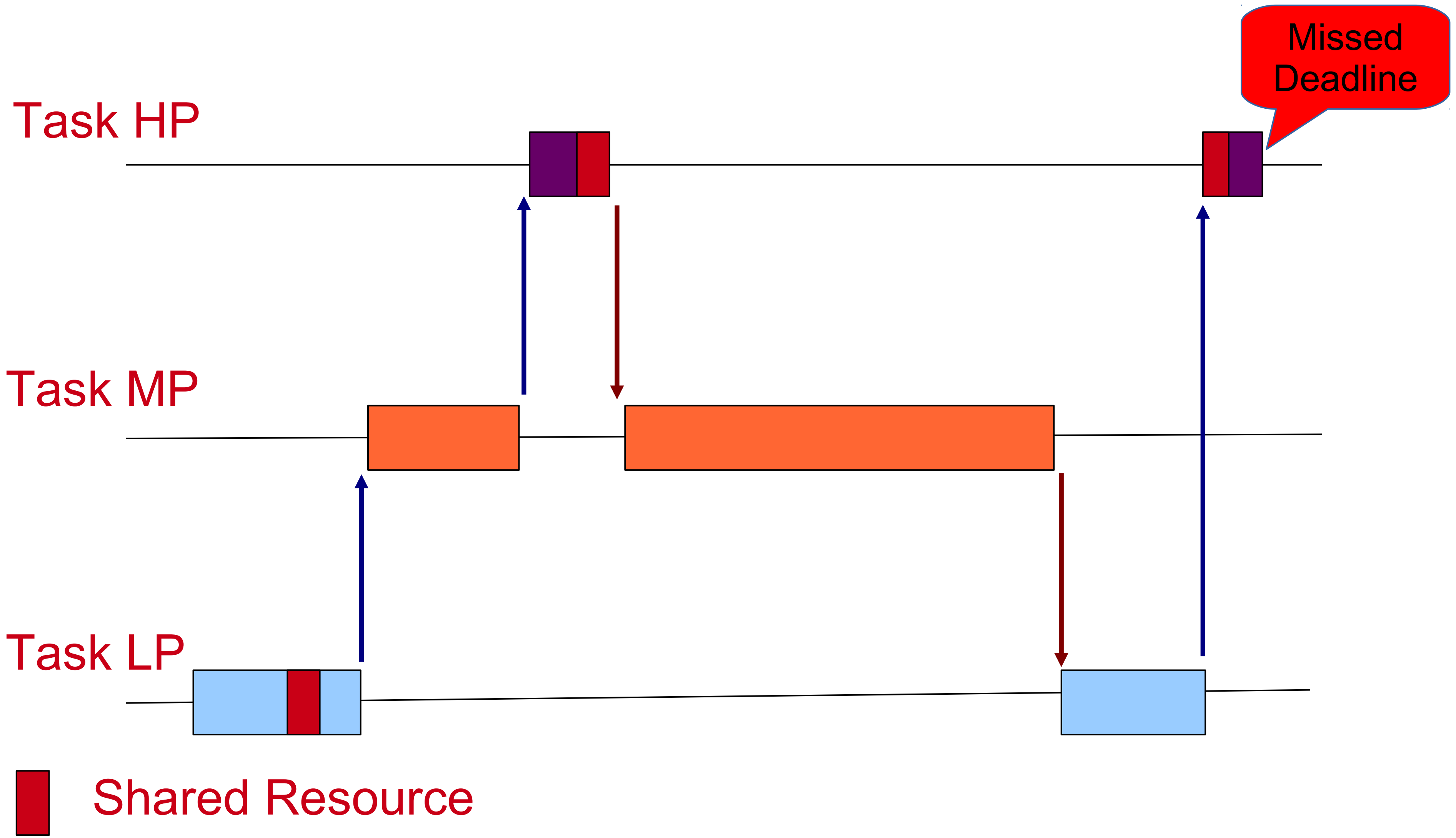
Real Time Tasks

- **An RT application splits the work to be done 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, Delay, Stack Pointer
  - **The Task Stack** – Maintains context information of the task during switches

Task States



Priority Inversion, Inheritance and Ceiling



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

Small Fast Proprietary Kernels

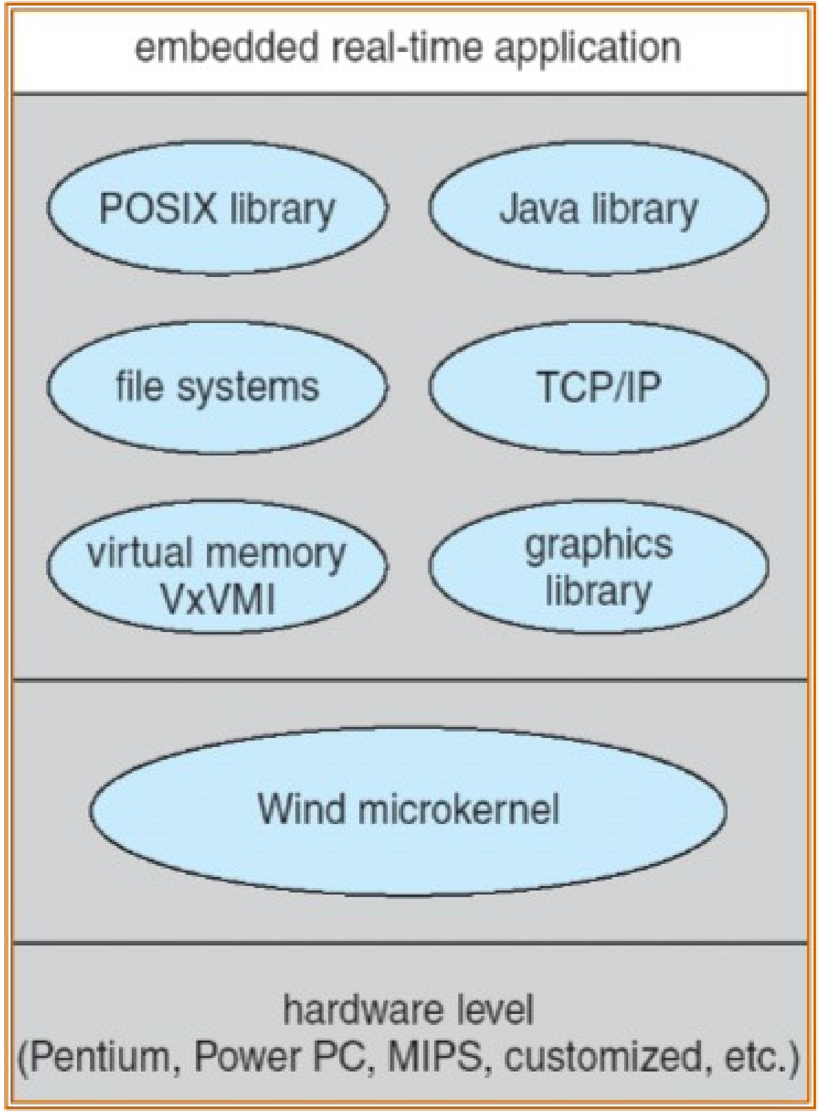
- **Stripped down and optimized versions of time-sharing operating systems.**
- **Intended to be fast**
  - a fast context switch
  - external interrupts recognized quickly (low interrupt latency)
  - special sequential files that can accumulate data at a fast rate
- **To deal with timing requirements**
  - a real-time clock with special alarms and timeouts
  - bounded execution time for most primitives (determinism)
  - real-time queuing disciplines such as earliest deadline first
  - primitives to delay/suspend/resume execution
  - priority-driven best-effort or a table-driven scheduling mechanism
- **Communication and synchronization -** mailboxes, events, signals, and semaphores

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



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

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

Variants of Real-Time Linux Extensions

- **Real Time Linux (RTLinux)**
  - 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 latency - 15us
  - Used to control robots, data acquisition systems, manufacturing plants, and other time-sensitive instruments and machines

- **Real Time Application Interface (RTAI)**
  - 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, and RPCs
  - POSIX 1003.1c & POSIX 1003.1b(pqueues)
  - Avg interrupt response - 20us

- **Xenomai Framework**
  - Implementing and migrating real time applications, based on standard APIs or emulators of proprietary RTOS interfaces, such as VxWorks and pSOS
  - Linux-hosted dual kernel, with pure Adeos. User space RT tasks
  - Platfroms - x86, ARM, POWER, IA-64, Blackfin, nios