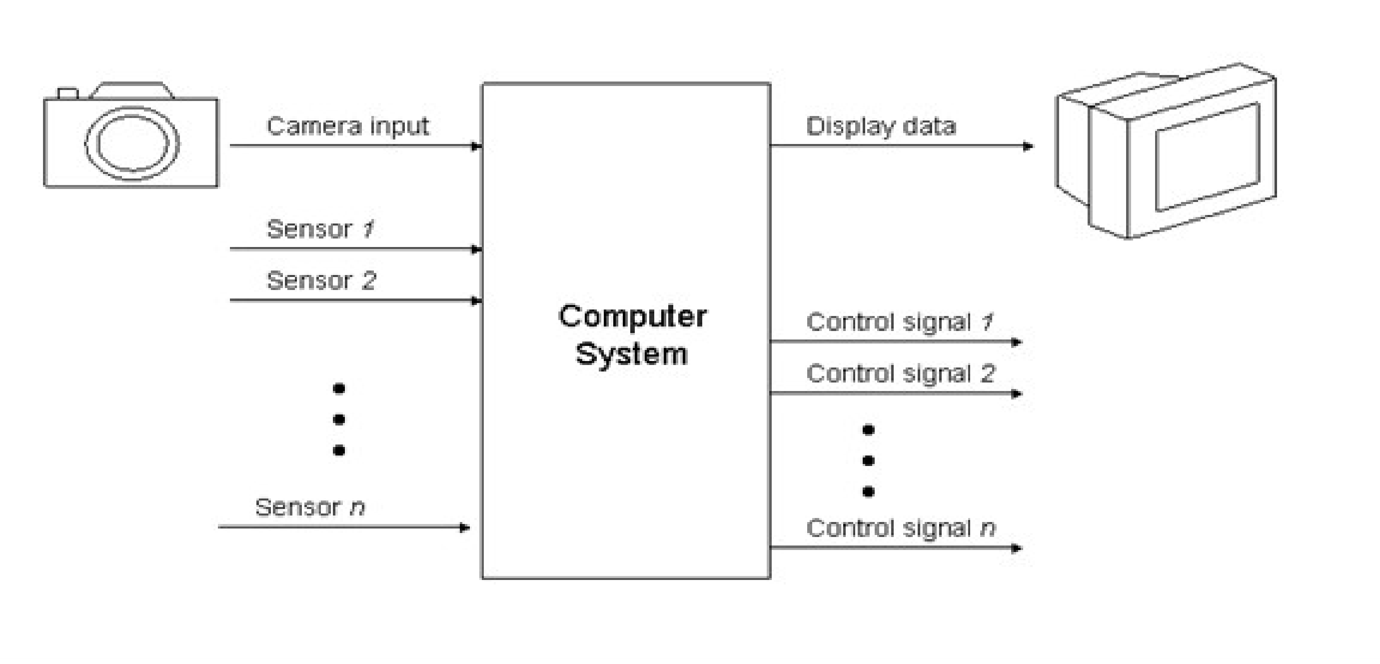
RTOS Concepts
&
RTAI

Sowjanya
C-DAC, Hyderabad

Agenda

- Real Time Systems
- RTOS Vs GPOS
- RTOS Services
- Real Time Linux Variants
- Real Time Application Interface(RTAI)

Real Time Systems



- Typical real-time control system including inputs from sensors and imaging devices and producing control signals and display information.

Real Time Systems

- **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 Systems

- Classification of RT systems
  - **Soft real-time system**
    - Performance is degraded but not destroyed by failure to meet response-time constraints.
    - Eg: Mutlimedia, Interactive video games
  - **Firm real-time system**
    - Missing more than few deadlines, may lead to catastrophe
    - Eg: Robot weed killer
  - **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

Real Time Tasks

- **Periodic tasks**
  - Time-driven, characteristics are known a priori (pi, ci)
  - Eg: Task monitoring temperature of a patient in an ICU
- **Aperiodic tasks**
  - Event-driven, characteristics not known a priori (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

Real Time Tasks

- The CPU utilization or time-loading factor, is a measure of the percentage of non-idle processing
  - Utilization factor u_i for a Task t_i is (c_i / p_i)
  - Overall system utilization (U) = $\sum u_i$

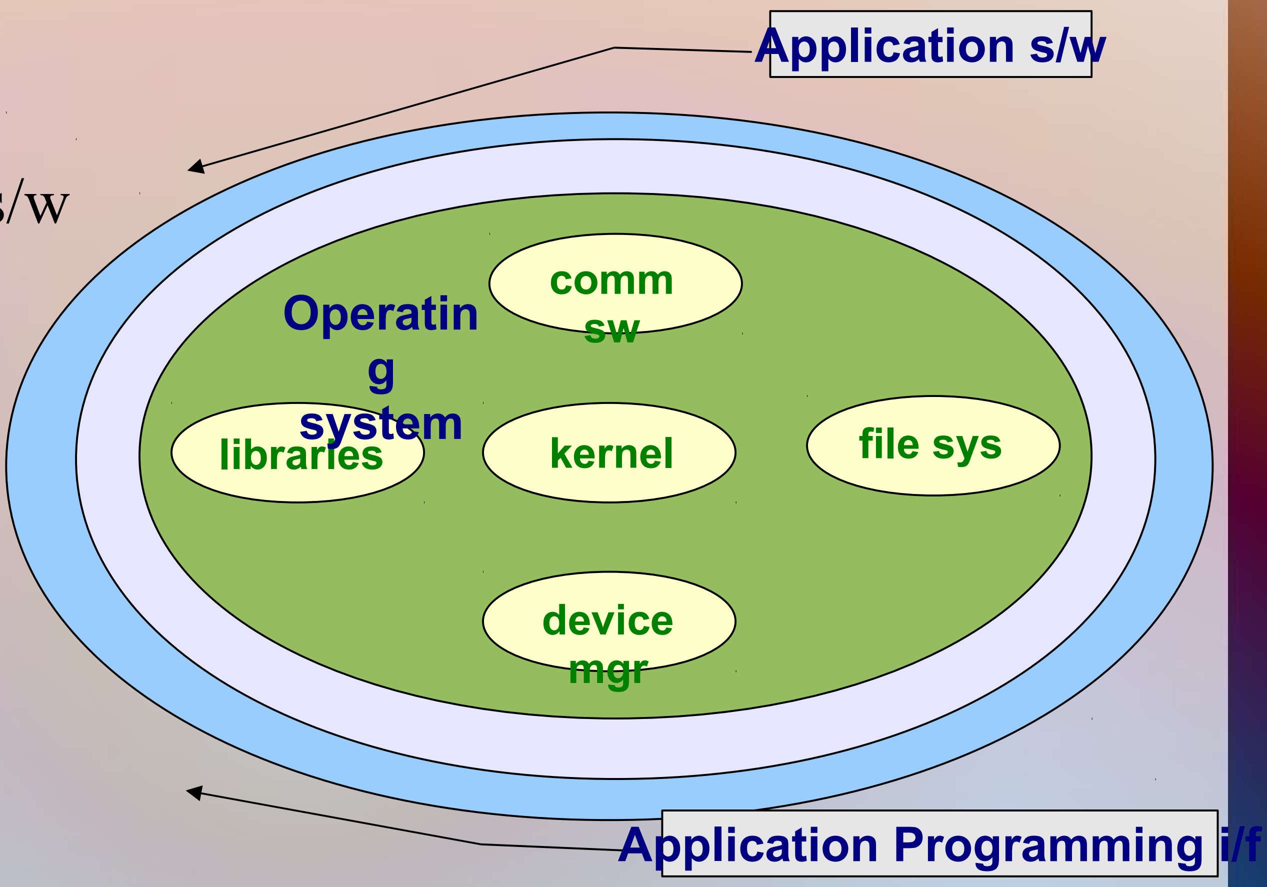
Utilization (%)	**Zone Type**	**Typical Application**
0-25	significant excess processing power – CPU may be more powerful than necessary	various
26-50	very safe	various
51-68	safe	various
69	theoretical limit	embedded systems
70-82	questionable	embedded systems
83-99	dangerous	embedded systems
100+	overload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

RTOS Vs GPOS

#	Metric of Evaluation	General Purpose OS	Real Time OS
1	Determinism	Non Deterministic	Deterministic – Kernel functions should execute in a fixed amount of time
2	Load Independent Timing	Not Applicable – Response becomes sluggish as number of tasks increase	Remains Constant – Generally Priority based execution. Time Slices are available only among tasks that hold the same priority
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	Priority based Preemptive Scheduling – 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 performace metrics

RTOS Architecture

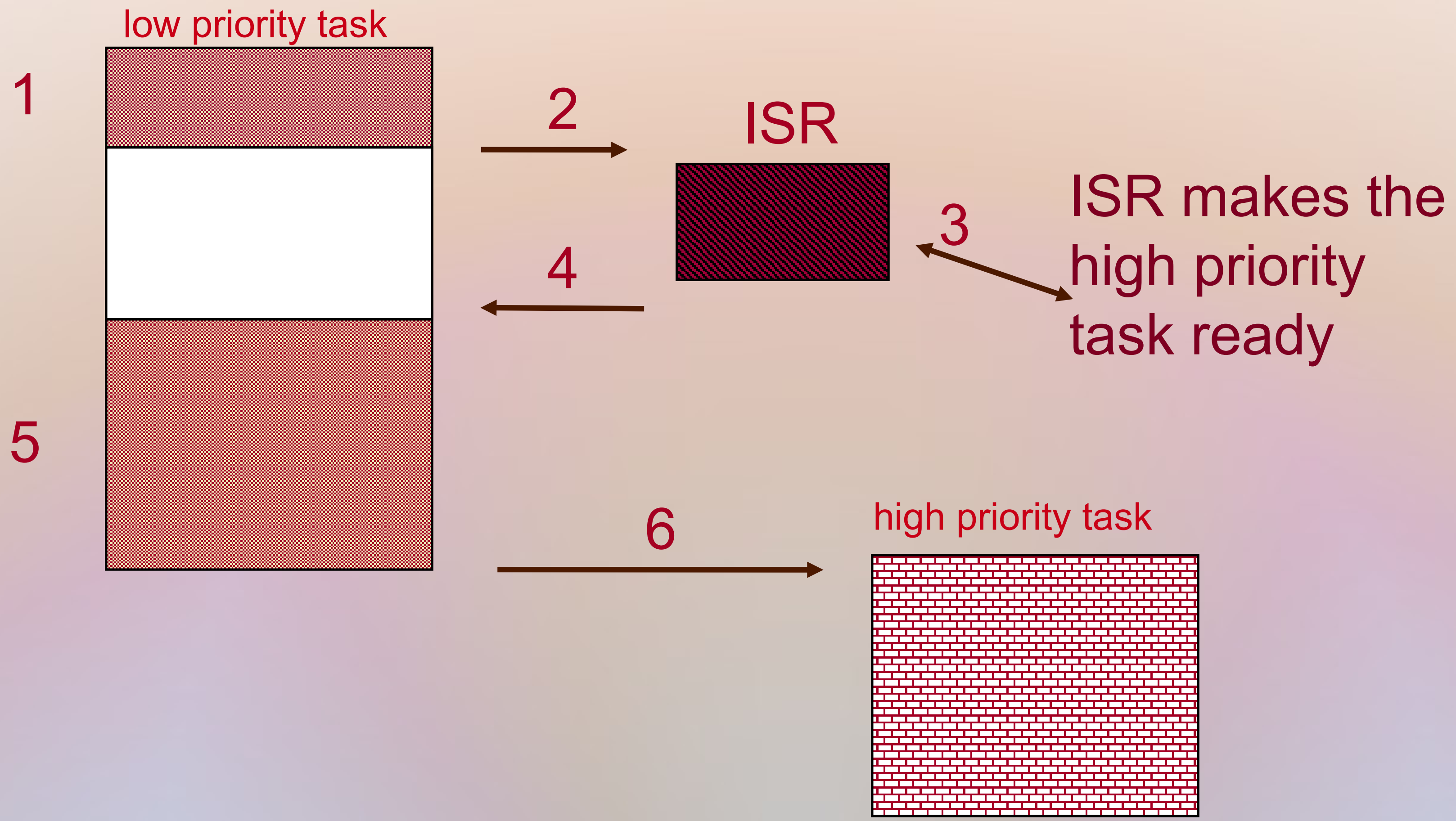
- An RTOS consists of
  - Kernel (micro-kernel)
  - Device Manager
  - Networking protocol s/w
  - Libraries
  - File sys (optional)



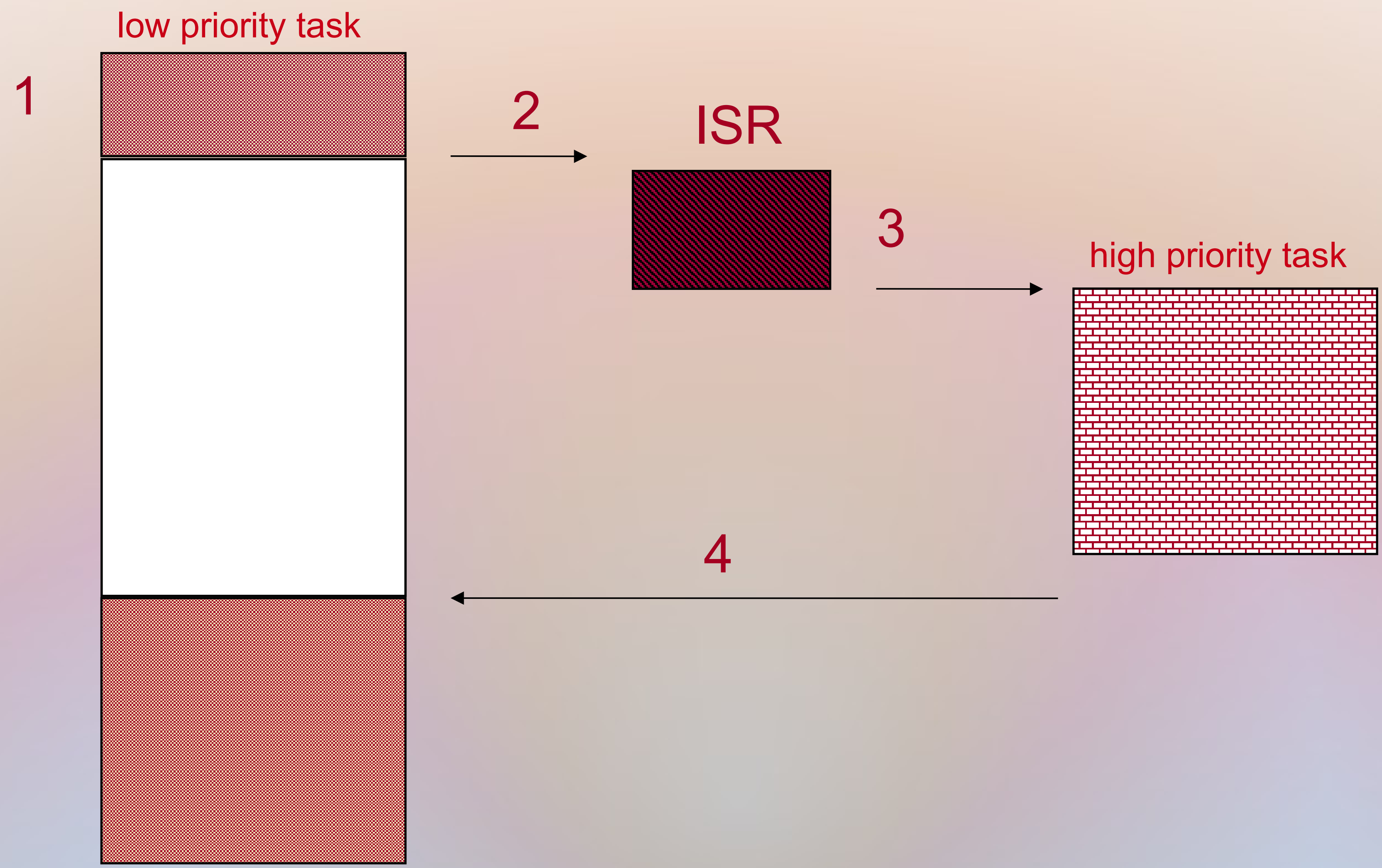
RTOS Services

- Multi-tasking, Task & Thread Management
  - Creation and scheduling of tasks.
  - Priority based pre-emptive scheduling
- Inter Task Synchronization and Inter Task Communications through mutual exclusions, signals, messages, shared memory, etc
- Usually no Memory Management
- Timer Services such as periodic and aperiodic interrupts

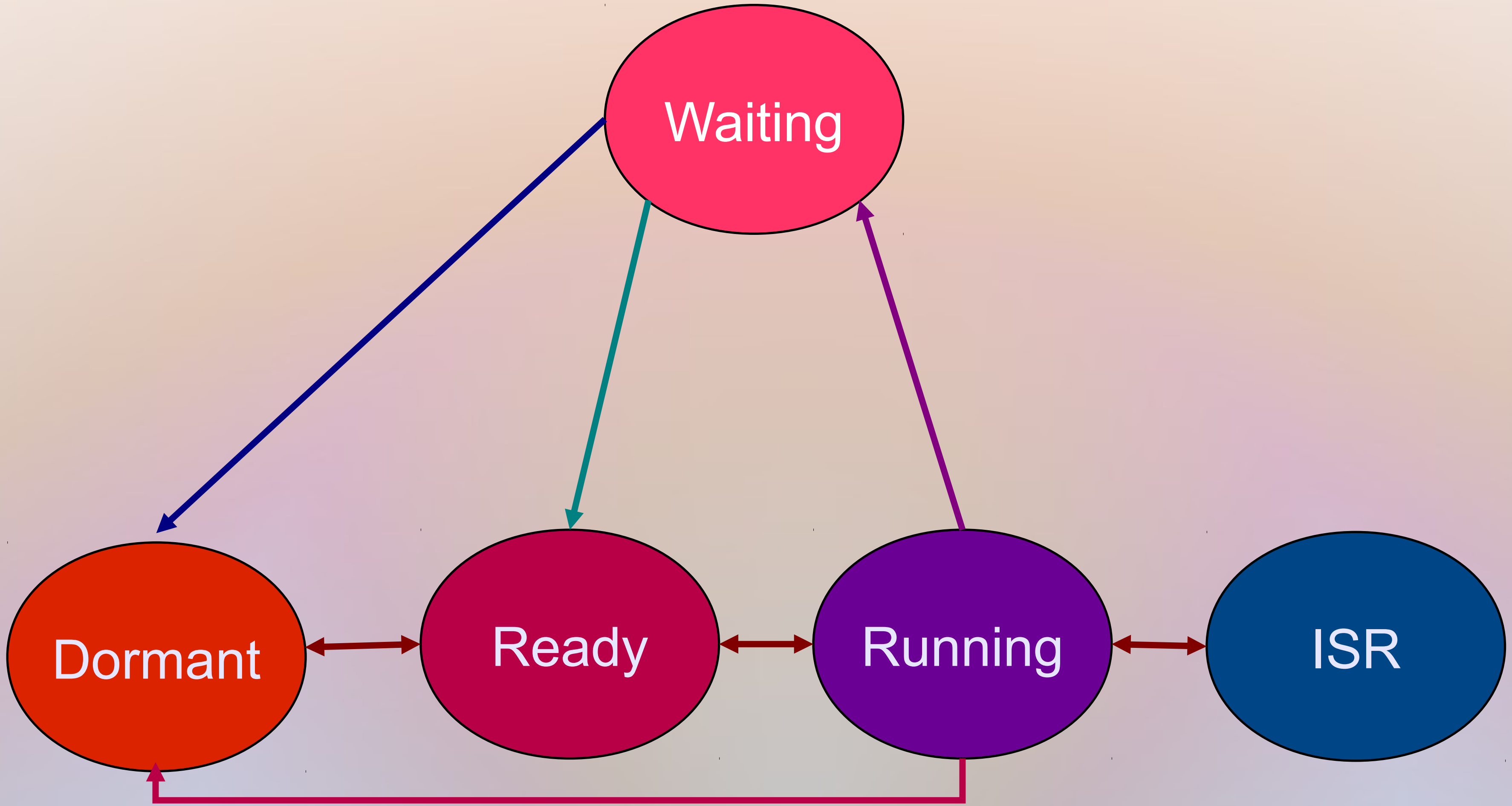
Kernel Types – Non Preemptive Kernel



Kernel Types – Preemptive Kernel



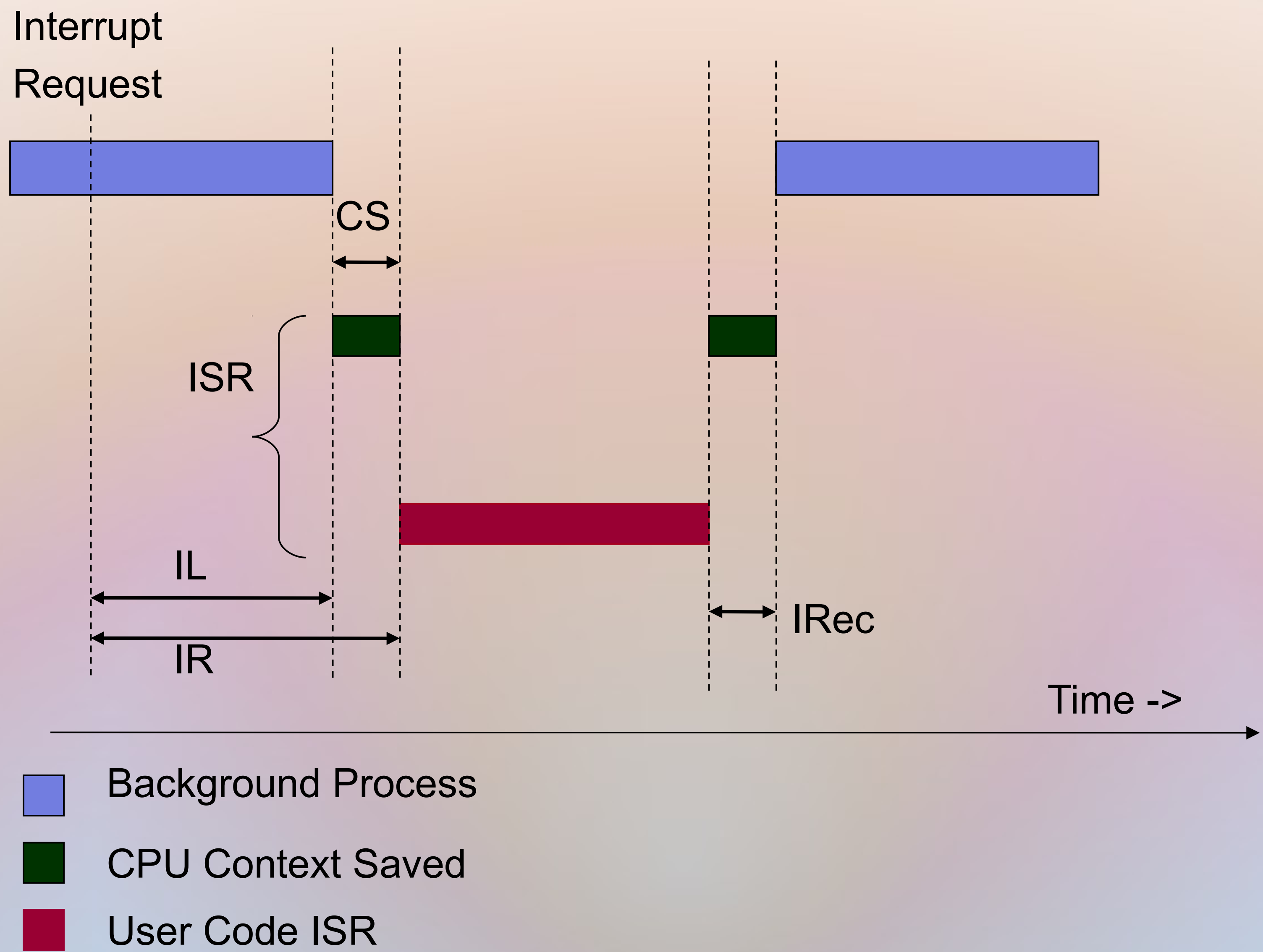
Task States



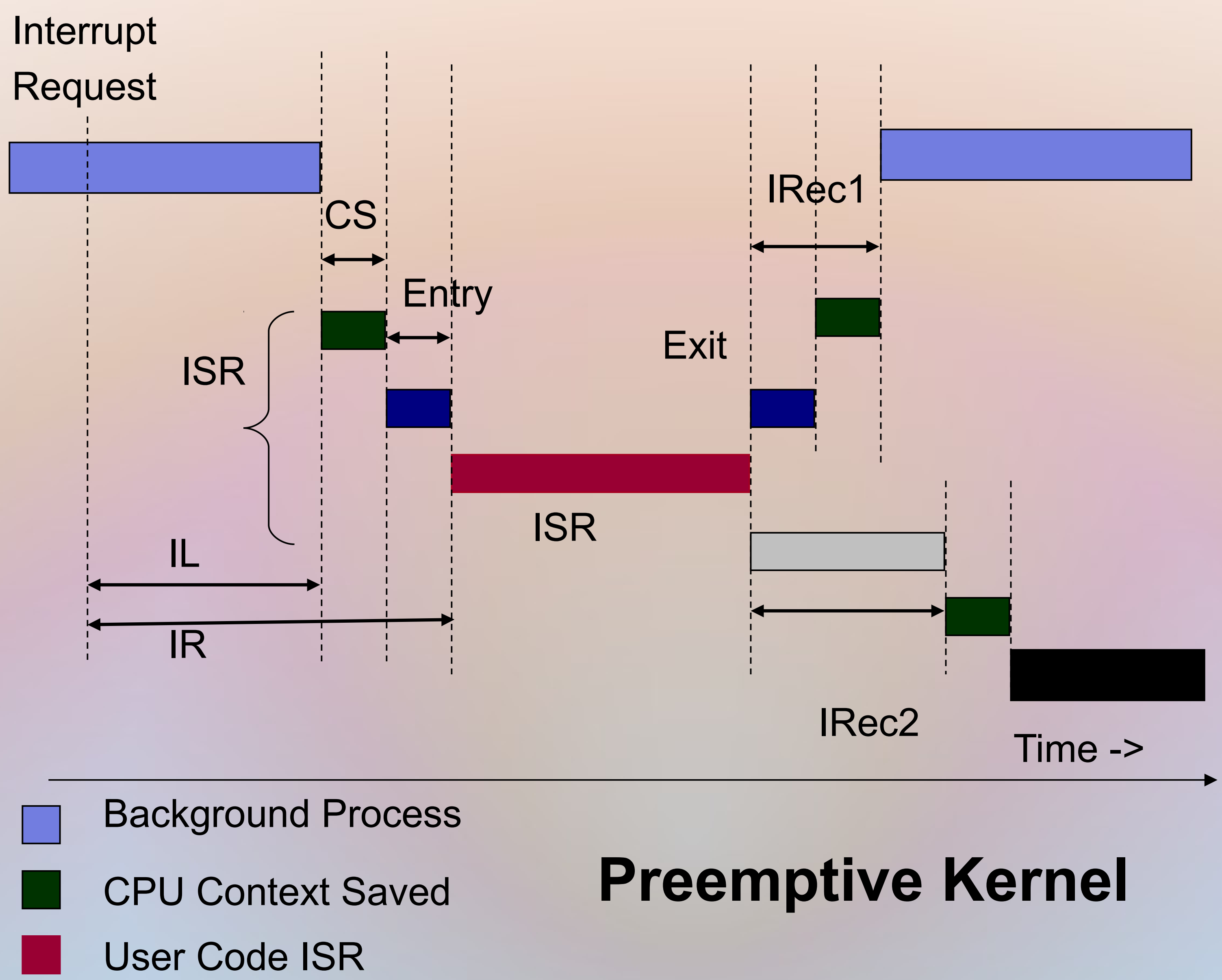
Interrupts

- 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

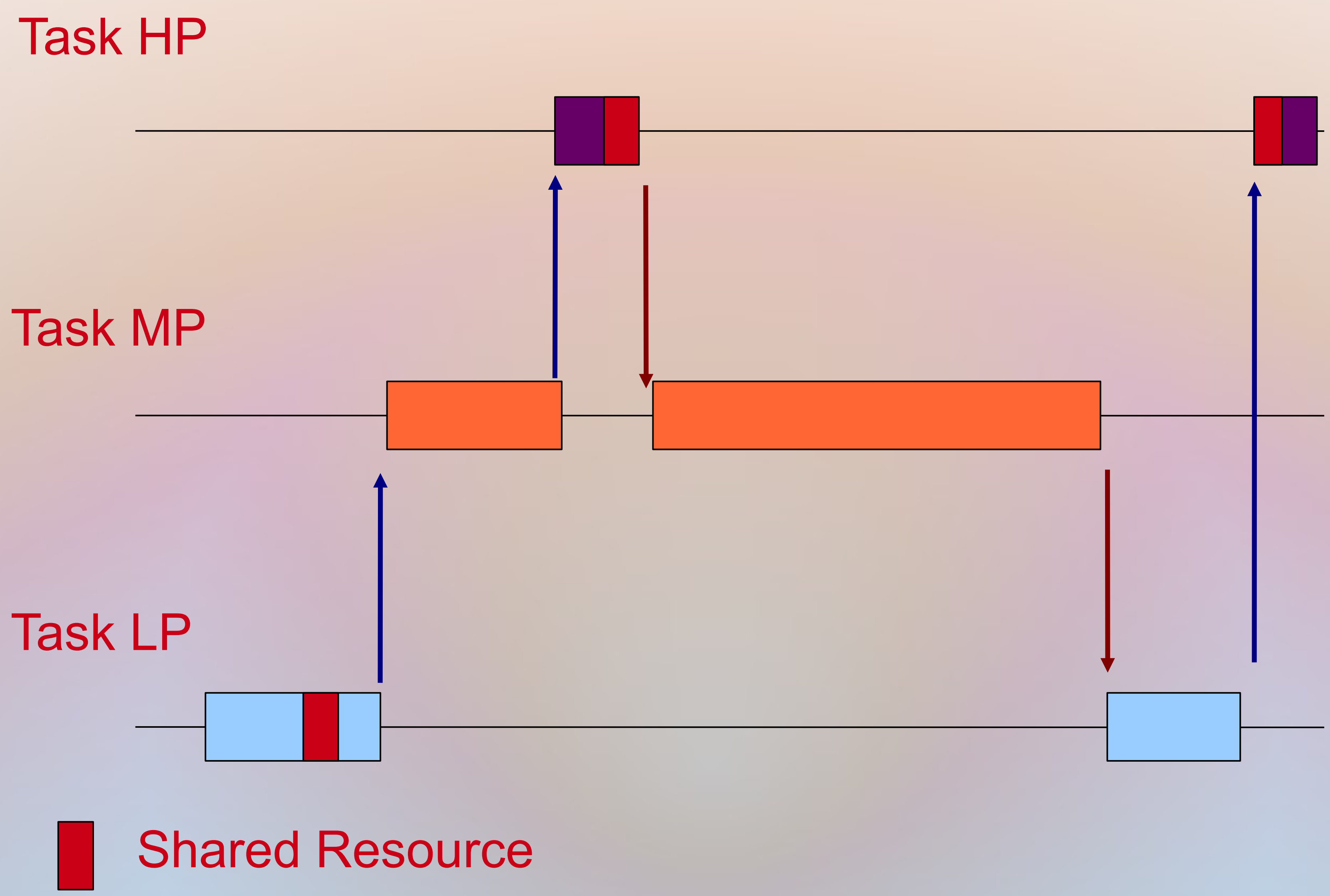
Interrupts in Non Preemptive Kernels



Interrupts in a preemptive kernel

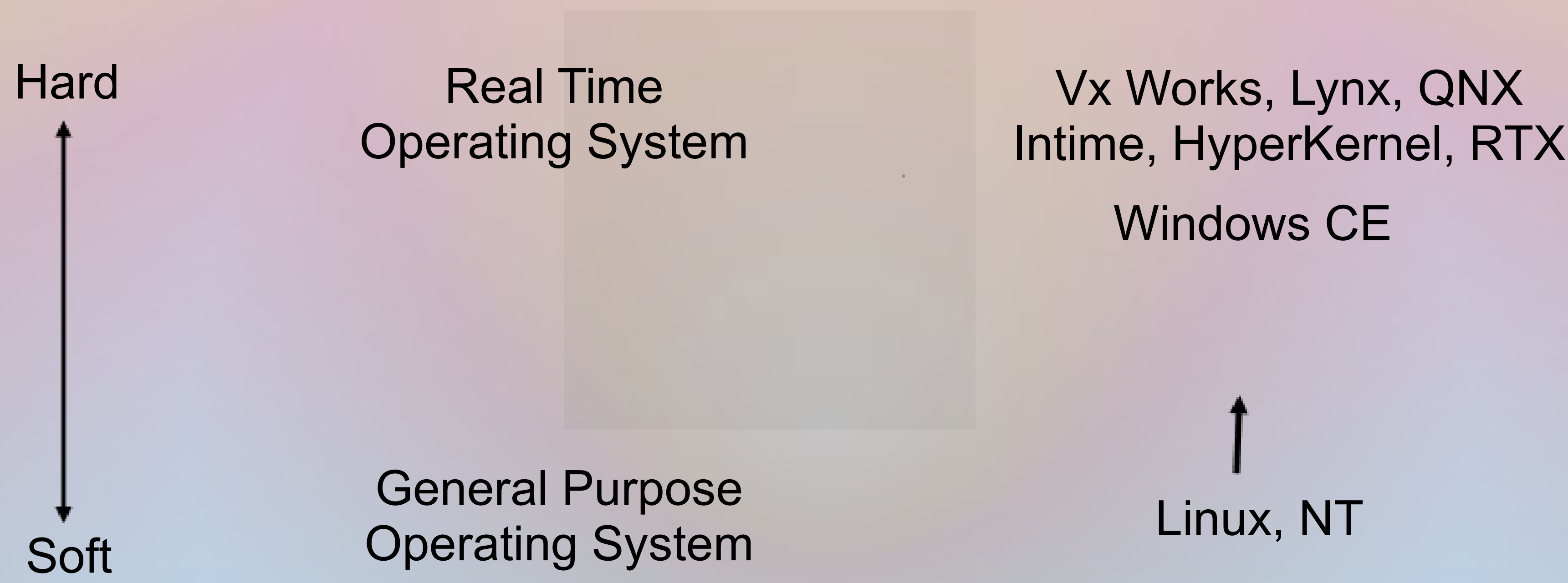


Priority Inversion, Inheritance and Ceiling



Overview of available RTOS’s

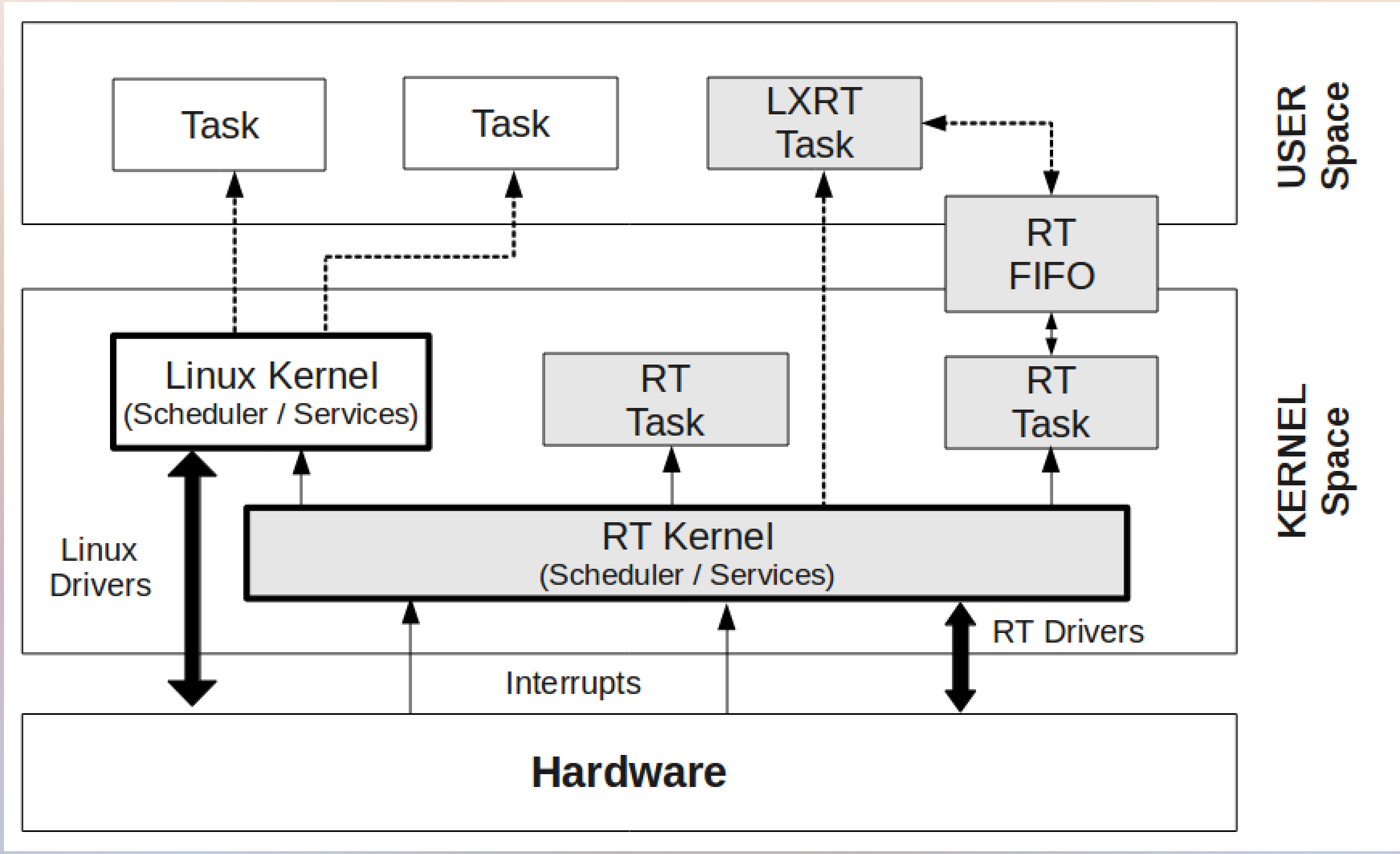
- Three categories of real-time operating systems:
  - Small, proprietary kernels. e.g. VRTX32, pSOS, VxWorks, Windows CE, MicroC-OS/III*
  - Real-time Linux extensions: RT-Linux, Xenomai, RTAI
  - Research kernels: MARS, ARTS, Spring, Polis, MicroC-OS/II



Variants of Real-Time Linux

- Two primary variants of hard real-time Linux available are RTLinux and RTAI.
  - **Real Time Linux (RTLinux)** was developed at the New Mexico Institute of Technology
    - www.rtlinux.org
  - **Real-Time Application Interface (RTAI)** was developed by programmers at the Department of Aerospace Engineering, Milano
    - www.aero.polimi.it/~rtai

Real Time Linux



The Real-Time Linux

- In a real-time Linux system
  - Real-time linux scheduler treats Linux kernel as the idle task
    - Linux only executes when there are no real time tasks to run, and when the real time kernel is inactive.
  - Linux is not permitted to disable hardware interrupts, I.e cannot add latency to the interrupt response time
    - When Linux code tries to disable interrupts, the real time system intercepts the request, records it. Doesn't actually disable ITRs
  - When an interrupt occurs, the real time kernel intercepts the interrupt and dispatches
    - First real time handlers are invoked then Linux handlers

The Real-Time Linux

- Real time kernel never waits for the Linux side to release any resources.
  - Communication links that are used to transfer data between real time tasks and Linux processes are non-blocking on the real time side.
- Linux operating system does as much as is practicable
  - System and Device initialization
  - Blocking dynamic resource allocation
  - Thread of execution that can be blocked
  - Loadable module mechanism to install components of the Real-Time system.

The Real-Time Linux

- The Real Time kernel, all its component parts, and the real time applications run in Linux kernel address space as kernel modules.
  - Advantage: Aids in modularity
  - Disadvantage: A bug in a real time task can crash the whole system.
- Real-time Linux decouples the services of the real time kernel from the services of the general purpose Linux kernel.
  - Real-time kernel can be kept small and simple, each service can be optimized independently

Real-Time Application Interface (RTAI)

- RTAI versions over 3.0 use an Adeos kernel patch, comprising an Interrupt Pipeline, where different Operating System Domains register interrupt handlers.
- Adeos (Adaptive Domain Environment for Operating Systems) is a nanokernel hardware abstraction layer (HAL) that operates between computer hardware and the operating system that runs on it.
  - Provides environment for sharing hardware resources among multiple OS, or multiple instances of a single OS

Real-Time Application Interface (RTAI)

- RTAI provides deterministic and preemptive performance in addition to allowing the use of all standard Linux drivers, applications and functions.
- RTAI's feactures:
  - Traditional RTOS IPCs including: Semaphores, mailboxes, FIFOs, shared memory, and RPCs
  - POSIX 1003.1c (Pthreads, mutexes and condition variables) & POSIX 1003.1b (Pqueues only) compatibility

Real-Time Application Interface (RTAI)

- RTAI's feactures:
  - /proc interface – which provides information on the real-time tasks, modules, services and processes extending the standard Linux /proc file-system support.
  - LXRT – which allows the use of the RTAI system calls from within standard user space
  - UniProcessor, Multi-UniProcessor and Symmetric Multi-processor (SMP) support
  - One-shot and periodic schedulers

RTAI Performance

- RTAI's performance is very competitive with the best commercial Real Time Operating Systems (such as VxWorks, QNX etc)
- Offers:
  - Context switch time: 4 uSec
  - Interrupt response: 20 uSec
  - 100 KHz periodic tasks
  - 30 KHz one-shot task rate

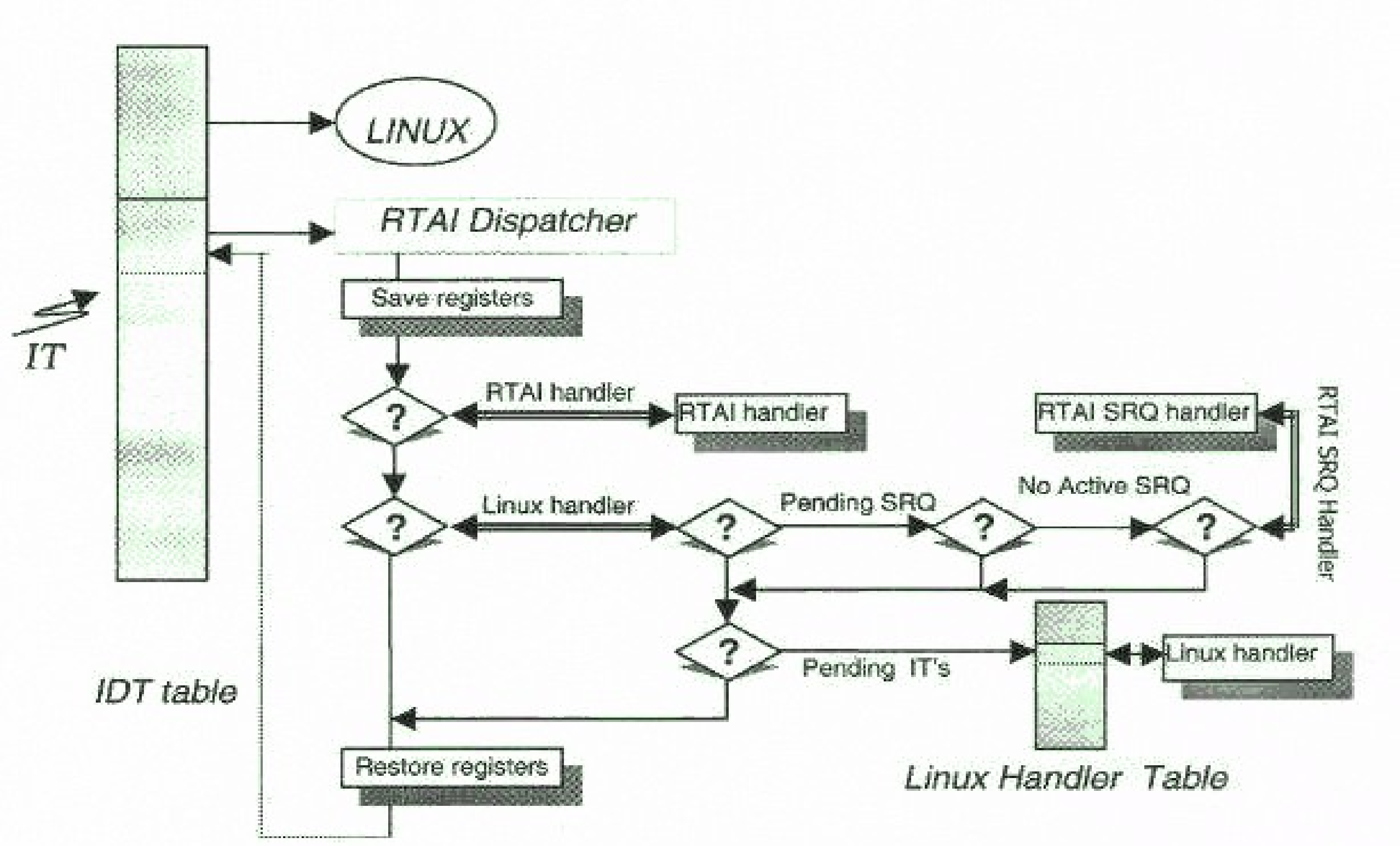
RTAI Components

- The three main RTAI components are:
  - The Interrupt Dispatcher
  - The Scheduler
  - The FIFO

RTAI Interrupt Dispatcher

- The dispatcher is a handler called via the IDT when interrupts occur.
    - Activates the right handler depending on the owner of the driver: RTAI, Linux or both (RTAI first then Linux).
    - The addresses of the Linux handlers are saved before changing the IDT in the HAL.
    - Takes into account the “SRQ(service request) handlers”.
        - SRQ is a handler used for a system call implementation. SRQ can be either RTAI handler (e.g.: fifo) or User handler.
    - RTAI manages up to 31 SRQ's. SRQ handler addresses are stored in IDT.

RTAI Interrupt Dispatcher



RTAI Scheduler

- The scheduler is in charge of distributing the CPU to different tasks present in the system (including Linux).
    - The RTAI distribution includes three different priority based, pre-emptive real time schedulers:
        - Uni-Processor (UP) scheduler;
        - Multi Uni-Processor (MUP) scheduler;
        - Symmetric Multi-Processor (SMP) scheduler
  - During the installation process, a scheduler is determined based on the hardware configuration of the target machine.

RTAI Scheduler

- UP scheduler - For uni-processor platforms.
  - Timer supports either one-shot or periodic scheduling but not both simultaneously.
- SMP scheduler - For multi-processor machines.
  - Timer supports either one-shot or periodic scheduling but not both simultaneously.
  - Tasks can run symmetrically on any or a cluster of CPUs, or be bound to a single CPU.
  - By default all tasks are defined to run on any of the CPUs and are automatically moved between CPUs as the system's processing and load requirements change.

RTAI Scheduler

- Multi-Uniprocessor scheduler - For multiprocessor platforms only and supports both one-shot and periodic scheduling simultaneously.
  - The main advantage is the ability to be able to use mixed timers simultaneously, i.e. periodic and one-shot timers.
  - Periodic timers can be based on different periods.
  - Like the SMP schedulers, the MUP can use inter-CPU services related to semaphores, messages and mailboxes.

Scheduler Functionality

- The RTAI multitasking scheduler, uses interrupt-driven, priority-based task scheduling.
- The scheduler elects the first highest priority task in a READY state.
    - Priority 0 is the highest priority and 0x3fffFfff the lowest for rtai tasks
    - Linux is given priority 0x7fffFfff.
    - RTAI takes care of Linux activation. When timer handler is running and the next Linux time period is reached a Linux pending interrupt is raised and will be served by the dispatcher.

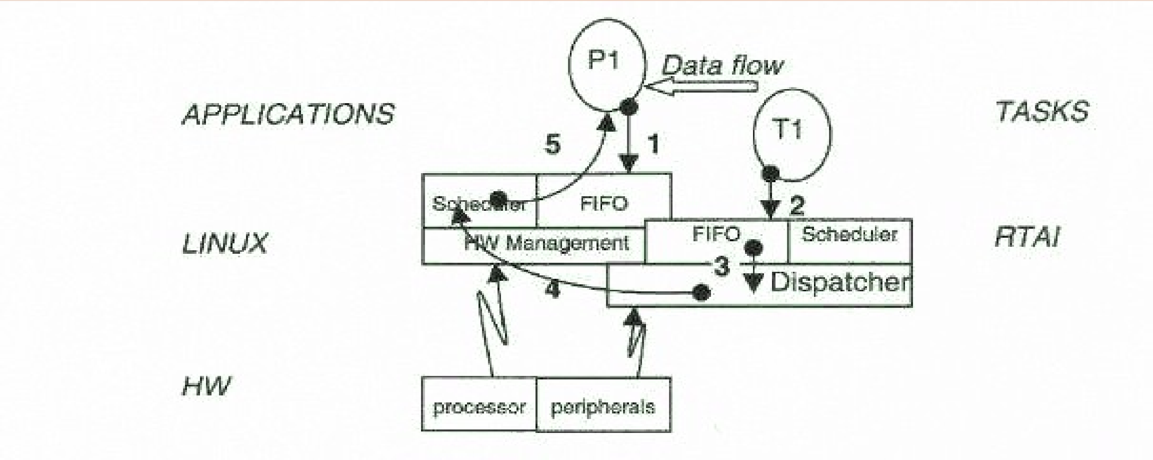
Scheduler Functionality

- All RTAI schedulers incorporate standard RTOS scheduling services like resume, yield, suspend, make periodic, wait until etc.
- The scheduler looks for ready tasks or tasks with expired period.
  - A task may be in the following states : READY, SUSPENDED, DELAYED, SEMAPHORE, SEND, RECEIVE, RPC, RETURN, RUNNING, DELETED
  - On Timer event, the timer handler changes the task state.

RTAI FIFOs

- Fifos allow Linux processes and RTAI tasks to exchange byte-oriented data steams.
  - A fifo is a one way channel therefore a duplex communication needs 2 fifos.
  - From a Linux process point of view the mechanism allows either blocking or non-blocking IO depending on user program.
  - From a task point of view fifo allows non-blocking or asynchronous IO.
  - To perform synchronous reads (and to avoid polling) RTAI makes it possible to attach a user real time handler to a fifo.

RTAI FIFOs



- blocking read.
- rtf_put call.
- SRQ initialization.
- wake-up action in queue.
- wake-up action de-queued at next Linux scheduling

RTAI LXRT Module

- LXRT is a module that allows to use all the services made available by RTAI and its schedulers in user space, both for soft and hard real time.