EE414 Embedded Systems

Ch 10.
Operating Systems

Part 2/2

Byung Kook Kim
School of Electrical Engineering
Korea Advanced Institute of Science and Technology

Overview

- 10.7 Scheduling Policies
- 10.8 Power Optimization
- 10.9 Real-Time Systems

10.7 Scheduling Policies

Metrics

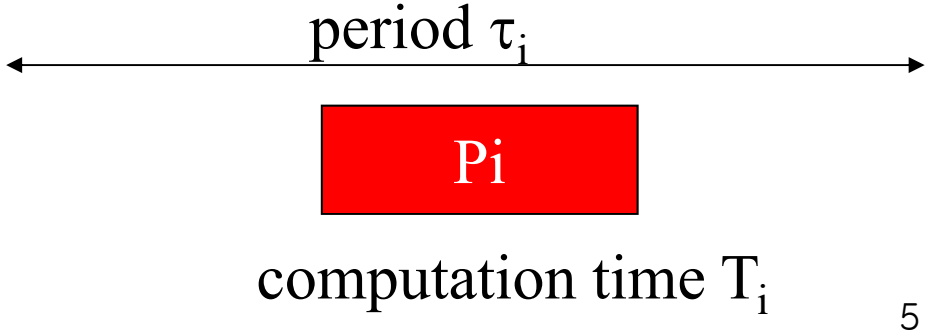
- How do we *evaluate* a scheduling policy:
  - Ability to satisfy all *deadlines*.
  - CPU utilization
    - Percentage of time devoted to useful work.
  - Scheduling overhead
    - Time required to make scheduling decision.

A. Rate Monotonic Scheduling

- **Rate Monotonic Scheduling (RMS)**
  - Liu and Layland, 1973
  - Widely-used, analyzable scheduling policy.
  - Static scheduling policy.
  - Analysis is known as Rate Monotonic Analysis (RMA).

RMA Model

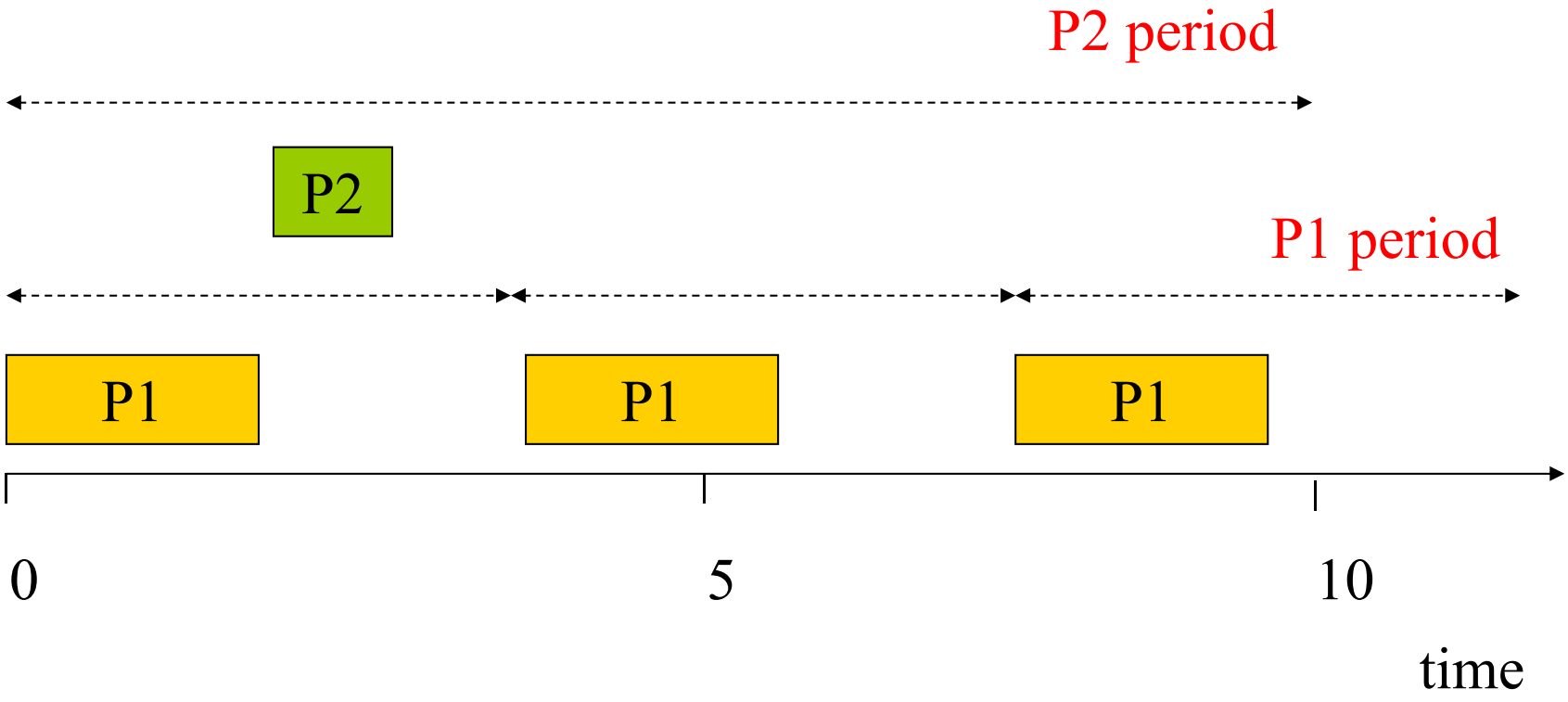
- RMA model
  - All process run on single CPU.
  - Zero context switch time.
  - No data dependencies between processes.
  - Process execution time is constant.
  - Deadline is at the end of period.
  - Highest-priority ready process runs.
- Process parameters
  - T_i is computation time of process i;
  - τ_i is period of process i.



RMS Priorities

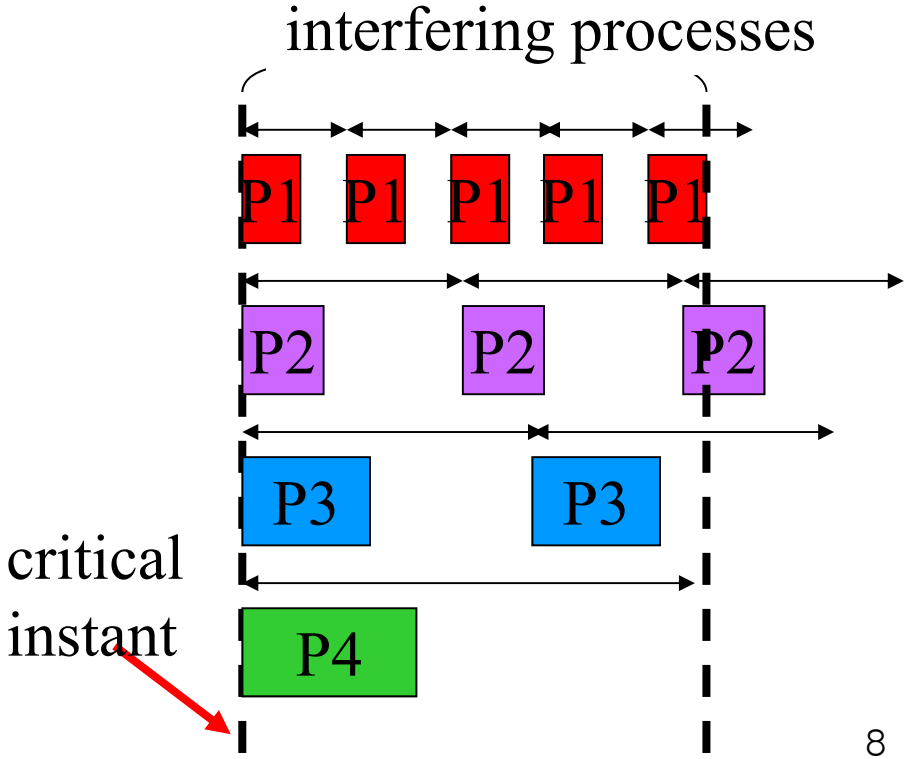
- RMS priority
  - Priority inversely proportional to period
  - Shortest-period process gets highest priority
  - Break ties arbitrarily.
- Optimal fixed priority assignment:
  - No fixed-priority scheme does better.

RMS Example



Rate-Monotonic Analysis

- Response time:
  - Time required to finish the process
- Critical instant:
  - Scheduling state that gives worst response time.
  - Critical instant occurs when all higher-priority processes are ready to execute.



RMS CPU Utilization

- Utilization for n processes is
  - $\sum_i T_i / \tau_i$
- As number of tasks approaches infinity, maximum utilization approaches ln 2 = 69%.
- RMS cannot asymptotically guarantee use 100% of CPU, even with zero context switch overhead.
- Must keep idle cycles available to handle worst-case scenario.
- However, RMS guarantees all processes will always meet their deadlines.

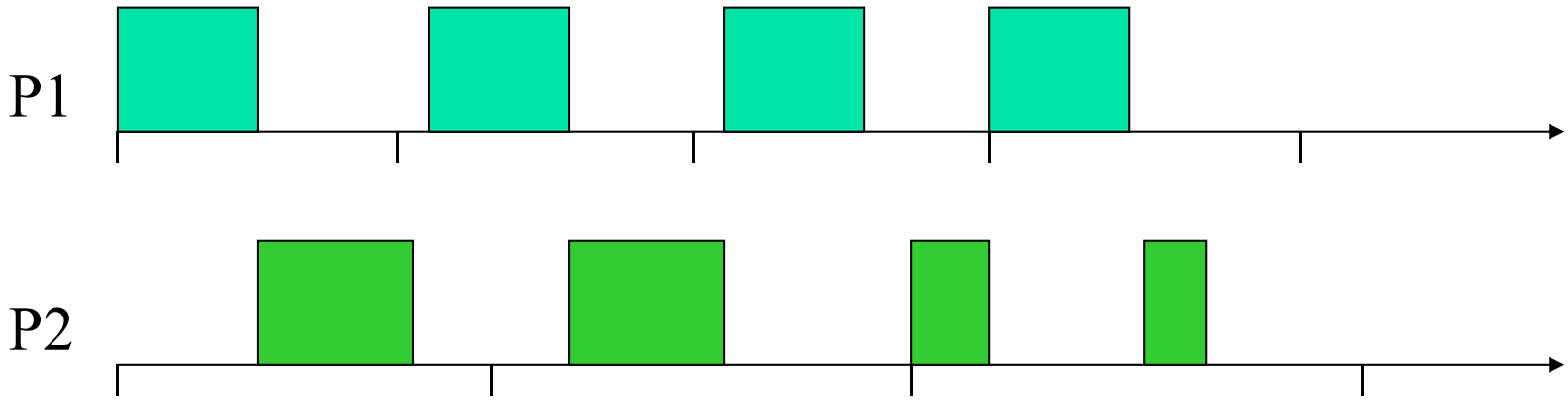
RMS Implementation

- Efficient implementation:
  - Processes sorted by priority in advance
  - Scan processes & choose highest-priority active process.
    - Complexity O(n)
- POSIX
  - SCHED_FIFO
    - Strict priority-based scheduling scheme: Rate monotonic scheduling
    - Within a priority, process run FCFS
    - sched_setparam(pid, &params)

B. Earliest-Deadline-First Scheduling

- **Earliest Deadline First (EDF)**
  - Dynamic priority scheduling scheme.
  - Process closest to its deadline has highest priority.
  - Requires recalculating deadlines (priorities) of processes at every timer interrupt.
  - POSIX does not currently support EDF.

EDF Example



EDF Analysis & Implementation

- EDF analysis
  - EDF can use 100% of CPU.
  - But EDF may miss a deadline.
- EDF Implementation
  - On each timer interrupt:
    - Compute time to deadline for all processes;
    - Choose process closest to deadline.
  - Generally considered too expensive to use in practice.

Fixing Scheduling Problems

- What if your set of processes is unschedulable?
  - *Change deadlines* in requirements.
  - *Reduce execution times* of processes.
  - *Get a faster CPU*.

Closer Look

C. Priority Inversion

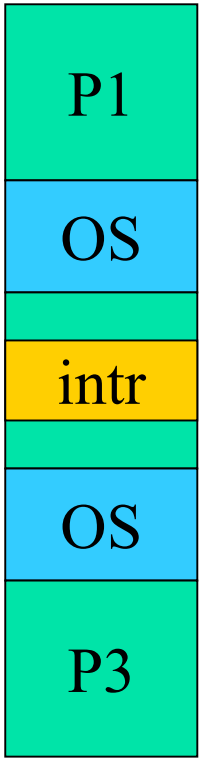
- Priority inversion
  - Low-priority process keeps high-priority process from running.
  - Improper use of system resources can cause scheduling problems:
    - Low-priority process grabs I/O device.
    - High-priority device needs I/O device, but can't get it until low-priority process is done.
  - Can cause deadlock.

Solving Priority Inversion

- Priority inheritance
  - Give priorities to system resources.
  - Have process inherit the priority of a resource that it requests.
    - Low-priority process inherits priority of device if higher.

What about Interrupts?

- Interrupts take time away from processes.
- Perform minimum work possible in the interrupt handler.
  - Interrupt service routine (ISR) performs minimal I/O.
    - Get register values, put register values.
  - *Interrupt service process/thread* performs most of device function.



10.8 Power Optimization

CPU power consumption

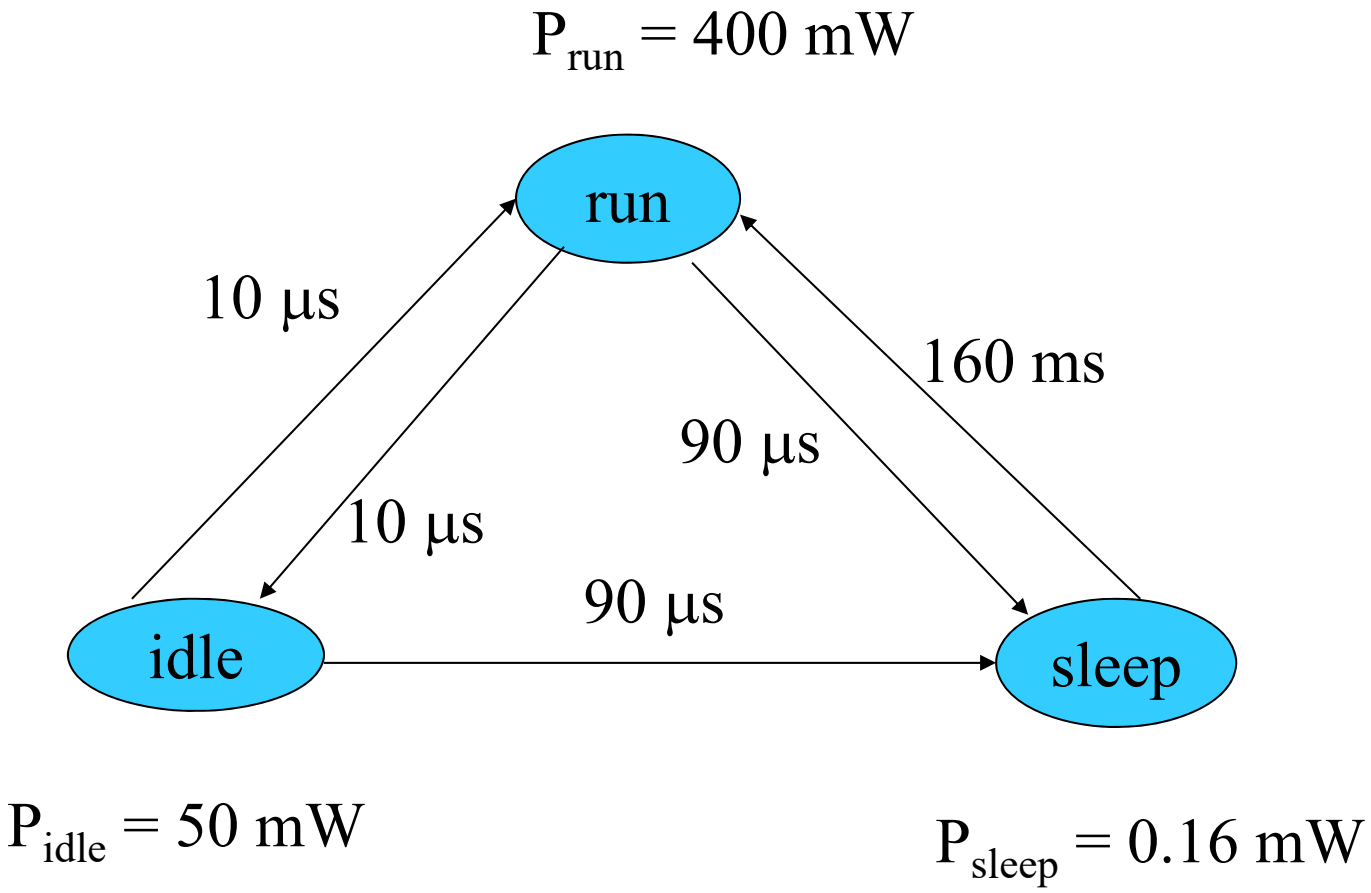
- Most modern CPUs are designed with power consumption in mind to some degree.
  - Battery powered system

- Power
  - Heat depends on power consumption
    - Increase reliability
- Energy
  - Battery life depends on energy consumption
    - Reduce system cost.

CMOS Power Consumption

- Voltage drops: power consumption proportional to V^2.
  - Reduce the power supply voltage
- Toggling: more activity means more power.
  - Reduce the clock speed
  - Eliminate unnecessary changes to the inputs of a CMOS circuit.
- Leakage: basic circuit characteristics
  - Can be eliminated by disconnecting power.

SA-1100 power state machine

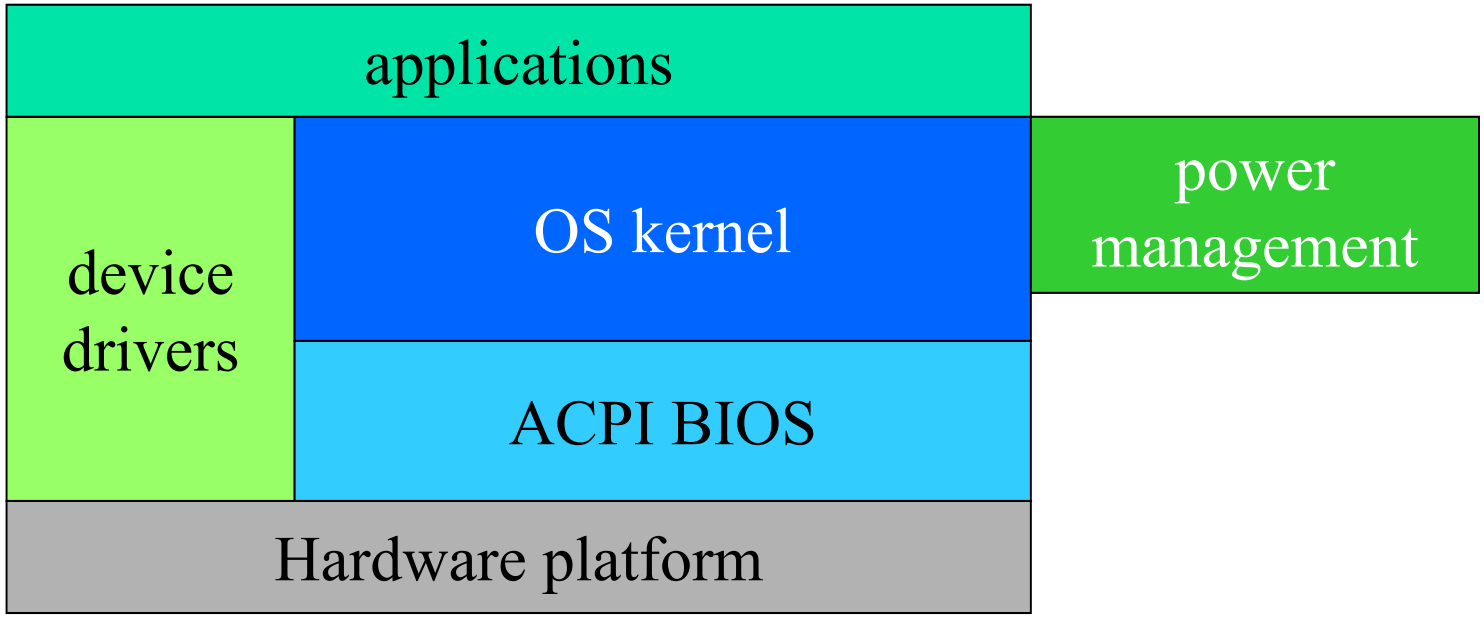


Power Optimization in OS

- Power management
  - Determining how system resources are scheduled/used to control power consumption.
  - OS can manage for power just as it manages for time.
  - OS reduces power by shutting down units.
    - May have partial shutdown modes.

Advanced Configuration and Power Interface

- ACPI (Advanced Configuration and Power Interface)
  - Open industry standard for power management services.
  - Initially targeted for PC

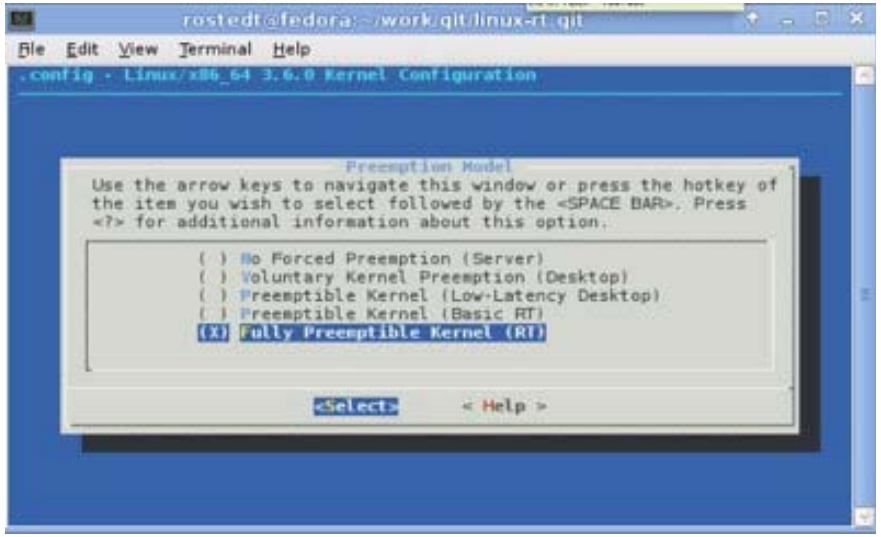


10.9 Real-Time Systems

- Systems composed of 2 or more cooperating, concurrent processes with stringent execution time constraints
  - E.g., set-top boxes have separate processes that read or decode video and/or sound concurrently and must decode 20 frames/sec for output to appear continuous
  - Other examples with stringent time constraints are:
    - digital cell phones
    - navigation and process control systems
    - assembly line monitoring systems
    - multimedia and networking systems
    - etc.
  - Communication and synchronization between processes for these systems is critical
  - Therefore, concurrent process model best suited for describing these systems

Preempt_RT

- **Philosophy of PREEMPT_RT**
  - The key point of the PREEMPT_RT patch is to minimize the amount of kernel code that is non-preemptible.
  - Also minimizing the amount of code that must be changed in order to provide this added preemptibility.
- **Features of PREEMPT_RT**
  - Preemptible critical sections
  - Preemptible interrupt handlers
  - Preemptible "interrupt disable" code sequences
  - Priority inheritance for in-kernel spinlocks and semaphores
  - Deferred operations
  - Latency-reduction measures.



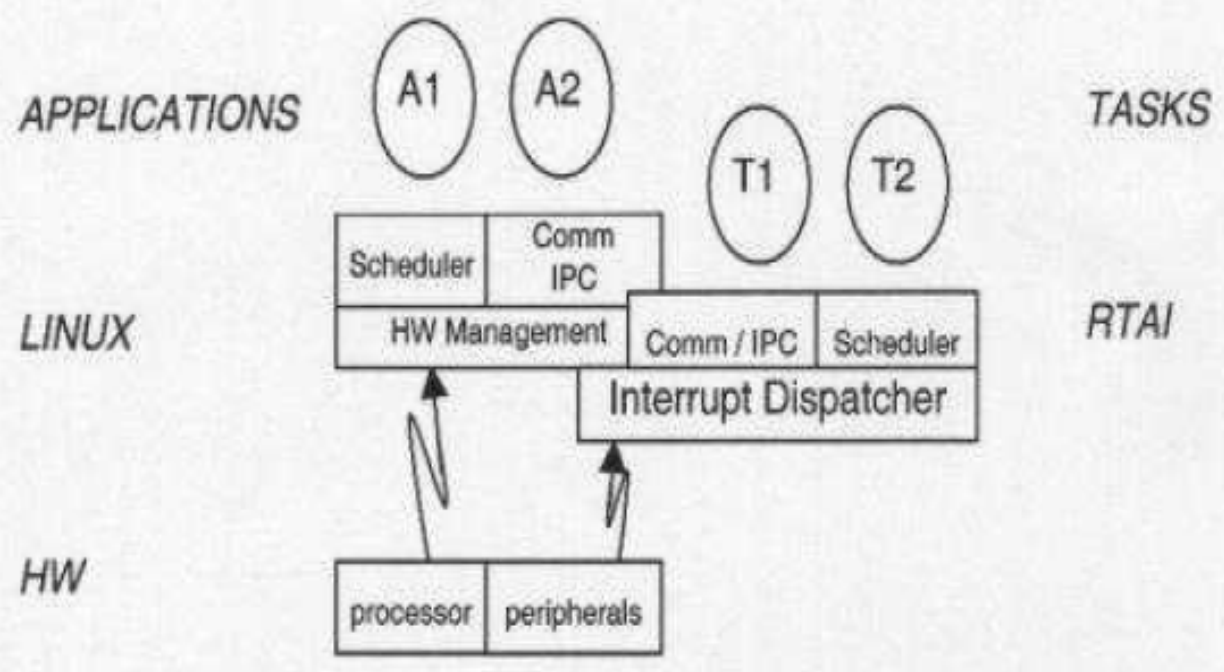
RTAI

- **Real-Time Application Interface (RTAI)**
  - Developed by DIAPM (Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano) by Paolo Mantegazza's team.

- *Overview of RTAI*
  - The RTAI plug-in should help Linux to fulfill some real time constraints (few milliseconds deadline, no event loss).
  - It is based on a RTHAL: Real Time Hardware Abstraction Layer.
    - This concept is also known in Windows NT.
  - The HAL exports some Linux data & functions close related to HW.
    - RTAI modifies them to get control over the HW platform.
    - That allows RTAI real time tasks to run concurrently with Linux processes.
    - The HAL defines a clear interface between RTAI & Linux.

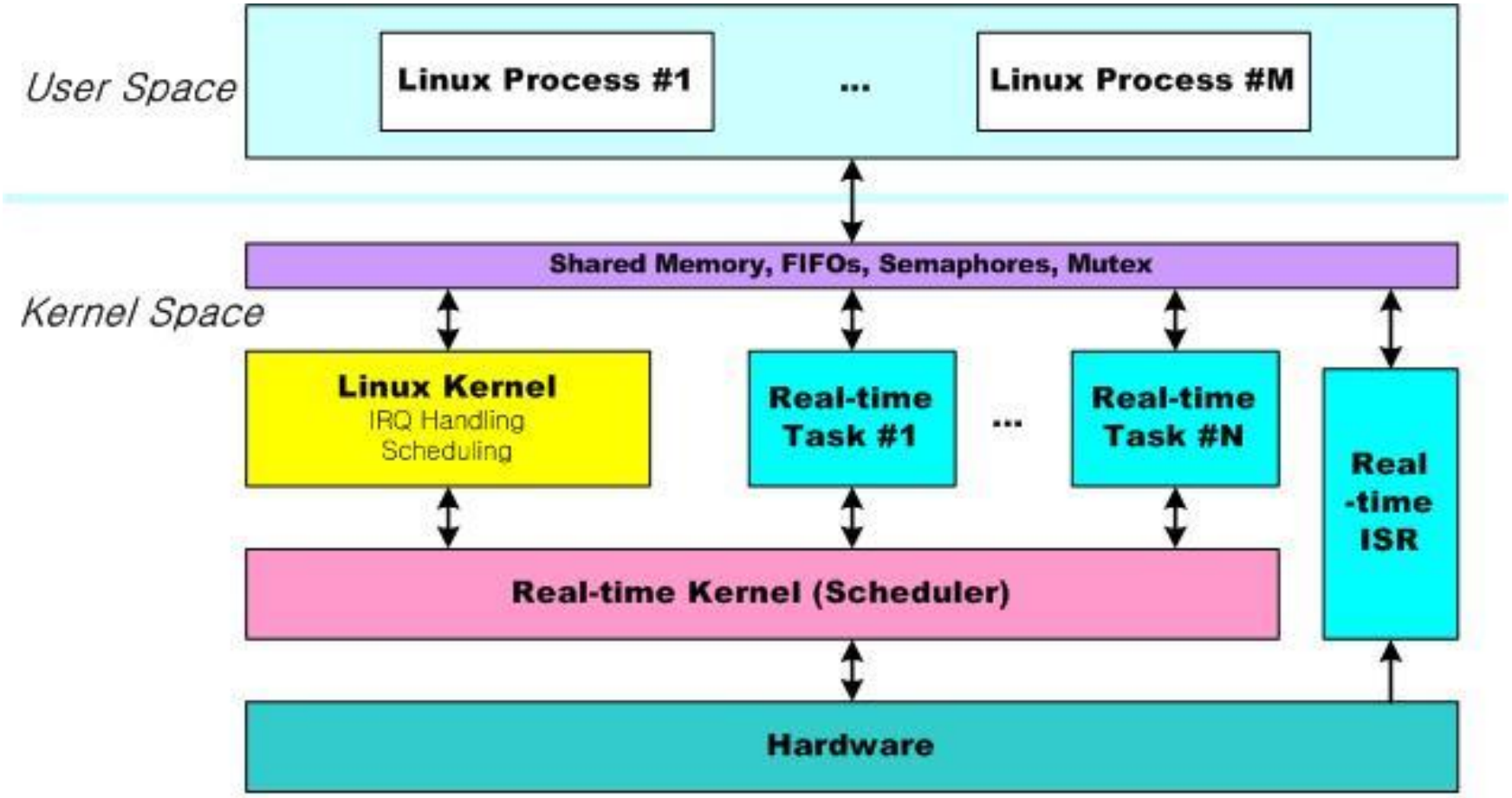
RTAI (II)

- RTAI is a small proprietary executive offering some services related to:
  - HW management layer dealing with peripherals.
  - Scheduler classes dealing with tasks, priorities, hard real-time.
  - Communications means among tasks & processes (at least FIFO).



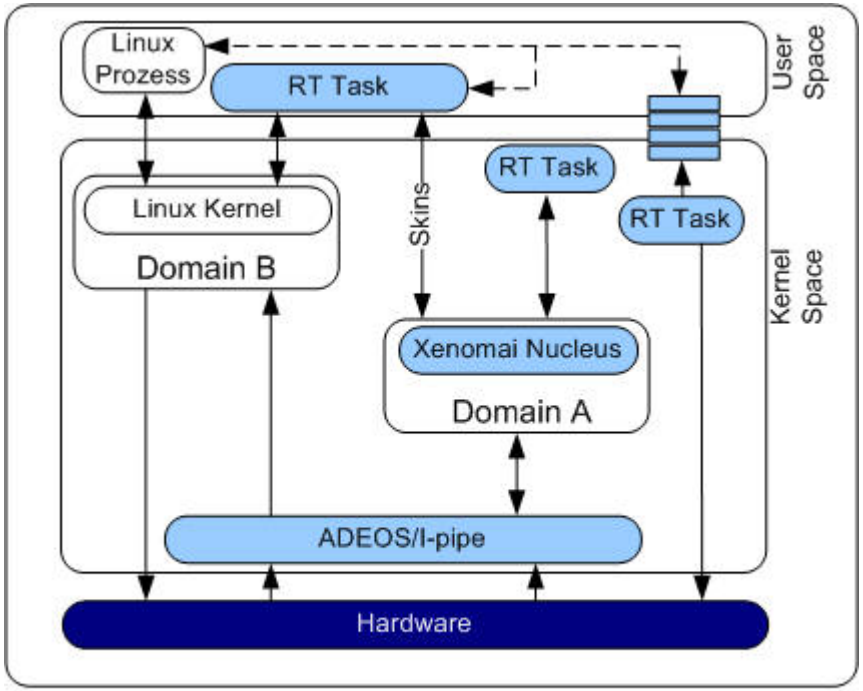
RTAI Block Description

- Dual kernel architecture



Xenomai

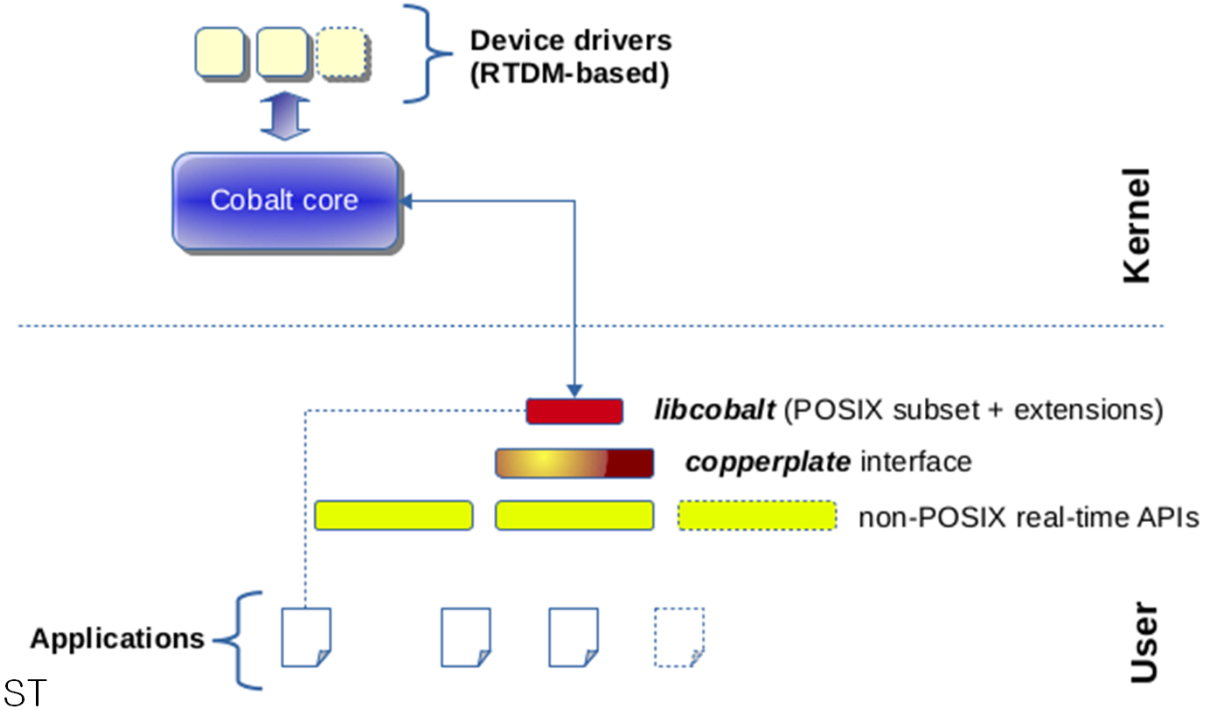
- A real-time development framework cooperating with the Linux kernel, to provide a pervasive, interface-agnostic, hard real-time support to user space applications, seamlessly integrated into the Linux environment.
- Xenomai is based on an abstract RTOS core, usable for building any kind of real-time interface, over a nucleus which exports a set of generic RTOS services.
- Any number of RTOS personalities called "skins" can then be built over the nucleus, providing their own interface to the applications, by using the services of a single generic core to implement it.



Xenomai

Xenomai 3

- Dual-kernel Cobalt architecture
  - The dual kernel nicknamed *Cobalt*, is a significant rework of the Xenomai 2.x system.
  - *Cobalt* implements the RTDM specification for interfacing with real-time device drivers.



References

- [1] Wayne Wolf, “Computers as Components”, Morgan Kaufman, 2001.
- [2] Preempt_RT, https://rt.wiki.kernel.org/index.php/Main_Page
- [3] RTAI, http://www.rtai.org
- [4] Xenomai, http://www.xenomai.org