7. Support in Operating Systems and in the Ada Language

UNIVERSIDAD DE CANTABRIA

7.1 Introduction

7.2 Available scheduling policies

7.3 Implementation of typical arrival patterns

7.4 Implementation of synchronization protocols

7.5 Implementation of aperiodic servers

7.6 Effects of operating system services

7.7 Limited representation of system information

Real-time operating systems

UNIVERSIDAD DE CANTABRIA

In the past, many real-time systems did not need an operating system

Today, many applications require operating system services such as:

- **concurrent programming, communication networks, file system, etc.**

The timing behavior of a program depends strongly on the operating system's behavior

POSIX definition of real-time in operating systems:
"The ability of the operating system to provide a required level of service in a *bounded response time*."

What is POSIX?

UNIVERSIDAD DE CANTABRIA

Portable Operating System Interface

Based on UNIX operating systems

The goal is portability of

- **applications (at the source code level)**
- **programmers**

Sponsored by the IEEE and The Open Group

POSIX real-time profiles (POSIX.13)

UNIVERSIDAD DE CANTABRIA

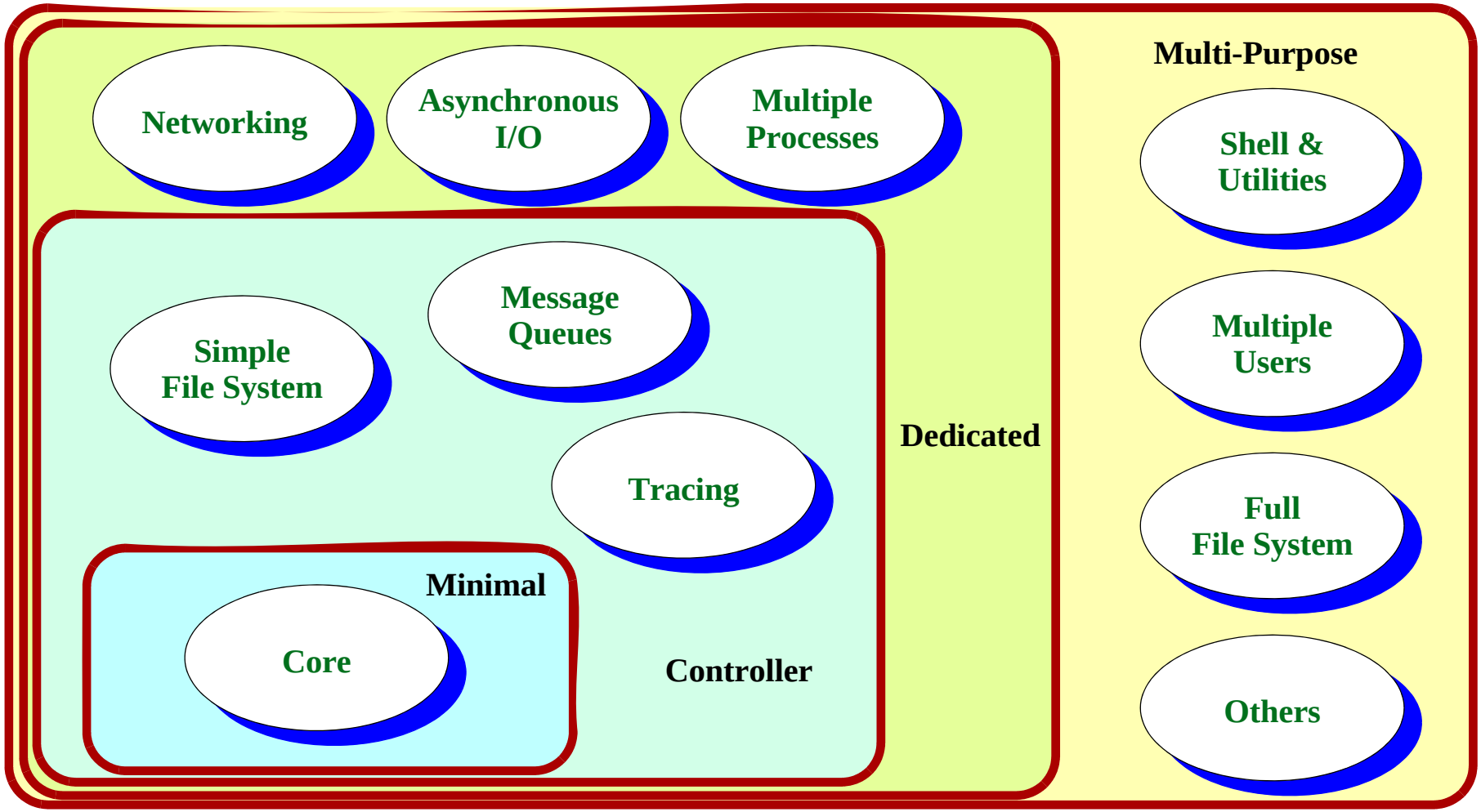
The POSIX Standard:

- Allows writing portable real-time applications
- Very large: inappropriate for embedded real-time systems

POSIX.13:

- Defines four real-time system subsets (profiles)
- C and Ada language options

Summary of RT Profiles



Programming languages for real-time systems

UNIVERSIDAD DE CANTABRIA

Some languages do not provide support for concurrency or real-time

- C, C++
- support achieved through OS services (POSIX)

Other languages provide support in the language itself

- Java (concurrency) and RTSJ (real-time)
  - API based
- Ada: concurrency and real-time
  - integrated into the core language

Programming languages for real-time systems

UNIVERSIDAD DE CANTABRIA

Language support helps in

- increased reliability
- more expressive power
- reduced development and maintenance costs

7.2 Available Scheduling Policies

The policies available in real-time POSIX are:

- `SCHED_OTHER` (default policy)
- `SCHED_FIFO`: preemptive fixed priorities, FIFO for equal priority
- `SCHED_RR`: preemptive fixed priorities, round-robin for equal priority, with fixed quantum
- `SCHED_SPORADIC`: sporadic server

Since they are compatible, they are selectable for each thread

- initially, from a thread attribute
- dynamically, with specific functions

Scheduling policies in Ada

UNIVERSIDAD DE CANTABRIA

The Ada 2005 RT annex specifies different scheduling policies that may be selected via:

- for a single scheduler:

```
pragma Task_Dispatching_Policy (policy_identifier)
```

- for hierarchical scheduling with an underlying fixed priority scheduler, schedulers are defined for different priority bands:

```
pragma Priority_Specific_Dispatching
  (policy_identifier, first_priority,
   last_priority)
```

Scheduling policies

The policies defined are:

- `FIFO_Within_Priorities` (is the standard policy in Ada 95)
- `Non_Preemptive_FIFO_Within_Priorities` (not available for hierarchical scheduling)
- `Round_Robin_Within_Priorities`
- `EDF_Across_Priorities`

Fixed Priorities

Fore fixed priorities, the priority is declared with

```
pragma Priority (value)
pragma Interrupt_Priority (value)
```

And may be modified with the `Ada.Dynamic_Priorities` package

```
procedure Set_Priority
  (Priority : System.Any_Priority,
   T : Ada.Task:Identification.Task_Id:=
   Ada.Task_Identification.Current_Task);
```

EDF

For EDF, the relative deadline is declared with

```
pragma Relative_Deadline (interval)
```

And may be modified with the `Ada.Dispatching.EDF` package:

```
function Get_Deadline
   (T : Ada.Task_Identification.Task_Id)
    return Deadline;
procedure Set_Deadline
   (D : Deadline,
    T : Ada.Task_Identification.Task_Id);
procedure Delay_Until_And_Set_Deadline
   (Delay_Until_Time : Ada.Real_Time.Time,
    Deadline_Offset : Ada.Real_Time.Time_Span);
```

Round Robin

UNIVERSIDAD DE CANTABRIA

A time quantum is defined and when a task consumes its quantum it is preempted by the next ready task

Package `Ada.Dispatching.Round_Robin` contains operations to set and get the quantum

7.3 Implementation of Arrival Patterns: Periodic Tasks in Ada

UNIVERSIDAD DE CANTABRIA

```
with Ada.Real_Time;
use Ada.Real_Time;
task body Periodic_Task is
   Period : constant Time_Span:=
        To_Time_Span(0.010); --10 msec
   Next_Start : Time := Clock;
begin
   loop
      -- do task work
      Next_Start := Next_Start + Period;
      delay until Next_Start;
   end loop;
end Periodic_Task;
```

Precise Scheduling of Periodic Tasks

Note that if a relative delay is used in the above example, there may be a race condition

```
delay ( Next_Start - Clock );
```

- the task may be preempted after reading clock and before executing the delay statement

Periodic Tasks in Ada, under EDF

```
Period : constant Time_Span:=
      To_Time_Span(0.010); --10 msec
Relative_Deadline : constant Deadline :=Period;

task Periodic_Task is
  pragma Relative_Deadline (Relative_Deadline);
       -- the first absolute deadline is
       -- current_time+relative_deadline
end Periodic_Task
```

Periodic Tasks in Ada, under EDF

```
task body Periodic_Task is
  Next_Start : Time := Clock;
begin
  loop
    -- do task work
    Next_Start := Next_Start + Period;
    Delay_Until_And_Set_Deadline
      (Next_Start, Relative_Deadline);
        -- the relative deadline is added
        -- to the absolute one
  end loop;
end Periodic_Task;
```

7.4 Implementation of Synchronization Protocols

POSIX defines three synchronization protocols for mutexes (but not for semaphores, nor for reader-writer locks)

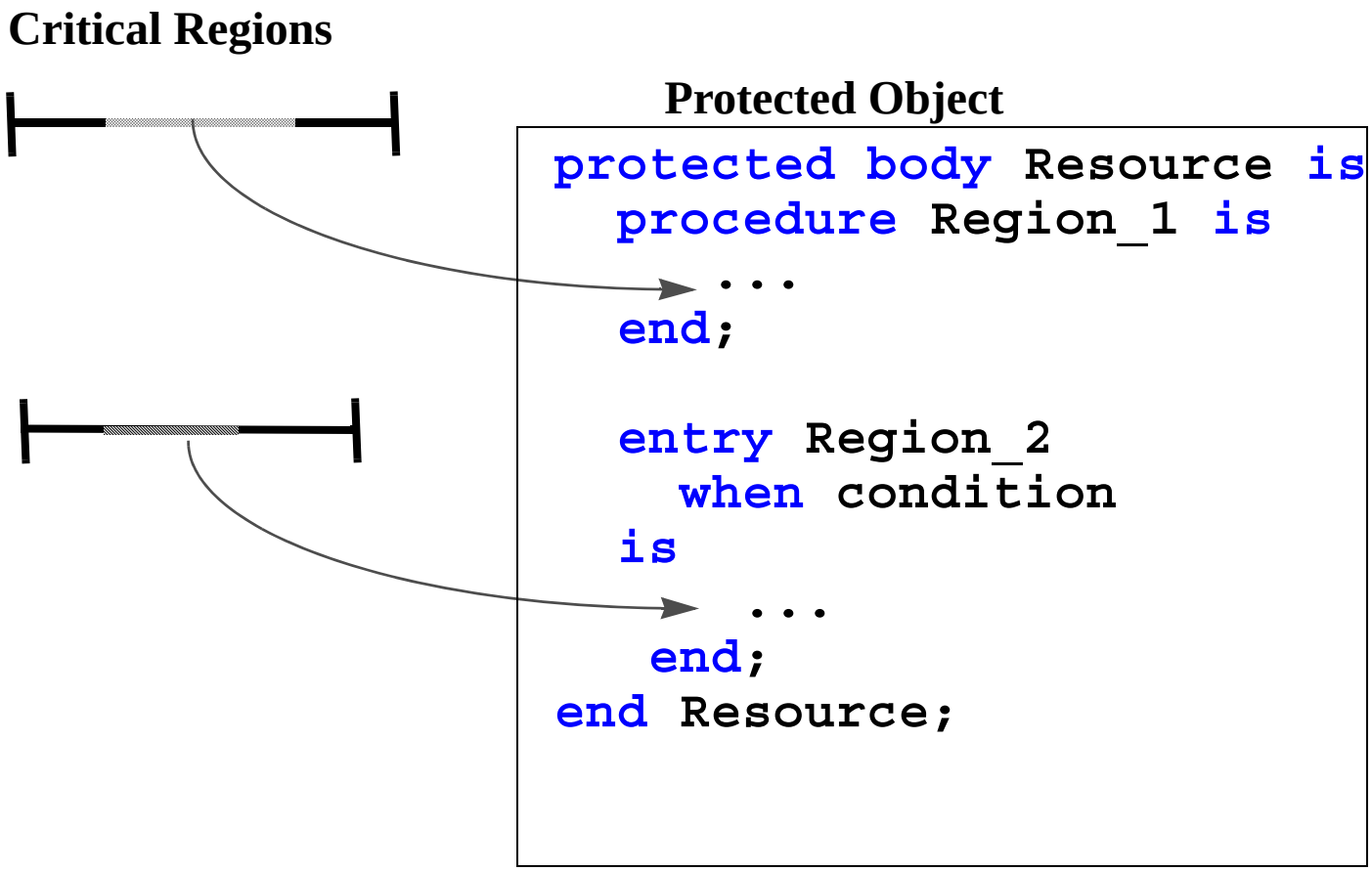
- `PTHREAD_PRIO_NONE`: no handling of priorities
- `PTHREAD_PRIO_INHERIT`: basic inheritance
- `PTHREAD_PRIO_PROTECT`: immediate priority ceiling

These are chosen as a mutex attribute

Priority ceilings are also a mutex attribute

- they may be changed dynamically

Critical Regions In Ada



Locking policies

The locking policies for protected objects are established with

```
pragma Locking_Policy (locking_policy_identifier)
```

The locking policy defined is the immediate priority ceiling

```
Ceiling Locking
```

Ceilings are assigned with

```
pragma Priority(value)
attribute 'Priority (for dynamic changes)
```

Rules are defined so that the locking policy behaves as the SRP in EDF across priorities

- Priorities are used as preemption levels

User-level implementation of IPC

UNIVERSIDAD DE CANTABRIA

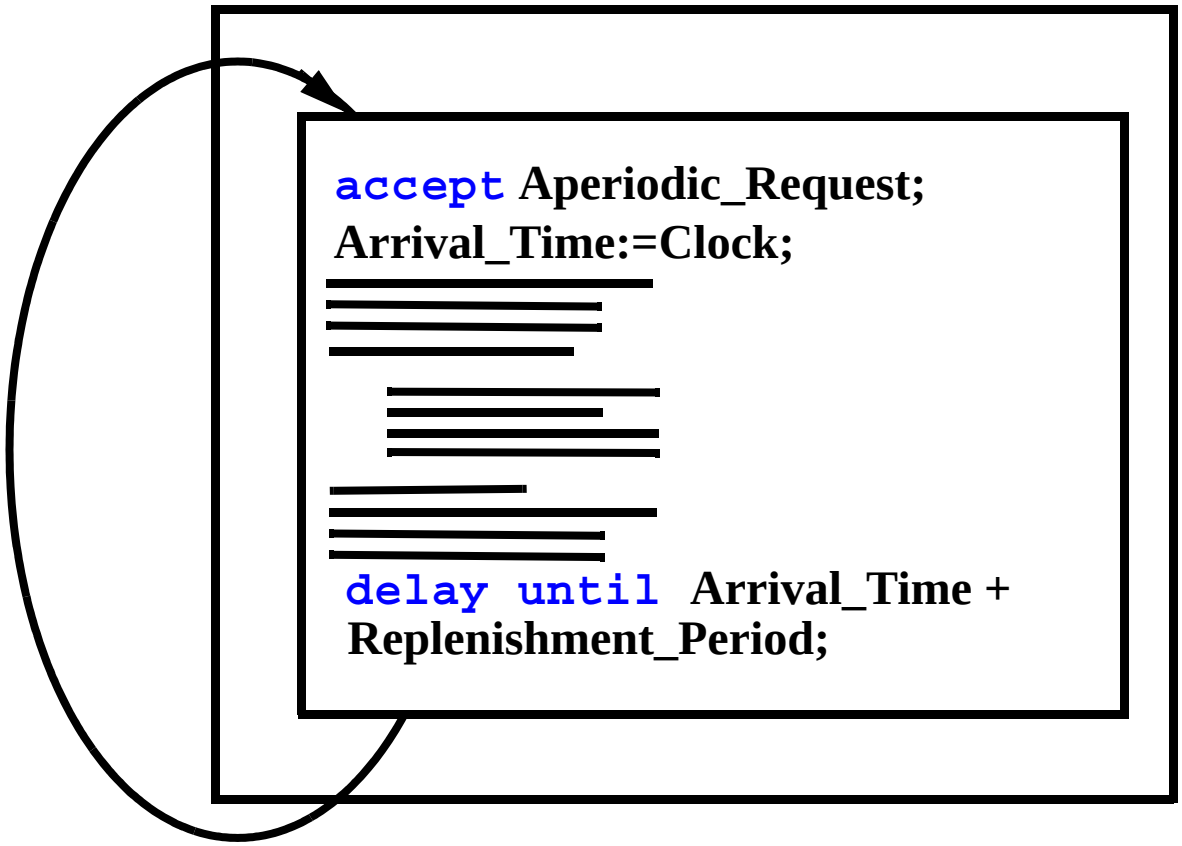
If a priority inheritance or priority ceiling protocol is not available under fixed priorities, IPC can be emulated

Pseudocode of critical section:

```
raise priority to ceiling
lock mutex
do useful work
unlock mutex
lower priority to normal level
```

7.5 Implementation of aperiodic servers

User-Implemented Sporadic Servers



7.6 Effects of Operating System Overheads

UNIVERSIDAD DE CANTABRIA

New events and actions

Implicit OS resources

Limited representation of system parameters

Presence or absence of OS features

Scheduling Policies

New events and Actions

UNIVERSIDAD DE CANTABRIA

New events and actions must be considered in the analysis:

- **new events:**
  - **system tick, diagnostic processes, etc.**
- **actions at caller's priority:**
  - **library-level services**
- **actions at a higher priority:**
  - **kernel-level services, context switch actions, etc.**
- **atomic actions on the CPU:**
  - **interrupt service routines, delay expiration, etc.; they may have a severe impact on timing response.**

Notes:

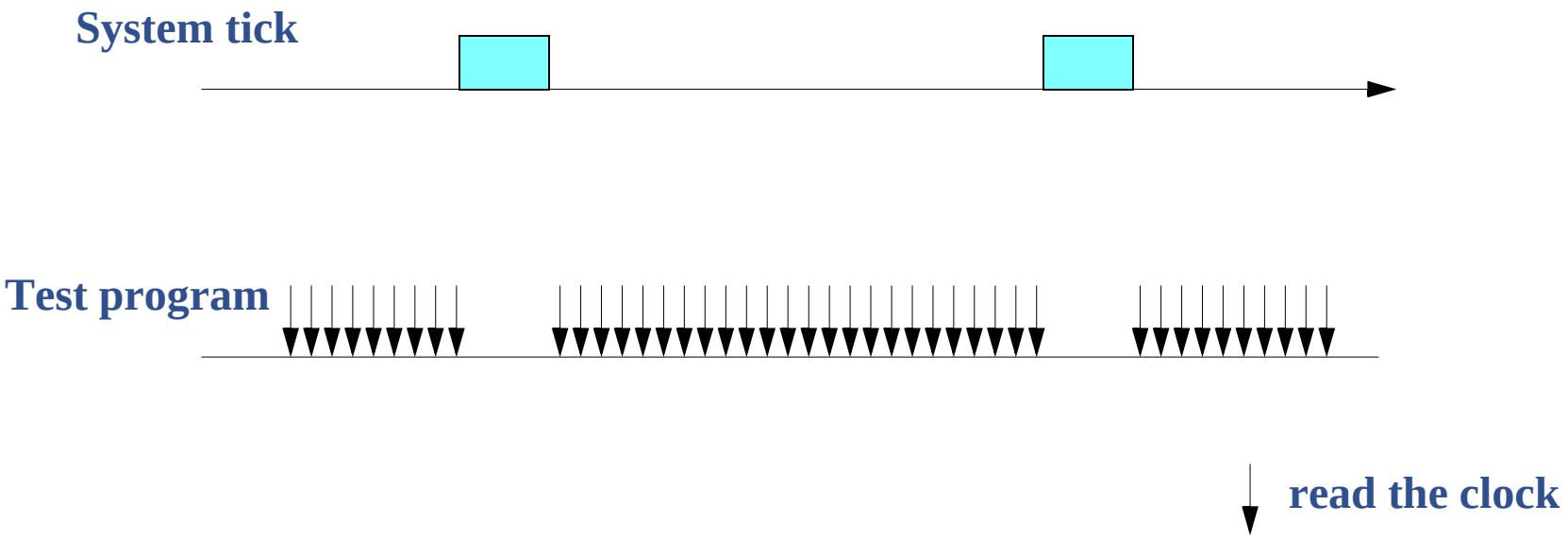
UNIVERSIDAD DE CANTABRIA

The new events and actions introduced by the operating system must be taken into account in the analysis:

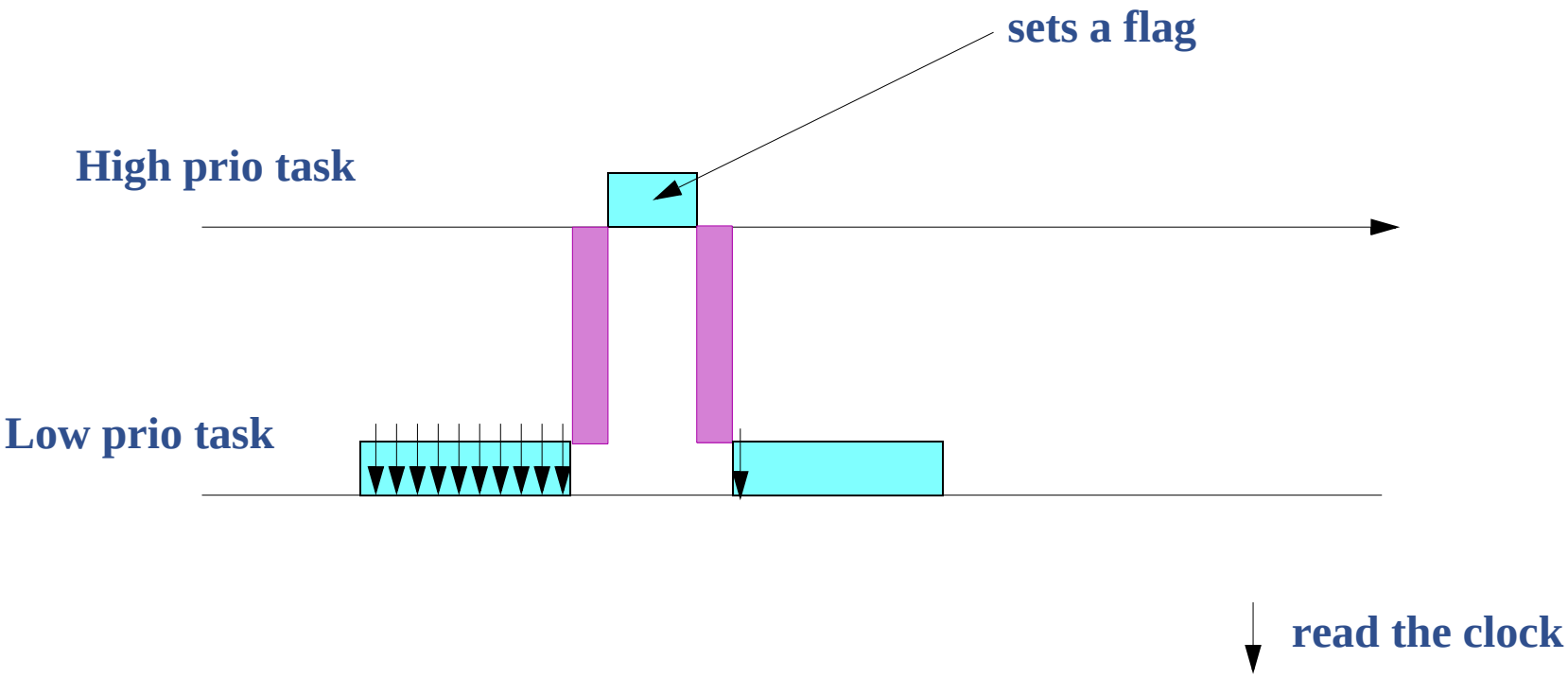
- New events, such as the system tick: modeled as new tasks. We need to know the period, priority and execution time of these events.
- Actions at caller's priority, such as library-level services, are added to the execution time of the task that invokes that service.
- Actions at a higher priority, such as kernel-level services, are treated as different actions within the task, and analyzed using the method for the case when priorities vary.
- Atomic actions, such as Interrupt Service Routines are also treated using the techniques for varying priorities. Notice that when analyzing high priority tasks, actions such as delay expirations are preemptive, and all their effects must be added in order to get the worst-case response of a given task. This may significantly impact the task's timing behavior.

System tick and other OS events

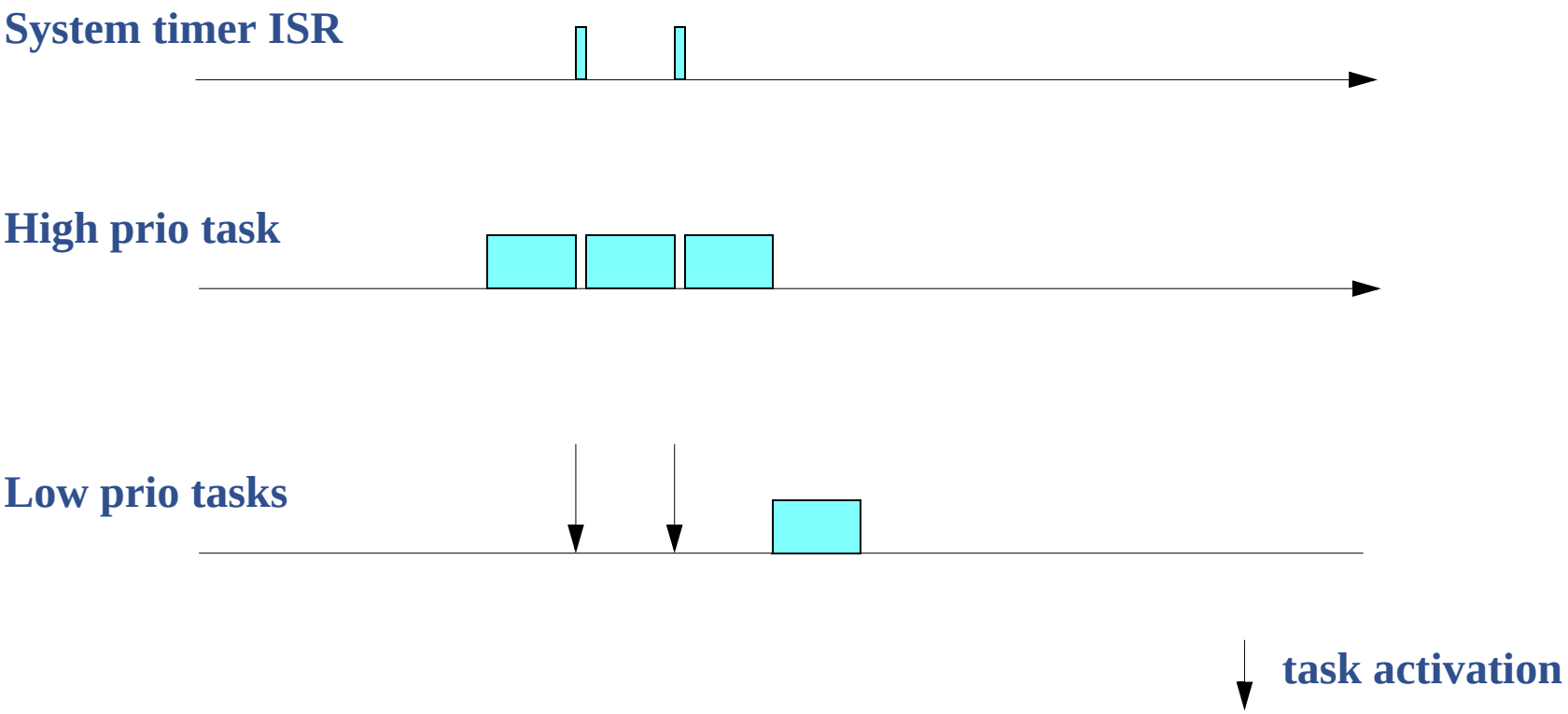
Can be measured with a program that constantly reads the clock and records differences large than usual



Measuring context switches



Delay expirations



Implicit Operating System Resources

The OS services may use some resources such as memory, OS databases, etc., in a mutually exclusive manner

To prevent unbounded priority inversion:

- use an operating system that uses the appropriate synchronization protocol,
- or preallocate all resources during initialization,
- or increase the task's priority while it is using such OS services,
- or lock a semaphore that is free of unbounded priority inversion while using such OS services.

7.7 Limited Representation of System Information

UNIVERSIDAD DE CANTABRIA

Coarse time granularity:

- sometimes makes system unschedulable
- modeled as “release jitter”

Insufficient number of priorities:

- causes a schedulability loss:

$$U_{\text{real}}(n) = \begin{cases} ln\left(\frac{2}{r}\right) + 1 - \frac{1}{r}, & 1 < r < 2 \\ \frac{1}{r}, & r \geq 2 \end{cases} \quad , \quad r = \left(\frac{T_{max}}{T_{min}}\right)^{\frac{1}{n_{prio}}}$$

Notes:

UNIVERSIDAD DE CANTABRIA

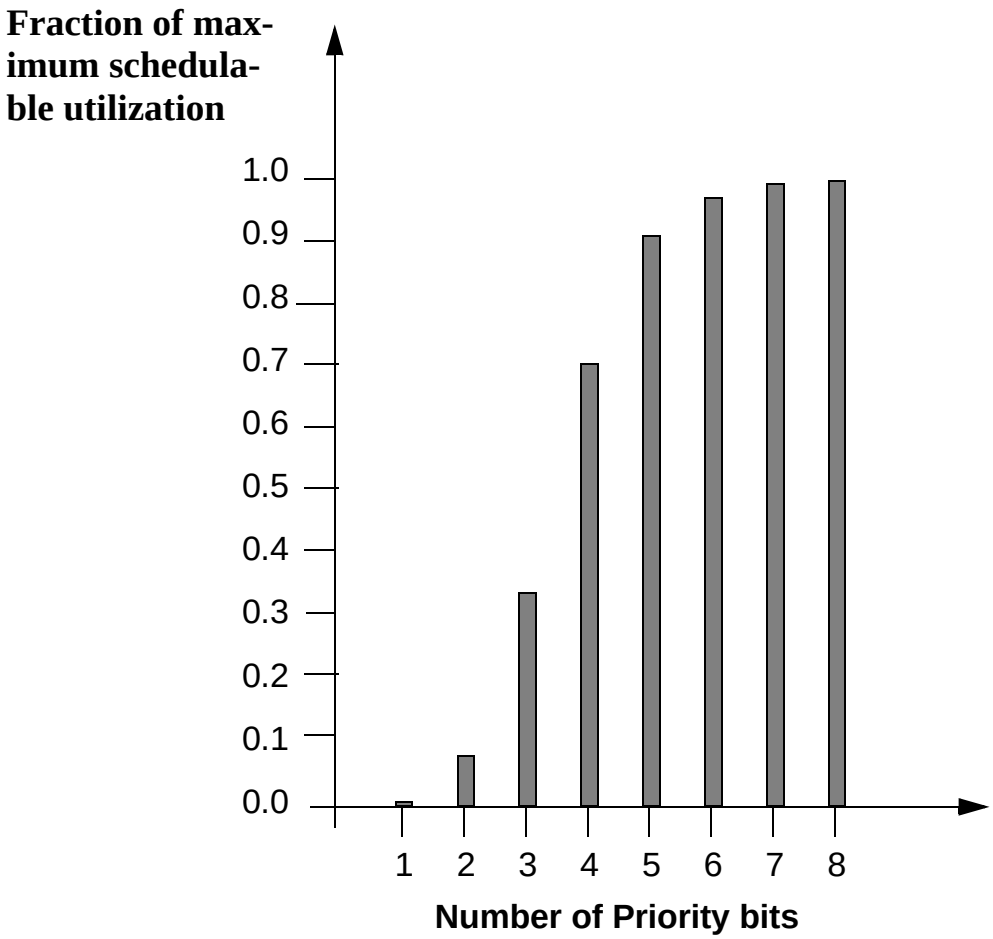
There are two system parameters that may have poor or insufficient representations in a particular implementation: time, and priority.

The granularity of time is an extremely important issue in real-time systems. The timing requirements cannot be smaller than the resolution of the timing services that are used to schedule the application.

Another important issue is to have a sufficiently large number of priorities. Optimum priority assignments such as rate-monotonic or deadline-monotonic require that each task is assigned a unique priority. When the number priorities is less than the number of tasks, some tasks that should have different priorities in an optimum priority assignment get the same priority. This implies a suboptimal priority assignment and consequently, a schedulability loss.

For large numbers of tasks, the real schedulability bound is a function of the ratio of the task periods and the number of priorities, as it is shown in the expression above.

Insufficient Number of Priorities



Notes:

The result that relates the schedulability loss due to an insufficient number of priorities for a large number of tasks is plotted here as a function of the number of bits used to represent priority. The relation (T_{max}/T_{min}) for this case was 100000.

We can see that when 32 or more priority levels are available, the schedulability is within 95% of the ideal case with infinite priorities. Therefore, it is considered that 32 priority levels is a sufficient number.

Presence or absence of OS features

UNIVERSIDAD DE CANTABRIA

Absolute and relative timers:

- absolute timers are necessary to implement periodic tasks and absolute timeouts
- relative timers are necessary to schedule time intervals

Enforcing execution capacities:

- allows detecting when the execution capacity has been exceeded
- increases the system robustness to timing errors

Disabling virtual memory swapping

- needed for predictability of memory access times

Notes:

We saw in the Ada implementation part of this chapter that the presence of a relative delay statement is not sufficient to ensure periodic activation of tasks. An absolute delay is needed in this case. However, in other cases where measuring a time interval is important, relative delays may be necessary.

Execution capacities should be enforced in a real-time system to assure that the results of the schedulability analysis are applicable in the final system, even in the presence of timing errors. If, for example, a task exceeds the expected worst-case execution time, other tasks that are working within their limits may miss their deadlines. If execution capacities are always enforced, the system can detect these situations and execute the appropriate error-handling procedures.

Virtual memory is a very important mechanism used to achieve good average-case behavior and a high independence of a particular system architecture, but it is unusable for tasks with real-time requirements because it destroys the predictability of memory access times. Real-time operating systems must provide a way to disable this mechanism, either for all the application, or for selected parts of it.

Scheduling Policies

UNIVERSIDAD DE CANTABRIA

Fixed priority schedulers:

- should have user-modifiable priorities.

Problems with multilevel schedulers:

- All tasks in a process (program) may be suspended when a blocking service is invoked; if possible, asynchronous services should be used.
- Priorities are not globally consistent across the system.

Notes:

UNIVERSIDAD DE CANTABRIA

The operating system used to schedule a real-time system that is analyzable using RMA must have a fixed-priority preemptive scheduler. In addition, it is very useful that the system allows task priorities to be modifiable from the application. This allows flexibility of the program structures, as well as the user-level implementation of the priority protect protocol, or sporadic servers.

In many systems the scheduler operates at different levels and this may cause severe effects in the schedulability of a real-time application. For example in UNIX, it is common to have a two-level scheduler. One level takes precedence over the other (the process scheduler). The other scheduler (for lightweight threads or Ada tasks) only schedules threads or tasks within the selected process. This approach has two major problems:

- When a thread or task invokes a blocking service, such as *read a file*, the complete program may be suspended. Usually we only want the calling task to be suspended. This problem can be prevented if the system provides equivalent asynchronous services, that do not block.
- Priorities are not globally consistent across the system. A high priority task in process A cannot preempt a low priority task in process B. This makes priority assignment a very difficult problem in these systems.

In general, a global or mixed scheduler is preferred for real-time applications, even tough a pure two-level scheduler may provide better average-case performance.

Ada Real-Time Features

Real-time Features	Support in Ada
Priority preemptive scheduler	Default policy in the RT Annex
Sufficient number of priorities	At least 31 priority levels defined in the Real-Time Annex
Priorities modifiable at run-time	Yes, in the Real-Time Annex
Known context switch times, etc.	Not provided
Synchronization primitives free of unbounded priority inversion	Immediate priority ceiling protocol for protected objects
Periodic task activation	Absolute delay (delay until)
Execution-time budgets	Ada.Execution_Time
Sporadic server	Not provided
Synchronization primitives free of remote blocking	Immediate priority ceiling protocol for protected objects

Real-Time Features in POSIX

Real-time Features	Support in POSIX
Priority preemptive scheduler	SCHED_FIFO scheduling policy
Sufficient number of priorities	At least 32 priority levels
Priorities modifiable at run-time	Yes
Known context switch times, etc.	Not provided
Synchronization primitives free of unbounded priority inversion	PRIO_INHERIT and PRIO_PROTECT for mutexes
Periodic task activation	Yes
Execution-time budgets	Execution time clocks and timers
Sporadic server	SCHED_SPORADIC scheduling policy
Synchronization primitives free of remote blocking	Immediate priority protocol for mutexes