Reliable Handling of Real-Time Scheduling Attributes on Multiprocessor Platforms in Ada 2012

Sergio Sáez, Jorge Real, Alfons Crespo

Universitat Politécnica de Valéncia, Spain

19th International Conference on Reliable Software Technologies - Ada-Europe 2014 - Paris, France, June 23-27, 2014

Outline

- Introduction
- Data type for real-time attributes
- Problem statement
- Design alternatives and their implementation
- Conclusions

Introduction

- Real-time scheduling attributes determine how resources are allocated to tasks
  - Priority, deadline, CPU, …
- Ada 2012 supports their handling
  - System.Multiprocessors.Dispatching_Domains
    - Querying and setting a task's CPU
    - Delay_Until_And_Set_CPU
  - Ada.Dispatching.EDF
    - Querying and setting a task's deadline
    - Delay_Until_And_Set_Deadline
  - Ada.Dynamic_Priorities
    - Querying and setting a task's priority
    - NO *Delay_Until_And_Set_Priority*

Introduction

- However, it is not possible to atomically modify several attributes at a time…
  - …neither immediately nor after a delay
- This would be most useful in cases such as
  - Job partitioning – next activation
  - Task splitting, Dual-priority – after some time
  - Mode changes – upon mode change request

Data type for real-time attributes

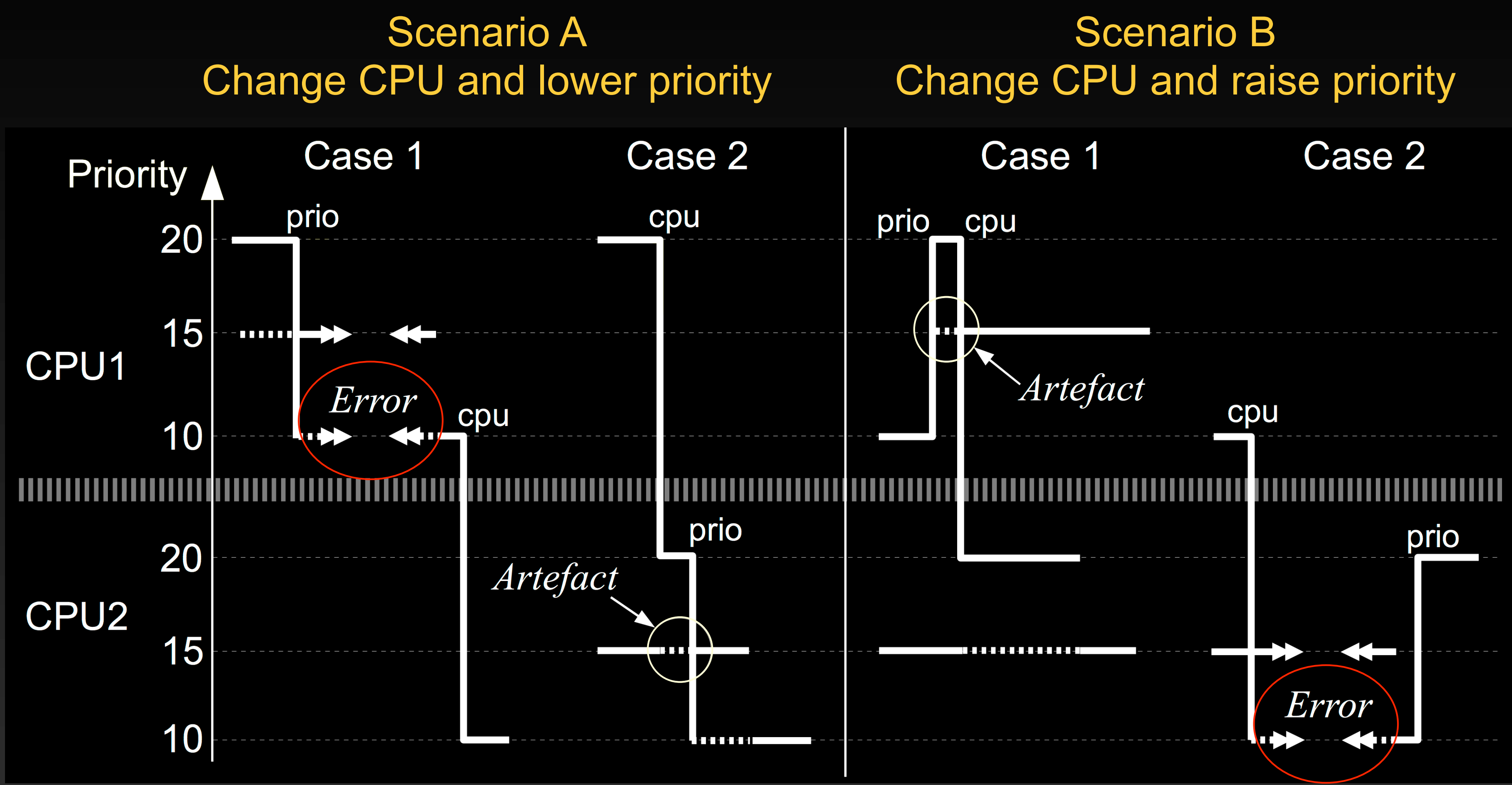
```
package Ada_Real_Time.Scheduling_Attributes is

  type Scheduling_Attributes is tagged private;
  procedure Set_Priority  (SP: in out Scheduling_Attributes;    Prio: Any_Priority);
  function Get_Priority     (SP: in        Scheduling_Attributes)   return Any_Priority;
  procedure Set_CPU       (SP: in out Scheduling_Attributes;    CPU_Nr: CPU_Range);
  function Get_CPU          (SP: in        Scheduling_Attributes)   return CPU_Range;
  procedure Retrieve_Scheduling_Attributes  (SP: out Scheduling_Attributes;
                                             T_Id: Task_Id := Current_Task);

  type Any_Scheduling_Attributes is access all Scheduling_Attributes'Class;
  procedure Apply_Scheduling_Attributes (SP: Any_Scheduling_Attributes;
                                         T_Id: Task_Id := Current_Task);
  procedure Delay_Until_And_Apply_Scheduling_Attributes (SP: Any_Scheduling_Attributes;
                                                         Delay_Until_Time: Time);
private
  type Scheduling_Attributes is tagged record
    Prio: Any_Priority := Default_Priority;
    CPU_Nr: CPU_Range := Not_A_Specific_CPU;
  end record;

  procedure Enforce_Scheduling_Attributes (SP: Scheduling_Attributes; T_Id: Task_Id);
end Ada_Real_Time.Scheduling_Attributes;
```

Data type for real-time attributes

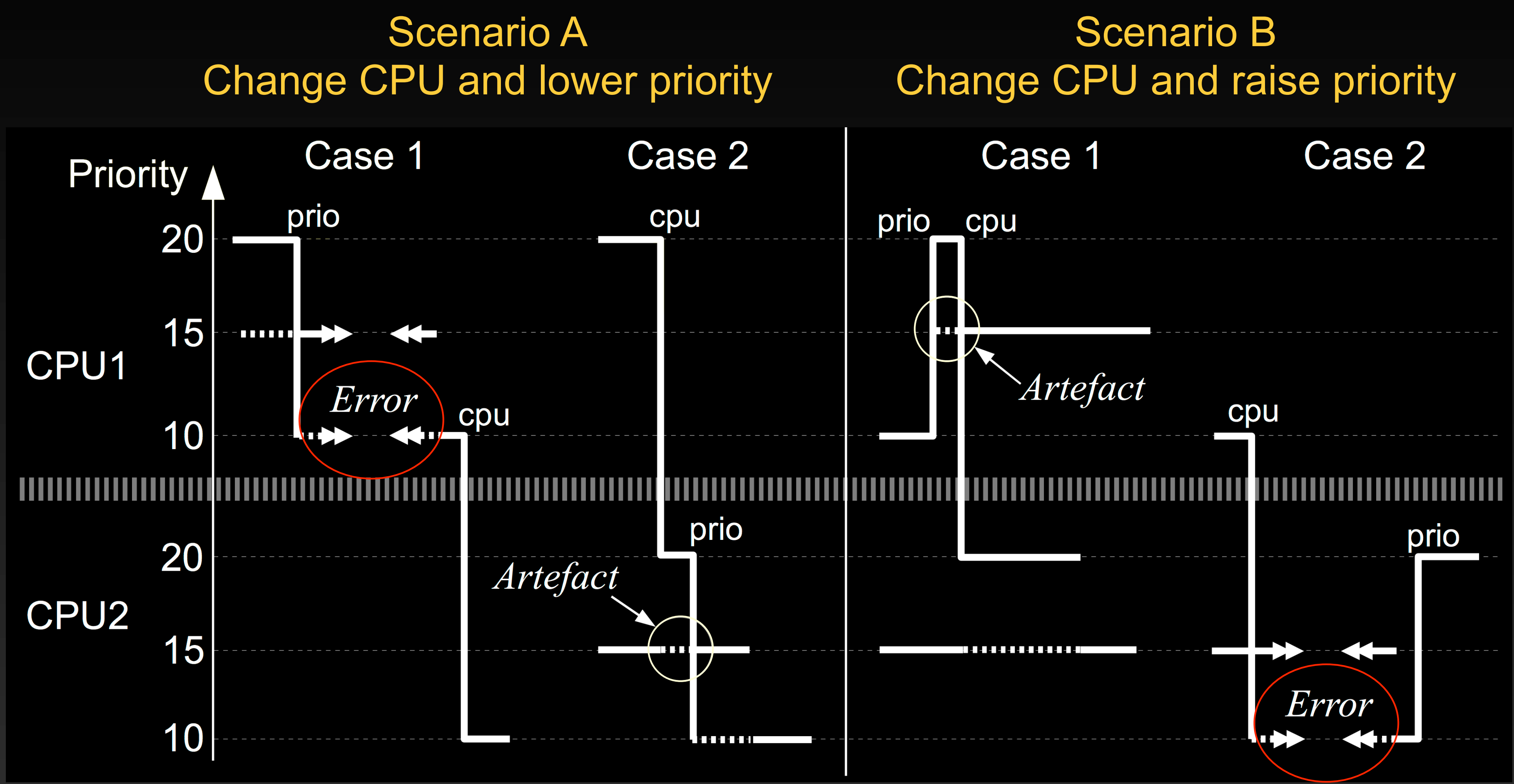
- Root subprogram ultimately applying the change of attributes
- It has to be implemented for each extension of the type
  - For example:

```
procedure Enforce_Scheduling_Attributes (SP: Scheduling_Attributes; T_Id: Task_Id) is
begin
  Ada.Dynamic_Priorities.Set_Priority (Priority => SP.Prio; T => T_Id);
  System.Multiprocessors.Dispatching_Domains.Set_CPU (CPU => SP.CPU_Nr; T => T_Id);
end Enforce_Scheduling_Attributes;
```

Problem statement



Problem statement



- Order of enforcement matters
- Even in the correct order, artefacts occur since changes are not atomic

Problem statement

- Goal: reliably change several attributes at a time
  - Removing the *errors*
    - Attributes must be changed in the right order
    - Changing several attributes requires atomicity
  - If *artefacts* cannot be removed, they must be precisely identified
    - Minimal, bounded duration
    - Affected CPU must be known
- Using *only* Ada 2012
- Plan: explore use of Ada mechanisms that provide atomicity

Design alternatives

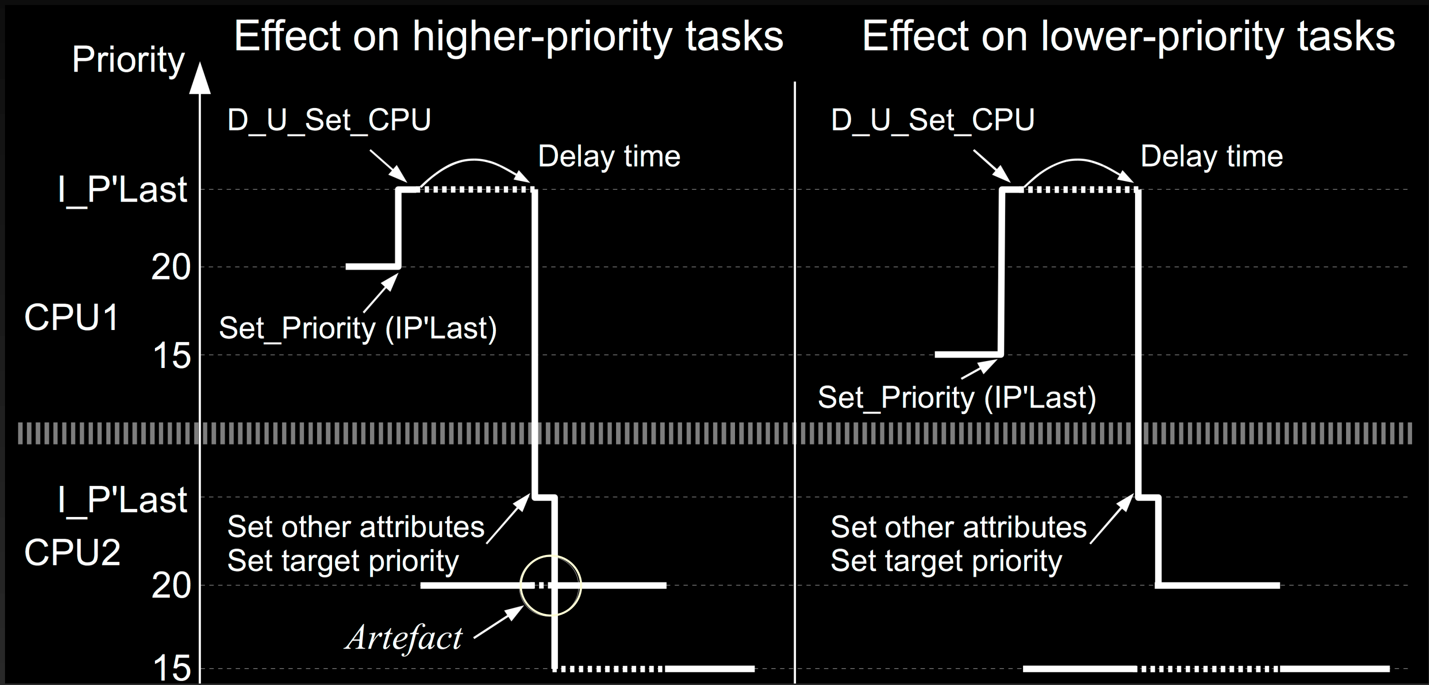
- Plan: explore Ada mechanisms for atomicity
  - 1 Use a protected operation
  - 2 Self-change from the highest priority
  - 3 Use timing event handlers
  - 4 Use rendezvous with server task at high priority

Design alternative 1: Protected operation

- Changing priority, deadline, CPU, are task dispatching points…
- …but deferred until the end of protected action
  - Scheduling errors would seem to disappear…
- *Delay_Until_And_Set_Scheduling_Atributes* could be obtained in combination with requeue to a closed entry, later opened with a timing event
- But the runtime will ultimately apply the changes in some order, hence reproducing errors and artefacts
  - Reliable application-level solution not guaranteed by PO

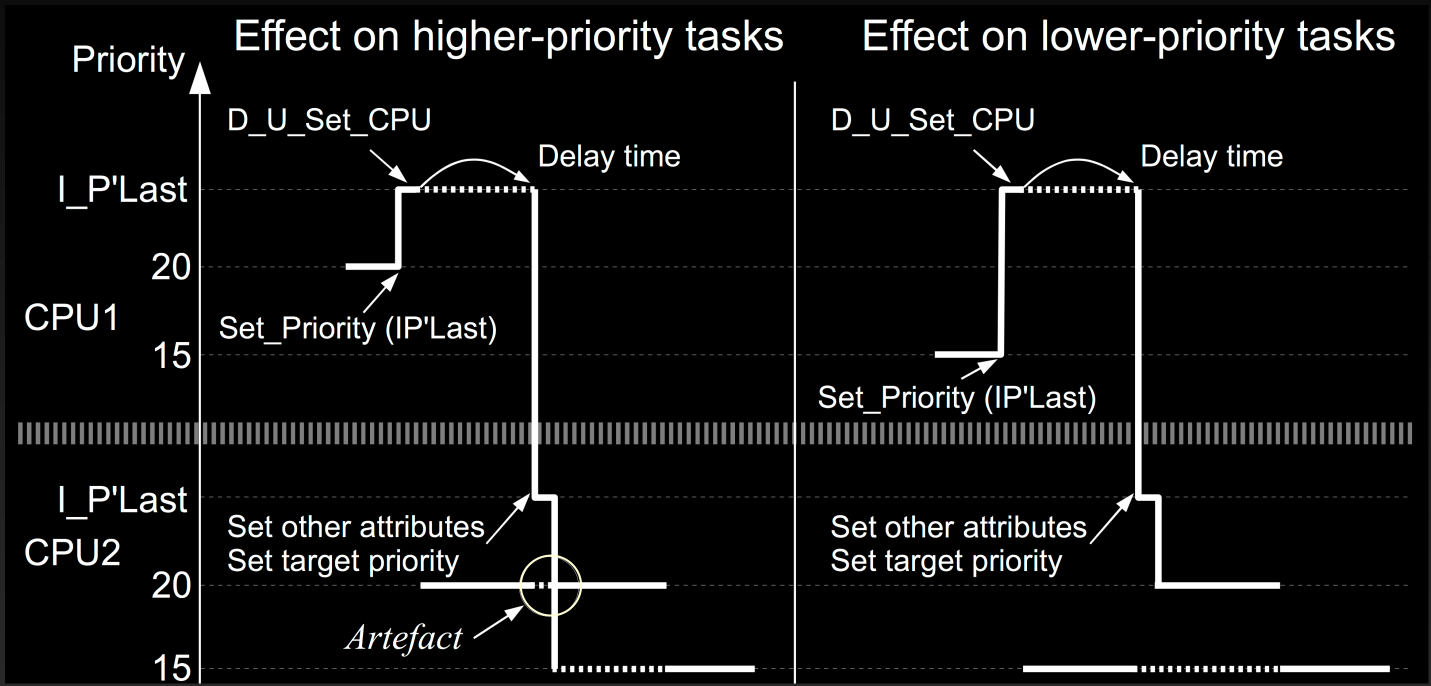
Design alternative 2: Self-change at IP'Last

- Task changes its own attributes from a sufficiently high priority (e.g. Interrupt_Priority'Last)



Design alternative 2: Self-change at IP'Last

- Task changes its own attributes from a sufficiently high priority (e.g. Interrupt_Priority'Last)



- Artefacts affect higher-priority tasks on the destination CPU
- *Regular* interference on lower-priority tasks

Design alternative 2: Self-change at IP'Last

- Implementation of delayed attribute change

```
procedure Delay_Until_And_Apply_Scheduling_Attributes (
                        SP: Any_Scheduling_Attributes;
                        Delay_Until_Time: Time) is
begin
  Set_Priority (Interrupt_Priority'Last );                  -- Rise caller's priority to highest
  Delay_Until_And_Set_CPU (Delay_Until_Time, SP.CPU_Nr);

  -- Caller wakes up from delay in the destination CPU, still with the highest priority

  SP.Enforce_Scheduling_Attributes (Current_Task);          -- Update other attributes
  Set_Priority (SP.Prio);               -- Decrease caller's priority down to target priority

end Delay_Until_And_Apply_Scheduling_Attributes;
```

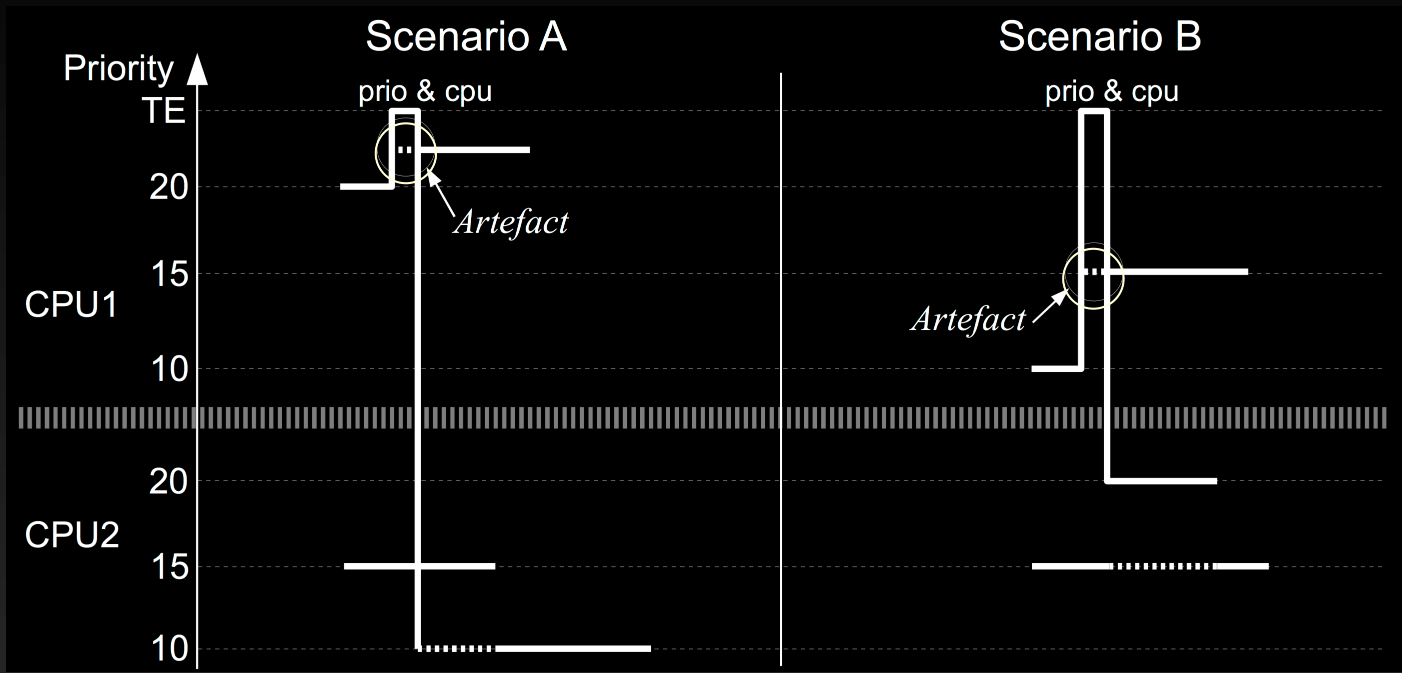
- Drawback: artefact and added interference may occur too often in destination CPU in case of bursts of migrating tasks

Design alternative 3: Timing events

- Timing events are handled at the highest priority
  - Promising, in terms of atomicity
- Plus, they can be programmed for the future
  - Handy for deferred attribute changes
    - For immediate effect, use a time in the past
- Plus, efficient implementation (vs. PO's)
- A TE handler is a protected procedure with the highest priority (IP'Last, under ceiling locking)
  - But we said PO's are not a good idea…
  - But using a TE, we apply changes to *another* task
    - All changes applied at highest priority → no re-schedule

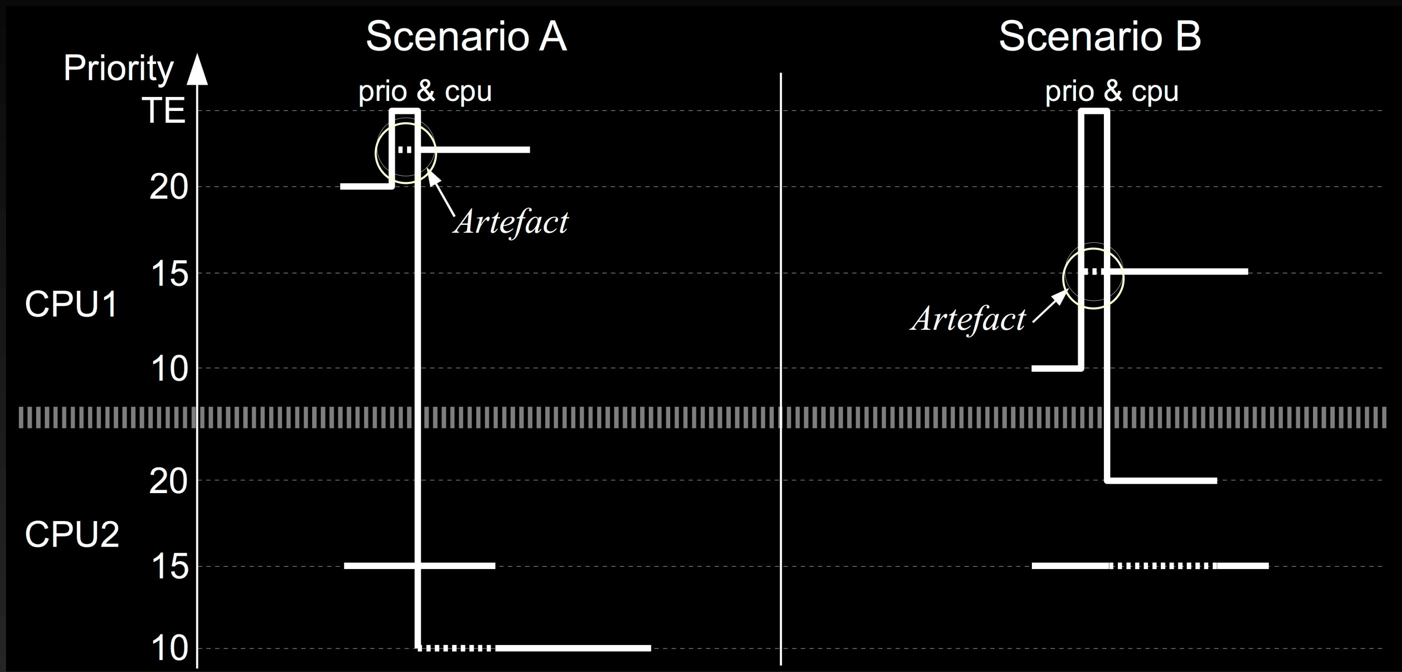
Design alternative 3: Timing events

- Timing event scenarios



Design alternative 3: Timing events

- Timing event scenarios



- Errors *may* disappear, artefacts are still present
- Drawback: effects impact an unknown CPU
  - CPU affinity for timing events could solve the issue

Design alternative 3: Timing events

- Implementation scheme:
  - One PO with TE handler per task (with changing attributes)
  - *Owner* task of the PO is known by the PO

Design alternative 3: Timing events

- Use of (hypothetical) timing event affinities

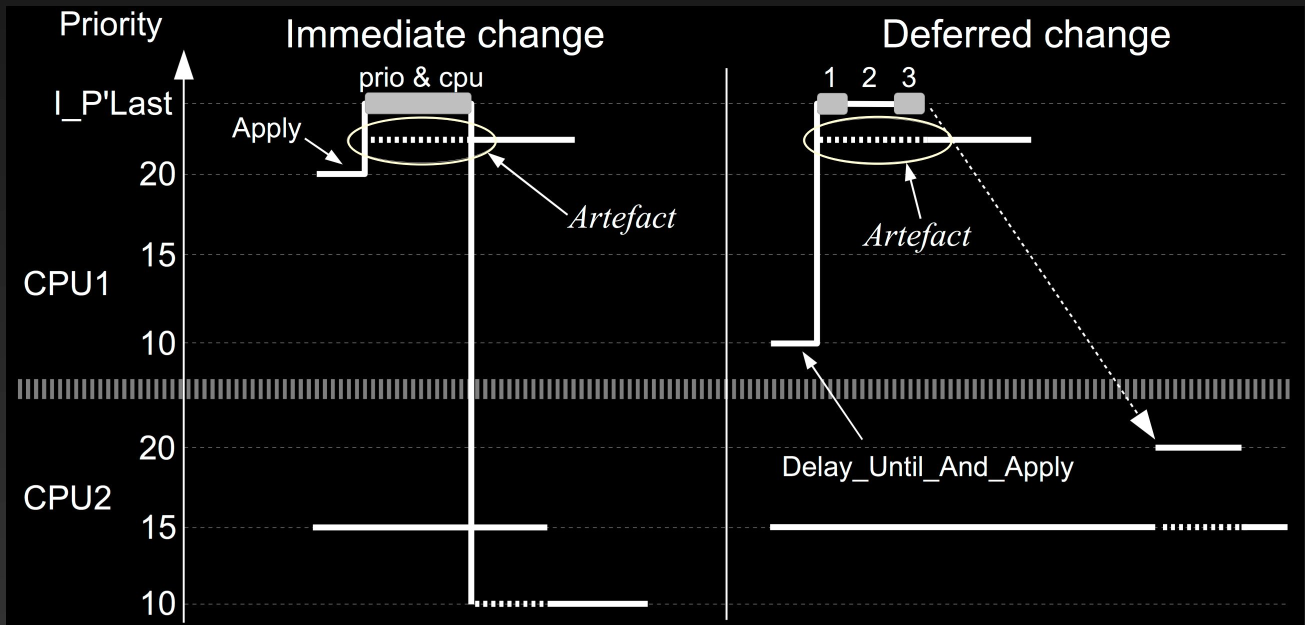
```
protected body Scheduling_Manager is

   entry Apply_Scheduling_Attributes (SP: Any_Scheduling_Attributes) when True is
   begin
      Task_Waiting := True; -- Barrier for entry Wait
      -- An immediate timing event is programmed...
      Timing_Ev.Set_Handler (Time_First, Handler'Access, SP.Get_CPU);
      -- The client task is requeued to Wait until Handler updates its attributes
      requeue Wait;
   end Apply_Scheduling_Attributes;

   procedure Handler (Event : in out Timing_Event) is
   begin
      Sched_Params.Enforce_Scheduling_Attributes (Owner_Task);
      Task_Waiting := False;
   end Handler;

   entry Wait when not Task_Waiting is
   begin
      null;
   end Wait;

end Scheduling_Manager;
```

Design alternative 4: Server task & rendezvous

- A server task in charge of changing another (client) tasks' attributes
  - The server runs at the highest priority (IP'Last)
- Client calls the appropriate server entry
  - Immediate or deferred change
- We want both client and server at IP'Last, even some time after the rendezvous
  - Hence client raises to IP'Last before calling the server
- During the rendezvous, the calling task is blocked → Its attributes are changed while it is not running
  - Eliminates chances for glitches

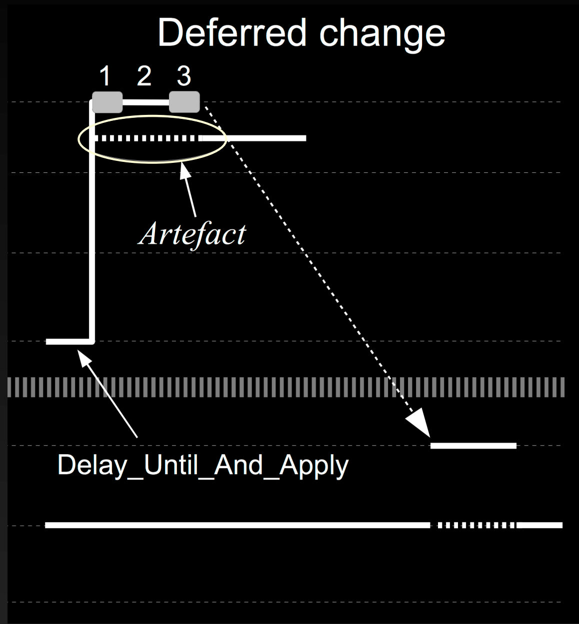
Design alternative 4: Server task & rendezvous

- Immediate change is relatively simple
  - Client raises prio to IP'Last
  - Client calls server → caller blocked, rendezvous starts at IP'Last
  - Server enforces new client's attributes while client is blocked
  - Server loops back to selective accept waiting for new calls
  - Client is now at the new priority queue of the new CPU



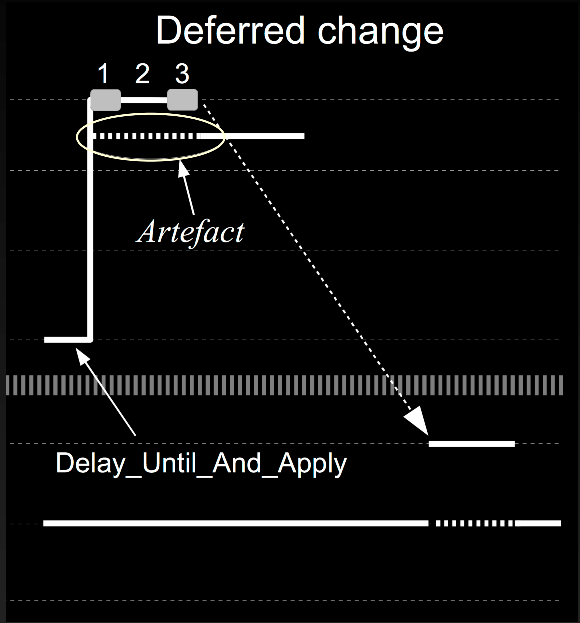
Design alternative 4: Server task & rendezvous

- Deferred change requires carefully considering the delay
  - Client raises priority to IP'Last and calls server
    - Caller blocked, rendezvous starts at IP'Last
  - Step 1: Server makes local copy of new attributes and rendezvous ends
    - Client and server both at IP'Last
  - Step 2: Server yields control back to client
    - Client executes Delay_Until_And_Set_CPU and suspends
  - Step 3: Control goes back to server
    - Server then enforces new client's attributes



Design alternative 4: Server task & rendezvous

- Deferred change requires carefully considering the delay
  - Client raises priority to IP'Last and calls server
    - Caller blocked, rendezvous starts at IP'Last
  - Step 1: Server makes local copy of new attributes and rendezvous ends
    - Client and server both at IP'Last
  - Step 2: Server yields control back to client
    - Client executes Delay_Until_And_Set_CPU and suspends
  - Step 3: Control goes back to server
    - Server then enforces new client's attributes



- No scheduling errors
- Artefact is blocking time for tasks of higher priority
- It only affects tasks in origin CPU
  - This is an important advantage with respect to TE

Design alternative 4: Server task & rendezvous

- Server task implementation: spec

```
task type Scheduling_Manager_Type (CPU_Nr: CPU := Next_CPU)
  with Interrupt_Priority   => Interrupt_Priority'Last,
       CPU                  => CPU_Nr is

  entry Apply_Attributes_Immediately (
          SP: Any_Scheduling_Attributes; T_Id: Task_Id);

  entry Apply_Attributes_On_Suspend (
          SP: Any_Scheduling_Attributes;  T_Id: Task Id);

end Scheduling_Manager_Type;
```

Design alternative 4: Server task & rendezvous

- Server task implementation: body

```
task body Scheduling_Manager_Type is
  Sched_Param: Any_Scheduling_Attributes; Target_Task: Task_Id;
begin
  loop
    select
      accept Apply_Attributes_Immediately (SP: Any_Scheduling_Attributes; T_Id: Task_Id) do
        SP.Enforce_Scheduling_Attributes (T_Id); -- Change task's attributes
      end Apply_Attributes_Immediately ;
    or
      accept Apply_Attributes_On_Suspend (SP: Any_Scheduling_Attributes;  T_Id: Task Id) do
        Target_Task := T_Id; -- Stores the target task and new attributes
        Sched_Param := SP;  -- Step 1
      end Apply_Attributes_On_Suspend;
      delay 0.0; -- Step 2: Yield to allow client task to execute "delay until"
      -- Step 3: Change the attributes of the suspended client task
      Sched_Param.Enforce_Scheduling_Attributes (Target_Task);
    or
      terminate;
    end select;
  end loop;
end Scheduling_Manager_Type;
```

Design alternative 4: Server task & rendezvous

- Class-wide subprograms in Ada_R_T.Scheduling_Attributes

```
procedure Apply_Scheduling_Attributes (SP: Any_Scheduling_Attributes;
                                        T_Id : Task_Id := Current_Task) is
begin
   Set_Priority (Interrupt_Priority'Last);
   Scheduling_Manager (Current_CPU).Apply_Parameters_Immediately (SP, T_Id);
end Apply_Scheduling_Attributes;


procedure Delay_Until_And_Apply_Scheduling_Attributes (SP: Any_Scheduling_Attributes;
                                                        Delay_Until_Time: Time) is
begin
   Set_Priority (Interrupt_Priority'Last);  -- Rise priority to IP'Last
   Scheduling_Manager (Current_CPU).Apply_Attributes_On_Suspend (SP, Current Task);
   delay until Delay_Until_Time; –– Attributes will be changed during suspension (Step 2)
   -- Task will wake up with the new attributes applied
end Delay_Until_And_Apply_Scheduling_Attributes;
```

Conclusions

- Changing several scheduling attributes at a time is challenging
  - Especially when the CPU is one of them
- From the four application-level schemes explored we conclude…

Conclusions

- …
  - A PO-based, application-level scheme does not guarantee absence of scheduling issues in itself
    - Ultimately depends on how the runtime/OS enforces the attributes at the end of a protected action
  - Self-changing the attributes from IP'Last introduces remote interference in the destination CPU
    - Bursts of migrating tasks challenge schedulability
  - Artefacts introduced by the timing-event scheme affect unknown CPUs
    - Could be mitigated if timing events had affinity
  - The server task approach is the most reliable, although it's not for free…

Conclusions

- ...
  - The server task approach is the most reliable, although it's not for free…
    - Requires up to one server task per CPU
    - The interference affects only the origin CPU
      - Can be accounted for as blocking time for tasks of hp(τ)
    - Changes are applied while task is suspended
      - It will wake up with the new attributes enforced
    - There is a runtime cost involved in the double context switch
      - But it is predictable
- All in all…
  - Doable in Ada 2012
  - Timing event affinities would enable more efficient solution