dit
UPM

Esquemas de tareas de tiempo real

Juan Antonio de la Puente
DIT/UPM

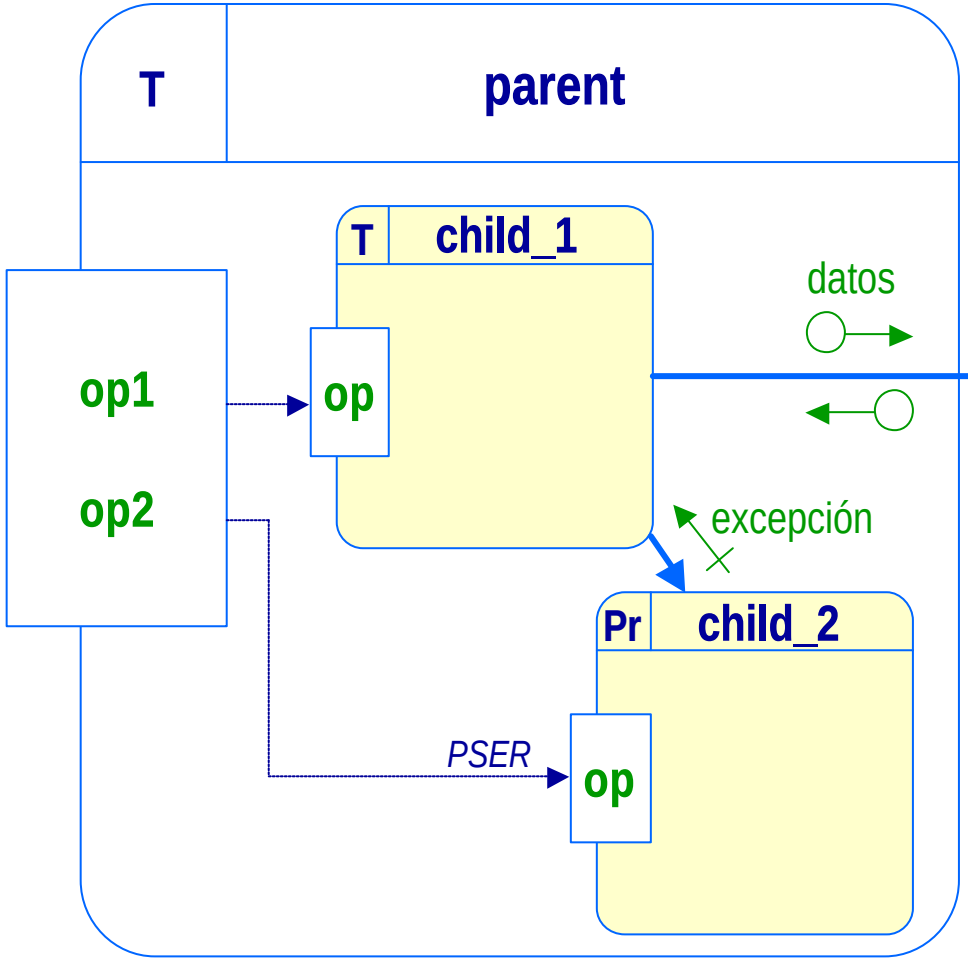
Objetivos

- Reconocer un conjunto de esquemas básicos para construir sistemas de tiempo real
  - Tareas periódicas
  - Tareas esporádicas
  - Recursos compartidos
  - Sincronización
- Veremos esquemas basados en HRT-HOOD
  - También algunos esquemas más generales

HRT-HOOD

- Un sistema de tiempo real se representa mediante un conjunto de objetos
  - Objetos no terminales: se descomponen en otros más simples
  - Objetos terminales: no se pueden descomponer más
- Un objeto se caracteriza por sus:
  - Operaciones: que pueden invocar otros objetos (modelo de llamada a procedimiento)
  - Comportamiento
    - » Activo (cíclico, esporádico) / pasivo / protegido
    - » Restricciones en las operaciones
      - Sincronización
      - Estado

Objetos y relaciones



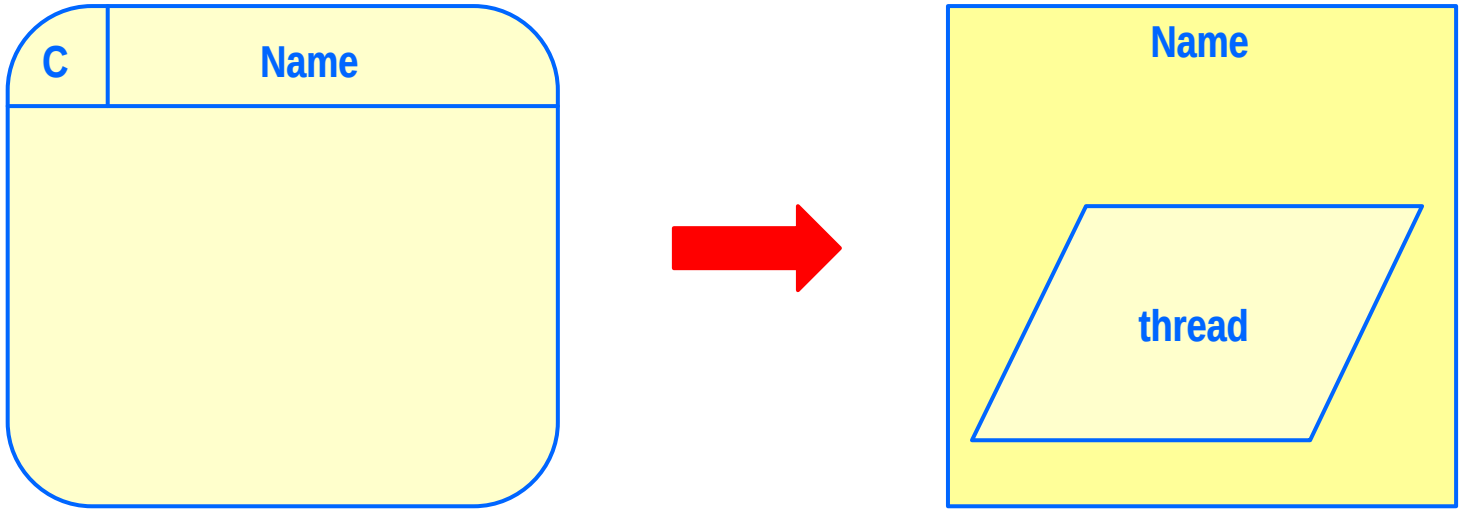
El objeto **parent** ***contiene***
los objetos **child_1** y **child_2**

parent ***usa*** **uncle**
child_1 ***usa*** **child_2** y **uncle**

child_1.op ***implementa*** **parent.op1**
child_2.op ***implementa*** **parent.op2**

child_2.op tiene una
restricción de sincronización

Objeto cíclico



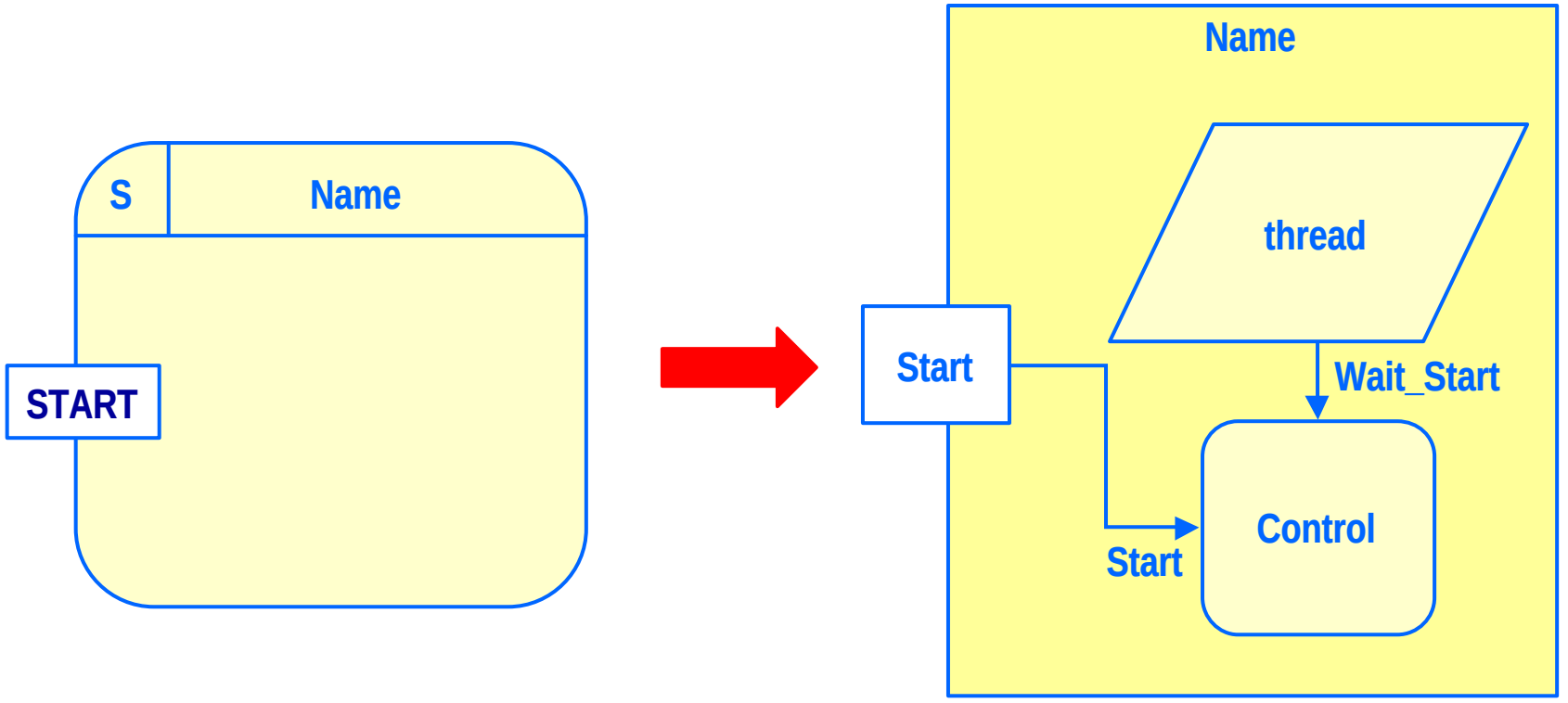
Objeto cíclico: realización

```
with Ada.Real_Time; use Ada.Real_Time;
with System;        use System;
package Name is
  Start_Time      : Time      := ... ;
  Period          : Time_Span := ... ;
  Thread_Priority : Priority  := ... ;
  pragma Elaborate_Body (Name);
end Name;
```

- No hay operaciones visibles
  - Elaborate_Body indica que el paquete tiene cuerpo
- Atributos temporales
  - Start_Time
  - Period
  - Thread_Priority
- No se detecta si se excede el plazo de ejecución o el tiempo de cómputo

```
with Ada.Real_Time; use Ada.Real_Time;
-- other context clauses
package body Name is

  task Thread is
    pragma Priority (Thread_Priority);
  end Thread;

  task body Thread is
    Next_Time : Time := Start_Time ;
  begin
    -- thread initialization
    loop
      delay until Next_Time;
      Periodic_Action;
      Next_Time := Next_Time + Period;
    end loop;
  end Thread;

begin
  -- package initialization
end Name;
```

Objeto esporádico



Objeto esporádico: realización

```
with System; use System;
package Name is

  Thread_Priority : Priority  := ... ;

  procedure Start;

end Name;
```

- La única operación visible es **Start**
- No se vigila el cumplimiento de la separación mínima entre sucesos
- No se detecta si se excede el plazo de ejecución o el tiempo de cómputo

```
with Ada.Synchronous_Task_Control;
use  Ada.Synchronous_Task_Control;
-- other context clauses
package body Name is

  task Thread is
    pragma Priority (Thread_Priority);
  end Thread;

  Control : Suspension_Object;

  task body Thread is
  begin
    -- thread initialization
    loop
      Suspend_Until_True(Control);
      Sporadic_Action;
    end loop;
  end Thread;

  procedure Start is
  begin
    Set_True (Control);
  end Start;

begin
  -- package initialization
end Name;
```

Objeto esporádico: Sincronización con objeto protegido (1)

```
with System; use System;
package Name is

  Thread_Priority : Priority  := ... ;
  Control_Priority: Priority  := ... ;

  procedure Start;
```

```
private

 protected Control is
   pragma Priority (Control_Priority);
   procedure Start;
   entry     Wait_Start;
 private
    Started : Boolean := False;
 end Control;

 procedure Start
    renames Control.Start;
end Name;
```

- El objeto de sincronización se sustituye por un objeto protegido
- Funcionalmente es equivalente, aunque menos eficiente
- Es fácil de extender para tener en cuenta la separación mínima entre sucesos

Objeto esporádico: Sincronización con objeto protegido (2)

```
with System; use System;
package body Name is
   task Thread is
      pragma Priority (Thread_Priority);
   end Thread;

   task body Thread is
   begin
      -- thread initialization
      loop
         Control.Wait_Start;
         Sporadic_Action;
      end loop;
   end Thread;
```

```
   protected body Control is

      procedure Start is
      begin
         Started := True;
      end Start;

      entry Wait_Start when Started is
      begin
         Started := False;
      end Wait_Start;

   end Control;

begin
   -- package initialization
end Name;
```

Objeto esporádico: Separación mínima (1)

```
with Ada.Real_Time; use Ada.Real_Time;
with System; use System;
package Name is

  Separation      : Time_Span := ... ;
  Thread_Priority : Priority  := ... ;
  Control_Priority: Priority  := ... ;

  procedure Start;
```

```
private

 protected Control is
   pragma Priority (Control_Priority);
   procedure Start;
   entry Wait_Start
    (Start_Time : out Time);
 private
   Started    : Boolean := False;
   Event_Time : Time;
 end Control;

 procedure Start
   renames Control.Start;
end Name;
```

- El objeto **Control** registra el tiempo en que se hace **Start**
  (no es necesariamente el mismo en que se ejecuta la acción esporádica)

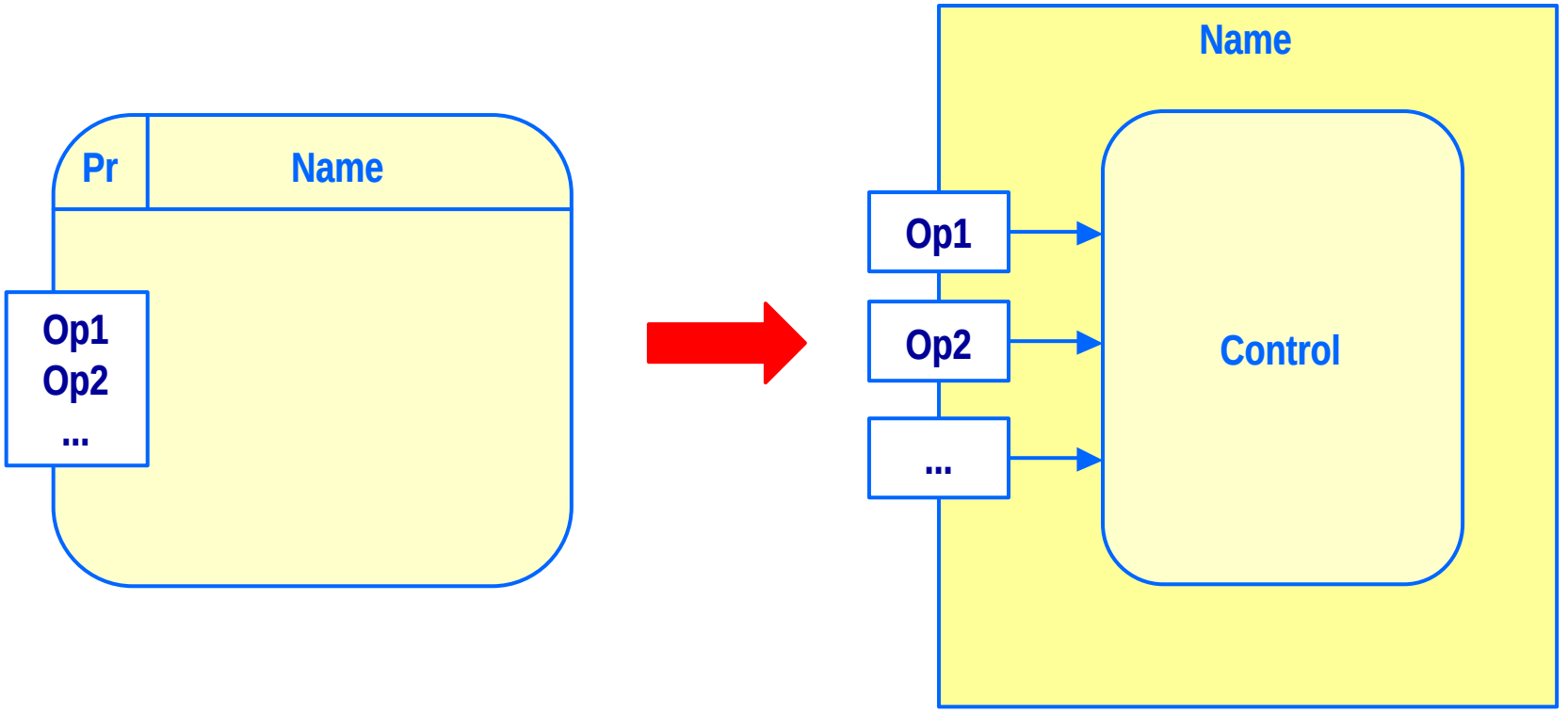
Objeto esporádico: Separación mínima (2)

```
with Ada.Real_Time; use Ada.Real_Time;
with System; use System;
package body Name is
  task Thread is
    pragma Priority (Thread_Priority);
  end Thread;

  task body Thread is
    Start_Time : Time;
  begin
    -- thread initialization
    loop
      Control.Wait_Start(Start_Time);
      Sporadic_Action;
      delay until
        Start_Time + Separation;
    end loop;
  end Thread;
```

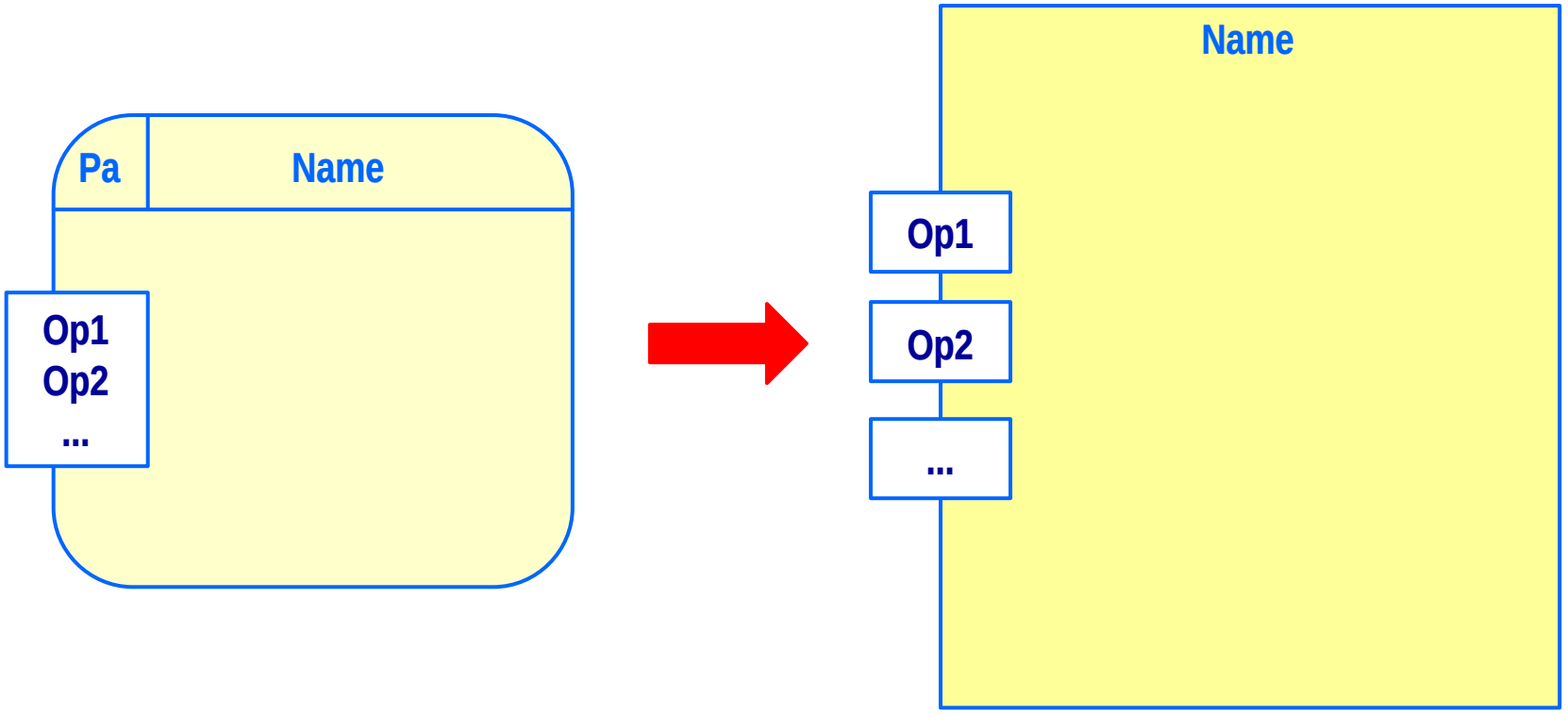
```
  protected body Control is

    procedure Start is
    begin
      Started := True;
    end Start;

    entry Wait_Start
     (Start_Time : out Time)
    when Started is
    begin
      Started := False;
      Start_Time := Event_Time;
    end Wait_Start;

  end Control;

begin
  -- package initialization
end Name;
```

- Inconveniente: dos cambios de contexto
  - si se está seguro de la separación mínima es mejor el esquema anterior

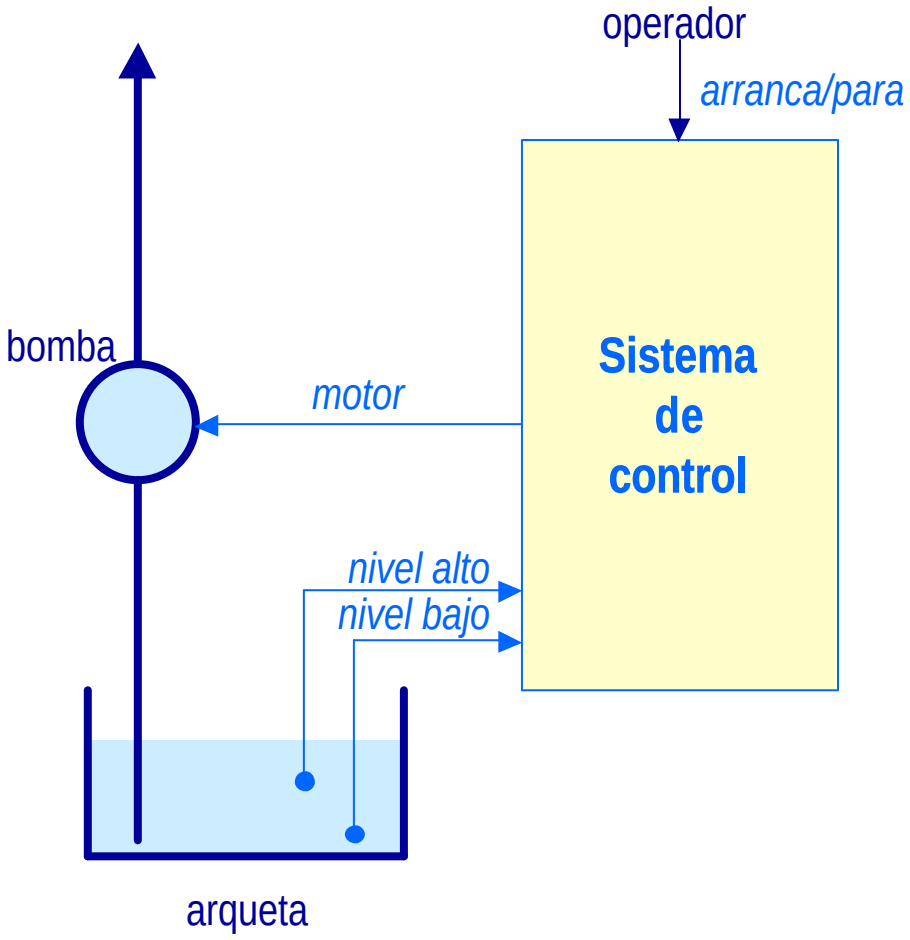
Objeto protegido



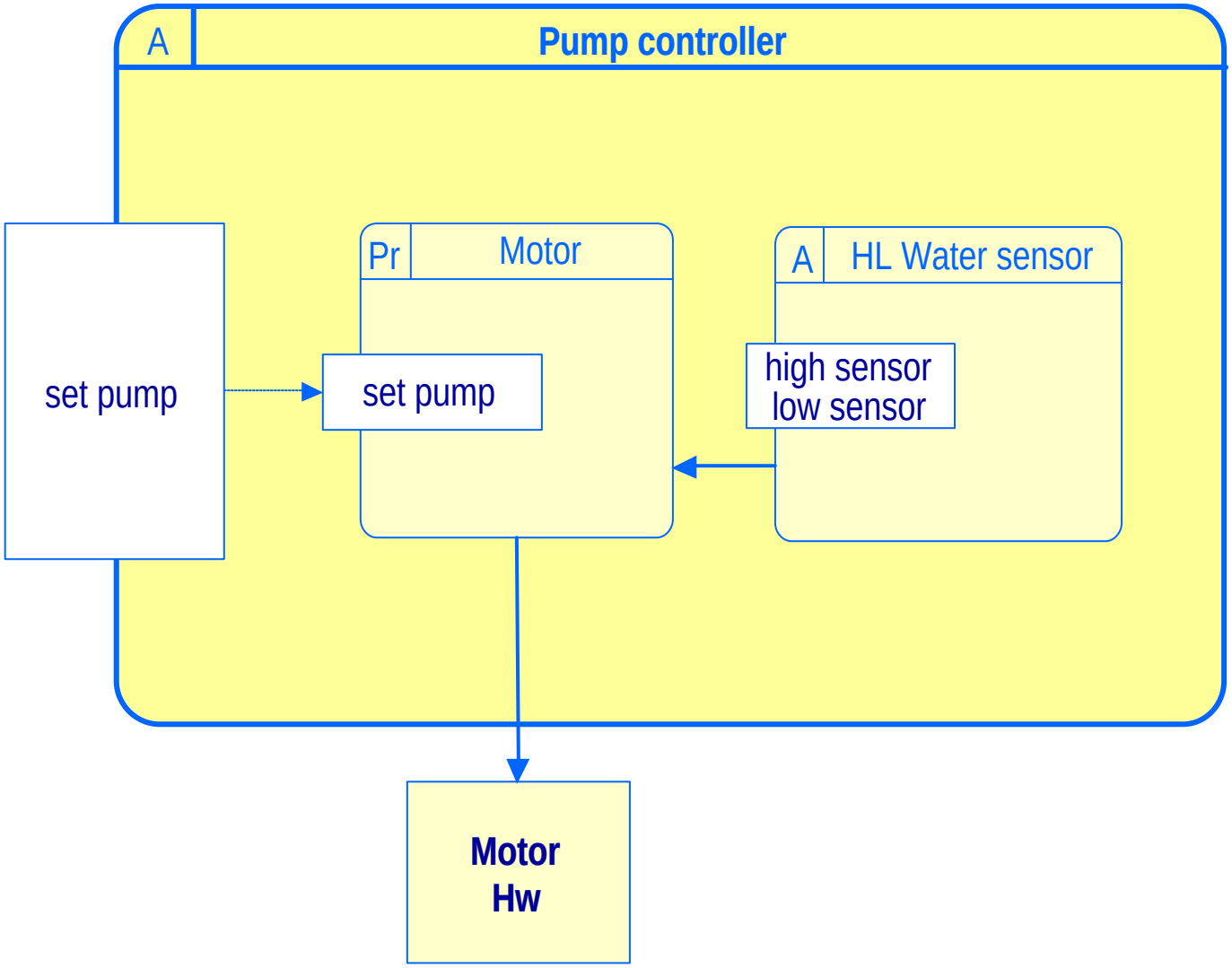
Objeto pasivo



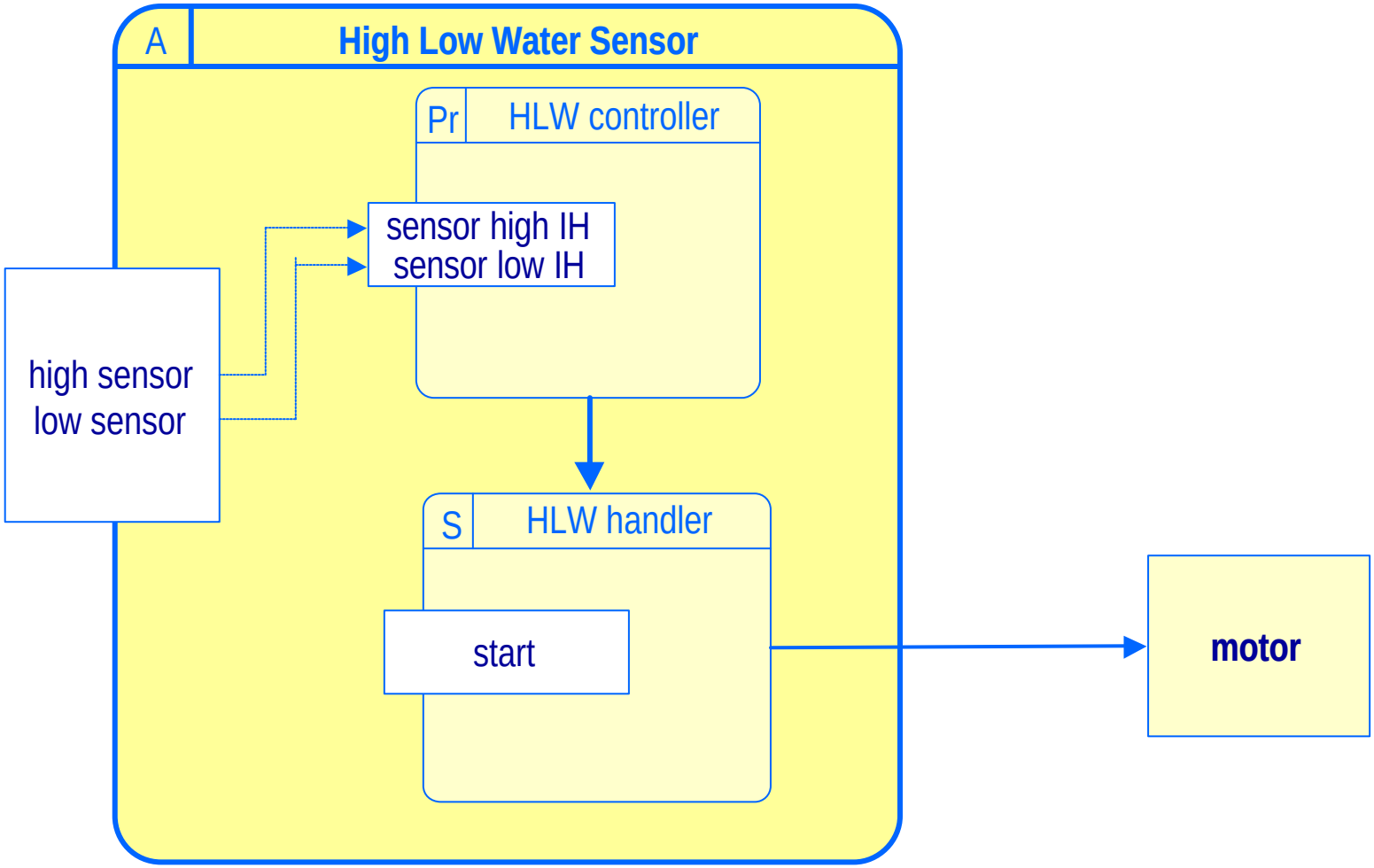
Ejemplo: control de una bomba de agua



Diseño con HRT-HOOD (1)



Diseño con HRT-HOOD (2)



Realización: Motor (1)

```
package Motor is -- protected object
  type Pump_Status    is  (On, Off);
  procedure Set_Pump (To : Pump_Status);
end Motor;
```

```
with Motor.HW;
package body Motor is -- protected object

  protected Control is
    procedure Set_Pump (To : Pump_Status);
  private
    Motor_Status : Pump_Status := Off;
  end Control;

  procedure Set_Pump (To : Pump_Status) is
  begin
    Control.Set_Pump (To);
  end Set_Pump
```

Realización: Motor (2)

```
   protected body Control is

      procedure Set_Pump (To : Pump_Status) is
      begin
        case To is
          when On =>
            Motor.HW.Start;
            Control.Motor_Status := On;
          when Off =>
            Motor.HW.Stop;
            Control.Motor_Status := Off;
        end case;
      end Set_Pump;

   end Control;

end Motor;
```

Realización: HLW Controller (1)

```
package HLW_Controller is -- protected object
  procedure Sensor_High_IH;
  procedure Sensor_Low_IH;
end HLW_Controller;
```

```
with HLW_Handler;
...
package body HLW_Controller is -- protected object

  protected Control is
    procedure Sensor_High_IH;
    procedure Sensor_Low_IH;
    -- we will see later how to attach these procedures
    -- to interrupt sources
  end Control;
```

Realización: HLW Controller (2)

```
   protected body Control is

      procedure Sensor_High_IH is
      begin
        HLW_Handler.Start (High);
      end Sensor_High_IH;

      procedure Sensor_Low_IH is
      begin
        HLW_Handler.Start (Low);
      end Sensor_Low_IH;
   end Control;

end HLW_Controller ;
```

Realización: HLW Handler (1)

```
package HLW_Handler is -- sporadic object
  type Water_Mark is (High,Low);
  procedure Start (Level : Water_Mark);
end HLW_Handler;
```

```
package body HLW_Handler is -- sporadic object

  procedure Sporadic_Code (Level : Water_Mark) is
  begin
    case Level is
      when High => Motor.Set_Pump(On);
      when Low  => Motor.Set_Pump(Off);
    end case;
  end Sporadic_Code;

  task Thread;
```

Realización: HLW Handler (2)

```
protected Control is
  procedure Start      (Level :     Water_Mark);
  entry     Wait_Start (Level : out Water_Mark);
private
  Started : Boolean := False;
  Water   : Water_Mark;
end Control;

task body Thread is
  Level : Water_Mark;
begin
  loop
    Control.Wait_Start(Level);
    Sporadic_Code(Level);
  end loop;
end Thread;
```

Realización: HLW Handler (3)

```
   protected body Control is
     procedure Start      (Level :     Water_Mark) is
     begin
       Water := Level;
       Started := True;
     end Start;

     entry Wait_Start (Level : out Water_Mark) when Started is
     begin
       Level := Water;
       Started := False;
     end Wait_Start;

   end Control;

end HLW_Handler;
```

Realización: procedimiento principal

```
with Motor, HLW_Controller, HLW_Handler;
procedure Pump_Control_System is
begin
  null;
end Pump_Control_System;
```

- ¡El procedimiento principal no hace nada!
- La tarea de entorno elabora los paquetes
  - al elaborar HLW_Handler arranca la tarea esporádica

  y luego llama al procedimiento principal
- ¡Una tarea de Ada no termina hasta que han terminado todos sus descendientes!
  - la tarea de entorno espera que termine la tarea esporádica