Allocation of Limited Resources:

Consider a small school using special pencils during tests. Students use those pencils on special forms, which are then processed by a mark sense exam scoring machine. The school has 100 pencils, housed in principal's office.

Every time a teacher wants to conduct a test, he obtains the necessary number of pencils, which are to be returned after the test. Everything goes smoothly as long as the principal has sufficient number of pencils on hand.

However, a teacher attempting to "break the bank" must be put on hold until other teacher(s) return their pencils. Given that pencils wear out and some get stolen, some teachers do not return all pencils they were allocated.

New pencils are occasionally added to the inventory.

1st attempt at resource allocation:

```
package Resource_Allocation_1st_Attempt is

   protected type Manager (Initial_Inventory : Positive) is
      entry Take (Number_Required : in Positive);
      -- Make a request for resources.
      -- The caller is blocked
      -- until the amount requested is available

      procedure Replace (Number_Returned : in Positive);
      -- Return resources so some other task may use them.
   private
      Inventory : Natural := Initial_Inventory;
   end Manager;

end Resource_Allocation_1st_Attempt;
```

Implementation: Trivially easy, eh?

```
package body Resource_Allocation_1st_Attempt is

   protected body Manager is

      entry Take (Number_Required : in Positive)
                  when Number_Required >= Inventory is
      begin
         Inventory := Inventory - Number_Required;
      end Take;

      procedure Replace (Number_Returned : in Positive) is
      begin
         Inventory := Inventory + Number_Returned;
      end Replace;

   end Manager;

end Resource_Allocation_1st_Attempt;
```

Not really. Compiler complains

```
      entry Take (Number_Required : in Positive)
                  when Number_Required >= Inventory is

*** ERROR** “Number_Required” is undefined
```

We may not use entry call parameters in barrier evaluations. Those values are known only when the entry is accepted.

Correct solution:

```
package Resource_Allocation is

   protected type Manager (Initial_Inventory : Positive) is

      entry Take (Number_Required : in Positive);
      -- Make a request for resources.
      -- The caller is blocked
      -- until the amount requested is available

      procedure Replace (Number_Returned : in Positive);
      -- Return resources so some other task may use them.

   private
      Inventory         : Natural := Initial_Inventory;
      Tasks_Waiting     : Natural;
      Resource_Released : Boolean := False;
      entry Wait (Number_Required : in Positive);
   end Manager;

end Resource_Allocation;
```

Correct implementation is trickier:

```
package body Resource_Allocation is

   protected body Manager is

      entry Take (Number_Required : in Positive)
                  when Inventory > 0 is
      begin
         if Number_Required <= Inventory then
            Inventory := Inventory - Number_Required;
         else
            requeue Wait;
         end if;
      end Take;

      procedure Replace (Number_Returned : in Positive) is
      begin
         Inventory := Inventory + Number_Returned;
         Tasks_Waiting := Wait'Count;
         if Tasks_Waiting > 0 then
            Resource_Released := True;
         end if;
      end Replace;

      entry Wait (Number_Required : in Positive)
            when Resource_Released is
      begin
         -- Decrement number of tasks waiting
         Tasks_Waiting := Tasks_Waiting - 1;
         -- Am I the last task to be checked?
         if Tasks_Waiting = 0 then
            -- Close the barrier
            Resource_Released := False;
         end if;
         -- Are there enough resources for my request
         if Number_Required <= Inventory then
            Inventory := Inventory - Number_Required;
         else  -- Get back in line and wait for resources
            requeue Wait;
         end if;
      end Wait;
```

```
   end Manager;

end Resource_Allocation;
```

Demo:

```
with Resource_Allocation;
with Ada.Numerics.Discrete_Random;
with Ada.Numerics.Float_Random;
with Protected_Output;
procedure Resource_Allocation_Demo is

   subtype Request_Range is Integer range 1 .. 8;

   package Random_Num is new Ada.Numerics.Discrete_Random
(Request_Range);

   -- The resource manager
   Manager : Resource_Allocation.Manager (Initial_Inventory
=> 10);

   ------------------------------------------------------
   protected Random_Request is
      procedure Get (Number : out Request_Range);
      -- Returns a random request number
   private
      First_Call   : Boolean := True;
      My_Generator : Random_Num.Generator;
   end Random_Request;

   protected body Random_Request is
      procedure Get (Number : out Request_Range) is
      begin
         if First_Call then
            -- On the first call, reset the generator
            Random_Num.Reset (My_Generator);
            First_Call := False;
         end if;

         -- Get a random float value
         -- and convert it to a duration
         Number := Random_Num.Random (My_Generator);
      end Get;
   end Random_Request;

   ------------------------------------------------------
```

```
   protected Random_Duration is
      procedure Get (Item : out Duration);
      -- Returns a random duration value between 0.0 and 2.0
seconds
   private
      First_Call   : Boolean := True;
      My_Generator : Ada.Numerics.Float_Random.Generator;
   end Random_Duration;

   protected body Random_Duration is
      procedure Get (Item : out Duration) is
      begin
         if First_Call then
            -- On the first call, reset the generator
            Ada.Numerics.Float_Random.Reset (My_Generator);
            First_Call := False;
         end if;
         -- Get a random float value
         -- and convert it to a duration
         Item := 2 * Duration
(Ada.Numerics.Float_Random.Random (My_Generator));
      end Get;
   end Random_Duration;

   ------------------------------------------------------
   task type Requester (ID : Positive);
   task body Requester is
      My_Request : Request_Range;
      My_Delay   : Duration;
   begin
      loop
         -- Request a random number of resources
         Random_Request.Get (My_Request);
         -- Get the resources
         Manager.Take (My_Request);
         Protected_Output.Put_Line
         ("Task" & Positive'Image (ID) & " got "
         & Integer'Image (My_Request));
         Random_Duration.Get (My_Delay);
         delay My_Delay;
         -- Replace the resources
         Manager.Replace (My_Request);
```

```
         Random_Duration.Get (My_Delay);
         delay My_Delay;
      end loop;
   end Requester;

   -- A number of requester tasks
   R1 : Requester (1);
   R2 : Requester (2);
   R3 : Requester (3);
   R4 : Requester (4);


begin
   null;
end Resource_Allocation_Demo;
```