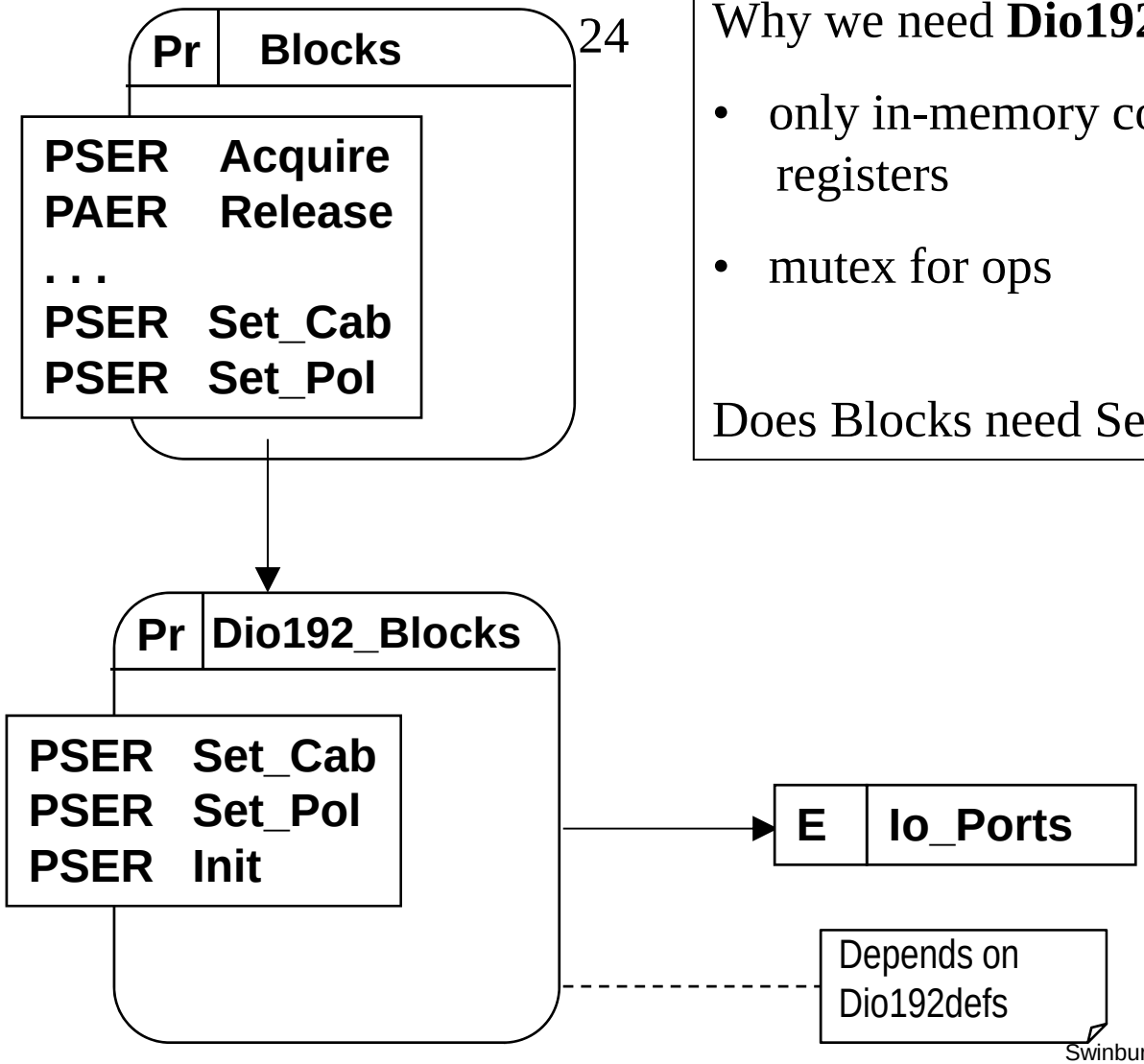
SWE30001 Real-Time Programming

Lecture 8B: Initialisation & Timeout

Ref Hugues, McCormick & Singhoff sections 4.4, 5.3.1

SWINBURNE UNIVERSITY OF TECHNOLOGY

One Design for Blocks

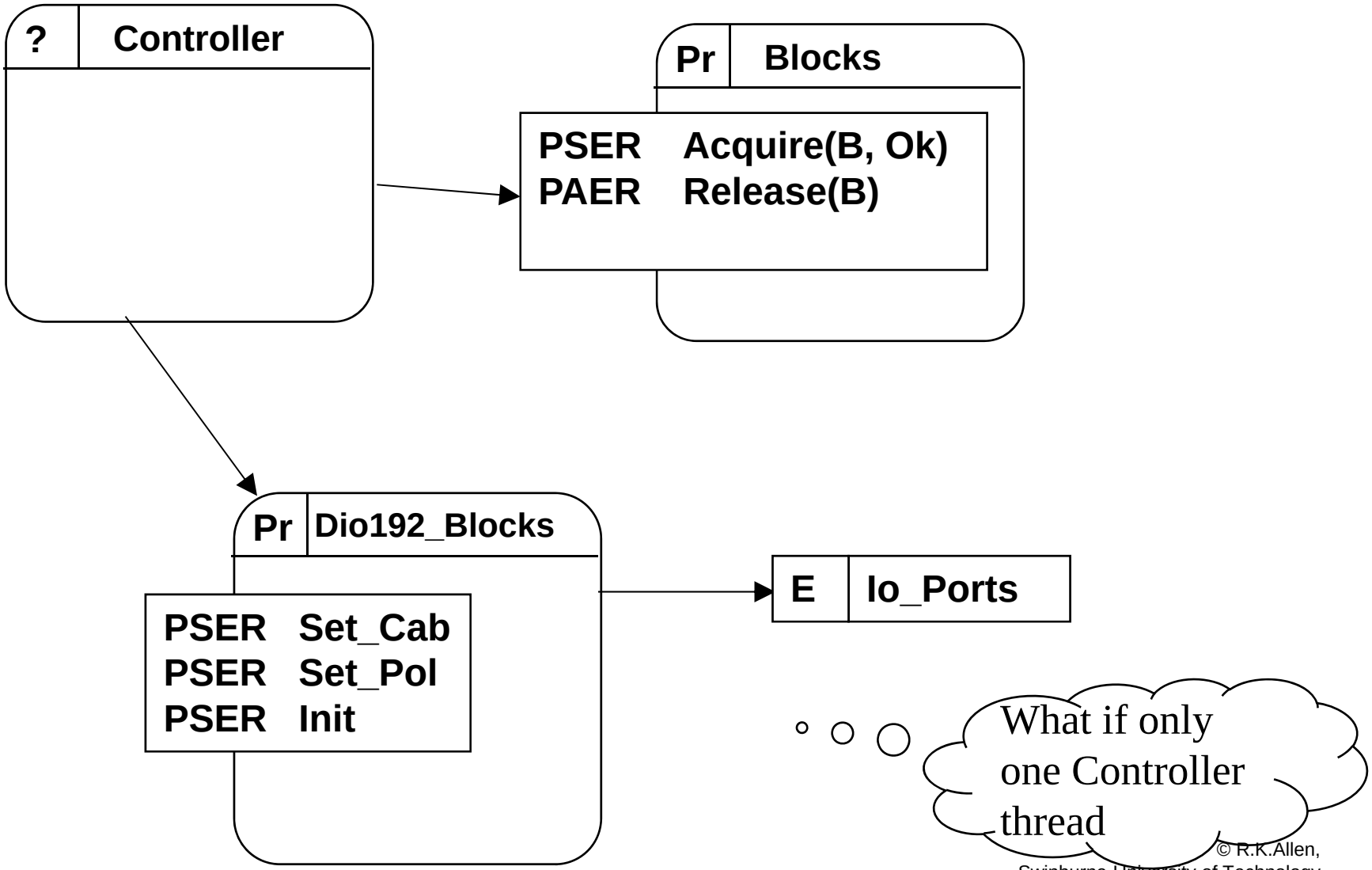


Why we need **Dio192_Blocks**?

- only in-memory copy of registers
- mutex for ops

Does Blocks need Set_Cab?

Another Design for Blocks



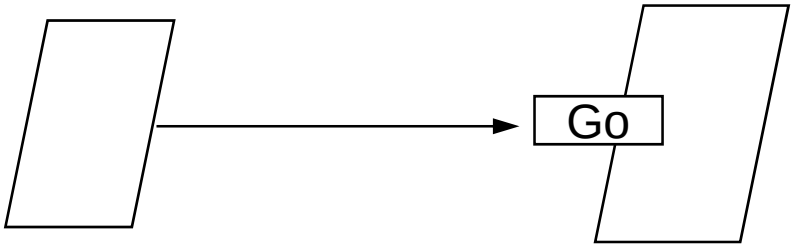
Synchronizing Startup

```
task body My_Cyclic is
   T : Time := Clock;
   Interval : Duration := 0.1;  -- or …
begin
  loop
    Do_Something;              -- what's the problem?
    T := T + Interval;
    delay until T;
  end loop;
end My_Cyclic;
---------- we need:  -----------------
task body My_Cyclic is
…
begin
      block on something
      loop . . .
```

Synchronizing Startup - Rendezvous

Don’t use!

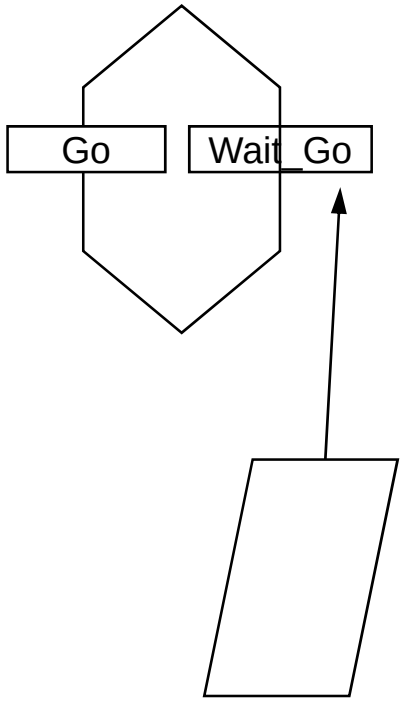
```
task Starter;
task My_Cyclic is
   entry Go;
end My_Cyclic;

task body My_Cyclic is
…
begin
      accept Go; -- rendezvous
      loop . . .
```



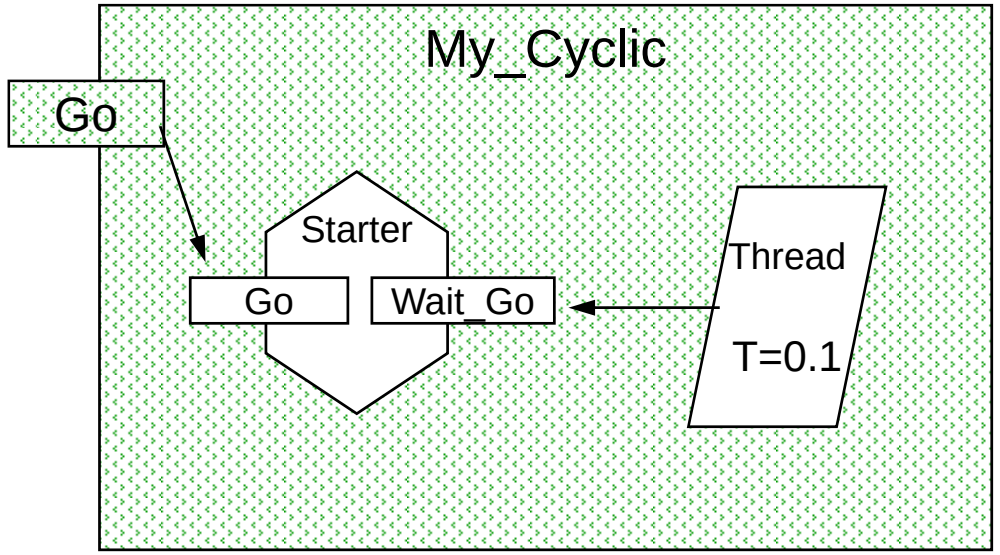
Synchronizing Startup – Protected Entry

- Effectively HRT-HOOD discourages rendezvous
  - The Ravenscar Profile (later) forbids them
- Should call a protected entry:

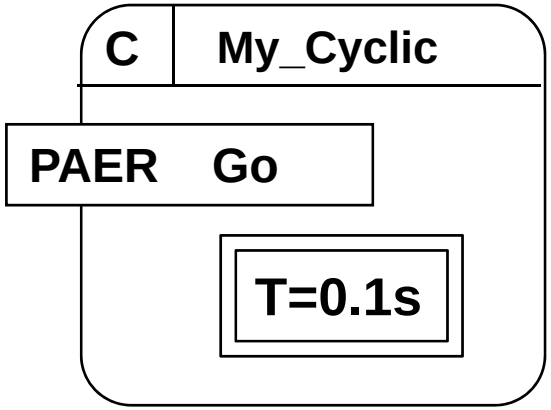
```
package body My_Cyclic is
  protected Starter is
    entry Wait_Go;
    procedure Go;
  private
    Released : Boolean := False;
  end Starter ;   -- code like binary semaphore
    …
  task body Thread is
    …
    begin
      Starter.Wait_Go;
      loop . . .
```



Synchronizing Protected Entry: Diagram



Structurally, it looks like a Sporadic!
It isn't.



Don't get confused – it isn't an ordinary op.
You prob don't need it.

Identity and Initial Data

- Also:
- To pass initial data you can use type discriminants
  - See Lec 6 slides “Multiple Sporadics in a Package”
- or the task can get it from a data-manager protected object.
- This combines with startup control

Initialization by Protected Entry -1

```
type Ant_Id is range 1..1000;
task type Ant_Type;
protected Ant_Starter is
        entry Wait_Init (Id : out Ant_Id; Home : out Nest_Ref);
        procedure Go_Ants( …
```

```
task body Ant_Type is
        Me : Ant_Id;  My_Home : Nest_Ref;   . . .
begin
        Ant_Starter.Wait_Init (Me, My_Home);
        loop
                . . .
```

…
Ants : array (Ant_Id) of Ant_Type;
-- each one waits

Initialization by Protected Entry -2

```
…
-- Ant_Starter continued:

private

  Next_Id :  Ant_Id := Ants'first;              -- or := 1;
  The_Nest : Nest_Ref;                          -- initially null

end Ant_Starter;


… body:

  entry Wait_Init (Id : out Ant_Id; Home : out Nest_Ref)
                          when The_Nest /= null is

  begin

        Id := Next_Id;
        Next_Id := Ant_Id'succ(Next_Id);  -- or := Next_Id + 1;
        Home := The_Nest;

end Wait_Init;
 …
```

Notes

- This is also an example of out parameters and avoidance of “magic numbers”
  - Note the use of Ants’first
  - don’t hard-code whether type Ant_Id starts from 0 or 1 or even if the array indexes from the first value of the type.
  - Note the use of Ant_Id’succ()
  - we don’t even assume that Ant_Id is a number, just take the next value.
    - Recall Character’pos and Character’val.
- Exercise: write procedure Go_Ants.

Initialization by Discriminant with Dynamic Creation

Not exam

```
type Ant_Id is range 1..1000;
task type Ant_Type(Me : Ant_Id;  My_Home : Nest_Ref);
type Ant_Ref is access Ant_Type;
```

Double discriminant

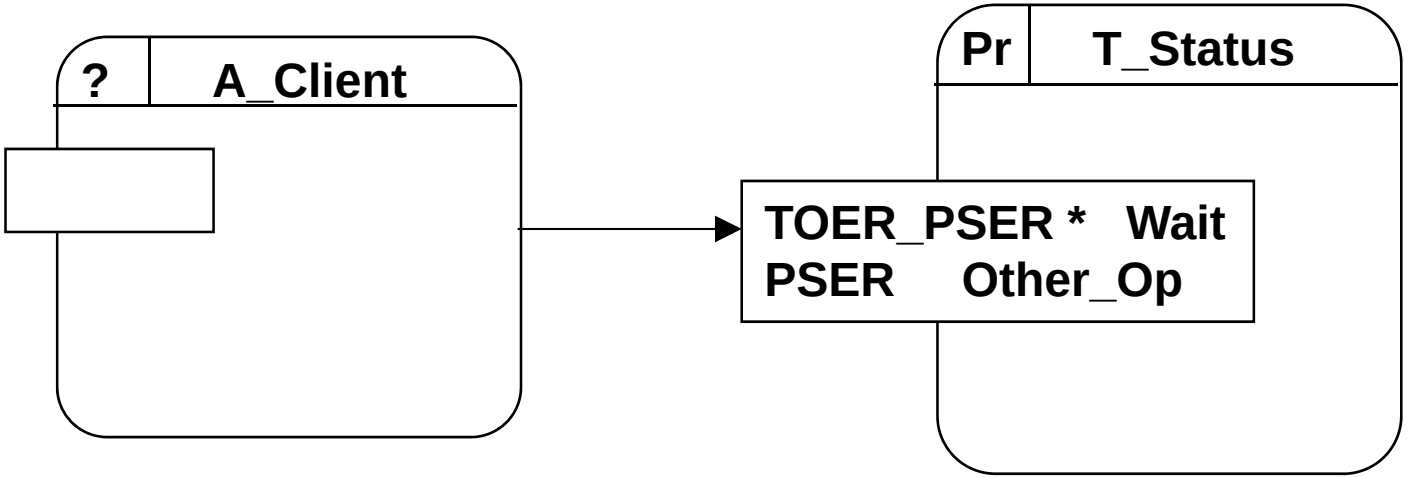
```
Ants : array (Ant_Id) of Ant_Ref;
The_Nest : …
…
for Id in Ant_Id loop
        Ants(Id) := new Ant_Type(Id, The_Nest);
end loop;
                . . .
```

When this loop runs, Ant tasks come into existence with Me and My_Home set

Task with Me=j is at position j in the array.

(Not useful for railroad)

Waiting Timeout - 1



Client:

```
T_Status.Wait(1.0, Success=>Completed);
if Completed then ...
```

NB: not recommended for Acquire/Release!

- No delays in Train control code (previous lecture)
- Maybe(?) relevant to turnout timing

Timed Entry Call

```
select
    server.entry1 [ (params) ];
    [ statements ]
or
     delay some_interval;
    alternative statements
end select;
```

cf Conditional Entry Call

```
select
        server.entry1 [ (params) ];
        [ statements ]
else
    alternative statements
end select;
```

Server must be a protected (or a task)

Can also have *delay until*

Waiting Timeout - 2

- Implementation

```
package T_Status is
   procedure Wait(
        T : in Duration;
        Success : out Boolean);
   procedure Release;
end T_Status;
```

timed entry call

```
package body T_Status is
    protected Obcs is
          entry Wait;
          ….
    end Obcs;

    procedure Wait(T : in Duration;
          Success : out Boolean ) is
    begin
          select
             Obcs. Wait;
             Success := True;
          or
              delay T;
              Success := False;
          end select;
    end Wait;
    …
```