Lecture 5:

HRT-HOOD Implementation & Example

Coding HRT-HOOD objects

Example

(Ref: Burns & Wellings 3rd ed pp 658-682)

SWINBURNE UNIVERSITY OF TECHNOLOGY

Outline

- HRT-HOOD implementation in Ada95
  - General guidelines
  - Cyclic
  - Protected
    - Note on Readers-Writers
  - Sporadic
    - Single item buffer
    - Open issues
- HRT-HOOD case study
  - Problem outline
  - First, second and third level decomposition
  - Example code

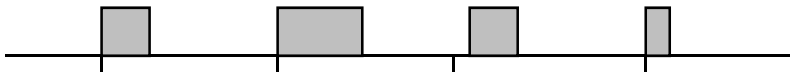
HRT-HOOD Implementation

Guidelines:

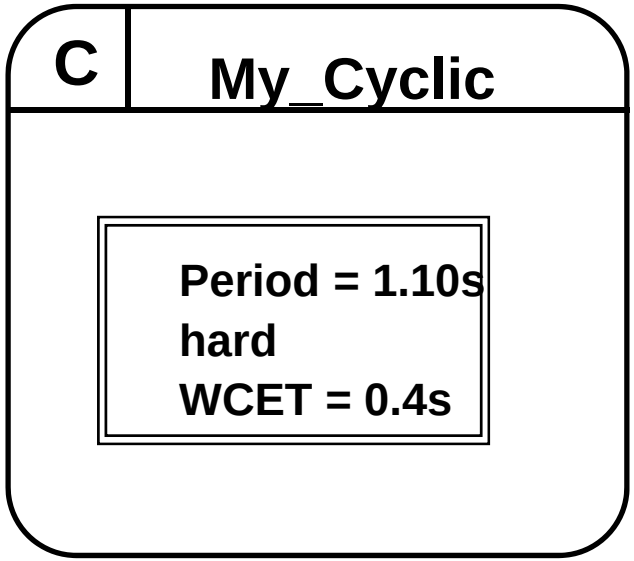
- Each HRT-HOOD object becomes a package
  - ASM style
- Package specs are minimal
  - Maximise data hiding: details hidden in spec
  - (no private section needed)
- Some packages may be needed for global definitions (spec only)
- Some HRT-HOOD objects may disappear:
  - Composites (A) that simply call-through
- Preserve design in comments

Cyclic Object - 1

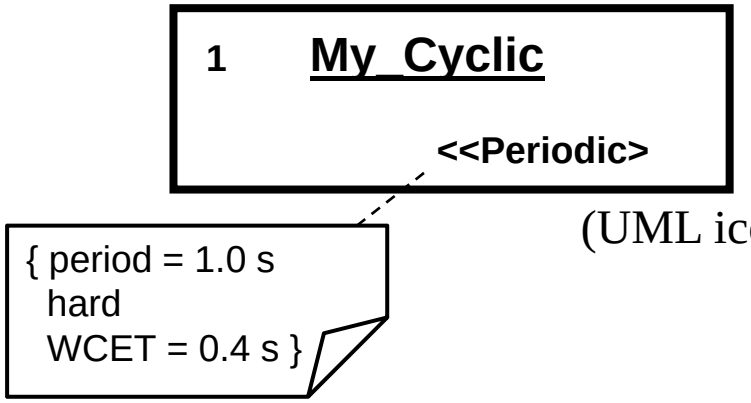
- here simplest form:
  - no operations
  - cannot act as a server or collaborator
- WCET
  - worst case execution time (**cpu** time) of each cycle



- other real-time attributes:
  - priority
  - deadline (= period by default)



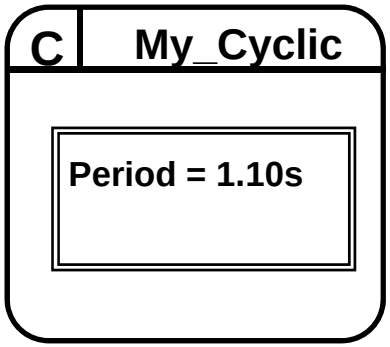
(HRT-HOOD icon)



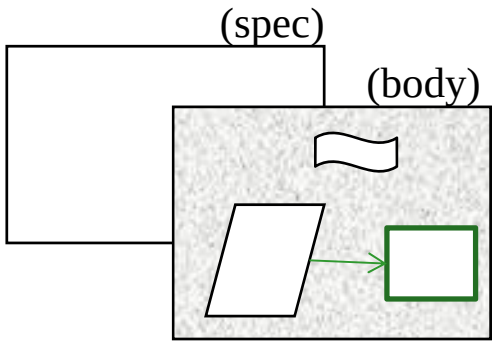
(UML icon)

Cyclic Implementation

(HRT-HOOD icon)



(Booch icons)

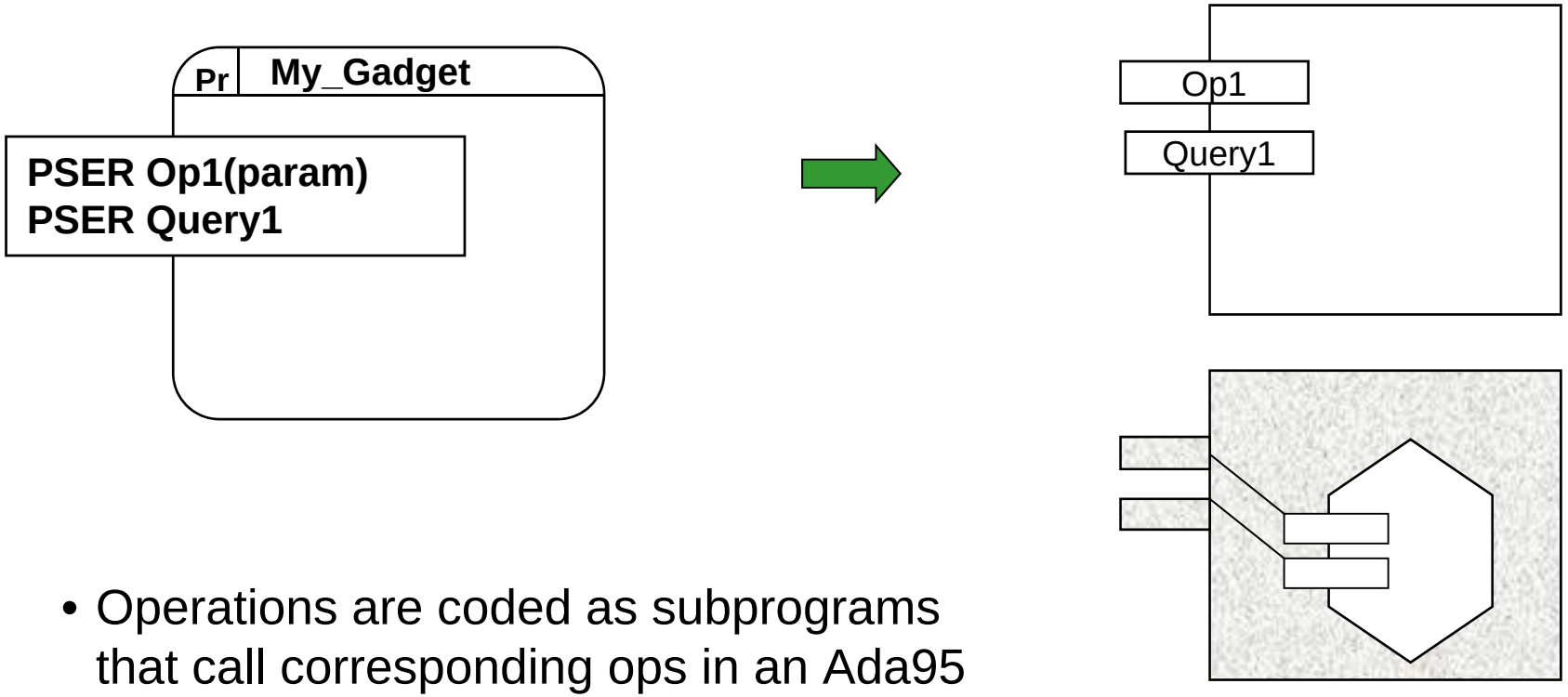



```
package My_Cyclic is  -- CYCLIC
  Period : constant := 1.10;
end My_Cyclic;
```

- No operations defined

```
with Things;
package body My_Cyclic is  -- CYCLIC
   … -- whatever needed here
   task Thread;
   task body Thread is
            …
   begin
     loop Cyclic_Op;  delay Period;
      end loop;
   end Thread;
end My_Cyclic;
```

Protected Implementation -1



- Operations are coded as subprograms that call corresponding ops in an Ada95 protected
- The Ada95 protected is completely hidden in the package body
- Data is inside that

Protected Implementation - 2

```
package My_Gadget is
    -- PROTECTED

procedure Op1( -- PSER
    Param1 : in A_Type);

function Query1 -- PSER
   return B_Type;

end My_Gadget;
```

```
-- similar to
procedure Query1(
   B : out B_Type);
```

```
with Various;
package body My_Gadget is – PROTE..
  protected Object is
    procedure Op1(
          Param1 : in A_Type);
    function Query1 return B_Type;
  private
    Data1 : A_Type := initial_value;
    Data2 : …
  end Object;

  procedure Op1(
          Param1 : in A_Type) is
  begin
    Object.Op1(Param1);
  end Op1;
```

Protected Implementation - 3

```
function Query1 return B_Type is
begin
      return Object.Query1;
end Query1;

protected body Object is
   procedure Op1(
         Param1 : in A_Type) is
   begin
         Data1 := …. ;
         …
   end Op1;

   function Query1 return B_Type is
   begin
      …
      return Data2;
   end Query1;
end Object;

end My_Gadget;
```

procedures allowed to modify data

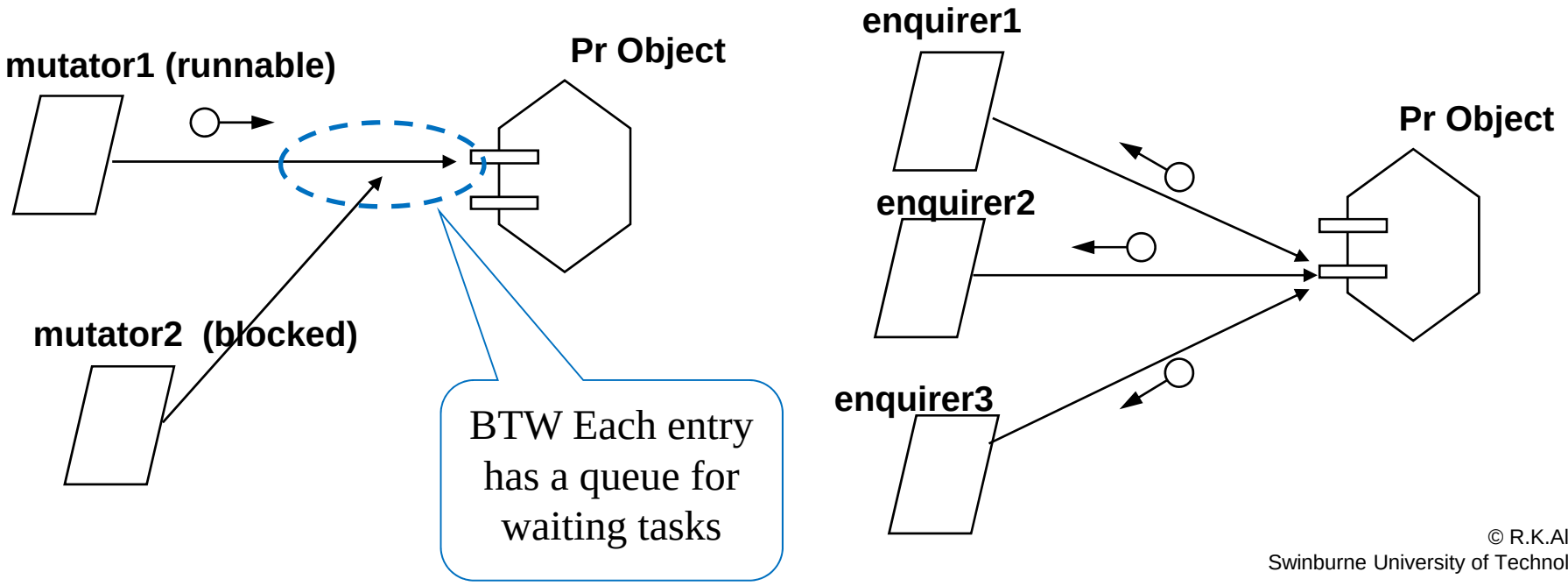
the real work gets done here!

functions are guaranteed to NOT modify data

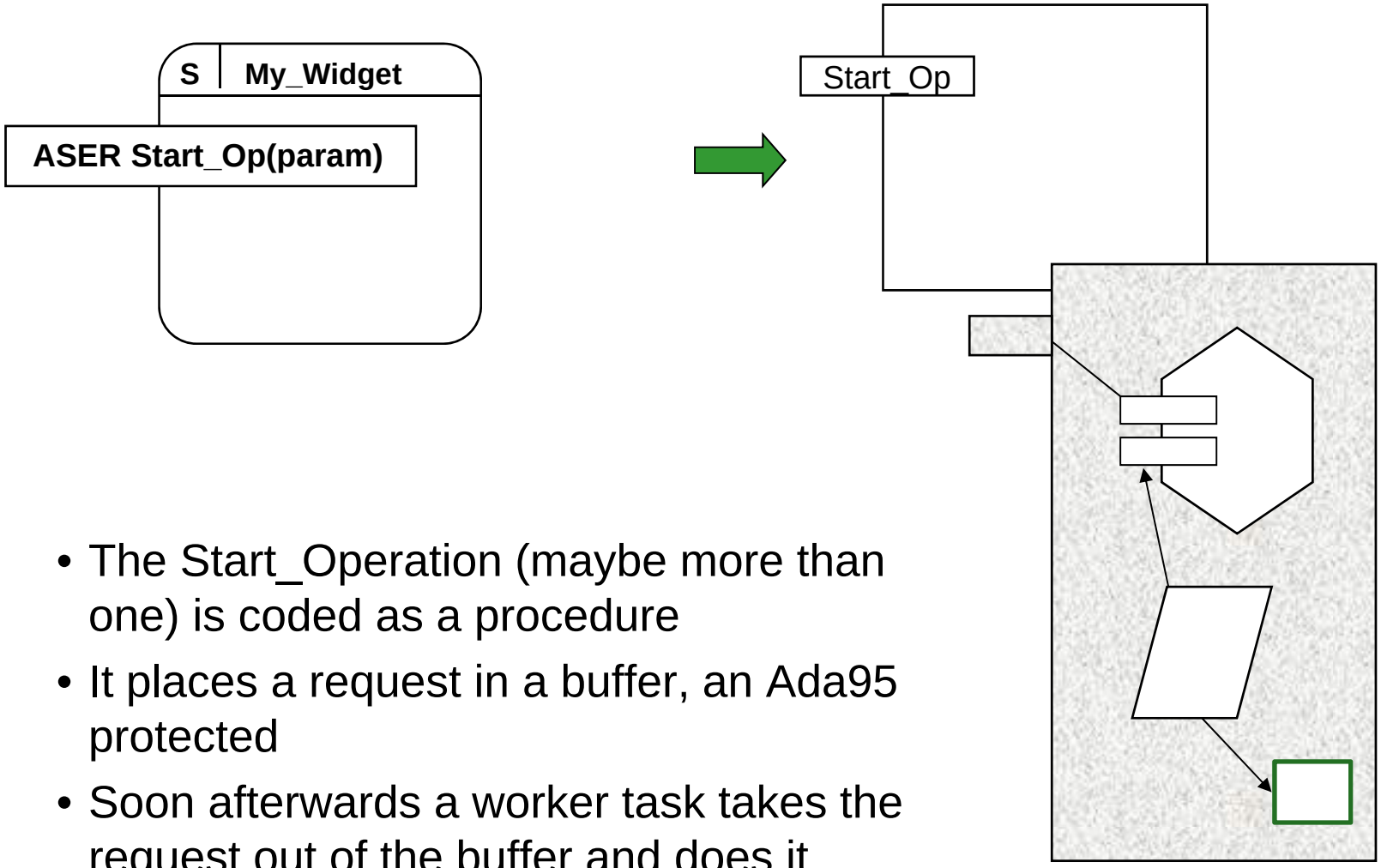
Both guaranteed not to block, eg delay

Note on Readers-Writers Protocol

- What threads can be “inside” a protected?
  at any instant
  - none
  - one task that might modify the state -- procedure
  - many tasks (n) that read the state -- function

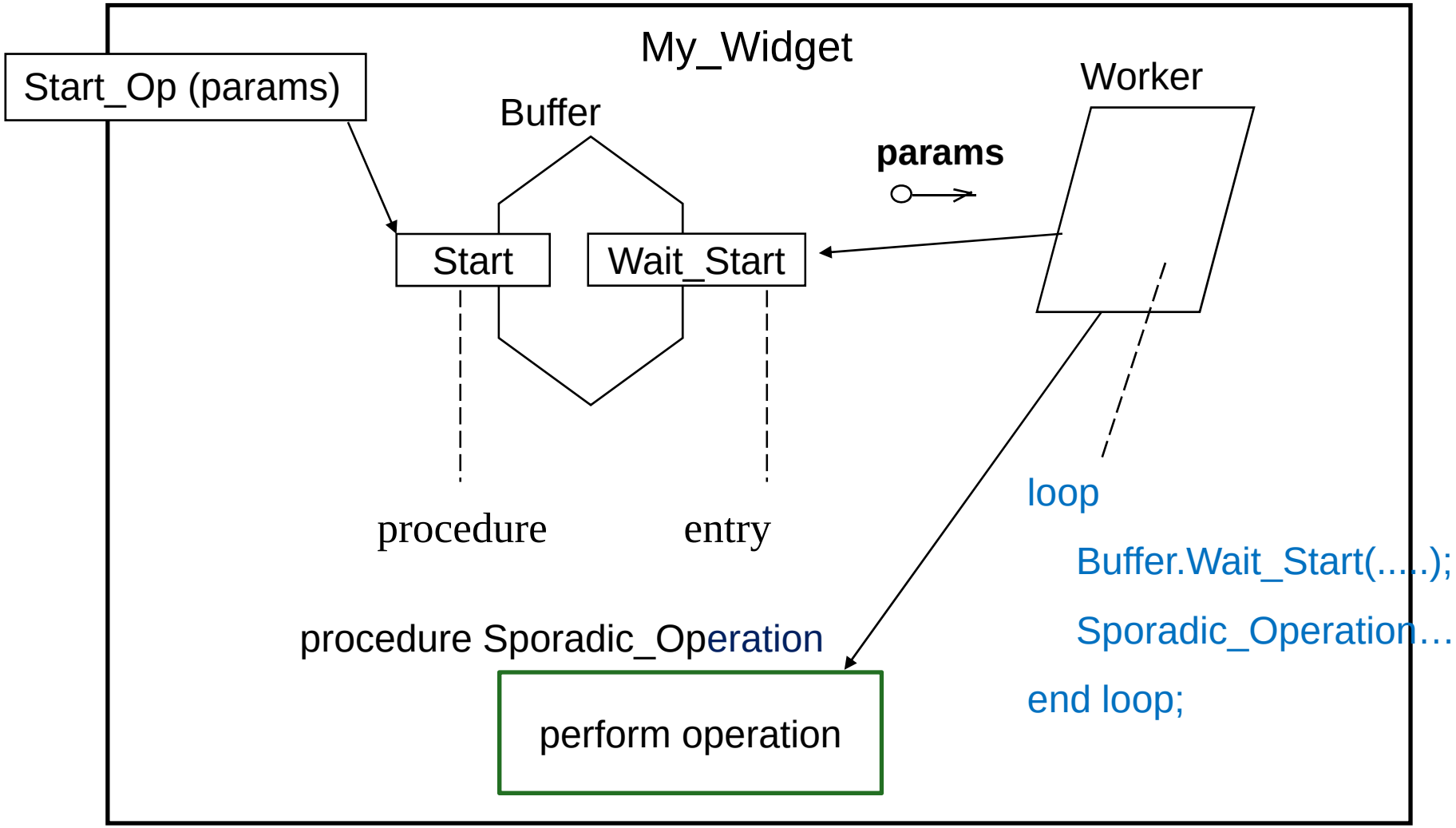


Sporadic Implementation - 1



- The Start_Operation (maybe more than one) is coded as a procedure
- It places a request in a buffer, an Ada95 protected
- Soon afterwards a worker task takes the request out of the buffer and does it

Sporadic Implementation - 2



Sporadic Implementation - 3

w/o overrun detection

```
-- Single item buffer
protected Buffer is
    procedure Start(
      Request: in Request_Type);

    entry Wait_Start(
      Request: out Request_Type
     );
private
    Item : Request_Type;
    Item_Available
                        : Boolean := False;
end Buffer;
---
protected body Buffer is
    procedure Start(
      Request: in Request_Type) is
    begin
      Item_Available := True;

      Item := Request;
end Start;

entry Wait_Start(
    Request: out Request_Type
     )
    when Item_Available is
begin
     Request := Item;

     Item_Available := False;

end …
```

Sporadic Implementation - 4

```
-- Single item buffer
protected Buffer is
    procedure Start(
      Request: in Request_Type);

    entry Wait_Start(
      Request: out Request_Type;
      Over_Run : out Boolean );
private
    Item : Request_Type;
    Item_Available,
    Too_Fast        : Boolean := False;
end Buffer;
---
protected body Buffer is
    procedure Start(
      Request: in Request_Type) is
    begin
      if Item_Available then
        Too_Fast := True;
      else
        Item_Available := True;
      end if;
      Item := Request;  -- ignore old
end Start;

entry Wait_Start(
    Request: out Request_Type;
    Over_Run : out Boolean )
    when Item_Available is
begin
     Request := Item;
     Over_Run := Too_Fast;
     Item_Available := False;
     Too_Fast := False;
end …
```

Sporadic Implementation - 4

```
procedure Sporadic_Operation(
      Request: in Request_Type) is
begin
     …
end Sporadic_Operation;

task Worker;

task body Worker is
   Req : Request_Type;
    Oops : Boolean := false;
begin
    loop
       Buffer.Wait_Start(Req, Oops);
       if Oops then ...

       Sporadic_Operation(Req);
    end loop;
end Worker;

end Widget; -- package
```

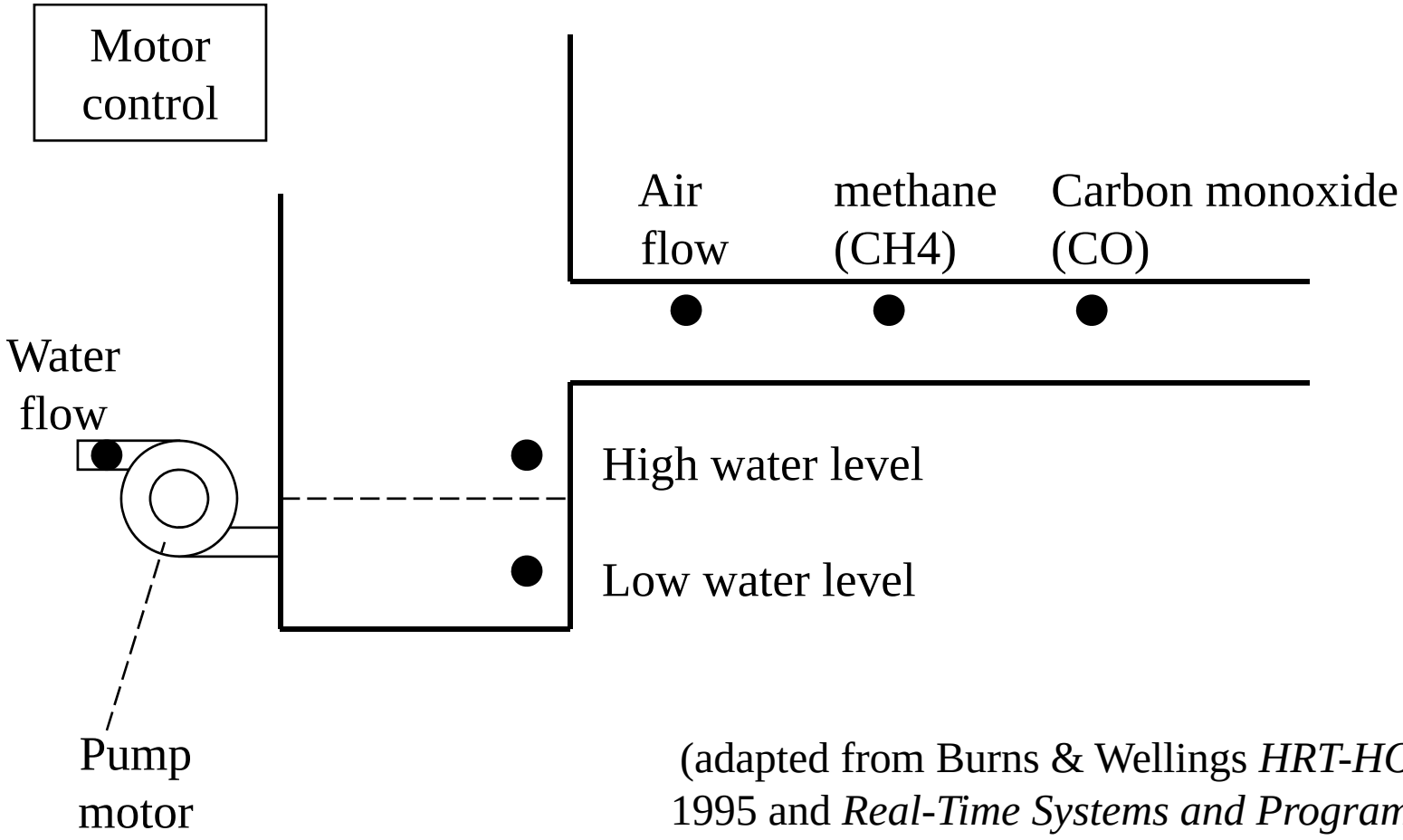
It's good style to separate op code from loop code.

Display error message etc.

Sporadic Implementation - 5

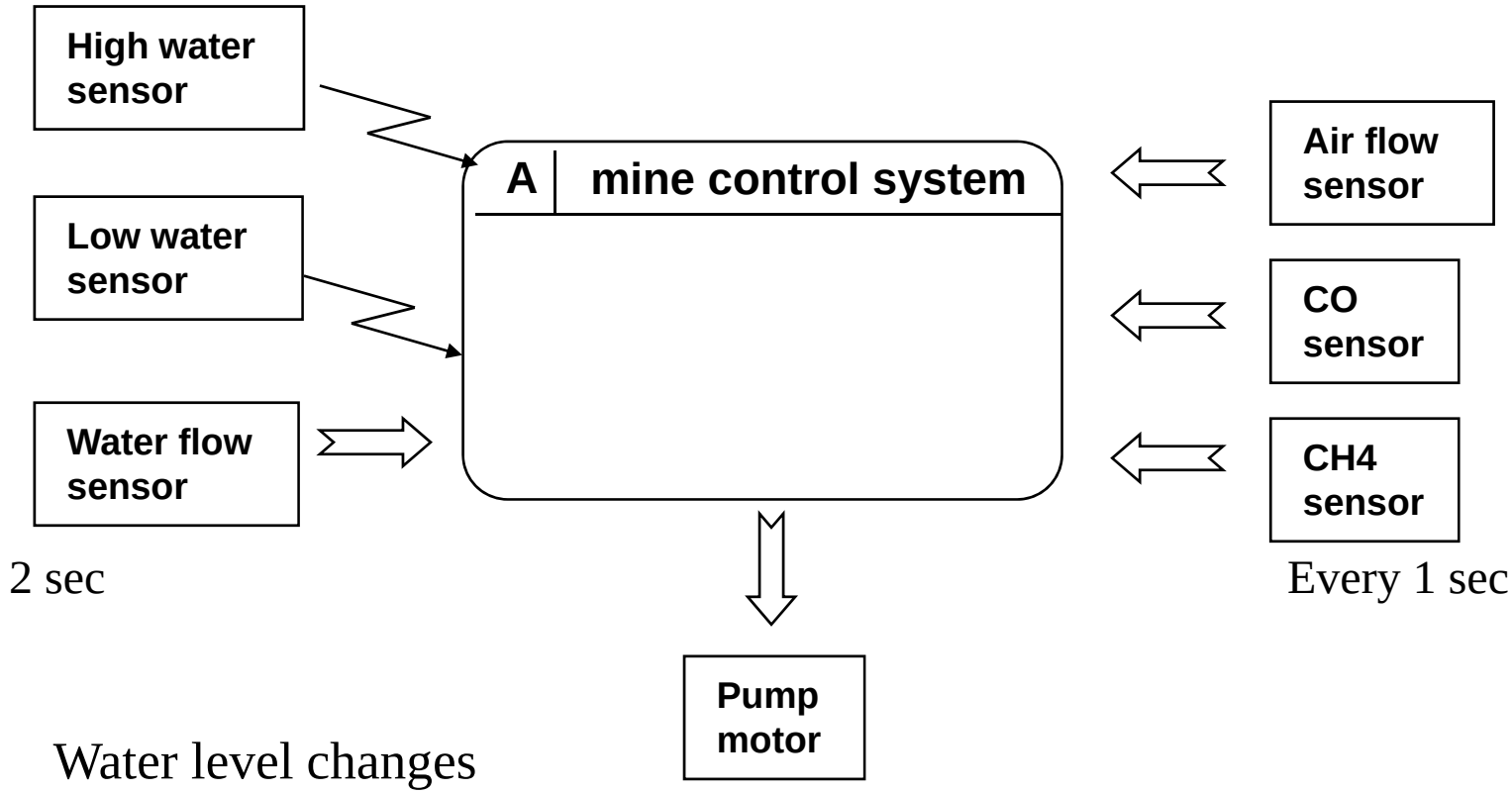
- Notes:
  - The above code does not assume response is complete before the next request arrives – it has single-buffering.
  - It does detect “buffer full” but doesn’t block the client
  - The worker is responsible for error message/cleanup on over-run
- Issues
  - If there is an over-run ignore old data or new data?
  - Longer buffer? How do we implement it?
  - Multiple operations (Op1, Op2, etc)?
  - Where do we put instance variables? None?
  - Multiple identical objects?
- Answers and more details
  - Next week

HRT-HOOD Example: The Mine



(adapted from Burns & Wellings *HRT-HOOD*, 1995 and *Real-Time Systems and Programming Languages*, 3rd ed, 2001)

Data Flows and Rates



Water level changes approx 100 sec apart

Other times 5 sec

Unsafe to run motor with CH_4 present!

Timing Characteristics

	arrival time	deadline	kind
• CH4 Sensor	1 sec	0.3 sec	CYCLIC
• CO Sensor	5 sec	3 sec	"
• Water_Flow	2 sec	1 sec	"
• Air_Flow	5 sec	5 sec	"

Poll their devices

• Water level detectors	>= 100	20 sec	INTERRUPT

Identifying Objects

NEW

- Beyond “underline the noun”, look for
  - Physical devices
  - Real world items
  - Causal objects
  - Control elements
  - Service providers
  - Messages and info flow
  - Key concepts
  - Transactions
  - Persistent data
  - Visual elements
- and apply scenarios

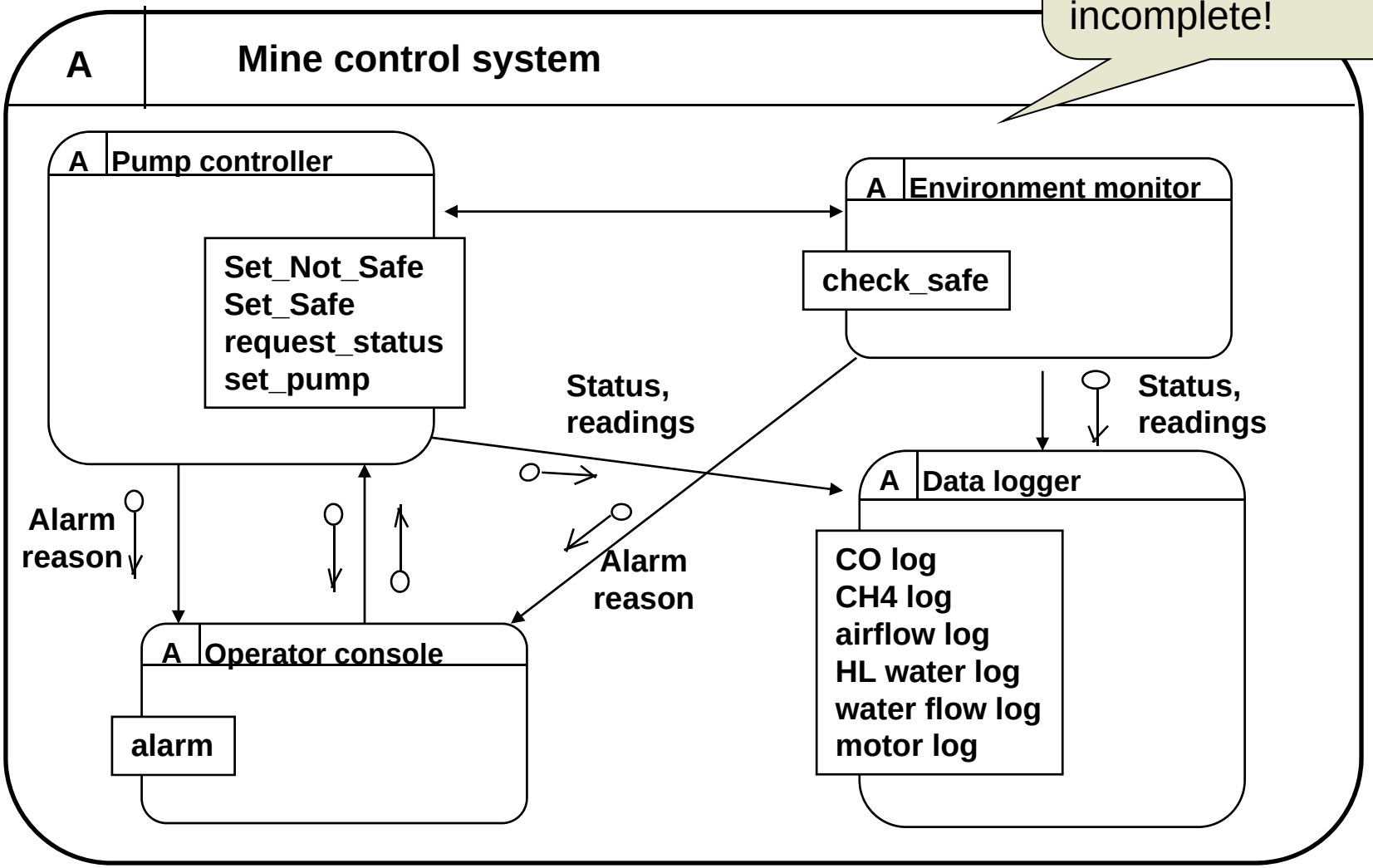
(adapted from Douglass, B.P. *Real Time UML* 3rd ed, Addison Wesley 2004)

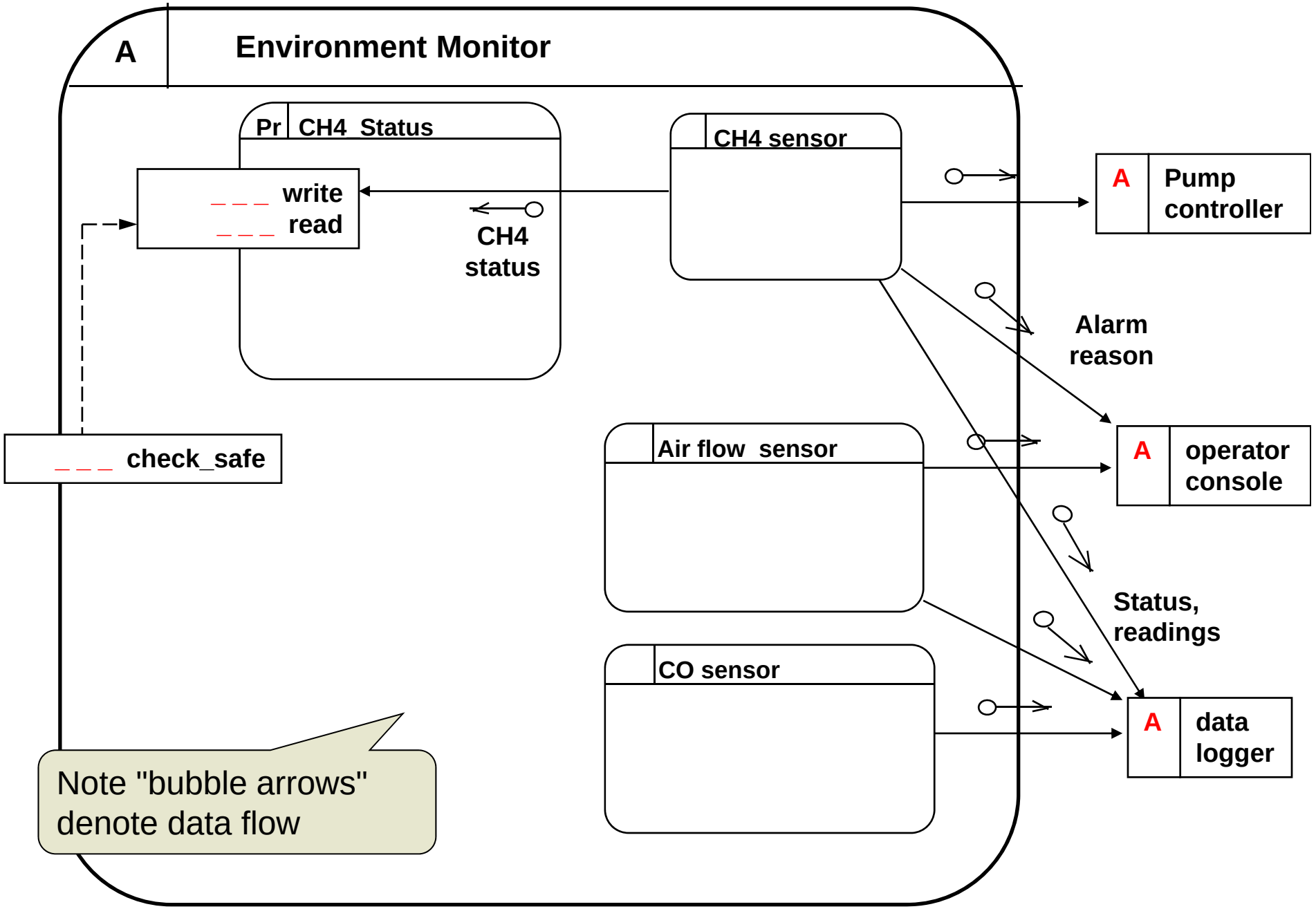
Hierarchical Design Process

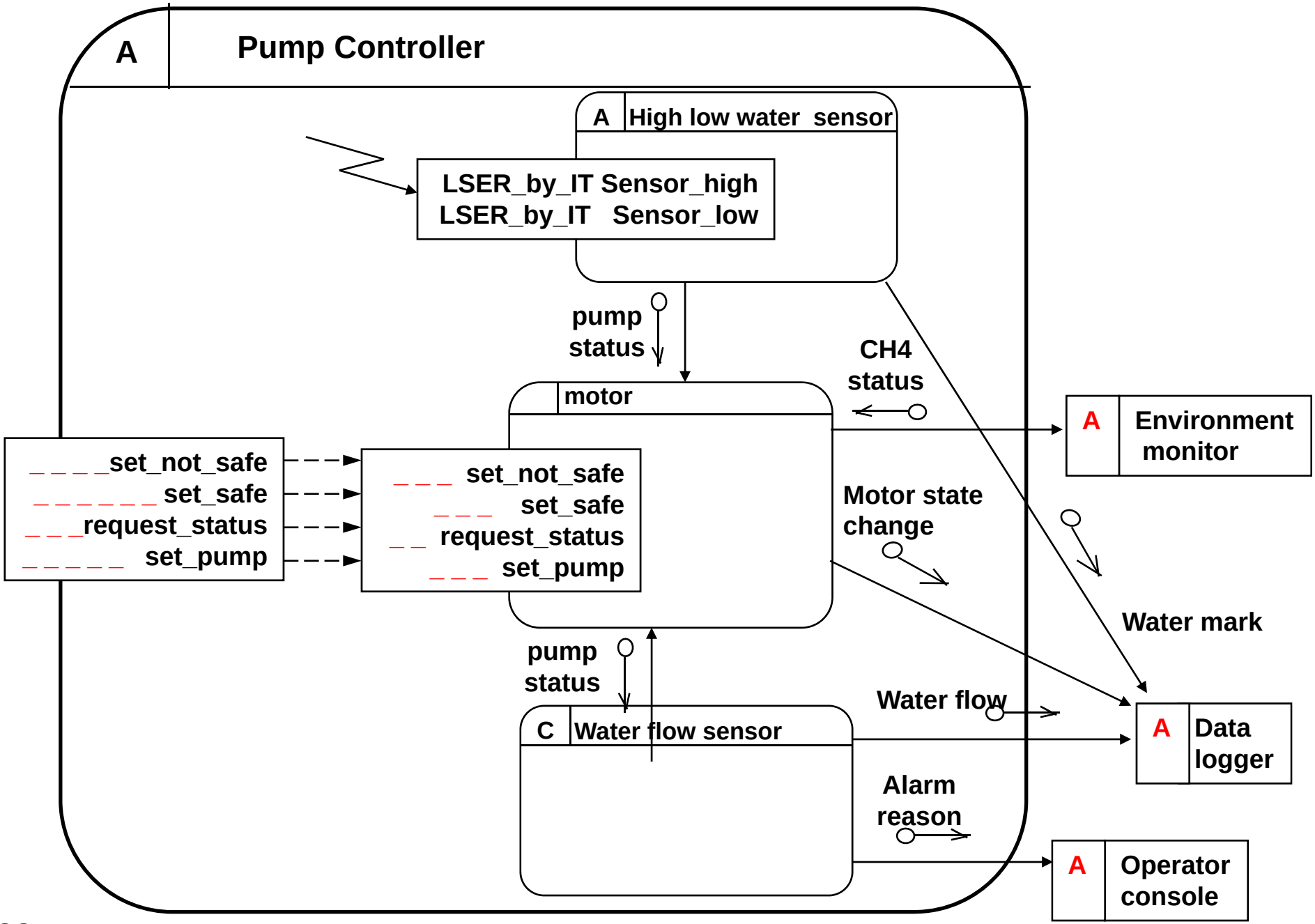
- Starting from the top level object we divide it ("decompose it") into subsystems, objects.
- Each 2nd level object is decomposed…
- etc
- We stop when we have no more composite objects, only "terminal" objects.
- In a fully HRT-HOOD design (here) all terminal objects are non-Active.
  - Cyclic, Passive, Protected, Sporadic
    - (Environment objects are considered to be outside the system being designed.)
- In this unit, we designate/mark composite objects as Active.
- Sometimes it may be necessary to have terminal objects that don't fit the HRT-HOOD types.
  - terminal Active
  - Possibly can't be analyzed, so maybe only safe to use these for "background", low priority tasks.

First Level Decomposition

A lot of work to identify these operations, but still incomplete!

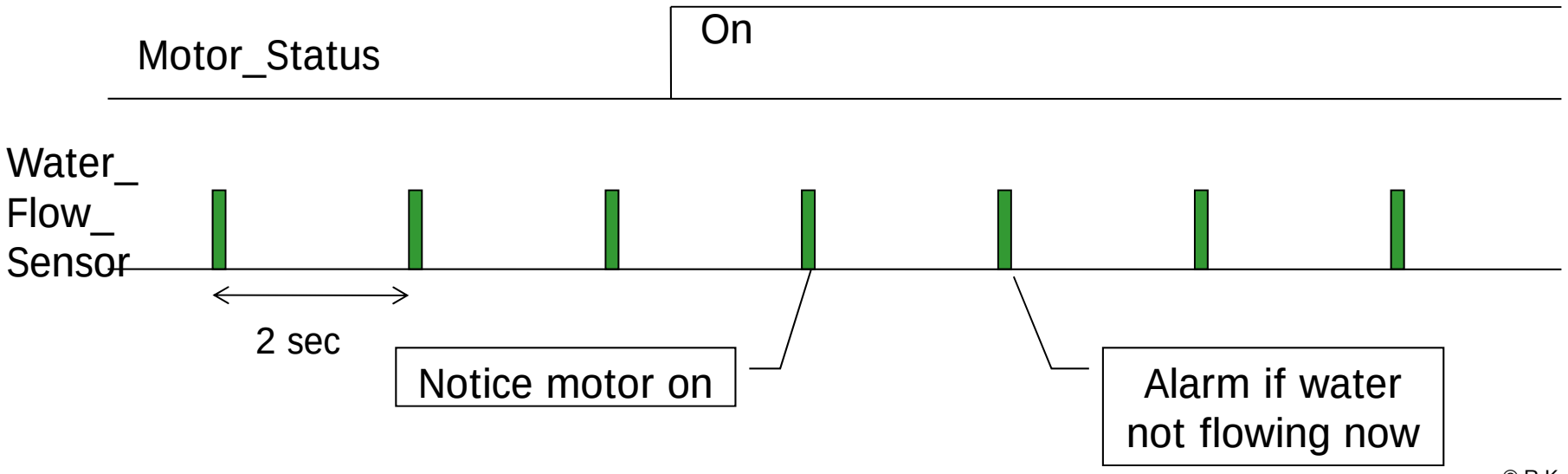


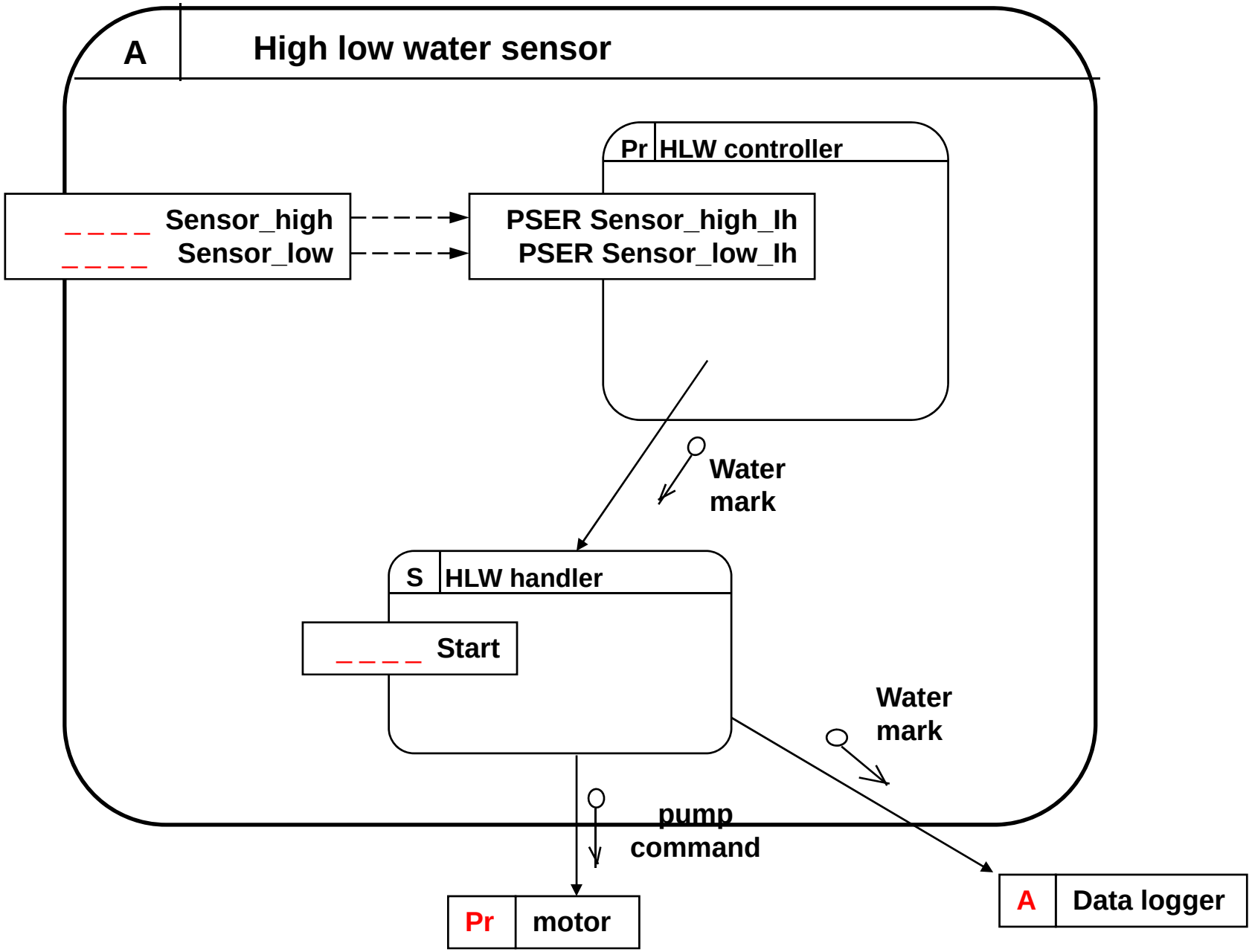

A
Environment Monitor
Pr
CH4 Status
write
read
CH4
status
CH4 sensor
A
Pump
controller
Alarm
reason
check_safe
Air flow sensor
A
operator
console
Status,
readings
CO sensor
A
data
logger
Note "bubble arrows"
denote data flow


A
Pump Controller
A
High low water sensor
LSER_by_IT Sensor_high
LSER_by_IT Sensor_low
pump
status
CH4
status
motor
A
Environment
monitor
set_not_safe
set_safe
request_status
set_pump
set_not_safe
set_safe
request_status
set_pump
Motor state
change
Water mark
pump
status
Water flow
C
Water flow sensor
A
Data
logger
Alarm
reason
A
Operator
console

Notes: Water_Flow_Sensor

- This cyclic object checks every 2 seconds
  - whether water is flowing in the pipe, and
  - whether it should be flowing: by calling Motor.Request_Status
- It sends an alarm message to Operator_Console if the water flow doesn't start/stop within 2-4 seconds
  Example:




A
High low water sensor
Pr
HLW controller
Sensor_high
Sensor_low
PSER Sensor_high_lh
PSER Sensor_low_lh
Water
mark
S
HLW handler
Start
Water
mark
pump
command
Pr
motor
A
Data logger

Translation to Ada

- First/Second Level:
  - procedure Mine_Control_System
  - package Operator_Console
  - object Environment Monitor
    - packages CH4_Status, CH4_Sensor, …
  - object Pump Controller
    - packages Motor, Water_Flow_Sensor, HLW_Controller
  - package Data_Logger

Style 1:
Drop the outer packages

In this style we maintain the design hierarchy only in comments.

We also document the HRT-HOOD object types in comments, eg

```
-- PROTECTED part of Pump_Controller
package Motor is
```

Translation to Ada : Child Packages

- **Or** Style 2: use child packages

Recommended

  - object Environment Monitor becomes
    - packages Env, Env.CH4_Status, Env.CH4_Sensor, …
  - object Pump Controller becomes
    - packages Pump, Pump.Motor, Pump.Water_Flow_Sensor, Pump.HLW_Controller

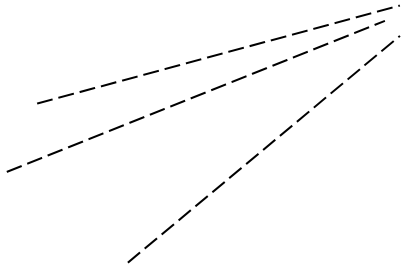
The parent may be empty and clients refer directly to the "provided" (ie public) child operations. File details:

```
pump.ads          package Pump is
                  end Pump;
pump-motor.ads    package Pump.Motor is   ← note hyphen vs dot
                        ….
                  end Pump.Motor;
```

Pump Motor Spec

- package Pump.Motor is -- PROTECTED

  type Pump_Status is (On, Off);
  -- … (other type declarations)



These declarations are only here because they are needed by clients

  Pump_Not_Safe : exception; -- raised by Set_Pump

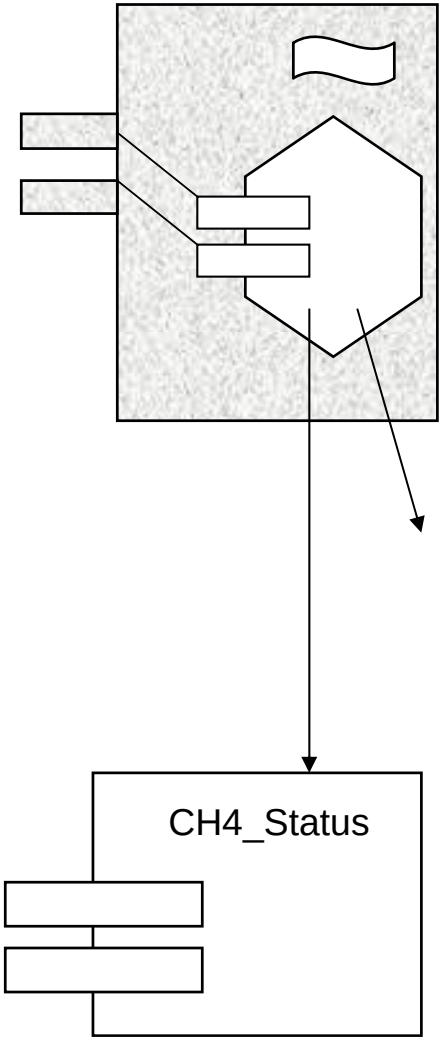
  procedure Set_Not_Safe; -- set
  procedure Set_Safe; -- set
  function Request_Status return Pump_Status;
  procedure Set_Pump(To : in Pump_Status); -- on or off

end Pump.Motor;

See mine_case_study.zip

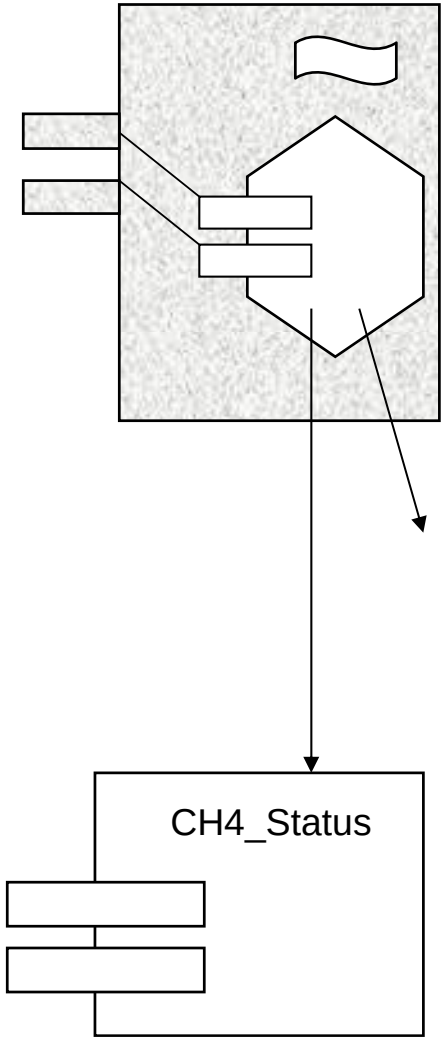
Pump Motor Body - 1

```
with Data_Logger, Env.CH4_Status, Device_Defs; …
package body Pump.Motor is

  Device_Reg : …

  protected Object is
            procedure Not_Safe;  -- set
            procedure Is_Safe …
  private
            Motor_Status : …
  end Object;

  procedure Set_Not_Safe is
  begin
            Object.Not_Safe;
  end Set_Not_Safe;
  …
```



Pump Motor Body - 2

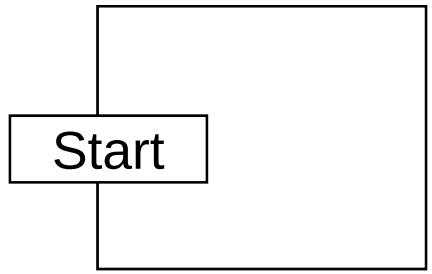
```
…

protected body Object is

        procedure Not_Safe is
        begin
            if …. -- lots of code
        end Not_Safe;

        …  -- other ops

    end Object;

end Pump.Motor;
```



HLW_Handler -- 1

- package Pump.Hlw_Handler is -- SPORADIC
  type Water_Mark is (High, Low);

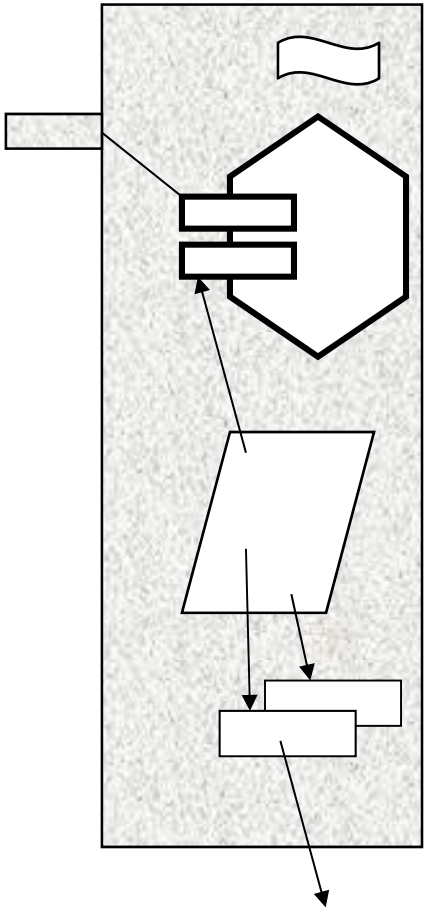
  procedure Start(Intr : Water_Mark);
  -- 2nd stage of interrupt ("intr") handler
  end Hlw_Handler;



- with Data_Logger, Pump.Motor, Device_Defs; …
  package body Pump.Hlw_Handler is
  Hw_Control_Reg : …
  …
  procedure Sporadic_Code(Intr : Water_Mark) is
  begin
  if Intr = High then … else …
  end Sporadic_Code;
  procedure Initiallise is …

HLW Handler -- 2

```
task Thread;

protected Buffer is
        procedure Start(Intr : Water_Mark);
        entry Wait_Start(Intr : out Water_Mark);
 private
        W : Water_Mark; …
 end Buffer;

 procedure Start(Intr : Water_Mark) is
 begin
   Buffer.Start(Intr);
 end Start;

 protected body Buffer is ...
```



HLW Handler - 3

```
task body Thread is
     Intr : Water_Mark;
begin
      Initiallise;
      loop
        Buffer.Wait_Start(Intr);
        Sporadic_Code(Intr);
      end loop;
end Thread;

end Pump.Hlw_Handler;
```

Note: no over-run code