Real-Time Systems

Hermann Härtig

Real-Time Programming Languages

(ADA and Esterel as Examples)

TECHNISCHE UNIVERSITÄT DRESDEN

1/11/17

RT-HLL Issues

- Concurrency and Synchronization/Communication
- Time
  - Access to
  - Control over ("timeout", ...)
- Scheduling/Resource Management
  - Built in
  - Explicit
- Recurrent processes

RT Language Classes

Synchronous HLL (tick driven)

- Esterel
- Lustre
- (State Charts)

Imperative HLL with rt-extensions

- ADA
- RT-Java
- PEARL
- CHILL
- RT-Euclid (designed to enable static analysis)

Esterel

For further study

Gerard Berry,
Esterel Language Primer
http://www.esterel-technologies.com/files/primer.zip

Video of Artist summer school 2008
http://www.artist-embedded.org/artist/Videos-Slides

Caveat:

This lecture: introduction into principles only

Ignored: Extensive Tool Chain, Verification, ...

Esterel at a glance

Starting Point (Berry):

"Ideal Systems produce their outputs synchronously with their inputs."

Esterel:

Most statements are instantaneous

(starts and terminates at the same instant of time)

Stepwise execution,

everything completes in each step/cycle/tick

Time consumption explicit (e.g., „Pause")

Esterel: „Statements“

Consume no time (unless explicitly said otherwise)

- Await A: „consumes one A“
- Pause: „consumes one time step“ (tick)

- X:= Y: assigns values to variables
- S1;S2
- S1 || S2
- Loop S end
  starts s, repeats if not terminated
  (s must consume time)

Esterel „Data“: Variables and Signals

Variable: Value of any type

Signal: Value and Status

Value of any type

Status: Present/non present

Newly evaluated at every step

present when emitted

Signals

- Emit x(y): sets signal x present, assigns value y
- ?σ current value:
  value just emitted (if so) or value of previous instant (otherwise)
  pre(?S): previous value
- Present σ then s1 else s2 end (conditional)
- Abort S when σ do R end abort;
  starts S, terminates when σ becomes active, does R
- Suspend S when σ
  suspends S when σ active
  no emission when suspended
- Trap σ in S end trap
  starts S, aborts when σ present

Signals vs Variables

Gerard Berry

```
Emit Count(pre(?Count) + 1) vs V:= V+1;
```

Beware:

writing “emit COUNT(?COUNT+1)” is tempting but incorrect.
Since ?COUNT is the current value of COUNT, it cannot be incremented and reemitted right away as itself. It is necessary to use the previous value pre(?COUNT).

More Statements

- halt: loop pause end
- await σ : abort halt when σ end abort
- sustain x(t): loop emit x(t); pause end
- loop S each R

  restarts S at each occurance of R
- every σ do S end every:

  await σ ; loop S each σ

Examples (all by Berry): ABRO

Gerard Berry

Specification ABRO:
Emit an output O as soon as two inputs A and B have occurred.
Reset this behavior each time the input R occurs.

module ABRO:

```
input A, B, R;
output O;
loop
    [ await A || await B ];
  emit O
each R
end module
```

Counting

Gerard Berry

Specification COUNT:
Count the number of occurrences of the input I seen so far, and broadcast it as the value of a COUNT signal at each new I.

module COUNT:

```
input I;
output COUNT := 0 : integer;
every I do
  emit COUNT(pre(?COUNT) + 1)
end every
end module
```

Beware:
“emit COUNT(?COUNT+1)” is tempting but incorrect.

Speed

Gerard Berry

Specification SPEED:
Count the number of centimeters run per second, and broadcast that number as the value of a Speed signal every second.

module SPEED:

```
input Centimeter, Second;
relation Centimeter # Second;
output Speed : integer;
loop
  var Distance := 0 : integer in
    abort
      every Centimeter do
        Distance := Distance+1
      end every
    when Second do
      emit Speed(Distance)
    end abort
  end var
end loop
end module
```

ADA

Used intensively, e.g. Military, Aircraft (B777), Space

> "most commonly used language in US weapons modernization"

Ada 83 - result of a competition ...

Ada 95 - major redesign (ISO/IEC 8652: 1995)

Ada 2005, includes Ravenscar: subset

Annex: Real-Time Systems

Few general points

Ada has “Annexes”:
  in this lecture: Real-Time Annex

Ada has “profiles”:
  relevant for this lecture “Ravenscar”
  reduced functionality for Hard-RT

Ada has “pragmas” (compiler directives)

CAVEAT:
  In this lecture: very limited extract relevant for RTS
  Especially, not covered explicitely:
  Packages(library), OO, Type-System, Generics, exceptions ..
  we rely on your intuition

THE ADA-RT Book

Concurrent and Real-Time Programming in Ada

by

Alan Burns and **Andy Wellings**

Cambridge University Press

ISBN 978-0521866972

New edition appeared, not yet here!

Most code examples taken from this source.

Many more resources available.

Burns, Wellings

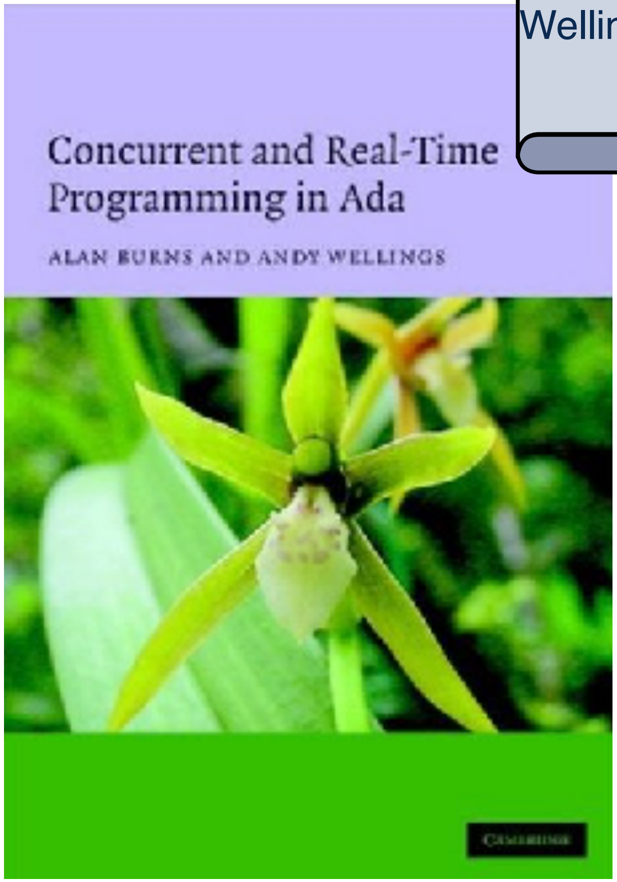


Image Source:
amazon.com/dp/B001GS6TBO/

Concurrency

Tasks

- are entities whose execution may proceed in parallel.
- have a thread of control.
- proceed independently,
  except at points where they synchronize.
- are created and activated via
  - an object declaration or
  - created dynamically using an “access type”
    Ptr:= new …

Example: Operator/Subscriber

Burns, Wellings
Ch. 5.1
Page 80

```
task type Subscriber;

task type Telephone_Operator is
  entry Directory_Enquiry(Person : in Name; Addr : in Address;
                          Num : out Number);
end Telephone_Operator;



S1, S2, S3 : Subscriber;
An_Op : Telephone_Operator;



task body Subscriber is
  Stuarts_Number : Number;
begin
  ...



end Subscriber;
```

Termination of Tasks

Every task has a “master” and “depends” on it:

block, subprogram etc.
containing the declaration of the task object or
of the access object type

Before leaving the master, the parent task waits for all dependent tasks to terminate.

Communication

- Protected objects (ignored in this lecture)
  for synchronized access to shared data
- Rendezvous

  for synchronous communication between tasks
- Unprotected access to shared data (global variables)

The Rendezvous

Based on client/server model:

- One task (client) calls an entry of an other task
- Other task accepts a call (Server)
- Calling task placed on a queue

Example: Operator/Subscriber

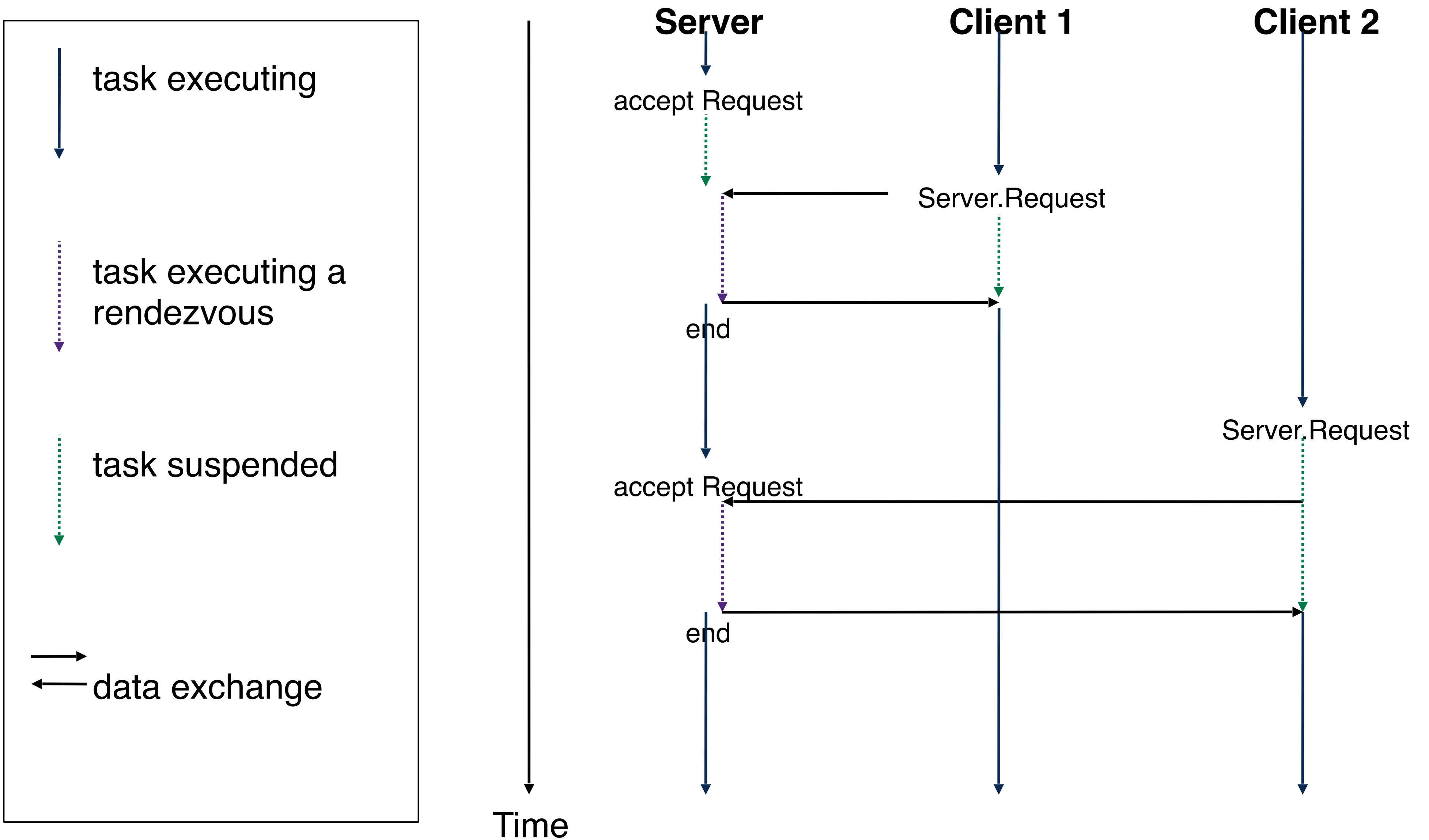
Burns, Wellings
Ch. 5.1
Page 80 f.

```
task body Subscriber is
  Stuarts_Number : Number;
begin
-- ...
  An_Op.Directory_Enquiry("STUART JONES",
    "10 MAIN STREET, YORK", Stuarts_Number);
  -- phone Stuart
-- ...
end Subscriber;
```

```
task body Telephone_Operator is
begin
  loop
    -- prepare to accept next call
    accept Directory_Enquiry(Person : in Name;
          Addr : in Address; Num : out Number) do
      -- look up telephone number and
      -- assign the value to Num
      null; --.RM
    end Directory_Enquiry;
    -- undertake housekeeping such as logging all calls
  end loop;
end Telephone_Operator;
```

In detail

Burns, Wellings
Ch. 5.1
Page 82



Select Statement

A task can have multiple entries:

```
select
   when (expression) =>
   accept E1 do  bla end E1;
or
   when (expression) =>
   accept E2 do  bla end E2;
or ...
end select;
```

Select Statement

- Arbitrary entry, whose expression is evaluated to true, is called.
- Exception if no expression evaluates to true.
- The Boolean expression is evaluated only once per execution of select

  (do not use global variables in when clause !)

More on “select” later (timings)

Priorities

Burns, Wellings
Ch. 13.2
Page 320

```
pragma task_dispatching_policy(policy identifier);
```

Supported by ADA 205:

- Preemptive fixed priority
- Non-Preemptive fixed priority
- Round robin
- EDF
- And mixtures thereof

Fixed Priorities

- task (type) T is   pragma Priority(P); ....
- Distinct Run Queue per (active) priority

  released tasks at the end of queue

  preempted tasks at the beginning

- Such priorities are called the ***base*** priorities of the task

  in contrast to ***active*** priority

Priority Ceiling

Burns, Wellings
Ch. 13.3
Page 325

```
Pragma Locking_Policy(Ceiling_Locking);
Protected object is pragma priority(...) ... ;
```

Implements the “immediate ceiling protocol”

- The object ceiling priority must be maximum priority of any calling task
- the task executing a protected operation executes at the ceiling priority of the protected object

Active Priority

Base priority or

- Ceiling priority if calling a protected object
- The creating task's priority if higher than the base p
- During rendezvous:

  the priority of the task executing the accept statement inherits

  the priority of the calling task

Run-Time Priorities

```
procedure Set_Priority(Priority: Any_Priority;
                       T: Task_ID := Current_Task);

function Get_Priority(T: Task_ID := Current_Task)
                      return Any_Priority;
```

Resets base priority.

EDF Dispatching

Burns, Wellings
Ch. 14.3
Page 336 f.

How to set deadlines:

```
Package ada.dispatching.EDF is
procedure set_deadline (D: in deadline,
                        T: in TaskId)
Procedure DelayUntilAndSetDeadline(...)
Procedure GetDeadline (...)
```

Or

```
Pragma Relative_Deadline(Milliseconds(3))
& Explicit call to set first deadline of periodic task
```

EDF and Ceiling

- Implements “Preemption Level Control Protocol”
  look forward to:
  Dr. Hamann’s class on Real-Time Scheduling
  - Urgency (EDF): absolute deadline
  - Preemption level: relative deadline

- Using Priority Ceiling of Protected Objects as Preemption Level

- Rationale: see Ted Baker(91) and 

(complicated)

Mixed Scheduling Policies, example

Burns, Wellings
Ch. 14.4
Page 347

Specify “priority partitions” to set scheduling disciplines

```
Pragma Priority_Specific_Dispatching
(Fifo_Within_Priorities, 10, 16)

Pragma Priority_Specific_Dispatching
(EDF_Across_Priorities, 2,9)

Pragma Priority_Specific_Dispatching
(RoundRobin_Within_Priorities, 1, 1)
```

Entry Queuing: Implicit Policies

Burns, Wellings
Ch. 13.4
Page 327

Standard: FIFO

```
pragma Queuing_Policy(Priority_Queuing);
```

"the user can override the default FIFO policy with the pragma Queuing_Policy"

per partition (not per entries or tasks)

passing of dynamic priorities as implicit parameters

Explicit Request Ordering: Requeue

Explicit treatment of request orders, example:

- A request enters entry or barrier
- Parameters inspected in body code
- Possible decision: requeue at different entry

Action:

- Requeue a request of a caller to some entry or barrier

Not easy to use !!!

Example

```
protected body AirportGate is
      entry EnterGateBusiness(Ticket)
   begin
      if Ticket.Economy then
      requeue EnterGateEconomy;
      end if;
      HandleBusinessPassenger
   end EnterGateBusiness;

   entry EnterGateEconomy(Ticket)
   when AllBusinessPassengersHaveEntered
   begin HandleEcenomyPassenger end EnterGateEconomy
end AirportGate;
```

Timing events

Burns, Wellings
Ch. 15.2
Page 363

```
Package Ada.Real_time.Timing_events is
--...

Procedure Set_Handler(Event: in out Timing_Event;
At-Time: Time; Handler: Timing_Event_Handler);
--...
Procedure Set_Handler(Event: in out Timing_Event;
At-Time: Time_Span; Handler: Timing_Event_Handler);
```

Causes Handler to be called at chosen times.

Handlers are called by Clock_Interrupt Handler

Must not block.

Used for periodic action and watchdogs

Example: Watchdog

Burns, Wellings Ch. 15.2 Page 364 f.

```
protected Watchdog is
  pragma Interrupt_Priority (Interrupt_Priority'Last);
  entry Alarm_Control;
     -- Called by alarm handling task.
  procedure Call_In;
     -- Called by application code every 50ms if alive.
  procedure Timer(Event : in out Timing_Event);
     -- Timer event code, ie the handler.
private
  Alarm : Boolean := False;
end Watchdog;

Fifty_Mil_Event : aliased Timing_Event;
TS : Time_Span := Milliseconds(50);

Set_Handler(Fifty_Mil_Event, TS, Timer);
```

Example: Watchdog

Burns, Wellings
Ch. 15.2
Page 365

```
protected body Watchdog is
  entry Alarm_Control when Alarm is
  begin
    Alarm := False;
  end Alarm_Control;

  procedure Timer(Event : in out Timing_Event) is
  begin
    Alarm := True;
    -- Note no use is made of the parameter in this example
  end Timer;

  procedure Call_in is
  begin
    Set_Handler(Fifty_Mil_Event, TS, Timer);
    -- This call to Set_Handler cancels the previous call
  end Call_in;
end Watchdog;
```

Time: Delay Statement

```
delay
   duration
   point in time

delay 5.0;          -- delay for at least 5 seconds
delay until A_Time; -- delay at least until A_Time
```

specifies minimum delay

Delay and Select, server side

```
select
   accept An_Entry do  bla
   end An_Entry;
or
   delay 10.0;
   Put("An_Entry: timeout");
end select;
```

Select terminates if entry is not called within 10 time units.

Delay and Select, client side(1)

```
select
   Operator.Enquiry()
or
   delay 10;
end select;
```

Select terminates if entry is not accepted within 10 time units.

Only one call alternative allowed

Client side (2): “Asynchronous” Select

Burns, Wellings
Ch. 9.3
Page 202

```
select trigger
triggering_alternative --- (entry-call or delay)
then abort
abortable_part
end select;
```

- If delay or entry-call complete before the abortable part, the abortable part is aborted
- abortable_part must not an accept statement

Example

```
select
    delay 5.0;              -- triggering alternative
then abort
    CalculationComplete:= false;
    Invert_Giant_Matrix(M);   -- abortable part
    CalculationComplete:= true;
end select;
```

Careful: notice the race condition !

Example: Watchdog

Burns, Wellings
Ch. 6.3
Page 104

```
task type Watchdog is
  entry All_Is_Well;
end Watchdog;

task body Watchdog is
begin
  loop
    select
      accept All_Is_Well;
    or
      delay 10.0;
      -- signal alarm, potentially the client has failed
      exit;
    end select;
  end loop;
  -- any further required action
end Watchdog;
```

Example: Operator/Subscriber

Burns, Wellings
Ch. 6.9
Page 119

```
task type Subscriber;

task body Subscriber is
  Stuarts_Number : Number;
begin
  loop
    --...
    select
      An_Op.Directory_Enquiry("STUART JONES",
            "10 MAIN STREET, YORK", Stuarts_Number);
      -- log the cost of a directory enquiry call
    or
      delay 10.0;
      -- phone up his parents and ask them,
      -- log the cost of a long distance call
    end select;
    --...
  end loop;
end Subscriber;
```

Simple Periodic Task With Static Priority

Burns, Wellings
Ch. 14.3
Page 345

```
task A is
  pragma Priority(5);
end A;

task body A is
  Next_Release: Real_Time.Time;
begin
  Next_Release := Real_Time.Clock;
  loop
    -- code
    Next_Release := Next_Release + Real_Time.Milliseconds(10);
    delay until Next_Release;
  end loop
end A;
```

Recurrent Tasks as Package (1)

Burns, Wellings

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Real_Time; use Ada.Real_Time;
package Periodic_Scheduler is
  procedure Set_Characteristic(T : Task_Id; Period : Time_Span;
                               First_Schedule : Time);
  procedure Wait_Until_Next_Schedule; -- potentially blocking
end Periodic_Scheduler;
```

- - Periodic tasks can now be encoded as

```
task Periodic_Task;
task body Periodic_Task is

begin
  loop
    -- statements to be executed each period
       Periodic_Scheduler.Wait_Until_Next_Schedule
  end loop;
end Periodic_Task;
```

Recurrent Tasks as Package (2)

Burns, Wellings
Ch. 12.4
Page 315

```
procedure Set_Characteristic(T : Task_Id; Period : Time_Span;
                             First_Schedule : Time) is
begin
  Set_Value((Period, First_Schedule), T );
end Set_Characteristic;

procedure Wait_Until_Next_Schedule is
  Task_Info : Task_Information := Value;
  Next_Time : Time;
begin
    Next_Time := Task_Info.Period +
         Task_Info.Next_Schedule_Time;
    Set_Value((Task_Info.Period,Next_Time));
   delay until Next_Time;
end Wait_Until_Next_Schedule;
end Periodic_Scheduler;
```

Missing in this RT-HLL lecture

- RT-Java and RT-Garbage Collection
- Language with built-in periodic processes

To take away ...

- Principles of synchronous languages
- Mechanisms to explicitely handle timing
- Mechanisms to handle asynchronous events
- “scheduling” of processes, queues, ...