The Evolution of Real-Time Programming Revisited

Programming the Giotto Model in Ada 2005

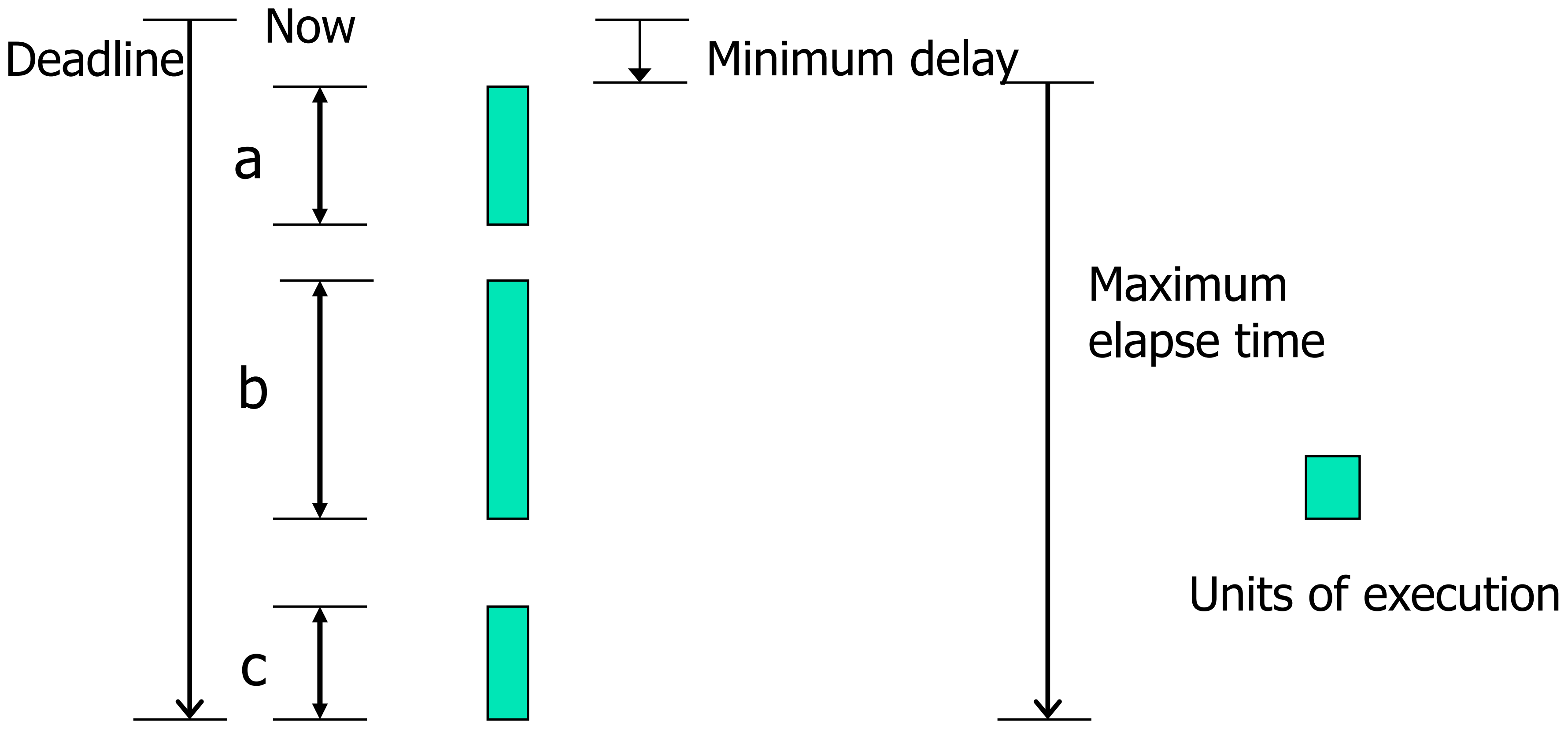
Structure

- Kirsch and Sengupta original paper
- Temporal Scopes
- Giotto
- Controlling Input and Output Jitter in Ada
- Conclusions

Kirsch and Segupta

- Physical execution time model
  - assembly languages
- Bounded execution time model
  - Ada, Real-Time Java, RTOS
- Zero execution time model
  - Esterel, Lustre
- Logical execution time model
  - Giotto

Temporal Scopes

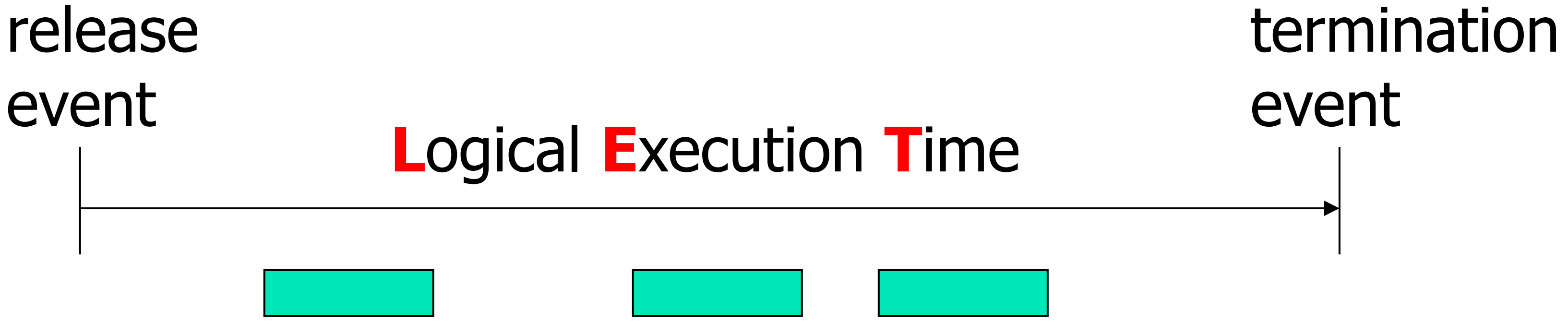


Maximum execution time = a + b +c

Giotto

- A language for control applications
- Output produced at deadline, not before
- A task is logically executing from release to deadline
- Supports
  - Time Safety and
  - I/O Composability

The Logical Execution-Time Model



Input ports updated at release event

Actual execution can occur at any time

Output ports updated at termination event

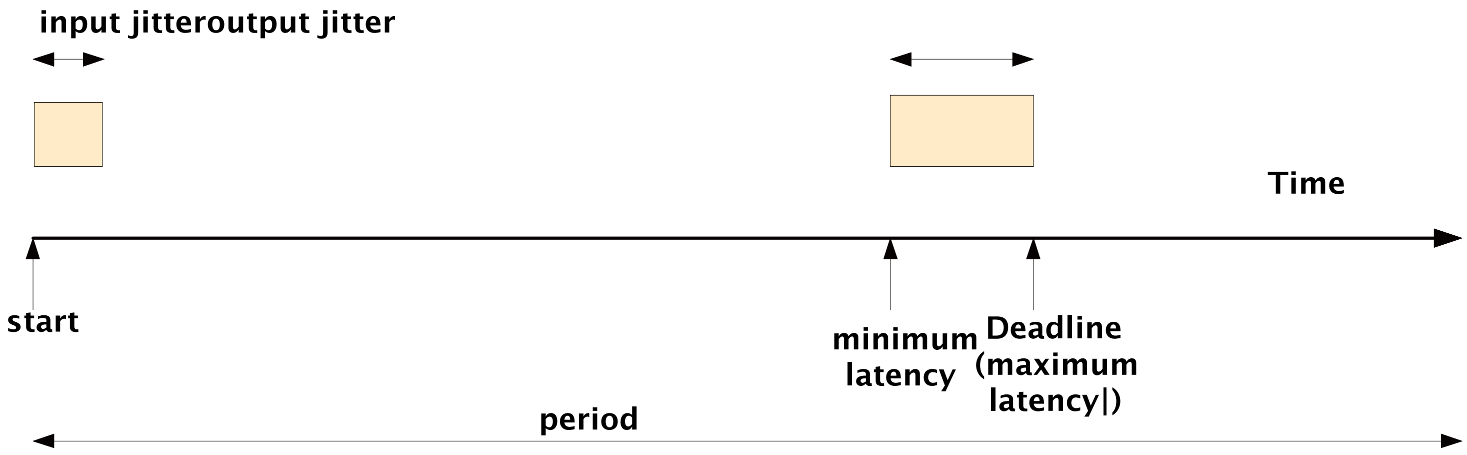
Example – pseudo code

```
sensor
  port temperature type integer range 10 .. 500
  port pressure type integer range 0 .. 750
actuator
  port heater type (on, off)
  port pump type integer 0 .. 9
input
  T1 type integer range 10 .. 500
  PI type integer range 0 .. 750
output
  TO type (on, off)
  PO type integer 0 .. 9
```

Controlling I/O Jitter

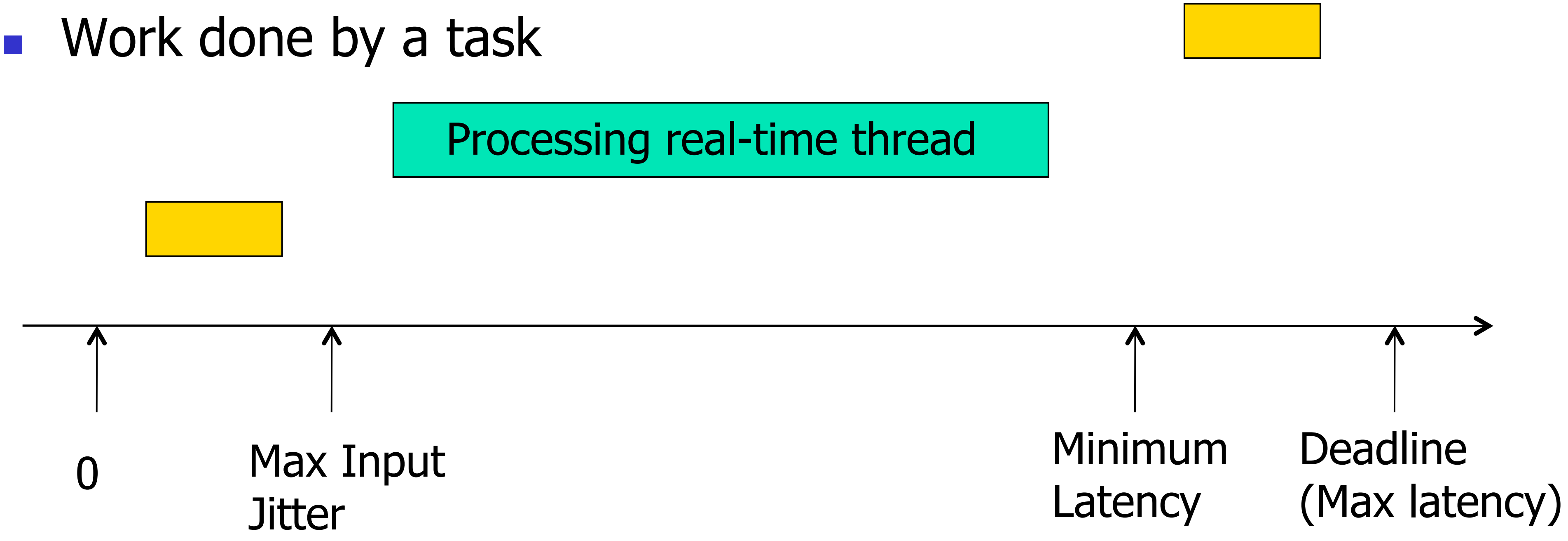
- A periodic control task needs to take input from the environment is a very regular way, and similarly produce output with little variation in time
  - Input jitter
  - Output jitter
- This is the key issue the LET model addresses
  - I/O composability
  - Time safety by schedulability analysis

Example of Input/Output Jitter



Controlling Input and Output Jitter

- Sensors and actuators are read and written by asynchronous event handlers
- Work done by a task



Controlling jitter in Ada

- Use a timing event for input and a separate timing event for output
- Use a task for processing the input data to produce the output
- Assume a period of 40ms in a controller

Sensor Reader spec

```
protected type Sensor_Reader is
  pragma Interrupt_Priority (Interrupt_Priority’Last);
  procedure Start;
  entry Read(Data : out Sensor_Data);
  procedure Timer(Event : in out Timing_Event);
end Sensor_Reader;

Input_Jitter_Control : Timing_Event;
Input_Period : Time_Span := Milliseconds(40);
```

Sensor Reader body

```
protected body Sensor_Reader is
  procedure Start is
  begin
    Reading := Read_Sensor;
    Next_Time := Clock + Input_Period;
    Data_Available := True;
    Set_Handler(Input_Jitter_Control,
                Next_Time, Timer'Access);
  end Start;

  entry Read(Data : out Sensor_Data) when Data_Available is
  begin
    Data := Reading;
    Data_Available := False;
  end Read;
```

Sensor Reader body

```
  procedure Timer(Event : in out Timing_Event) is
  begin
     -- Reading from sensor interface
     Data_Available := True;
     Next_Time := Next_Time + Input_Period;
     Set_Handler(Input_Jitter_Control, Next_Time,
                 Timer'Access);
  end Timer;

end Sensor_Reader;
```

Output jitter control

- A type is also defined for output jitter control (`Actuator_Writer`)
- Assuming a deadline of 30ms (period is 40ms) and max output jitter of 4ms:

```
SR.start; -- of type Sensor_Reader
delay 0.026; -- ie 26ms later
AW.start;   -- of type Actuator_Writer
```

Controlling task

```
task type Control_Algorithm
                        (Input : access Sensor_Reader;
                         Output : access Actuator_Writer);
task body Control_Algorithm is
  Input_Data : Sensor_Data;
  Output_Data : Actuator_Data;
begin
  loop
    Input.Read(Input_Data);
    -- process data;
    Output.Write(Output_Data);
  end loop;
end Control_Algorithm;
```

A Three-task model

- The main disadvantage of the LET model (and the Ada solution) is that all input and output is treated identically
- It is not possible to take in to account processing that is more tolerant to the noise introduced by input jitter
- A three-task solution allows each tasks to be given a deadline and be scheduled accordingly

Conclusions

- Kirsch and Sengupta do not take into account “expressive power” and “ease of use”
- The LET model has limited expressive power but has great ease of use
  - but only if your application requirements exactly match the supported model
- Ada 2005 has greater expressive power
  - Lower-level mechanisms allow more applications requirements to be met
  - Ease of use is the compromise
  - Real-time utilities can help