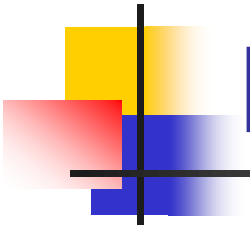
Deadline-Aware Programming and Scheduling

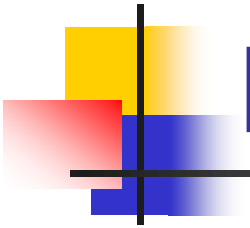
Alan Burns and Andy Wellings



Real-Time Systems

- Correctness depends on satisfying temporal requirements as well as functional ones
- This usually takes the form of meeting **deadlines**
- **Deadlines** should therefore be an abstraction available to the programmer
- Programs should be able to identify and react to missed deadlines

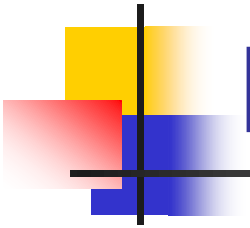
RTS York



Deadlines

- Key notions
  - Tasks give rise to a sequence of **jobs**
  - **Relative deadline**, relative to release of a job from a task, denoted by D
  - **Absolute deadline**, time by which job must finish, denoted by d
  - d = D + s (start time)

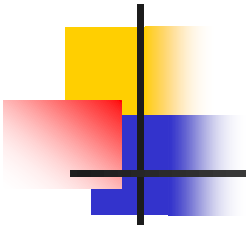
RTS York



Real-Time Scheduling

- With concurrent systems the most effective way of scheduling tasks is **EDF – earliest deadline first**
- This applies to single processor systems, and multiprocessor systems with static partitioning
- The run-time must therefore be aware of task deadlines – and obtain this data from the program

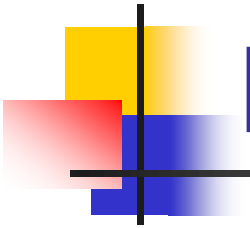
RTS York



Paper's Contribution

- Concurrent programs that share resources need to utilise an effective sharing protocol
- For Fixed Priority scheduling a **priority ceiling protocol** (PCP) is usually employed
- For EDF, the **stack resource policy** (SRP) is the protocol of choice
- An alternative protocol has recently been defined (**deadline floor protocol**, DFP)
- In this paper we consider how DFP can be supported in Ada

RTS York



Priority Inversion

- A well known problem with fixed priority systems is **priority inversion**
  - Low priority task locks a resource (a protected object)
  - High priority tasks must wait if they need to access these locked resources
  - Middle priority tasks execute in preference to Low and hence in preference to High

RTS York

Priority Inheritance

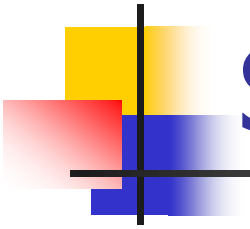
- Solution is to use some form of priority inheritance such as PCP (**Priority Ceiling Protocol**)
  - All protected objects (POs) have ceiling priorities
    - Max pri of tasks that use the PO
  - When a task accesses a PO its priority is raised to ceiling
  - This reduces inversion, stops deadlocks, provides mutual exclusion etc

RTS York

Resource Sharing in EDF

- Inversion also occurs with EDF
  - Task with short deadline needs resource held by a task with long deadline
- Standard solution is **Stack Resource Policy** (SRP) – this is supported in Ada

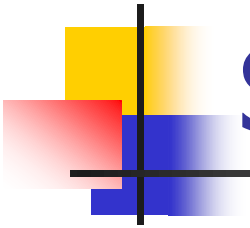
RTS York



Stack Resource Policy

- Not going to define this is detail
  - Tasks have deadlines and preemption levels
  - To preempt, a task must have shorter deadline and higher preemption level
- Has all the properties of PCP

RTS York



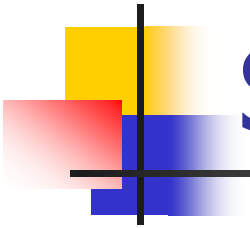
SRP in Ada

- Decided to support SRP with existing (modified) Locking Policy
  - EDF is defined to work in a given band of priority
  - Priority is used for preemption level
  - By default, the active priority of an EDF task is the lowest priority in its EDF priority band
  - A task will inherit priorities; in particular, when an EDF task executes a protected operation it will inherit the priority (preemption level) of the protected object
  - But, for EDF tasks, the ARM must defines a further source of priority inheritance

RTS York

SRP Rule

- For arbitrary task T it will be assigned the highest priority P, if any, less than the base priority of T such that one or more tasks are executing within a protected object with ceiling priority P and task T has an earlier deadline than all such tasks; and furthermore T has an earlier deadline than all other tasks on ready queues with priorities in the given EDF_Across_Priorities range that are strictly less than P



SRP Rule

- This is not straightforward
  - Initially rule was wrong and had to be modified [24]
  - First implementation had an error [15]
  - Correct implementation is far from efficient [1]
- So perhaps there is a better way

RTS York

Deadline Floor Protocol

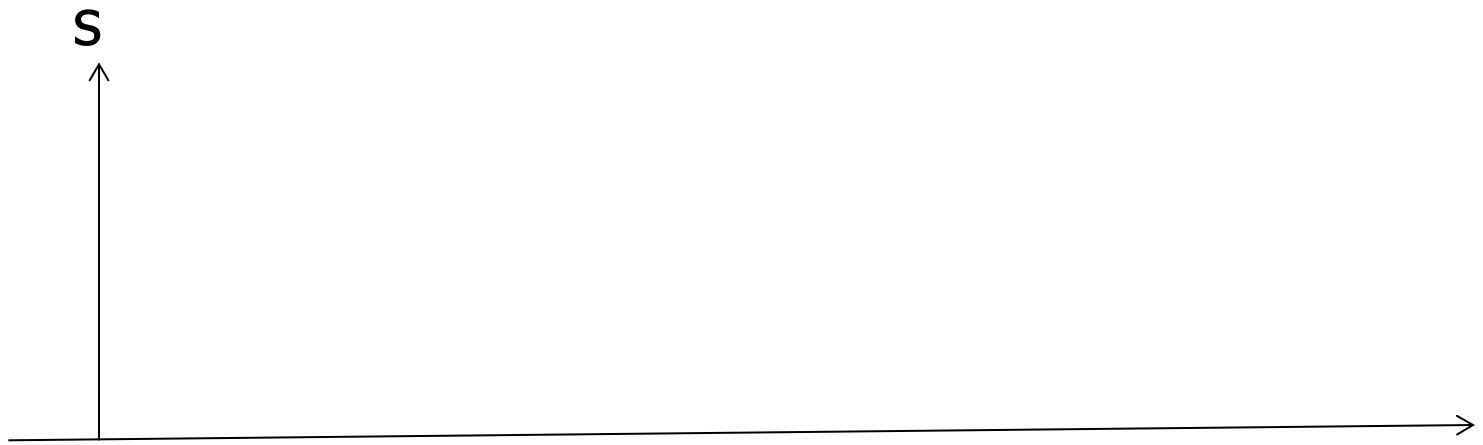
- All tasks have relative deadlines
  - deadline is release time + relative deadline
- All POs have relative deadlines
  - The minimum of the relative deadlines of tasks that use the PO
  - As minimum is used the protocol is called Deadline Floor (as it works in the same way as Priority Ceiling)
  - Priority is not used
    - All tasks have the same priority

RTS York

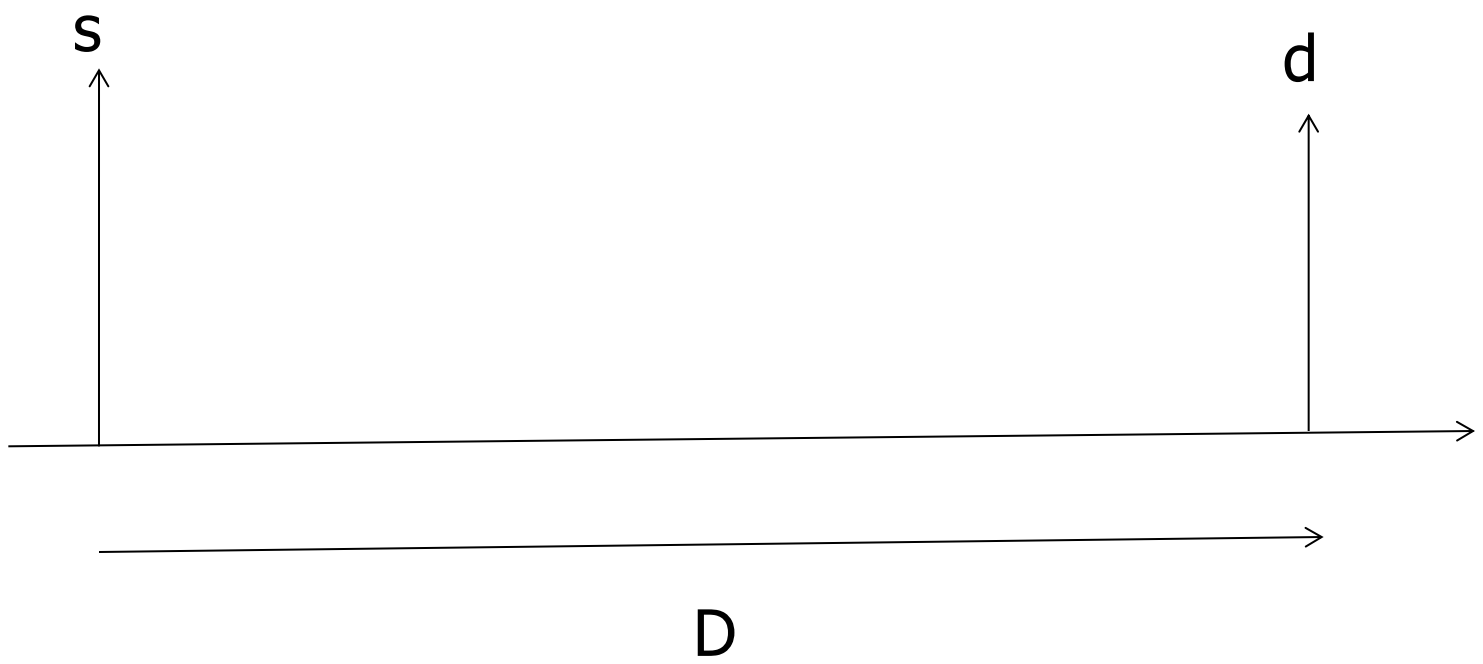
Deadline Floor Protocol

- When a task released at time s, with relative deadline D calls a PO with deadline floor F, at time t
  - $d = s + D$
  - $F \leq D$
  - $s < t$
- Then
  - Its current deadline (d) is reduced from (s + D) to (t + F)

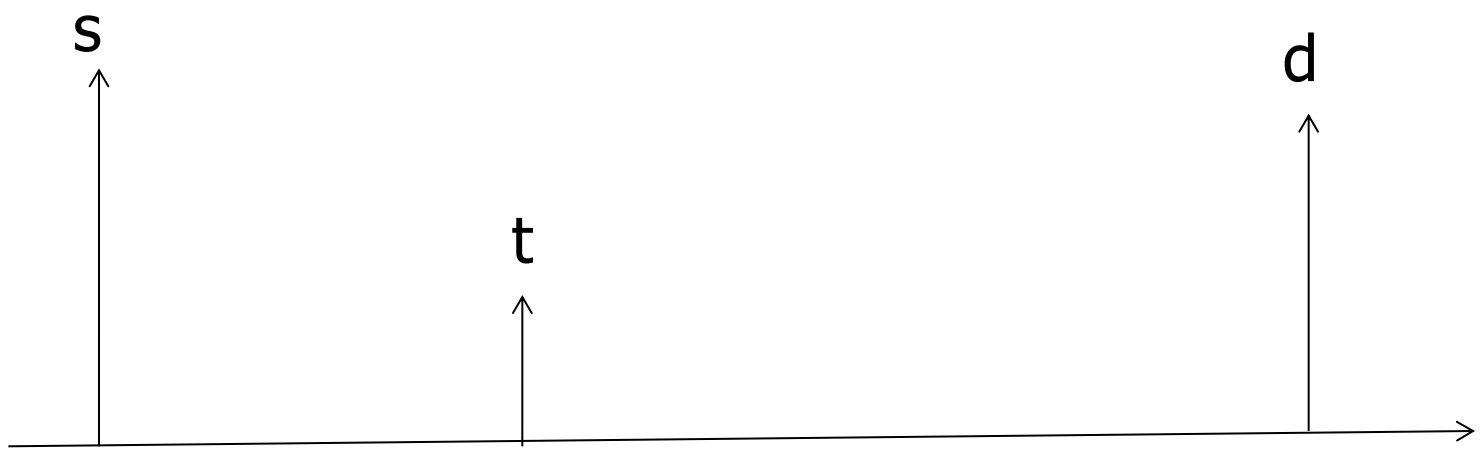
Example



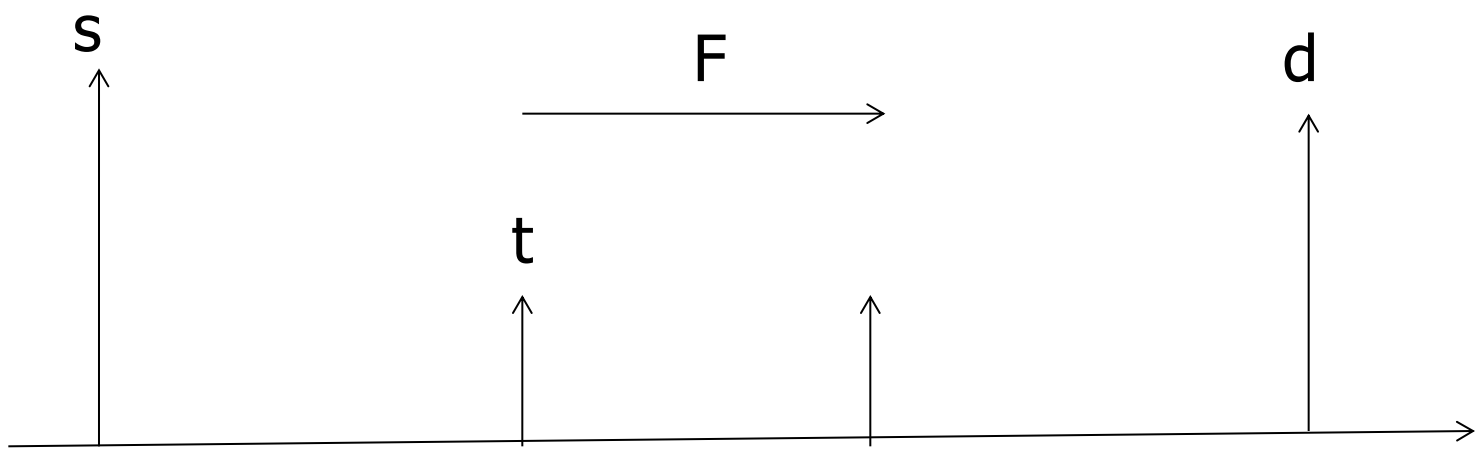
Example



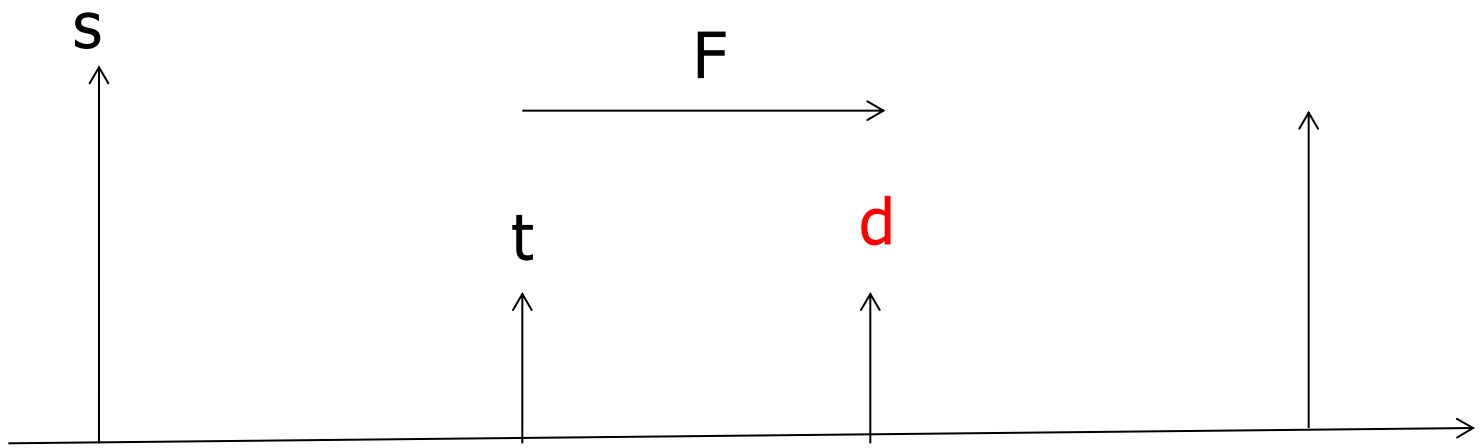
Example



Example



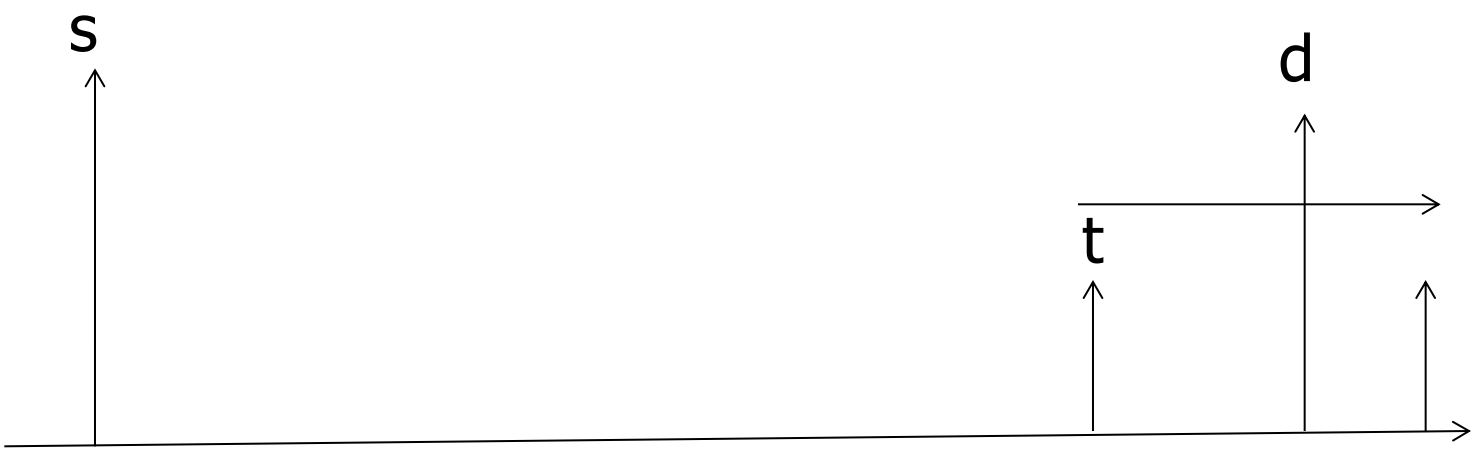
Example

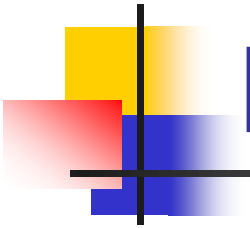


Deadline Floor Protocol

- When a task released at time s, with relative deadline D calls a PO with deadline floor F, at time t
  - $d = s + D$
  - $F \leq D$
  - $s < t$
- Then
  - Its current deadline (d) is reduced from (s +D) to (t + F)
  - Unless $t + F > s + D$ (in which case there is no change to d)

Example

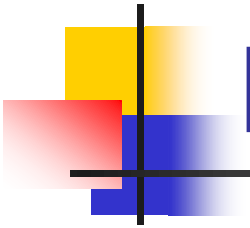




DFP Properties

- It has been proved that DFP has all the excellent scheduling properties of PCP and SRP [7,9]
- It has been shown to be more efficient to implement than SRP [1]
- I would argue it is more intuitive and hence easier to understand

RTS York



Deadlines and DFP in Ada

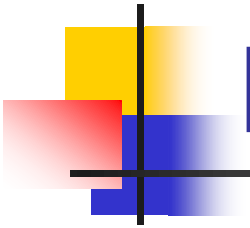
- All tasks must have a relative deadline assigned via an aspect/pragma or a routine defined in a library package
- Protected objects must have also a relative deadline (floor) assigned via an aspect/pragma
- Default relative deadline values must be defined for tasks and protected objects (and their types)

RTS York

Deadlines and DFP in Ada

- Rules for EDF scheduling must be extended to include a new locking policy: Floor_Locking
- Rules for EDF scheduling need simplifying to remove the `across priorities' feature of the current definition
- For completeness (and parity with priority ceilings) means of modifying the relative deadline attribute of tasks and protected objects should be defined

RTS York



Library Packages

- Deadline and relative deadline are fundamental concepts and should be supported even if EDF is not used

- We propose a new library package, `Deadlines`

```
package Ada.Deadlines is
   subtype Deadline is Real_Time.Time;
   subtype Relative_Deadline is
                           Real_Time.Time_Span;

   Default_Deadline : constant Deadline :=
                           Real_Time.Time_Last;
   Default_Relative_Deadline : constant
            Relative_Deadline :=
                  Real_Time.Time_Span_Last;

   procedure Set_Deadline(D : in Deadline;
      T : in Task_Identification.Task_ID :=
           Task_Identification.Current_Task);
```

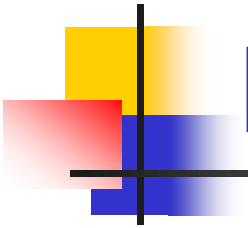
```
function Get_Deadline(T : in
        Task_Identification.Task_ID :=
              Task_Identification.Current_Task)
        return Deadline;

procedure Set_Relative_Deadline(R : in
          Relative_Deadline;
          T : in Task_Identification.Task_ID :=
              Task_Identification.Current_Task);

function Get_Relative_Deadline(T : in
          Task_Identification.Task_ID :=
               Task_Identification.Current_Task)
        return Relative_Deadline;

procedure Delay_Until_And_Set_Deadline(
         Delay_Until_Time : in Real_Time.Time;
         TS : in Real_Time.Time_Span :=
                     Get_Relative_Deadline);

end Ada.Deadlines;
```



Key Changes

- Change of name and library position
- Introduction of a type for relative deadline and a default value
- Set and Get routines added for relative deadlines
- A default relative deadline provided for `Delay_Until_And_Set_Deadline`
- Aspect/Pragma defined to set initial deadline and relative deadline of a task

RTS York

New Locking Policy

- Whenever a task is executing outside a protected action, its active deadline is equal to its base deadline
- When a task executes a protected action its active deadline will be reduced to (if it is currently greater than) `now' plus the deadline floor of the corresponding protected object
- When a task completes a protected action its active deadline returns to the value it had on entry
- When a task calls a protected operation, a check is made that there is no task currently executing within the corresponding protected object; `Program_Error` is raised if this check fails

RTS York

New Dispatching Policy

- Currently `EDF_Across_Priorities`
- Now `EDF_Within_Priorities`

- In mixed and hierarchical scheduling use both `Ceiling_Locking` and `Floor_Locking`

- In Ravenscar do not allow changes to relative deadline

RTS York

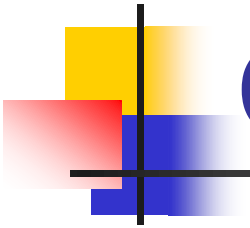
Programming Template

```
task type Periodic_Task
              Period_In_Milliseconds : Positive;
              Rel_Deadline_In_Milliseconds : Positive);

task body Periodic_Task is
    Interval : Time_Span :=
               Milliseconds(Period_In_Milliseconds);
    Rel_Deadline : Time_Span :=
            Milliseconds(Rel_Deadline_In_Milliseconds);
    Next_Release_Time : Time;
  begin
    Set_Relative_Deadline(Rel_Deadline);
    Next_Release_Time := Clock;
    Set_Deadline(Next_Release_Time + Rel_Deadline);
```

```
loop
   select
     delay until Get_Deadline;
          -- handle deadline miss here
     then abort
          -- undertake the work of the task
   end select;
   Next_Release_Time := Next_Release_Time + Interval;
   Delay_Until_And_Set_Deadline(Next_Release_Time);
end loop;

end Periodic_Task;
```



Conclusions

- Deadlines are key to real-time programs, they should be a first class abstraction even with priority based scheduling
- With EDF scheduling, we argue that DFP is a better protocol to support than SRP
- Ada supports two level scheduling
  - Priority (preemptive or non-preemptive) at the top level
  - FIFO, RR or EDF at secondary level
- Previously EDF did not fit this scheme
- Now with DFP the clear two level structure is maintained

RTS York

IRTAW 17 (2015)

- Vermont, New York
- Within the week 20-24 April 2015
- Papers (position statements) by 4th Feb 2015
- See call for papers for details
- Participation by invitation only