RTAS 2022

Formal Correctness Proof for an EDF Scheduler Implementation

Florian Vanhems – florian.vanhems@univ-lille.fr

19th July, 2022

Motivations

- Critical software of embedded systems
- EDF is optimal and well understood
- Experiment with our proof methodology

Aspirations

- Proof of concept - not a production ready scheduler
- Share our conception and understanding of software proofs

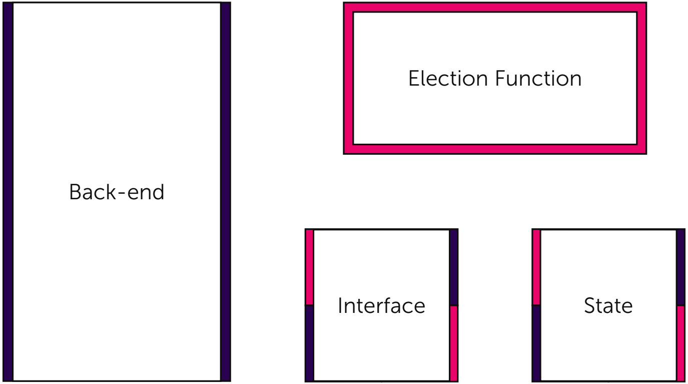
Overview

First formally proven implementation

of an Earliest Deadline First scheduler

for arbitrary sequences of jobs

Overview of the scheduler



Star of the talk

Election Function

Star of the talk

Election Function

Written directly in Coq

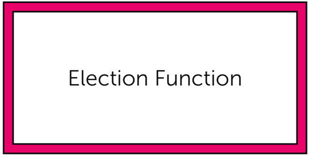
Star of the talk

Election Function

Written directly in Coq

Formally proven properties

Star of the talk



Written directly in Coq

Formally proven properties

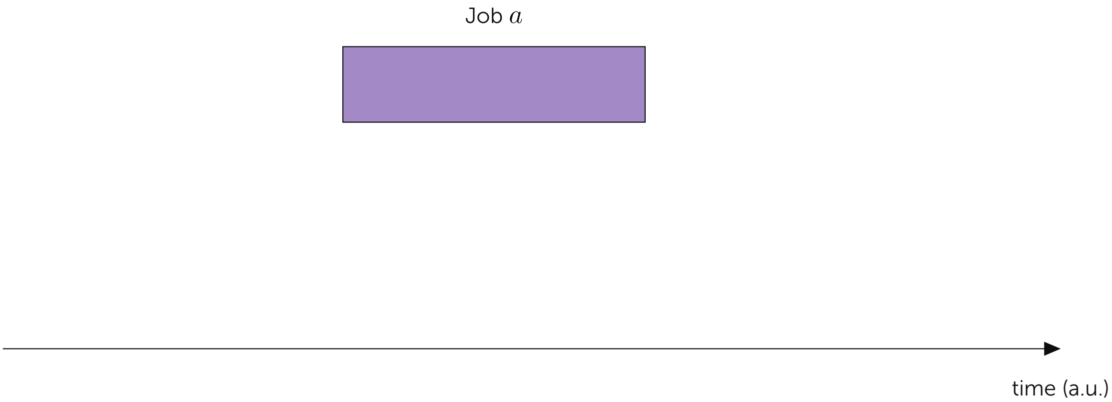
Translated word to word to C

General informations about the scheduler

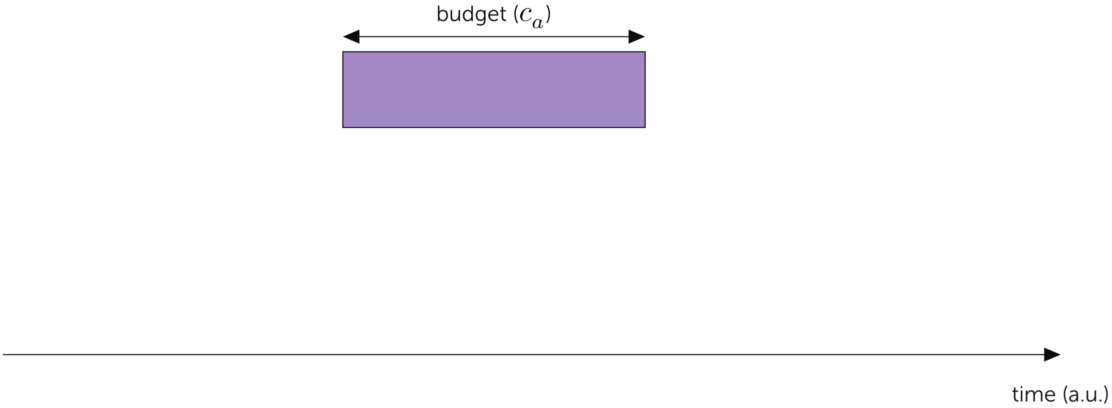
- Schedules arbitrary sequences of jobs (as opposed to *tasks*)
- Periodically called
- Online [1]

[1]: while the election function schedules online, our scheduler feeds it hard coded jobs for simplicity

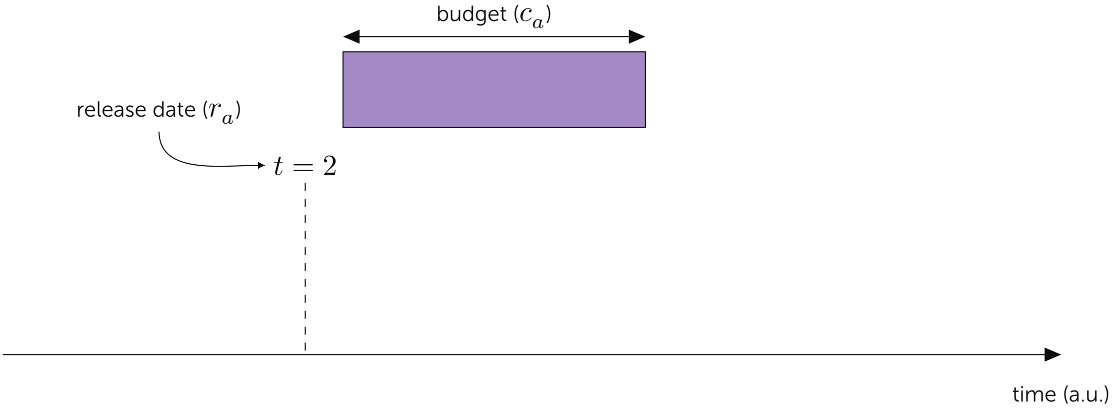
Election function of an Earliest Deadline First scheduler



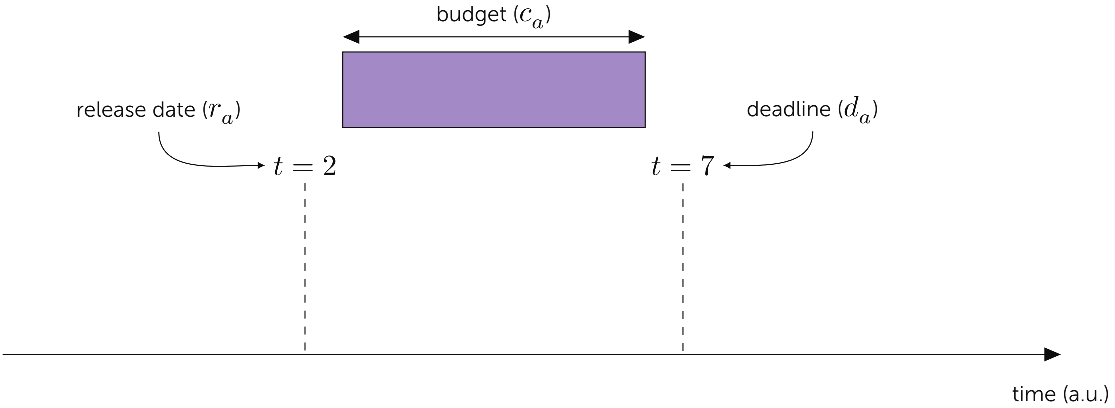
Election function of an Earliest Deadline First scheduler



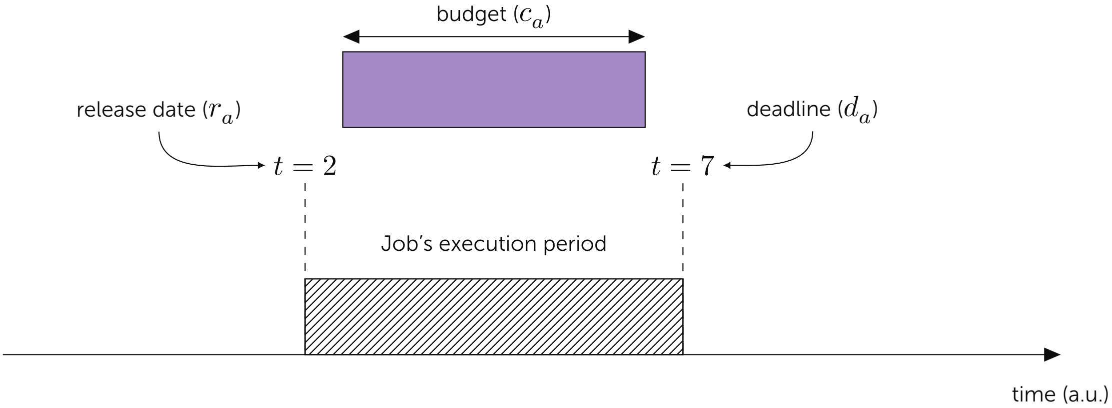
Election function of an Earliest Deadline First scheduler



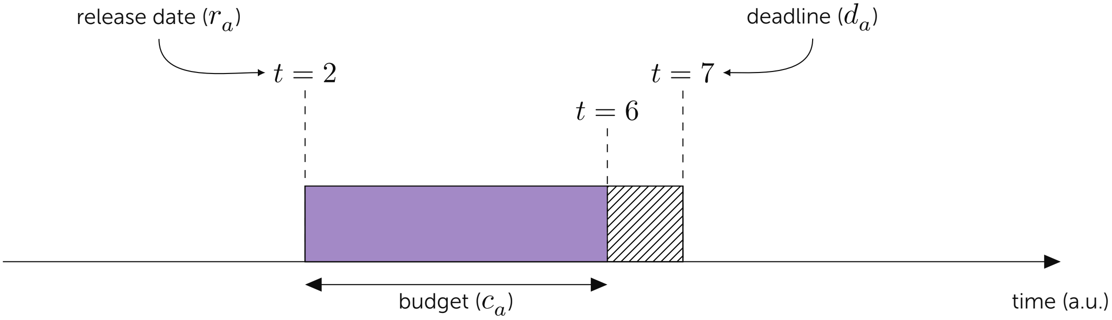
Election function of an Earliest Deadline First scheduler



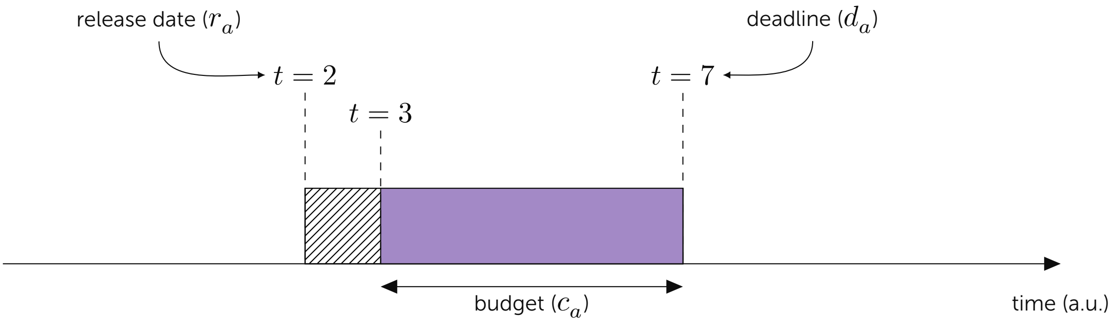
Election function of an Earliest Deadline First scheduler



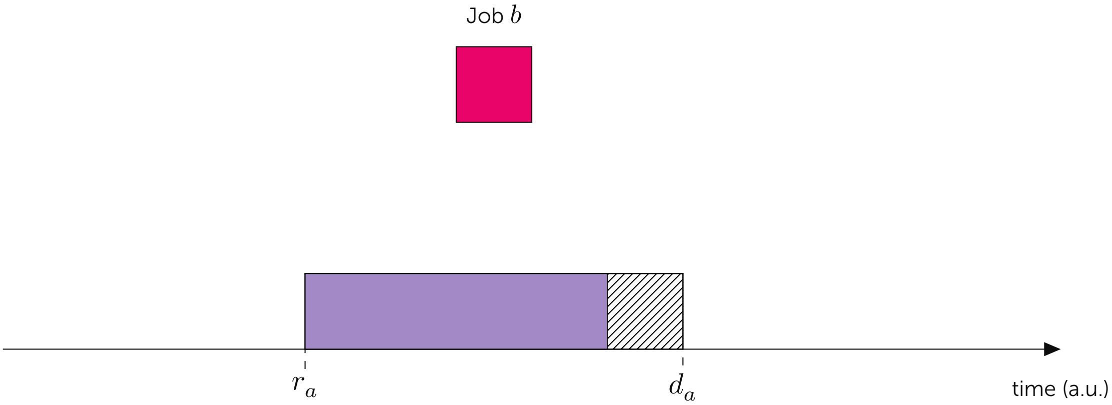
Election function of an Earliest Deadline First scheduler



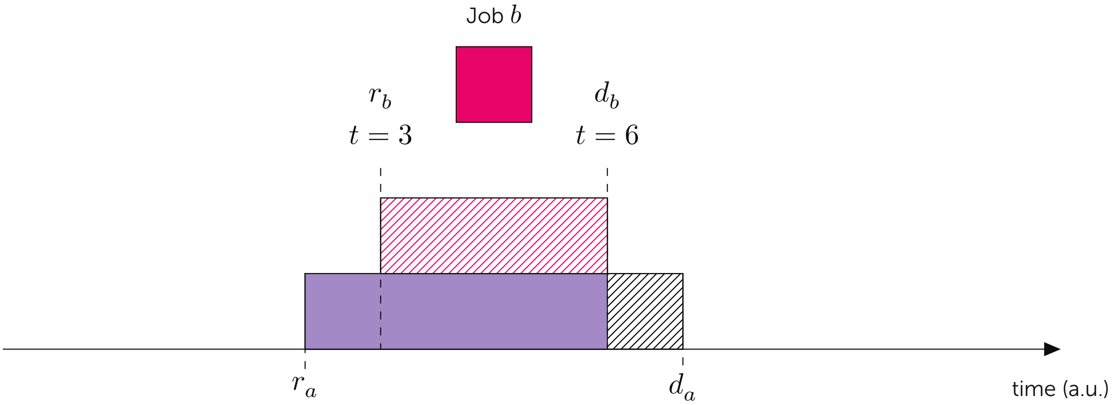
Election function of an Earliest Deadline First scheduler



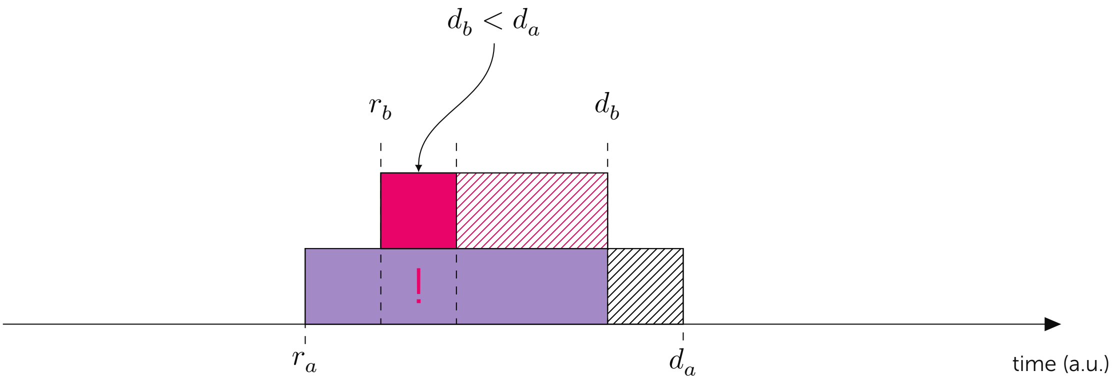
Election function of an Earliest Deadline First scheduler



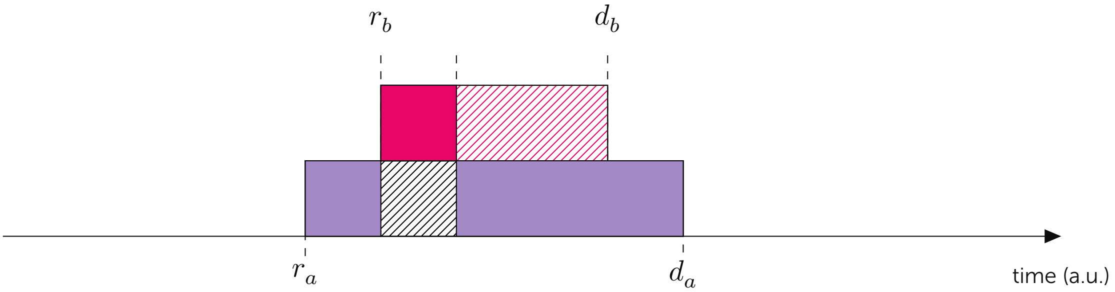
Election function of an Earliest Deadline First scheduler



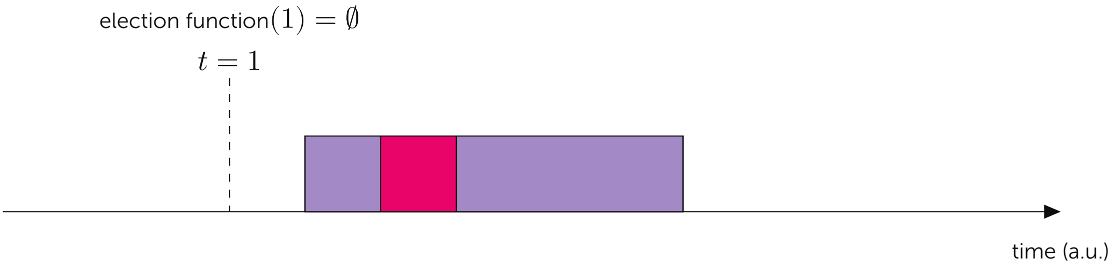
Election function of an Earliest Deadline First scheduler



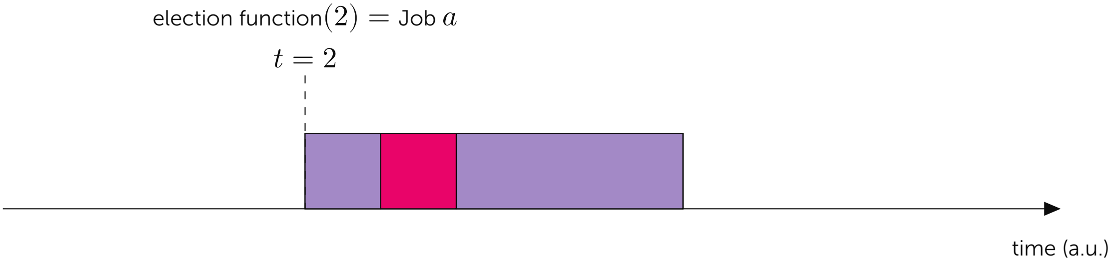
Election function of an Earliest Deadline First scheduler



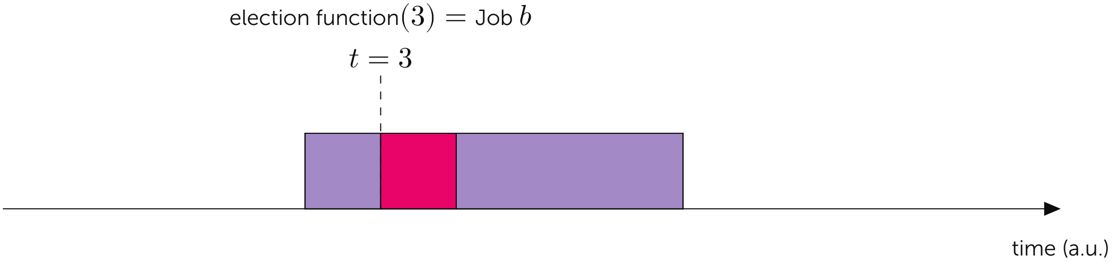
Election function of an Earliest Deadline First scheduler



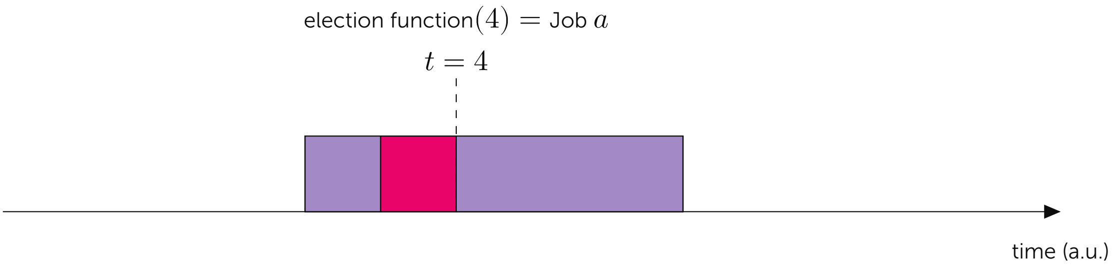
Election function of an Earliest Deadline First scheduler



Election function of an Earliest Deadline First scheduler



Election function of an Earliest Deadline First scheduler



Proofs & Hypotheses

So what did we prove?

Schedulable job sets are scheduled such that no job misses its deadline

Schedulability property

Given any two moments t, t', let $\Gamma_{t,t'}$ be the set of jobs j to schedule in the interval $[t, t']$.
If the sum of the budget c_j of the jobs in that set is less than $t' - t$, then the job set is *schedulable*.

Definition (Schedulability property)

$$\forall\, t, t'.\, t < t' \implies \sum_{j \in \Gamma_{t,t'}} c_j \leqslant t' - t$$

Well-formedness assumptions

For each job :

- $r_i + c_i \leqslant d_i$: the deadline comes late enough for the job to complete its execution if executed alone on the processor
- $0 < \delta_j \leqslant c_j$: the actual duration δ of a job is strictly positive and less than its budget c
- unique identifiers
- released exactly once

Three steps to reach correctness

Earliest Deadline First policy

EDF scheduling policy

For any job j and any time instant t, if the job j is running at instant t, then for any other job j' that is ready to run at the same instant, it holds that $d_j \leqslant d_{j'}$.

Applying the policy on a job set (up to a certain time instant t) is defined as :

$$\text{EdfPolicyUpTo } t$$

EDF policy correctnesss property

$$\text{schedulable} \implies \forall j. \forall t. \text{ EdfPolicyUpTo } t \implies \neg\text{overdue } j\ t$$

Correctness of an intermediate election function

Implement an idealised election function that acts like the one that will be executed.

The next step is to prove it implements the EDF policy defined previously.

Functional election function implements EDF policy

$$\forall t, \forall o, \forall s.\ \texttt{idealised\_scheduler}(t) = (o, s) \implies \text{EdfPolicyUpTo}\ t.$$

From this property follows the correctness of this idealised election function.

Correctness of the election function

Implement the final, translatable to C, election function *that relies on a chosen set of primitives*.

The next step is to prove that it acts the same way as the functional one.

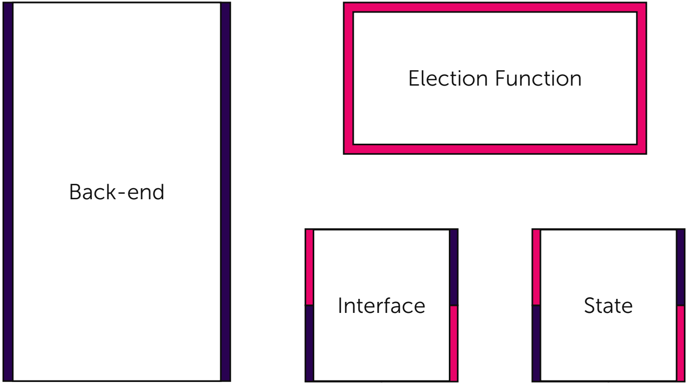
Actual election function has same effects as functional

$$\begin{gathered}\forall t.\\ \{\, \mathit{env} = E \land s = \mathit{init} \,\}\\ (o, s') := \texttt{scheduler}\,(t)\\ \{\, \texttt{idealised\_scheduler}\,(t) = (o, s') \,\}\end{gathered}$$

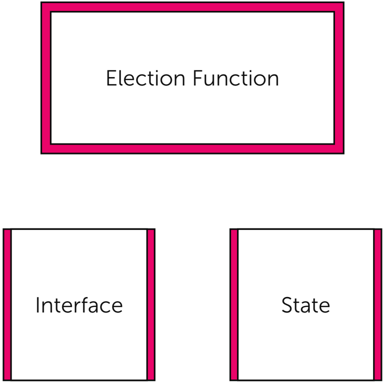
From this property follows the correctness of the scheduler.

Models & Implementation

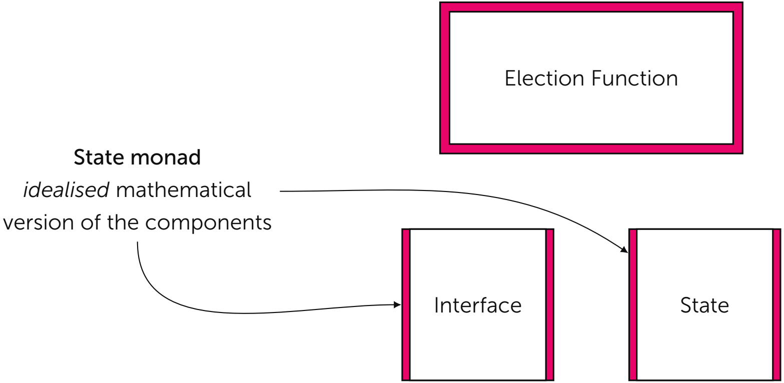
Overview of the scheduler



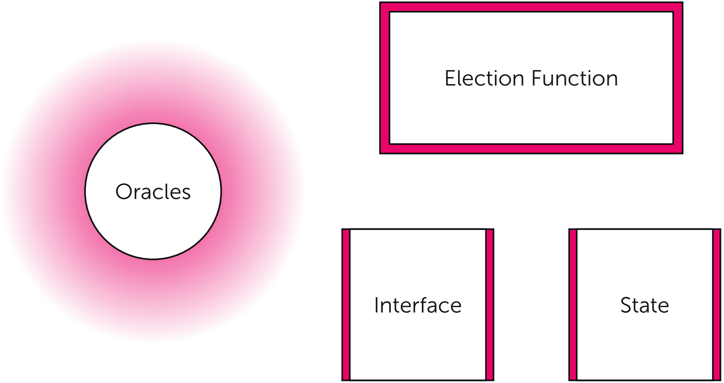
The scheduler from the proof's point of view



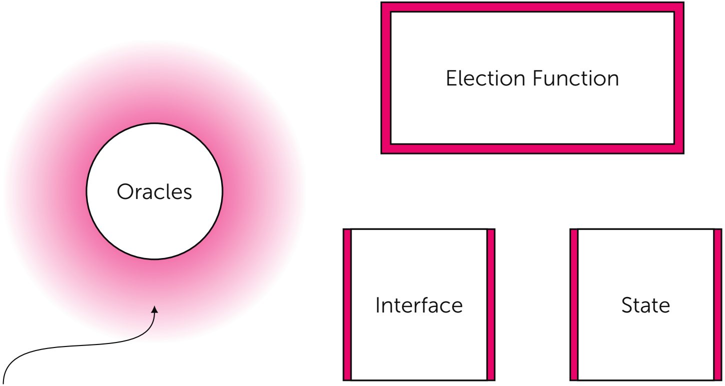
The scheduler from the proof's point of view



The scheduler from the proof's point of view



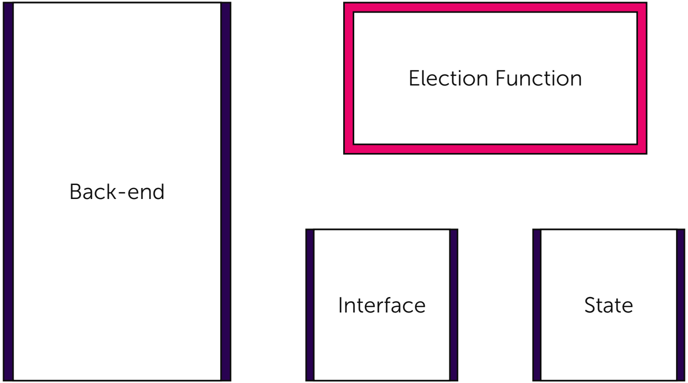
The scheduler from the proof's point of view



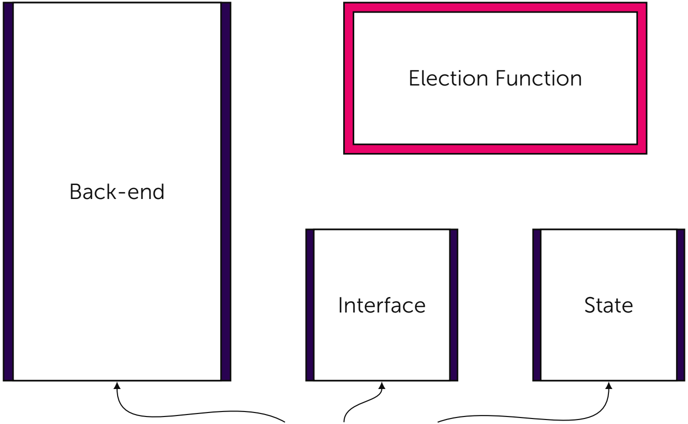
Environment monad

constraints on the behaviour, no direct description

The actual scheduler



The actual scheduler



Actual implementation

Different from the model, but arguably close enough

Most common yet forgotten assumption

The models properly describe the behaviour of components we rely on.

Conclusion

Conclusion

We have shown an EDF scheduler with a proved election function, describing :

- The role of the election function, the interface and state, and the back-end
- The correctness of the election function
- The assumptions
- The monadic approach

Thank you for your attention!

Sources & directions to run the scheduler can be found on our repository :

`https://github.com/2xs/pip_edf_scheduler`

and it passed the artifact validation process