Programming adaptative real-time systems

Thesis Work

Frédéric Fort

Univ. Lille, Inria, CNRS, Centrale Lille,
UMR 9189 CRIStAL, F-59000 Lille, France

Defended October 4, 2022

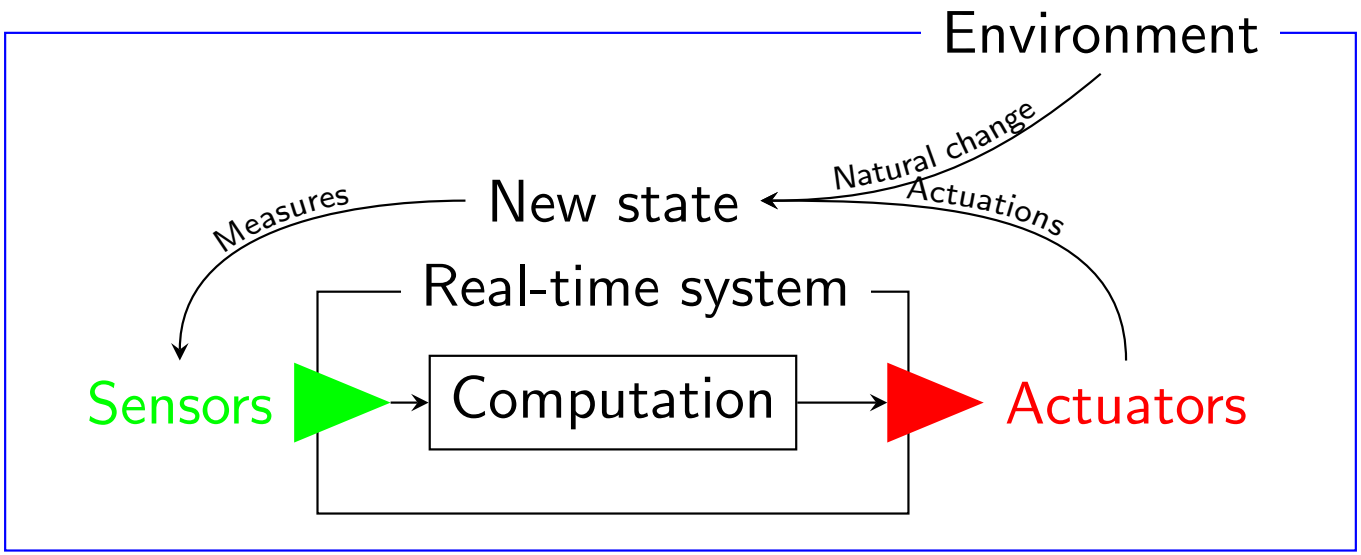
Table of contents

Table of contents

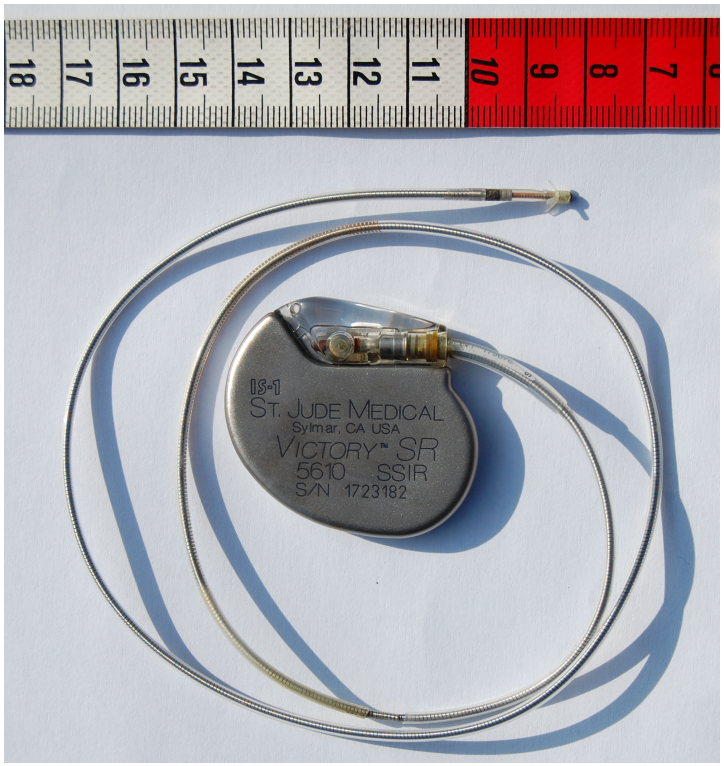
Table of contents

Overview

- Cyber-physical system executing within an *environment*
- Simultaneously *observes* and *influences* environment
- Critical timing constraints



Example — Pacemaker



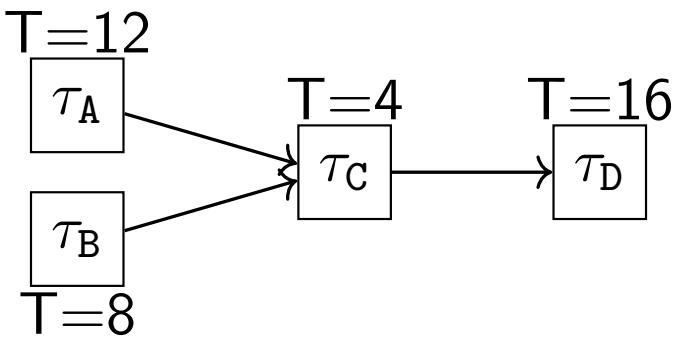
- Maintains heartbeat
  - Typical heartbeat each 0.5s-1s
  - Detects potential failures
  - Electric impulses help natural function
- Central timing constraints

  Delays ; Irregularities
- $\neq$ High-performance computing
  - Speed gains beyond a certain point useless

  $\Rightarrow$ Just respect constraints
  - Prefer predictability gains

Context — Multiple timing constraints

- *Task set* $\tau_i \in \mathcal{T}$
- Job τ_i^n : n-th execution of τ_i
- Period T_i : Time between two 2 activations of consecutive jobs of τ_i



Context — Synchronous reactive model

- Synchronous paradigm standard for critical real-time systems

$\Rightarrow$ Logical time

  - Abstract time as a sequence of clearly separated instant

  $\Rightarrow$ Exact physical date of computations doesn't matter, iff

    - Instants sufficiently frequent
    - Computations never cross instant borders

- *Clock calculus* : Checks temporal consistency of programs

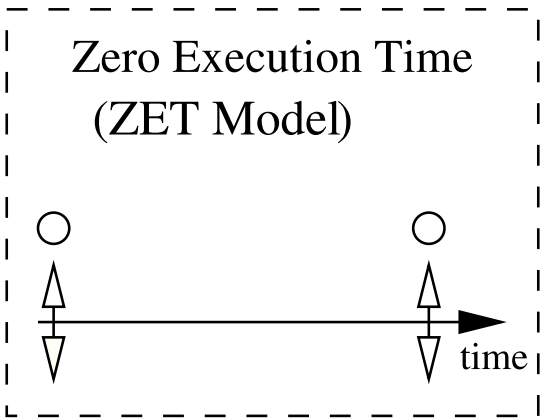
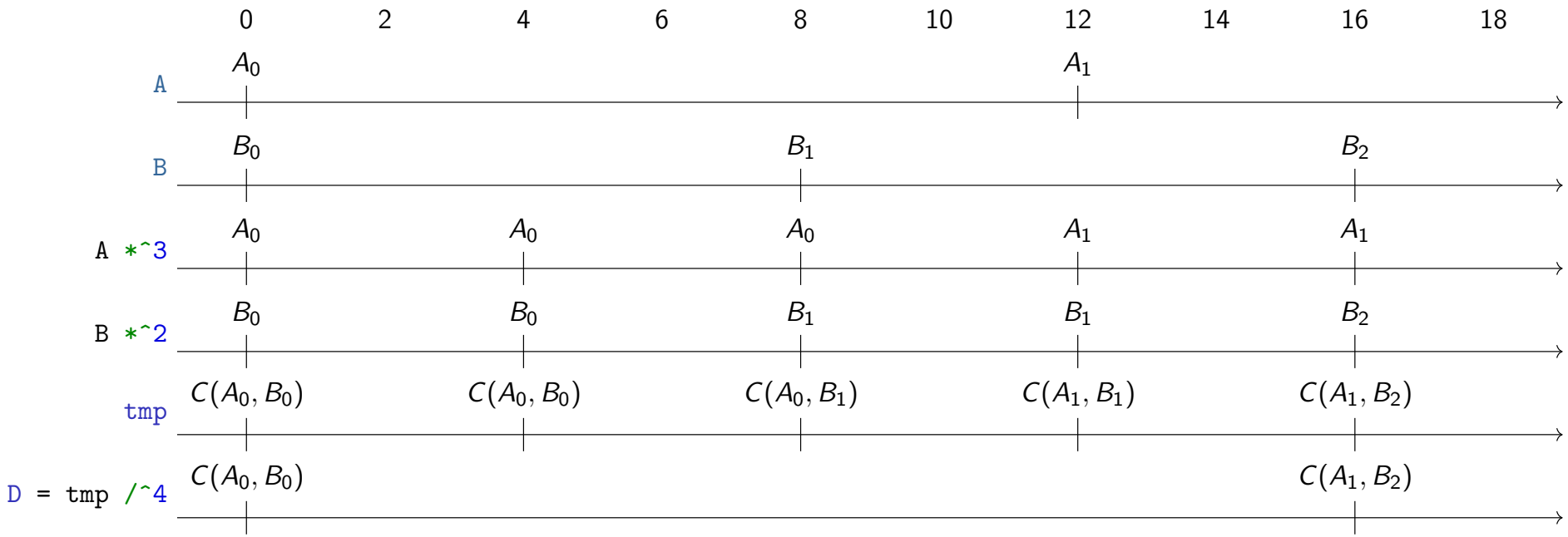


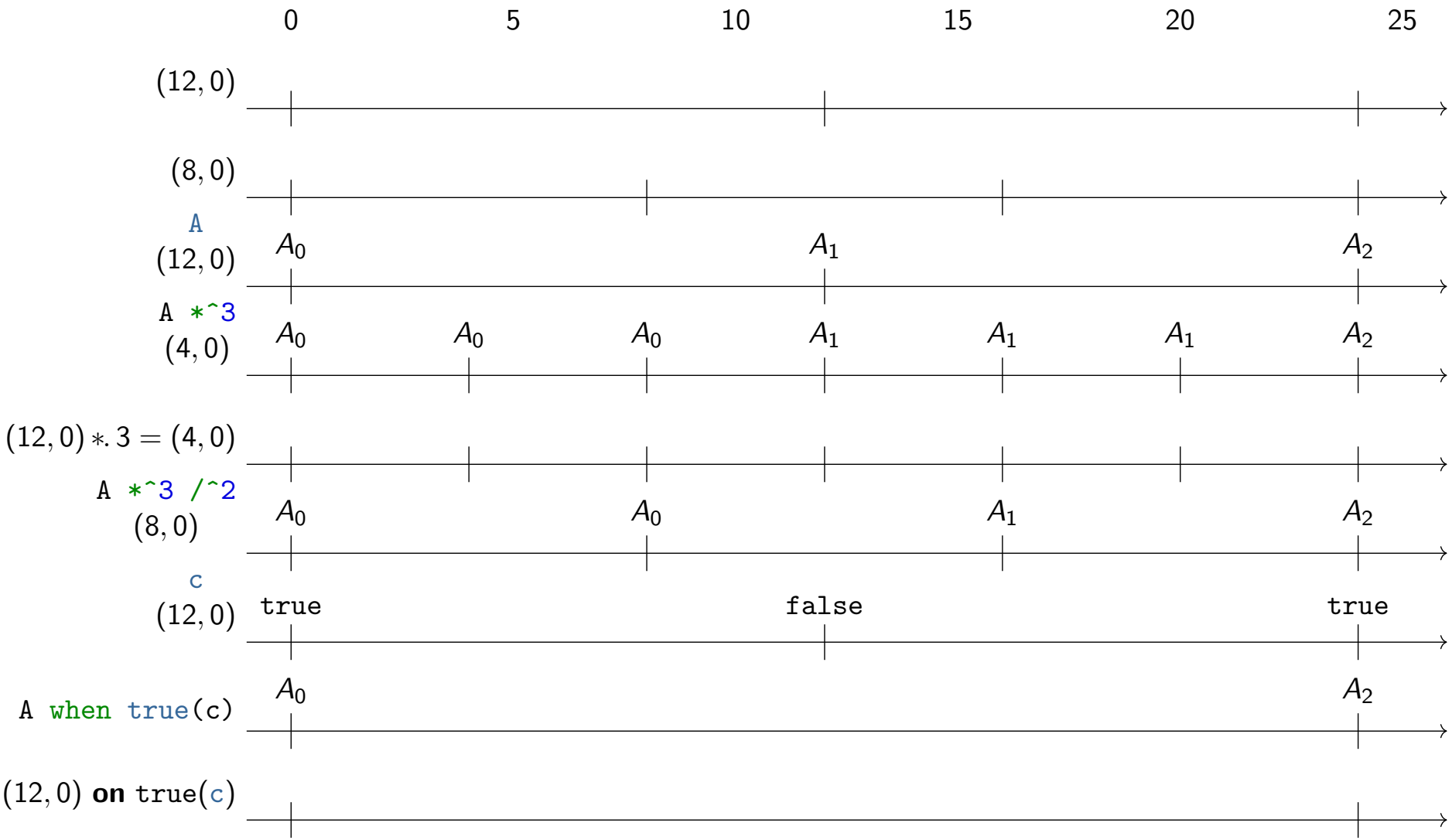
Table of contents

A multi-rate synchronous dataflow language

```
node main (A: rate (12,0); B: rate (8,0))
  returns (D: rate (16,0))
var tmp;
let
  tmp = C(A *^3, B *^2);
  D = tmp /^4;
tel
```



Clocks and flows



Compiling Prelude into task sets

Generate tasks for

- Inputs of “main” node (Sensors)
- Output of “main” node (Actuators)
- Node calls (intermediate tasks)

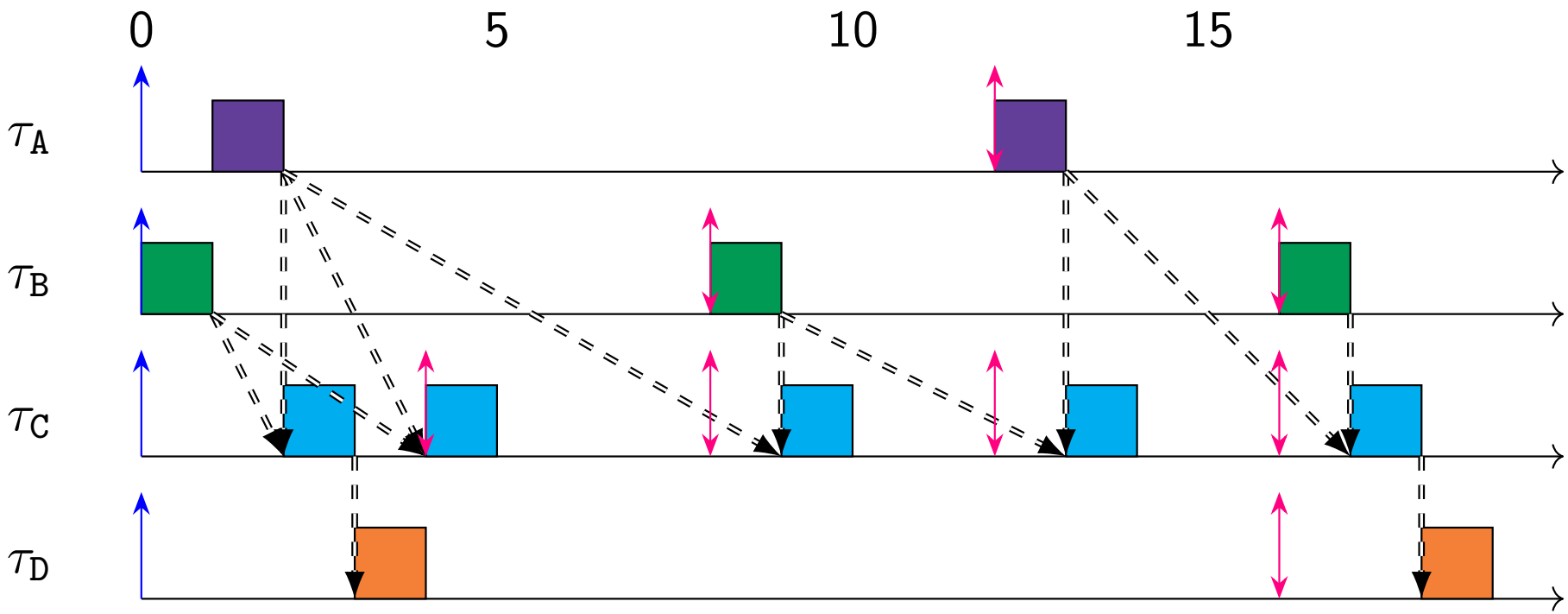
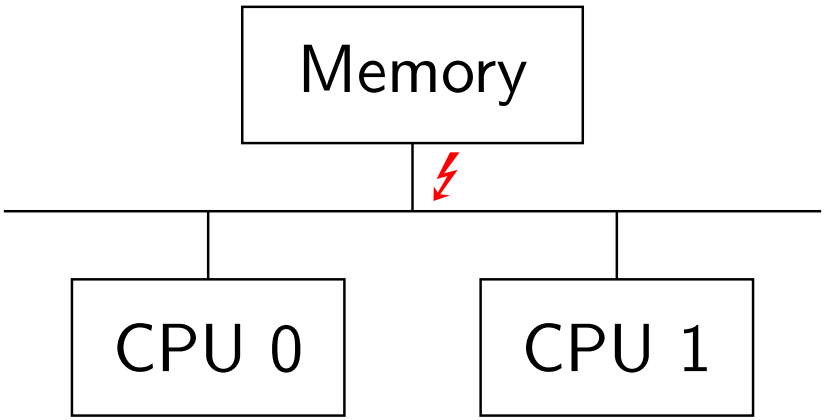


Table of contents

Table of contents

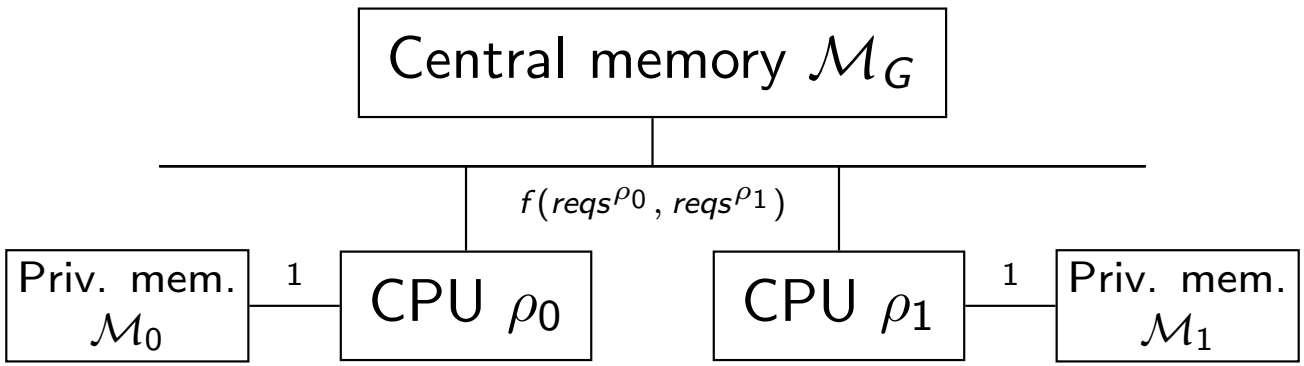
Multicore platforms

- Promise of performance gains « unkept » in RT
  - Optimized for the average case
  - Worst-case performances potentially lower than in monocore
- Problem source : Shared central memory
  - All cores can and must access it
  - Constant competition for access



Multicore hardware model

- Components graph
  - Processor cores ρ_i
  - Private memories $\mathcal{M}_i$ (cache, scratchpad, etc.)
  - Central memory $\mathcal{M}_G$



Multi-phase execution model AER

- Improves predictability

$\Rightarrow$ Explicits the usage of local memory

- Divides job execution in phases
  1. *Acquisition* : Copy inputs to local memory
  2. *Execution* : Compute using only local memory
  3. *Restitution* : Copy results to central memory
- A/R phases : memory access contentions, but little computations
- E phase : Computation rich, but no contentions

Limits of the AER model

- Manual implementation tedious and error-prone
- Illusion of uniform memory accesses
- AER property easily broken by accident
  - e.g. Buffer access in a deeply nested function call

$\Rightarrow$ Need for a higher level of abstraction (synchronous language)

$\Rightarrow$ Shift responsibilities to compiler

```
void do_computation()
{
  //Reads buffer 1
  int x = f()
  //Reads buffer 2
  //Writes buffer 3
  g(x)
}
```

```
void do_computation()
{
  int x = load_buf1()
  int y = load_buf2()
  x = f(x)
  y = g(x, y)
  store_buf3(y)
}
```

Multi-phase and synchronous models compatibilities

- Semantics restrict memory accesses to :
  - Input acquisition
  - Output production

$\Rightarrow$ Memory access semantics $\approx$ A/R phases

Table of contents

Overview

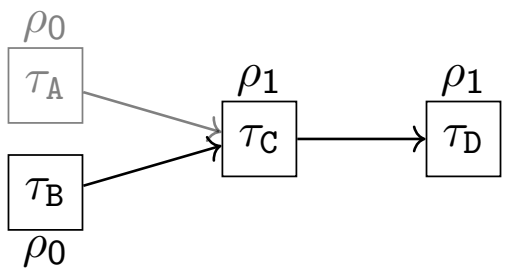
- Improvements of the PRELUDE *back-end*
  1. Define phase set $\mathcal{P}$
  2. Adapt code generation

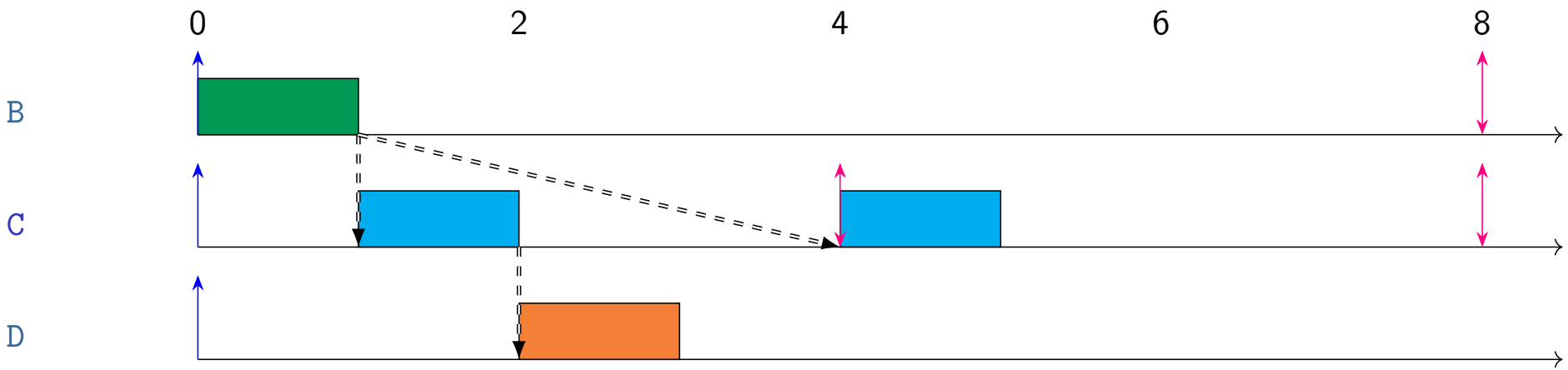
Hypothesis

- Task-to-core partitioning

$\Rightarrow$ τ_i may only execute on core π_i

Phase set $\mathcal{P}$ definition



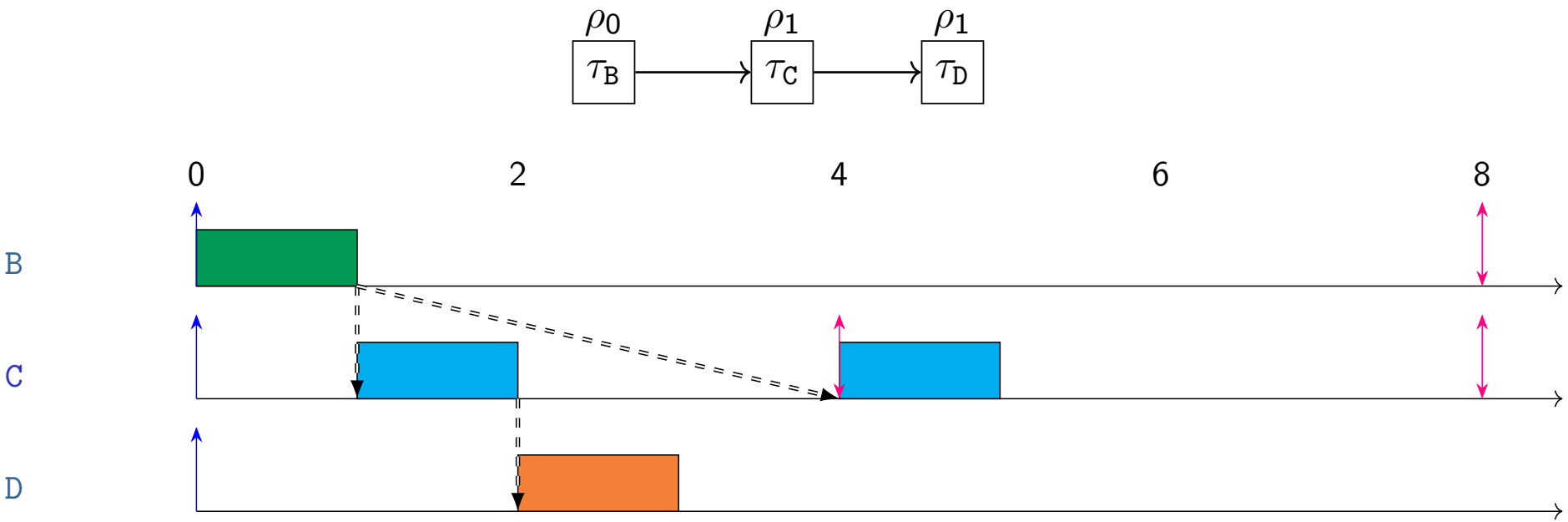


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

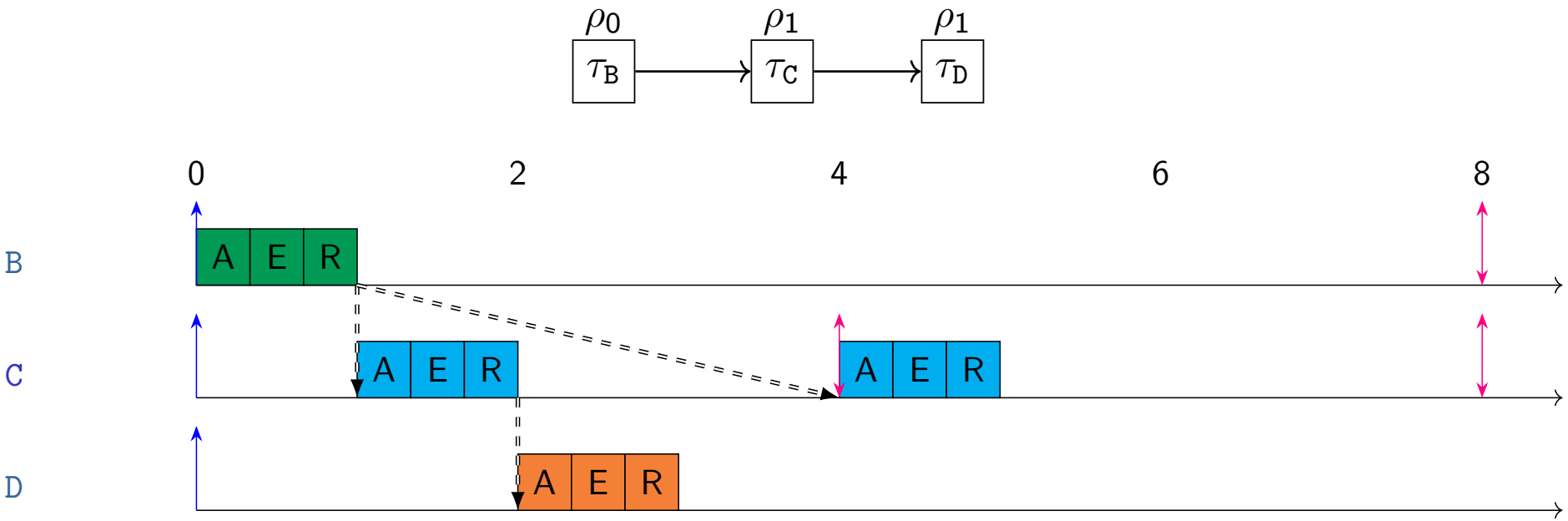


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

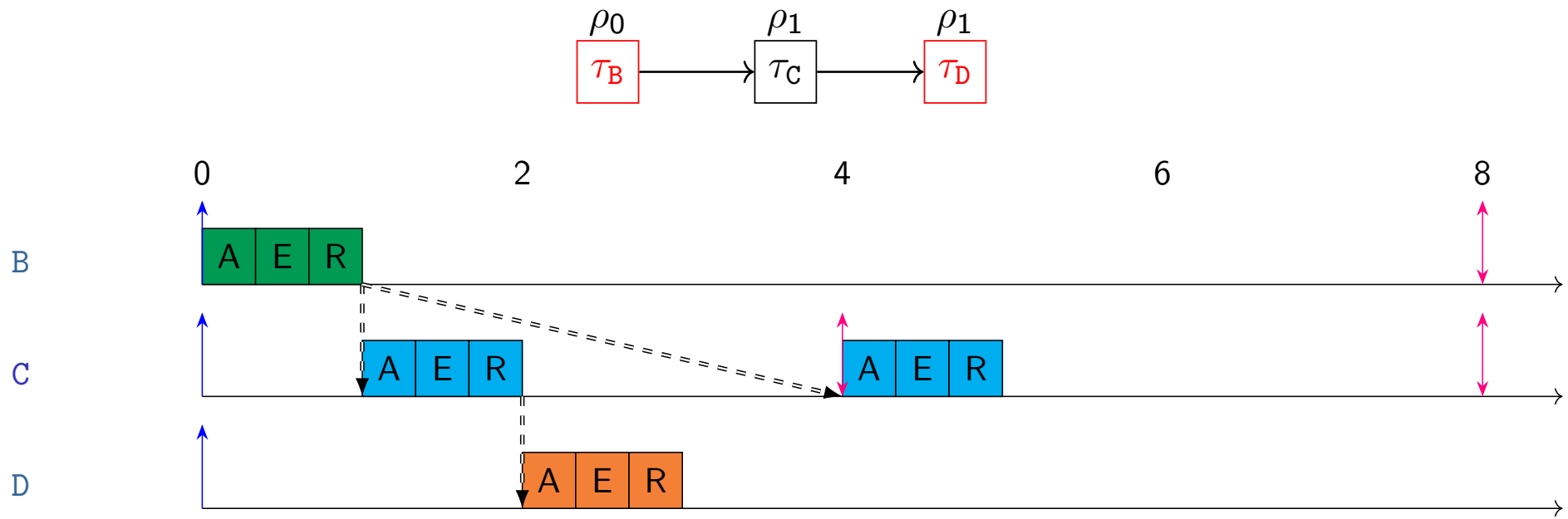


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

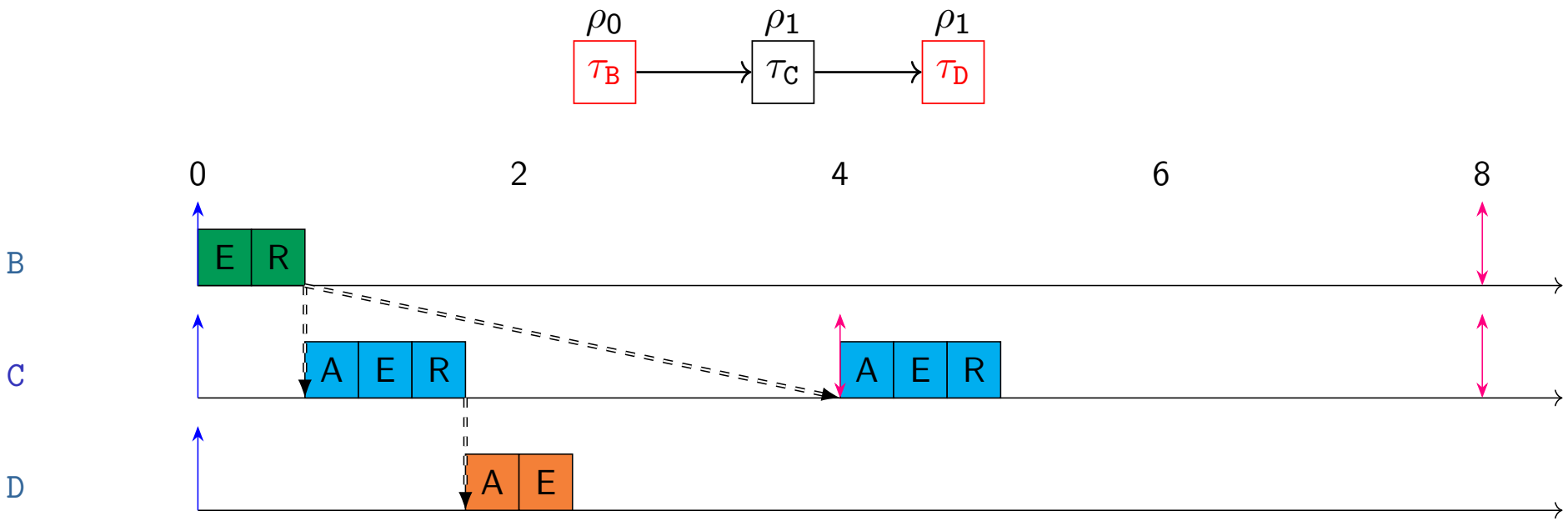


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

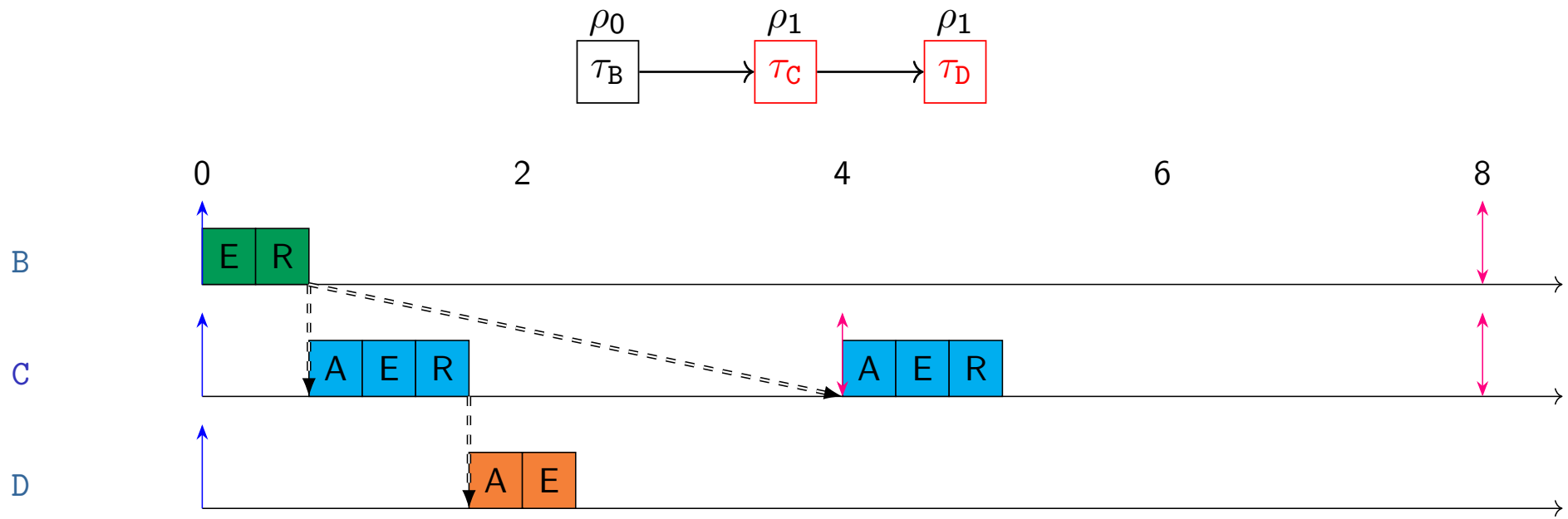


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

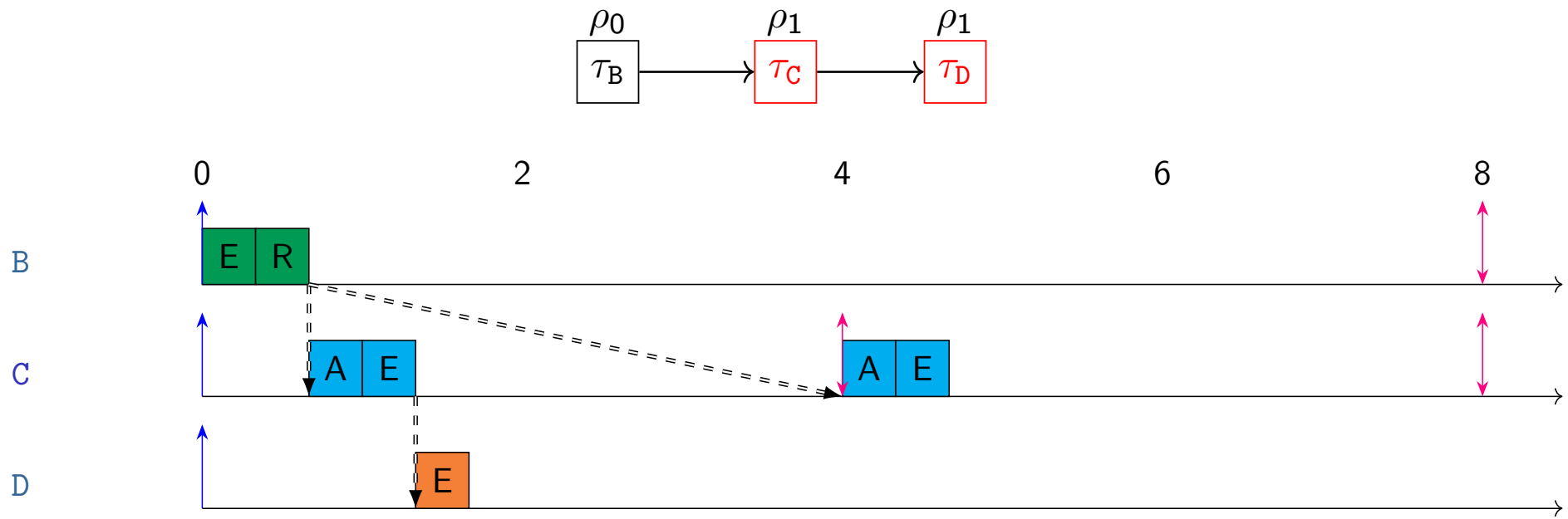


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

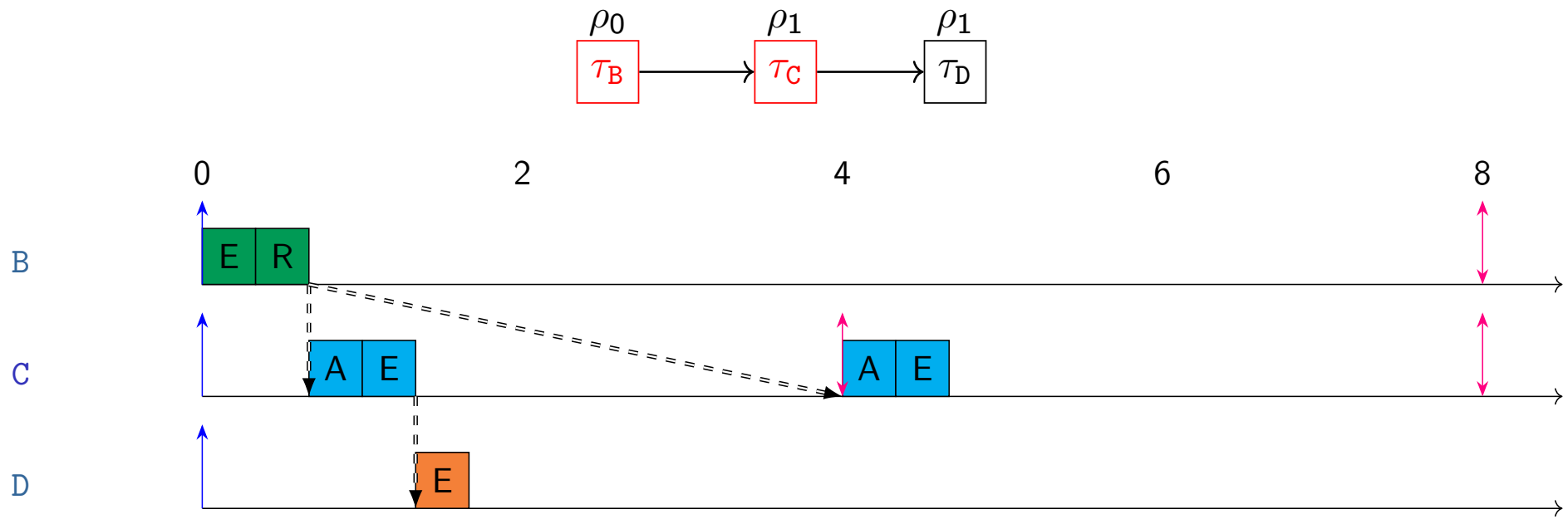


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition

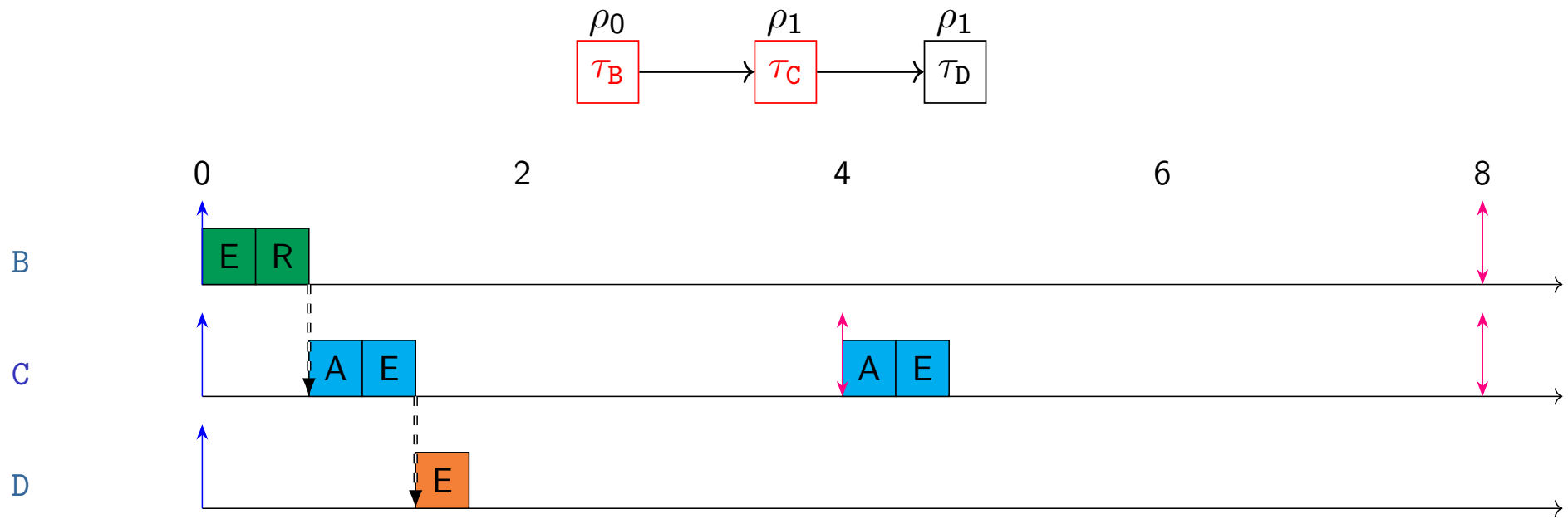


Simplification n°1 : Sensors/Actuators have no A/R phase

Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Phase set $\mathcal{P}$ definition



Simplification n°1 : Sensors/Actuators have no A/R phase

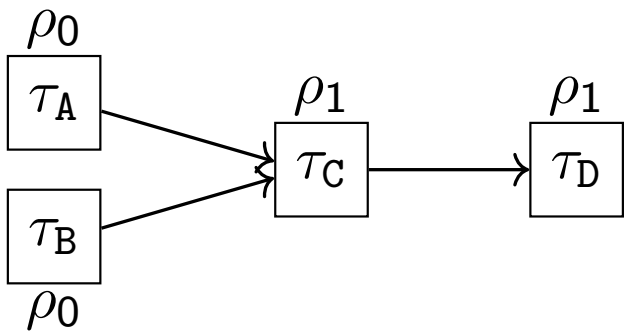
Simplification n°2 : Colocated E phases communicate directly

Simplification n°3 : Remove redundant communications

Code generation

```
1   void C_A()
2   {
3     wait_sem(sem_A_C);
4
5     if (must_wait_B_C())
6       wait_sem(sem_B_C);
7
8     a_loc = read_val(A_C_buff);
9
10    if (must_change_B_C())
11      b_loc = read_val(B_C_buff);
12  }
13
14  void C_E()
15  {
16    c_out = C(a_loc, b_loc);
17
18    C_D_buff = c_out;
19
20    post_sem(sem_C_D);
21  }
```

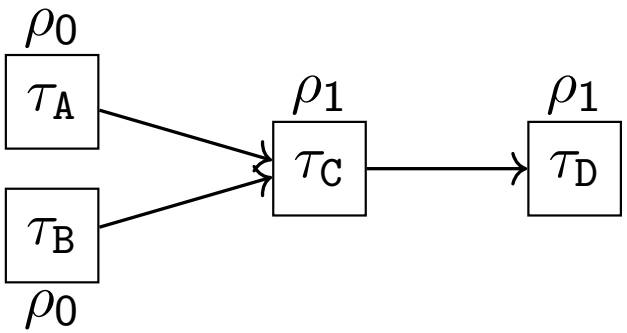
```
1   void B_E()
2   {
3     b_loc = B();
4   }
5
6   void B_R()
7   {
8     if (must_write_B_C())
9       write_val(B_C_buff, b_loc);
10
11    if (must_post_B_C())
12      post_sem(sem_B_C);
13  }
```



Code generation

```
1   void C_A()
2   {
3     wait_sem(sem_A_C);
4
5     if (must_wait_B_C())
6       wait_sem(sem_B_C);
7
8     a_loc = read_val(A_C_buff);
9
10    if (must_change_B_C())
11      b_loc = read_val(B_C_buff);
12  }
13
14  void C_E()
15  {
16    c_out = C(a_loc, b_loc);
17
18    C_D_buff = c_out;
19
20    post_sem(sem_C_D);
21  }
```

```
1   void B_E()
2   {
3     b_loc = B();
4   }
5
6   void B_R()
7   {
8     if (must_write_B_C())
9       write_val(B_C_buff, b_loc);
10
11    if (must_post_B_C())
12      post_sem(sem_B_C);
13  }
```

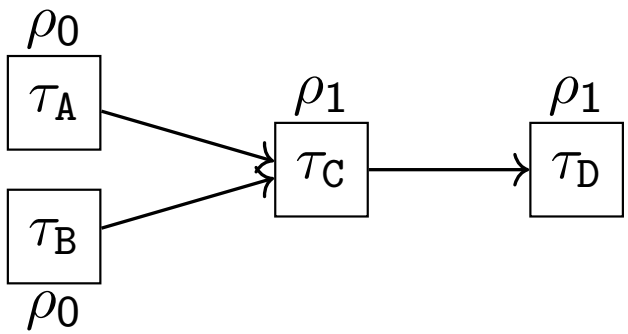


- `{read,write}_val` R/W buffers in $\mathcal{M}_G$
- Direct assignments in $\mathcal{M}_i$

Code generation

```
1   void C_A()
2   {
3     wait_sem(sem_A_C);
4
5     if (must_wait_B_C())
6       wait_sem(sem_B_C);
7
8     a_loc = read_val(A_C_buff);
9
10    if (must_change_B_C())
11      b_loc = read_val(B_C_buff);
12  }
13
14  void C_E()
15  {
16    c_out = C(a_loc, b_loc);
17
18    C_D_buff = c_out;
19
20    post_sem(sem_C_D);
21  }
```

```
1   void B_E()
2   {
3     b_loc = B();
4   }
5
6   void B_R()
7   {
8     if (must_write_B_C())
9       write_val(B_C_buff, b_loc);
10
11    if (must_post_B_C())
12      post_sem(sem_B_C);
13  }
```

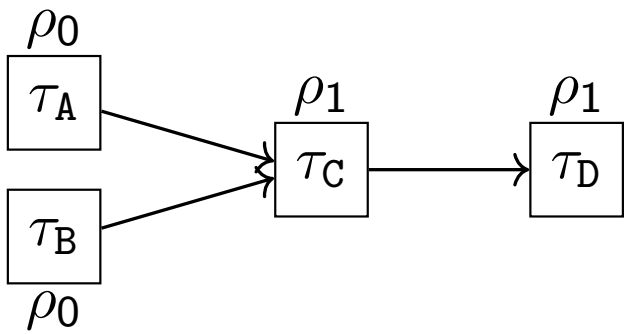


- `sem_X_Y` Binary semaphores between `X` and `Y`

Code generation

```
1   void C_A()
2   {
3     wait_sem(sem_A_C);
4
5     if (must_wait_B_C())
6       wait_sem(sem_B_C);
7
8     a_loc = read_val(A_C_buff);
9
10    if (must_change_B_C())
11      b_loc = read_val(B_C_buff);
12  }
13
14  void C_E()
15  {
16    c_out = C(a_loc, b_loc);
17
18    C_D_buff = c_out;
19
20    post_sem(sem_C_D);
21  }
```

```
1   void B_E()
2   {
3     b_loc = B();
4   }
5
6   void B_R()
7   {
8     if (must_write_B_C())
9       write_val(B_C_buff, b_loc);
10
11    if (must_post_B_C())
12      post_sem(sem_B_C);
13  }
```

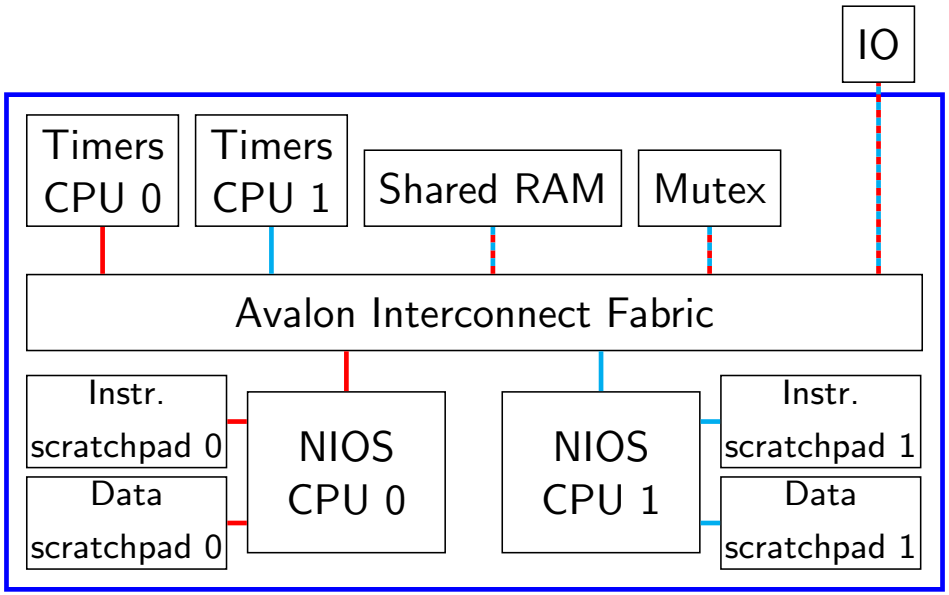


- `must_*_X_Y` Multi-periodic protocol between `X` and `Y`
- Only generated, if necessary

Table of contents

Evaluation

- In the context of our RTCSA'19 [1] paper
- ROSACE case study (longitudinal flight controller)
- Implement 2 multicore architectures on FPGA
  - ⇒ Only difference : local memories (cache vs scratchpad)
- Measure AER code speedup vs. non-AER code
- Variation in RAM speed



Evaluation

- In the context of our RTCSA'19 [1] paper
- ROSACE case study (longitudinal flight controller)
- Implement 2 multicore architectures on FPGA
  - ⇒ Only difference : local memories (cache vs scratchpad)
- Measure AER code speedup vs. non-AER code
- Variation in RAM speed

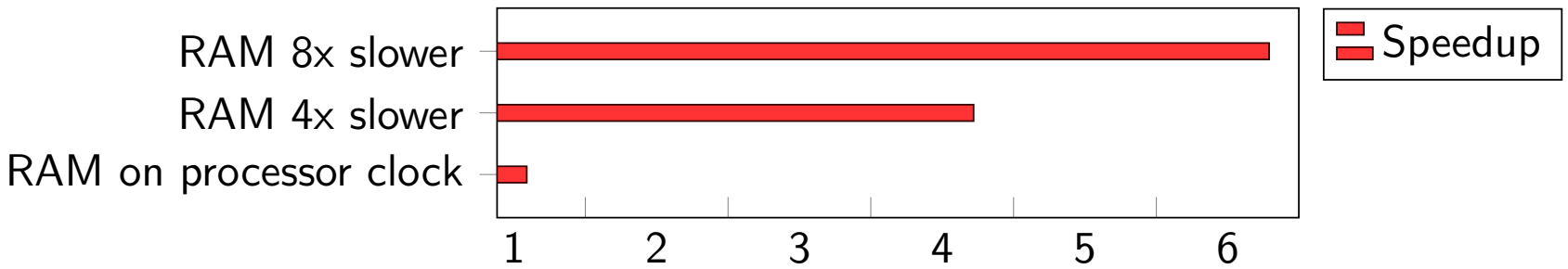


Table of contents

Table of contents

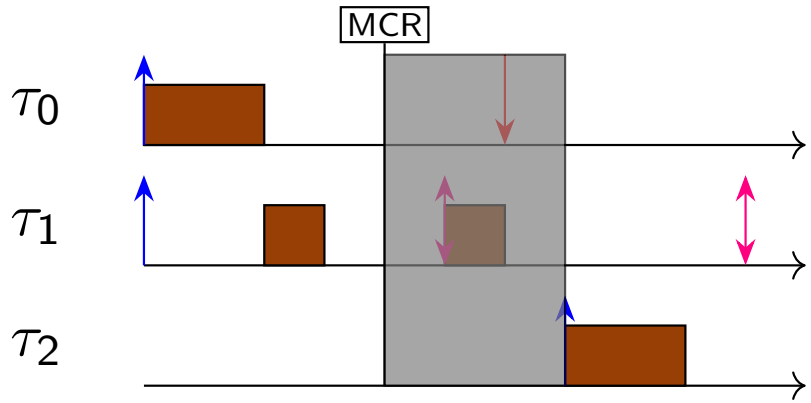
Multi-mode systems

- Functional requirements of RT evolve during execution
- Handling all requirements at all time superfluous

$\Rightarrow$ How to change from one « mode » to another ?

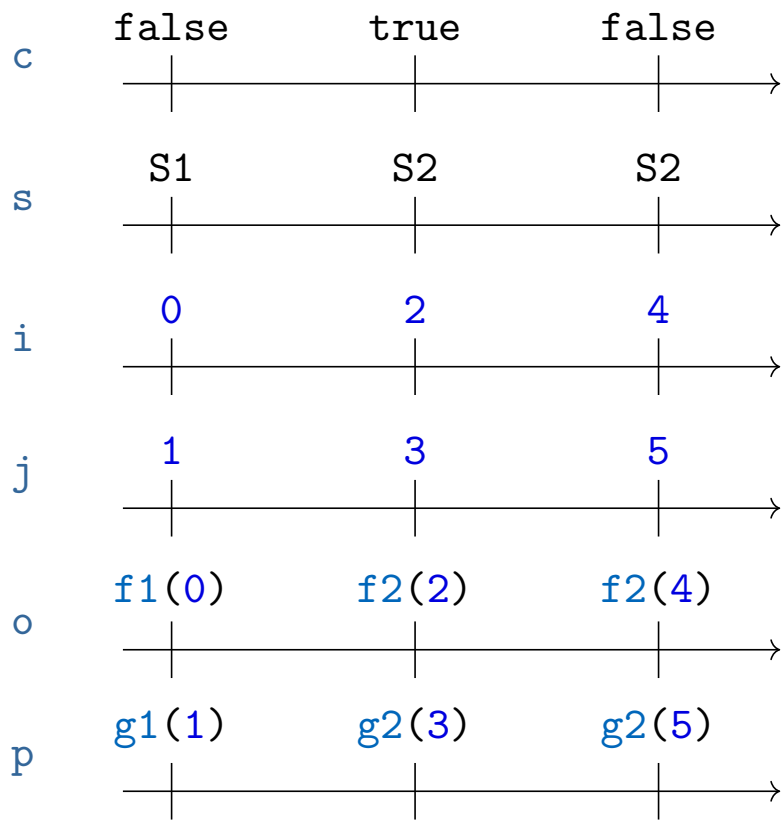
Context — Problems linked to the transition phase

- *Overload* : System unable to execute two modes simultaneously
- *Consistency* : Define system semantics between mode executions



Context — Synchronous State Machines (Lustre)

```
node main (i,j,c)
  returns (o,p)
let
  automaton
  | S1 ->
    unless c then S2;
    o = f1(i);
    p = g1(j);
  | S2 ->
    unless c then S1;
    o = f2(i);
    p = g2(j);
  end
tel
```




```
s = ...;

o_S1 = f1(i when S1(s));
o_S2 = f2(i when S2(s));
```

Limits of existing conditional clocks

e when C(s)

The dataflow of values of e for the instants when the state s is C

- Which semantics for when e and s are not synchronous ?
- How to combine rate-transition operators and **when** ?
- What is the clock of such a flow ?

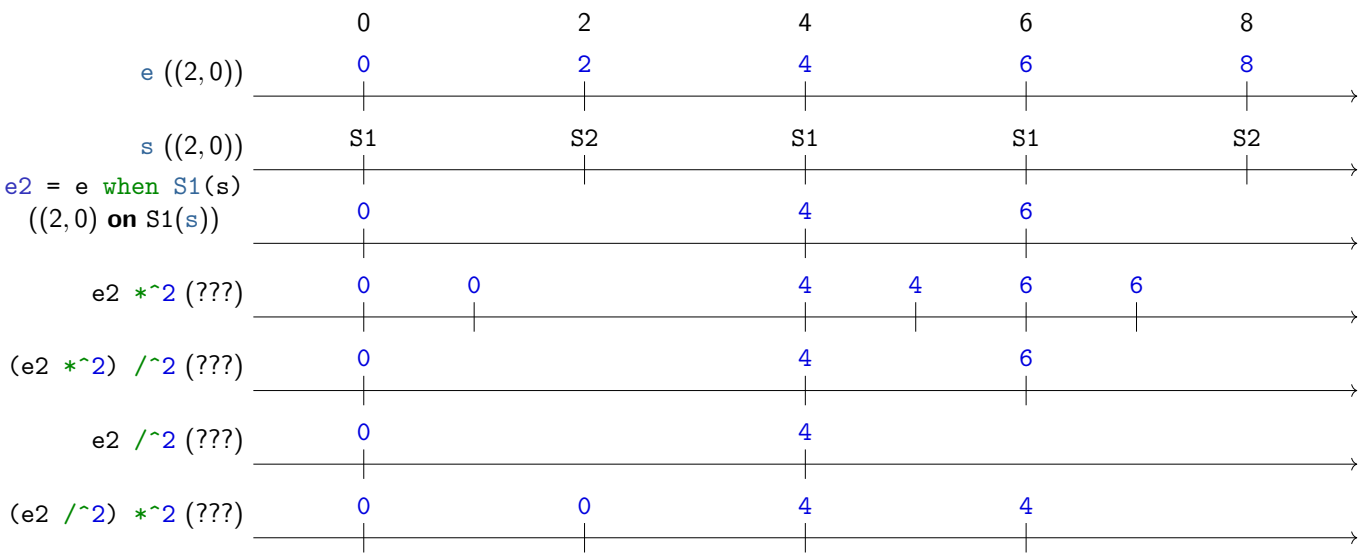
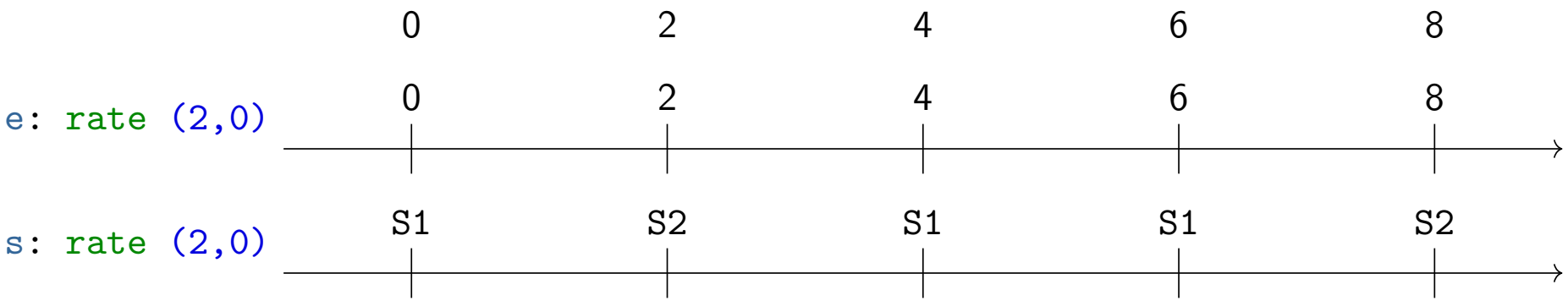
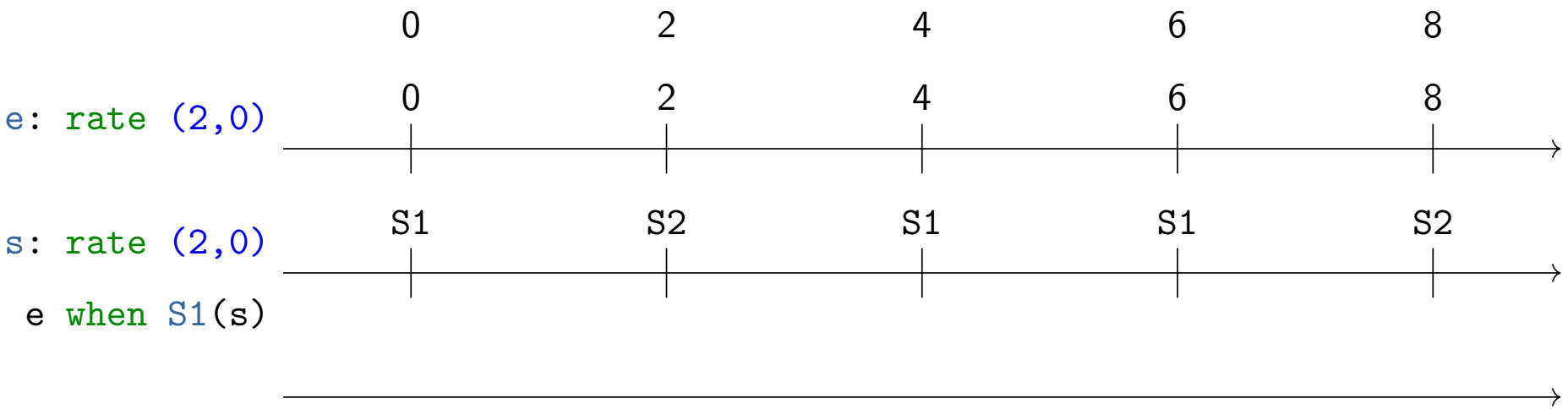


Table of contents

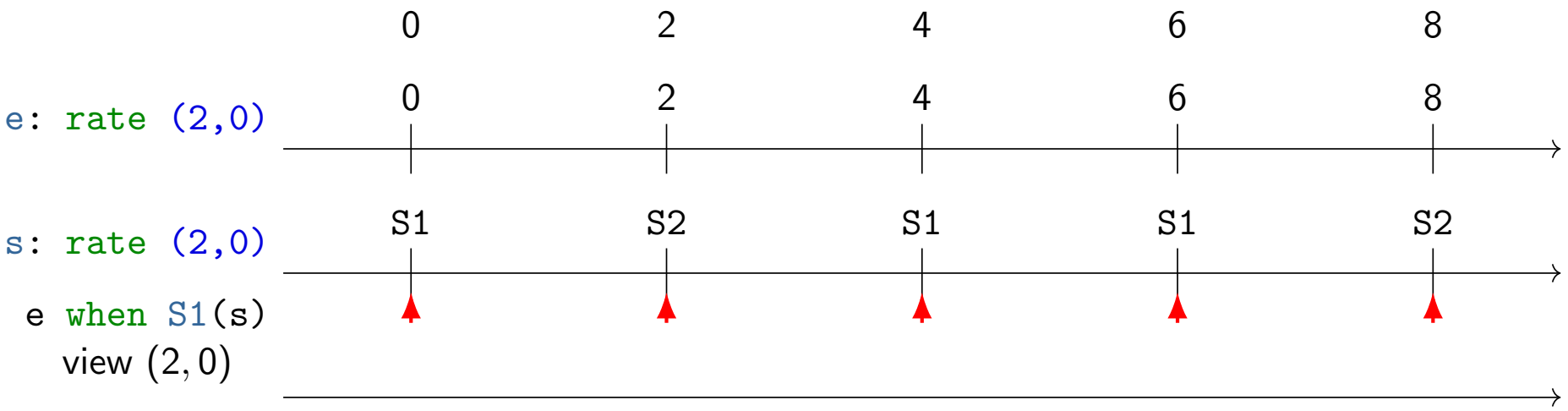
Clock Views — Intuition



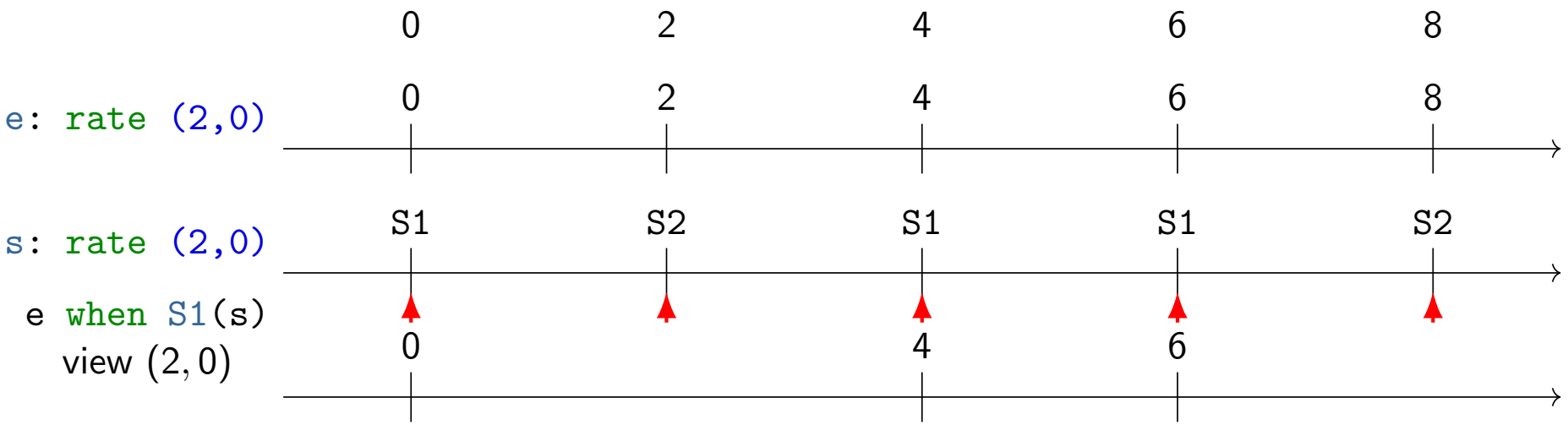
Clock Views — Intuition



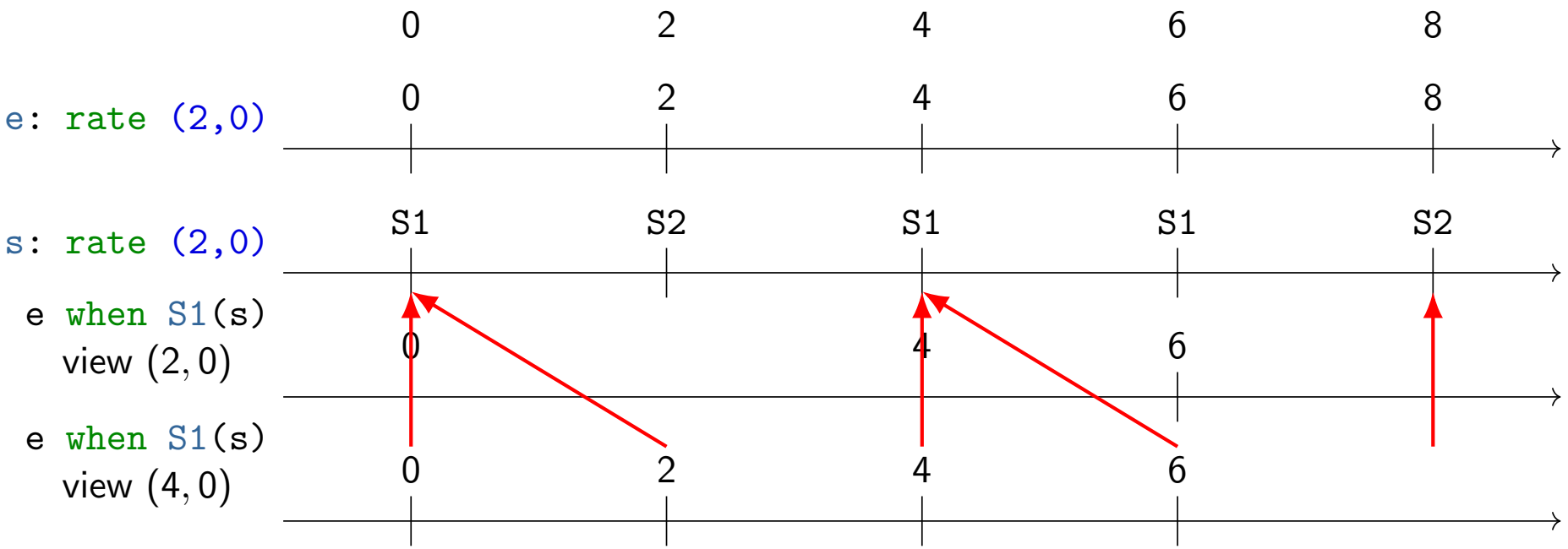
Clock Views — Intuition



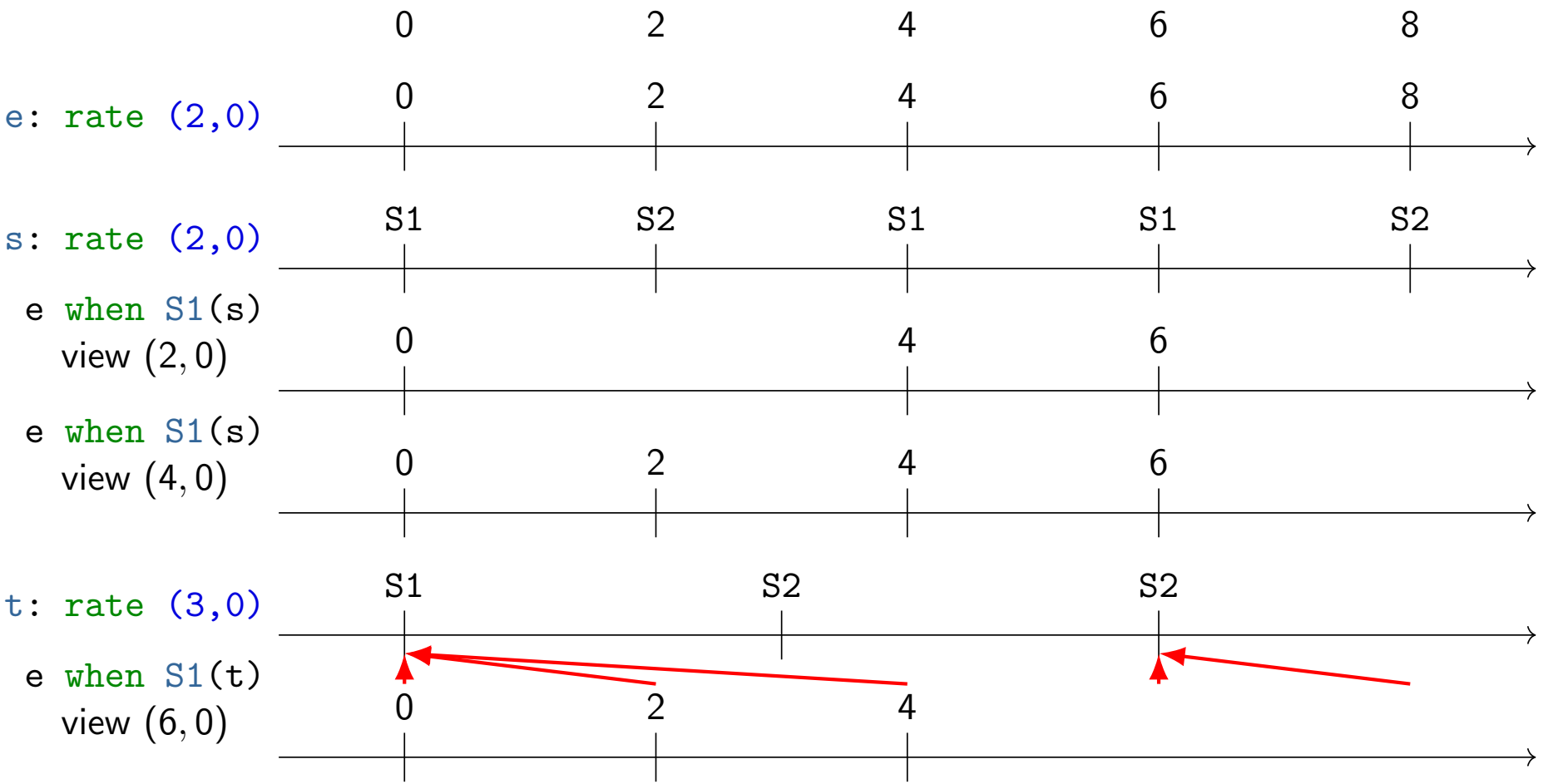
Clock Views — Intuition



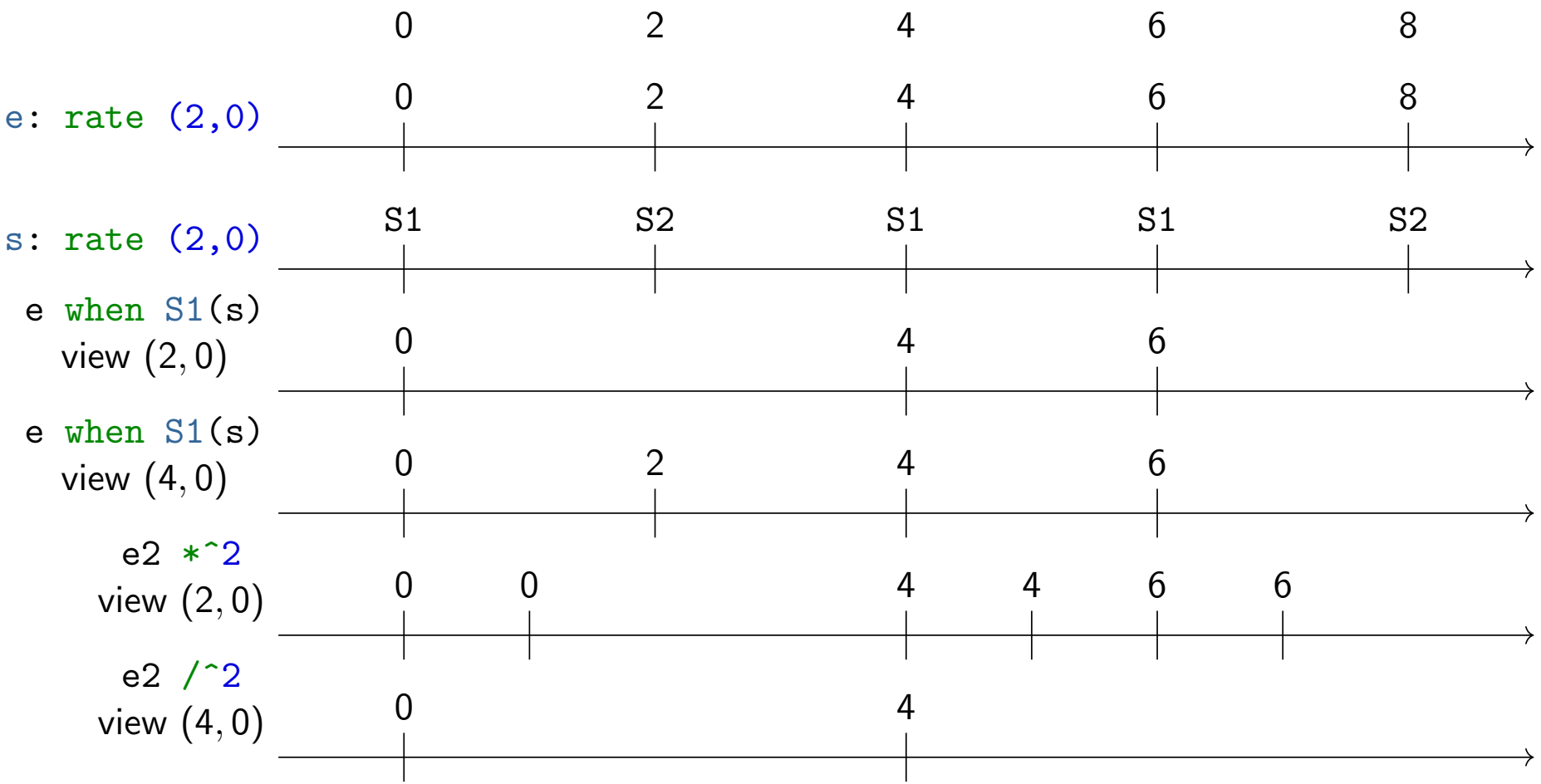
Clock Views — Intuition



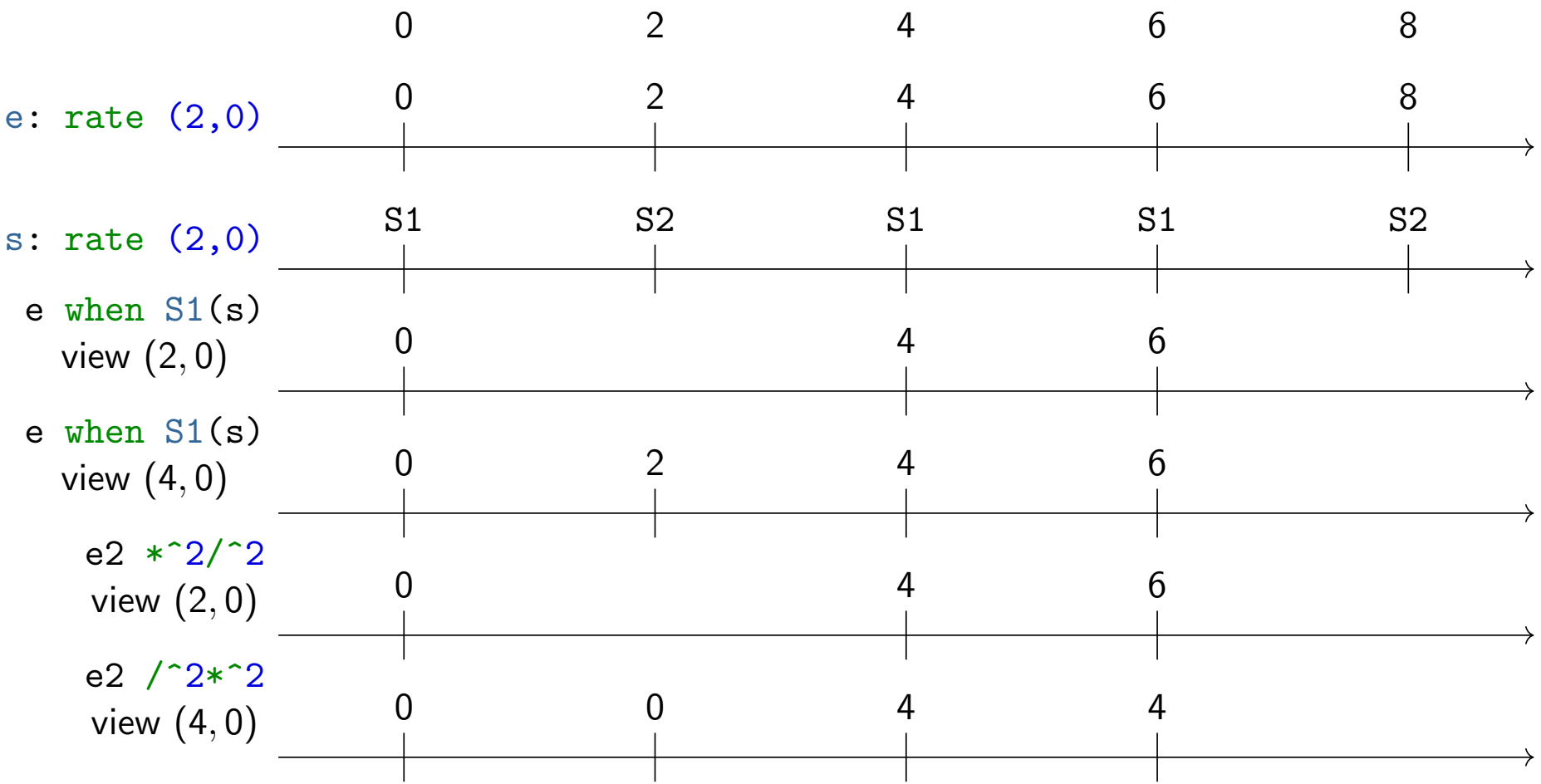
Clock Views — Intuition



Clock Views — Intuition

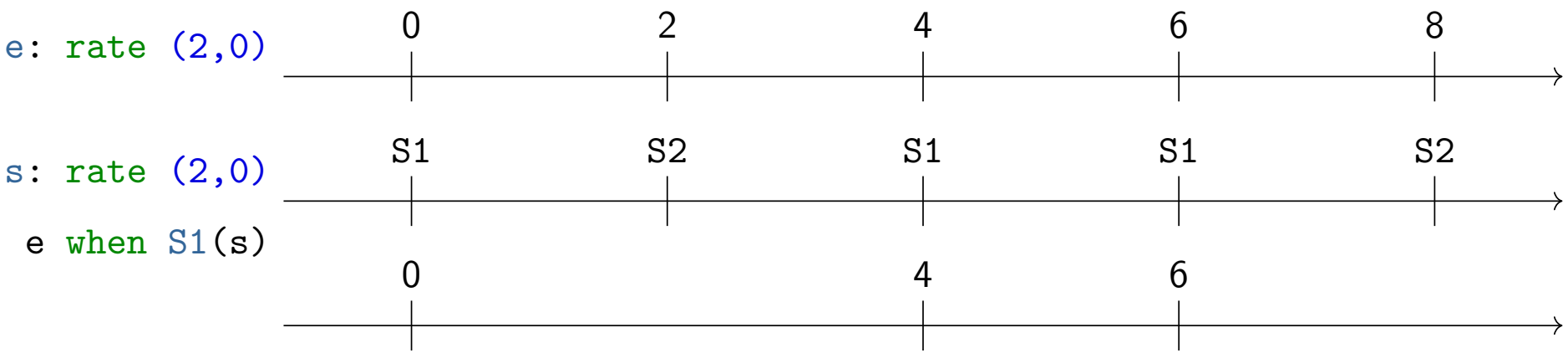


Clock Views — Intuition



Clock Views — Formal definition

$$\texttt{when}^{\#}\,(e, s, C) = \{(v, t) \mid (v, t) \in e, t \in (\widehat{e}\ \textbf{on}\ C(s))^{\#}\}$$



Clock Views — Formal definition

$$\texttt{when}^{\#}(e, s, C) = \{(v, t) \mid (v, t) \in e, t \in (\widehat{e}\ \textbf{on}\ C(s))^{\#}\}$$

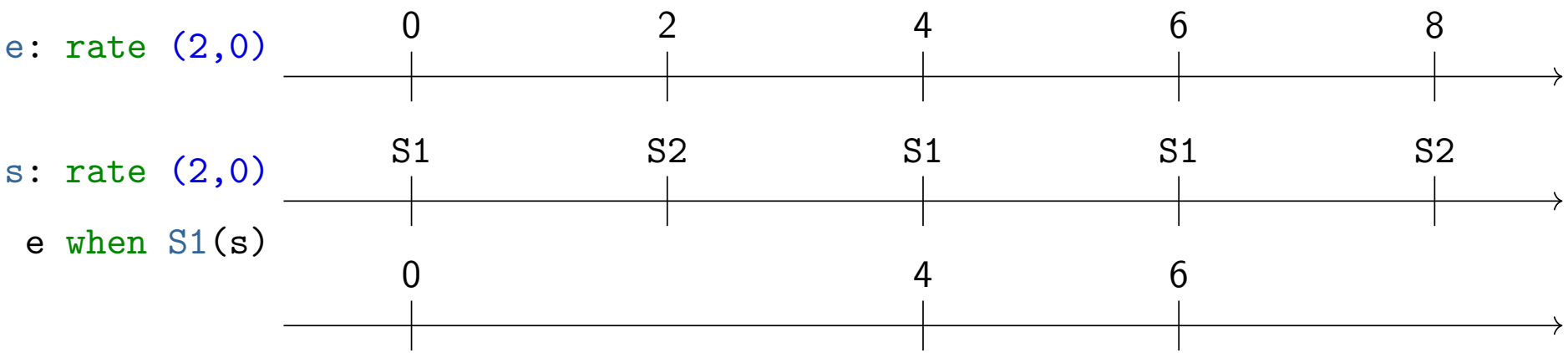
$$(ck\ \textbf{on}\ C(s))^{\#} = \{t \mid t \in ck^{\#}, (v, t'') \in s^{\#}, v = C \wedge t'' = t\}$$

$v = C$

- Filters according the values of s

$t = t''$

- Assigns each date of ck a value of s ($\Rightarrow$ mono-periodicity)



Clock Views — Formal definition

$$\texttt{when}^{\#}(e, s, C) = \{(v, t) \mid (v, t) \in e, t \in (\widehat{e}\ \textbf{on}\ C(s))^{\#}\}$$

$$(ck\ \textbf{on}\ C(s, (n, p)))^{\#} = \{t \mid t \in ck^{\#}, t' \in (n, p)^{\#}, (v, t'') \in s^{\#}, v = C \wedge t' \leq t < t' + n \wedge t'' = t'\}$$

$v = C$

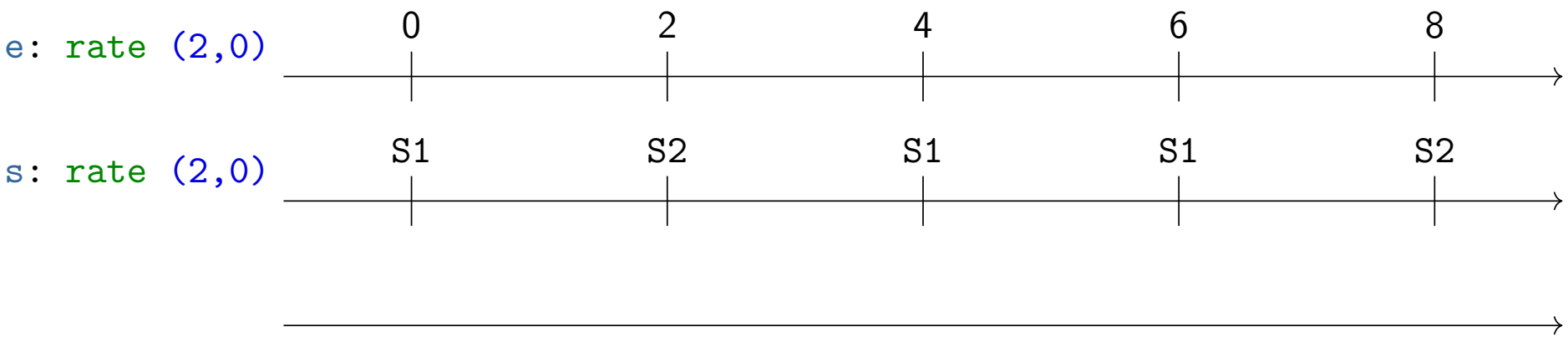
- Filters according to the values of s

$t' \leq t < t' + n$

- Assigns each date of ck an interval defined by the view

$t'' = t'$

- Chooses the single value of s observed within a view interval



Clock Views — Formal definition

$$\texttt{when}^{\#}(e, s, C) = \{(v, t) \mid (v, t) \in e, t \in (\hat{e}\ \textbf{on}\ C(s))^{\#}\}$$

$$(ck\ \textbf{on}\ C(s, (n, p)))^{\#} = \{t \mid t \in ck^{\#}, t' \in (n, p)^{\#}, (v, t'') \in s^{\#}, v = C \wedge t' \leq t < t' + n \wedge t'' = t'\}$$

$v = C$

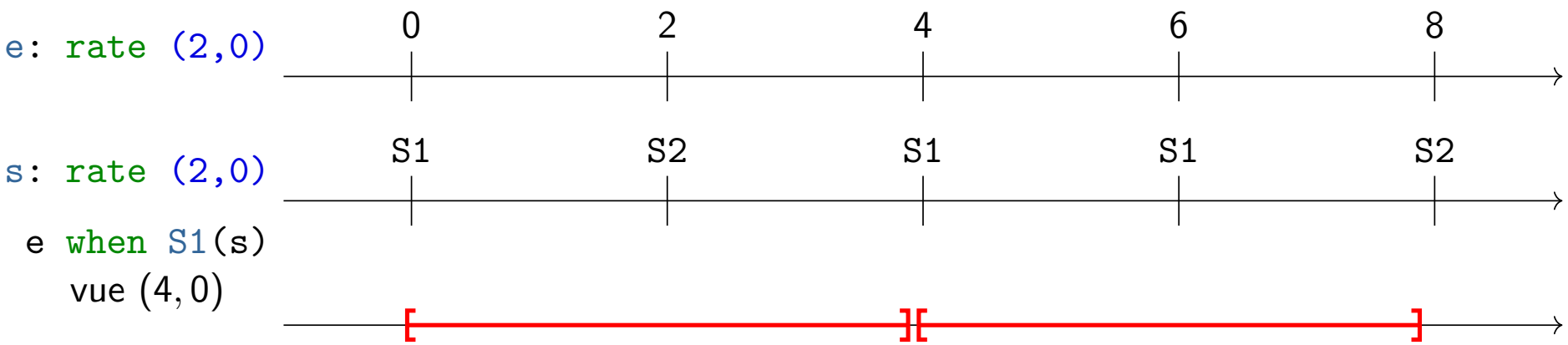
- Filters according to the values of s

$t' \leq t < t' + n$

- Assigns each date of ck an interval defined by the view

$t'' = t'$

- Chooses the single value of s observed within a view interval



Clock Views — Formal definition

$$\texttt{when}^{\#}(e, s, C) = \{(v, t) \mid (v, t) \in e, t \in (\hat{e}\ \textbf{on}\ C(s))^{\#}\}$$

$$(ck\ \textbf{on}\ C(s, (n, p)))^{\#} = \{t \mid t \in ck^{\#}, t' \in (n, p)^{\#}, (v, t'') \in s^{\#}, v = C \wedge t' \leq t < t' + n \wedge t'' = t'\}$$

$v = C$

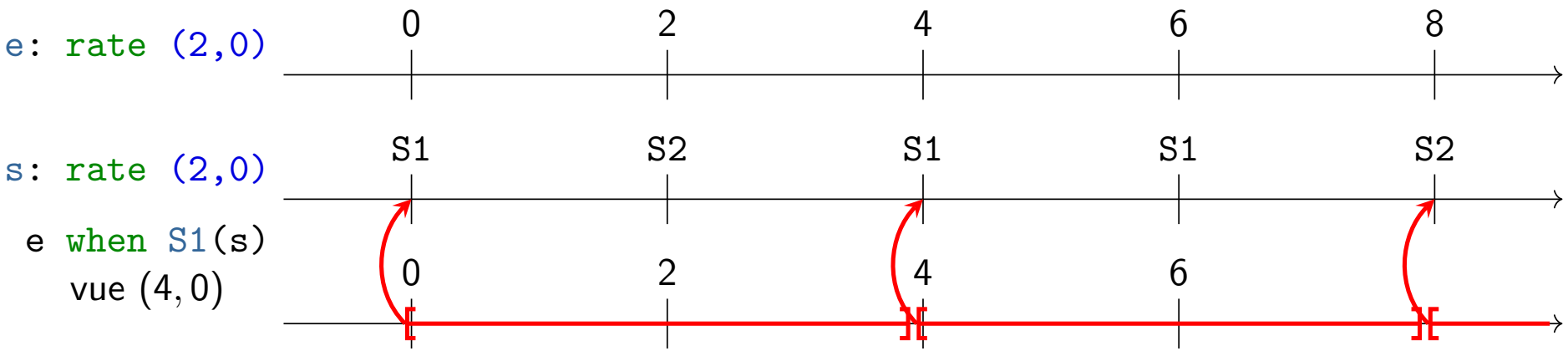
- Filters according to the values of s

$t' \leq t < t' + n$

- Assigns each date of ck an interval defined by the view

$t'' = t'$

- Chooses the single value of s observed within a view interval



Clock Views — Formal definition

$$\texttt{when}^{\#}(e, s, C, (n, p)) = \{(v, t) \mid (v, t) \in e, t \in (\hat{e}\ \textbf{on}\ C(s, (n, p)))^{\#}\}$$

$$(ck\ \textbf{on}\ C(s, (n, p)))^{\#} = \{t \mid t \in ck^{\#}, t' \in (n, p)^{\#}, (v, t'') \in s^{\#}, v = C \wedge t' \leq t < t' + n \wedge t'' = t'\}$$

$v = C$

- Filters according to the values of s

$t' \leq t < t' + n$

- Assigns each date of ck an interval defined by the view

$t'' = t'$

- Chooses the single value of s observed within a view interval

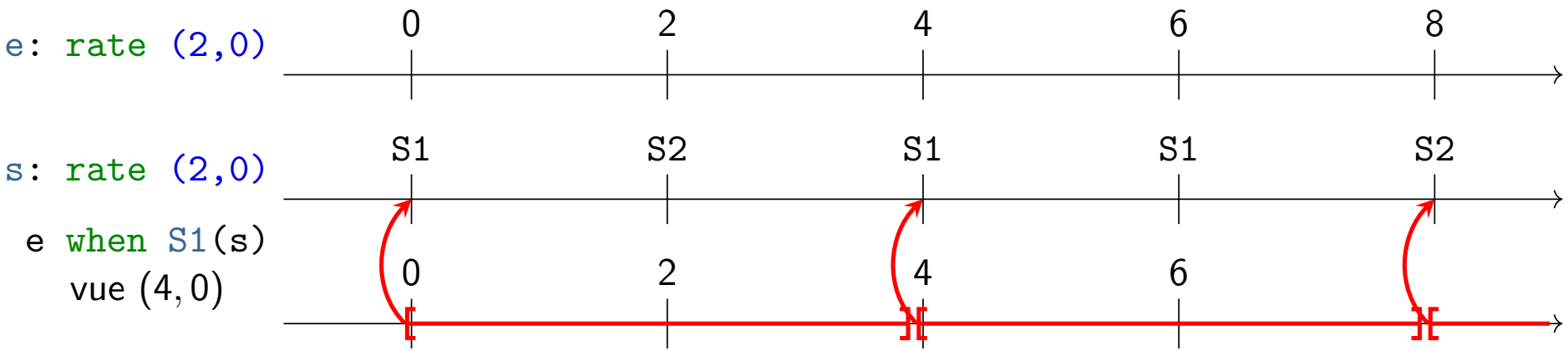


Table of contents

Clock Calculus as a type system

- Check dataflow and clock consistency

$\Rightarrow$ Dedicated type system

$\Rightarrow$ Assigns each dataflow a clock

- Previously based on HM extended with subtyping
- View computation requires a more expressive type system

$\Rightarrow$ Clock calculus based on refinement typing

Refinement typing

- Extends an existing typing system
- *Refines* types with predicates (in a decidable logic)
- SMT solver verifies predicates

$\{\nu{:}b \mid r\}$

The type of all values ν of *base type* b (p.ex. `int`, `int list`) such that the boolean predicate r, called refinement, (e.g. $\nu \geq 0 \wedge \nu < x$) is true for all values inhabiting the refinement type $\{\nu{:}b \mid r\}$.

```
4                    int
()          int -> int -> int
```

Refinement typing

- Extends an existing typing system
- *Refines* types with predicates (in a decidable logic)
- SMT solver verifies predicates

$\{\nu{:}b \mid r\}$

The type of all values ν of *base type* b (p.ex. `int`, `int list`) such that the boolean predicate r, called refinement, (e.g. $\nu \geq 0 \wedge \nu < x$) is true for all values inhabiting the refinement type $\{\nu{:}b \mid r\}$.

`4`	$\{\nu{:}\texttt{int} \mid \nu = 4\}$
`()`	`int -> int -> int`

Refinement typing

- Extends an existing typing system
- *Refines* types with predicates (in a decidable logic)
- SMT solver verifies predicates

$\{\nu{:}b \mid r\}$

The type of all values ν of *base type* b (p.ex. `int`, `int list`) such that the boolean predicate r, called refinement, (e.g. $\nu \geq 0 \wedge \nu < x$) is true for all values inhabiting the refinement type $\{\nu{:}b \mid r\}$.

$$\begin{array}{cc} 4 & \{\nu{:}\texttt{int} \mid \nu = 4\} \\ (/) & x{:}\texttt{int} \to y{:}\{\nu{:}\texttt{int} \mid \nu \neq 0\} \to \{\nu{:}\texttt{int} \mid \nu = x/y\} \end{array}$$

Refinement clock calculus

Clock	Clock type
$(12, 0)$	$\{\nu{:}\mathtt{pck} \mid \pi(\nu) = 12 \wedge \varphi(\nu) = 0\}$
$(8, 0)$	$\langle \mathtt{pck} \mid 8, 0 \rangle$
$(12, 0)$ **on** $\mathtt{S1}(\mathtt{s}, (24, 0))$	$\langle \mathtt{pck}$ **on** $\mathtt{S1}\,(\mathtt{s}, \langle \mathtt{pck} \mid 24, 0\rangle) \mid 12, 0\rangle$
*^2	$e{:}\{\nu{:}\mathtt{pck} \mid 2$ **div** $\pi(\nu)\} \rightarrow \langle \mathtt{pck} \mid \pi(e)/2, \varphi(e)\rangle$

Steps of refinement clock calculus

1. *Structural typing*
   - Solve base types (ck_b : `pck`, **on**)
   - Refinements opaque holes $\star$
2. *Refinement typing*
   - ck_b fixed
   - Solve non-view refinements
3. *View closing*
   - Solve clock views

Example

```
node main (i: rate (10,0); j: rate (20,0); s: rate (15,0))
returns (o)
var x, y, z;
let
  x = i when S1(s);
  y = j when S2(s);
  z = y *^2;
  o = merge(s, S1->x, S2->z);
tel
```

$$\texttt{i} : \langle \texttt{pck} \mid 10, 0 \rangle \quad \texttt{j} : \langle \texttt{pck} \mid 20, 0 \rangle \quad \texttt{s} : \langle \texttt{pck} \mid 15, 0 \rangle \quad \texttt{o} : \langle \texttt{pck} \mid 10, 0 \rangle$$

$$\texttt{x} : \langle \texttt{pck} \textbf{ on } S1\,(\texttt{s}, \langle \texttt{pck} \mid 60, 0 \rangle) \mid 10, 0 \rangle$$

$$\texttt{y} : \langle \texttt{pck} \textbf{ on } S1\,(\texttt{s}, \langle \texttt{pck} \mid 60, 0 \rangle) \mid 20, 0 \rangle$$

$$\texttt{z} : \langle \texttt{pck} \textbf{ on } S1\,(\texttt{s}, \langle \texttt{pck} \mid 60, 0 \rangle) \mid 10, 0 \rangle$$

Example — Rate inconsistency

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2-> y );
```

$$\frac{\begin{array}{c}\pi(\mathrm{x}) = 10 \wedge \varphi(\mathrm{x}) = 0 \wedge \\ (\forall ck.\ (\pi(ck) = 20 \wedge \varphi(ck) = 0) \implies \\ (\pi(ck) = \pi(\mathrm{x}) \wedge \varphi(ck) = \varphi(\mathrm{x})))\end{array}}{\begin{array}{c}H \vdash \mathrm{y} \Leftarrow \langle \mathrm{pck}\ \mathbf{on}\ \mathrm{S2}\,(\mathrm{s}, \langle \mathrm{pck} \mid \pi(w), \varphi(w)\rangle) \mid \pi(\mathrm{x}), \varphi(\mathrm{x})\rangle \\ H \vdash \mathrm{x} \Rightarrow \langle \mathrm{pck}\ \mathbf{on}\ \mathrm{S1}\,(\mathrm{s}, w) \mid 10, 0\rangle \\ H \vdash \mathrm{y} \Rightarrow \langle \mathrm{pck}\ \mathbf{on}\ \mathrm{S2}\,(\mathrm{s}, w') \mid 20, 0\rangle\end{array}}{H \vdash \texttt{merge(s, S1->x, S2->y)} \Rightarrow}$$

Example — Rate inconsistency

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2-> y );
```

$$\frac{\pi(\mathtt{x}) = 10 \wedge \varphi(\mathtt{x}) = 0 \wedge (\forall ck.\ (\pi(ck) = 20 \wedge \varphi(ck) = 0) \implies (\pi(ck) = \pi(\mathtt{x}) \wedge \varphi(ck) = \varphi(\mathtt{x})))}{H \vdash \mathtt{y} \Leftarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, \langle \mathtt{pck} \mid \pi(w), \varphi(w)\rangle) \mid \pi(\mathtt{x}), \varphi(\mathtt{x})\rangle \quad H \vdash \mathtt{x} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S1}\,(\mathtt{s}, w) \mid 10, 0\rangle \quad H \vdash \mathtt{y} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, w') \mid 20, 0\rangle}{H \vdash \mathtt{merge(s,\ S1->x,\ S2->y)} \Rightarrow}$$

Example — Rate inconsistency

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2-> y );
```

$$\frac{\begin{array}{c}\pi(\mathtt{x}) = 10 \wedge \varphi(\mathtt{x}) = 0 \wedge \\ (\forall ck.\ (\pi(ck) = 20 \wedge \varphi(ck) = 0) \implies \\ (\pi(ck) = \pi(\mathtt{x}) \wedge \varphi(ck) = \varphi(\mathtt{x})))\end{array}}{H \vdash \mathtt{y} \Leftarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, \langle \mathtt{pck} \mid \pi(w), \varphi(w)\rangle) \mid \pi(\mathtt{x}), \varphi(\mathtt{x})\rangle}$$

$$\frac{\begin{array}{c}H \vdash \mathtt{x} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S1}\,(\mathtt{s}, w) \mid 10, 0\rangle \\ H \vdash \mathtt{y} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, w') \mid 20, 0\rangle\end{array}}{H \vdash \mathtt{merge(s,\ S1->x,\ S2->y)} \Rightarrow}$$

Example — Rate inconsistency

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2-> y );
```

$$\frac{\begin{array}{c}\pi(\mathtt{x}) = 10 \wedge \varphi(\mathtt{x}) = 0 \wedge \\ (\forall ck.\ (\pi(ck) = 20 \wedge \varphi(ck) = 0) \implies \\ (\pi(ck) = \pi(\mathtt{x}) \wedge \varphi(ck) = \varphi(\mathtt{x})))\end{array}}{\begin{array}{c}H \vdash \mathtt{y} \Leftarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, \langle \mathtt{pck} \mid \pi(w), \varphi(w)\rangle) \mid \pi(\mathtt{x}), \varphi(\mathtt{x})\rangle \\ H \vdash \mathtt{x} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S1}\,(\mathtt{s}, w) \mid 10, 0\rangle \\ H \vdash \mathtt{y} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, w') \mid 20, 0\rangle\end{array}}{H \vdash \mathtt{merge(s,\ S1->x,\ S2->y)} \Rightarrow}$$

Example — Rate inconsistency

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2-> y );
```

$$\frac{\begin{array}{c}\pi(\mathtt{x}) = 10 \wedge \varphi(\mathtt{x}) = 0 \wedge \\ (\forall ck.\ (\pi(ck) = 20 \wedge \varphi(ck) = 0) \implies \\ (\pi(ck) = \pi(\mathtt{x}) \wedge \varphi(ck) = \varphi(\mathtt{x})))\end{array}}{\begin{array}{c}H \vdash \mathtt{y} \Leftarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, \langle \mathtt{pck} \mid \pi(w), \varphi(w)\rangle) \mid \pi(\mathtt{x}), \varphi(\mathtt{x})\rangle \\ H \vdash \mathtt{x} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S1}\,(\mathtt{s}, w) \mid 10, 0\rangle \\ H \vdash \mathtt{y} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, w') \mid 20, 0\rangle\end{array}}{H \vdash \mathtt{merge(s,\ S1->x,\ S2->y)} \Rightarrow}$$

Example — Rate inconsistency

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2-> y );
```

$$\frac{\begin{array}{c}\pi(\mathtt{x}) = 10 \wedge \varphi(\mathtt{x}) = 0 \wedge \\ (\forall ck.\ (\pi(ck) = 20 \wedge \varphi(ck) = 0) \implies \\ (\pi(ck) = \pi(\mathtt{x}) \wedge \varphi(ck) = \varphi(\mathtt{x}))) \implies \bot \end{array}}{\begin{array}{c} H \vdash \mathtt{y} \Leftarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, \langle \mathtt{pck} \mid \pi(w), \varphi(w)\rangle) \mid \pi(\mathtt{x}), \varphi(\mathtt{x})\rangle \\ H \vdash \mathtt{x} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S1}\,(\mathtt{s}, w) \mid 10, 0\rangle \\ H \vdash \mathtt{y} \Rightarrow \langle \mathtt{pck}\ \mathbf{on}\ \mathtt{S2}\,(\mathtt{s}, w') \mid 20, 0\rangle \end{array}}{H \vdash \mathtt{merge(s,\ S1->x,\ S2->y)} \Rightarrow}$$

Example — View closing

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2->z);
```

$\mathtt{i} : \langle \mathtt{pck} \mid 10, 0 \rangle \quad \mathtt{j} : \langle \mathtt{pck} \mid 20, 0 \rangle \quad \mathtt{s} : \langle \mathtt{pck} \mid 15, 0 \rangle \quad \mathtt{o} : \langle \mathtt{pck} \mid 10, 0 \rangle$

$$
\begin{array}{ll}
\mathtt{x} & \langle \mathtt{pck}\ \textbf{on}\ S1\,(\mathtt{s}, w) \mid 10, 0 \rangle \\
\mathtt{y} & \langle \mathtt{pck}\ \textbf{on}\ S1\,(\mathtt{s}, w') \mid 20, 0 \rangle \\
\mathtt{z} & \langle \mathtt{pck}\ \textbf{on}\ S1\,(\mathtt{s}, w'') \mid 10, 0 \rangle \\
w & \{\nu{:}\mathtt{pck} \mid 10\ \textbf{div}\ \pi(\nu) \wedge 15\ \textbf{div}\ \pi(\nu)\} \\
w' & \{\nu{:}\mathtt{pck} \mid 20\ \textbf{div}\ \pi(\nu) \wedge 15\ \textbf{div}\ \pi(\nu)\} \\
w'' & \{\nu{:}\mathtt{pck} \mid \pi(\nu) = \pi(w') \wedge \pi(\nu) = \pi(w)\}
\end{array}
$$

Example — View closing

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2->z);
```

$$10 \textbf{ div } w_{period} \wedge 15 \textbf{ div } w_{period} \wedge 20 \textbf{ div } w'_{period} \wedge 15 \textbf{ div } w'_{period} \wedge$$
$$w''_{period} = w'_{period} \wedge w''_{period} = w_{period}$$
$$minimize(w, w', w'')$$

$$\begin{array}{ll} w & \{\nu{:}\texttt{pck} \mid 10 \textbf{ div } \pi(\nu) \wedge 15 \textbf{ div } \pi(\nu)\} \\ w' & \{\nu{:}\texttt{pck} \mid 20 \textbf{ div } \pi(\nu) \wedge 15 \textbf{ div } \pi(\nu)\} \\ w'' & \{\nu{:}\texttt{pck} \mid \pi(\nu) = \pi(w') \wedge \pi(\nu) = \pi(w)\} \end{array}$$

Example — View closing

```
x = i when S1(s);
y = j when S2(s);
z = y *^2;
o = merge(s, S1->x, S2->z);
```

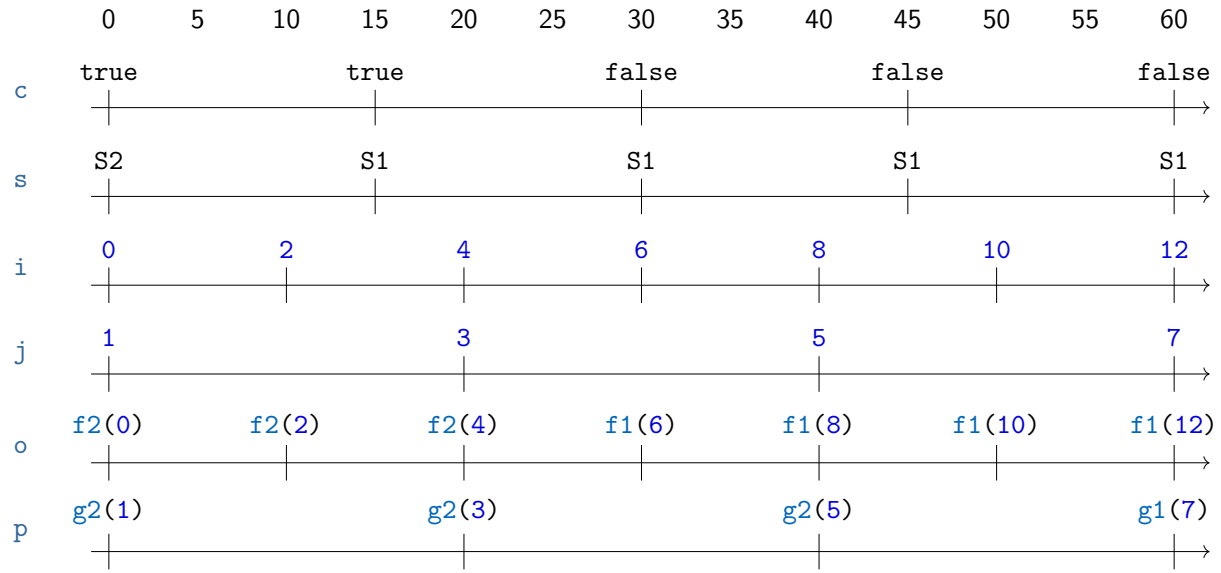
$$10 \textbf{ div } w_{period} \wedge 15 \textbf{ div } w_{period} \wedge 20 \textbf{ div } w'_{period} \wedge 15 \textbf{ div } w'_{period} \wedge$$
$$w''_{period} = w'_{period} \wedge w''_{period} = w_{period}$$
$$minimize(w, w', w'')$$

w	$\langle \texttt{pck} \mid 60, 0 \rangle$
w'	$\langle \texttt{pck} \mid 60, 0 \rangle$
w''	$\langle \texttt{pck} \mid 60, 0 \rangle$

Table of contents

Controlling the kind of mode change protocol

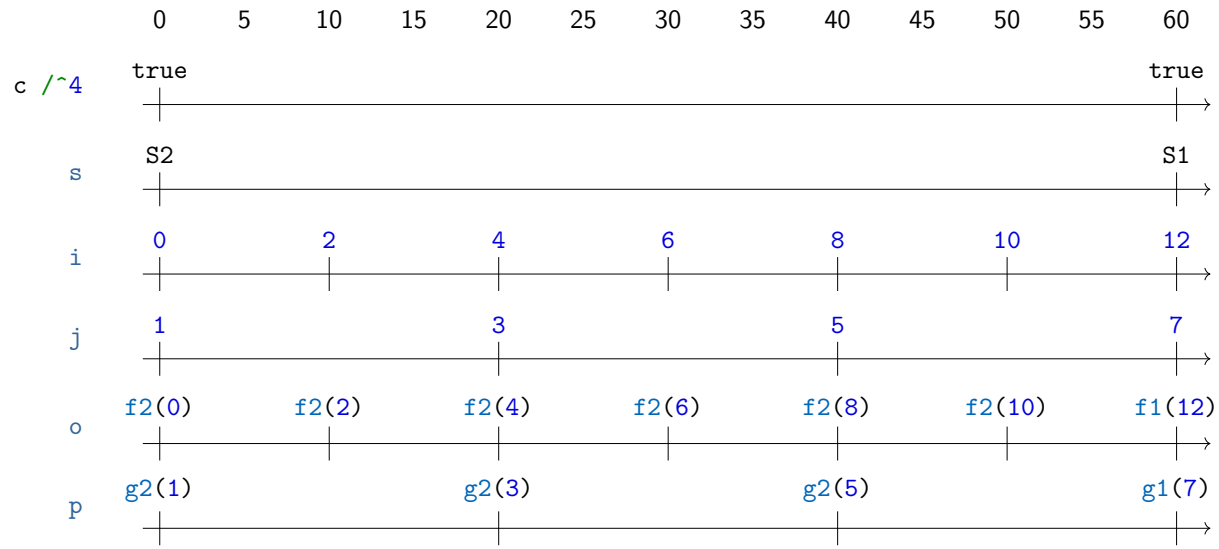
```
node main (i : rate (10,0); j : rate(20,0); c : rate(15,0))
  returns (o,p)
let
  automaton
  | S1 ->
    unless c then S2;
    o = f1(i);
    p = g1(j);
  | S2 ->
    unless c then S1;
    o = f2(i);
    p = g2(j);
  end
tel
```



- View of o : $(30, 0)$
- View of p : $(60, 0)$

Controlling the kind of mode change protocol

```
node main (i : rate (10,0); j : rate(20,0); c : rate(15,0))
  returns (o,p)
let
  automaton
  | S1 ->
    unless c/^4 then S2;
    o = f1(i);
    p = g1(j);
  | S2 ->
    unless c/^4 then S1;
    o = f2(i);
    p = g2(j);
  end
tel
```



- View of o : $(60, 0)$
- View of p : $(60, 0)$

Table of contents

Table of contents

Summary

Defined 2 extensions of the PRELUDE language

- Generating predictable multicore code
  - *Back-end* extension
  - Based on the AER model
  - FPGA evaluation shows benefits of delegating to the compiler
- Synchronous semantics of multi-mode systems
  - *Front-end* extension
  - Views decouple execution rate from mode change rate
  - Reuse of the transpilation technique
  - Refinement clock calculus checks rate consistency

Perspectives — Multi-mode synchronous semantics

Code generation for clock views

- Compiler stops after clock calculus
- Clock views need changes in code generation

⇒ Seems feasible by reusing existing protocol generation

Typing system

- Refinement typing is expressive, but complex

⇒ Find a sufficient, but less complex, typing discipline

- Lack of polymorphism

⇒ Definition of a *liquid* typing system

Program synthesis

- Clock views are a poor man's program synthesis

⇒ Use SMT solver to complete partial programs

Publications


Fort, Frédéric et Julien Forget (2019). « Code generation for multi-phase tasks on a multi-core distributed memory platform ». In : *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, p. 1-6.


Fort, Frédéric et Julien Forget (2022). « Synchronous semantics of multi-mode multi-periodic systems ». In : *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, p. 1248-1257.