Synchronous Logical Execution Time: towards formal verification

Synchron'21

Fabien Siron <fabien.siron@krono-safe.com>

November 24, 2021

KRONO-SAFE
Safe design in real-time

Agenda

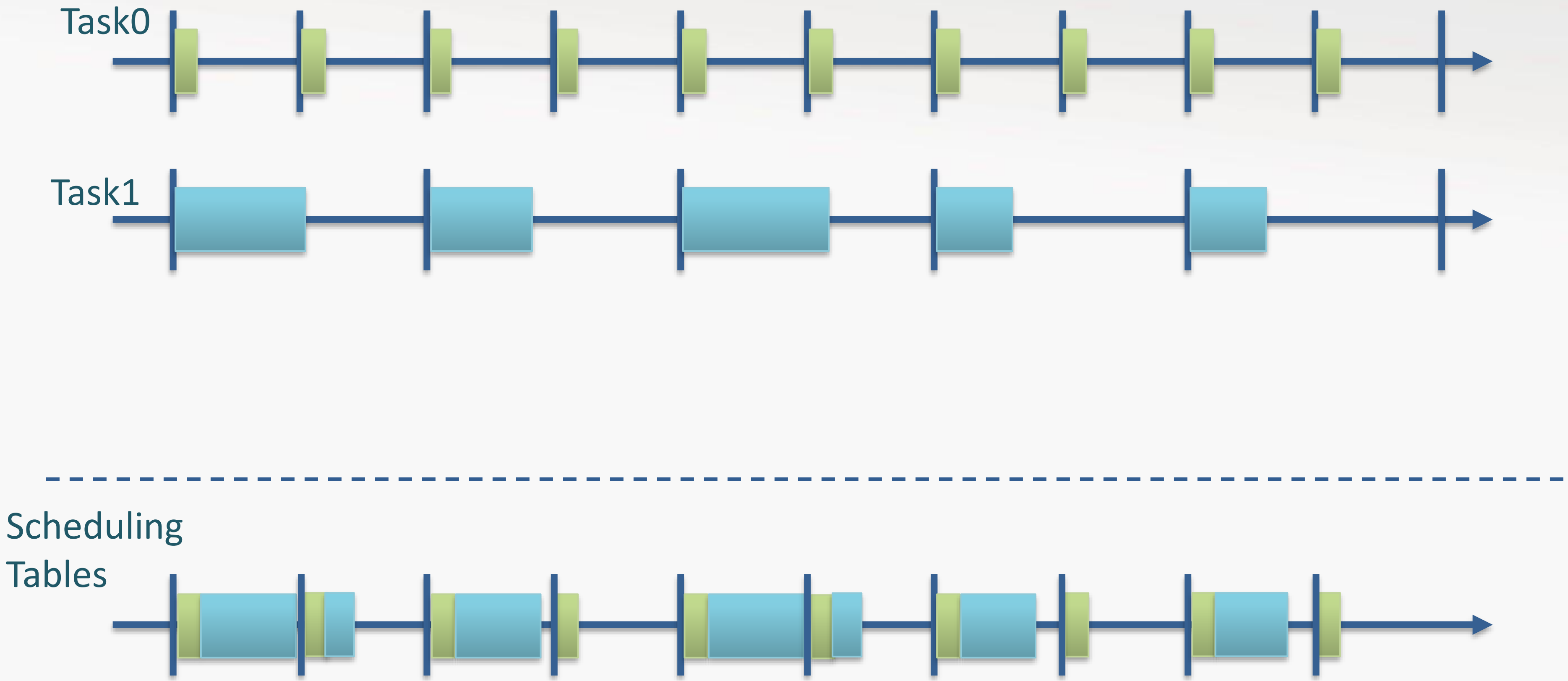
- Context and motivation
- synchronous Logical Execution Time (sLET)
- The PsyC language
- Overview of the PsyC semantics
- Toward Formal Verification
- Conclusion/Perspectives

Safety-Critical Real-Time Systems

- **Krono-Safe:**
  - Software provider in the context of safety-critical systems
    - Domains: nuclear, avionics…
- **Certification issues:**
  - Predictability & Reproducibility
  - Temporal and functional requirements
- **Common solution:**
  - Use high-level deterministic formalisms
  - Verify high-level requirements using formal methods

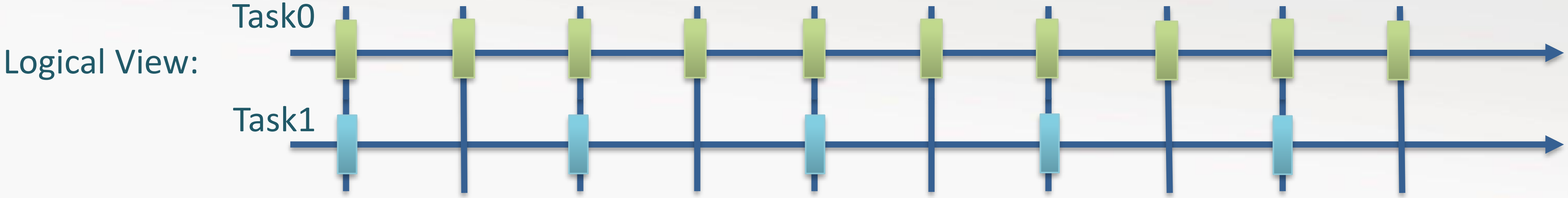
Motivation: modeling of real-time systems

- **Classical example: periodic real-time**

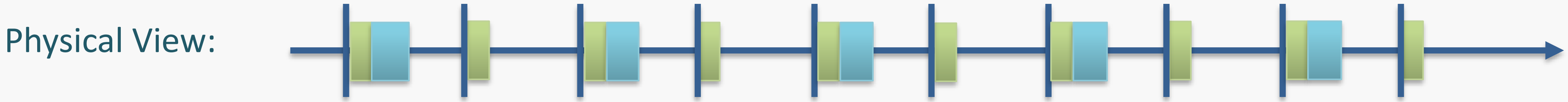


Motivation: modeling of real-time systems

- **The synchronous approach (Esterel, Lustre [1]):**
  - **Model only activation in *logical time***



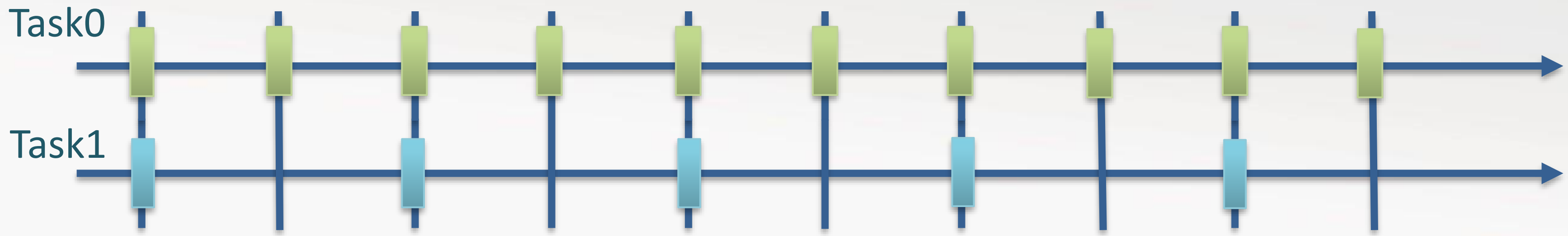
  - **Traditional real-time interpretation:**
    - **Bind logical clocks to physical clocks (e.g. timer)**
    - **Synchronous cycle computation can use all physical time until next cycle triggering**



Motivation: modeling of real-time systems

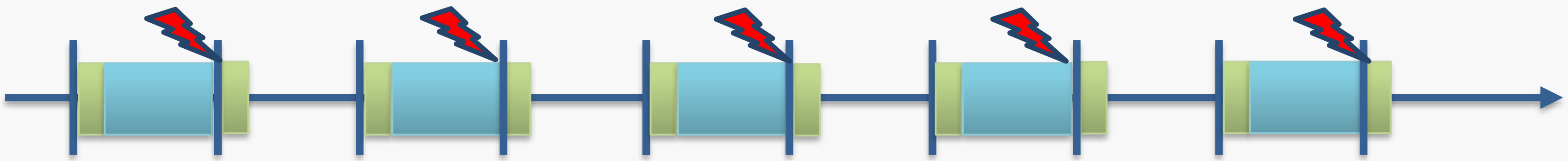
- **Problem: modeling long-period tasks**
  - *Invisible in the synchronous model*

Logical View:



  - Consider than the duration of task 2 is longer than the duration between successive cycle triggers:

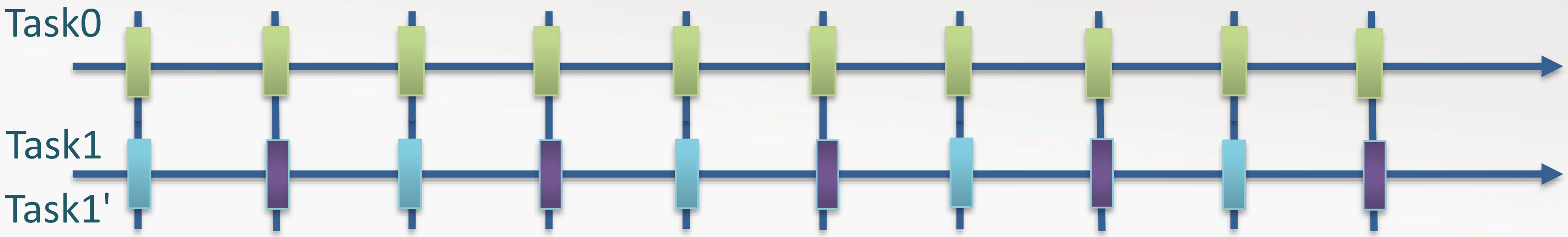
Physical View:



Motivation: modeling of real-time systems

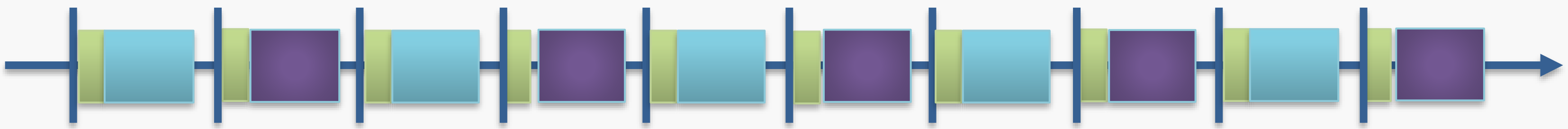
- **Problem: modeling long-period tasks**
  - *Possible synchronous solution: retiming/slicing*

Logical View:



  - **BUT: static and manual slicing is difficult and non-portable**

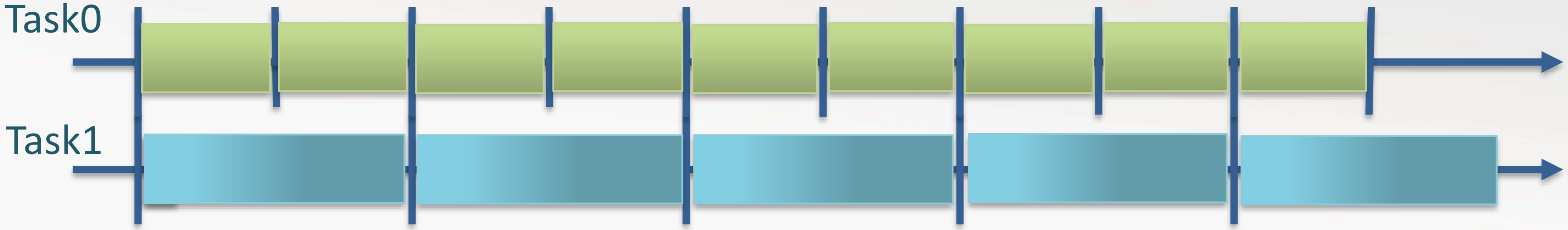
Physical View:



Motivation: modeling of real-time systems

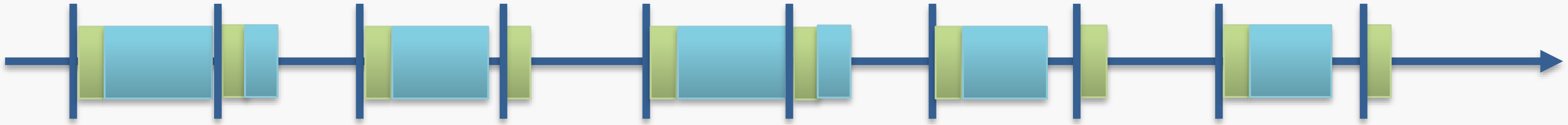
- **Logical Execution Time [2]: extension of synchrony**
  - Specify the *logical time* cycle a task may span

Logical View:



  - Classical synchronous composition is lost, but
  - Real-time interpretation is more general
    - ➢ **Contract separating application design and platform resource**

Physical View:



The synchronous LET paradigm

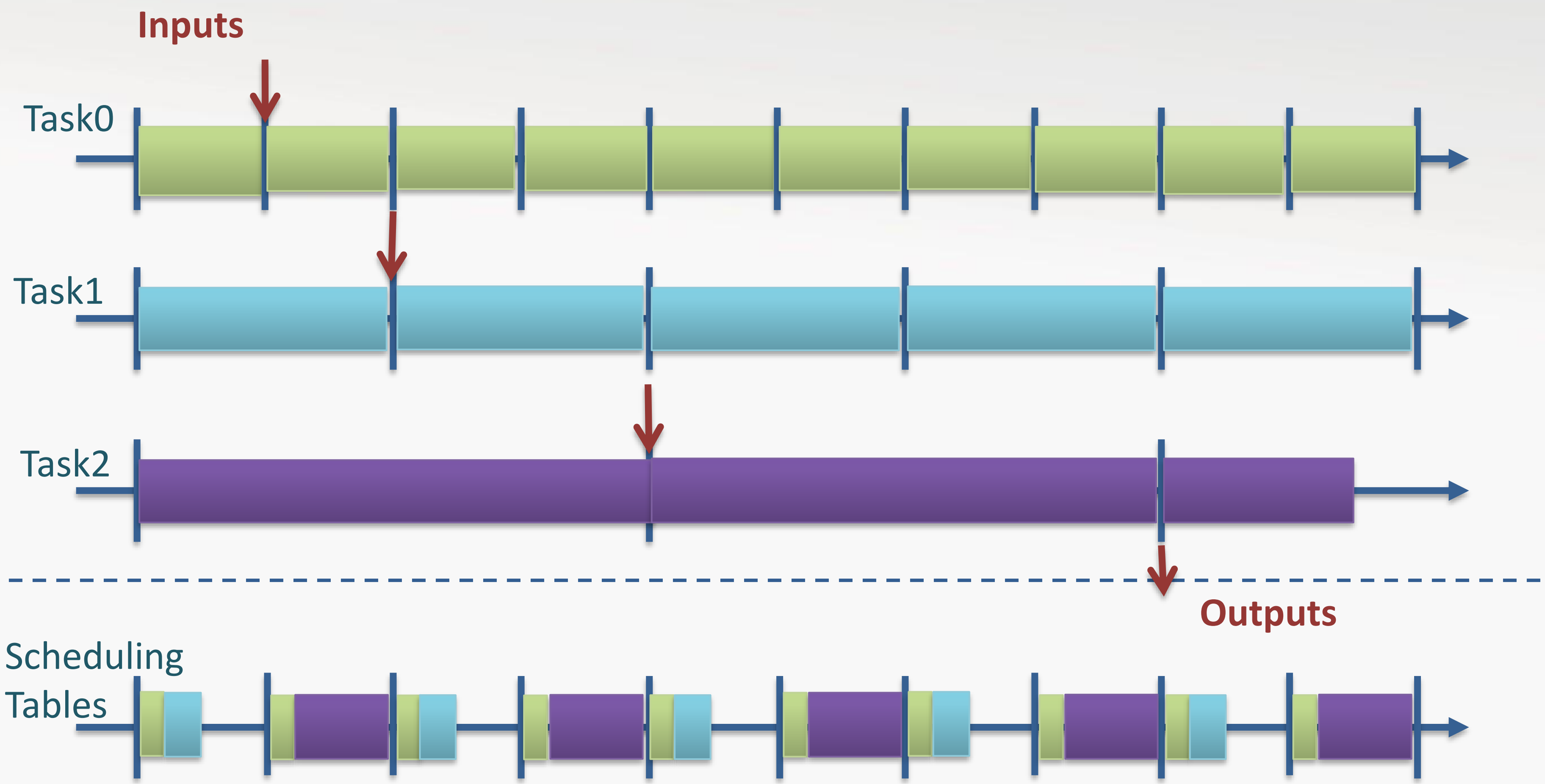
- **A synchronous extension of LET: PsyC [4]**
  - Given a statement advance which fixes the bounds of a logical interval:

$$\begin{matrix} advance\ 5\ with\ min; \\ advance\ 1\ with\ hour \end{matrix} \neq advance\ 65\ with\ min$$

  - Contrary to classical LET, time is not cumulative
  - Bounds are relative to the ticks of some logical clocks

Logical Execution Time: temporal requirements

- **A typical verification problem: end-to-end latencies [3]**



The PsyC language: concepts

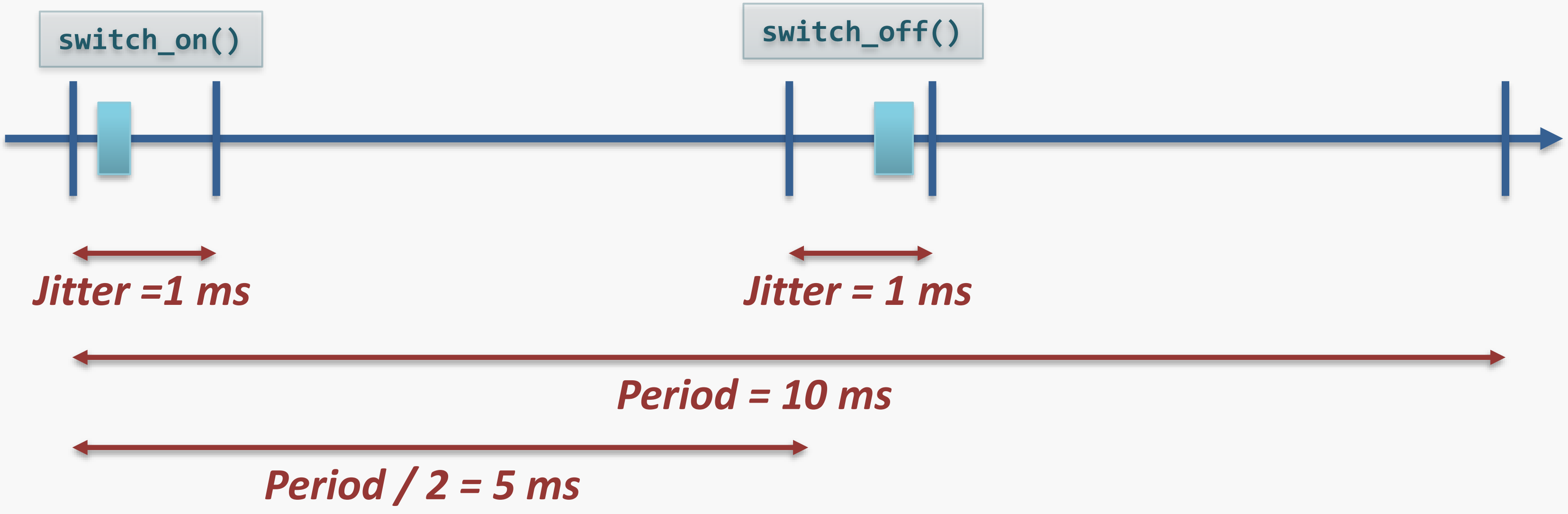
- **Primary temporal Sources**
  - generating global rhythms
  - most often, only one linked to real time
- **Periodic Clocks**
  - subdivising source ticks
- **Program reactions**
  - may span a fixed interval: sLET interval durations
- **Temporal variables**
  - share values between agents
  - persistent values and values updated on sLET interval bounds

The PsyC language

- **Produced by Krono-Safe, dedicated to the safety-critical real-time software integration.**
  - Based on a technology developed by the CEA (Oasis and PharOS projets)
- **Implement the (s)LET model**
  - Enable a **deterministic** communication model
  - Allow complex, dynamic temporal behavior
- **Extension of the C language:**
  - Multiple concurrent agents with **functional** (C code) and **non-functional** parts (advance statement)
  - Temporal sources and clocks
  - Communication means

The PsyC language: a simple example

- **LED-blinking example – specification:**
  - Period = 10ms
  - Duty Cycle (ON) between 0.4 and 0.6
  - Switching jitter <= 1ms



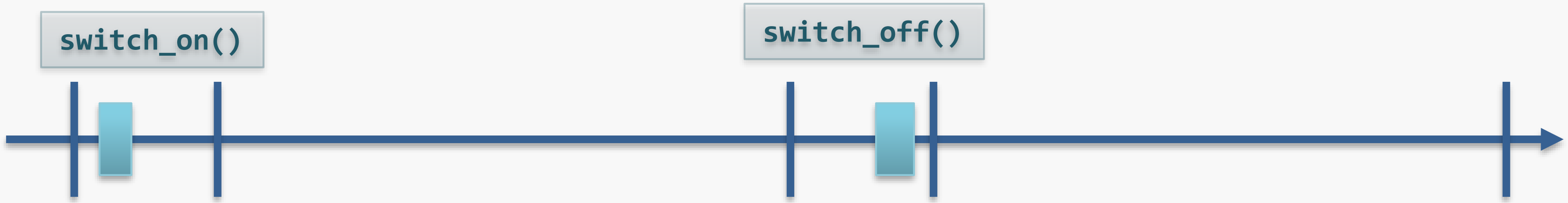
The PsyC language: a simple example

- **LED-blinking example – PsyC code:**

```
source realtime_ms;
clock c_jitter = realtime_ms;
clock c_half_period = 5*realtime_ms;
clock c_period = 2*c_half_period;

agent Blinker
{
  body start
  {
    switch_on();
    advance 1 with c_jitter;
    /* do nothing */
    advance 1 with c_half_period;

    switch_off();
    advance 1 with c_jitter;
    /* do nothing */
    advance 1 with c_period;
  }
}
```



The PsyC language: a simple example

- **LED-blinking example – PsyC code:**

```
temporal mode = OK with c_ms;

agent Blinker {
  body start {
    if ($[0]mode == ERROR)
      jump blink;
    advance 1 with c_ms;
  }
  body blink {
    switch_on();
    advance 1 with c_jitter;
    /* do nothing */
    advance 1 with c_half_period;

    switch_off();
    advance 1 with c_jitter;
    /* do nothing */
    advance 1 with c_period;
  }
}
```

The PsyC language: a simple example

- **LED-blinking example – PsyC code:**

```
temporal mode = OK with c_ms;

agent Blinker {
  body start {
    if ($[0]mode == ERROR)
      jump blink;
    advance 1 with c_ms;
  }
  body blink {
    switch_on();
    advance 1 with c_jitter;
    /* do nothing */
    advance 1 with c_half_period;

    switch_off();
    advance 1 with c_jitter;
    /* do nothing */
    advance 1 with c_period;
  }
}
```

```
agent ErrorManager {
  body start {
    if (/* some condition … */)
      mode = ERROR;
    advance 1 with c_ms;
  }
}
```

The PsyC semantics: abstract syntax

- **Syntax overview:**

$$
\begin{array}{lcl}
\boldsymbol{decl} & ::= & \boldsymbol{clk} \mid \boldsymbol{agt} \mid \ldots (\boldsymbol{coms}) \\
\boldsymbol{clk} & ::= & \boldsymbol{source\ c} \mid \boldsymbol{clock\ c = n_1 \times c_p + n_2} \\
\boldsymbol{agt} & ::= & \boldsymbol{agent\ id\ \ body^{+}} \\
\boldsymbol{body} & ::= & \boldsymbol{body\ id\ stmt}
\end{array}
$$

The PsyC semantics: abstract syntax

- **Syntax overview:**

$$
\begin{array}{lcl}
stmt & ::= & id := expr \\
 & | & stmt_1; stmt_2 \\
 & | & advance\ n\ with\ c \\
 & | & if\ expr\ then\ stmt_1\ else\ stmt_2 \\
 & | & \ldots
\end{array}
$$

The PsyC semantics: Esterel translation

- **Esterel translation:**
  - Synchronous interpretation of PsyC
    - Semantics through translation
    - Allow to re-use existing tools
  - Both PsyC and Esterel are imperative and control-flow
  - Main ideas:
    - Clock ticks are signals
    - Advance are await
    - Local variables are Esterel variables
    - Temporal variables are valued signals

The PsyC semantics: Esterel translation

- **Esterel Translation: clocks**

$$T(clock\ c = p\ * c_p + o) \stackrel{\text{def}}{=} \begin{array}{l} await\ o; \\ loop \\ \quad emit\ c \\ each\ p\ c_p \end{array}$$

The PsyC semantics: Esterel translation

- **Esterel Translation: agent statements**

$$T(id\ := expr) \stackrel{\text{def}}{=} id := \ T(expr)$$

$$T(stmt_1; stmt_2) \stackrel{\text{def}}{=} T(stmt_1); T(stmt_2)$$

$$T(if\ expr\ then\ s_1 else\ s_2) \stackrel{\text{def}}{=} if\ T(expr) then\ T(s_1) else\ T(s_2)$$

$$T(advance\ n\ with\ c) \stackrel{\text{def}}{=} await\ n\ c; run\ UpdateOutputs(vars \dots)$$

– *UpdateOutputs* emit valued signals for each local variable

Toward Formal Verification

- **Global methodology:**
  - ➢ Model properties as synchronous observers in Esterel [5]
- **Example:**

```
/* blinker start body */
loop
[
  if ?mode = ERROR then
    next_body := blink;
    exit body;
  await 1 c_ms;
]
```

```
private_mode := OK;
/* error manager start body */
loop
[
  if /* some condition */ then
    private_mode := ERROR;
  await 1 c_ms;
  emit mode(private_mode);
]
```

Toward Formal Verification

- **Global methodology:**
  - ➢ Model properties as synchronous observers in Esterel [5]
- **Example:**

```
/* blinker blink body */
loop
[
  /* switch_on() */
  await 1 c_jitter;
  await 1 c_half_period;
  /* switch_off(); … */
  await 1 c_jitter;
  await 1 c_period;
]
```

Toward Formal Verification

- **Global methodology:**
  - ➢ Model properties as synchronous observers in Esterel [5]
- **Example: minimum duty-cycle (>= 4 ms)**

```
/* blinker blink body */
loop
[
  /* switch_on() */
  await 1 c_jitter;
  abort
    await 1 c_half_period;
    emit ERROR;
  when 4 realtime_ms;
  /* switch_off(); … */
]
```

Conclusion and Perspectives

- **Sum-up:**
  - Logical Execution Time extends synchrony with logical durations
  - Synchronous interpretation of Logical Execution Time with PsyC
- **In practice in industry:**
  - (s)LET languages are usually used as integretion/coordination language:
    - i.e. software integration of synchronous (functional) components
- **Perspectives:**
  - Model more complex properties (e.g. end-to-end latencies)
  - Optimize to only represent « noticeable » instants: primary for efficient verification

References

[1] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Guernic, and R. Simone, “The Synchronous Languages 12 Years Later,” *Proceedings of the IEEE*, vol. 91, pp. 64–83, Feb. 2003, doi: 10.1109/JPROC.2002.805826.

[2] C. M. Kirsch and A. Sokolova, “The Logical Execution Time Paradigm,” in *Advances in Real-Time Systems*, S. Chakraborty and J. Eberspächer, Eds. Berlin, Heidelberg: Springer, 2012, pp. 103–120. doi: 10.1007/978-3-642-24349-3_5.

[3] R. Wyss, F. Boniol, C. Pagetti, and J. Forget, “End-to-end latency computation in a multi-periodic design,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC ’13*, Coimbra, Portugal, 2013, p. 1682. doi: 10.1145/2480362.2480678.

[4] C. Aussagues, C. Cordonnier, M. Aji, V. David, and J. Delcoigne, “OASIS: A New Way to Design Safety Critical Applications,” *IFAC Proceedings Volumes*, vol. 29, no. 5, pp. 21–26, Nov. 1996, doi: 10.1016/S1474-6670(17)46349-X.

[5] N. Halbwachs, F. Lagnier, and P. Raymond, “Synchronous Observers and the Verification of Reactive Systems,” in *Algebraic Methodology and Software Technology (AMAST’93)*, London, 1994, pp. 83–96. doi: 10.1007/978-1-4471-3227-1_8.