KRONO-SAFE

Safe design in real-time

ASTERIOS

D'un modèle de calcul parallèle déterministe à la réalisation d'un système Temps-Réel Critique

Ecole d'Eté Temps-Réel 2017

Emmanuel OHAYON

emmanuel.ohayon@krono-safe.com

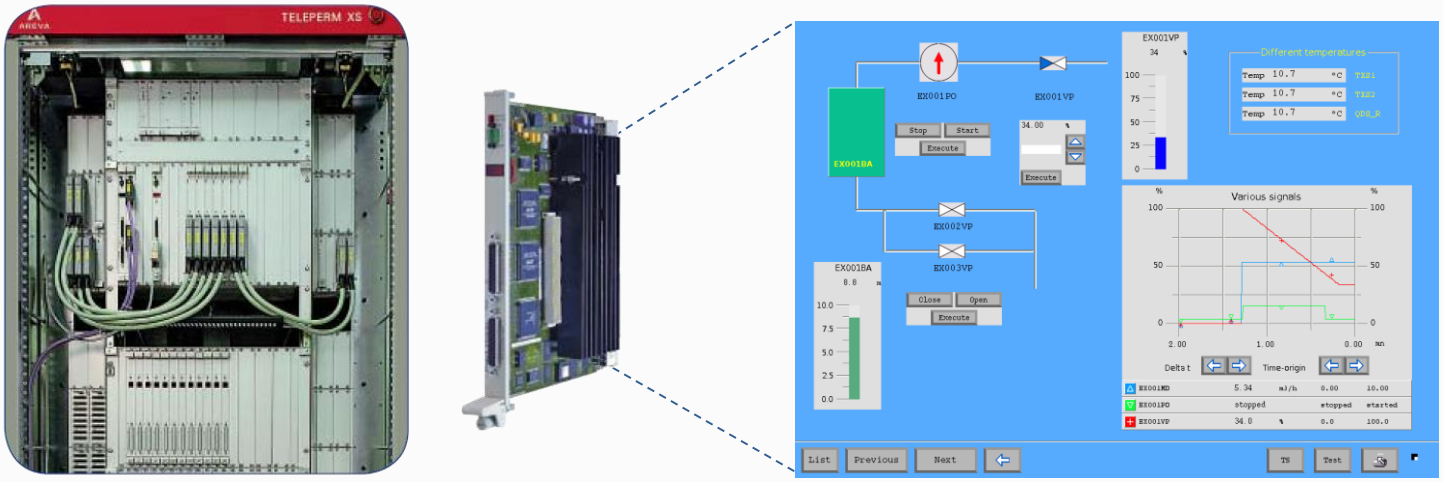
Who We Are and What We Do

- **The company: Krono-Safe**
  - Software Editor
  - Founded in 2011
  - Roughly 40 employees based mainly in Orsay

- **The technology: ASTERIOS®**
  - Software Suite for Real-Time, Safety-Critical Applications
    - Real-Time, Multi-Task and safety-oriented Programming Model
    - Dedicated Programming Language and its compiler
    - Real-Time Operating System in charge of running the application on bare-metal
    - User-Friendly IDE with adapted tools for analysis, simulation & debug
  - Based on a technology developed by CEA
    - Used by Areva in last generation nuclear power-plant
  - Targeted Markets: Aerospace & Automotive
    - Ongoing Certification at the highest level of trust for airborne applicative software (DO-178C DAL-A)

Areva

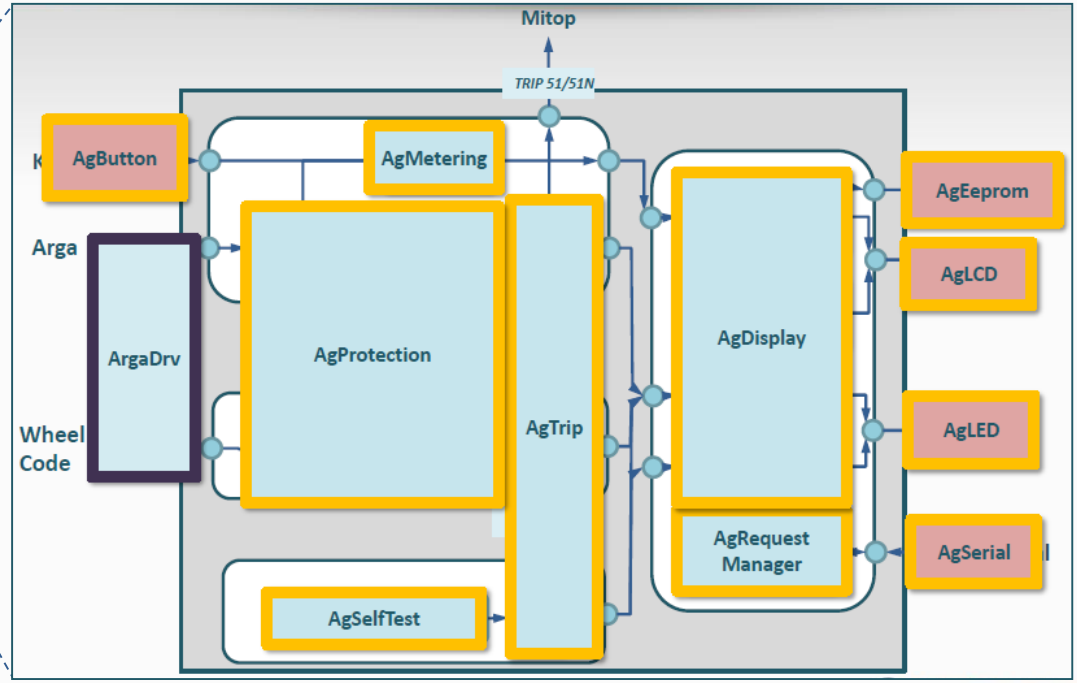
- **First implementation by AREVA and CEA**
- **TELEPERM XS – QDS is based on OASIS technology, which is used by ASTERIOS, and has achieved IEC 60880 highest level of certification:**
  - Qualified Display System
  - Data processing & visualization
  - I/Os: Ethernet, touchscreen, keyboard, mouse, etc.



Reactor Instrumentation & Control (start, stop and post-accident phases)

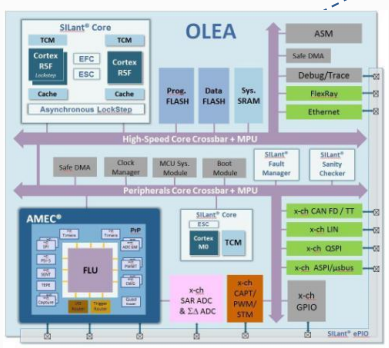
Schneider Electric

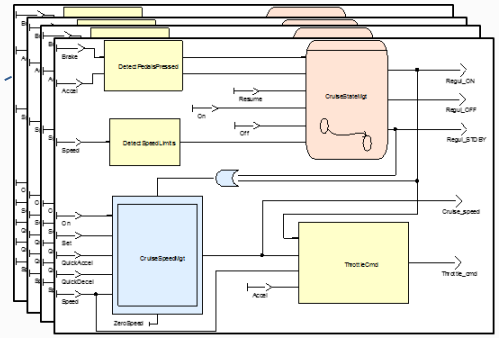
- **Integration with a Circuit Breaker**
  - Same hardware used (STM32 Cortex M3 / single-core)
  - More than 95% of legacy code reuse
  - Demonstrated error confinement (improved availability)
  - Full automated configuration (partition, buffer, stack, scheduling)

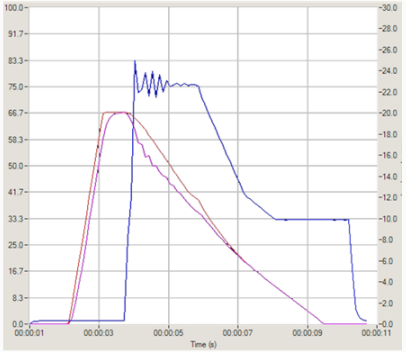


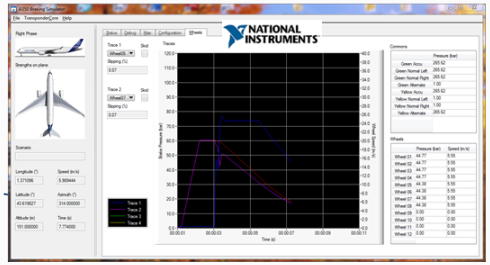
Safran

- **Integration within a breaking system**
  - Same hardware used (Scaleo Chip Cortex R5F / dual-core)
  - Full reuse of code generated by SCADE Suite
  - Simulation then integration on target
  - Automated generation of scheduling









History of Computation Models

- **Formal system of operations that can implement an algorithm**

- **1936: 1st contribution by Turing & Church**
  - **Defined universal computing machines**
  - ***Colossus* was functional in December 1943 (!)**

- **1950s: Finite-State Machines & Formal Languages**

- **1980s: Parallel and Distributed Computing**

Computation Models are Everywhere

- **Famous Examples**
  - Sequential
    - Finite State Machines
    - Turing Machine, Lambda Calculus, µ-Recursive Functions
  - Parallel
    - Petri Nets
    - Kahn Process Network
    - GALS
- **Typical use:**
  1. Write your algorithm with the CM of your choice
  2. Mathematically prove properties on the algorithm (*halting, correctness, complexity…*)
  3. Program your algorithm into a machine that implements your Computation Model
     - For simulation…
     - …or use an automated tool that translates your program to a simpler Computation Model (*code generator, compiler…*) for final execution
  4. Test it (*preferably…*)

- **Yet all these model fail to address one major parameter in software engineering: time**

Some Timed Computation Models

- **Timed (Büchi) Automata**
  - Extension of finite automata with *clock* objects
  - Transitions may carry guards made of clock constraints

- **The BIP Framework (Verimag)**

- **Lustre/Esterel synchronous model**
  - Although not real-time model *per se*: based on logical time only – more of a dataflow model
  - Popular for aerospace applications

A Real-Time Programming Model

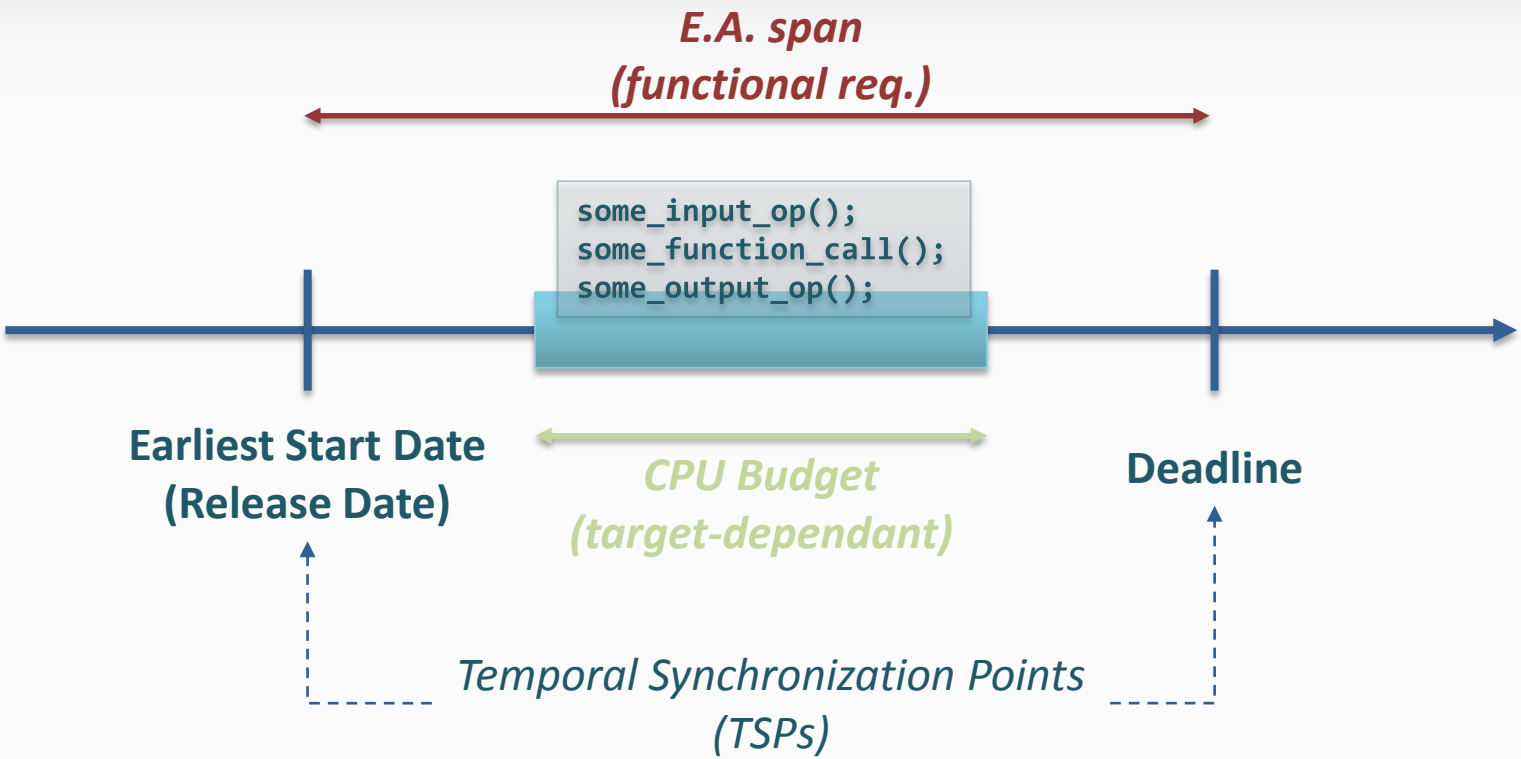
- **Multi-task programming model**
  - 1 task = 1 sequential execution unit
  - Maps to mono- or multi-core architectures: 1 task is mapped to 1 core
- **Time-Triggered Paradigm**
  - Task activation only set by the current date
    - No lock/semaphore/mutex nightmare: no deadlock
  - Enables a fully deterministic communication model
- **Allows complex, dynamic temporal behaviors**
  - Offers an abstraction for the design of temporal behavior

Elementary Action

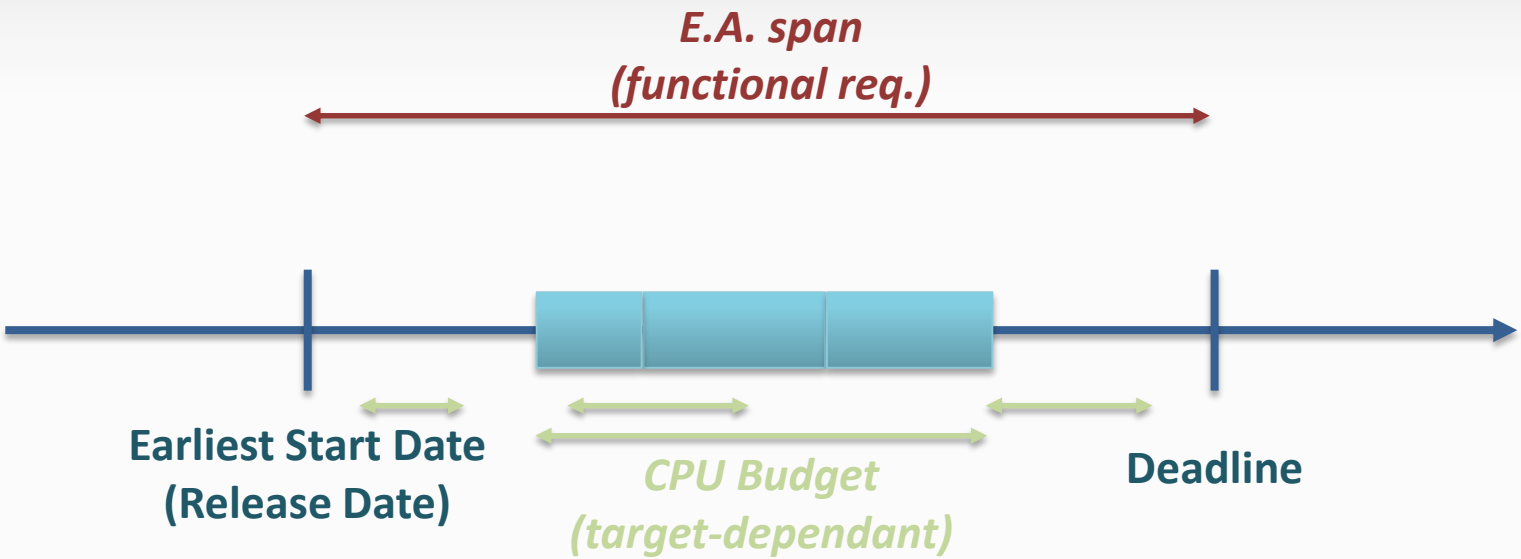
- **Building block of a real-time application**
  - Agent = Task = Succession of Elementary Actions

- **An Elementary Action is:**
  - temporally bounded by 2 Temporal Synchronization Points (TSPs):
    - an Earliest Start Date (or Release Date)
    - a Deadline;
  - a piece of functional code
    - that shall be executed sequentially (*e.g a C function*);
    - runs some time after the Earliest Start Date, and before the Deadline;
    - in all, shall complete in a given total CPU time called budget

Elementary Action



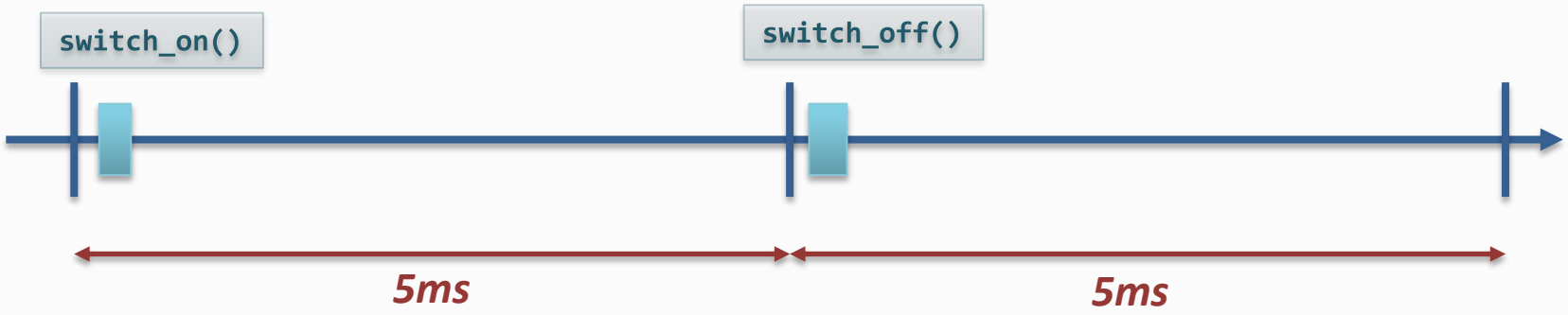
Elementary Action with Preemptions



Elementary Action – Simple Example

- **Let's make a LED blink, shall we?**

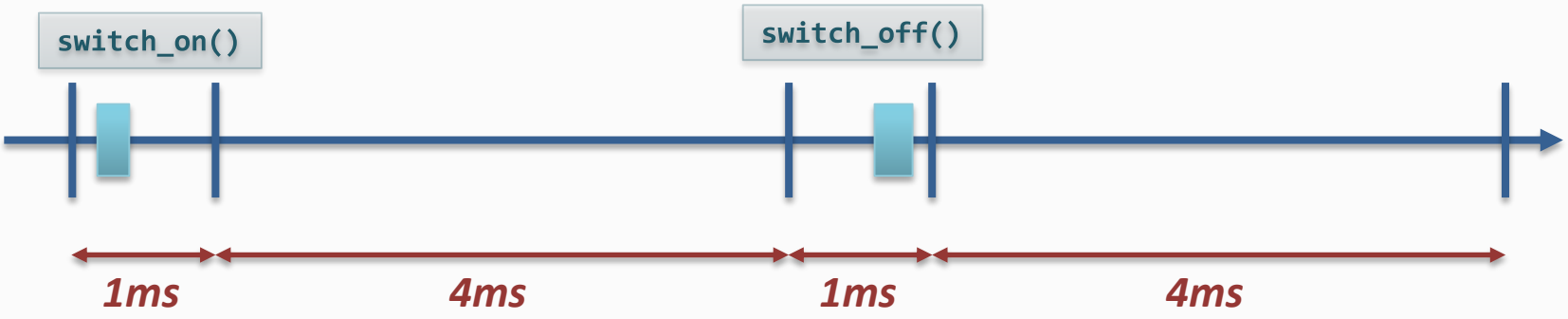
- **Specs:**
  - **Period = 10ms**
  - **Duty Cycle (ON) = .5**



Elementary Action – Simple Example?

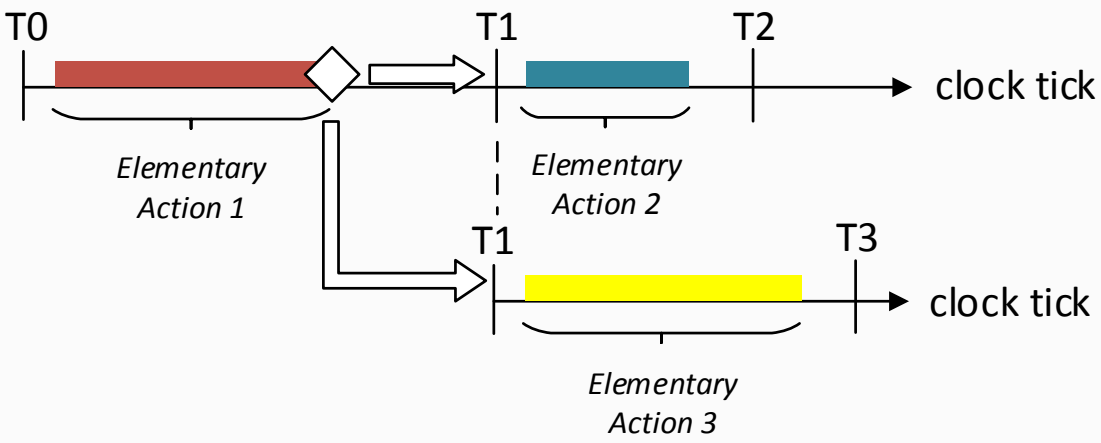
- **Let's make a LED blink, shall we?**

- **Specs:**
  - Period = 10ms
  - Duty Cycle (ON) between 0.4 and 0.6
  - Switching jitter <= 1ms



A bit further than Periodic Tasks

```
compute X
compute Y
synchronize on T1
if (condition)
  compute Z
  synchronize on T2
else
  compute Z'
  synchronize on T3
```



What PsyC looks like

- **PsyC is an implementation of the Psy programming model**

- **Extension of the C language: additional syntax to**
  - **declare agents**
  - **express temporal constraints and describe the E.As of each agent**
  - **declare communication channels**
  - **…and much more**

- **Able to call any external function linked with the application:**
  - **C/C++ (*no dyn. allocation)***
  - **Ada**
  - **…**

What PsyC looks like

- **LED-blinking example: *blinker.psy***

```
#include <asterios.h>

clock c_base = ast_realtime_ms;
clock c_half_period = 5*c_base;

agent Blinker(uses realtime, starttime=10, defaultclock c_base)
{
  body start
  {
    switch_on();
    advance 1;
    /* do nothing */
    advance 1 with c_half_period;

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

What Asterios IDE Looks Like

Asterios - Hello World TV

Projet | Développement | Exécution | Dimensionnement

Hello World TV

Cacher les fichiers non sélectionn

ksim-rsf-default

- Hello World TV
  - ksim-default
  - ksim-rsf-default
  - edit.h
  - edit_ksim.c
  - HelloWorld_TV.psy

```
KSIM : Krono-Safe Simulateur v2016.B
KSIM : licence validée
KSIM : démarré
KSIM : en attente d'événements
KSIM : chargement de la simulation
KSIM : MPU activée
KSIM : trace activée
KSIM : lancement de la simulation
```

HelloWorld_TV.psy

```
11
12 /*-----------------------------------------
13 / COMMUNICATION DEFINITION
14 /-----------------------------------------*/
15
16 temporal char welcome_tv[15]  with c_base;          /* TV declaration */
17
18 /*-----------------------------------------
19 / AGENT 1 : Update the TV
20 /-----------------------------------------*/
21
22 agent ag_producer(uses realtime, starttime = 1, defaultclock c_base)
23 {
24   display
25   {
26     welcome_tv : ag_consumer;                       /* ag_consumer could read the TV */
27   }
28
29   global
30   {
31     char var[15]="0 Hello World";                   /* variable to initialized the TV */
32     int i;
33   }
34
35   body start
36   {
37
38     var[0]++;                                       /* increase the counter [0..9] */
39     if(var[0] > '9') var[0] = '0';
40
41
42     for (i=0;i<15;i++)                              /* copy var in the TV */
43       welcome_tv[i] = var[i];
44
45     edit("ag_producer : %s\n",welcome_tv);
46
47     advance 1;
48
49   }
50 }
51
```

What Asterios IDE Looks Like

Asterios - Hello World TV

Projet | Développement | Exécution | Dimensionnement

Hello World TV

Cacher les fichiers non sélectionn

ksim-rsf-default

- Hello World TV
  - ksim-default
  - ksim-rsf-default
  - edit.h
  - edit_ksim.c
  - HelloWorld_TV.psy

ns | Coeur 0 | RSF 0 | – Agents – | Comparer avec :

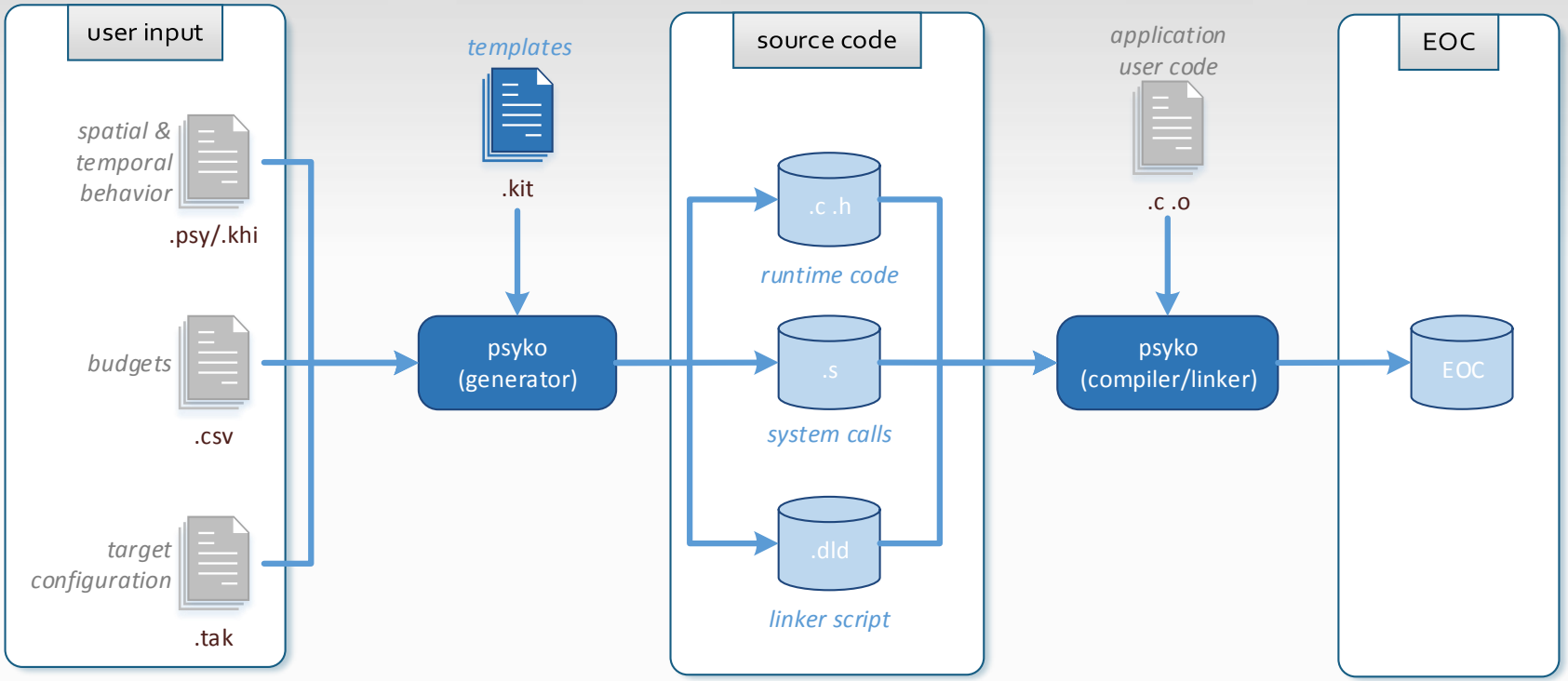
Configuration de compilation de ksim-rsf-default

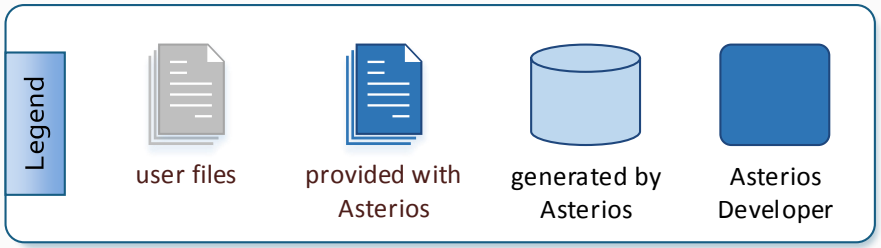
Agents	Budgets	Valeurs	Origine	Coeurs
ag_consu...	B2	400000000	Psyko	0
ag_consu...	B3	400000000	Psyko	0
ag_produ...	B2	400000000	Psyko	0
ag_produ...	B3	400000000	Psyko	0

HelloWorld_TV.psy

```
46
47      advance 1;
48
49    }
50  }
51
52  /*----------------------------------------
53  / AGENT 2 : display the TV
54  /----------------------------------------*/
55
56  agent ag_consumer(uses realtime, starttime = 2, defaultclock c_
57  {
58    consult
59    {
60      1 $ welcome_tv;                                  /* read t
61    }
62
63    body start
64    {
65
66      edit("ag_consumer : %s\n", $[0]welcome_tv);      /* display
67
68      advance 1;
69    }
70  }
71
72
73
74
75
76
```

Code Generation Tools





KRONO-SAFE
Safe design in real-time

Communications Determinism

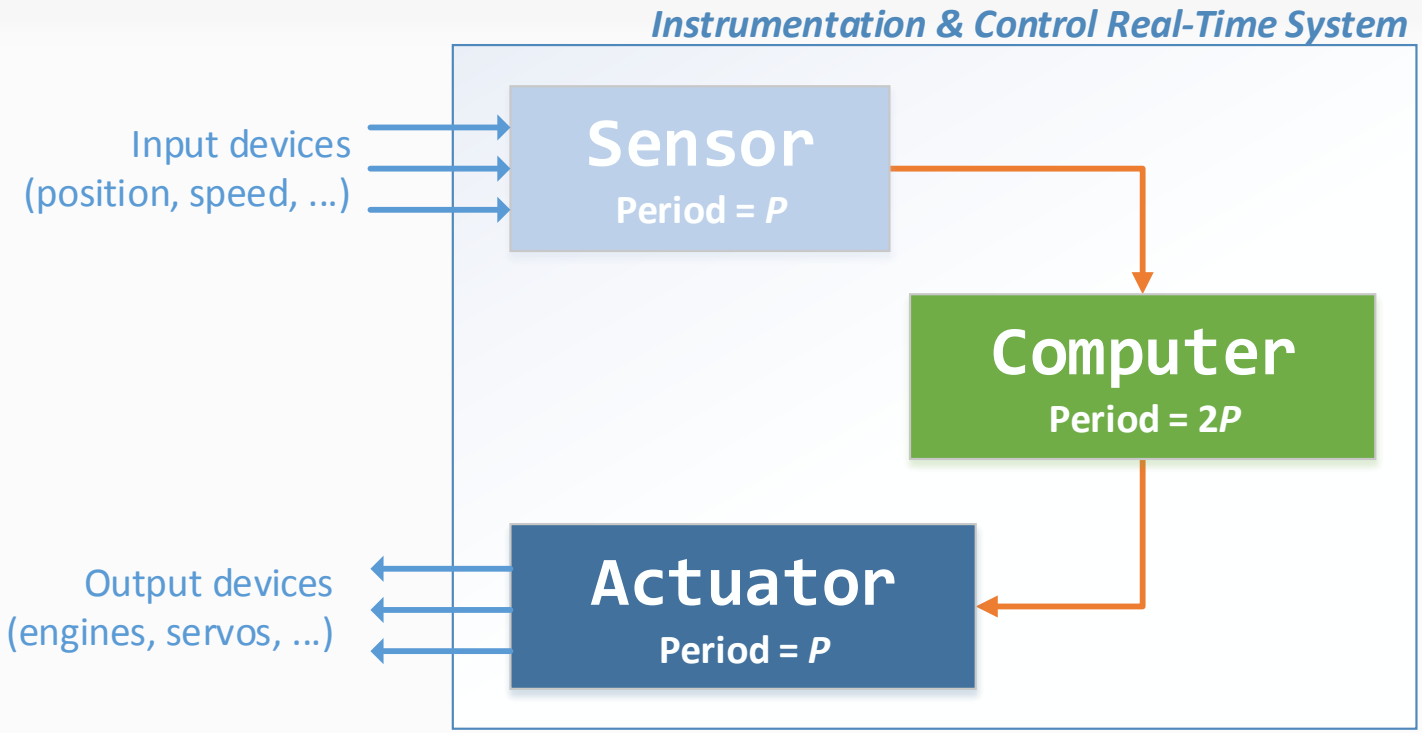
- **Key Advantages of Determinism**
  - For system validation
    - Test scenarios are predictable and their results reproducible
  - For error blaming & containment
  - For scalability and modularity (strong determinism)
    - Different implementations have identical results, except for the sizing

- **Quite the Eldorado with real-time systems**
  - Yet a very shallow definition: same inputs implies same outputs?

- **In Asterios: the *observable* input & outputs of each task shall remain the same, from one execution to another**

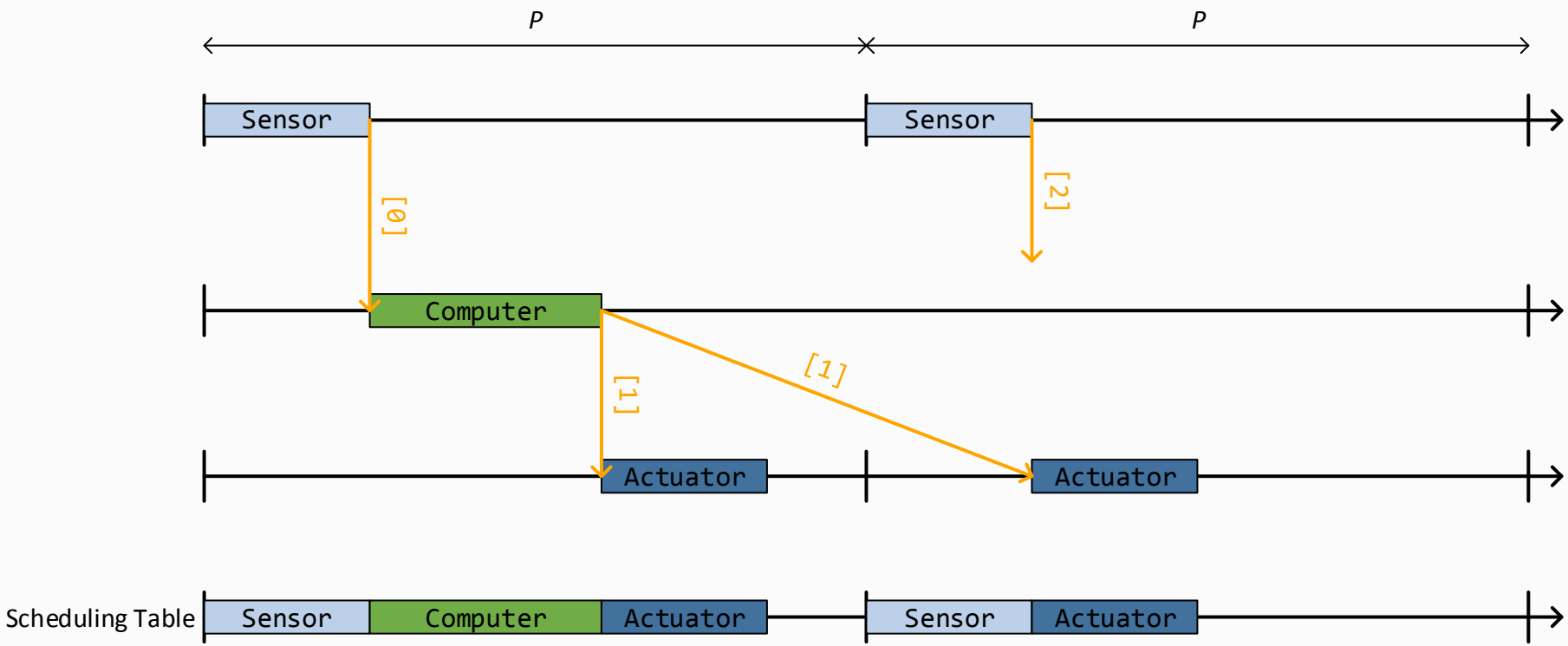
The problem with parallelism

- **Consider a simple I&C application**



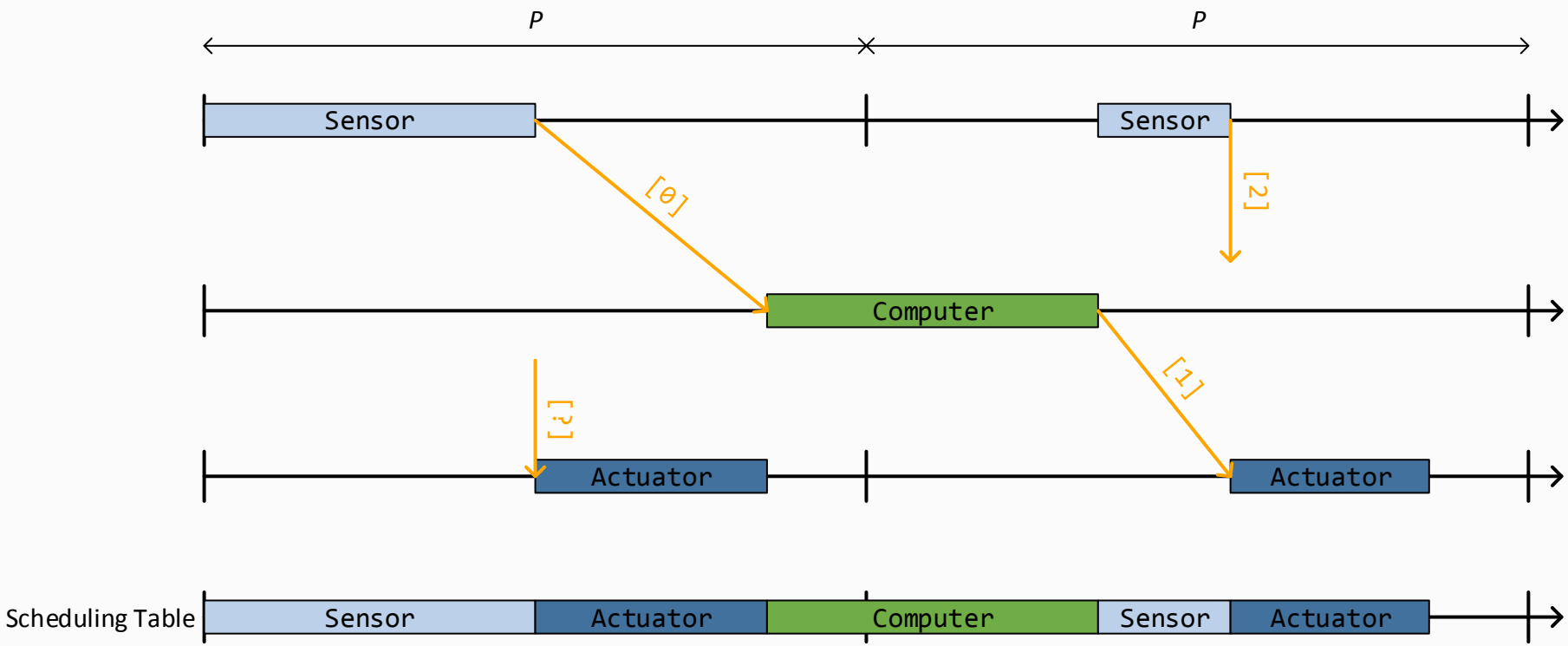
The problem with parallelism

- **Communication through a FIFO**
  - assume consistency is guaranteed



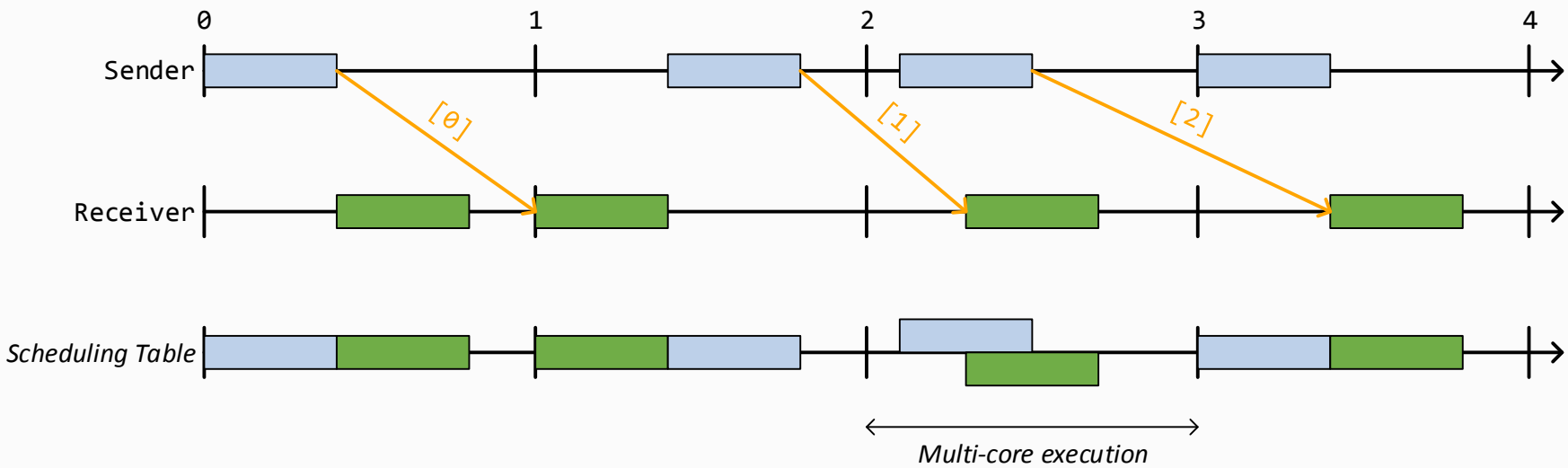
The problem with parallelism

- **What if Sensor takes more time to execute on the first run?**

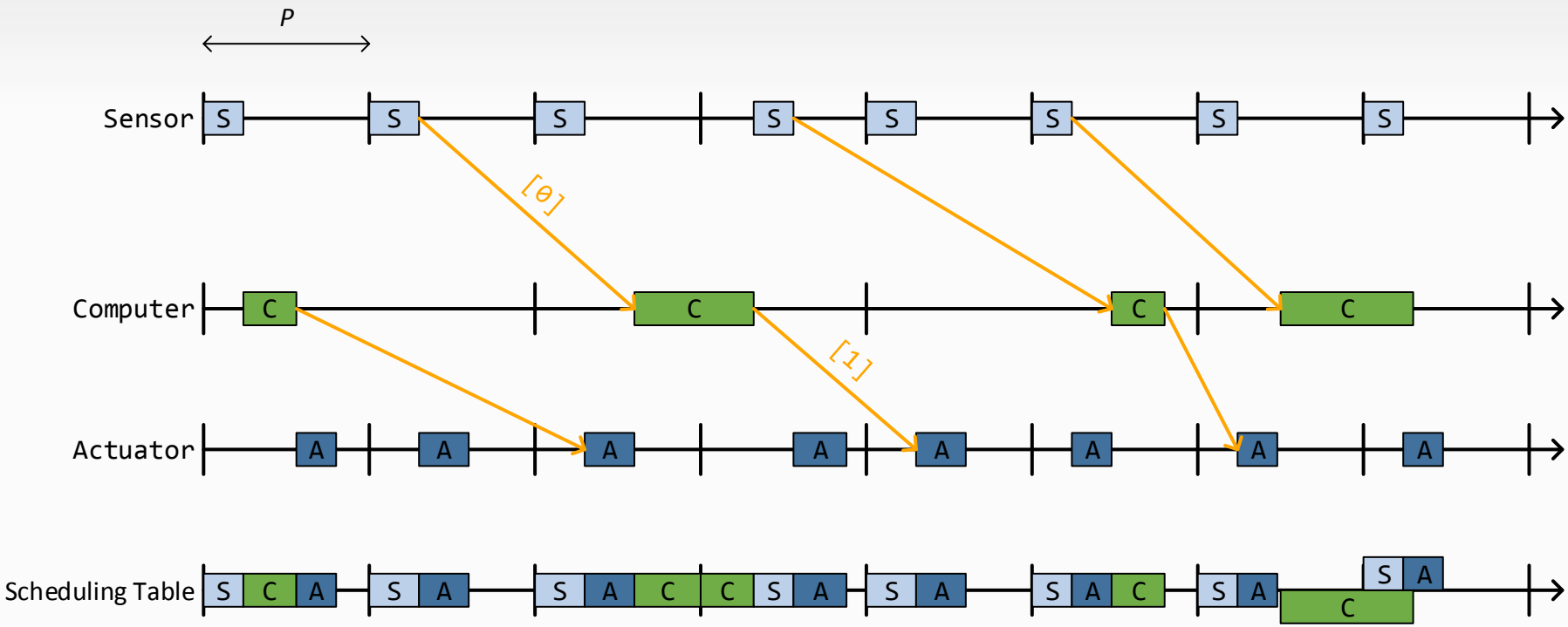


The Visibility Principle

- **Any data exchanged between agents is *timestamped***
  - **This timestamp is also called the *visibility date* of the sample**
- **Visibility Principle:**
  - **The visibility date of a sample is always greater than the deadline of the E.A. that produced the sample;**
  - **The E.A of an agent can only read a sample if its visibility date is before the release date of the E.A.**



Application to the I&C loop



Communication in PsyC

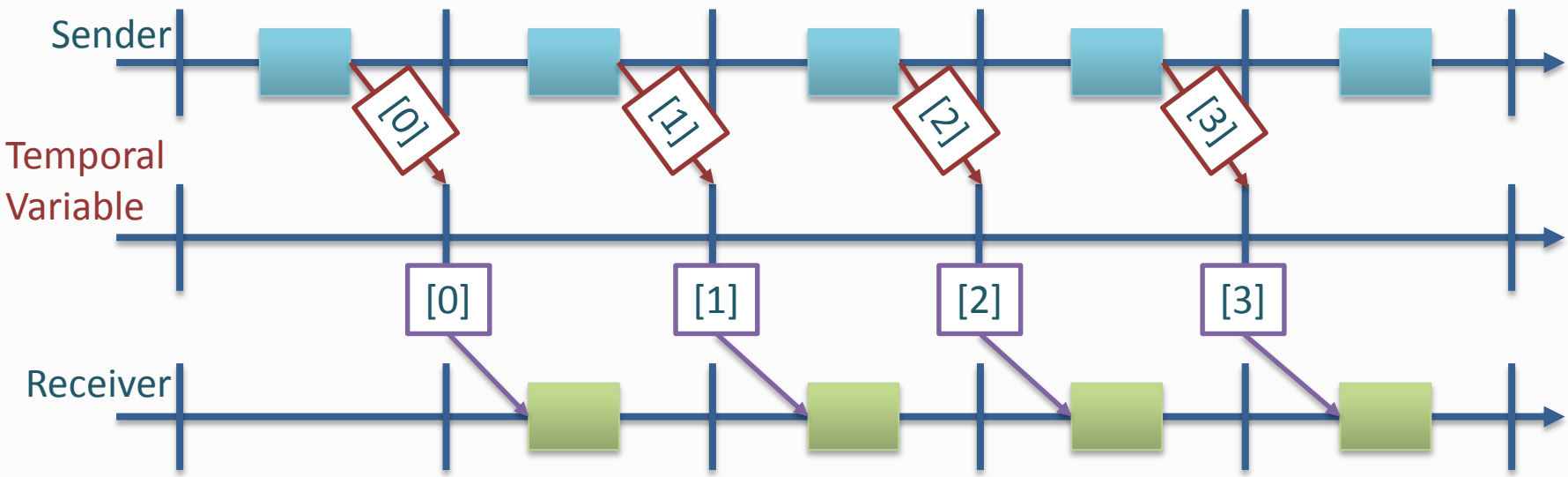
- **Implemented in PsyC through 2 types of channels:**
  - **Temporal Variables (*1 to n, periodic messages*)**
  - **Message Boxes (*n to 1, sporadic sending*)**

- **These mechanisms ensure:**
  - **data consistency**
  - **timestamping & visibility principle → determinism**
  - **lock-free, pre-emptible and efficient implementation**

- **Comm. buffer sizes are computed automatically by the PsyC compiler**

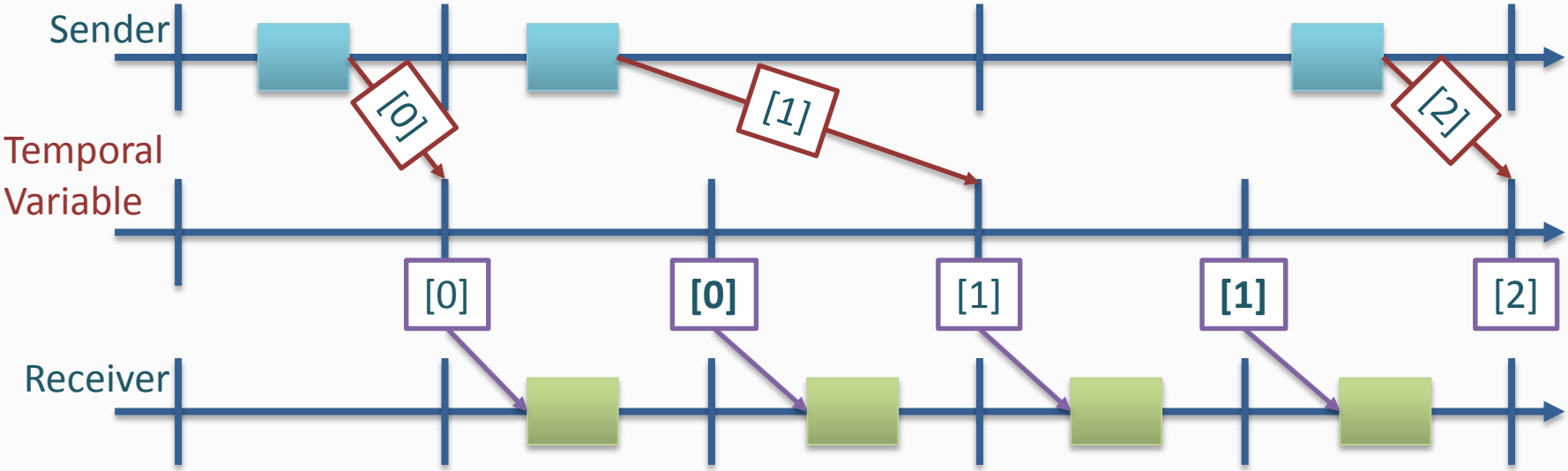
Temporal Variables in PsyC

- **1-to-N communication**
- **Periodic Sampling**
  - **the period is completely independent from the sender and the receiver(s)**



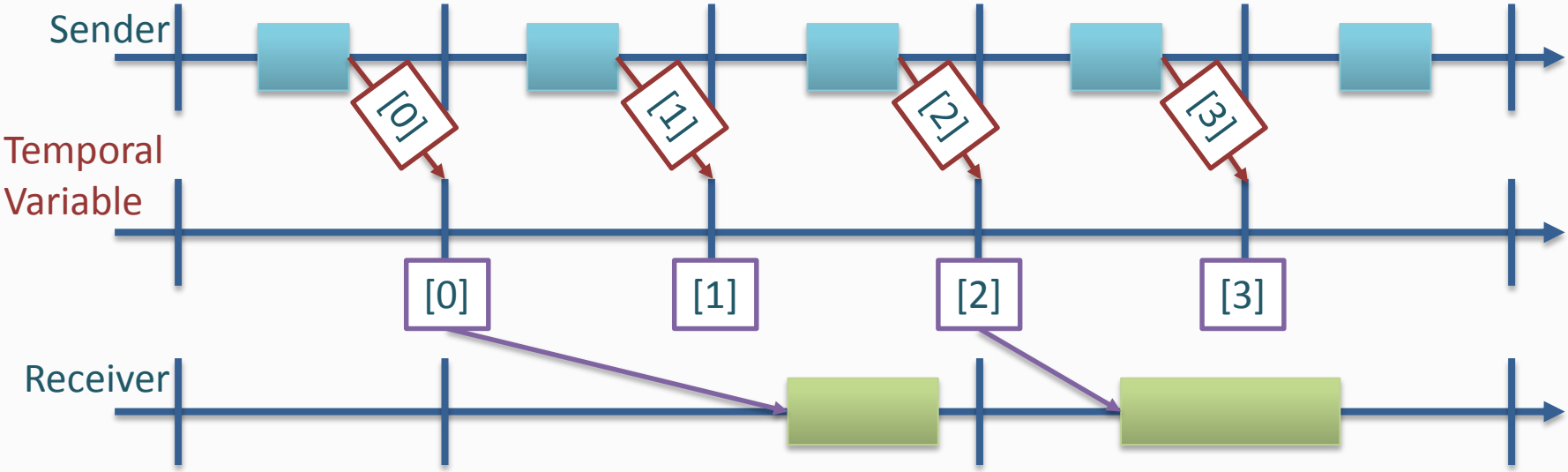
Temporal Variables in PsyC

- **Illustration:**
  - Slow Sender
  - in-phase Receiver



Temporal Variables in PsyC

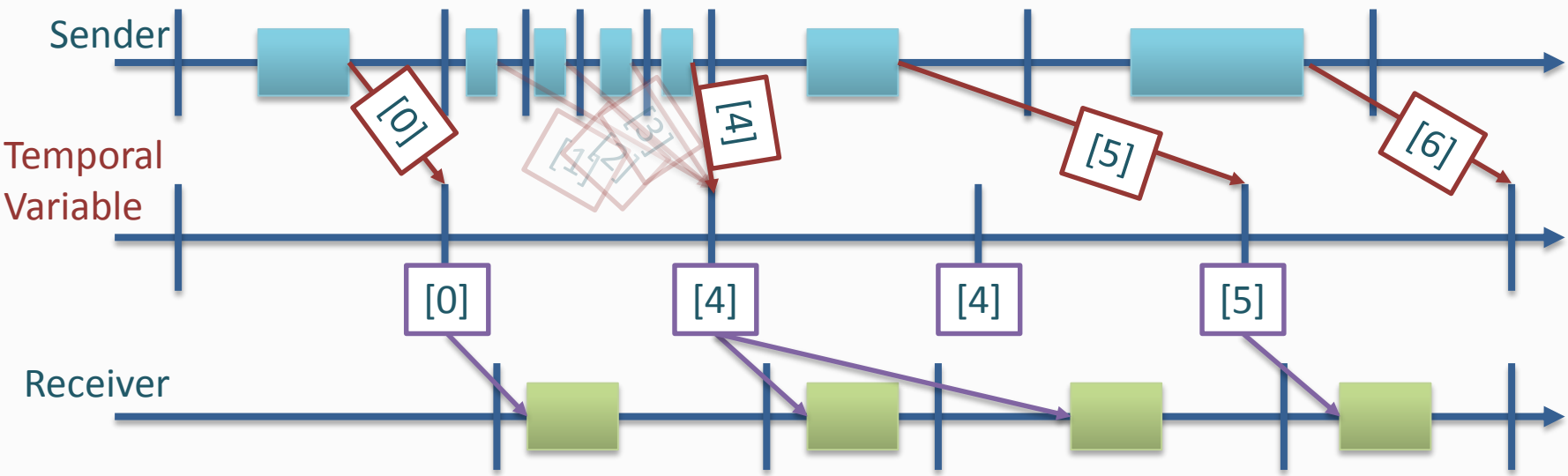
- **Illustration:**
  - in-phase Sender
  - slow Receiver



Temporal Variables in PsyC

- **Illustration:**
  - slow/fast/de-phased Sender and Receiver

- **Nota: this kind of temporal behavior *can* be expressed in PsyC**



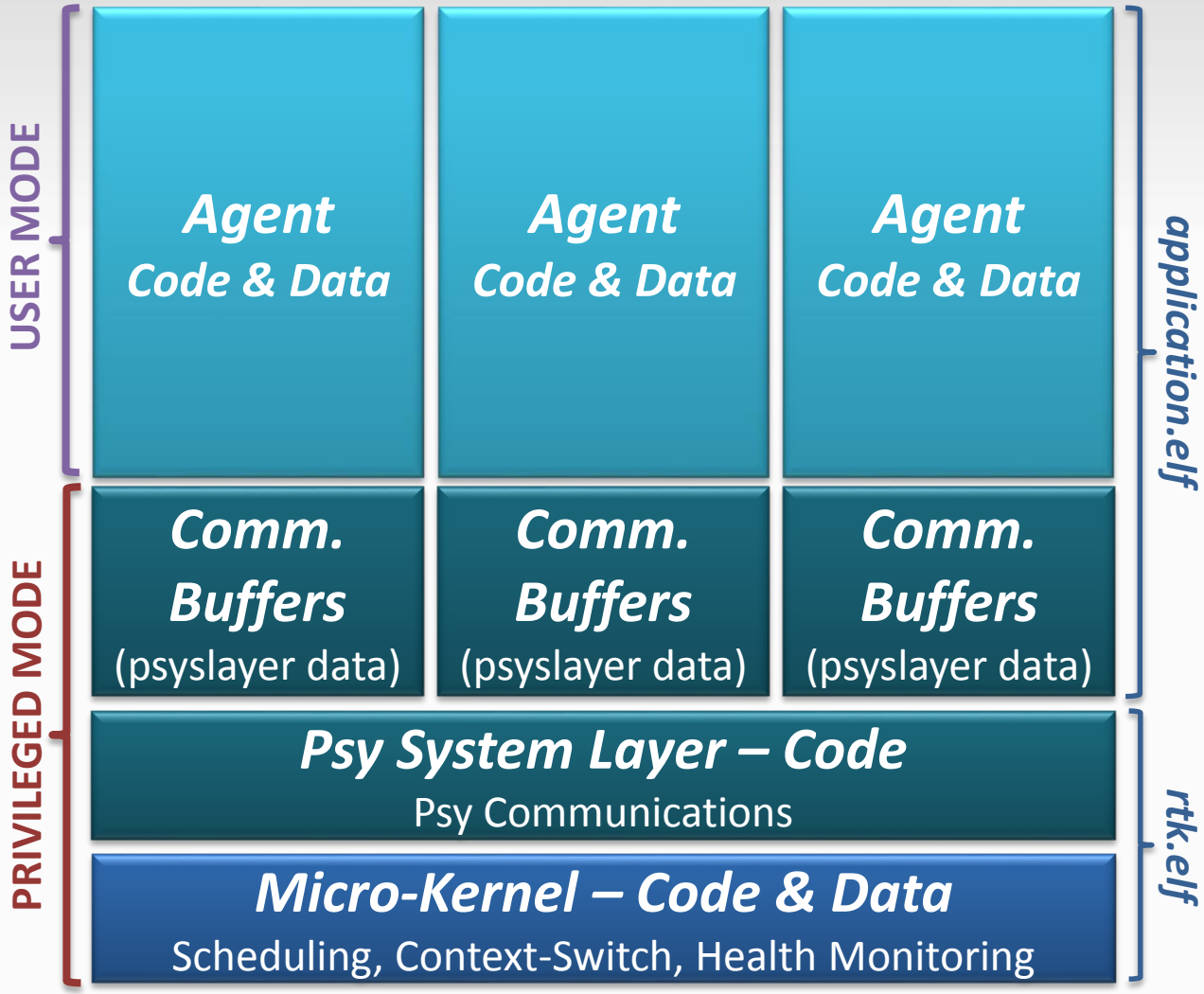
Leveraging End-to-End Delays

- **Driver tasks**
  - Fast-paced tasks with higher privileges for I/O ops
  - Capable of transferring data to and from an agent *within* its Elementary Action

- **Generalization: Null-Latency Communication**
  - Advanced feature, coming up in the next release of Asterios

- **In both cases, communication determinism is preserved (!)**

Execution Model: Memory Protection



- Agents run in non-privileged mode
- No shared data whatsoever (*no hidden comm. channel*)

- Psyslayer: com. services implemented by the RTK
- For each agent, the compiler has generated the comm. buffers to be used by the psyslayer

---

- Micro-kernel runs in privileged mode

Agents in the Execution Model

- **Autonomous parallel execution unit**
  - Sequential execution
- **Spatial and temporal protection unit**
  - Also error confinement unit
- **Have their own execution context :**
  - User memory (data, code, stack)
  - CPU time
- **Can exchange data with other agents:**
  - Only with communications means defined in PsyC
  - No shared memory

Psyslayer in the Execution Model

- **Essentially implements the communication channels**

- **System service: the psyslayer is always executed *on behalf of an agent***
  - **System code that only runs on resources of the agent being served:**
    - **memory (com. buffers)**
    - **CPU time**
  - **Wait-free and pre-emptible**

Cadence and Scheduling

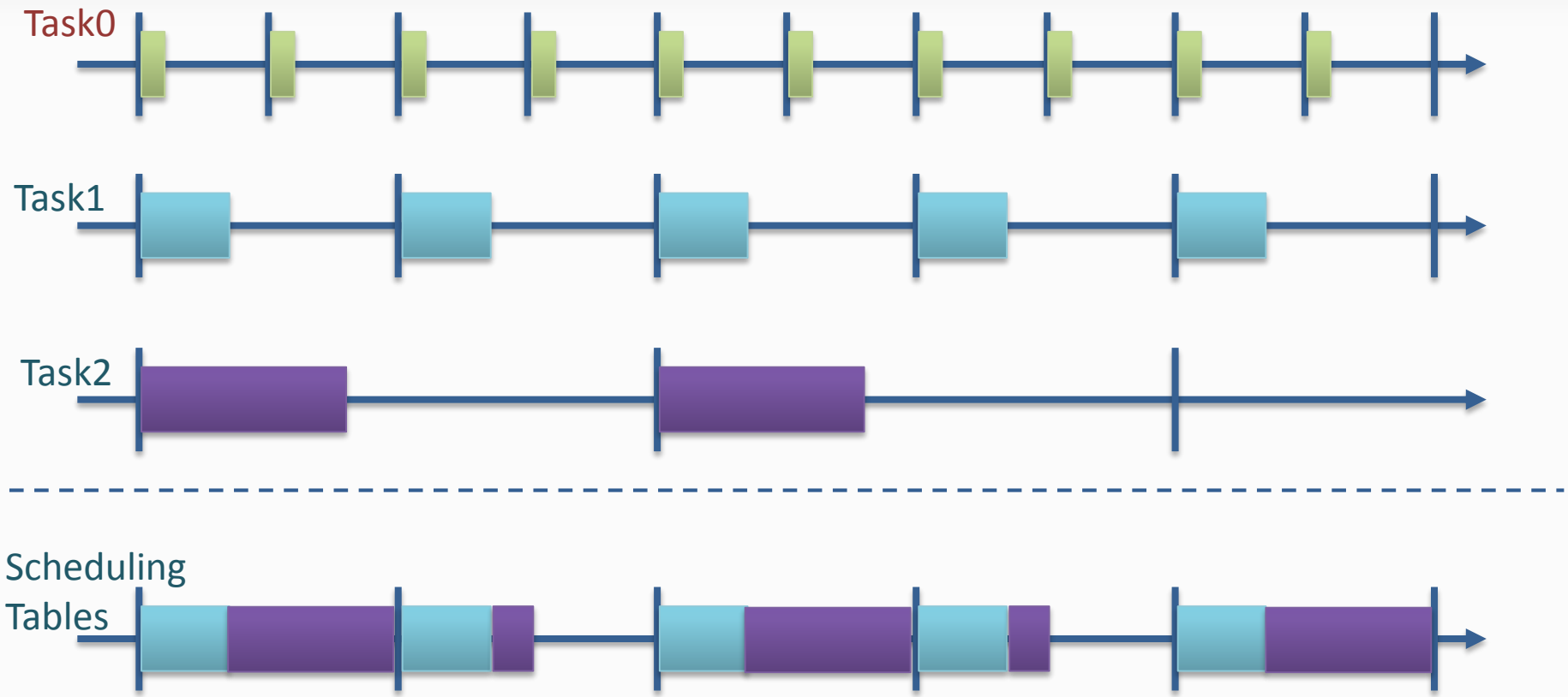
- **The Psy programming model lets the developer focus on its functional needs**
  - **"Cadence": span of the Elementary Actions**
  - **Set for each agent, *independently***
  - **Remains a basic modular brick, insensitive to**
    - **adding a new task to the system**
    - **upgrading to a faster CPU**
    - **…**

- **In Psy, scheduling does not require *any* configuration by the developer**
  - **no priorities**
  - **no locks**

- **Actually, the Psy programming model can be used with any scheduling algorithm**
  - **…but for certification purposes, static scheduling is often necessary**

Cadence and Scheduling

- **Typical Example where scheduling should *not* be an issue**



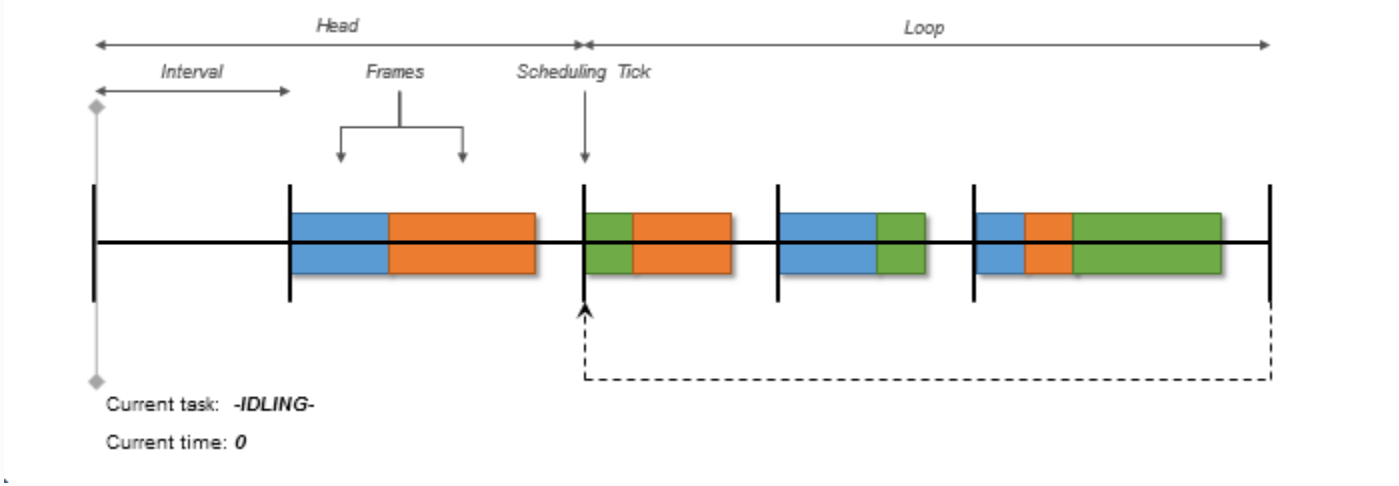
Scheduling a Psy Application

- **The Psy programming model is independent from the chosen scheduling policy**
  - **as long as all E.As are fully executed *within their boundaries***
  - **determinism is enforced, regardless of preemptions or multi-core execution**

- **Our implementation of the Psy programming model (*the PsyC compiler toolchain and the kernel*) offers an automatic scheduler**
  - **based on timing constraints (E.A. spans) and CPU budgets**
  - **automatic generation of a static and periodic scheduling plan (*patented execution process*)**
  - **very low online overhead: scheduling in O(1)**
  - **seamless transition from mono- to multi-core**

Repetitive Sequence of Frames

- **Periodic Scheduling Plan**
- **Ensures that *all* CPU budget requirements are satisfied**
  - **for each agent**
  - **for any execution path followed by this agent**

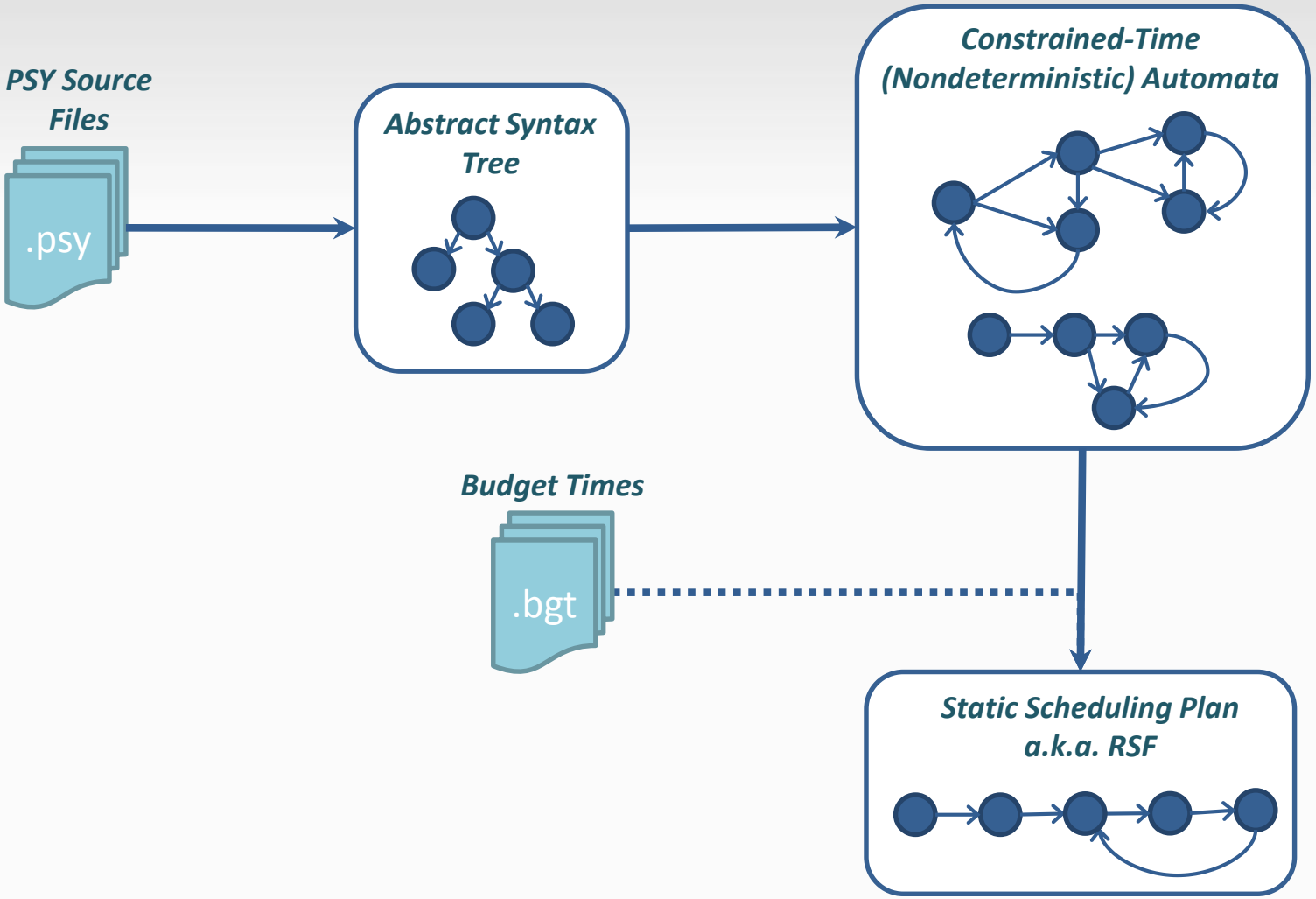


Repetitive Sequence of Frames

- **Good Properties for Certification**
  - enables to tell in advance which task will be executed at any moment in time
  - know in advance the maximum number of preemptions that can occur
    - can be taken into account by giving the agent additional CPU budget

- **Enables Modular Integration**
  - can save CPU time for future tasks to be inserted
  - flexible: can be "*folded*" to any temporal length to adapt the memory footprint

Quick Peek: How to build a RSF



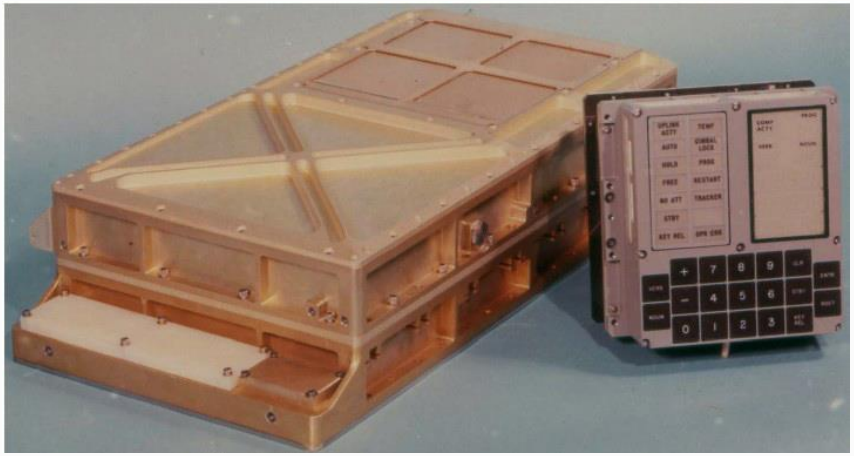
Conclusion

- **Real-Time Programming Model suited for Safety-Critical, Modular Applications**
  - Multi-Task, Multi-CPU
  - Deterministic Communication with auto Buffer sizing
  - High Expressiveness (complex temporal behaviors)

- **Dedicated Programming Language**
  - Source-to-source compiler
  - Automatic generation of static scheduling plan

- **Bare Metal RTOS with micro-kernel architecture**
  - Constant time micro-kernel execution
  - Pre-emptible, wait-free communication services

The First Embedded Real-Time System



- **Apollo Guidance Computer**
  - 8 kb of erasable magnetic memory
  - 64 kb of read-only rope memory
- **RTOS**
  - multi-tasking with fixed priorities
  - preemptive scheduler with error handling
  - virtual machine with pseudo-instructions for complex ops
- **1st Lunar Landing on *July 20th, 1969***

Source: Wikipedia

Source Version Control before Git

Margaret Hamilton, Director of the Software Engineering Division at MIT

1969, standing next to the listings of the software for the Command & Lunar Modules

Source: Wikipedia