A Multi-Rate Extension of the ForeC Precision Timed Programming Language for Multi-Cores

Alain Girault — UGA Univ. Grenoble Alpes, Inria

Nicolas Hili — IRT Saint Exupery

Eric Jenn — IRT Saint Exupery, Thales

Eugene Yip — Otto-Friedrich-Universität Bamberg, SWT tomorrow's software technologies

Synchron 2019, Aussois, France

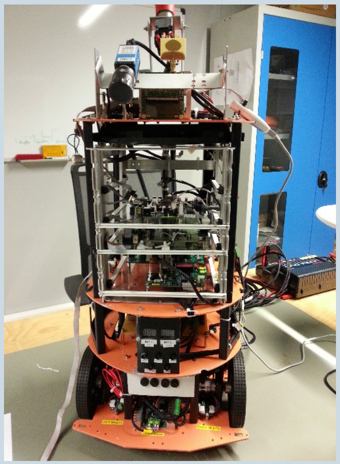
A. Girault, N. Hili, E. Jenn, and E. Yip. *A Multi-Rate Precision Timed Programming Language for Multi-Cores*. Forum for Specification and Design Languages (FDL), United Kingdom, 2019

Outline

- Background and motivation
- Multi-rate ForeC synchronous language
- Bare-metal implementation
- Conclusions and Outlook

Cyber-Physical and Real-Time Systems

Embedded System



Analyses and decides

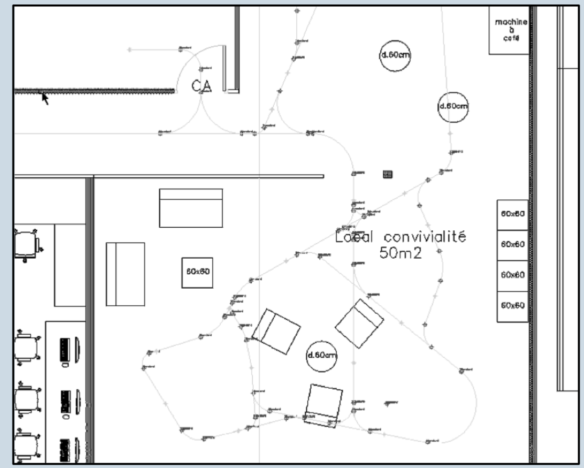
Sense

- odometry
- orientation
- obstacles

Actuate

- motors
- speakers
- lights

Physical Environment



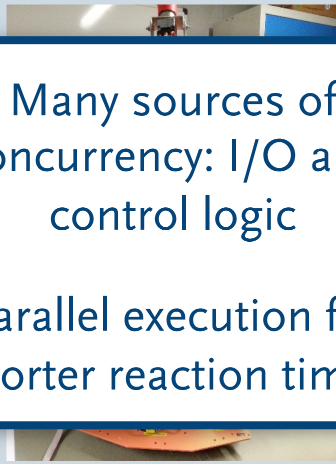
Evolves continuously

Correct outputs required at correct times!

Otherwise: collisions, erratic behaviour, endangerment of life, failure to complete mission, ...

Cyber-Physical and Real-Time Systems

Embedded System



Many sources of concurrency: I/O and control logic

Parallel execution for shorter reaction times

Analyses and decides

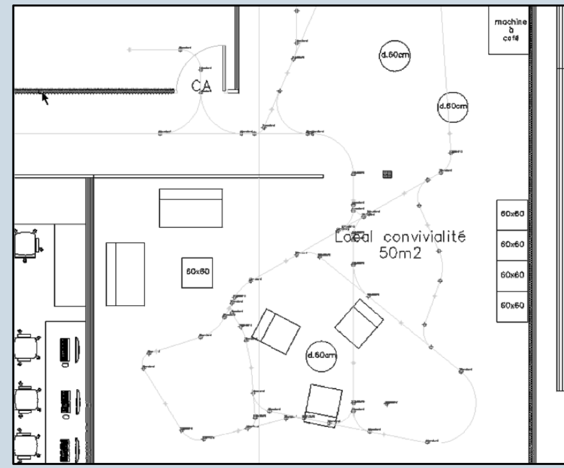
Sense

- odometry
- orientation
- obstacles

Actuate

- motors
- speakers
- lights

Physical Environment

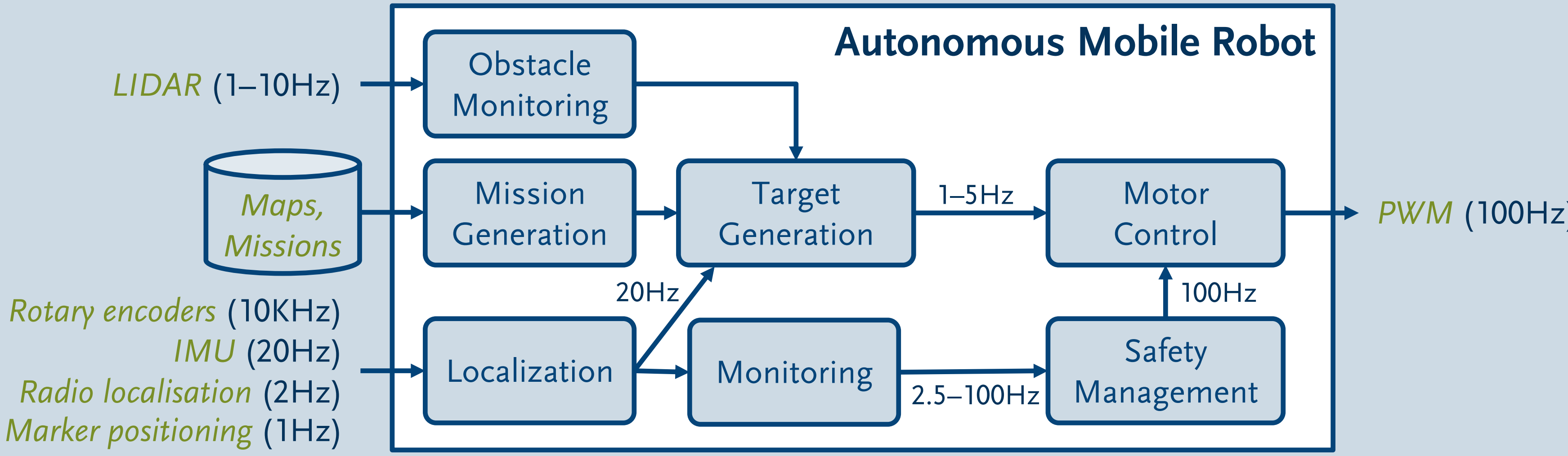


Evolves continuously

Correct outputs required at correct times!

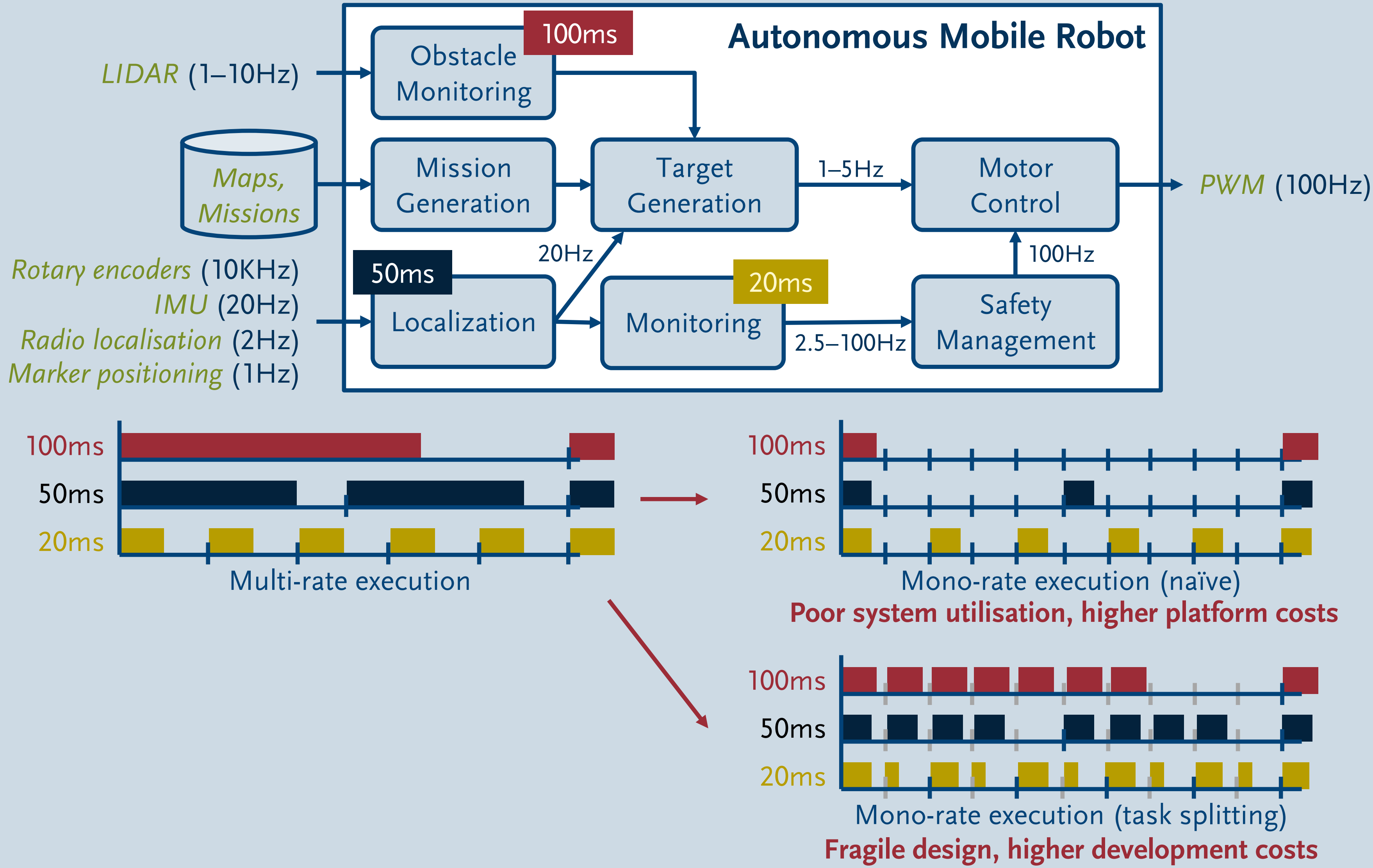
Otherwise: collisions, erratic behaviour, endangerment of life, failure to complete mission, ...

Real-Time and Multi-Rate Concerns



- System interfaces with *different aspects* of its environment, each *evolving at their own pace*
- Rates affect, e.g., the system's *performance* and *compliance* to safety requirements

Real-Time and Multi-Rate Concerns



ForeC Language

- C-based, multi-threaded, mono-rate, synchronous language for embedded multi-cores

- Safety-critical/formalised *subset* of C
  - E.g., MISRA-C, Power of 10, Clight, or Cyclone
  - E.g., No unbounded loops and recursion, pointer reassignments, `goto`s, and dynamic memory allocations
- Minimal set of *synchronous constructs*
  - `in`, `out`, `env`, and `shared` variables
  - `par(`*st*`,`*st*`)`, `pause`, `abort` *st* `when (`*cond*`)`
  - Formally defined by *small-step* semantic rules

ForeC Timing Model

- *Synchrony hypothesis*
  - Sampling, computing, and emitting all *take zero time*
  - Verified by WCRT analysis
- Inputs sampled at start of tick, outputs emitted at end of tick
  - Timing behaviour reminiscent of *Logical Execution Time* (LET)
  - Threads execute in isolation

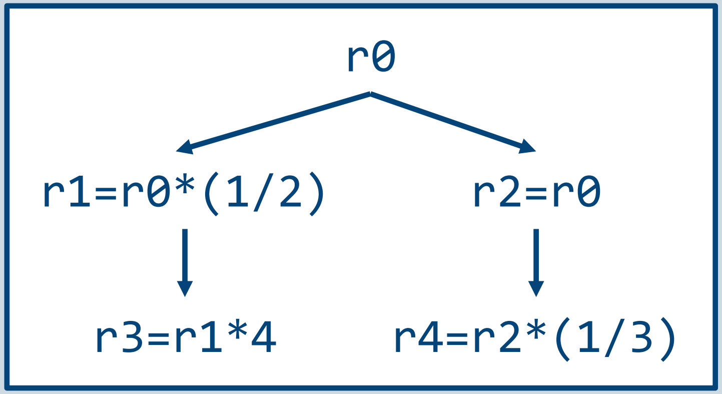
MULTI-RATE ForeC LANGUAGE

- Defining thread rates
- Multi-rate synchronisation, communication, and preemption
- Maintain backwards compatibility with mono-rate ForeC

Defining Thread Rates

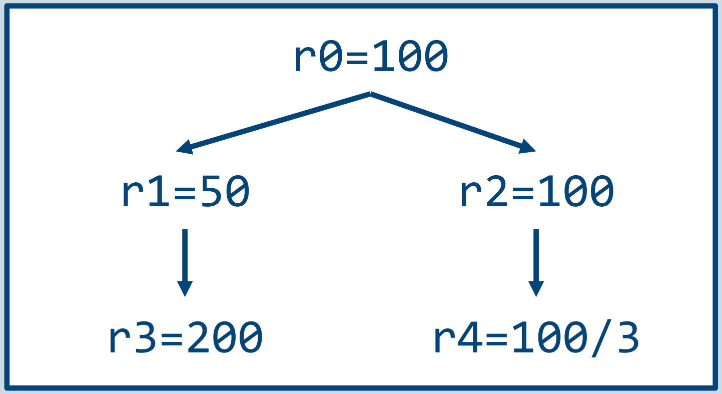
- *Logical rates* defined in the ForeC program file (`*.forec`)

```
const rate r0;
const rate r1=r0*(1/2),
           r2=r0,
           r3=r1*4,
           r4=r2*(1/3);
```



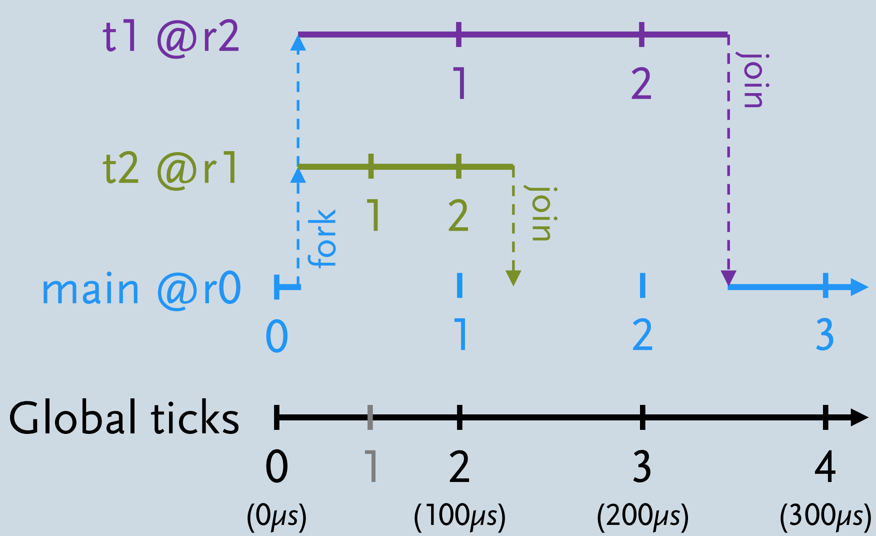
- *Concrete rates* defined in the ForeC header file (`*.foreh`)

```
// Rate in microseconds (μs)
const rate r0=100

architecture: multipret
core0: t1
core1: t2
```



Multi-Rate Synchronisation

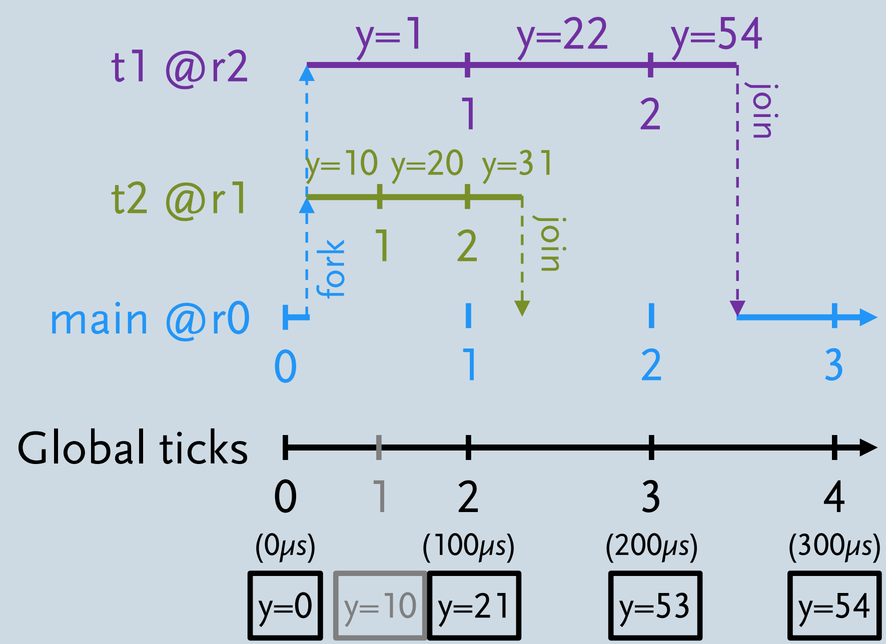
```
const rate r0;           // 100µs
const rate r1=r0*(1/2),  //  50µs
           r2=r0;        // 100µs

env in int x=0;
env out shared int y=0 combine mod with +;

thread t1(void) @r2 {
  y+=1; pause;
  y+=1; pause;
  y+=1;
}

thread t2(void) @r1 {
  y+=10; pause;
  y+=10; pause;
  y+=10;
}

void main(void) @r0 {
  par(t1(), t2());
}
```



- Total global tick: All threads end their tick together
- Partial global tick: Some threads end their tick together

Multi-Rate Communication

```
const rate r0;            // 100µs
const rate r1=r0*(1/2), //  50µs
           r2=r0;         // 100µs

env in int x=0;
env out shared int y=0 combine mod with +;

thread t1(void) @r2 {
  y+=1; pause;
  y+=1; pause;
  y+=1;
}

thread t2(void) @r1 {
  y+=10; pause;
  y+=10; pause;
  y+=10;
}

void main(void) @r0 {
  par(t1(), t2());
}
```



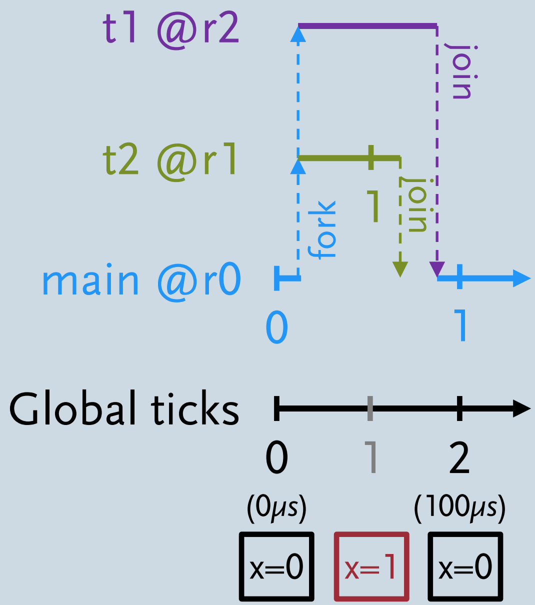
- Threads copy the shared variable and modify in isolation
- Copies are combined when the tick ends

Combine Functions and Policies

- *Thread isolation* for thread-local reasoning
  - Reduces the frequency of inter-core synchronisations
- Combine functions shall be *commutative* and *associative*
  - `plus( plus(th1,th2) , plus(th3,th4) )`
  - Parallelisation and determinism
- *Policies* to control what copies are combined
  - `all`, `mod`, and `new`
- *Race conditions* are avoided

Multi-Rate Preemption

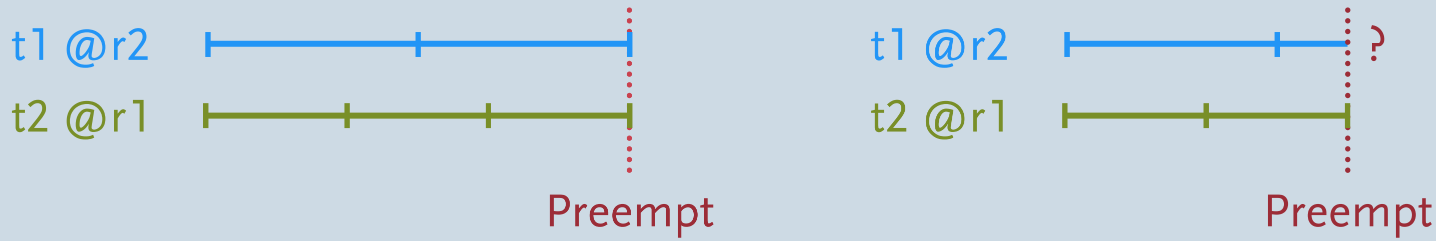
```
const rate r0;           // 100μs
const rate r1=r0*(1/2),  //  50μs
           r2=r0;        // 100μs

env in int x=0;
env out shared int y=0 combine mod with +;

thread t1(void) @r2 {
  y+=1; pause;
  y+=1; pause;
  y+=1;
}

thread t2(void) @r1 {
  y+=10; pause;
  y+=10; pause;
  y+=10;
}

void main(void) @r0 {
  weak abort {
    par(t1(), t2());
  } when (x==1);
}
```



- A weak abort allows its body to execute one last time
- The immediate checking of condition is possible

Thoughts on Strong Preemption

- Possible behaviour
  - Preemption is checked at the start of every (total/partial) global tick
  - When triggered, all enclosed threads are terminated immediately
  - Execution continues after the `abort` statement



- **Problem: Some threads may already be in their local tick!**
  - Rollback threads' executions (e.g., shared variables, outputs)?
  - Wrong assumption about strong preemption when their tick began?
  - *Strong preemptions allowed when enclosing threads are mono-rate*
- **Want performant and statically analysable implementations!**

Related Multi-Rate Synchronous Languages

<table>
<tr><th colspan="2">Language</th><th>Style</th><th>Multi-Rate Multi-Clock</th><th>Communication</th><th>Code Generation</th><th>Timing Analysable</th></tr>
<tr><td>Esterel</td><td rowspan="2">Founding synchronous languages</td><td>Imperative</td><td>Partial</td><td rowspan="3">Instantaneous signals</td><td rowspan="2">Sequential (bare-metal)</td><td rowspan="2">✓</td></tr>
<tr><td>Lustre</td><td rowspan="2">Dataflow</td><td rowspan="2">✓</td></tr>
<tr><td>Prelude</td><td>Lustre with multi-rate flows</td><td>Tasks (OS)</td><td>OS dependent</td></tr>
<tr><td>PRET-C</td><td>Esterel inspired constructs</td><td rowspan="2">Imperative, C-based</td><td>✗</td><td>Seq. shared variables</td><td>Sequential (bare-metal)</td><td rowspan="3">✓
(if bare-metal)</td></tr>
<tr><td>Multi-rate ForeC</td><td>Designed to exploit multi-cores</td><td>Partial</td><td>Parallel shared variables</td><td>Threads (OS or bare-metal)</td></tr>
<tr><td>Simulink</td><td>Block diagram for dynamical systems</td><td>Dataflow</td><td rowspan="2">✓</td><td>Signals</td><td>Sequential (bare-metal), Tasks (OS)</td></tr>
<tr><td>Giotto</td><td>Time-triggered, logical execution time</td><td>Coordination</td><td>Delayed signals</td><td>Tasks (OS)</td><td>✓
(Embedded machine)</td></tr>
</table>

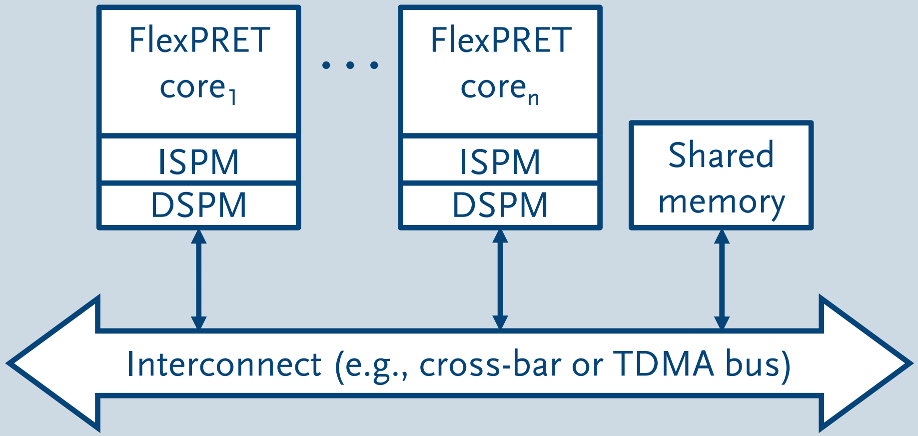
- **Multi-rate/-clock is a well established concept**
- **Inclusion in ForeC improves its practicality and usability for industrial acceptance**

BARE-METAL IMPLEMENTATION

- MultiPRET
- Compilation

MultiPRET Processor

- *FlexPRET* cores connected to shared memory via cross-bar
  - *RISC-V* core designed according to *PREcision Timed (PRET) philosophy*
  - Multi-threaded pipeline, scratchpad memories, PRET instructions
  - Instructions with *repeatable execution times*
- Timing instructions
  - `get_time`
  - `set_compare`
  - `delay_until`
  - `delay_until_periodic`
  - `interrupt_on_expire`
  - `deactivate_exception`



- *Multi-rate ForeC as a PRET programming language*
  - Programmers control physical timing behaviour via thread rates

Multi-Rate ForeC Compilation for MultiPRET

- *Bootup* routine for each core
  - Initialise task scheduler (master core executes the `main` thread)
  - Synchronise to initiate the first global tick
- *Scheduler* routines
  - Sample inputs, compute the global ticks that threads participate in
  - Manage the forking/joining of threads
  - Combine the shared variables of participating threads, emit outputs
- *Time-triggered execution* via the PRET timing instructions
  - Trigger the execution of threads and scheduling routines at precise times, e.g., at local/global tick boundaries
- Perform *worst-case execution time* (WCET) analysis
  - Verify that all threads will complete their local ticks within their rate

CONCLUSIONS AND OUTLOOK

Conclusions and Outlook

- A real-time system may need to react to *different aspects* of its environment, each at their *own pace*
- Extended *ForeC* with the ability to model *multi-rate activities*
  - C-based, PRET language designed for multi-core execution
  - `rate`, `in`, `out`, `env`, `shared`, `pause`, `par`, and `abort`
- *Bare-metal* implementation feasible for a *MultiPRET processor*

- Explore multi-rate *abort semantics* further
- Generalise *rates with offsets*
- Explore the support for a *forest of rates*
- Define the *formal semantics* of Multi-rate ForeC

References (Multi-Rate)

[BS01] G. Berry and E. Sentovich. *Multiclock Esterel*. IFIP Conference on Correct Hardware Design and Verification Methods, ser. LNCS, vol. 2144. Springer, 2001.

[FBLP08] J. Forget, F. Boniol, D. Lesens, and C. Pagetti. *A Multi-periodic Synchronous Data-flow Language*. HASE, 2008.

[ARG10] S. Andalam, P. S. Roop, and A. Girault. *Predictable Multithreading of Embedded Applications using PRET-C*. MEMOCODE, 2010.

[YRGA16] E. Yip, P. S. Roop, A. Girault, and M. Biglari-Abhari. *Synchronous Deterministic Parallel Programming for Multicores with ForeC: Programming Language, Semantics, and Code Generation*. Inria Research Report RR-8943, 2016.

[MW19] The MathWorks, Inc. *Simulink – Simulation and Model-based Design*. Available online https://www.mathworks.com/products/simulink.html

[HHK01] T. Henzinger, B. Horowitz, and C. Kirsch. *Giotto: A Time-triggered Language for Embedded Programming*. EMSOFT, 2001.

References (MultiPRET and Safe-subset of C)

[HGJ19] N. Hili, A. Girault, and E. Jenn. *Worst-case Reaction Time Optimization on Deterministic Multi-core Architectures with Synchronous Languages*. RTCSA, 2019.

[YRAG13] E. Yip, P. S. Roop, M. Biglari-Abhari, and A. Girault. *Programming and Timing Analysis of Parallel Programs on Multicores*. ACSD, 2013.

[MISRA13] Motor Industry Software Reliability Association. *MISRA-C: 2012: Guidelines for the Use of the C Language in Critical Systems*. HORIBA MIRA Limited, 2013.

[H06] G. J. Holzmann. *The Power of 10: Rules for Developing Safety-Critical Code*. IEEE Computer, vol. 39, no. 6, 2006.

[BL09] S. Blazy and X. Leroy. *Mechanized Semantics for the Clight Subset of the C Language*. Journal of Automated Reasoning, vol. 43, no. 3, Springer, 2009.

[JMG+02] T. Jim, J. G. Morrisett, D. Grossman, M. W. Hicks, J. Cheney, and Y. Wang. *Cyclone: A Safe Dialect of C*. USENIX ATC, 2002.

Thank you!

Questions?