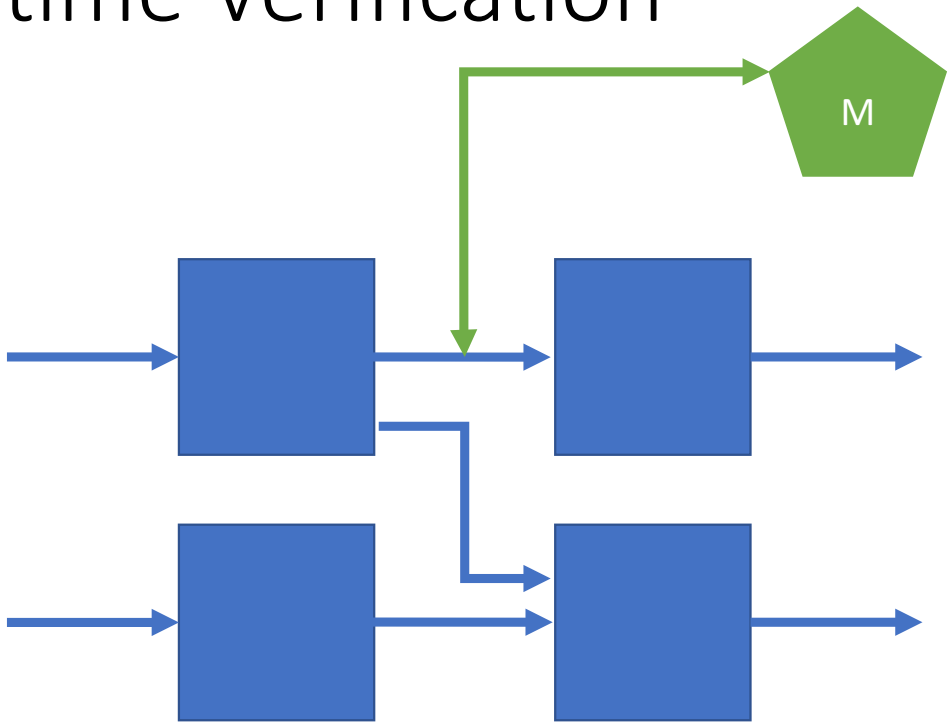
Hardware-based runtime verification with Tessla

Martin Leucker
University of Lübeck

Joint work with Normann Decker, Cesar Sanchez, Torben Scheffel, Malte Schmitz, Daniel Thoma et al.

Runtime Verification



- Monitor analyzes the execution of the system
- Synthesized from high-level specification
- Has to *see* the execution
- Used for finding bugs

Aging Bugs – Mandelbugs – Heisenbugs

- Some bugs only occur
  - after a long execution time
  - under “weird” circumstances

Observation changes the System
Heisenberg Uncertainty Principle

See what’s going on?

- Program annotation?
- Program annotation changes timing

  Change of the underlying system

- Run what you test and test what you run

Code annotation – Program cooperates

Original source code

```
void foo()
{
  bool found=false;
  for (int i=0; (i<100) && (!found); ++i)

  {
    if (i==50) break;


    if (i==20) found=true;


    if (i==30) found=true;



  }
  printf("foo\n");

}
```

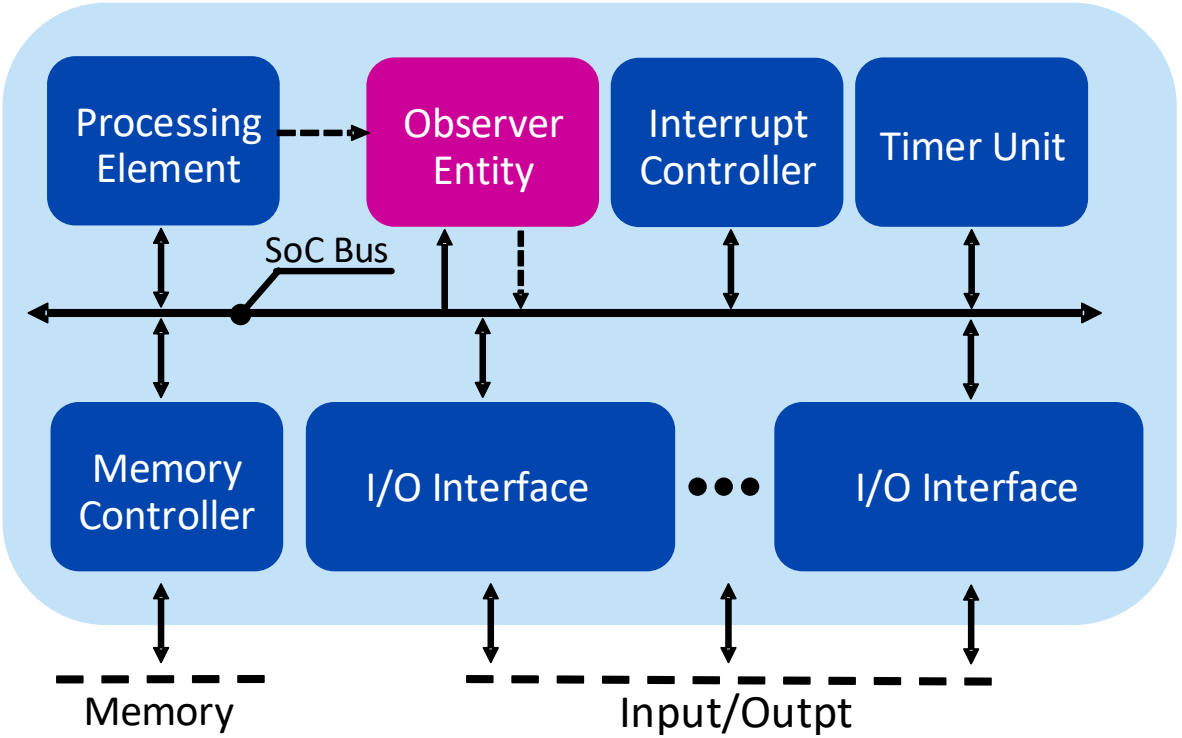
Instrumented source code

```
char inst[15];
void foo()
{
   bool found=false;
   for (int i=0;((i<100)?inst[0]=1:inst[1]=1,0) &&
     ((!found)?inst[2]=1:inst[3]=1,0); ++i)
   {
     if ((i==50?inst[4]=1:inst[5]=1,0))
     { inst[6]=1; break;
     }
     if ((i==20?inst[7]=1:inst[8]=1,0))
     { inst[9]=1; found=true;
     }
     if ((i==30?inst[10]=1:inst[11]=1,0))
     { inst[12]=1; found=true;
     }
     inst[13]=1;
   }
   printf("foo\n");
   inst[14]=1;
}
```

- Slowdown typically not acceptable for production code
- Unless done in a clever way? Printf??
- Here: Use additional hardware resources instead

Hardware-based Runtime Verification

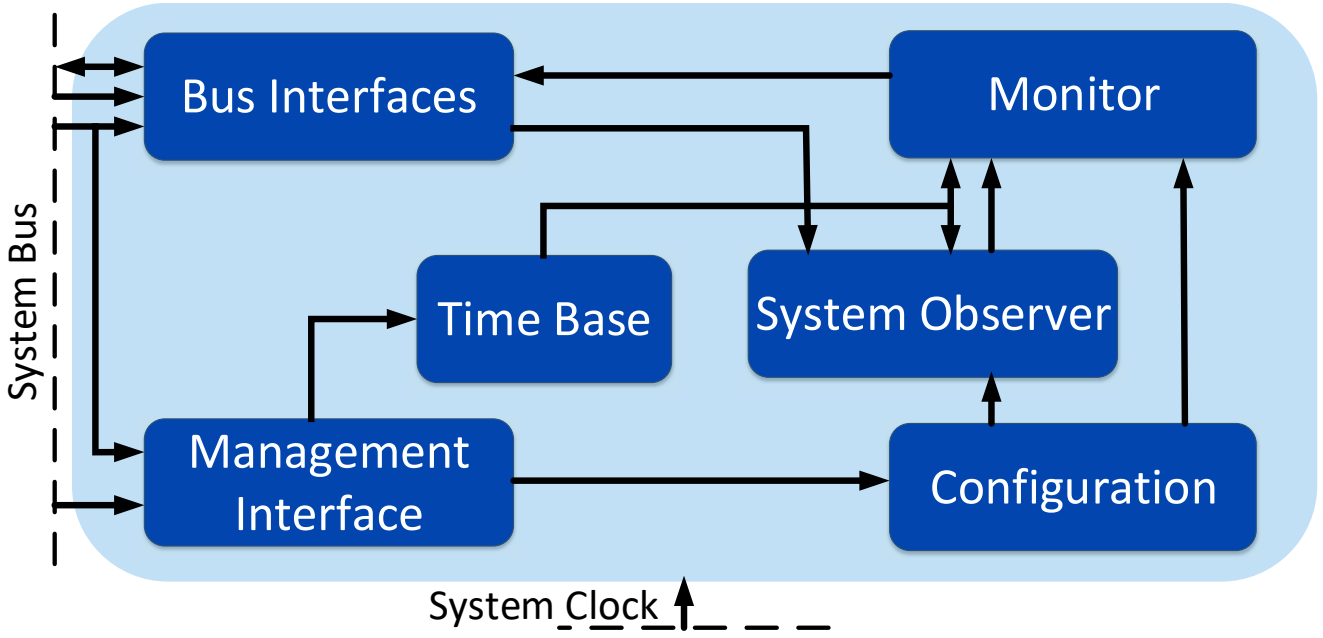
SoC approach – Archticture



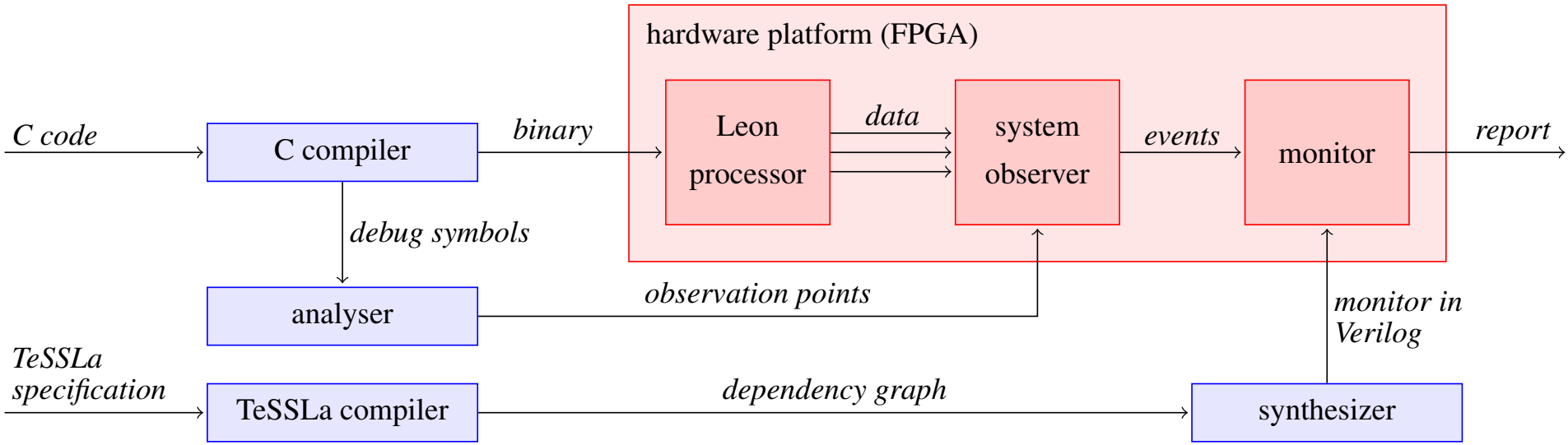
Non-intrusive Runtime Verification within a System-on-Chip, RUME 2018

José Rufino, António Casimiro , Felix Dino Lange, Martin Leucker, Torben Scheffel, Malte Schmitz, Daniel Thoma

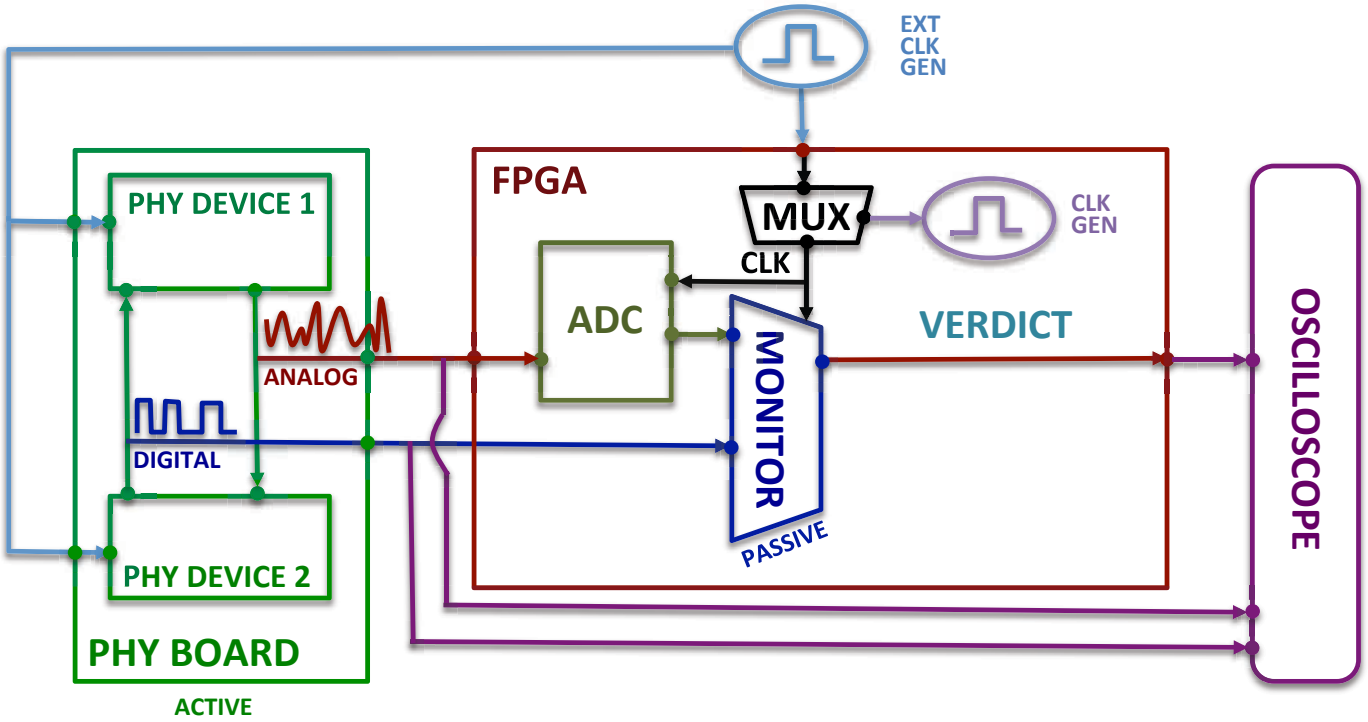
SoC approach – Observer Entity



SoC approach with TeSSLa specifications

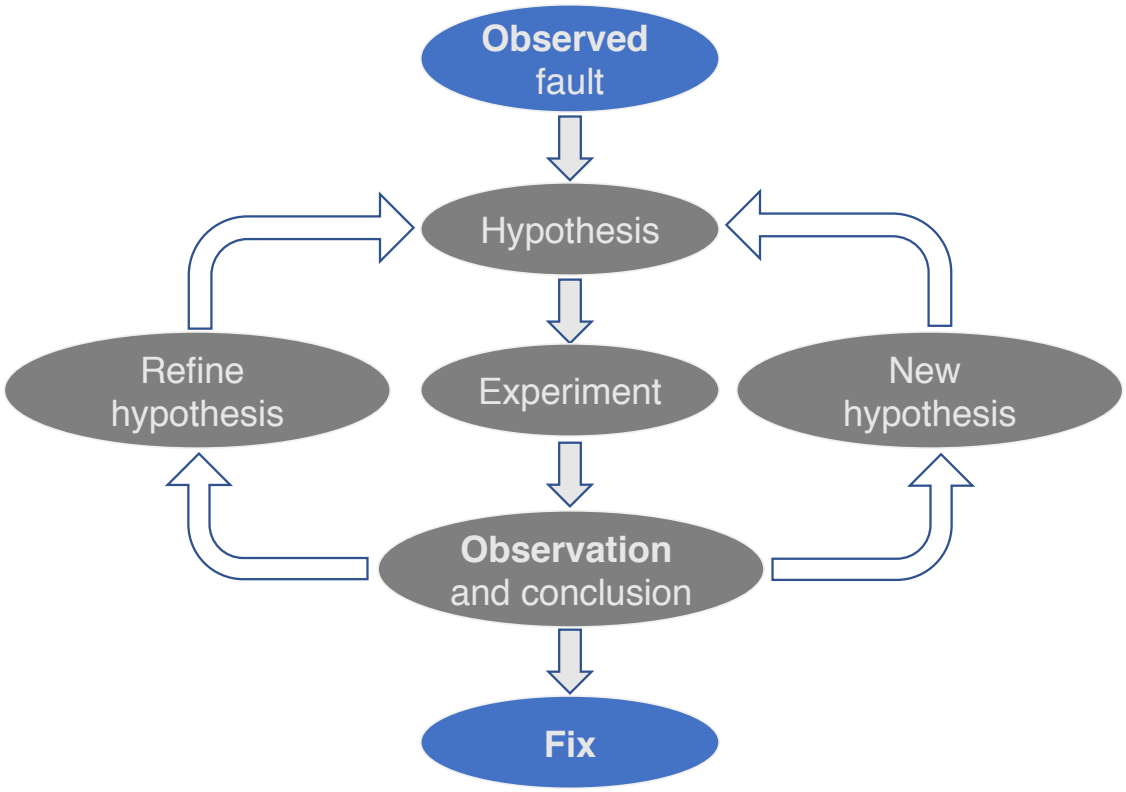


Fixed Monitors for RV



From: Ezio Bartocci, Yliès Falcone, Adrian Francalanza, Giles Reger. Introduction to Runtime Verification. Lectures on Runtime Verification. Introductory and Advanced Topics, 10457, Springer, pp.1-33, 2018, Lecture Notes in Computer Science

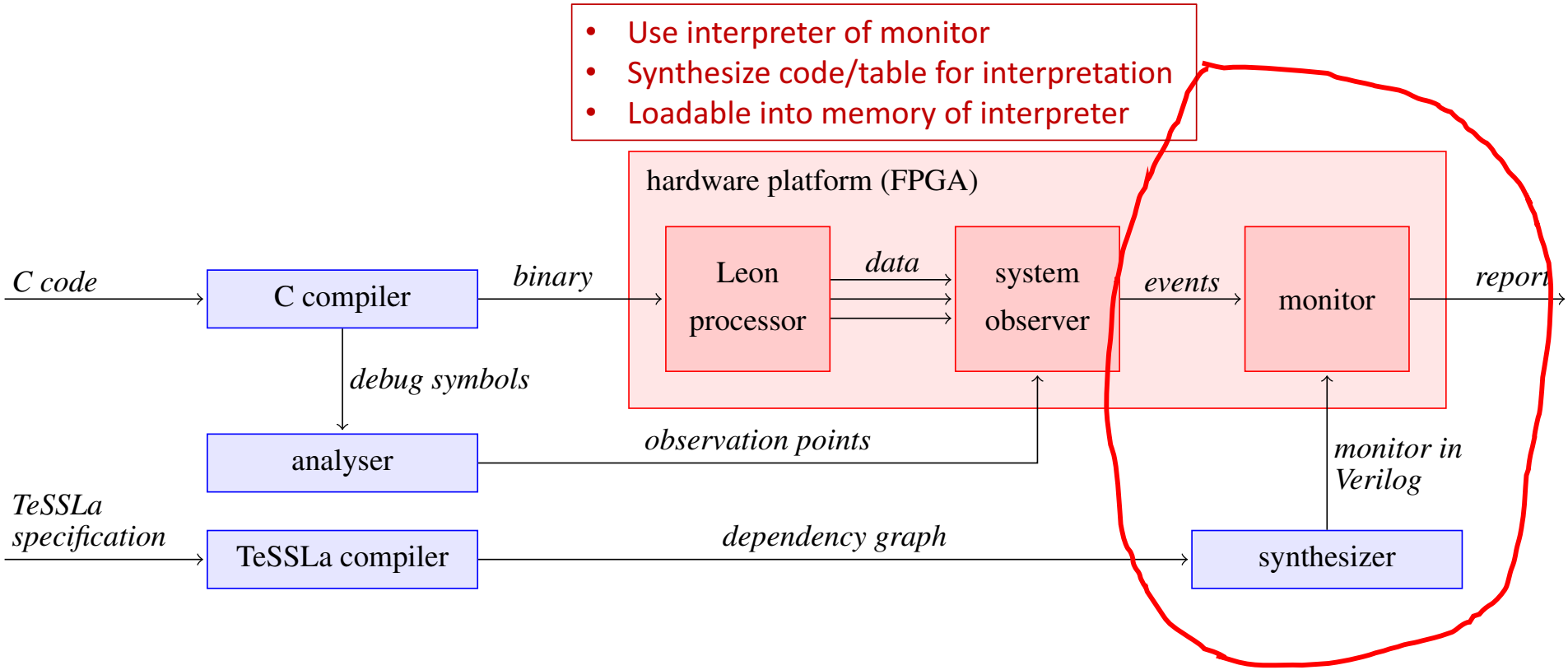
Debugging



Requirements

- Quick loop for synthesizing new properties on the test system
- Still long-term observability useful
- Change monitoring focus dependent on previous outcome

SoC approach with TeSSLa specifications



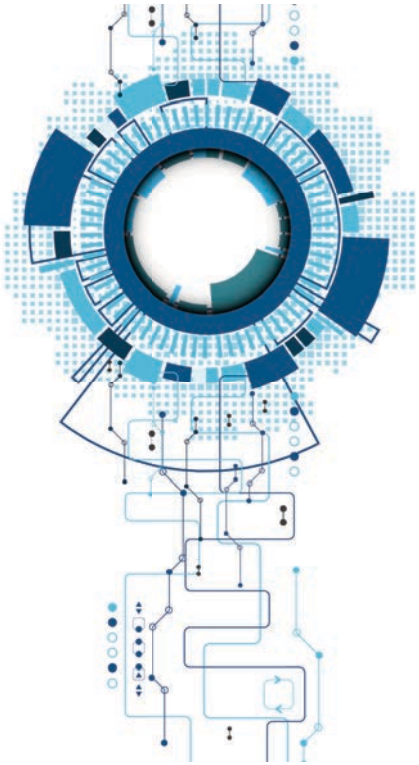
Rico Backasch, Christian Hochberger, Alexander Weiss, Martin Leucker, Richard Lasslop:
Runtime verification for multicore SoC with high-quality trace data. ACM Trans. Design Autom. Electr. Syst. 18(2): 18:1-18:26 (2013)

The COEMS approach

Continuous Online Observation for Embedded Multi-core Systems
EU Horizon 2020 project

The COEMS Consortium

ICT-10-2016
COEMS
Continuous Observation of
Embedded Multicore Systems



University of Lübeck

Accemic Technologies

Thales Romania

Thales Austria

Høgskulen på Vestlandet / Western Norway University of Applied Science

Airbus

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732016.

Objectives

- Increase test efficiency
- Increase debug efficiency
- Increase test effectivity
- Improve embedded systems performance

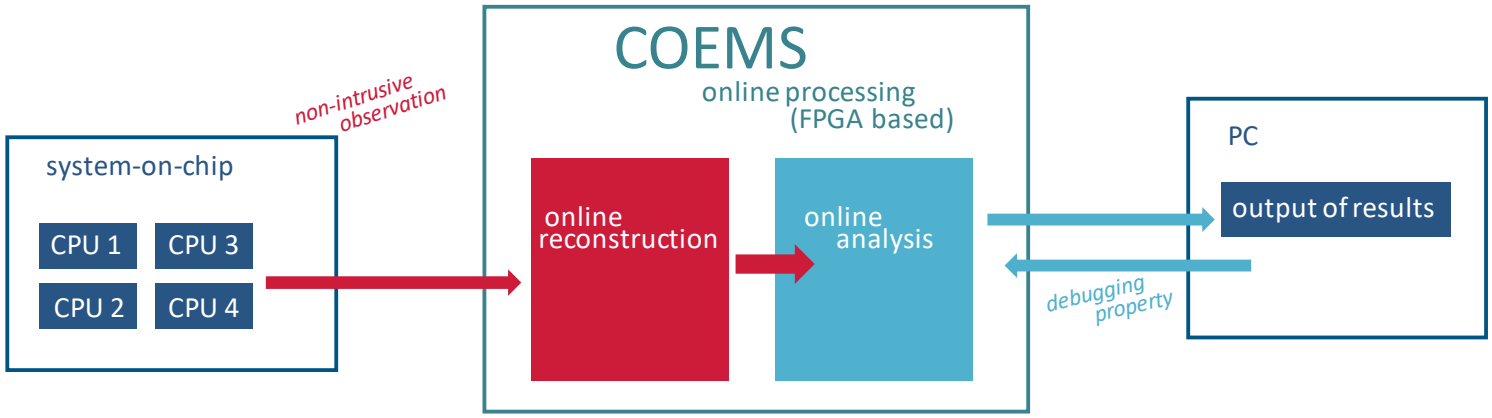
- For embedded multicore systems (ARM etc.)
- Running (multithreaded) C programs
- Perhaps on Linux

Applications – Semi-Formal Verification

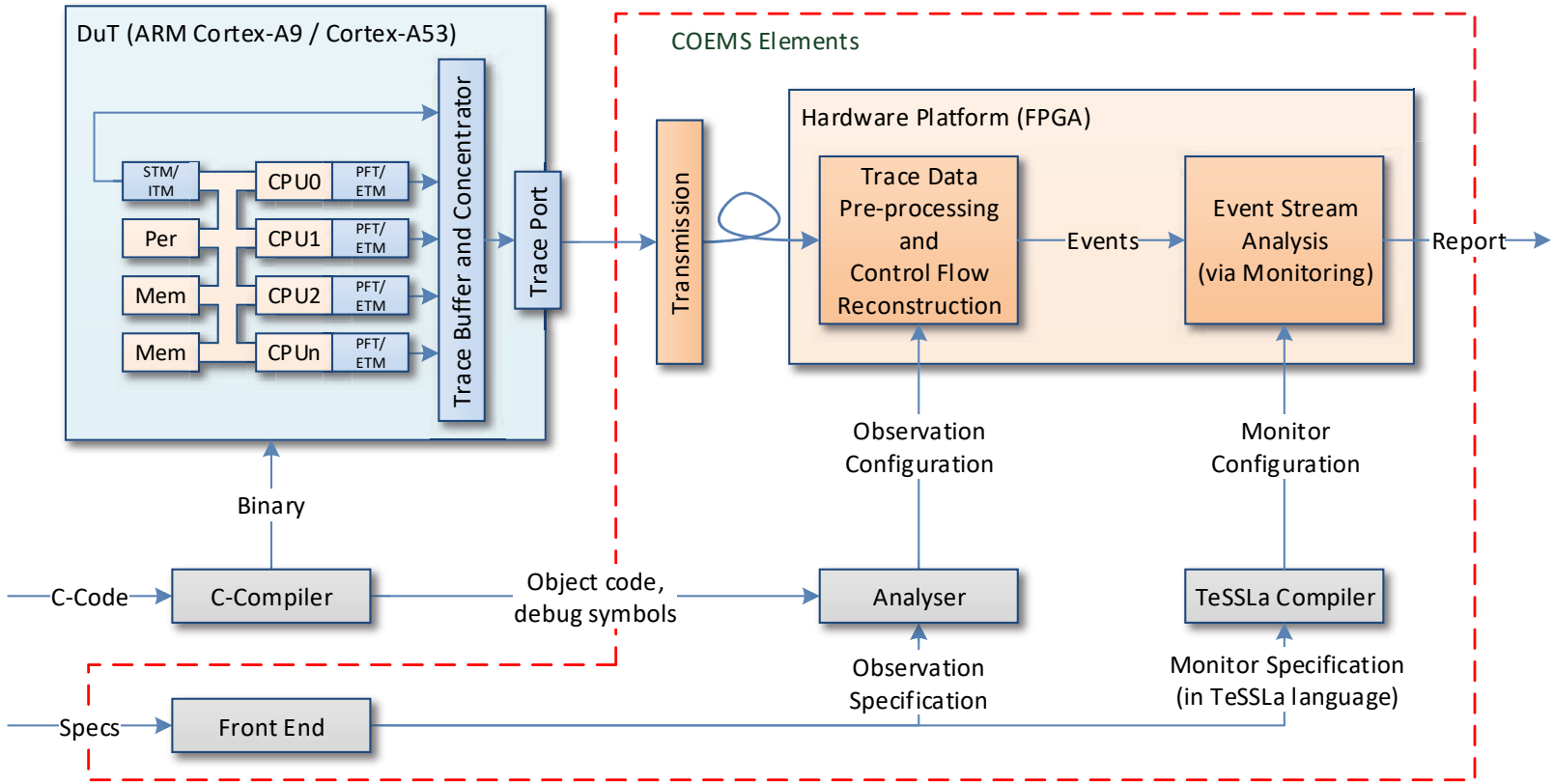
- Finding Data Races
- Finding Timing Bugs
- Finding Functional Bugs
- Measuring Coverage
- Measurement of Worst-Case Execution Time and Worst-Case Response Time

COEMS set-up

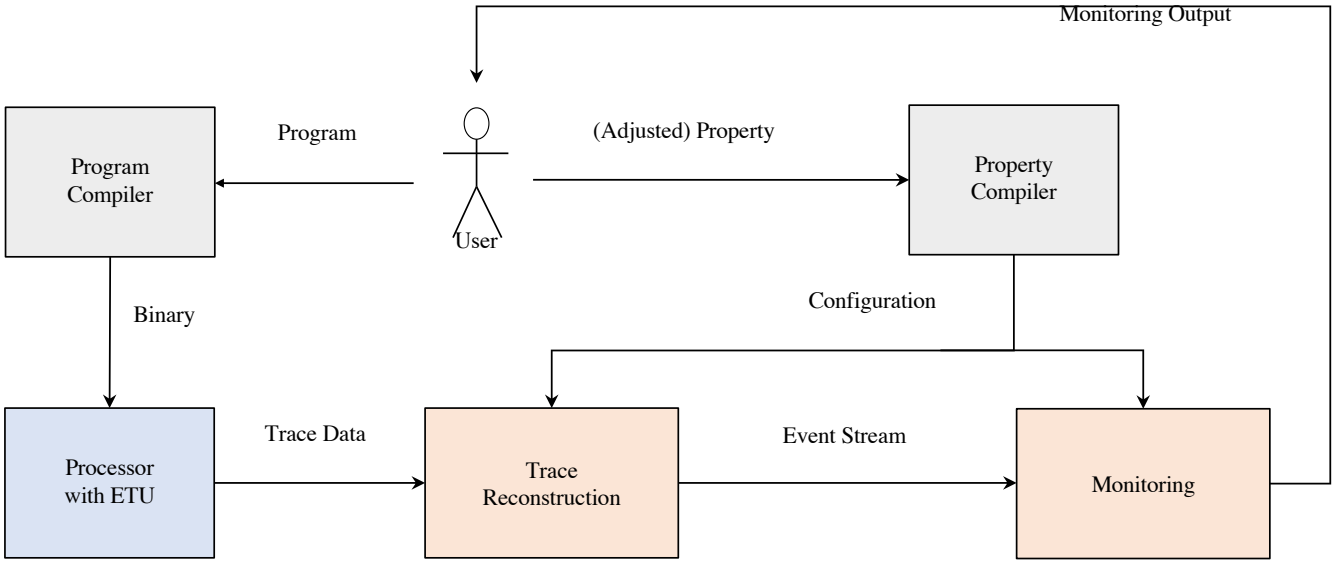

VIRTEX-7


COEMS
online processing
(FPGA based)
non-intrusive
observation
system-on-chip
CPU 1
CPU 3
CPU 2
CPU 4
online
reconstruction
online
analysis
PC
output of results
debugging
property

System Overview



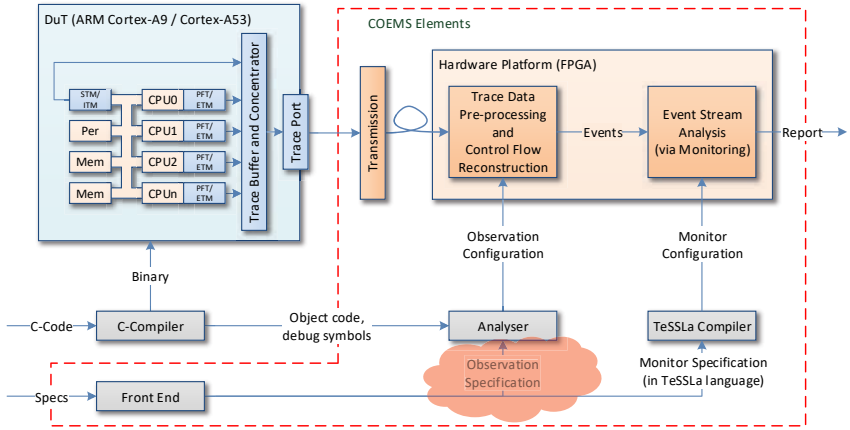
Rapidly adjustable Embedded Trace Online Monitoring – RETOM



Observation Specification

Specification of atomic artefacts to be observed

- Elements of Source Code
  - Program Line
  - Entering / Leaving a Function / Exception
  - Reading / Writing variables
- Elements of the Binary
  - PC Address
  - Calls / Returns
  - Specific Operations
    (e.g. Floating Point Operations)
- Hardware Supported Instrumentation
  - ITM, STM



Observation Specification

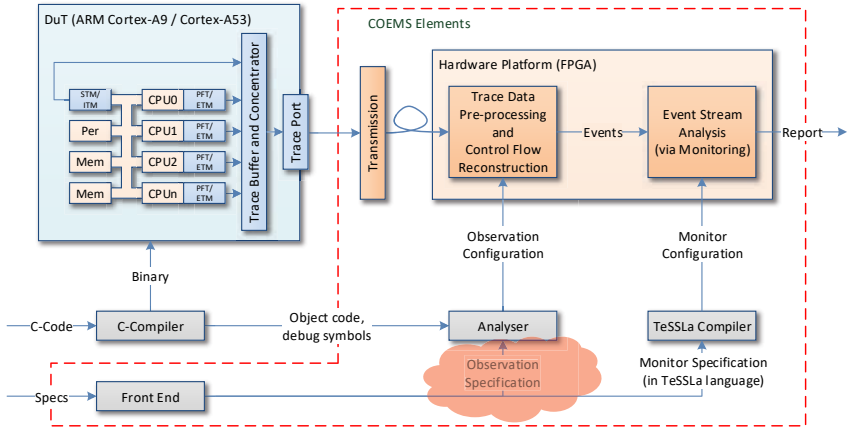
ICT-10-2016 COEMS
Continuous Observation of Embedded Multicore Systems

C Code

```
1  int main() {
2    int sum = 0;
3    for (int i = 0; i < 5; i++) {
4      sum = add(sum, sub(i,2));
5    }
6    sum = add(sum, add(21,21));
7    for (int i = 0; i < 5; i++) {
8      sum = add(sum, sub(i,2));
9    }
10 }
11
12 int add(int x, int y) {
13   return x+y;
14 }
15
16 int sub(int x, int y) {
17   return x-y;
18 }
19
```

Specification

```
1  def add_event := function_call("add")
2  def add_count := eventCount(add_event)
3
4  def sub_event := function_call("sub")
5  def sub_count := eventCount(sub_event)
6
7  def diff := add_count - sub_count
8
9  def error := diff >= 2
10
11 out diff
12 out error
13
```



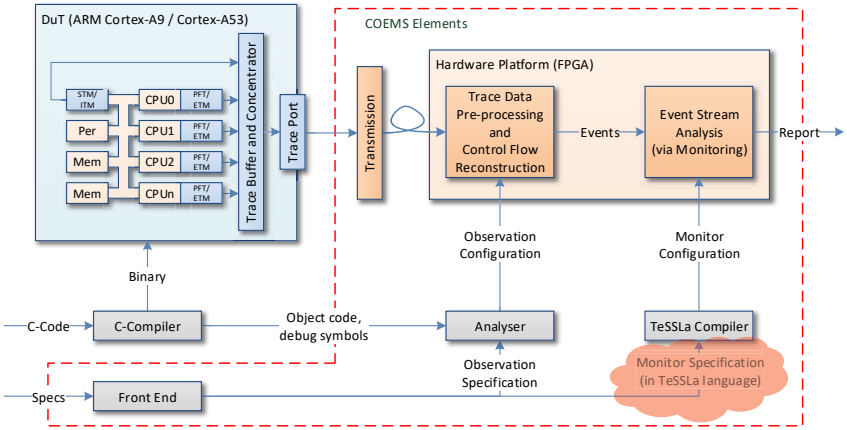
Monitor Specification Language (TeSSLa)

ICT-10-2016 COEMS
Continuous Observation of Embedded Multicore Systems

Specification

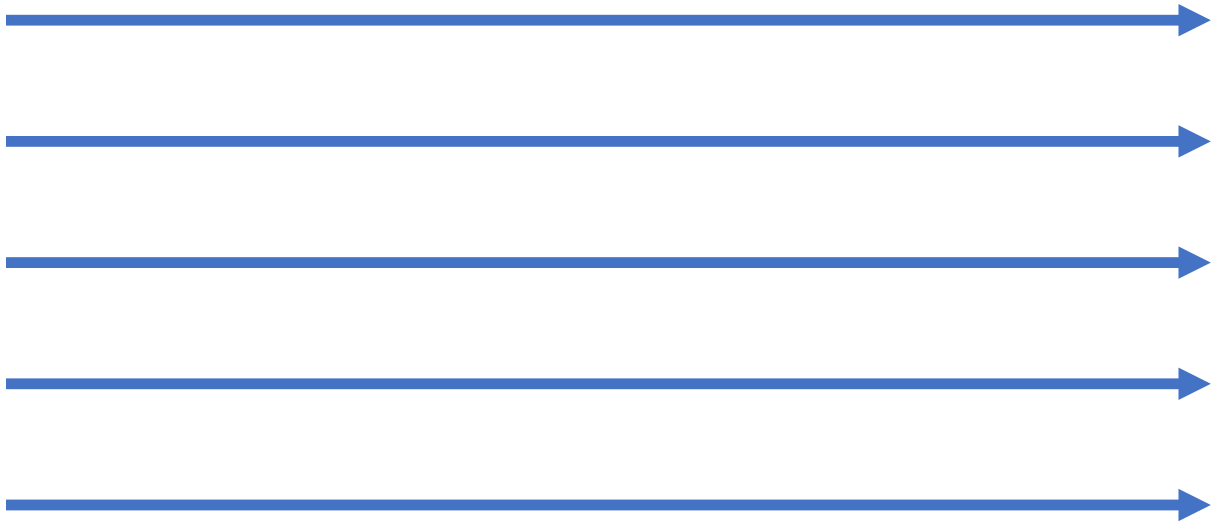
```
1  def add_event := function_call("add")
2  def add_count := eventCount(add_event)
3
4  def sub_event := function_call("sub")
5  def sub_count := eventCount(sub_event)
6
7  def diff := add_count - sub_count
8
9  def error := diff >= 2
10
11 out diff
12 out error
13
```

- **Event-Stream-Analysis**
  Event ordering constraints, timing constraints and quantitative analysis
- **Declarative style**
  Describe correctness criterion or analysis goal without having to think about the algorithmic check
- **Modularity**
  allowing abstractions based on few primitives
- **Time**
  as first-class citizen
- **Events & signals**
  based on one common abstraction
- **Finite memory**
  allowing execution on FPGA



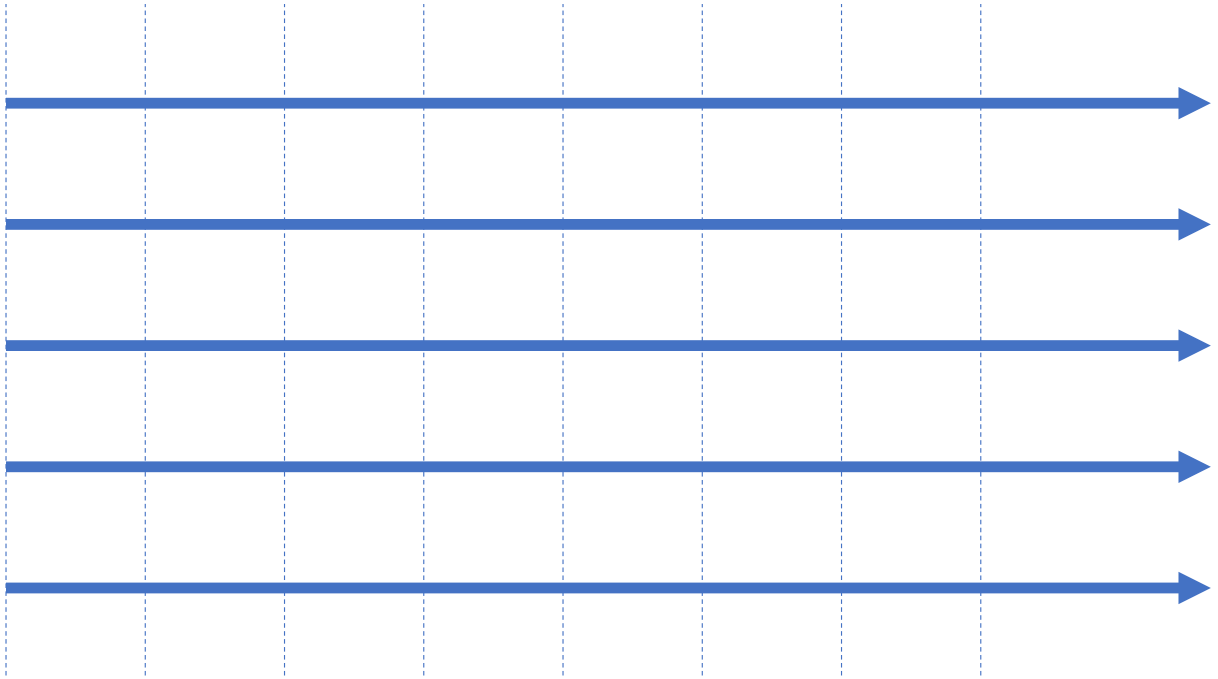
Specification Languages for RV

Streams



Concurrency/Distribution

Streams



Time? Synchrony/Ticks

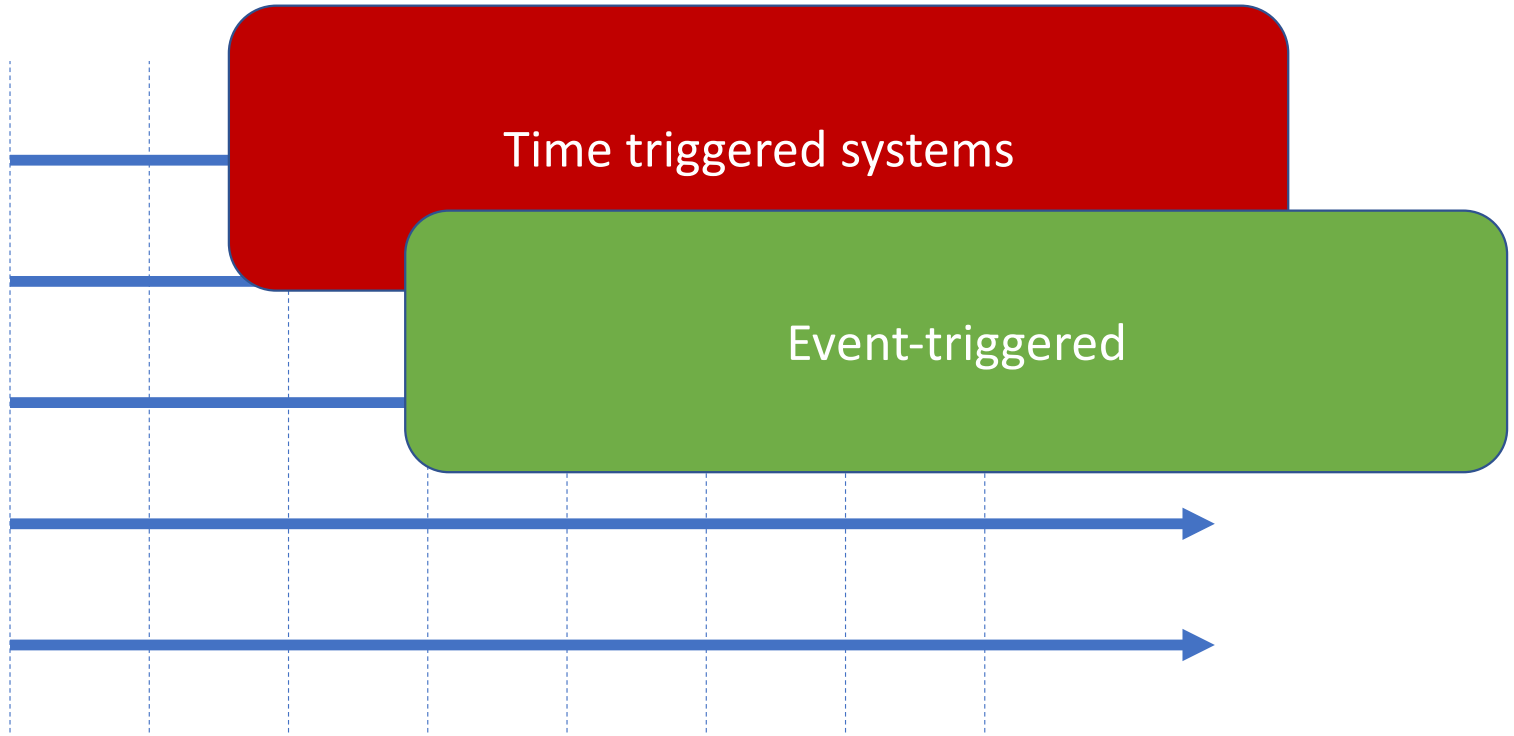
Equational specifications, data, time, concurrency

LOLA

[D'Angelo et al.]

$$
\begin{aligned}
s_1 &= \mathbf{true} \\
s_2 &= t_3 \\
s_3 &= t_1 \vee (t_3 \leq 1) \\
s_4 &= ((t_3)^2 + 7) \ mod \ 15 \\
s_5 &= \mathtt{ite}(s_3, s_4, s_4 + 1) \\
s_6 &= \mathtt{ite}(t_1, t_3 \leq s_4, \neg s_3) \\
s_7 &= t_1[+1, \mathbf{false}] \\
s_8 &= t_1[-1, \mathbf{true}] \\
s_9 &= s_9[-1, 0] + (t_3 \ mod \ 2) \\
s_{10} &= t_2 \ \vee \ (t_1 \ \wedge \ s_{10}[1, \mathbf{true}])
\end{aligned}
$$

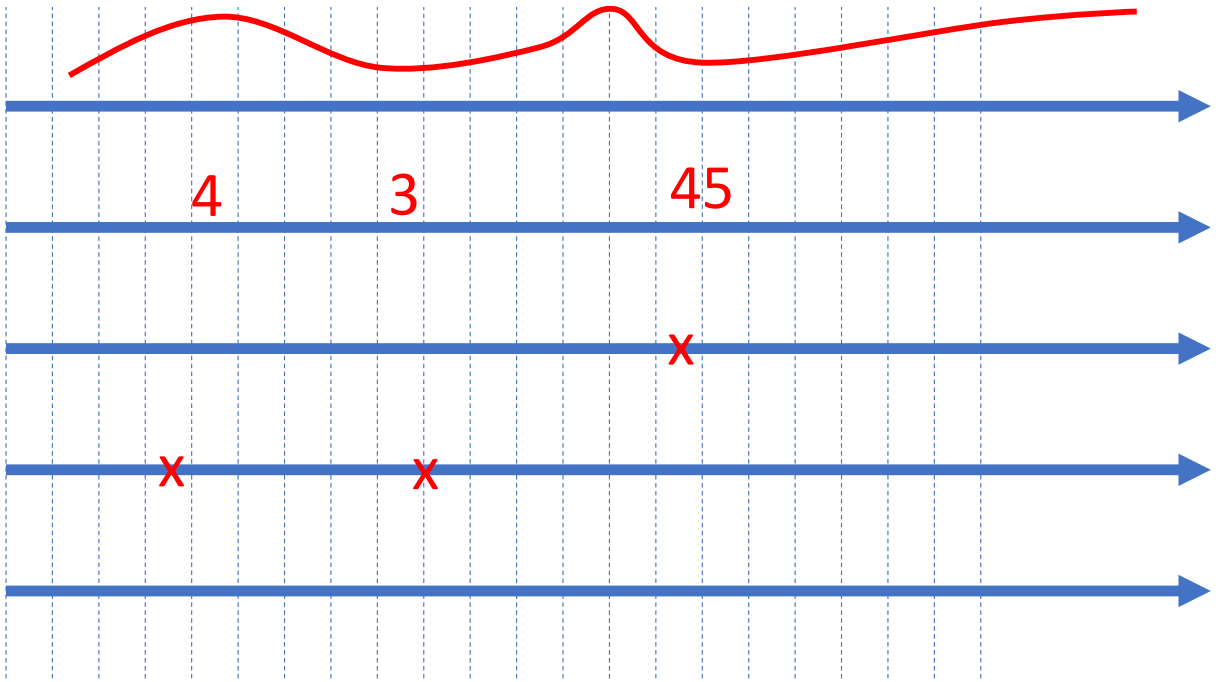
Streams



Time? Synchrony/Ticks

TeSSLa’s Streams

Streams



Time? Events

Streams of Programs - After Discretization

Values

e.g., of a program variable `x`

Program events

e.g., call to `my_func()`

Streams

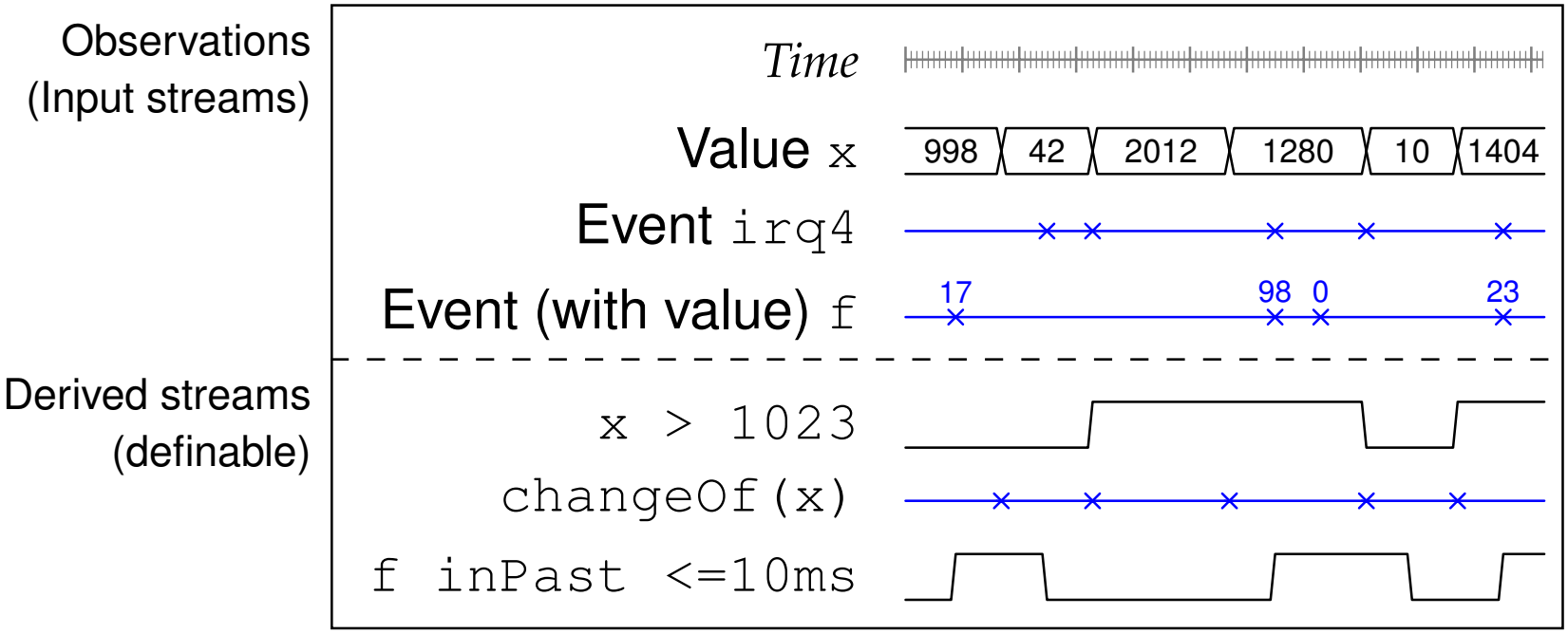
Values

e.g., of a program variable `x`

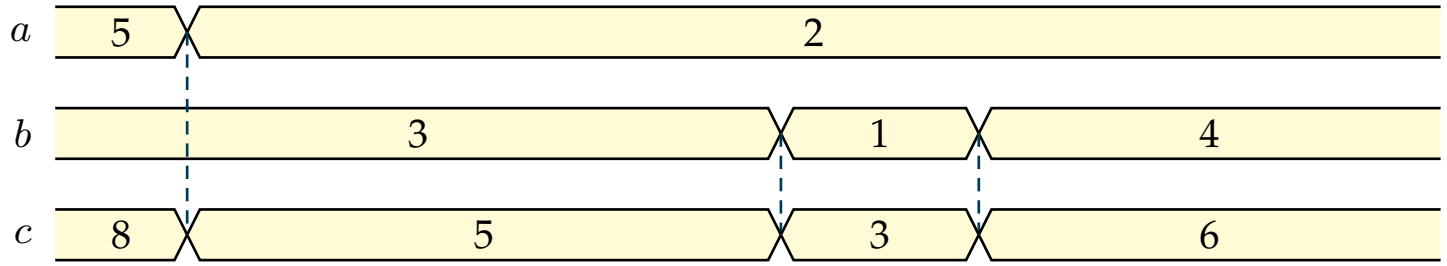
Program events

e.g., call to `my_func()`

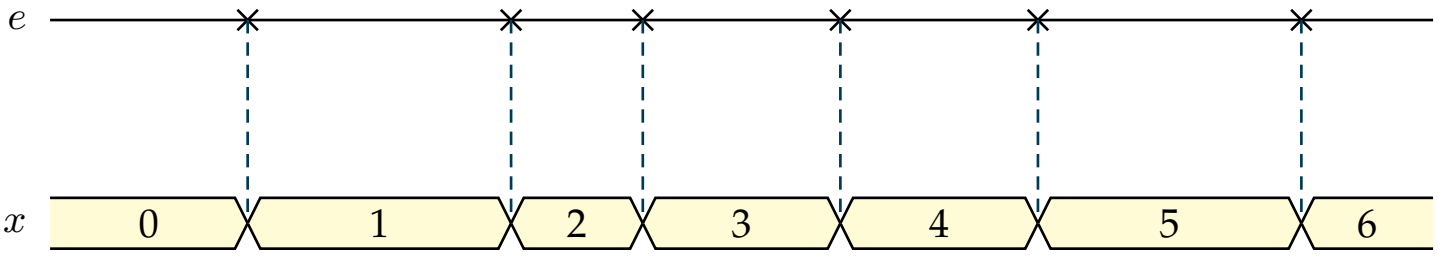
Streams here for RV



TeSSLa by Example

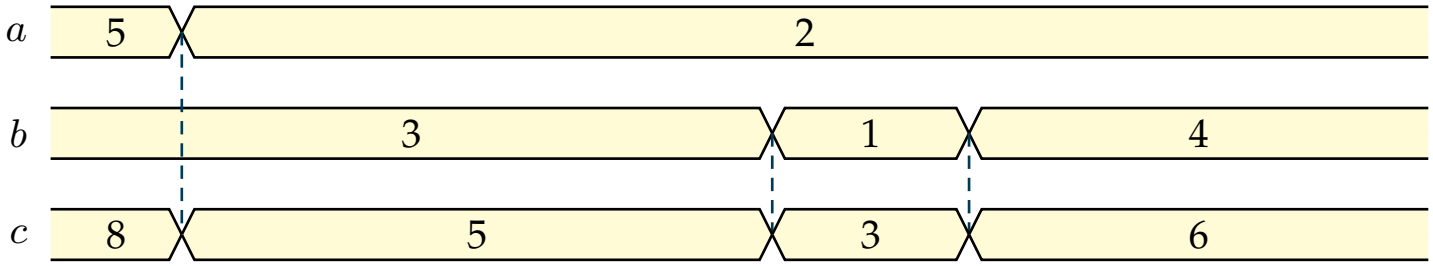



```
def c := a + b
```

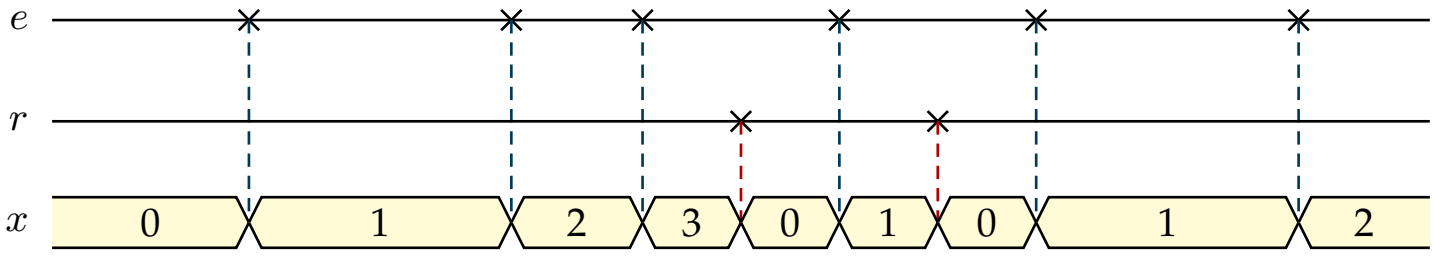



```
def x := eventCount(e)
```

TeSSLa by Example

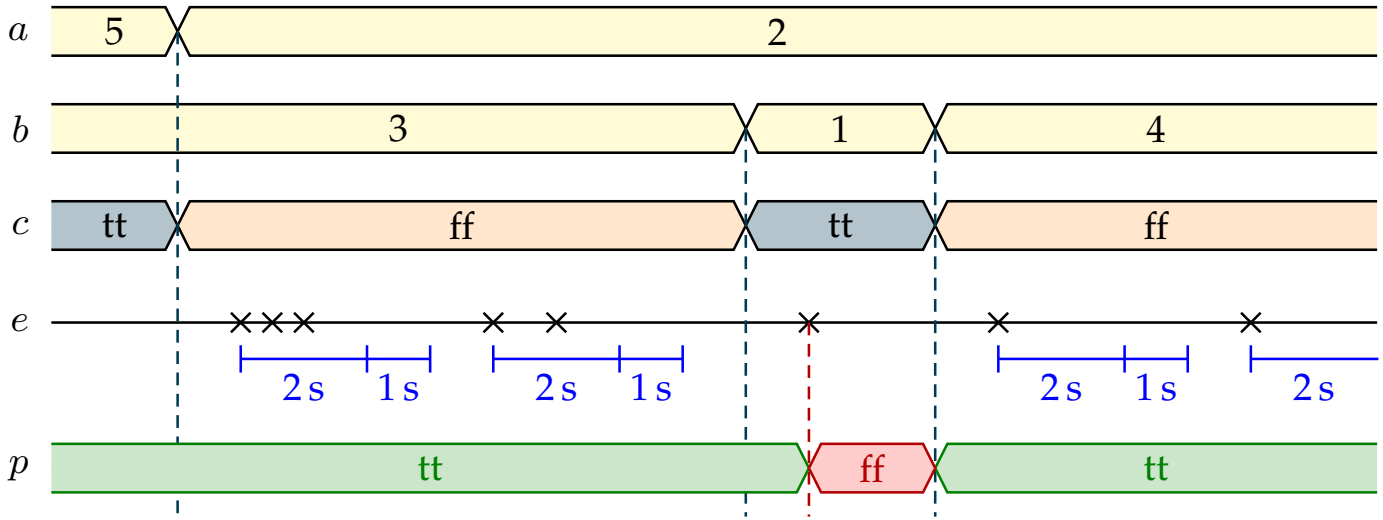



```
def c := a + b
```



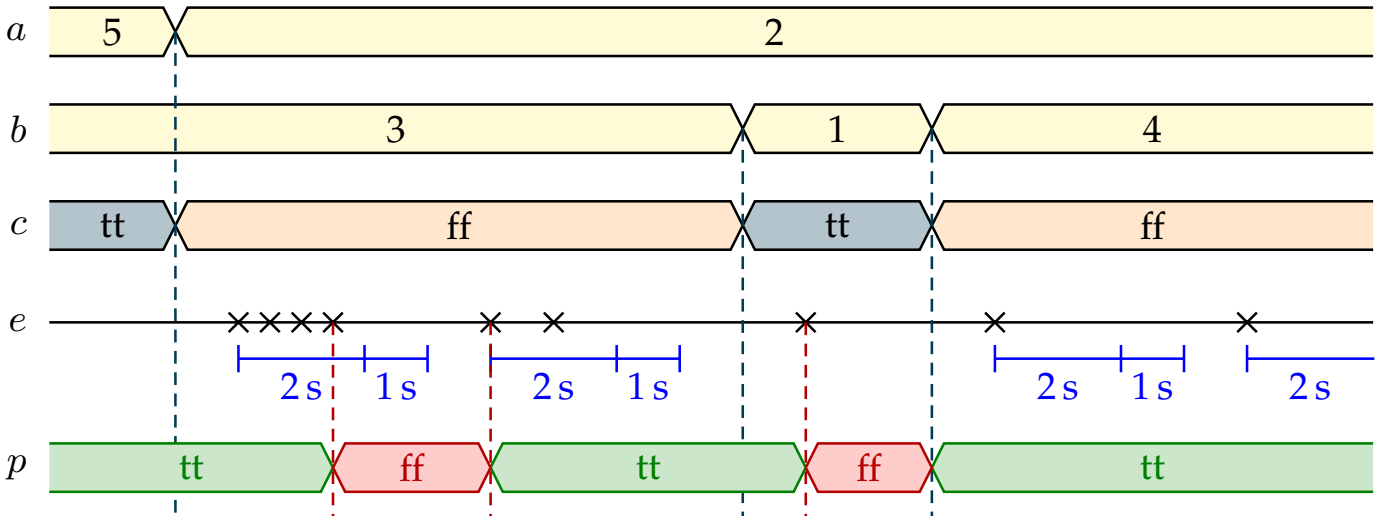

```
def x := eventCount(e, reset = r)
```

TeSSLa – Burst Pattern (Macros)



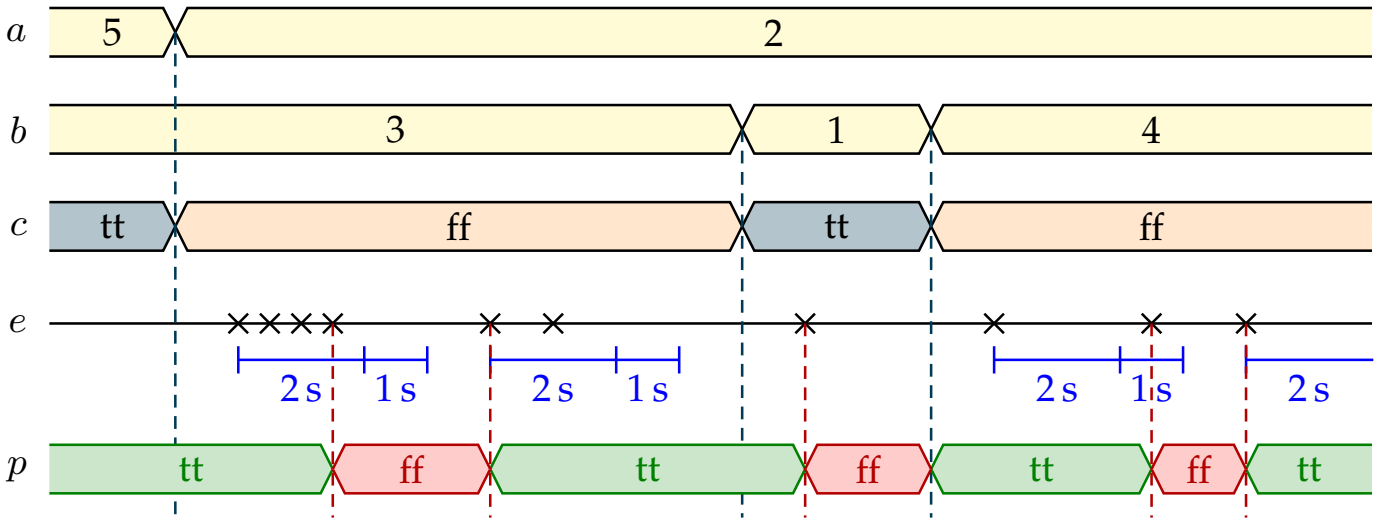

```
def c := a > b
def p := if c
         then noEvent(e, since = rising(c))
         else bursts(e, burstLength = 2s,
                        waitingPeriod = 1s,
                        burstAmount = 3,
                        since = falling(c))
```

TeSSLa – Burst Pattern (Macros)



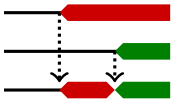

```
def c := a > b
def p := if c
         then noEvent(e, since = rising(c))
         else bursts(e, burstLength = 2s,
                        waitingPeriod = 1s,
                        burstAmount = 3,
                        since = falling(c))
```

TeSSLa – Burst Pattern (Macros)



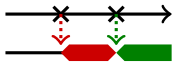

```
def c := a > b
def p := if c
         then noEvent(e, since = rising(c))
         else bursts(e, burstLength = 2s,
                        waitingPeriod = 1s,
                        burstAmount = 3,
                        since = falling(c))
```

TeSSLa's operators



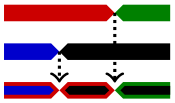
default, **defaultFrom**

- Initialize streams
- Start of recursion



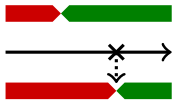
time

- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps



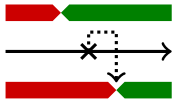
lift

- Lifts standard functions to streams
- Used to manipulate data, events, ...



last

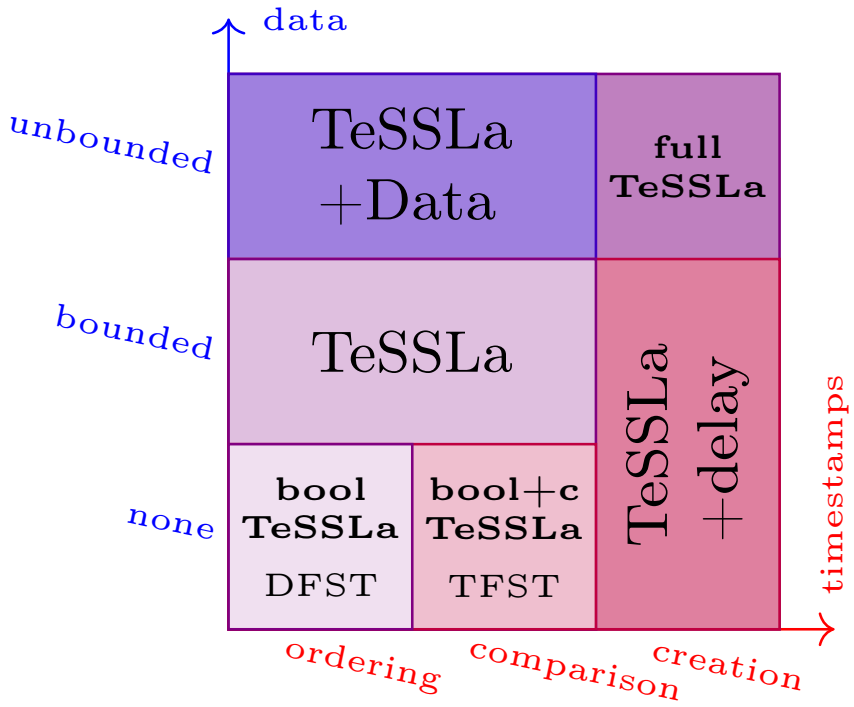
- Refers to previous value of a stream
- Recursion



delayedLast

- Only way to create events
- Takes a stream and delays events by its current value
- Output events have the previous value of another given stream

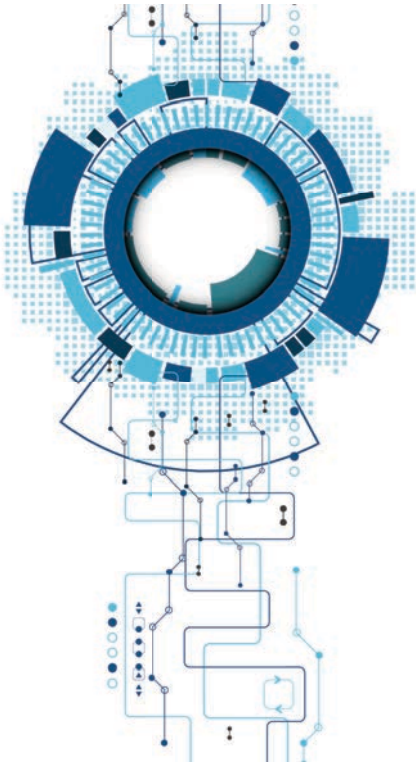
TeSSLa's fragments



EU Horizon 2020 Project: COEMS

The COEMS Consortium

ICT-10-2016
COEMS
Continuous Observation of
Embedded Multicore Systems



UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

University of Lübeck

ACCEMIC
TECHNOLOGIES

Accemic Technologies

THALES

Thales Romania

Thales Austria

Høgskulen
på Vestlandet

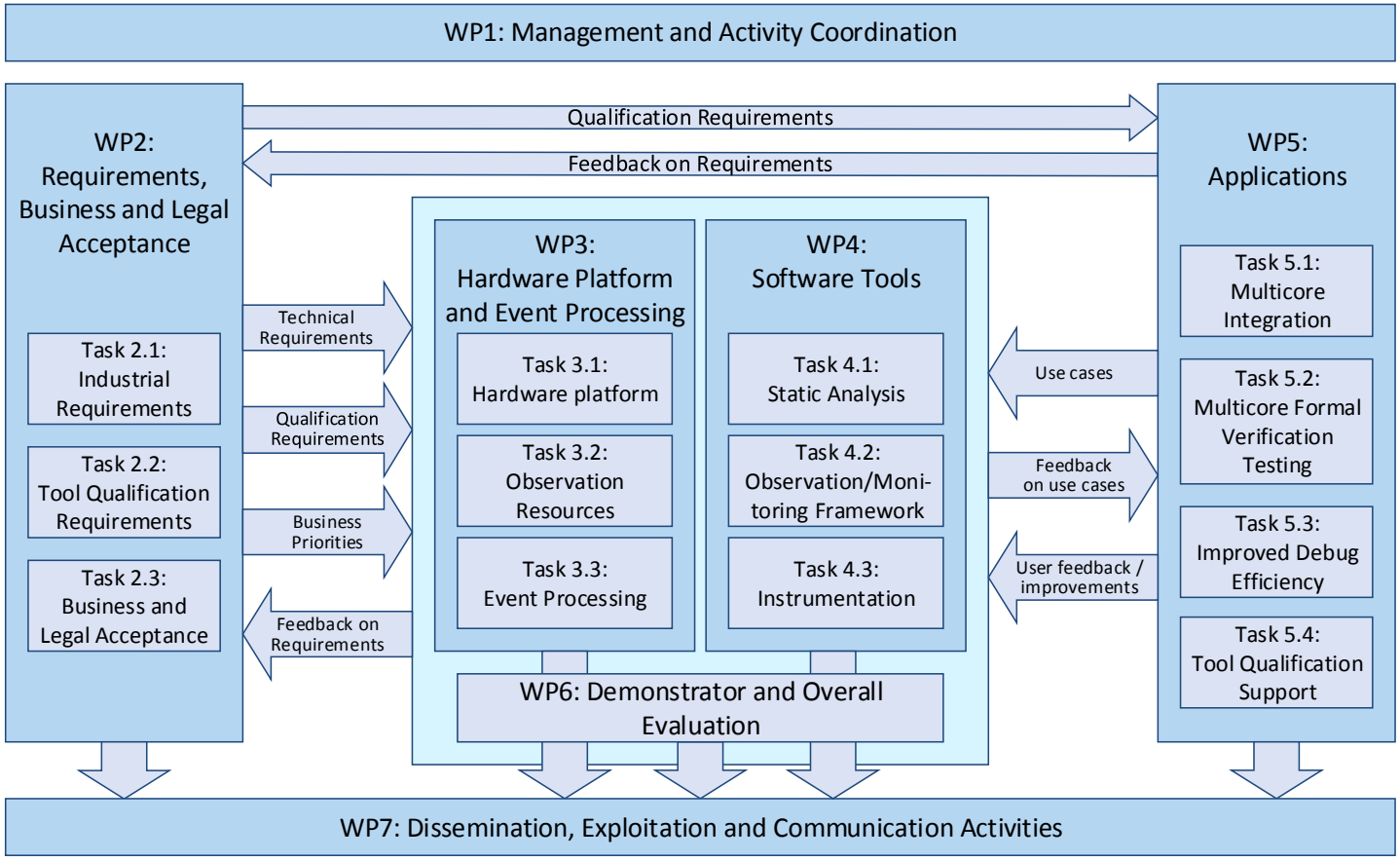
Høgskulen på Vestlandet / Western Norway
University of Applied Science

AIRBUS

Airbus

This project has received funding from the European
Union's Horizon 2020 research and innovation programme
under grant agreement no. 732016.

WP Structure



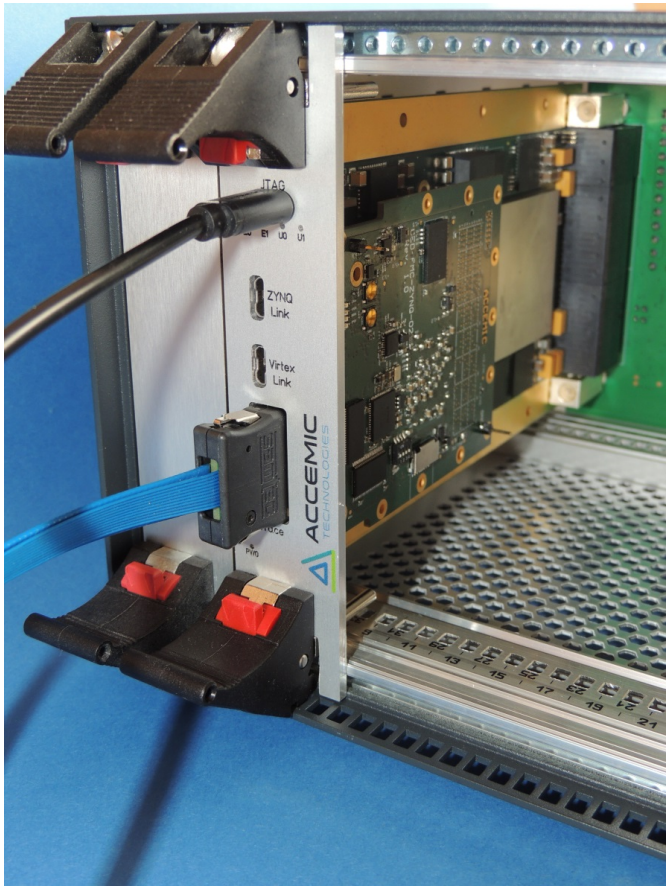
COEMS set-up


VIRTEX-7
COEMS
online processing
(FPGA based)
non-intrusive
observation
system-on-chip
CPU 1
CPU 3
CPU 2
CPU 4
online
reconstruction
online
analysis
PC
output of results
debugging
property

Hardware Platform

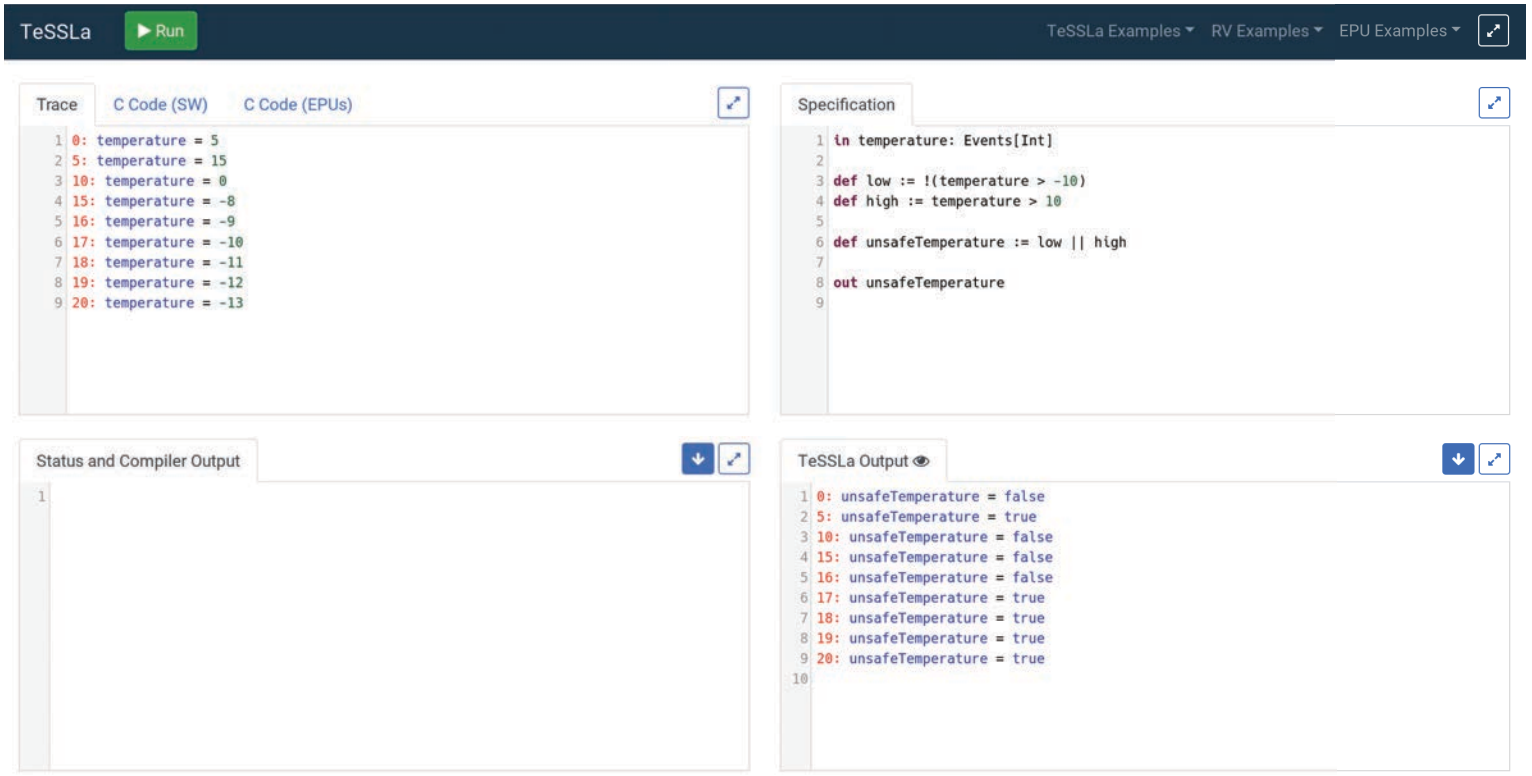
ICT-10-2016
COEMS
Continuous Observation of Embedded Multicore Systems

- Hardware
  - Virtex-7 series FPGA (available)
  - Zynq Ultrascale+ SOC (under development)
  - RLDRAM3 memory for fast lookup tables
  - Interface to Aurora (Nexus, HSSTP)
  - VPX / FMC form factor
- Functionality
  - Online trace data processing
    (Coresight trace data -> event stream)
  - Supported architectures:
    ARM Cortex-A9, ARM Cortex-A53*,
    QorIQ PPC*, Infineon Aurix*
  - Online processing of event stream

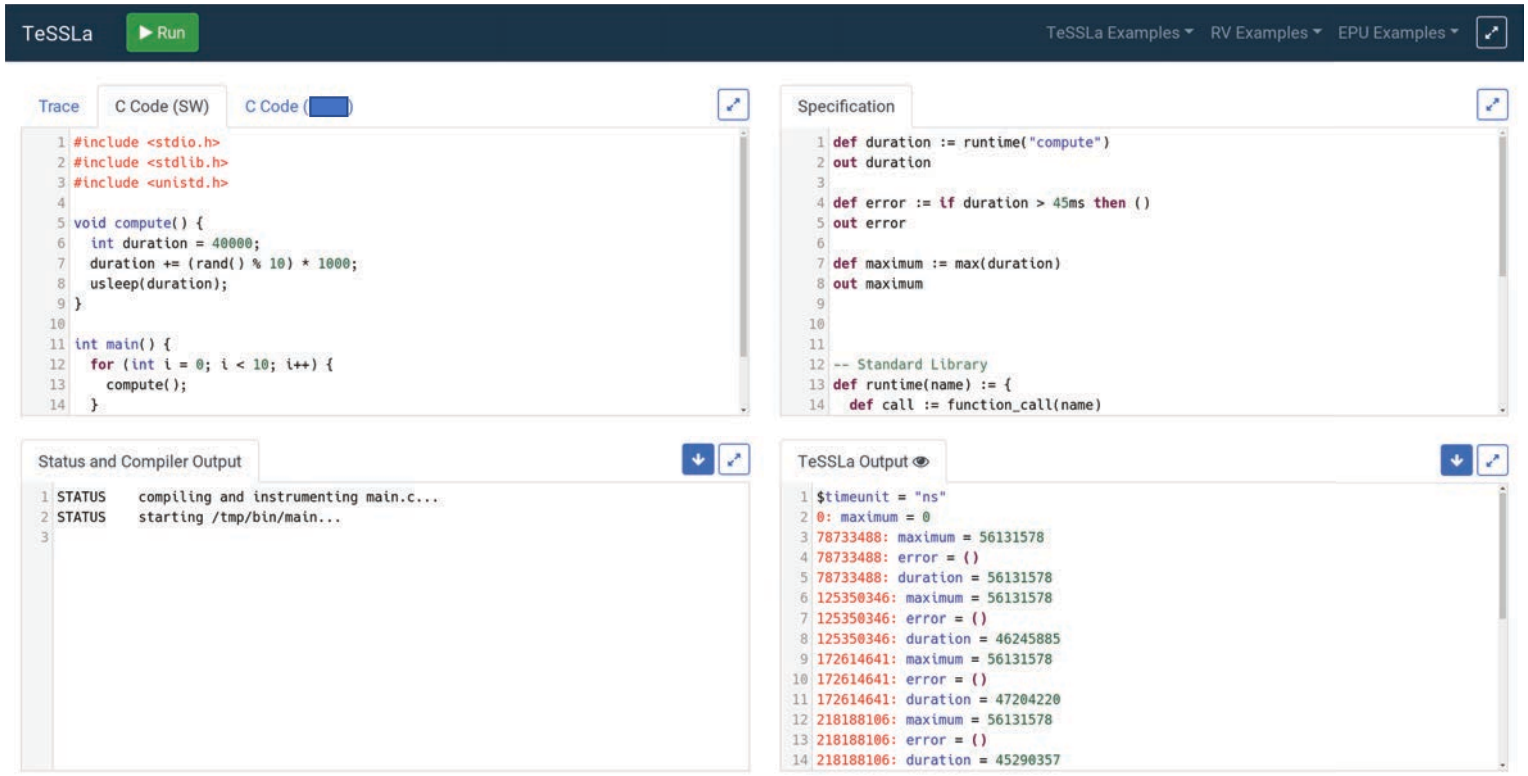


*under development

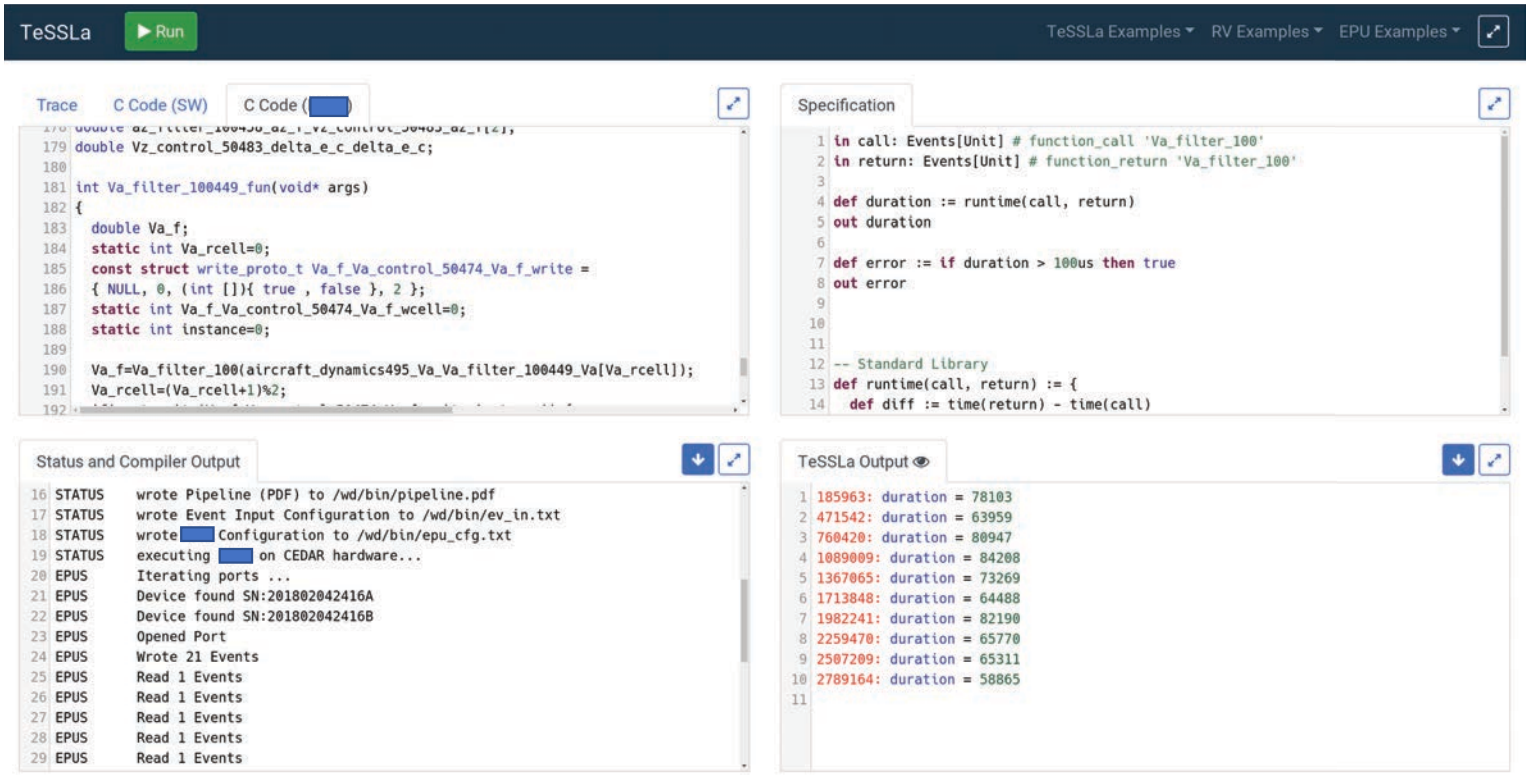
Web IDE – with trace



Web IDE with C-code – software backend



Web IDE – with C-code – hardware backend



Conclusions

Summary

- Sometimes software annotations not acceptable
- Usage of trace functionality of modern processors feasible
- Extra hardware may be used to monitor non-intrusively
- Sophisticated ideas necessary to make overall approach feasible
- Hardware-based RV might be a game changer

Future Work

- Abstractions in TeSSLa
- Precise Relation of TeSSLa fragments to STL
- Partial-Order Semantics

- Support for ITM traces
- Increase performance of implementation
- Achieve TRL6
- Enhance training material