UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

Rapidly Adjustable Non-Intrusive Online Monitoring for Multi-core Systems

N. Decker P. Gottschling C. Hochberger M. Leucker
T. Scheffel M. Schmitz A. Weiss

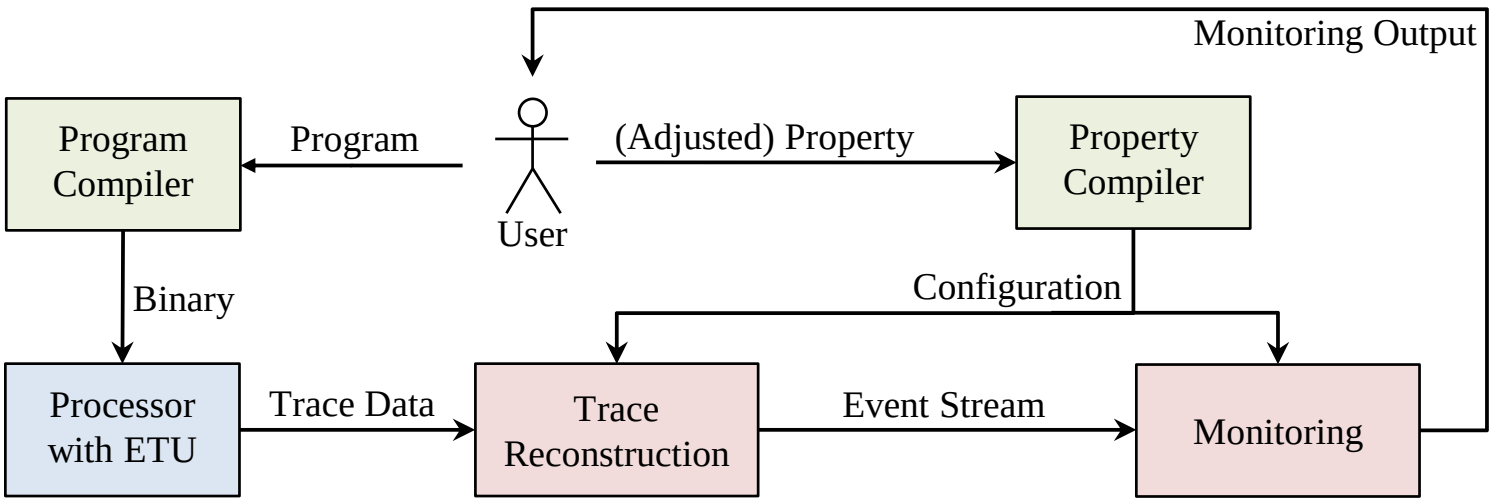
Institute for Software Engineering and Programming Languages,
University of Lübeck, Germany

Brazilian Symposium on Formal Methods 2017

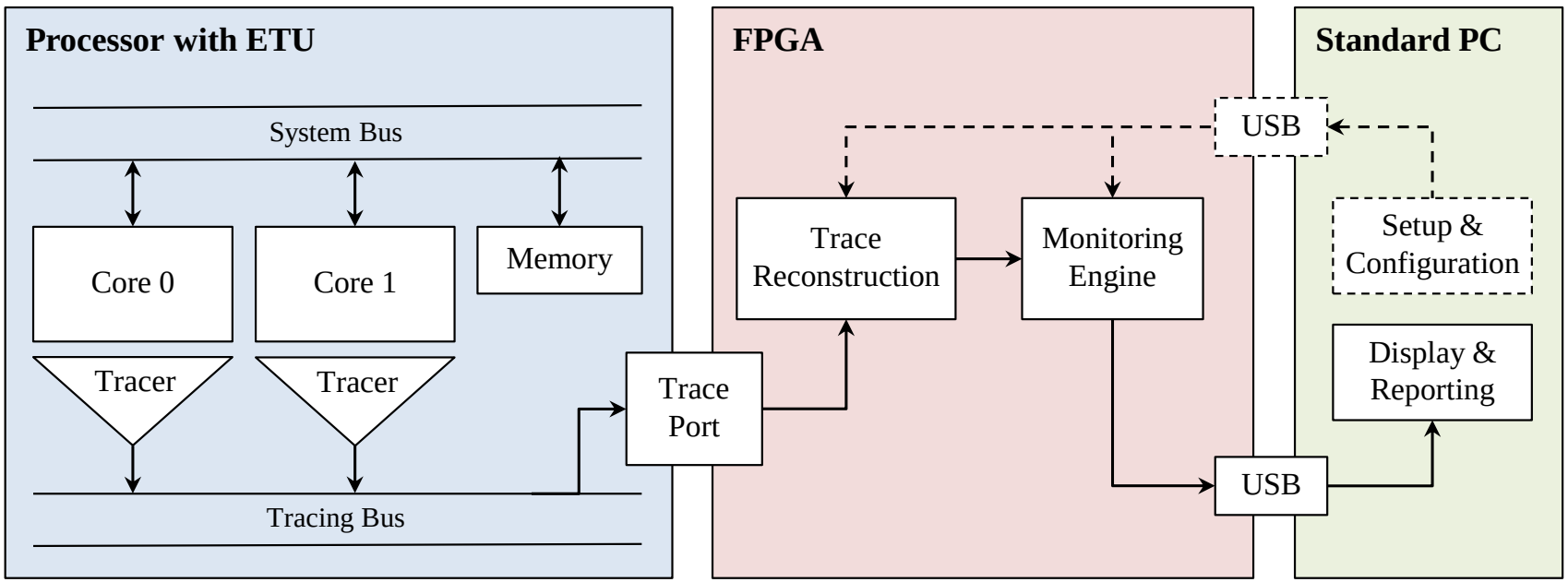
Goals

- Online monitoring
  - $\Rightarrow$ No offline computation
  - $\Rightarrow$ Live trace reconstruction and event processing on FPGA
- Non-intrusive for the system under test
  - $\Rightarrow$ No instrumentation
  - $\Rightarrow$ External trace port
- Rapidly adjustable monitors
  - $\Rightarrow$ Not synthesizing the monitor
  - $\Rightarrow$ Synthesize flexible FPGA architecture which can be configured
- Common trace source
  - $\Rightarrow$ No special hardware
  - $\Rightarrow$ Embedded Trace Unit (ETU)

RETOM Workflow



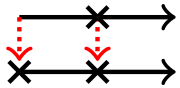
Architecture



Temporal Stream-based Specification Language

Temporal Stream-based Specification Language

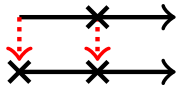
default, defaultFrom



- Initialize streams
- Start of recursion

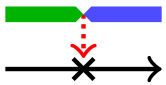
Temporal Stream-based Specification Language

default, defaultFrom



- Initialize streams
- Start of recursion

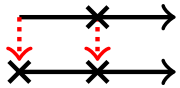
time



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

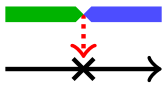
Temporal Stream-based Specification Language

default, defaultFrom



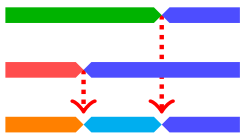
- Initialize streams
- Start of recursion

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, ...

TeSSLa: Lifted Addition

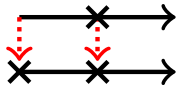
Lift arbitrary functions to streams with signal semantics.

$$\mathsf{lift}_{\mathbb{D}_1,\ldots,\mathbb{D}_n,\mathbb{D}'} : (\mathbb{D}_1 \times \cdots \times \mathbb{D}_n \rightharpoonup \mathbb{D}') \to (\mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n} \to \mathcal{S}_{\mathbb{D}'})$$

$$\mathsf{add} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z},$$
$$z = \mathsf{lift}(\mathsf{add})(x, y)$$

Temporal Stream-based Specification Language

default, defaultFrom

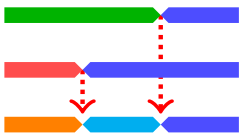


- Initialize streams
- Start of recursion

time

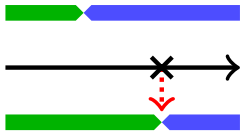
- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps

lift



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

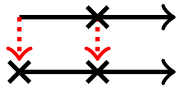
last



- Refers to previous value of a stream
- Recursion

Temporal Stream-based Specification Language

default, defaultFrom

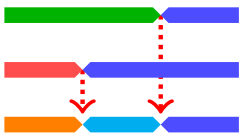


- Initialize streams
- Start of recursion

time

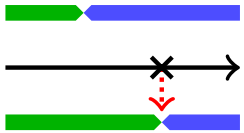
- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps

lift



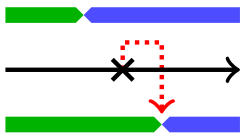
- Lifts standard functions to streams
- Used to manipulate data, events, ...

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

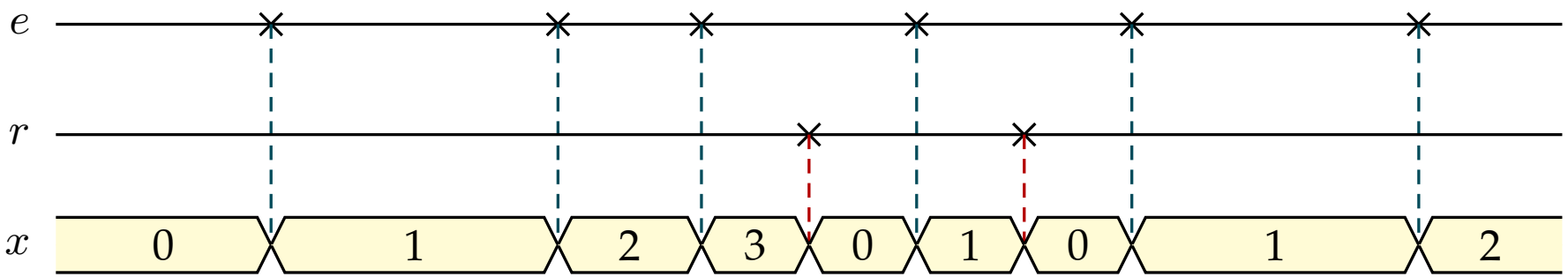
TeSSLa: easy example

Let $a \in \mathcal{S}_{\text{Int}}$ be an input stream, $x, y \in \mathcal{S}_{\text{Int}}$ be intermediate streams and $z \in \mathcal{S}_{\text{Int}}$ be the output stream.

TeSSLa specifications can be seen as a system of equations:

```
x = last(z, a)
y = default(x, 0)
z = y + a
```

TeSSLa by Example



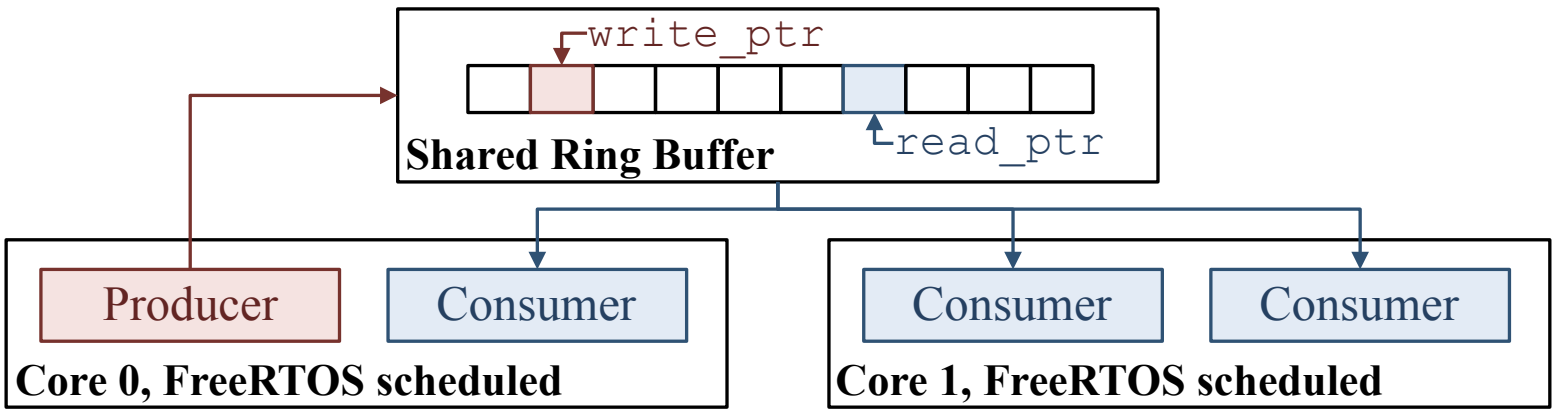

```
def x := eventCount(e, reset = r)
```

TeSSLa Standard Library

Counting events until reset

```
def eventCount(values, reset) :=
  default(
    if default(time(reset) > time(values), false)
    then 0
    else if default(time(reset) == time(values), false)
    then 1
    else last(eventCount, values) + 1
  , 0)
```

FPGA configuration and case study



Case study properties

Property a)

```
-- INPUT STREAM DEFINITION
def ptrChanged := merge(codeLine("core0.c:27"),
                        codeLine("core1.c:27"))
def stop :=  functionCalls("core0.c:stopConsumers")
def start := functionCalls("core0.c:startConsumers")

-- PROPERTY
def clk := merge(stop, ptrChanged, start)
def output :=
  monitor("always(stop implies
     (not(ptrChanged) until start))",
     step := clk)
out output
```

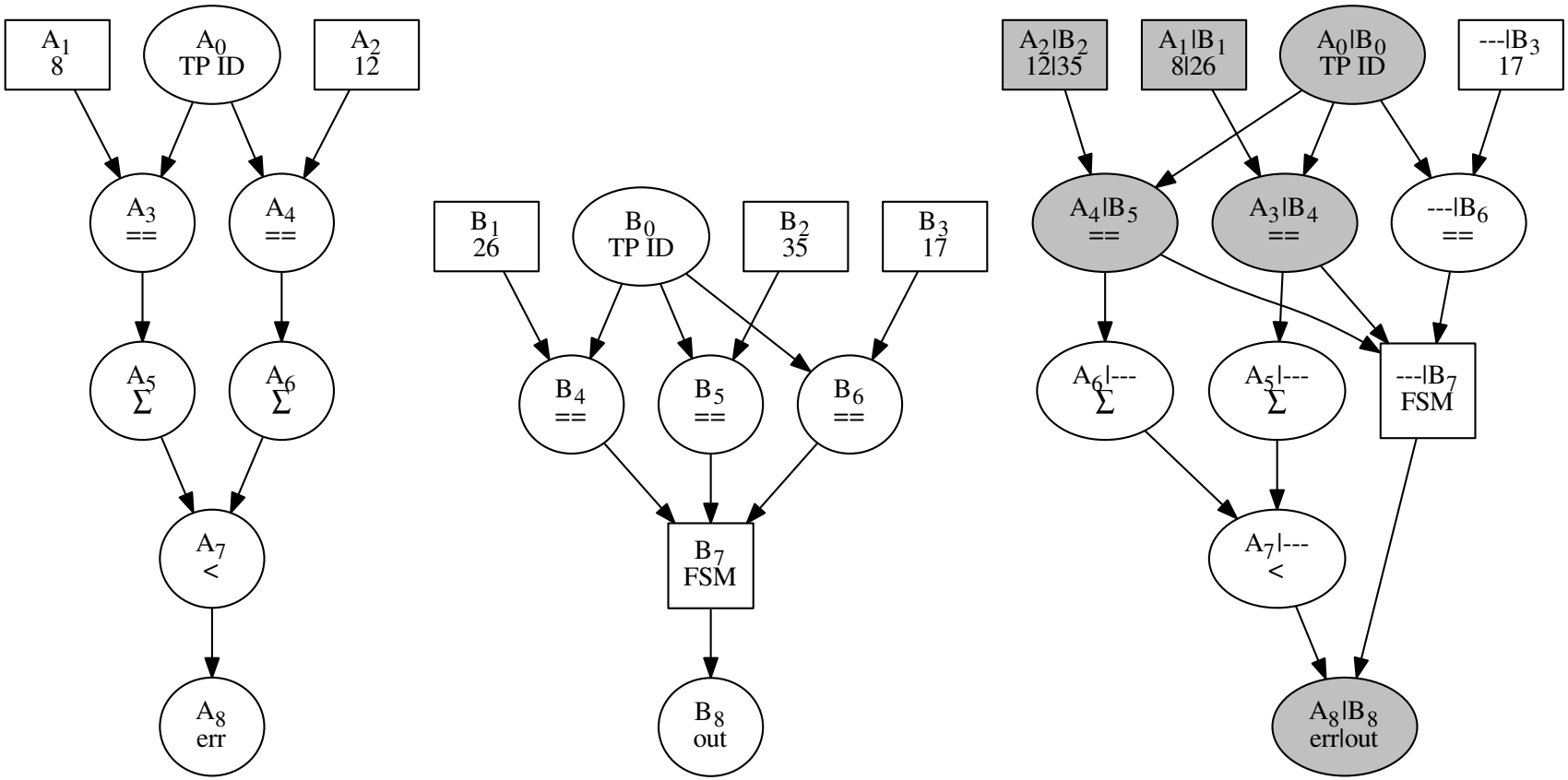
Case study properties

Property b)

```
-- INPUT STREAM DEFINITION
def write := codeLine("core0.c:37")
def read0 := codeLine("core0.c:24")
def read1 := codeLine("core1.c:24")

-- PROPERTY
def err :=
eventCount(read0) + eventCount(read1) > eventCount(write)
out err
```

FPGA configuration example



FPGA reconfiguration

Property c)

```
-- MACRO CALL
out doubleRead(
  read := codeLine("core0:24"),
  ptrChanged := codeLine("core0:27"))
```

where `doubleRead` is a macro defined as follows:

```
-- MACRO DEFINITION
def doubleRead(read, ptrChanged) := {
  def clk := merge(read, ptrChanged)
  monitor("always(read implies
    next(not(read) until ptrChanged))",
    step := clk)
}
```

Same for core 1 by replacing the zeros with ones.

Outlook – TeSSLa

In the future, the following features would be desirable for TeSSLa:

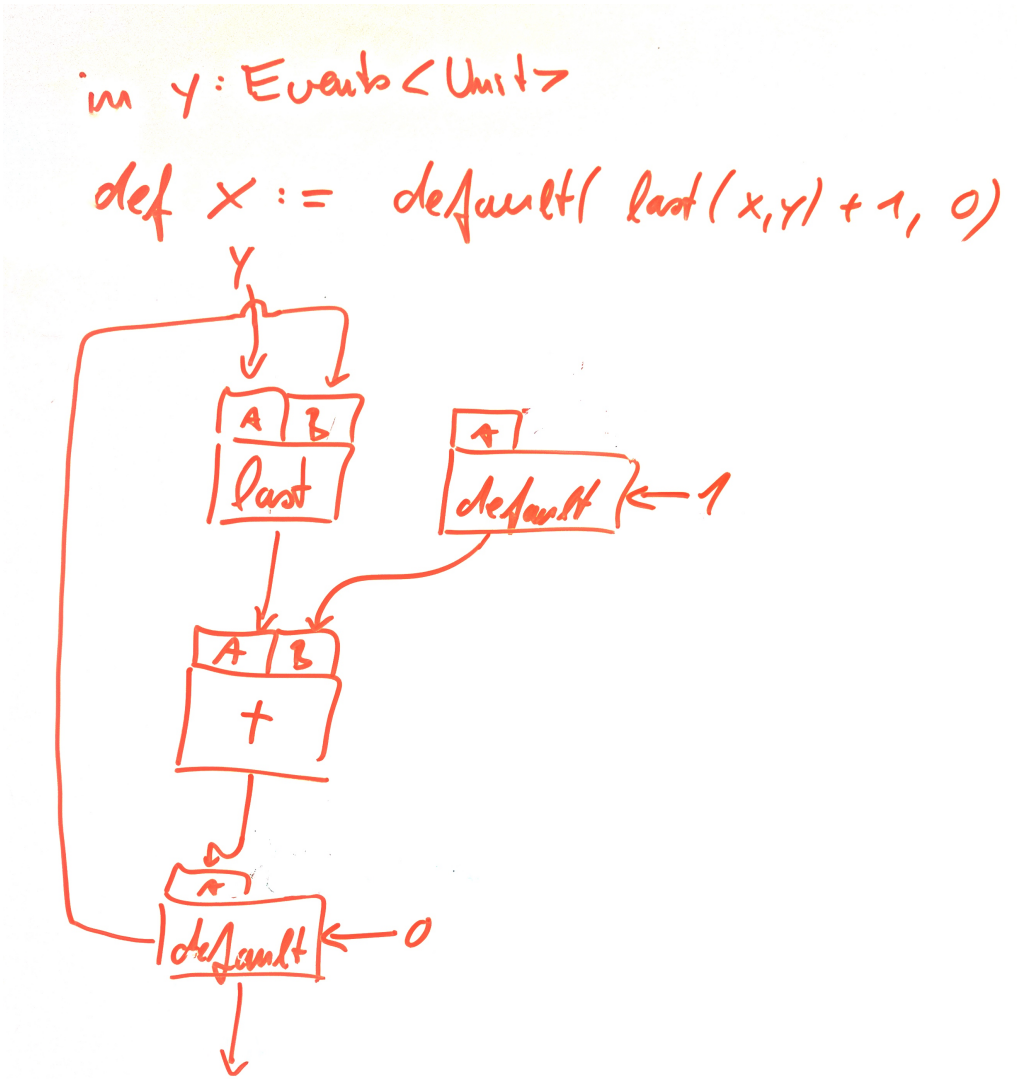
- High level data structures like maps and queues
- Possibility to handle gaps and uncertainties in streams
- ...

This will result in

$$\underset{\text{Bounded maps / queues, ...}}{\text{TeSSLa}_{\text{Hardware}}} \subset \underset{\text{Effectively monitorable}}{\text{TeSSLa}_{\text{Software}}} \subseteq \text{TeSSLa}$$

Outlook – FPGA Monitors

- More flexible approach
- Configurable RV-Engines instead of graphs
- Core functions instead of high level ones



Appendix

Temporal Stream-based Specification Language

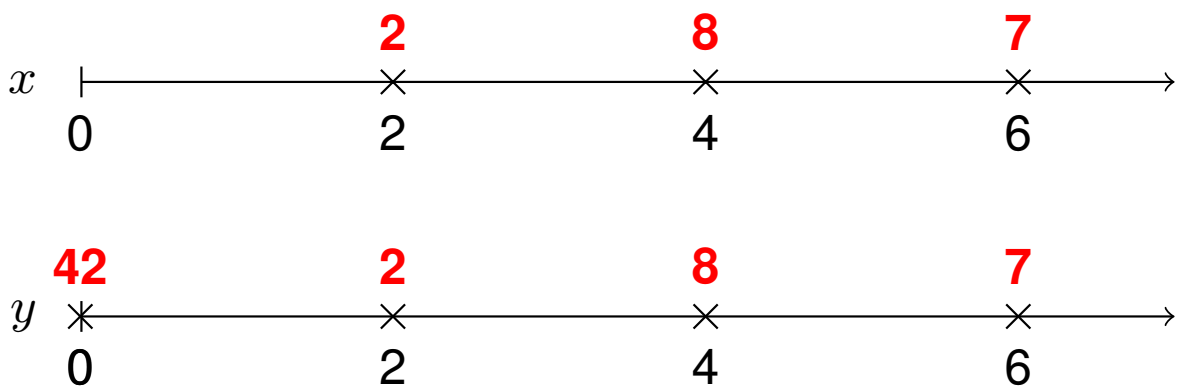
- Reasons over a set of **non-synchronized real-time streams**
- **Declarative** style: Specification rather than implementation
- **Modularity**: Allowing abstractions based on six core operators
- **Time** as first-class citizen
- Abstraction for both, **events** and **signals**
- **Recursion** to reason about the past
- Implementable with **finite memory**

TeSSLa Core: Default

Add a value at time zero if not present.

$$\mathsf{default}_{\mathbb{D}} : \mathbb{D} \times \mathcal{S}_{\mathbb{D}} \to \mathcal{S}_{\mathbb{D}}$$

$$y = \mathsf{default}(42, x)$$

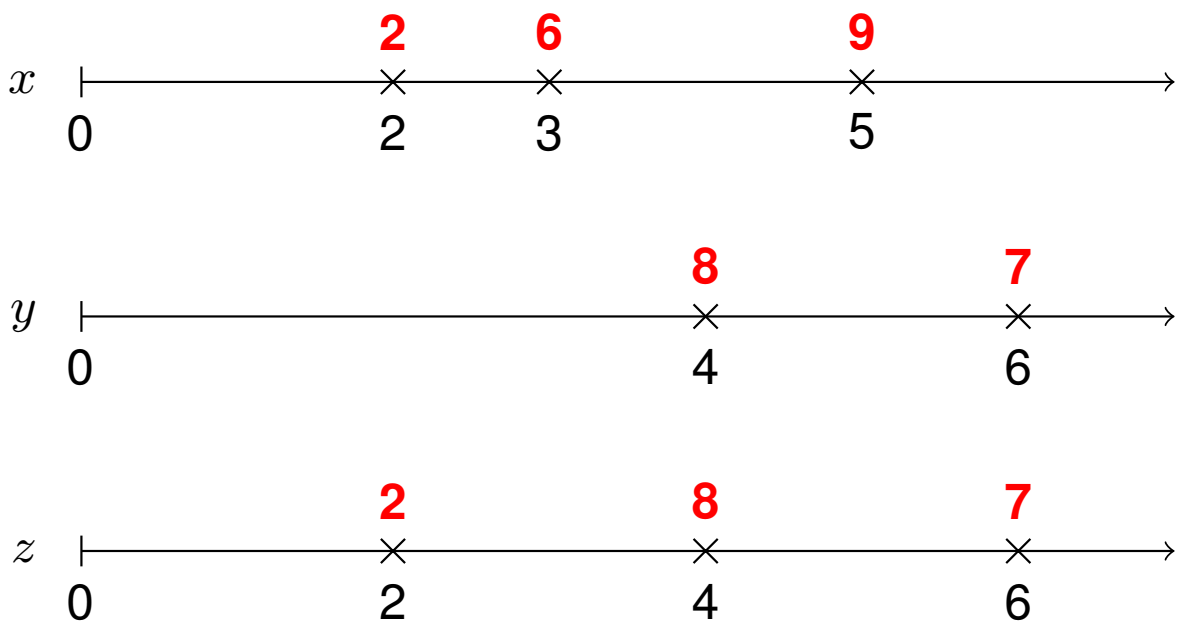


TeSSLa Core: Default From

Add first value from another stream if nothing happened before.

$$\mathsf{defaultFrom}_{\mathbb{D}} : \mathcal{S}_{\mathbb{D}} \times \mathcal{S}_{\mathbb{D}} \to \mathcal{S}_{\mathbb{D}}$$

$$z = \mathsf{defaultFrom}(x, y)$$

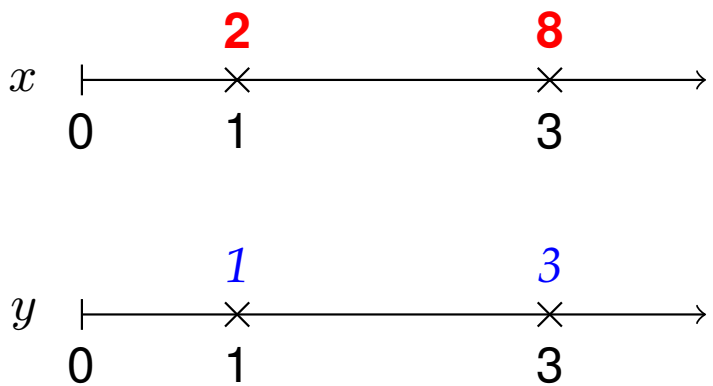


TeSSLa Core: Time

Get the time of events.

$$\mathsf{time}_{\mathbb{D}} : \mathcal{S}_{\mathbb{D}} \to \mathcal{S}_{\mathbb{T}}$$

$$y = \mathsf{time}(x)$$



TeSSLa Core: Lifted Addition

Lift arbitrary functions to streams with signal semantics.

$$\mathsf{lift}_{\mathbb{D}_1,\ldots,\mathbb{D}_n,\mathbb{D}'} : (\mathbb{D}_1 \times \cdots \times \mathbb{D}_n \rightharpoonup \mathbb{D}') \to (\mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n} \to \mathcal{S}_{\mathbb{D}'})$$

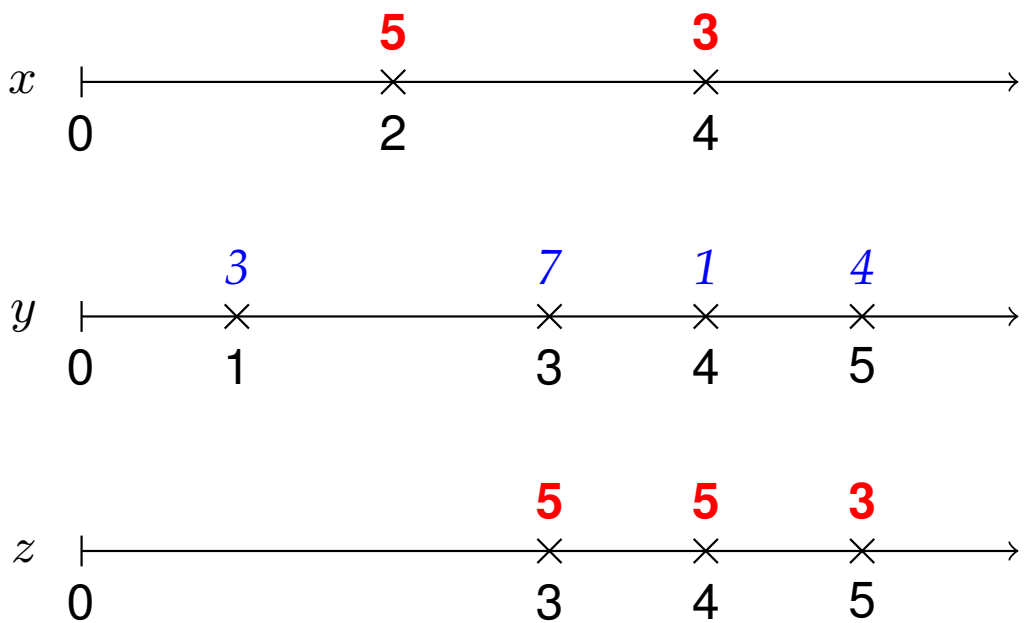
$$\mathsf{add} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z},$$
$$z = \mathsf{lift}(\mathsf{add})(x, y)$$

TeSSLa Core: Last

Get the last value of a stream at a certain point in time.

$$\mathsf{last}_{\mathbb{D},\mathbb{D}'} : \mathcal{S}_{\mathbb{D}} \times \mathcal{S}_{\mathbb{D}'} \to \mathcal{S}_{\mathbb{D}}$$

$$z = \mathsf{last}(x, y)$$

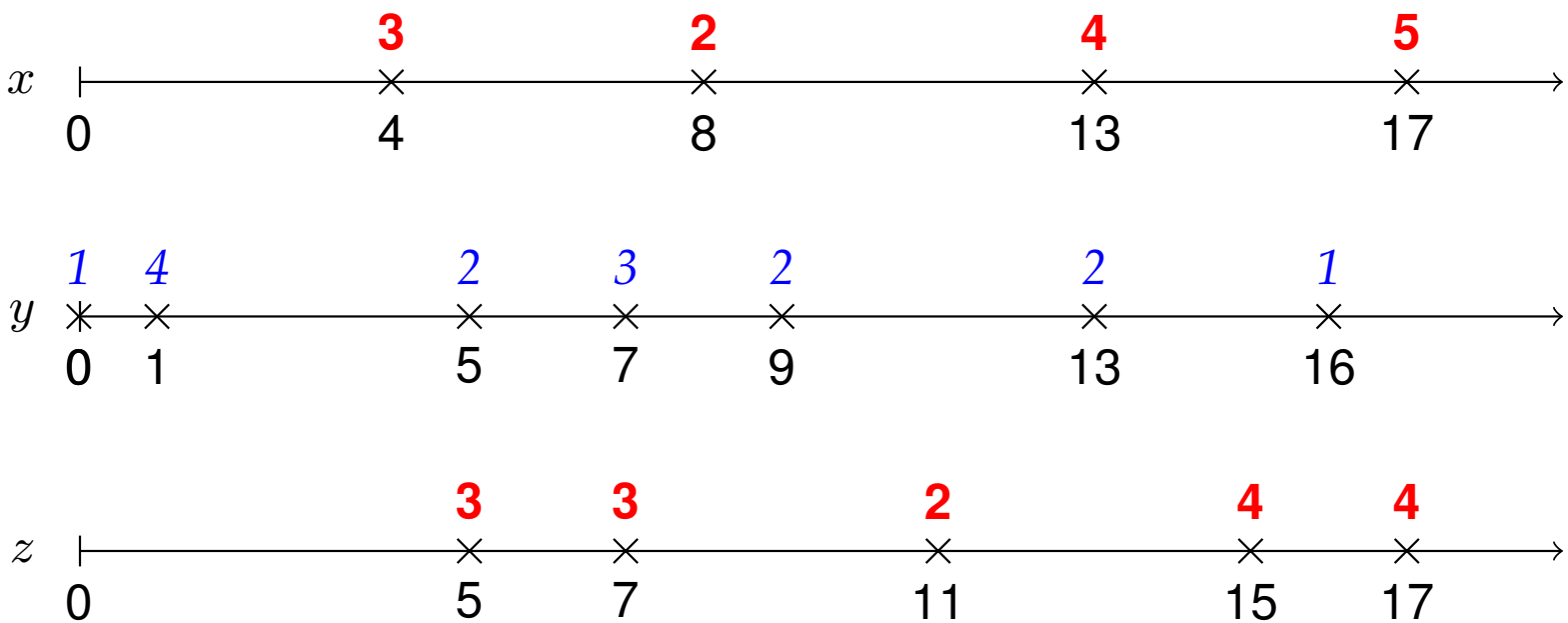


TeSSLa Core: Delayed Last

Delay a stream by delays given in a stream.

$$\mathsf{delayedLast}_{\mathbb{D}} : \mathcal{S}_{\mathbb{D}} \times \mathcal{S}_{\mathbb{T}\setminus\{0\}} \to \mathcal{S}_{\mathbb{D}}$$

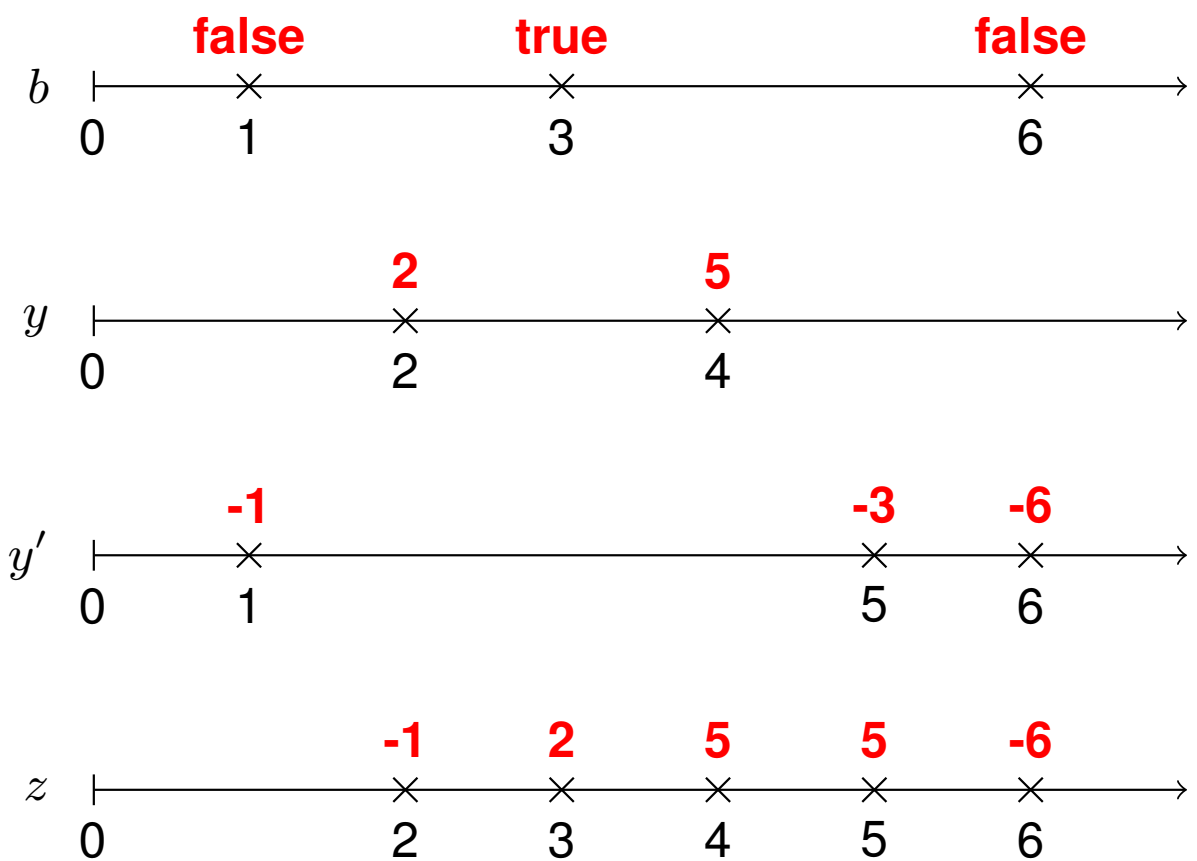
$$z = \mathsf{delayedLast}(x, y)$$



TeSSLa Core: Lifted If-Then-Else

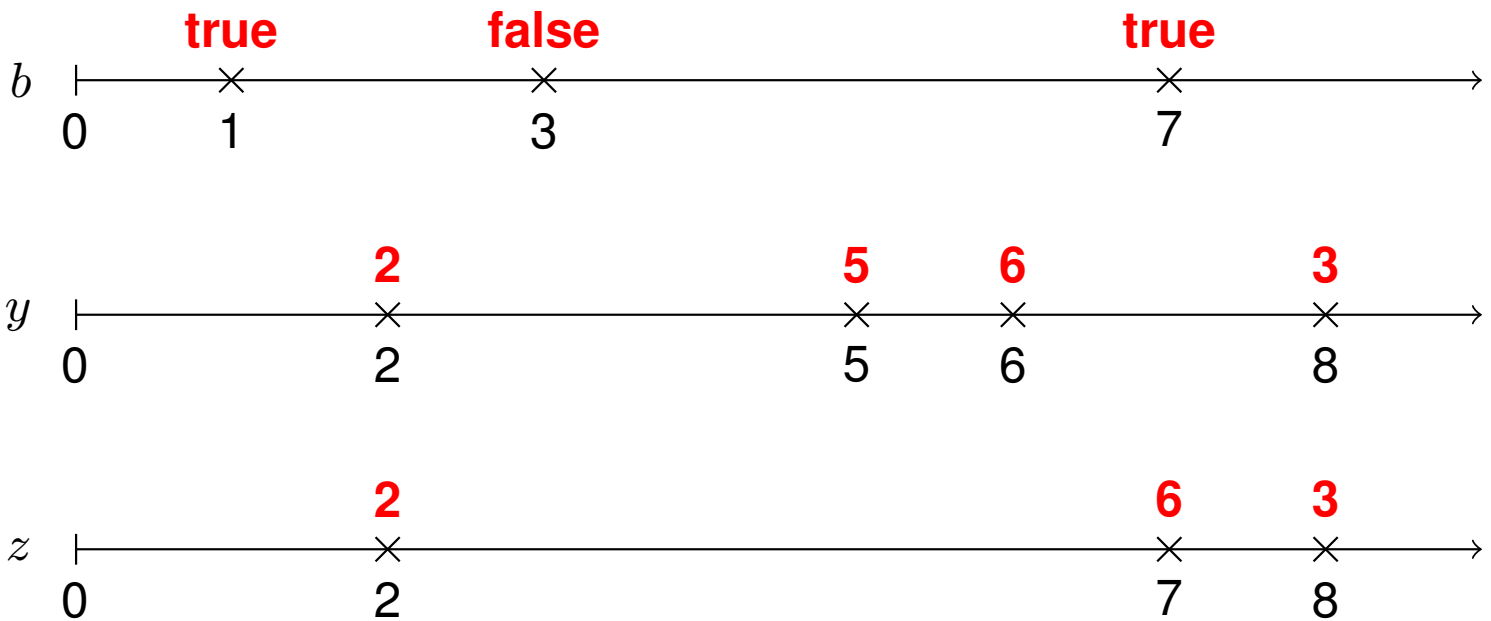
$$\mathsf{ifthenelse}_A : \mathbb{B} \times A \times A \to A$$

$$z = \mathsf{lift}(\mathsf{ifthenelse})(b, y, y')$$

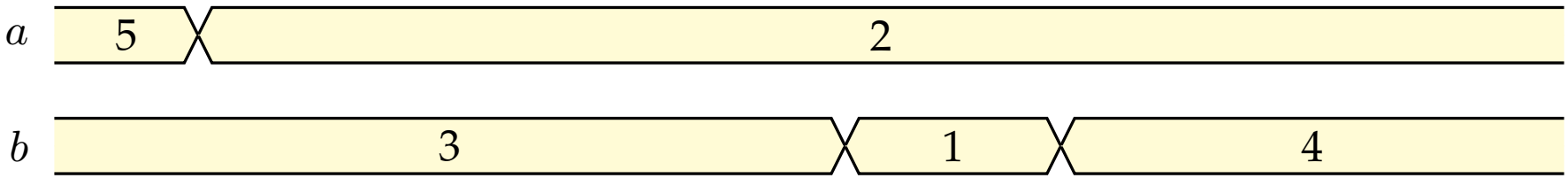


TeSSLa Core: Lifted If-Then

$$\mathsf{ifthen}_A : \mathbb{B} \times A \rightharpoonup A$$
$$z = \mathsf{lift}(\mathsf{ifthen})(b, y)$$

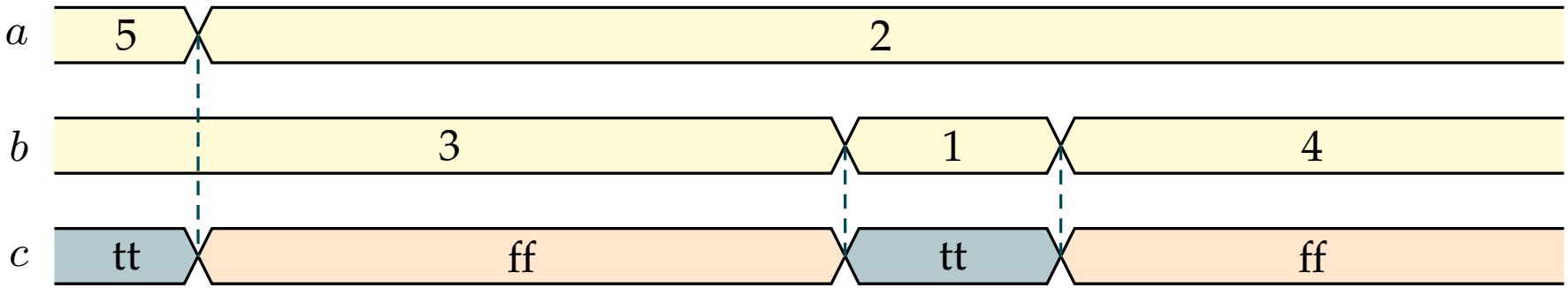


TeSSLa by a More Complex Example



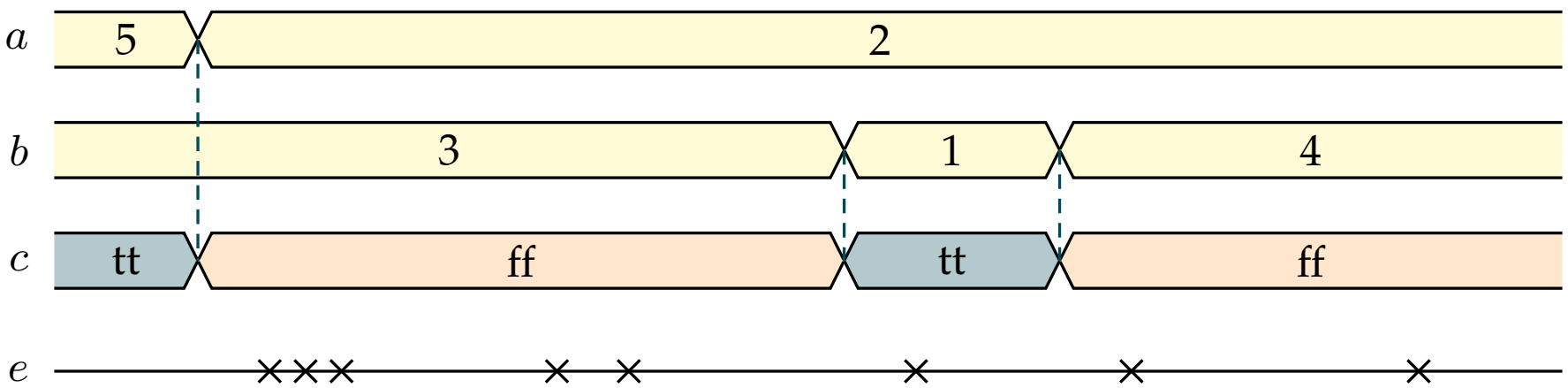

```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
```

TeSSLa by a More Complex Example



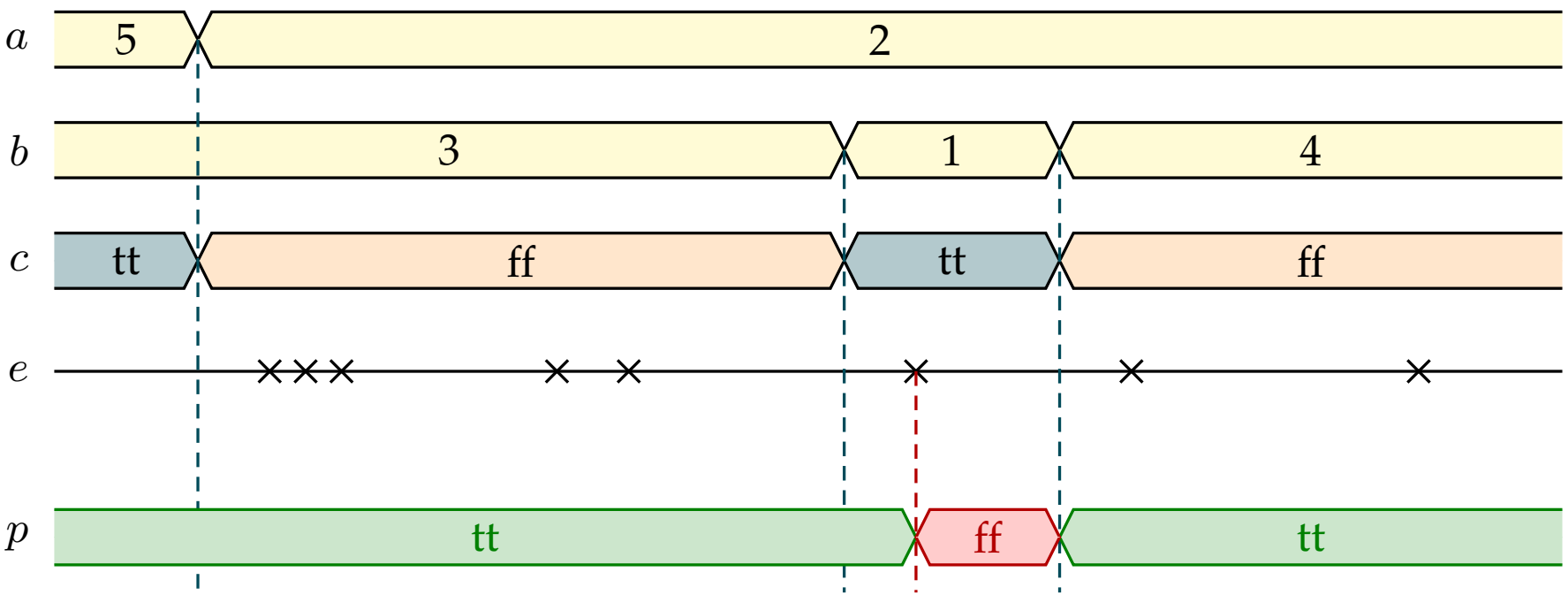

```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
```

TeSSLa by a More Complex Example



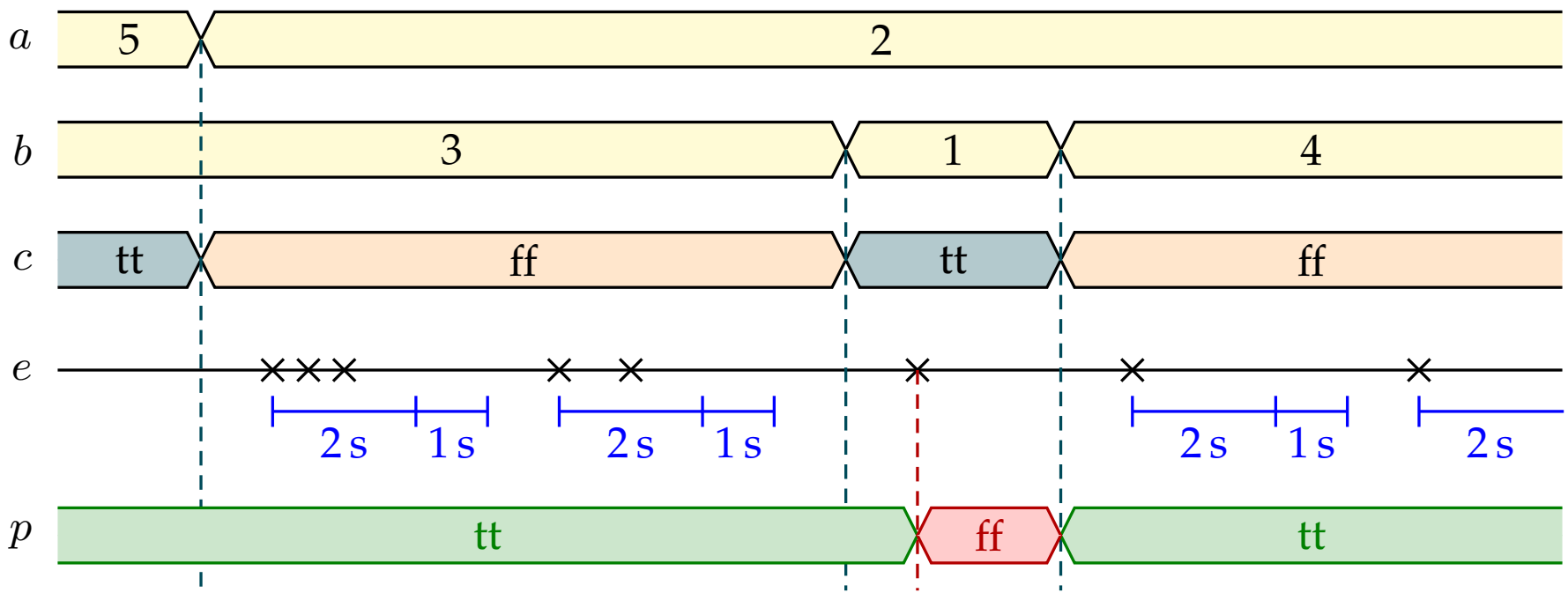

```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
```

TeSSLa by a More Complex Example



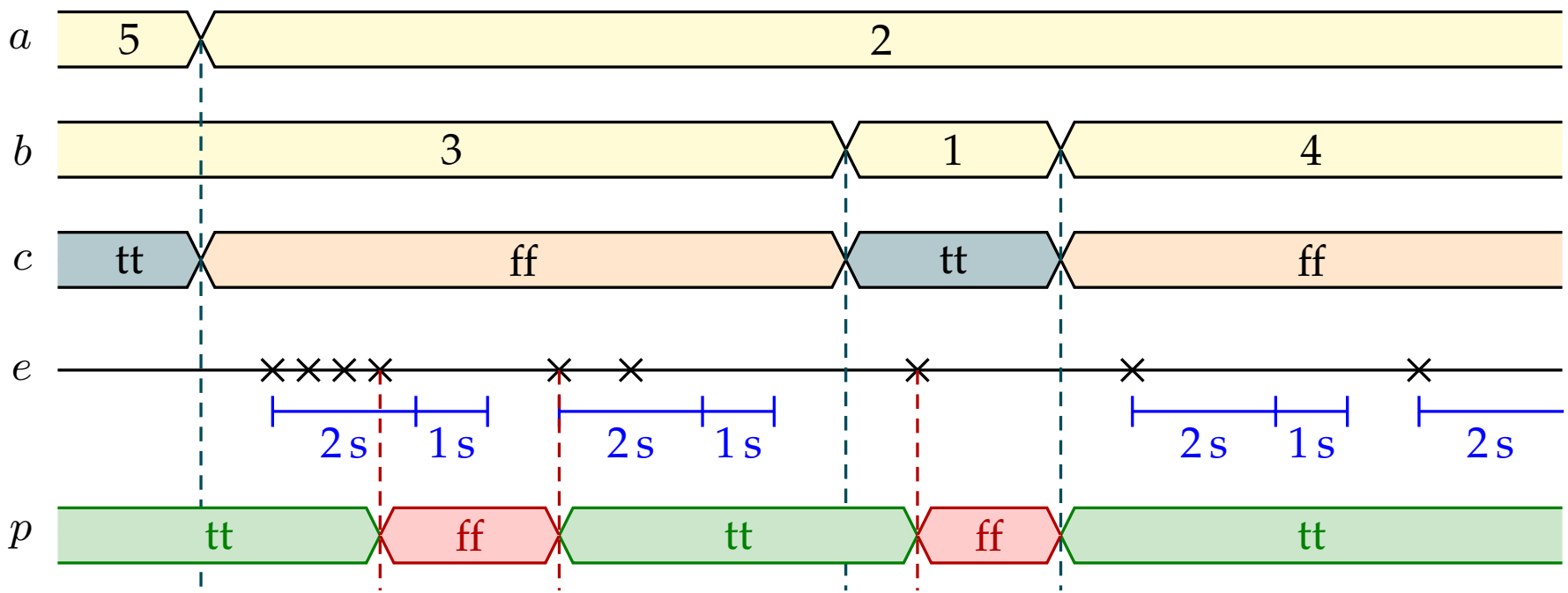

```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
 true
```

TeSSLa by a More Complex Example



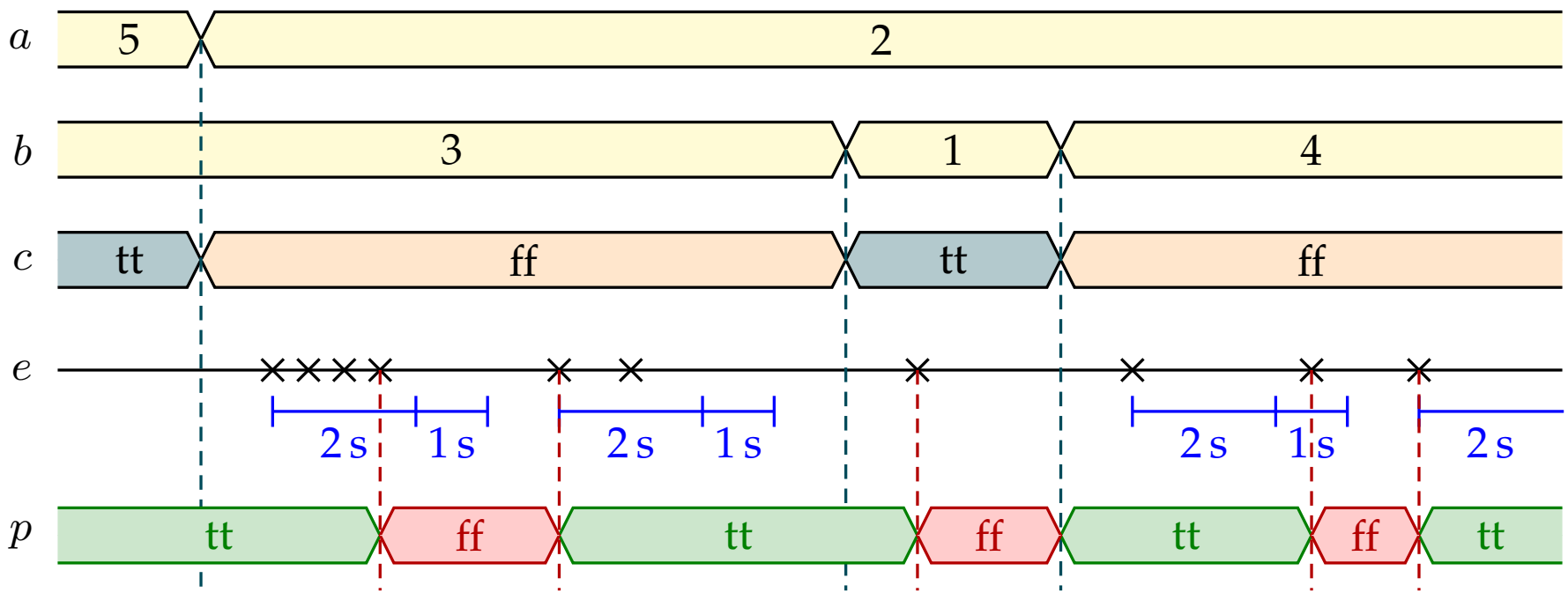

```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
 bursts(e, burstLength = 2s,
ingPeriod = 1s,
tAmount = 3s)
```

TeSSLa by a More Complex Example




```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
 bursts(e, burstLength = 2s,
ingPeriod = 1s,
tAmount = 3s)
```

TeSSLa by a More Complex Example




```
slide<2->def c := a > b->def p := if c
 noEvent(e, since = rising(c))
 bursts(e, burstLength = 2s,
ingPeriod = 1s,
tAmount = 3s)
```

TeSSLa Standard Library

Detect rising edge of a signal

```
rising(condition) :=
ondition && !last(condition) then ()
```

No event happened since last reset

```
noEvent(on, since) :=
tCount(on, reset = since) == 0
```

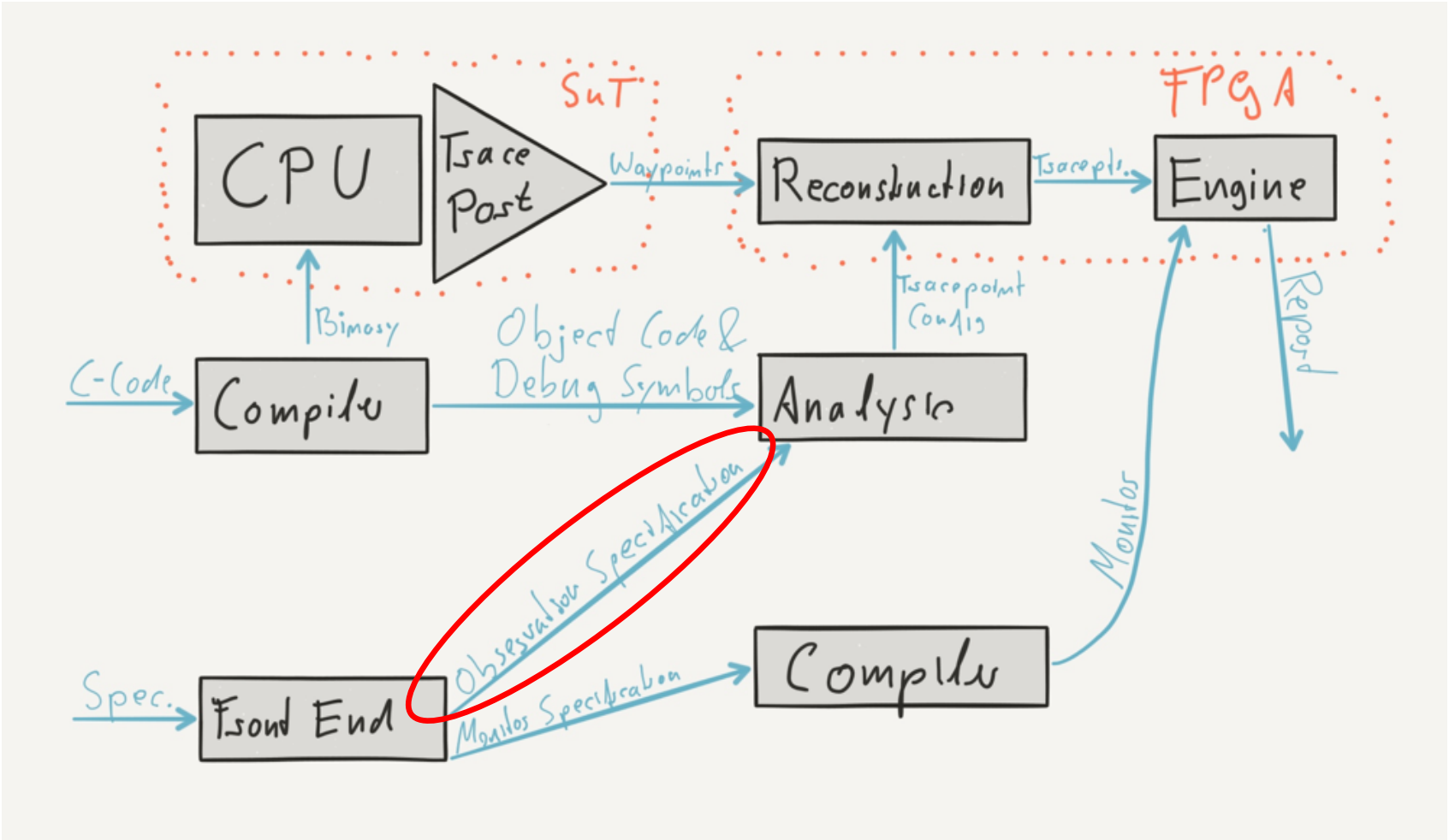
TeSSLa Standard Library

Burst Pattern based on AUTOSAR Timing Extension

```
bursts(e, burstLength, waitingPeriod,
urstAmount) := {
burstStarts :=
ultFrom(
ime(e) - last(time(burstStarts), e) >=
tLength + waitingPeriod
 e,

tCount(e, reset = burstStarts) <= burstAmount &&
(e) < time(burstStarts) + burstLength
```

Outlook: Observation Specification



Outlook: Observation Specification

Elements of the Source Code

- Program Line
- Entering / Leaving a Function
- Reading / Writing Variables
- ...

Elements of the Binary

- Program Counter Address
- CALLS and RETURNS
- Specific Operations (e.g. Floating Point Operations)
- ...