UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

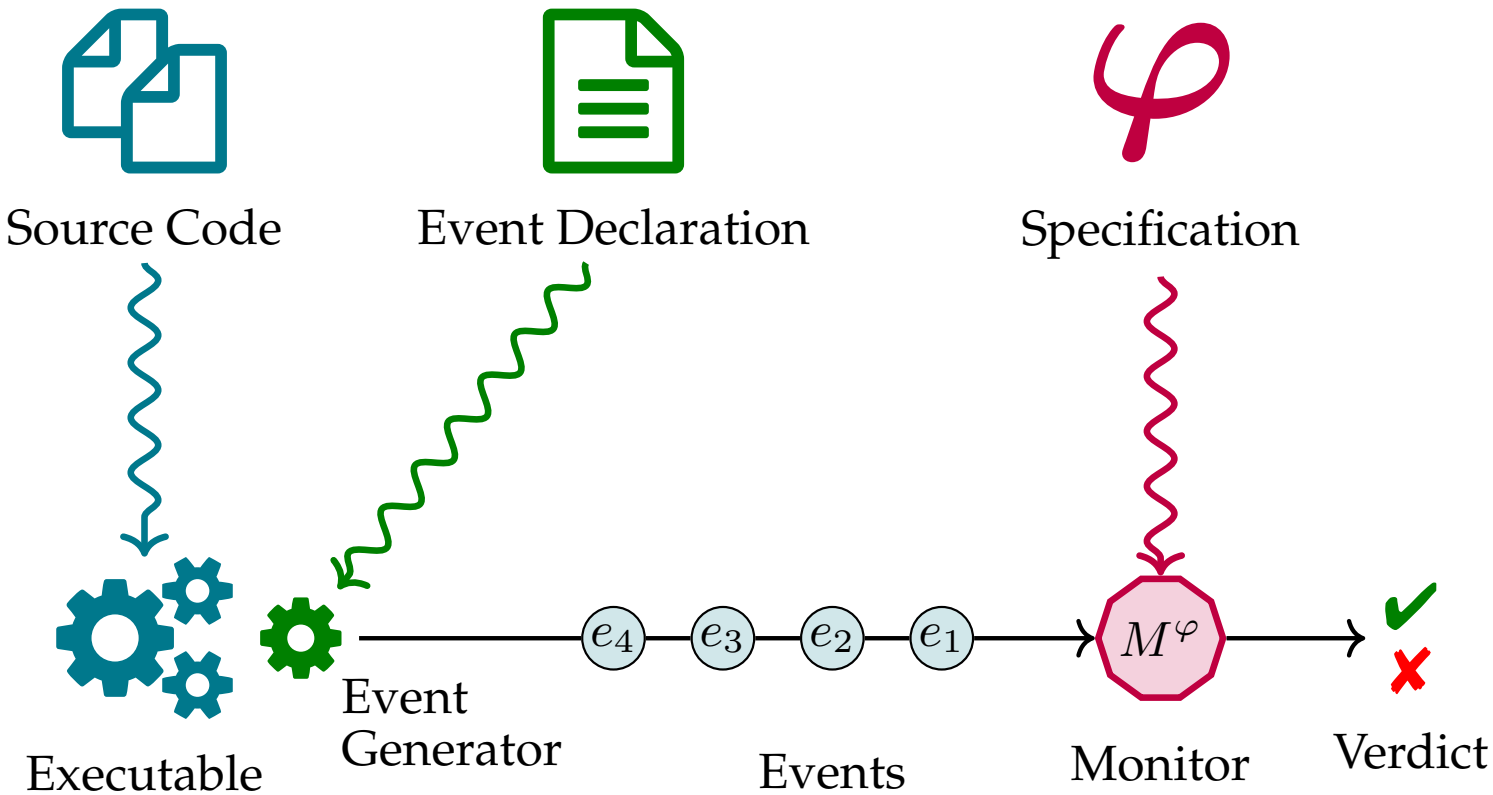
TOOL PAPER: Tessla-ROS-Bridge Runtime Verification of Robotic Systems

Marian Johannes Begemann **Hannes Kallwies** Martin Leucker
Malte Schmitz

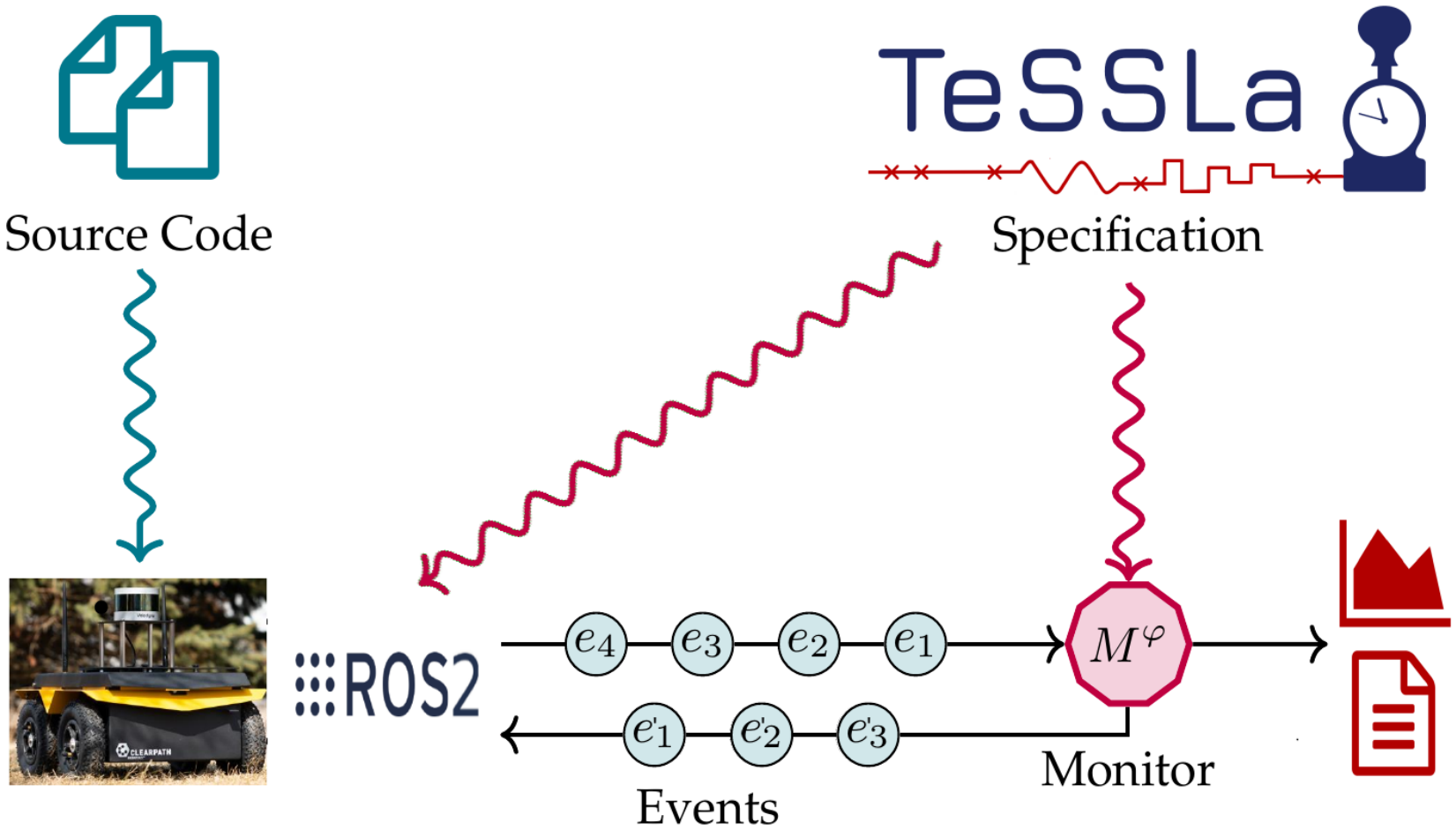
Institute for Software Engineering and Programming Languages,
University of Lübeck, Germany

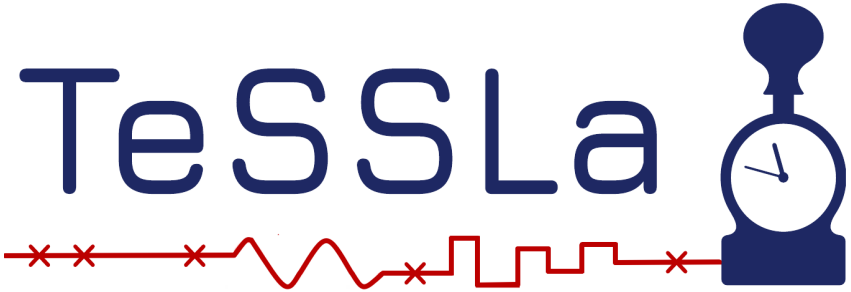
20th International Colloquium on Theoretical Aspects of Computing, December 2023

Traditional (Stream) Runtime Verification



In this paper: Extension to robot systems





TeSSLa is a general purpose Stream-based Specification language:

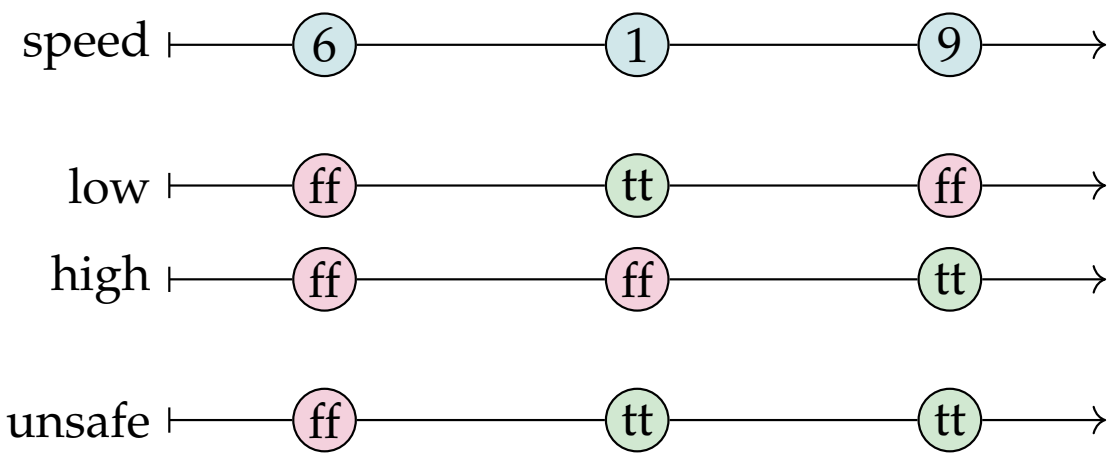
Every monotonous, continuous and future-independent stream transformation function f can be specified in TeSSLa

Possible fields of application:

- Online Monitoring
- Logfile Analysis
- Event pattern generation
- Analysis of the specification
- ...

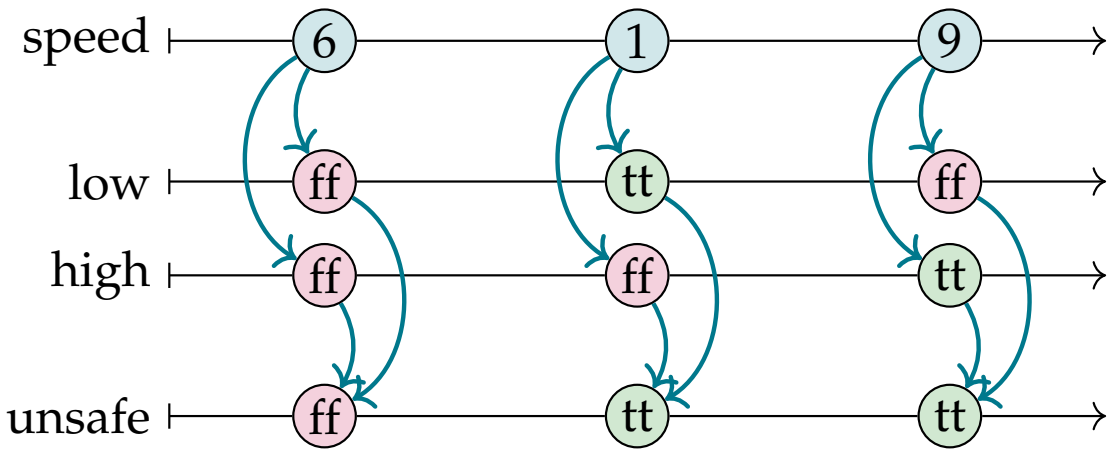
Basic concept: Combining streams

Correctness property: Speed is between 2 and 8.



Basic concept: Combining streams

Correctness property: Speed is between 2 and 8.



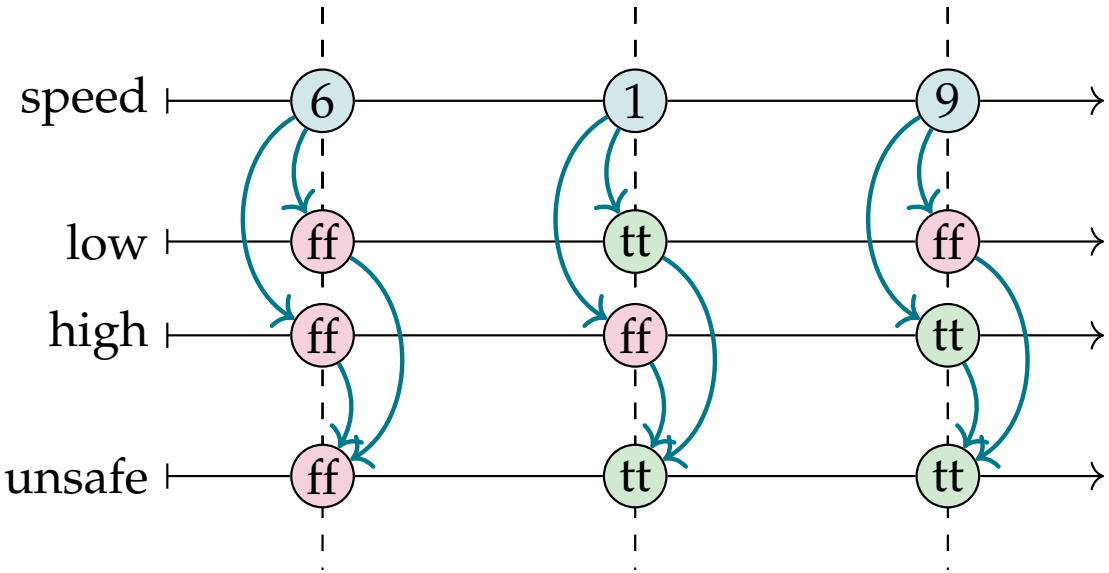

```
in speed: Events[Int]

def low = (speed < 2)
def high = (speed > 8)
def unsafe = low || high

out unsafe
```

Basic concept: Synchronous streams

Correctness property: Speed is between 2 and 8.



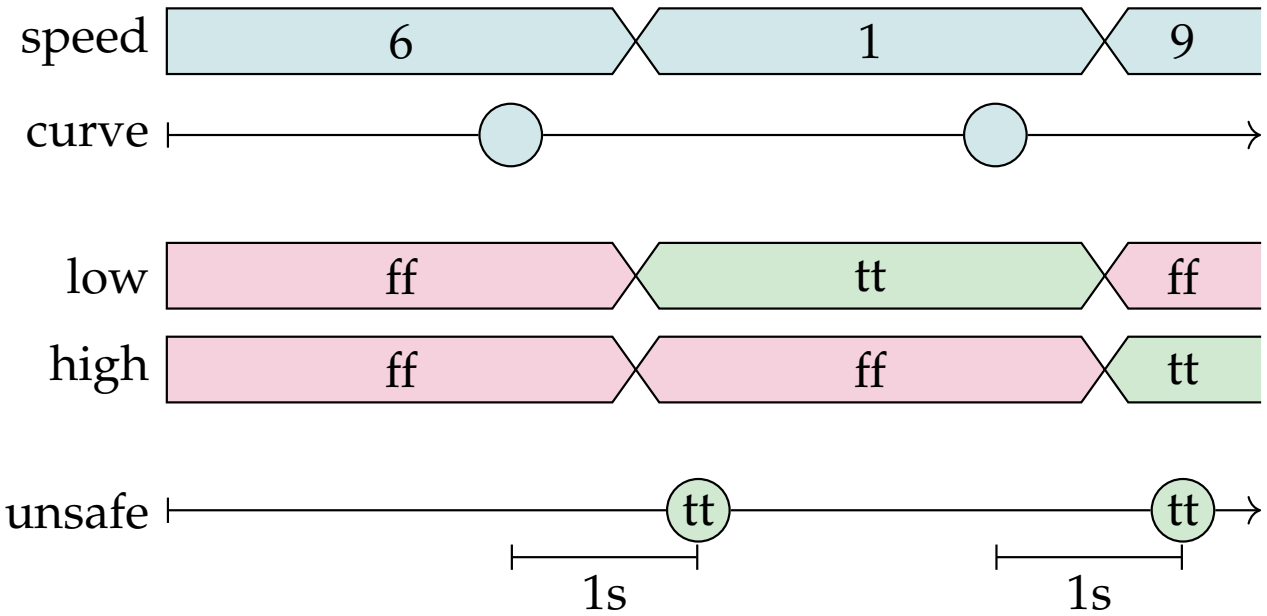

```
in speed: Events[Int]

def low = (speed < 2)
def high = (speed > 8)
def unsafe = low || high

out unsafe
```

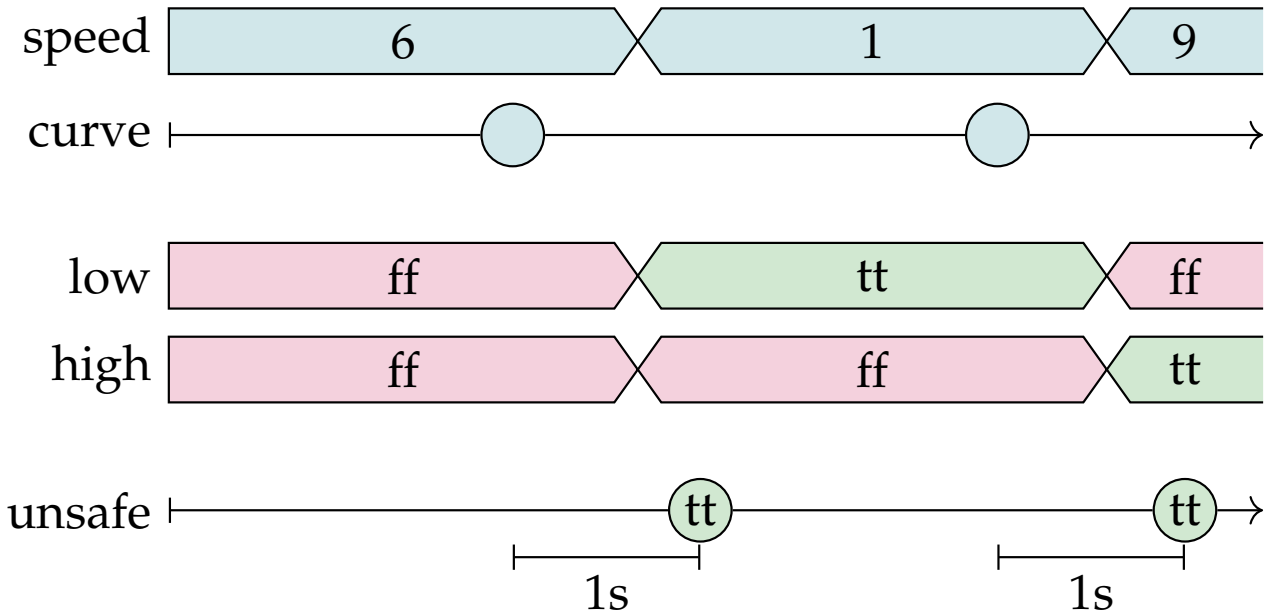
Basic concept: Asynchronous streams

Correctness property: Speed is between 2 and 8, one second after robot intends driving a curve.



Basic concept: Asynchronous streams

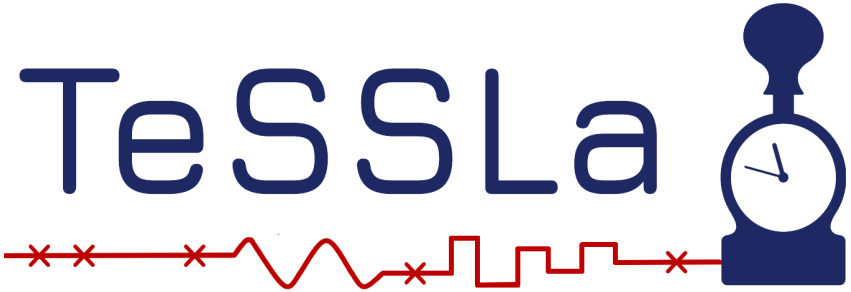
Correctness property: Speed is between 2 and 8, one second after robot intends driving a curve.



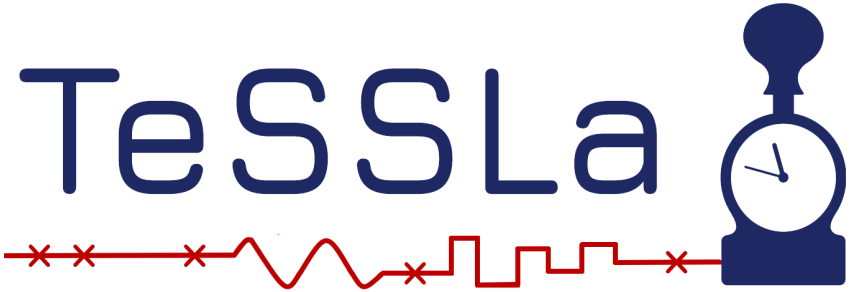

```
in speed: Events[Int]
in curve: Events[Unit]

def low = (speed < 2)
def high = (speed > 8)
def unsafe = on(low || high, delay(1s, curve))

out unsafe
```



- Abstractions for both **events** and **signals**
- Description of **asynchronous streams**
- **Time** as first-class citizen
- Useful for description of **Cyber Physical Systems**



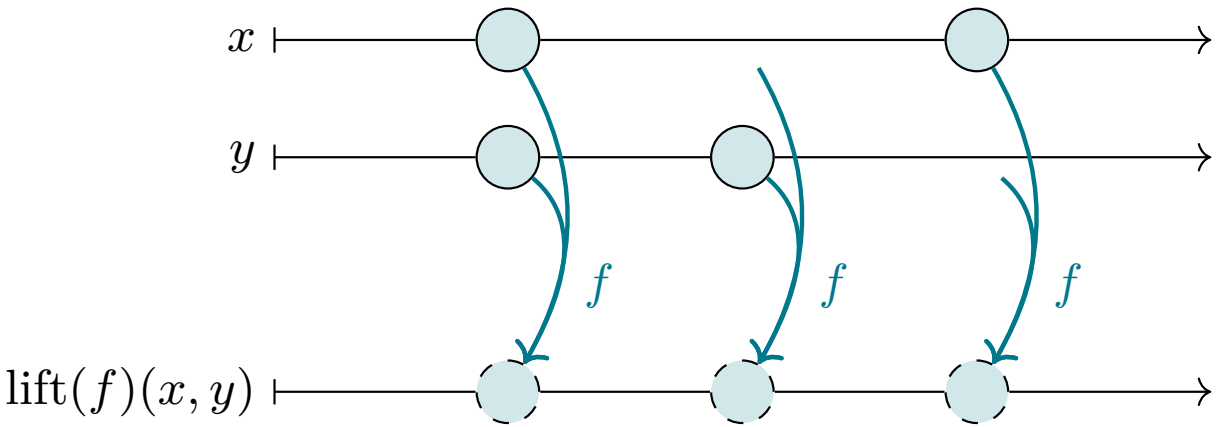
TeSSLa: Language and features

Language based on **five core operations** plus

- Type system
- Macro system
- Module system
- Standard library and several user libraries
- Meta Data/annotation concept

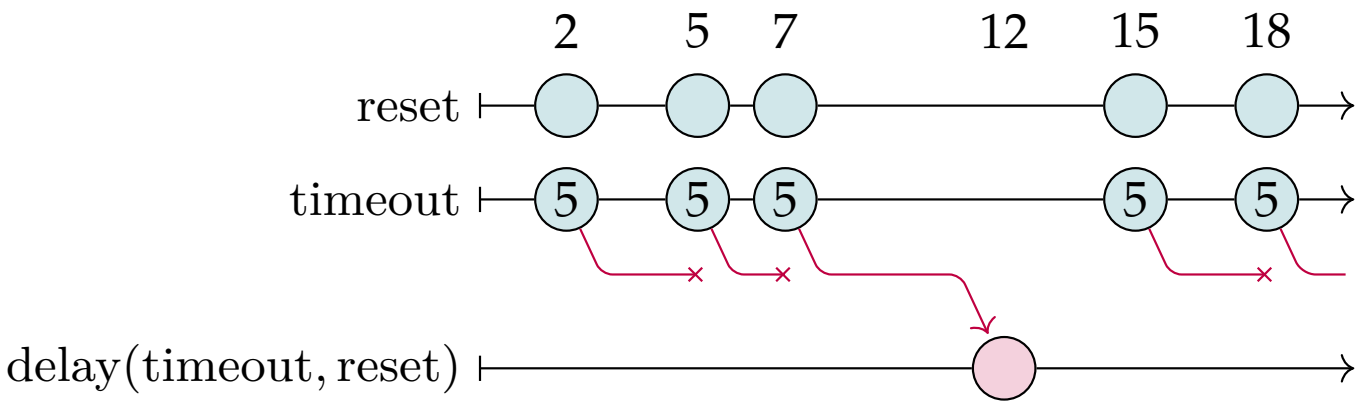
TeSSLa core operations: Lift

- *Lift* applies a function to the current events on a certain number of streams
- e.g. adds two numerical event values



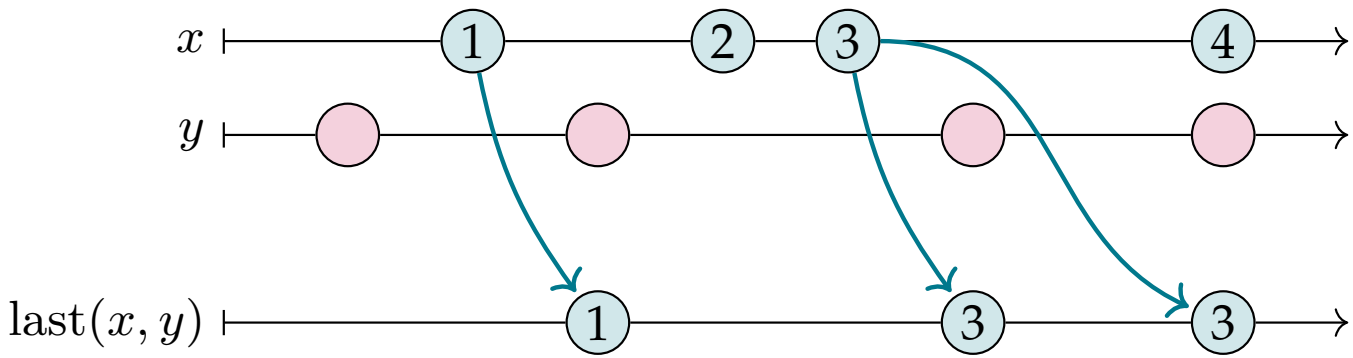
TeSSLa core operations: Delay

- *Delay* creates a new event some time after a reset event
- Possibility to create output events at timestamps without input events



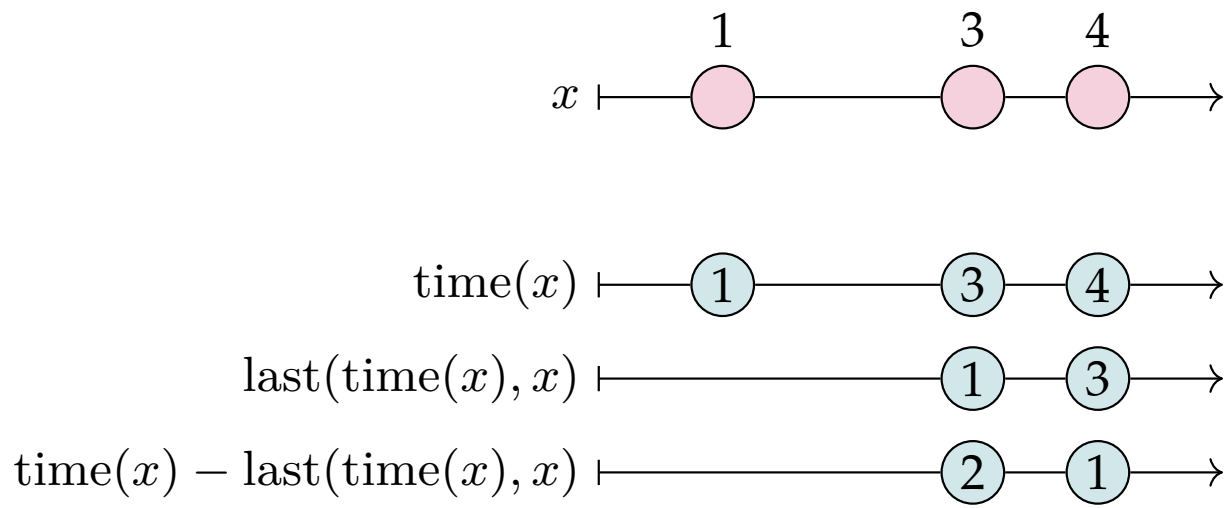
TeSSLa core operations: Last

- *Last* allows to access the values of events on one stream that occurred strictly before the events on another stream
- Important for accessing streams with signal semantics



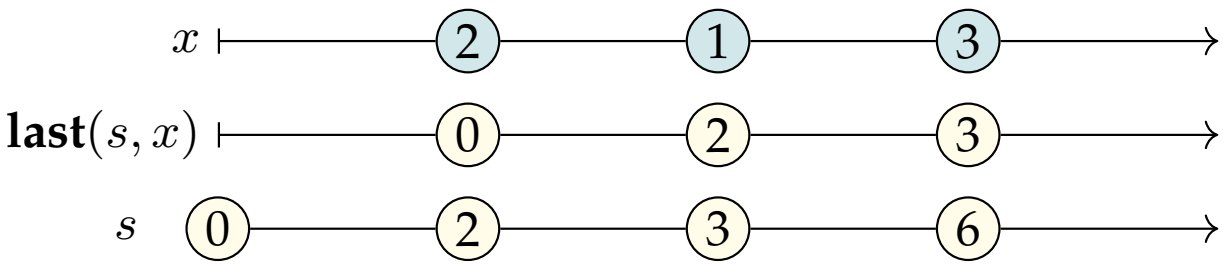
TeSSLa core operations: Time

- *Time* provides access to the timestamps of events
- Produces events carrying their timestamps as data value
- Hence all operators for data values can be applied to timestamps.



Recursive Equations in TeSSLa

$$s = \mathbf{lift}(merge)(\mathbf{last}(s, x) + x, 0)$$



Macro-System

- Possibility to extend minimal language core by arbitrary functions

Macro Definition Fold

```
def fold[T,R](stream: Events[T], init: R,
              f: (Events[R],Events[T]) => Events[R]) = result
where
{
    def result: Events[R] = merge(f(last(result, stream),
                                     stream), init)
}
```

Usage of Fold

```
def y = fold(x, 0, (c: Int, x: Int) => c+x)
```

Module System

Modules

```
module myModule {
  module mySubmodule {
    def myCount[A](a: Events[A]) := c where {
      def c: Events[Int] := merge(last(c, a) + 1, 0)
    }
  }
}

in x: Events[Unit]
def y := myModule.mySubmodule.myCount(x)
out y
```

Module System

Modules

```
module myModule {
  module mySubmodule {
    def myCount[A](a: Events[A]) := c where {
      def c: Events[Int] := merge(last(c, a) + 1, 0)
    }
  }
}

in x: Events[Unit]
def y := myModule.mySubmodule.myCount(x)
out y
```

⇒ Possibility to create TeSSLa libraries.

Standard & User Libraries

Standard Library Defines a high number of macros to make the usage of TeSSLa comfortable

- Basic operations: `Merge, Signal Lift, Const, Filter, ...`
- Aggregation functions: `Minimum, Maximum, Fold, Reduce, ...`
- Common datastrucutre functions: `Set.contains, Map.getOrElse, ...`
- Application specific functions: `Burst-Pattern recognition, Event-Chain recognition, ...`

Standard & User Libraries

Standard Library Defines a high number of macros to make the usage of TeSSLa comfortable

- Basic operations: `Merge, Signal Lift, Const, Filter, ...`
- Aggregation functions: `Minimum, Maximum, Fold, Reduce, ...`
- Common datastrucutre functions: `Set.contains, Map.getOrElse, ...`
- Application specific functions: `Burst-Pattern recognition, Event-Chain recognition, ...`

User Libraries e.g. for

- special logics
- AUTOSAR Timex extension

TeSSLa Language: Typesystem

- Built-in basic types can be extended by user-defined types
- Supports externally defined nominal types
- Record types
- Generics

Supported basic types:

- Unit
- Int
- Float
- Boolean
- String

Supported complex datastructures:

- Lists
- Sets
- Maps

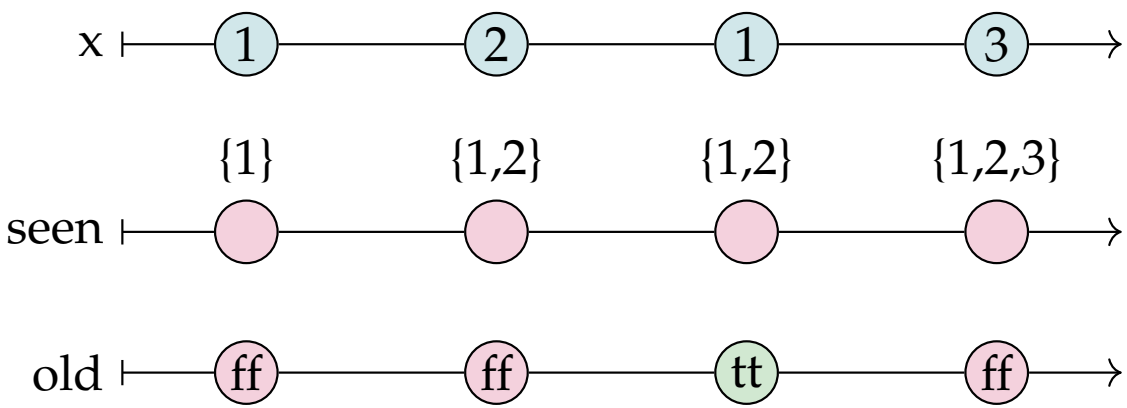
Complex datastructures

Complex datastructures

```
in x: Events[Int]

def seen: Events[Set[Int]] := fold(x, Set.empty[Int], Set.add)

out Set.contains(last(seen, x), x) as old
```



Meta Data/Annotations

Possibility to pass event declaration to connected tools:

- `@InstFunctionCall(func\_name)`
- `@VisSignal`
- `@RosSubscription(topic, datatype, qos_profile)`
- `@RosPublisher(topic, datatype, qos_profile)`
- ...

```
@RosSubscription("/sensor1", "int64", "10")
in x : Events[Int]

[...]

@RosPublisher("/actor1", "int64", "10") @VisSignal
out y
```

TeSSLa-ROS-Integration

Idea: Use TeSSLa to monitor robot systems

TeSSLa-ROS-Integration

Idea: Use TeSSLa to monitor robot systems

ROS (Robot Operating System):

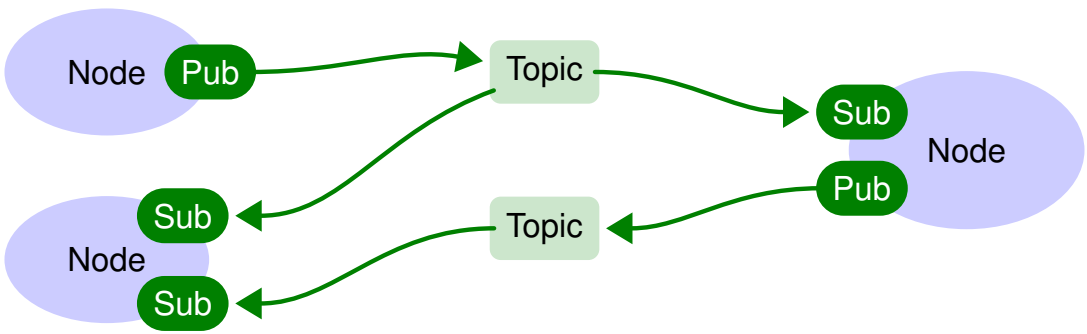
TeSSLa-ROS-Integration

Idea: Use TeSSLa to monitor robot systems

ROS (Robot Operating System):

Basic concept:

- Tasks of a robot (motor control, image recognition etc.) are running parallel in nodes



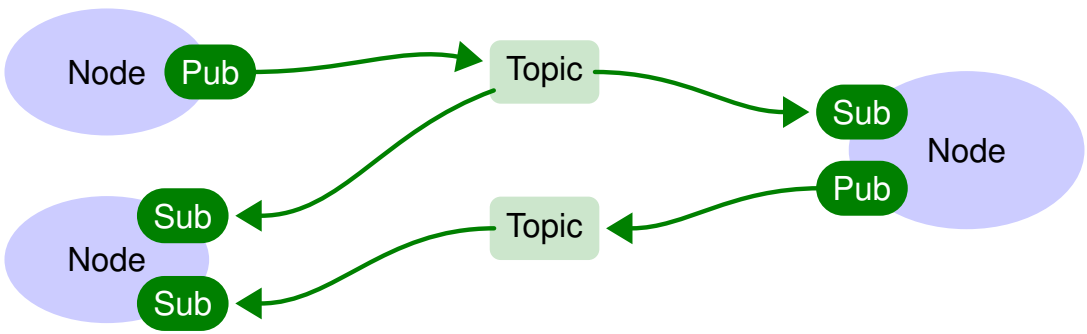
TeSSLa-ROS-Integration

Idea: Use TeSSLa to monitor robot systems

ROS (Robot Operating System):

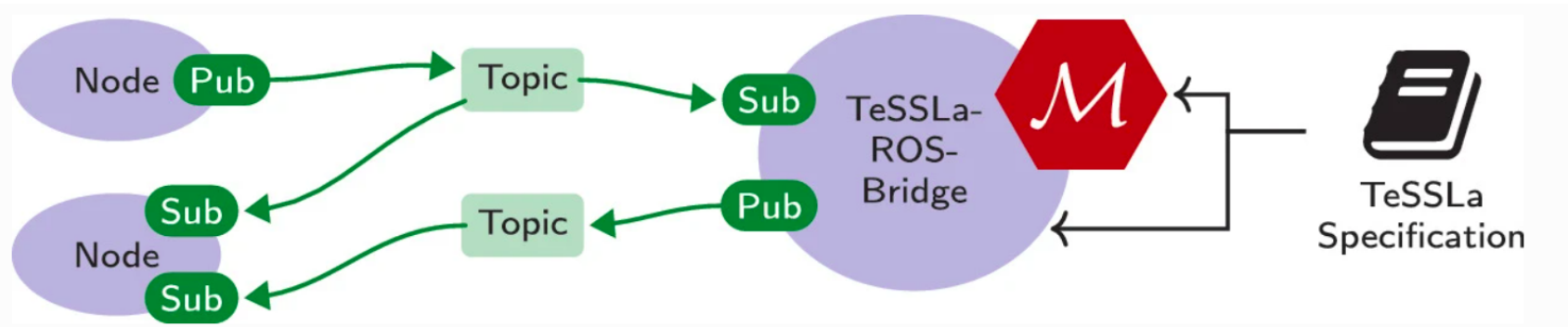
Basic concept:

- Tasks of a robot (motor control, image recognition etc.) are running parallel in nodes
- Communication between nodes via publisher/subscriber pattern



TeSSLa-ROS-Integration: Architecture

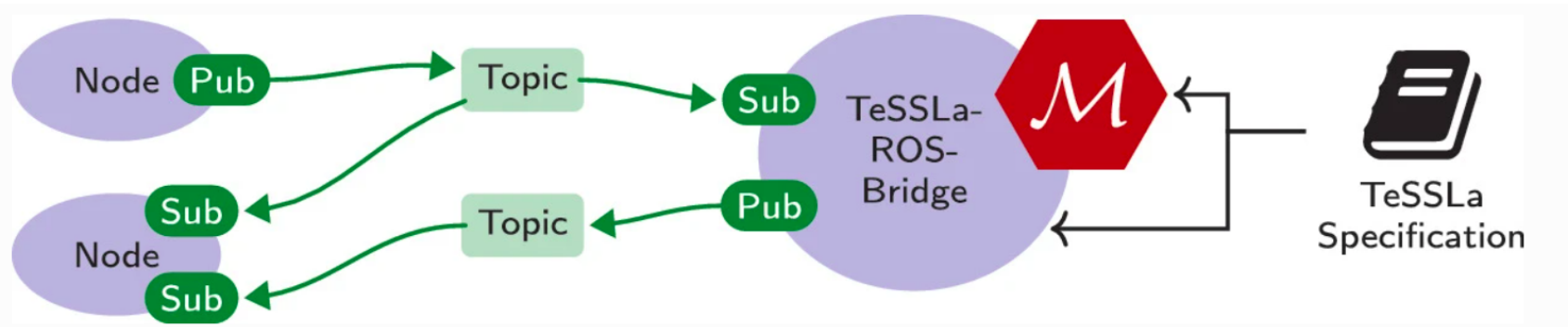
Idea:



TeSSLa-ROS-Integration: Architecture

Idea:

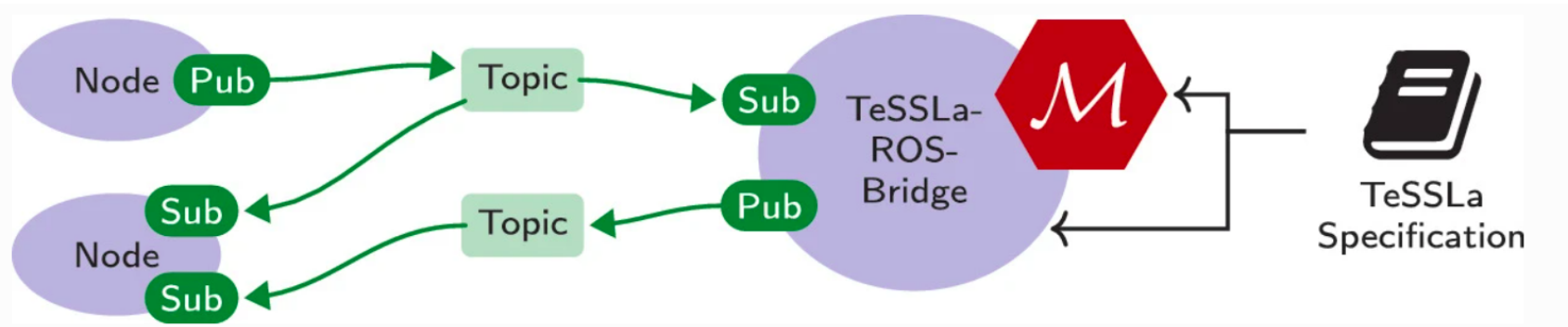
- Use TeSSLa-to-Rust compilation to generate **TeSSLa monitor** from specification



TeSSLa-ROS-Integration: Architecture

Idea:

- Use TeSSLa-to-Rust compilation to generate **TeSSLa monitor** from specification
- Run monitor in **separate node** for **shielding** of safety-critical part of the system

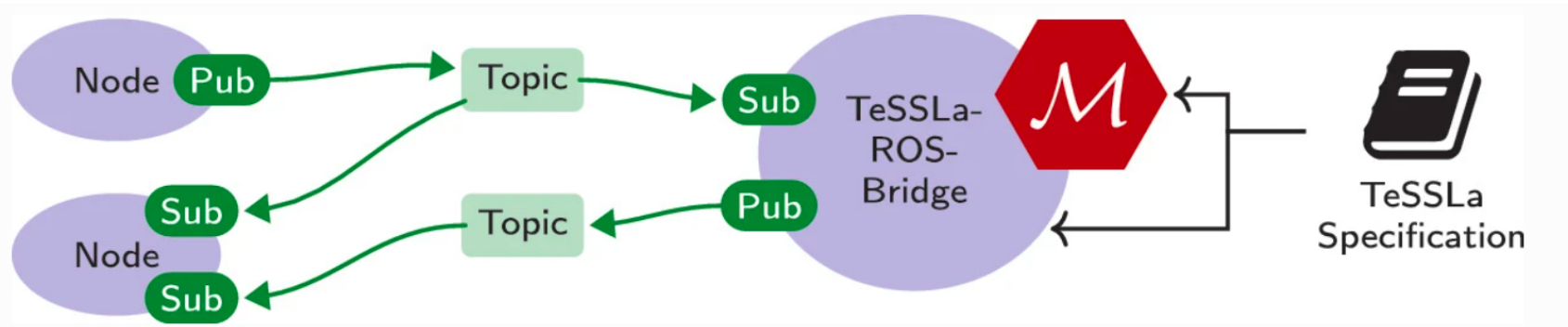


TeSSLa-ROS-Integration: Architecture

Idea:

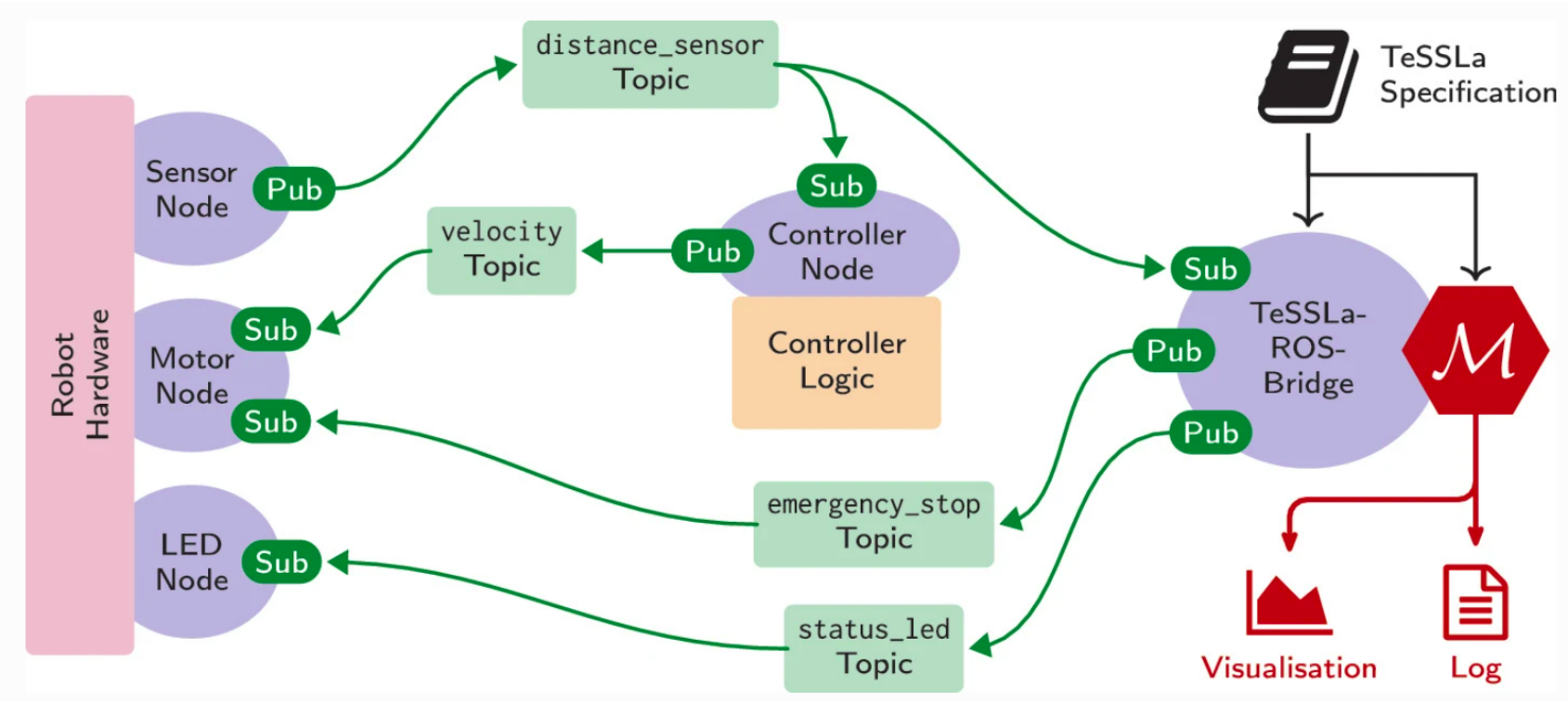
- Use TeSSLa-to-Rust compilation to generate **TeSSLa monitor** from specification
- Run monitor in **separate node** for **shielding** of safety-critical part of the system
- Connect monitor automatically to other nodes via **annotations**

```
@RosSubscription(topic, datatype, qos_profile)
@RosPublisher(topic, datatype, qos_profile)
```



TeSSLa-ROS-Integration: Usage Example

- Robot driving around with distance sensor
- Must stop temporarily whenever something is too close
- Must stop permanently if something was close several times in short period of time



TeSSLa-ROS-Integration: Usage Example

Example specification

```
include "TesslaROSBridge.tessla"

def RED = 0; def YELLOW = 1; def GREEN = 2

module MyModule {
    def cntTimeReset[A](cnt: Events[A], resetTime: Int) =
        resetCount(cnt, delay(const(resetTime, cnt), cnt))
}

@RosSubscription("/distance_sensor", "int64", "10")
in distance: Events[Int]

def tooClose = default(distance < 20, false)
def tooManyErrors = cntTimeReset(rising(tooClose), 30s) > 5
def stop = tooClose || LTL.once(tooManyErrors)
def ledCode = if tooClose then RED else if stop then YELLOW else GREEN

@RosPublisher("/emergency_stop", "bool", "10") @VisBool
out stop

@RosPublisher("/status_led", "int64", "10") @VisSignal
out ledCode

@VisSignal
out tooClose
```

TeSSLa-ROS-Integration: Usage Example

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS
- **ROS:** Modular, highly common operating system for robots

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS
- **ROS:** Modular, highly common operating system for robots
- Combined both approaches for user-friendly shielding of robotic systems

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS
- **ROS:** Modular, highly common operating system for robots
- Combined both approaches for user-friendly shielding of robotic systems
- Small case study to evaluate convenience of approach

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS
- **ROS:** Modular, highly common operating system for robots
- Combined both approaches for user-friendly shielding of robotic systems
- Small case study to evaluate convenience of approach

Future Work

- Try extended RV approaches with robotic domain (e.g. uncertainty)

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS
- **ROS:** Modular, highly common operating system for robots
- Combined both approaches for user-friendly shielding of robotic systems
- Small case study to evaluate convenience of approach

Future Work

- Try extended RV approaches with robotic domain (e.g. uncertainty)
- TeSSLa macros specially suited for robotic context

Recap & Future Work

- **TeSSLa:** Stream Runtime Verification language with several features
  $\Rightarrow$ well suited for specification of CPS
- **ROS:** Modular, highly common operating system for robots
- Combined both approaches for user-friendly shielding of robotic systems
- Small case study to evaluate convenience of approach

Future Work

- Try extended RV approaches with robotic domain (e.g. uncertainty)
- TeSSLa macros specially suited for robotic context
- Use TeSSLa for control tasks of the monitor

Find out more

TeSSLa Website:
www.tessla.io
www.tessla.io/blog/ros-bridge

TeSSLa Playground:
play.tessla.io

TeSSLa Sourcecode:
git.tessla.io