www.obpcdl.org

A Semantic Bridge Between Executable Specifications and Formal Verification Tools

Ciprian TEODOROV

https://teodorov.github.io/

ciprian.teodorov@ensta.fr

P4S, Lab-STICC, UMR CNRS 6285

ensta-bretagne.fr/teodorov

Academia @ Lab-STICC, ENSTA Bretagne, Brest

[23-…] *Full Professor*

[15-23] *Associate professor*

Lead the OBP2 *Semantic Diagnosis & Formal Verification* Lab. (http://www.obpcdl.org/)

[13-15] *Postdoc*

Verification MBSE, Concurrent system modeling, and verification (https://gemoc.org/)

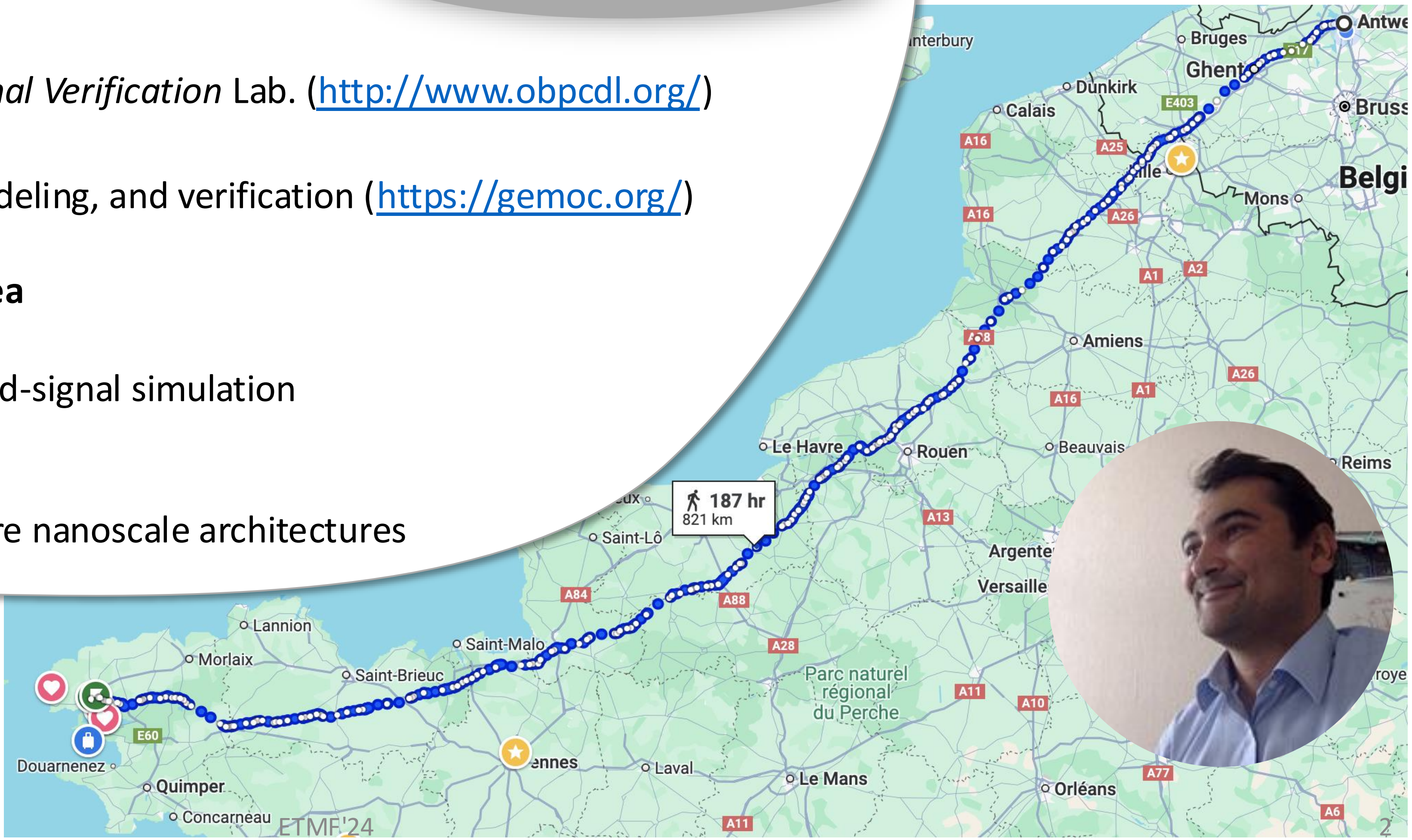
Industry @ Dolphin Integration - Grenoble Area

[11-13] *Electronics CAD engineer*

Compilation of VHDL, VHDL-AMS for mixed-signal simulation

PhD in Computer Science @ UBO – Brest

[08-11] Model-driven physical design for future nanoscale architectures



Lab-STICC

The Road Today

1. Executable specifications & behavior analysis monitors

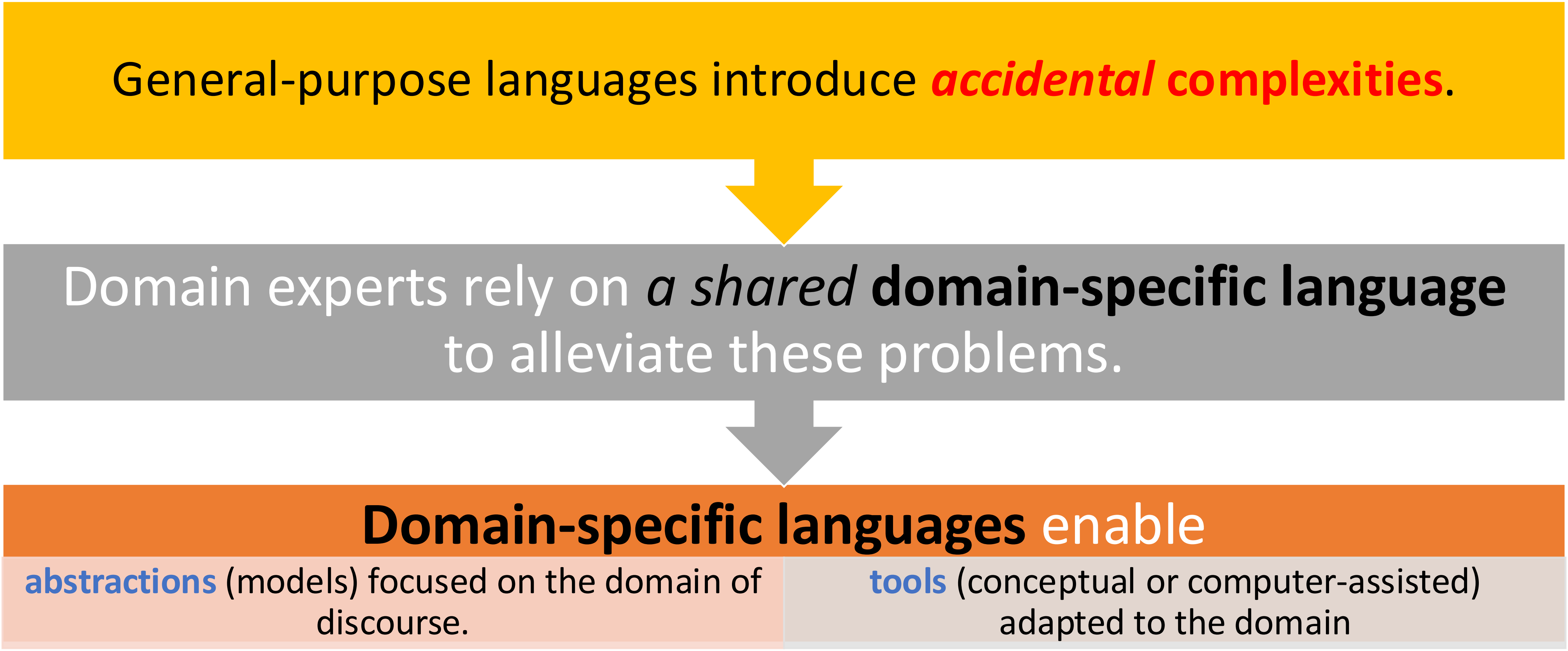
2. The shy semantics and the inaccessible monitors.

3. G∀min∃: If the semantics opens up the monitors are interested.

4. When G∀min∃ experiences the real world.

5. Sum up and ways forward.

Context: Domain-specific languages



General-purpose languages introduce ***accidental* complexities**.

Domain experts rely on *a shared* **domain-specific language** to alleviate these problems.

Domain-specific languages enable

abstractions (models) focused on the domain of discourse.

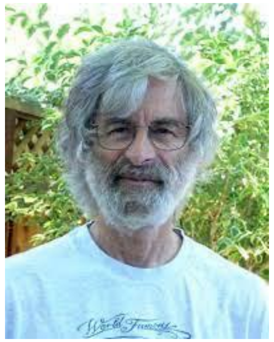
tools (conceptual or computer-assisted) adapted to the domain

Context: Executable specifications

- eXecutable Domain-Specific Languages (xDSL) for handling behaviors.
  - Programming languages = prescriptive xDSLs

    force the computer to perform some behavior.
  - Thinking above the code [1], specifying, requires a problem-oriented mindset

- Executable-Specifications capture the behavior to study it in captivity
  - Descriptive xDSL that reflects how the object behaves



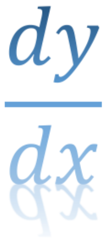
Descriptive [2]:

- presenting observations about the characteristics of something
- factually grounded or informative rather than normative, prescriptive or emotive

[1] Leslie Lamport: Thinking Above the Code

[2] (https://www.merriam-webster.com/dictionary/descriptive)

a Zoo of Executable Specification Languages



Physical processes

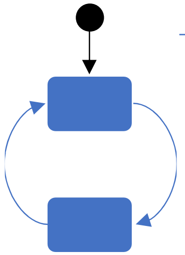
- Calculus [Newton and Leibniz]

Temporal logic

- LTL
- CTL*
- Temporal Logic of Actions (TLA+)

Computable functions

- Lambda calculus
- Turing machines

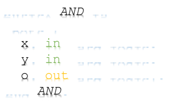


Automata

- NFA
- PDA
- Statecharts

Concurrency

- Petri nets
- CSP – Hoare
- Actor models – Hewitt



HDLs

- VHDL[-AMS]
- [System-]Verilog[-A]

Terminology

Language monitoring [KHC91] is the process of observing the **execution of a computer program** expressed in a given **programming language**.

[KHC91] Amir Kishon, Paul Hudak, and Charles Consel. 1991. Monitoring semantics: a formal framework for specifying, implementing, and reasoning about execution monitors. In Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation (PLDI '91). Association for Computing Machinery, New York, NY, USA, 338–352. https://doi.org/10.1145/113445.113474

Terminology: In our context

Language monitoring is the process of observing the **behavior of an executable specification** expressed in a given **specification language**.

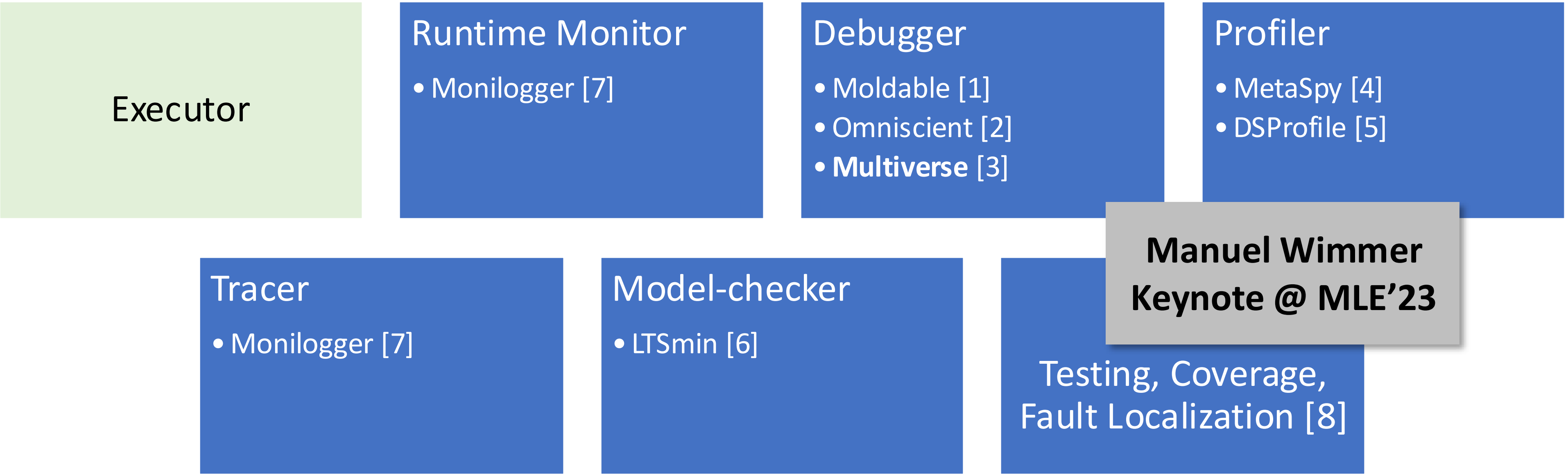
In the following:

the tools that enable this process will be referred to as:

***language* monitors**, or simply **monitors**

***runtime* monitors** are a subclass of ***language* monitors**

a Zoo of Language Monitors



Executor

Runtime Monitor
- Monilogger [7]

Debugger
- Moldable [1]
- Omniscient [2]
- **Multiverse** [3]

Profiler
- MetaSpy [4]
- DSProfile [5]

Manuel Wimmer Keynote @ MLE'23

Tracer
- Monilogger [7]

Model-checker
- LTSmin [6]

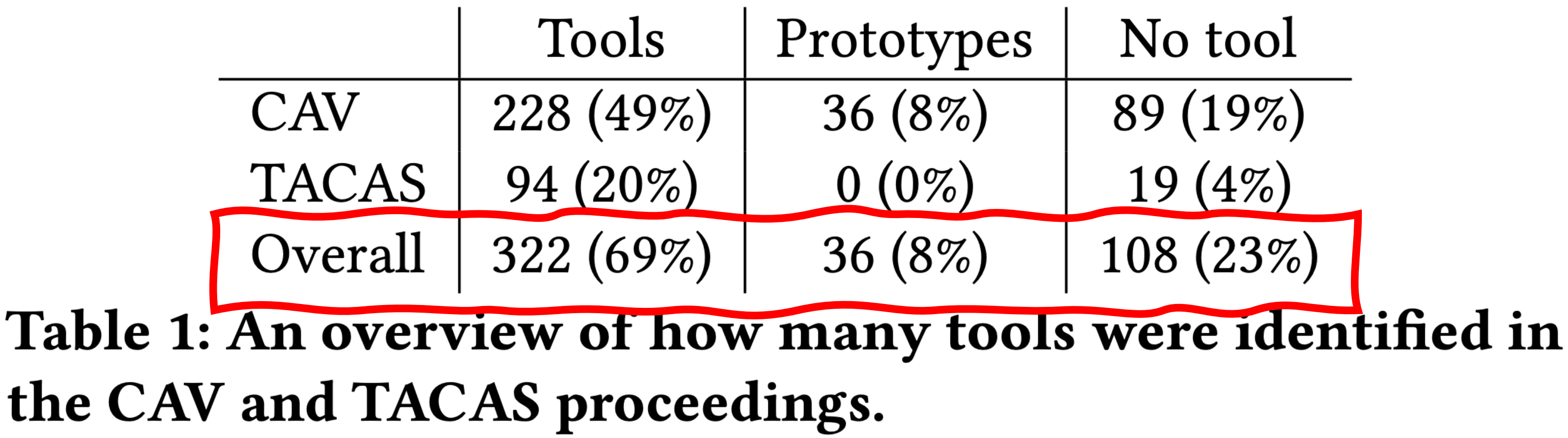
Testing, Coverage, Fault Localization [8]

[1] Chiş et al. ”*The Moldable Debugger: A Framework for Developing Domain-Specific Debuggers.*” SLE 2014.
[2] Bousse et al. “*Omniscient Debugging for Executable DSLs.*” JSS 2018.
[3] Torres Lopez et al. "*Multiverse debugging: Non-deterministic debugging for non-deterministic programs.*" ECOOP 2019.
[4] Bergel et al. "*Domain-specific profiling.*" TOOLS 2011.
[5] Sloane et al. "*Domain-specific program profiling and its application to attribute grammars and term rewriting.*" *SCP* 2014.
[6] Kant et al. ”*LTSmin: High-Performance Language-Independent Model Checking.*” TACAS 2015.
[7] Leroy et al. *“Monilogging for executable domain-specific languages.” SLE 2021*
[8] Khorram et al. *“From Coverage Computation to Fault Localization: A Generic Framework for Domain-Specific Languages.“ SLE 22*

Program Verification Tools [1]

https://slebok.github.io/proverb/

tool papers from TACAS 2016–2021
all papers from CAV 2017–2021



	Tools	Prototypes	No tool
CAV	228 (49%)	36 (8%)	89 (19%)
TACAS	94 (20%)	0 (0%)	19 (4%)
Overall	322 (69%)	36 (8%)	108 (23%)

Table 1: An overview of how many tools were identified in the CAV and TACAS proceedings.

[1] *Sophie Lathouwers and Vadim Zaytsev. 2022. Modelling program verification tools for software engineers. In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22). Association for Computing Machinery, New York, NY, USA, 98–108. https://doi.org/10.1145/3550355.3552426*

Questions to ponder

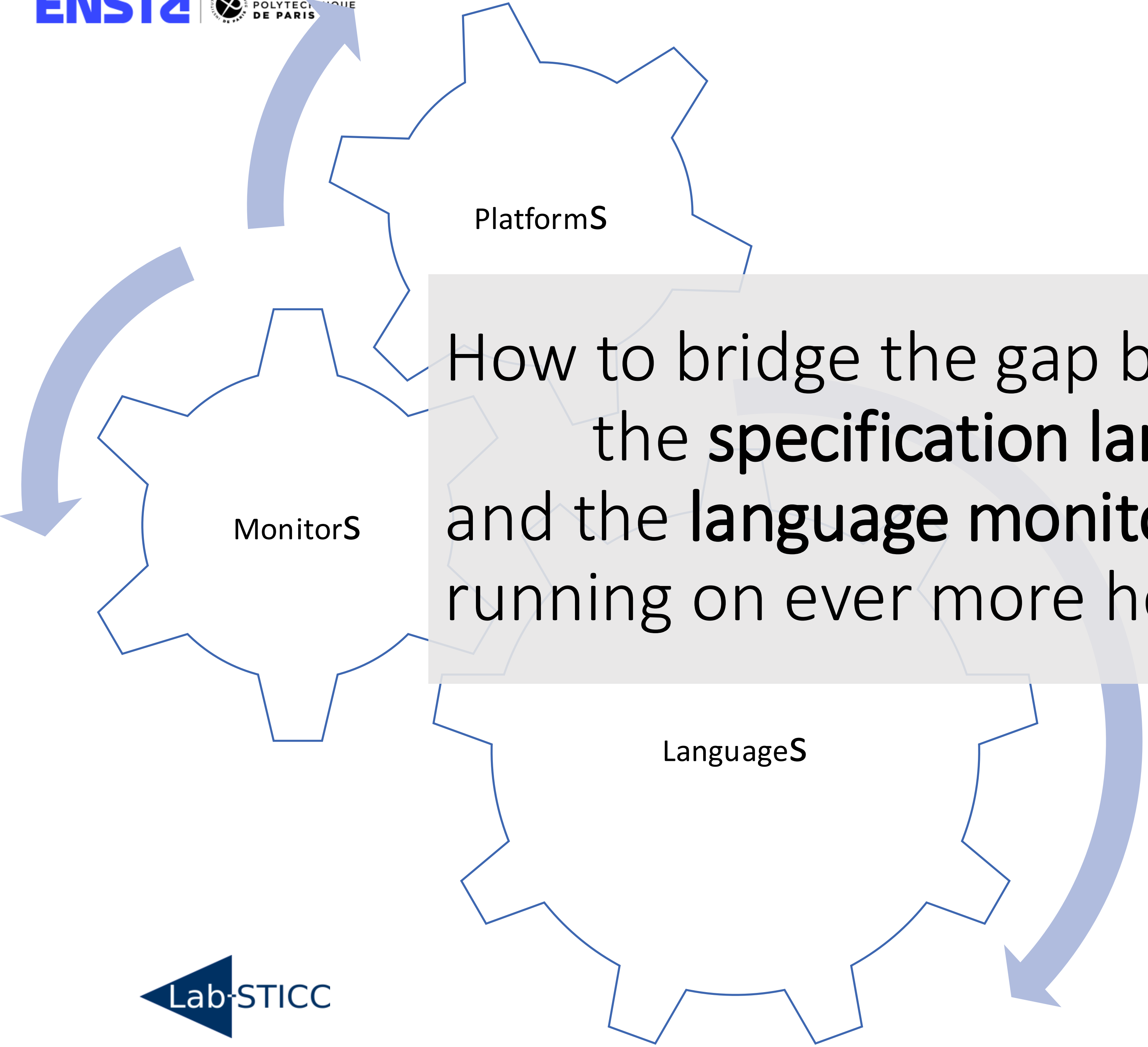
Sustainability?

How to write code that survives?

Does your favorite **specification** language:

- has a debugger? Can it go back in time?
- comes with a model-checker?
- offers support for random testing?

Why do we still lack these basic tools for so many **practically important** specification languages?



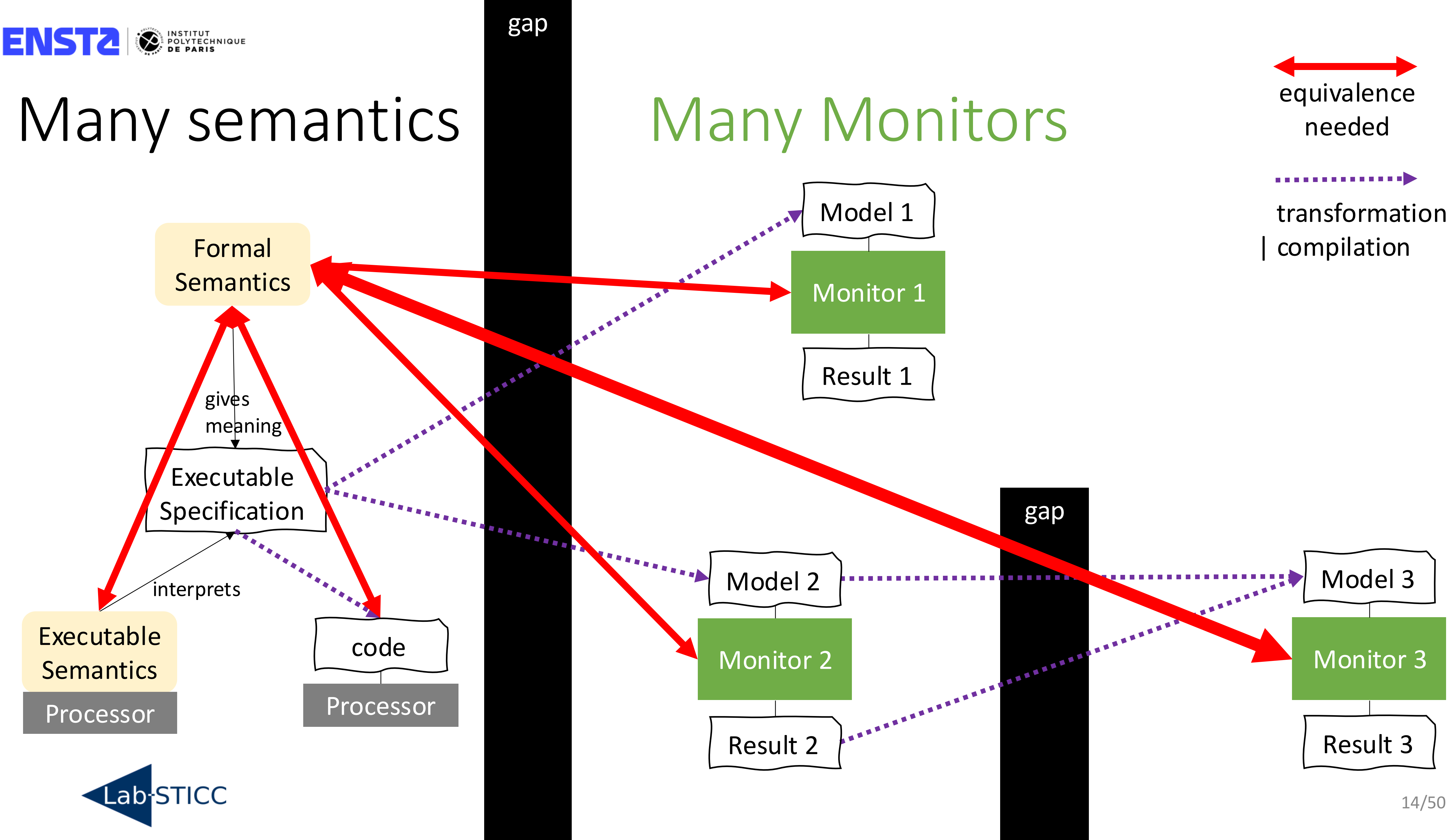
How to bridge the gap between
the **specification languages**
and the **language monitors**
running on ever more heterogeneous **platforms**?

2. The Shy Semantics and the Inaccessible Monitors.

- Understanding the problem
- Looking for high-level solutions

Many semantics

Many Monitors



gap

equivalence needed

transformation | compilation

Formal Semantics

gives meaning

Executable Specification

interprets

Executable Semantics

Processor

code

Processor

Model 1

Monitor 1

Result 1

Model 2

Monitor 2

Result 2

gap

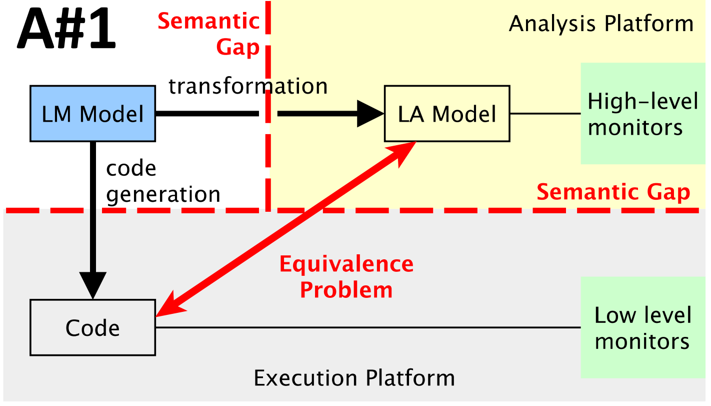
Model 3

Monitor 3

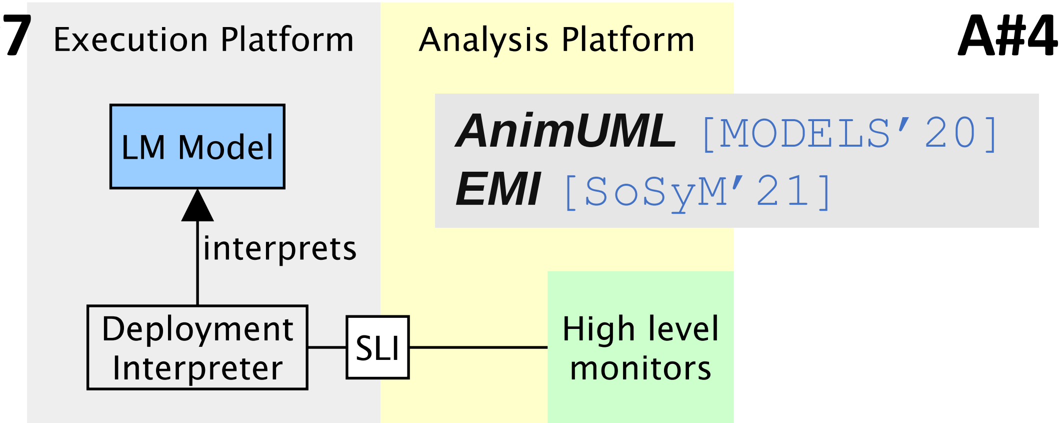
Result 3

ENSTA embeddr, IF [Dragomir+22]

A#1



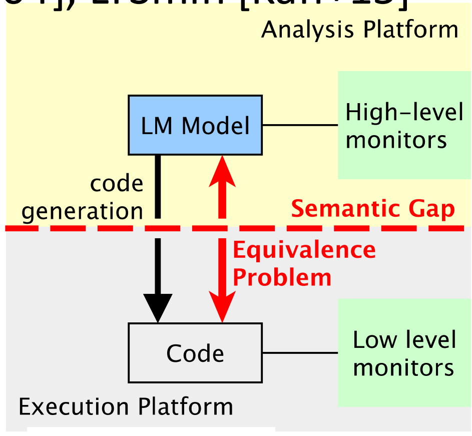
A#7



AnimUML [MODELS'20]
EMI [SoSyM'21]

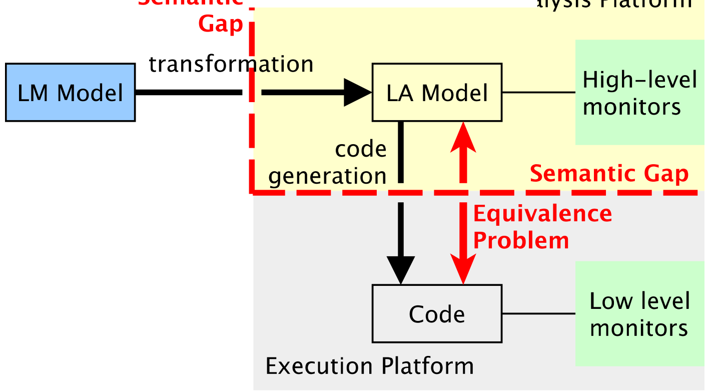
SPOT [DP04], LTSmin [Kan+15]

A#4



A#2

Spin [Hol97]

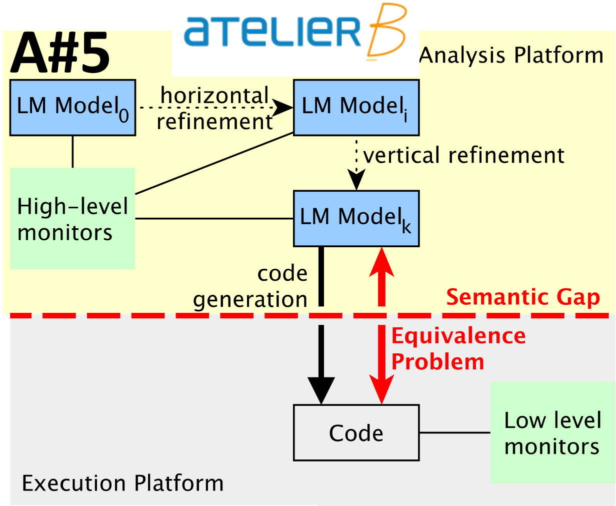


P#1 Semantic gap between design model and analysis model

P#2 Semantic gap between design model and executable code

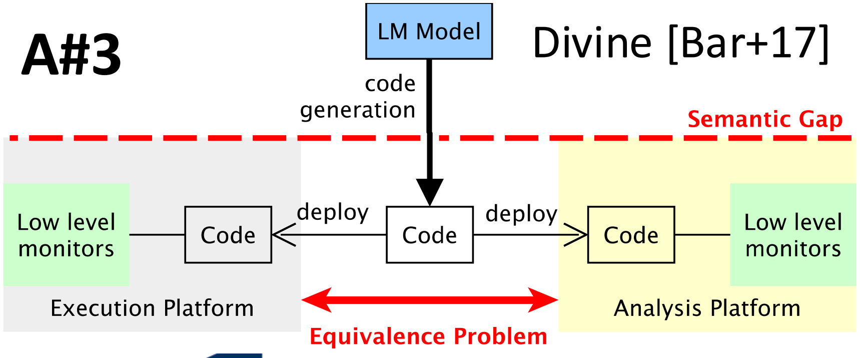
P#3 Equivalence problems between the analysis model and executable code

A#5



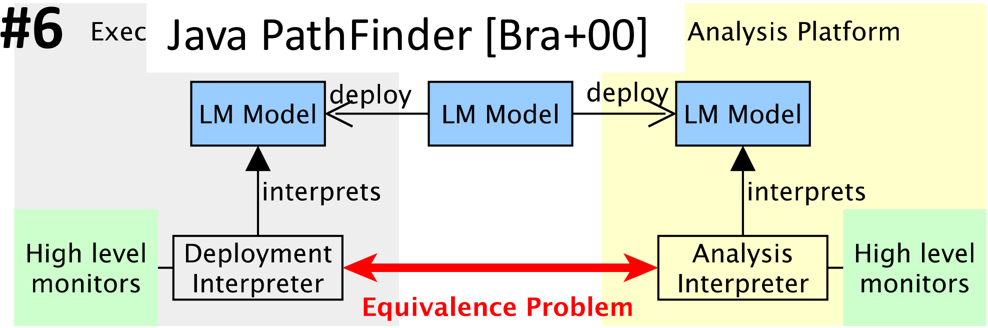
A#3

Divine [Bar+17]



A#6

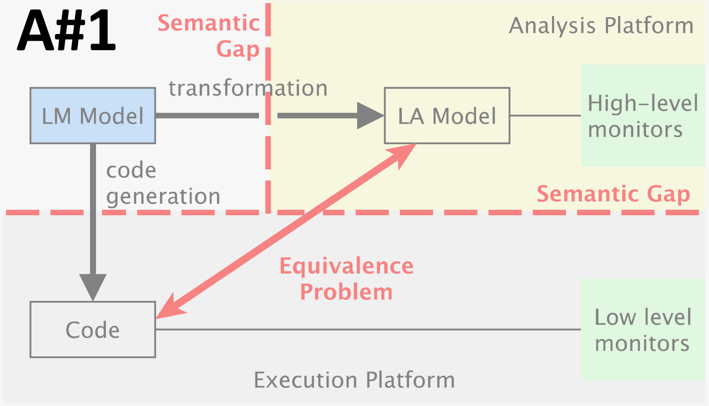
Java PathFinder [Bra+00]



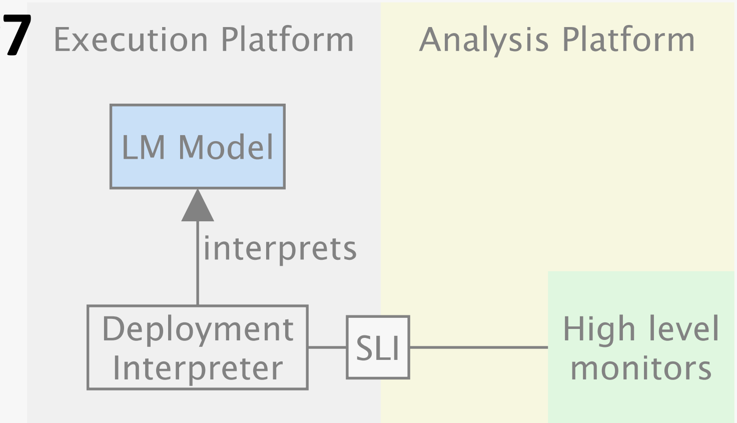
Lab-STICC

V. BESNARD, "**EMI: Une approche pour unifier l'analyse et l'exécution embarquée à l'aide d'un interpréteur de modèles pilotable**", Application aux modèles UML des systèmes embarqués, *Ph.D. Thesis*, Dec. 2020.

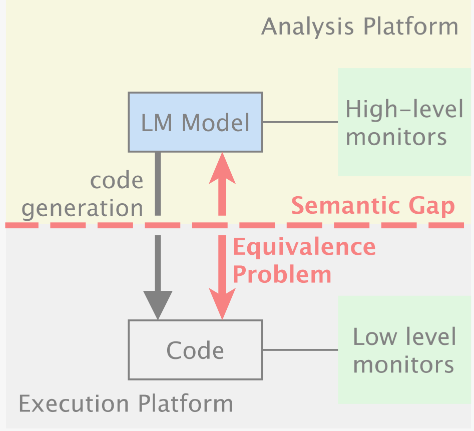
A#1



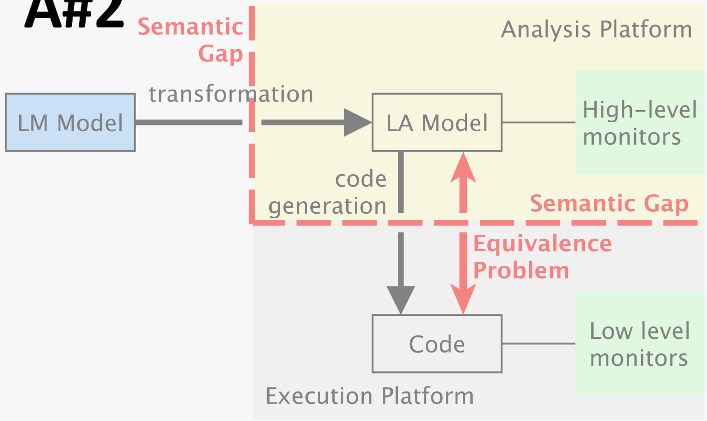
A#7



A#4

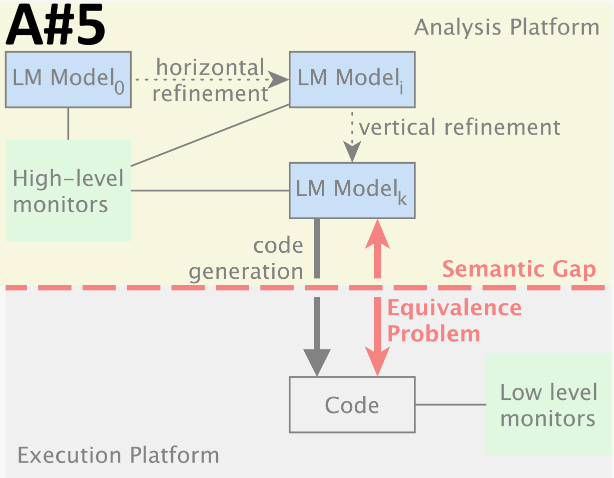


A#2

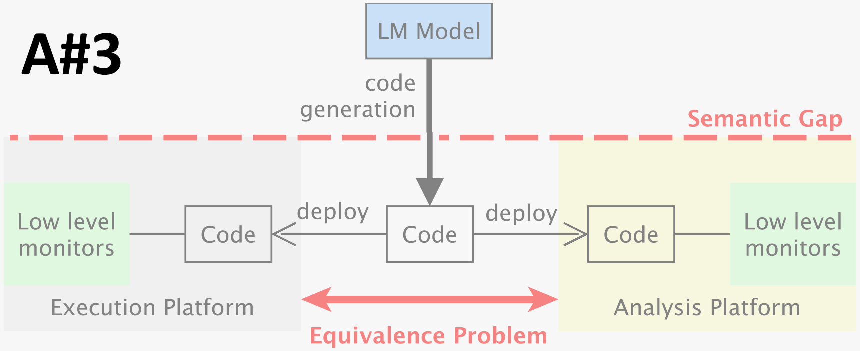


	A#1	A#2	A#3	A#4	A#5	A#6	A#7
P#1	✗	✗	✓	✓	✓	✓	✓
P#2	✗	✗	✗	✗	✗	✓	✓
P#3	✗	✗	✗	✗	✗	✗	✓

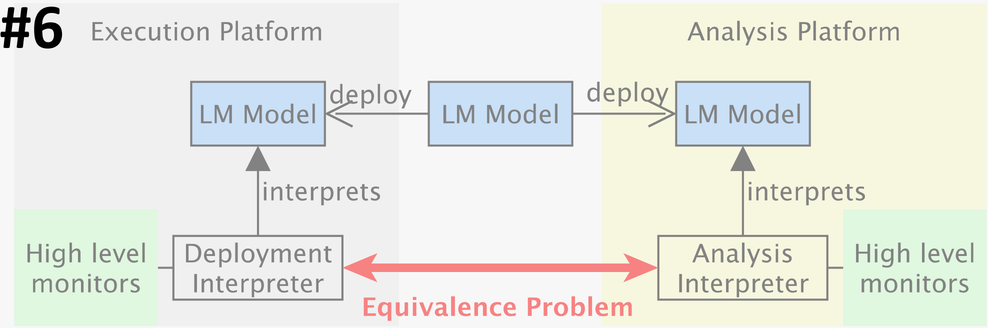
A#5



A#3



A#6

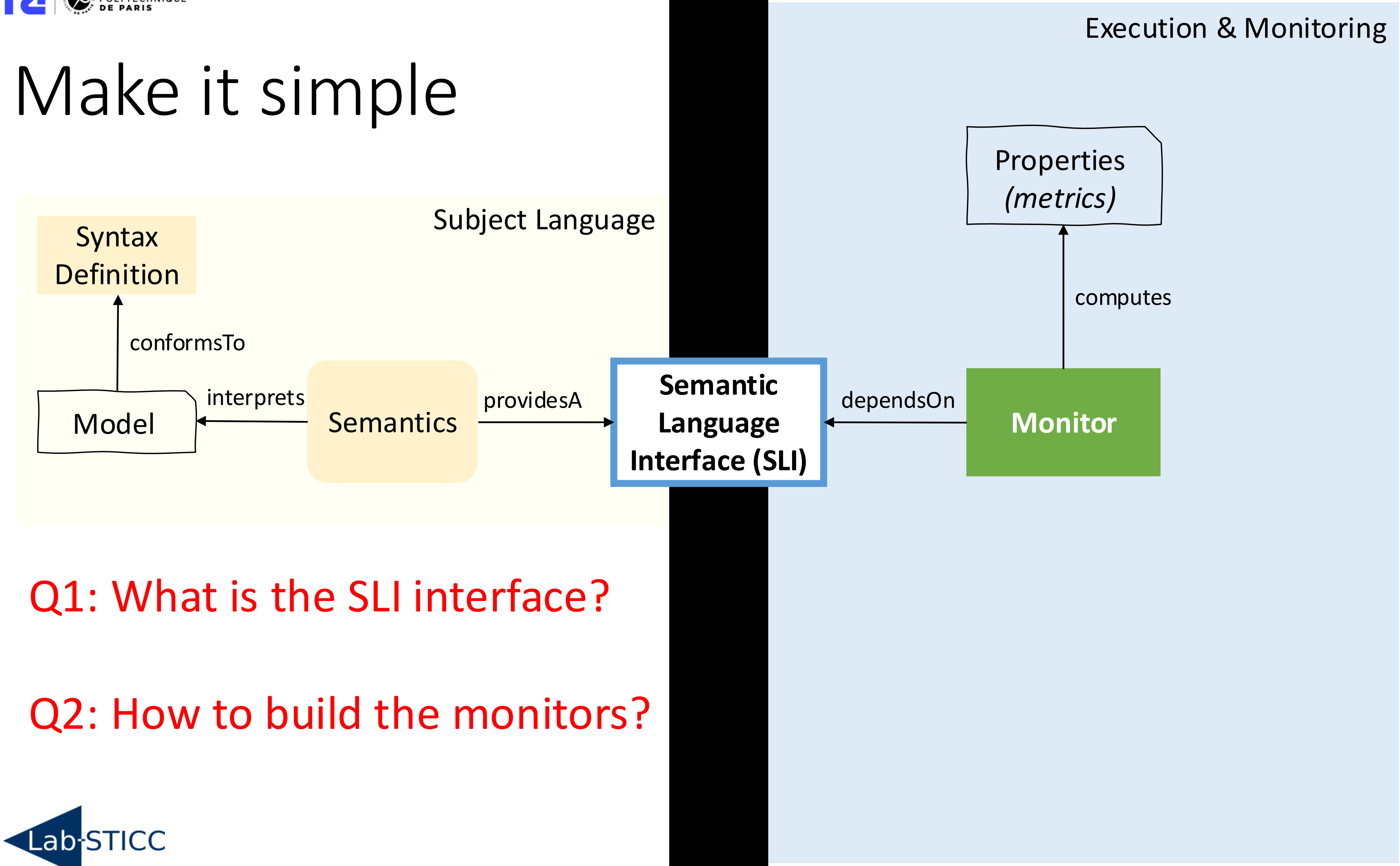


V. BESNARD, “**EMI: Une approche pour unifier l’analyse et l’exécution embarquée à l’aide d’un interpréteur de modèles pilotable**”, Application aux modèles UML des systèmes embarqués, *Ph.D. Thesis*, Dec. 2020.

3. G∀min∃: If the Semantics Opens Up the Monitors are Interested.

- Requirements
- G∀min∃ Semantic Language Interface
- An illustration

Make it simple

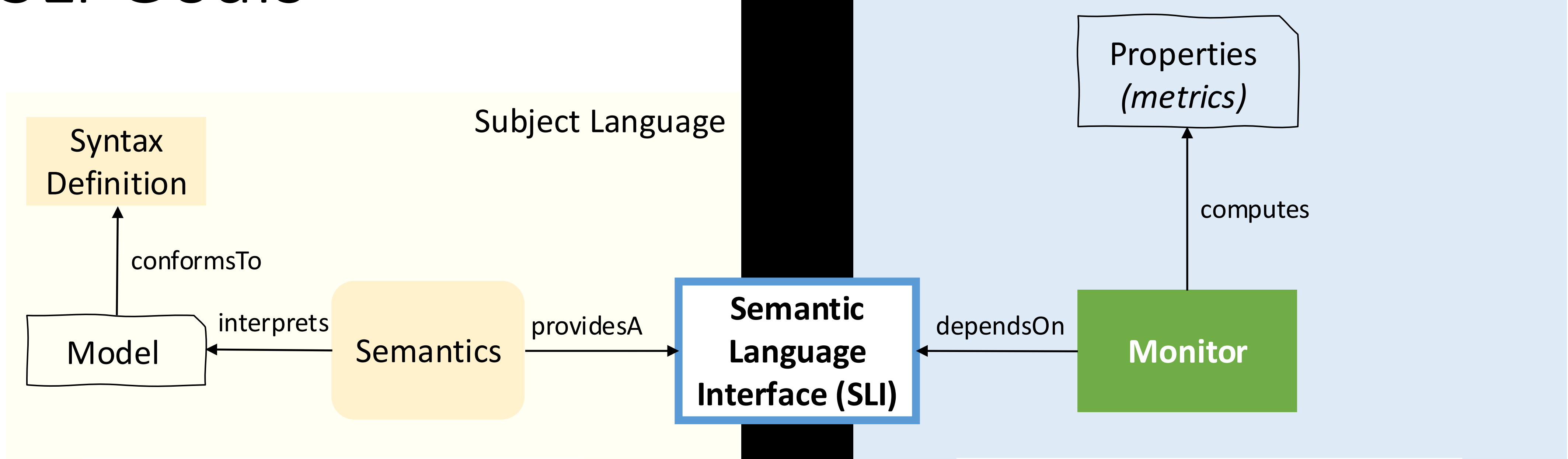


Q1: What is the SLI interface?

Q2: How to build the monitors?

[SLE'16]

SLI Goals



[G01] Completeness
[G02] Non-Interference

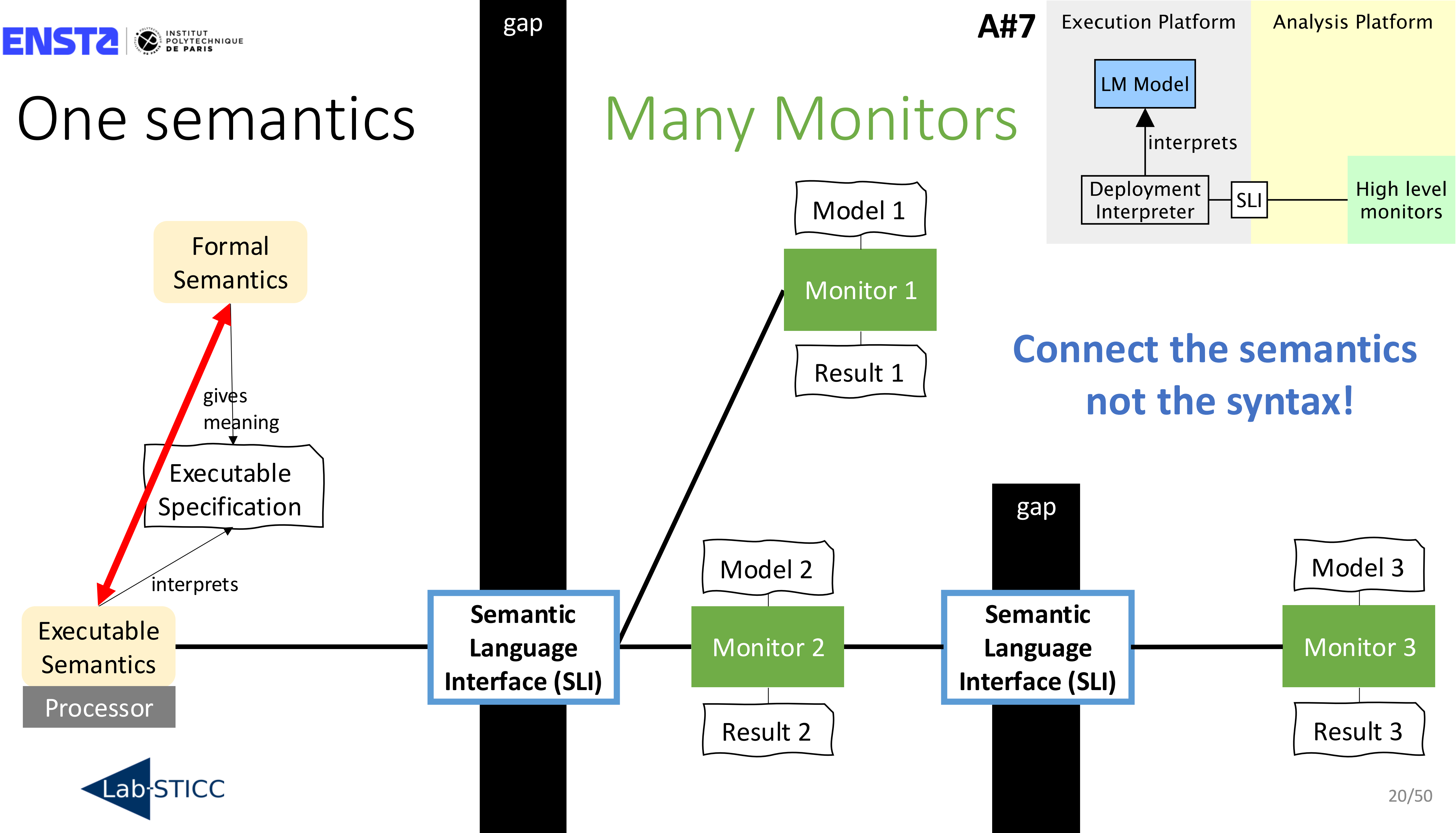
[G03] Break the Rules
[G04] Minimize the Gap

[G05] Composability
[G06] Portability
[G07] Genericity
[G08] Unanticipated Monitoring
[G09] *Ease the* Integration

One semantics

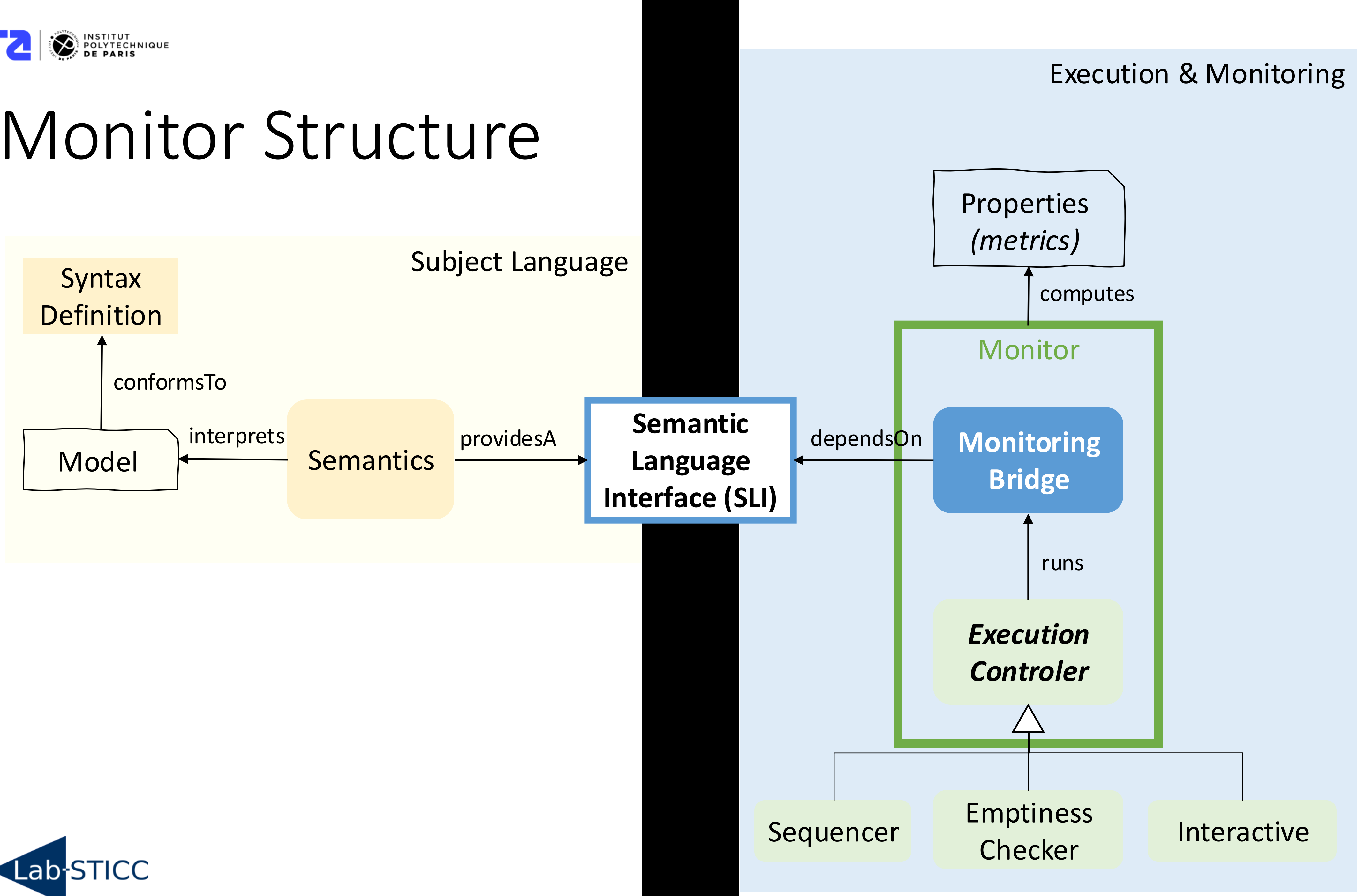
Many Monitors

A#7



Connect the semantics not the syntax!

Monitor Structure



G∀min∃ Semantic Language Interface (SLI)

```
SLI {
    semantics: (C A) {
        initial:    set C
        actions:    C → set A
        execute: A → C → set C
    }

              Expression   execution step   Value

    evaluate: E → (C x A x C) → V -- questions

    reduce: R → C → α                   -- reductions

    π: (C A V α T) {…}                  -- projections
}
```

Generic Types:

C*onfiguration:*

Example CEK-style

C ≜ ⟨control, env, [Frame]⟩

A*ction:*

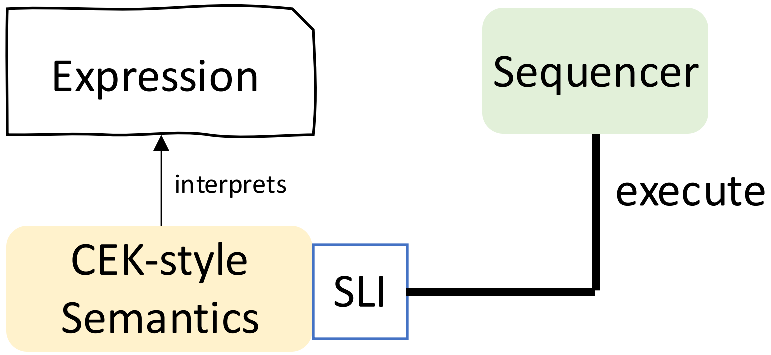
Example CEK-style

A ≜ from-predicate ⟶ to-C

Similar semantic approaches:

Lamport L. "*The temporal logic of actions.*" TOPLAS. **1994** https://doi.org/10.1145/177492.177726

Charguéraud, et al. "*Omnisemantics: Smooth Handling of Nondeterminism.*" TOPLAS. **2023**, https://doi.org/10.1145/3579834



Expression

interprets

CEK-style Semantics

SLI

Sequencer

execute

Example CEK-style
C ≜ ⟨control, env, [Frame]⟩

A ≜ from-predicate ⟶ to-C

SLI Semantics for a CEK-style abstract machine

```
rules: { lookup, app, arg, body, … }

SLI.semantics: (C A) {
 initial: set C := {⟨exp, Ø, []⟩}

 actions: C  → set A
 | c => rules.where(r => r.enabledIn c)

 execute: A → C → set C
 | r c => { r.applyIn c }
}
```

```
Sequencer(sli) {
        current = sli.initial.any
        while (current != NULL) {
                action = sli.actions(current).any
                if (actions == NULL) break;
                current= sli.execute(action,current).any
        }
}
```

- If **sli** exposes a <u>deterministic</u> semantics → *exactly one sequence*

<=>

∀ a c, |*initial*| = |*actions* c| = |*execute* a c| = 1

4. When G∀min∃ experiences the real world.

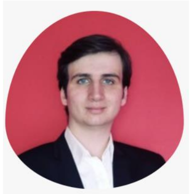
- **Some experiences unravel reusable monitoring bridges**
- Exploring hardware execution
- Multiverse debugging made simple and more powerful
- Transfer to commercial products -- *OBP2 inside*
- Transfer to future practitioners -- *From zero to model-checker*

OBP2 Research Vehicle

2015-2025

Emilien FOURNIER
2022

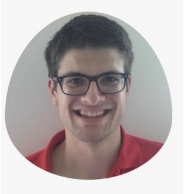
Nicolas SUN
2022



Matthias PASQUIER
2024

Luka LE ROUX
2018

Vincent LEILDE
2019



Valentin BESNARD
2020

ENSTA BRETAGNE

Safety & Liveness
Temporal Requirements

Semantic
Language
Interface

J.C. ROGER

B. DROUOT

T. BOLLENGIER

L.LE ROUX

F. GOLRA

Projects:

ONEWAY	(DGAC)
Ker-SEVECO	(R.Bretagne,EU)
JoinSafeCyber	(AID)
VeriMoB	(RAPID)
EASE4SE	(RAPID)
DEPARTS	(PIA)
GeMoC	(ANR)

DGA

Commercial Products [*PragmaDEV*]

sdl PRAGMADEV STUDIO

BPMN PRAGMADEV PROCESS

Academic Prototypes [*in-house*]

AnimUML

UNIFIED MODELING LANGUAGE UML

EMI-UML

AEFD SNCF

Lucio

eseo GRANDE ÉCOLE D'INGÉNIEURS

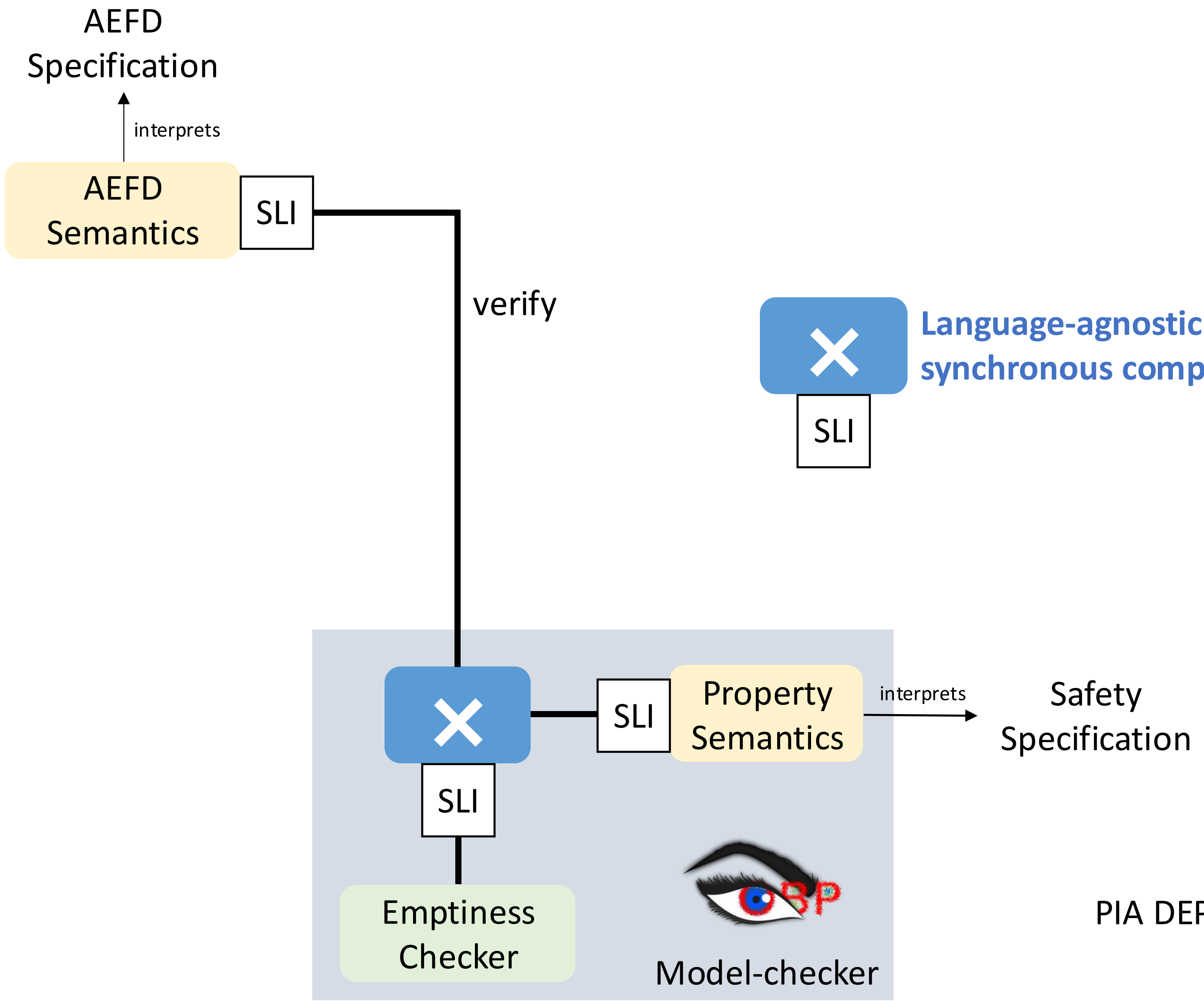
DAVIDSON CONSULTING

Reuse [OTS]

TLA+

Fiacre

PhD Luka
LE ROUX



AEFD Specification

interprets

AEFD Semantics

SLI

verify

SLI

Property Semantics

interprets

Safety Specification

SLI

Emptiness Checker

Model-checker

SLI

Language-agnostic synchronous composition operator

PIA DEPARTS

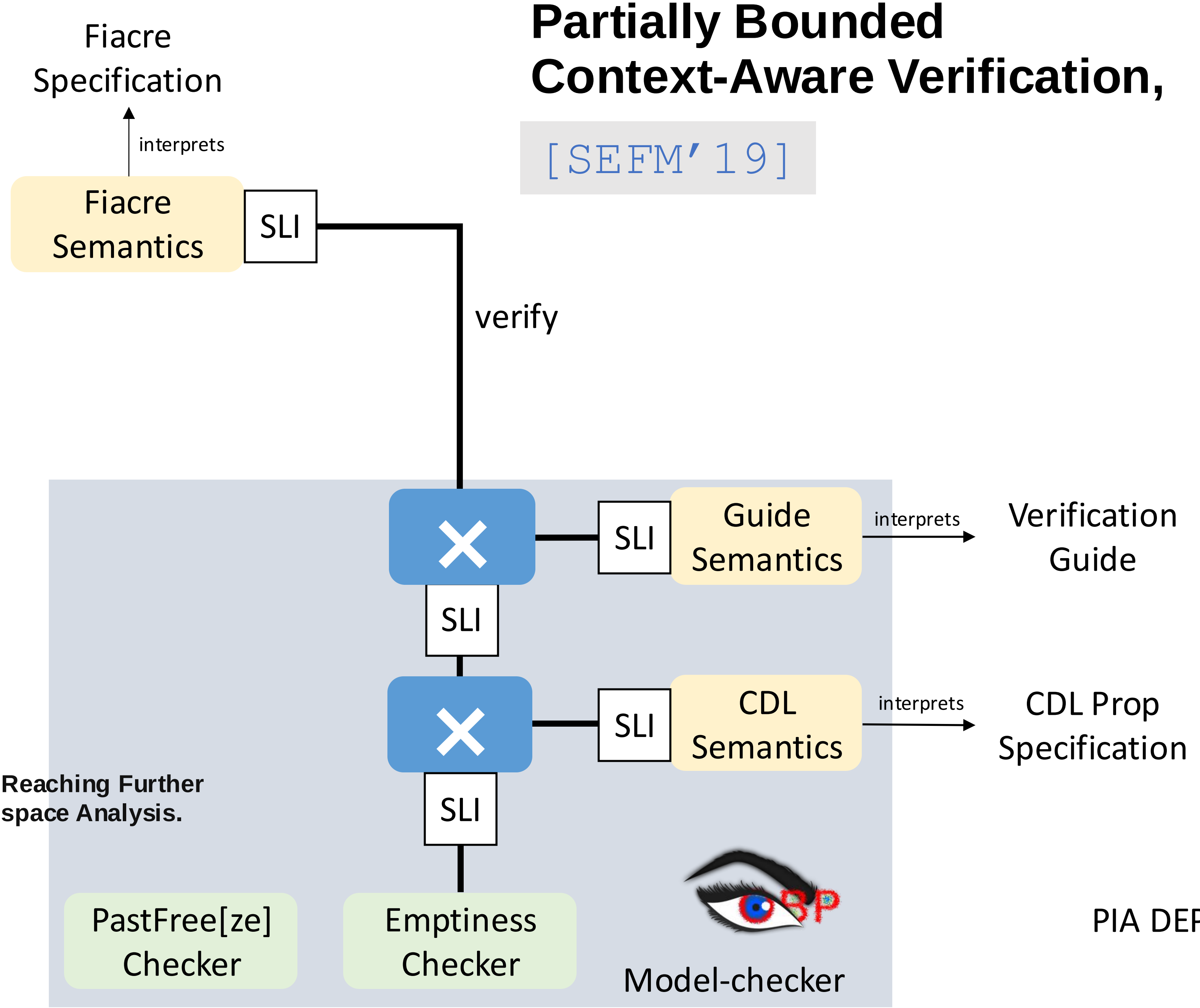
AEFD SNCF

Lab-STICC

Partially Bounded Context-Aware Verification,

[SEFM'19]

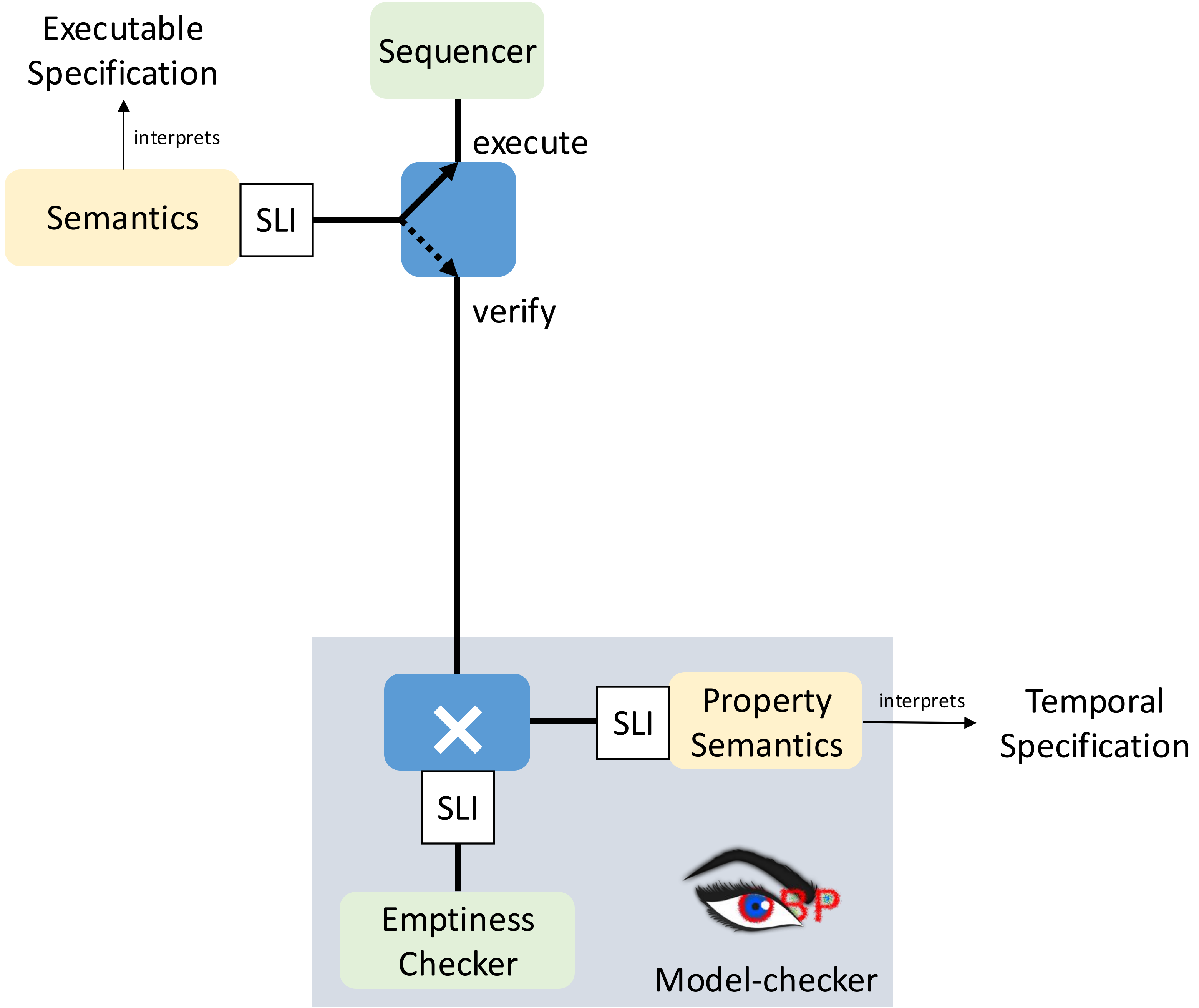
PhD Luka LE ROUX



Past-Free[ze] Reachability Analysis: Reaching Further with DAG-directed Exhaustive State-space Analysis.

[STVR'16]

PIA DEPARTS



Executable Specification

interprets

Semantics

SLI

Sequencer

execute

verify

SLI

Property Semantics

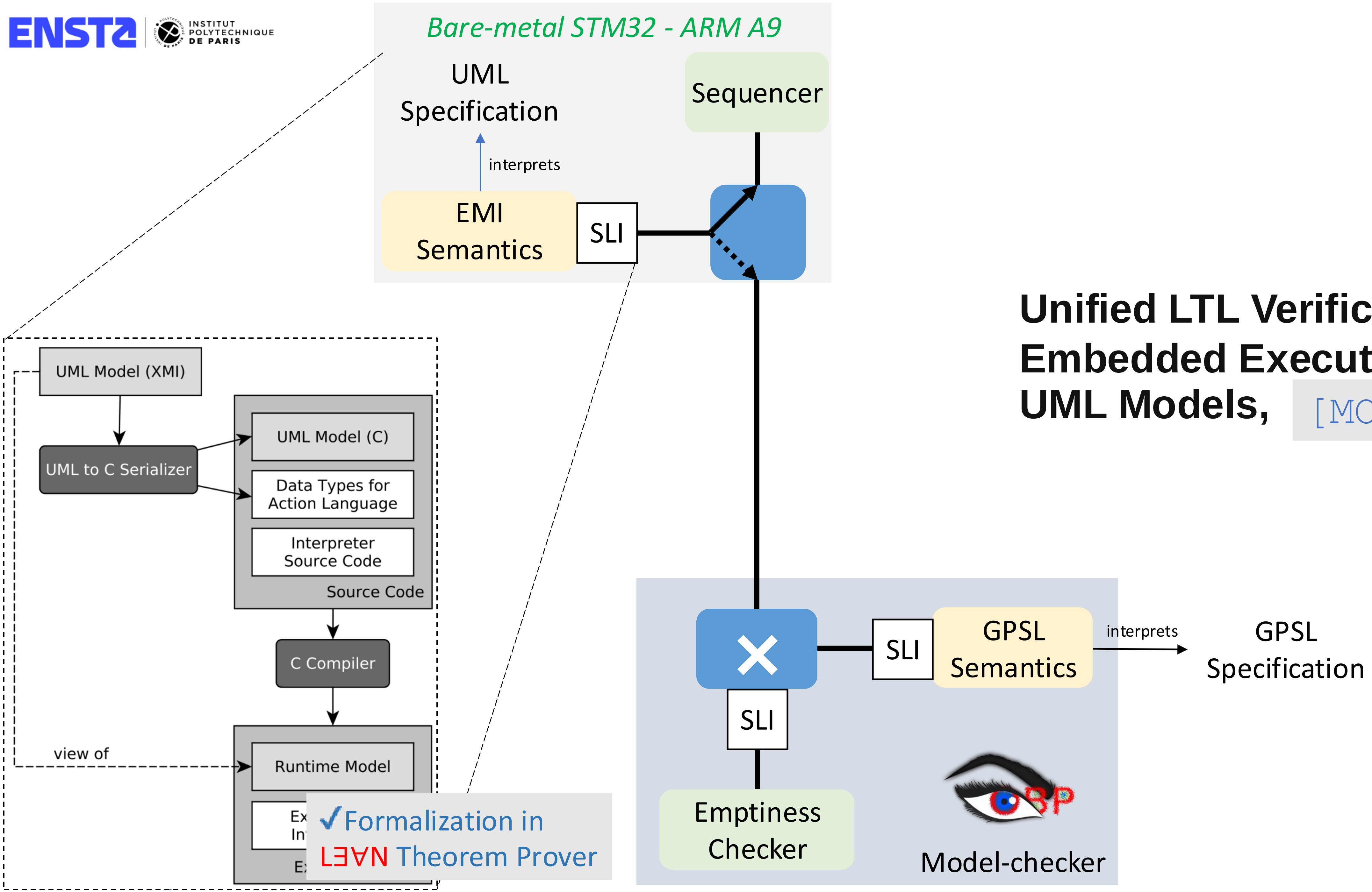
interprets

Temporal Specification

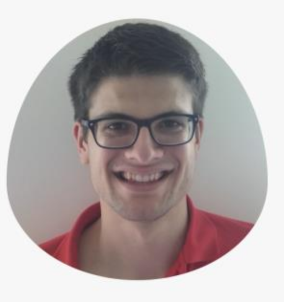
SLI

Emptiness Checker

Model-checker

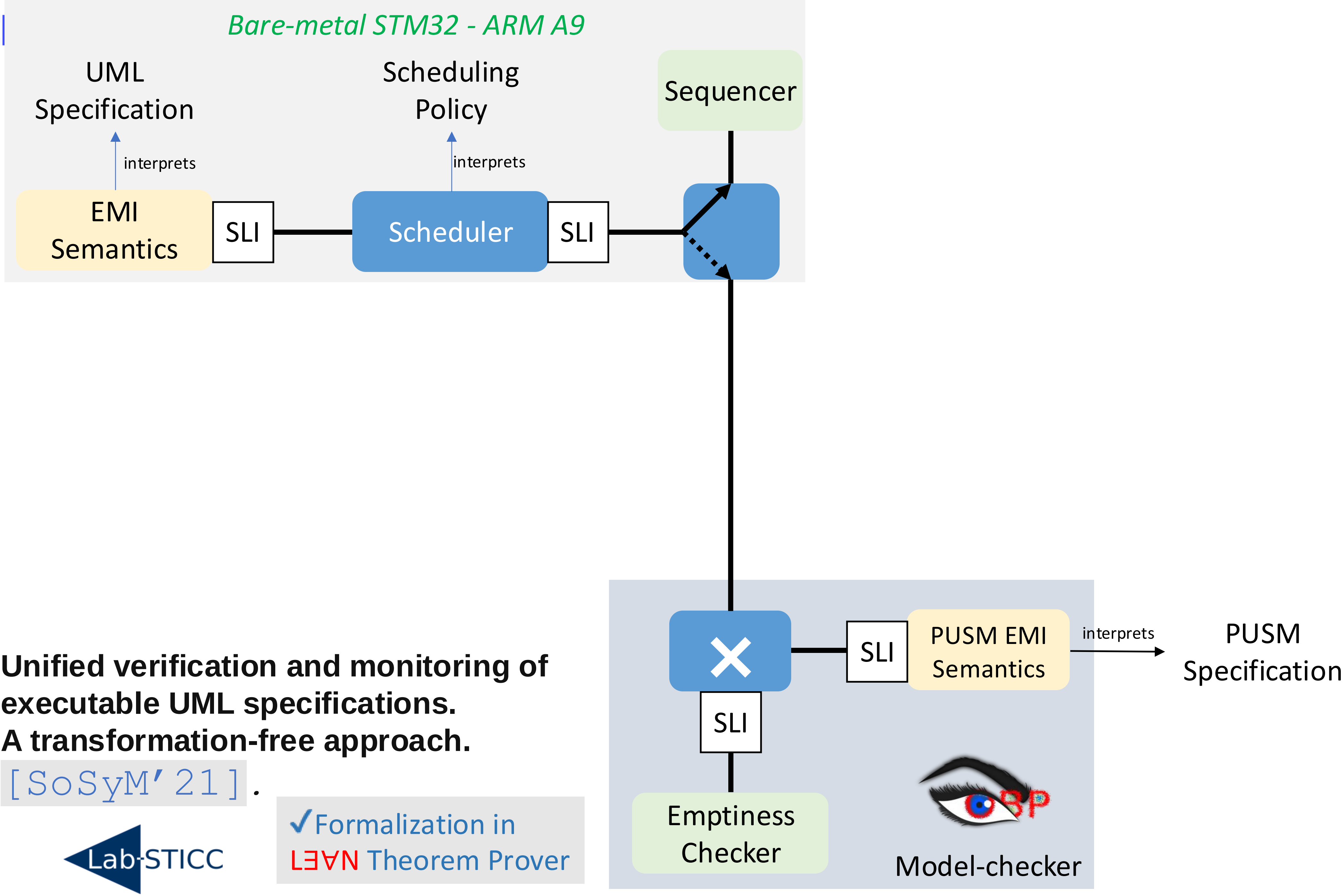


✓Formalization in
L∃∀N Theorem Prover



PhD Valentin
BESNARD

Unified LTL Verification and Embedded Execution of UML Models, [MODELS’18]



Bare-metal STM32 - ARM A9

UML Specification

Scheduling Policy

Sequencer

interprets

interprets

EMI Semantics

SLI

Scheduler

SLI

SLI

PUSM EMI Semantics

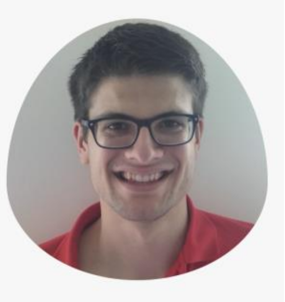
interprets

PUSM Specification

SLI

Emptiness Checker

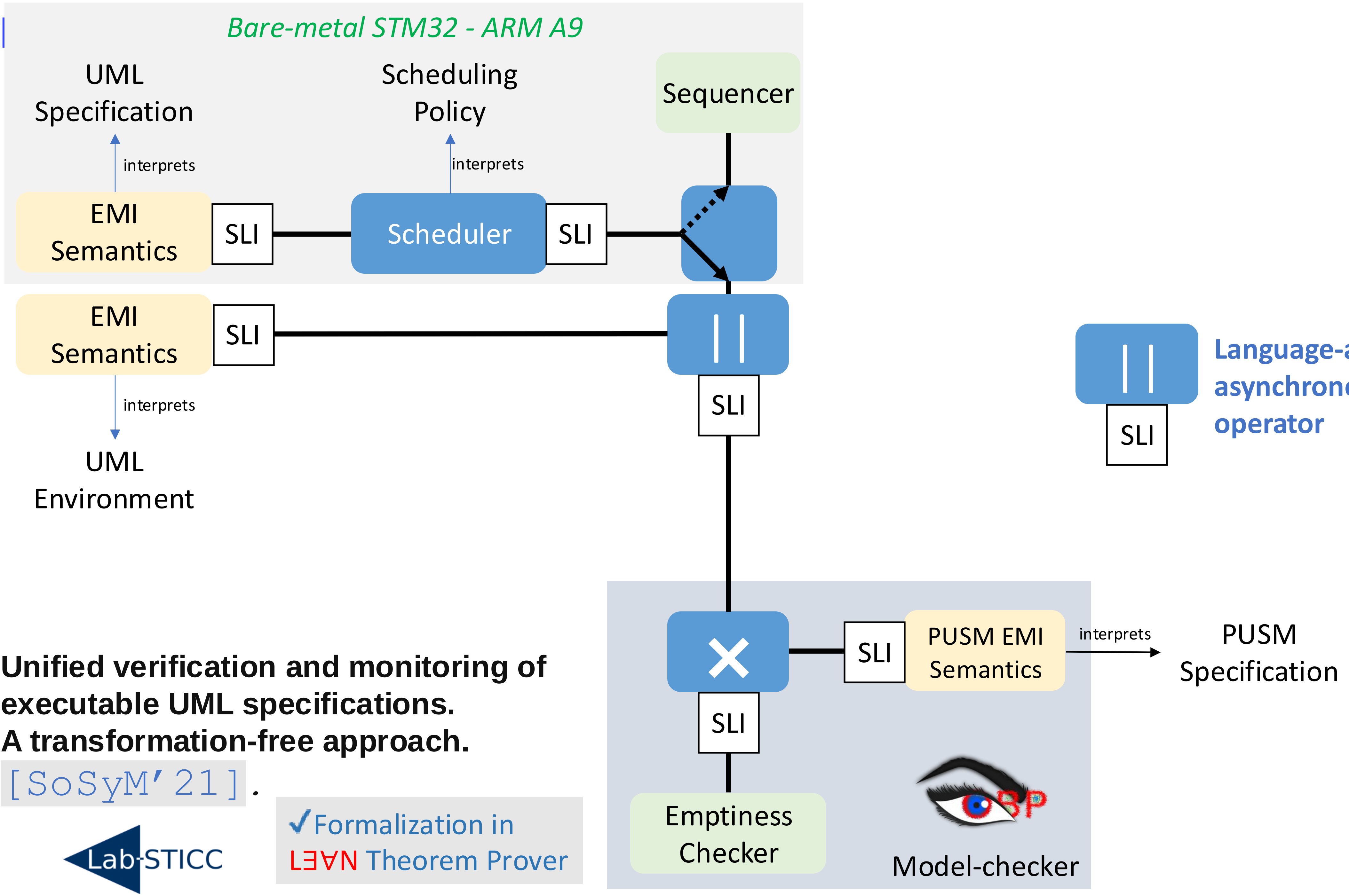
Model-checker

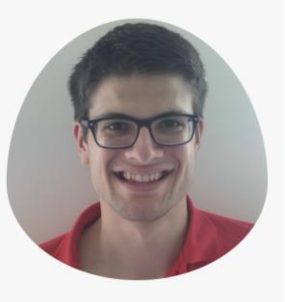


PhD Valentin BESNARD

Unified verification and monitoring of executable UML specifications.
A transformation-free approach.
[SoSyM’21].

✓Formalization in LEAN Theorem Prover





PhD Valentin BESNARD

| |
SLI

Language-agnostic asynchronous composition operator

Unified verification and monitoring of executable UML specifications. A transformation-free approach.

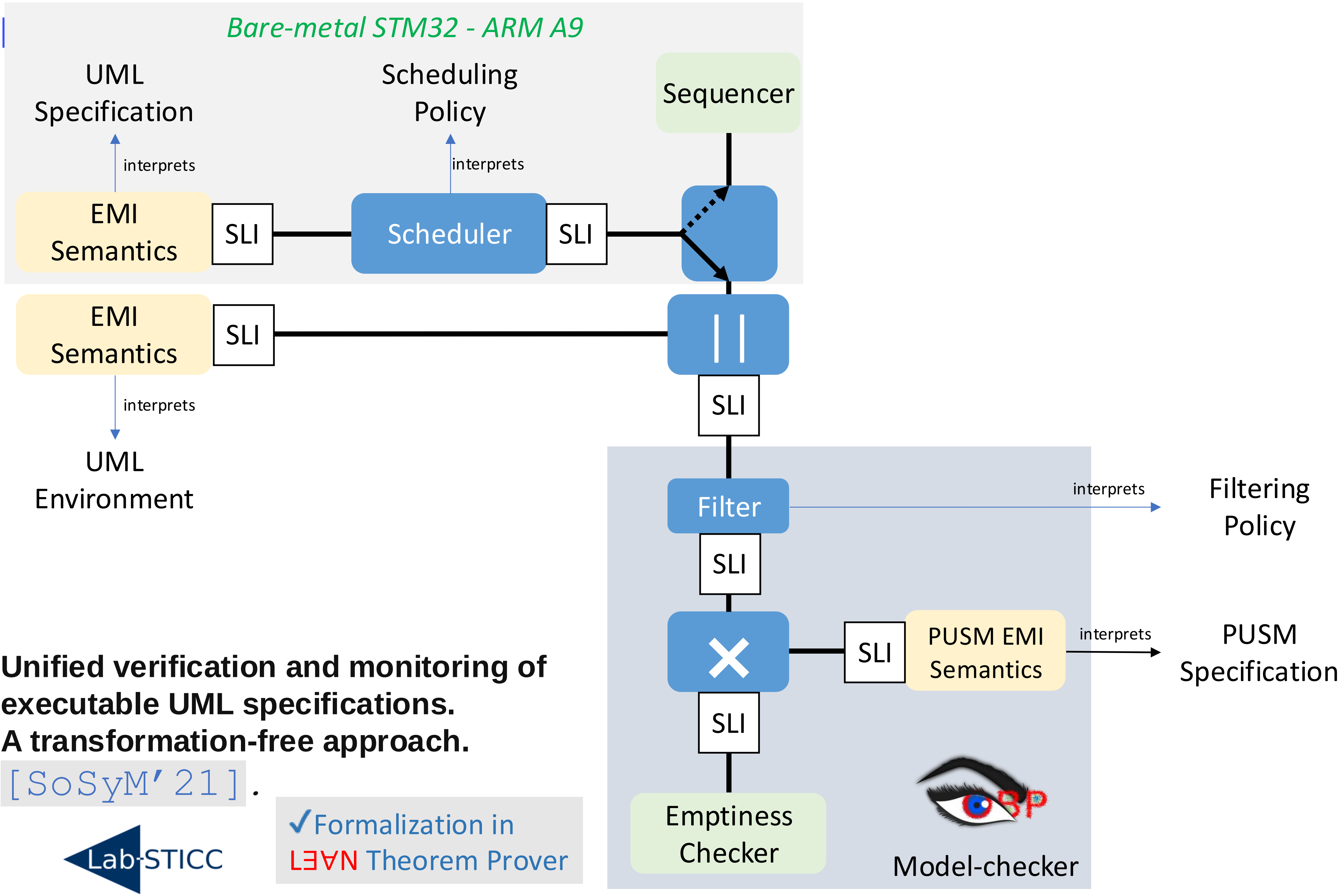
[SoSyM'21].

✓Formalization in LƎ∀N Theorem Prover

Lab-STICC

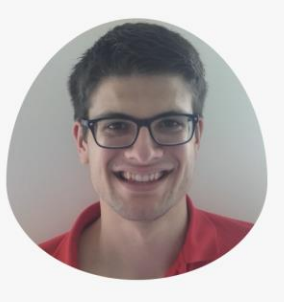
DAVIDSON CONSULTING

eseo GRANDE ÉCOLE D'INGÉNIEURS



Unified verification and monitoring of executable UML specifications. A transformation-free approach. [SoSyM'21].

✓Formalization in L∃∀N Theorem Prover

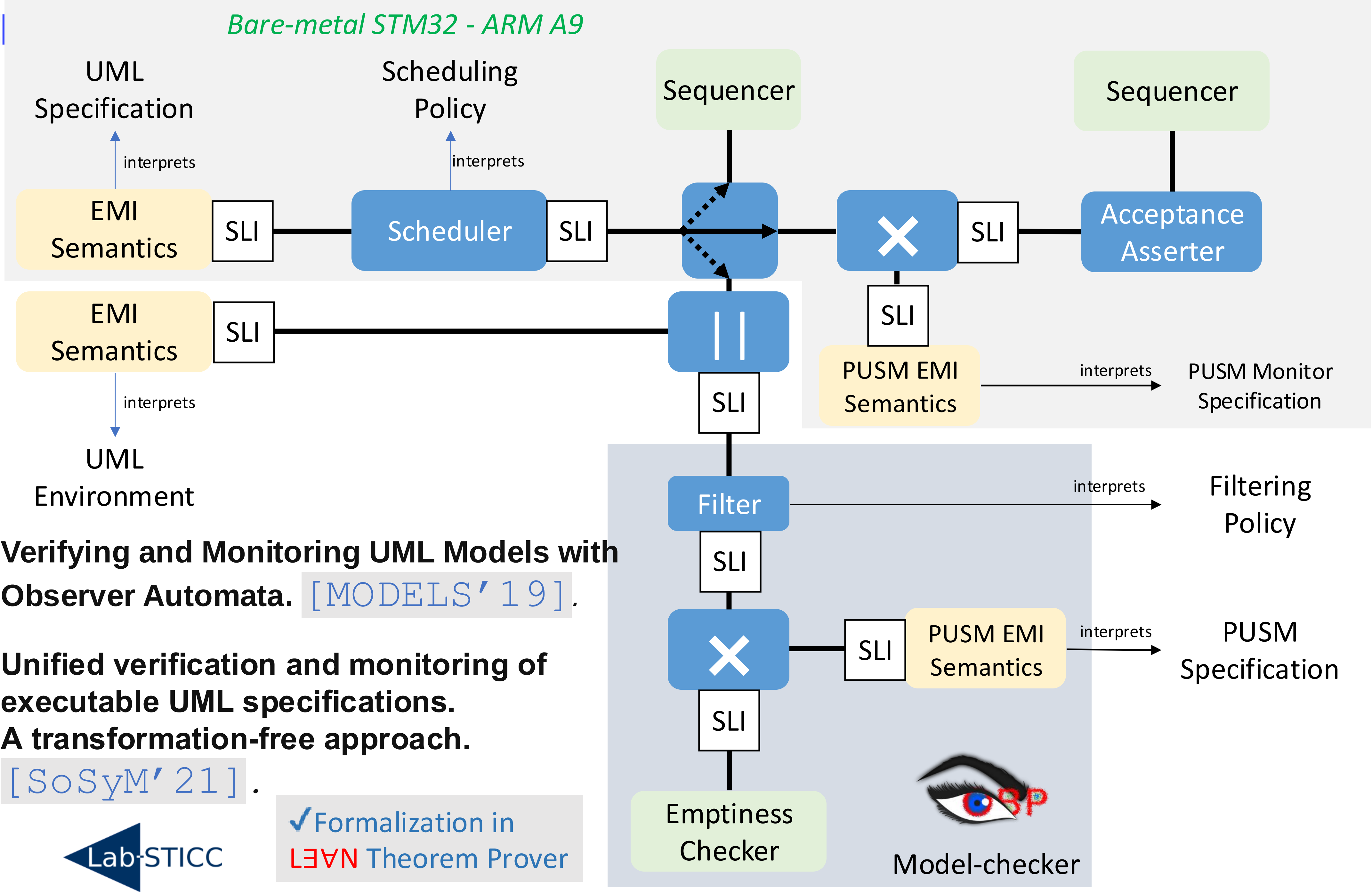


PhD Valentin BESNARD

Lab-STICC

DAVIDSON CONSULTING

eseo GRANDE ÉCOLE D'INGÉNIEURS



Bare-metal STM32 - ARM A9

UML Specification

interprets

EMI Semantics

SLI

Scheduling Policy

interprets

Scheduler

SLI

Sequencer

Sequencer

×

SLI

Acceptance Asserter

SLI

PUSM EMI Semantics

interprets

PUSM Monitor Specification

EMI Semantics

SLI

||

SLI

interprets

UML Environment

Filter

interprets

Filtering Policy

SLI

×

SLI

PUSM EMI Semantics

interprets

PUSM Specification

SLI

Emptiness Checker

Model-checker

Verifying and Monitoring UML Models with Observer Automata. [MODELS’19].

Unified verification and monitoring of executable UML specifications. A transformation-free approach. [SoSyM’21].

✓Formalization in LƎ∀N Theorem Prover

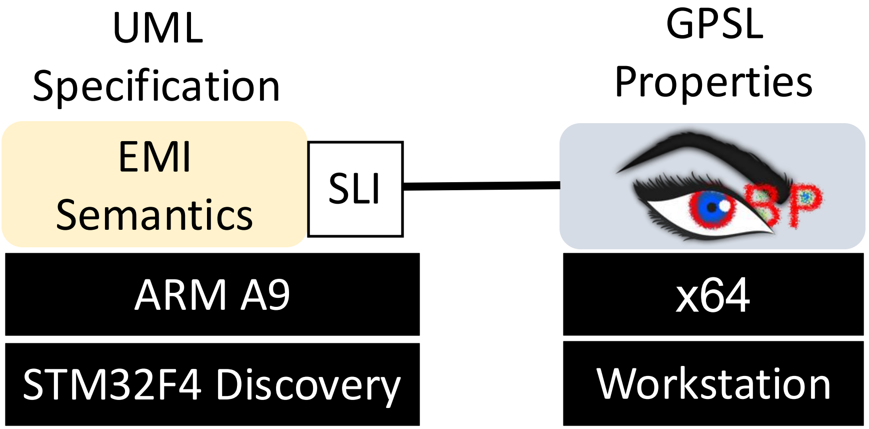
PhD Valentin BESNARD

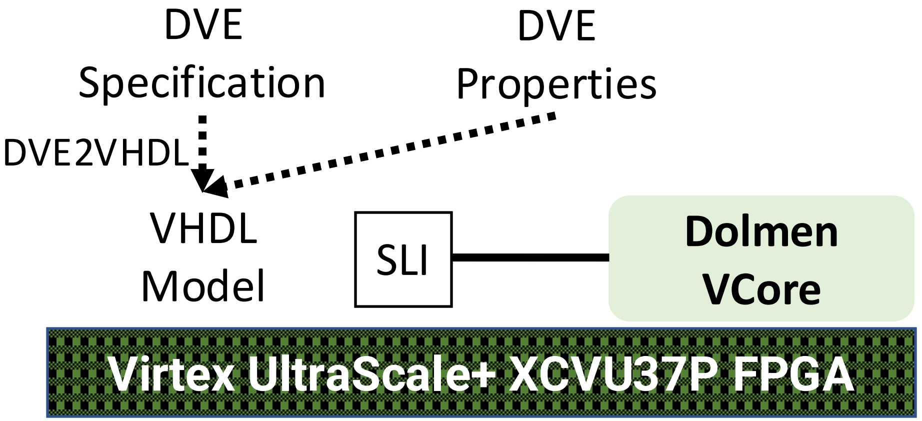
4. When G∀min∃ experiences the real world.

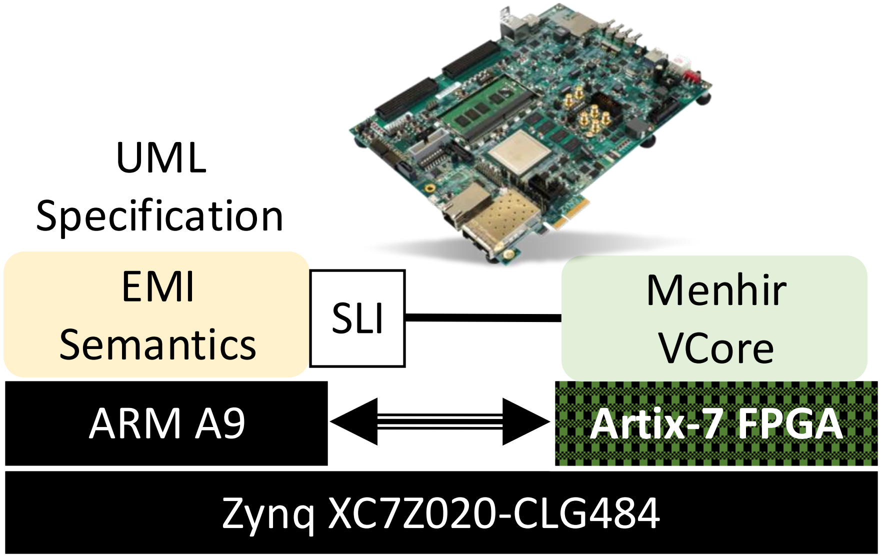
- Some experiences unravel reusable monitoring bridges
- **Exploring hardware execution**
- Multiverse debugging made simple and more powerful
- Transfer to commercial products -- *OBP2 inside*
- Transfer to future practitioners -- *From zero to model-checker*

From Embedded to Hardware Execution

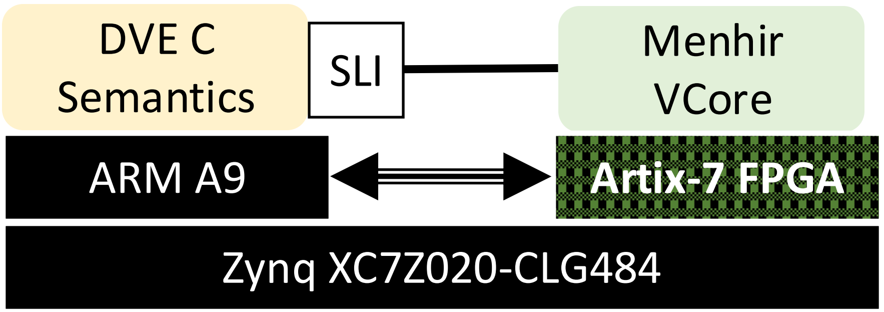
PhD Valentin BESNARD



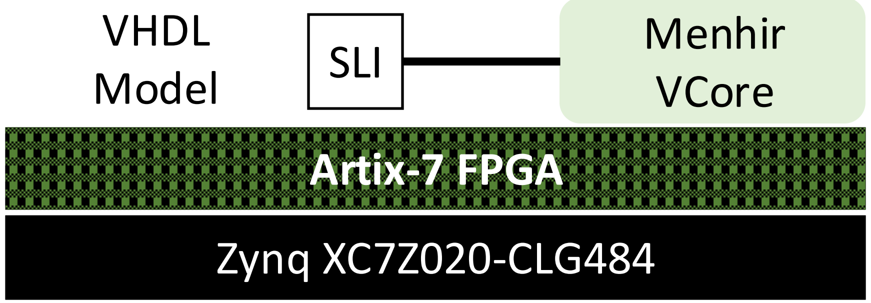




PhD Emilien FOURNIER



[DSD'20]
[FPL'21]
[DATE'22]

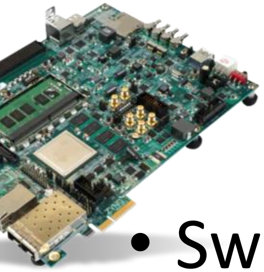


✓Guided by TLA+ formalization

Région Bretagne
CPER CyberSSI

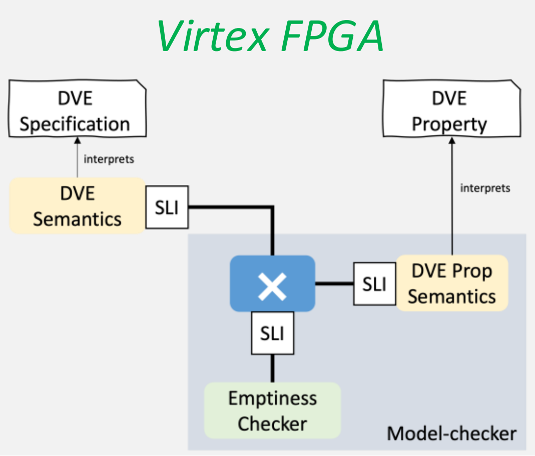
[DATE'22]

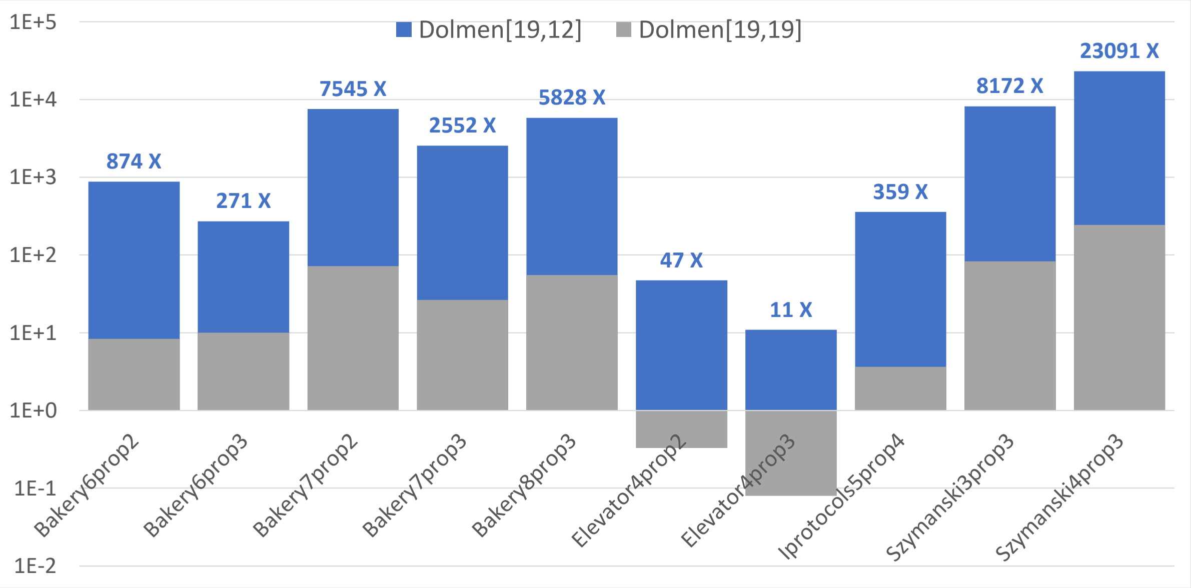
Dolmen: 1st Hardware Swarm Engine for **Both** Safety & Liveness Verification

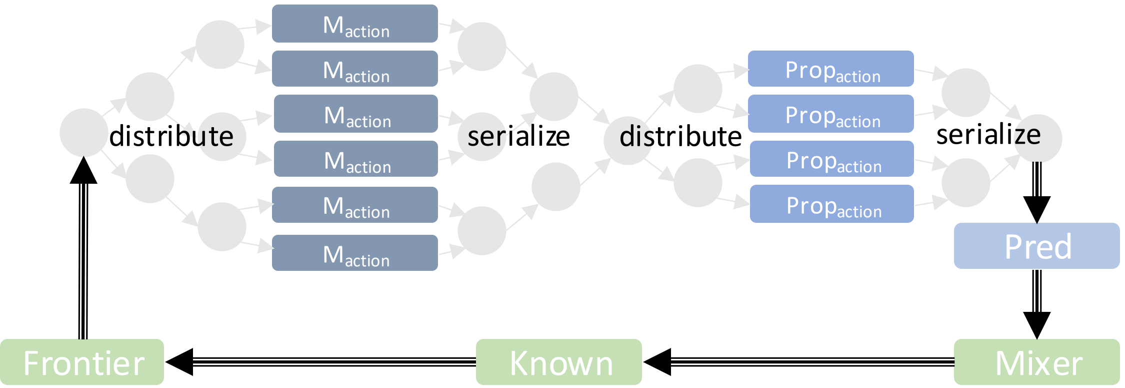


PhD Emilien FOURNIER

- Swarm of 32 deeply pipelined verification cores
- Distributed control architecture, for large SSI-FPGAs
- 4874x average speedup over software (Divine 3)







Région Bretagne
CPER CyberSSI

4. When G∀min∃ experiences the real world.

- Some experiences unravel reusable monitoring bridges
- Exploring hardware execution
- **Multiverse debugging made simple and more powerful**
- Transfer to commercial products -- *OBP2 inside*
- Transfer to future practitioners -- *From zero to model-checker*

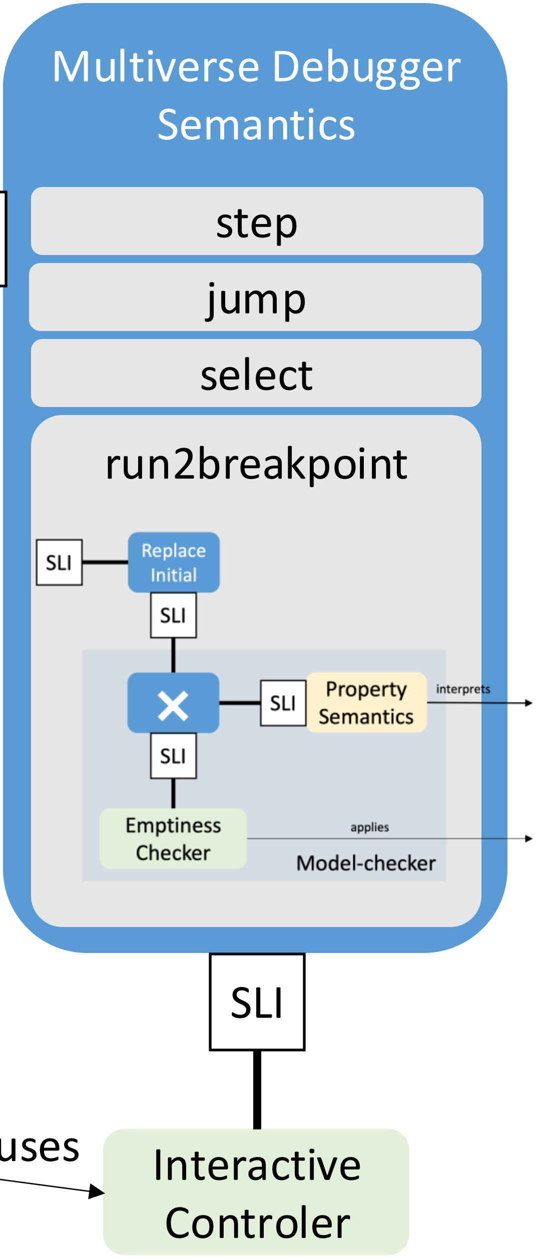
AnimUML Specification

interprets

AnimUML Semantics

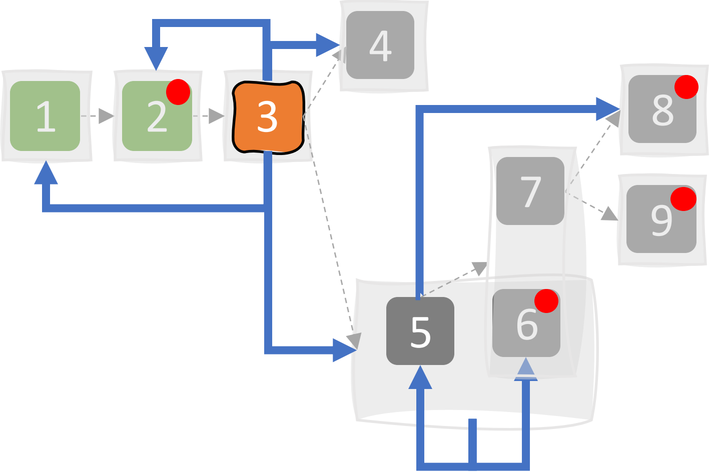
SLI

SLI

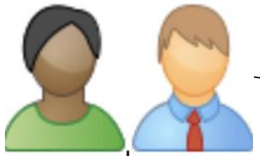


Temporal Breakpoints [SLE'23]

Reduction [MODELS'22]



User



uses

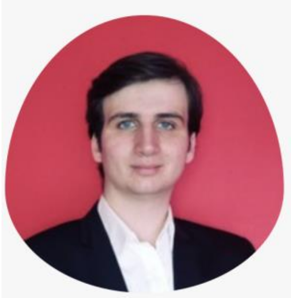
✓Formalization in LƎ∀N Theorem Prover

Language-agnostic

Non-trivial Monitor Composition

Scalability

+Expressivity, no instrumentation



PhD Matthias PASQUIER

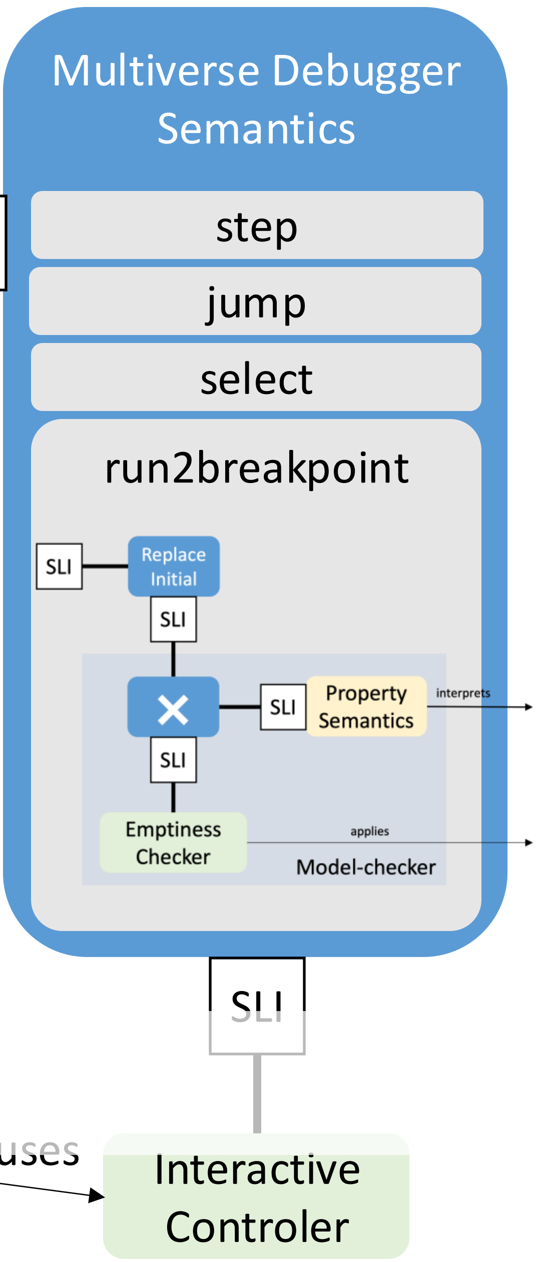
AnimUML Specification

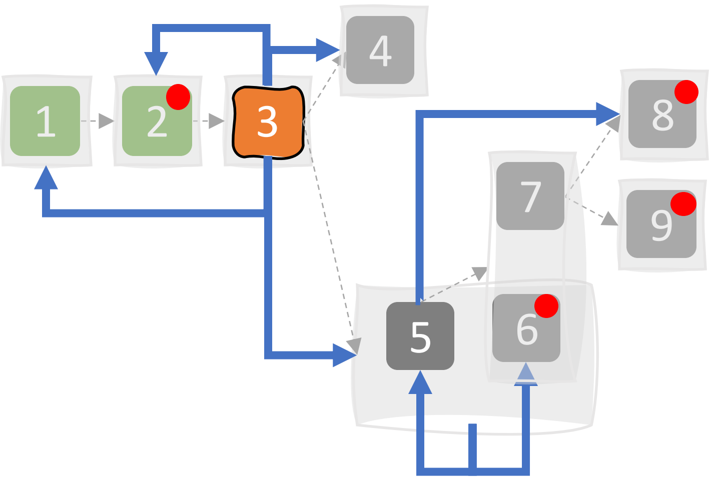
interprets

AnimUML Semantics

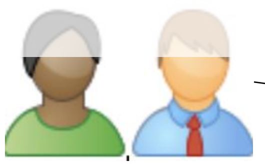
SLI

SLI



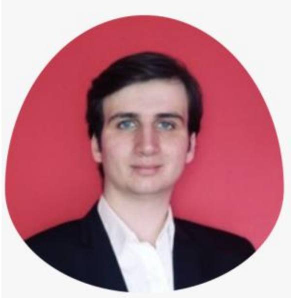


User



uses

✓Formalization in L∃∀N Theorem Prover



PhD Matthias PASQUIER

Language-agnostic

Non-trivial Monitor Composition

Scalability

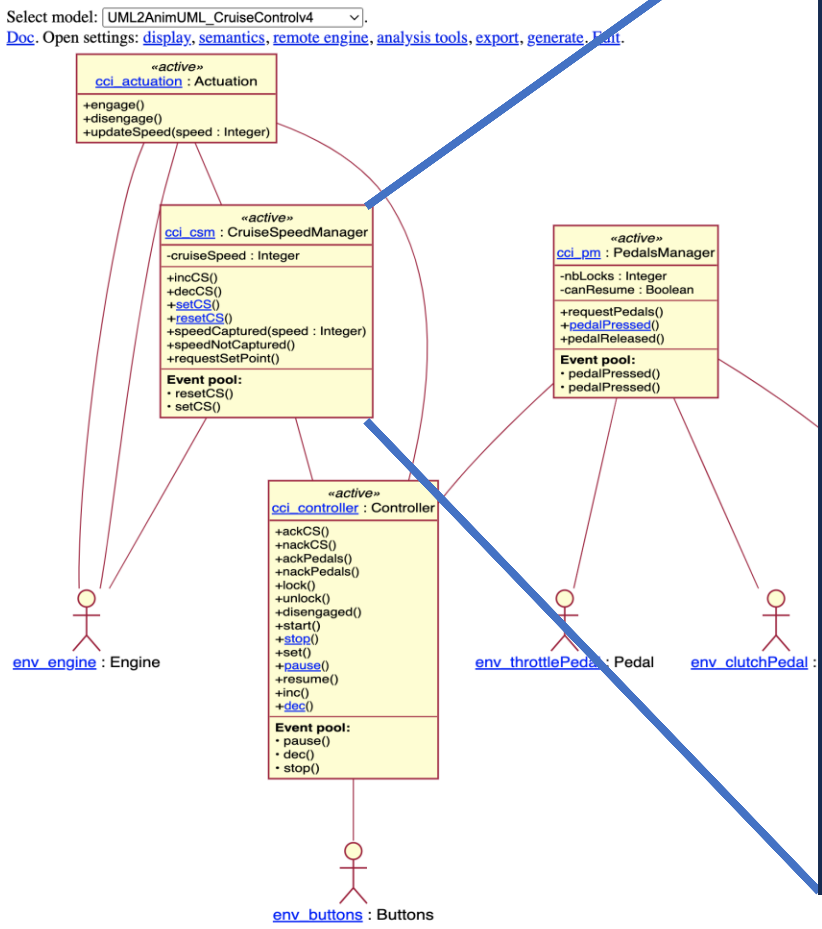
+Expressivity, no instrumentation

Temporal Breakpoints [SLE'23]

Reduction [MODELS'22]

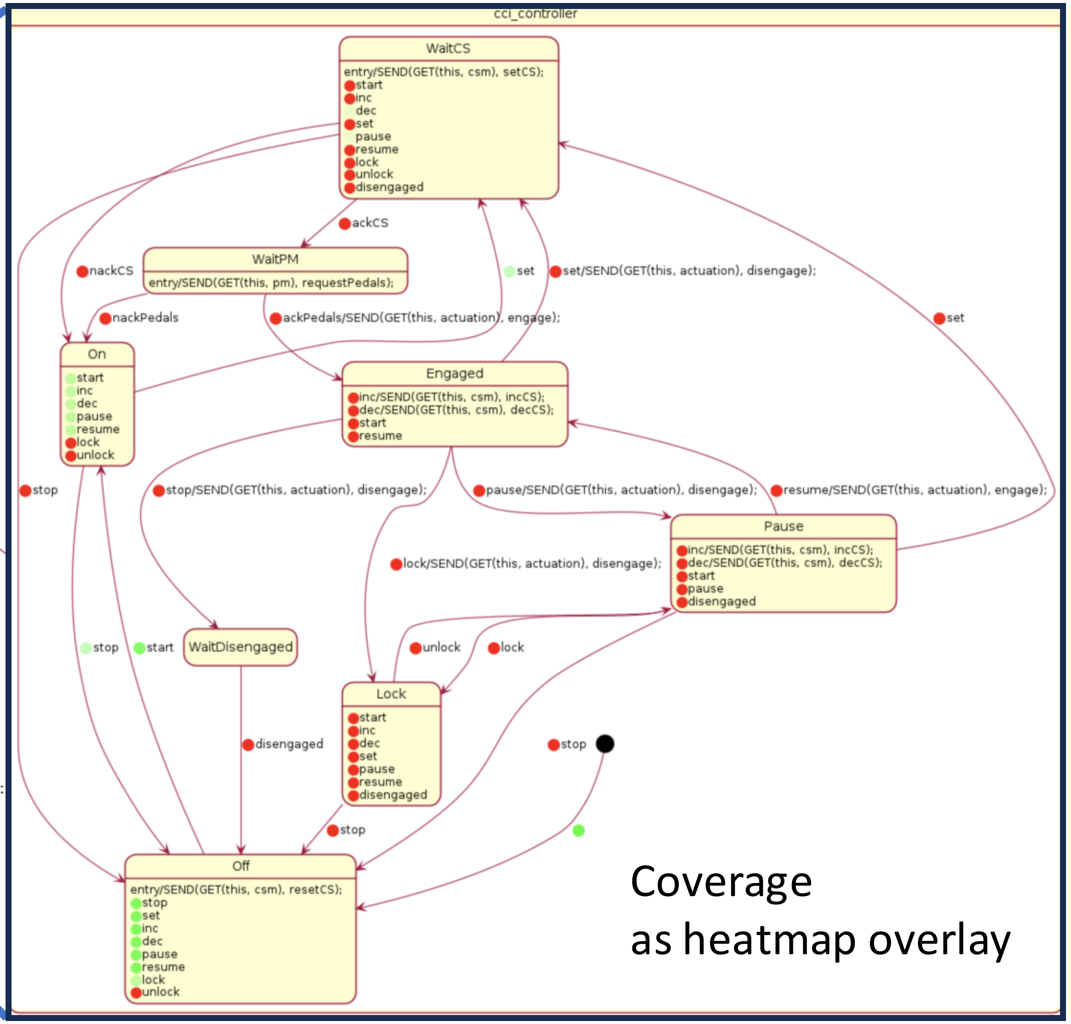
AnimUML

Designing, Animating, and Verifying Partial UML Models [MODELS'20].



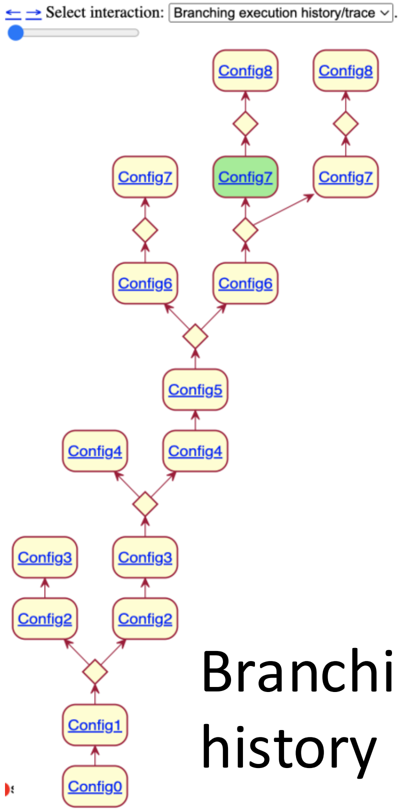
Select model: UML2AnimUML_CruiseControlv4

Doc. Open settings: display, semantics, remote engine, analysis tools, export, generate, Edit.



cci_controller
WaitCS
entry/SEND(GET(this, csm), setCS);
start
inc
dec
set
pause
resume
lock
unlock
disengaged
ackCS
WaitPM
entry/SEND(GET(this, pm), requestPedals);
nackCS
set
set/SEND(GET(this, actuation), disengage);
nackPedals
ackPedals/SEND(GET(this, actuation), engage);
set
On
start
inc
dec
pause
resume
lock
unlock
Engaged
inc/SEND(GET(this, csm), incCS);
dec/SEND(GET(this, csm), decCS);
start
resume
stop
stop/SEND(GET(this, actuation), disengage);
pause/SEND(GET(this, actuation), disengage);
resume/SEND(GET(this, actuation), engage);
Pause
inc/SEND(GET(this, csm), incCS);
dec/SEND(GET(this, csm), decCS);
start
pause
disengaged
lock/SEND(GET(this, actuation), disengage);
stop
start
WaitDisengaged
unlock
lock
Lock
start
inc
dec
set
pause
resume
disengaged
disengaged
stop
stop
Off
entry/SEND(GET(this, csm), resetCS);
stop
set
inc
dec
pause
resume
lock
unlock

Coverage
as heatmap overlay



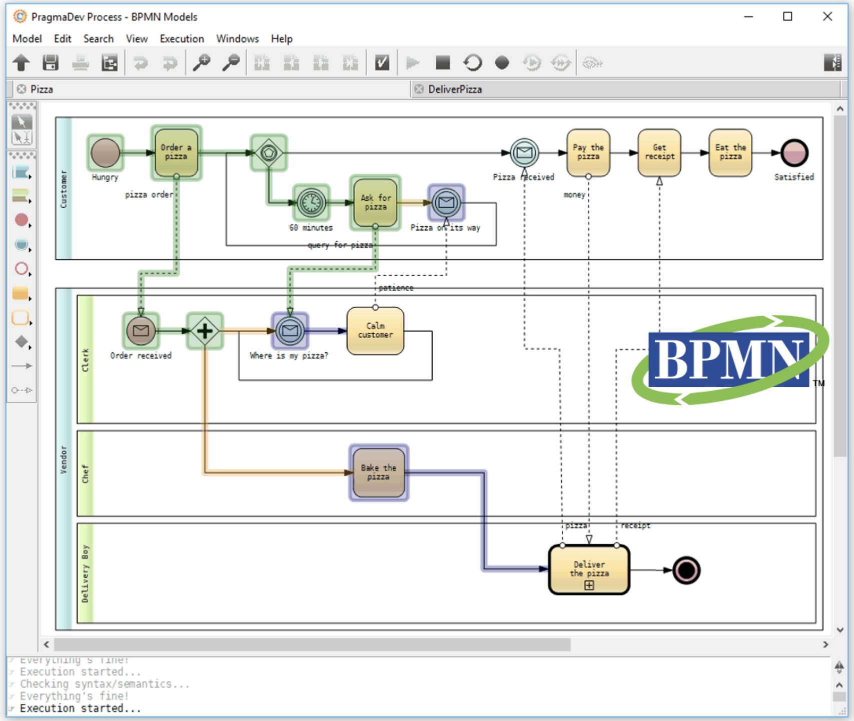
Select interaction: Branching execution history/trace

Branching history

https://github.com/ESEO-Tech/AnimUML

4. When G∀min∃ experiences the real world.

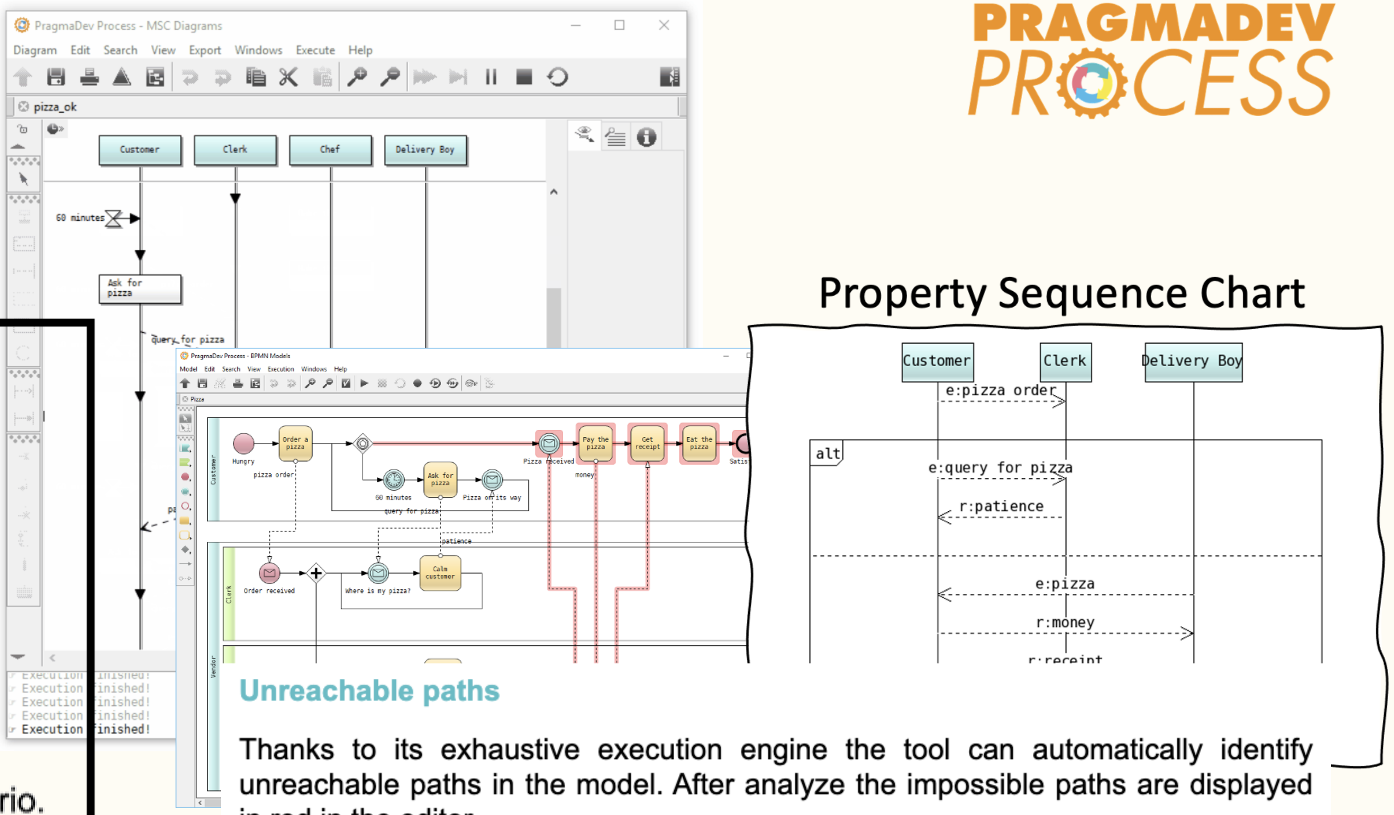
- Some experiences unravel reusable monitoring bridges
- Exploring hardware execution
- Multiverse debugging made simple and more powerful
- **Transfer to commercial products -- *OBP2 inside***
- Transfer to future practitioners -- *From zero to model-checker*



```
Everything's fine!
Execution started...
Checking syntax/semantics...
Everything's fine!
Execution started...
```

Coverage

Coverage information can be extracted and merged after each scenario.



```
Execution finished!
Execution finished!
Execution finished!
Execution finished!
Execution finished!
```

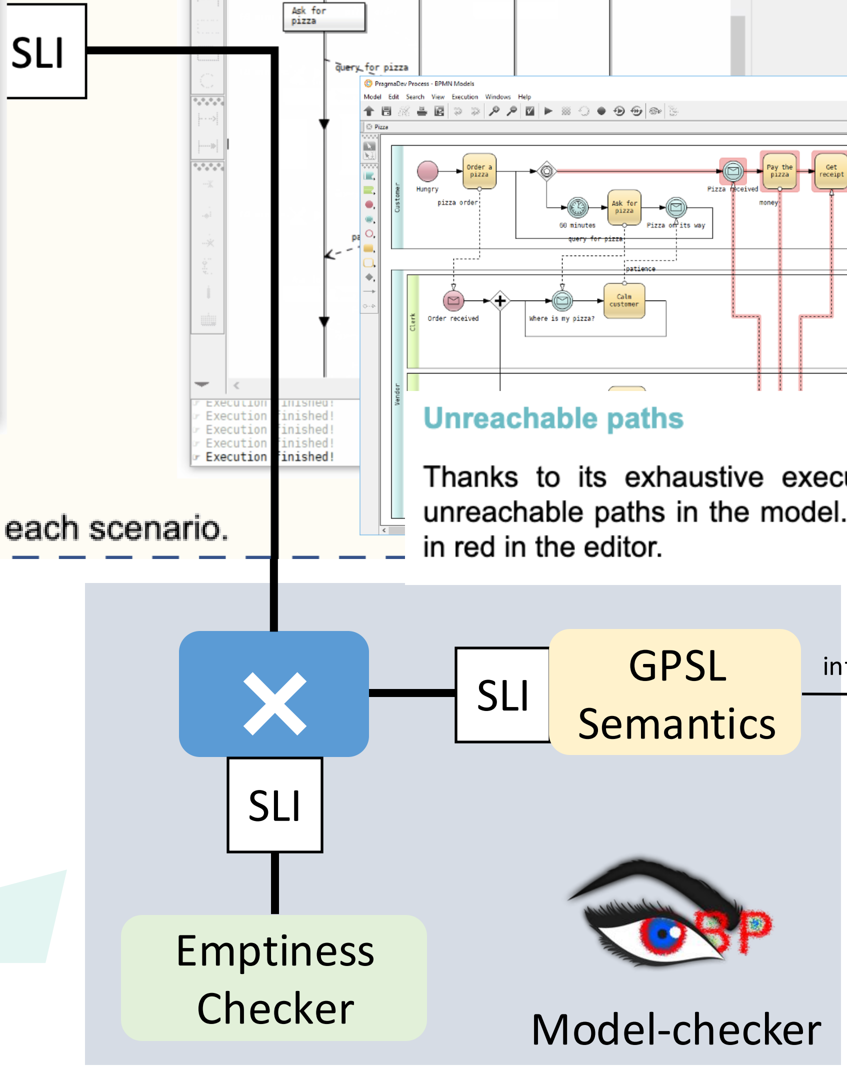
Unreachable paths

Thanks to its exhaustive execution engine the tool can automatically identify unreachable paths in the model. After analyze the impossible paths are displayed in red in the editor.

Press release

PRAGMADEV modeling and testing tools

PragmaDev Process, a new tool to verify business processes.



interprets

GPSL Specification

[ERTS’20]

[CSD&M’20]

RAPID VeriMoB

DGAC OneWay

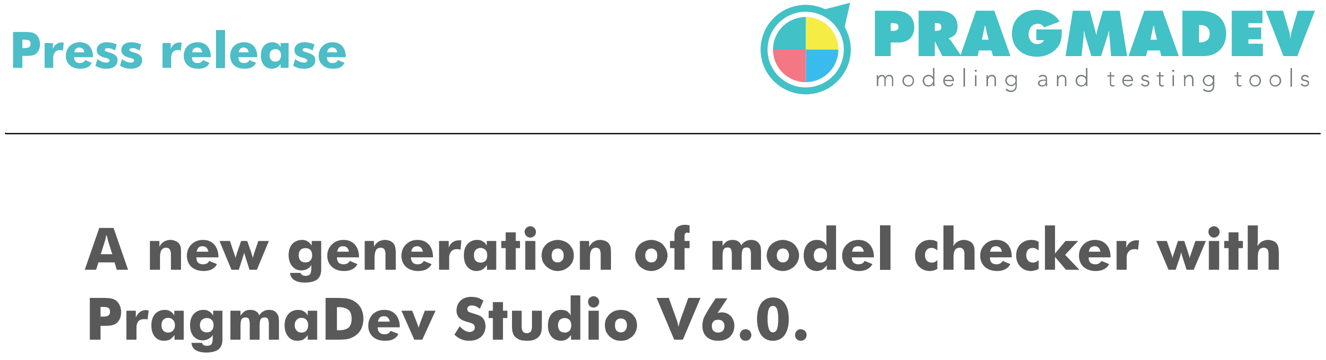
PRAGMADEV PROCESS

PRAGMADEV modeling and testing tools

Lab-STICC

Successful transfer to industry lead to

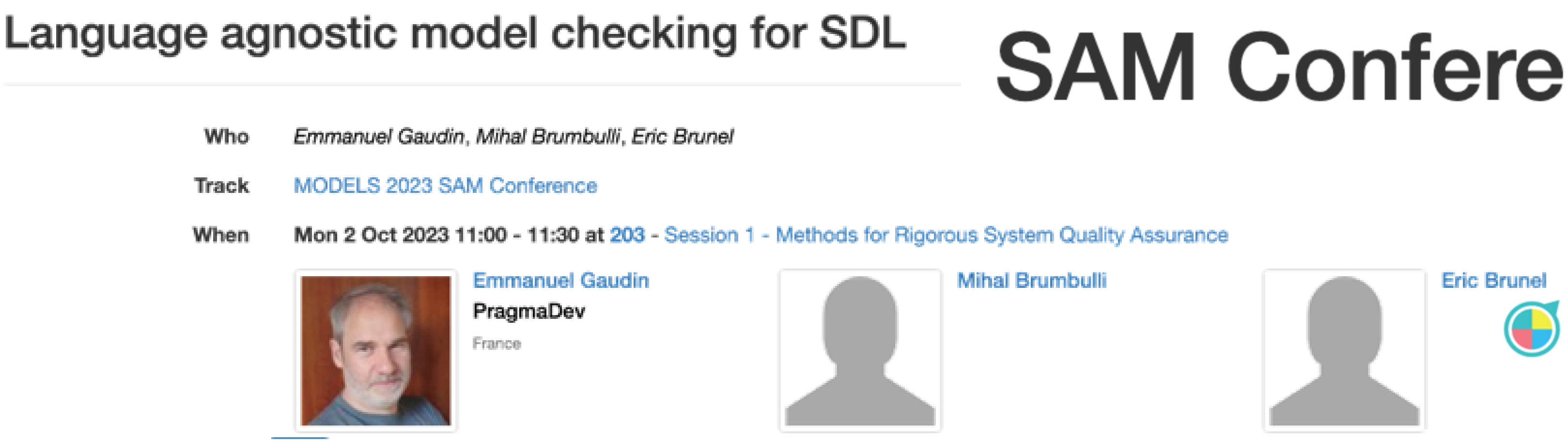
- **Adoption** in new products – PROCESS for BPMN
- **Empowering** the practitioners – coverage, unreachable paths, …
- **Retrofitting** existing products – STUDIO for SDL



Press release

PRAGMADEV modeling and testing tools

A new generation of model checker with PragmaDev Studio V6.0.



Language agnostic model checking for SDL

Who	*Emmanuel Gaudin, Mihal Brumbulli, Eric Brunel*
Track	MODELS 2023 SAM Conference
When	Mon 2 Oct 2023 11:00 - 11:30 at 203 - Session 1 - Methods for Rigorous System Quality Assurance

Emmanuel Gaudin
PragmaDev
France

Mihal Brumbulli

Eric Brunel

SAM Conference

4. When G∀min∃ experiences the real world.

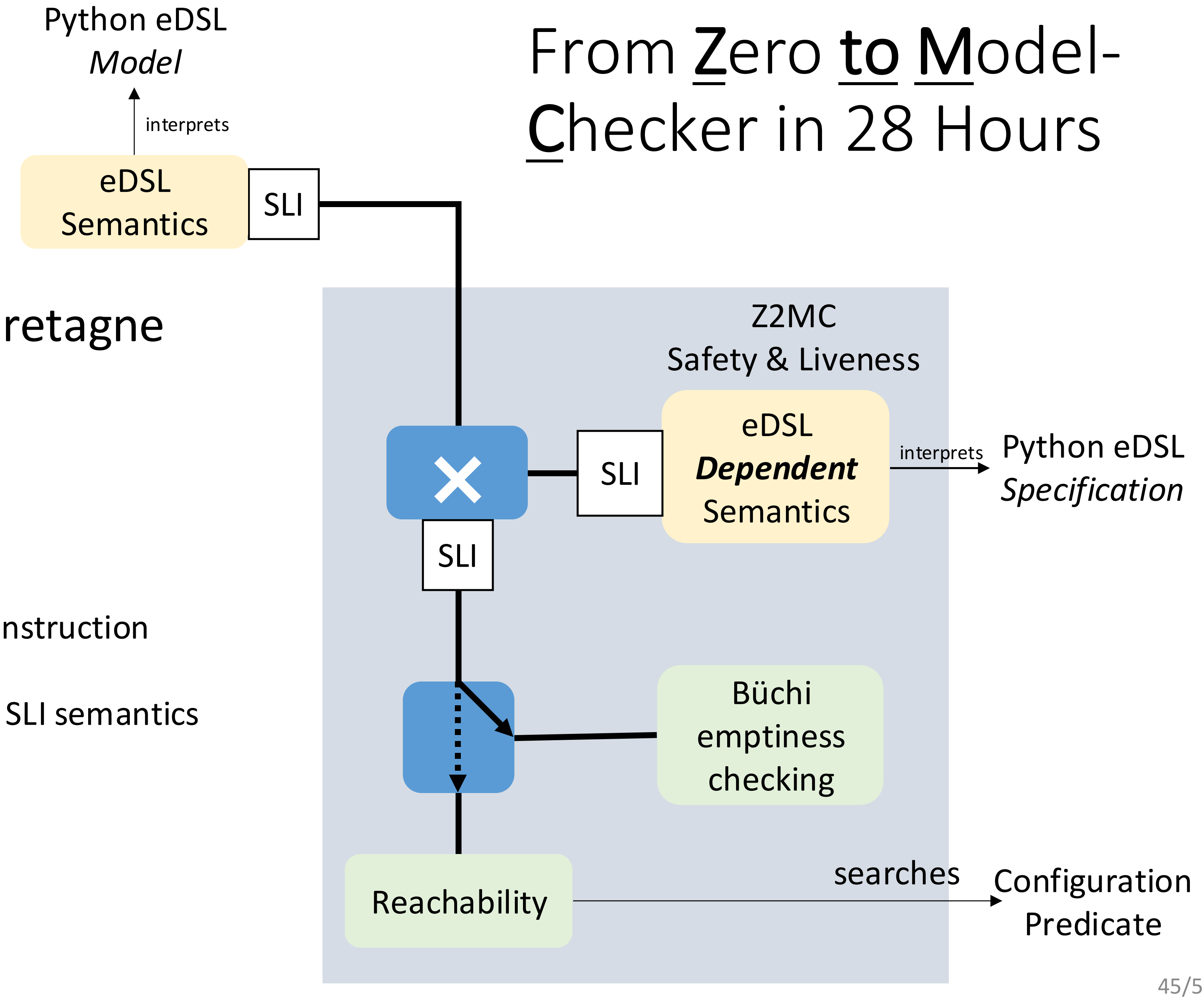
- Some experiences unravel reusable monitoring bridges
- Transfer to commercial products -- *OBP2 inside*
- Exploring hardware execution
- Multiverse debugging made simple and more powerful
- **Transfer to future practitioners -- *From zero to model-checker***

Transfer to Future Practioners

From Zero to Model-Checker in 28 Hours

Master-level class at ENSTA Bretagne
the last 2 years
seven 4-hour sessions

1. Model-independent graph traversal
2. Predicate-based search and witness construction
3. SLI by refactoring the graph API
4. Lambda-based guard-action eDSL with SLI semantics
5. Dependent SLI semantics
6. Step-based synchronous composition
7. Büchi emptiness checking



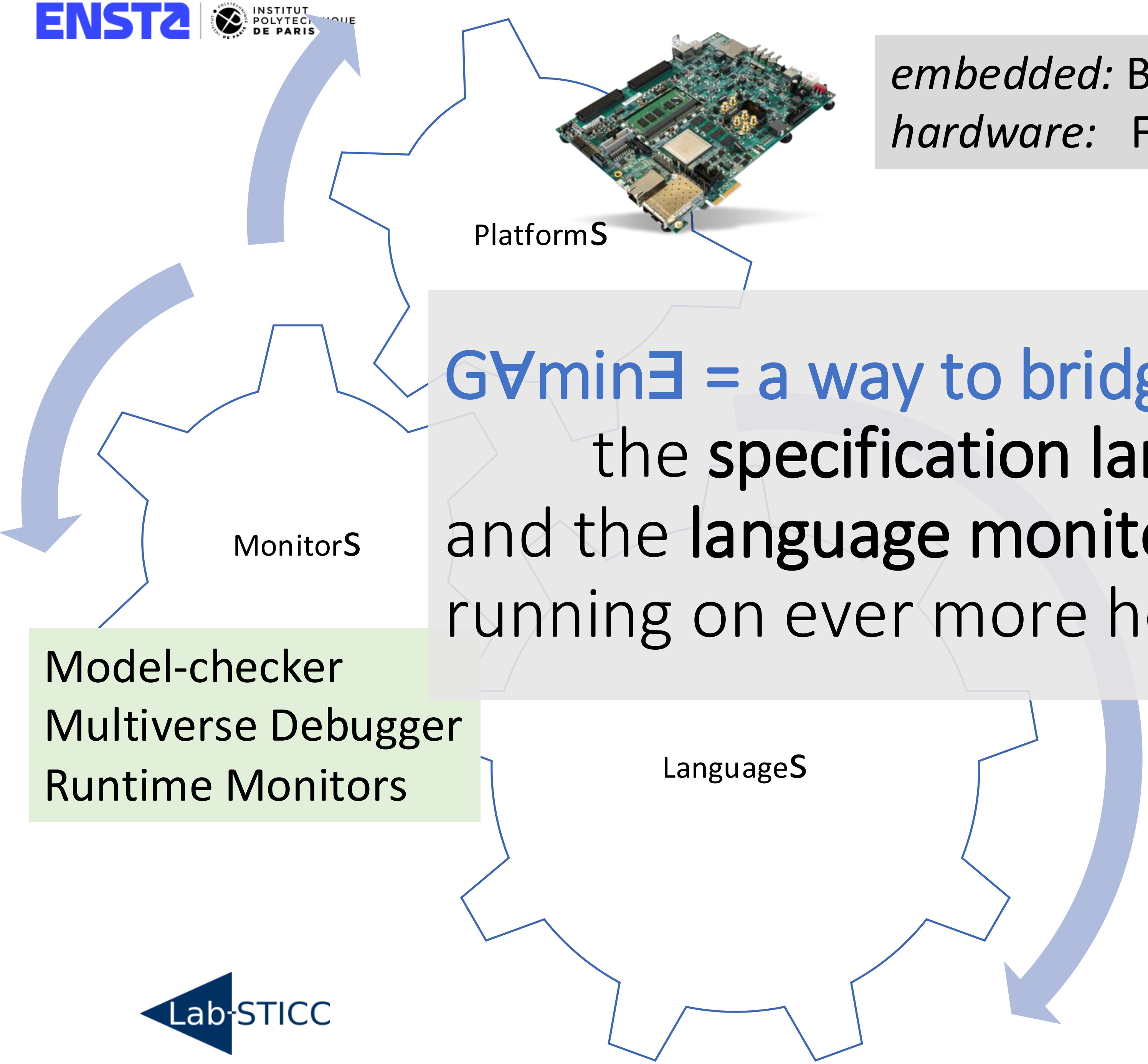
5. Sum Up & Ways Forward

Conclusion

Major Breakthroughs

Perspectives

Track Record



embedded: Bare-metal
hardware: FPGA

G∀min∃ = a way to bridge the gap between
the **specification languages**
and the **language monitors**
running on ever more heterogeneous **platforms**?

Model-checker
Multiverse Debugger
Runtime Monitors

industrial: BPMN, SDL
reuse: TLA+, Fiacre
academic: UML, AEFD

Major Contributions

A **sustainable** & **composable** approach for **language monitoring**

simple and versatile, the SLI offers a radically better cost structure
step-based evaluation plays a major role

1st Hardware Swarm Engine for **Both Safety and Liveness Verification**

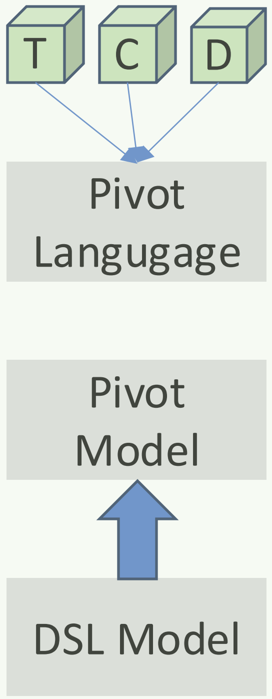
pipelined reformulation of the verification architecture

Established a **continuum** between **debugging** and **model-checking**

language-agnostic under-approximations for scalability
temporal breakpoints for expressivity without instrumentation

Manuel Wimmer
Keynote @ MLE’23

Reuse based on Pivot Language



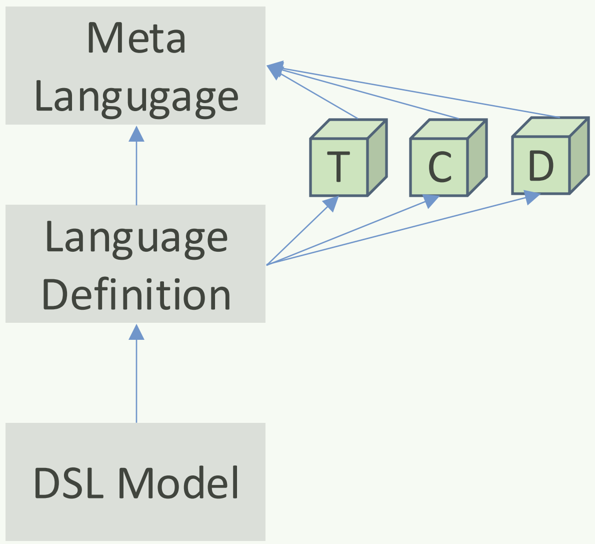
T C D

Pivot Langugage

Pivot Model

DSL Model

Reuse based on Meta-Language



Meta Langugage

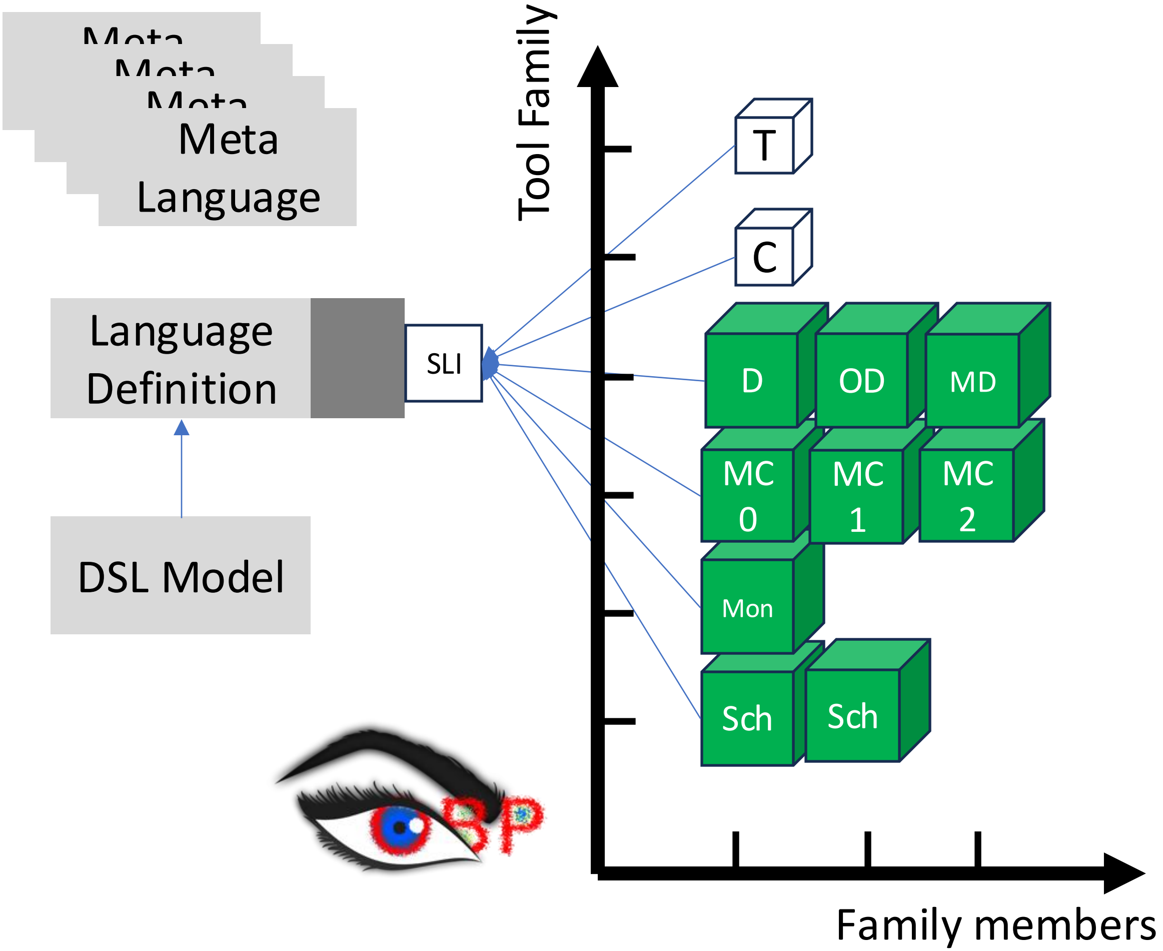
T C D

Language Definition

DSL Model

Hypothesis:
black box semantics
-> any meta-language

Reuse based on Confluent Tool Requirements



Meta
Meta
Meta
Meta Language

Language Definition

SLI

DSL Model

Tool Family

T
C
D OD MD
MC 0 MC 1 MC 2
Mon
Sch Sch

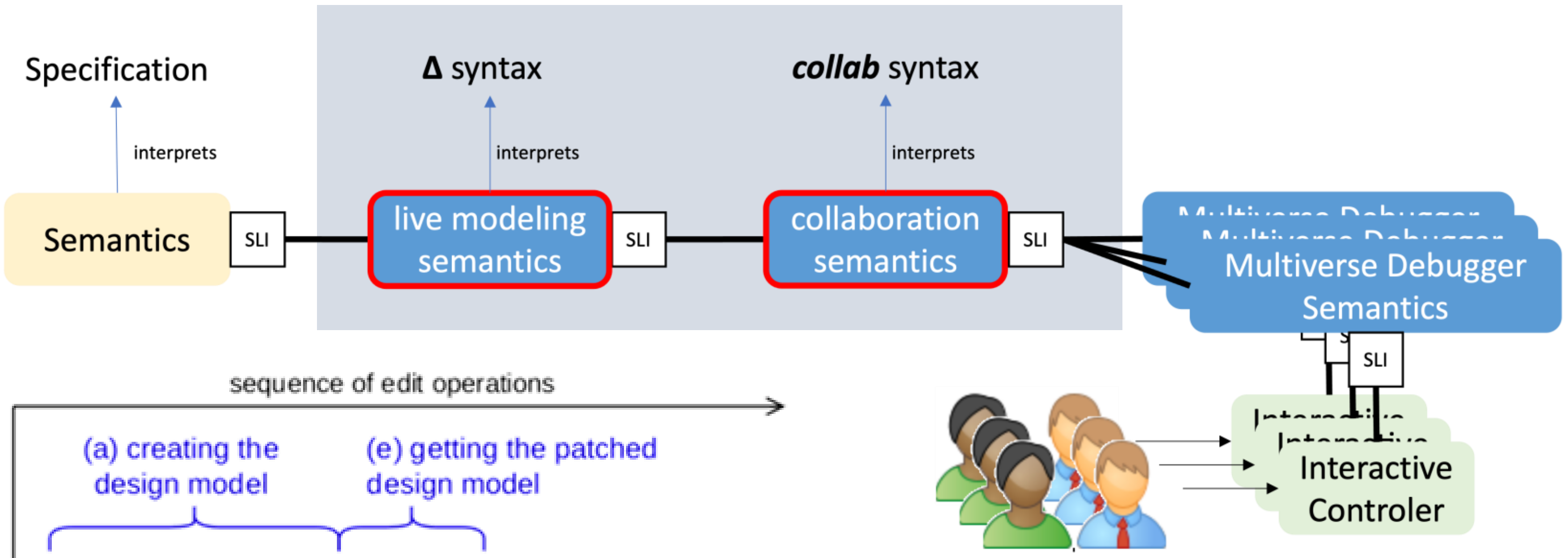
Family members

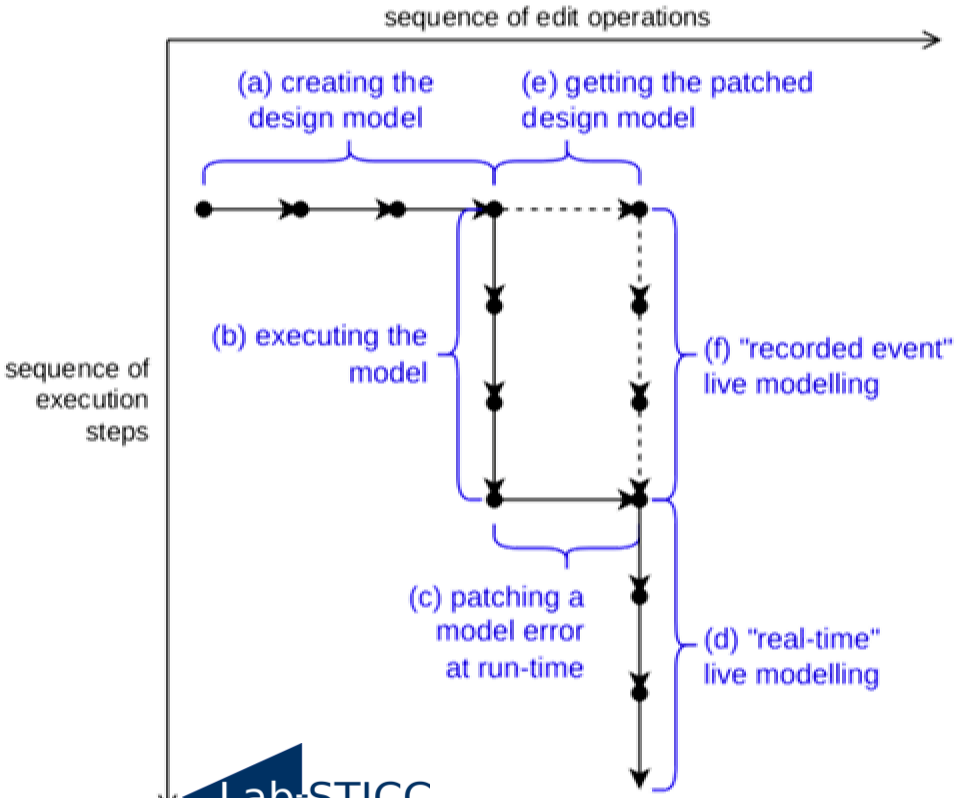
Ways Forward

Generalizing the G∀min∃ language monitoring for the future of specification-driven engineering.

Collaborative Live Modelling

Joeri EXELMANS





sequence of edit operations
sequence of execution steps
(a) creating the design model
(e) getting the patched design model
(b) executing the model
(f) "recorded event" live modelling
(c) patching a model error at run-time
(d) "real-time" live modelling

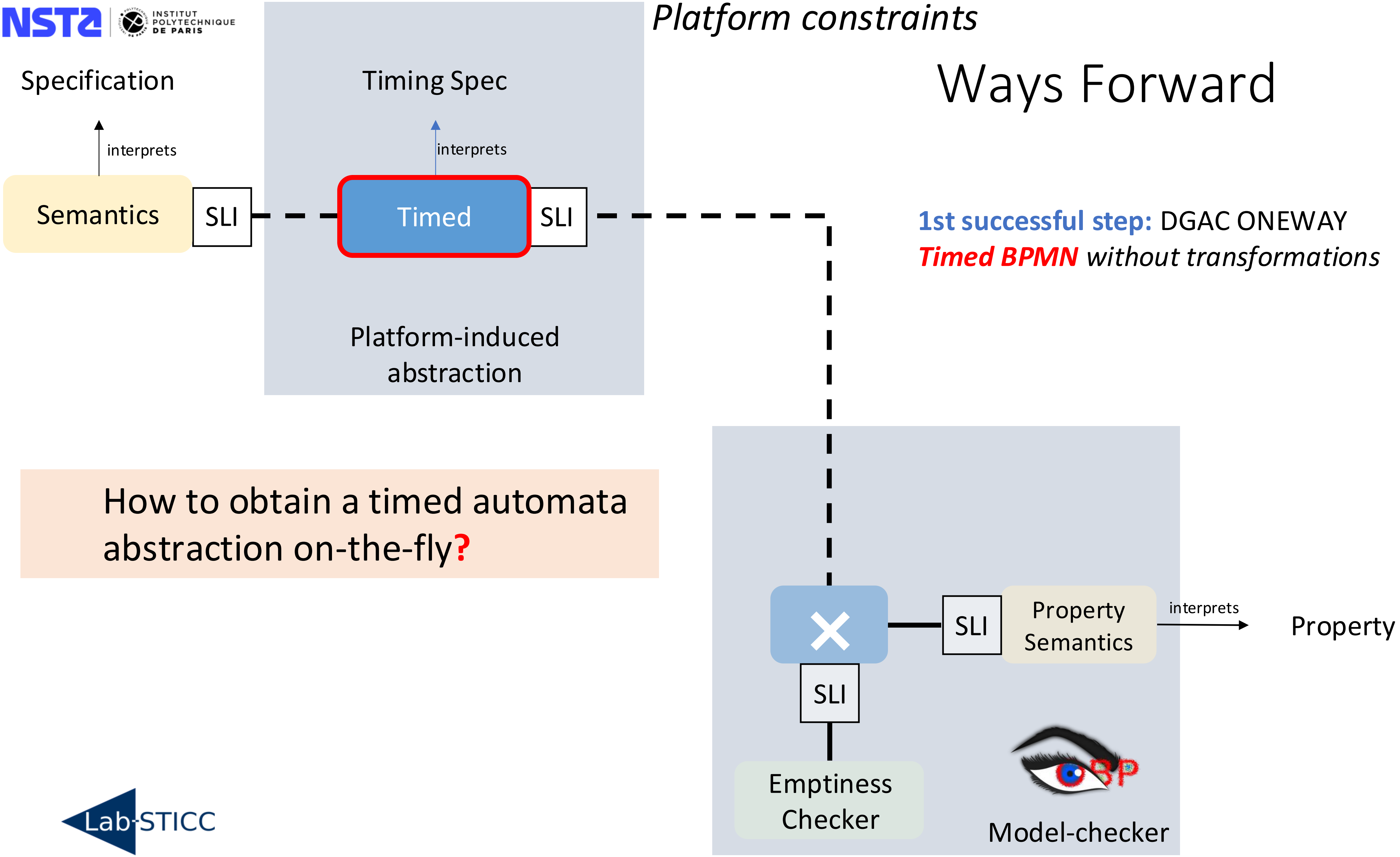
Composition?

Language-agnostic?
Without redoing the language

[…]
[SoSyM'24]
[MLE'23]

Ways Forward

Platform constraints

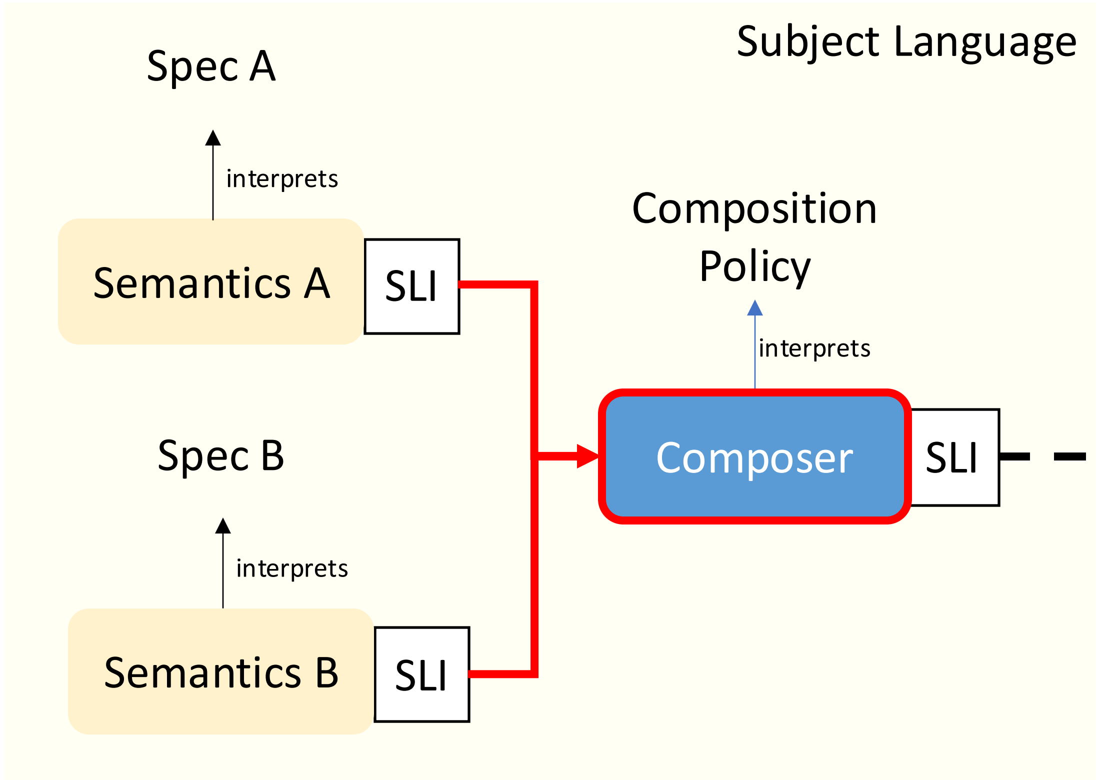


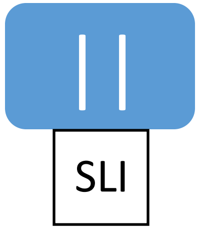
1st successful step: DGAC ONEWAY

Timed BPMN *without transformations*

How to obtain a timed automata abstraction on-the-fly**?**

Ways Forward

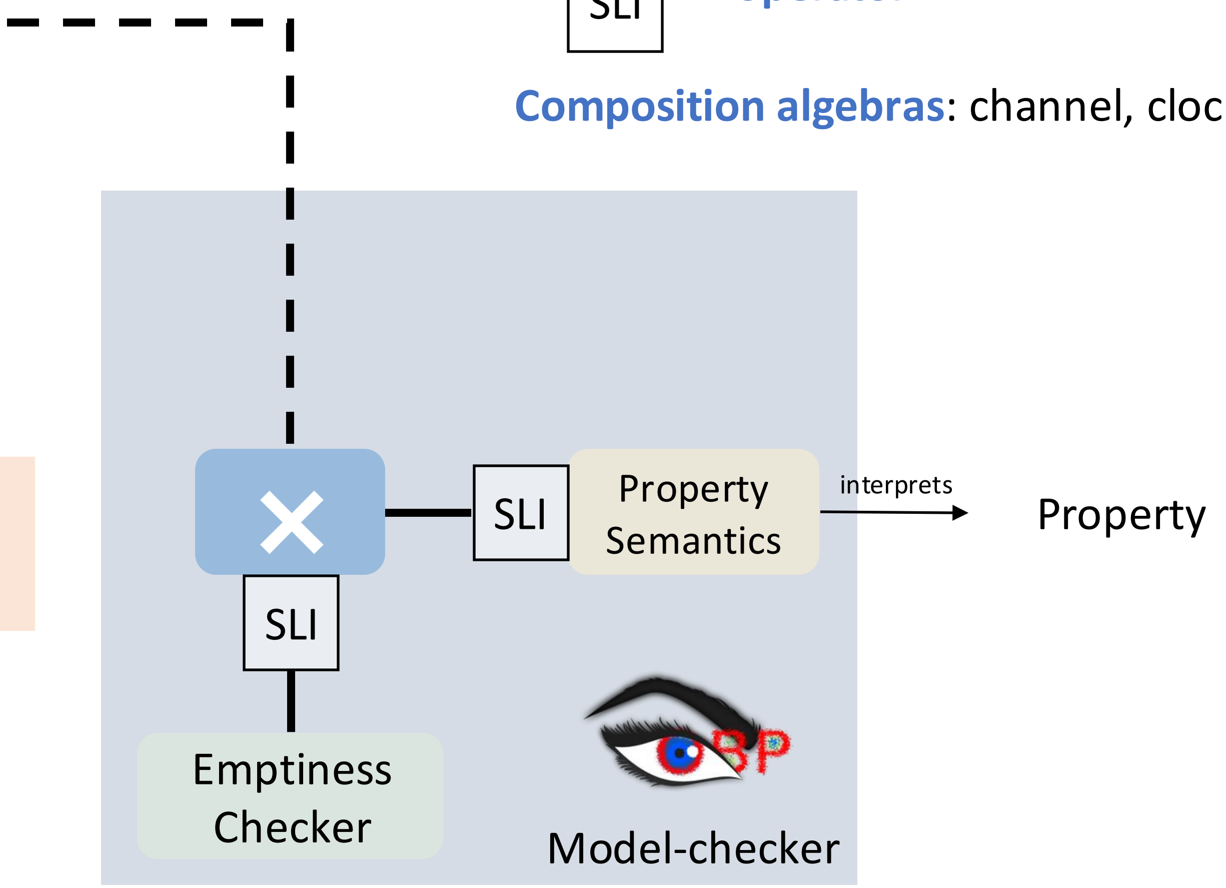




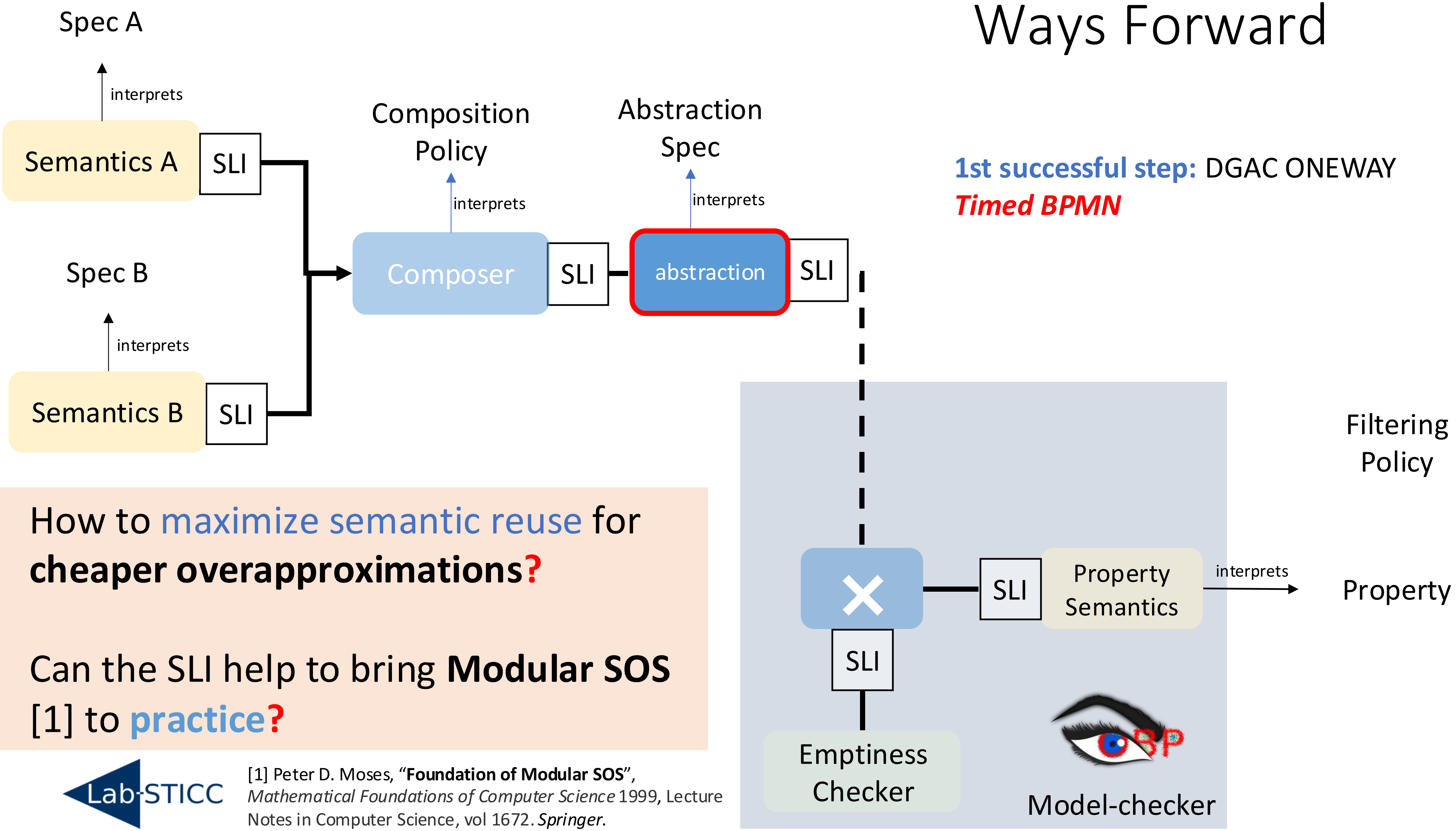
Language-agnostic asynchronous composition operator

Composition algebras: channel, clocks, events

Can the SLI be used for defining semantic-level operators?



Ways Forward



1st successful step: DGAC ONEWAY

Timed BPMN

How to maximize semantic reuse for **cheaper overapproximations?**

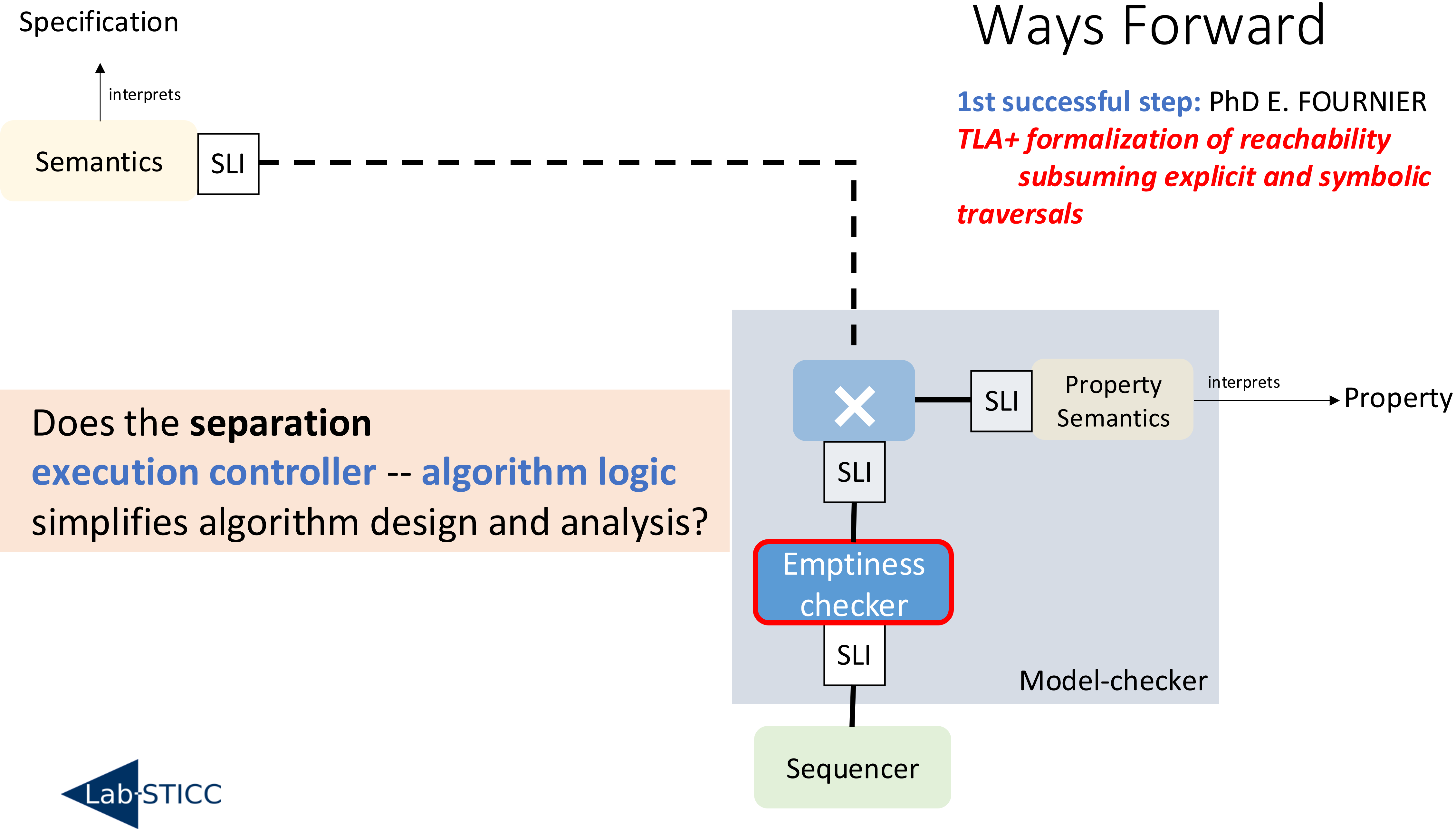
Can the SLI help to bring **Modular SOS** [1] to **practice?**

[1] Peter D. Moses, "**Foundation of Modular SOS**", *Mathematical Foundations of Computer Science* 1999, Lecture Notes in Computer Science, vol 1672. *Springer*.

Ways Forward

1st successful step: PhD E. FOURNIER

TLA+ formalization of reachability subsuming explicit and symbolic traversals



Does the **separation**
execution controller -- **algorithm logic**
simplifies algorithm design and analysis?

Ways Forward

How to standardize the SLI?

- Harmonization with the LSP and Debug Adapter Protocol

How to get a provably sound *language-agnostic* **portfolio-based diagnosis** toolkit?
Will it be fast enough?

How to Survive the Multicore Software Revolution

How to write code that will survive the ~~many-core~~ revolution S

Generalizing the G∀m ∃ language oring

for t

lution in software

Understanding DevOps: A Revolution in Software

Softw

Are live specification environments the ***next revolution?***

Let's get cracking, and Talk About It.