www.obpcdl.org

G∀min∃: Exploring the Boundary between Executable Specification Languages & Behavior Analysis Tools

Habilitation à diriger des recherches

Ciprian TEODOROV

https://www.ensta-bretagne.fr/teodorov

ciprian.teodorov@ensta-bretagne.fr

P4S, Lab-STICC, UMR CNRS 6285

ENSTA Bretagne Lab-STICC

3 april 2023

Overview

1. Executable specifications & behavior analysis monitors
2. Transformations: The shy semantics and the inaccessible monitors.
3. When the semantics decides to open 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

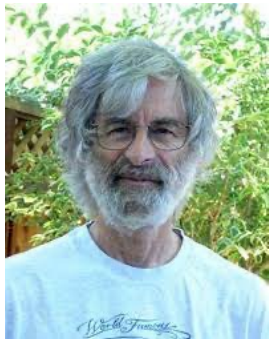
abstractions (models) focused on the domain of discourse.	**tools** (conceptual or computer-assisted) adapted to the domain

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 reflect how the object behaves



Descriptive [2]:
- presenting observations about the characteristics of something
- factually grounded or informative rather than normative, prescriptive or emotive

ENSTA Bretagne Lab-STICC

[1] Leslie Lamport: Thinking Above the Code
[2] (https://www.merriam-webster.com/dictionary/descriptive)

a Zoo of Executable Specification Languages

$\frac{dy}{dx}$

Physical processes

- Calculus [Newton and Leibniz]

$[](<>\psi)$

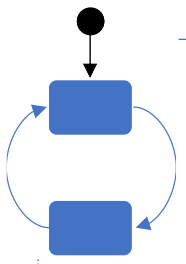
Temporal logic

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

$\lambda f.\lambda x.x$

Computable functions

- Lambda calculus
- Turing machines



Automata

- NFA
- PDA
- Statecharts

$a \rightarrow P \sqcap b \rightarrow Q$

Concurrency

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

```
entity AND is
 port (
  x: in  std_logic;
  y: in  std_logic;
  o: out std_logic);
end AND;
```

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:

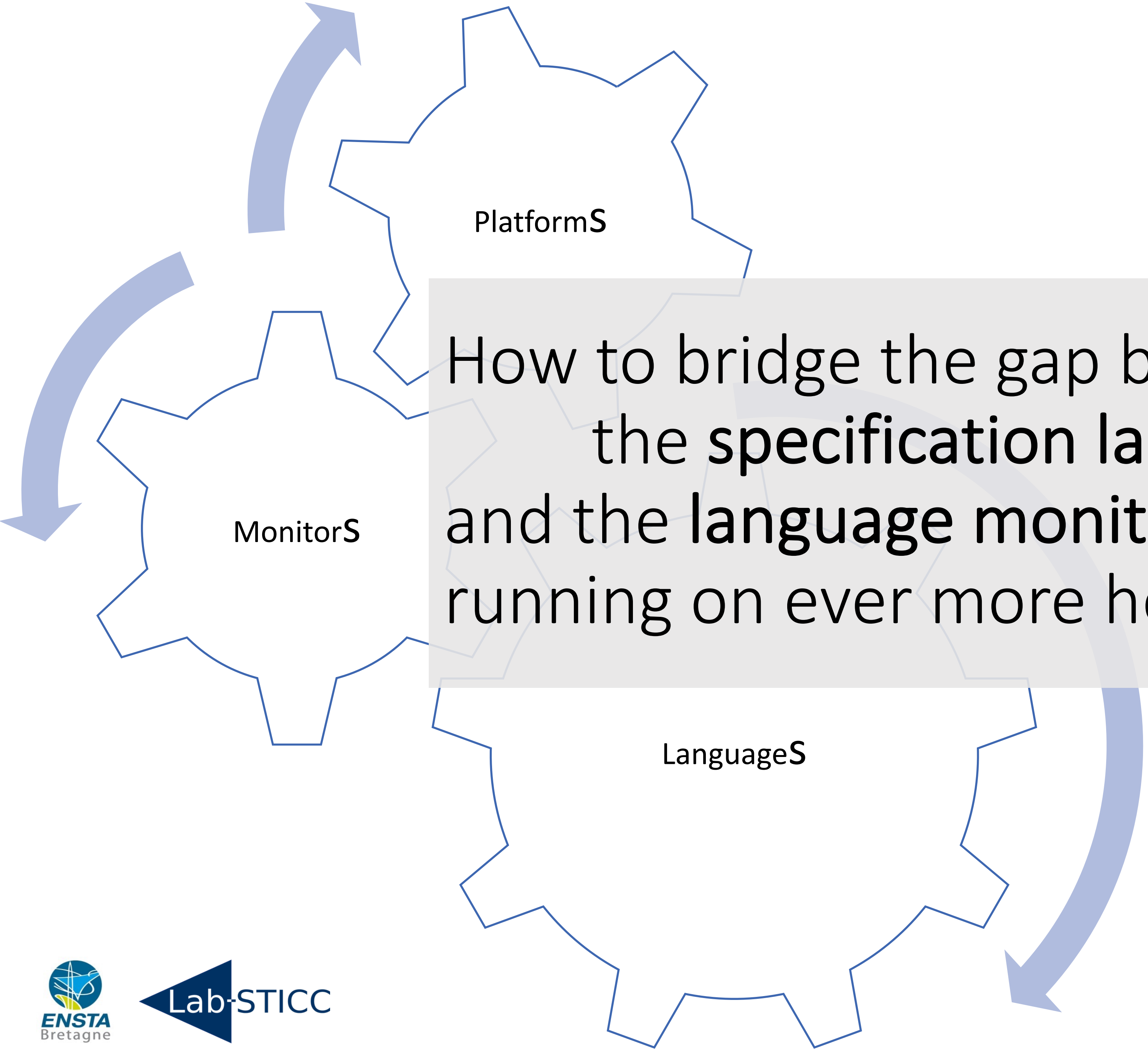
***language* monitors**, or simply **monitors**

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

a Zoo of Language Monitors

Executor	Monitor	**Debugger** • Moldable [1] • Omniscient [2] • Multiverse [3]
Profiler • MetaSpy [4] • DSProfile [5]	Tracer	**Model-checker** • LTSmin [6]

[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.



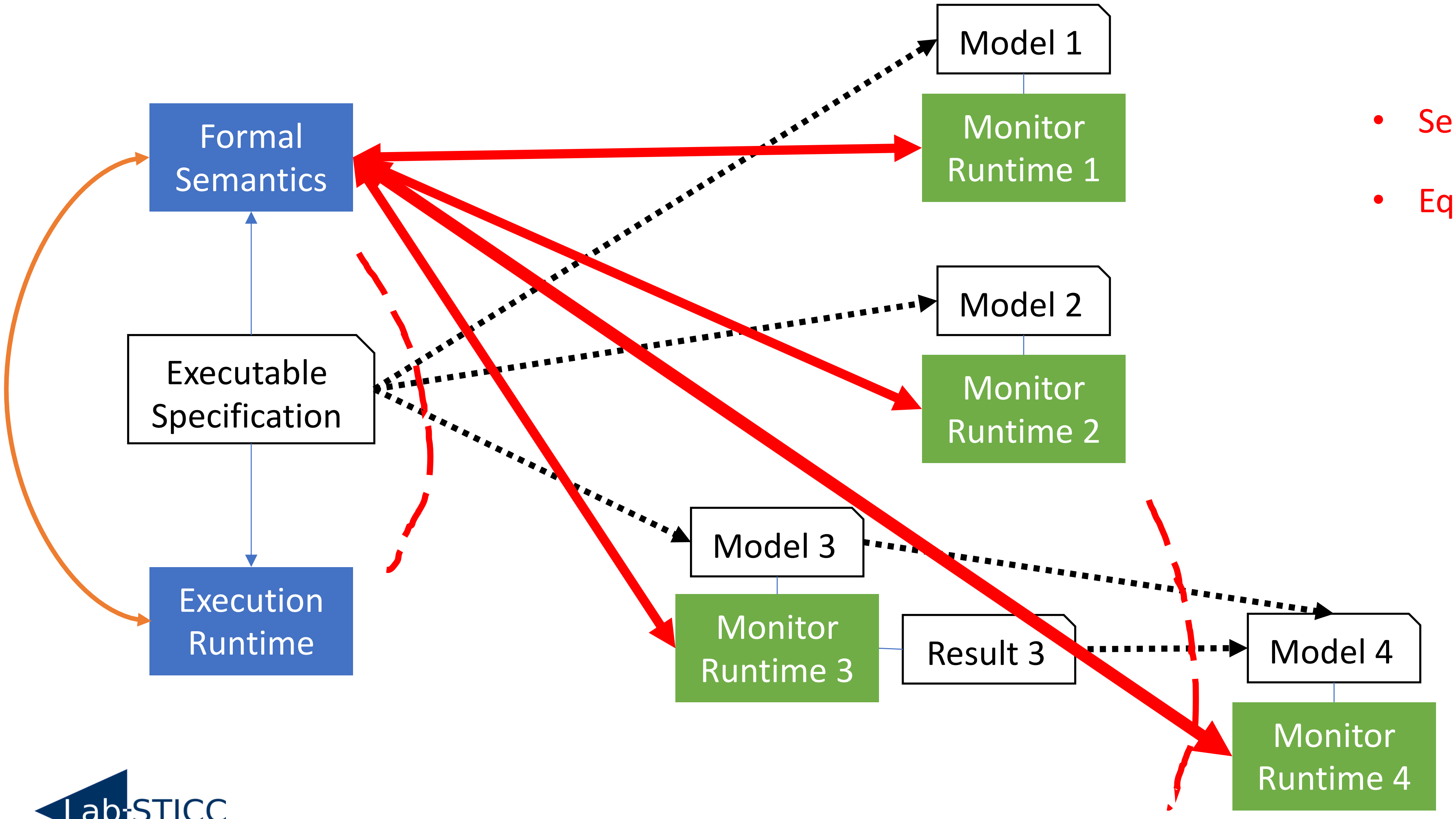
How to bridge the gap between
the **specification languages**
and the **language monitors**
running on ever more heterogeneous **platforms**?

ENSTA Bretagne
Lab-STICC

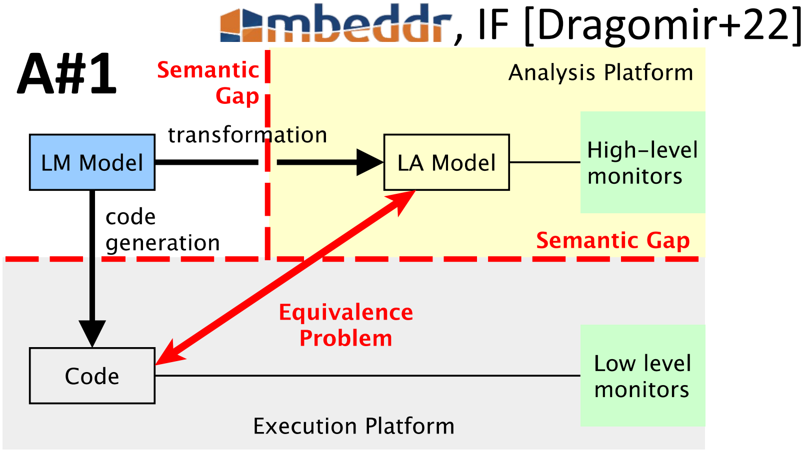
2. Transformations: the Shy Semantics and the Inaccessible Monitors.

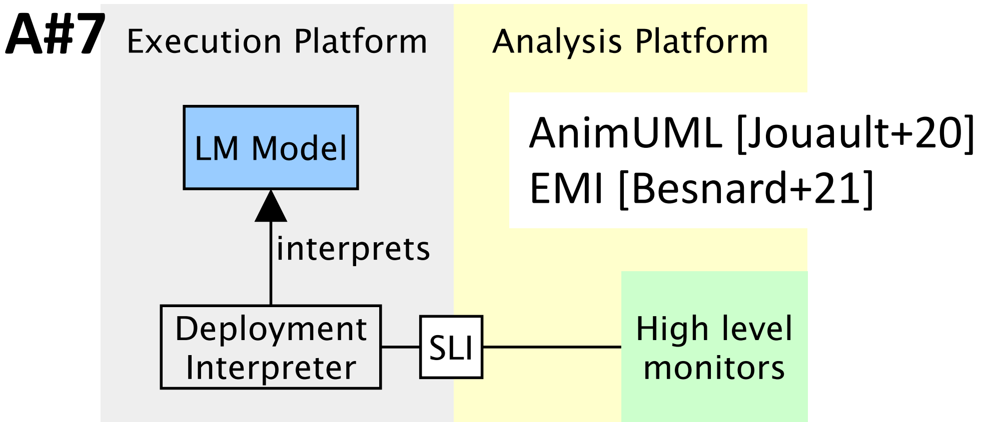
- Understanding the problem
- Looking for high-level solutions

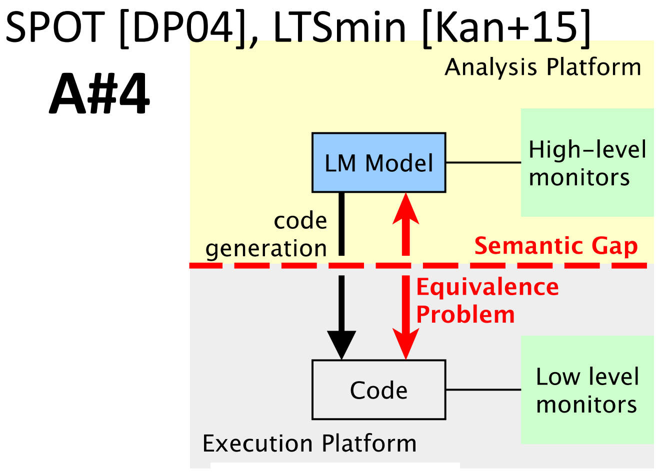
Many Semantics – Many Runtime Monitors

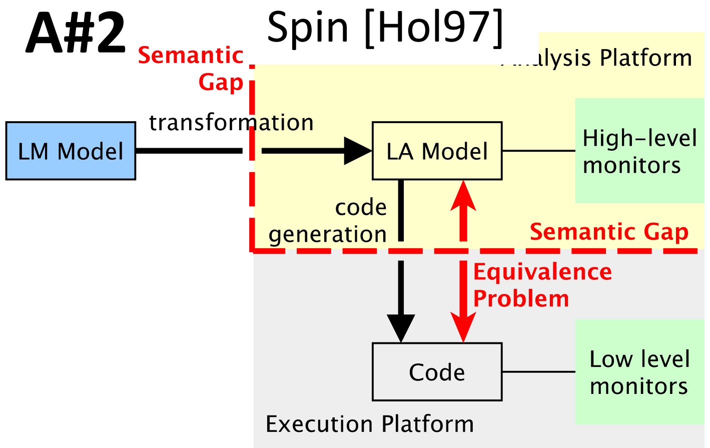


- Semantic gaps
- Equivalence problems





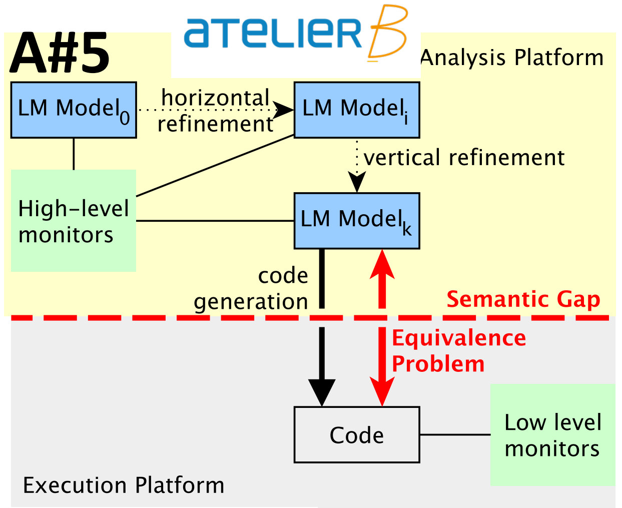


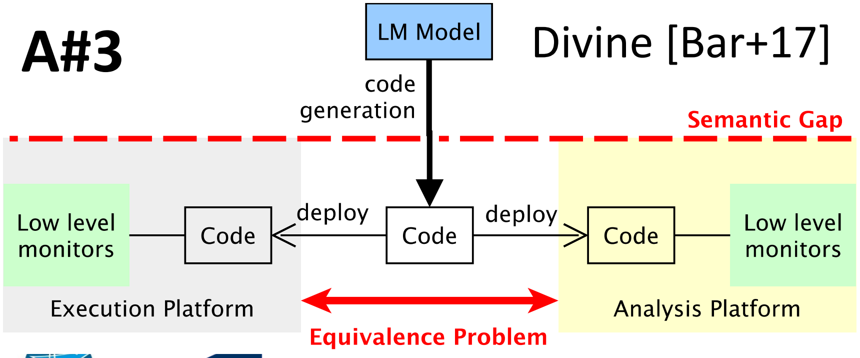


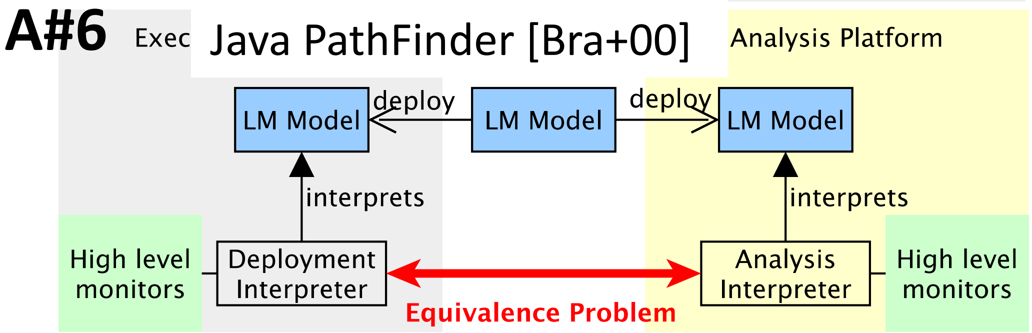
P#1 Semantic gap between design model and analysis model

P#2 Semantic gap between design model and executable code

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



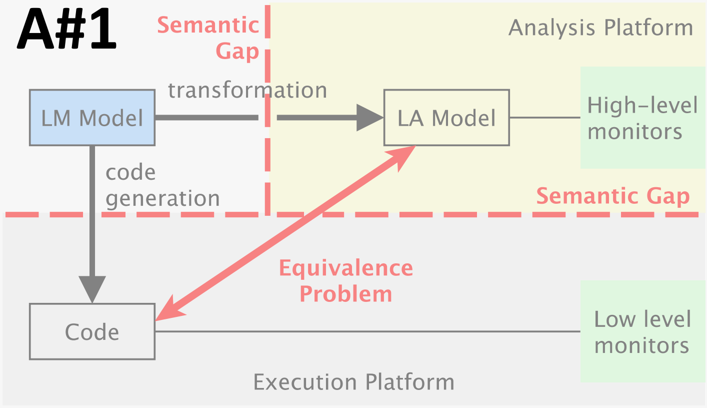




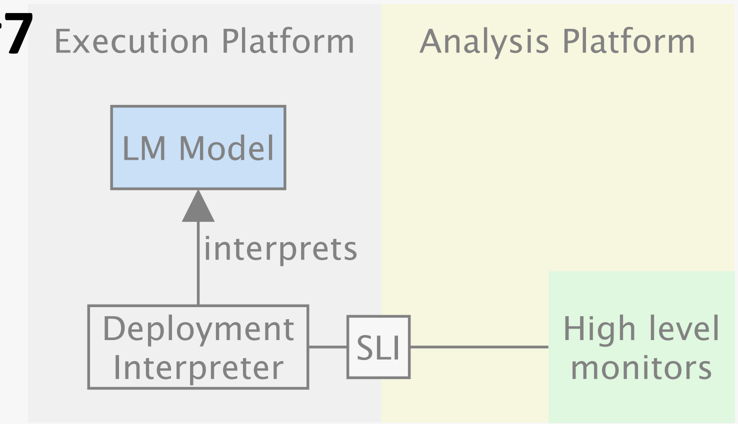
ENSTA Bretagne Lab-STICC

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.

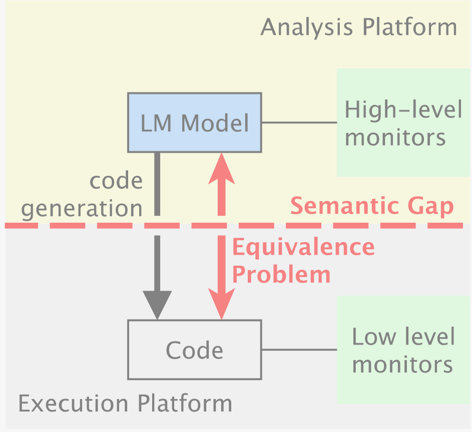
A#1



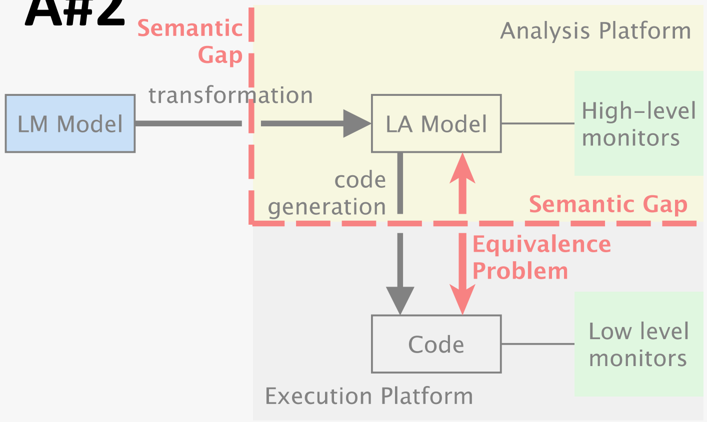
A#7



A#4

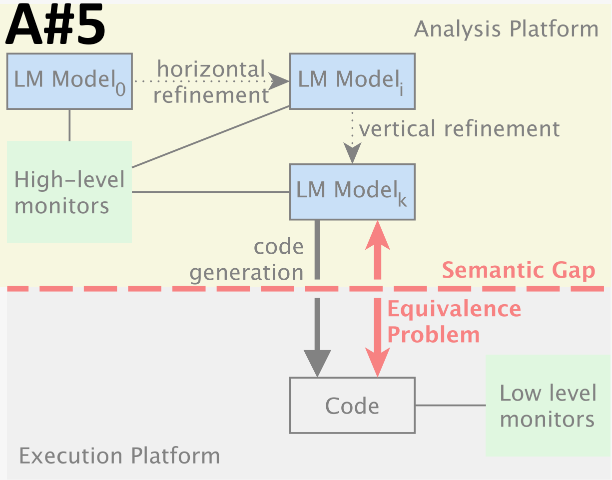


A#2

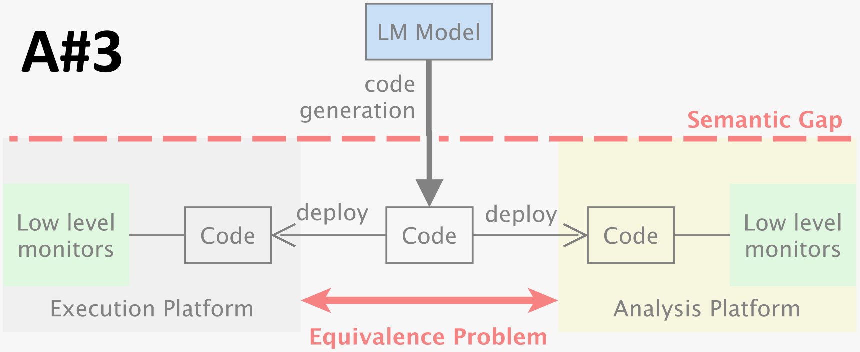


	A#1	A#2	A#3	A#4	A#5	A#6	A#7
P#1	✗	✗	✓	✓	✓	✓	✓
P#2	✗	✗	✗	✗	✗	✓	✓
P#3	✗	✗	✗	✗	✗	✗	✓

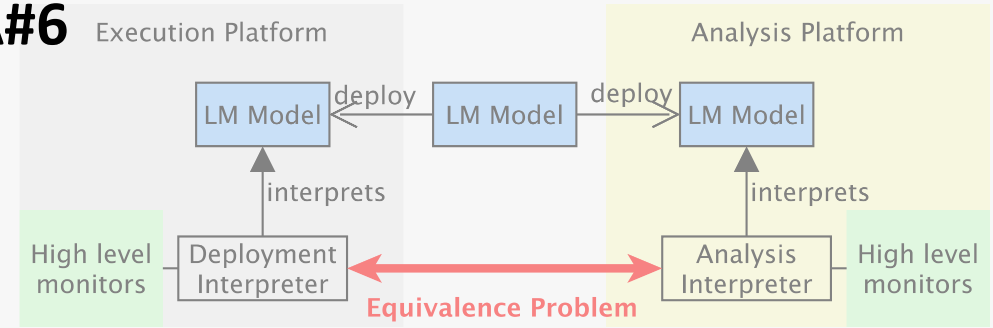
A#5



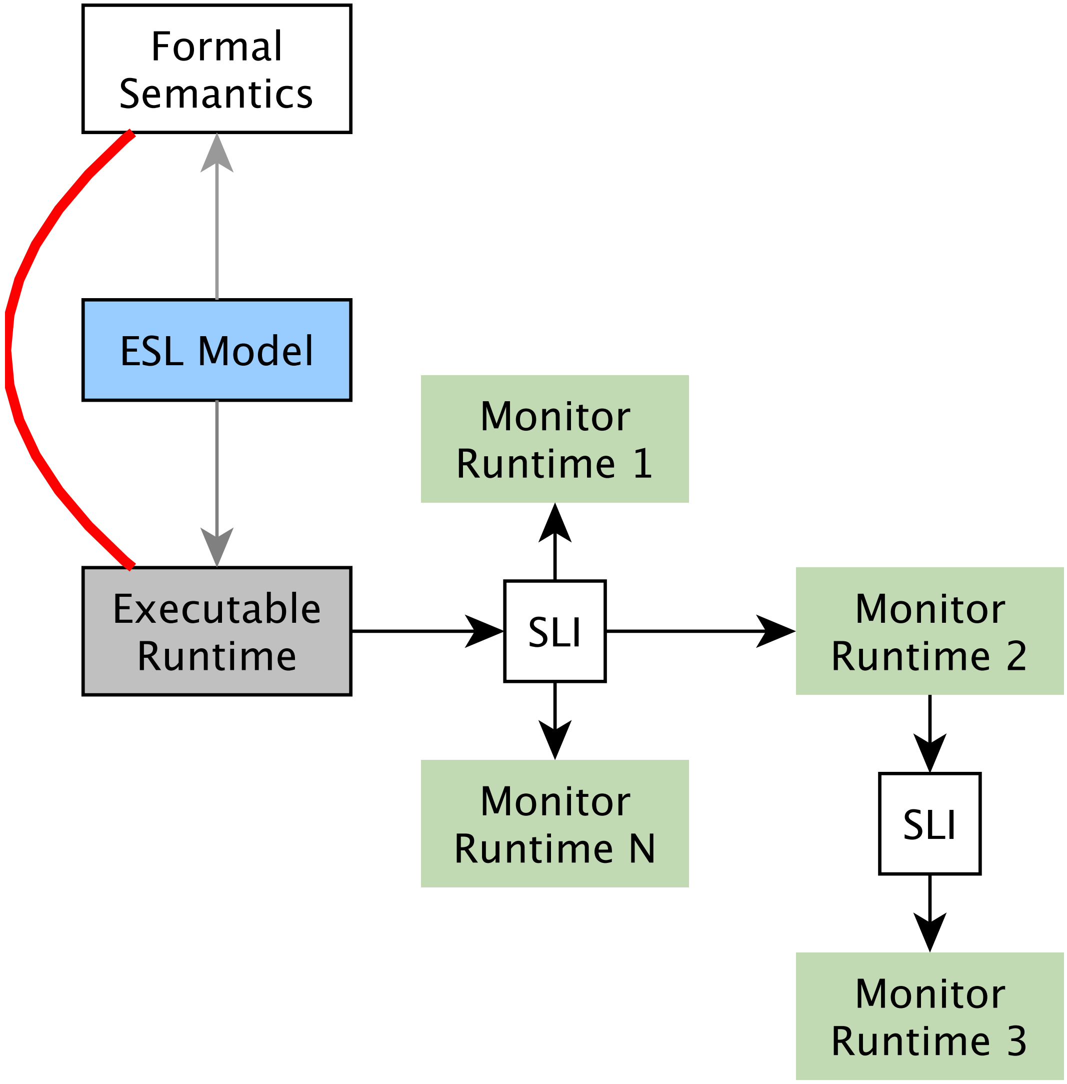
A#3

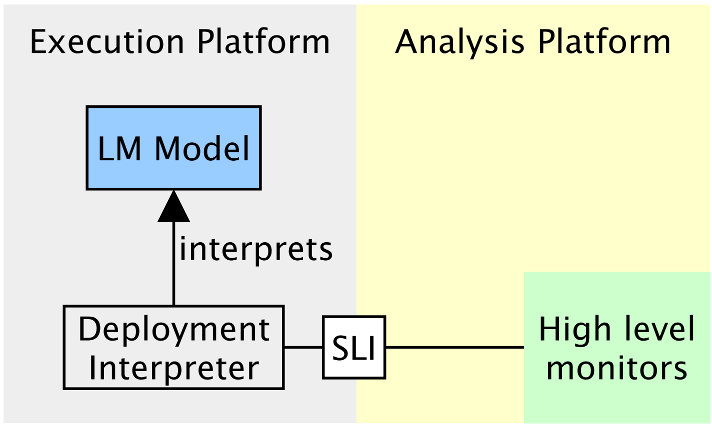


A#6

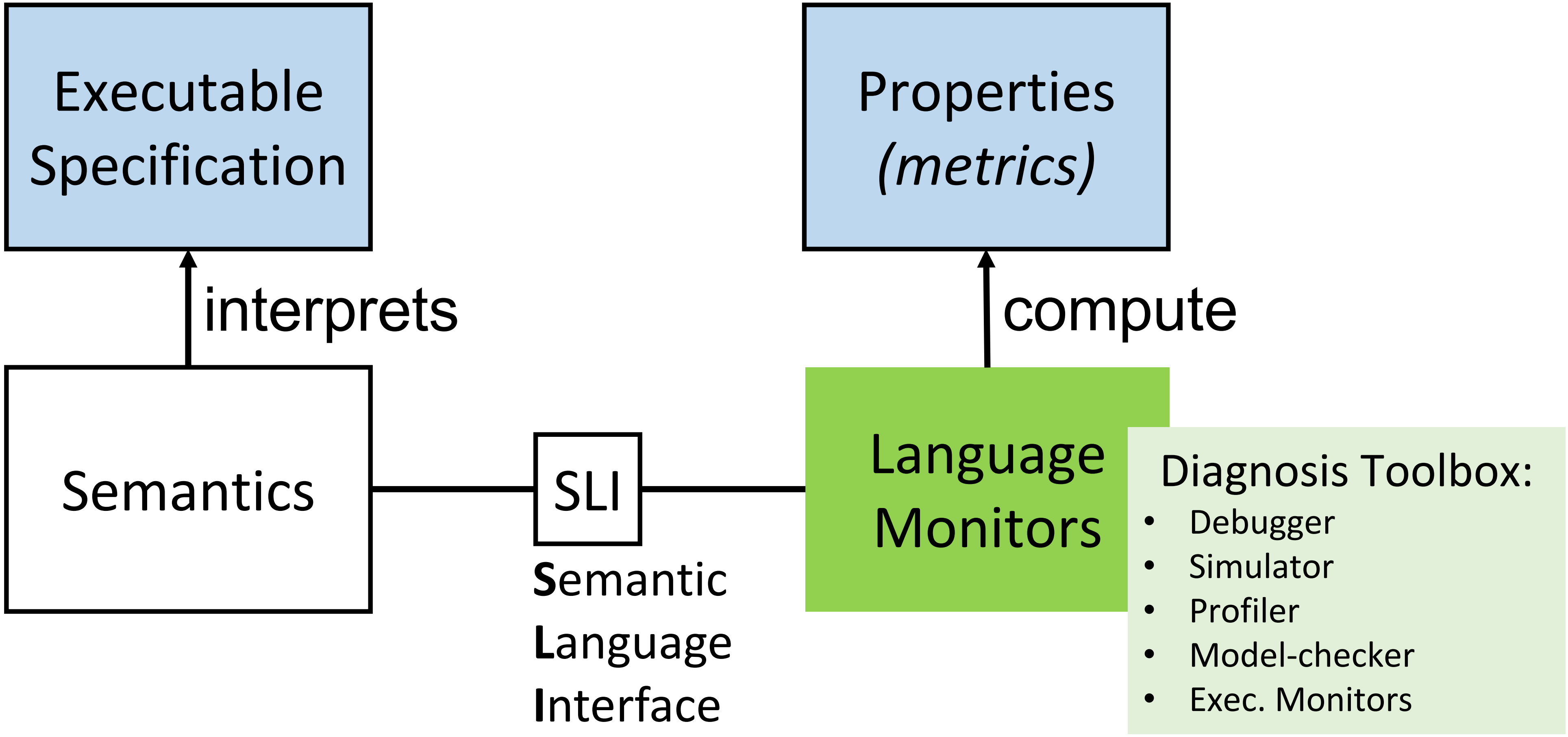


One Semantics – Many Language Monitors





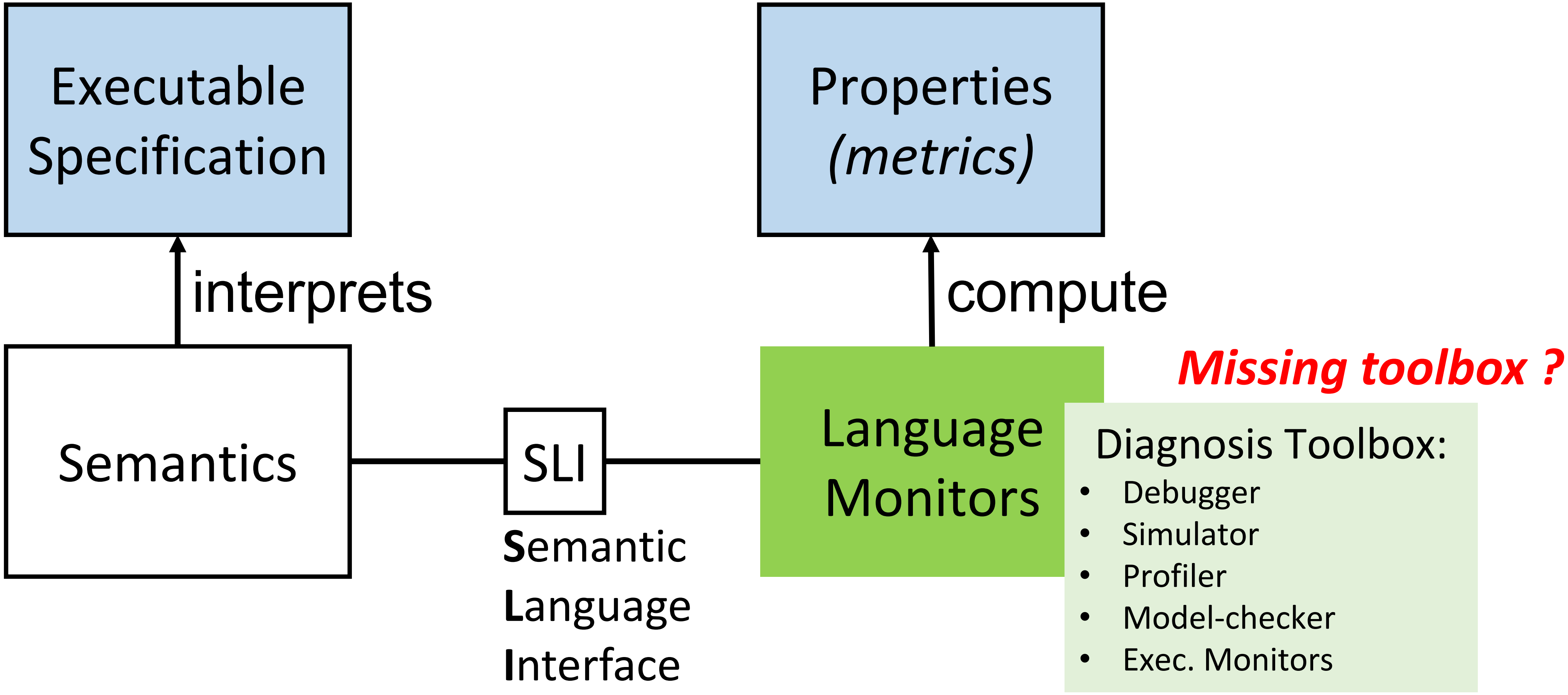
Make it Simple & Modular



Make it Simple & Modular

Q1: What is the SLI interface?

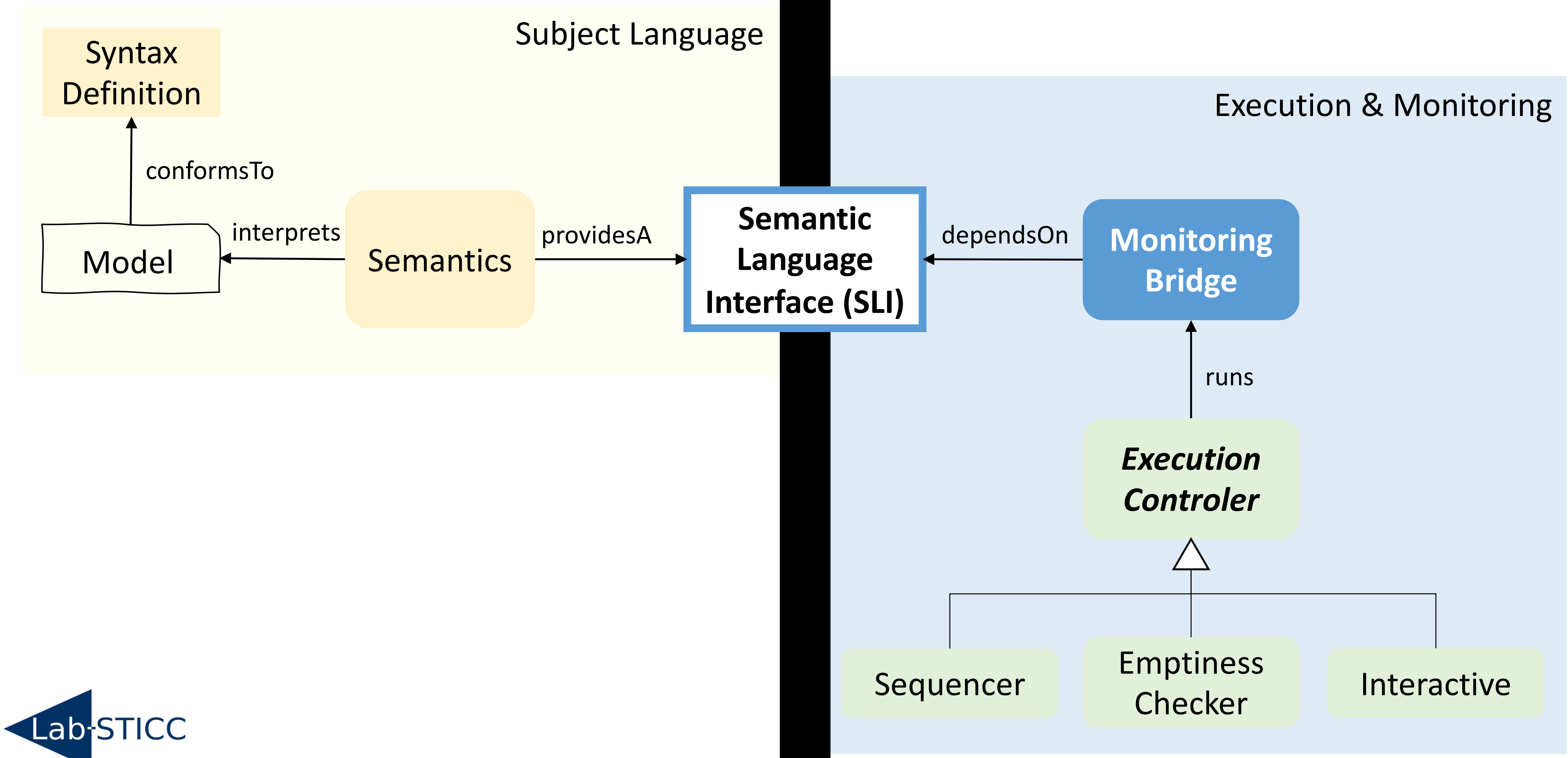
Q2: Where is the toolbox?



3. When the **Semantics** Decides to **Open up** the **Monitors** are **Interested**.

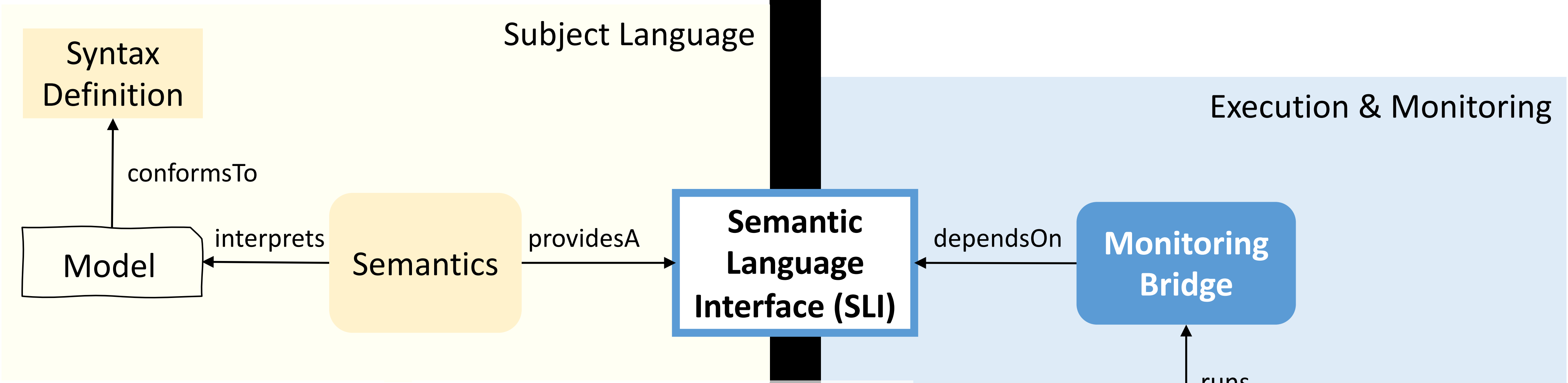
- Requirements
- G∀min∃ Semantic Language Interface
- An illustration

Ingredients:



ENSTA Bretagne Lab-STICC

Requirements:

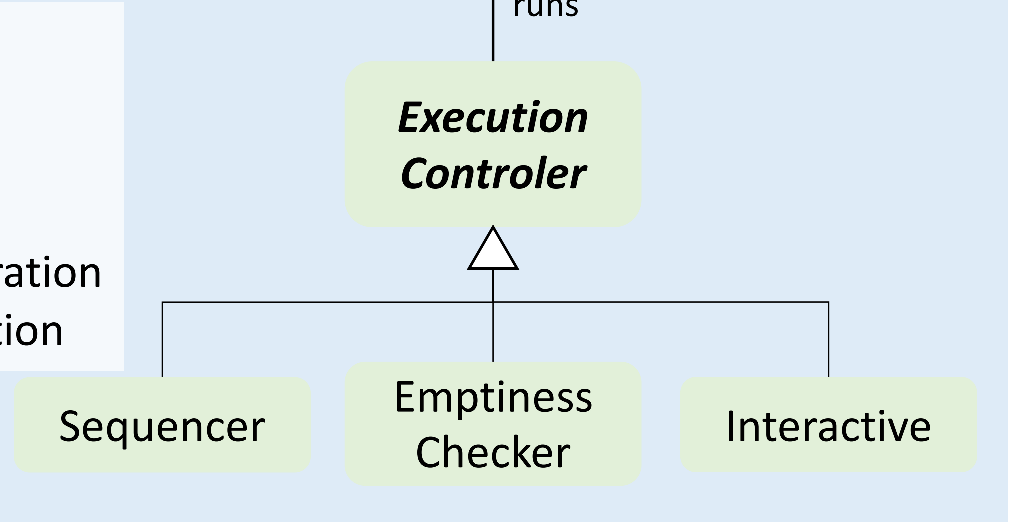


[R01] Completeness
[R02] Non-Interference

[R03] Genericity
[R04] Composability
[R05] Unanticipated Monitoring

[R06] Minimize the Gap
[R07] Break the Rules

[R08] Portability
[R09] Ease Semantics Integration
[R10] Ease Monitor Integration



ENSTA Bretagne Lab-STICC

G∀min∃ Semantic Language Interface (SLI)

```
SLI (C A E V R α) {
    semantics: (C A) {
        initial:            set C
        actions:        C → set A
        execute: A → C → set C
    }

                    execution step
                  ⏞
    evaluate: E → (C → A → C) → V -- questions

    reduce: R → C → α              -- reductions

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

Generic Types:
Configuration,
Action,
Expression,
Value,
Reduction Exp.
α: Reduced Config.

SLI for Lambda Calculus

Syntax:

```
E ≜ x          //variable
  | E₁ E₂      //application
  | λ x. E     //abstraction
```

CEK-style Semantics [ABM'14]:

lookup	≜⟨ x, ρ ,	κ⟩	⟶⟨	ρ[x].1,	ρ[x].2	,	κ⟩
app	≜⟨ e_1 e_2, ρ ,	κ⟩	⟶⟨	e_1,	ρ	,⟨○ e_2 ρ ⟩::κ⟩	
arg	≜⟨ v, ρ_1,	⟨○ e ρ_2⟩::κ⟩	⟶⟨	e ,	ρ_2	,⟨v ○ ρ_1⟩::κ⟩	
body	≜⟨ v, ρ_1,	⟨λx.e ○ ρ_2⟩::κ⟩	⟶⟨	e ,	ρ_2[x↦⟨v,ρ_1⟩],		κ⟩

SLI Semantics Definition

```
rules: { lookup, app, arg, body }
semantics: (C A) {
    initial: set C := {⟨exp, ∅, []⟩}

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

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

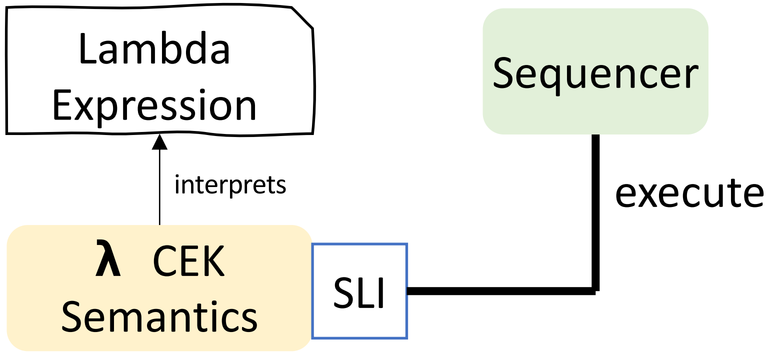
Domains:

```
Value ≜ λx.e                    Closure ≜ ⟨v,ρ⟩
ρ ≜ {variable ↦ closure}// Environment
Frame ≜ ⟨c ○⟩ | ⟨○ e ρ⟩

C ≜ ⟨E, ρ, [Frame]⟩            //Configuration
A ≜ ⟨ ⟩⟶⟨ ⟩                   //Action = rule
```

[ABM'14] B. Accattoli, P. Barenbaum, and D. Mazza. *Distilling Abstract Machines. ICFP '14*

ENSTA Bretagne   Lab-STICC




```
Sequencer(sli) {
    current = sli.initial.any
    while (current != ∅) {
        action = sli.actions(current).any
        current= sli.execute(action,current).any
    }
}
```

where:

- `sli` is **deterministic** **<=> ∀ a c, |*initial*| = |*actions* c| = |*execute* a c| = 1**

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

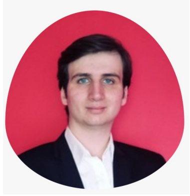
- **Some experiences unravel reusable monitoring bridges**
- Transfer to commercial products -- OBP2 inside
- Exploring hardware execution
- Multiverse debugging made simple and more powerful
- From zero to model-checker in 30 Hours

OBP2 Research Vehicle

2015-2023

Emilien FOURNIER
2022

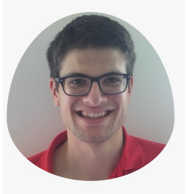
Nicolas SUN
2022



Matthias PASQUIER
in progress

Luka LE ROUX
2018

Vincent LEILDE
2019



Valentin BESNARD
2020

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,ERDF)
JoinSafeCyber	(AID)
VeriMoB	(RAPID)
EASE4SE	(RAPID)
DEPARTS	(PIA)
GeMoC	(ANR)

Commercial Products [*PragmaDEV*]

sdl PRAGMADEV STUDIO

BPMN PRAGMADEV PROCESS

Academic Prototypes [*in-house*]

AnimUML

UNIFIED MODELING LANGUAGE UML

EMI-UML

AEFD

Lucio

eseo GRANDE ÉCOLE D'INGÉNIEURS

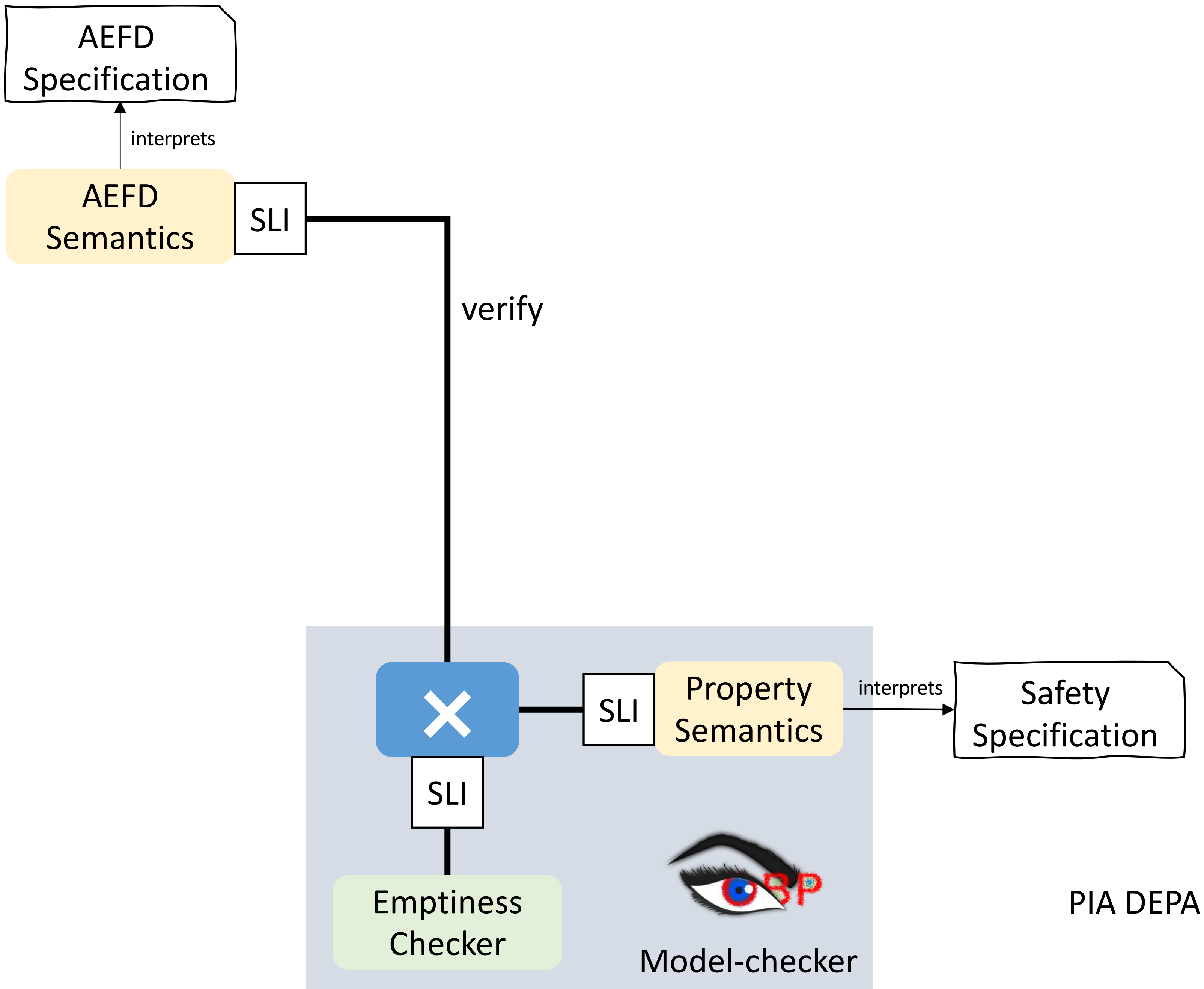
DAVIDSON CONSULTING

SNCF

Reuse [OTS]

TLA+

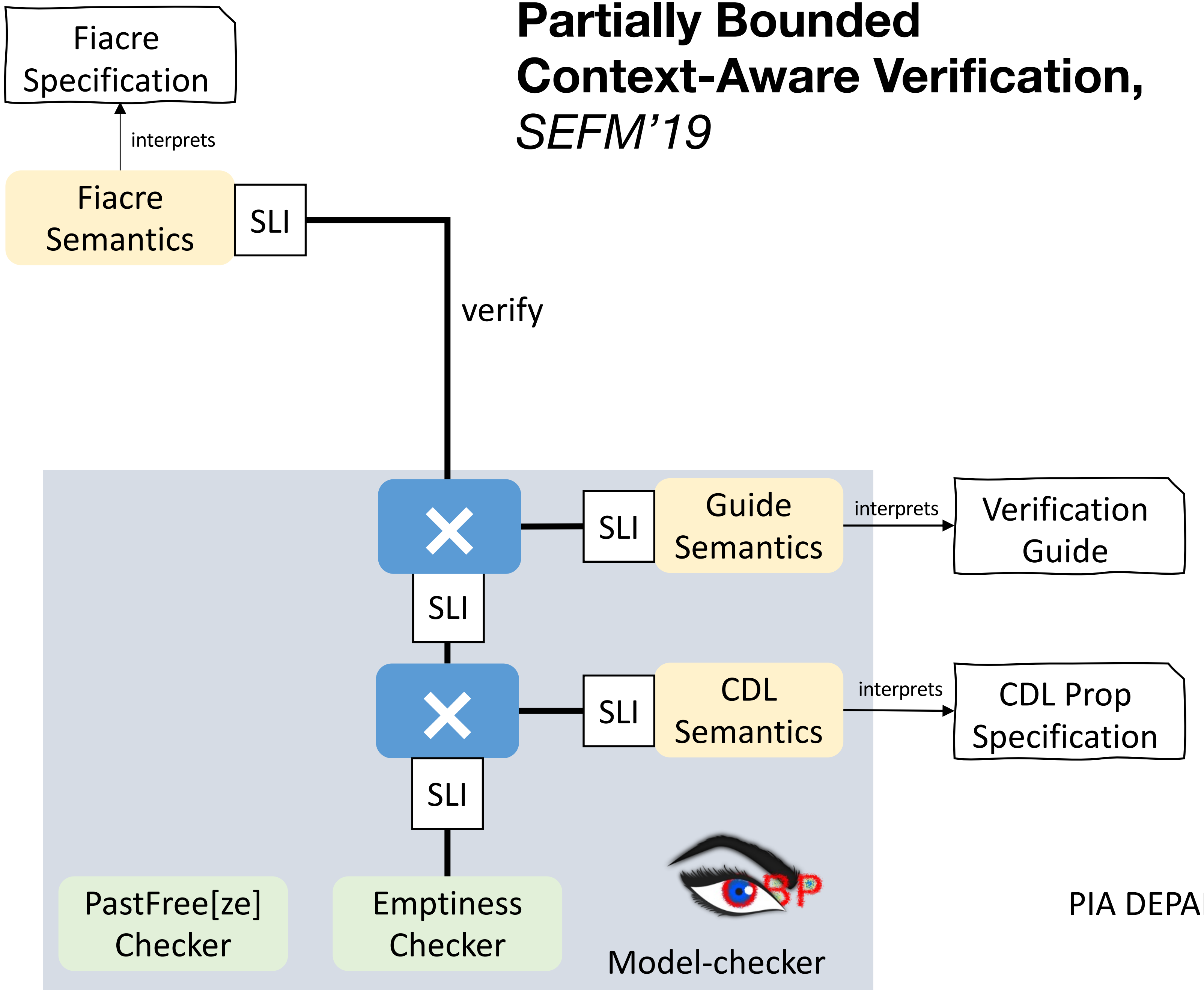
Fiacre



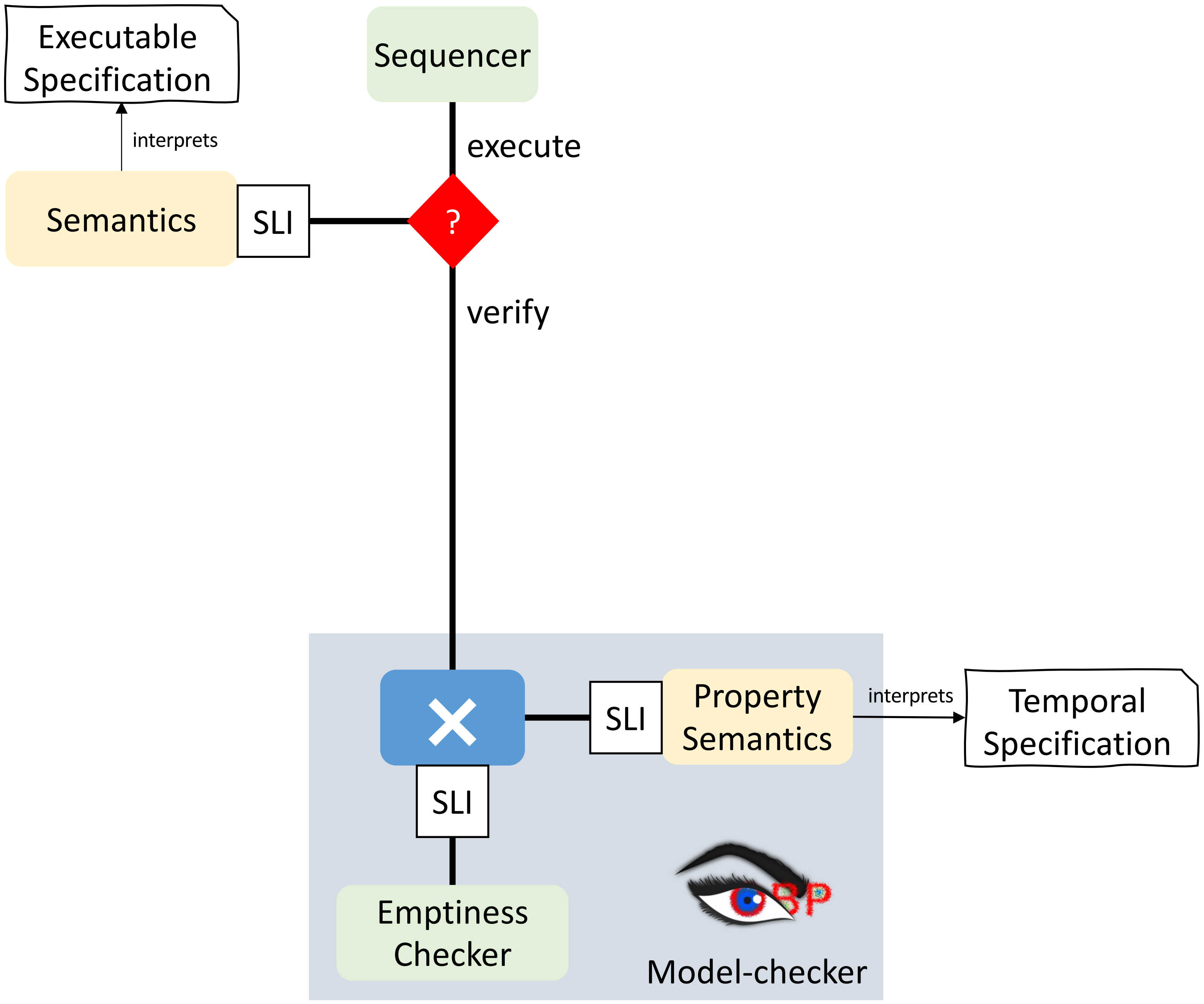

PhD Luka LE ROUX

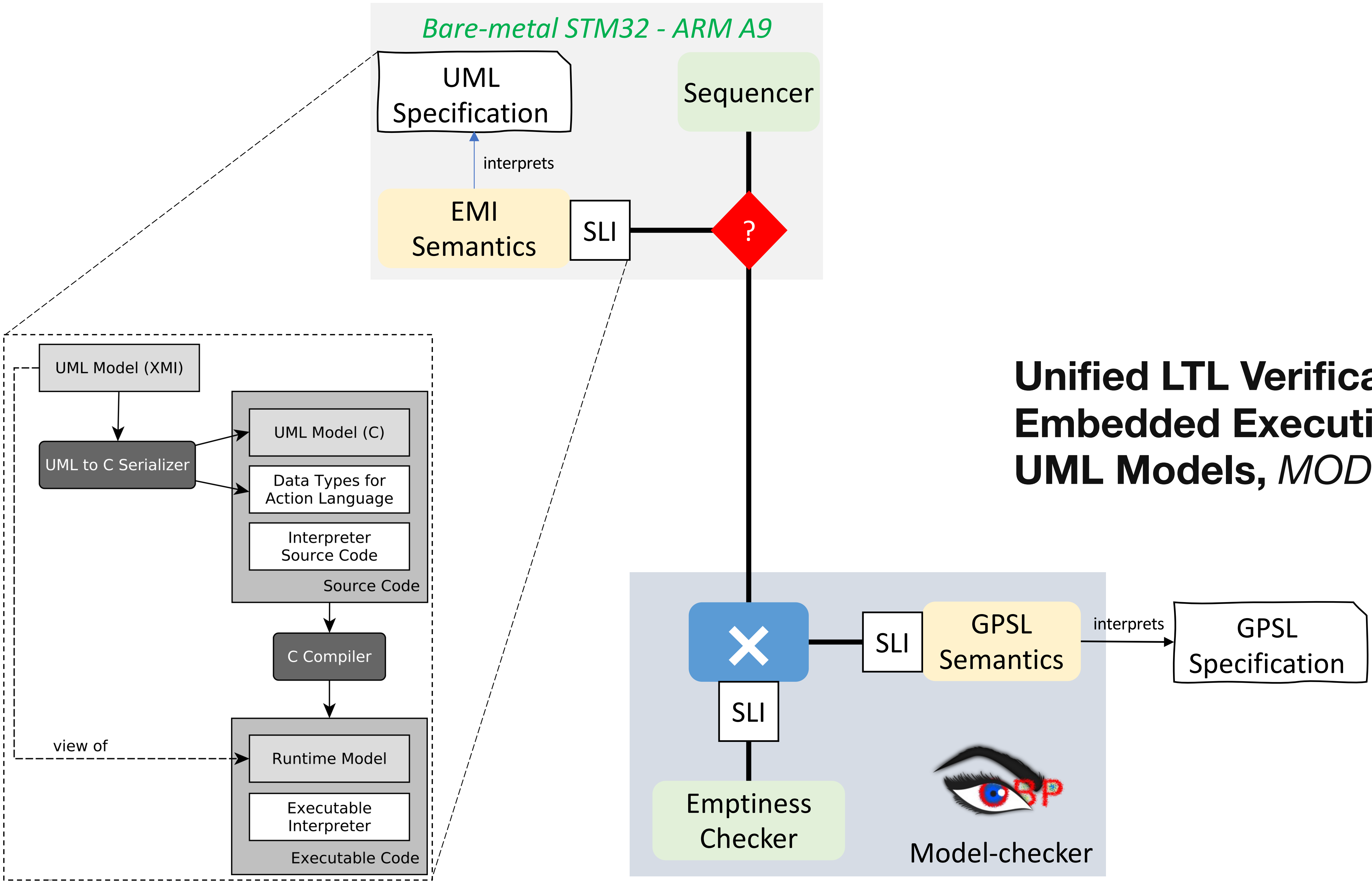
Partially Bounded Context-Aware Verification,
SEFM'19

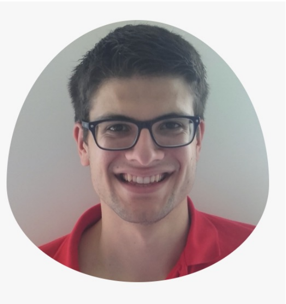
PhD Luka LE ROUX



PIA DEPARTS


Executable Specification
interprets
Semantics
SLI
Sequencer
execute
?
verify
SLI
Property Semantics
interprets
Temporal Specification
SLI
Emptiness Checker
Model-checker



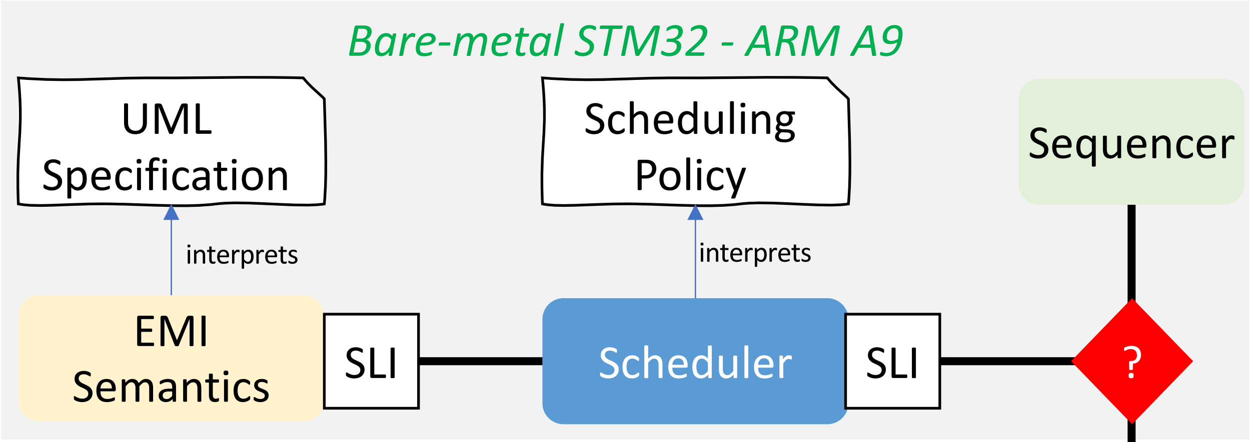


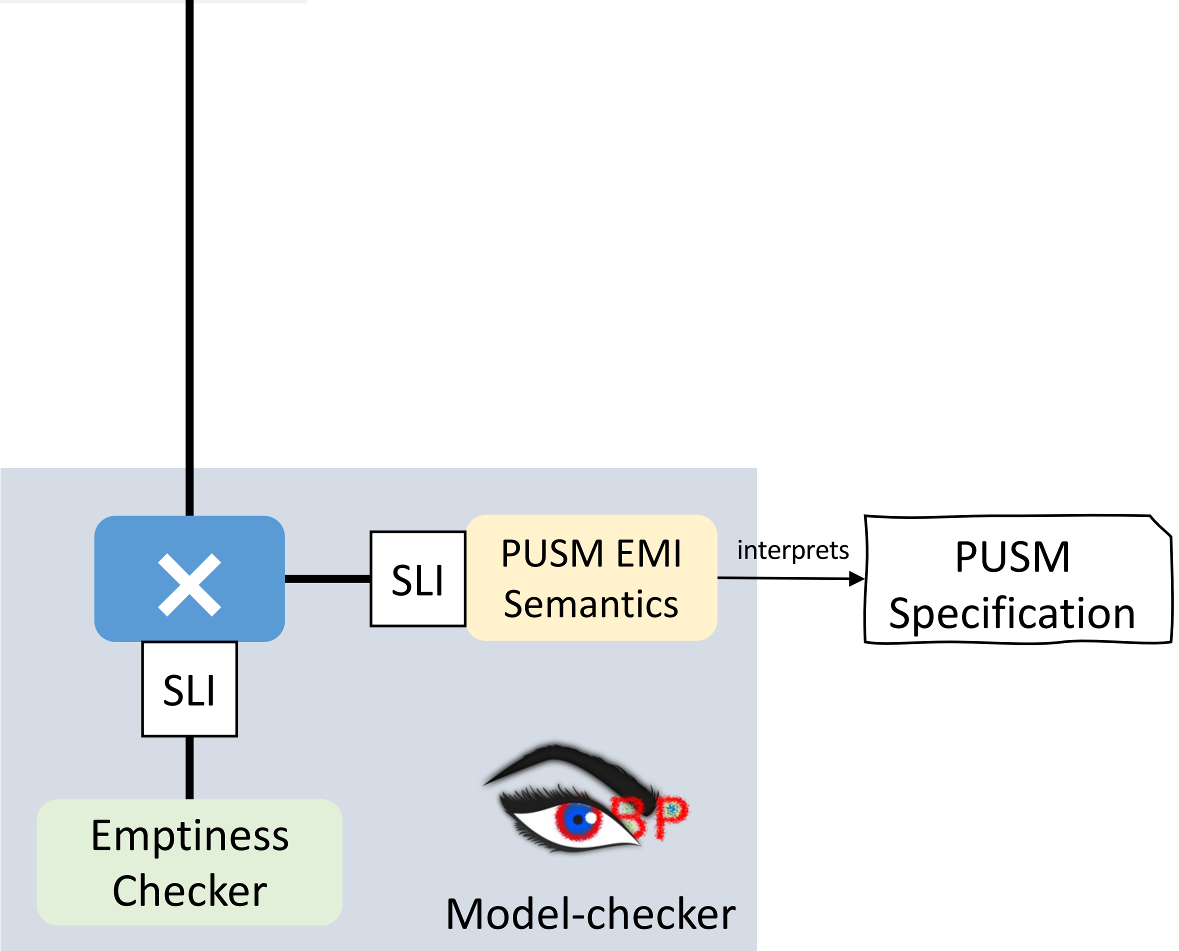
PhD Valentin BESNARD

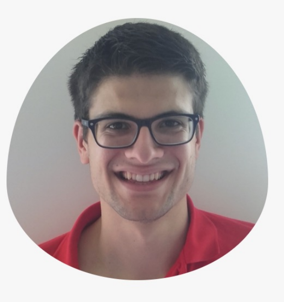
Unified LTL Verification and Embedded Execution of UML Models, *MODELS'18*

DAVIDSON CONSULTING

eseo GRANDE ÉCOLE D'INGÉNIEURS







PhD Valentin BESNARD

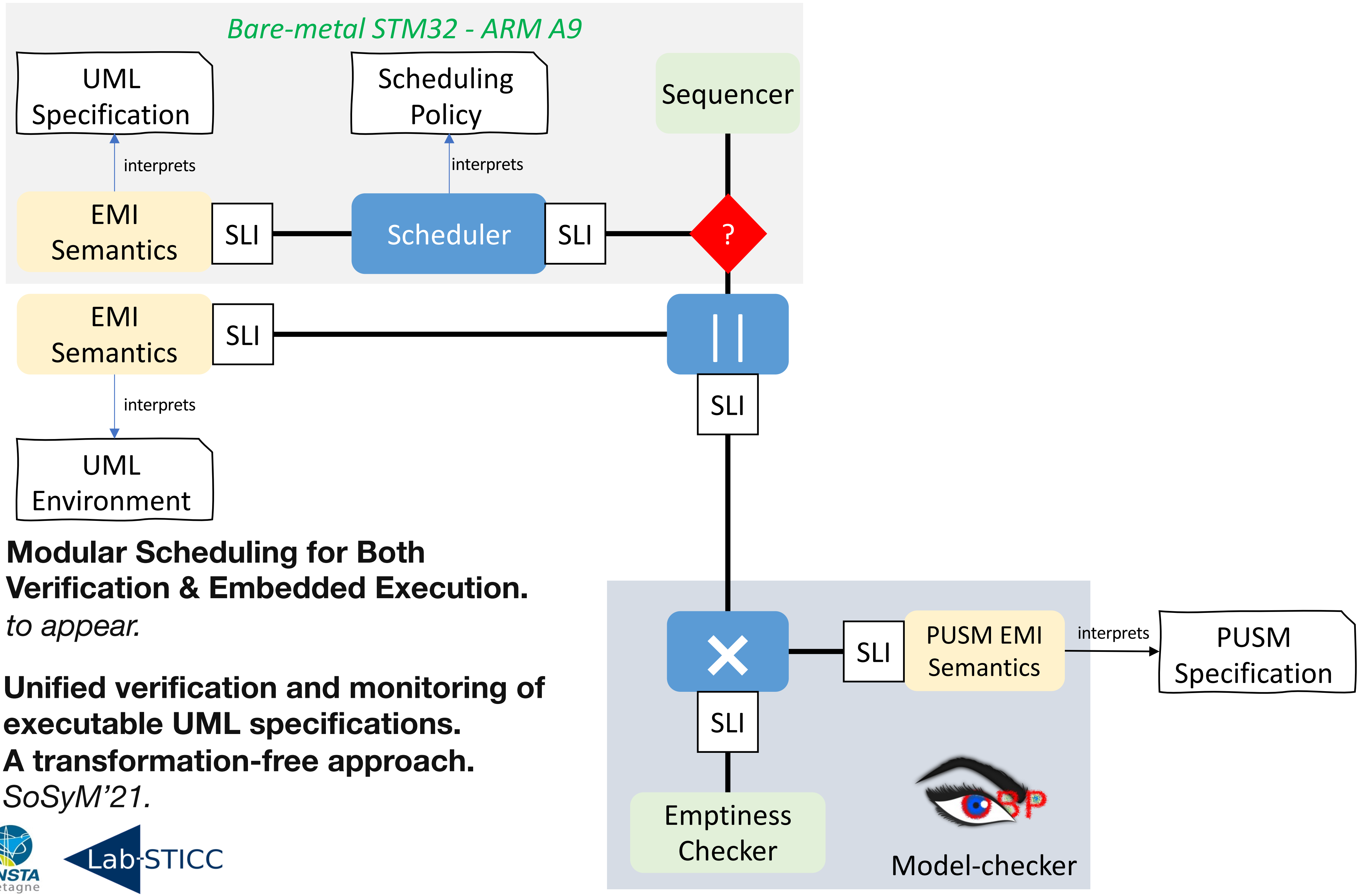
Modular Scheduling for Both Verification & Embedded Execution.
to appear.

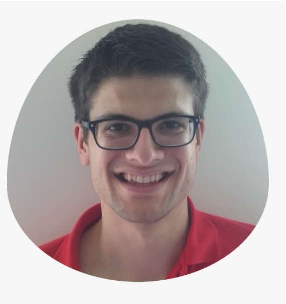
Unified verification and monitoring of executable UML specifications. A transformation-free approach.
SoSyM'21.

ENSTA Bretagne Lab-STICC

DAVIDSON CONSULTING

eseo GRANDE ÉCOLE D'INGÉNIEURS





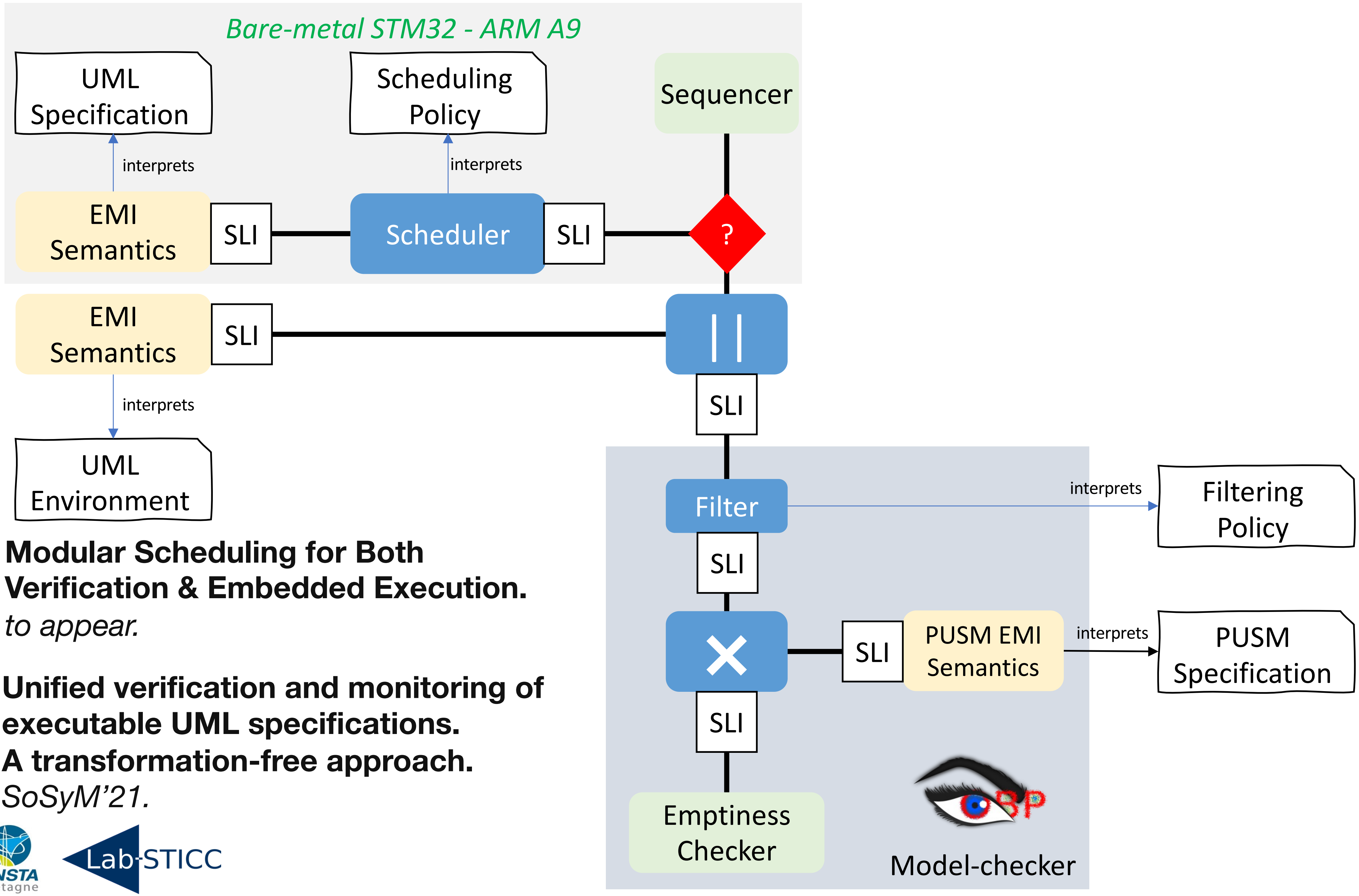
PhD Valentin BESNARD

Modular Scheduling for Both Verification & Embedded Execution.
to appear.

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

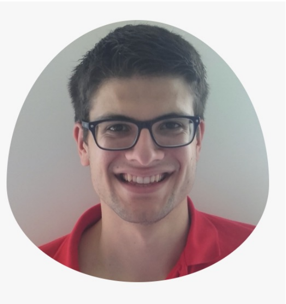
ENSTA Bretagne
Lab-STICC

DAVIDSON CONSULTING
eseo GRANDE ÉCOLE D'INGÉNIEURS

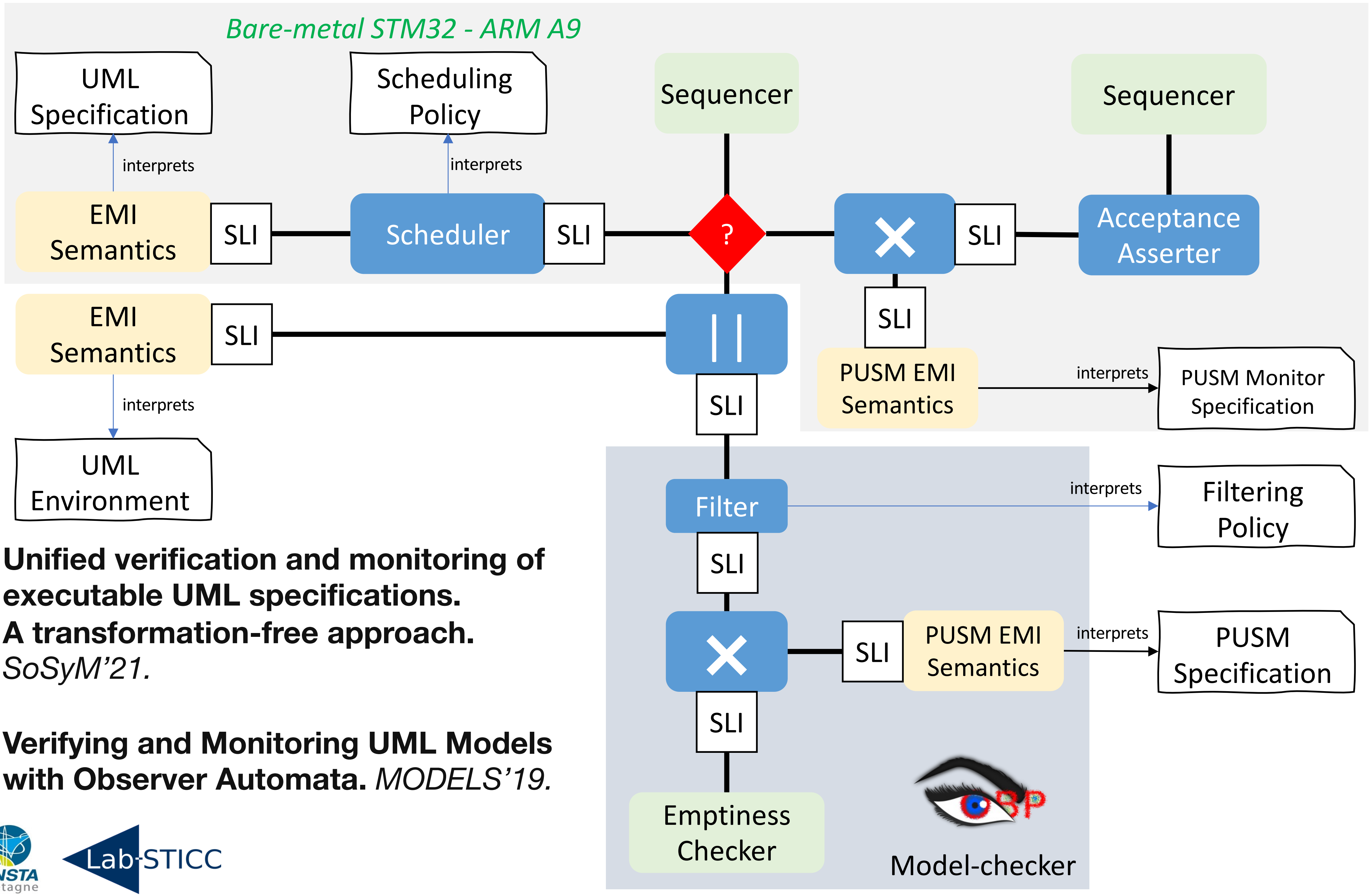


Modular Scheduling for Both Verification & Embedded Execution.
to appear.

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

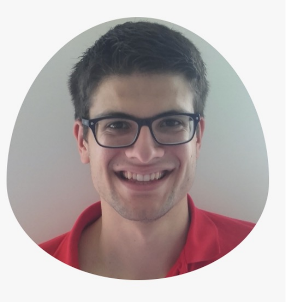


PhD Valentin BESNARD



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

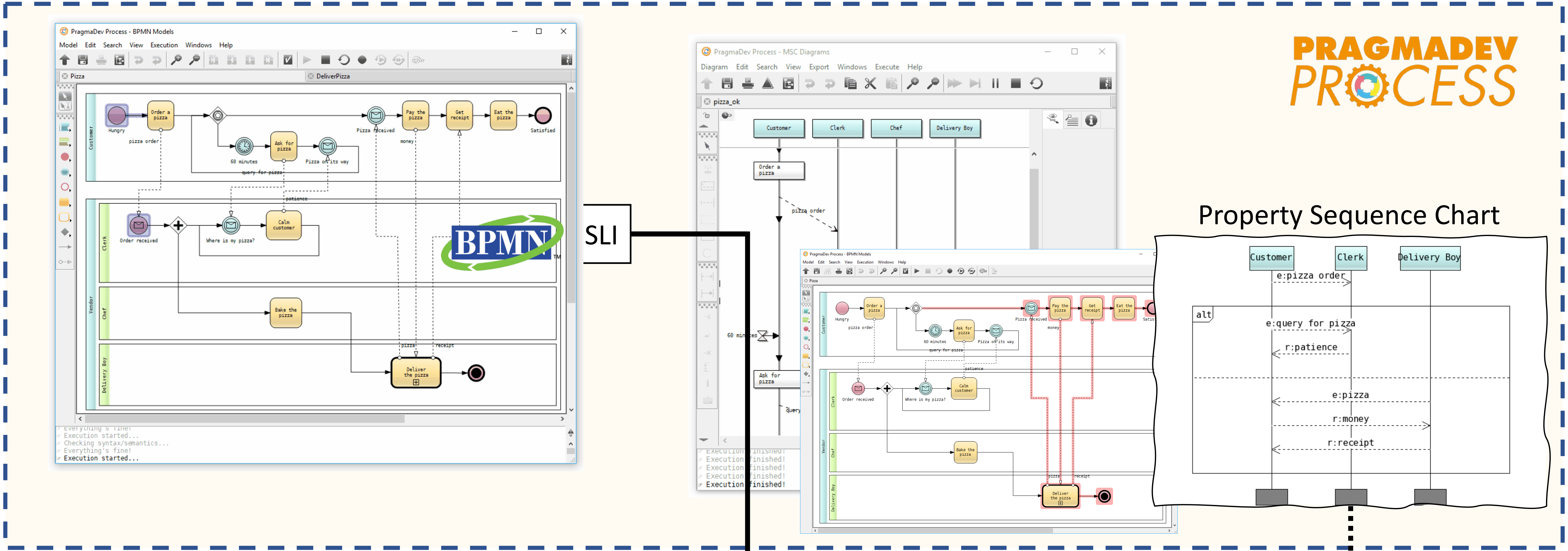
Verifying and Monitoring UML Models with Observer Automata. *MODELS'19.*

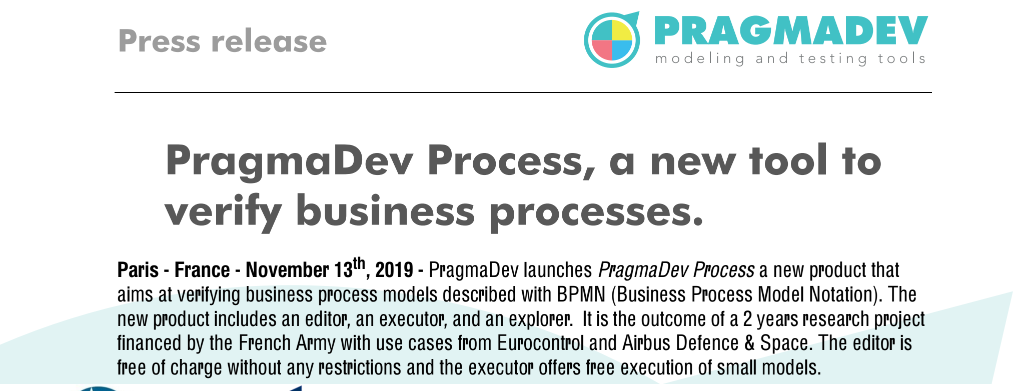


PhD Valentin BESNARD

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

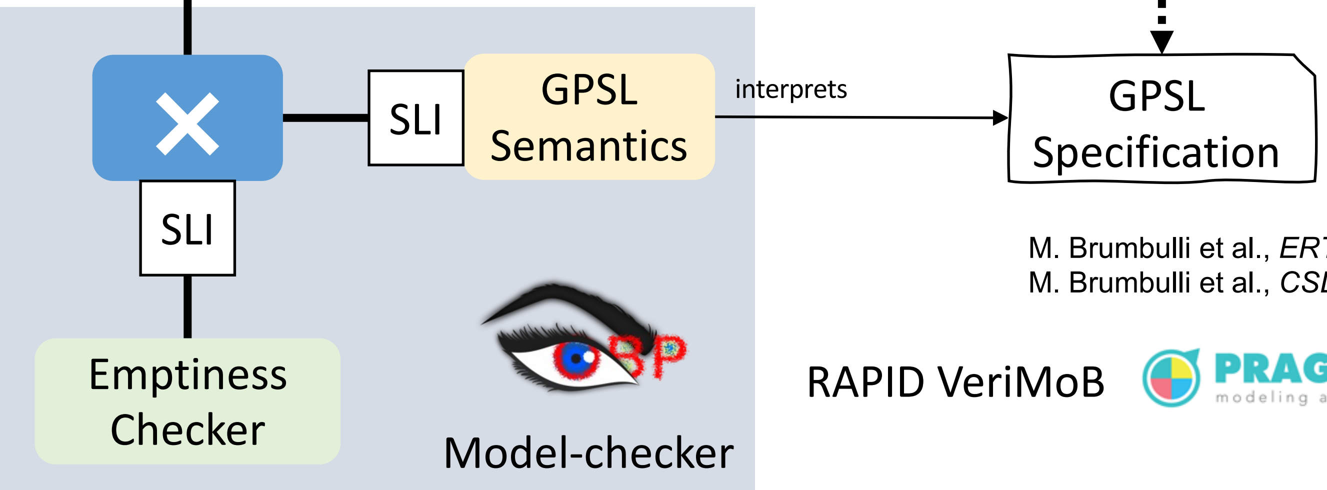
- Some experiences unravel reusable monitoring bridges
- **Transfer to commercial products -- OBP2 inside**
- Exploring hardware execution
- Multiverse debugging made simple and more powerful
- From zero to model-checker in 30 Hours




Press release

PragmaDev Process, a new tool to verify business processes.

Paris - France - November 13th, 2019 - PragmaDev launches *PragmaDev Process* a new product that aims at verifying business process models described with BPMN (Business Process Model Notation). The new product includes an editor, an executor, and an explorer. It is the outcome of a 2 years research project financed by the French Army with use cases from Eurocontrol and Airbus Defence & Space. The editor is free of charge without any restrictions and the executor offers free execution of small models.



M. Brumbulli et al., *ERTS 2020*
M. Brumbulli et al., *CSD&M 2020*

RAPID VeriMoB

Press release

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

Paris - France - June 14th, 2022 - *PragmaDev Studio* V6.0 introduces a new generation of model checker and the support of the new SDL broadcast, making it the best modeling tool to specify and design safe communicating software.

Following a long collaboration with ENSTA Bretagne research lab, PragmaDev has integrated in its latest release of PragmaDev Studio, ENSTA Bretagne model checker OBP (Observer Based Prover).

The primary objective of model checking is to explore all possible scenarios in the model. During the exploration it is possible to detect dead locks, unreachable model branches, or to verify properties. This is a major feature that leads to a safer and more secure design.

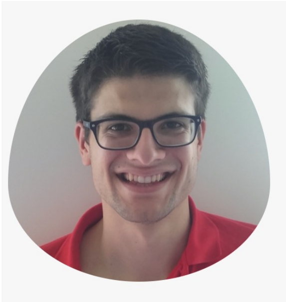
The key characteristic of OBP is that it does not rely on a dedicated language. It relies on a third party executor to execute the model. In PragmaDev Studio V6 OBP is interacting with PragmaDev SDL executor to execute the transitions. OBP does not actually know anything about the model it is exploring. It is the same principle with the properties. OBP evaluates complex properties made of atomic properties that are evaluated by the execution engine.

Communicating systems inherently create a lot of possible cases due to the fact that their designs are based on concurrent state machines. This creates a lot of possible paths of execution. PragmaDev Studio has built-in ways to reduce the state size during exploration:

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

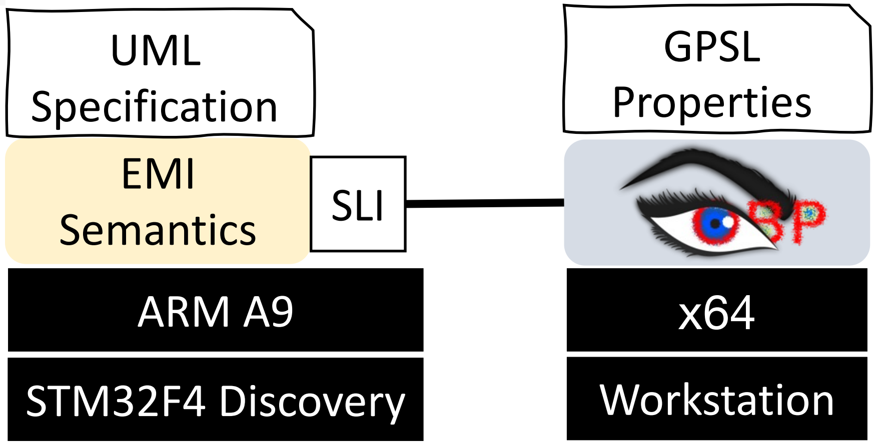
- Some experiences unravel reusable monitoring bridges
- Transfer to commercial products -- OBP2 inside
- **Exploring hardware execution**
- Multiverse debugging made simple and more powerful
- From zero to model-checker in 30 Hours

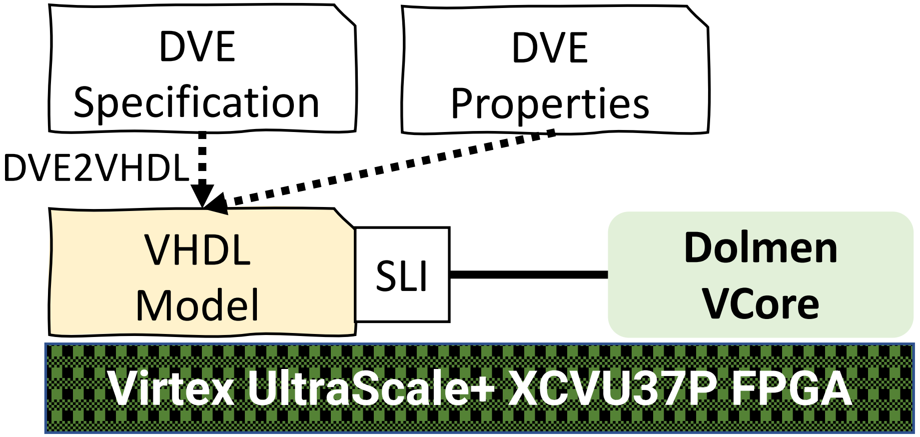
ENSTA Bretagne Lab-STICC

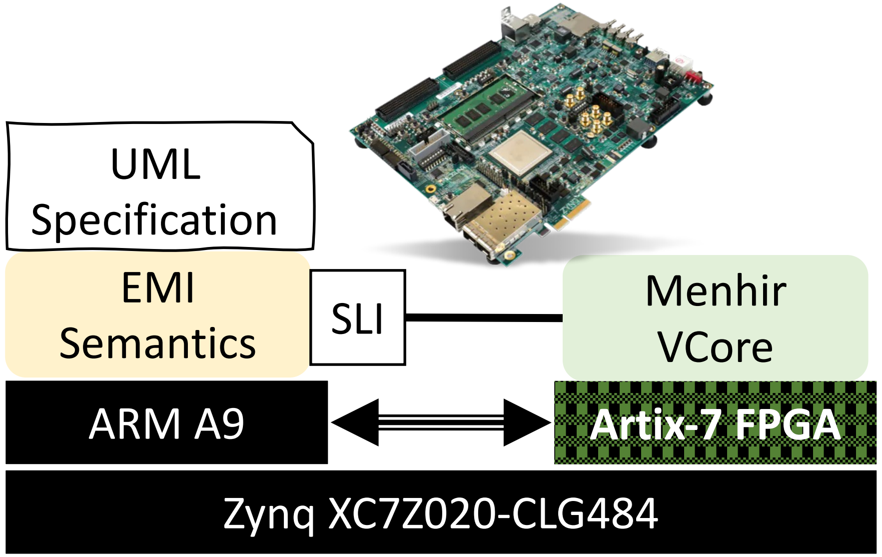


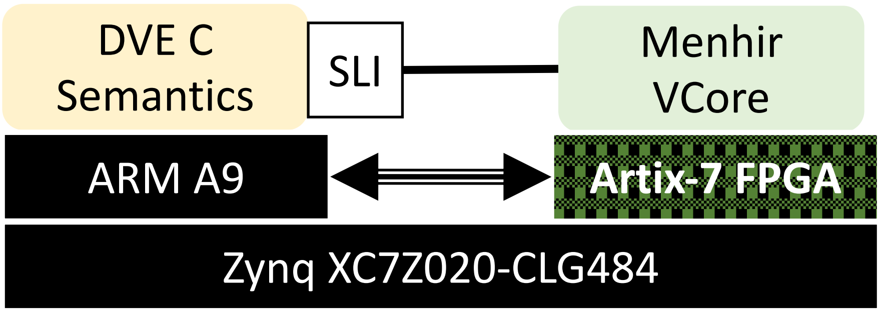
From Embedded to Hardware Execution

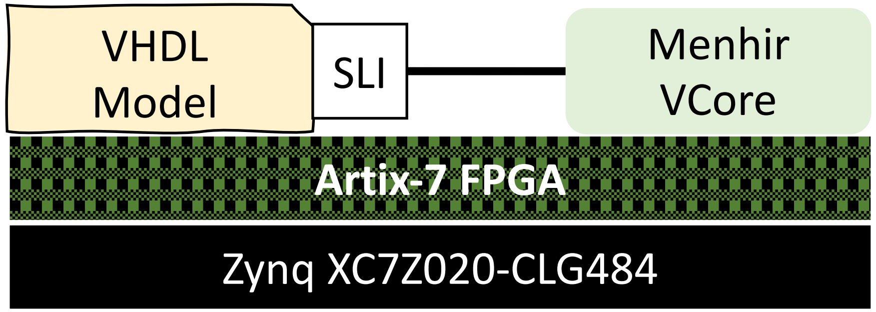
PhD Valentin BESNARD











PhD Emilien FOURNIER

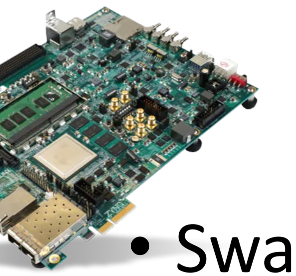
DSD'20
FPL'21
DATE'22

Région Bretagne
CPER CyberSSI

ENSTA Bretagne
Lab-STICC

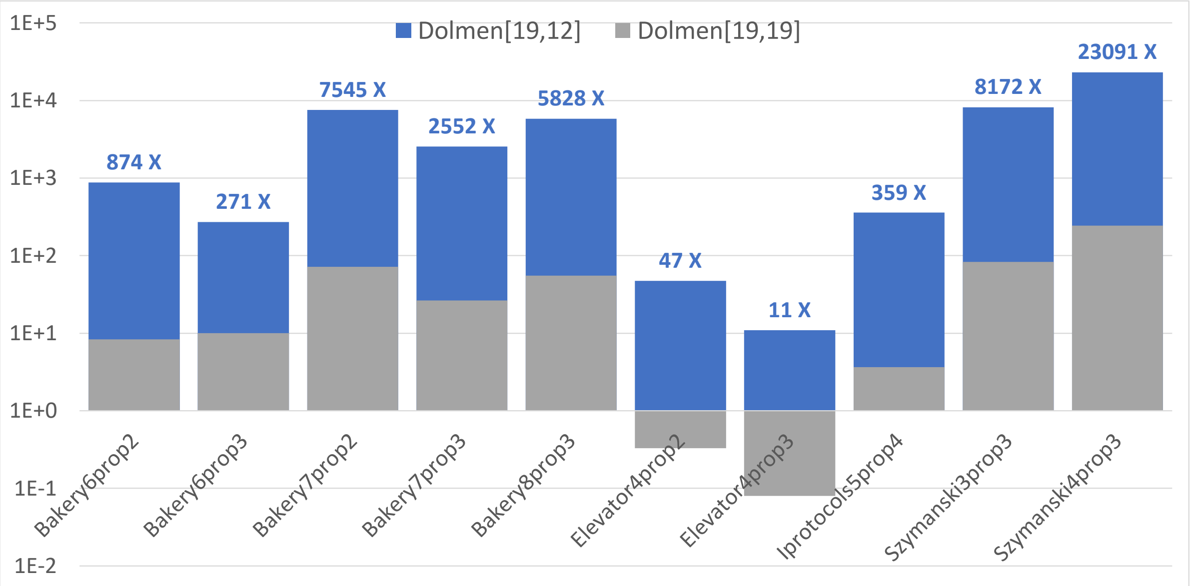
Dolmen: 1st Hardware Swarm Engine for Both Safety & Liveness Verification

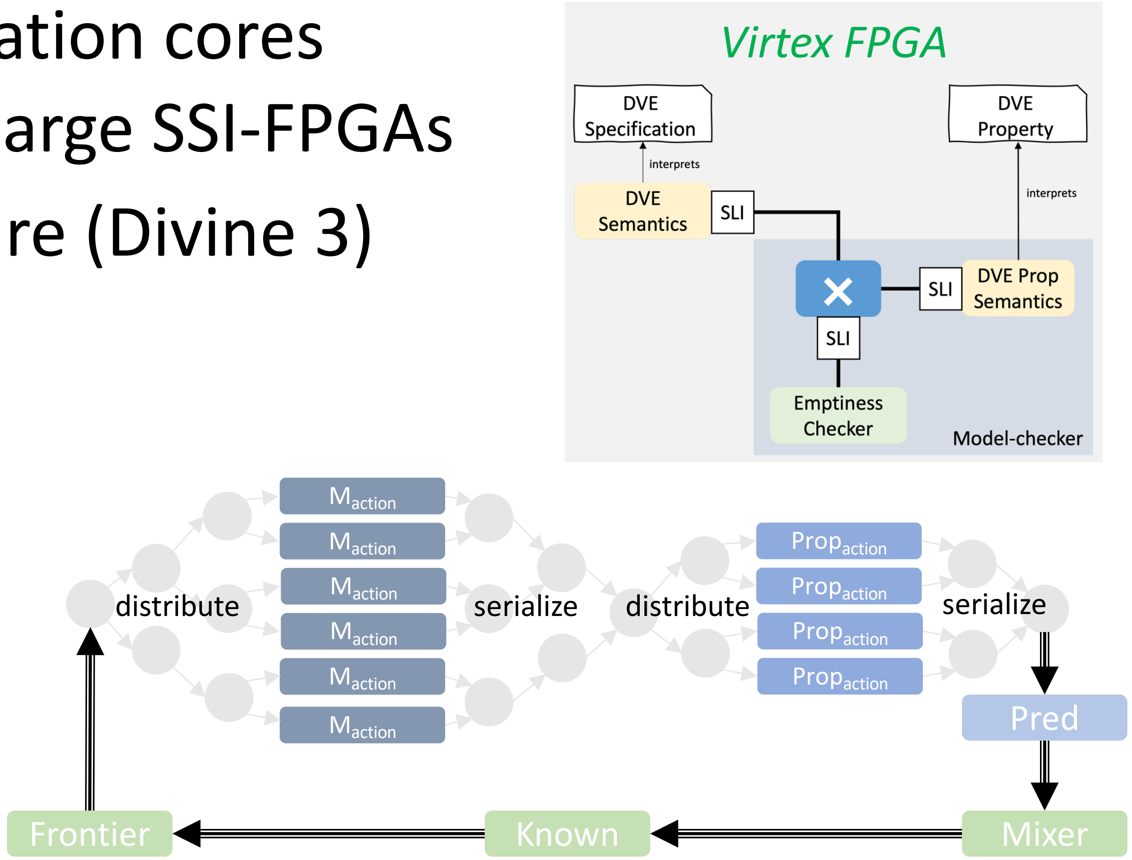
DATE'22



PhD Emilien FOURNIER

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

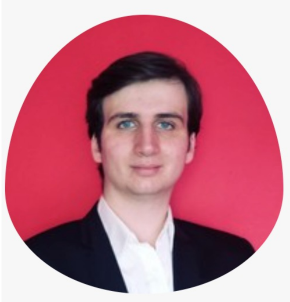




Région Bretagne
CPER CyberSSI

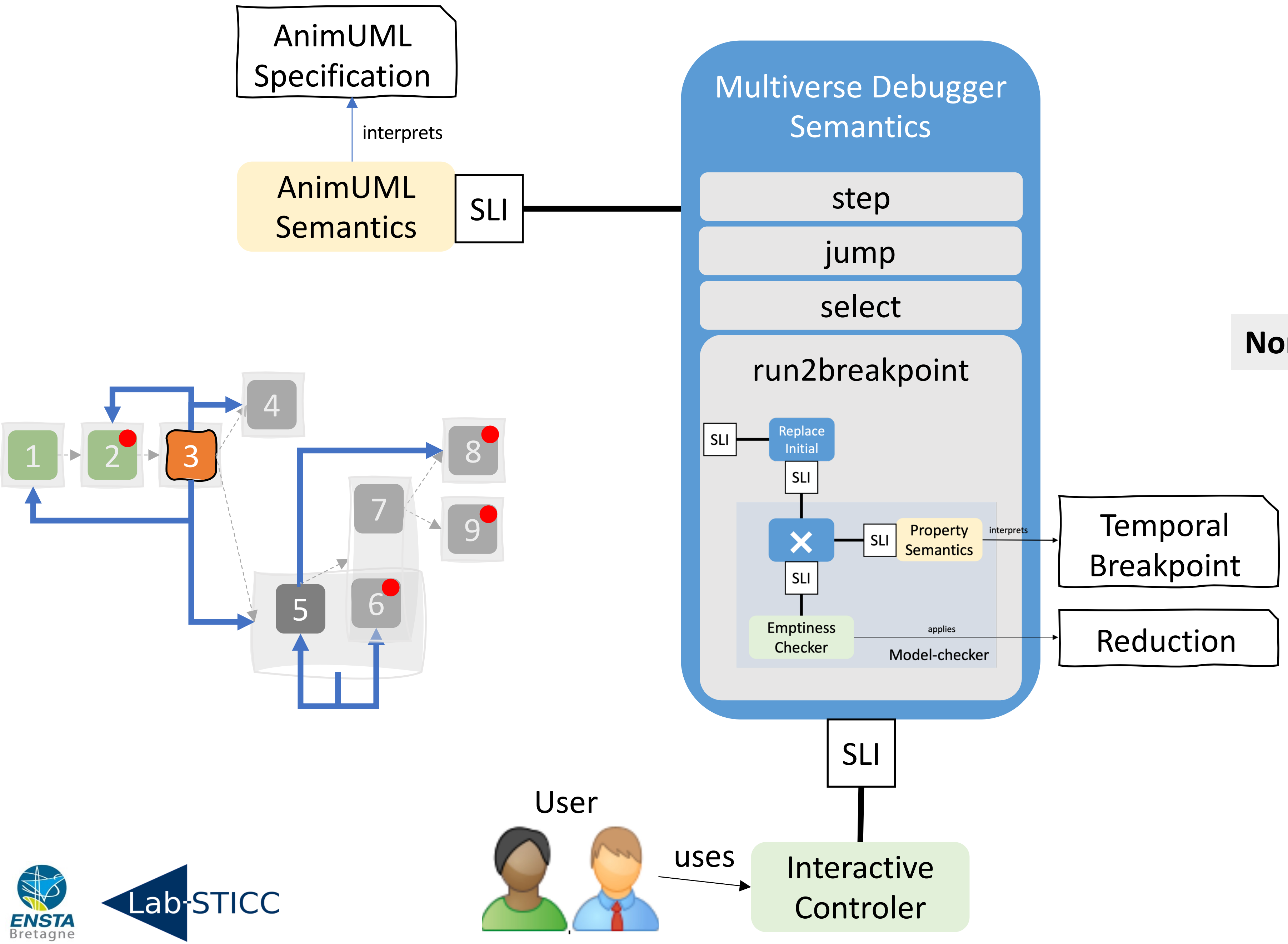
4. When G∀min∃ experiences the real world.

- Some experiences unravel reusable monitoring bridges
- Transfer to commercial products -- OBP2 inside
- Exploring hardware execution
- **Multiverse debugging made simple and more powerful**
- From zero to model-checker in 30 Hours



PhD Matthias PASQUIER

Non-trivial Monitor Composition



Submitted to ECMFA'23

Models'22

4. When G∀minƎ 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 practioners -- From zero to model-checker in 30 Hours**

From Zero To Model-Checker in 30 Hours

- Class at ENSTA Bretagne the last 2 years

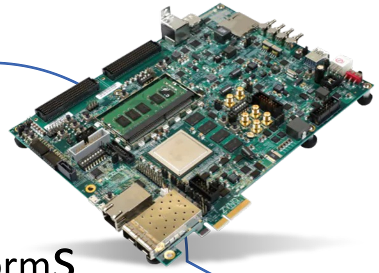
5. Sum Up & Ways Forward

Conclusion

Major Breakthroughs

Perspectives

Track Record



PlatformS

embedded: Bare-metal
hardware: FPGA

MonitorS

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

Model-checker
Multiverse Debugger
Execution Monitor

LanguageS

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

ENSTA Bretagne
Lab-STICC

Major Breakthroughs

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

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

ENSTA Bretagne Lab-STICC

Perspectives

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

- Short term:
  - Unifying scheduling and partial-order reduction
  - Language-agnostic timed semantics
- Midterm:
  - Towards open and dynamic abstraction-refinement
    - Heterogeneous refinement mappings
    - Overapproximations with maximal reuse of the base semantics
  - Heterogeneous models
- Long term:
  - Moldable diagnosis cockpit: language-agnostic portofolio-based diagnosis
    - Derive the proof of the soundness of the monitor
  - Algebraic algorithm specification
    - The isolation of the execution controller in Gamine can be seen as a generalization of recursion schemes from trees to arbitrary graphs.
    - Allow non-determinism during algorithm design = design algorithm families
    - Dataflow-focus to reduce over-constraining

ENSTA Bretagne

Lab-STICC

Track Record

Be curious, Explore,
Expand our understanding,
Share the insights

Phd students:
- Matthias Pasquier
- Emilien Fournier
- Tithnara Sun
- Valentin Besnard (prix GDR-GPL)
- Vincent Leilde
- Luka Le Roux
- Lamia Allal
- Jean-Philippe Schneider

Postdocs:
- Luka Le Roux
- Valery Monthe
- Bastien Drouout
- Fahad Golra
- Jean-Charles Roger
- Vincent Leilde

Engineers:
- Hiba Hnaini
- Sylvain Guerin
- Fatma Zarka
- Nadia Menad
- Sebastien Tleye
- Ismail Chaida

Papers:
- 1 patent
- 9 journal papers
- 49 conference papers

Software:
- OBP2 *nominated Systematic Paris-Région '20*
- ClockSystem
- Phadeo
- EMI UML
- AnimUML
- 50+ github repos

Main Projects: ONEWAY, Ker-SEVECO, JoinSafeCyber, VeriMoB, EASE4SE, DEPARTS, GEMOC, Ardyt, Morpheus, ValMadeo
Contracts: DAVIDSON, ERTOSGENER

www.obpcdl.org

ENSTA Bretagne
Lab-STICC