
CERN

Dániel Darvas (CERN / TU Budapest)

Practice-oriented formal methods for PLC programs of industrial control systems

PhD Symposium at iFM2016
05/06/2016, Reykjavík, Iceland

Contains joint work of B. Fernández, E. Blanco, S. Bliudze, J.O. Blech,
J-C. Tournier, T. Bartha, A. Vörös, I. Majzik, R. Speroni

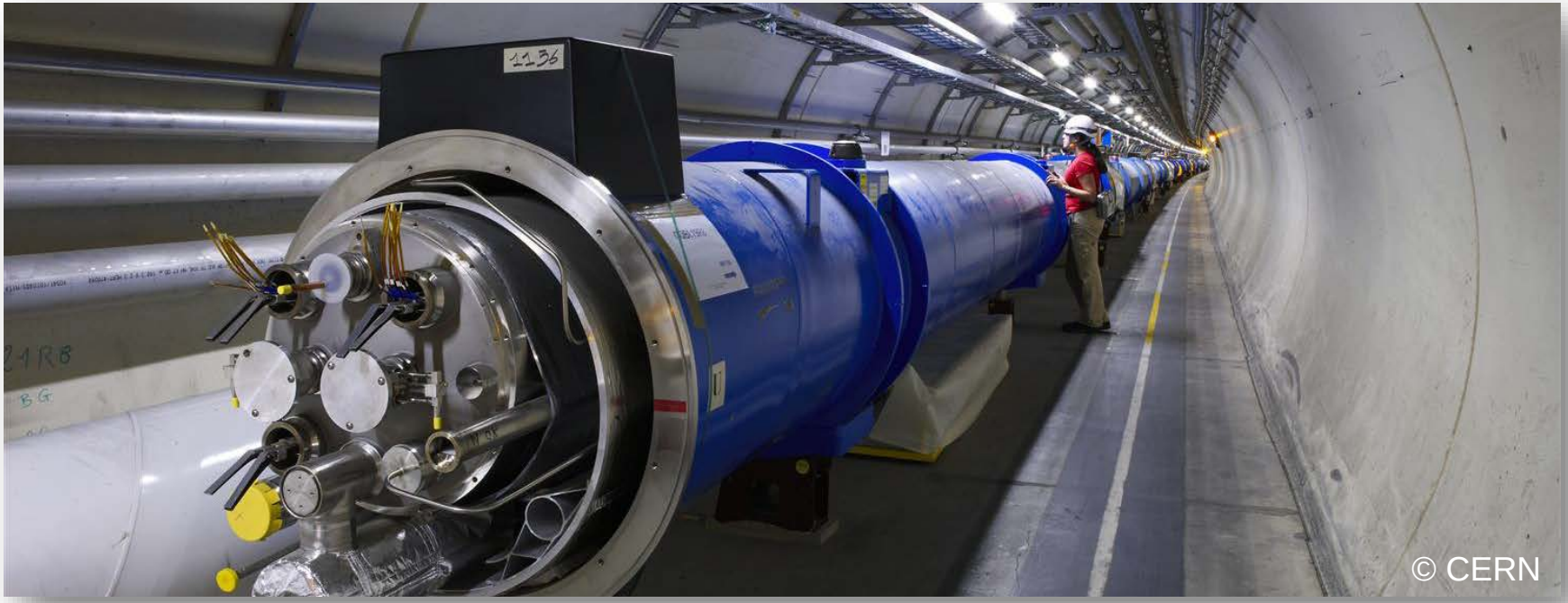
CERN

MŰEGYETEM 1782

http://go.cern.ch/6Lgf

CERN *European Org. for Nuclear Research*

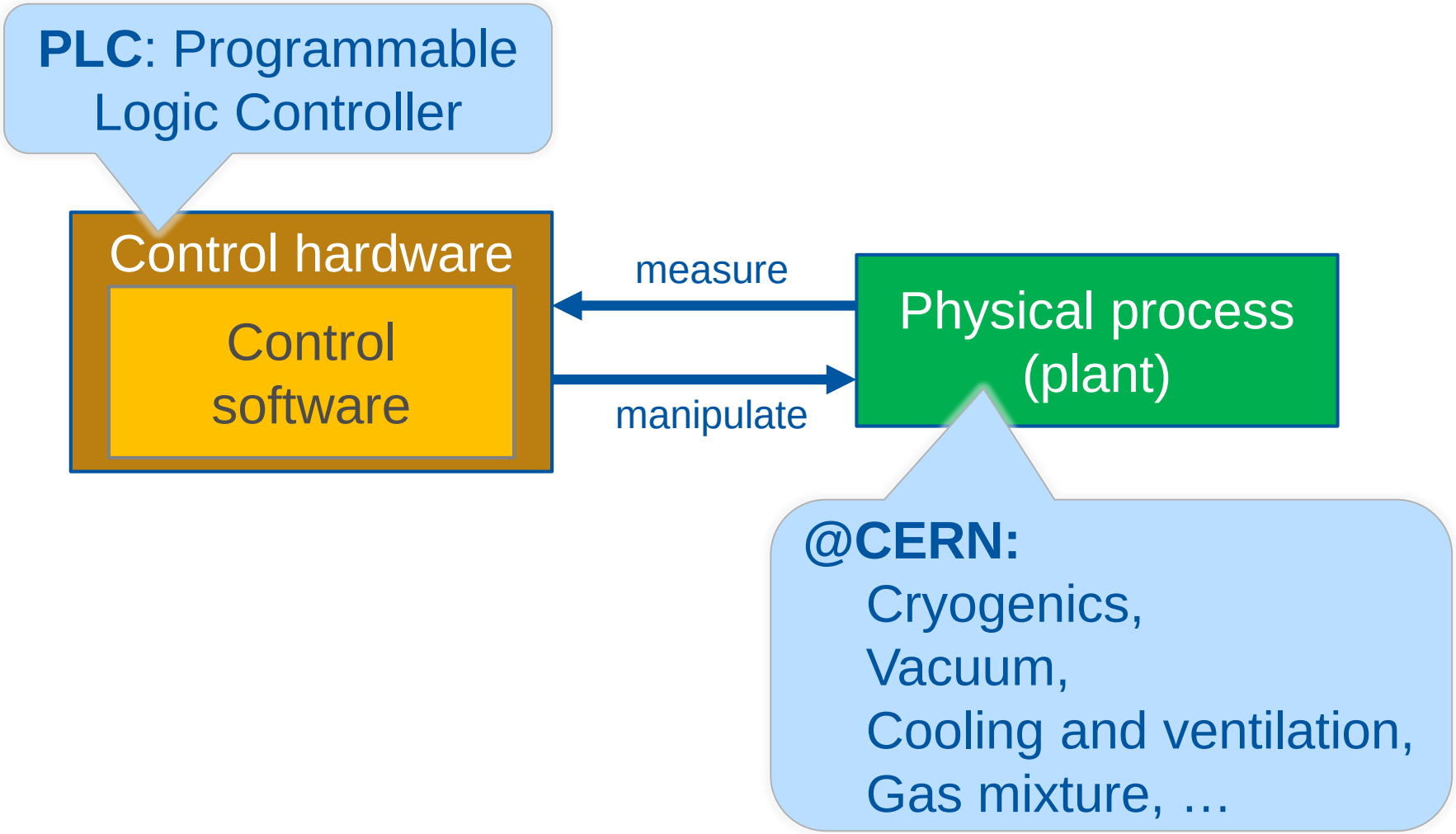
- Largest **particle physics laboratory**
- Large Hadron Collider (LHC)
  - Proton* beams with high energies



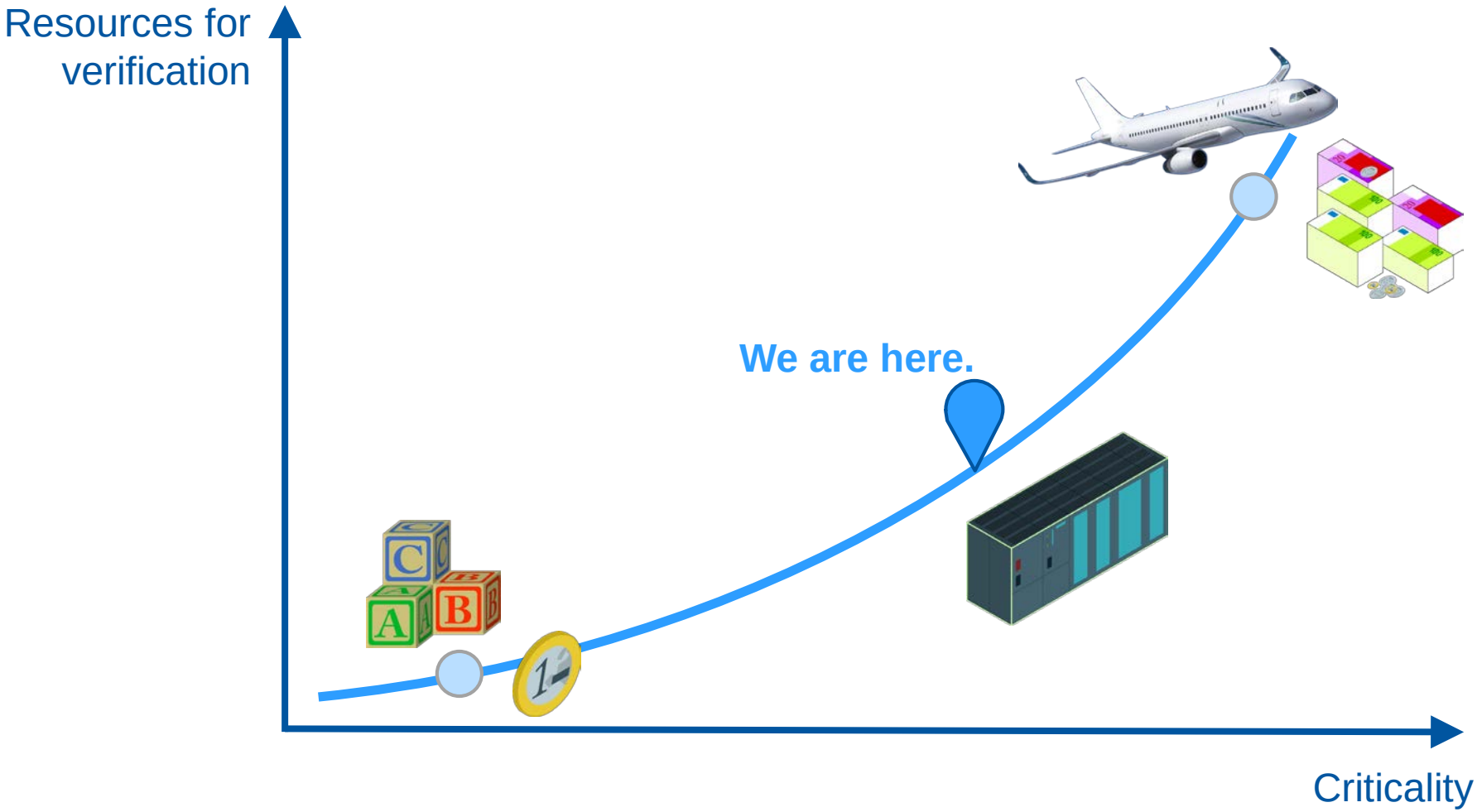
CERN

MŰEGYETEM 1782

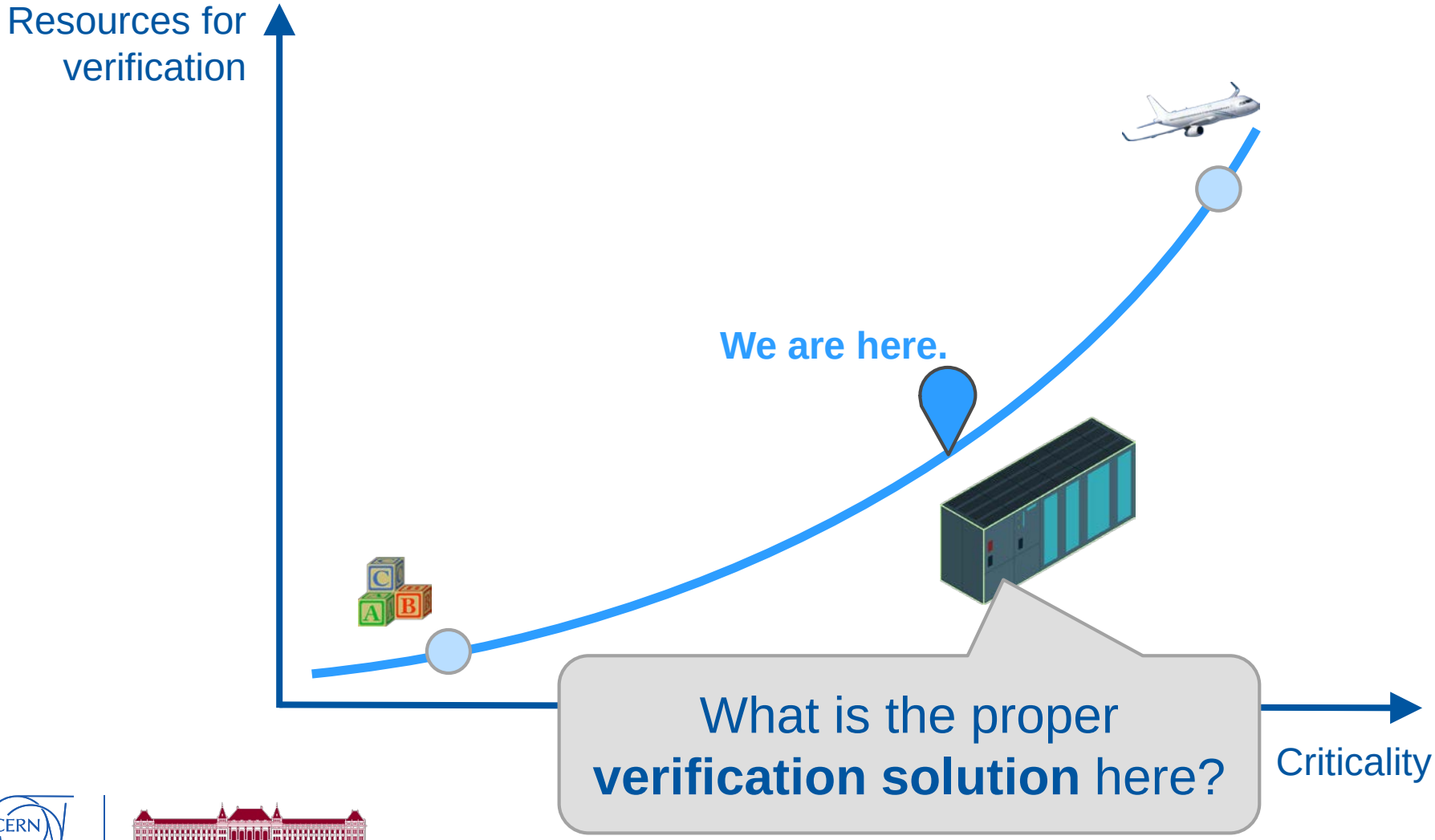
Industrial control systems (@ CERN)



Verification for medium-critical systems



Verification for medium-critical systems



Three challenges of the domain

- **Medium criticality**
  - Cannot afford special verification experts
  - Developers do the verification too
- **Typically non-IT background**
  - Cannot build on “base IT knowledge” (mathematics, UML, …)
- **Special devices**
  - Special programming languages
  - Lack of tools
  - Behind mainstream IT

How to ensure the quality of the software?

- **Testing**
  - Traditional, but not enough
- **Formal methods**
  - **Without involving formal methods experts!**

Three proposed approaches

	Inf. spec.	Impl.	Formal spec.	
#1	+	+		Model checking
#2	+	+	+	Conformance checking
#3	+		+	Code generation

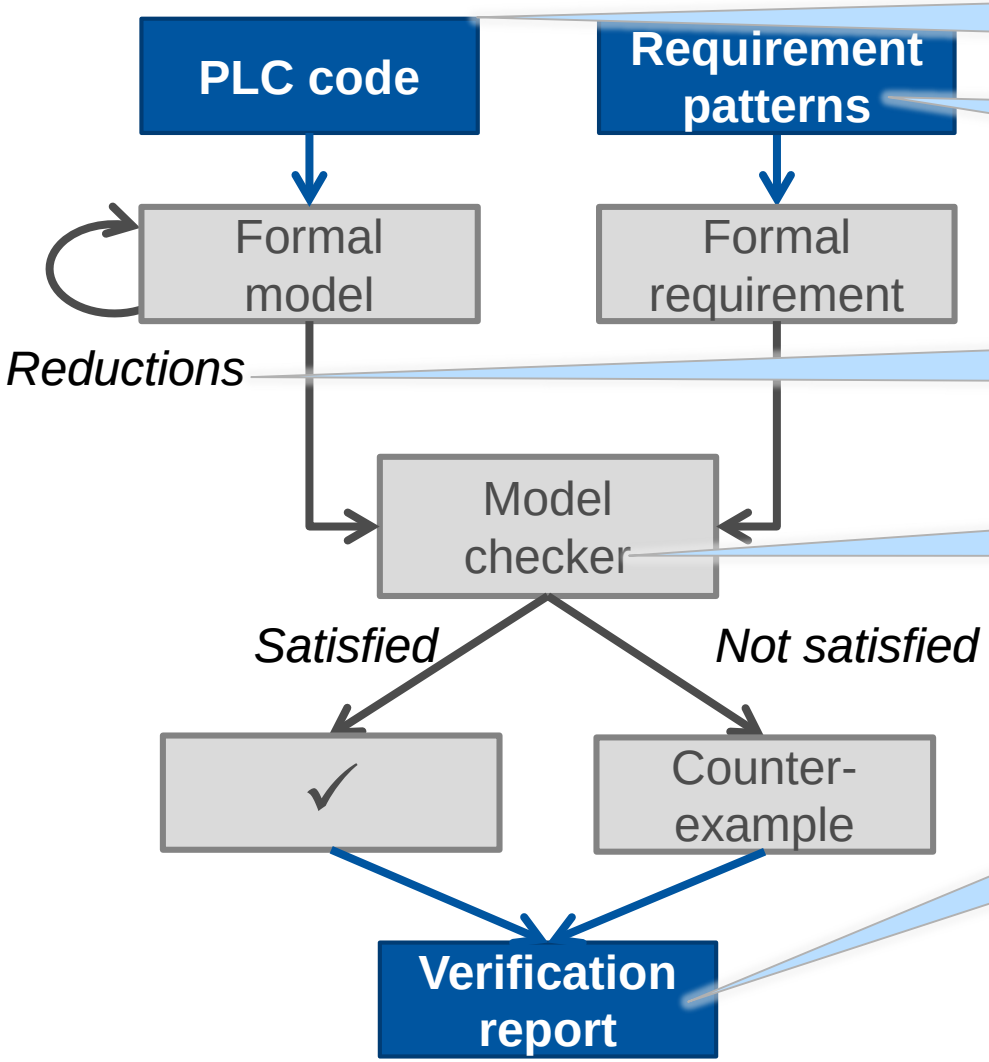
Challenges

- **Approach #1 (Model checking)**
  - Difficulty of using model checkers
  - Verification performance on huge models
  - Specifying requirements in temporal logic

- **Approach #2 (Conformance checking)**
  - Difficulty of using formal specification methods
  - Too strict conformance relations

- **Approach #3 (Code generation)**
  - Generated code should be understandable, modifiable
  - Generated code should match the specification
  - Difficult to describe invariant properties

Appr. #1: Model checking (PLCverif)



Based on the **implementation**

User-friendly **requirem't** specification

Heavily automated **reductions**

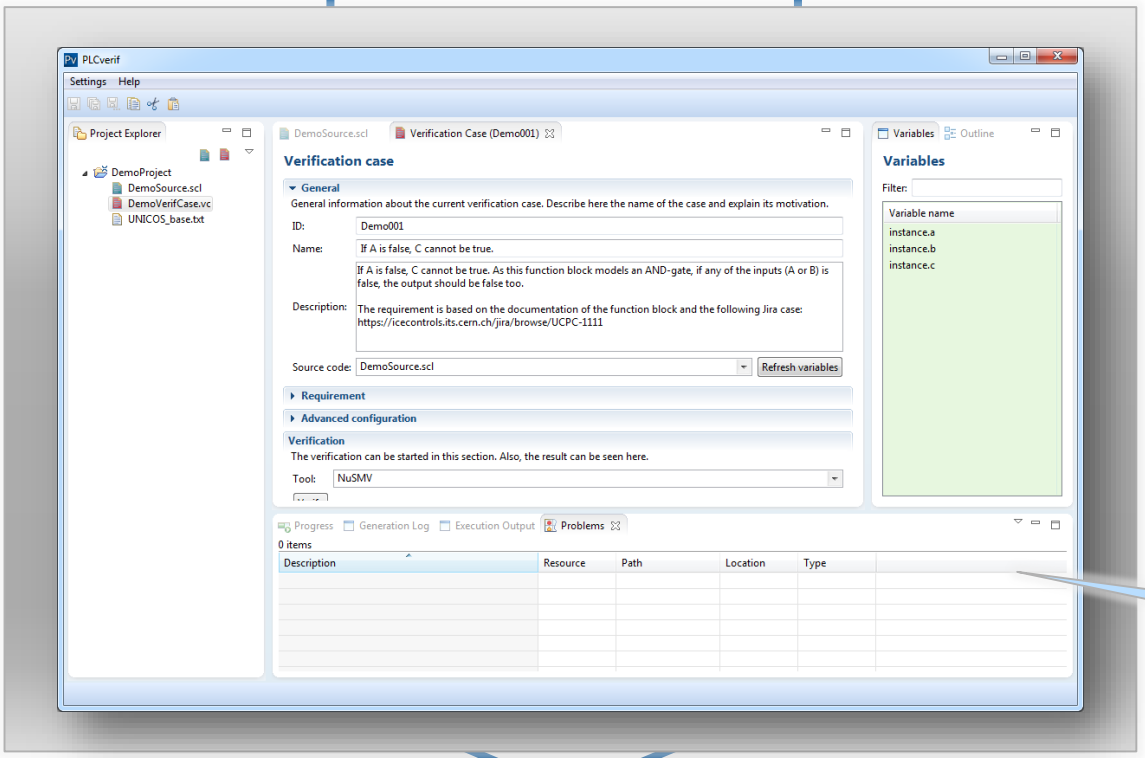
Replaceable **external model checker**

Self-contained **report** with counterexample

Appr. #1: Model checking (PLCverif)

PLC code

Requirement patterns



Verification report

Based on the **implementation**

User-friendly **requirem't** specification

Heavily automated **reductions**

Replaceable **external model checker**

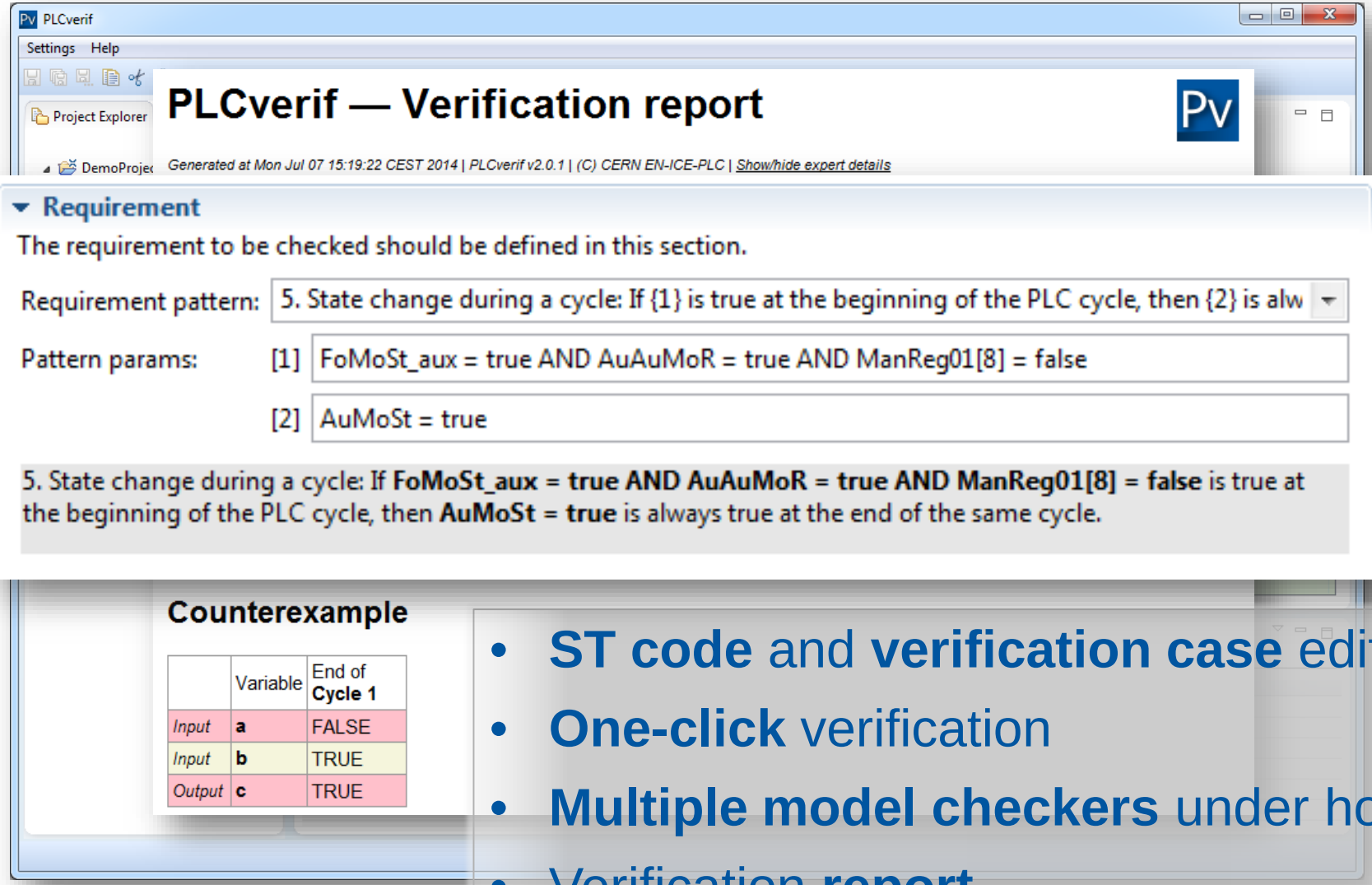
Self-contained **report** with counterexample

Tool hiding the formal details

CERN

MŰEGYETEM 1782

PLCverif



- **ST code** and **verification case** editor
- **One-click** verification
- **Multiple model checkers** under hood
- Verification **report**

http://cern.ch/plcverif

CERN

MŰEGYETEM 1782

Challenges

- **Approach #1 (Model checking)**
  - Difficulty of using model checkers
  - Verification performance on huge models
  - Specifying requirements in temporal logic

- **Approach #2 (Conformance checking)**
  - Difficulty of using formal specification methods
  - Too strict conformance relations

- **Approach #3 (Code generation)**
  - Generated code should be understandable, modifiable
  - Generated code should match the specification
  - Difficult to describe invariant properties

PLCspecif (example)

ExampleModule

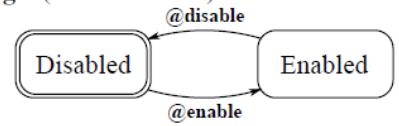
Assigned inputs:
- ValueReq : INT16
- EnableReq_fromLogic : BOOL
- EnableReq_fromScada : BOOL
- EnableReq_fromField : BOOL
- DisableReq : BOOL
- PMin : INT16 param
- PMax : INT16 param

Assigned outputs:
- Value : INT16
- Status : BOOL

Input definitions: — (none)

Event definitions:
- @disable ⇐ `rising_edge`(DisableReq) (`pri`=1)
- @enable ⇐ EnableReq_fromLogic OR EnableReq_fromScada OR EnableReq_fromField (`pri`=2)

Core logic (state machine)



Output definitions:
- _Value =

`ValueReq` < `PMin`	`ValueReq` > `PMax`	result
T	·	`PMin`
F	T	`PMax`
F	F	`ValueReq`

- Value =

`in_state`(Enabled)	result
T	`_Value`
F	0

- Status = `in_state`(Enabled)

Invariant properties:
- ALWAYS *PMin* ≤ *Value* ≤ *PMax* ASSUMING *PMin* ≤ *PMax*

Detailed behaviour specification

Structured, **hierarchical**

Separation of concerns

Domain-specific semantics

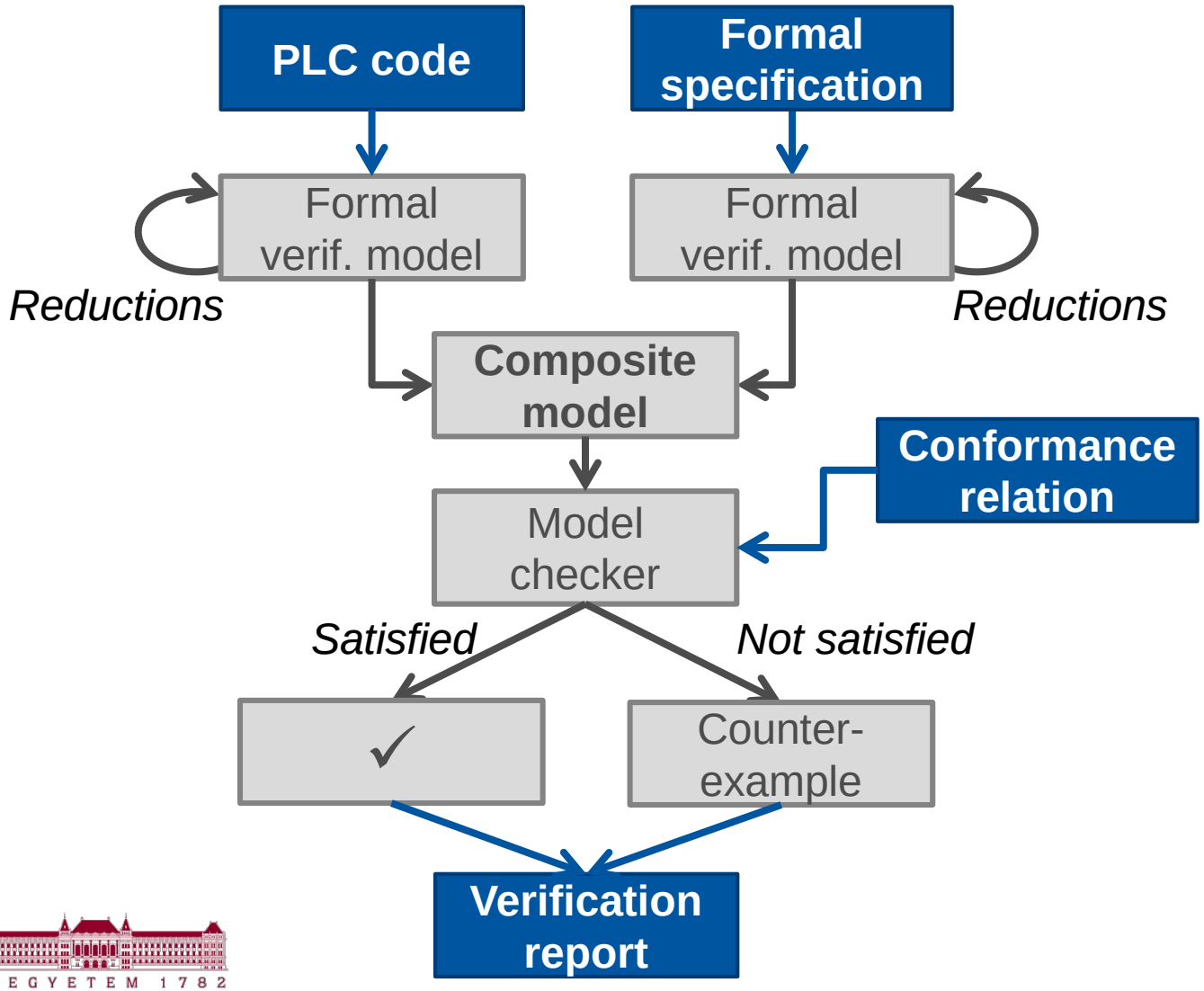
Unifies different semi-formal formalisms

Supports **verification**

CERN

MŰEGYETEM 1782

Appr. #2: Formal specification + conformance checking



Challenges

- **Approach #1 (Model checking)**
  - Difficulty of using model checkers
  - Verification performance on huge models
  - Specifying requirements in temporal logic

- **Approach #2 (Conformance checking)**
  - Difficulty of using formal specification methods
  - Too strict conformance relations

- **Approach #3 (Code generation)**
  - Generated code should be understandable, modifiable
  - Generated code should match the specification
  - Difficult to describe invariant properties

Appr. #3: Code generation

- Code generation defined based on the **formal semantics** of PLCspecif

- **Special needs**:
  - **Configurable** code generation
  - Possibility to **edit manually**!

    → Understandable code needed

    → **Conformance checkin**g required

- Not possible for **fail-safe** PLCs

Which one is the best? – Comparison

- **"Coverage"**
  - *Model checking*: only the specified requirements
  - *Other methods*: complete specification
- **"Intrusiveness"**
  - *Model checking*: development process is extended only
  - *Conformance checking*: not intrusive, but formal specification needed
  - *Code generation*: radical change with mixed feelings
- **Complementary methods** for different purposes

Challenges and our solutions

- **Approach #1 (Model checking)**
  - **Wrappers** for model checkers, generic **intermediate model**
  - **Generic** and **domain-specific reductions**
  - **Requirement patterns**
- **Approach #2 (Conformance checking)**
  - Building on **semi-formal languages with unified semantics**
  - **Permissive conformance relations**
- **Approach #3 (Code generation)**
  - **Customisable code generation**
  - Code generation **based on the formal semantics**
  - **Additional support for defining and checking invariants**

Use case: Magnet testing plant

- **Goal:** ensuring **safety** by allowing/forbidding tests
- Verified in **development phase**
- **Problems** found:
  - **Requirement misunderstanding**
  - **Functionality problems**
  - **Safety problems** (some well-hidden problems)

→ More details in our **iFM2016 paper**

CERN

MŰEGYETEM 1782

www.cern.ch

For more information…

- Project website (with publication list)
  http://cern.ch/project-plc-formalmethods/
- PLCverif tool's website
  http://cern.ch/plcverif
- PLCspecif's website
  http://cern.ch/plcspecif
- CERN website – http://home.cern

- **Contact me**
  daniel.darvas@cern.ch
  http://cern.ch/ddarvas

Model checking at CERN

- D. Darvas et al. **Formal verification of complex properties on PLC programs**. Formal Techniques for Distributed Objects, Components, and Systems (LNCS 8461), pp. 284-299, Springer, 2014.
- B. Fernández et al. **Bringing automated model checking to PLC program development – A CERN case study**. Proc. of the 12th Int. Workshop on Discrete Event Systems, pp. 394-399, 2014.
- D. Darvas et al. **PLCverif: A tool to verify PLC programs based on model checking techniques.** Proc. of the 15th Int. Conf. on Accelerator & Large Experimental Physics Control Systems, pp. 911-914, JaCoW, 2015. http://dx.doi.org/10.18429/JACoW-ICALEPCS2015-WEPGF092
- B. Fernández et al. **Applying model checking to industrial-sized PLC programs.** IEEE Transactions on Industrial Informatics, 11(6):1400-1410, 2015. http://dx.doi.org/10.1109/TII.2015.2489184

Formal specification at CERN

- D. Darvas et al. **Requirements towards a formal specification language for PLCs.** 2014. http://dx.doi.org/10.5281/zenodo.14907
- D. Darvas et al. **A formal specification method for PLC-based applications.** Proc. of the 15th Int. Conf. on Accelerator & Large Experimental Physics Control Systems, pp. 907-910, JaCoW, 2015. http://dx.doi.org/10.18429/JACoW-ICALEPCS2015-WEPGF091
- D. Darvas et al. **Syntax and semantics of PLCspecif.** CERN Report, EDMS 1523877, 2015. https://edms.cern.ch/document/1523877
- D. Darvas et al. **Formal verification of safety PLC based control software.** Integrated Formal Methods (LNCS 9681), pp. 508-522, Springer, 2016. http://dx.doi.org/10.1007/978-3-319-33693-0_32