CYBERSECURITY FOR EMBEDDED SYSTEMS

SEBASTIANO FABRIZIO FINOCCHIARO

POLITECNICO DI TORINO – DAUIN – COMPUTER AND CONTROL ENGINEERING
PHD CYCLE: XXXII
ACADEMIC SUPERVISOR: GIANPIERO CABODI

POLITECNICO DI TORINO

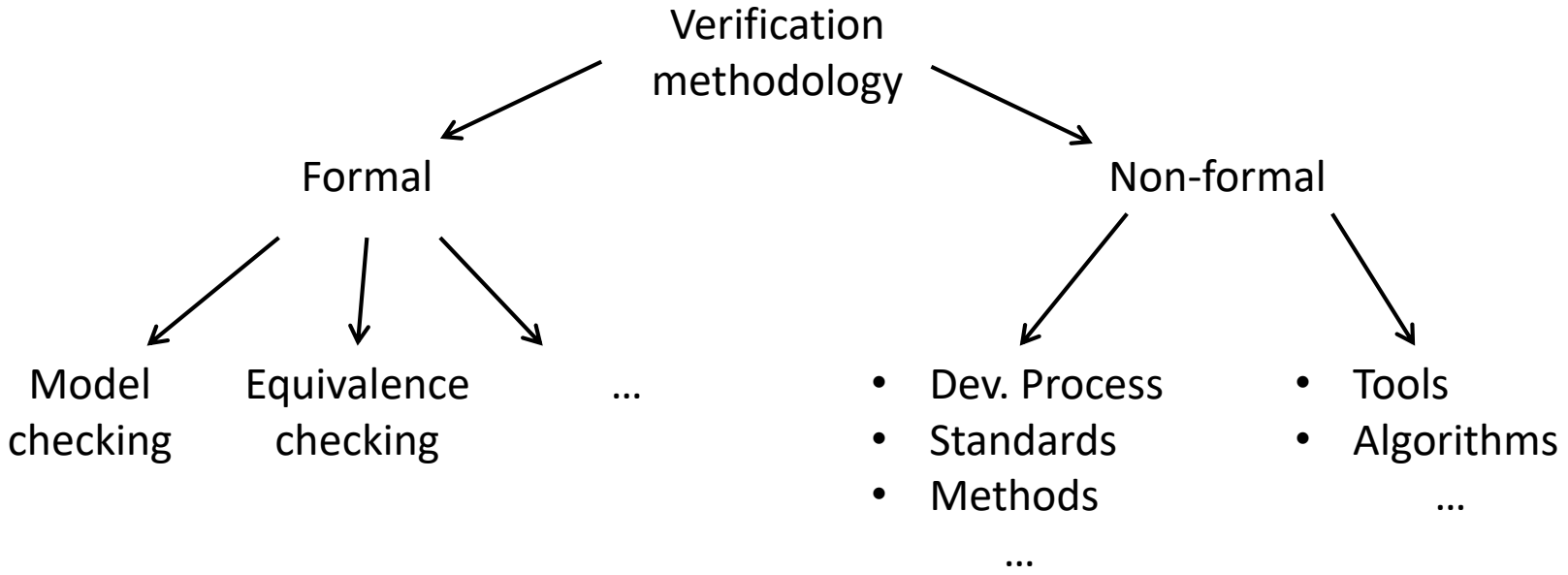
Outline

- Introduction
- Cyber Physical Systems
- Formal verification
- Taint properties
- Path properties
- Secure path verification
- Spectre and Meltdown
- Conclusions

General overview

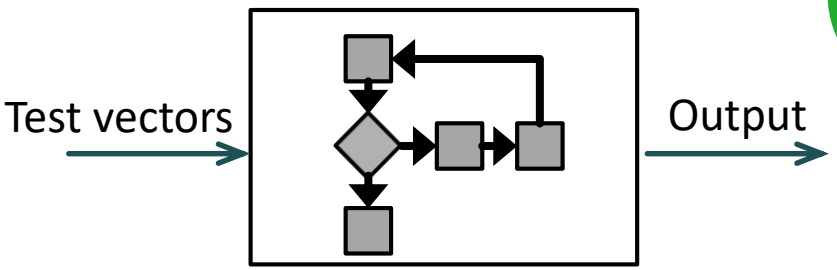
- Research topic: CYBERSECURITY FOR EMBEDDED SYSTEMS

- Collaboration with Magneti Marelli:
  - This PhD grant is funded by Magneti Marelli – Automotive Lighting (Orbassano)

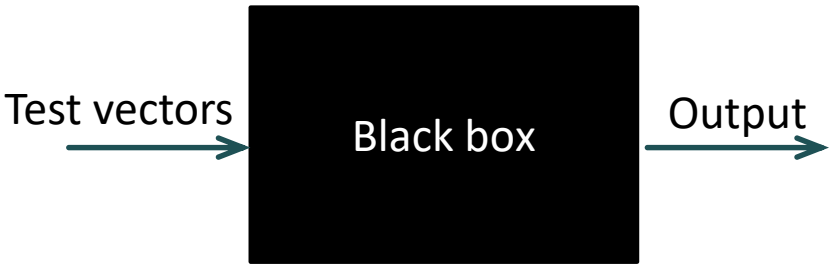


Model checking vs traditional verification

Traditional approach

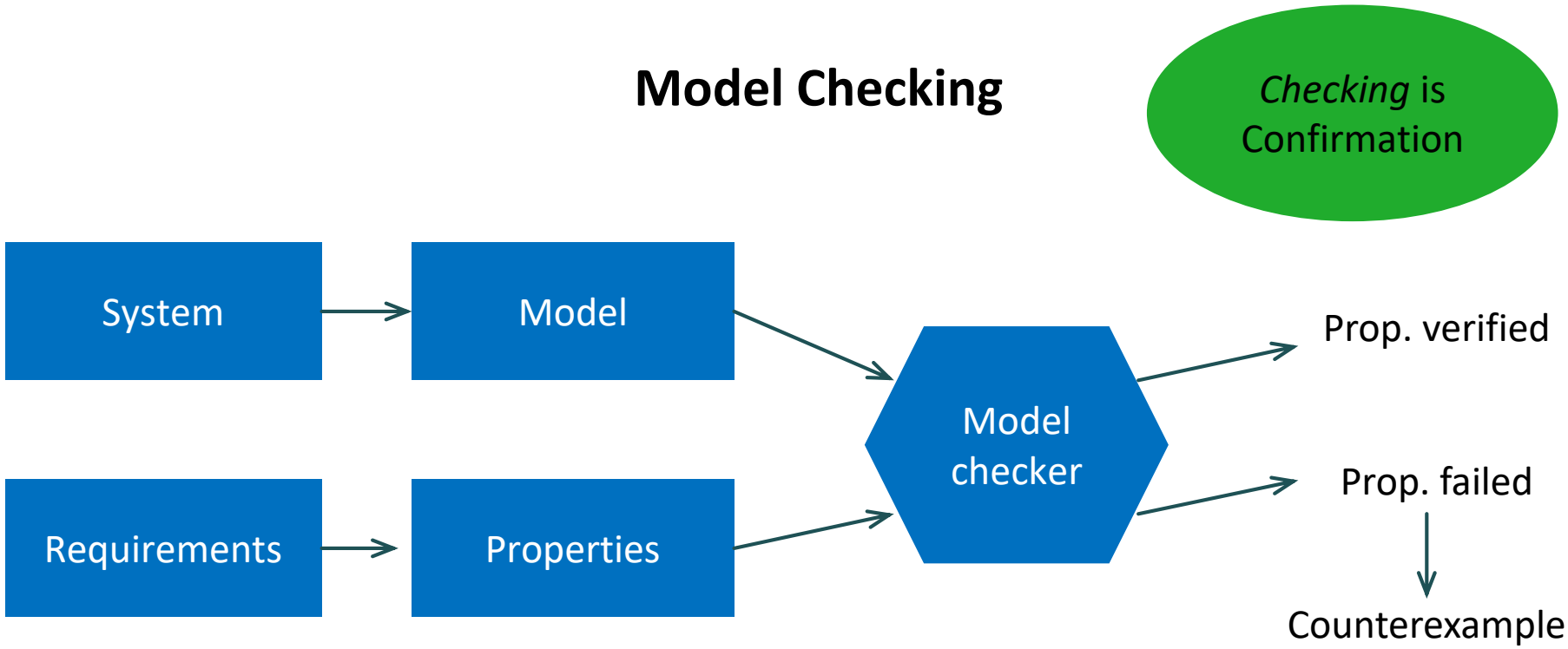


Testing is Exploration and Learning



- Cannot cover all possible cases
- Likelihood of uncovered subtle bugs

Model checking vs traditional verification



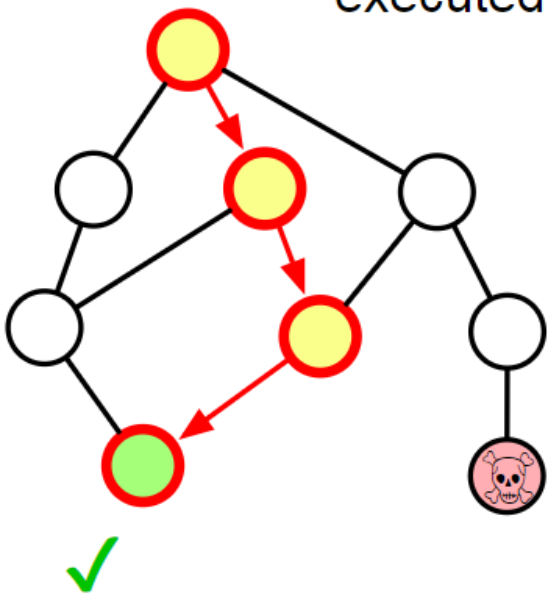
- Equivalent to simulating **all** cases in simulation
- No bug (according to the property)
- More complex but more exhaustive then simulation
- Suited for hard problems

Testing vs Model Checking

testing:

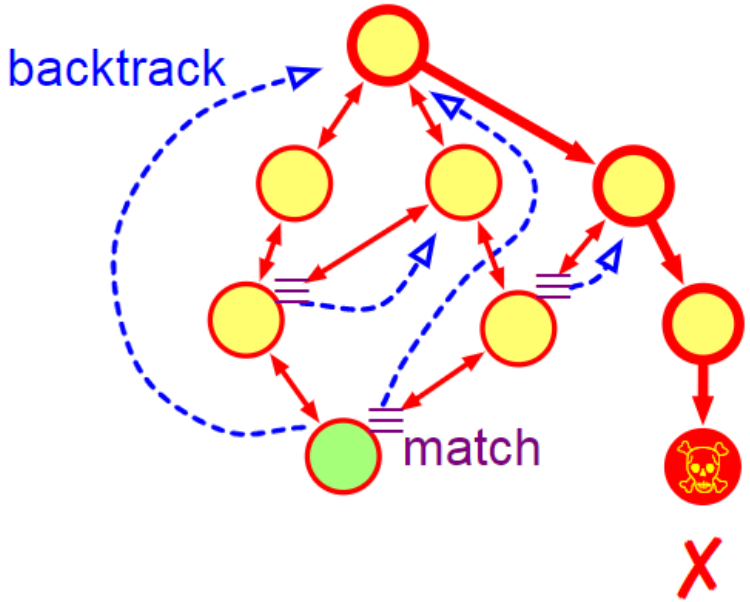
{d}

based on input set {d}
only one path
executed at a time



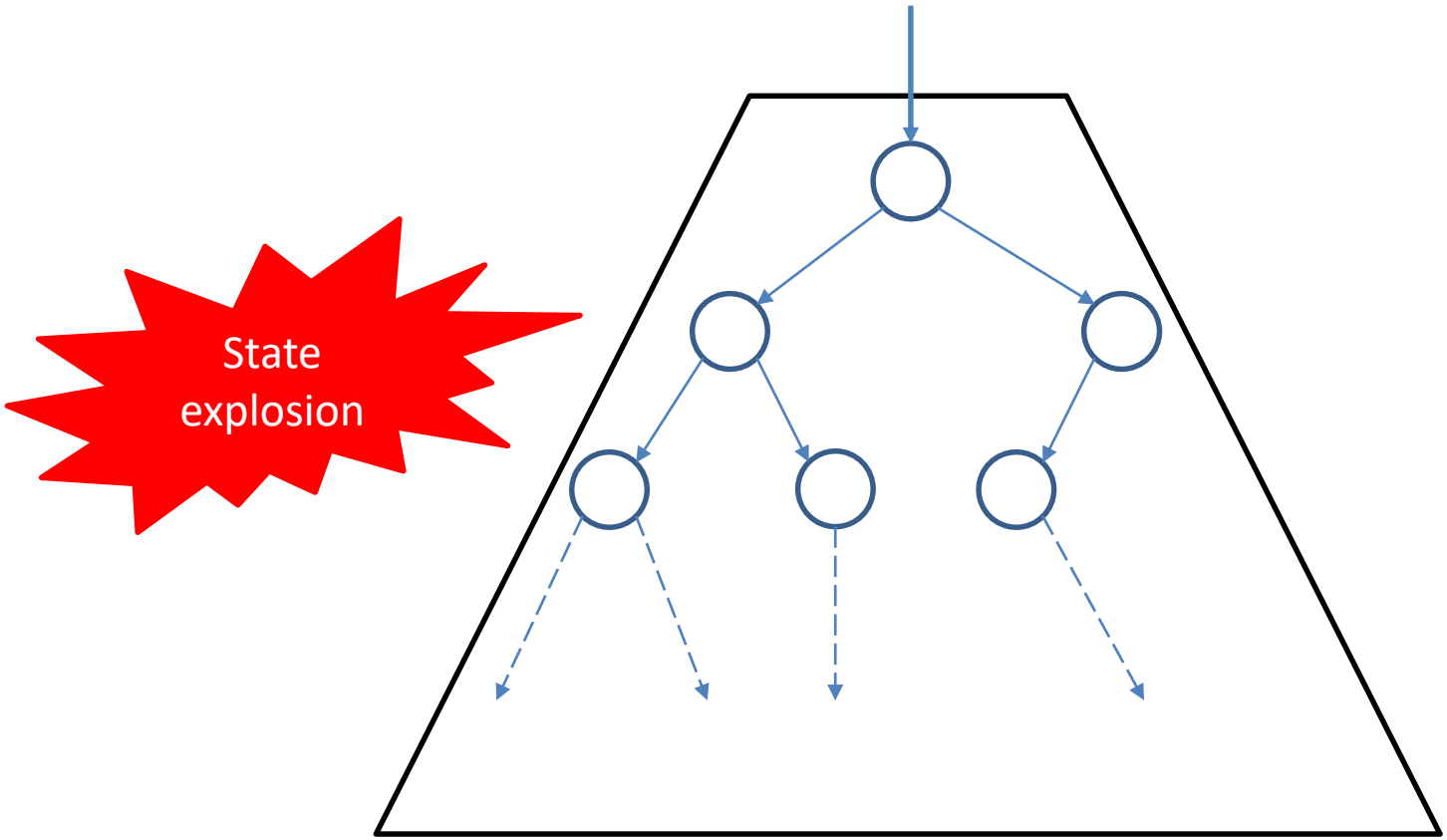
model checking:

all program state are explored
until none left or defect found



State space explosion

The global state space of a concurrent system has size **exponential** in the number of concurrent processes (i.e. 2^n)



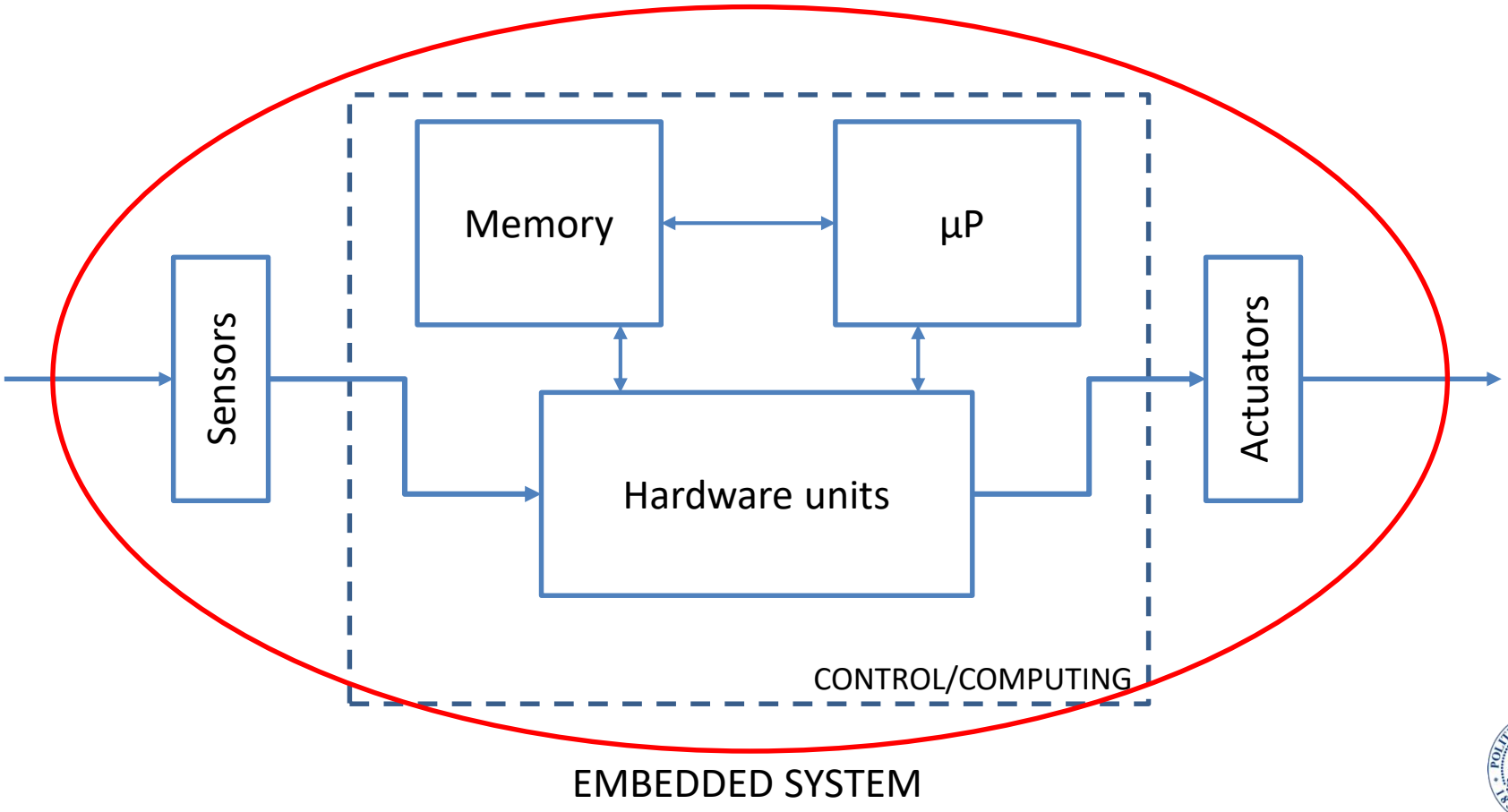
State space explosion

Approaches to the state space explosion:

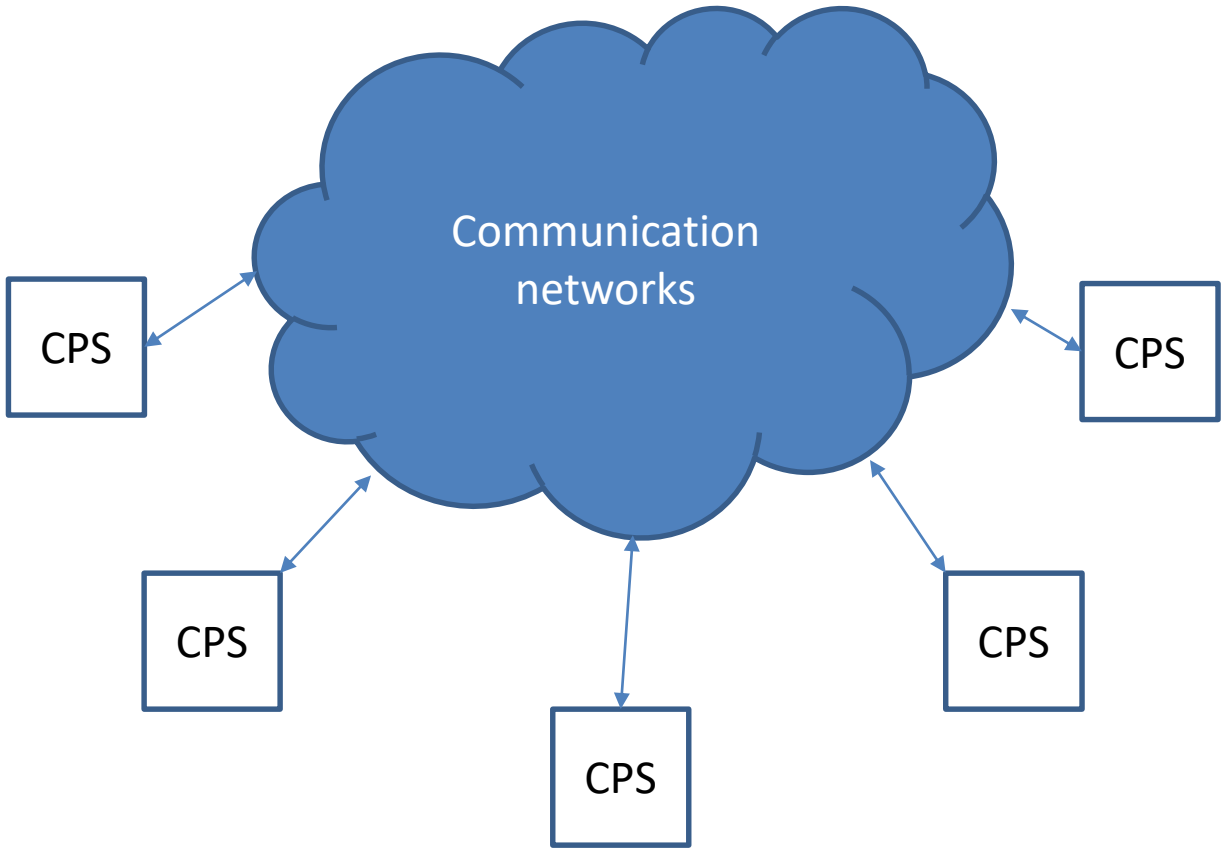
- Binary Decision Diagrams (BDDs)
- Partial order reduction
- Bounded Model Checking (BMC)
- Abstraction and refinement

Cyber Physical Systems

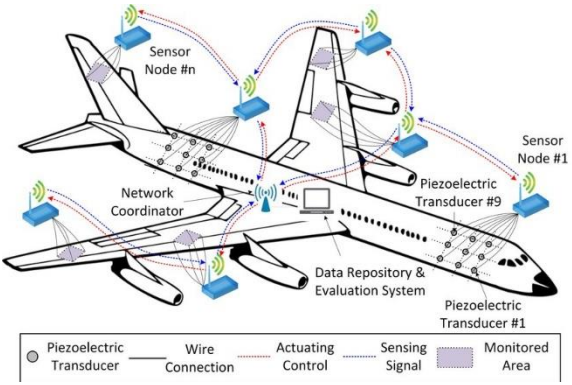
- Cyber-Physical Systems (CPS) are integrations of computation and physical processes within a networked environment

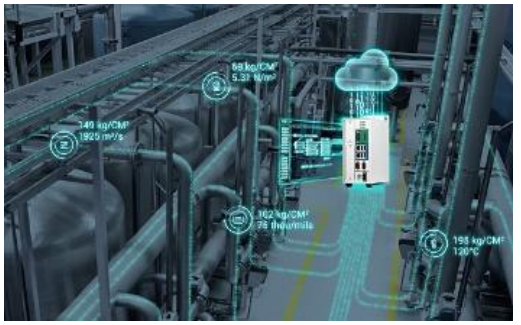


Cyber Physical Systems



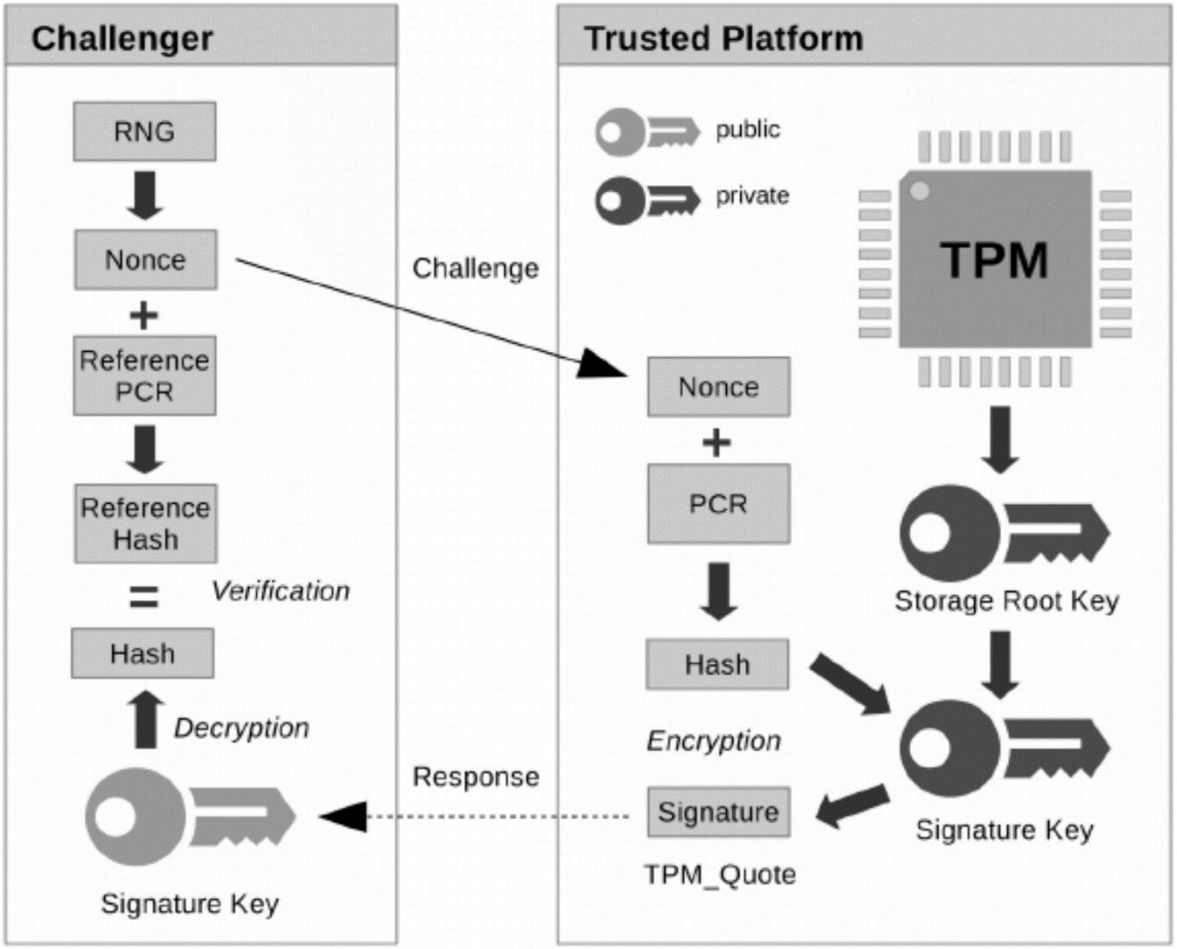
Cyber Physical Systems



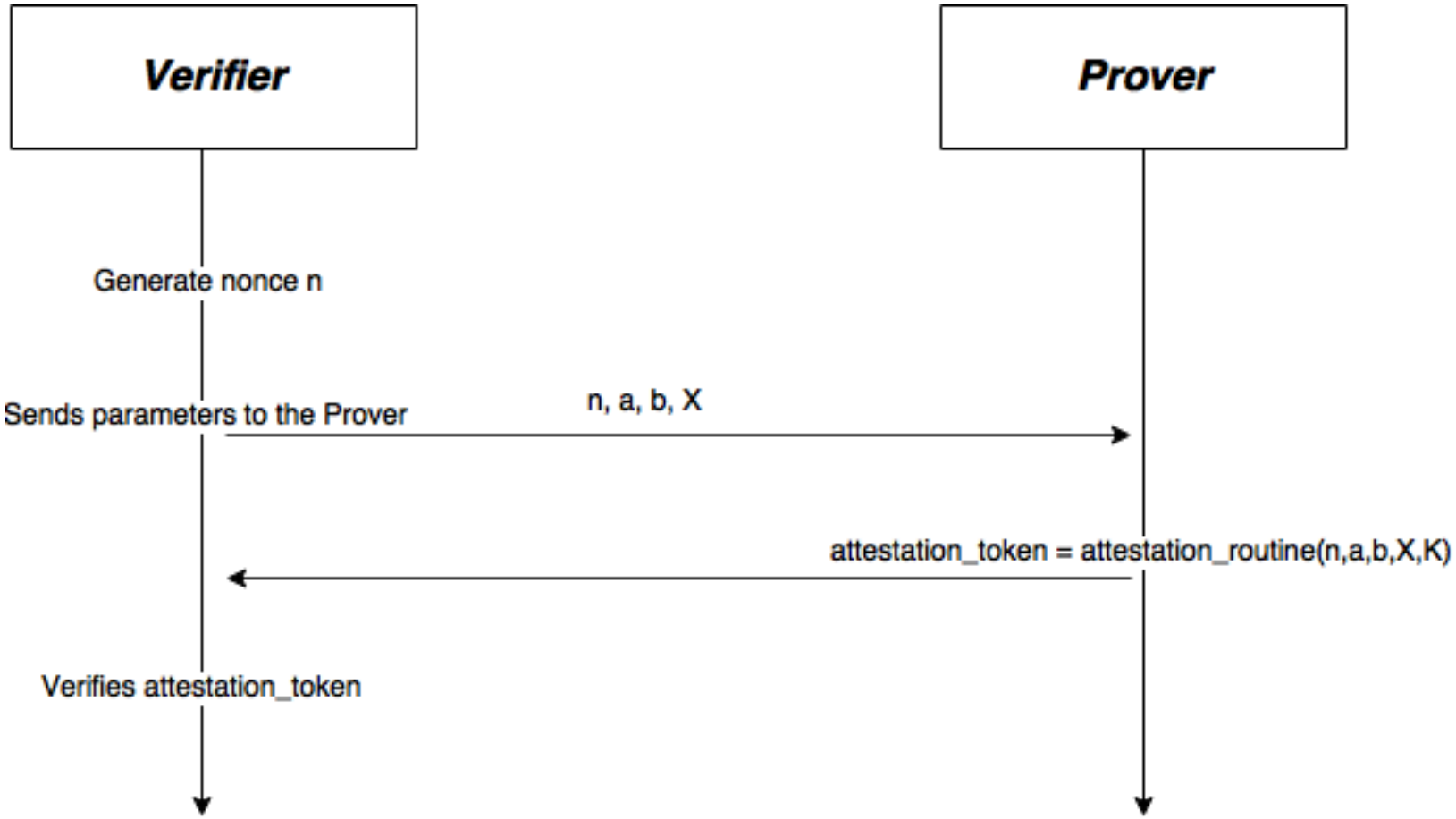


Security requirements: CIA Triad

Trusted Platform Module/Root of trust

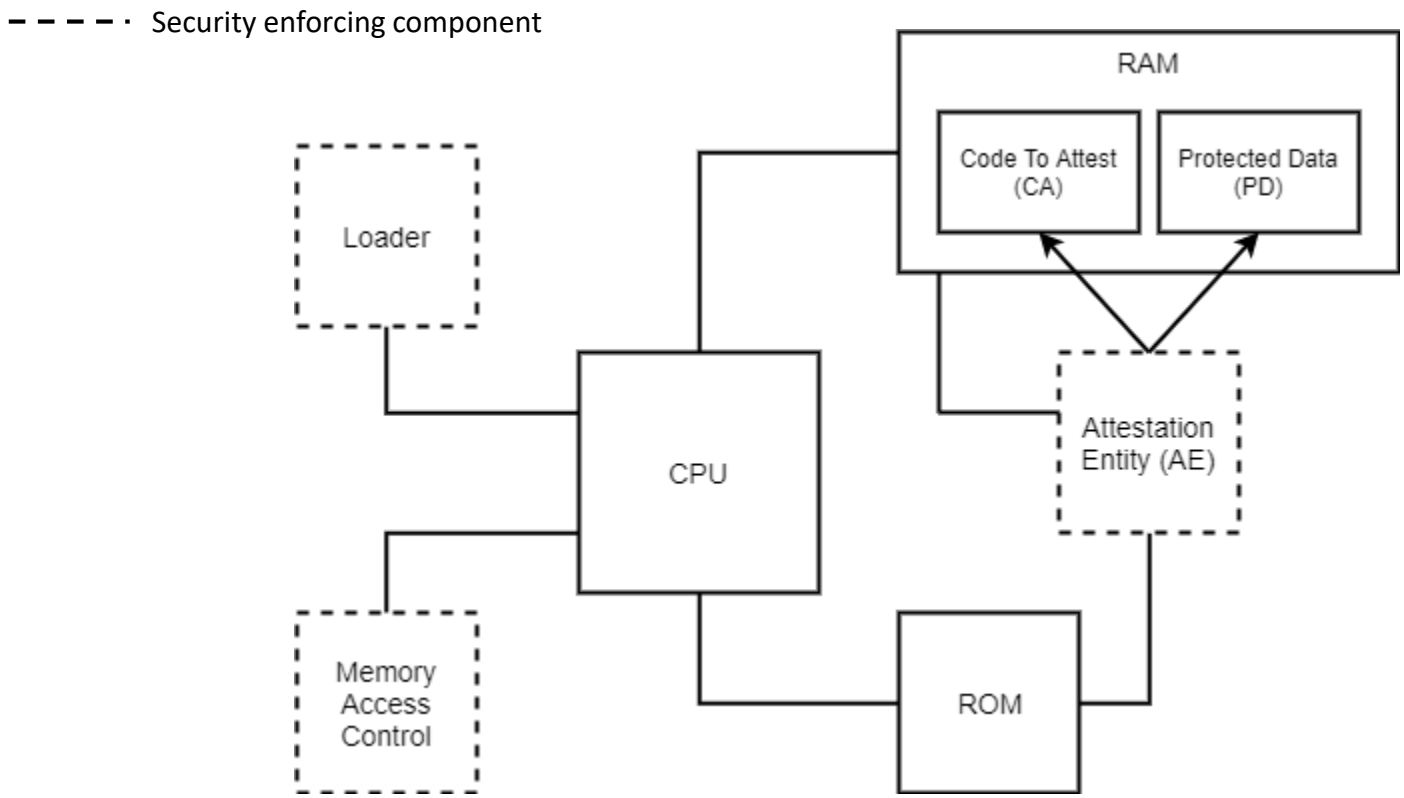


Remote Attestation



Abstract model

Based on the remote attestation architectures SMART and SANCUS



Abstract model: mapping

Mapping between the abstract model, SMART and SANCUS

Abstract Model	**SMART**	**SANCUS**
Attestation Entity (AE)	Attestation Routine (ROM)	Text Section of SMs (RAM)
Code to Attest (CA)	Code (A,B)	SM
Memory Access Control (MAC)	Bus Controller	Memory Access Logic (MAL)
Loader	-	Modified CPU

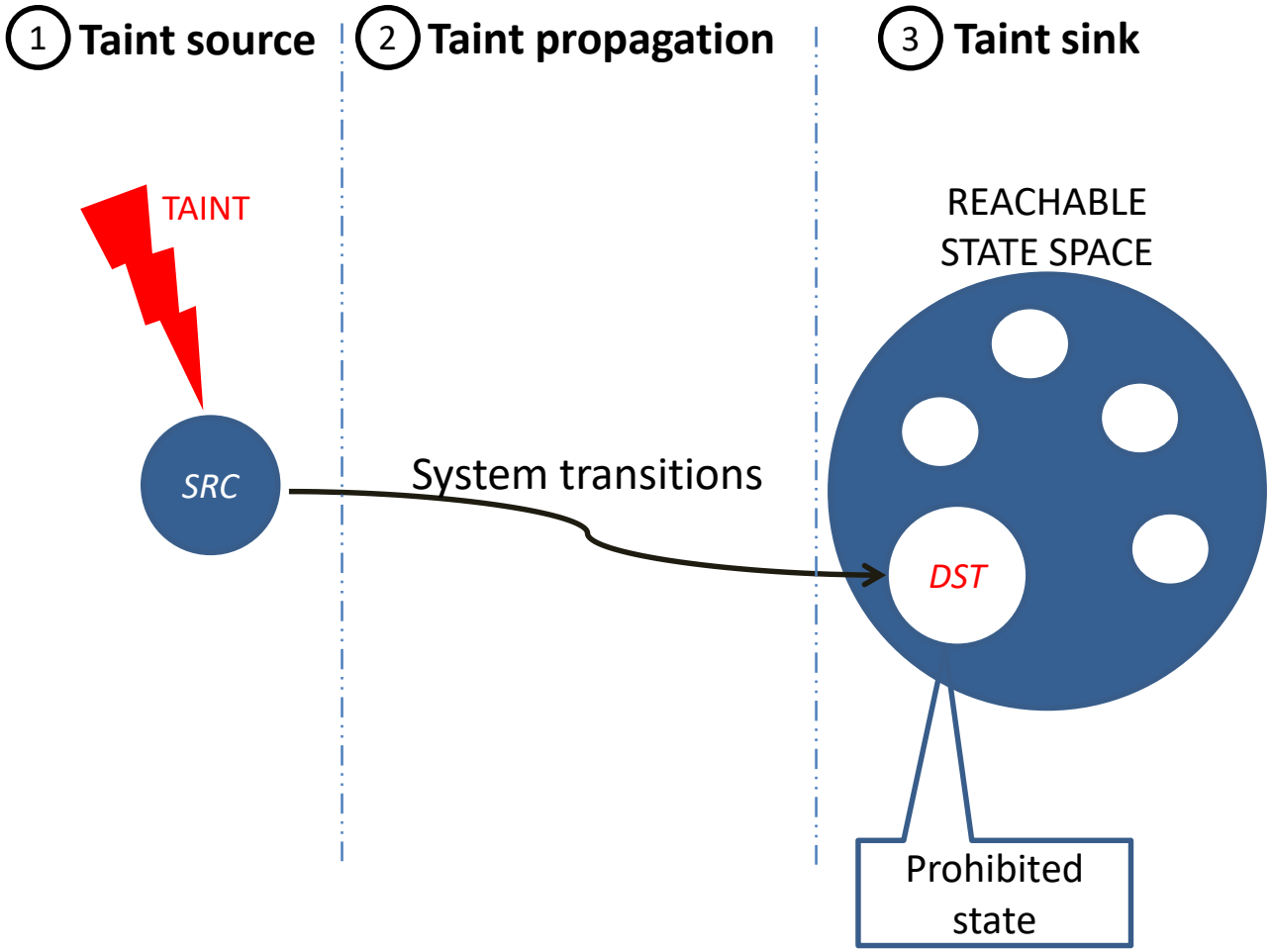
Research questions

- Is it possible to read any security related data from a certain location?
- Is it possible to modify any security related data?
- Are there failure modes that would compromise any security related data?

Security requirements

- Key secrecy
- Exclusive access
- No leakage
- Isolation (Immutability and Mode separation)
- Uninterruptibility
- Fixed entry point

Taint Property



Taint Property: example

Example of Confidentiality: **Exclusive access to the key**

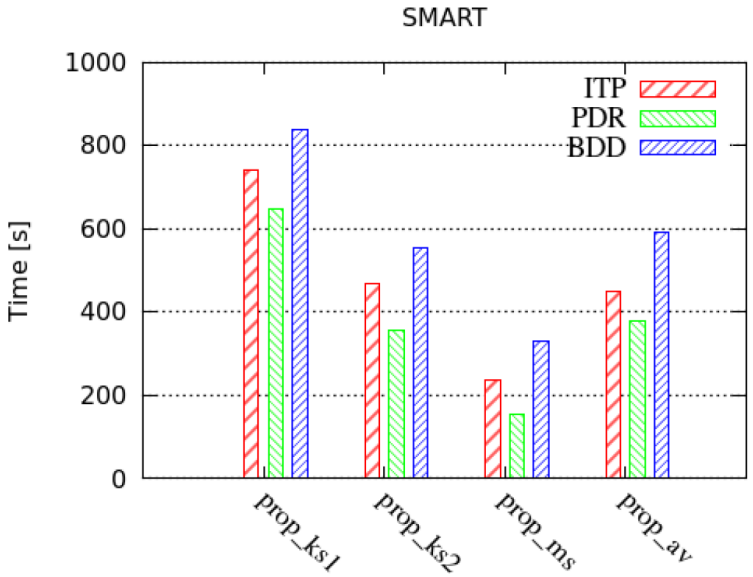
All user programs (untrusted entities) are not allowed to load the key K into any CPU registers.
Only the Attestation Entity (AE) is allowed to access the key. The Memory Access Control (MAC) logic implements the controls related to key secrecy.
The MAC checks if the address in the Program Counter (PC) belongs to the AE address space.

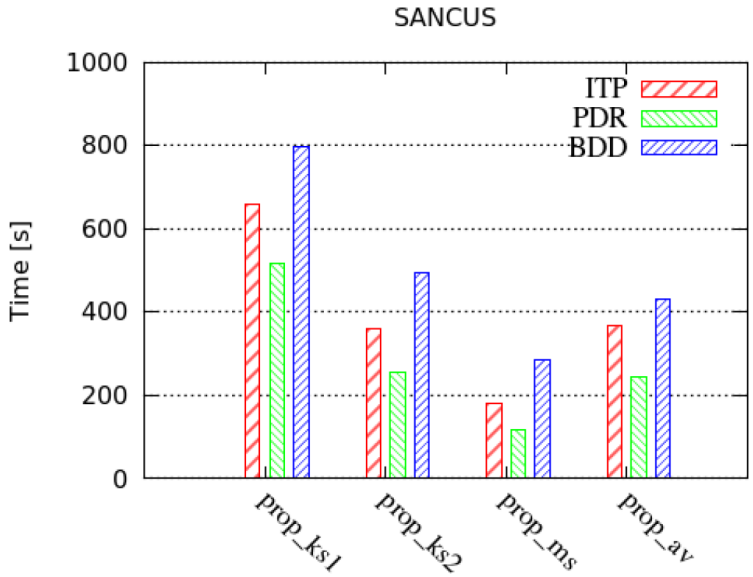
$$AG\Big(\neg is\_in\_AE\_scope \wedge fetchedInstr = load(key\_addr, CPU.registerX) \rightarrow AX\Big(\neg isIn(K, CPU.registerX)\Big)\Big)$$

TAINT: only the Attestation Entity can access the key, therefore the taint could be any untrusted entity accessing the memory region.

Experimental results

Property	SMART		SANCUS	
	Time[s]	*Result*	*Time[s]*	*Result*
prop_ks1	648.22	PASS	518.82	PASS
prop_ks2	357.45	PASS	254.35	PASS
prop_ms	156.7	PASS	116.74	PASS
prop_av	378.24	FAIL	245.29	FAIL





Contributions

- Abstraction and modelling of a specific class of security systems: root of trust/remote attestation
- Definition of a portfolio of security-related Taint Properties
- Formal verification on real architectures
- Model checker support

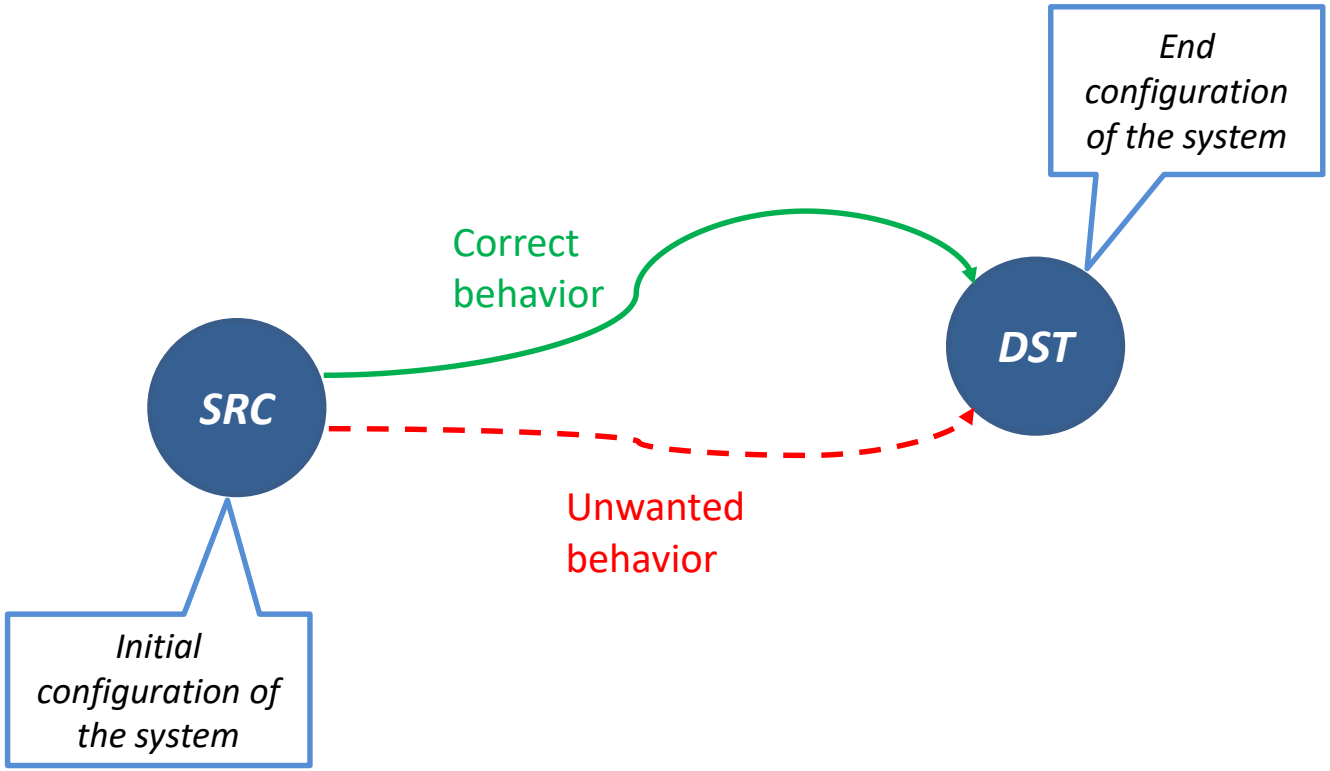
[1] Cabodi, G.; Camurati, P.; Finocchiaro, S.F.; Loiacono, C.; Savarese, F.; Vendraminetto, D. (2016) "Secure Embedded Architectures: Taint Properties Verification". In: International Conference on Development and Application Systems.

From Taint to Path properties

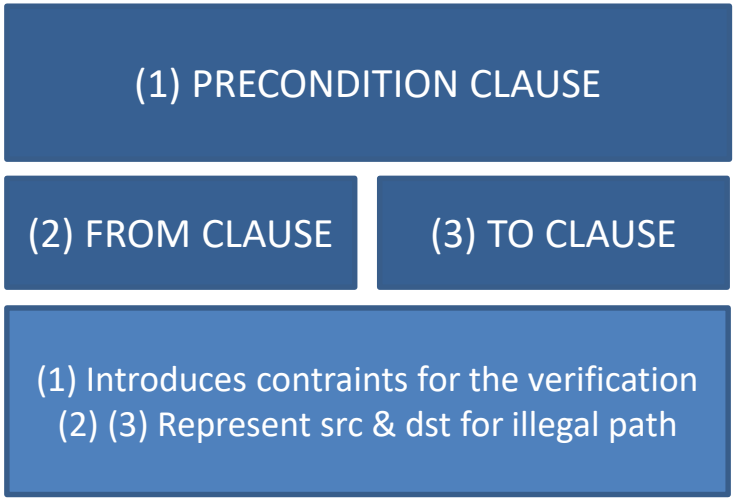
Taint properties are information flow properties because they involve data propagation through the system, but they only check for the final state.

What if we want to express properties about the propagation itself?

Path Property



Path Property



Example of Confidentiality: **Exclusive access to the key**

$$assert\_no\_path \quad from \quad \mathbf{AE} \quad to \quad \mathbf{CPU.registerX}$$

$$pre\_cond:$$

$$\neg is\_in\_AE\_scope \wedge load(key\_addr, CPU.registerX)$$

Contributions

- Definition of a new class of property: Path property
- Definition of a portfolio of Path properties
- Introduction of a methodology to verify Path properties
- Verification of Path properties on two selected embedded architectures
- Model checker support

[2] Cabodi, G.; Camurati, P.; Finocchiaro, S.F.; Loiacono, C.; Savarese, F.; Vendraminetto, D. (2016) "Secure Path Verification". In: IEEE International Verification and Security Workshop.

Secure path verification

Combine the two approaches together:

- Portfolio of security properties, both Taint (now called State properties: SP) and Path (PP) Properties
- Systematic comparison between SPs and PPs
- Definition and implementation of a verification approach for SPs and PPs within a standard model checker
- Feasibility of the combined approach and comparing SPs and PPs in terms of performance
- Expressivity analysis: when it is better to use one property or the other

[3] Cabodi, Gianpiero, et al. "Embedded systems secure path verification at the hardware/software interface." IEEE Design & Test 34.5 (2017): 38-46.

Speculation based attacks

In 2018 massive security vulnerabilities landed on the computing world: **Meltdown** and **Spectre**

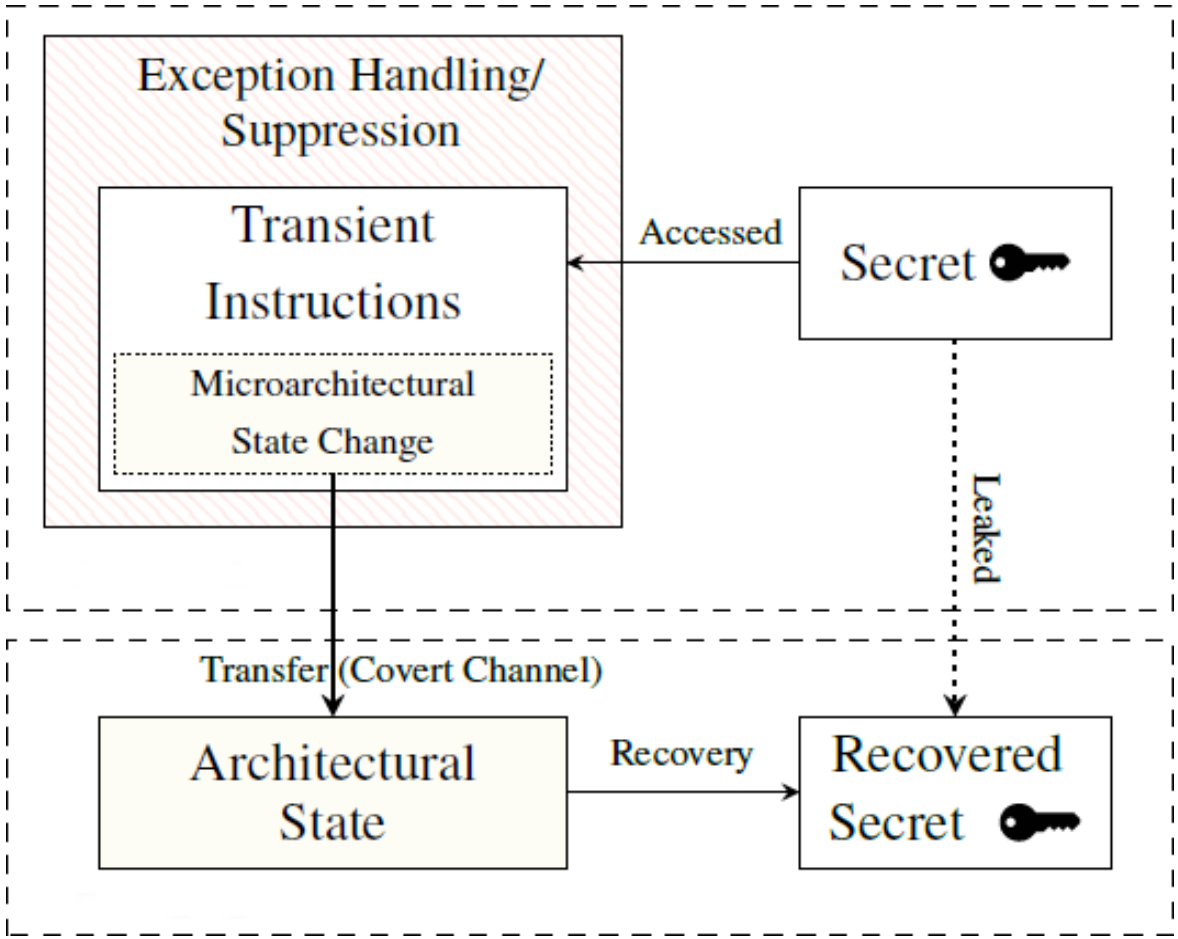
Speculation based attacks

	 Meltdown	 Spectre
Affected CPU Types	Intel, Apple	Intel, Apple, ARM, AMD
Attack Vector	Execute Code on the System	Execute Code on the System
Method	Intel Privilege Escalation & Speculative Execution *(CVE-2017-5754)*	Branch Prediction & Speculative Execution *(CVE-2017-5715 / -5753)*
Exploit Path	Read Kernel Memory from User Space	Read Memory Contents from Other Applications
Remediation	Software Patches	Software Patches

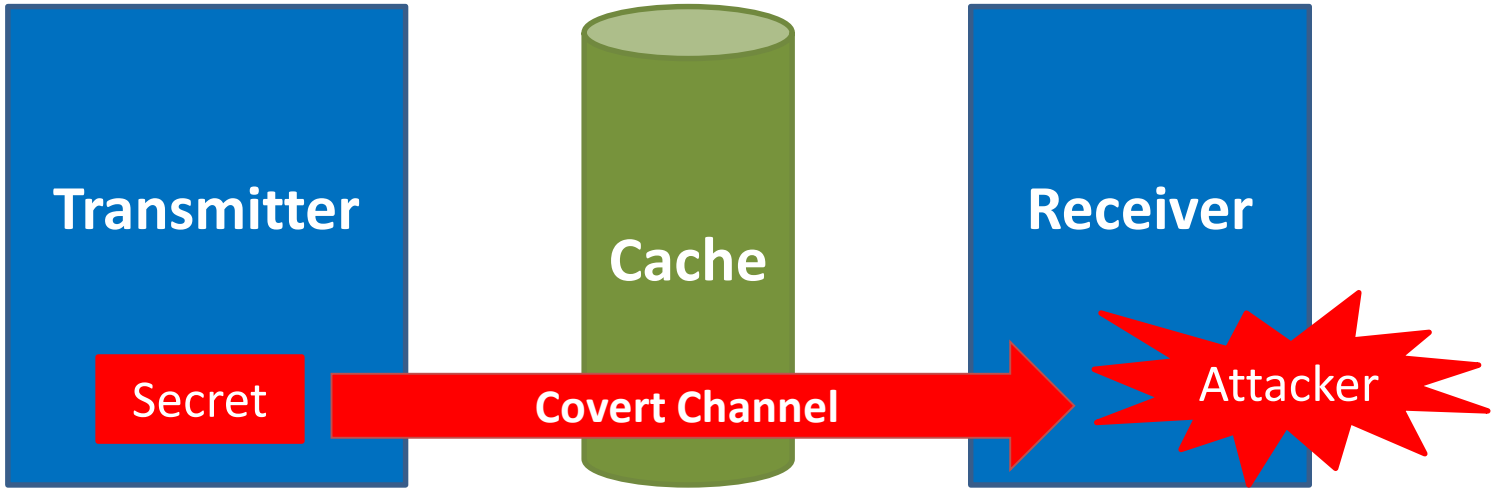
Research questions

- Spectre/Meltdown: groundbreaking attacks!
- How do we detect such attacks?
- Why didn't hardware designers discover such design flaws?
- Is there a verification method that applies to such vulnerabilities?
- Can we prevent these attacks in the future?

Microarchitectural state vs Architectural states



Side channel



Speculative Execution

Side Channel

Normal Execution

Attack description

1. The content of an attacker-chosen memory location, which is directly unreachable by the attacker, is loaded into a register
2. A transient instruction accesses a cache line based on the secret content of the register
3. The attacker uses a covert channel to retrieve the accessed cache line and hence the secret stored at the chosen memory location

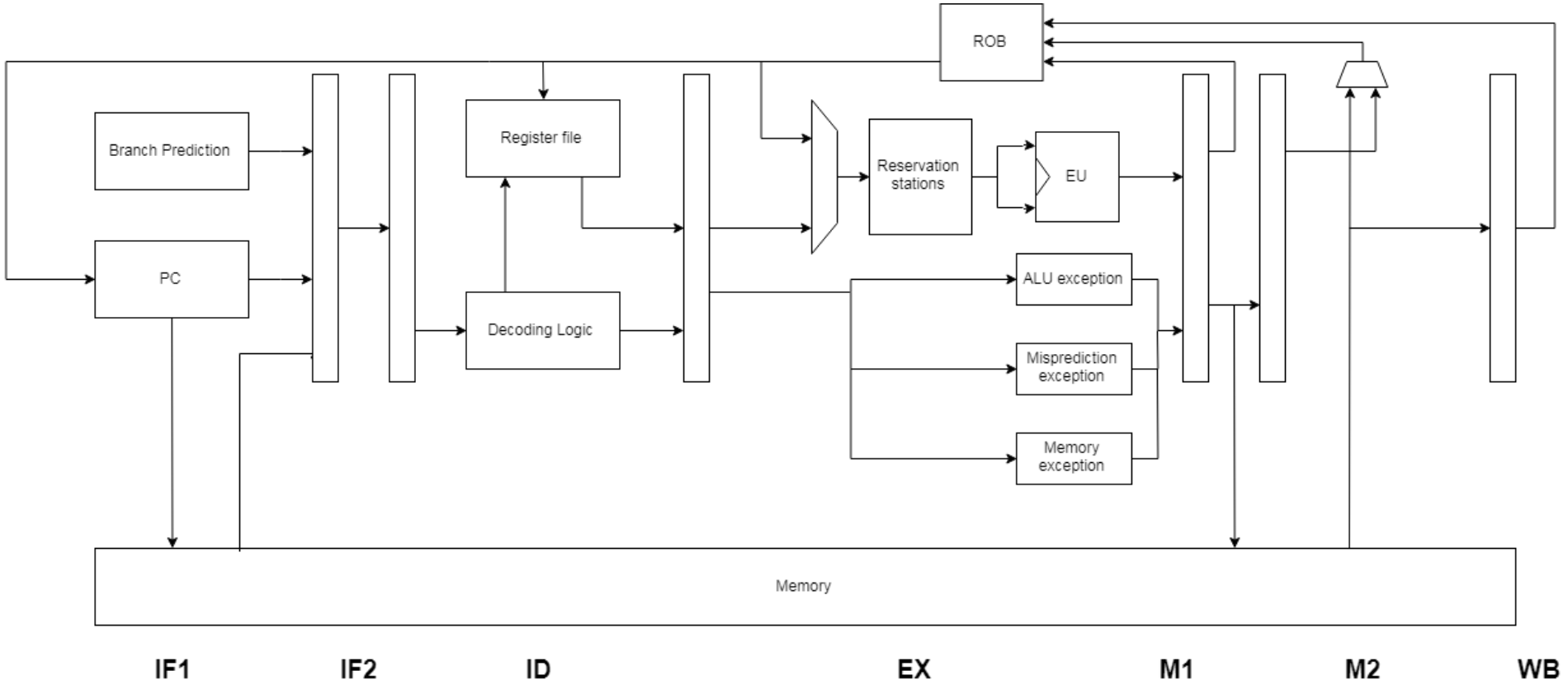
```
1 ; R1 = invalid address
2 ; R3 = probe array
3 LW   R2, 0(R1)     ; illegal  read
4 ADD R1, R2, R3     ; offset = probe_array+secret
5 LW   R1, 0(R1)     ; read from secret-dependent offset
```

Processor model

- Based on the DLX architecture (academic RISC architecture)
- Pipelined (7 stages)
- Speculation-ready
- Out of order execution
  - Reorder buffer
  - Reservation stations

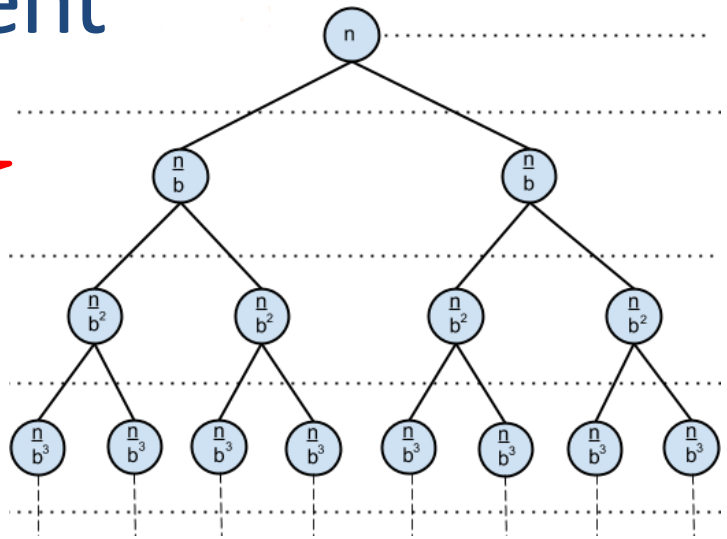
Processore model

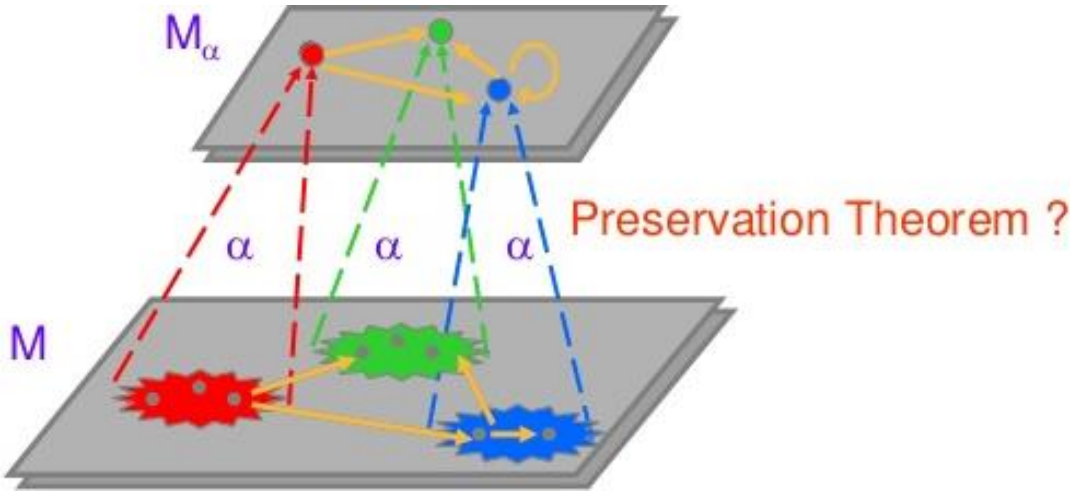
Based on Hennessy's **DLX** RISC processor



Abstraction and refinement

State explosion





Data abstraction

Any data (thus model behaviour) can be over-approximated provided that verification is sound, i.e. an abstract counterexample always implies a concrete one.

- **Bit width reduction:** $V_i \Rightarrow V_i^+$
- Processor logic and arithmetic ⇒ considered correct (i.e. already verified)
- Branch misprediction logic ⇒ replaced by a non-deterministic choice

Data abstraction

Logic is enhanced with tainting: taint value T_i is added

	Concrete model	Abstract model
	V_i	$\{V_i^+, T_i\}$
Data evaluation ⟶	$V_k = OP(V_i, V_j)$	$V_k^+ = OP(V_i^+, V_j^+)$
Taint propagation ⟶	$-$	$T_k = OP^T(V_i^+, T_i, V_j^+, T_j)$

Example:

$$T_k = OP^T(T_i, T_j) = T_i \vee T_j$$

Caveat: transformations must be sound

Data abstraction

The degree of abstraction could be tuned between two corner cases:

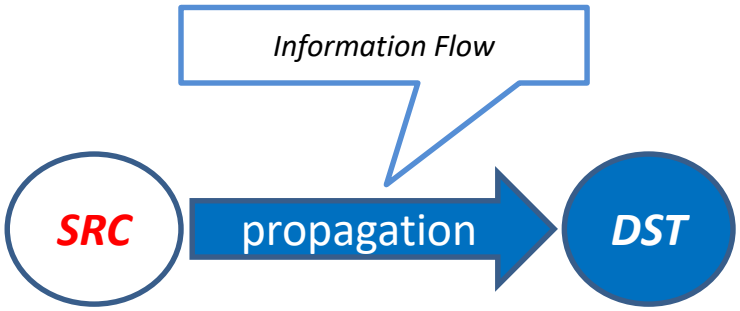
1) Full data dependance: data values are fully involved in taint computation
2) Full data abstraction: taint propagation does not involve data values

EXAMPLES

1	2
- Bitwise OR when V_i^+ is all 1's - Multiplication when V_i^+ = 0 ⇓ Mask the taint on the other operand T_j	The propagation happens whenever at least one of the operands is tainted ⇓ $T_k = OP^T(T_i, T_j) = T_i \vee T_j$

The choice does have a *significant* impact on the soundness of the overall approach

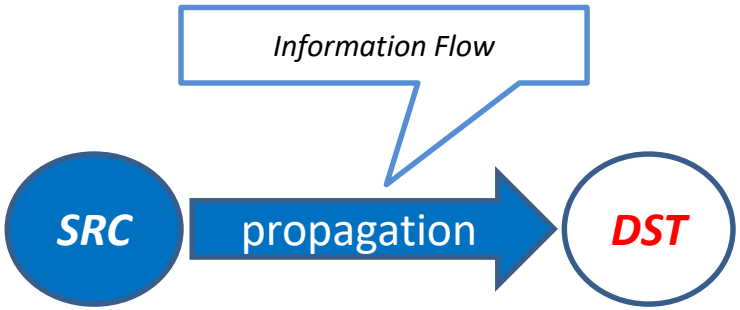
Taint source



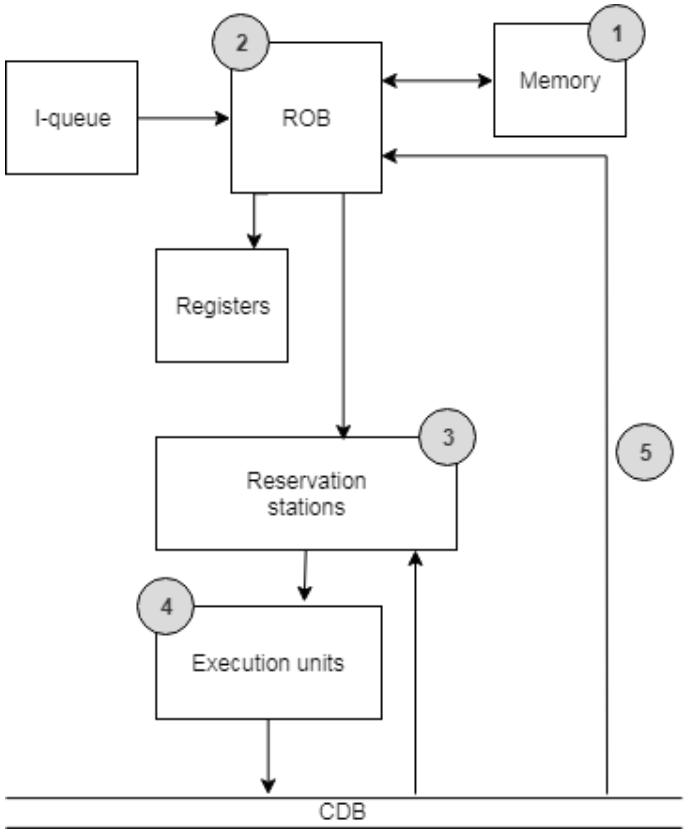
A taint is injected at the memory data input, whenever a protected/invalid address is used.

$$M_{D_{in}} = protected(M_{Addr})\ ?\ TAINT : NOTAINT$$

Taint sink: Property



Target: What do we observe? Memory address logic



$$M_{Addr} == TAINT$$

Correctness

The correctness of our approach is related to the completeness and the soundess of model transformations.*

The **completeness** is guaranteed under two conditions:

- Mimic real sequences of instructions in an out-of-order processor
- The abstract reorder buffer takes into account source-to-sink taint propagation delay

**Burch J.R., Dill D.L., Automatic verification of pipelined microprocessors control. (1994)*

Manolios et al., A complete compositional reasoning framework for the efficient verification of pipelined machines (2005)

Correctness

Model approximation ⇒ False positives

Taint encoding correctness presents two possible choices:

- Adopt a finer grained taint abstraction (refine out all false positive counterexamples)
- Accept false positives and then post-process them: treat them as constraints in a following model checker run.

Completeness is achieved as this approach over-approximates the real behaviour

Experimental results

	Concrete	Abstract
AND gates	120K	3K
Latches	3K	0.1K

BMC run found a **counterexample** of 9 clock steps in less than 1 second!

Conclusions

- The counterexample showed what we expected: a sequence of instructions leaking sensitive data
- Security bug removed (i.e. fixed by a patch) and model checked again: no counterexamples found
- Though preliminary, this approach is feasible as shown by experimental results

[4] Cabodi, G., Camurati, P., Finocchiaro, F., & Vendraminetto, D. (2019, April). "Model Checking Speculation-Dependent Security Properties: Abstracting and Reducing Processor Models for Sound and Complete Verification". In International Conference on Codes, Cryptology, and Information Security (pp. 462-479). Springer, Cham.
[5] Cabodi, G., Camurati, P., Finocchiaro, F., & Vendraminetto, D. "Model Checking Speculation-Dependent Security Properties: Abstracting and Reducing Processor Models for Sound and Complete Verification". In: Electronics 2019, 8 1057- ISSN 2079-9292.

Future work

- Apply this approach to another class of attacks
- Fully or partially automate the abstraction and refinement process
- Proof of completeness and soundness

Thank you!