



12 December 2025 | MoVe Seminar | Paris, France

Verifying Timed Properties of Programs

Using Parametric Time Petri Nets

Dylan Marinho

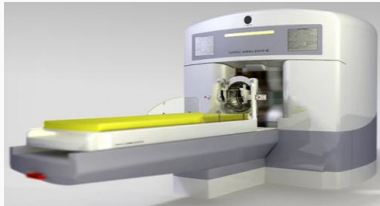
Sorbonne Université, CNRS UMR 7606, LIP6

*Based on joint work with Étienne André, Jean-Luc Béchenec, Sudipta Chattopadhyay, Sébastien Faucou, Didier Lime, Olivier H. Roux,
Jun Sun*

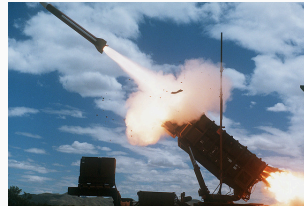


Context: Verifying complex timed systems

- ▶ **Critical** systems: Failures may result in **dramatic** consequences
- ▶ Need for early bug detection
 - ▶ Bugs discovered when final testing: **expensive**
 - ▶ Need for a thorough **specification** and **verification** phase



Therac-25
(USA, 1980s)



MIM-104 Pat. Mis. Fail.
(Iraq, 1991)



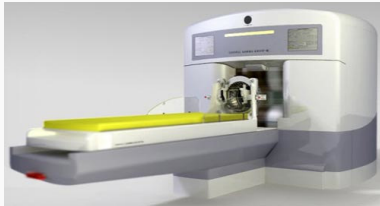
Sleipner A offshore platform
(Norway, 1991)



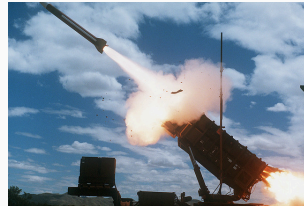
Ariane flight V88
(France, 1996)

Context: Verifying complex timed systems

- ▶ **Critical** systems: Failures may result in **dramatic** consequences
- ▶ Need for early bug detection
 - ▶ Bugs discovered when final testing: **expensive**
 - ▶ Need for a thorough **specification** and **verification** phase



Therac-25
(USA, 1980s)



MIM-104 Pat. Mis. Fail.
(Iraq, 1991)



Sleipner A offshore platform
(Norway, 1991)



Ariane flight V88
(France, 1996)

- ▶ Verification is needed to ensure the absence of bugs

Side-channel attacks



Threats to a system using non-algorithmic weaknesses

- ▶ *e.g., power consumption, electromagnetic radiation, cache usage, **timing**, acoustic emissions, temperature variations, etc.*

¹home.xnet.com/~warinner/pizzacites.html (1990s)

Side-channel attacks



Threats to a system using non-algorithmic weaknesses

▶ e.g., power consumption, electromagnetic radiation, cache usage, **timing**, acoustic emissions, temperature variations, etc.

Example



Number of pizzas (and order time) ordered by the white house prior to major war announcements¹

¹home.xnet.com/~warinner/pizzacites.html (1990s)

Side-channel attacks



Threat

MIDDLE EAST

What is the Pizza Meter? The signal that spiked on Saturday during Iran's attack on Israel

ons,

Example



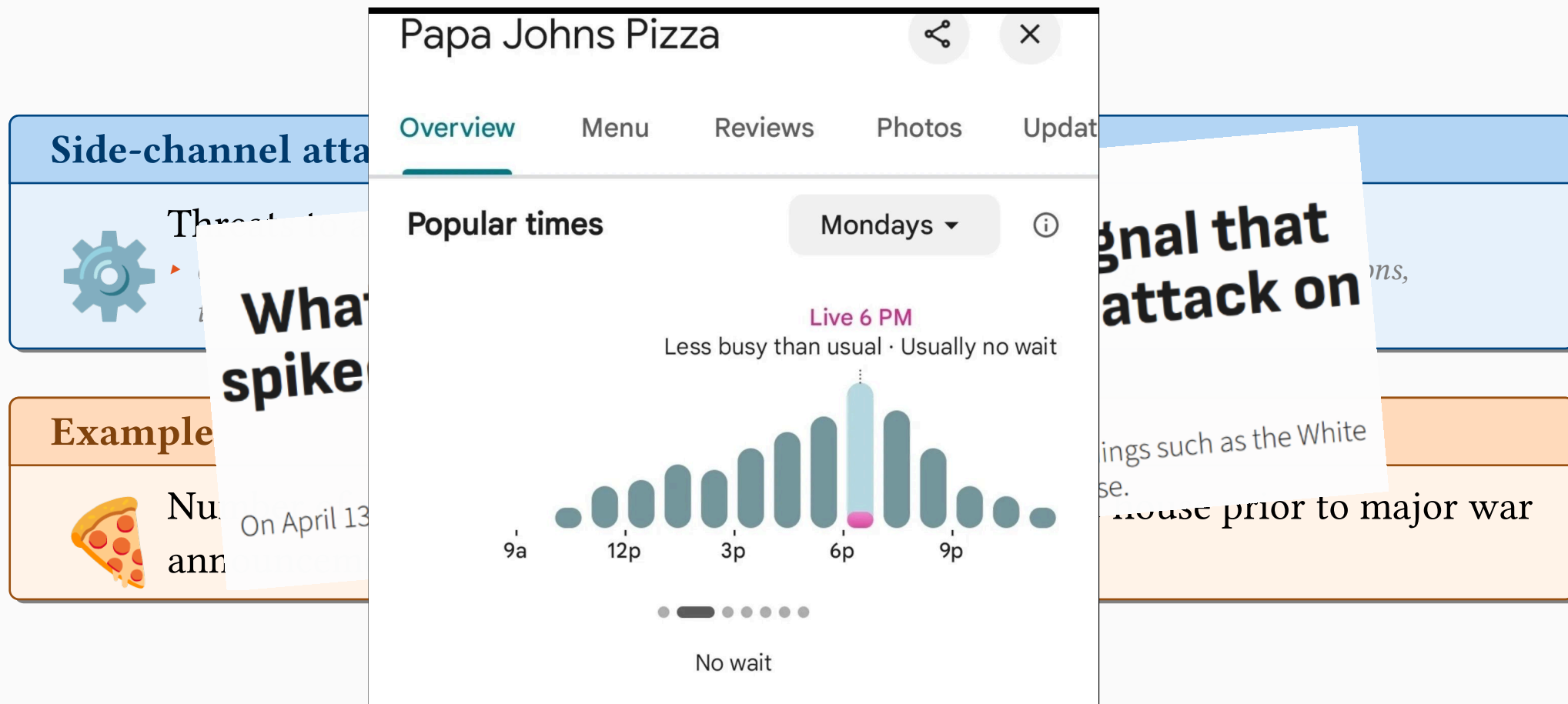
Nu
ann

On April 13 there was a surge in pizza orders from U.S. government buildings such as the White House, the Pentagon, and the Department of Defense.

House prior to major war

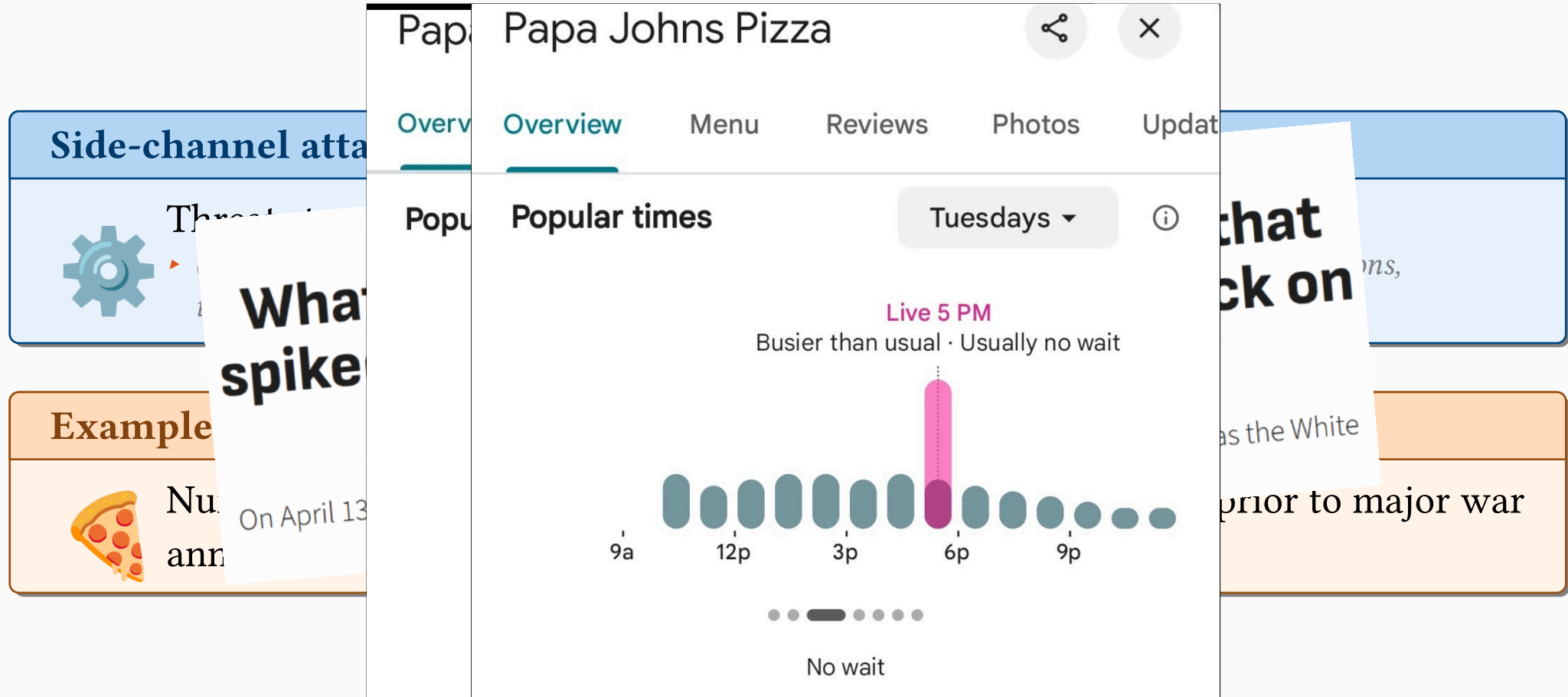
¹home.xnet.com/~warinner/pizzacites.html (1990s)

Context: Side-channel attacks



¹home.xnet.com/~warinner/pizzacites.html (1990s)

Context: Side-channel attacks



¹home.xnet.com/~warinner/pizzacites.html (1990s)

Context: Side-channel attacks

But also in Paris (2024) !

Side-channel att



Threat

What
spike

Example



Nu
ann
On April 1

that
ck on ns,

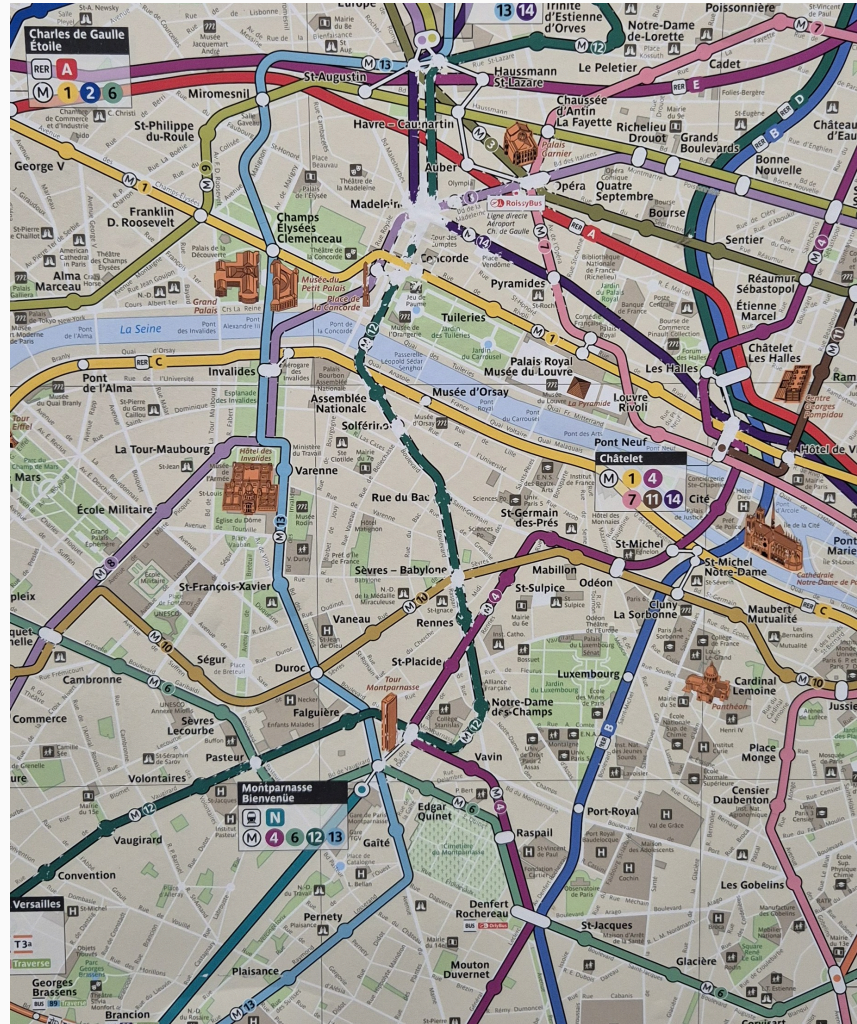
s the White

prior to major war



¹home.xnet.com/~warinner/pizzacites.html (1990s)

Your turn: where was this picture taken?



Context: Timing attacks over programs

```
1 // input pwd      : Real password
2 // input attempt: Tentative password
3 for (i = 0; i < min(len(pwd), len(attempt)); i++) {
4     if(pwd[i] != attempt[i]){
5         return false
6     }
7 }
8 return true
```

pwd

A L M A S T Y

attempt

A L S O C

 Execution time (ET):

Context: Timing attacks over programs

```
1 // input pwd      : Real password
2 // input attempt: Tentative password
3 for (i = 0; i < min(len(pwd), len(attempt)); i++) {
4     if(pwd[i] != attempt[i]){
5         return false
6     }
7 }
8 return true
```



pwd

A

L

M

A

S

T

Y

attempt

A

L

S

O

C



Execution time (ET): ε

Context: Timing attacks over programs

```
1 // input pwd      : Real password
2 // input attempt: Tentative password
3 for (i = 0; i < min(len(pwd), len(attempt)); i++) {
4     if(pwd[i] != attempt[i]){
5         return false
6     }
7 }
8 return true
```



pwd	A	L	M	A	S	T	Y
attempt	A	L	S	O	C		

 Execution time (ET): ε ε

Context: Timing attacks over programs

```
1 // input pwd      : Real password
2 // input attempt: Tentative password
3 for (i = 0; i < min(len(pwd), len(attempt)); i++) {
4     if(pwd[i] != attempt[i]){
5         return false
6     }
7 }
8 return true
```



pwd	A	L	M	A	S	T	Y
attempt	A	L	S	O	C		

 Execution time (ET): ε ε ε

Context: Timing attacks over programs

```
1 // input pwd      : Real password
2 // input attempt: Tentative password
3 for (i = 0; i < min(len(pwd), len(attempt)); i++) {
4     if(pwd[i] != attempt[i]){
5         return false
6     }
7 }
8 return true
```

pwd	A	L	M	A	S	T	Y
attempt	A	L	S	O	C		

 Execution time (ET): $\varepsilon \quad \varepsilon \quad \varepsilon \quad = 3\varepsilon \Rightarrow 2$ correct characters

Problem: The ET is proportional to the **number of consecutive correct** characters from the beginning of attempt

Need to detect **timing-leak** vulnerabilities

- ▶ We want **formal guarantees** → formal methods
 - ▶ Various methods:
 - ▶ Abstract interpretation
 - ▶ Static analysis
 - ▶ Model checking
 - ▶ Theorem proving



Context: Model checking overview

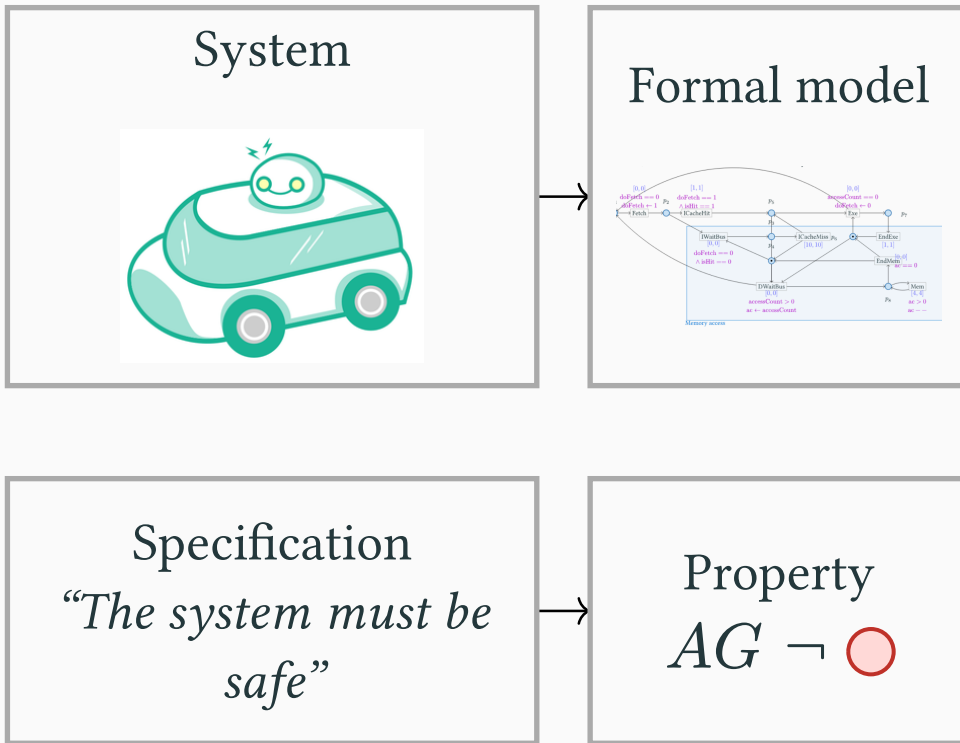
System



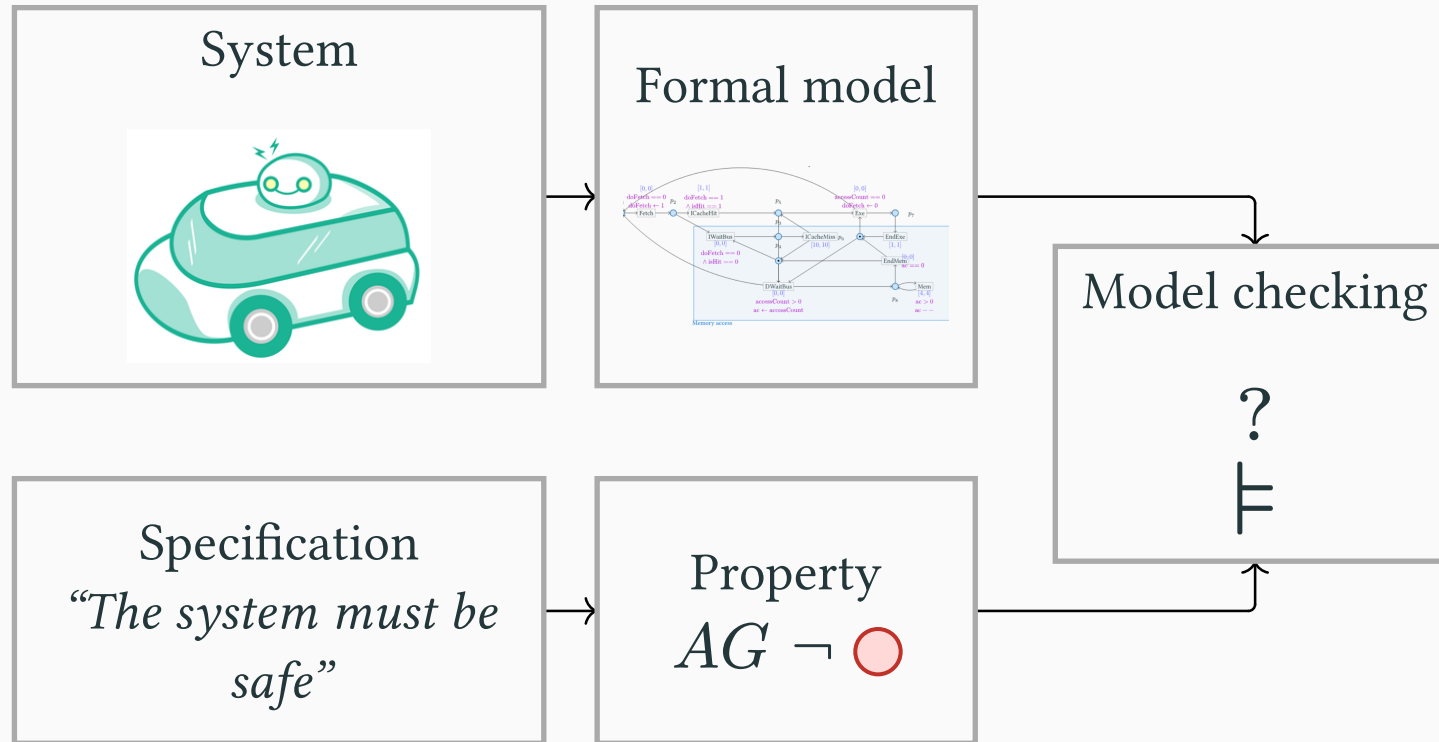
Specification

*“The system must be
safe”*

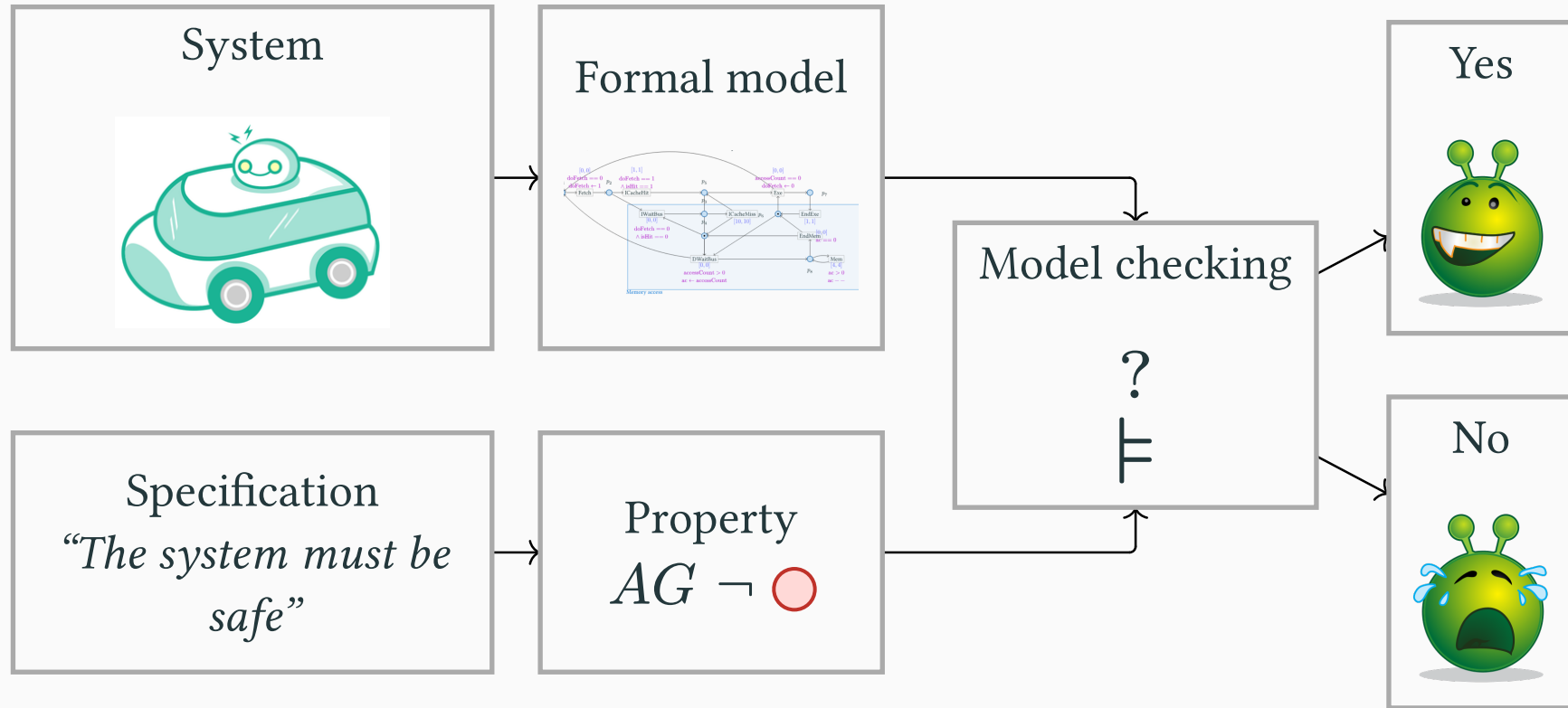
Context: Model checking overview



Context: Model checking overview



Context: Model checking overview



Question: does the model of the system **satisfy** the property?

Timing analysis of programs is **hard**: it depends not just on code, but also on **low-level details** of execution

Impact of hardware



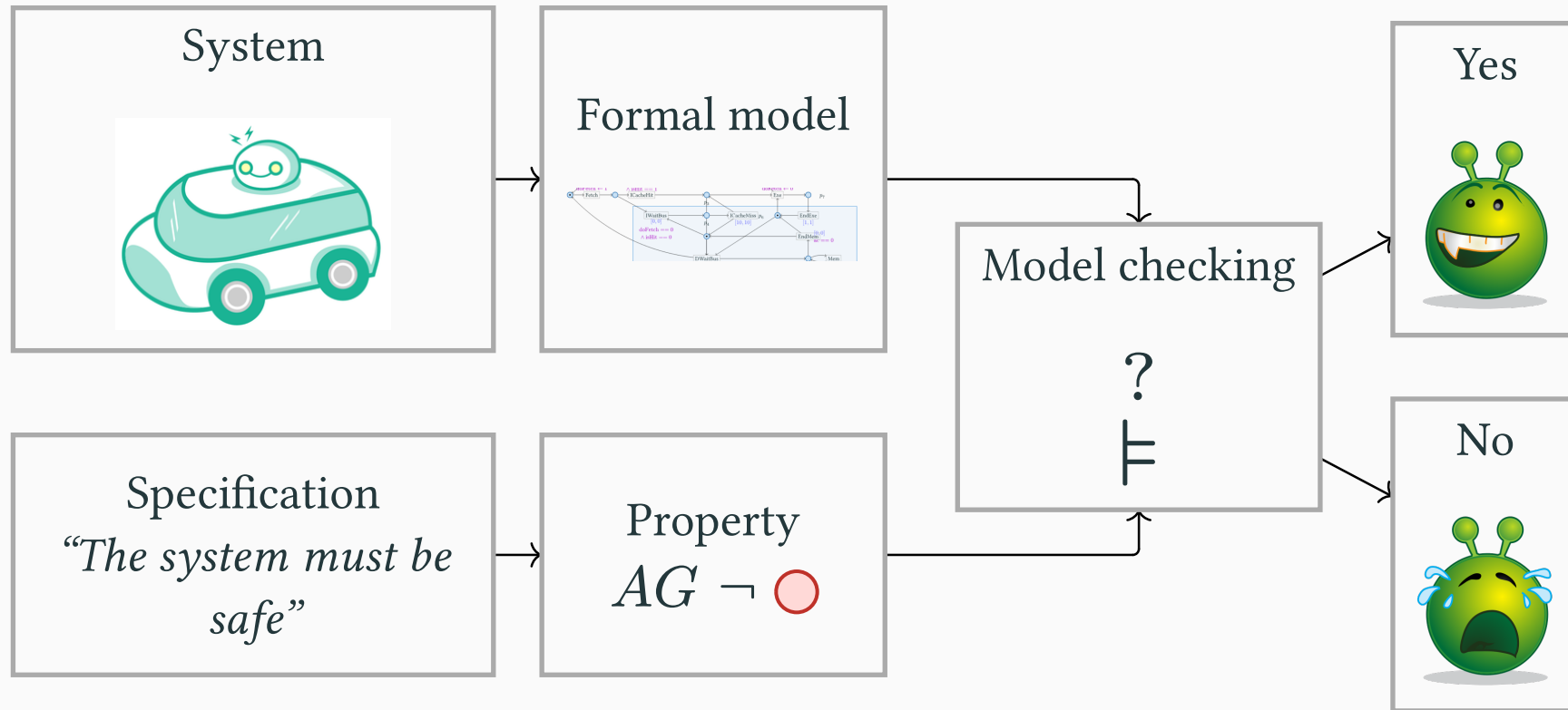
- ▶ ET is heavily influenced by the **micro-architecture**
 - ▶ Especially: pipelines, caches, memory hierarchy

Limitations of existing techniques



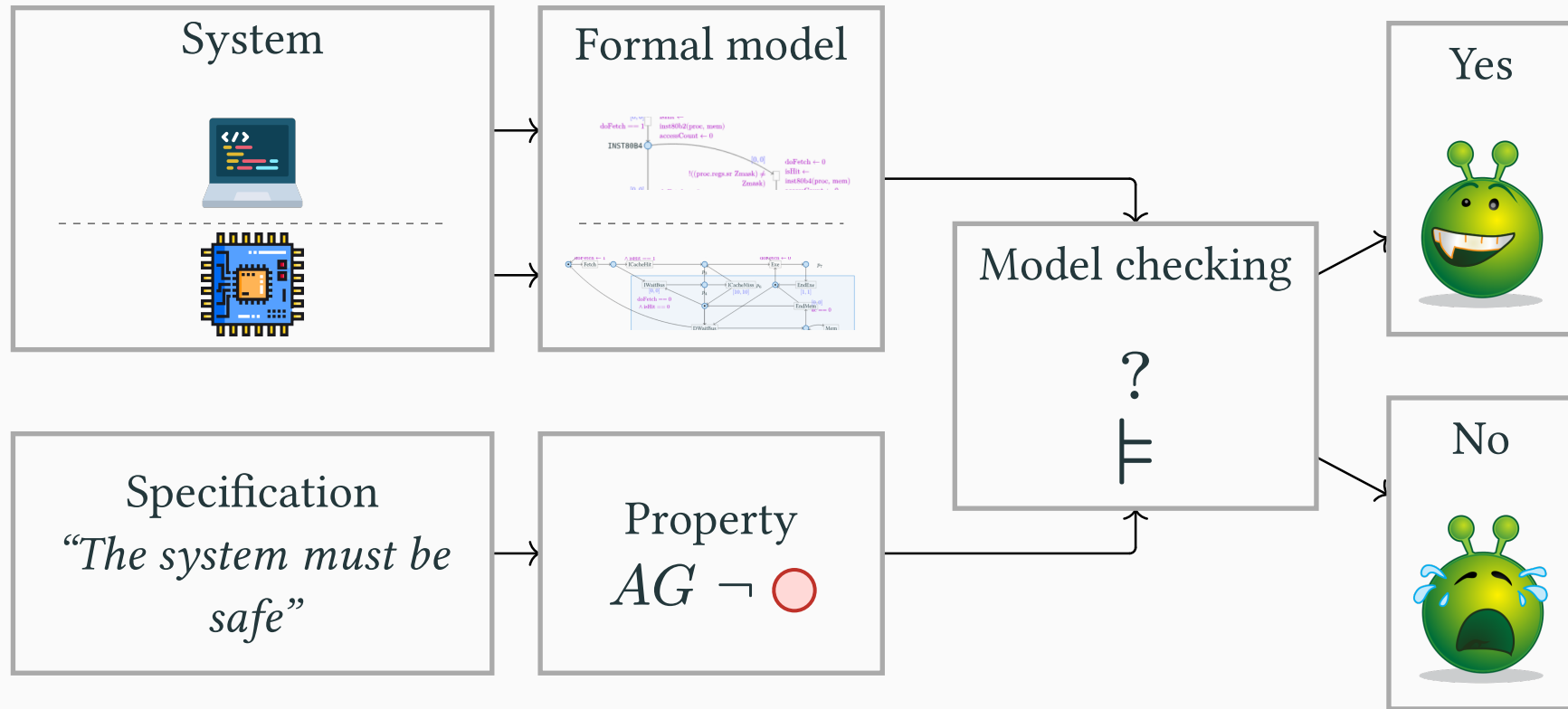
- ▶ Most abstract time away or focus on **coarse properties**
 - ▶ e.g., schedulability analysis, worst-case execution time (WCET)
- ▶ **Insufficient** for fine-grained timing behaviors
 - ▶ e.g., detecting or mitigating *timed side-channels*

Model checking



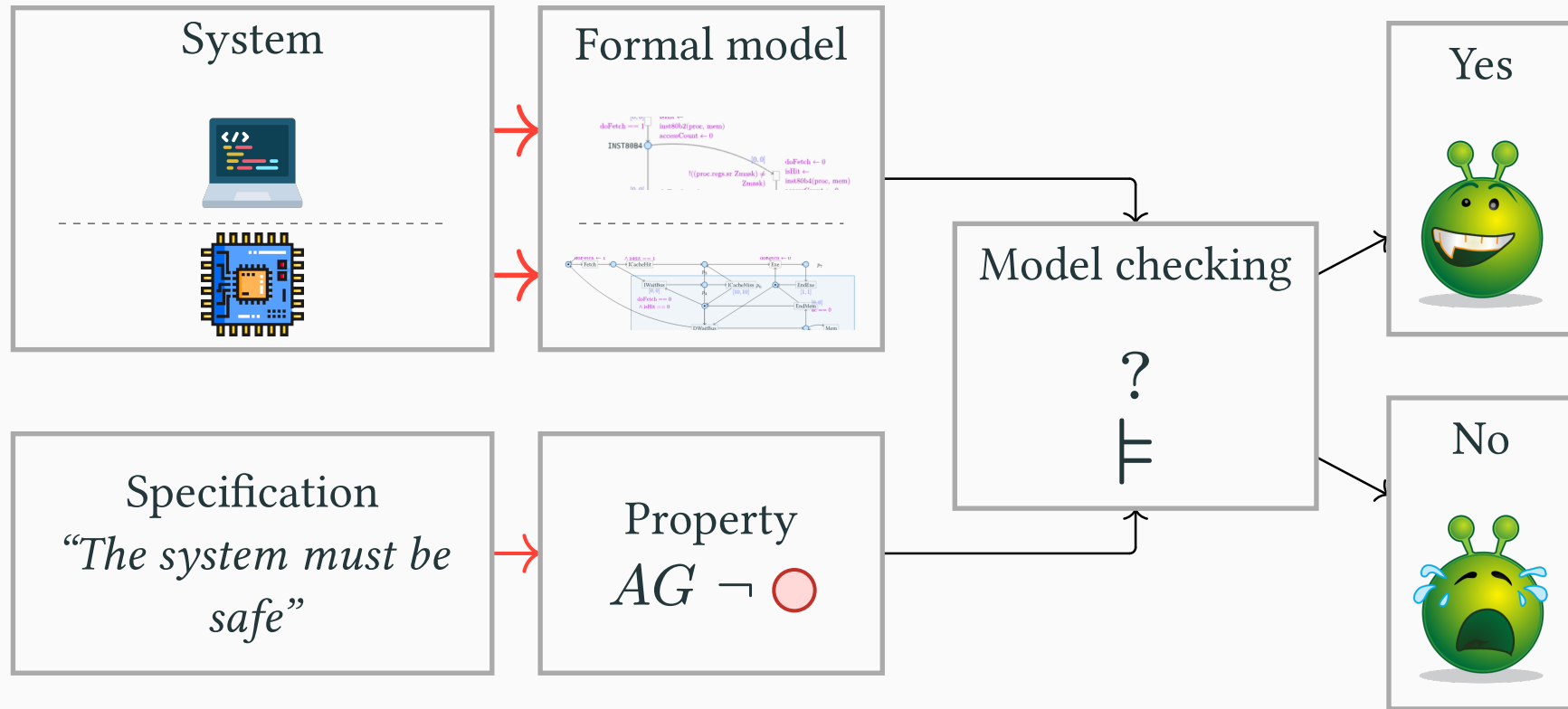
Question: does the model of the system **satisfy** the property?

Model checking



Question: does the model of the system **satisfy** the property?

Model checking



Question: does the model of the system **satisfy** the property?

[And+25]

⚙ A **modular** and **automated** approach to build formal models to analyze **timing behaviors**

- *binary code with the hardware*

[And+25] Étienne André *et al.*, “Verifying Timed Properties of Programs in IoT nodes using Parametric Time Petri Nets,” in *SAC 2025*, 2025.

 A **modular** and **automated** approach to build formal models to analyze **timing behaviors**

- *binary code with the hardware*

 An **implementation**

- *targeting a realistic micro-architecture of a simple micro-controller*
- *producing **time Petri nets** models*

 A **modular** and **automated** approach to build formal models to analyze **timing behaviors**

- *binary code with the hardware*

 An **implementation**

- *targeting a realistic micro-architecture of a simple micro-controller*
- *producing **time Petri nets** models*

 An application to **timing attacks** in **C programs** using the **Roméo** model checker

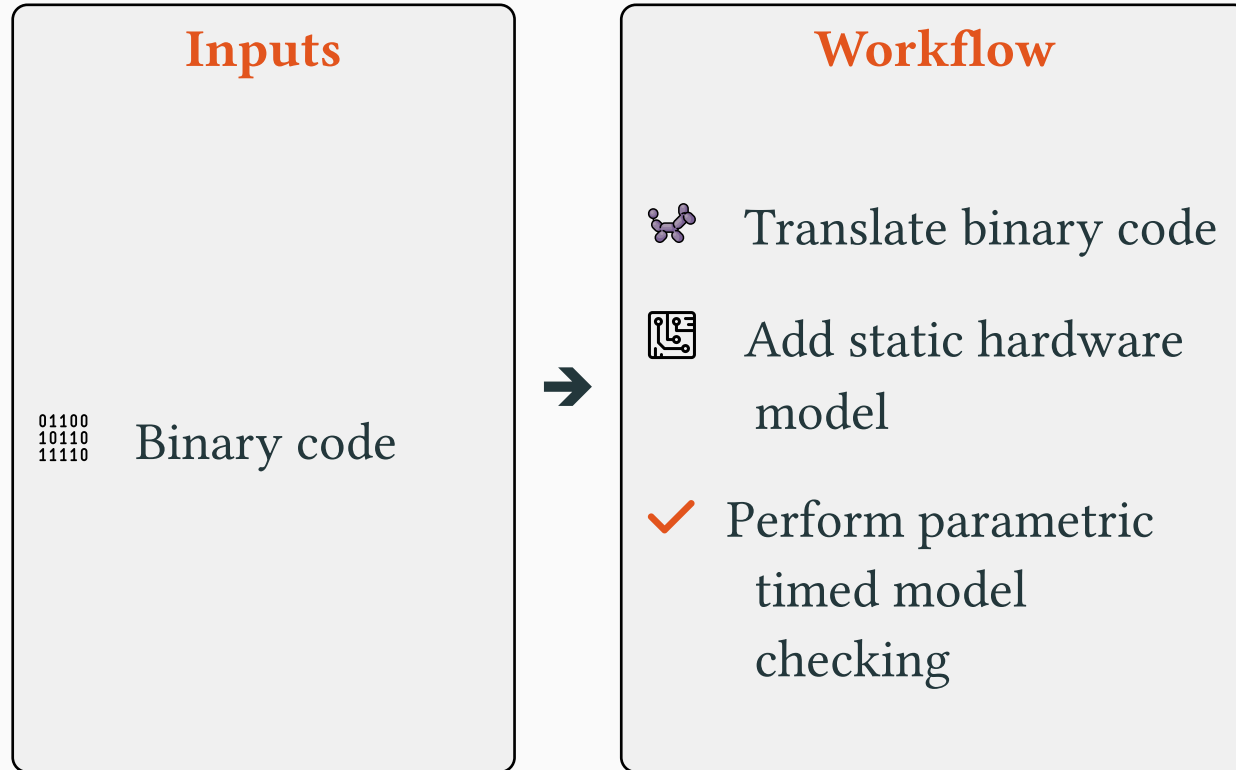
Methodology

Inputs

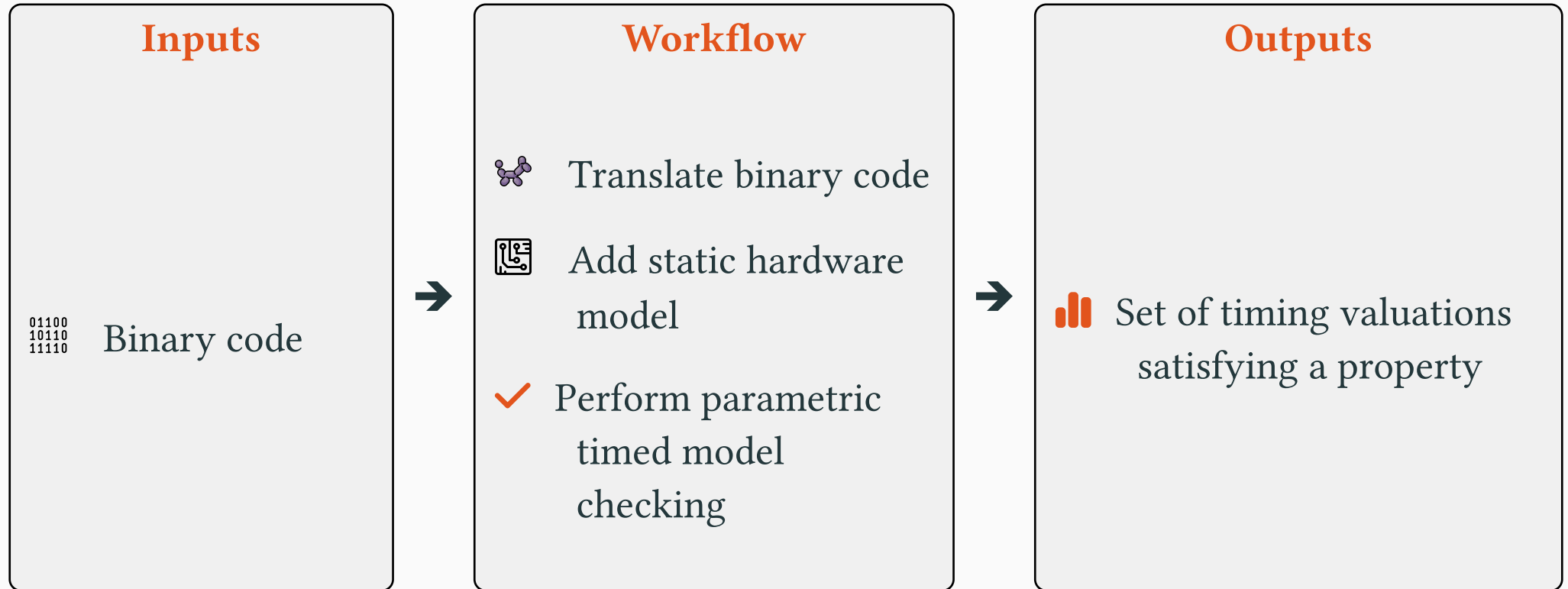
01100
10110
11110

Binary code

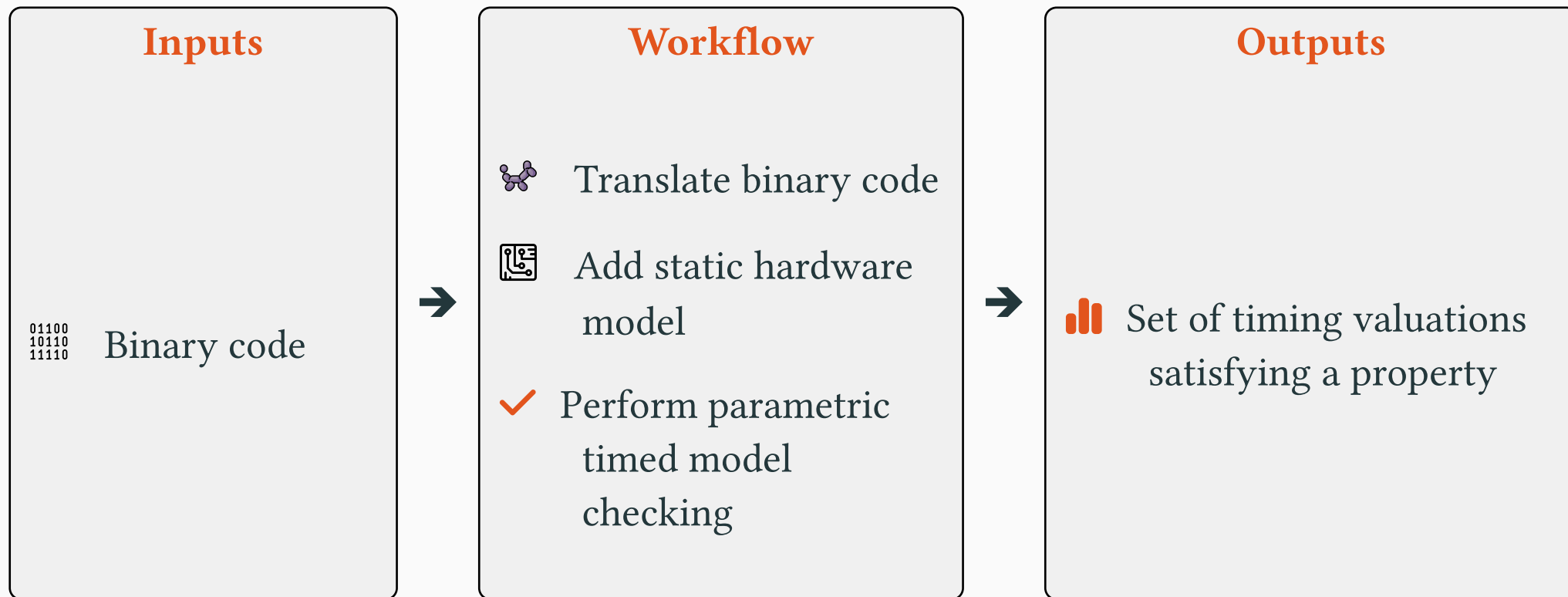
Overall methodology



Overall methodology



Overall methodology



- ▶ *e.g., possible execution times*
- ▶ *application: password leak detection*

Parametric time Petri nets with variables

Extension of **Petri nets** with



[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, “Parametric Model-Checking of Stopwatch Petri Nets,” *Journal of Universal Computer Science*, 2009.

Parametric time Petri nets with variables

Extension of **Petri nets** with

- firing times
- timing parameters



[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, “Parametric Model-Checking of Stopwatch Petri Nets,” *Journal of Universal Computer Science*, 2009.

Parametric time Petri nets with variables

- Extension of **Petri nets** with
- ▶ firing times
 - ▶ timing parameters
 - ▶ integer-valued variables (with guards and updates)

[TLR09]



[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, “Parametric Model-Checking of Stopwatch Petri Nets,” *Journal of Universal Computer Science*, 2009.

Our hardware



- ▶ Model of the **processor architecture**
 - ▶ *relatively simple micro-architecture similar to ARM Cortex M0+ core, with a 2-stage pipeline (Fetch and Execute)*
- ▶ Model of the **instruction set architecture (ISA)**
 - ▶ *ARMv6-M ISA*

Features

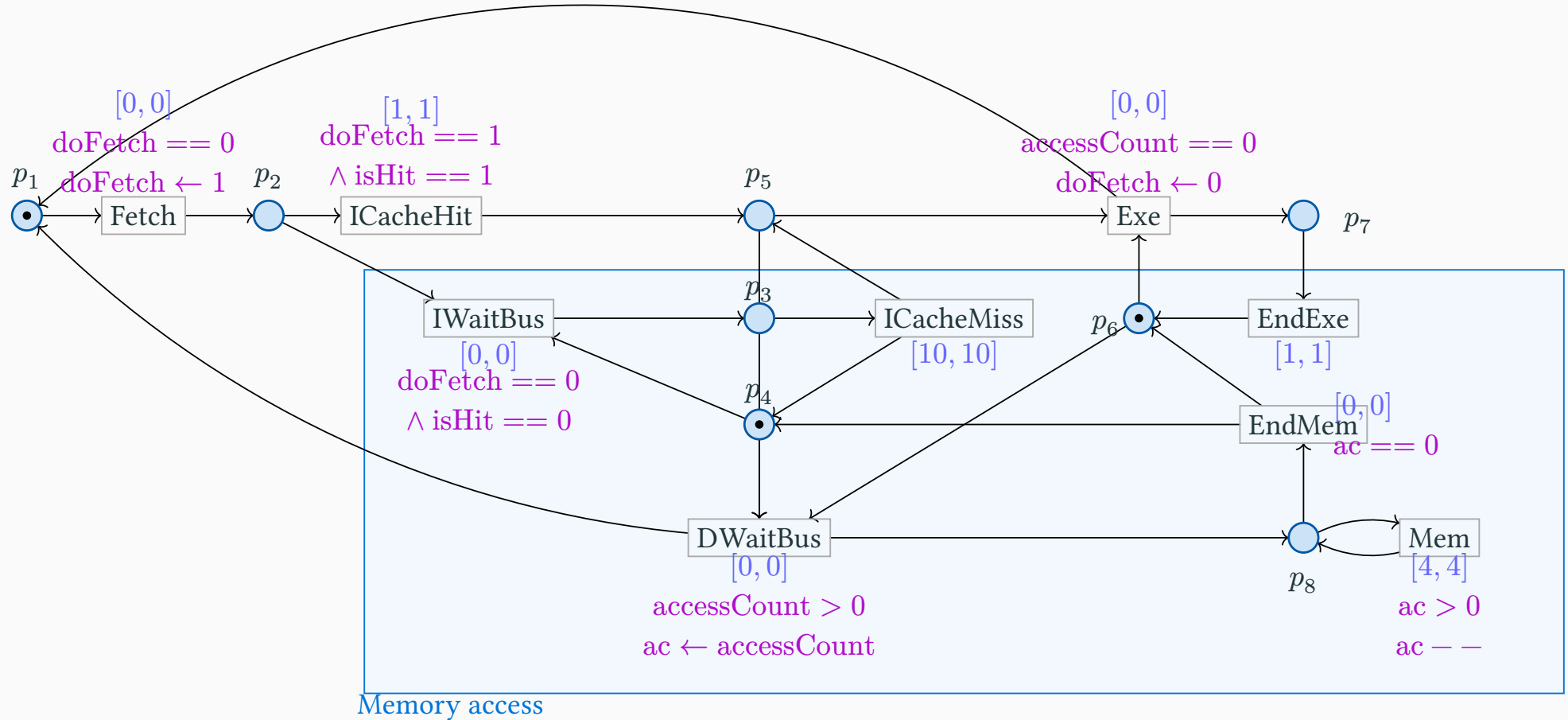


- ▶ Execution pipeline of the processor
- ▶ Unique memory space
 - ▶ *(instructions and data)*
- ▶ Bus between the processor and memory
- ▶ Direct-mapped **instruction cache**
 - ▶ *with 16 lines of 32 bytes*
 - ▶ *no actual instructions, but only information about their presence*
- ▶ **No data cache**

-
- ▶ Among the limitations: no switch/case, function pointers. . .

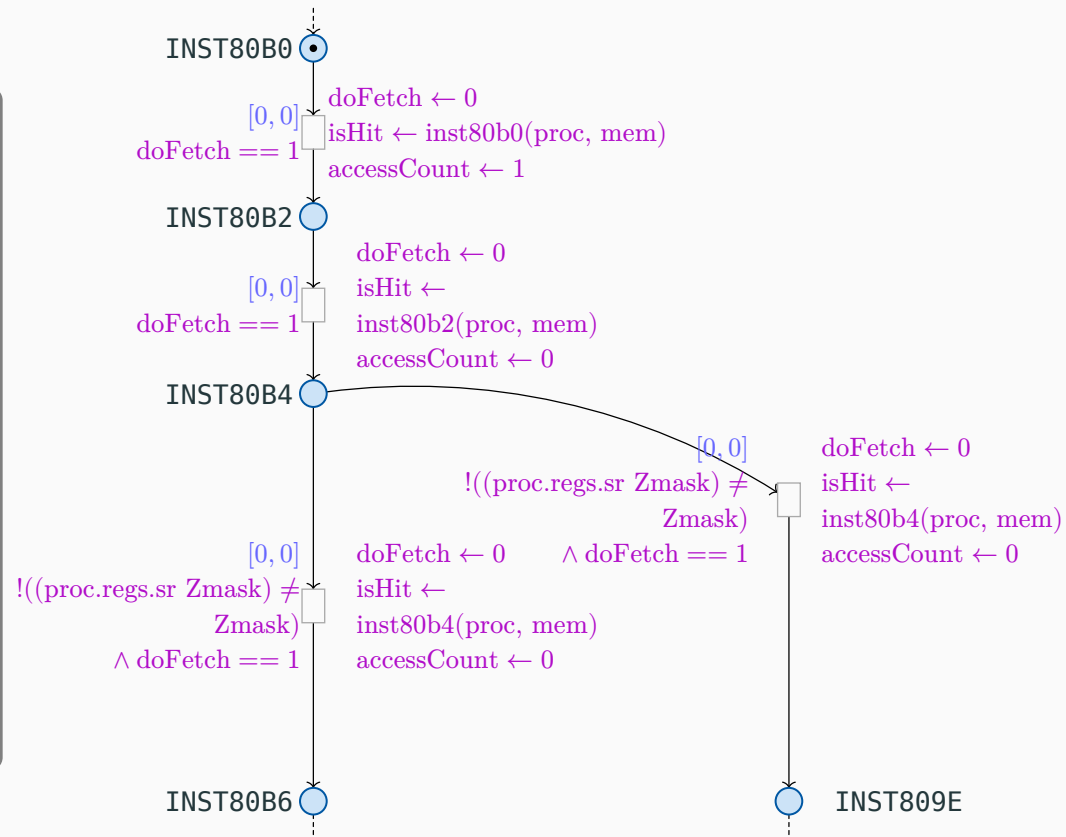
PTPN hardware model

- ▶ **doFetch**, **isHit** and **accessCount**: variables used to synchronize with the software



Software

- ▶ Captures the **binary code** of the program (*ARMv6-M*)
 - ▶ Firing a transition corresponds to executing the instruction
 - ▶ Pipeline fetch: **doFetch**
 - ▶ Memory access: **accessCount** and **isHit**
- ▶ Structurally identical to the **control flow graph**

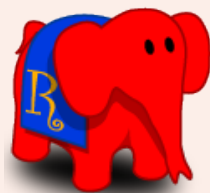


A fully automated translation



- ▶ Including the hardware and software models
- ▶ Written in  and 
- ▶ All the way from the  source code to the PTPN model
- ▶  Entirely open source (github.com/DylanMarinho/codeToPN/)

Target model checker: ROMÉO [Lim+09]



- ▶ **Parametric timed model checker** supporting (extensions) of PTPNs
- ▶ Including **C-like code** to be executed during transitions

[Lim+09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez, “Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches,” in *TACAS 2009*, 2009.

Application to security properties

Timing attacks



- ▶ Attacker can **infer information** about the secret key by measuring the **execution time** of the program
 - ▶ e.g., *password checking program*

Execution-time opacity [And+23]



“Can the attacker deduce internal behavior by only observing the execution time?”

[And+23] Étienne André, Engel Lefauchaux, Didier Lime, Dylan Marinho, and Jun Sun, “Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata,” in *TiCSA@ETAPS 2023*, 2023.

Timing attacks



- ▶ Attacker can **infer information** about the secret key by measuring the **execution time** of the program
 - ▶ e.g., *password checking program*

Execution-time opacity [And+23]



“Can the attacker deduce internal behavior by only observing the execution time?”

- ▶ Use of **timing parameters**: to measure execution times

[And+23] Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun, “Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata,” in *TiCSA@ETAPS 2023*, 2023.

Which of the following two programs is not secure?

Program 1

```
1  int main() {  
2      int i; int length = 10;  
3      char ca[11] = "patehenaff";  
4      char cb[11] = "pasta";  
5  
6      for (i = 0; i < length; i++){  
7          if (ca[i] != cb[i]) {  
8              return 0; // false  
9          }  
10     }  
11     return 1; // true  
12 }
```

Program 2

```
1  int main() {  
2      int i; int length = 10;  
3      char ca[11] = "patehenaff";  
4      char cb[11] = "pasta";  
5  
6      int result = 1; // true  
7  
8      for (i = 0; i < length; i++){  
9          result &= (ca[i] == cb[i]);  
10     }  
11     return result;  
12 }
```

Which of the following two programs is not secure?

Program 1

```
1  int main() {  
2      int i; int length = 10;  
3      char ca[11] = "patehenaff";  
4      char cb[11] = "pasta";  
5  
6      for (i = 0; i < length; i++){  
7          if (ca[i] != cb[i]) {  
8              return 0; // false  
9          }  
10     }  
11     return 1; // true  
12 }
```

Unsecure

Program 2

```
1  int main() {  
2      int i; int length = 10;  
3      char ca[11] = "patehenaff";  
4      char cb[11] = "pasta";  
5  
6      int result = 1; // true  
7  
8      for (i = 0; i < length; i++){  
9          result &= (ca[i] == cb[i]);  
10     }  
11     return result;  
12 }
```

Secure

Which of the following two programs is not secure?

Program 1

```
1  int main() {  
2      int i; int length = 10;  
3      char ca[11] = "patehenaff";  
4      char cb[11] = "pasta";  
5  
6      for (i = 0; i < length; i++){  
7          if (ca[i] != cb[i]) {  
8              return 0; // false  
9          }  
10     }  
11     return 1; // true  
12 }
```

Unsecure - ET sensitive

- ▶ 758 for the secret password
- ▶ {362, 404, 446, 488, 530, 572, 614, 656, 698, 740} for any other password

Program 2

```
1  int main() {  
2      int i; int length = 10;  
3      char ca[11] = "patehenaff";  
4      char cb[11] = "pasta";  
5  
6      int result = 1; // true  
7  
8      for (i = 0; i < length; i++){  
9          result &= (ca[i] == cb[i]);  
10     }  
11     return result;  
12 }
```

Secure - Constant ET: 876

Is this third program secure?

Program 3

```
1  int main () {  
2      int i ;  
3      int length = 10; // length of the strings  
4      char ca[11] = " patehenaff " ;  
5      char cb[11] = " pasta " ;  
6  
7      int result = 1; // true  
8      for (i=0; i<length ; i++) {  
9          if (ca[i] == cb[i]) {  
10             result &= 1;  
11         } else {  
12             result &= 0;  
13         }  
14     }  
15     return result ;  
16 }
```

Is this third program secure?

Program 3

```
1  int main () {
2      int i ;
3      int length = 10; // length of the strings
4      char ca[11] = " patehenaff " ;
5      char cb[11] = " pasta " ;
6
7      int result = 1; // true
8      for (i=0; i<length ; i++) {
9          if (ca[i] == cb[i]) {
10             result &= 1;
11         } else {
12             result &= 0;
13         }
14     }
15     return result ;
16 }
```

- ▶ It seems so: very close to the former secure program
- ▶ But it is not due to the **instruction cache**
 - ▶ 876 for the secret password
 - ▶ {816, 822, 828, 834, 840, 846, 852, 858, 864, 870} for any other password

Is this third program secure?

Program 3

```
1  int main () {
2      int i ;
3      int length = 10; // length of the strings
4      char ca[11] = " patehenaff " ;
5      char cb[11] = " pasta " ;
6
7      int result = 1; // true
8      for (i=0; i<length ; i++) {
9          if (ca[i] == cb[i]) {
10             result &= 1;
11         } else {
12             result &= 0;
13         }
14     }
15     return result ;
16 }
```

- ▶ It seems so: very close to the former secure program
- ▶ But it is not due to the **instruction cache**
 - ▶ 876 for the secret password
 - ▶ {816, 822, 828, 834, 840, 846, 852, 858, 864, 870} for any other password



We can **reconfigure** the program, by **making it opaque**

- ▶ *adding 6 nop instructions at the end of one branch*
- ▶ *(see paper)*

Conclusion and perspectives

End-to-end approach on **binary code timing analysis**, subject to micro-architectural constraints



- ▶ automated production of **timed formal models** of both the program and the hardware architecture
- ▶ using (parametric) time Petri nets

Illustrative case-study: **detection of timing leaks** in  programs



- ▶ via parameter synthesis techniques using **Roméo**
- ▶ (manual) **reconfiguration** of the program to make it opaque



- ▶ Modeling and analysis of programs on **multicore** architectures
- ▶ Automatic modification of a program to make it **opaque**



- ▶ Handling **more complex attacks**
 - ▶ Fault-injection
 - ▶ Cache side-channels
 - ▶ *flush and reload, prime and probe*
 - ▶ Energy-based attacks



- ▶ Formal proof of our translation?



12 December 2025 | MoVe Seminar | Paris, France

Verifying Timed Properties of Programs

Using Parametric Time Petri Nets

Dylan Marinho

Sorbonne Université, CNRS UMR 7606, LIP6

*Based on joint work with Étienne André, Jean-Luc Béchenec, Sudipta Chattopadhyay, Sébastien Faucou, Didier Lime, Olivier H. Roux,
Jun Sun*

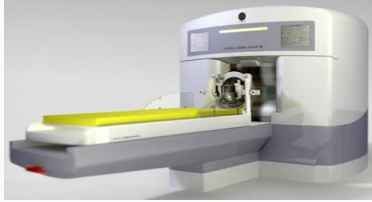


Bibliography

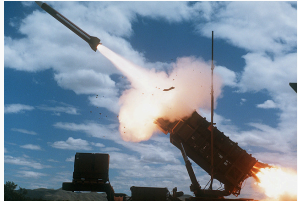
- [And+25] Étienne André *et al.*, “Verifying Timed Properties of Programs in IoT nodes using Parametric Time Petri Nets,” in *SAC 2025*, 2025.
- [TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, “Parametric Model-Checking of Stopwatch Petri Nets,” *Journal of Universal Computer Science*, 2009.
- [Lim+09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez, “Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches,” in *TACAS 2009*, 2009.
- [And+23] Étienne André, Engel Lefauchaux, Didier Lime, Dylan Marinho, and Jun Sun, “Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata,” in *TiCSA@ETAPS 2023*, 2023.

Additional information

Explanation of the pictures



- ▶ Therac-25 bug
- ▶ Computer bug, race condition
- ▶ Consequences: multiple fatalities



- ▶ Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
- ▶ 28 fatalities, hundreds of injured
- ▶ Computer bug: software error (clock drift)
- ▶ (Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

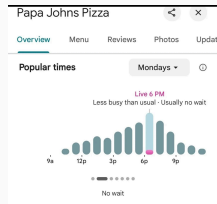


- ▶ Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
- ▶ No fatalities
- ▶ Computer bug: inaccurate finite element analysis modeling
- ▶ (Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Explanation of the pictures

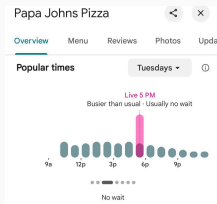


- ▶ Ariane flight V88 (France, 1996)
- ▶ Computer bug (notably integer overflow)
- ▶ Consequences: US\$370 million



- ▶ USA, June 2025
- ▶ Empty bars during Iran's riposte against US military bases.

(Dr. Dominic Ng)



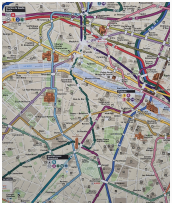
- ▶ USA, 24 June 2025
- ▶ After the Israel-Iran ceasefire

(Dr. Dominic Ng)

Explanation of the pictures



- ▶ Prefecture de Police, Paris (France, 17th July 2024 at 10:10 PM)
- ▶ Delivers in front of the Prefecture de Police, Paris
- ▶ The day before closing the center of Paris to prepare the 2024 Olympic Games



- ▶ Paris Metro map, Madeleine station (Paris, France)

Licensing

Sources of the graphics



- ▶ Title : Explosion of first Ariane 5 flight, June 4, 1996
- ▶ Author : ESA
- ▶ Source : https://www.esa.int/ESA_Multimedia/Images/2009/09/Explosion_of_first_Ariane_5_flight_
- ▶ License : ESA Standard Licence

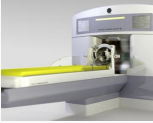


- ▶ Title : Deepwater Horizon Offshore Drilling Platform on Fire
- ▶ Author : ideum
- ▶ Source : <https://secure.flickr.com/photos/ideum/4711481781/>
- ▶ License : Creative Commons cc-by-sa



- ▶ Title : DA-SC-88-01663
- ▶ Author : incomkorea
- ▶ Source : <https://secure.flickr.com/photos/incomkorea/3017886760>
- ▶ License : Creative Commons cc-by-nc-nd

Sources of the graphics



- ▶ Title : Therac-25
- ▶ Author : ?
- ▶ Source : <https://arquivonuclear.blogspot.com/2011/03/therac-25.html>
- ▶ License : unknown

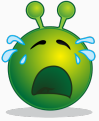


- ▶ Title : Autonomous robot vehicle or ADV typically used for food or grocery delivery
- ▶ Author : Rlistmedia
- ▶ Source : https://commons.wikimedia.org/wiki/File:Autonomous_delivery_robot_vehicles_ADV.png
- ▶ License : Creative Commons cc-by



- ▶ Title : Smiley green alien big eyes (aaah)
- ▶ Author : LadyofHats
- ▶ Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
- ▶ License : Public domain

Sources of the graphics



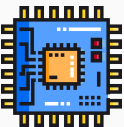
- ▶ Title : Smiley green alien big eyes (cry)
- ▶ Author : LadyofHats
- ▶ Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
- ▶ License : Public domain



- ▶ Title : Smiley green alien exterminate
- ▶ Author : LadyofHats
- ▶ Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_exterminate.svg
- ▶ License : Public domain



- ▶ Source: Flaticon.com



- ▶ Source: Flaticon.com

License of this document

This presentation can be published, reused and modified under the terms of the license
Creative Commons **Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**



creativecommons.org/licenses/by-nc-sa/4.0/

Authors: Étienne André, **Dylan Marinho**