Wasm-WCET

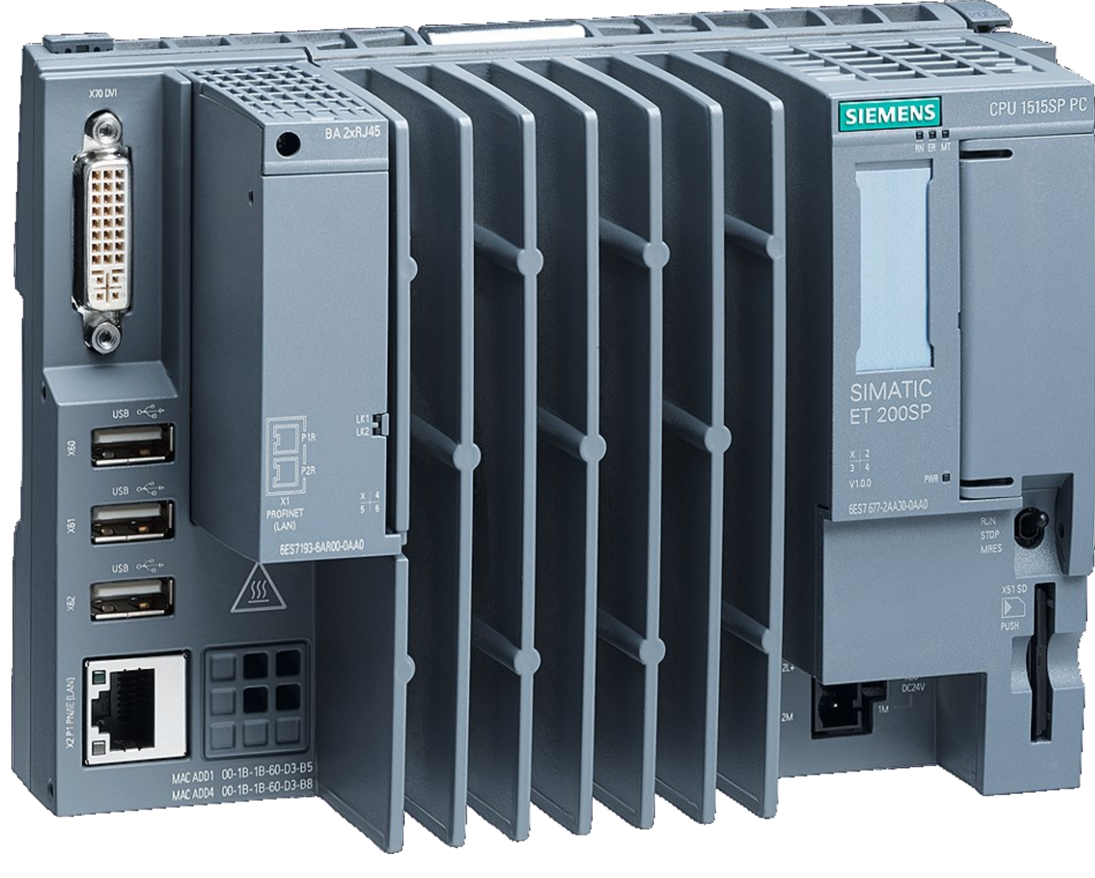
Worst-Case Execution-Time Analysis of WebAssembly Modules on Updatable Resource-Constrained Embedded Devices

Maximilian Seidler[1,2], Martin Michelis[1], Peter Wägemann[1], Rüdiger Kapitza[1]

[1]Friedrich-Alexander-Universität Erlangen-Nürnberg
[2]Siemens AG, Research and Predevelopment

 Public | © Siemens 2026

Introduction: Programmable Logic Controllers (PLCs)



 Public | © Siemens 2026

Introduction: PLCs

Firmware/OS

Customized by Operators

Firmware/OS

Driver Driver Driver

Produced by OEMs

OEM: Original Equipment Manufacturer
OS: Operating System
PLC: Programmable Logic Controller

 Public | © Siemens 2026

Introduction: PLCs

Prog | Prog | Prog

Firmware/OS

Hardware

1010
1010

Compiler

Customized by operators

Provided by OEMs

OEM: Original Equipment Manufacturer
OS: Operating System
PLC: Programmable Logic Controller

Public | © Siemens 2026

Introduction: PLCs

No interference

Wasm application

Customized by operators

App 1

App 2

App 3

WA

Compiler

Runtime

Firmware/OS

Hardware

Provided by OEMs

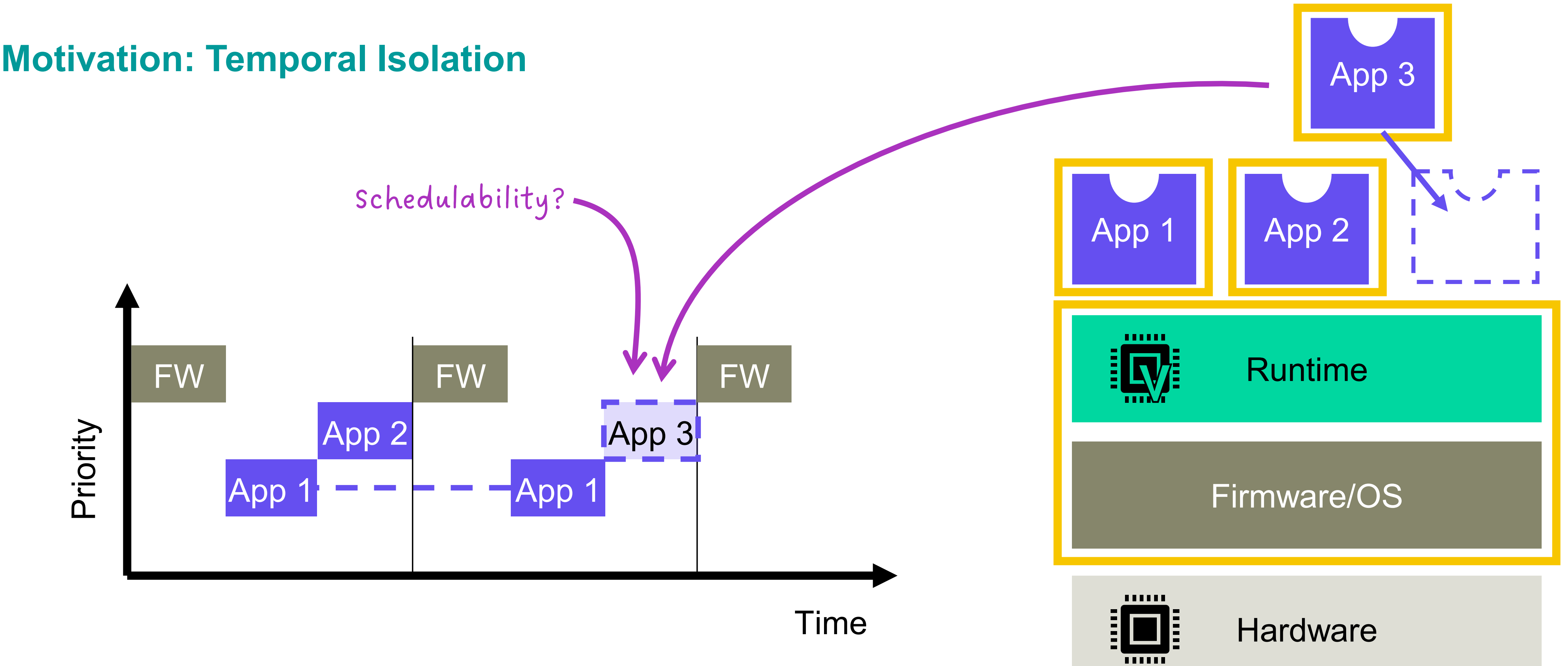
- Many source languages
- Platform agnostic ➡ "portable"
- Memory isolation
- Formal specification

Good "updateability"

OEM:	Original Equipment Manufacturer	Wasm:	WebAssembly
OS:	Operating System		
PLC:	Programmable Logic Controller		

Public | © Siemens 2026

Motivation: Temporal Isolation

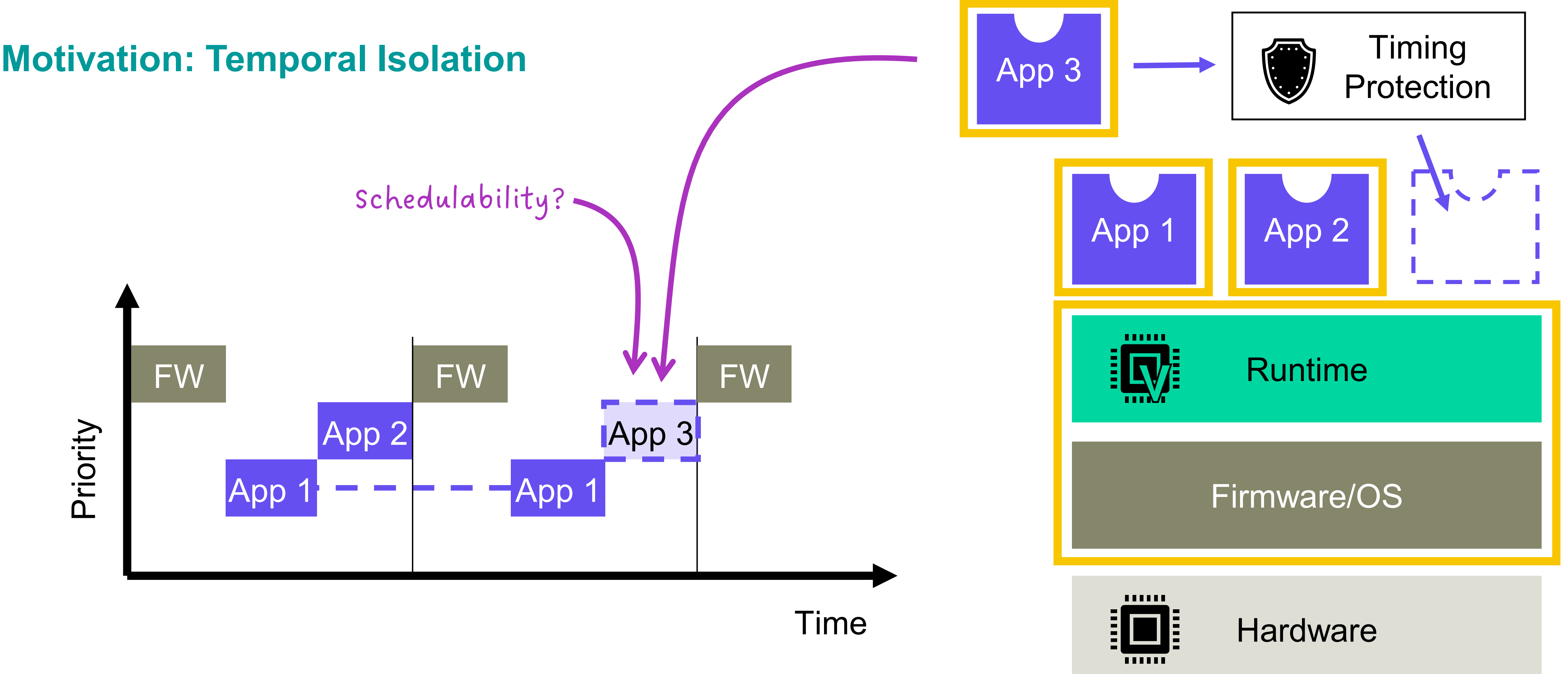


FW: Firmware
OS: Operating System
Wasm: WebAssembly

Public | © Siemens 2026

FAU | SIEMENS

Motivation: Temporal Isolation

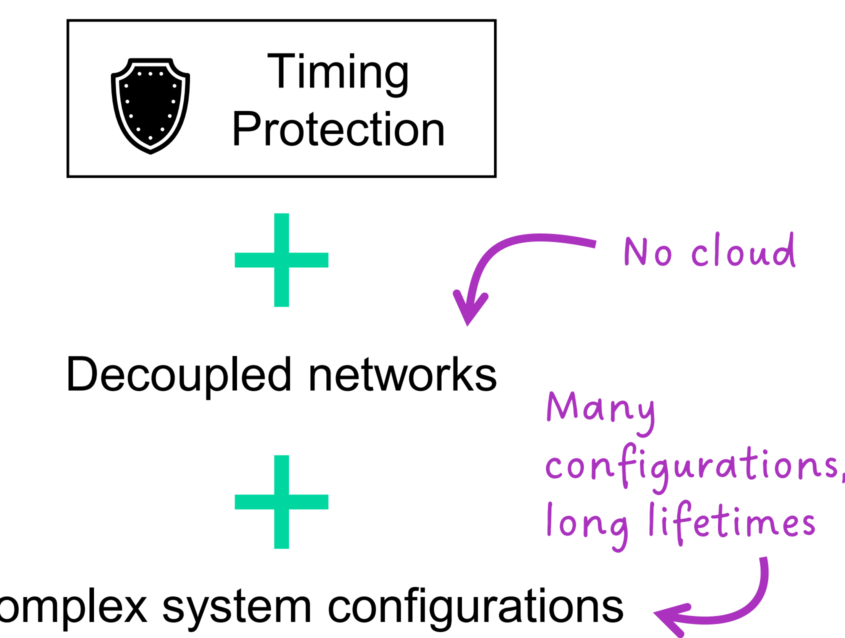


FW: Firmware
OS: Operating System
Wasm: WebAssembly

Public | © Siemens 2026

FAU | SIEMENS

Self-Containment & On-Device



OEM: Original Equipment Manufacturer

 Public | © Siemens 2026

Goal and Challenges

Enable WCET analysis…

Timing Protection

of portable Wasm bytecode, (interpreted) → 1. No available interpreter model

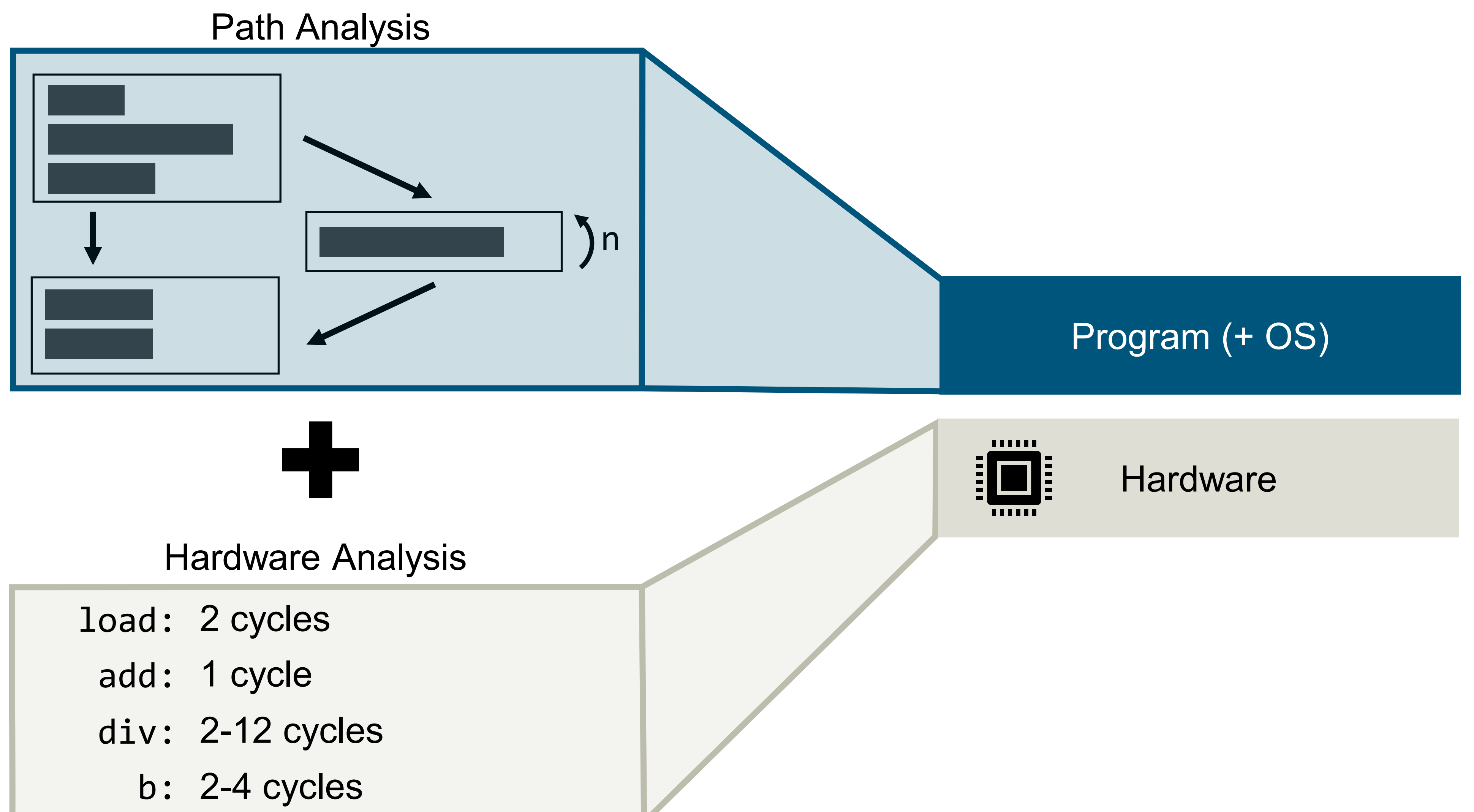
in a self-contained manner → 2. Cannot rely on external dependencies

conducted on-device. → 3. Tight RAM budgets (< 256 kiB)

ILP: Integer Linear Problem
WCET: Worst-Case Execution-Time

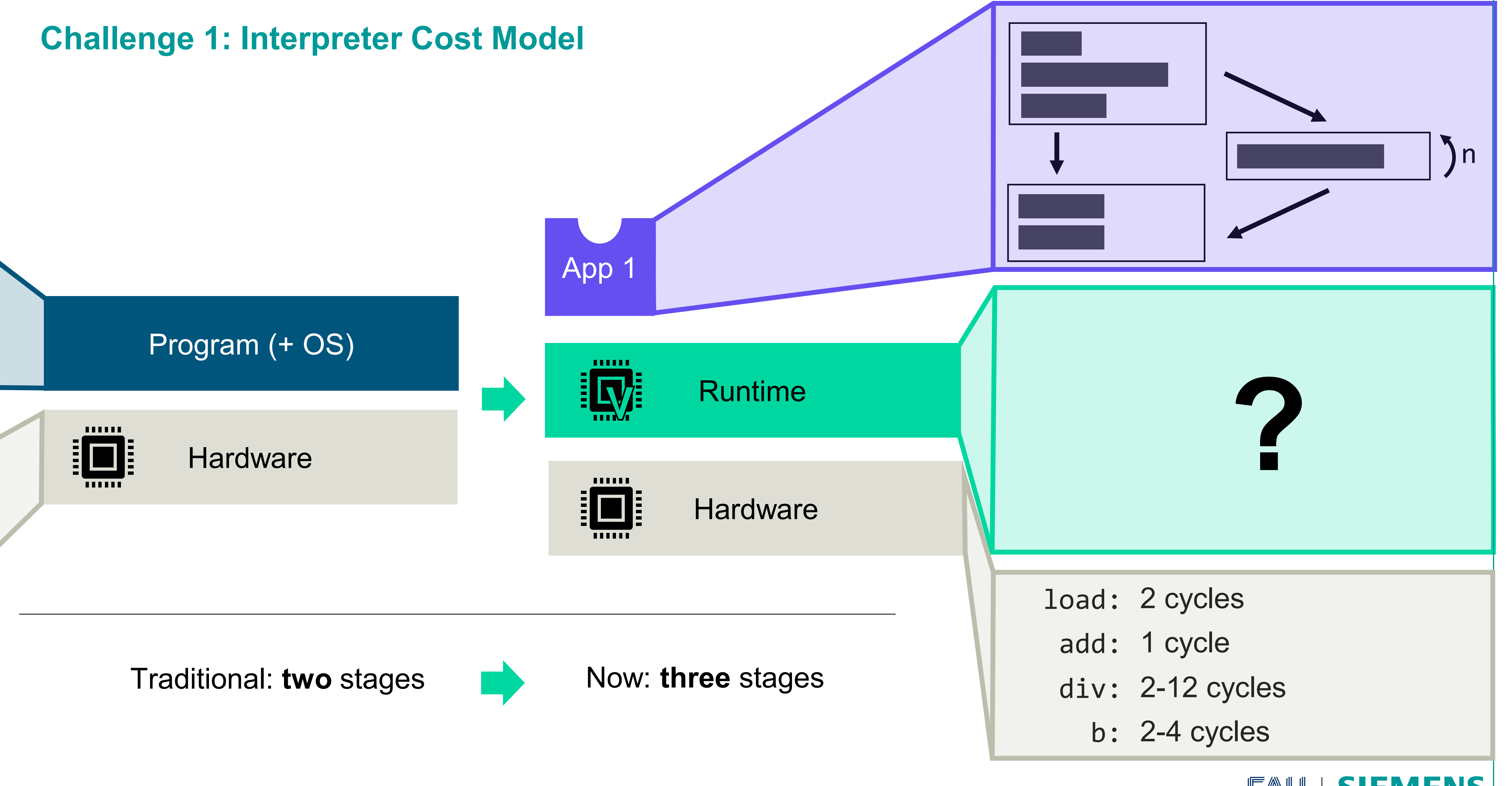
Public | © Siemens 2026

Challenge 1: Interpreter Cost Model



 Public | © Siemens 2026

Challenge 1: Interpreter Cost Model



 Public | © Siemens 2026

Challenge 1: Interpreter Cost Model

Instruction costs
contain control flow

Path analysis

n

Hardware analysis

load: 2 cycles

add: 1 cycle

div: 2-12 cycles

b: 2-4 cycles

Path an

n

Interpreter analysis

add

const

n

end

n

br_if

n

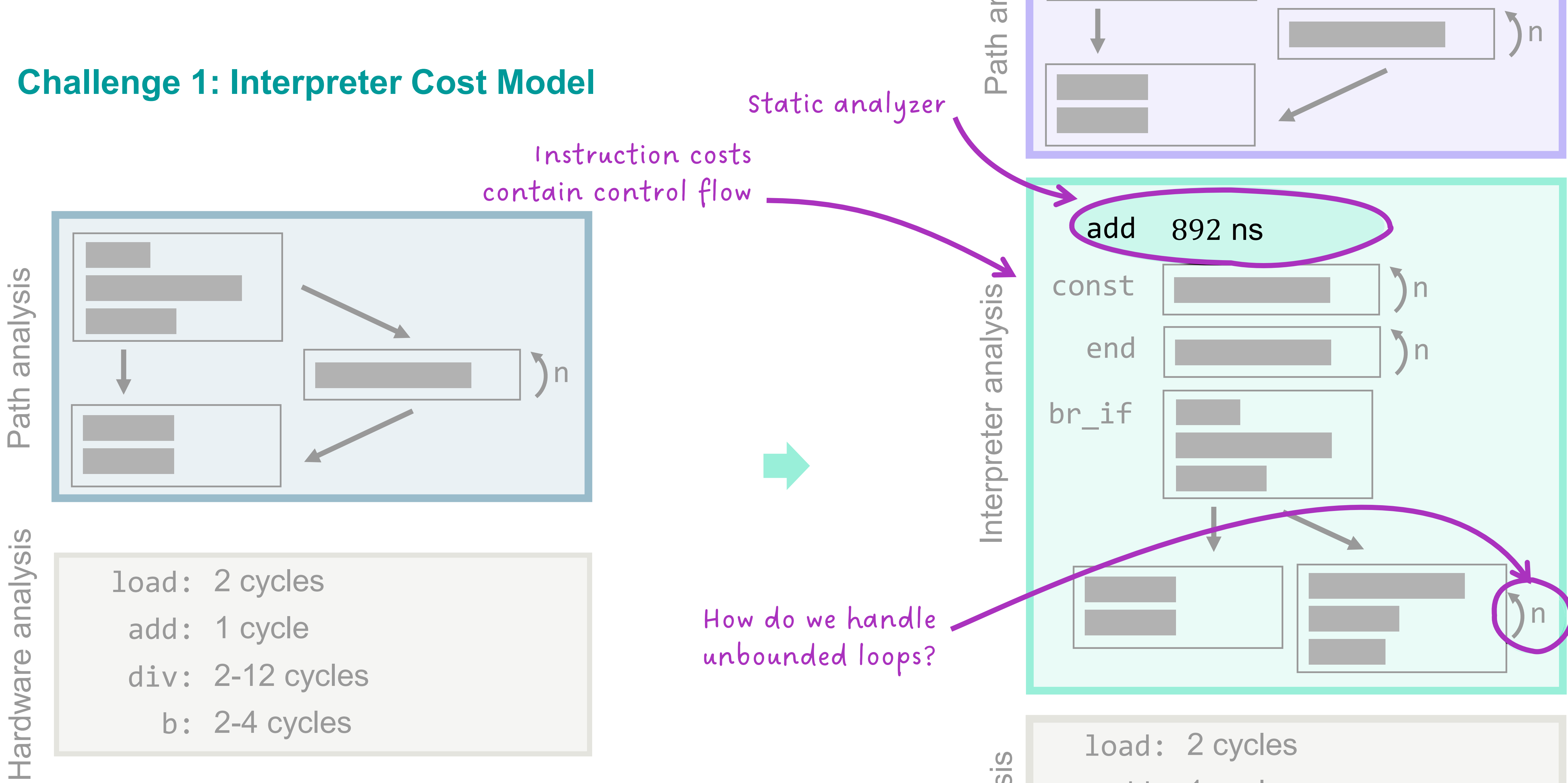
ysis

load: 2 cycles

add: 1 cycle

Public | © Siemens 2026
FAU | SIEMENS

Challenge 1: Interpreter Cost Model



 Public | © Siemens 2026

FAU | SIEMENS

Challenge 1: Interpreter Cost Model

add 892 ns
const)n
end)n
br_if
)n

Where do they come from?

Table: Most used Wasm instructions

#	Operation	Occurrences	Number of Unbounded Loops
1	local.get	22.58%	1 ×
2	i32.const	19.92%	1 ×
	...	...	...
14	i32.store8	1.15%	0x
15	br_if	1.13%	16 ×
16	i32.shl	1.06%	0x
	...	...	...

WCET: Worst-Case Execution-Time

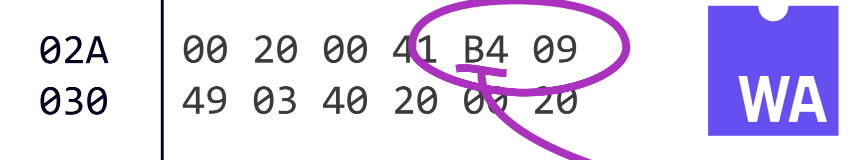
Public | © Siemens 2026

Sources of Unbounded Loops

1. Bytecode specification
   ➡ *Manual bounds analysis*

```
7    local.get 0
8    i32.const 1204
9    i32.lt_u
```

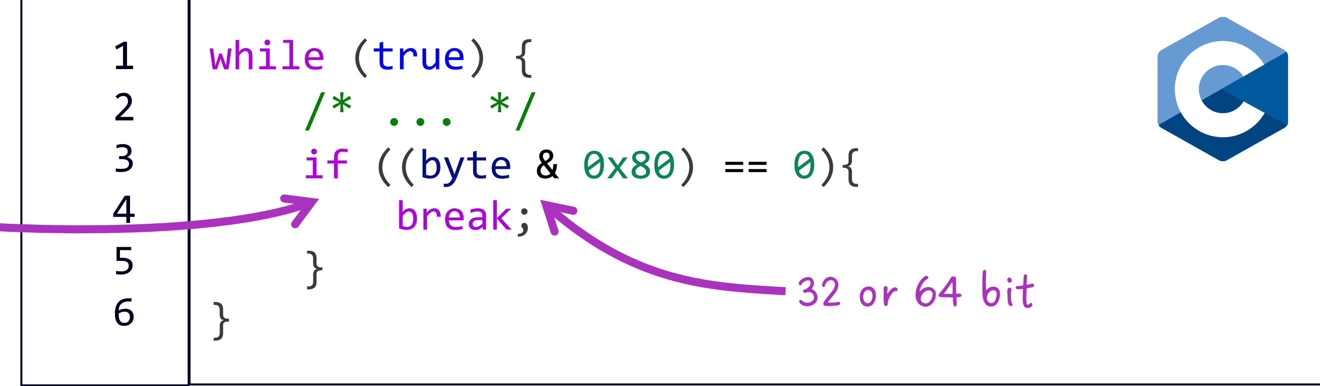
WA



5.2.2 Integers

All integers are encoded using the LEB128[28] variable-length integer encoding, in either unsigned or signed variant.

WebAssembly Working Group, *WebAssembly Specification 1.0*, 2019. https://www.w3.org/TR/wasm-core-1/



Solution

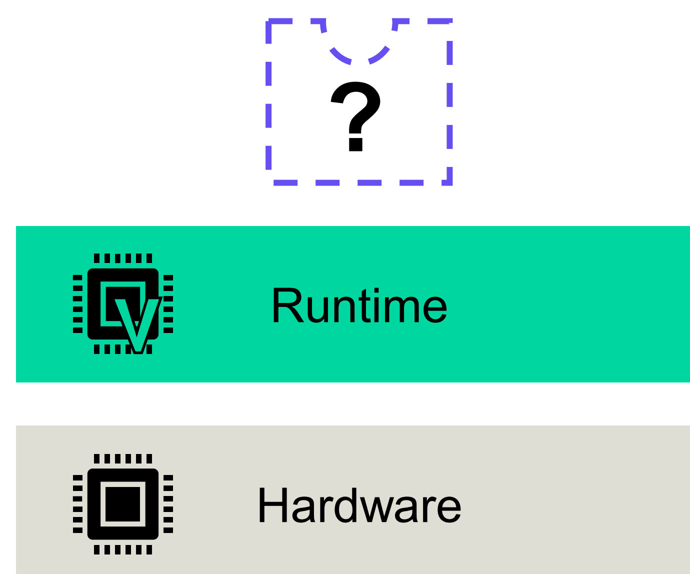
Manually understand and reconstruct loop bound

Public | © Siemens 2026

Sources of Unbounded Loops

1. Bytecode specification ✓
   ➡ *Manual bounds analysis*

2. Unknown binary



 Public | © Siemens 2026

Sources of Unbounded Loops

1. Bytecode specification
   ➡ *Manual bounds analysis*

2. Unknown binary

Control stack

0	End of i

```
1   (func (param i32)
2     (local i32)
3     block
4       local.get 0
5       i32.eqz
6       br_if 0
7       local.get 0
8       i32.const 1204
9       i32.lt_u
10      loop
11        local.get 0
12        local.get 1
13        i32.rem_u
14        i32.eqz
15        br_if 1
16        local.get 0
17        local.get 1
18        i32.add
19        i32.const 0
20        i32.ge_u
21        br_if 0
22      end
23      local.set 0
24    end
25  )
```

WA

 Public | © Siemens 2026

FAU | SIEMENS

Sources of Unbounded Loops

1. Bytecode specification ✓
   ➡ *Manual bounds analysis*

2. Unknown binary

```
1   HANDLE_OP(WASM_OP_BR_IF):
2       while (p < code_end_addr) {
3           opcode = *p++;
4           switch (opcode) {
5               case EXT_OP_BLOCK:
6               case EXT_OP_LOOP:
7               case EXT_OP_IF:
8                   skip_leb_int32(p, p_end);
9                   block_nested_depth++;
10                  break;
```

```
1   (func (param i32)
2     (local i32)
3     block
4       local.get 0
5       i32.eqz
6       br_if 0
7       local.get 0          ← Is end?
8       i32.const 1204       ← Is end?
9       i32.lt_u             ← Is end?
10      loop                 ← Is end?
11        local.get 0        ← Is end?
12        local.get 1        ← Is end?
13        i32.rem_u          ← Is end?
14        i32.eqz            ← Is end?
15        br_if 1            ← Is end?
16        local.get 0        ← Is end?
17        local.get 1        ← Is end?
18        i32.add            ← Is end?
19        i32.const 0        ← Is end?
20        i32.ge_u           ← Is end?
21        br_if 0            ← Is end?
22      end                  ← Is end?
23      local.set 0          ← Is end?
24    end                    ← Is end?
25  )
```

Found it

 Public | © Siemens 2026

Sources of Unbounded Loops

1. Bytecode specification ✓
   ➡ *Manual bounds analysis*

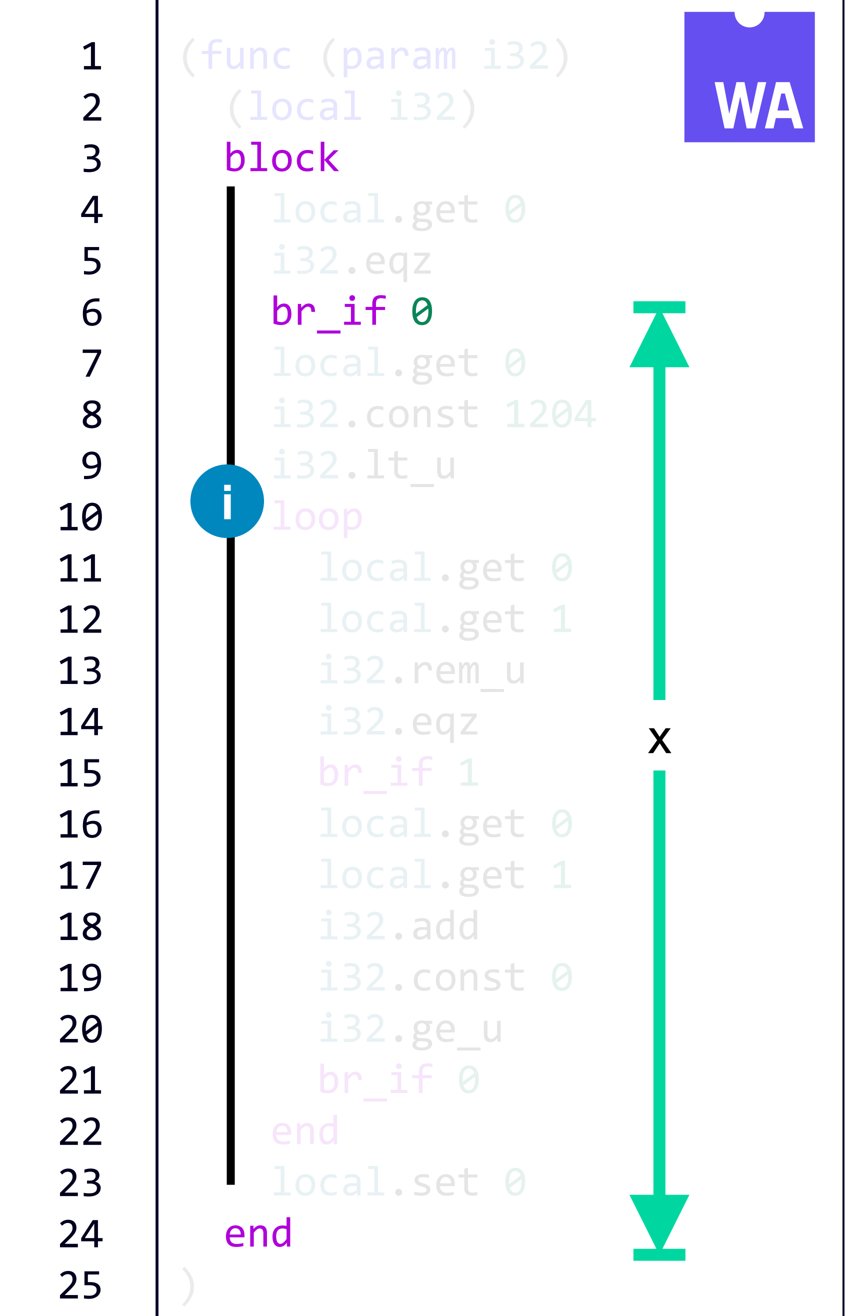
2. Unknown binary ✓
   ➡ *Load-time interpreter model specialization*

Solution

But: Impractical maxima: bytecode length – position

Solution

Load-time interpreter model specialization: Find maximum distance during loading



 Public | © Siemens 2026

FAU | SIEMENS

Sources of Unbounded Loops

1. Bytecode specification ✓
   ➡ *Manual bounds analysis*
2. Unknown binary ✓
   ➡ *Load-time interpreter model specialization*
3. Unknown runtime state ✓
   ➡ *Cache model*

Solution

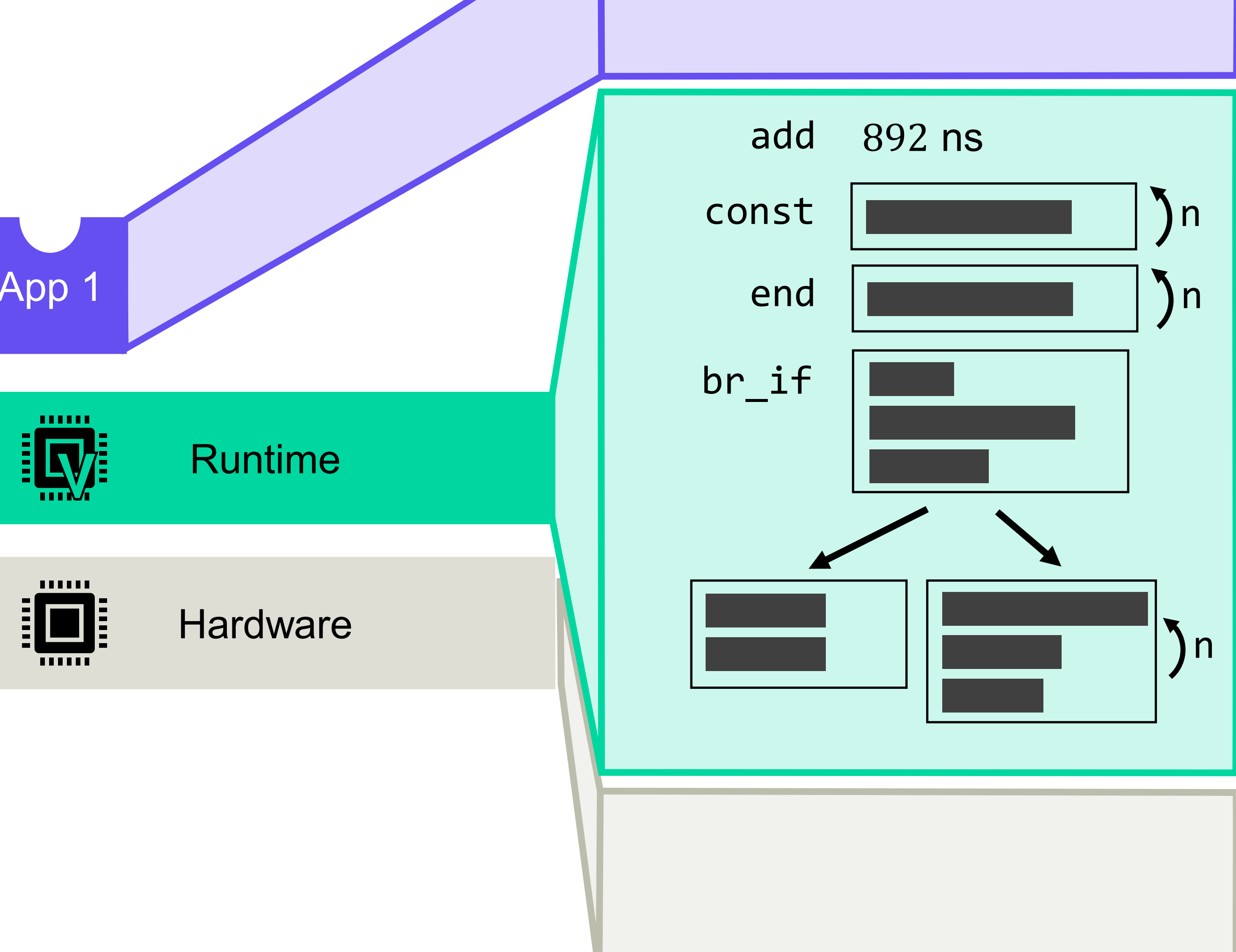
Cache model: Separate instruction times with cache hit and miss

```
1   (func (param i32)
2     (local i32)
3     block
4       local.get 0
5       i32.eqz
6       br_if 0
7       local.get 0
8       i32.const 1204
9       i32.lt_u
10      loop
11        local.get 0
12        local.get 1
13        i32.rem_u
14        i32.eqz
15        br_if 1
16        local.get 0
17        local.get 1
18        i32.add
19        i32.const 0
20        i32.ge_u
21        br_if 0
22      end
23      local.set 0
24    end
25  )
```


Public | © Siemens 2026
FAU | SIEMENS

Challenge 1: Interpreter Cost Model

1. Bytecode specification
   ➡ *Manual bounds analysis*
2. Unknown binary
   ➡ *Load-time interpreter model specialization*
3. Unknown runtime state
   ➡ *Cache model*



Public | © Siemens 2026
FAU | SIEMENS

Challenge 1: Interpreter Cost Model

„Static Interpreter Model“

1. Bytecode specification
   ➜ *Manual bounds analysis*

2. Unknown binary ✓
   ➜ *Load-time interpreter model specialization*

3. Unknown runtime state ✓
   ➜ *Cache model*

App 1

Runtime

Hardware

add 892 ns

const 8192 ns

end 9184 ns

br_if

n

 Public | © Siemens 2026

FAU | SIEMENS

Challenge 1: Interpreter Cost Model

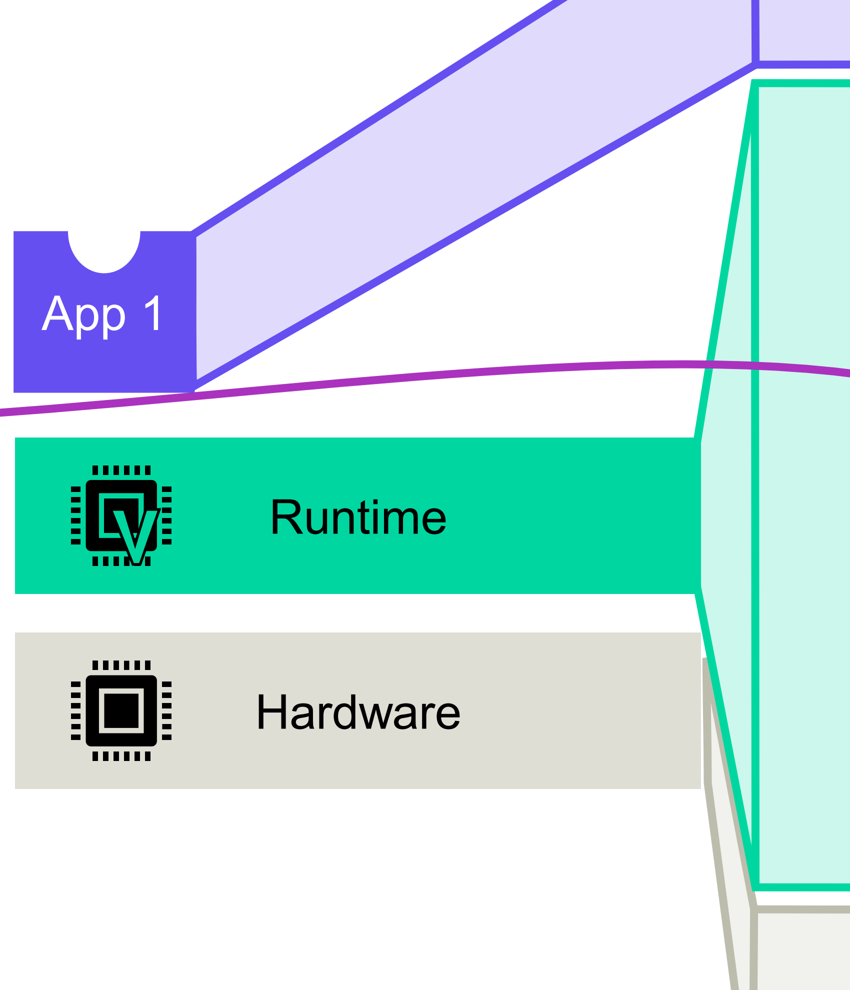
„Static Interpreter Model"

1. Bytecode specification
   ➡ *Manual bounds analysis*

„Load-Time Specialization"

2. Unknown binary
   ➡ *Load-time interpreter model specialization*

3. Unknown runtime state
   ➡ *Cache model*



Public | © Siemens 2026

Challenge 1: Interpreter Cost Model

„Static Interpreter Model“

1. Bytecode specification
   ➡ *Manual bounds analysis*

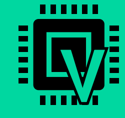
„Load-Time Specialization“

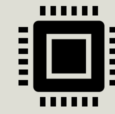
2. Unknown binary
   ➡ *Load-time interpreter model specialization*

„Cache Model“

3. Unknown runtime state
   ➡ *Cache model*

App 1


Runtime


Hardware

`add`	892 ns
`const`	8192 ns
`end`	9184 ns
`br_if`	$11158x + 167351$ ns
Cache hit	$500x + 100001$ ns

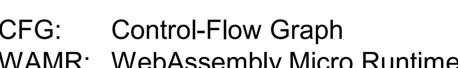
 Public | © Siemens 2026

FAU | SIEMENS

Challenge 2: External Dependencies

➡ Cannot rely on external tools

1. Parse binary ➡ Adapted parser from WAMR ✅

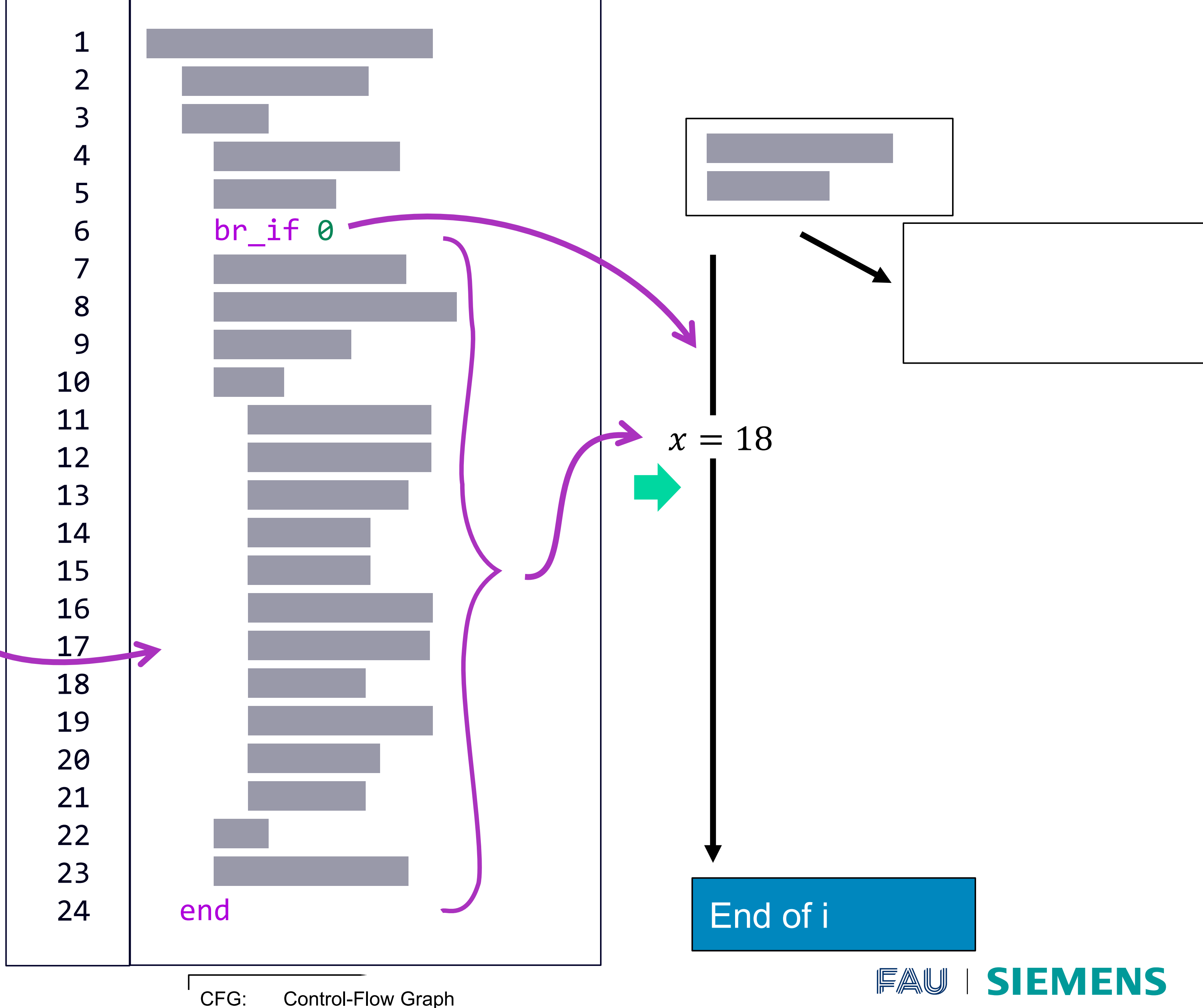


 Public | © Siemens 2026

Challenge 2: External Dep.

➡ Cannot rely on external tools

1. Parse binary ✓
2. Create CFG



CFG: Control-Flow Graph

Public | © Siemens 2026
FAU | SIEMENS

Challenge 2: External Dependencies

➡ Cannot rely on external tools

1. Parse binary ✓
2. Create CFG ✓
3. Calculate WCET ➡ No external dependencies

Solution

Timing Schema [1]

- Sum up sequential instructions
- Use maximum for branches
- Multiply with loop bounds

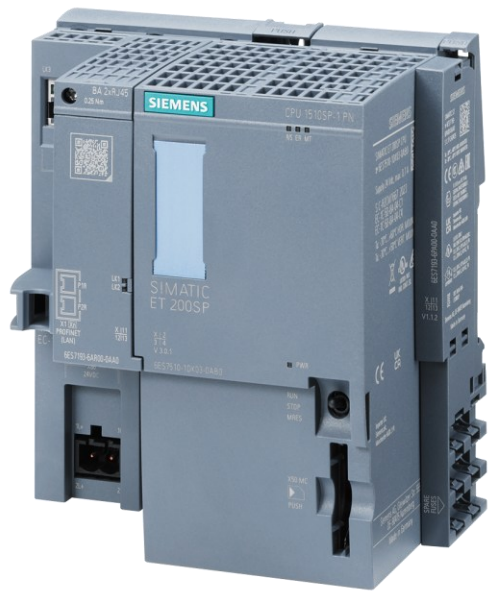
[1] A. Colin and I. Puaut, "A modular and retargetable framework for tree-based WCET analysis," in Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS), 2001

CFG: Control-Flow Graph
ILP: Integer Linear Problem
WCET: Worst-Case Execution-Time

 Public | © Siemens 2026

FAU | SIEMENS

Challenge 3: Resource Constraints



SIMATIC ET 200SP: 200 kiB RAM, 32-bit, 160 MHz

➡ Memory constraints

Solution

CFG pruning during estimation

Control stack

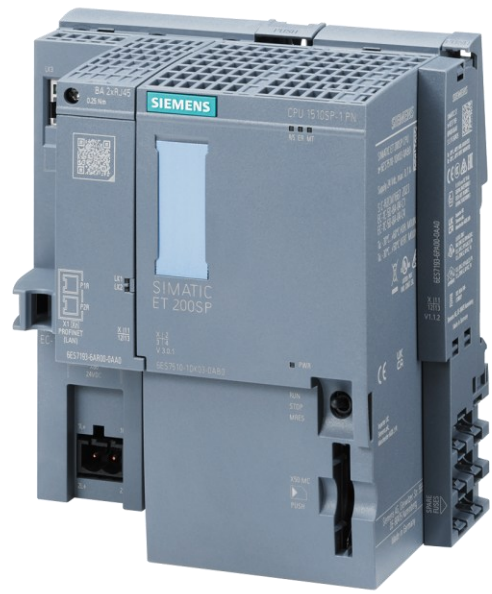
0	~~Start of iii~~
1	Start of ii
2	End of i

i
ii
iii
End
End
End

Structured control flow

Public | © Siemens 2026

Challenge 3: Resource Constraints



SIMATIC ET 200SP: 200 kiB RAM, 32-bit, 160 MHz

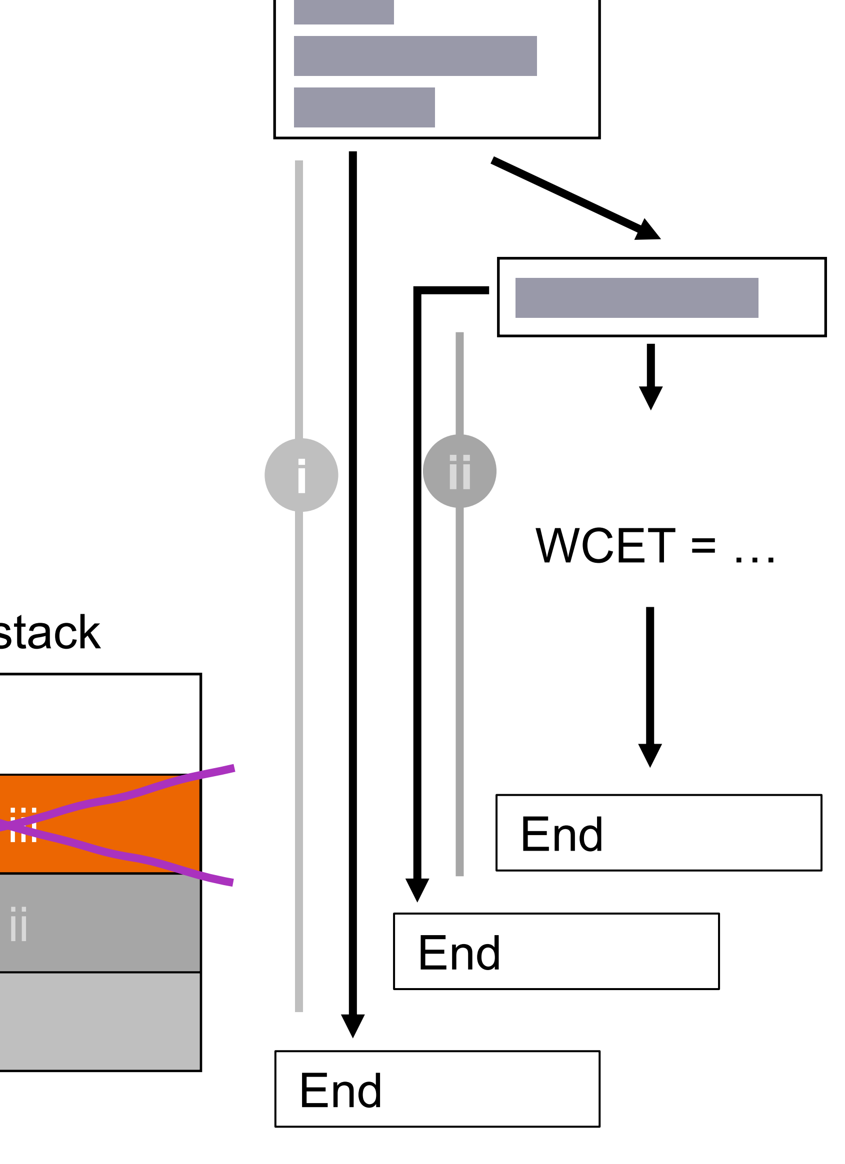
➡ Memory constraints

Solution

CFG pruning during estimation

Control stack

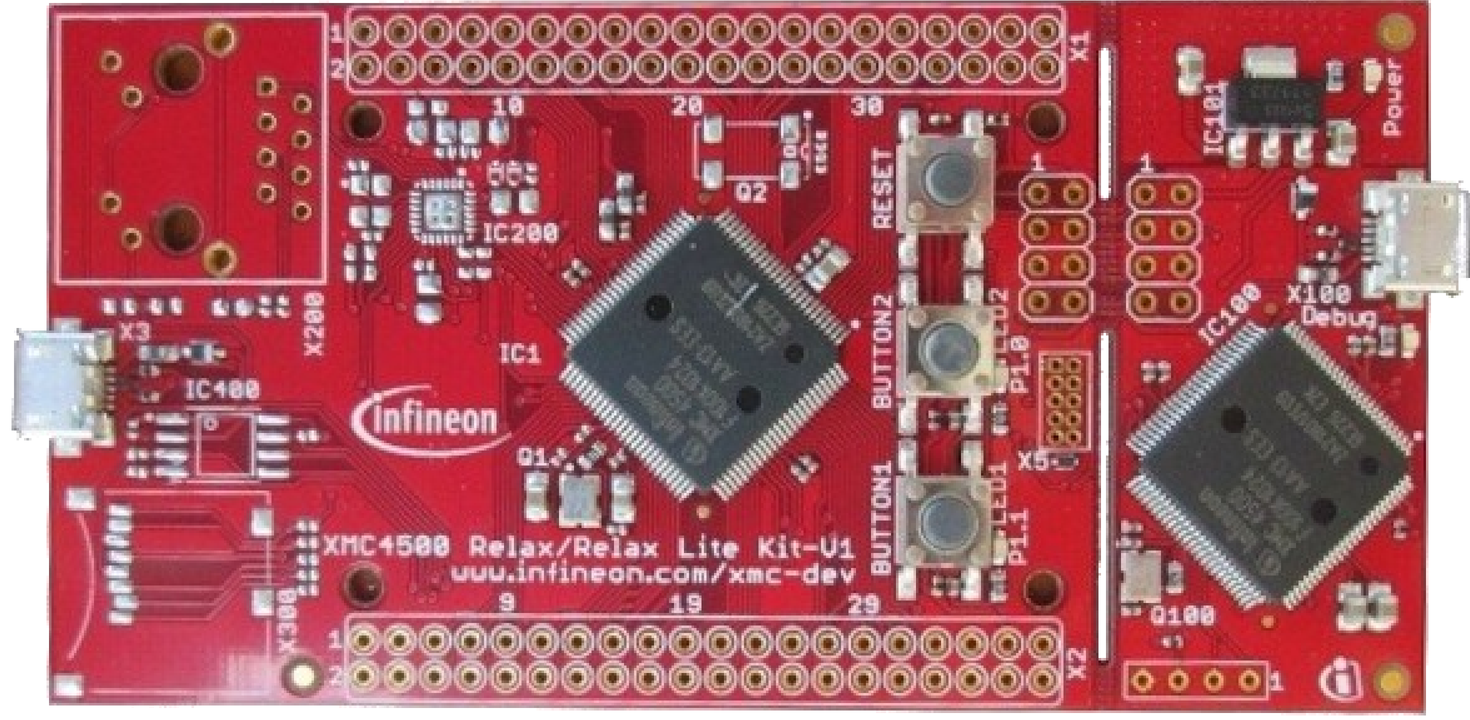
0	~~Start of iii~~
1	Start of ii
2	End of i



 Public | © Siemens 2026

FAU | SIEMENS

Evaluation



XMC4500 Relax Kit Lite: 2×64 kiB, 32-bit, 120 MHz

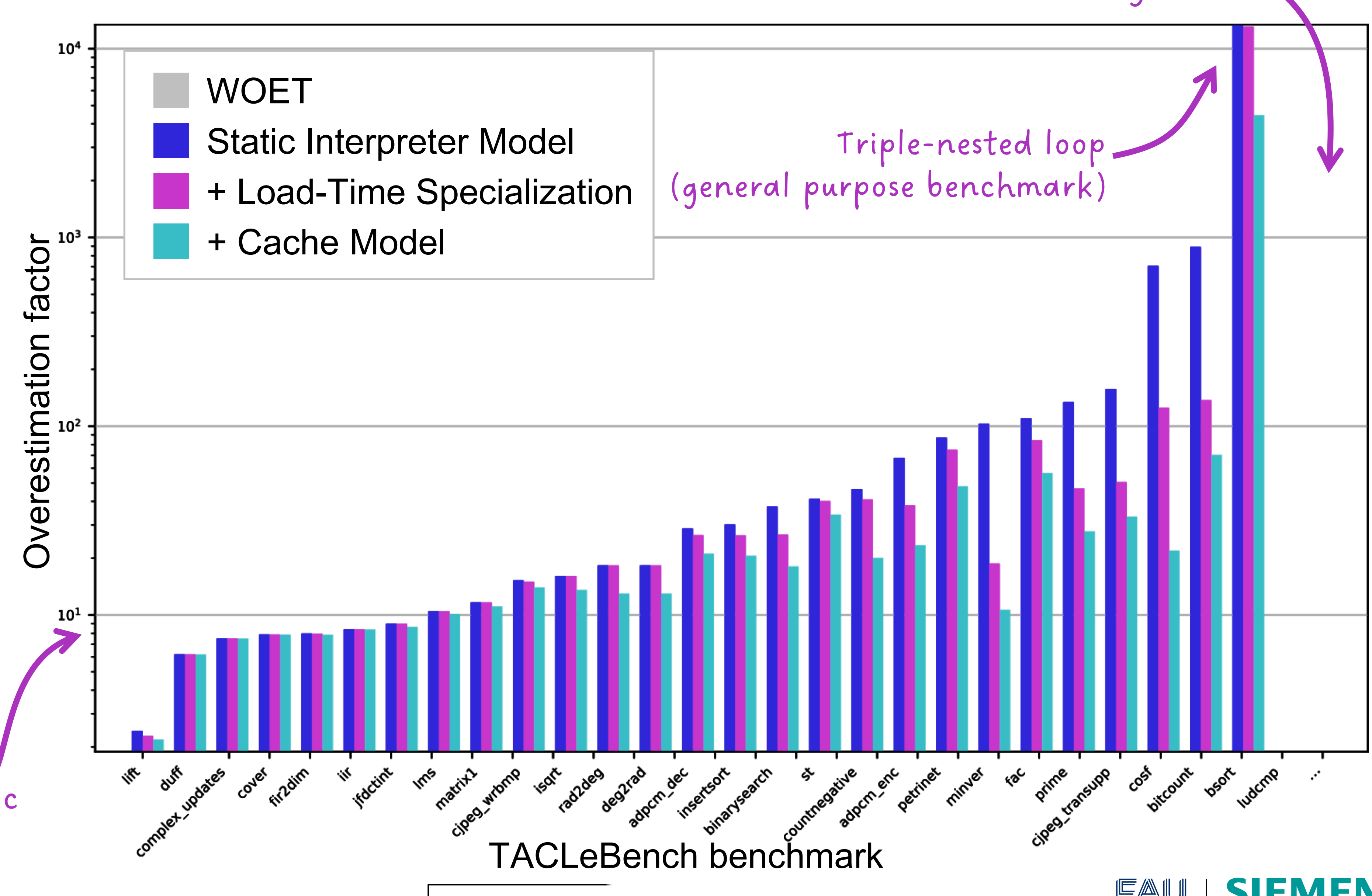
PLC: Programmable Logic Controller

 Public | © Siemens 2026

Evaluation: Overestimation of TACLeBench

Take-Away

- Analysis is feasible for **larger programs**
- **Measures** reduce overestimation
- **Pinpoint** sources of overestimation



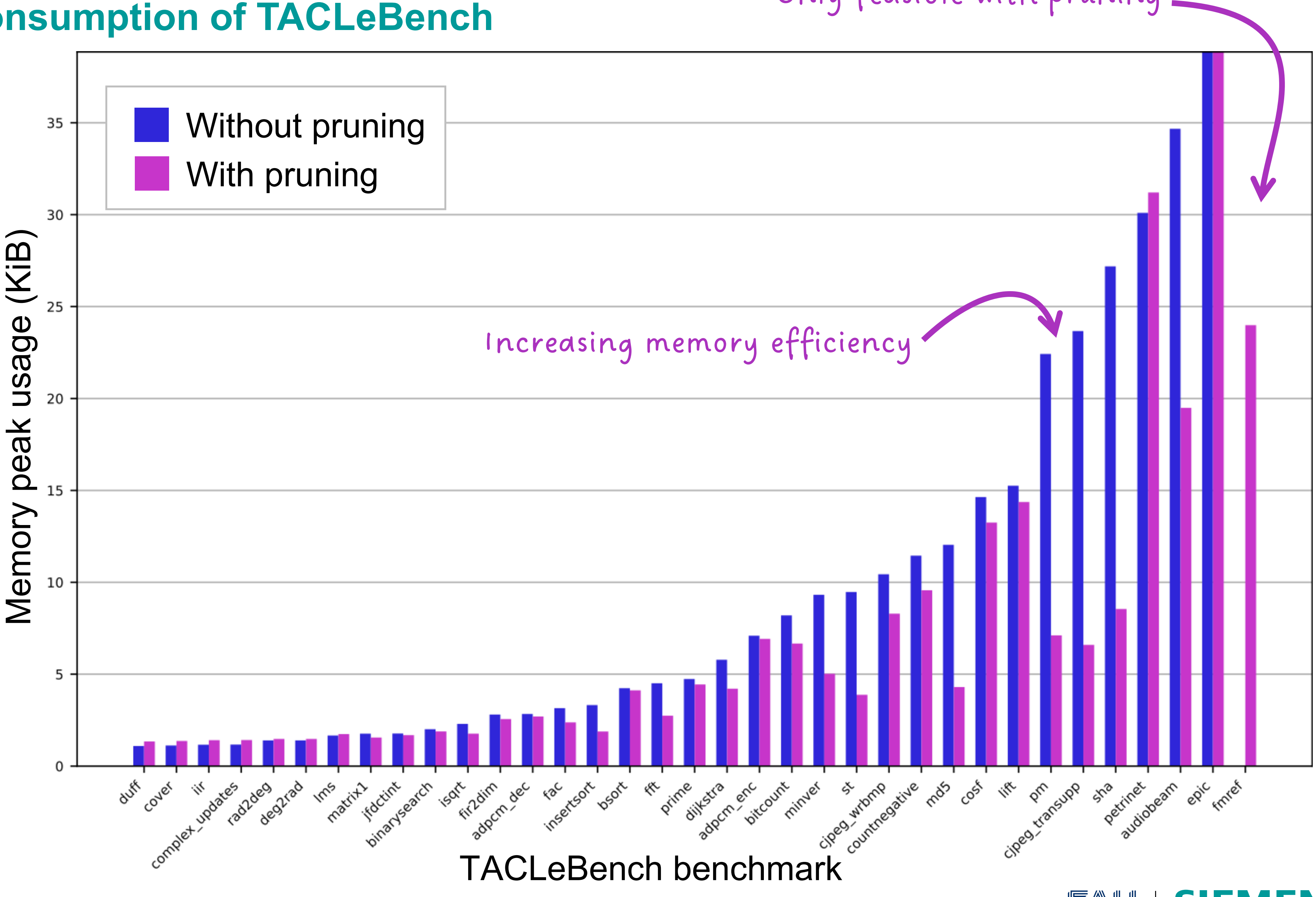
WOET: Worst-Observed Execution-Time

Public | © Siemens 2026

Evaluation: RAM Consumption of TACLeBench

Take-Away

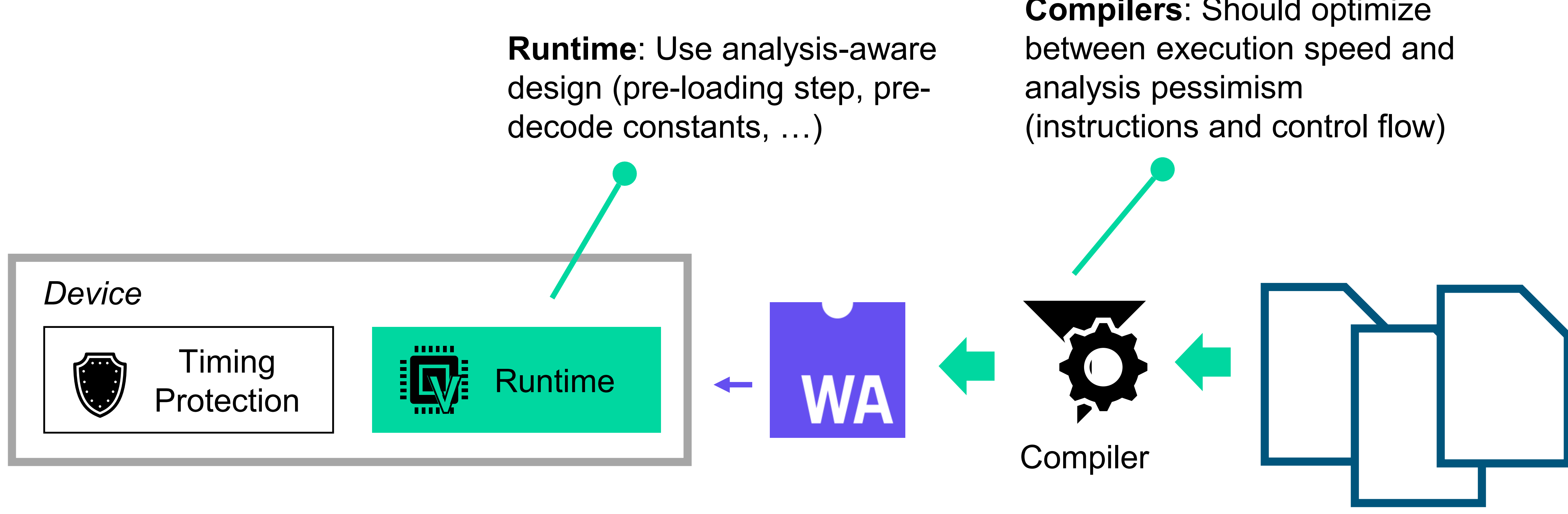
- Analysis is feasible on **resource-constrained** devices
- **Pruning** reduces memory consumption in many cases



 Public | © Siemens 2026

FAU | SIEMENS

Outlook



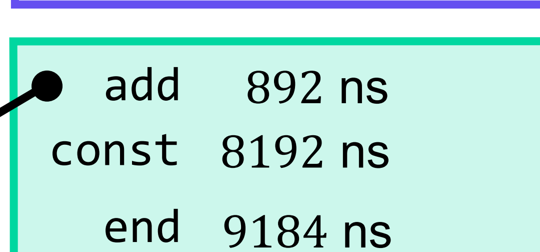
 Public | © Siemens 2026

FAU | SIEMENS

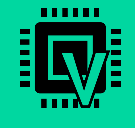
Summary

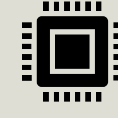
Wasm-WCET: Enable WCET analysis…

1. of portable **Wasm bytecode**,
   - ➔ Static Interpreter Model
   - ➔ Load-Time Specialization
   - ➔ Cache Model
2. in a **self-contained** manner,
   - ➔ Timing Schema [1]
3. conducted **on-device**.
   - ➔ CFG Pruning

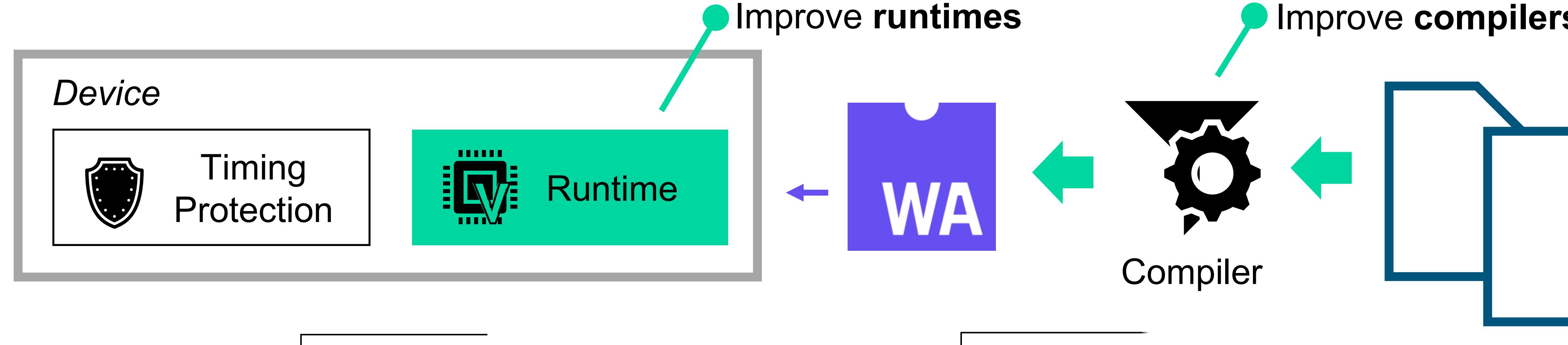


App 1





Outlook: "Analysis-aware runtimes & compilers"



[1] A. Colin and I. Puaut, "A modular and retargetable framework for tree-based WCET analysis," in Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS), 2001

CFG: Control-Flow Graph
Wasm: WebAssembly
WCET: Worst-Case Execution-Time

 Public | © Siemens 2026

FAU | SIEMENS