Securing interruptible enclaved execution on small microprocessors

Matteo Busi

Joint work with:
J. Noorman, J. Van Bulck, L. Galletta, P. Degano, J.T. Mühlberg, F. Piessens

21/02/2021

Isolation mechanisms

Isolation mechanisms

Abstractions for **well-defined interaction** among (untrusted) programs

Isolation mechanisms

Abstractions for **well-defined interaction** among (untrusted) programs

1. Processes
2. User/kernel isolation

Isolation mechanisms

Abstractions for **well-defined interaction** among (untrusted) programs

1. Processes
2. User/kernel isolation
3. TEEs (TrustZone, KeyStone, SGX, …)
4. Capabilities (CHERI, Arm Morello)

Isolation mechanisms

Abstractions for **well-defined interaction** among (untrusted) programs

1. Processes
2. User/kernel isolation
3. TEEs (TrustZone, KeyStone, SGX, …)
4. Capabilities (CHERI, Arm Morello)

Programmers (unknowingly) use them for security!

Isolation mechanisms … are broken

Isolation mechanisms ... are broken

Advanced μarch. features have proven to break **security**!

Isolation mechanisms … are broken

Advanced μarch. features have proven to break **security**!

Isolation mechanisms ... are broken

Advanced μarch. features have proven to break **security**!

Isolation mechanisms ... are broken

Advanced μarch. features have proven to break **security**!

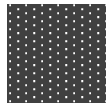
When isolation … isolates?

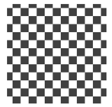
When isolation … isolates?

“Abstractions for **well-defined interaction** among (untrusted) programs”

When isolation … isolates?

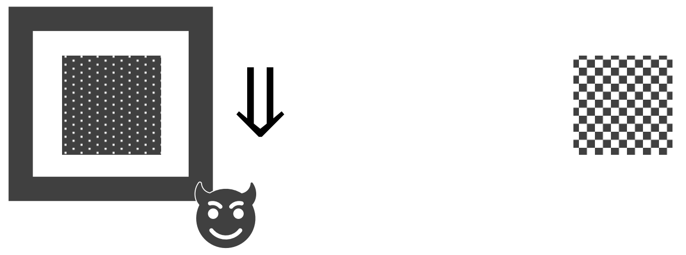
“Abstractions for **well-defined interaction** among (untrusted) programs”





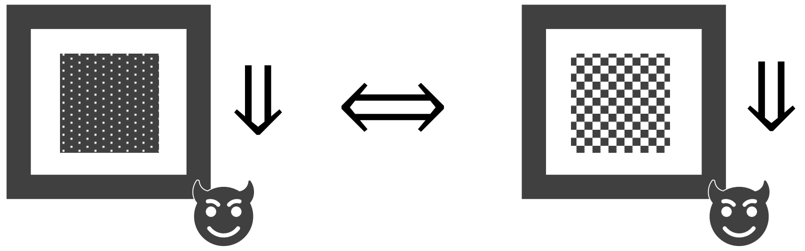
When isolation … isolates?

“Abstractions for **well-defined interaction** among (untrusted) programs”



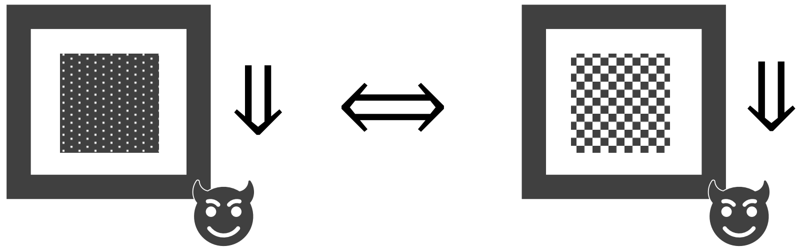
When isolation … isolates?

“Abstractions for **well-defined interaction** among (untrusted) programs”



When isolation … isolates?

“Abstractions for **well-defined interaction** among (untrusted) programs”



Isolation **security** ≜ equiconvergence under any attacker, i.e., **contextual equivalence**

Goals

Goals

(**H**) High-language ≈

ISA with an isolation mechanism ≈
the programmer’s mental model

Goals

(**H**) High-language ≈

ISA with an isolation mechanism ≈ the programmer’s mental model

(**L**) Low-language ≈

High-language + carefully implemented “problematic” feature(s)

Goals

(**H**) High-language ≈
ISA with an isolation mechanism ≈ the programmer's mental model

(**L**) Low-language ≈
High-language + carefully implemented "problematic" feature(s)

We want:

- isolation of **L** **not weaker** than that of **H**, and
- backwards compatibility

Goals (formally)

Goals (formally)

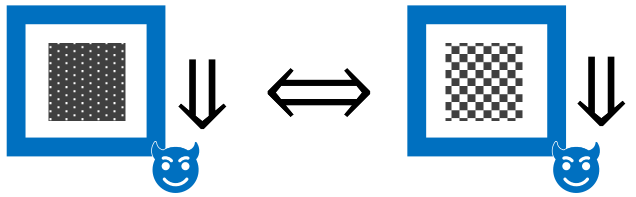
(H) High-language ≈
ISA with an isolation mechanism ≈
the programmer’s mental model

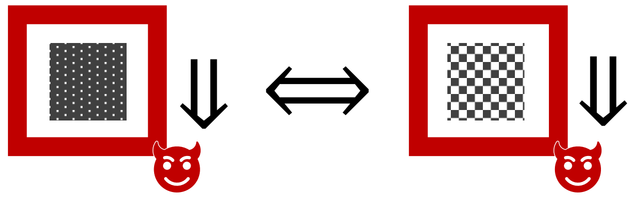
(L) Low-language ≈
High-language + carefully
implemented “problematic” feature(s)

Goals (formally)

(H) High-language ≈
ISA with an isolation mechanism ≈
the programmer's mental model

(L) Low-language ≈
High-language + carefully
implemented "problematic" feature(s)

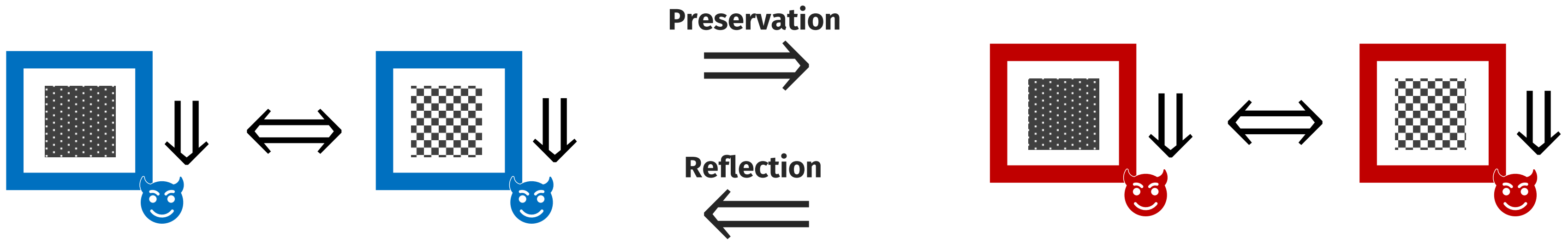




Goals (formally)

(H) High-language ≈
ISA with an isolation mechanism ≈ the programmer's mental model

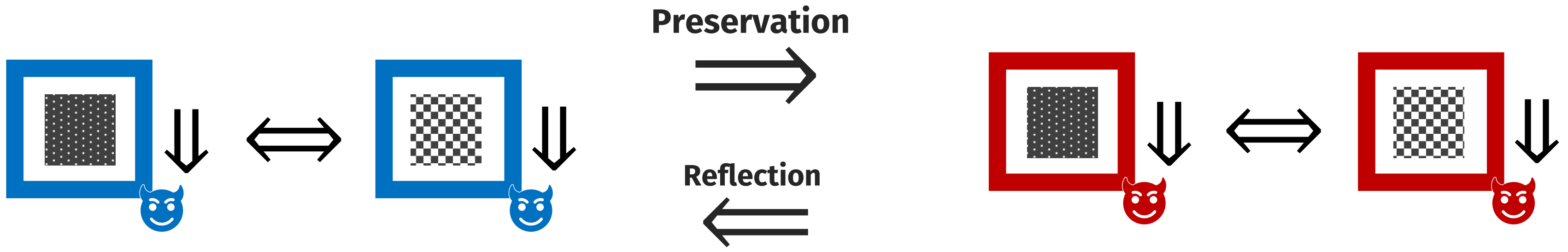
(L) Low-language ≈
High-language + carefully implemented "problematic" feature(s)



Goals (formally)

(H) High-language ≈
ISA with an isolation mechanism ≈
the programmer's mental model

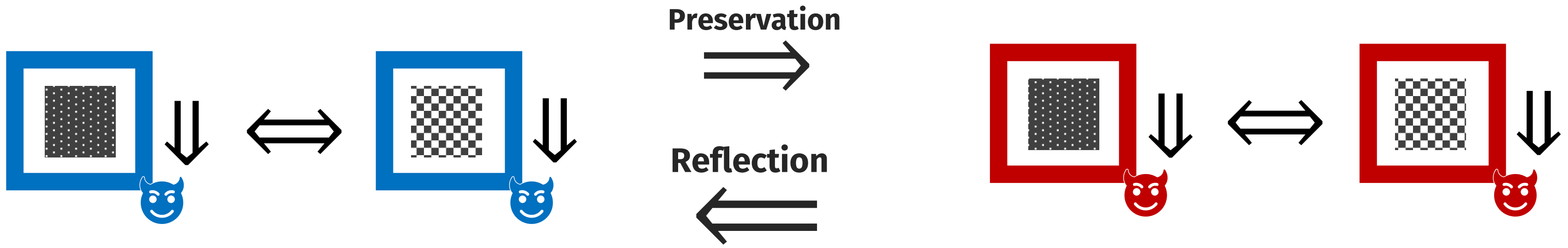
(L) Low-language ≈
High-language + carefully
implemented "problematic" feature(s)



Goals (formally)

(H) High-language ≈
ISA with an isolation mechanism ≈
the programmer's mental model

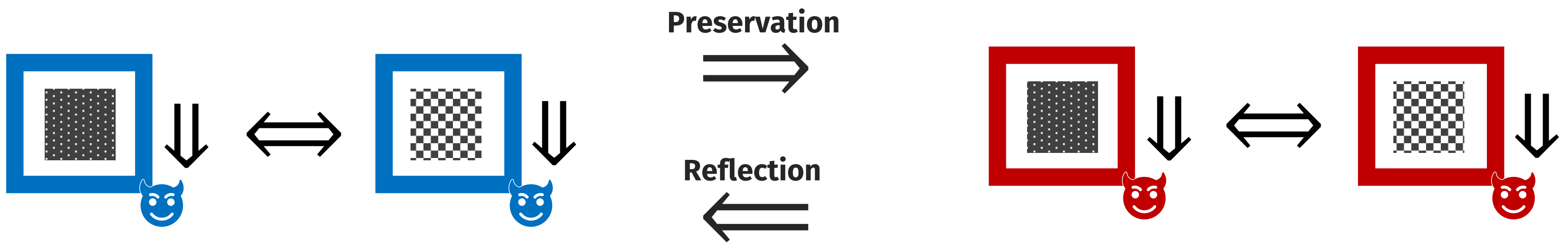
(L) Low-language ≈
High-language + carefully
implemented "problematic" feature(s)



Goals (formally)

(H) High-language ≈
ISA with an isolation mechanism ≈
the programmer’s mental model

(L) Low-language ≈
High-language + carefully
implemented “problematic” feature(s)



i.e., **H** and **L** are fully abstract

Our case: enclaves as isolation mechanism

"Dedicated" execution environments for secure remote computation

- **Attacker model:** everything outside the enclave (incl. OS, I/O devices, ...)
- Code and data integrity and confidentiality, via attestation & access control

Our case: enclaves as isolation mechanism

“Dedicated” execution environments for secure remote computation

- **Attacker model:** everything outside the enclave (incl. OS, I/O devices, …)
- Code and data integrity and confidentiality, via attestation & access control

Our focus: just isolation aspects

Our case: enclaves as isolation mechanism

"Dedicated" execution environments for secure remote computation

- **Attacker model:** everything outside the enclave (incl. OS, I/O devices, ...)
- Code and data integrity and confidentiality, via ~~attestation &~~ access control

Our focus: just isolation aspects

Sancus

Enclaved-execution (embedded) architecture on top of TI MSP430

- RISC instruction set
- Each instruction may take a different amount of time
- 64KB of memory, split into **protected** (enclaved) and **unprotected**
- No speculative execution, no interruptible enclaves, …

https://distrinet.cs.kuleuven.be/software/sancus/

Sancus

Enclaved-execution (embedded) architecture on top of TI MSP430

- RISC instruction set
- Each instruction may take a different amount of time
- 64KB of memory, split into **protected** (enclaved) and **unprotected**
- No speculative execution, **no interruptible enclaves**, …

https://distrinet.cs.kuleuven.be/software/sancus/

Why no **interruptible enclaves**?

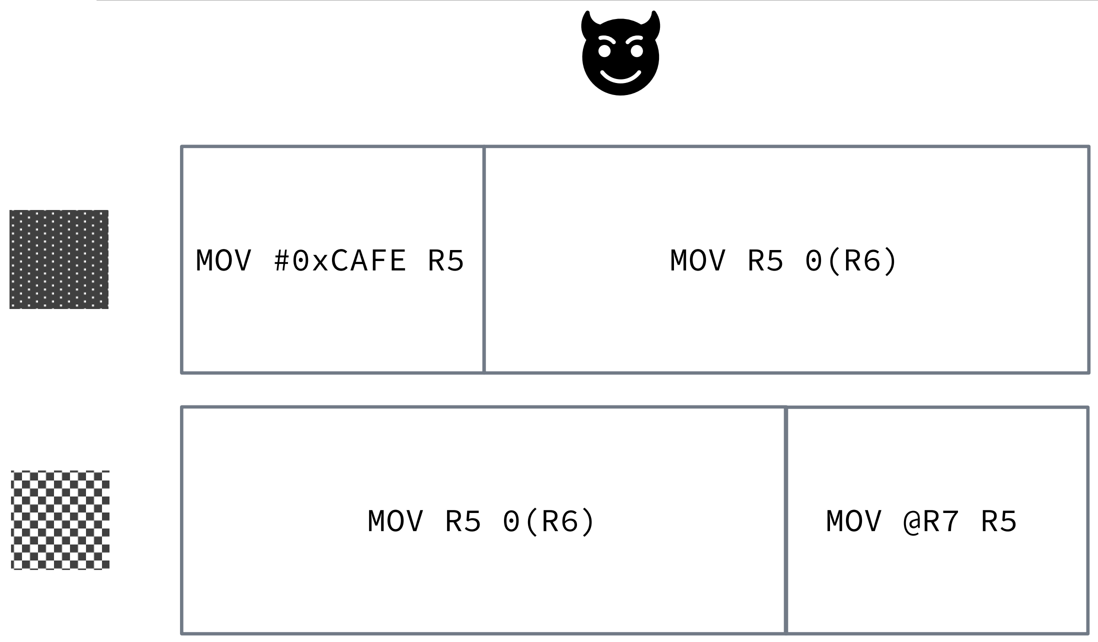
Why no interruptible enclaves?

Nemesis attack! [Van Bulck et al., CCS’18]

Why no **interruptible enclaves**?

Nemesis attack! [Van Bulck et al., CCS’18]

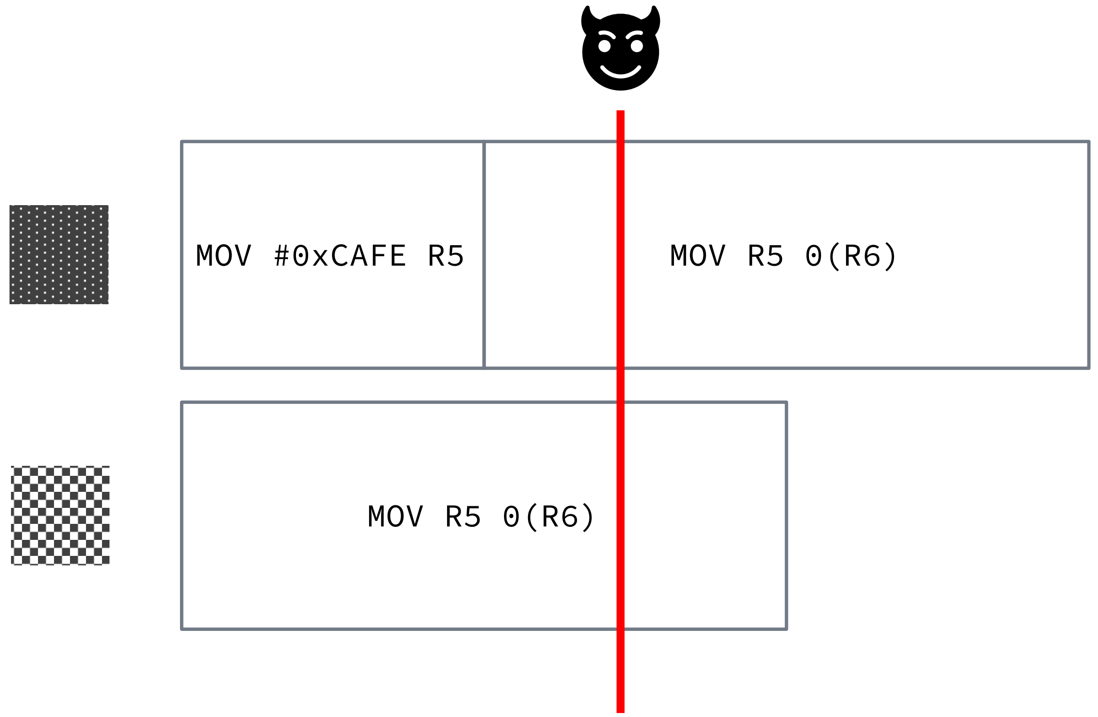
Why no interruptible enclaves?



```
MOV #0xCAFE R5    MOV R5 0(R6)
MOV R5 0(R6)      MOV @R7 R5
```

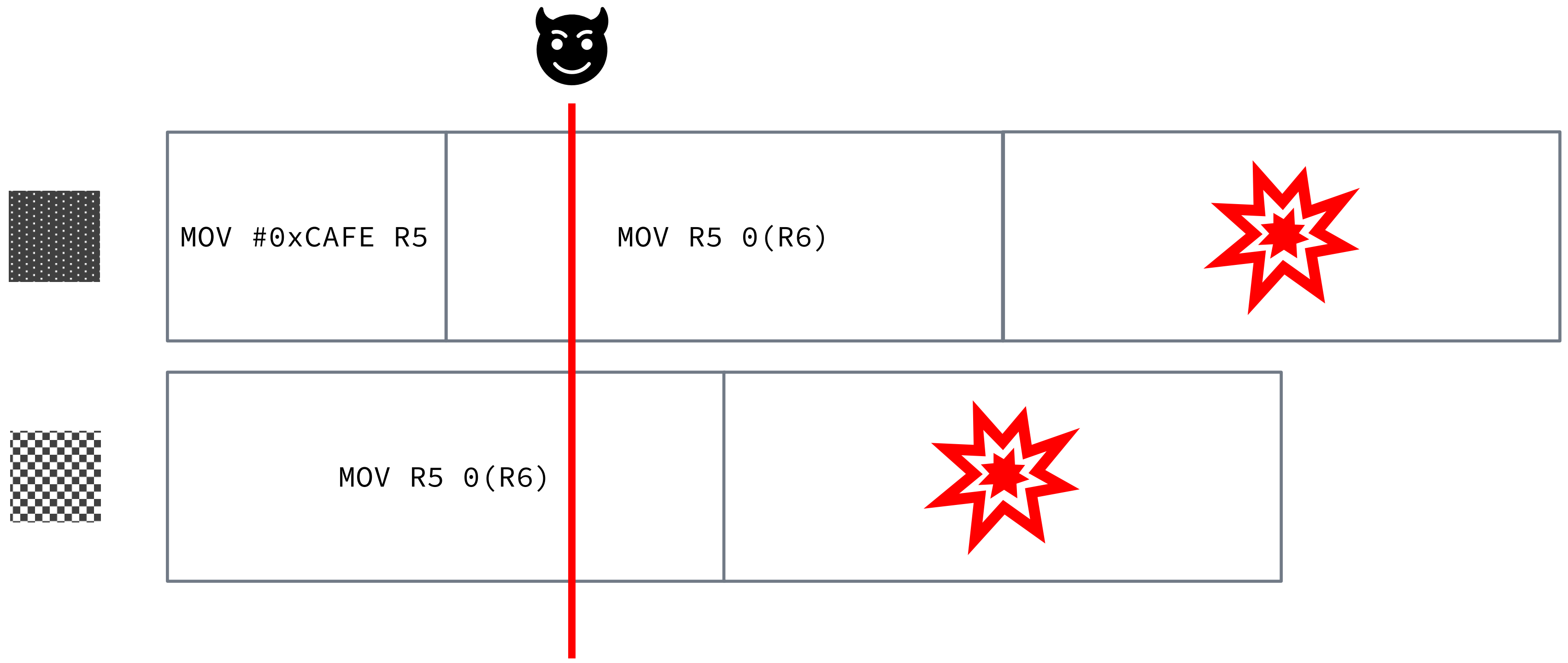
Nemesis attack! [Van Bulck et al., CCS’18]

Why no interruptible enclaves?



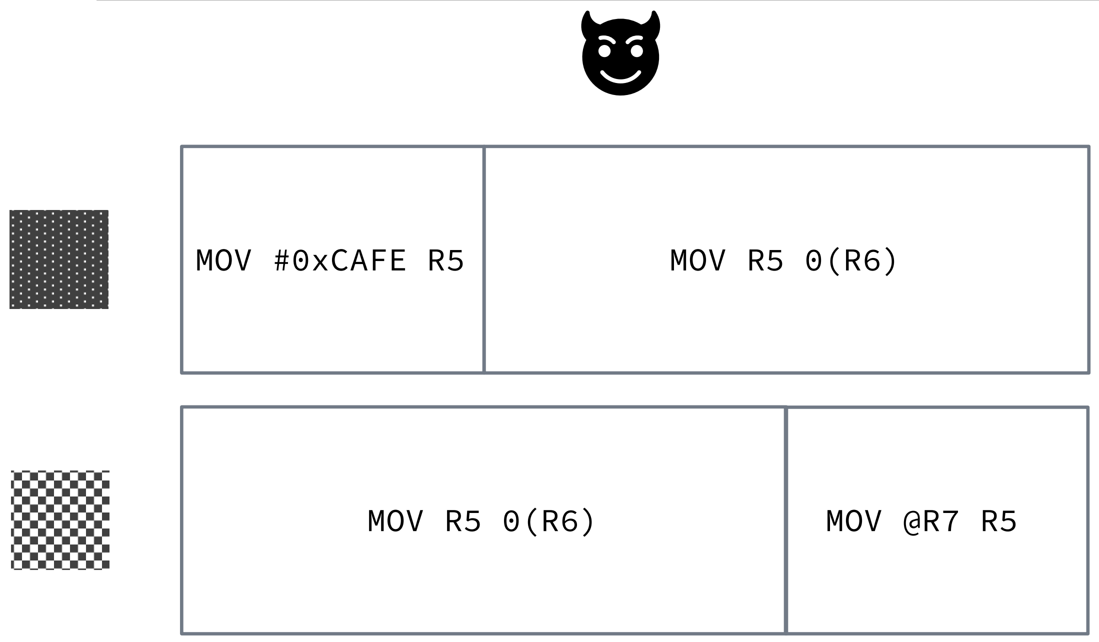
Nemesis attack! [Van Bulck et al., CCS’18]

Why no interruptible enclaves?

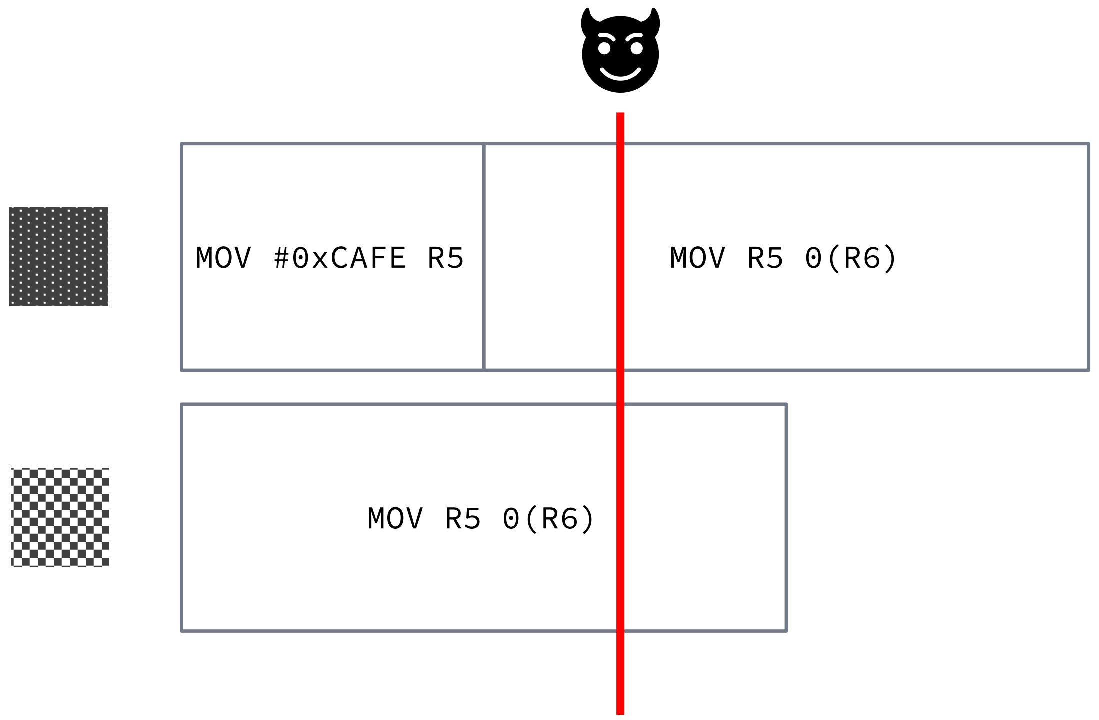


Nemesis attack! [Van Bulck et al., CCS’18]

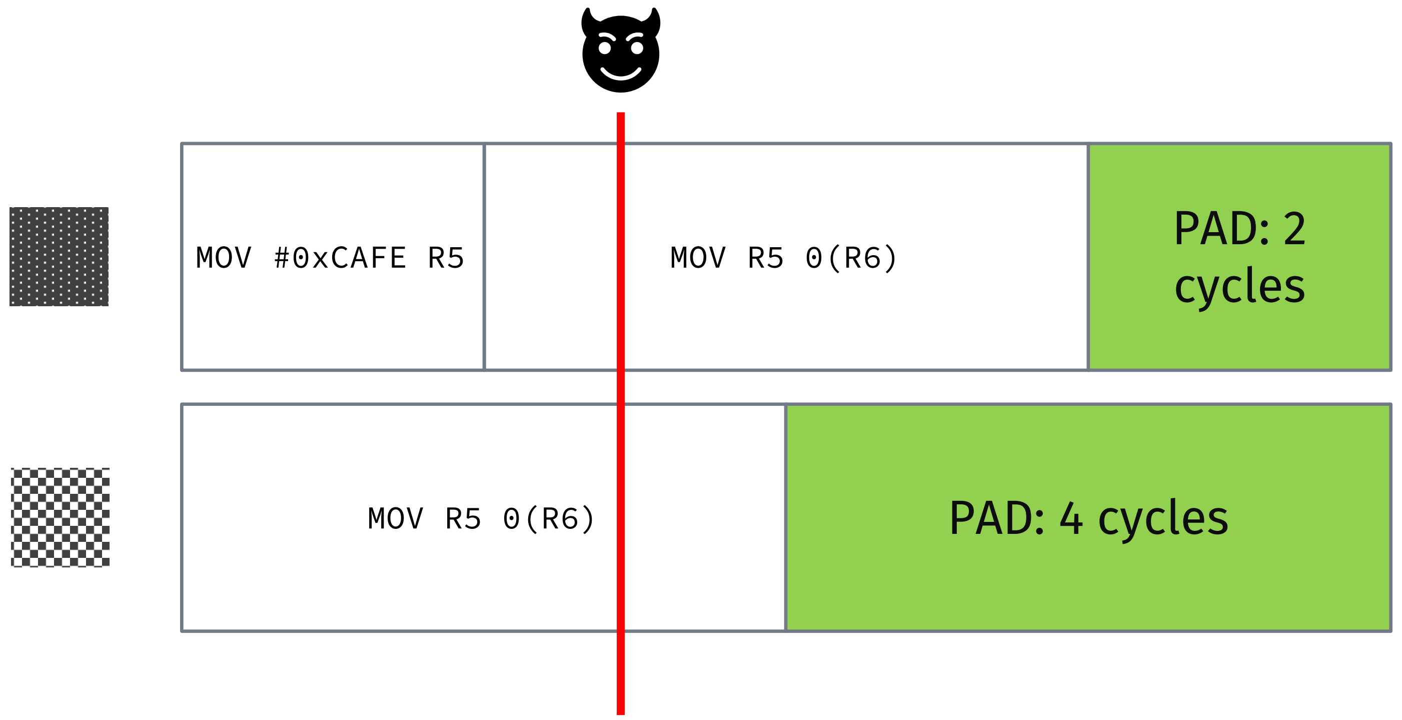
First try: constant delay



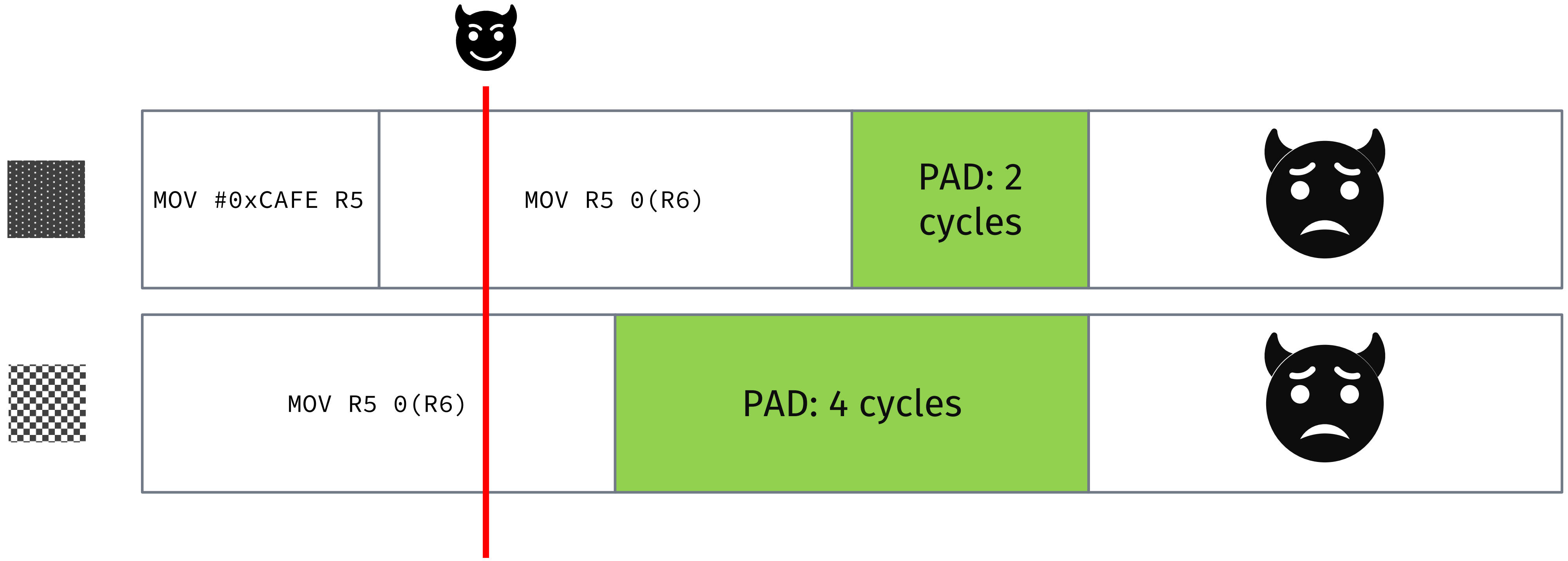
First try: constant delay



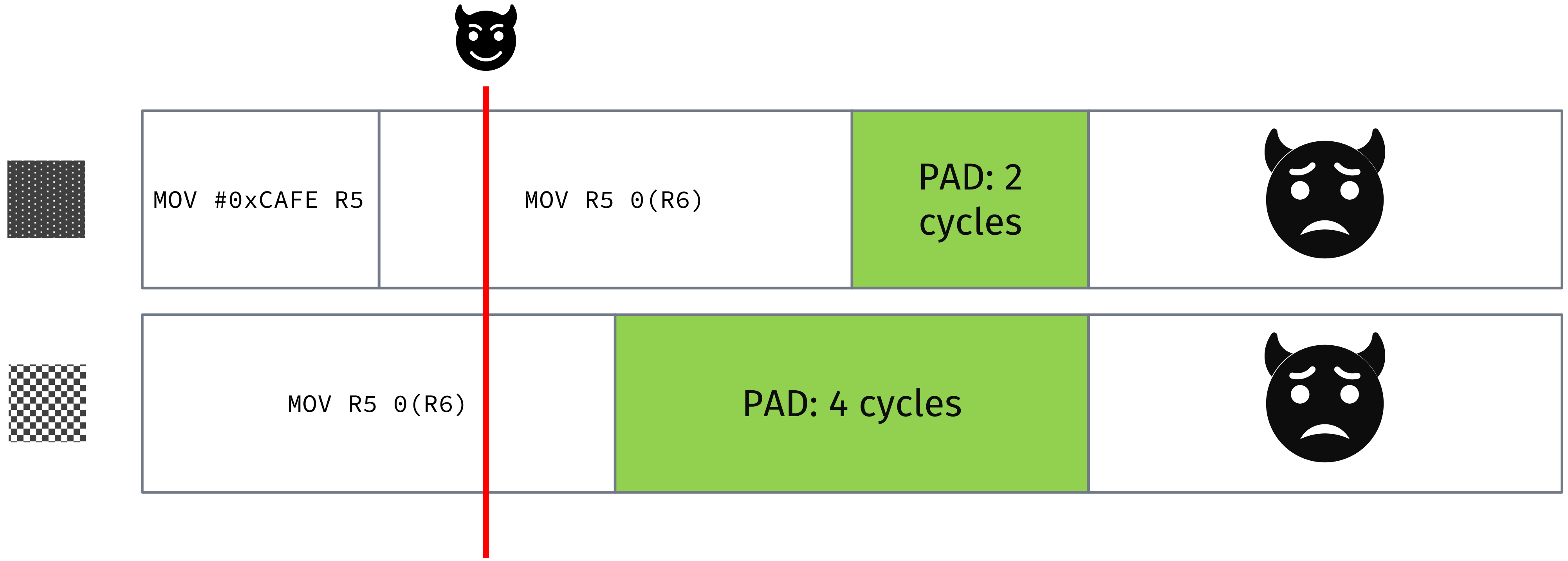
First try: constant delay



First try: constant delay



First try: constant delay



Is Nemesis fixed?

Nope, it is not!

A fair number of details:

Nope, it is not!

A fair number of details:

- “Resume-to-end” attacks: further padding is needed **after interrupt handlers**

Nope, it is not!

A fair number of details:

- “Resume-to-end” attacks: further padding is needed **after interrupt handlers**
- What if an interrupt arrives **during the padding**?

Nope, it is not!

A fair number of details:

- “Resume-to-end” attacks: further padding is needed **after interrupt handlers**
- What if an interrupt arrives **during the padding**?
- What if another interrupt arrives **before the previous have been handled**?

Nope, it is not!

A fair number of details:

- “Resume-to-end” attacks: further padding is needed **after interrupt handlers**
- What if an interrupt arrives **during the padding**?
- What if another interrupt arrives **before the previous have been handled**?
- Can memory be shared between the enclave and the rest of the system?

Nope, it is not!

A fair number of details:

- “Resume-to-end” attacks: further padding is needed **after interrupt handlers**
- What if an interrupt arrives **during the padding**?
- What if another interrupt arrives **before the previous have been handled**?
- Can memory be shared between the enclave and the rest of the system?
- … And a few other subtle cases!

Nope, it is not!

A fair number of details:

- “Resume-to-end” attacks: further padding is needed **after interrupt handlers**
- What if an inte
- What if anothe
- Can memory b
- … And a few o

How do we know we are done? 🤔

Nope, it is not!

A fair number of details:

- "Resume-to-end" attacks: further padding is needed **after interrupt handlers**
- What if an inte
- What if anothe
- Can memory b
- … And a few o

How do we know we are done? 😰

1. Model Sancus as **H** and **L**
2. Prove **full abstraction**, i.e., preservation + reflection!

Step 1: Sancus as H and L

Step 1: Sancus as H and L

High-language is Sancus^H

Core of Sancus:

- Core of MSP430 ISA
- **Isolation mech.**: One single enclave

Step 1: Sancus as H and L

High-language is Sancus^H

Core of Sancus:

- Core of MSP430 ISA
- **Isolation mech.**: One single enclave

Low-language is Sancus^L

Sancus^H +
Interrupts handled in constant-time inside enclaves

Step 1: Sancus as H and L

High-language is Sancus^H

Core of Sancus:

- Core of MSP430 ISA
- **Isolation mech.**: One single enclave

Low-language is Sancus^L

Sancus^H +

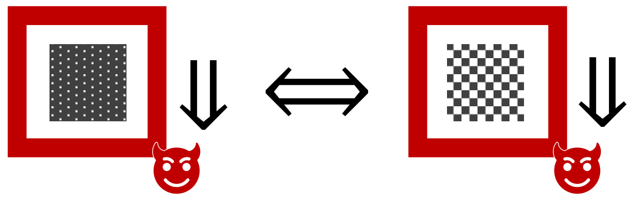
Interrupts handled in constant-time inside enclaves

Attackers:

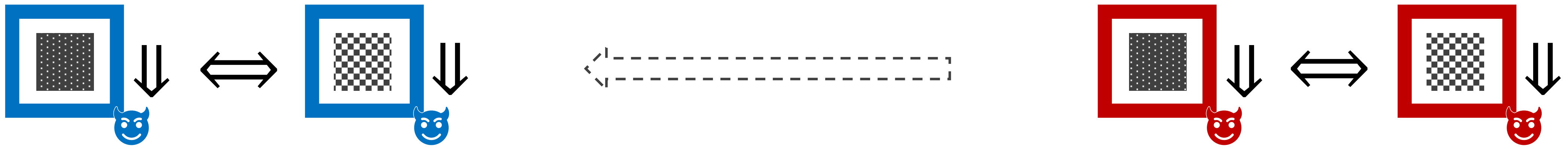
- **memory outside enclave**, including ISR
- **I/O device** for raising interrupts/counting cycles/…

Step 2: full abstraction, reflection

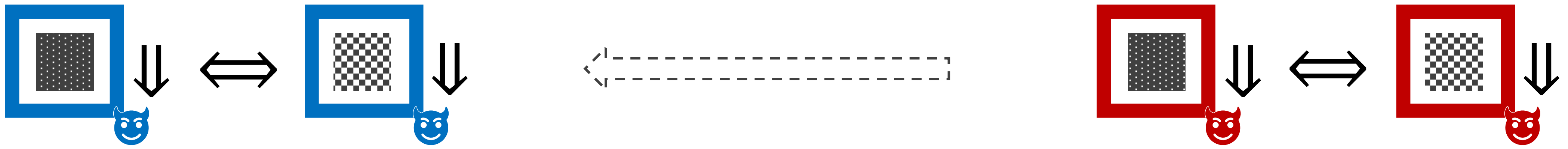
Step 2: full abstraction, reflection



Step 2: full abstraction, reflection

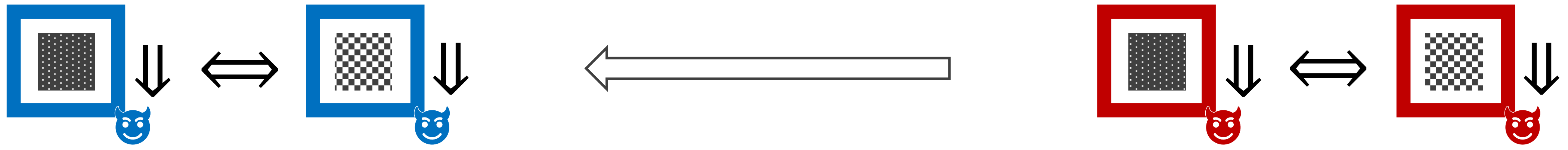


Step 2: full abstraction, reflection



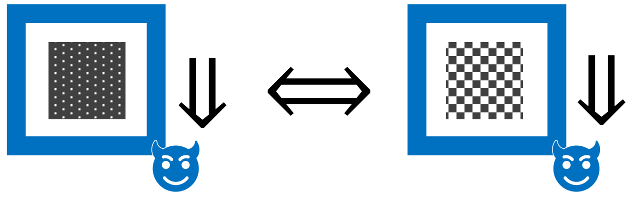
- This is the easy part!

Step 2: full abstraction, reflection

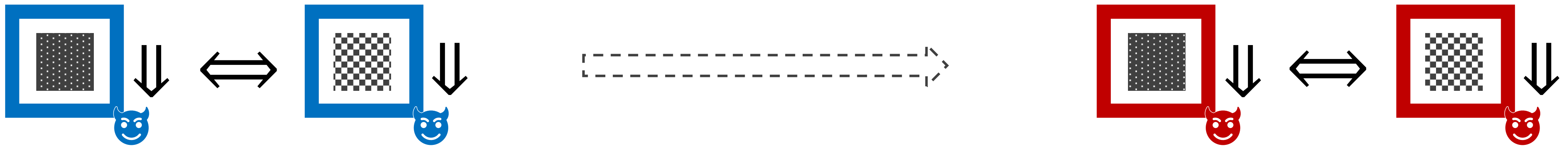


- This is the easy part!
- Attackers in Sancus^{H} ⊆ Attackers in Sancus^{L}

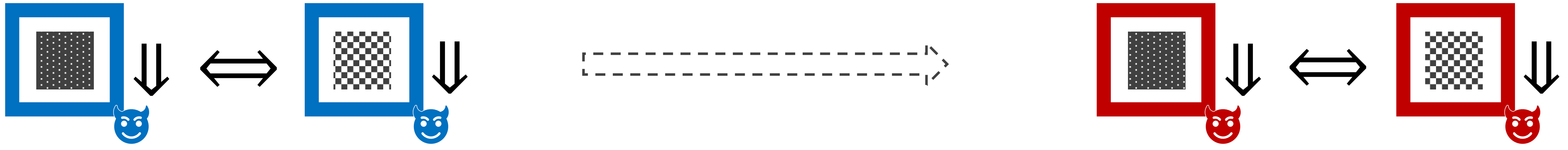
Step 2: full abstraction, preservation

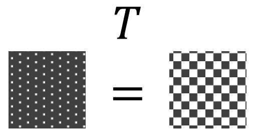


Step 2: full abstraction, preservation



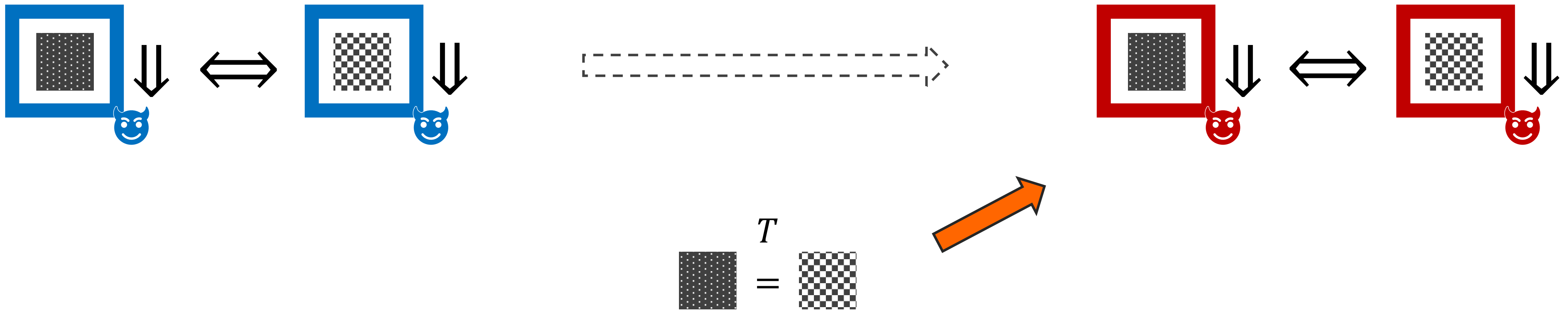
Step 2: full abstraction, preservation





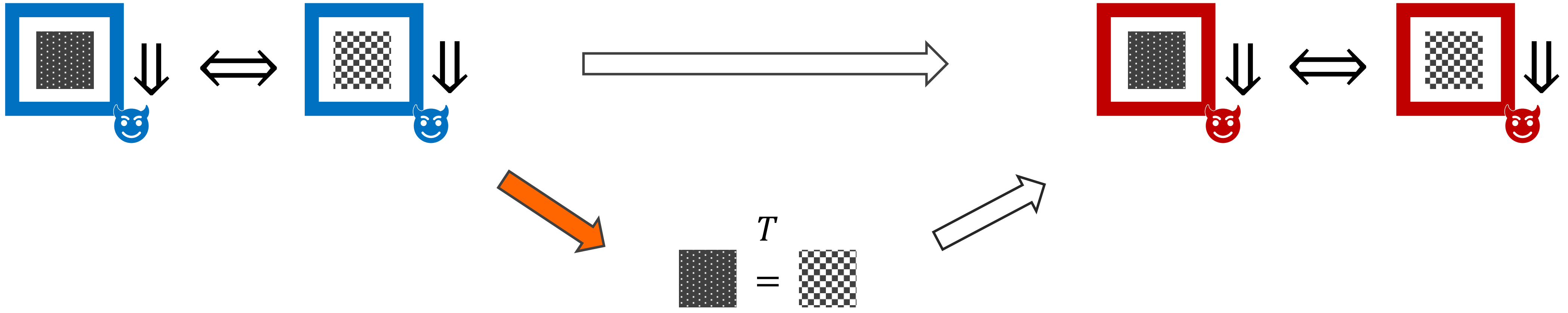
Notion of observable behavior in Sancus^{L}:
traces and trace equivalence

Step 2: full abstraction, preservation



- Trace equivalence ⇒ no Sancus^L attacker distinguishes the two programs
- This amounts to show that our **mitigations are enough**!

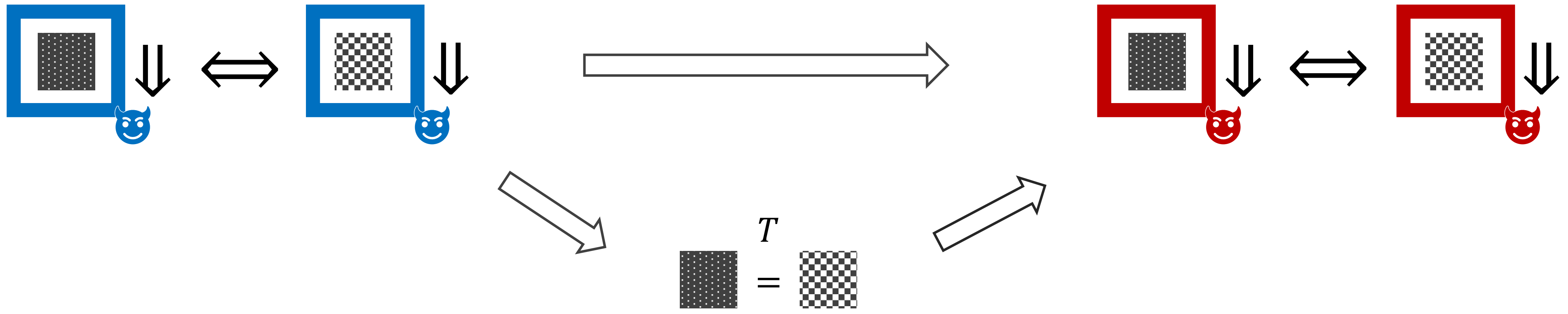
Step 2: full abstraction, preservation

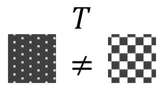
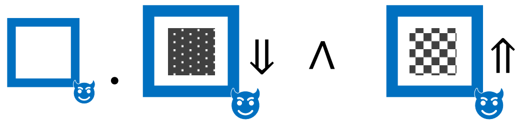


$\Downarrow$ $\Longleftrightarrow$ $\Downarrow$
$\Downarrow$ $\Longleftrightarrow$ $\Downarrow$
$\overset{T}{=}$

- Contrapositive: ▩ $\overset{T}{\neq}$ ▨ implies $\exists$ ☐ . ▩ $\Downarrow$ $\wedge$ ▨ $\Uparrow$ (+ symm)
- Proof by **backtranslation**:
  - Given a witness of non-trace equality, we build a witness of a source attack
  - Source attackers have fixed memory, traces are not limited:
    - Attacker strategy encoded in the I/O device!

Step 2: full abstraction, preservation



- Contrapositive:  implies ∃  (+ symm)
- Proof by **backtranslation**:
  - Given a witness of non-trace equality, we build a witness of a source attack
  - Source attackers have fixed memory, traces are not limited:
    - Attacker strategy encoded in the I/O device!

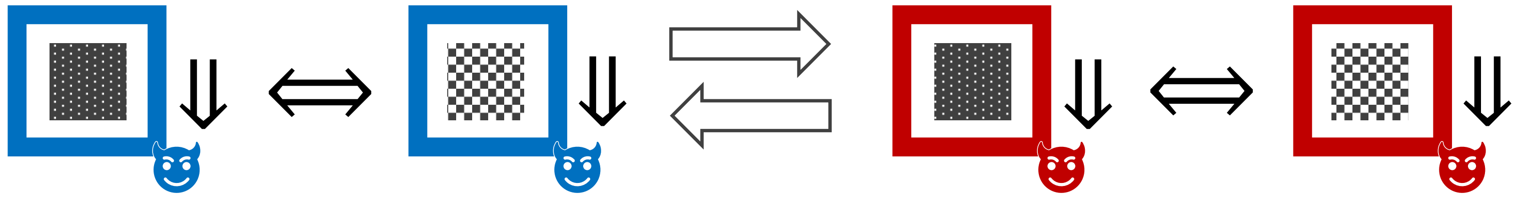
Full abstraction gives you more... 🙃

For free: preservation of robust ⇓-sensitive/⏱-sensitive non-interference:
- Standard, well-studied notion in secure compilation
- Easy **Corollary** of full abstraction!

Other notions of robust non-interference preservation:
- **⇓-insensitive/⏱-sensitive**: corollary of full abstraction + HP of equiconv. in Sancus^{H}
- **stepwise ⇓-sensitive/⏱-sensitive**: for free as corollary of FA!
- **⏱-insensitive:** not meaningful (we know our attacker measures time!)

Conclusions

- **Initial question:** is there a way to add processor features securely while keeping backwards-compatibility?
- **Proposal:** use full abstraction, well-fitted for the scope
- **Our case:** proved that Sancus^H and Sancus^L are fully abstract



Future work

- What about **other features** (e.g., caches, spec. execution, …)?
- Can we make the full abstraction approach **compositional**?
- Can we deal with stronger attackers?
- Also, what about **quantitative** measures of security?

Thanks

Questions?