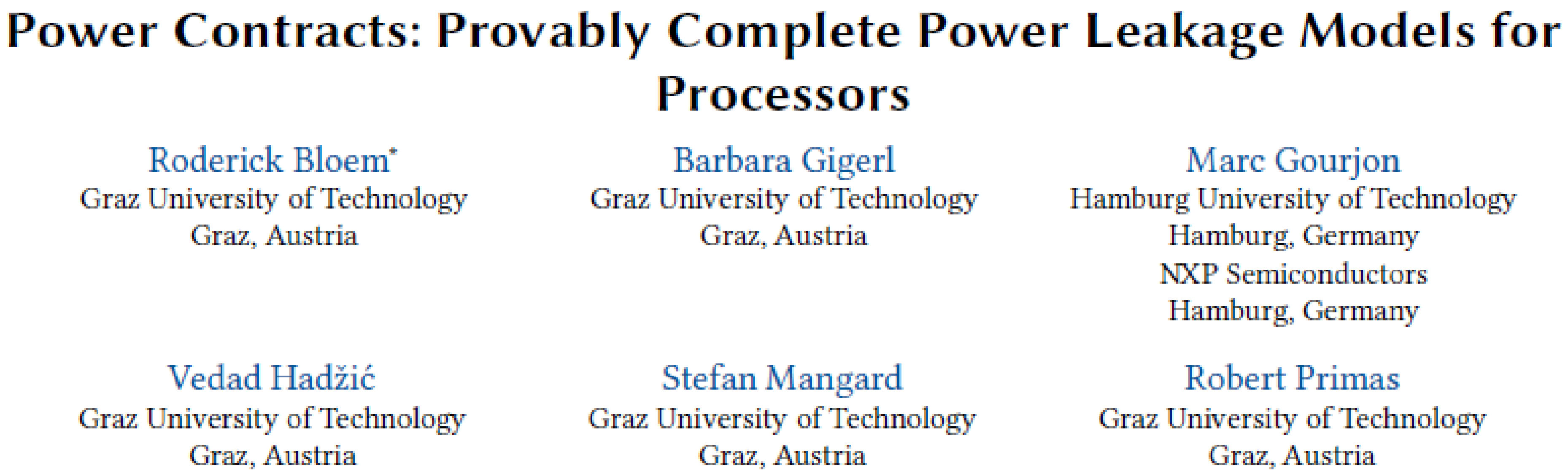
TO APPEAR AT ACM CCS’22



Power Contracts: Provably Complete Power Leakage Models for Processors

Roderick Bloem*
Graz University of Technology
Graz, Austria

Barbara Gigerl
Graz University of Technology
Graz, Austria

Marc Gourjon
Hamburg University of Technology
Hamburg, Germany
NXP Semiconductors
Hamburg, Germany

Vedad Hadžić
Graz University of Technology
Graz, Austria

Stefan Mangard
Graz University of Technology
Graz, Austria

Robert Primas
Graz University of Technology
Graz, Austria

PUT (VERY) SIMPLY
DEPENDABLE PRE-SILICON VERIFICATION

Contract = dependable leakage model

PUT (VERY) SIMPLY
DEPENDABLE PRE-SILICON VERIFICATION

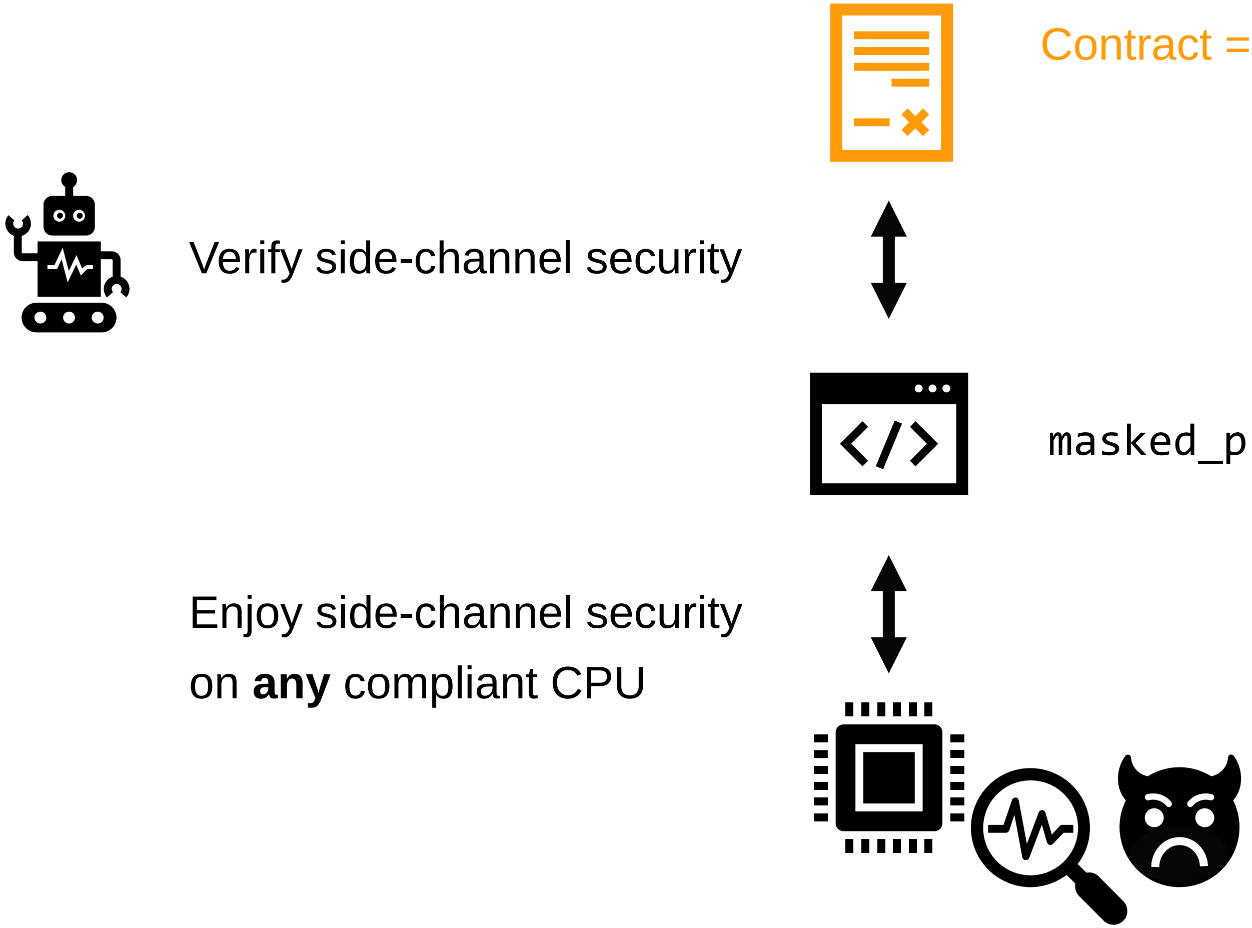
Contract = dependable leakage model

Verify side-channel security

↕

`masked_program.s`

PUT (VERY) SIMPLY
DEPENDABLE PRE-SILICON VERIFICATION



Contract = dependable leakage model

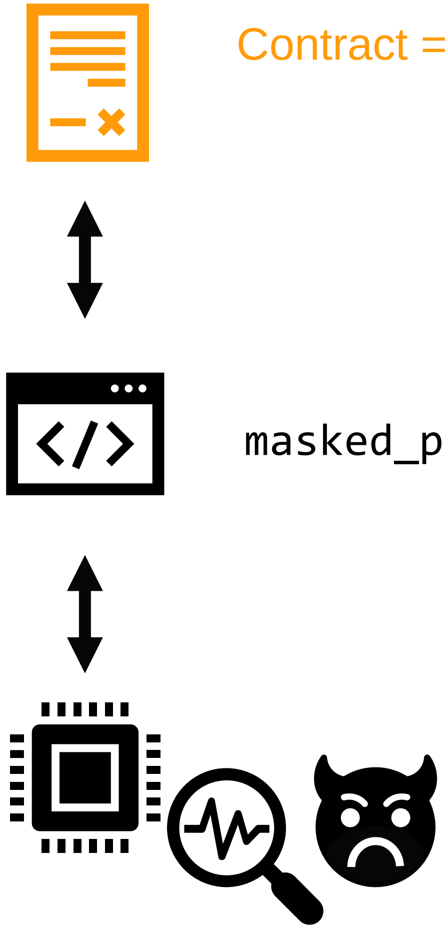
Verify side-channel security

`masked_program.s`

Enjoy side-channel security
on **any** compliant CPU

gate-level probing adversary guaranteed to fail

PUT (VERY) SIMPLY
DEPENDABLE PRE-SILICON VERIFICATION



Contract = dependable
leakage model

Verify side-channel security

`masked_program.s`

Enjoy side-channel security
on **any** compliant CPU

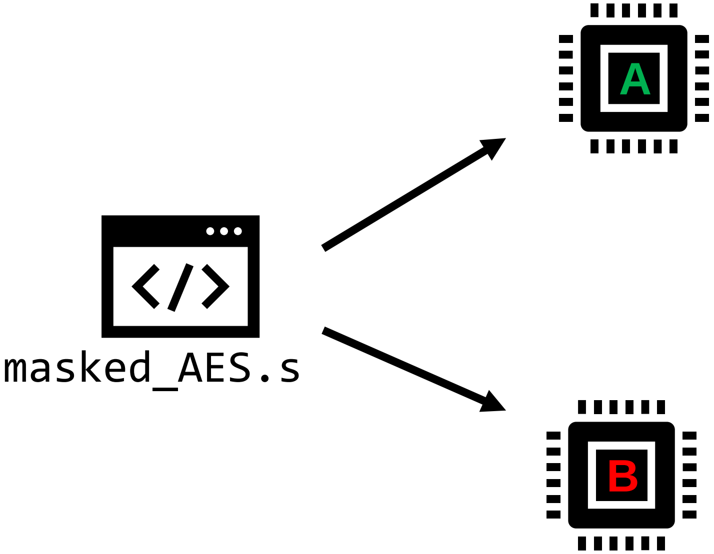
gate-level probing
adversary guaranteed to
fail

In this talk

- **Contracts**
  More precise modeling of leakage
- **HW Compliance**
  Proving completeness of leakage model

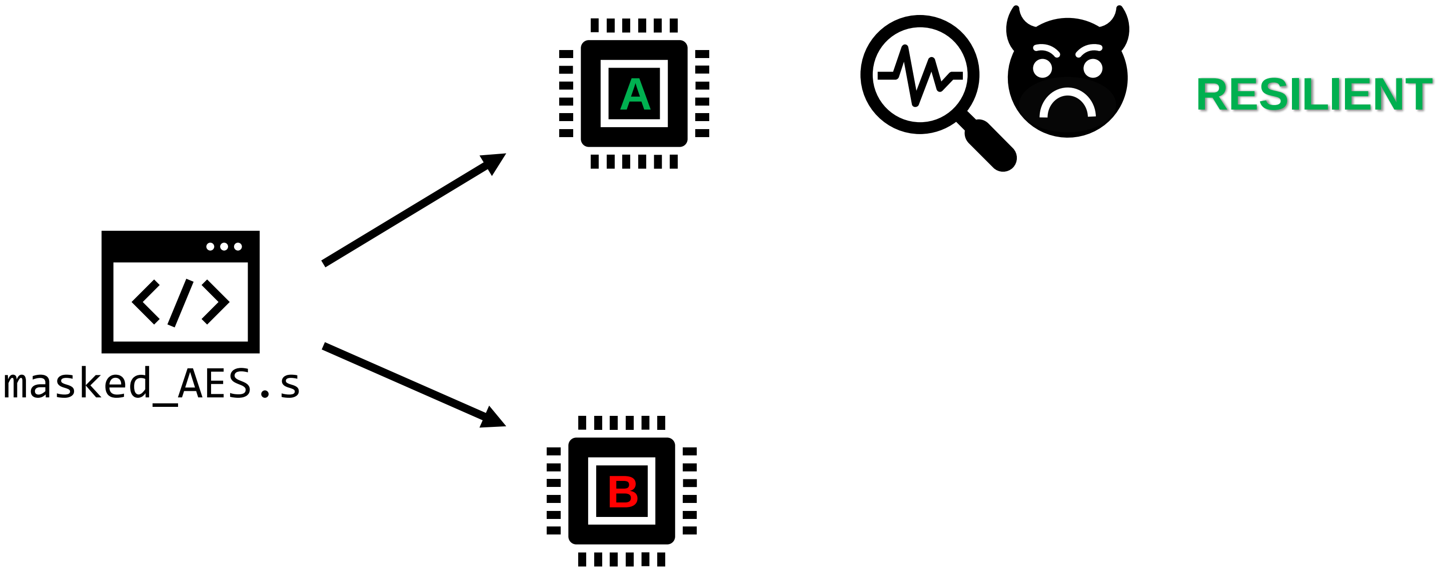
DEVICE-SPECIFIC LEAKAGE (1)

Problem: Same program has different



DEVICE-SPECIFIC LEAKAGE (1)

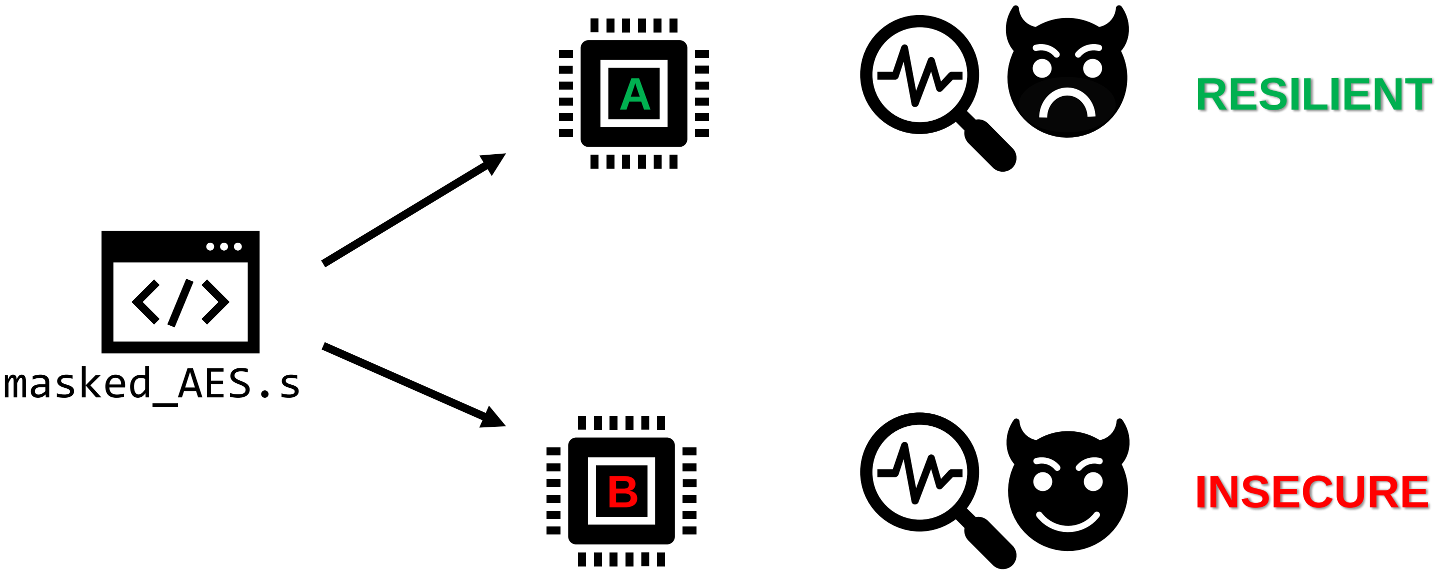
Problem: Same program has different



RESILIENT

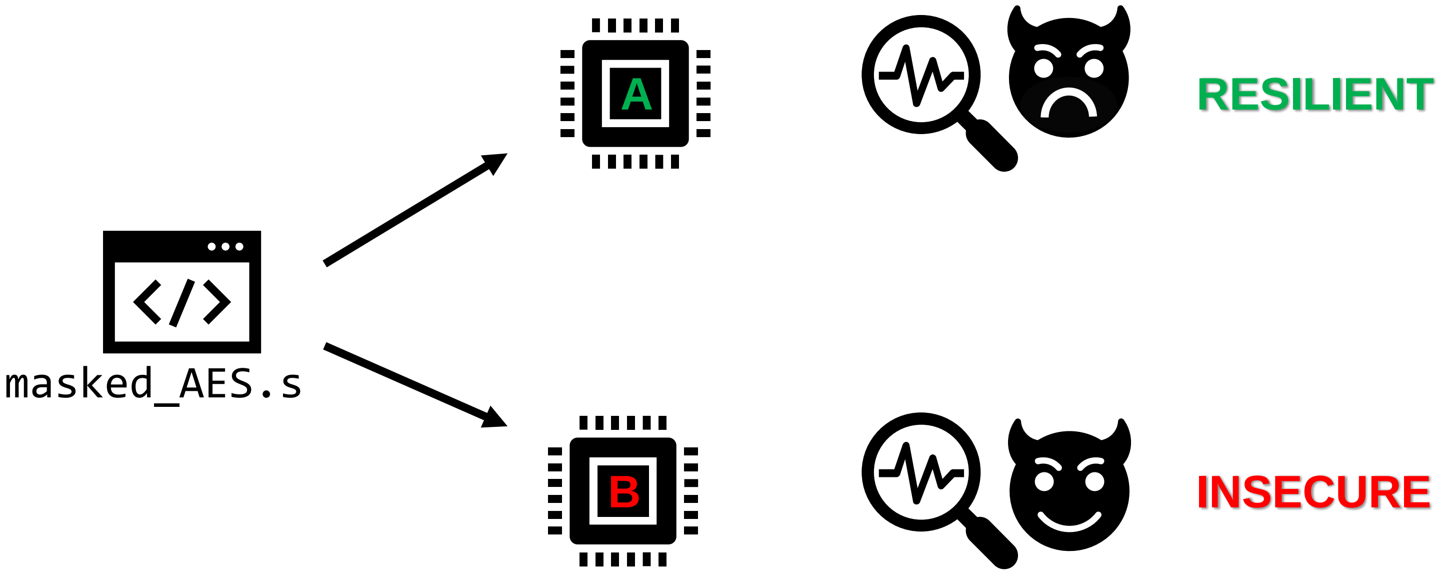
DEVICE-SPECIFIC LEAKAGE (1)

Problem: Same program has different



DEVICE-SPECIFIC LEAKAGE (1)

Problem: Same program has different

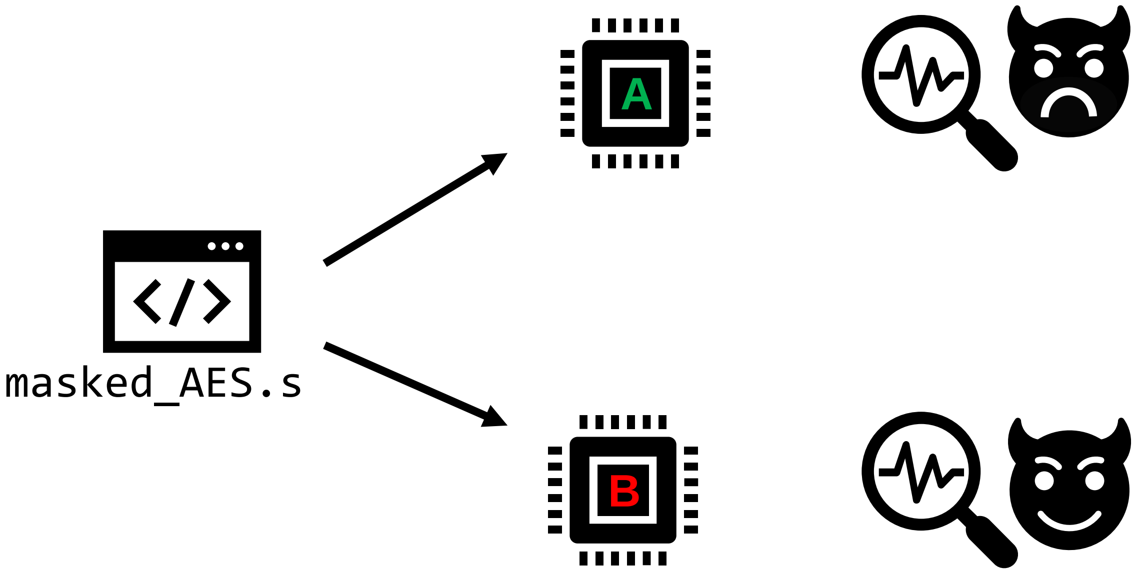


```
…
xor x1, x2, x3
and x4, x5, x6
…
```

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

DEVICE-SPECIFIC LEAKAGE (1)

Problem: Same program has different



RESILIENT

```
HD(x2, x3)
HD(x5, x6)
```

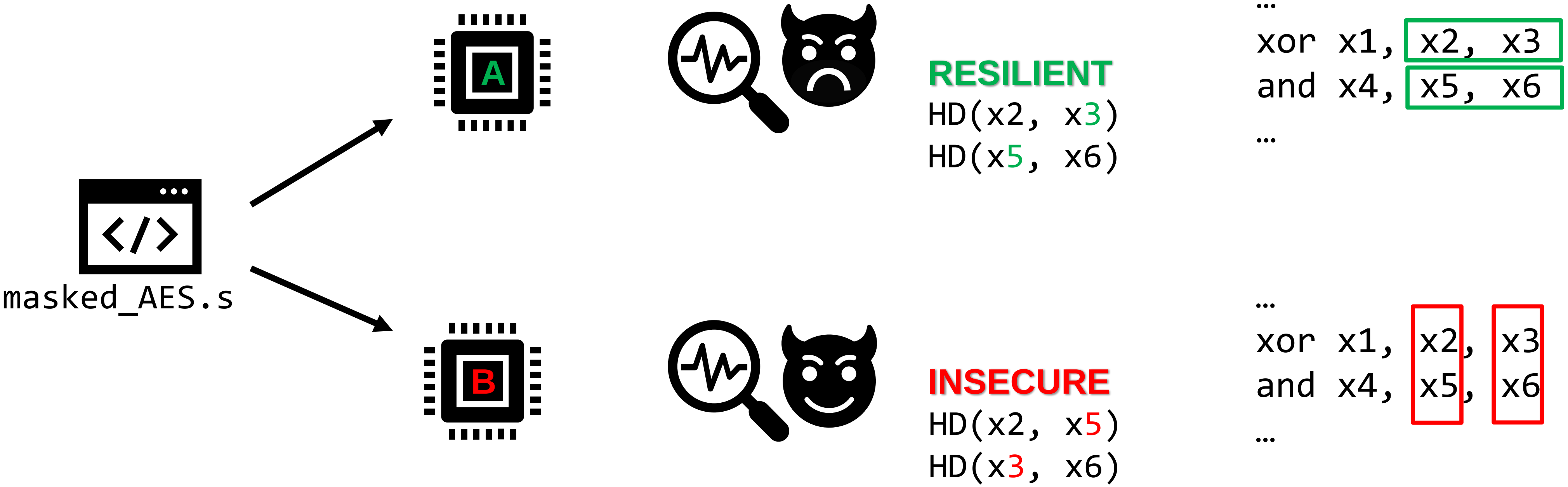
```
…
xor x1, x2, x3
and x4, x5, x6
…
```

INSECURE

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

DEVICE-SPECIFIC LEAKAGE (1)

Problem: Same program has different



`masked_AES.s`

RESILIENT

HD(x2, x3)
HD(x5, x6)

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

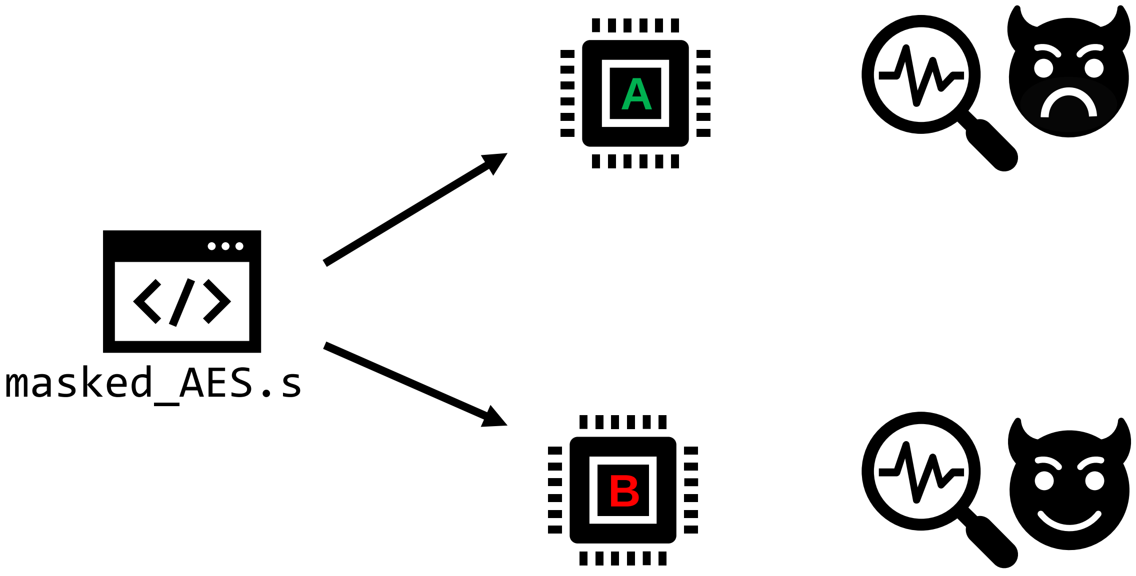
INSECURE

HD(x2, x5)
HD(x3, x6)

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

DEVICE-SPECIFIC LEAKAGE (1)

Problem: Same program has different microarchitecture



masked_AES.s

RESILIENT

HD(x2, x3)
HD(x5, x6)

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

INSECURE

HD(x2, x5)
HD(x3, x6)

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

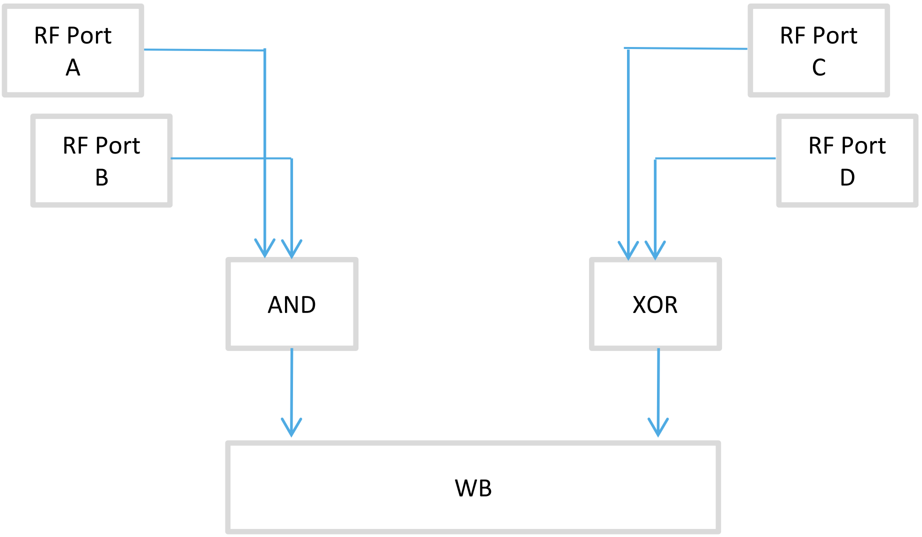
Cause: Processor’s implementation → microarchitecture

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

B

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

A



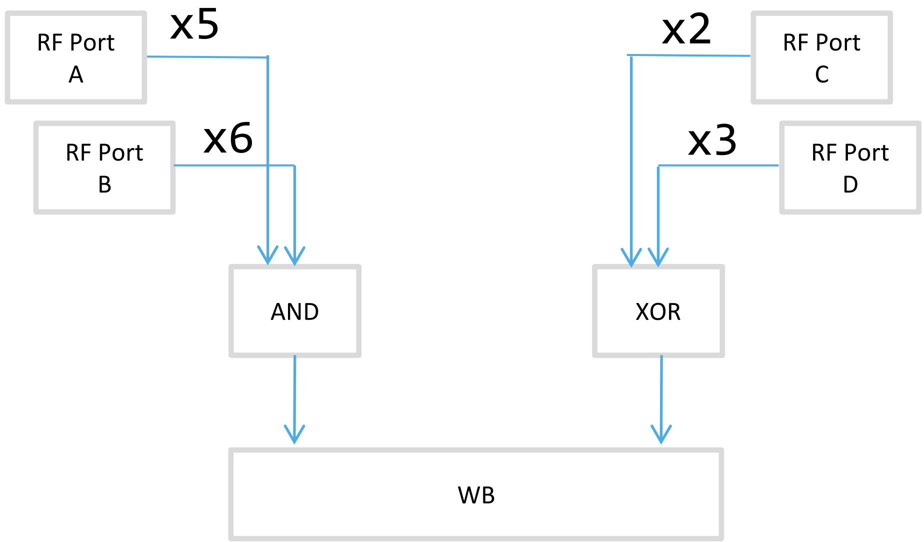
Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert. CHES 2018.

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

B

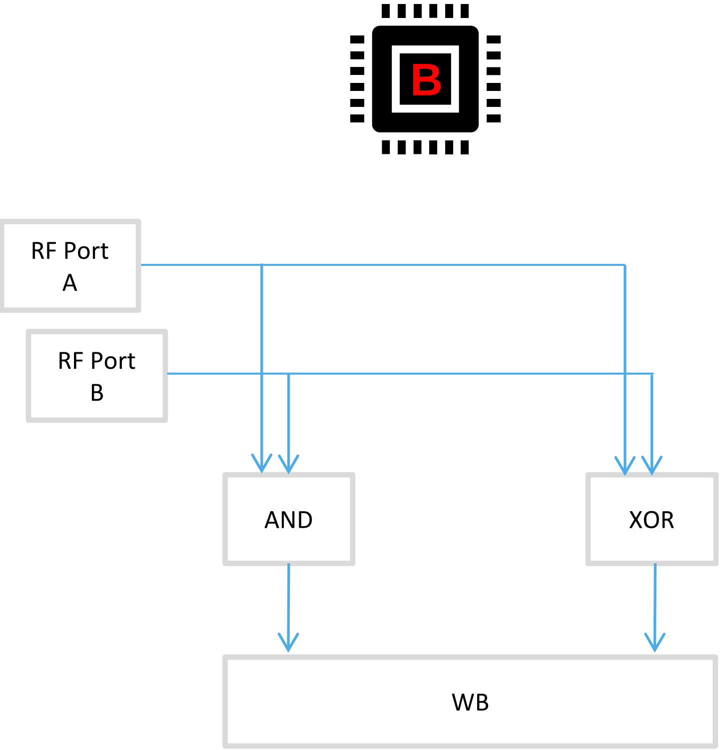
```
…
xor x1, x2, x3
and x4, x5, x6
…
```

A

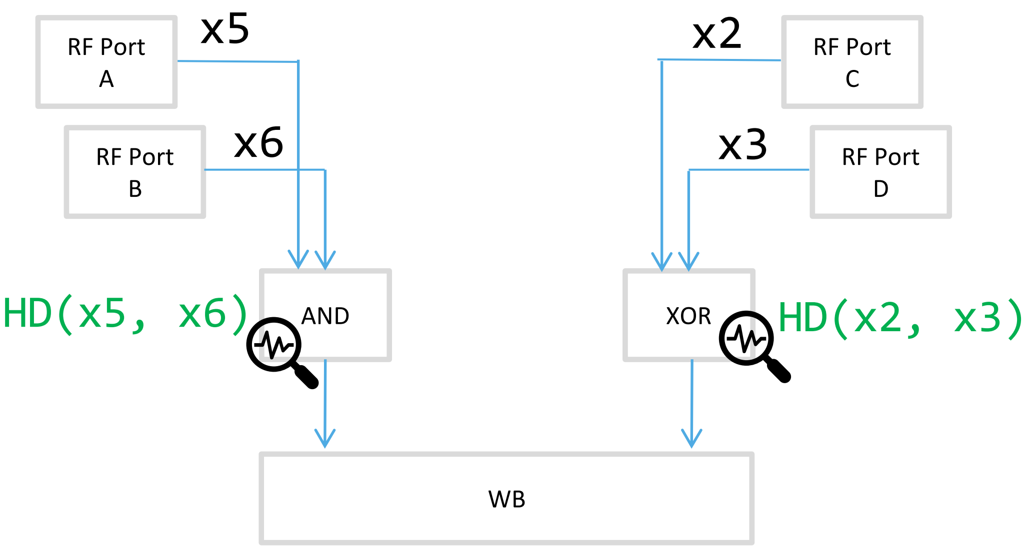


Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert. CHES 2018.

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE



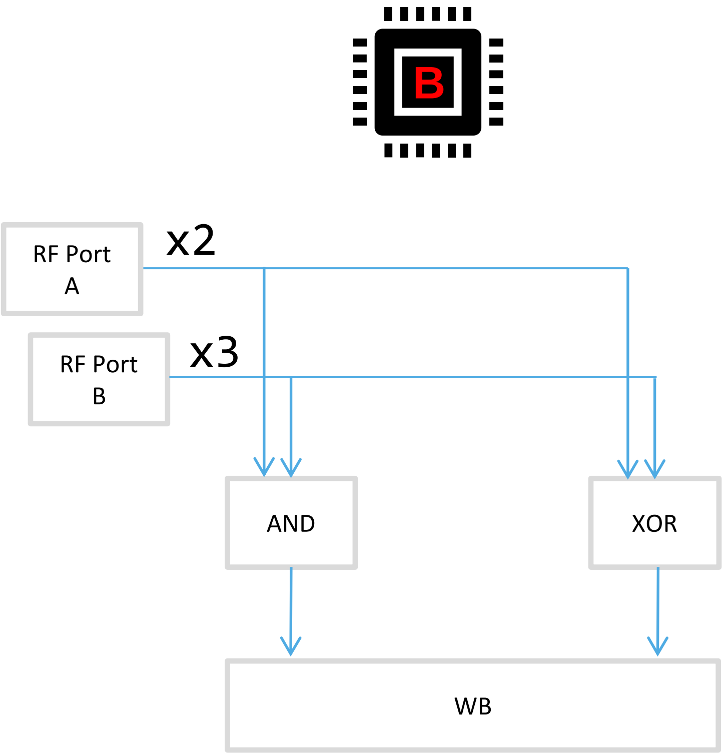
```
…
xor x1, x2, x3
and x4, x5, x6
…
```

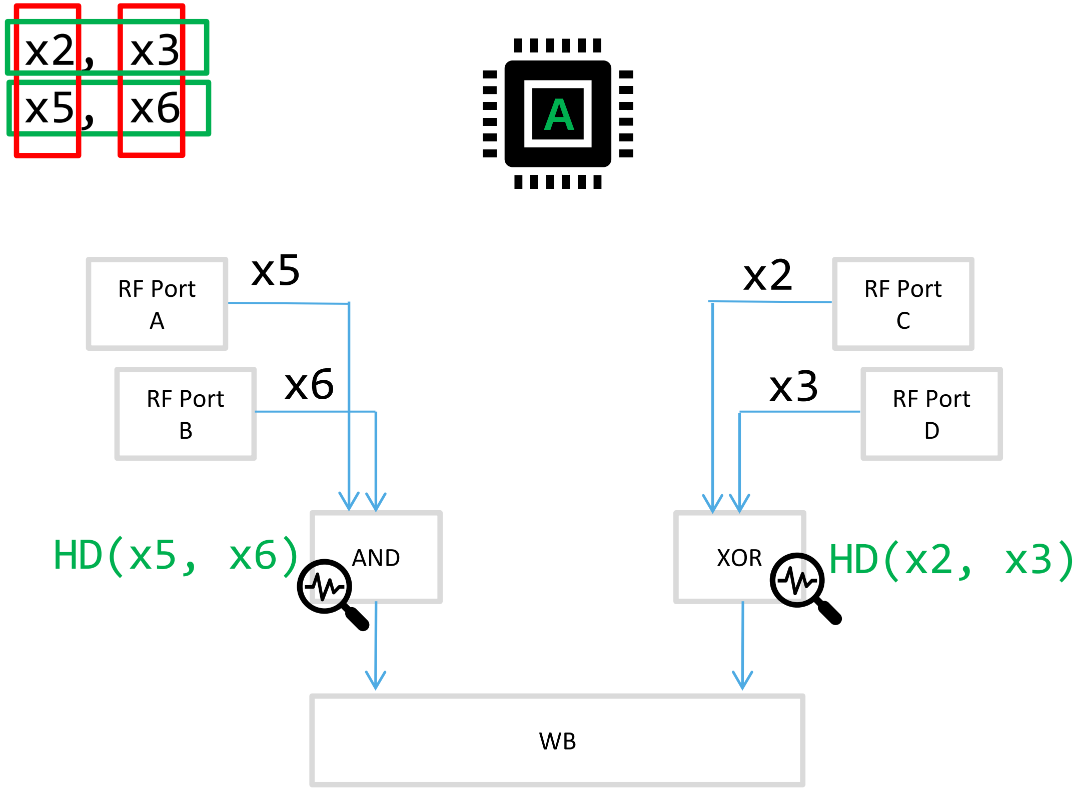


Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert. CHES 2018.

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

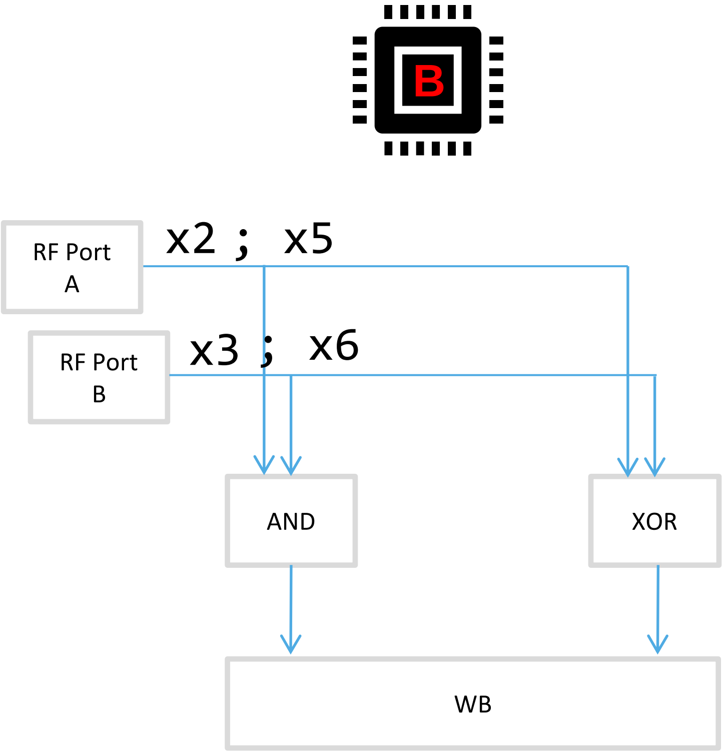


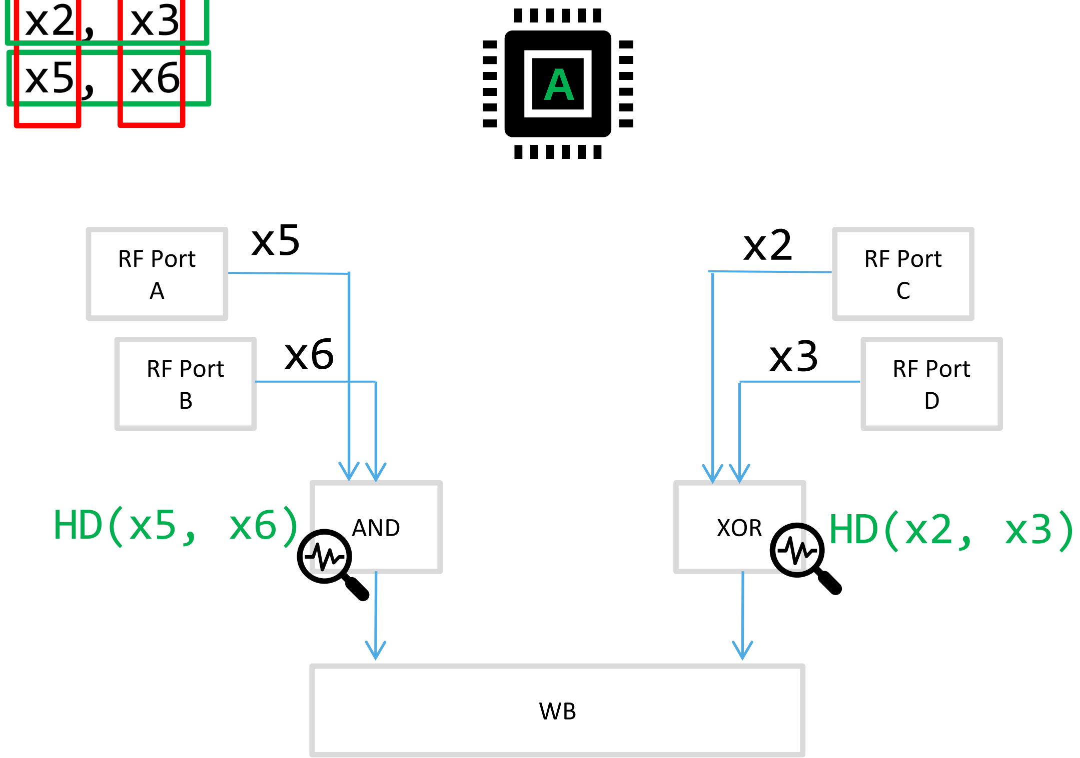


Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert. CHES 2018.

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

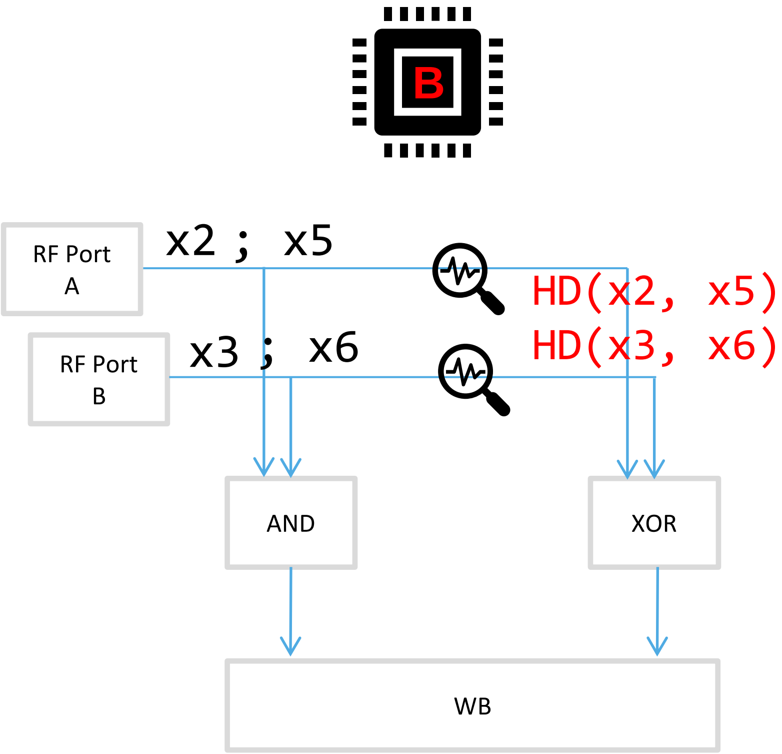


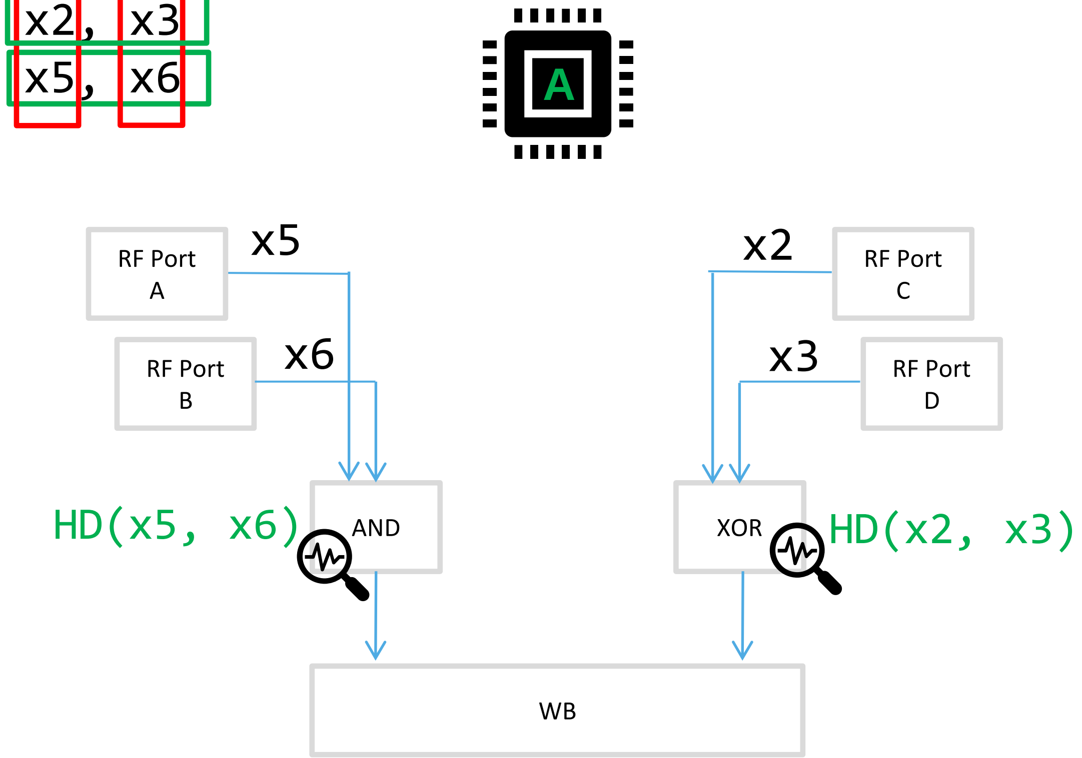


Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert. CHES 2018.

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

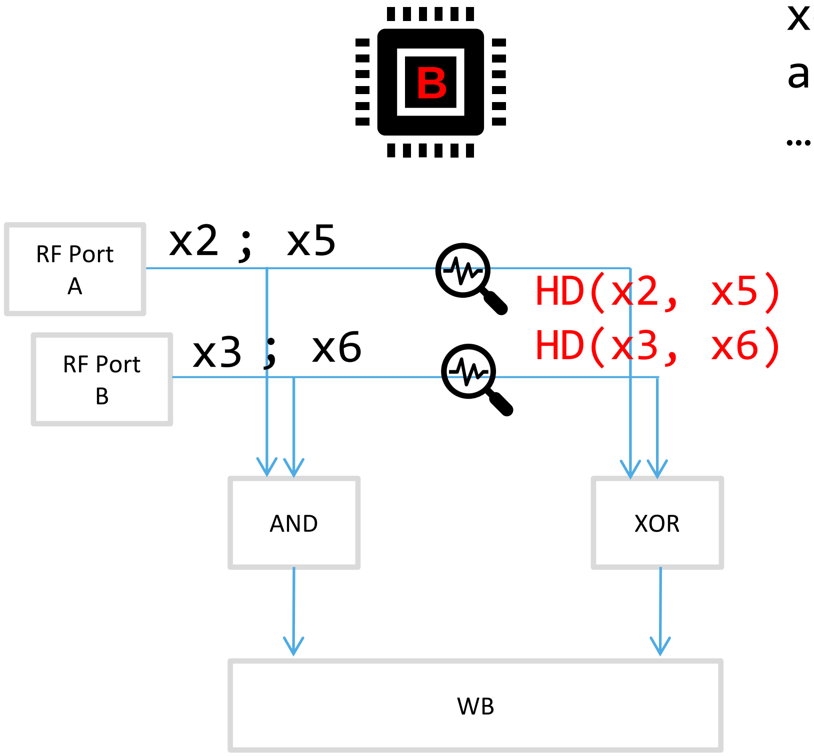


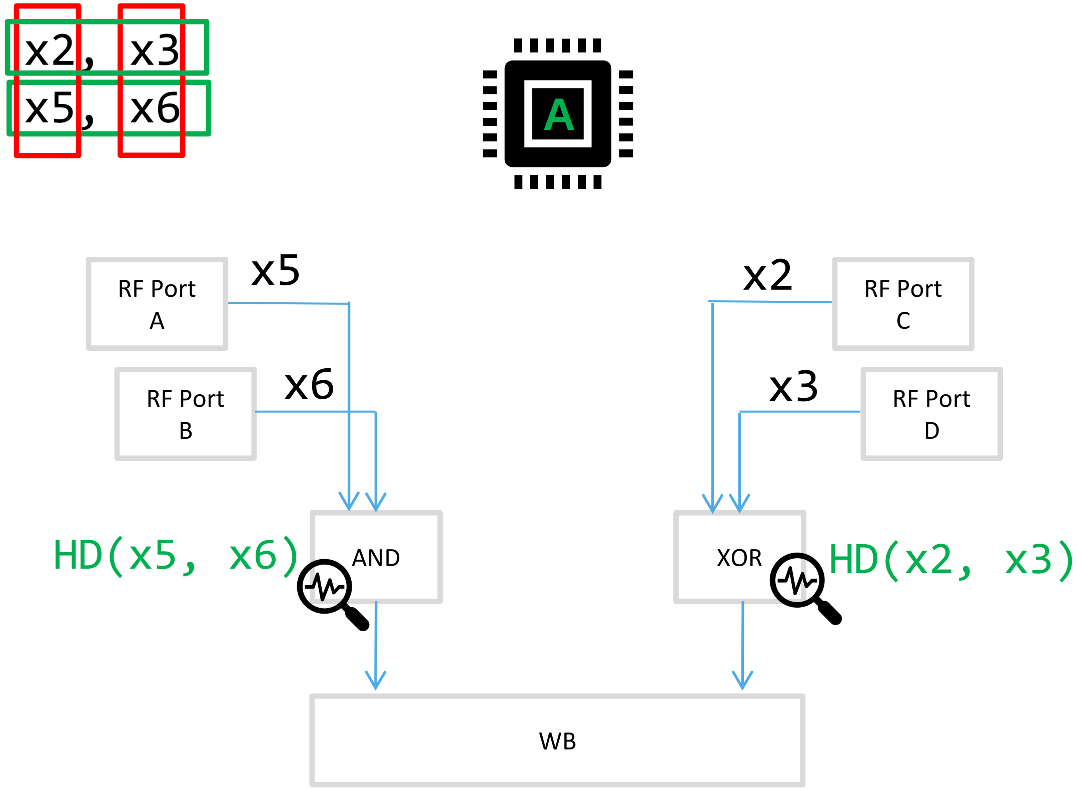


Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert. CHES 2018.

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

```
…
xor x1, x2, x3
and x4, x5, x6
…
```

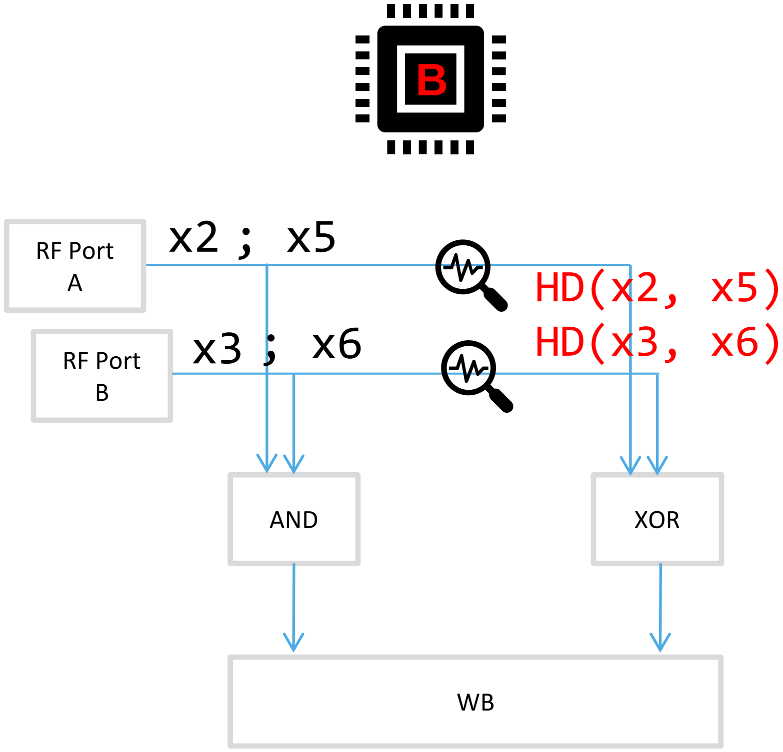


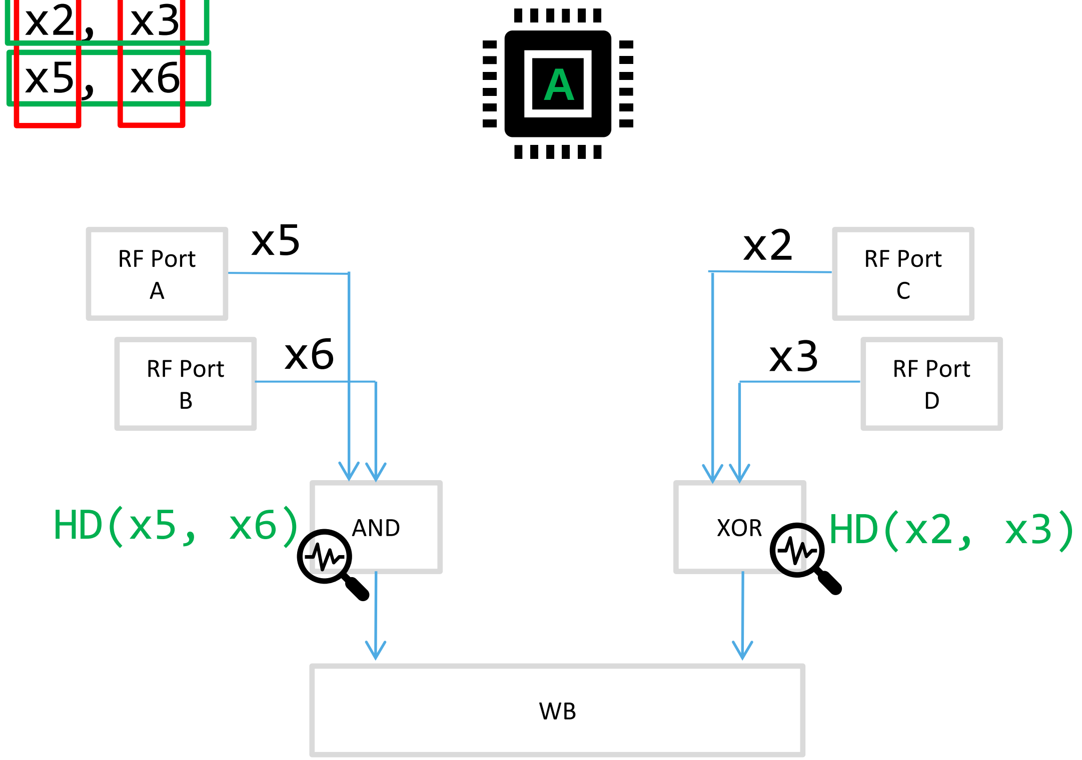


```
xor rD, rN, rM
        leak HD(rN, rM)

and rD, rN, rM
        leak HD(rN, rM)
```

DEVICE-SPECIFIC LEAKAGE (2)
MICROARCHITECTURE

```
…
xor x1, x2, x3
and x4, x5, x6
…
```





```
xor rD, rN, rM
        leak HD(rN, previous(rN))
        leak HD(rM, previous(rM))

and rD, rN, rM
        leak HD(rN, previous(rN))
        leak HD(rM, previous(rM))
```

```
xor rD, rN, rM
        leak HD(rN, rM)

and rD, rN, rM
        leak HD(rN, rM)
```

MODELING LEAKAGE (1)

- Formal leakage model in GENOA := SAIL DSL [1] + `leak` [2]

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);                 // read register rs1
  let rs2_val = X(rs2);                 // read register rs2

  let result = rs1_val ^ rs2_val;      // compute XOR operation





  X(rd) = result;                       // write result to rd
  return RETIRE_SUCCESS
}
```

[2]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

[1]: **ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS.** Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. POPL 2019.

MODELING LEAKAGE (1)

- Formal leakage model in GENOA := SAIL DSL [1] + `leak` [2]

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);             // read register rs1
  let rs2_val = X(rs2);             // read register rs2

  let result = rs1_val ^ rs2_val;   // compute XOR operation

  leak(HD(X(rs1), X(rs2)));         // leakage between operands



  X(rd) = result;                   // write result to rd
  return RETIRE_SUCCESS
}
```

[2]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

[1]: **ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS.** Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. POPL 2019.

MODELING LEAKAGE (1)

- Formal leakage model in GENOA := SAIL DSL [1] + `leak` [2]

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);              // read register rs1
  let rs2_val = X(rs2);              // read register rs2

  let result = rs1_val ^ rs2_val;    // compute XOR operation

  leak(HD(X(rs1), X(rs2)));          // leakage between operands
  leak(   X(rs1), X(rs2) );          // leakage between operands


  X(rd) = result;                    // write result to rd
  return RETIRE_SUCCESS
}
```

[2]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

[1]: **ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS.** Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. POPL 2019.

MODELING LEAKAGE (1)

- Formal leakage model in GENOA := SAIL DSL [1] + `leak` [2]

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);                // read register rs1
  let rs2_val = X(rs2);                // read register rs2

  let result = rs1_val ^ rs2_val;     // compute XOR operation

  leak(HD(X(rs1), X(rs2)));            // leakage between operands
  leak(   X(rs1), X(rs2) );            // leakage between operands
  leak(   X(rd),   result );           // transition leakage, e.g., HD

  X(rd) = result;                      // write result to rd
  return RETIRE_SUCCESS
}
```

[2]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

[1]: **ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS.** Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. POPL 2019.

MODELING LEAKAGE (1)

- Formal leakage model in GENOA := SAIL DSL [1] + `leak` [2]

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);              // read register rs1
  let rs2_val = X(rs2);              // read register rs2

  let result = rs1_val ^ rs2_val;    // compute XOR operation

  leak(HD(X(rs1), X(rs2)));          // leakage between operands
  leak(   X(rs1), X(rs2) );          // leakage between operands
  leak(   X(rd),  result );          // transition leakage, e.g., HD

  X(rd) = result;                    // write result to rd
  return RETIRE_SUCCESS
}
```

core of a *contract*

---

[2]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

[1]: **ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS.** Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. POPL 2019.

MODELING LEAKAGE (2)
LEAKAGE STATE

```
…
xor x1, x2, x3

and x4, x5, x6

…
```

```
// see license in Listing L
// execute a XOR instruction, similar for AND
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;







  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (2)
LEAKAGE STATE

```
…
xor x1, x2, x3

and x4, x5, x6


…
```

```
// see license in Listing L
// execute a XOR instruction, similar for AND
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;




  rf_pA = rs1_val; // leakage state to remember operand 1


  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (2)
LEAKAGE STATE

```
…
xor x1, x2, x3

and x4, x5, x6

…
```

```
// see license in Listing L
// execute a XOR instruction, similar for AND
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(   X(rs1), rf_pA); // leak of rs1 & previous rs1


  rf_pA = rs1_val; // leakage state to remember operand 1


  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (2)
LEAKAGE STATE

```
…
xor x1, x2, x3
        rf_pA = x2
and x4, x5, x6
        leak (x5, rf_pA)
…
```

```
// see license in Listing L
// execute a XOR instruction, similar for AND
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(   X(rs1), rf_pA); // leak of rs1 & previous rs1


  rf_pA = rs1_val; // leakage state to remember operand 1


  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (2)
LEAKAGE STATE

```
…
xor x1, x2, x3
        rf_pA = x2
and x4, x5, x6
        leak (x5, rf_pA)

…
```

```
// see license in Listing L
// execute a XOR instruction, similar for AND
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(   X(rs1), rf_pA); // leak of rs1 & previous rs1
  leak(   X(rs2), rf_pB);

  rf_pA = rs1_val; // leakage state to remember operand 1
  rf_pB = rs2_val;

  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (2)
LEAKAGE STATE

```
…
xor x1, x2, x3
        rf_pA = x2
and x4, x5, x6
        leak (x5, rf_pA)
…
```

B

```
// see license in Listing L
// execute a XOR instruction, similar for AND
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(   X(rs1), rf_pA); // leak of rs1 & previous rs1
  leak(   X(rs2), rf_pB);

  rf_pA = rs1_val; // leakage state to remember operand 1
  rf_pB = rs2_val;

  X(rd) = result;
  return RETIRE_SUCCESS
}
```

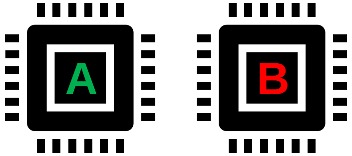
A

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(   X(rs1), X(rs2) );




  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (3)

CONTRACT

- One **contract** for many processors

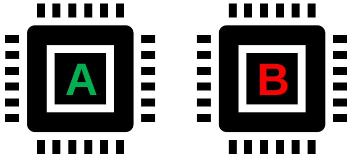


```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(   X(rs1), rf_pA,
          X(rs2), rf_pB);

  rf_pA = rs1_val;
  rf_pB = rs2_val;

  X(rd) = result;
  return RETIRE_SUCCESS
}
```

MODELING LEAKAGE (3)
CONTRACT

- One **contract** for many processors

```
// see license in Listing L
// execute a XOR instruction
execute (XOR(rs2, rs1, rd)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);

  let result = rs1_val ^ rs2_val;

  leak(    X(rs1), rf_pA,
           X(rs2), rf_pB);

  rf_pA = rs1_val;
  rf_pB = rs2_val;

  X(rd) = result;
  return RETIRE_SUCCESS
}
```



```
…
xor x1, x2, x3
and x4, x5, x6
…
```

POWER CONTRACT

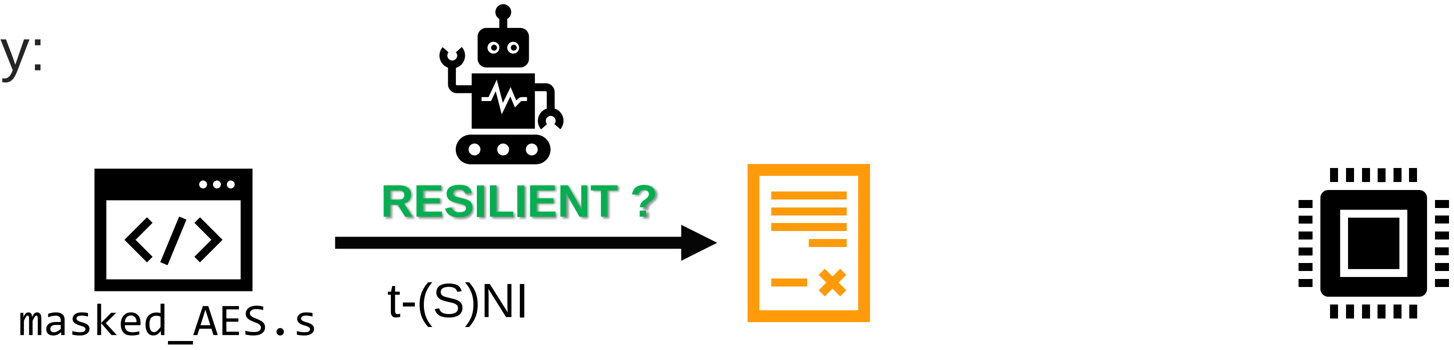
- Contract enables to execute entire programs symbolically
- See License in Listing L

```
function step_ibex (op : bits(32)) -> bool = {
  nextPC = PC + 4;

  let instruction = encdec(op);
  let ret = execute(instruction);

  let success : bool =
    match ret {
      RETIRE_SUCCESS => true,
      RETIRE_FAIL => false
    };
  tick_pc();
  return success
}
function common_leakage(rs1_val, rs2_val) = {
  leak(rs1_val, rs2_val, rf_pA, rf_pB,
  mem_last_addr, mem_last_read);
  rf_pA = rs1_val;
  rf_pB = rs2_val; /* update read ports */
  mem_last_read = 0; /* clear data memory port */
}
```

```
// decode or encode an ADD instruction
// add rd rs1 rs2 ==> RTYPE(rs2, rs1, rd, RISCV_ADD)
mapping clause encdec = RTYPE(rs2, rs1, rd, RISCV_ADD)
  <-> 0b0000000 @ rs2 @ rs1 @ 0b000 @ rd @ 0b0110011

// execute a decoded instruction
function clause execute (RTYPE(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);              // read register rs1
  let rs2_val = X(rs2);

  common_leakage(rs1_val, rs2_val);

  let result = match op {            // match-case
    RISCV_ADD  => rs1_val + rs2_val, // compute ADD operation
          ...
    RISCV_AND  => rs1_val & rs2_val,
  };

  overwrite_leakage(rd, result);

  X(rd) = result;                    // write result to rd
  return RETIRE_SUCCESS
}
```

E2E SECURITY

- E2E Security:



E2E SECURITY

- E2E Security:

`masked_AES.s`

RESILIENT ?

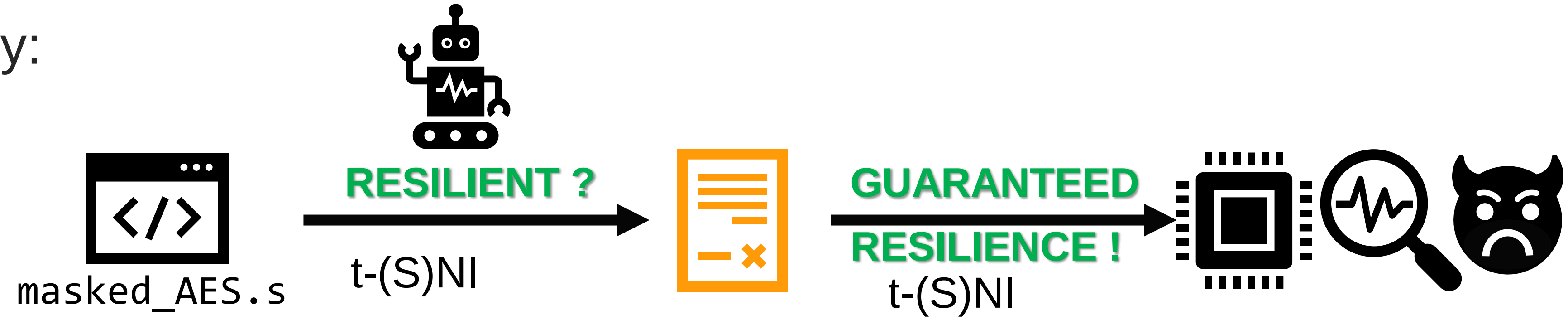
t-(S)NI

GUARANTEED

RESILIENCE !

E2E SECURITY

- E2E Security:



masked_AES.s

RESILIENT ?

t-(S)NI

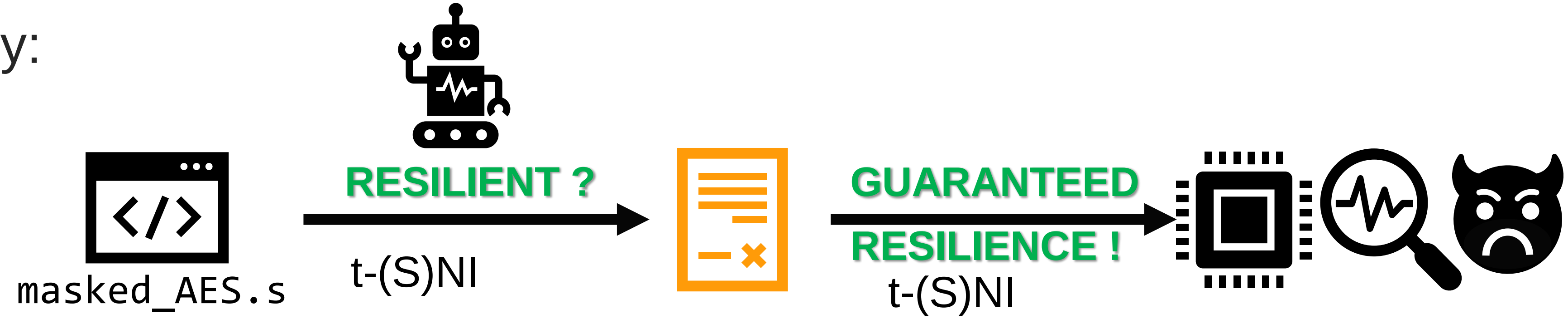
GUARANTEED

RESILIENCE !

t-(S)NI

E2E SECURITY

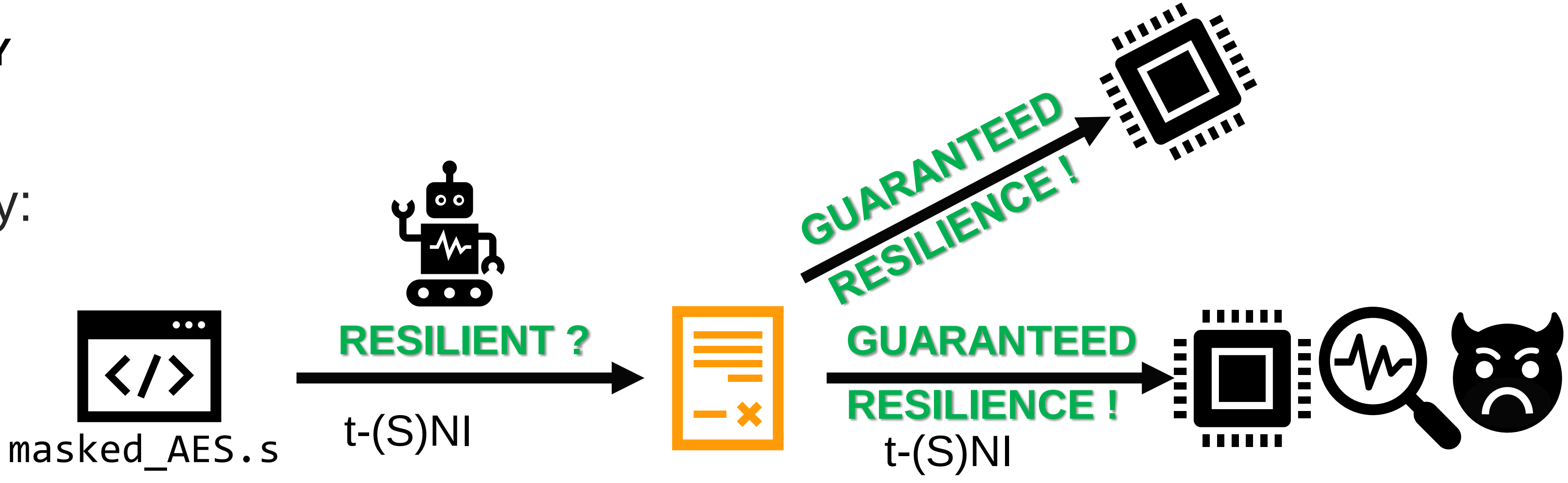
- E2E Security:



- Guarantee of contracts:
  - t-(S)NI @ Contract **implies** t-(S)NI @ **any** compliant HW for **any** program

E2E SECURITY

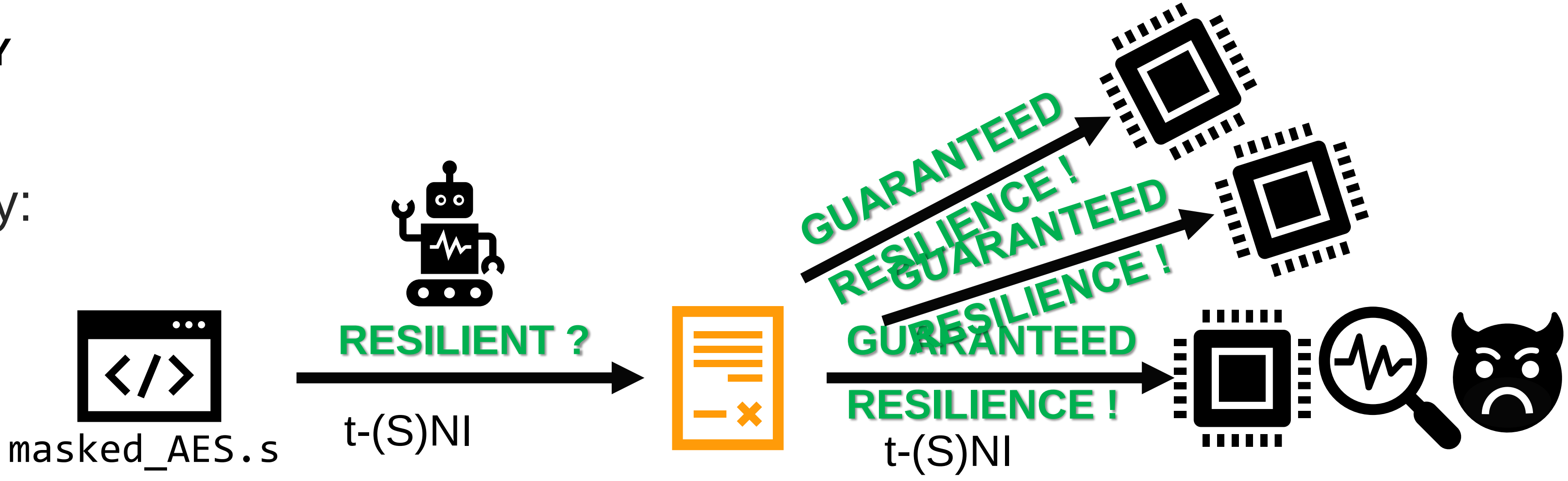
- E2E Security:



- Guarantee of contracts:
  - t-(S)NI @ Contract **implies** t-(S)NI @ **any** compliant HW for **any** program
  - Holds also for threshold probing security, PINI, TI, …

E2E SECURITY

- E2E Security:



masked_AES.s

RESILIENT ?

t-(S)NI

GUARANTEED RESILIENCE !

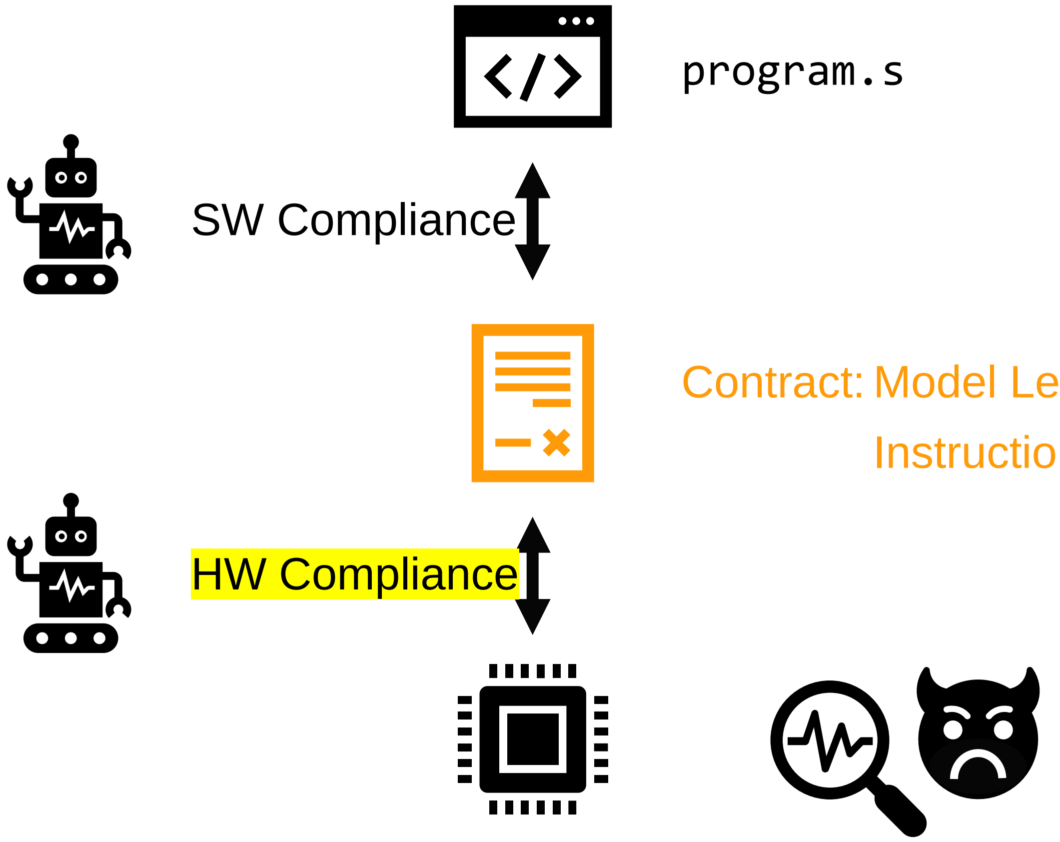
GUARANTEED RESILIENCE !

GUARANTEED RESILIENCE !

t-(S)NI

- Guarantee of contracts:
  - t-(S)NI @ Contract **implies** t-(S)NI @ **any** compliant HW for **any** program
  - Holds also for threshold probing security, PINI, TI, …

BIG PICTURE



`program.s`

`scVerif` [1]

SW Compliance

Contract: Model Leakage +
Instruction Semantic

`coco-genoa`

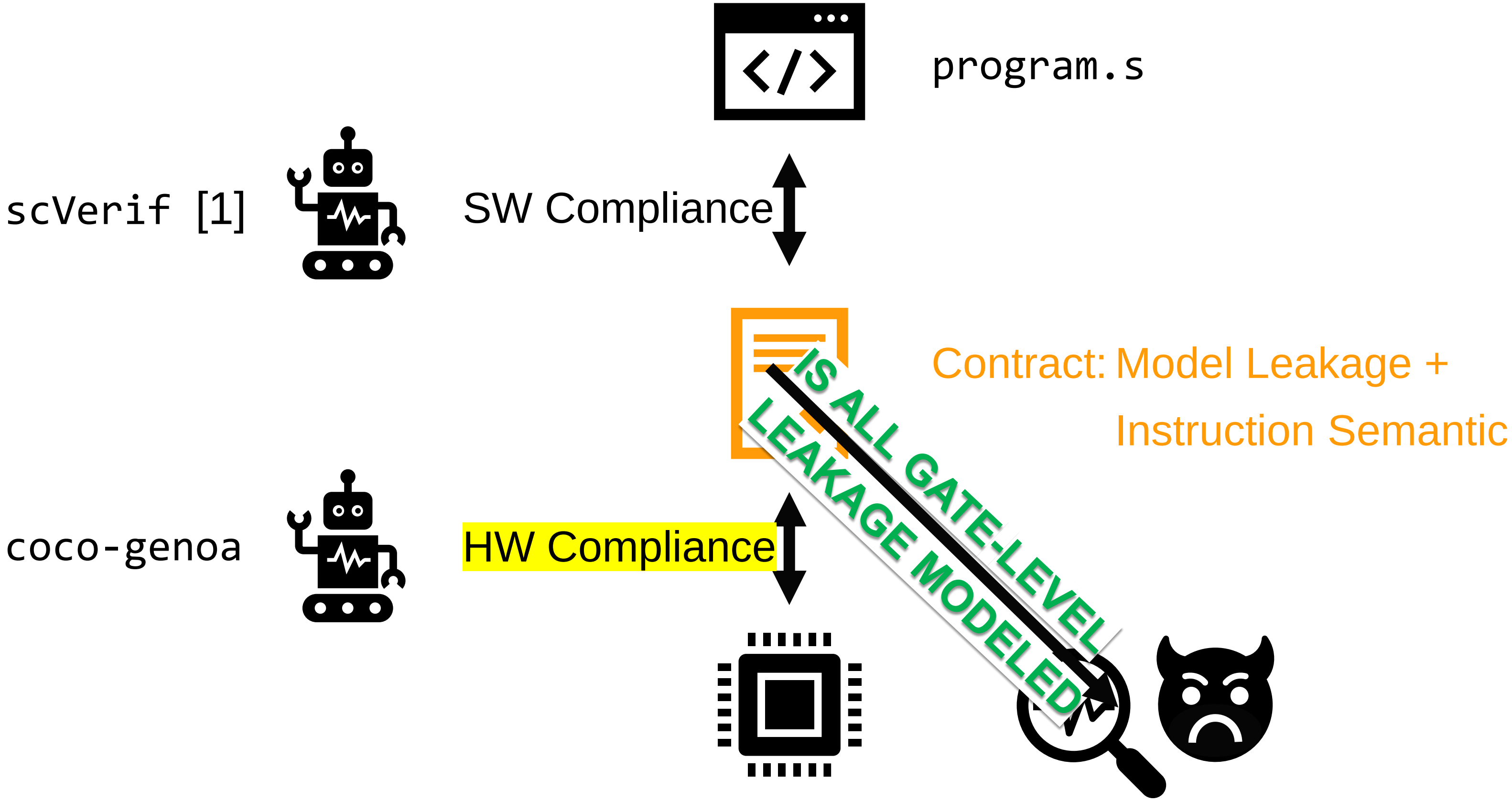
HW Compliance

gate-level probing
adversary
guaranteed to fail

---

[1]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

BIG PICTURE



program.s

scVerif [1]

SW Compliance

Contract: Model Leakage + Instruction Semantic

IS ALL GATE-LEVEL LEAKAGE MODELED

coco-genoa

HW Compliance

gate-level probing adversary guaranteed to fail

[1]: **Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification.** Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, Lars Porth. CHES 2021.

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract

$$[\![\mathrm{P}]\!]^c \left(\sigma_0^c\right)$$

$$[\![\mathrm{P}]\!]^h \left(\sigma_0^h\right)$$

Definition 6 (Compliance: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract

$$[\![\mathrm{P}]\!]^c\left(\sigma_0^c\right): \qquad \sigma_0^c \xrightarrow{\mathcal{L}_0^c} \sigma_1^c \xrightarrow{\mathcal{L}_1^c} \dots \xrightarrow{\mathcal{L}_{n-1}^c} \sigma_n^c$$

$$[\![\mathrm{P}]\!]^h\left(\sigma_0^h\right): \qquad \sigma_0^h \xrightarrow{\color{red}\mathcal{L}_0^h} \sigma_1^h \xrightarrow{\color{red}\mathcal{L}_1^h} \dots \xrightarrow{\color{red}\mathcal{L}_{m-1}^h} \sigma_m^h$$

$\uparrow$

starting state

DEFINITION 6 (COMPLIANCE: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract

$$[\![\mathrm{P}]\!]^{c}\left(\sigma_0^c\right): \quad \sigma_0^c \xrightharpoondown{\mathcal{L}_0^c} \sigma_1^c \xrightharpoondown{\mathcal{L}_1^c} \ldots \xrightharpoondown{\mathcal{L}_{n-1}^c} \sigma_n^c$$

$$[\![\mathrm{P}]\!]^{h}\left(\sigma_0^h\right): \quad \sigma_0^h \xrightharpoonup{\mathcal{L}_0^h} \sigma_1^h \xrightharpoonup{\mathcal{L}_1^h} \ldots \xrightharpoonup{\mathcal{L}_{m-1}^h} \sigma_m^h$$

starting state

final state

DEFINITION 6 (COMPLIANCE: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract

$$[\![\mathrm{P}]\!]^c\left(\sigma_0^c\right): \quad \sigma_0^c \xrightharpoonup{\mathcal{L}_0^c} \sigma_1^c \xrightharpoonup{\mathcal{L}_1^c} \ldots \xrightharpoonup{\mathcal{L}_{n-1}^c} \sigma_n^c$$

$$[\![\mathrm{P}]\!]^h\left(\sigma_0^h\right): \quad \sigma_0^h \xrightharpoondown{\mathcal{L}_0^h} \sigma_1^h \xrightharpoondown{\mathcal{L}_1^h} \ldots \xrightharpoondown{\mathcal{L}_{m-1}^h} \sigma_m^h$$

starting state

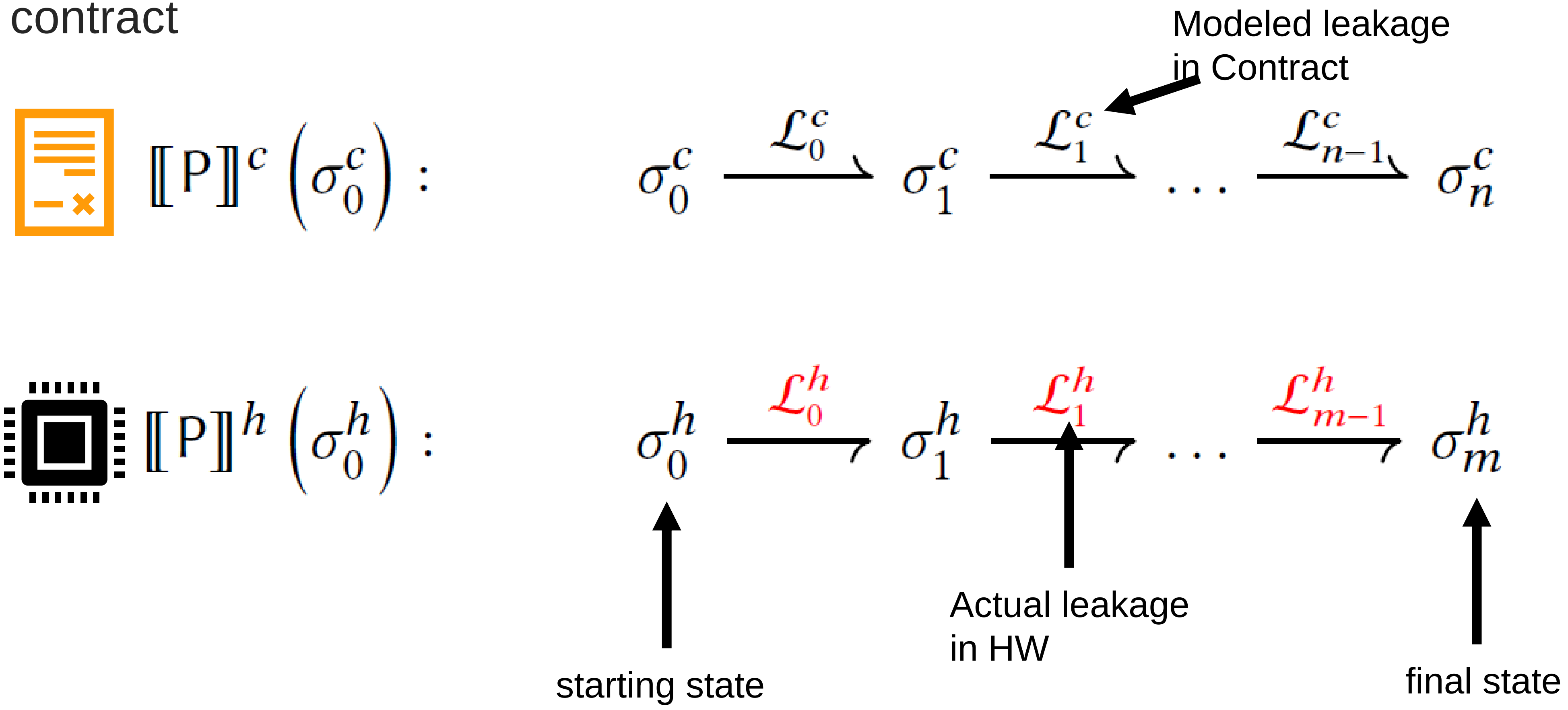
Actual leakage in HW

final state

DEFINITION 6 (COMPLIANCE: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract



Modeled leakage in Contract

$$[\![\mathrm{P}]\!]^c\left(\sigma_0^c\right): \qquad \sigma_0^c \xrightharpoonup{\mathcal{L}_0^c} \sigma_1^c \xrightharpoonup{\mathcal{L}_1^c} \ldots \xrightharpoonup{\mathcal{L}_{n-1}^c} \sigma_n^c$$

$$[\![\mathrm{P}]\!]^h\left(\sigma_0^h\right): \qquad \sigma_0^h \xrightharpoonup{\mathcal{L}_0^h} \sigma_1^h \xrightharpoonup{\mathcal{L}_1^h} \ldots \xrightharpoonup{\mathcal{L}_{m-1}^h} \sigma_m^h$$

starting state

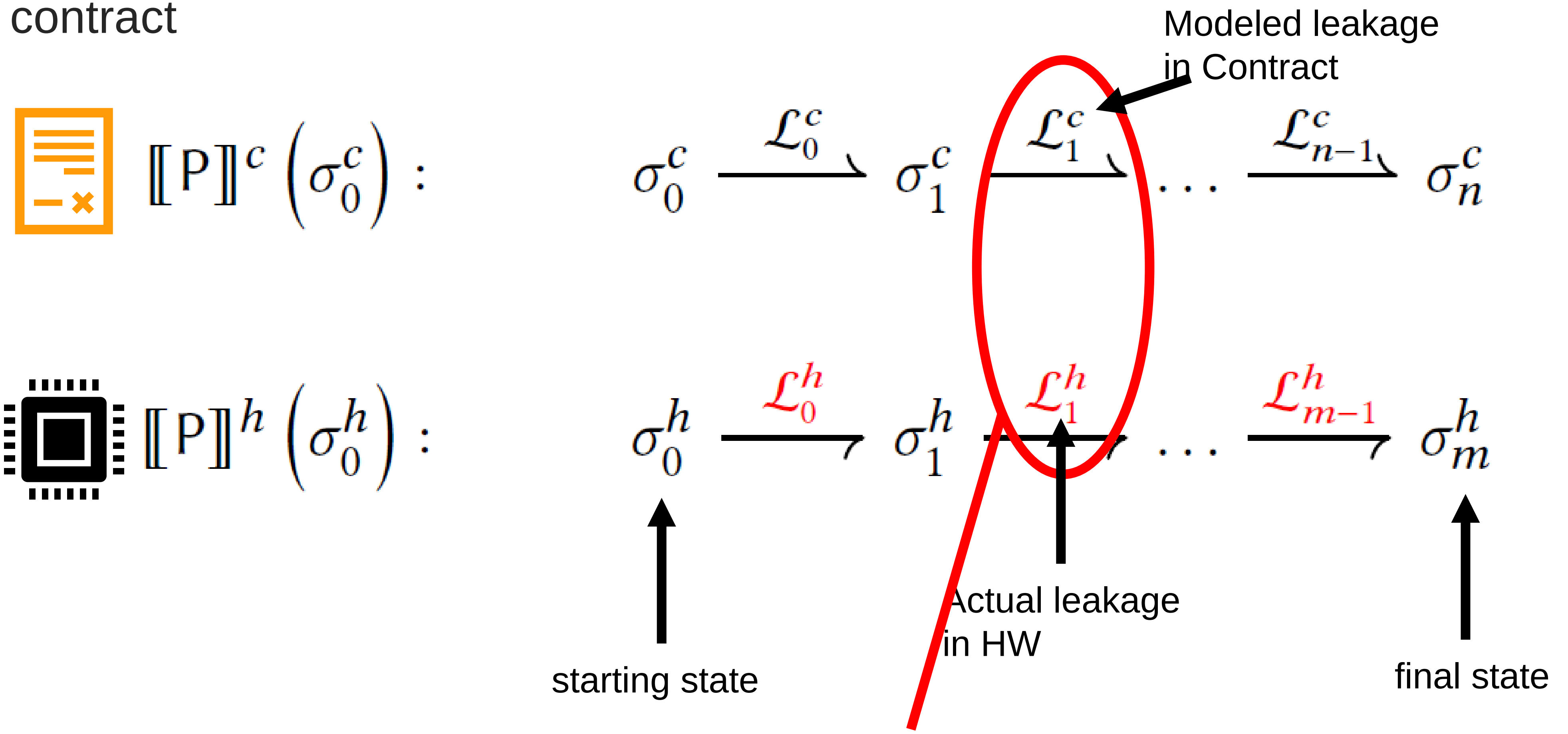
Actual leakage in HW

final state

DEFINITION 6 (COMPLIANCE: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract



$$[\![P]\!]^c\left(\sigma_0^c\right): \quad \sigma_0^c \xrightarrow{\mathcal{L}_0^c} \sigma_1^c \xrightarrow{\mathcal{L}_1^c} \dots \xrightarrow{\mathcal{L}_{n-1}^c} \sigma_n^c$$

$$[\![P]\!]^h\left(\sigma_0^h\right): \quad \sigma_0^h \xrightarrow{\mathcal{L}_0^h} \sigma_1^h \xrightarrow{\mathcal{L}_1^h} \dots \xrightarrow{\mathcal{L}_{m-1}^h} \sigma_m^h$$

Modeled leakage in Contract

starting state

Actual leakage in HW

final state

Prove that HW leakage can be modeled from some `leak` statement in contract

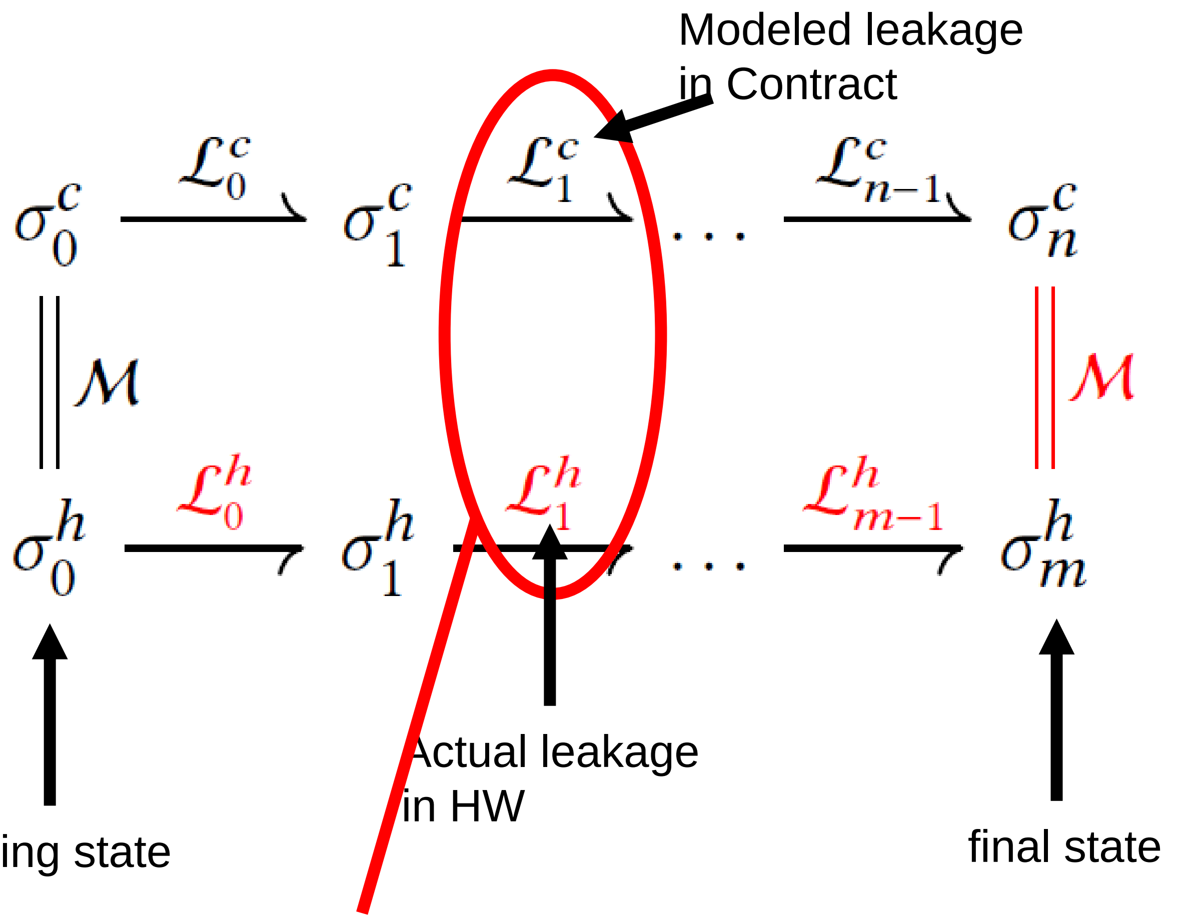
DEFINITION 6 (COMPLIANCE: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

VERIFYING COMPLETENESS IN A NUTSHELL
HW COMPLIANCE

- E2E security reduction based on ability to model any HW probe from modeled leakage in the contract

$[\![\mathrm{P}]\!]^c\left(\sigma_0^c\right):$

$[\![\mathrm{P}]\!]^h\left(\sigma_0^h\right):$



$$\begin{tikzcd}
\sigma_0^c \arrow[r, harpoon', "\mathcal{L}_0^c"] \arrow[d, equal, "\mathcal{M}"] & \sigma_1^c \arrow[r, harpoon', "\mathcal{L}_1^c"] & \cdots \arrow[r, harpoon', "\mathcal{L}_{n-1}^c"] & \sigma_n^c \arrow[d, equal, "\mathcal{M}"] \\
\sigma_0^h \arrow[r, harpoon, "\mathcal{L}_0^h"] & \sigma_1^h \arrow[r, harpoon, "\mathcal{L}_1^h"] & \cdots \arrow[r, harpoon, "\mathcal{L}_{m-1}^h"] & \sigma_m^h
\end{tikzcd}$$

Modeled leakage in Contract

Actual leakage in HW

starting state

final state

Prove that HW leakage can be modeled from some `leak` statement in contract

DEFINITION 6 (COMPLIANCE: $[\![\cdot]\!]^h \vdash_{\mathcal{M}} [\![\cdot]\!]^c$).

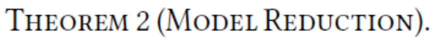
VERIFYING COMPLETENESS IN A NUTSHELL (1)
CHECKING THE ABILITY TO MODEL HW LEAKAGE FROM CONTRACT LEAKS

- Is there a function f(`e1,e2`) = y such that y = λ_g for all executions of a program?

$$\mathcal{L}_i^c \texttt{ := \{…, leak(e1, e2), …\}}$$

$$\mathcal{L}_j^h \texttt{ := \{…, } \lambda_g(\sigma_{j-1}^h, \sigma_j^h)\texttt{, …\}}$$

- Rationale:
  - If I know `e1,e2` which are exposed in the contract, then
  - I can simulate the observation of leakage λ_g of gate g in HW which an adversary could make



THEOREM 2 (MODEL REDUCTION).

VERIFYING COMPLETENESS IN A NUTSHELL (1)
CHECKING THE ABILITY TO MODEL HW LEAKAGE FROM CONTRACT LEAKS

- Is there a function f(`e1,e2`) = y such that y = λ_g for all executions of a program?

$\mathcal{L}_i^c$ `:= {…, leak( e1, e2 ), …}`

$\mathcal{L}_j^h$ `:= {…,` $\lambda_g(\sigma_{j-1}^h, \sigma_j^h)$`, …}`

∃ f(`e1,e2`) = λ_g ?

- Rationale:
  - If I know `e1,e2` which are exposed in the contract, then
  - I can simulate the observation of leakage λ_g of gate g in HW which an adversary could make

THEOREM 2 (MODEL REDUCTION).

VERIFYING COMPLETENESS IN A NUTSHELL (3)
CHECKING THE ABILITY TO MODEL HW LEAKAGE FROM CONTRACT LEAKS

? ∃ f(`e1,e2`) = λ_g

THEOREM 5 (EXISTENCE OF MODELING FUNCTION).

- check on **two pairs of executions**
  - starting in σ_0^c, σ_0^h respectively, $\sigma_0^{c\prime}$, $\sigma_0^{h\prime}$
  - Is there an execution leading to `e1 = e1'` and `e2 = e2'` but $\lambda_g \neq \lambda_g'$?
  - Then there is no single f since it would need to output different values for the same inputs
  - Encode as SMT for automated check

VERIFYING COMPLETENESS IN A NUTSHELL (4)
VERIFICATION OVERVIEW

- Any single instruction
- Deal with pipelines

```
…
xor rD, rN, rM
…
```

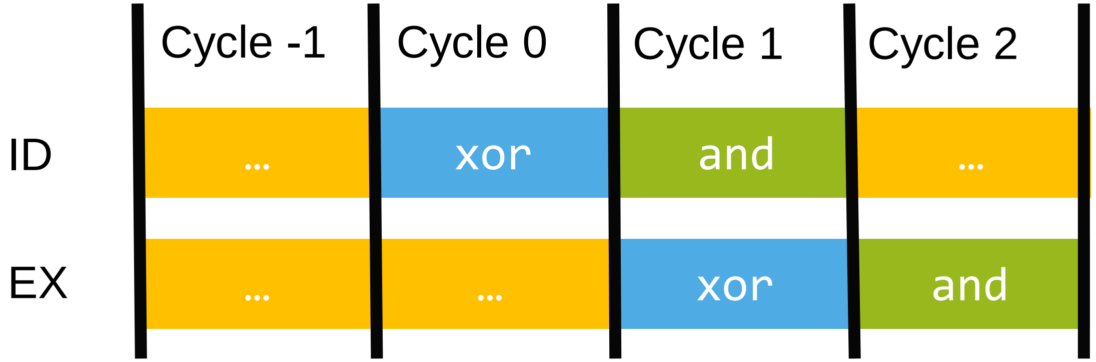
$$\begin{tikzcd}
& \sigma_i^c \arrow[rr, "\mathcal{L}_i^c"] & & \sigma_{i+1}^c \arrow[dr, equal, "\mathcal{M}"] & \\
\sigma_j^h \arrow[ur, equal, "\mathcal{M}"] \arrow[rr, "\mathcal{L}_j^h"] & & \cdots \arrow[rr, "\mathcal{L}_{j+k-1}^h"] & & \sigma_{j+k}^h
\end{tikzcd}$$

- Constraints
  - New instruction issued in cycle j
  - Instruction successfully retires in cycle $j + k$
  - Assert no exceptions, reset, debug or interrupt
  - Memory access immediately granted, no errors
  - …
- Verify for instructions lengths k = 1 to max. length

VERIFYING COMPLETENESS IN A NUTSHELL (5)
CONFIGURATION

- Dealing with pipelined execution

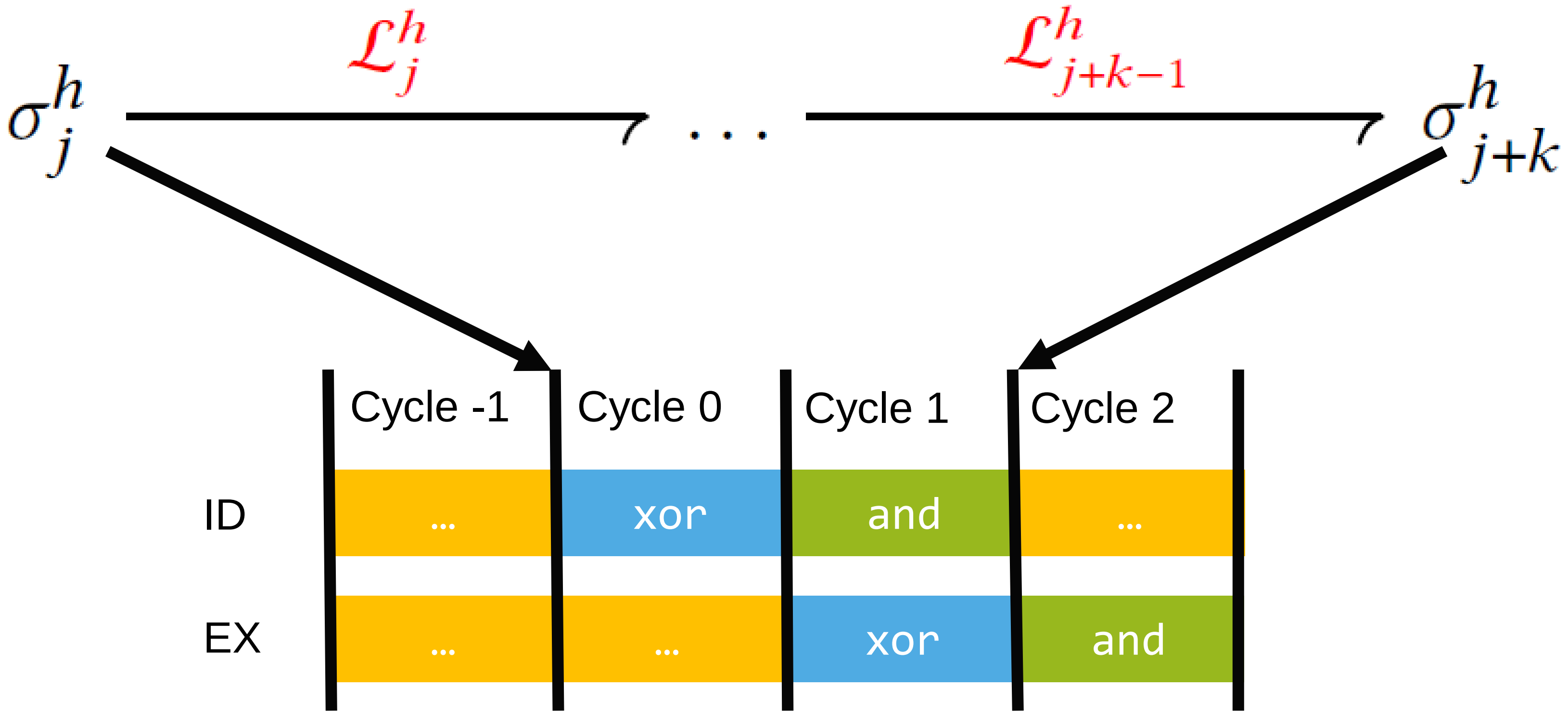
$$\sigma_j^h \xrightarrow{\mathcal{L}_j^h} \cdots \xrightarrow{\mathcal{L}_{j+k-1}^h} \sigma_{j+k}^h$$



	Cycle -1	Cycle 0	Cycle 1	Cycle 2
ID	…	xor	and	…
EX	…	…	xor	and

VERIFYING COMPLETENESS IN A NUTSHELL (5)
CONFIGURATION

- Dealing with pipelined execution

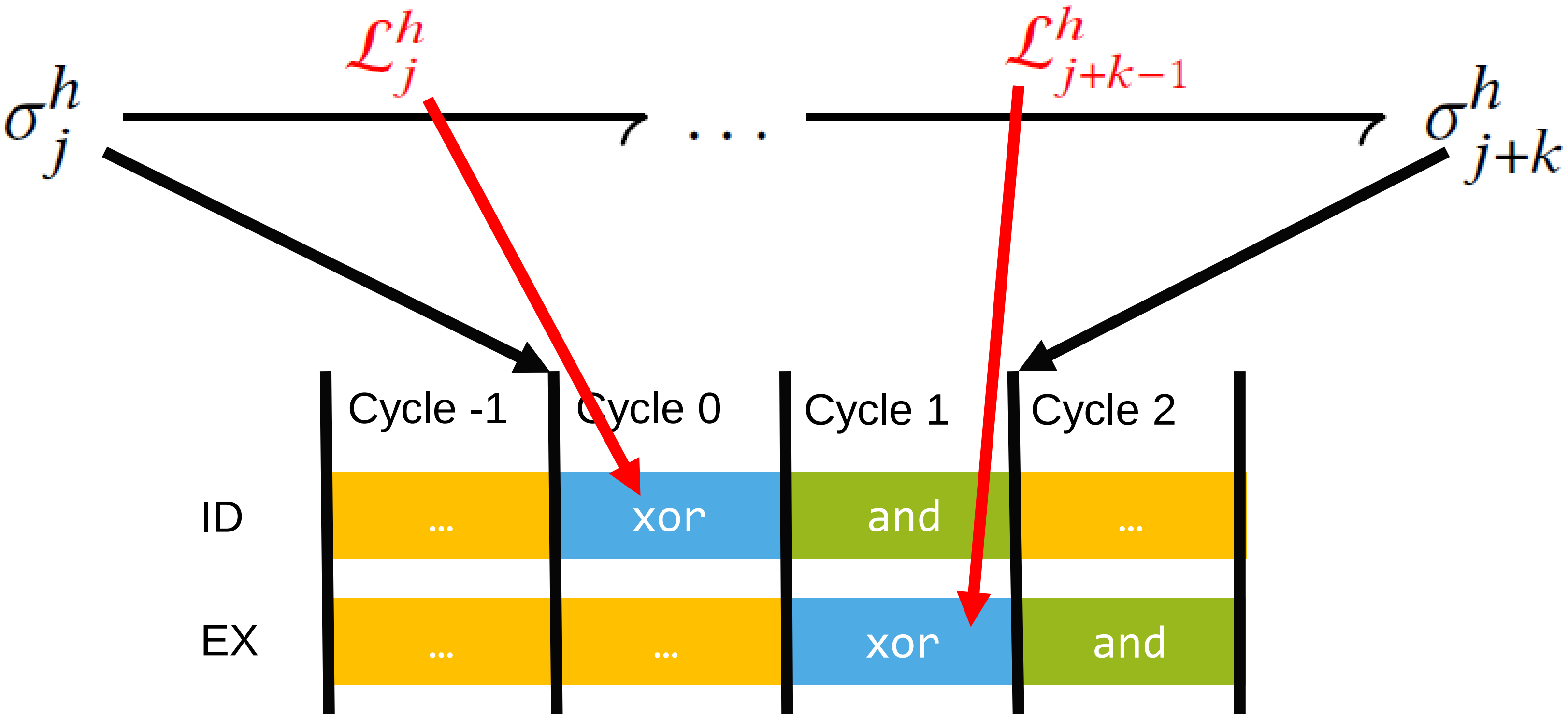


$$\sigma_j^h \xrightarrow{\mathcal{L}_j^h} \cdots \xrightarrow{\mathcal{L}_{j+k-1}^h} \sigma_{j+k}^h$$

	Cycle -1	Cycle 0	Cycle 1	Cycle 2
ID	…	xor	and	…
EX	…	…	xor	and

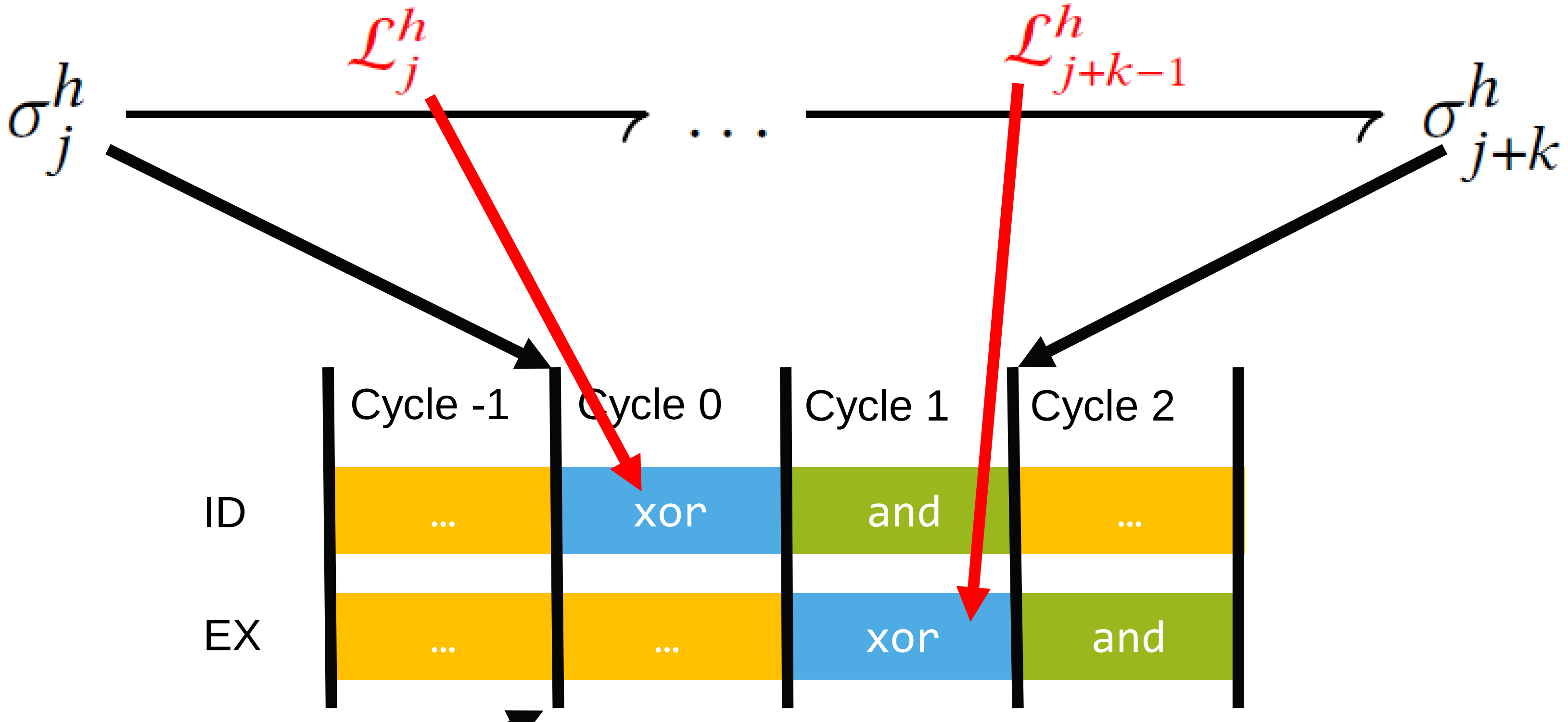
VERIFYING COMPLETENESS IN A NUTSHELL (5)
CONFIGURATION

- Dealing with pipelined execution



VERIFYING COMPLETENESS IN A NUTSHELL (5) CONFIGURATION

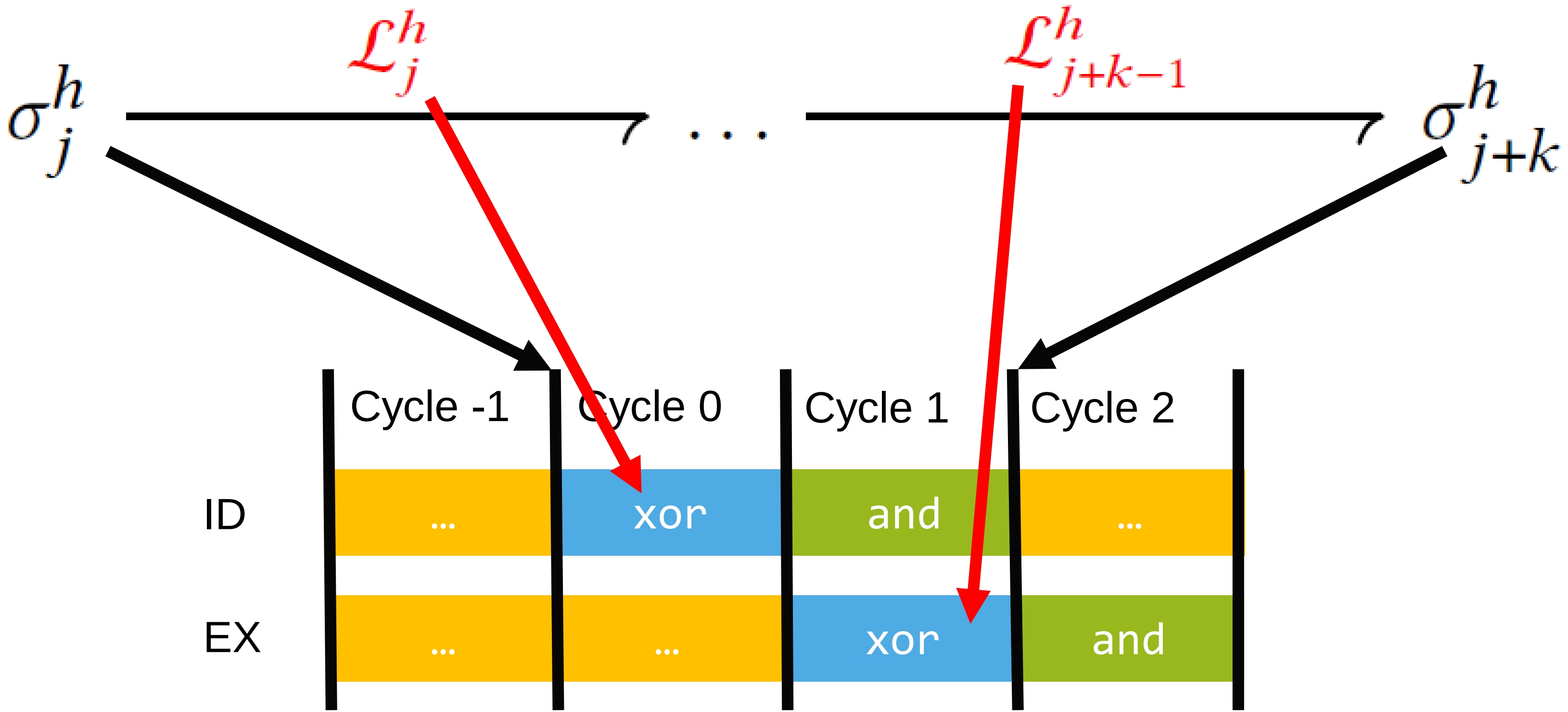
- Dealing with pipelined execution



New instruction issued in cycle *j*

VERIFYING COMPLETENESS IN A NUTSHELL (5) CONFIGURATION

- Dealing with pipelined execution

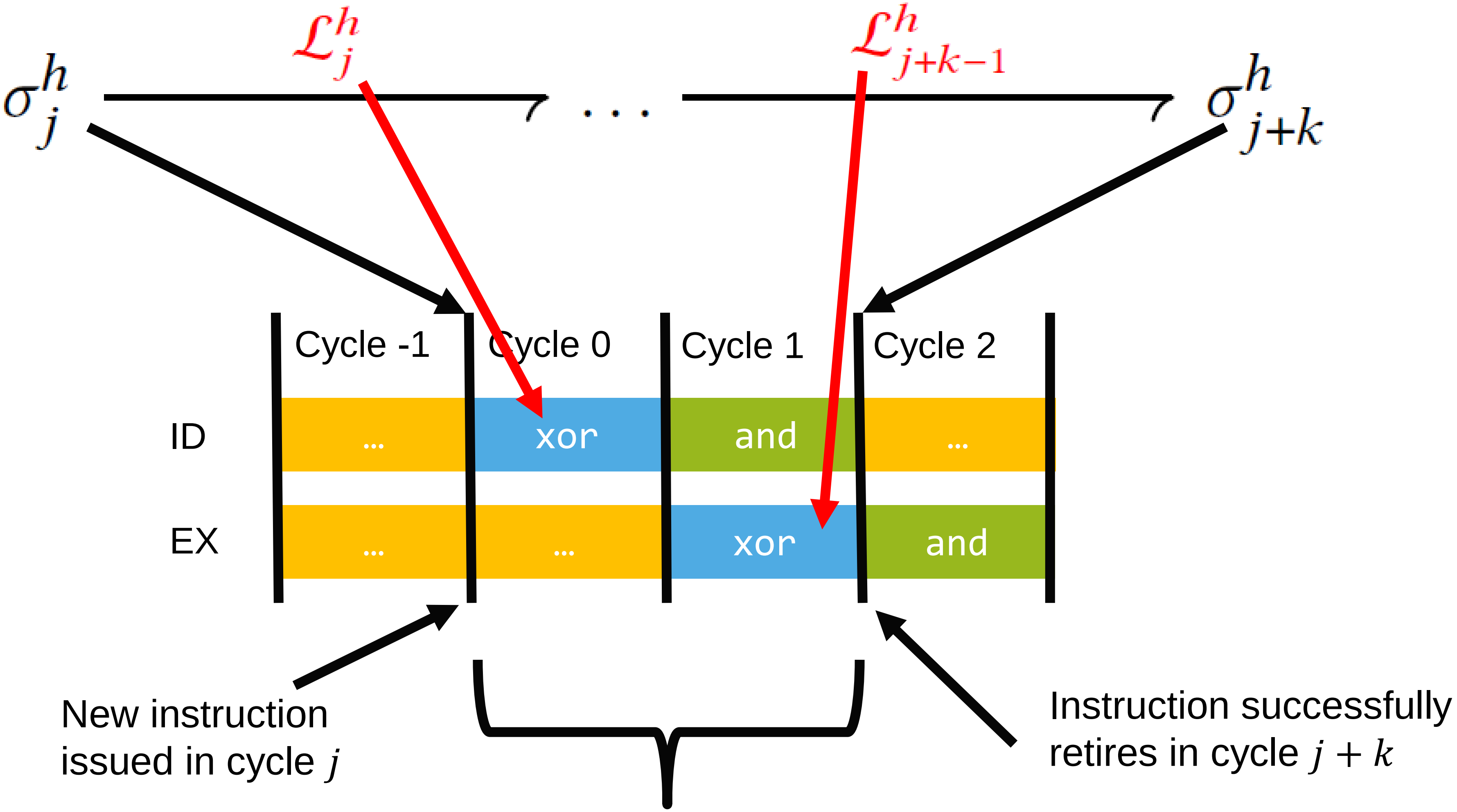


New instruction issued in cycle j

Instruction successfully retires in cycle $j + k$

VERIFYING COMPLETENESS IN A NUTSHELL (5) CONFIGURATION

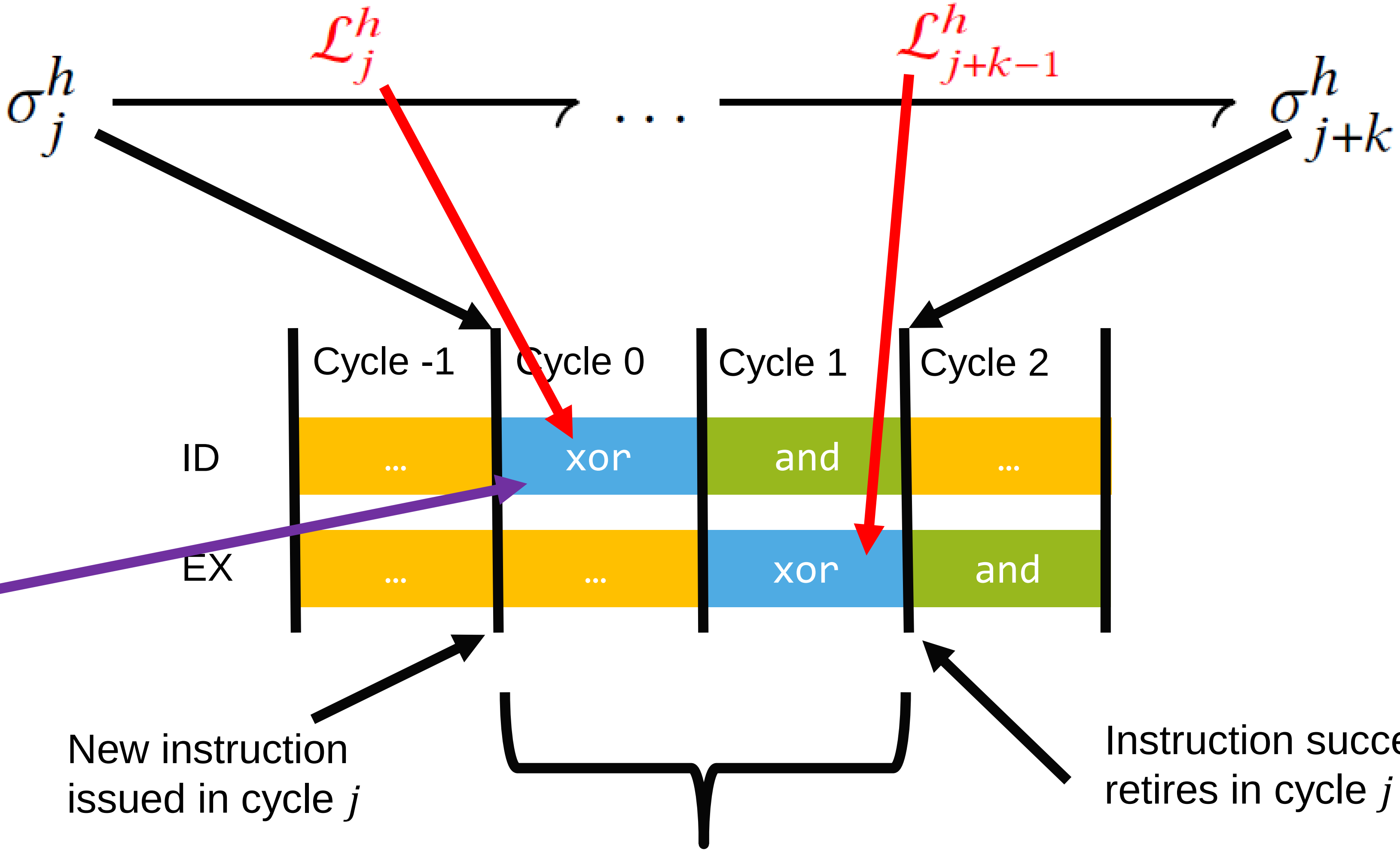
- Dealing with pipelined execution



no exceptions, reset, debug or interrupt

VERIFYING COMPLETENESS IN A NUTSHELL (5) CONFIGURATION

- Dealing with pipelined execution



Configuration for any supported instruction

New instruction issued in cycle j

Instruction successfully retires in cycle $j + k$

no exceptions, reset, debug or interrupt

SUMMARY OF CONTRIBUTIONS

- Contracts express precise leakage behavior
- Method & tool to
  - check functional correctness of contract
  - check completeness of leakage specification → provably complete leakage models
- Proven end-to-end resilience
  - Proofs of security based on the contract also hold for adversaries attacking compliant hardware
  - “the execution of **any** program on **any** compliant HW is secure if security against the contract has been shown”
- Applied to IBEX processor
  - Open-source contract
- Contract can be compiled to fast emulator
  - E.g., for power trace simulator or statistical security assessment

LIMITATIONS

- HW leakage model
  - Tool does not (yet) support glitches / couplings / tech-mapped netlist
  - Methodology extends seamlessly

- Random probing security, Noisy leakage model
  - Contract does not (yet) carry information on leakage rates
  - Existing approaches to security reductions [1], [2]
  - Is it possible to augment contracts with leakage/noise rates and to verify these bounds against netlists?

[1]: **Unifying Leakage Models: From Probing Attacks to Noisy Leakage.** Alexandre Duc, Stefan Dziembowski, Sebastian Faust. J. of Cryptologoy 2019.

[2]: **The Mother of All Leakages: How to Simulate Noisy Leakages via Bounded Leakage (Almost) for Free.** Gianluca Brian, Antonio Faonio, Maciej Obremski, João Ribeiro, Mark Simkin, Maciej Skórski, and Daniele Venturi. EUROCRYPT 2021.

LISTING L
LICENSE OF SHOWN CODE-SNIPPETS

RISCV Sail Model

This Sail RISC-V architecture model, comprising all files and directories except for the snapshots of the Lem and Sail libraries in the prover_snapshots directory (which include copies of their licences), is subject to the BSD two-clause licence below.

Copyright (c) 2017-2021 Prashanth Mundkur, Rishiyur S. Nikhil and Bluespec Inc., Jon French, Brian Campbell, Robert Norton-Wright, Alasdair Armstrong, Thomas Bauereiss, Shaked Flur, Christopher Pulte, Peter Sewell, Alexander Richardson, Hesham Almatary, Jessica Clarke, Microsoft, for contributions by Robert Norton-Wright and Nathaniel Wesley Filardo, Peter Rugg and Aril Computer Corp., for contributions by Scott Johnson.

Copyright 2020-2022 - TUHH, TU Graz

All rights reserved.

This software was developed by the above within the Rigorous Engineering of Mainstream Systems (REMS) project, partly funded by EPSRC grant EP/K008528/1, at the Universities of Cambridge and Edinburgh.

This software was developed by SRI International and the University of Cambridge Computer Laboratory (Department of Computer Science and Technology) under DARPA/AFRL contract FA8650-18-C-7809 ("CIFV"), and under DARPA contract HR0011-18-C-0016 ("ECATS") as part of the DARPA SSITH research programme.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 789108, ELVER).

This software has received funding from the Federal Ministry of Education and Research (BMBF) as part of the VE-Jupiter project grant 16ME0231K.

This work was supported by the Austrian Research Promotion Agency (FFG) through the FERMION project (grant number 867542).

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SECURE CONNECTIONS
FOR A SMARTER WORLD