Axiomatic Hardware-Software Security Contracts

Hamed Nemati, KTH

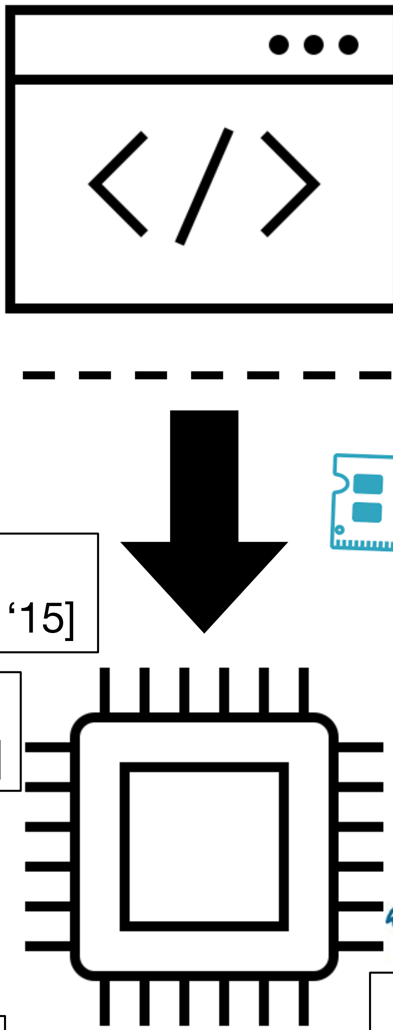
Joint work with Nicholas Mosier, Hanna Lachnitt, Caroline Trippel

CIDS Spring Conference 2024

KTH Royal Institute of Technology

Slides courtesy of Nicholas Mosier

Hardware Underpins Software Security



If one considers the union of all optimizations on this slide, **no instruction operand/result or data at rest is safe [Vicarte+, ISCA'21].**

Subnormal floating point
[Andrysco+, S&P '15]

DRAM
[Google Project Zero '15]

Caches
[Osvik+, CT-RSA '06]
[Yarom+, USENIX '14]

Coherence
[Guanciale+, Oakland '16]

Division early exit
[Coppens, S&P '09]

Digit-serial multiplication
[Großschäd+, ISISC '09]

OoO Execution
[Lipp+, USENIX '18]

Indirect memory prefetchers
[Vicarte+, ISCA '21]

Register-file compression
[Vicarte+, ISCA '21]

Compressed Caches
[Tsai+, ISCA '20]

Silent stores
[Vicarte+, ISCA '21]

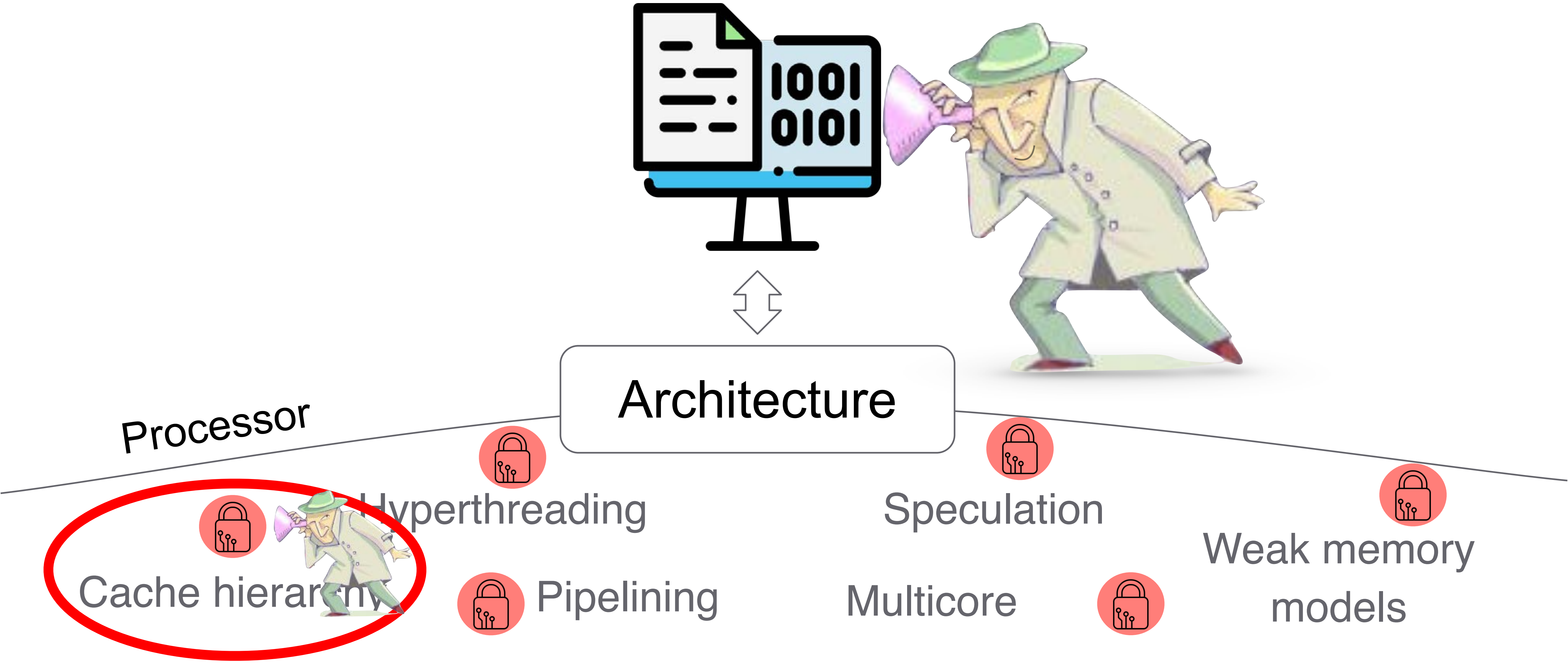
Value prediction
[Vicarte+, ISCA '21]

Speculation
[Kocher+, S&P '19]

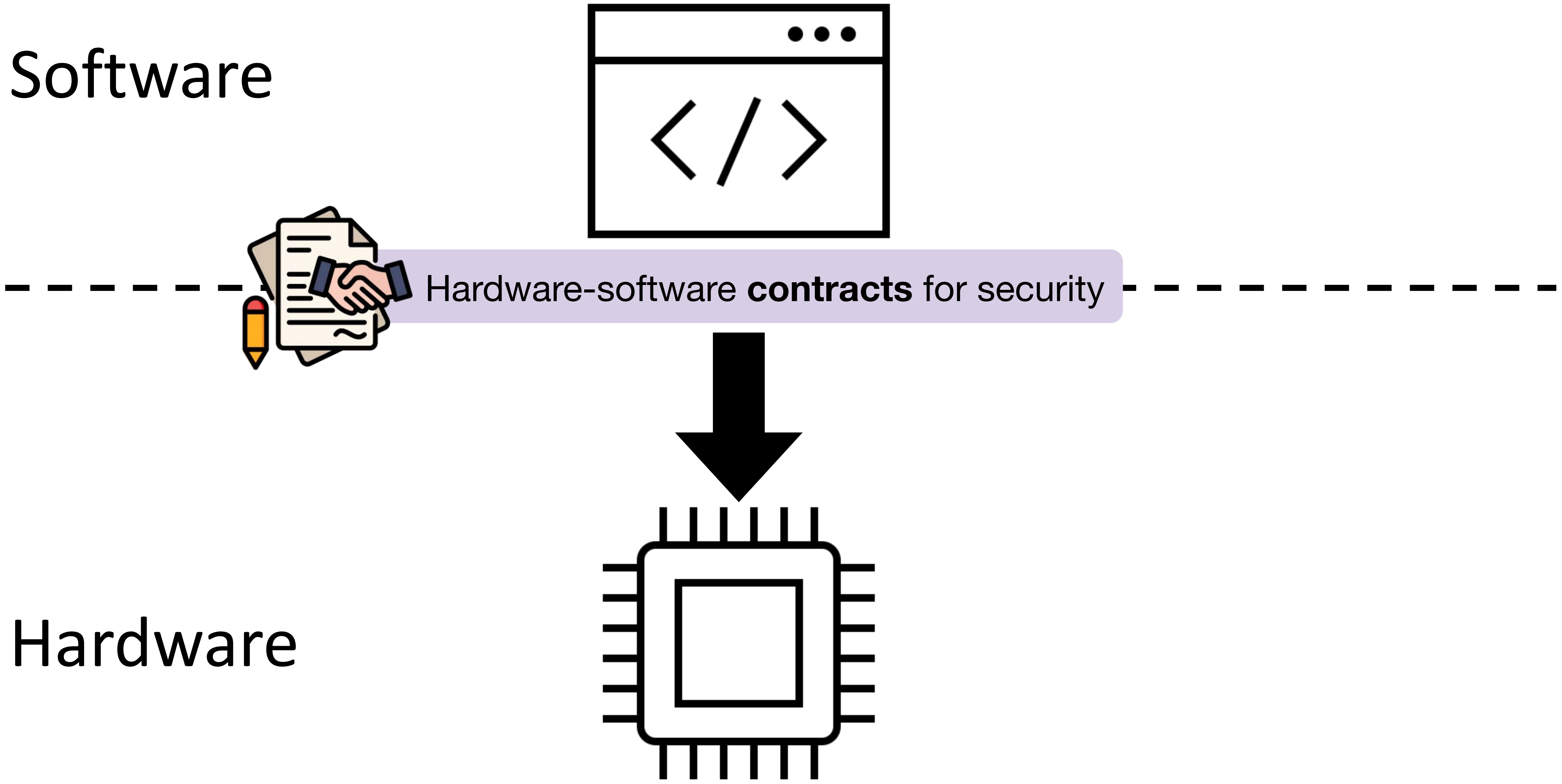
Computation reuse
[Vicarte+, ISCA '21]

And many more …

Side-Channel Attacks



Hardware-Software Contracts



Lesson Learned from the PL Community

Peter Sewell

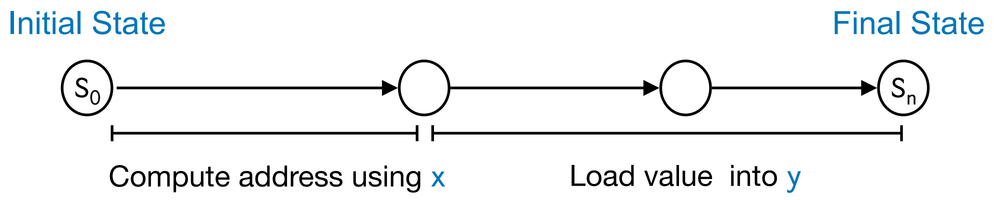
Jade Alglave

1990s
Weak consistency
(Operational)

2010s
Weak consistency
(Axiomatic)

- Operational : Step-by-step state evolution

Example of **O**perational **S**pecifications: Read(x) $\mapsto$ y

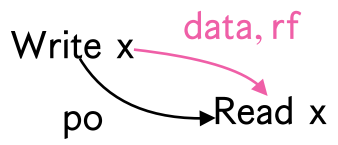


- Axiomatic: take arbitrary behavior, filter those not accepted by the semantics

Example of **C**andidate **E**xecution:

Instructions:	Events:
Write(α) $\mapsto$ x	Write x
Read(x) $\mapsto$ y	Read x

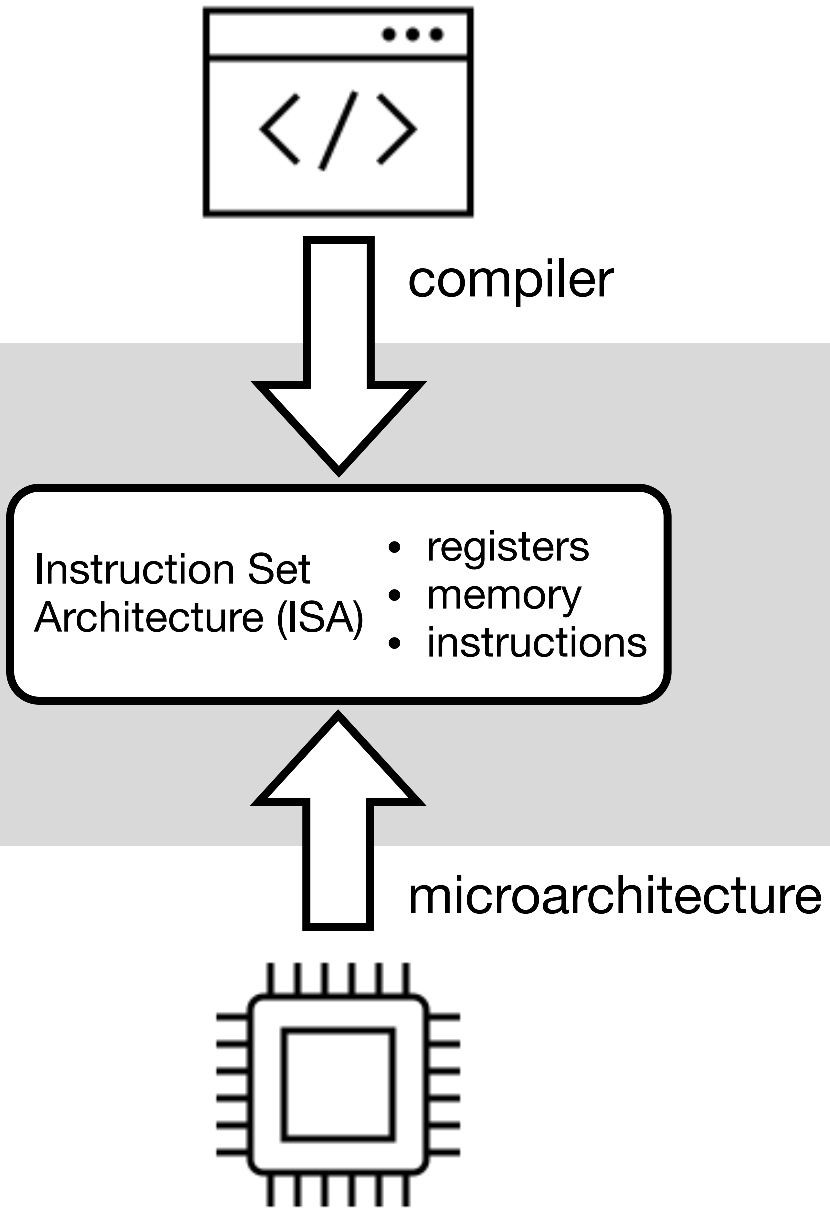
Event Graph:



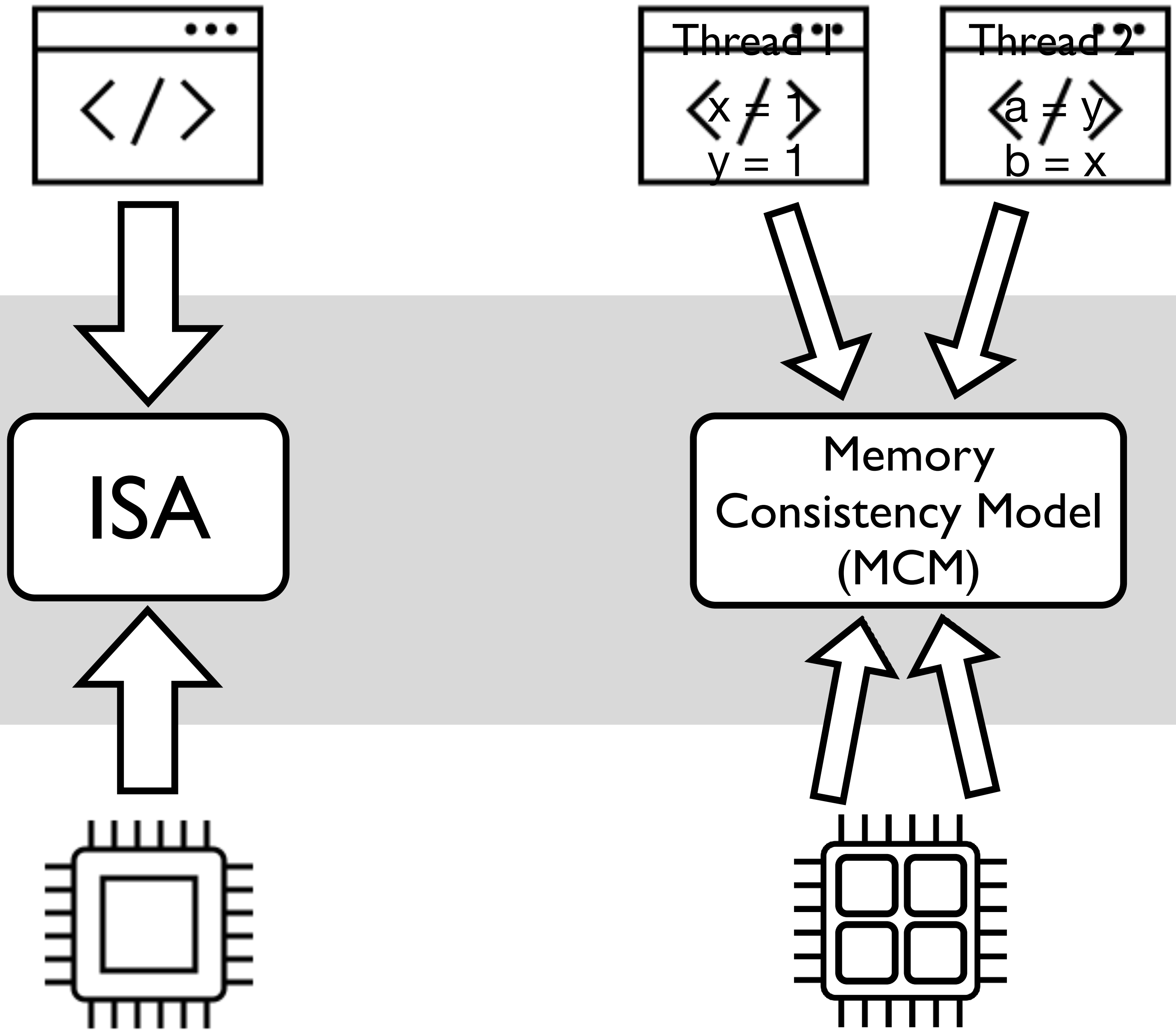
Slide courtesy of Hernán Ponce de León

Roadmap

- **Background:** Hardware-Software Contracts & Memory Consistency Models (MCMs)
- Building Blocks of **Microarchitectural Leakage**
- **Leakage Containment Models:** Modeling Microarchitectural Leakage
- **Clou:** Detecting and Mitigating Microarchitectural Leakage in Programs


compiler
Instruction Set Architecture (ISA)
• registers
• memory
• instructions
microarchitecture

Can a = 1, b = 0? ⟺ Can we observe a re-ordering of Thread 1's stores or Thread 2's loads?

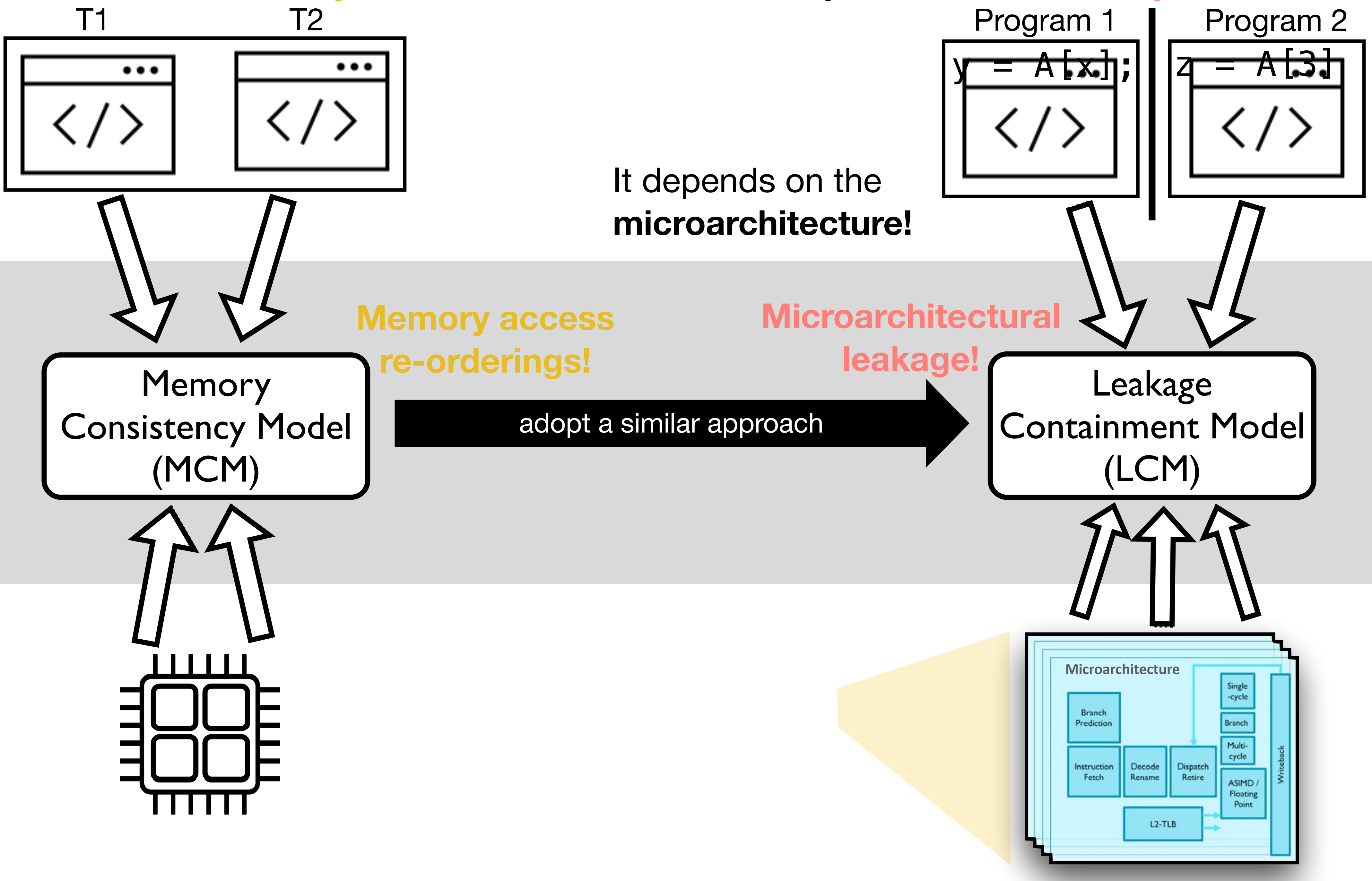


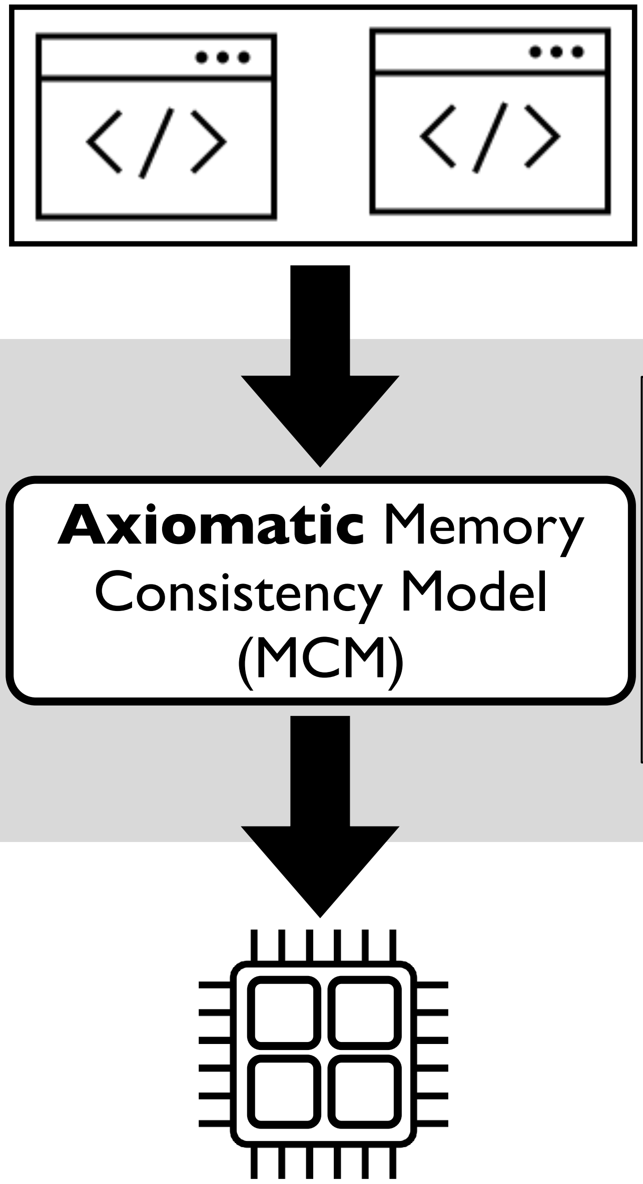
It depends on the **architecture!**

- ✗ intel
- ✓ arm
- ✓ IBM
- ✓ RISC-V

Can T2 observe a re-ordering of T1's stores?

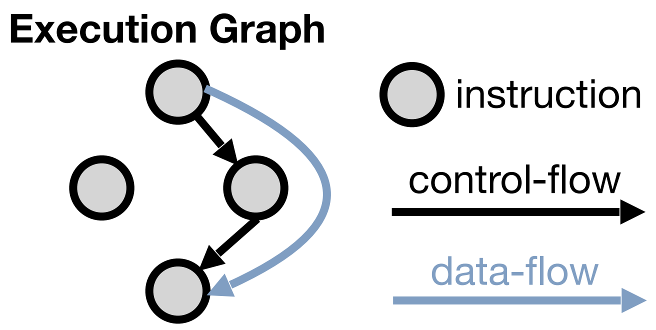
Can Program 2 observe leakage of x from Program 1?

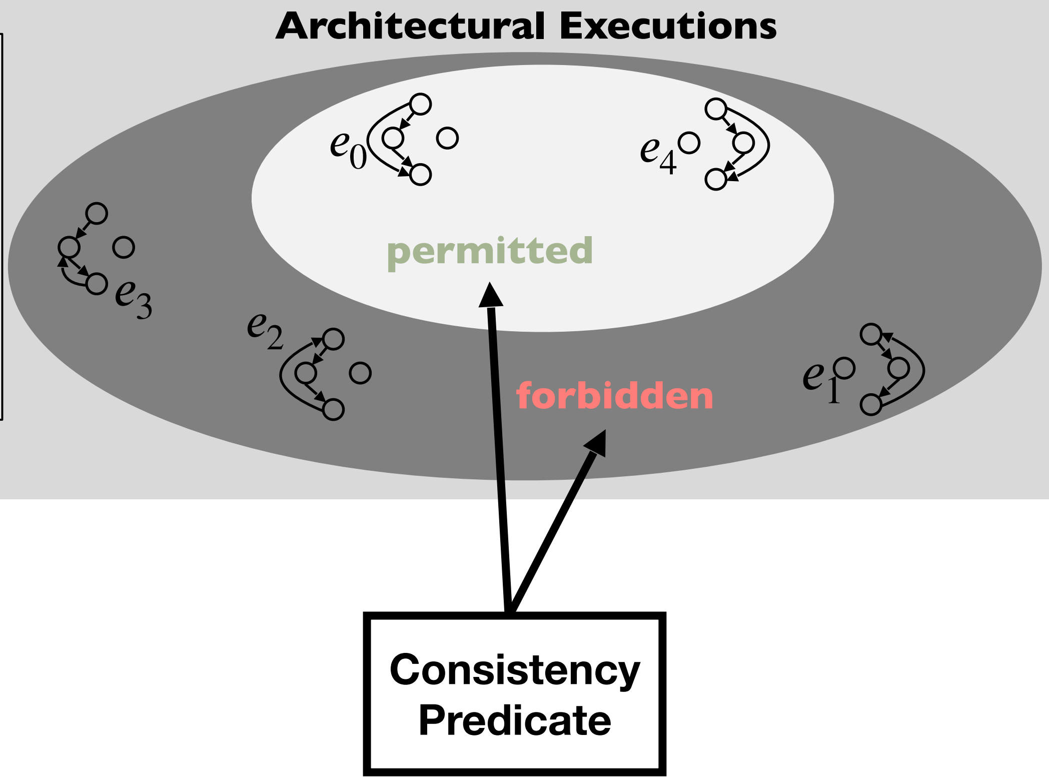




MCMs:
- Define the legal ordering + visibility of shared memory accesses

Axiomatic MCMs:
- Model architectural executions of a program as directed graphs
  - *Nodes:* instructions
  - *Directed edges:* happens-before relations
- *Consistency predicate* defines legal executions





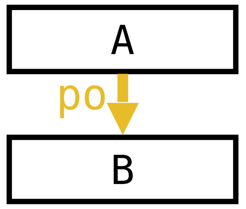
Modeling Program Executions Axiomatically With Happens-Before Relations

control-flow

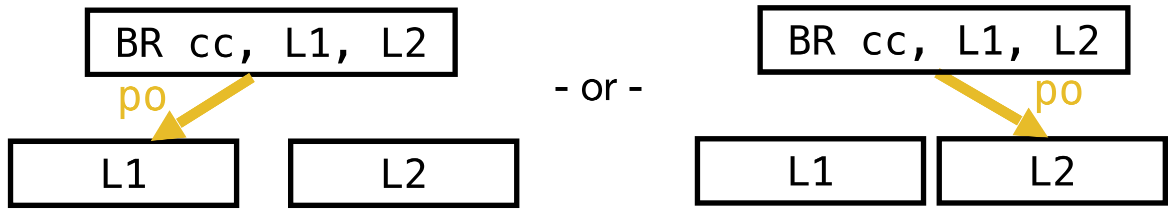
po

Encodes branch outcomes.

```
A; B;
```



```
if (cc) L1 else L2
```

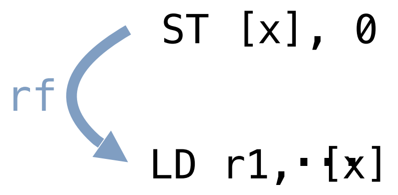


data-flow

rf, co, fr

Encodes dynamic data-flow through memory.

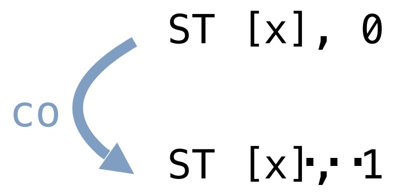
```
x = 0;
... = x;
```



```
ST [x], 0
LD r1, [x]
```

reads-from (rf)
relates store→load if load reads from store

```
x = 0;
x = 1;
```



```
ST [x], 0
ST [x], 1
```

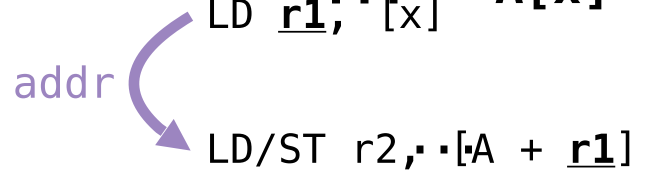
coherence order (co)
constructs a total order on same-address stores

dependencies

addr, data, ctrl

Encodes syntactic data-flow through registers.

```
r1 = A[x]
```



```
LD r1, [x]
LD/ST r2, [A + r1]
```

address dependency (addr):
relates load→access where accesses uses load in address computation

Roadmap

- **Background:** Hardware-Software Contracts & Memory Consistency Models
- Building Blocks of **Microarchitectural Leakage**
- **Leakage Containment Models:** Modeling Microarchitectural Leakage
- **Clou:** Detecting and Mitigating Microarchitectural Leakage in Programs

Microarchitectural Data-flow Enables Leakage

Program 1	Program 2
`y = A[x];`	`z = A[3];`

```
y = A[3]


z = A[3]
```

Cache

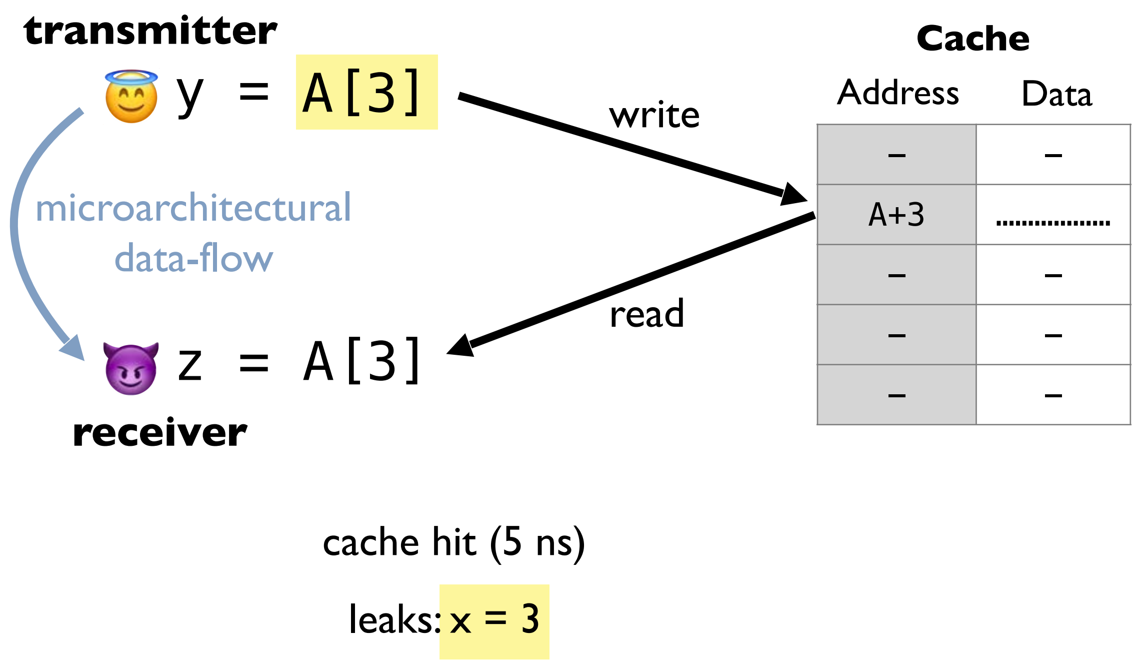
Address	Data
–	–
–	–
–	–
–	–
–	–

Ingredients for modeling **microarchitectural leakage:**

1. Instructions exhibit **>1** **different executions.**
2. Which execution is realized **depends on hardware information flows.**

Microarchitectural Data-flow Enables Leakage

Program 1	Program 2
`y = A[x];`	`z = A[3];`

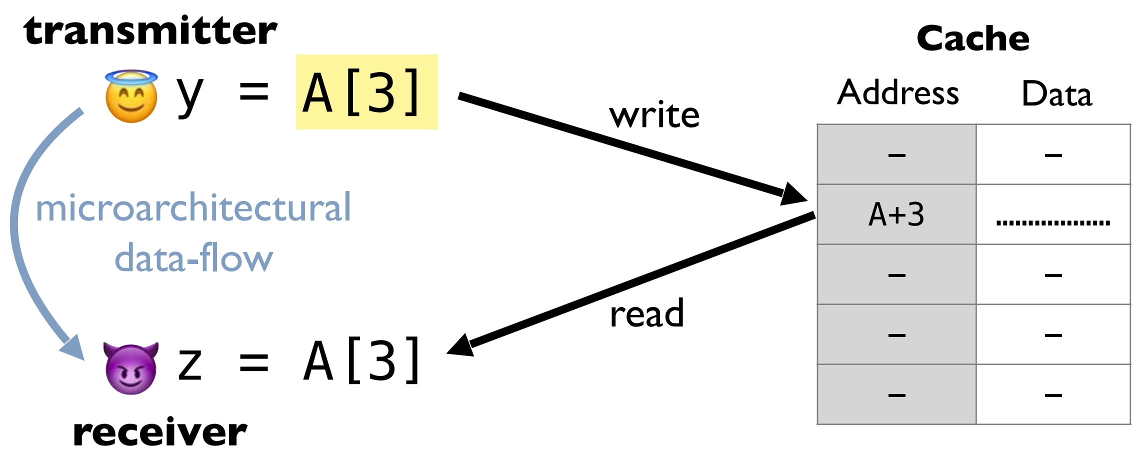


Ingredients for modeling **microarchitectural leakage:**

1. Instructions exhibit **>1 different executions.**
2. Which execution is realized **depends on hardware information flows.**

Microarchitectural Data-flow Enables Leakage

Program 1	Program 2
`y = A[x];`	`z = A[3];`



Cache

Address	Data
–	–
A+3	....
–	–
–	–
–	–

```
y = A[5]

z = A[3]
```

Cache

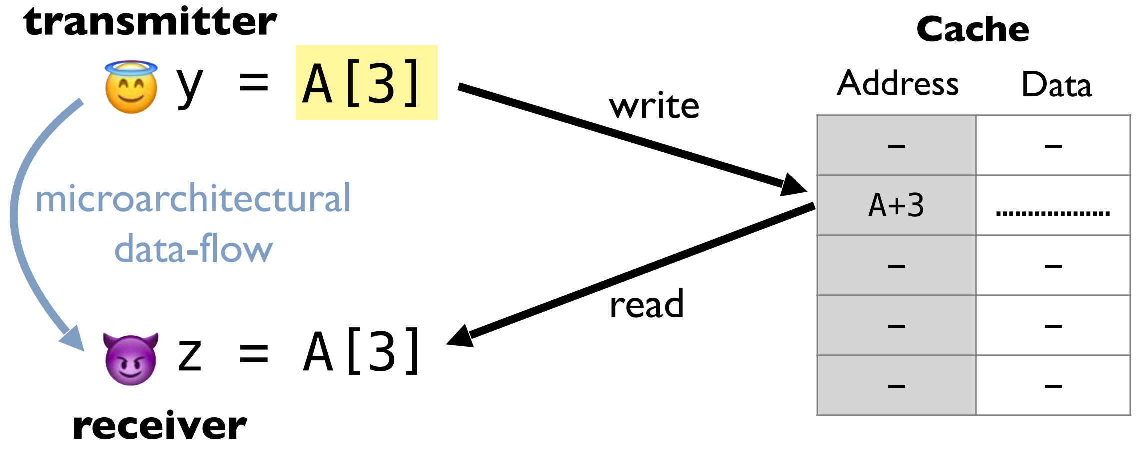
Address	Data
–	–
–	–
–	–
–	–
–	–

cache hit (5 ns)

leaks: x = 3

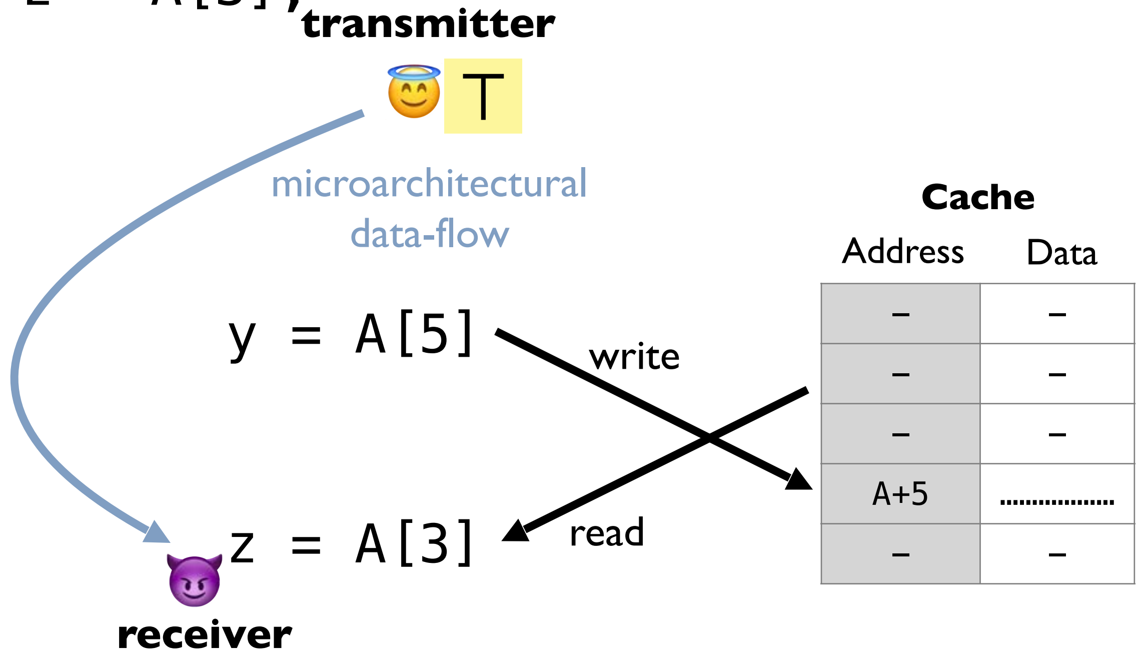
Microarchitectural Data-flow Enables Leakage

Program 1	Program 2
`y = A[x];`	`z = A[3];`



cache hit (5 ns)

leaks: x = 3



cache miss (50 ns)

leaks: $x \neq 3$

Microarchitectural Control Flow Increases Leakage Scope

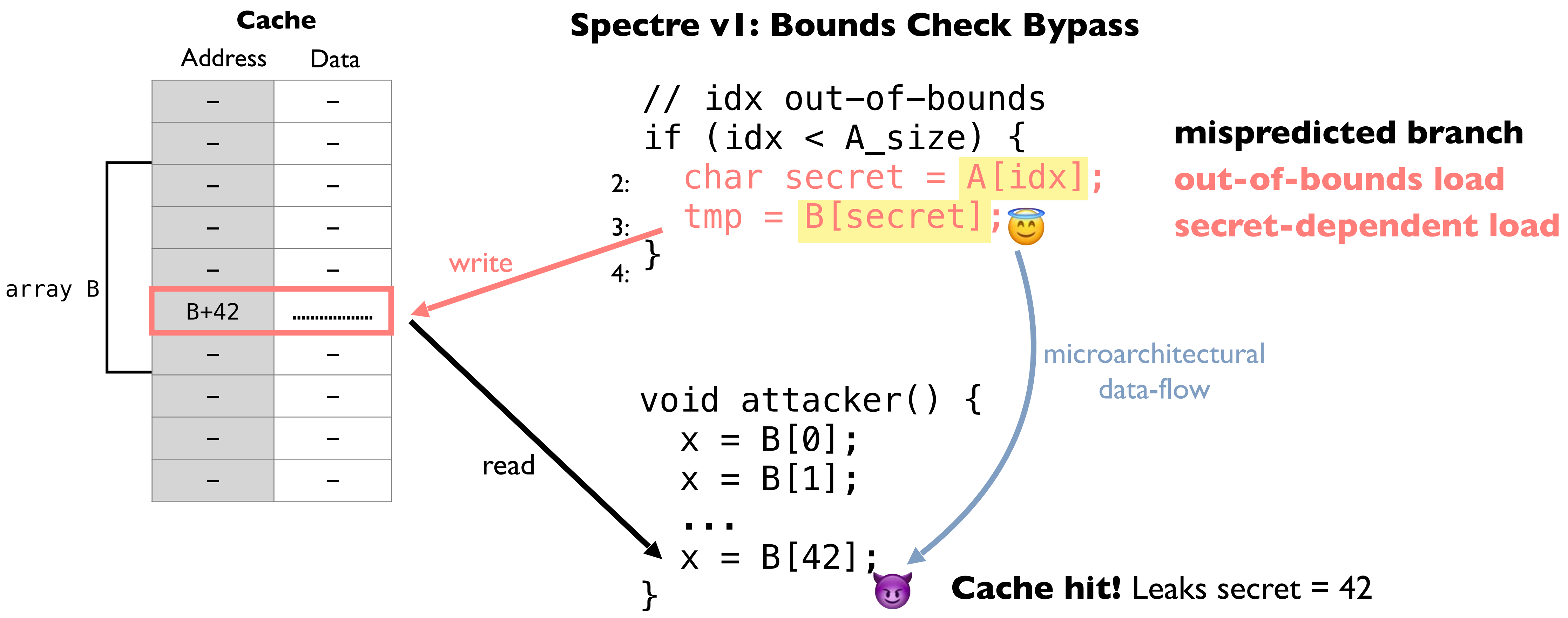
Spectre v1: Bounds Check Bypass

```
// idx out-of-bounds
if (idx < A_size) {
  char secret = A[idx];
  tmp = B[secret];
}
```

mispredicted branch

Modern hardware predicts branch outcomes and **speculatively executes** instructions along predicted paths.

Microarchitectural Control Flow Increases Leakage Scope



Spectre v1: Bounds Check Bypass

```
// idx out-of-bounds
if (idx < A_size) {
  char secret = A[idx];
  tmp = B[secret];
}
```

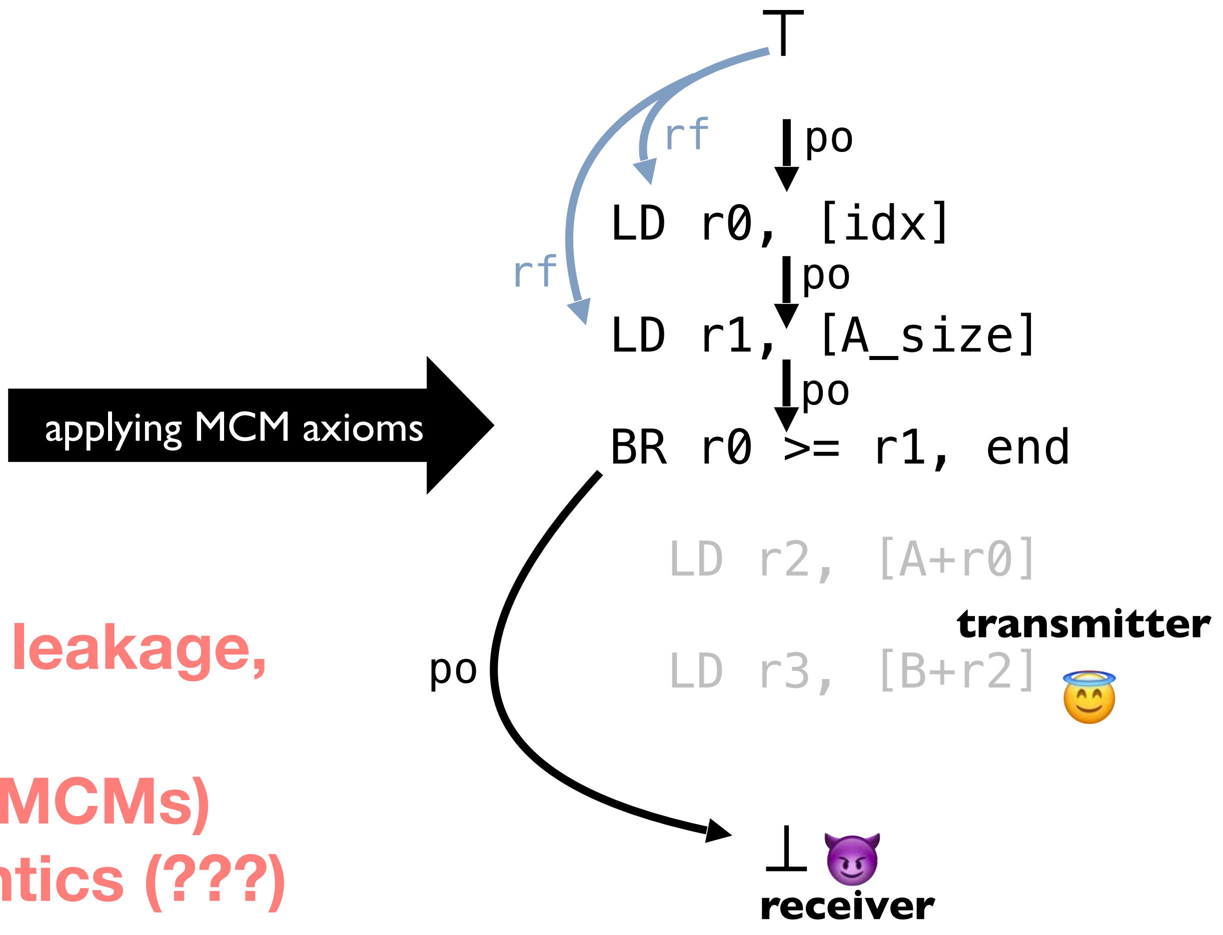
```
void attacker() {
  x = B[0];
  x = B[1];
  ...
  x = B[42];
}
```

Cache hit! Leaks secret = 42

Modern hardware predicts branch outcomes and **speculatively executes** instructions along predicted paths.

MCMs Lay the Foundation for LCMs But Fall Short for Modeling Microarchitectural Leakage

```
if (idx < A_size) {
  char secret = A[idx];
  tmp = B[secret];
}
```



To model microarchitectural leakage, we need:

LCMs {
1. **Architectural semantics (MCMs)**
2. **Microarchitectural semantics (???)**

MCMs do not capture **microarchitectural control-flow** or **microarchitectural data-flow**
… but they tell us how to construct a model that does!

Roadmap

- **Background:** Hardware-Software Contracts & Memory Consistency Models
- Building Blocks of **Microarchitectural Leakage**
- **Leakage Containment Models:** Modeling Microarchitectural Leakage
- **Clou:** Detecting and Mitigating Microarchitectural Leakage in Programs

Deriving a Microarchitectural Semantics From Architectural MCMs

	MCMs / LCMs Arch. Semantics	LCMs Microarch. Semantics
abstraction level	architecture	microarchitecture
communication medium	memory location	xstate
control-flow	po	tfo
data-flow	rf, co	rfx, cox
legal executions	consistency predicate	confidentiality predicate

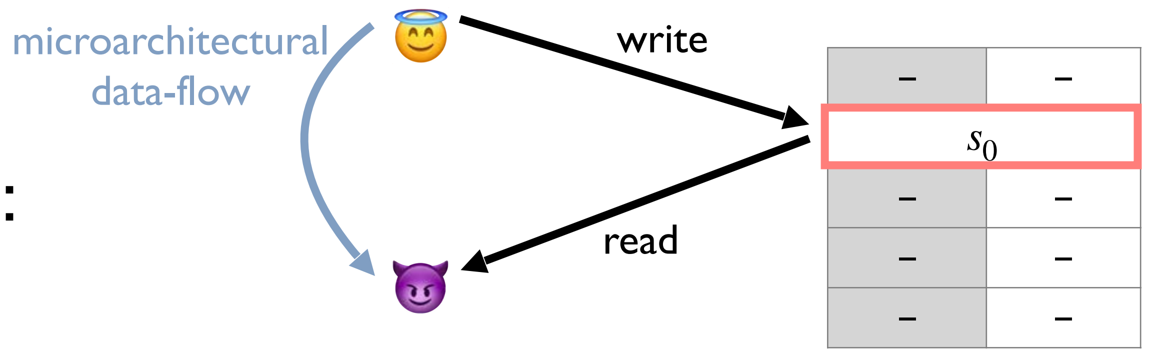
Encodes **SW-visible** execution

Encodes **HW-visible** execution

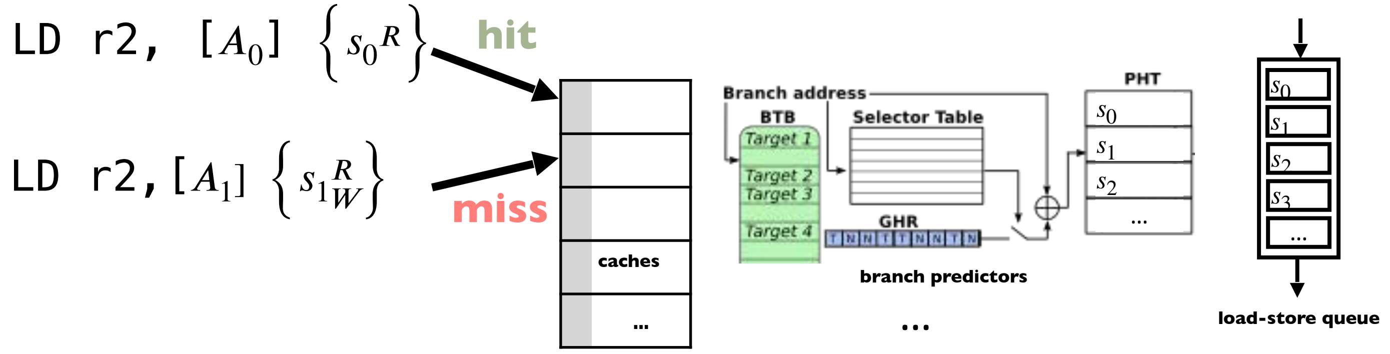
LCMs Model Microarchitectural Data-Flow Through xstate

- **xstate:** any non-architectural state in a microarchitecture
- **xstate variables** represent hardware state elements which:
    - facilitate **microarchitectural data-flow** between instructions
    - be **read from** *and* **written to** by instructions
- Instructions may **read and/or write** xstate variable(s)

xstate facilitates microarchitectural dataflow



xstate examples



Detecting Leakage in Programs with LCMs

High Level Idea: Architectural Noninterference $\longrightarrow$ Microarchitectural Noninterference **else, microarch. leakage**

Key Idea: apply the standard notion of conditional non-interference using rf and rfx to represent architectural and microarch. Observation, rspct.

- **Observation:** searching for instances of microarchitectural leakage in programs can be reduced to searching for violations of **three non-interference rules.**

Example rule: rfx **non-interference** (😇$\nrightarrow$😈) holds if for all writes $\boldsymbol{w}$ and all reads $\boldsymbol{r}$,

$$\boldsymbol{w} \xrightarrow{\text{rf}} \boldsymbol{r} \implies \boldsymbol{w} \xrightarrow{\text{rfx}} \boldsymbol{r}$$

Else, there is an interfering transmitter $\boldsymbol{w}'$ where $\overset{😇}{\boldsymbol{w}'} \xrightarrow{rfx} \overset{😈}{\boldsymbol{r}}$

rfx Non-Interference Detects Spectre v1 Leakage

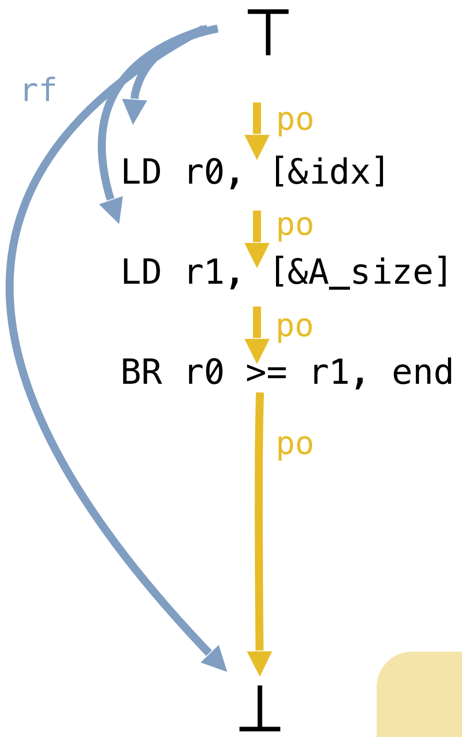
High Level Idea: Architectural Noninterference $\longrightarrow$ Microarchitectural Noninterference **else, microarch. leakage**

Example:

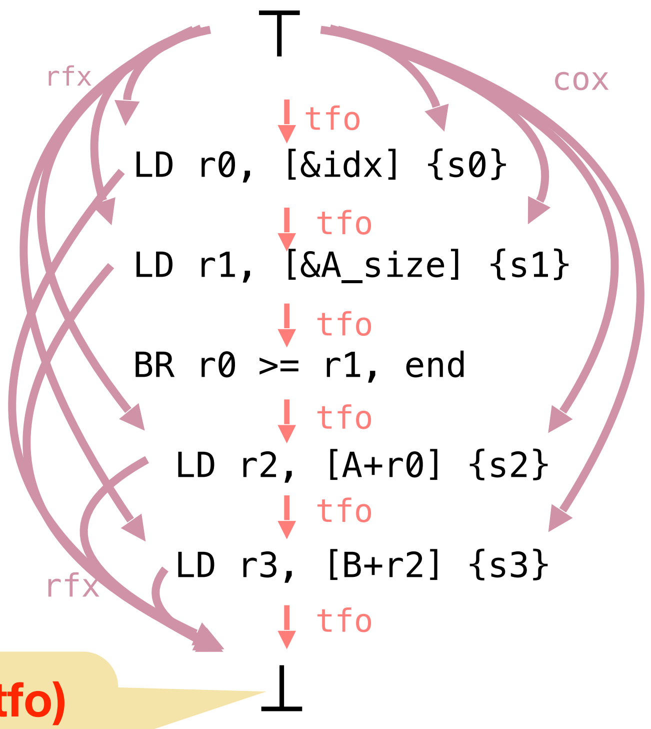
Spectre V1

```
if (idx < A_size) {
  char secret = A[idx];
  tmp = B[secret];
}
```

Architectural execution:



Microarchitectural execution:



Transient fetch order (tfo) models the **transient execution paths** of a program.

rfx Non-Interference Detects Spectre v1 Leakage

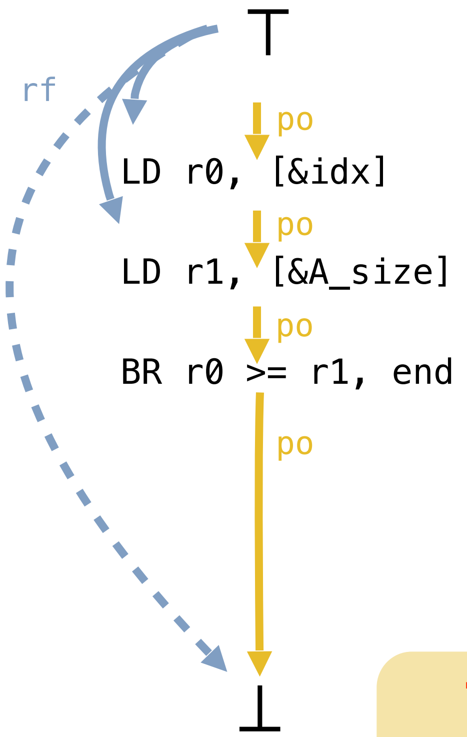
High Level Idea: Architectural Noninterference → Microarchitectural Noninterference **else, microarch. leakage**

Example:

Spectre V1

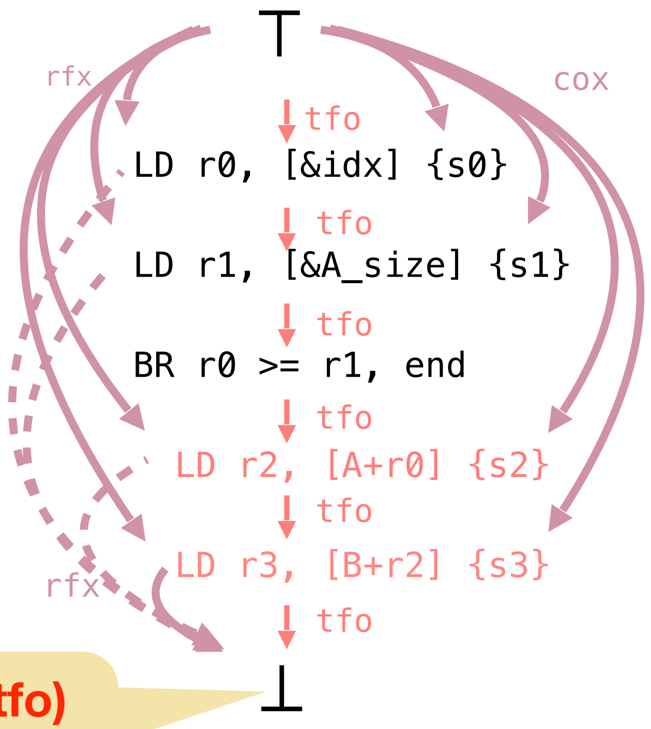
```
if (idx < A_size) {
  char secret = A[idx];
  tmp = B[secret];
}
```

Architectural execution:



rfx *noninterference violations*

Microarchitectural execution:

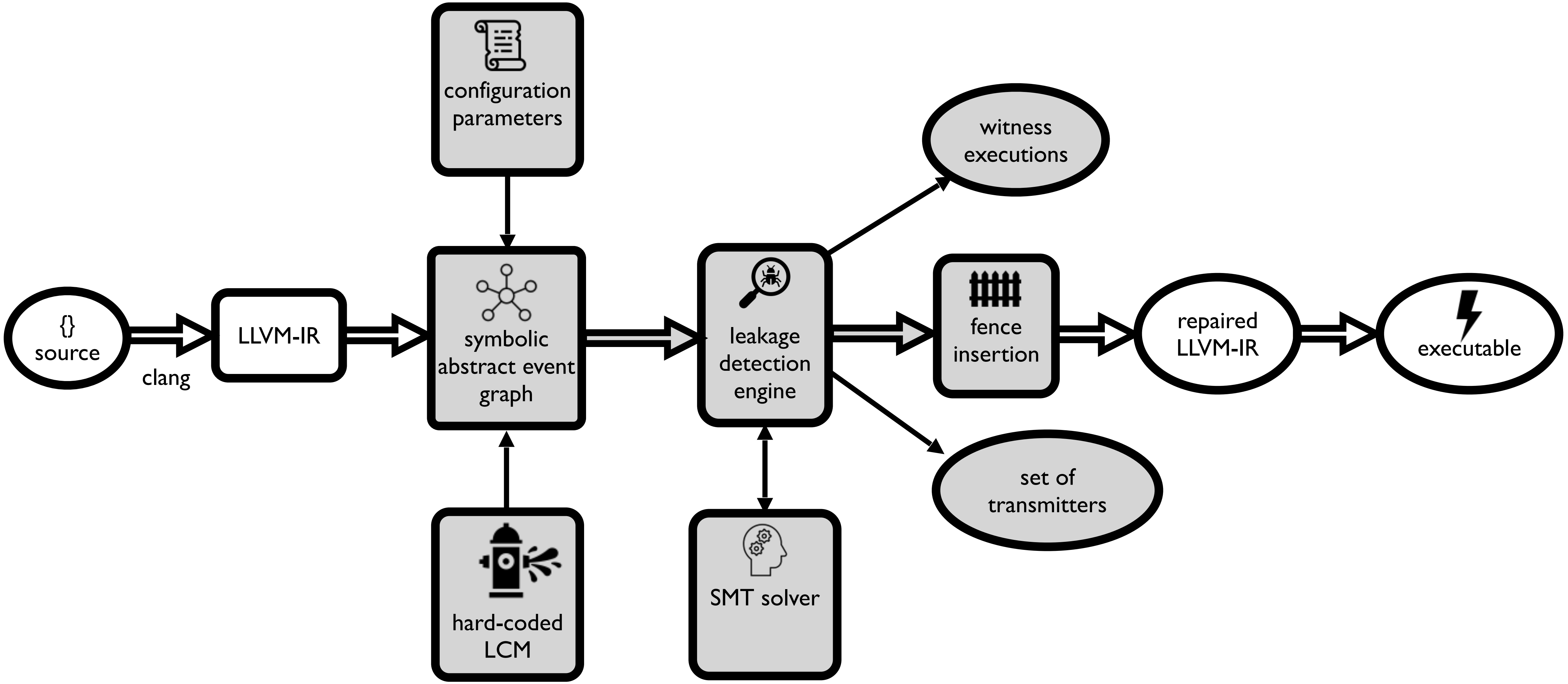


Transient fetch order (tfo) models the **transient execution paths** of a program.

Roadmap

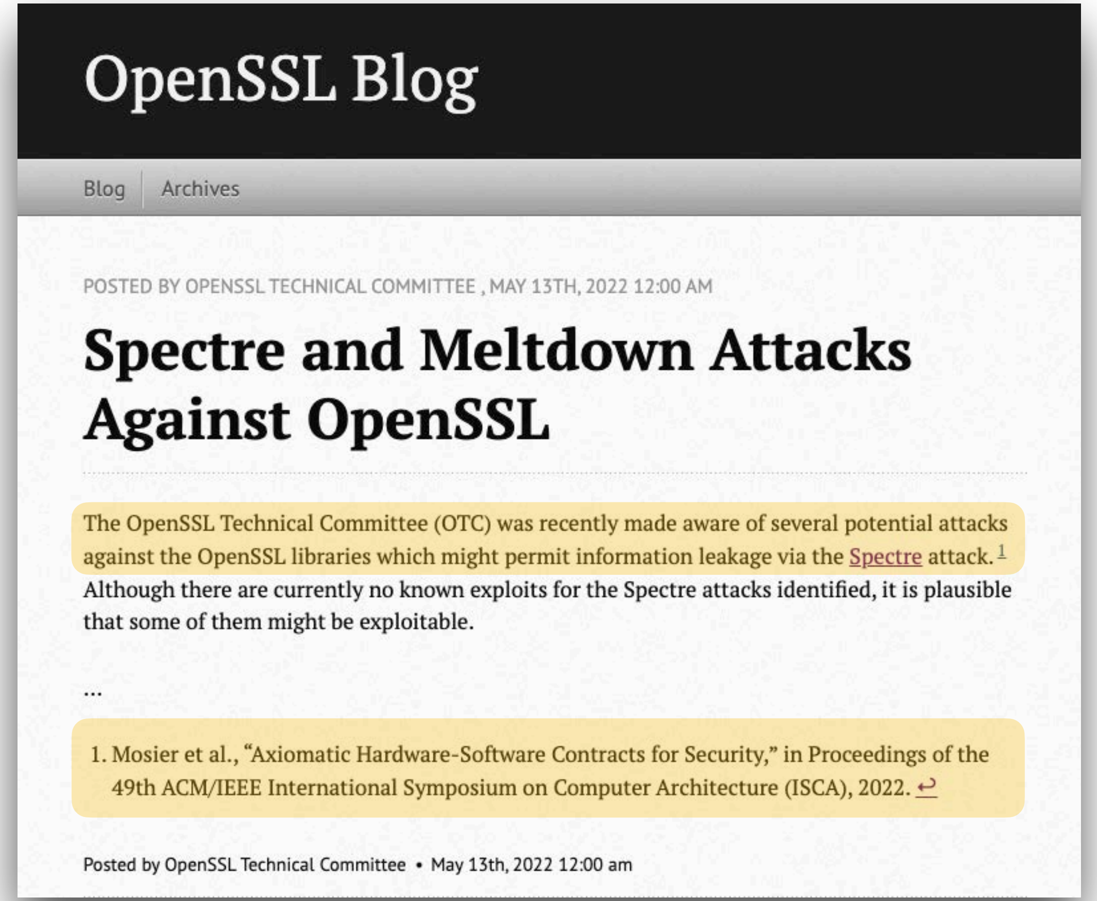
- **Background:** Hardware-Software Contracts & Memory Consistency Models
- Building Blocks of **Microarchitectural Leakage**
- **Leakage Containment Models:** Modeling Microarchitectural Leakage
- **Clou:** Detecting and Mitigating Microarchitectural Leakage in Programs

Clou Automats Leakage Detection



Clou Found Bugs in Real-World Code

- More scalable than previous tools:
  - Binsec/Haunted [Daniel+ NDSS21]
  - Pitchfork [Cauligi+ PLDI20])
- Reported **7 new Spectre v4 vulnerabilities** in libsodium
- Reported **5 new Spectre v1 vulnerabilities** in OpenSSL



OpenSSL Blog

POSTED BY OPENSSL TECHNICAL COMMITTEE , MAY 13TH, 2022 12:00 AM

Spectre and Meltdown Attacks Against OpenSSL

The OpenSSL Technical Committee (OTC) was recently made aware of several potential attacks against the OpenSSL libraries which might permit information leakage via the Spectre attack.[1] Although there are currently no known exploits for the Spectre attacks identified, it is plausible that some of them might be exploitable.

...

1. Mosier et al., "Axiomatic Hardware-Software Contracts for Security," in Proceedings of the 49th ACM/IEEE International Symposium on Computer Architecture (ISCA), 2022. ↩

Posted by OpenSSL Technical Committee • May 13th, 2022 12:00 am

Key Takeaways

- LCMs expose microarchitectural **control** and **data** flow to software to reason about the security implications of hardware on software
- LCMs can precisely **pinpoint a wide variety of leakage** in different microarchitectures
- LCMs **abstract away unnecessary implementation details**
- LCMs are **easy to adopt**