Axiomatic Hardware-Software Contracts for Security*

Nicholas Mosier[1], Hanna Lachnitt[1], Hamed Nemati[1,2], Caroline Trippel[1]

April 6, 2022 • Stanford Security Workshop

[1]Stanford University [2]CISPA Helmholtz Center for Information Security

**to appear in the 49th ACM/IEEE International Symposium on Computer Architecture (ISCA), June 2022*

software

compiler

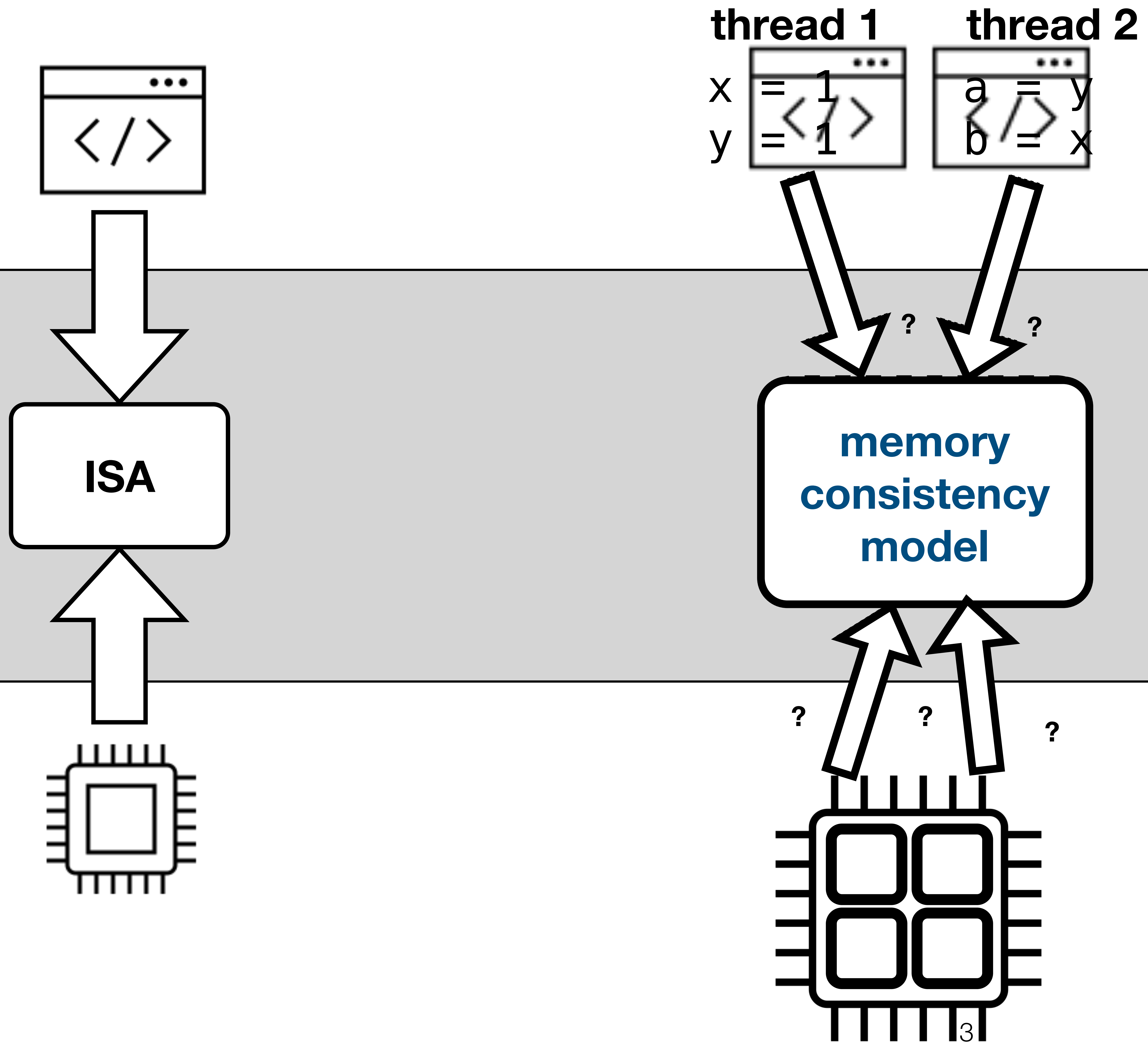
instruction set architecture

- registers
- memory
- instructions

microarchitecture

hardware

Can `a = 1, b = 0`? ⟺ Can we observe a **re-ordering** of T1's stores or T2's loads?



thread 1

```
x = 1
y = 1
```

thread 2

```
a = y
b = x
```

It **depends** on the **architecture!**

✗ intel

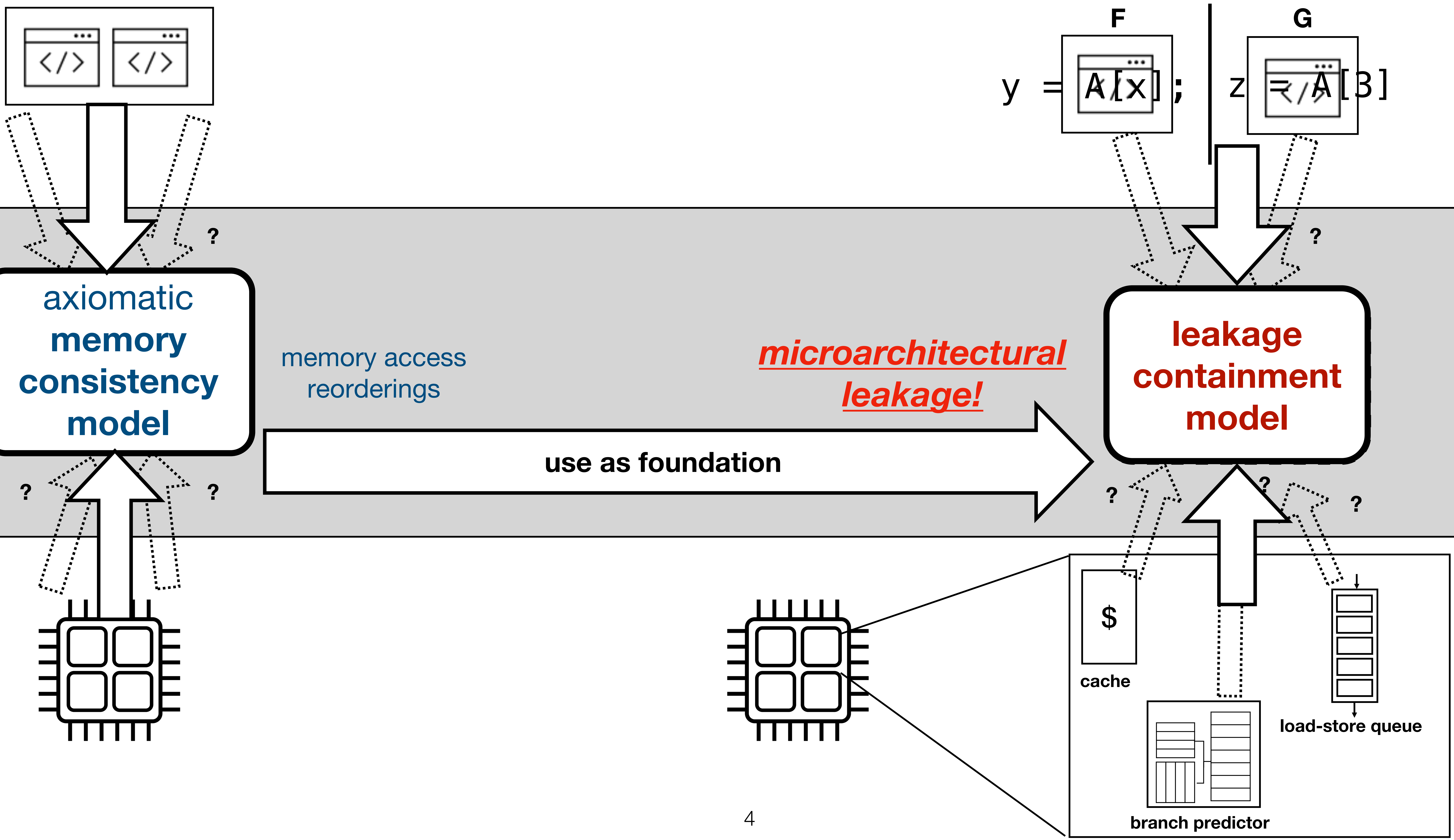
✓ arm

✓ IBM

✓ RISC-V

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

Can G observe **leakage** of F's variable x?



axiomatic **memory consistency model**

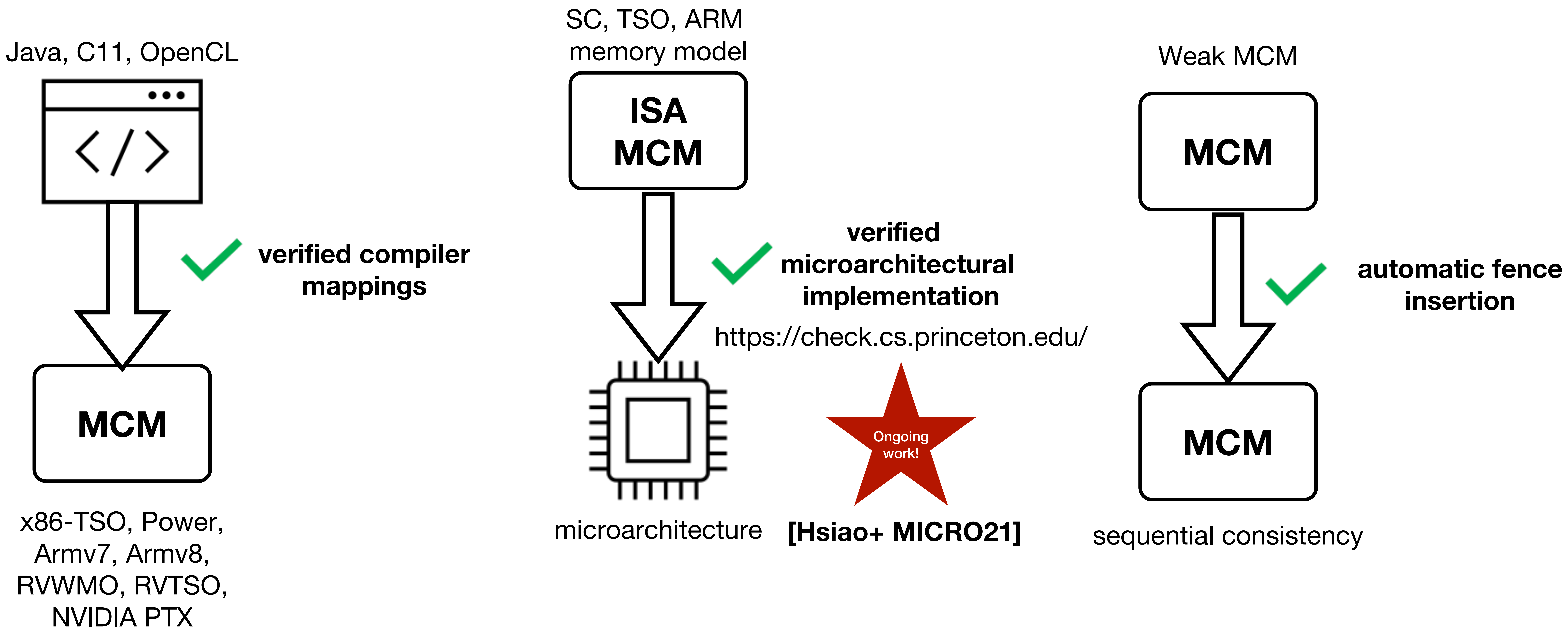
memory access reorderings

microarchitectural leakage!

use as foundation

leakage containment model

Axiomatic **MCMs** have spawned an **ecosystem of tools and research**



Java, C11, OpenCL

✓ **verified compiler mappings**

MCM

x86-TSO, Power,
Armv7, Armv8,
RVWMO, RVTSO,
NVIDIA PTX

SC, TSO, ARM
memory model

ISA
MCM

✓ **verified microarchitectural implementation**

https://check.cs.princeton.edu/

Ongoing work!

microarchitecture

[Hsiao+ MICRO21]

Weak MCM

MCM

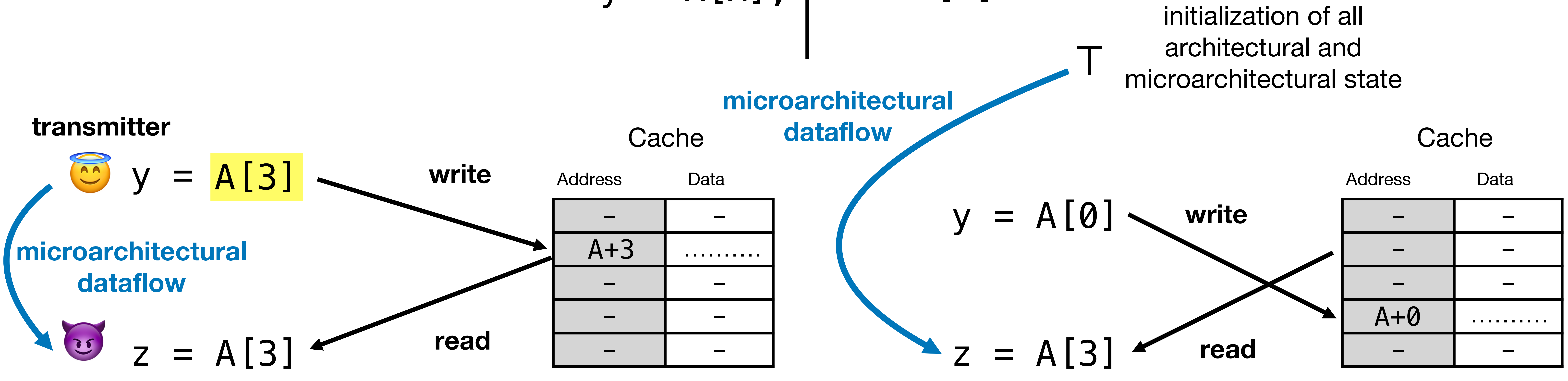
✓ **automatic fence insertion**

MCM

sequential consistency

Microarchitectural dataflow enables **leakage**

F	**G**
`y = A[x];`	`z = A[3]`



transmitter
😇 y = A[3]
microarchitectural dataflow
😈 z = A[3]
receiver
write
read
Cache
microarchitectural dataflow
T
initialization of all architectural and microarchitectural state
y = A[0]
write
z = A[3]
read
Cache

Cache

Address	Data
–	–
A+3	....
–	–
–	–
–	–

Cache

Address	Data
–	–
–	–
–	–
A+0	....
–	–

cache hit (5 ns)

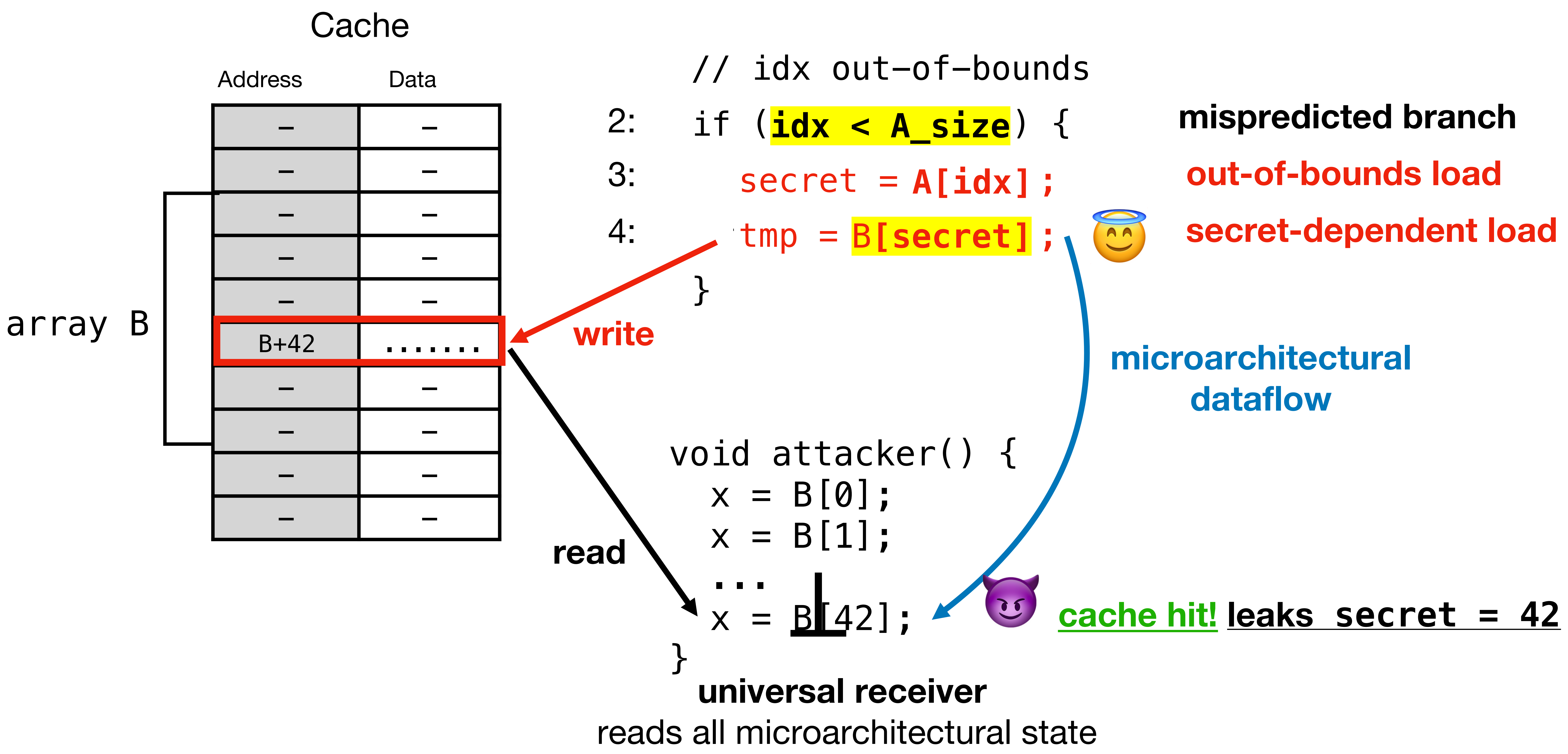
cache miss (50 ns)

leaks: x = 3

leaks: x ≠ 3

Microarchitectural control-flow increases the **scope** of what can **leak**

Spectre v1: Bounds Check Bypass



Cache

Address	Data
–	–
–	–
–	–
–	–
–	–
B+42	....
–	–
–	–
–	–
–	–

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

mispredicted branch

out-of-bounds load

secret-dependent load

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

cache hit! **leaks `secret = 42`**

universal receiver
reads all microarchitectural state

Overview

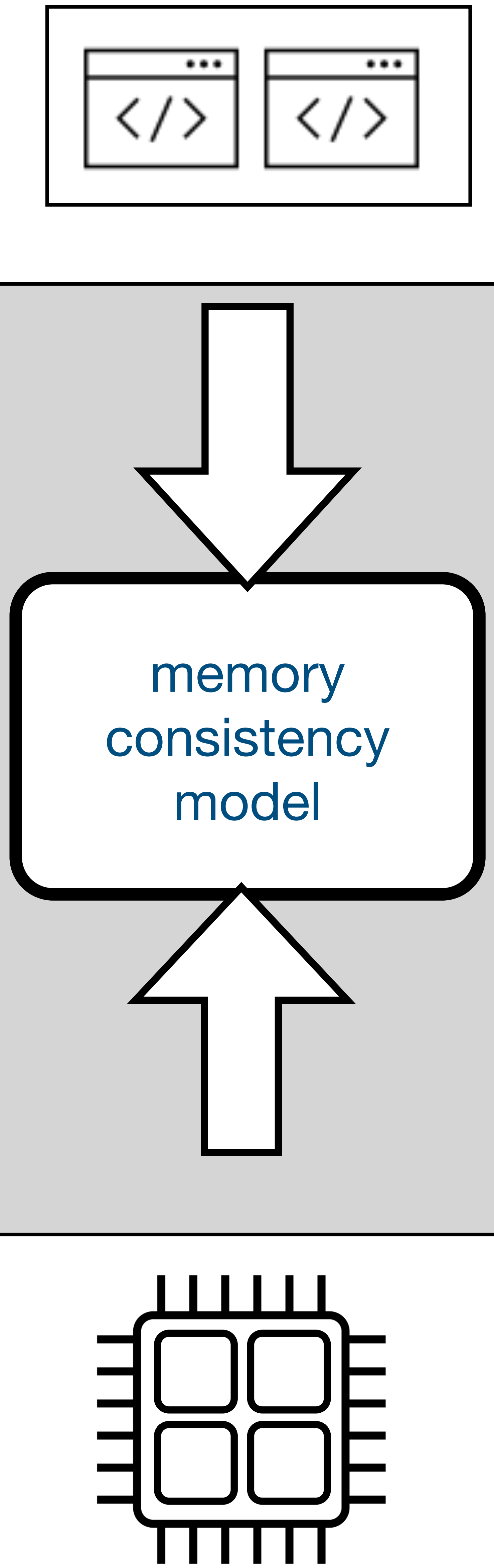
1. **Leakage Containment Models (LCMs)**: Axiomatic Security Contracts
2. **`Clou`**: Detect and Mitigate Microarchitectural Program Leakage with LCMs

Overview

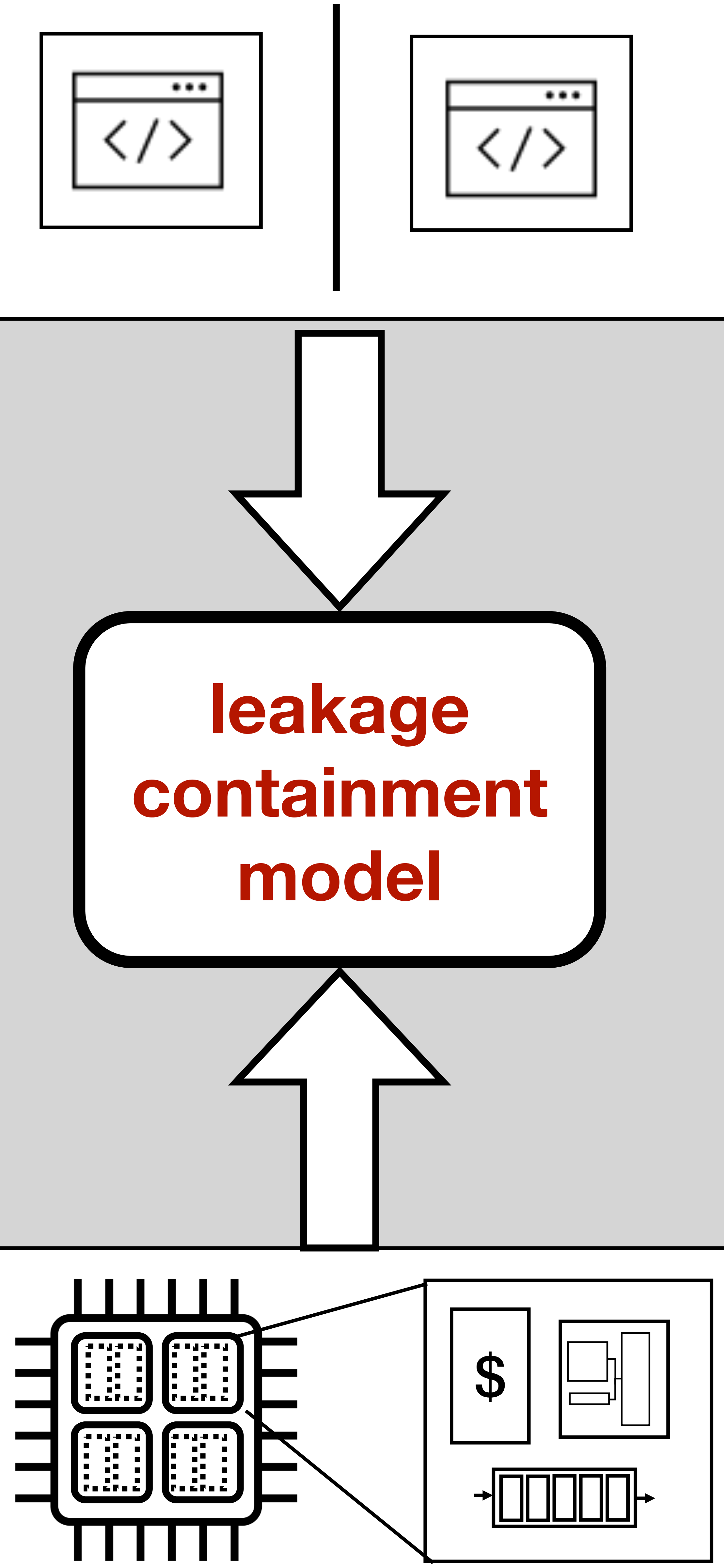
1. **Leakage Containment Models (LCMs)**: Axiomatic Security Contracts
2. **`Clou`**: Detect and Mitigate Microarchitectural Program Leakage with LCMs

Leakage Containment Models (LCMs) extend **MCMs** to provide **microarchitectural semantics**

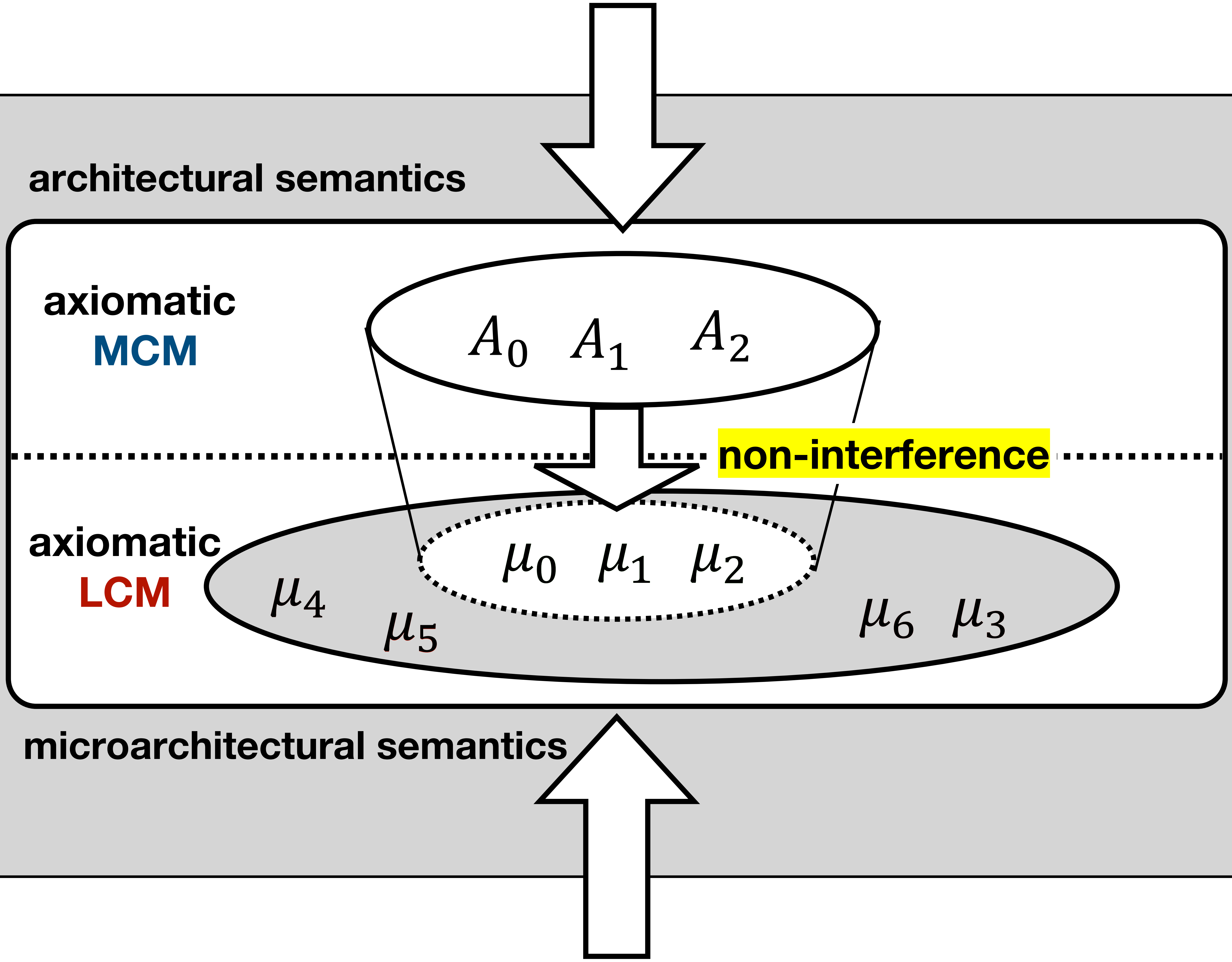
Legend **leakage containment model**
(memory consistency model)

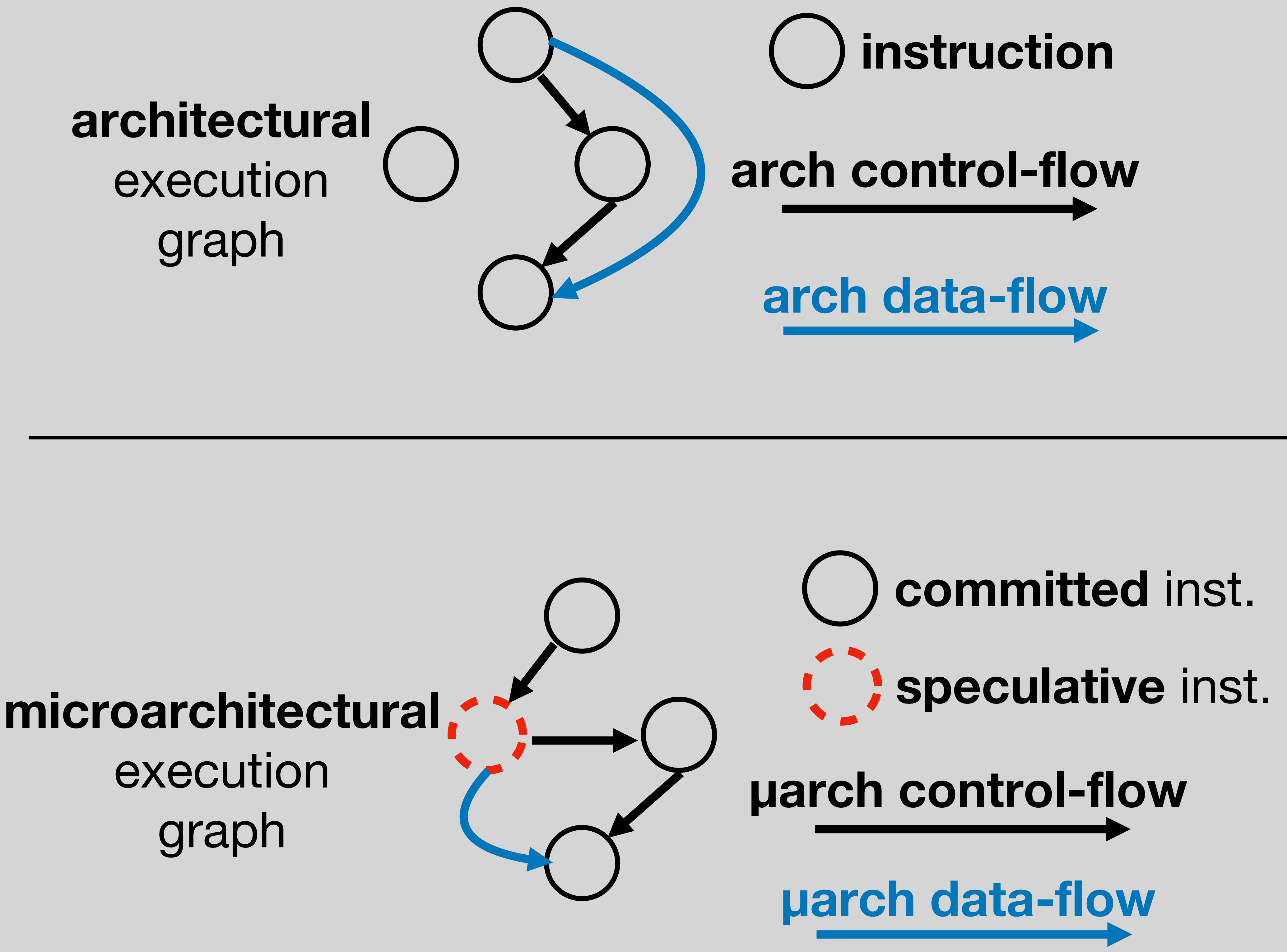


- hardware-software contract that exposes **leakage** to software
  (reorderings)
- communication through **xstate**
  (memory)
- model **microarchitectural control-flow + data-flow**
  (architectural)
- **identify unwanted leakage**
  (reorderings)
- eliminate unwanted **speculative leakage** with **fences**
  (reorderings)



LCMs **compare** MCMs' **architectural semantics** to LCMs' new **microarchitectural semantics.**





LCM microarchitectural semantics mirror **MCM architectural semantics**

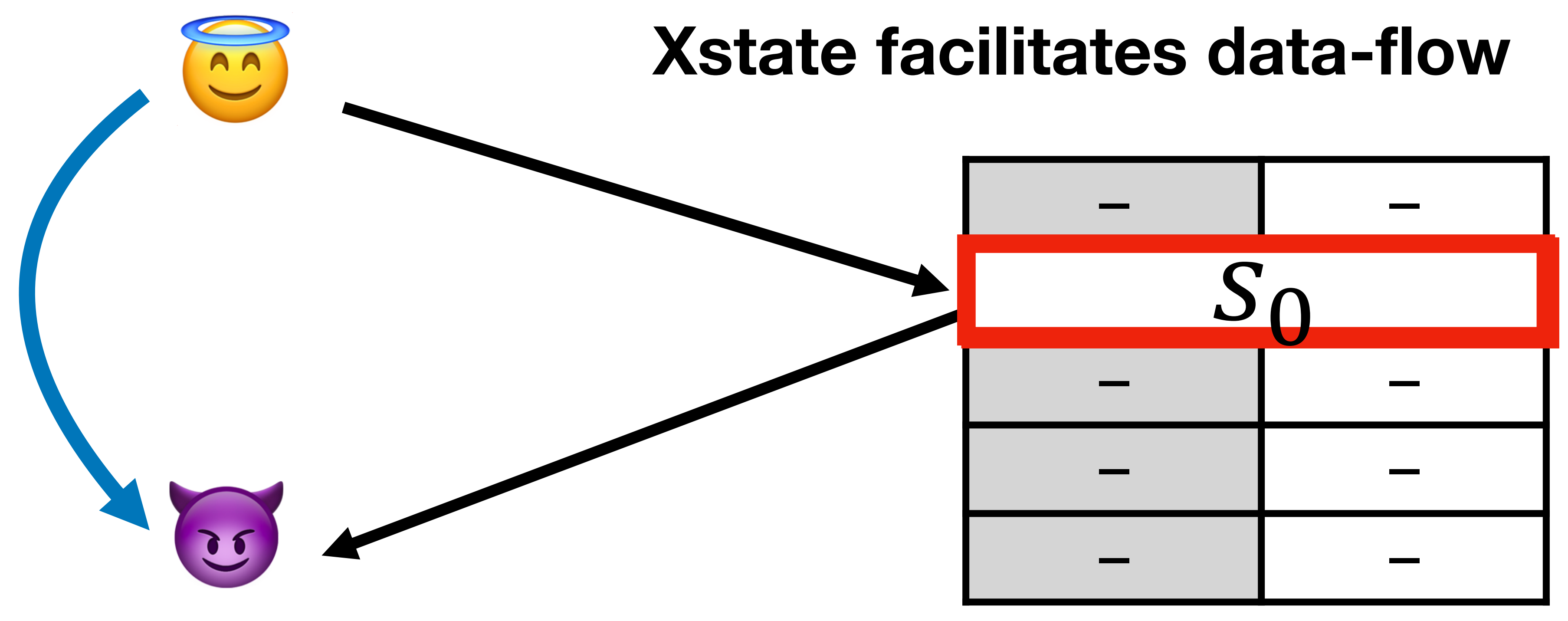
	MCMs	LCMs
abstraction level	architecture	microarchitecture
communication medium	memory location	xstate variable
control-flow		
data-flow		

LCMs model **microarchitectural data-flow** through **xstate**

- **Extra-architectural state (xstate)**: microarchitectural state not corresponding to architectural state.

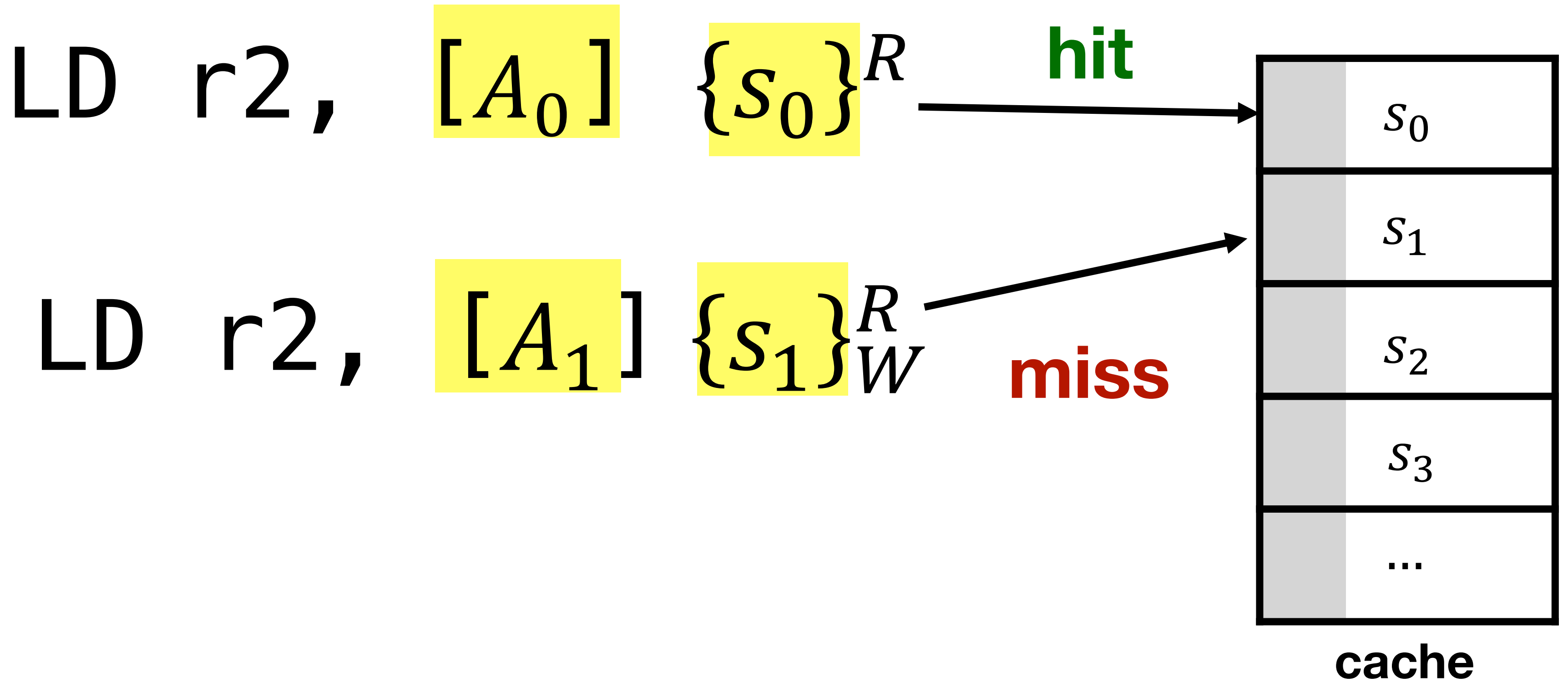
- **xstate variables** represent abstract microarchitectural data-flow elements

- Instructions read and/or write different xstate variables depending on execution context.

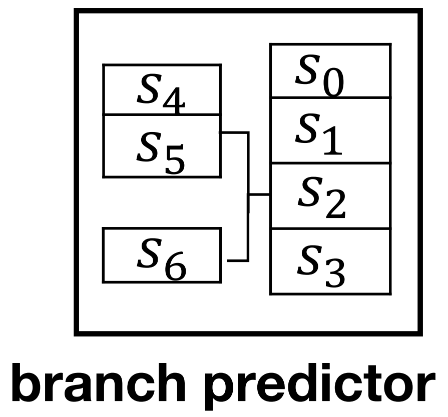


Xstate facilitates data-flow

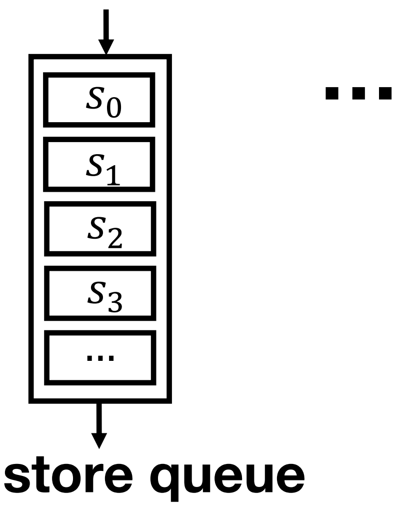
xstate examples



cache



branch predictor



load-store queue

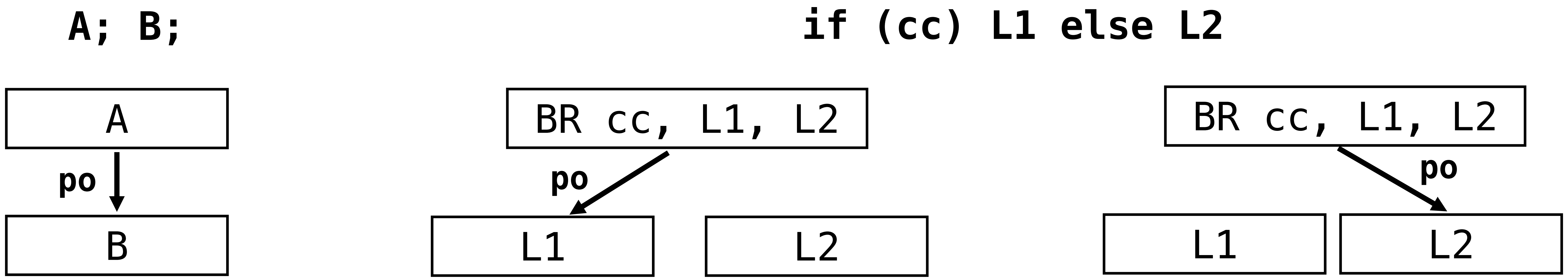
For now, we'll focus on cache xstate to model leakage through the memory system.

	MCMs	LCMs
abstraction level	architecture	microarchitecture
communication medium	memory location	xstate variable
control-flow	po	tfo
data-flow		

LCMs introduce a new **speculative control-flow** semantics

po: program order

Decides the architectural execution path

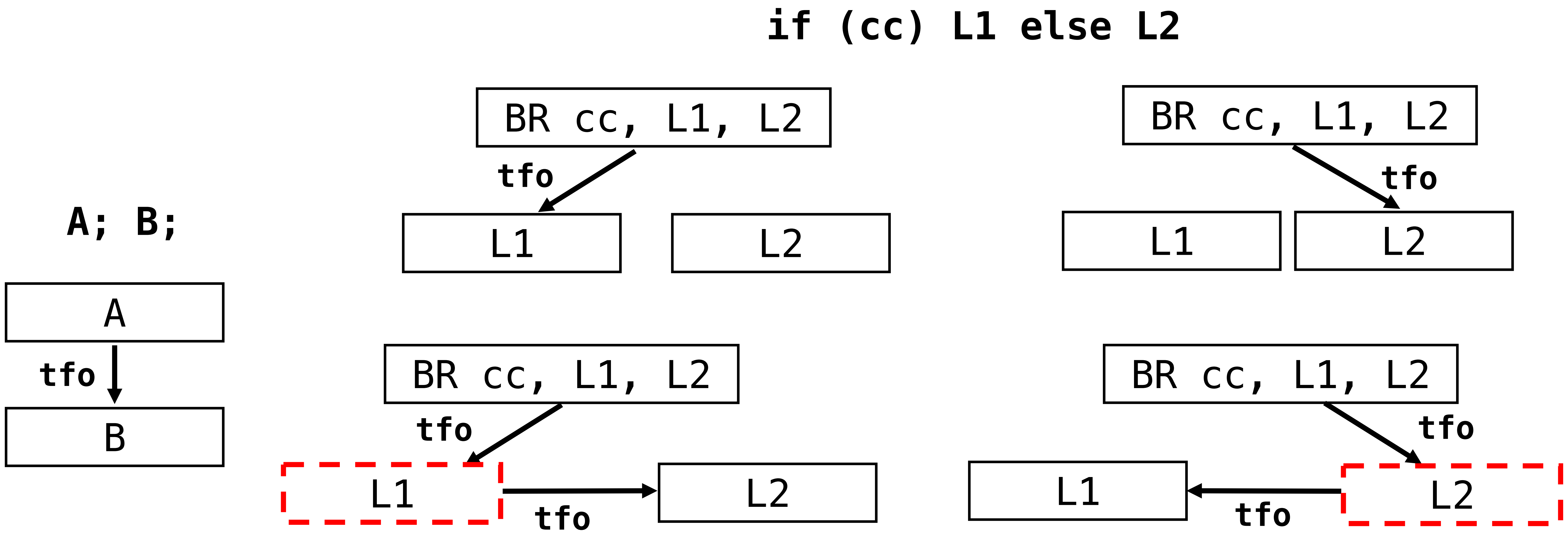


architecture MCM

microarchitecture LCM

tfo: transient fetch order

Decides the microarchitectural execution path



Legend:

speculative execution

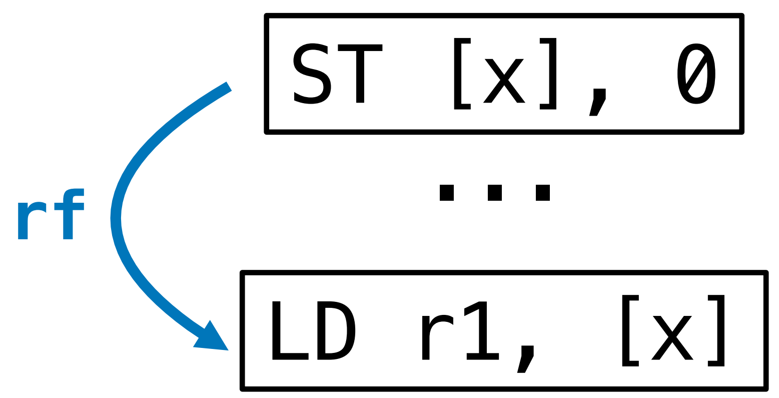
LCM's **microarchitectural dataflow semantics** model information flow through **xstate**

MCM

architectural communication:
dynamic data-flows through **memory**

rf, co, fr

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



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

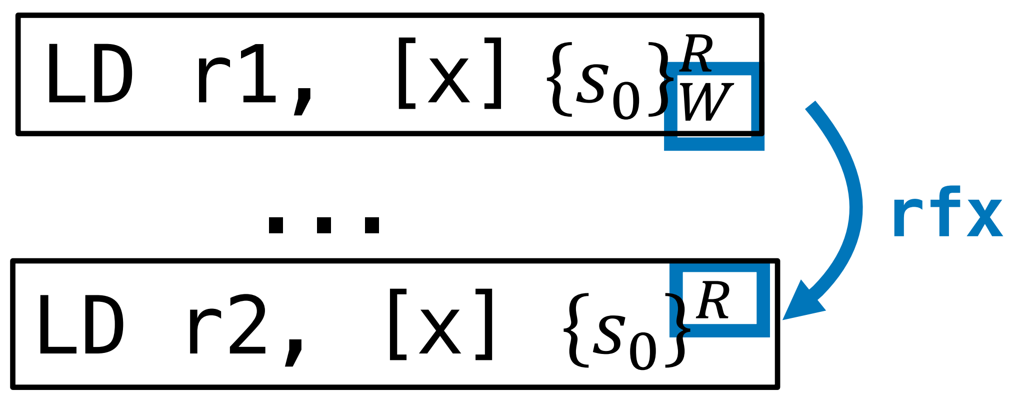
reads-from (rf):
relates (store, load)
if load reads from store

LCM

microarchitectural communication:
dynamic data-flows through **xstate**

rfx, cox, frx

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



LD r1, [x] $\{s_0\}^{R}_{\boxed{W}}$

...

LD r2, [x] $\{s_0\}^{\boxed{R}}$

reads-from xstate:
relates a xstate write to an
xstate read that reads from it

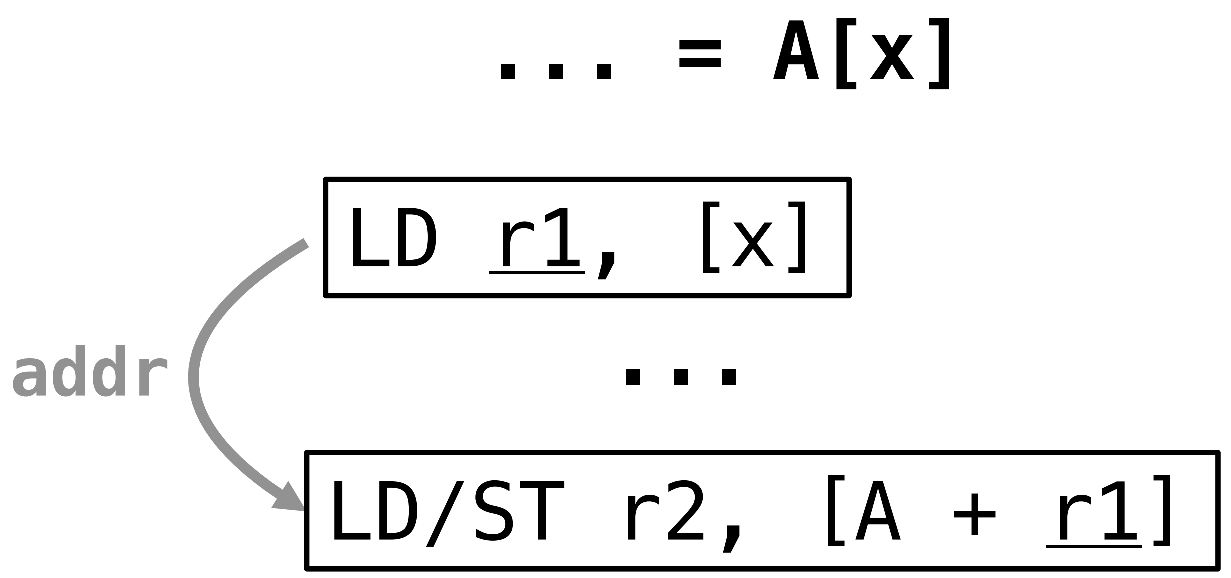
Dependencies model syntactic dependencies through registers

MCM + LCM

dependencies

addr, data, ctrl

syntactic data-flow through registers



```
... = A[x]

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

address (addr): relates load to access that uses load in address computation

Matching architectural and microarchitectural semantics imply **leakage-free execution**

High level: architectural non-interference $\implies$ microarchitectural non-interference

`rfx` **noninterference** (😇↛😈) holds

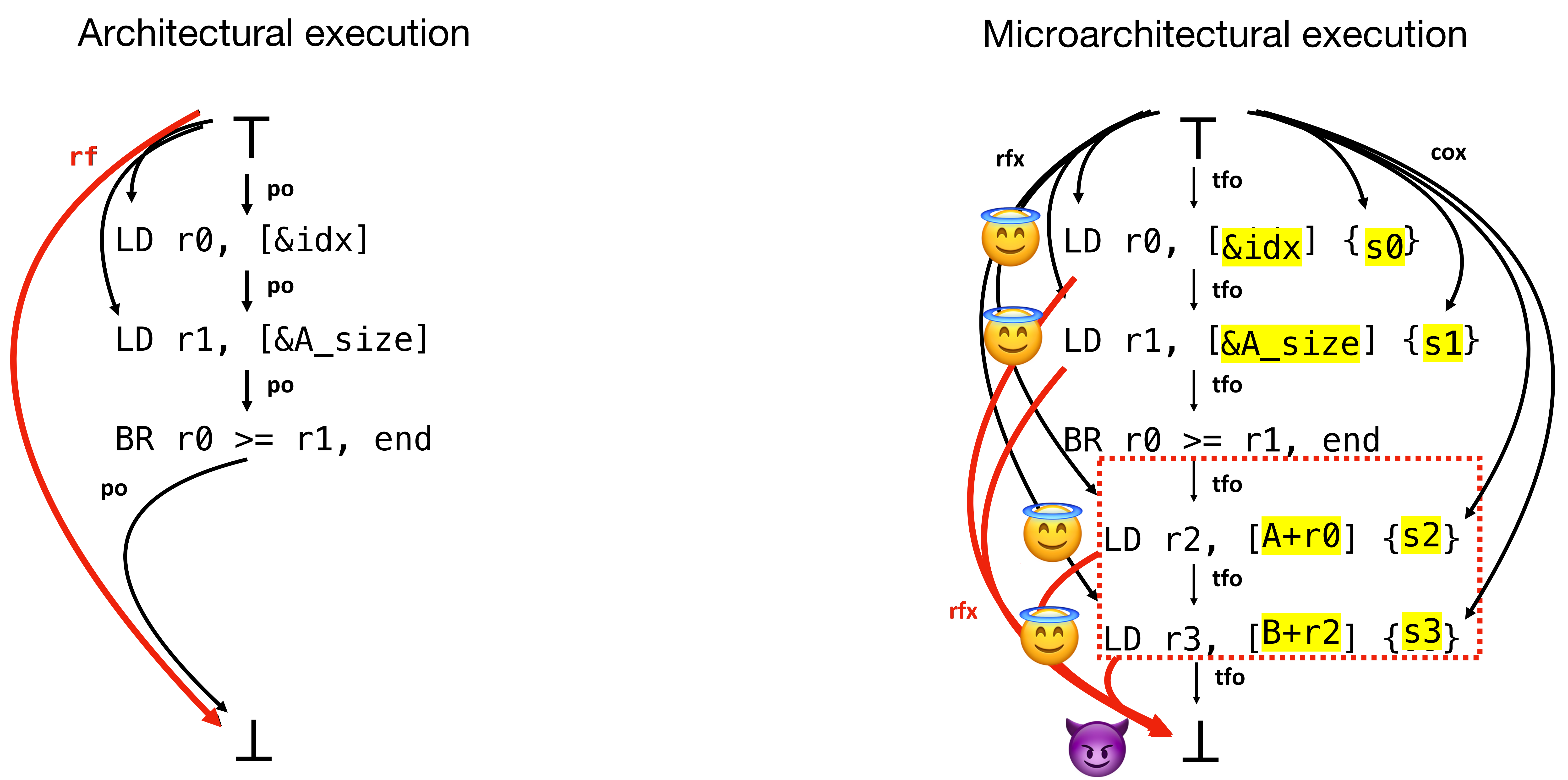
if for all writes w and reads r,

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

otherwise there's an interfering transmitter w' where $w' \xrightarrow{\text{rfx}} r$

😇 😈

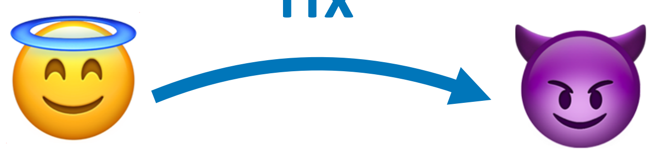
`rfx` non-interference detects leakage in Spectre v1



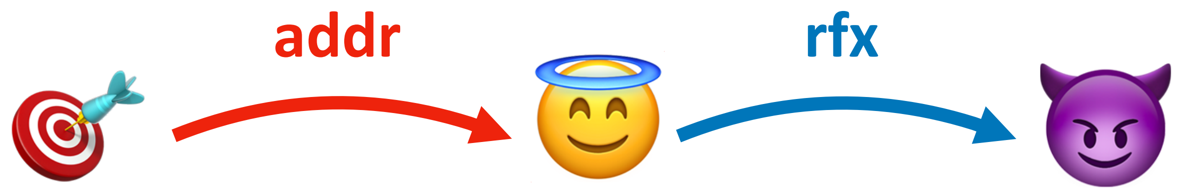
… and in many other speculative and non-speculative attacks.

LCMs introduce a **new taxonomy** for **classifying** xstate **transmitters** by severity

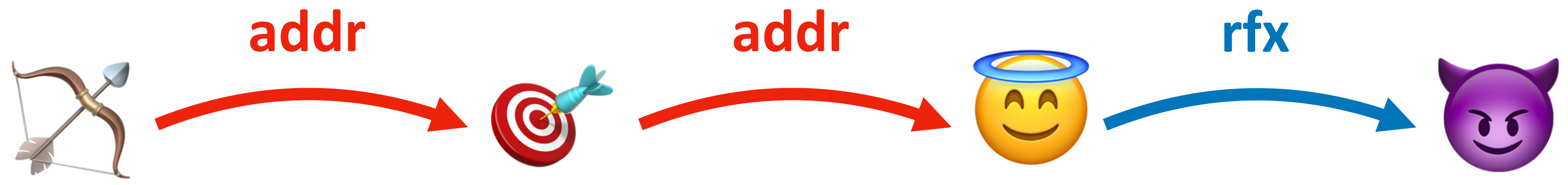
address transmitter **(!)**



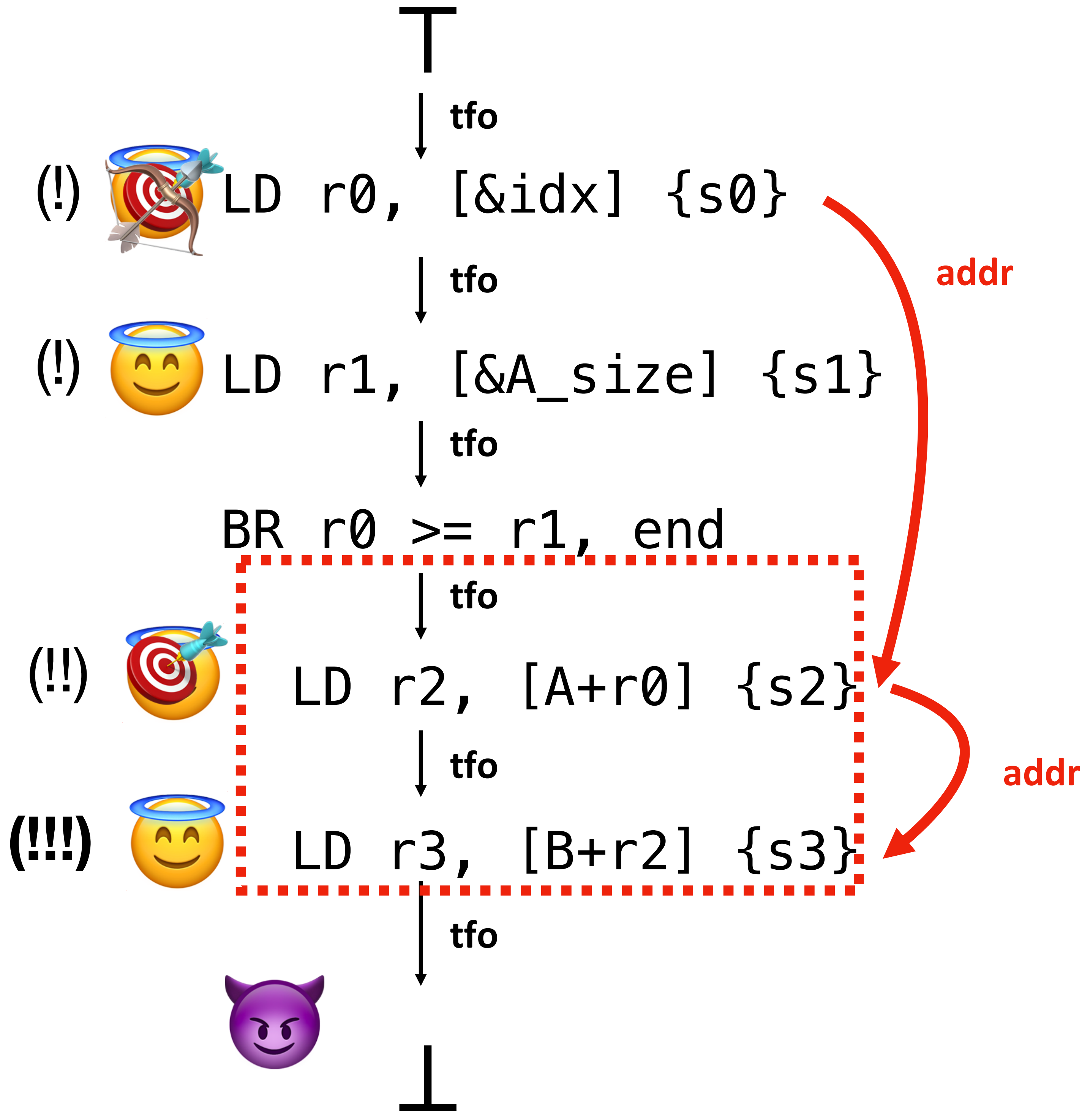
data transmitter **(!!)**



universal data transmitter **(!!!)**



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



⊤

tfo

(!) LD r0, [&idx] {s0}

tfo

(!) LD r1, [&A_size] {s1}

tfo

BR r0 >= r1, end

tfo

(!!) LD r2, [A+r0] {s2}

tfo

(!!!) LD r3, [B+r2] {s3}

tfo

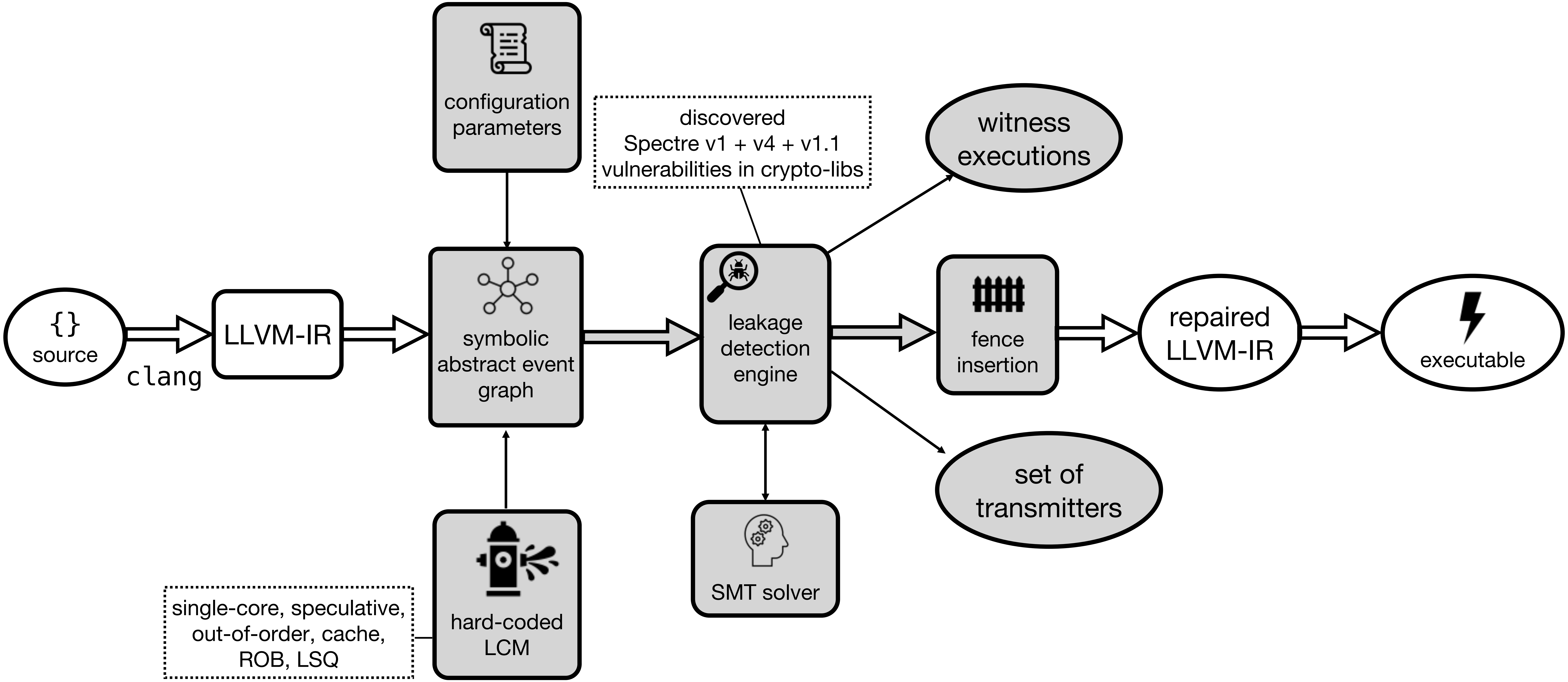
⊥

Overview

1. **Leakage Containment Models (LCMs):** Axiomatic Security Contracts

2. **`Clou`**: Detecting and Mitigating Microarchitectural Program Leakage with LCMs

`Clou`: a compiler pass to detect and mitigate **speculative universal data leakage** using **LCMs**



`Clou` is **fast**, **scalable**, and has found **bugs** in real-world code

- Successfully detects all leakage in benchmarks: `PHT`, `STL`, `FWD`, `NEW`
- 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`

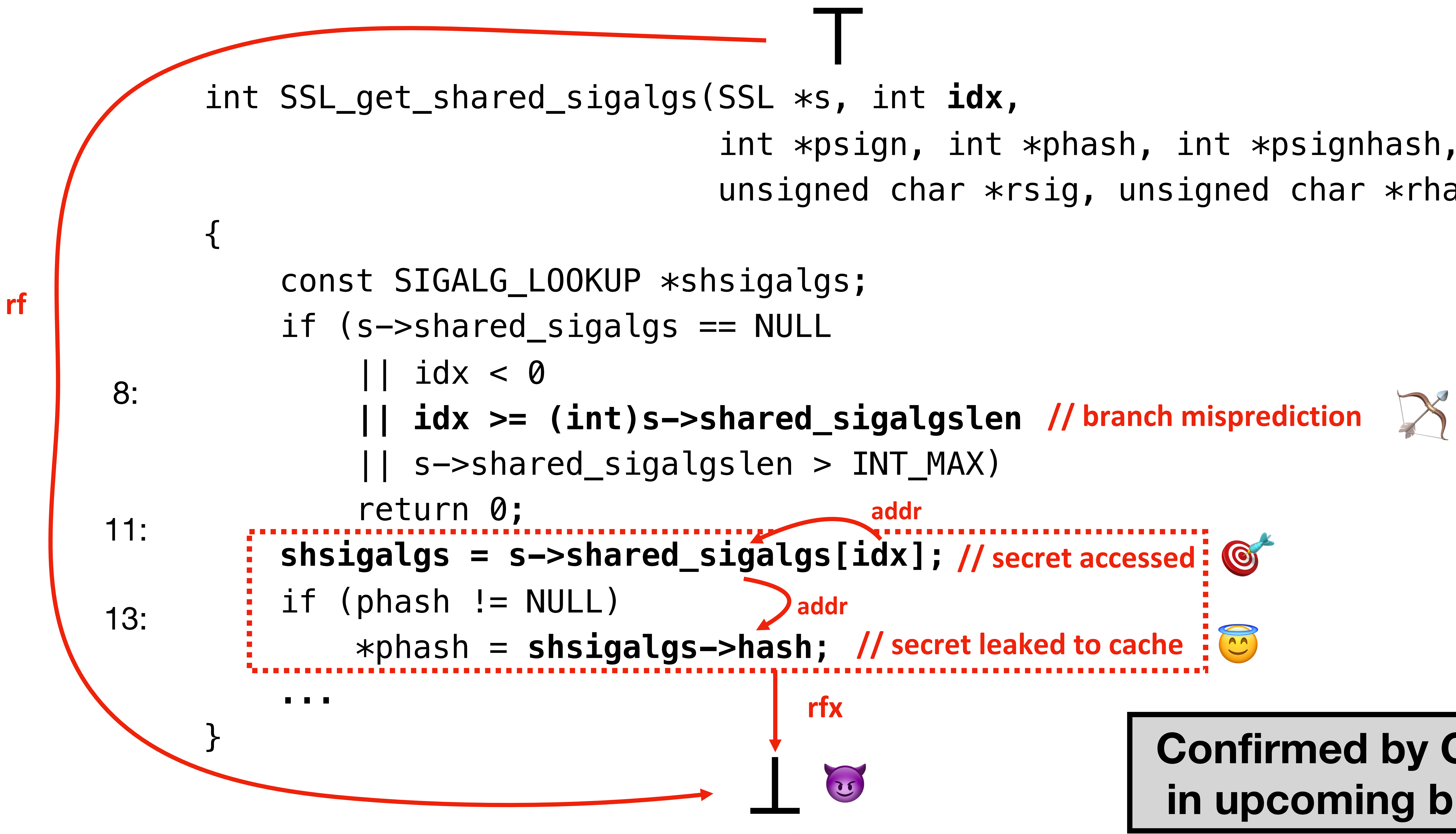
Runtimes (universal data leakage)

	BH runtime (s)	Clou runtime (s)
`PHT`	20.9	**2.8**
`STL`	6.1	**4.3**
`FWD`	589.3	**4.1**
`NEW`	32.5	**1.0**
`tea`	18.8	**1.14**
`donna`	TO	**112052**
`secretbox`	TO	**1008**
`ssl3-digest`	TO	**1318**
`mee-cbc`	TO	**95900**

Crypto-Library Analysis (universal data leakage)

	% funcs. analyzed	% LOC analyzed
`libsodium API`	100%	100%
`OpenSSL API`	90% / 81%	58% / 60%

Clou: OpenSSL Vulnerability



```
int SSL_get_shared_sigalgs(SSL *s, int idx,
                           int *psign, int *phash, int *psignhash,
                           unsigned char *rsig, unsigned char *rhash)
{
    const SIGALG_LOOKUP *shsigalgs;
    if (s->shared_sigalgs == NULL
        || idx < 0
        || idx >= (int)s->shared_sigalgslen  // branch misprediction 🏹
        || s->shared_sigalgslen > INT_MAX)
        return 0;
    shsigalgs = s->shared_sigalgs[idx]; // secret accessed 🎯
    if (phash != NULL)
        *phash = shsigalgs->hash;  // secret leaked to cache 😇
    ...
}
```

Confirmed by OpenSSL in upcoming blog post

Contributions

- Proposed **leakage containment models (LCMs)**, a novel hardware-software security contract
- Formally defined **microarchitectural leakage** using LCMs
- Demonstrated LCMs capture a **wide variety** of leakage
- Defined a **taxonomy** for classifying leakage by severity
- Developed **`Clou`**, a static analysis tool using an LCM to find speculative leakage in programs
- Found multiple confirmed **speculative execution vulnerabilities** in **crypto-libraries**