Hardware-Software Contracts for Secure Speculation

Jan Reineke @ UNIVERSITÄT DES SAARLANDES

Joint work with
Marco Guarnieri, Pepe Vila @ IMDEA Software, Madrid
Boris Köpf @ Microsoft Research, Cambridge, UK

Supported by the European Research Council and an
Intel Strategic Research Alliance (ISRA)

The Need for HW/SW Contracts

ISA: Benefits

$$\begin{tikzcd}
\text{High-level language} \arrow[d] \\
\boxed{\text{Instruction set architecture (ISA)}} \arrow[d] \\
\text{Microarchitecture}
\end{tikzcd}$$

Can program **independently** of microarchitecture

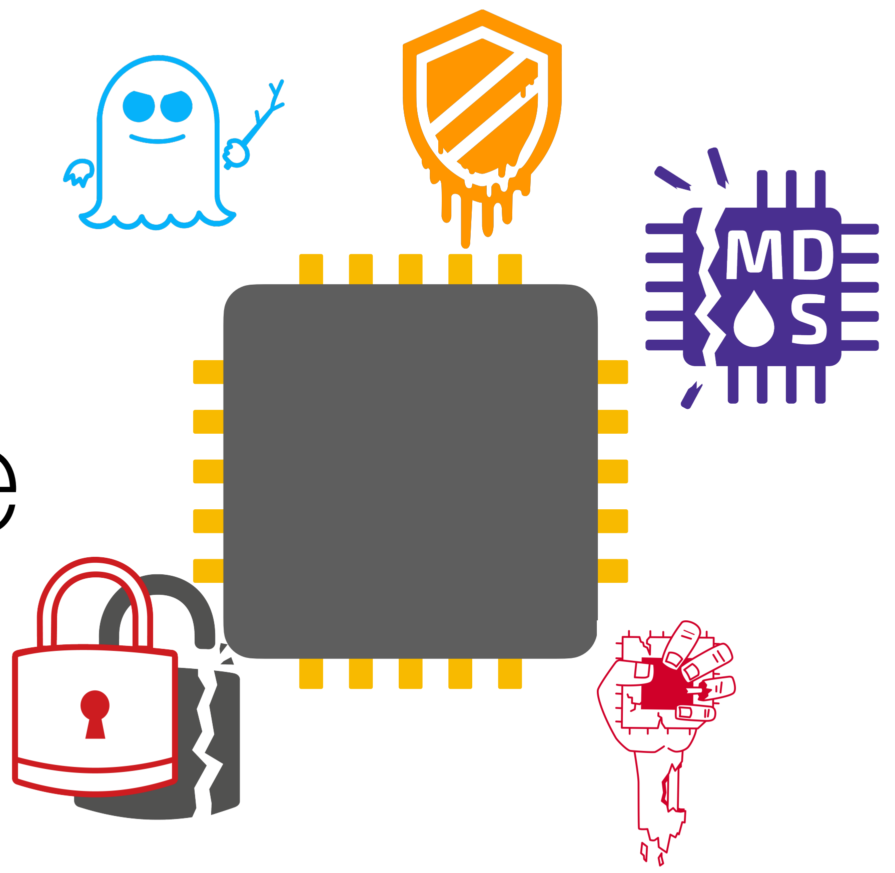
Can implement **arbitrary optimizations** as long as ISA semantics are obeyed

Inadequacy of the ISA: Side channels

High-level language

Instruction set architecture (ISA)

Microarchitecture



Impossible to program securely
- cryptographic algorithms?
- sandboxing untrusted code?

No guarantees about side channels

Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

A Way Forward: HW/SW Security Contracts

Can program **securely** on top of contract **independently** of microarchitecture

HW/SW contract = ISA + X

Succinctly captures possible information leakage

Can implement **arbitrary** ~~insecure~~ **optimizations** as long as contract is obeyed

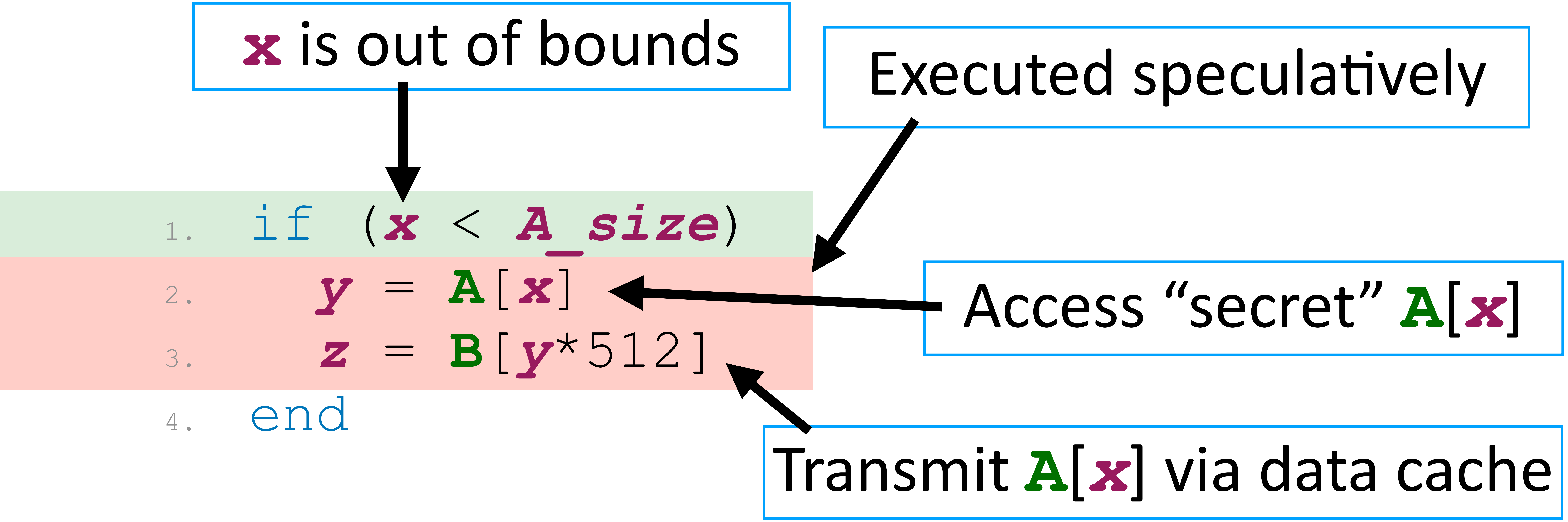
A Concrete Challenge: Spectre

SPECTRE

Exploits *speculative execution*

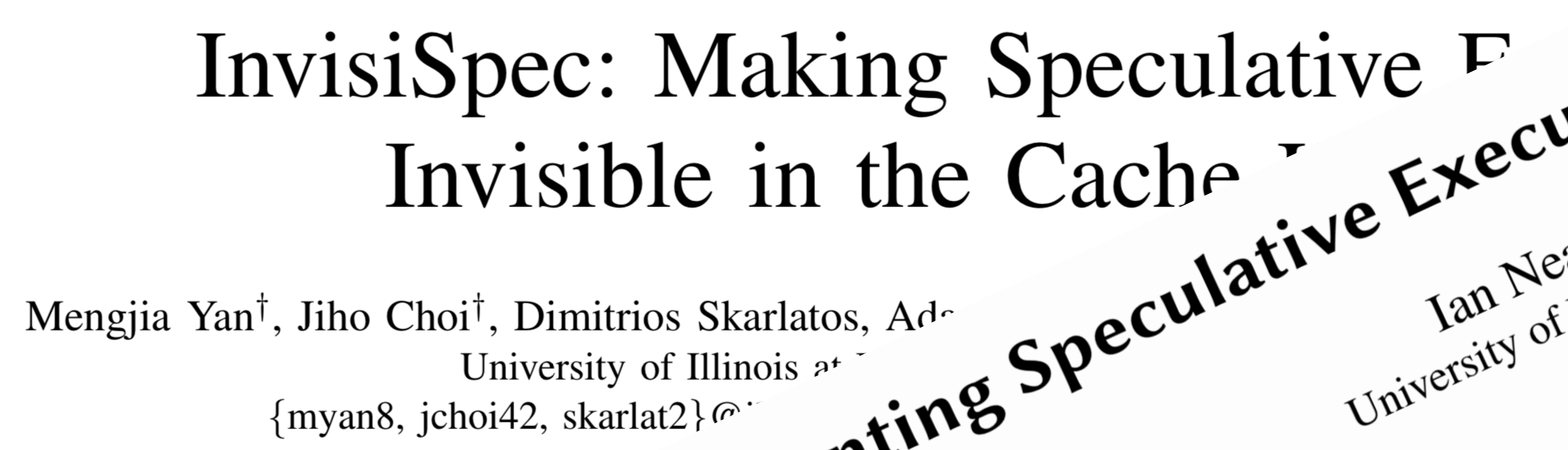
Almost *all* modern *CPUs* are *affected*

Example: Spectre v1 Gadget



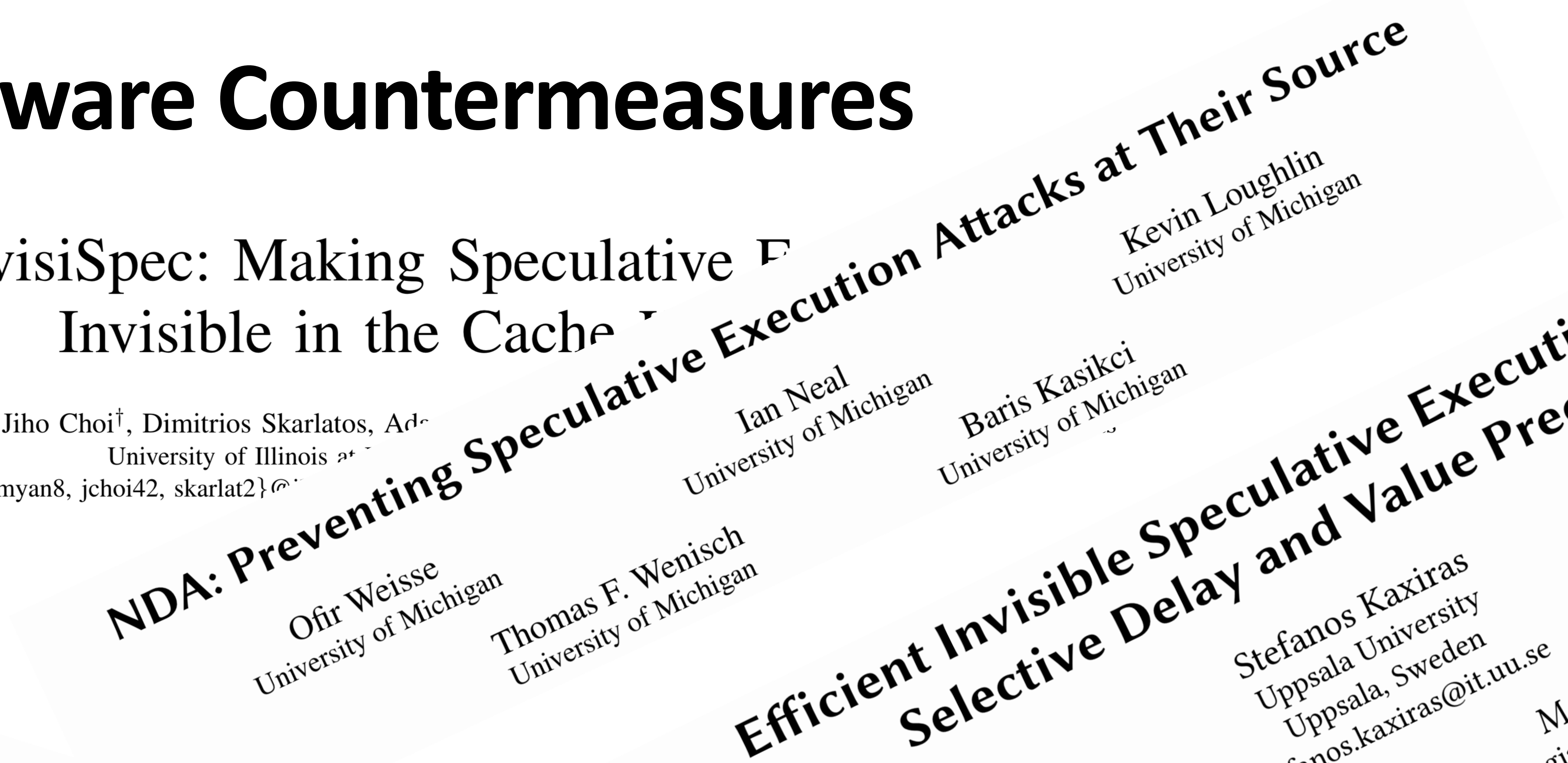
```
if (x < A_size)
    y = A[x]
    z = B[y*512]
end
```

Hardware Countermeasures



InvisiSpec: Making Speculative E
Invisible in the Cache

Mengjia Yan†, Jiho Choi†, Dimitrios Skarlatos, Ad
University of Illinois at
{myan8, jchoi42, skarlat2}@



NDA: Preventing Speculative Execution Attacks at Their Source

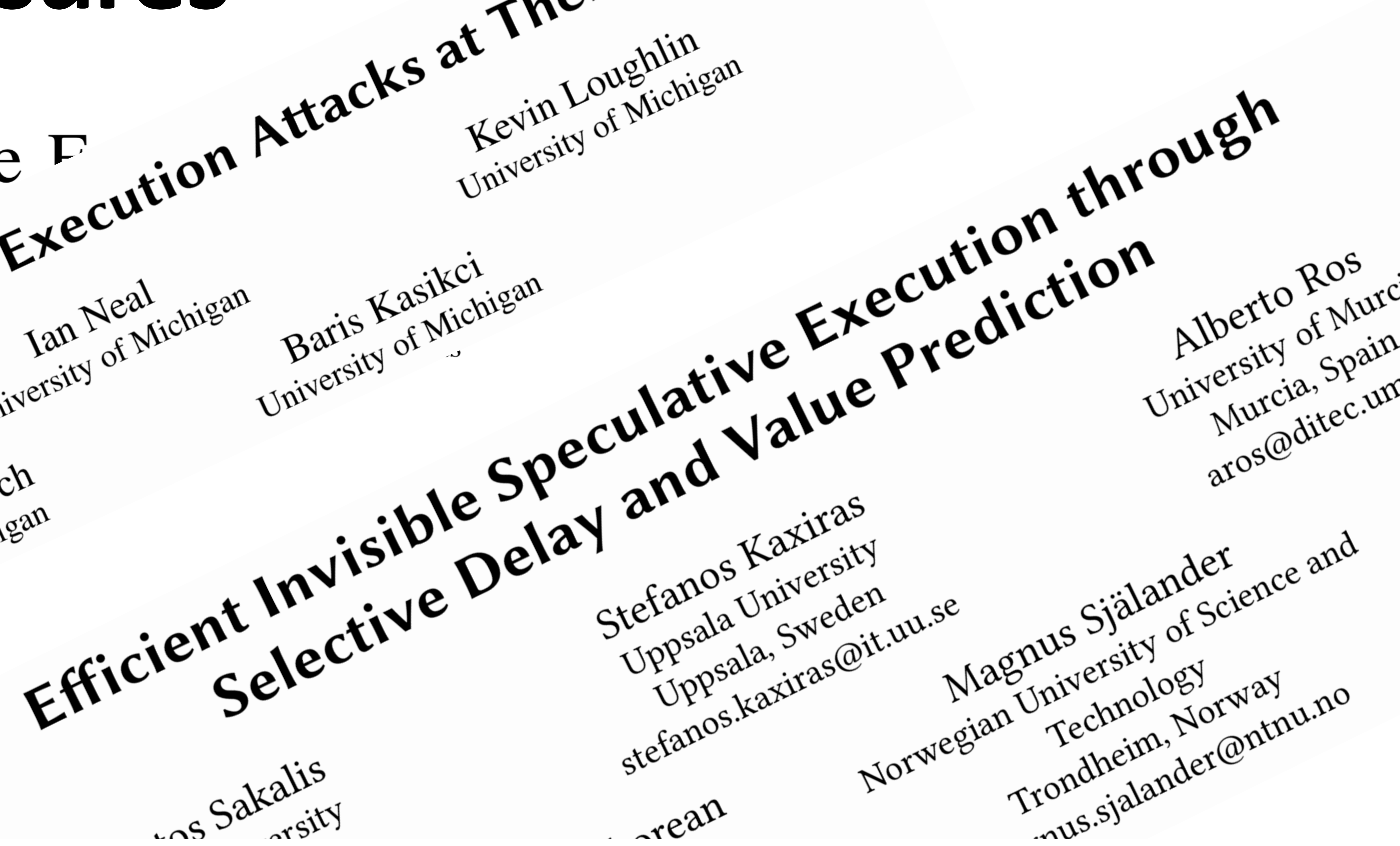
Ofir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Kevin Loughlin
University of Michigan



Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

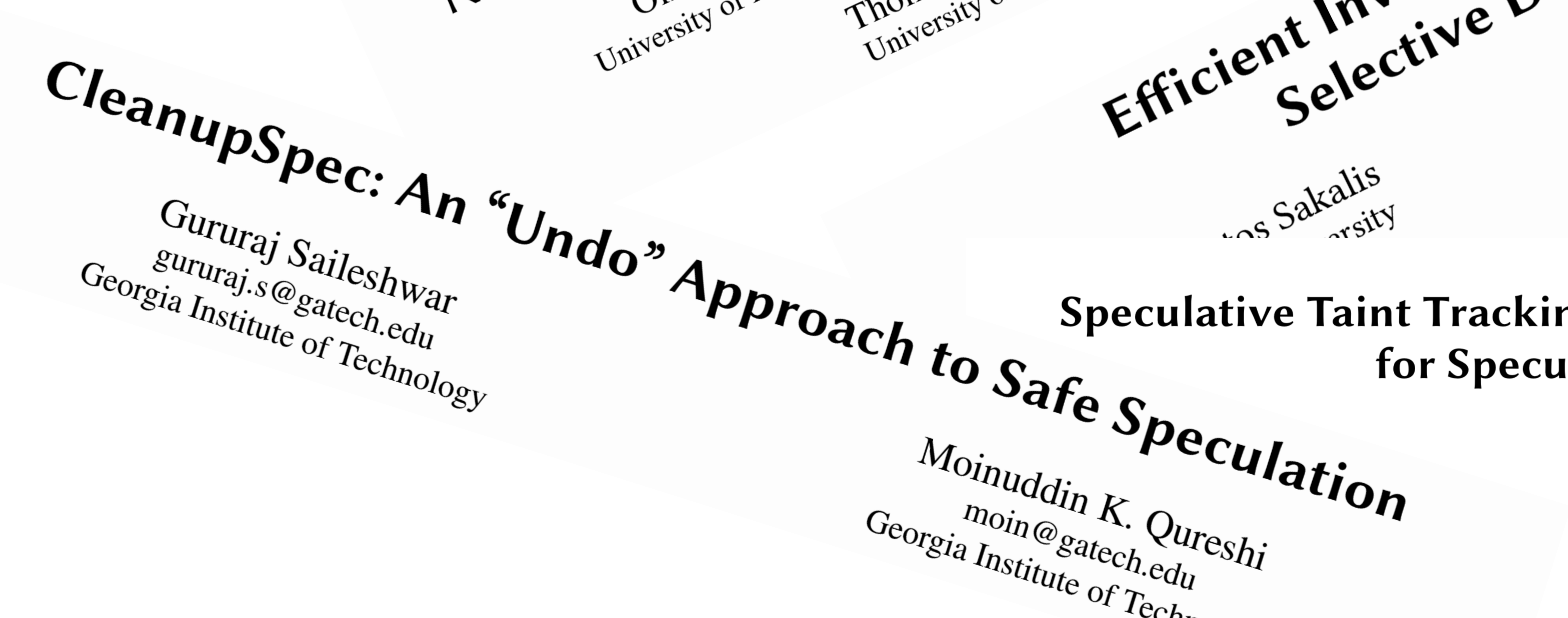
Stefanos Kaxiras
Uppsala University
Uppsala, Sweden
stefanos.kaxiras@it.uu.se

Alberto Ros
University of Murci
Murcia, Spain
aros@ditec.um

Magnus Själander
Norwegian University of Science and Technology
Trondheim, Norway
nus.sjalander@ntnu.no

tos Sakalis
ersity

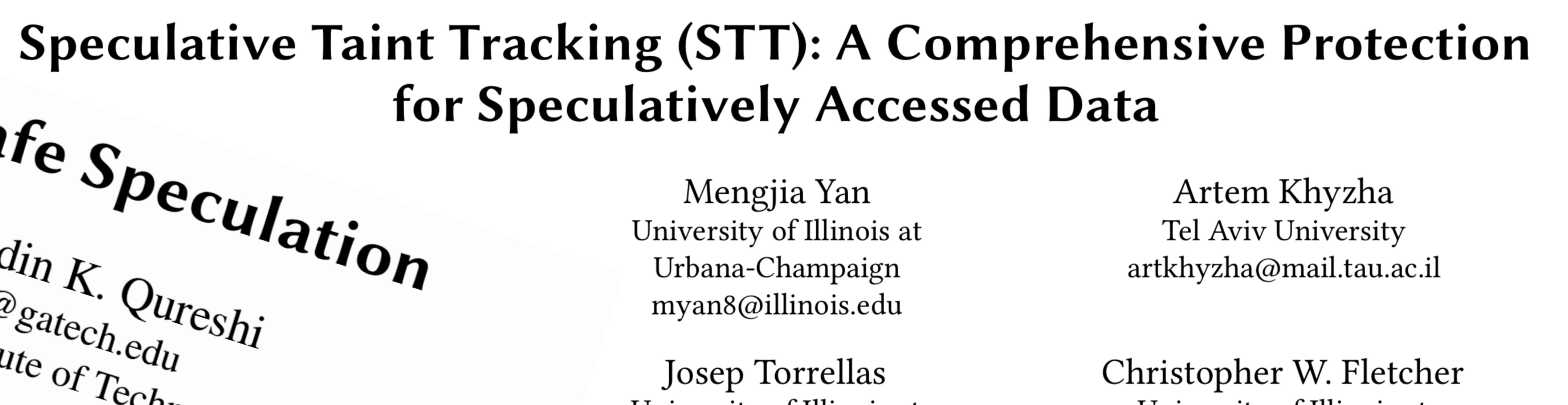
orean



CleanupSpec: An "Undo" Approach to Safe Speculation

Gururaj Saileshwar
gururaj.s@gatech.edu
Georgia Institute of Technology

Moinuddin K. Qureshi
moin@gatech.edu
Georgia Institute of Tech



Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data

Mengjia Yan
University of Illinois at Urbana-Champaign
myan8@illinois.edu

Artem Khyzha
Tel Aviv University
artkhyzha@mail.tau.ac.il

Josep Torrellas

Christopher W. Fletcher

Examples

```
if (x < A_size)
    y = A[x]
    z = B[y*512]
end
```

Delay loads until they can be retired [Sakalis et al., ISCA'19]

Delay loads until they cannot be squashed [Sakalis et al., ISCA'19]

Taint speculatively loaded data + delay tainted loads [STT and NDA, MICRO'19]

Examples

```
y = A[x]
if (x < A_size)
   z = B[y*512]
end
```

Delay loads until they can be retired

[Sakalis et al., ISCA'19]

Delay loads until they cannot be squashed

[Sakalis et al., ISCA'19]

Taint speculatively loaded data + delay tainted loads

[STT and NDA, MICRO'19]

What security properties do HW countermeasures enforce?

How can we program securely?

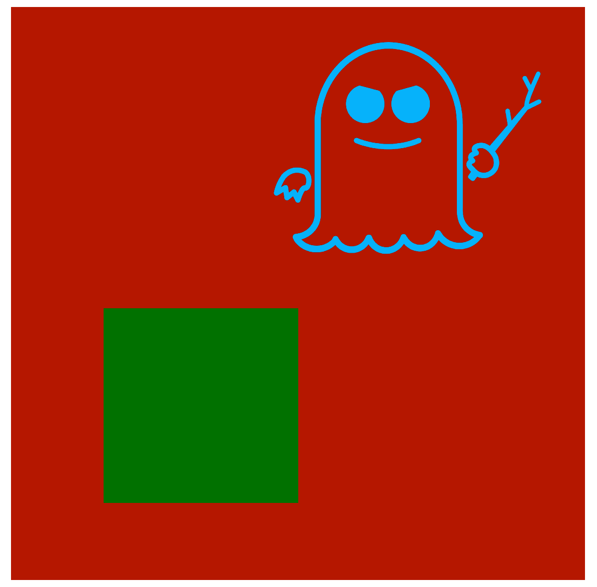
A Proof of Concept

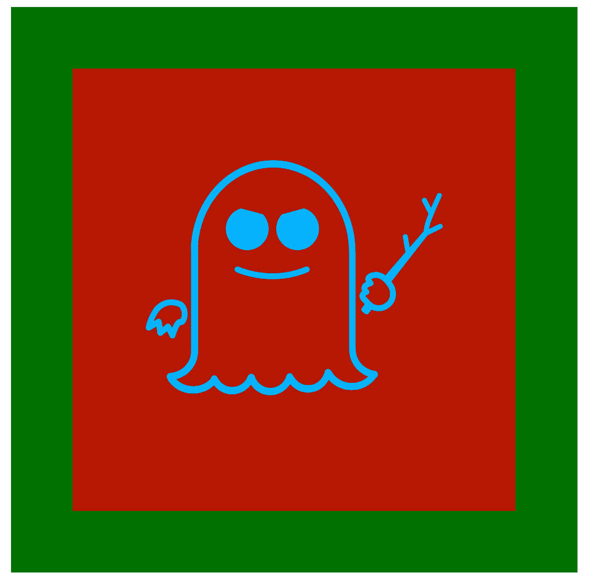
M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware-Software Contracts for Secure Speculation
S&P (Oakland) 2021

Hardware-Software Contracts

HW/SW Contracts for Secure Speculation

Secure Programming


Constant-time


Sandboxing

HW/SW Contracts for Secure Speculation

Desiderata: simple, precise, mechanism-independent, at "ISA level"

Hardware Countermeasures

No speculation

Load Delay

Taint Tracking

No countermeasures

HW/SW Contracts for Secure Speculation

Contracts specify which ***program executions*** a side-channel ***adversary cannot distinguish***

Contract

ISA + $\boxed{\text{observations}}$ $\longrightarrow$ **“What leaks”** about an execution

Contract traces: ≣ (p, σ)

Contracts

Attacker observes sequences of ***μarch states***

Contract

ISA + observations

Contract traces: ▤(p, σ)

Hardware

Formal model of processor

$\boxed{\text{Hardware traces: } ■(p, \sigma)}$

Contract satisfaction

Hardware ■ satisfies contract ▤ if for all programs p and arch. states σ, σ': if $\boxed{▤(p, \sigma) = ▤(p, \sigma')}$ then $\boxed{■(p, \sigma) = ■(p, \sigma')}$

Contracts for Secure Speculation

Contract =

$\boxed{\text{Execution Mode}} \cdot \boxed{\text{Observer Mode}}$

How are programs executed?

What is visible about the execution?

Contracts for Secure Speculation

Contract =

$\boxed{\text{Execution Mode}}$ · Observer Mode

seq — sequential execution

spec — mispredict branch instructions

Contracts for Secure Speculation

Contract =

Execution Mode · $\boxed{\text{Observer Mode}}$

pc — only program counter

ct — **pc** + addr. of loads and stores

arch — **ct** + loaded values

A Lattice of Contracts

$$\begin{tikzcd}
 & & \text{seq-ct} \arrow[r] & \top \\
 & \text{seq-arch} \arrow[ru] & \text{seq-ct+spec-pc} \arrow[u] & \\
 & & \text{spec-ct} \arrow[u] & \\
\bot \arrow[r] & \text{spec-arch} \arrow[uu] \arrow[ru] & &
\end{tikzcd}$$

Leaks "nothing"

Leaks all data accessed non-speculatively

Leaks addresses of non-spec. loads/stores/ instruction fetches

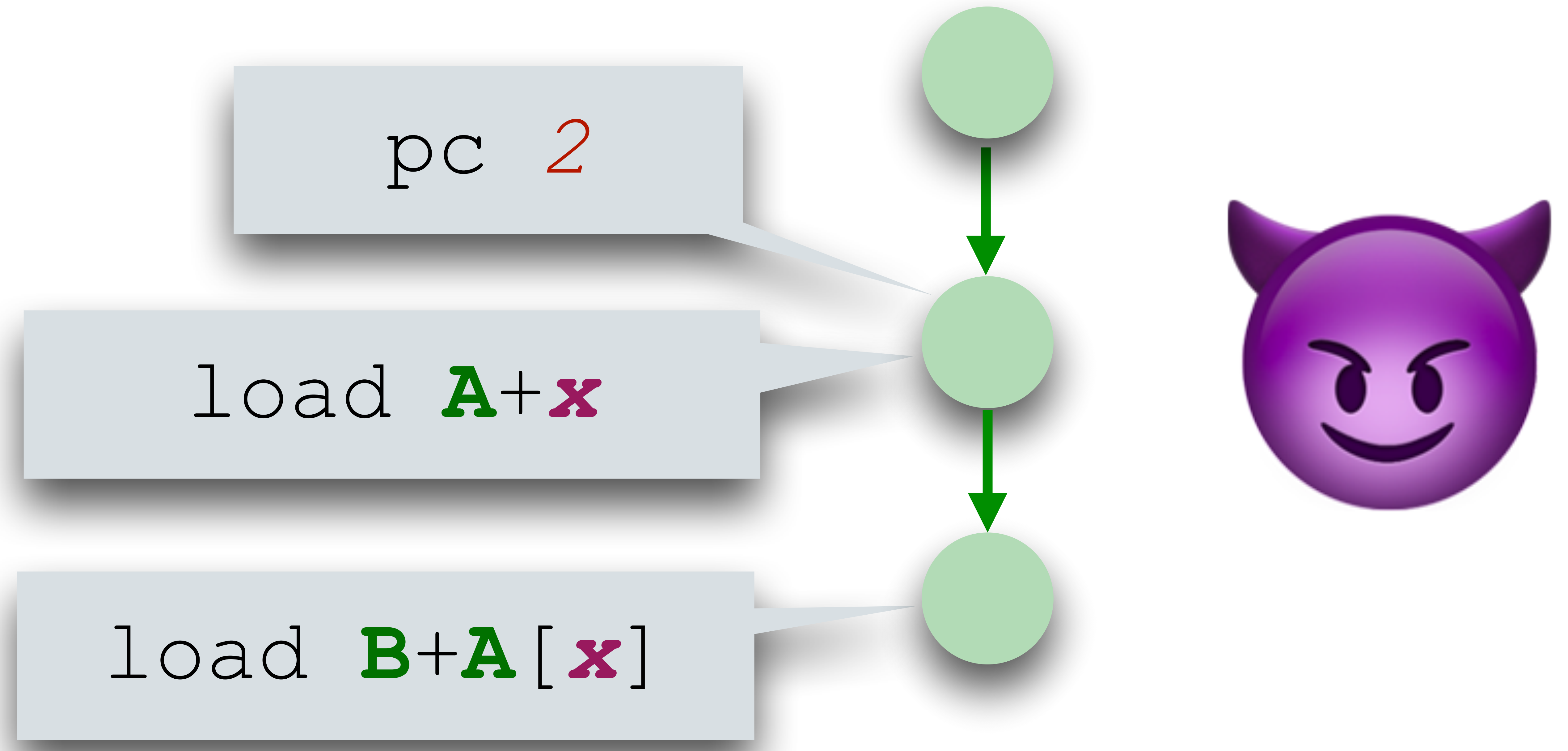
Leaks addresses of all loads/stores/instruction fetches

Leaks "everything"

Example: seq-ct

Assume *x* < *A_size*

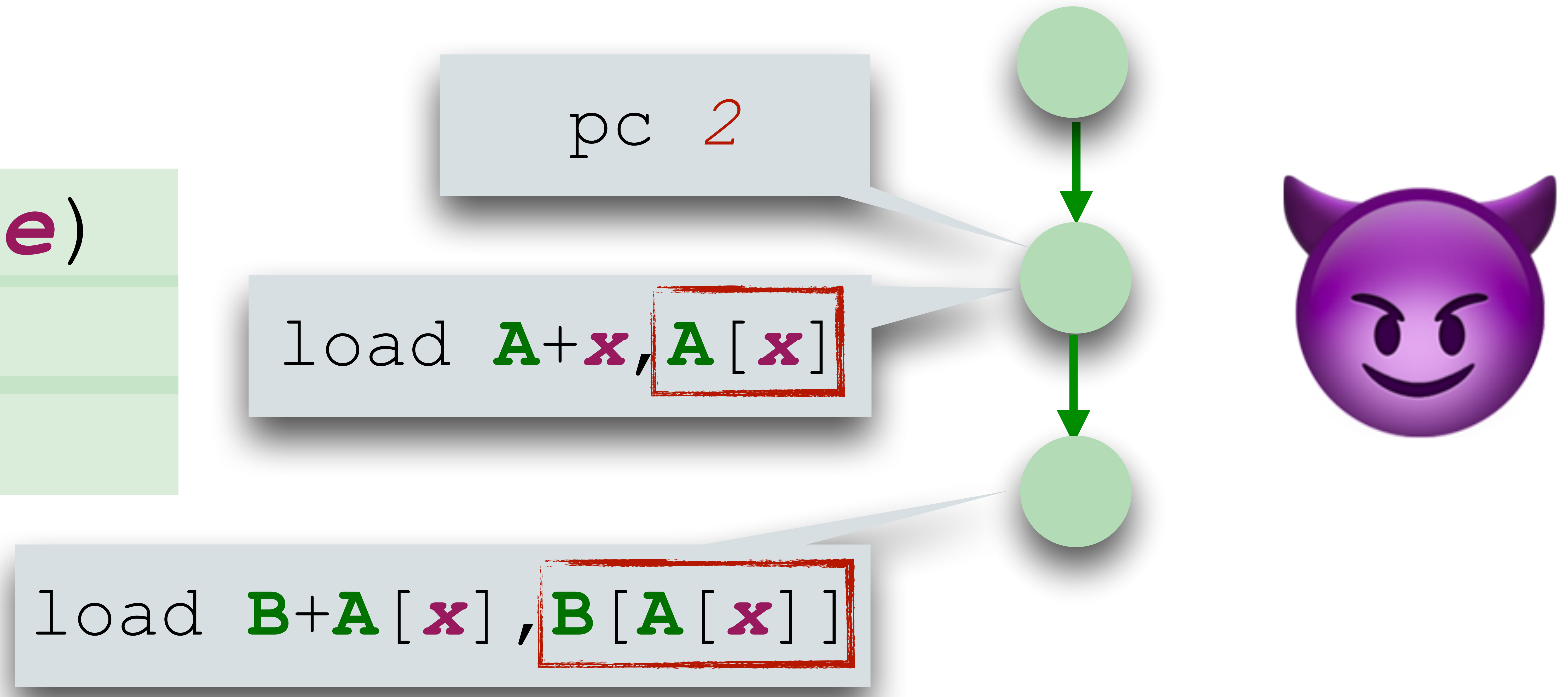
```
if (x < A_size)
  y = A[x]
  z = B[y]
end
```



Example: seq-arch

Assume `x < A_size`

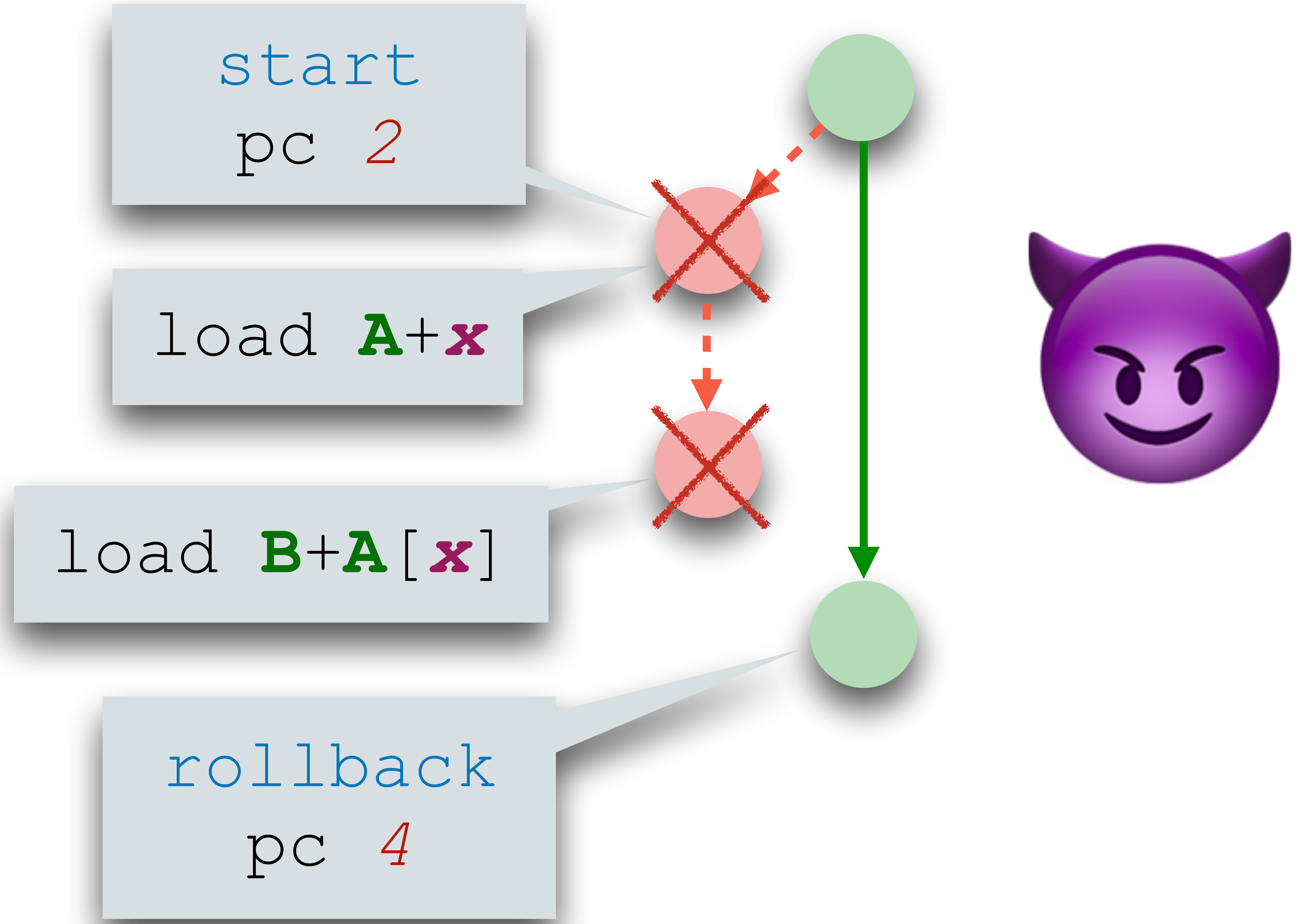
```
if (x < A_size)
   y = A[x]
   z = B[y]
end
```



Example: spec-ct

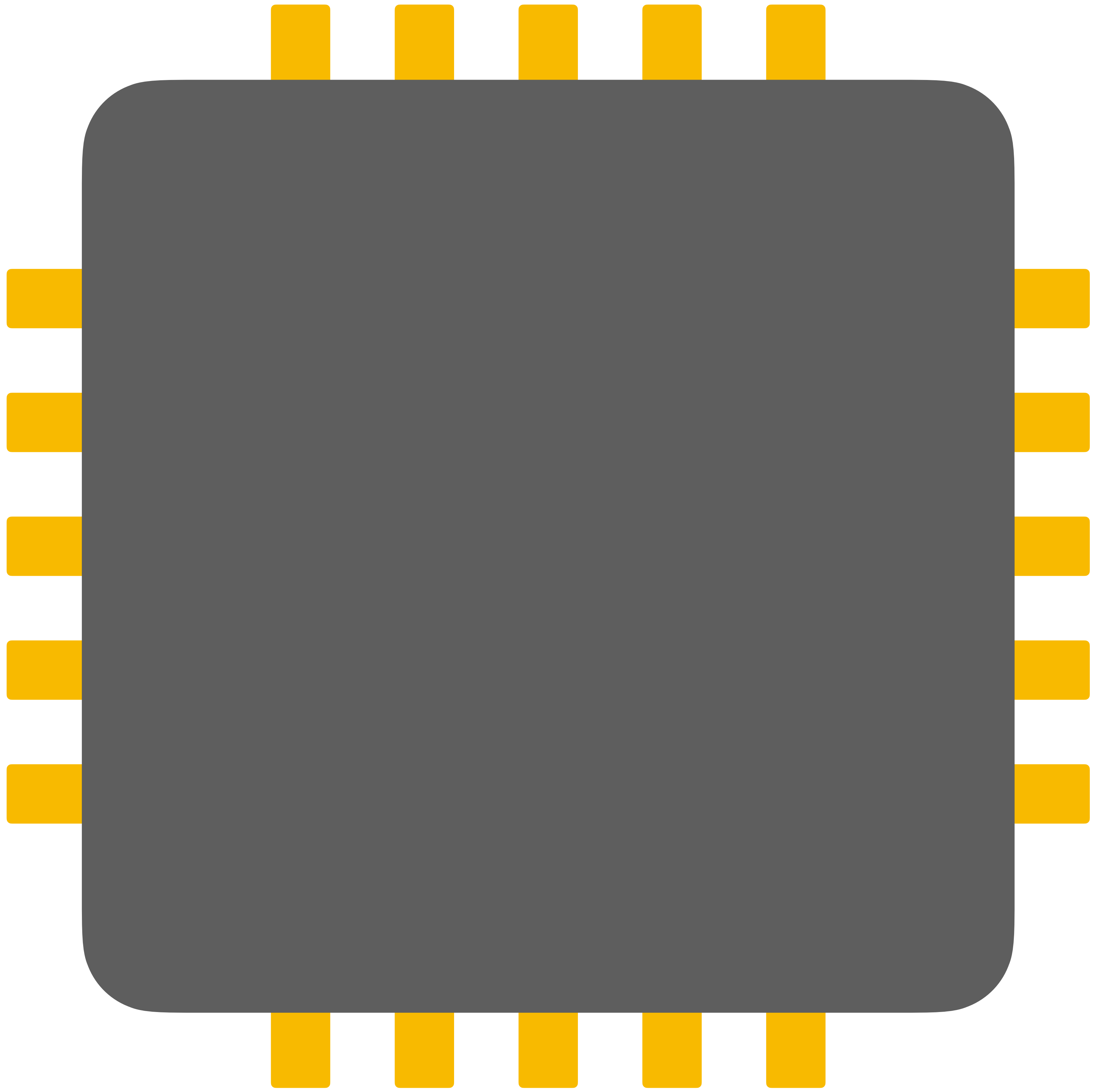
Assume *x* > *A_size*

```
if  (x < A_size)
    y = A[x]
    z = B[y]
end
```



Hardware Countermeasures

A Simple Processor



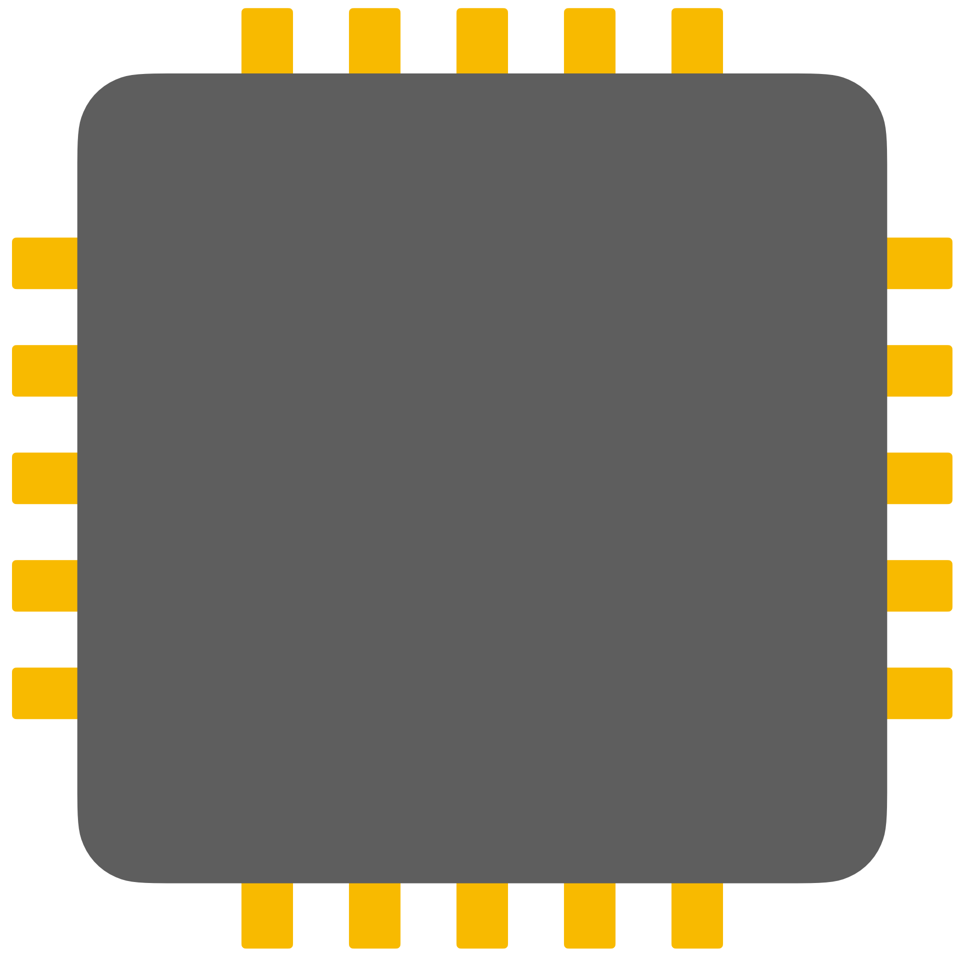
3-stage pipeline
(fetch, execute, retire)

Speculative and *out-of-order* execution

Parametric in *branch predictor* and *memory hierarchy*

Different *schedulers* for different countermeasures

Disabling Speculative Execution



Instructions are executed *sequentially*:
(fetch, execute, retire)*

🥳 No speculative leaks 🥳

Satisfies **seq-ct**

Eager Load Delay *[Sakalis et al., ISCA’19]*

Security guarantees?

Eager Load Delay *[Sakalis et al., ISCA’19]*

```
if (x < A_size)
    z = A[x]
    y = B[z]
```

A[x] and B[z] delayed until x < A_size is resolved

🥳 No speculative leaks 🥳

Eager Load Delay *[Sakalis et al., ISCA’19]*

```
z = A[x]
if (x < A_size)
    y = B[z]
```

B[z] delayed until x < A_size is resolved

🥳 No speculative leaks 🥳

Eager Load Delay *[Sakalis et al., ISCA'19]*

```
z = A[x]
if (x < A_size)
  if (z==0)
    skip
```

`if (z==0)` is *not* delayed

Program speculatively leaks A[x] 😞

Observation: Can only leak data **accessed non-speculatively**

➡

Satisfies **seq-arch**

Satisfies **seq-ct+spec-pc**

Taint Tracking *[Yu et al. 2019, Weisse et al. 2019]*

Taint speculatively loaded data

Security guarantees?

Delay tainted operations

Taint Tracking *[Yu et al. 2019, Weisse et al. 2019]*

```
if (x < A_size)
    z = A[x]
    y = B[z]
```

A[x] tainted as *unsafe*

B[z] *delayed* until

A[x] is safe

🥳 No speculative leaks 🥳

Taint Tracking *[Yu et al. 2019, Weisse et al. 2019]*

```
z = A[x]
if (x < A_size)
    y = B[z]
```

A[x] tagged as *safe*

B[z] *not delayed*

Program speculatively leaks A[x] 😞

Also satisfies **seq-arch**

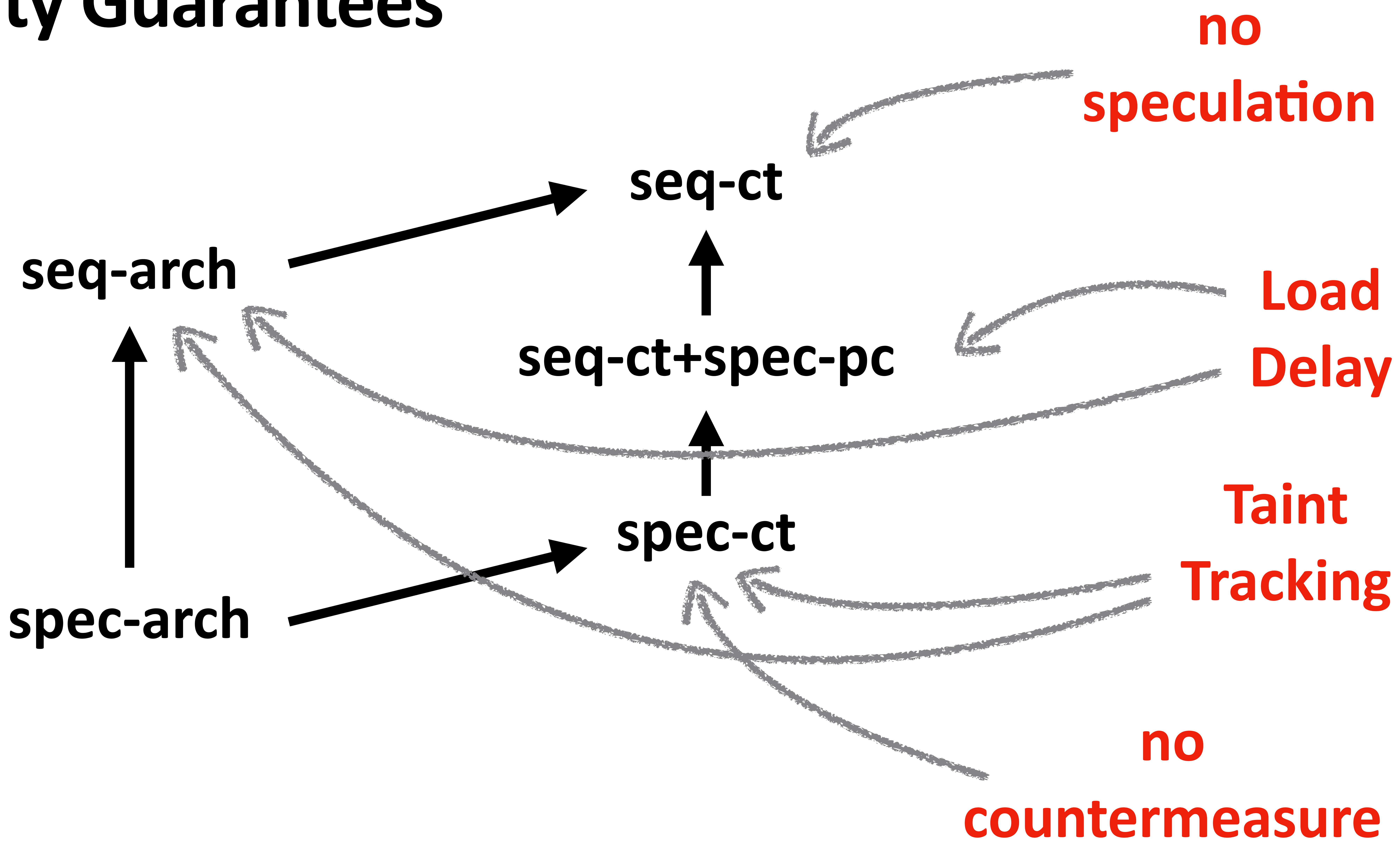
No Countermeasure *[The World until 2018]*

```
if (x < A_size)
    z = A[x]
    y = B[z]
```

Leaks addressed of speculative and non-speculative accesses

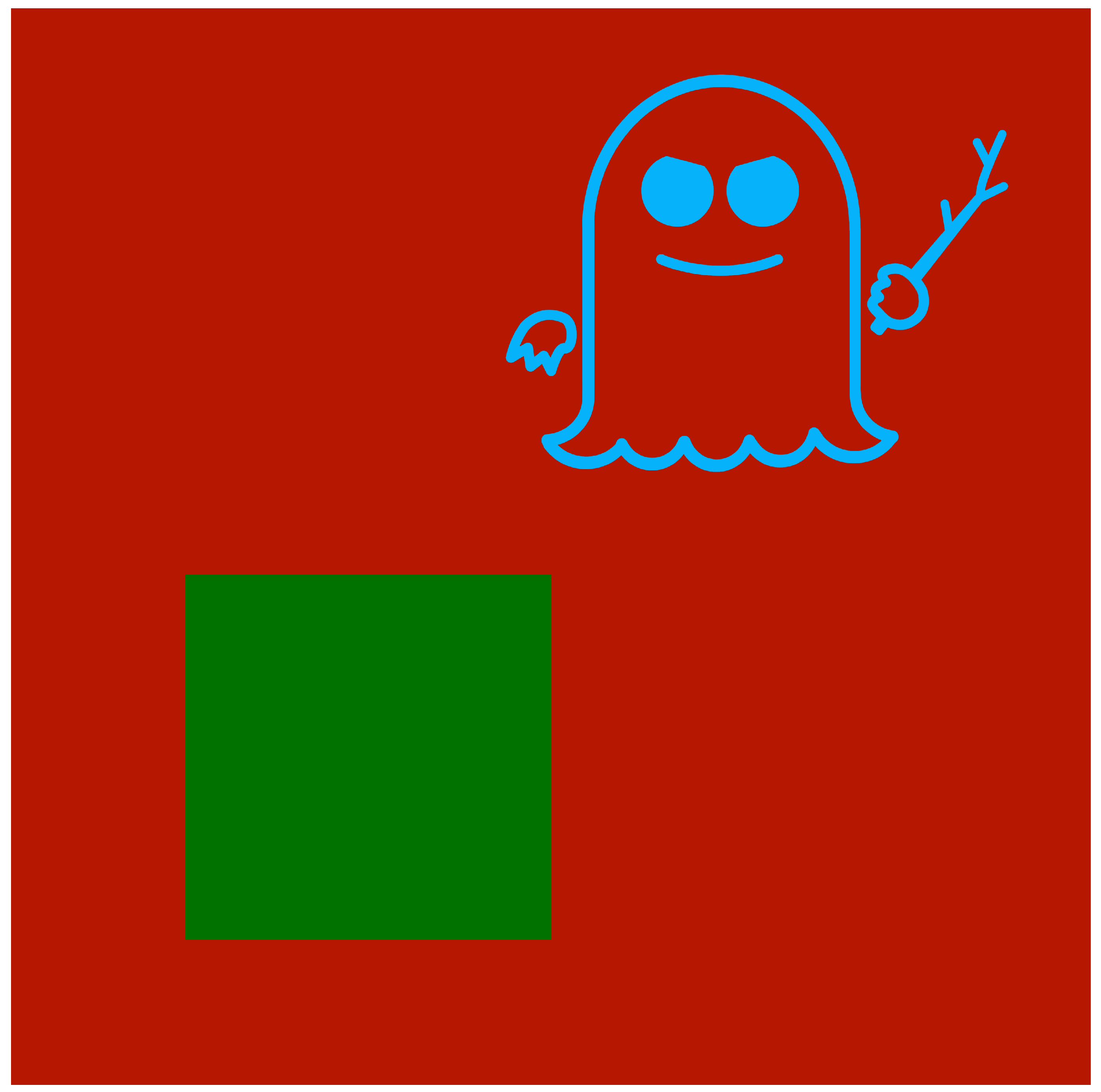
→ Satisfies **spec-ct**

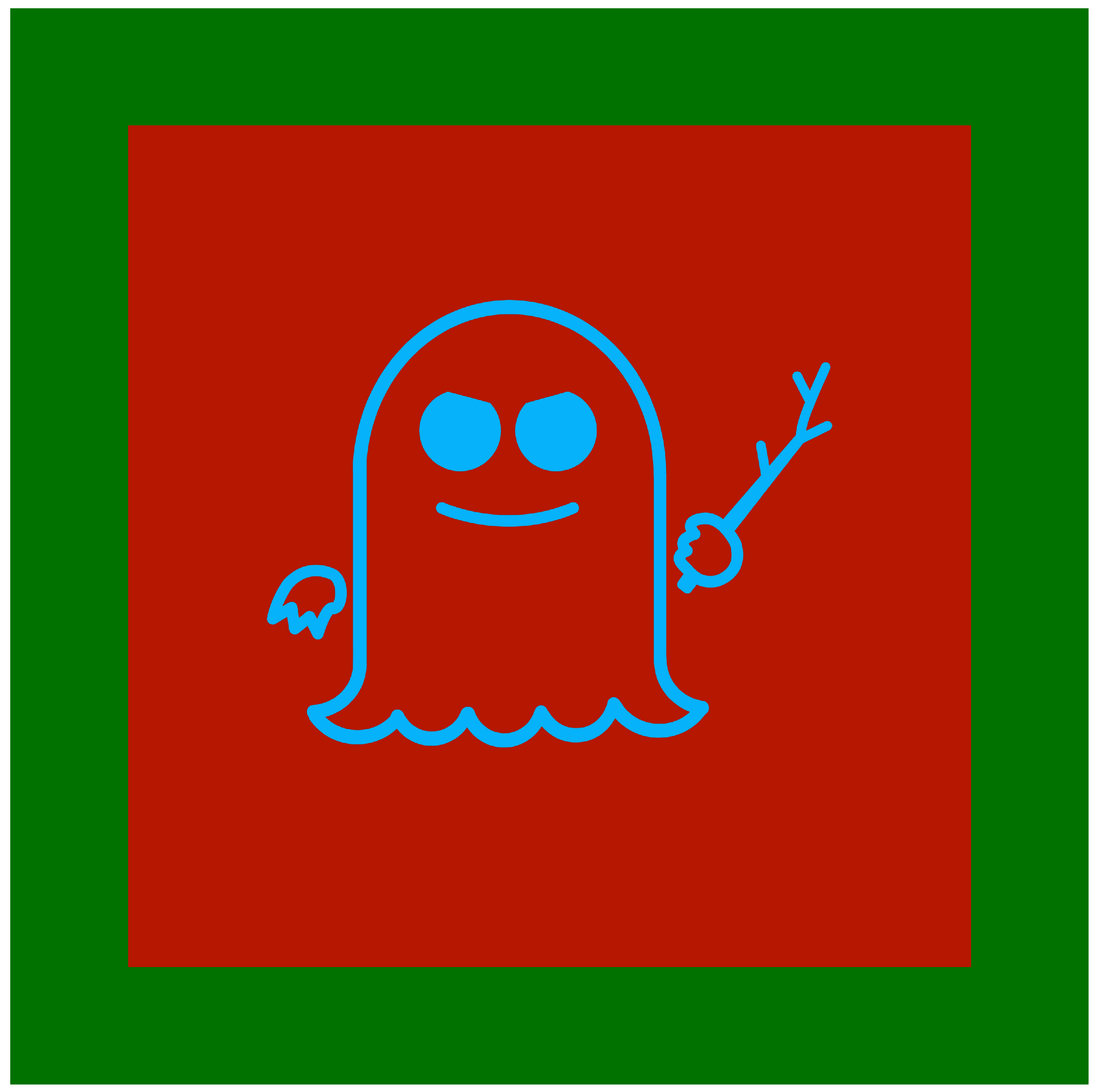
Security Guarantees



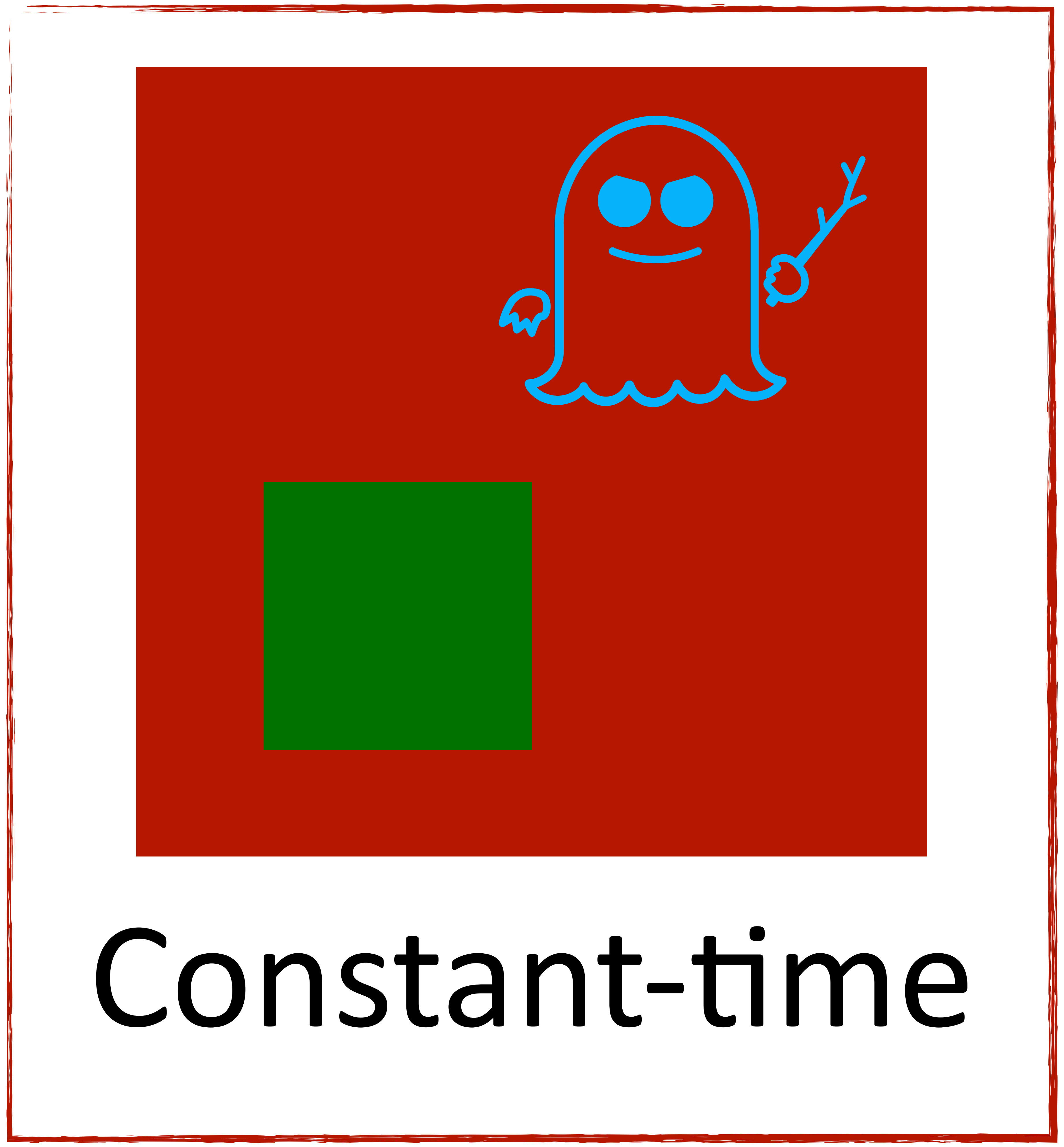
Secure Programming

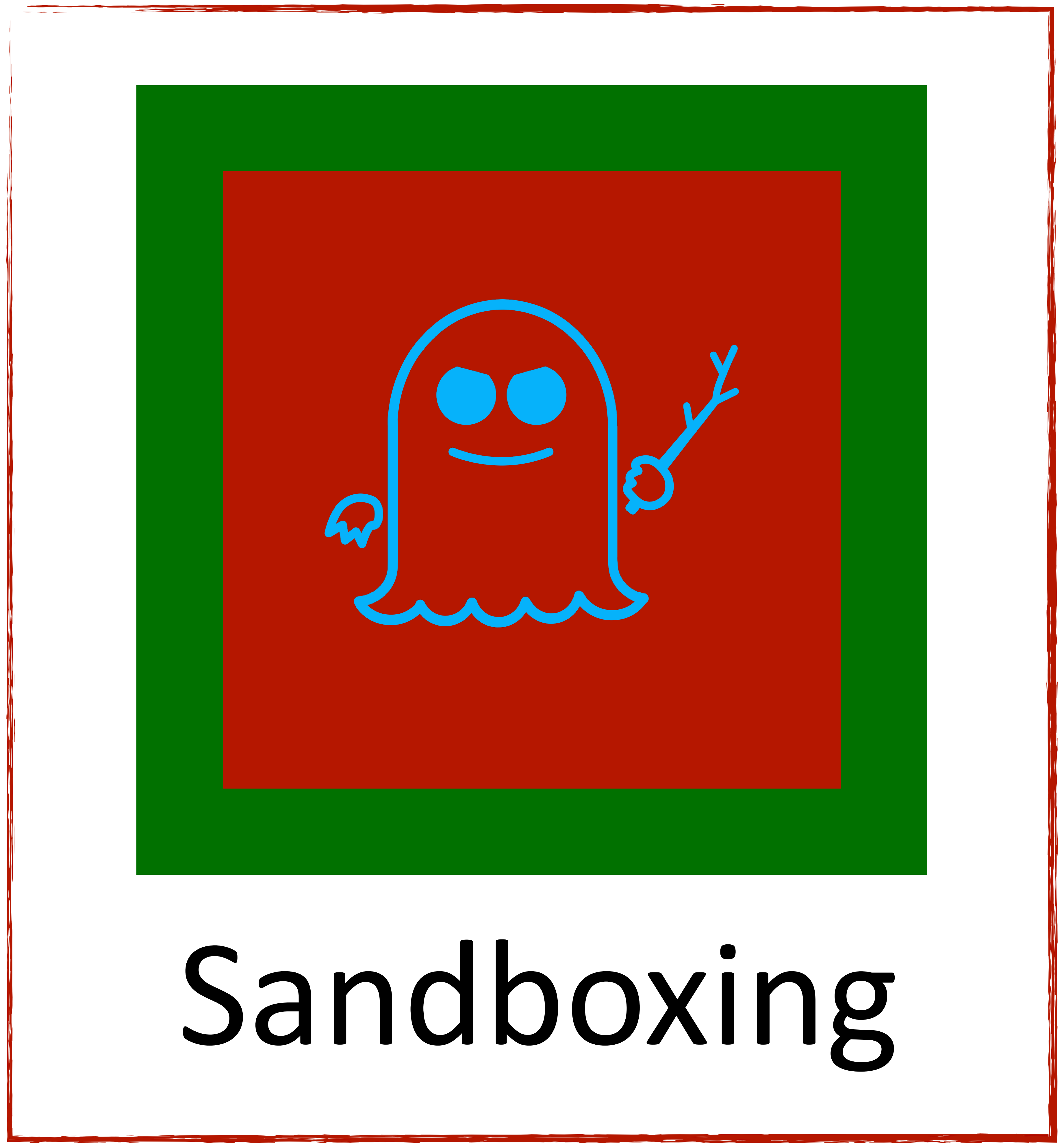
Two Flavors of Secure Programming


Constant-time


Sandboxing

Two Flavors of Secure Programming





Secure Programming: Foundations

Specify secret data

Program p is ***non-interferent*** wrt contract $[\![\cdot]\!]$ and $\boxed{\text{policy } \pi}$
if for all arch. states σ, σ': if $\boxed{\sigma \approx_\pi \sigma'}$ then $\boxed{[\![p]\!](\sigma) = [\![p]\!](\sigma')}$

Theorem

If p is ***non-interferent*** wrt contract $[\![\cdot]\!]$ and policy π,
and hardware $\{\!|\cdot|\!\}$ satisfies $[\![\cdot]\!]$, then
p is ***non-interferent*** wrt hardware $\{\!|\cdot|\!\}$ and policy π.

Sandboxing

Programs **never access** secret memory locations (out-of-sandbox)

$\boxed{\textit{Traditional SB} \text{ wrt policy } \pi} \equiv$ non-interference wrt **seq-arch** and π

General SB wrt π and $[\![\cdot]\!] \equiv$

Traditional SB wrt π + non-interference wrt π and $[\![\cdot]\!]$

Checking Sandboxing

	General sandboxing
seq-ct	Traditional sandboxing (= non-interference wrt **seq-arch**)
seq-arch	Traditional sandboxing
spec-ct	… + weak SNI

Constant-time Programming

Control flow and memory accesses do not depend on secrets

$\boxed{\textit{Traditional CT} \text{ wrt policy } \pi} \equiv$ non-interference wrt **seq-ct** and π

General CT wrt π and $[\![\cdot]\!]$ $\equiv$ non-interference wrt $[\![\cdot]\!]$ and π

Checking Constant-time Programming

	General constant-time
seq-ct	Traditional constant-time (= non-interference wrt **seq-ct**)
seq-arch	Non-interference wrt **seq-arch** ⚡
spec-ct	… + Spec. non-interference [Spectector, S&P'20]

No access to secrets!

Work in progress: Contracts meet “the real world”

Contracts for Real ISAs + Real CPUs

3-stage pipeline

Manual proof

Toy ISA (6 instr.)
+ observer modes

Microarchitecture $\models$ **Contract**

Register transfer level designs

Automatic proof

Real ISA
+ observer modes

Contracts for Real ISAs + Real CPUs

Microarchitecture $\models$ **Contract**

Register transfer level designs

Automatic proof

Real ISA + observer modes

Open-source RISC-V cores

SMT solvers + Invariant inference

Separate ISA from observer mode

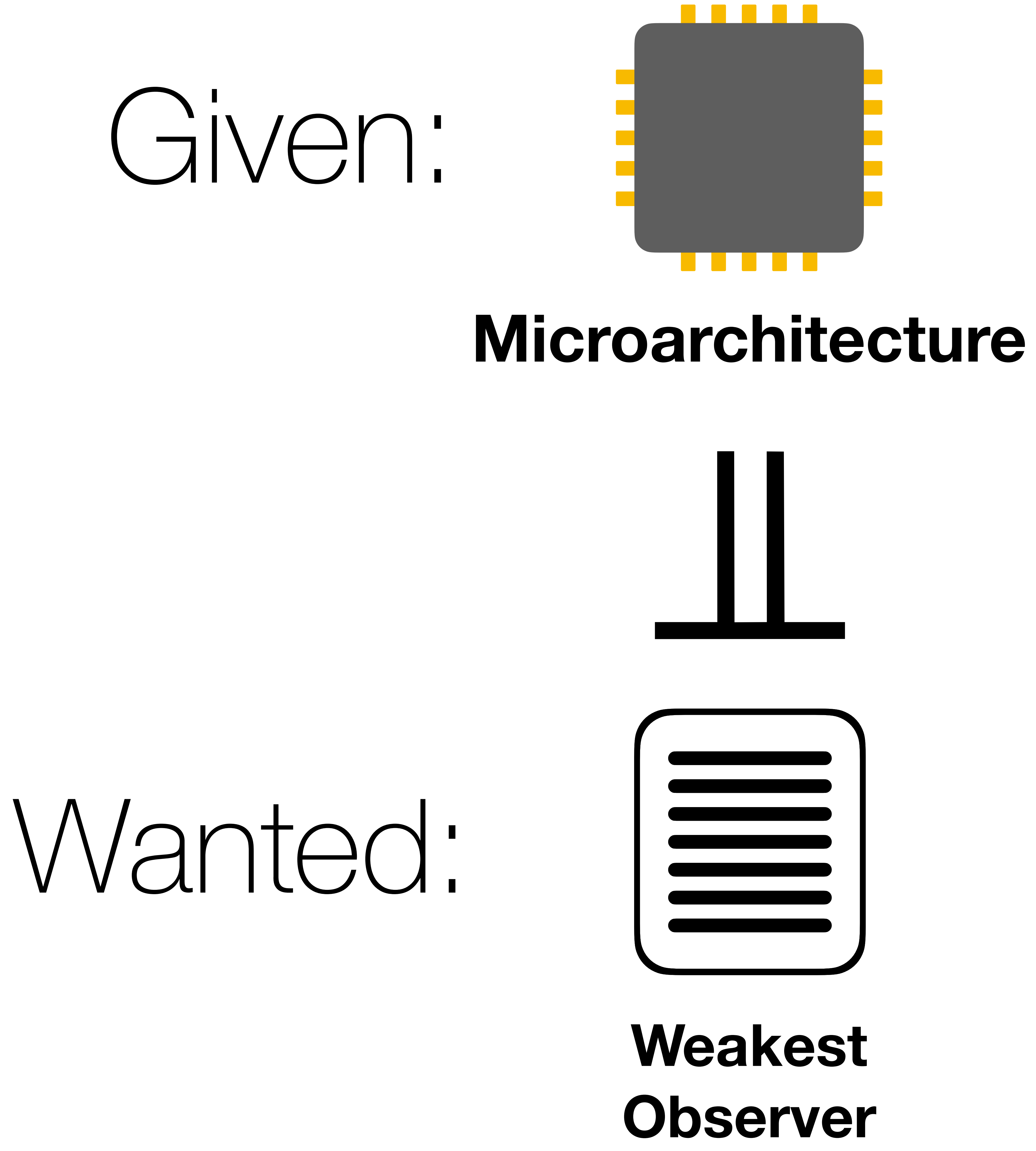
Separating Observer from ISA satisfaction

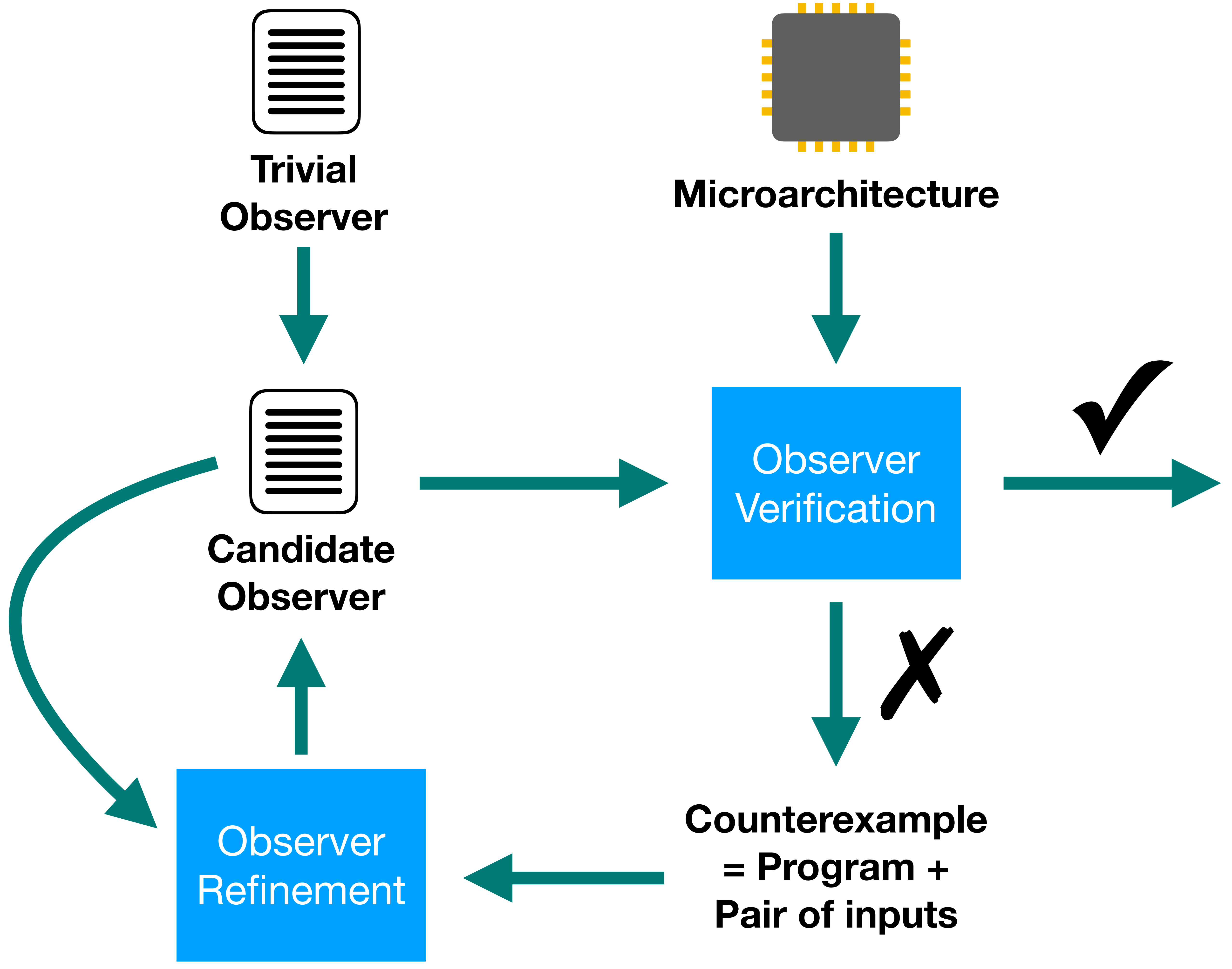
Contract satisfaction

Hardware ■ satisfies contract ▤ if for all programs p and arch. states σ, σ': if ▤(p, σ)=▤(p, σ') then ■(p, σ) =■(p, σ')

ISA satisfaction **+** ***Observer satisfaction***

Observer Inference





Conclusions

Need to rethink **hardware-software contracts** with security in mind

Find out more in our paper:

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware–Software Contracts for Secure Speculation
S&P (Oakland) 2021