Hardware-Software Contracts for Safe and Secure Systems

Jan Reineke @ UNIVERSITÄT DES SAARLANDES

Joint work with
Marco Guarnieri, Pepe Vila @ IMDEA Software, Madrid
Boris Köpf @ Microsoft Research, Cambridge, UK
Andreas Abel, Sebastian Hahn, Valentin Touzeau @ Saarland University

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

The Need for HW/SW Contracts

"Stone-age" Computing

Applications implemented data transformations:
- e.g. payroll processing

"Stone-age" Computing

Applications implemented data transformations:
  e.g. payroll processing

Hardware:
- isolated, on-site
- limited interaction with environment

IBM System 360/30

Author: ArnoldReinhold License: CC BY-SA 3.0

"Stone-age" Computing

Applications implemented data transformations:
  e.g. payroll processing

Hardware:
- isolated, on-site
- limited interaction with environment

IBM System 360/30

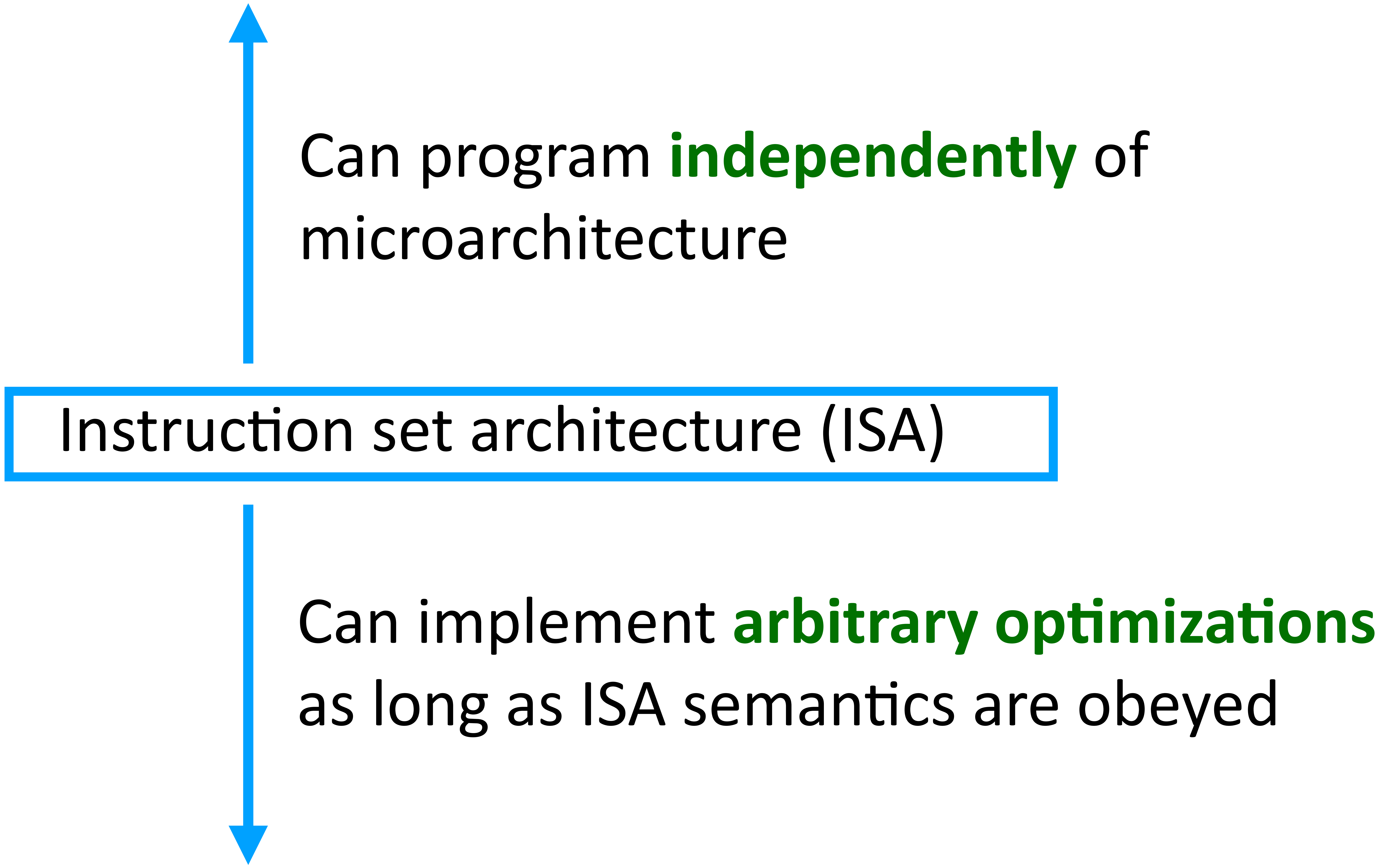
Author: ArnoldReinhold License: CC BY-SA 3.0

HW/SW Contract: Instruction Set Architecture

ISA Abstraction

$$\begin{tikzcd}
\text{High-level languages} \arrow[d, "\text{Compiler}"] \\
\boxed{\text{Instruction set architecture (ISA)}} \arrow[d, "\text{Implementation}"] \\
\text{Microarchitecture}
\end{tikzcd}$$

ISA Abstraction: Benefits



Can program **independently** of microarchitecture

Instruction set architecture (ISA)

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

"Modern" (?) Computing

Applications are:

- *Data-driven*: e.g. deep neural networks
- *Distributed*: e.g. locally + in the cloud
- *Open*: e.g. untrusted code in the browser
- *Real-time*: interacting with the physical environment

"Modern" (?) Computing

Applications are:

- *Data-driven*: e.g. deep neural networks
- *Distributed*: e.g. locally + in the cloud
- *Open*: e.g. untrusted code in the browser
- *Real-time*: interacting with the physical environment

What are the implications for HW/SW contracts?

Inadequacy of the ISA + current µArchitectures: Real-time Systems

Instruction set architecture (ISA) **Abstracts from time**

Inadequacy of the ISA + current µArchitectures: Real-time Systems

Instruction set architecture (ISA) **Abstracts from time**

↓

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

Inadequacy of the ISA + current µArchitectures: Real-time Systems

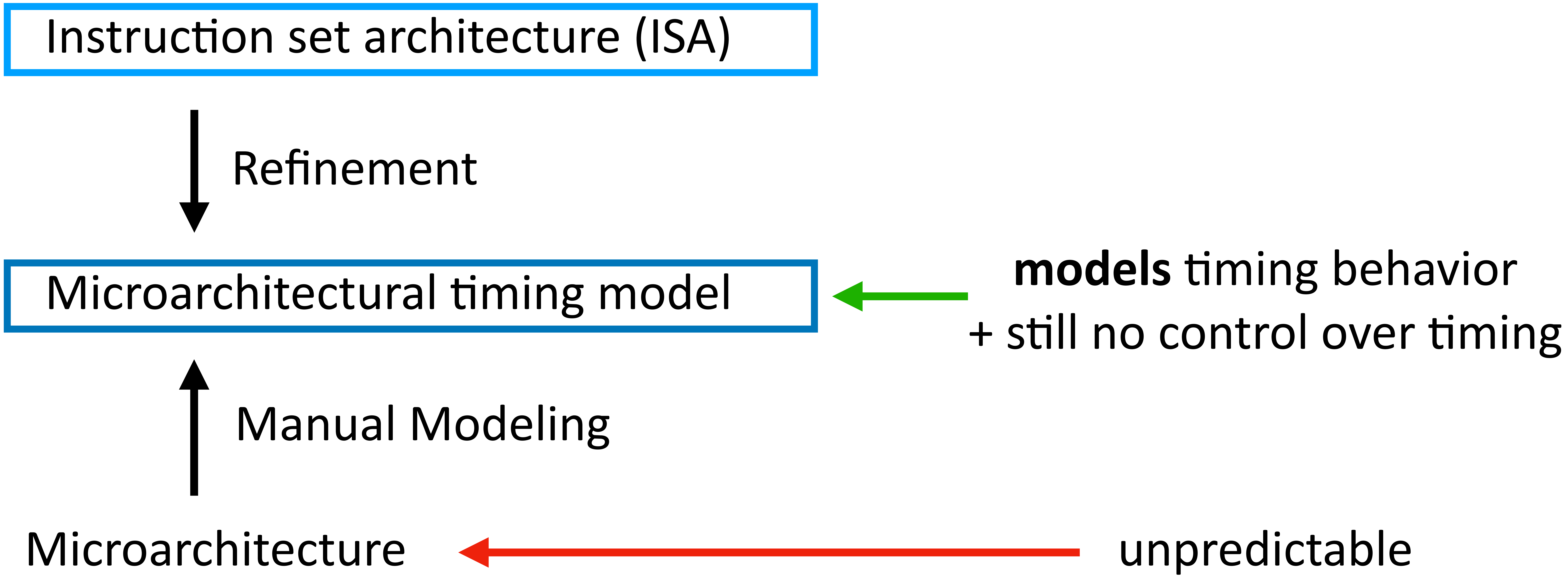
Programs do not have a **timed semantics**

Programs have **no control** over timing

Instruction set architecture (ISA) **Abstracts from time**

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

State-of-the-art: Handcrafted Microarchitectural Timing Models


Instruction set architecture (ISA)
Refinement
Microarchitectural timing model
models timing behavior
+ still no control over timing
Manual Modeling
Microarchitecture
unpredictable

State-of-the-art: Handcrafted Microarchitectural Timing Models

Instruction set architecture (ISA)

↓ Refinement

Microarchitectural timing model ← **models** timing behavior
+ still no control over timing

↑ Manual Modeling

Microarchitecture ← unpredictable

Models are
limited to particular microarchitectures
+ probably incorrect
+ yield expensive or imprecise analysis

Wanted: Timed HW/SW Contracts

Timed Instruction Set Architecture

Wanted: Timed HW/SW Contracts

Timed Instruction Set Architecture

Admit **wide range** of high-performance microarchitectural implementations

Wanted: Timed HW/SW Contracts

Programs have a **timed semantics** that is **efficiently predictable**
Programs have **control** over timing

Timed Instruction Set Architecture

Admit **wide range** of high-performance
microarchitectural implementations

Wanted: Timed HW/SW Contracts

Some answers:

D. Bui, E. Lee, I. Liu, H. Patel, and J. Reineke:
Temporal Isolation on Multiprocessing Architectures
DAC 2011

S. Hahn and J. Reineke:
Design and Analysis of SIC:
A Provably Timing-Predictable Pipelined Processor Core
RTSS 2018

Inadequacy of the ISA + current μArchitectures: Side-channel security

Instruction set architecture (ISA)

No guarantees about side channels

Inadequacy of the ISA + current µArchitectures: Side-channel security

Instruction set architecture (ISA)

No guarantees about side channels

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

SPECTRE

Inadequacy of the ISA + current μArchitectures: Side-channel security

Impossible to program securely on top of ISA
- cryptographic algorithms?
- sandboxing untrusted code?

Instruction set architecture (ISA)

No guarantees about side channels

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

A Way Forward: HW/SW Security Contracts

Hardware-Software Contract = ISA + X

Succinctly captures possible information leakage

A Way Forward: HW/SW Security Contracts

Hardware-Software Contract = ISA + X

Succinctly captures possible information leakage

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

A Way Forward: HW/SW Security Contracts

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

Hardware-Software Contract = ISA + X

Succinctly captures
possible information leakage

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

A Concrete Challenge: 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
```

Example: Spectre v1 Gadget

1. **x** is out of bounds

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

Example: Spectre v1 Gadget

1. **x** is out of bounds

2. Executed speculatively

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

Example: Spectre v1 Gadget

1. x is out of bounds

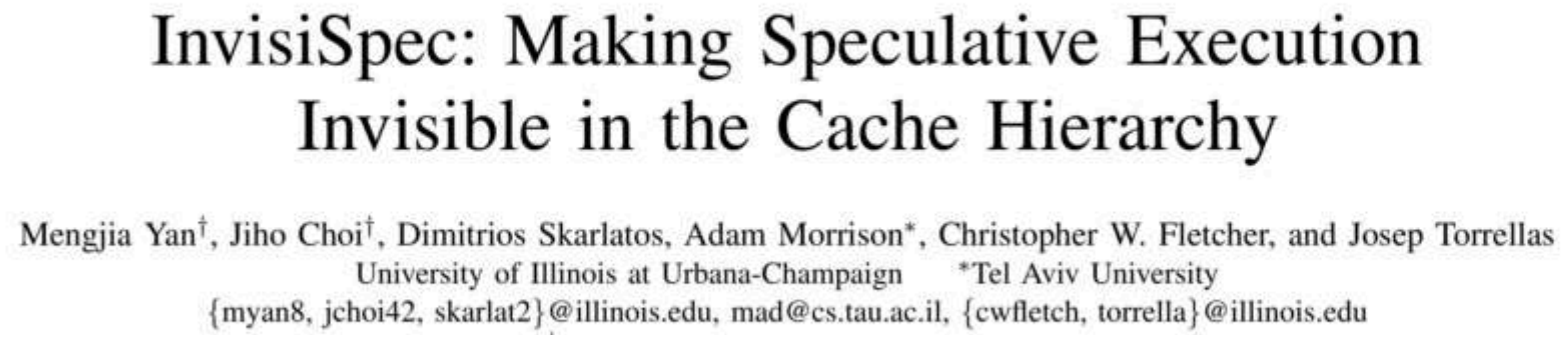
2. Executed speculatively

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

3. Leaks A[x] via data cache

Hardware Countermeasures

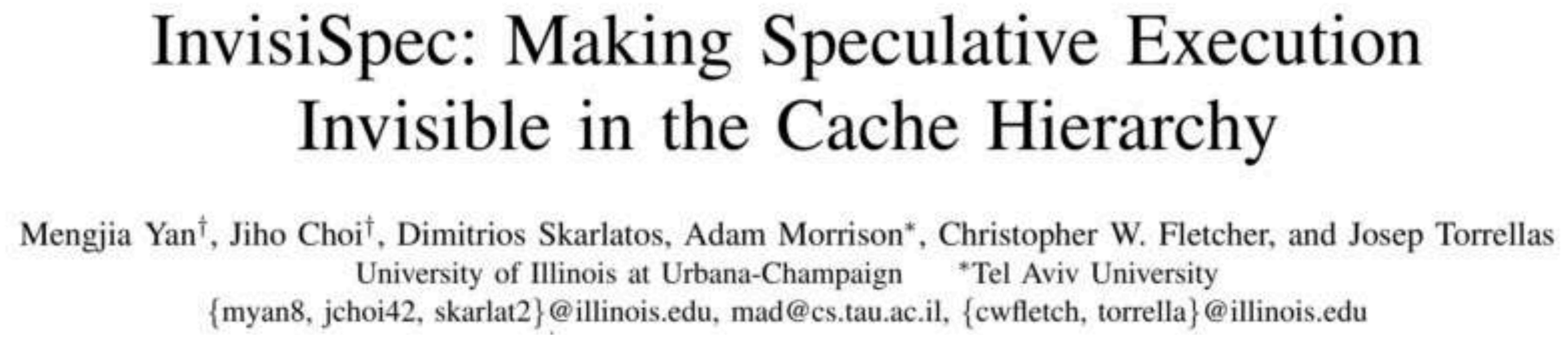
Hardware Countermeasures



InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

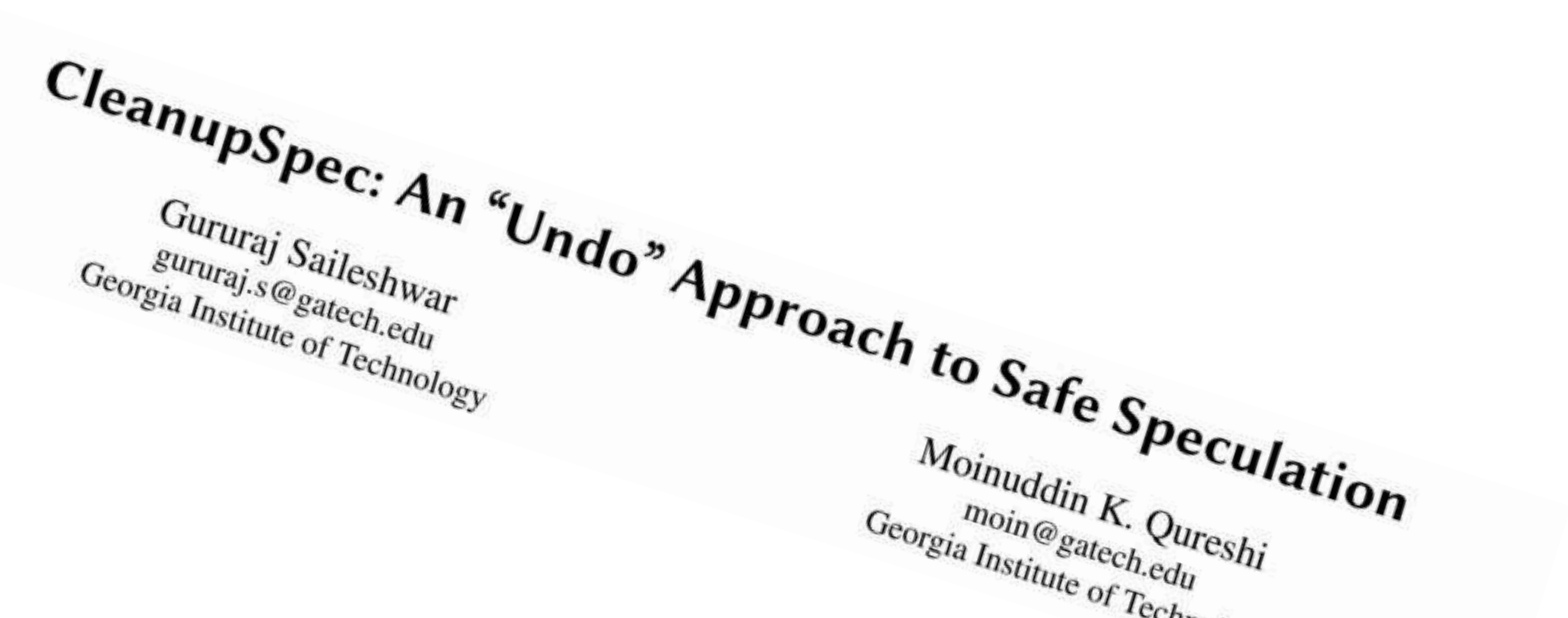
Mengjia Yan[†], Jiho Choi[†], Dimitrios Skarlatos, Adam Morrison[*], Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign [*]Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletch, torrella}@illinois.edu

Hardware Countermeasures



InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan[†], Jiho Choi[†], Dimitrios Skarlatos, Adam Morrison[*], Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign [*]Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletch, torrella}@illinois.edu

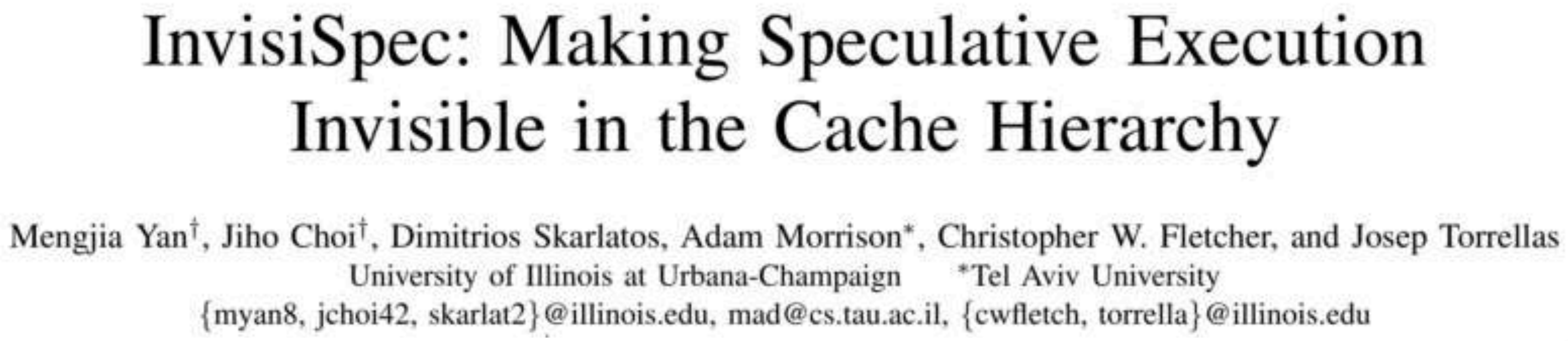


CleanupSpec: An “Undo” Approach to Safe Speculation

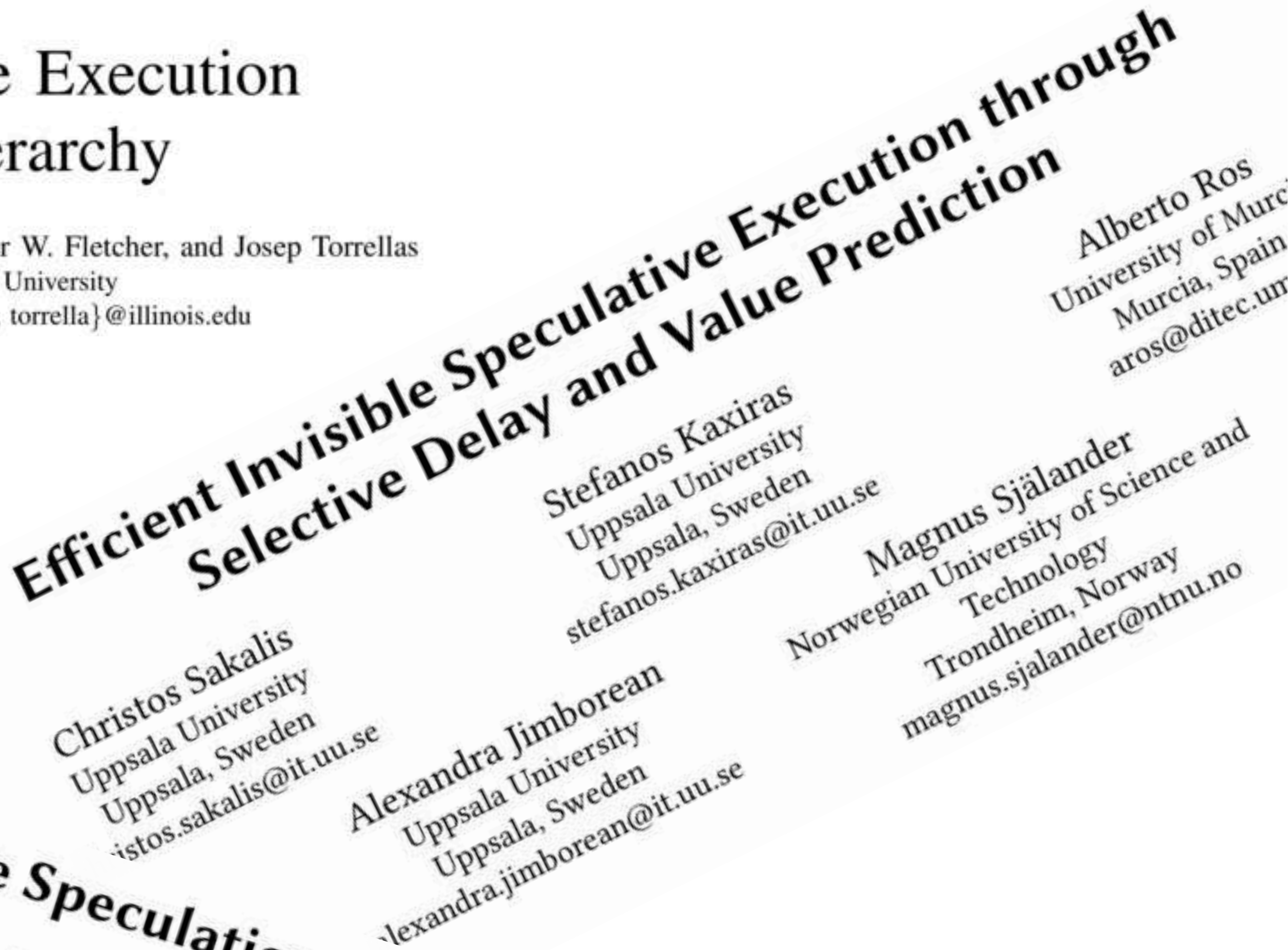
Gururaj Saileshwar
gururaj.s@gatech.edu
Georgia Institute of Technology

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

Hardware Countermeasures


InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan[†], Jiho Choi[†], Dimitrios Skarlatos, Adam Morrison[*], Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign [*]Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletch, torrella}@illinois.edu


Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

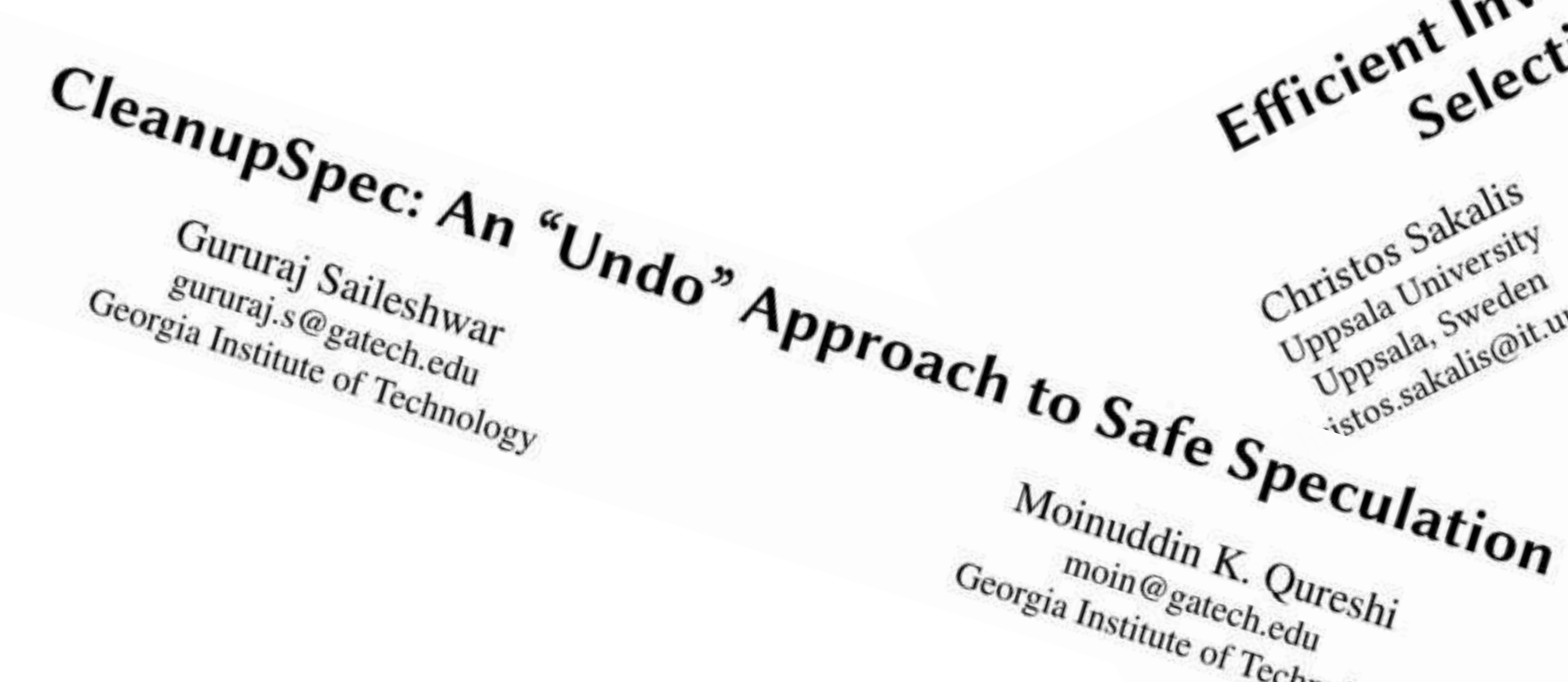
Christos Sakalis
Uppsala University
Uppsala, Sweden
istos.sakalis@it.uu.se

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

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

Alexandra Jimborean
Uppsala University
Uppsala, Sweden
lexandra.jimborean@it.uu.se

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


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

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

Christos Sakalis
Uppsala University
Uppsala, Sweden
istos.sakalis@it.uu.se

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

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

Alexandra Jimborean
Uppsala University
Uppsala, Sweden
lexandra.jimborean@it.uu.se

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

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

Hardware Countermeasures

InvisiSpec: Making Speculative E
Invisible in the Cache

Mengjia Yan[†], Jiho Choi[†], Dimitrios Skarlatos, Ad
University of Illinois a
{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

tos Sakalis
ersity

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

rean

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

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

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
```

Examples

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

Examples

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

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]

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
```

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]

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?

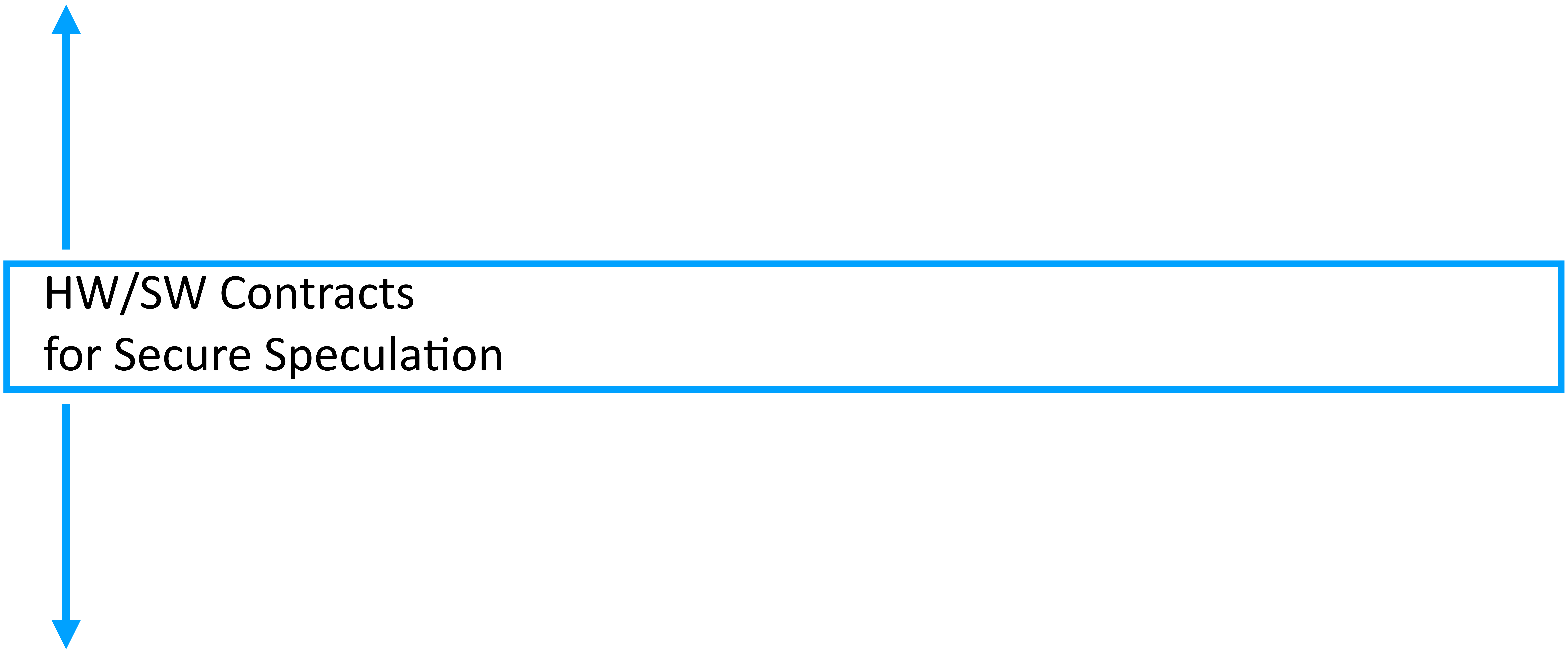
How can we program securely?

A Proof of Concept

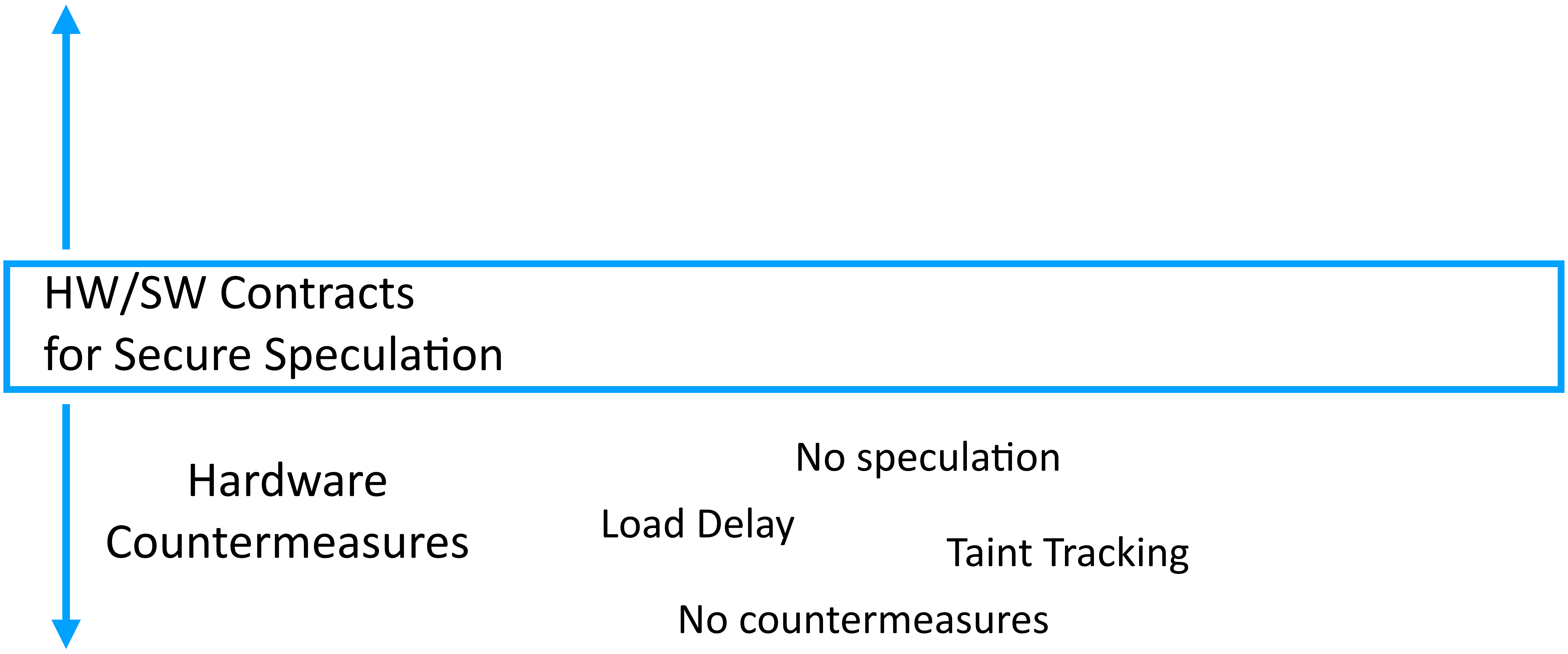
M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware-Software Contracts for Secure Speculation
S&P (Oakland) 2021

Hardware-Software Contracts

HW/SW Contracts for Secure Speculation

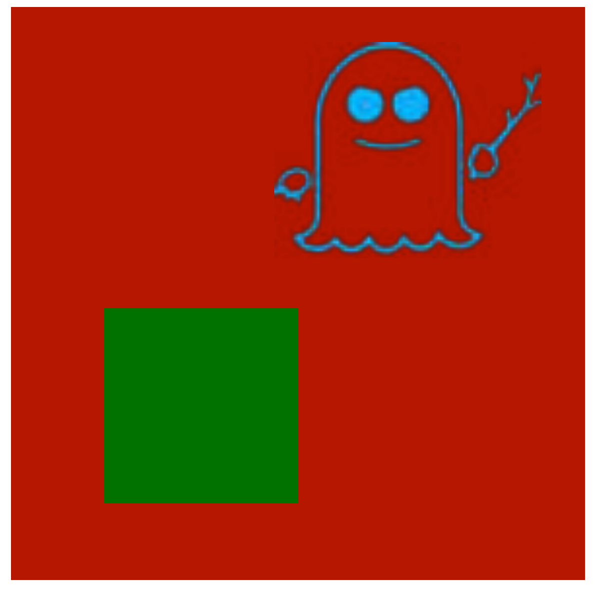


HW/SW Contracts for Secure Speculation

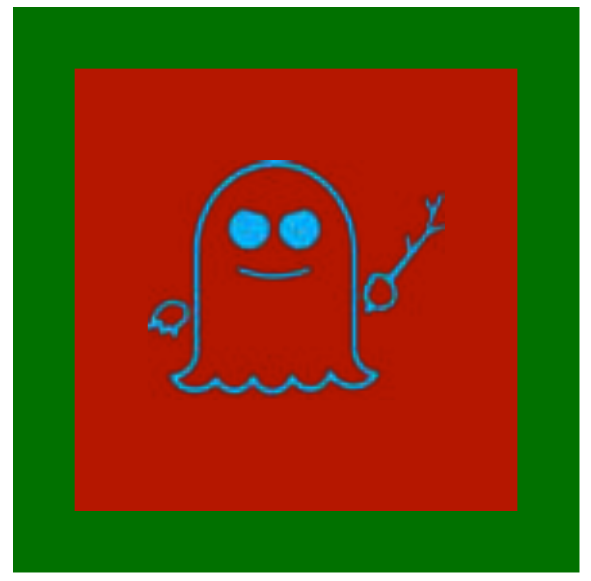


HW/SW Contracts for Secure Speculation

Secure Programming



Constant-time



Sandboxing

HW/SW Contracts for Secure Speculation

Hardware Countermeasures

No speculation

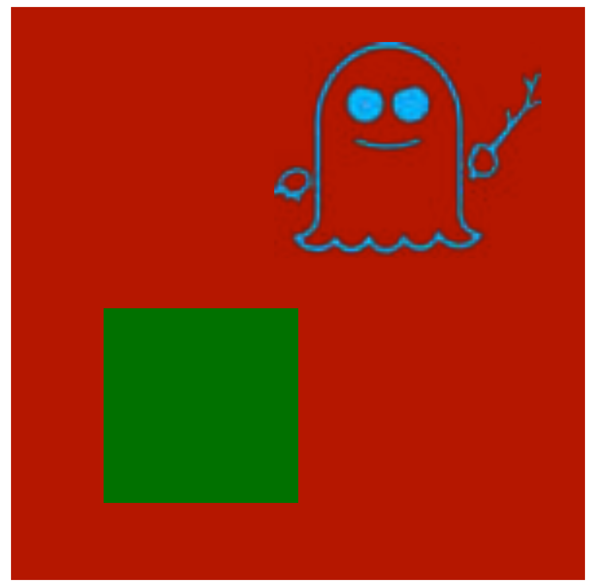
Load Delay

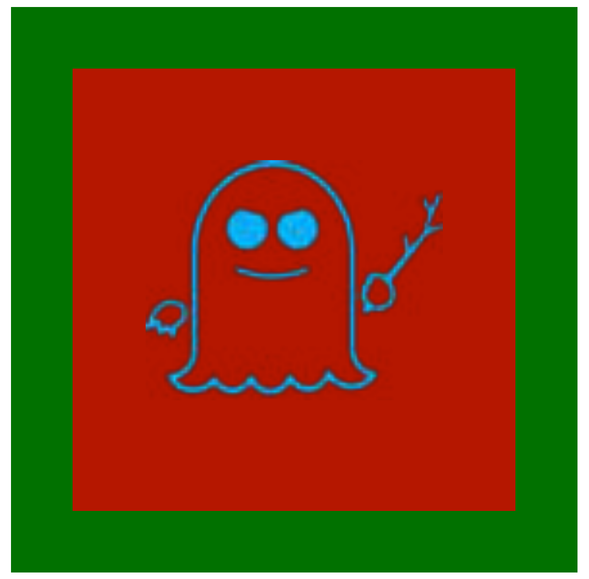
Taint Tracking

No countermeasures

HW/SW Contracts for Secure Speculation

Secure Programming


Constant-time


Sandboxing

HW/SW Contracts for Secure Speculation

Desiderata: simple mechanism-independent precise

Hardware Countermeasures

No speculation

Load Delay

Taint Tracking

No countermeasures

Ingredients of a Formalization

Ingredients of a Formalization

Instruction Set Architecture

Arch. states: σ

Arch. semantics: $\sigma \rightsquigarrow \sigma'$

Ingredients of a Formalization

Instruction Set Architecture

Arch. states: σ

Arch. semantics: $\sigma \rightsquigarrow \sigma'$

Microarchitecture

Hardware states: $\langle \sigma, \mu \rangle$

Hardware semantics: $\langle \sigma, \mu \rangle \Rightarrow \langle \sigma', \mu' \rangle$

Ingredients of a Formalization

Instruction Set Architecture

Arch. states: σ

Arch. semantics: $\sigma \rightsquigarrow \sigma'$

Microarchitecture

Hardware states: $\langle \sigma, \mu \rangle$

Hardware semantics: $\langle \sigma, \mu \rangle \Rightarrow \langle \sigma', \mu' \rangle$

Adversary model

µArch traces: $\{\!|\, p \,|\!\}(\sigma) = \mu_0\mu_1 \ldots \mu_n$

Contracts

Contracts

Contract

A deterministic, labelled semantics $\xrightarrow{\tau}$ for the ISA

Contracts

Observations expose security-relevant *μArch events*

Contract

A deterministic, labelled semantics $\boxed{\xrightarrow{\tau}}$ for the ISA

Contracts

Observations expose security-relevant *μArch events*

Contract

A deterministic, labelled semantics $\boxed{\xrightarrow{\tau}}$ for the ISA

Contract traces: $[\![p]\!](\sigma) = \tau_1\tau_2\ldots\tau_n$

Contracts

Observations expose security-relevant *μArch events*

Contract

A deterministic, labelled semantics $\boxed{\xrightarrow{\tau}}$ for the ISA

Contract traces: $[\![p]\!](\sigma) = \tau_1\tau_2\ldots\tau_n$

Contract satisfaction

Hardware $\{\!|\cdot|\!\}$ satisfies contract $[\![\cdot]\!]$ if for all programs p and arch. states σ, σ': if $[\![p]\!](\sigma) = [\![p]\!](\sigma')$ then $\{\!|p|\!\}(\sigma) = \{\!|p|\!\}(\sigma')$

Contracts for Secure Speculation

Contracts for Secure Speculation

Contract =

Execution Mode · Observer Mode

Contracts for Secure Speculation

Contract =

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

How are programs executed?

Contracts for Secure Speculation

Contract =

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

How are programs executed?

What is visible about the execution?

Contracts for Secure Speculation

Contract =

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

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 · $\boxed{\text{Observer Mode}}$

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

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

A Lattice of Contracts

$$\begin{tikzcd}
 & & \textbf{seq-ct} \arrow[r] & \top \\
 & \textbf{seq-arch} \arrow[ur] & & \\
 & & \textbf{seq-ct+spec-pc} \arrow[uu] & \\
 & & \textbf{spec-ct} \arrow[u] & \\
\bot \arrow[r] & \textbf{spec-arch} \arrow[uuu] \arrow[ur] & & \\
\text{Leaks “everything”} \arrow[u, bend left] & & &
\end{tikzcd}$$

A Lattice of Contracts

$$\begin{tikzcd}
 & & & \text{Leaks “nothing”} \arrow[d] \\
 & & \textbf{seq-ct} \arrow[r] & \top \\
 & \textbf{seq-arch} \arrow[ur] & & \\
 & & \textbf{seq-ct+spec-pc} \arrow[uu] & \\
 & & \textbf{spec-ct} \arrow[u] & \\
\bot \arrow[r] & \textbf{spec-arch} \arrow[uuu] \arrow[ur] & & \\
\text{Leaks “everything”} \arrow[u] & & &
\end{tikzcd}$$

A Lattice of Contracts

Leaks “nothing”

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

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

Leaks “everything”

A Lattice of Contracts

Leaks "nothing"

Leaks all data accessed non-speculatively

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

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

Leaks "everything"

A Lattice of Contracts

Leaks "nothing"

Leaks all data accessed non-speculatively

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

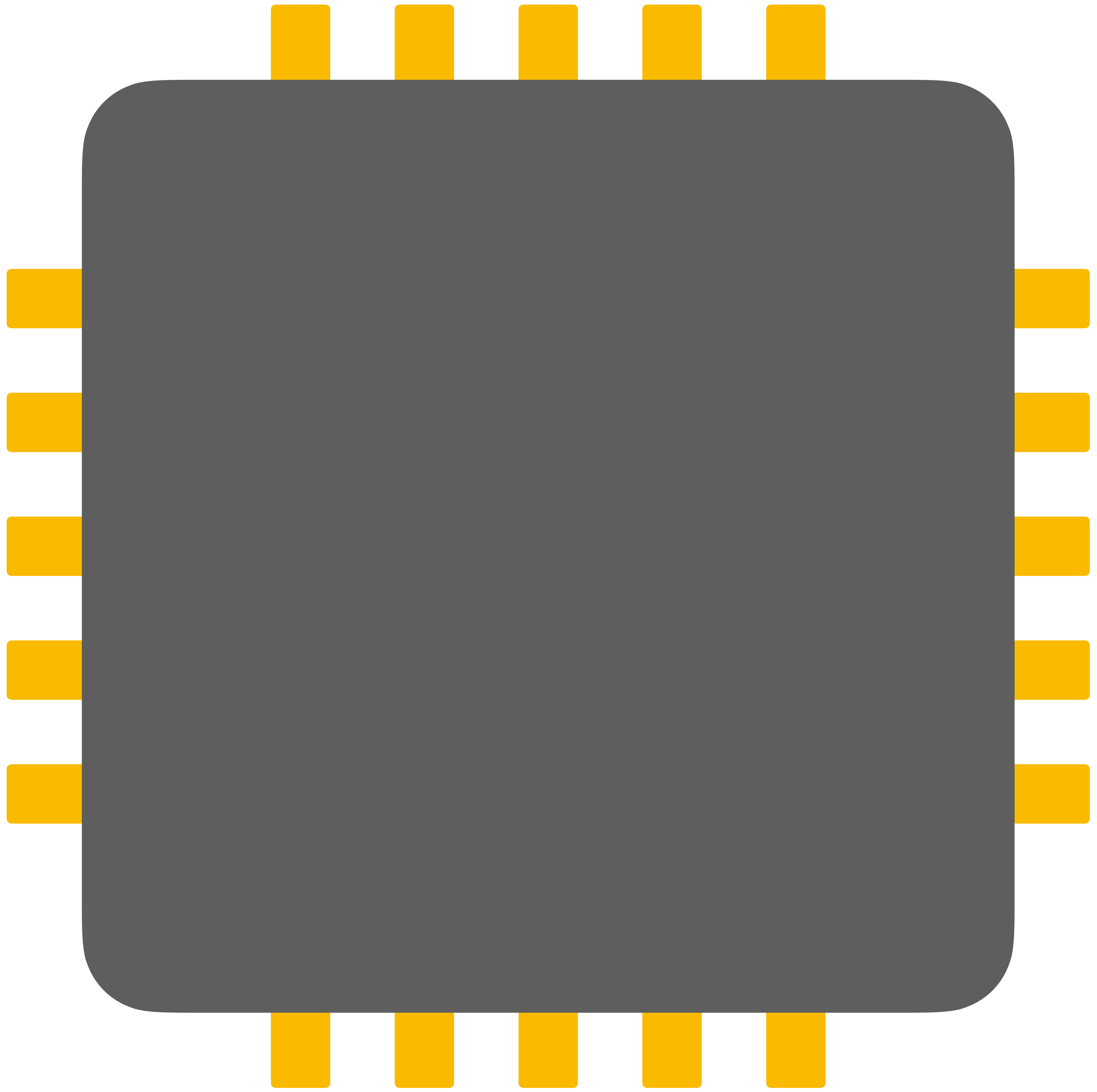
Leaks addresses of non-spec. loads/stores/ instruction fetches

Leaks "everything"

Leaks addresses of all loads/stores/ instruction fetches

Hardware Countermeasures

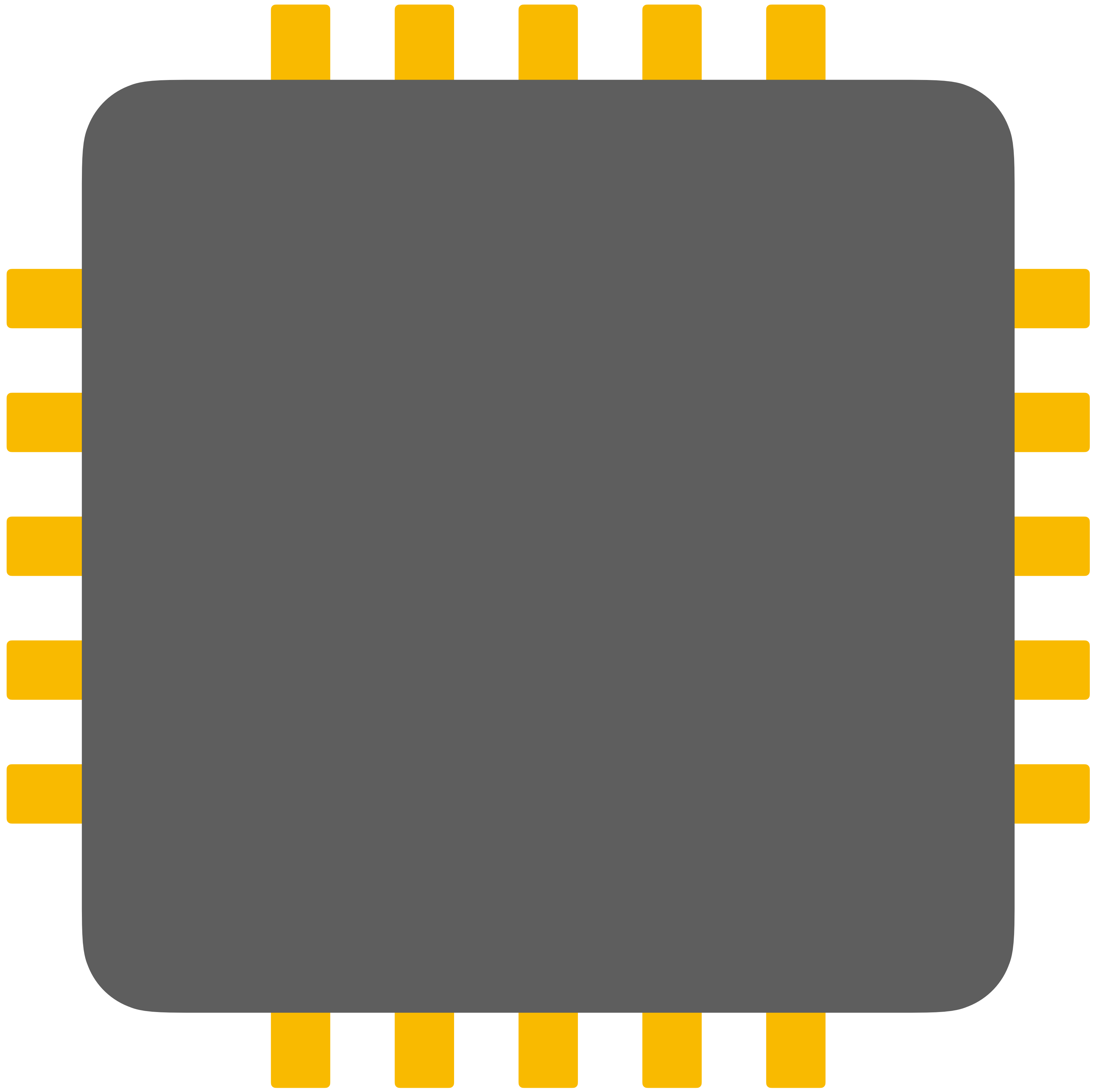
A Simple Processor



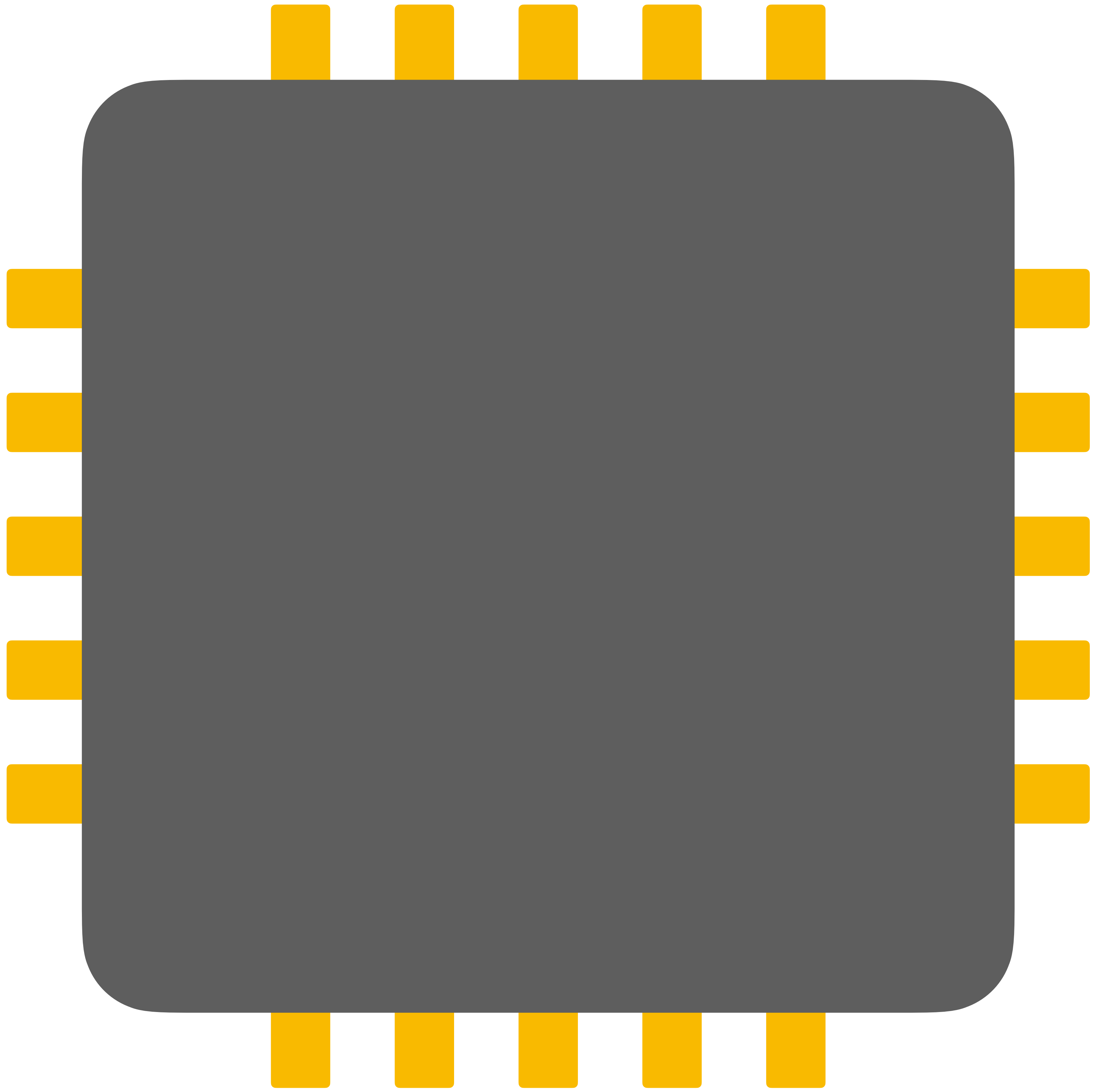
A Simple Processor

3-stage pipeline

(fetch, execute, retire)



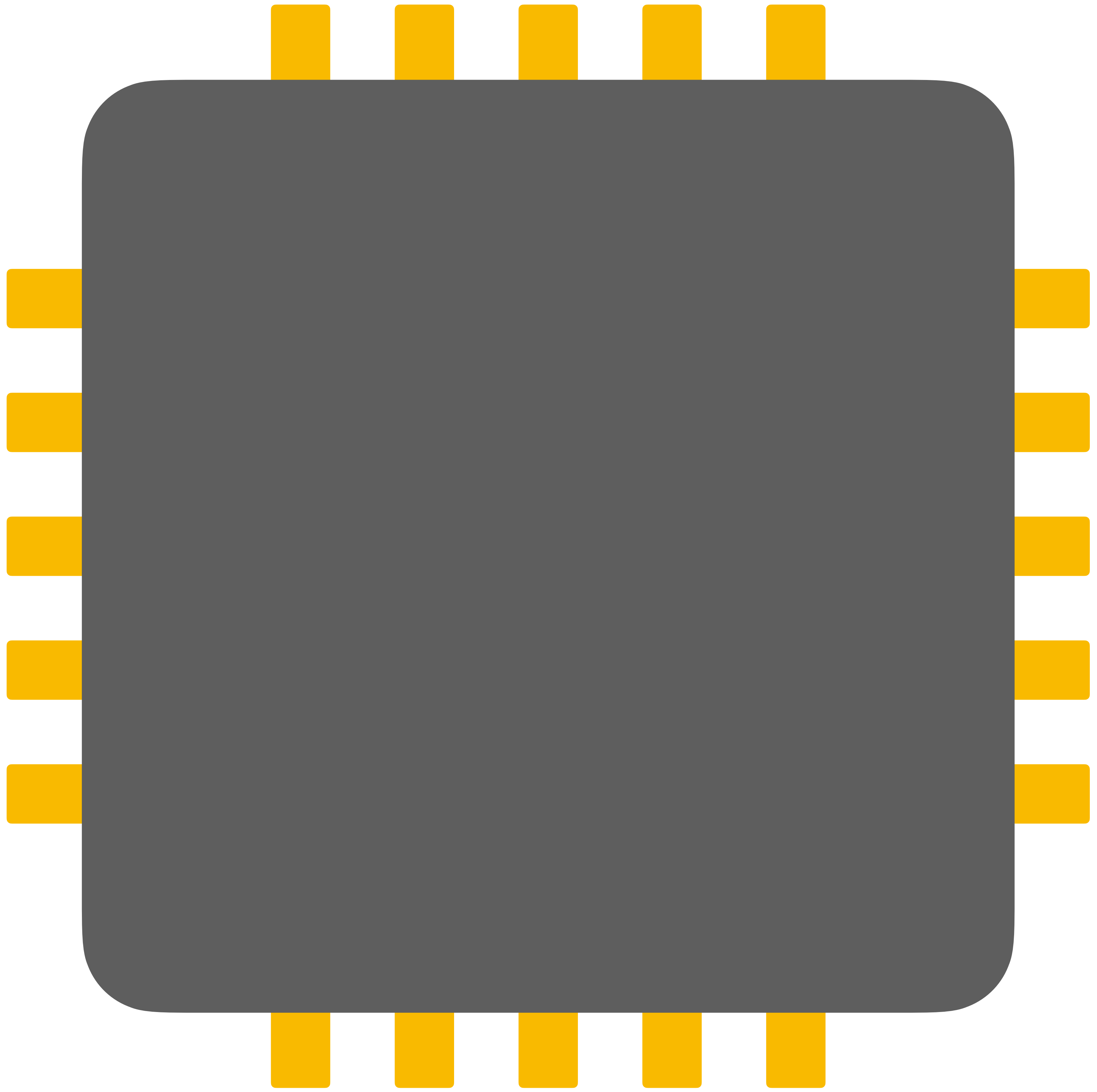
A Simple Processor



3-stage pipeline
(fetch, execute, retire)

Speculative and *out-of-order* execution

A Simple Processor

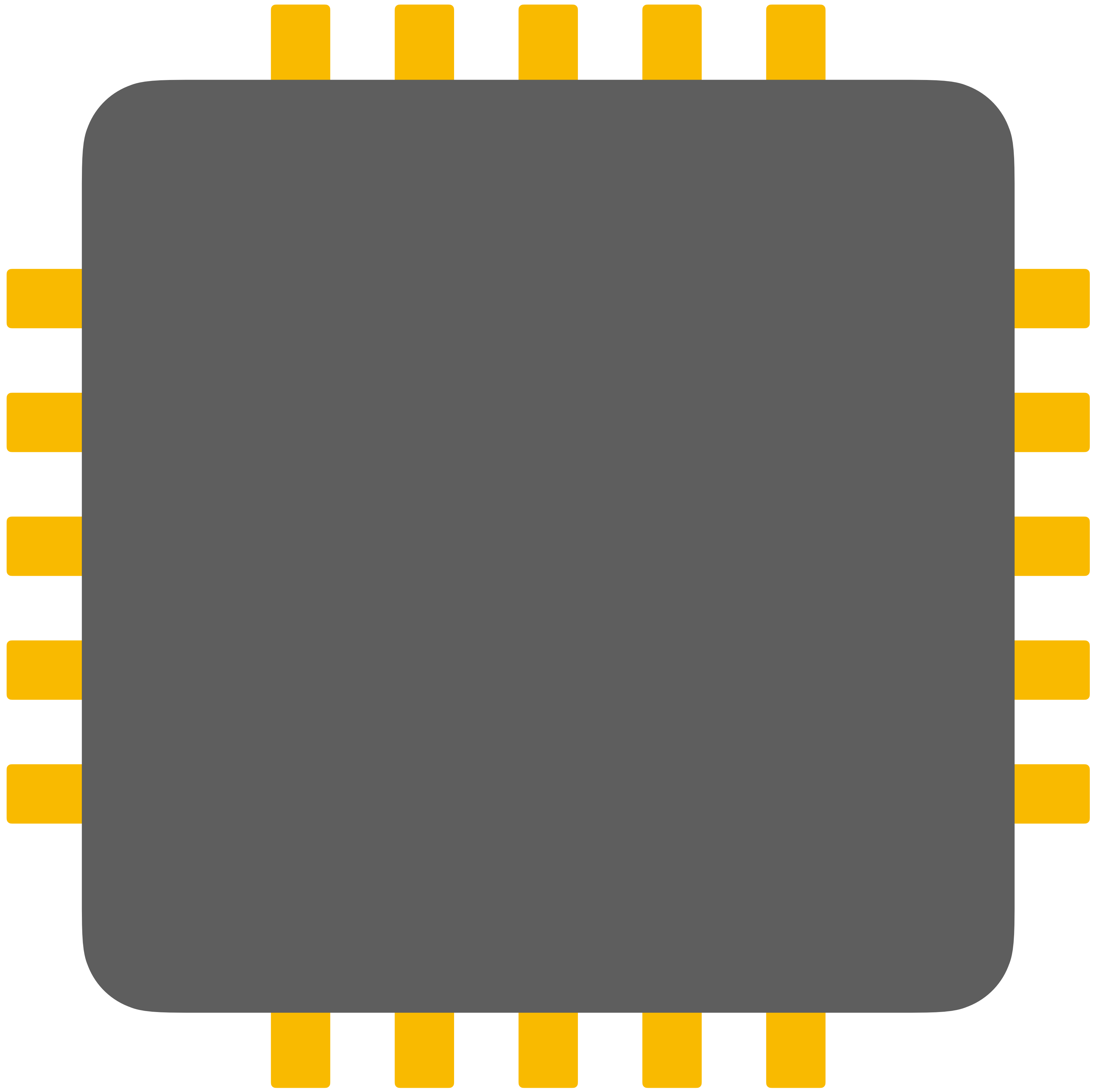


3-stage pipeline
(fetch, execute, retire)

Speculative and *out-of-order* execution

Parametric in *branch predictor* and *memory hierarchy*

A Simple Processor



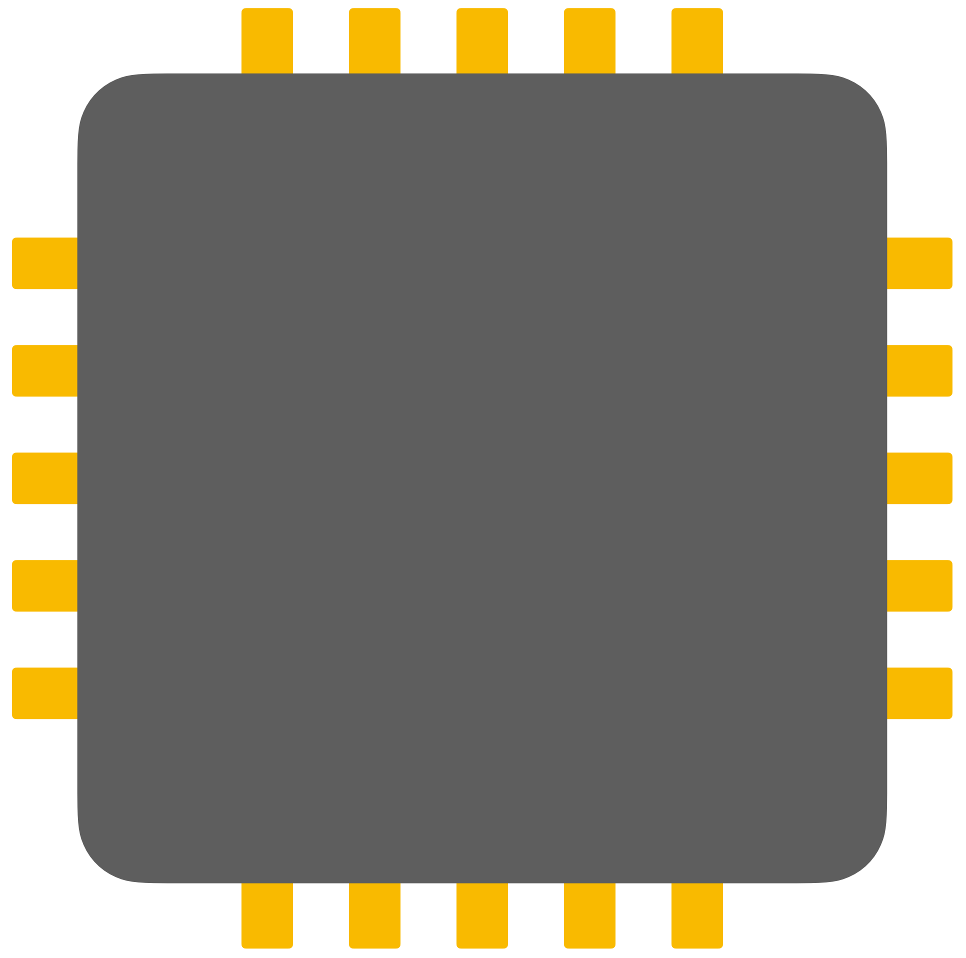
3-stage pipeline
(fetch, execute, retire)

Speculative and *out-of-order* execution

Parametric in *branch predictor* and *memory hierarchy*

Different *schedulers* for different countermeasures

Disabling Speculative Execution



Disabling Speculative Execution

Instructions are executed *sequentially*:

(fetch, execute, retire)*

Disabling Speculative Execution

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

🥳 No speculative leaks 🥳

Disabling Speculative Execution

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

🥳 No speculative leaks 🥳

Satisfies **seq-ct**

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

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

Delaying loads until all sources of *speculation are resolved*

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]
```

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

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

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

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
```

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

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

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

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*] 😞

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**

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

Taint speculatively loaded data

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

Taint speculatively loaded data

Propagate taint through computation

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

Taint speculatively loaded data

Propagate taint through computation

Delay tainted operations

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]
```

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

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

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

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]
```

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

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

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*

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] 😞

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 Countermeasures *[The World until 2018]*

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

No Countermeasures *[The World until 2018]*

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

Leaks addressed of speculative and non-speculative accesses

No Countermeasures *[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

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

Security Guarantees

$$\begin{tikzcd}
 & \text{seq-ct} & \text{no speculation} \arrow[l, bend right] \\
\text{seq-arch} \arrow[ur] & \text{seq-ct+spec-pc} \arrow[u] & \\
\text{spec-arch} \arrow[u] \arrow[r] & \text{spec-ct} \arrow[u] &
\end{tikzcd}$$

Security Guarantees

$$\begin{tikzcd}
 & \text{seq-ct} & \text{no speculation} \arrow[l, bend right] \\
\text{seq-arch} \arrow[ur] & \text{seq-ct+spec-pc} \arrow[u] & \text{Load Delay} \arrow[l, bend right] \arrow[ll, bend left] \\
 & \text{spec-ct} \arrow[u] & \\
\text{spec-arch} \arrow[uu] \arrow[ur] & &
\end{tikzcd}$$

Security Guarantees

$$\begin{tikzcd}
 & \text{seq-ct} & \text{no speculation} \arrow[l] \\
\text{seq-arch} \arrow[ur] & \text{seq-ct+spec-pc} \arrow[u] & \text{Load Delay} \arrow[l] \arrow[ll, bend left] \\
\text{spec-arch} \arrow[u] \arrow[r] & \text{spec-ct} \arrow[u] & \text{Taint Tracking} \arrow[l] \arrow[ull, bend left]
\end{tikzcd}$$

Security Guarantees

$$\begin{tikzcd}
 & \text{seq-ct} & \text{no speculation} \arrow[l] \\
\text{seq-arch} \arrow[ur] & & \\
 & \text{seq-ct+spec-pc} \arrow[uu] & \text{Load Delay} \arrow[l] \arrow[ull, bend left] \\
 & \text{spec-ct} \arrow[u] & \text{Taint Tracking} \arrow[l] \arrow[uull, bend left] \\
\text{spec-arch} \arrow[uuu] \arrow[ur] & & \\
 & & \text{no countermeasure} \arrow[uul, bend left]
\end{tikzcd}$$

Secure Programming

Secure Programming: Foundations

Secure Programming: Foundations

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

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 $\sigma \approx_\pi \sigma'$ then $[\![p]\!](\sigma) = [\![p]\!](\sigma')$

Secure Programming: Foundations

Specify secret data

$\uparrow$

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

Secure Programming: Foundations

Specify secret data

$\uparrow$

Program p is ***non-interferent*** wrt contract $[\![\cdot]\!]$ and policy π

if for all arch. states σ, σ': if $\sigma \approx_\pi \sigma'$ then $[\![p]\!](\sigma) = [\![p]\!](\sigma')$

Secure Programming: Foundations

Specify secret data

$\uparrow$

Program p is ***non-interferent*** wrt contract $[\![\cdot]\!]$ and policy π

if for all arch. states σ, σ': if $\sigma \approx_\pi \sigma'$ then $[\![p]\!](\sigma) = [\![p]\!](\sigma')$

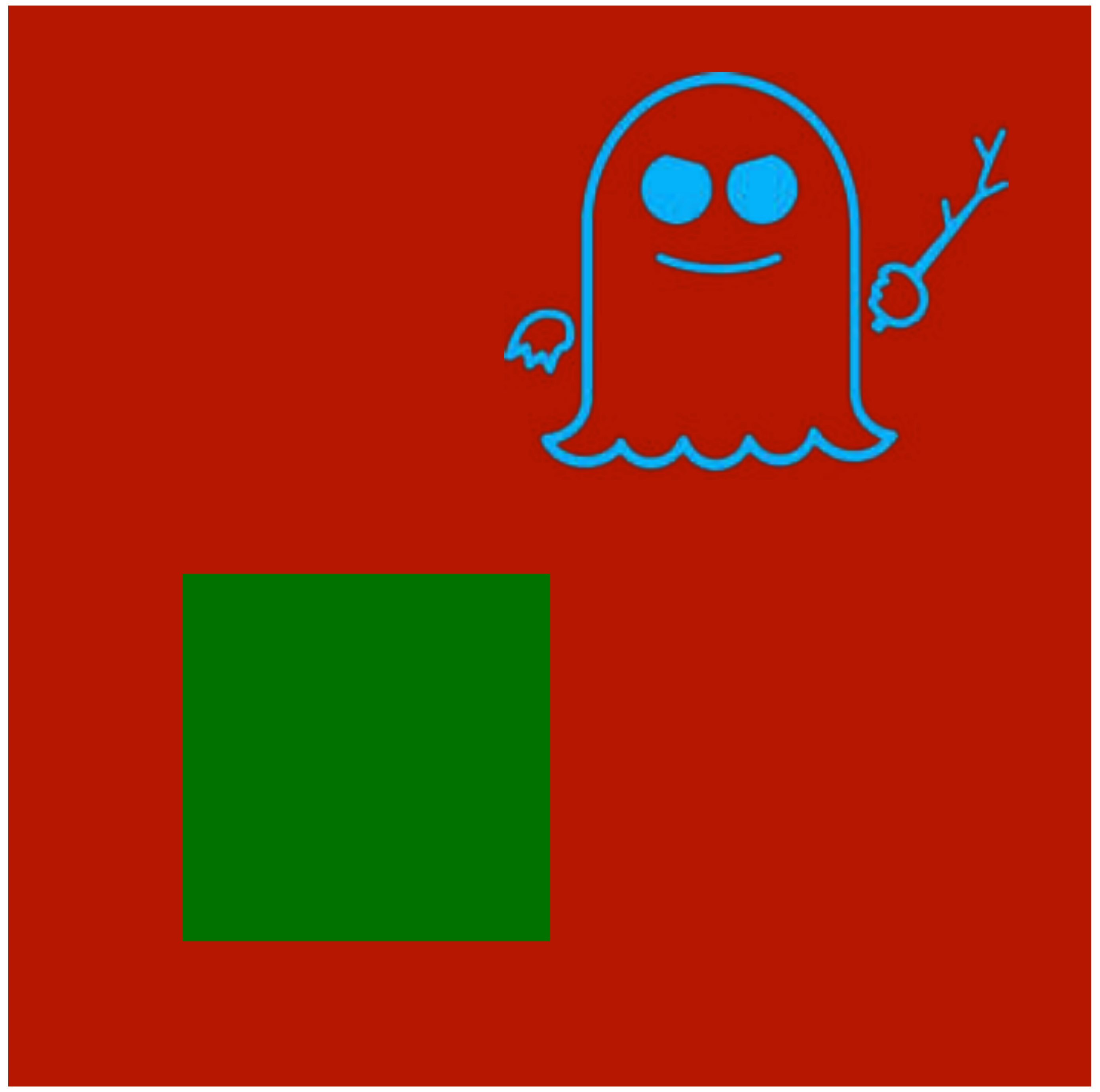
Theorem

If p is ***non-interferent*** wrt contract $[\![\cdot]\!]$ and policy π,

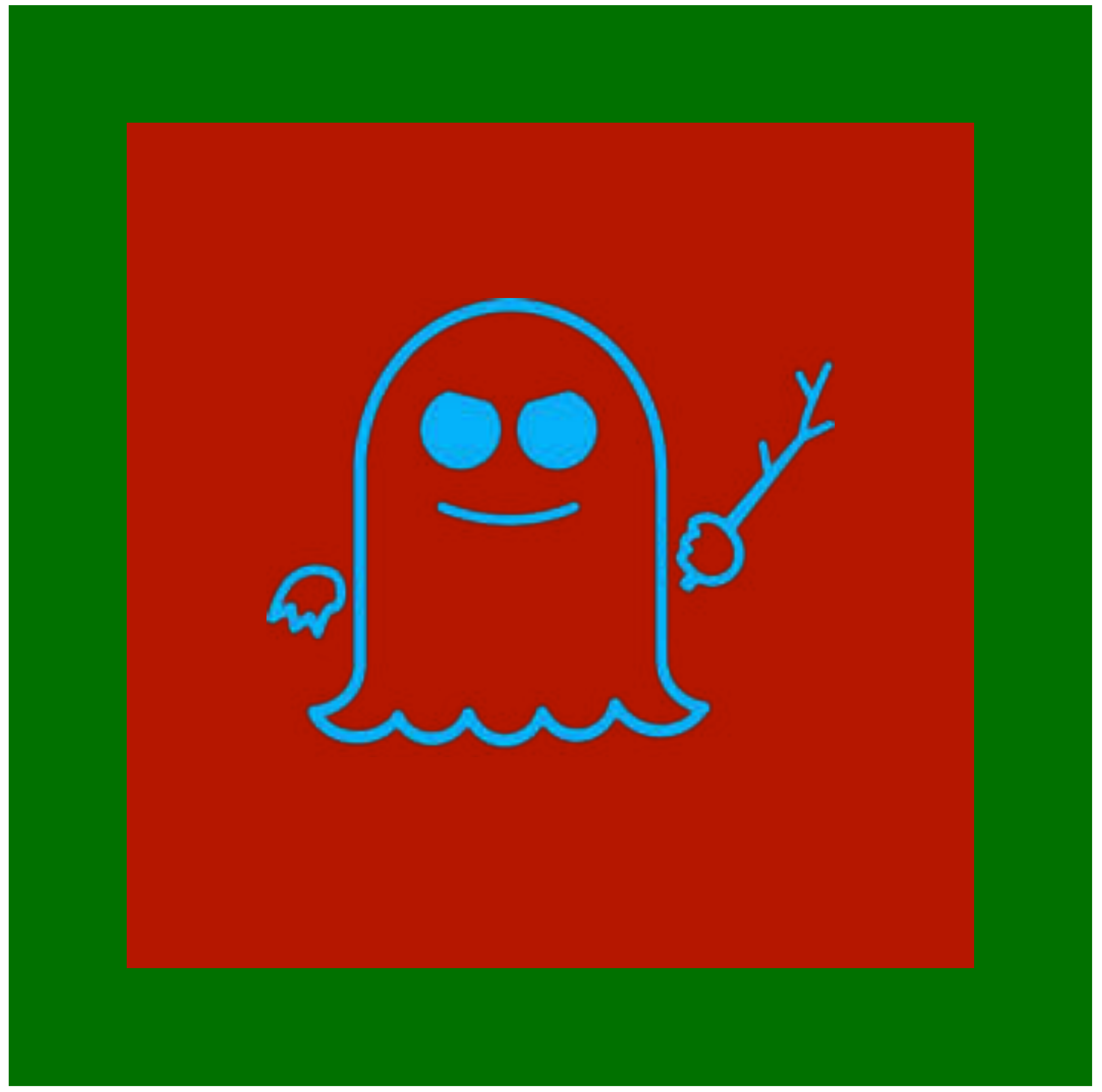
and hardware $\{\!|\cdot|\!\}$ satisfies $[\![\cdot]\!]$, then

p is ***non-interferent*** wrt hardware $\{\!|\cdot|\!\}$ and policy π

Two Flavors of Secure Programming

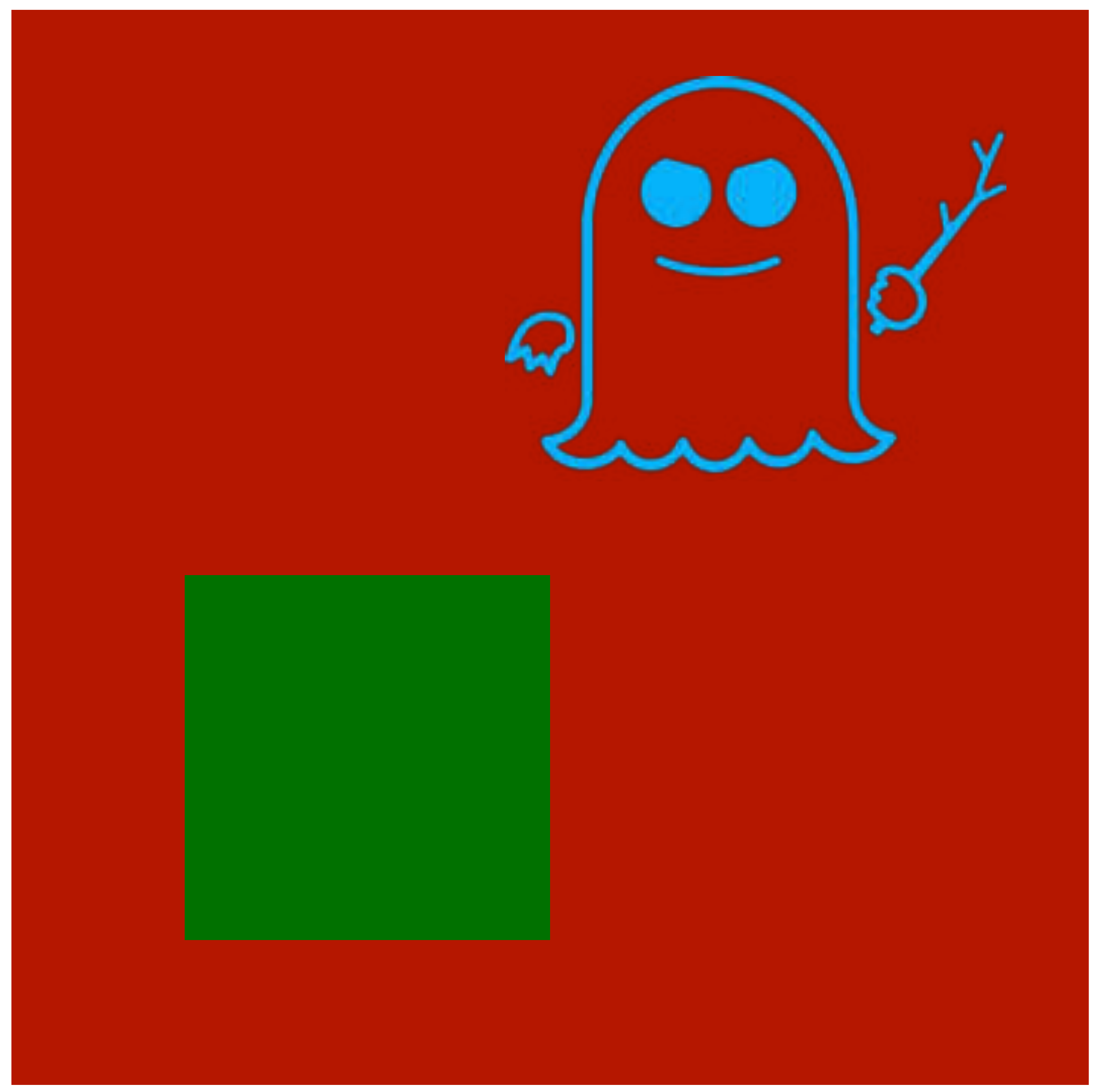


Constant-time

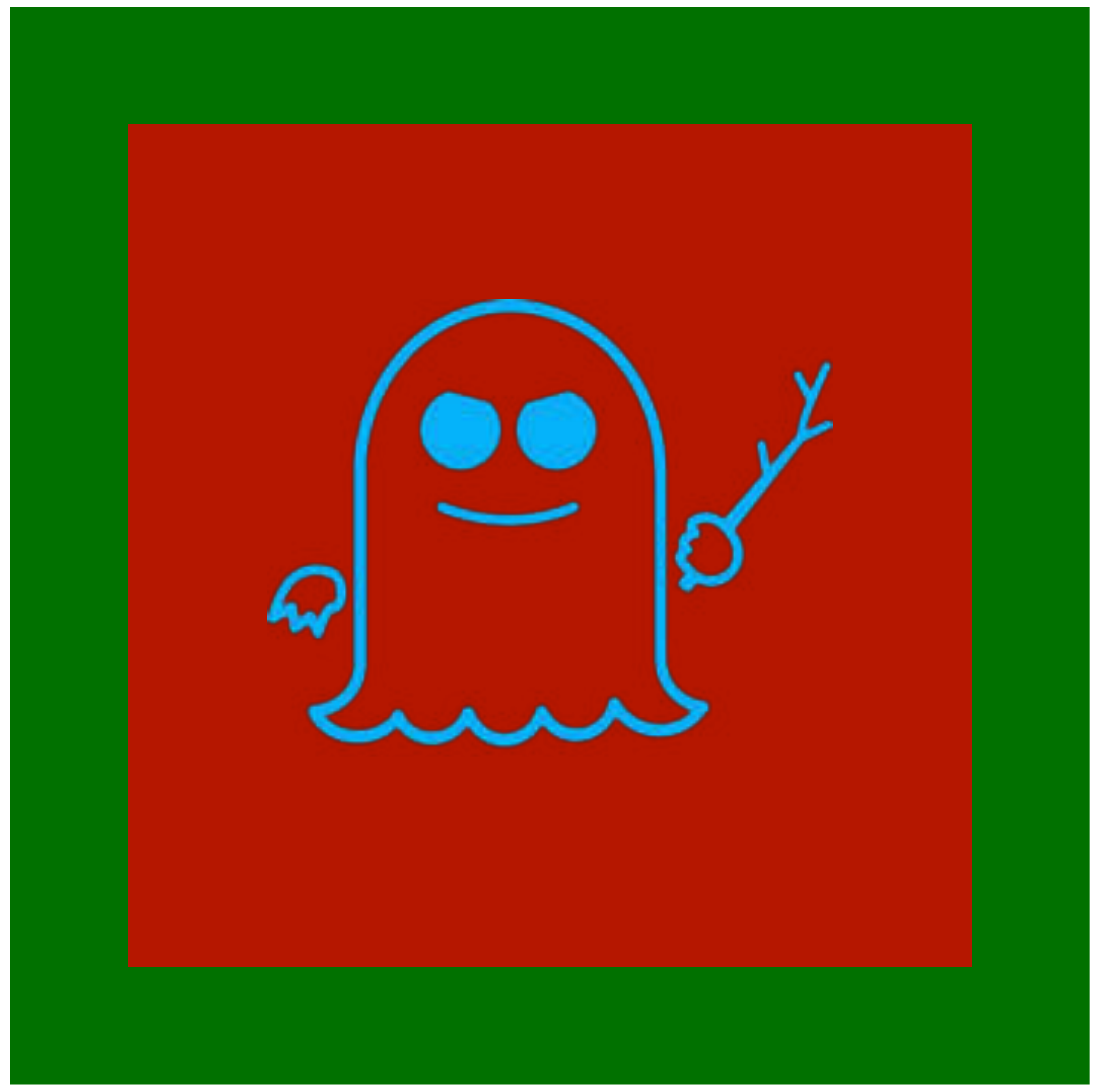


Sandboxing

Two Flavors of Secure Programming

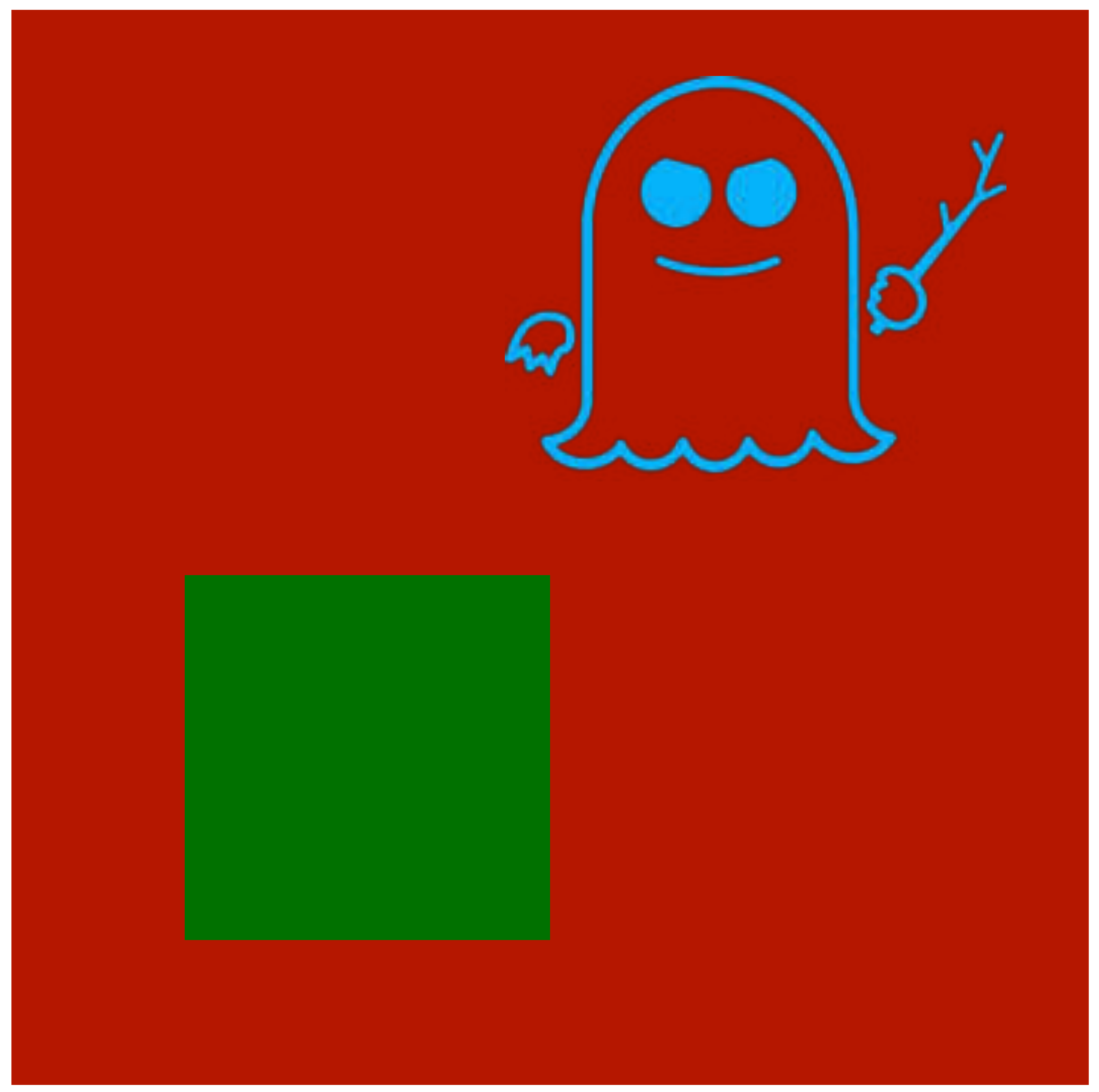


Constant-time

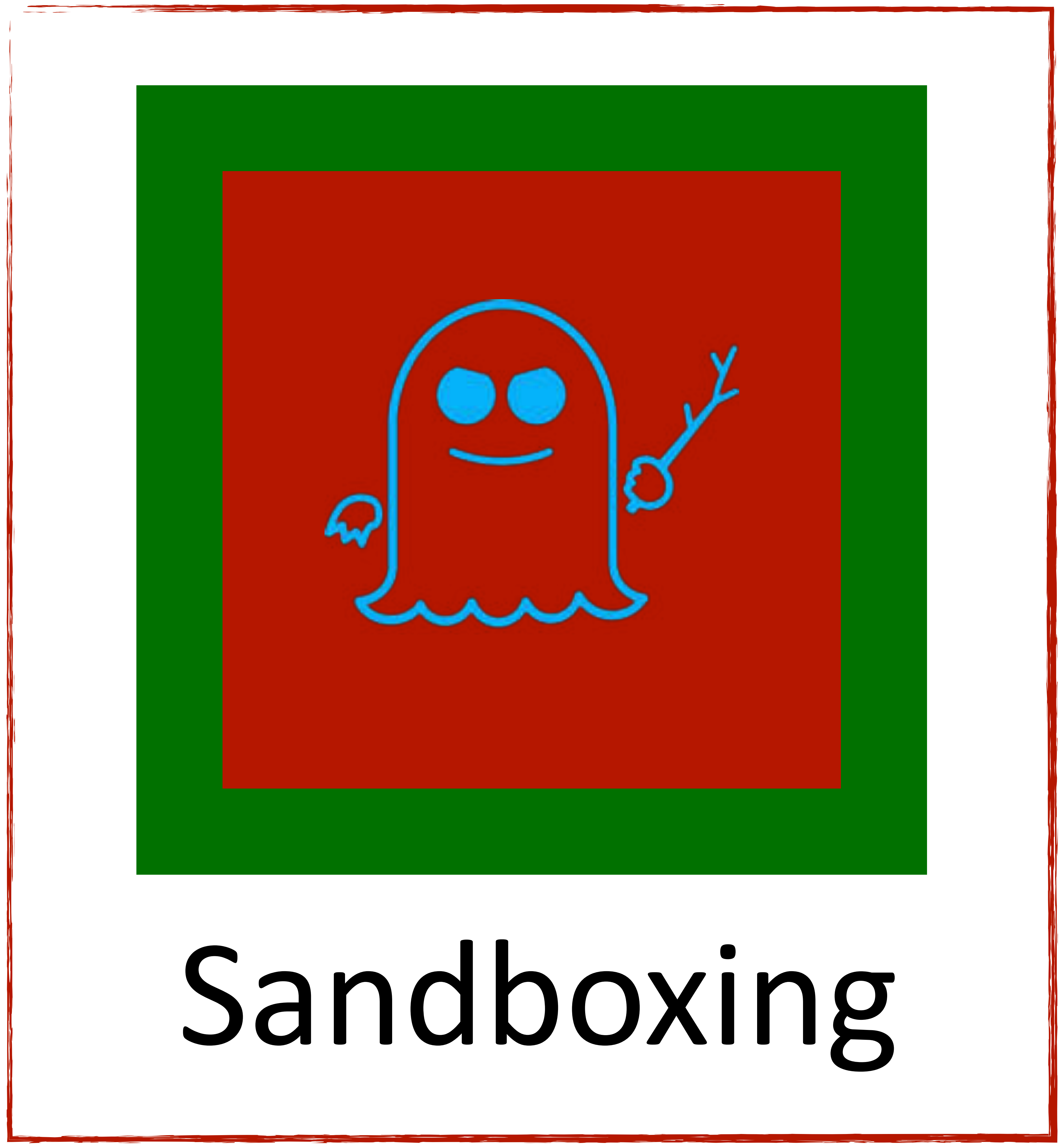


Sandboxing

Two Flavors of Secure Programming



Constant-time



Sandboxing

Constant-time Programming

Constant-time Programming

Traditional CT wrt policy $\pi \equiv$ non-interference wrt **seq-ct** and π

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

Constant-time Programming

Control-flow and memory accesses do not depend on secrets

Traditional CT wrt policy π $\equiv$ non-interference wrt **seq-ct** and π

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

Sandboxing

Sandboxing

Traditional SB wrt policy $\pi \equiv$ non-interference wrt **seq-arch** and π

Sandboxing

Programs never access high memory locations (out-of-sandbox)

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

Sandboxing

Programs never access high 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 Secure Programming

	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]*

Checking Secure Programming

	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]*

Checking Secure Programming

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

Checking Secure Programming

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

Conclusions

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

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

Should strive for **simple** and **mechanism-independent** contracts.

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

Should strive for **simple** and **mechanism-independent** contracts.

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