Hardware-software security contracts

Principled foundations for building secure microarchitectures

Marco Guarnieri
IMDEA Software Institute

Based on joint work with
Boris Köpf @ Microsoft Research
Jan Reineke @ Saarland University
José F. Morales, Pepe Vila, Andrés Sánchez @ IMDEA Software Institute
Marco Patrignani @ CISPA

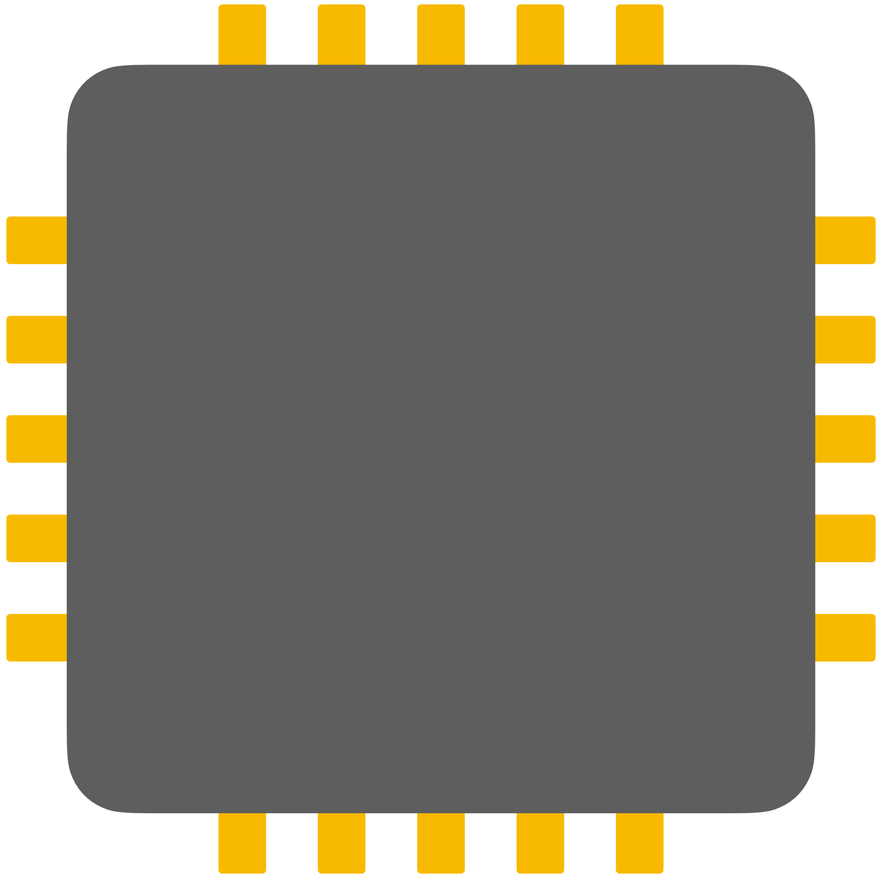
Contacts:
marco.guarnieri@imdea.org
@MarcoGuarnier1

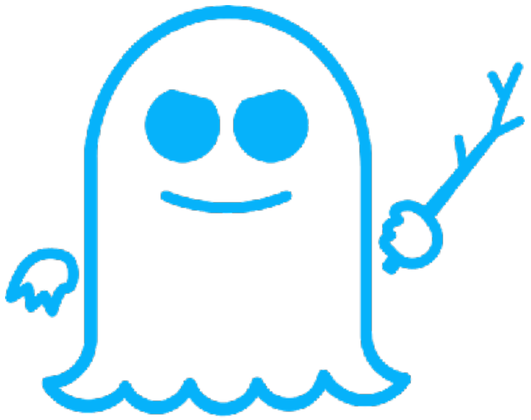
institute IMdea software

- Security
- Programming languages
- Formal methods

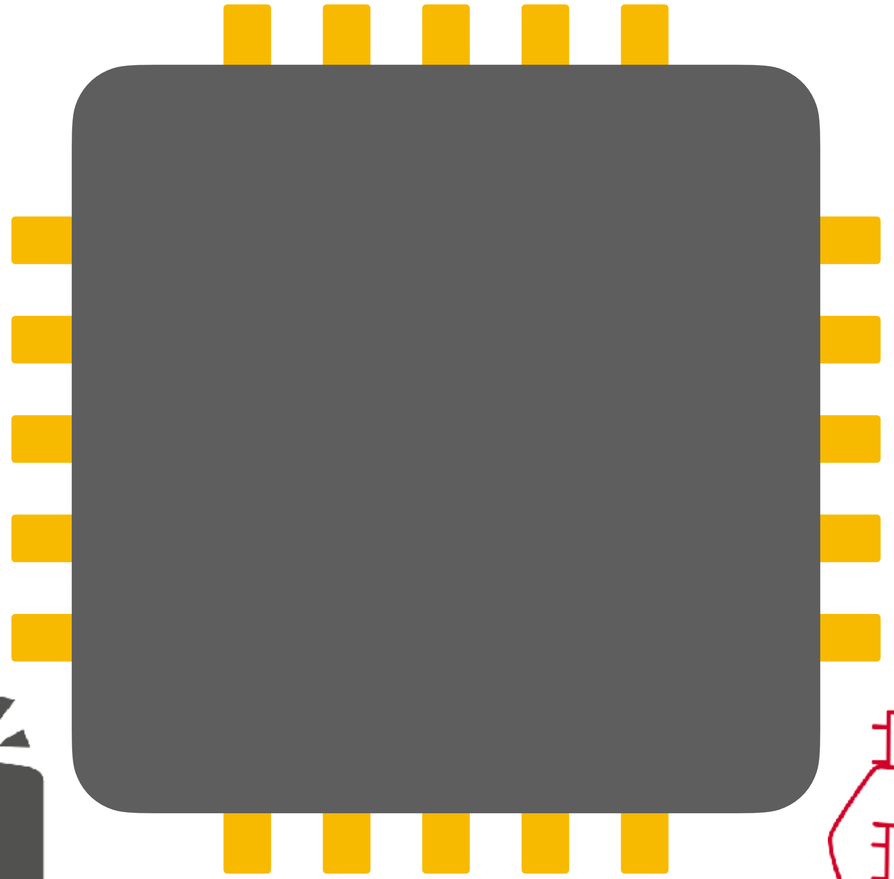
Outline

- Introduction
- Speculative execution attacks
- Hardware countermeasures and limitations
- Hardware/software security contracts for secure speculation
- Next steps & challenges



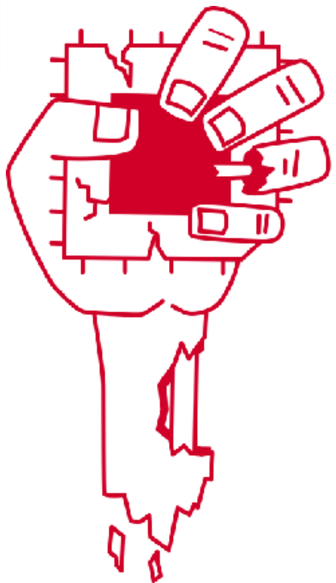


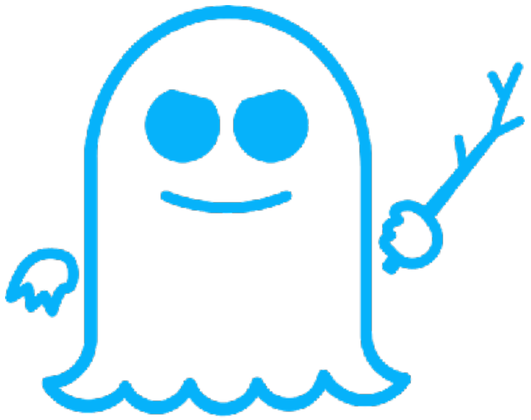
SPECTRE



MELTDOWN

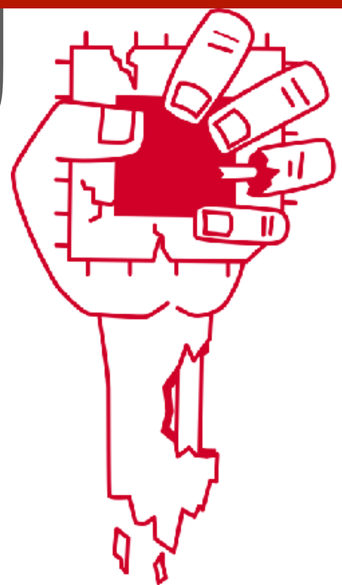
FORESHADOW





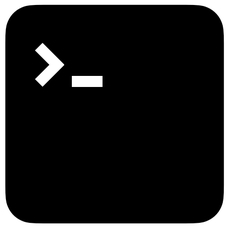
Attacks exploit μarchitectural side-effects invisible at ISA level

FORESHADOW



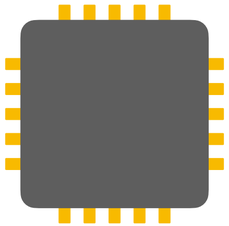
ISA: Benefits

High-level language



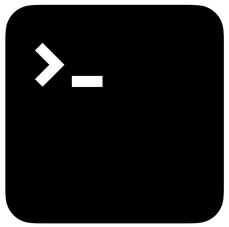
Instruction set architecture (ISA)

Microarchitecture



ISA: Benefits

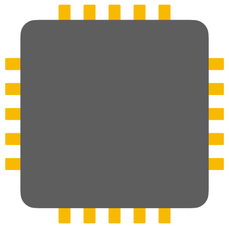
High-level language



Can program **independently** of microarchitecture

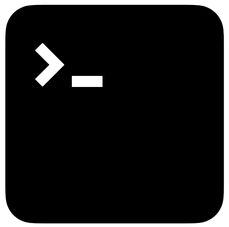
Instruction set architecture (ISA)

Microarchitecture



ISA: Benefits

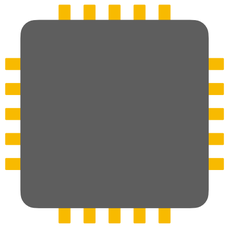
High-level language



Can program **independently** of microarchitecture

Instruction set architecture (ISA)

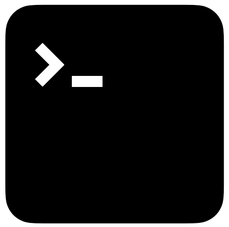
Microarchitecture



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

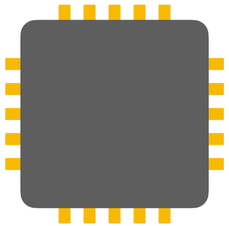
ISA: (security) pitfalls

High-level language



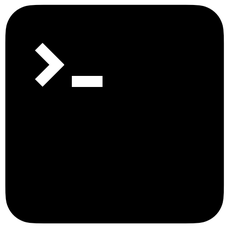
Instruction set architecture (ISA)

Microarchitecture



ISA: (security) pitfalls

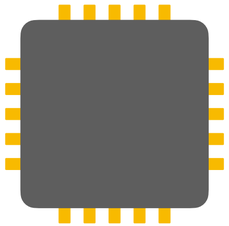
High-level language



Instruction set architecture (ISA)

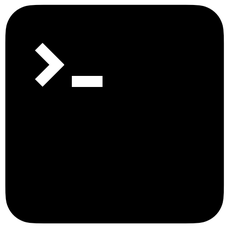
No guarantees about timing and side channels

Microarchitecture



ISA: (security) pitfalls

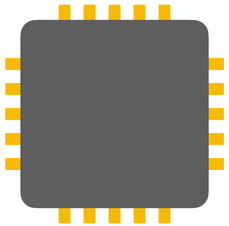
High-level language



Instruction set architecture (ISA)

No guarantees about timing and side channels

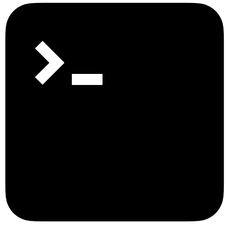
Microarchitecture



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

ISA: (security) pitfalls

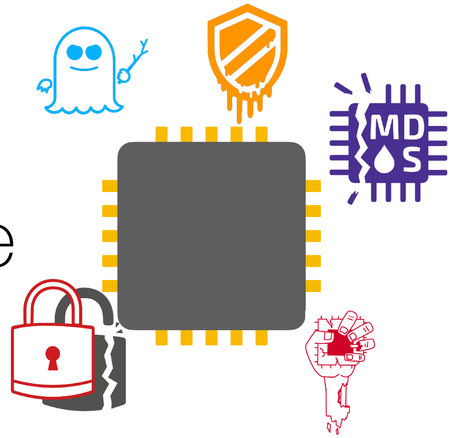
High-level language



Instruction set architecture (ISA)

No guarantees about timing and side channels

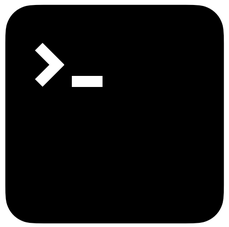
Microarchitecture



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

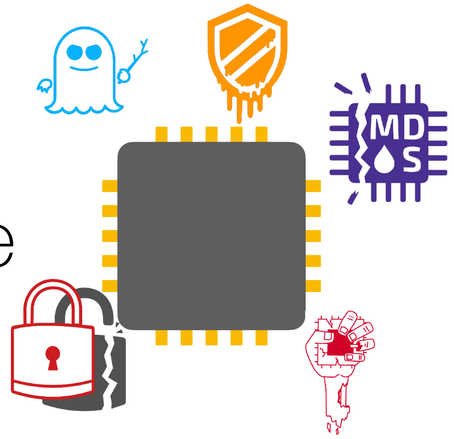
ISA: (security) pitfalls

High-level language



Instruction set architecture (ISA)

Microarchitecture



Impossible to program securely
cryptographic algorithms?
sandboxing untrusted code?

No guarantees about timing and side channels

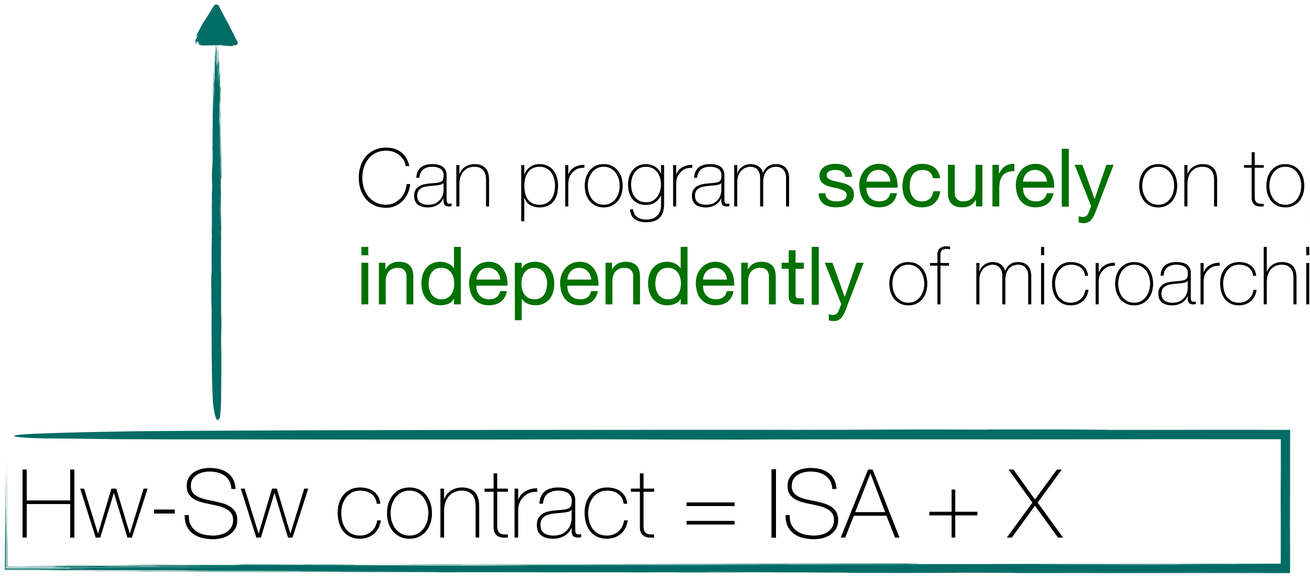
Can implement arbitrary **insecure** optimizations as long as ISA semantics are obeyed

A Way Forward: HW/SW Security Contracts

Hw-Sw contract = ISA + X

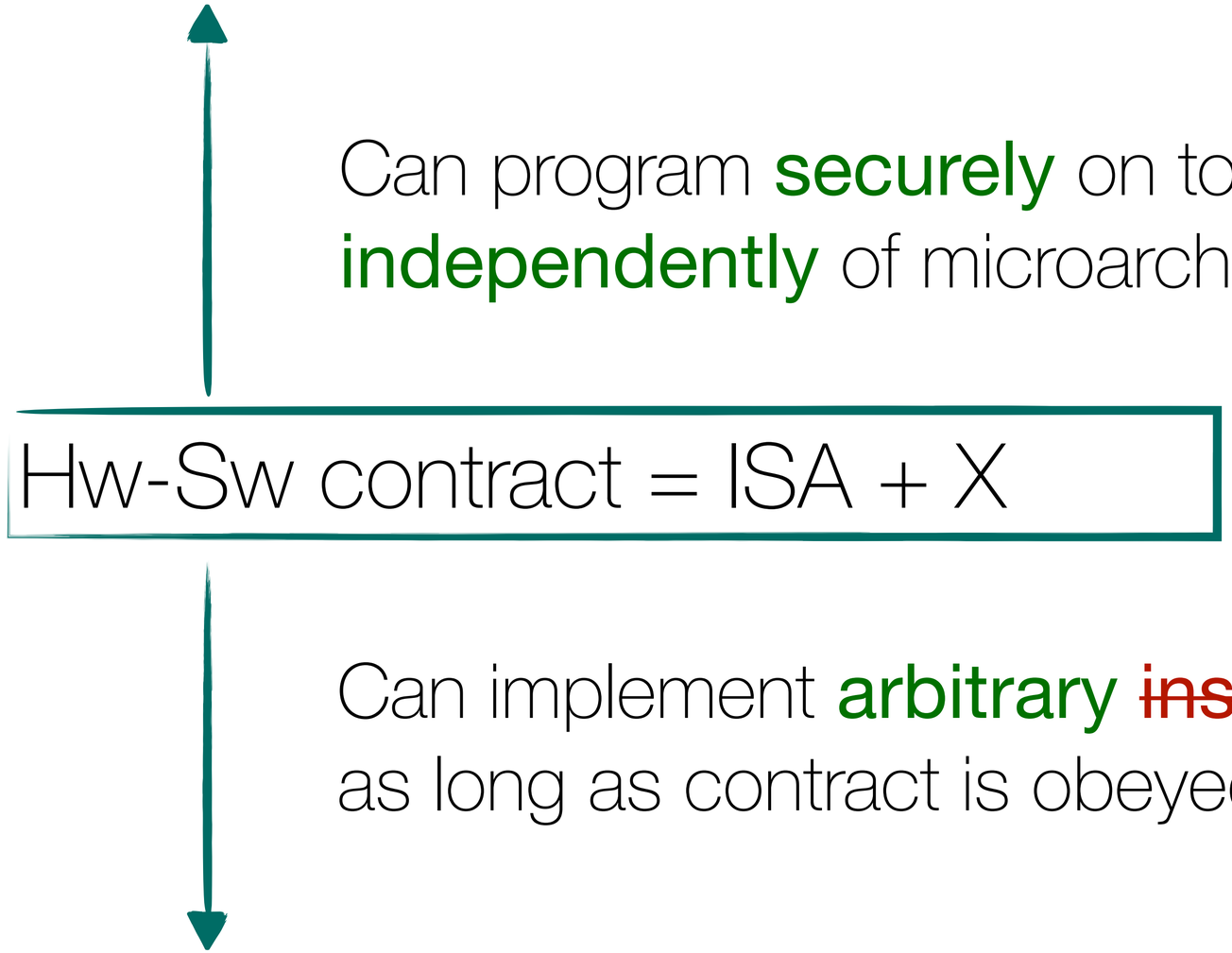
Succinctly captures possible information leakage

A Way Forward: HW/SW Security Contracts



Succinctly captures possible information leakage

A Way Forward: HW/SW Security Contracts



Succinctly captures possible information leakage

A Way Forward: HW/SW Security Contracts

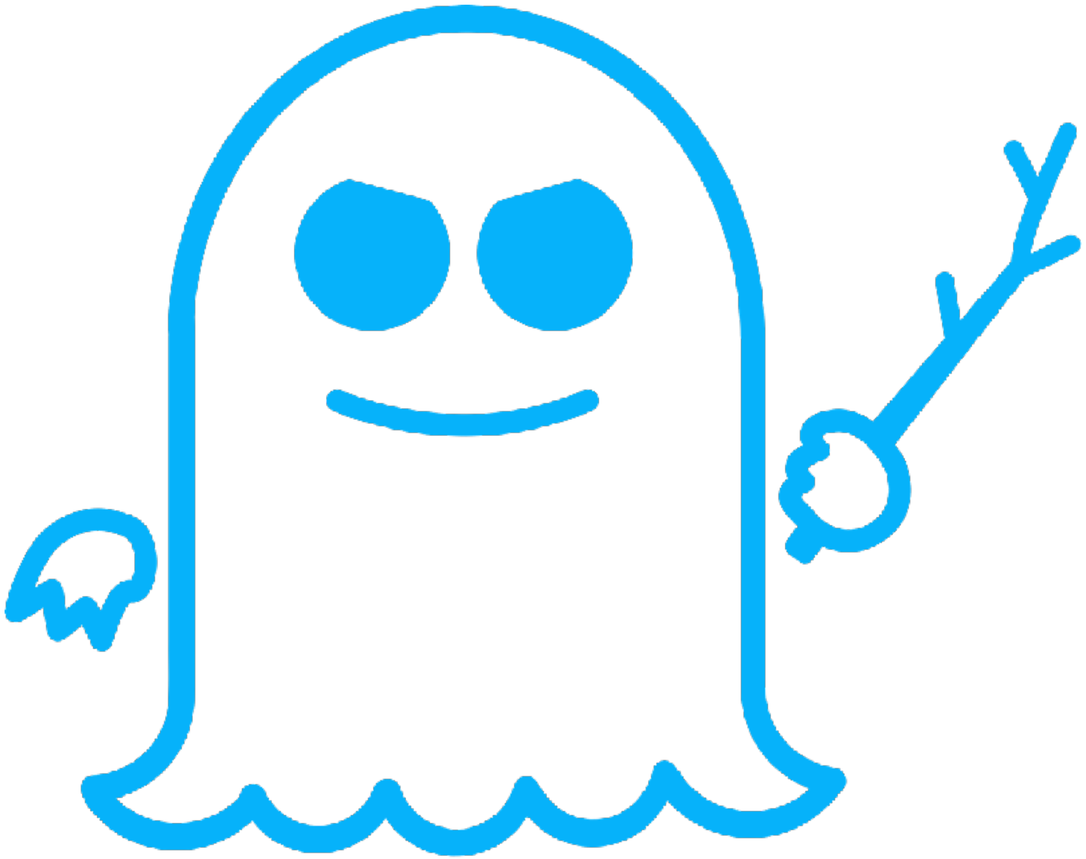
Can program **securely** on top contract

In this talk

Proof-of-concept of **HW/SW contracts** for ***secure speculation***

as long as contract is obeyed

Speculative execution attacks



SPECTRE

Exploits ***speculative execution***

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

P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom — Spectre Attacks: Exploiting Speculative Execution — S&P 2019

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

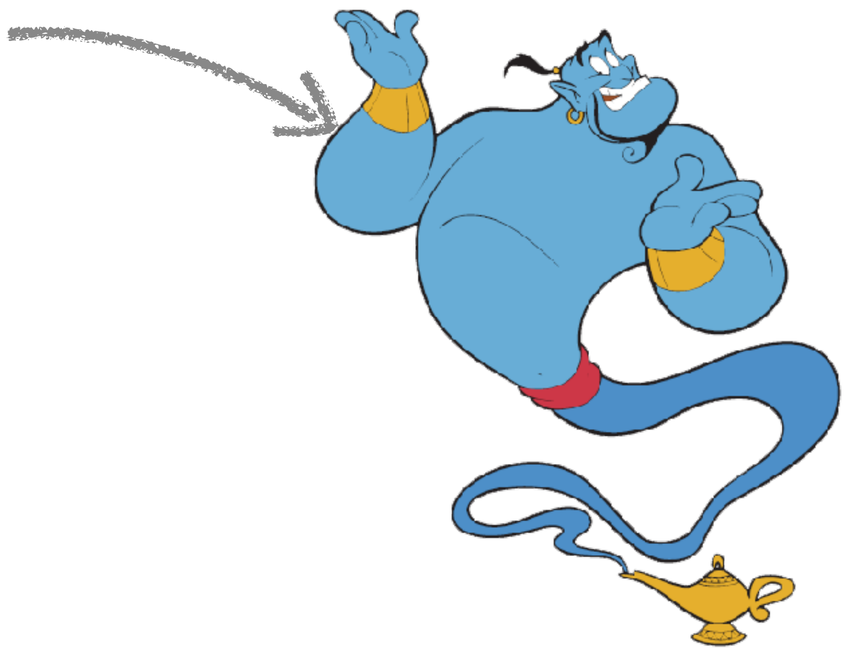
Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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



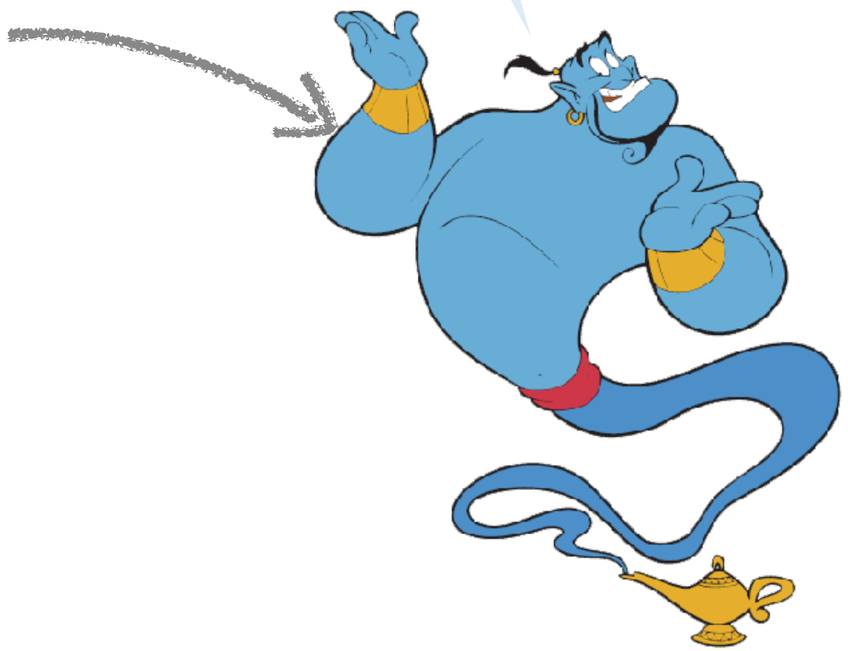
Branch predictor

Speculative execution + branch prediction

Size of array **A**

Prediction based on ***branch history*** & ***program structure***

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



Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

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

Branch predictor

Wrong predicton? ***Rollback changes***!

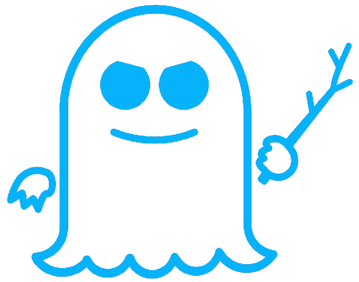
- Architectural (ISA) state
- Microarchitectural state

Spectre 101

Spectre 101

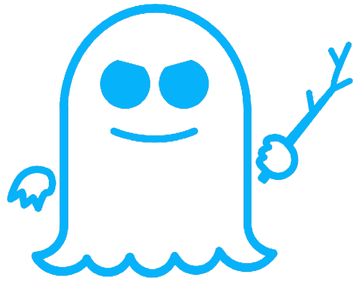
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

Spectre 101

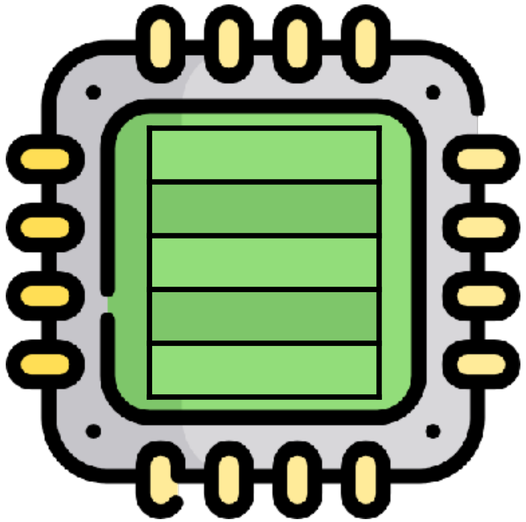


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

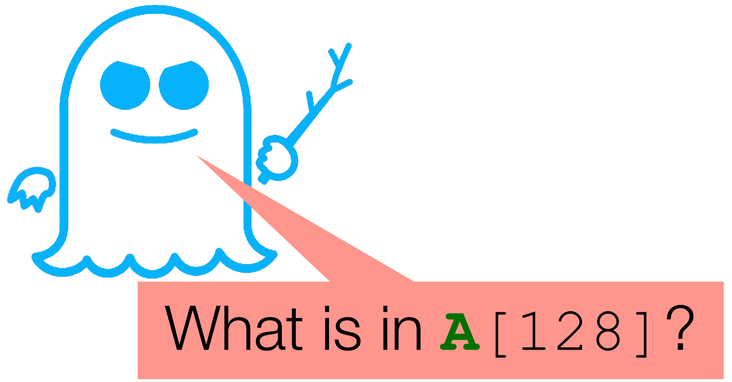
Spectre 101



```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

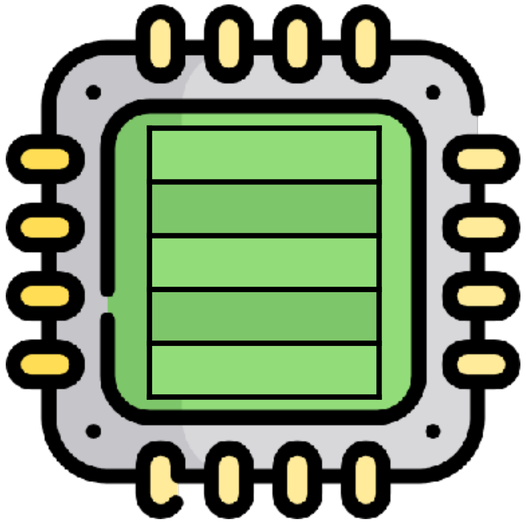


Spectre 101

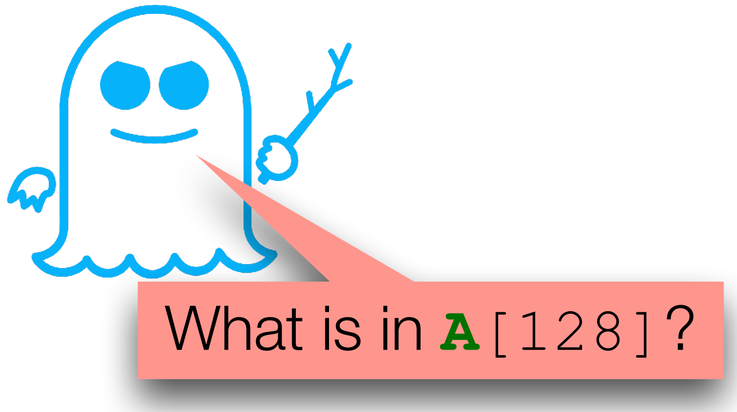


A_size=16


```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



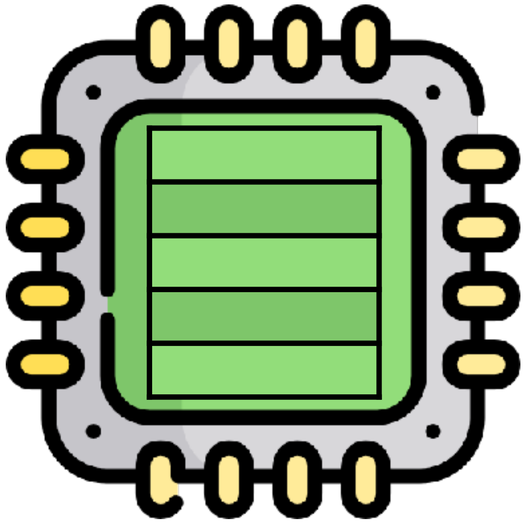
Spectre 101



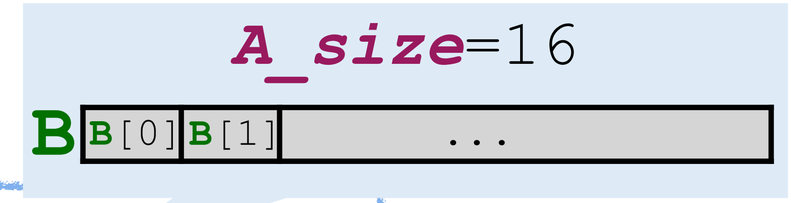

```
A_size=16
```

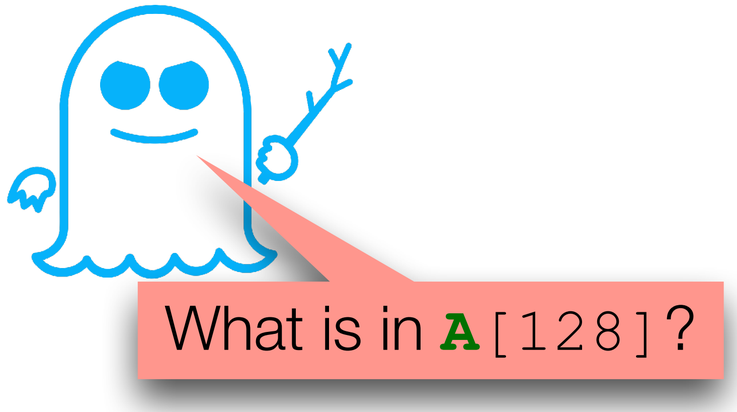

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Train branch predictor



Spectre 101

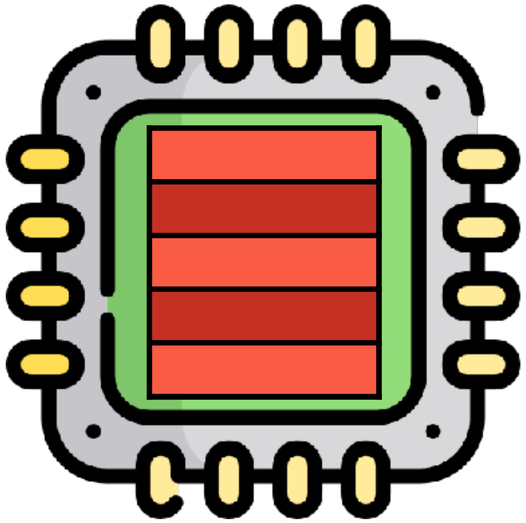




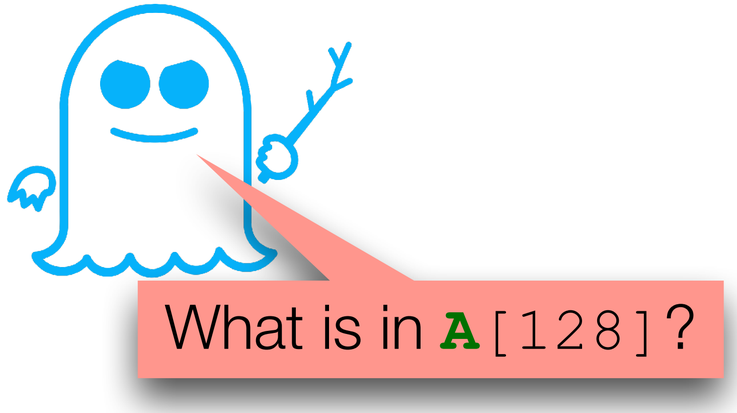

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Train branch predictor

2) Prepare cache

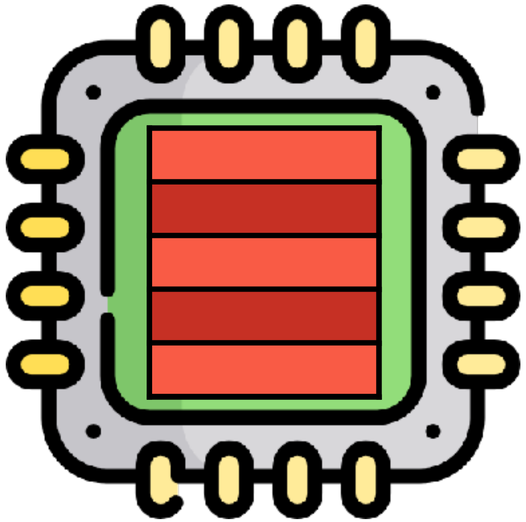


Spectre 101



A_size=16


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

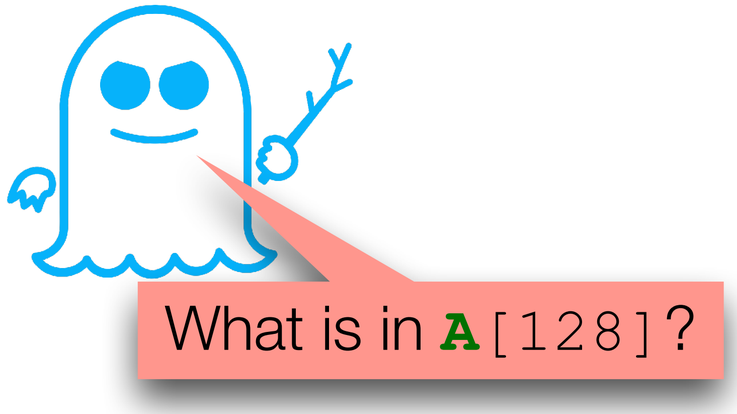


1) Train branch predictor

2) Prepare cache

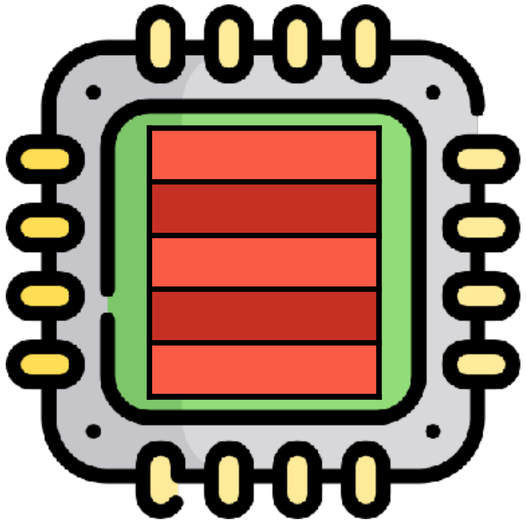
3) Run with *x* = 128

Spectre 101




```
A_size=16
```


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

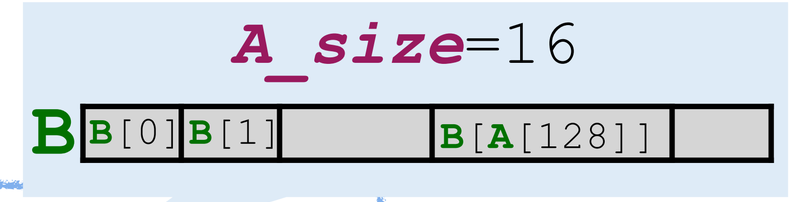


1) Train branch predictor

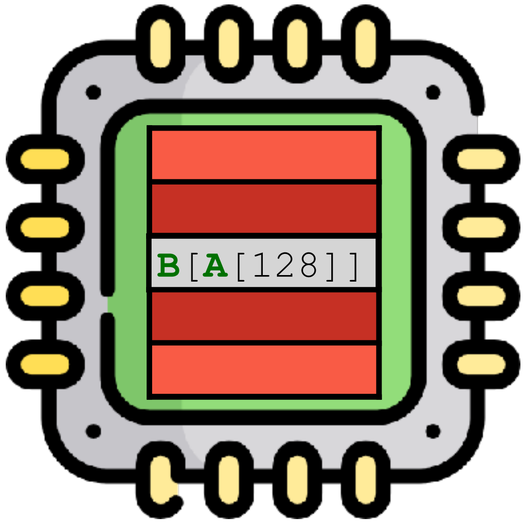
2) Prepare cache

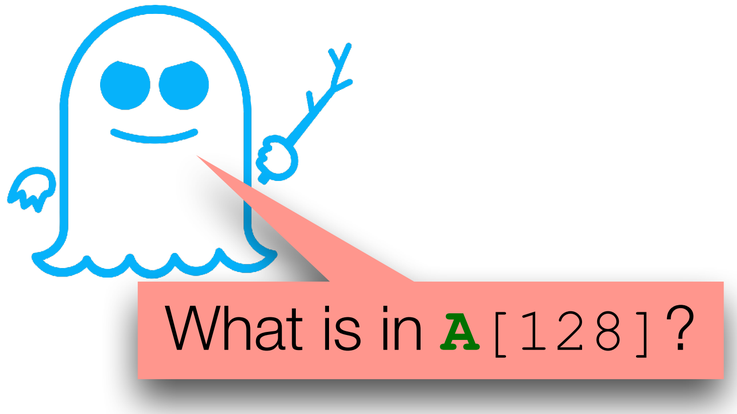
3) Run with `x = 128`

Spectre 101




```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



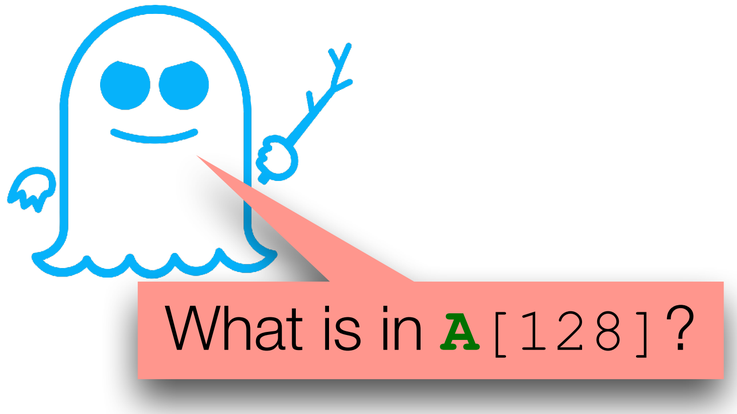


1) Train branch predictor

2) Prepare cache

3) Run with `x = 128`

Spectre 101



A_size=16

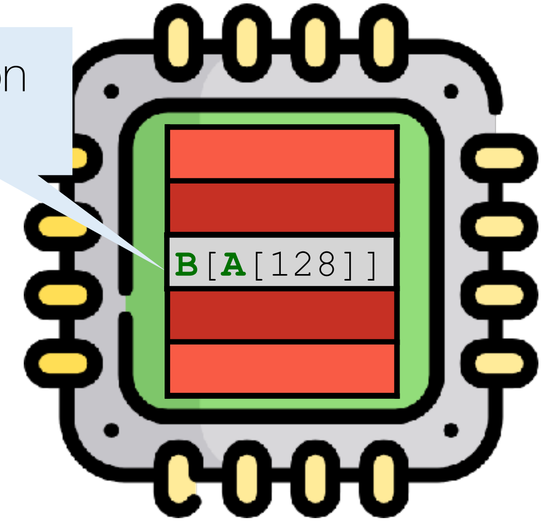

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Train branch predictor

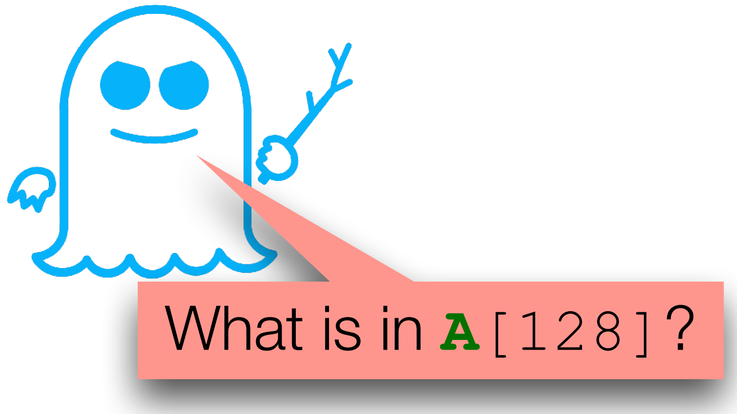
2) Prepare cache

3) Run with *x* = 128

Depends on A[128]



Spectre 101




```
A_size=16
```

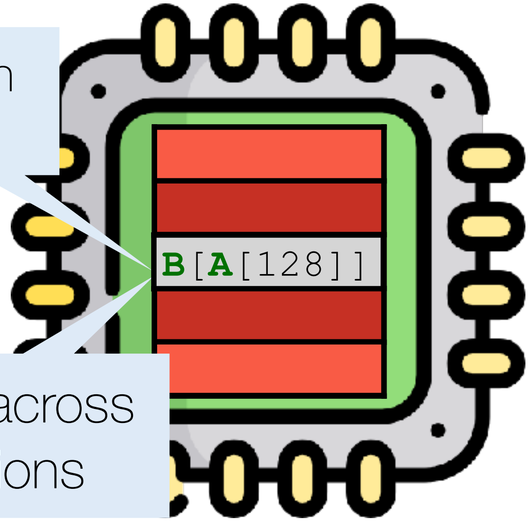

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Train branch predictor

2) Prepare cache

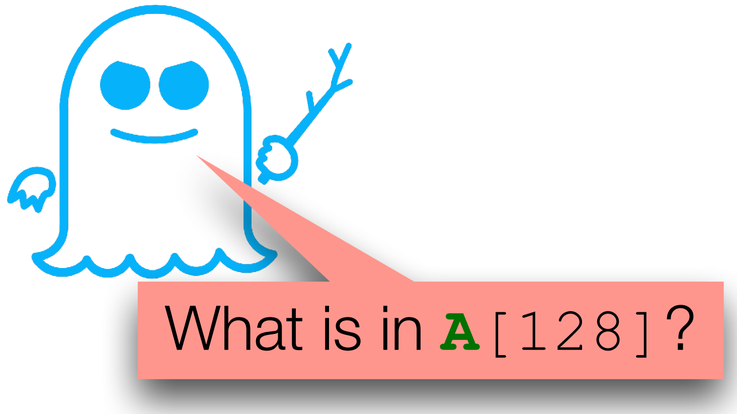
3) Run with `x = 128`

Depends on `A[128]`



Persistent across speculations

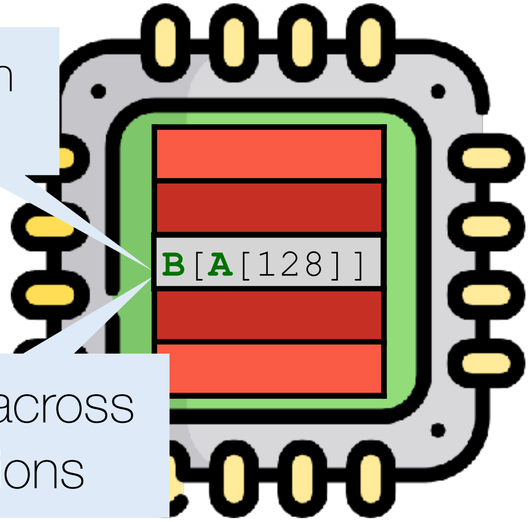
Spectre 101




```
A_size=16
```


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

Depends on A[128]



Persistent across speculations

1) Train branch predictor

2) Prepare cache

3) Run with `x = 128`

4) Extract from cache

Countermeasures

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 Technology

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 Technology

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

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

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

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

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

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

Hardware countermeasures

InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan†, Jiho Choi†, Dimitrios Skarlatos, Adam Morrison*, [illegible]
University of Illinois at Urbana-Champaign
{myan8, jchoi42, skarlat2}@illinois.edu, [illegible]

NDA: Preventing Speculative Execution Attacks at Their Source

Kevin Loughlin
University of Michigan

Ofir Weisse
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Thomas F. Wenisch
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 Murc
Murcia, Spain
aros@ditec.um

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

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

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

CleanupSpec: A [illegible]

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

Approach to Safe Speculation

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

Hardware countermeasures

InvisiSpec: Making Speculative Execution
Invisible in the Cache Hierar

Mengjia Yan†, Jiho Choi†, Dimitrios Skarlatos, Adam Morrison*,
University of Illinois at Urbana-Champ
{myan8, jchoi42, skarlat2}@illinois.edu, ma

NDA: Preventing Speculative Execution Attacks at Their Source

Kevin Loughlin
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Ofir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

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

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

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

Sakalis

CleanupSpec: A

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

Approach to Safe Speculation

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

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

Jiyong Yu
ity of Illinois at
aign

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

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

Josep Torrellas
University of Illinois at Urbana-Champaign
torrella@illinois.edu

Christopher W. Fletcher
University of Illinois at Urbana-Champaign
cwfletch@illinois.edu

Hardware countermeasures

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

Hardware countermeasures

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

Hardware countermeasures

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

Hardware countermeasu

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  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]

Hardware countermeasu

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  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]

Hardware countermeasures

```
1.  y = A[x]
2.  if (x < A_size)
3.     z = B[y]
4.  end
```

Hardware countermeasu

```
1.  y = A[x]
2.  if (x < A_size)
3.     z = B[y]
4.  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]

Hardware countermeasu

```
1.  y = A[x]
2.  if (x < A_size)
3.     z = B[y]
4.  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]

Hardware countermeasures

InvisiSpec: Making Speculative Execution
Invisible in the Cache Hierar

Mengjia Yan†, Jiho Choi†, Dimitrios Skarlatos, Adam Morrison*,
University of Illinois at Urbana-Champ
{myan8, jchoi42, skarlat2}@illinois.edu, ma

NDA: Preventing Speculative Execution Attacks at Their Source

Kevin Loughlin
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Ofir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

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

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

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

Sakalis

CleanupSpec: A

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

Approach to Safe Speculation

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

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

Jiyong Yu
ity of Illinois at
aign

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

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

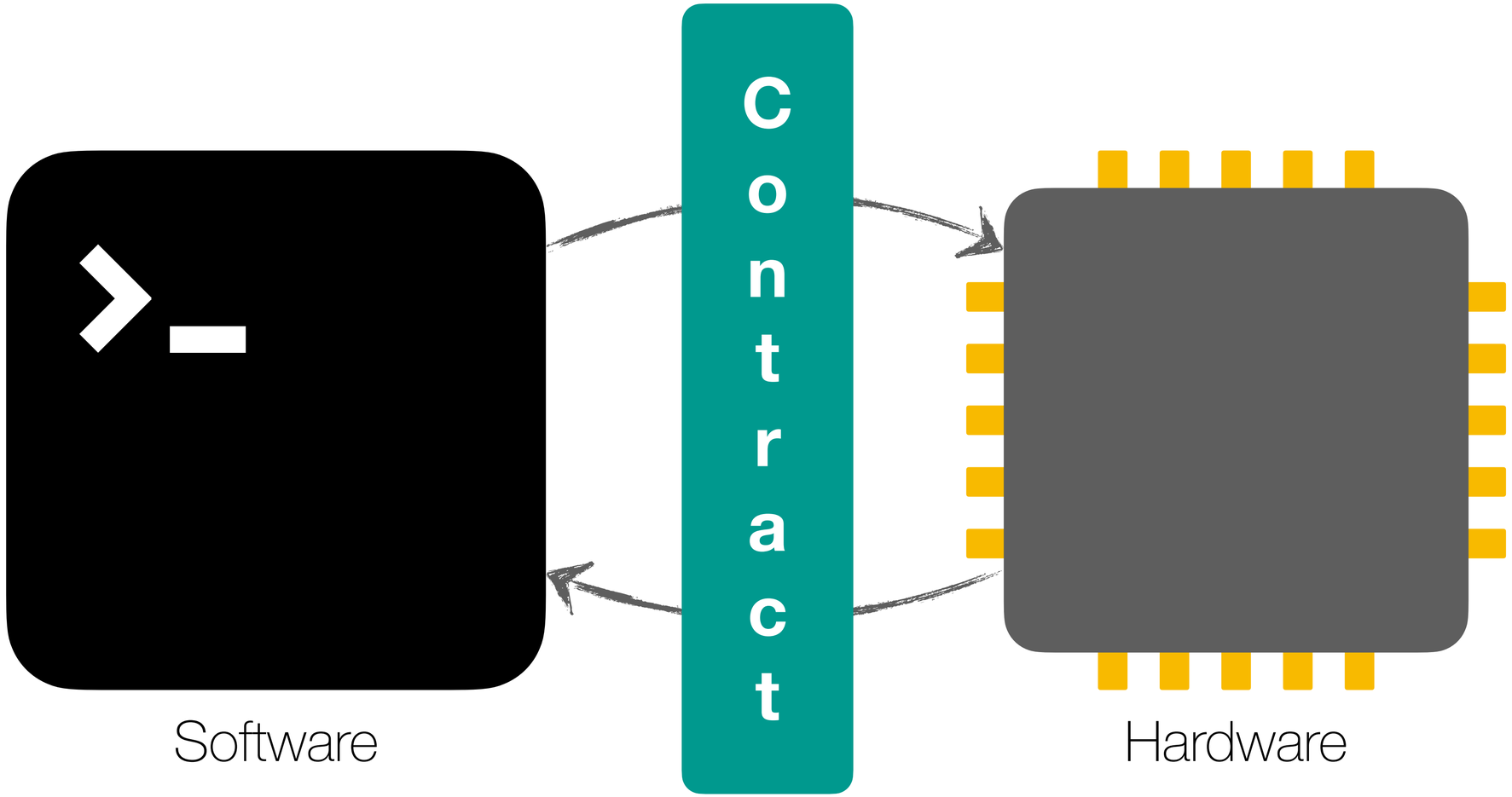
Josep Torrellas
University of Illinois at Urbana-Champaign
torrella@illinois.edu

Christopher W. Fletcher
University of Illinois at Urbana-Champaign
cwfletch@illinois.edu

Hardware countermeasures

Security guarantees?

How can we capture security guarantees?



Hardware/software security contracts

Guarnieri, Köpf, Reineke, Vila — Hardware-software contracts for secure speculation — IEEE S&P 2021
https://arxiv.org/abs/2006.03841

Contracts for side-channel-free systems

Contracts for side-channel-free systems

Contracts specify which ***program executions*** a side-channel ***adversary can distinguish***

Contracts for side-channel-free systems

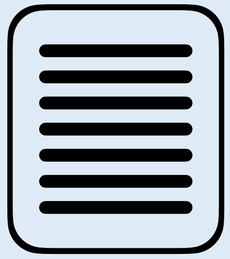
Contracts specify which ***program executions*** a side-channel ***adversary can distinguish***

Goals

- Capture security ***guarantees of HW***
- ***Basis*** for ***secure programming***

Contracts

Contracts

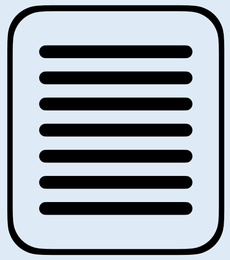


Contract

ISA extended with observations

Contracts

Observations expose security-relevant ***μarch events***

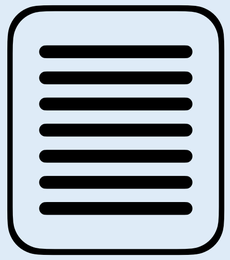


Contract

ISA extended with observations

Contracts

Observations expose security-relevant ***μarch events***



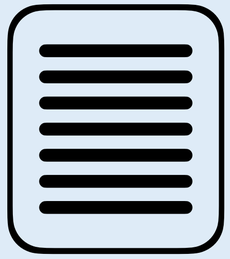
Contract

ISA extended with observations

Contract traces: (p, σ)

Observations expose security-relevant ***μarch events***

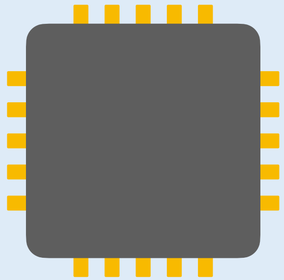
Contracts



Contract

ISA extended with observations

Contract traces: (p, σ)

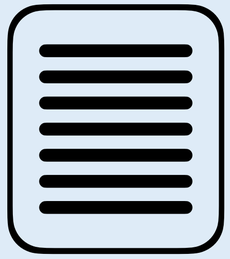


Hardware

Formal model of processor

Observations expose security-relevant ***μarch events***

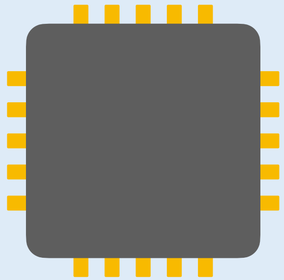
Contracts



Contract

ISA extended with observations

Contract traces: (p, σ)



Hardware

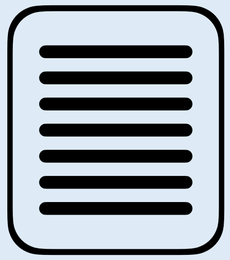
Formal model of processor

Hardware traces: (p, σ)

Contracts

Observations expose security-relevant ***μarch events***

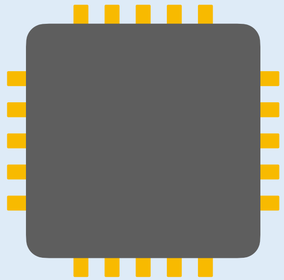
Attacker observes sequences of ***μarch states***



Contract

ISA extended with observations

Contract traces: (p, σ)



Hardware

Formal model of processor

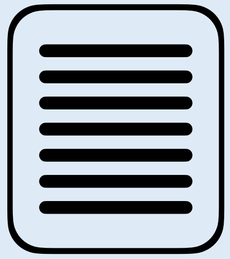
Hardware traces: (p, σ)

Contracts

Observations expose security-relevant ***μarch events***

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

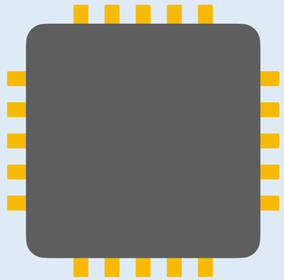
Contract



ISA extended with observations

Contract traces: (p, σ)

Hardware



Formal model of processor

Hardware traces: (p, σ)

Contract satisfaction

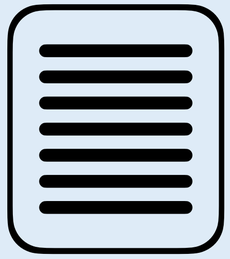
Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$

Contracts

Observations expose security-relevant ***μarch events***

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

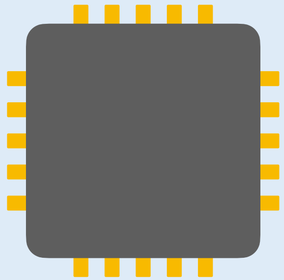
Contract



ISA extended with observations

Contract traces: (p, σ)

Hardware



Formal model of processor

Hardware traces: (p, σ)

Contract satisfaction

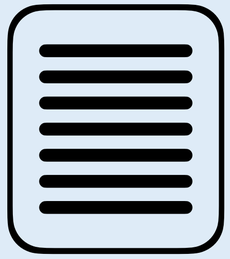
Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$

Contracts

Observations expose security-relevant ***μarch events***

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

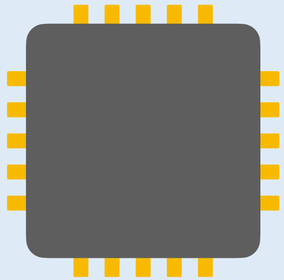
Contract



ISA extended with observations

Contract traces: (p, σ)

Hardware



Formal model of processor

Hardware traces: (p, σ)

Contract satisfaction

Hardware satisfies contract 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 = Execution Mode · Observer Mode

How are programs executed?

Contracts for secure speculation

Contract =

Execution Mode · Observer Mode

How are programs executed?

What is visible about the execution?

Contracts for secure speculation

Contract =

Execution Mode · Observer Mode

seq — sequential execution

spec — mispredict branch instrs.

Contracts for secure speculation

Contract =

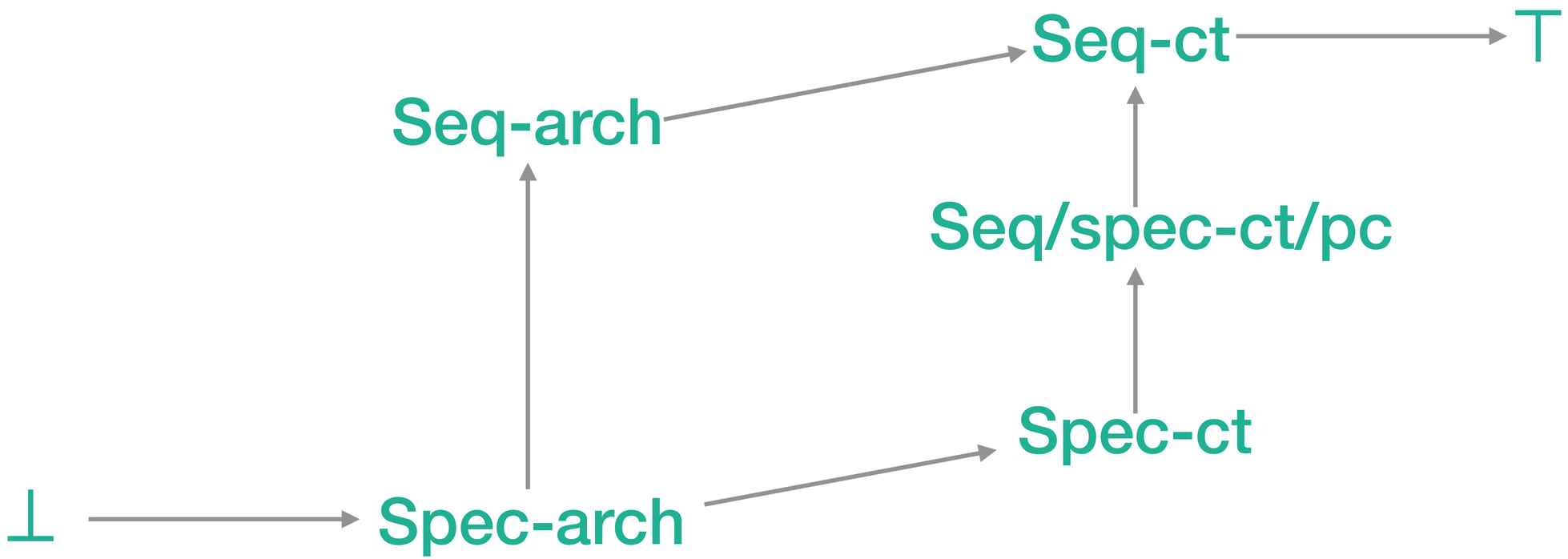
Execution Mode · Observer Mode

pc — only program counter

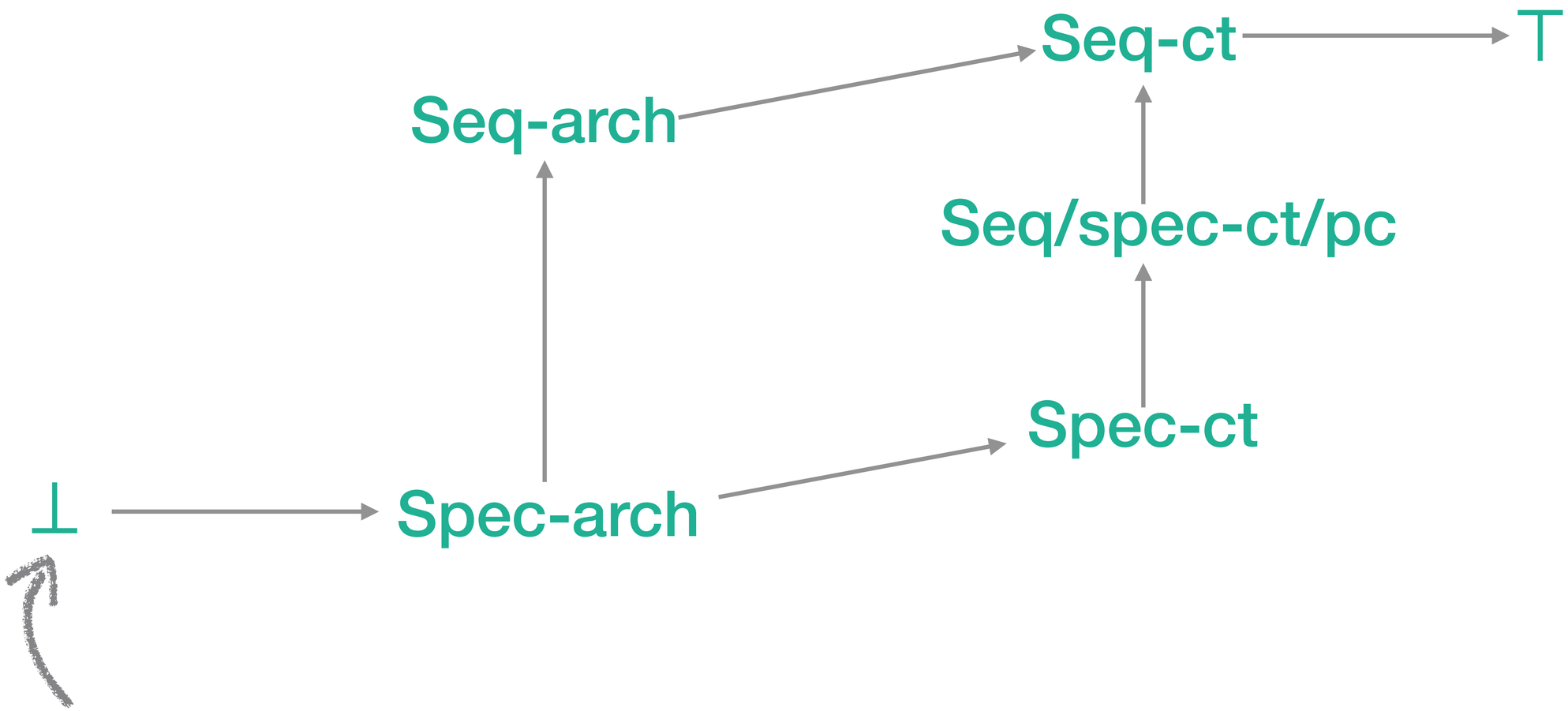
ct — **pc** + addr. of loads and stores

arch — **ct** + loaded values

A lattice of contracts

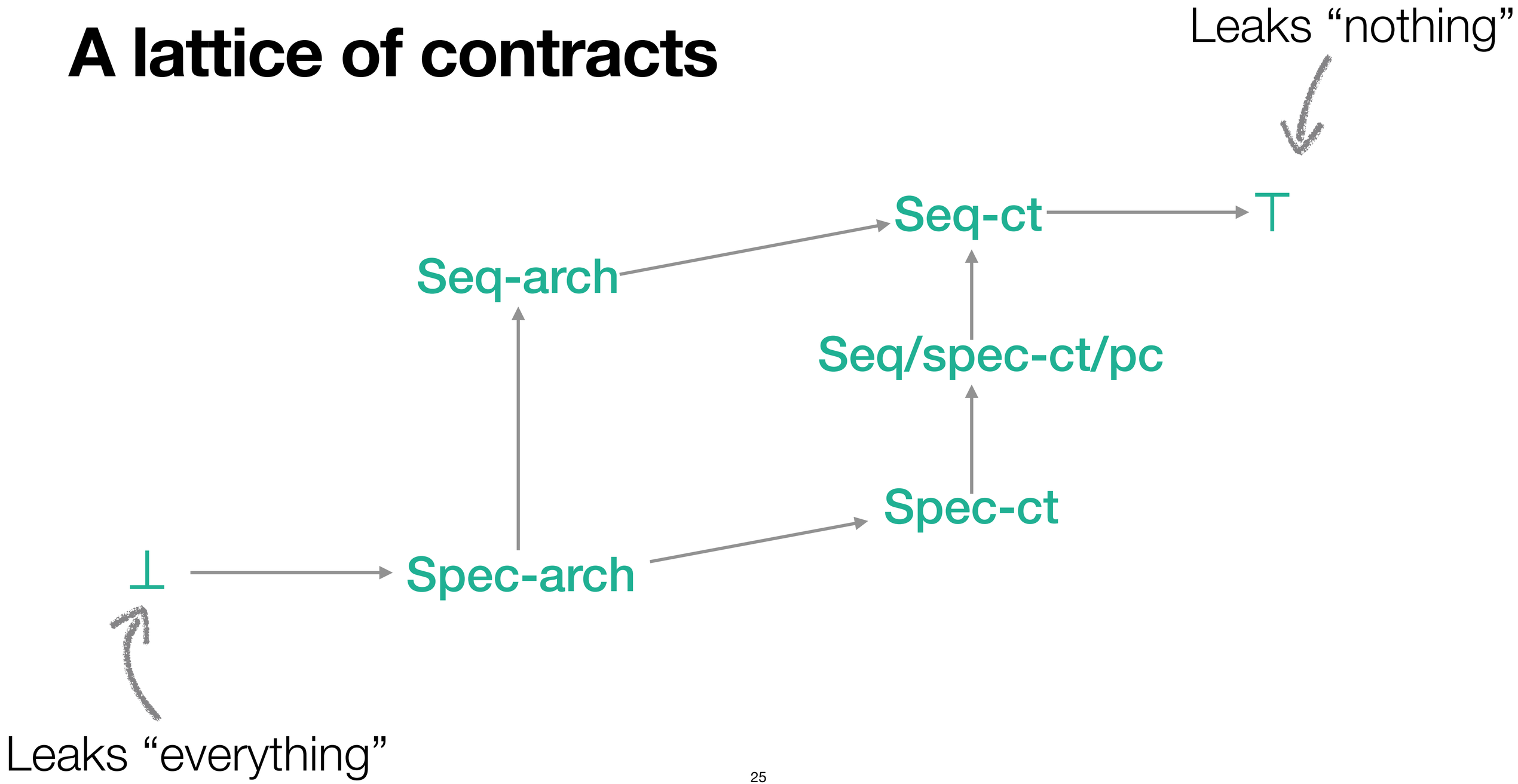


A lattice of contracts

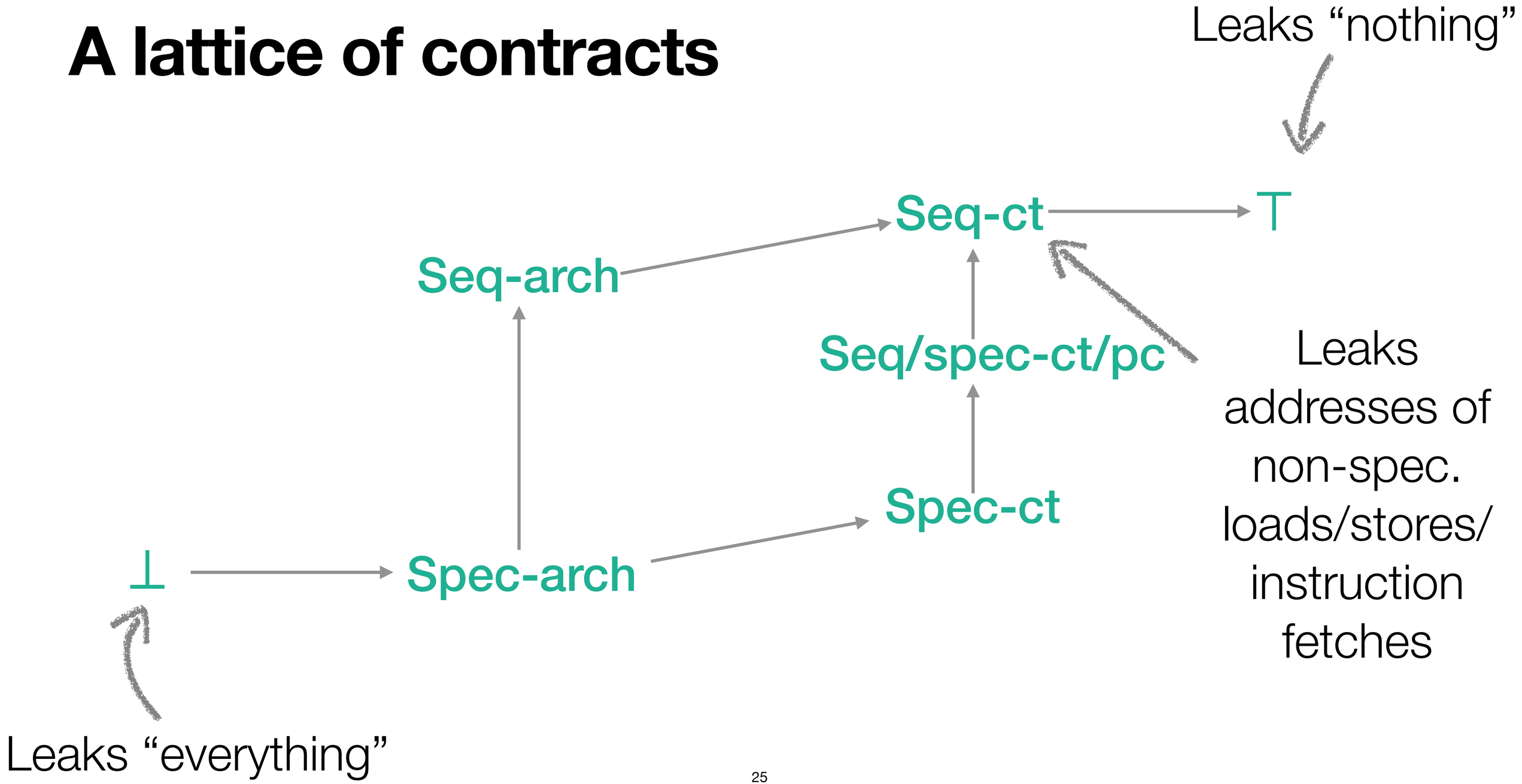


Leaks “everything”

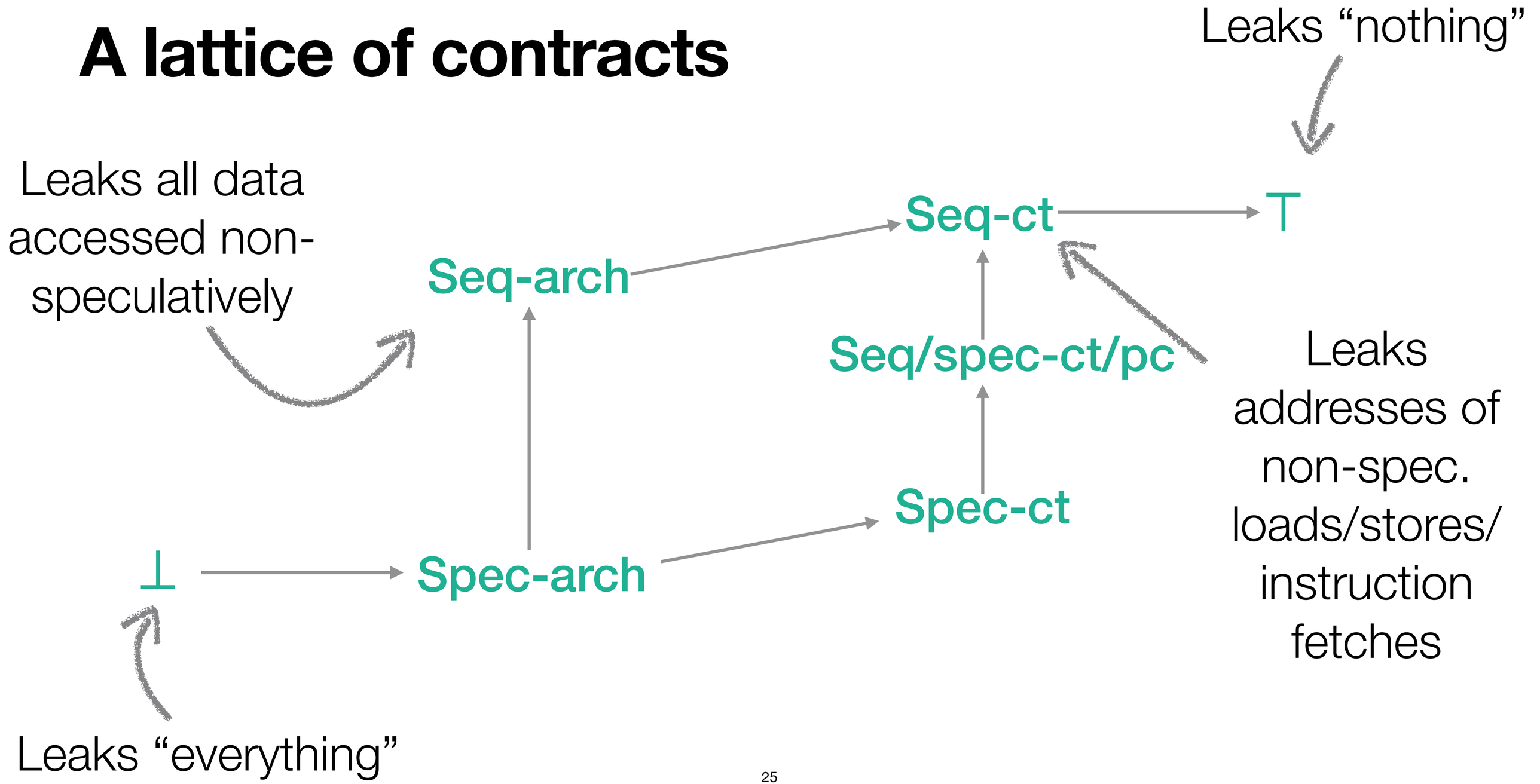
A lattice of contracts



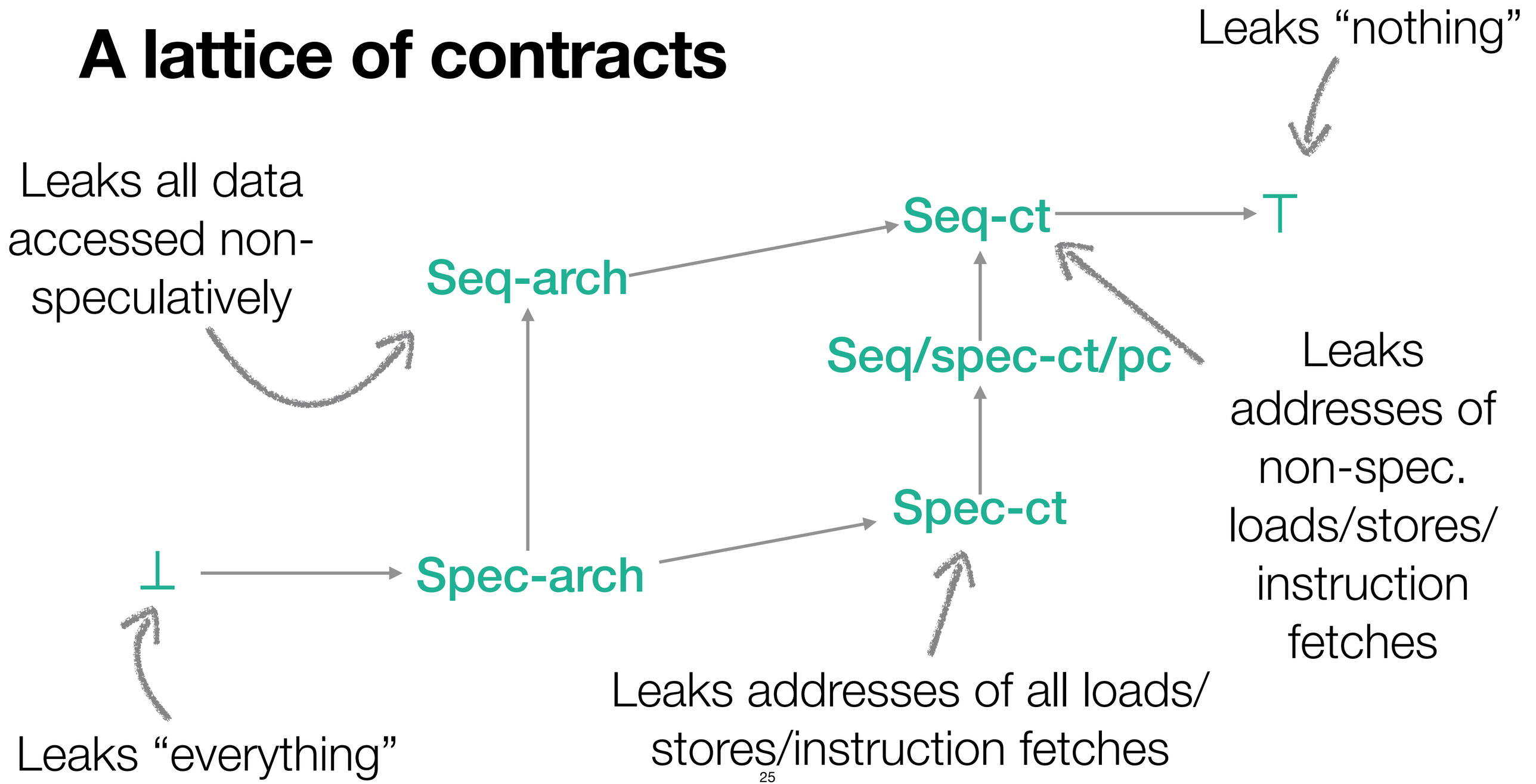
A lattice of contracts



A lattice of contracts



A lattice of contracts



Example: seq-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

Example: seq-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x < A\_size$

Example: seq-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

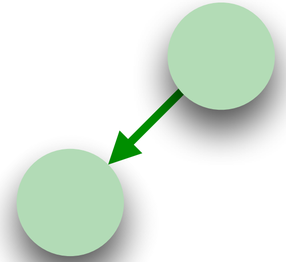
$x < A\_size$

pc 2

Example: seq-ct

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

$x < A\_size$

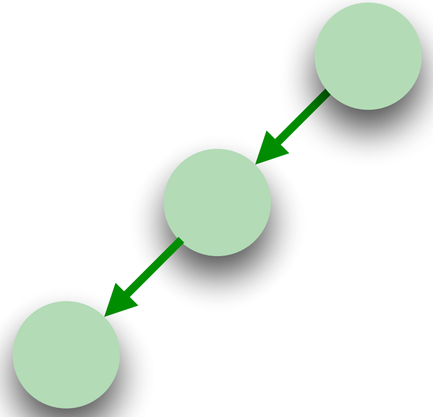


load A+x

Example: seq-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x < A\_size$



load B+A[x]

Example: seq-arch

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

Example: seq-arch

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x < A\_size$

Example: seq-arch

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

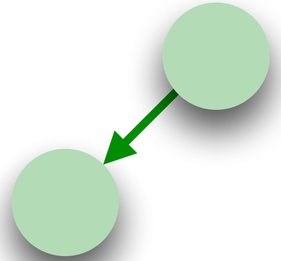
$x < A\_size$

pc *2*

Example: seq-arch

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

$x < A\_size$

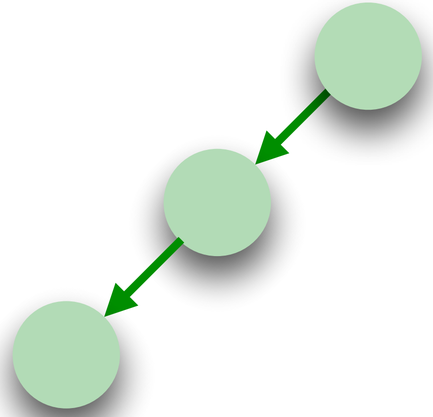


load A+x,A[x]

Example: seq-arch

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x < A\_size$



load B+A[x],B[A[x]]

Example: spec-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

Example: spec-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x > A\_size$

Example: spec-ct

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

$x > A\_size$

Example: spec-ct

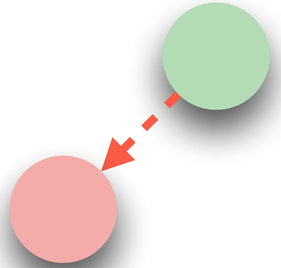
```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x > A\_size$

start
pc 2

Example: spec-ct

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

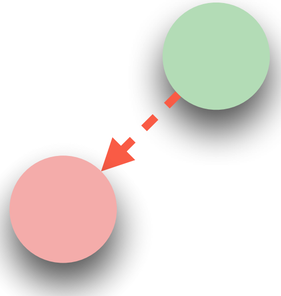


`x > A_size`

Example: spec-ct

```
1.  if (x < A_size)
2.      y = A[x]
3.      z = B[y]
4.  end
```

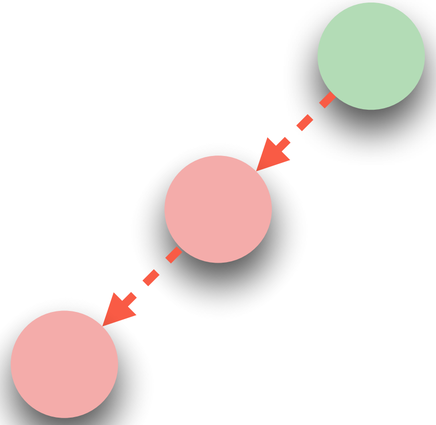
x > A_size



load **A**+**x**

Example: spec-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

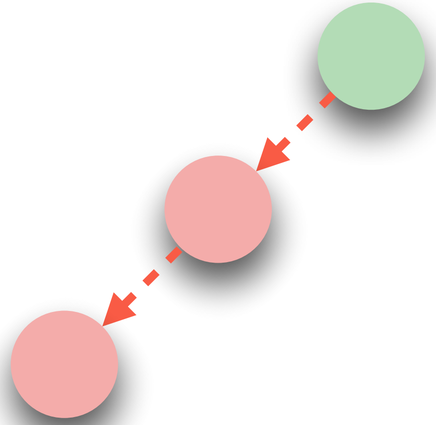


x > A_size

Example: spec-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

$x > A\_size$

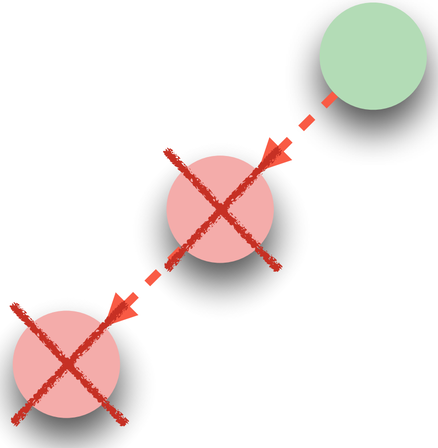


load B+A[x]

Example: spec-ct

```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```

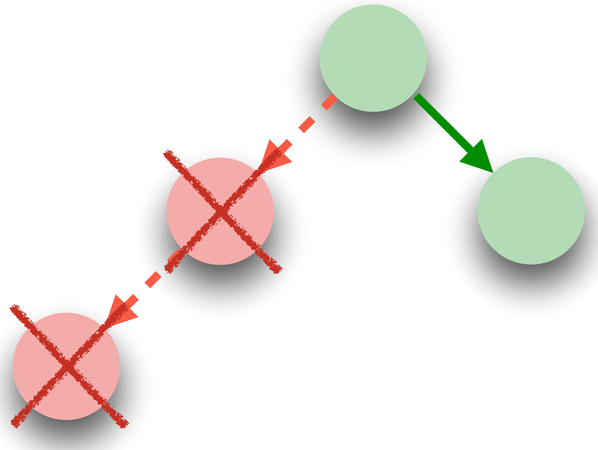
x > A_size



rollback
pc *4*

Example: spec-ct

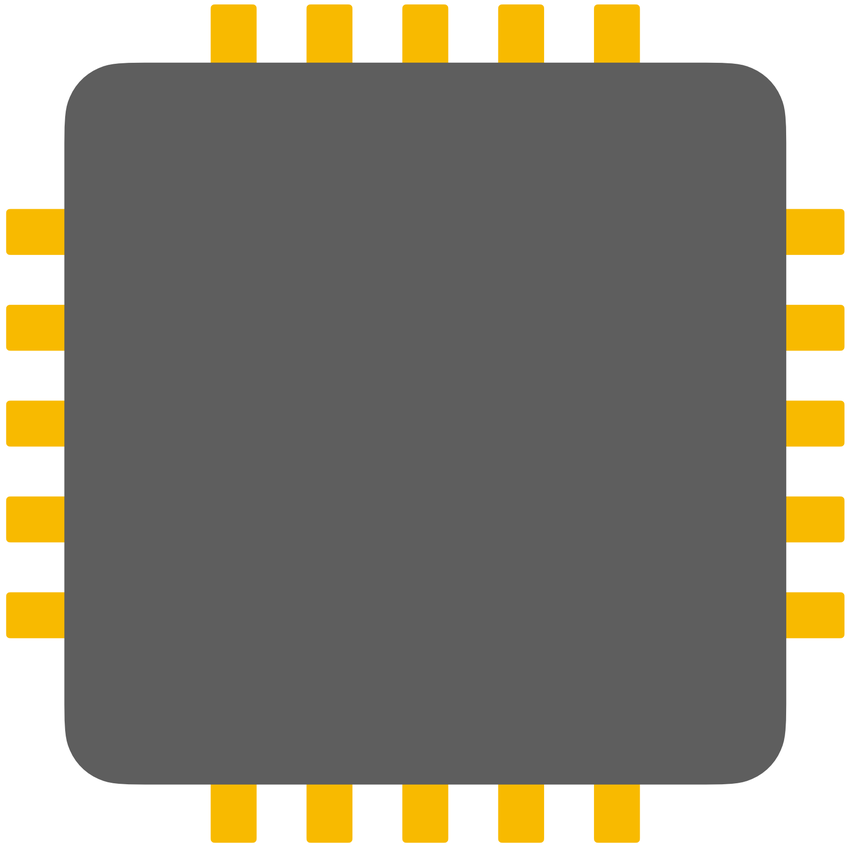
```
1.  if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4.  end
```



$x > A\_size$

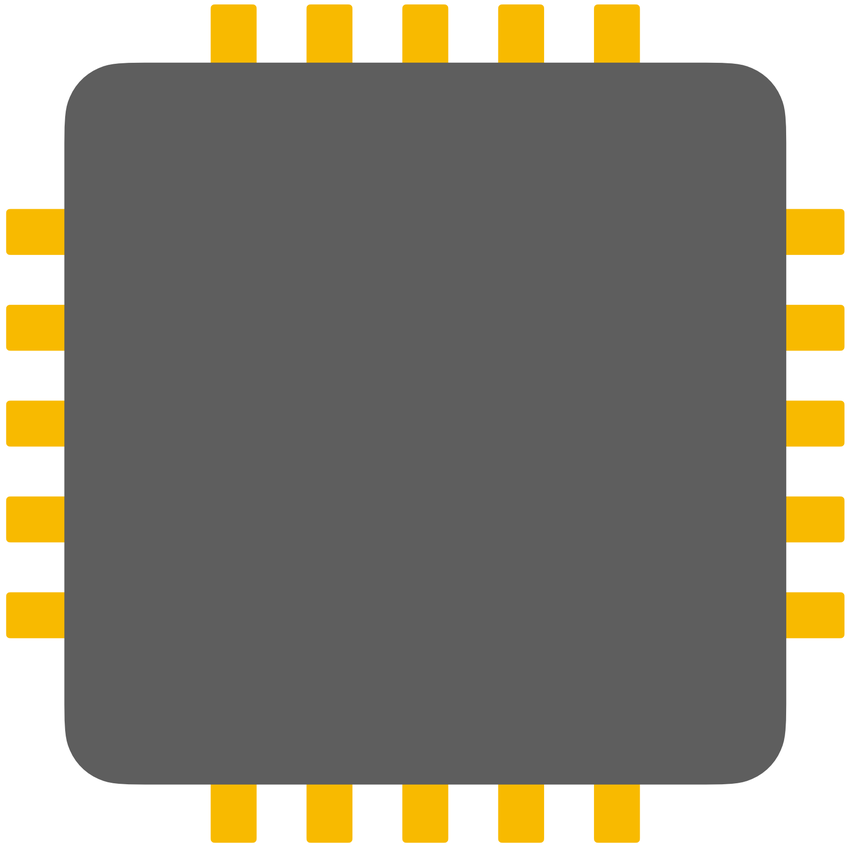
Hardware countermeasures

A simple processor



See paper for: processor operational semantics and security proofs

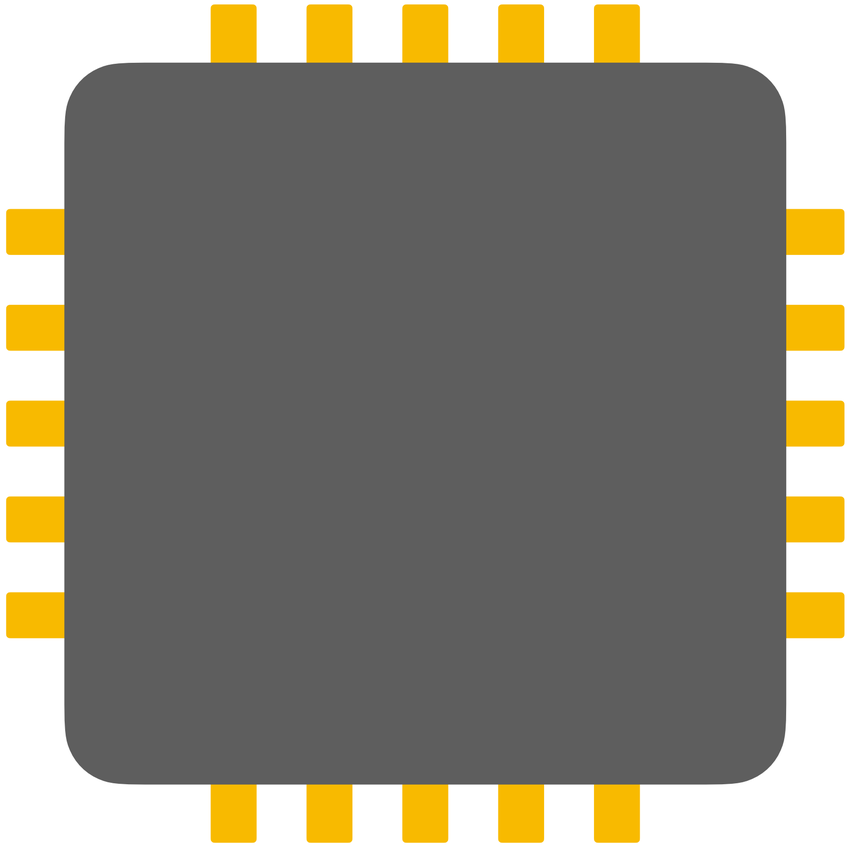
A simple processor



Speculative and ***out-of-order*** executions

See paper for: processor operational semantics and security proofs

A simple processor

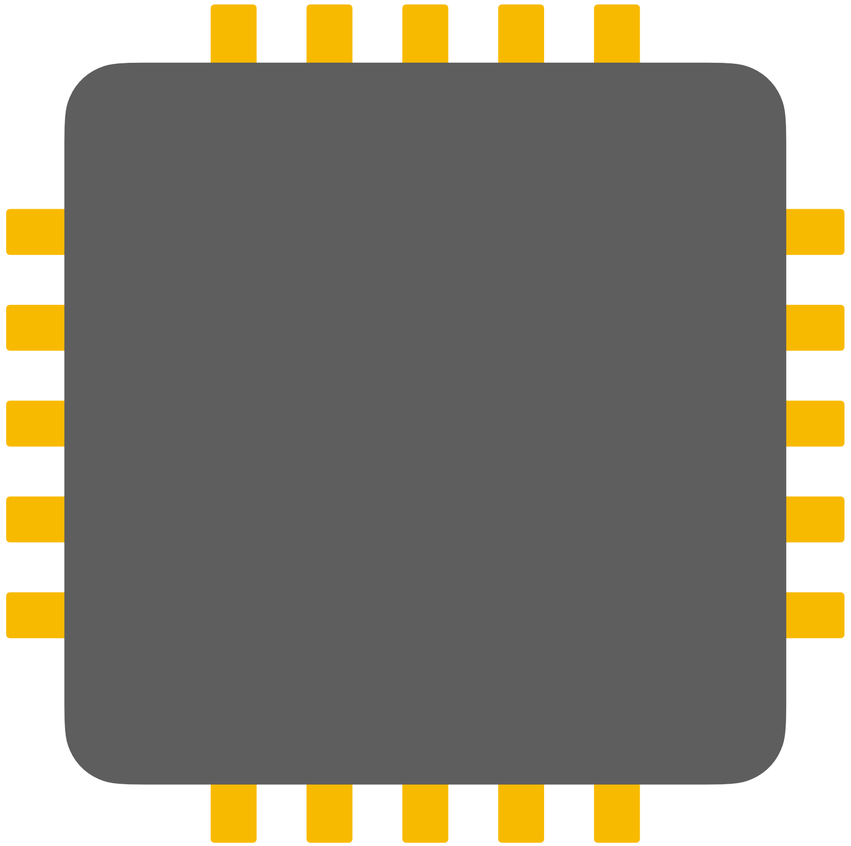


Speculative and ***out-of-order*** executions

3-stage pipeline (fetch, execute, retire)

See paper for: processor operational semantics and security proofs

A simple processor



Speculative and ***out-of-order*** executions

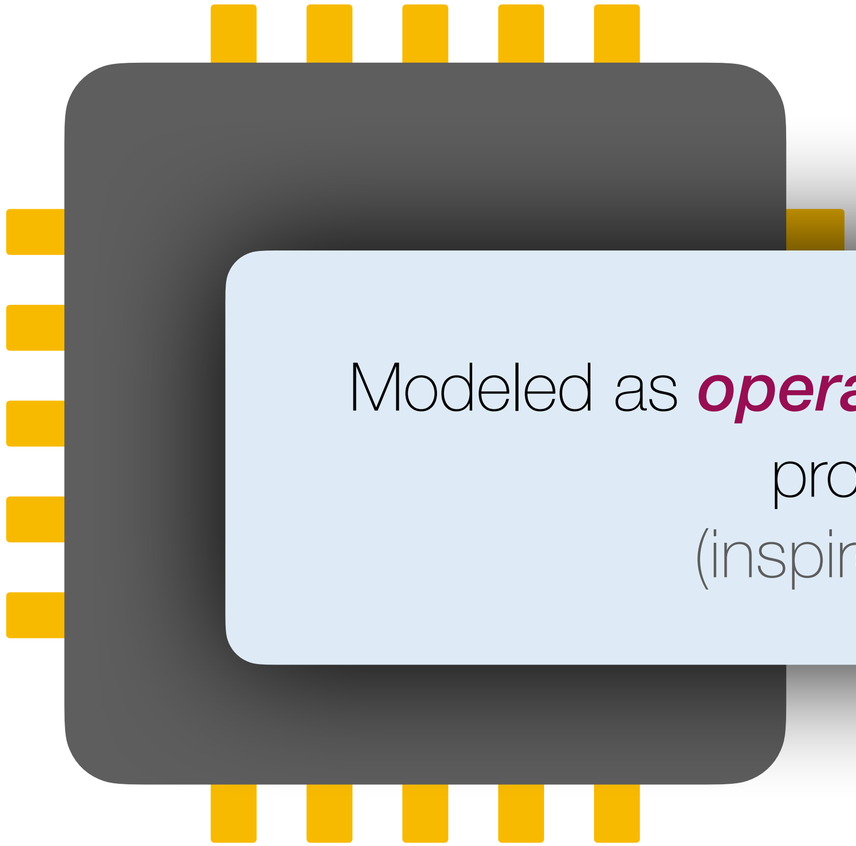
3-stage pipeline (fetch, execute, retire)

Parametric in ***branch predictor*** and ***memory hierarchy***

See paper for: processor operational semantics and security proofs

A simple processor

Speculative and ***out-of-order*** executions

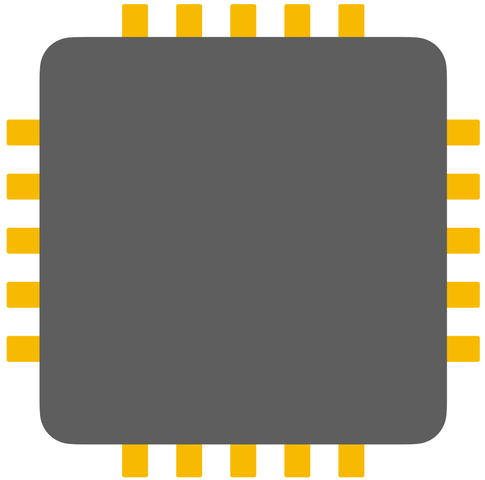


Modeled as ***operational semantics*** ⇒ describing how processor's state changes (inspired by *[Cauligi et al. 2019]*)

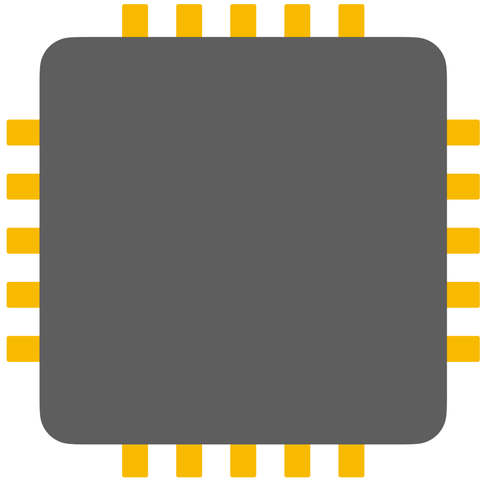
Parametric in ***branch predictor*** and ***memory hierarchy***

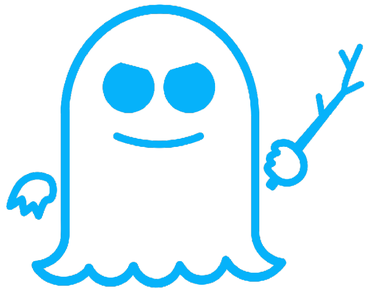
See paper for: processor operational semantics and security proofs

No countermeasures *[The world before Spectre]*



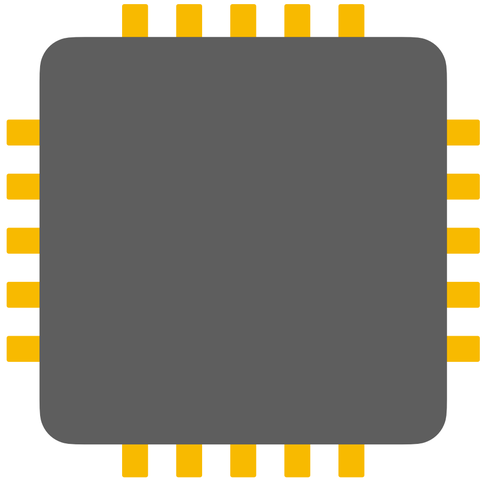
No countermeasures *[The world before Spectre]*

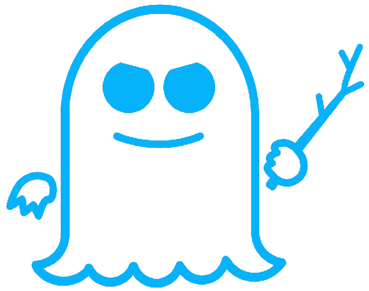




```
if (x < A_size)
    z = A[x]
    y = B[z]
```

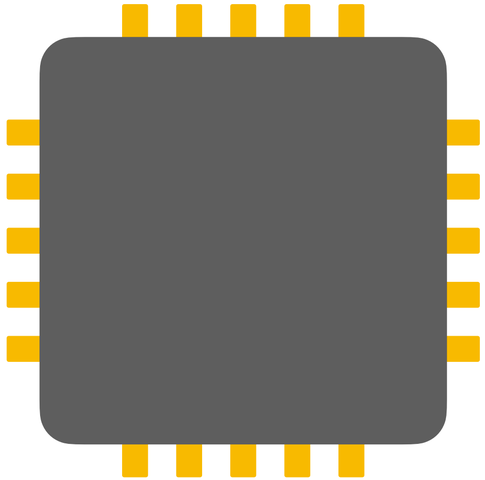
No countermeasures *[The world before Spectre]*

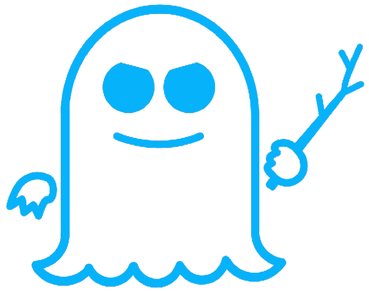




```
if (x < A_size)
    z = A[x]
    y = B[z]
```

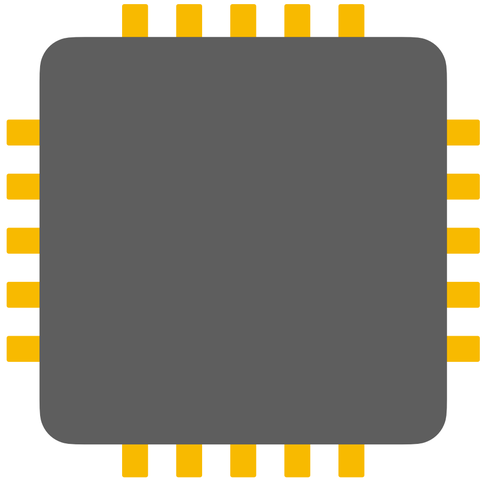
No countermeasures *[The world before Spectre]*

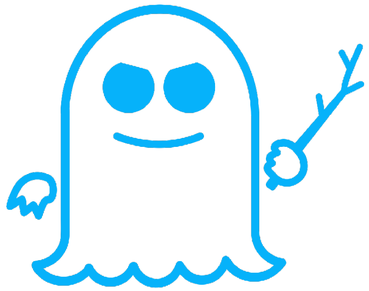




```
if (x < A_size)
    z = A[x]
    y = B[z]
```

No countermeasures *[The world before Spectre]*

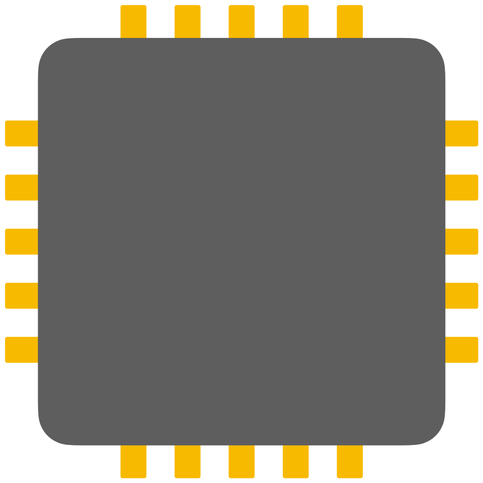




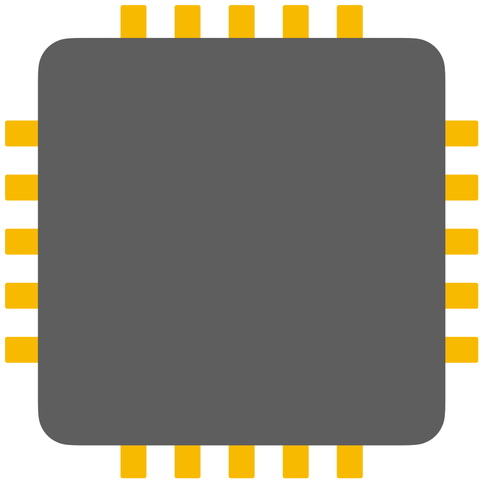
```
if (x < A_size)
    z = A[x]
    y = B[z]
```

Satisfies **spec-ct**

Disabling speculative execution

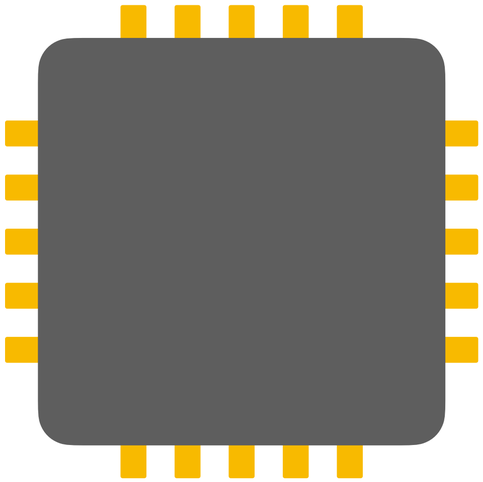


Disabling speculative execution



Instructions are executed ***sequentially***

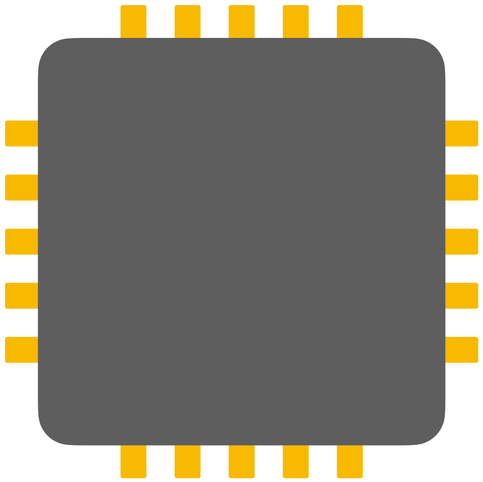
Disabling speculative execution



Instructions are executed ***sequentially***

🥳 No speculative leaks 🥳

Disabling speculative execution

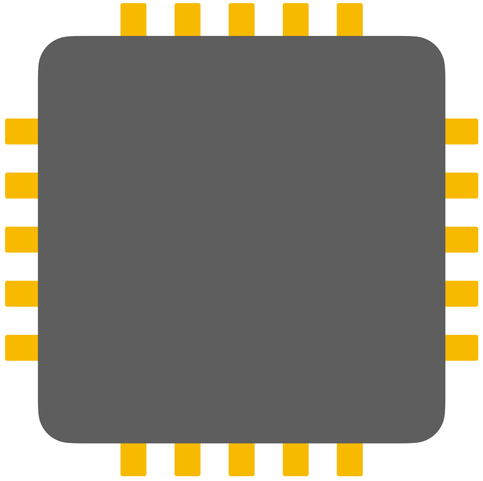


Instructions are executed ***sequentially***

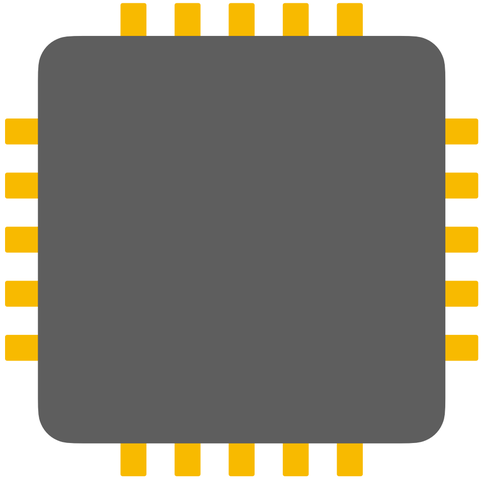
🥳 No speculative leaks 🥳

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)
  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

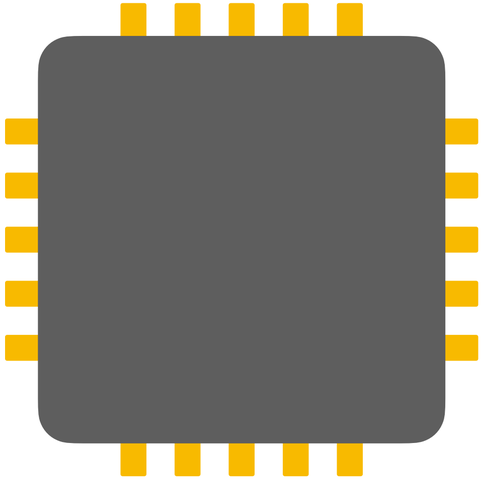
Eager load delay *[Sakalis et al., ISCA'19]*

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

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

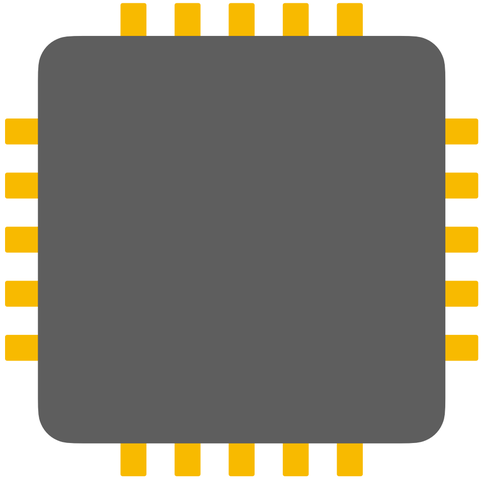
Program speculatively leaks **A**[***x***] 😞

Eager load delay *[Sakalis et al., ISCA'19]*



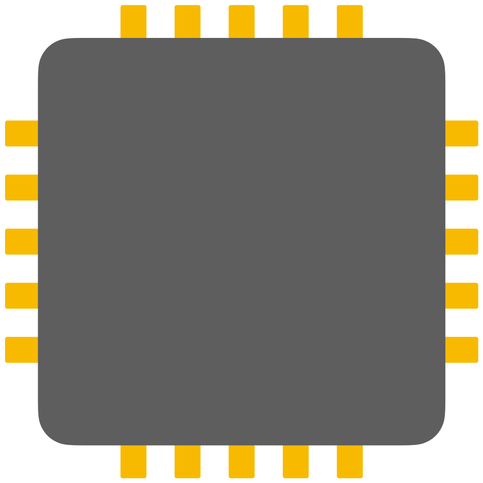
Eager load delay *[Sakalis et al., ISCA'19]*

Satisfies **seq-arch**

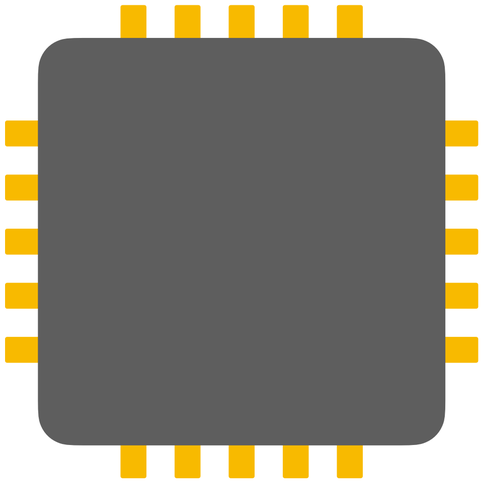


Satisfies **seq/spec-ct/pc**

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

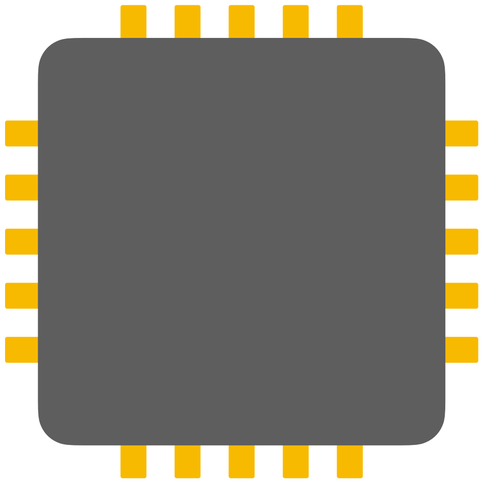


Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*



Taint speculatively loaded data

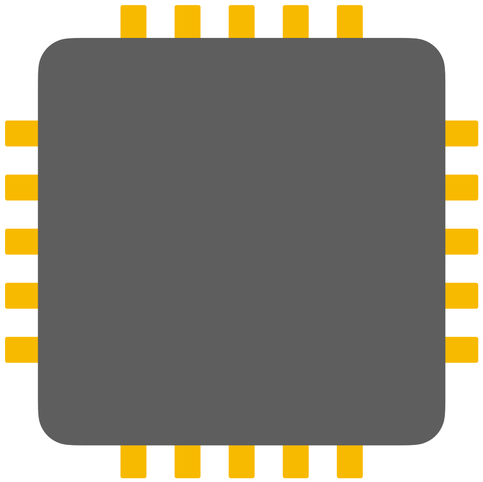
Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*



Taint speculatively loaded data

Propagate taint through computation

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*



Taint speculatively loaded data

Propagate taint through computation

Delay tainted operations

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

Taint speculatively loaded data

Security guarantees?

Delay tainted operations

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

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

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

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

Hardware 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

Hardware 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 🥳

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

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

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

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

Hardware 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***

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

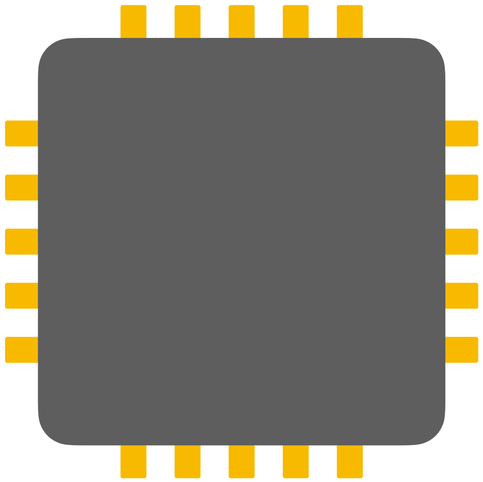
```
z = A[x]
if (x < A_size)
    y = B[z]
```

A[***x***] tagged as ***safe***

B[***z***] ***not delayed***

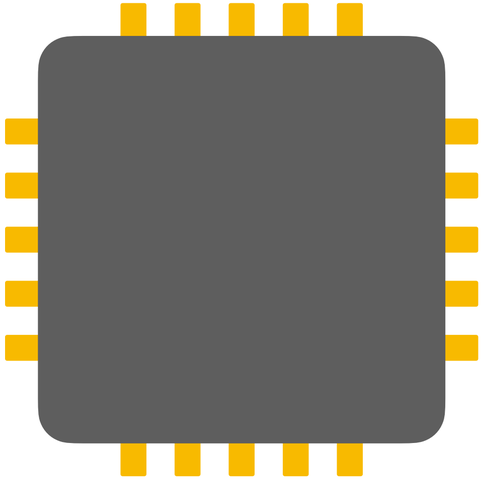
Program speculatively leaks **A**[***x***] 😞

Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*



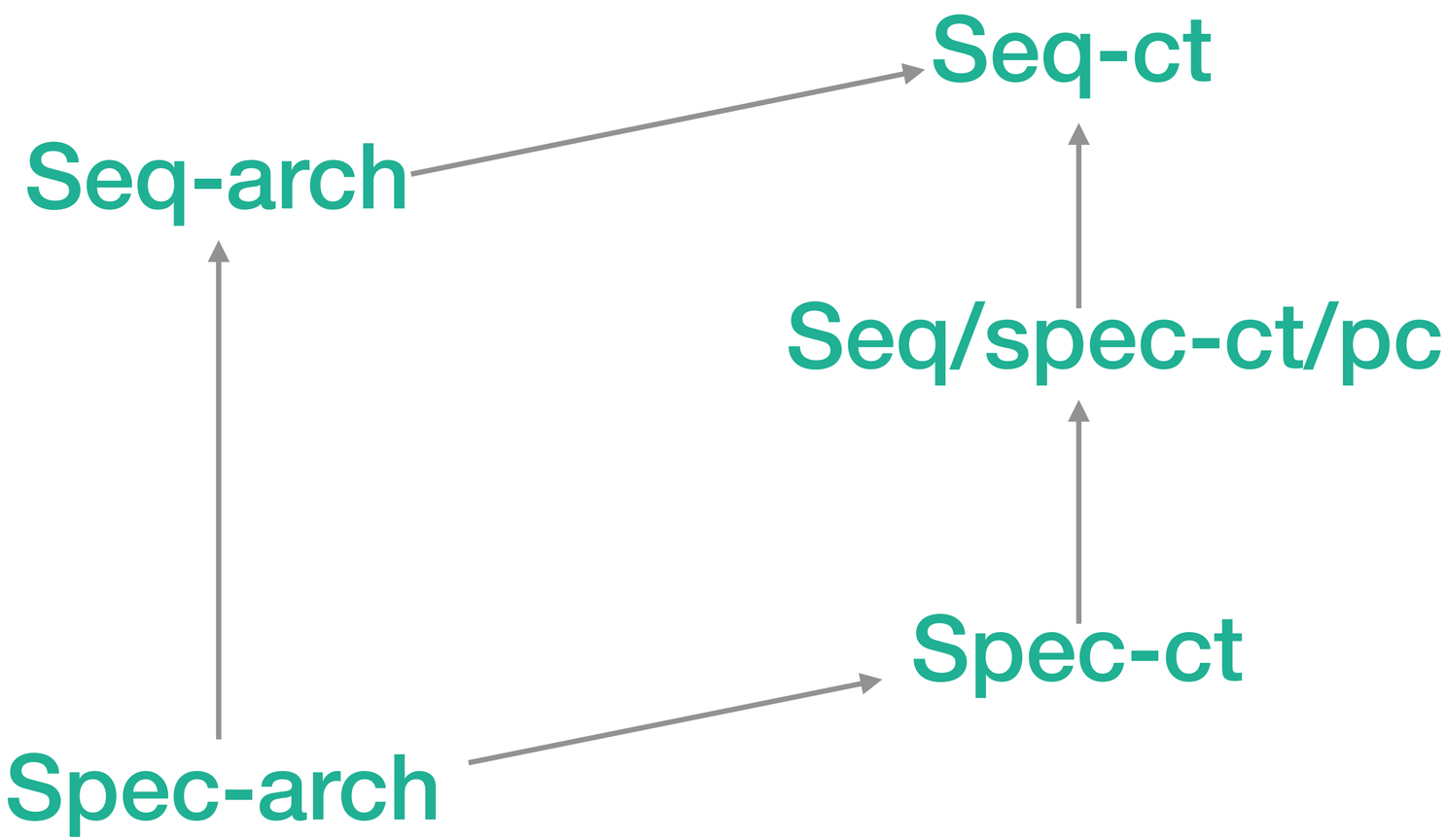
Hardware taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

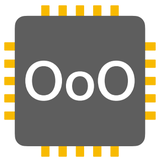
Satisfies **seq-arch**



Satisfies **spec-ct**

Security guarantees

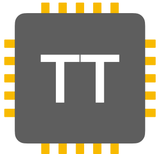




Vanilla OoO CPU + spec. exec

In-order CPU (no specExec)

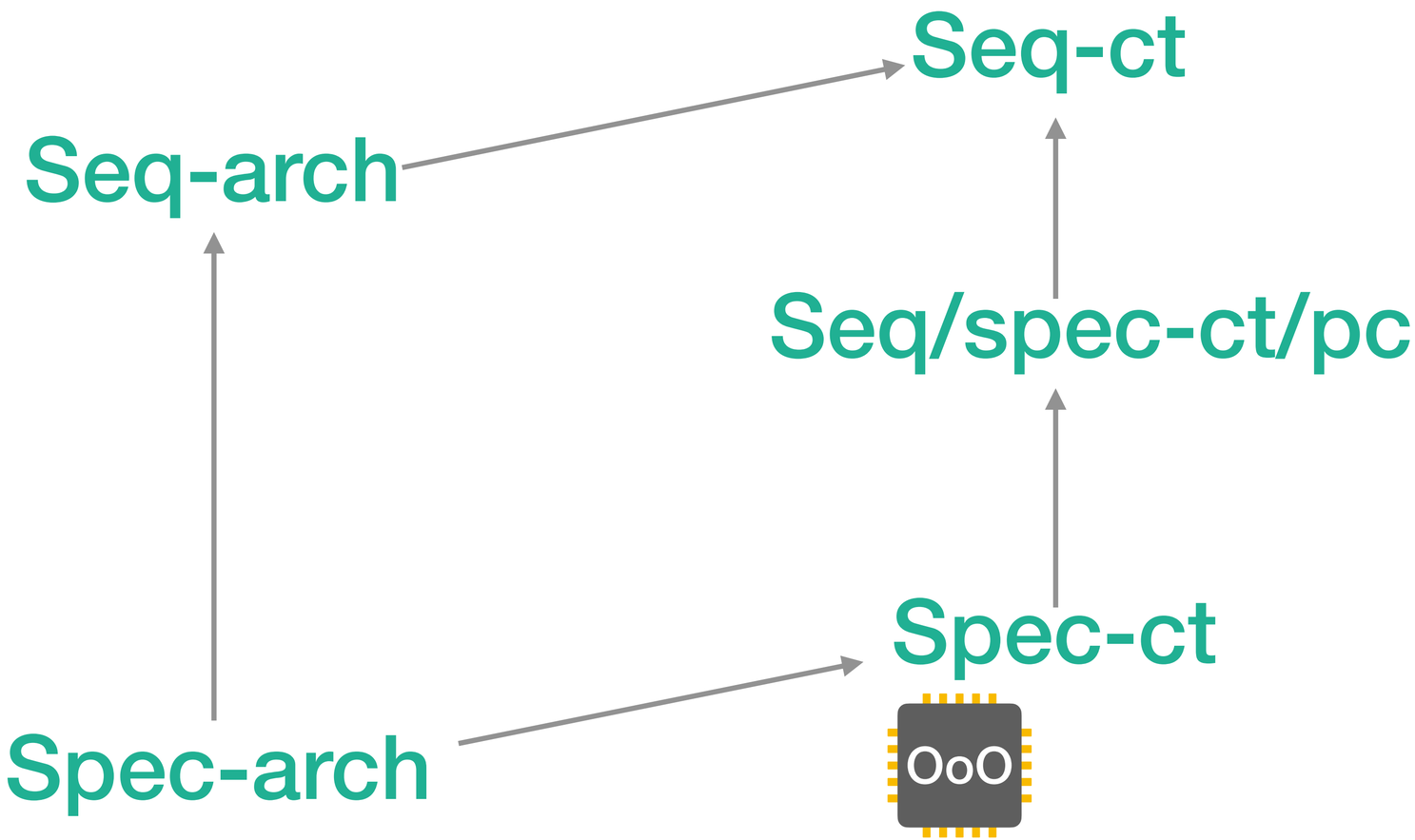
OoO CPU+load delay



OoO CPU+taint tracking

See paper for: models and security proofs

Security guarantees



OoO Vanilla OoO CPU + spec. exec

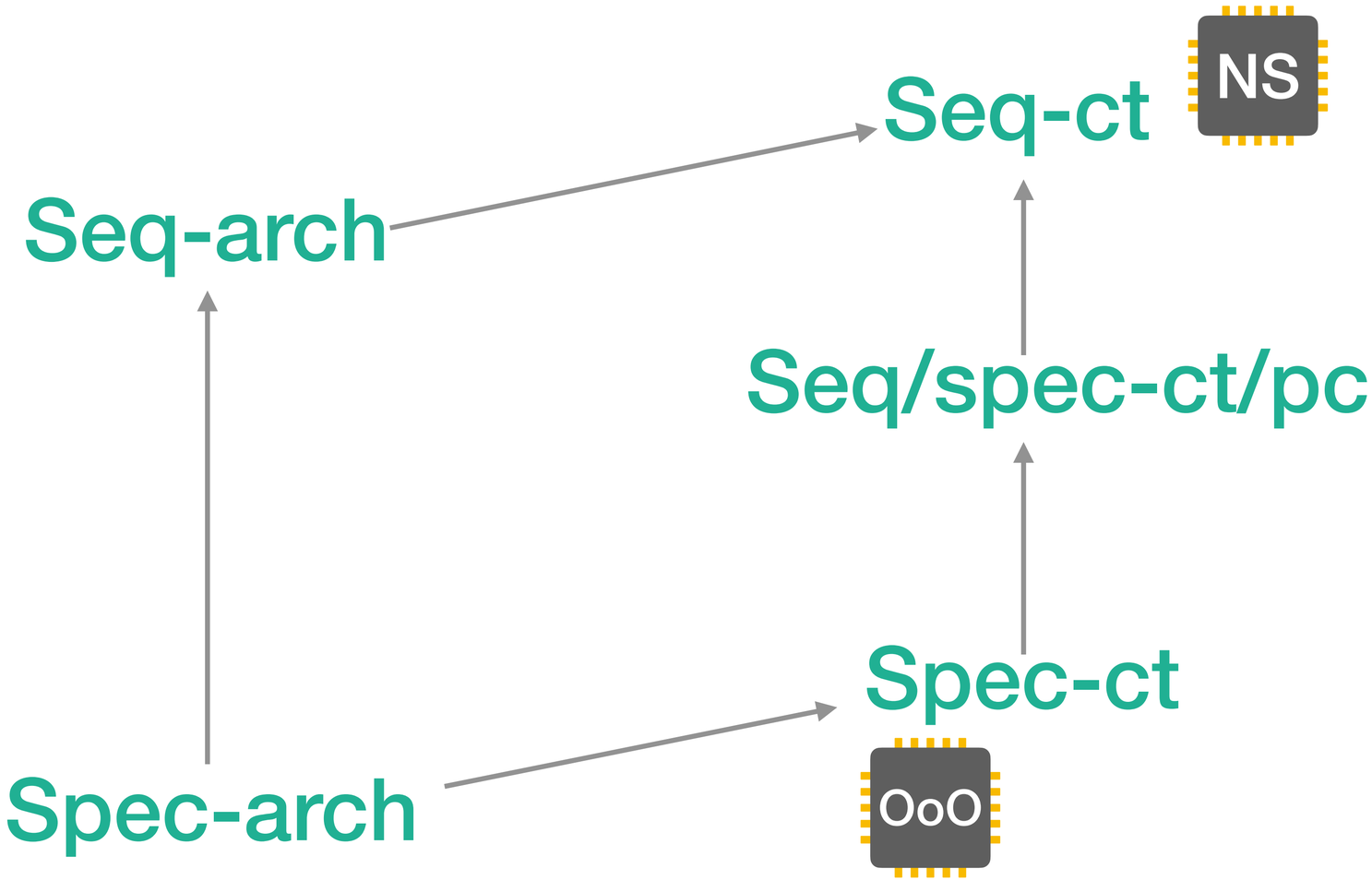
NS In-order CPU (no specExec)

LD OoO CPU+load delay

TT OoO CPU+taint tracking

See paper for: models and security proofs

Security guarantees



OoO — Vanilla OoO CPU + spec. exec

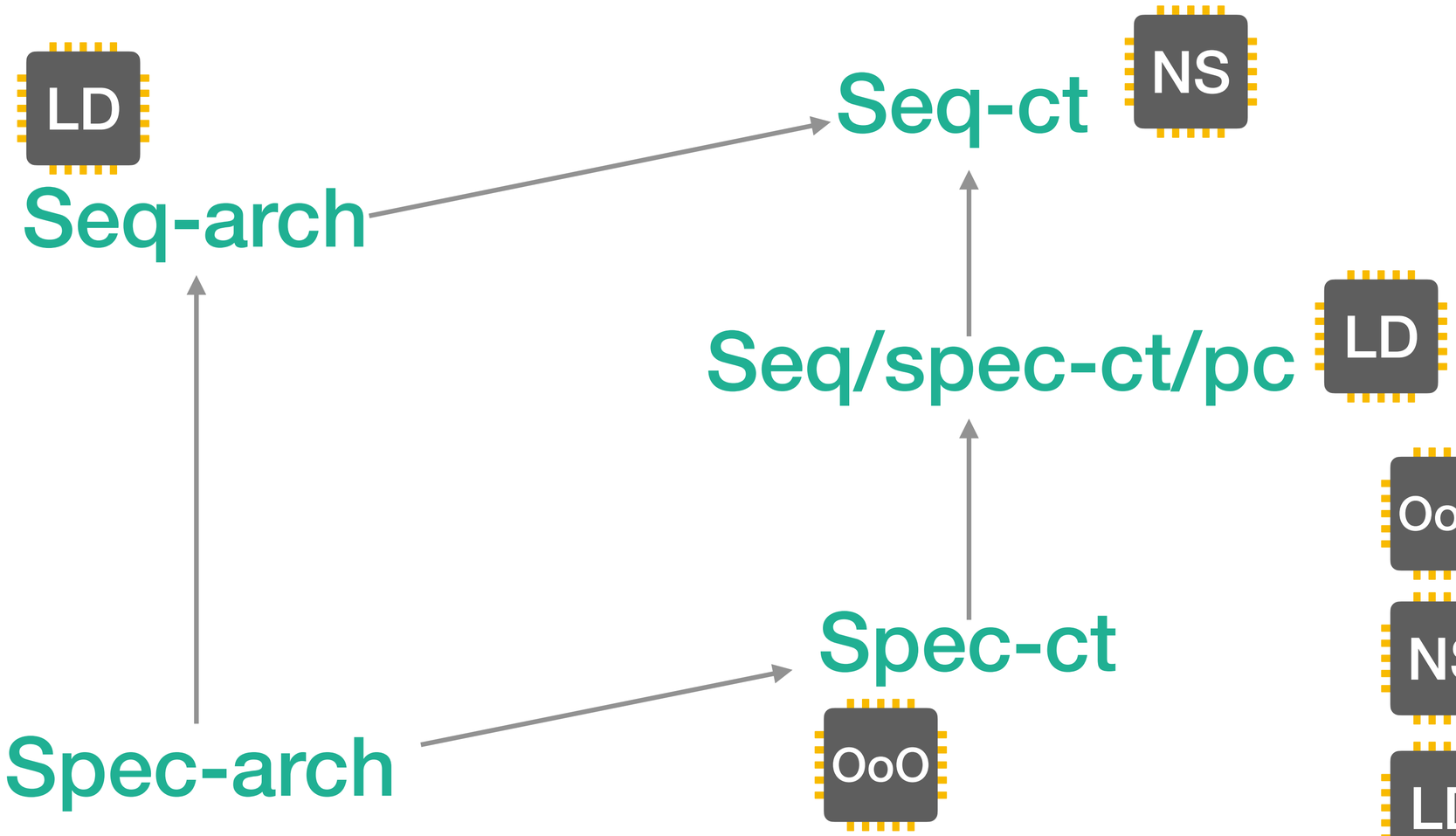
NS — In-order CPU (no specExec)

LD — OoO CPU+load delay

TT — OoO CPU+taint tracking

See paper for: models and security proofs

Security guarantees



OoO — Vanilla OoO CPU + spec. exec

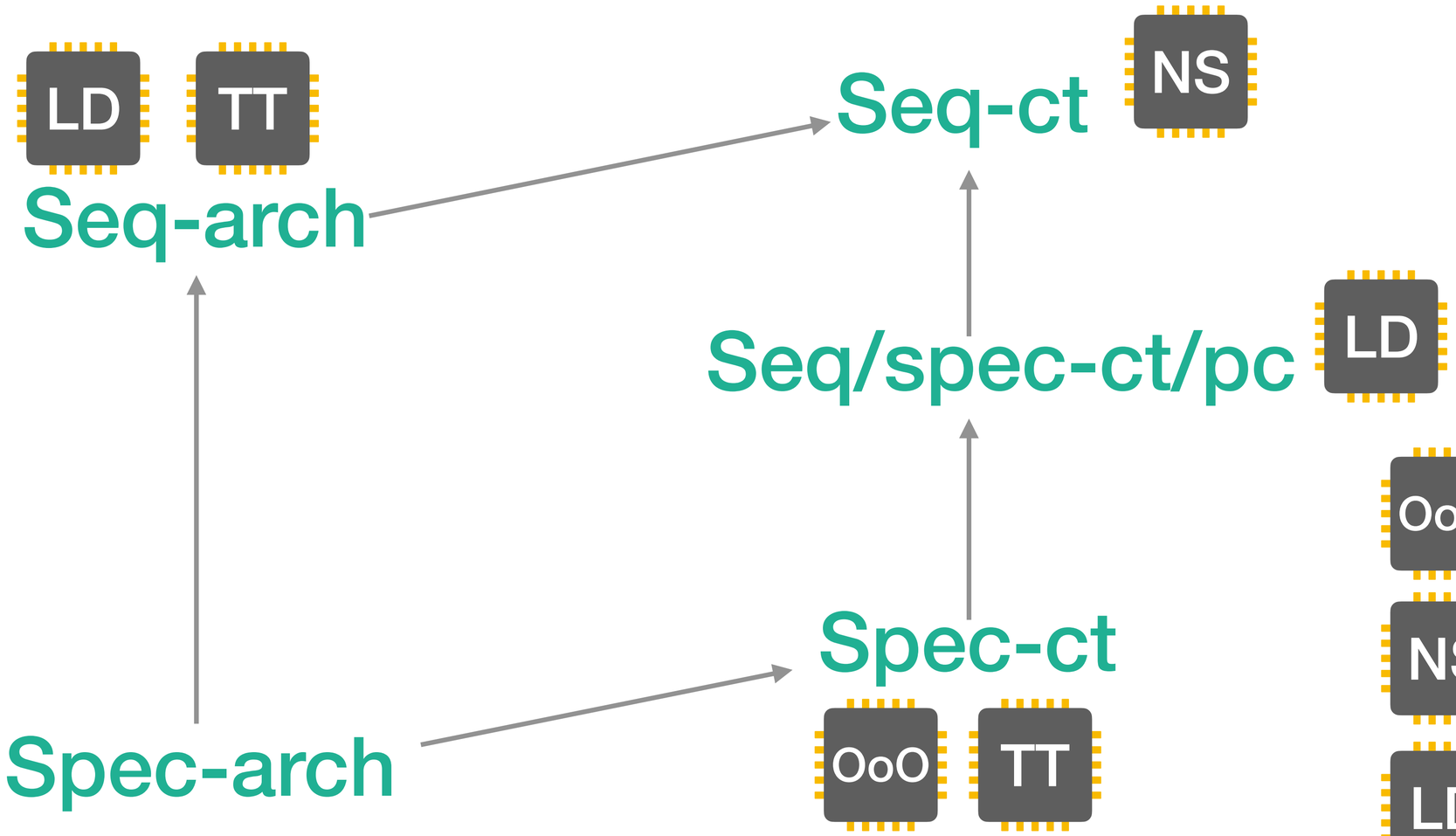
NS — In-order CPU (no specExec)

LD — OoO CPU+load delay

TT — OoO CPU+taint tracking

See paper for: models and security proofs

Security guarantees



OoO — Vanilla OoO CPU + spec. exec

NS — In-order CPU (no specExec)

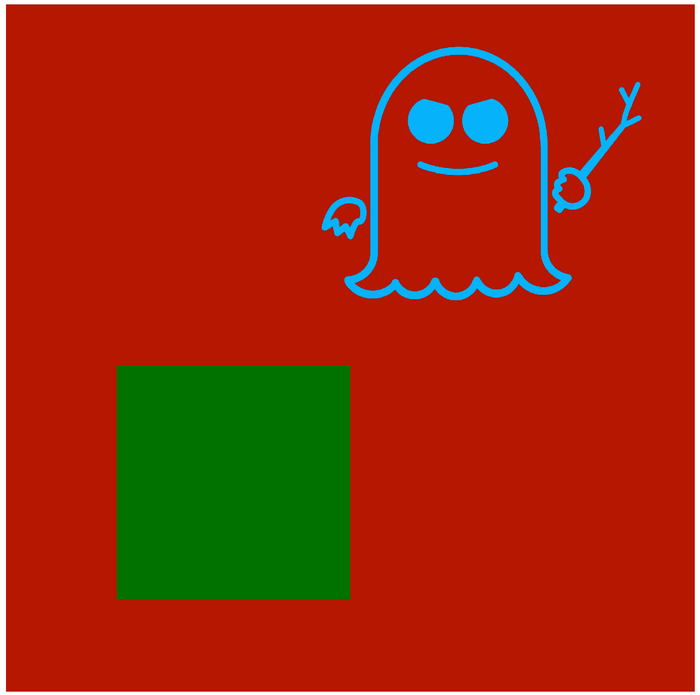
LD — OoO CPU+load delay

TT — OoO CPU+taint tracking

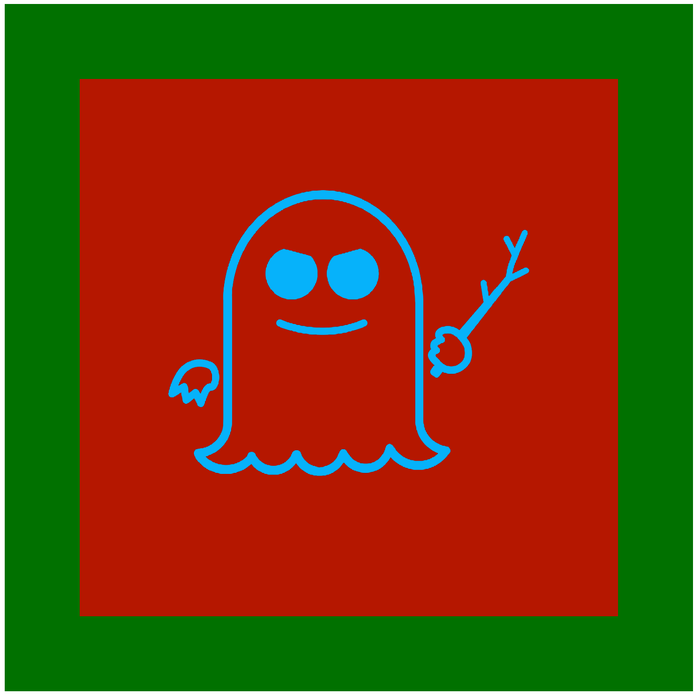
See paper for: models and security proofs

Secure programming

Two flavors of secure programming

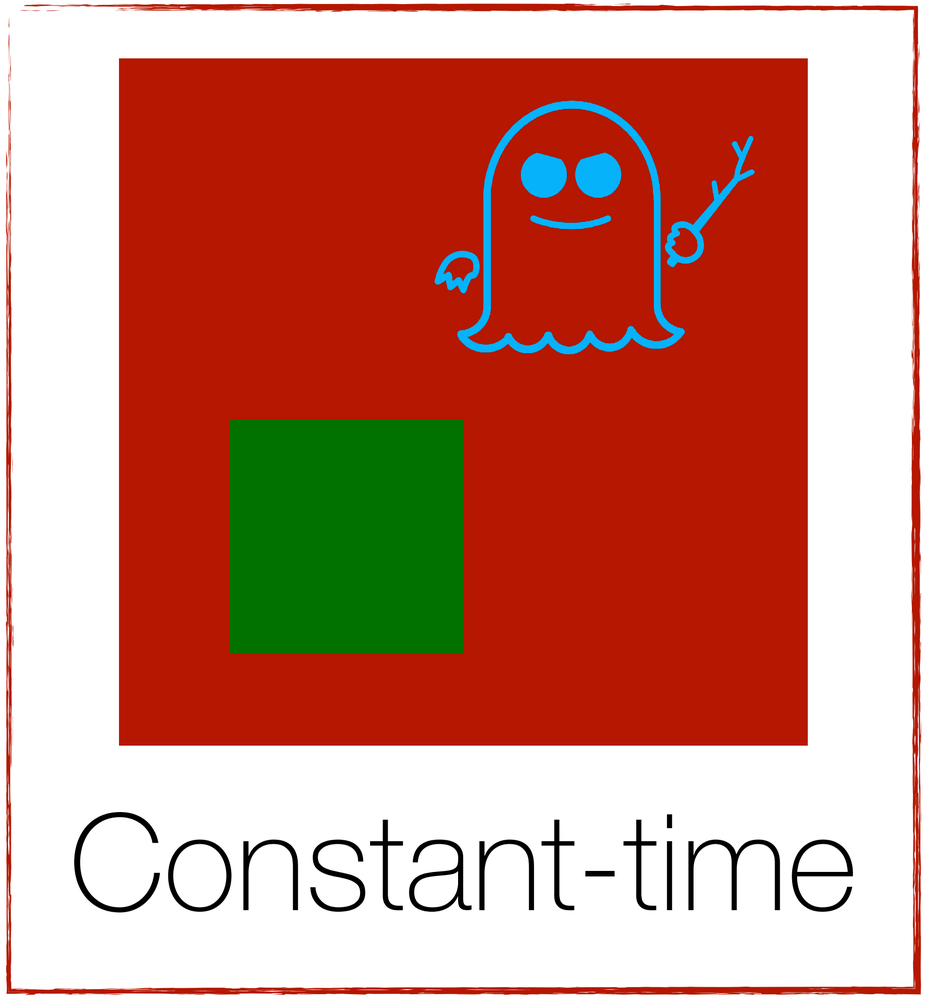


Constant-time

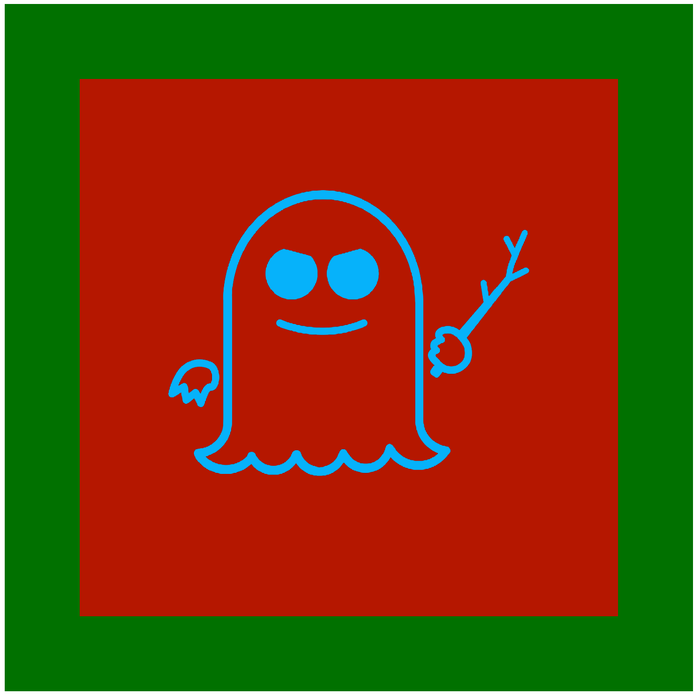


Sandboxing

Two flavors of secure programming

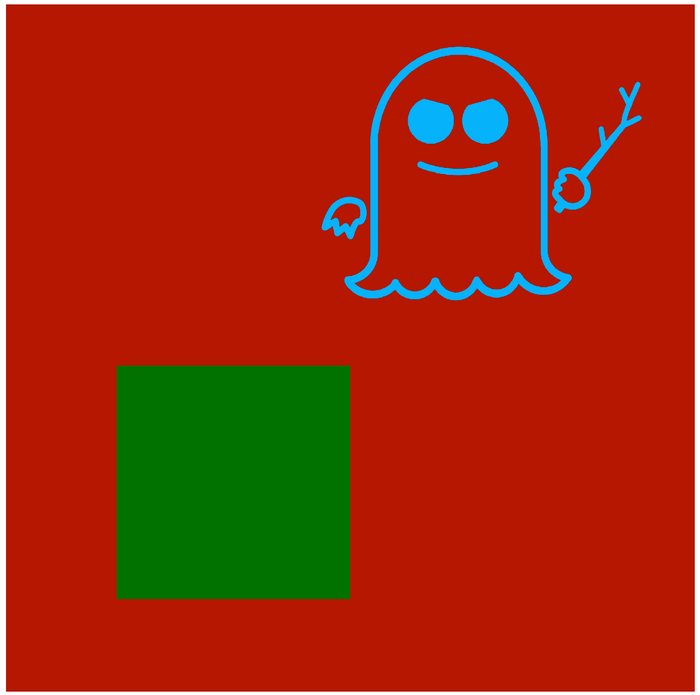


Constant-time

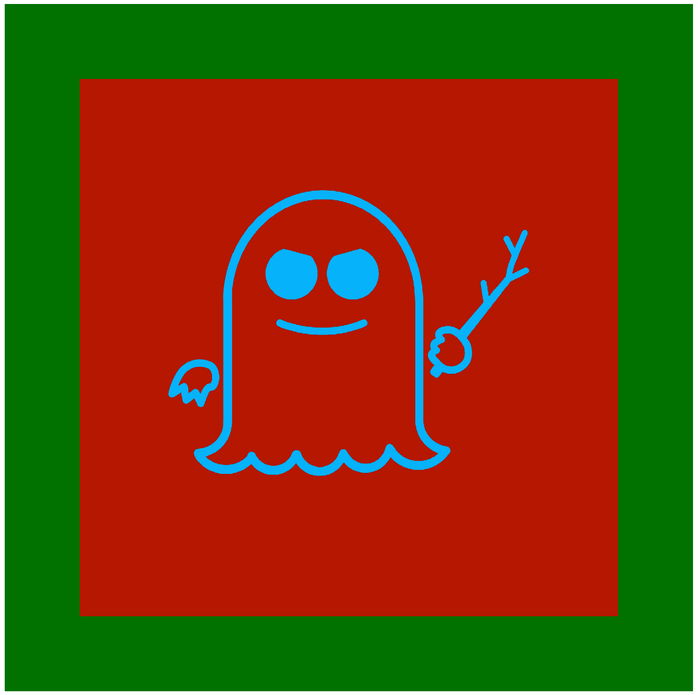


Sandboxing

Two flavors of secure programming



Constant-time



Sandboxing

Checking secure programming

	Constant-time	*Sandboxing*
Seq-ct	✓	✓
Seq-arch	✗	✓
Spec-ct	✗	✗

See paper for: additional security checks, proofs, automation

Checking secure programming

	Constant-time	*Sandboxing*
Seq-ct	✓	✓
Seq-arch	✗	✓
Spec-ct	✗	✗

See paper for: additional security checks, proofs, automation

What’s next?

Hardware

Software

Hw-Sw security contracts

Modeling and specifications

Hardware

Software

Hw-Sw security contracts

Modeling and specifications

Contracts & Modeling

Contracts & Modeling

So far: toy ISA + observation modes for cache/control-flow leaks + execution modes for branch speculation

Contracts & Modeling

So far: toy ISA + observation modes for cache/control-flow leaks + execution modes for branch speculation

Challenge: scale to real-world ISAs

Contracts & Modeling

So far: toy ISA + observation modes for cache/control-flow leaks + execution modes for branch speculation

Challenge: scale to real-world ISAs

What do we need? Formal models of ISAs + adequate extensions to capture leaks

Hardware

Software

Hw-Sw security contracts

Modeling and specifications

Software

Modeling and specifications

Contracts & Hardware

Contracts & Hardware

So far: manual security proofs on toy CPU/ISA

Contracts & Hardware

So far: manual security proofs on toy CPU/ISA

Challenge: automation for realistic CPUs

Contracts & Hardware

So far: manual security proofs on toy CPU/ISA

Challenge: automation for realistic CPUs

What do we need? Automated verification and testing techniques for contract satisfaction

Hardware

Software

Hw-Sw security contracts

Modeling and specifications

Hardware

Software

Hw-Sw security contracts

Modeling and specifications

Contracts & Software

Contracts & Software

So far: contracts as secure programming foundations + tools targeting “fixed contracts”

Contracts & Software

So far: contracts as secure programming foundations + tools targeting “fixed contracts”

Challenge: support for large classes of contracts

Contracts & Software

So far: contracts as secure programming foundations + tools targeting “fixed contracts”

Challenge: support for large classes of contracts

What do we need? **Contract-aware** program analyses and secure compilation passes

Conclusions

Contracts

Contract
ISA extended with observations

Contract traces: $\square(p,\sigma)$

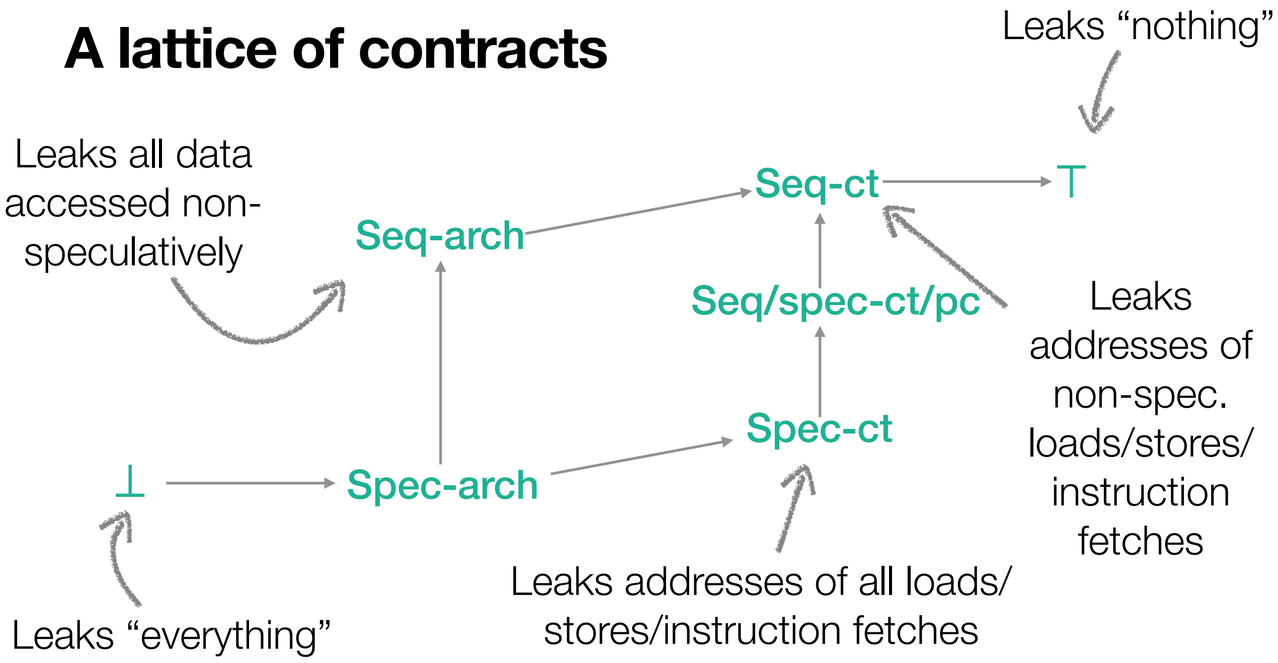
Hardware
Formal model of processor

Hardware traces: $\blacksquare(p,\sigma)$

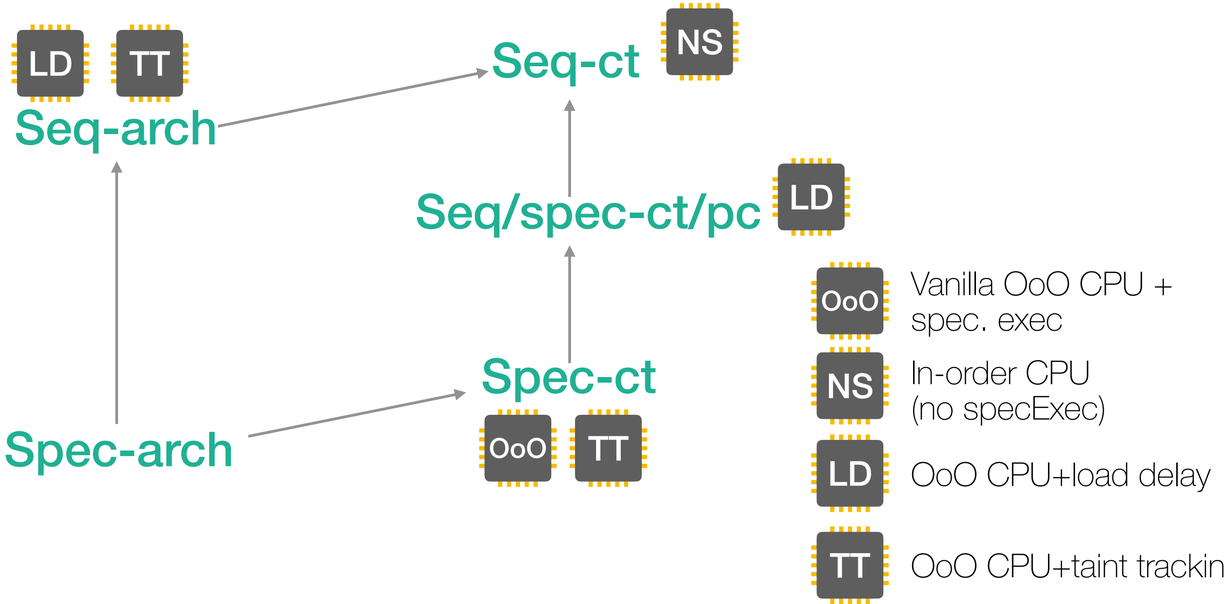
Contract satisfaction

Hardware ■ satisfies contract □ if for all programs p and arch. states σ, σ': if $\square(p,\sigma)=\square(p,\sigma')$ then $\blacksquare(p,\sigma)=\blacksquare(p,\sigma')$

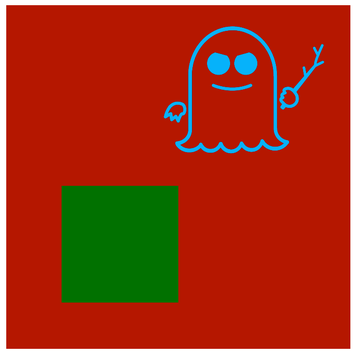
A lattice of contracts



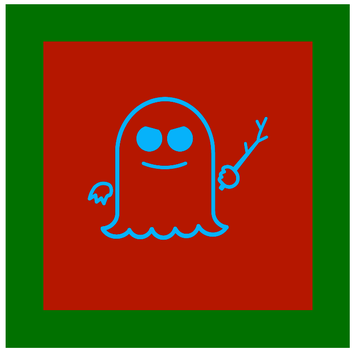
Security guarantees



Two flavors of secure programming



Constant-time



Sandboxing

Contracts

Contract
ISA extended with observations

Contract traces

Hardware
Formal model of processor

Hardware sat
states σ, σ': if

A lattice of contracts

Leaks all data accessed non-speculatively

Leaks "nothing"

Seq-arch → Seq-ct → T

t/pc

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

loads/
ches

Find out more in the paper:
https://arxiv.org/abs/2006.03841

@ *marco.guarnieri@imdea.org*

@MarcoGuarnier1

Security gua

ing

LD TT
Seq-arch

Spec-arch

Spec-ct
OoO TT

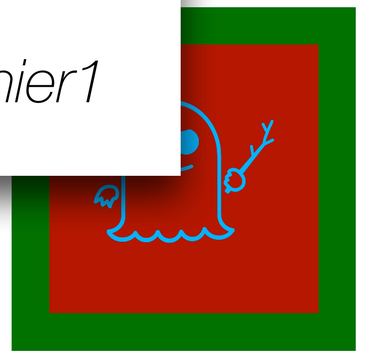
OoO Vanilla OoO CPU + spec. exec

NS In-order CPU (no specExec)

LD OoO CPU+load delay

TT OoO CPU+taint trackin

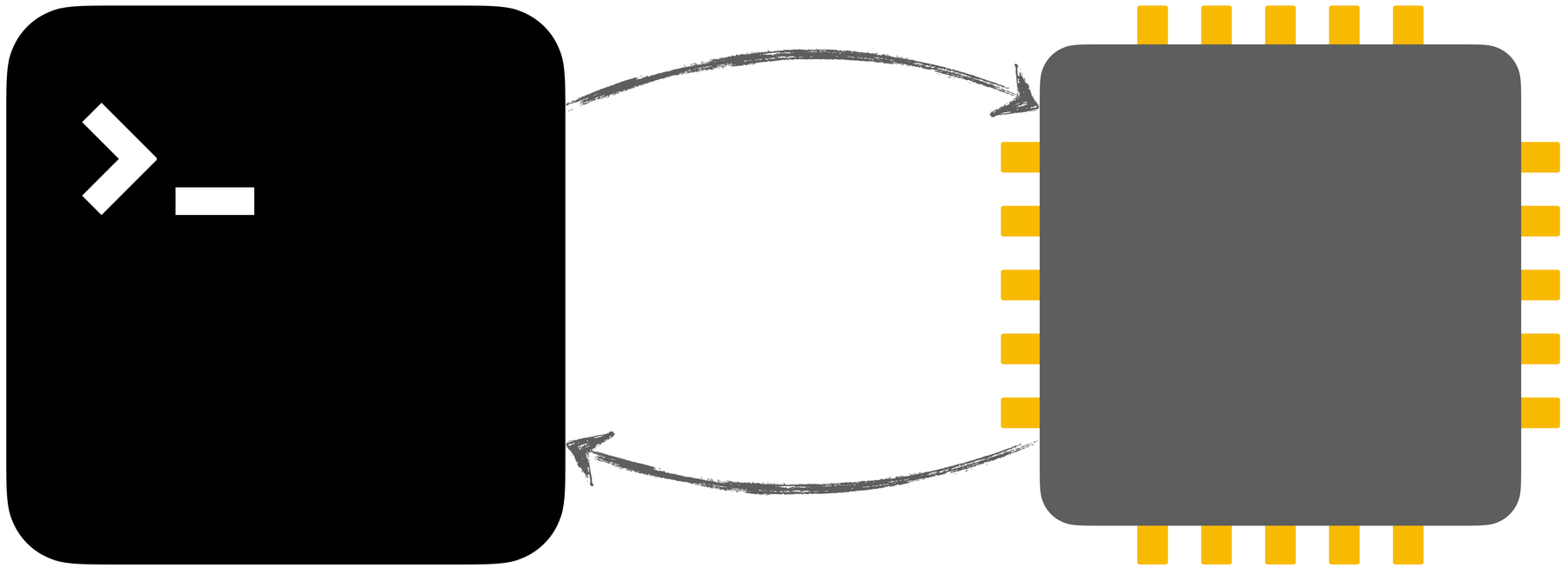
Constant-time

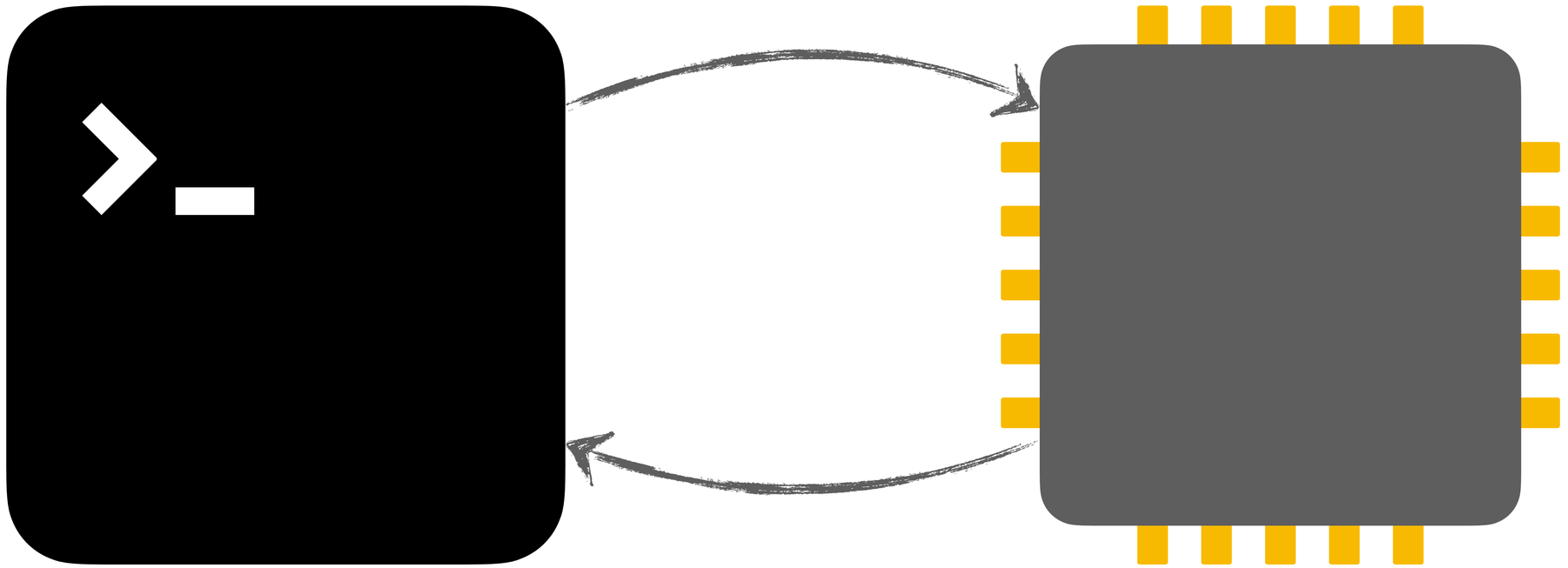


Sandboxing

Backup

What’s next?

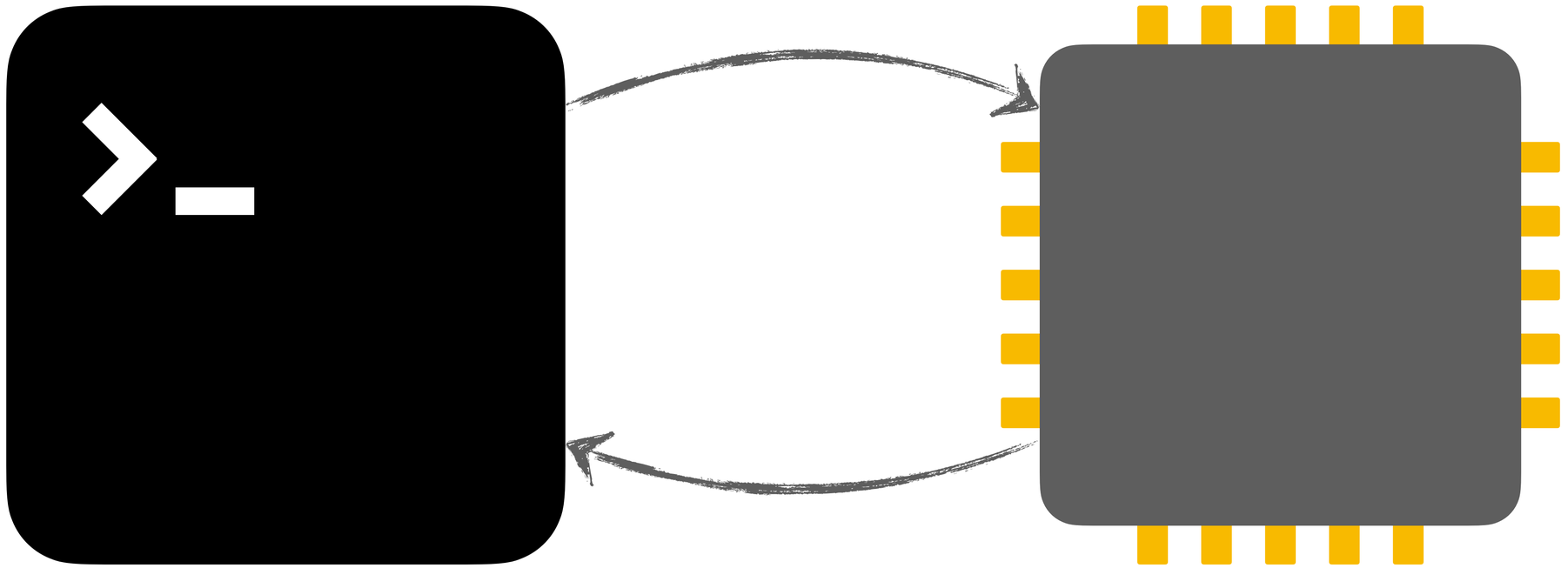

Software
Hardware



Software

Hardware

Knows what ***data*** is ***sensitive***



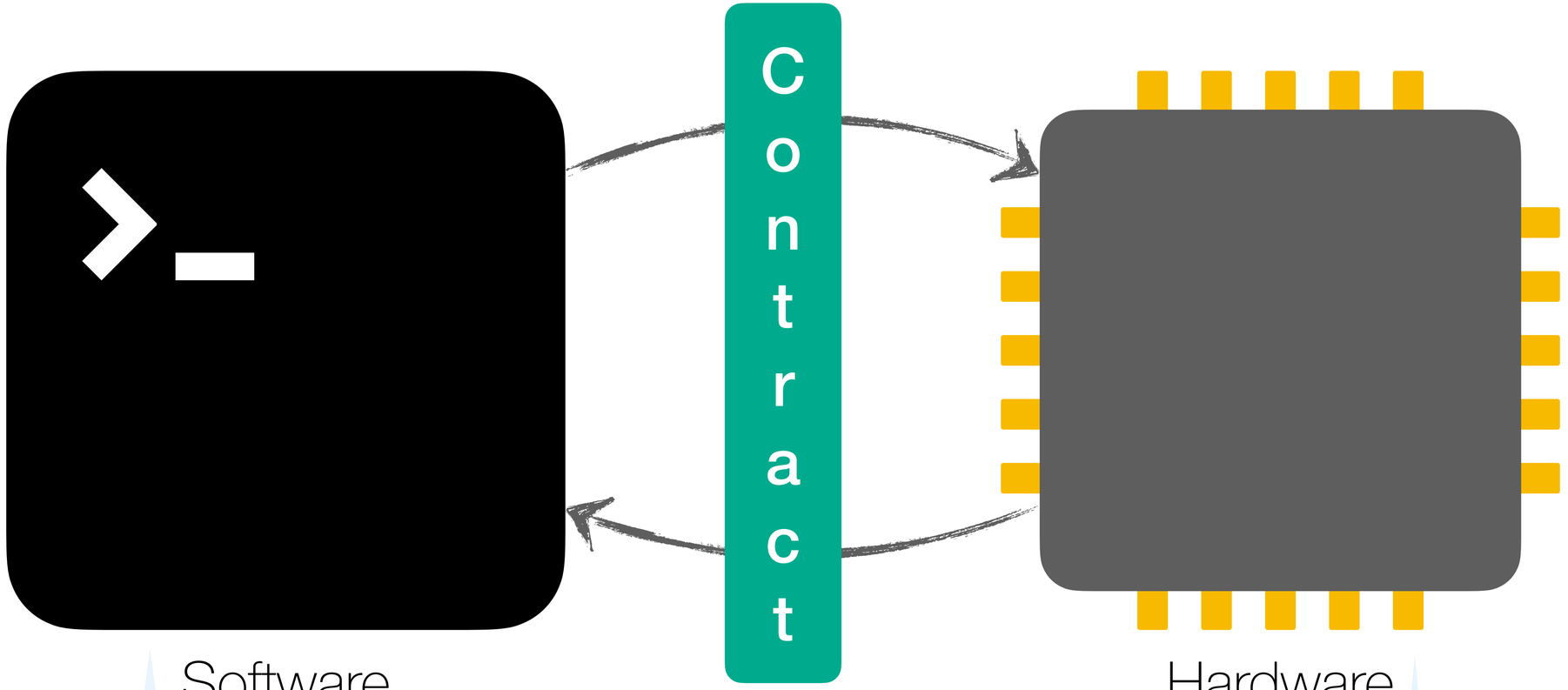
Software

Hardware

Knows what ***data*** is ***sensitive***

Controls
microarchitectural state

Contracts as basis for HW/SW co-design



Knows what ***data*** is ***sensitive***

Controls
microarchitectural state

Contracts @ Software

Contracts @ Software

Program analysis

Program analyzers that work for arbitrary ***contracts***

Guarnieri et al., Spectector: Principled detection of speculative leaks, S&P 2020

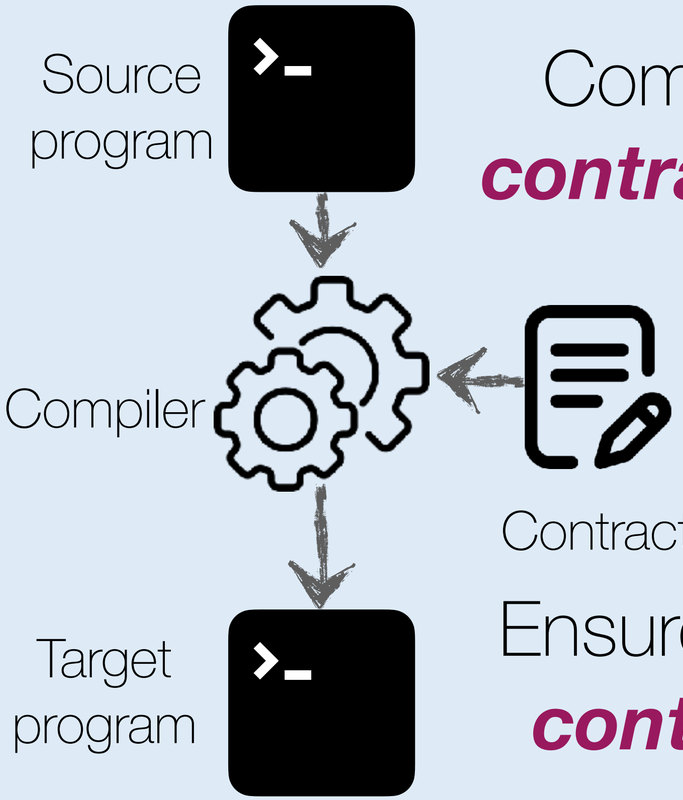
Contracts @ Software

Program analysis

Program analyzers that work for arbitrary ***contracts***

Guarnieri et al., Spectector: Principled detection of speculative leaks, S&P 2020

Secure compilation

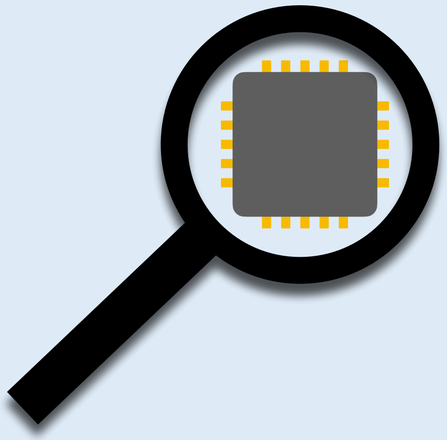


Patrignani and Guarnieri, Exorcising Spectre with secure compilers, CCS 2021

Contracts @ Hardware

Contracts @ Hardware

Verification

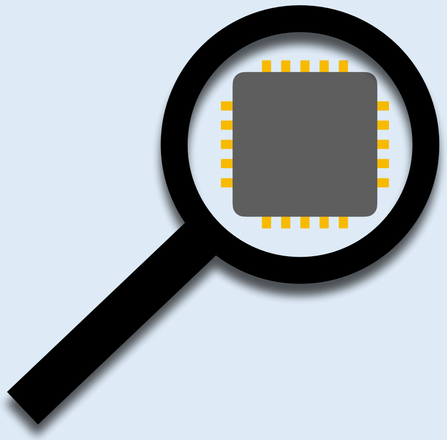


Tools for
checking
compliance
with contracts

(Partially) automate
contract satisfaction
proofs

Contracts @ Hardware

Verification



Tools for checking ***compliance with contracts***

(Partially) automate ***contract satisfaction proofs***

Testing/Fuzzing

Contracts as test oracles

Black box: see Oleksenko et al., Revizor: Fuzzing for Leaks in Black-box CPUs https://arxiv.org/abs/2105.06872

White box: WIP!

A simple processor — Configurations

A simple processor — Configurations

Configurations: State of the processor during computation

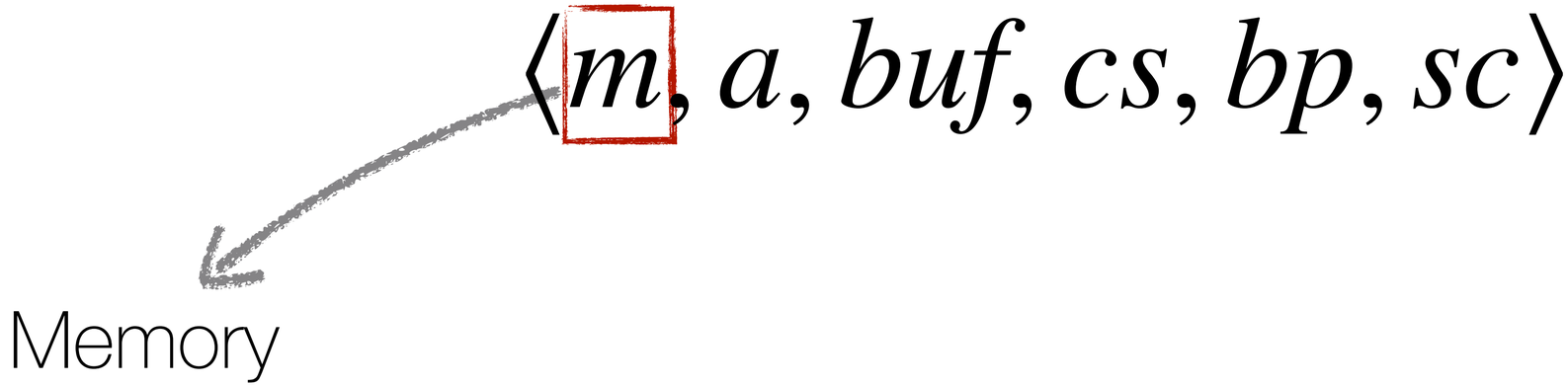
A simple processor — Configurations

Configurations: State of the processor during computation

$$\langle m, a, buf, cs, bp, sc \rangle$$

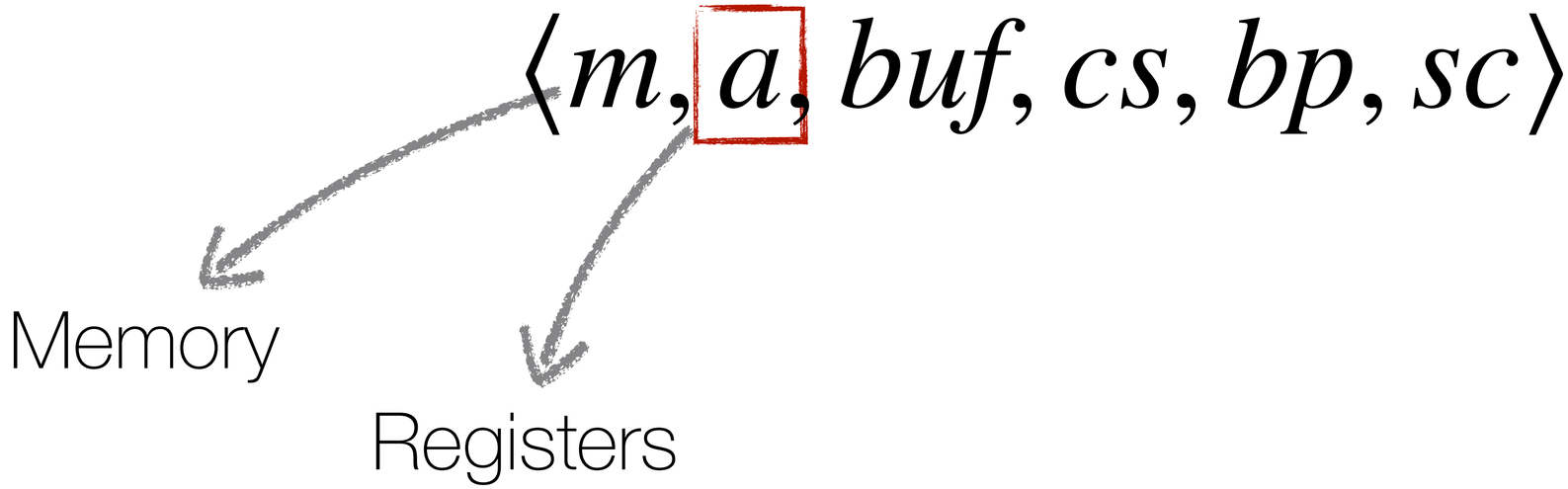
A simple processor — Configurations

Configurations: State of the processor during computation



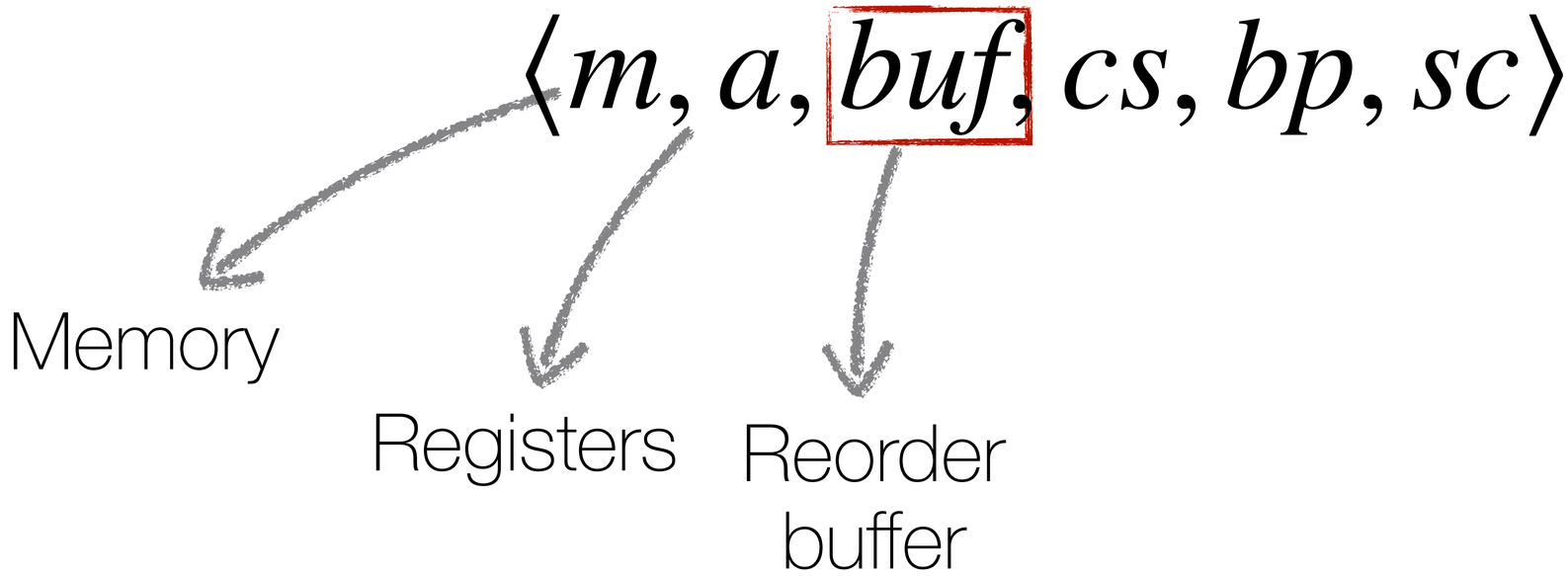
A simple processor — Configurations

Configurations: State of the processor during computation



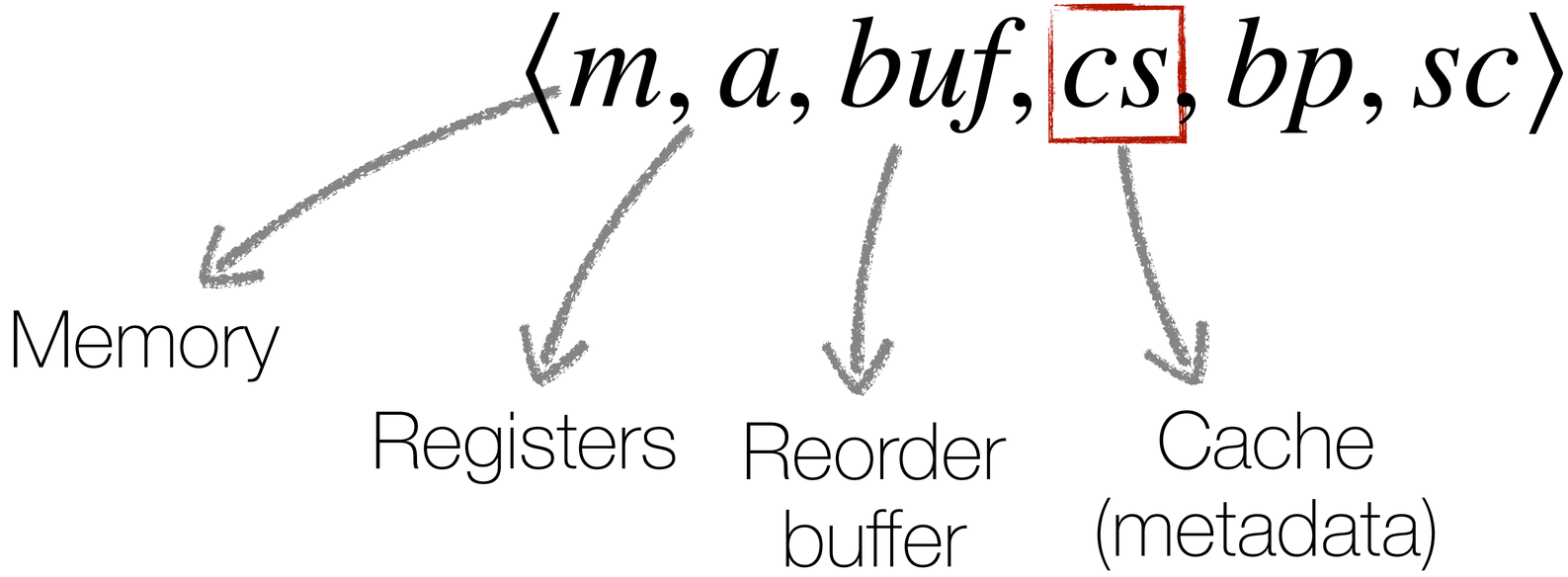
A simple processor — Configurations

Configurations: State of the processor during computation



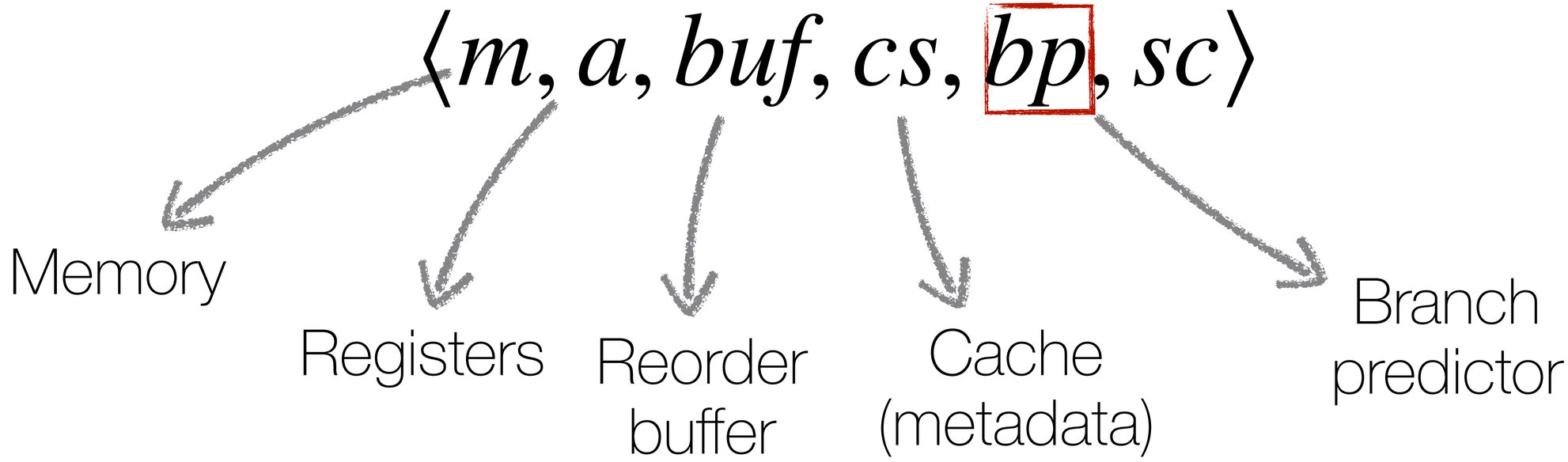
A simple processor — Configurations

Configurations: State of the processor during computation



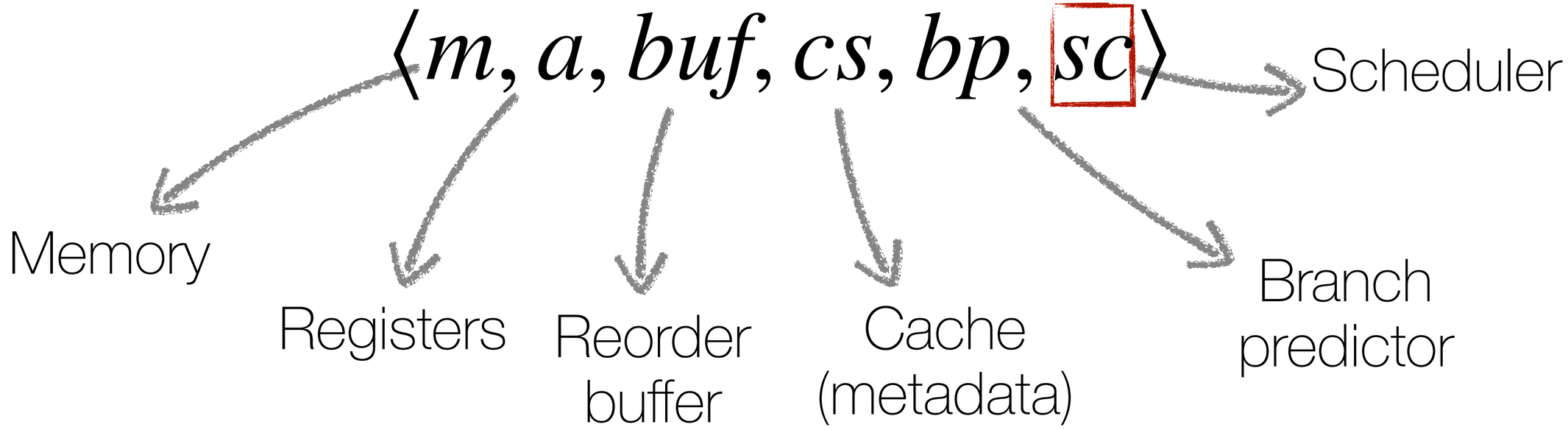
A simple processor — Configurations

Configurations: State of the processor during computation



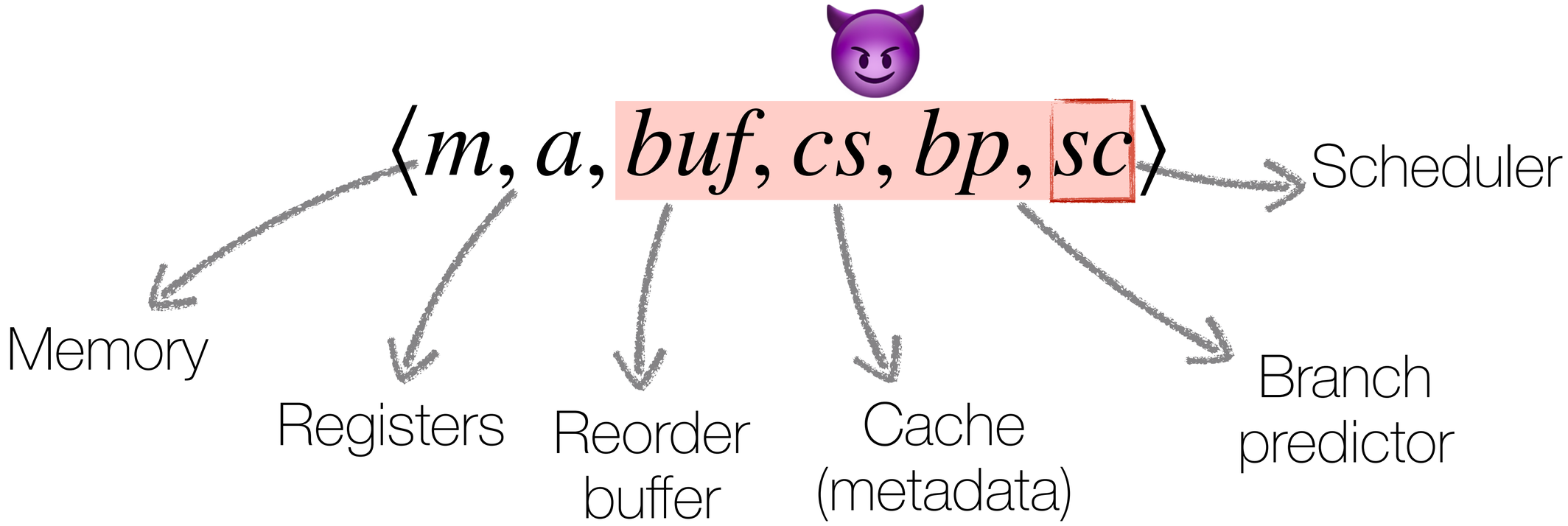
A simple processor — Configurations

Configurations: State of the processor during computation



A simple processor — Configurations

Configurations: State of the processor during computation



A simple processor — Semantics

A simple processor — Semantics

Semantics: Describe how configurations evolve

A simple processor — Semantics

Semantics: Describe how configurations evolve

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \qquad d = next(sc) \qquad sc' = update(sc, buf'\downarrow)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow \langle m', a', buf', cs', bp', sc'\rangle}$$

A simple processor — Semantics

Semantics: Describe how configurations evolve

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \qquad d = next(sc) \qquad sc' = update(sc, buf'\downarrow)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow \langle m', a', buf', cs', bp', sc'\rangle}$$

Current state

A simple processor — Semantics

Semantics: Describe how configurations evolve

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \qquad d = next(sc) \qquad sc' = update(sc, buf'\downarrow)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow \langle m', a', buf', cs', bp', sc'\rangle}$$

Current state

Next state

A simple processor — Semantics

Semantics: Describe how configurations evolve

Directive from scheduler

fetch, ***execute*** i, ***retire***

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \qquad d = next(sc) \qquad sc' = update(sc, buf'\downarrow)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow \langle m', a', buf', cs', bp', sc'\rangle}$$

Current state

Next state

A simple processor — Semantics

Semantics: Describe how configurations evolve

Directive from scheduler
fetch, ***execute*** i, ***retire***

Semantics for ***pipeline stages***

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \qquad d = next(sc) \qquad sc' = update(sc, buf'\downarrow)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow \langle m', a', buf', cs', bp', sc'\rangle}$$

Current state

Next state

Rules capture effect of directives - Fetch

FETCH-BRANCH-HIT

$$\frac{\begin{array}{ccc} a' = apl(buf, a) & |buf| < \mathbf{w} & a'(\mathbf{pc}) \neq \bot \\ p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell & \ell' = predict(bp, a'(\mathbf{pc})) & \\ access(cs, a'(\mathbf{pc})) = \texttt{Hit} & update(cs, a'(\mathbf{pc})) = cs' & \end{array}}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\textbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

Rules capture effect of directives - Fetch

applying reorder buffer
to register state

FETCH-BRANCH-HIT

$$\frac{a' = apl(buf, a) \qquad |buf| < \mathbf{w} \qquad a'(\mathbf{pc}) \neq \bot \qquad p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \qquad \ell' = predict(bp, a'(\mathbf{pc})) \qquad access(cs, a'(\mathbf{pc})) = \texttt{Hit} \qquad update(cs, a'(\mathbf{pc})) = cs'}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\mathbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

Rules capture effect of directives - Fetch

applying reorder buffer to register state

reorder buffer is not full

FETCH-BRANCH-HIT

$$\frac{a' = apl(buf, a) \qquad |buf| < \mathbf{w} \qquad a'(\mathbf{pc}) \neq \bot \\ p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \qquad \ell' = predict(bp, a'(\mathbf{pc})) \\ access(cs, a'(\mathbf{pc})) = \texttt{Hit} \qquad update(cs, a'(\mathbf{pc})) = cs'}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\textbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

Rules capture effect of directives - Fetch

applying reorder buffer to register state

reorder buffer is not full

instruction is a branch

FETCH-BRANCH-HIT

$$\frac{a' = apl(buf, a) \qquad |buf| < \mathbf{w} \qquad a'(\mathbf{pc}) \neq \bot \\ p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \qquad \ell' = predict(bp, a'(\mathbf{pc})) \\ access(cs, a'(\mathbf{pc})) = \texttt{Hit} \qquad update(cs, a'(\mathbf{pc})) = cs'}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\mathbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

Rules capture effect of directives - Fetch

applying reorder buffer to register state

reorder buffer is not full

branch predictor says ℓ' is next

FETCH-BRANCH-HIT

$$\frac{a' = apl(buf, a) \qquad |buf| < \mathbf{w} \qquad a'(\mathbf{pc}) \neq \bot \\ p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \qquad \ell' = predict(bp, a'(\mathbf{pc})) \\ access(cs, a'(\mathbf{pc})) = \texttt{Hit} \qquad update(cs, a'(\mathbf{pc})) = cs'}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\mathbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

instruction is a branch

Rules capture effect of directives - Fetch

applying reorder buffer to register state

reorder buffer is not full

branch predictor says ℓ' is next

FETCH-BRANCH-HIT

$$\frac{a' = apl(buf, a) \qquad |buf| < \mathbf{w} \qquad a'(\mathbf{pc}) \neq \bot \qquad p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \qquad \ell' = predict(bp, a'(\mathbf{pc})) \qquad access(cs, a'(\mathbf{pc})) = \texttt{Hit} \qquad update(cs, a'(\mathbf{pc})) = cs'}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\mathbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

instruction is a branch

Cache hit + updating cache state

Rules capture effect of directives - Fetch

applying reorder buffer to register state

reorder buffer is not full

branch predictor says ℓ' is next

FETCH-BRANCH-HIT

$$\frac{a' = apl(buf, a) \qquad |buf| < \mathbf{w} \qquad a'(\mathbf{pc}) \neq \bot \qquad p(a'(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \qquad \ell' = predict(bp, a'(\mathbf{pc})) \qquad access(cs, a'(\mathbf{pc})) = \texttt{Hit} \qquad update(cs, a'(\mathbf{pc})) = cs'}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\mathbf{fetch}} \langle m, a, buf \cdot \mathbf{pc} \leftarrow \ell' @ a'(\mathbf{pc}), cs', bp\rangle}$$

instruction is a branch

add change of pc to reorder buffer

Cache hit + updating cache state

Rules capture effect of directive - Retire

$$\textsc{Retire-Assignment}\quad \frac{buf = x \leftarrow v@\varepsilon \cdot buf' \qquad v \in Vals}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\textbf{retire}} \langle m, a[x \mapsto v], buf', cs, bp\rangle}$$

Rules capture effect of directive - Retire

result of instruction at head of reorder buffer is resolved

RETIRE-ASSIGNMENT

$$\frac{buf = x \leftarrow v@\varepsilon \cdot buf' \qquad v \in Vals}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\textbf{retire}} \langle m, a[x \mapsto v], buf', cs, bp\rangle}$$

Rules capture effect of directive - Retire

result of instruction at head of reorder buffer is resolved

RETIRE-ASSIGNMENT

$$\frac{buf = x \leftarrow v@\varepsilon \cdot buf' \qquad v \in Vals}{\langle m, a, buf, cs, bp\rangle \xRightarrow{\textbf{retire}} \langle m, a[x \mapsto v], buf', cs, bp\rangle}$$

apply change to registers and remove entry from reorder buffer

Eager load delay *[Sakalis et al. 2019]*

STEP-EAGER-DELAY

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \quad d = next(sc) \quad sc' = update(sc, buf'\downarrow) \quad d = \textbf{execute}\ i \quad buf|_i = \textbf{load}\ x, e \quad \forall \textbf{pc} \leftarrow \ell@\ell' \in buf[0..i-1].\ \ell' = \varepsilon}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\textbf{loadDelay}} \langle m', a', buf', cs', bp', sc'\rangle}$$

STEP-OTHERS

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \quad d = next(sc) \quad sc' = update(sc, buf'\downarrow) \quad d \in \{\textbf{fetch}, \textbf{retire}\} \vee (d = \textbf{execute}\ i \wedge buf|_i \neq \textbf{load}\ x, e)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\textbf{loadDelay}} \langle m', a', buf', cs', bp', sc'\rangle}$$

Eager load delay *[Sakalis et al. 2019]*

STEP-EAGER-DELAY

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \quad d = next(sc) \quad sc' = update(sc, buf'\downarrow) \quad d = \textbf{execute } i \quad buf|_i = \textbf{load } x, e \quad \forall \textbf{pc} \leftarrow \ell@\ell' \in buf[0..i-1].\ \ell' = \varepsilon}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\textbf{loadDelay}} \langle m', a', buf', cs', bp', sc'\rangle}$$

STEP-OTHERS

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \quad d = next(sc) \quad sc' = update(sc, buf'\downarrow) \quad d \in \{\textbf{fetch}, \textbf{retire}\} \vee (d = \textbf{execute } i \wedge buf|_i \neq \textbf{load } x, e)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\textbf{loadDelay}} \langle m', a', buf', cs', bp', sc'\rangle}$$

Loads are ***executed*** only if prior branch instructions are ***resolved***

Eager load delay *[Sakalis et al. 2019]*

STEP-EAGER-DELAY

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \quad d = next(sc) \quad sc' = update(sc, buf'\downarrow) \quad d = \textbf{execute}\ i \quad buf|_i = \textbf{load}\ x, e \quad \forall \textbf{pc} \leftarrow \ell@\ell' \in buf[0..i-1].\ \ell' = \varepsilon}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\textbf{loadDelay}} \langle m', a', buf', cs', bp', sc'\rangle}$$

Loads are ***executed*** only if prior branch instructions are ***resolved***

STEP-OTHERS

$$\frac{\langle m, a, buf, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf', cs', bp'\rangle \quad d = next(sc) \quad sc' = update(sc, buf'\downarrow) \quad d \in \{\textbf{fetch}, \textbf{retire}\} \vee (d = \textbf{execute}\ i \wedge buf|_i \neq \textbf{load}\ x, e)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\textbf{loadDelay}} \langle m', a', buf', cs', bp', sc'\rangle}$$

Everything else works ***as before***

Hw-level taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

$$\textsc{Step}\quad \frac{d = next(sc) \qquad buf_{ul} = unlbl(buf, d) \qquad \langle m, a, buf_{ul}, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf'_{ul}, cs', bp'\rangle \qquad sc' = update(sc, buf'\downarrow) \qquad buf' = lbl(buf'_{ul}, buf, d)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\mathbf{tt}} \langle m', a' buf', cs', bp', sc'\rangle}$$

Hw-level taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

$$\textsc{Step}\quad \frac{d = next(sc) \qquad buf_{ul} = unlbl(buf, d) \qquad \langle m, a, buf_{ul}, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf'_{ul}, cs', bp'\rangle \qquad sc' = update(sc, buf'\downarrow) \qquad buf' = lbl(buf'_{ul}, buf, d)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\mathbf{tt}} \langle m', a', buf', cs', bp', sc'\rangle}$$

Entries in the ***reorder buffer*** are labelled as ***safe***/***unsafe***

Hw-level taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

Derive ***unlabeled buffer*** (hides information tagged as unsafe)

$$\textsc{Step}\quad \frac{d = next(sc) \qquad buf_{ul} = unlbl(buf, d) \qquad \langle m, a, buf_{ul}, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf'_{ul}, cs', bp'\rangle \qquad sc' = update(sc, buf'\downarrow) \qquad buf' = lbl(buf'_{ul}, buf, d)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\mathbf{tt}} \langle m', a' buf', cs', bp', sc'\rangle}$$

Entries in the ***reorder buffer*** are labelled as ***safe***/***unsafe***

Hw-level taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

Computation on ***unlabeled buffer***

Derive ***unlabeled buffer*** (hides information tagged as unsafe)

$$\textsc{Step}\quad \frac{d = next(sc) \qquad buf_{ul} = unlbl(buf, d) \qquad \langle m, a, buf_{ul}, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf'_{ul}, cs', bp'\rangle \qquad sc' = update(sc, buf'\downarrow) \qquad buf' = lbl(buf'_{ul}, buf, d)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\mathbf{tt}} \langle m', a' buf', cs', bp', sc'\rangle}$$

Entries in the ***reorder buffer*** are labelled as ***safe***/***unsafe***

Hw-level taint-tracking *[Yu et al. 2019, Weisse et al. 2019]*

Computation on ***unlabeled buffer***

Derive ***unlabeled buffer*** (hides information tagged as unsafe)

$$\textsc{Step}\quad \frac{d = next(sc) \qquad buf_{ul} = unlbl(buf, d) \qquad \langle m, a, buf_{ul}, cs, bp\rangle \overset{d}{\Rightarrow} \langle m', a', buf'_{ul}, cs', bp'\rangle \qquad sc' = update(sc, buf'\downarrow) \qquad buf' = lbl(buf'_{ul}, buf, d)}{\langle m, a, buf, cs, bp, sc\rangle \Rightarrow_{\mathbf{tt}} \langle m', a' buf', cs', bp', sc'\rangle}$$

Update labels

Entries in the ***reorder buffer*** are labelled as ***safe***/***unsafe***

Checking secure programming

	Constant-time	*Sandboxing*
$[\![\cdot]\!]^{seq}_{ct}$	✓	✓
$[\![\cdot]\!]^{seq}_{arch}$	.. + NI wrt $[\![\cdot]\!]^{seq}_{arch}$	✓
$[\![\cdot]\!]^{spec}_{ct}$	.. + Speculative NI *[Guarnieri et al., S&P'20]*	.. + weak spec. NI

See paper for: security definitions

Checking secure programming

	Constant-time	*Sandboxing*
$[\![\cdot]\!]^{seq}_{ct}$	✓	✓
$[\![\cdot]\!]^{seq}_{arch}$	.. + NI wrt $[\![\cdot]\!]^{seq}_{arch}$	✓
$[\![\cdot]\!]^{spec}_{ct}$	.. + Speculative NI *[Guarnieri et al., S&P'20]*	.. + weak spec. NI

See paper for: security definitions

Background: Reorder buffer

Background: Reorder buffer

- Key hardware data structure for
  out-of-order and speculative execution

Background: Reorder buffer

- Key hardware data structure for
  out-of-order and speculative execution

- Keeps track of “in-flight instructions”

Background: Reorder buffer

- Key hardware data structure for out-of-order and speculative execution
- Keeps track of “in-flight instructions”
- Example:

Entry	*Instruction*	*Control Dep.*
0:	`c ⟵ x < A_size`	-
1:	`beqz c, END`	-
2:	-	-
3:	-	-
...	...	...

```
      c ⟵ x < A_size
      beqz c, END
L1:   load t, A + x
      load y, B + t
END:
```

Background: Reorder buffer

- Key hardware data structure for out-of-order and speculative execution
- Keeps track of “in-flight instructions”
- Example:

```
      c ⟵ x < A_size
      beqz c, END
L1:   load t, A + x
      load y, B + t
END:
```

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ x < A_size	-
1:	beqz c, END	-
2:	-	-
3:	-	-
...	...	...

Speculative Instruction Fetch

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ x < A_size	-
1:	beqz c, END	-
2:	load t, A + x	1
3:	-	-
...	...	...

Background: Reorder buffer

- Key hardware data structure for out-of-order and speculative execution
- Keeps track of “in-flight instructions”
- Example:

```
      c ⟵ x < A_size
      beqz c, END
L1:   load t, A + x
      load y, B + t
END:
```

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ x < A_size	-
1:	beqz c, END	-
2:	load t, A + x	1
3:	-	-
...	...	...

Speculative Instruction Fetch

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ x < A_size	-
1:	beqz c, END	-
2:	load t, A + x	1
3:	load y, B + t	1
...	...	...

Background: Reorder buffer

- Key hardware data structure for out-of-order and speculative execution
- Keeps track of “in-flight instructions”
- Example:

```
      c ⟵ x < A_size
      beqz c, END
L1:   load t, A + x
      load y, B + t
END:
```

Entry	Instruction	Control Dep.
0:	c ⟵ x < A_size	-
1:	beqz c, END	-
2:	load t, A + x	1
3:	load y, B + t	1
...	...	...

Evaluate `x < A_size`

Entry	Instruction	Control Dep.
0:	c ⟵ 0	-
1:	beqz c, END	-
2:	load t, A + x	1
3:	load y, B + t	1
...	...	...

Background: Reorder buffer

- Key hardware data structure for out-of-order and speculative execution
- Keeps track of “in-flight instructions”
- Example:

```
      c ⟵ x < A_size
      beqz c, END
L1:   load t, A + x
      load y, B + t
END:
```

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ 0	-
1:	beqz c, *END*	-
2:	load t, **A** + x	1
3:	load y, **B** + t	1
...	...	...

Rollback mis-speculation

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ 0	-
1:	beqz c, *END*	-
2:	-	-
3:	-	-
...	...	...

Background: Reorder buffer

- Key hardware data structure for out-of-order and speculative execution
- Keeps track of “in-flight instructions”
- Example:

```
      c ⟵ x < A_size
      beqz c, END
L1:   load t, A + x
      load y, B + t
END:
```

Entry	*Instruction*	*Control Dep.*
0:	c ⟵ 0	-
1:	beqz c, *END*	-
2:	–	-
3:	–	-
...	...	...

Retire

Entry	*Instruction*	*Control Dep.*
0:	beqz c, *END*	-
1:	–	-
2:	–	-
3:	–	-
...	...	...

Speculative execution attacks 101

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

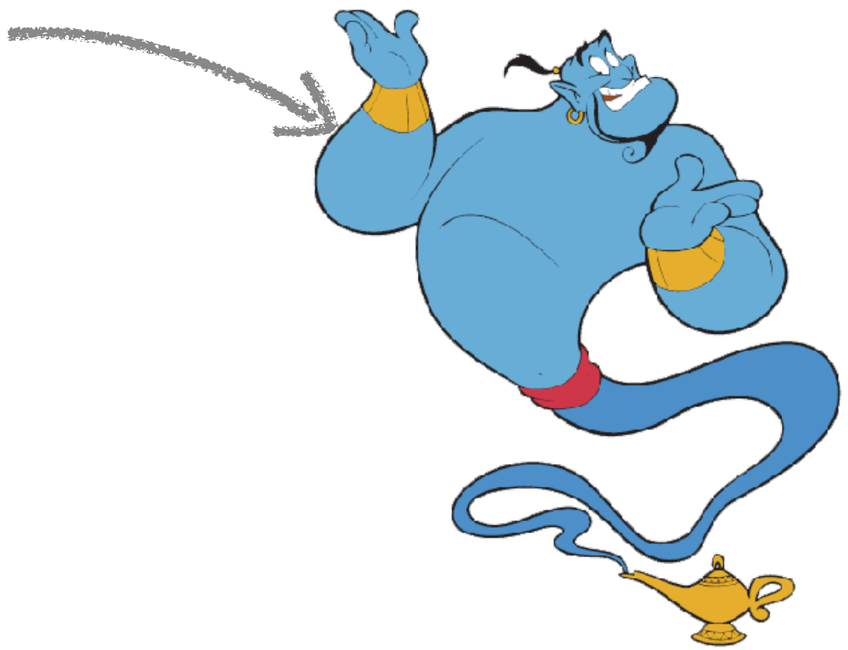
Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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



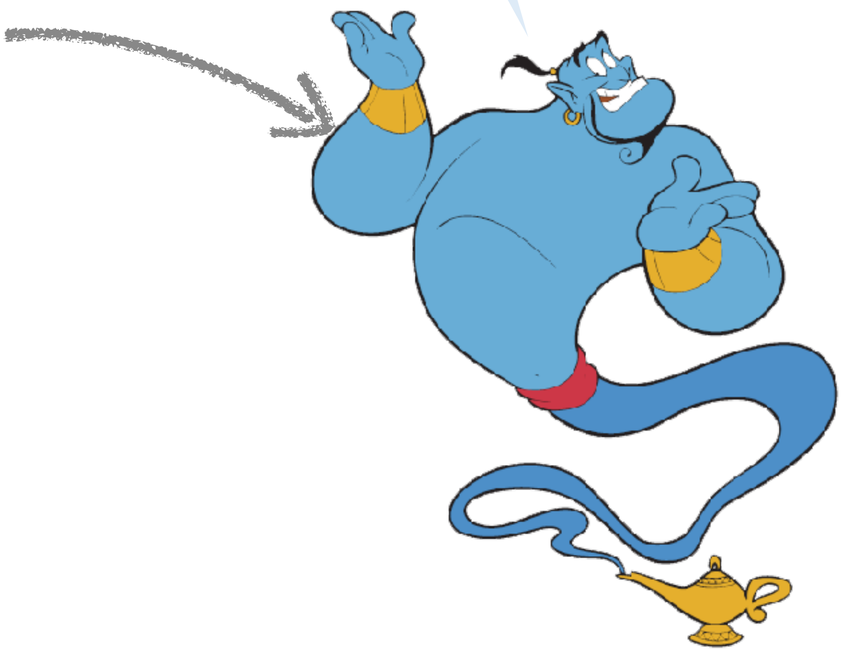
Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

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



Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

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

Branch predictor

Wrong predicton? ***Rollback changes***!

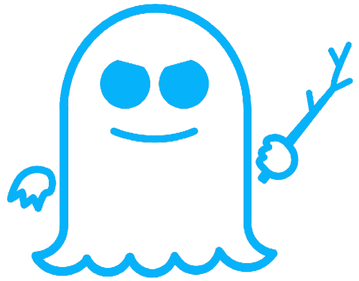
- Architectural (ISA) state
- Microarchitectural state

Spectre V1

Spectre V1

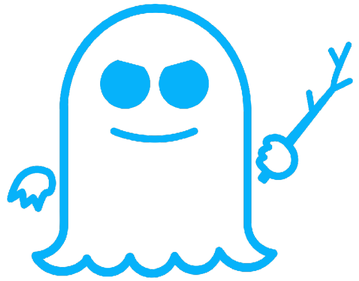
```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

Spectre V1

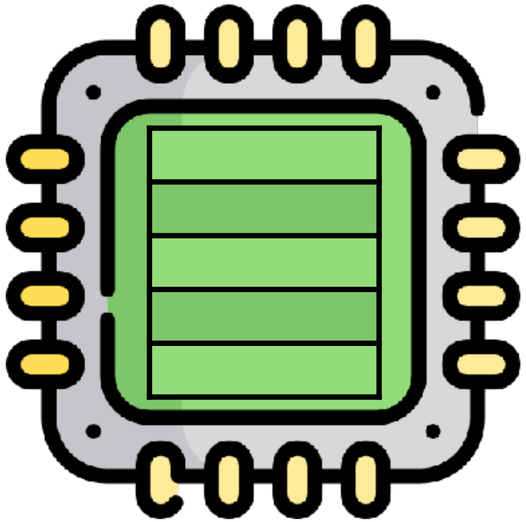


```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

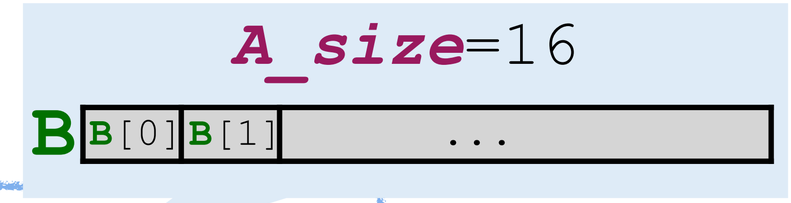
Spectre V1



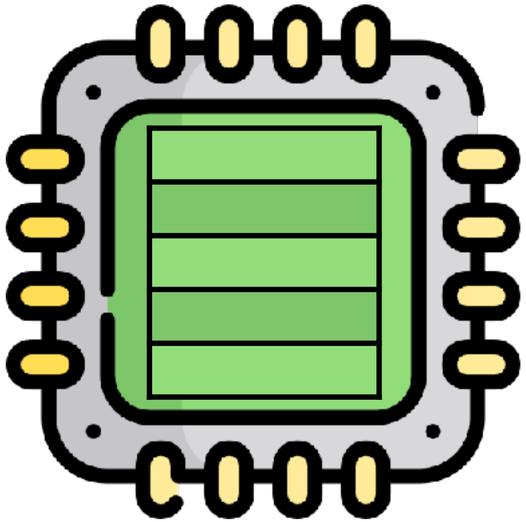
```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

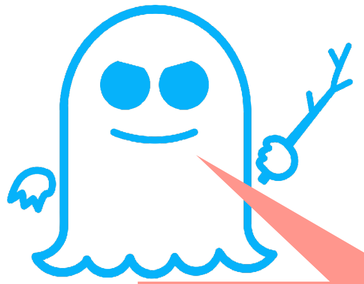


Spectre V1



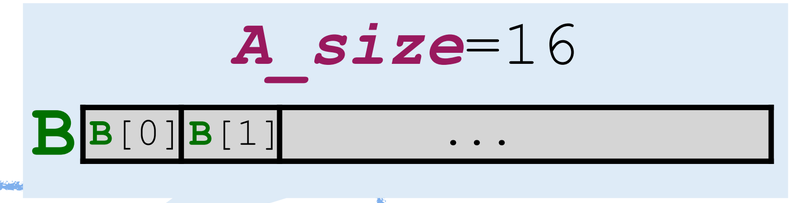

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

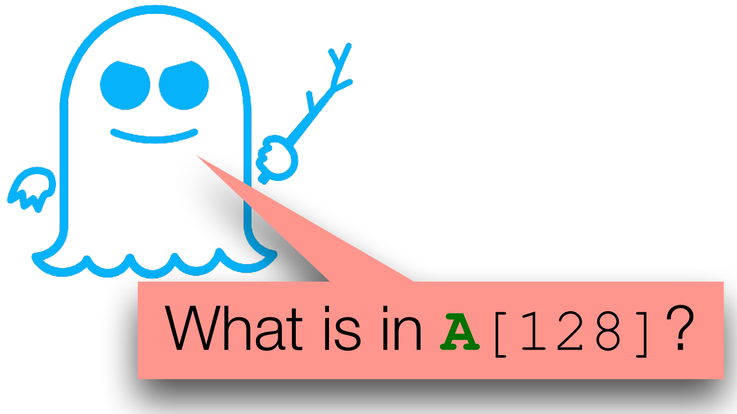




What is in A[128]?

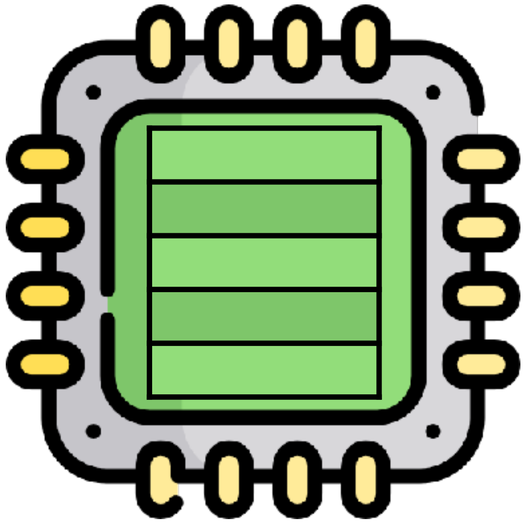
Spectre V1



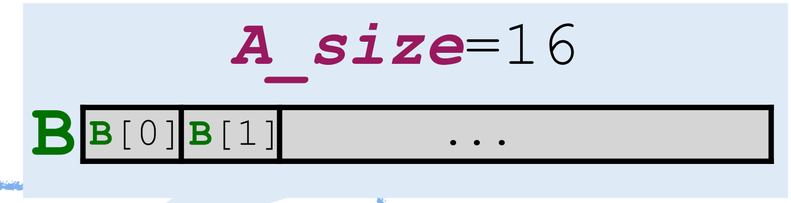


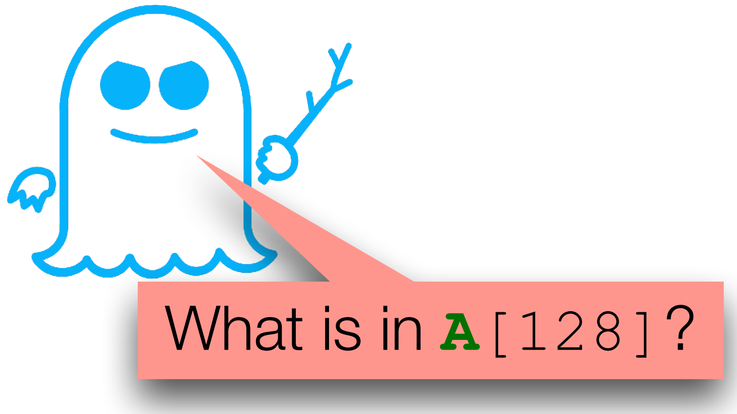

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Train branch predictor



Spectre V1

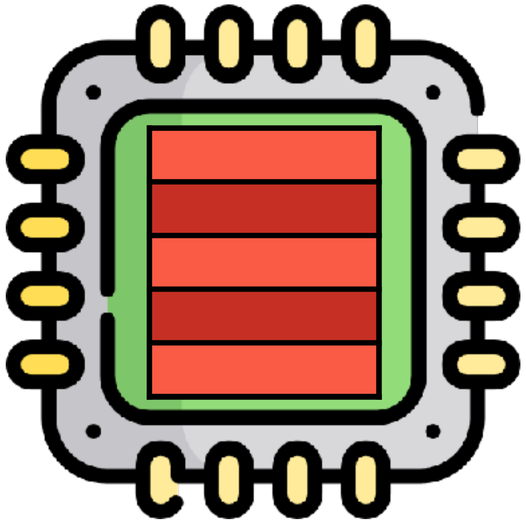




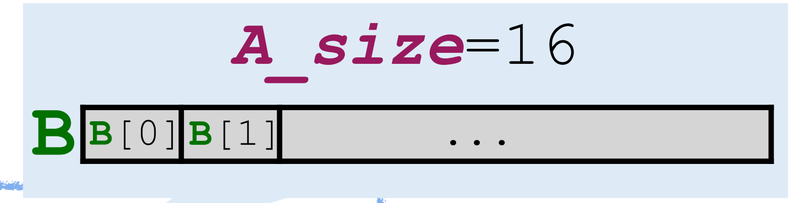

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Train branch predictor

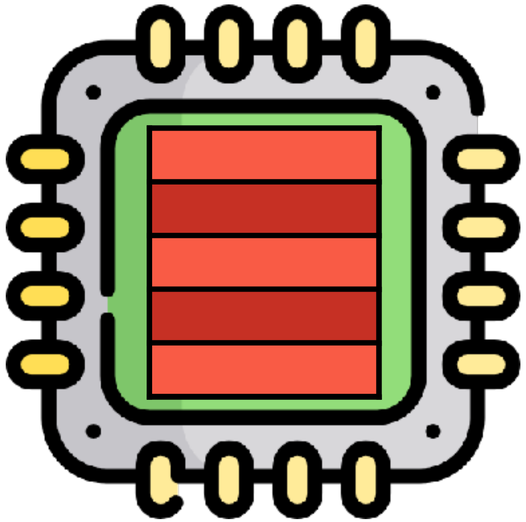
2) Prepare cache

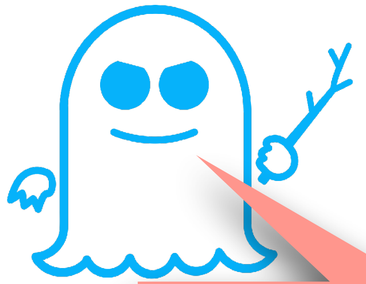


Spectre V1




```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```





What is in `A[128]`?

1) Train branch predictor

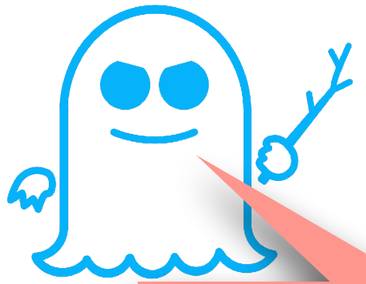
2) Prepare cache

3) Run with `x = 128`

Spectre V1

```
A_size=16
```


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

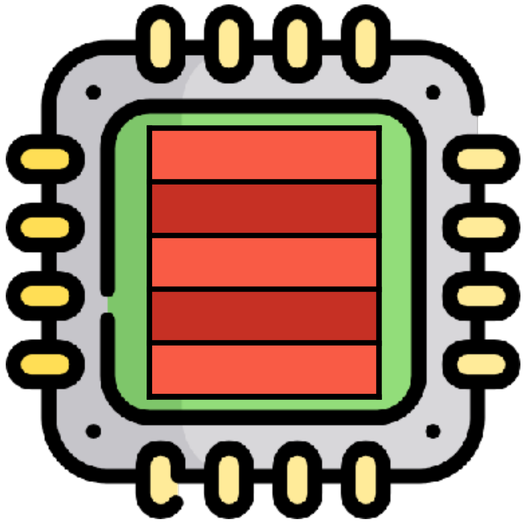


What is in `A[128]`?

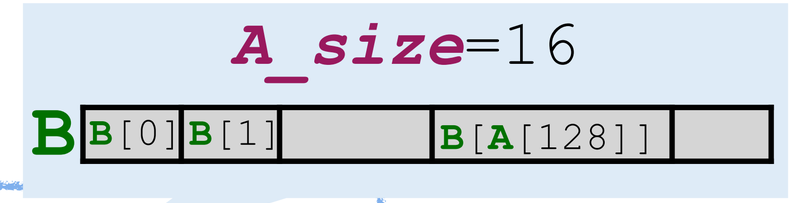
1) Train branch predictor

2) Prepare cache

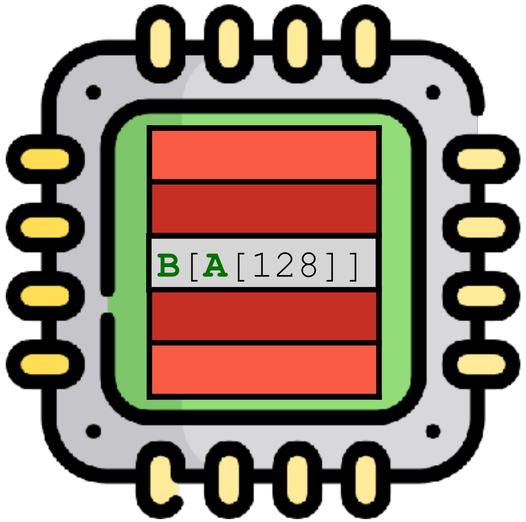
3) Run with `x = 128`

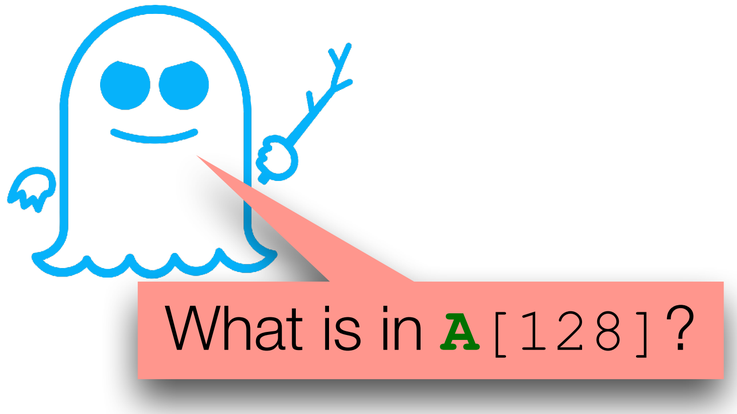


Spectre V1




```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



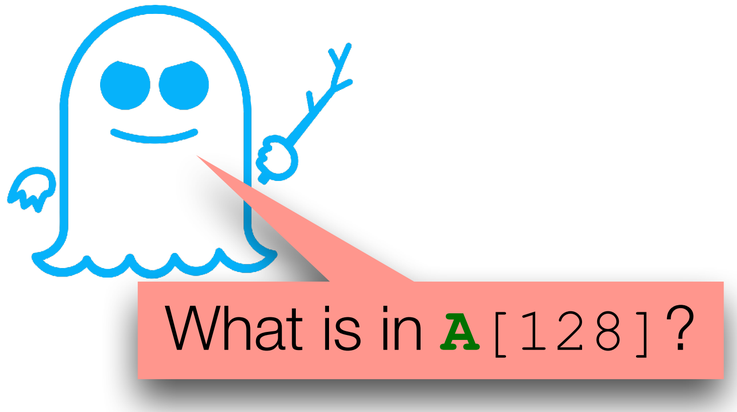


1) Train branch predictor

2) Prepare cache

3) Run with `x = 128`

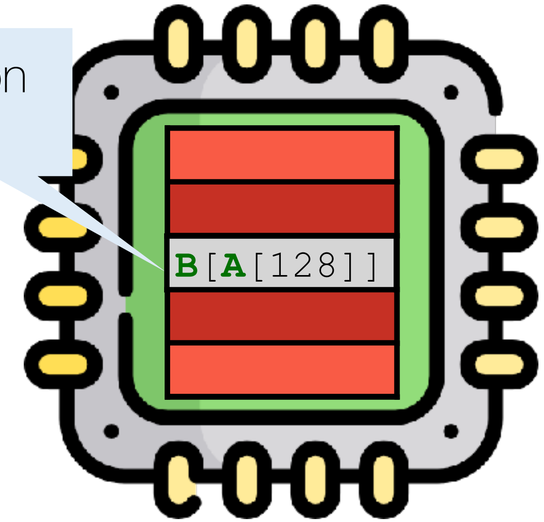
Spectre V1




```
A_size=16
```


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

Depends on A[128]

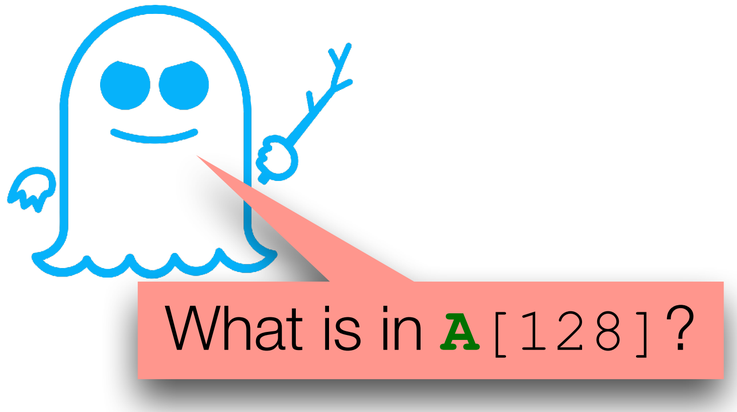


1) Train branch predictor

2) Prepare cache

3) Run with x = 128

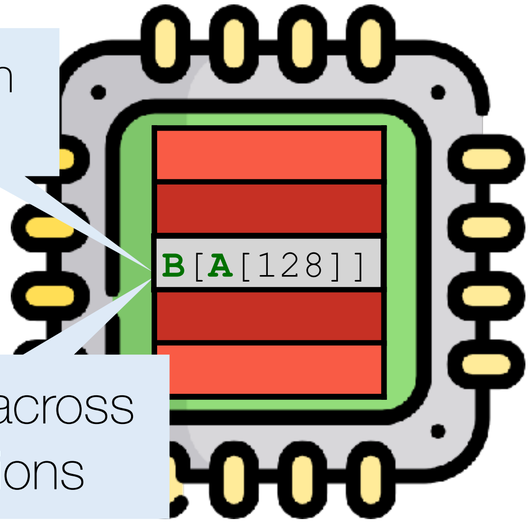
Spectre V1



A_size=16

B | B[0] | B[1] | | B[A[128]] | |

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



Depends on A[128]

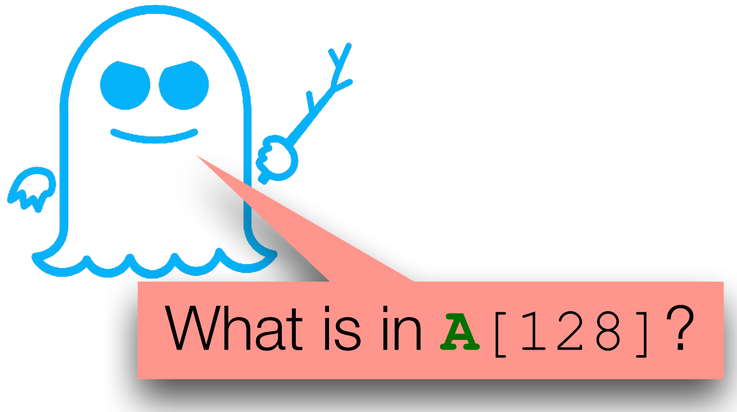
Persistent across speculations

1) Train branch predictor

2) Prepare cache

3) Run with *x* = 128

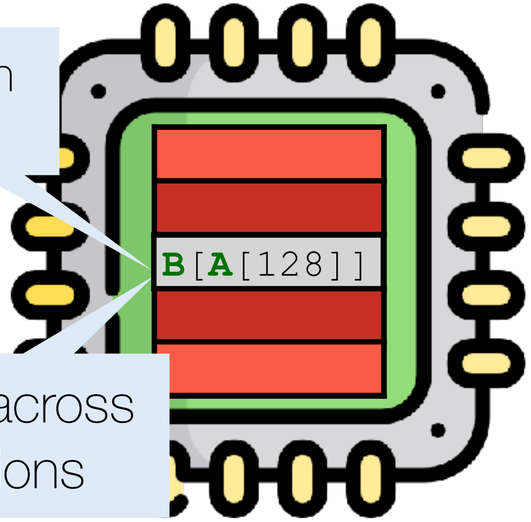
Spectre V1




```
A_size=16
```


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

Depends on A[128]



Persistent across speculations

1) Train branch predictor

2) Prepare cache

3) Run with `x = 128`

4) Extract from cache

Secure programming — foundations

Secure programming — foundations

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

Secure programming — foundations

Specify secret data

Program p is ***non-interferent*** wrt contract 📄 and policy π

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

Secure programming — foundations

Specify secret data

Program p is ***non-interferent*** wrt contract 📄 and policy π

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

Secure programming — foundations

Specify secret data

Program p is ***non-interferent*** wrt contract 📄 and policy π

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

Secure programming — foundations

Specify secret data

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

If p is ***non-interferent*** wrt contract 📄 and policy π, and
hardware 🔲 satisfies 📄, then
p is ***non-interferent*** wrt hardware 🔲 and policy π

Secure programming — foundations

Specify secret data

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

If p is ***non-interferent*** wrt contract 📄 and policy π, and hardware 🔲 satisfies 📄, then
p is ***non-interferent*** wrt hardware 🔲 and policy π

Secure programming — foundations

Specify secret data

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

If p is ***non-interferent*** wrt contract 📄 and policy π, and
hardware 🔲 satisfies 📄, then
p is ***non-interferent*** wrt hardware 🔲 and policy π

Secure programming — foundations

Specify secret data

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

If p is ***non-interferent*** wrt contract 📄 and policy π, and
hardware 🔲 satisfies 📄, then

p is ***non-interferent*** wrt hardware 🔲 and policy π

Constant-time

Constant-time

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

Constant-time

Control-flow and memory accesses do not depend on secrets

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

Constant-time

Control-flow and memory accesses do not depend on secrets

Traditional CT wrt policy π ≡ non-interference wrt **seq-ct** and π

General CT wrt π and 📄 ≡ non-interference wrt 📄 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)

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

Sandboxing

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

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

General SB wrt π and  $\equiv$

Traditional SB wrt π + non-interference wrt π and 

Checking secure programming

	Constant-time	*Sandboxing*
$[\![\cdot]\!]^{seq}_{ct}$	Traditional CT	Traditional SB
$[\![\cdot]\!]^{seq}_{arch}$	.. + NI wrt $[\![\cdot]\!]^{seq}_{arch}$	Traditional SB
$[\![\cdot]\!]^{spec}_{ct}$	.. + speculative NI *[Guarnieri et al., S&P'20]*	.. + weak spec. NI

Checking secure programming

	Constant-time	***Sandboxing***
$[\![\cdot]\!]^{seq}_{ct}$	Traditional CT	Traditional SB
$[\![\cdot]\!]^{seq}_{arch}$	.. + NI wrt $[\![\cdot]\!]^{seq}_{arch}$	Traditional SB
$[\![\cdot]\!]^{spec}_{ct}$	.. + speculative NI *[Guarnieri et al., S&P'20]*	.. + weak spec. NI