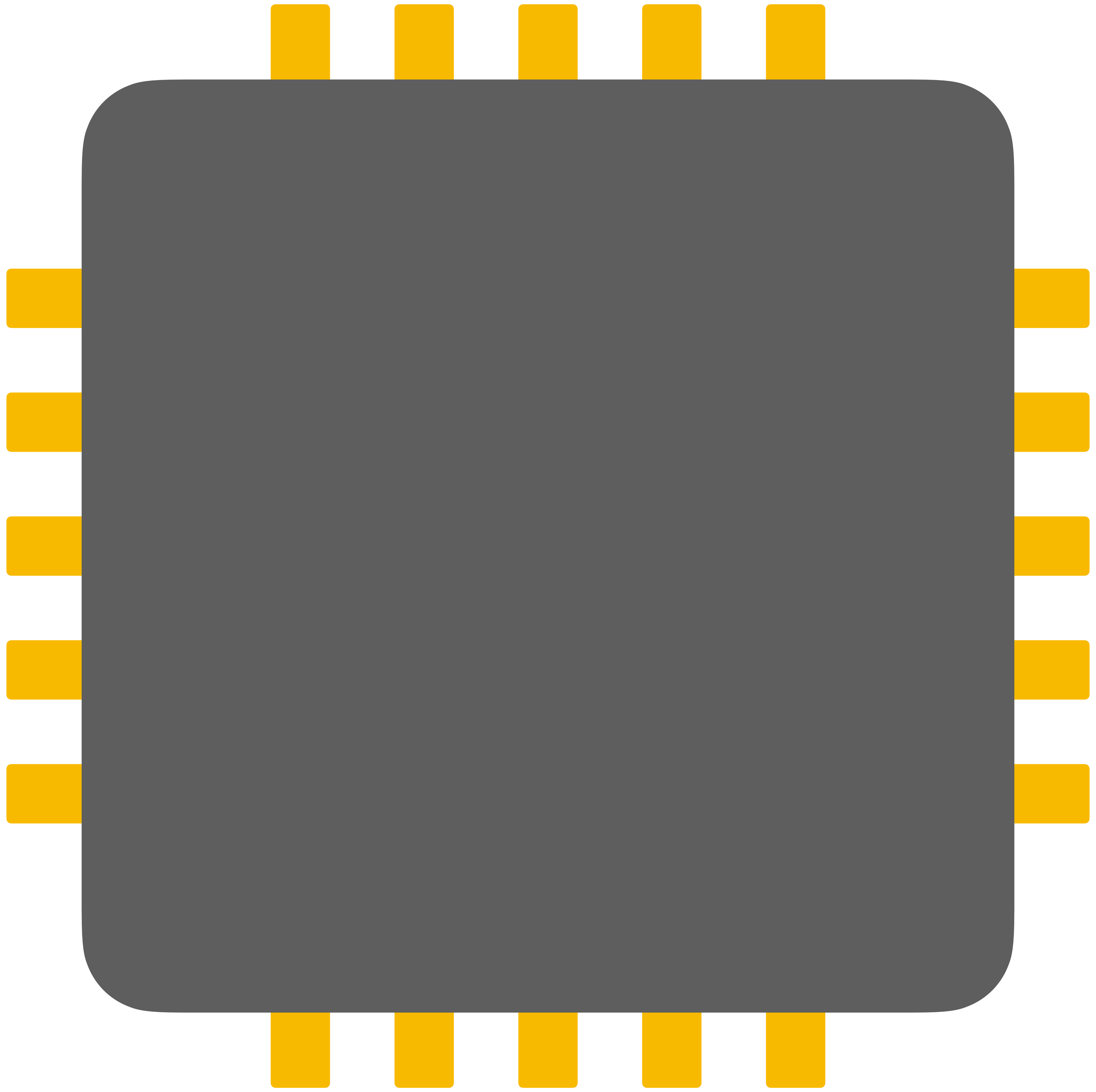
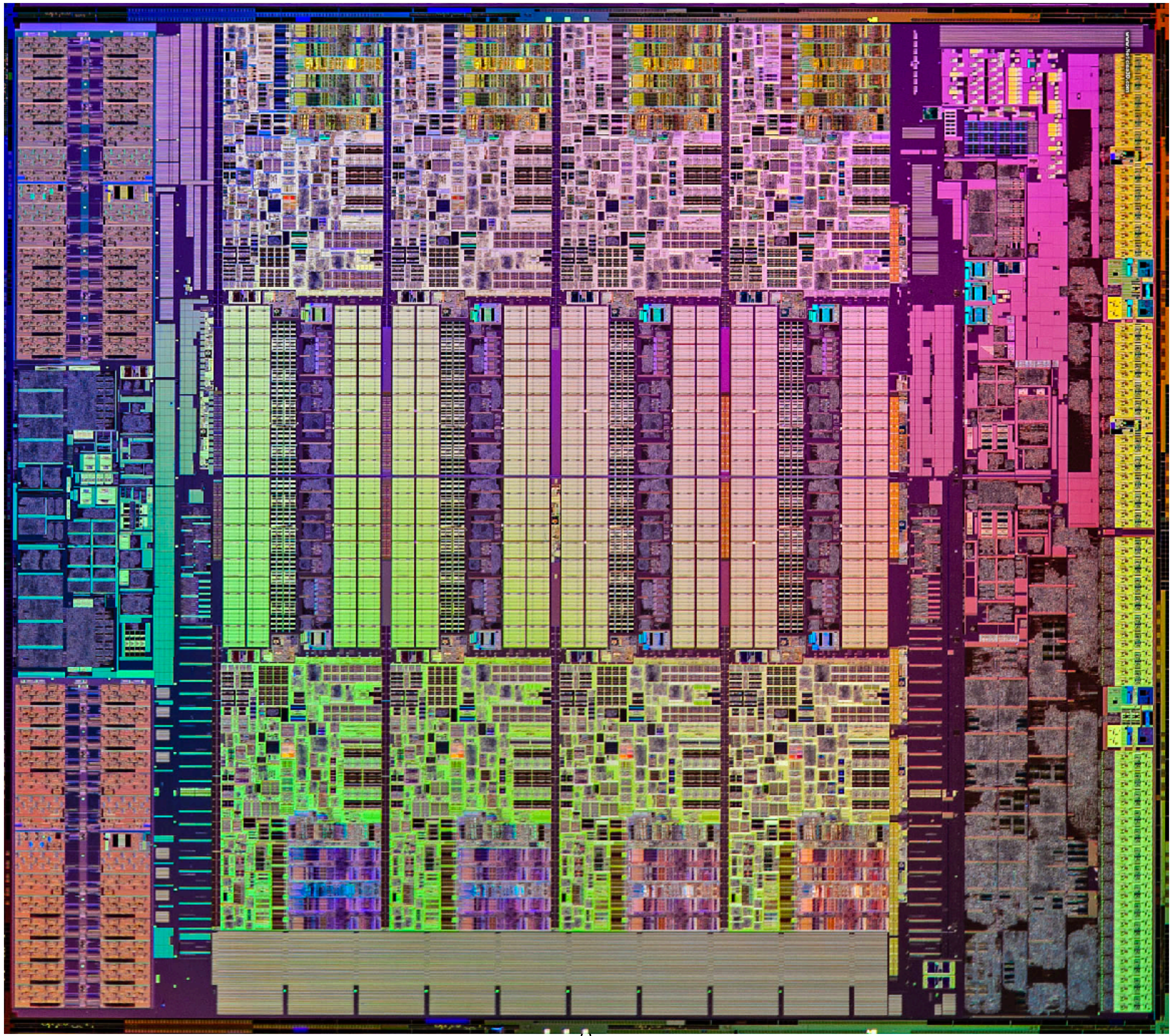
Principled foundations for microarchitectural security

Marco Guarnieri
IMDEA Software Institute

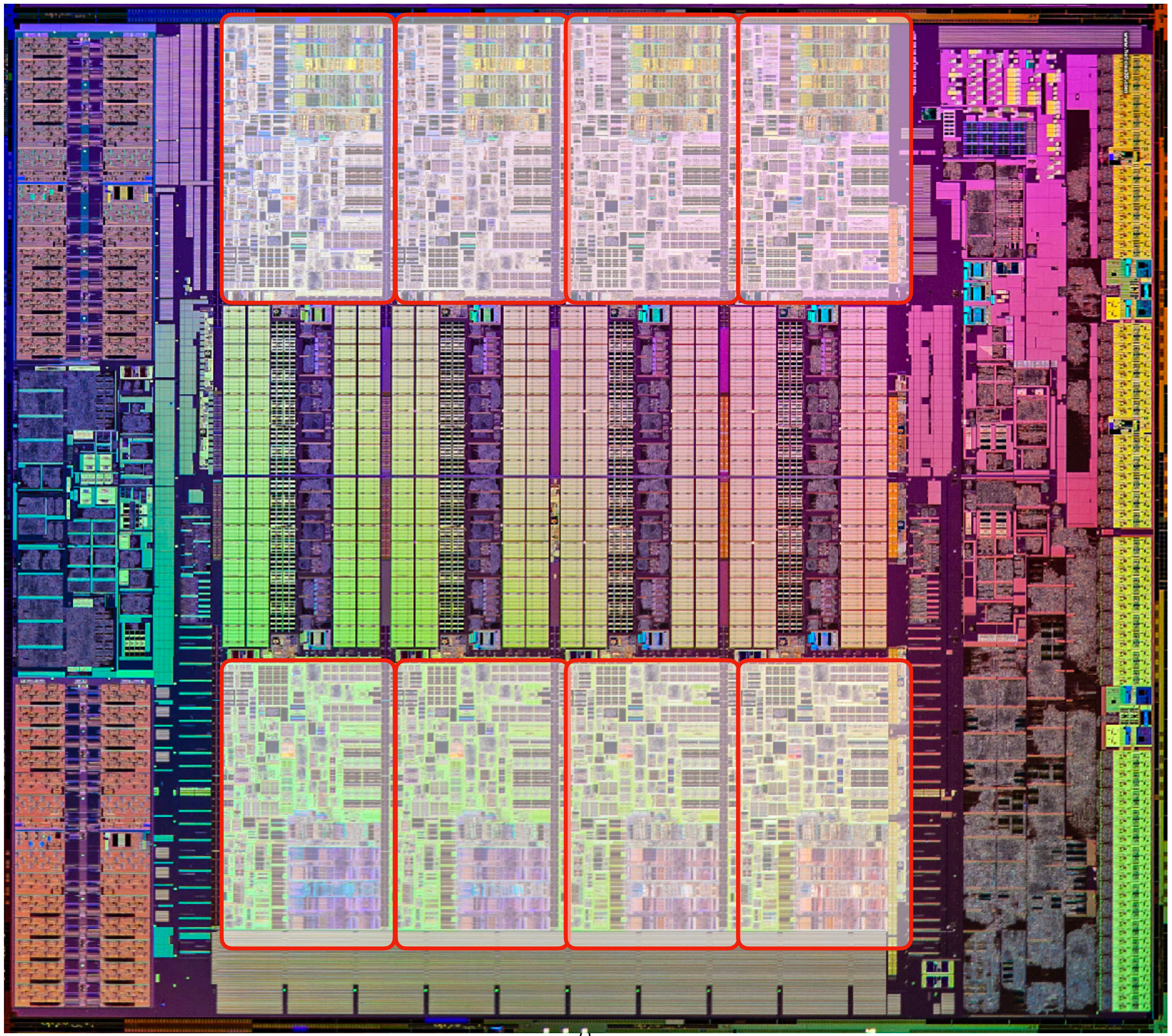
SILM, 06-06-2022 @ Genova

Contacts:
marco.guarnieri@imdea.org
@MarcoGuarnier1

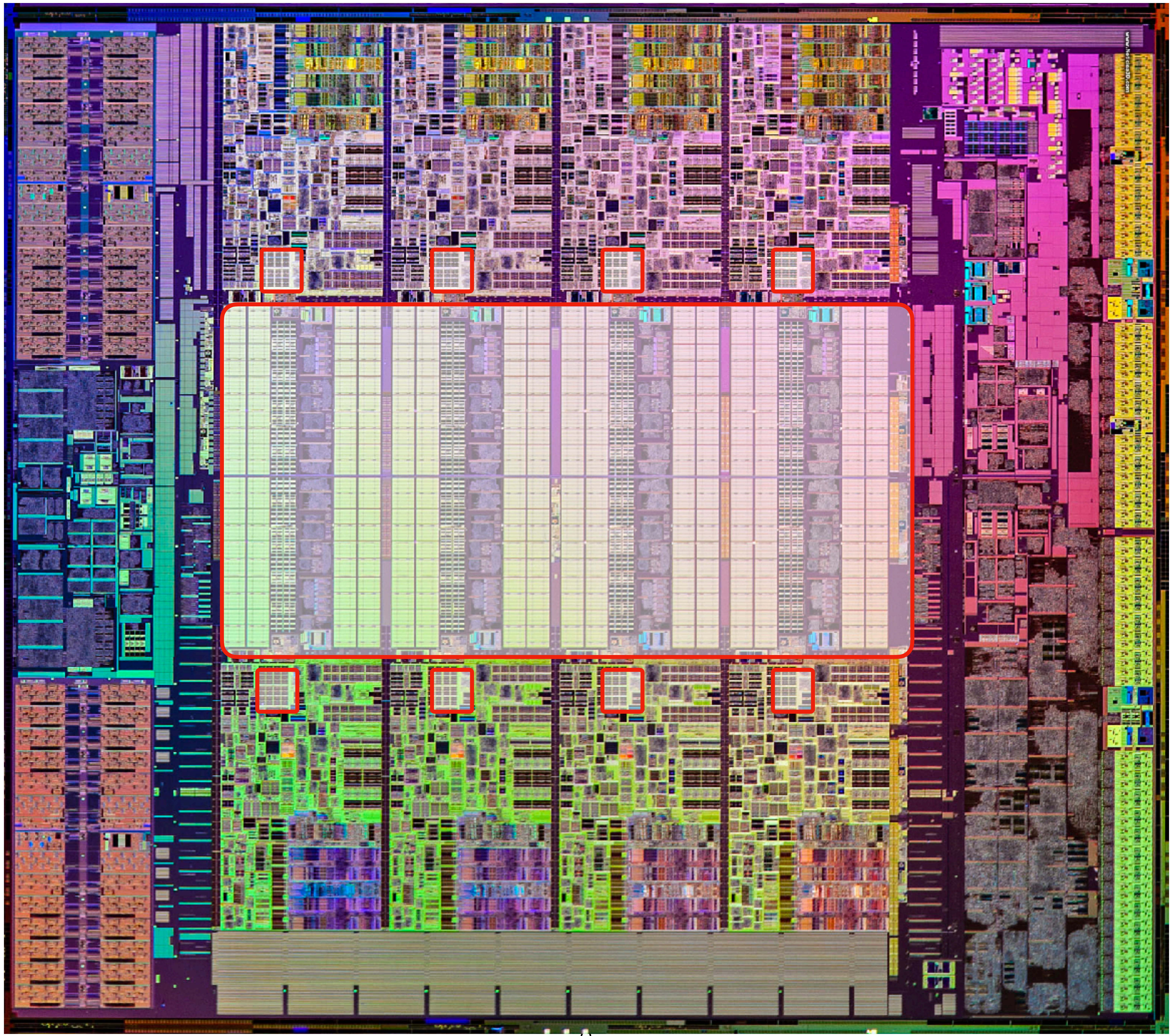




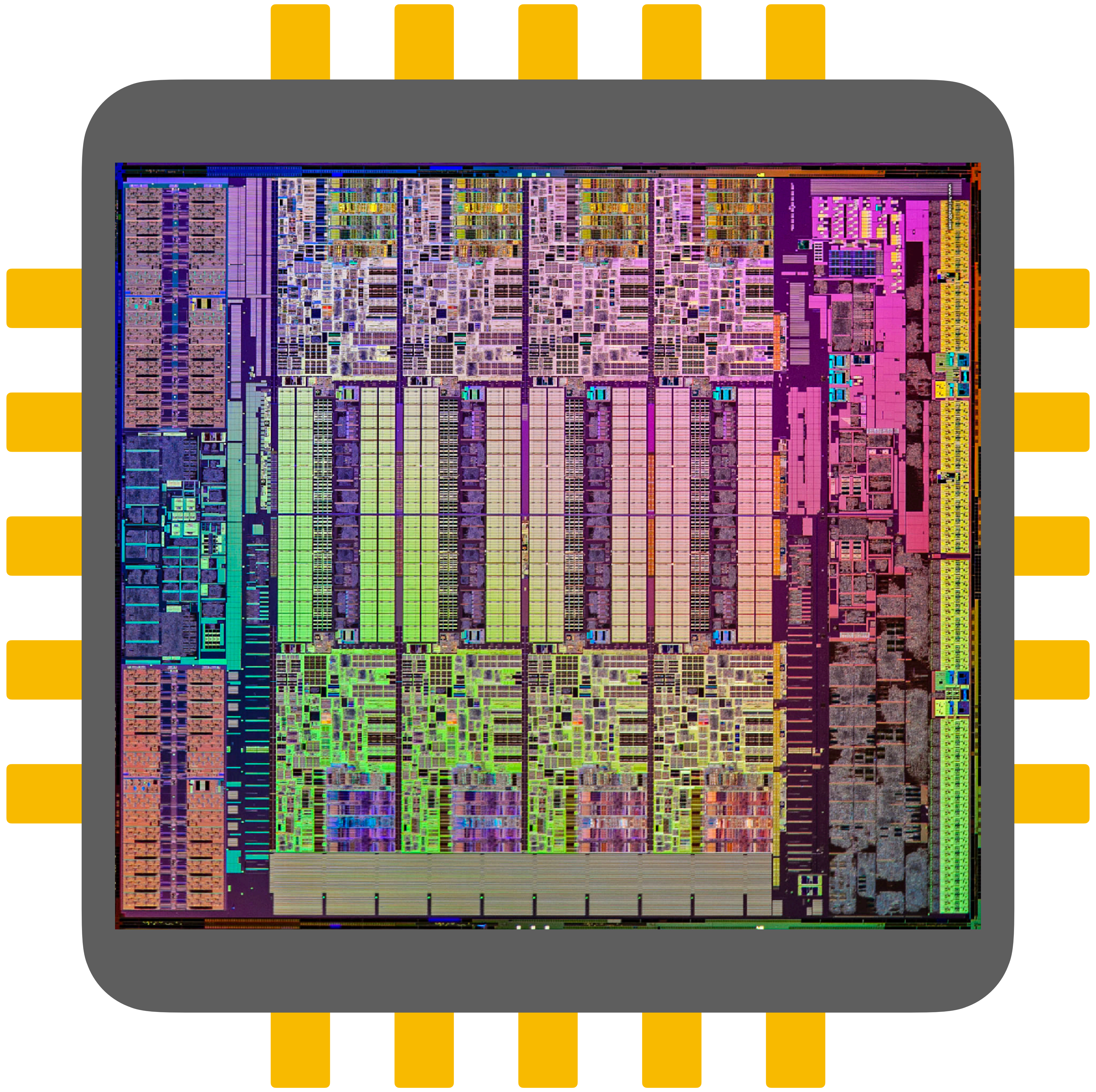
Picture of Intel “Haswell-E” Eight Core CPU



Picture of Intel “Haswell-E” Eight Core CPU



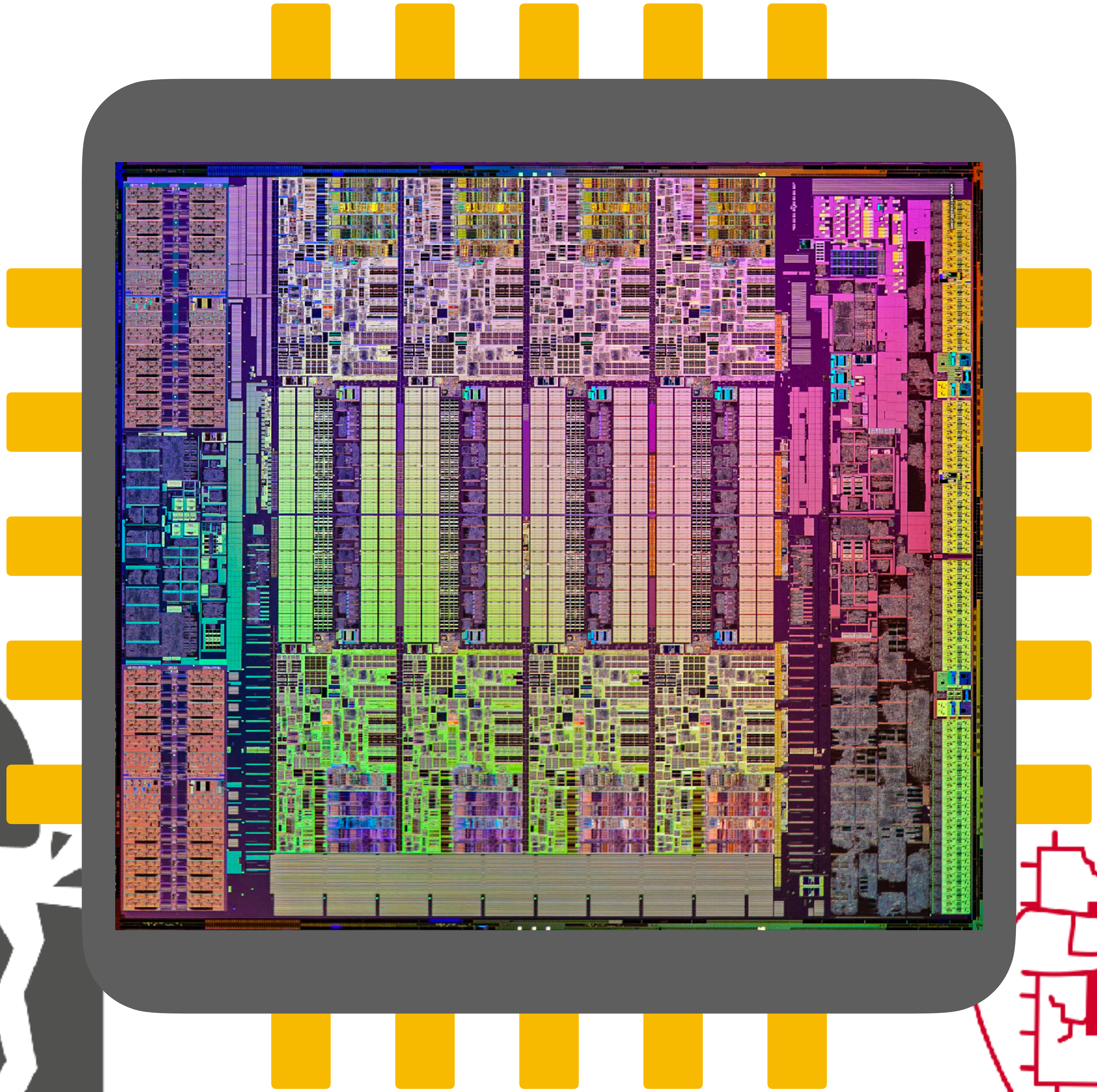
Picture of Intel “Haswell-E” Eight Core CPU



Picture of Intel “Haswell-E” Eight Core CPU

SPECTRE

MELTDOWN



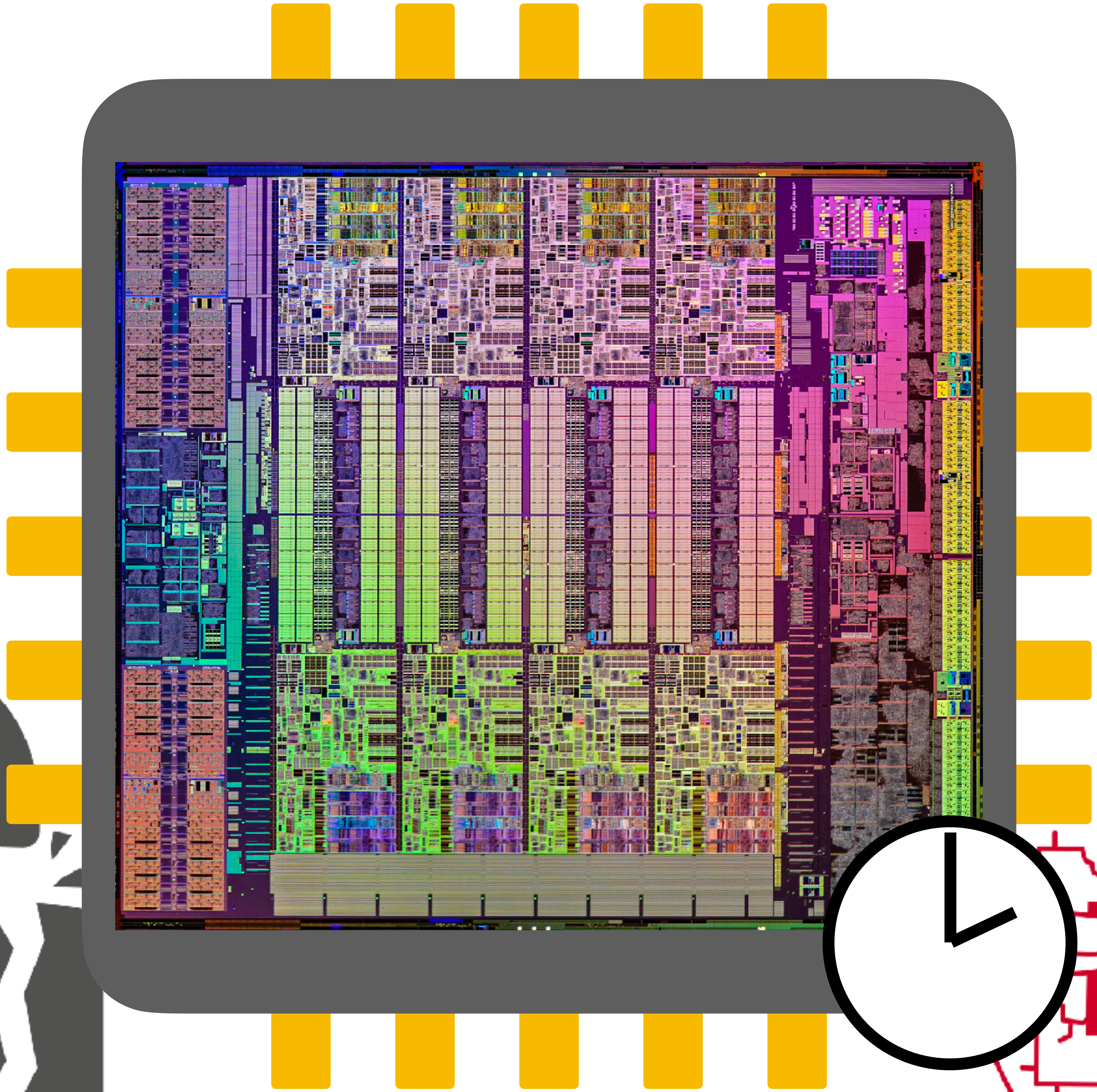
FORESHADOW

MDS

Picture of Intel “Haswell-E” Eight Core CPU

SPECTRE

MELTDOWN



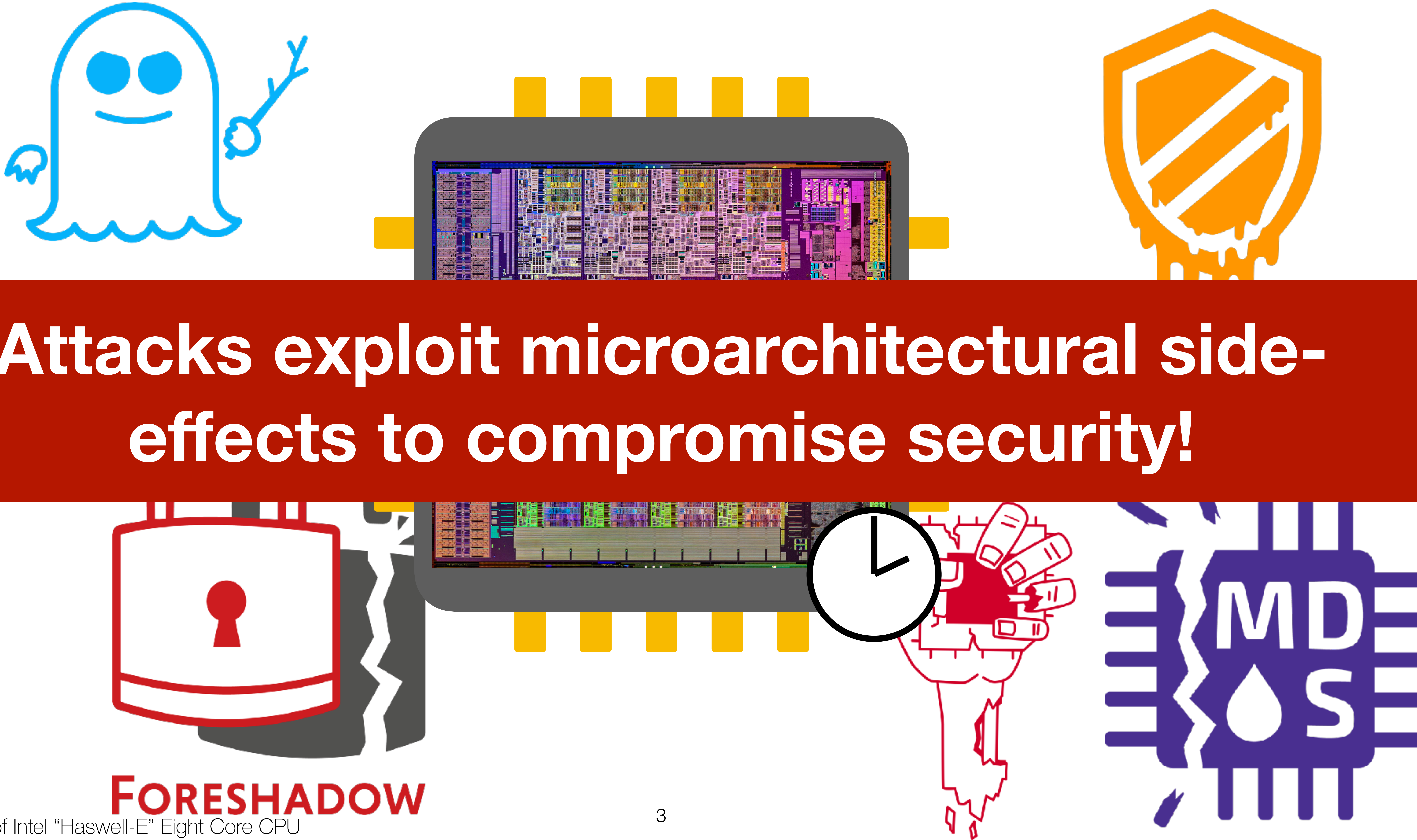
FORESHADOW

MDS

Picture of Intel “Haswell-E” Eight Core CPU

Attacks exploit microarchitectural side-effects to compromise security!

Picture of Intel “Haswell-E” Eight Core CPU



Attacks exploit microarchitectural side-effects to compromise security!

Picture of Intel “Haswell-E” Eight Core CPU

it microarchitect[...] side-compromise s[...]ity!

The New York Times
Researchers Discover Two Major Flaws in the World's Computers

The New York Times
Intel Fixes a Security Flaw It S[...] Repaired 6 Months Ago

BBC
Intel Zombieload bug fix to slow data centre computers

BBC
Meltdown fix can make some machines slower - Intel
Intel has admitted that patches to fix the Spectre and Meltdown chip flaws could slow machines "in some cases"

FOX BUSINESS
WASHINGTON, D.C.
NEWS ALERT
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
WINTER STORM
@FOXB
DEVELOPING STORY
COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
LIVE
CNN
DAX ▲164.69
NEWS STREAM
MDS

Picture c

The New York Times

Researchers Discover Two Major Flaws in the World's Computers

NEWS ALERT

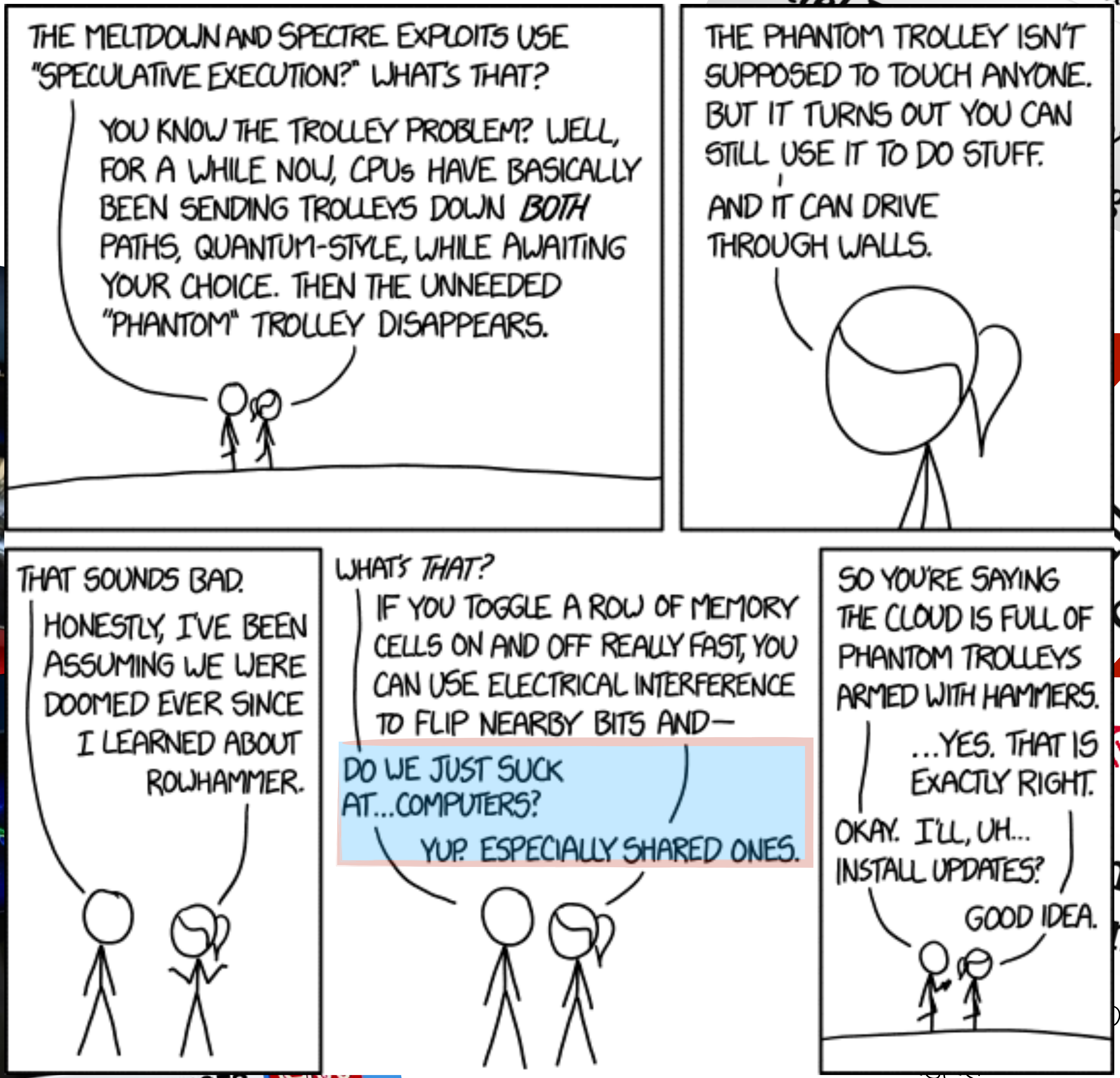
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA

DEVELOPING STORY

COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES

DAX ▲164.69

NEWS STREAM


THE MELTDOWN AND SPECTRE EXPLOITS USE "SPECULATIVE EXECUTION?" WHAT'S THAT?
YOU KNOW THE TROLLEY PROBLEM? WELL, FOR A WHILE NOW, CPUs HAVE BASICALLY BEEN SENDING TROLLEYS DOWN BOTH PATHS, QUANTUM-STYLE, WHILE AWAITING YOUR CHOICE. THEN THE UNNEEDED "PHANTOM" TROLLEY DISAPPEARS.
THE PHANTOM TROLLEY ISN'T SUPPOSED TO TOUCH ANYONE. BUT IT TURNS OUT YOU CAN STILL USE IT TO DO STUFF.
AND IT CAN DRIVE THROUGH WALLS.
THAT SOUNDS BAD.
HONESTLY, I'VE BEEN ASSUMING WE WERE DOOMED EVER SINCE I LEARNED ABOUT ROWHAMMER.
WHAT'S THAT?
IF YOU TOGGLE A ROW OF MEMORY CELLS ON AND OFF REALLY FAST, YOU CAN USE ELECTRICAL INTERFERENCE TO FLIP NEARBY BITS AND—
DO WE JUST SUCK AT...COMPUTERS?
YUP. ESPECIALLY SHARED ONES.
SO YOU'RE SAYING THE CLOUD IS FULL OF PHANTOM TROLLEYS ARMED WITH HAMMERS.
...YES. THAT IS EXACTLY RIGHT.
OKAY. I'LL, UH... INSTALL UPDATES?
GOOD IDEA.

The New York Times

Intel

Flaw It

bieload bug fix to slow data computers

ake some ntel

fix the could slow

MDS

side-

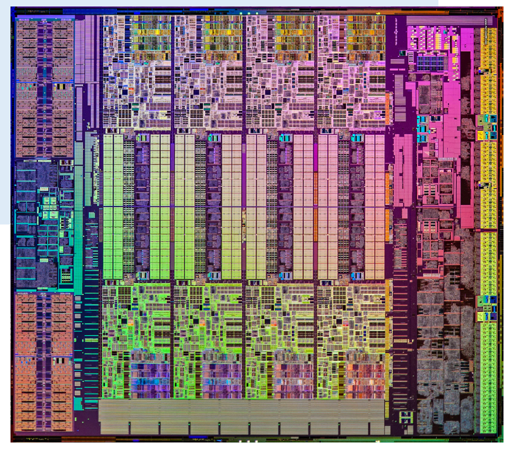
ity!

Picture c

What is the problem?

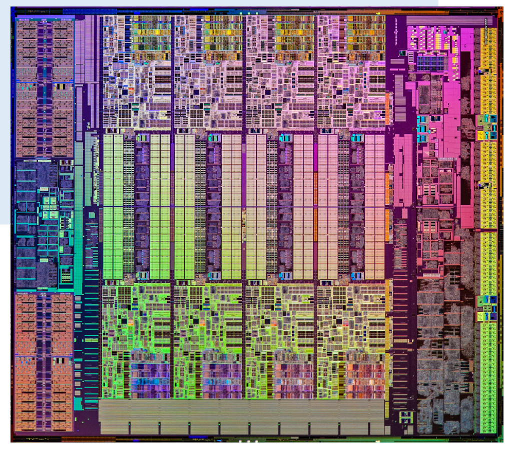
What is the problem?

Microarchitectural leakage depends on ***specific hardware details***



What is the problem?

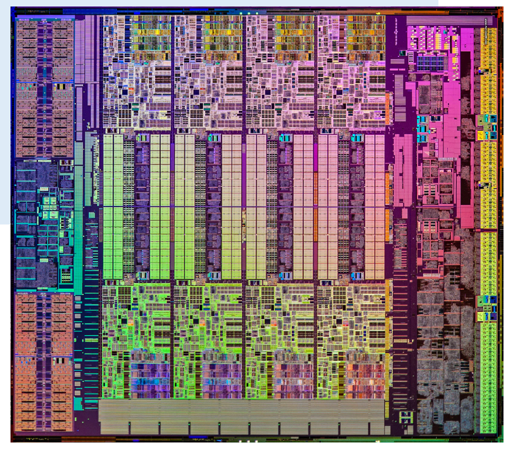
Microarchitectural leakage depends on ***specific hardware details***



No ***faithful***, ***precise*** models capturing ***microarchitectural leakage***

What is the problem?

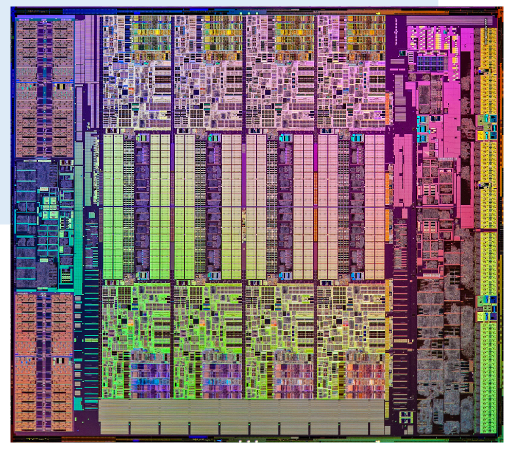
Microarchitectural leakage depends on ***specific hardware details***



No ***faithful***, ***precise*** models capturing ***microarchitectural leakage***

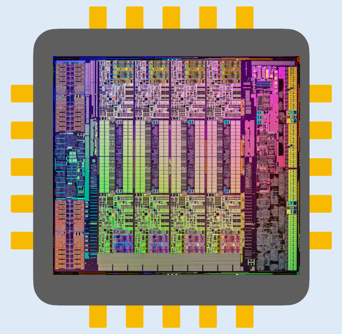
What is the problem?

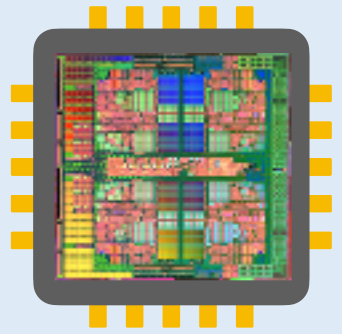
Microarchitectural leakage depends on ***specific hardware details***



No ***faithful***, ***precise*** models capturing ***microarchitectural leakage***

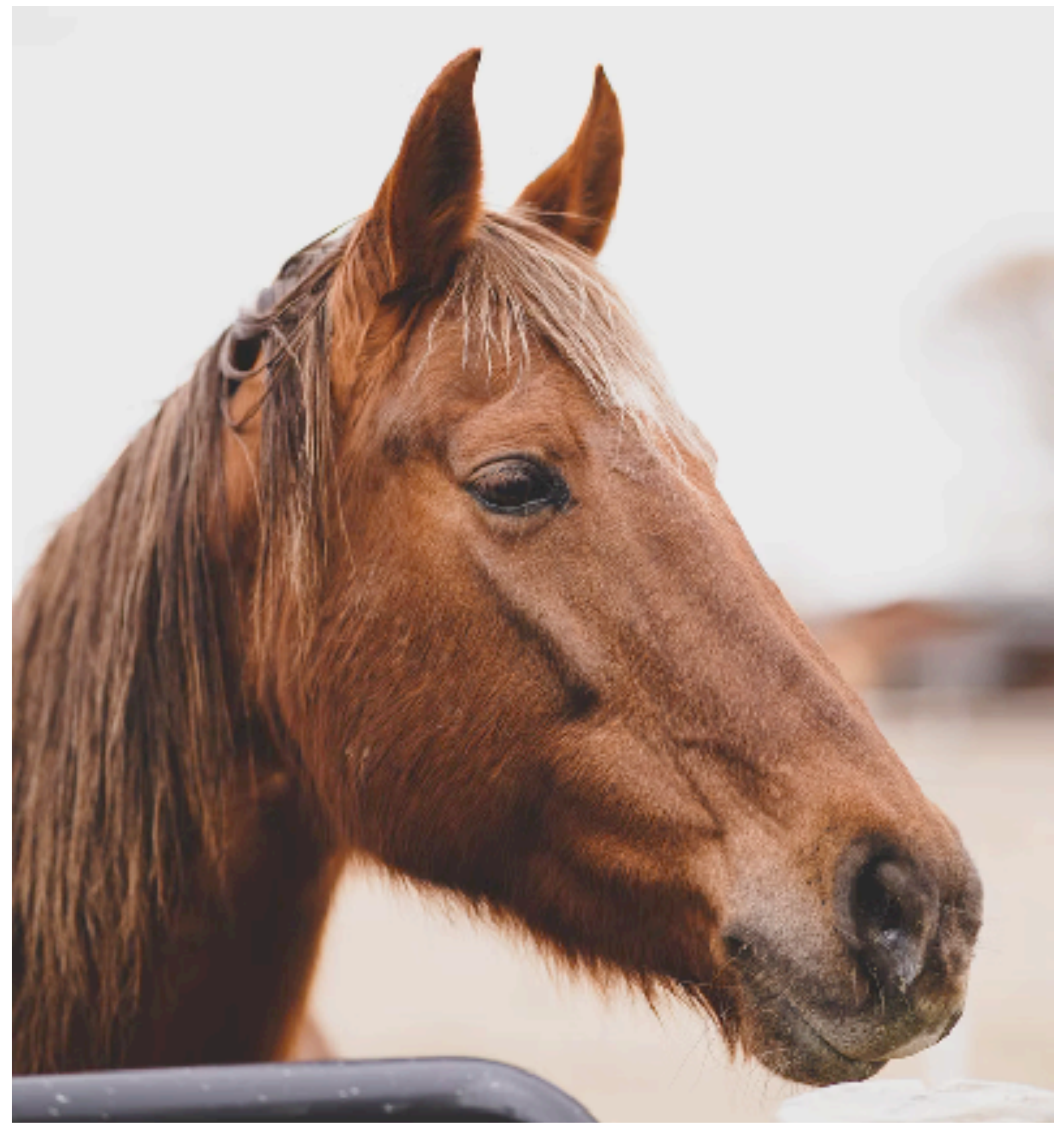
Writing secure code is almost **impossible**

P +  = **Secure**

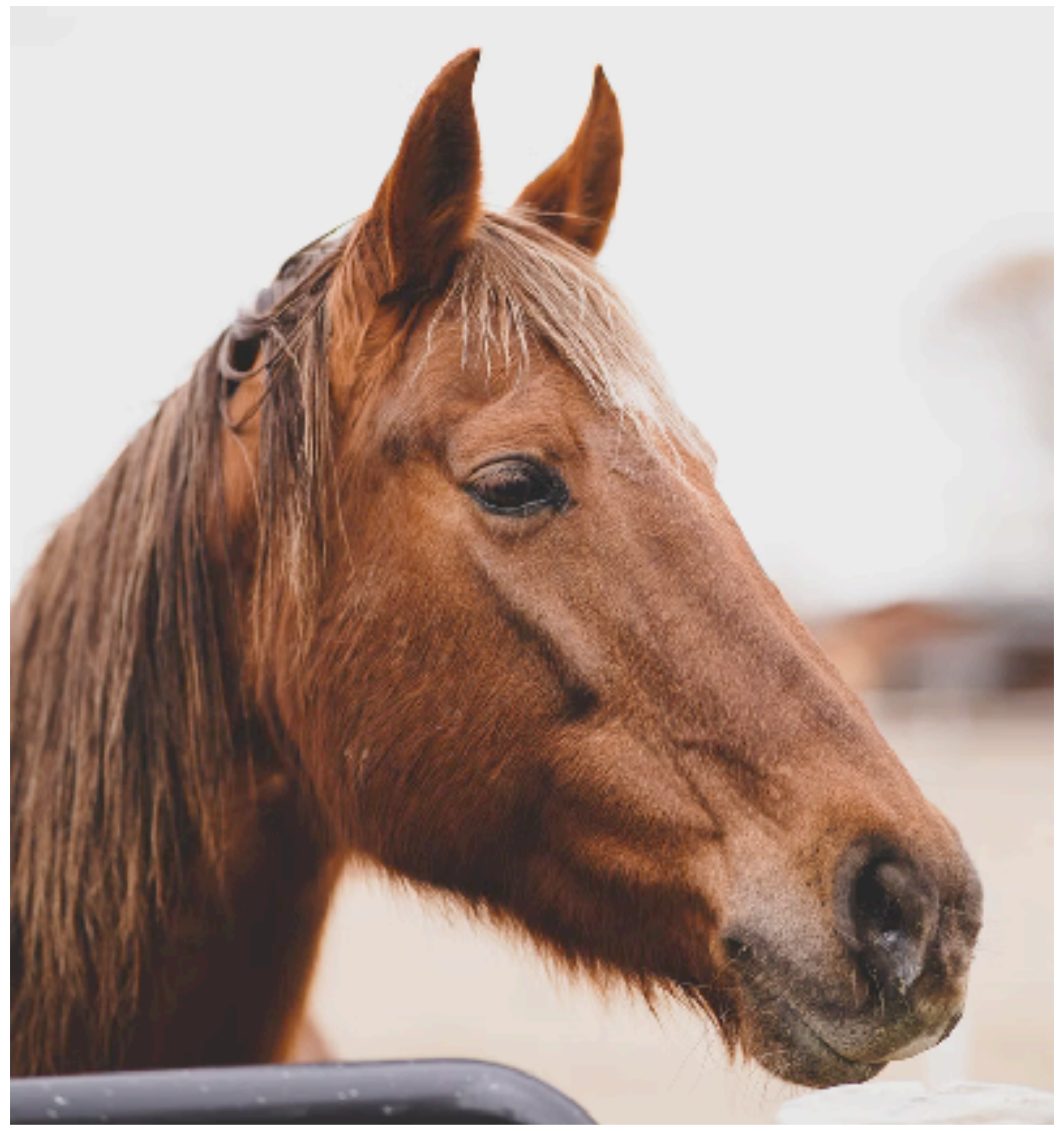
P +  = **Insecure**

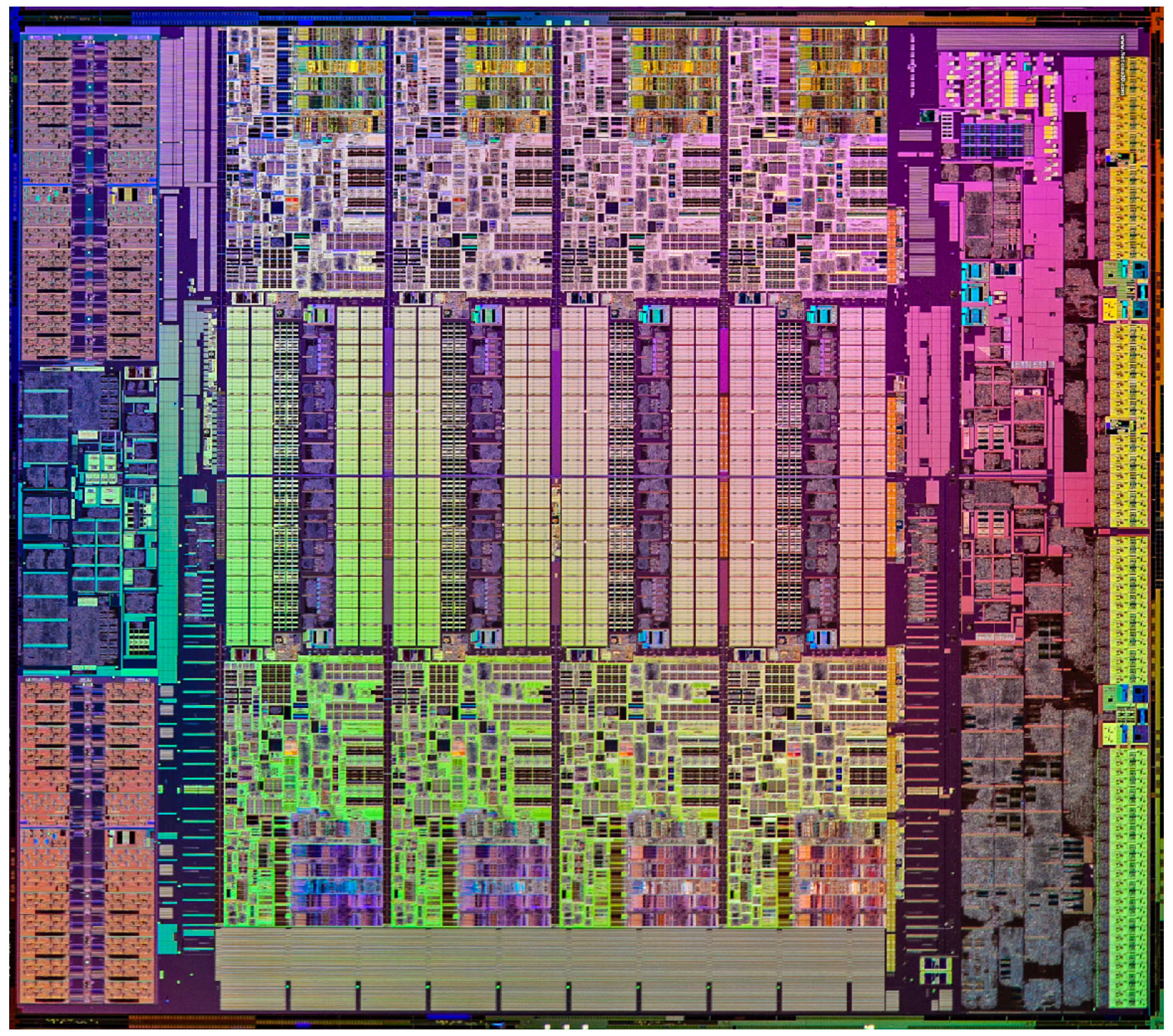
A problem of (missing) abstractions

A problem of (missing) abstractions



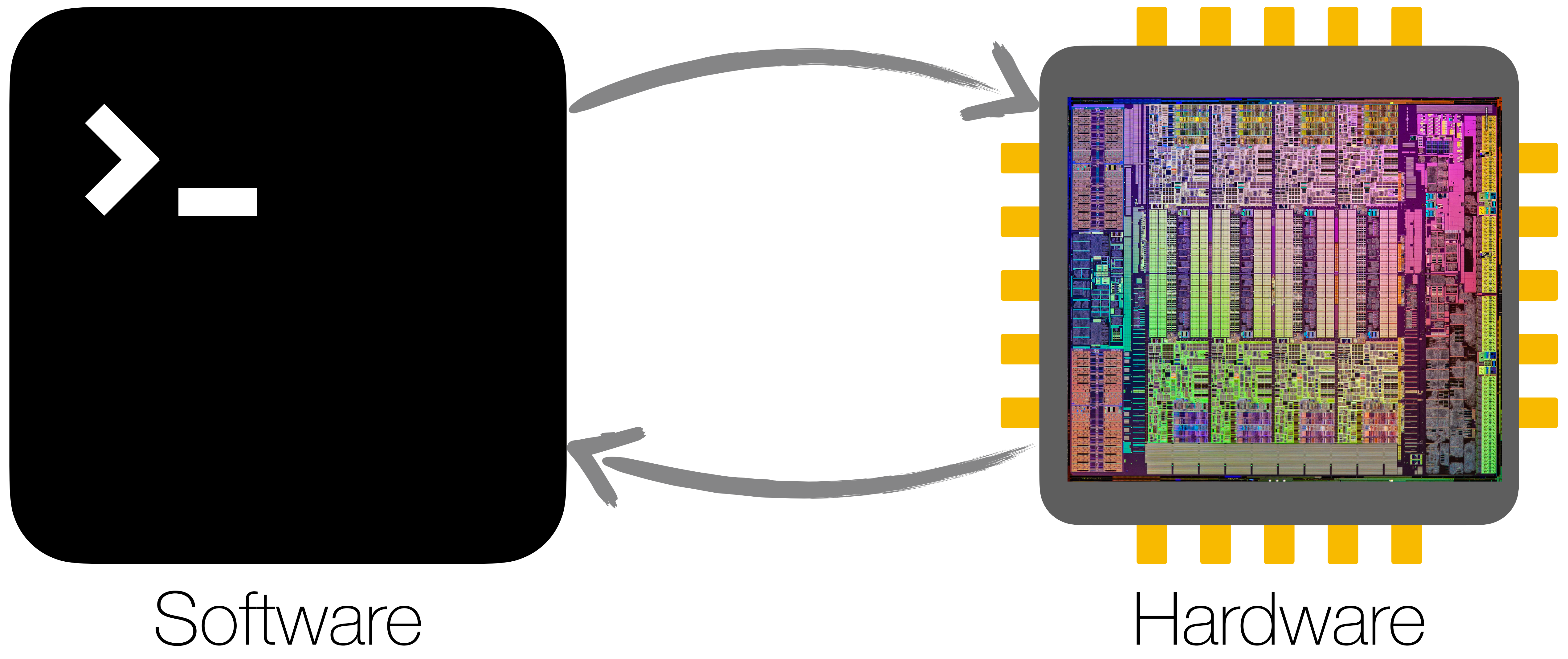
A problem of (missing) abstractions





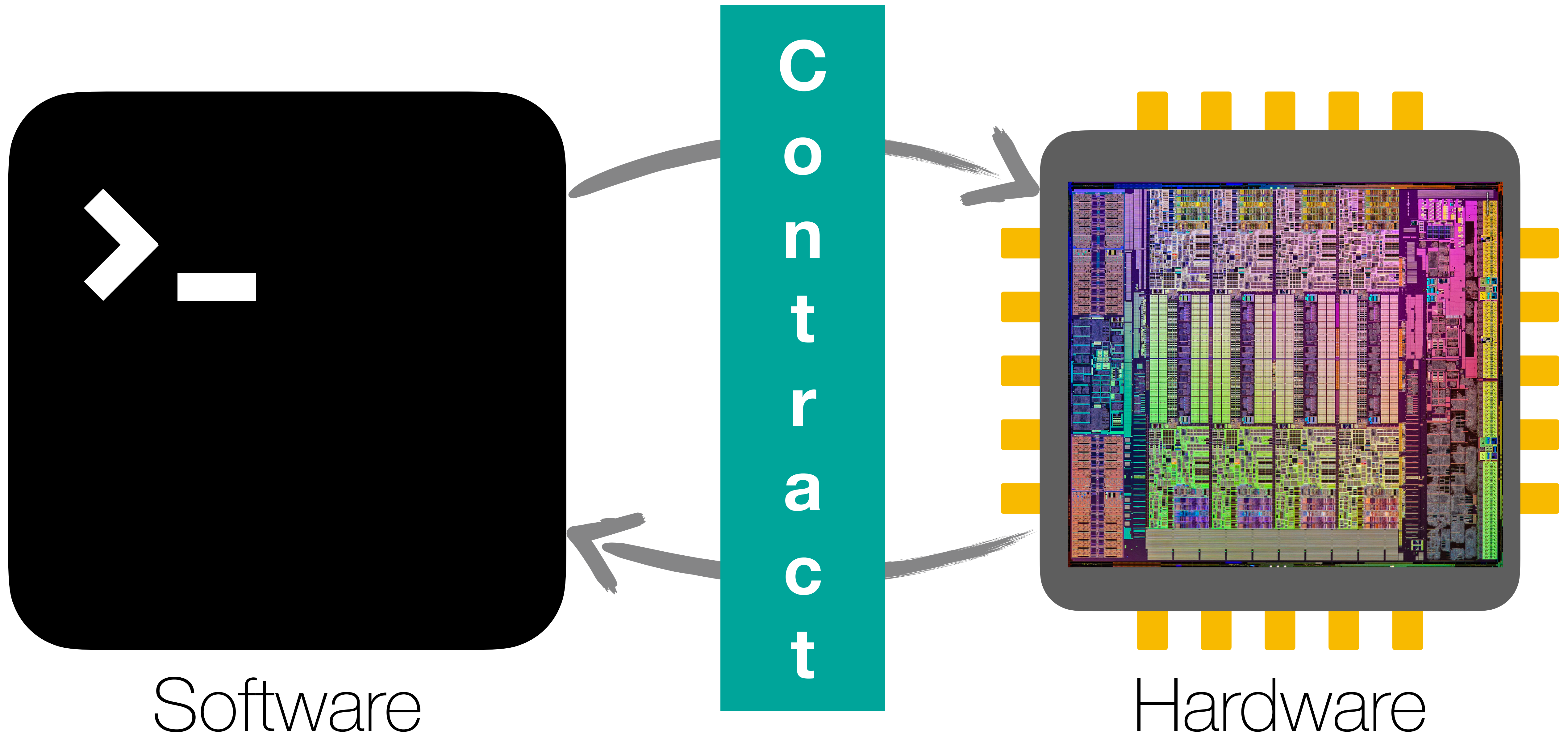
🧐???

What is a good abstraction?



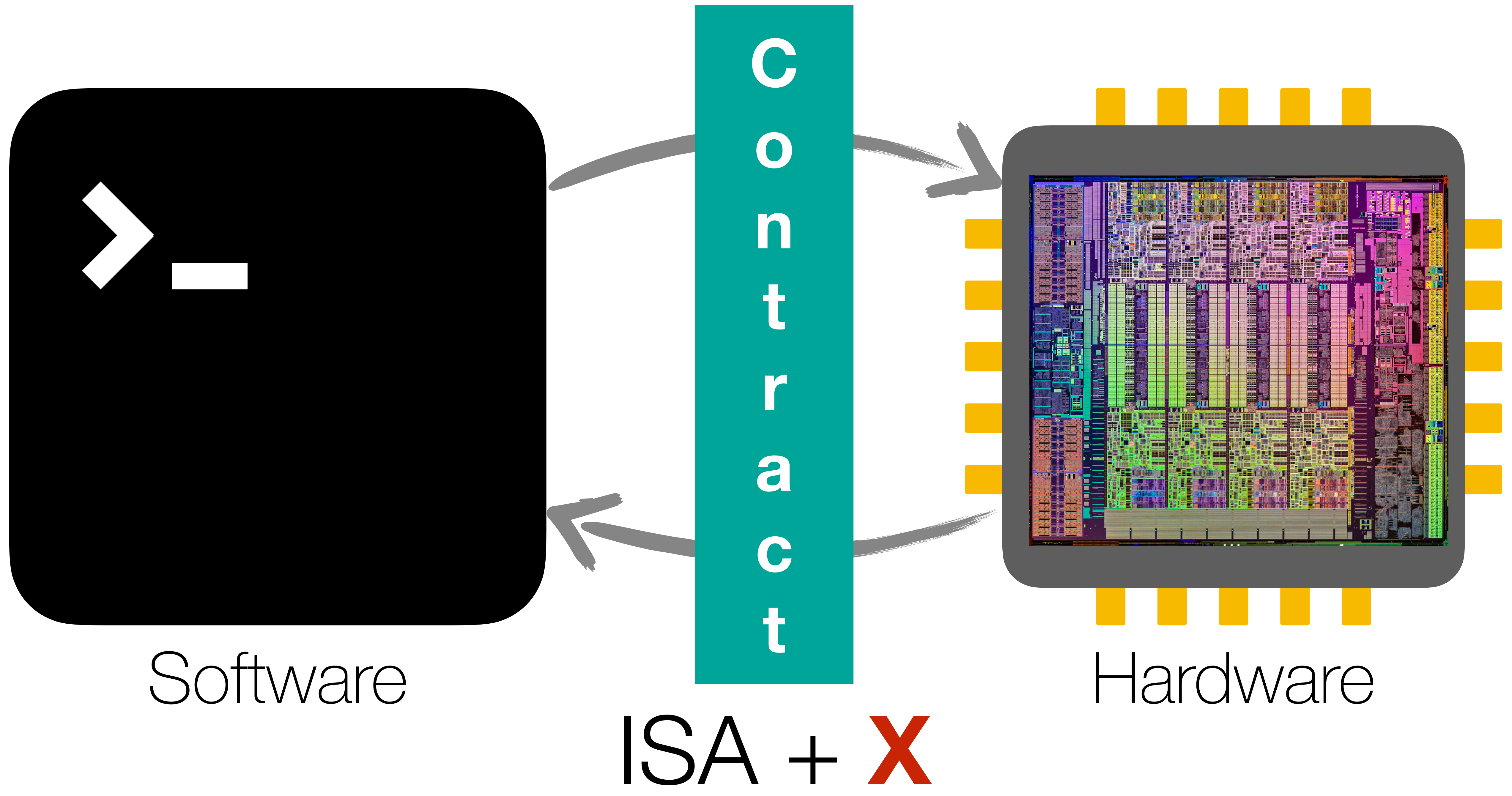
What is a good abstraction?

Hardware-software **contracts** for ***security***



What is a good abstraction?

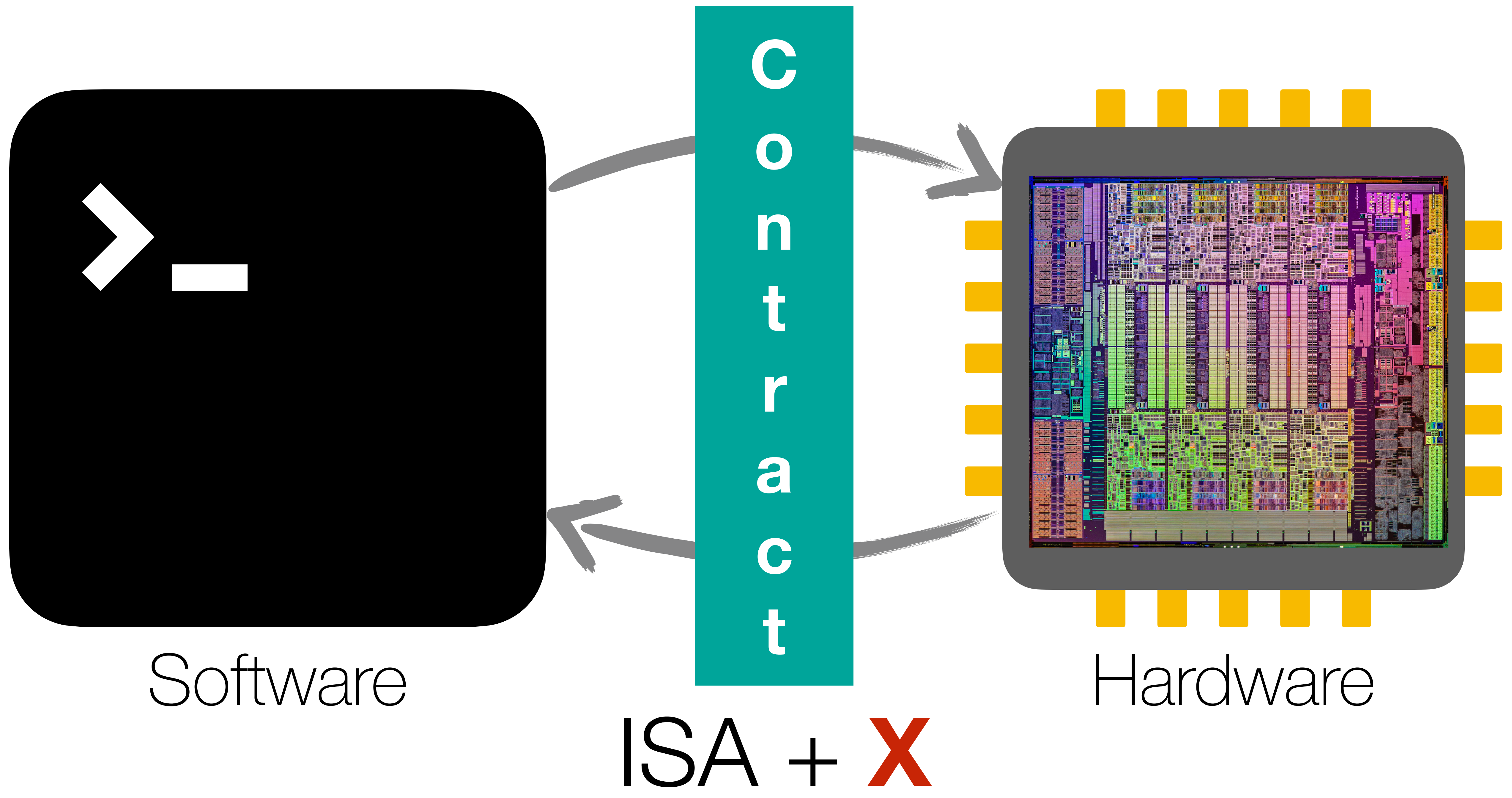
Hardware-software **contracts** for ***security***



What is a good abstraction?

Hardware-software **contracts** for ***security***

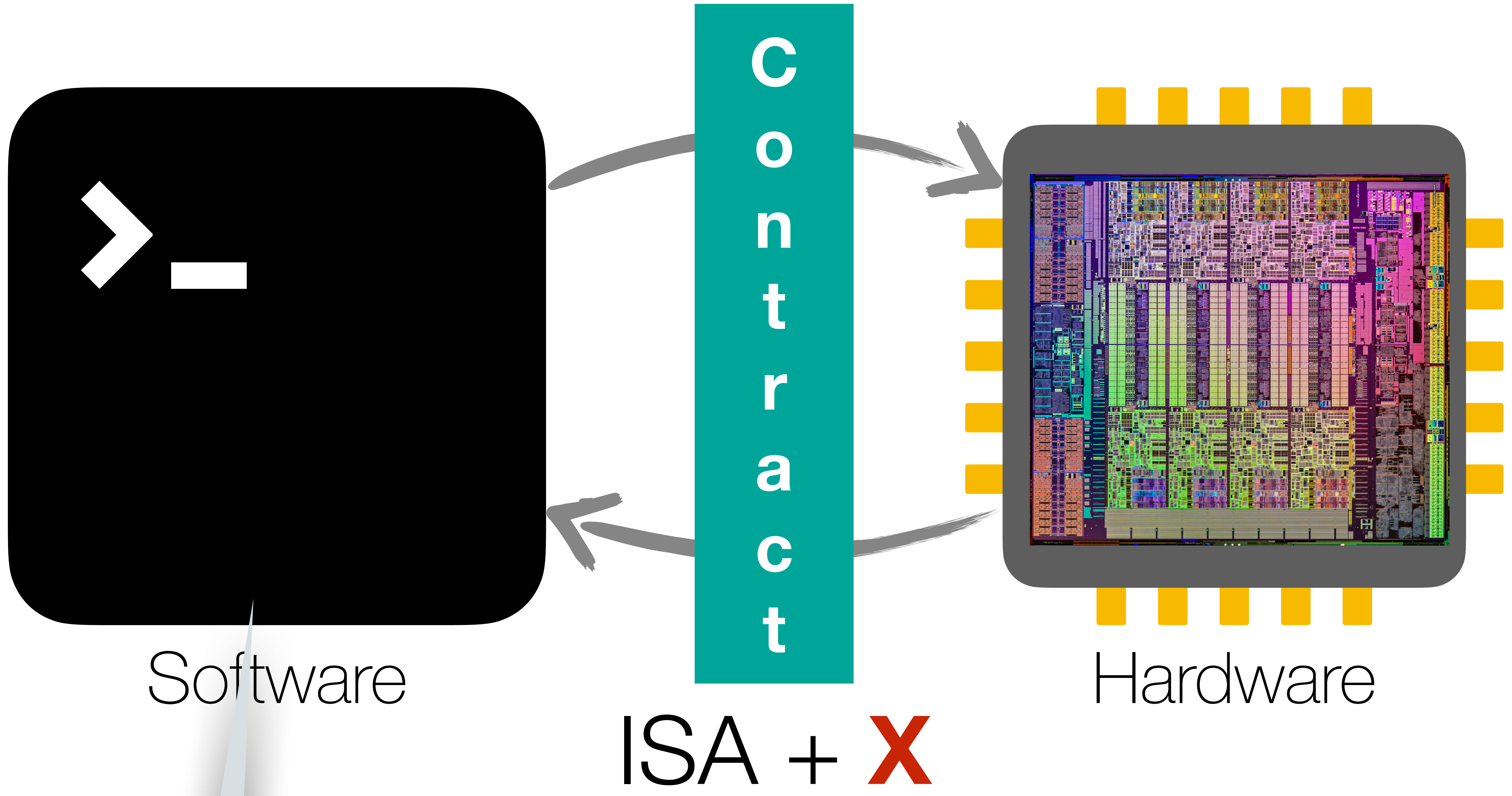
Capture all possible ***microarchitectural leaks***!



What is a good abstraction?

Hardware-software **contracts** for ***security***

Capture all possible ***microarchitectural leaks***!

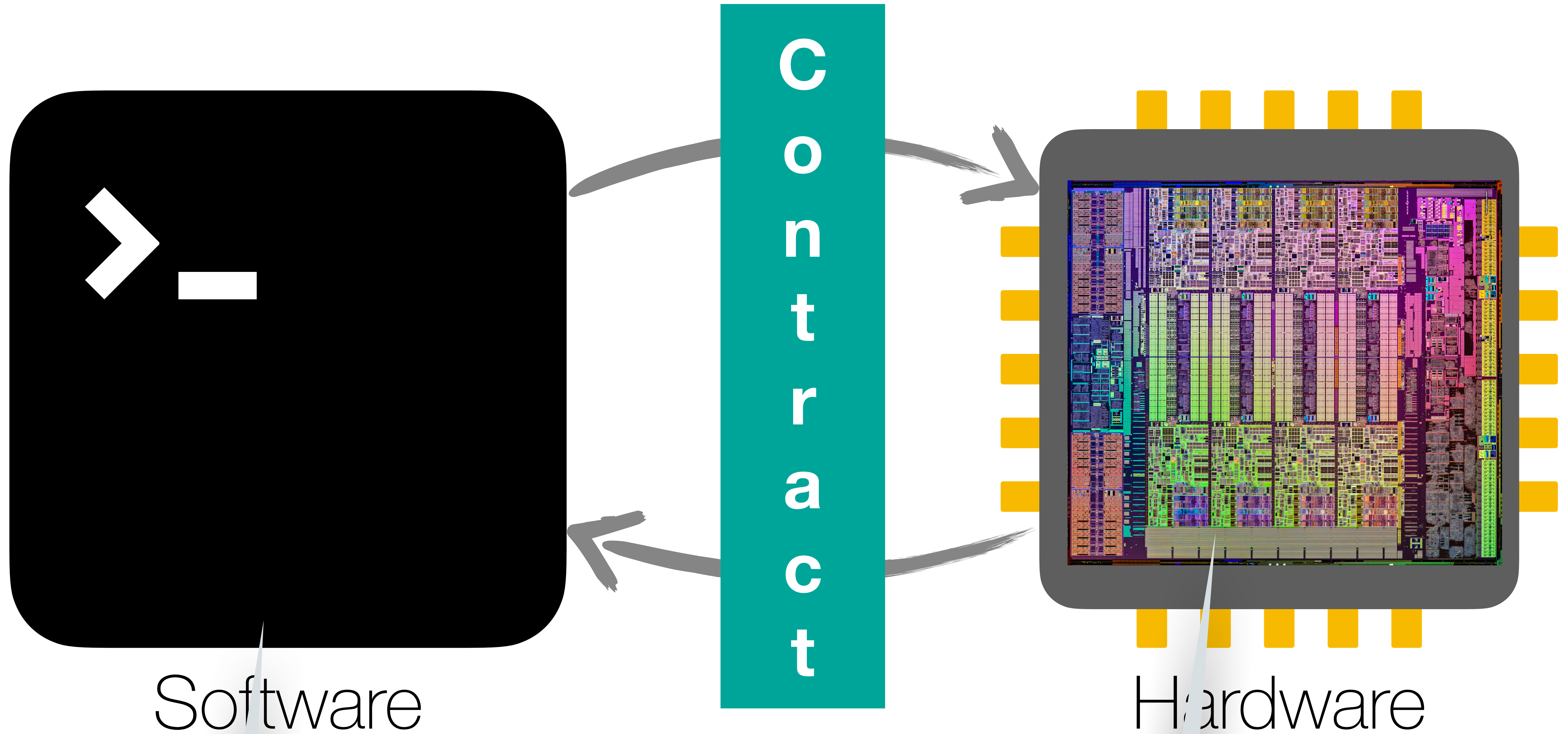


Secure programming **independently** of specific microarchitecture

What is a good abstraction?

Hardware-software **contracts** for ***security***

Capture all possible ***microarchitectural leaks***!



ISA + **X**

Secure programming **independently** of specific microarchitecture

Implement ***optimizations*** **compliant** with contract

In this talk

In this talk

HW/SW contracts for ***secure speculation***

In this talk

HW/SW contracts for ***secure speculation***

Contracts + ***Hardware***

In this talk

HW/SW contracts for ***secure speculation***

Contracts + ***Hardware***

Contracts + ***Software***

Outline

1. Speculative execution attacks
2. Modeling speculative leaks
3. Hardware-software contracts for secure speculation
4. What about hardware?
5. What about software?
6. Conclusions

Outline

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]]
```

HELP

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]]
```

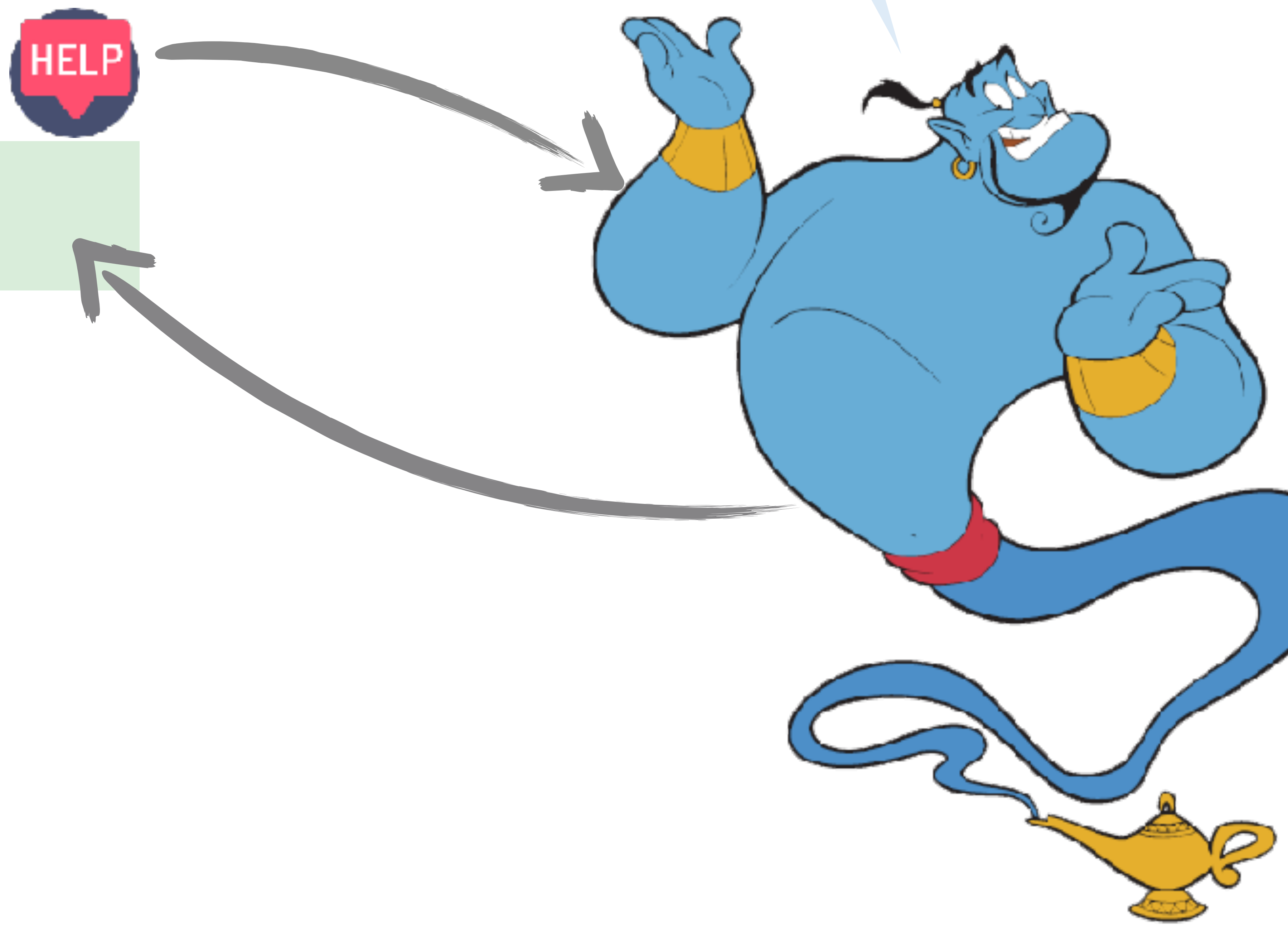
Branch predictor

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]]
```

HELP

Branch predictor

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

✓ Architectural (ISA) state

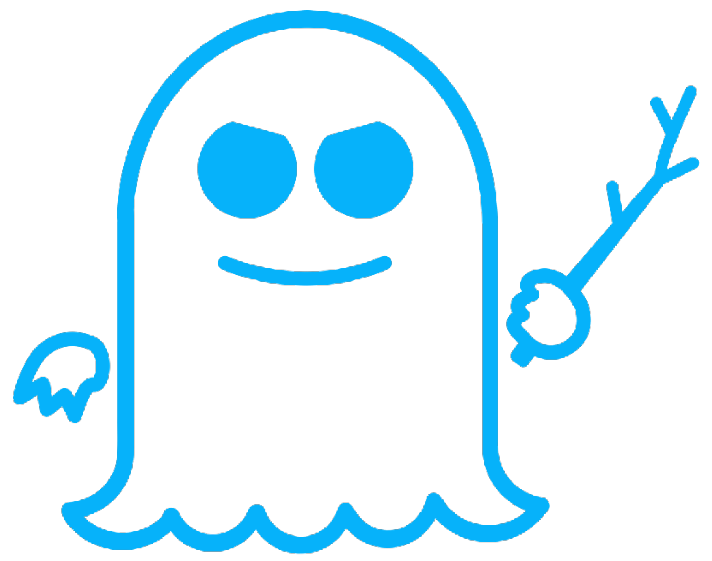
✗ Microarchitectural state

Spectre v1

Spectre v1

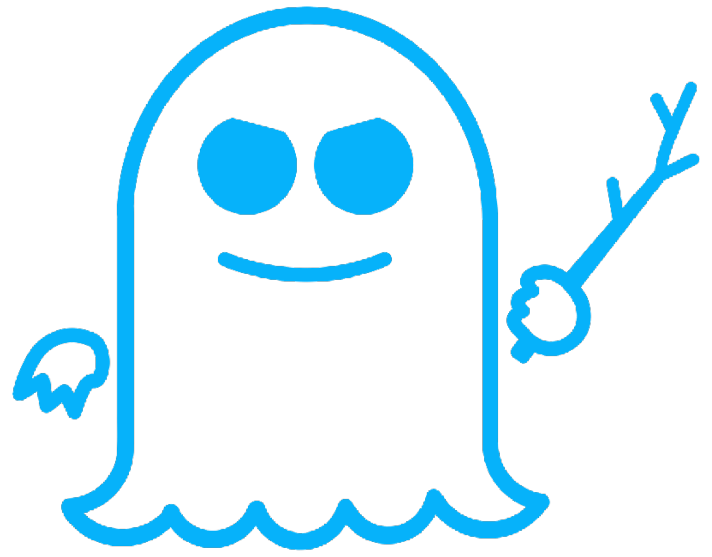
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

Spectre v1

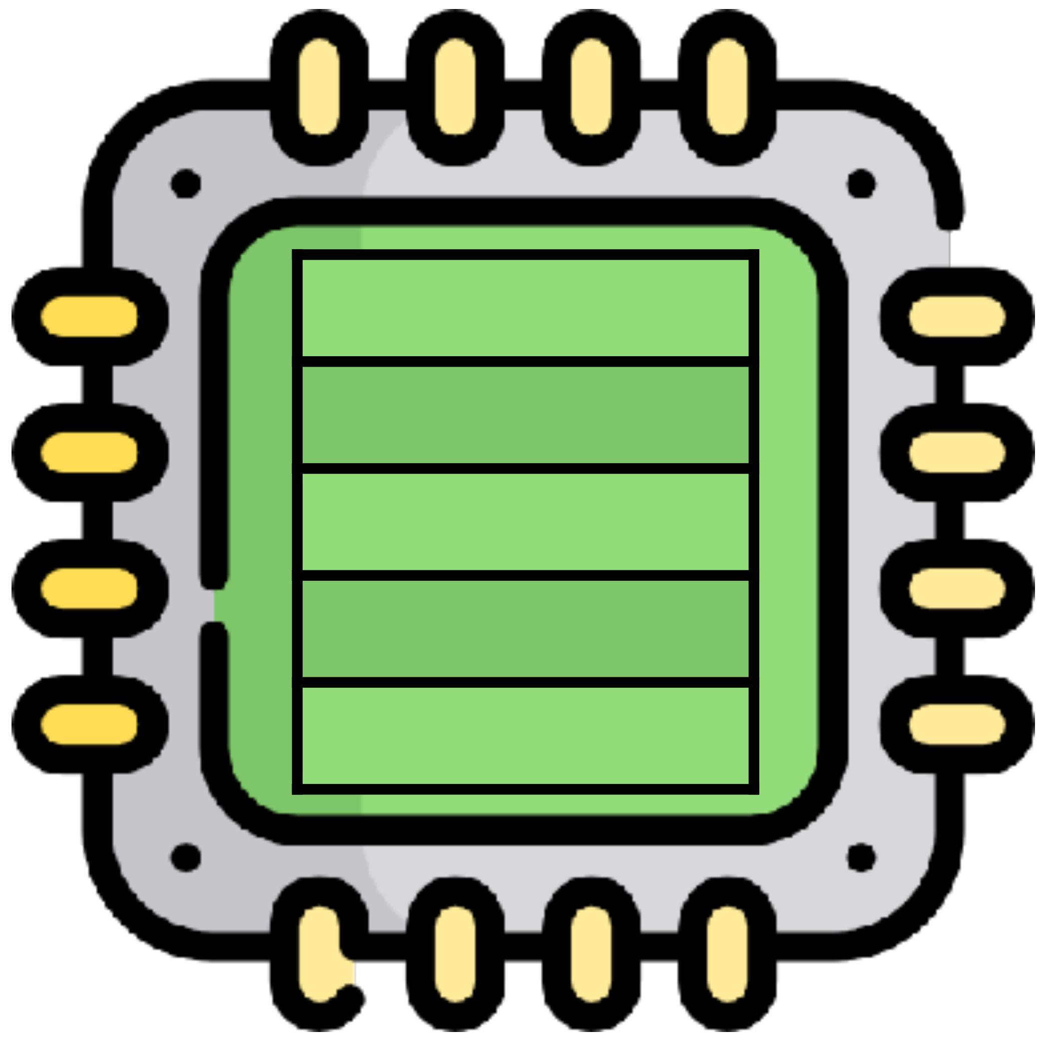


```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

Spectre v1



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

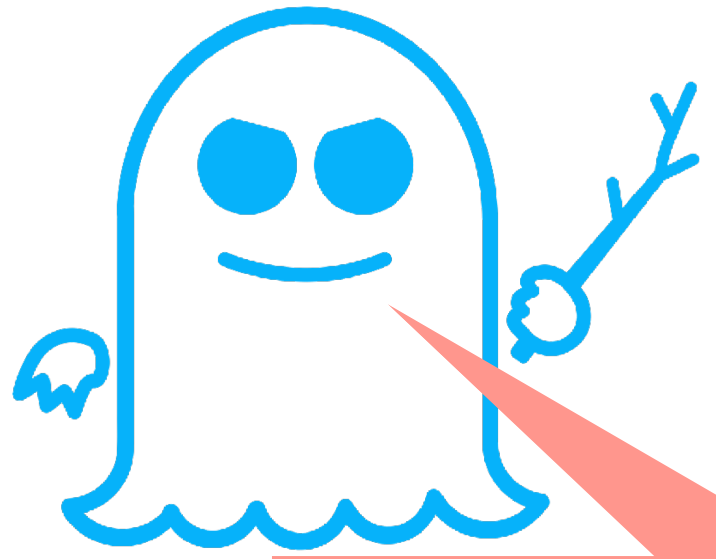


Spectre v1

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

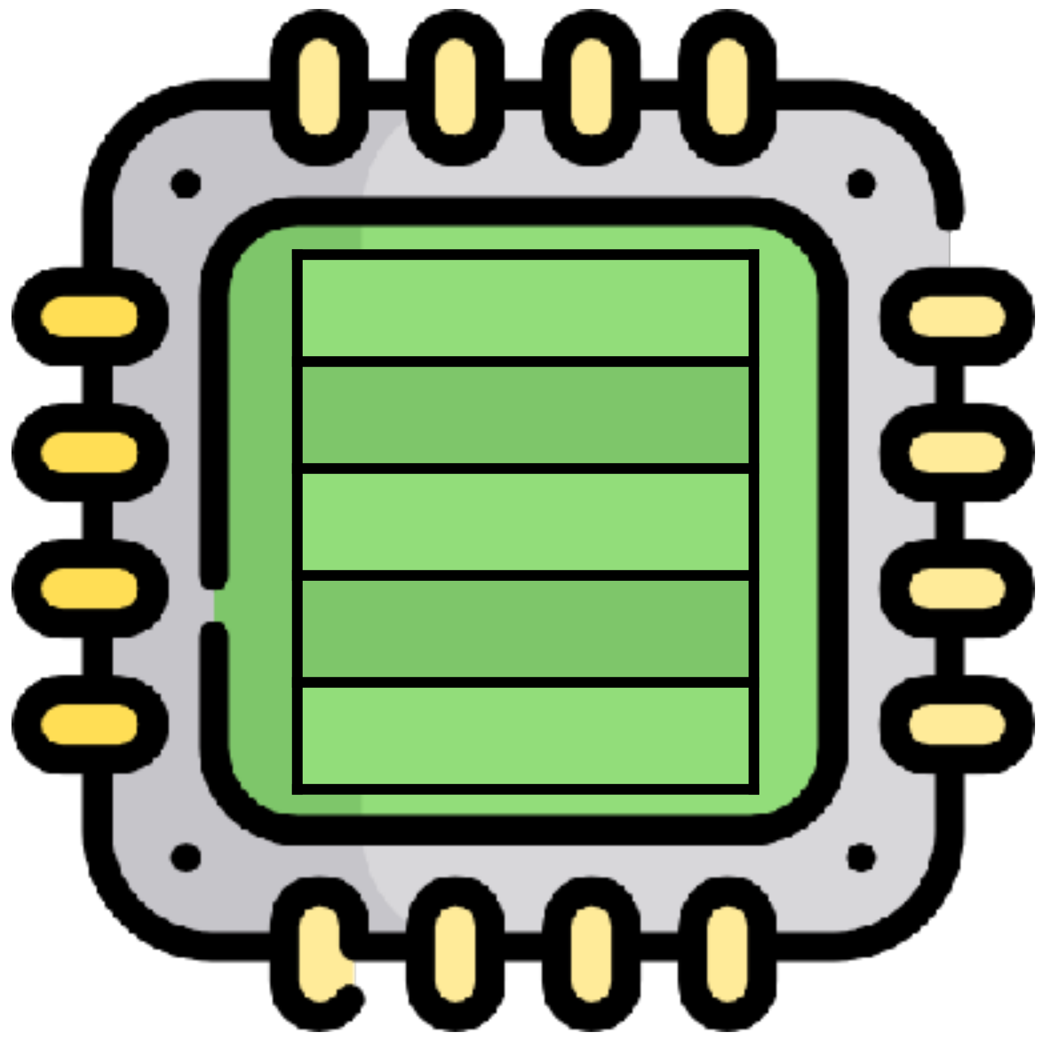
A_size=16

B	B[0]	B[1]	...



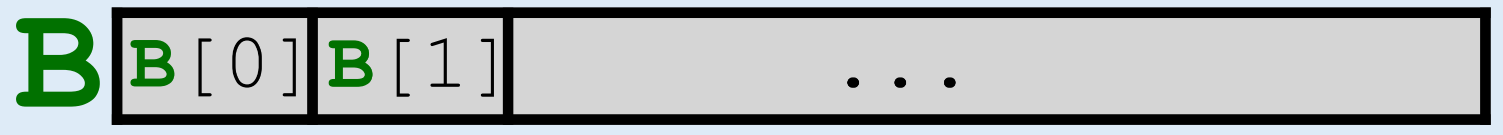
Secret data

What is in A[128]?

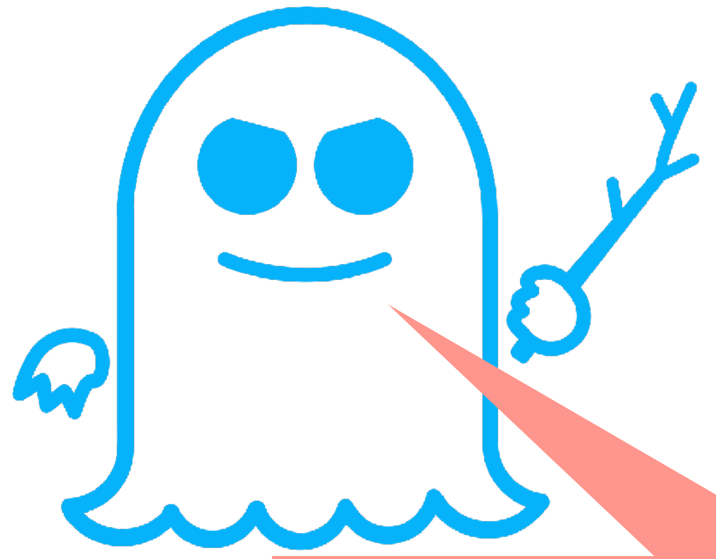


Spectre v1

A_size=16



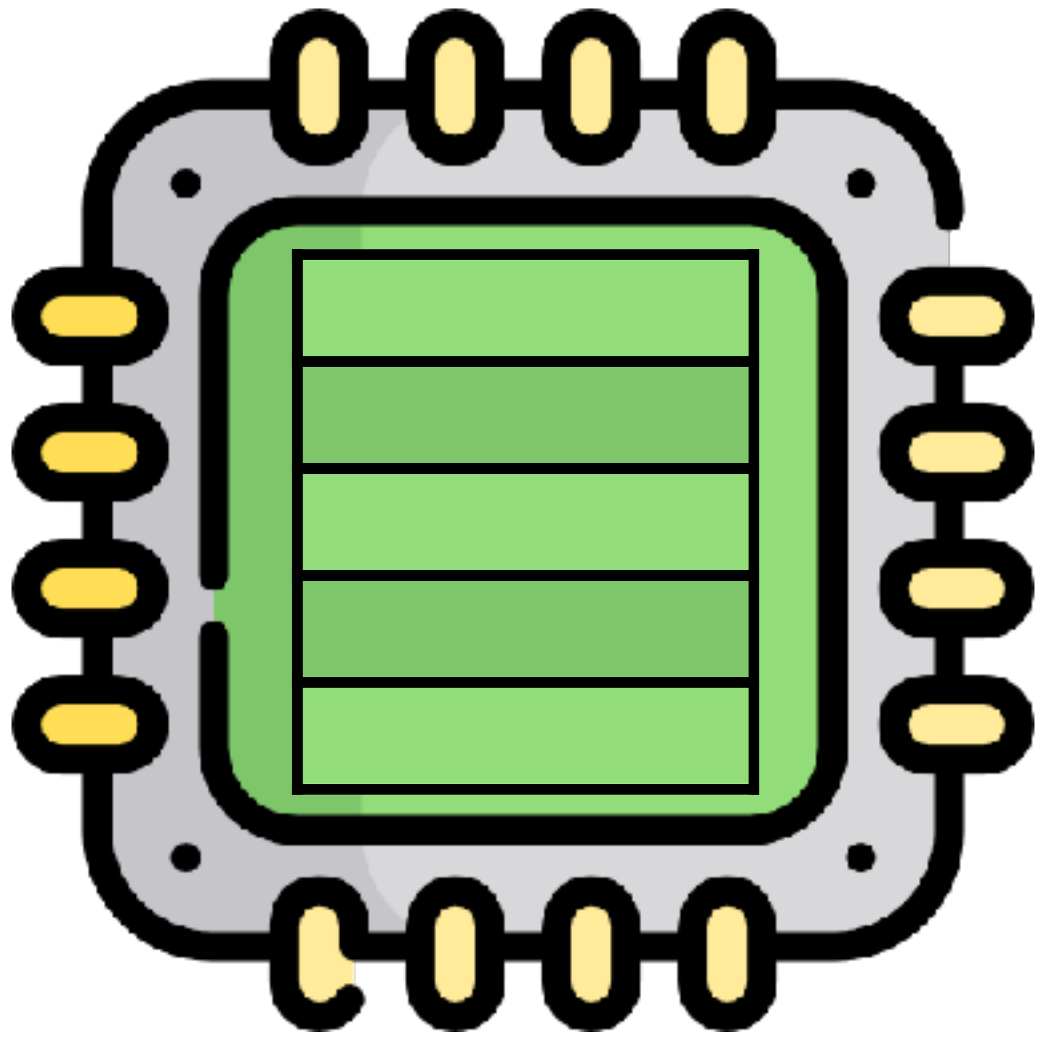
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



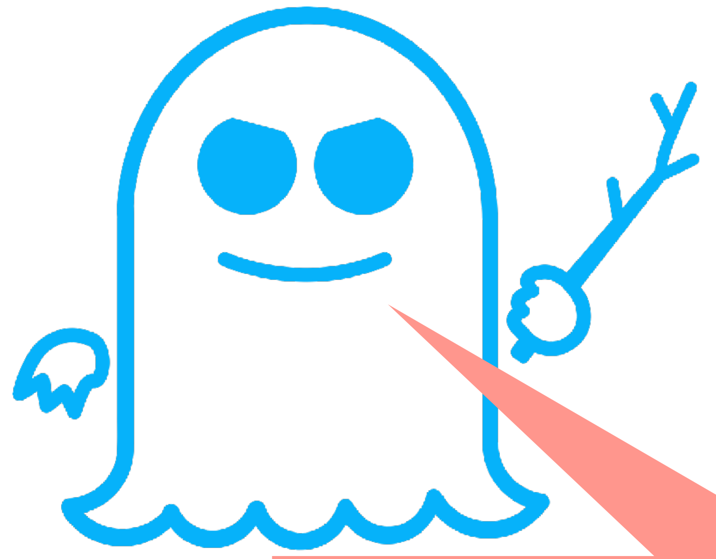
Secret data

What is in A[128]?

1) Train branch predictor



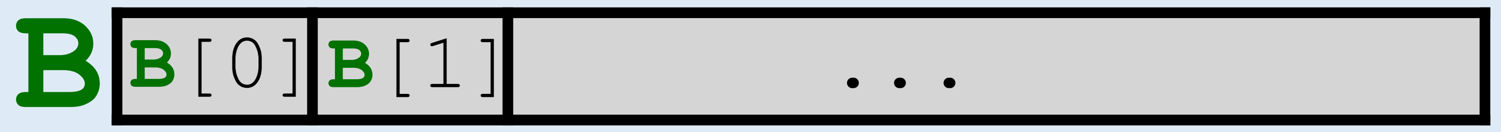
Spectre v1



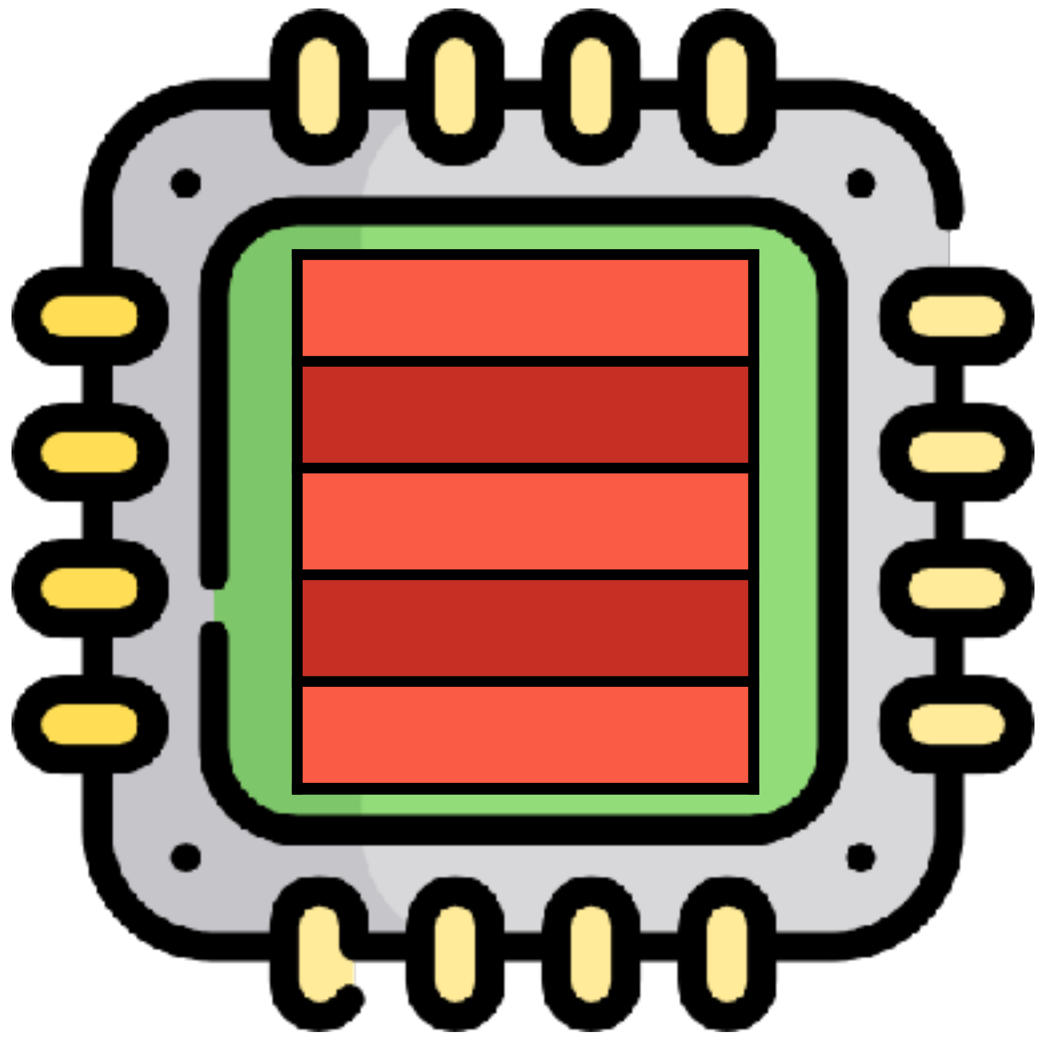
Secret data

What is in A[128]?

A_size=16



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

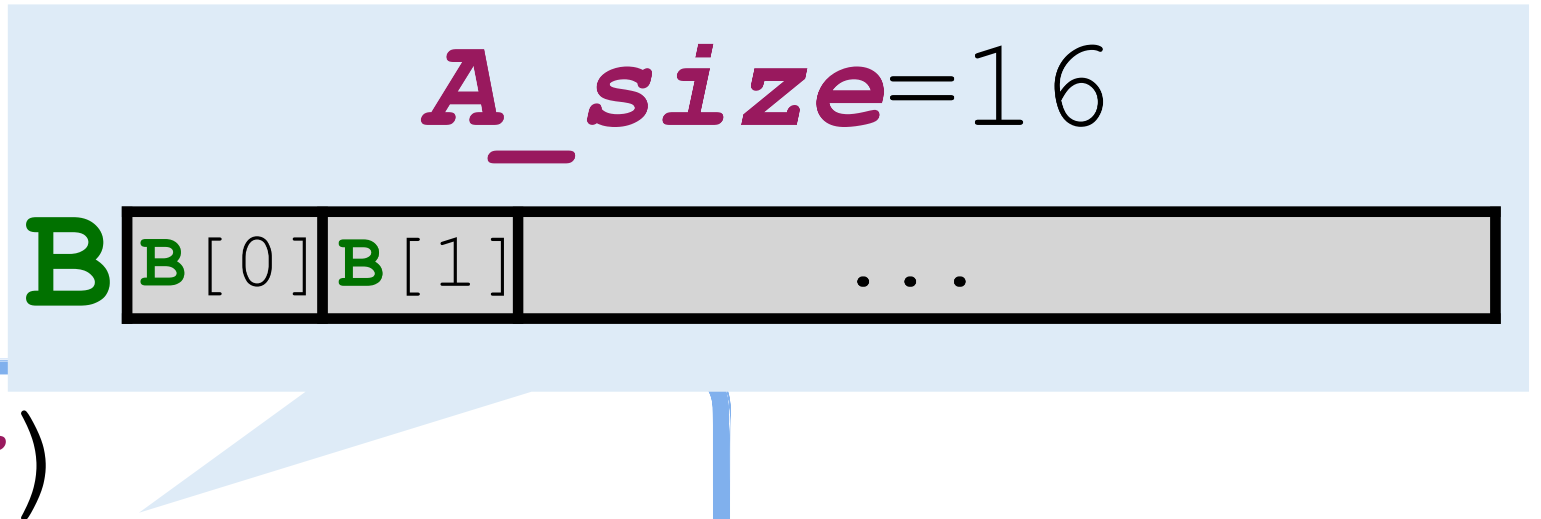


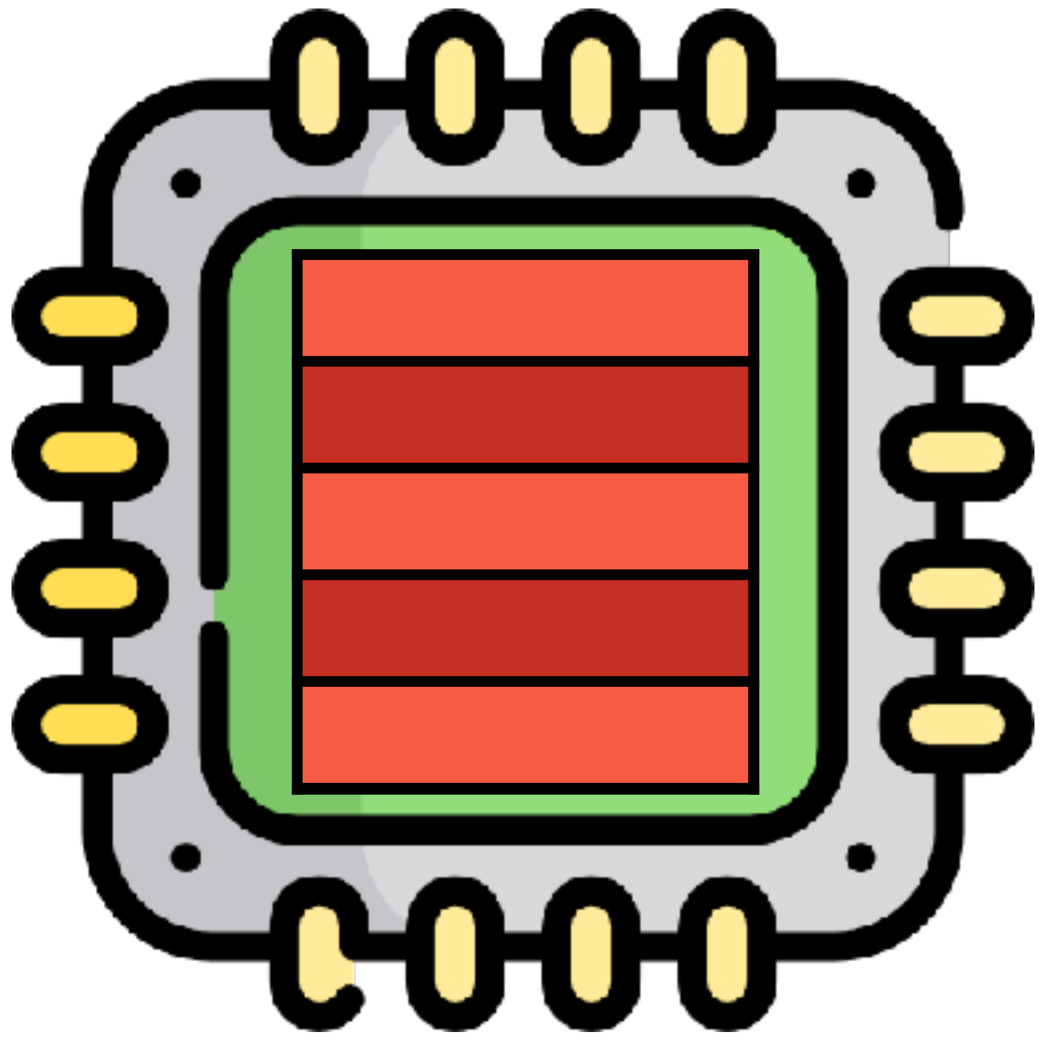
1) Train branch predictor

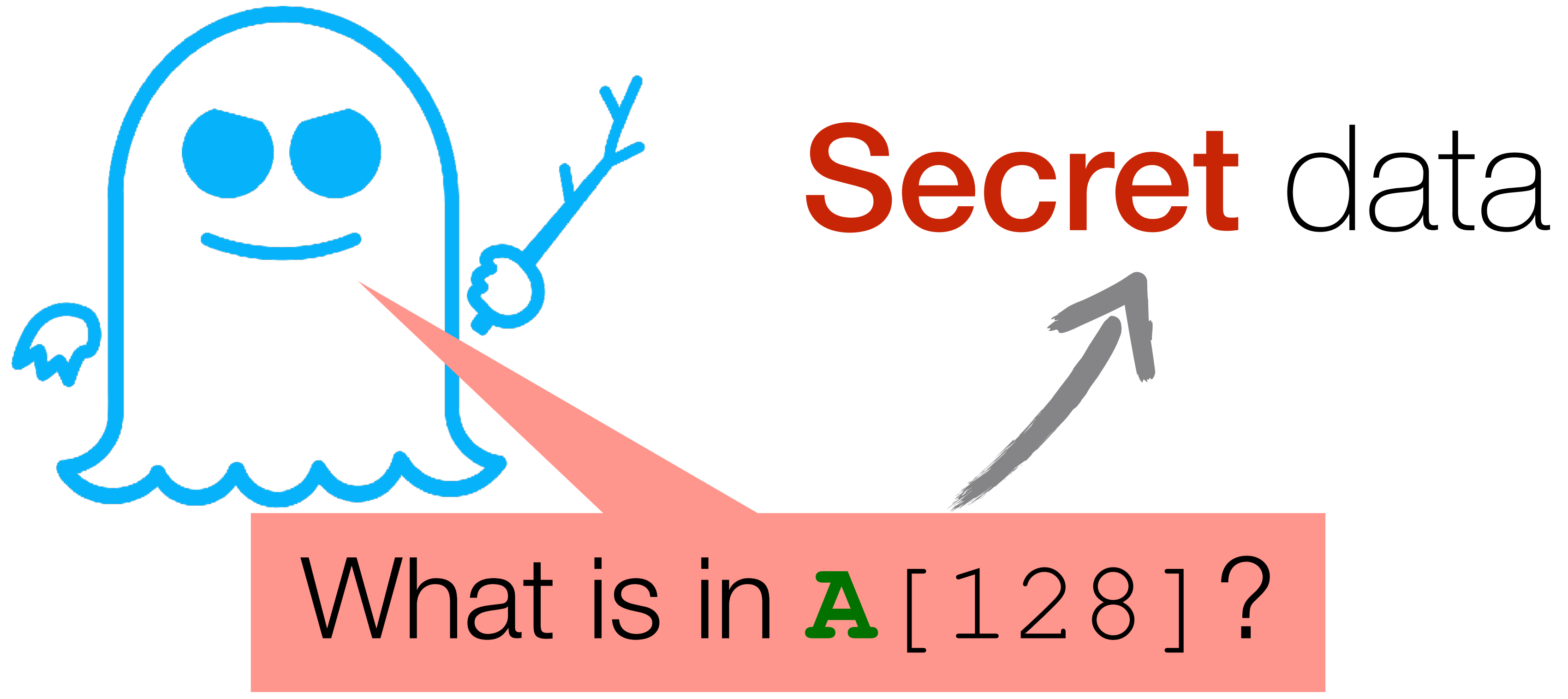
2) Prepare cache

Spectre v1

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







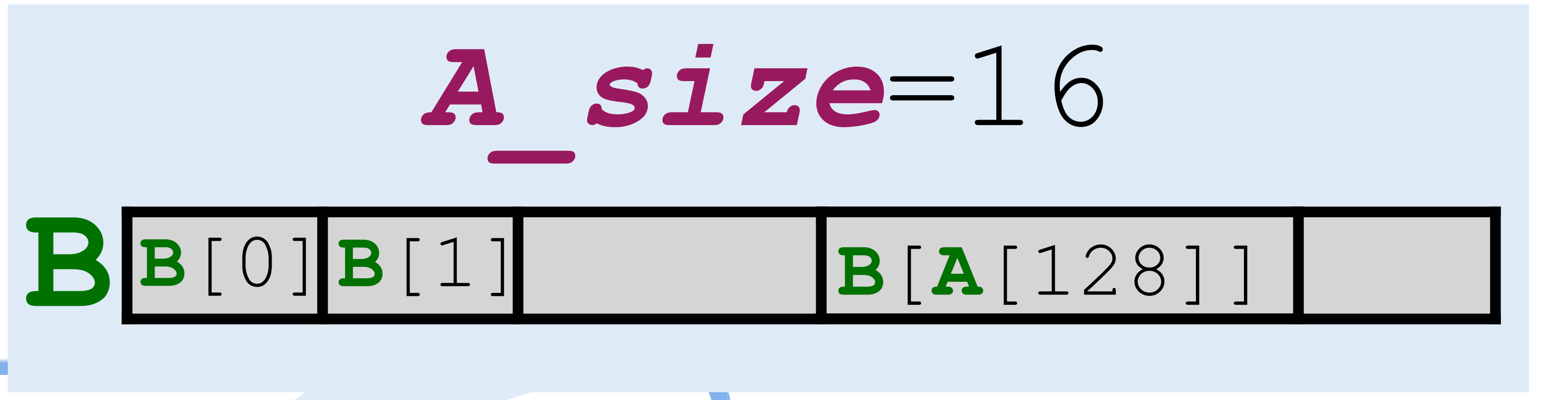
1) Train branch predictor

2) Prepare cache

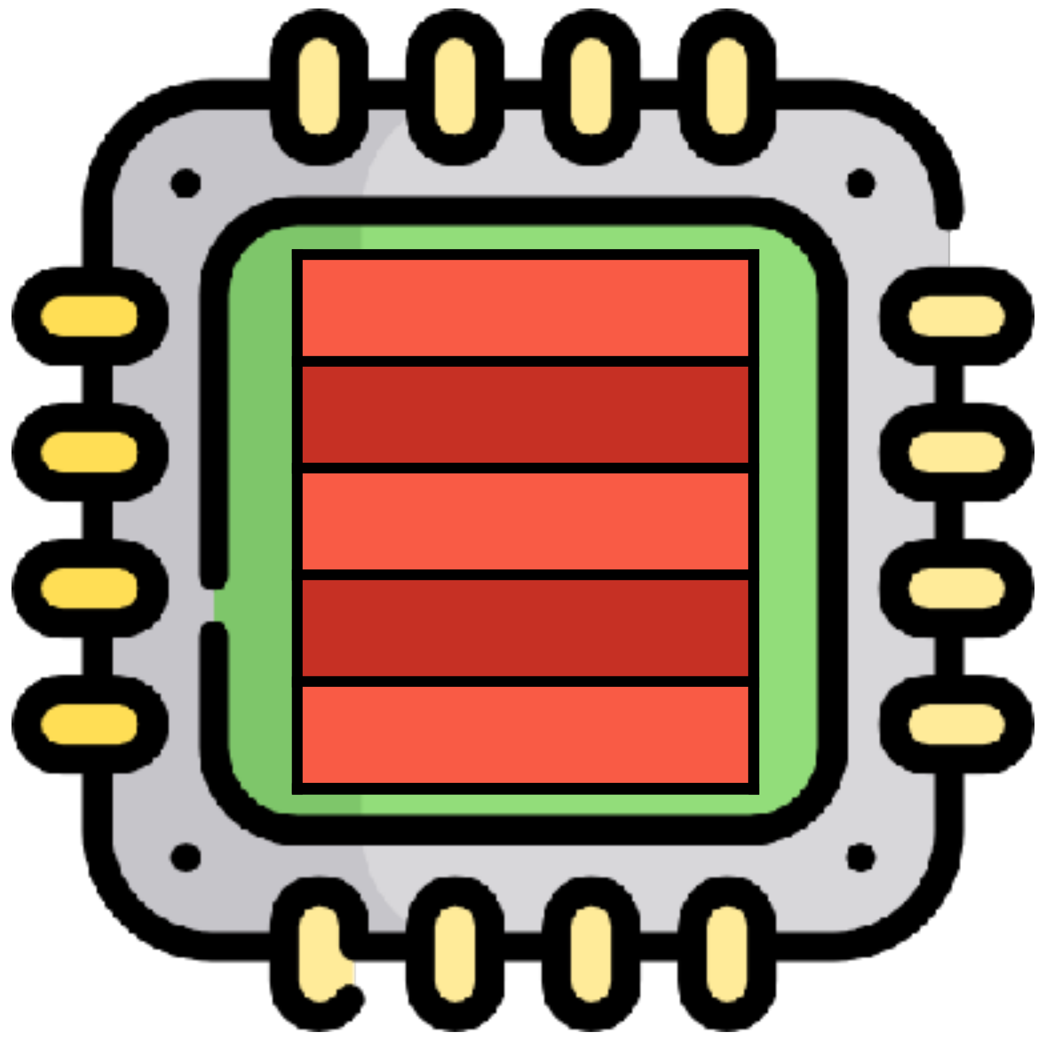
3) Run with x = 128

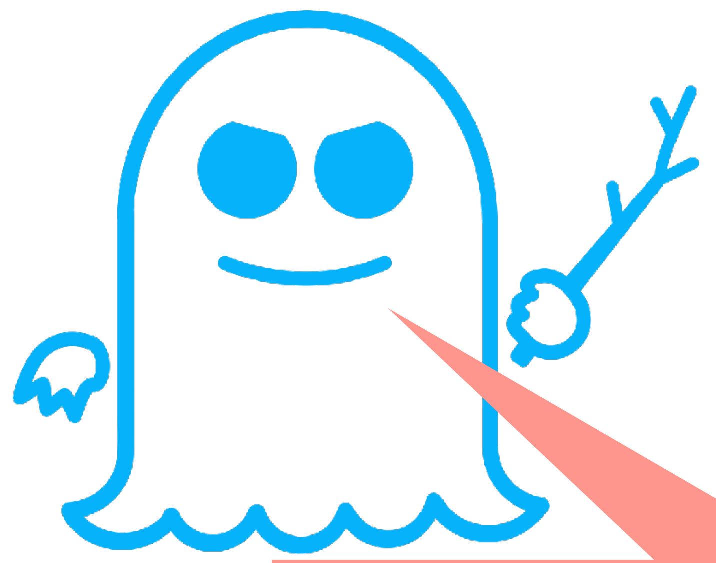
Spectre v1

```
A_size=16
```



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





Secret data

What is in `A[128]`?

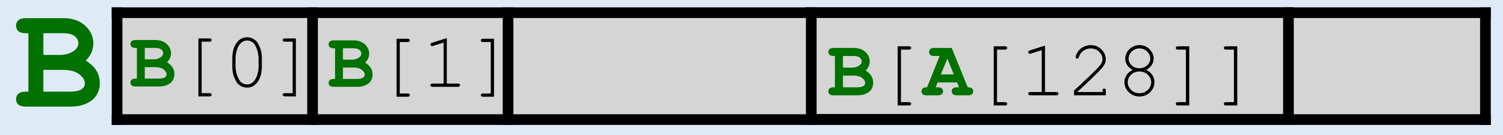
1) Train branch predictor

2) Prepare cache

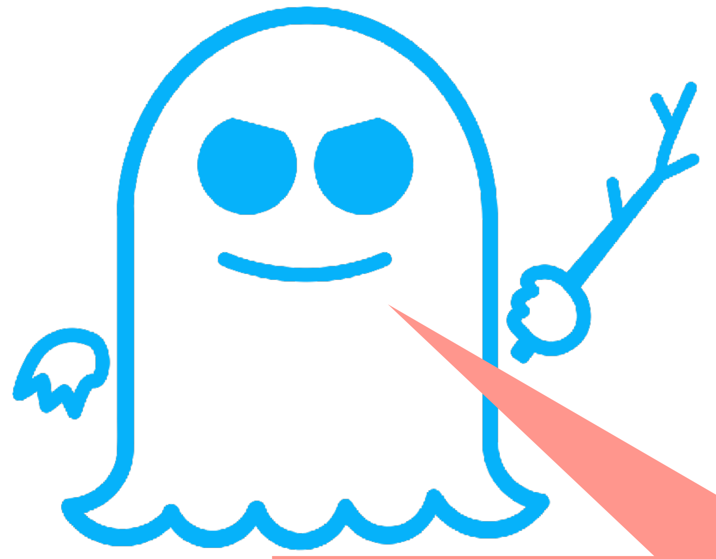
3) Run with `x = 128`

Spectre v1

A_size=16

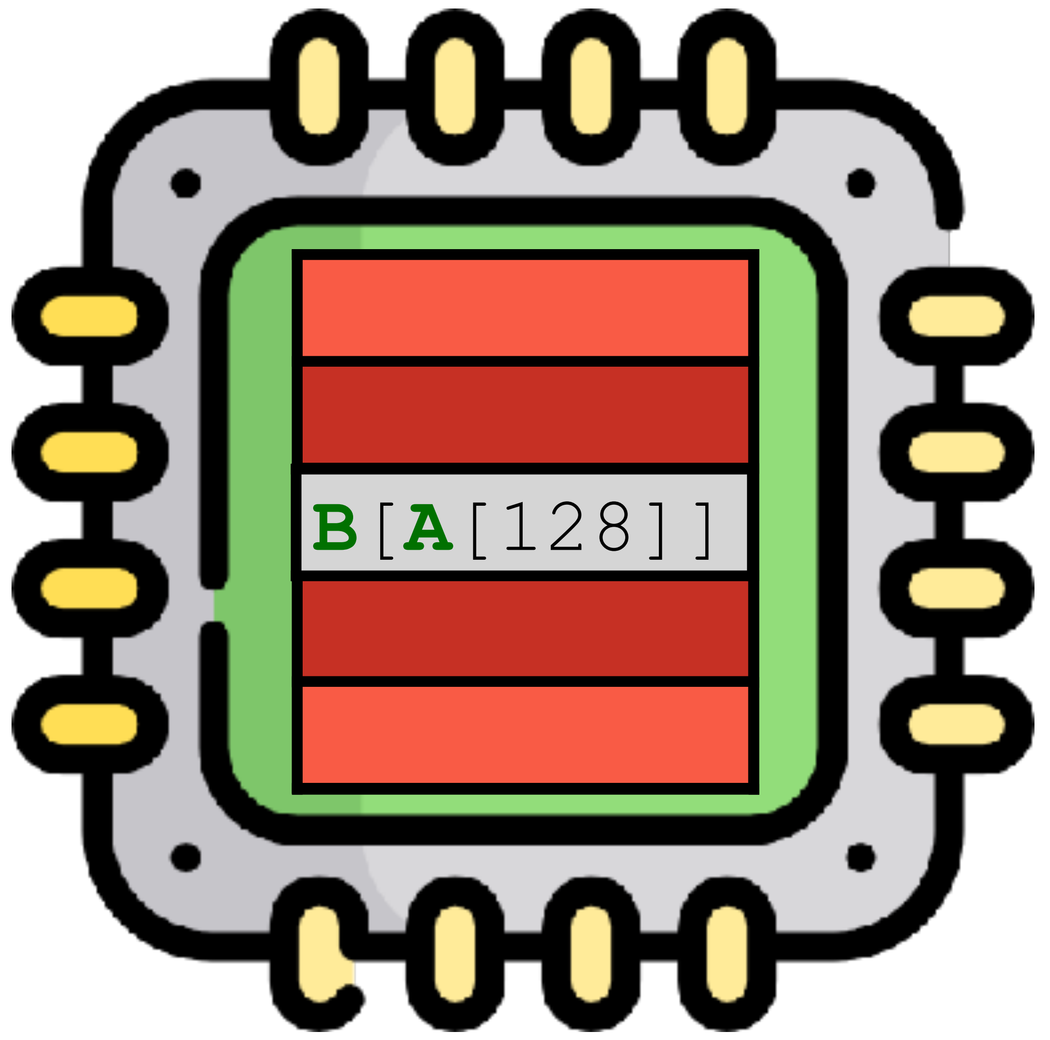


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



Secret data

What is in A[128]?



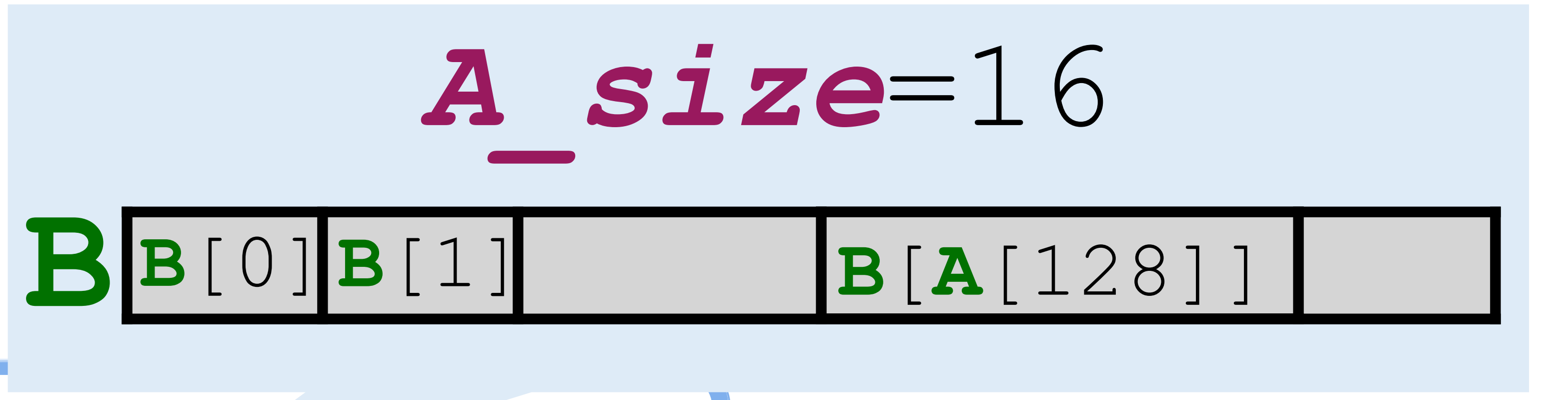
1) Train branch predictor

2) Prepare cache

3) Run with x = 128

Spectre v1

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



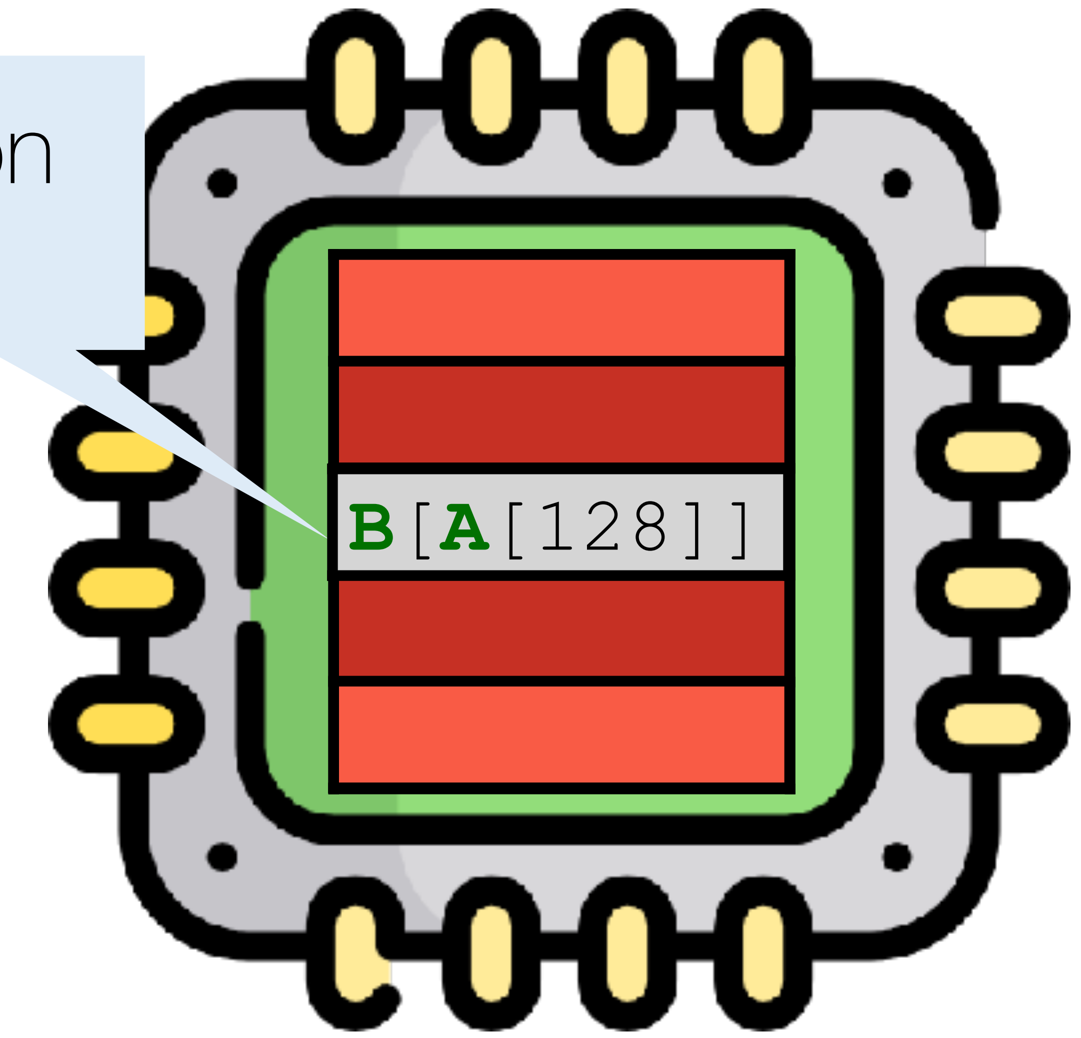
What is in A[128]?

1) Train branch predictor

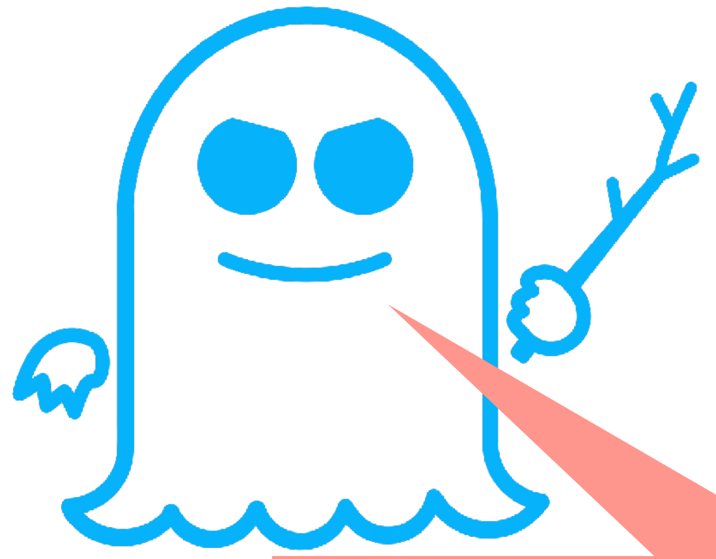
2) Prepare cache

3) Run with x = 128

Depends on A[128]



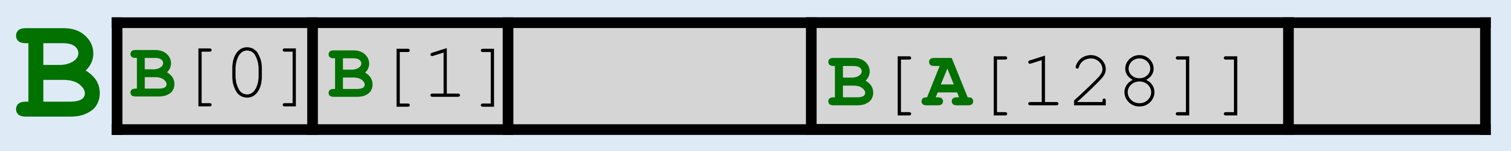
Spectre v1



Secret data

What is in A[128]?

A_size=16



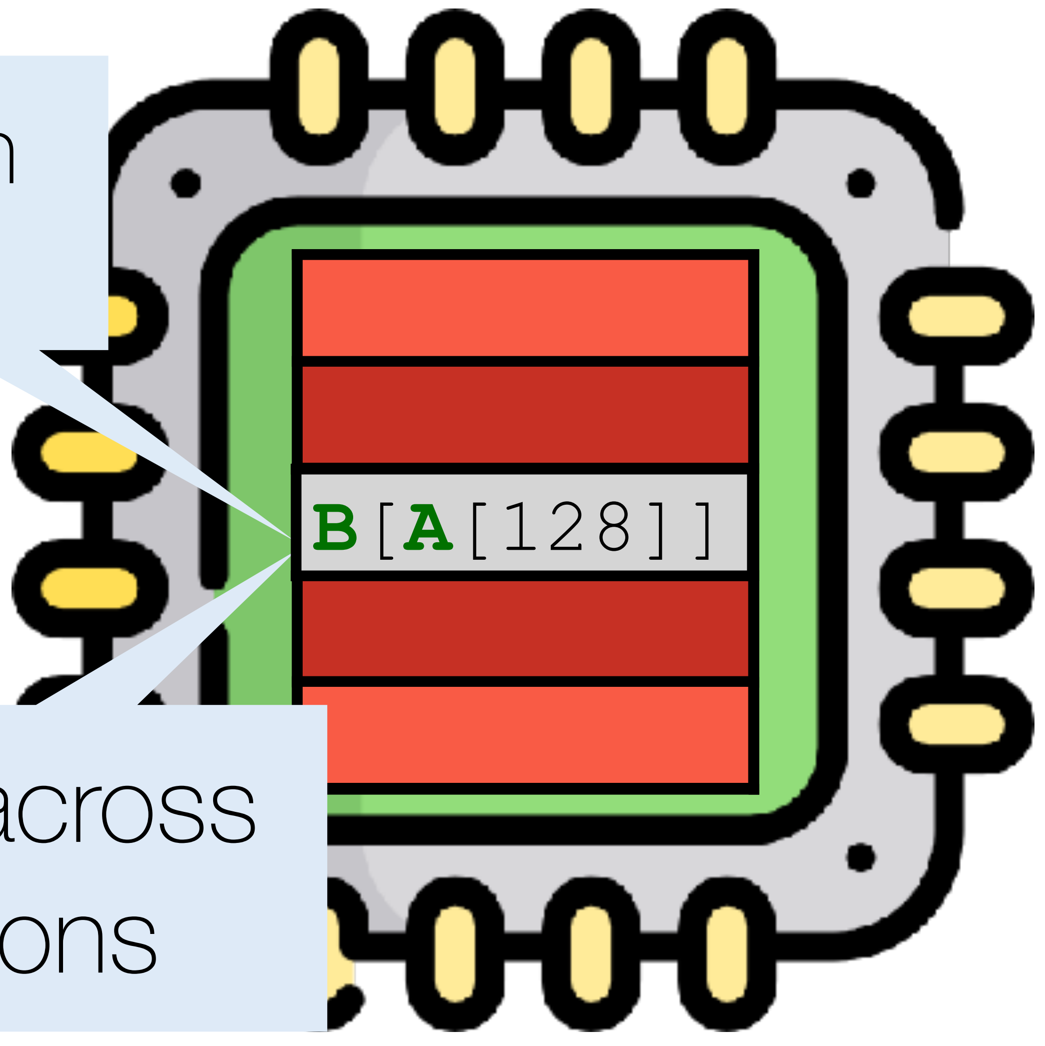
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) **Train branch predictor**

2) **Prepare cache**

3) **Run with x = 128**

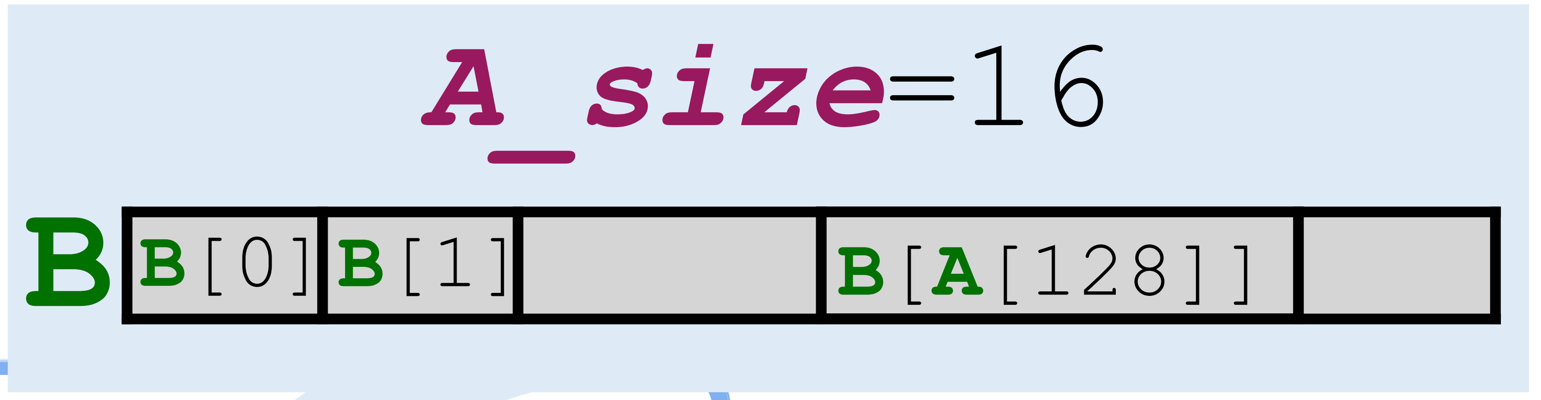
Depends on A[128]

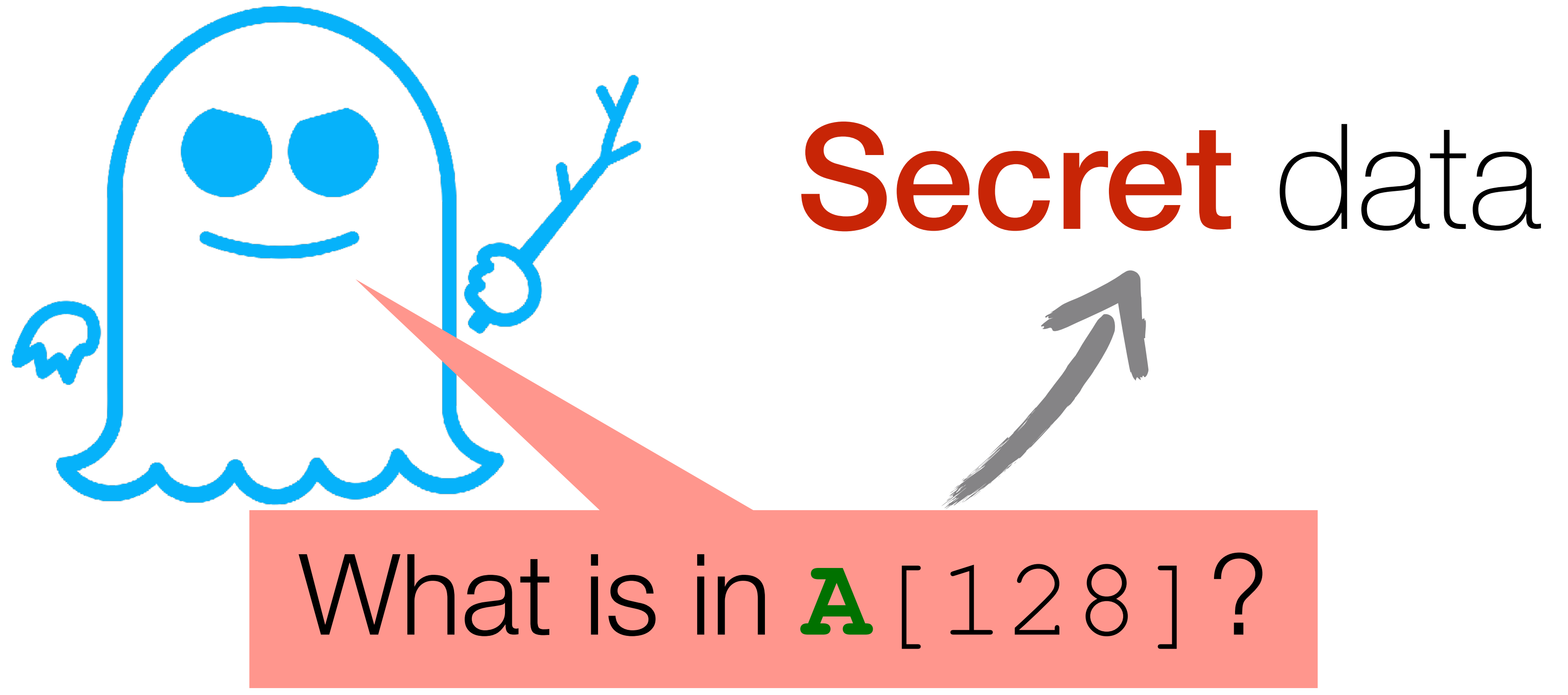


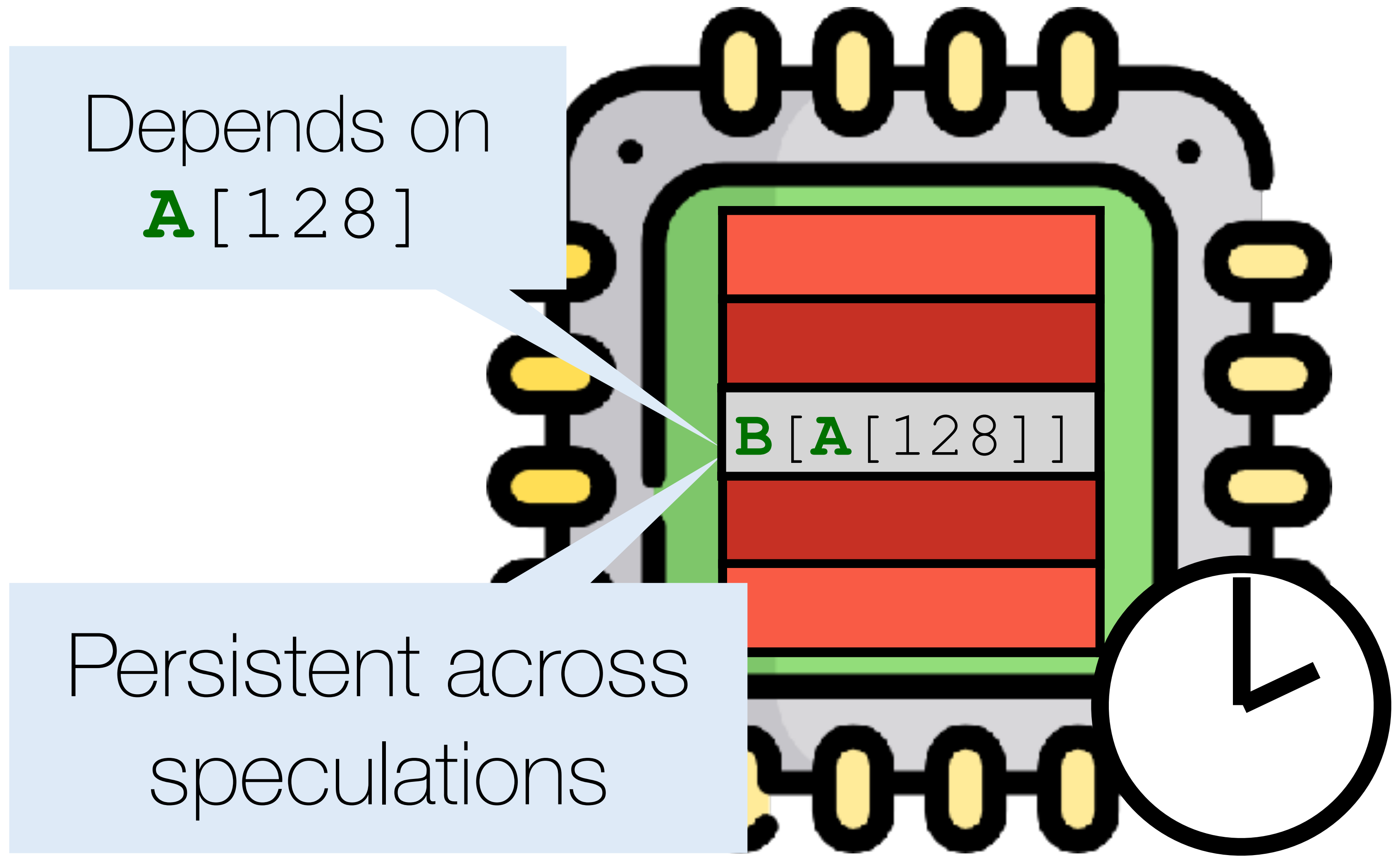
Persistent across speculations

Spectre v1

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







1) Train branch predictor

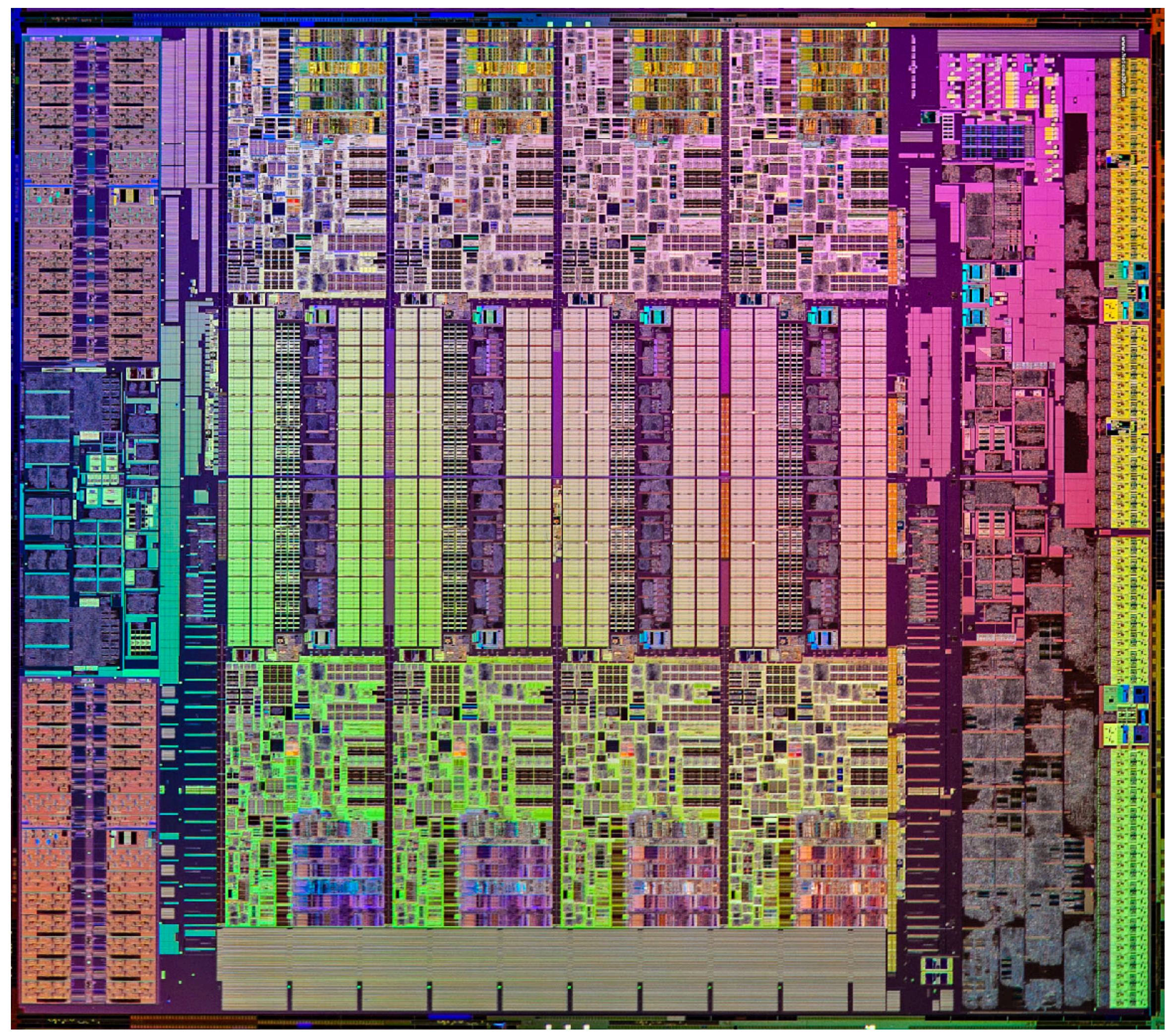
2) Prepare cache

3) Run with x = 128

4) Extract from cache

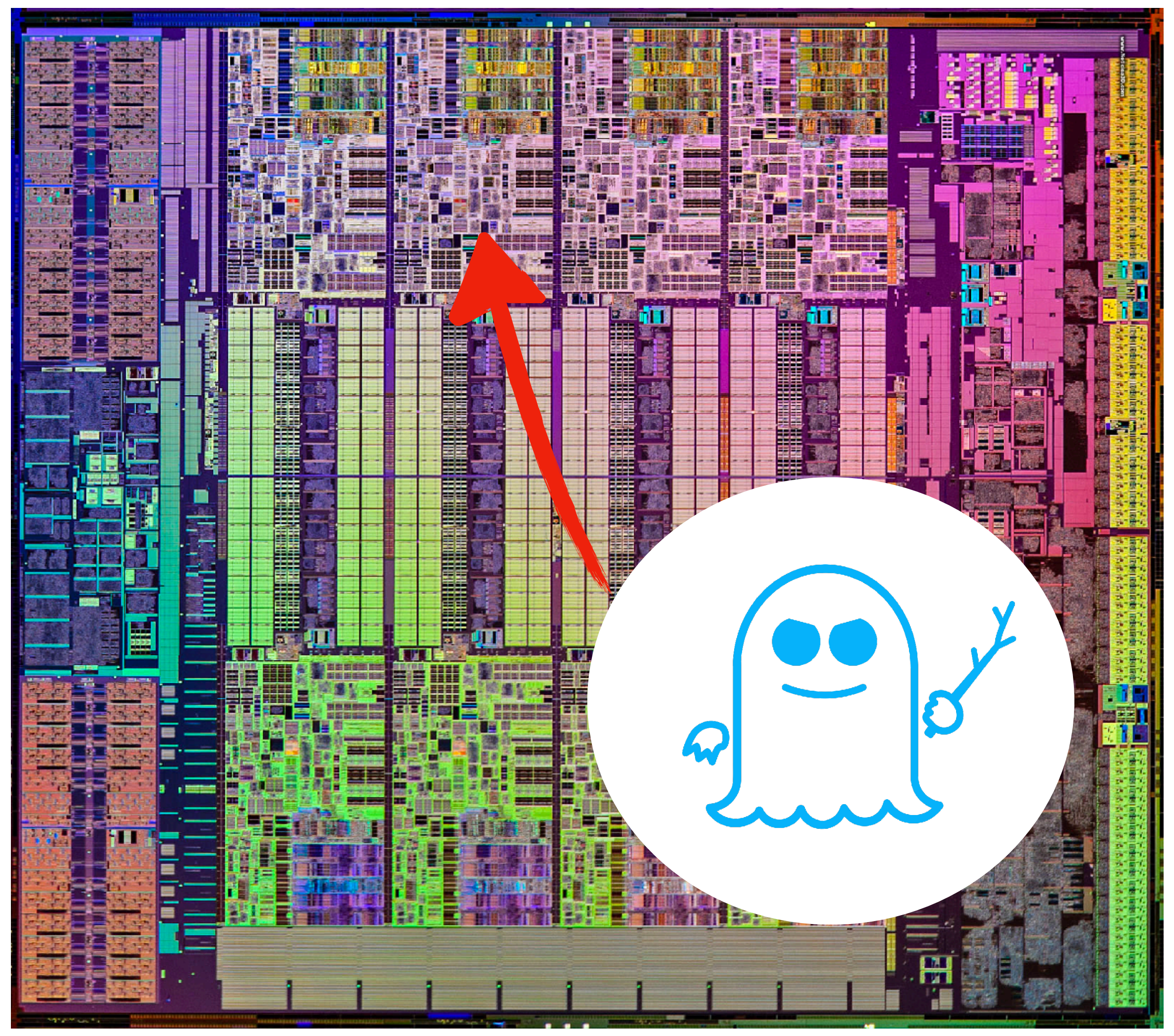
Outline

Speculative leaks at program level



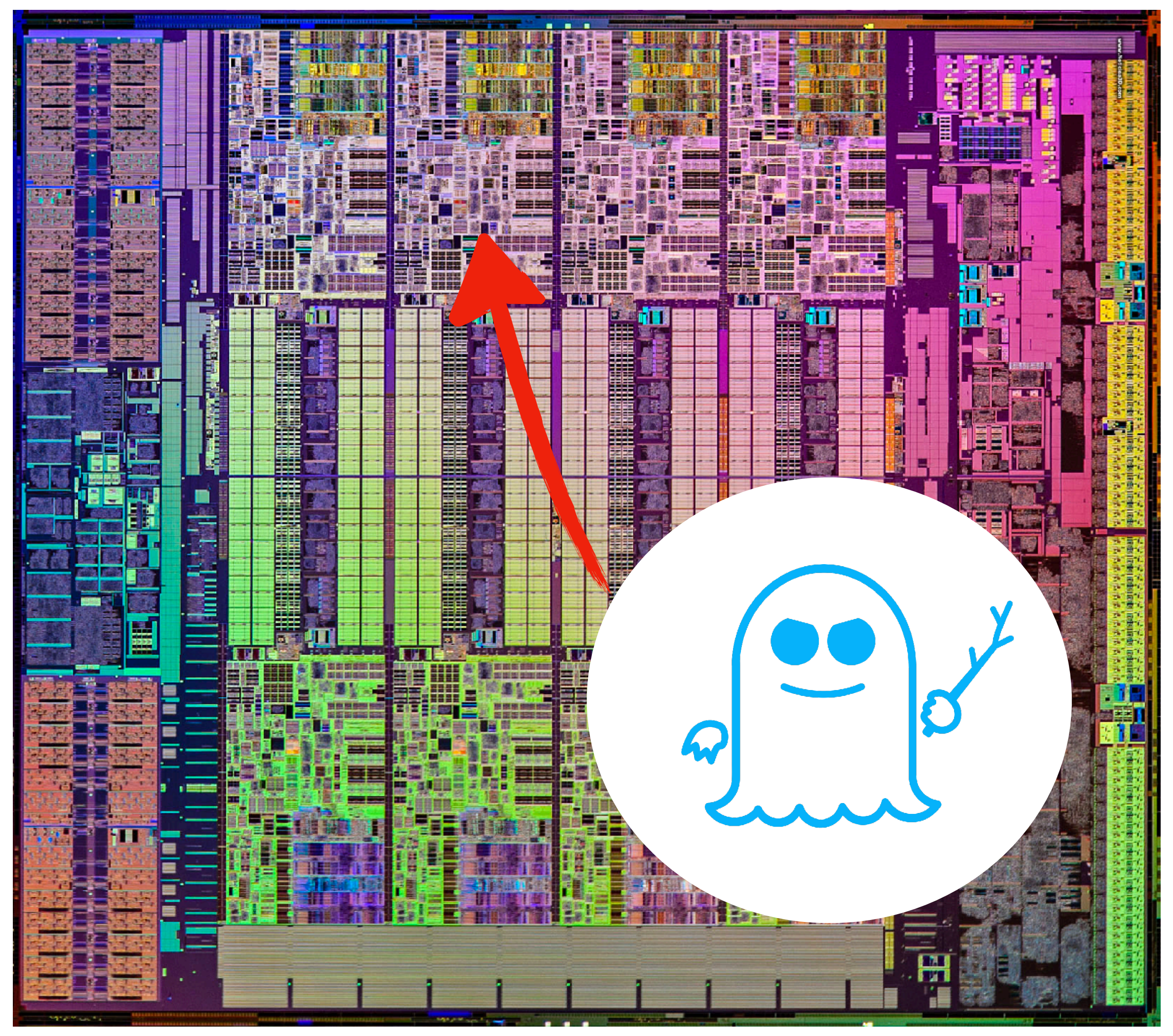
Guarnieri, Köpf, Morales, Reineke, Sánchez — Spectector: Principled detection for speculative leaks — IEEE S&P 2020 — https://arxiv.org/abs/1812.08639

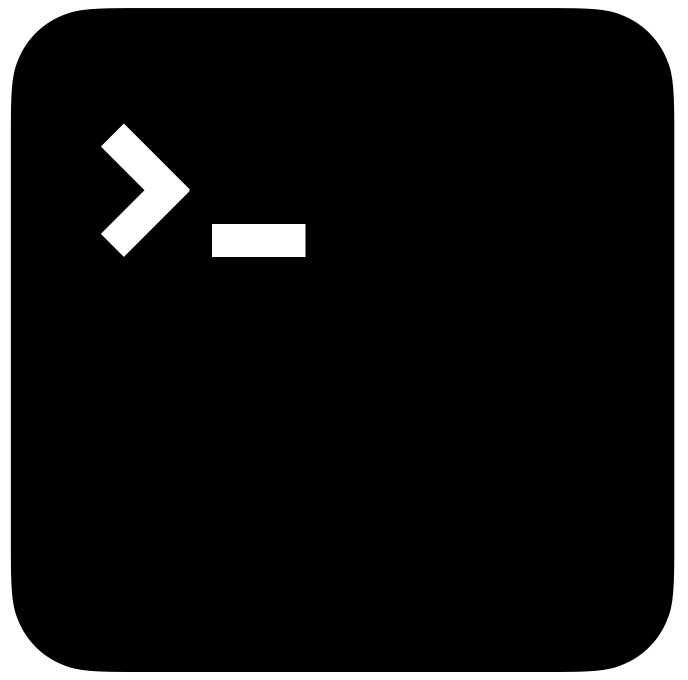
Speculative leaks at program level



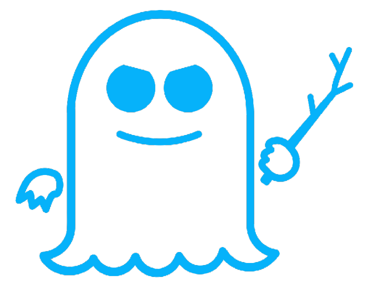
Guarnieri, Köpf, Morales, Reineke, Sánchez — Spectector: Principled detection for speculative leaks — IEEE S&P 2020 — https://arxiv.org/abs/1812.08639

Speculative leaks at program level



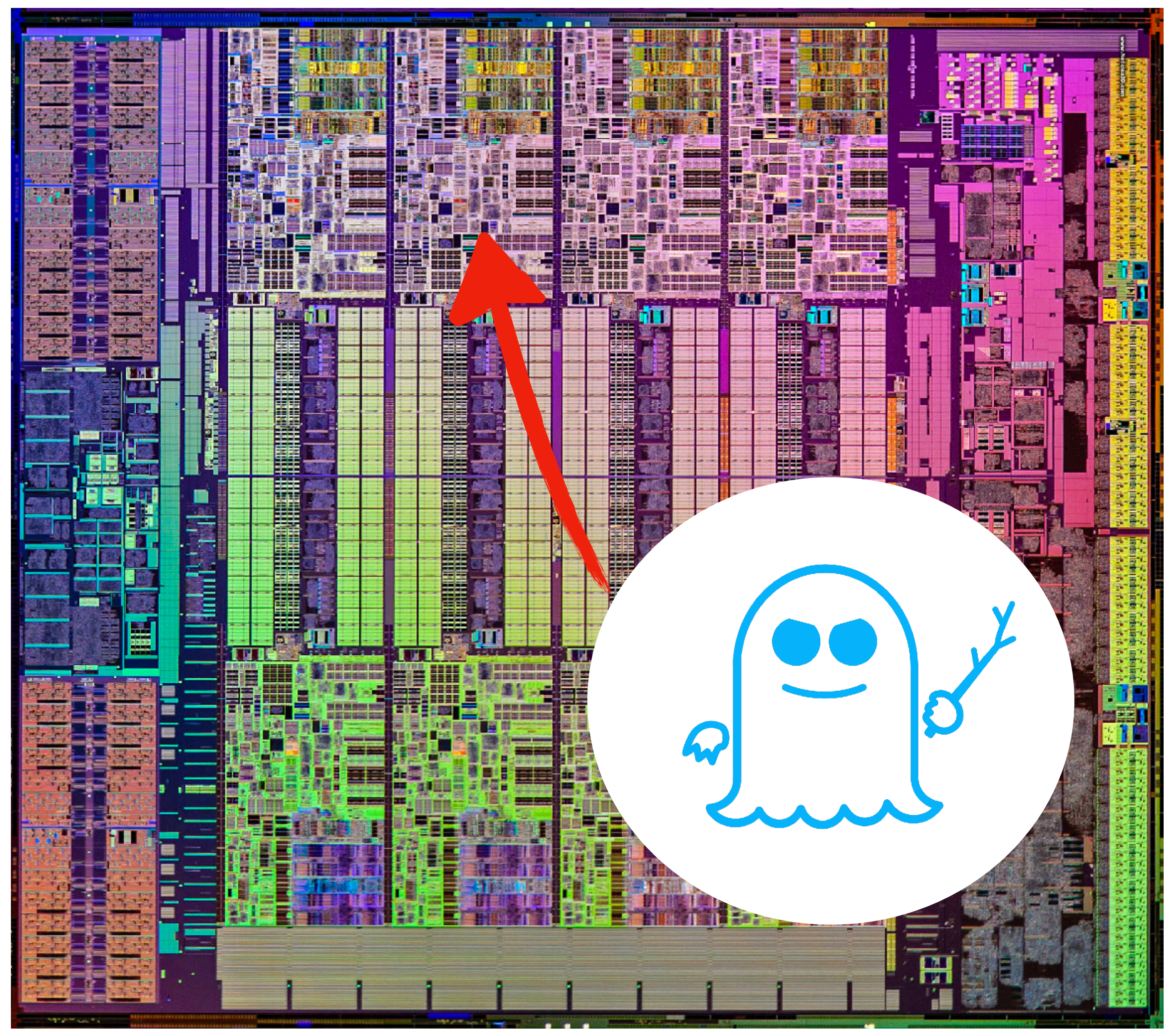


+



Speculative semantics

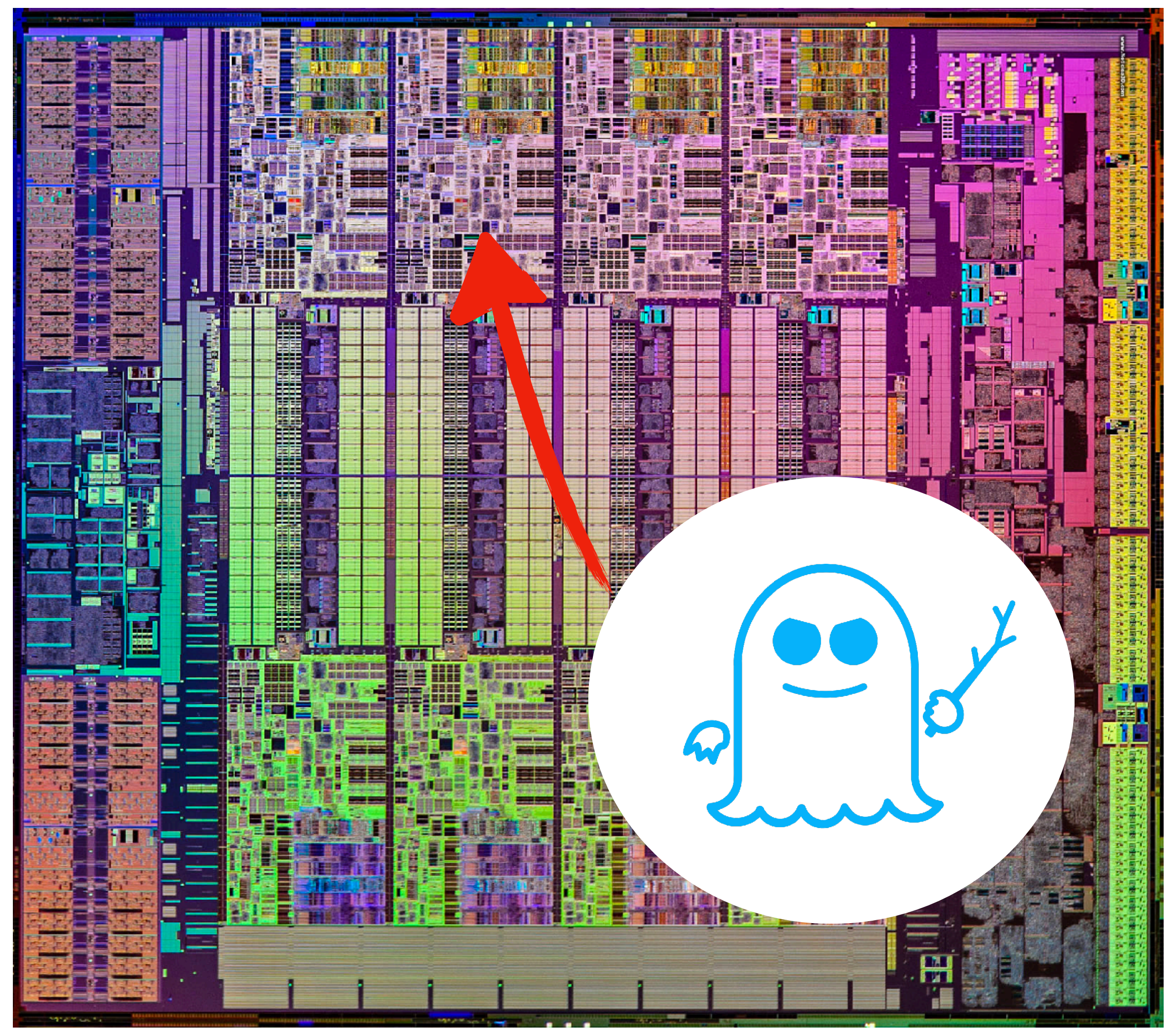
Speculative leaks at program level

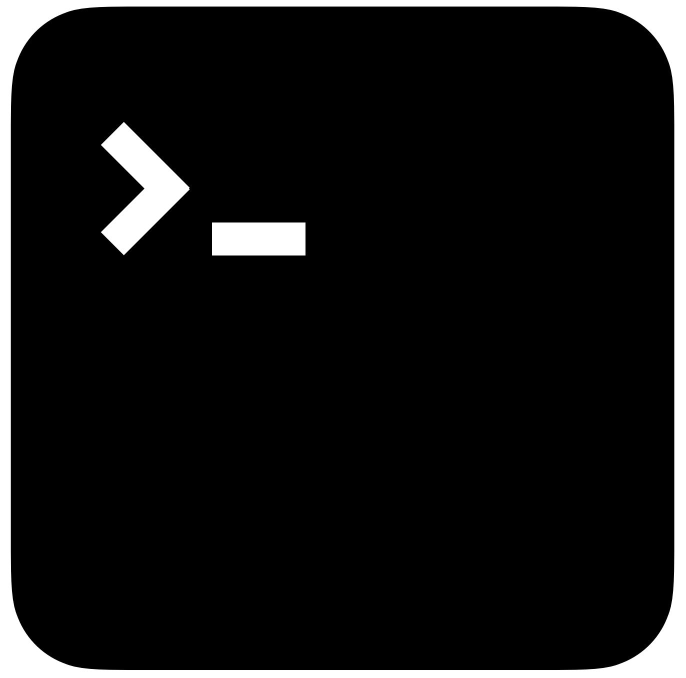


+ Speculative semantics

Execution mode + *Observer mode*

Speculative leaks at program level





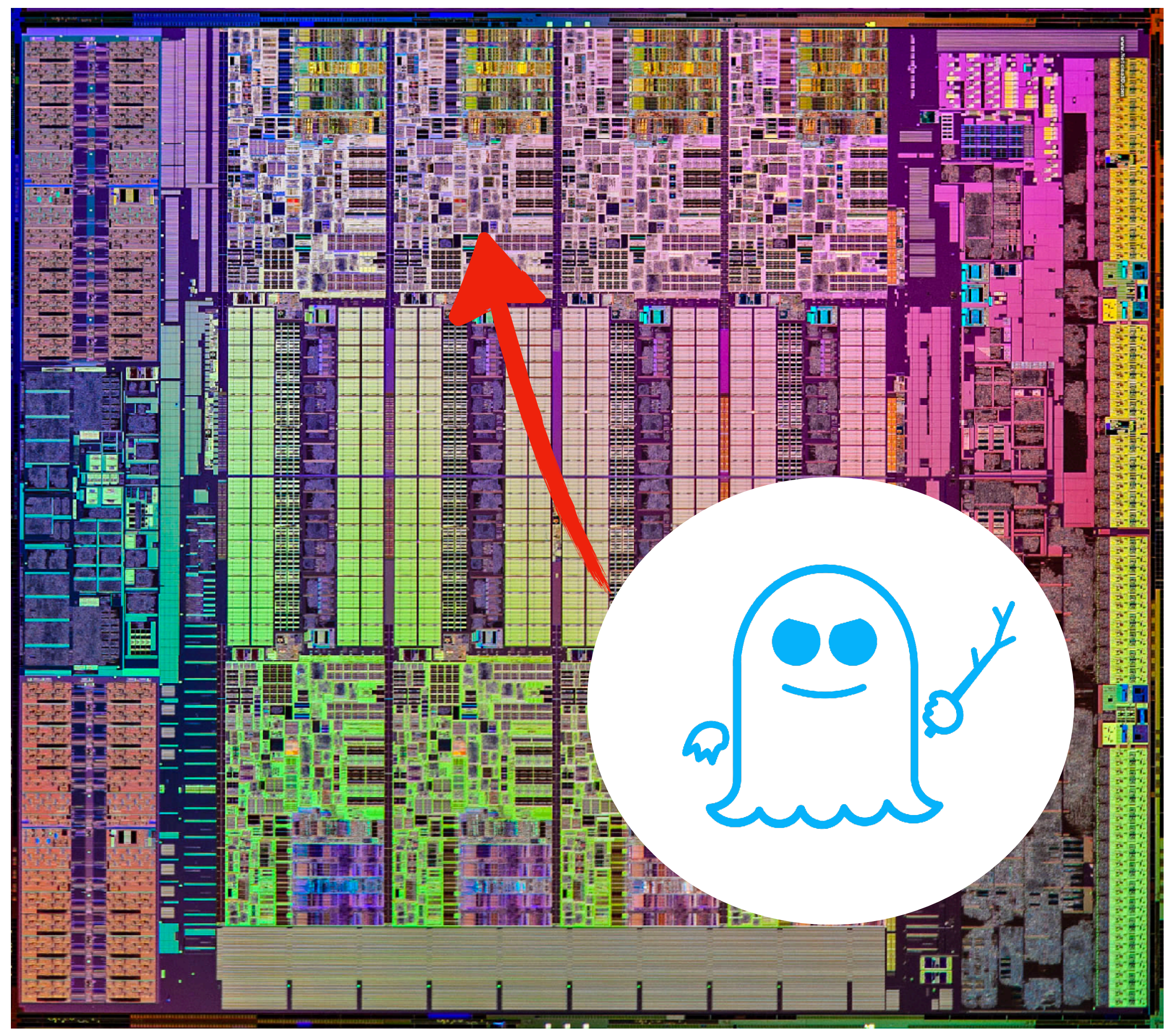
+

Speculative semantics

Execution mode **+** ***Observer mode***

Models how instructions are executed

Speculative leaks at program level



+ Speculative semantics

Execution mode + *Observer mode*

Models how instructions are executed

Capture attacker's observational power

Modeling speculation

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

Modeling speculation

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

Save ***program state*** before executing ***branch*** instructions

Modeling speculation

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

Save ***program state*** before executing ***branch*** instructions

Mispredict ***all*** branch instructions

Modeling speculation

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

Save ***program state*** before executing ***branch*** instructions

Mispredict ***all*** branch instructions

Fixed speculative window

Modeling speculation

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

Save ***program state*** before executing ***branch*** instructions

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback speculation

Modeling speculation

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

Save ***program state*** before executing ***branch*** instructions

Mispredict ***all*** branch instructions

Fixed speculative window

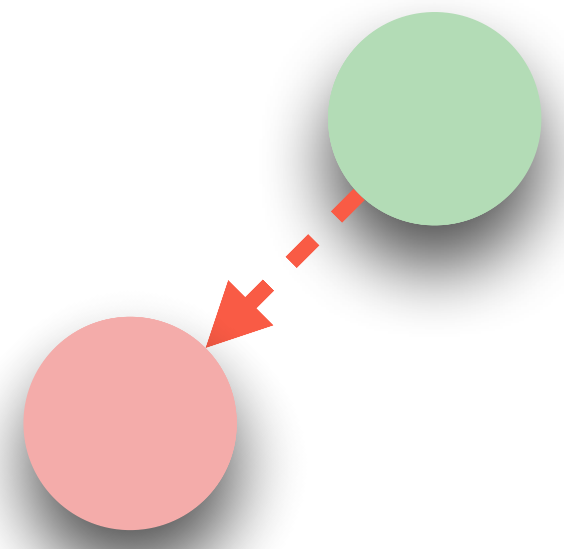
Rollback speculation

🟩 Non-speculative

🟥 Speculative

Modeling speculation

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



Save ***program state*** before executing ***branch*** instructions

Mispredict ***all*** branch instructions

Fixed speculative window

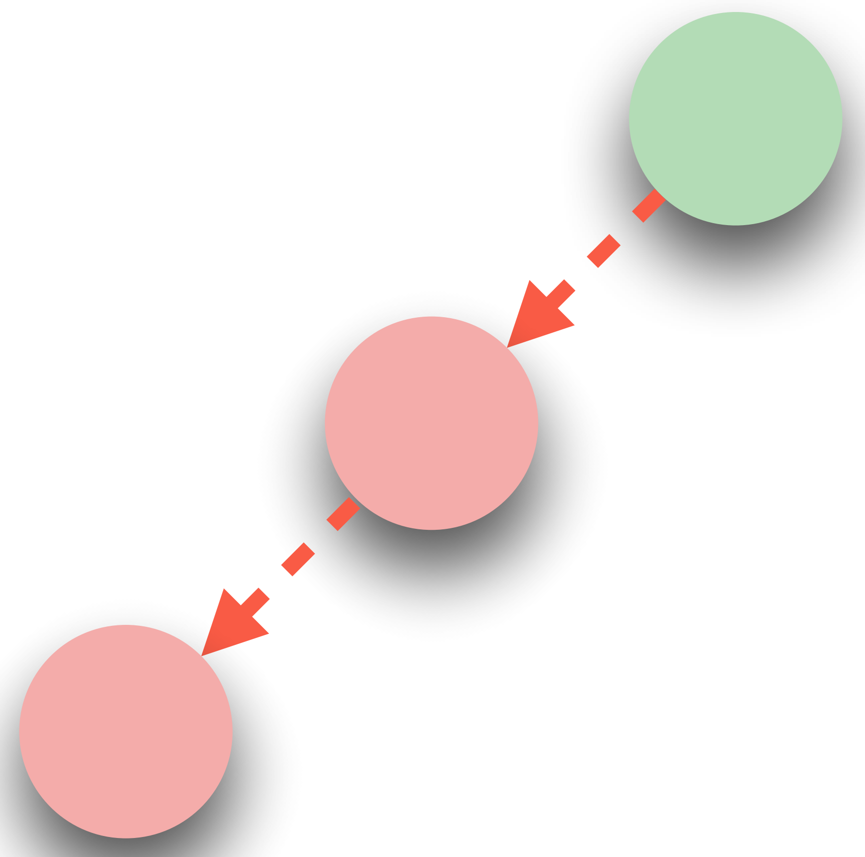
Rollback speculation

🟩 Non-speculative

🟥 Speculative

Modeling speculation

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



🟩 Non-speculative

🟥 Speculative

Save ***program state*** before executing ***branch*** instructions

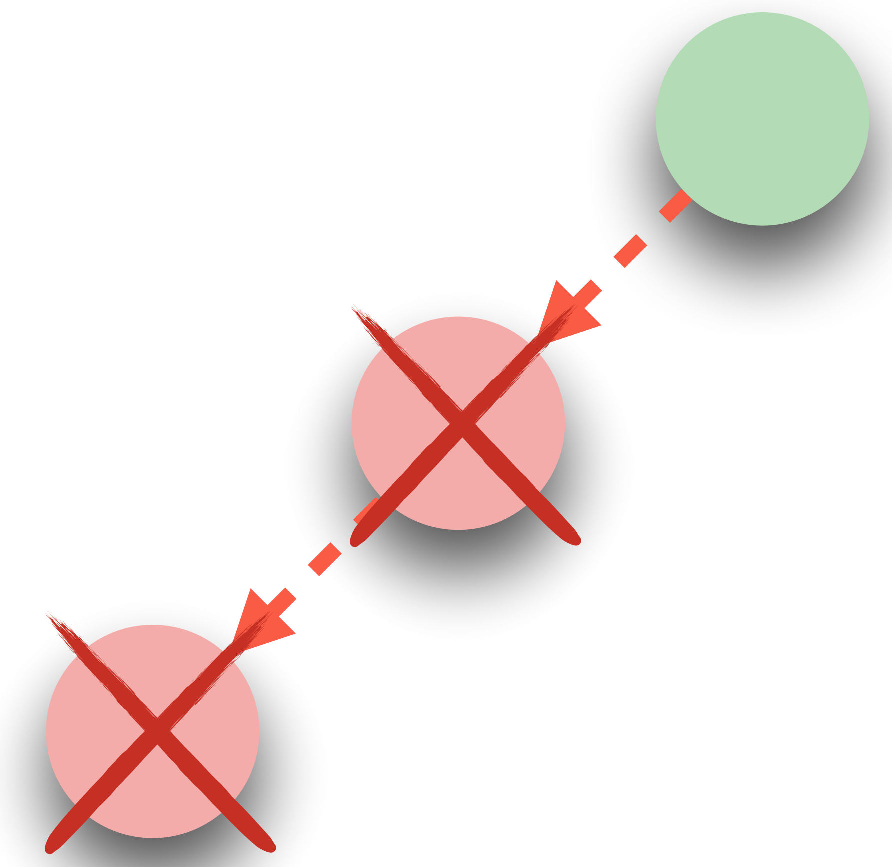
Mispredict ***all*** branch instructions

Fixed speculative window

Rollback speculation

Modeling speculation

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



🟩 Non-speculative

🟥 Speculative

Save ***program state*** before executing ***branch*** instructions

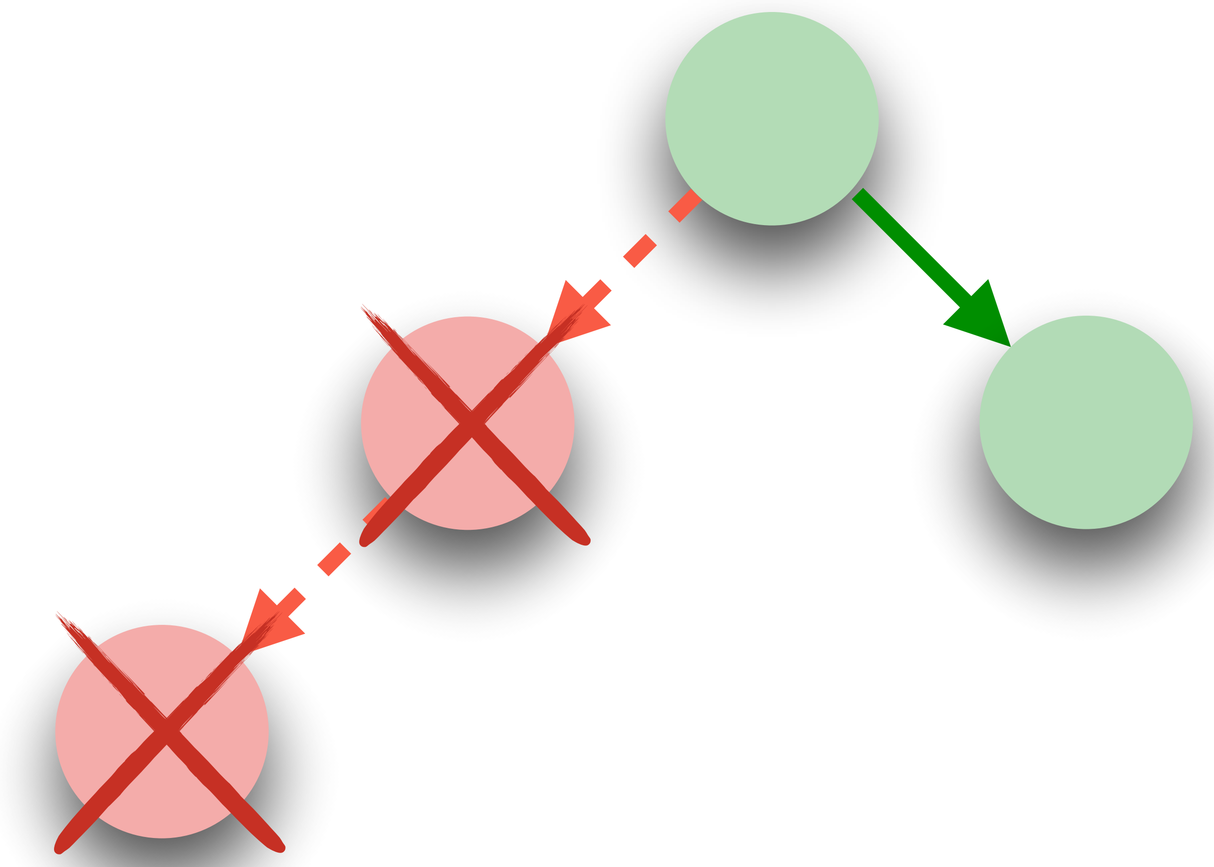
Mispredict ***all*** branch instructions

Fixed speculative window

Rollback speculation

Modeling speculation

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



🟩 Non-speculative

🟥 Speculative

Save ***program state*** before executing ***branch*** instructions

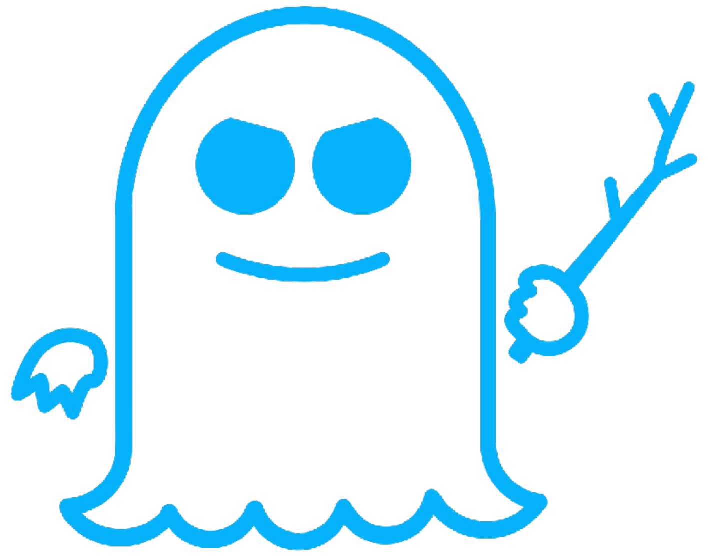
Mispredict ***all*** branch instructions

Fixed speculative window

Rollback speculation

Leakage into microarchitecture

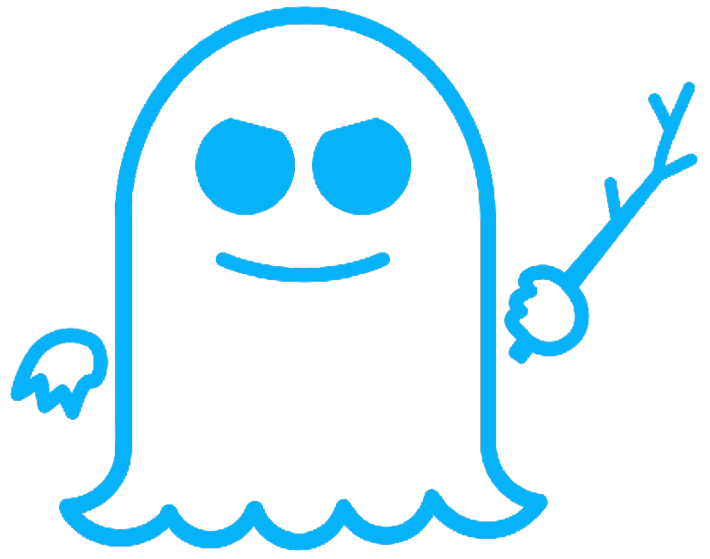
```
if (x < A_size)
  y = A[x]
  z = B[y]
end
```



Leakage into microarchitecture

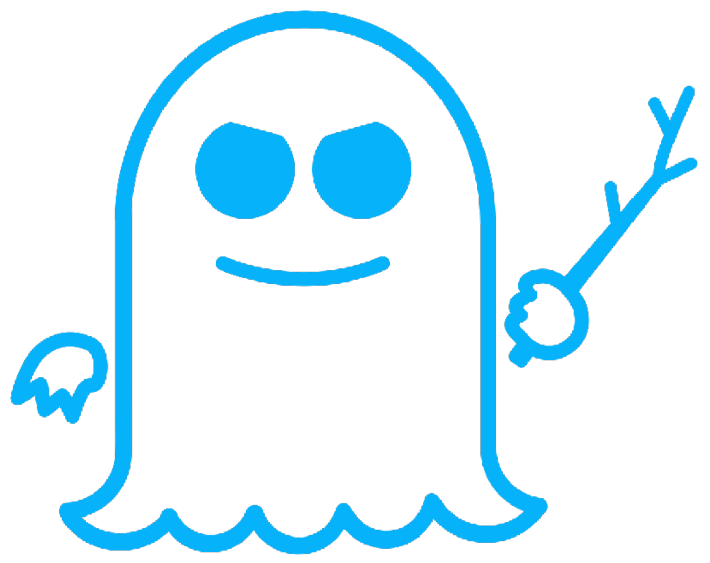
```
if (x < A_size)
   y = A[x]
   z = B[y]
end
```

Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution



Leakage into microarchitecture

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



Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

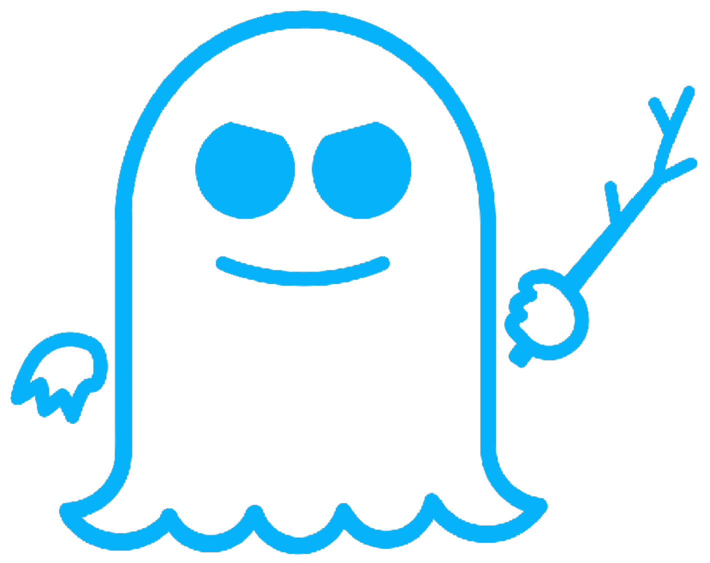
Inspired by "constant-time" requirements

Leakage into microarchitecture

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

Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" requirements

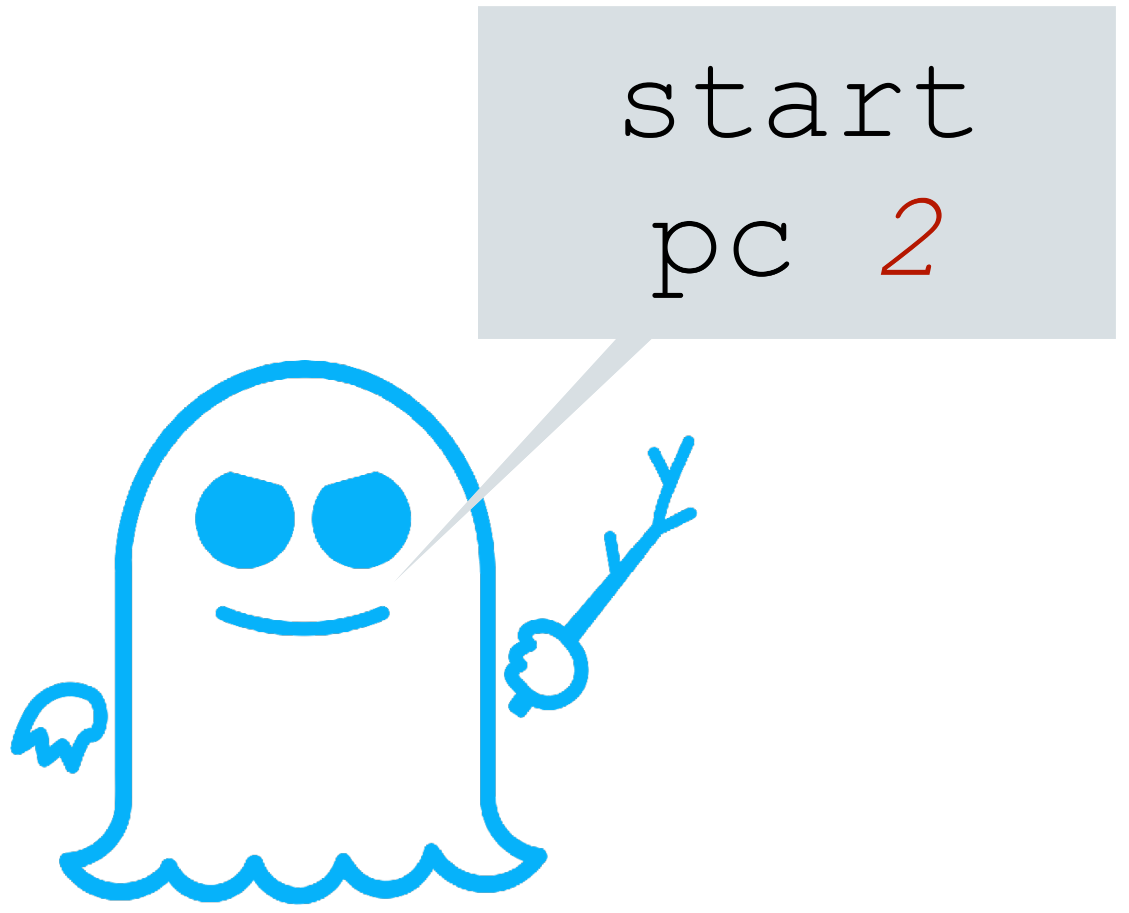


🟩 Non-speculative

🟥 Speculative

Leakage into microarchitecture

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



Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

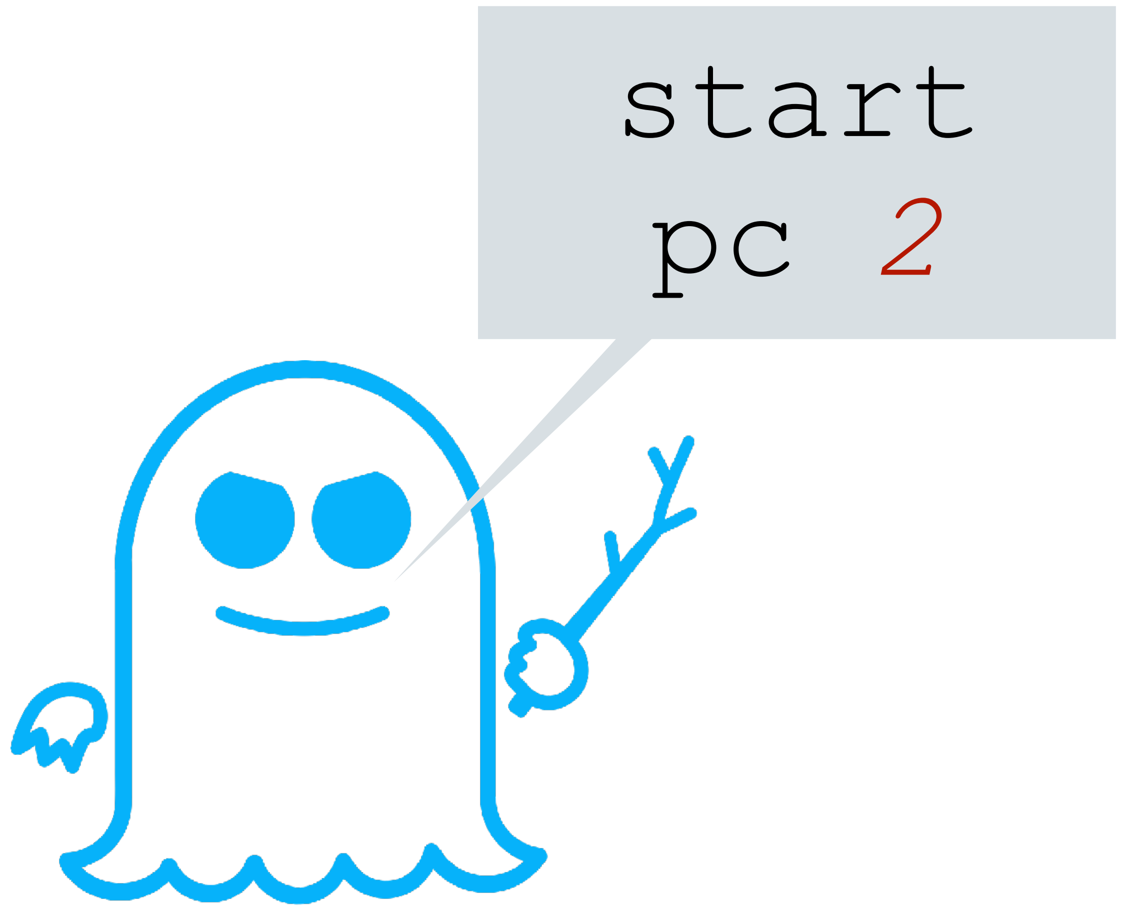
Inspired by "constant-time" requirements

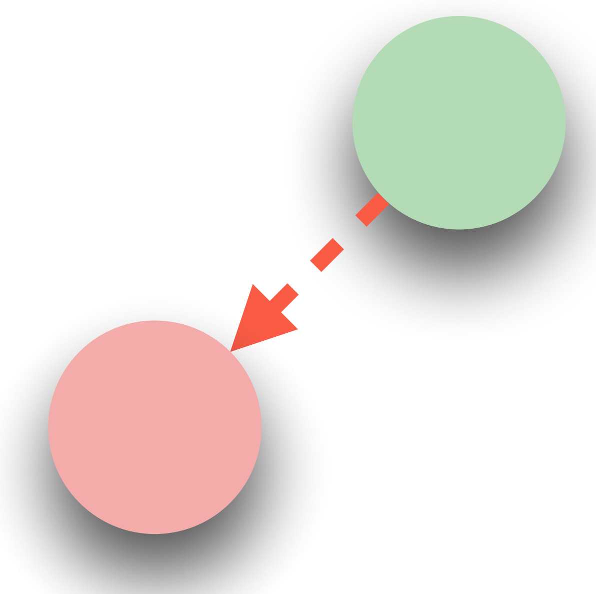
🟩 Non-speculative

🟥 Speculative

Leakage into microarchitecture

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





Attacker observes:
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

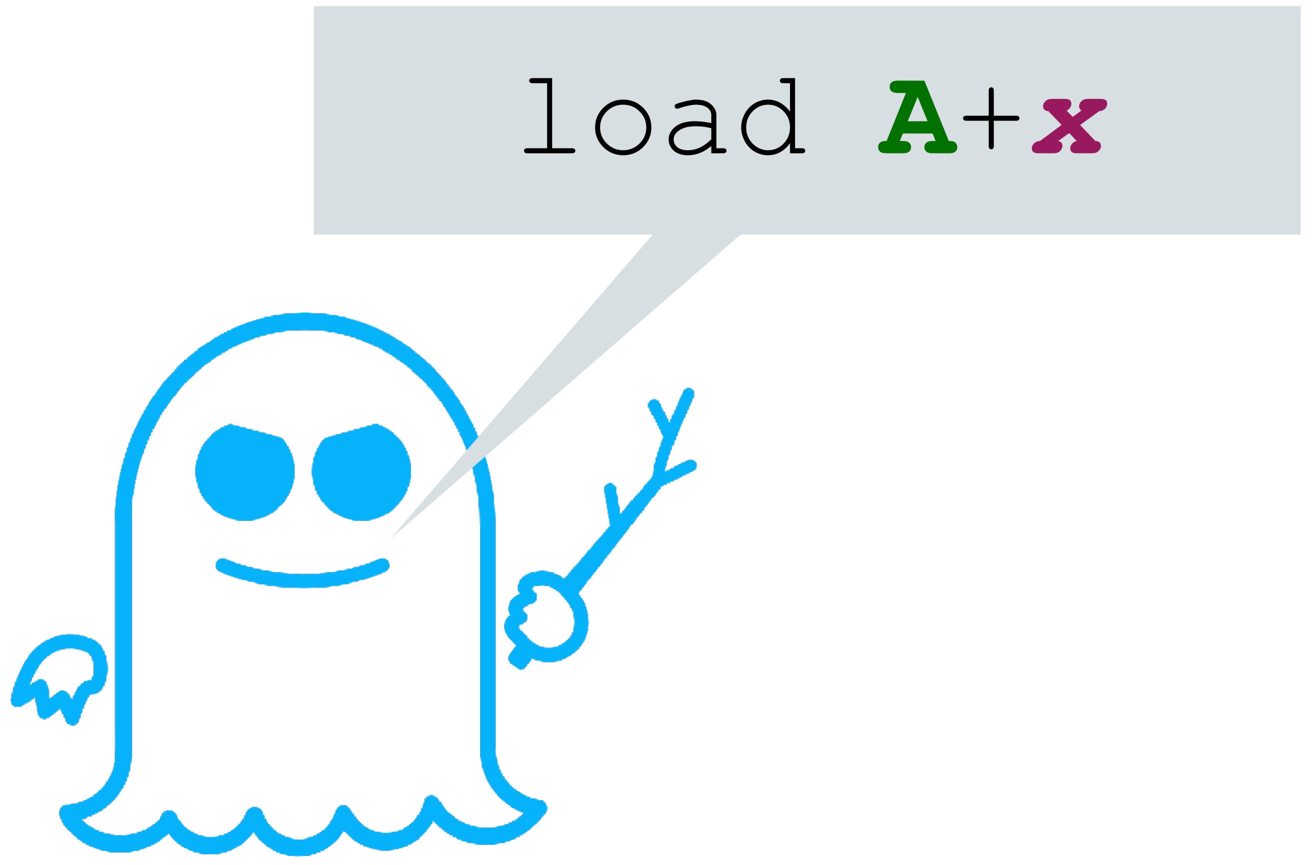
Inspired by "constant-time" requirements

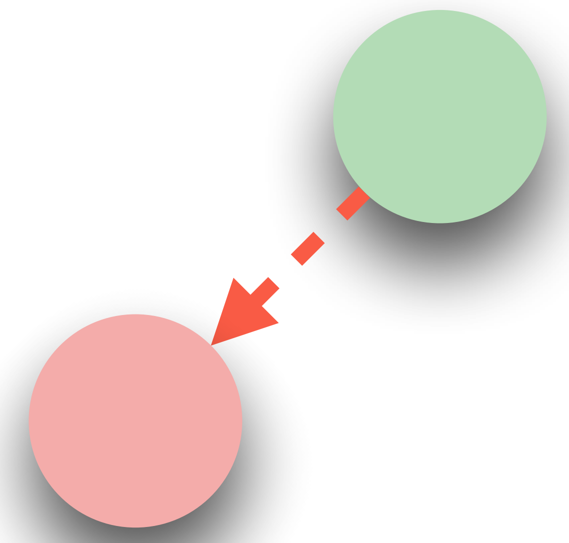
🟩 Non-speculative

🟥 Speculative

Leakage into microarchitecture

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





Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

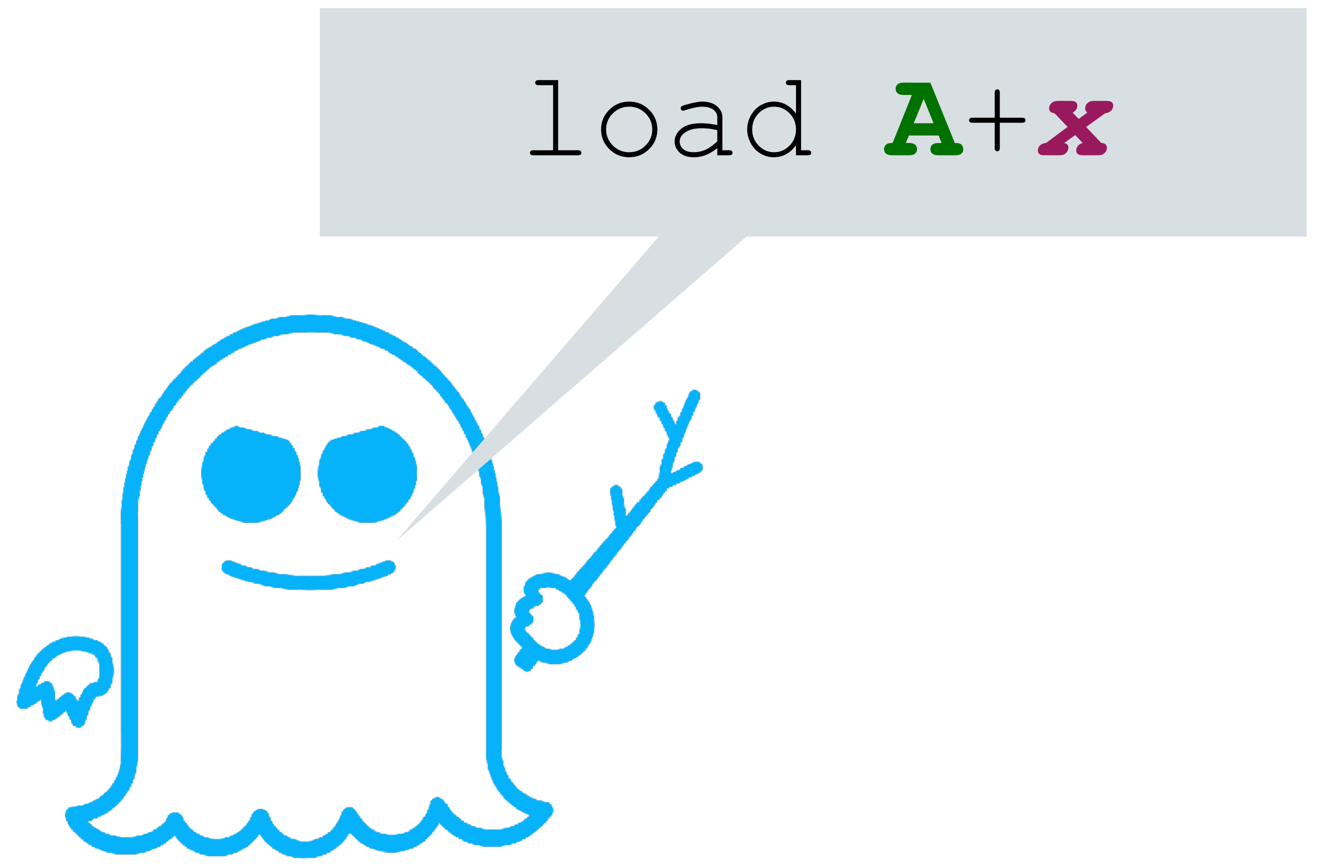
Inspired by “constant-time” requirements

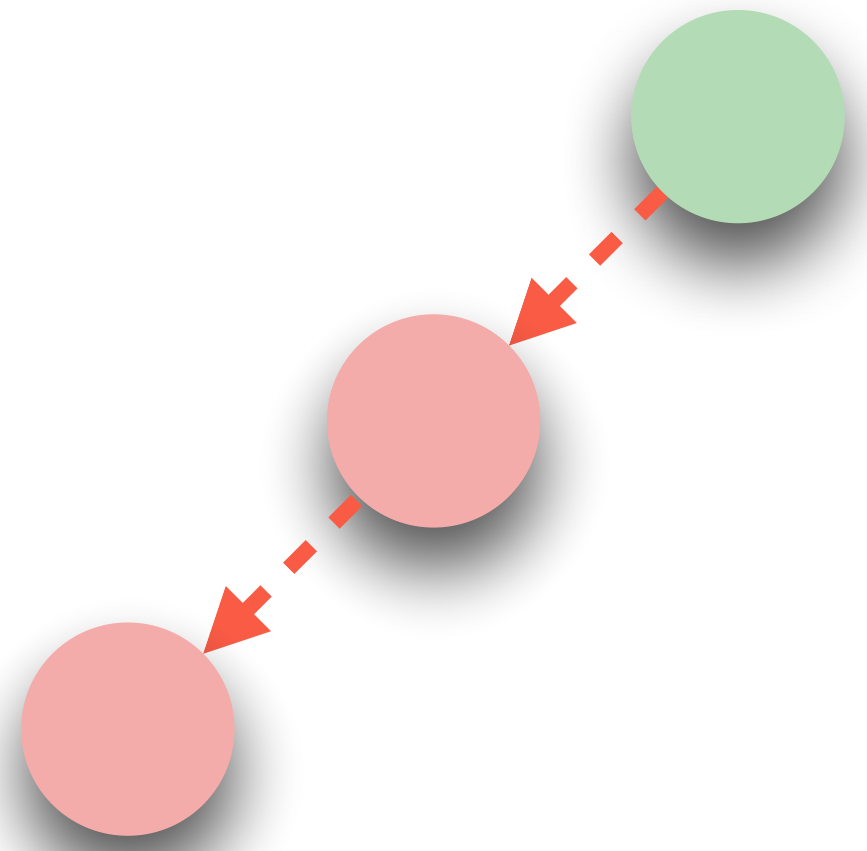
🟩 Non-speculative

🟥 Speculative

Leakage into microarchitecture

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





🟩 Non-speculative

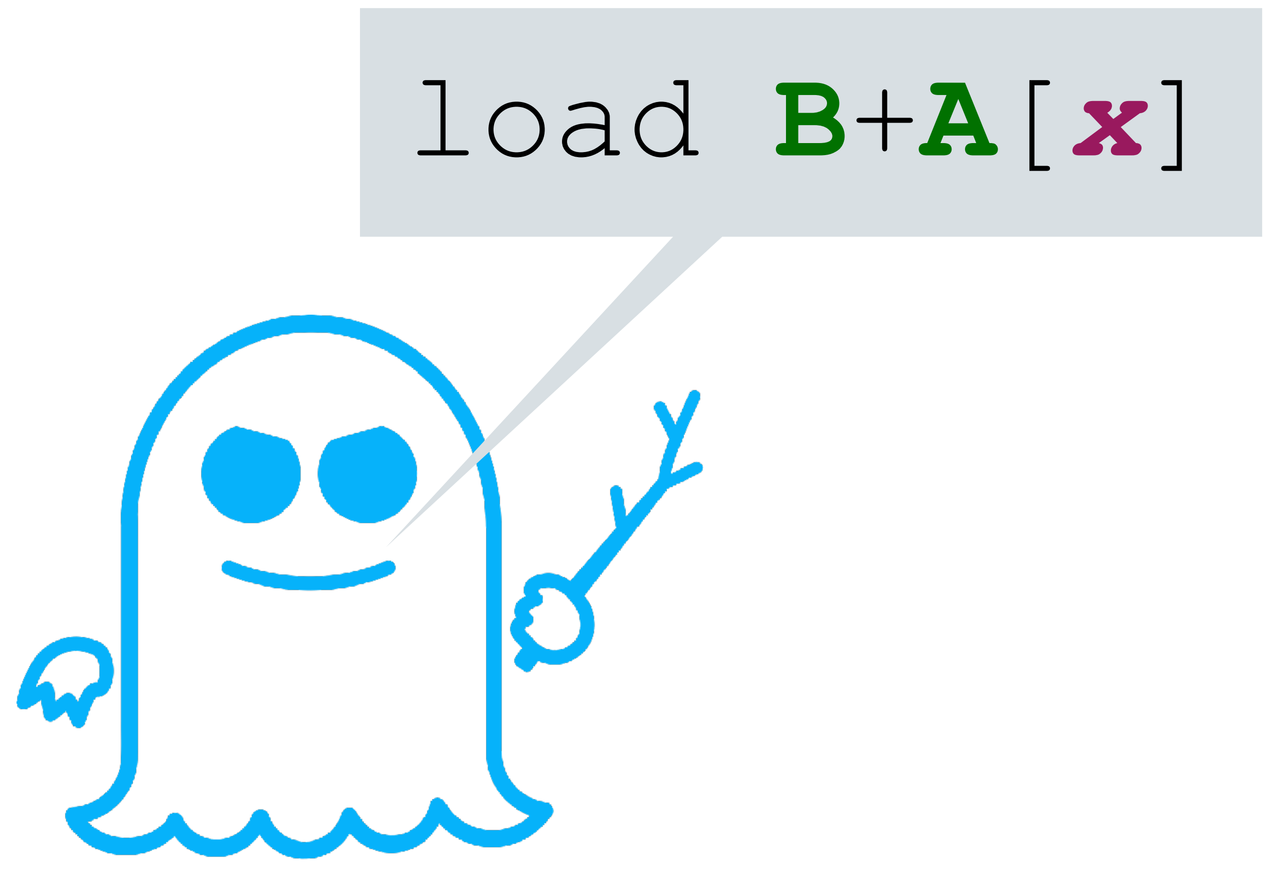
🟥 Speculative

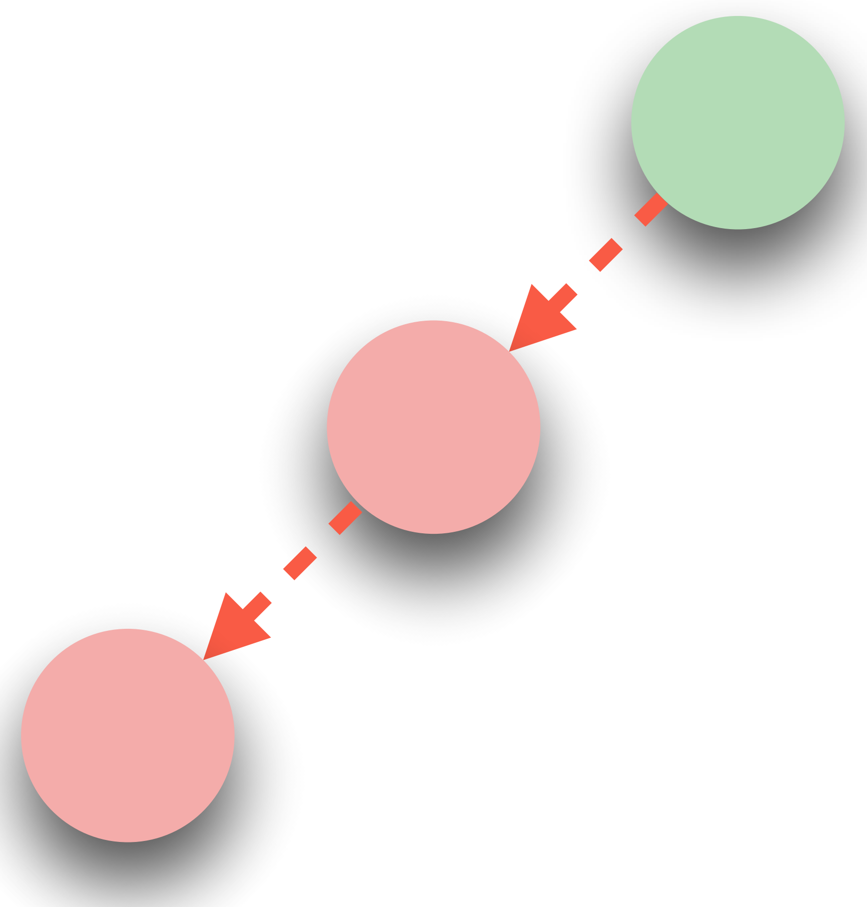
Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" requirements

Leakage into microarchitecture

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





Attacker observes:
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

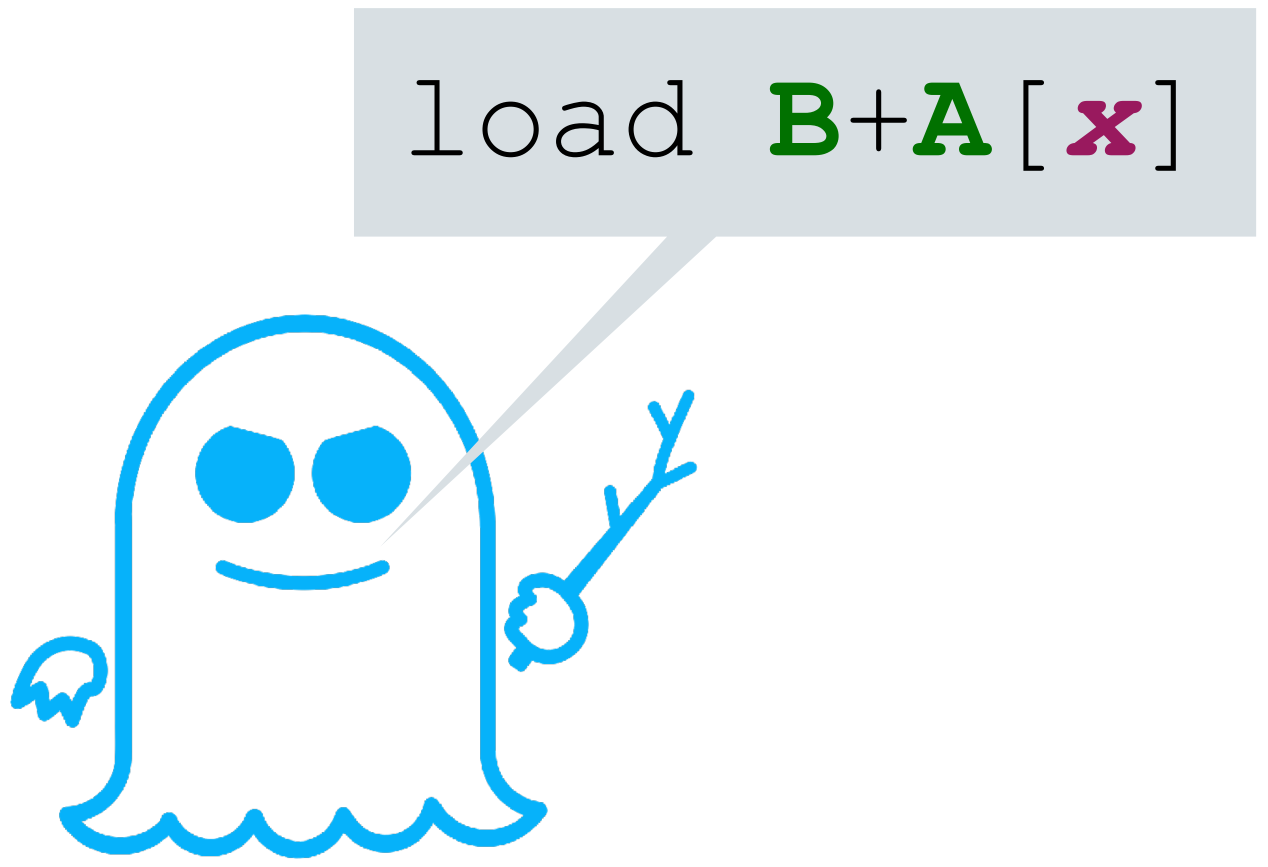
Inspired by "constant-time" requirements

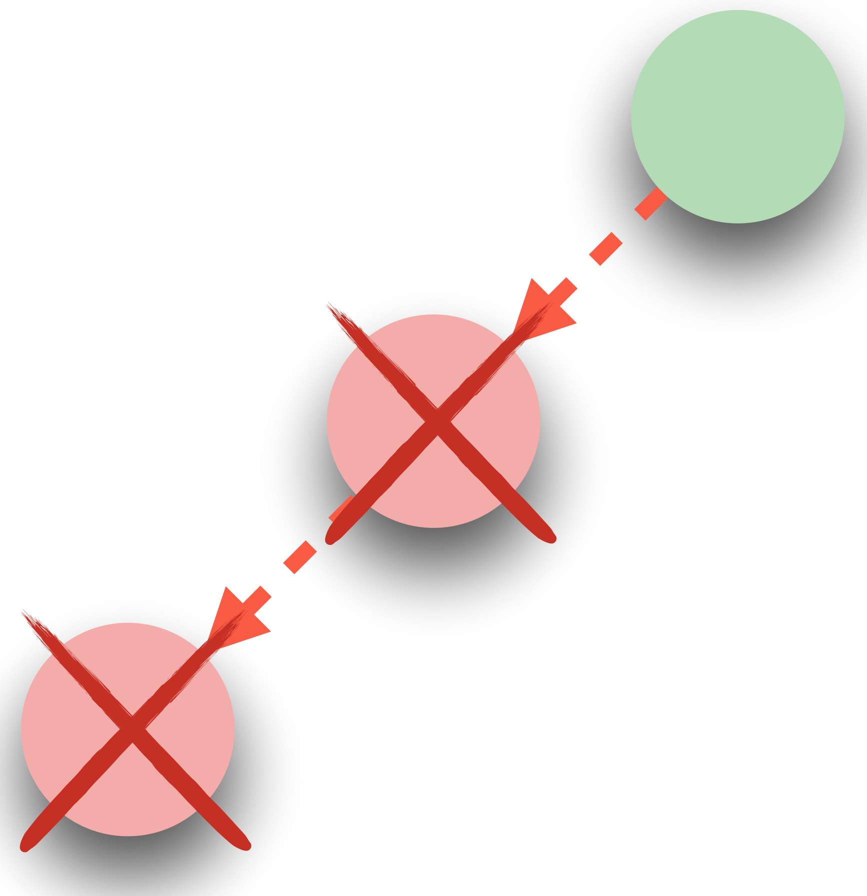
🟩 Non-speculative

🟥 Speculative

Leakage into microarchitecture

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





🟩 Non-speculative

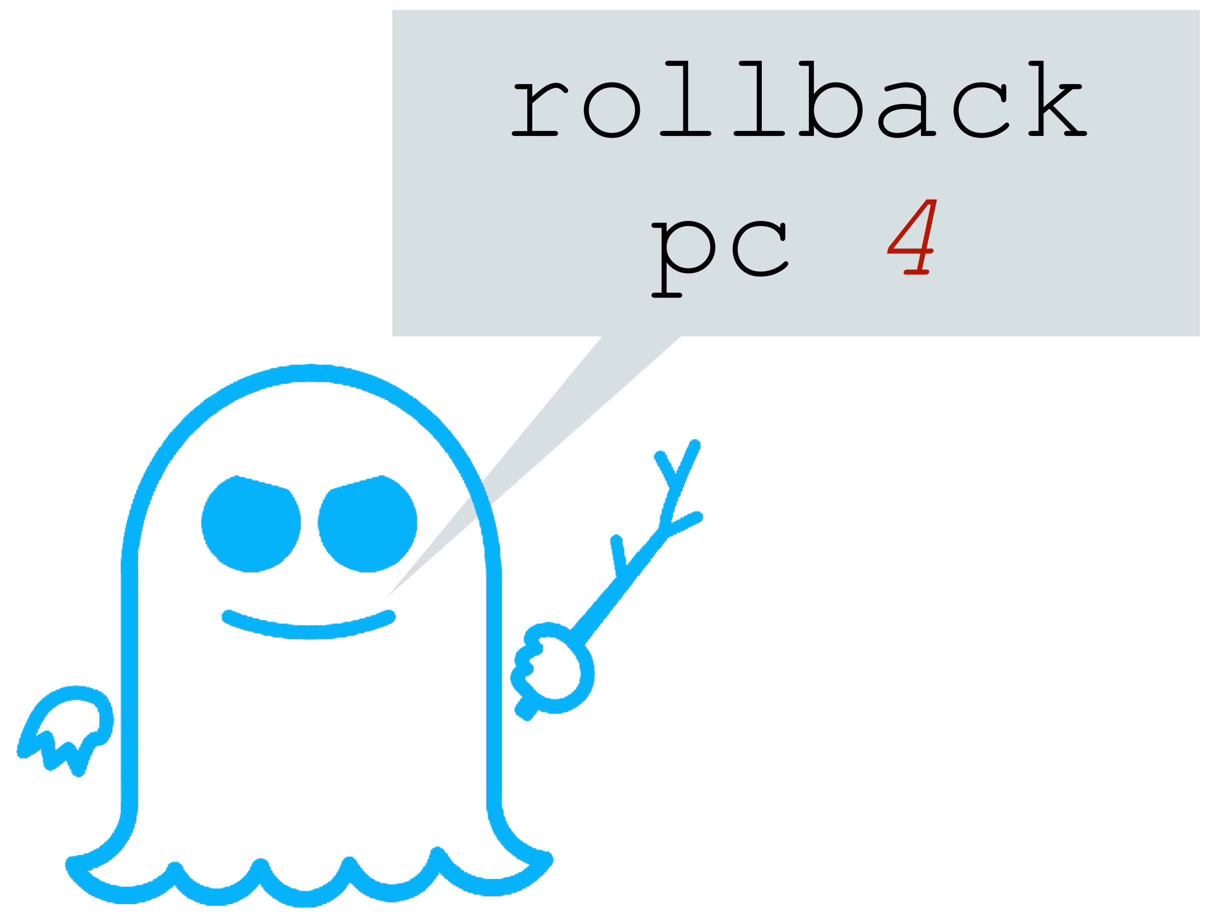
🟥 Speculative

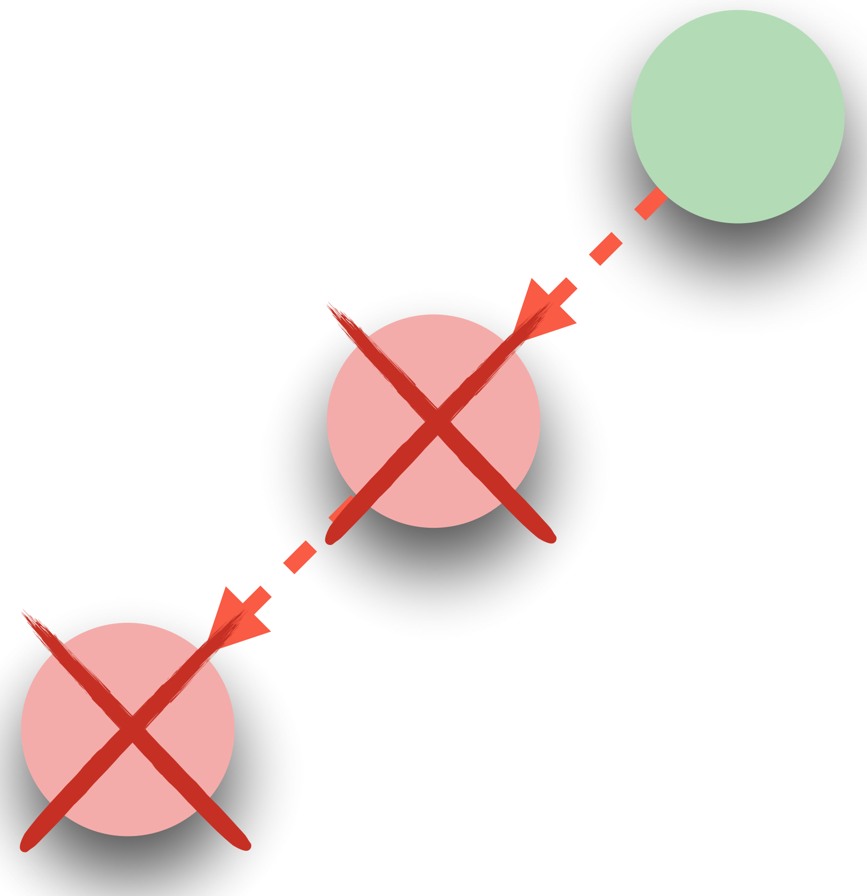
Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” requirements

Leakage into microarchitecture

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





Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" requirements

🟩 Non-speculative

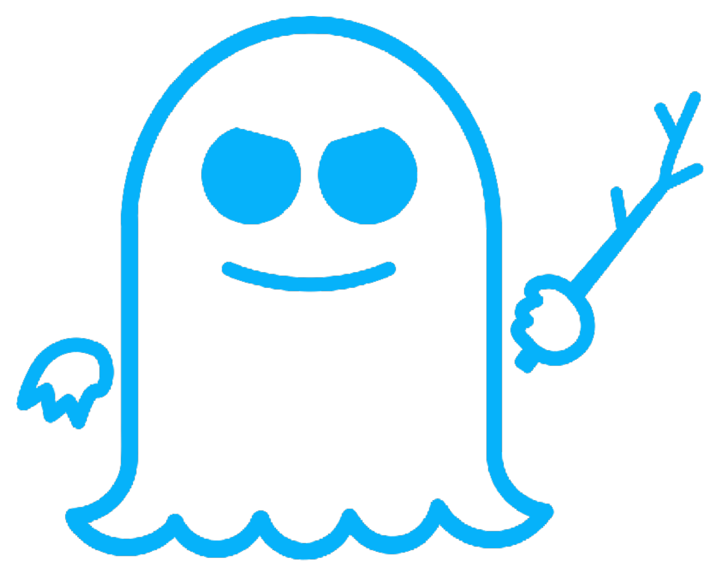
🟥 Speculative

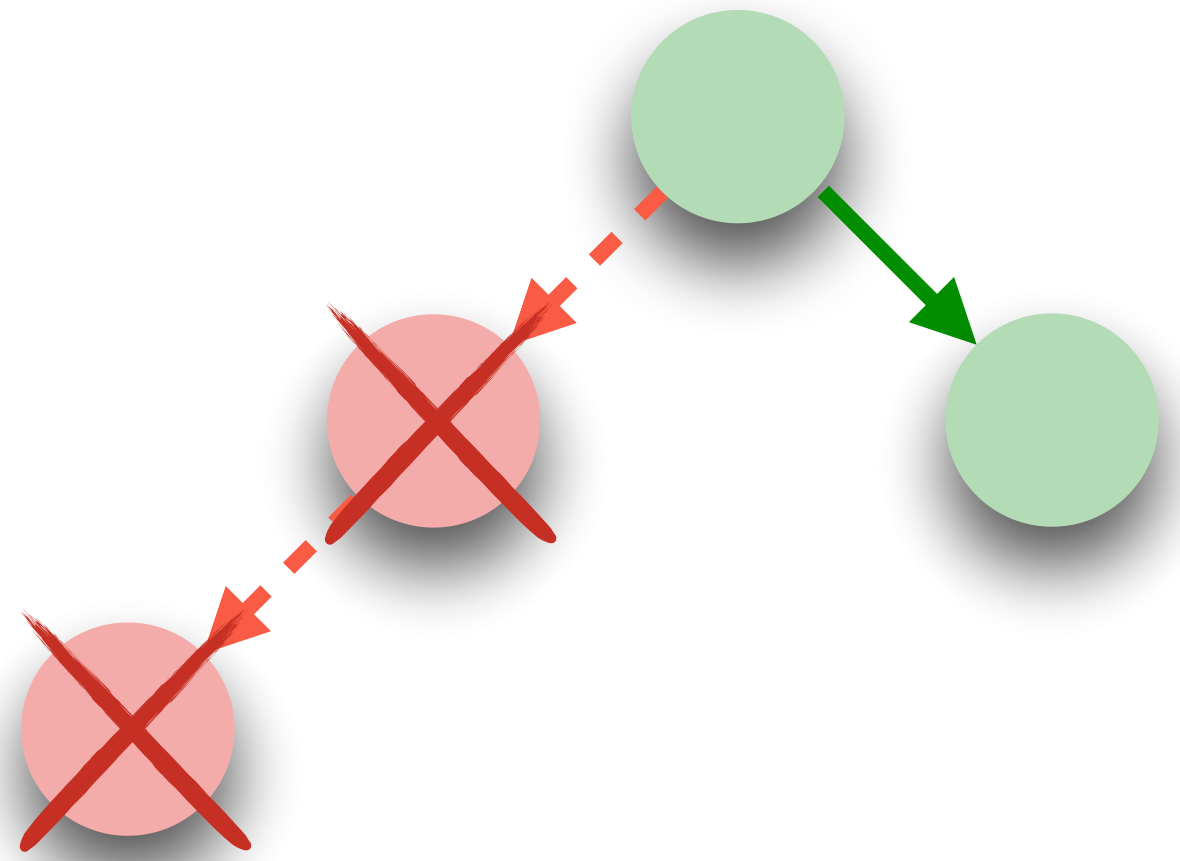
Leakage into microarchitecture

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

Attacker observes:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" requirements





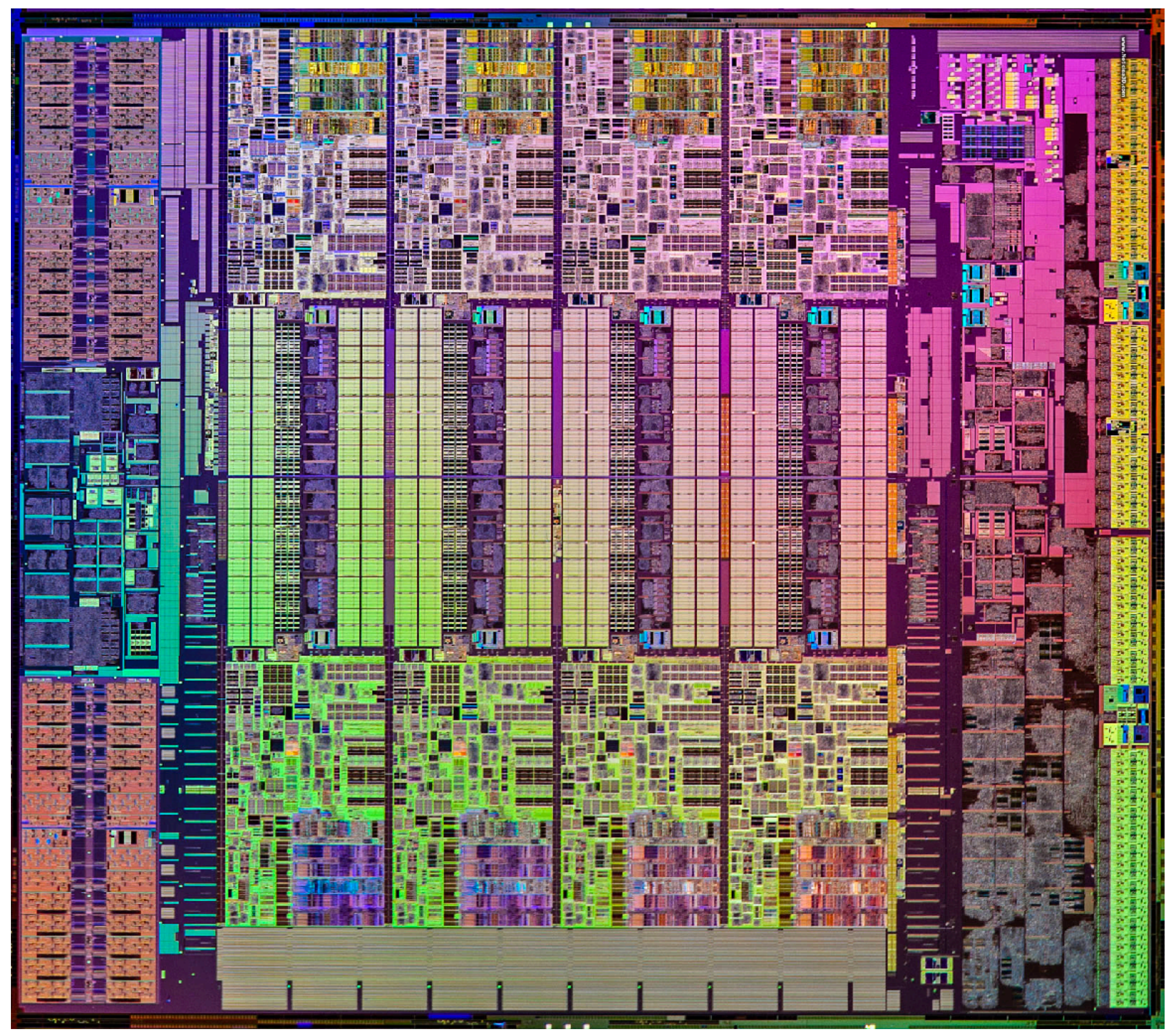
🟩 Non-speculative

🟥 Speculative

Outline

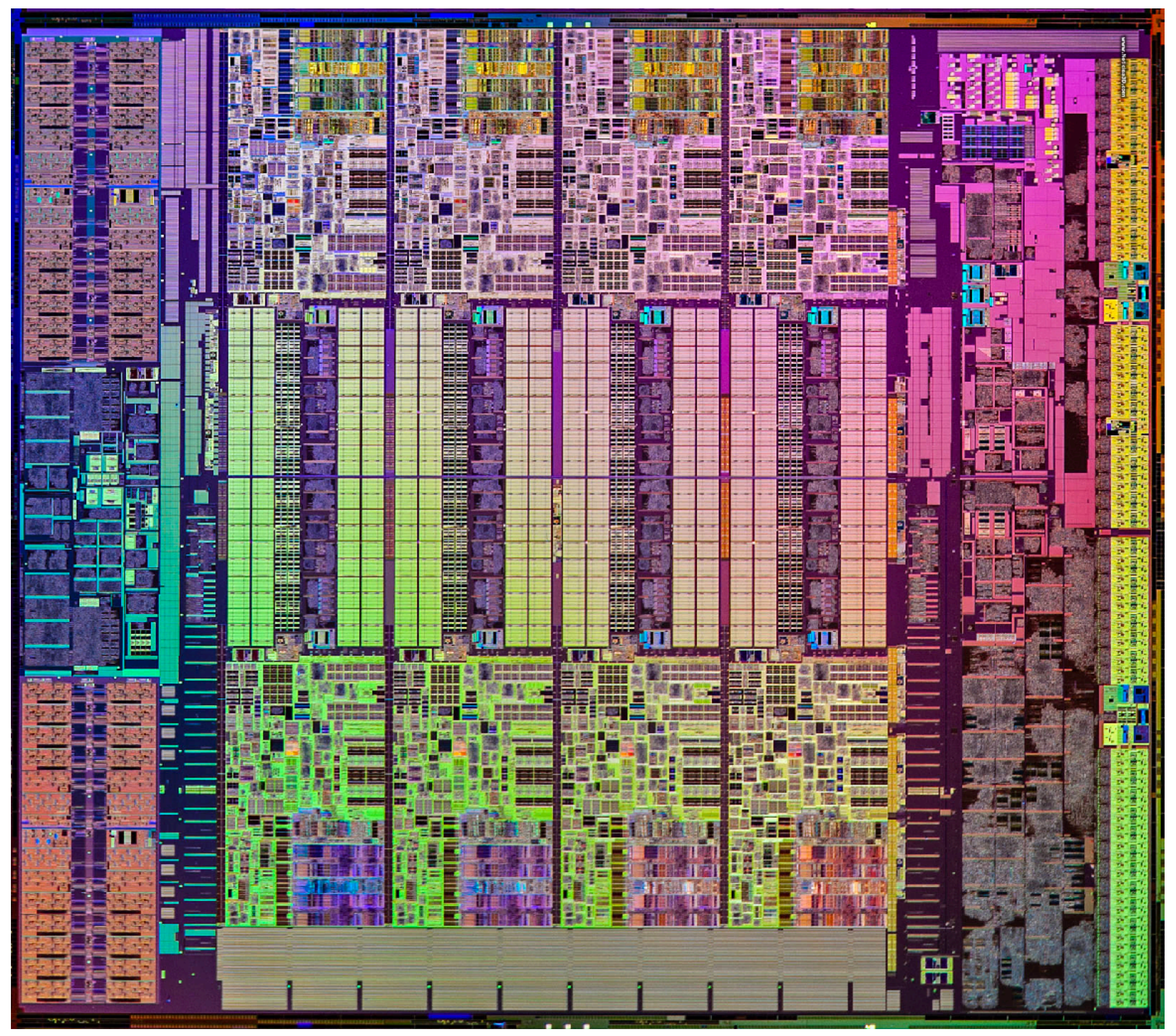
1. Speculative execution attacks
2. Modeling speculative leaks
3. Hardware-software contracts for secure speculation
4. What about hardware?
5. What about software?
6. Conclusions

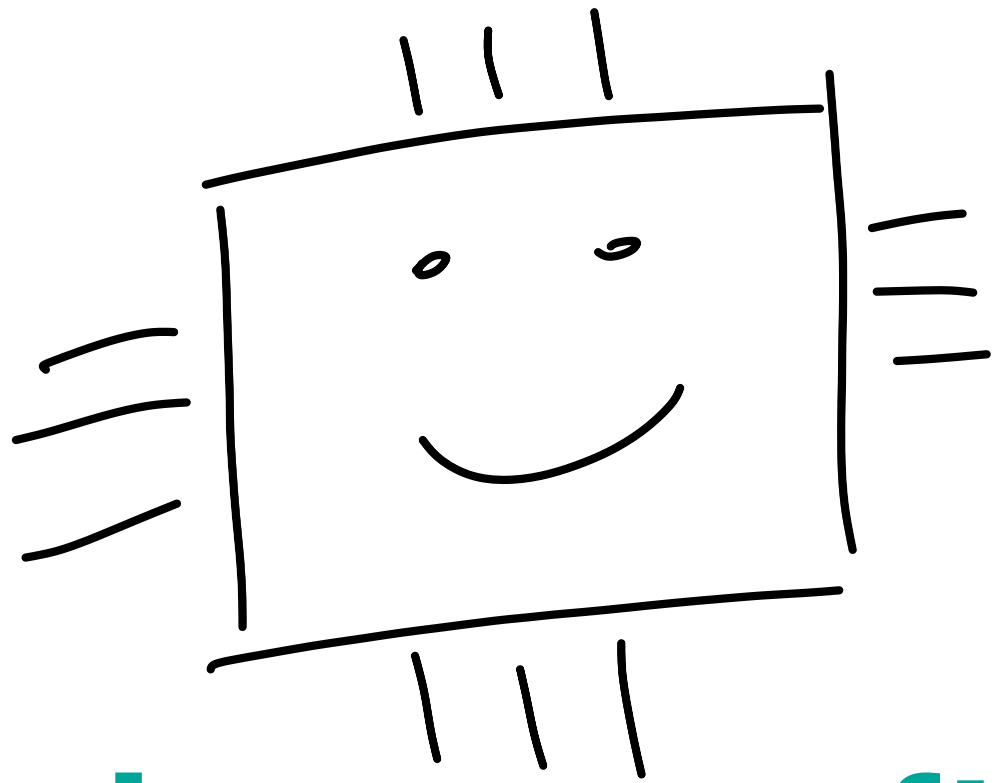
Building sound leakage abstractions



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

Building sound leakage abstractions

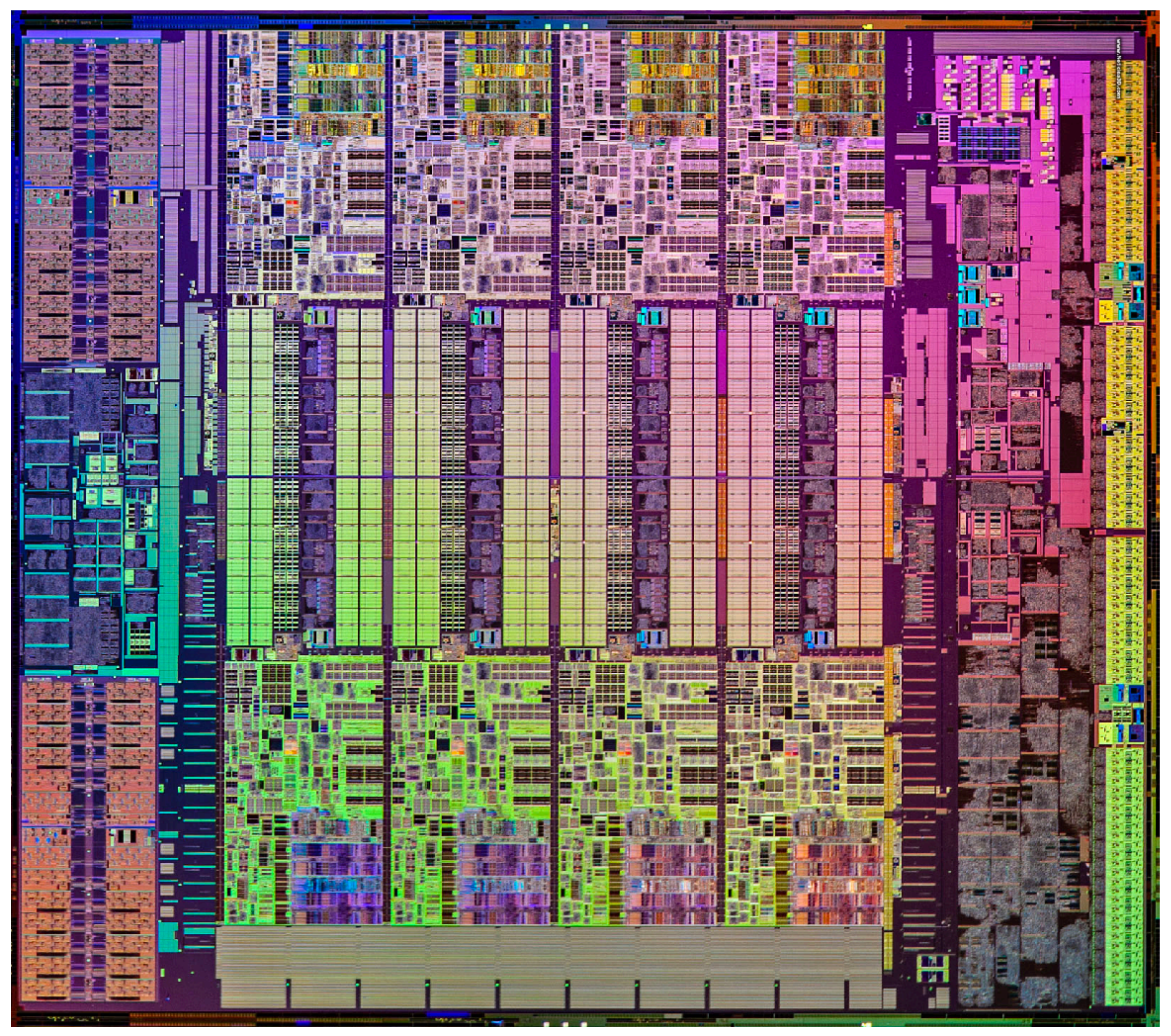


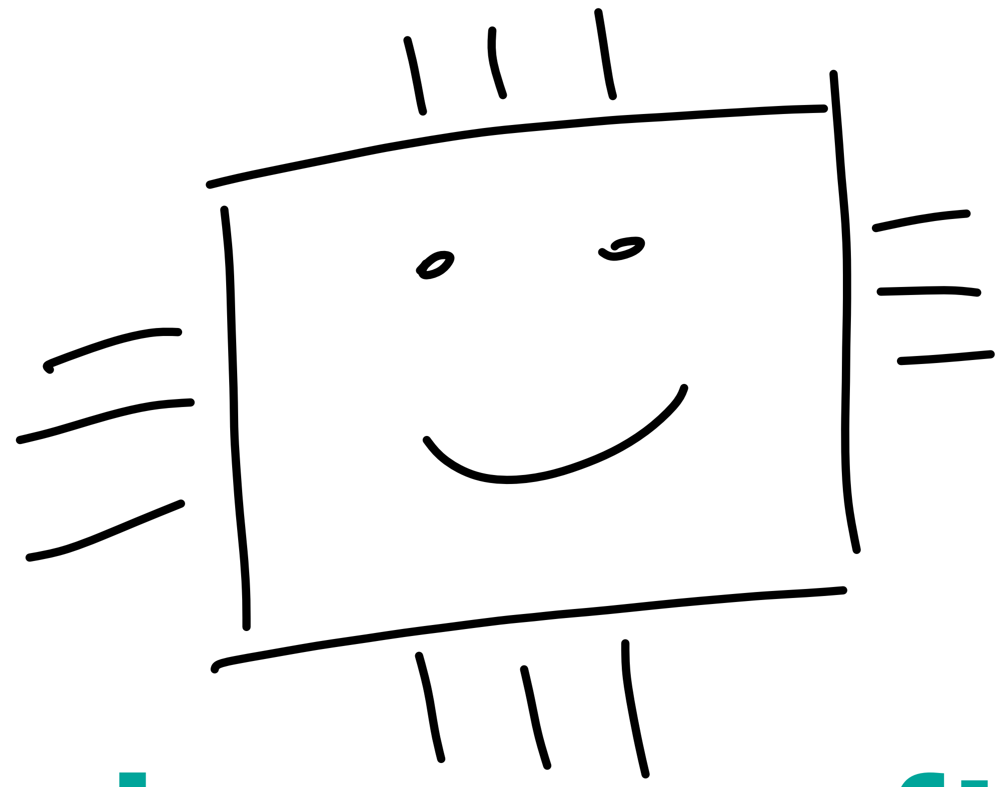


Hardware-software contract

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

Building sound leakage abstractions



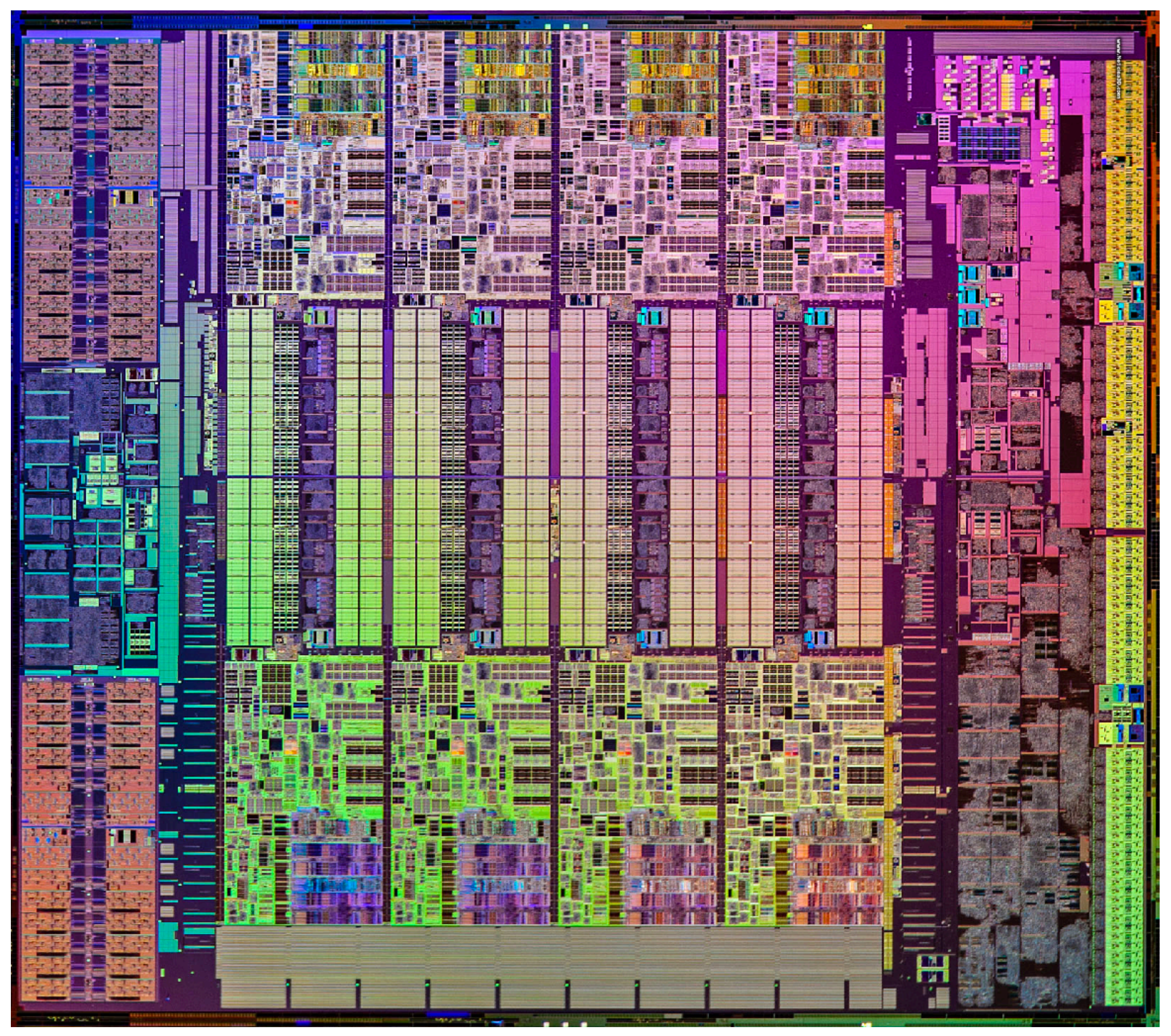


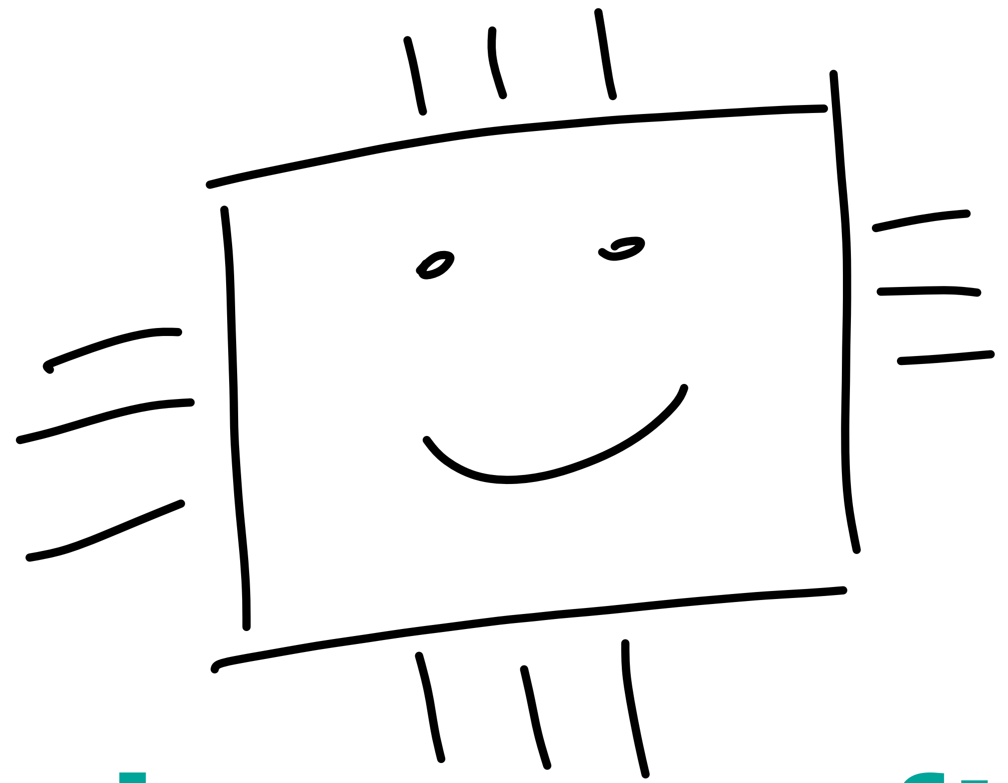
Hardware-software contract

Contracts specify which ***program executions*** a microarchitectural ***adversary can distinguish***

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

Building sound leakage abstractions





Hardware-software contract

Contracts specify which ***program executions*** a microarchitectural ***adversary can distinguish***

Goals

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

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

Contracts

Contracts

Contract

ISA extended with observations

Contracts

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

Contract

ISA extended with observations

Contracts

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

Contract

ISA extended with observations

Contract traces: ▤(p, σ)

Contracts

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

Contract

ISA extended with observations

Contract traces: ≣(p, σ)

Hardware

Processor+attacker observations

Contracts

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

Contract

ISA extended with $\boxed{\text{observations}}$

Contract traces: ▤(p, σ)

Hardware

Processor+attacker observations

Hardware traces: ■(p, σ)

Contracts

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

Hw traces model attacker's observational power

Contract

ISA extended with observations

Contract traces: ≣(p, σ)

Hardware

Processor+attacker observations

Hardware traces: ■(p, σ)

Contracts

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

Hw traces model attacker's observational power

Contract

ISA extended with observations

Contract traces: ≣(p, σ)

Hardware

Processor+attacker observations

Hardware traces: ■(p, σ)

Contract satisfaction

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

Contracts

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

Hw traces model attacker's observational power

Contract

ISA extended with observations

Contract traces: ☰(p, σ)

Hardware

Processor+attacker observations

Hardware traces: ■(p, σ)

Contract satisfaction

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

Contracts

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

Hw traces model attacker's observational power

Contract

ISA extended with observations

Contract traces: ≣(p, σ)

Hardware

Processor+attacker observations

Hardware traces: ■(p, σ)

Contract satisfaction

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

Contracts for secure speculation

Contracts for secure speculation

Contract =

Execution Mode · Observer Mode

Contracts for secure speculation

At ISA level

Contract =

Execution Mode · Observer Mode

Contracts for secure speculation

At ISA level

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

How are programs executed?

Contracts for secure speculation

At ISA level

Contract =

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

How are programs executed?

What is visible about the execution?

Contracts for secure speculation

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

seq — sequential execution
spec — mispredict branch instructions

Contracts for secure speculation

Contract =

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

seq — sequential execution

spec — mispredict branch instructions

Contracts for secure speculation

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

pc — only program counter

ct — **pc** + address of loads/stores

arch — **ct** + loaded values

Contracts for secure speculation

Contract =
Execution Mode · Observer Mode

pc — only program counter

ct — **pc** + address of loads/stores

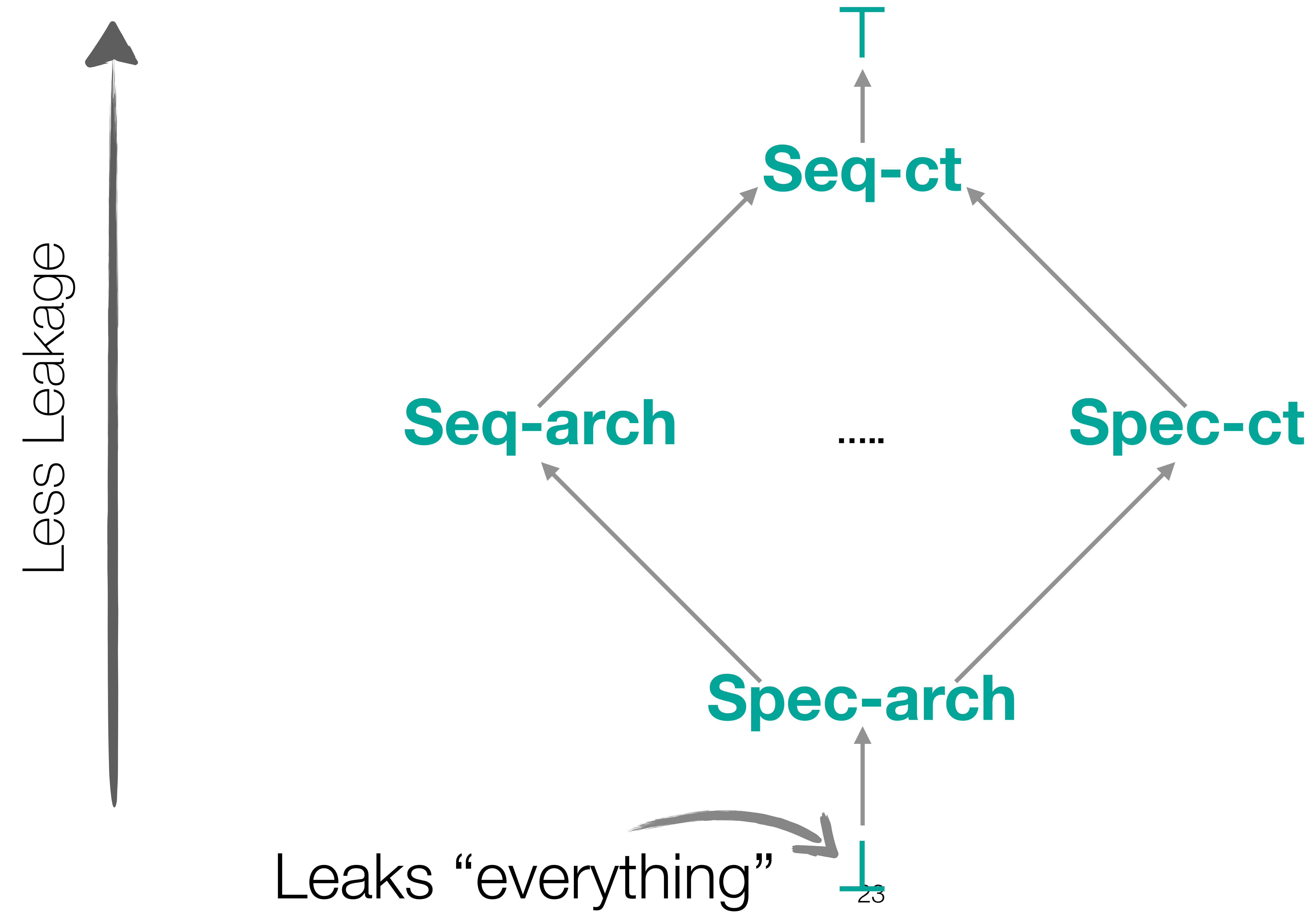
arch — **ct** + loaded values

A lattice of contracts

Less Leakage

$$\begin{tikzcd}
& \top & \\
& \textbf{Seq-ct} \arrow[u] & \\
\textbf{Seq-arch} \arrow[ur] & \text{.....} & \textbf{Spec-ct} \arrow[ul] \\
& \textbf{Spec-arch} \arrow[ul] \arrow[ur] & \\
& \bot \arrow[u] &
\end{tikzcd}$$

A lattice of contracts



$$\begin{tikzcd}
 & \top & \\
 & \text{Seq-ct} \arrow[u] & \\
\text{Seq-arch} \arrow[ur] & \ldots & \text{Spec-ct} \arrow[ul] \\
 & \text{Spec-arch} \arrow[ul] \arrow[ur] & \\
 & \bot \arrow[u] &
\end{tikzcd}$$

A lattice of contracts

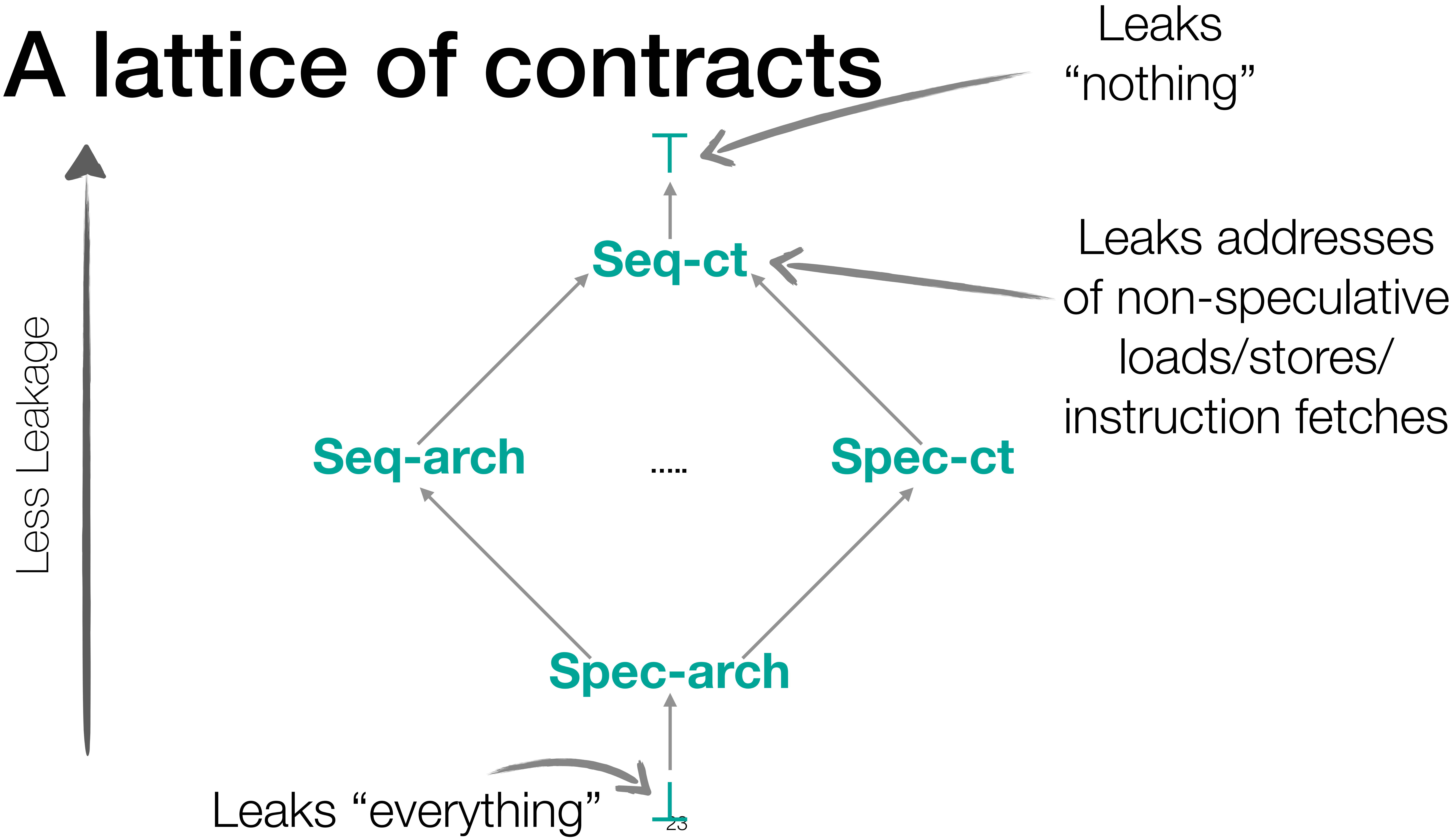
Leaks “nothing”

$$\begin{tikzcd}
 & \top & \\
 & \textbf{Seq-ct} \arrow[u] & \\
\textbf{Seq-arch} \arrow[ur] & \ldots\ldots & \textbf{Spec-ct} \arrow[ul] \\
 & \textbf{Spec-arch} \arrow[ul] \arrow[ur] & \\
 & \bot \arrow[u] &
\end{tikzcd}$$

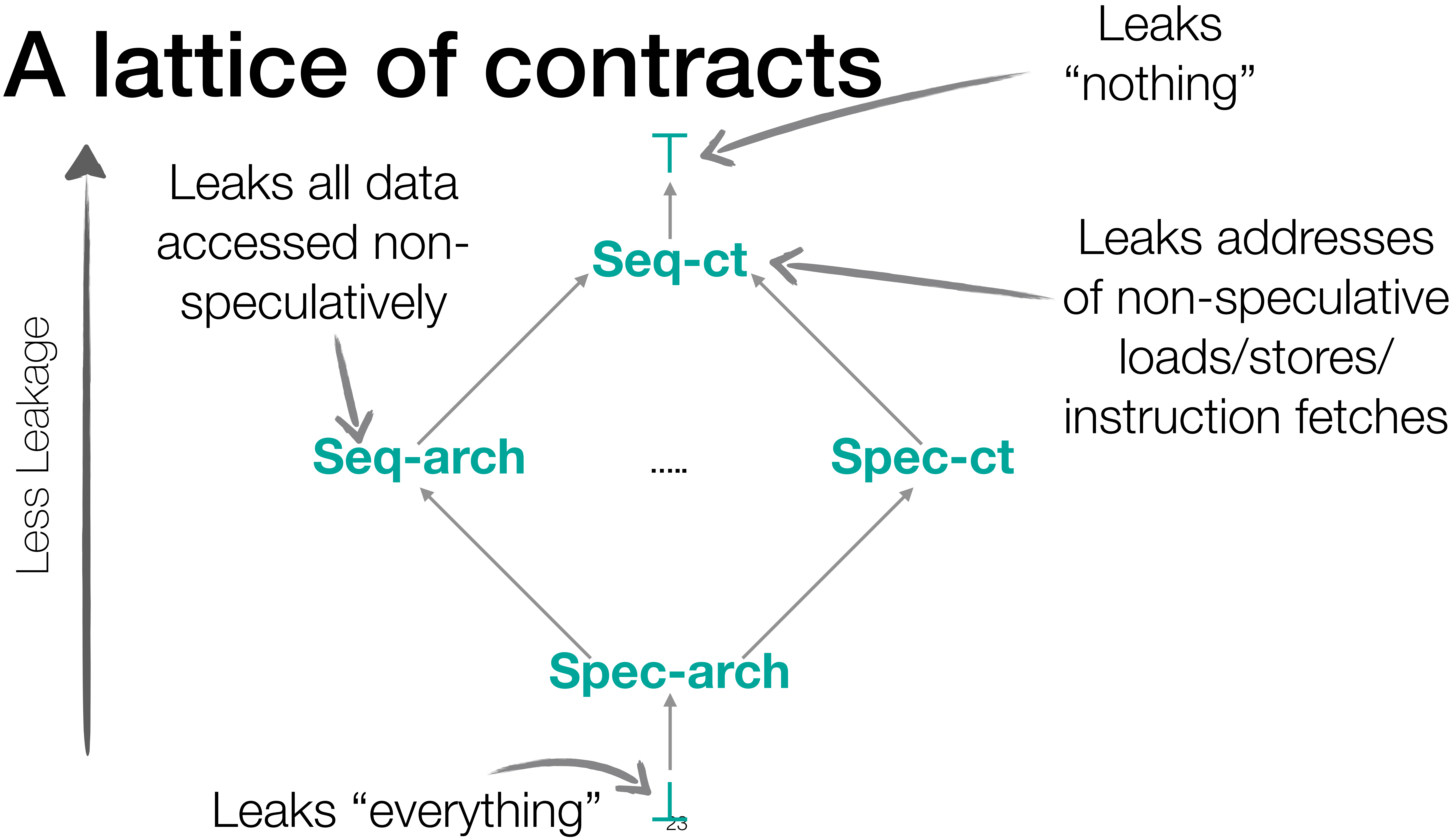
Less Leakage

Leaks “everything”

A lattice of contracts



A lattice of contracts


Leaks "nothing"
⊤
Leaks all data accessed non-speculatively
Seq-ct
Leaks addresses of non-speculative loads/stores/ instruction fetches
Less Leakage
Seq-arch
.....
Spec-ct
Spec-arch
Leaks "everything"
⊥

A lattice of contracts

Less Leakage

$$\begin{tikzcd}
 & \top & \\
 & \textbf{Seq-ct} \arrow[u] & \\
\textbf{Seq-arch} \arrow[ur] & \dots & \textbf{Spec-ct} \arrow[ul] \\
 & \textbf{Spec-arch} \arrow[ul] \arrow[ur] & \\
 & \bot \arrow[u] &
\end{tikzcd}$$

Leaks "nothing"

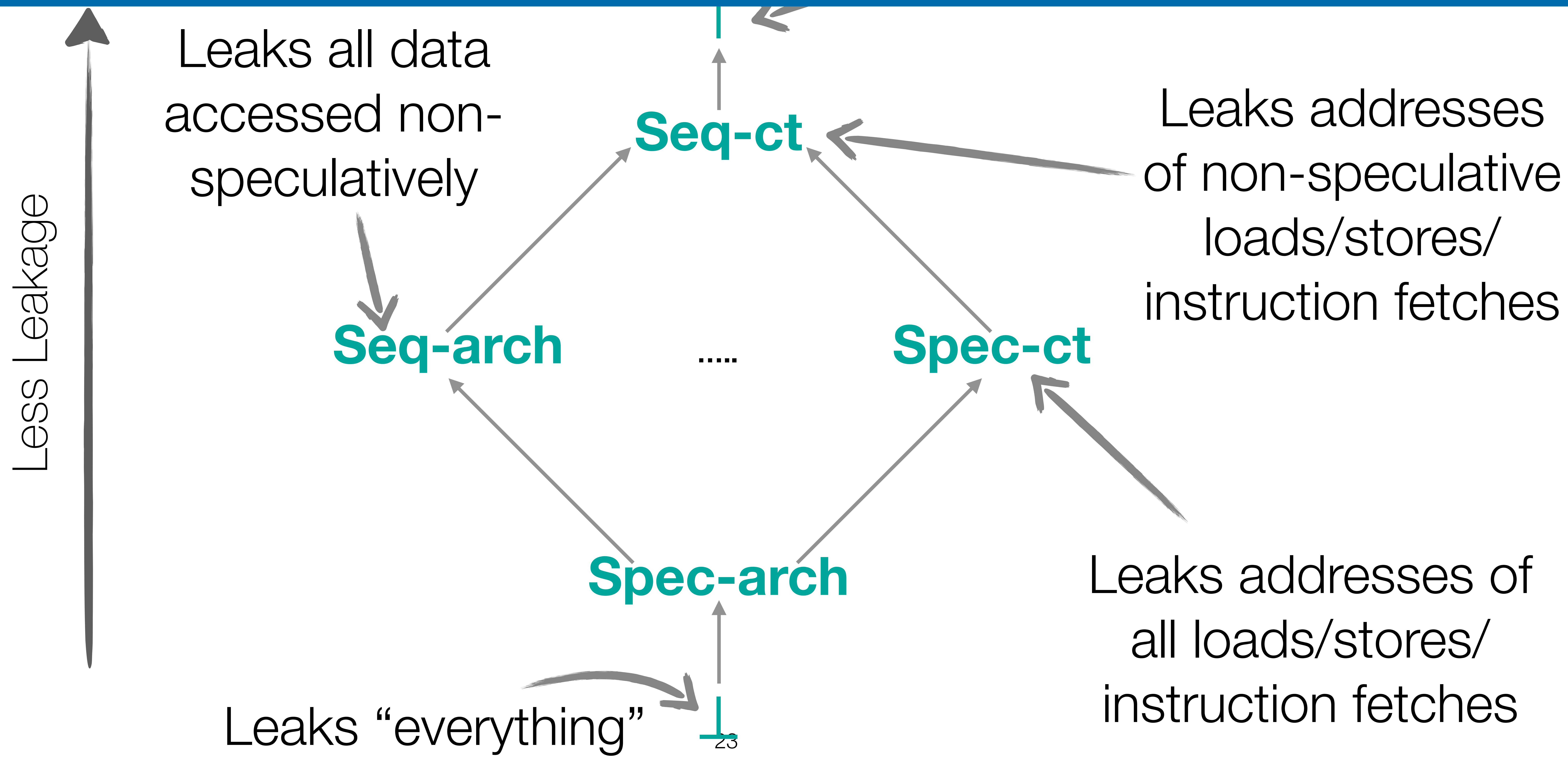
Leaks all data accessed non-speculatively

Leaks addresses of non-speculative loads/stores/ instruction fetches

Leaks addresses of all loads/stores/ instruction fetches

Leaks "everything"

Model different security guarantees! 🥳



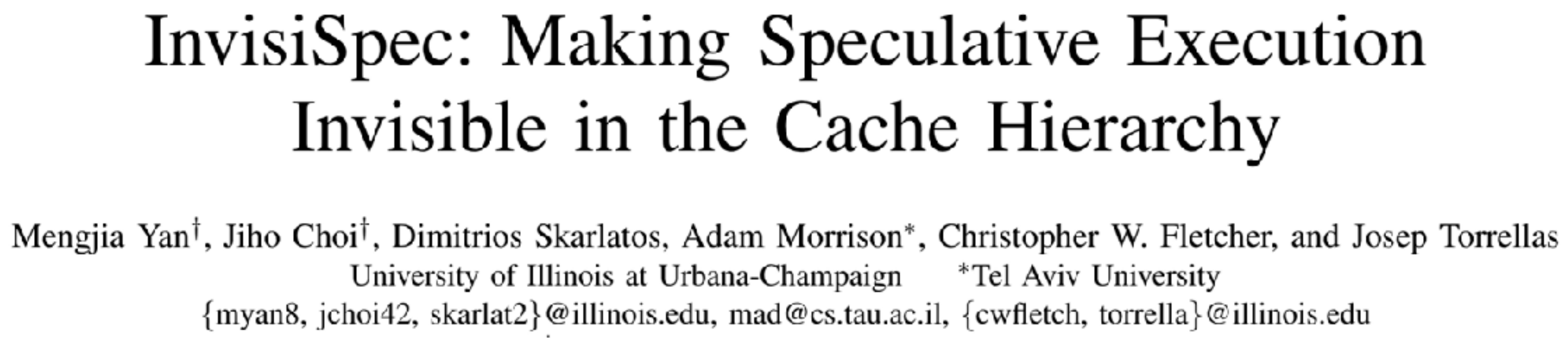
Less Leakage
Leaks all data accessed non-speculatively
Leaks addresses of non-speculative loads/stores/ instruction fetches
Seq-ct
Seq-arch
.....
Spec-ct
Spec-arch
Leaks addresses of all loads/stores/ instruction fetches
Leaks “everything”
⊥

Outline

1. Speculative execution attacks
2. Modeling speculative leaks
3. Hardware-software contracts for secure speculation
4. What about hardware?
5. What about software?
6. Conclusions

Hardware countermeasures

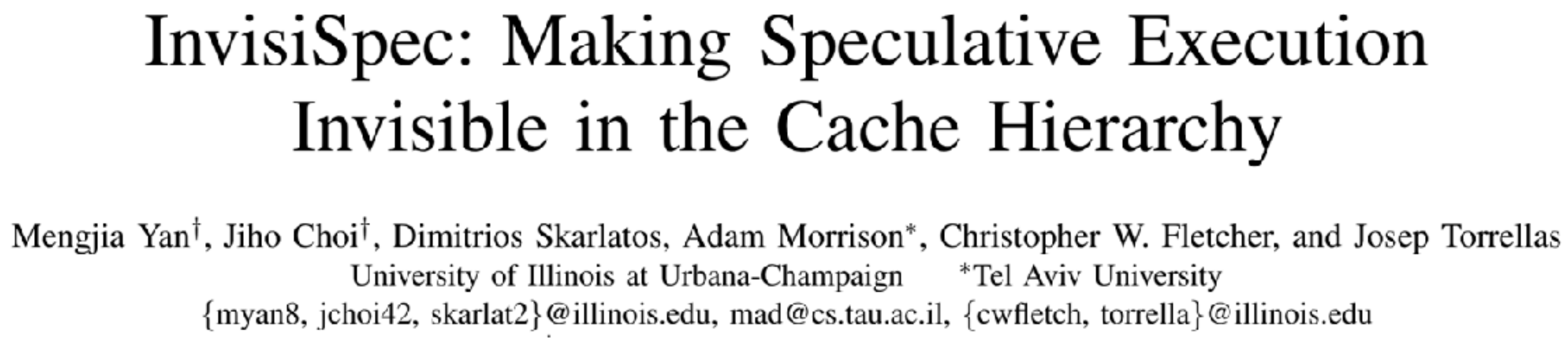
Hardware countermeasures



InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

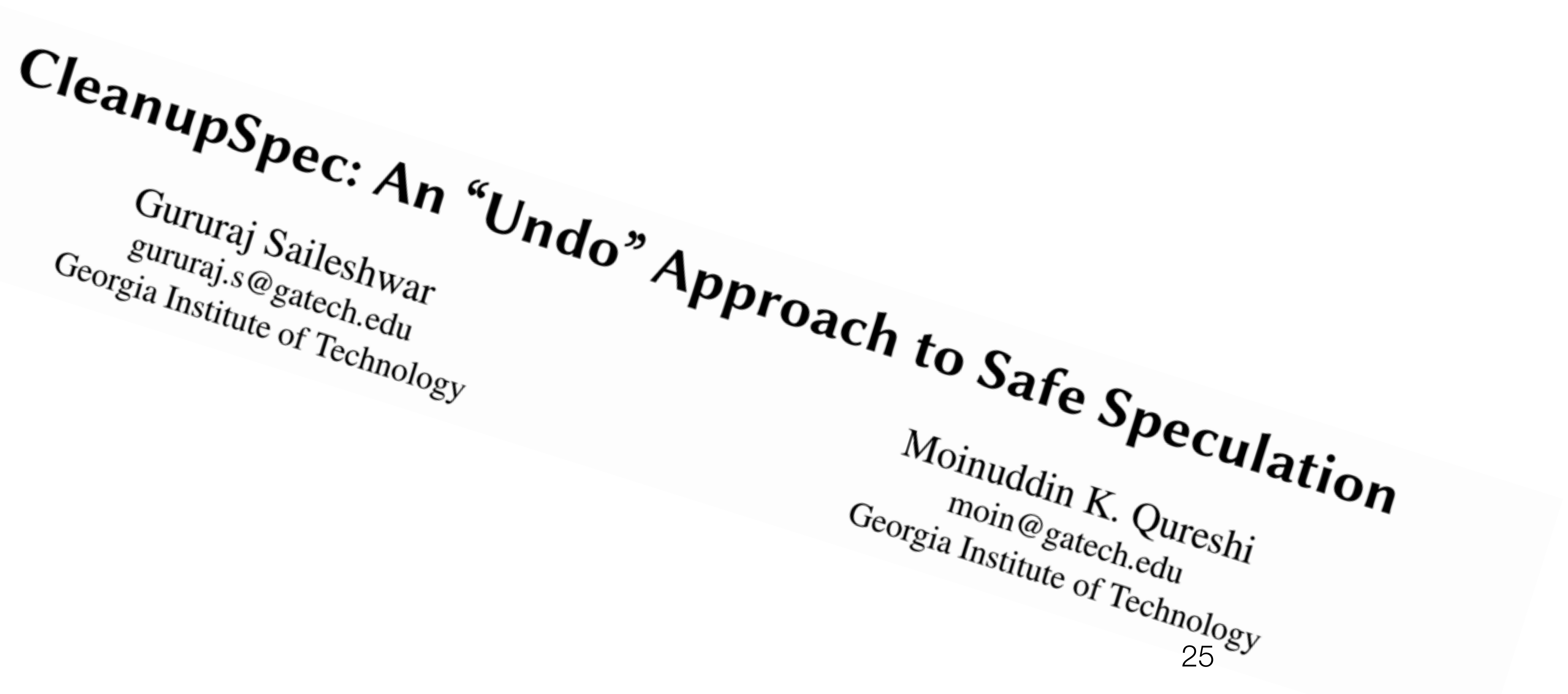
Mengjia Yan[†], Jiho Choi[†], Dimitrios Skarlatos, Adam Morrison[*], Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign [*]Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletch, torrella}@illinois.edu

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

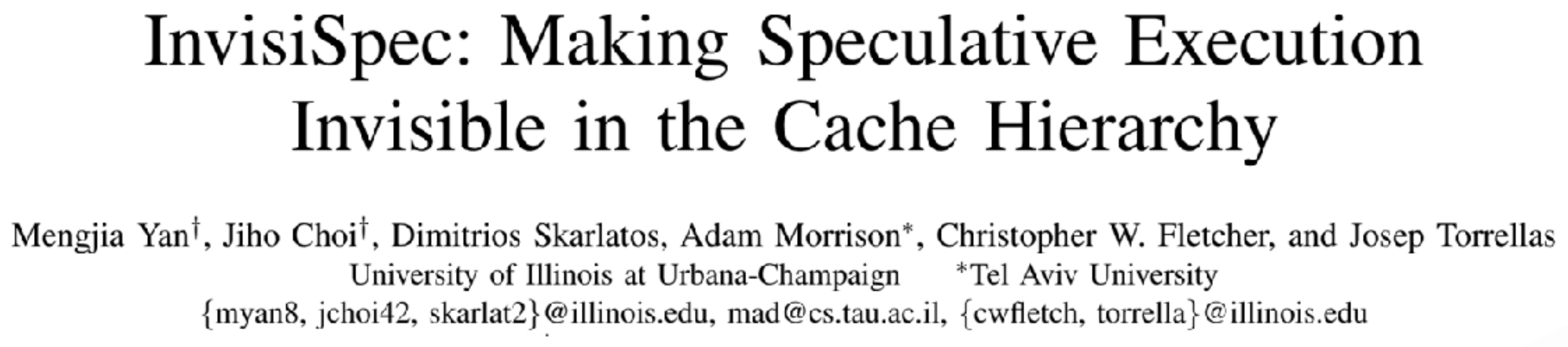


CleanupSpec: An “Undo” Approach to Safe Speculation

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

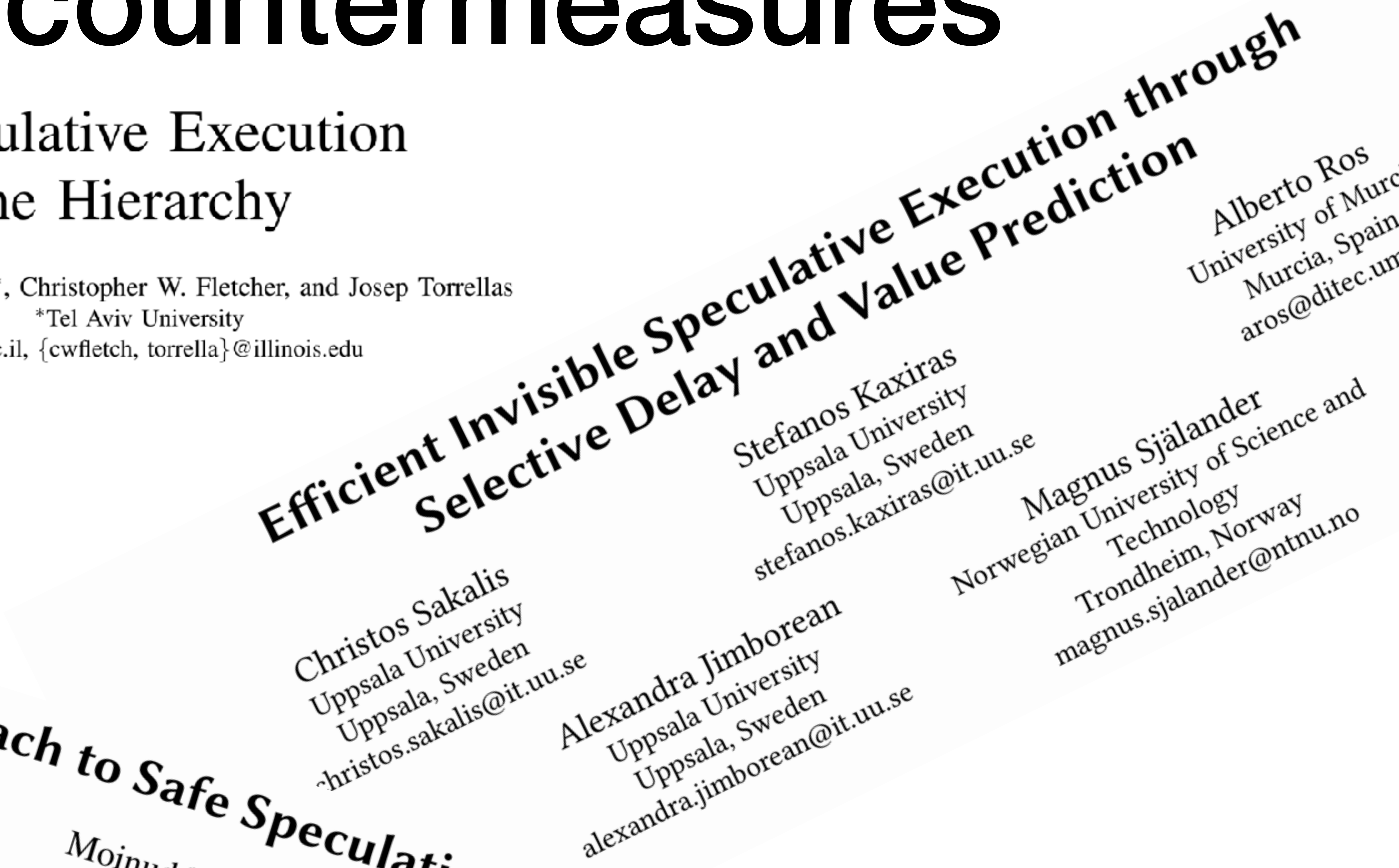
Moinuddin K. Qureshi
moin@gatech.edu
Georgia Institute of Technology

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



Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

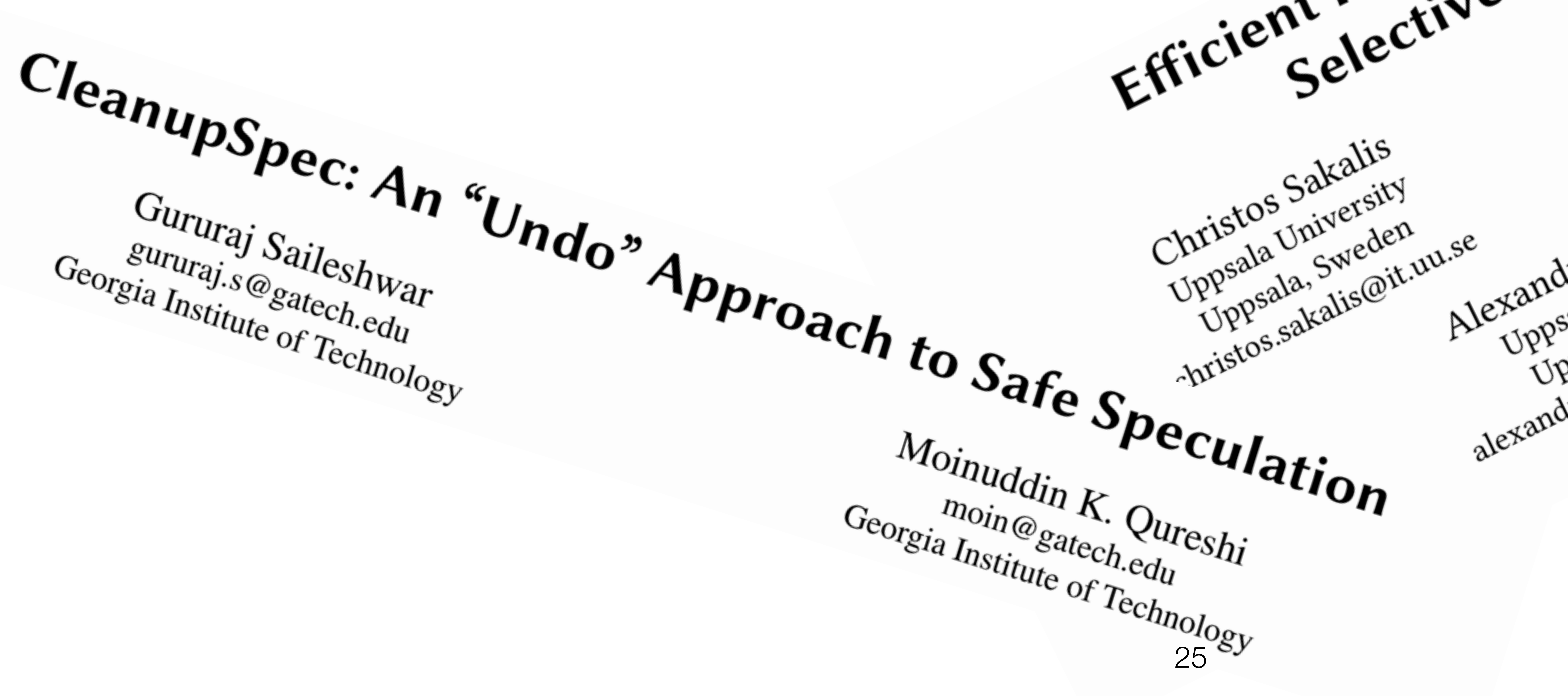
Christos Sakalis
Uppsala University
Uppsala, Sweden
hristos.sakalis@it.uu.se

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

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

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

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



CleanupSpec: An "Undo" Approach to Safe Speculation

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

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

Hardware countermeasures

InvisiSpec: Making Speculative Executi
Invisible in the Cache Hierar

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

NDA: Preventing Speculative Execution Attacks at Their Source

Ofir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Kevin Loughlin
University of Michigan

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

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

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

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

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

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

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

Hardware countermeasures

InvisiSpec: Making Speculative Executi
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

Ofir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Kevin Loughlin
University of Michigan

Efficient Invisible Speculative Execution through
Selective Delay and Value Prediction

Sakalis

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

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

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

CleanupSpec: A

Approach to Safe Speculation

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

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

InvisiSpec: Making Speculative Executi

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

enting Speculative Execution Attacks at Their Source

Ian Neal
University of Michigan

Kevin Loughlin
University of Michigan

Baris Kasikci
University of Michigan

visible Speculative Execution through

Delay and Value Prediction

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

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

Security guarantees?

Hardware countermeasures

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

Hardware countermeasures

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

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

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

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

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

Delay loads until they are no longer speculative
[Sakalis et al., ISCA'19]

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

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

Delay loads until they are no longer speculative
[Sakalis et al., ISCA'19]

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

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

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

Hardware countermeasures

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

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

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

Delay loads until they are no longer speculative

[Sakalis et al., ISCA'19]

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

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

Delay loads until they are no longer speculative
[Sakalis et al., ISCA'19]

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

🟩 Non-speculative

🟥 Speculative

Hardware countermeasures

```
y = A[x]
```

Delay loads until they are no longer speculative
[Sakalis et al., ISCA'19]

Countermeasures block different leaks!

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

🟩 Non-speculative

🟥 Speculative

Guarantees

$$\begin{tikzcd}
 & \text{Seq-ct} \\
\text{Seq-arch} \arrow[ur] & \\
 & \text{Seq/spec-ct/pc} \arrow[uu] \\
 & \text{Spec-ct} \arrow[u] \\
\text{Spec-arch} \arrow[uuu] \arrow[ur] &
\end{tikzcd}$$

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Guarantees

$$\begin{tikzcd}
 & \text{Seq-ct} \\
\text{Seq-arch} \arrow[ur] & \\
 & \text{Seq/spec-ct/pc} \arrow[uu] \\
 & \text{Spec-ct} \arrow[u] \\
\text{Spec-arch} \arrow[uuu] \arrow[ur] &
\end{tikzcd}$$

3-stage pipeline with ***speculative*** and ***out-of-order*** (OoO) execution

Formalized as ***operational semantics***

Attacker observes part of ***microarchitectural state***

OoO Vanilla out-of-order (OoO) CPU

NS In-order CPU (no speculative execution)

LD OoO CPU+load delay

TT OoO CPU+taint tracking

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

Guarantees

$$\begin{tikzcd}
 & \text{Seq-ct} \\
\text{Seq-arch} \arrow[ur] & \\
 & \text{Seq/spec-ct/pc} \arrow[uu] \\
 & \text{Spec-ct} \arrow[u] \\
\text{Spec-arch} \arrow[uuu] \arrow[ur] & \text{OoO}
\end{tikzcd}$$

3-stage pipeline with ***speculative*** and ***out-of-order*** (OoO) execution

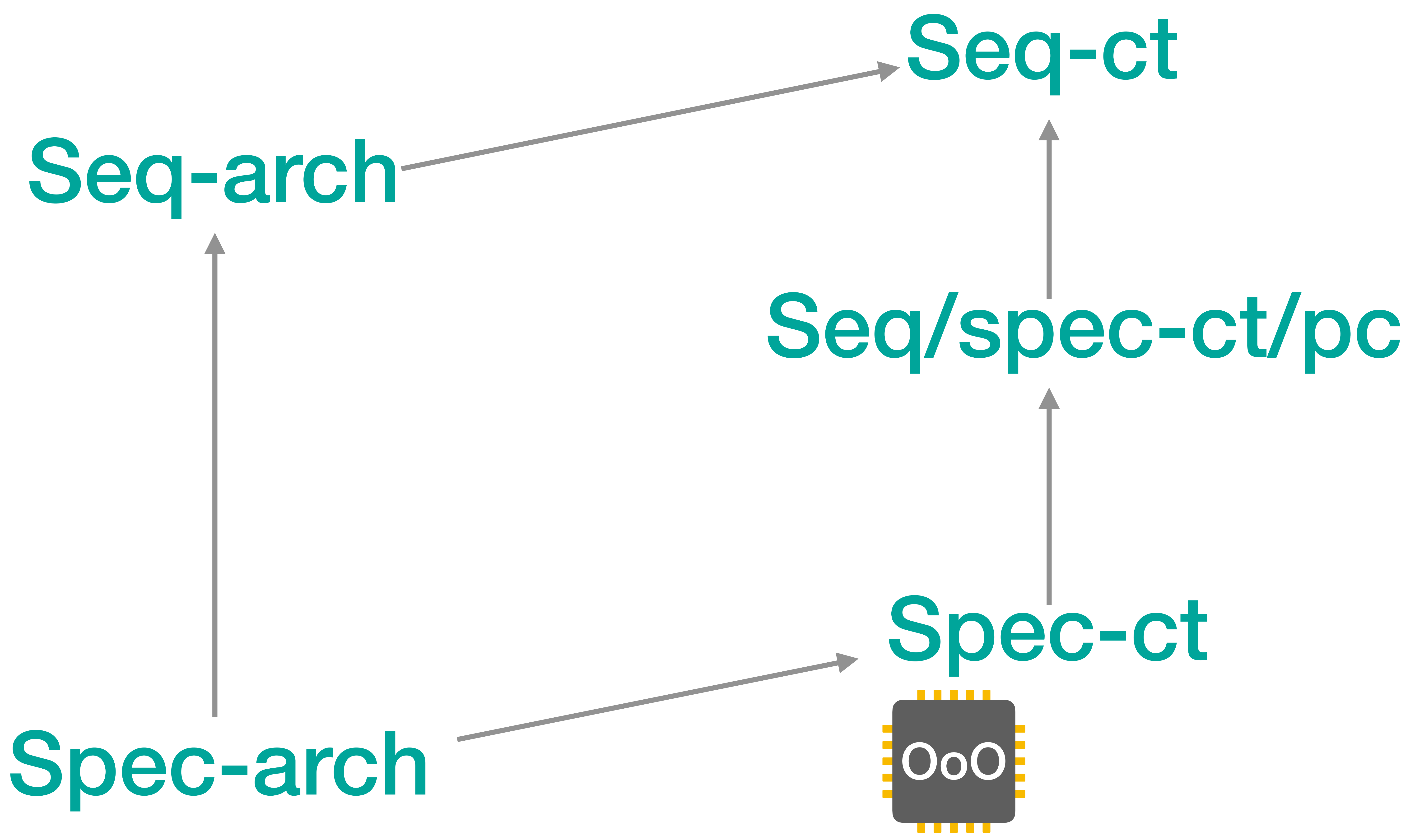
Formalized as ***operational semantics***

Attacker observes part of ***microarchitectural state***

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Guarantees



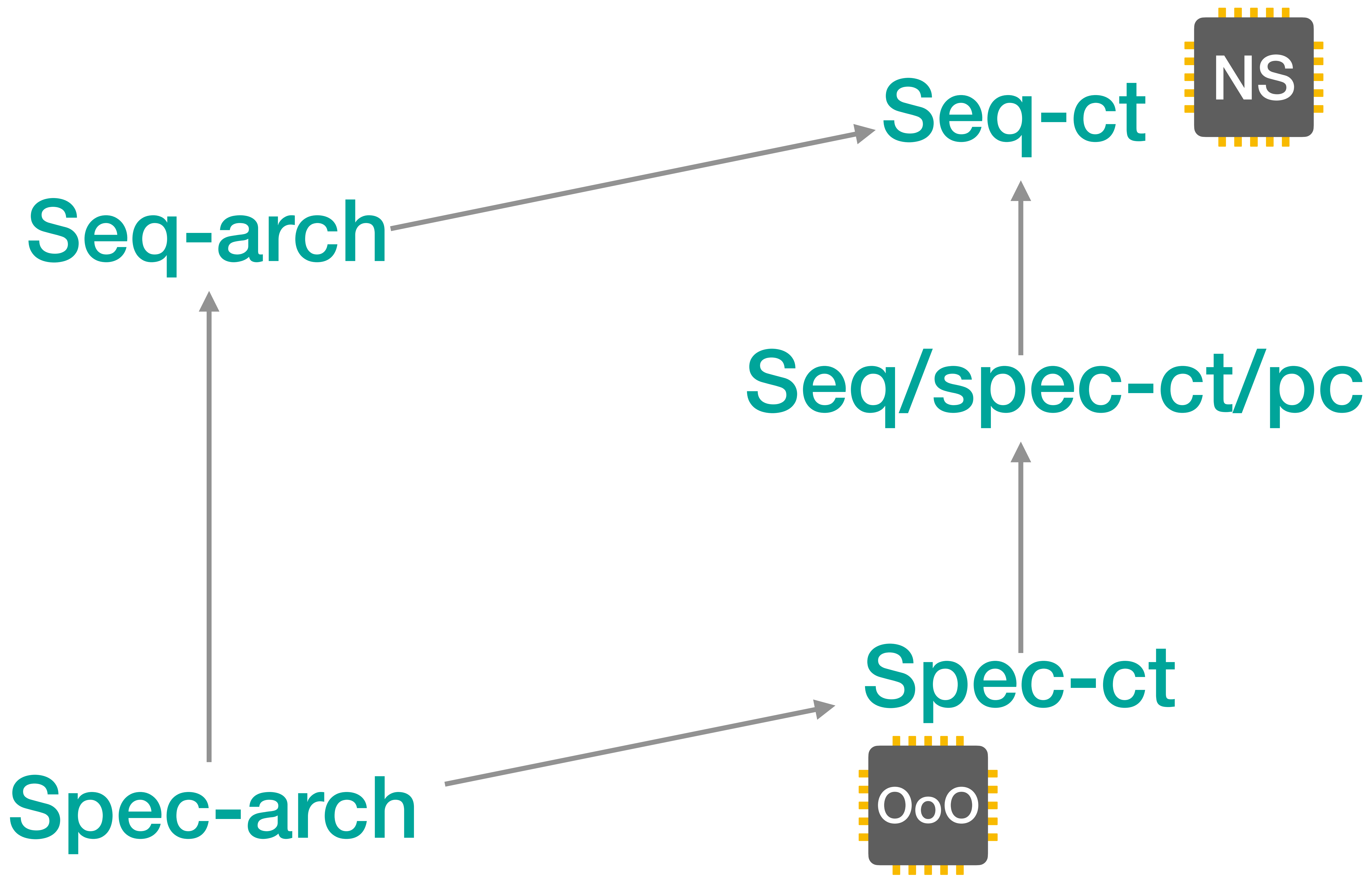
No speculative and out-of-order execution

Instructions executed ***in-order***

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Guarantees



No speculative and out-of-order execution

Instructions executed ***in-order***

OoO Vanilla out-of-order (OoO) CPU

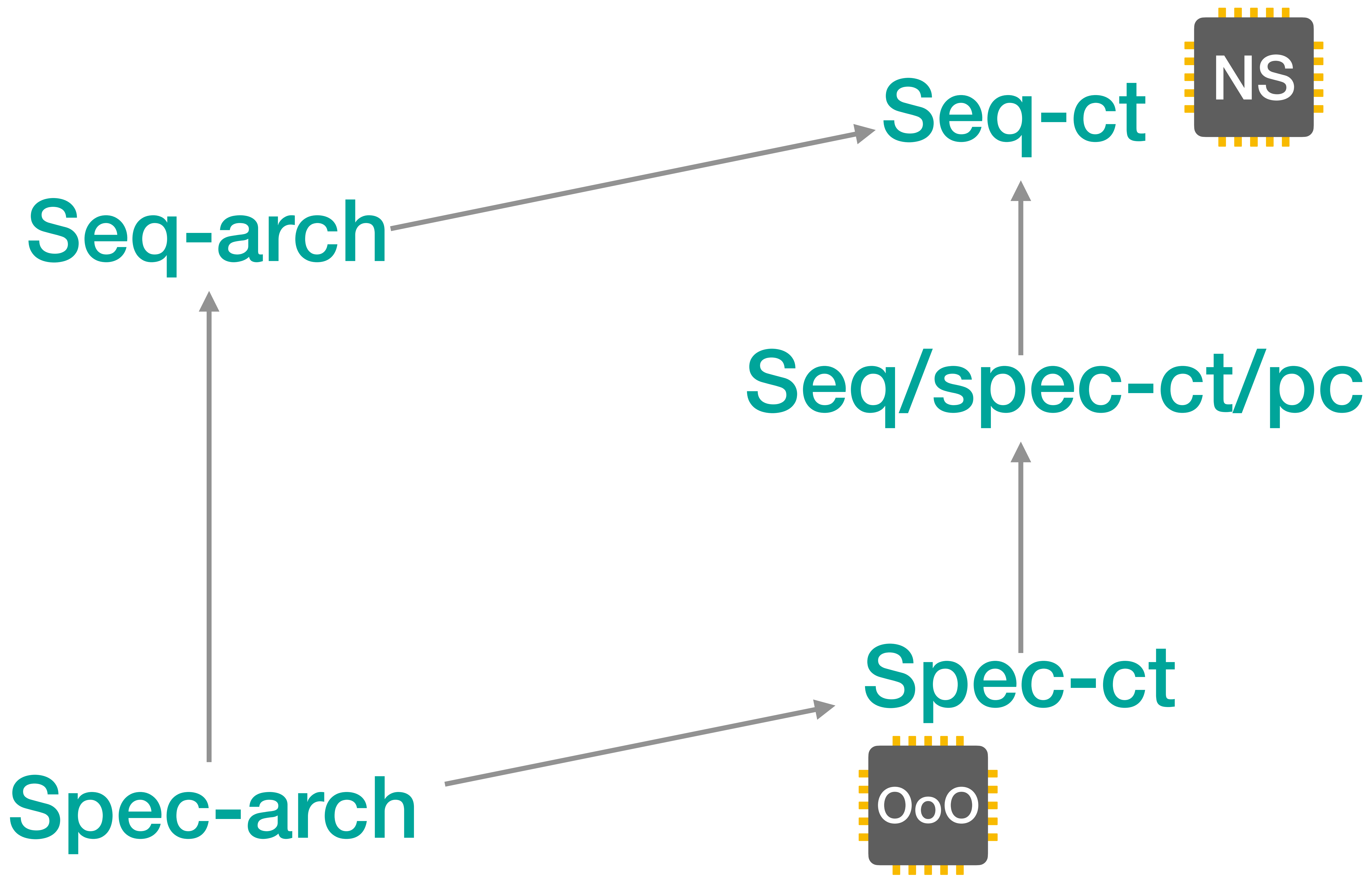
NS In-order CPU (no speculative execution)

LD OoO CPU+load delay

TT OoO CPU+taint tracking

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

Guarantees



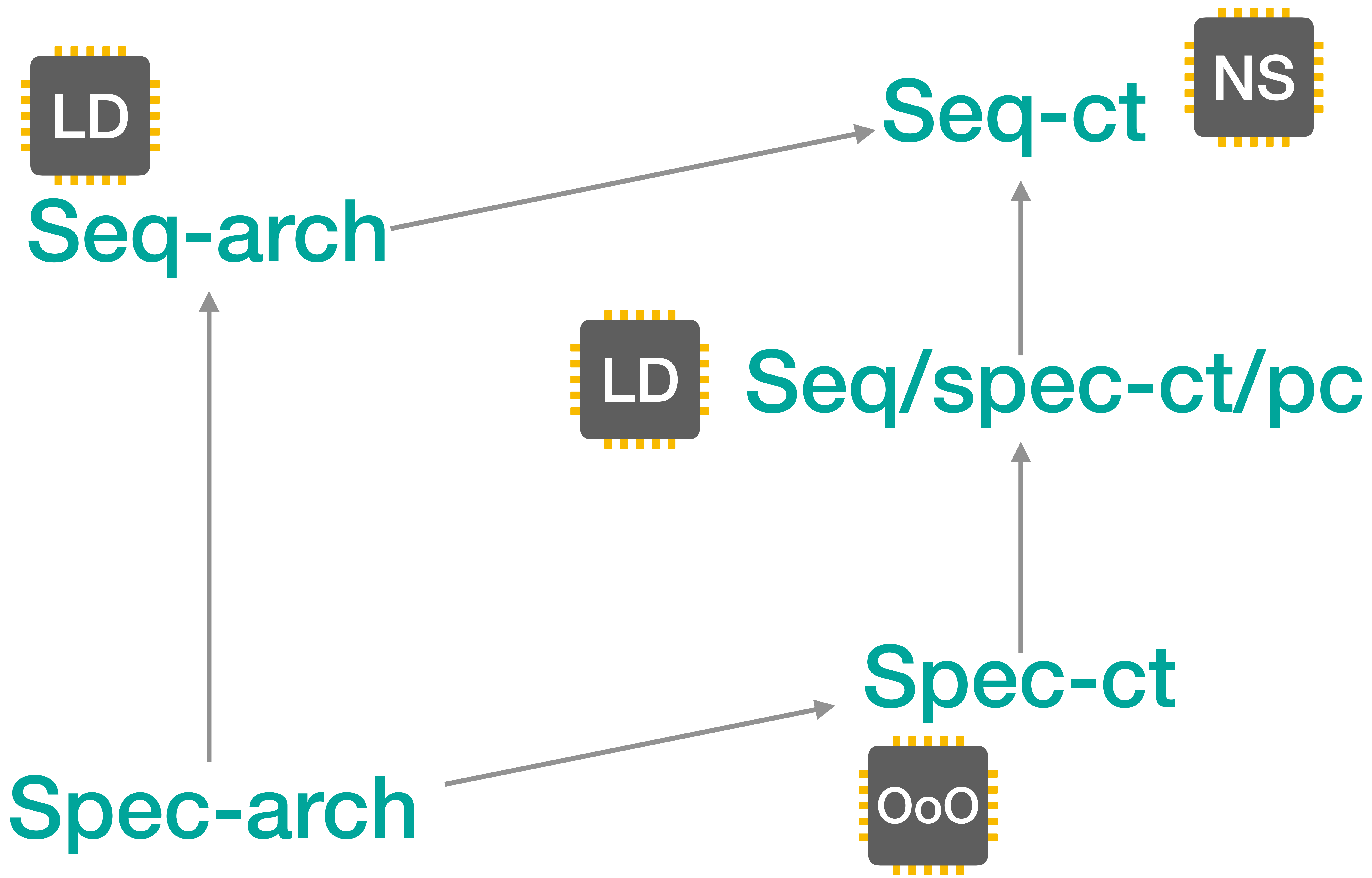
Delaying loads until all sources of ***speculation are resolved***

Sakalis et al., ISCA'19

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Guarantees



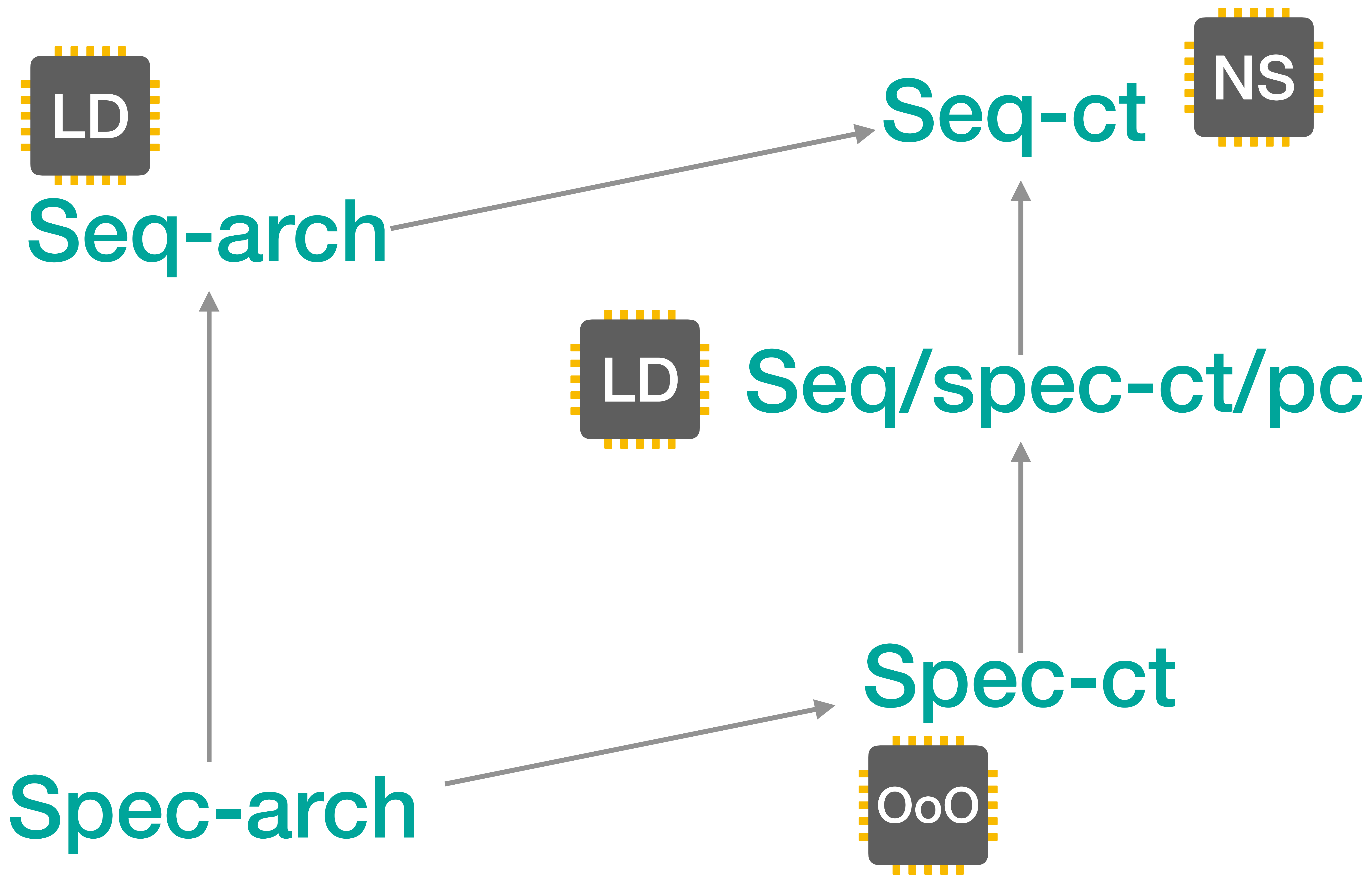
Delaying loads until all sources of ***speculation are resolved***

Sakalis et al., ISCA'19

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Guarantees



Taint speculative data

Propagate taint through computation

Delay tainted operations

STT and NDA, MICRO'19

OoO Vanilla out-of-order (OoO) CPU

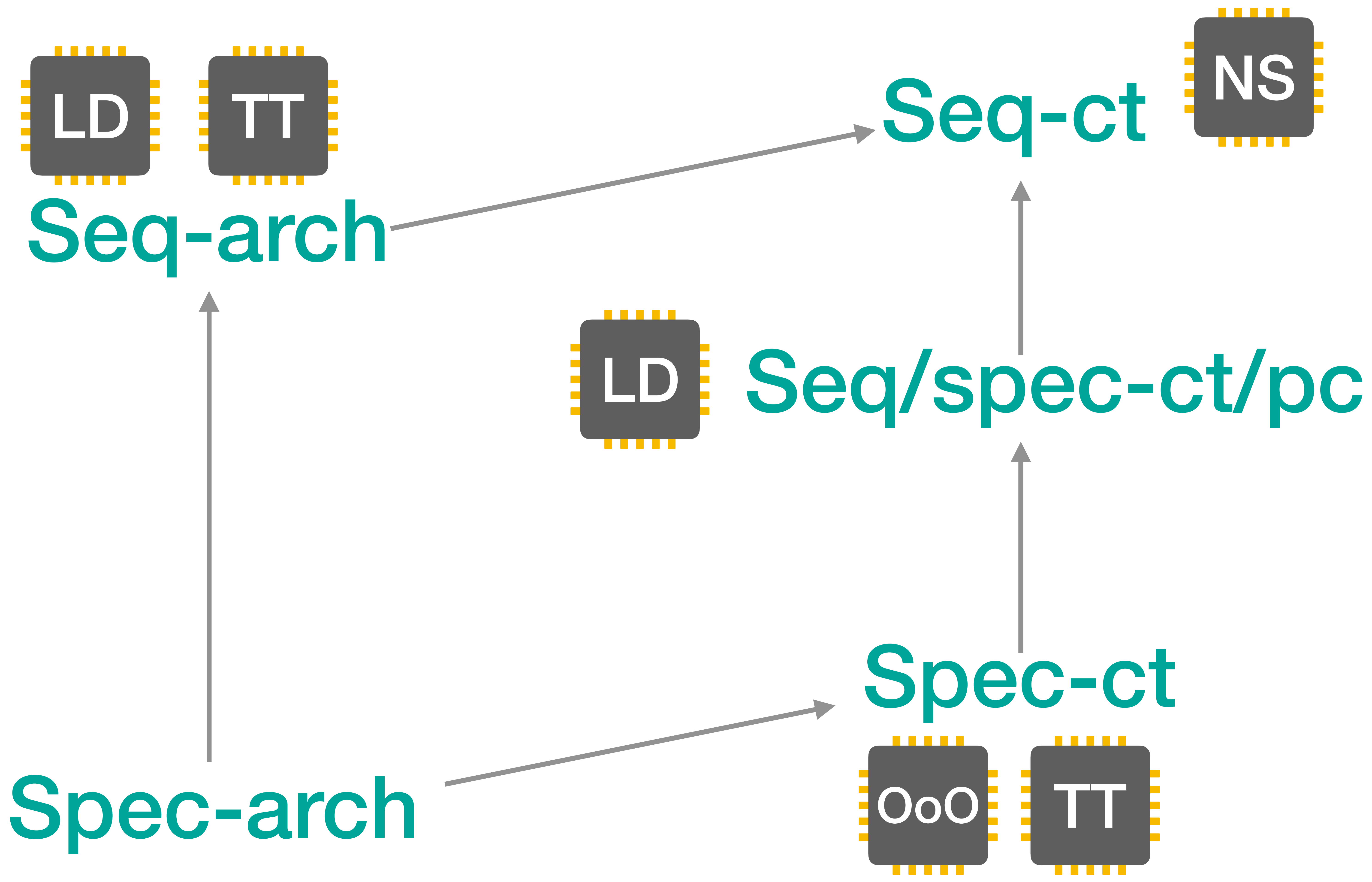
NS In-order CPU (no speculative execution)

LD OoO CPU+load delay

TT OoO CPU+taint tracking

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

Guarantees



Taint speculative data

Propagate taint through computation

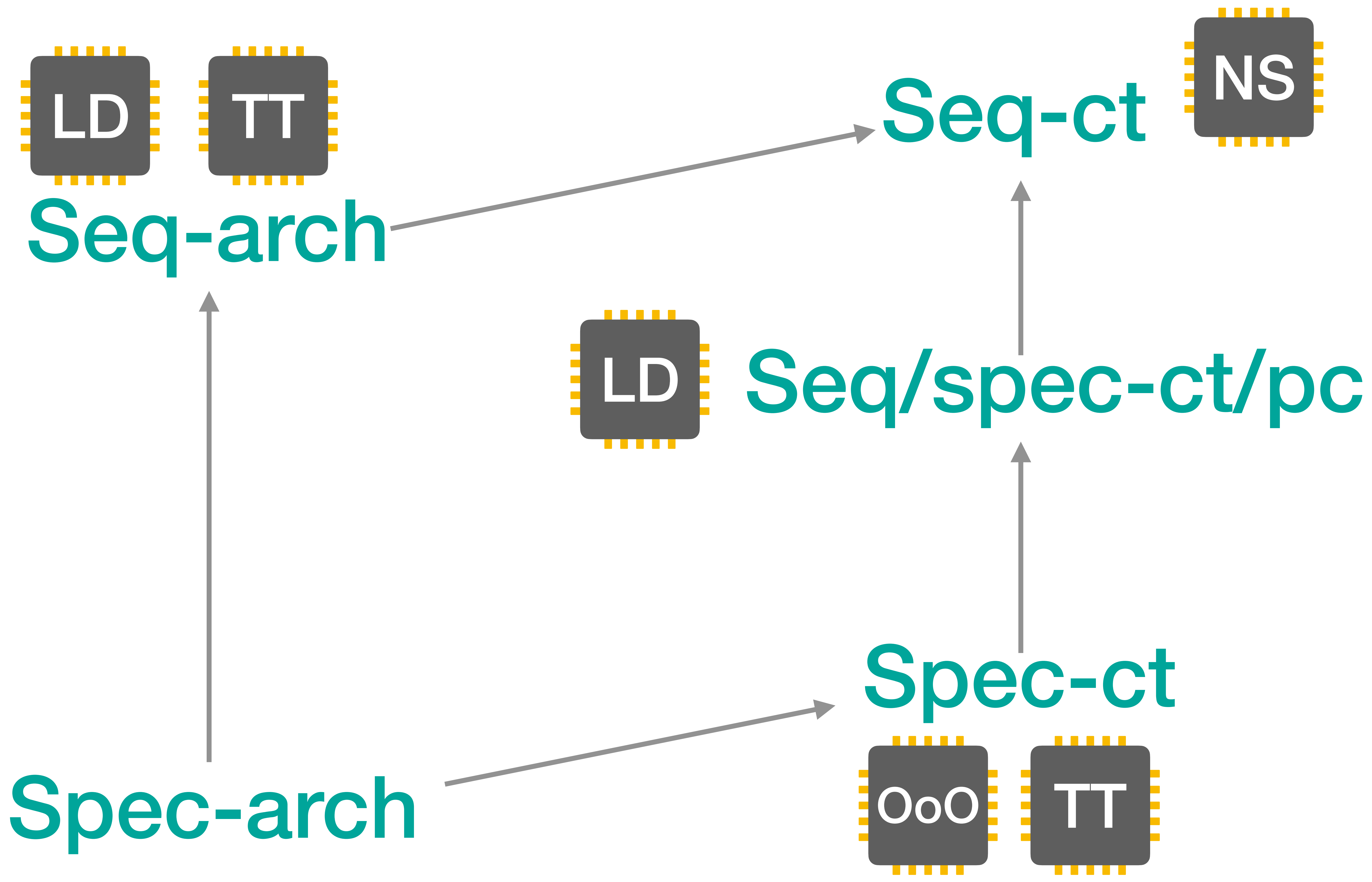
Delay tainted operations

STT and NDA, MICRO'19

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Characterize and compare security guarantees! 🥳



Propagate taint through computation

Delay tainted operations

STT and NDA, MICRO'19

OoO	Vanilla out-of-order (OoO) CPU
NS	In-order CPU (no speculative execution)
LD	OoO CPU+load delay
TT	OoO CPU+taint tracking

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

Outline

1. Speculative execution attacks
2. Modeling speculative leaks
3. Hardware-software contracts for secure speculation
4. What about hardware?
5. What about software?
6. Conclusions

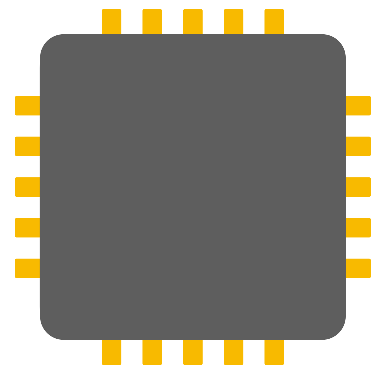
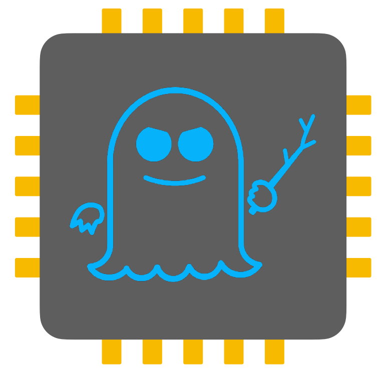
Speculative leaks in programs

Program + CPU with ***speculative execution*** = Secure?

Guarnieri, Köpf, Morales, Reineke, Sánchez — Spectector: Principled detection for speculative leaks — IEEE S&P 2020 — https://arxiv.org/abs/1812.08639

Speculative non-interference

Program **P** is **speculatively non-interferent** if

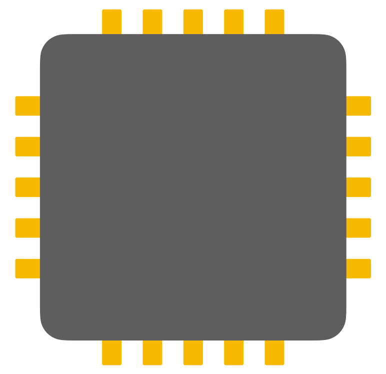
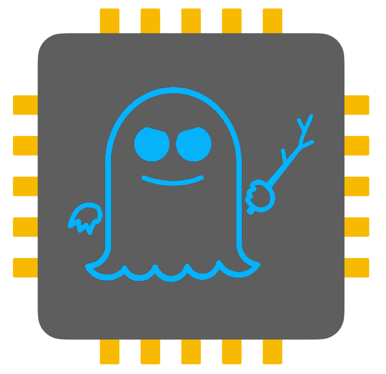
Leakage(**P**, ) = Leakage(**P**, )

Information leaked by executing **P** ***without*** speculative execution

Information leaked by executing **P** ***with*** speculative execution

Speculative non-interference

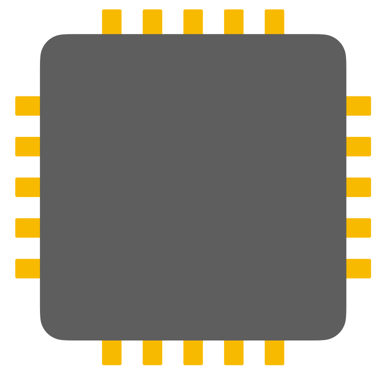
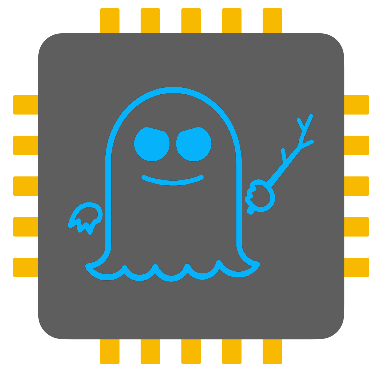
Program **P** is **speculatively non-interferent** if

Leakage(**P**, ) = Leakage(**P**, )

Executed under **seq-ct**

Speculative non-interference

Program `P` is **speculatively non-interferent** if

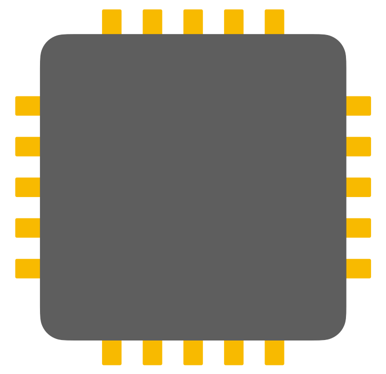
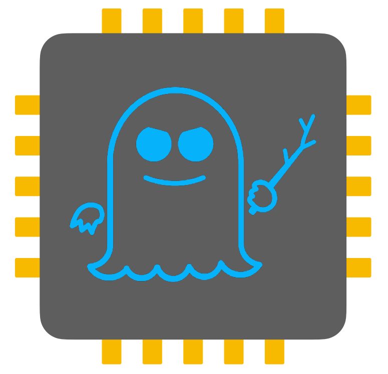
Leakage(**`P`**, ) = Leakage(**`P`**, )

Executed under **seq-ct**

Executed under **spec-ct**

Speculative non-interference

Program **P** is **speculatively non-interferent** if

Leakage(**P**, ) = Leakage(**P**, )

For all program states σ and σ':

Speculative non-interference

Program **P** is **speculatively non-interferent** if

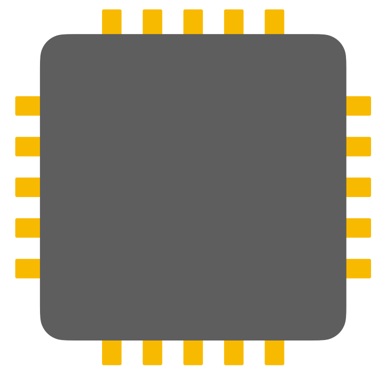
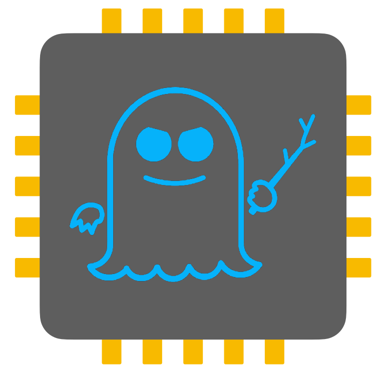
Leakage(**P**,) = Leakage(**P**,)

For all program states σ and σ':

seq-ct(**P**,σ) = **seq-ct**(**P**,σ')

Speculative non-interference

Program **P** is **speculatively non-interferent** if

Leakage(**P**, ) = Leakage(**P**, )

For all program states σ and σ':

seq-ct(**P**,σ) = **seq-ct**(**P**,σ')

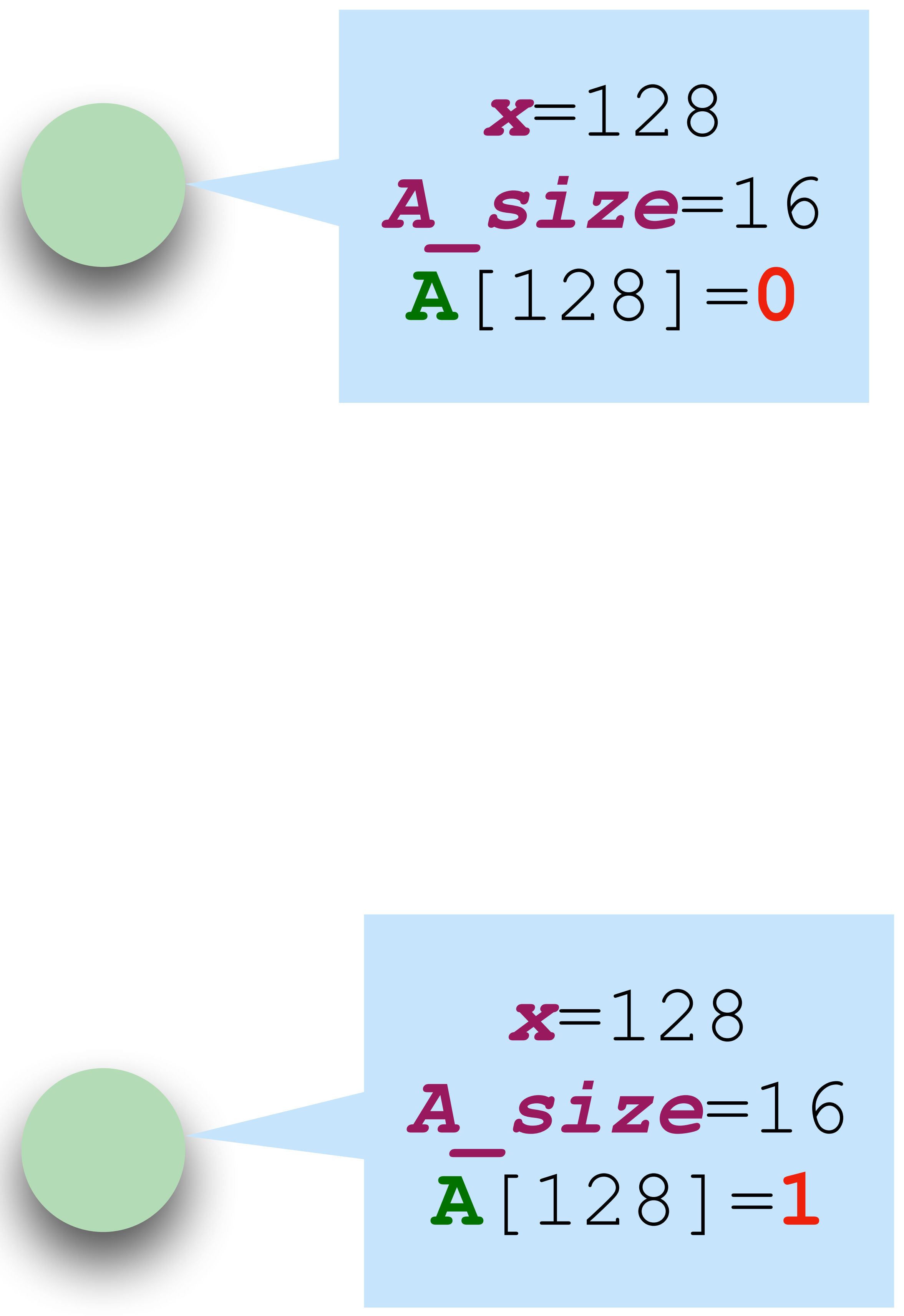
$\Longrightarrow$ **spec-ct**(**P**,σ) = **spec-ct**(**P**,σ')

Speculative non-interference

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

Speculative non-interference

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

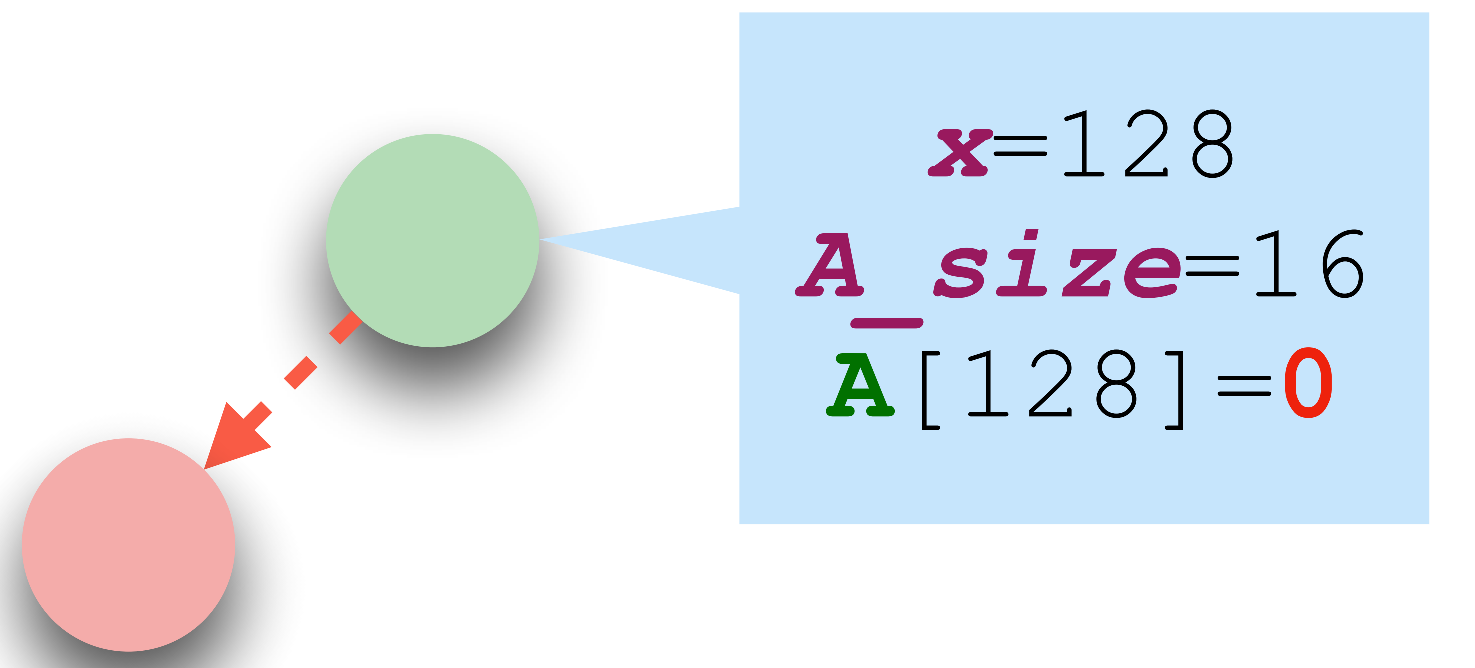


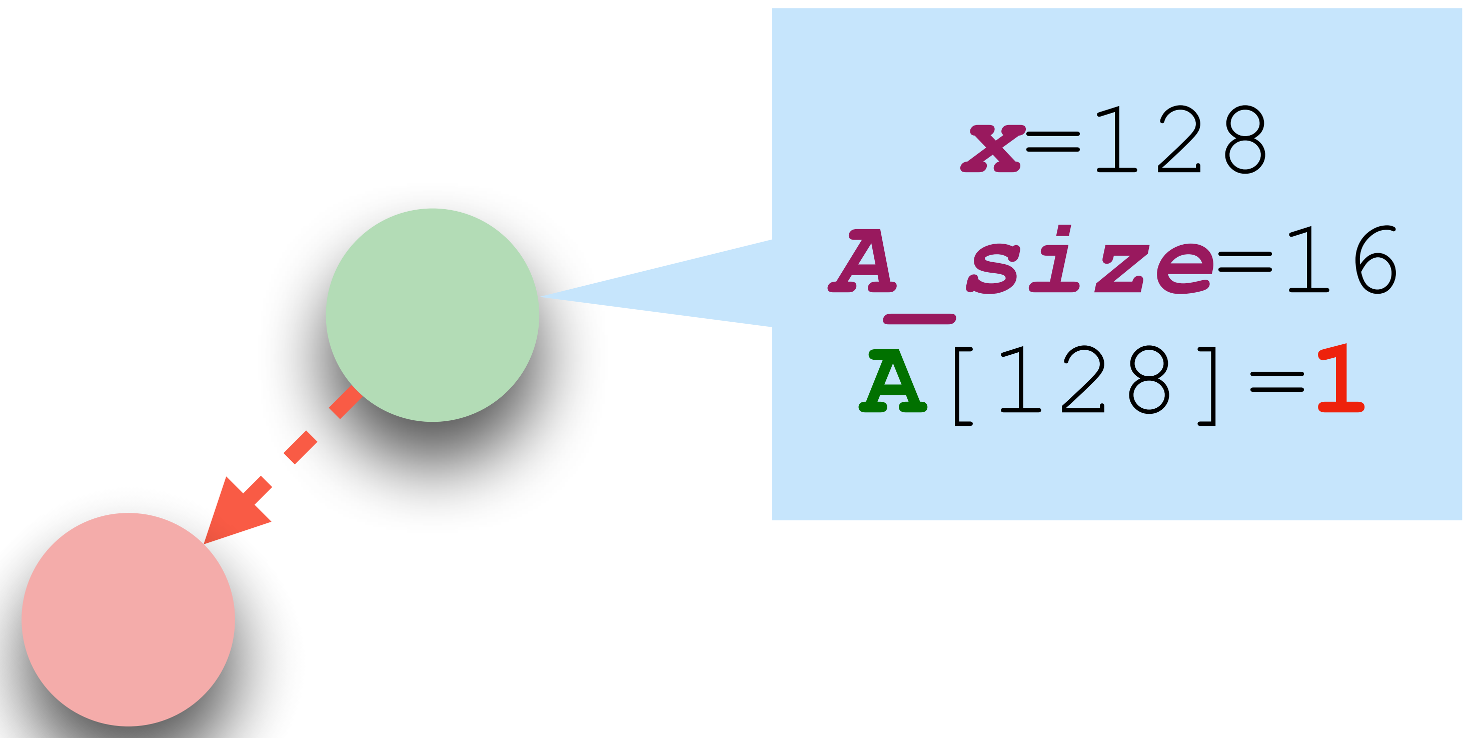
🟩 Non-speculative

🟥 Speculative

Speculative non-interference

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

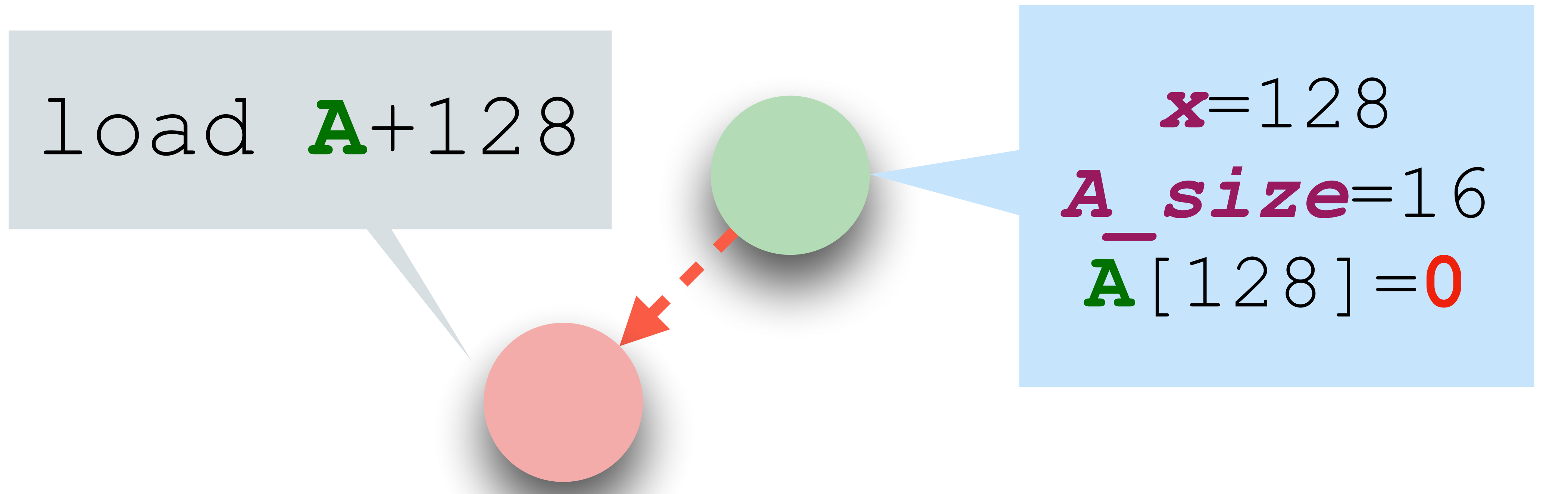




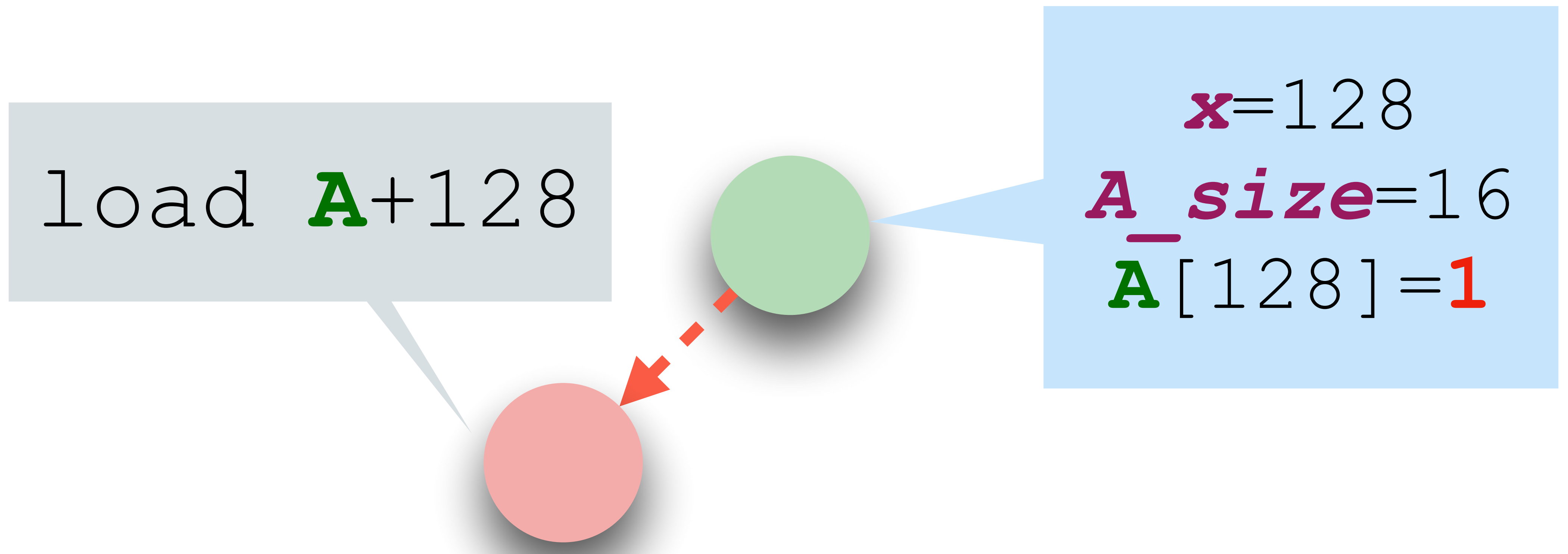
🟩 Non-speculative

🟥 Speculative

Speculative non-interference



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

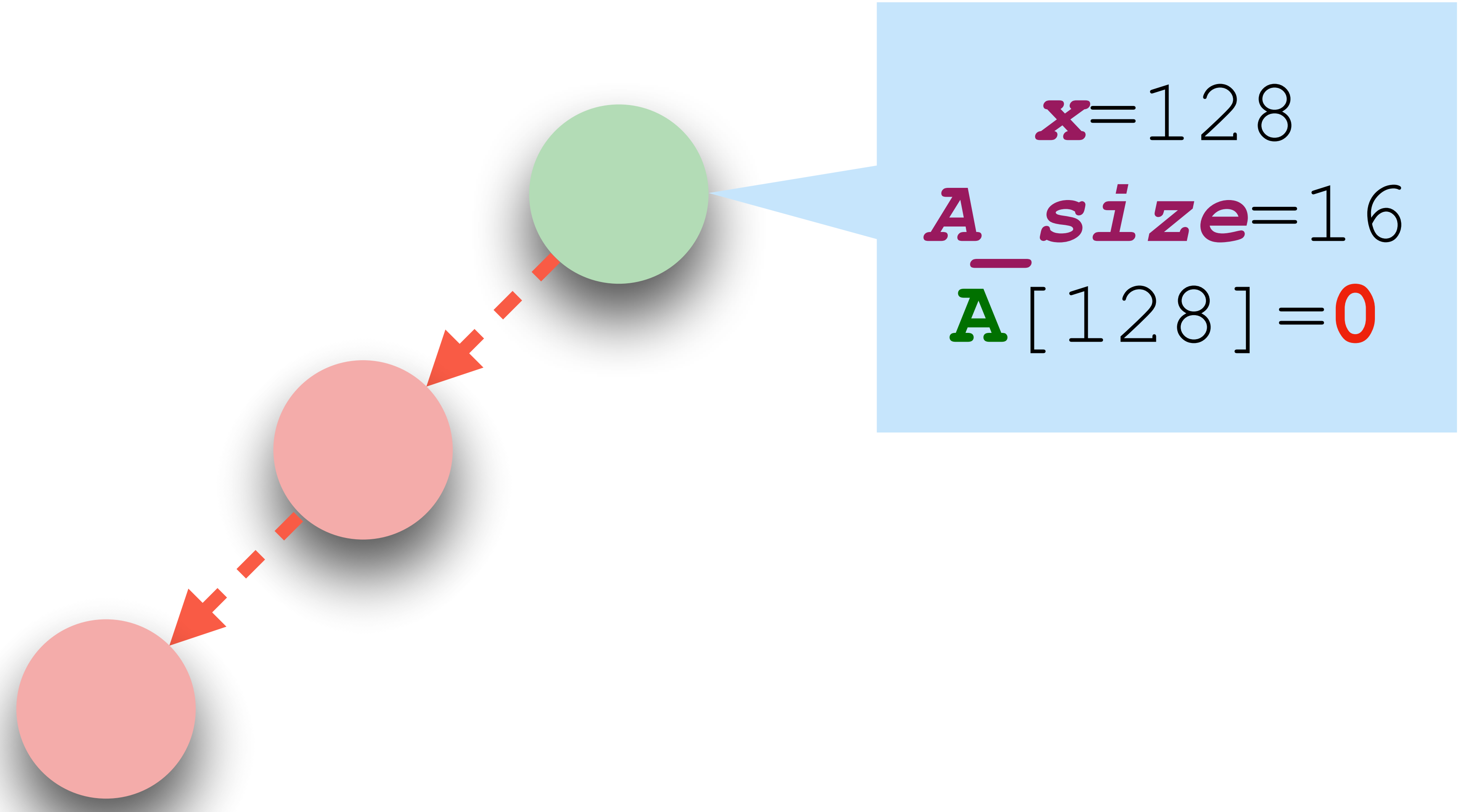


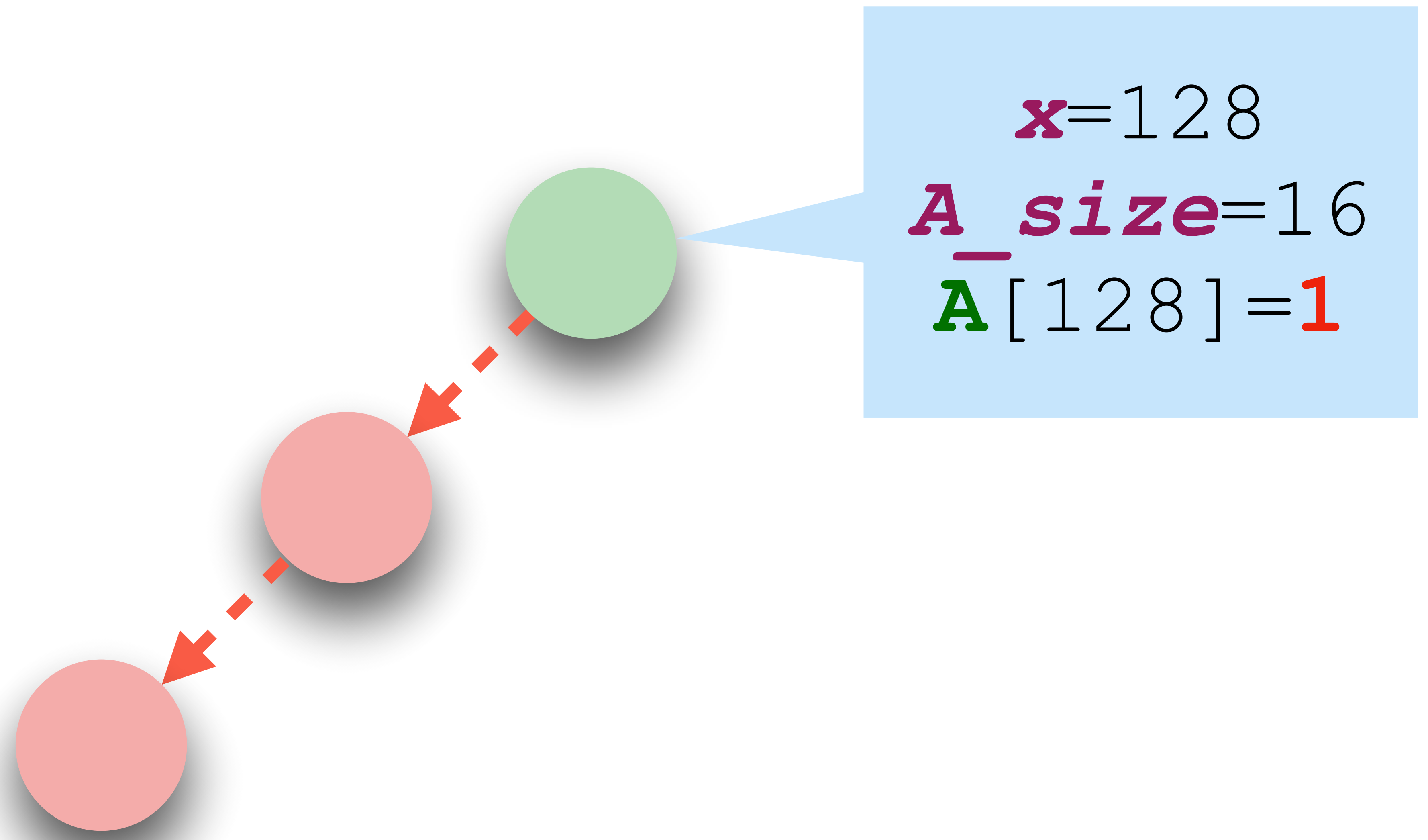
🟩 Non-speculative

🟥 Speculative

Speculative non-interference

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

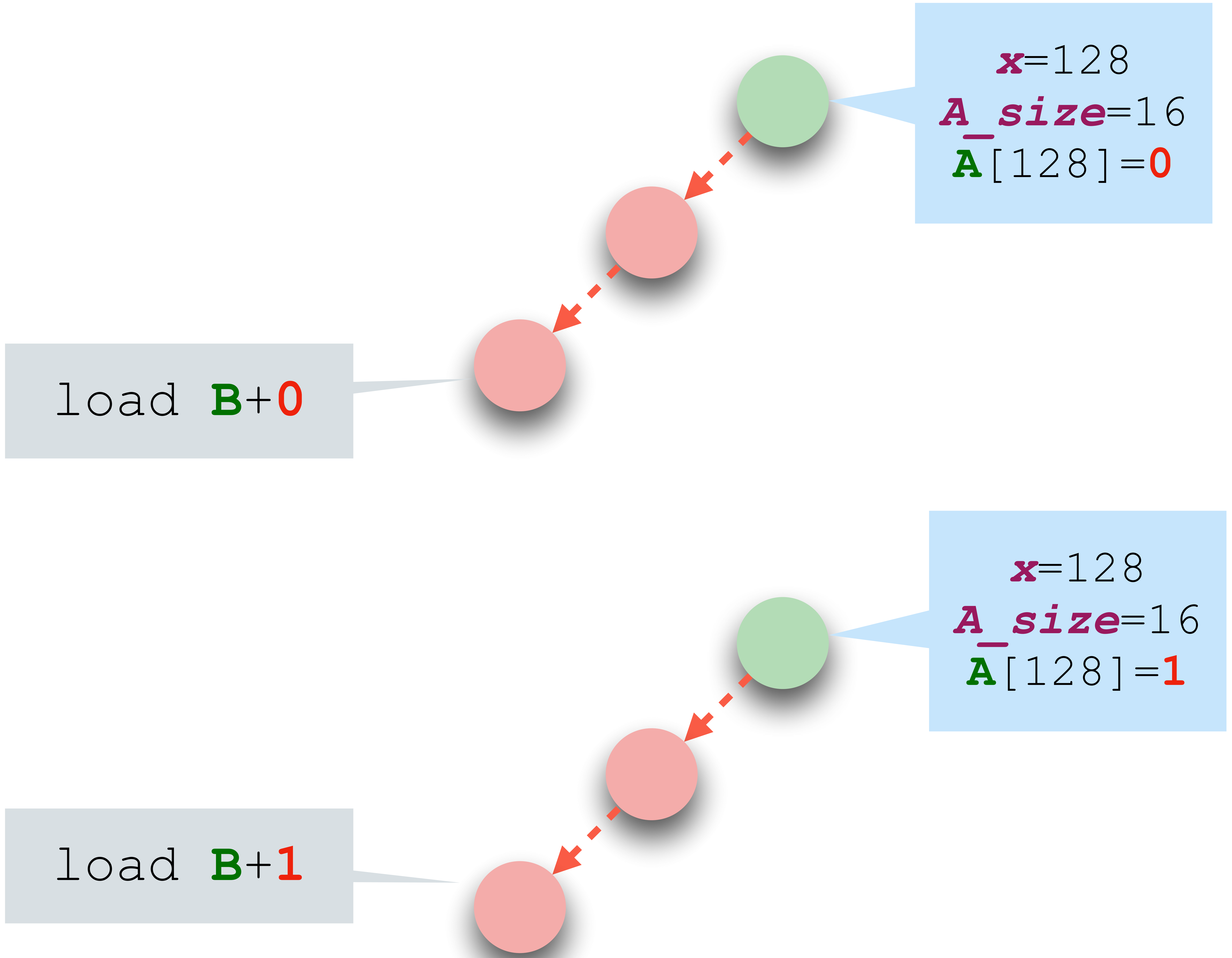




🟩 Non-speculative
🟥 Speculative

Speculative non-interference

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

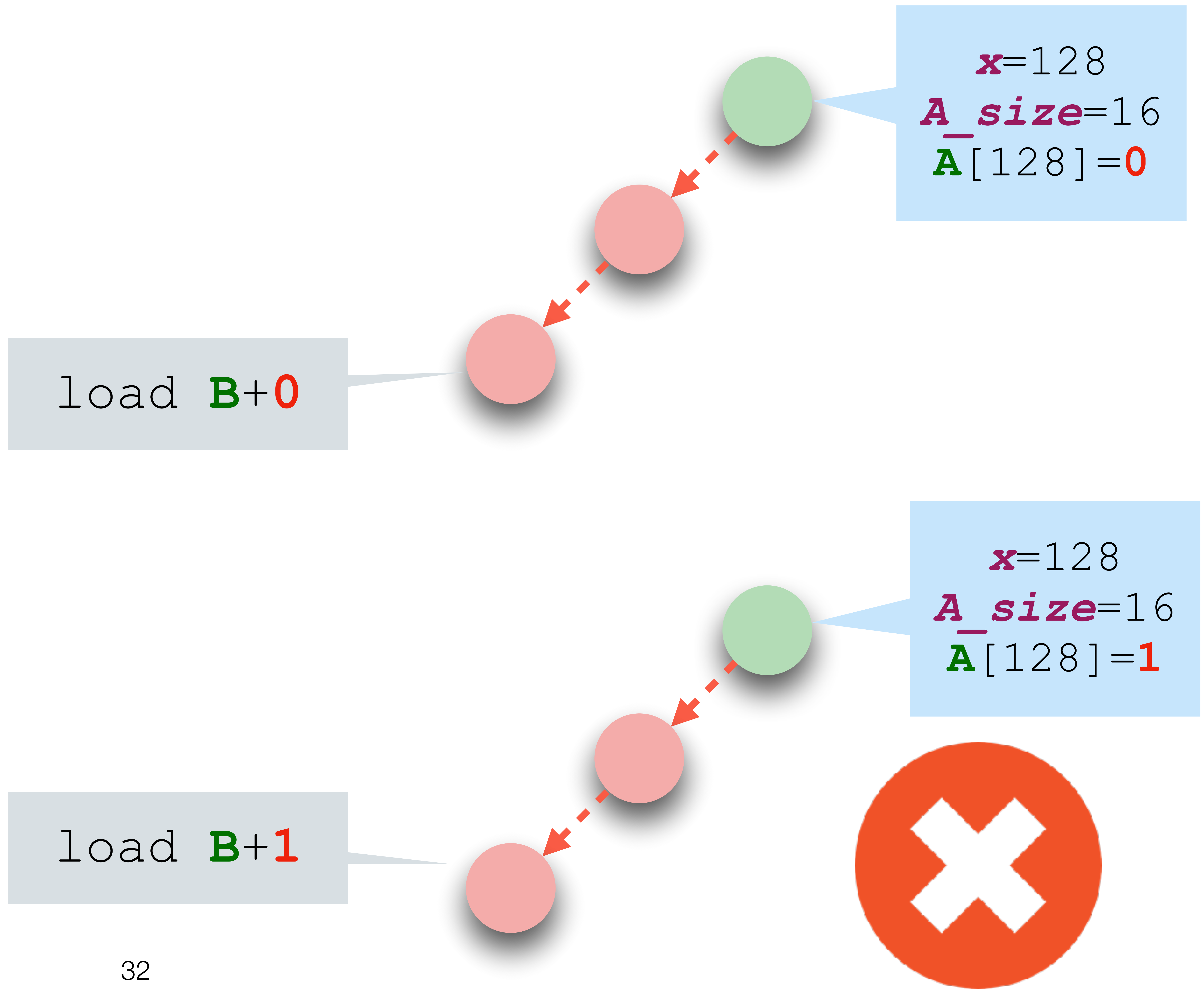


🟩 Non-speculative

🟥 Speculative

Speculative non-interference

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

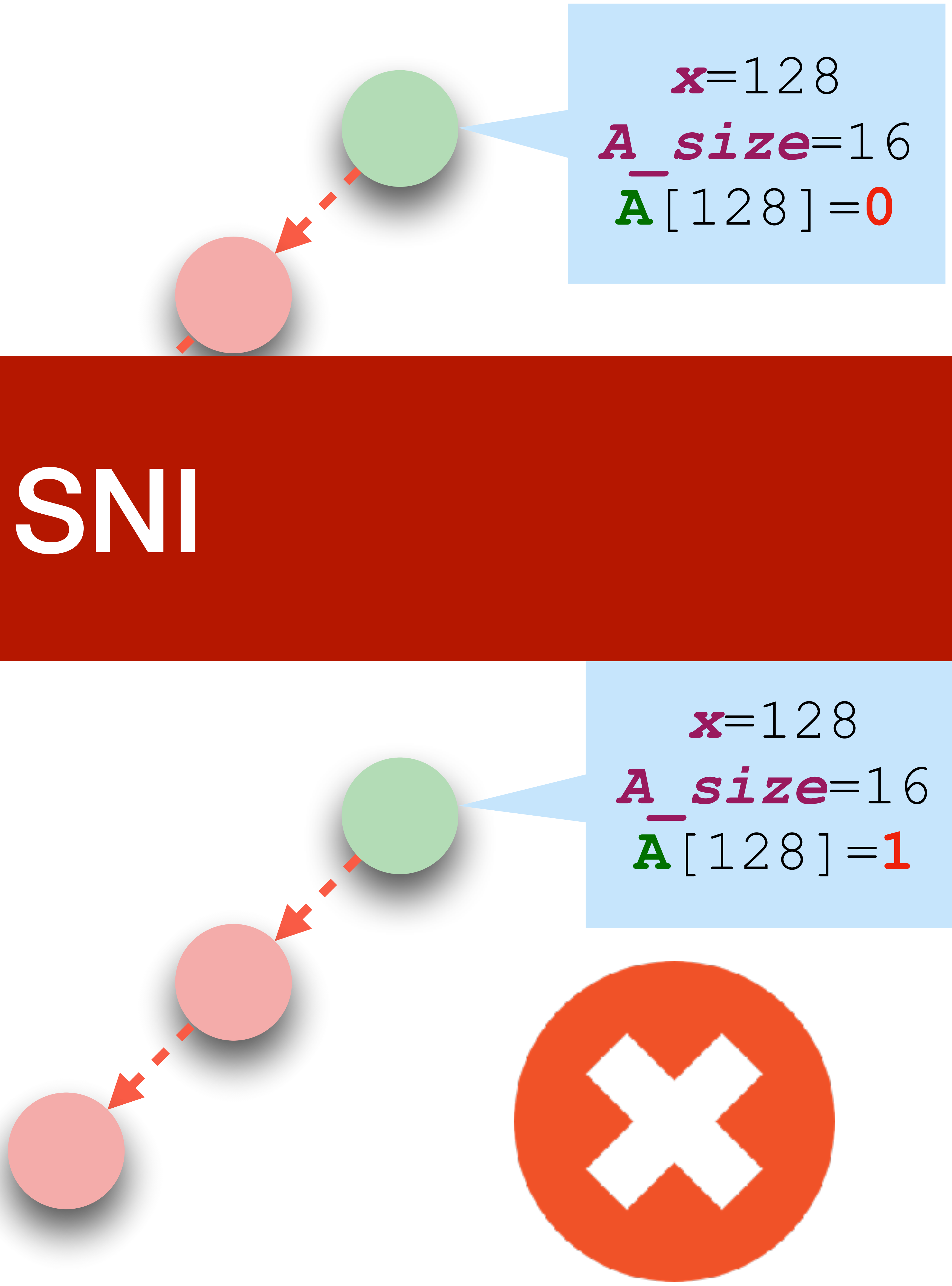


🟩 Non-speculative

🟥 Speculative

Speculative non-interference

```
       z = B[y]
  end
```



Spectre v1 violates SNI

🟩 Non-speculative

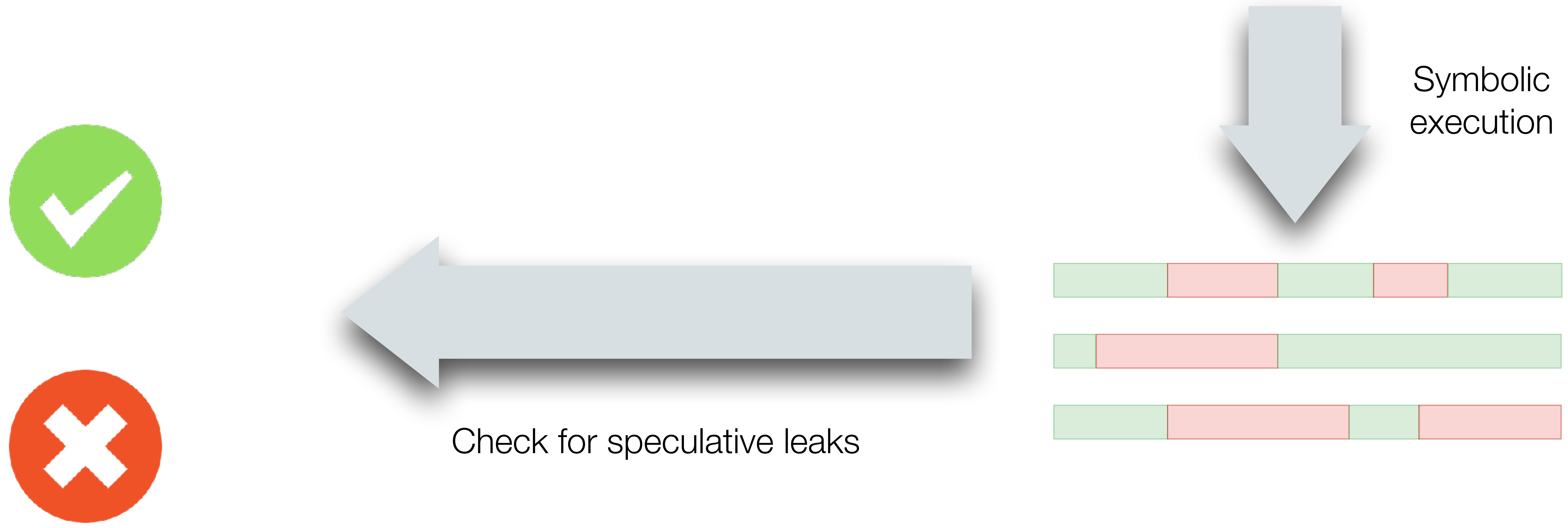
🟥 Speculative

Detecting speculative leaks

```
        mov     rax, A_size
        mov     rcx, x
        cmp     rcx, rax
        jae     END
    L1: mov     rax, A[rcx]
        mov     rax, B[rax]
```

x64 to µASM

```
              rax <- A_size
              rcx <- x
              jmp rcx≥rax, END
        L1:   load rax, A + rcx
              load rax, B + rax
        END:
```



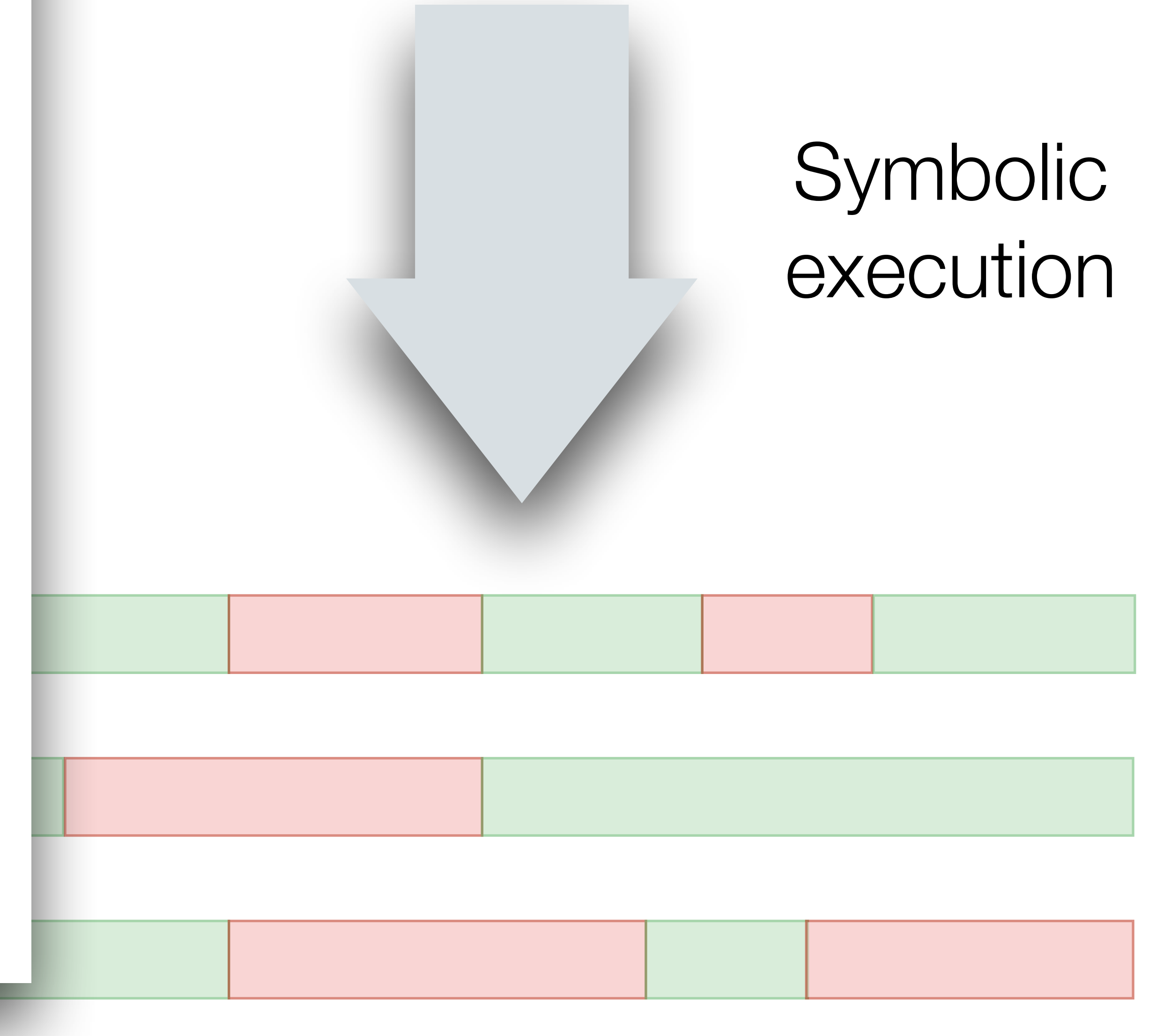
Detecting speculative leaks

```
    mov    rax, A_s
    mov    rcx, x
    cmp    rcx, rax
    jae    END
L1: mov    rax, A[r
    mov    rax, B[r
```

```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
load rax, A + rcx
load rax, B + rax
```

Spectector

https://spectector.github.io



Check for speculative leaks

Case study: compiler mitigations

Patrignani, Guarnieri — Exorcising spectres with secure compilers — CCS 2021
https://arxiv.org/abs/1910.08607

Case study: compiler mitigations

Injection of LFENCEs

LFENCE ***stops*** speculation

Compilers (***ICC***, ***MSVC***) insert LFENCE after ***branch instructions***

Patrignani, Guarnieri — Exorcising spectres with secure compilers — CCS 2021
https://arxiv.org/abs/1910.08607

Case study: compiler mitigations

Injection of LFENCEs

LFENCE ***stops*** speculation

Compilers (***ICC***, ***MSVC***) insert LFENCE after ***branch instructions***

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

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

Patrignani, Guarnieri — Exorcising spectres with secure compilers — CCS 2021
https://arxiv.org/abs/1910.08607

Case study: compiler mitigations

Injection of LFENCEs

LFENCE ***stops*** speculation

Compilers (***ICC***, ***MSVC***) insert LFENCE after ***branch instructions***

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

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

Patrignani, Guarnieri — Exorcising spectres with secure compilers — CCS 2021
https://arxiv.org/abs/1910.08607

Case study: compiler mitigations

Injection of LFENCEs

LFENCE ***stops*** speculation

Compilers (***ICC***, ***MSVC***) insert LFENCE after ***branch instructions***

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

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

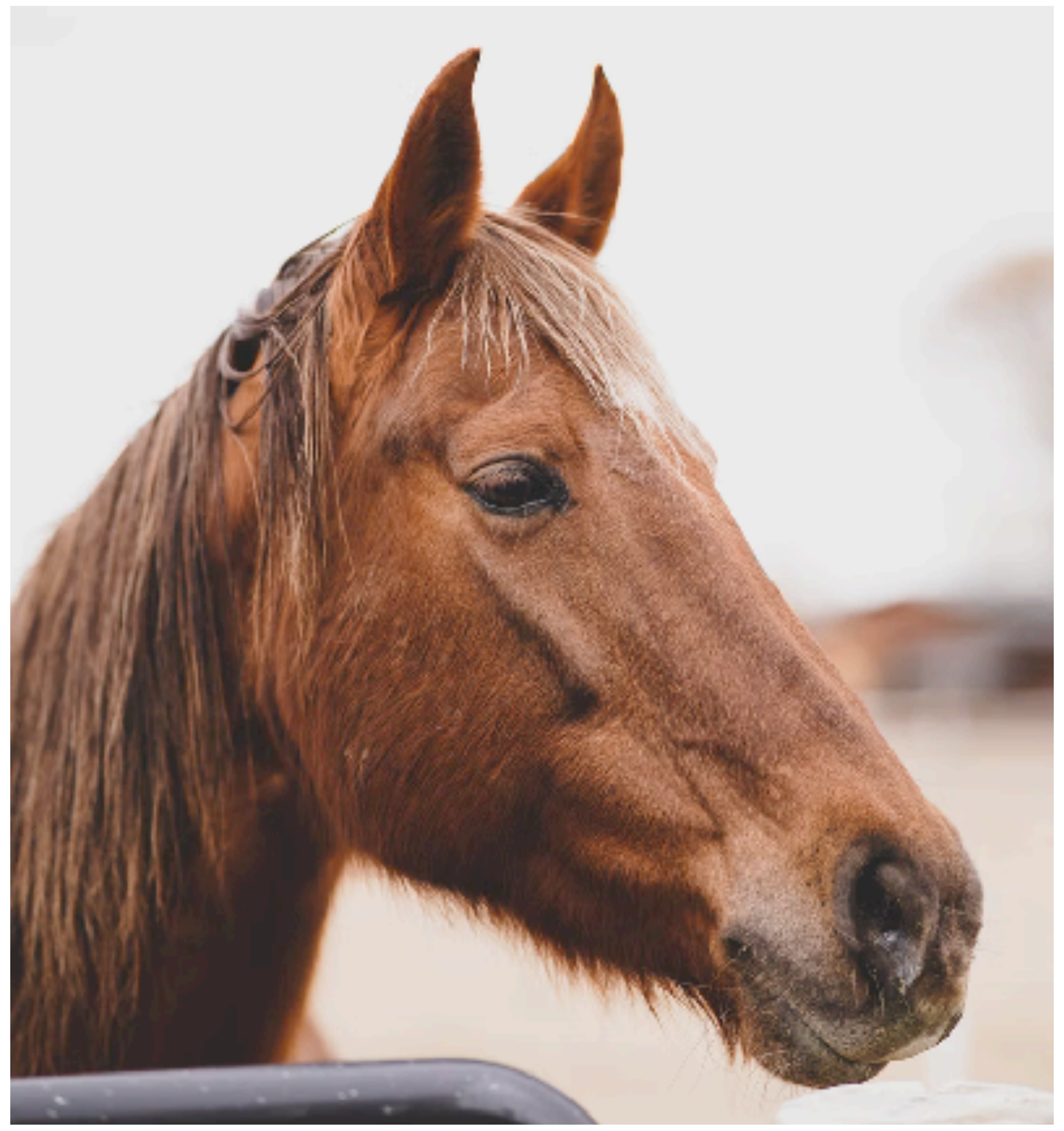
ICC enforces ***SNI*** (security proof) + unnecessary LFENCEs

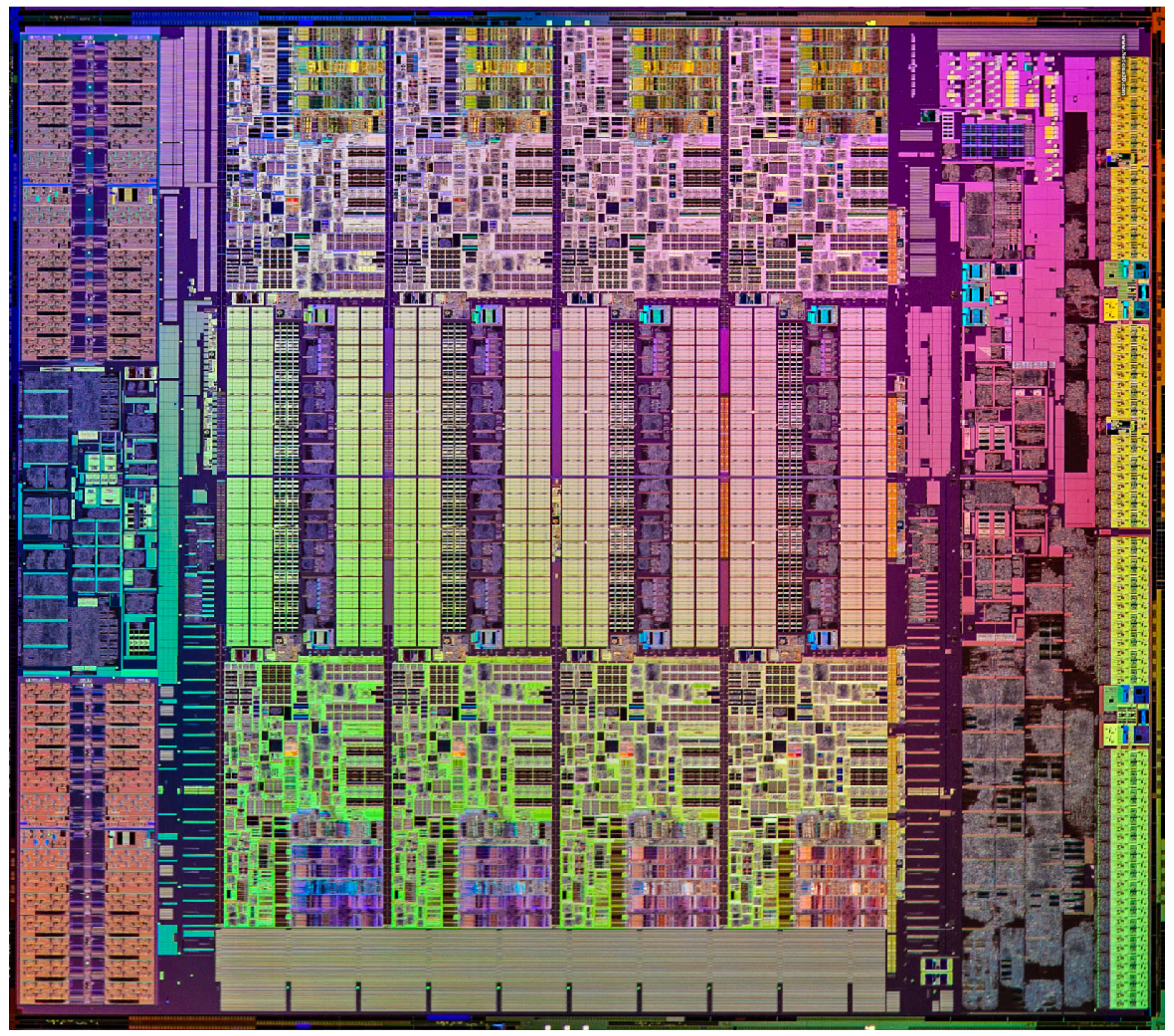
MSVC is **insecure** — leaks checked with Spectector

Patrignani, Guarnieri — Exorcising spectres with secure compilers — CCS 2021
https://arxiv.org/abs/1910.08607

Outline

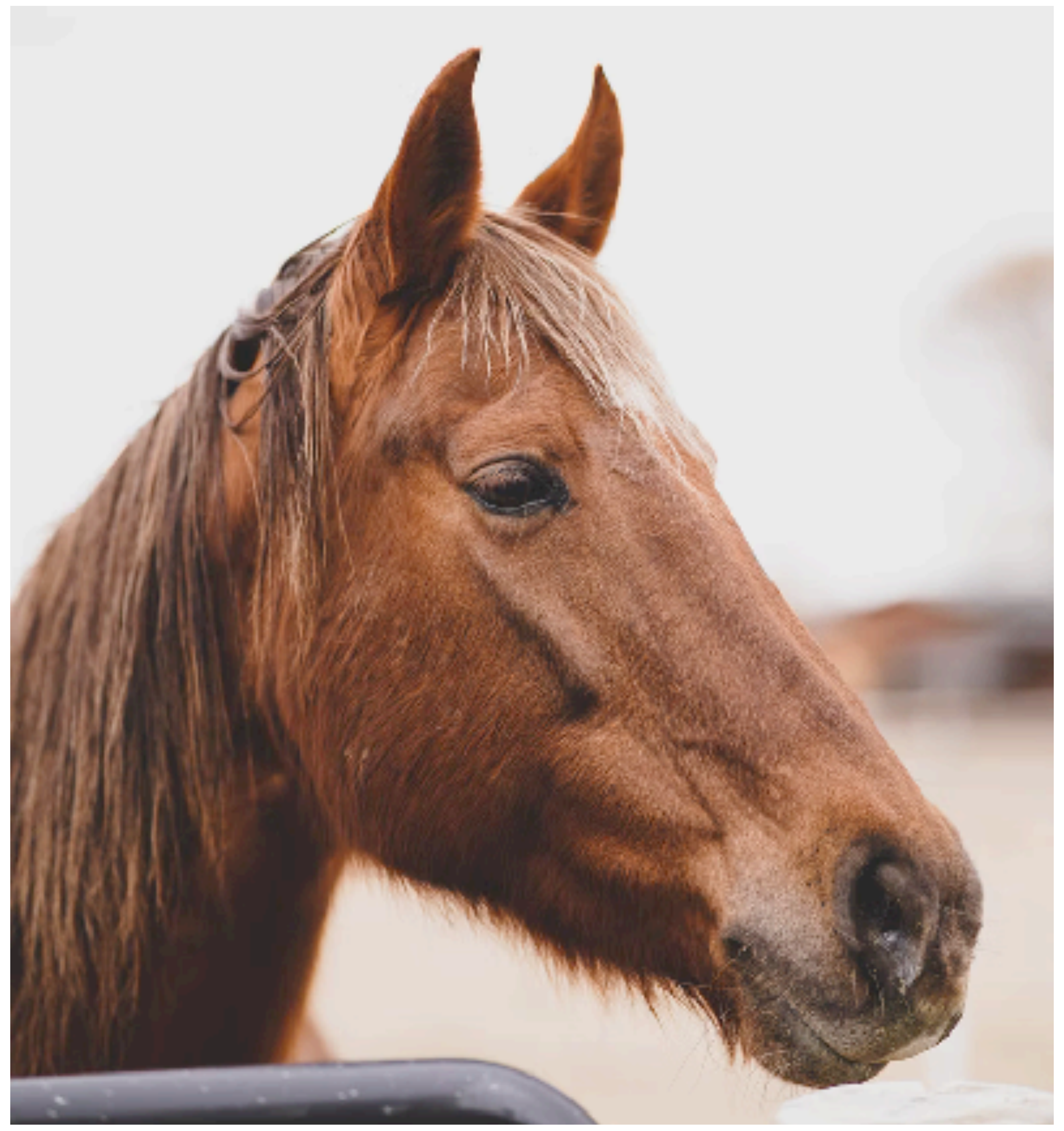
A problem of (missing) abstractions

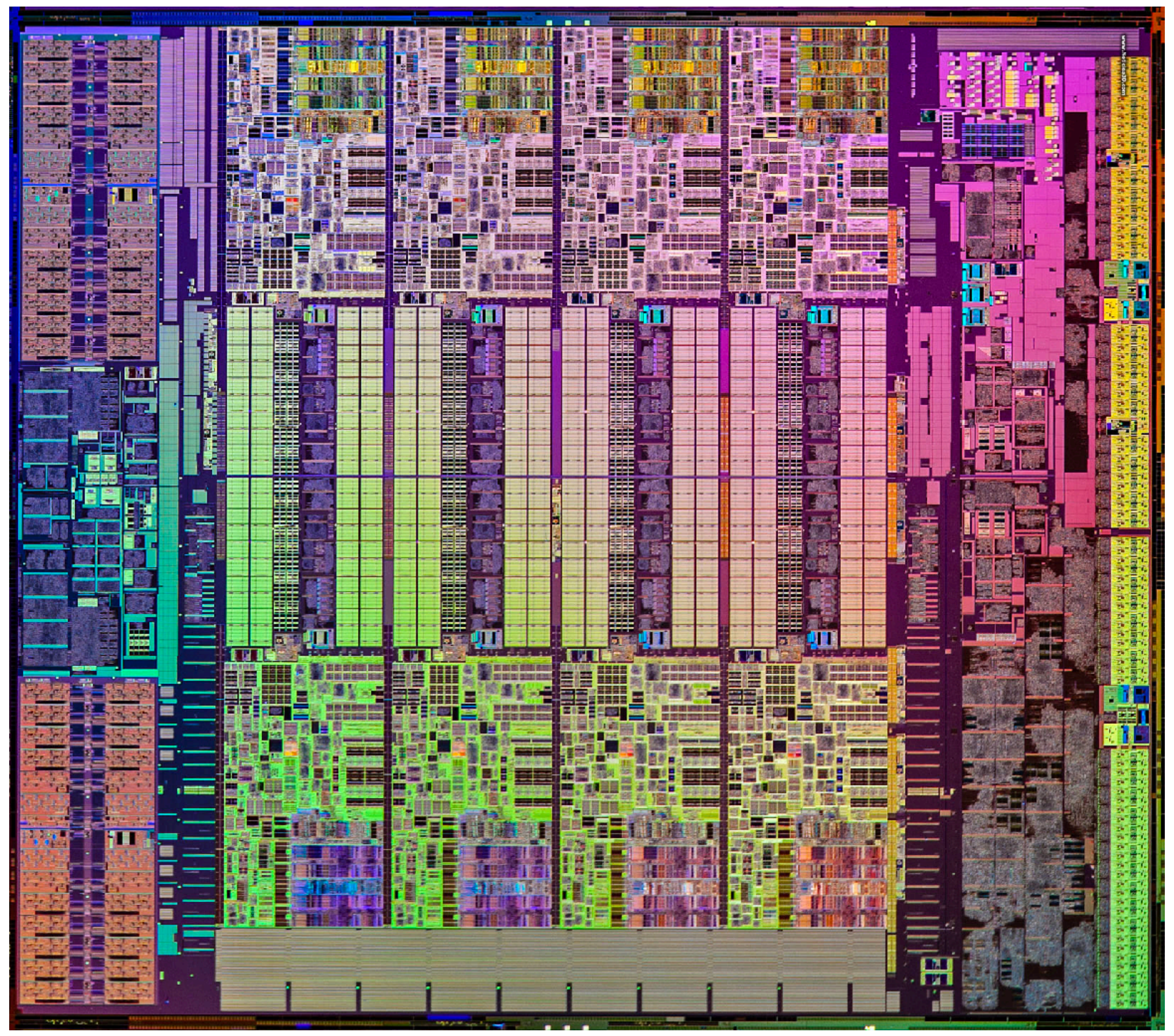


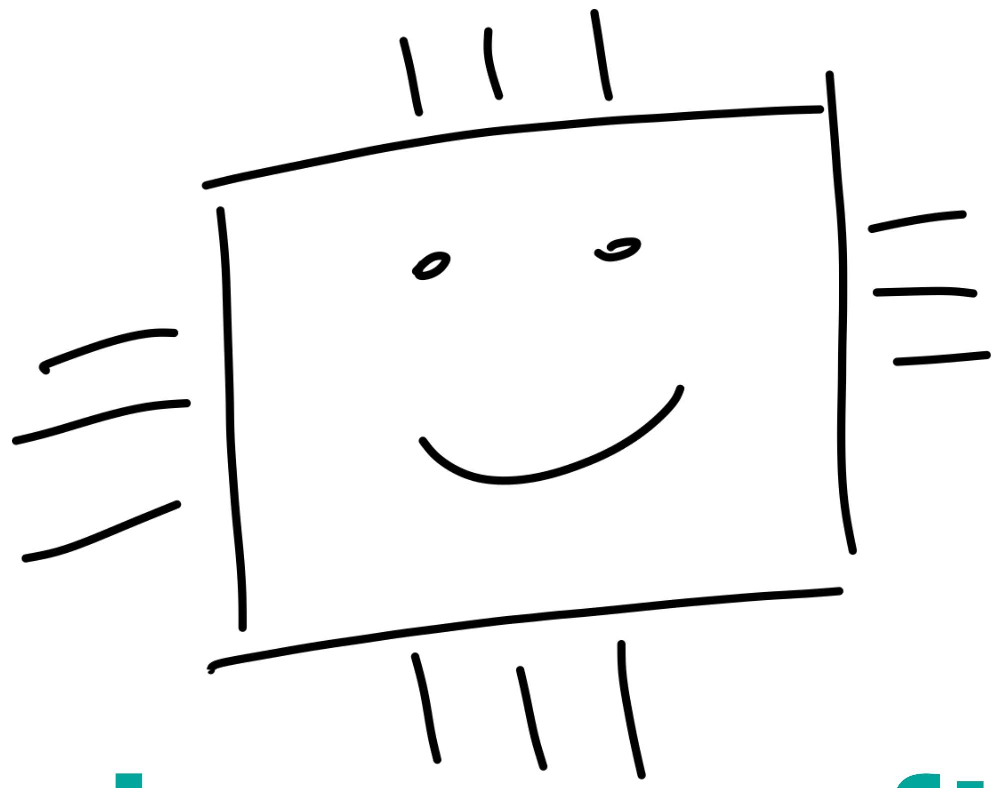


???

A problem of (missing) abstractions







Hardware-software contract

Challenges

We need ***precise*** and ***simple*** **hardware-software contracts** for ***security***

Challenges

We need ***precise*** and ***simple*** **hardware-software contracts** for ***security***

Challenge 1: (Languages and abstractions for) contracts that scale to real-world ISAs + other microarchitectural side-effects

Challenges

We need ***precise*** and ***simple*** **hardware-software contracts** for ***security***

Challenge 1: (Languages and abstractions for) contracts that scale to real-world ISAs + other microarchitectural side-effects

Challenge 2: Techniques for testing/verifying if hardware complies with a given contract (or even inferring compliant contract!)

Challenges

We need ***precise*** and ***simple*** **hardware-software contracts** for ***security***

Challenge 1: (Languages and abstractions for) contracts that scale to real-world ISAs + other microarchitectural side-effects

Challenge 2: Techniques for testing/verifying if hardware complies with a given contract (or even inferring compliant contract!)

Challenge 3: Contract-aware analysis and secure compilation techniques to enforce program security

Collaborators

- Boris Köpf @ Microsoft Research
- Jan Reineke @ Saarland University
- José F. Morales @ IMDEA Software
- Pepe Vila @ IMDEA Software
- Andrés Sánchez @ IMDEA Software
- Marco Patrignani @ University of Trento

Supported by Intel Strategic Research Alliance (ISRA) "Information Flow Tracking across the Hardware-Software Boundary"

We need ***precise*** and ***simple*** **hardware-software contracts** for ***security***

Challenge 1: (Languages and abstractions for) contracts that scale to real-world ISAs + other microarchitectural side-effects

Challenge 2: Techniques for testing/verifying if hardware complies with a given contract (or even inferring compliant contract!)

Challenge 3: Contract-aware analysis and secure compilation techniques to enforce program security